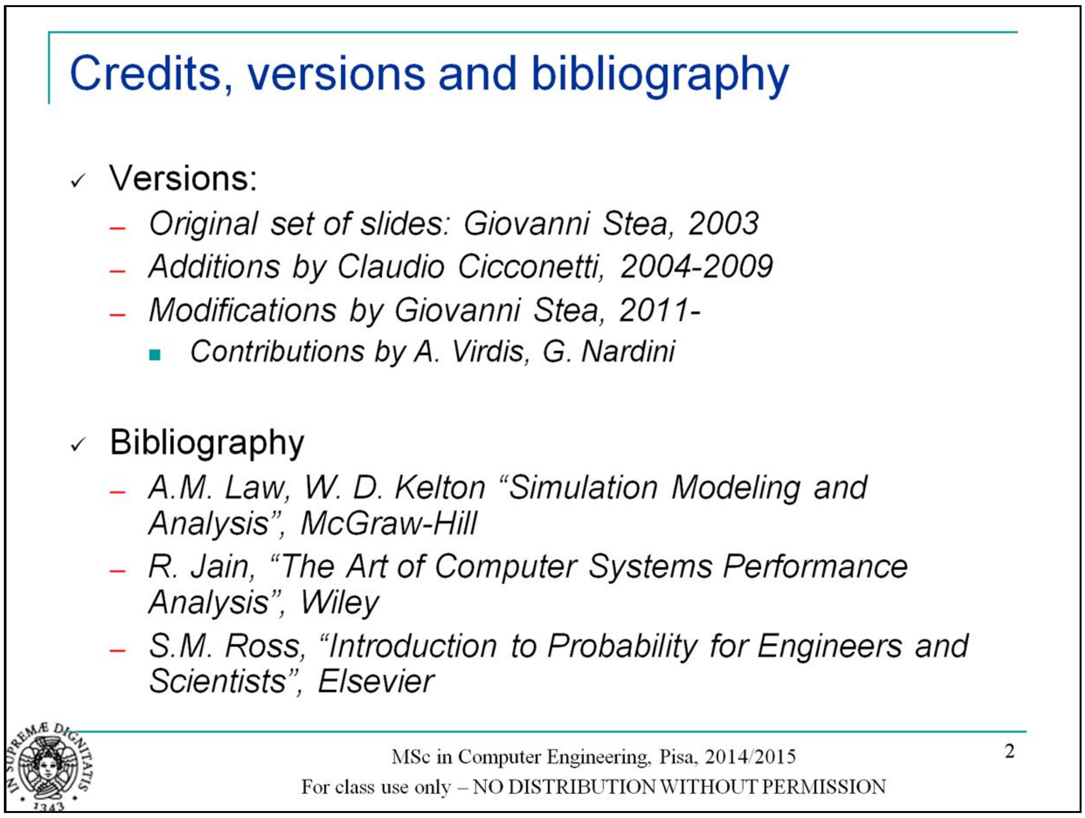
Simulation

Giovanni Stea
Dipartimento di Ingegneria dell'Informazione
University of Pisa, Italy



Credits, versions and bibliography

✓ Versions:

- *Original set of slides: Giovanni Stea, 2003*
- *Additions by Claudio Cicconetti, 2004-2009*
- *Modifications by Giovanni Stea, 2011-*
  - *Contributions by A. Virdis, G. Nardini*

✓ Bibliography

- *A.M. Law, W. D. Kelton "Simulation Modeling and Analysis", McGraw-Hill*
- *R. Jain, "The Art of Computer Systems Performance Analysis", Wiley*
- *S.M. Ross, "Introduction to Probability for Engineers and Scientists", Elsevier*

MSc in Computer Engineering, Pisa, 2014/2015

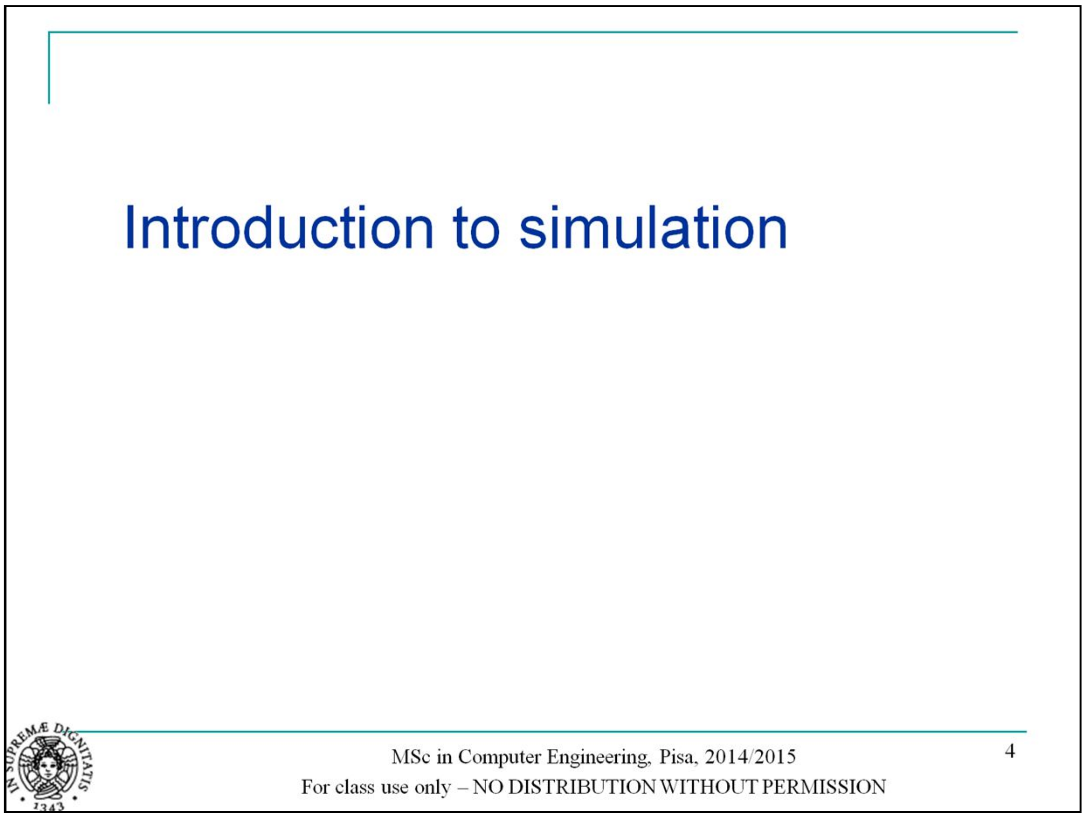
For class use only – NO DISTRIBUTION WITHOUT PERMISSION

2

The part on analysis of output data can be found on both Law-Kelton's and Ross's books. What is here is taken from Ross's book.

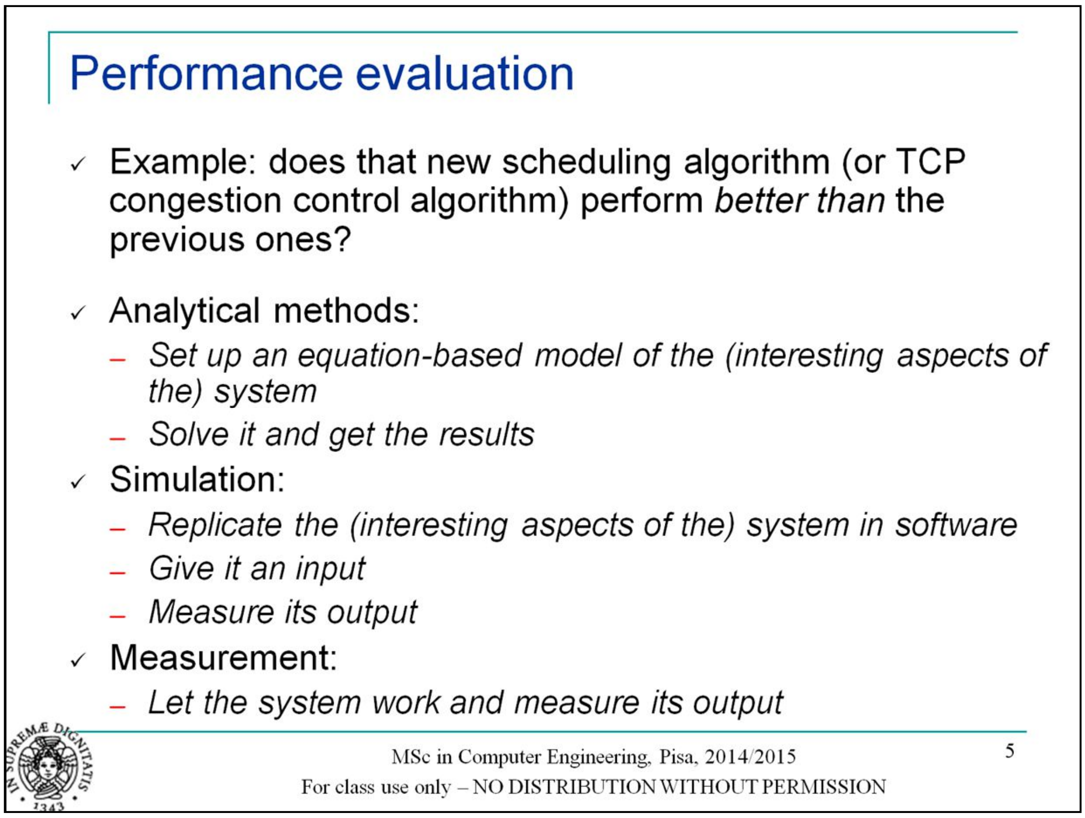
Outline

- ✓ General introduction to simulation (~1h)
  - *Why simulation*
- ✓ Part **one**: how to build a simulator (~4h)
  - *Discrete event simulators*
  - *Data structures and code*
  - *Random number generation*
- ✓ Part **two**: how to do a simulation analysis (~2h)
  - *Simulation workflow*
- ✓ Part **three**: how to gather meaningful data (~2h)
  - *Steady-state analysis*
  - *Statistically sound output results*
- ✓ Part **four**: what to look for
  - *QoS/QoE*
  - *What to measure*



Introduction to simulation

Most of the examples will be taken from networking. However, the concepts are quite general, and can be applied to simulating pretty much anything.



Performance evaluation

- ✓ Example: does that new scheduling algorithm (or TCP congestion control algorithm) perform *better than* the previous ones?
- ✓ Analytical methods:
  - *Set up an equation-based model of the (interesting aspects of the) system*
  - *Solve it and get the results*
- ✓ Simulation:
  - *Replicate the (interesting aspects of the) system in software*
  - *Give it an input*
  - *Measure its output*
- ✓ Measurement:
  - *Let the system work and measure its output*

What is the "performance evaluation" of a system?

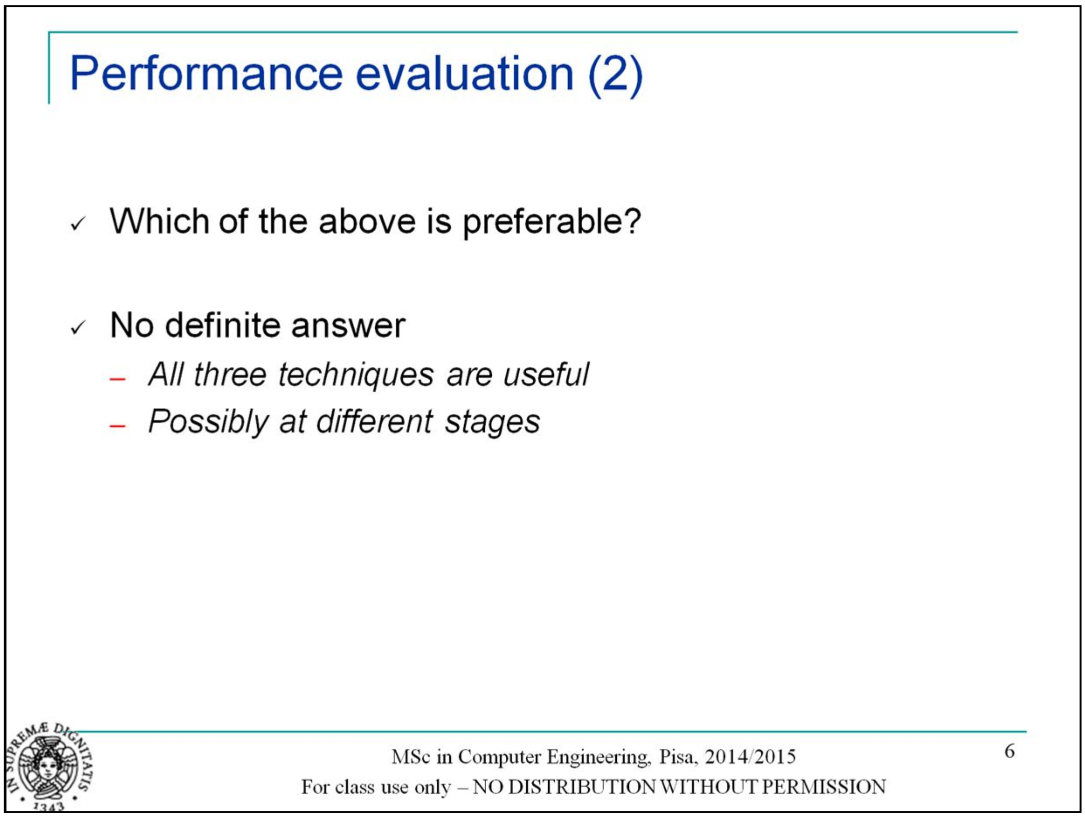
It consists in **monitoring** some interesting quantities, how they **evolve over time**, observing their **statistics** (e.g. the mean value), or computing some **limit values** (e.g., the maximum value, etc.).

This can be done essentially in three ways (or any combination thereof):

a) **analytically**: you setup an **analytical model of the system**, i.e. **equations** that bind the quantities of interest. You solve these equations and end up with the results.
   - Problem: for **complex** systems, writing down these equations can be extremely difficult. For less-than-trivial systems, **solving** these equations is often impossible.
   - You can **simplify** the models. You abstract away some of the details, and leave only the "important stuff". The problem is that you never know in advance what is important enough, and you often end up underestimating the importance of the aspects you neglect.

b) **Through simulation**: you setup a **software replica** of your system; you give your software an input, and you get some output in return.
   - If the system is complex, you can replicate only **a part of it** (or **some aspects of it**). If you want to evaluate a routing algorithm, you probably don't need to simulate all the details of a router (and, more specifically, you don't need to know any details on the switching fabric of the router, its memory access policy, etc.).

c) **Through measurement**: you setup a prototype (e.g., buy hardware and write down the actual code), and have it run either "in the wild" or in a controlled environment, and take measurements.



Performance evaluation (2)

- ✓ Which of the above is preferable?
- ✓ No definite answer
  - *All three techniques are useful*
  - *Possibly at different stages*

MSc in Computer Engineering, Pisa, 2014/2015
For class use only – NO DISTRIBUTION WITHOUT PERMISSION
6

It depends on what "preferable" means: Descriptive power / Cost.

Analytical methods yield **input-output relationships**, which are extremely useful. Simulation gives you **one output** for **a set of input parameters**, which is considerably less useful (you have to re-run the simulator many times to get the same information).

Unfortunately, unless systems are **very simple**, analytical techniques are of little help. The hypotheses that are required to make them tractable are normally heavy enough as to make **models** arbitrary distant from reality. Solving systems of dynamic equations is also **quite difficult**.

Measurement techniques are also not entirely devoid of problems. [yes, there **are** techniques about measuring quantities (it is no piece of cake)]

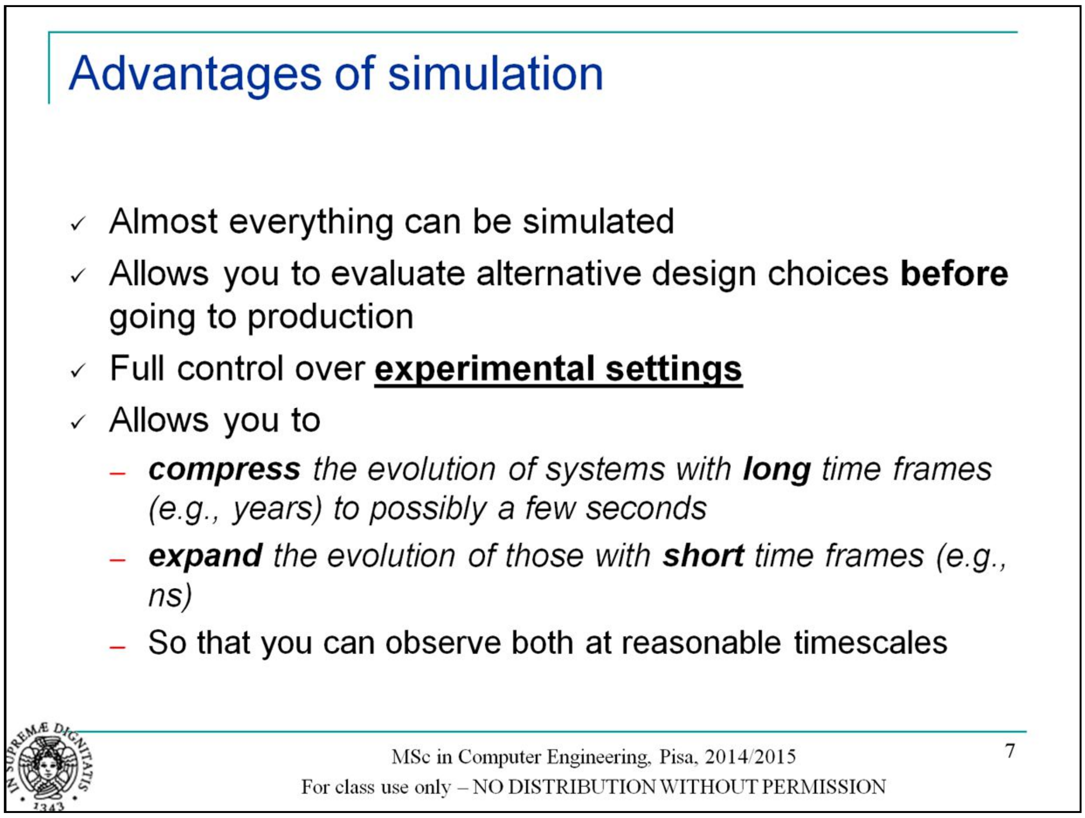
- The system **may not exist at all**: you often want to evaluate something that you want to deploy. Then you have to resort to **prototypes**, which is often costly (not so much as long as it is a software prototype. If there's hardware involved, then you easily get large figures as far as money is involved).
- The system may be **harmful** or **safety-critical** (weaponry, vehicles, medical implements, etc.) or impossible to test
  1. "**in the wild**" it is sometimes not a good idea to test a prototype "in the wild" (e.g., an assisted-driving system, a new congestion control algorithm for TCP – you may end up congesting the Internet while you fine-tune the parameters)
  2. In a "**controlled environment**": you have to be sure that the conditions of the controlled environment are similar to those "in the wild" (which is seldom true).

Simulation, unlike the other techniques, can be used almost everywhere:

- Even though the system is too complex, writing down some code which simulates it may be **reasonably simple**
- A simulated model is hardly harmful, unlike the real thing
- You can do things that are difficult or costly to do *in vivo*: **stress tests**, i.e. trying to find out what is the load that breaks a system (e.g., a bridge).

The 3 techniques can be used **concurrently** to aid different stages of the design and manufacturing process:

- Analytical techniques may be good as a "quick'n'dirty" way of figuring out a system dimensioning/cost in an early phase
- A good simulation analysis can help to gain more **insight** in the behavior of a system during the detailed design process
- A prototype may be the **last** thing before going live, e.g. testing in a controlled environment first, and in the wild later.



Advantages of simulation

- ✓ Almost everything can be simulated
- ✓ Allows you to evaluate alternative design choices **before** going to production
- ✓ Full control over **experimental settings**
- ✓ Allows you to
  - *__compress__ the evolution of systems with __long__ time frames (e.g., years) to possibly a few seconds*
  - *__expand__ the evolution of those with __short__ time frames (e.g., ns)*
  - So that you can observe both at reasonable timescales

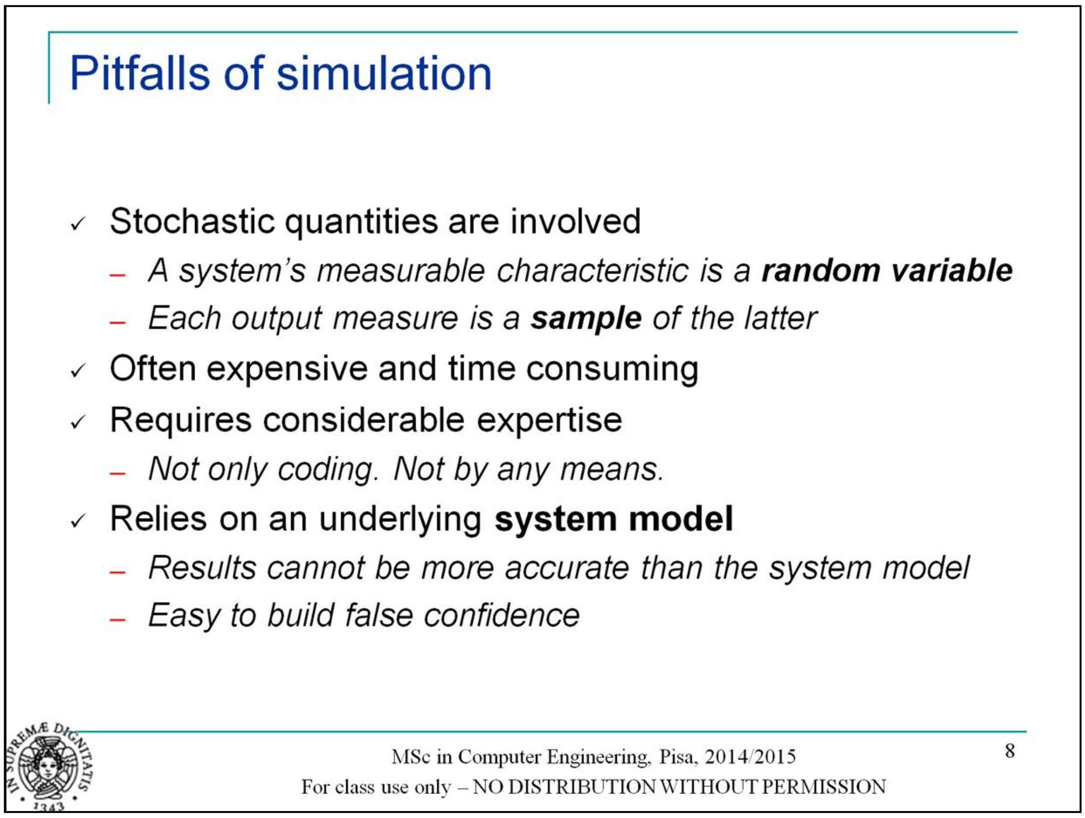
MSc in Computer Engineering, Pisa, 2014/2015 7
For class use only – NO DISTRIBUTION WITHOUT PERMISSION

This lecture is about simulation.

Simulation is one of the most used performance evaluation techniques (besides *optimization*). In the field of computer engineering (and, more specifically, of computer networks), it is by far the most widely used one.

It is also an active **research field**. There are journals and conferences dedicated to **simulation techniques**, and its scientific aspects.

Your "experimental settings": when you do simulation, you simulate not only your system, but also **everything your system interacts with.** You can control your environment in a simulation setting, not quite so in a live measurement. A typical case is **scalability assessment**: you want to measure what happens if you multiply your input (e.g., a number of users) by a factor *k*. You can do this easily in a simulation, not so in the real life.


Pitfalls of simulation

- ✓ Stochastic quantities are involved
  - *A system's measurable characteristic is a* ***random variable***
  - *Each output measure is a* ***sample*** *of the latter*
- ✓ Often expensive and time consuming
- ✓ Requires considerable expertise
  - *Not only coding. Not by any means.*
- ✓ Relies on an underlying **system model**
  - *Results cannot be more accurate than the system model*
  - *Easy to build false confidence*

MSc in Computer Engineering, Pisa, 2014/2015 8
For class use only – NO DISTRIBUTION WITHOUT PERMISSION

A simulator takes as input **data sampled from probability distributions**.

For instance, if I am simulating a network node that transmits packets, the **length** of these packets and their **interarrival** times will often be generated **randomly**. This means that the **node throughput** is itself a **random variable**. The **queuing delay** of these packets is a RV as well.

Each measure that is obtained from the simulator is a **sample** of an **output random variable**, and should be treated as such.

Writing down a simulator is no piece of cake. Writing down the simulator of the LTE technology took 2 MY of highly specialized PhD students, and resulted in 40k+ lines of code. This was obtained **relying on a pre-existing** (general-purpose) **simulation framework**, i.e. customizing something that was already available.

Furthermore, **simulation analysis** (not *writing down the code, but* ***using that code)*** requires **a lot of time**, and (mostly) **a lot of experience**. You need to know:

a) The system to be analyzed
b) Probability theory
c) Statistics
d) Computer Science

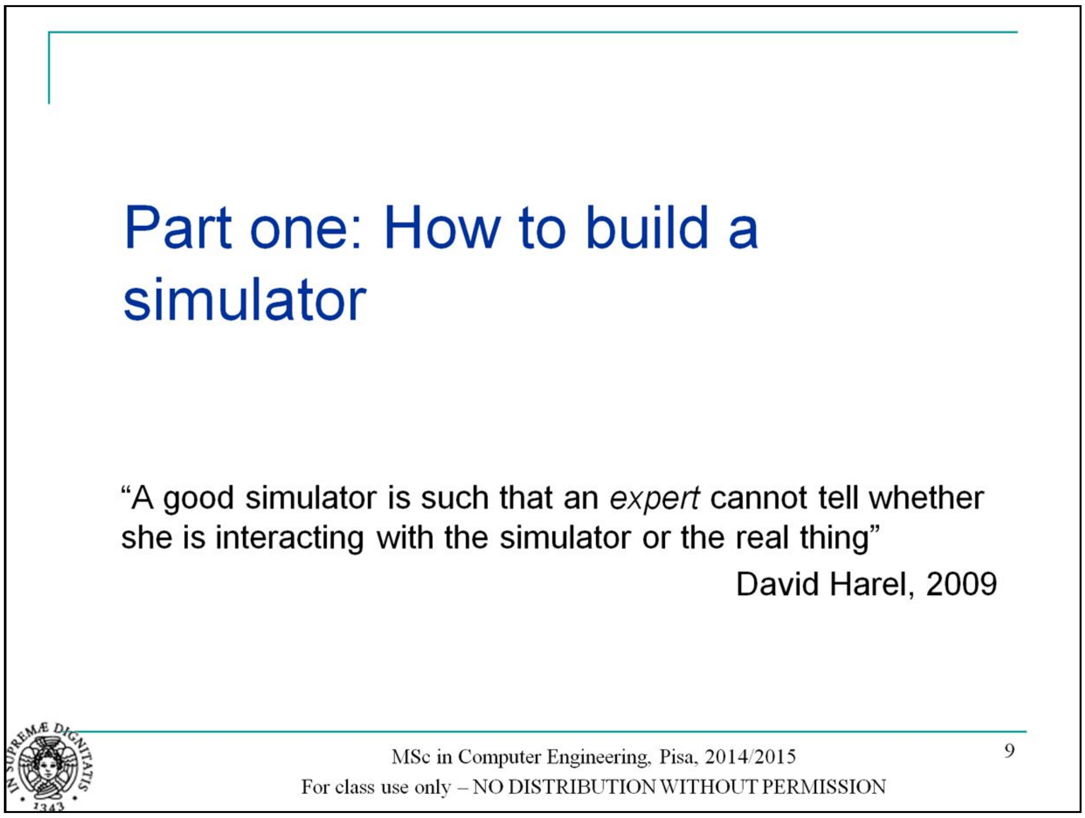
Which makes simulation a typical **interdisciplinary field**.

Never think that simulation analysis is all about **writing code**. This is just the beginning, and it is the **easiest part** for a computer engineer.

It is difficult to realize that a simulator is a replica of **a model of a system**, and not **of the system itself**. A model is a **simplified** thing, where you neglect some aspects, and make abstractions. If, when doing the **modeling part**, you take some **wrong modeling steps**, the fact that you make everything else state-of-the-art is completely **pointless**. The results that you obtain will have nothing to do with reality.

A simulator can normally be employed to produce a **huge amount of information**, e.g. estimates of many interesting quantities. It is easy to build **false confidence** on the correctness of the data when you are faced with a lot of data. You can only be certain when **the same results** are obtained in the **real system**.

(In)Famous example of a perfect simulator of a wrong system model: **the ns-2 module for sensor networks.**



Part one: How to build a simulator

“A good simulator is such that an *expert* cannot tell whether she is interacting with the simulator or the real thing”

David Harel, 2009

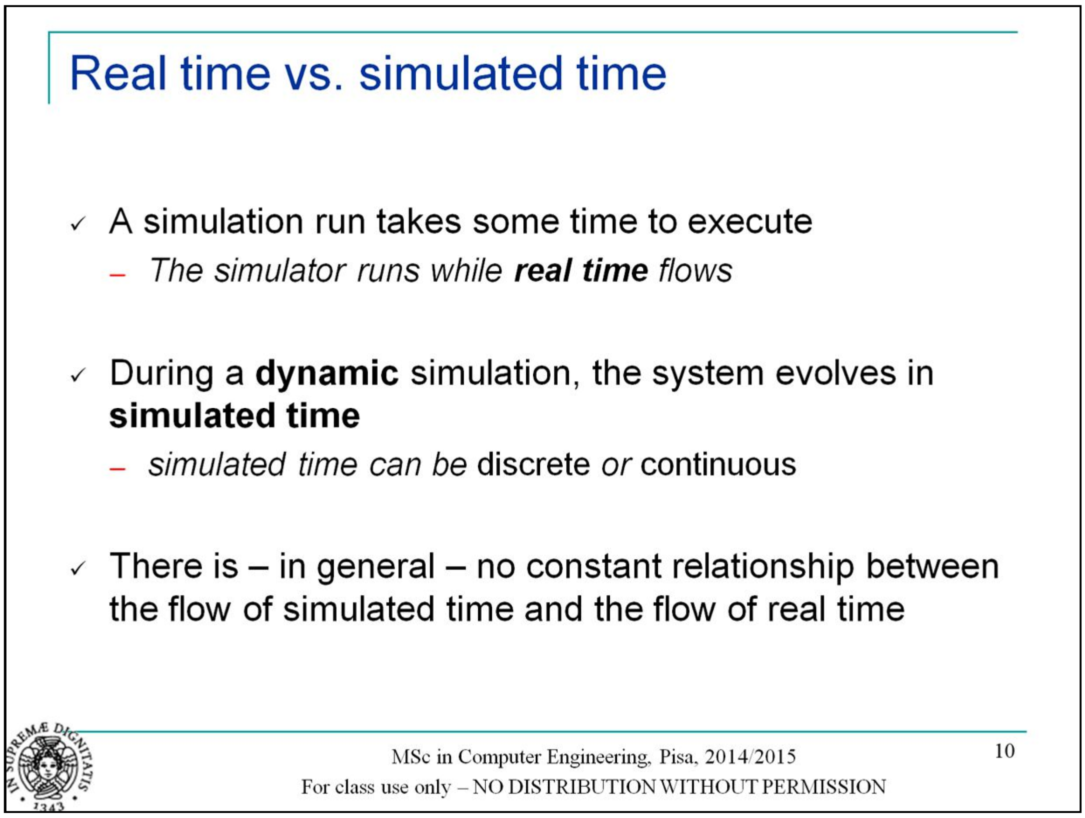
MSc in Computer Engineering, Pisa, 2014/2015

For class use only – NO DISTRIBUTION WITHOUT PERMISSION

9

From now on, for the sake of concreteness, we will suppose that we are analyzing a **network system**. This hypothesis comes with no loss of generality.

You can apply the same analysis to almost everything else.



Real time vs. simulated time

- ✓ A simulation run takes some time to execute
  - *The simulator runs while **real time** flows*
- ✓ During a **dynamic** simulation, the system evolves in **simulated time**
  - *simulated time can be* discrete *or* continuous
- ✓ There is – in general – no constant relationship between the flow of simulated time and the flow of real time

MSc in Computer Engineering, Pisa, 2014/2015 10
For class use only – NO DISTRIBUTION WITHOUT PERMISSION

We are talking about **dynamic simulators**, i.e. those where **time** plays a role. A dynamic simulator is a piece of code that replicates something that is supposed to **happen as time progresses.** There are also **static simulators**, which can be used to solve differential equation systems, or to understand input/output relationships in complex systems. These are not within our interest.

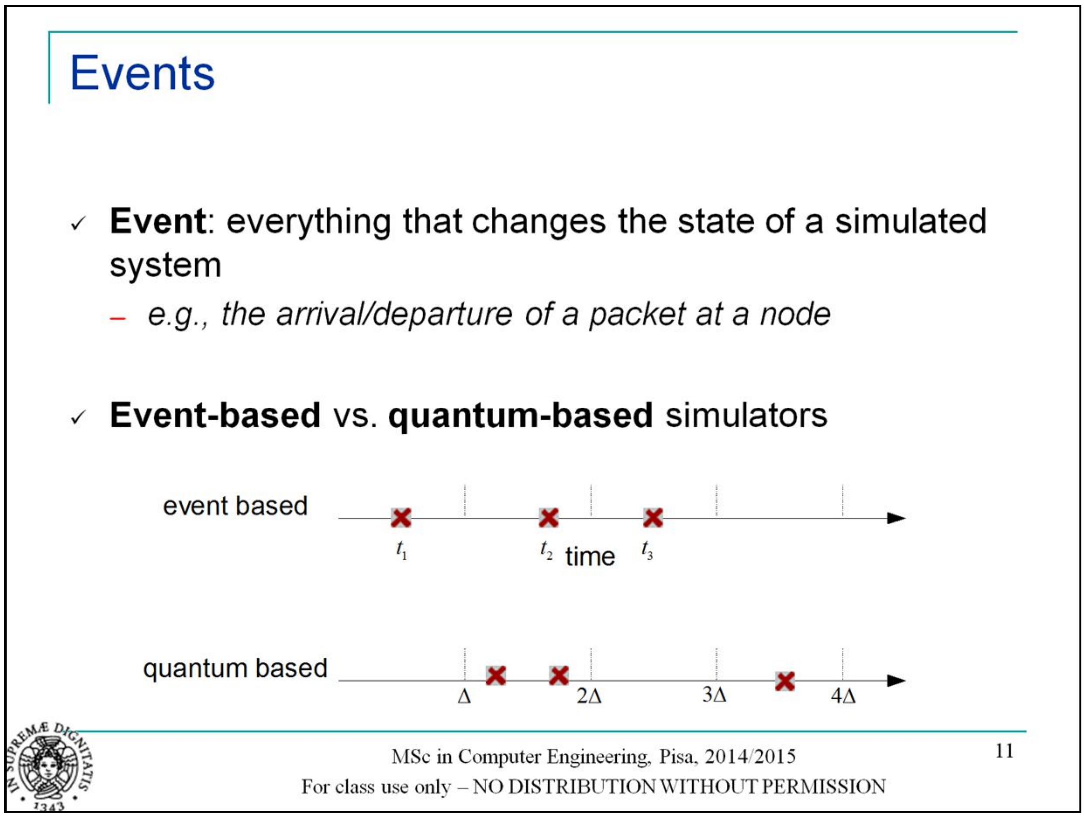
Simulated time: the time as measured **within** the simulation (the time at which events occur in the simulator). This is not the **real time** (otherwise you talk about **emulation**, not **simulation**). In order to simulate one second of simulated time, you may need either more or less than one second of real time, depending on what happens in the simulated system.

In general, the relationship between the two depends on what the system is doing during that span of simulated time:

-Many events -> a larger span of real time

-Few events -> a smaller span of real time per second of simulated time.

Example: to simulate 200s of activity of a 4G cellular network, our simulator takes from 10s to some hours, depending on how many users are in the cell.



Events

- ✓ **Event**: everything that changes the state of a simulated system
  - *e.g., the arrival/departure of a packet at a node*
- ✓ **Event-based** vs. **quantum-based** simulators

event based
t_1
t_2
time
t_3
quantum based
Δ
2Δ
3Δ
4Δ

MSc in Computer Engineering, Pisa, 2014/2015
For class use only – NO DISTRIBUTION WITHOUT PERMISSION
11

How do you make time evolve in a simulator?

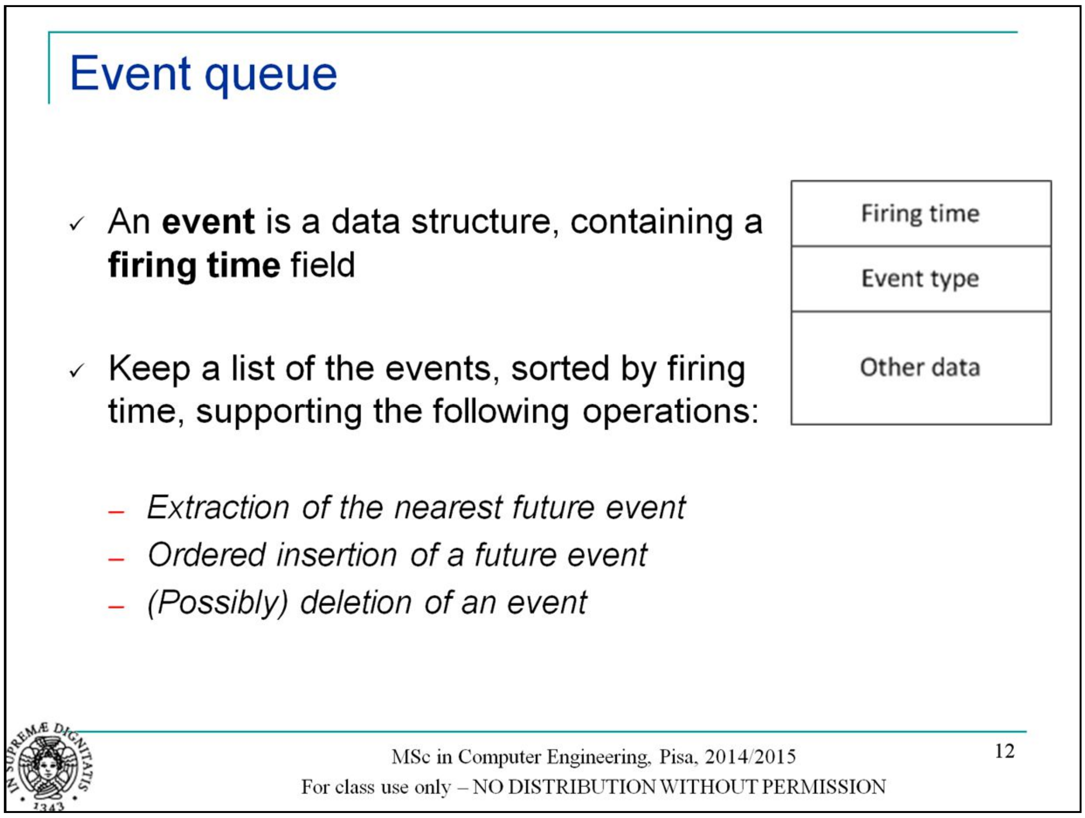
Event-based: simulated time advances when an event is processed. The system processes an event at time t1 (e.g., the arrival of a packet at a node). Processing this event implies **generating** another event (e.g., its departure from the node) at time t2. The system's clock (i.e., simulated time) advances based on the event sequence. Therefore, it is clear that the **real-time duration** of one second of simulated time varies depending on how many events are in that second (and what kind of processing they require).

Quantum-based: the system clock advances **by quanta** (of simulated time). At time 2D, you process all the events that occur between D and 2D. This implies two problems and one advantage:
- If the **granularity of the events is >>D**, you waste a lot of (real) time just to advance the clock when nothing happens. With event-based simulators this would not happen
- If, on the other hand, the granularity is **<<D**, events that are very distant are processed at the same time, hence they are treated as **simultaneous**.
- However, in this case, you can use a **discrete variable** as a temporal quantum (i.e. , an integer instead of a float, which is useful for implementation purposes).

A quantum-based simulator makes some sense if the system is **intrinsically time-discrete** (e.g., ATM, where time is divided into slots). Otherwise it gives you more problems than benefits.

From now on, we will consider **event-based simulators** only. **"discrete-event dynamic simulation" (DEDS)**

I mentioned that "you waste a lot of time": in simulation, **optimizing the system performance** is extremely important (serious simulation studies may last for weeks). The running time of a simulator is something to care about right from the start.



Event queue

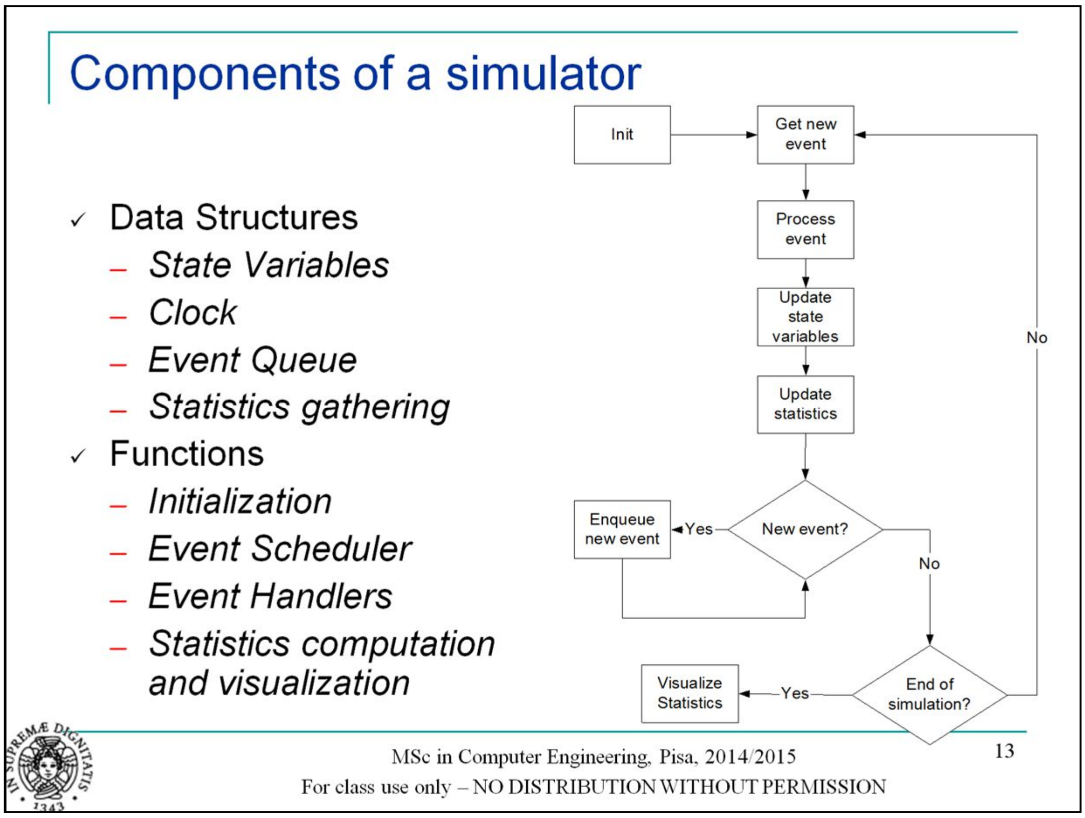
- ✓ An **event** is a data structure, containing a **firing time** field
- ✓ Keep a list of the events, sorted by firing time, supporting the following operations:
  - *Extraction of the nearest future event*
  - *Ordered insertion of a future event*
  - *(Possibly) deletion of an event*

Firing time
Event type
Other data

MSc in Computer Engineering, Pisa, 2014/2015 12

For class use only – NO DISTRIBUTION WITHOUT PERMISSION

Event handling is a critical issue for system performance. Depending on how efficiently the queuing/dequeuing of events is performed, the (real) time that it takes for a simulation to run can easily change by **orders of magnitude.**



Components of a simulator

- ✓ Data Structures
  - *State Variables*
  - *Clock*
  - *Event Queue*
  - *Statistics gathering*
- ✓ Functions
  - *Initialization*
  - *Event Scheduler*
  - *Event Handlers*
  - *Statistics computation and visualization*

Init
Get new event
Process event
Update state variables
Update statistics
New event?
Yes
Enqueue new event
No
End of simulation?
Yes
Visualize Statistics
No

MSc in Computer Engineering, Pisa, 2014/2015
For class use only – NO DISTRIBUTION WITHOUT PERMISSION
13

Events in a simulator are handled in an **event queue**, which has to be implemented in the **most efficient possible way** (it is the critical point for the performance of a simulator).

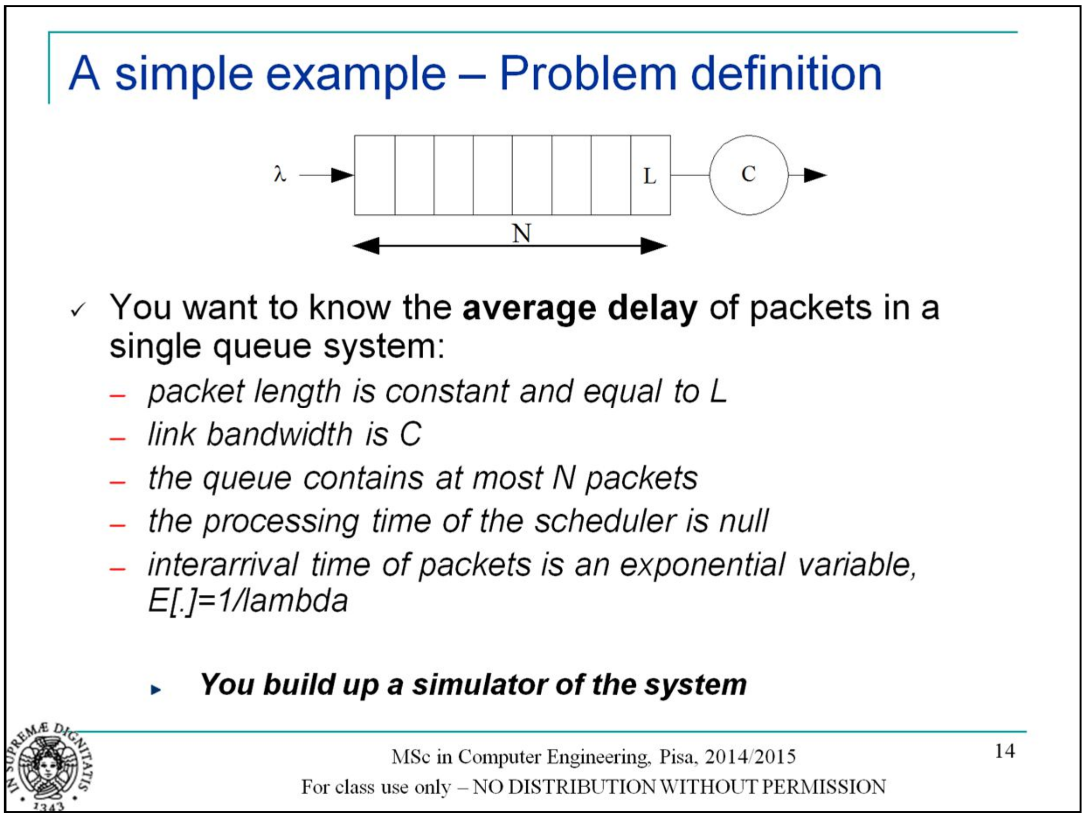
The activity of a simulator consists of a continuous cycle where:

-You extract the nearest future event from the queue, thus advancing the simulated time (**clocking function**)

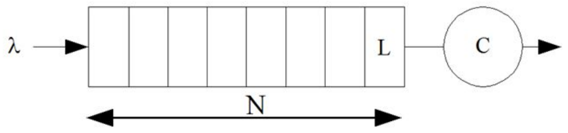
-You process that event using an **event handler**. In doing this:

- You update the state of the simulator,
- You update the statistics that you want to observe at the end of the simulation
- Possibly, you generate one or more new events, and insert them into the event queue

-At the end of the simulation, you visualize the statistics of interest.



A simple example – Problem definition



- You want to know the **average delay** of packets in a single queue system:
  - *packet length is constant and equal to L*
  - *link bandwidth is C*
  - *the queue contains at most N packets*
  - *the processing time of the scheduler is null*
  - *interarrival time of packets is an exponential variable, E[.]=1/lambda*

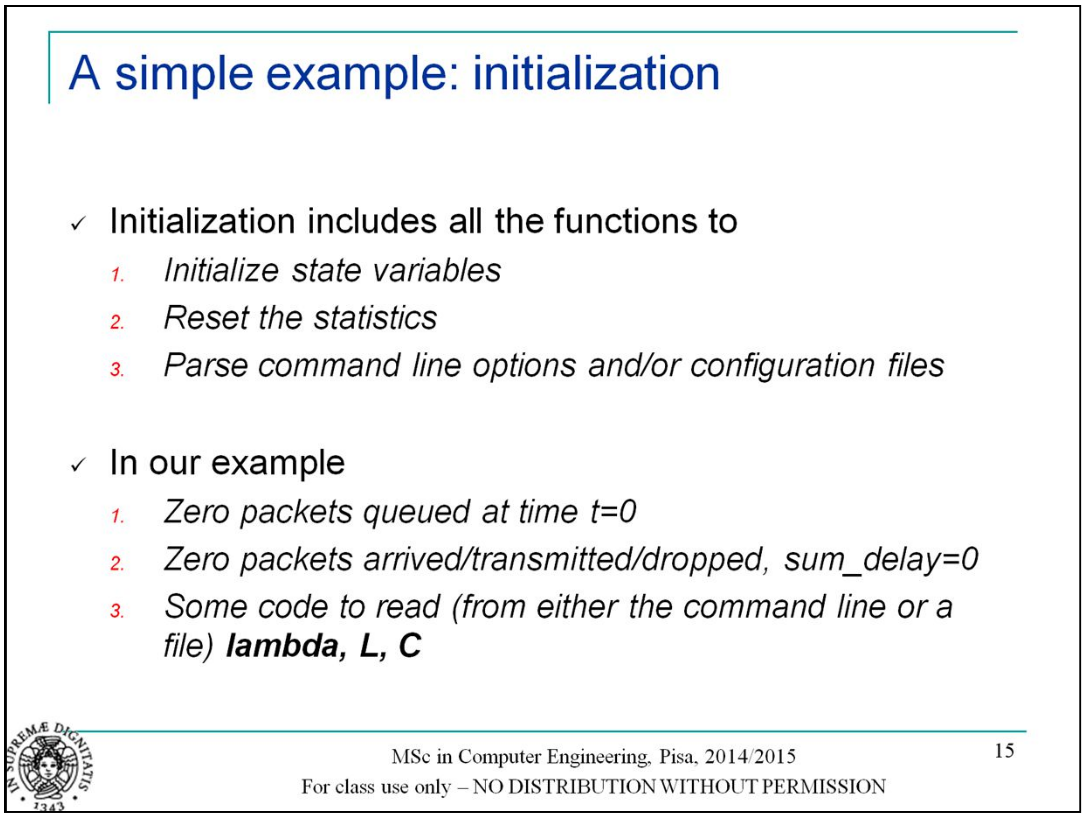
    - ***You build up a simulator of the system***

MSc in Computer Engineering, Pisa, 2014/2015
For class use only – NO DISTRIBUTION WITHOUT PERMISSION
14

Take a system (or, to be precise, a **model** of a system, i.e. the abstract representation of something – a node in a computer network with a FIFO service). This is something that can be **fully described** using analytical techniques, of course (it is an M/D/1/N queuing system).

In this case, the ***state*** of the system is the number of packets in the queue.

We go through the event loop of the previous slide using this example as a case study.



A simple example: initialization

- ✓ Initialization includes all the functions to
  1. *Initialize state variables*
  2. *Reset the statistics*
  3. *Parse command line options and/or configuration files*
- ✓ In our example
  1. *Zero packets queued at time t=0*
  2. *Zero packets arrived/transmitted/dropped, sum_delay=0*
  3. *Some code to read (from either the command line or a file)* ***lambda, L, C***

MSc in Computer Engineering, Pisa, 2014/2015 15
For class use only – NO DISTRIBUTION WITHOUT PERMISSION

If you want to compute the mean delay, you should keep **two counters**: one for the number of **transmitted packets**, one for the **sum of the delays**. The latter are to be incremented on every packet departure. At the end we will output the ratio of the second to the first.

Of course you can also

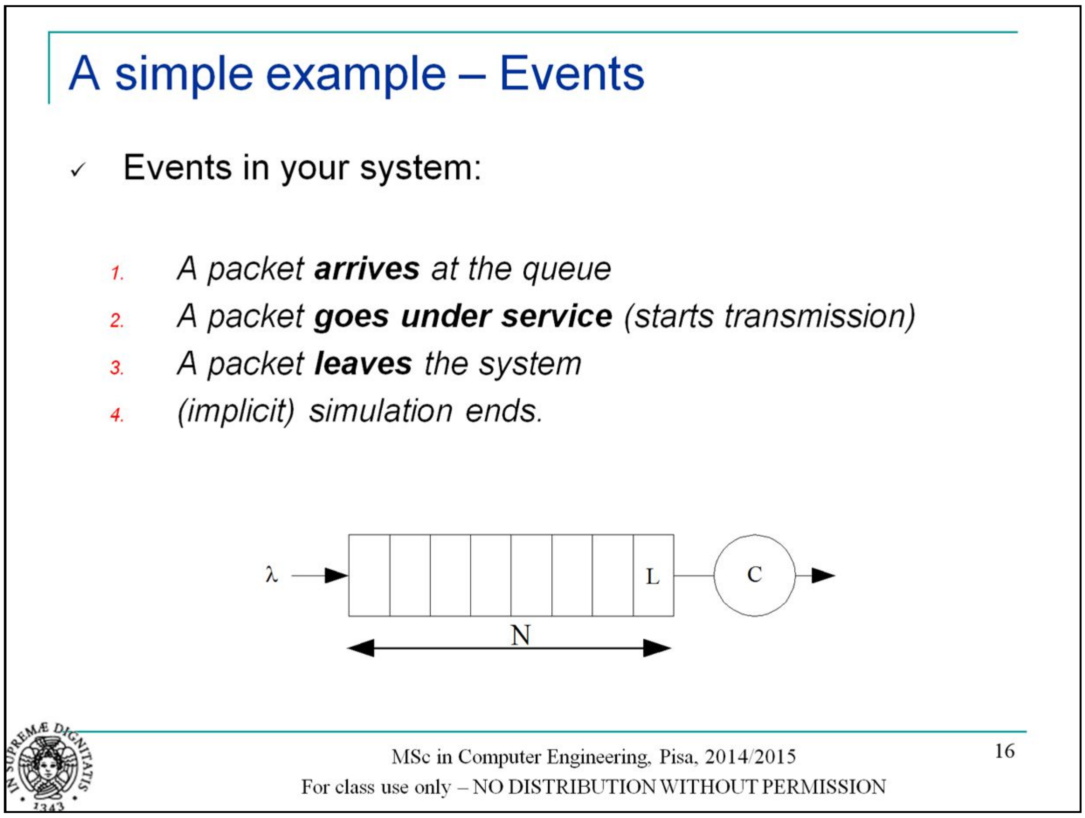
Maintain other statistics (i.e., the number of dropped packets, etc.) for future use.

In a simulator that has some more functionalities than the toy example we are using as a reference, **configuration parameters** may be in the **tens or hundreds**. In this case, it is extremely impractical to pass them through the command line.

For this reason, you normally define **scenario files**, where you store, using a **textual syntax**, the values of the parameters that are used in a simulation run.

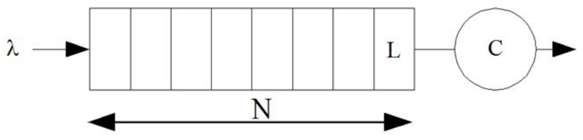
The syntax **should** be textual, because scenarios are indeed written by a user, and the user should be able to read them again after some months and remember what she did without wasting too much time.

In any case, you also need some code to **parse input configuration parameters**, either from the command line or from a configuration file. Parameters are **never hardcoded** within the simulator (otherwise you would be forced to recompile it every time a scenario changes).



A simple example – Events

- ✓ Events in your system:
    1. *A packet **arrives** at the queue*
    2. *A packet **goes under service** (starts transmission)*
    3. *A packet **leaves** the system*
    4. *(implicit) simulation ends.*

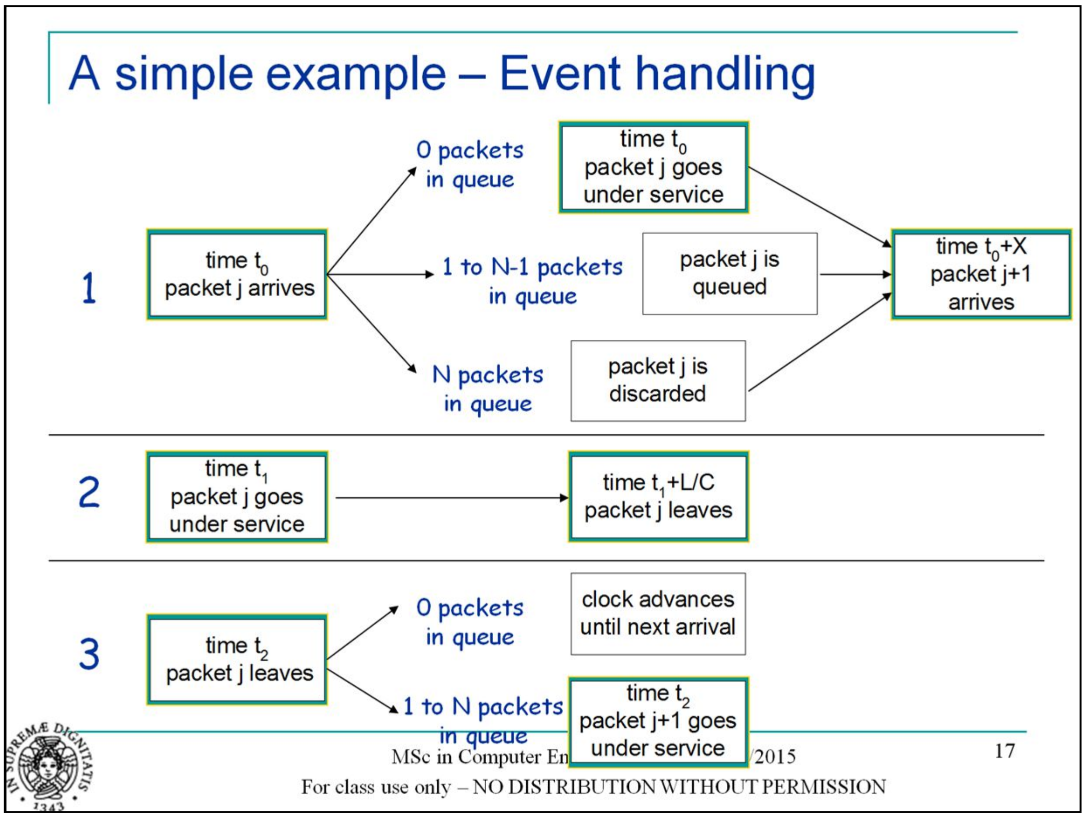


Why should ***these*** be events?

Because they **change the state of the system**.

The "end of simulation" is normally modeled as an event, which is inserted in the event queue at the beginning of the simulation (this is part of the init function). This is an event because you may want to handle it in a special way (e.g., to compute "a posteriori" statistics, etc.).

Since every event has its own handler, it is customary to make all end-of-simulation processing part of the handler for the "simulation end" event. In our case, that handler should compute the ratio between the sum total of delays and the number of transmitted packets and print it (plus some other tasks, possibly).


A simple example – Event handling
1
time t_0 packet j arrives
0 packets in queue
time t_0 packet j goes under service
1 to N-1 packets in queue
packet j is queued
N packets in queue
packet j is discarded
time t_0+X packet j+1 arrives
2
time t_1 packet j goes under service
time t_1+L/C packet j leaves
3
time t_2 packet j leaves
0 packets in queue
clock advances until next arrival
1 to N packets in queue
time t_2 packet j+1 goes under service
MSc in Computer En… /2015
17
For class use only – NO DISTRIBUTION WITHOUT PERMISSION

Events are **circled in yellow** in this picture. Handling these events may generate more events.

[**Describe the events** – with reference to the simulator flowchart some slides ago]

Observation: given that the length of packets is constant, a type-2 event could also be eliminated. More so if you consider that a type-2 event always generates a new type-3 event at a constant temporal distance, whereas a type-3 event also generates a new type-2 event at a constant (null) temporal distance. Why keeping these separate, then?

Because, in a future, you may want to **modify your simulator** so as to keep into account variable-sized packets, and you will want to do that without re-writing it anew.

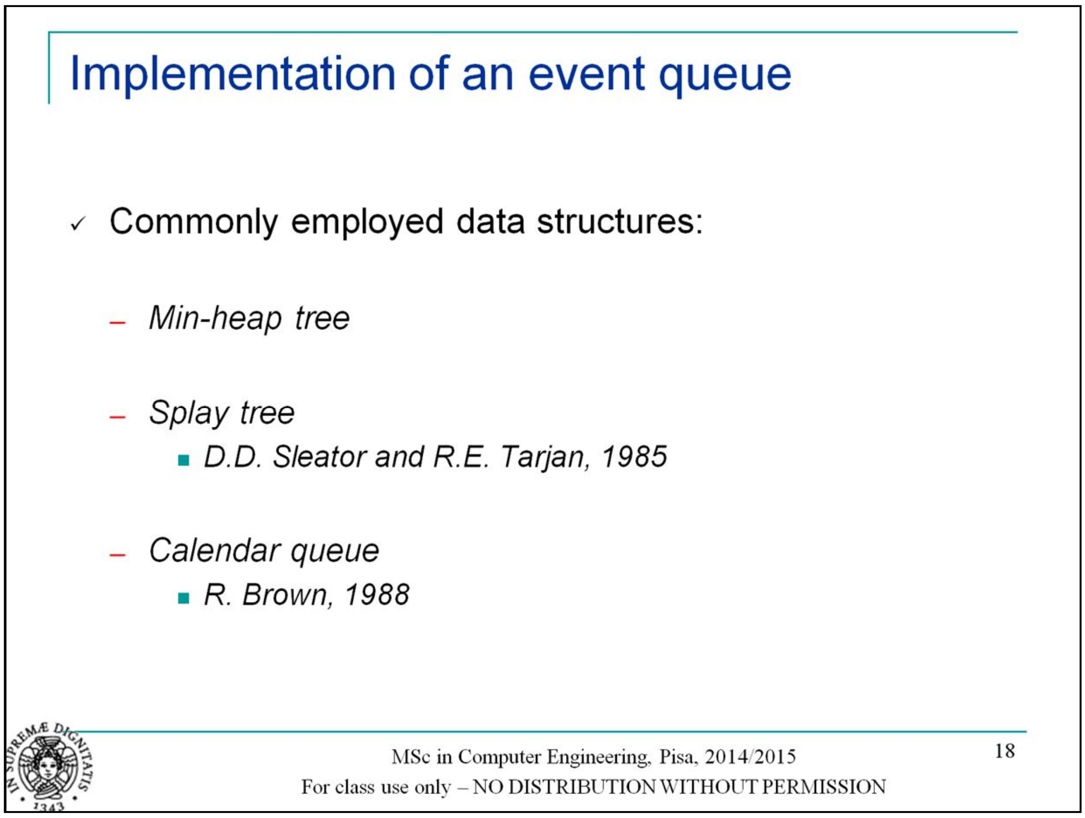
What data structure should I use to store the events? For a system like this, the most efficient choice would probably be to devise an ad-hoc data structure. For instance, if L/C << E[X], it is very likely that

- Type-1 events are **tail** insertions
- Type-2/3 events are **front** insertions.

This knowledge could be exploited to make the system more efficient.

Take care not to **optimize too much**, because the more you optimize, the less flexible your simulator becomes. You have to strike a **trade-off** between flexibility (reconfigurability) and performance.

There are, however, safe choices which are near-optimal in most cases.

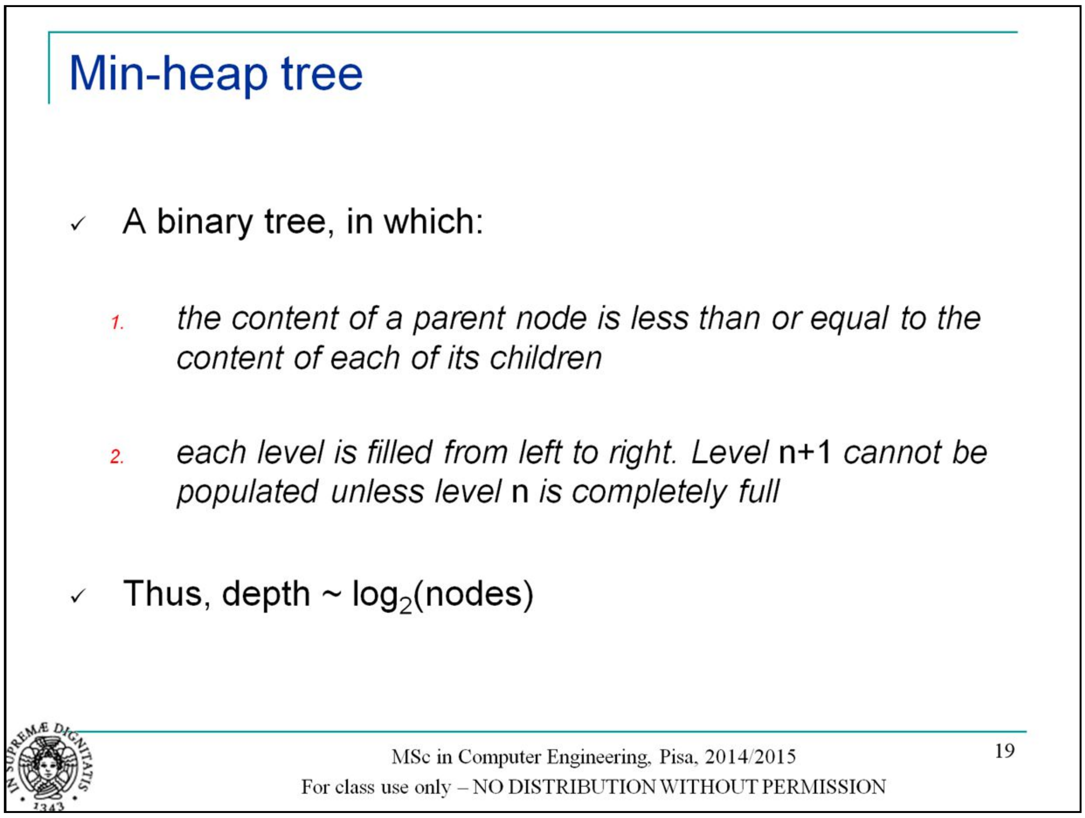


Implementation of an event queue

- Commonly employed data structures:
  - *Min-heap tree*
  - *Splay tree*
    - *D.D. Sleator and R.E. Tarjan, 1985*
  - *Calendar queue*
    - *R. Brown, 1988*

MSc in Computer Engineering, Pisa, 2014/2015
For class use only – NO DISTRIBUTION WITHOUT PERMISSION
18

If you profile your code using a profiler (e.g., Valgrind), you quickly discover that a significant portion of real time is spent by handling the event queue.



Min-heap tree

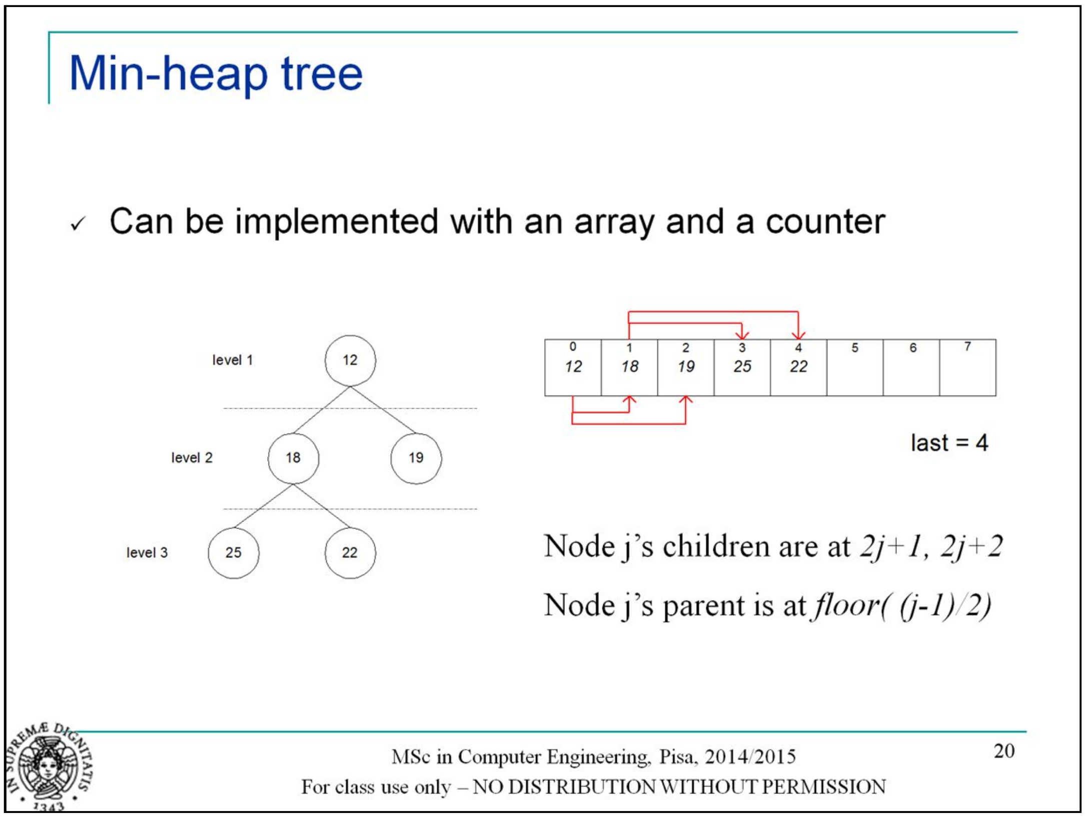
- ✓ A binary tree, in which:
  1. *the content of a parent node is less than or equal to the content of each of its children*
  2. *each level is filled from left to right. Level* n+1 *cannot be populated unless level* n *is completely full*
- ✓ Thus, depth ~ $\log_2$(nodes)

MSc in Computer Engineering, Pisa, 2014/2015
For class use only – NO DISTRIBUTION WITHOUT PERMISSION
19

The basic observation here is that you do not need to keep **all the events sorted**: you just need to be able to find the **next in time** quickly. For this, you can use a min heap.

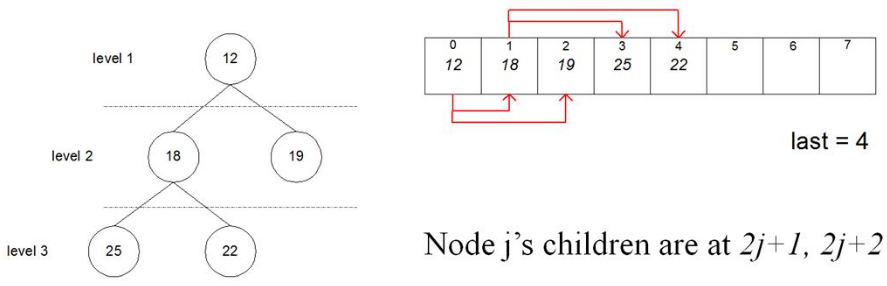
Note that, if you preserve the property of **quasi-completeness** when making insertions/extractions, the tree depth is **logarithmic** with the number of nodes (events).

In this case, we use the **event firing time** as a sorting key.



Min-heap tree

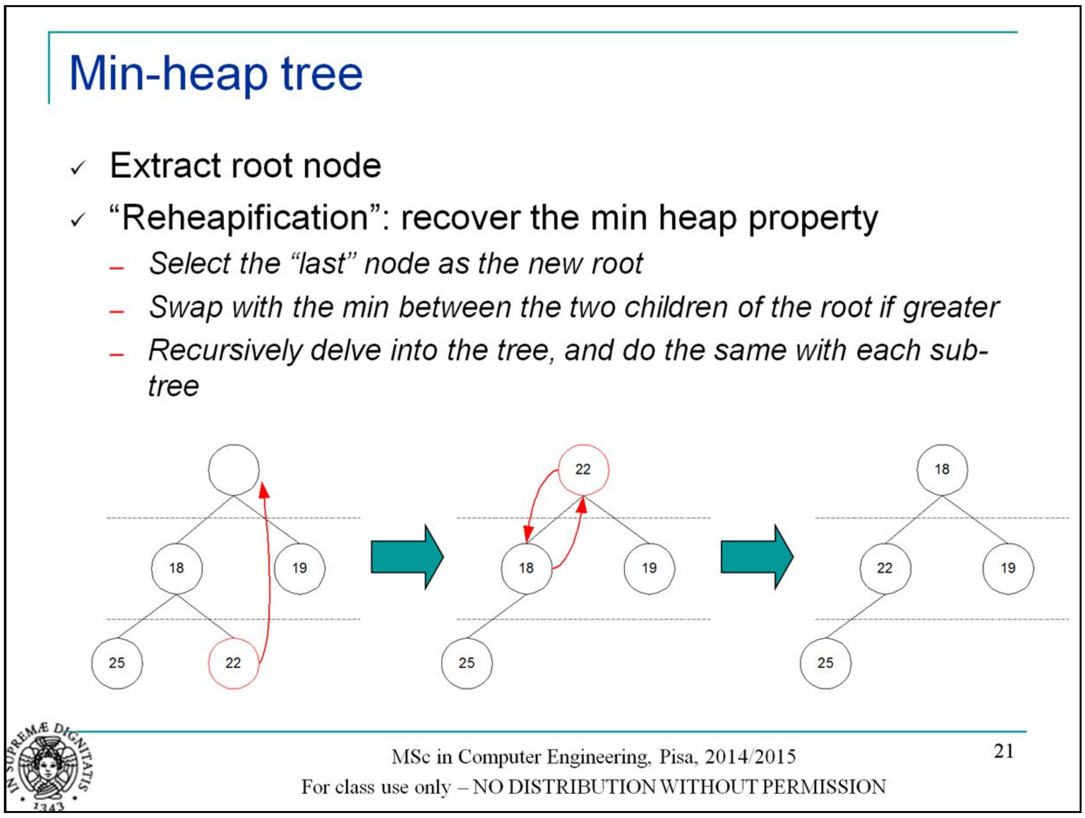
- ✓ Can be implemented with an array and a counter



Node j's children are at $2j+1,\ 2j+2$

Node j's parent is at $floor(\ (j\text{-}1)/2)$

The counter is needed to know how many positions are occupied in the array.
A min heap of a **known maximum size** can be implemented very efficiently.

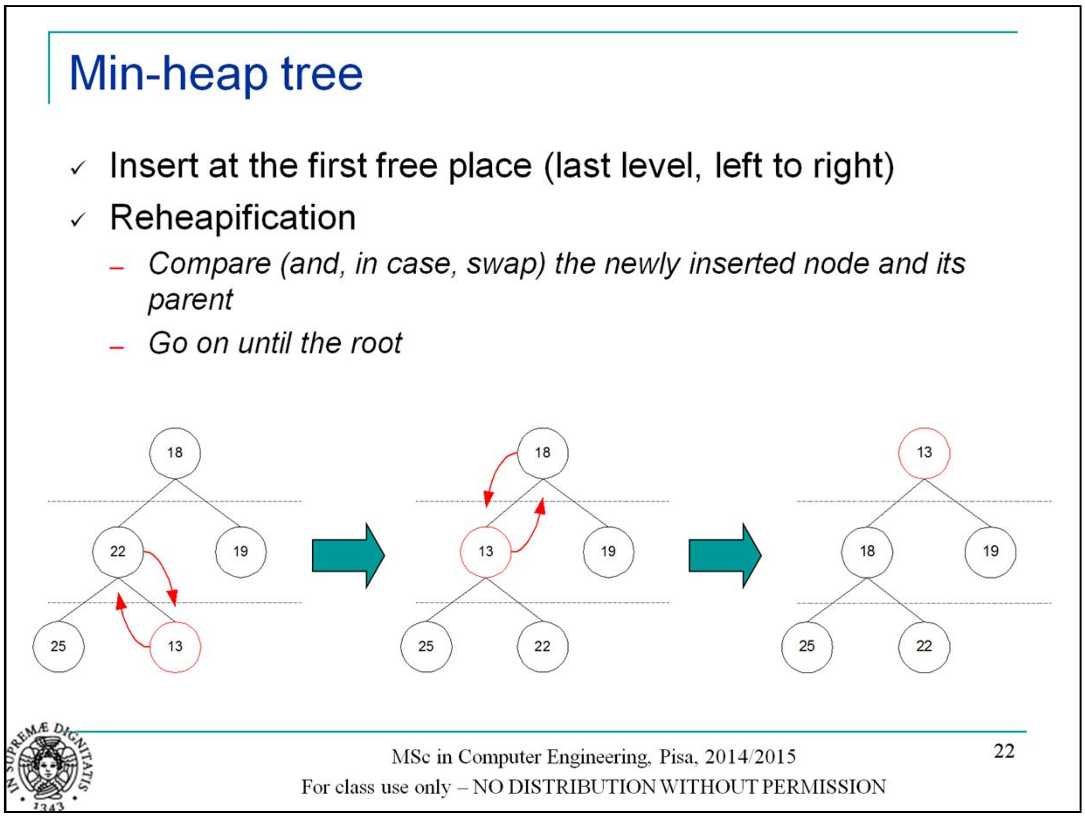


Min-heap tree

- ✓ Extract root node
- ✓ "Reheapification": recover the min heap property
  - *Select the "last" node as the new root*
  - *Swap with the min between the two children of the root if greater*
  - *Recursively delve into the tree, and do the same with each sub-tree*

18
19
25
22
22
18
19
25
18
22
19
25

The extraction of the nearest future event is **the extraction of the root node**.
After the root has been extracted, **reheapification** occurs, i.e. the min-heap property is obtained again.



Min-heap tree

- ✓ Insert at the first free place (last level, left to right)
- ✓ Reheapification
  - *Compare (and, in case, swap) the newly inserted node and its parent*
  - *Go on until the root*

18
22
19
25
13
18
13
19
25
22
13
18
19
25
22

MSc in Computer Engineering, Pisa, 2014/2015
For class use only – NO DISTRIBUTION WITHOUT PERMISSION

22

The **maximum** cost of an insertion/extraction in a minheap is **O(logN),** since it is at most equal to the number of levels among which you need to swap. This is because the tree is quasi-complete.

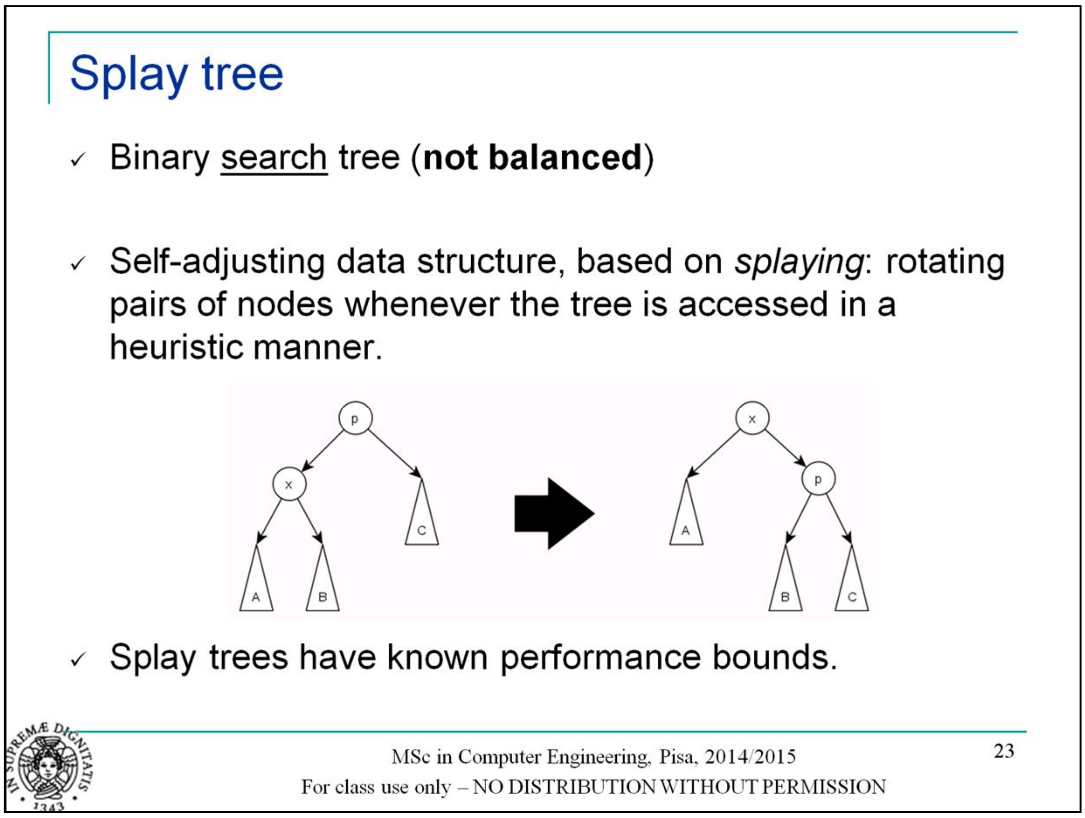
What about **deletion of an event from the heap?**

- Once you locate it, cancel it and put the **last event on the bottom level** in its stead (thus preserving quasi-completeness)
- Reheapify (either upward or downward), which is still O(logN)

So, is deletion O(logN) as well?

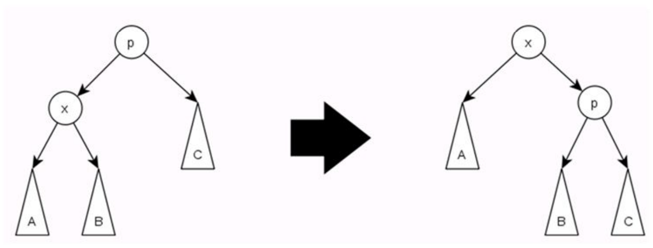
The answer is no, since you need to **locate** the element first, and this takes O(N). This is not a **search** tree, and the min-heap property is not enough to locate an element in log time.

Bottom line: if you need frequent event deletion, don't use a min heap.



Splay tree

- Binary search tree (**not balanced**)
- Self-adjusting data structure, based on *splaying*: rotating pairs of nodes whenever the tree is accessed in a heuristic manner.



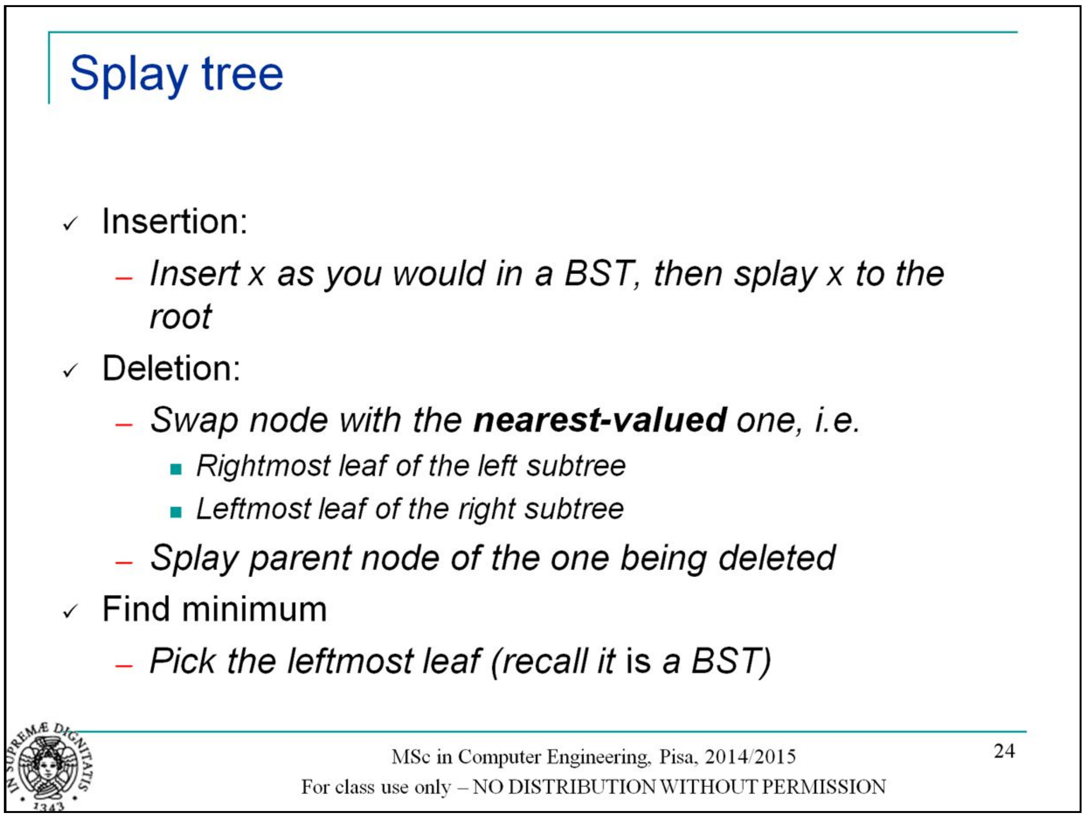
- Splay trees have known performance bounds.

A binary search tree is one where the left subtree of a node contains **smaller** values, and the right subtree contains **higher** values than the root (recursively down, of course).

Splay trees are **not balanced**. Their height can thus be O(n), whereas a balanced tree (e.g., a min-heap) has O(log n) height. However, the **amortized complexity** for a splay tree is still logarithmic.

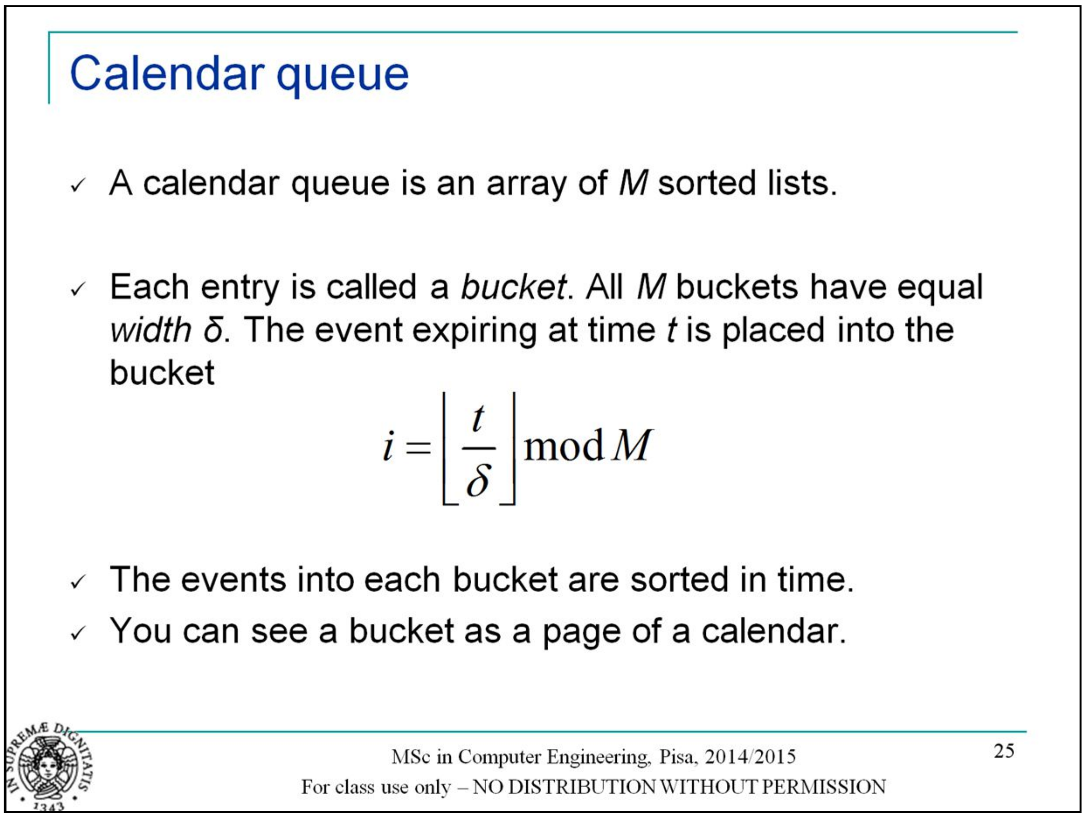
In splay trees, whenever an element is accessed, a **splay operation** brings it to the root of the tree. This way, elements close to those that have been accessed recently will be close to the root (locality of references, which is useful in tons of applications).

Triangles can be sub-trees, not necessarily elements in this picture.



Splay tree

- ✓ Insertion:
  - *Insert x as you would in a BST, then splay x to the root*
- ✓ Deletion:
  - *Swap node with the **nearest-valued** one, i.e.*
    - *Rightmost leaf of the left subtree*
    - *Leftmost leaf of the right subtree*
  - *Splay parent node of the one being deleted*
- ✓ Find minimum
  - *Pick the leftmost leaf (recall it* is *a BST)*



Calendar queue

- ✓ A calendar queue is an array of *M* sorted lists.
- ✓ Each entry is called a *bucket*. All *M* buckets have equal *width δ*. The event expiring at time *t* is placed into the bucket

$$i = \left\lfloor \frac{t}{\delta} \right\rfloor \bmod M$$

- ✓ The events into each bucket are sorted in time.
- ✓ You can see a bucket as a page of a calendar.

MSc in Computer Engineering, Pisa, 2014/2015
For class use only – NO DISTRIBUTION WITHOUT PERMISSION

25

[Describe the Calendar queue]

If the events are distributed more or less uniformly, the probability that you find an "empty" calendar day (bucket) is quite small.

Moreover, if the calendar year is «long», then the number of collisions of events of different years on the same day is low.

-> the number of operations to perform in order to locate the next event is low **on average**

In a worst case, it may happen that:

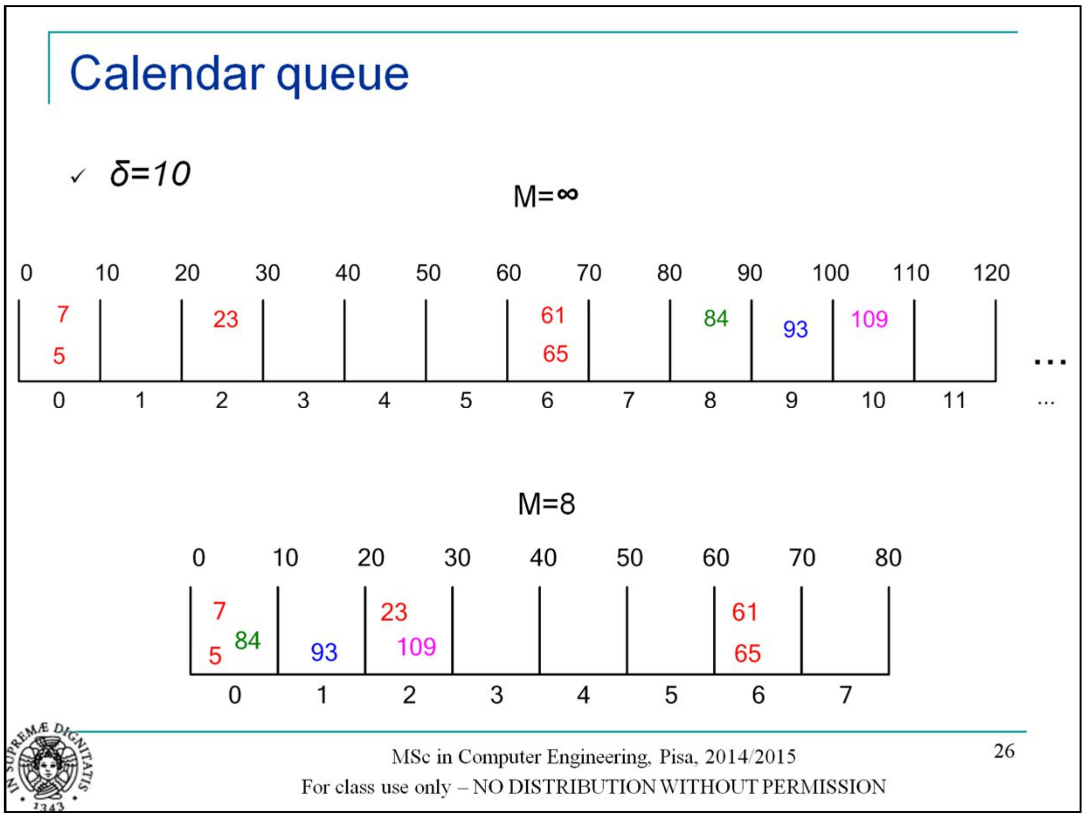
- All events are in the same bucket – O(N) cost in a linear queue, O(logN) if min-heap is used
- You need to cycle through O(M) empty buckets

So the calendar queue has a **worst-case** cost of O(N+M) – or O(logN+M), although the **average** number of operations required to insert/extract events is quite small.

Which of the two should I pay attention to, the worst-case or the average cost?

It depends on what you need to do

a) worst-case cost <-> you have to guarantee a **maximum execution time** (e.g., a real-time system)
b) Average cost <-> you have to minimize the overall running time of a system (e.g., a simulator)



Calendar queue

- δ=10

Events in bucket 0 have to be ordered (e.g., using a min heap).
Events queued on each bucket have to be ordered w.r.t. their firing time.

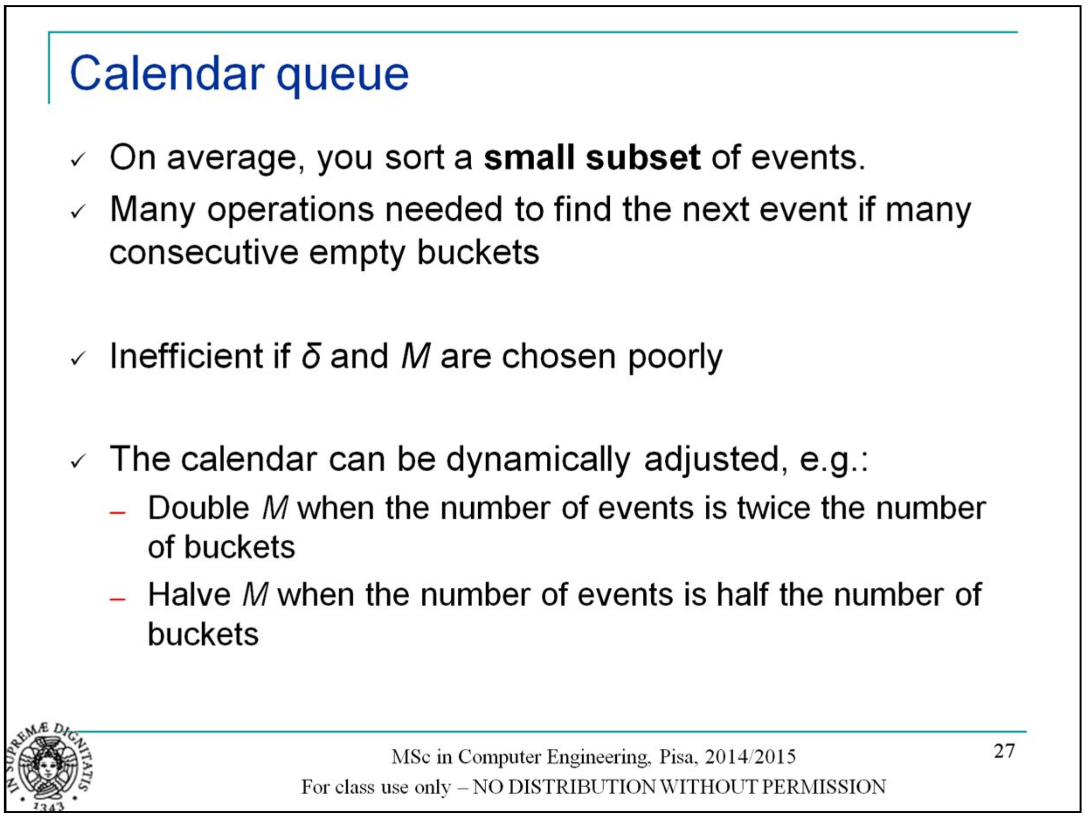
If:
-Events are **evenly distributed** in the buckets
-There aren't too many **consecutive empty buckets**

Then it is quite easy to locate the next event to be scheduled, and it is not computationally expensive to keep each bucket's queue sorted (the cost of sorting grows with the number of events to be sorted).

Note that there is no ambiguity related to the fact that the same calendar page (bucket) contains event related to different years (e.g., 23 and 109 in bucket 2). If bucket 2 is the current bucket, and 23 is the next event to be scheduled, it is perfectly obvious that 109 is an event for the next year, and should only be considered at the subsequent round, after the event scheduler has cycled all the way back to bucket 2.

You just need to maintain a **year counter**

```
At year j:
If (firing time of top event in current bucket) >= j*M*delta
Then
        increase bucket                 // not in the current year
Else
        pick next event from current bucket
```


Calendar queue

- ✓ On average, you sort a **small subset** of events.
- ✓ Many operations needed to find the next event if many consecutive empty buckets
- ✓ Inefficient if δ and M are chosen poorly
- ✓ The calendar can be dynamically adjusted, e.g.:
  - Double M when the number of events is twice the number of buckets
  - Halve M when the number of events is half the number of buckets

MSc in Computer Engineering, Pisa, 2014/2015
For class use only – NO DISTRIBUTION WITHOUT PERMISSION

27

Delta and *M* can be chosen

-Statically

-Dynamically

If they are **static**, then the efficiency of a CQ is determined by their value as related to the granularity of events (there are optimal choices if you know how your events are interleaved in time). More specifically, if you know the **event hold time**, i.e. the RV that measures the time between the *scheduling* and the *firing* of an event, then you can compute optimal values for the two parameters. Just Google "optimizing calendar queues" to see some of the available literature on the subject.

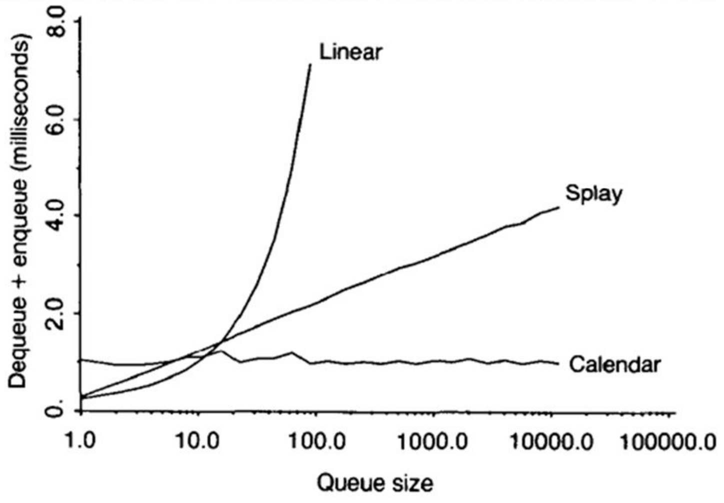
If they are **dynamic**, the queue can be dynamically adjusted. For instance, the # of buckets can be adapted to the # of events (with some hysteresis, lest it oscillates too often). There are two strategies to do this:

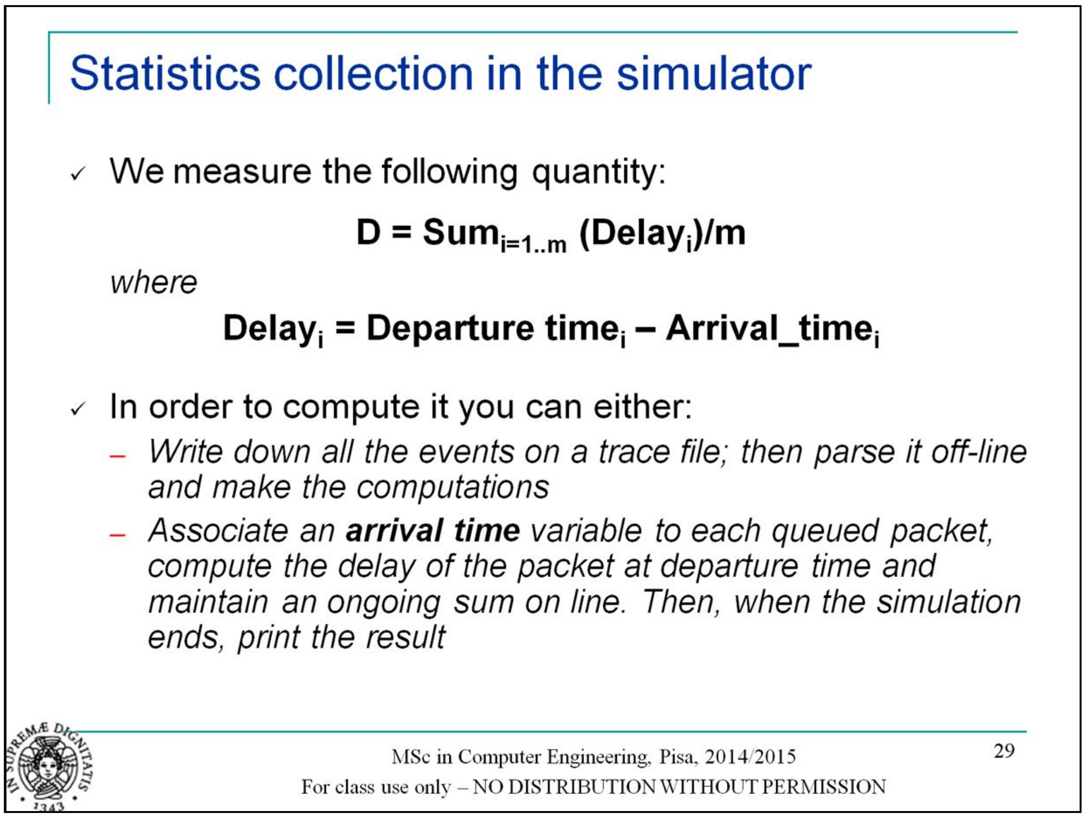
-**freeze**: when you want to use a different *M,* you stop the simulation, build the new CQ, move all the events from the **old queue** to the **new queue**, and then you resume (cons: you waste time managing your data structure while simulated time is, in fact, frozen).

-**Gradual**: you build the new queue, and start populating it with the new events. While you do this, you keep extracting events from the old CQ, until it is empty, and then discard it. The problem is that you still need to check whether the **next event** is in the **old queue or in the new queue** (and you have to move the "end of simulation" event in any case)

Calendar queue

- ✓ R. Brown. Calendar Queues: A Fast O(1) Priority Queue Implementation for the Simulation Event Set Problem. 1988.





Statistics collection in the simulator

- ✓ We measure the following quantity:

$$D = \mathrm{Sum}_{i=1..m}\,(\mathrm{Delay}_i)/m$$

where

$$\mathrm{Delay}_i = \text{Departure time}_i - \text{Arrival\_time}_i$$

- ✓ In order to compute it you can either:
  - *Write down all the events on a trace file; then parse it off-line and make the computations*
  - *Associate an* ***arrival time*** *variable to each queued packet, compute the delay of the packet at departure time and maintain an ongoing sum on line. Then, when the simulation ends, print the result*

MSc in Computer Engineering, Pisa, 2014/2015 29
For class use only – NO DISTRIBUTION WITHOUT PERMISSION

[Recall the simulator flowchart of a few slides ago]

A simulator processes events in order to compute some output numbers. The part devoted to computing the measures of interest is called **statistics collection**. Let us review what fundamental choices we have to meet on that part.

Assume that we want the average delay of a flow of packets as a result. There are **two (extreme) strategies to obtain it**:

a) Log all the events in the simulation (related to packet arrival/departure etc.) on a **trace file**. After the simulation terminates, take the trace file and parse it, thus computing the sample mean of the delay

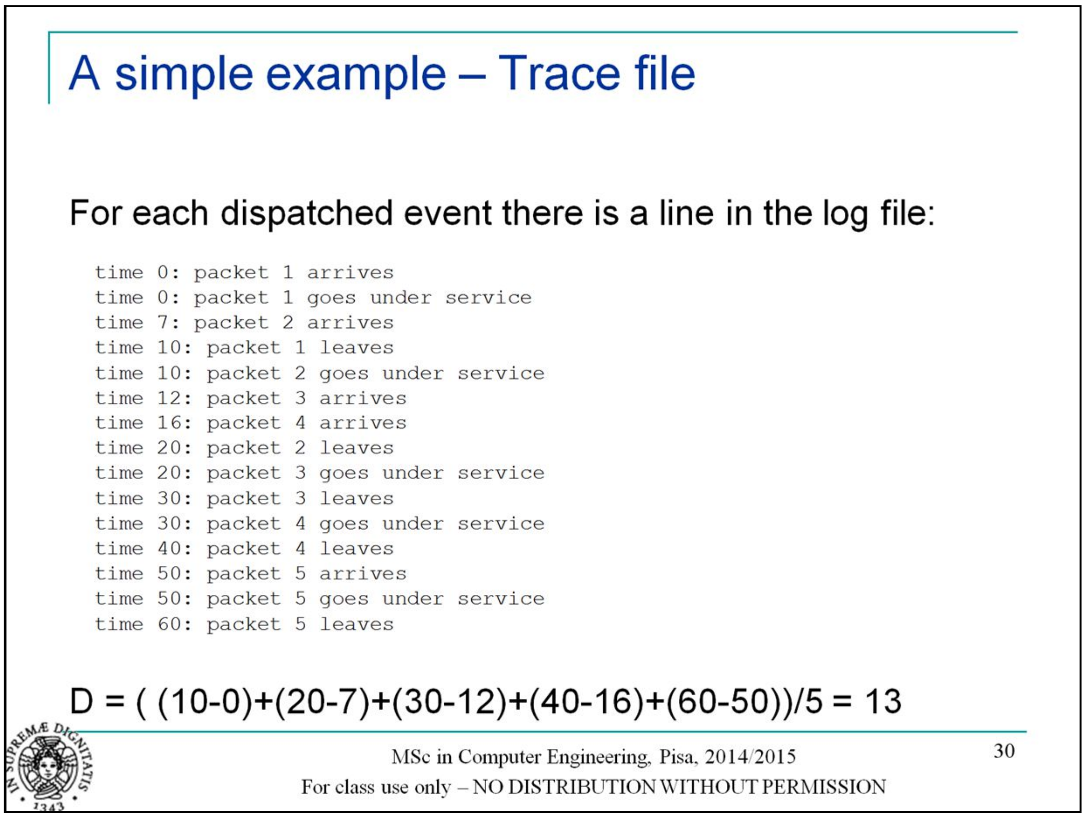
b) Store all the information you need in **internal data structures** (those that we called “statistics gathering” some slides ago), and compute the final result only. In this case, it is enough that you use a) a counter, which stores the sum, and b) a counter which is increased at every packet departure.

These are the two extremes, but anything in between is possible. For instance, you may want to use a tracefile, and log the delay of each packet (instead of logging arrival/departures etc.). Or you may want to store the event trace in memory, and then dump it all on the disk only at the end of the simulation. What are the pros/cons of each approach?

A) Longer execution times: **disk writes cost a lot** On the other hand, you get **much more information** from a complete tracefile. For instance, I might want to know the **sample variance** of the delay, and I would be able to compute it from the tracefile without i) modifying the simulator, and ii) running the same simulation again. A tracefile gives you a lot of information which is useful during the **debugging phase**.

B) Shorter execution times (often considerably so), less flexibility (you can only get out of the simulator what you programmed it for).

The optimal choice depends, thus, on your objectives. Event logging is normally most useful during the debug phase, and should be disabled afterwards (unless you have good cause to do otherwise).



A simple example – Trace file

For each dispatched event there is a line in the log file:

```
time 0: packet 1 arrives
time 0: packet 1 goes under service
time 7: packet 2 arrives
time 10: packet 1 leaves
time 10: packet 2 goes under service
time 12: packet 3 arrives
time 16: packet 4 arrives
time 20: packet 2 leaves
time 20: packet 3 goes under service
time 30: packet 3 leaves
time 30: packet 4 goes under service
time 40: packet 4 leaves
time 50: packet 5 arrives
time 50: packet 5 goes under service
time 60: packet 5 leaves
```

D = ((10-0)+(20-7)+(30-12)+(40-16)+(60-50))/5 = 13

Note that this approach allows you to compute other quantities, e.g.
-Buffer occupancy
-Throughput
-Jitter
-…

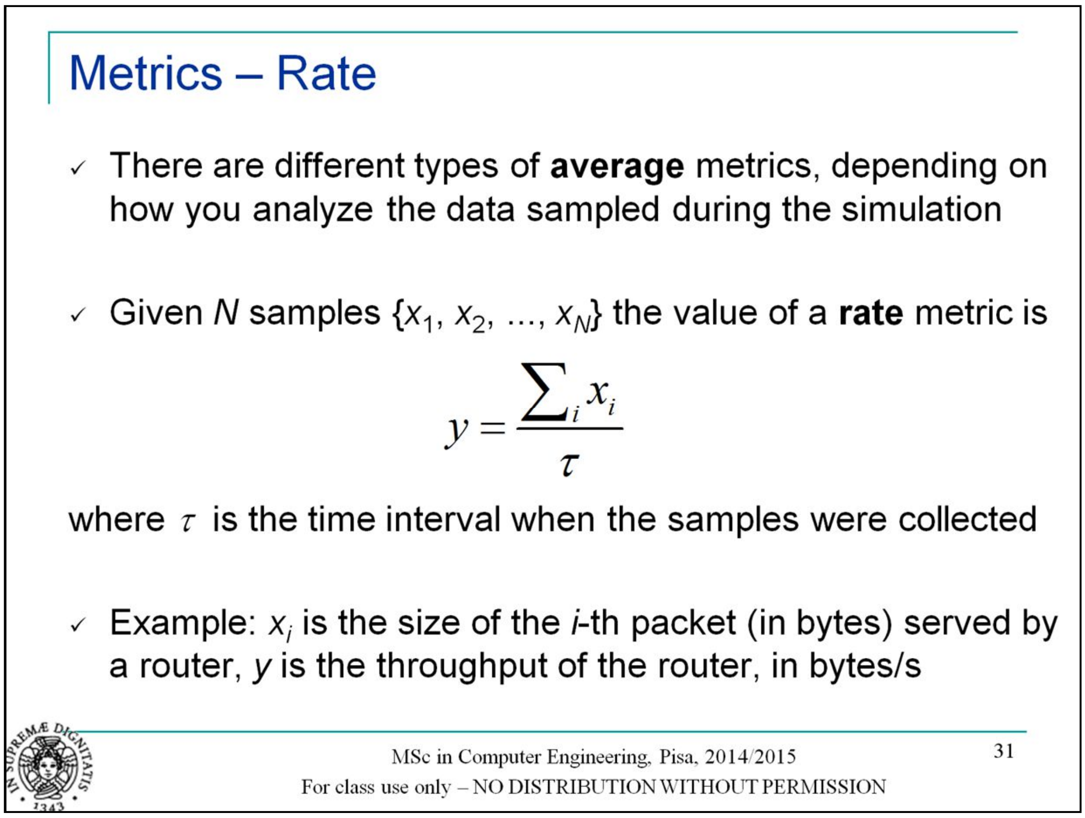
In order to parse these files, you should resort to:
-Scripting languages (PERL, Python, etc.)
-Ad-hoc existing tools (GAWK)
-Home-brewed C code (strongly discouraged: it takes a lot of time, it is highly inflexible, and error prone).

Backlog computation in GAWK (more or less):

```
BEGIN       {count=0;}
$5=="arrives"          {count++; print $2, count;}
$5=="leaves"           {count--; print $2, count;}
```

This gets you a column-formatted file with records (t, b), which is ready for plotting

In the time you get to do the same in C++, you can install Cygwin (if you work under Windows), read all the gawk man page, write the above code and be done with it.



Metrics – Rate

- ✓ There are different types of **average** metrics, depending on how you analyze the data sampled during the simulation
- ✓ Given N samples $\{x_1, x_2, \ldots, x_N\}$ the value of a **rate** metric is

$$y = \frac{\sum_i x_i}{\tau}$$

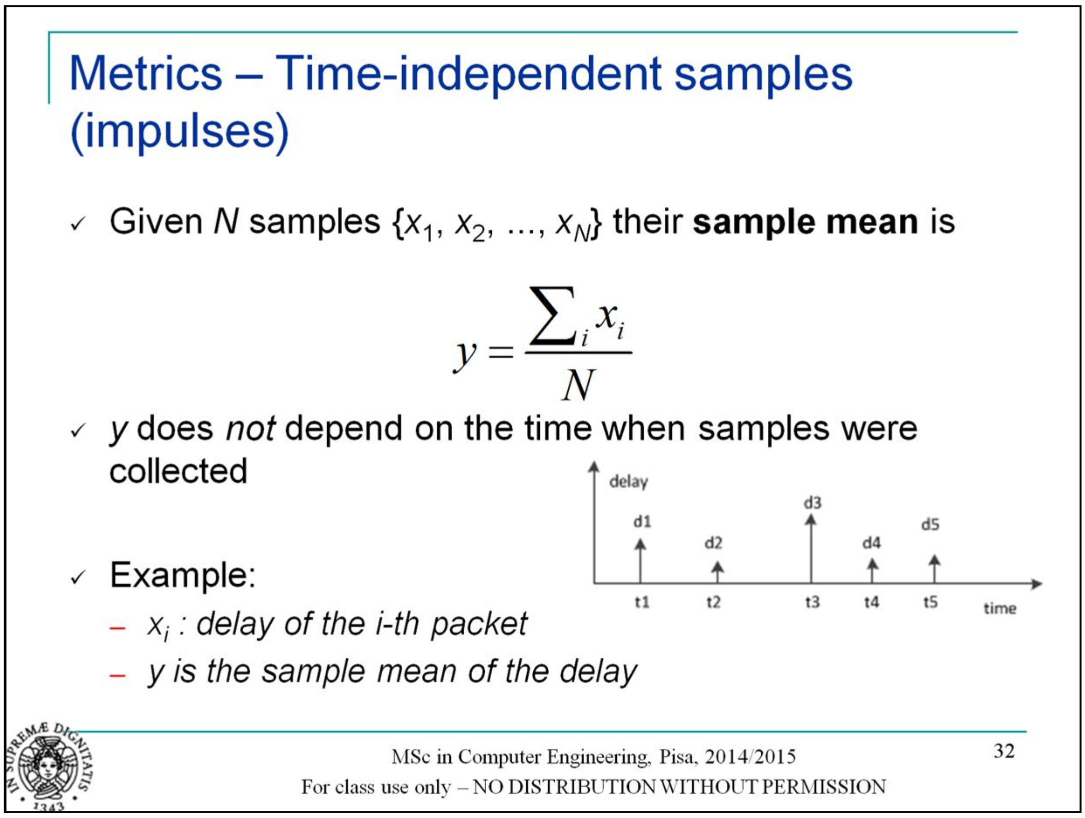
where τ is the time interval when the samples were collected

- ✓ Example: x_i is the size of the i-th packet (in bytes) served by a router, y is the throughput of the router, in bytes/s

MSc in Computer Engineering, Pisa, 2014/2015 31
For class use only – NO DISTRIBUTION WITHOUT PERMISSION

There are several types of metrics that are commonly extracted from a simulator. In most cases, they are **averages** of quantities measured during a simulation run.

There are, however, **several types of averages**, and different data structures involved in maintaining them.

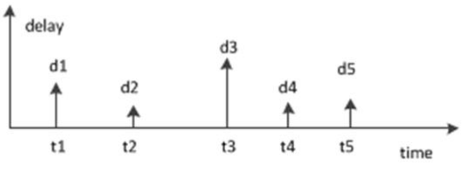


Metrics – Time-independent samples (impulses)

- ✓ Given N samples $\{x_1, x_2, \ldots, x_N\}$ their **sample mean** is

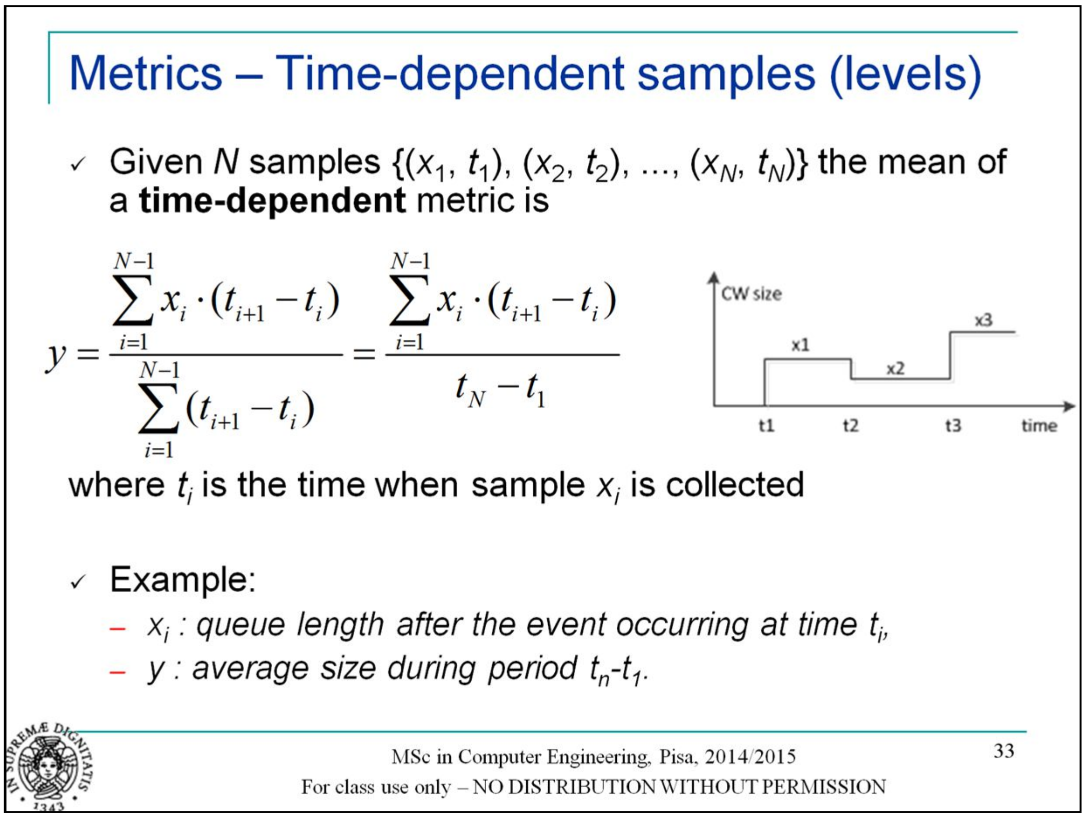
$$y = \frac{\sum_i x_i}{N}$$

- ✓ y does *not* depend on the time when samples were collected



- ✓ Example:
  - *x_i : delay of the i-th packet*
  - *y is the sample mean of the delay*

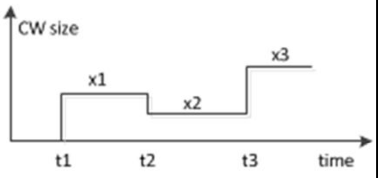
“impulses” are values that are meaningful at the moment they are generated, like delays.
If you change the time instant tj of observation j, then nothing happens to y.



Metrics – Time-dependent samples (levels)

- ✓ Given N samples $\{(x_1, t_1), (x_2, t_2), \ldots, (x_N, t_N)\}$ the mean of a **time-dependent** metric is

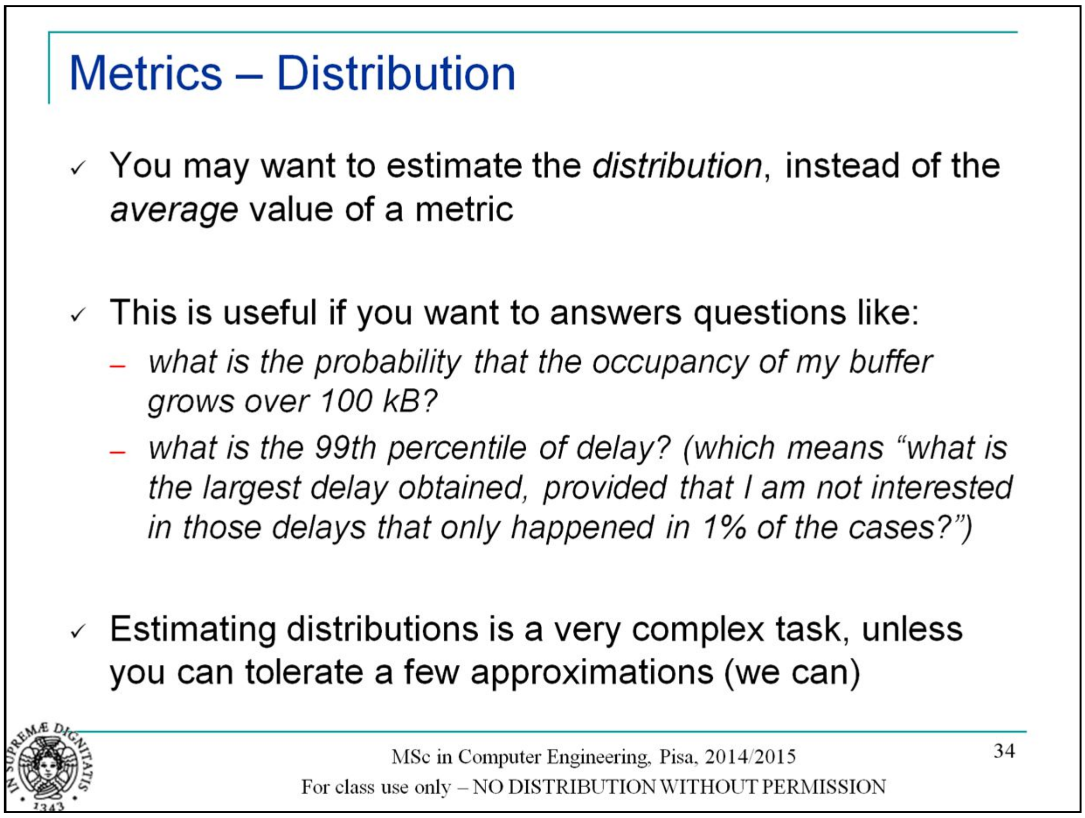
$$y = \frac{\sum_{i=1}^{N-1} x_i \cdot (t_{i+1} - t_i)}{\sum_{i=1}^{N-1} (t_{i+1} - t_i)} = \frac{\sum_{i=1}^{N-1} x_i \cdot (t_{i+1} - t_i)}{t_N - t_1}$$



where t_i is the time when sample x_i is collected

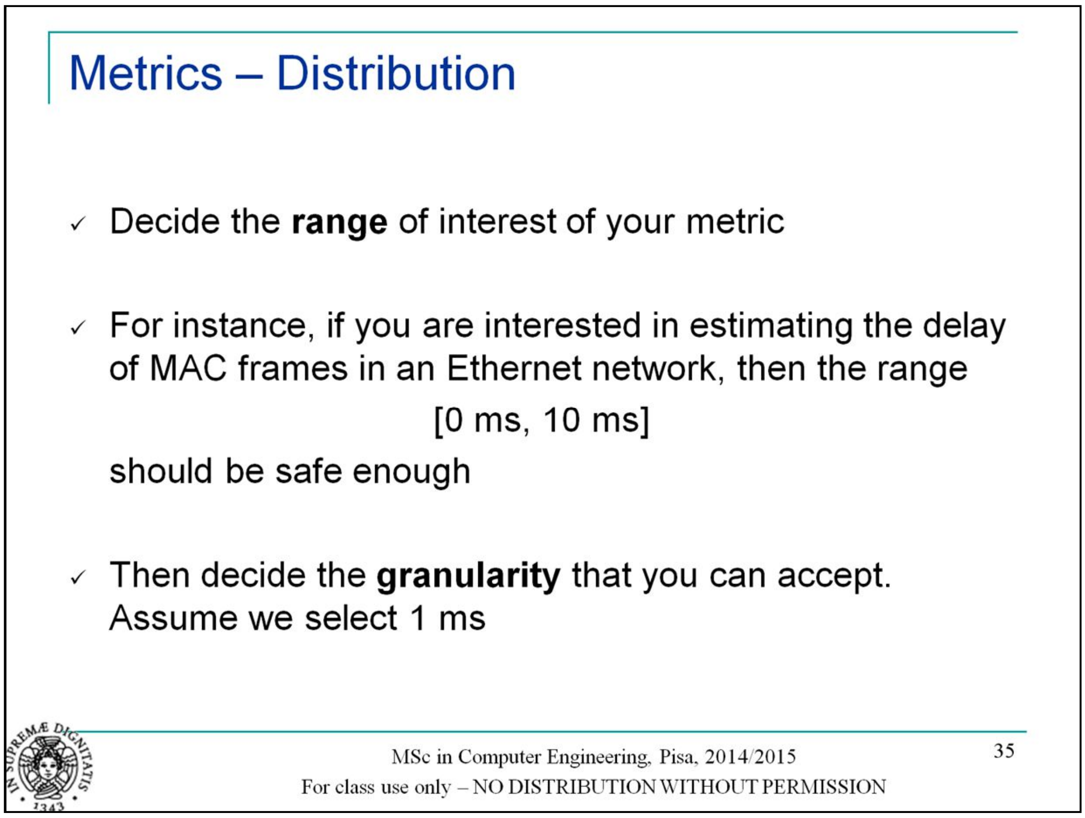
- ✓ Example:
  - *x_i : queue length after the event occurring at time t_i,*
  - *y : average size during period t_n-t_1.*

These are not impulses, they are **levels**: values that are meaningful **until they change**.



Metrics – Distribution

- ✓ You may want to estimate the *distribution*, instead of the *average* value of a metric
- ✓ This is useful if you want to answers questions like:
  - *what is the probability that the occupancy of my buffer grows over 100 kB?*
  - *what is the 99th percentile of delay? (which means "what is the largest delay obtained, provided that I am not interested in those delays that only happened in 1% of the cases?")*
- ✓ Estimating distributions is a very complex task, unless you can tolerate a few approximations (we can)

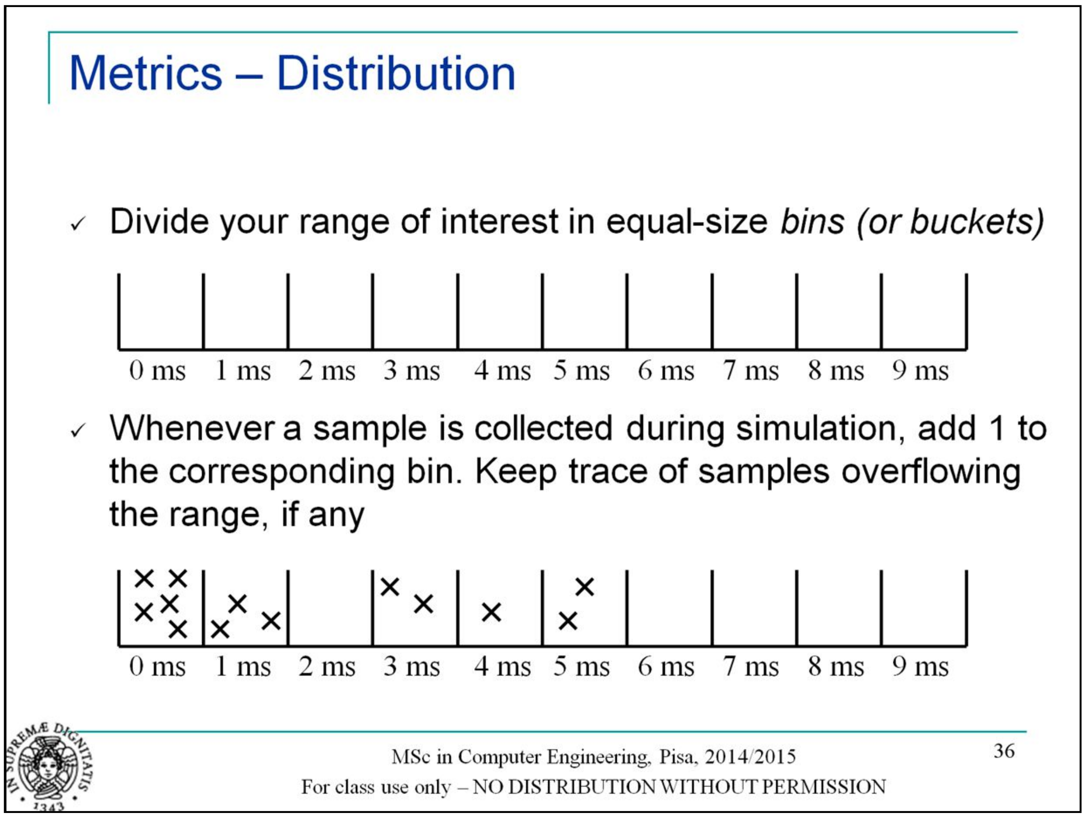


Metrics – Distribution

- ✓ Decide the **range** of interest of your metric
- ✓ For instance, if you are interested in estimating the delay of MAC frames in an Ethernet network, then the range

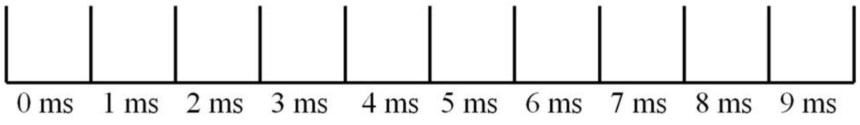
  [0 ms, 10 ms]

  should be safe enough
- ✓ Then decide the **granularity** that you can accept. Assume we select 1 ms

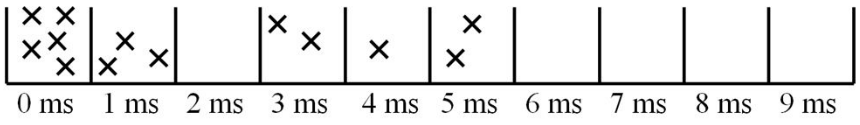


Metrics – Distribution

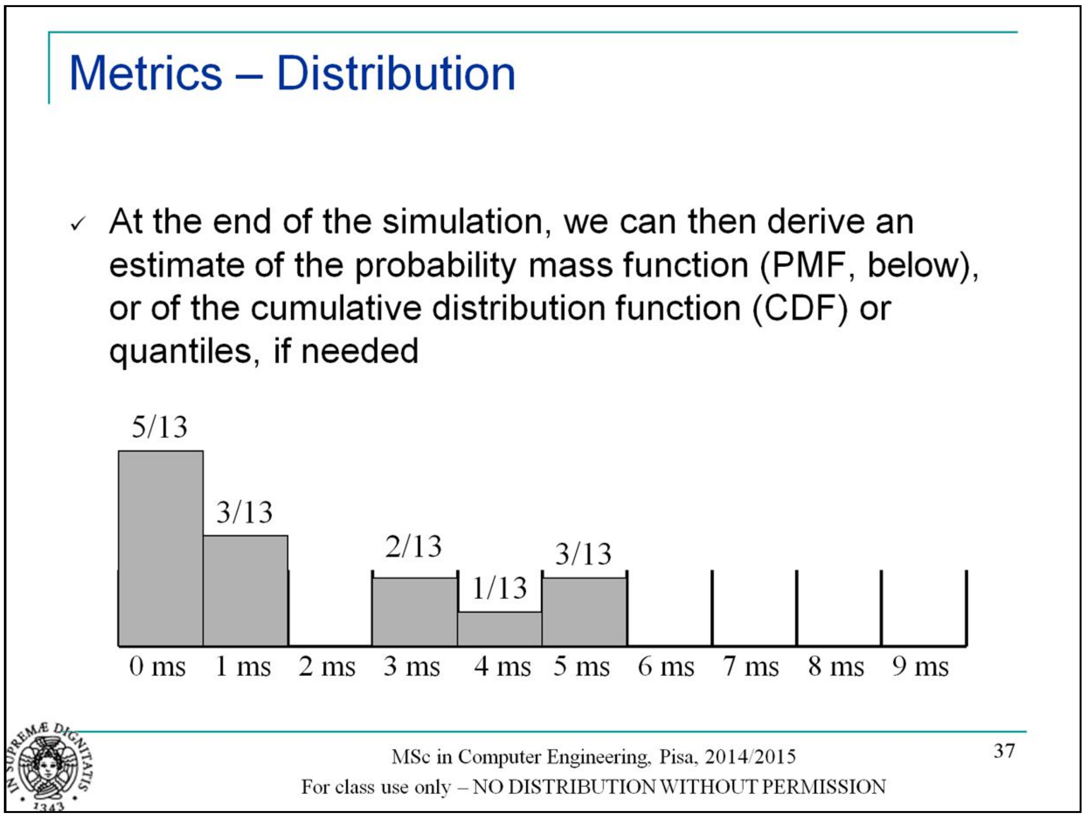
- ✓ Divide your range of interest in equal-size *bins (or buckets)*



- ✓ Whenever a sample is collected during simulation, add 1 to the corresponding bin. Keep trace of samples overflowing the range, if any

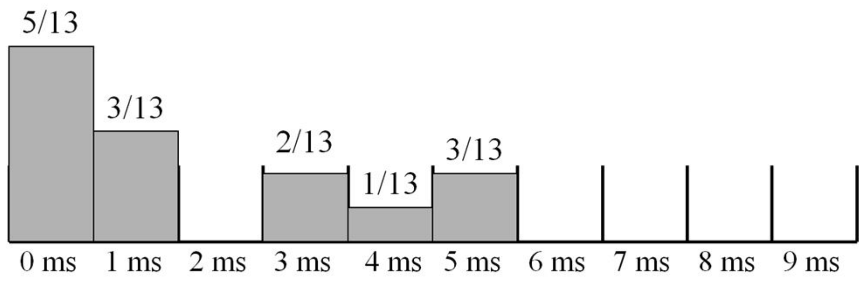


At the end of the simulation, you should **count** the number of observations that are in the "overflow" bin. If they are few, then your selection of extremes for the measurement interval are correct, otherwise you may want to select a larger interval and start again.



Metrics – Distribution

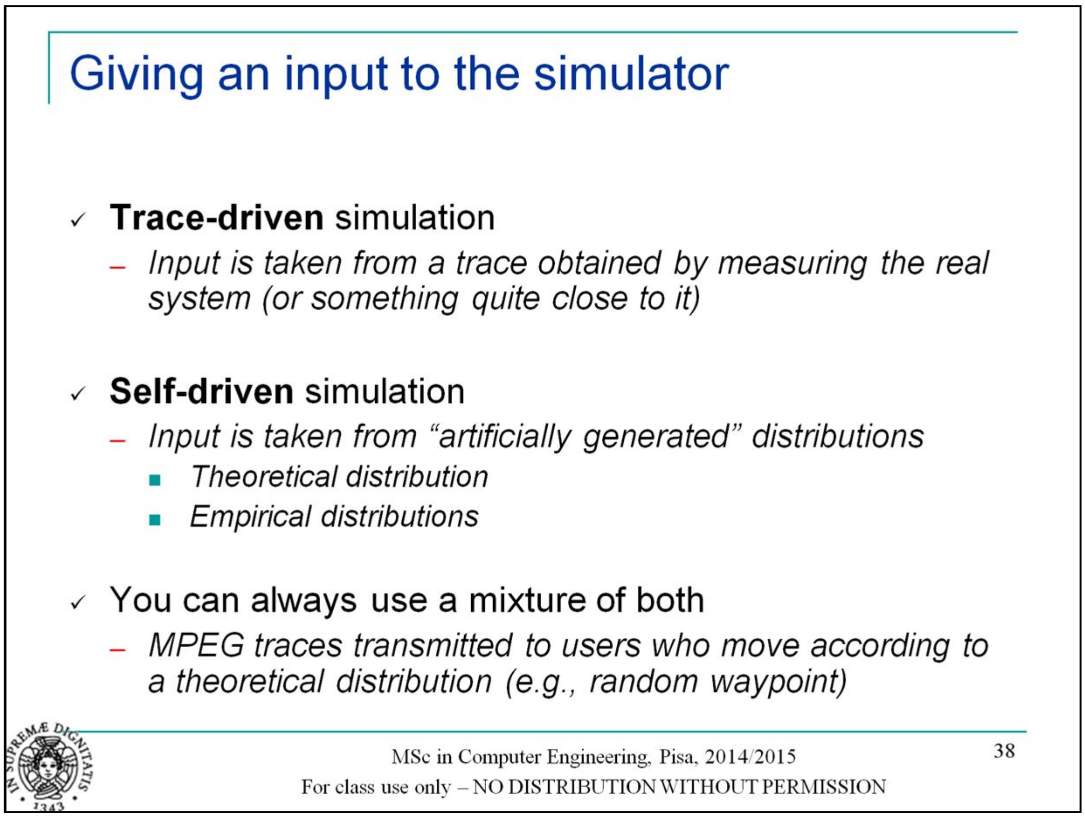
- ✓ At the end of the simulation, we can then derive an estimate of the probability mass function (PMF, below), or of the cumulative distribution function (CDF) or quantiles, if needed



Note that the **size of the bins** determines the way data will look.

-Too large: lack of accuracy and details (very different samples end up in the same bin)

-Too small: lack of resolution: you end up having mostly **empty** bins or bins with **one or few** samples within. This does not tell you anything about the real shape of the pmf/cdf



Giving an input to the simulator

- ✓ **Trace-driven** simulation
  - *Input is taken from a trace obtained by measuring the real system (or something quite close to it)*
- ✓ **Self-driven** simulation
  - *Input is taken from "artificially generated" distributions*
    - *Theoretical distribution*
    - *Empirical distributions*
- ✓ You can always use a mixture of both
  - *MPEG traces transmitted to users who move according to a theoretical distribution (e.g., random waypoint)*

MSc in Computer Engineering, Pisa, 2014/2015 38
For class use only – NO DISTRIBUTION WITHOUT PERMISSION

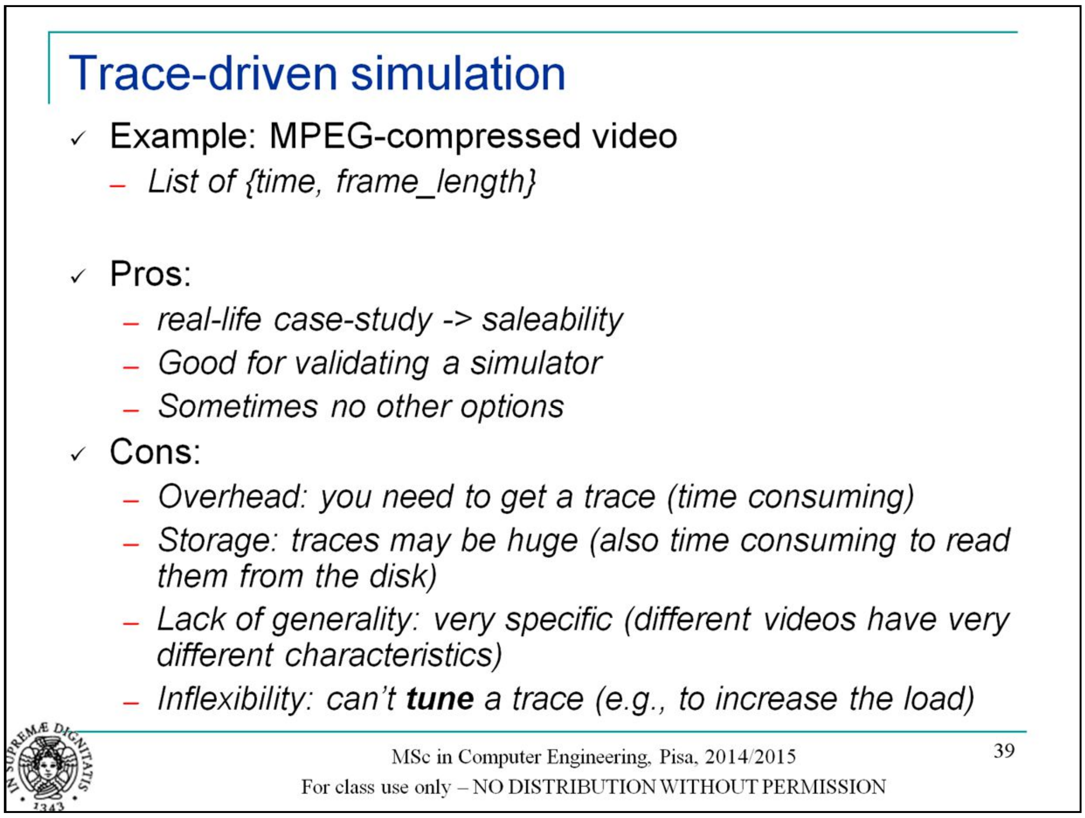
In our toy example (queue+server) input is **artificially generated** (synthetic input, self-driven simulation). In order to generate that input I need **random number generators** (recall that interarrival times have to be exponentially distributed).

There are two basic approaches to generate input to a simulator

a) **Trace-driven** approach: you obtain a trace of the input of the system over time, and you feed it to the simulator. In our case, this means to obtain a file listing (time, bytes) for every packet arrival. Typical case in networking: **compressed** videos, since they are difficult to model.

b) **Self-driven** approach: this is what we have assumed in our example. It consists in generating input through **probability distributions**. Our approach was to use a **theoretical** one, i.e. the exponential. This is because, of course, we have **measured some data and fitted it to an exponential model**, or we are knowledgeable enough to state that the interarrival times can be expected to be exponential.

There are cases when **you can't fit your data** to a known distribution (either because there isn't one, or you can't find it). In these cases, you may resort to **extracting a distribution from the PMF of your data**. This is called using **empirical distributions**. This should be done as a last resort.



Trace-driven simulation

- Example: MPEG-compressed video
  - *List of {time, frame_length}*
- Pros:
  - *real-life case-study -> saleability*
  - *Good for validating a simulator*
  - *Sometimes no other options*
- Cons:
  - *Overhead: you need to get a trace (time consuming)*
  - *Storage: traces may be huge (also time consuming to read them from the disk)*
  - *Lack of generality: very specific (different videos have very different characteristics)*
  - *Inflexibility: can't* ***tune*** *a trace (e.g., to increase the load)*

Example of a **trace-driven simulation**: if I want to test the performance of a scheduling algorithm when transmitting **telnet** packets, I can sit at my desk, open a telnet session somewhere, intercept the packets that are transmitted by my system and generate an **input trace** in a table form such as:

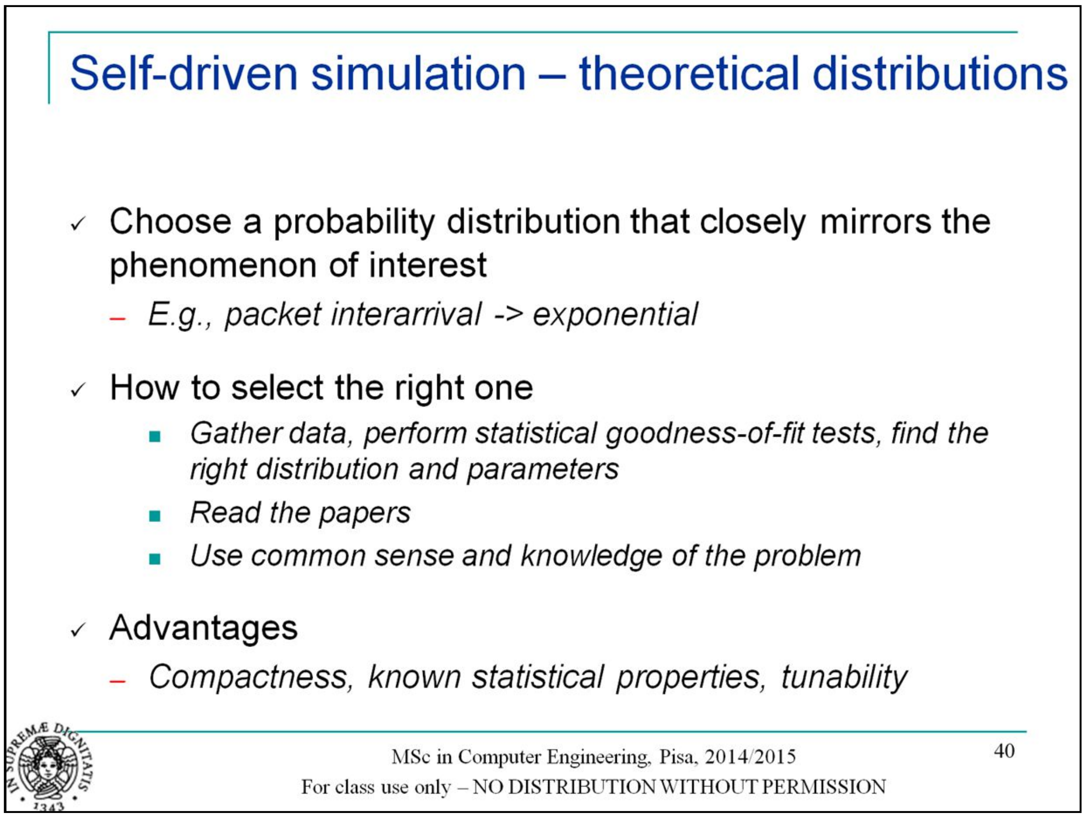
Time	**length**
0.01s	**150 bytes**
….	**….**

Afterwards, I can **use that trace file** to run my simulation. Whenever I need to generate a new "packet arrival" event, I can read the arrival time and packet length from the next line of the tracefile.

Real traces **normally give more reliable information** on the working conditions of your system. However, you need **experience and common sense** in order to be able to choose wisely.

A simulation study is often aimed at inferring **general properties** of a system, which are true under widely general assumptions. A trace-driven simulation says that your system performs (say) better than another **under those specific settings**, which may be hardly general. Especially with videos, an action movie is very different from a sport event or a talk show program. Nevertheless, it may be the only way to do anything meaningful with some kinds of traffic (notably: compressed video), since synthetic generators are not up to the job.

Traces cannot be tuned: if you have a trace of a lo-quality video, and you need a trace of the same video at a higher resolution, you **cannot scale your trace up**. You will need to encode it anew. The same occurs if you have a trace of n customers, and you need a trace for $2n$ customers: you cannot just halve the interarrival times, because it will not work.



Self-driven simulation – theoretical distributions

- ✓ Choose a probability distribution that closely mirrors the phenomenon of interest
  - *E.g., packet interarrival -> exponential*
- ✓ How to select the right one
  - *Gather data, perform statistical goodness-of-fit tests, find the right distribution and parameters*
  - *Read the papers*
  - *Use common sense and knowledge of the problem*
- ✓ Advantages
  - *Compactness, known statistical properties, tunability*

MSc in Computer Engineering, Pisa, 2014/2015 40
For class use only – NO DISTRIBUTION WITHOUT PERMISSION

The opposite w.r.t. the previous case:

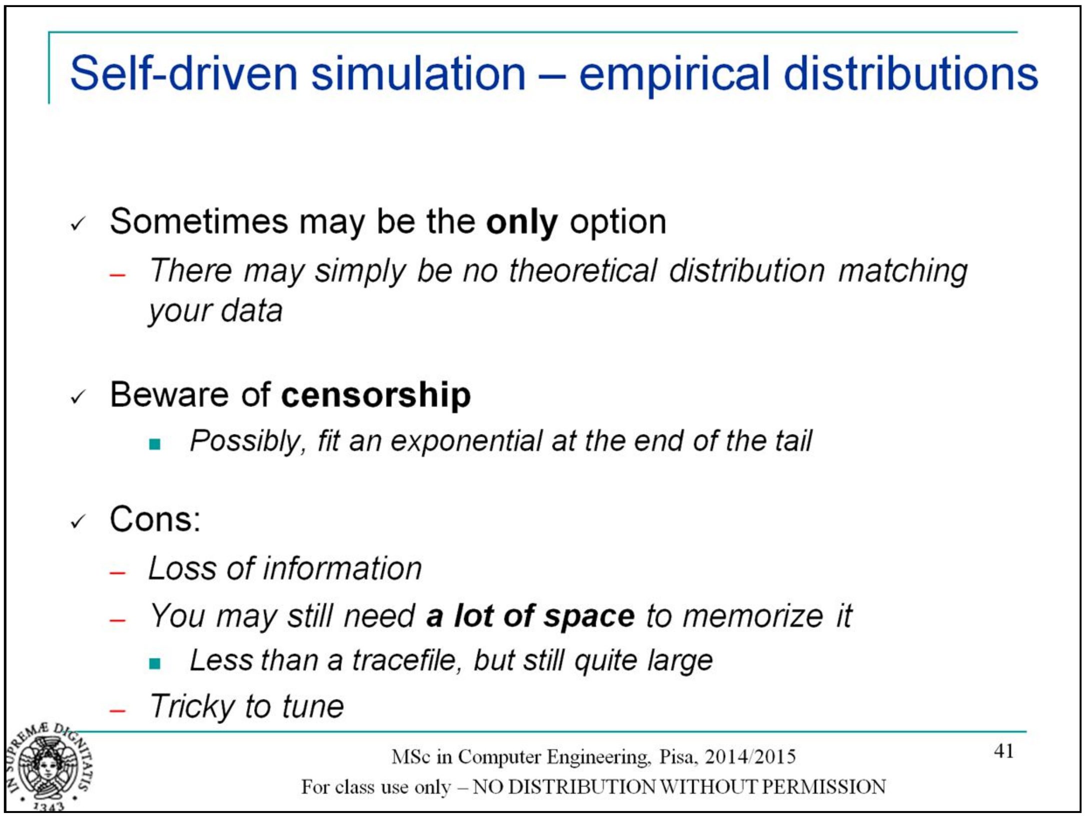
-I can **modify the distribution** by varying its parameters (e.g., ask myself what happens if packets arrive 10% faster). I can't do this using a tracefile: if I want to change the frame-rate of an MPEG movie (or its definition), I have to **re-encode** the entire movie, since scaling is not linear (not by a long shot)

-Artificial distributions require **little information** to be stored.

How do you choose the right one? Either you do the work yourself, or you **rely on the literature**: if the problem is real, it is highly likely that someone else has already dealt with it, and you can re-use her results (with a grain of salt, as usual).

The advantages of a theoretical distribution are several:

- Compactness (you can generate it with very few lines of code usually – more on this later)
- Known statistical properties (it helps you to validate your simulator)
- tunability



Self-driven simulation – empirical distributions

- ✓ Sometimes may be the **only** option
  - – *There may simply be no theoretical distribution matching your data*
- ✓ Beware of **censorship**
  - ▪ *Possibly, fit an exponential at the end of the tail*
- ✓ Cons:
  - – *Loss of information*
  - – *You may still need **a lot of space** to memorize it*
    - ▪ *Less than a tracefile, but still quite large*
  - – *Tricky to tune*

MSc in Computer Engineering, Pisa, 2014/2015 41
For class use only – NO DISTRIBUTION WITHOUT PERMISSION

Sometimes you are **forced** to use empirical distributions, simply because there is **no alternative**.

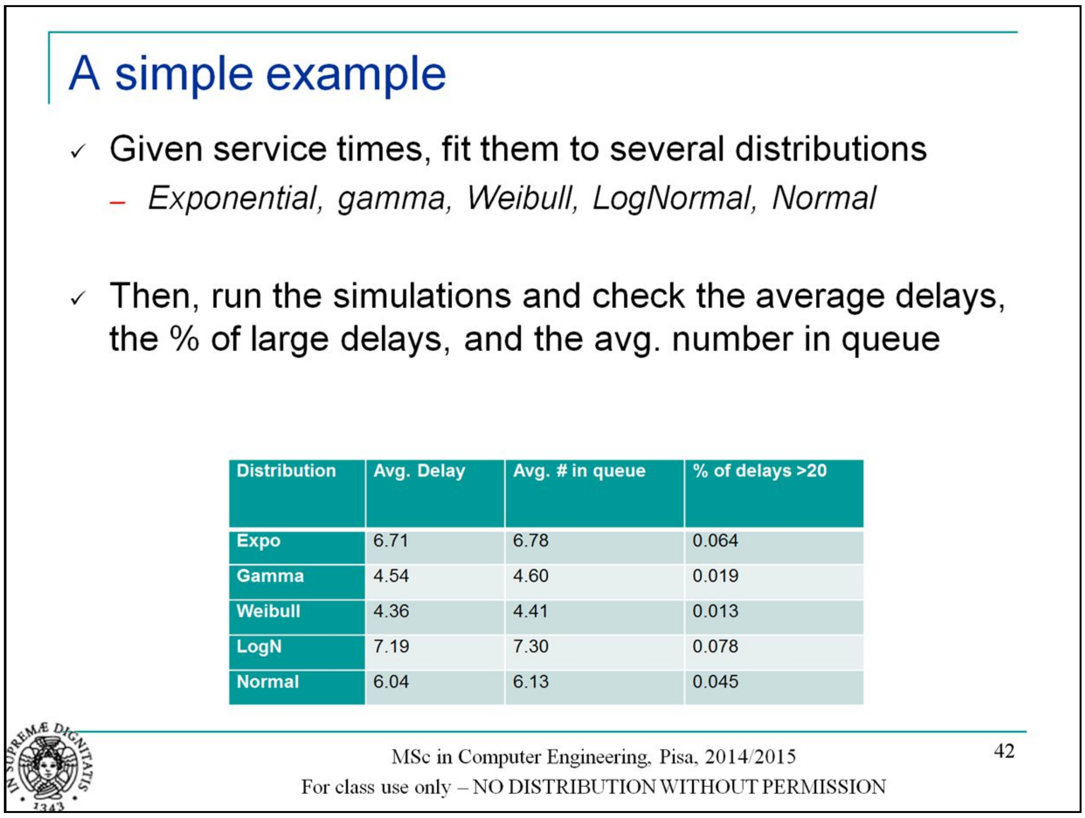
In this case, you must **be aware of censorship**. Your data will certainly be censored, and it is customary to add an **exponential tail** to the right to take into account the probability to obtain **larger values**.

In case, the approach is the following: you take a trace, and

a) you find an EPMF (first problem: how to choose the histogram buckets)
b) You compute the probabilities
c) You generate the same input with the same probabilities

If you do this, you may end up **losing a lot of information**. For instance, correlation between successive records (which is **very high** in MPEG videos, due to the GOP structure) would be lost, and your traffic would never look like an MPEG video.

Always be wary of empirical distributions, and use them if no other options are available.



A simple example

- ✓ Given service times, fit them to several distributions
  - *Exponential, gamma, Weibull, LogNormal, Normal*
- ✓ Then, run the simulations and check the average delays, the % of large delays, and the avg. number in queue

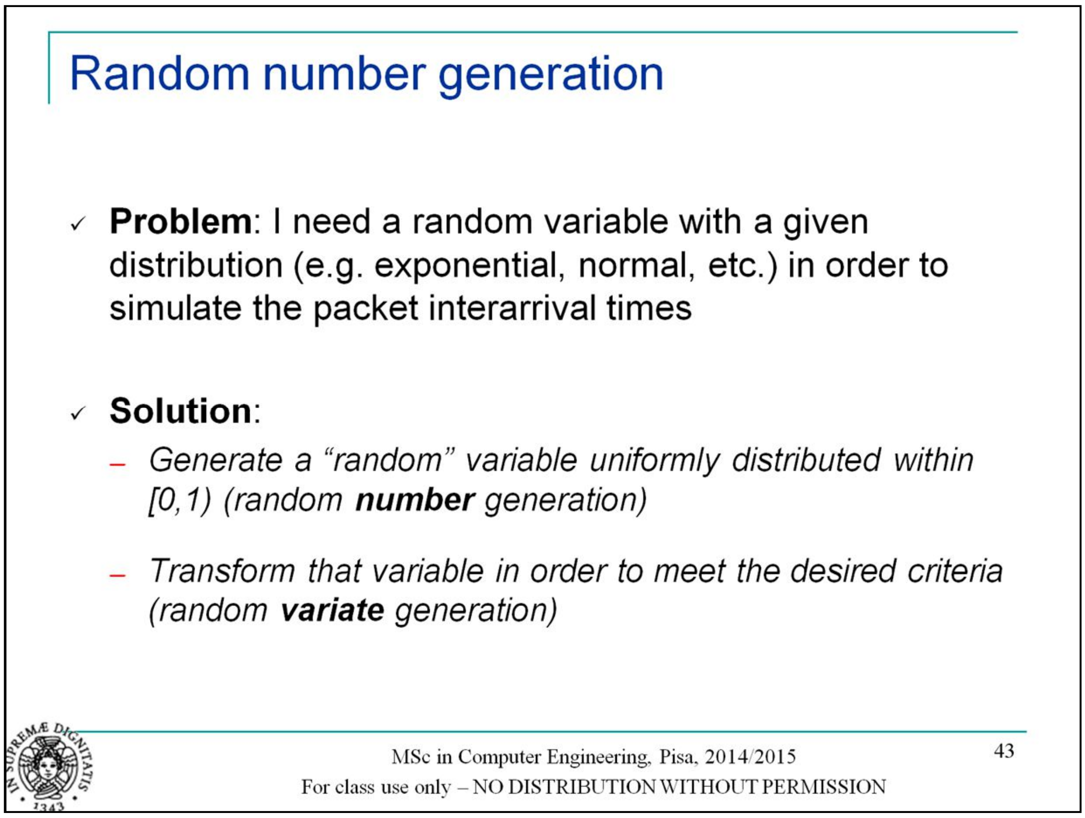
Distribution	Avg. Delay	Avg. # in queue	% of delays >20
Expo	6.71	6.78	0.064
Gamma	4.54	4.60	0.019
Weibull	4.36	4.41	0.013
LogN	7.19	7.30	0.078
Normal	6.04	6.13	0.045

MSc in Computer Engineering, Pisa, 2014/2015
For class use only – NO DISTRIBUTION WITHOUT PERMISSION

42

Fitting is **not to be taken lightly**. Your simulation results will depend **a lot** on the probability distributions that you choose for the input. Especially if they have a **fatter or slimmer tail**, things will change a lot.

Comment the example.



Random number generation

- ✓ **Problem**: I need a random variable with a given distribution (e.g. exponential, normal, etc.) in order to simulate the packet interarrival times
- ✓ **Solution**:
  - *Generate a "random" variable uniformly distributed within [0,1) (random **number** generation)*
  - *Transform that variable in order to meet the desired criteria (random **variate** generation)*

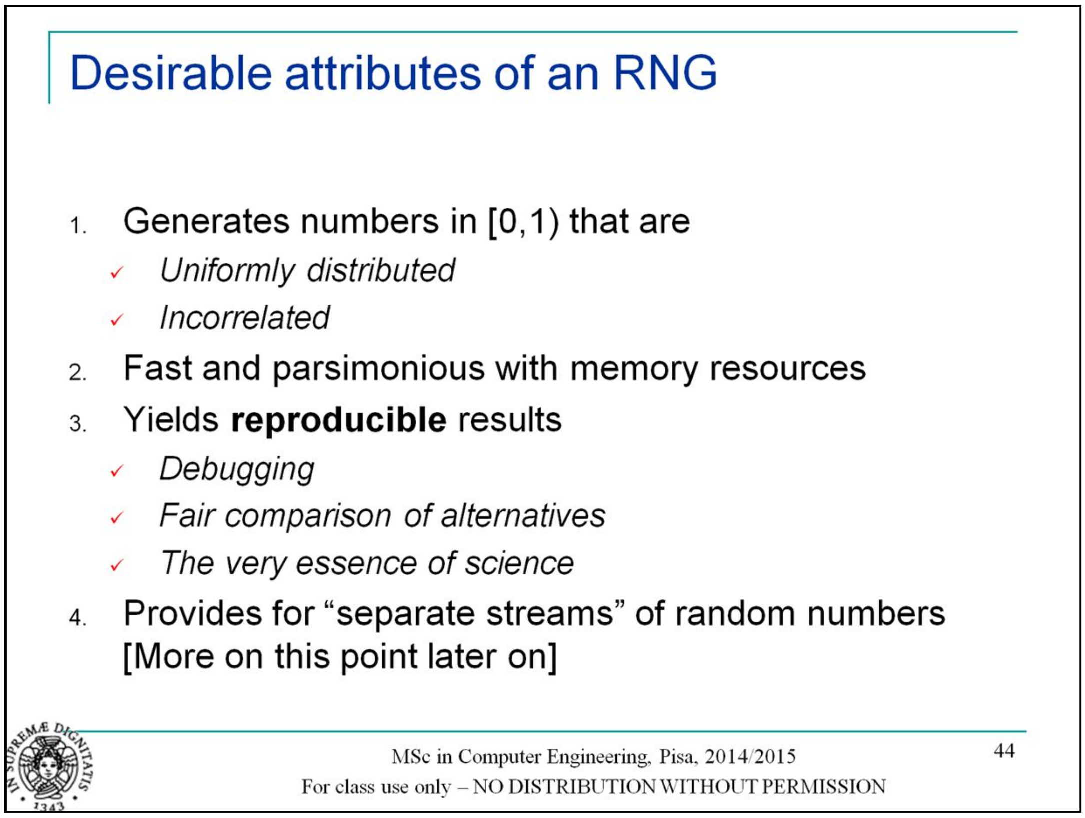
MSc in Computer Engineering, Pisa, 2014/2015 43
For class use only – NO DISTRIBUTION WITHOUT PERMISSION

In any case, a self-driven approach relies on **generating random numbers** according to a certain distribution.

Let us set something straight right from the start: **it is <u>impossible</u> to generate random numbers on a PC**. If someone tells you differently (or, worse yet, sells you some code that is supposed to actually generate random numbers), just doubt it.

In any case, you **don't need random numbers** as such. You need sequences of numbers that are:
-Distributed like the RV that you want to generate
-Highly **incorrelated**, so that the IID assumption is reasonable.

Such numbers can be generated using perfectly **determinisitic** algorithms, which will be shown hereafter.


Desirable attributes of an RNG

1. Generates numbers in [0,1) that are
   - ✓ *Uniformly distributed*
   - ✓ *Incorrelated*
2. Fast and parsimonious with memory resources
3. Yields **reproducible** results
   - ✓ *Debugging*
   - ✓ *Fair comparison of alternatives*
   - ✓ *The very essence of science*
4. Provides for "separate streams" of random numbers [More on this point later on]

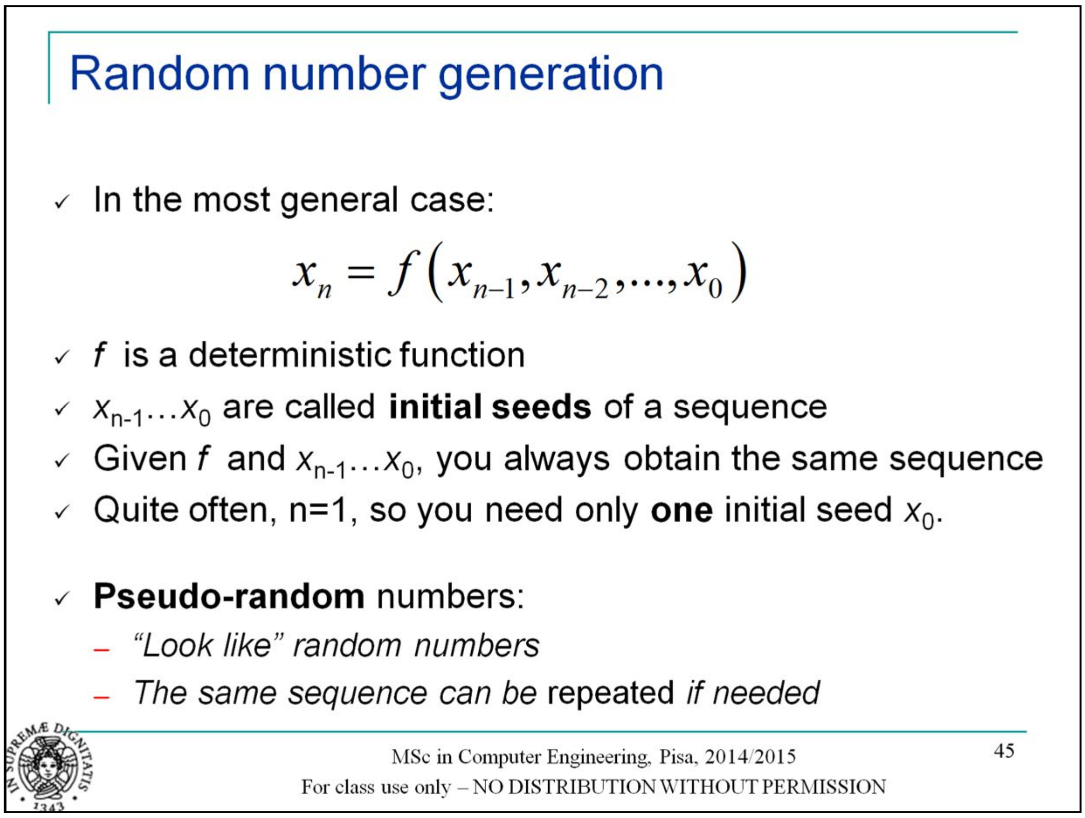
-Point 1 is fundamental. An RNG that lacks this characteristic **is never ever to be used**. The results that you get by using it are **totally unreliable**.

-Point 2 is obvious. A simulator has to be as efficient as possible, and the RNG should **not be the bottleneck.** Moreover, you will not need just **one,** but **many RNGs** (we'll see this later on).

-Point 3 is what perplexes most of the students. I don't need **random numbers**. I need **streams of numbers** that exhibit the **same statistical characteristics as random numbers**, but are **generated using deterministic algorithms:**

- You can't **debug** your code unless you are able to **reproduce** the exact scenario that led you to firing the bug. Thus, you need a deterministic path that leads you there.
- Simulation is often used to **compare alternative design choices**. A comparison might be **unfair** if the two alternative competitors weren't given exactly the same input. Again, you need deterministic input to do that
- The very definition of **science** involves **repeatable experiments**. Let's see if you obtain extremely good result from an experiment, but you can't say how you obtained it because you can't reproduce the same conditions. What is not repeatable is not science.

-Point 4 is somewhat delicate, and will be dealt with later on.



Random number generation

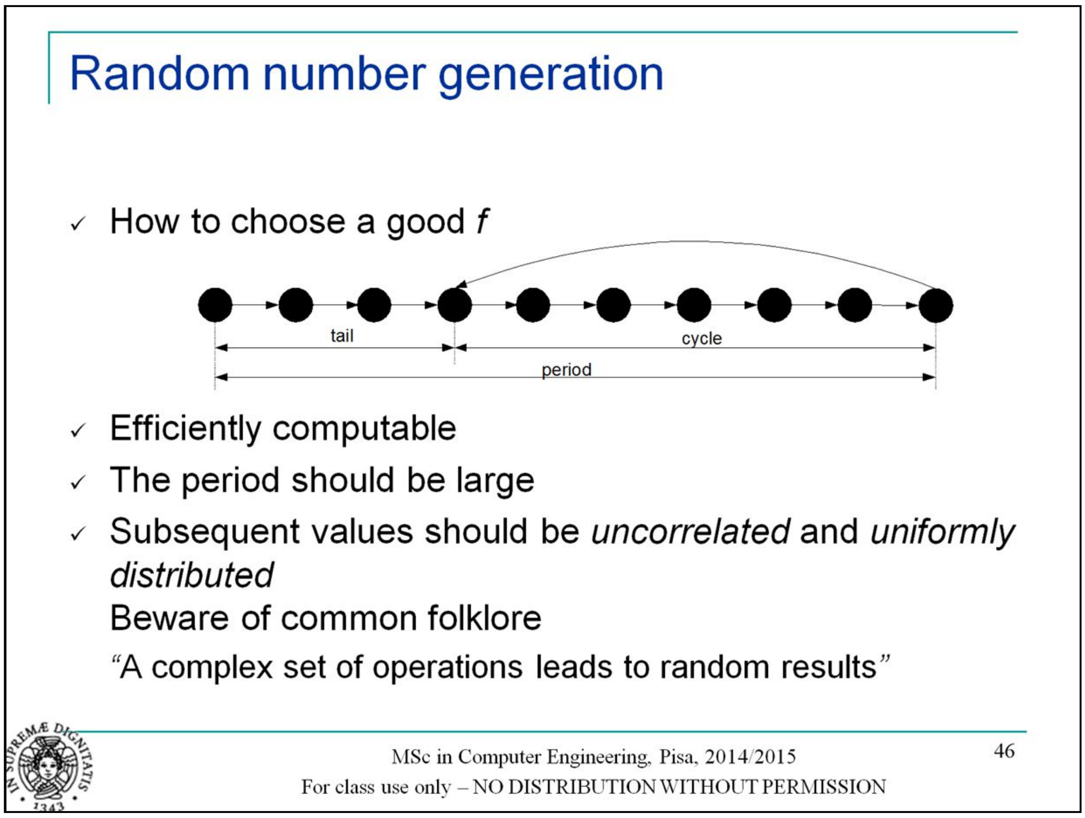
- ✓ In the most general case:

$$x_n = f\left(x_{n-1}, x_{n-2}, \ldots, x_0\right)$$

- ✓ *f* is a deterministic function
- ✓ $x_{n-1} \ldots x_0$ are called **initial seeds** of a sequence
- ✓ Given *f* and $x_{n-1} \ldots x_0$, you always obtain the same sequence
- ✓ Quite often, n=1, so you need only **one** initial seed x_0.
- ✓ **Pseudo-random** numbers:
  - *"Look like" random numbers*
  - *The same sequence can be* repeated *if needed*

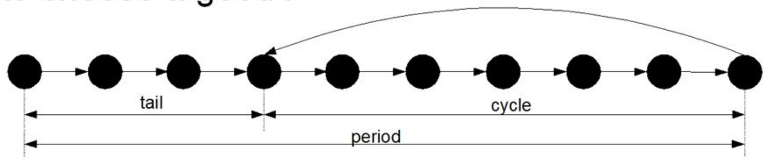
MSc in Computer Engineering, Pisa, 2014/2015 45
For class use only – NO DISTRIBUTION WITHOUT PERMISSION

You choose a function *f* **with memory**, meaning that each output is a function of the previous outputs. Most of the times, it is n=1, so you only need **one initial seed x0**.



Random number generation

- ✓ How to choose a good *f*



- ✓ Efficiently computable
- ✓ The period should be large
- ✓ Subsequent values should be *uncorrelated* and *uniformly distributed*
  Beware of common folklore
  "A complex set of operations leads to random results"

MSc in Computer Engineering, Pisa, 2014/2015
For class use only – NO DISTRIBUTION WITHOUT PERMISSION

46

A pseudo-random generator **ends up in a loop** sooner or later. At the very least, since it outputs results that are on *n* bits, it will produce at most 2^n numbers before hitting one that it has already produced.

The number of iterations before you hit an "old" number is called **period** of the generator. From the moment you follow the backwards arc, then **the values will not be independent anymore**, hence they should not be used. If you do a lag plot at a lag equal to the cycle, you find a straight line (perfect correlation).

The period depends:

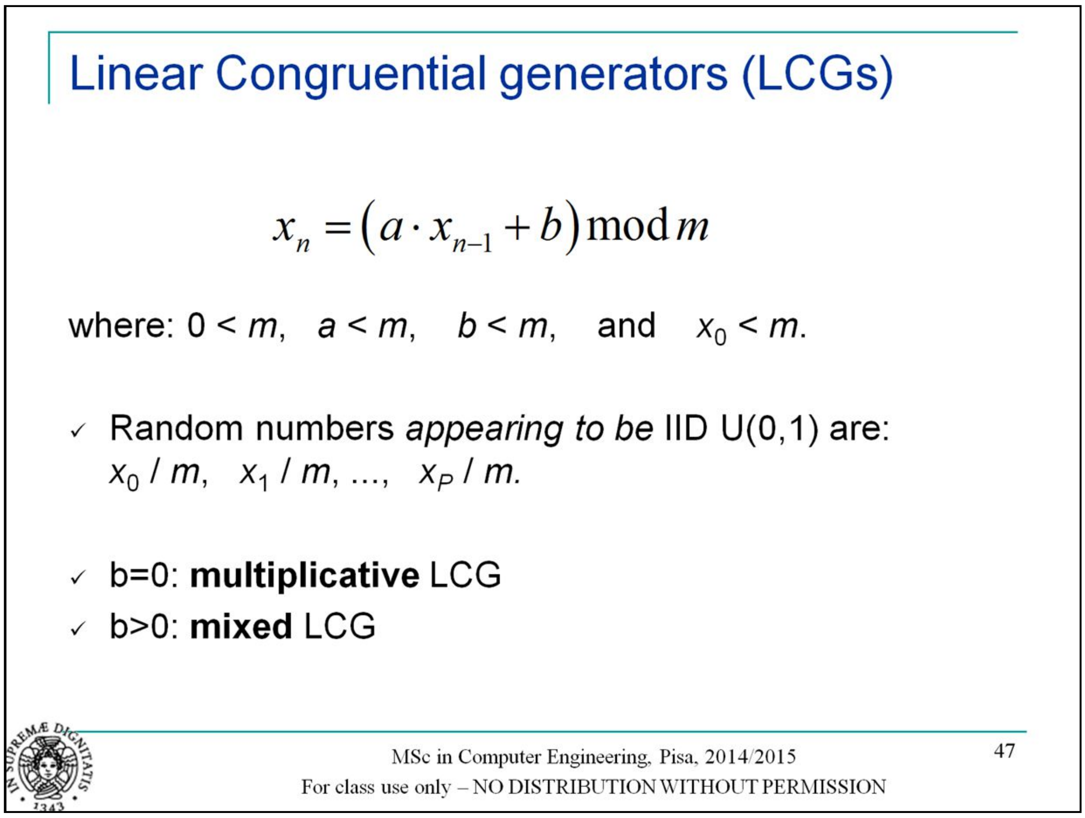
-On the shape of the function

-On the initial seed as well.

There are mathematically rigorous tests to prove that pseudo-random numbers are uncorrelated. These are the same tests that you may use to test the hypothesis that a set of values are extracted from a given distribution (you are actually doing the same thing in both cases, if you think about it).

Beware of common folklore. The statement in the last line is **totally absurd**. "random" does not mean "I can't figure out how you obtained this".

Most programming languages have libraries or instructions to generate random numbers. **Before using them, do read the code** (or the documentation at least), or **test them** using the conformity tests that RNGs are expected to pass (using test methods that we are going to introduce later on). The fact that they are in a library does not make them suitable for any scientifically sound purpose.



Linear Congruential generators (LCGs)

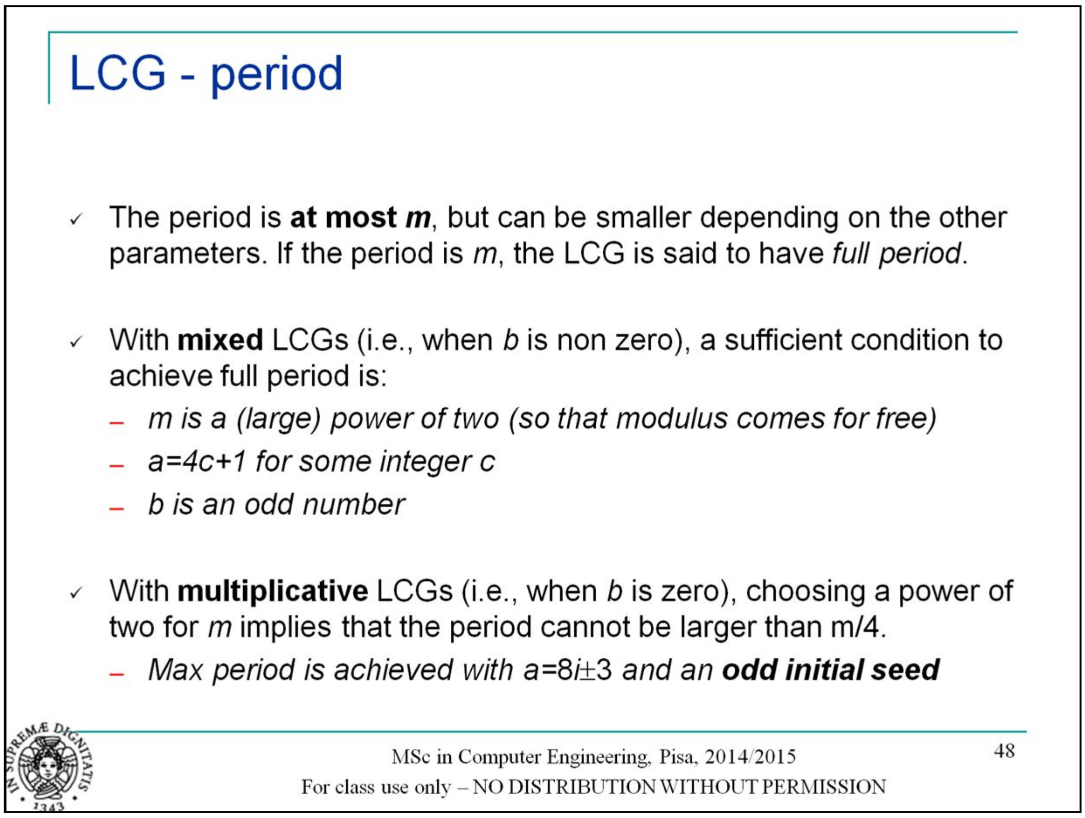
$$x_n = \left(a \cdot x_{n-1} + b\right) \bmod m$$

where: $0 < m$, $a < m$, $b < m$, and $x_0 < m$.

- ✓ Random numbers *appearing to be* IID U(0,1) are: x_0 / m, x_1 / m, ..., x_P / m.
- ✓ b=0: **multiplicative** LCG
- ✓ b>0: **mixed** LCG

MSc in Computer Engineering, Pisa, 2014/2015
For class use only – NO DISTRIBUTION WITHOUT PERMISSION
47

If you want numbers distributed in [0,1) rather than in [0,m), then you normalize by dividing by *m*. Note that the right extreme (i.e., 1) cannot occur as an output.

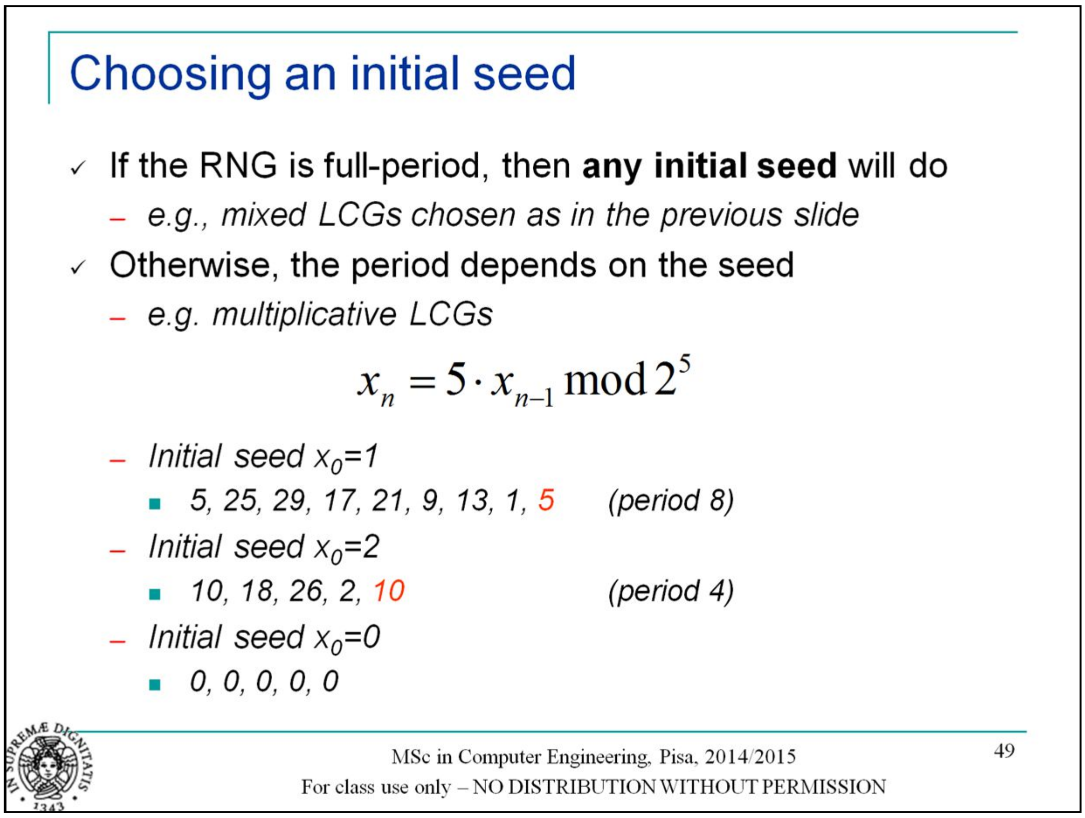


LCG - period

- ✓ The period is **at most *m***, but can be smaller depending on the other parameters. If the period is *m*, the LCG is said to have *full period*.
- ✓ With **mixed** LCGs (i.e., when *b* is non zero), a sufficient condition to achieve full period is:
  - *m is a (large) power of two (so that modulus comes for free)*
  - *a=4c+1 for some integer c*
  - *b is an odd number*
- ✓ With **multiplicative** LCGs (i.e., when *b* is zero), choosing a power of two for *m* implies that the period cannot be larger than m/4.
  - *Max period is achieved with a=8i±3 and an **odd initial seed***

Why a **large** power of two? Because the period cannot be larger than m, of course.

The necessary and sufficient conditions for achieving full period with mixed LCGs are that:
-m and b should be relatively prime
-All prime factors of m are factors of a-1 as well
-if m is a multiple of 4, then a-1 is a multiple of 4 as well.



Choosing an initial seed

- ✓ If the RNG is full-period, then **any initial seed** will do
  - *e.g., mixed LCGs chosen as in the previous slide*
- ✓ Otherwise, the period depends on the seed
  - *e.g. multiplicative LCGs*

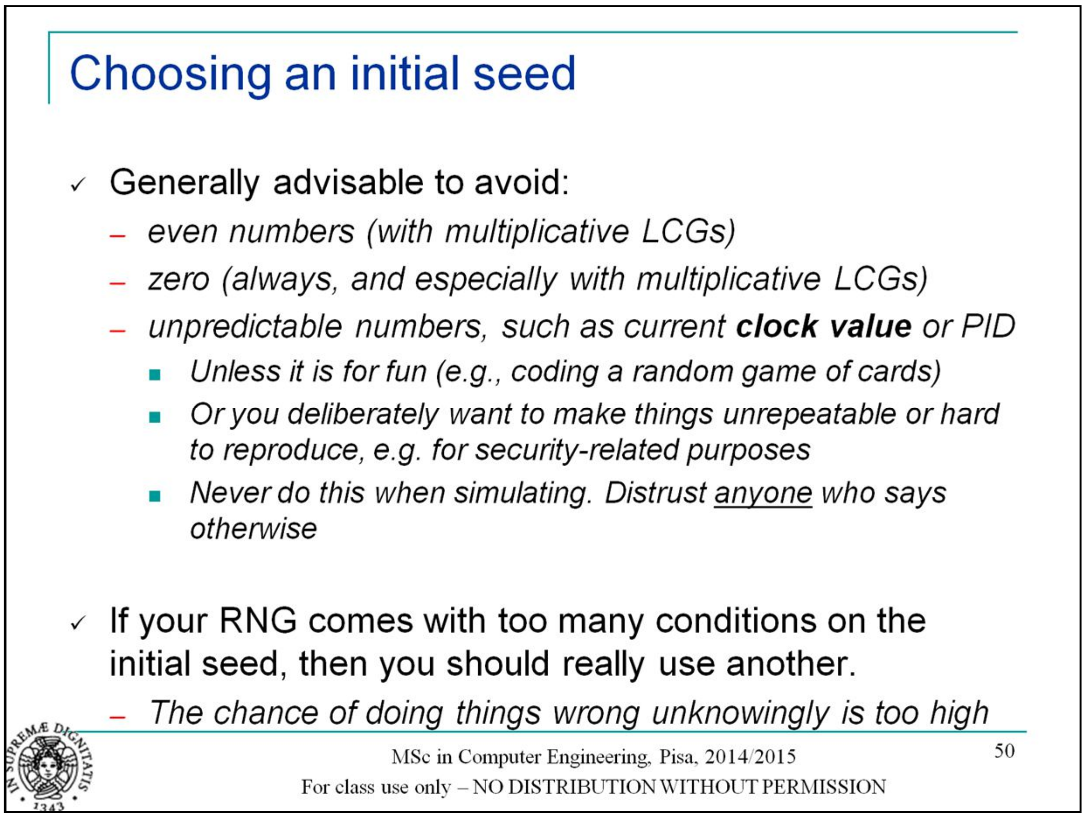
$$x_n = 5 \cdot x_{n-1} \bmod 2^5$$

  - *Initial seed x_0=1*
    - *5, 25, 29, 17, 21, 9, 13, 1, 5 (period 8)*
  - *Initial seed x_0=2*
    - *10, 18, 26, 2, 10 (period 4)*
  - *Initial seed x_0=0*
    - *0, 0, 0, 0, 0*

This multiplicative LCG has a maximum period equal to 8 (in fact, a=8*1-3). That period can be achieved only with an odd seed (e.g., 1), and not with even seeds (e.g., 2).

Few people stop to think that a multiplicative LCG will always yield a sequence of zeroes if seeded with zero.

If you are using a random number generator whose period depends on the initial seed, this should be clearly stated by those who coded it. In this case, they should provide either rules or a list of **good seeds** for that RNG.


Choosing an initial seed

- ✓ Generally advisable to avoid:
  - *even numbers (with multiplicative LCGs)*
  - *zero (always, and especially with multiplicative LCGs)*
  - *unpredictable numbers, such as current **clock value** or PID*
    - *Unless it is for fun (e.g., coding a random game of cards)*
    - *Or you deliberately want to make things unrepeatable or hard to reproduce, e.g. for security-related purposes*
    - *Never do this when simulating. Distrust anyone who says otherwise*
- ✓ If your RNG comes with too many conditions on the initial seed, then you should really use another.
  - *The chance of doing things wrong unknowingly is too high*

A common misconception (even among learned scholars) is that you should use a **“random”** initial seed, such as the output of the **time()** function. This is **dead wrong**.

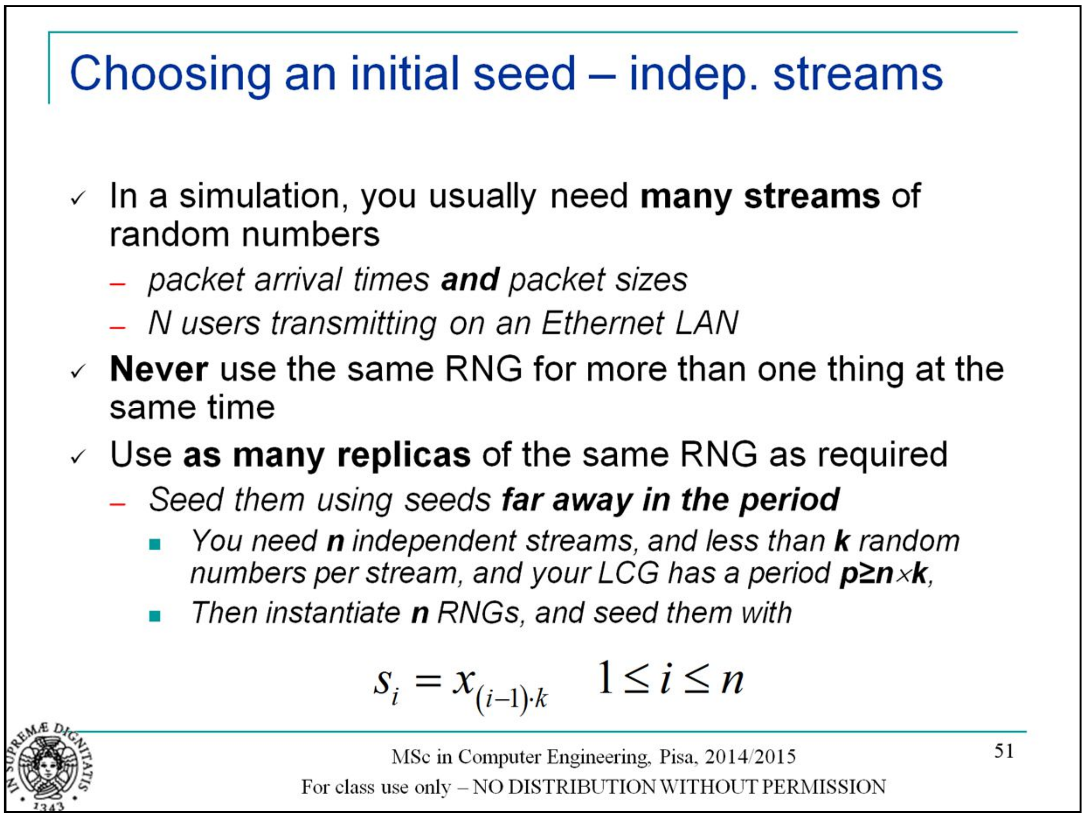
•Assume that you spot an odd behavior in your simulator, hence you suspect a bug. You will never be able to recreate the same conditions again

•You will never be able to compare two systems in exactly the same conditions

•You will never be able to repeat the result again, if someone asks you.

You are **not doing science**.

Note that you **might** want to have unpredictable (pseudo)-random sequences, e.g. when you need them for **cryptography,** other security-related matters, or simply **games** involving randomness. In these cases, you would do well to seed the RNG using the current clock value, since this makes the sequence **unrepeatable**. In our cases, you should **never** do that, because you want experiments to be repeatable.



Choosing an initial seed – indep. streams

- ✓ In a simulation, you usually need **many streams** of random numbers
  - *packet arrival times* ***and*** *packet sizes*
  - *N users transmitting on an Ethernet LAN*
- ✓ **Never** use the same RNG for more than one thing at the same time
- ✓ Use **as many replicas** of the same RNG as required
  - *Seed them using seeds* ***far away in the period***
    - *You need* ***n*** *independent streams, and less than* ***k*** *random numbers per stream, and your LCG has a period* $\mathbf{p \geq n \times k}$,
    - *Then instantiate* ***n*** *RNGs, and seed them with*

$$s_i = x_{(i-1)\cdot k} \quad 1 \leq i \leq n$$

MSc in Computer Engineering, Pisa, 2014/2015 51

For class use only – NO DISTRIBUTION WITHOUT PERMISSION

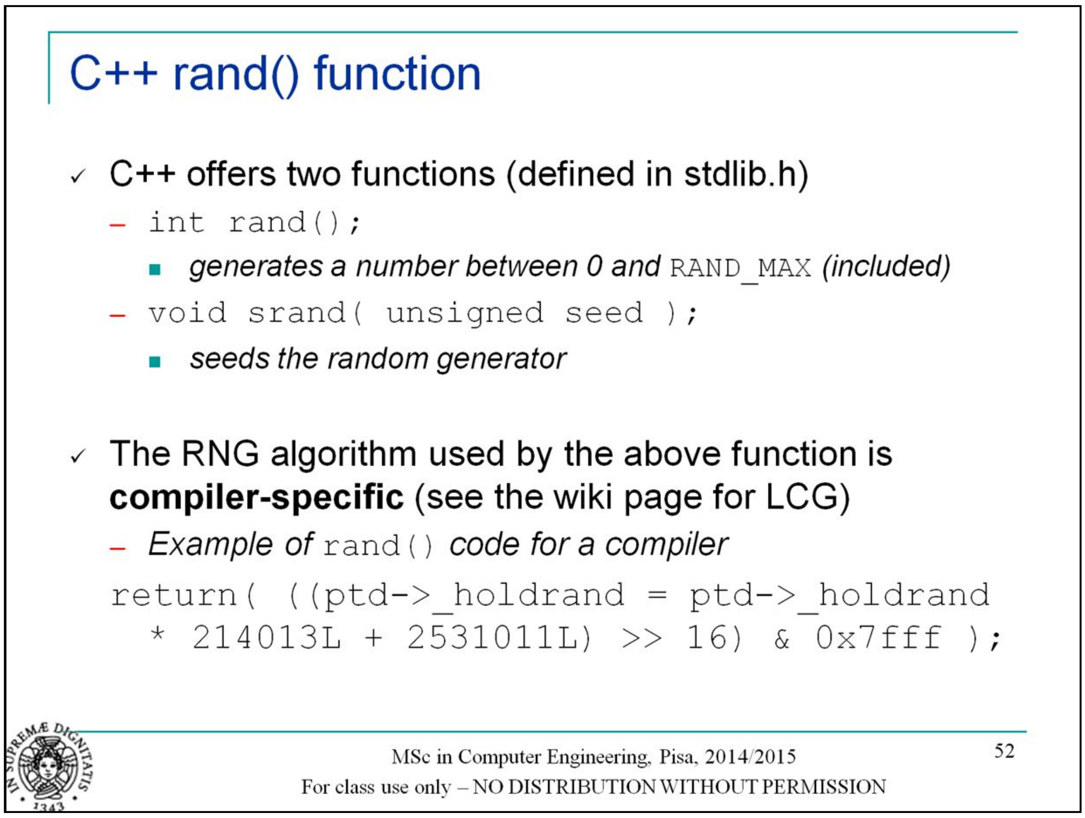
In many cases, you don't need **just one stream of pseudo-random numbers**, but you need **many** (suppose you have to simulate a cellular network with *k* users transmitting traffic). These should, of course, be **independent**. For this reason,

<u>You can't use a single RNG</u> to extract all the values for the *k* streams. In fact, the tests for the RNG may not be able to guarantee that "arbitrary samplings" of your RNG yield streams of uncorrelated numbers. Even if xi and x(i+1) are uncorrelated, x(i) and x(i+k) (with a large k) may be correlated, or they may not be uniform. If you are using the same RNG to produce k streams, you may end up believing that the single substreams have statistical properties that they do not possess in fact.

You should instead instantiate ***n*** **replicas of the same RNG,** and use **different seeds** for each replica. The seeds should be chosen based on the period *p*. Each seed should be a number which is k elements away, assuming that the simulation requires each RNG to generate at most k numbers.

Of course, using the **same seed** for all generators (a common mistake) is a very dumb thing to do.

More on this later on, again.



C++ rand() function

- C++ offers two functions (defined in stdlib.h)
  - `int rand();`
    - *generates a number between 0 and* `RAND_MAX` *(included)*
  - `void srand( unsigned seed );`
    - *seeds the random generator*
- The RNG algorithm used by the above function is **compiler-specific** (see the wiki page for LCG)
  - *Example of* `rand()` *code for a compiler*

```
return( ((ptd->_holdrand = ptd->_holdrand
  * 214013L + 2531011L) >> 16) & 0x7fff );
```

MSc in Computer Engineering, Pisa, 2014/2015
For class use only – NO DISTRIBUTION WITHOUT PERMISSION

52

C++ has a random generator, which is called using the **rand()** function.

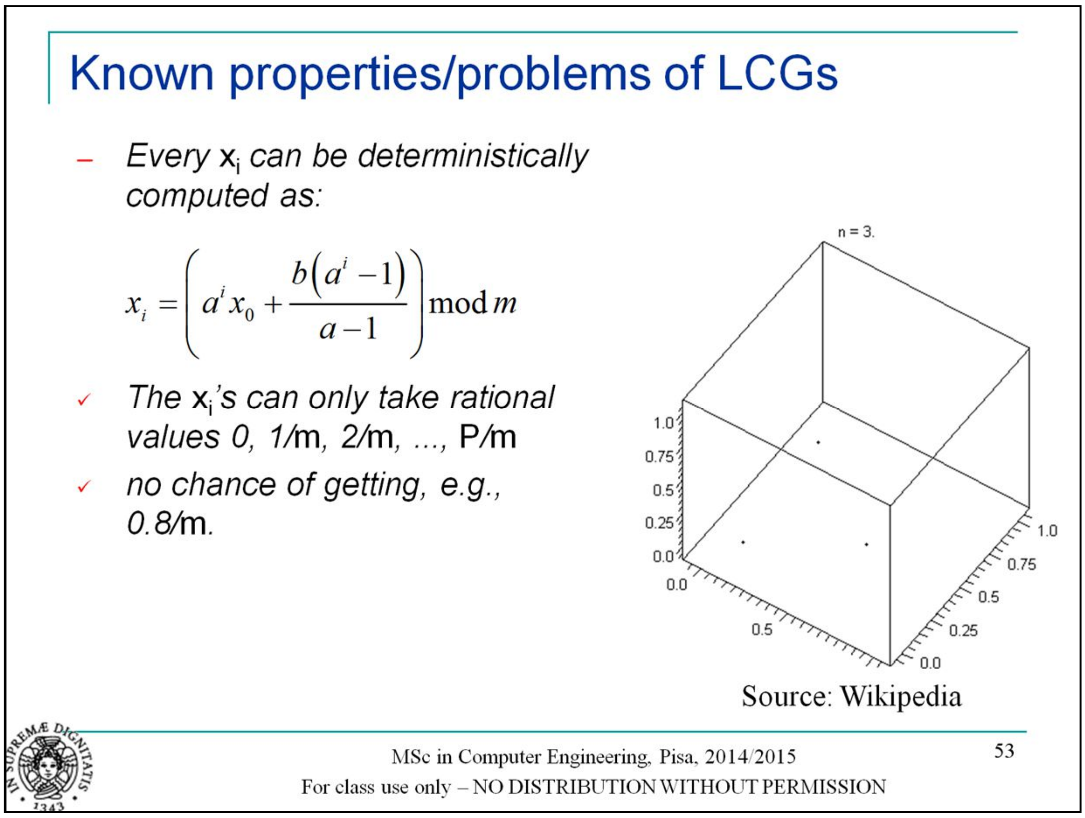
There is **no specification of an RNG algorithm**, so you had better assume that:

-The same code will produce different streams if compiled on two machines, even with the same seed, thus being highly **non-portable**

-There is no guarantee that the stream of numbers produced by these generator will pass any **statistical test.** In particular, the one in the slide is not even an LCG, so you don't really know its statistical properties.

-You can't have different, **independent streams** of pseudo-random numbers.

Bottom line: **never** use this in simulations.

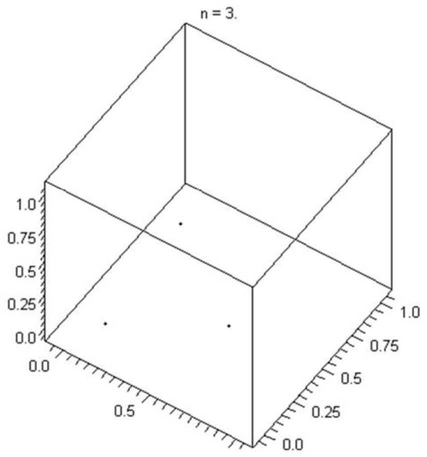


Known properties/problems of LCGs

- Every x_i can be deterministically computed as:

$$x_i = \left(a^i x_0 + \frac{b\left(a^i - 1\right)}{a-1} \right) \bmod m$$

  - ✓ The x_i's can only take rational values 0, 1/m, 2/m, ..., P/m
  - ✓ no chance of getting, e.g., 0.8/m.



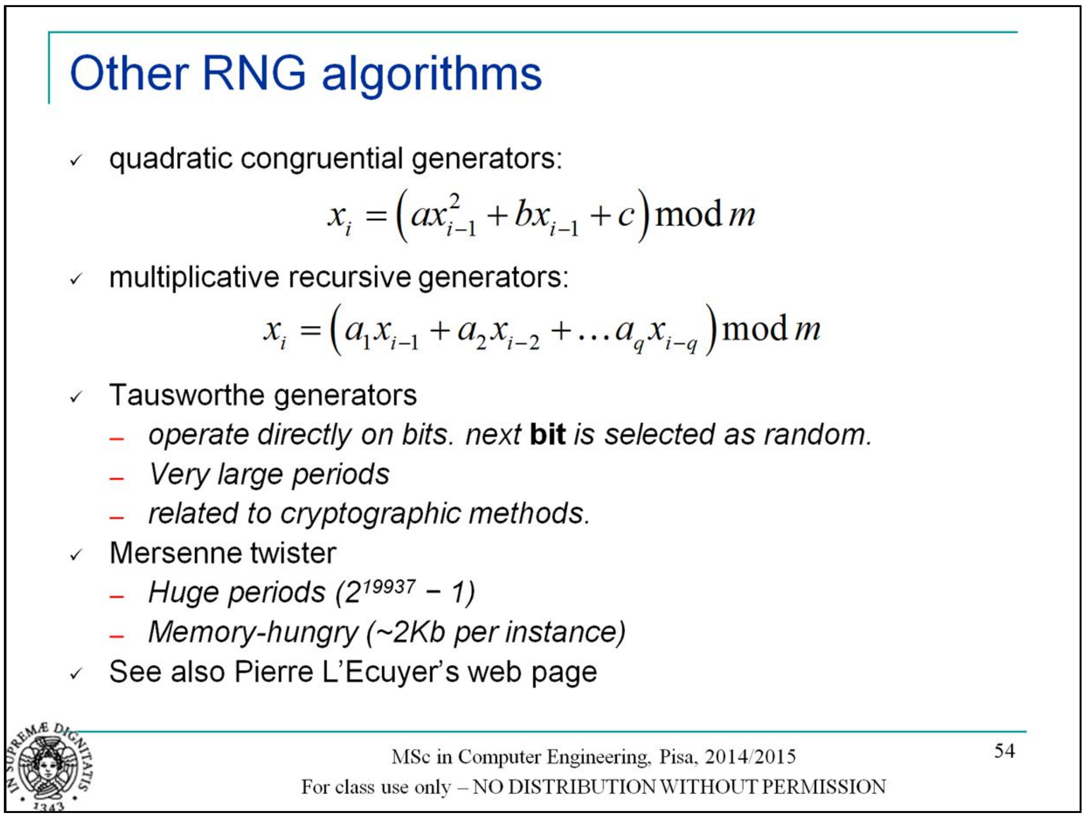
Source: Wikipedia

This works for both multiplicative and mixed LCGs.

Note that the fact that you can compute xi deterministically can be put to good use. You can figure out **values far away** and use them to seed multiple instances of the generator as in the previous slide, **provided that** you are able to make the above computation **exactly** (it may not be easy if *a, i* and *b* are large).

If you analyze tuples of subsequent numbers from the same LCG, e.g., by plotting vector [X(i), X(i+1), X(i+2)] in a 3D space, you observe that points tend to align on a **finite number of hyperplanes**. Depending on the LCG parameters, the number of hyperplans can be **very small**, even with full-period mixed LCGs. This means that there are **regions of n-dimensional space** that are never explored by this LCG.

In other words, there are **many sequences** that will never come out. This is bad, because you are missing out on a large portion of space.



Other RNG algorithms

- ✓ quadratic congruential generators:

$$x_i = \left(ax_{i-1}^2 + bx_{i-1} + c\right) \bmod m$$

- ✓ multiplicative recursive generators:

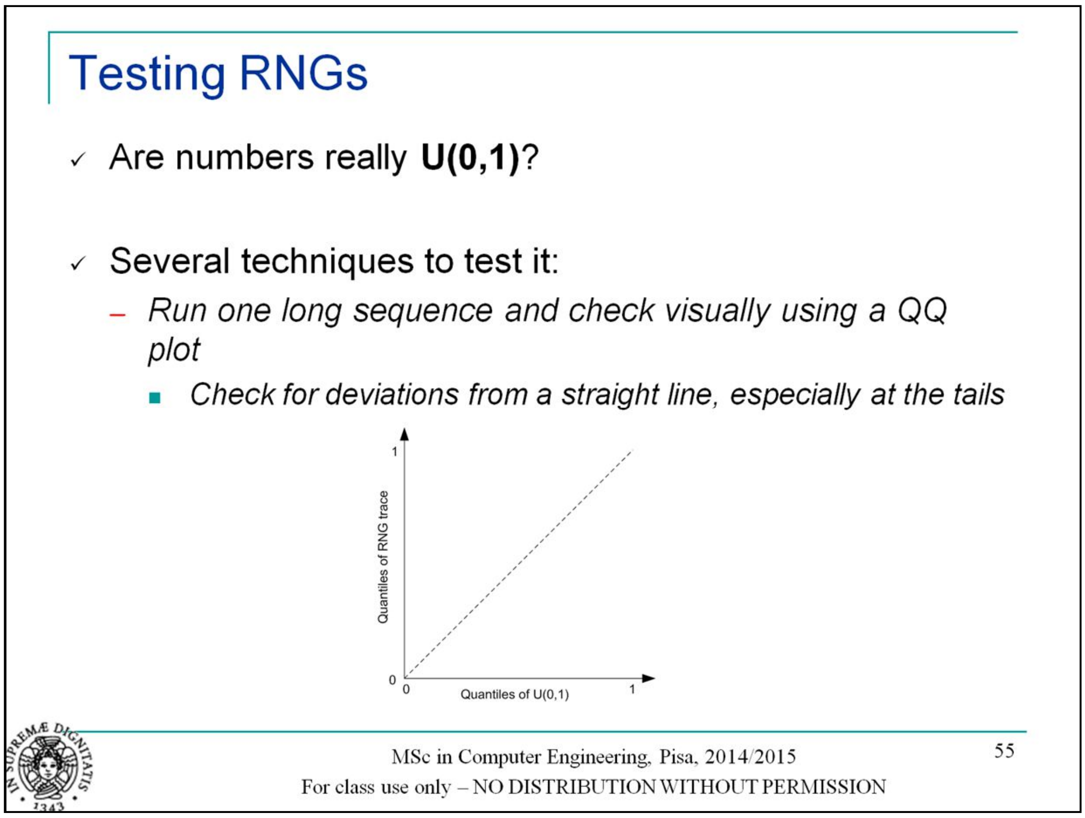
$$x_i = \left(a_1 x_{i-1} + a_2 x_{i-2} + \ldots a_q x_{i-q}\right) \bmod m$$

- ✓ Tausworthe generators
  - *operate directly on bits. next* **bit** *is selected as random.*
  - *Very large periods*
  - *related to cryptographic methods.*
- ✓ Mersenne twister
  - *Huge periods ($2^{19937} - 1$)*
  - *Memory-hungry (~2Kb per instance)*
- ✓ See also Pierre L'Ecuyer's web page

MSc in Computer Engineering, Pisa, 2014/2015 54

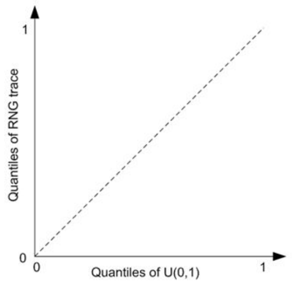
For class use only – NO DISTRIBUTION WITHOUT PERMISSION

Memory-hungry-ness is not a big problem nowadays. However, take into account that you may need **many** RNGs (~O(1000)), in which case it may become a problem. Compare this occupancy with the one of an LCG (which is sizeof(int)).



Testing RNGs

- ✓ Are numbers really **U(0,1)**?
- ✓ Several techniques to test it:
  - *Run one long sequence and check visually using a QQ plot*
    - *Check for deviations from a straight line, especially at the tails*



If you are **proposing** an RNG algorithm yourself (which is unlikely) or you are **testing** an existing RNG which you found in someone else's code (which is more likely), here is what you should do. You should test that the RNG:

a) Produces a stream of **uniform numbers in [0,1)**, and

b) That those numbers are **uncorrelated**.

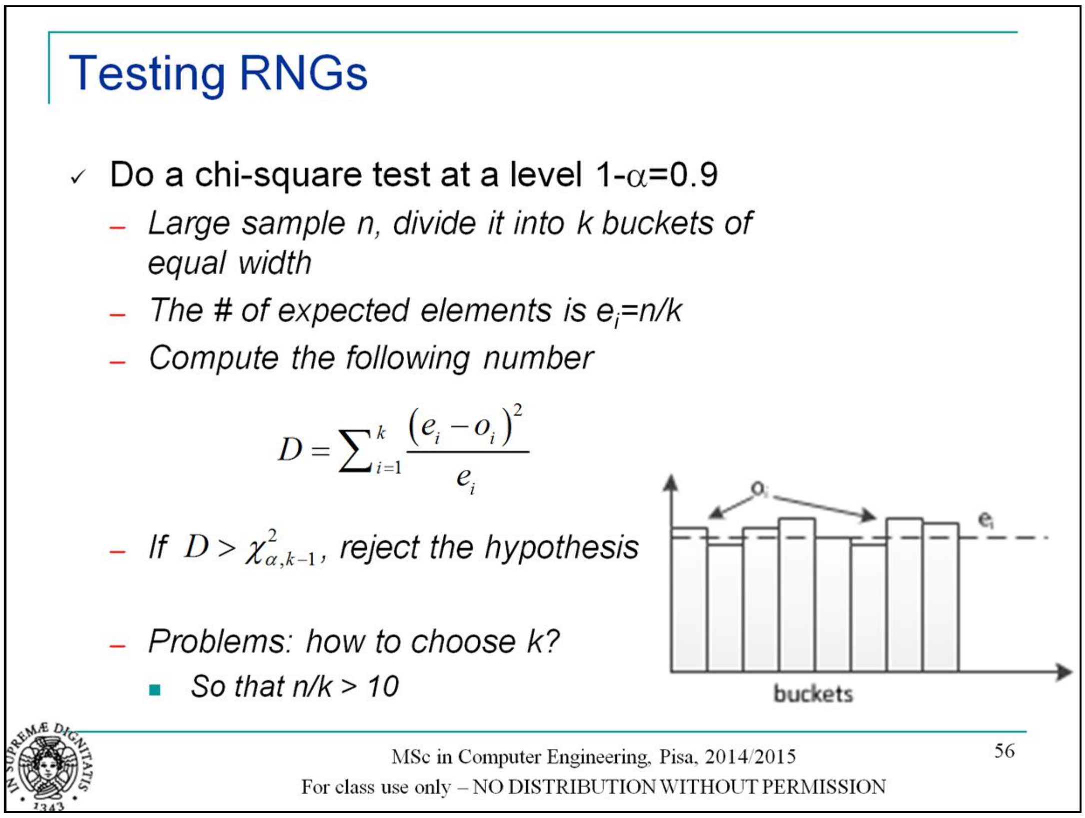
Let us start with the first property.

The first test that you use on an RNG should ascertain whether the RNG actually produces numbers that are U(0,1). This can be done in two ways:

-Visual tests, such as a QQ-plot

-Numerical tests, such as a chi-square test.

Both are particularly straightforward when testing for a uniform probability.

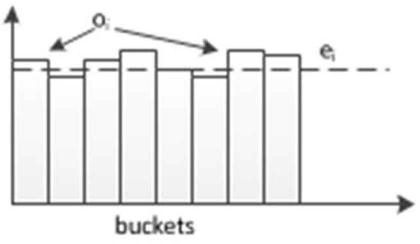


Testing RNGs

- ✓ Do a chi-square test at a level 1-α=0.9
  - *Large sample n, divide it into k buckets of equal width*
  - *The # of expected elements is e_i=n/k*
  - *Compute the following number*

$$D=\sum_{i=1}^{k}\frac{(e_i-o_i)^2}{e_i}$$

  - *If $D>\chi^2_{\alpha,k-1}$, reject the hypothesis*
  - *Problems: how to choose k?*
    - *So that n/k > 10*



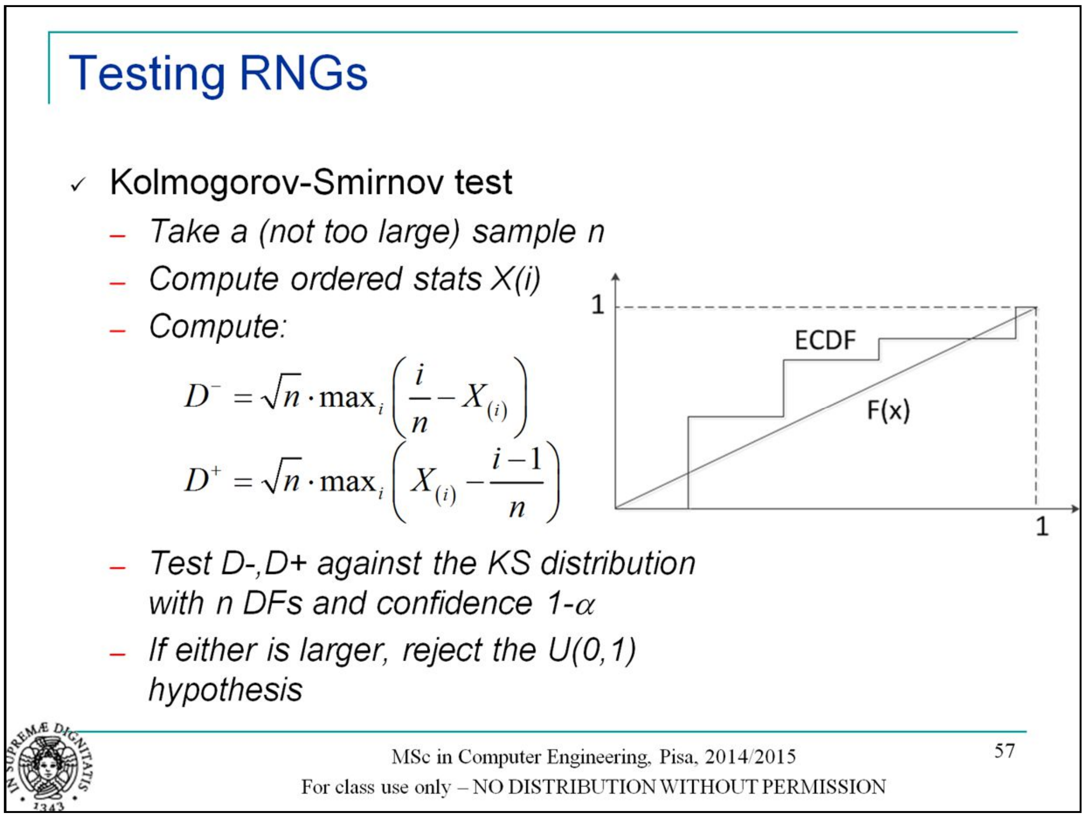
The chi-square test can be used against **any** distribution (not just the uniform).

Its result depends a lot on how you select the number of buckets, their width (which needs not be equal), and the confidence level. Always be wary of these tests (prefer QQ plot when in doubt).

The differences should be **normal**: the further away the deviation from the expected value, the less likely. If you sum several **squared normals** you obtain a Chi-square. Therefore, if D is an **unlikely value** for a chi-square, you should reject the hypothesis.

It can be extended to *n* dimensions. If you want to make a 2-dimensional test, you take each **couple of random numbers**, x,y, and put it into any of the k^2 buckets that you have prepared. Each of these buckets should include e(I,j)=n/(k^2) observations at the end, etc. etc.

The Chi-squared test works with **large samples** (~O(1000)). Otherwise the test is unreliable.



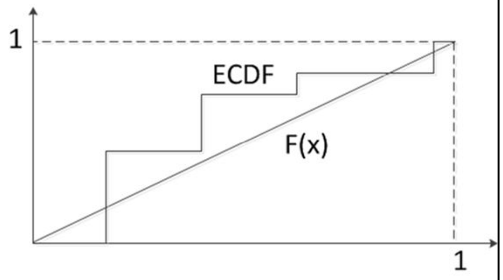
Testing RNGs

- ✓ Kolmogorov-Smirnov test
  - *Take a (not too large) sample n*
  - *Compute ordered stats X(i)*
  - *Compute:*

$$D^- = \sqrt{n} \cdot \max_i \left(\frac{i}{n} - X_{(i)} \right)$$

$$D^+ = \sqrt{n} \cdot \max_i \left(X_{(i)} - \frac{i-1}{n} \right)$$

  - *Test D-,D+ against the KS distribution with n DFs and confidence 1-α*
  - *If either is larger, reject the U(0,1) hypothesis*



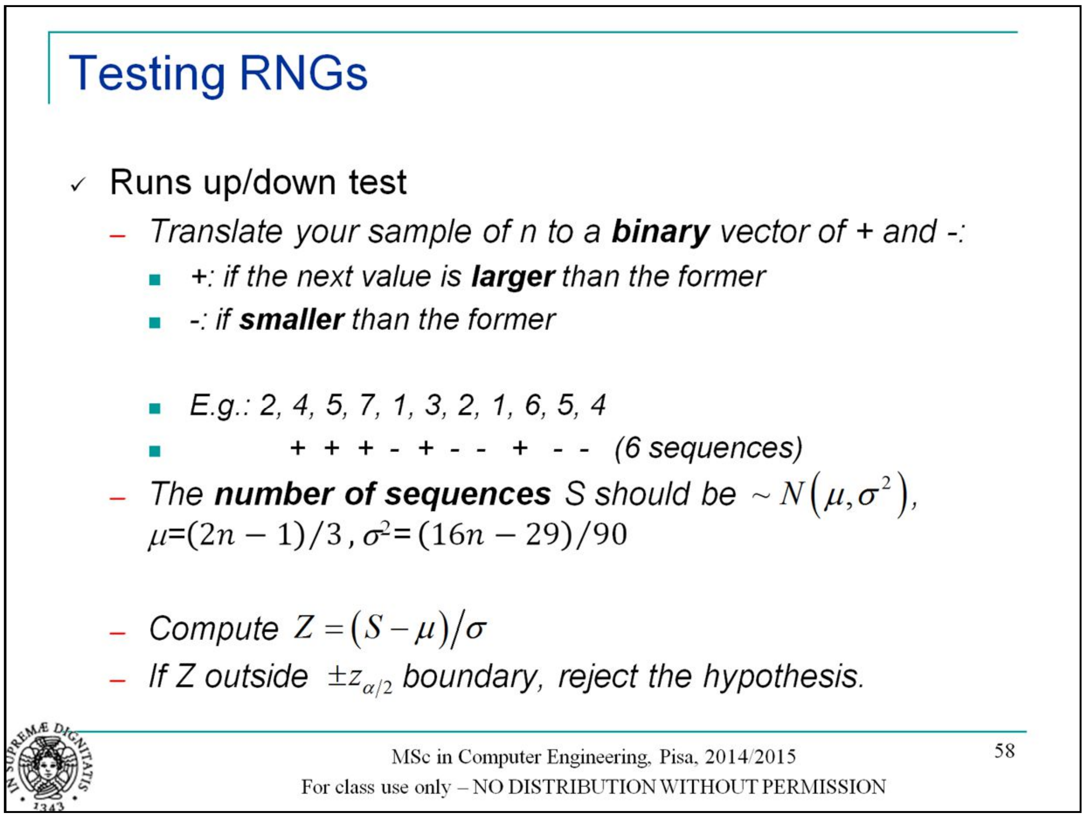
MSc in Computer Engineering, Pisa, 2014/2015
For class use only – NO DISTRIBUTION WITHOUT PERMISSION

57

This test amounts to finding the **maximum** vertical distance between a theoretical distribution (in this case, a uniform) and the ECDF of your data (in this case, the O.S.s of your RNG). The two values D-, D+ are in fact these maximum distances (times a coefficient).

If these distances are **small enough**, then the assumption of matching cannot be rejected. The test should be done against a particular distribution, called the K.S. distribution, which you can find on tables. This depends on the number of samples and the required confidence.

This test can be used with pretty much any distribution. It works better with small samples (when the chi-square test would instead be less significant).

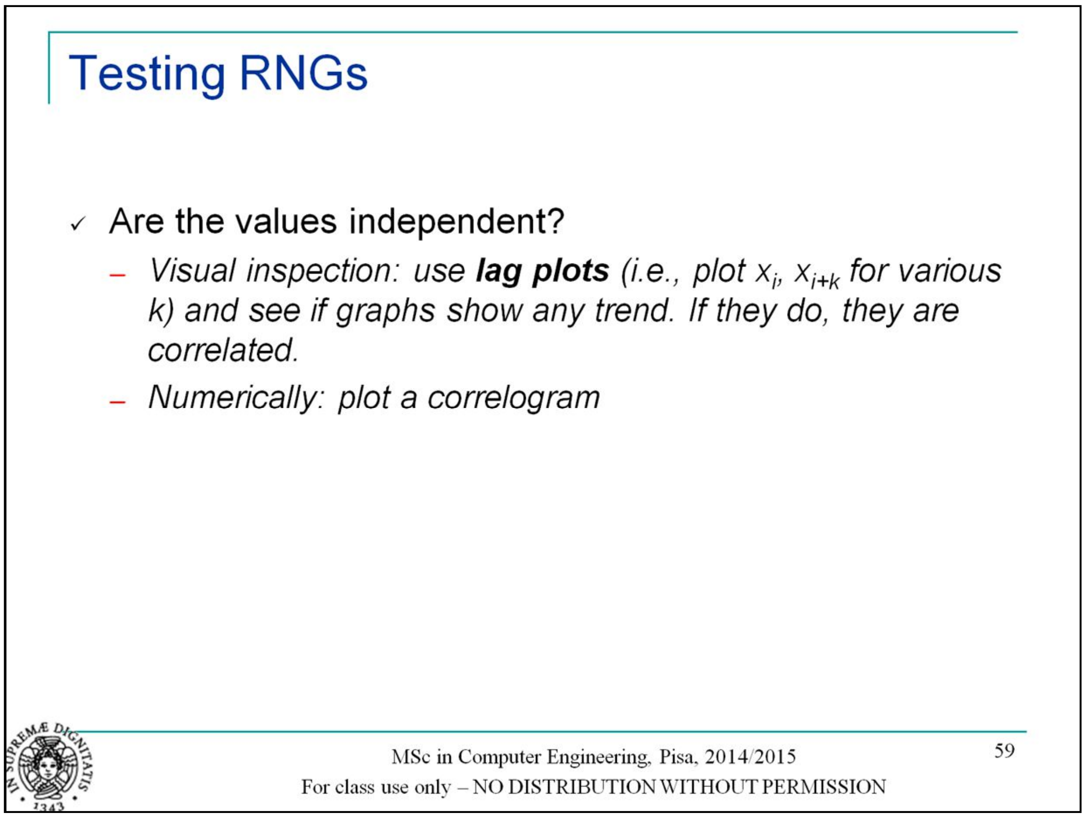


Testing RNGs

- Runs up/down test
  - *Translate your sample of n to a **binary** vector of + and -:*
    - *+: if the next value is **larger** than the former*
    - *-: if **smaller** than the former*
    - *E.g.: 2, 4, 5, 7, 1, 3, 2, 1, 6, 5, 4*
    - *+ + + - + - - + - - (6 sequences)*
  - *The **number of sequences** S should be* $\sim N\left(\mu,\sigma^2\right)$, $\mu=(2n-1)/3$, $\sigma^2=(16n-29)/90$
  - *Compute* $Z=\left(S-\mu\right)/\sigma$
  - *If Z outside* $\pm z_{\alpha/2}$ *boundary, reject the hypothesis.*

MSc in Computer Engineering, Pisa, 2014/2015 58
For class use only – NO DISTRIBUTION WITHOUT PERMISSION

This test can only be used (in this form) against a uniform distribution.

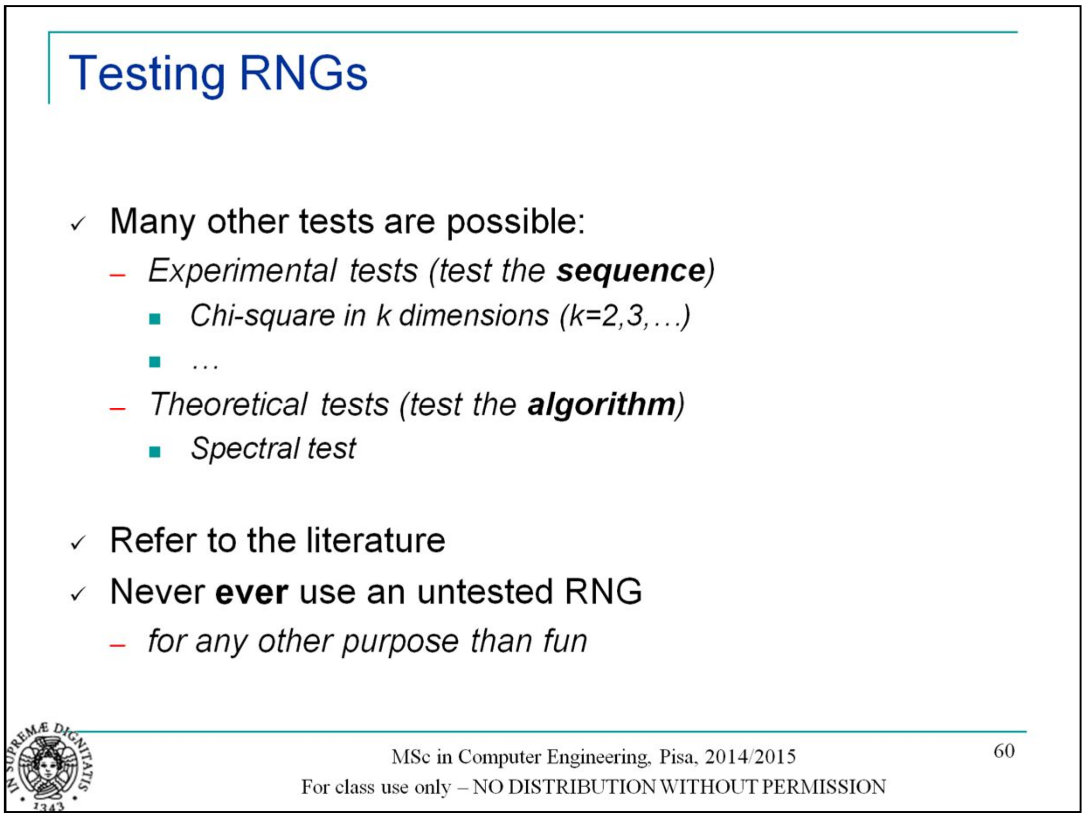


Testing RNGs

- ✓ Are the values independent?
  - *Visual inspection: use **lag plots** (i.e., plot x_i, x_{i+k} for various k) and see if graphs show any trend. If they do, they are correlated.*
  - *Numerically: plot a correlogram*

Now, a sequence of n values equal to 1/n, 2/n, 3/n… is perfectly uniform. It would pass the chi-square test at any significance level. However, it is not to be mistaken for a pseudo-random sequence.

The reason is that subsequent numbers are **correlated** (positively so, in this case). Correlation can be spotted easily in a **lag plot**. Lag plots should be done for several lags.

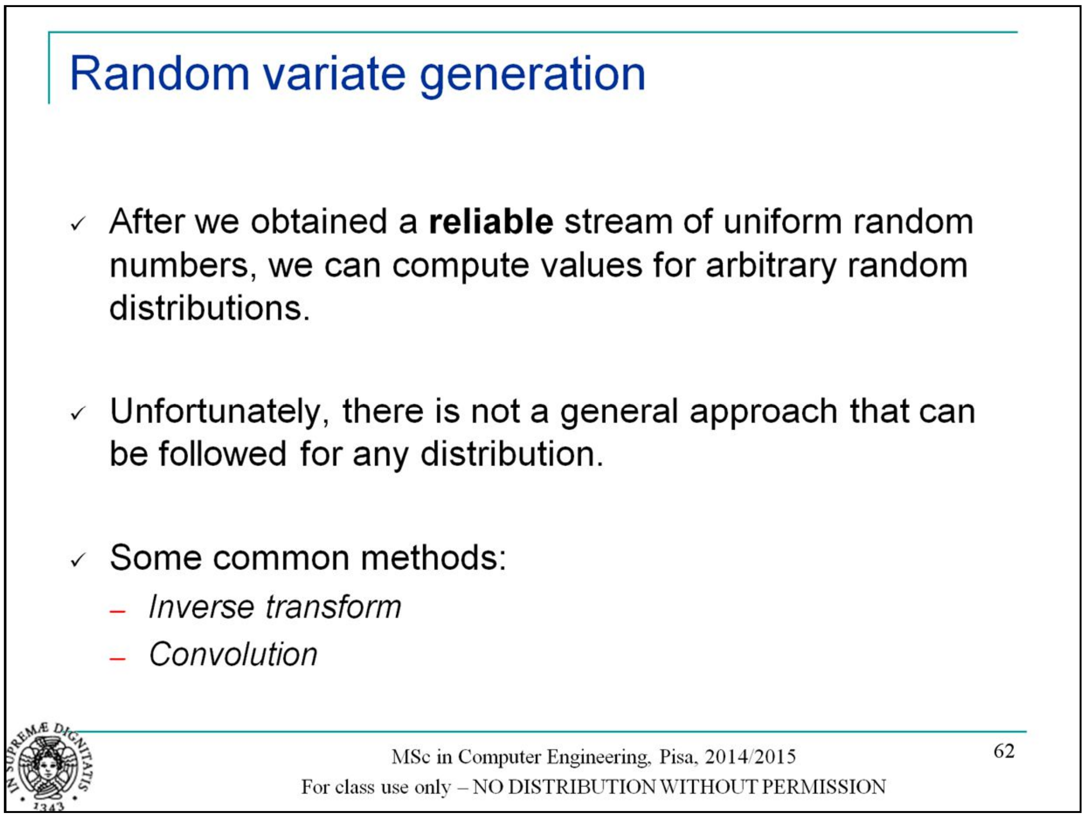


Testing RNGs

- ✓ Many other tests are possible:
  - *Experimental tests (test the **sequence**)*
    - *Chi-square in k dimensions (k=2,3,…)*
    - *…*
  - *Theoretical tests (test the **algorithm**)*
    - *Spectral test*
- ✓ Refer to the literature
- ✓ Never **ever** use an untested RNG
  - *for any other purpose than fun*

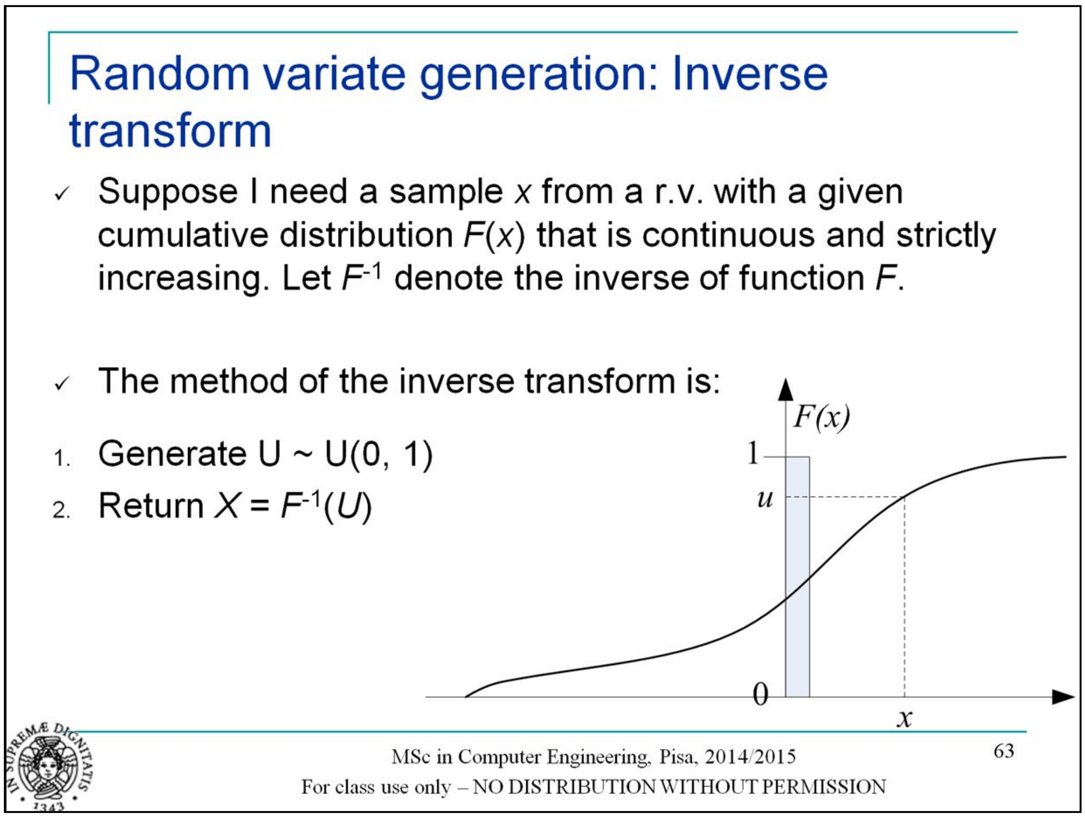
Random number generation

TOUR OF ACCOUNTING
OVER HERE WE HAVE OUR RANDOM NUMBER GENERATOR.
NINE NINE NINE NINE NINE NINE
10/25/01 © 2001 United Feature Syndicate, Inc.
ARE YOU SURE THAT'S RANDOM?
THAT'S THE PROBLEM WITH RANDOMNESS: YOU CAN NEVER BE SURE.



Random variate generation

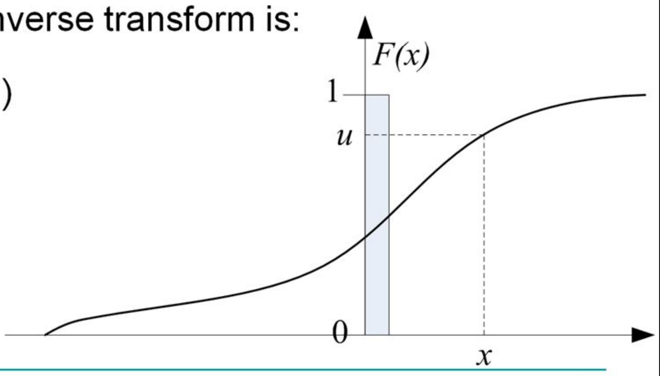
- ✓ After we obtained a **reliable** stream of uniform random numbers, we can compute values for arbitrary random distributions.
- ✓ Unfortunately, there is not a general approach that can be followed for any distribution.
- ✓ Some common methods:
  - *Inverse transform*
  - *Convolution*



Random variate generation: Inverse transform

- ✓ Suppose I need a sample x from a r.v. with a given cumulative distribution $F(x)$ that is continuous and strictly increasing. Let F^{-1} denote the inverse of function F.
- ✓ The method of the inverse transform is:

1. Generate U ~ U(0, 1)
2. Return $X = F^{-1}(U)$



This is one way to generate RV with a given distribution. It is not the only one, of course.

It is quie clever: the codomain of a CDF is [0,1], so if I generate a RN which is ~U(0,1) I can map it to a value of the domain.

Random variate generation: Inverse transform

- ✓ For instance, suppose we want samples from an exponentially distributed RV with mean 1/λ, i.e. its PDF is:

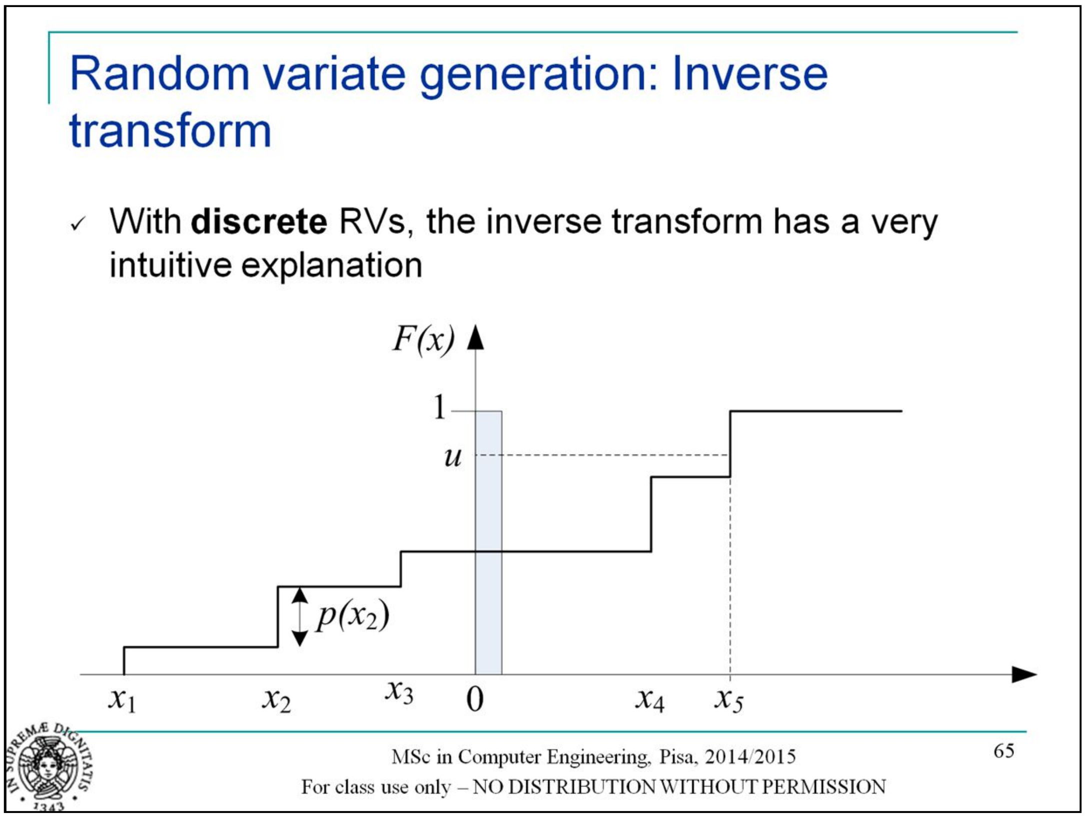
$$f(x) = \lambda \cdot e^{-\lambda x}$$

- ✓ Take a RV *y* uniformly distributed within [0,1).
- ✓ First, we compute the exponential CDF:

$$F(x) = 1 - e^{-\lambda x}$$

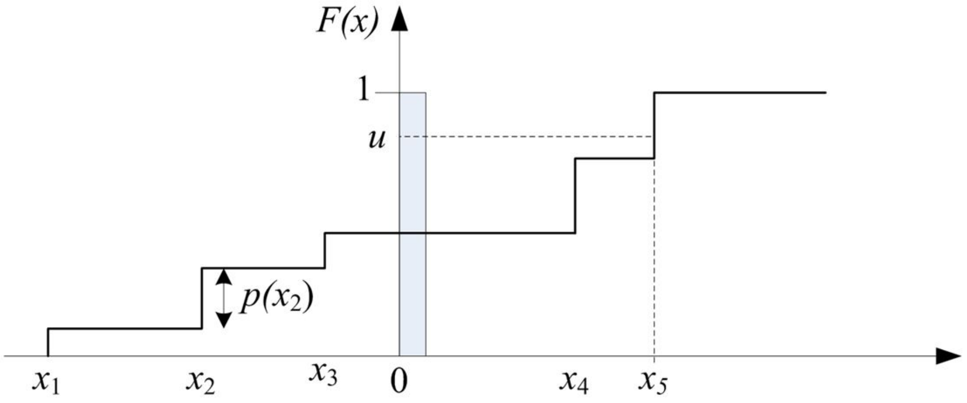
- ✓ Then, we invert the expression and obtain:

$$x = -1/\lambda \cdot \ln(1-y)$$



Random variate generation: Inverse transform

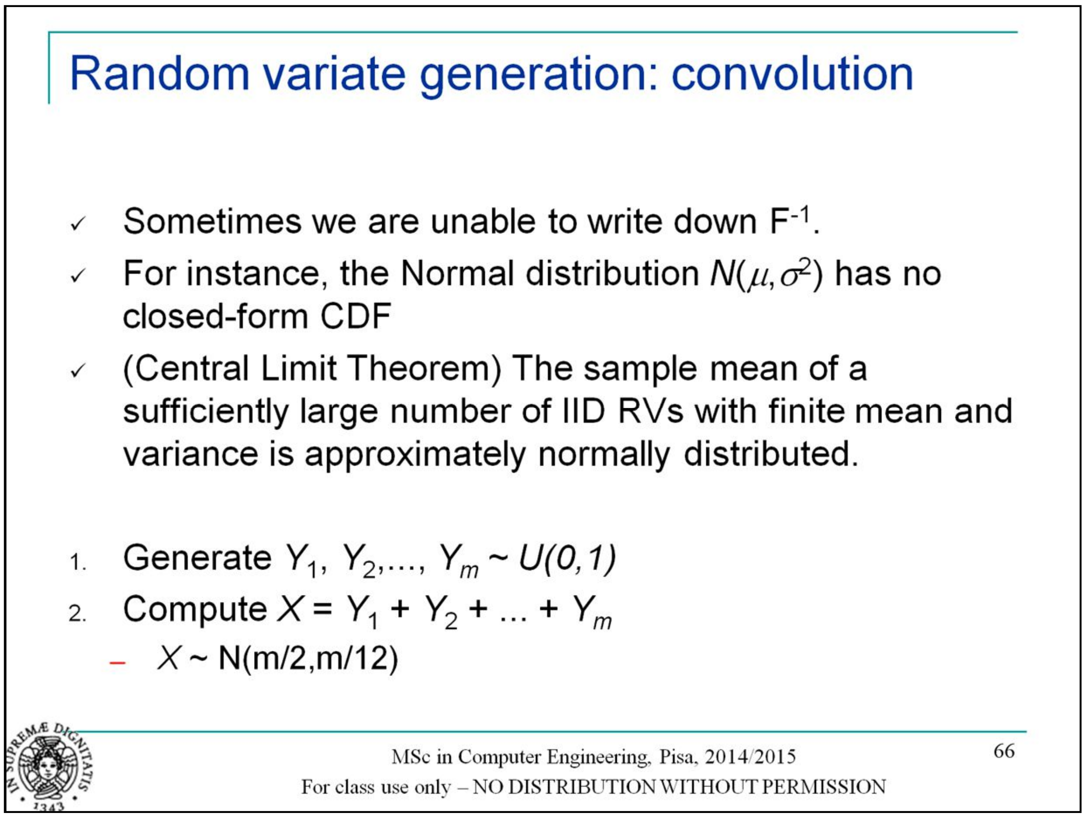
- ✓ With **discrete** RVs, the inverse transform has a very intuitive explanation



With discrete RVs, interval [0,1) is divided into subintervals which are as large as the values of the PMF. Assume that I extract a U(0,1) RV, and the value corresponds to the step of x5: this means that I have extracted value x5 according to that discrete RV.

Obviously, when transforming for generic **discrete variates**, especially those with a large number of values, we need to do things **efficiently**. At the very least, a logarithmic search is due.

A discrete RV can always be represented with an n-vector of couples {x, F(x)}, sorted by increasing x.



Random variate generation: convolution

- ✓ Sometimes we are unable to write down F^{-1}.
- ✓ For instance, the Normal distribution $N(\mu,\sigma^2)$ has no closed-form CDF
- ✓ (Central Limit Theorem) The sample mean of a sufficiently large number of IID RVs with finite mean and variance is approximately normally distributed.

1. Generate $Y_1, Y_2,\ldots, Y_m \sim U(0,1)$
2. Compute $X = Y_1 + Y_2 + \ldots + Y_m$
   - $X \sim$ N(m/2,m/12)

MSc in Computer Engineering, Pisa, 2014/2015 66
For class use only – NO DISTRIBUTION WITHOUT PERMISSION

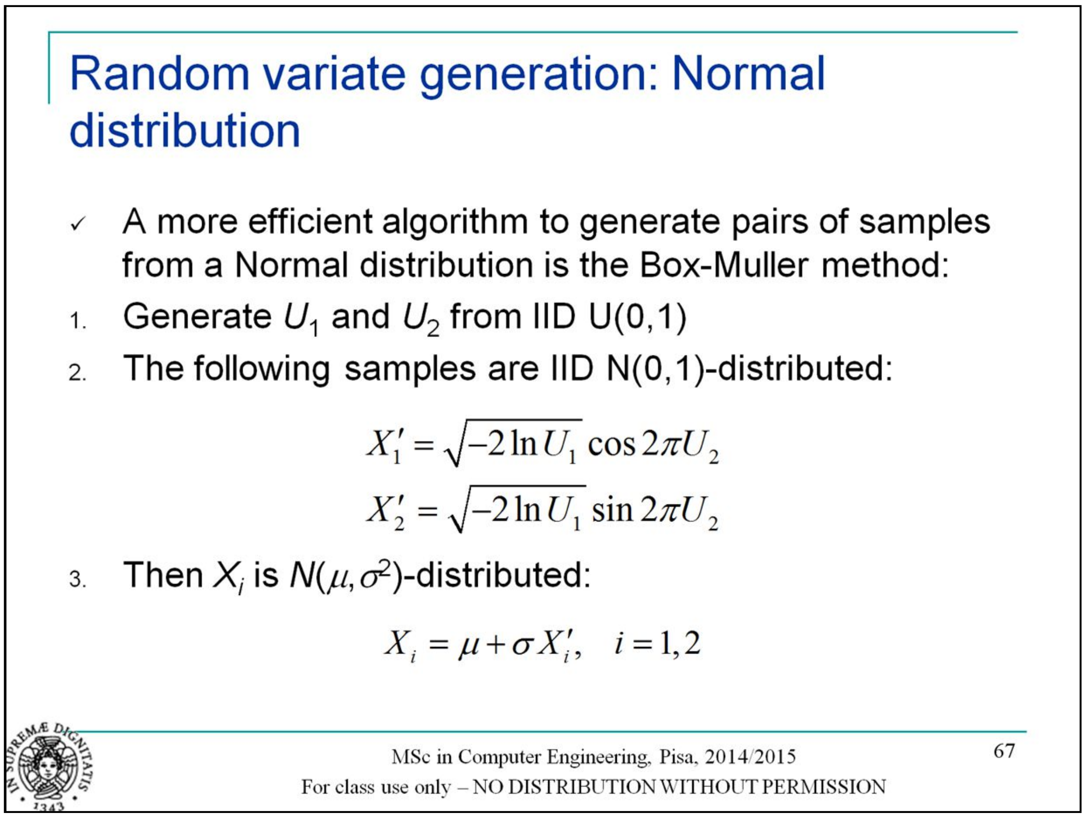
The **inverse-transform method** does not work with all the RVs that are somewhat **related to the Normal distribution** (including the normal itself). Notably, it does not work when the **CDF is not known** in a closed form (hence it can't be inverted).

In order to generate Normal variables, you can resort to the **Central Limit Theorem**. You generate a high number of IID Uniform RVs, and their sum will be distributed as a Normal variable. This is called **convolution method**, since the PDF of the sum of IID RVs is in fact the convolution of the single PDFs.

In general, you need some IID RVs (at least 6-7). This method has two drawbacks:

a) It is costly, in any case. You need to use 6-7 good seeds of an RNG to produce just one normal.
b) You introduce errors: you will never observe large-modulus values, since uniforms are finite. If you want very large values, you need to waste very many seeds.

Note that you might, in theory, use the standard approximation for the Normal percentiles in order to invert the Normal. However, this introduces **errors**, especially at the tails, so you can't risk it. One thing is to use it for (approximate) visual plots, another is to generate good-quality Random Variates.



Random variate generation: Normal distribution

- ✓ A more efficient algorithm to generate pairs of samples from a Normal distribution is the Box-Muller method:

1. Generate U_1 and U_2 from IID U(0,1)
2. The following samples are IID N(0,1)-distributed:

$$X_1' = \sqrt{-2\ln U_1}\cos 2\pi U_2$$
$$X_2' = \sqrt{-2\ln U_1}\sin 2\pi U_2$$

3. Then X_i is $N(\mu,\sigma^2)$-distributed:

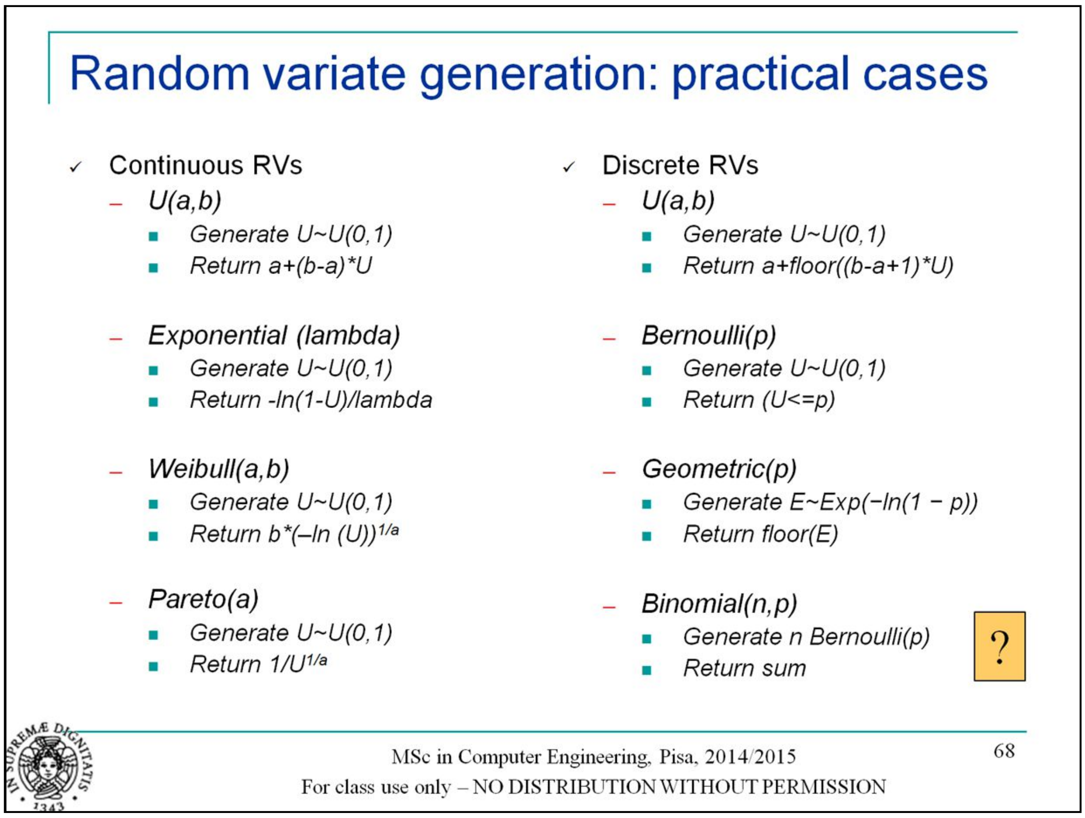
$$X_i = \mu + \sigma X_i', \quad i = 1,2$$

http://en.wikipedia.org/wiki/Box-Muller_transform

Box-Muller transform has something to do with changing the coordinates from Cartesian to polar.

Note that in this case you **will** observe very large values (in modulus), since you have the logarithm of something which is in [0,1).

The drawback is that you will need to do complex floating-point operations, such as logs, sin/cosine, and square roots.



Random variate generation: practical cases

- Continuous RVs
  - *U(a,b)*
    - *Generate U~U(0,1)*
    - *Return a+(b-a)*U*
  - *Exponential (lambda)*
    - *Generate U~U(0,1)*
    - *Return -ln(1-U)/lambda*
  - *Weibull(a,b)*
    - *Generate U~U(0,1)*
    - *Return* $b*(-\ln(U))^{1/a}$
  - *Pareto(a)*
    - *Generate U~U(0,1)*
    - *Return* $1/U^{1/a}$
- Discrete RVs
  - *U(a,b)*
    - *Generate U~U(0,1)*
    - *Return a+floor((b-a+1)*U)*
  - *Bernoulli(p)*
    - *Generate U~U(0,1)*
    - *Return (U<=p)*
  - *Geometric(p)*
    - *Generate E~Exp(−ln(1 − p))*
    - *Return floor(E)*
  - *Binomial(n,p)*
    - *Generate n Bernoulli(p)*
    - *Return sum*

?

MSc in Computer Engineering, Pisa, 2014/2015
For class use only – NO DISTRIBUTION WITHOUT PERMISSION

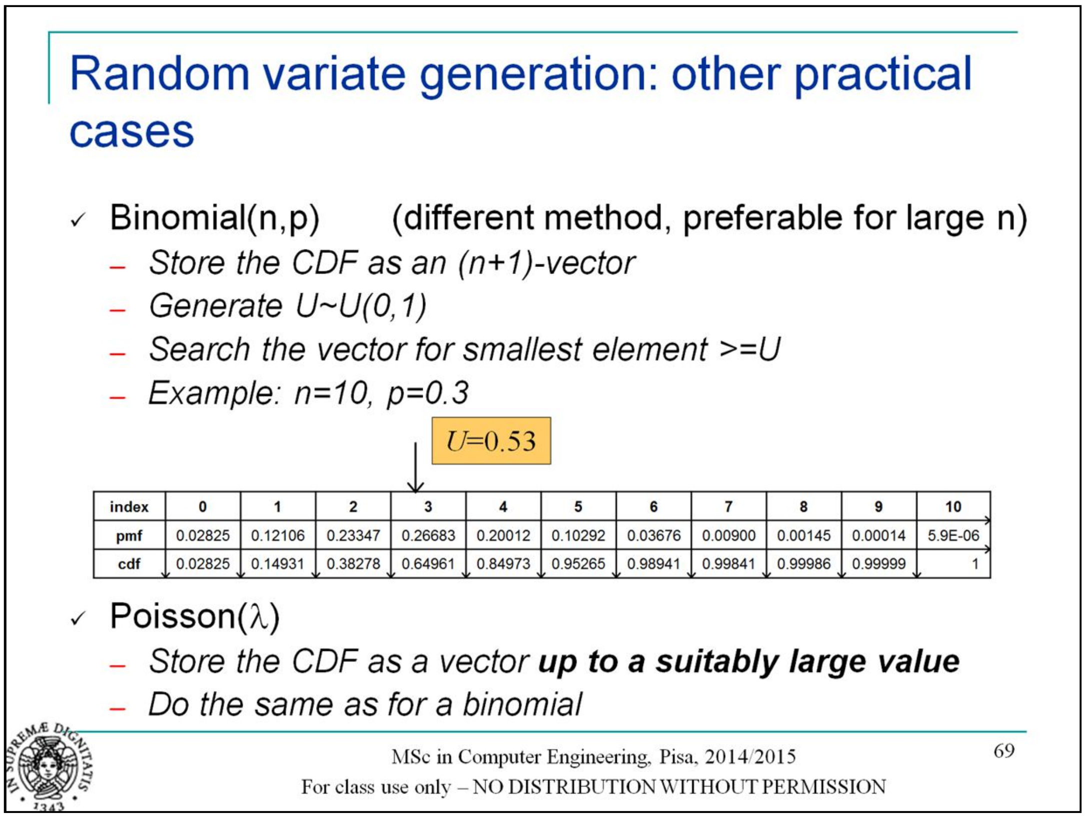
68

More examples can be found in the Law-Kelton book.

Note that generation of random variates must be **fast**. If you write down code for this, it should be highly optimized: perform all the operations involving constants **once**, and store the results.

The explanation for the geometric comes from the known relationship between the exponential and the geometric.

Note that generating a binomial the way it is explained here is **inefficient**, especially for a large n. There is another method, shown in the next slide.



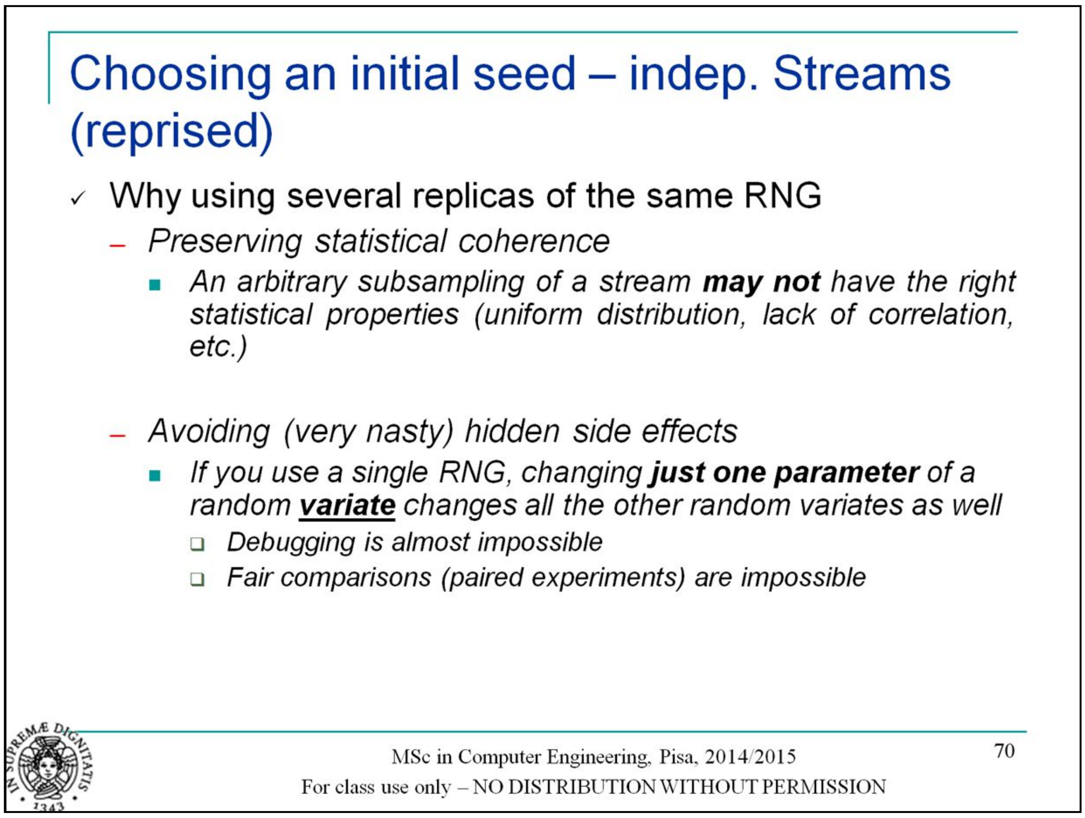
Random variate generation: other practical cases

- ✓ Binomial(n,p) (different method, preferable for large n)
  - *Store the CDF as an (n+1)-vector*
  - *Generate U~U(0,1)*
  - *Search the vector for smallest element >=U*
  - *Example: n=10, p=0.3*

U=0.53

index	0	1	2	3	4	5	6	7	8	9	10
pmf	0.02825	0.12106	0.23347	0.26683	0.20012	0.10292	0.03676	0.00900	0.00145	0.00014	5.9E-06
cdf	0.02825	0.14931	0.38278	0.64961	0.84973	0.95265	0.98941	0.99841	0.99986	0.99999	1

- ✓ Poisson(λ)
  - *Store the CDF as a vector* ***up to a suitably large value***
  - *Do the same as for a binomial*



Choosing an initial seed – indep. Streams (reprised)

- ✓ Why using several replicas of the same RNG
  - *Preserving statistical coherence*
    - *An arbitrary subsampling of a stream* ***may not*** *have the right statistical properties (uniform distribution, lack of correlation, etc.)*
  - *Avoiding (very nasty) hidden side effects*
    - *If you use a single RNG, changing* ***just one parameter*** *of a random* ***variate*** *changes all the other random variates as well*
      - *Debugging is almost impossible*
      - *Fair comparisons (paired experiments) are impossible*

MSc in Computer Engineering, Pisa, 2014/2015 70

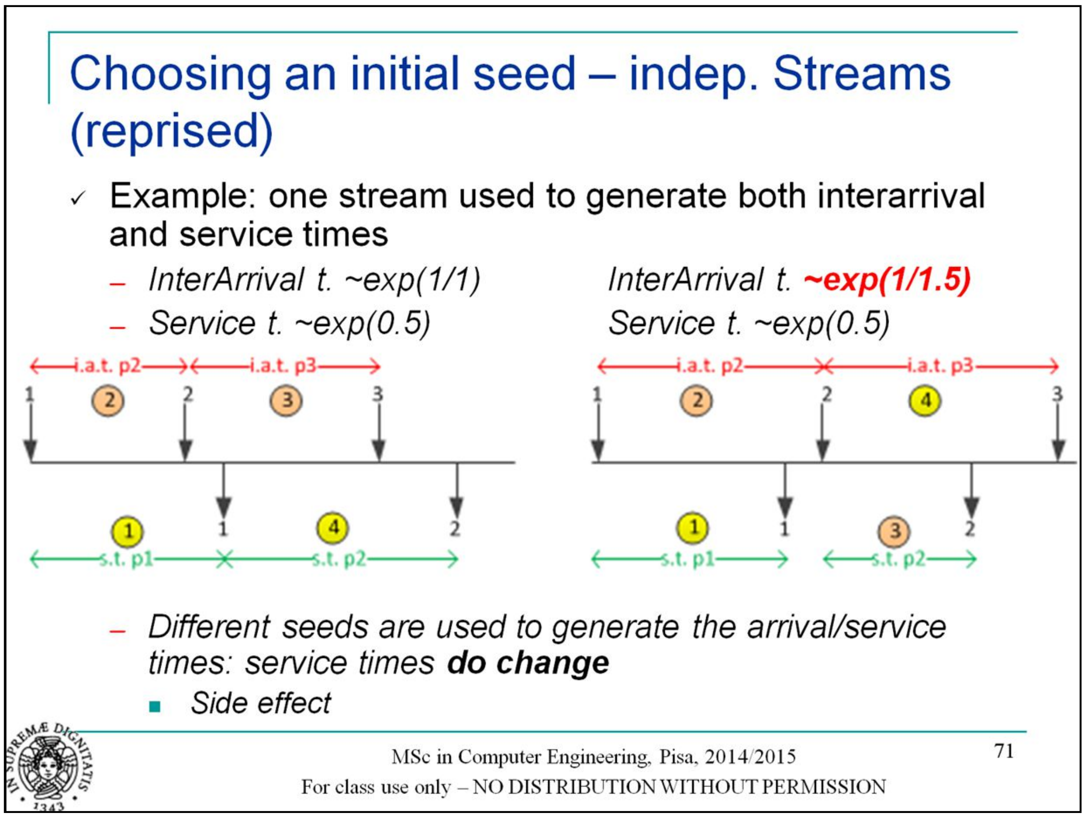
For class use only – NO DISTRIBUTION WITHOUT PERMISSION

You can't use a single RNG to extract all the values for the *k* streams.

1) In fact, the tests for the RNG may not be able to guarantee that "arbitrary samplings" of your RNG yield streams of uncorrelated numbers. Even if xi and x(i+1) are uncorrelated, x(i) and x(i+k) (with a large k) may be correlated. If you are using the same RNG to produce k streams, you may end up believing that streams are independent when they are not.

One may counter that **good RNGs** should be exempt from this kind of problem. They may be, but then there is still a residual probability that, by subsampling (at a non-constant pace) a stream of random numbers, you do end up with something which is not uniform in [0,1]. Perhaps you only see "large" values, and you cannot exclude it, nor you can know it a priori (and you will never check it a posteriori, of course).

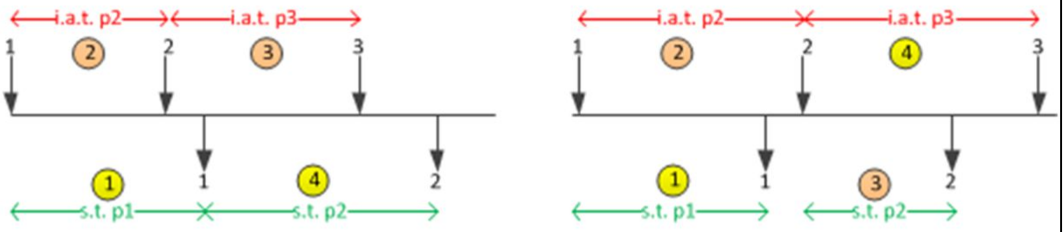
2) This is a lot more important. If you are using the same RNG for different purposes, you are introducing **dependences** between streams of RNs that should be independent of each other. Here's an example



Choosing an initial seed – indep. Streams (reprised)

- Example: one stream used to generate both interarrival and service times
  - *InterArrival t. ~exp(1/1)*
  - *Service t. ~exp(0.5)*

InterArrival t. ***~exp(1/1.5)***
Service t. ~exp(0.5)



- *Different seeds are used to generate the arrival/service times: service times* ***do change***
  - *Side effect*

Assume that you use **one** stream to generate **both** interarrival and service times:

- When a packet arrives
1) If the queue is empty, the packet goes under service, and its service time is generated
2) The next interarrival time is generated.

- When a packet departs
1) If the queue is non empty, another packet goes under service, and its service time is generated

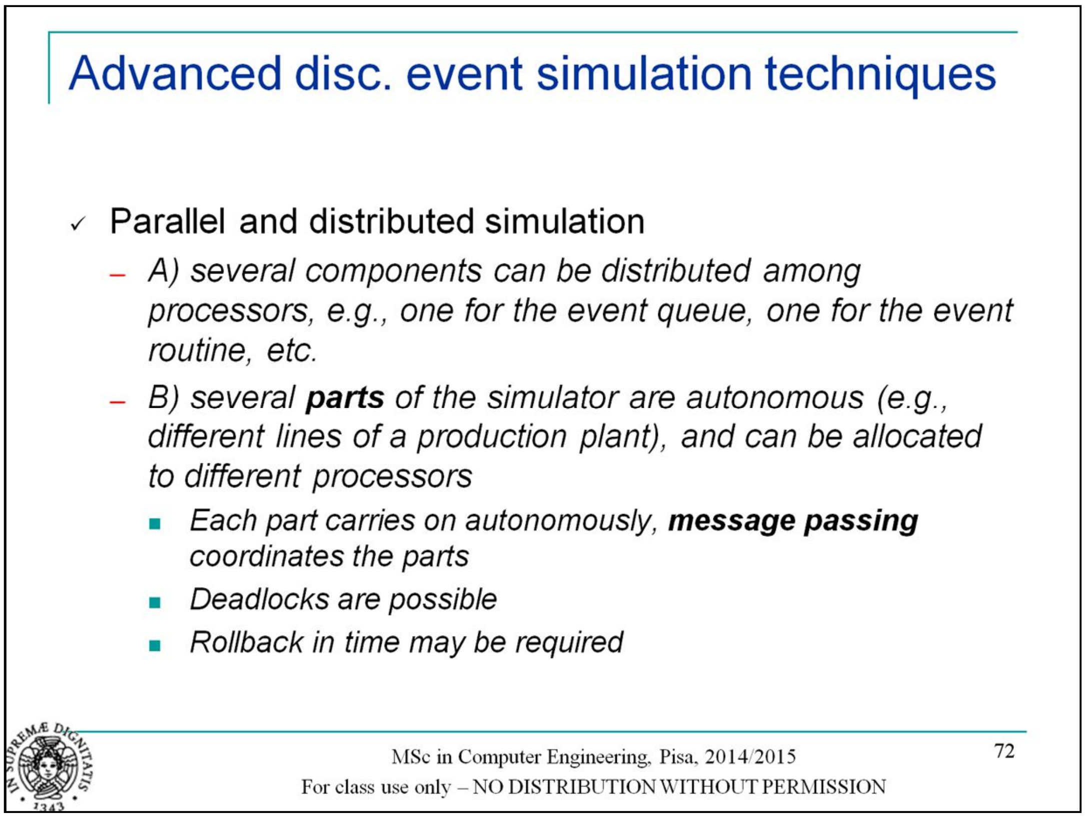
Every time a new random time is generated, I will use a new seed from the (one and only) RNG. Therefore, I will use
- seed 1 to generate the service time of packet 1
- seed 2 to generate the iat of packet 2, etc.

The sequence of events is thus: A1, A2, D1, A3, D2.

Now, suppose that I change the **average interarrival time**, i.e., by increasing it. I would expect that the sequence of observed service times should be **identical to the previous case**. In fact, I am using the same initial seed, and I have not modified the average of the exponential distribution of the service times.

Instead, this is not the case, since modifying the interarrival time constant **alters the sequence of events**: it is now A1, **D1,** A2, **D2,** A3. This implies that the service times of individual packets are computed using **different seeds**, even though the initial seed is the same as before.

This makes debugging **very hard**, and – worse yet – makes **paired experiments impossible**. In fact, the sequence of your service times comes to depend not only on the initial seed, but also on the sequence of arrival times. You will never be able to replicate the same sequence of service times, unless you also replicate the same sequence of arrival times.



Advanced disc. event simulation techniques

- ✓ Parallel and distributed simulation
  - *A) several components can be distributed among processors, e.g., one for the event queue, one for the event routine, etc.*
  - *B) several **parts** of the simulator are autonomous (e.g., different lines of a production plant), and can be allocated to different processors*
    - *Each part carries on autonomously, **message passing** coordinates the parts*
    - *Deadlocks are possible*
    - *Rollback in time may be required*

MSc in Computer Engineering, Pisa, 2014/2015 72
For class use only – NO DISTRIBUTION WITHOUT PERMISSION

P&D simulation is an active research field, which we will not deal with in the course of this lecture.

Of course, designing a simulator to be parallel and/or distributed is an entirely different task. The main problem is to maintain a consistent, shared notion of **time**.

Module X might generate event x at time t, which requires module Y to do something at time t.

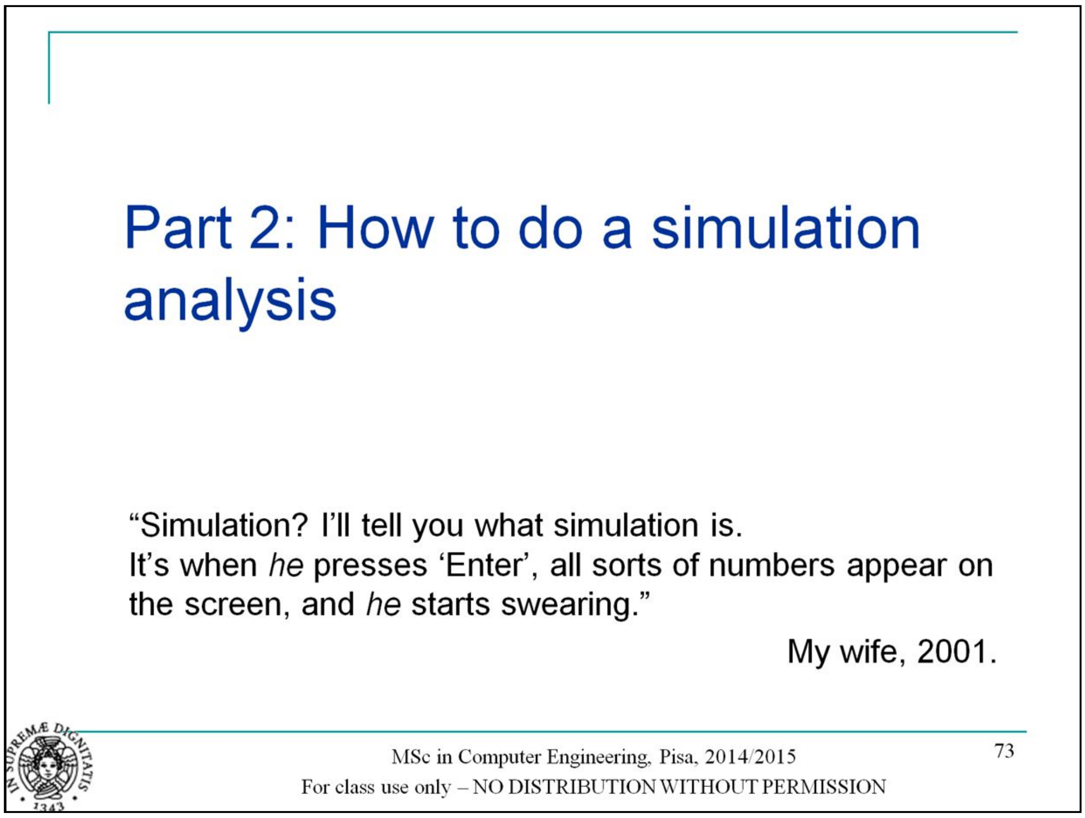
Meanwhile, module Y could be already working at time t+D, and generating events of its own. If it gets an event for time t (i.e., earlier in time), then it needs to check whether everything that it has done in [t, t+D] is still meaningful. If it is not, it should roll back in time and take a different path.

When you play with synchronization, you always run the risk of incurring deadlocks.

The important thing to know is that a simulator is **designed from the start** to be either sequential or parallel.

With **parallel simulators**, the speedup due to parallelism can be large to huge, especially if the modules are **semi-autonomous** and **computation-intensive**: they do a lot of processing and little interaction.

If, instead (as happens with networks) the modules do **very little processing,** and have **a lot of interactions**, then it is preferable to go sequential. Going parallel buys you no speedup at all.



Part 2: How to do a simulation analysis

"Simulation? I'll tell you what simulation is.
It's when *he* presses 'Enter', all sorts of numbers appear on the screen, and *he* starts swearing."

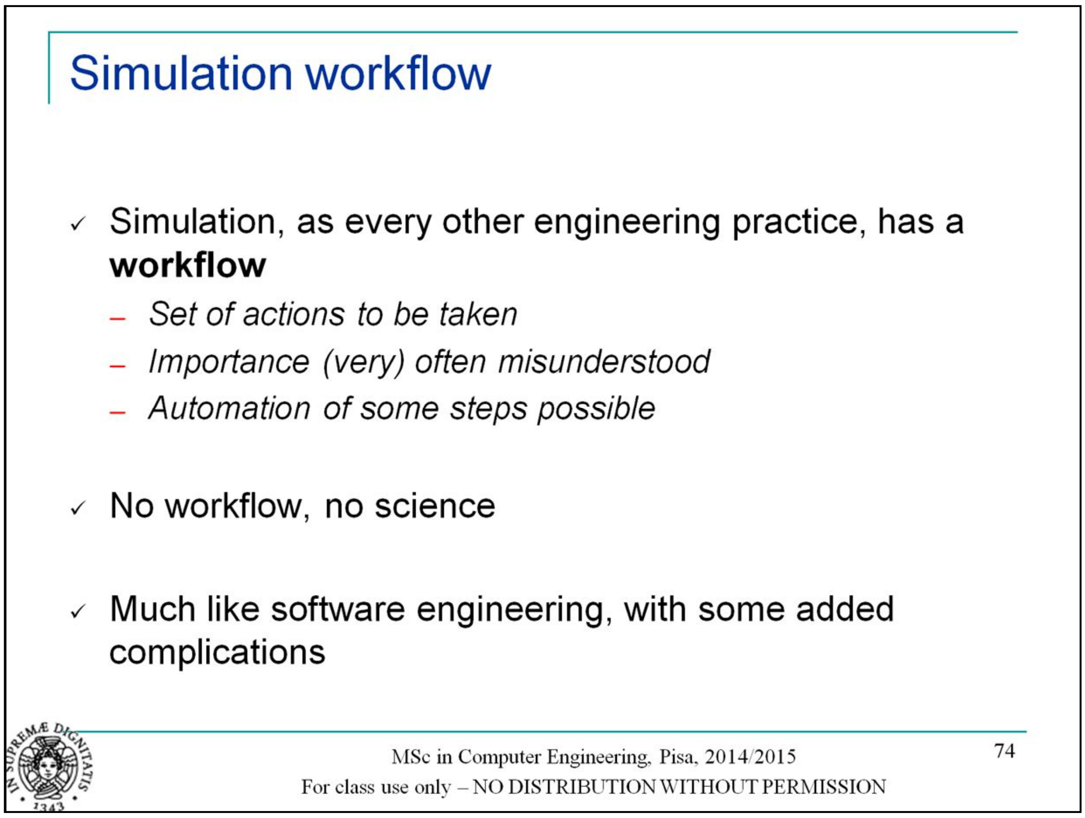
My wife, 2001.

This lecture is aimed at advising those who want to embark on a simulation project, especially a large-scale one.

Some lessons are taken from books, others have been learned the hard way.

Most of the things that are stated here are common to **performance studies** at large, i.e., even those not involving simulation. Some are instead specific to simulation.

We will assume the perspective of the **leader of a large team** about to embark on a **large-scale** study for a big customer. Lots of money involved, several people to coordinate.



Simulation workflow

- ✓ Simulation, as every other engineering practice, has a **workflow**
  - *Set of actions to be taken*
  - *Importance (very) often misunderstood*
  - *Automation of some steps possible*
- ✓ No workflow, no science
- ✓ Much like software engineering, with some added complications

From this slide on, we check the list of everything that is required when you make a simulation analysis.

Take the following perspective: you are leading a **large-scale simulation project**, coordinating several scientists and engineers, to do **something important** for a **big customer**.

The checklist items will be shown **in a sequence**. However, the real process often requires jumping forwards and (mostly) backwards. Every time that, at a given stage, you feel that something is wrong, you just step back and set it right.

The importance of managing the workflow correctly is normally **underestimated**, in both industrial and scientific environments. Most people think that once you have written a bug-free simulator (given a system model), what you feed it as an input, how you analyze its output, etc. are not particularly important aspects, as if the code itself would be able to guarantee the correctness of its usage.

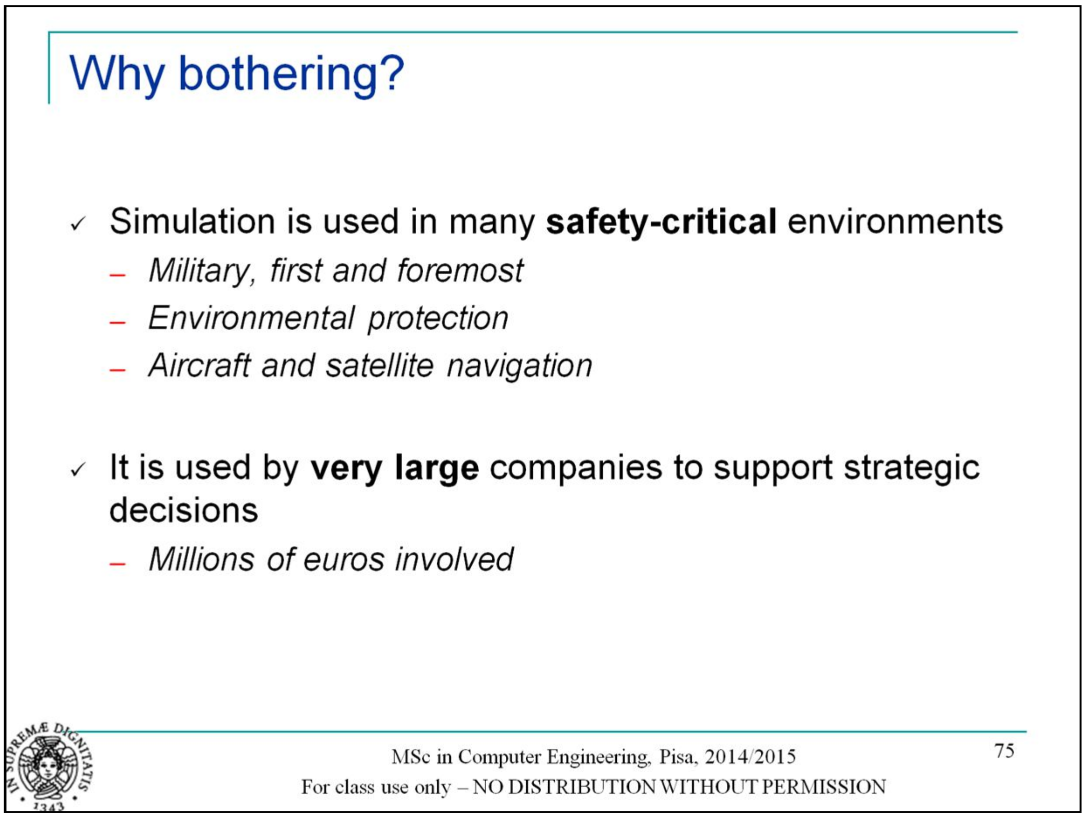
This is, however, plainly wrong. If you concentrate on the code and neglect all the rest, you don't just **risk to fail**, you are simply **certain that you will**. And all the time spent in honing the code to perfection will be – of course – wasted, since the results will not have anything to do with reality. Most computer engineers/scientists do not possess the background to **appreciate the importance** of what lies **around the simulator**.

Now, some of the steps of the simulation workflow are **boring as hell**, especially if you are expected to do them manually: file comparisons, computations, etc. You shouldn't be surprised to learn that these are done **manually most of the time**. This makes simulation analysis **costly and error prone**.

Every time you have long, boring tasks at hand, the obvious solution is **to write some code that automates those tasks**. There are some solutions, but the field is largely unexplored.

Simulation, in a sense, is not too different from software engineering. If you need a 200-LOC program for your own purposes, just write it any way you wish. If, instead, you do **programming in the large**, then you can't improvise: you have to follow standards, best practices, etc. (otherwise you won't be able to obtain state-of-the-art results).

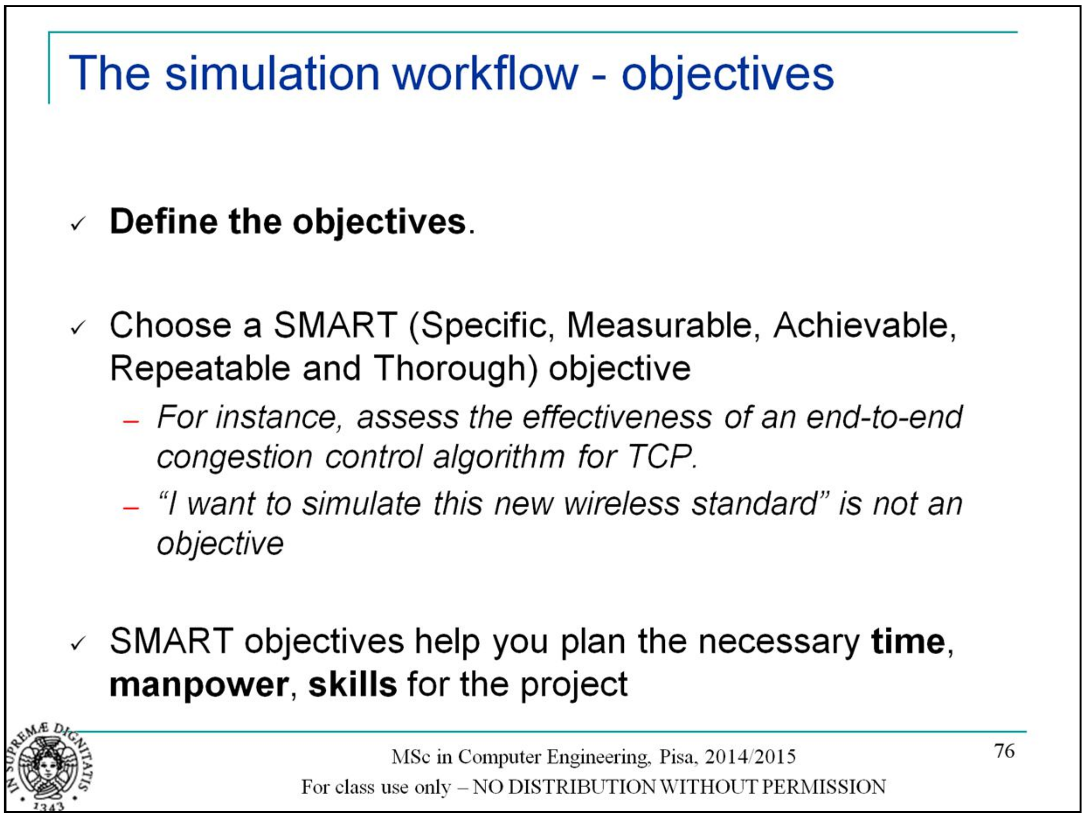
Similarly, if you have to write a 200-LOC simulator for your own purposes (e.g., a quick-and-dirty evaluation of two alternatives in a simplified setting), just do it (and never think of using its results for anything else). If you have to evaluate whether to buy equipment A or B, and your company is investing some ME on equipment, your analysis can't be anything less than state-of-the art.



Why bothering?

- ✓ Simulation is used in many **safety-critical** environments
  - *Military, first and foremost*
  - *Environmental protection*
  - *Aircraft and satellite navigation*
- ✓ It is used by **very large** companies to support strategic decisions
  - *Millions of euros involved*

These are fields where prototyping shouldn't be used. And, of course, you can't improvise.



The simulation workflow - objectives

- ✓ **Define the objectives**.
- ✓ Choose a SMART (Specific, Measurable, Achievable, Repeatable and Thorough) objective
  - *For instance, assess the effectiveness of an end-to-end congestion control algorithm for TCP.*
  - *"I want to simulate this new wireless standard" is not an objective*
- ✓ SMART objectives help you plan the necessary **time**, **manpower**, **skills** for the project

Trivial as it may seem, this is fundamental. Never start to write down a simulator **before** knowing what you want to do with it.

Specific as in "not vague"

Measureable as in "we can tell when we're finished"

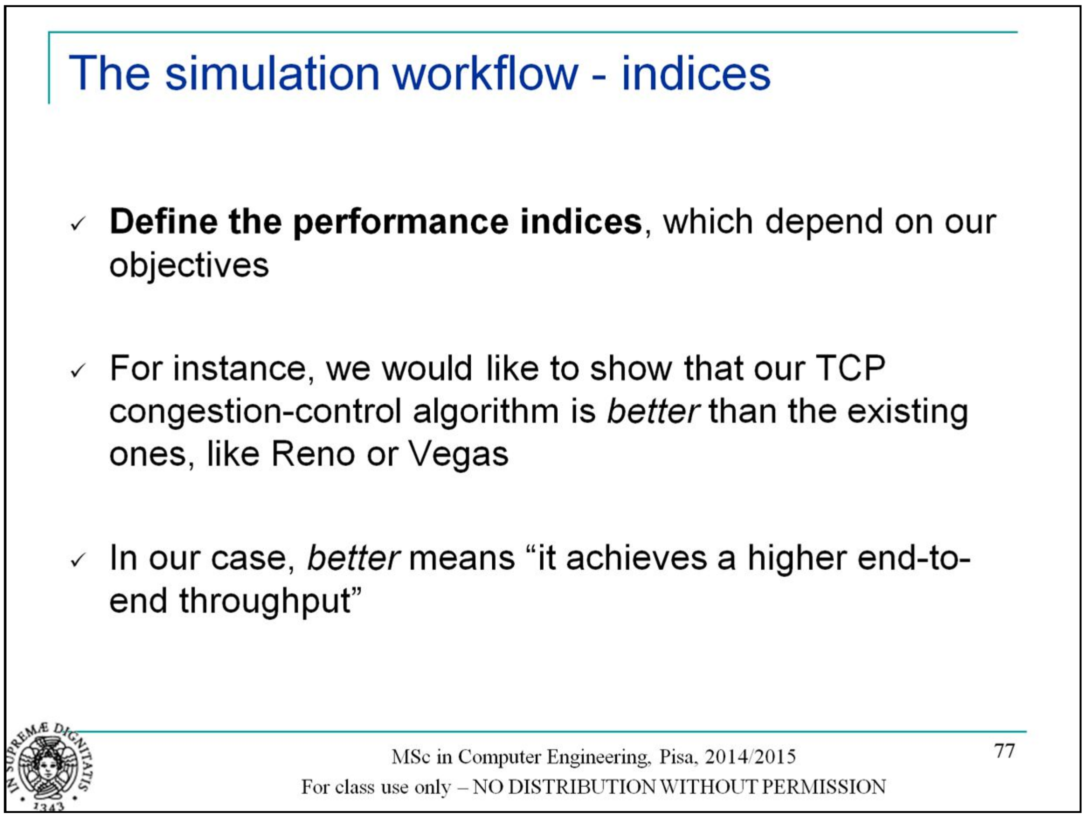
Achievable as in "it something that really *can* be done with the time, money and expertise we have at our disposal"

Note that the quoted statement fails at all the requirements, yet is what most people say when they embark on a simulation project.

-There is no achievable goal. You will never know if you're finished (when you're bored, most likely)

-It is unspecific: one thing is to assess the performance of **web browsing** on that technology, another is to assess its **bit error rate**. You need very different skills to assess the two things, and very different models

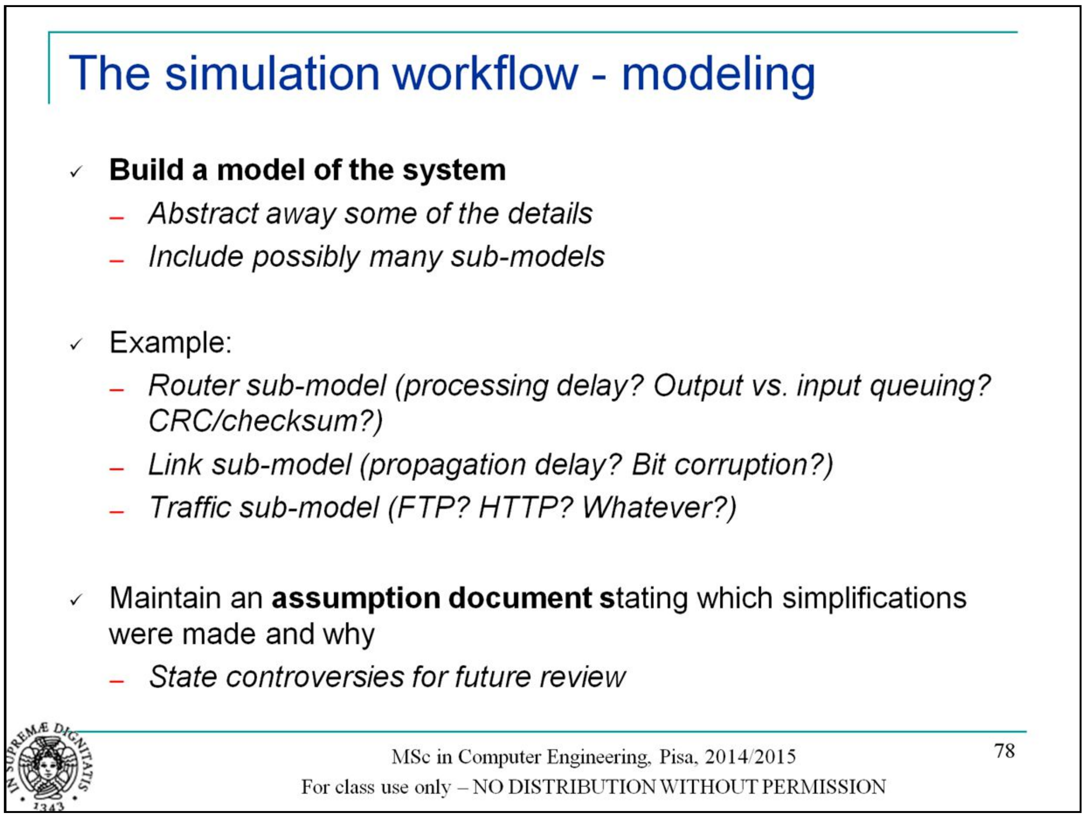
- In the first case, you can probably model the channel as something which gives you an error probability (a Markov chain will do most of the times). On the other hand, you will need the full protocol stack, with good models of user behavior, of a web server, etc.
- In the second case, you need a very accurate model of signal propagation at a given frequency, given the power level of the transmitter, the coding scheme, the antenna layout, etc. In this case, all the models required in the previous case are unnecessary.



The simulation workflow - indices

- ✓ **Define the performance indices**, which depend on our objectives
- ✓ For instance, we would like to show that our TCP congestion-control algorithm is *better* than the existing ones, like Reno or Vegas
- ✓ In our case, *better* means "it achieves a higher end-to-end throughput"

This is trivial as well (but needs to be stated nonetheless): once you have a high-level objective, you need to translate it into **measurable performance indices**.



The simulation workflow - modeling

- ✓ **Build a model of the system**
  - *Abstract away some of the details*
  - *Include possibly many sub-models*
- ✓ Example:
  - *Router sub-model (processing delay? Output vs. input queuing? CRC/checksum?)*
  - *Link sub-model (propagation delay? Bit corruption?)*
  - *Traffic sub-model (FTP? HTTP? Whatever?)*
- ✓ Maintain an **assumption document s**tating which simplifications were made and why
  - *State controversies for future review*

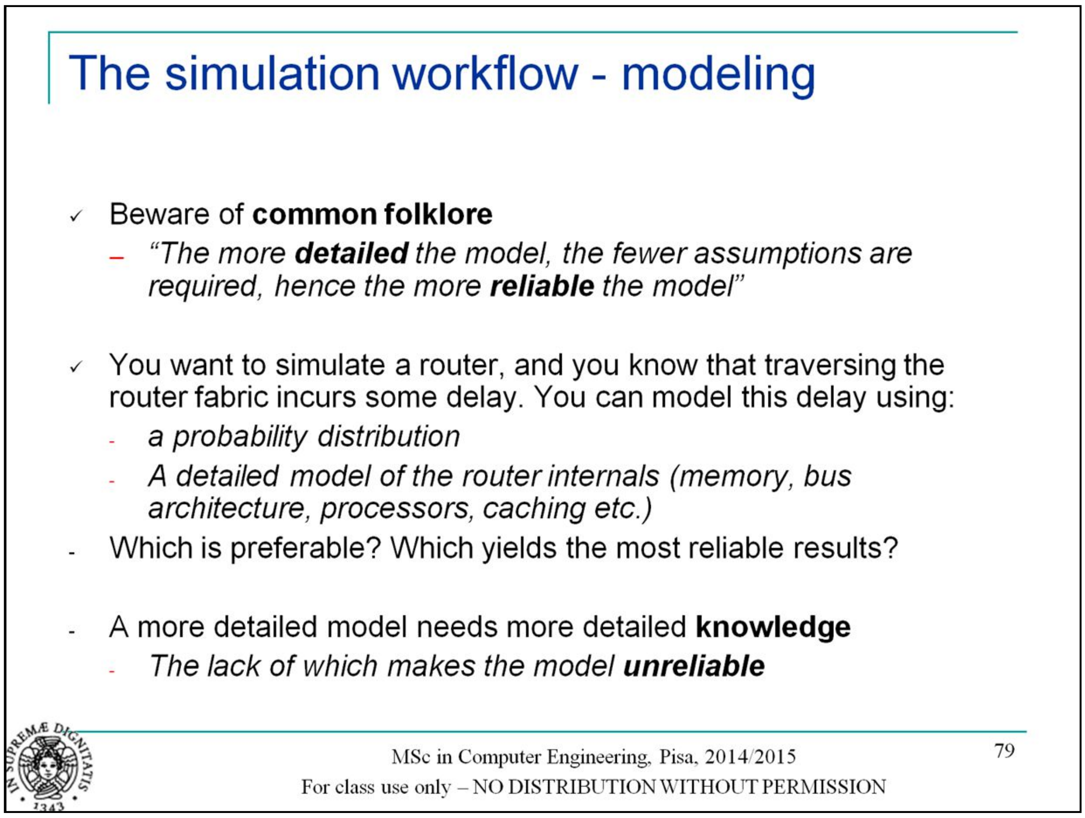
MSc in Computer Engineering, Pisa, 2014/2015 78
For class use only – NO DISTRIBUTION WITHOUT PERMISSION

Once objectives are defined, the next step involves creating models that abstract away some of the complexity of the real system. The models must retain enough detail to allow for the goals of the simulation study to be met. **Modeling is a complex activity**; one that requires a good dose of insight and a fair amount of experimentation.

You need to **know deeply** that piece of reality, possibly enlisting the help of someone else (your customer, for instance).

Most of the times, you can only setup a simulation study by **interacting with the users**. If you are hired to simulate a manufacturing plant, chances are that you know nothing about either manufacturing or plants. In this case, modeling is by necessity **a joint effort** with the users (likely, the owners of the plant, the workers, the technicians, etc.). Whenever there's interaction, there must be **documentation**. It is also very helpful to **cover your back** later on.

Modeling is really an **art.** Much like playing music or writing songs. The more you do it, the better you become at it. Unless you're hopelessly tone-deaf, of course.


The simulation workflow - modeling

- ✓ Beware of **common folklore**
  - *"The more **detailed** the model, the fewer assumptions are required, hence the more **reliable** the model"*
- ✓ You want to simulate a router, and you know that traversing the router fabric incurs some delay. You can model this delay using:
  - *a probability distribution*
  - *A detailed model of the router internals (memory, bus architecture, processors, caching etc.)*
- Which is preferable? Which yields the most reliable results?
- A more detailed model needs more detailed **knowledge**
  - *The lack of which makes the model **unreliable***

MSc in Computer Engineering, Pisa, 2014/2015 79
For class use only – NO DISTRIBUTION WITHOUT PERMISSION

The problem is that it is not too **difficult** to build a software replica of how a router behaves. However, you need to insert **parameters** in this model, and you may not have the slightest idea of what acceptable values for those parameters may be.

This way, you simply end up making **wrong assumptions**, which may invalidate the whole model.

If you stick to, e.g., an exponential distribution of router traversal time (because that's what you find in the literature), with a mean time taken from vendor data sheets, you surely make **some error**.

If you, instead, try to model the internals of the router without being the very expert in router architectures, you **surely** end up forgetting something crucial (or making wild assumptions about details that you can't fathom), and may

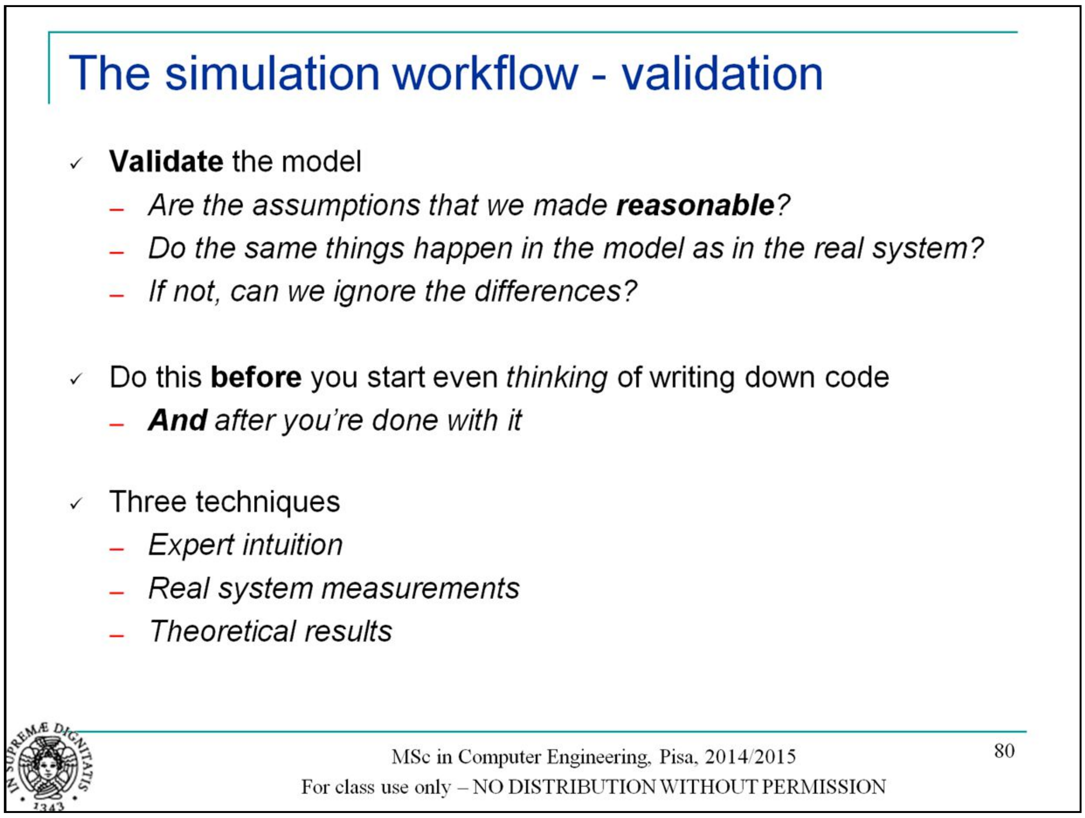
•Miss the true result, even by orders of magnitude

•Harbor the **false certainty** that your model is reliable **because it is so detailed**.

"I don't know enough" is by far the hardest phrase to force out of (computer) engineers.

There's another subtlety in this, and it's **upgradability**. If you are spending a lot of time in modeling something, you may want to be able to **upgrade** your model when conditions change.

In this example, you may want to assess what changes if new-generation routers come out. These may have a completely different architecture. Perhaps you spent some times honing the fine details of the old router bus management strategy, and all this **effort will be wasted**. If you had stuck to a **higher-level model** (e.g., an exponential distribution of router traversal times), you would simply need to change the mean (or, in a worst case, the probability distribution), which takes half an hour at most.



The simulation workflow - validation

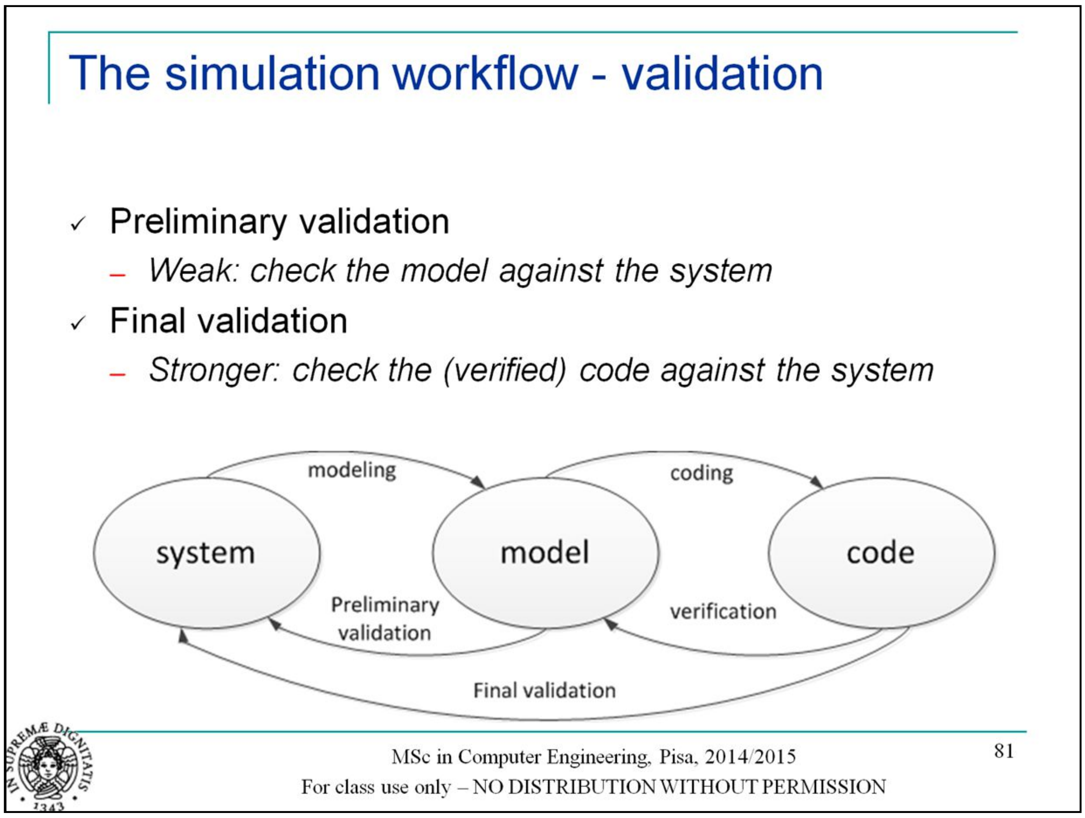
- ✓ **Validate** the model
  - *Are the assumptions that we made **reasonable**?*
  - *Do the same things happen in the model as in the real system?*
  - *If not, can we ignore the differences?*
- ✓ Do this **before** you start even *thinking* of writing down code
  - ***And** after you're done with it*
- ✓ Three techniques
  - *Expert intuition*
  - *Real system measurements*
  - *Theoretical results*

Models need to be validated, that is, one must make sure that they correctly represent the system for which they stand in.

Validating a model means verifying if the assumptions that have been made to obtain that model lead to something that behaves reasonably like the real system.

You can do this using three techniques, which may or may not be available for all the models. You can use the techniques concurrently, of course.

Depending on the situation, you may not be able to do validation (i.e., when you are simulating a completely new system, that does not exist yet, for which no theoretical results are available and that no one is an expert of).



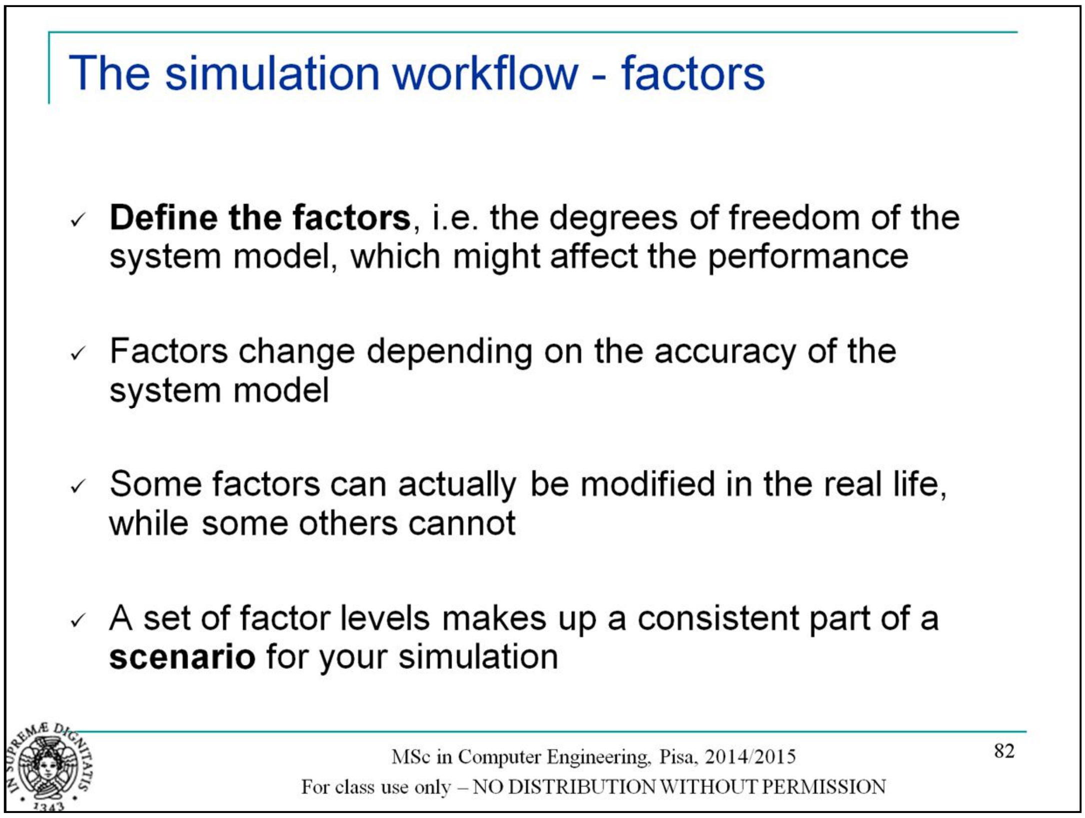
The simulation workflow - validation

- ✓ Preliminary validation
  - *Weak: check the model against the system*
- ✓ Final validation
  - *Stronger: check the (verified) code against the system*

$$\begin{tikzcd} \text{system} \arrow[r, bend left, "\text{modeling}"] & \text{model} \arrow[l, bend left, "\text{Preliminary validation}"] \arrow[r, bend left, "\text{coding}"] & \text{code} \arrow[l, bend left, "\text{verification}"] \arrow[ll, bend left=60, "\text{Final validation}"] \end{tikzcd}$$

MSc in Computer Engineering, Pisa, 2014/2015 81
For class use only – NO DISTRIBUTION WITHOUT PERMISSION

The initial validation is weaker: a model is a **document**, so it's hard to compare against a system. When you have the **code** (which has to be verified first), you can make the **final, a-posteriori** validation.


The simulation workflow - factors

- ✓ **Define the factors**, i.e. the degrees of freedom of the system model, which might affect the performance
- ✓ Factors change depending on the accuracy of the system model
- ✓ Some factors can actually be modified in the real life, while some others cannot
- ✓ A set of factor levels makes up a consistent part of a **scenario** for your simulation

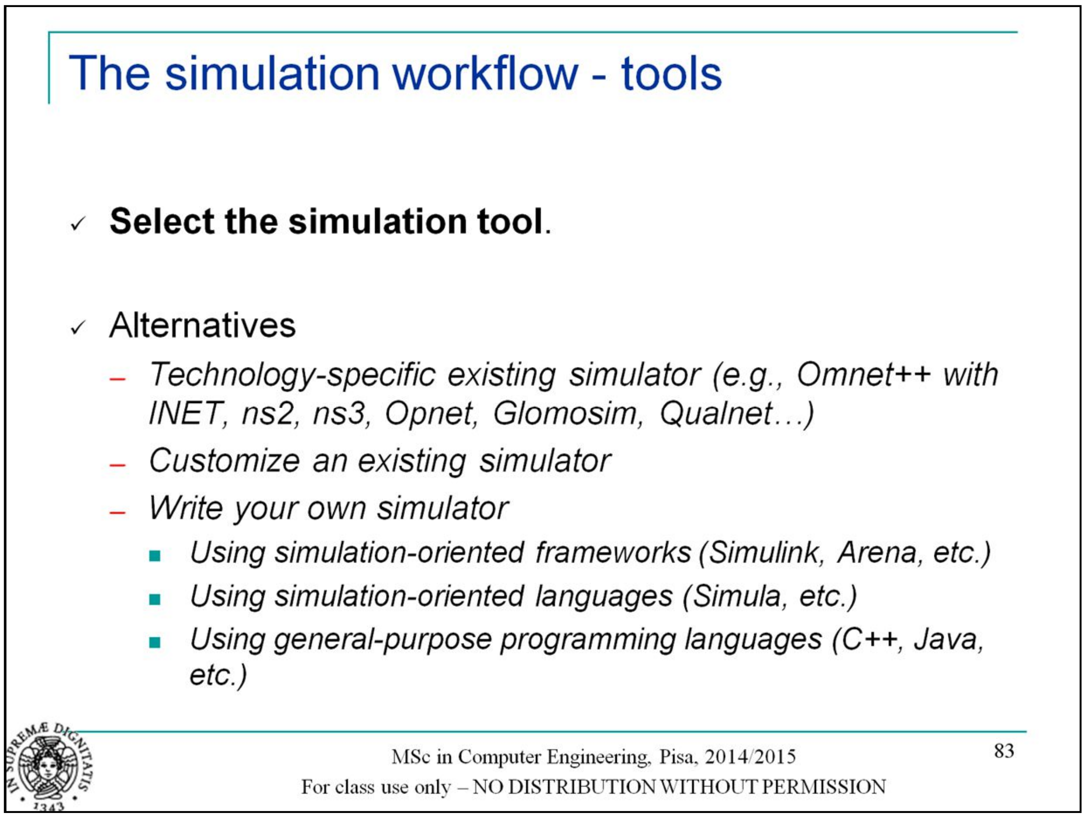
MSc in Computer Engineering, Pisa, 2014/2015 82
For class use only – NO DISTRIBUTION WITHOUT PERMISSION

One key aspect of modeling is **deciding factors**. What are the tuning knobs for my system? How much should I be able to turn them?

In the case of a TCP connection, some of the factors I can define are:
-The **application traffic model**. This is a **qualitative** (i.e., not quantitative) factor.
-The **number of hops** on which I want to test the TCP, which is instead a **quantitative** factor
-The **buffer size** of the TCP receiver, which is quantitative
-The **congestion control algorithm** (seeing as I want to compare some of them using simulation), which is not quantitative again.

Some of the above can only be considered as factors in a **simulated environment**. For instance, the number of hops between two endpoints cannot be changed in real life, since it depends on routing, which is outside my control. I can change it in a simulation environment.

When you run a simulation, the set of factor levels composes part of a **simulation scenario**.



The simulation workflow - tools

- ✓ **Select the simulation tool**.
- ✓ Alternatives
  - *Technology-specific existing simulator (e.g., Omnet++ with INET, ns2, ns3, Opnet, Glomosim, Qualnet…)*
  - *Customize an existing simulator*
  - *Write your own simulator*
    - *Using simulation-oriented frameworks (Simulink, Arena, etc.)*
    - *Using simulation-oriented languages (Simula, etc.)*
    - *Using general-purpose programming languages (C++, Java, etc.)*

MSc in Computer Engineering, Pisa, 2014/2015
For class use only – NO DISTRIBUTION WITHOUT PERMISSION
83

Once you have a clear idea of what you are going to do, how to prove it, varying what, just **pause and reflect before starting to write the code.**

There are several options:

-ready-made simulators for the type of system that you want to simulate (e.g., network simulators)

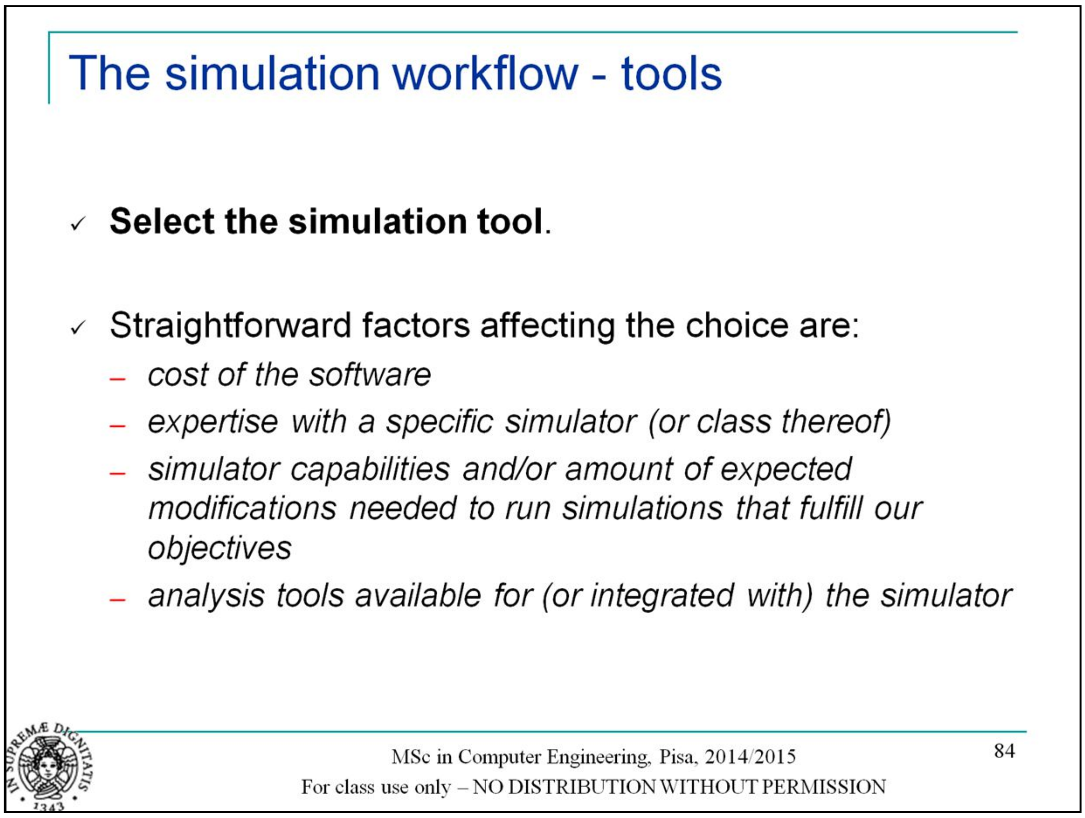
-Simulators that already include **many elements** of the model that you want to simulate, but need be **enhanced** to incorporate some of the functionalities that you require

> -Typical example: I invent a new congestion control algorithm for TCP. My life will probably be considerably easier if I select a simulator that already has all the models that I need (a router model, a link model, traffic generators, TCP Reno and Vegas), and limit myself to **writing down the code for my algorithm only**.

-Simulation-oriented programming languages provide users with simulation-oriented data structures (e.g., event queues) and functionalities

-There are simulation frameworks where you build up a simulator by connecting blocks (as in a block diagram), e.g., Simulink, Arena, etc.

-General-purpose languages, e.g. C++ or Java. These should be object-oriented, for fairly obvious reasons



The simulation workflow - tools

- ✓ **Select the simulation tool**.
- ✓ Straightforward factors affecting the choice are:
  - *cost of the software*
  - *expertise with a specific simulator (or class thereof)*
  - *simulator capabilities and/or amount of expected modifications needed to run simulations that fulfill our objectives*
  - *analysis tools available for (or integrated with) the simulator*

MSc in Computer Engineering, Pisa, 2014/2015 84
For class use only – NO DISTRIBUTION WITHOUT PERMISSION

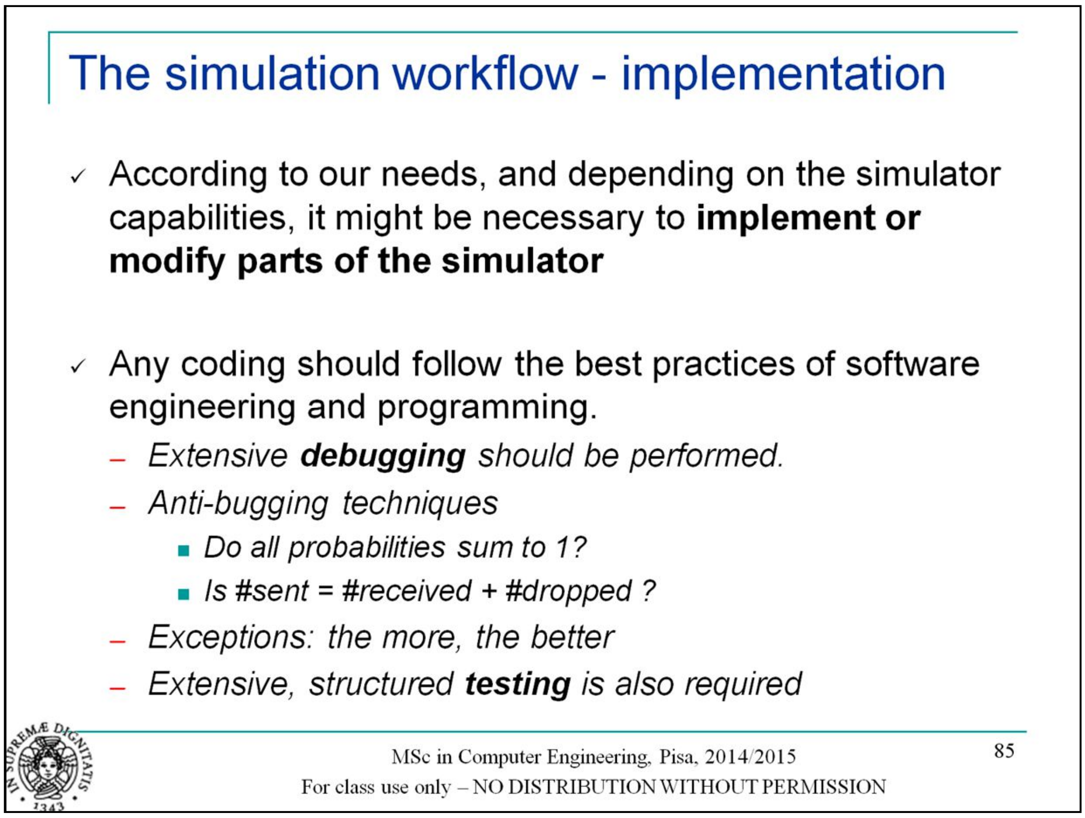
Even selecting a simulation tool is not an easy task. The choice should be made based on engineering principles and common sense.

The last item should not be neglected. A large portion of the time involved in a simulation analysis is spent in **analyzing the output data**.
Some simulators offer a varying degree of support to the **latter steps of the simulation workflow,** meaning that they can automate some of the latter steps (those that we will see at the end). If they do offer some support, this is going to save you a lot of time later on. If they don't, you should write that part of code yourself (or, god forbid, do the thing *manually*)

If you need to simulate (say) a **manufacturing plant**, it is unlikely that you will find a model in an existing simulation framework (say, Omnet++). However, if you **do** use that framework, you enjoy:

- Event queue routines
- Good RNGs
- Analysis and workflow automation tools
- (what's more important) **a programming paradigm**: the way you build objects and have them communicate follows a precise **rigid paradigm**. This is heaven, especially if the project is on a large scale and consists in the contributions of several hands.

If you don't go for a simulation framework, you have to do all the above on your own. It's time consuming, error-prone, and will probably lead you to a dead end.

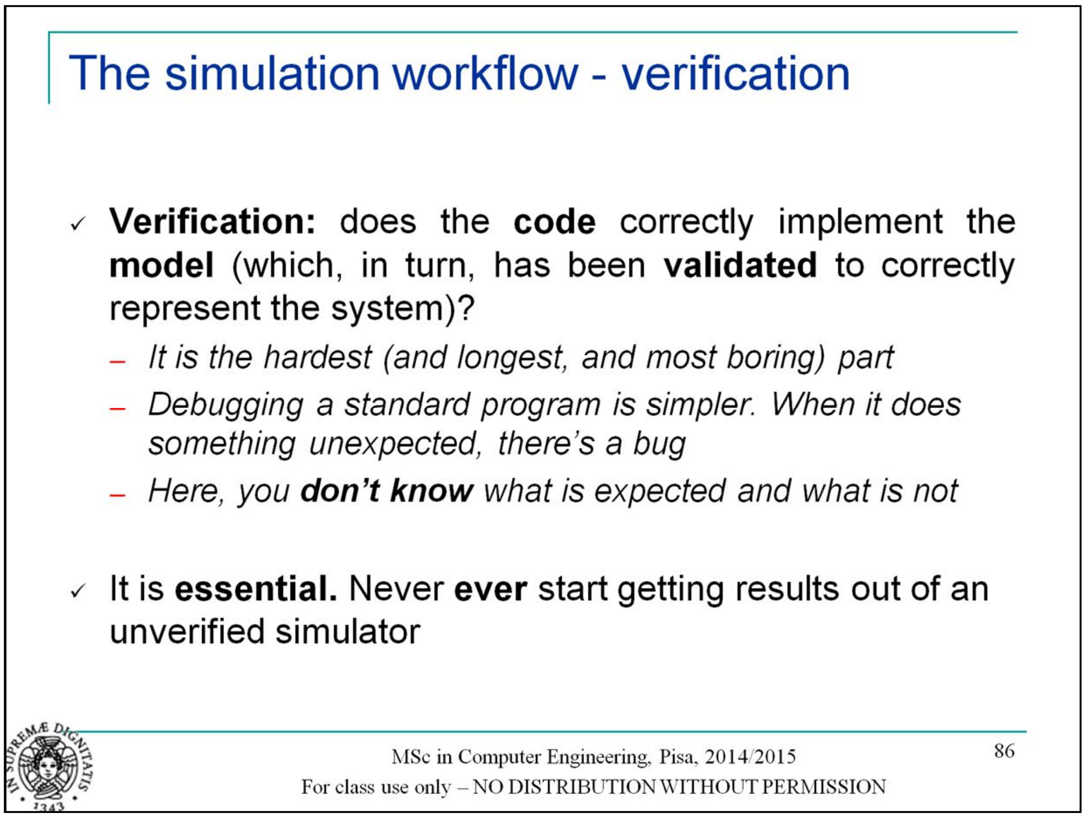

The simulation workflow - implementation

- ✓ According to our needs, and depending on the simulator capabilities, it might be necessary to **implement or modify parts of the simulator**
- ✓ Any coding should follow the best practices of software engineering and programming.
  - *Extensive* ***debugging*** *should be performed.*
  - *Anti-bugging techniques*
    - *Do all probabilities sum to 1?*
    - *Is #sent = #received + #dropped ?*
  - *Exceptions: the more, the better*
  - *Extensive, structured* ***testing*** *is also required*

MSc in Computer Engineering, Pisa, 2014/2015 85
For class use only – NO DISTRIBUTION WITHOUT PERMISSION

Coding must be state-of-the art, of course. Even more so in a simulator.

This is because, as we see in a minute, there is a lot more to ask from a simulator than just being bug-free.


The simulation workflow - verification

- ✓ **Verification:** does the **code** correctly implement the **model** (which, in turn, has been **validated** to correctly represent the system)?
  - *It is the hardest (and longest, and most boring) part*
  - *Debugging a standard program is simpler. When it does something unexpected, there's a bug*
  - *Here, you **don't know** what is expected and what is not*
- ✓ It is **essential.** Never **ever** start getting results out of an unverified simulator

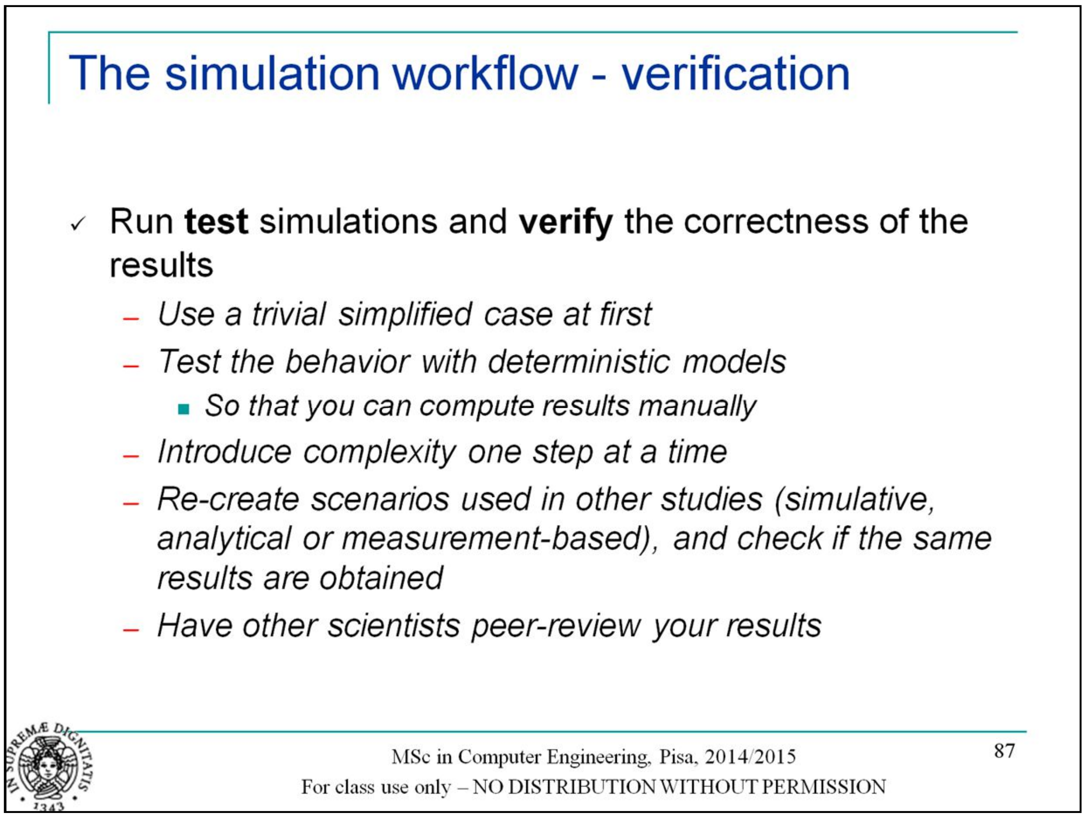
Most people believe that the code is ready when it **stops crashing**. This is obviously false.

-**Verification:** the code correctly implements the **model**. There are no "logical bugs", i.e. the model and the code yield the same result in any case

-**Validation** (of the model, through the code): the model is a correct abstraction of the real thing. If you have the real thing, you should obtain the same results in the real thing and as an output from the simulator.

However, you should always do a proper verification. Take into account that verification **absorbs a huge amount of time**.

Never **<u>ever</u>** start running simulations before terminating this part. You definitely **will find some "logical bugs",** and you will be forced to throw away everything that you have done so far.


The simulation workflow - verification

- ✓ Run **test** simulations and **verify** the correctness of the results
  - *Use a trivial simplified case at first*
  - *Test the behavior with deterministic models*
    - *So that you can compute results manually*
  - *Introduce complexity one step at a time*
  - *Re-create scenarios used in other studies (simulative, analytical or measurement-based), and check if the same results are obtained*
  - *Have other scientists peer-review your results*

MSc in Computer Engineering, Pisa, 2014/2015
For class use only – NO DISTRIBUTION WITHOUT PERMISSION
87

Part of the verification consists in running **simple scenarios**, of growing complexity, whose results can be **verified manually**.

If I have to test the new congestion control algorithm for TCP, I can start with a scenario with:

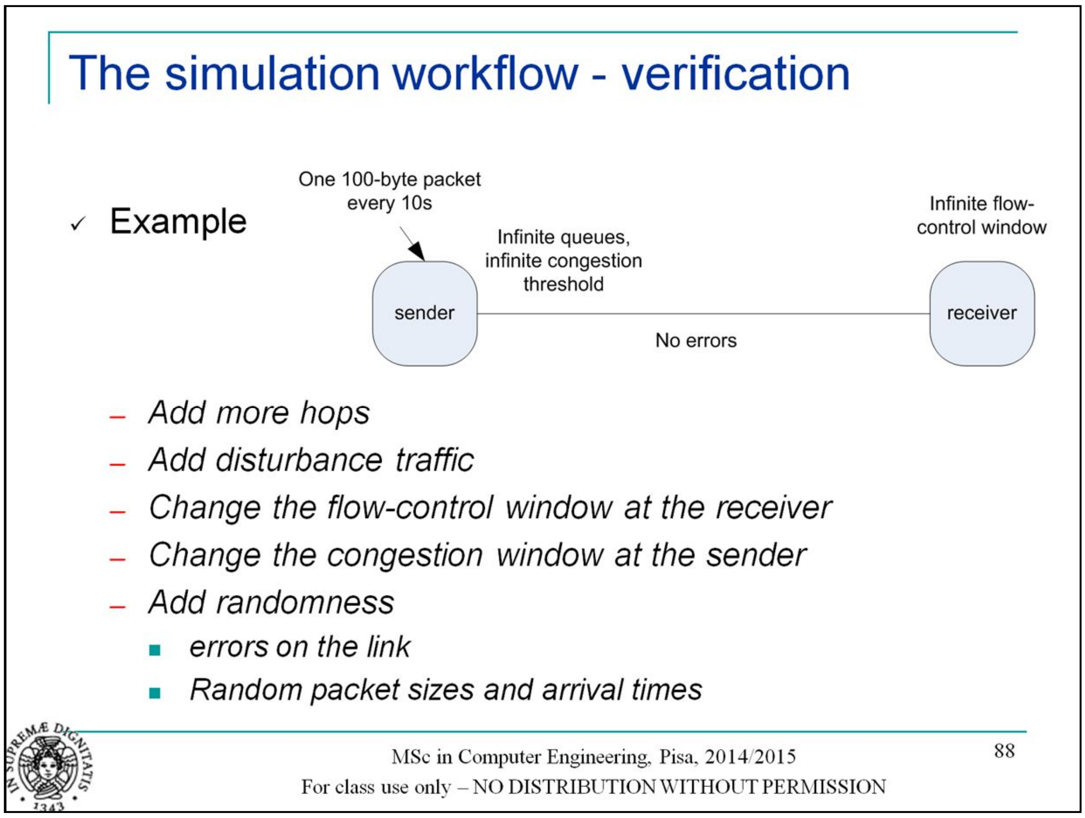
- Two nodes
- One link (with a null error probability)
- Just one application

In this scenario, I can do the computations **manually**. I can therefore verify the correctness of what the simulator does in that scenario. As the confidence in the correctness of the data increases, I can complicate the scenario at will

- Add more hops
- Add disturbance traffic
- Add an error probability on the link

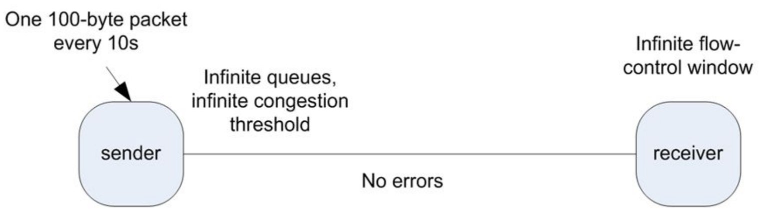
Possibly, not all at the same time (otherwise I wouldn't know where to look for problems if any do occur).

Sometimes verification can be done by comparing **your simulator results** with something else (e.g., some other simulator's, or the real system, if available).



The simulation workflow - verification

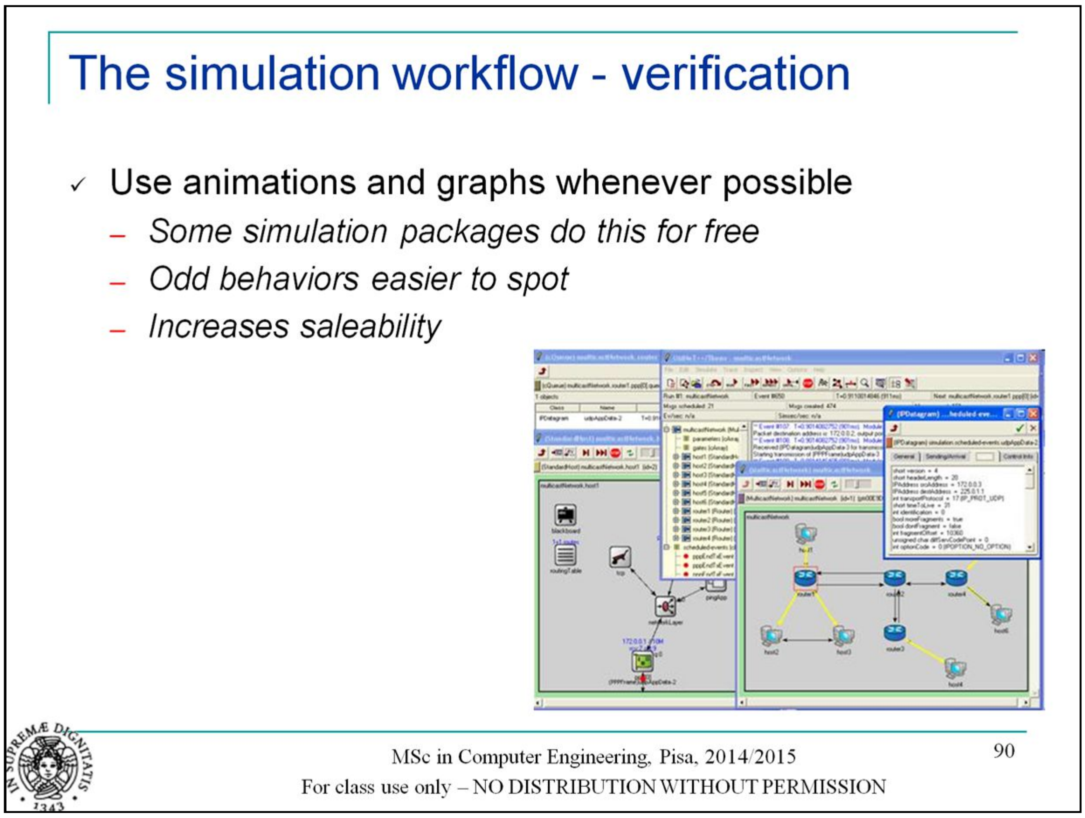
- ✓ Example


One 100-byte packet every 10s
Infinite queues, infinite congestion threshold
Infinite flow-control window
sender
receiver
No errors

- *Add more hops*
- *Add disturbance traffic*
- *Change the flow-control window at the receiver*
- *Change the congestion window at the sender*
- *Add randomness*
  - *errors on the link*
  - *Random packet sizes and arrival times*

The simulation workflow - verification

- ✓ Structured walk-through (rubber-duck debugging)
  - *Explain your code to someone else (possibly more than one person), line by line*
  - *Only step to the next line if everyone agrees on what's been done so far*
  - *If they can't understand what you're doing, there's probably a flaw in the code*
- ✓ Add and use **event traces** to see what your code actually does
  - *If an event trace has enough details, you can spot bugs by "simply" looking at it*
    - *Or, use gawk scripts to spot particular conditions*



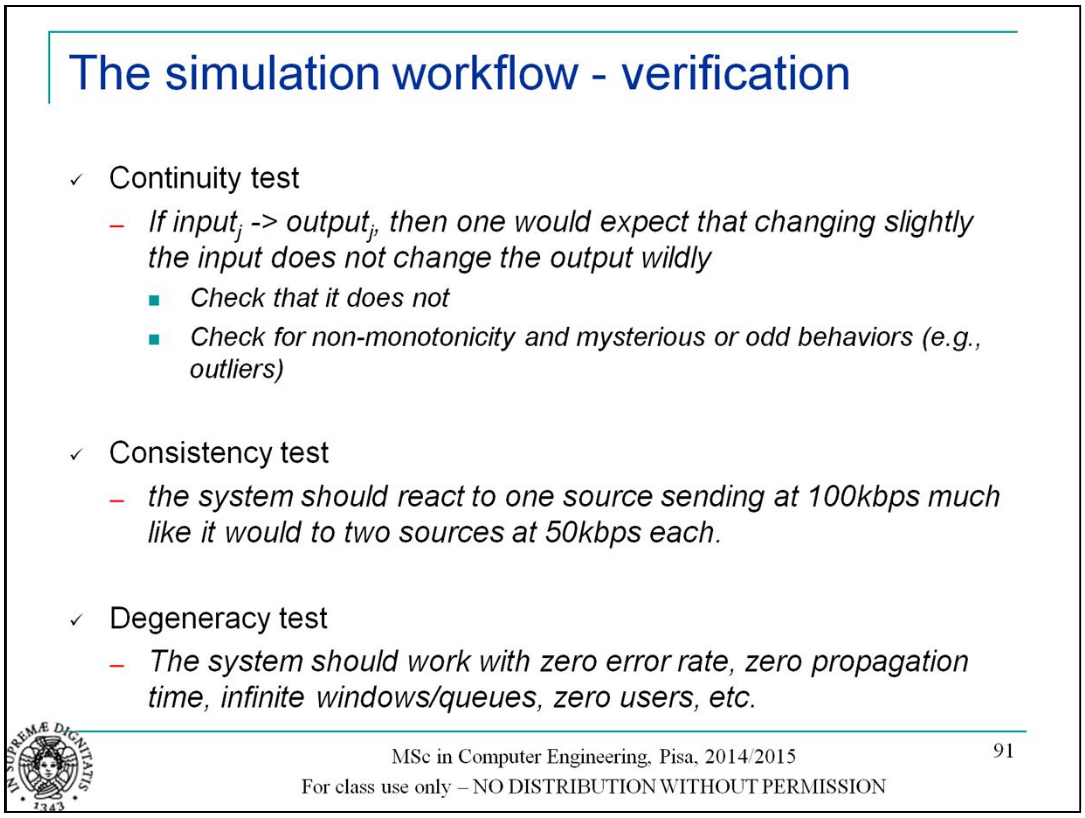
The simulation workflow - verification

- ✓ Use animations and graphs whenever possible
  - *Some simulation packages do this for free*
  - *Odd behaviors easier to spot*
  - *Increases saleability*

MSc in Computer Engineering, Pisa, 2014/2015

90

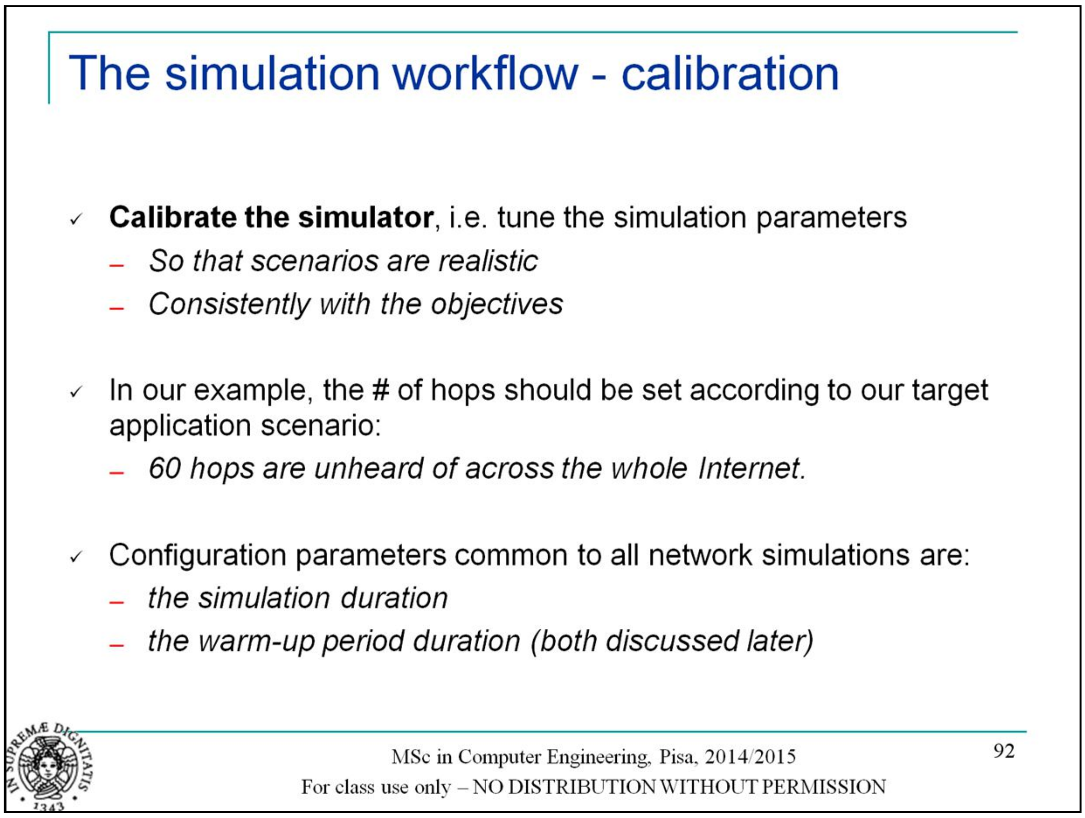
Recall that you're doing this for a **customer**. You have to impress her/him.


The simulation workflow - verification

- ✓ Continuity test
  - *If input$_j$ -> output$_j$, then one would expect that changing slightly the input does not change the output wildly*
    - *Check that it does not*
    - *Check for non-monotonicity and mysterious or odd behaviors (e.g., outliers)*
- ✓ Consistency test
  - *the system should react to one source sending at 100kbps much like it would to two sources at 50kbps each.*
- ✓ Degeneracy test
  - *The system should work with zero error rate, zero propagation time, infinite windows/queues, zero users, etc.*

For instance, if you change the link error probability by 1%, then you would not expect that the throughput would change dramatically. Check that it does not.

Most of the times you can find good continuity/consistency tests for your simulator by leveraging your experience of the system being simulated (or asking someone to confirm this)



The simulation workflow - calibration

- ✓ **Calibrate the simulator**, i.e. tune the simulation parameters
  - *So that scenarios are realistic*
  - *Consistently with the objectives*
- ✓ In our example, the # of hops should be set according to our target application scenario:
  - *60 hops are unheard of across the whole Internet.*
- ✓ Configuration parameters common to all network simulations are:
  - *the simulation duration*
  - *the warm-up period duration (both discussed later)*

MSc in Computer Engineering, Pisa, 2014/2015 92
For class use only – NO DISTRIBUTION WITHOUT PERMISSION

In a simulator there are tuning knobs, or – better yet – **configuration variables** which should be given a value. Some of these are **factors** of the simulation, i.e. those that you want to change in the analysis. Others are simply **configuration parameters**, which you will keep constant.

The set of all these represents the **simulation scenario.**

Scenarios must be **carefully engineered**. They should mirror operating conditions which are **real-life-like,** and **apt to verify your claim,** i.e., consistent with your objectives.

Regarding **realism**:

-It makes no sense to test TCP on 60 hops. Paths are never that long (30 hops get you to New Zealand). The number of hops in the Internet is bound to **decrease** over time (since connectivity is on the rise). The fact that my congestion control algorithm works well/fails on paths with more than 50 hops is **totally irrelevant** to its marketability.

-It makes no sense to test TCP using applications that **send 1 byte/day**. No applications do this, and those that do never congestion links.

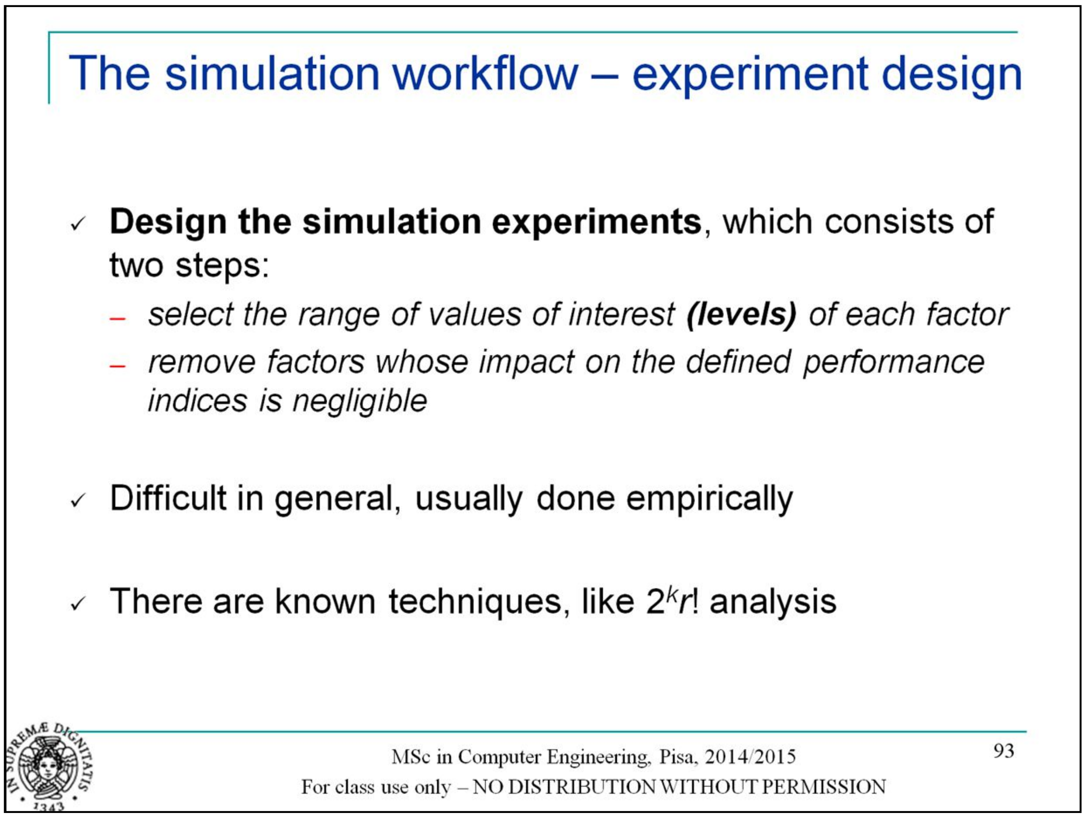
Regarding **consistency with you objectives**:

-It is still a good laugh (among scientists) to recall that (see Camp, 2004) ad hoc network routing protocols have been tested for years on a network scenario where the average number of hops between 2 endpoints was 1.2. This means that more than 80% of the times the sender and receivers are directly connected, hence no routing takes place.

There are, in particular, two parameters that have to be considered **very carefully**, since they make a large part of the (in)credibility of your simulation study

-The simulation duration

-The warm-up time



The simulation workflow – experiment design

- ✓ **Design the simulation experiments**, which consists of two steps:
  - *select the range of values of interest* ***(levels)*** *of each factor*
  - *remove factors whose impact on the defined performance indices is negligible*
- ✓ Difficult in general, usually done empirically
- ✓ There are known techniques, like $2^k r$! analysis

MSc in Computer Engineering, Pisa, 2014/2015
For class use only – NO DISTRIBUTION WITHOUT PERMISSION

93

You need to decide the **range of values** for each factor.

Note that (by the basic principle of counting), if you have two factors, with n1 and n2 values each, you would need n1*n2 experiments to test all the possible combinations. This is clearly impossible when the number of factors and values grows beyond a few units, since the number of experiments grows **exponentially**.

On the other hand, you should be aware that factors may **interplay in strange ways**, so that very peculiar outcomes may be produced when two or more factors have a certain relationship (which you don't know in advance, of course).

For instance, I might vary the factors as follows:

-# of hops, from 2 to 30

-Link bandwidth, from 1Mbps to 1Gbps (possibly logarithmically)

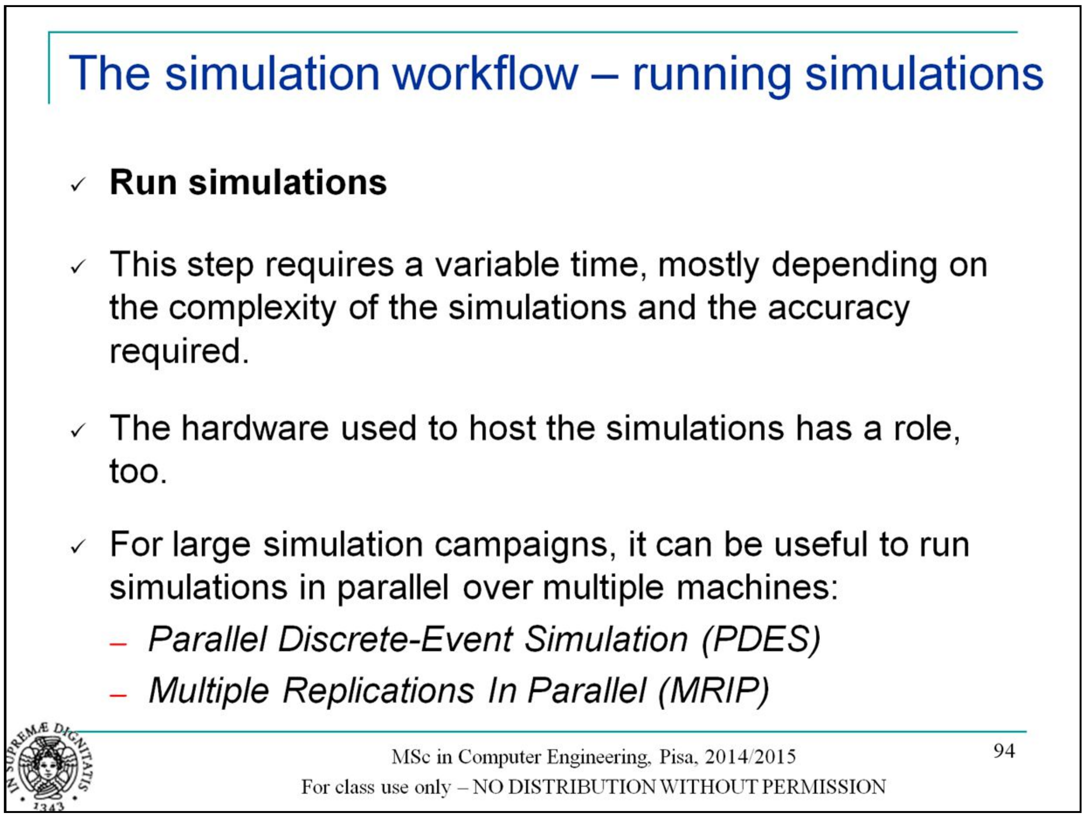
-Bit error rate, from 10^-10 to 10^-3 (again, possibly logarithmically)

-Buffer space on the nodes, say from 4kbyte to 1Mbyte

I can't state in advance whether the bandwidth plays a different role in conjunction with large/small buffers.

This puts a simulator **user** in a very uncomfortable position. There are techniques to reduce the number of factors. One is **2^kr! analysis**, which tells you at a reasonable cost which factors have the highest impact on the performance.

Other than this, you can only use your expertise on the system.


The simulation workflow – running simulations

- ✓ **Run simulations**
- ✓ This step requires a variable time, mostly depending on the complexity of the simulations and the accuracy required.
- ✓ The hardware used to host the simulations has a role, too.
- ✓ For large simulation campaigns, it can be useful to run simulations in parallel over multiple machines:
  - *Parallel Discrete-Event Simulation (PDES)*
  - *Multiple Replications In Parallel (MRIP)*

MSc in Computer Engineering, Pisa, 2014/2015 94
For class use only – NO DISTRIBUTION WITHOUT PERMISSION

Running simulations **takes time** (and a lot of it, in many cases). For this reason, **think before running a simulation**.

- Is this the **one scenario** that I want to test?
- Are the parameters **set to the right values**? Are they **mutually consistent?**

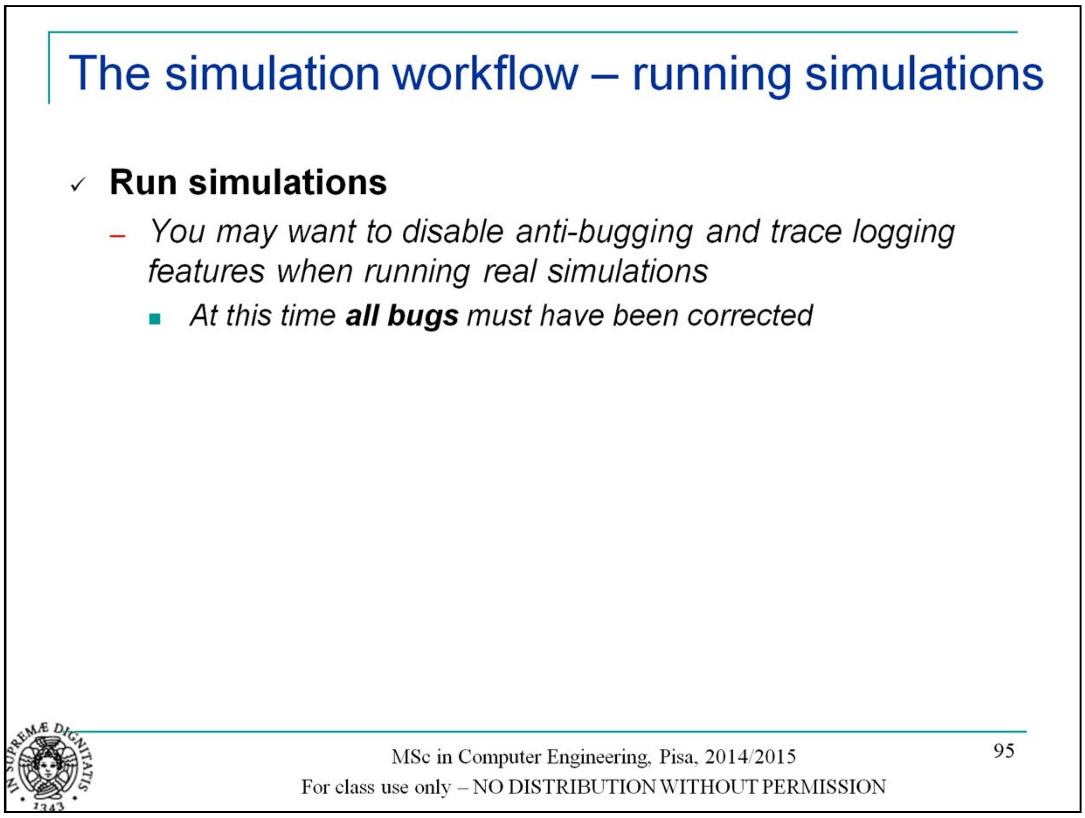
It happens all too often that some parameters has been set to one value or another without giving it much thought, and this choice invalidates days and days of work.

Simulation can be hardware-hungry. Normally **memory** plays a major role (because data structures, especially packets, use up a lot of memory), and disk swaps should be avoided if possible.
Multicore processors can be used to distribute the load.

Now, if you want to speed up your simulation, you have two alternatives:

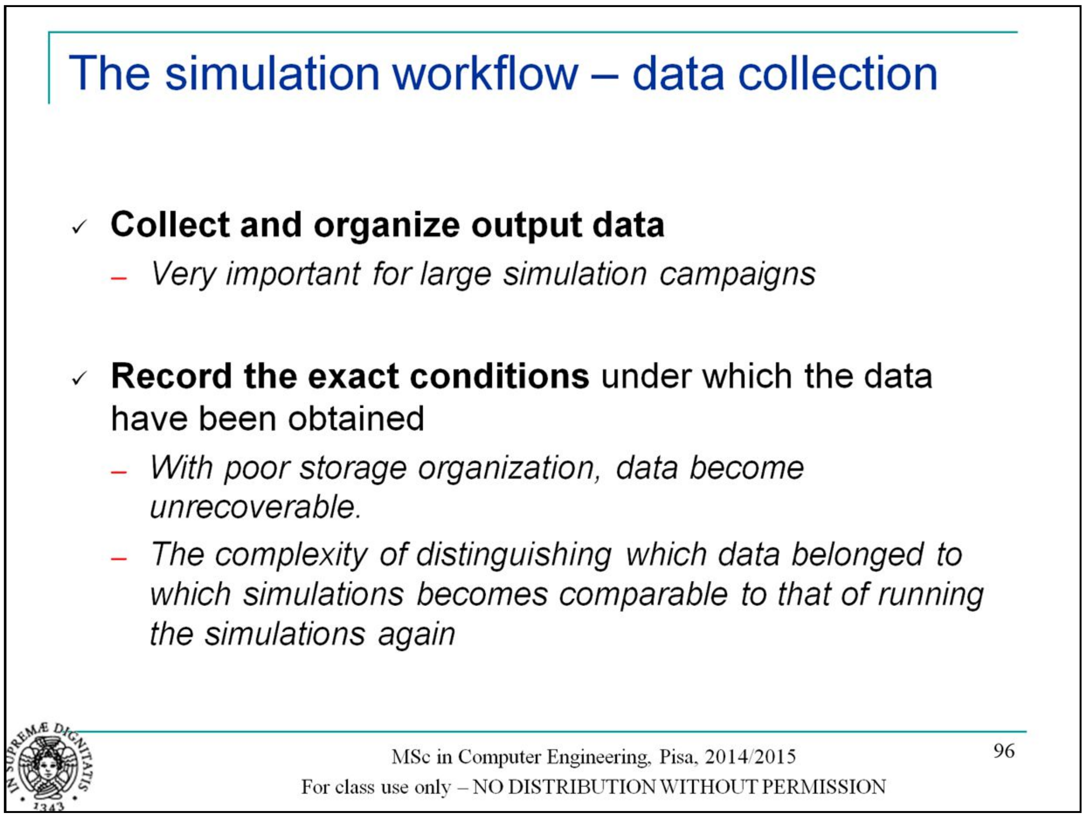
- **Go parallel**: this is, however, a **design choice**. If you have designed your simulator to be parallel, you can distribute it on several cores/machines and harvest some (sublinear) speedup
- Distribute **several scenarios** (or, better yet, **replicas** of a scenario) among several PCs (MRIP) (or several cores of your processor).

Of course, you should not be doing this **manually**, but you need a **simulation automation software** that does this for you.



The simulation workflow – running simulations

- ✓ **Run simulations**
  - – *You may want to disable anti-bugging and trace logging features when running real simulations*
    - ■ *At this time **all bugs** must have been corrected*



The simulation workflow – data collection

- ✓ **Collect and organize output data**
  - *Very important for large simulation campaigns*
- ✓ **Record the exact conditions** under which the data have been obtained
  - *With poor storage organization, data become unrecoverable.*
  - *The complexity of distinguishing which data belonged to which simulations becomes comparable to that of running the simulations again*

When you simulate **in the large**, before you start producing results, you need to tackle the problem of **storing the results**.

It is quite common to produce hundreds of MB of data, and a **file system is not up to the task**. More specifically, a file system does not offer you any support to **associate a scenario file** to an **output file** that contains your metrics. If you rely on the file system alone, you have to do the association **manually**, which is **utterly error-prone**:

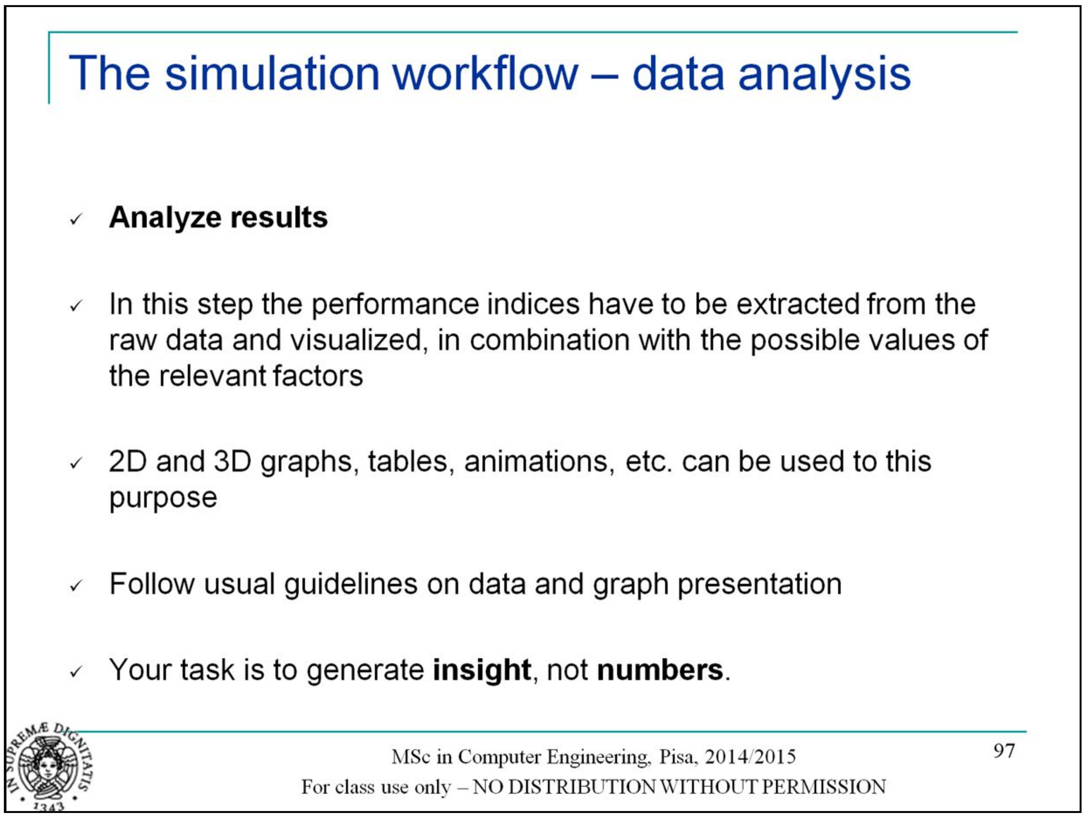
-You need to create hundreds of folders

-Replicate the right scenario file in each of the folders

-Enforce *manually* a hierarchy on the factors (since the file system *is* structured hierarchically)

The correct solution is to **use a relational database** to store **both the scenarios and the results**.

- Or use a simulation environment that helps you with this step.



The simulation workflow – data analysis

- ✓ **Analyze results**
- ✓ In this step the performance indices have to be extracted from the raw data and visualized, in combination with the possible values of the relevant factors
- ✓ 2D and 3D graphs, tables, animations, etc. can be used to this purpose
- ✓ Follow usual guidelines on data and graph presentation
- ✓ Your task is to generate **insight**, not **numbers**.

MSc in Computer Engineering, Pisa, 2014/2015 97

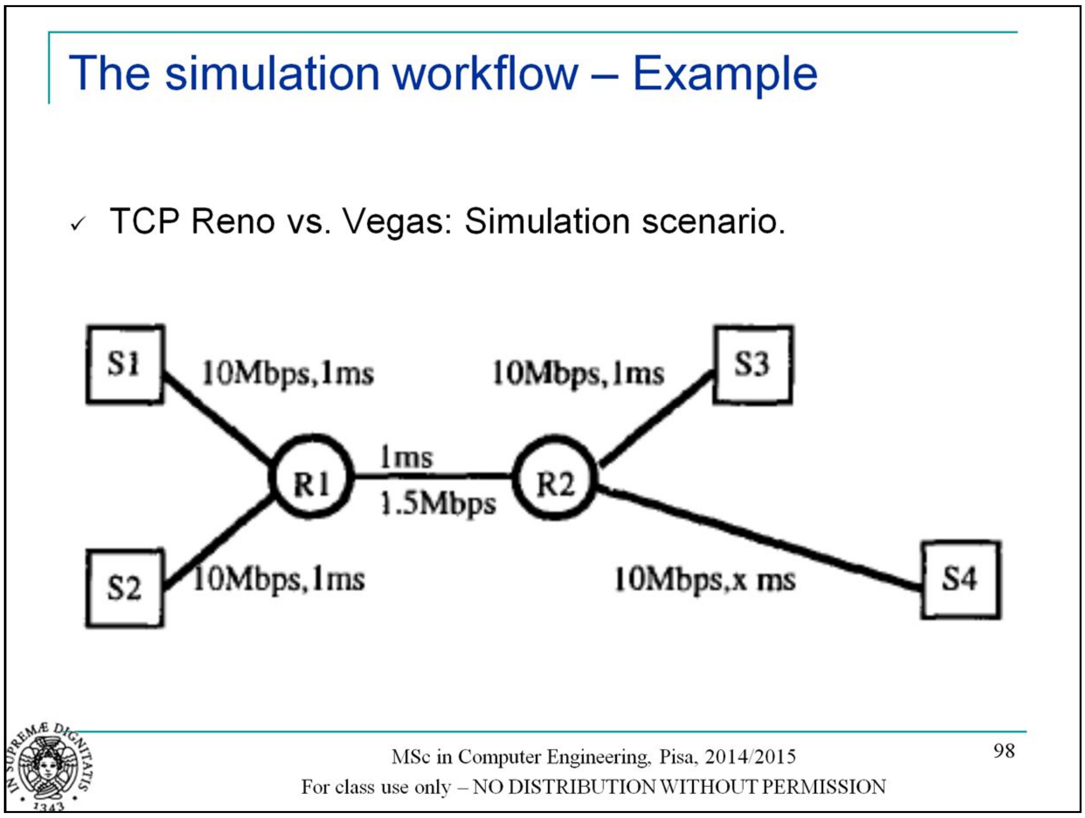
For class use only – NO DISTRIBUTION WITHOUT PERMISSION

The **raw results** need be analyzed to produce something usable: **graphs, tables, animations,** etc.

It is up to you to locate the **result that better shows your research claim**. For instance, when you compare your invention against something existing, a good practice is to test your invention in the **most unfavorable conditions**. If it outperforms the competition in that scenario, it will do the same in all others.

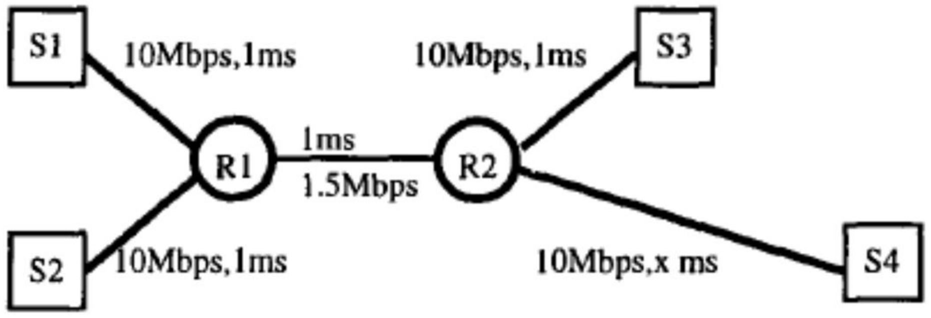
You should rely on the **quality** of the information you produce, not on the **quantity**. When doing a presentation, no one will pay attention to more than 3-4 graphs.

It is not necessary to show **how much work** you have done, e.g. showing all the graphs even when they are meaningless *just because you took the pain to draw them*.



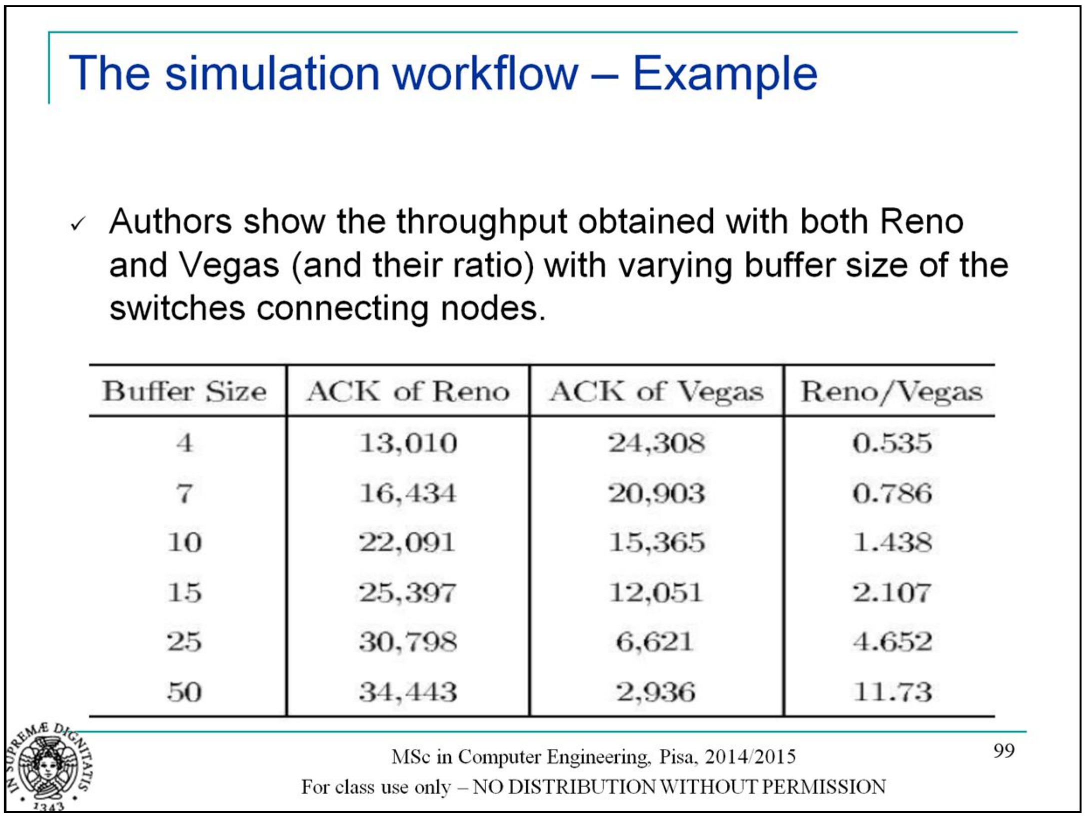
The simulation workflow – Example

- ✓ TCP Reno vs. Vegas: Simulation scenario.



Analysis and comparison of TCP Reno and Vegas
Mo, J. La, R.J. Anantharam, V. Walrand, J.
Dept. of Electr. Eng. & Comput. Sci., California Univ., Berkeley, CA;
This paper appears in: INFOCOM '99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE
Publication Date: 21-25 Mar 1999
Volume: 3, On page(s): 1556-1563 vol.3

One of the two sources sends packets using TCP Reno, the other one uses TCP Vegas. The link in the middle acts as a bottleneck. Authors want to show the throughput of the two connections when the buffer on link R1-R2 varies.



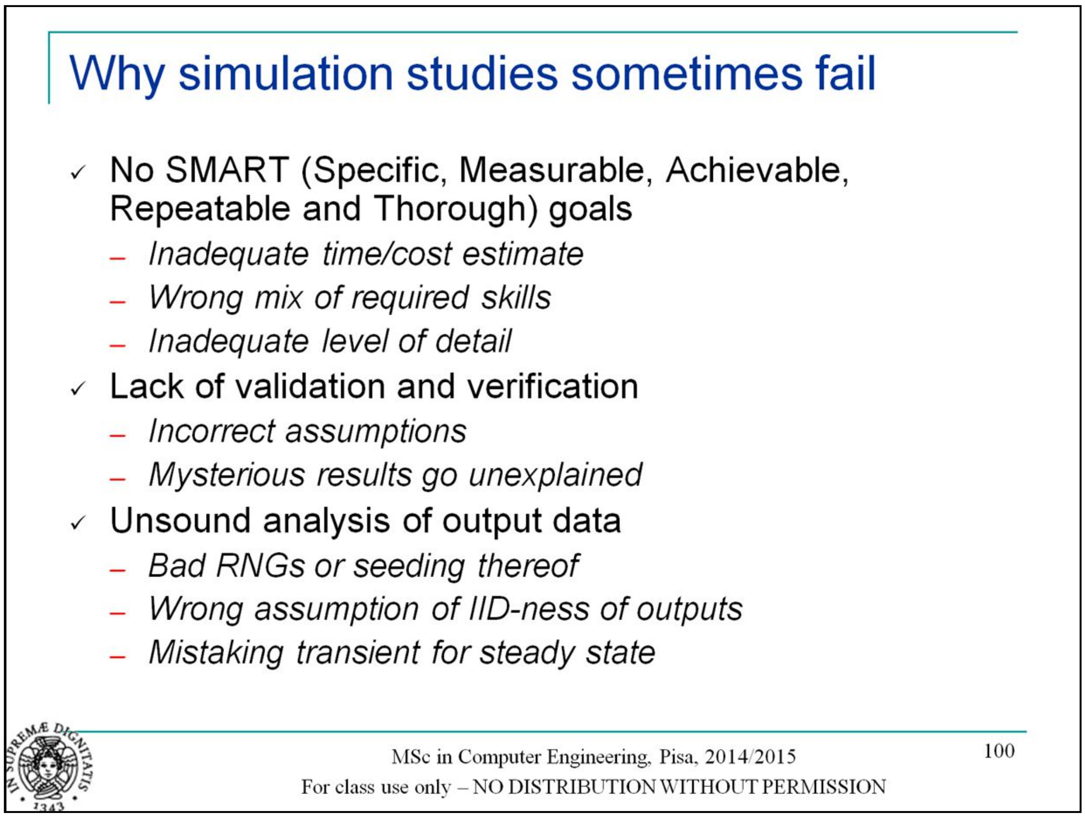
The simulation workflow – Example

- ✓ Authors show the throughput obtained with both Reno and Vegas (and their ratio) with varying buffer size of the switches connecting nodes.

Buffer Size	ACK of Reno	ACK of Vegas	Reno/Vegas
4	13,010	24,308	0.535
7	16,434	20,903	0.786
10	22,091	15,365	1.438
15	25,397	12,051	2.107
25	30,798	6,621	4.652
50	34,443	2,936	11.73

The buffer size is in packets, not bytes.

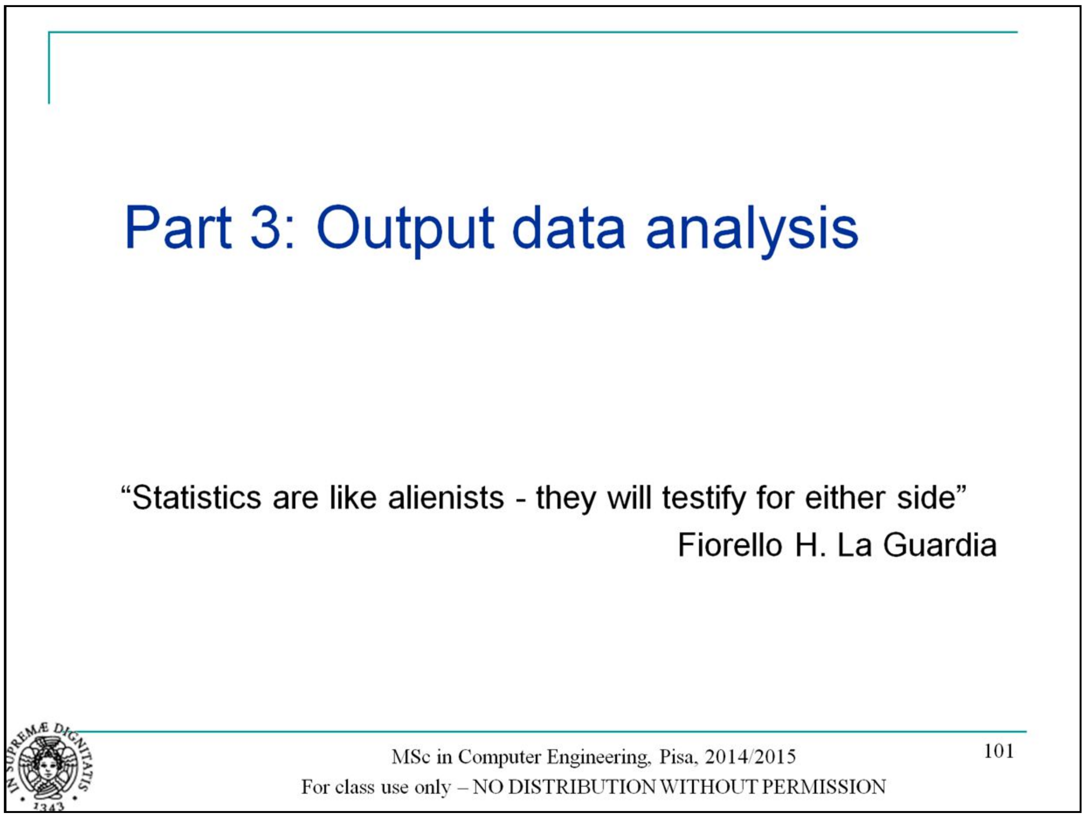
When the buffer size is small (too small), Vegas outperforms Reno. However, for realistic buffer sizes, TCP Reno takes all the bandwidth, and Vegas is left with a much smaller share.



Why simulation studies sometimes fail

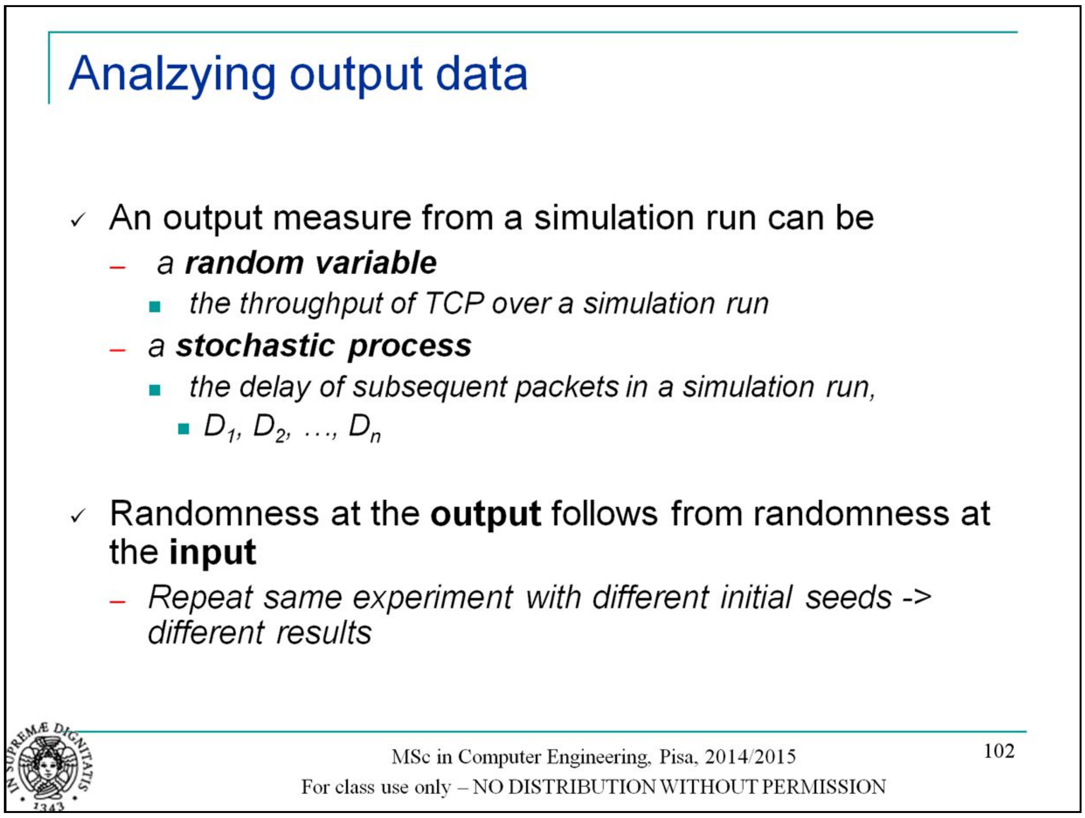
- ✓ No SMART (Specific, Measurable, Achievable, Repeatable and Thorough) goals
  - *Inadequate time/cost estimate*
  - *Wrong mix of required skills*
  - *Inadequate level of detail*
- ✓ Lack of validation and verification
  - *Incorrect assumptions*
  - *Mysterious results go unexplained*
- ✓ Unsound analysis of output data
  - *Bad RNGs or seeding thereof*
  - *Wrong assumption of IID-ness of outputs*
  - *Mistaking transient for steady state*

By "fail" we mean that we get **meaningless** results (or no result at all).



Part 3: Output data analysis

"Statistics are like alienists - they will testify for either side"
Fiorello H. La Guardia



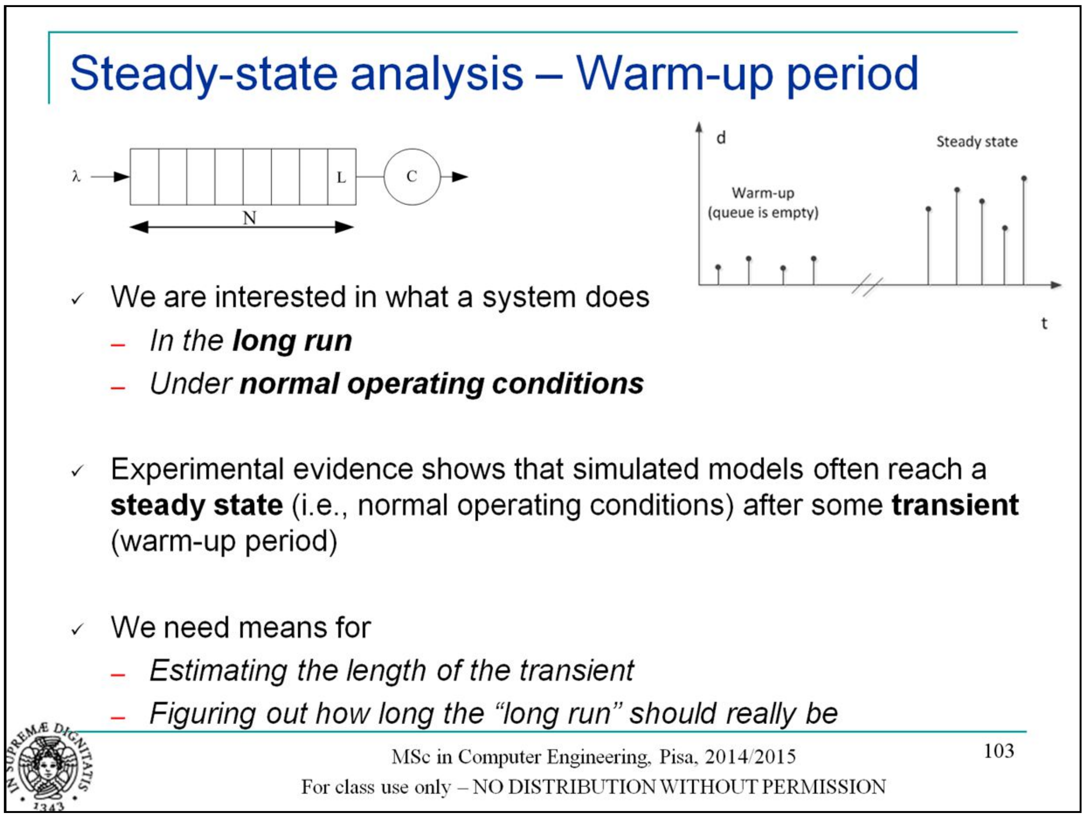
Analzying output data

- ✓ An output measure from a simulation run can be
  - a ***random variable***
    - *the throughput of TCP over a simulation run*
  - *a* ***stochastic process***
    - *the delay of subsequent packets in a simulation run,*
    - $D_1, D_2, \ldots, D_n$
- ✓ Randomness at the **output** follows from randomness at the **input**
  - *Repeat same experiment with different initial seeds -> different results*

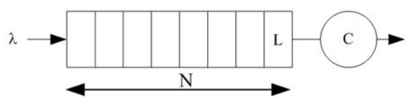
MSc in Computer Engineering, Pisa, 2014/2015
For class use only – NO DISTRIBUTION WITHOUT PERMISSION
102

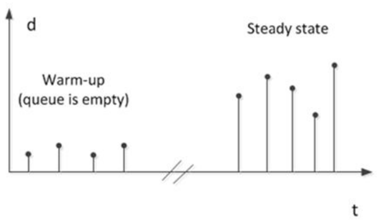
What you obtain from a simulation is – almost always – random.

It can be a single variable (first case), or a **process** (second case, which includes the first one).



Steady-state analysis – Warm-up period





- ✓ We are interested in what a system does
  - *In the **long run***
  - *Under **normal operating conditions***
- ✓ Experimental evidence shows that simulated models often reach a **steady state** (i.e., normal operating conditions) after some **transient** (warm-up period)
- ✓ We need means for
  - *Estimating the length of the transient*
  - *Figuring out how long the "long run" should really be*

MSc in Computer Engineering, Pisa, 2014/2015 103
For class use only – NO DISTRIBUTION WITHOUT PERMISSION

Models often reach a steady-state. **Systems** seldom do in practice.

This is because the conditions that you set at the beginning of the simulation (and assume true for the whole duration of the simulation) do change in reality. For instance, the number of hops in a TCP connection is bound to change because routing is dynamic in the Internet. In the **model,** you can fix it, whereas in **reality** it does change.

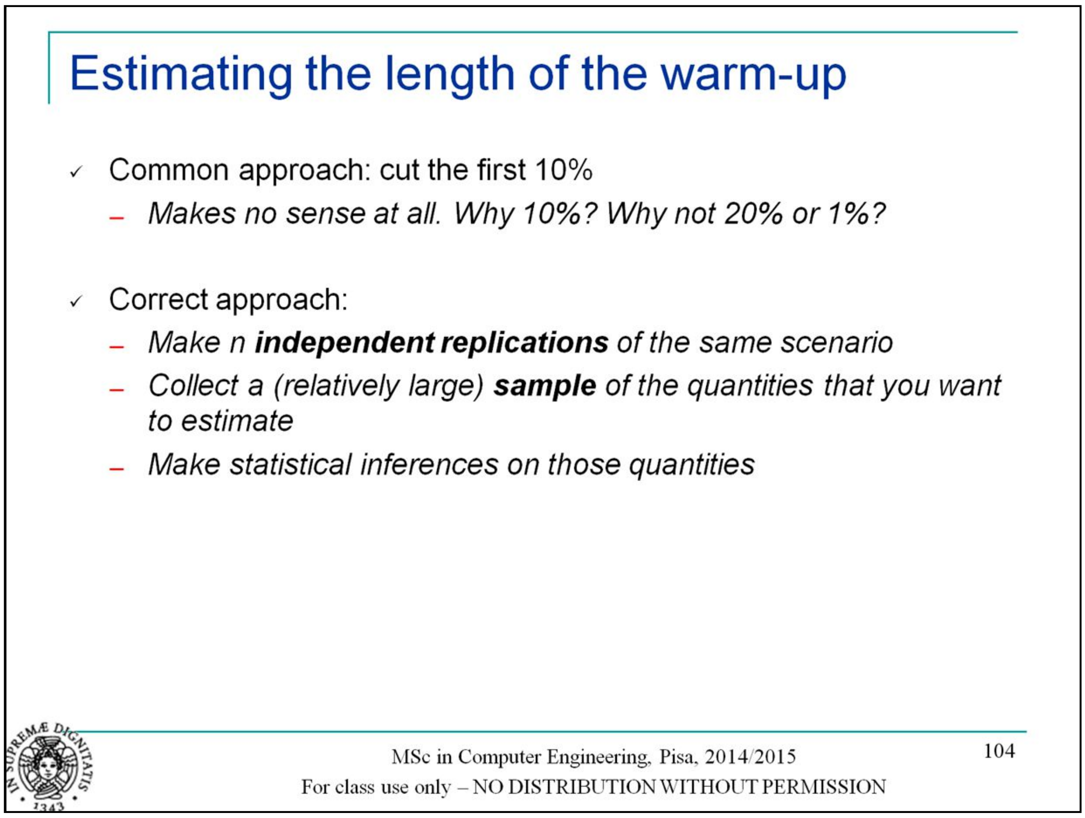
However, we are most often interested in the **steady-state behavior** of a system, i.e. the behavior that it reaches once the transient phase is extinguished.

If you start with an empty queue+server, the first few packets will have **a very low delay** and will not be dropped, which may not be representative of what a packet can expect when arriving after, say, 1000s from the start.

This poses the problem of **estimating the length of the warm-up phase**, because you should **avoid taking samples** before the system is in a steady-state, lest you set a **bias** on your statistics.

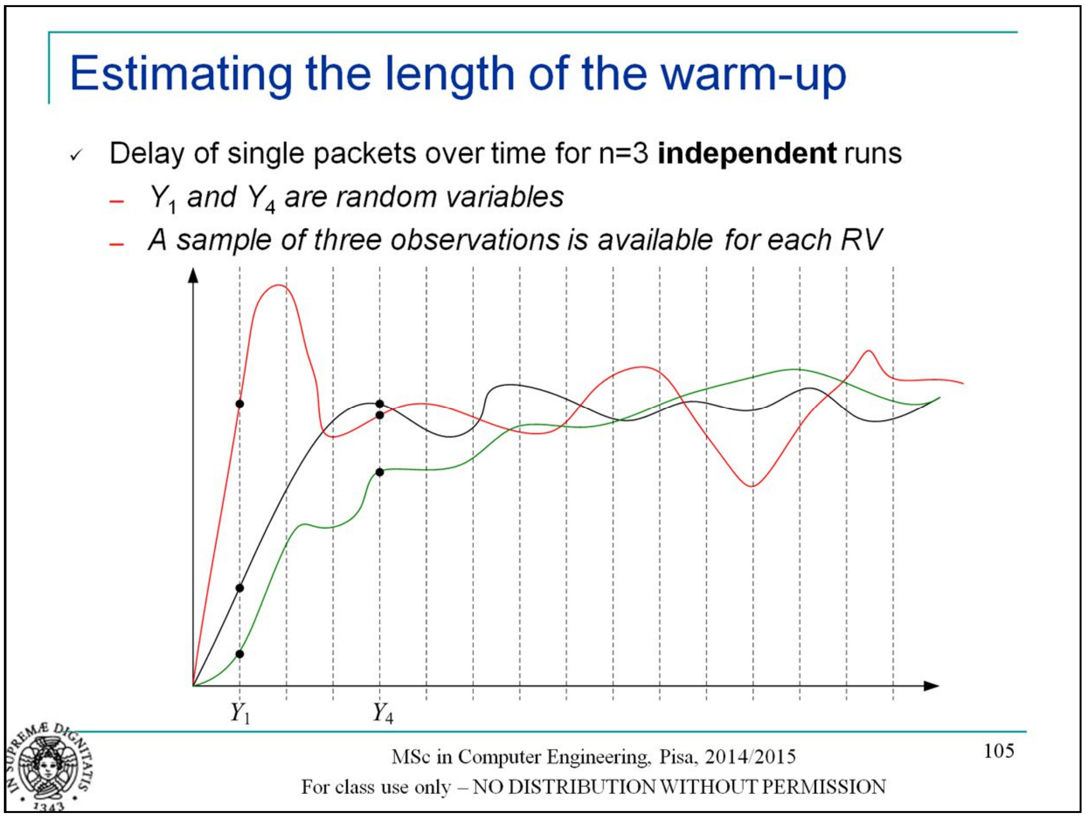
You can do this, as usual

- Using some **common sense**: you may know something about the **time constants of the system**. If you know that all your traffic sources activate in the first 10 seconds, and that they send 100 packets/second on average, it is reasonable to think that your system will be in a steady state starting from 12-13.
- Using **statistical tests**: you can take a moving average of some quantities (over a time window), and see when they stabilize around a constant value.



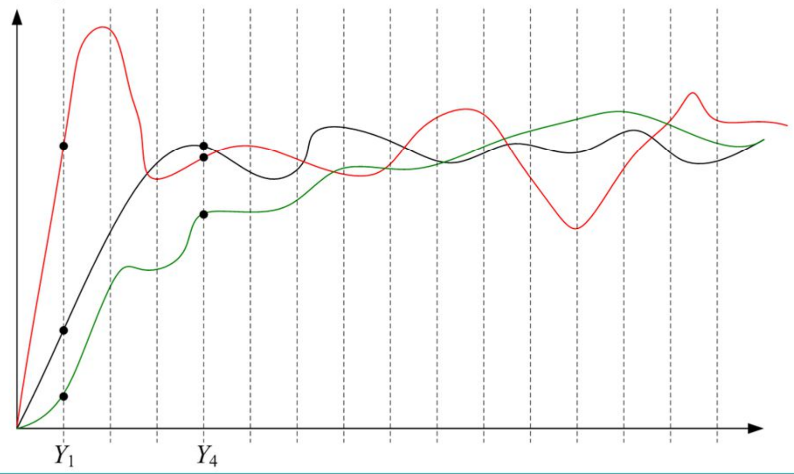
Estimating the length of the warm-up

- ✓ Common approach: cut the first 10%
  - *Makes no sense at all. Why 10%? Why not 20% or 1%?*
- ✓ Correct approach:
  - *Make n* ***independent replications*** *of the same scenario*
  - *Collect a (relatively large)* ***sample*** *of the quantities that you want to estimate*
  - *Make statistical inferences on those quantities*



Estimating the length of the warm-up

- ✓ Delay of single packets over time for n=3 **independent** runs
  - – *Y_1 and Y_4 are random variables*
  - – *A sample of three observations is available for each RV*

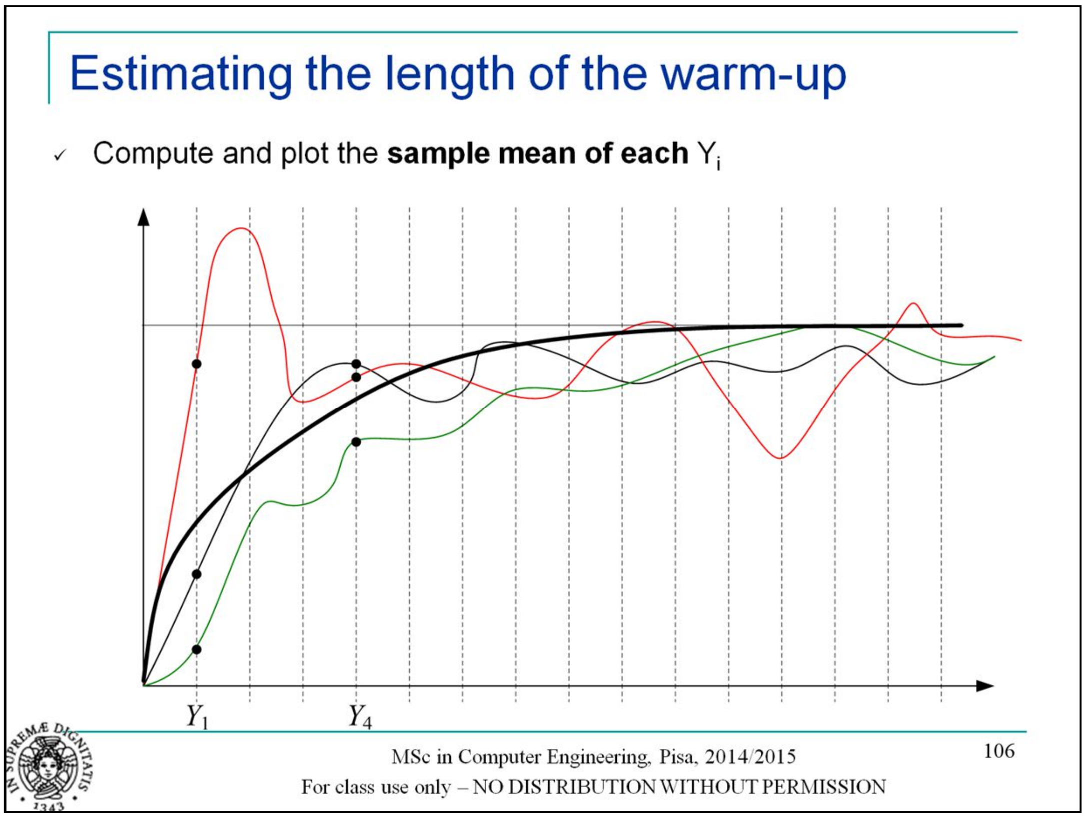


If you use **different seeds for the random generators**, then you make **independent replicas** of the same scenario, and this is the statistically sound way to make things.

If you consider the delay of the j-th packet in each replica (the same across all replicas), this is a random variable, whose distribution you can **estimate** using statistical techniques.

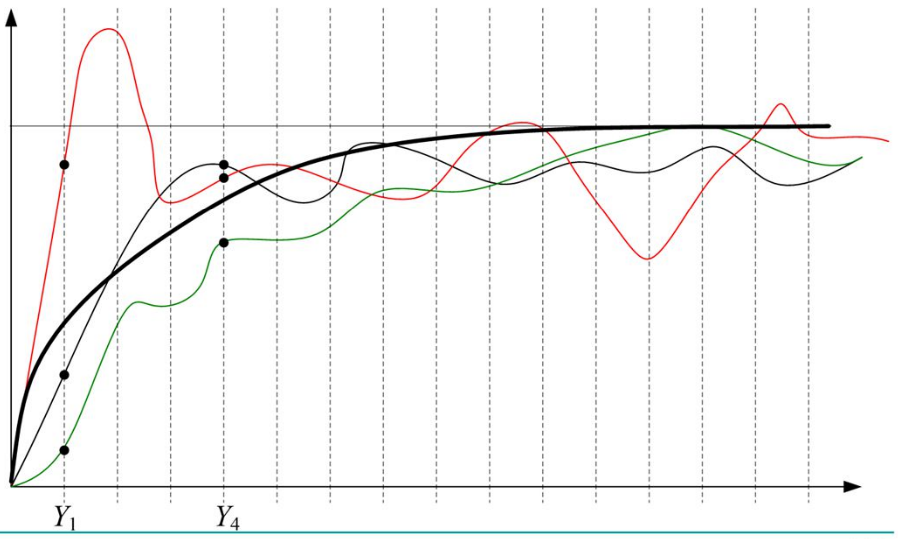
If, starting from a certain packet *j*, the distribution (or, at least, the **mean**) stabilizes to a constant value (or around a constant value), then we can say that the system is in a steady state. You should find this value *j*, and discard everything that happens earlier.

Note that a reasonable number of observations is around 10 (three is way too small).



Estimating the length of the warm-up

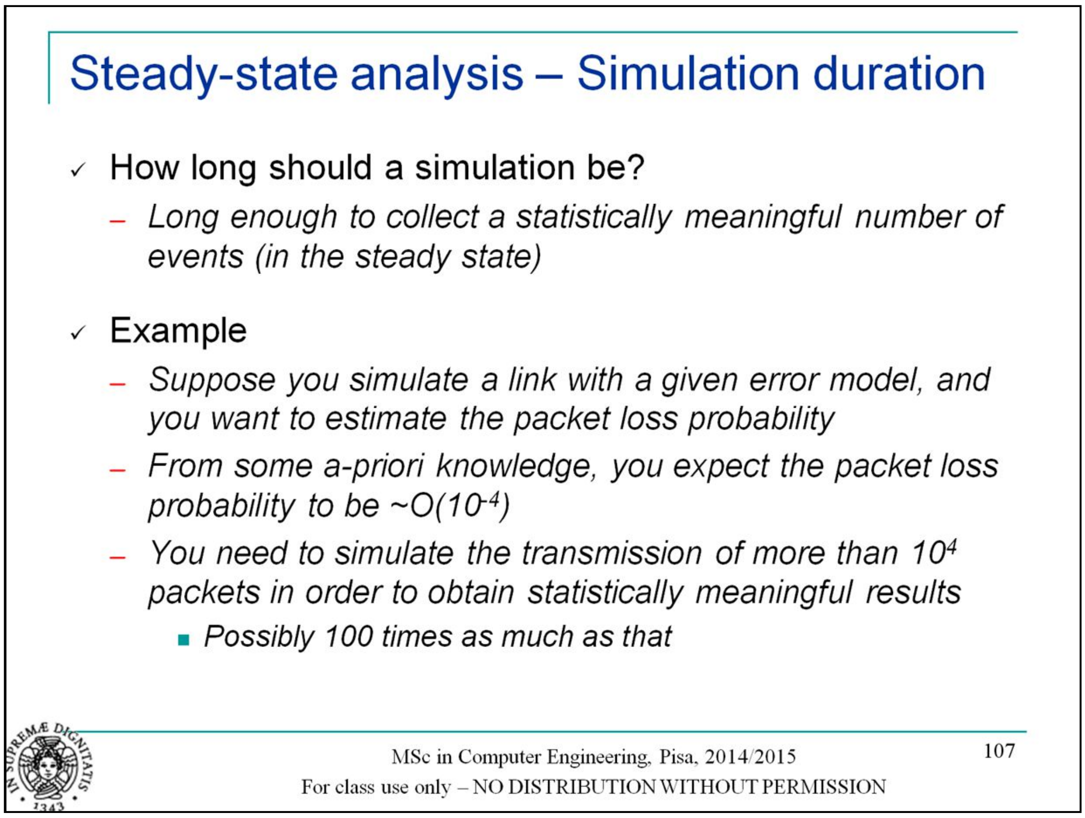
- ✓ Compute and plot the **sample mean of each** Y_i



MSc in Computer Engineering, Pisa, 2014/2015 106
For class use only – NO DISTRIBUTION WITHOUT PERMISSION

In many cases, the graph of the **sample mean** converges around some value (possibly fluctuating around it). When it does, the system is in a steady state.

If the sample mean of the delays (of the same packet across independent replications) is too **jittery**, then you may want to take a **moving average** (over the packet index *j*) of the latter, so as to smooth the variations. If the thing still does not converge, try to **increase the sample width** (i.e., of independent replicas).

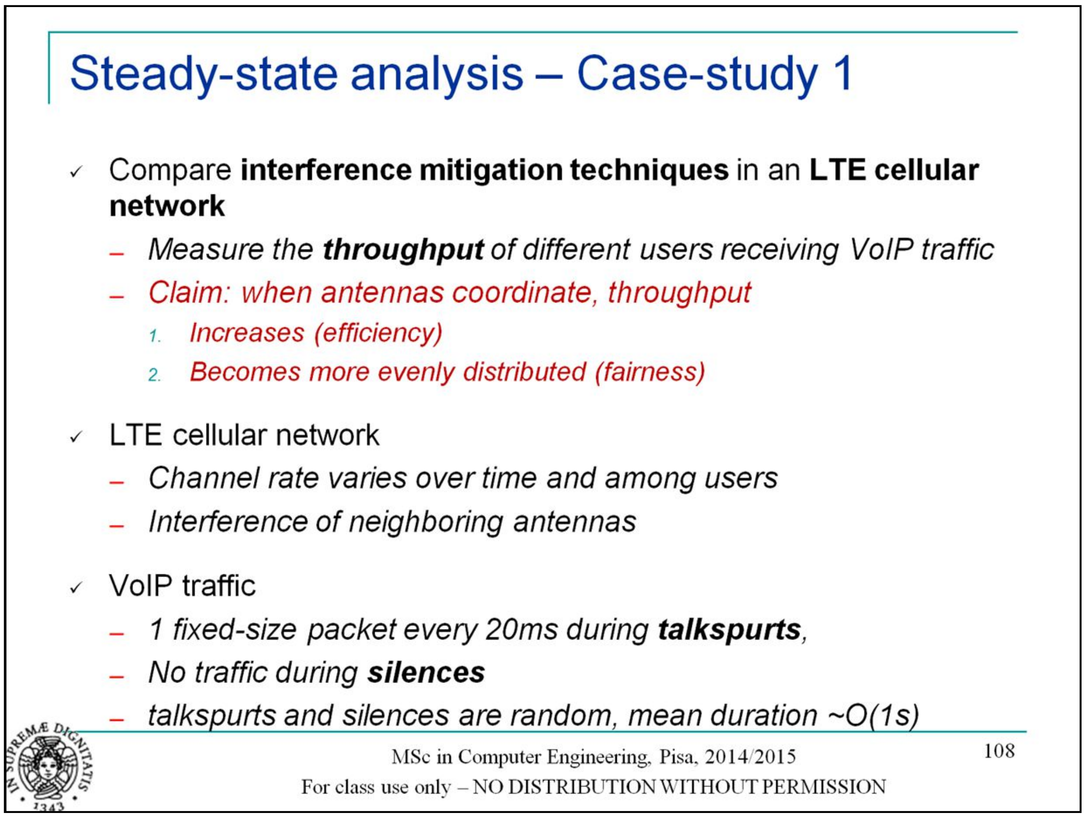


Steady-state analysis – Simulation duration

- How long should a simulation be?
  - *Long enough to collect a statistically meaningful number of events (in the steady state)*
- Example
  - *Suppose you simulate a link with a given error model, and you want to estimate the packet loss probability*
  - *From some a-priori knowledge, you expect the packet loss probability to be ~O(10^{-4})*
  - *You need to simulate the transmission of more than 10^4 packets in order to obtain statistically meaningful results*
    - *Possibly 100 times as much as that*

MSc in Computer Engineering, Pisa, 2014/2015 107
For class use only – NO DISTRIBUTION WITHOUT PERMISSION

This is a bit simplistic, of course. We do know more advanced techniques (e.g., related to the confidence intervals for Bernoulli experiments).



Steady-state analysis – Case-study 1

- ✓ Compare **interference mitigation techniques** in an **LTE cellular network**
  - *Measure the **throughput** of different users receiving VoIP traffic*
  - *Claim: when antennas coordinate, throughput*
    1. *Increases (efficiency)*
    2. *Becomes more evenly distributed (fairness)*
- ✓ LTE cellular network
  - *Channel rate varies over time and among users*
  - *Interference of neighboring antennas*
- ✓ VoIP traffic
  - *1 fixed-size packet every 20ms during **talkspurts**,*
  - *No traffic during **silences***
  - *talkspurts and silences are random, mean duration ~O(1s)*

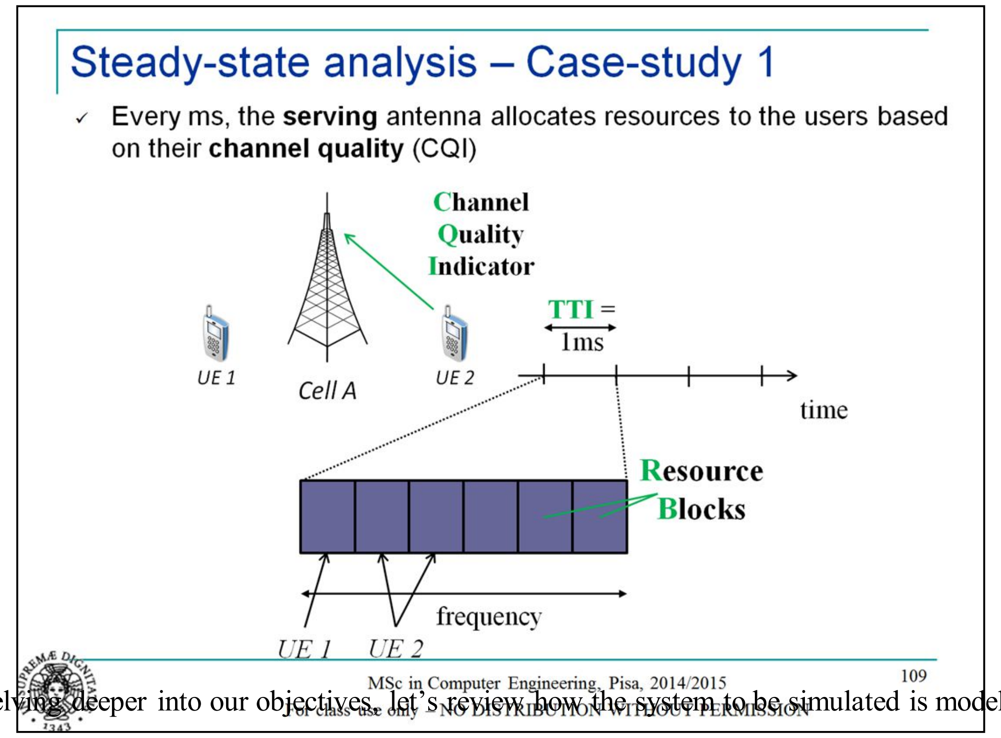
The claim that I want to prove is written in red in the slide.

How should I simulate this network? For how long? When should I start taking measurements? **Throughput** is of course a long-term measure, hence should be measured in the steady state.

The rest of these few slides show how to determine

a) When the initial transient is over

b) How long should the simulation be in the steady state.



Steady-state analysis – Case-study 1

- ✓ Every ms, the **serving** antenna allocates resources to the users based on their **channel quality** (CQI)

Channel
Quality
Indicator
UE 1
Cell A
UE 2
TTI =
1ms
time
Resource
Blocks
frequency
UE 1
UE 2

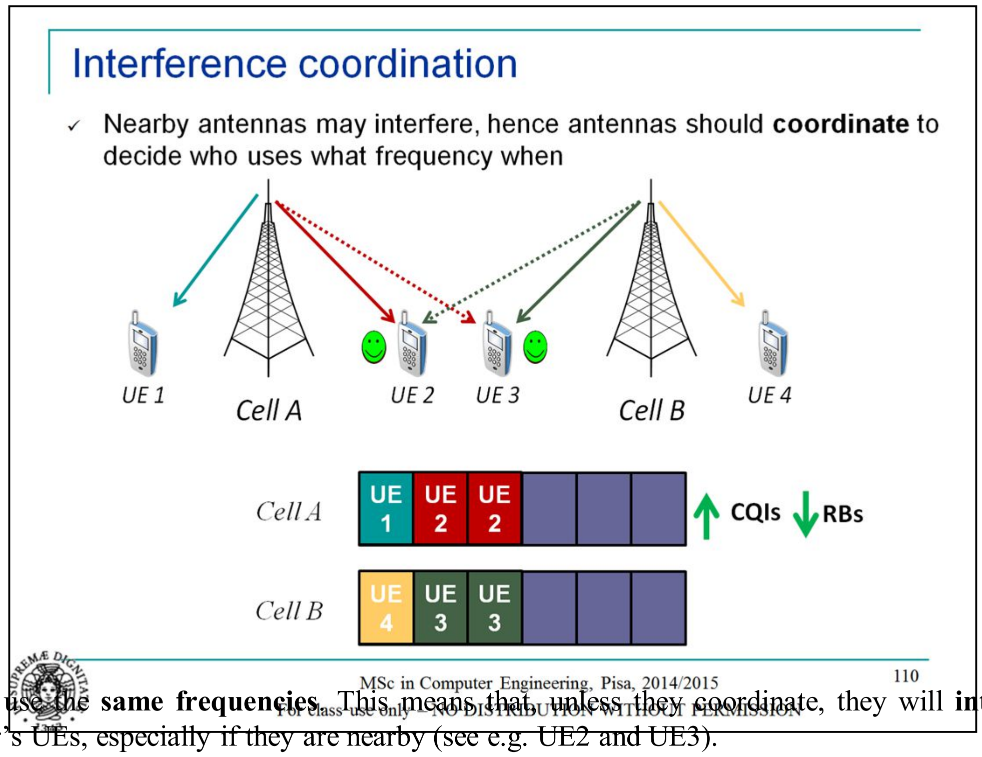
Before delving deeper into our objectives, let's review how the system to be simulated is modeled.

-Downlink scheduling

-UE Queues are at the eNB

-Downlink scheduling is done based on a) the CQIs received by the UEs, and b) the type and amount of data sitting in the UE queues at the eNB.

CQI: how many bits I can fit in one Resource Block



Interference coordination

- ✓ Nearby antennas may interfere, hence antennas should **coordinate** to decide who uses what frequency when

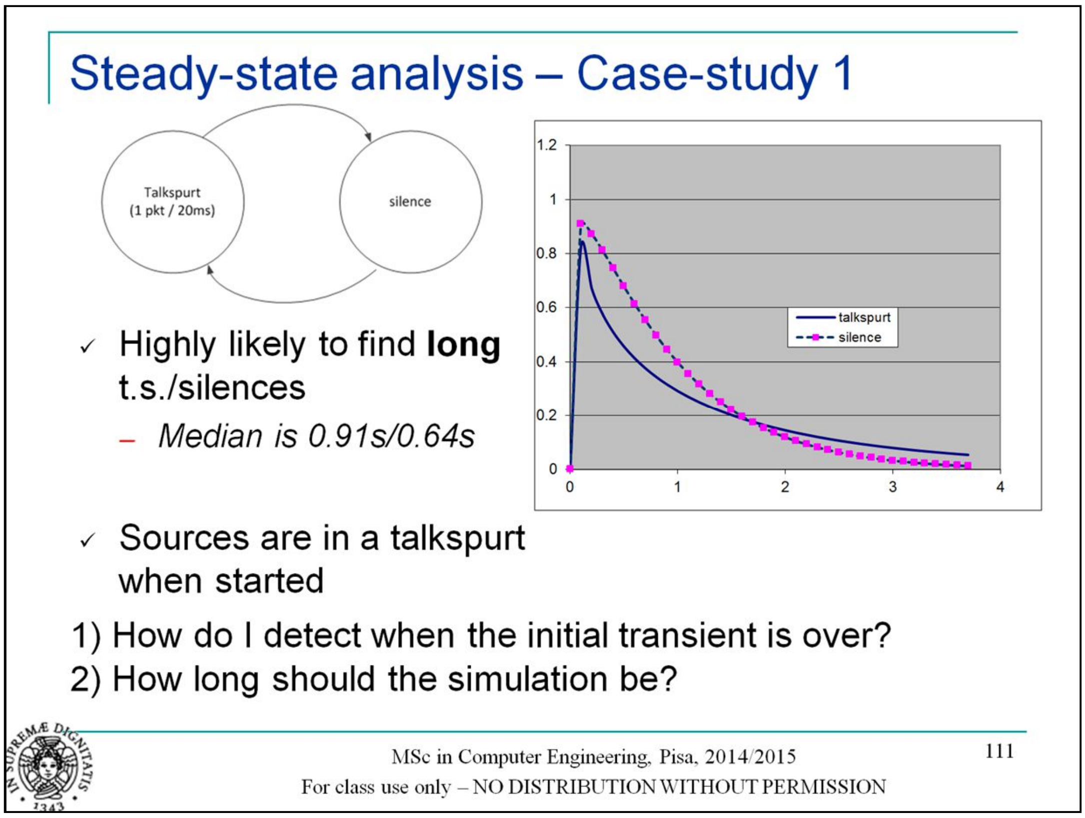
UE 1
Cell A
UE 2
UE 3
Cell B
UE 4
Cell A
UE 1
UE 2
UE 2
Cell B
UE 4
UE 3
UE 3
CQIs
RBs

Antennas use the **same frequencies**. This means that, unless they coordinate, they will **interfere** on each other's UEs, especially if they are nearby (see e.g. UE2 and UE3).

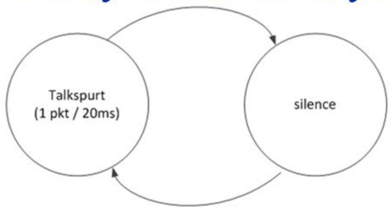
Coordination means that they agree first on who is going to use which frequencies, so as to mitigate interference.

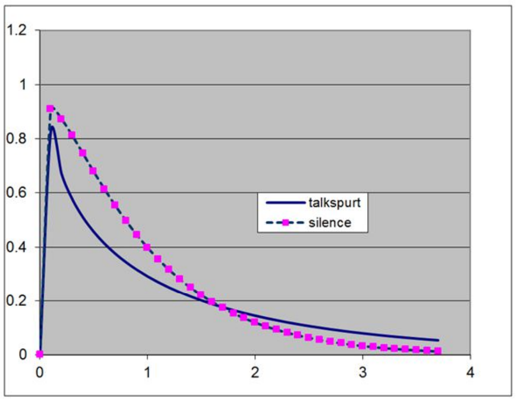
This has a threefold benefit:

- The CQI on coordinated RBs increases
- The same data can be inserted into **fewer RBs**, thus freeing resources to serve more users simultaneously
- Less power is consumed (operators pay the bill for the power they consume, much like we do)



Steady-state analysis – Case-study 1





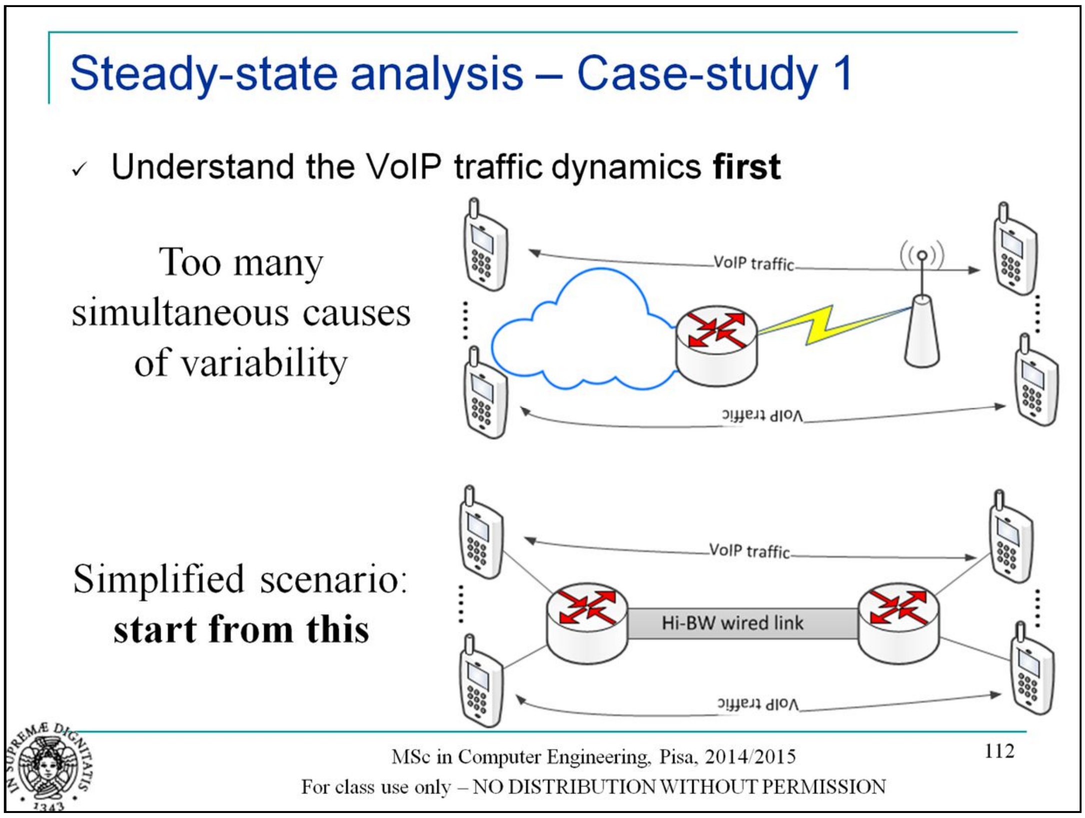
- ✓ Highly likely to find **long** t.s./silences
  - *Median is 0.91s/0.64s*
- ✓ Sources are in a talkspurt when started

1) How do I detect when the initial transient is over?
2) How long should the simulation be?

Let us take a look at what traffic is transmitted in this simulation.

The distribution of the duration of the talkspurts and silences are Weibull, with the above median. Their **mean** is even higher, since distributions are skewed to the left.

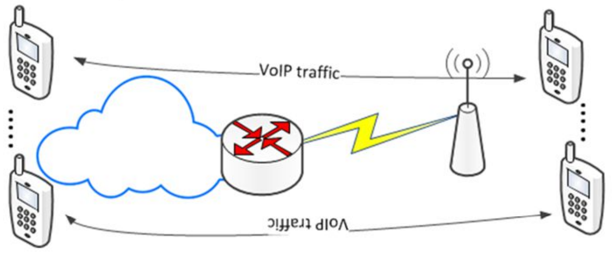
By looking at the above PDFs, it is quite clear that you can encounter **long** silences and talkspurts with a non negligible probability. By "long" I mean 4-5 seconds, or 5-10 times the median. This will have some bearing on our selection of the warm-up and simulation duration.



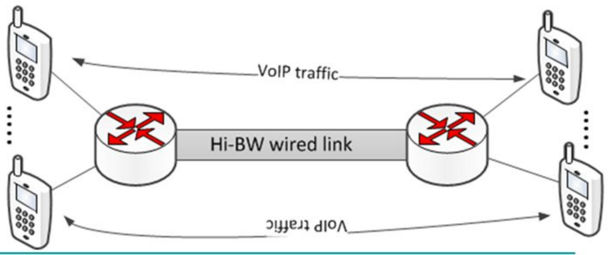
Steady-state analysis – Case-study 1

✓ Understand the VoIP traffic dynamics **first**

Too many simultaneous causes of variability



Simplified scenario: **start from this**



In our case-study, we need to show how **throughput changes** due to the interference conditions, and what happens if you eliminate interference.

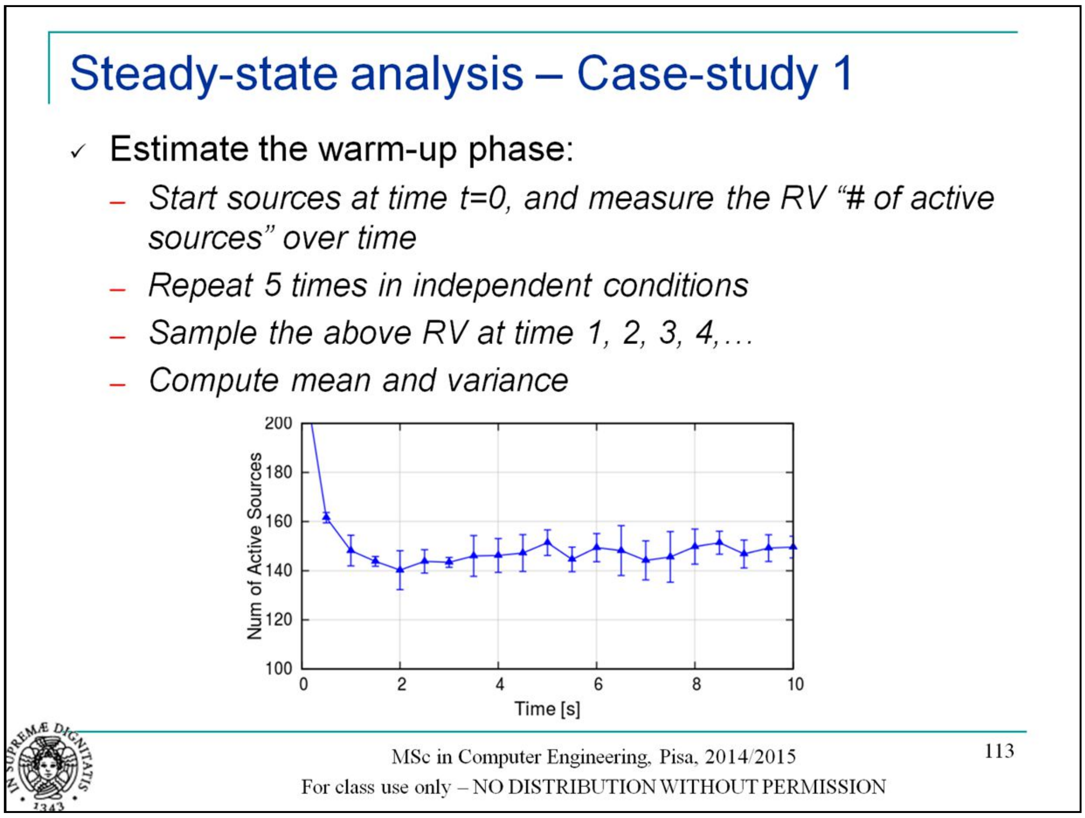
My point is: sources are **random**, so there will be differences in throughputs which are due to **randomness** in the activity ratio, and others that are due to **differences in interference**. I want to measure the impact of the **latter only**, and discount the effects of randomness as much as possible.

So, I need to understand **randomness first**.

In order to understand it, it is preferable to **simplify the scenario as much as possible**. Try running the same number of VoIP conversations **without** an LTE network in the middle first, and only **then** you put it back.

The network in the simplified scenario is **as ideal as it can be**: an overdimensioned bandwidth, no wireless effects (such as packet losses or retransmissions), no core-network delay, etc.

In the above example, the only sources of randomness are the **different activity ratios**.



Steady-state analysis – Case-study 1

✓ Estimate the warm-up phase:
- *Start sources at time t=0, and measure the RV "# of active sources" over time*
- *Repeat 5 times in independent conditions*
- *Sample the above RV at time 1, 2, 3, 4,…*
- *Compute mean and variance*

Num of Active Sources
Time [s]
200
180
160
140
120
100
0
2
4
6
8
10

We want to measure the **throughput**, i.e., the following expression: tj=[# packets received by UE j]/[t1-t0].

Our claim is that the throughput:

a) Increases in general;

b) Becomes more evenly distributed (i.e., **fairer**).

If coordination among antennas is used.

In a system where the resources are **finite** (the # of RBs to be shared among the UEs), we have a steady-state situation if the number of **active** VoIP sources (i.e., those in a talkspurt) is **statistically** constant over time.

We should therefore ask ourselves when these conditions are verified. This helps us to give a value to t0 in the above window.

In order to verify this, we run **the same number of VoIP sources without the LTE network**, and plot the number of those that are in a talkspurt at any given time.

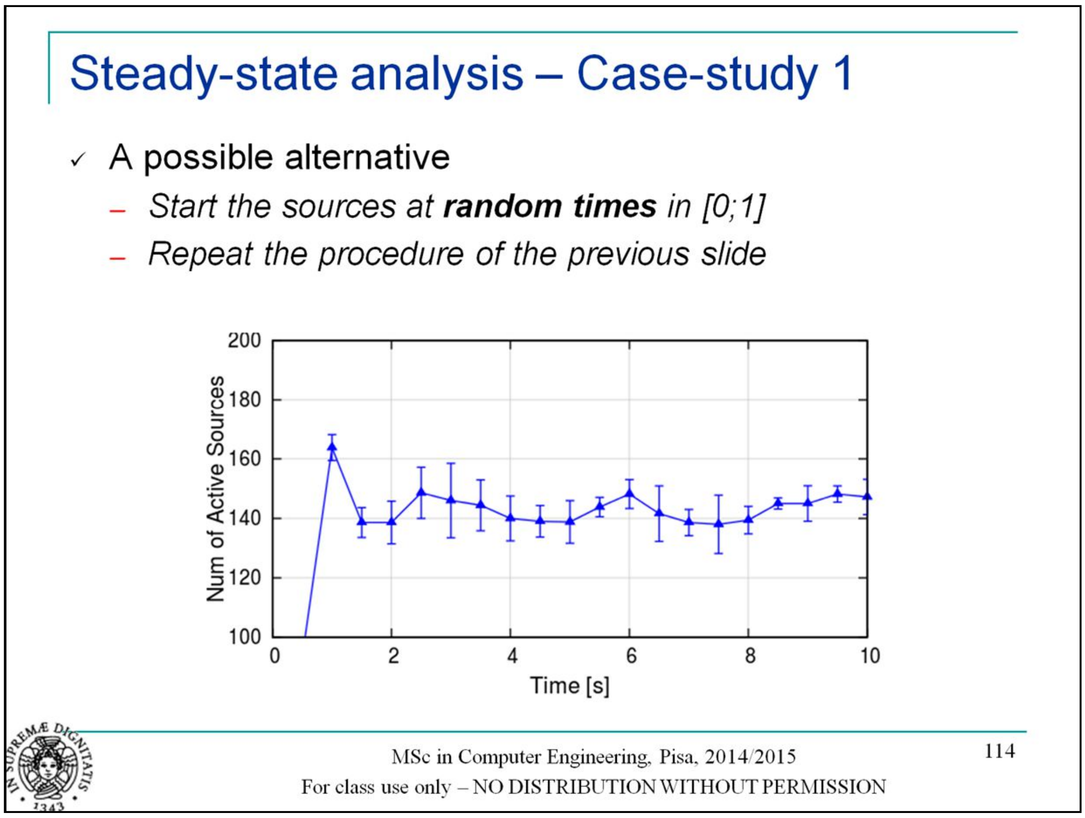
All sources start at t=0 (by design of the simulation model for VoIP).

We observe the following:

- initially, all sources are in a talkspurt (see the left part of the graph)
- After a while, the randomness of the duration of talkspurts and silences takes over, and the number of active sources evens out to a steady-state number
- That number should be equal to N*E[ts]/(E[ts]+E[silence])=0.65*N (otherwise something is wrong in the simulation, e.g. random number generation).

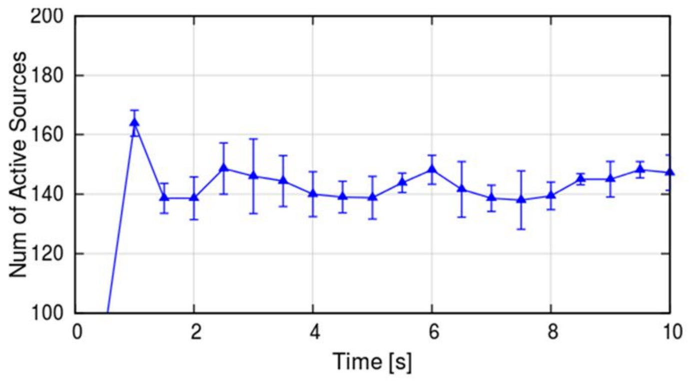
Our conclusions are that the **warm-up** period, in this case, lasts until 3-4 s, so we should set **t0=4**.

Can we do anything to reduce the initial transient? Not much of a problem here, but let's assume it is.



Steady-state analysis – Case-study 1

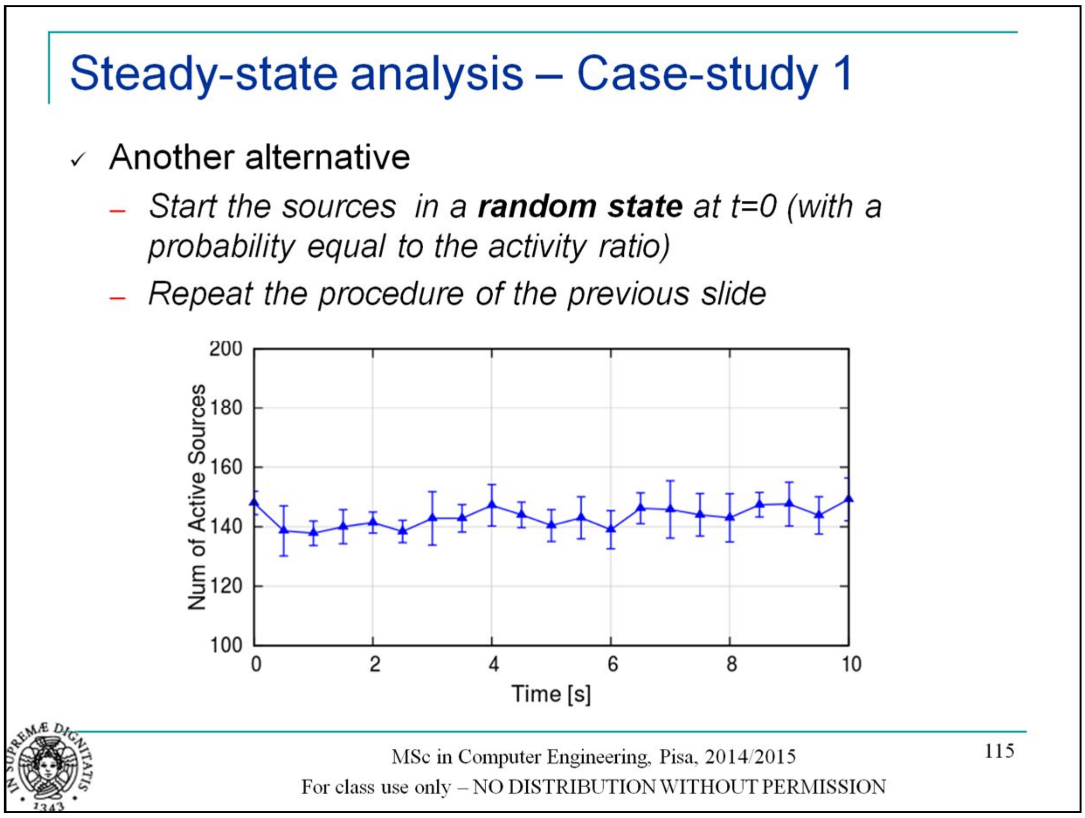
- ✓ A possible alternative
  - *Start the sources at **random times** in [0;1]*
  - *Repeat the procedure of the previous slide*



One idea: instead of starting **all the sources at time t=0**, we might try to **start them at random times**, e.g. in U(0,1).

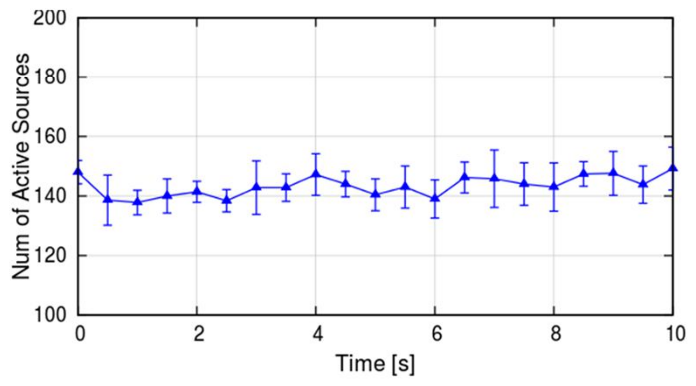
The result is that the transient does not end sooner. In fact, in this case too we see that it is not safe to start collecting measures before 2-3 s.

The idea did not sort out any good effect.



Steady-state analysis – Case-study 1

- Another alternative
  - *Start the sources in a* ***random state*** *at t=0 (with a probability equal to the activity ratio)*
  - *Repeat the procedure of the previous slide*

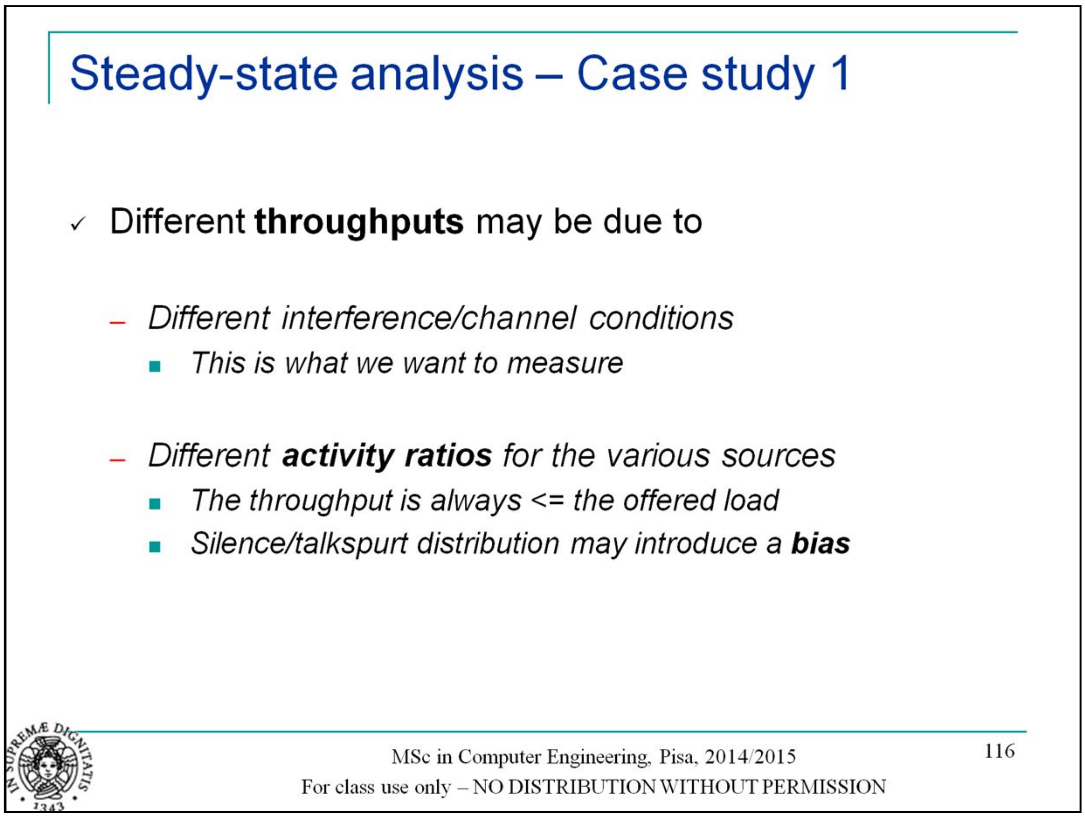


We start a source in a talkspurt state or in a silence state **at random**, with a probability p=E[ts]/(E[ts]+E[silence]). This way, the number of sources in a talkspurt is **(stochastically) constant** over time.

The RV appears to be in a steady state right at time t=0.

So, depending on how you initialize your scenario, the initial transient will terminate sooner or later.

It would not be a problem in any case in this simulation, since the transient is **short** in any case. But still, it pays to look at the problem carefully.



Steady-state analysis – Case study 1

- ✓ Different **throughputs** may be due to
  - *Different interference/channel conditions*
    - *This is what we want to measure*
  - *Different **activity ratios** for the various sources*
    - *The throughput is always <= the offered load*
    - *Silence/talkspurt distribution may introduce a **bias***

We now have the problem of selecting **t1**. How long should the simulation be?

We are measuring the **throughput**. The latter is the **# of received packets** over a window of time, and cannot be higher than the **# of transmitted packets** in the same time window.

Since the ts and silences have **random duration**, the last quantity will be **random**. The question is: how **variable is it?**

Because if it is too variable, I will never be able to tell whether differences in **throughput** are consequences of:

a) The differences in **interference conditions** (which is what I want to measure)

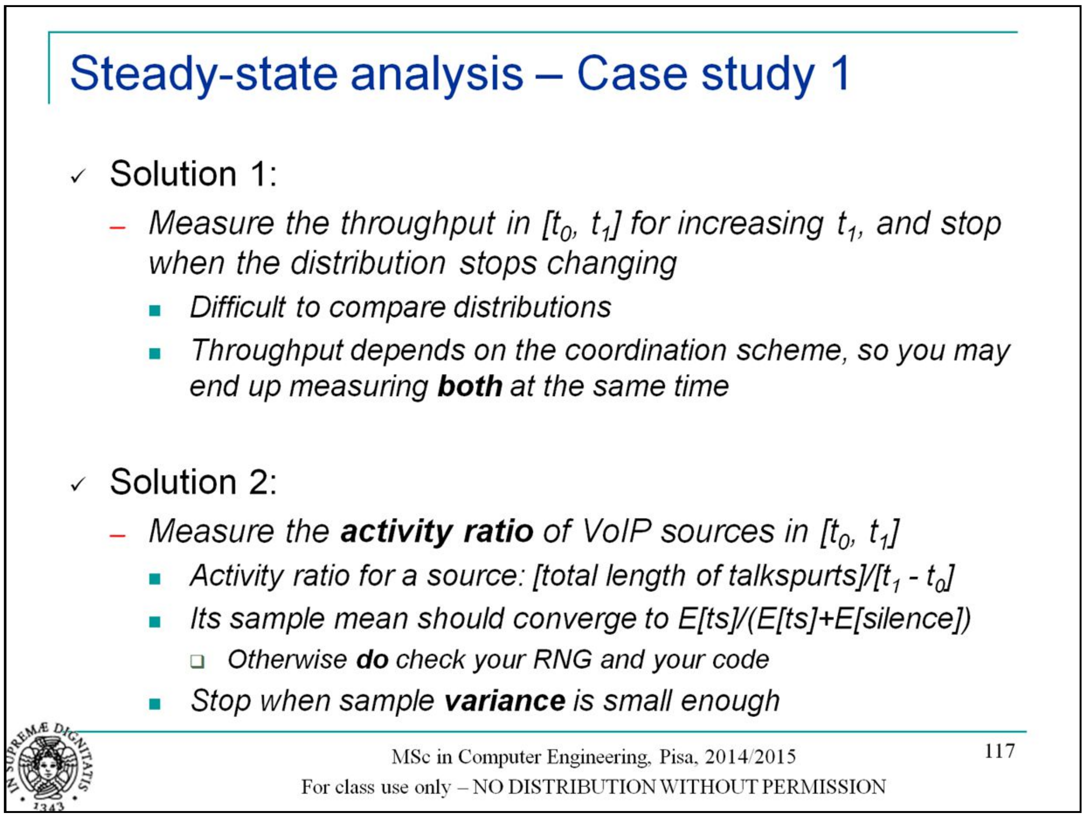
b) The differences in **activity ratios** (which is just a nuisance I would gladly do without).

Assume that I measure the above network for 10s after removing the initial transient, and that I spot some differences in throughput.

I cannot say for certain that those differences are due to **different interference conditions**, because they may be due to the fact that **one source has much longer silences than another**, and this may affect its throughput **more than interference would do**.

There is a possible bias due to the different **activity ratios**, hence I should remove it before attempting to draw a conclusion (recall the discussion on the **scientific method**: do not believe a conclusion unless it is thoroughly proved).

How can I remove the activity ratio bias?



Steady-state analysis – Case study 1

- ✓ Solution 1:
  - *Measure the throughput in $[t_0, t_1]$ for increasing t_1, and stop when the distribution stops changing*
    - *Difficult to compare distributions*
    - *Throughput depends on the coordination scheme, so you may end up measuring **both** at the same time*
- ✓ Solution 2:
  - *Measure the **activity ratio** of VoIP sources in $[t_0, t_1]$*
    - *Activity ratio for a source: [total length of talkspurts]/$[t_1 - t_0]$*
    - *Its sample mean should converge to E[ts]/(E[ts]+E[silence])*
      - *Otherwise **do** check your RNG and your code*
    - *Stop when sample **variance** is small enough*

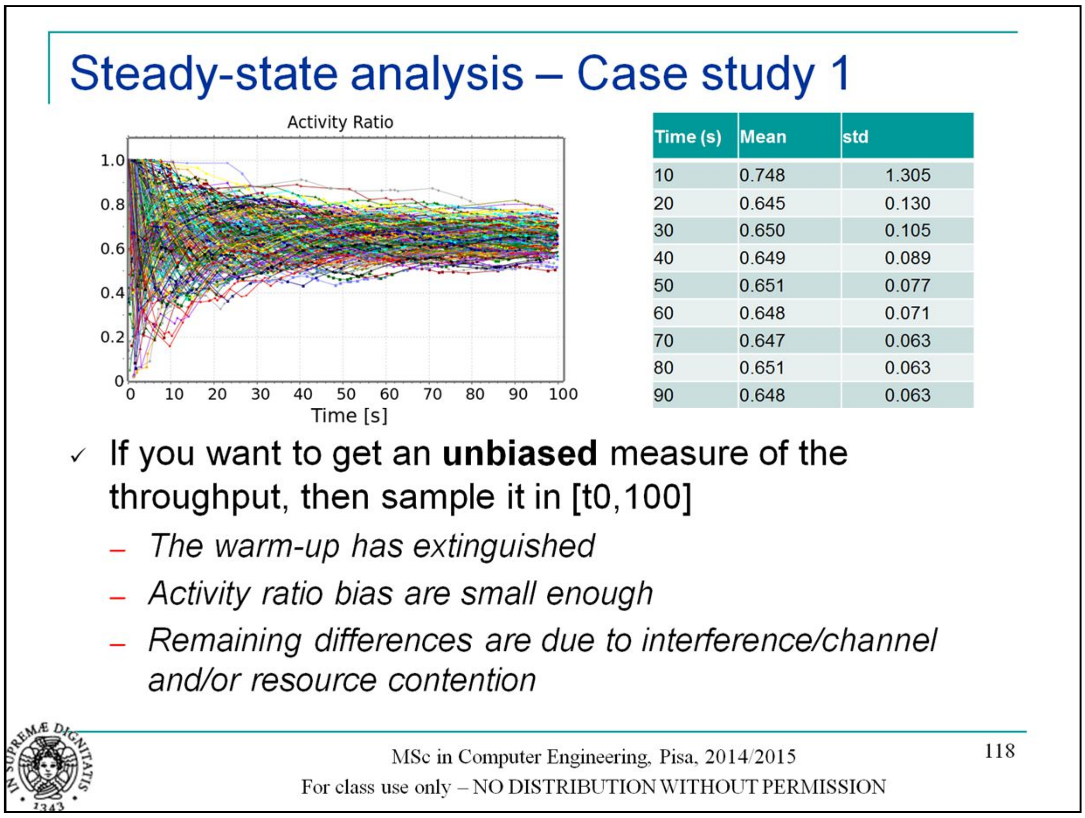
If I choose the first option, then I will have some difficulties to understand which effect is due to which cause.

The second alternative is definitely preferable, because it does not require running the whole system - you can **disable everything** in the simulator except your traffic sources, and have them run

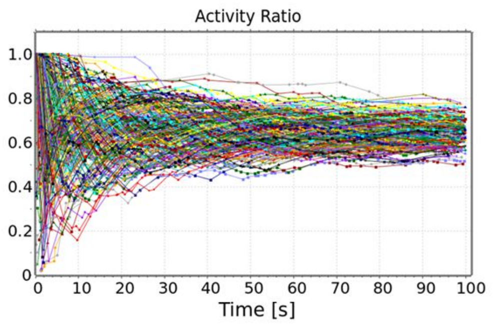
Now, the activity ratio in [t0, t1] will be a **random variable:** each source will provide one sample. Samples will be independent because they are generated using different RNG instances, with “good”, well-spaced seeds.

Therefore, I can compute the (sample) mean and variance of the AR of the N sources in [t0,t1].

- The sample mean should **converge to** E[ts]/(E[ts]+E[silence]) as t1 grows larger (otherwise there *is* a problem).
- What I am really interested in is its **sample variance**: if this is large, then it means that **large differences in the activity ratios among UEs should be expected**, hence large difference in throughputs will ensue from them, whatever coordination scheme I adopt. If, instead, the sample variance is small, then the differences in AR will be negligible.



Steady-state analysis – Case study 1



Time (s)	Mean	std
10	0.748	1.305
20	0.645	0.130
30	0.650	0.105
40	0.649	0.089
50	0.651	0.077
60	0.648	0.071
70	0.647	0.063
80	0.651	0.063
90	0.648	0.063

- ✓ If you want to get an **unbiased** measure of the throughput, then sample it in [t0,100]
  - *The warm-up has extinguished*
  - *Activity ratio bias are small enough*
  - *Remaining differences are due to interference/channel and/or resource contention*

Each line in the figure reports the activity ratio of **one source** as a function of the simulation time. We can clearly see that, for smaller simulation times, there is a wide spread, and that the spread gets smaller as the time window increases.

The table reports the **(sample) mean and standard deviation** for the activity ratios as a function of the simulation time. We see that the **sample mean converges quite fast** (it is stable already at t=20). In fact, even though all sources start in a talkspurt (hence the sample mean of the AR is initially overestimated), after a small while there is enough randomness to even out all the differences.

However, the **standard deviation** is still quite large at t=20. It is:

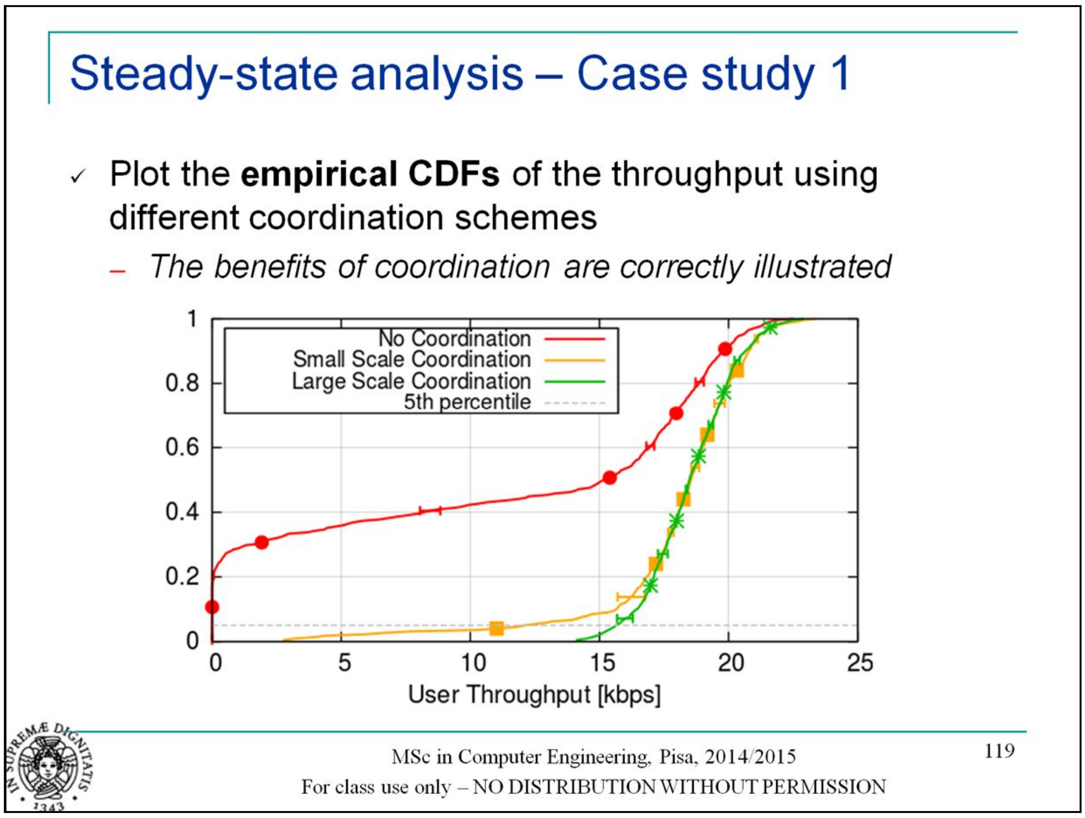
a) At t=20, equal to 0.13

b) At t=90, equal to 0.063 (i.e., one order of magnitude below the mean)

Assume that the distribution of the activity ratios among all VoIP sources at any one time is **Normal**. This means that, if you stop your simulation at t=20, your normal distribution will be ~N(0.645, 0.13^2), thus you will have 95% of your activity ratios in [mu-2*sigma, mu+2*sigma]=[0.385; 0.905]. This means that large differences in throughputs may only be due to similarly large differences in the activity ratios. The ratio between the 97.5th and 2.5th percentiles of the activity ratios (i.e., the extremes of the above interval) is 2.3.

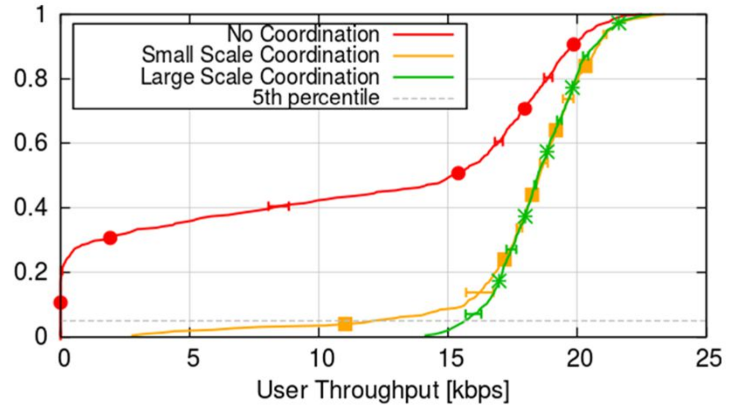
If, instead, you simulate for 90 seconds, you will find that 95% of the activity ratios are in [mu-2*sigma, mu+2*sigma]=[0.519; 0.771]. The ratio between the 97.5th and 2.5th percentile of the activity ratios is 1.5, which is a lot smaller than before.

Moreover, you notice that **the std stops decreasing** after a while. This means that probably it will not get any smaller if you simulate that system for a longer time. After 100s, then, it is probably as small as it will ever get, and you will have to live with the remaining throughput unfairness.



Steady-state analysis – Case study 1

- ✓ Plot the **empirical CDFs** of the throughput using different coordination schemes
  - *The benefits of coordination are correctly illustrated*



We wanted to show that coordination:

a) Increases the throughput in general

b) Yields a **fairer** distribution of throughput among Ues.

What is in this graph? Is this the most effective way to prove these claims? What are the alternatives?

What is a **benchmark** to compare our solution against?

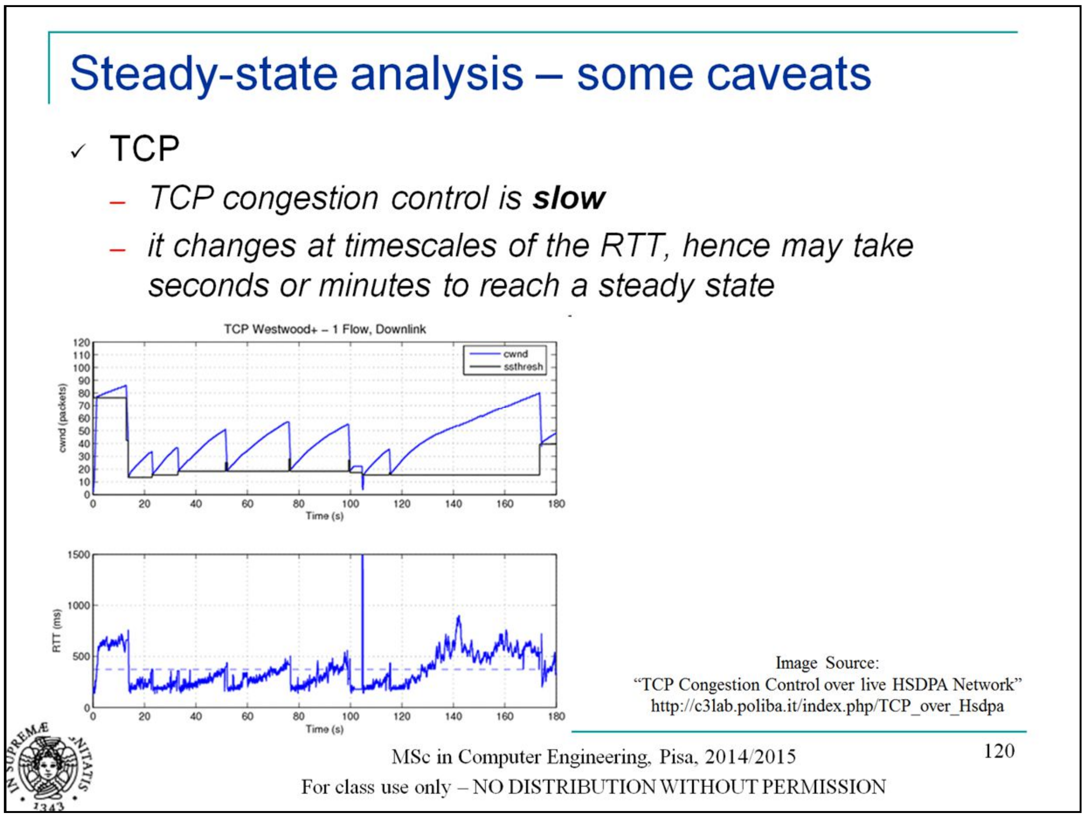
-**Pessimistic baseline**: no coordination

-**Optimistic baseline**: each station gets exactly the same throughput, equal to the voice activity ratio times the packet injection rate during talkspurts (i.e., packet size/20ms bps)

The optimistic baseline does not take into account the fact that **activity ratio biases** are still present in the graph, if we simulate for 100s.

Therefore, we may want to compare our graph against a **more realistic optimistic baseline**, which is the distribution that we obtain **without the LTE system at all**, which takes into account the activity ratio bias as well.

Back-of-the-envelope computation: if you try to figure out the **97.5th and 2.5th** percentiles of the green curve, you end up with (roughly) 22 and 15. Their ratio is ~1.47, which is **equal to the ratio of the same percentiles in the AR** (see the previous slide). This suggests that, if you use the "green" solution, then your throughputs will be distributed **pretty much the same as the activity ratios**: in other words, you are doing the best you can, since you are **not adding any more unfairness.**



Steady-state analysis – some caveats

✓ TCP
- *TCP congestion control is **slow***
- *it changes at timescales of the RTT, hence may take seconds or minutes to reach a steady state*

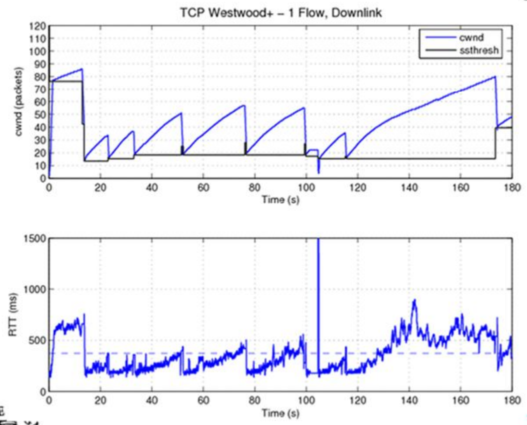


Image Source:
"TCP Congestion Control over live HSDPA Network"
http://c3lab.poliba.it/index.php/TCP_over_Hsdpa

When doing steady-state analysis, you should be aware that some phenomena have **time constants that are normally quite large** (with respect to packet-transmission timescales). This means that, if the effects of these phenomena are relevant to your simulation, you should select the simulation duration accordingly.

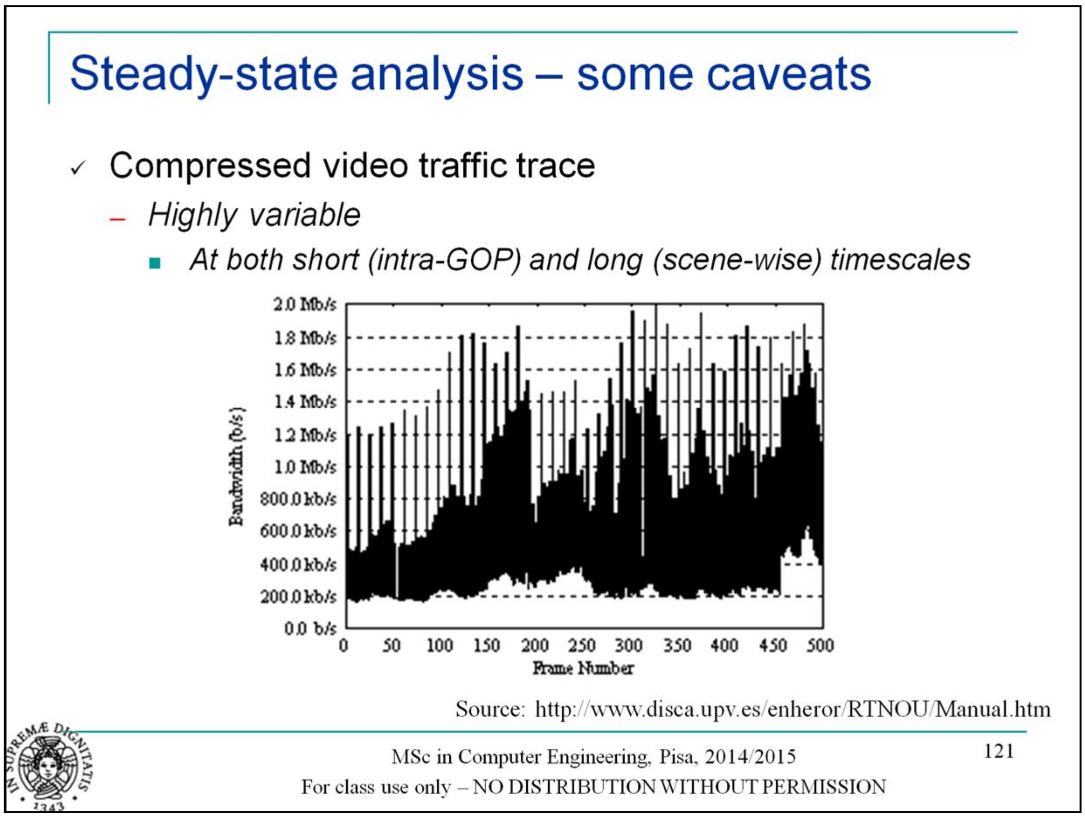
Here is a review of the most common ones.

The first one is **TCP congestion control**. (comment the graph).

Recommendation:

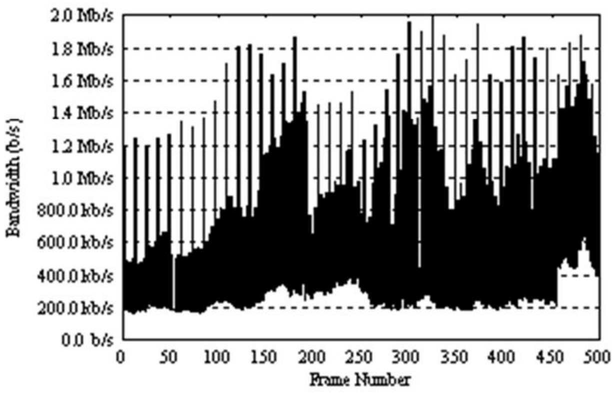
-When simulating TCP traffic, simulations should run for **at least** several hundred seconds

-Sources should be **de-synchronized**, by having them start some seconds one from the other



Steady-state analysis – some caveats

- ✓ Compressed video traffic trace
  - *Highly variable*
    - *At both short (intra-GOP) and long (scene-wise) timescales*



Source: http://www.disca.upv.es/enheror/RTNOU/Manual.htm

The second phenomenon is **compressed video**.

Video traces are **highly variable**. They follow a pre-defined transmission pattern (called a Group of Pictures, GOP), where frames with the same position have different information (and, accordingly, different length). For instance, the initial frame of a GOP ("I" frame) is normally considerably larger than the following ones ("B" or "P" frames). This clearly shows in the figure.

Moreover, the information content of a GOP varies greatly with the **type of scene** that it is represented. A "static" scene has a low bandwidth (e.g., frames 0-50), whereas a "highly dynamic" one has a high bandwidth (e.g., frames 150-200).

Recommendation:

- When using a **video trace** to simulate video transmission, use **different pointers in the trace for different users**, so that

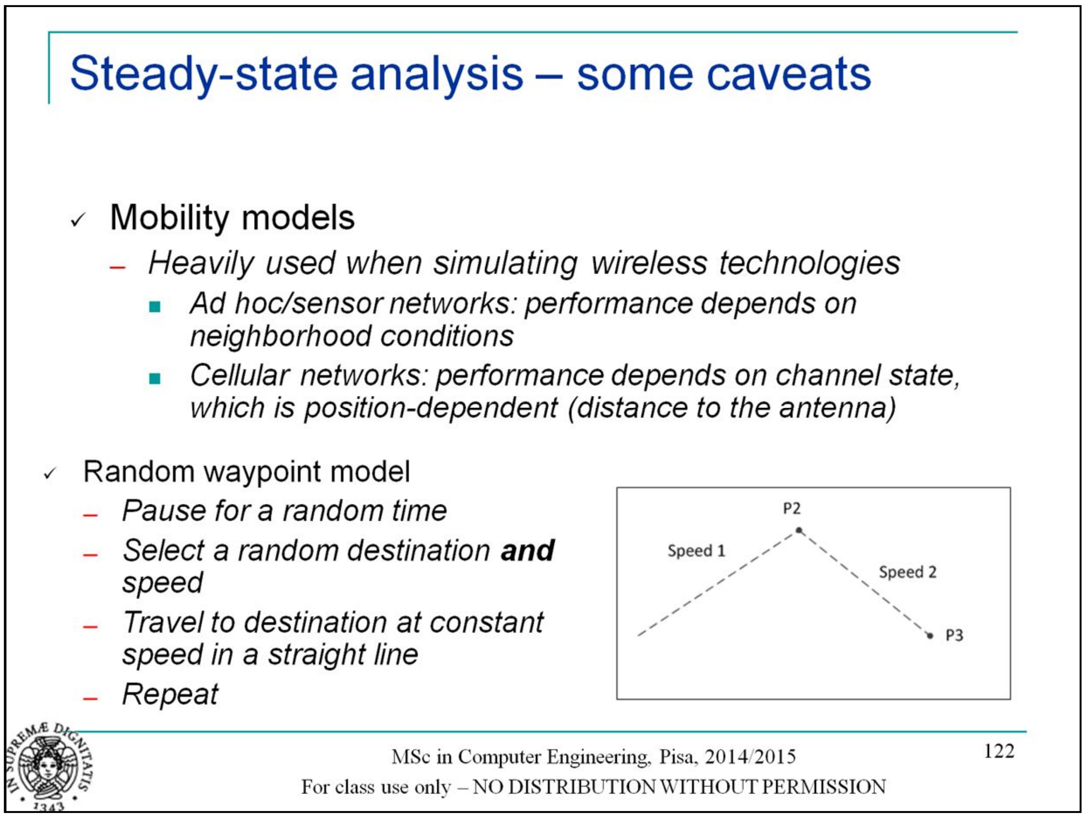
a) most of the trace is actually used

b) anti-synchronization occurs

-Simulation should be long enough to collect many GOPs

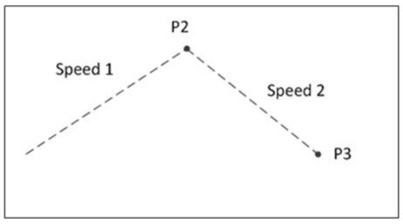
-Take into account that the **throuhgput** may vary **a lot** from one user to another, depending on the fragment of the trace that they are receiving/transmitting. The effect of varying **offered load** is in place here, but it is considerably heavier than in the previous case.

-If your **average bandwidth** for a 1.5h movie is (say) 1Mbps, sampling 1 minute of it may yield arbitrarily different values (say, 100kbps or 10Mbps), depending on which part you sample.



Steady-state analysis – some caveats

- ✓ Mobility models
  - *Heavily used when simulating wireless technologies*
    - *Ad hoc/sensor networks: performance depends on neighborhood conditions*
    - *Cellular networks: performance depends on channel state, which is position-dependent (distance to the antenna)*
- ✓ Random waypoint model
  - *Pause for a random time*
  - *Select a random destination **and** speed*
  - *Travel to destination at constant speed in a straight line*
  - *Repeat*



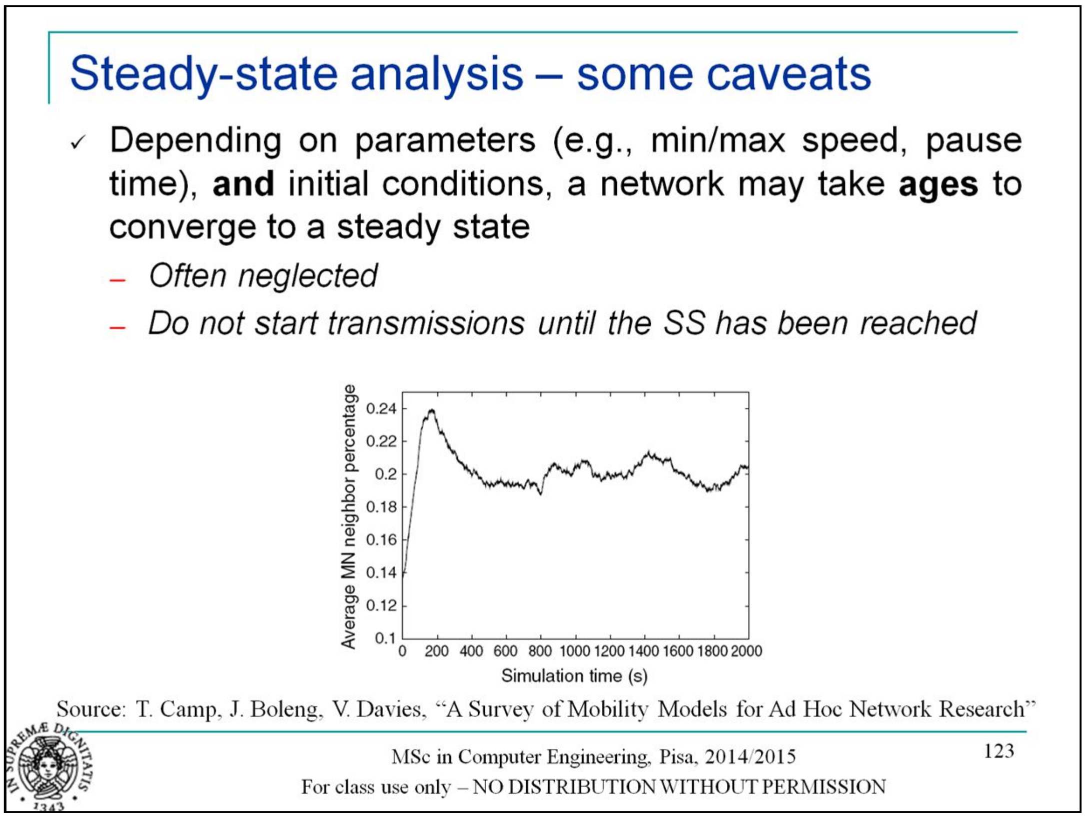
The third phenomenon is **human mobility.**

The **random waypoint** model is one of the most widely used, mainly because it is easy to implement and to analyze statistically (not certainly because people move that way).

Assume speed is drawn in U(min, max), and that the network is simulated for 1h. What will the **average sample speed** be at the end of the simulation?

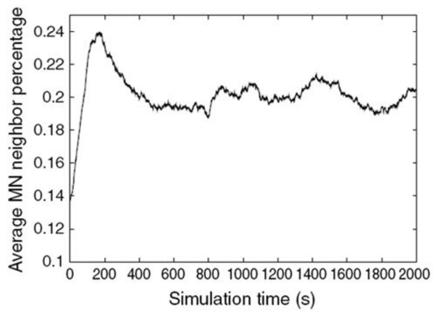
Answer: it will be close to **min** (the longer the simulation, the closer it gets). This is surprising at first. Not if you think that a **slow** movement takes a lot more time than a **fast one**. Therefore, if you fix **time**, your mobile will be spending a much higher percentage of it going **slow** than going **fast**.

This is why, if you draw the speed in U(0,max), and you wait long enough, then your simulation **freezes**, i.e. you end up with quasi-static users, which is probably not what you wanted.



Steady-state analysis – some caveats

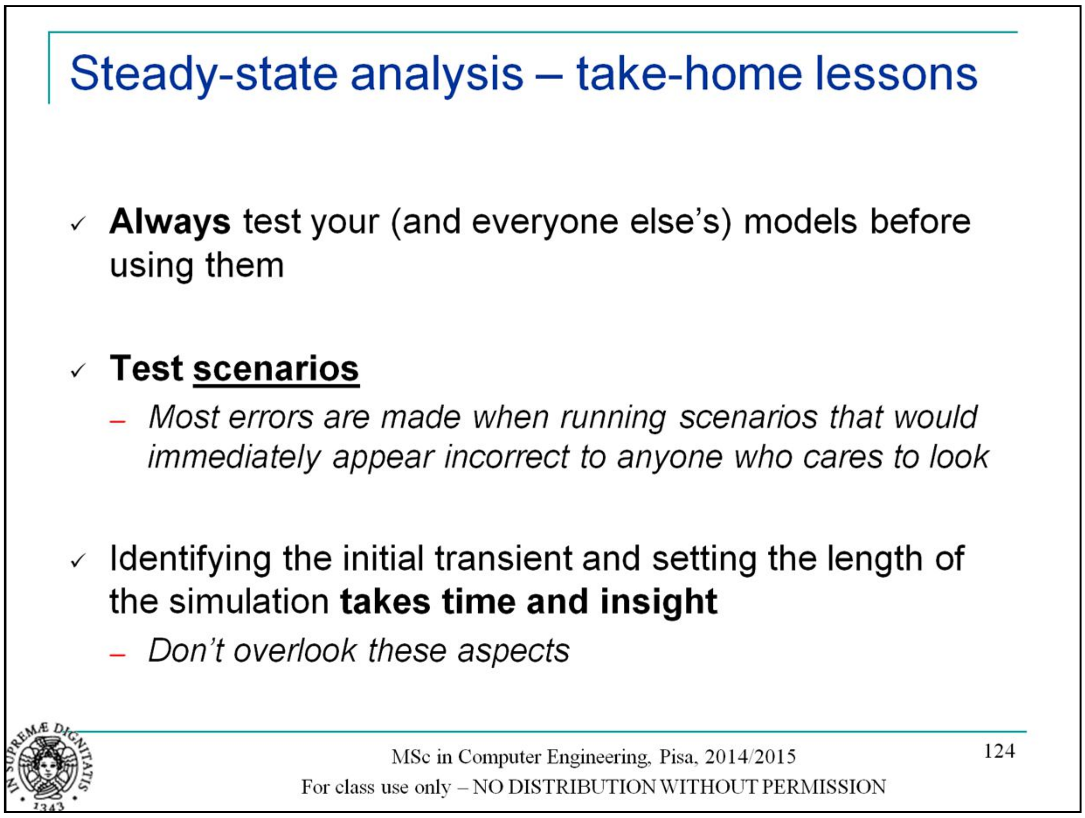
- ✓ Depending on parameters (e.g., min/max speed, pause time), **and** initial conditions, a network may take **ages** to converge to a steady state
  - *Often neglected*
  - *Do not start transmissions until the SS has been reached*



Source: T. Camp, J. Boleng, V. Davies, "A Survey of Mobility Models for Ad Hoc Network Research"

If you do things **blindly**, you may just
-Discard the first 10s
-Simulate the network for 50s

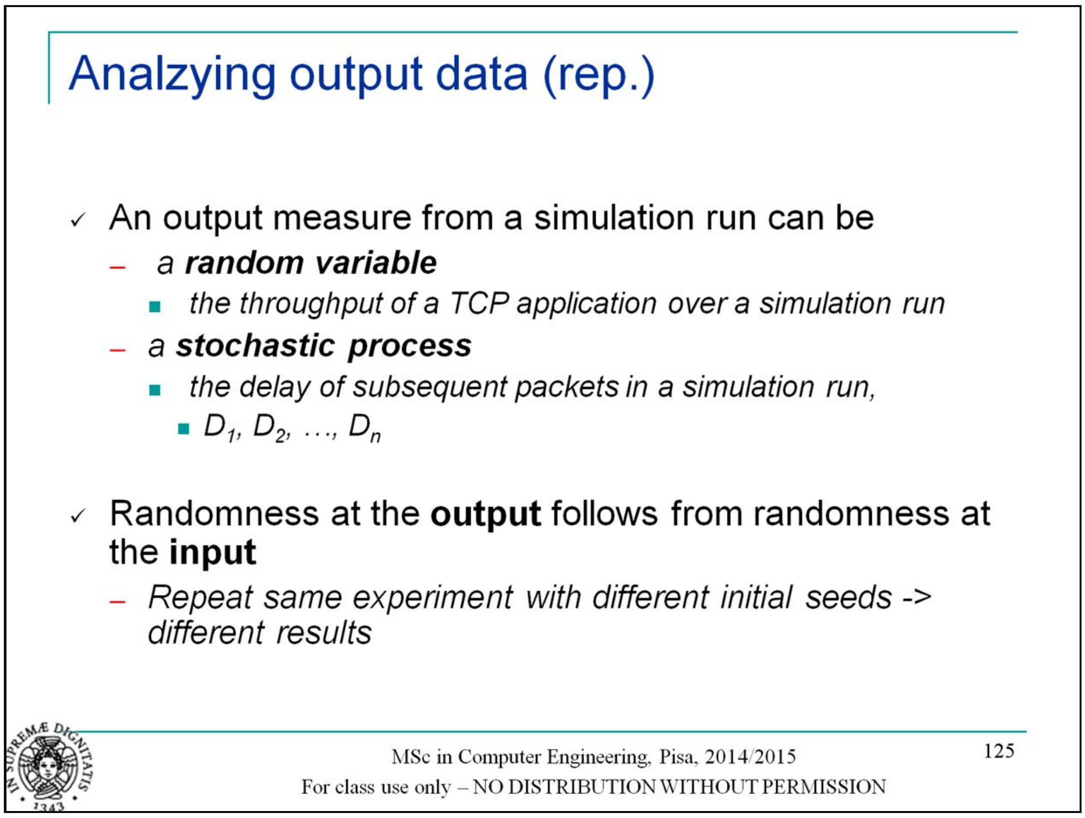
Then you would end up measuring transmissions when the number of neighbors is quite different from the one at the steady state (smaller, in this case), and **increasing over time**. You are not analyzing the steady-state properties of your system (e.g., a routing algorithm for ad hoc/sensor networks).



Steady-state analysis – take-home lessons

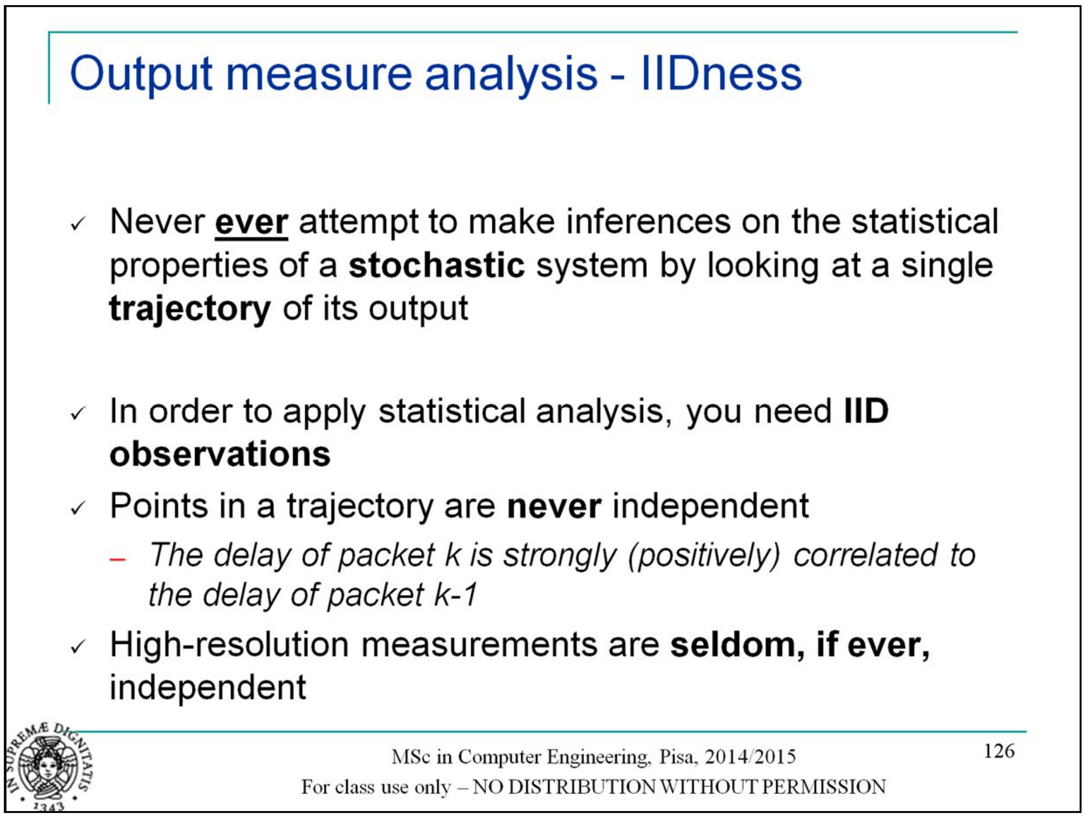
- ✓ **Always** test your (and everyone else's) models before using them
- ✓ **Test <u>scenarios</u>**
  - *Most errors are made when running scenarios that would immediately appear incorrect to anyone who cares to look*
- ✓ Identifying the initial transient and setting the length of the simulation **takes time and insight**
  - *Don't overlook these aspects*

Again, performance evaluation is **an art**. Art takes **patience and craft**. There is no easy, mechanical way to do things


Analzying output data (rep.)

- ✓ An output measure from a simulation run can be
  - a ***random variable***
    - *the throughput of a TCP application over a simulation run*
  - *a* ***stochastic process***
    - *the delay of subsequent packets in a simulation run,*
      - $D_1, D_2, \ldots, D_n$
- ✓ Randomness at the **output** follows from randomness at the **input**
  - *Repeat same experiment with different initial seeds -> different results*

What you obtain from a simulation is – almost always – random.
It can be a single variable (first case), or a **process** (second case, which includes the first one).



Output measure analysis - IIDness

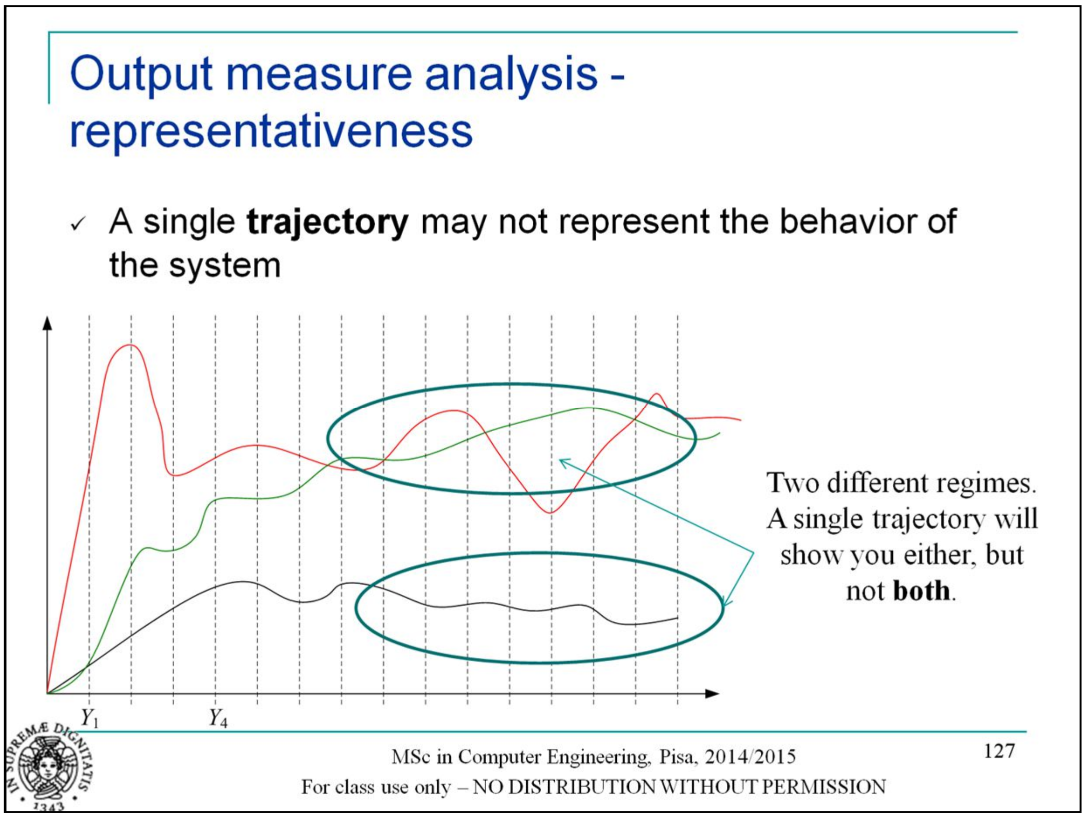
- ✓ Never **ever** attempt to make inferences on the statistical properties of a **stochastic** system by looking at a single **trajectory** of its output
- ✓ In order to apply statistical analysis, you need **IID observations**
- ✓ Points in a trajectory are **never** independent
  - *The delay of packet k is strongly (positively) correlated to the delay of packet k-1*
- ✓ High-resolution measurements are **seldom, if ever,** independent

MSc in Computer Engineering, Pisa, 2014/2015 126
For class use only – NO DISTRIBUTION WITHOUT PERMISSION

You obviously **do not know** the distribution of D (if you did, you wouldn't be simulating your system).

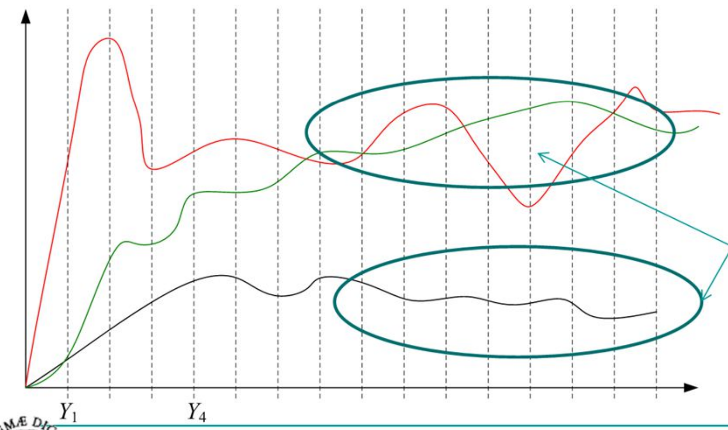
From the output of a **single run of a simulation** I cannot make inferences on any statistical property of the quantity that I am measuring.

Normally, the higher the resolution of a measurement (in space or time), the more likely it is that subsequent observations are **positively correlated**, instead of independent. This is because physical quantities do not change abruptly in space or time.



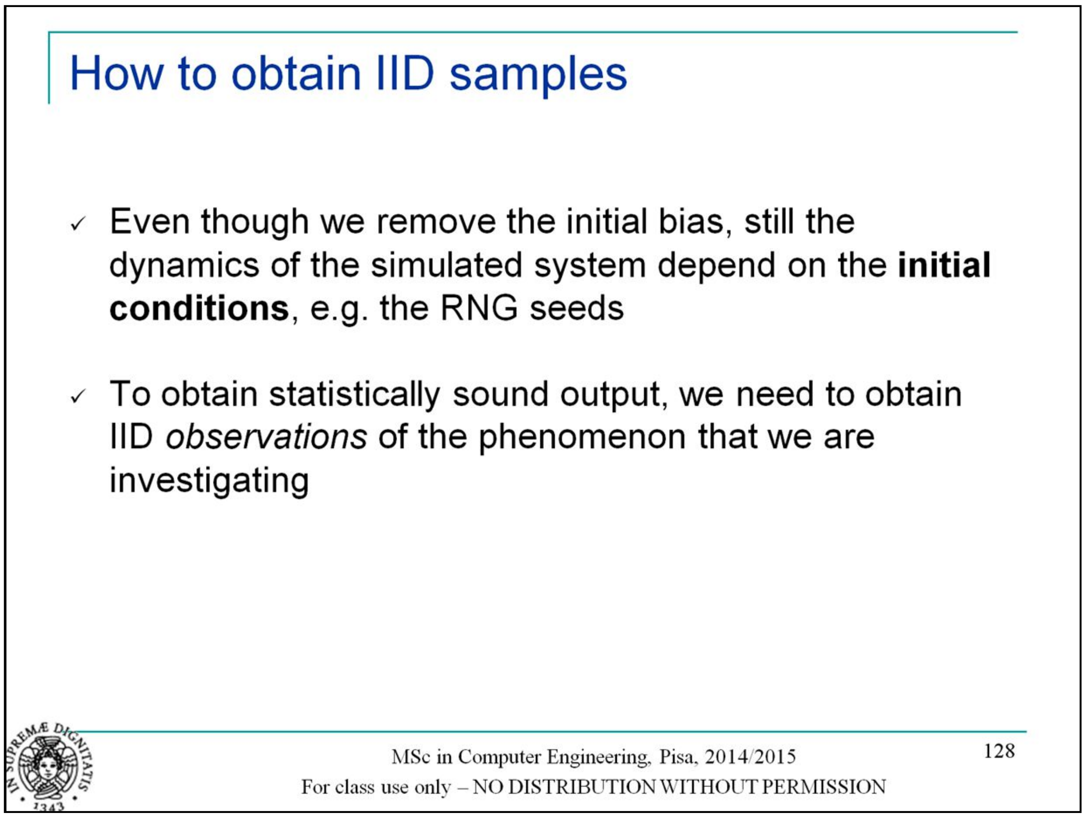
Output measure analysis - representativeness

- ✓ A single **trajectory** may not represent the behavior of the system



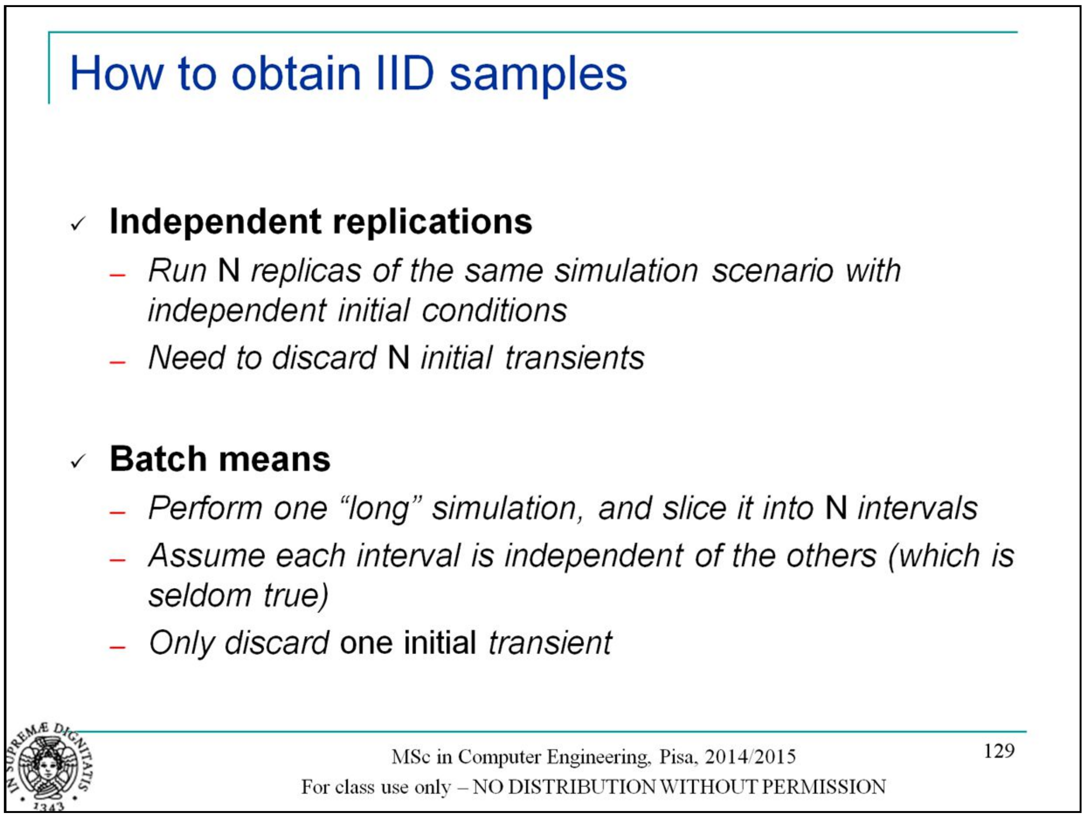
In order to produce insight on a system like this, you need to acknowledge the fact that it may work in either of two regimes, each one of which has a very different steady state, and that each regime may occur with a given probability.

You are never going to see this unless you repeat your simulation so as to observe both behaviors.



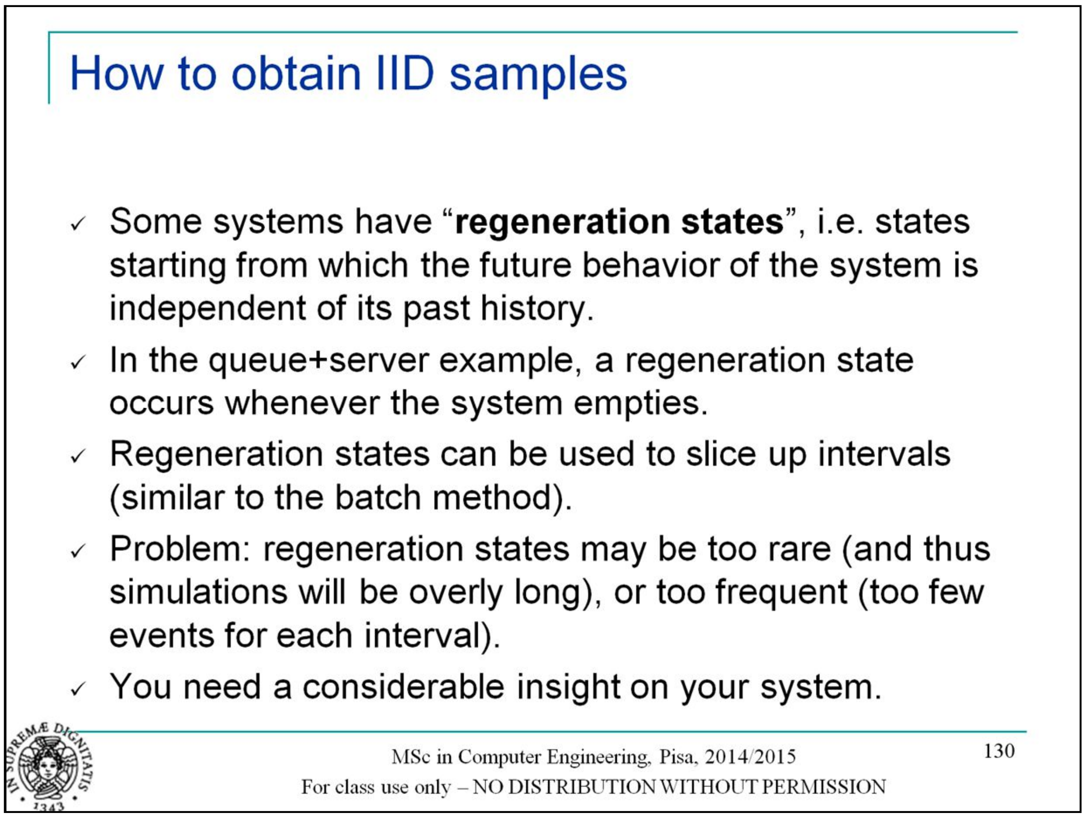
How to obtain IID samples

- ✓ Even though we remove the initial bias, still the dynamics of the simulated system depend on the **initial conditions**, e.g. the RNG seeds
- ✓ To obtain statistically sound output, we need to obtain IID *observations* of the phenomenon that we are investigating


How to obtain IID samples

- ✓ **Independent replications**
  - *Run* N *replicas of the same simulation scenario with independent initial conditions*
  - *Need to discard* N *initial transients*
- ✓ **Batch means**
  - *Perform one "long" simulation, and slice it into* N *intervals*
  - *Assume each interval is independent of the others (which is seldom true)*
  - *Only discard* one initial *transient*

Independent initial conditions means "using different seeds for the RNGs"



How to obtain IID samples

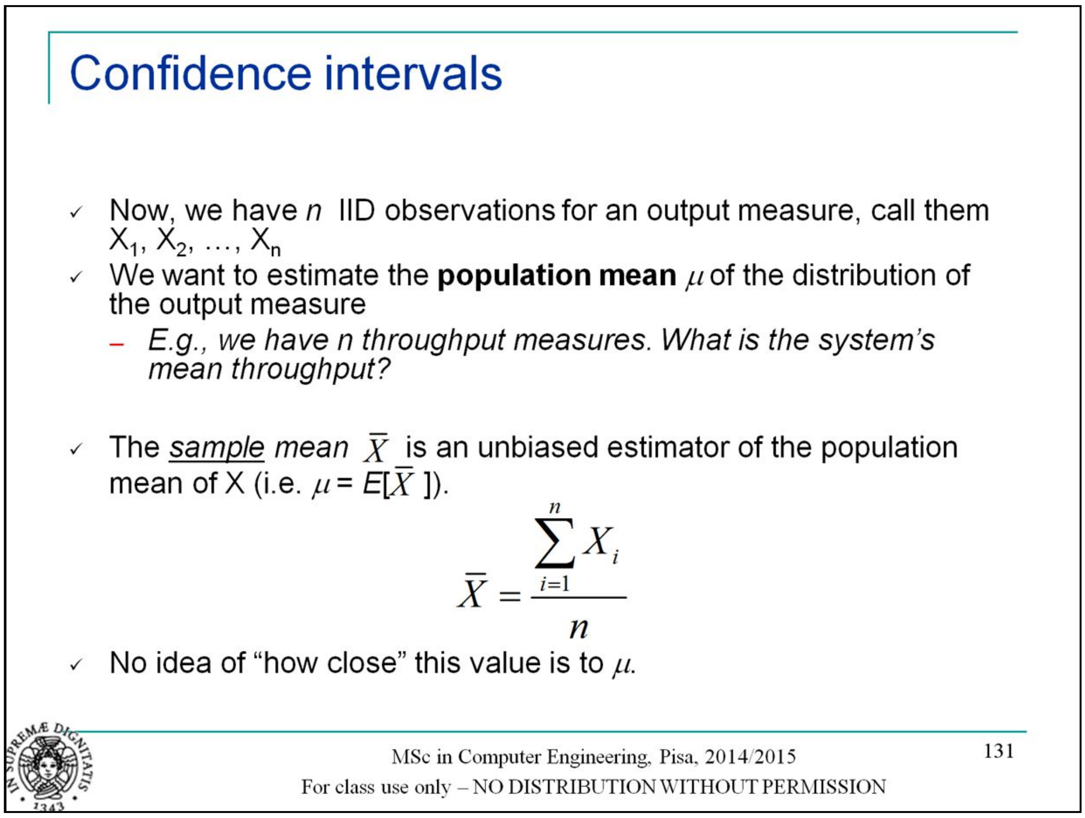
- ✓ Some systems have "**regeneration states**", i.e. states starting from which the future behavior of the system is independent of its past history.
- ✓ In the queue+server example, a regeneration state occurs whenever the system empties.
- ✓ Regeneration states can be used to slice up intervals (similar to the batch method).
- ✓ Problem: regeneration states may be too rare (and thus simulations will be overly long), or too frequent (too few events for each interval).
- ✓ You need a considerable insight on your system.

MSc in Computer Engineering, Pisa, 2014/2015 130
For class use only – NO DISTRIBUTION WITHOUT PERMISSION

If you **know** that

a) Your system has regeneration states
b) Where to find them
c) How they are distributed over time

Then you probably know enough to deal with this system using analytical methods instead of simulation.



Confidence intervals

- Now, we have *n* IID observations for an output measure, call them $X_1, X_2, \ldots, X_n$
- We want to estimate the **population mean** μ of the distribution of the output measure
  - *E.g., we have n throughput measures. What is the system's mean throughput?*
- The *sample* *mean* $\bar{X}$ is an unbiased estimator of the population mean of X (i.e. $\mu = E[\bar{X}]$).

$$\bar{X} = \frac{\sum_{i=1}^{n} X_i}{n}$$

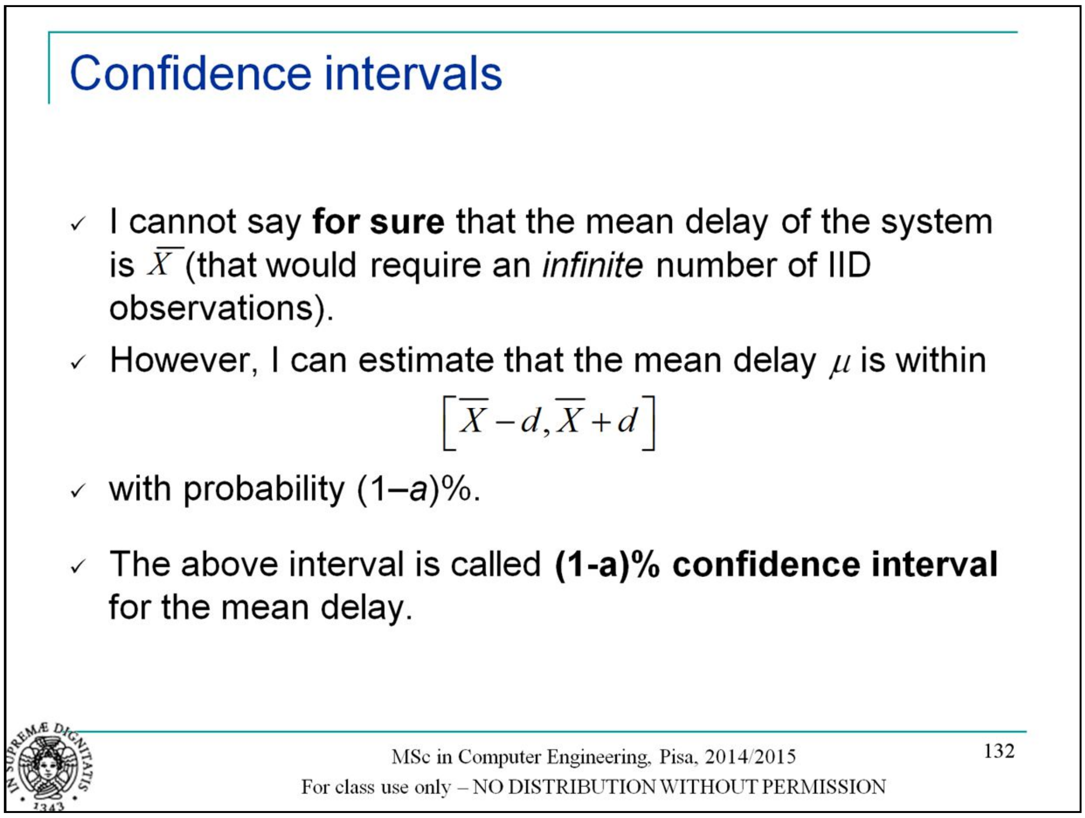
- No idea of "how close" this value is to μ.

MSc in Computer Engineering, Pisa, 2014/2015 131

For class use only – NO DISTRIBUTION WITHOUT PERMISSION

\bar{D}(n) being an unbiased estimator of \mu means that if we perform a very large number of independent replications, each resulting in an \bar{D}(n), the average of the \bar{D}(n)'s will be \mu.

The bias of an estimator is the distance beween the average of the collection of estimates and the single parameter being estimated.

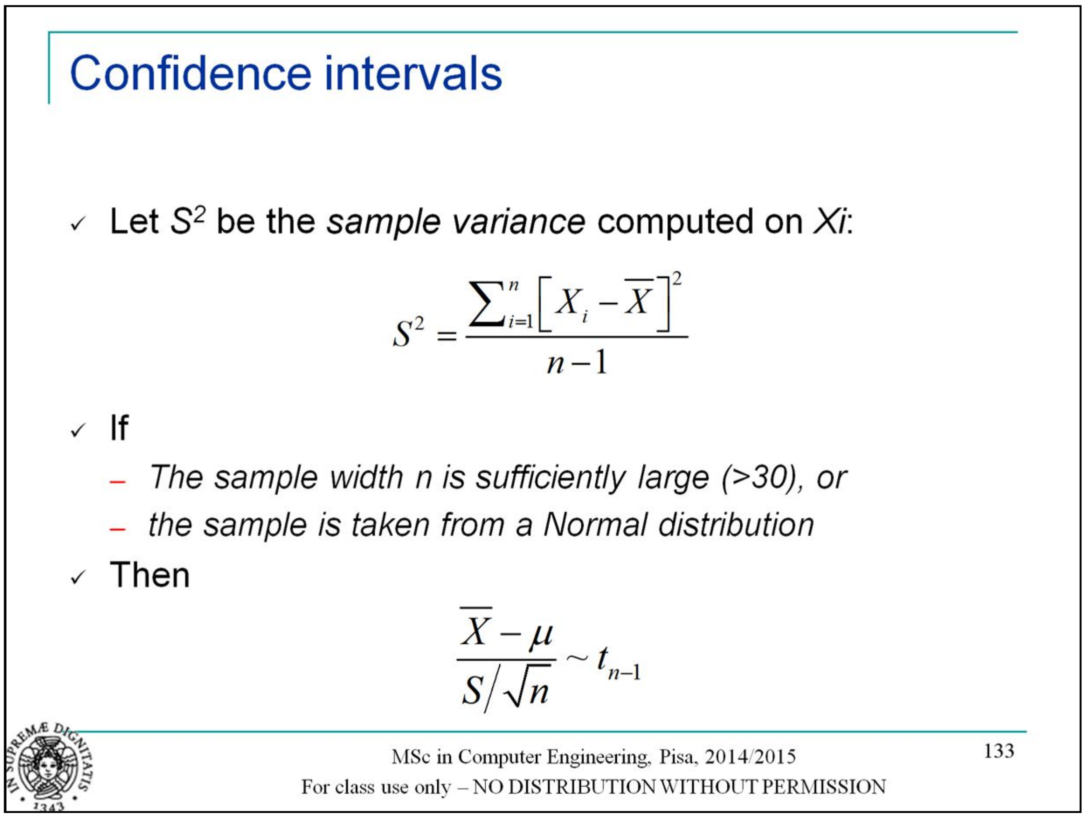


Confidence intervals

- ✓ I cannot say **for sure** that the mean delay of the system is $\overline{X}$ (that would require an *infinite* number of IID observations).
- ✓ However, I can estimate that the mean delay μ is within

$$\left[\overline{X} - d, \overline{X} + d\right]$$

- ✓ with probability (1–*a*)%.
- ✓ The above interval is called **(1-a)% confidence interval** for the mean delay.



Confidence intervals

- Let S^2 be the *sample variance* computed on *Xi*:

$$S^2 = \frac{\sum_{i=1}^{n} \left[X_i - \overline{X} \right]^2}{n-1}$$

- If
  - *The sample width n is sufficiently large (>30), or*
  - *the sample is taken from a Normal distribution*
- Then

$$\frac{\overline{X} - \mu}{S / \sqrt{n}} \sim t_{n-1}$$

Note that, when n>30, Student’s T distribution overlaps a Standard *N*ormal, hence t_(n-1) ~ z

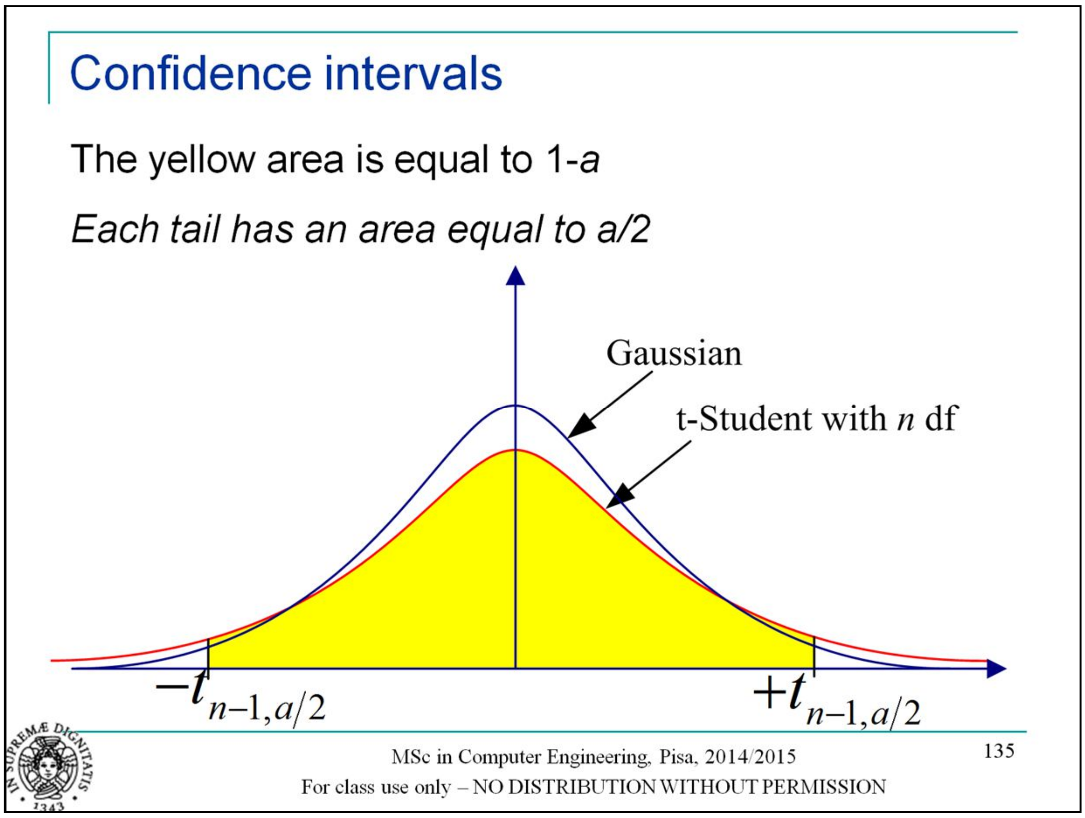
Confidence intervals

- ✓ The (1-a)% confidence interval for the population mean μ is:

$$\left[\overline{X}-t_{n-1,a/2}\cdot S/\sqrt{n},\ \overline{X}+t_{n-1,a/2}\,S/\sqrt{n}\right]$$

  - *where* t$_{n-1,1-a/2}$ *comes from the t-Student distribution function, whose values can be found in tables*

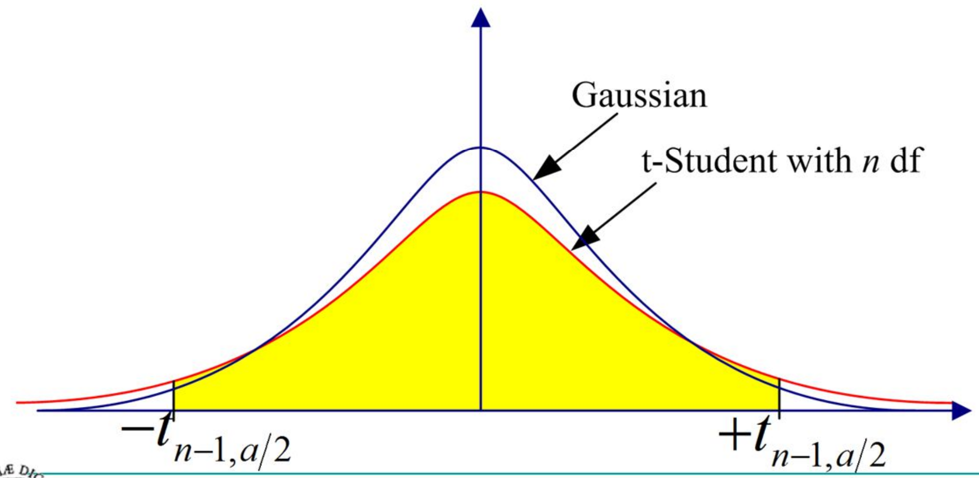
ddl	0,01	0,02	0,05	0,1
1	63,657	31,821	12,706	6,314
2	9,925	6,965	4,303	2,920
3	5,841	4,541	3,182	2,353
4	4,604	3,747	2,776	2,132
5	4,032	3,365	2,571	2,015
6	3,707	3,143	2,447	1,943
7	3,499	2,998	2,365	1,895
8	3,355	2,896	2,306	1,860
9	3,250	2,821	2,262	1,833
10	3,169	2,764	2,228	1,812
11	3,106	2,718	2,201	1,796



Confidence intervals

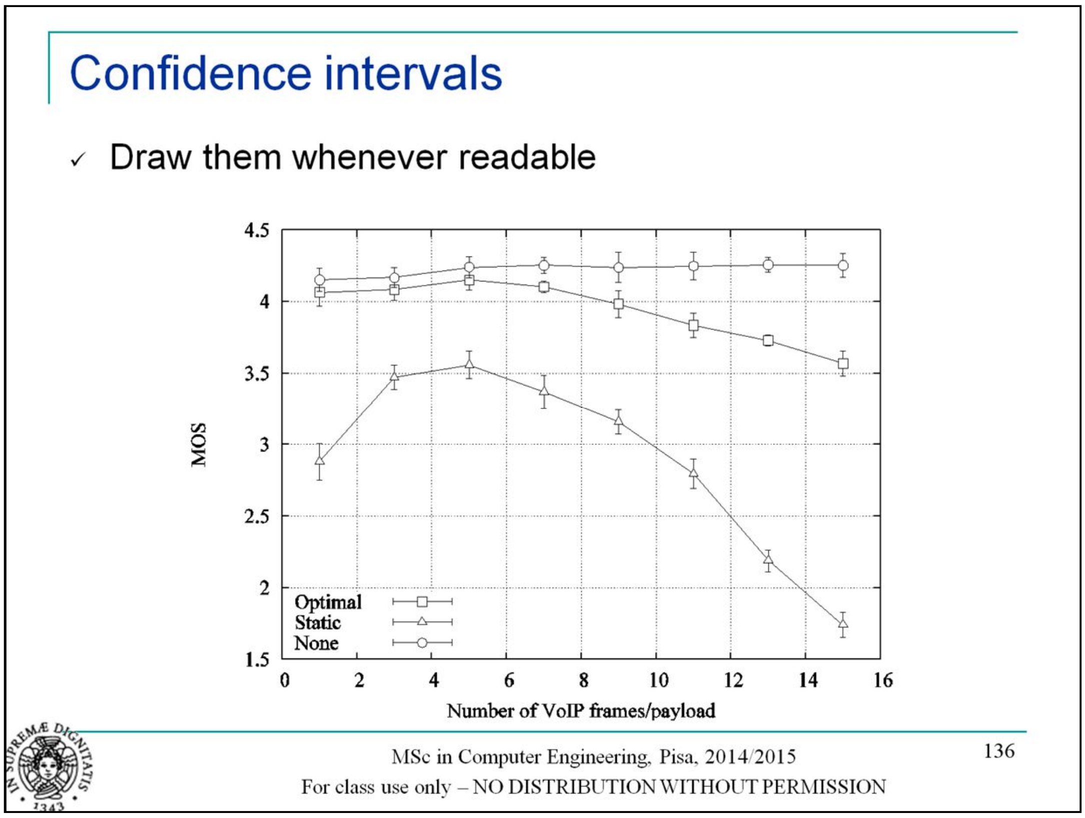
The yellow area is equal to 1-*a*

Each tail has an area equal to a/2



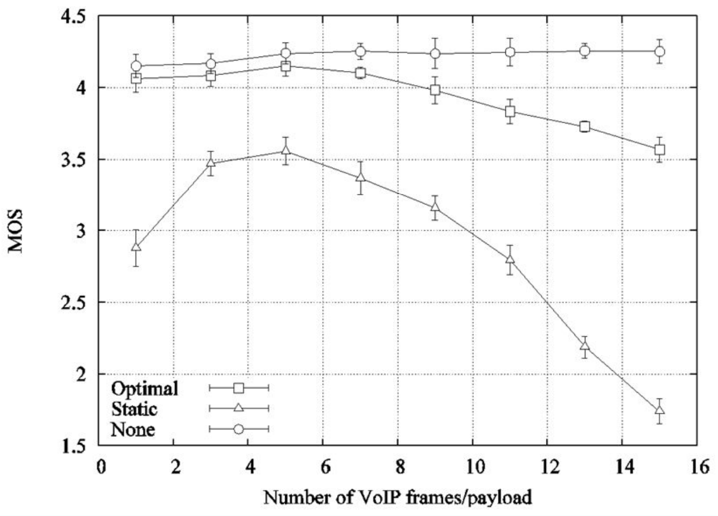
Note that confidence intervals are often **underestimated** w.r.t. reality (i.e., the probability is *less* than (1-a)%), for a number of reasons:

- You assume as independent quantities which are correlated
- You have errors in the choice of random seeds
- A Student's t can be used if the estimated variable has a normal distribution (which may not be true)



Confidence intervals

- ✓ Draw them whenever readable

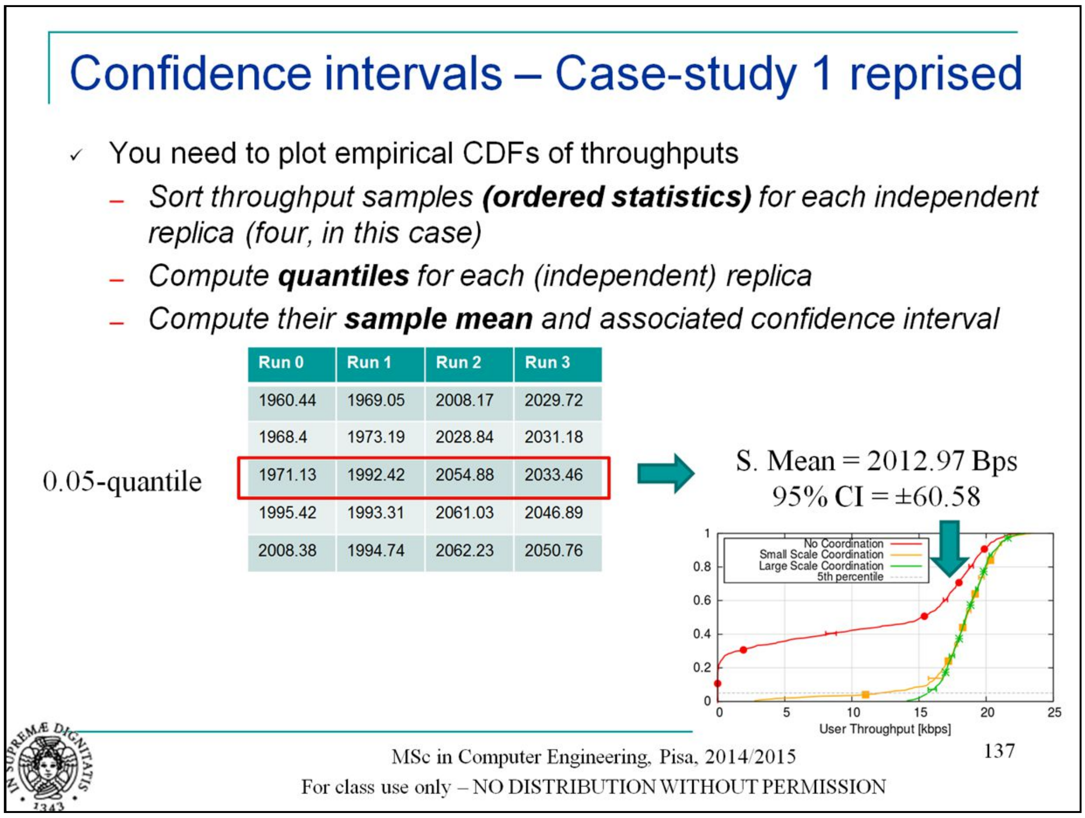


What confidence should I draw in my graphs? Good choices are 90%, 95% or 98% (the higher the confidence, the larger the interval).

If the intervals are too large, then measures may lose their meaningfulness. In this case, just **increase the number of independent replicas**, and see what happens.

Obviously enough, if you can rely on a **simulation automation software**, you can instruct it to perform as many replicas as required to have a small enough confidence interval (say, 10% of the mean), with a given maximum number (say, 30 – beyond that number you don't get any improvement).

There are cases when confidence intervals are **large**. It happens, for instance, when queueing systems are **close to saturation**, and the delays may vary greatly from one replica to another. In this case, there is little that you can do but acknowledge this fact.

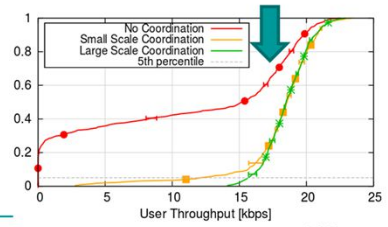


Confidence intervals – Case-study 1 reprised

- ✓ You need to plot empirical CDFs of throughputs
  - *Sort throughput samples* ***(ordered statistics)*** *for each independent replica (four, in this case)*
  - *Compute* ***quantiles*** *for each (independent) replica*
  - *Compute their* ***sample mean*** *and associated confidence interval*

	Run 0	Run 1	Run 2	Run 3
	1960.44	1969.05	2008.17	2029.72
	1968.4	1973.19	2028.84	2031.18
0.05-quantile	1971.13	1992.42	2054.88	2033.46
	1995.42	1993.31	2061.03	2046.89
	2008.38	1994.74	2062.23	2050.76

S. Mean = 2012.97 Bps
95% CI = ±60.58



How did we obtain an ECDF plot with confidence intervals?