# Systems and Information Security
# SEGSI

## Topic 7

## Architectures

Pinto Leite, Jorge (jpl@isep.ipp.pt)

Departamento de Engenharia Informática
Instituto Superior de Engenharia do Porto

# Architectures

- When designing a secure architecture some principles should be considered and, if possible, implemented
- A security failure might have has a starting point
  - A failure of a asset
  - A failure of an application
  - An exploit of a vulnerability
- The consequences might be an erroneous state or the disclosure of unintended information

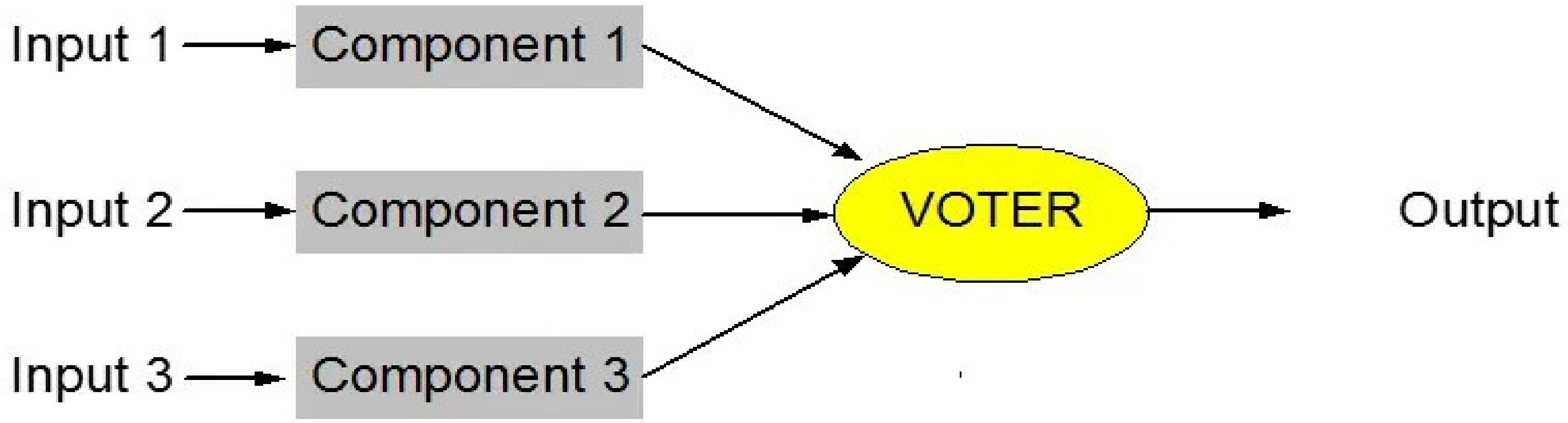
# Architectures

- In this topic we will focus on asset and application failures
- When designing an infrastructure we already saw (Topic 5) that failures might occur on any asset, and the strategies that might be implemented
  - Preventive fault detection
  - Retrospective fault detection
- Whatever strategy is implemented, the considerations for hardware and software differ

# Architectures

- In hardware terms, a fault-tolerant hardware can be based on *triple modular redundancy* (TMR)
  - There are three identical and replicated components that receive the same input and whose outputs are compared
  - If an output is different it is ignored and the respective component is assumed to be defective

# Architectures




Source: Engineers Community

Note: alternative implementations might be used

# Architectures

- The output comparator is generally a relatively simple hardware device
- Compare your entries and reject one that is different from the rest
- The selection of the “correct” exit is made through a vote in which the one with the most votes wins
- TMR's success in providing fault tolerance is based on two fundamental assumptions
  - Hardware components have no common design flaws
  - These components fail randomly and there is a low probability of simultaneous failure

# Architectures

- TMR-related problems can be perceived under different perspectives
  - Problem 1
    - Is it clear that if one component outputs a different (and, as so, minority) result it is undoubtedly the incorrect one?
  - Problem 2
    - Only one voter?
    - It might be a SPOF
  - Problem 3
    - If one component is turned off and the remaining two outputs different results, what would be the option of the voter?

# Architectures

- The solution to the above problems involves different approaches to architectural design
- Obviously, duplicating the architecture more and more is not a solution
- However, the assumptions stated on slide 6 should / must be taken into consideration
- In addition, and if possible
  - Ensure controls that allow better granularity of the definition of “correct”
  - Consider implementing a double line of components or at least voters
  - Analyze and test procedures in the event of a tie

# Architectures

- On slide 6 the following assumptions were presented
  - *Hardware components have no common design flaws*
  - *These components fail randomly and there is a low probability of simultaneous failure*
- However, it's not usually easy to hardware components to have no common design flaws
- If they all have the same architecture, a common error might state up, simultaneously
  - As well as on an application
- That is the thinking that drives ***redundancy*** and ***diversity***

# Architectures

- *Redundancy* is achieved if an environment is duplicated (at least), resulting in distint and absolutely identical environments
- *Diversity* is achieved if different versions of an environment is deployed
- An easy way to differ them is to imagine systems implementation
  - Redundancy
    - Two or more systems with same operating system, programming languages, applications, etc.
  - Diversity
    - Two or more systems with different operating systems and / or programming languages and / or applications, etc.
- This idea can be improved by having different manufacturers or equivalent

# Architectures

- The assumptions of TMR are not viable in software!
  - It is not recommended to simply replicate the same component, as this would cause them to have common design flaws
  - Therefore, the simultaneous failure of software components is virtually inevitable
  - Software systems must be diversified
- Thus, TMR analogies can be used on software

# Architectures

- Software design should / must use diversity
  - Different versions of the system are designed and implemented differently
    - The goal is to have different failure modes!
  - As well as different approaches to design
    - Different paradigms - object oriented, functional, imperative, etc.
    - Different implementation languages
    - Different tools and development environments
    - Different algorithms in the implementation



Departamento de ENGENHARIA INFORMÁTICA
Instituto Superior de Engenharia do Porto

# Architectures

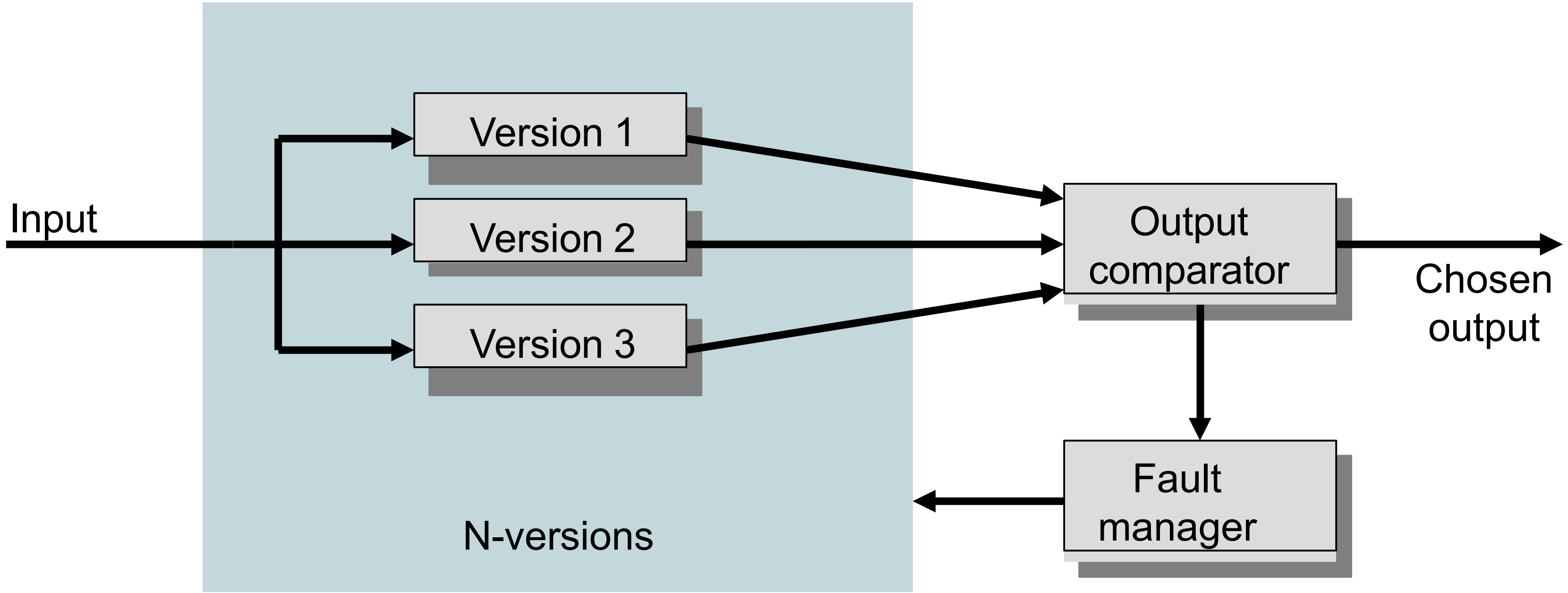
- Analogies of software TMR
  - Programming with N-versions
    - The same specification is implemented in different ways by different teams
    - All versions count simultaneously and the majority vote determines the chosen output
    - Most used approach in many models of commercial Airbus or Boeing aircraft and others
  - Recovery blocks
    - Several different versions of the same specification are designed and executed sequentially
    - An acceptance test is used to select the output that will be transmitted to the rest of the system

# Architectures

- N-Versions
  - All versions receive the same input at the same time
  - Output comparator might have a maximum wait time for all versions to provide an output, specially for real-time systems
  - The most voted output is chose to  continue for the next step / process
  - The most demanding paper here is the output comparator, in spite of being  a simple software component that uses a voting mechanism to choose the final output
  - As with hardware-based systems, the output comparator is a simple software component that uses a voting mechanism to choose the final output

# Architectures

- N-Versions

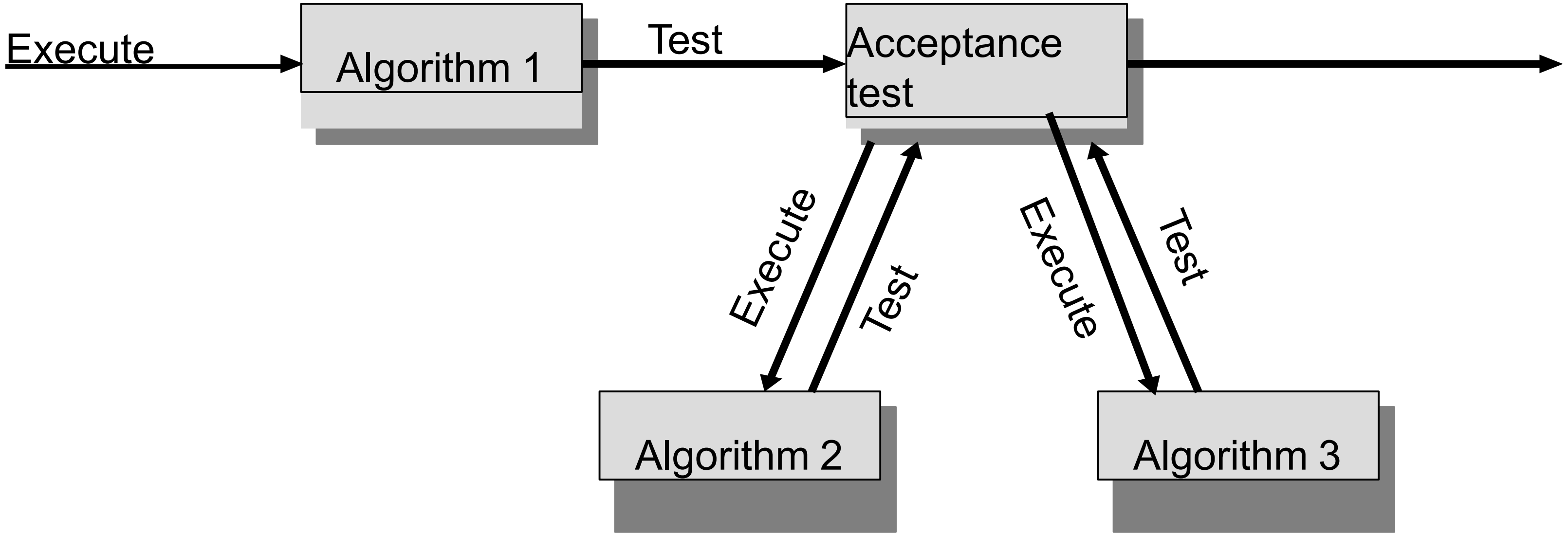


# Architectures

- Recovery blocks
  - There are different architectures to assure same functionality
  - Each recovery block contains
    - 1. An algorithm
    - 2. An acceptance test (might be common for all algorithms)
  - Algorithm 2 will only be called if the test result from algorithm 1 is error (or to check algorithm 1 result), and same applies to call algorithm 3
  - It is mandatory that a different algorithm be executed on the various blocks in order to reduce the probabilities of common errors
  - The design of the acceptance test is complicated as it must be independent of the computation performed
  - There are complex problems with this approach in the case of real-time systems due to the sequential execution of the different redundant blocks

# Architectures

- Recovery blocks



# Architectures

- Whatever analogy is used, the perspectives of slide 7 still applies
  - Problem 1
    - Is it clear that if one component (version or algorithm in this case) outputs a different (and, as so, minority) result it is undoubtedly the incorrect one?
  - Problem 2
    - Only one output comparator / acceptance test?
      - In spite of every algorithm can contain its own acceptance test on Recovery Blocks option
    - It might be a SPOF
  - Problem 3
    - If one version or algorithm is turned off and the remaining two outputs different results, what would be the option of the output comparator / acceptance test?