

Dependable Systems

Dependability Means

Dr. Peter Tröger

Sources:

J.C. Laprie. Dependability: Basic Concepts and Terminology

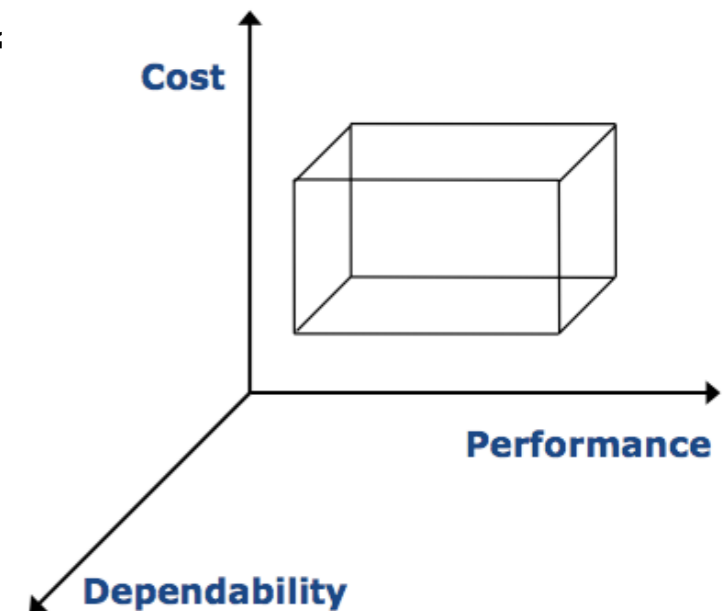
Eusgeld, Irene et al.: Dependability Metrics. 4909. Springer Publishing, 2008

Echtle, Klaus: Fehlertoleranzverfahren. Heidelberg, Germany : Springer Verlag, 1990.

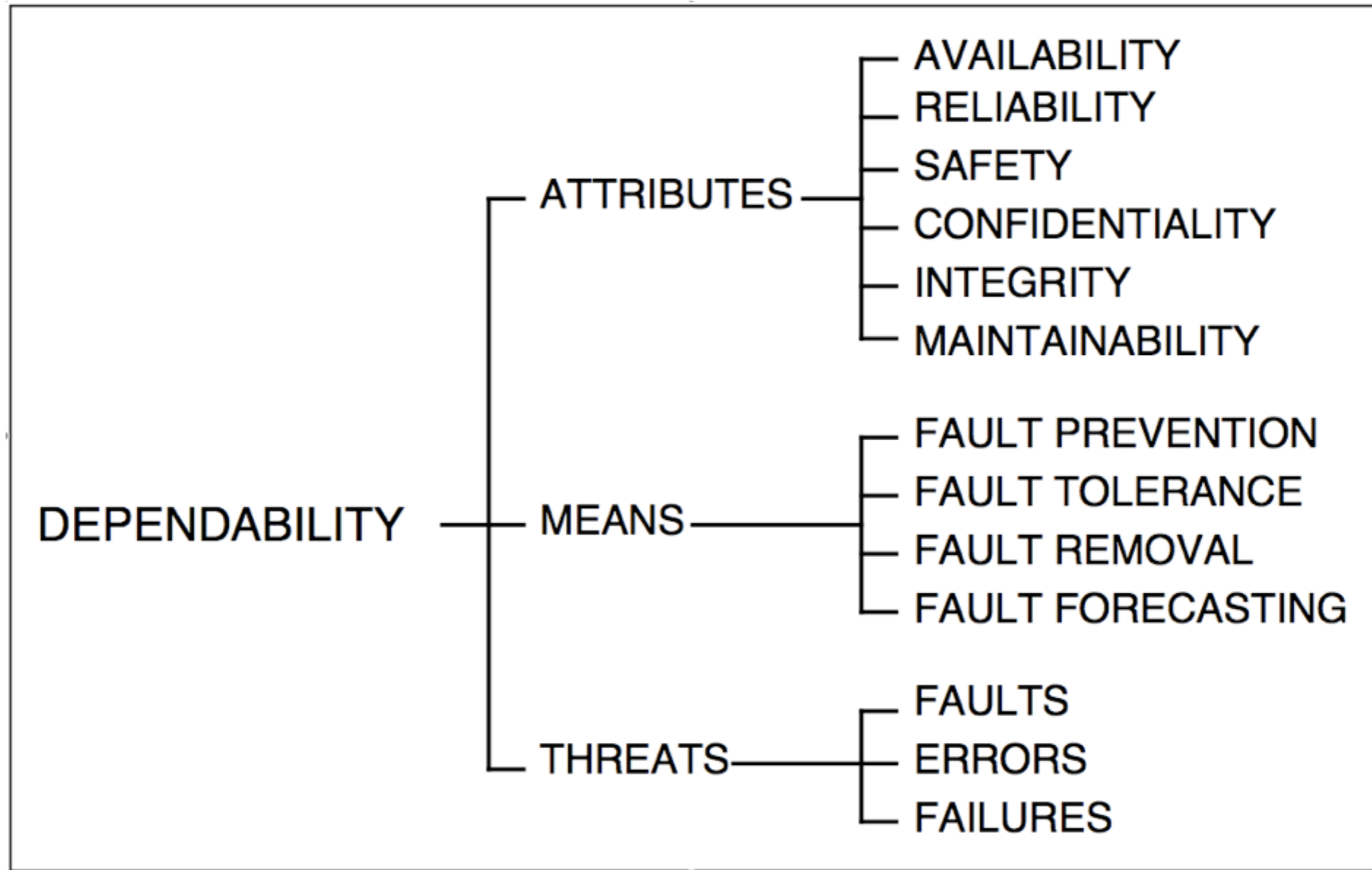
Pfister, Gregory F.: High Availability. In: In Search of Clusters. , S. 379-452

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‘

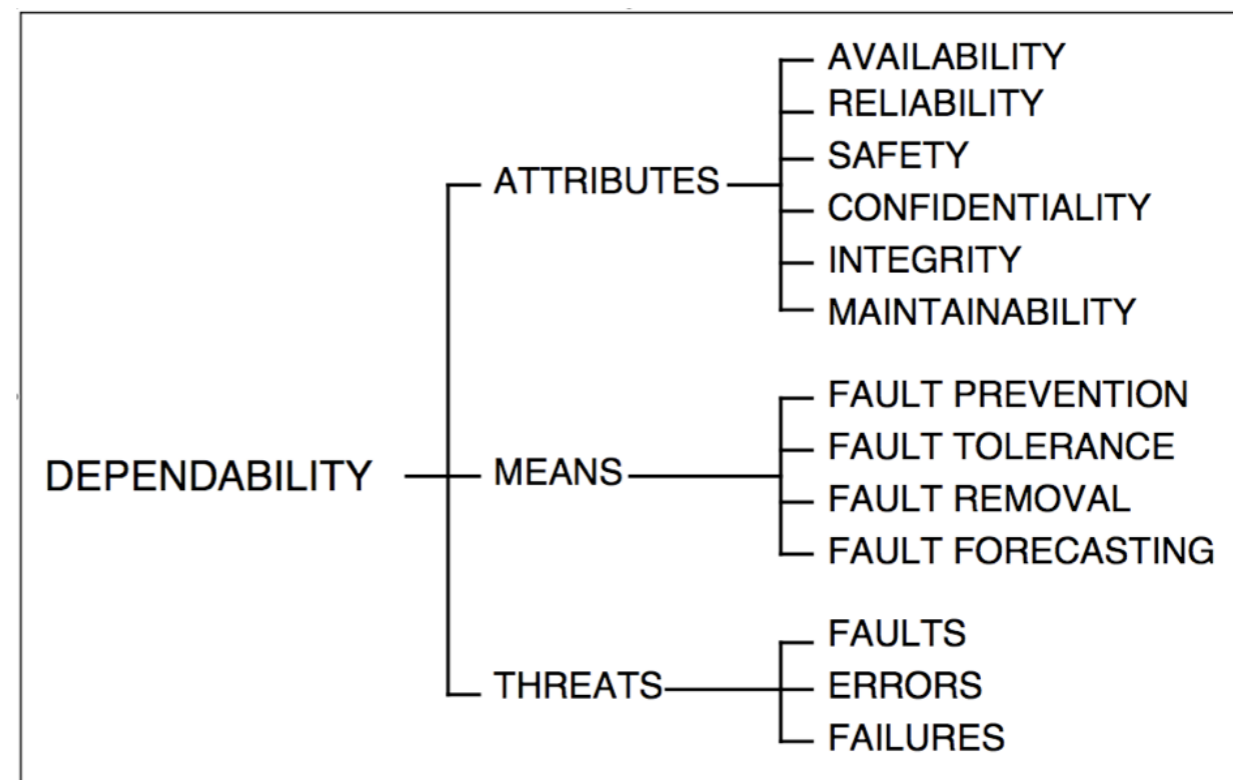


Dependability Tree (Laprie)



Means for Dependability

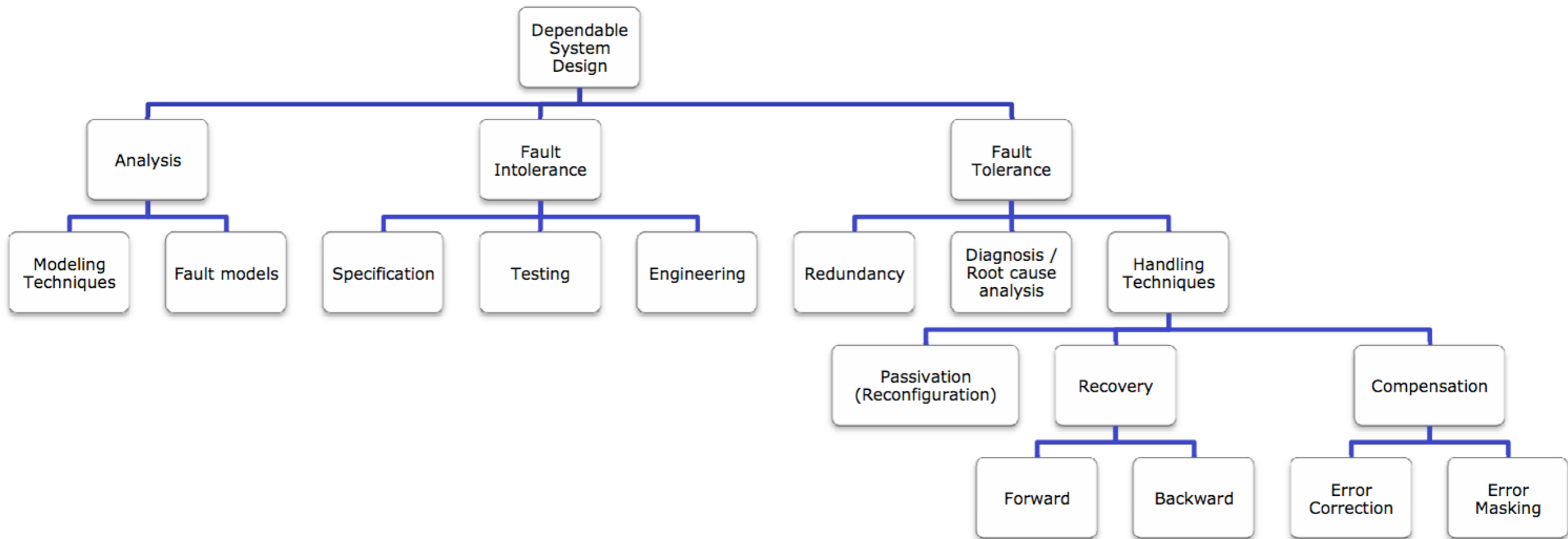
- **Fault prevention** - Prevent fault occurrence or introduction
- **Fault tolerance** - Provide service matching the specification under faults
- **Fault removal** - How to reduce the presence of faults
- **Fault forecasting**- Estimate the present number, future incidence, and the consequences of faults
- Combined utilization



Dependability Means (Laprie)

- Offline / online techniques
 - Fault intolerance techniques
 - **Fault prevention** - Prevent fault occurrence or introduction
 - **Fault removal** - Reduce the presence of faults
 - 100% fault-free servicing for the whole life time is not possible
 - Fault tolerance techniques
 - **Fault forecasting** - Estimate the present number, future incidence, and the consequences of faults
 - **Fault tolerance** - Provide service complying with specification in spite of faults
- Problems with **coverage** and **validation of the validator**

Dependability Means (Echtle)



Fault Prevention

- Specific approaches for avoiding faults
 - Specialized specification formalisms and techniques
 - Specialized development / manufacturing process to prevent design faults
 - Shielding
 - Only use ultra-reliable components
 - Choice of programming language
- General engineering approaches
 - Software engineering procedures
 - Quality management regulations and enforcement
 - Training and organization of maintenance departments

Fault Removal

- Make faults disappear before fault tolerance becomes relevant
- Step 1: Verification
 - Check if the system adheres to **verification conditions**; if not, take next steps
 - **Static verification**: Static analysis, data flow analysis, compiler checks
 - **Dynamic verification**: Symbolic execution or verification testing
- Step 2: Diagnosis
 - Find the faults that influenced the verification conditions
- Step 3: Correction
 - Fix the problem, repeat the steps (**regression**)
- Fault removal during operation: **Corrective maintenance (curative / preventive)**

Testing

- Selecting test inputs is driven from different view points
 - Testing purpose: **conformance testing, fault-finding testing**
 - System model: **functional testing** (with functional model) or **structural testing**
 - Fault model: enables **fault-based testing**
- Deterministic testing vs. random testing
- Structural testing of hardware is fault-finding, fault-based, structural testing
- Structural testing of software is fault-finding, non-fault-based, structural testing
- **Golden unit**: Reference system for comparison of output for a given input (e.g. reference implementation)

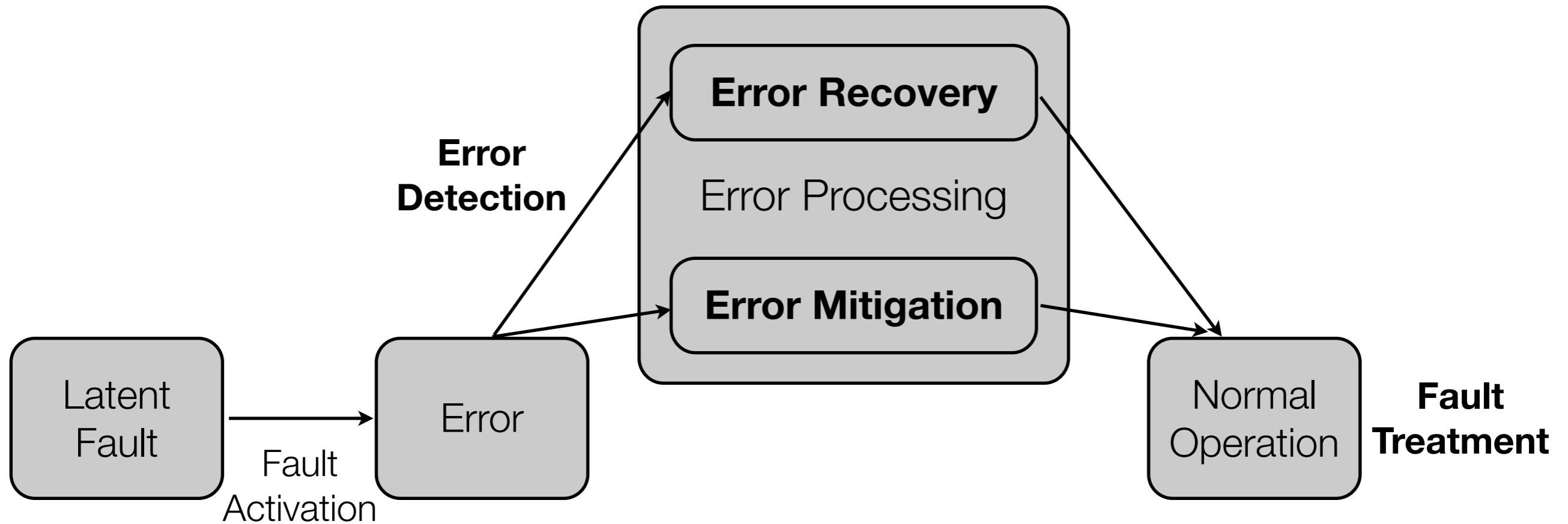
Fault Tolerance

- Fault tolerance is the ability of a system to operate correctly in presence of faults.
- or
- A system S is called **k-fault-tolerant** with respect to a set of algorithms $\{A_1, A_2, \dots, A_p\}$ and a set of faults $\{F_1, F_2, \dots, F_p\}$ if for every k -fault F in S , A_i is executable by a subsystem of system S with k faults. (Hayes, 9/76)
- or
- Fault tolerance is the use of **redundancy** (time or space) to achieve the desired level of system dependability - costs !
 - Accepts that an implemented system will not be fault-free
 - Implements automatic recovery from errors
 - Is a recursive concept (voter replication, self-checking checkers, stable memory)

Fault Tolerance

- Typical design methodology in many technical and biological systems
 - Spare wheel in cars, redundant organs, ...
- Fault tolerance mechanisms need to be evaluated by dependability attributes
 - Minimum, maximum, average reliability and availability
 - Easy to formulate and understand, hard to prove - failure rate remains unknown
 - Quantitative limits based on fault model (which faults in which components)
- Typically ,one-fault-at-a-time‘ assumption
- Different attributes of fault tolerance implementation to be checked
 - Functional verification, sensitivity analysis, minimum amount of resource resp. computational overhead, implementation performance, transparency, portability

Phases of Fault Tolerance (Hanmer)



Decomposition of Fault Tolerance (Lee & Anderson)

- **Error detection**

- Presence of fault is deduced by detecting an error in some subsystem
- Implies failure of the according component

- **Damage confinement**

- Delimit damage caused due to the component failure

- **Error processing - recovery / compensation**

- System recovers from the effect of an error

- **Fault treatment**

- Ensure that fault does not cause again failures

Fault Tolerance - Error Detection

- **Replication check**
 - Output of replicated components is compared / voted
 - Independent failures, physical causes -> many replicas possible (e.g. HW)
 - Finds also design faults, if replicated components are from different vendors
- **Timing checks** (,watchdog timers‘)
 - Timing violation often implies that component output is also incorrect
 - Typical solution for node failure detection in a distributed system
- **Reasonableness checks** - Run-time range checks, assertions
- Structural and coding checks, diagnostics checks, algorithmic checks
- Ideal: **Self-checking component** with clear **error confinement areas**

Fault Tolerance - Error Detection

- Replication checks are powerful and expensive, examples:
 - Execute identical copies on different hardware (component failures)
 - Execute separate and different versions (assumes independent design faults)
 - Execute same copies different times (transient faults)
 - Replicate only portion of the system
 - Works for both hardware and software
- Signaling aspect in the error detection task
 - Typical software model are exceptions, a way for implementing forward recovery
- Combination fault detection and fault location

Fault Tolerance - Damage Confinement (Taylor)

- **System decomposition**

- Every communication link might enable damage spreading
- Introduce mutual suspicion
- Hardware-based separation of software components
- OS-based separation (processes, runtime monitors, special shells)

- **Law-governed architecture**

- Externalize constraints on interaction by runtime rules

- **Strongly-typed language**

- Language guarantees the absence of unintended control flows

Fault Tolerance - Error Processing Through Recovery

- **Forward error recovery**

- Error is masked to reach again a consistent state (**fault compensation**)
- Corrective actions need detailed knowledge (**damage assessment**)
- New state is typically computed in another way
 - Examples with compensation: Error correcting codes, non-journaling file system check, advanced exception handlers, voters

- **Backward error recovery**

- Roll back to previous consistent state (**recovery point / checkpoint**)
- Very suitable for transient faults
- Computation can be re-done with same components (**retry**), with alternate components (**reconfigure**), or can be ignored (**skip frame**)

Forward & Backward Recovery - Example

- Transaction processing systems support ACID through journaling and backups
 - Journal: Audit trail of transactions and database changes
 - Forward recovery: Restore a database backup and apply the transaction journal
 - Journal provides the intended transaction operation (write-ahead logging) and the **,after-image'** for completed transactions
 - Replay of all conducted and intended transactions from safe point in history
 - Backward recovery (,backout'): Apply journal ,backwards' on current database
 - Demands **,before-image'** before each database modification
 - Involves logic of reprocessing aborted transactions, which is time-consuming
- Typically both applied - forward recovery for reaching a stable database status, and backward recovery of modifications from unfinished transactions

Fault Tolerance - Fault Treatment

- **Fault diagnosis** - determine error cause's location and nature
- **Fault passivation** - (remove faulty component &) reconfigure system
 - Error processing might already remove the fault - ,**soft fault**'
 - Typical examples are temporary faults
- Fault tolerance manager
 - Careful diagnosis with hardware support
 - Damage assessment by disabling faulty components automatically
 - Example: IBM mainframe architecture
- Software rejuvenation
 - Gracefully terminating an application and immediately restarting it at a clean internal state

Fault Tolerant Mindset (Hanmer)

- What can go wrong in any given situation ?
 - Mindset to be applied in all development stages
- „Every problem in computer science boils down to tradeoffs“ [Henschen]
 - Fault prevention vs. fault tolerance vs. failure severity
- KISS principles, leave out „bells and whistles“
- Incremental additions of reliability - long-term products
- Defensive Programming
 - Simple error handling; fix root cause, not symptoms; make data auditble; make code maintainable;

„Fail-Fast“

- A common concept from system engineering, company management, ...
- *„Report failure and stop immediately without further action“*
 - Discussed by Jim Gray in 1985 as part of his famous article *„Why do computers stop and what can be done about it ?“*
- Useful when benefit from recovery is not good enough for its costs, or if error propagation is highly probable
 - Single units of a redundant set
 - Deeply interwired IT system components
 - Components under heavy request load
 - And ... crappy start-up companies

Fault Tolerant Design Methodology (Hanmer)

- Assess things that can go wrong with the system (e.g. fault trees).
 - Find potential risks and according system failures.
- Define strategies to mitigate the identified risks.
 - Failure avoidance options, prevent faults from activation
- Create a mental model of the system design with redundancy.
- Design error detection and error processing capabilities.
- Design in the failure mitigation capabilities.
- Design human-computer interactions and modes of management.

Dependable Design Strategies (Malek)

- Decompose the system
 - Identify fault classes, fault latency and fault impact for the components
 - Identify “weak spots” and assess potential damage
 - Integrate partial recovery / reintegration / restart
- Determine qualitative and quantitative specs for fault tolerance and evaluate your design in specific environment
- Develop / utilize fault and error detection techniques and algorithms
- Develop / utilize fault isolation techniques and algorithms
- Refine fault tolerance, iterate for improvement
- Re-use proven components, but be aware of integration issues