Confiabilidade de Sistemas Distribuídos
Dependable Distributed Systems

DI-FCT-UNL, Nuno Preguiça

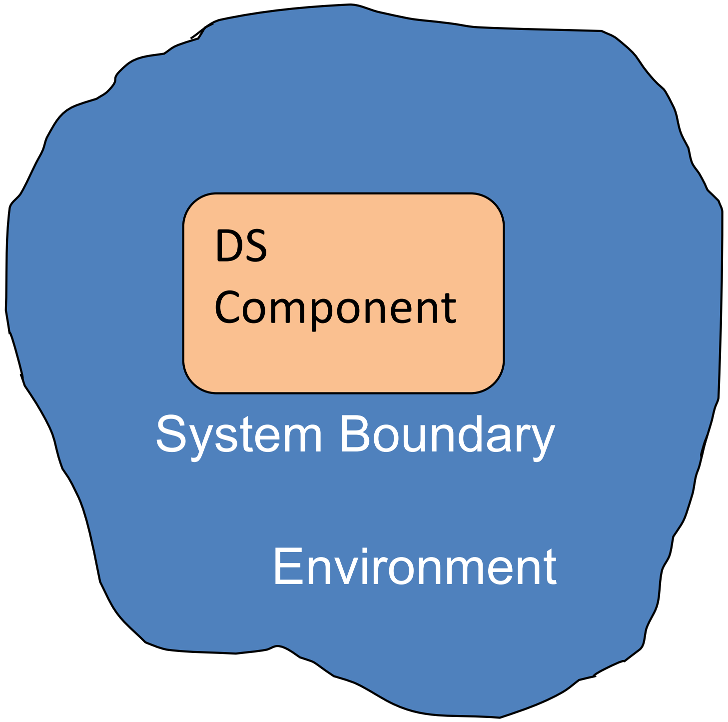
Lecture: Introduction
Dependable Distributed Systems
Principles, Concepts, Properties and Technology

2018/2019, 2nd SEM

MIEI
Mestrado Integrado em Engenharia Informática

Distributed Systems

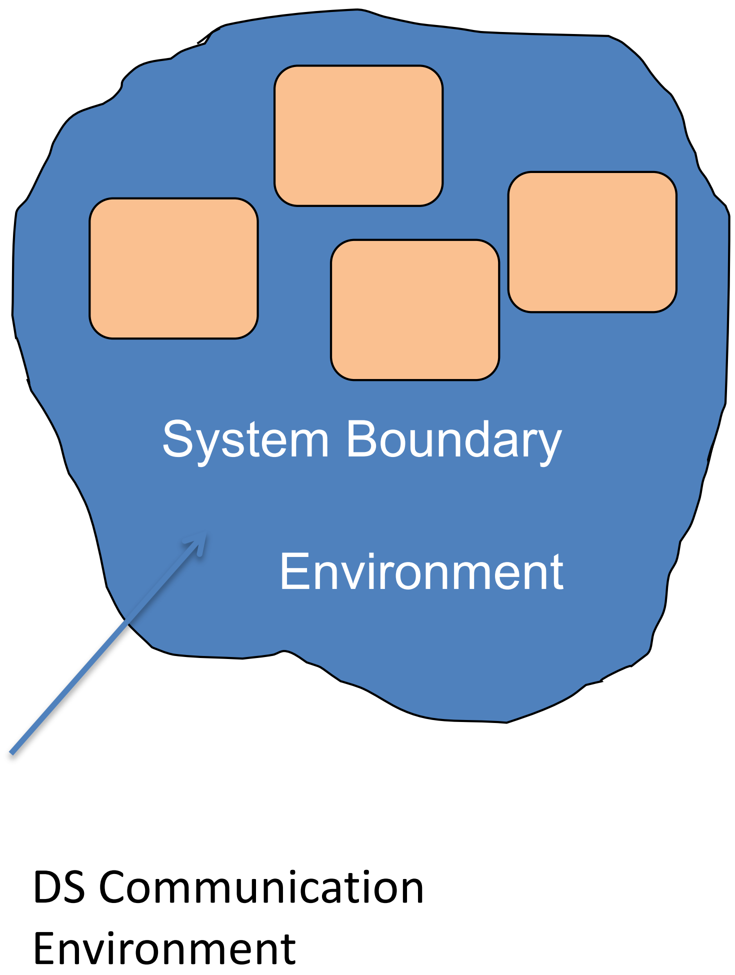
Distributed Systems and Components



- A **system** is an entity that interacts with other entities, i.e., other systems, including hardware, software, humans, and the physical world with its natural phenomena
- These other systems are the **environment** of the given system
- The **system boundary (perimeter)** is the common frontier between the system and its environment
- A system may consists of one or more **components**, such as nodes or processes

Distributed Components:

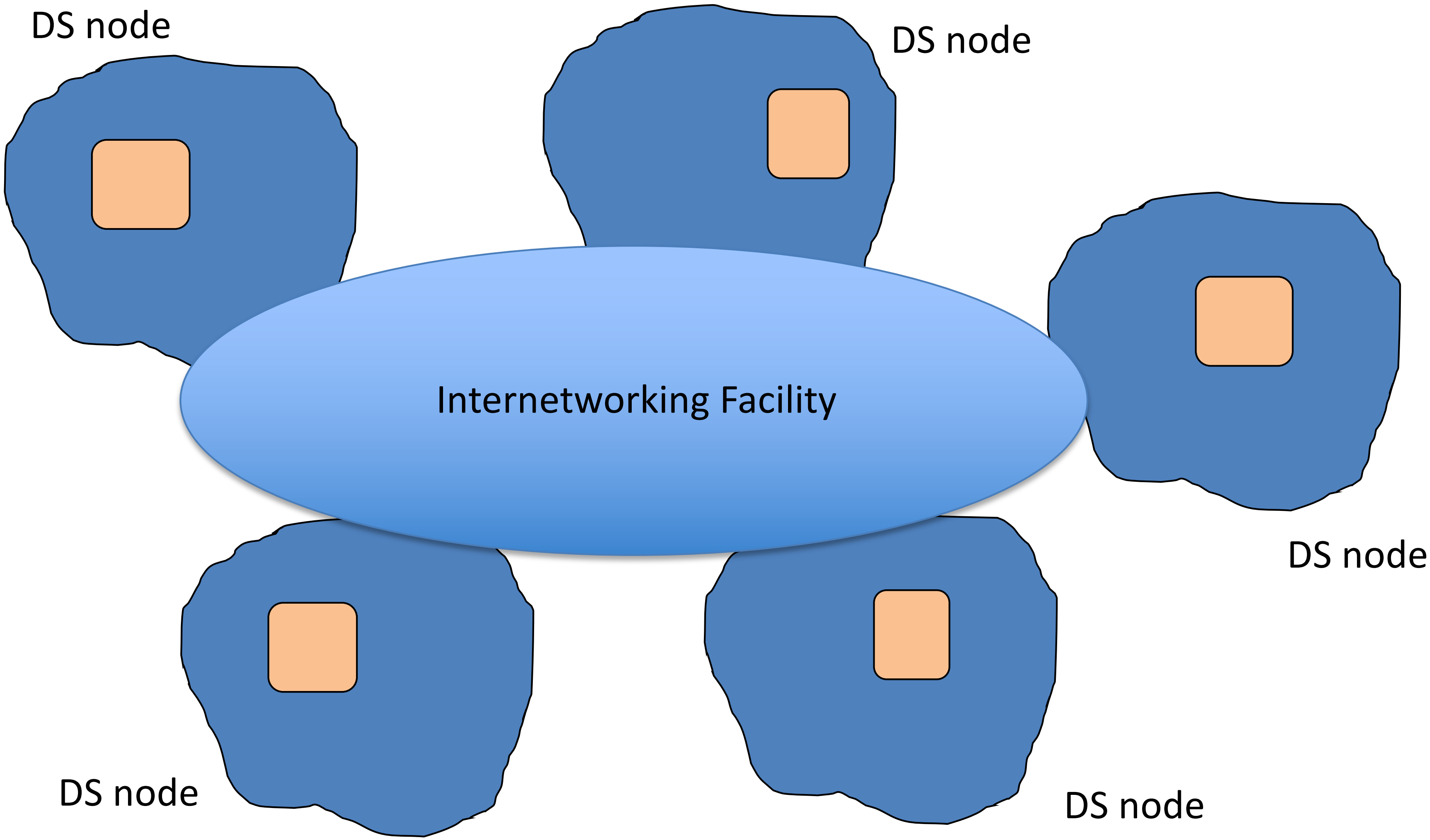
System of distributed components Environment, System Boundary (Perimeter)



- A **system** is an entity that interacts with other entities, i.e., other systems, including hardware, software, humans, and the physical world with its natural phenomena
- These other systems are the **environment** of the given system
- The **system boundary (perimeter)** is the common frontier between the system and its environment
- A system may consists of one or more **components**, such as nodes or processes

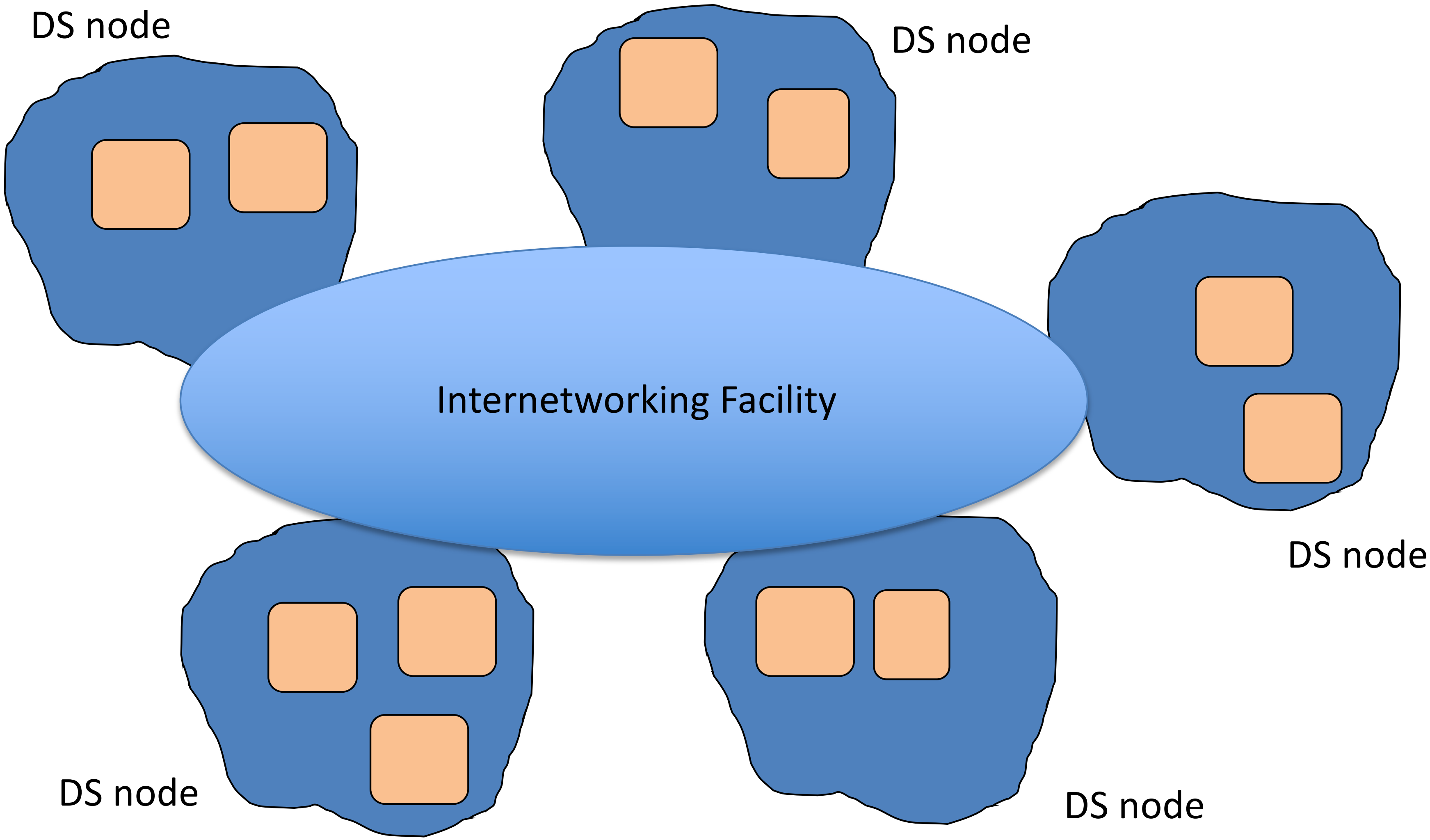
Distributed System and Granularity

Vision as a System of Systems:
Scale and Complexity

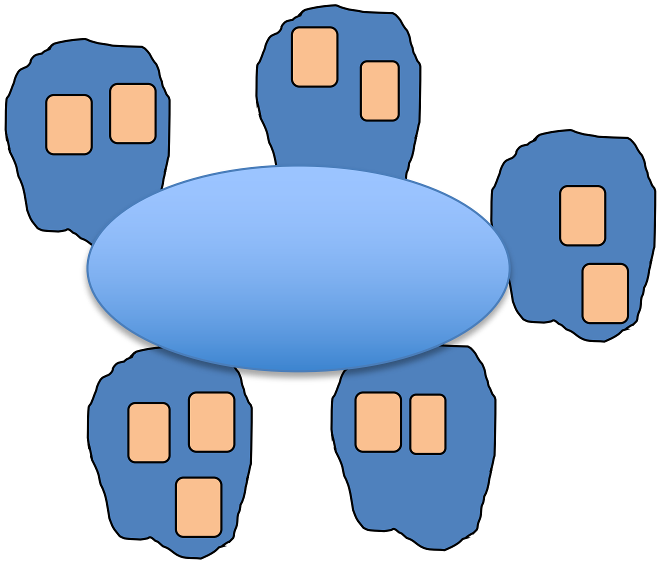


Distributed System and Granularity

Vision as a System of Systems: Scale and Complexity



Distributed systems and the vision of systems of systems



- DS: a **set of distributed nodes, executing autonomous computations (processes) that need to interact to coordinate their actions**

- **Baseline for discussion:**
  - **Synchronous or asynchronous by nature ?**
  - **Can we expect independent failures ?**
  - **How to identify (define and circumvent) the trust computing base and its foundations ?**
  - **The same for “dependability base”**

Failure Model Definition

- A definition expressing the typology of faults in the system model design assumptions

- Need very-well defined typology of faults considered in the system design model
  - Usually, looked as accidental failures

Threat or Adversary Model Definition

- A definition expressing the typology of threats or attacks (attack types as the concretization of defined threats) in the system model design assumptions

- Need very-well defined typology of attacks (vectors, adversary conditions and hypothesis) as the concretization of the threats potential considered in the system design model
  - Initial discussion: is it very different from the failure model?
  - See this as the injection of malicious failures by attackers

Dependability

Dependable Distributed Systems
What is “Dependability” ?

- Context (**Dependable Distributed System**):
  - We have components providing services to clients (or other components).
  - To provide services, components may require services from other components

    → We say that the component **may depend** on some other component

We say that a **component C depends on C*** if the **correctness of C's behaviour depends on the correctness of C*'s behaviour**.

Then C* is a Dependable System

C* is used by C as a DEPENDABLE COMPUTING BASE

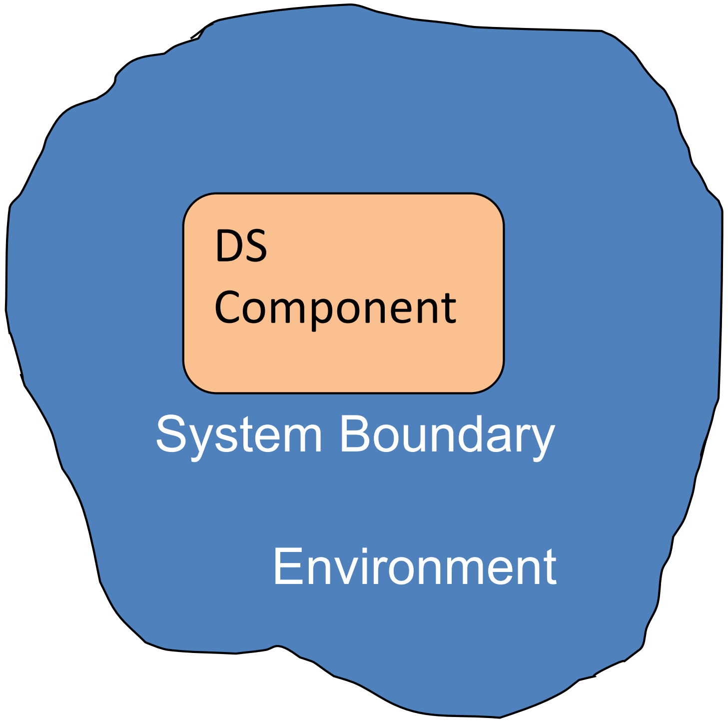
DEPENDABLE vs. TRUSTWORTHY

What are “these” components about ? : Different levels in a Distributed System Stack

A definition for Dependability

- Ability of a Distributed System to provide correct services (computations) to its users (or to other services depending from it), **despite** various **threats**, including undetected software defects or failures, or hardware failures, whether they are accidental or caused by malicious attacks
  - Note: **this includes intrusions causing malicious failures, induction of incorrect behaviour (SW or HW)**

DS Stack: where is the TCB ?



Computation, Processing
(Support Levels)

Application
Static Libraries

Runtime Libraries
Middleware Services

OS Services and Resources

Data Repository Services
(Local NVS)

Virtualized Environment
(Virtualization Layer)

Communication Channels

Dependability Property

- Conjugation of properties, mainly aggregated in two dimensions:
  - Fault Tolerance Properties
  - Security Properties
    - Communication attacks + intrusions

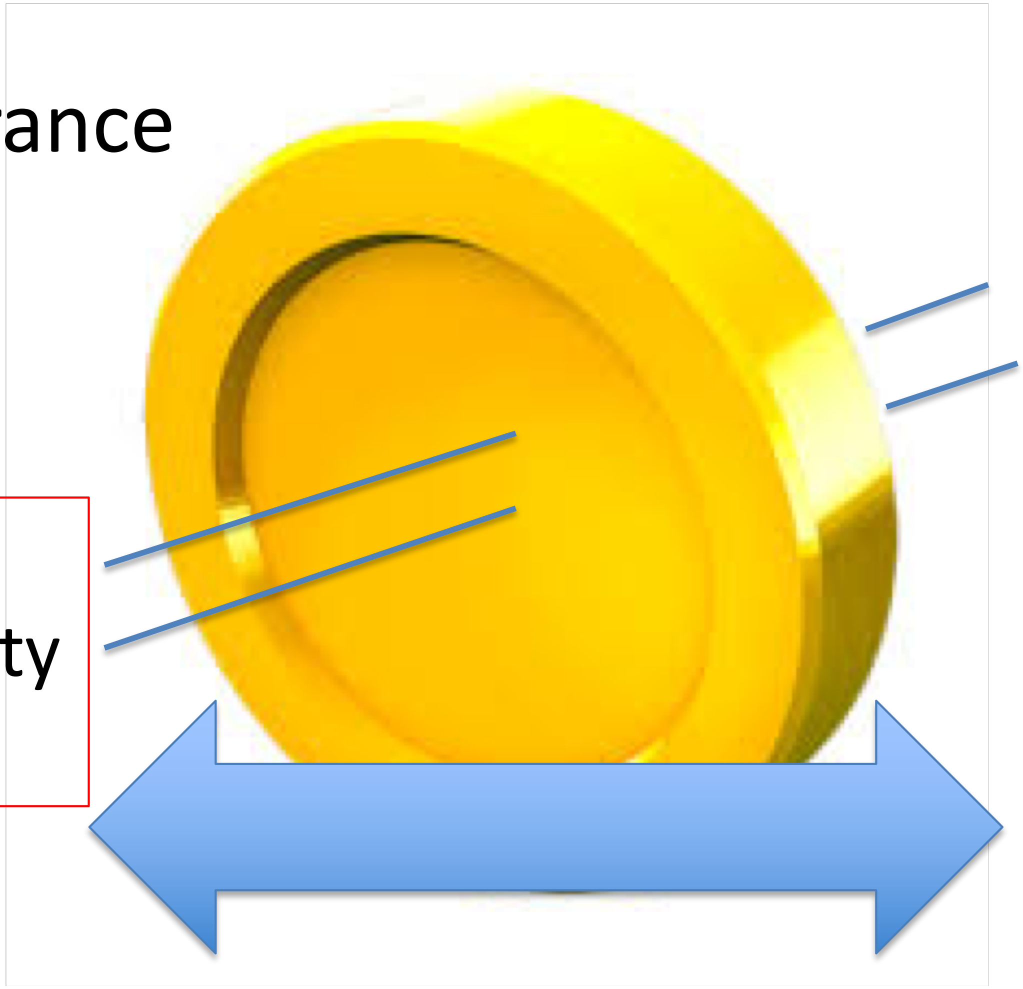
- Under a Well-Defined System Model and Software Architecture
  - A Well-Defined Fault Model
  - A Well-Defined Adversary Model

Dependability Dimensions

- As two “Faces” of the same coin
- Challenge/Trend: addressable “all-in-one” solution ?

Fault Tolerance Properties w/

Availability, and Reliability Properties



Security Properties w/

Intrusion Tolerance (Prevention + Detection + Recovery)

Fault-Tolerance Dimension

Dependability Properties: Fault-Tolerance Dimension

- **Base concepts**
  - **Availability**
    - Readiness for usage
  - **Reliability**
    - Continuity of service delivery
  - **Safety**
    - Very low probability of catastrophes/disasters
    - Accidental Failures (in typical approaches)
  - **Maintainability**
    - How easily can a failed system be repaired or recovered

Reliability vs. Availability (2)

- **Availability: A(*t*):**
  - **Average fraction of time** that a component has been up and running in the interval ***[0 , t]***
- Long-Term Avaiability (or Always Available):
  - ($A(\infty)$

Relating:

- **A = MTTF/MTBF**

  => A = MTTF / (MTTF + MTTR)

Reliability vs. Availability (1)

- **Reliability *R*(*t*):**
  - **probability** that a component has been **up and running** (correctly and continuously) in the time interval ***[0, t]***

Conventional Metrics:

- **MTTF:** Mean Time To Failure:
  - Average time until a component fails
- **MTTR:** Average time it takes to repair (recover) a failed component.
- **MTBF:** Mean Time Between Failures
  - MTTF + MTTR

Reliability vs. Availability (3)

- Important Observation:
  - Reliability and availability make sense:
  - **If we have an accurate notion of what a failure actually is**
  - **This Requires a "very well-defined" Failure Model, related to the System Model and Design Assumptions**

  - => Must address Reliability vs. Availability vs. Efficiency or prformance Tradeoffs BY DESIGN !

Safety

- **Safety.** Level of tolerance against catastrophic failures or accidents
  – Faults mainly as “accidental” failures (in the more typical approach of FAULT TOLERANT DISTRIBUTED SYSTEMS)

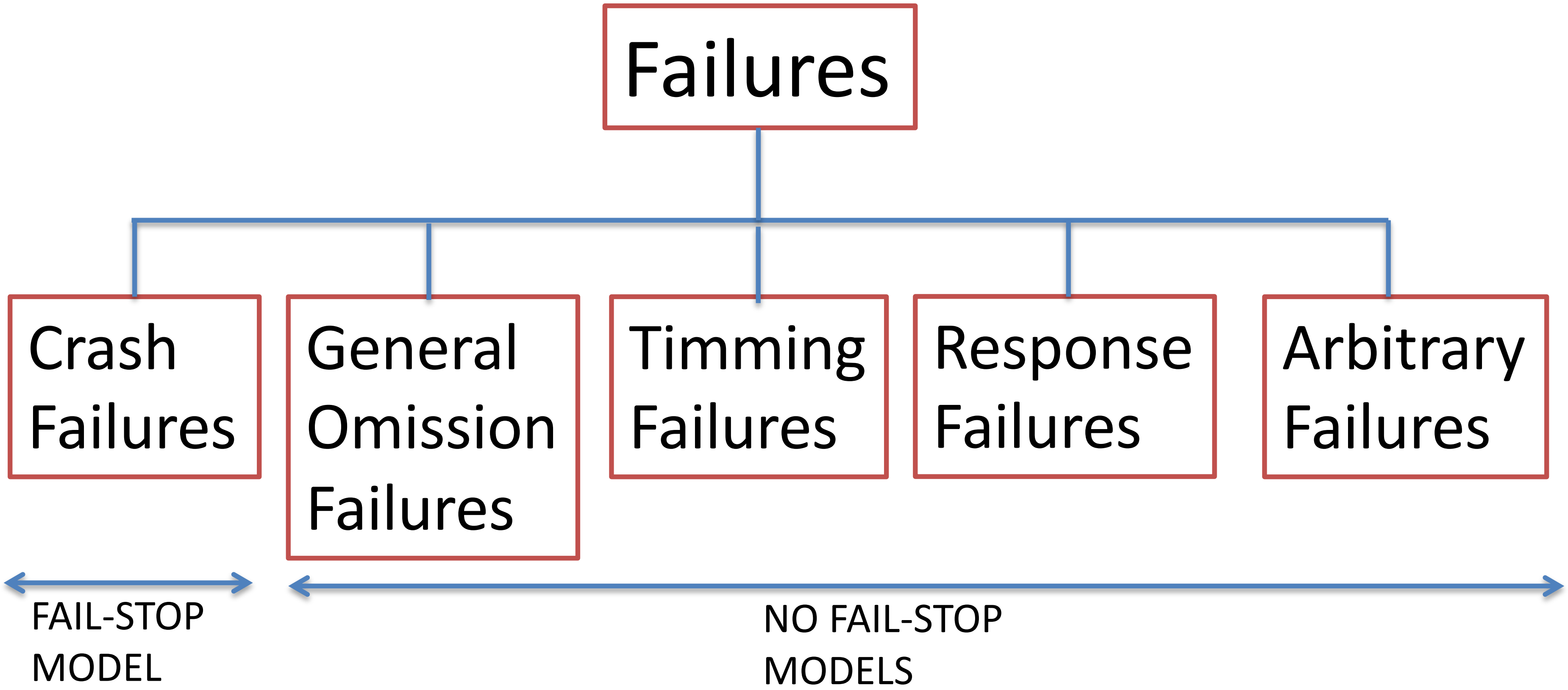
  – RISK = FAULTS x Operation Exposition

Need to establish concepts and metrics for FAULTS and Operation Exposition / Related to previous metrics

Maintainability

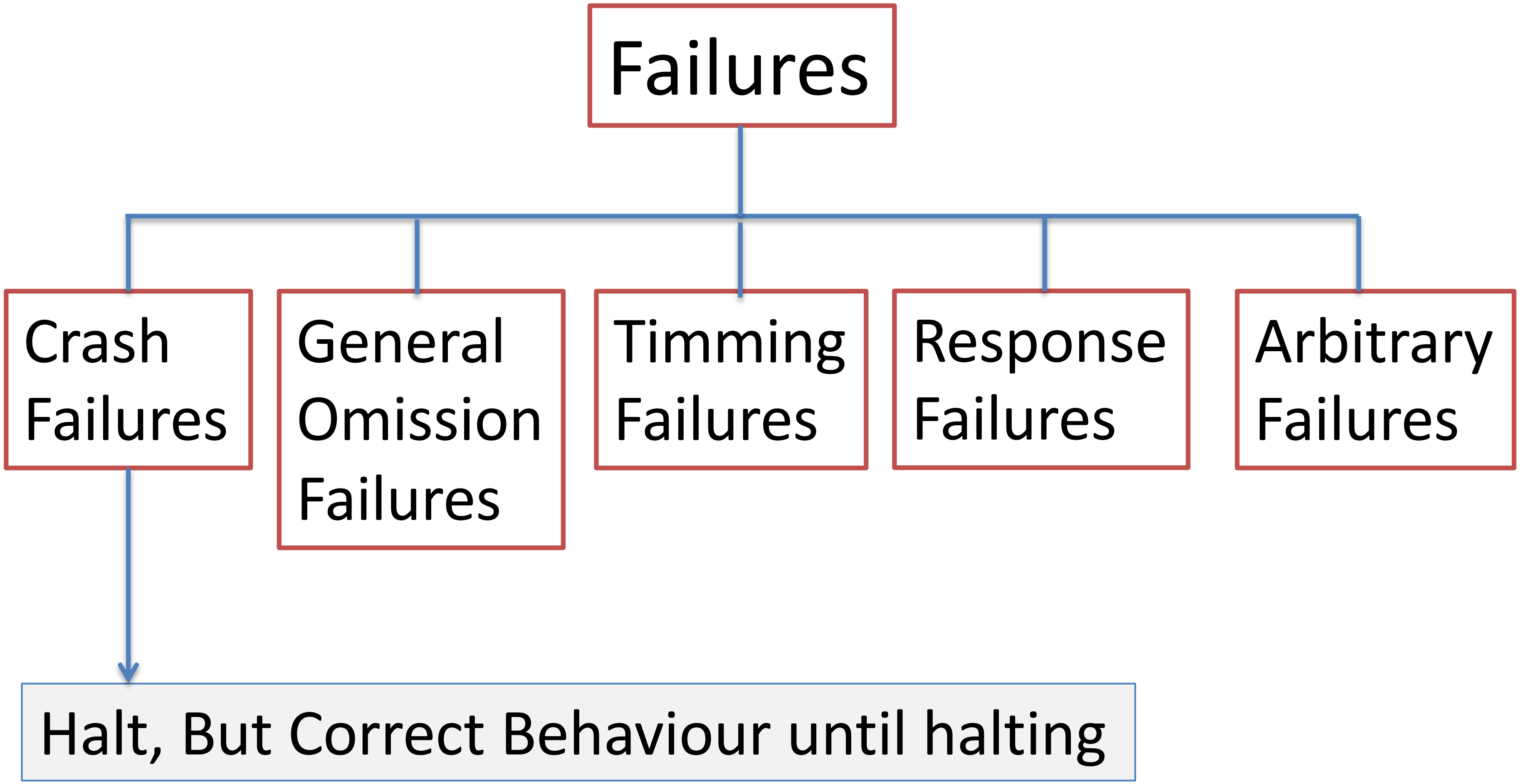
- Maintainability
  - How easy to recover a failed system (manual intervention or automatic mechanisms)
  - Some times only the manual intervention is considered
  - But fault-tolerance (for availability or operation continuity) requires automatic recovery mechanisms
    - Ex., Replication (w/ consistency)
    - Ex., Diversity
    - Ex., Virtualization
    - …. Possible “Onion” Supported Approaches

Failure Models and Types of Failures
Generic Characterization

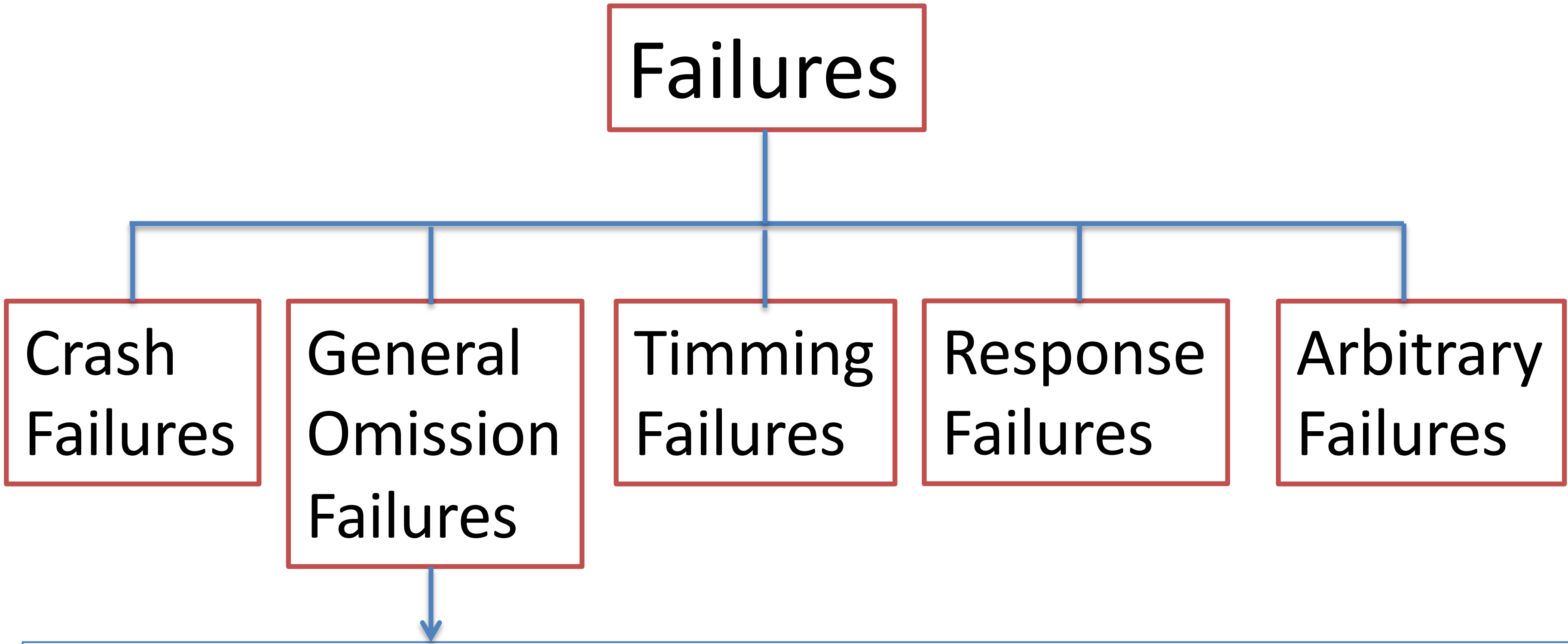


Typology (as ref. in Andrew Tanenbaum, Maarten Van Steen,
Distributed Systems - Principles and Paradigms,
Chap. 7 – Fault Tolerance (2nd Edition,

Failure Models
> Crash Failures



Failure Models
> Omission Failures

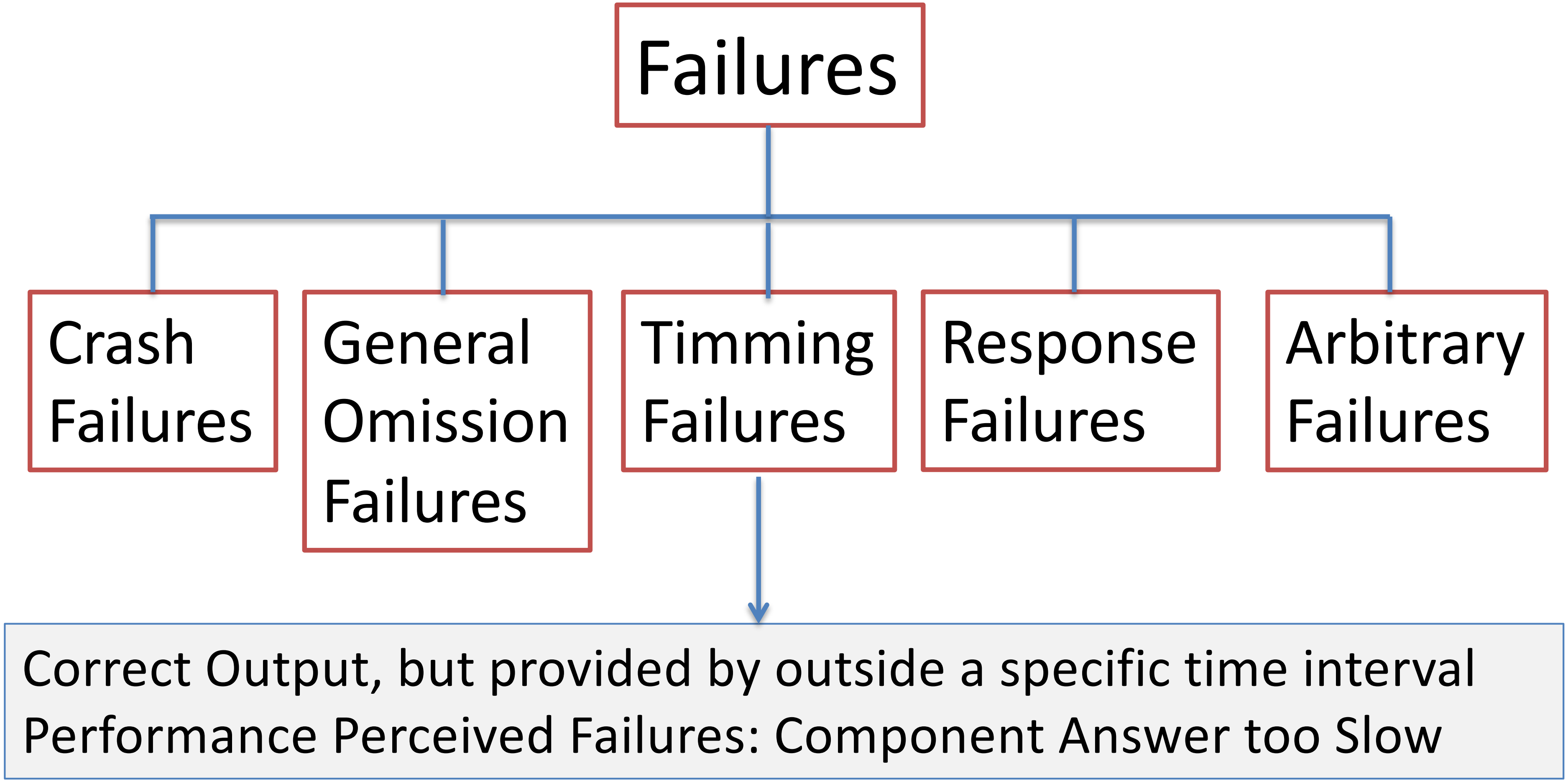


Failure in Sending or Receiving Messages

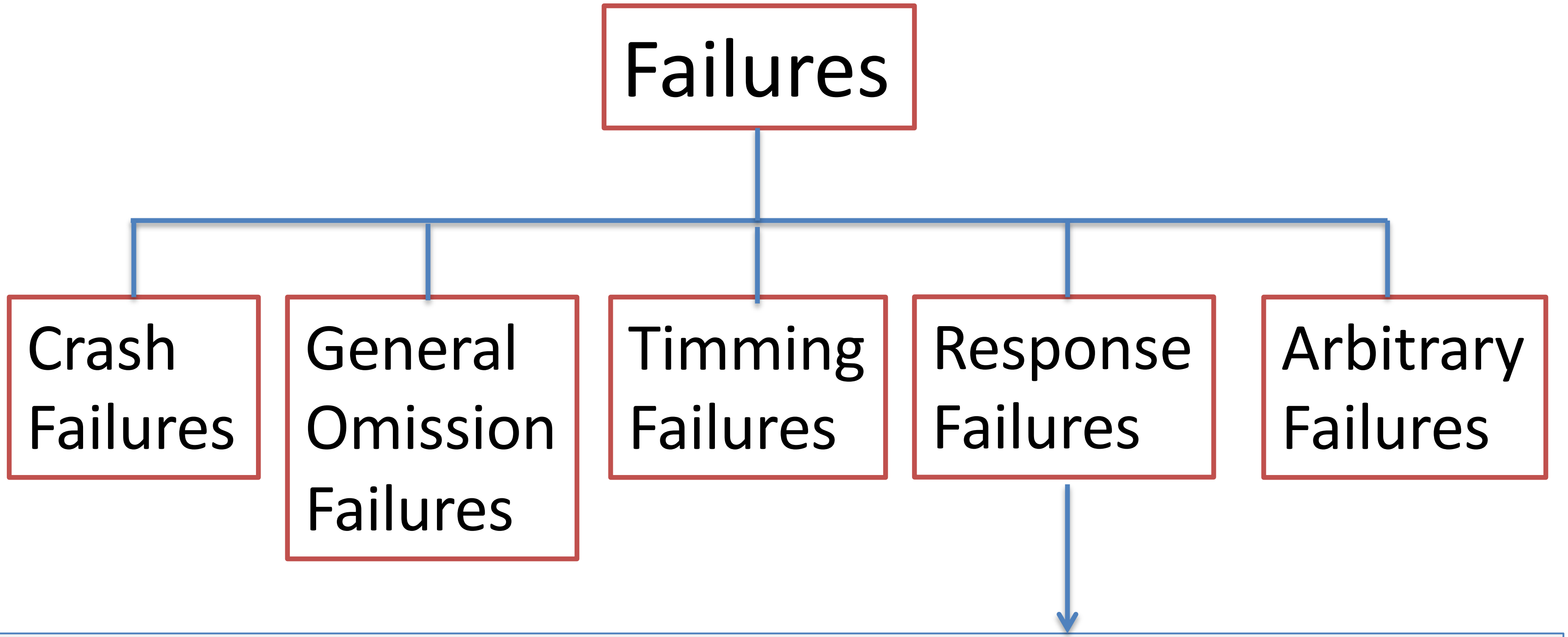
Recv Omissions: Correctly Sent Messages are not Received

Send Omissions: Messages not sent correctly (that should have)

Failure Models
> Timing Failures

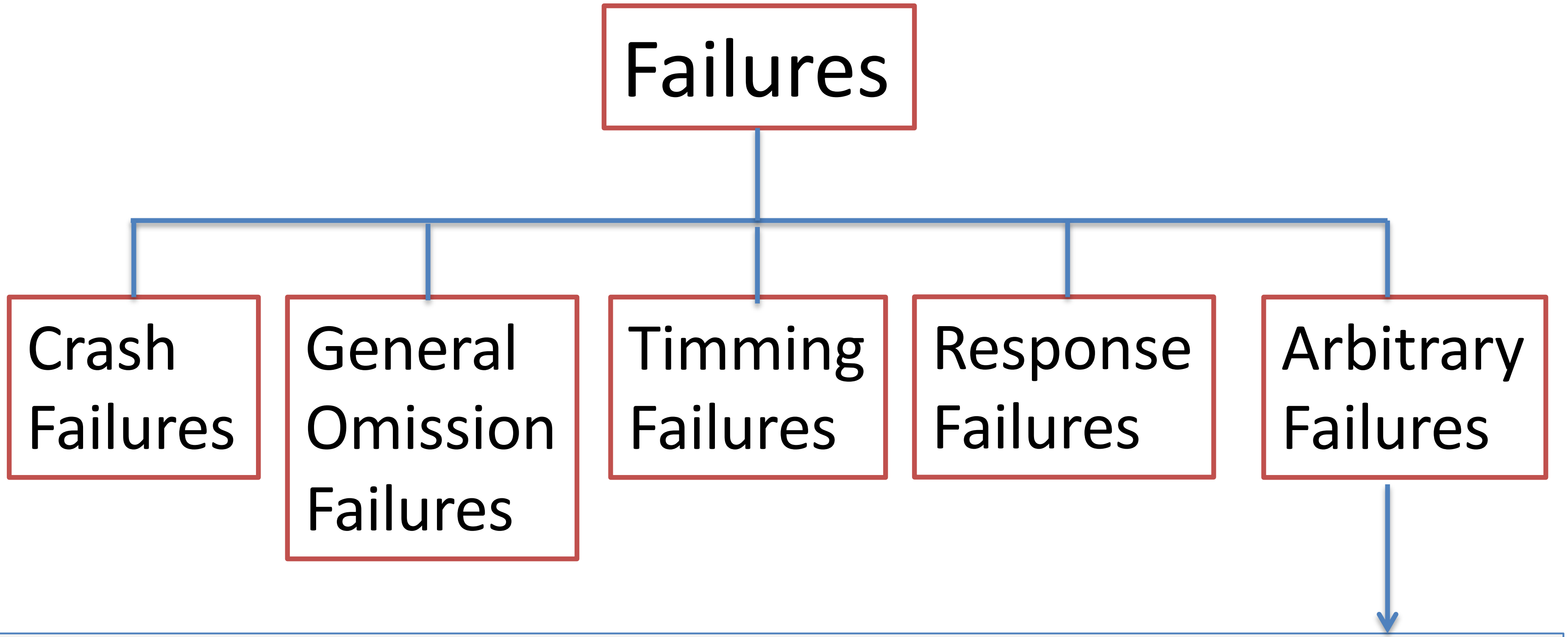


Failure Models
> Response Failures



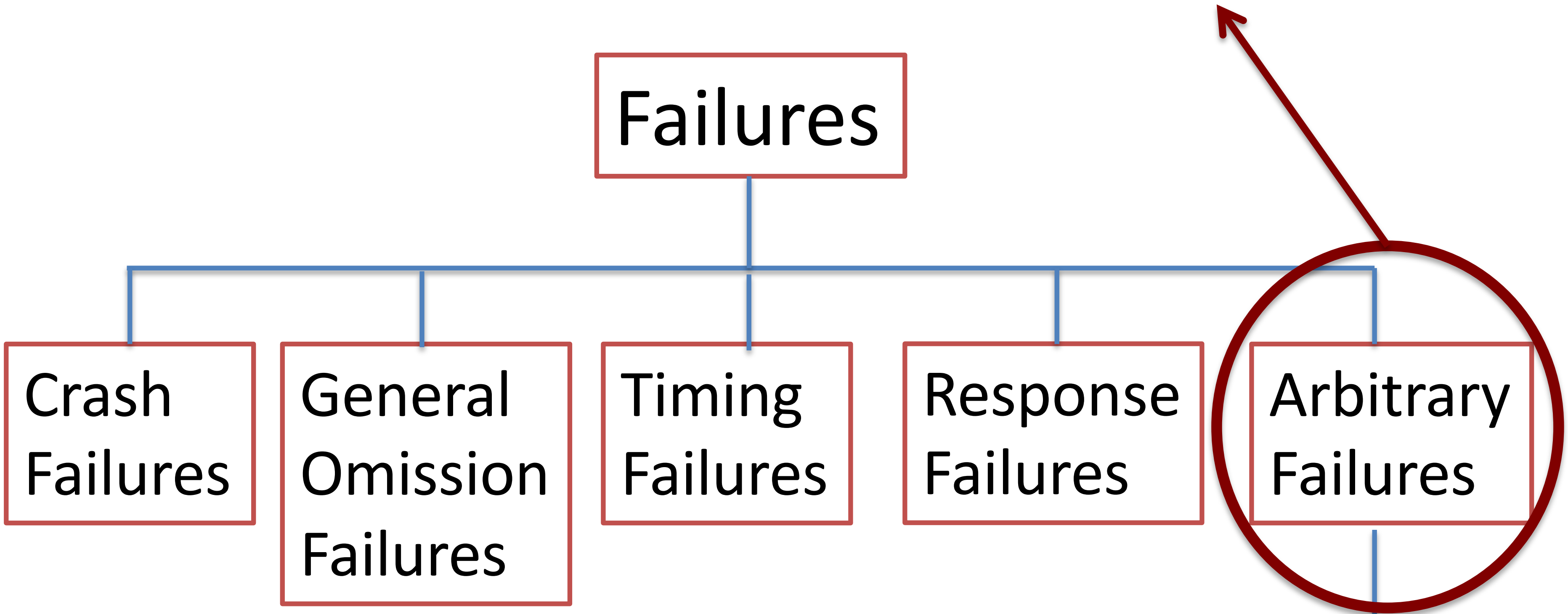
Incorrect output, but cannot be accounted to another component
> **Value Failures**: wrong output values
> **State-Transition Failures**: deviation from correct flow of control
(Note: this failure may initially not even be observable)

Failure Models
> Arbitrary Failures



Any (or any combination of) failure may occur, perhaps even unnoticed (silent failures) or not (noticed or detectable failures)

Failure Models
> Arbitrary Failures => Byzantine Faults



Any (or any combination of) failure may occur, perhaps even unnoticed (silent failures) or not (noticed or detectable failures)

BYZANTINE FAULT MODEL ASSUMPTIONS: Interesting for Dependability Assumptions. Why ?

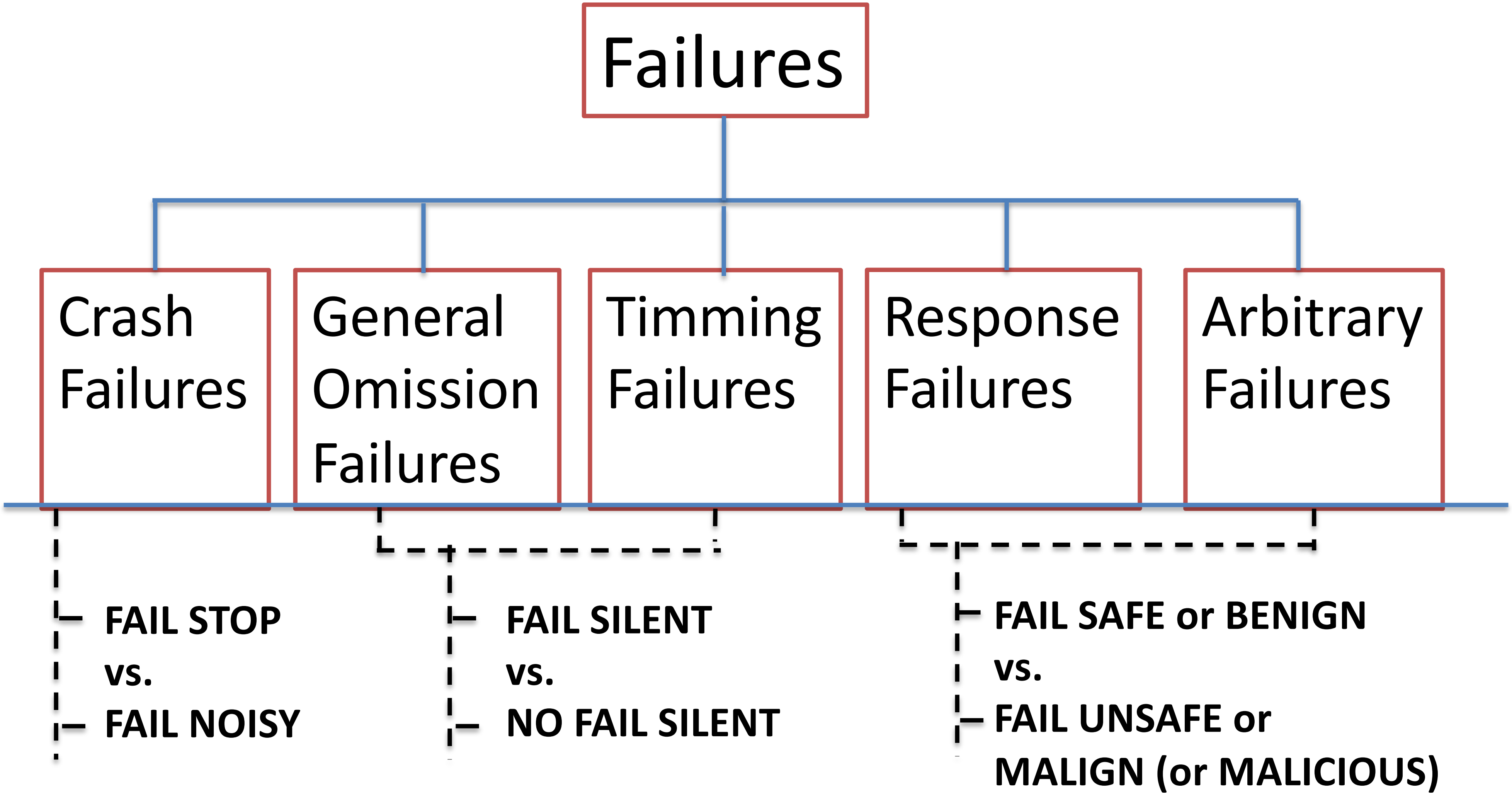
Can we detect halting failures?

- **Synchronous system model**: process execution speeds and message delivery times are bounded
  - → More easy and sometimes we can reliably detect omission and timing failures.
- **Asynchronous system**: no time-bound no assumptions about process execution speeds or message delivery times
  - → Cannot reliably detect crash failures, neither general omission failures or timing failures

Fault Tolerance: Halting Failures

- Sometimes we can consider **quasi or partially synchronous systems**:
  - Most of the time (quasi) or in some moments or parts (partially), we can assume the system to be synchronous
    - As example is also when we consider “synchronous behaviour in some component or in a component in our Trust Computing Base)
  - Yet there is no bound on the time that a system is asynchronous (rest of the time for the rest of the system)

Failure Models and Refinements



Failures: other classification criteria

- Nature of output : malicious vs. non-malicious failures
- Permanent vs. transient (or intermittent) failures
- Independent vs Correlated (or Colluding Failures)

Nature of output

Assumptions we can make (some authors):

- **Malicious fault:** The fault that causes a unit to behave arbitrarily or malicious. Also referred or associated by some authors to a **Byzantine fault**
  - A sensor sending conflicting outputs to different processors
  - Compromised software system that attempts to cause service failure
- **Non-malicious faults:** the opposite of malicious faults
  - Faults that are not caused with malicious intention
  - Faults that exhibit themselves consistently to all observers, e.g., fail-stop
    - A fail-stop system simply stops executing once it fails

- Of course: Malicious faults are much harder to detect than non-malicious faults

Permanent vs. Transient Faults

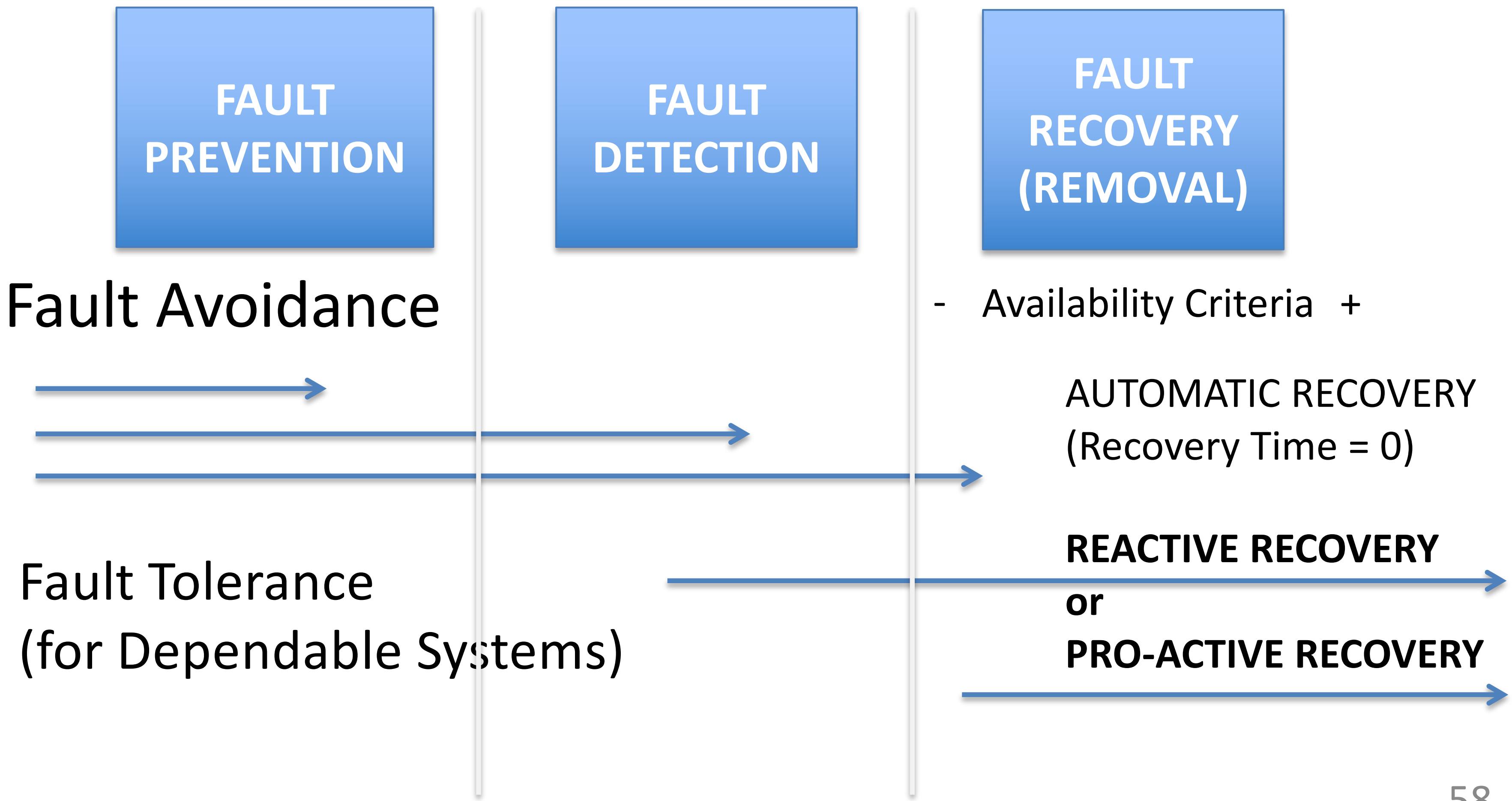
- **Permanent faults** are caused by irreversible device/software failures within a component due to damage, fatigue, or improper manufacturing, or bad design and implementation
  - Permanent software faults are also called **Bohrbugs**
  - Easier to detect
- **Transient/intermittent faults** are triggered by environmental disturbances (external excitation) or incorrect design
  - Transient software faults are also referred to as **Heisenbugs (ex., temporary memory corruption)**
  - Many studies show that Heisenbugs are the majority software faults
  - Harder to detect

Types by correlation criteria: independent vs. correlated failures

- Components fault may be independent of one another or correlated
  - A fault is said to be **independent** if it does not directly or indirectly cause another fault
  - Faults are said to be **correlated** if they are related. Faults could be correlated due to physical or electrical coupling of components
- Correlated faults are more difficult to detect than independent faults

Means for Fault-Tolerance

Means to achieve dependability (fault-model dimension)



Failure Masking and Redundancy

- To be Fault-Tolerant we must **hide the possible occurrence of failures** in distributed processes (**notion of failure masking**)
- **Key-Technique: Redundancy**
  - **Data** (or information) redundancy
    - Integrity codes (EC/FEC, Hamming codes, Integrity Checks, Data Replication and Erasure Coding…)
  - **Time** Redundancy (transient/intermittent failures)
    - Retry strategies (ex., distributed transactions)
  - **SW** redundancy
    - State and Processing (Replication services, Replicas, Process Groups)
  - **Physical redundancy w/ HW**

Failure Masking and Resilience: k-Fault Tolerant Groups

- **k-Fault-tolerant group:**
  - When a **group can mask any *k* concurrent member failures**
  - ***k*** is called **degree of fault tolerance**
    - Resilience Group Metrics for the Group Cardinality and the inherent Replication Protocol, consistency model and synchronicity model
    - Ex: ***N*= *3*f+1** for ***f*** byzantine failures, …. etc