Performance, Dependability, and Fault Tolerance in Distributed Systems

Hong-Linh Truong

linh.truong@aalto.fi

Note:

The content is based on the latest version that I lectured in 2014 in TU Wien

What is this lecture about?

- Service performance and service failures
- Basic performance metrics
- Dependability attributes, threats and means
- Basic mechanisms/algorithms of fault tolerance computing
- Performance and dependability of systems learned in other lectures

Learning Materials

- Main readings/references for this tutorial:
  - John Knight, Fundamentals of Dependable Computing for Software Engineers, CRC Press, 2012
    - Chapters 1-3
  - Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms, 2e, (c) 2007 Prentice-Hall
    - Chapter 8
  - George Coulouris, Jean Dollimore, Tim Kindberg, Gordon Blair„Distributed Systems – Concepts and Design“, 5nd Edition
    - Chapter 15
  - Algirdas Avizienis, Jean-Claude Laprie, Brian Randell, and Carl Landwehr. 2004. Basic Concepts and Taxonomy of Dependable and Secure Computing. IEEE Trans. Dependable Secur. Comput. 1, 1 (January 2004), 11-33

Outline

- Service performance and failures
- Performance
- Dependability
- Techniques for dealing faults

SERVICE/SYSTEM FAILURES AND QUALITY

System function, behavior, structure and service

- Fundamental properties of a system
  - Functionality
  - Performance, dependability, security, cost
    - Called non-functional properties
  - Usability, manageability, adaptability/elasticity
- Structure of a system
  - A set of composite and atomic components
  - A composite component is composed of a set of components
  - Many types of dependencies exist
- A (distributed) system delivers one or many services

Client requirements/expectations

- What would you expect when you send a picture to your friend?

- What would you expect when you search Google?

Clients require correct service w.r.t function and non-functional properties

Non-functional properties about performance, dependability, security and cost can be very subjective

Requirement/expectation from service providers

- Offer the correct functionality
- Avoid service failures, e.g.,
  - To avoid unexpected crashes
  - To able to detect and recover failures
- Improve quality of services, e.g.,
  - Reduce response time and cost, maximize service utilization
- Support „conformity“ and „specific“ requirements

To provide **correct and enhanced service** w.r.t function and non-functional properties

Function versus non-functional failures

Function

Correct service:

- Deliver the intended function described in the service specification

Service failure

- The delivered function deviates from the specified/intended one

Non-functional properties

Correct service

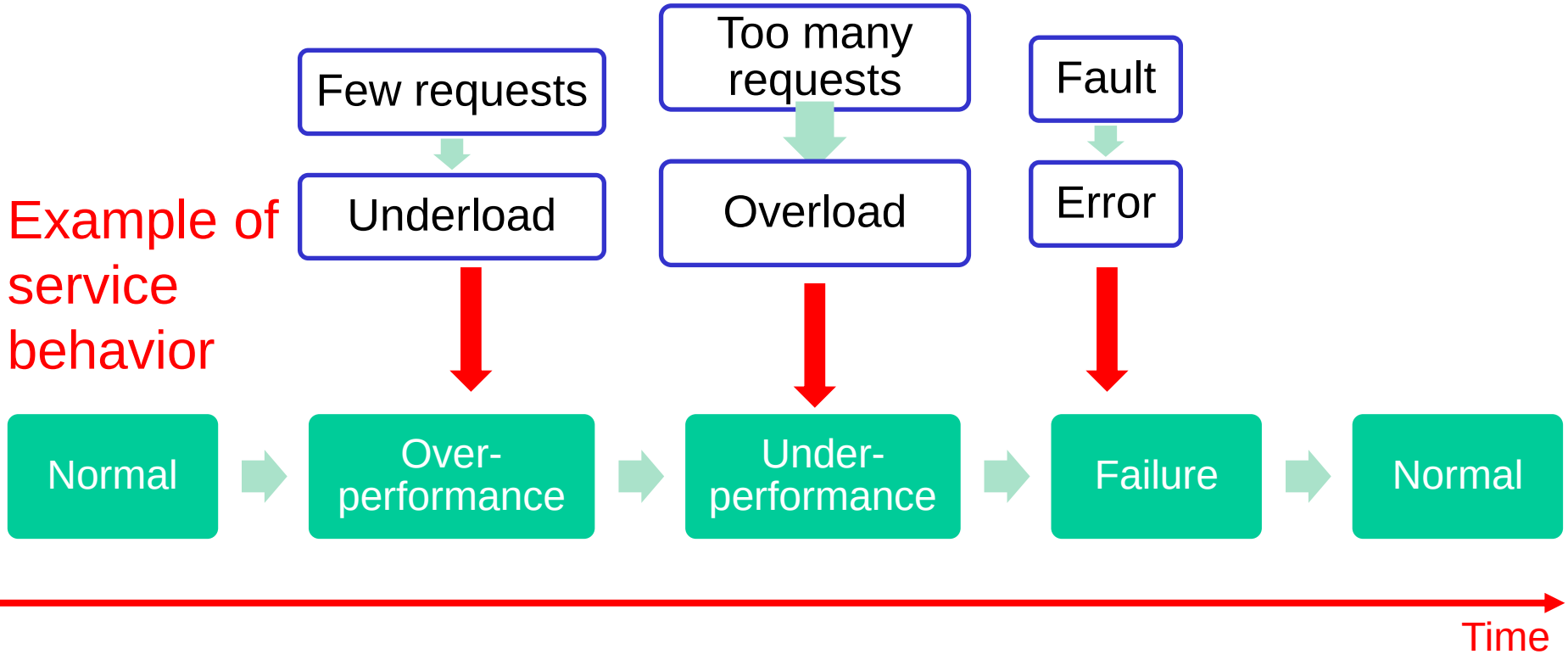
- Deliver the intended function within the specified non-functional properties

Service failure

- Nonfunctional properties do not meet the specified ones

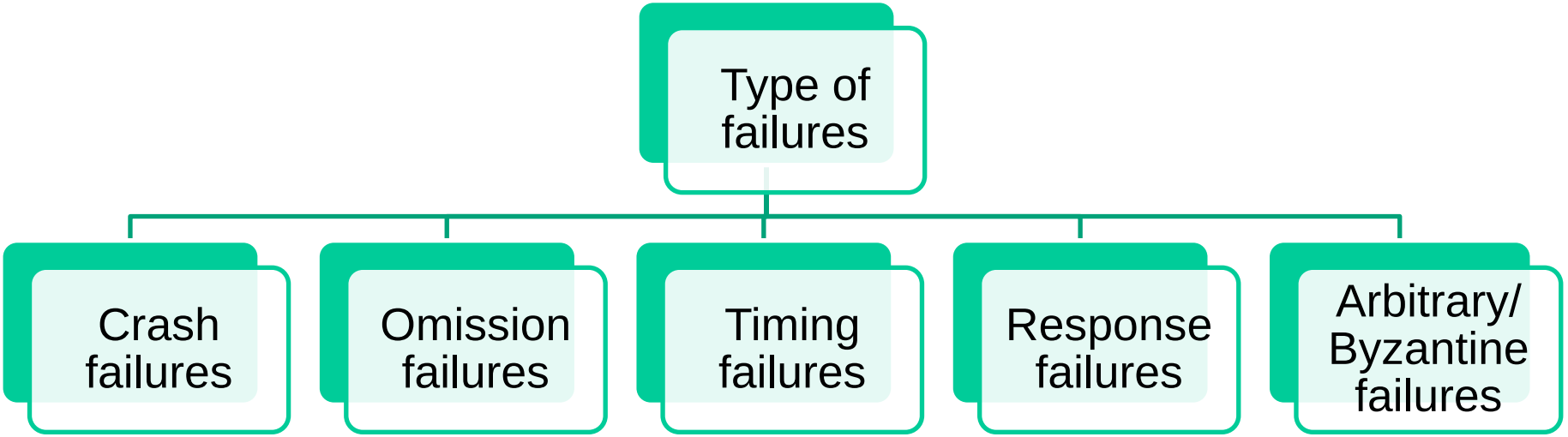
- But failures are inevitable in distributed systems!
- Performance is varying in distributed systems!

System behavior

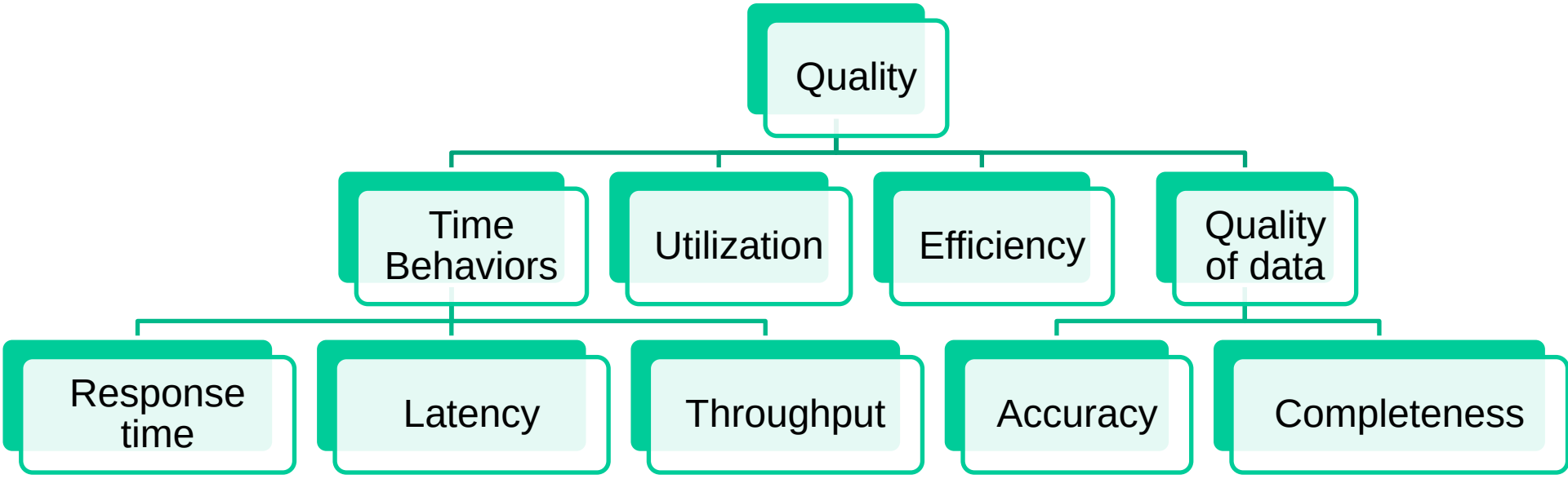


Normal: based on the service specification and design

Failure classification

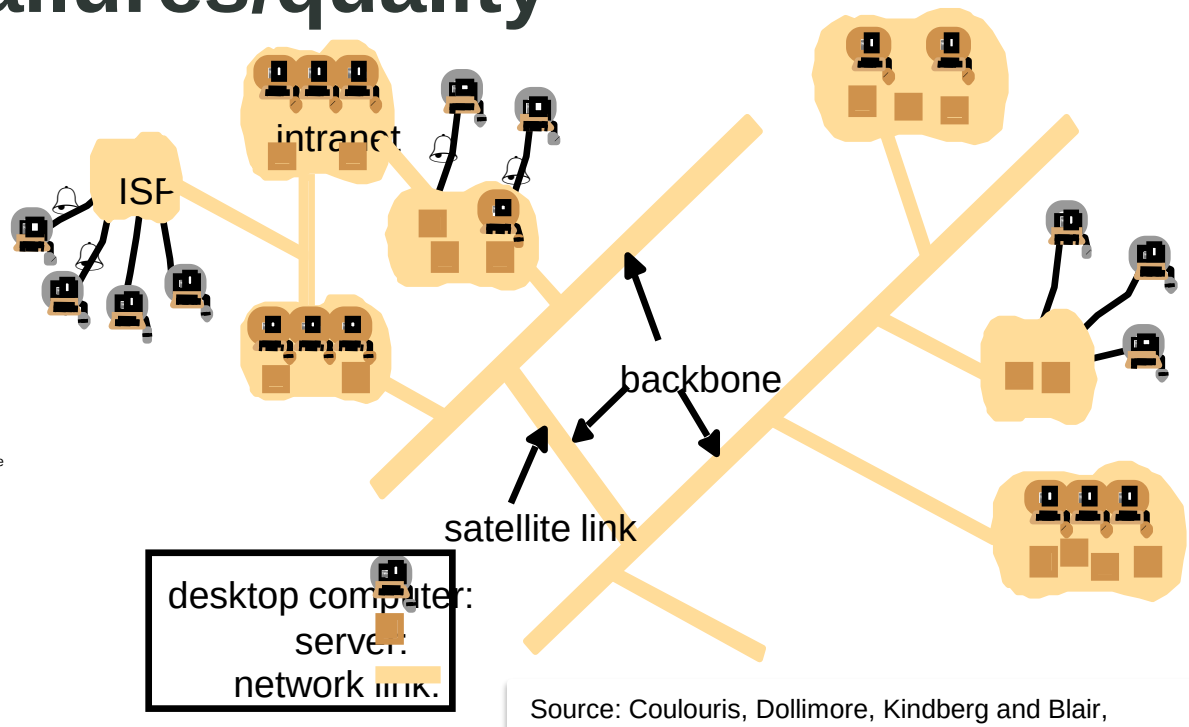


Quality of service improvement

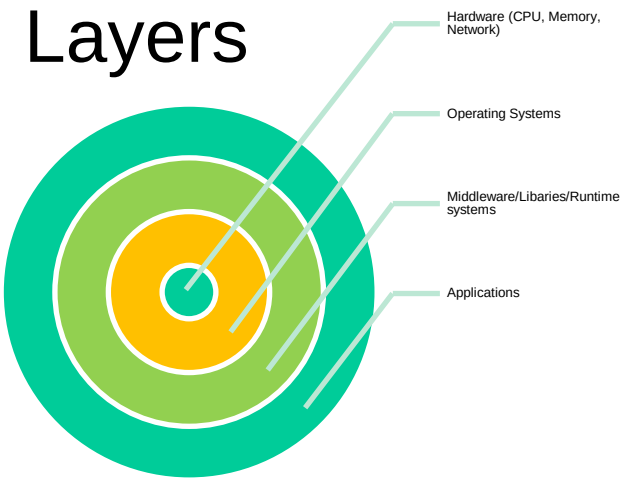


Understand the complexity in dealing with service failures/quality

Scale

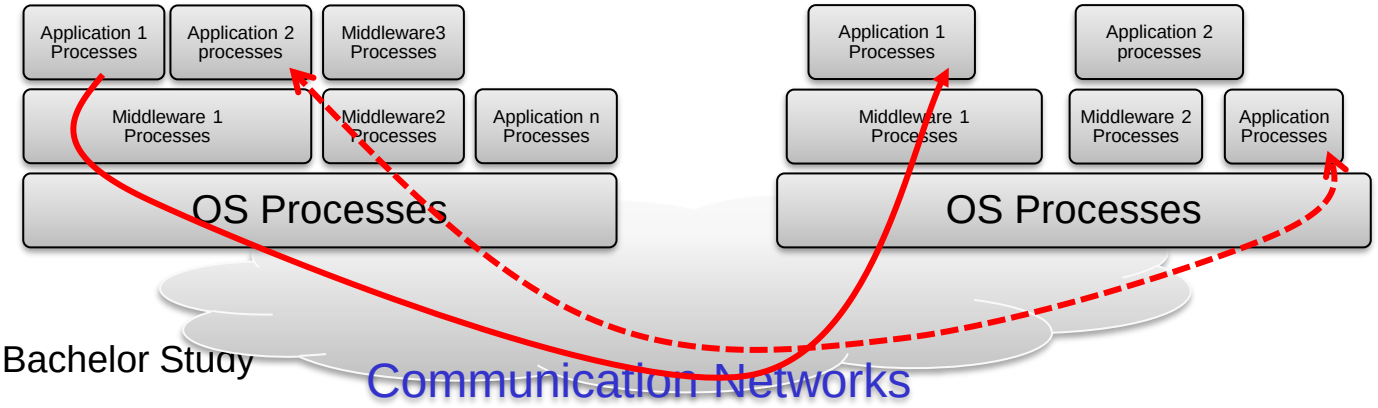


Layers



Source: Coulouris, Dollimore, Kindberg and Blair, Distributed Systems: Concepts and Design Edn. 5

Structure



Dealing with service failures and quality

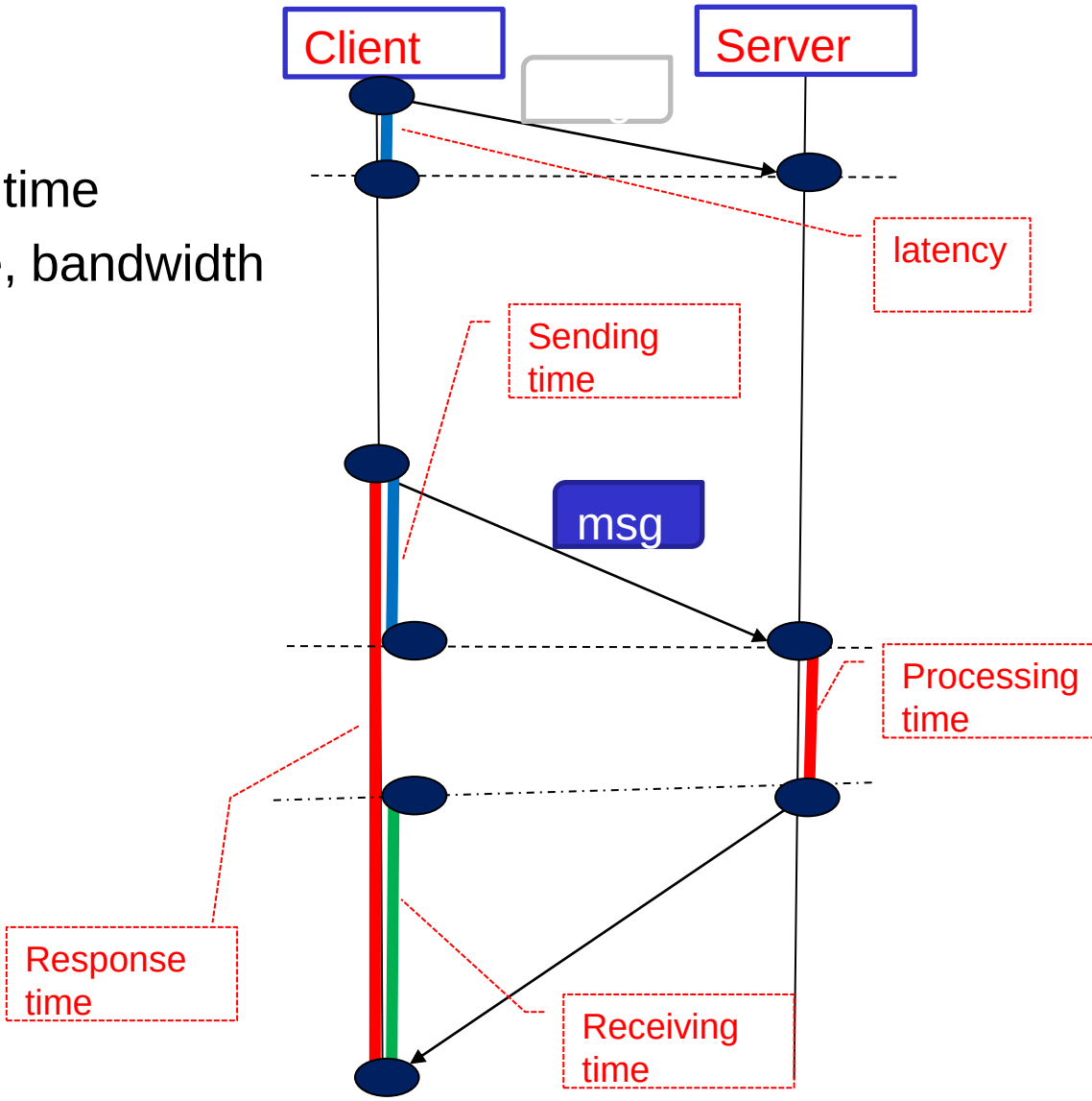
- Determines clearly system boundaries
  - The system under study, the system used to judge, and the environment
- Understands dependencies, e.g.
  - Among components in distributed systems
  - Single layer as well as cross-layered dependencies
- Determines types of metrics and failures and break down problems along the dependency path

PERFORMANCE

Performance metrics

- Timing behaviors
  - Communication
    - Latency/Transfer time
    - Data transfer rate, bandwidth
  - Processing
    - Response time
    - Throughput
- Utilization
  - Network utilization
  - CPU utilization
  - Service utilization
- Efficiency
- Data quality

Examples

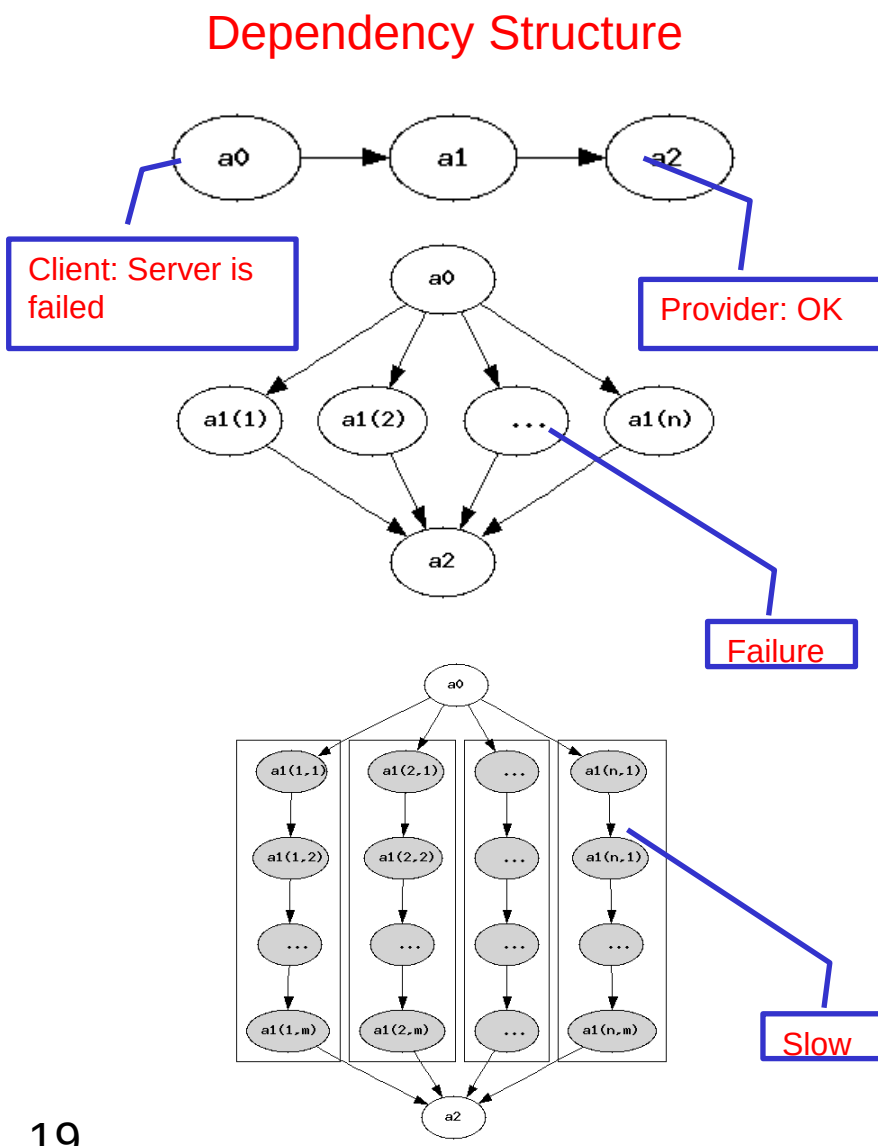


Measurement, Monitoring and Analysis

- Instrumentation and Sampling
  - Instrumentation: insert probes into systems so that you can measure system behaviors directly
  - Sampling: use components to take samples of system behaviors
- Monitoring
  - Probes or components perform sampling or measurements, storing and sharing measurements
- Analysis
  - Evaluate and interpret measurements for specific contexts
  - Can be subjective!

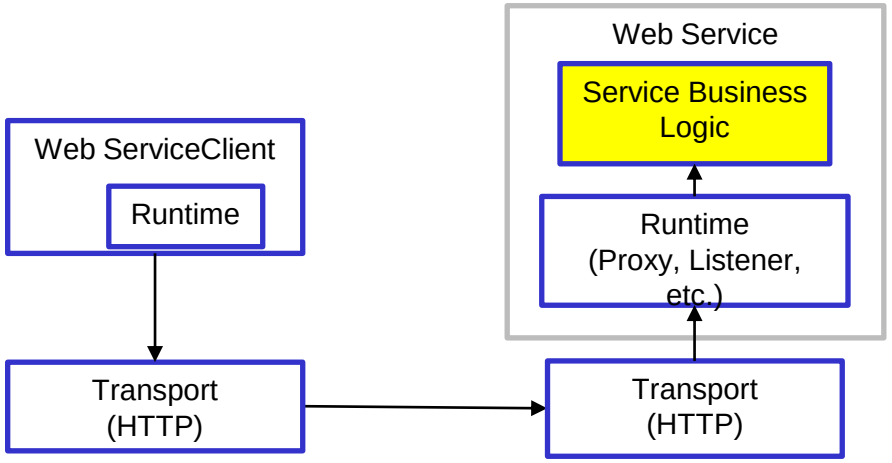
Composable methods and views

- Composable method
  - Divide a complex structure into basic common structures
  - Each basic structure has different ways to analyze specific failures/metrics
- Interpretation based on context/view
  - Client view or service provider view?
  - Conformity versus specific requirement assessment



Examples

- Which performance metrics can be measured?
- How can you measure these metrics?



DEPENDABILITY

Dependability

„The dependability of a system is the ability to avoid service failures that are more frequent and more severe than is acceptable“

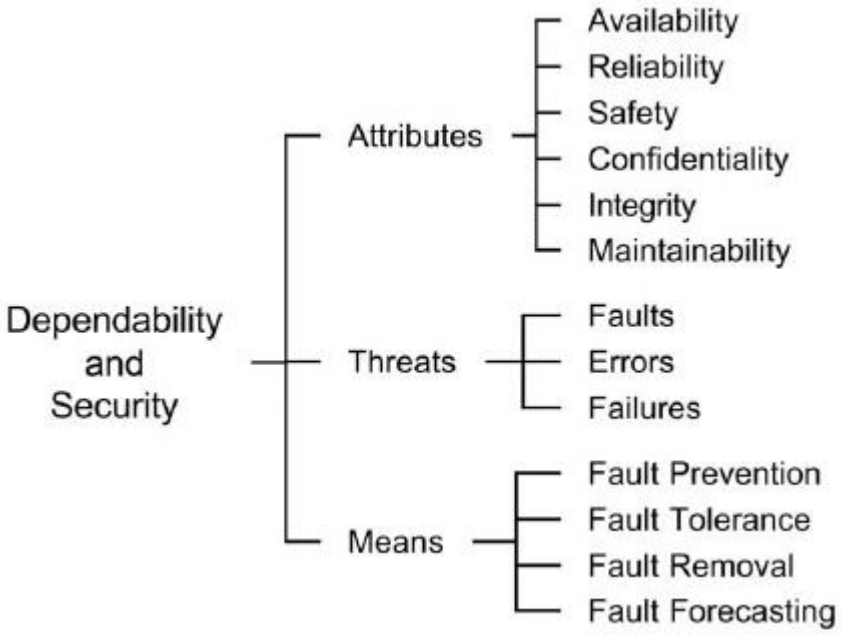
- Important characteristics
  - About avoiding service failures
  - Subjective
  - Defined in a specific context
  - Defined as an average

Source: John Knight, Fundamentals of Dependable Computing for Software Engineers, CRC Press, 2012

Performability

- What happens if the performance is unacceptable, e.g., the service cannot be scaled, the service is unreliable
- Technically, the system may still deliver its function
  - it may fail to deliver the expected non-functional properties as well as its function may fail eventually
- Performability measures a system performance and its dependability
  - Performance is currently not an attribute of dependability

Dependability Attributes, Threats and Means



See lecture 9

Sub mechanisms of resilience techniques

Algirdas Avizienis, Jean-Claude Laprie, Brian Randell, and Carl Landwehr. 2004. Basic Concepts and Taxonomy of Dependable and Secure Computing. IEEE Trans. Dependable Secur. Comput. 1, 1 (January 2004), 11-33.

Personal note: Performance should be an attribute as well!

Dependability attributes (1)

Availability: „probability that the system will operational at time t“ → readiness at a given time

John Knight, Fundamentals of Dependable Computing for Software Engineers, CRC Press, 2012
Algirdas Avizienis, Jean-Claude Laprie, Brian Randell, and Carl Landwehr. 2004. Basic Concepts and Taxonomy of Dependable and Secure Computing. IEEE Trans. Dependable Secur. Comput. 1, 1 (January 2004), 11-33.

- It would be more easy to understand availability by looking at „downtime“. One simple way is

$$Availability = \frac{Uptime}{Uptime + Downtime}$$

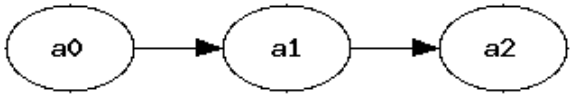
Availability	Downtime (in a year)
90% (1-nine)	36.5 days
99% (2-nines)	3.65 days
99.99 %(4-nines)	52 minutes, 33.6 seconds
99.999% (5-nines)	5 minutes, 15.5 seconds

Dependability attributes (2)

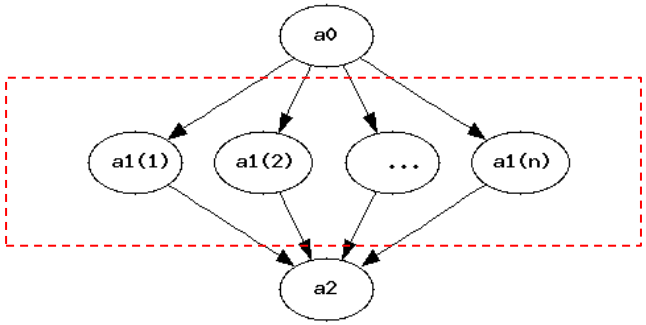
Reliability: „probability that the system will operate correctly in a specified operating environment up until time t“ → continuity without failures

John Knight, Fundamentals of Dependable Computing for Software Engineers, CRC Press, 2012
Algirdas Avizienis, Jean-Claude Laprie, Brian Randell, and Carl Landwehr. 2004. Basic Concepts and Taxonomy of Dependable and Secure Computing. IEEE Trans. Dependable Secur. Comput. 1, 1 (January 2004), 11-33.

- Some simple rules R_i is the probability of successful operations



$$Reliability = \prod_{i=1}^{n} R_i$$



Q_i is the probability of failure operations

$$Reliability = 1 - \prod_{i=1}^{n} Q_i$$

See http://www.csun.edu/~bjc20362/Billinton-Allan-Excerpt.pdf

Dependability attributes (3)

Risk: „expected loss per unit time that will be experienced by using a system“

- Loss: money, life, etc.

$$risk = \sum_i \mathrm{pr}(failure_i) \text{ x } \mathrm{loss}(failure_i)$$

Safety: „expected loss per unit time is less than a prescribed threshold“ → absence of catastrophic consequences

Source: John Knight, Fundamentals of Dependable Computing for Software Engineers, CRC Press, 2012

Dependability attributes (4)

Confidentiality: „the absence of unauthorized disclosure of information“

Integrity: „the absence of improper system alterations“

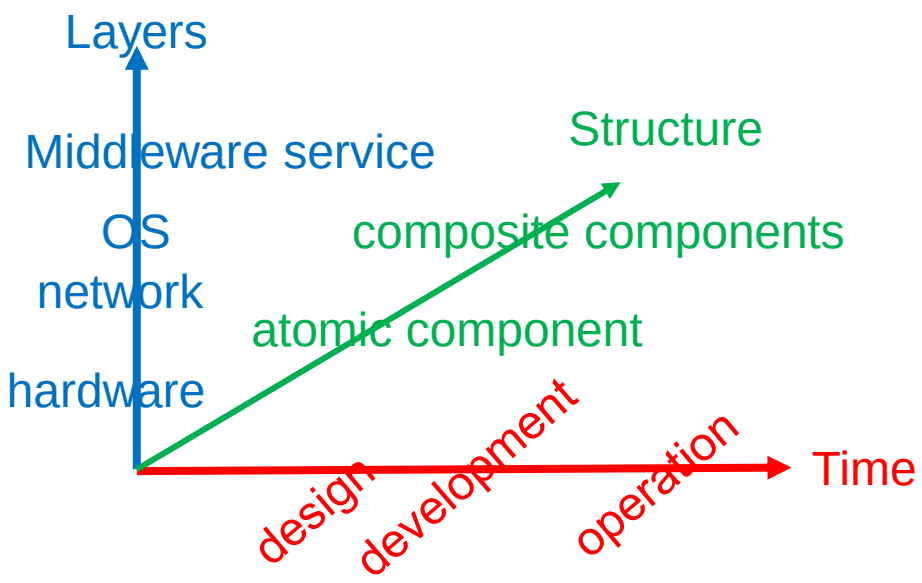
Maintainability: „the ability to undergo repairs and modifications“

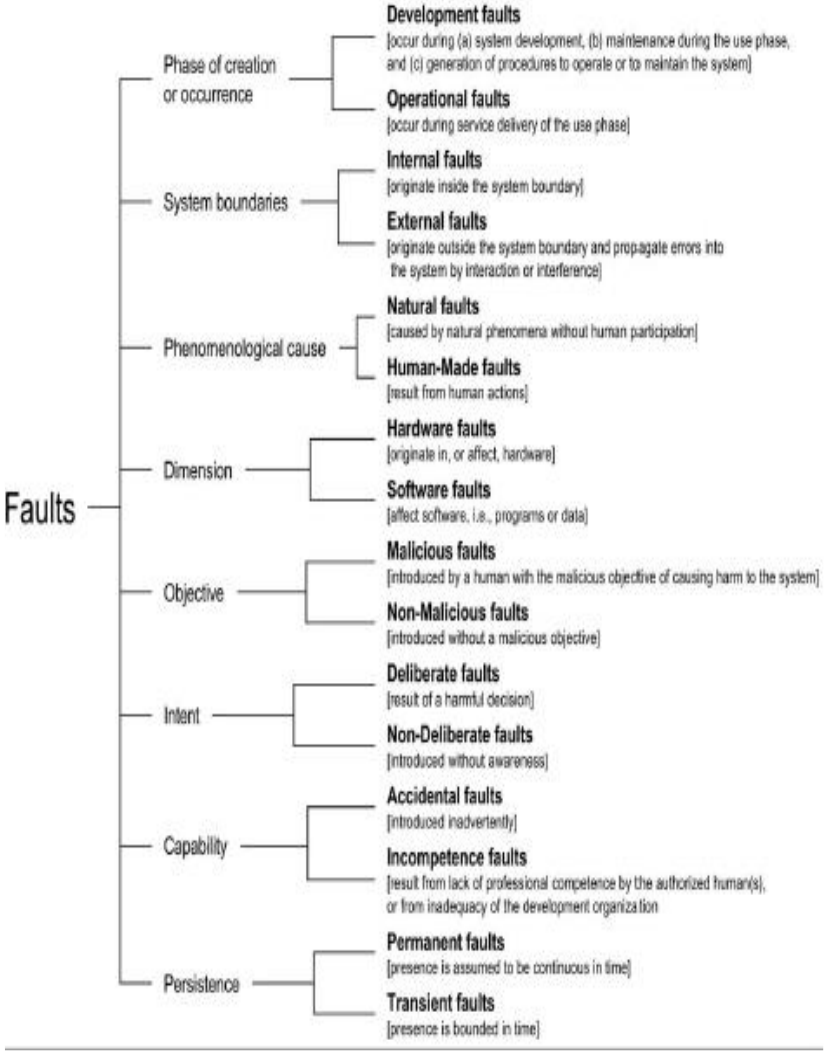
John Knight, Fundamentals of Dependable Computing for Software Engineers, CRC Press, 2012
Algirdas Avizienis, Jean-Claude Laprie, Brian Randell, and Carl Landwehr. 2004. Basic Concepts and Taxonomy of Dependable and Secure Computing. IEEE Trans. Dependable Secur. Comput. 1, 1 (January 2004), 11-33.

Threats to Dependability

- Error: Deviation of the actual system state from the correct service state
- Fault: the (actual or hypothesize) cause of an error
- Failure: an event when the delivered service deviates from correct service
  - Not comply with the functional specification
  - Often also not comply with the non-functional specification

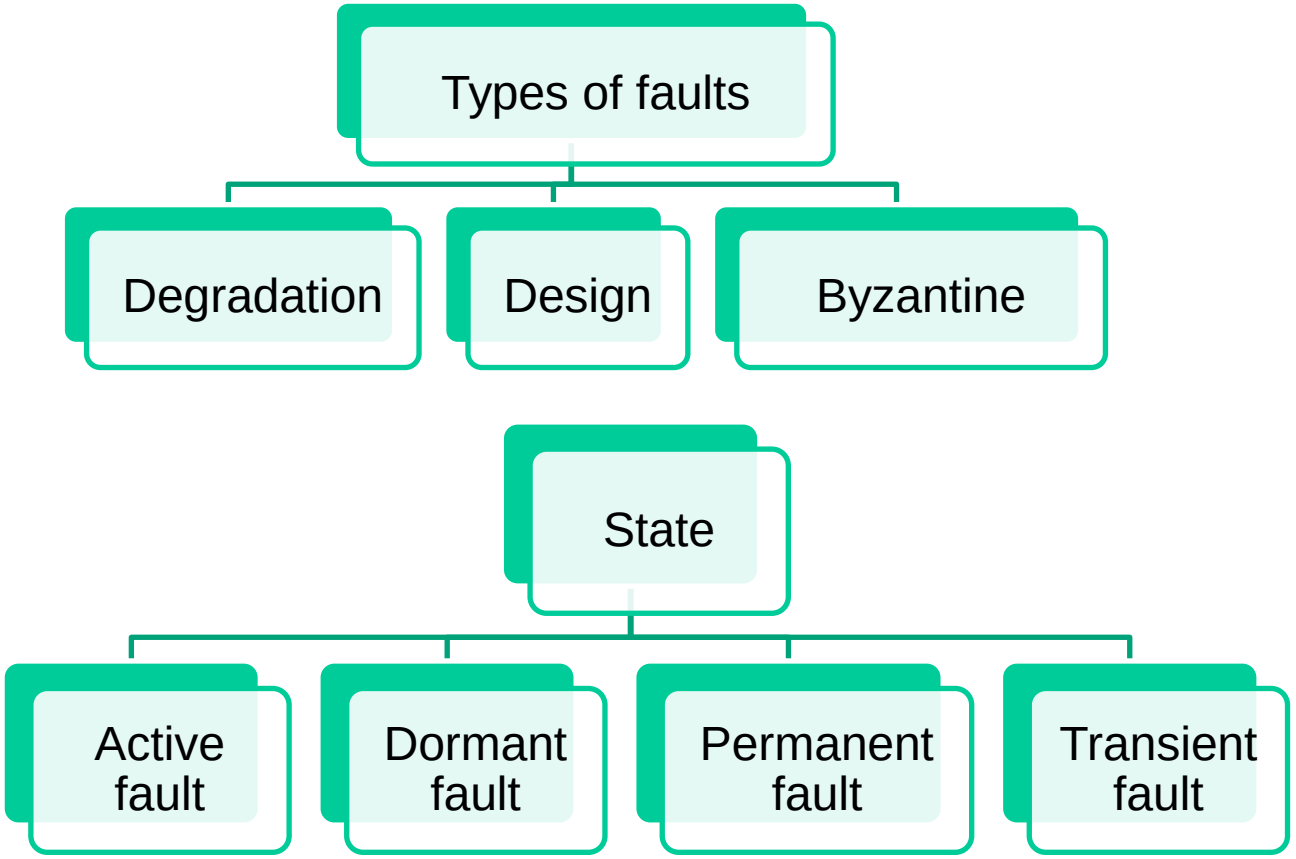
Types of faults (1)





Algirdas Avizienis, Jean-Claude Laprie, Brian Randell, and Carl Landwehr. 2004. Basic Concepts and Taxonomy of Dependable and Secure Computing. IEEE Trans. Dependable Secur. Comput. 1, 1 (January 2004), 11-33

Types of faults (2)



Examples of Failures, Errors, Faults

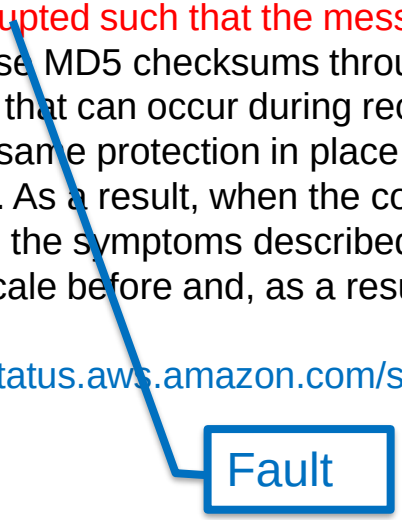
“ …. On Sunday, we saw a large number of servers that were spending almost all of their time gossiping and a disproportionate amount of servers that had failed while gossiping. With a large number of servers gossiping and failing while gossiping, Amazon S3 wasn't able to successfully process many customer requests.

…..

At 10:32am PDT, after exploring several options, we determined that we needed to shut down all communication between Amazon S3 servers, shut down all components used for request processing, clear the system's state, and

We've now determined that message corruption was the cause of the server-to-server communication problems. More specifically, we found that there were a handful of messages on Sunday morning that had a single bit corrupted such that the message was still intelligible, but the system state information was incorrect. We use MD5 checksums throughout the system, for example, to prevent, detect, and recover from corruption that can occur during receipt, storage, and retrieval of customers' objects. However, we didn't have the same protection in place to detect whether this particular internal state information had been corrupted. As a result, when the corruption occurred, we didn't detect it and it spread throughout the system causing the symptoms described above. We hadn't encountered server-to-server communication issues of this scale before and, as a result, it took some time during the event to diagnose and recover from it.”

Source: http://status.aws.amazon.com/s3-20080720.htm

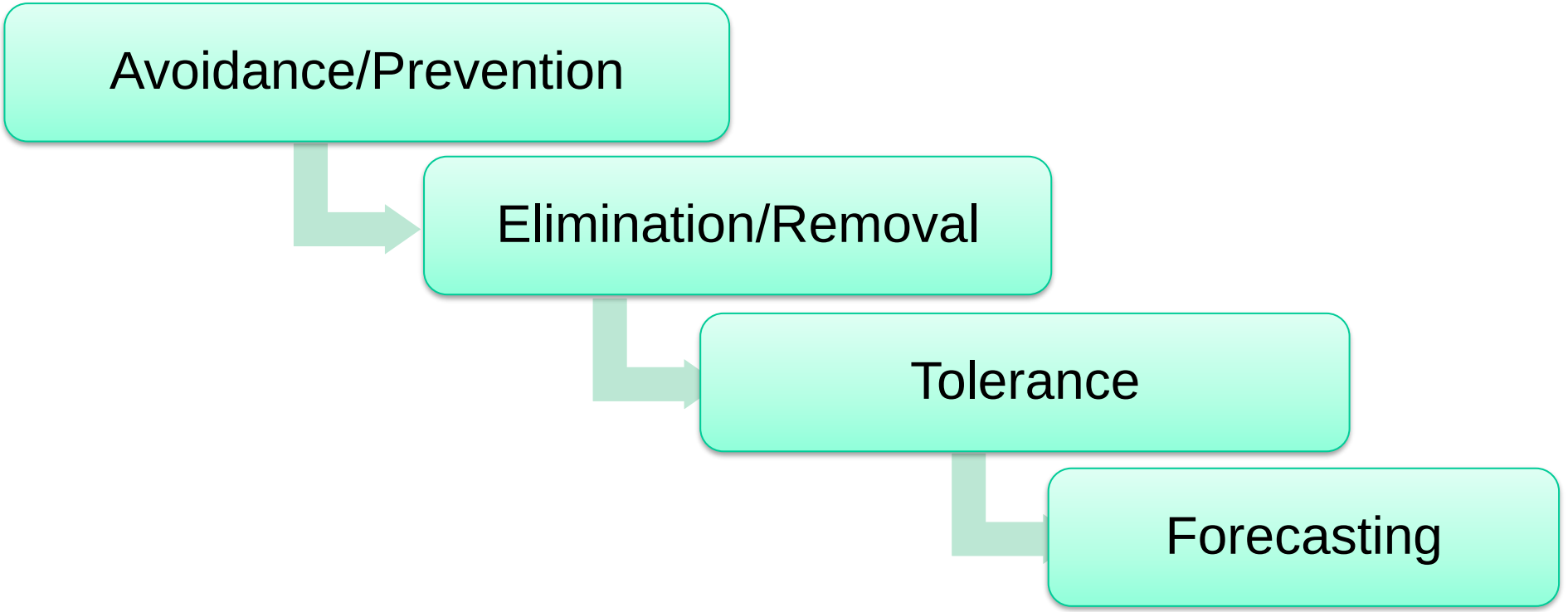


Failure models

Type of failure	Description
Crash failure	A server halts, but is working correctly until it halts
Omission failure *Receive omission* *Send omission*	A server fails to respond to incoming requests A server fails to receive incoming messages A server fails to send messages
Timing failure	A server's response lies outside the specified time interval
Response failure *Value failure* *State transition failure*	A server's response is incorrect The value of the response is wrong The server deviates from the correct flow of control
Arbitrary failure	A server may produce arbitrary responses at arbitrary times

Source: Andrew S. Tanenbaum and Maarten van Steen, Distributed Systems – Principles and Paradigms, 2nd Edition, 2007, Prentice-Hall

Means/Mechanisms for Dependability



DEALING WITH FAULTS

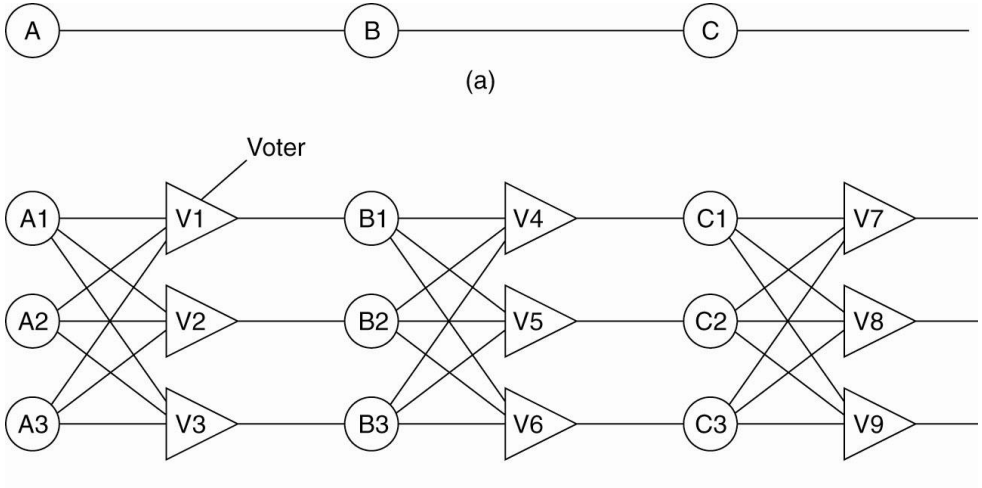
Dealing with faults

- Resilience and Elasticity → able to go back/to stretch
  - **Redundancy** and Replication
  - **Fault-tolerance**
    - **including checkpointing and recovery**
  - Elasticity (Elastic Computing)
    - Mainly for quality perspectives
  - Feedback Control (e.g., in Autonomic Computing)

Redundancy

- Information Redundancy: additional information is provided.
- Time Redundancy: actions are performed again
- Physical Redundancy: extra hardware or software components are used to tolerate the failures of some hardware/components

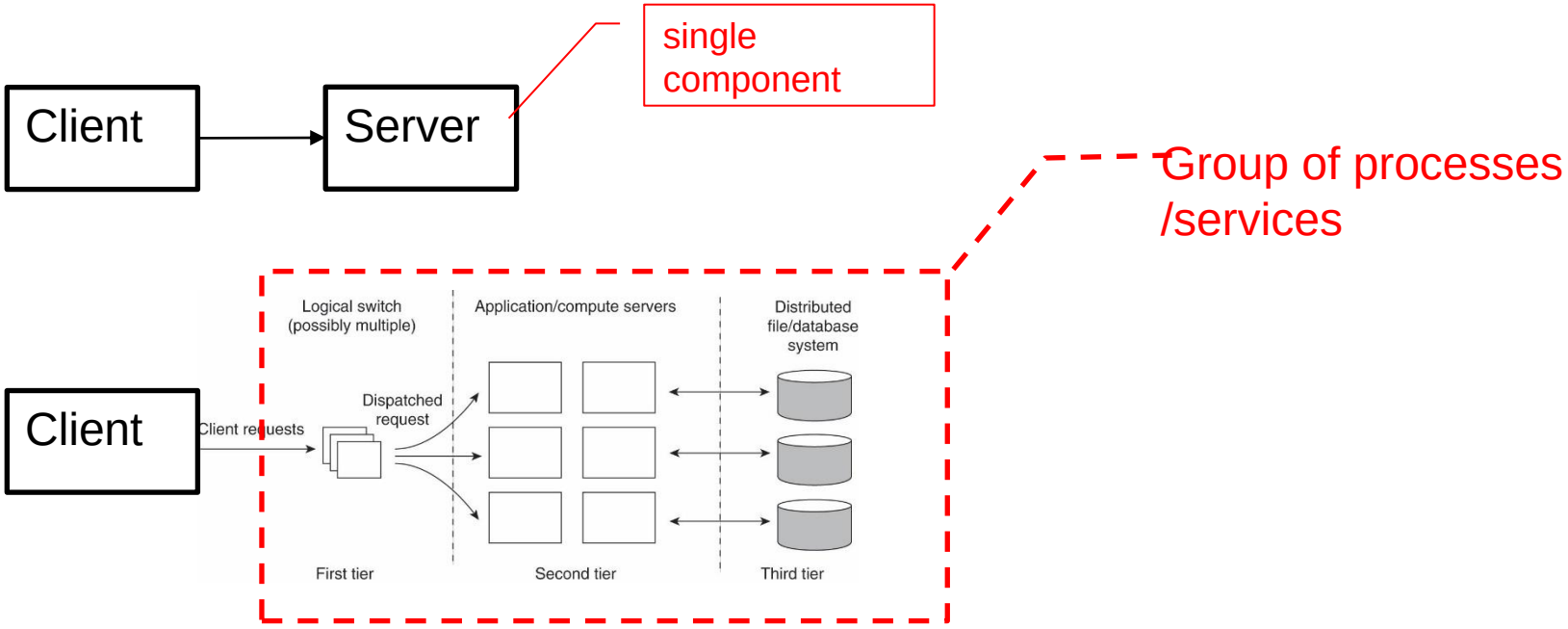
Example of Triple Modular Redundancy



Source: Andrew S. Tanenbaum and Maarten van Steen, Distributed Systems – Principles and Paradigms, 2nd Edition, 2007, Prentice-Hall

Group redundancy architecture

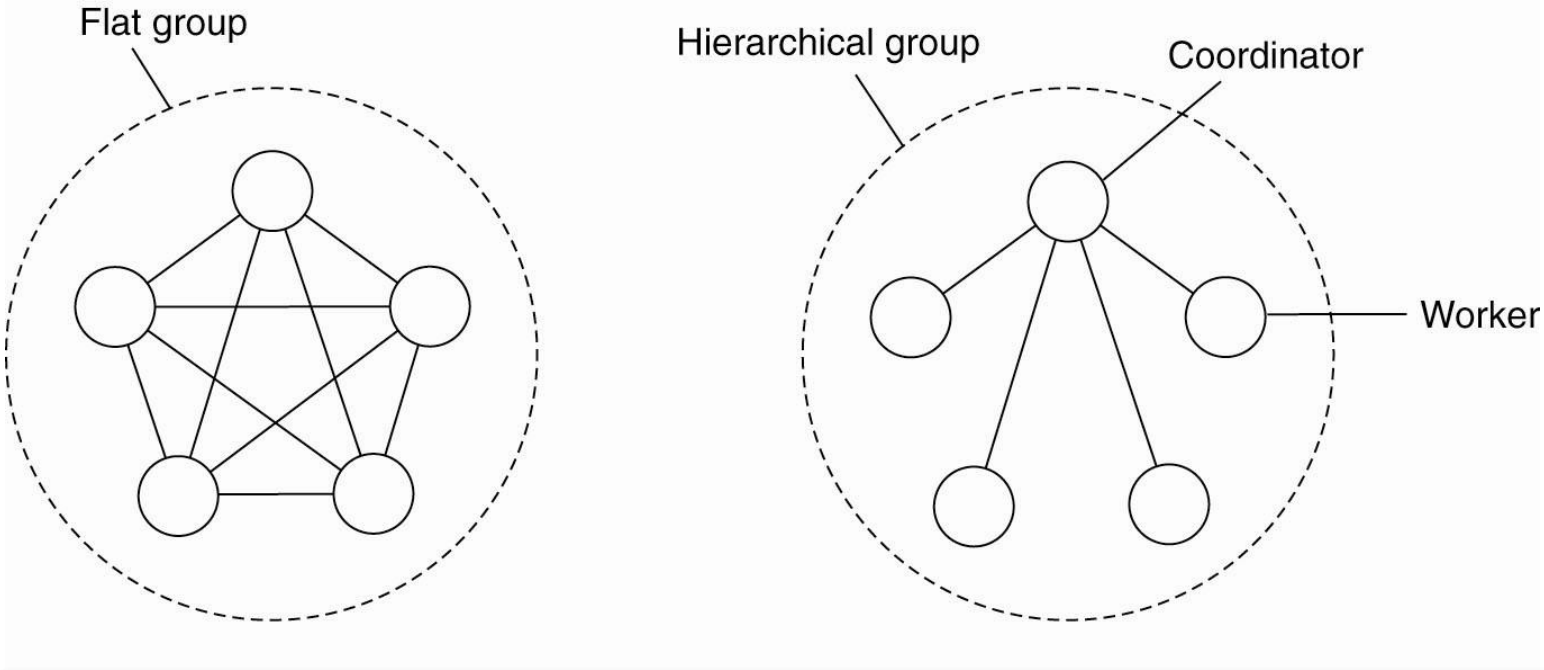
- Use group architecture for redundancy in order to support failure masking



Source: Andrew S. Tanenbaum and Maarten van Steen, Distributed Systems – Principles and Paradigms, 2nd Edition, 2007, Prentice-Hall

Design Flat Groups versus Hierarchical Groups

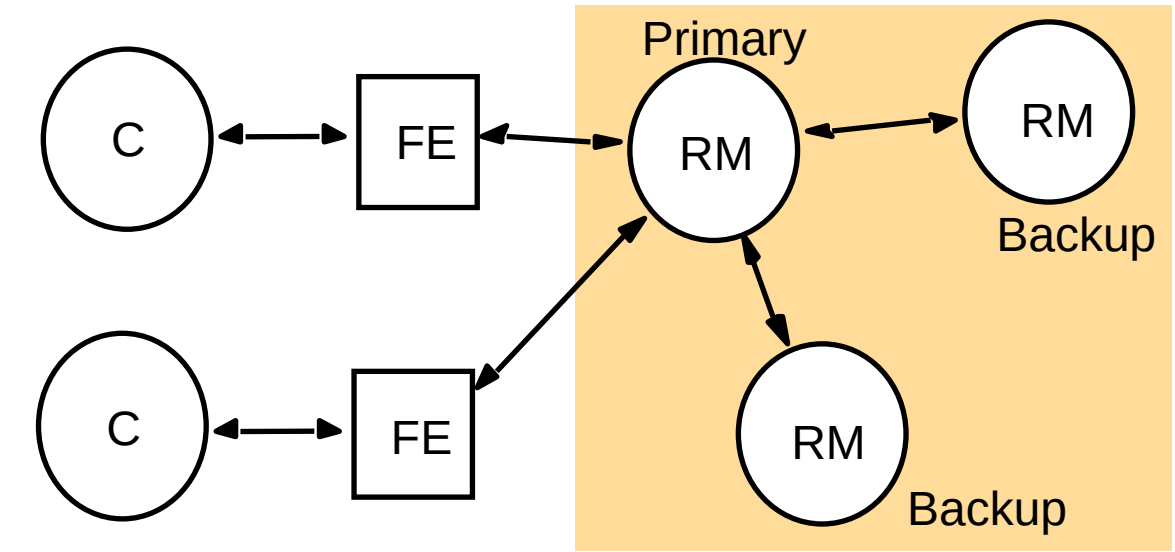
Structure a system (communication, servers, services, etc.) using a group so we can deal failures using collective capabilities



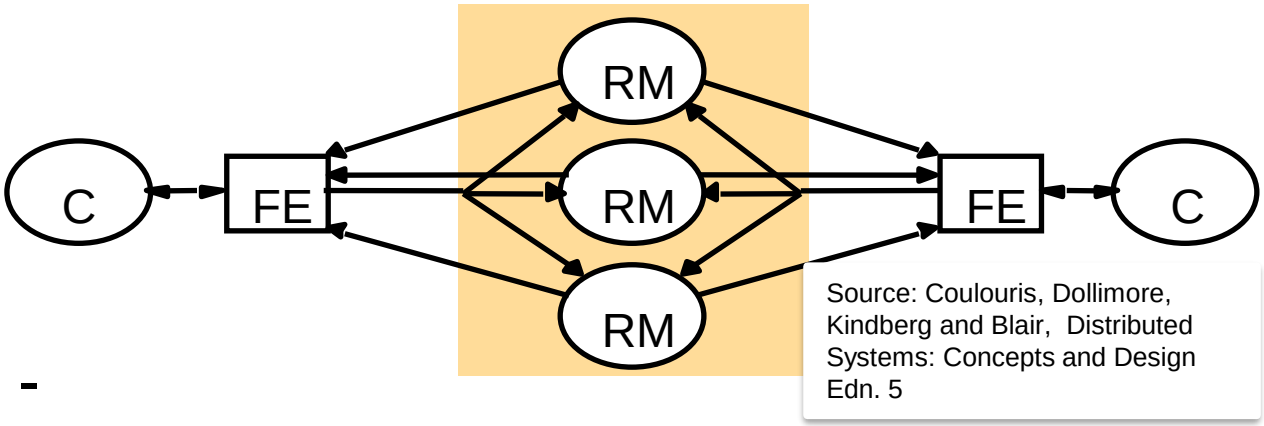
Source: Andrew S. Tanenbaum and Maarten van Steen, Distributed Systems – Principles and Paradigms, 2nd Edition, 2007, Prentice-Hall

Replication architecture

Passive (Primary backup) model



Active Replication



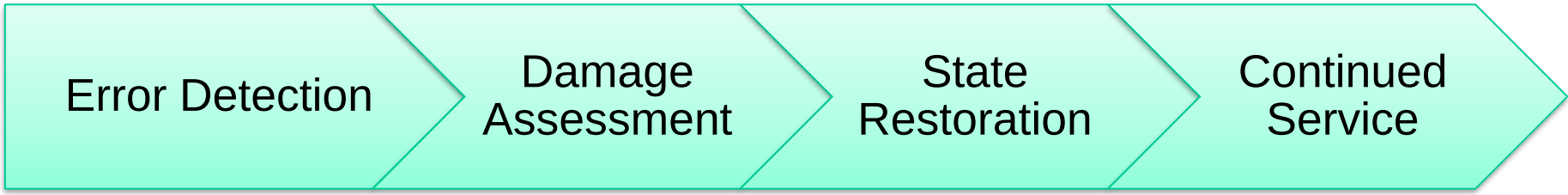
Source: Coulouris, Dollimore, Kindberg and Blair, Distributed Systems: Concepts and Design Edn. 5

Example: Cassandra - http://www.datastax.com/docs/1.0/cluster_architecture/replication

Fault-tolerance computing

Fault tolerance means to avoid service failures in the presence of faults

Main steps:



Failure detection

- Monitoring and analysis
  - Performance monitoring and analysis tools
    - Pinging, Gossip
  - Process + network monitoring
- Testing
  - Fault injection
- Evaluation
  - Message analysis, Data quality check, Auditing

Thanks for your attention