Dependable Systems

Dependability Threats

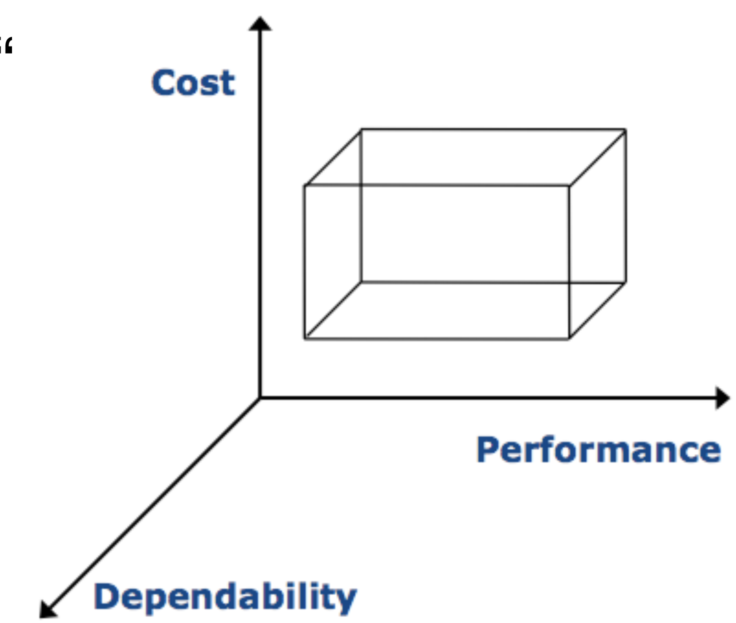
Dr. Peter Tröger

Sources:

J.C. Laprie. Dependability: Basic Concepts and Terminology
Eusgeld, Irene et al.: Dependability Metrics. 4909. Springer Publishing, 2008
Echtle, Klaus: Fehlertoleranzverfahren. Heidelberg, Germany : Springer Verlag, 1990.
Pfister, Gregory F.: High Availability. In: In Search of Clusters. , S. 379-452

Dependability

- **Umbrella term** for **operational** requirements on a system
  - IFIP WG 10.4: "*[..] the trustworthiness of a computing system which allows reliance to be justifiably placed on the service it delivers [..]*"
  - IEC IEV: "*dependability (is) the collective term used to describe the availability performance and its influencing factors : reliability performance, maintainability performance and maintenance support performance*"
  - Laprie: „ *Trustworthiness of a computer system such that reliance can be placed on the service it delivers to the user* “
- Adds a third dimension to system quality
- General question: How to deal with unexpected events ?
- In German: ‚Verlässlichkeit‘ vs. ‚Zuverlässigkeit‘

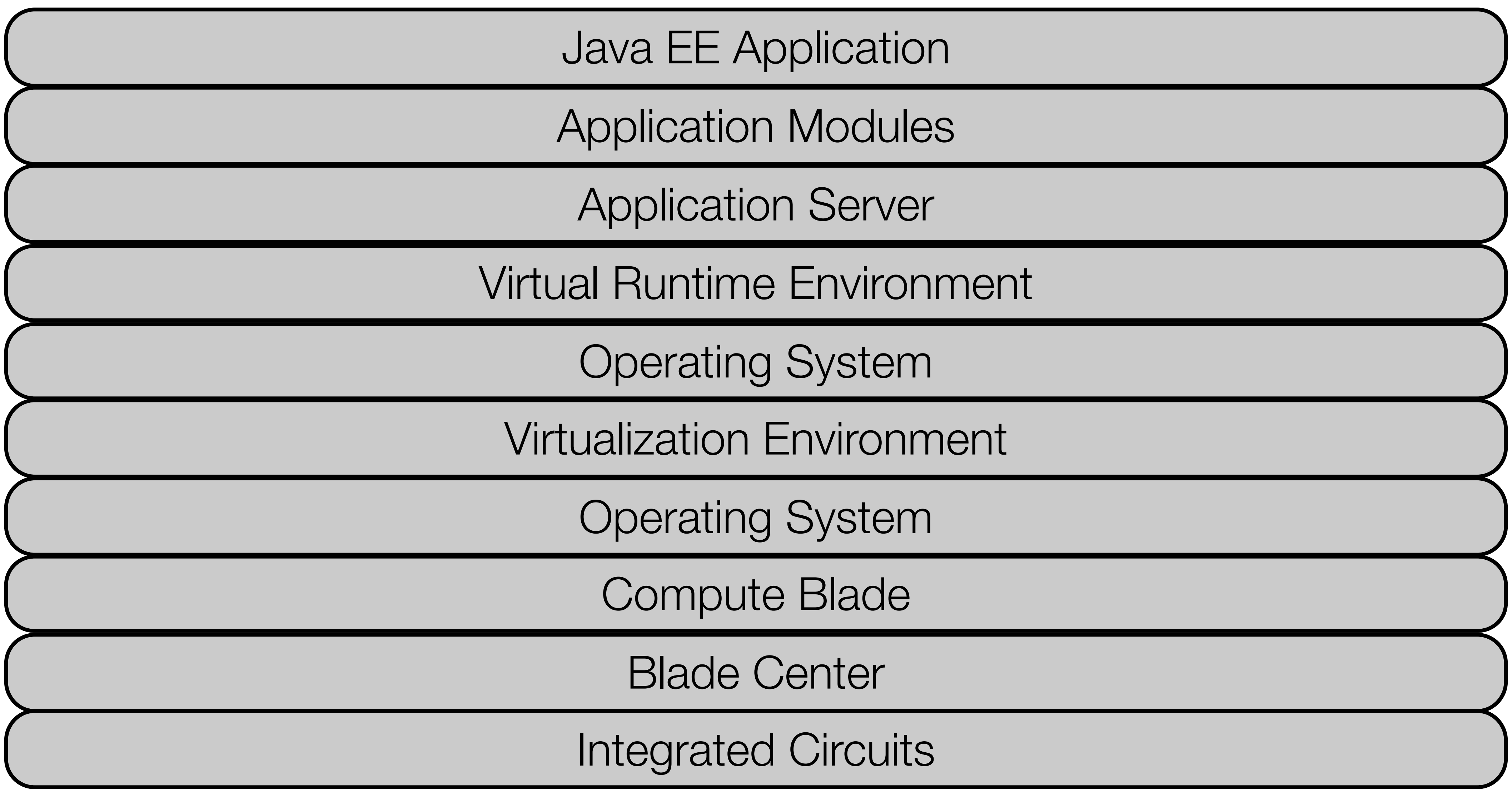


System Type Examples

- **Dependable (reliable) system**
  - Delivers a required service during its lifetime
- **Fault-tolerant computer system**
  - Continues correct service provisioning in the presence of faults
- **Real-time computer system**
  - Deliver a service within given time constraints (physical time, duration, ...)
- **Responsive computer system**
  - Fault-tolerant real-time system

System Integration Levels

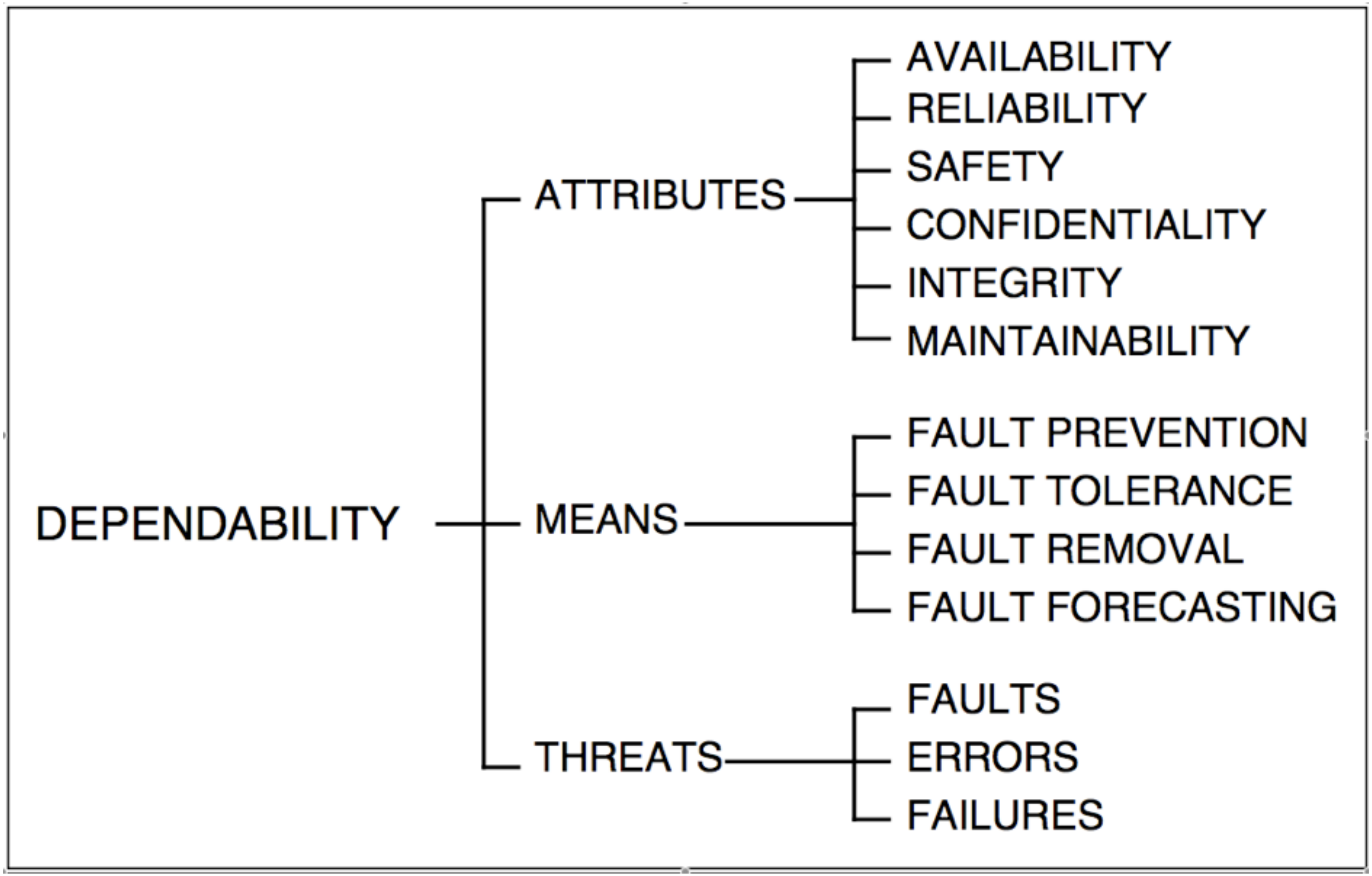
- Dependability has to be considered at every level
- Decomposition approach influences dependability success

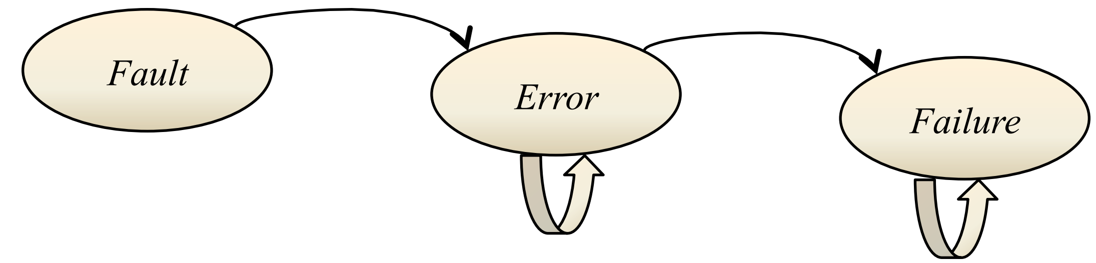


Dependability Stakeholders

- **System** - Entity with function, behavior, and structure
  - A number of components or subsystems, which interact under the control of a design [Robinson]
- **Service** - System behavior abstraction, as perceived by the user
- **User** - Human or physical system that interacts with the systems service
- **Specification** - Definition of expected service and delivery conditions
  - On different levels, can lead to specification fault
- Reliance demands assessment of **non-functional dependability attributes**
- Provide ability for trustworthy service delivery by **dependability means**
- Undesired (maybe expected) circumstances form **dependability threats**

Dependability Tree (Laprie)

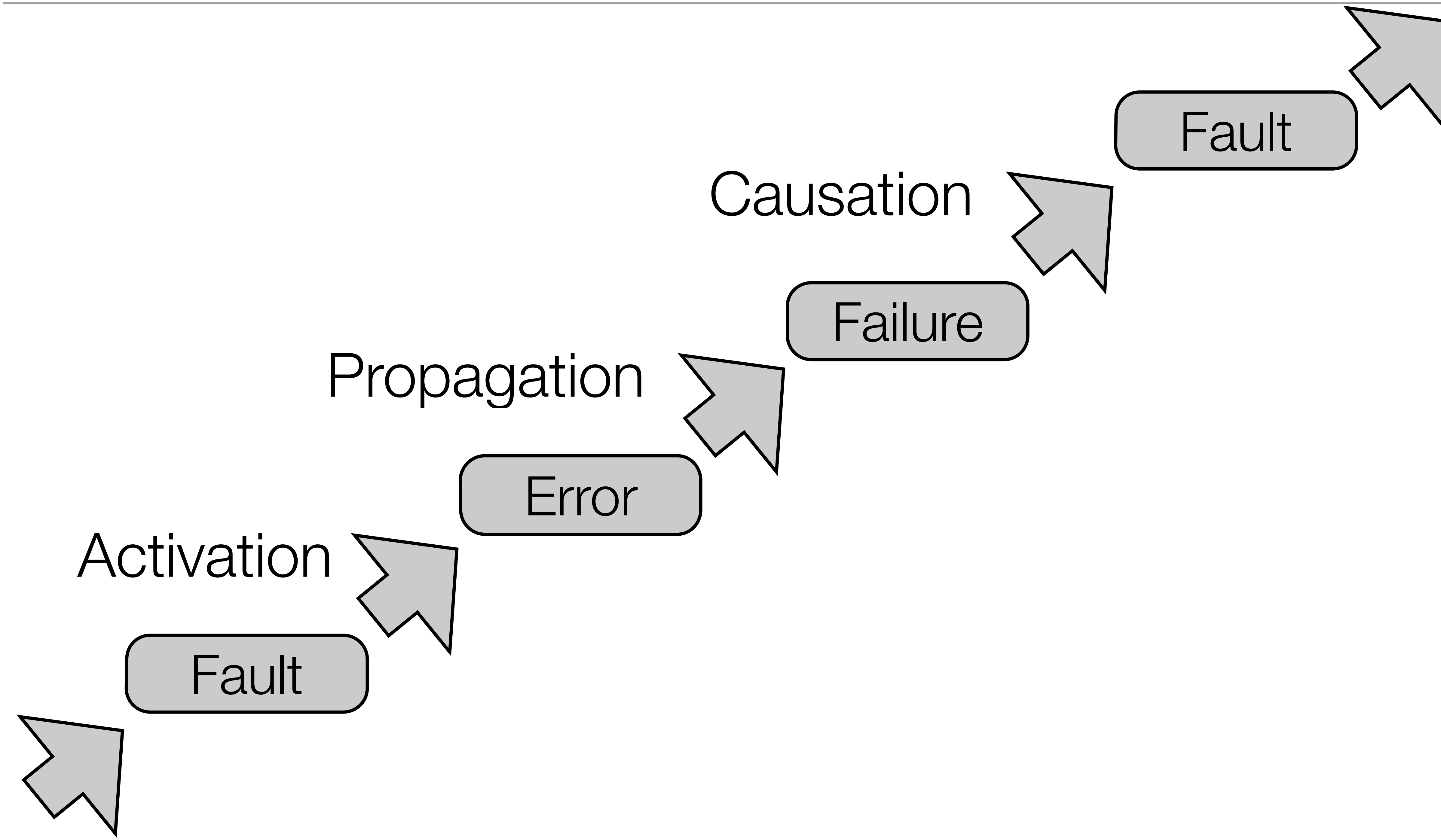




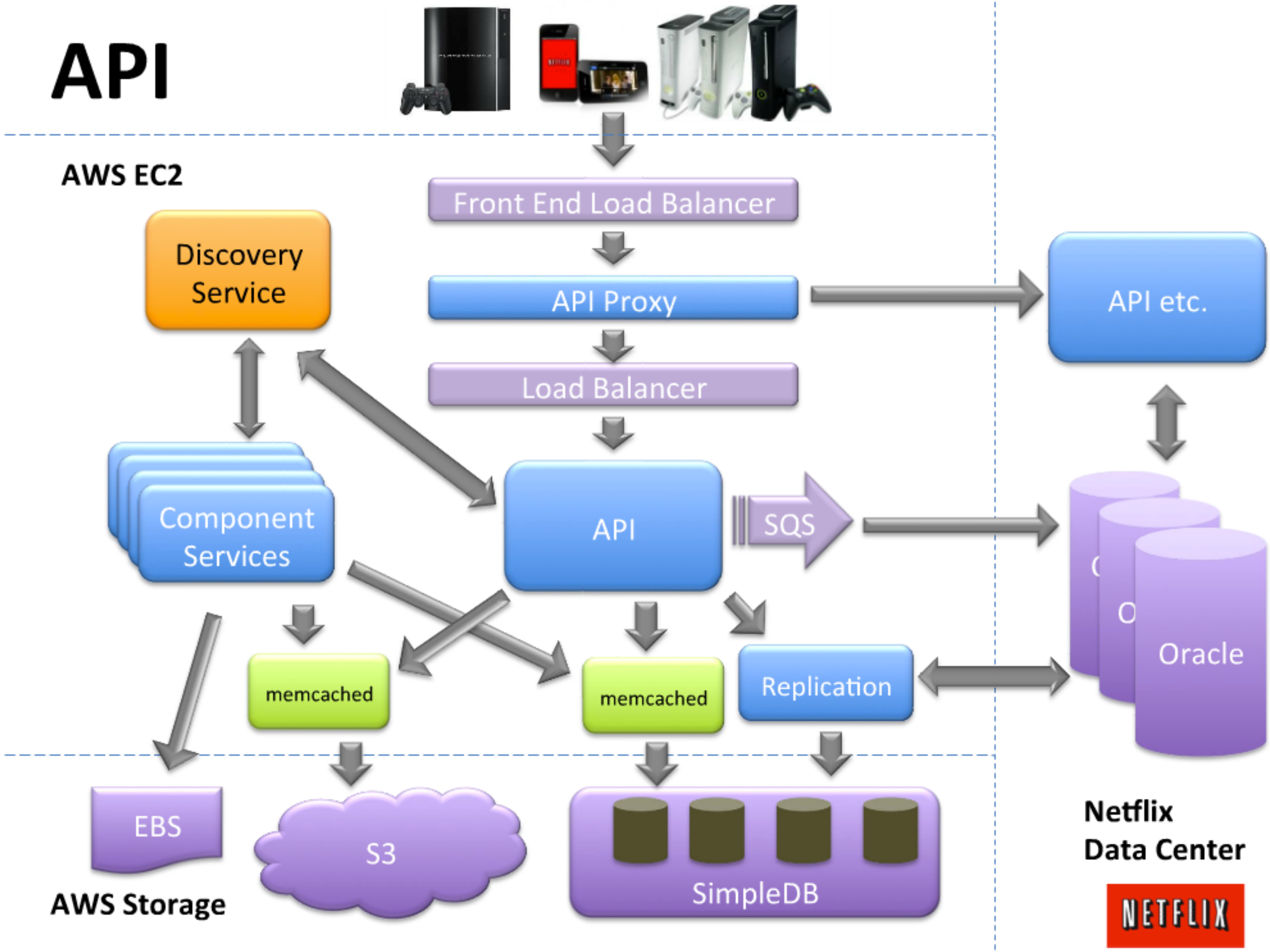
Dependability Threats

- System **failure - ‚Ausfall'**
  - Event that occurs when the service no longer complies with the specification / deviates from the correct service.
- System **error - ‚Fehler(zustand)'**
  - Part of system state that can lead to subsequent failure
  - Some sources define errors as active faults - not in this course ...
- System **fault - ‚Fehler(ursache)'**
  - Adjudged or hypothesized cause of an error
- Failure occurs when error state alters the provided service
- Systems are build from connected components, which are again systems
- Fault is the consequence of a failure of some other system to deliver its service

Chain of Dependability Threats (Avizienis)



API



[http://blog.programmableweb.com/]

Faults

- High diversity in possible sources and types
  - Fault nature
    - **Accidental** faults (‚Zufallsfehler‘) vs. **intentional** faults (‚Absichtsfehler‘)
    - Intentional faults are created deliberately, presumably malevolently
  - Fault origin viewpoints (not exclusive)
    - Phenomenological causes: **Physical / natural** faults vs. **human-made** faults
    - System boundaries: **Internal** faults (part of system state that produces an error) vs. **external** faults (interference with the environment)
    - Phase of creation: **Design** faults vs. **operational** faults
  - Temporal persistence
    - **Permanent** faults vs. **temporary** faults

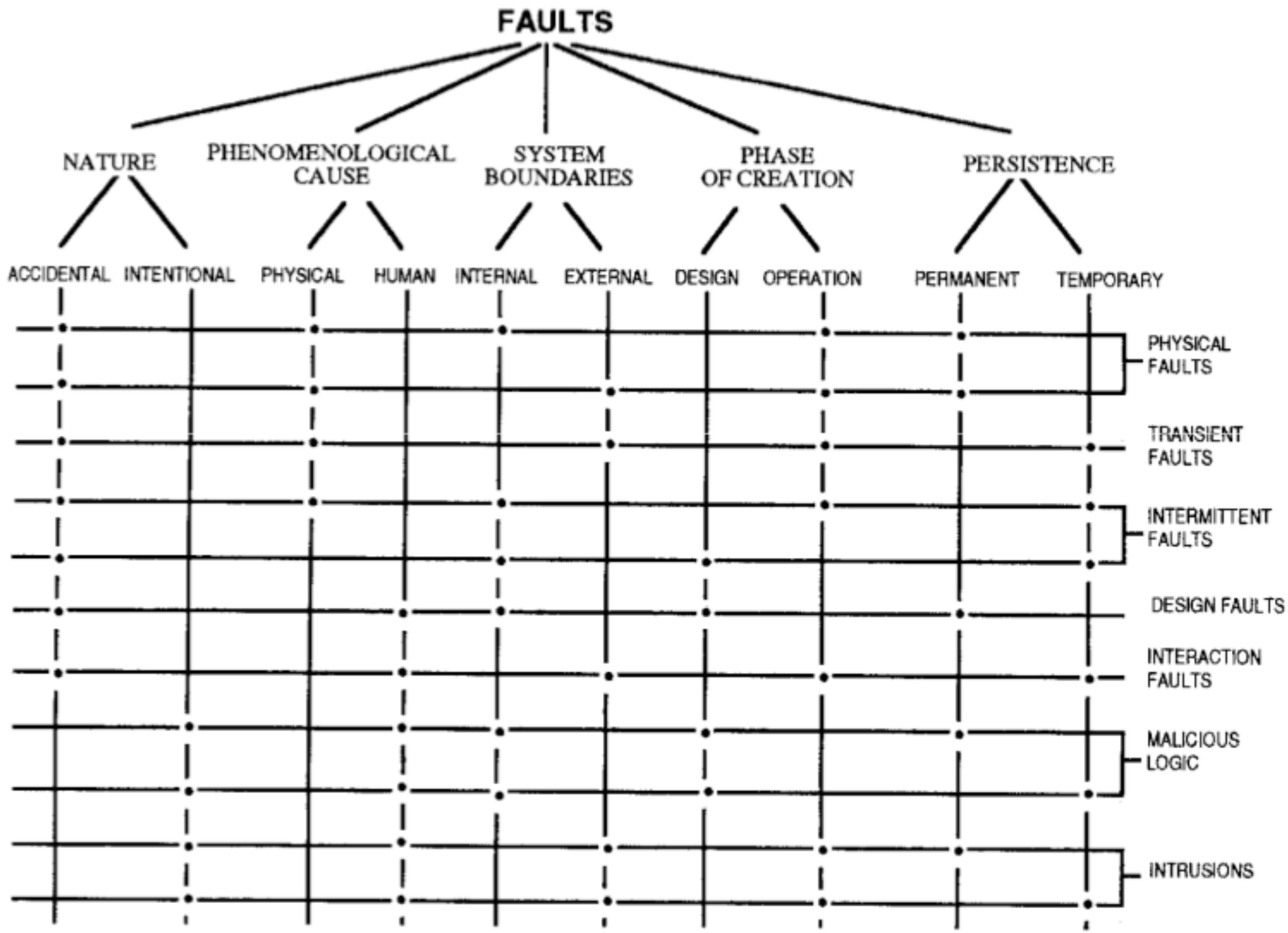
Observations on Faults

- An external fault is a design fault - inability or refusal to foresee all situations
- Design faults are created during system development, system modification, or operational procedure creation and establishment
- Just replacing broken version of the same component leads to **recurrent faults**
- Physical faults are **accidental faults**
- Temporary external accidental physical faults are also called **transient faults**
- Temporary internal accidental faults are also called **intermittent faults**
  - Examples: Pattern-sensitive memory hardware, system overload
  - Arbitrary concept - Permanent faults with unknown activation condition
- Intentional and design faults are human-made faults, might be **malicious faults**
- Hardware production defects are typically **physical faults**

Observations on Faults

- A fault is **active** when it produces an error
- A non-active internal fault is a **dormant / passive fault** (‚inaktive Fehlerursache')
  - Origin in hardware fault analysis - often cycling between dormant and active
- Many specialized versions of the term ‚fault', e.g. **bug**
  - **Heisenbug** - Intermittent software fault, **Bohrbug** - Permanent software fault
  - **Mandelbugs** - Appear chaotic due to many dependencies
- **Fault-tolerant system design** is a contradiction
  - Design demands specification, faults are non-specified cases
  - Solution: Specification for fault-free case + additional fault specification
- Fault can mean performance or timing faults (derivation from expected load / timing)

Fault Characterization (Laprie & Kanoun)



Fault Model

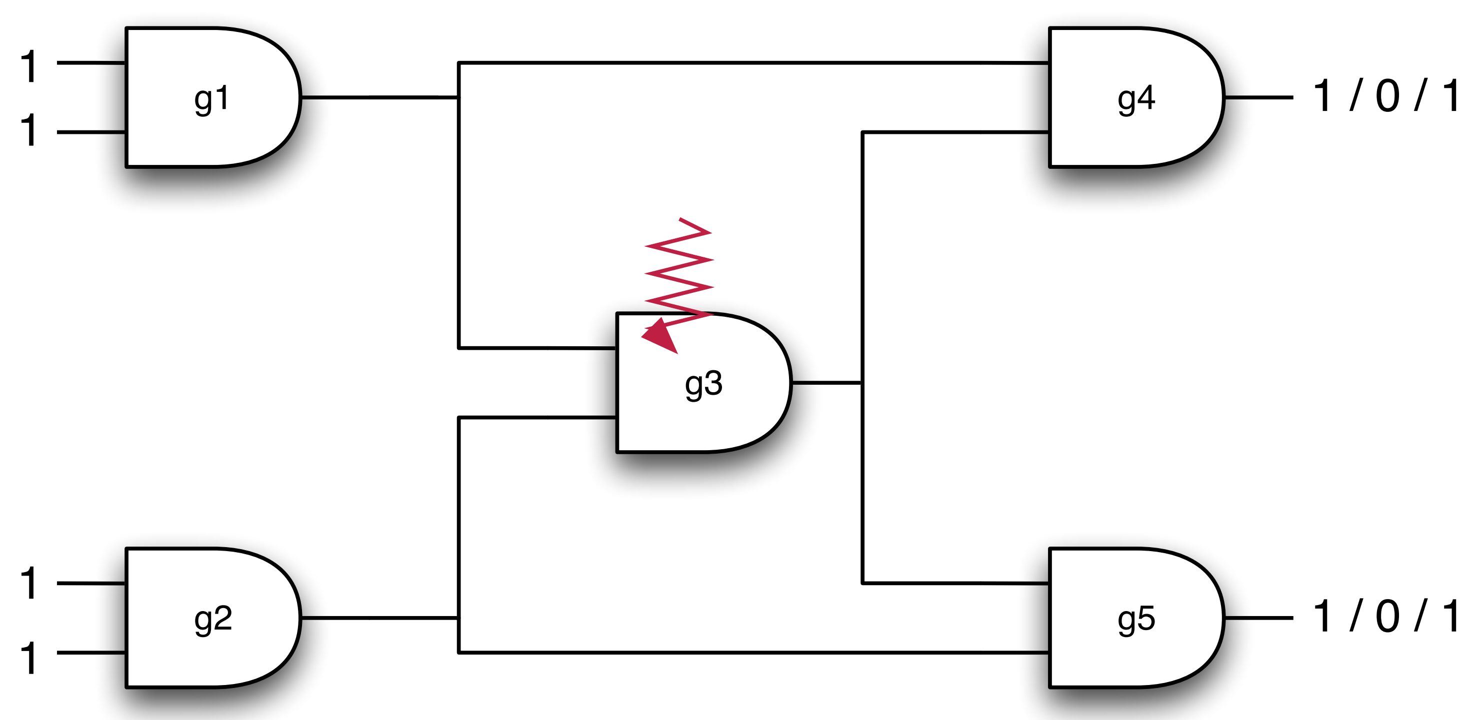
- Faults can be classified into different categories on different abstraction levels
  - Physics
  - Circuit level / switching circuit level
    - Interesting for hardware design research (not this course)
    - Investigate logical signals on connections
      - stuck-at-zero, stuck-at-one, bridging faults, stuck-open
  - Register transfer level
  - Processor-memory-switch (PMS) level
  - Hardware system level
  - ... (Software) ...

Physical Faults [Goloubeva]

- Highly energized particles originate from space, atmospheric, or ground radiation
  - Cosmic radiation, solar heavy ions, solar protons, ...
- Interaction of particle that strikes a circuit - atomic displacement, direct ionization, indirect ionization created by nuclear reactions
- Smaller structures are sensitive to ionization effects from all kinds of particles
- **Single Event Upset (SEU)** - injected charge modifies memory information
- Dynamic random access memory (DRAM) - typical building blocks for main memory
  - No inherent refreshing, storage capacitor changes value
- Static random access memory (SRAM), for caches, registers, pipeline, ...
  - Impact on restoring transistor leads to invalid refresh operation

Physical Faults [Goloubeva]

- Logic circuits: Shrinking size, reduction of power supply, increase of frequency
  - Noise margin is extremely reduced, single-event strike impacts circuit lines
- **Single Event Transient (SET):** Particles modify voltage in a combinational circuit
  - Can be modeled at gate level as erroneous transition on the gate output



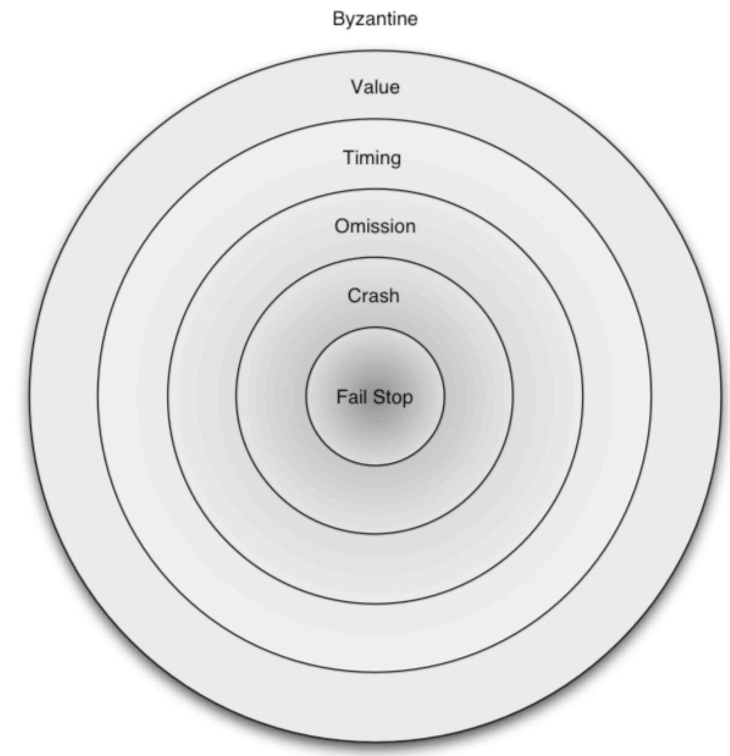
Fault Model for Semiconductor Memories

- **Stuck-at**-1 or stuck-at-0 (hard) faults, **transition / bit-flip faults** (0->1, 1->0)
- **Open and short circuits** - Too much or too little metallization; Also open bonds
- **Input and output leakage** - Leakage current in excess of the specified limit
- **Multiple writing** - Data written into more than one cell on write attempt in one cell
- **Pattern sensitivity** - Device does not perform reliably with certain data pattern(s)
- **Refresh dysfunction** - Data is lost during the specified minimum refresh time
- **Write recovery** - Write followed by read/write at different location results in read/write at same location
- **Sense amplifier recovery** - Data accessed remains the same for a number of cycles and then suddenly changed
- **Sleeping sickness** - Memory loses information in less than the stated hold time (typically tens of milliseconds)

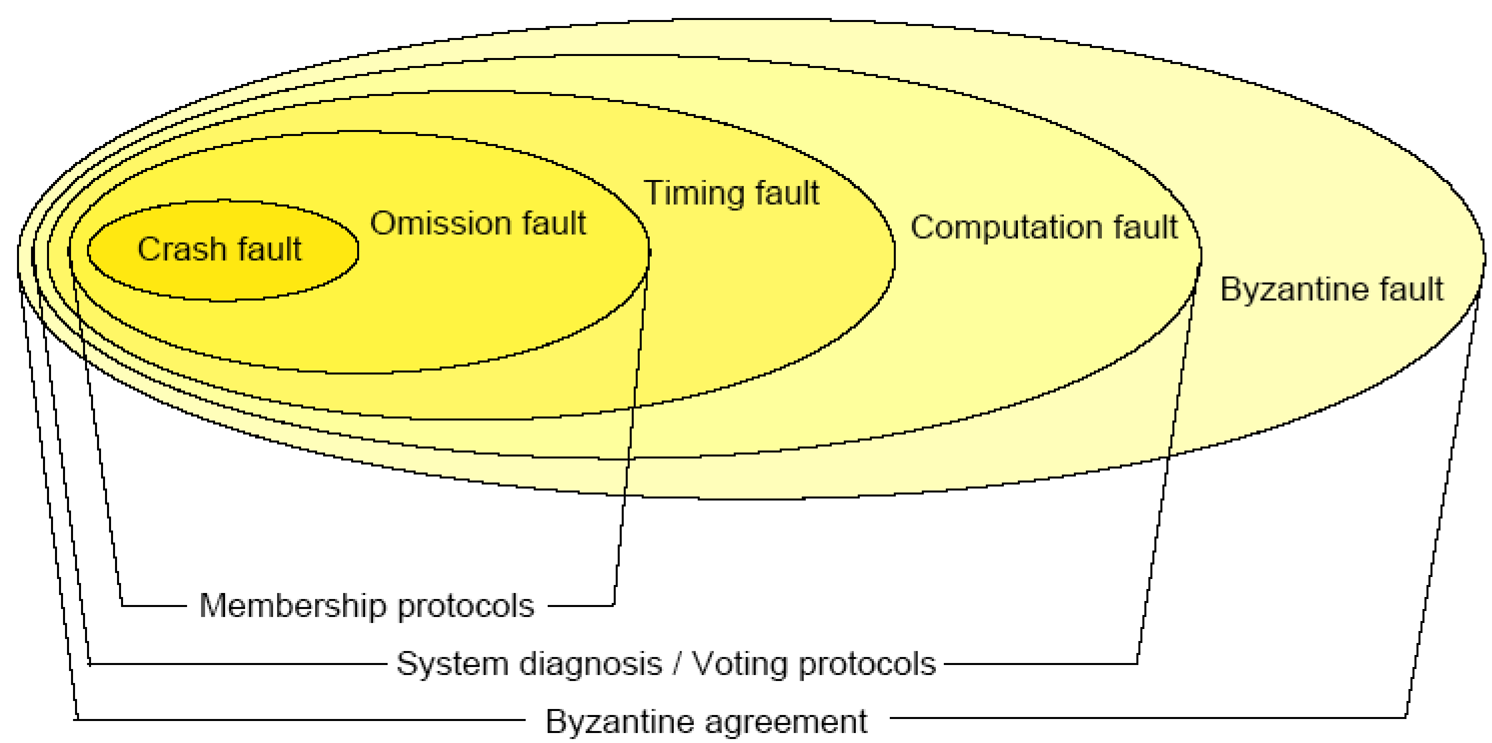
Fault Model for Semiconductor Memories

- **Decoder malfunction** - Inability to address same portions of the memory array
  - No cell accessed by certain address, multiple cells accessed by certain address
  - Certain cell not accessed by any address
  - Certain cell accessed by multiple addresses
- **Bridging fault** - Short between cells, AND type or OR type
- **State coupling fault** - Coupled (victim) cell is forced to 0 or 1 if coupling (aggressor) cell is in a given state
- **Inversion coupling fault** - Transition in coupling cell inverts coupled cell
- **Idempotent coupling fault** - Coupled cell is forced to 0 or 1 if coupling cell transits from 0 to 1 or 1 to 0
- **Disturb fault** - Victim cell forced to 0 or 1 if we read or write aggressor cell (may be the same cell)

System-Level Fault Model



- Fault model idea originates from hardware
  - How many faults of different classes can occur ? What do I tolerate ?
  - Timing of faults: Fault delay, repeat time, recovery time, ...
- Also mappable to software or even complete systems
  - Activities as black box, only look on input and output messages
- Link faults are mapped to the participating components
- Every participating component would need a fault model - pick the most urgent ones



System-Level Fault Model [Cristian]

- **Fail-Stop** Fault : System stops all operations, notifies the other ones
- **Crash** Fault : System looses internal state or stops without notification
- **Omission** Fault : System will break a deadline or does not react to some task at all
  - Send / Receiver Omission Fault: Necessary message was not not sent / not received in time
- **Timing** Fault / **Performance** Fault : System stops / reacts to a task before its time window, after its time window, or never
- **Incorrect Computation** Fault : No correct output on correct input
- **Byzantine** Fault / **Arbitrary** Fault : Every possible fault
  - Authenticated Byzantine Fault : Every possible fault, but authenticated messages cannot be tampered
- This maps to both shared-memory and shared-nothing systems (*system of systems*)

Vulnerabilities as Security Faults

- Different dependability attributes might lead to different terminology
- Example: Vulnerability assessment for nuclear *security* [Johnston]
  - **Threat**: Who might attack against what asset, using what resources, with what goal in mind, when / where / why, with what probability
  - **Threat assessment (TA)**: Attempting to predict the threats - proactive security
  - **Vulnerability**: Specific weakness in security that could be exploited (fault)
  - **Vulnerability assessment (VA)**: Attempting to discover / demonstrate them
  - **Risk management**: Deploy, modify, and re-assign security resources, based on TA results, VA results, assets, security breach consequences, and costs (time, money, human resources)
  - **Attack**: Attempt to harm valuable asset by exploiting one or more vulnerabilities, may lead to security failure

Security - Vulnerability Assessment [Johnston]

- Threats and vulnerabilities are different concepts, and must be treated separately
  - Vulnerabilities without threats are not interesting
  - Vulnerabilities do not define threats (bad locks do not imply thieves to show up)
- No one-to-one mapping, different attacks can exploit the same vulnerability
- TA involves mostly speculation about unknown people, so VA is more important
- Correct VA should identify large amount of issues with cheap countermeasures
- System features can become a vulnerability only in combination with an attack
- TA and VA are not pass / fail certifications

Errors

- State of the system, not an event !
- Escalates to failure depending on ...
  - ... intentional / unintentional redundancy
  - ... system activity
  - ... specification of a failure case from user perspective
    (i.e. maximum outage time, acceptable delay, retransmission rate)
- System activity can reverse the error state before damage is happening
- **Latent** (not recognized) vs. **detected error** resulting from an active fault
- Hardware often contains unintentional redundancy, makes it difficult to test

Hardware Error Models [Goloubeva]

- Hardware faults effect state information, e.g. register values
  - Stuck-at and other hardware faults therefore can also be denoted as error
- More interesting to investigate resulting effects on system-level
  - **Single data error** - Program data is corrupted (in cache, memory, or register)
  - **Single code error** - Effect on one instruction of the code
    - **Type 1/2** - Instruction modification without / with change of control flow
- Nature of error state may confirm to the nature of the originating fault
  - Transient vs. permanent, static vs. dynamic, single vs. multiple
  - Depends on utilized dependability means

Hardware Error Models [Goloubeva]

- Mapping of hardware-level single bit-flip error to other layers
  - **Memory data segment, processor data cache**: System-level single data error
  - **Memory code segment, processor code cache:** System-level single code error of type 1 (modification of target register) or type 2 (modification of branch target)
  - **Memory stack segment:** System-level data error or type 2 code error
  - **Processor register:** Depending on processor architecture and register type
    - Single data error if register holds data interpreted by the application
    - Single type 1 code error, if register holds address used by load/store operation
    - Single type 2 code error, if register holds address of a branch target
  - Processor control register: Everything could happen ...

Hardware Error Models - Code Errors [Goloubeva]

```
       MOV R0, 10
       MOV R1,  1
LOOP:  ADD R1, R1
       SUB R0,  1
       BNZ LOOP
```

→

```
       MOV R0, 10
       MOV R1,  1
LOOP:  SUB R1, R1
       SUB R0,  1
       BNZ LOOP
```

```
       MOV R0, 10
       MOV R1,  1
LOOP:  ADD R1, R1
       SUB R0,  1
       BNZ LOOP
```

→

```
       MOV R0, 10
       MOV R1,  1
LOOP:  ADD R1, R1
       SUB R0,  1
       BNZ FOOBAR
```

```
       MOV R0, 10
       MOV R1,  1
LOOP:  ADD R1, R1
       SUB R0,  1
       BNZ LOOP
```

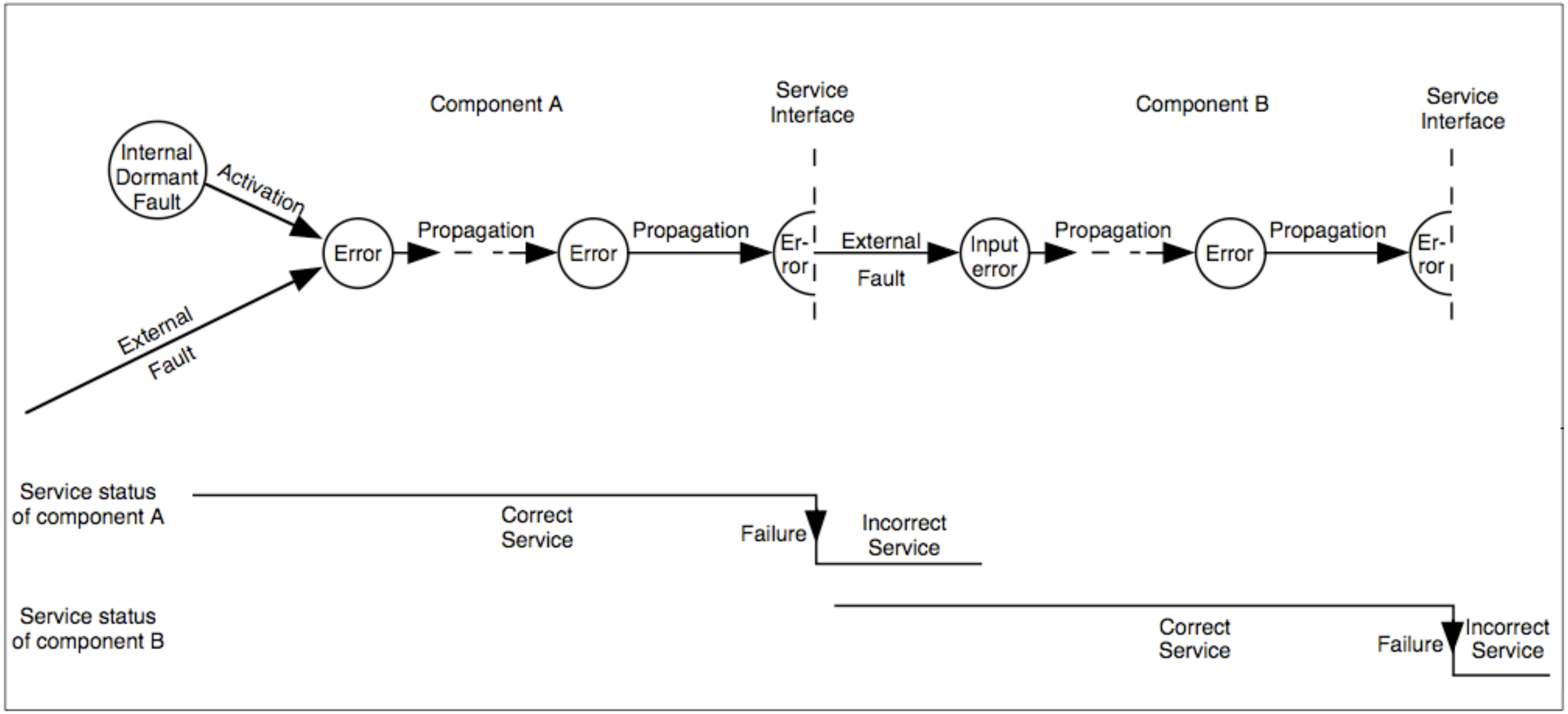
→

```
       MOV R0, 10
       MOV R1,  1
LOOP:  ADD R1, R1
       SUB R0,  1
       BZ LOOP
```

Software Error Models [Goloubeva]

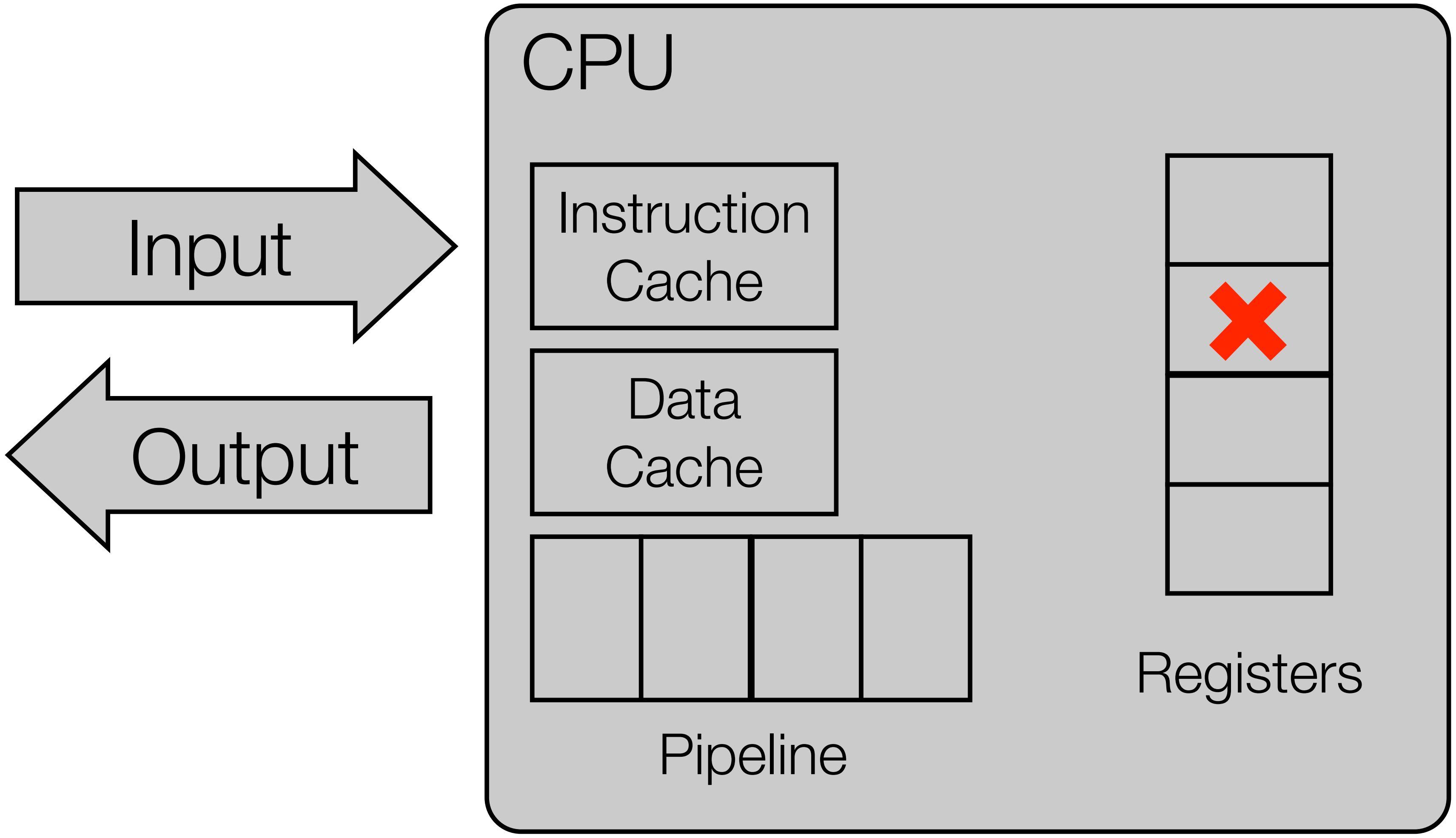
- Similar terminology, but completely different semantics
- **Syntactical errors** are handled by compiler, **semantical errors** occur at runtime
  - Static vs. dynamic, permanent vs. temporary errors
- Example for C programming language
  - Errors affecting assignments (missing / wrong local variable values)
  - Errors affecting conditional instructions (wrong boolean or iteration condition)
  - Errors affecting function call / return (wrong parameters, return statement)
  - Errors affecting algorithms (missing statements or function calls, wrong operators)
- Under research in the software engineering field - field studies, automated code analysis, developer interviews

Error Propagation



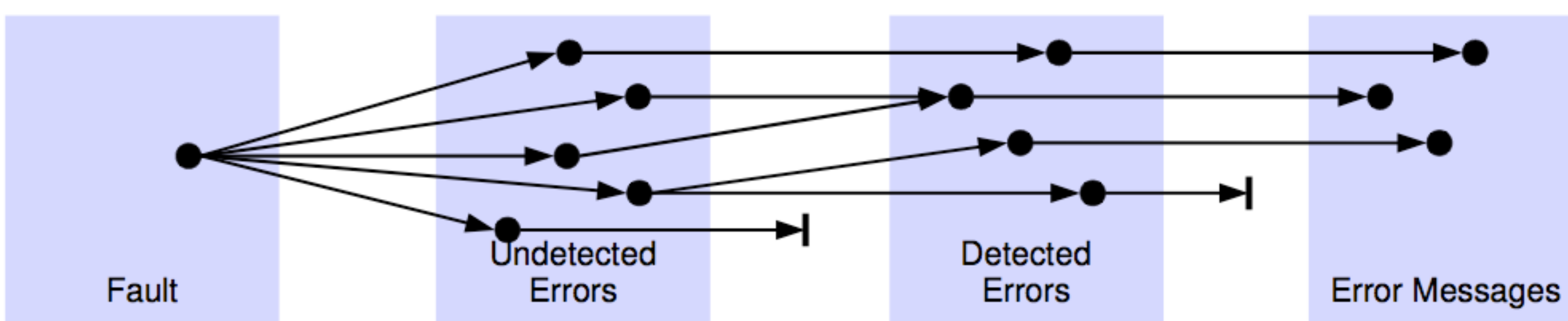
(C) Avizienis

Error Propagation [Goloubeva]



Error Message Occurrence (Hansen & Siewiorek)

- Same fault can lead to different (detected or undetected) errors
- Errors become detected by error detection mechanism
    - Some undetected errors are detected by several detectors
    - Some detectors report several undetected errors as one
    - Some undetected errors are never uncovered
- Detected errors might not be logged, if the system stops too fast



Hazard

- Several domains prefer the term **hazard** for a **safety error**
  - Situation (system state) that is threatening to life, health, property, or environment
  - An active hazard situation is an **incident**, leading to loss event called **accident**
  - Historically important in nuclear power, railroad and aviation industry
- Hazard analysis demands critical thinking
  - What can go wrong with which consequences ?

Failures

- Non-compliance with the specification - **arbitrary failure** (‘willkürlicher Ausfall‘)
- System failures can be further categorized in **failure modes**
  - **Fail-silent** / **crash failure** mode - incorrect results are not delivered
  - **Fail-stop** mode - constant value is delivered
- Failure mode **domain** - what is influenced
  - Service result - **value failures**
  - Service timeliness - **timing failures**
  - Service availability - **stopping failures**
- **User perception** in the mode - consistent / inconsistent for all users
- Failure mode **consequences** for ranking the identified issues

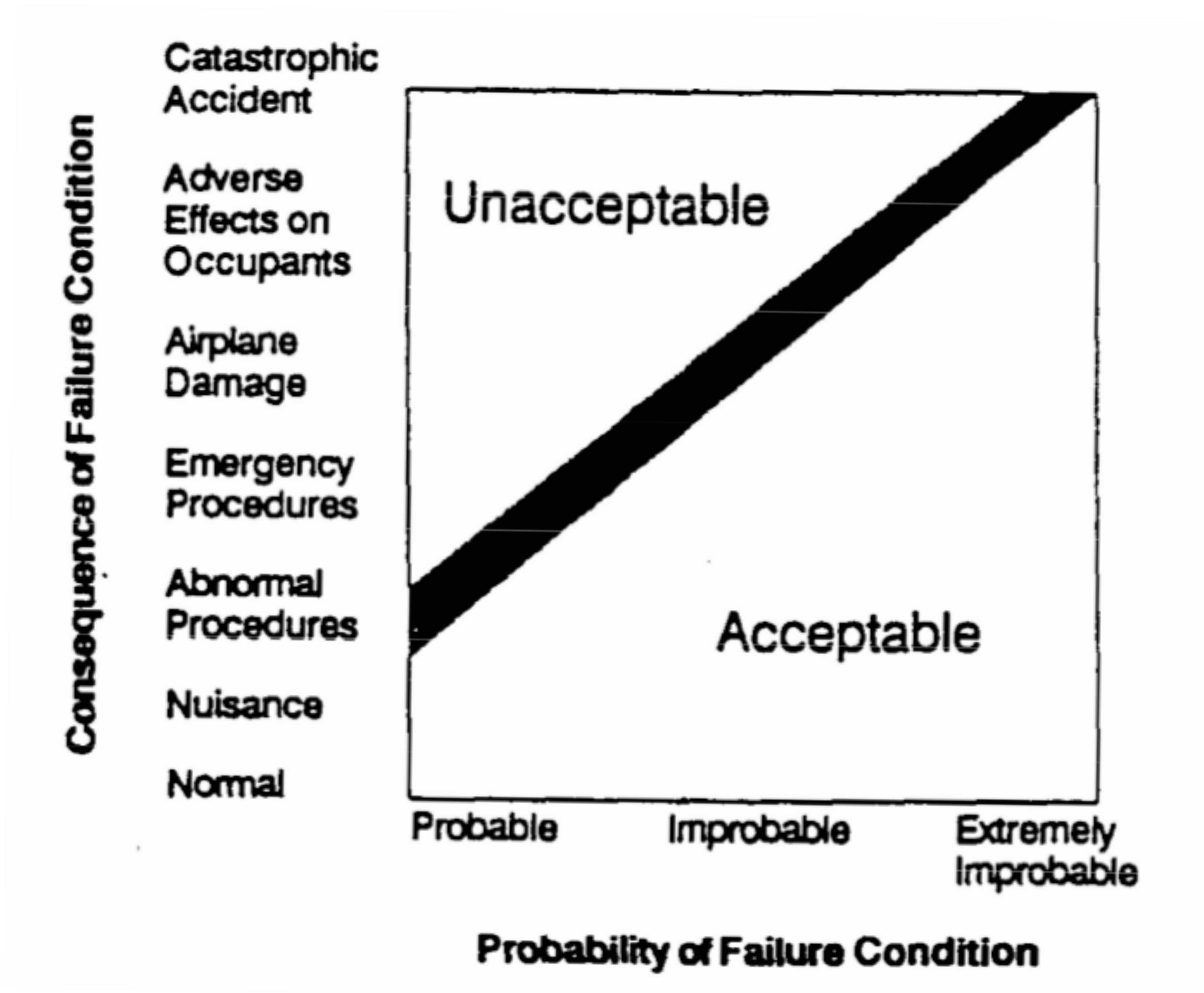
Failure Severity (‚Schweregrad des Ausfalls‘)

- Denotes consequences of failure
- **Benign failures** (‚unkritische Ausfälle‘)
  - Failure costs and operational benefits are similar
  - Sometimes also umbrella term for failures only detected by inspection
  - A system with only such failures is **fail-safe**
- **Catastrophic failures** (‚kritische Ausfälle‘)
  - Costs of failure consequences are much larger than service benefit
- **Significant / serious failures -** Intermediate steps expressing reduced service
- Grading of failure consequences on overall system depends on application
  - Flying airplane - Catastrophic stopping failure, Train - Benign stopping failure
- **Criticality** - Highest severity of possible failure modes in the system

Criticality Levels Example: DO-178B Standard

- *Software Considerations in Airborne Systems and Equipment Certification*
  - Mature document, developed for more than 20 years
- Definition of **severity of failure conditions** for airplane, crew, and passengers
  - *Catastrophic* - Loss of ability to continue safe flight and landing
  - *Major* - Reduced airplane or crew capability to cope with operating conditions
    - Reduction in safety margins and functional capabilities
    - Higher workload or physical distress for the crew
  - *Minor* - Not significantly reduced airplane safety, slight increase in workload (Example: Change of flight plan)
  - *No effect* - Failure results in no loss of operational capabilities and no increase in crew workload

Example: DO-178B Standard

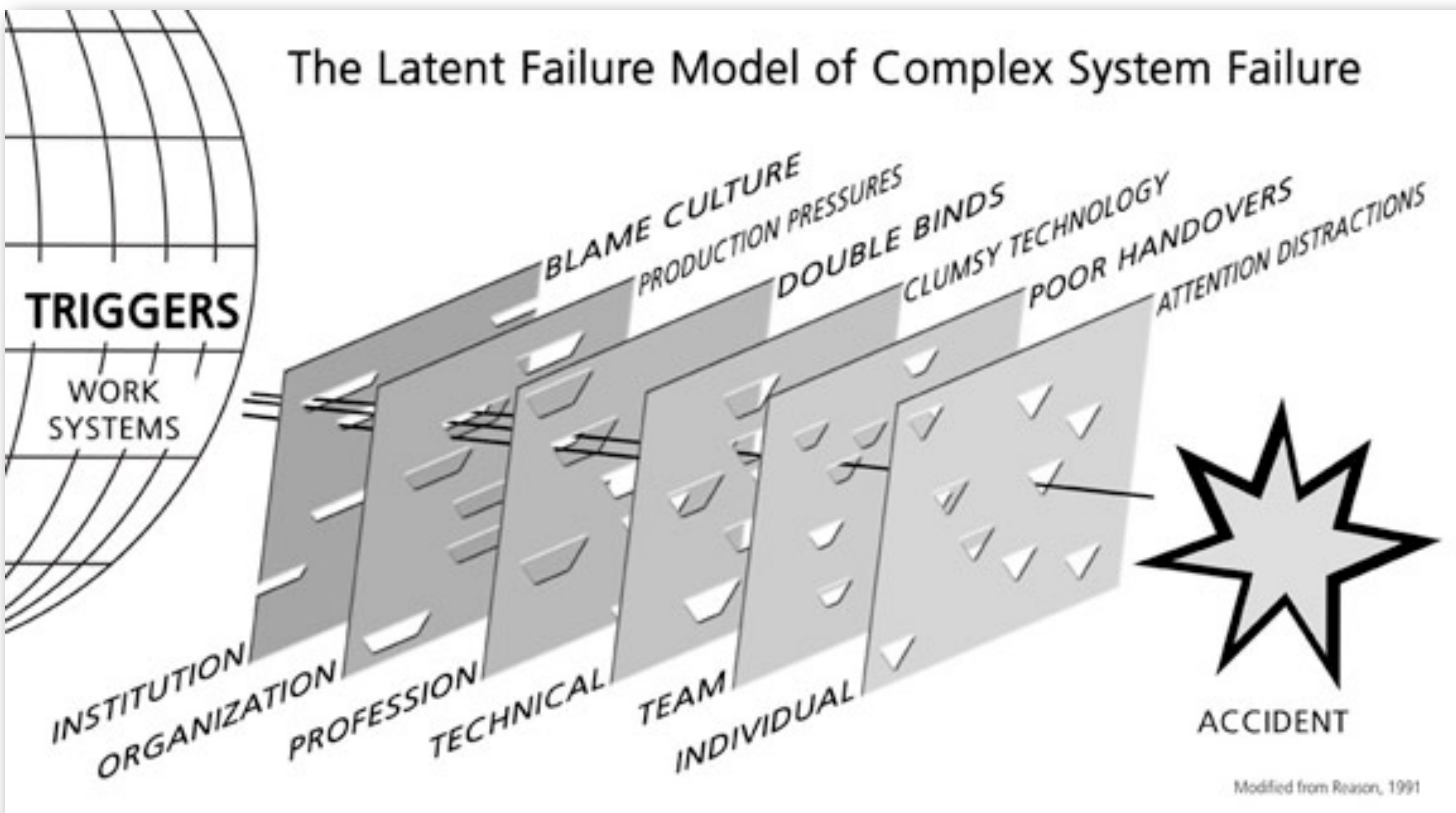


Failure Types

- Duration of the failure
  - **Permanent** failures - no possibility for repairing or replacement
  - **Recoverable** failures - back in operation after the system recovered from error state
  - **Transient** failures - short duration, no major recovery action
- Effect of the failure
  - **Functional** failures - system does not operate according to its specification
  - **Performance** failures - performance or SLA specifications not met
- Scope of the failure
  - **Partial** failure - only parts of the system become unavailable
  - **Total** failure - all services go down

Swiss Cheese Model (Prof. Reason)

- Origins in medical research
- Defenses, barriers, and safeguards might be penetrated by fault trajectory

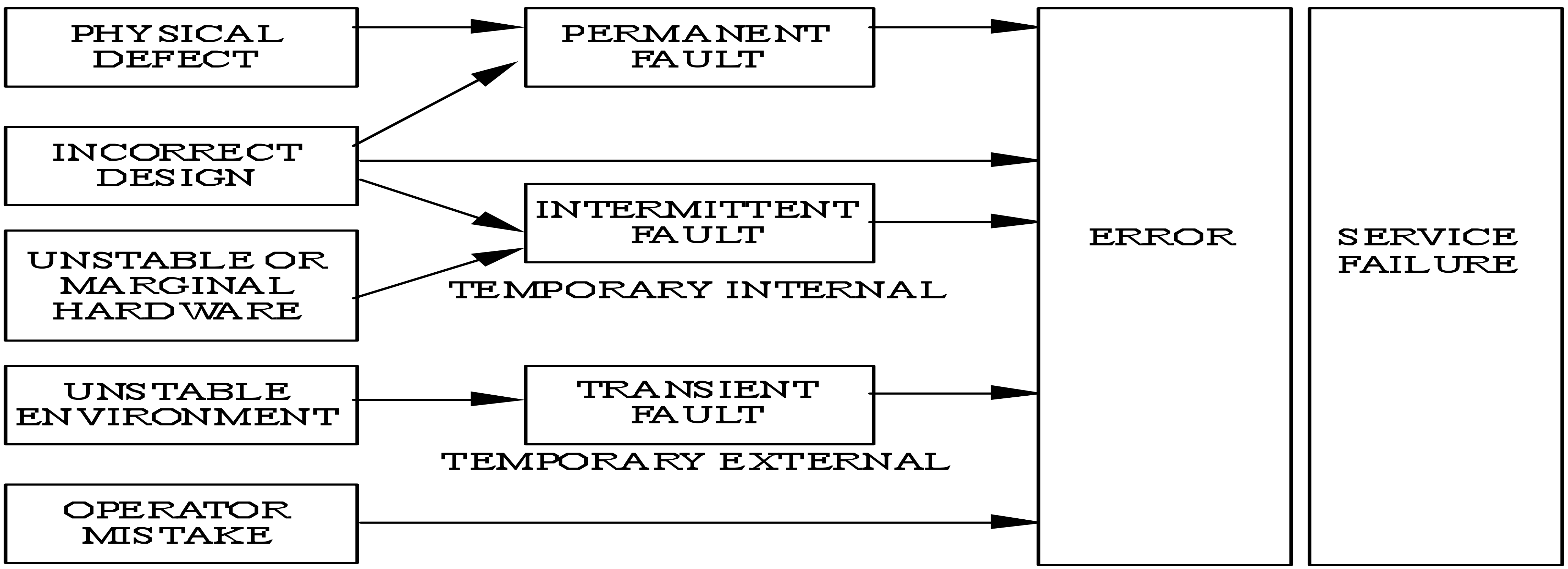


(C) Fernando Bernal

Observations on Failures

- Failures vs. Load
  - Typically positive correlation
    - Increasing load can lead to wear-out, so the failure probability increases
    - Higher load can activate dormant faults
    - Detected faults lead to recovery activities, which again increases the load
  - Possibility for unintended feedback effects in complex systems
- Related faults (attributed to a common cause) can lead to **common-mode failures**
  - Mostly reasoned by design faults that impact redundant copies of the component

Chain of Dependability Threats



SOURCES OF ERRORS

[from Siewiorek and Swarz]