Performance, Dependability, and Fault Tolerance in Distributed Systems

Hong-Linh Truong
Distributed Systems Group,
Vienna University of Technology

truong@dsg.tuwien.ac.at
dsg.tuwien.ac.at/staff/truong

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 reading:
  - 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
- Homework
- Summary

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
- 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

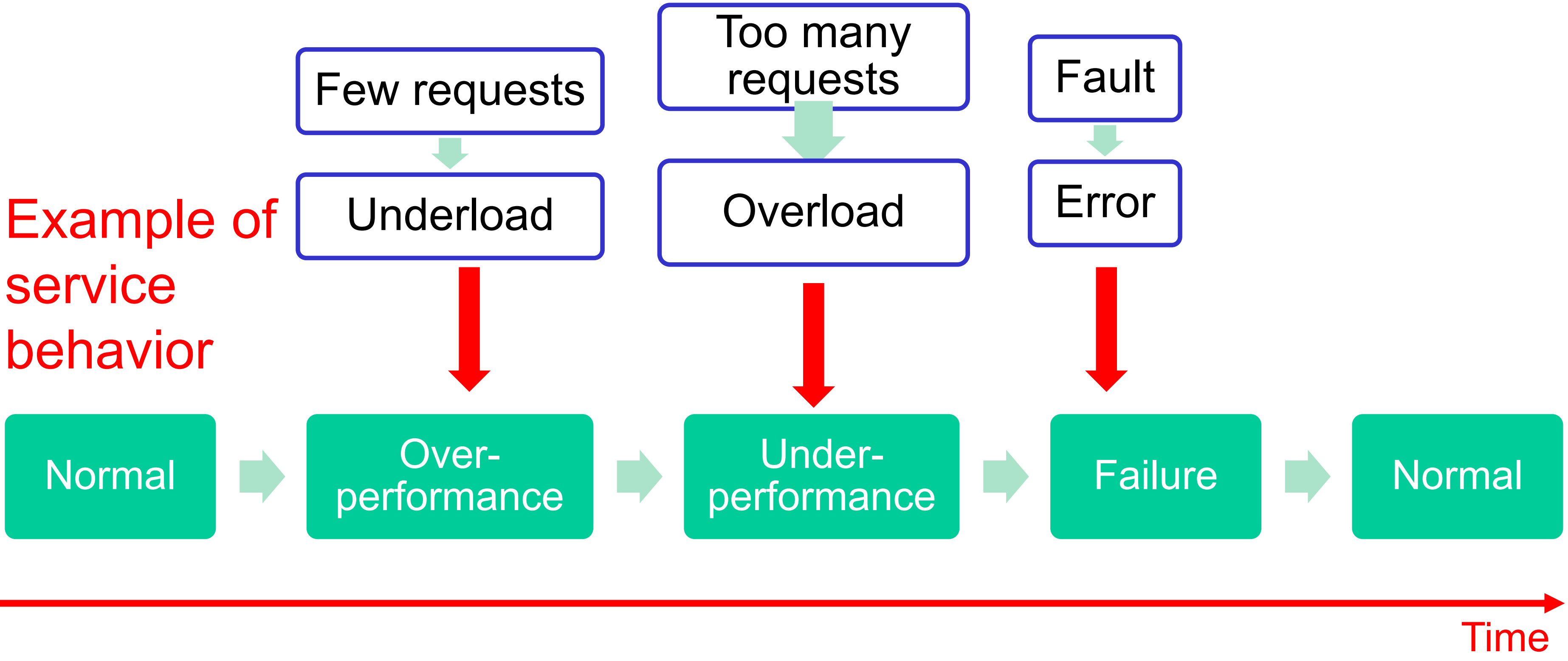
- Deliver the intended function within the specified non-functional properties

Service failure

- Non functional properties do not meet the specified ones

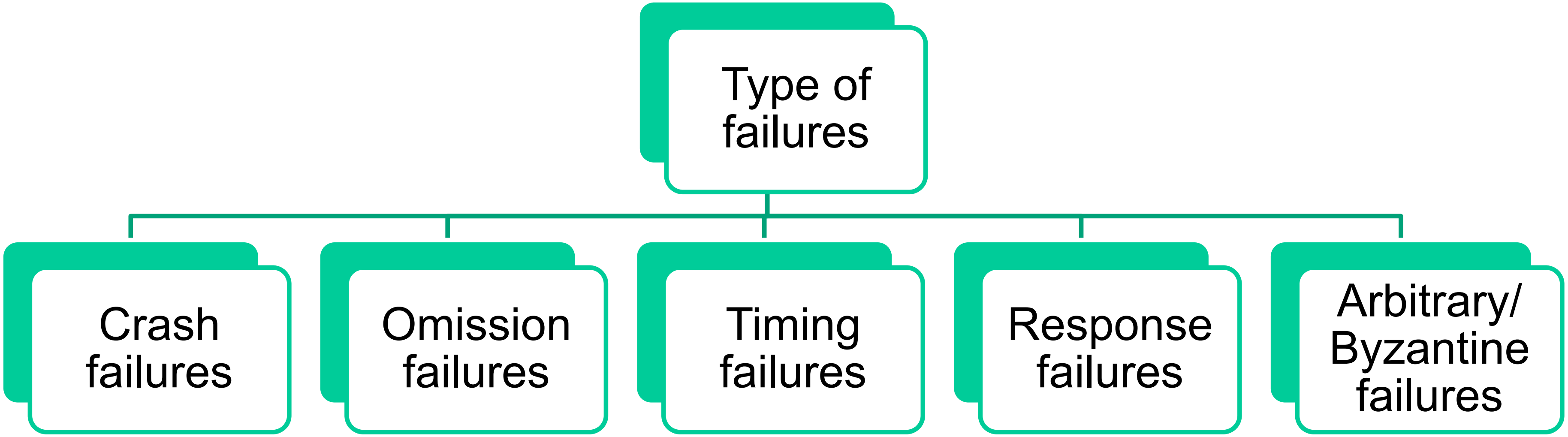
- But failures are inevitable in distributed systems!
- Performance is varying in distributed systems!

System behavior

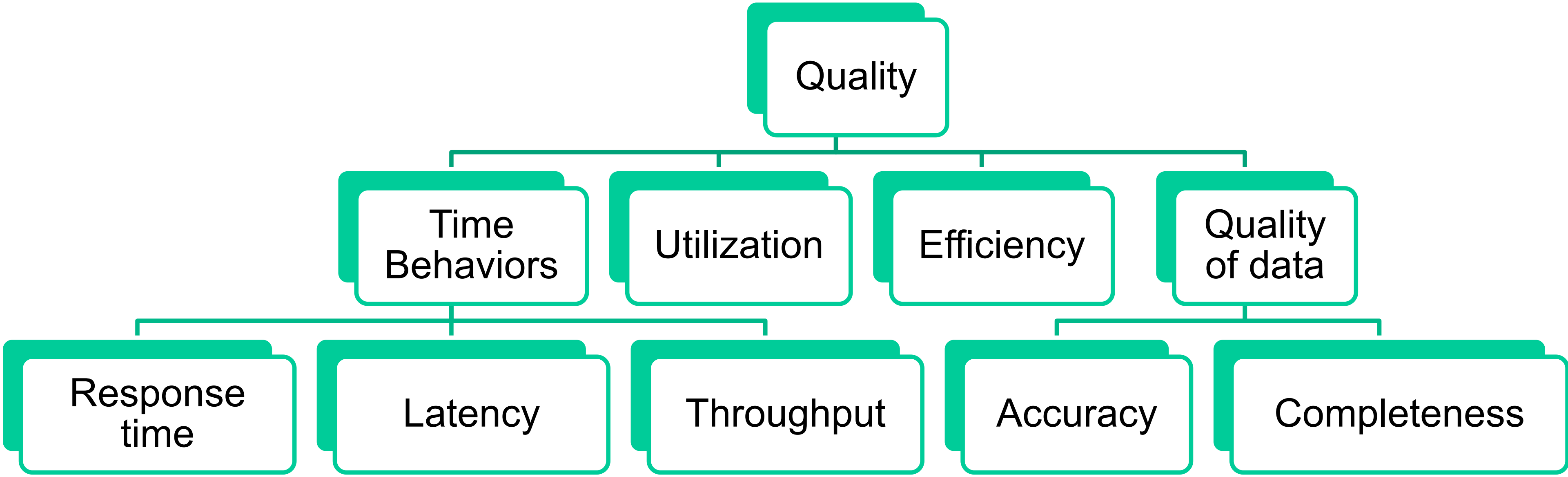


Normal: based on the service specification and design

Failure classification



Quality of service improvement


Quality

Time Behaviors

Utilization

Efficiency

Quality of data

Response time

Latency

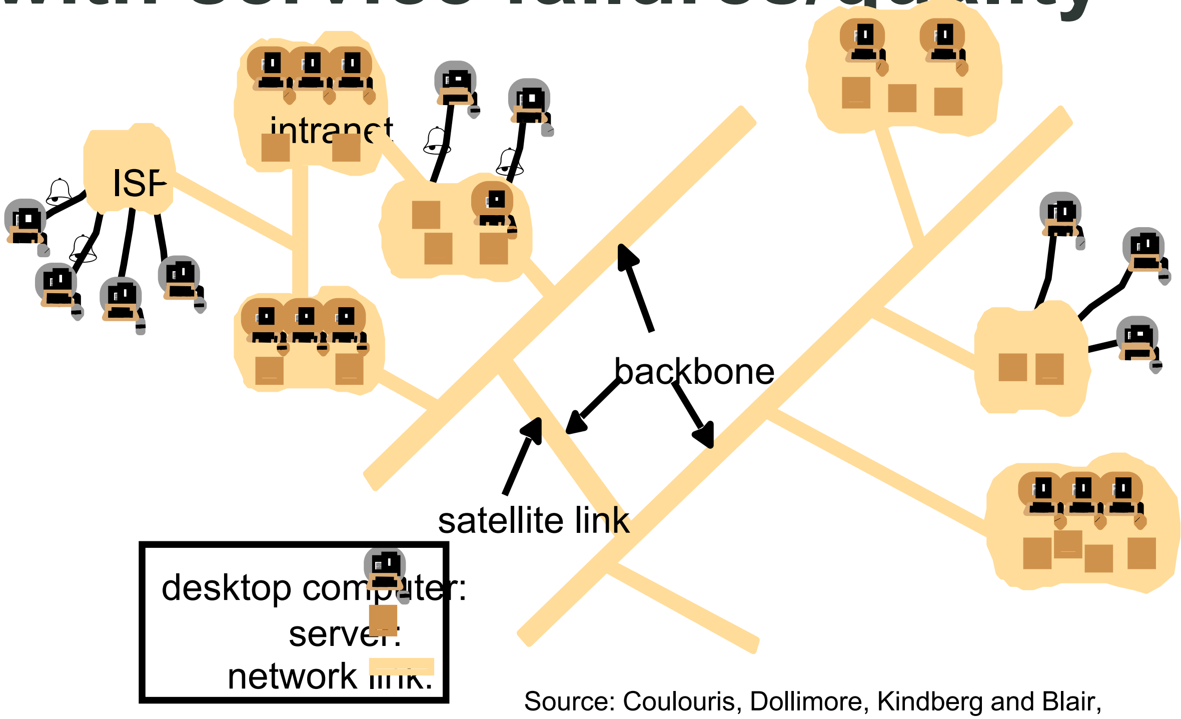
Throughput

Accuracy

Completeness

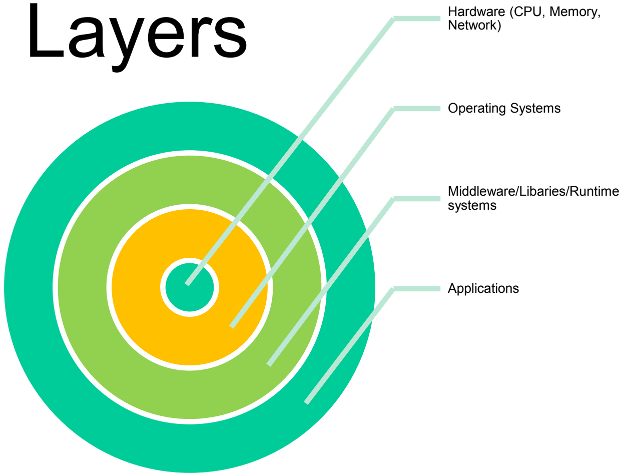
Understand the complexity in dealing with service failures/quality

Scale

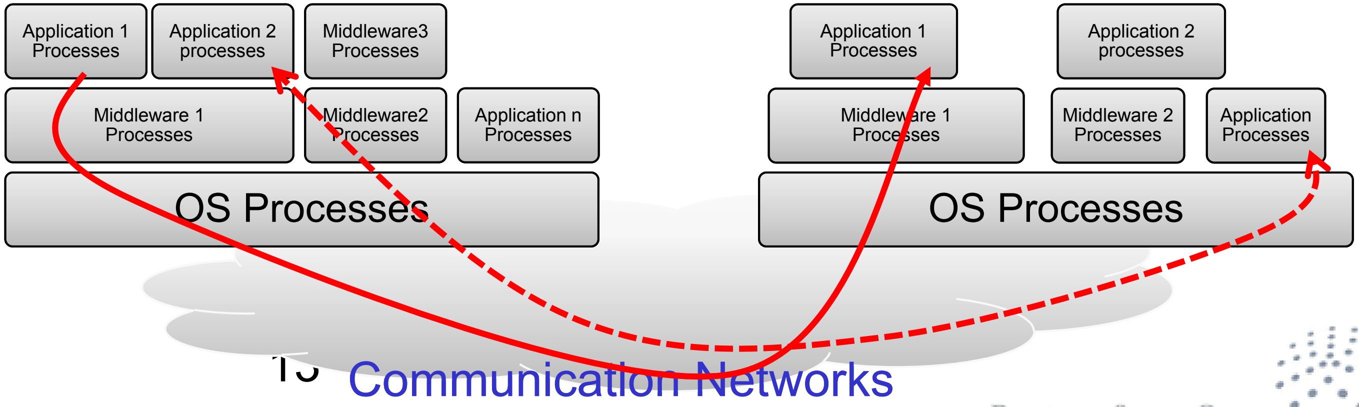


Source: Coulouris, Dollimore, Kindberg and Blair, Distributed Systems: Concepts and Design Edn. 5

Layers


Hardware (CPU, Memory, Network)
Operating Systems
Middleware/Libaries/Runtime systems
Applications

Structure



Dealing with service failures and quality

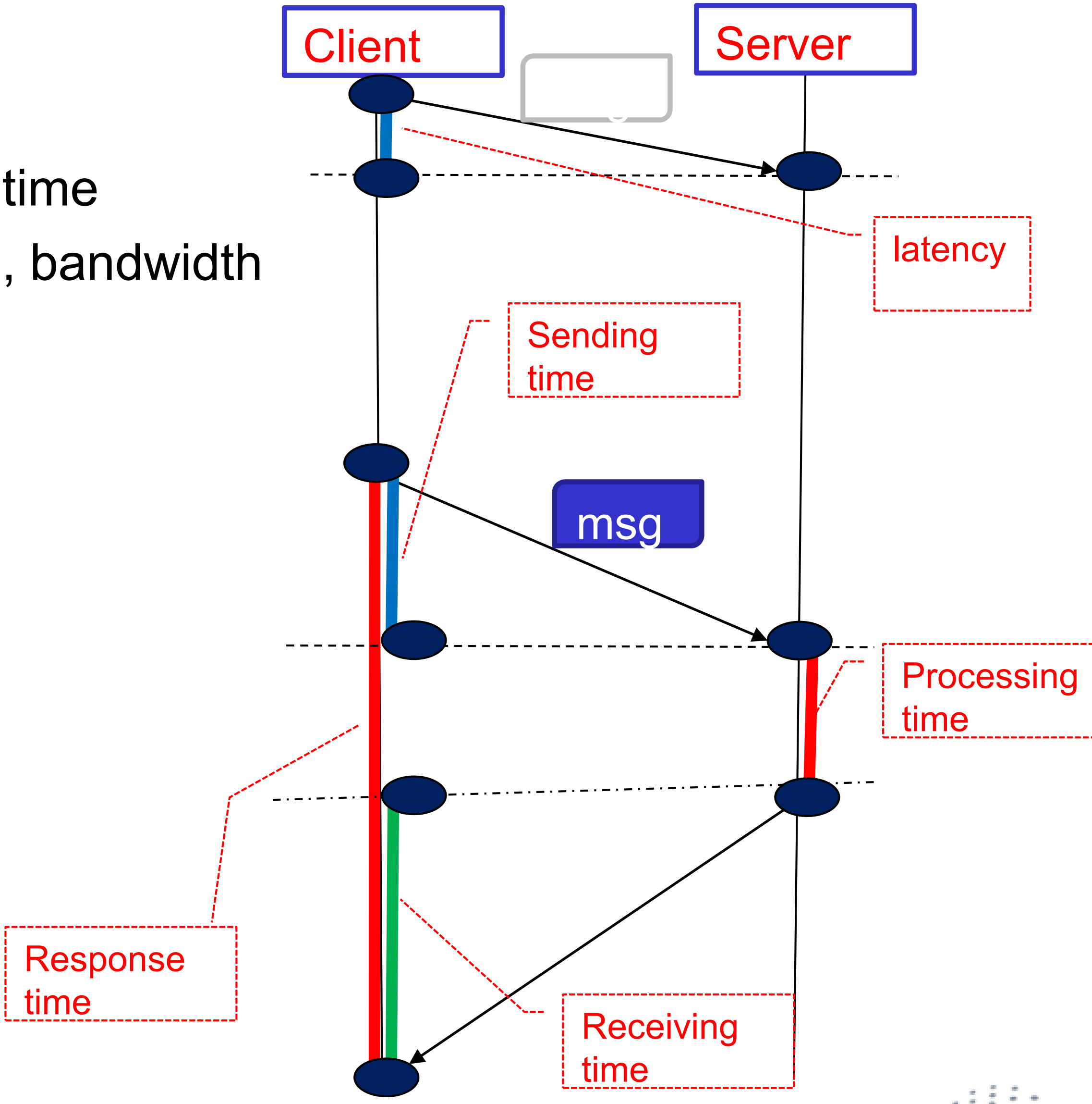
- Determines clearly system boundaries
  - The system under study, the system used to judge, and the environment
- Understands dependencies, e.g.
  - Among components in distributed systems
  - Single layer as well as cross-layered dependencies
- Determines types of metrics and failures and break down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

PERFORMANCE

Performance metrics

Examples

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



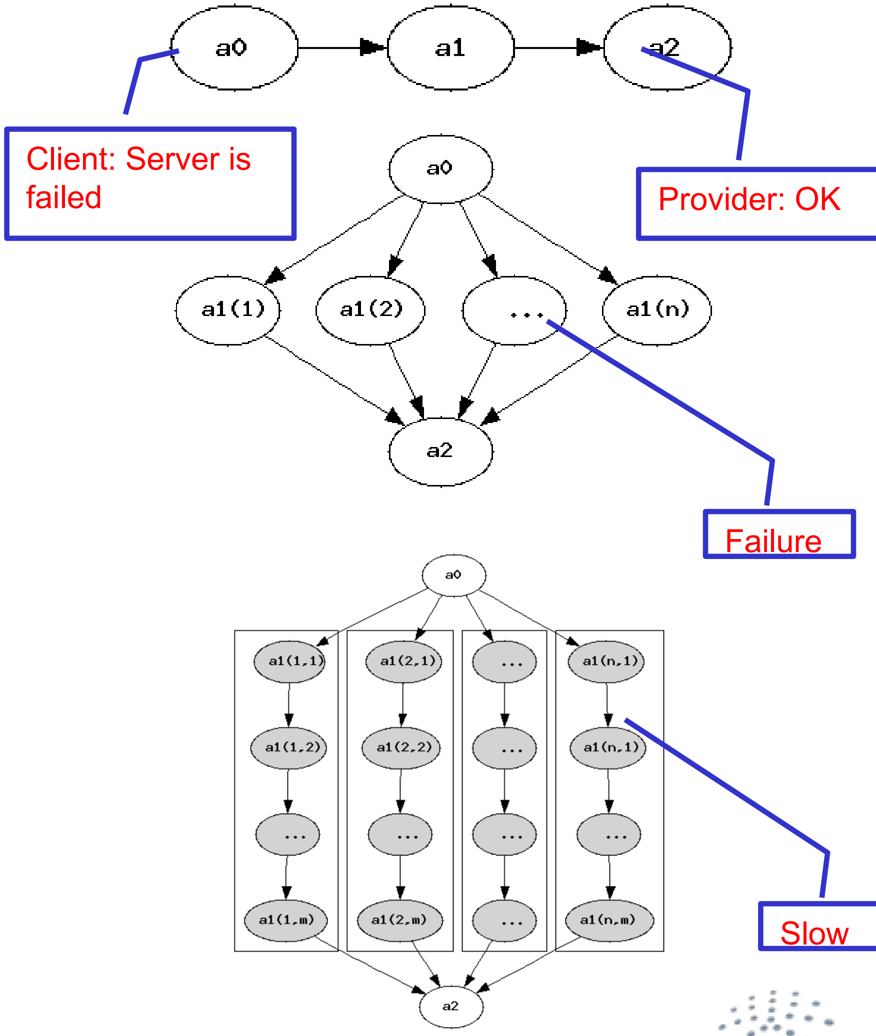
Measurement, Monitoring and Analysis

- 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 measurments
- 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

Dependency Structure



Client: Server is failed

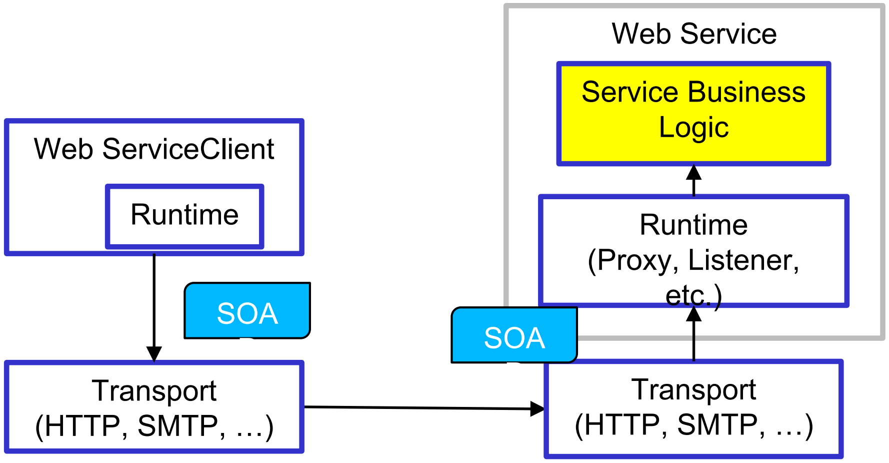
Provider: OK

Failure

Slow

Examples

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


Web Service
Service Business Logic
Web ServiceClient
Runtime
Runtime (Proxy, Listener, etc.)
SOA
SOA
Transport (HTTP, SMTP, …)
Transport (HTTP, SMTP, …)

DEPENDABILITY

Dependability

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

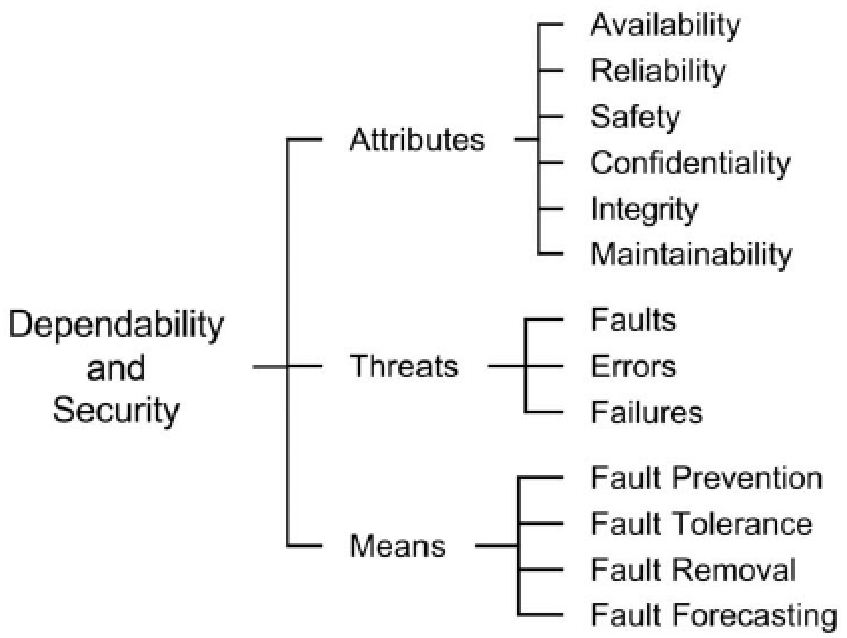
- Important characteristics
  - About avoiding service failures
  - Subjective
  - Defined in a specific context
  - Defined as an average

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

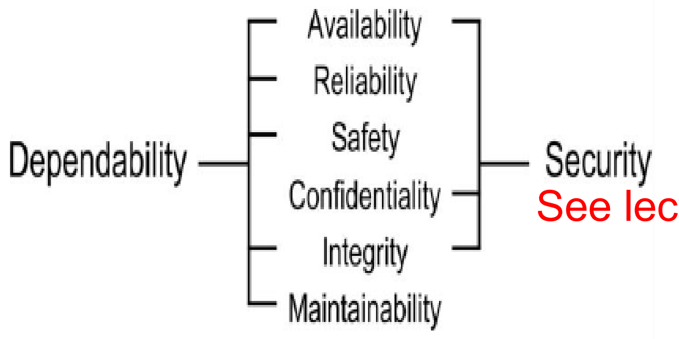
Performability

- 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



Sub mechanisms of resilience techniques



See lecture 9

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

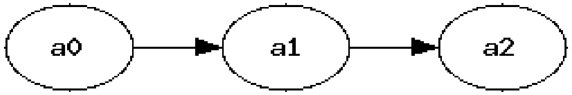
Dependability attributes (2)

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

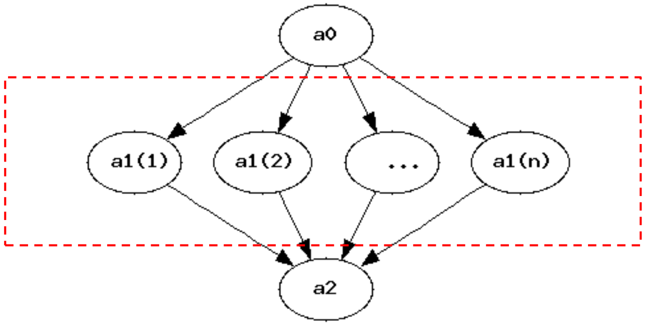
John Knight, Fundamentals of Dependable Computing for Software Engineers, CRC Press, 2012
Algirdas Avizienis, Jean-Claude Laprie, Brian Randell, and Carl Landwehr. 2004. Basic Concepts and Taxonomy of Dependable and Secure Computing. IEEE Trans. Dependable Secur. Comput. 1, 1 (January 2004), 11-33.

- 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.

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

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)

Note: See lecture 9

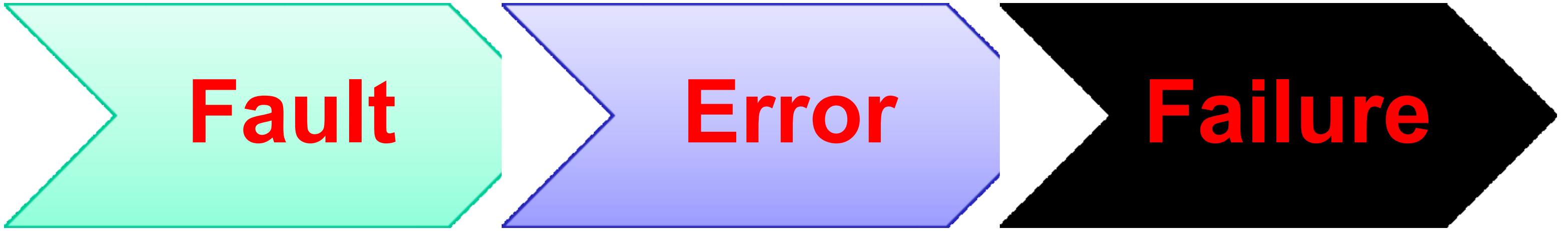
Confidentiality: „the absence of unauthorized disclosure of information“

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



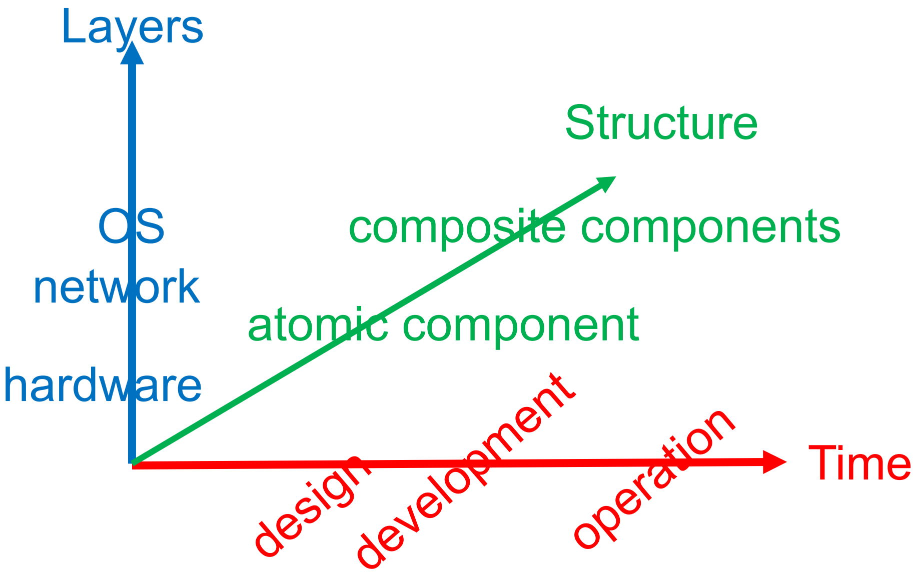
Fault

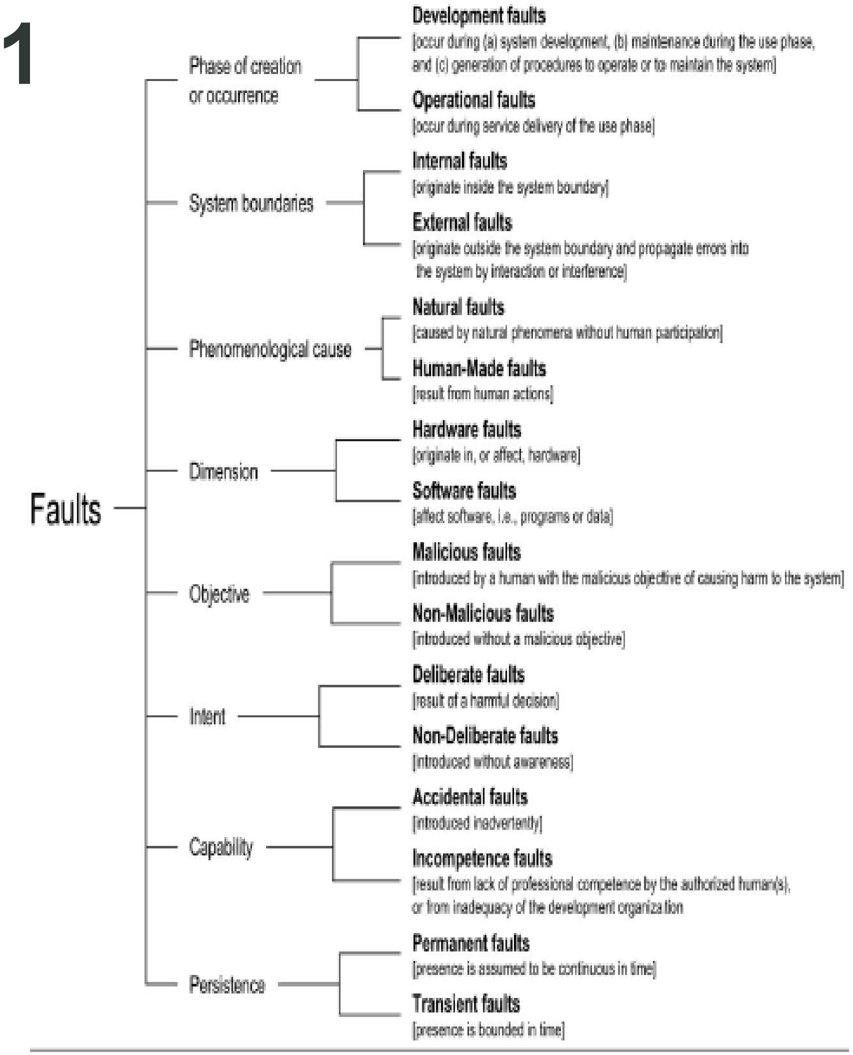
Error

Failure

- 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

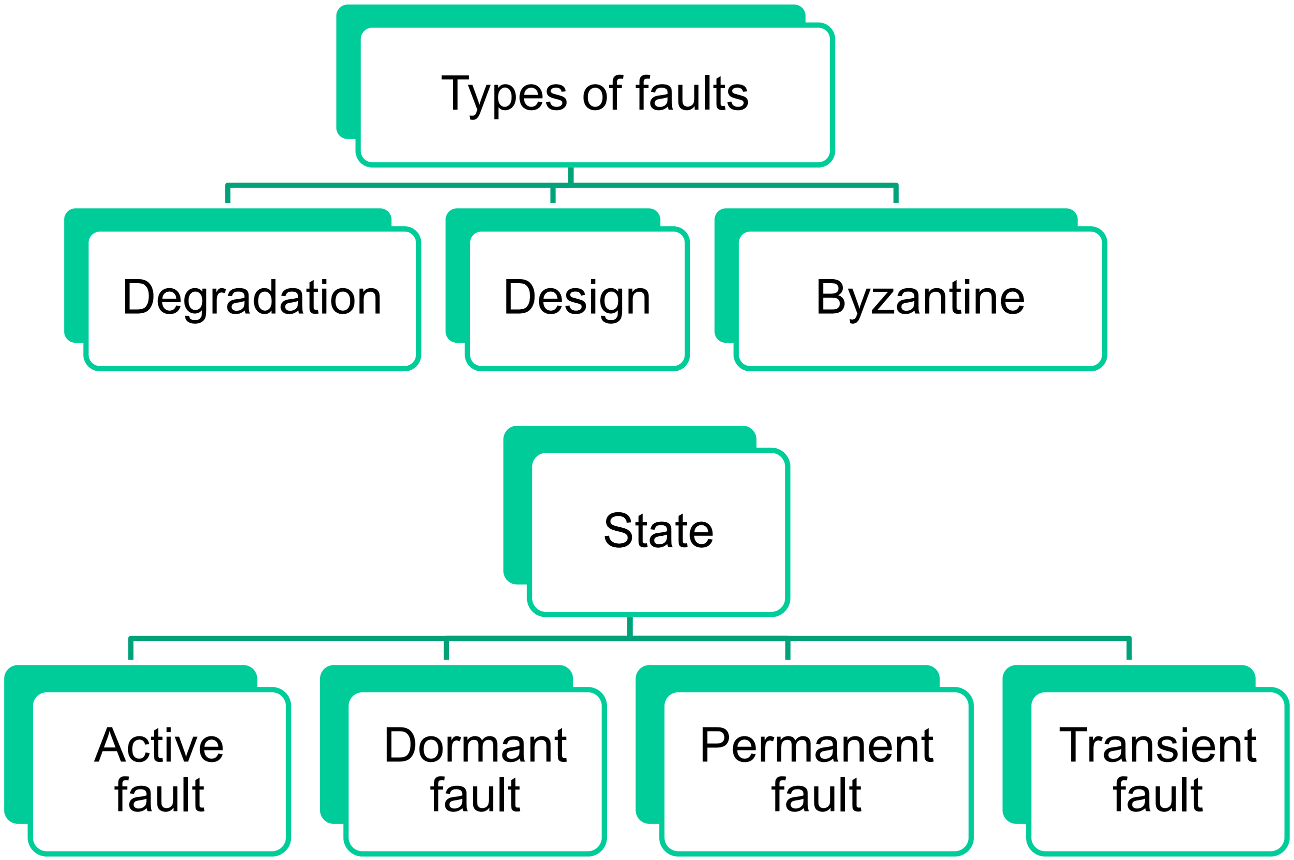
Types of faults (1




Faults
- Phase of creation or occurrence
  - **Development faults** [occur during (a) system development, (b) maintenance during the use phase, and (c) generation of procedures to operate or to maintain the system]
  - **Operational faults** [occur during service delivery of the use phase]
- System boundaries
  - **Internal faults** [originate inside the system boundary]
  - **External faults** [originate outside the system boundary and propagate errors into the system by interaction or interference]
- Phenomenological cause
  - **Natural faults** [caused by natural phenomena without human participation]
  - **Human-Made faults** [result from human actions]
- Dimension
  - **Hardware faults** [originate in, or affect, hardware]
  - **Software faults** [affect software, i.e., programs or data]
- Objective
  - **Malicious faults** [introduced by a human with the malicious objective of causing harm to the system]
  - **Non-Malicious faults** [introduced without a malicious objective]
- Intent
  - **Deliberate faults** [result of a harmful decision]
  - **Non-Deliberate faults** [introduced without awareness]
- Capability
  - **Accidental faults** [introduced inadvertently]
  - **Incompetence faults** [result from lack of professional competence by the authorized human(s), or from inadequacy of the development organization]
- Persistence
  - **Permanent faults** [presence is assumed to be continuous in time]
  - **Transient faults** [presence is bounded in time]

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

Failures

“ …. 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.

…..

Errror

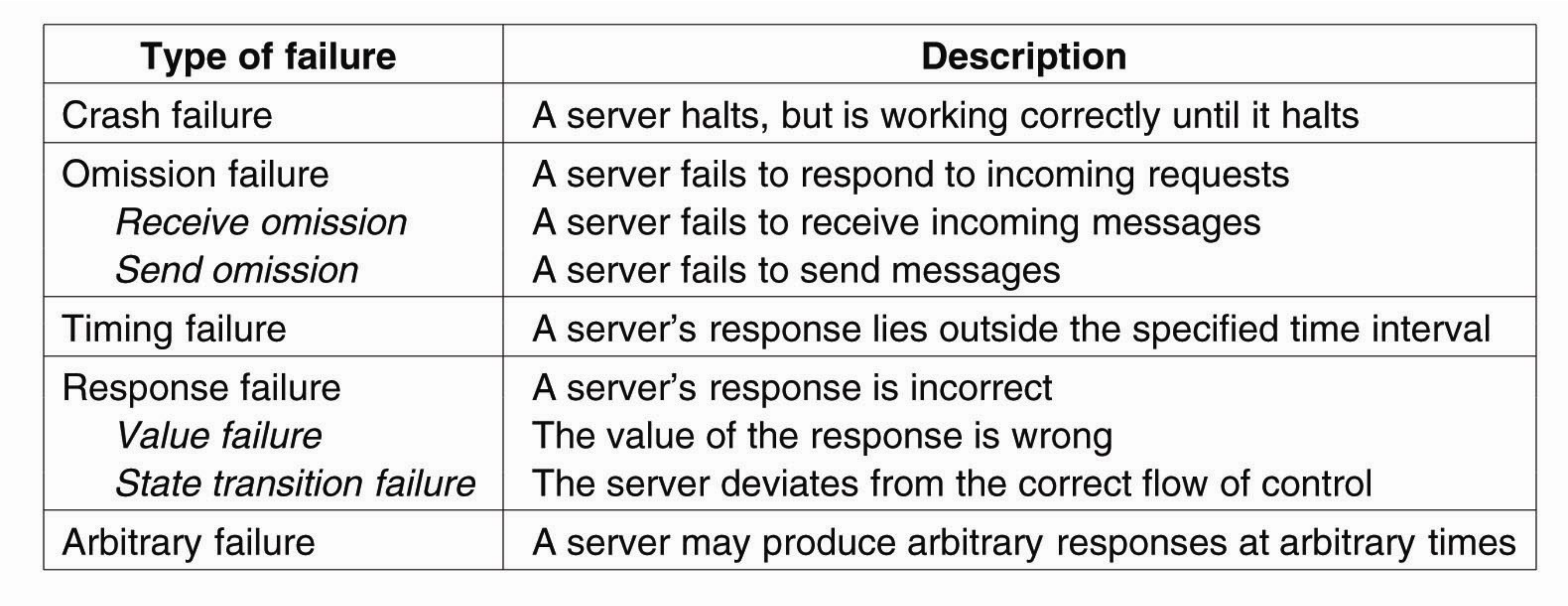
At 10:32am PDT, after exploring several options, we determined that we needed to shut down all communication between Amazon S3 servers, shut down all components used for request processing, clear the system's state, and

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

Fault

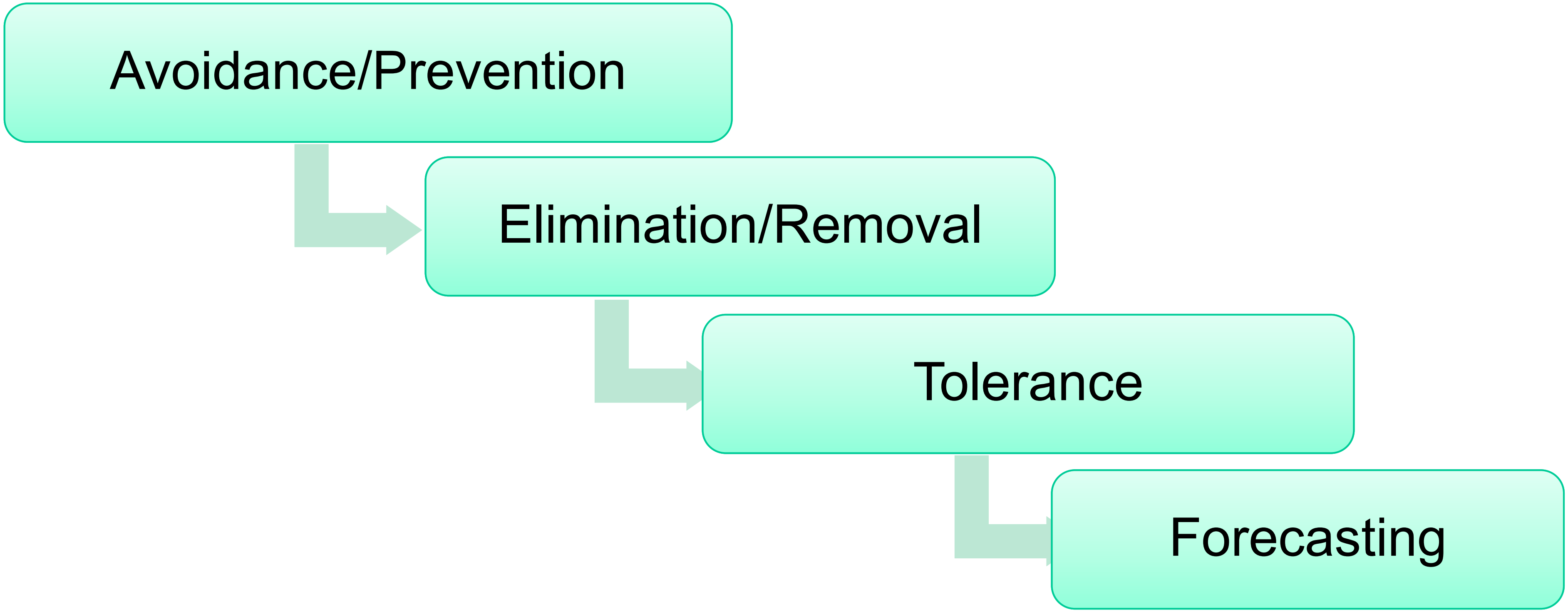
Failure models



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



Avoidance/Prevention

Elimination/Removal

Tolerance

Forecasting

DEALING WITH FAULTS

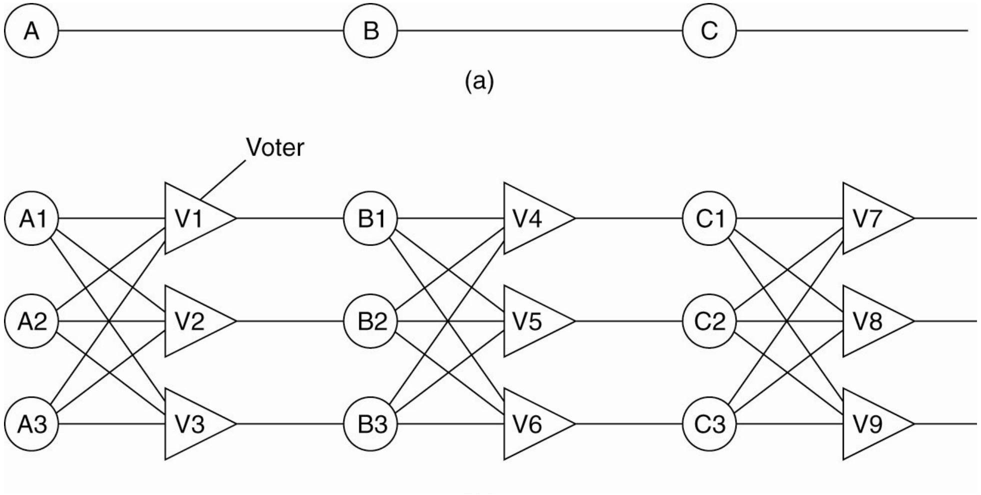
Dealing with faults

- 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

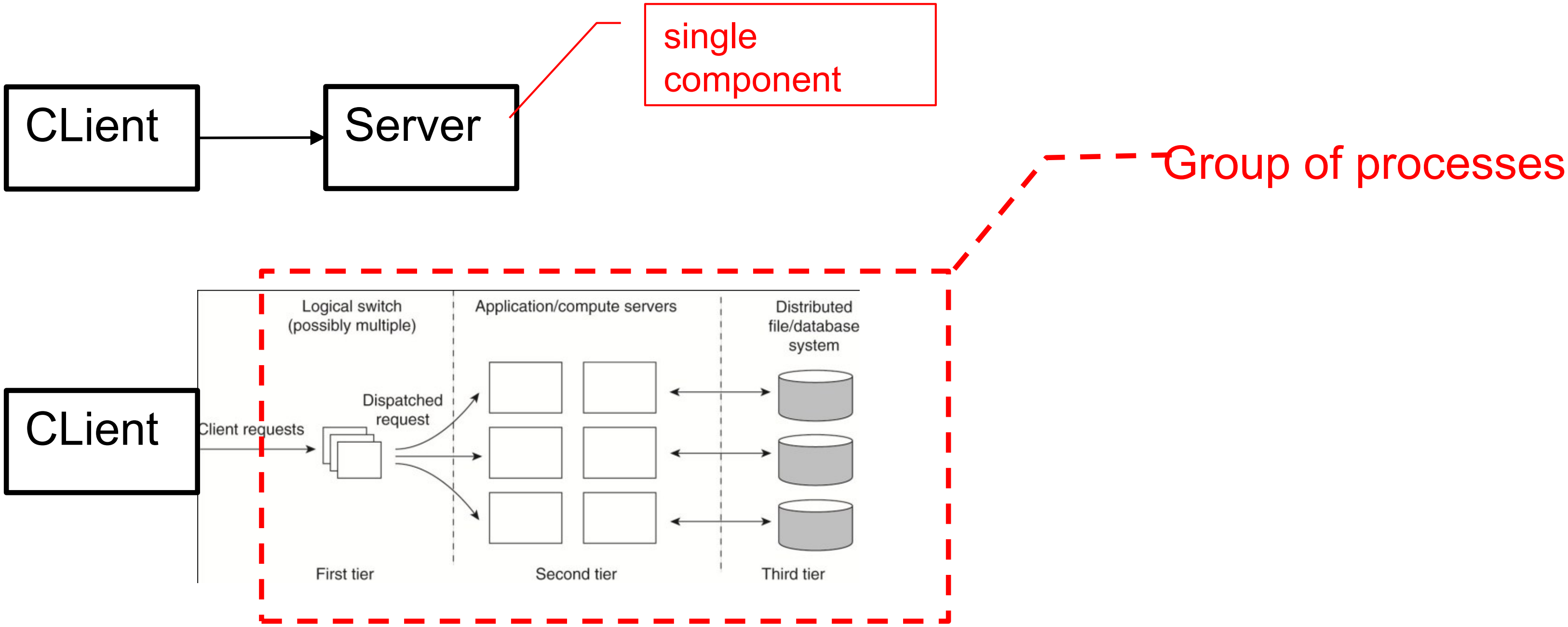
Example of Triple Modular Redundancy



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



CLient

Server

single component

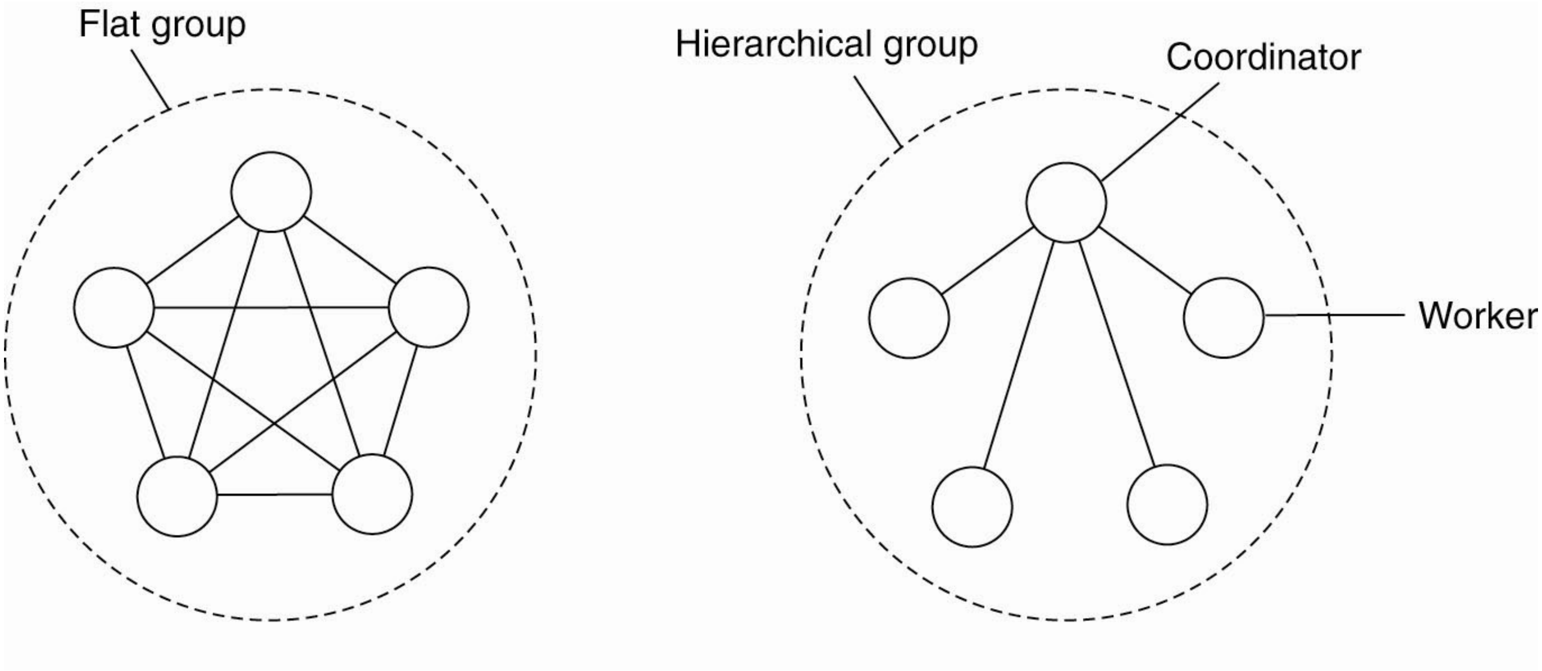
Group of processes

CLient

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

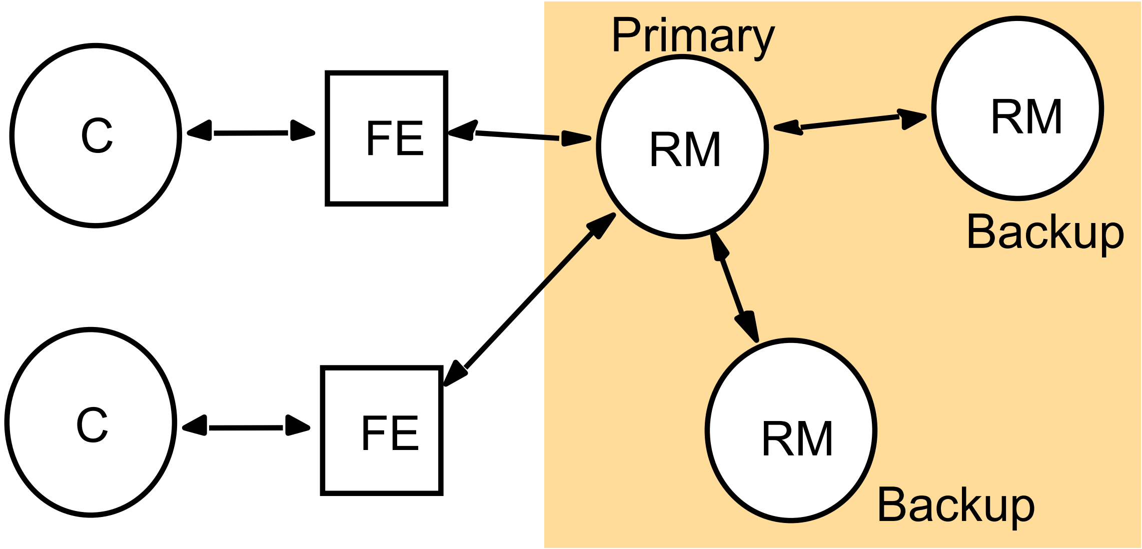


Source: Andrew S. Tanenbaum and Maarten van Steen, Distributed Systems – Principles and Paradigms, 2nd Edition, 2007, Prentice-Hall

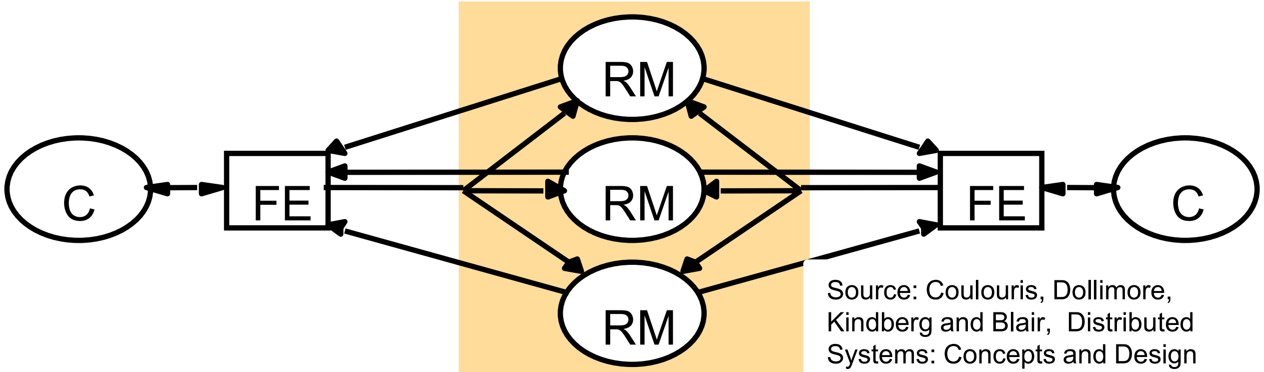
Replication architecture

See lectures 7-8

Passive (Primary backup) model



Active Replication



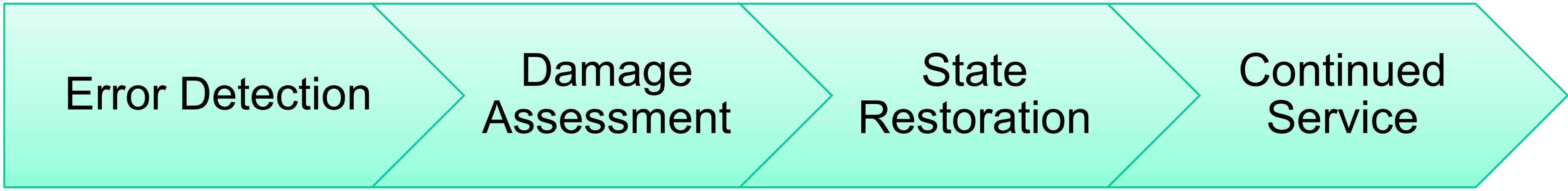
Source: Coulouris, Dollimore, Kindberg and Blair, Distributed Systems: Concepts and Design Edn. 5

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:



Error Detection

Damage Assessment

State Restoration

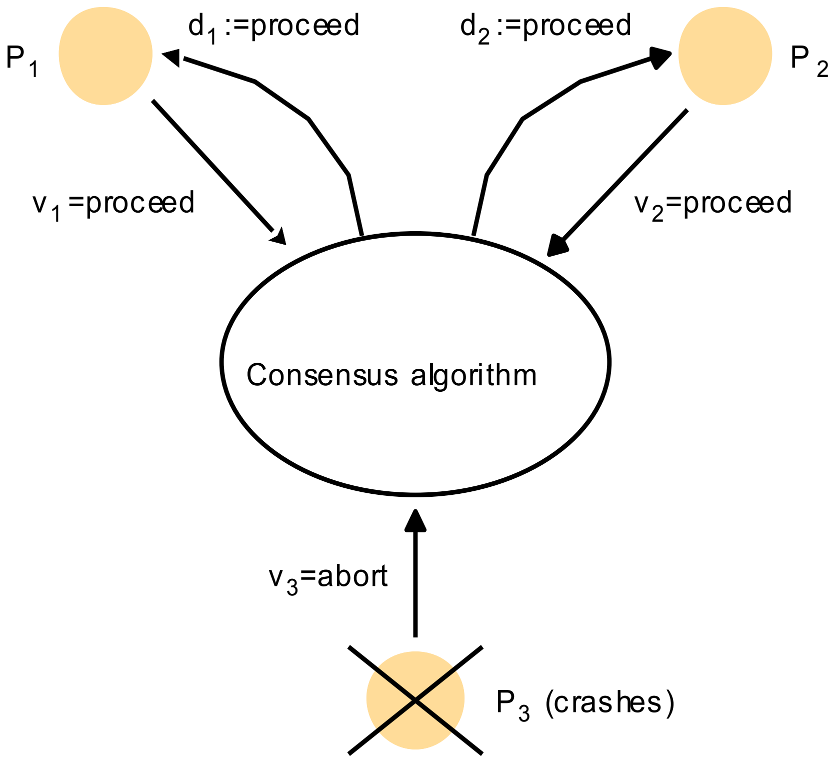
Continued Service

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

Fault-tolerance in arbitrary/Byzantine failures

- How do we deal with fault systems where faults are arbitrary (Byzantine faults)?
- Faults can be omission or commission
- The result is unpredictable



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

Fault tolerance in arbitrary/Byzantine failures

- Byzantine fault-tolerance algorithms are based on agreements among processes
  - A system consists of correct processes and faulty processes and we want to achieve correct service with faults in k processes
  - If an agreement is reached, even in the presence of faults → we could achieve byzantine fault-tolerance
  - Strongly related to consensus problems in distributed systems

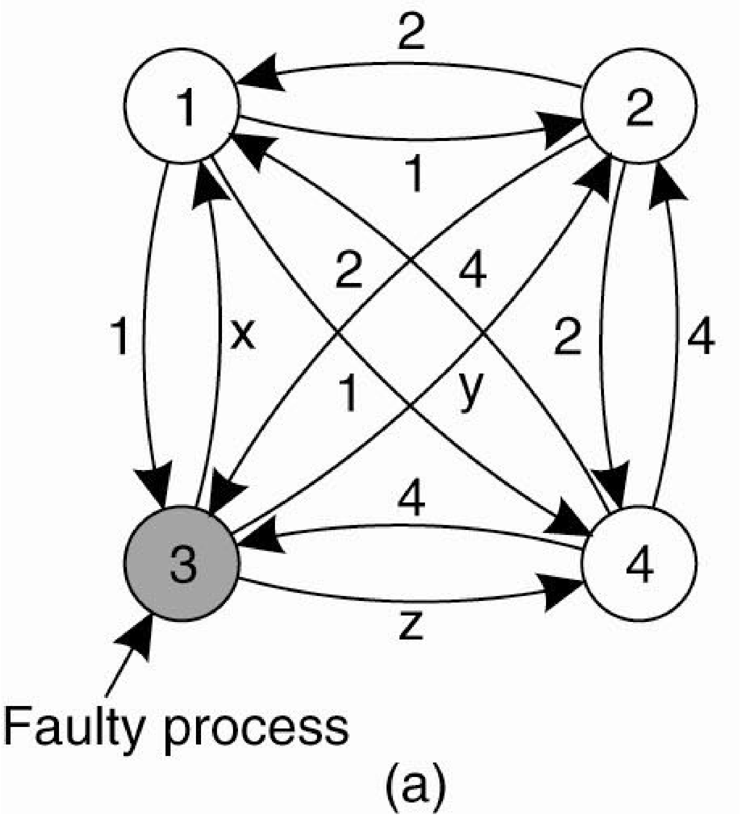
Byzantine fault tolerance algorithms

- Traditional approaches, e.g.
  - Direct/unicast synchronization communication: work if the number of faulty processes is less than one-third of the total number of processes: $n \geq 3k + 1$
  - Low performance
- Practical Byzantine Fault Tolerance
  - High performance Byzantine fault tolerance using replication
  - http://www.pmg.lcs.mit.edu/bft/
  - New protocols: HQ, UpRight, RBFT, BFT-SMaRt, (Archistar) http://archistar.at/, etc.

Fault tolerance in arbitrary/Byzantine failures in synchronization communication

- Synchronous communication
  - Point-to-point (Unicast) communication
  - Message delivery is ordered
  - Delay is bounded
- As long as correct processes agree, the system can move on (ignore the faulty processes)

Example of Byzantine with k fault, n =4



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

Step 1:

- Each process sends messages to all others

Step 2:

- Each process determines a vector of values based on received messages

Step 3

- Each process sends its vector to all other processes

Step 4

- Each process determines a result vector using a majority count for values from received vectors

Example of Byzantine with k fault, n =4

	Sender\Receiver	P1	P2	P3	P4
Step 1	P1	1	1	1	1
	P2	2	2	2	2
	P3	x	y	3	z
	P4	4	4	4	4
Step 2	Vector (P)	(1,2,x,4)	(1,2,y,4)	(1,2,3,4)	(1,2,z,4)

	Sender\Receiver	P1	P2	P3	P4
Step 3	P1		(1,2,x,4)	(1,2,x,4)	(1,2,x,4)
	P2	(1,2,y,4)		(1,2,y,4)	(1,2,y,4)
	P3	(a,b,c,d)	(e,f,g,h)		(i, j, k,l)
	P4	(1,2,z,4)	(1,2,z,4)	(1,2,z,4)	
Step 4	Majority Vote (? == UNKNOWN)	(1,2,?,4)	(1,2,?,4)		(1,2,?,4)

Recovery

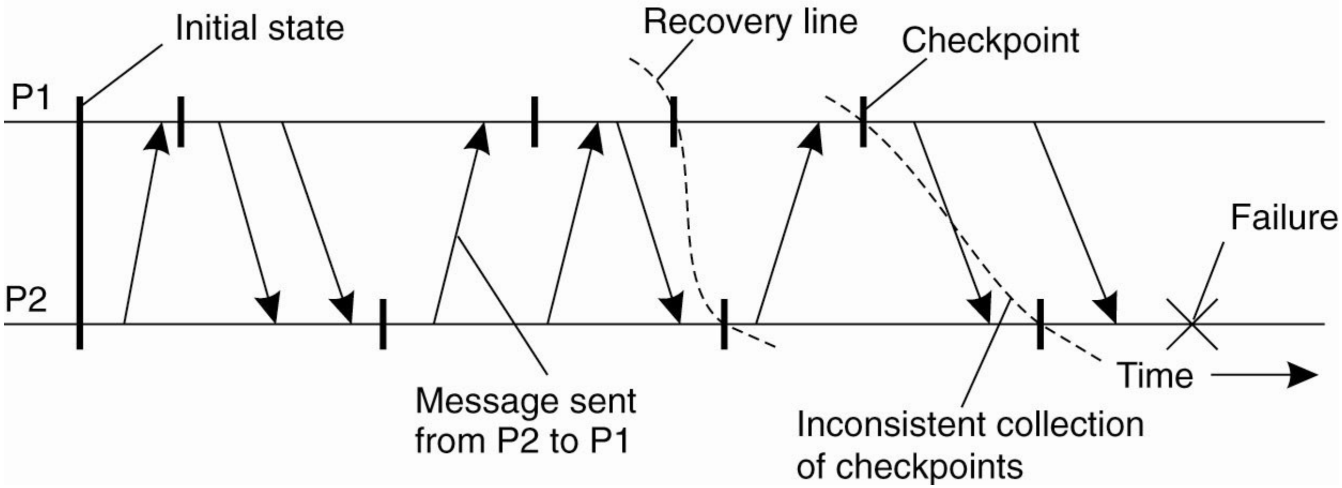
„To replace an erroneous state with an error-free state“

- Rollback- versus forward-recovery
  - Rollback (Backwards): go back to a previous correct state
  - Forward: bring the system into a correct new state
    - This means we have to know the error in advance
- Rollback requires historical records
  - Checkpoint-based rollback recovery
  - Log-based rollback recovery

Checkpointing

Goal: record a consistent global state – a distributed snapshot

Consistent global state: P sends Q a message m: if the state of Q reflects m receipt, then the state of P reflects sending m

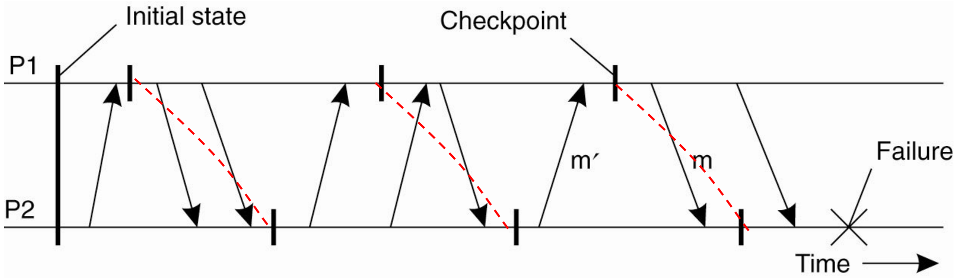

Initial state
Recovery line
Checkpoint
P1
P2
Failure
Time
Message sent from P2 to P1
Inconsistent collection of checkpoints

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

Independent versus coordinated checkpointing

Independent checkpointing:

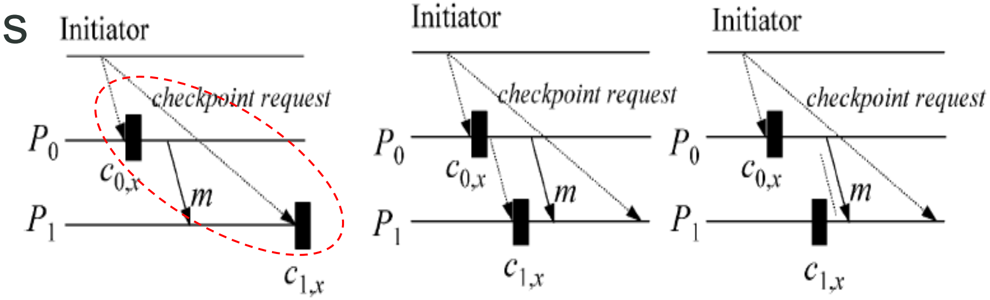
- Each process records its local state without any coordinated action
- Difficult to find a recovery line (domino effect)



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

Coordinated checkpointing:

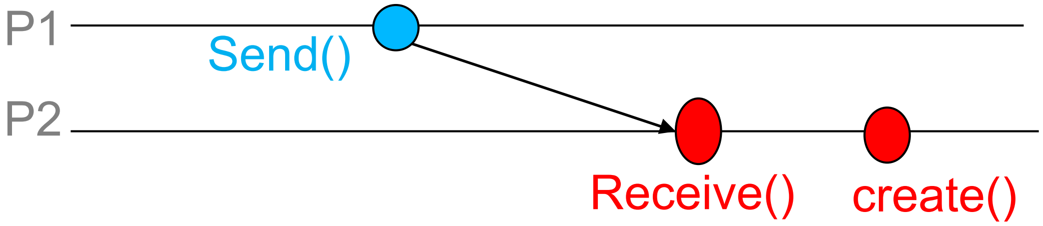
- Coordinate the record of states
- Require synchronization



Source: E. N. (Mootaz) Elnozahy, Lorenzo Alvisi, Yi-Min Wang, and David B. Johnson. 2002. A survey of rollback-recovery protocols in message-passing systems. ACM Comput. Surv. 34, 3 (September 2002), 375-408.

Log-based rollback recovery

- Given a process
  - Nondeterministic events (affect the process): receipt of a message, creation of a process, an internal event
  - Deterministic events (effects caused by the process): send a message
- An interval is an (a → b) (happen-before)

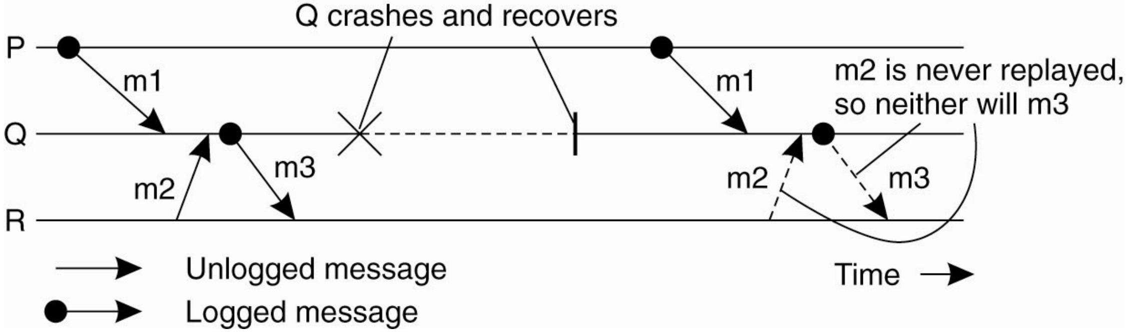


- Program execution is a sequence of deterministic state intervals, each starting with a nondeterministic event
  - all nondeterministic events can be identified and their determinants containing all information necessary for replaying can be logged

→ intervals can be replayed with a known result

Protocols

- Protocols
  - Pessimistic log-based rollback recovery
    - Assumption: A failure can occur after any nondeterministic event
  - Optimistic log-based rollback recovery
    - Assumption: logging will complete before a failure happens
  - Causal log-based rollback-recovery
- Avoid orphan processes is the most important point



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

HOMEWORK

Understanding performance and failures

- Goal: given a system
  - Present some important performance metrics and how to measure and optimize them
  - Identify some faults, possible techniques to deal with these faults
- Some systems/services under investigation
  - Socket with TCP/UDP, Brokers in MOM, MPI, Web services, naming service, clock synchronization systems, P2P, RPC
  - They are in lectures 2-5

Communication

- A socket communication using TCP/UDP
  - Performance: response time, message latency,
  - Dependability:
    - Types of failures: Omission (e.g., TCP lost), or crash (Connection failure)
- MOM
  - Performance: time breakdowns for end-to-end message delivery
- Gossip
  - Dependability: types of failures

RPC

- Performance
  - Metrics: Throughput, Response time, marshalling time, waiting time
  - How to measure them?
- Dependability
  - Types of Failures: Client cannot locate server, Client request is lost, Server crashes, Server response is lost, Client crashes
  - Fault-tolerance techniques

Server handling

- System model:
  - A system handling requests from multiple clients
  - The system utilizes several servers and a single load balancer
- Performance: throughput, response time, waiting time
- Dependability
  - Availability: does increasing the number of servers increase the availability?
  - Faults: which are possible faults at runtime?
  - Fault-tolerance: Peer-to-peer Group or Hierarchical Group

Summary

- Understanding performance and dependability is the key for designing, operating and optimizing distributed systems
- Dependability and performance are highly complex and interdependent

- Fault tolerance techniques are just a sub set of techniques for dealing with failures
  - With cloud computing we could introduce more techniques
- Evaluating performance and dependability requires a careful look at metrics, type of faults, and system structures based on different views

Thanks for your attention

Hong-Linh Truong
Distributed Systems Group
Vienna University of Technology
truong@dsg.tuwien.ac.at
http://dsg.tuwien.ac.at/staff/truong