Types of faults

All different faults cannot be enumerated

We can classify faults. Classifiction of faults is important because we can identify which fault tolerance mechanism protect us from a given class of faults.

Faults are classified according to basic viewpoints

The nature of faults

Natural faults (physical faults) relevant in hw

faults that are caused by natural phenomena without human participation.

Hw mainly breaks due to physical effects

Human-made faults (relavant in sw)

Result from human actions

Sw faults are related to programming

Prevention technologies are completely different:

Human-made faluts: rigorous development, testing, …

Natural faults: high quality material, optimize operation condition (temperature), shield the hardware, cooling ….

Phase of creation

when was the fault or the reason for the fault created

Development faults: the fault is part of the design

Operational faults: the fault occur during the use phase

System boundary

Internal: originate inside the system boundary

External: originate outside the system boundary and propagates errors into the system

Dimension

Hardware: affects the hardware / origin in the hardware

Software: affect the software (programs or data)

Objective

Non-malicious: introduced without malicious objectives

Malicoius: faults introduced with the malicious objective to alter the functioning of the system during use

Intent

Deliberate faults: due to intended actions that are wrong and cause faults (bad decision)

Non deliberate faults: due to mistakes, that is, unintended actions of which the developer, operator, maintainer, etc. is not aware

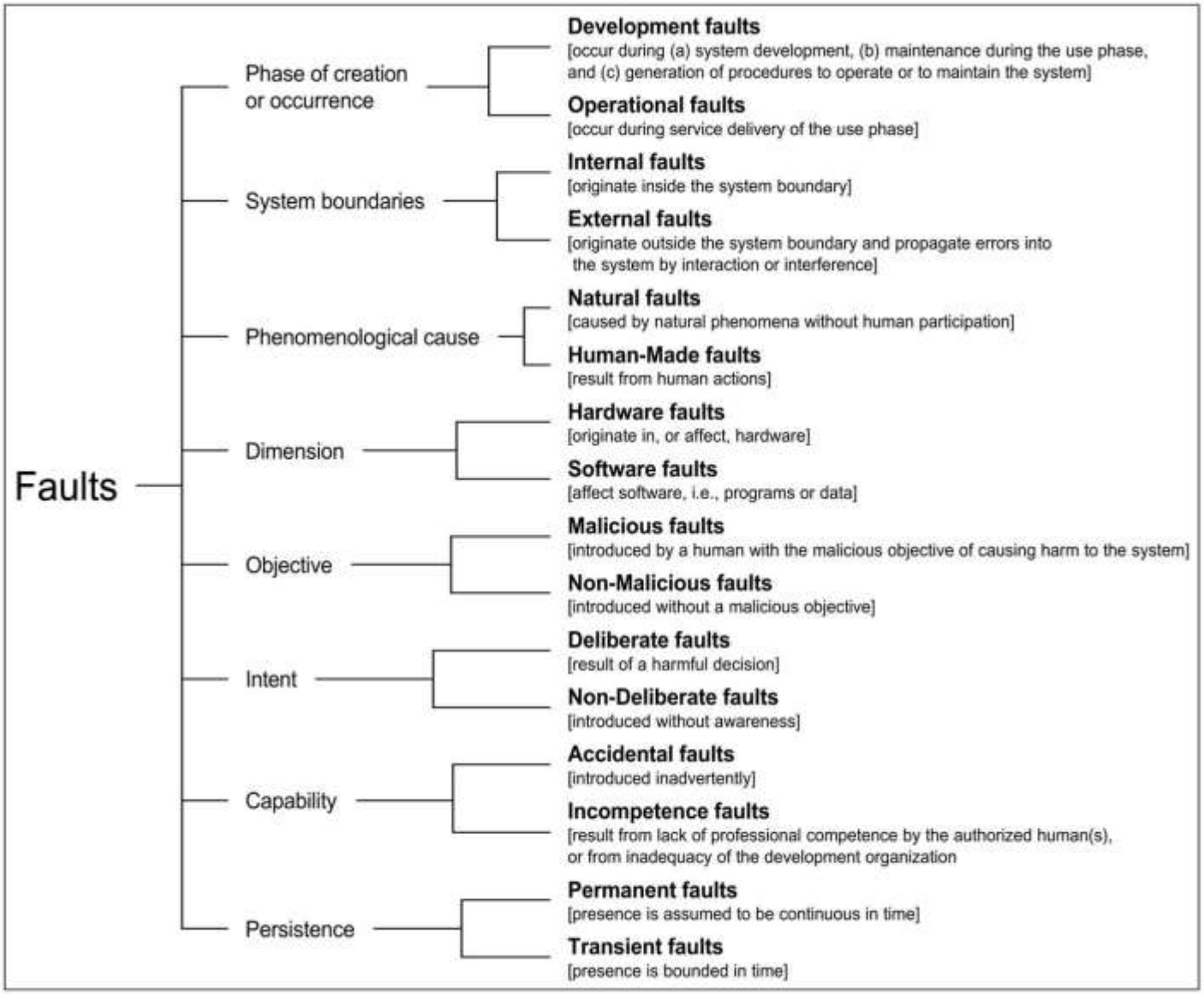
Temporal persistence

Temporary faults: a fault that can appear and disappear within a very short period of time. Faults that go away from themselves (short power outages)

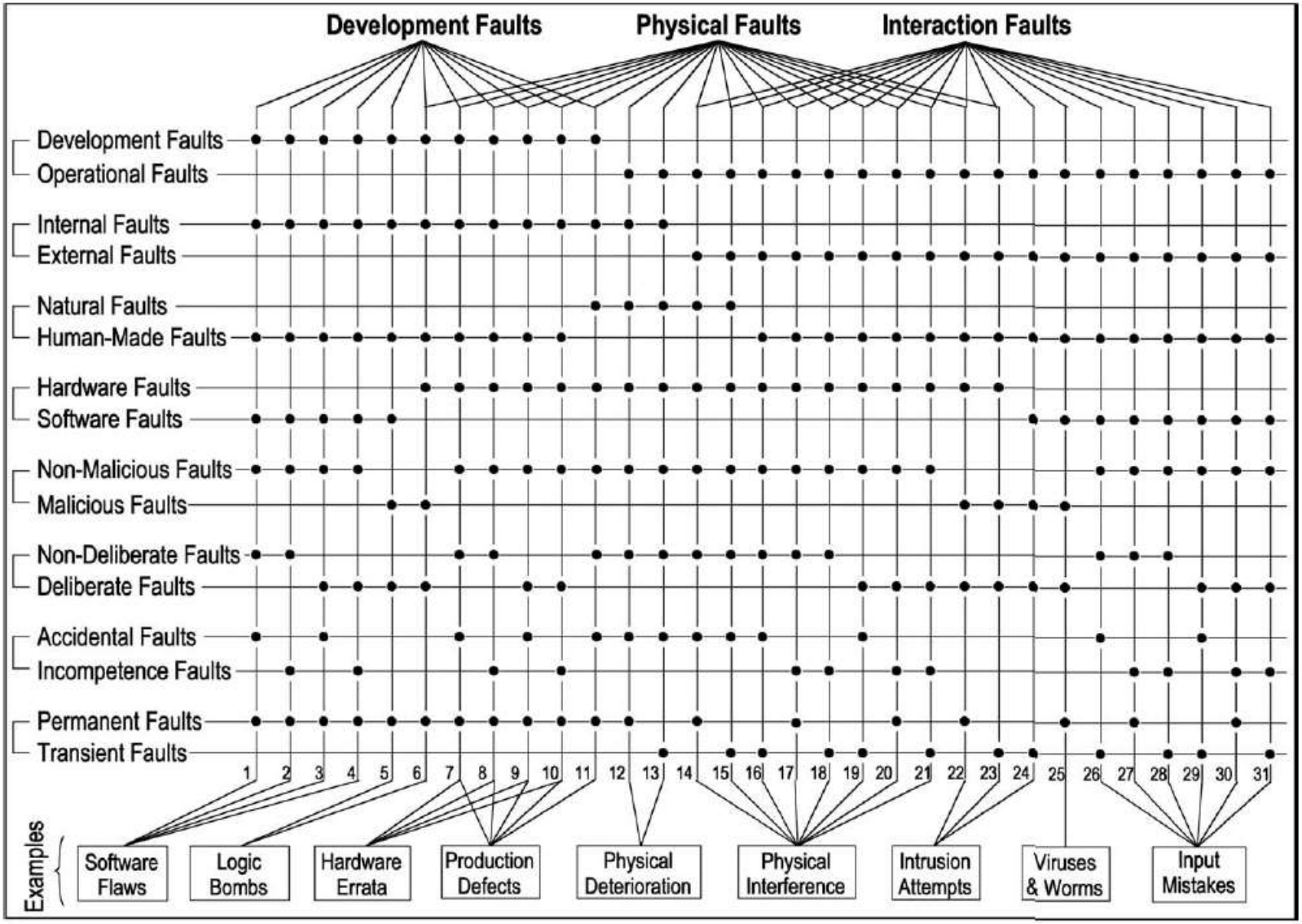
Permanent faults: a fault continuous and stable.
It remains in existence if no corrective action is taken. Disk sector damage.

Faults are classified according to eight basic viewpoints

From A. Avizienis, J.C. Laprie, B. Randell, C. Landwehr. Basic Concepts and Taxonomy of Dependable and Secure Computing, IEEE Transactions on Dependable and Secure Computing, Vol. 1, N. 1, 2004



Classification schema



names of some illustrative fault

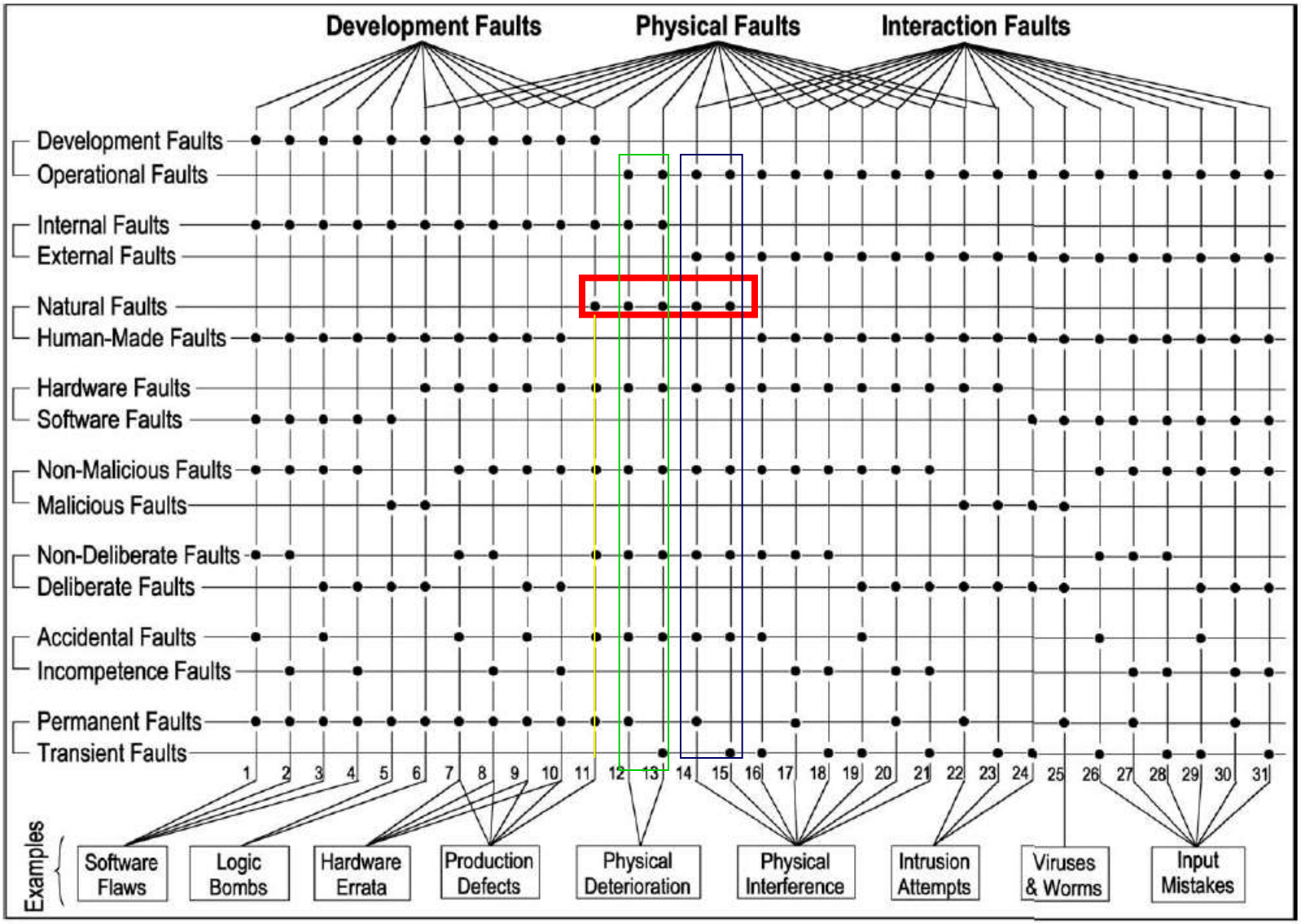
Identified combinations
belong to three major partially overlapping groupings

Development faults
that include all fault classes occurring during development

Physical faults
that include all fault classes that affect hardware

Interaction faults
that include all external faults.

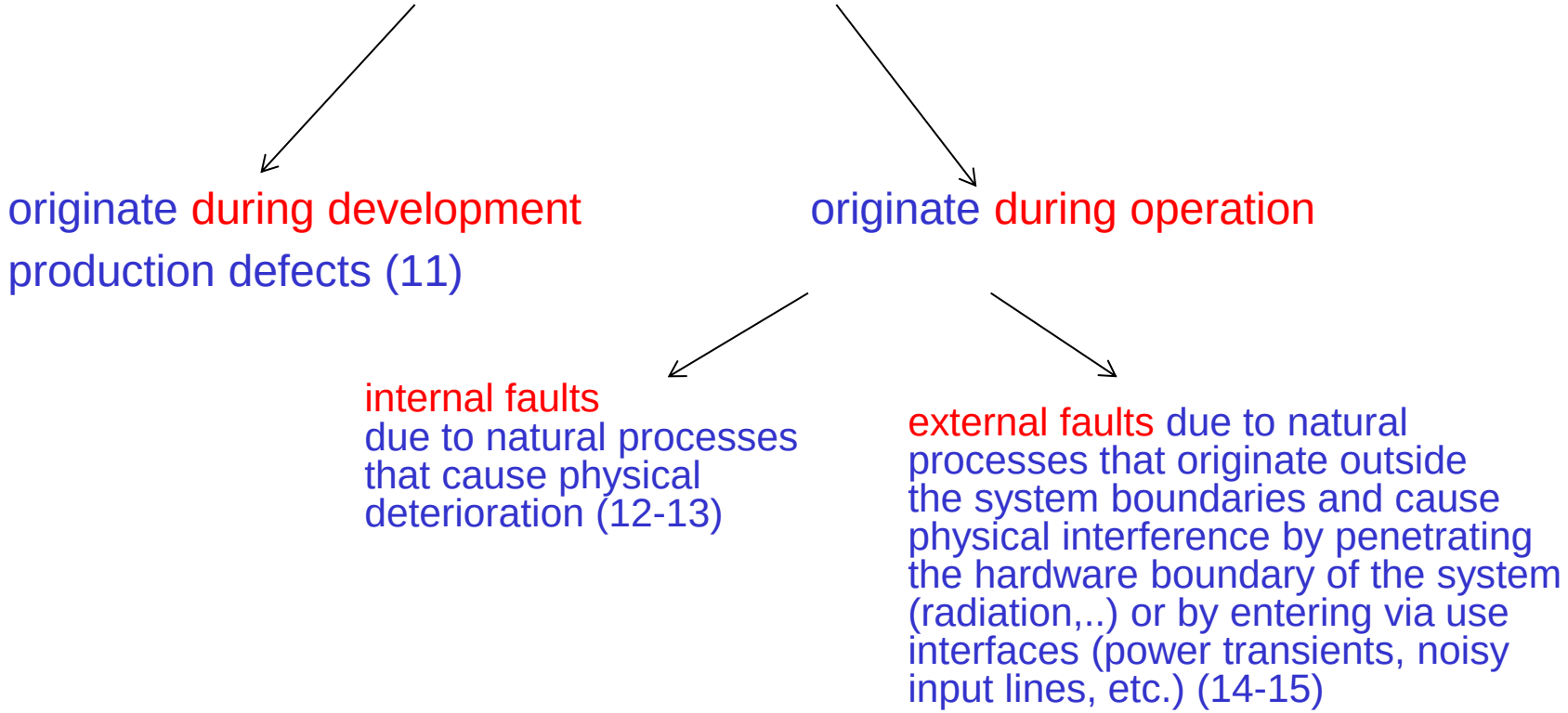
Natural faults



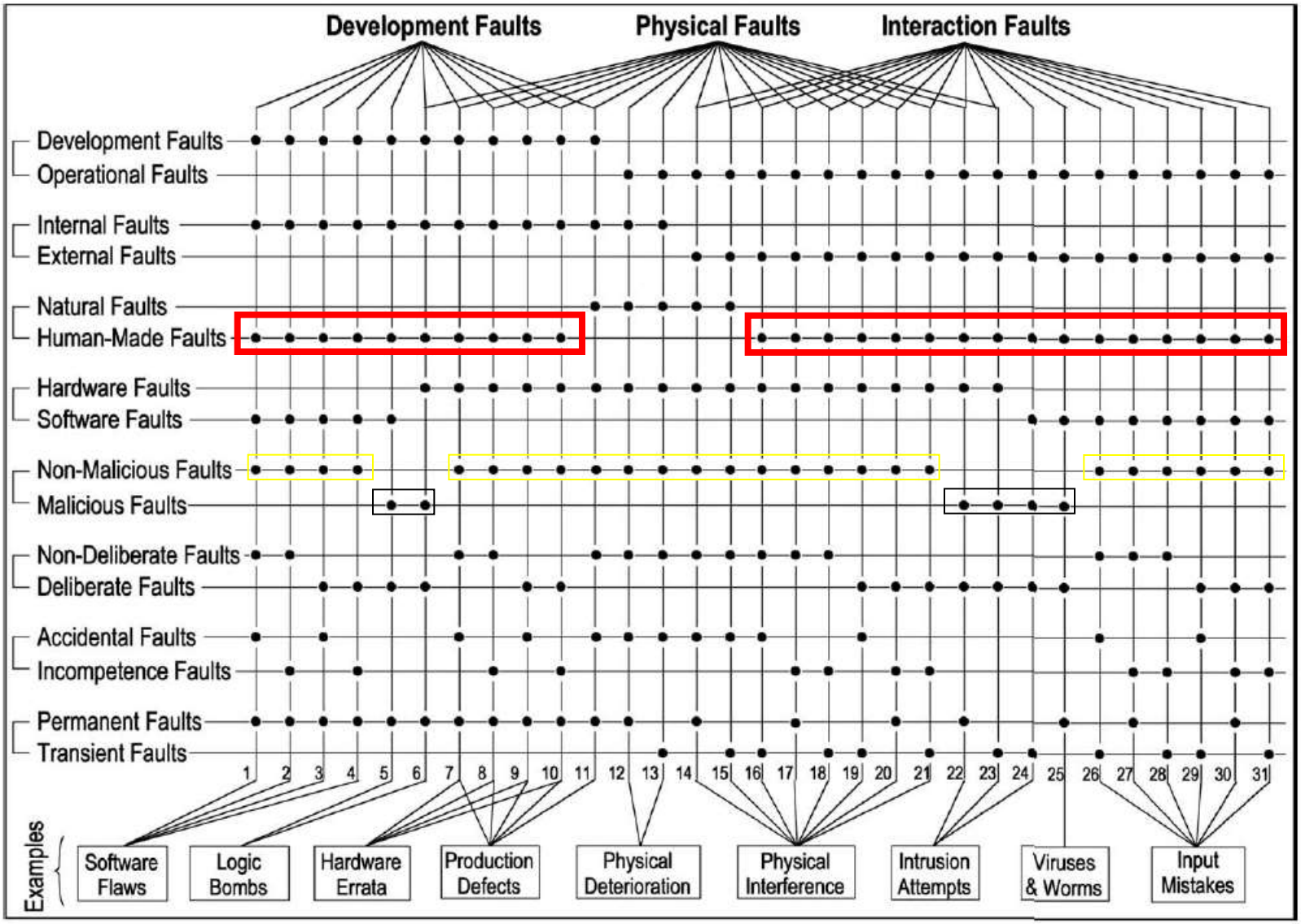
From A. Avizienis, J.C. Laprie, B. Randell, C. Landwehr. Basic Concepts and Taxonomy of Dependable and Secure Computing, IEEE Transactions on Dependable and Secure Computing, Vol. 1, N. 1, 2004

Natural faults

Natural faults (11-15) are physical (hardware) faults



Human-Made Faults



From A. Avizienis, J.C. Laprie, B. Randell, C. Landwehr. Basic Concepts and Taxonomy of Dependable and Secure Computing, IEEE Transactions on Dependable and Secure Computing, Vol. 1, N. 1, 2004

Human-Made Faults

result from human actions

Malicious faults, introduced during either system development with the objective to cause harm to the system during its use (5-6), or directly during use (22-25).

Non-malicious faults introduced without malicious objectives (1-4, 7-21, 26-31)

Deliberate faults: faults that are due to bad decisions, that is, intended actions that are wrong and cause faults (3, 4, 9, 10, 19-21, 29-31).

Non-deliberate faults that are due to mistakes, that is, unintended actions of which the developer, operator, maintainer, etc. is not aware (1, 2, 7, 8, 16-18, 26-28);

Development faults (3, 4, 9, 10) result generally from tradeoffs, either aimed at preserving acceptable performance, at facilitating system utilization, or induced by economic considerations

Interaction faults (19-21, 29-31) may result from the action of an operator either aimed at overcoming an unforeseen situation, or deliberately violating an operating procedure without having realized the possibly damaging consequences of this action.

Deliberate, nonmalicious faults are often recognized as faults only after an unacceptable system behavior; thus, a failure has ensued.

The developer(s) or operator(s) did not realize at the time that the consequence of their decision was a fault.

Malicious faults are all deliberate faults

Development physical (hardware) faults: microprocessor faults discovered after production (named Errata). They are listed in specification updates

Human-Made Deliberate Non-malicious Faults

Not all mistakes and bad decisions by nonmalicious persons are accidentals.

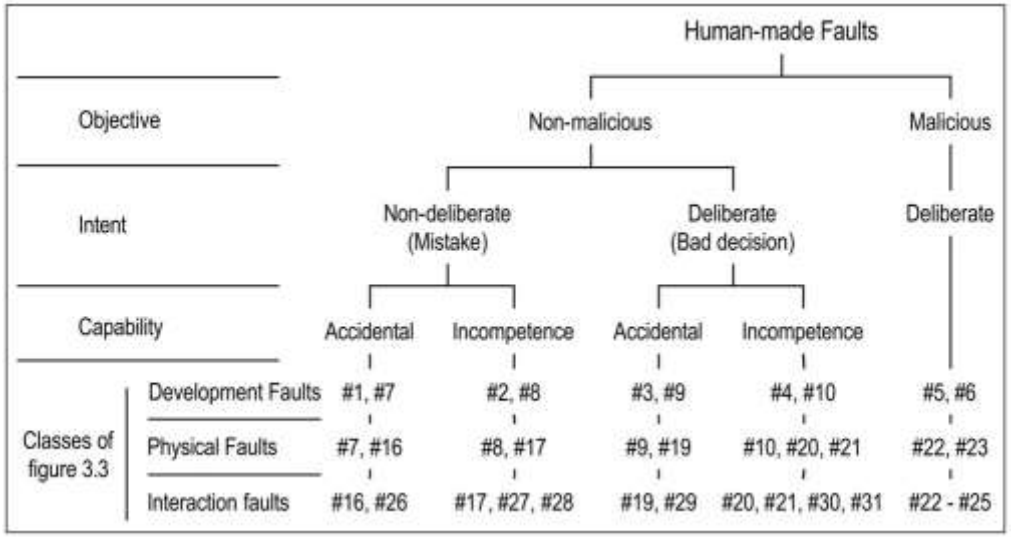
A further partitioning is introduced

accidental faults

incompetence faults

result from lack of professional competence or inadequasy of the development organization

How to recognize incompetence faults? Important when consequences that lead to economic losses or loss of human life.



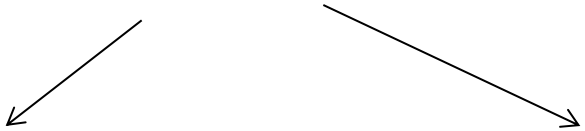
From A. Avizienis, J.C. Laprie, B. Randell, C. Landwehr. Basic Concepts and Taxonomy of Dependable and Secure Computing, IEEE Transactions on Dependable and Secure Computing, Vol. 1, N. 1, 2004

Malicious faults

Malicious human-made faults are introduced with the malicious objective to alter the functioning of the system during use.

The goals of such faults are:

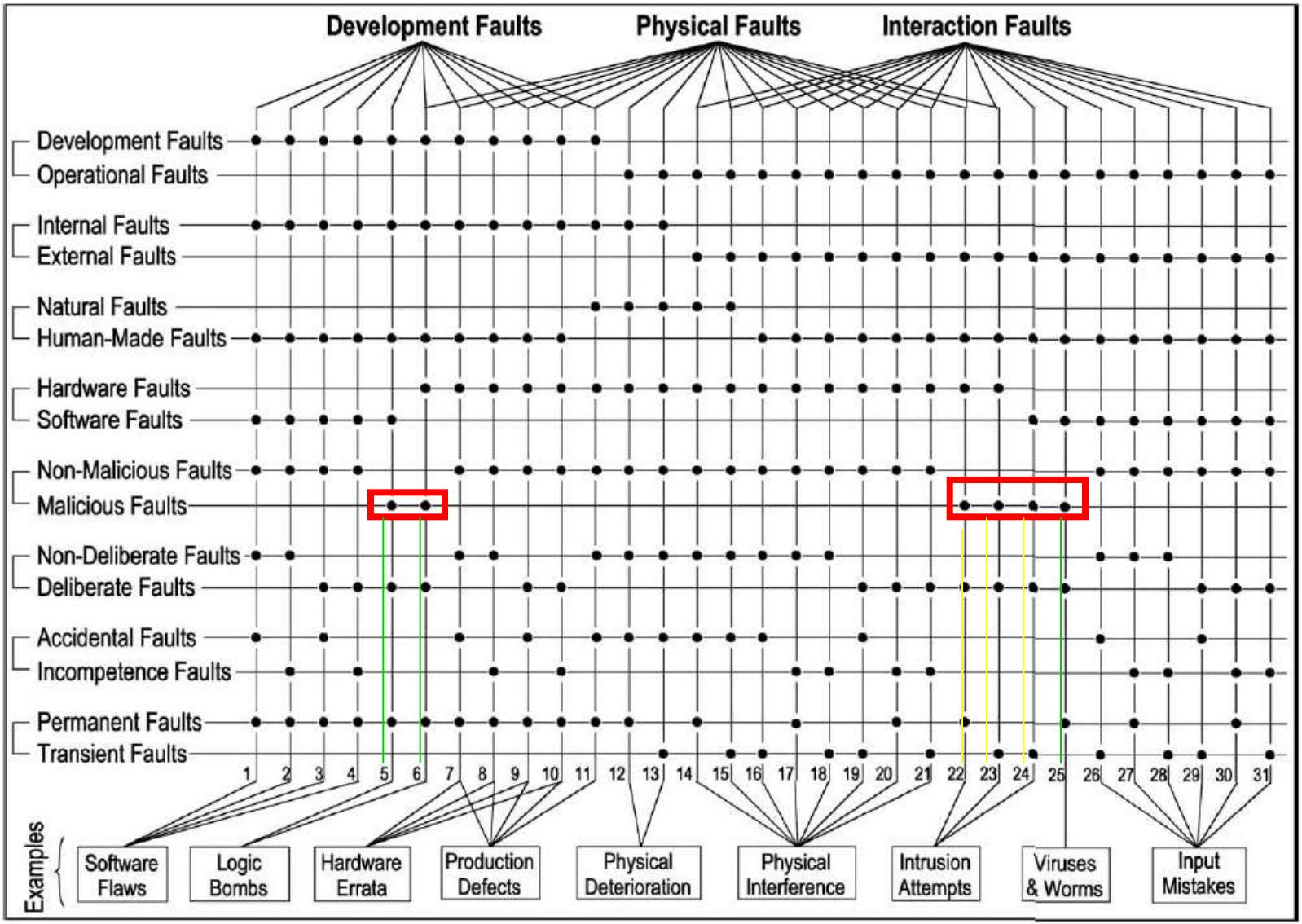
- to disrupt or halt service, causing denials of service;
- to access confidential information; or
- to improperly modify the system.



Developent faults
malicious logic faults (5,6) such as Trojan horses, logic or timing bombs, and trapdoors, as well as operational faults (25) such as viruses, worms, or zombies.

Operational external faults
intrusion attempts (22-24). The external character of intrusion attempts does not exclude the possibility that they may be performed by system operators or administrators who are exceeding their rights, and intrusion attempts may use physical means to cause faults: power fluctuation, radiation, wire-tapping, heating/cooling, etc.

Human-Made Malicious faults



From A. Avizienis, J.C. Laprie, B. Randell, C. Landwehr. Basic Concepts and Taxonomy of Dependable and Secure Computing, IEEE Transactions on Dependable and Secure Computing, Vol. 1, N. 1, 2004

Examples

An “exploit” is a software script that will exercise a system vulnerability and allow an intruder to gain access to, and sometimes control of, a system. Invoking the exploit is an operational, external, human-made, software, malicious interaction fault (24-25).

The vulnerability that an exploit takes advantage of is typically a software flaw (e.g., an unchecked buffer) that could be characterized as a developmental, internal, human-made, software, nonmalicious, nondeliberate, permanent fault (1-2).

Heating the RAM with a hairdryer to cause memory errors that permit software security violations would be an external, human-made, hardware, malicious interaction fault (22-23).

Interaction Faults

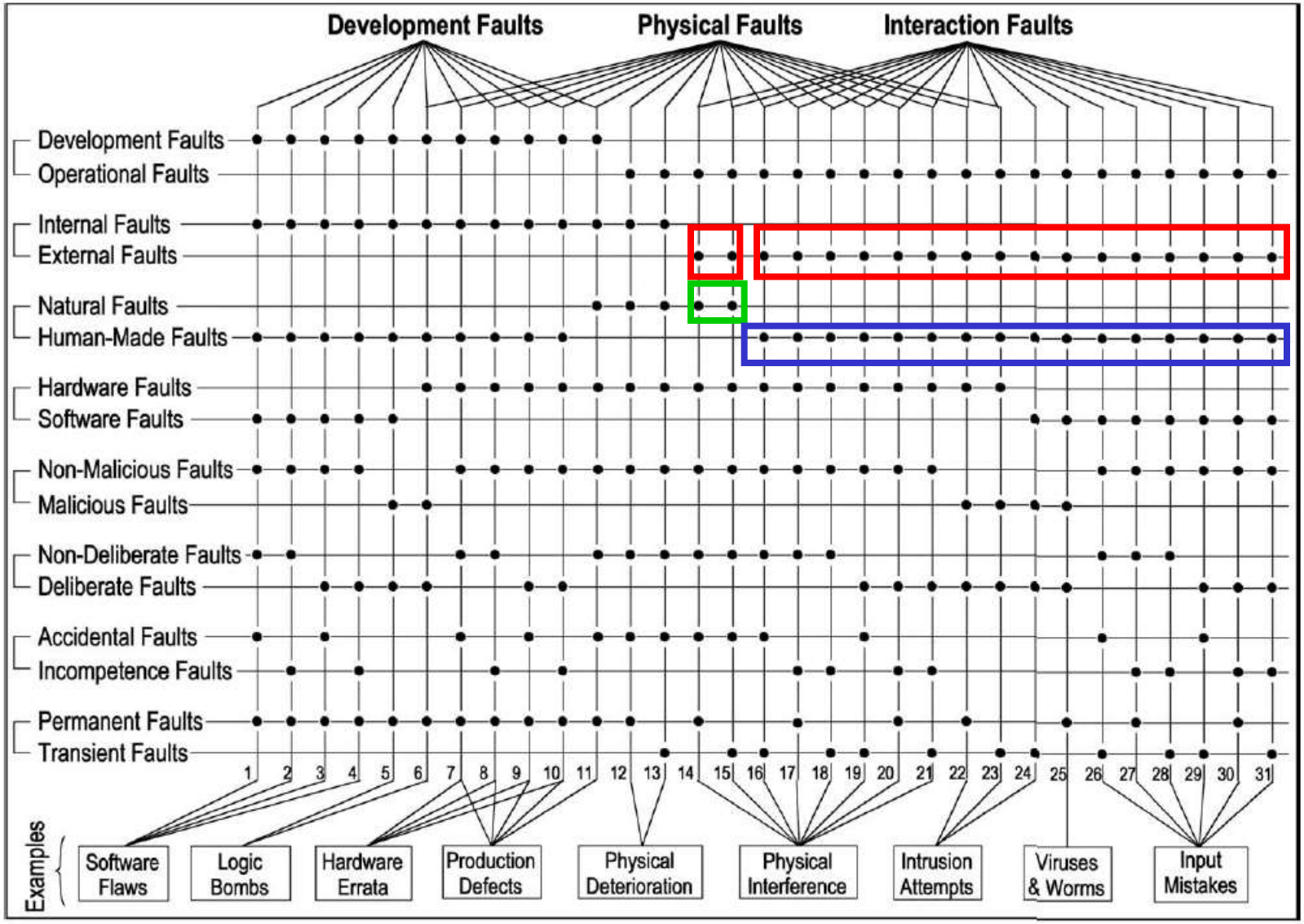
occur during the **use phase**, therefore they are all **operational faults**. They are caused by elements of the use environment interacting with the system; therefore, they are all external.

Human-made (16-31)
most classes originate due to some human action in the use environment

External natural faults (14-15)
caused by cosmic rays, solar flares, etc. Here, nature interacts with the system without human participation.

Configuration faults (i.e., wrong setting of parameters that can affect security, networking, storage, middleware, etc.): a broad class of human-made operational faults. Such faults can occur during configuration changes performed during adaptive or augmentative maintenance performed concurrently with system operation

Interaction faults



From A. Avizienis, J.C. Laprie, B. Randell, C. Landwehr. Basic Concepts and Taxonomy of Dependable and Secure Computing, IEEE Transactions on Dependable and Secure Computing, Vol. 1, N. 1, 2004

A common feature of interaction faults is that, in order to be "successful," they usually necessitate the prior presence of a vulnerability, i.e., an internal fault that enables an external fault to harm the system.

Vulnerabilities can be development or operational faults; they can be malicious or nonmalicious, as can be the external faults that exploit them.

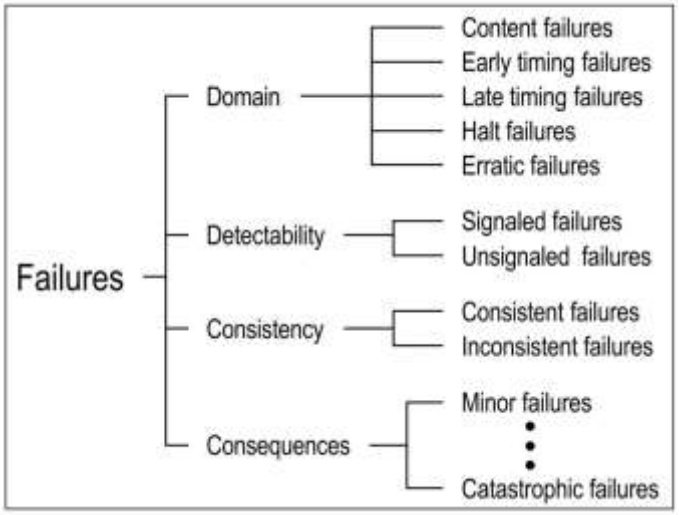
There are interesting and obvious similarities between an intrusion attempt and a physical external fault.

A vulnerability can result from a deliberate development fault, for economic or for usability reasons, thus resulting in limited protections, or even in their absence.

Failures

The failure modes characterize incorrect service according to four viewpoints:

1. the failure domain,
2. the consistency of failures,
3. the detectability of failures and
4. the consequences of failures on the environment.



From A. Avizienis, J.C. Laprie, B. Randell, C. Landwehr. Basic Concepts and Taxonomy of Dependable and Secure Computing, IEEE Transactions on Dependable and Secure Computing, Vol. 1, N. 1, 2004

Failures

1. Failure domain viewpoint:

content failures
the content of the information delivered at the service interface deviates from implementing the system function.

timing failures
the time of arrival or the duration of the information delivered at the service interface deviates from implementing the system function.

halt failure, or simply halt,

when the service is halted (the external state becomes constant, i.e., system activity, if there is any, is no longer perceptible to the users);
a special case of halt is silent failure, or simply silence, when no service at all is delivered at the service interface (e.g., no messages are sent in a distributed system).

erratic failures
when a service is delivered (not halted), but is erratic (e.g., babbling -the system repeatedly fails).

2. Consistency viewpoint (when a system has two or more users):

consistent failures.
the incorrect service is perceived identically by all system users.

inconsistent failures.
some or all system users perceive differently incorrect service (some users may actually perceive correct service); inconsistent failures are usually called, Byzantine failures.

3. Detectability viewpoint:

signaled failures: when the failures are detected and signaled by a warning signal to the users (based on a detecting mechanisms in the system that check the correctness of the delivered service).

unsignaled failures: otherwise.

The failure detecting mechanism has two failure modes:
- false alarm (signaling a loss of function when no failure has actually occurred)
- unsignaled failure (not signaling a function loss).

When the occurrence of service failures result in reduced modes of service, the system signals a degraded mode of service to the user(s).

Failures

4. Consequences viewpoint:

grading the consequences of the failures upon the system environment enables failure severities to be defined.

Two limiting levels can be defined according to the relation between the benefit provided by the service delivered in the absence of failure, and the consequences of failures:

minor failures

the harmful consequences are of similar cost to the benefits provided by correct service delivery

catastrophic failures

the cost of harmful consequences is orders of magnitude, or even incommensurably, higher than the benefit provided by correct service delivery

Errors

An error can be:

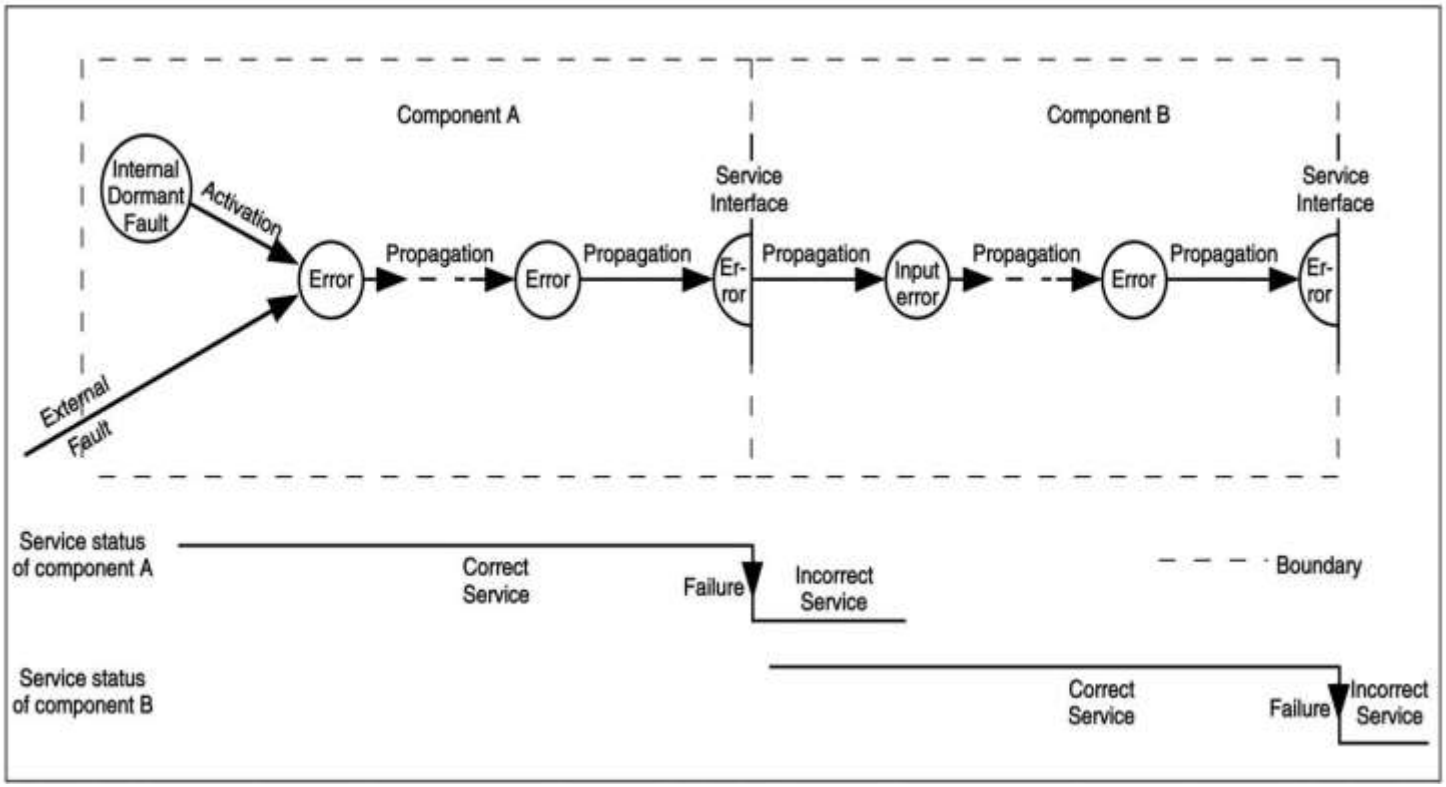
- detected if its presence is indicated by an error message or error signal.
- latent if it is present but not detected

Whether or not an error will actually lead to a failure depends on two factors:

1. The structure of the system, and especially the nature of any redundancy that exists in it: protective redundancy, introduced to provide fault tolerance, that is explicitly intended to prevent an error from leading to service failure. Unintentional redundancy (it is in practice difficult if not impossible to build a system without any form of redundancy) that may have the same presumably unexpected result as intentional redundancy.
2. The behavior of the system: the part of the state that contains an error may never be needed for service, or an error may be eliminated (e.g., when overwritten) before it leads to a failure.

Some faults (e.g., a burst of electromagnetic radiation) can simultaneously cause errors in more than one component. Such errors are called multiple related errors. Single errors are errors that affect one component only.

Chain of threats: Faults-Errors-Failures



From A. Avizienis, J.C. Laprie, B. Randell, C. Landwehr. Basic Concepts and Taxonomy of Dependable and Secure Computing, IEEE Transactions on Dependable and Secure Computing, Vol. 1, N. 1, 2004

Chain of threats

A fault is active when it produces an error; otherwise, it is dormant.

An active fault is either

- an internal fault that was previously dormant and that has been activated by the computation process or environmental conditions, or
- an external fault.

Fault activation:
is the application of an input (the activation pattern) to a component that causes a dormant fault to become active.

Most internal faults cycle between their dormant and active states.

Code is full of unactivated faults (faults are named bugs)

Testing can check the presence of faults, not the absence of faults.

We are interested in faults that may be activated.

We use the control flow and test coverage.

Chain of threats

Error propagation within a given component (i.e., internal propagation) is caused by the computation process.

An error is successively transformed into other errors.

Error propagation from component A to component B that receives service from A (i.e., external propagation) occurs when, through internal propagation, an error reaches the service interface of component A.

At this time, service delivered by A to B becomes incorrect, and the ensuing service failure of A appears as an external fault to B and propagates the error into B via its use interface.

Chain of threats

A service failure occurs when an error is propagated to the service interface and causes the service delivered by the system to deviate from correct service.

The failure of a component causes a permanent or transient fault in the system that contains the component.

Service failure of a system causes a permanent or transient external fault for the other system(s) that receive service from the given system.

Chain of threats

Given a system with defined boundaries, a single fault is a fault caused by one adverse physical event or one harmful human action.

Multiple faults are two or more concurrent, overlapping, or sequential single faults whose errors overlap in time, that is, the errors due to these faults are concurrently present in the system.

Consideration of multiple faults leads one to distinguish independent faults, that are attributed to different causes, and related faults, that are attributed to a common cause.

Related faults generally cause similar errors, i.e., errors that cannot be distinguished by whatever detection mechanisms are being employed

Independent faults usually cause distinct errors.

However, it may happen that independent faults (especially omissions) lead to similar errors, or that related faults lead to distinct errors.

Dependable system

Point 1)

Assumptions on how the system is used: very important

External faults during normal operation, caused by wrong assumptions on the operational conditions

Sometimes operational conditions are underspecified.

Point 2)

Faults are unexpected events. Something is happening in the system that we did not plan before.

How can we build a system that tolerates faults if we do not know faults?

«information from literature (knowledge of fault classes) and from experience allows the user to decide which faults should be included in the dependability specification»

Specification of the fault free system + fault assumption

Point 3)
Fault classes are relevant to choose the dependability mean.

Example: temporary/permanent faults

How can you deal wih temporary faults?

In the case of short power outages: extra battery can be used as fault tolerance mechanism

In case of network connections problems (network and connectivity are assumed temporary problem): retry can be used as fault tolerant mechanism.

Additional problem: if the net is gone, retry not help. Reconnect to a not existing entity problem .

How can you deal wih permanent faults?
Redundancy (you need to have a spare sw or hw component)

Exceptions in programs

How can you deal wih exceptions?
Catch is used as fault tolerant mechanism.

Examples of safety critical systems accidents

Arianne 5 - Flight 501

Ariane 5 is a European heavy lift launch vehicle that is a part of the Ariane rocket family, an expendable launch system used to deliver payloads into geostationary transfer orbit or low Earth orbit .

" The morning of the 4th of June 1996 was partially cloudy at Kourou in Guyana as the European Space Agency (ESA) prepared for the first launch of the French-built Ariane 5 rocket. The rocket lifted off at 09:34. Just 37 seconds later, the rocket veered on its side and began to break up. The range safety mechanism identified the impending catastrophe and initiated explosive charges that blew up the rocket to prevent further damages and possible casualties. An investigation by the ESA determined that the accident was caused by a software 'bug'. This is the story of that bug. "

From: The Bug That Destroyed a Rocket, Mordechai Ben-Ari. *SIGCSE Bulletin*, n. 2, 2008

ARIANE 5 Flight 501 failure, Inquiry Board Report, 1996 (http://sunnyday.mit.edu/accidents/Ariane5accidentreport.html)

- self-destruction started because boosters were ripping from the rocket on a populated area (explosion reduced the risk for population)

- on board computer reaction on the inertial system data

- the computer started to reconfigure the boosters accordingly

- this reconfiguration physically disrupted the boosters from the rocket body

Arianne 5

The data the computer collected were not data but diagnostic bit patterns which was output by some other subsystem that had an overflow error (e.g, failure messages). The control systems took this data.

There was a backup inertia system, but the backup systems died some milliseconds before.

Original problem:
the inertial system with the diagnostic bit patterns was from Arianne 4.
In Arianne 5, they reused the same software but they replaced the sensors.

"new sensor and old software that read these sensors"

The old software was not able to deal with large numbers produced by new sensors, we had an overflow problem.

This was never tested due to budget constraints.

In this example we have:

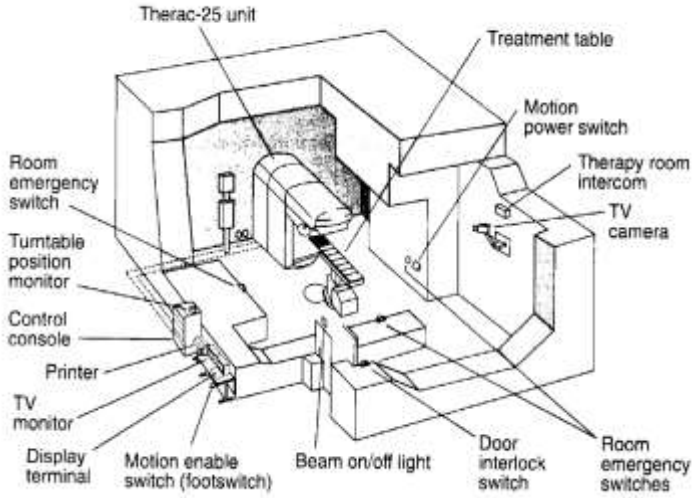
incompatible software versions, inertial system problem, unexpected data flow,

budget problems

Therac 25

Therac 25 is a machine for radiation therapy
(to treat cancer)

- Between June 1985 and January 1987 (at least) six patients received severe overdoses: two died shortly afterwards, two might have died but died because of cancer, the remaining two suffered of permanent disabilities



Functional principle:
"scanning magnets"are used to spread the beam and vary the beam energy

Two operation modes:

1) high power spreaded beam or

2) low power focus beam

Accident: high power mode (a lot of energy) without spreader plate activated
-> caused by software flaws

Software bug in the software control and there was a problem in the coordination between plain position sensors (microswitch failure) and control software

Second issue, the software was not independently reviewed, there was not failure mode analysis,….

Therac 25

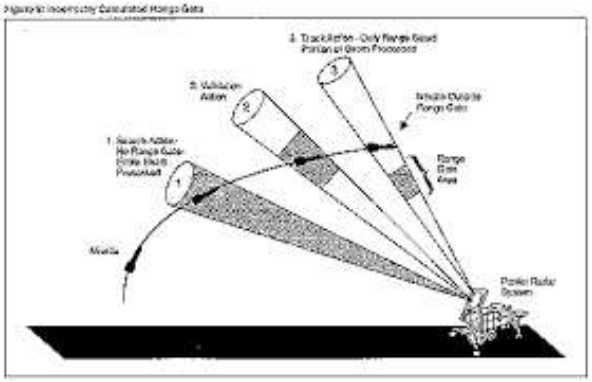
Lessons learned from Therac-25 accident:

- Accidents are the combination of distinct events, that lead to the disaster
- Overconfidence in software
- Unrealistic risk assessment
- Software-engineering practices not acceptable

Medical Devices: The Therac-25, Nancy Leveson, U. of Washington.
In *Safeware: System Safety and Computers*, Addison-Wesley, 1995

Patriot Missile Launcher (US Military)

During gulf war a Scud missile broke through the Patriot anti-missile defense barrier and hit American forces killing 28 people and injuring 98



Patriot missile launcher:

- mobile missile launcher
- radar sweeps the sky for threats. If an incoming threat is found, the launch of a missile is guided by the control station
- designed for a few hours of operations

Used for Scud defense operation, never designed for it, failed to intercept a Scud missile

Use of the system in an unexpected mode of operation. The station did not move, the system was at the same position for many days

Patriot Missile Launcher (US Military)

There was an aging problem in the software.The software run too long and started getting overflow, inaccuracy, …

Target velocity and time demanded as real values, was stored as 24 bit integers

with the advent of time this conversion loses accuracy (> 100 hours)

tracking of enemy missiles becomes inaccurate therefore faulty

The software problem was already known, and the update was delivered the next day

Air France AF 447(2009)

Airbus A330 flight from Rio de Janeiro to Paris disappeared over the ocean; it took one week to find the airplane (http://www.airfrance447.com/)

Problem:
bad weather conditions, flight at 10.000 m; air pressure sensors, used for speed computation, not designed for this flight heigh.

The plane did not know its speed. Speed is the main information you need and the autopilot did not know how to deal with it.

Flight speed instrumentation shut down and the autopilot shut down

The pilot seats for hours without doing anything, because the autopilot is doing the work. In a very short period of time (seconds), light are blinking, the autopilot shut down,
the pilot has to control the whole airplane

Information are requested manually from the pilot; the pilot was trained, but he has too much information to process

Pilot lost the control of the machine, and he tried to restart the autopilot (not successfull) to get control of the plane again.

Minimum standard you have to satisfy. :
more sensor (three at different positions from different vendors) - Dependability as a cost factor.

Observations

In real world, dependability problems are really subtle. There is a root cause that evolves. It propagates into the system, something happens in a subsystem, something else happins in another subsystem, …., and then we have a failure.

This is very complicated to predict; in this case fault tolerance can be usefull

System are designed to endure within a given operational conditions. It is very hard to anticipate the operational conditions correctly

In safety critical systems community, people are forced to document and publish the problems (accidents)

You have to publish problems, to document them, to analyse them to be sure that nobody else has the same problem again

In this way, data for dependability research can be collected