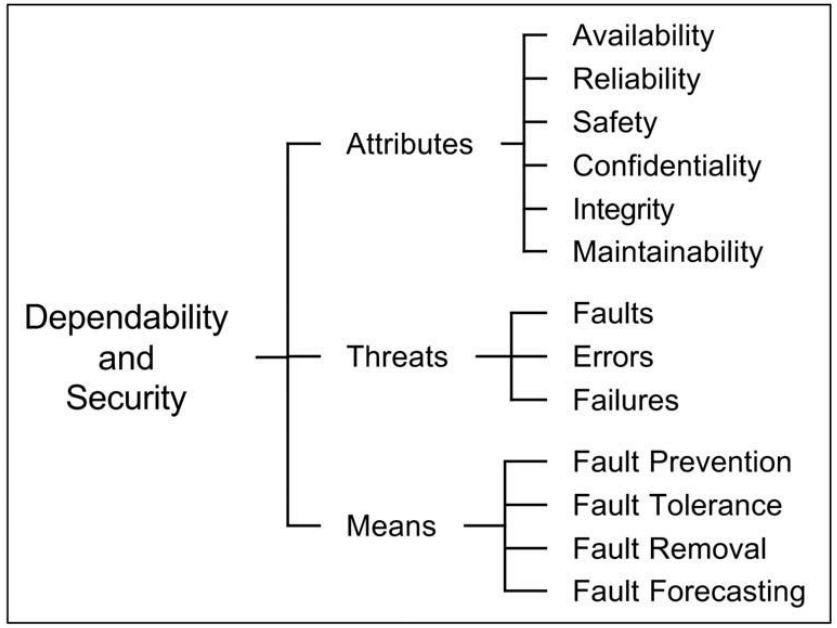
# Means for dependability

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

# Dependability tree




From [Avizienis et al., 2004]

# Means for achieving dependability

A combined use of methods can be applied as means for achieving dependability.

1. **Fault Prevention techniques**

   to prevent the occurrence and introduction of faults

   – rigorous developent, formal methods, testing, quality control methods, ... (write free of bugs code)

   – component screening, shielding, …

   (prevent to insert external faults is not possible)

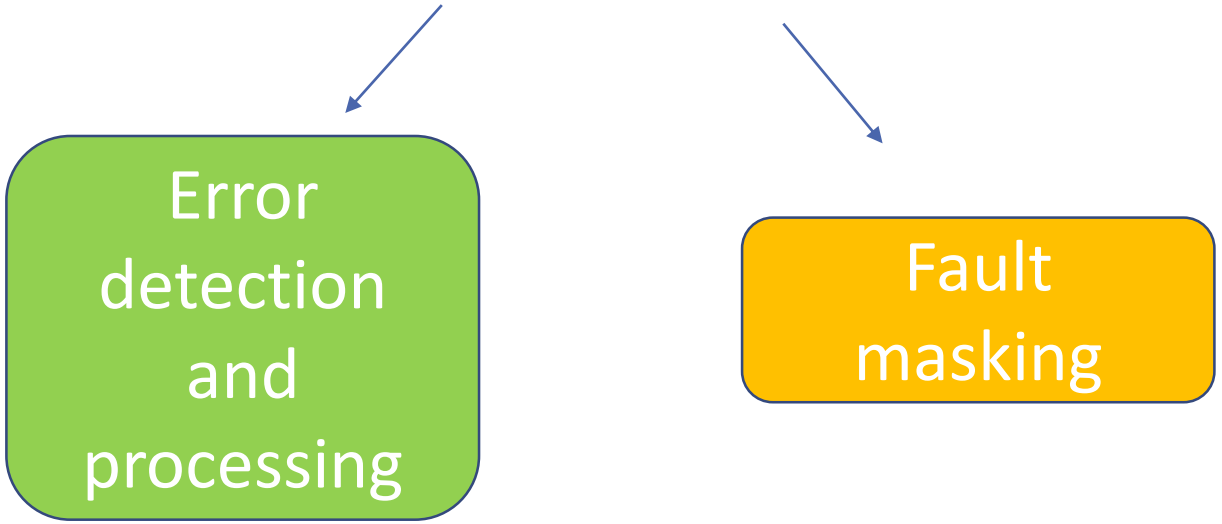
# Means for achieving dependability

2. **Fault Tolerance techniques**

deal with faults at run-time

(zero faults not possible)

deliver correct service in presence of activated faults and errors



# Means for achieving dependability

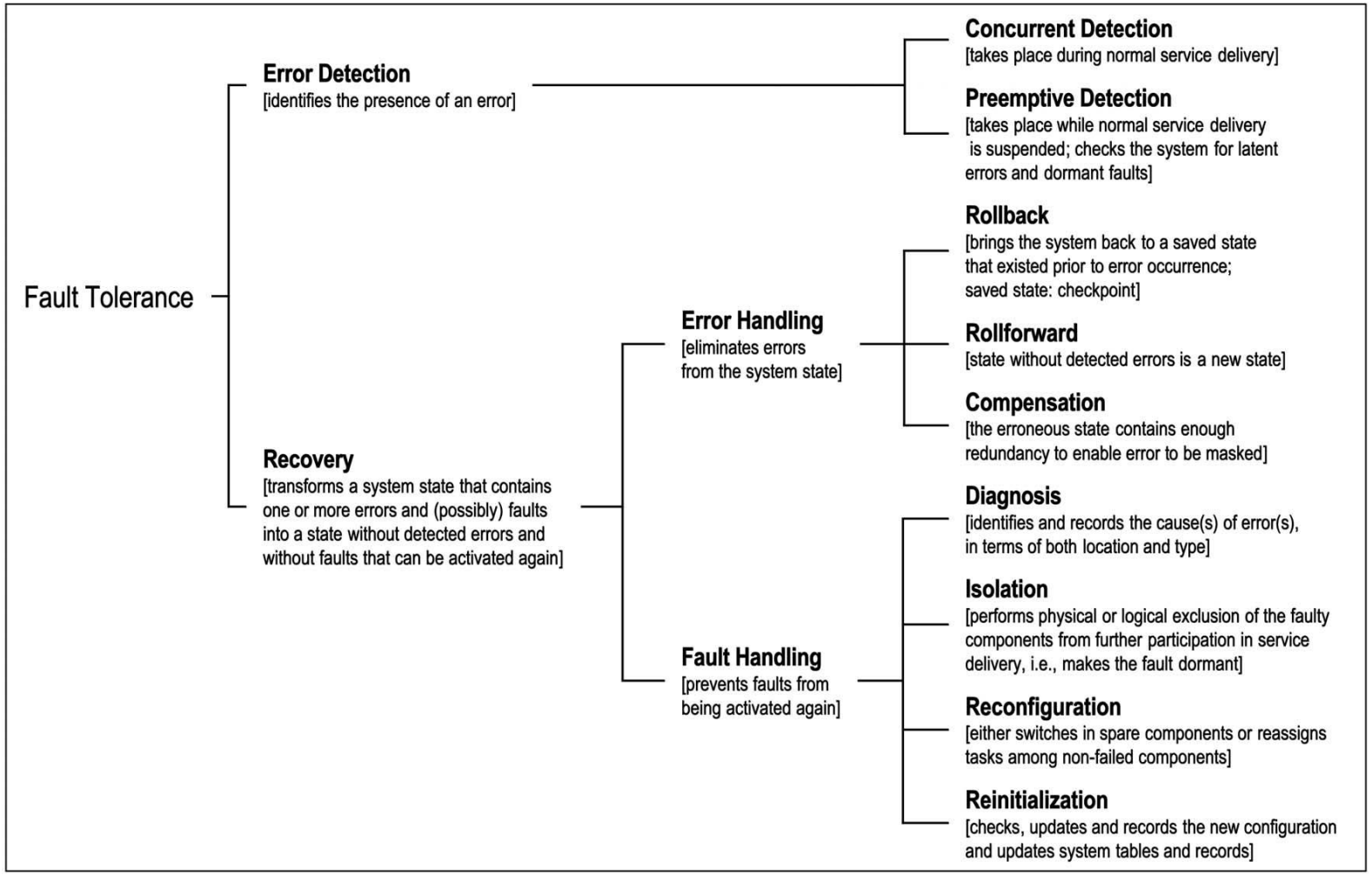
3. **Fault Removal techniques**

   remove faults in such a way that they are no more activated

4. **Fault Forecasting techniques**

   to estimate the present number, the future incidence, and the consequences of faults. Try to anticipate faults; do better design introducing fault tolerance techniques

# Organisation of fault tolerance




From [Avizienis et al., 2004]

# Techniques involved in fault tolerance

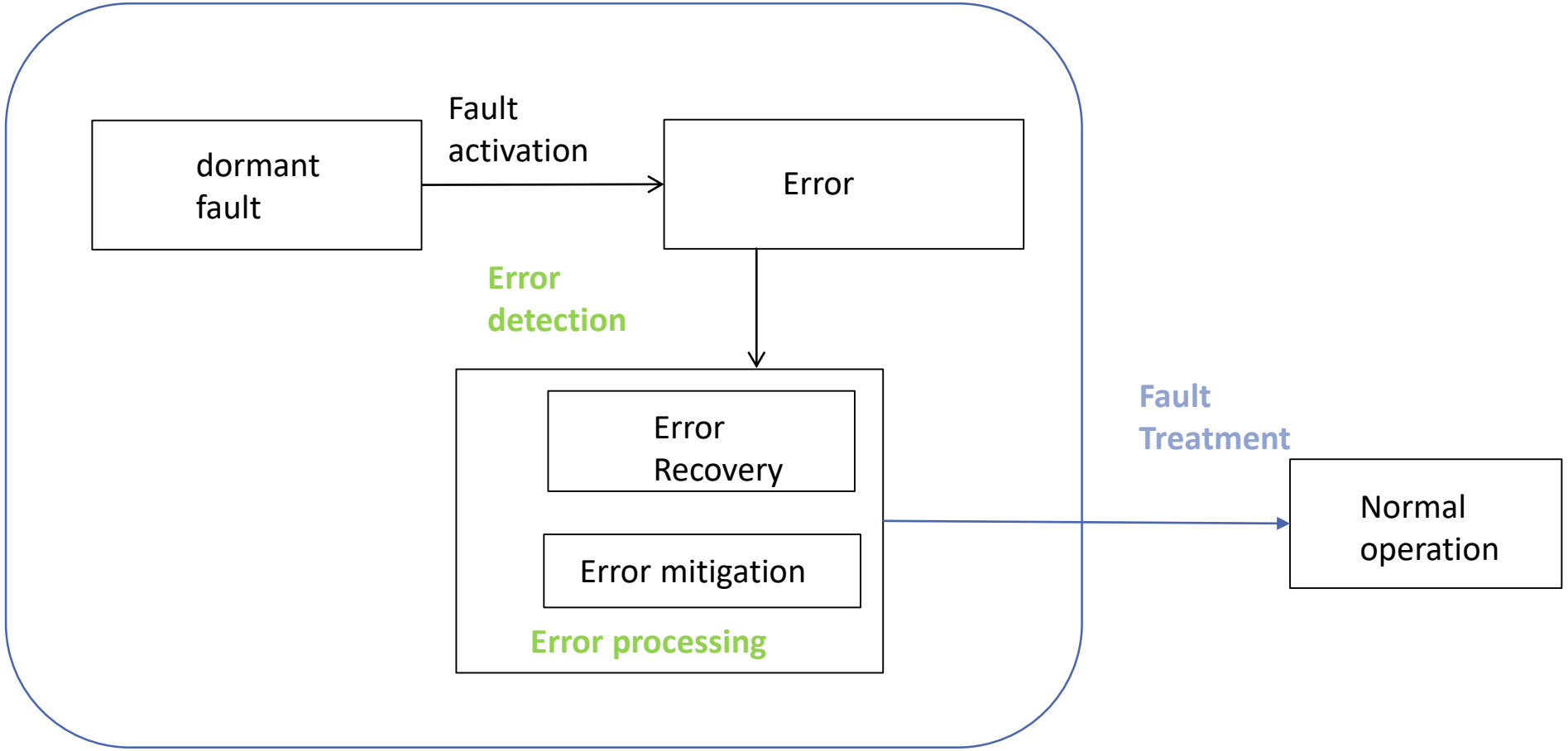
BASIC CONCEPT:
fault tolerance mechanisms detect error states (not faults)

Phases of fault tolerance:

- Error Detection
- Error Processing
  - Error recovery
  - Error mitigation
- Fault Treatment

# Techniques involved in fault tolerance

carried out via error detection, error processing and fault treatment



# Techniques involved in fault tolerance

UNIVERSITÀ DI PISA

**Fault treatment**

fix the original problem, in such a way that it never occurs again

Fault passivation

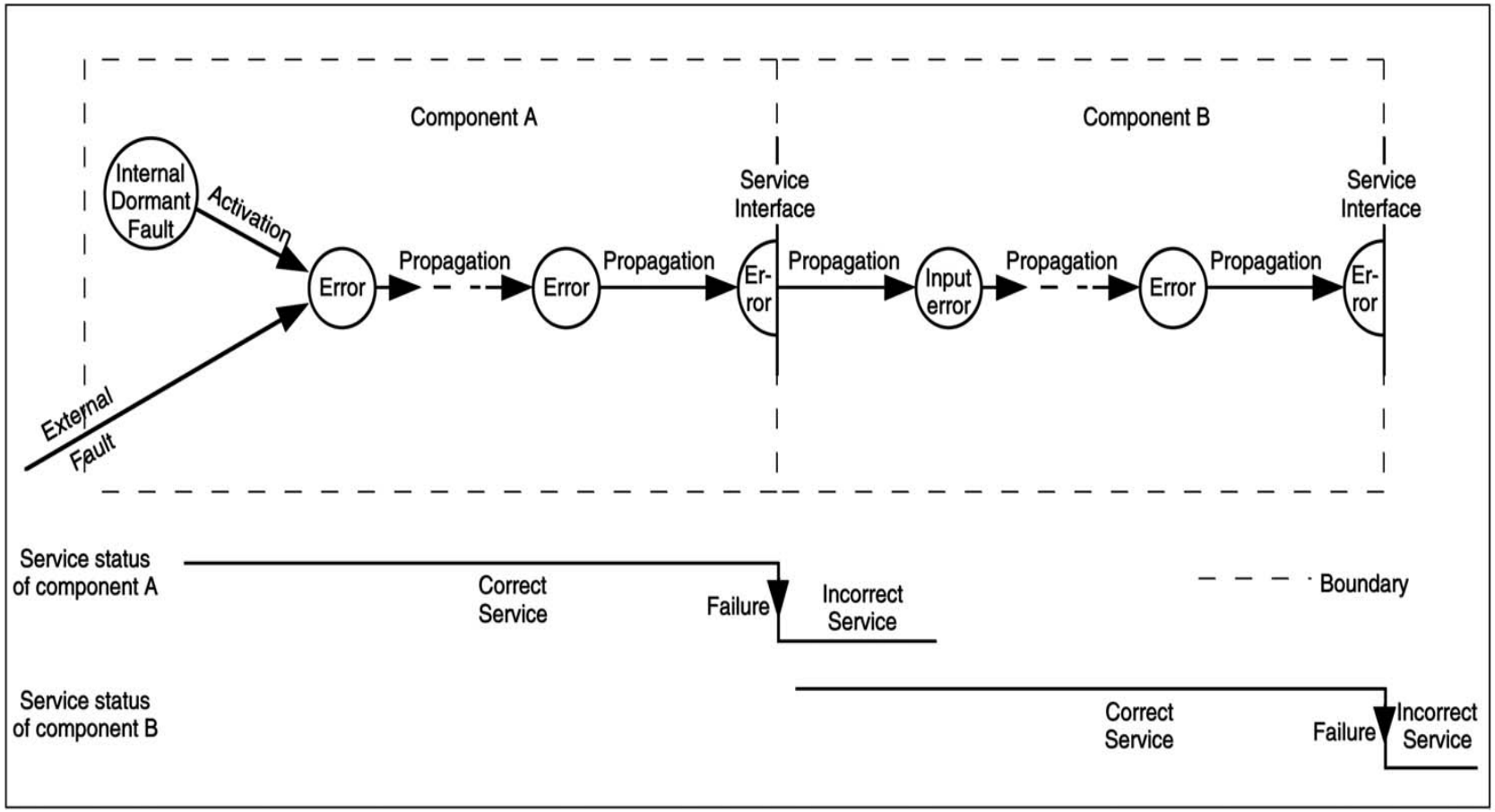
- Deactivate a corrupted memory module in a computer
- Broken computer no more used

Other important aspects

- **Damage confinement**
  before we start to use fault tolerance redundancy, we isolate the compromised components

- **Protective redundancy**
  additional components or processes that mask/correct errors or faults inside a system so they do not become failures. Signal the problem to the user.

# Chain of threats: Faults-Errors-Failures




From [Avizienis et al., 2004]

# Error detection: Types of checks

**Reasonableness Checks**

Acceptable ranges of variables
Acceptable transitions
Divide by 0
Probable results

…….

**Specification checks** (use the definition of "correct result")

Examples
Specification: find the solution of an equation
Check: substitute results back into the original equation

**Reversal Checks**

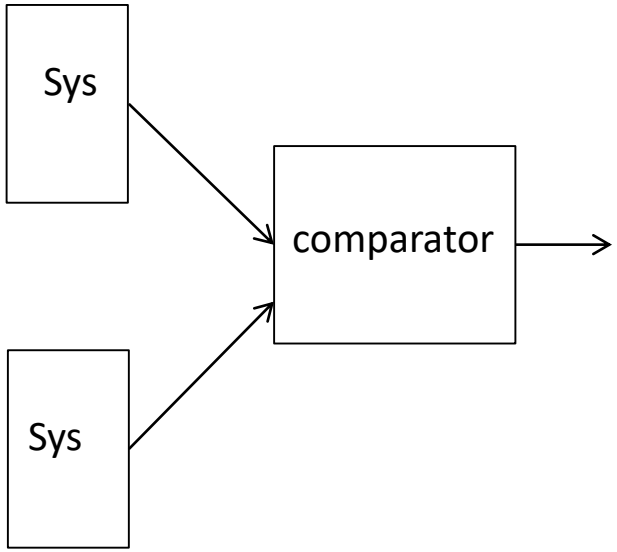
assume the specified function of the system is to compute a mathemathical function *output = F(input)*
if the function has an inverse function *F'(F(x))=x* we can compute *F'(output)* and verify that *F'(output) = input*

# Error detection: Types of checks

## Replication Checks

Based on copies and comparison of the results two or more copies

- a mechanism that compares them and declares an error if differ
- the copies must be unlikely to be corrupted together in the same way




Assumption on faults is very important.
most of the time single fault assumption

What is the fault model for sw? Same input, same bug in the software, they have a COMMON CAUSE FAILURE

In case of software fault, we do not tolerate the error (design diversity is needed) SW

What is the fault model for hw? Faults are independent.

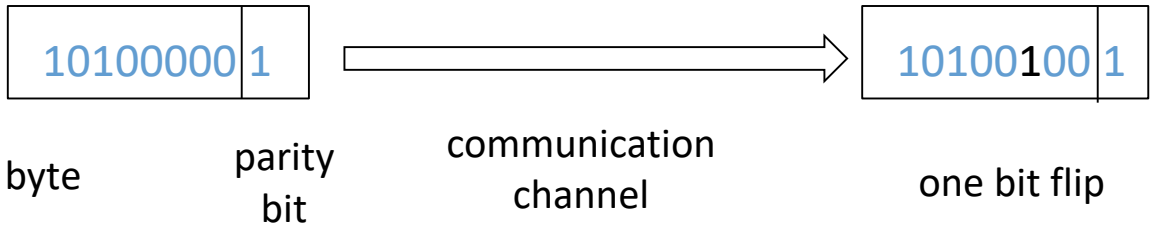
In case of hardware fault, we tolerate the error. HW

# Error detection: Types of checks

## Codes

add information to data in such a way that errors can be identified

fault: bit flip
mechanism: parity bit
error detection: data do not satisfy the parity bit




for each unit of data, e.g. 8 bits, add a parity bit so that the total number of 1's in the resulting 9 bits is odd

Two bit flips are not detected

# Error detection: Types of checks

UNIVERSITÀ DI PISA

**Self-checking component**

a component that has the ability to automatically detect the existence of the fault and the detection occurs during the normal course of its operations

Typically obtained using coding techniques:

inputs and outputs are encoded (also different codes can be used)

Applicable to small circuits:

Comparators, Voters, …

Clear error confinement

# Effectiveness of error detection (measured by)

**Coverage:**
probability that an error is detected conditional on its occurence

**Latency:**
time elapsing between the occurrence of an error and its detection
(a random variable)
how long errors remain undetected in the system

**Damage Confinement:**
error propagation path

the wider the propagation, the more likely that errors will spread outside the system

# Effectiveness of error detection (structural approach)

UNIVERSITÀ DI PISA

**Preventing error propagation**:

- "minimum priviledge"

- discriminating on type of use, users, ..

  For example, each component examines each request or data item from other components before acting on it

  - each software module checks legality and reasonableness of each request received
  - need for providing signalling back to requestor and own strategy for dealing with erroneous requests

- Make error confinement areas :

  error detection and error processing inside the module.

  If at the boundery, the module can signal if it is faulty

  -> avoid errors spread over the system

  -> create barriers at the interface of the faulty module

# Error Recovery

There is an error state. We have applied error confinement. We want to recover.

**Forward recovery**
transform the erroneous state in a new state from which the system can operate correctly

**Backward recovery**
bring the system back to a state prior to the error occurrence
- for example, recover from sw update by using the backup

# Forward Error Recovery

Requires to assess the damage caused by the detected error or by errors propagated before detection

Usually ad hoc

Example of application:

real-time control systems, an occasional missed response to a sensor input is tolerable

The system can recover by skipping its response to the missed sensor input

# Backward Error Recovery

Requires to store a previous correct state of the system

- Go backward to the saved state
- Retry
  Redo with the same component

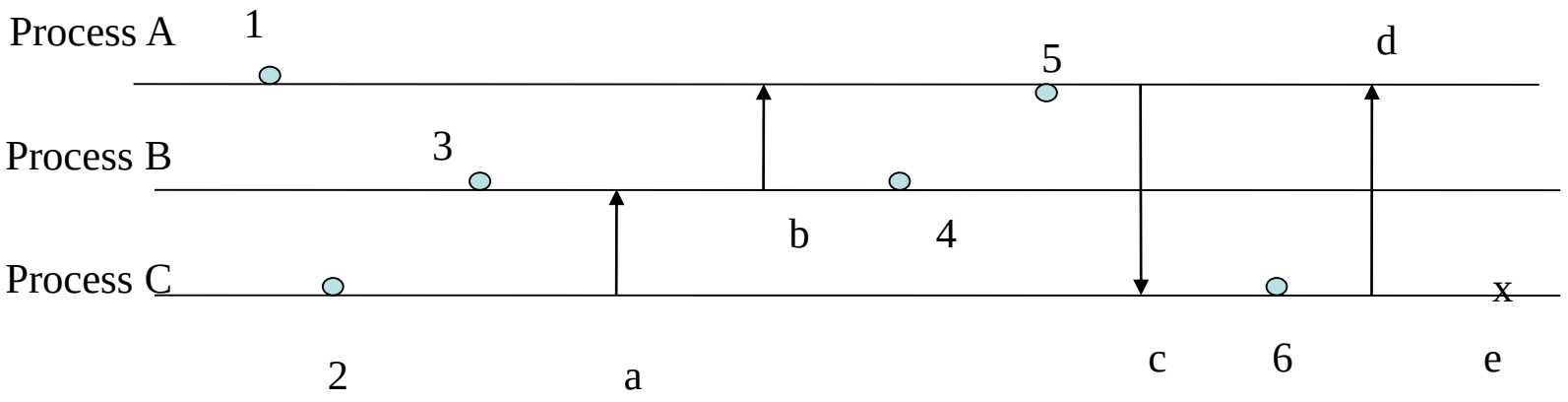
A copy of the global state is called checkpoint.

State of a computation
- Program visible variables
- Hidden variables (process descriptors, …)
- “External state”:
  files, outside words (for example alarm already given to the aircraft pilot, …

# Backward Error Recovery

Consistency of checkpoint in distributed systems

snapshot algorithms: determine past, consistent, global states




Checkpoint

x Error

Message passed

domino effect

# Backward Error Recovery

UNIVERSITÀ DI PISA

Checkpoints

- may be taken automatically (periodically) or upon request by program
- need to be correct (consistent)
- need eventually to be discarded
- survival of checkpoint data

## Basic issues:

- Loss of computation time between the checkpointing and the rollback
- Loss of data received during that interval

- Checkpointing/rollback (resetting the system and process state to the state stored at the latest checkpoint) need mechanisms in run-time support

- Overhead of saving system state (minimize the amount of state information that must be saved)

# Backward Error Recovery

Class of faults for which checkpoint is useful:

- transient faults (disapper by themselves)
- used in massive parallel computing, to avoid to restart all things from the beginning
- continue the computation from the checkpoint, saving the state from time to time

Class of faults for which checkpoint is not useful:

- hardware fault; design faults
  (the system redo the same things)

# Error mitigation

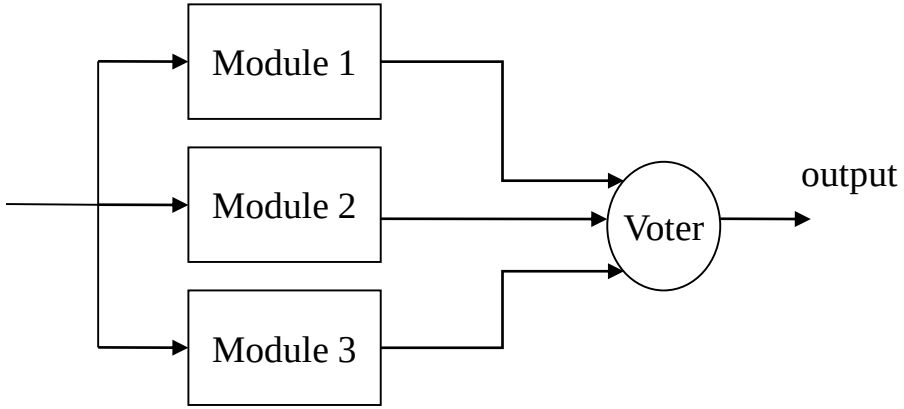
Systematic use of compensation **fault masking**

A general method to achieve fault masking is to perform multiple computations through multiple channels, either sequencially or concurrently and then apply majority vote on the outputs

Tolerance of physical faults
channels may be of identical design
(we have the assumption that **hardware components fail independently** )

Tolerance of software faults
channels must implement the same function via separate designs and implementations
(**design diversity**)

# Triple Modular Redundancy (TMR)




- 2/3 of the modules must deliver the correct results
- effects of faults neutralised without notification of their occurrence
- masking of a failure in any one of the three copies

Basic issue: loss of protective redundancy

Practical implementations of compensation: *masking and recovery*
(includes error detection and fault handling)

# Compensating actions

Compensating actions may be also necessary

Example:

A cash dispensing machine gives less money

compenating action:  tell the bank and ask for the money back

If an external communication is not correct, the computer may still limit or undo the damage by a compensating action

# Compensating actions

Example: assume a real-time program communicated with its environment and backward error recovery is invoked

Assume the environment would not be able to recover along with the program and the system would be left in an inconsistent state.

In this case, Forward recovery would help return the system to a consistent state by sending the environment a message informing it to disregard previous output from the program.

# Fault handling (fault isolation)

UNIVERSITÀ DI PISA

## Fault location

1. can the error detection mechanism identify the faulty component/task with sufficient precision?
   - LOG and TRACES are important
   - diagnostic checks
   - …

2. System level diagnosis:

   A system is a set of modules:
   - who tests whom is described by a testing graph
   - checks are never 100% certain

# Fault handling

## Fault location

What if diagnostic information / testing components are themselves damaged?

Suppose A tests B.

If B is faulty,

A has a certain probability (we hope close to 100%) of finding out.

But if A is faulty too,

it might conclude B is OK; or says that C is faulty when it isn't

# Fault handling

## Fault treatment

1. Faulty components could not be left in the system
   - faults can add up over time

2. Reconfigure faulty components out of the system
   - physical reconfiguration:
     turn off power, disable from bus access, ..

   - logical reconfiguration:
     don’t talk, don’t listen to it

# Fault handling

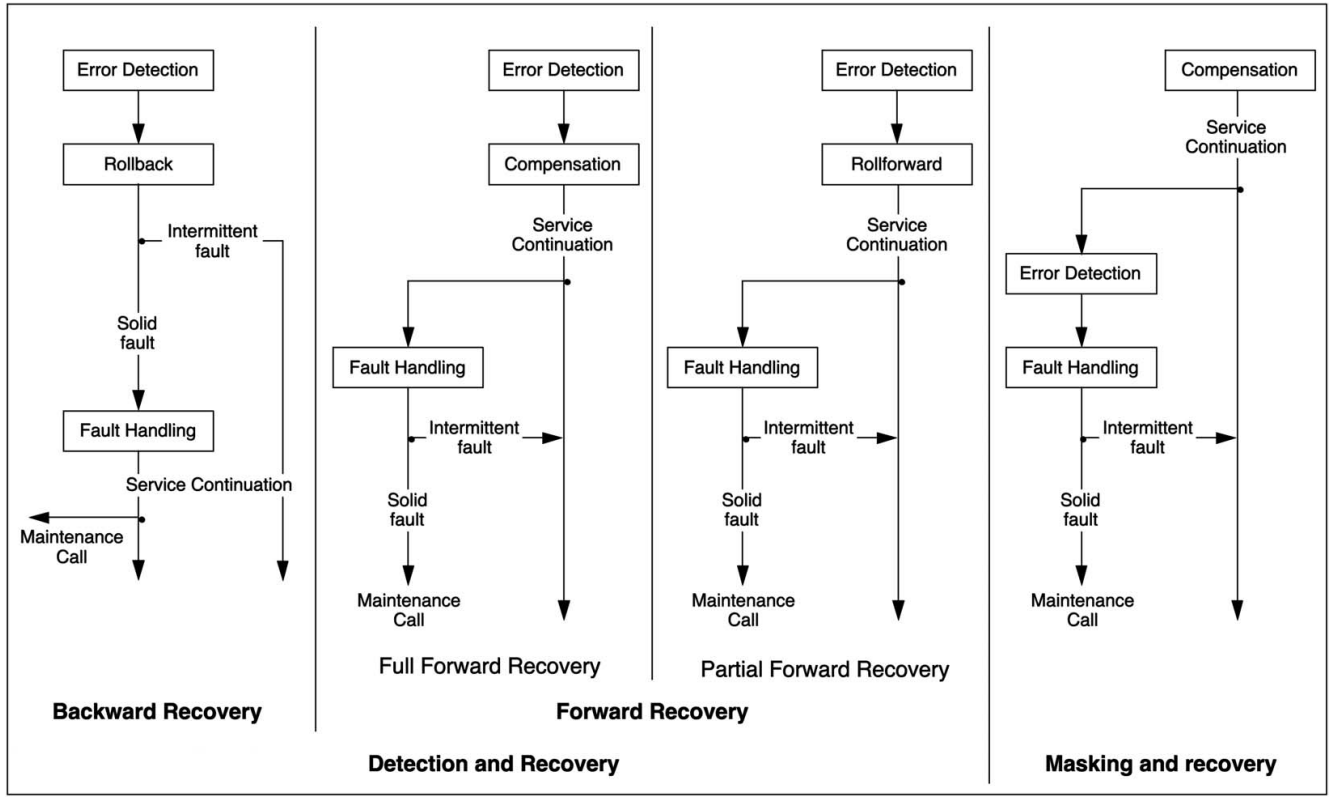
## Fault treatment

3. Excluding faulty components will in the end exhaust available redundancy
   - -insertion of spares
   - -reinsertion of excluded component after thorough testing, possibly repair

4. Newly inserted components may require:
   - - reallocation of software components
   - - bringing the recreated components up to current state

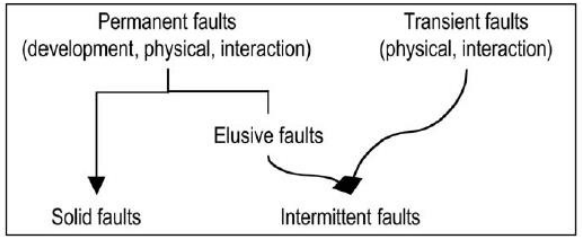
System recovery = error handling + fault handling

# Various strategies for implementing fault tolerance




From [Avizienis et al., 2004]

Solid faults: permanent faults whose activation is reproducible
Elusive faults: permanent faults whose activation is not systematically reproducible (e.g, conditions that occur in relation to the system load, pattern sensitive faults in semiconductor memories, …)
Intermittent faults: transient physical faults + elusive development faults



# Various strategies for implementing fault tolerance

The choice of the strategy depends upon the underlying fault assumption that is being considered in the development process

The classes of faults that can actually be tolerated depend
- on the fault assumption and
- on the independence of the redundancies with respect to the fault creation and activation

# Observations

Fault assumptions play a fundamental role

Fault tolerance applies to all classes of faults

Mechanisms that implements fault tolerance should be protected against the faults that might affect them

Fault tolerance uses replication for error detection and system recovery

Error detection must be a trustworthy  mechanism

# Observations

Fault tolerance relies on the independency of redundancies with respect to faults

When tolerance to physical faults is foreseen, the channels may be identical, based on the assumption that hardware components fail **independently**

When tolerance to design faults is foreseen, channels have to provide identical service through separate designs and implementation (through **design diversity**)

Fault masking will conceal a possibly progressive and eventually fatal loss of protective redundancy.

Practical implementations of masking generally involve error detection (and possibly fault handling), leading to masking and error detection and recovery