Features of Distribution

Distributed Systems

Sistemi Distribuiti

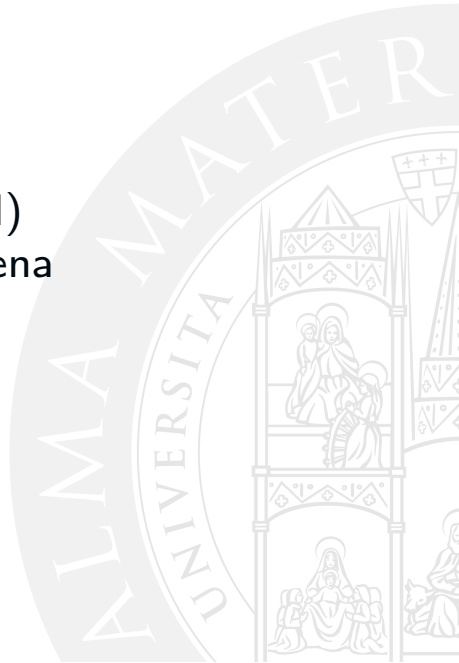
Andrea Omicini

andrea.omicini@unibo.it

Dipartimento di Informatica – Scienza e Ingegneria (DISI)

Alma Mater Studiorum – Università di Bologna a Cesena

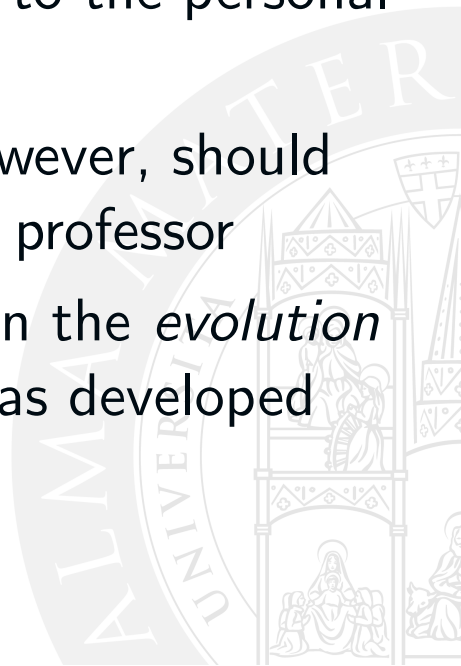
Academic Year 2017/2018



About these Slides

- the following slides borrow a great deal from the slides coming with the book adopted as the basic one for this course

  [Tanenbaum and van Steen, 2007]
- material from those slides (including pictures) has been re-used in the following, and integrated with new material according to the personal view of the professor
- every problem or mistake contained in these slides, however, should be attributed to the sole responsibility of this course's professor
- the goal, here, is to be exposed to the classical view on the *evolution* of distributed systems that the scientific community has developed over the last decades



Next in Line...

1. Replication & Consistency
2. Fault Tolerance

Reasons for Replication

Replication of data

- increasing the reliability of systems
- improving performance
- scaling
  - in numbers
  - in geographical area

Issues of Replication

Benefits in distributed systems

- reliability
- fault tolerance
- accessibility
- performance
- scalability

Problems in distributed systems

- costs
  - computational resources
  - bandwidth
- consistency

The Problem of Consistency

Consistency models

- before discussing how to face the problem of consistency, we need to define the notion of consistency itself
- different interpretations are available, some of them fitting one or more application scenarios
- this leads to the definition of different *models of consistency*...
- ... which are then amenable of different implementations, whose features may affect the effectiveness of the model

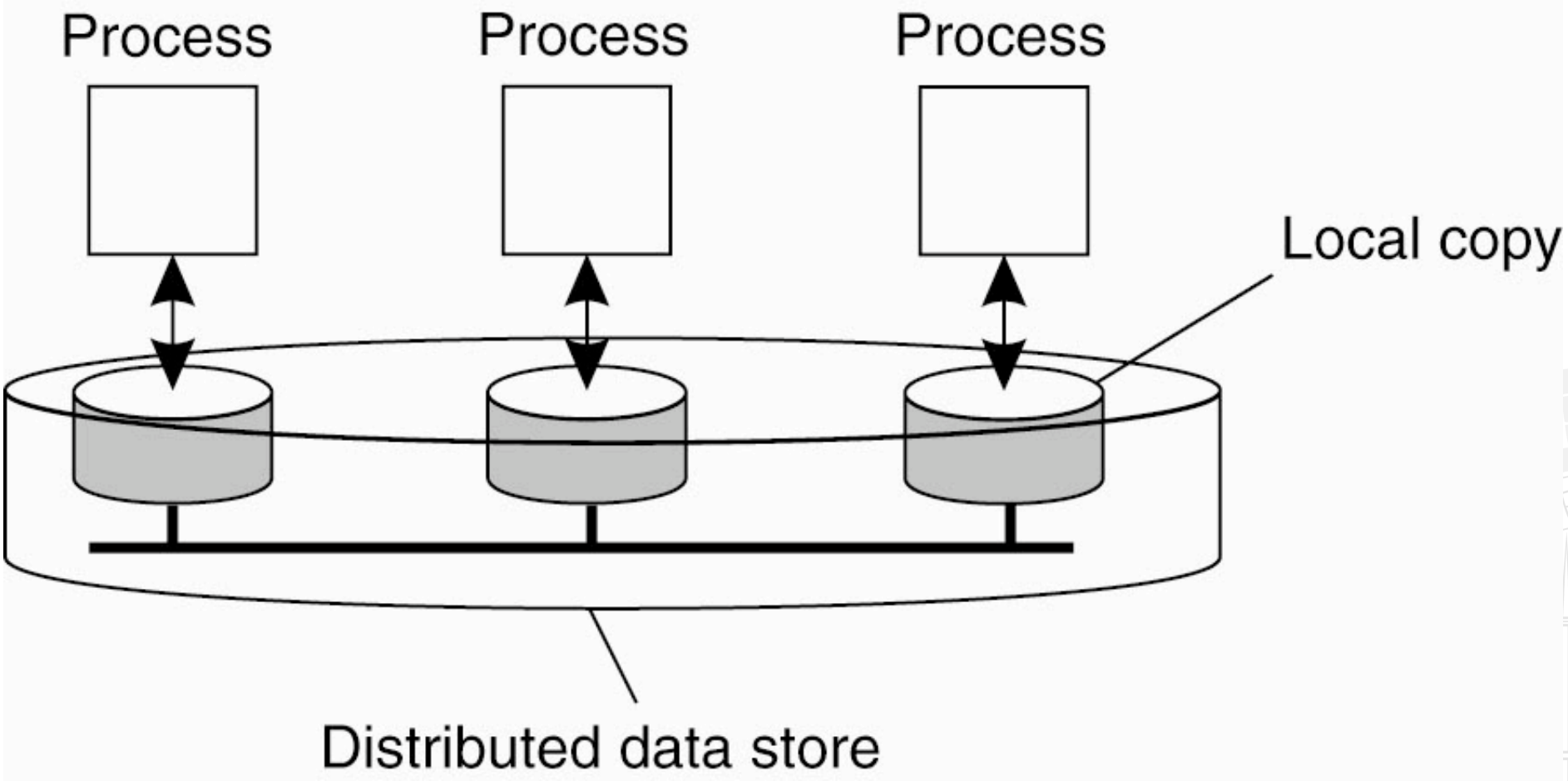
Focus on...

Conceptual Premise

Replicating data

- historically, the first things to be distributed are data
- so, the first problem to be addressed is how to ensure *consistency of data* across distributed copies...
- ... and the actions to be accounted for are *operations over data*

General Organisation of a Distributed Data Store



[Tanenbaum and van Steen, 2007]

Consistency Model

Definition

A consistency model is essentially a contract among the processes and the data stores, ensuring the correctness of data given a set of rules that processes have to follow

Of course, what is "correct" also depend on what processes expect—which might be also difficult to define in absence of a global clock

Observation

The above definition of consistency model shifts the focus from the replicated data to the processes using data—so, focussing on the notion of consistency in the *use* of data, based on the application needs

Continuous Consistency

Goal

Imposing limits to deviations between replicas

Deviations

- *numerical* deviations—absolute / relative
- *staleness* deviations—e.g., "fresh" weather reports
- *ordering* deviations—e.g., distributed updating of replicas, waiting for confirmation

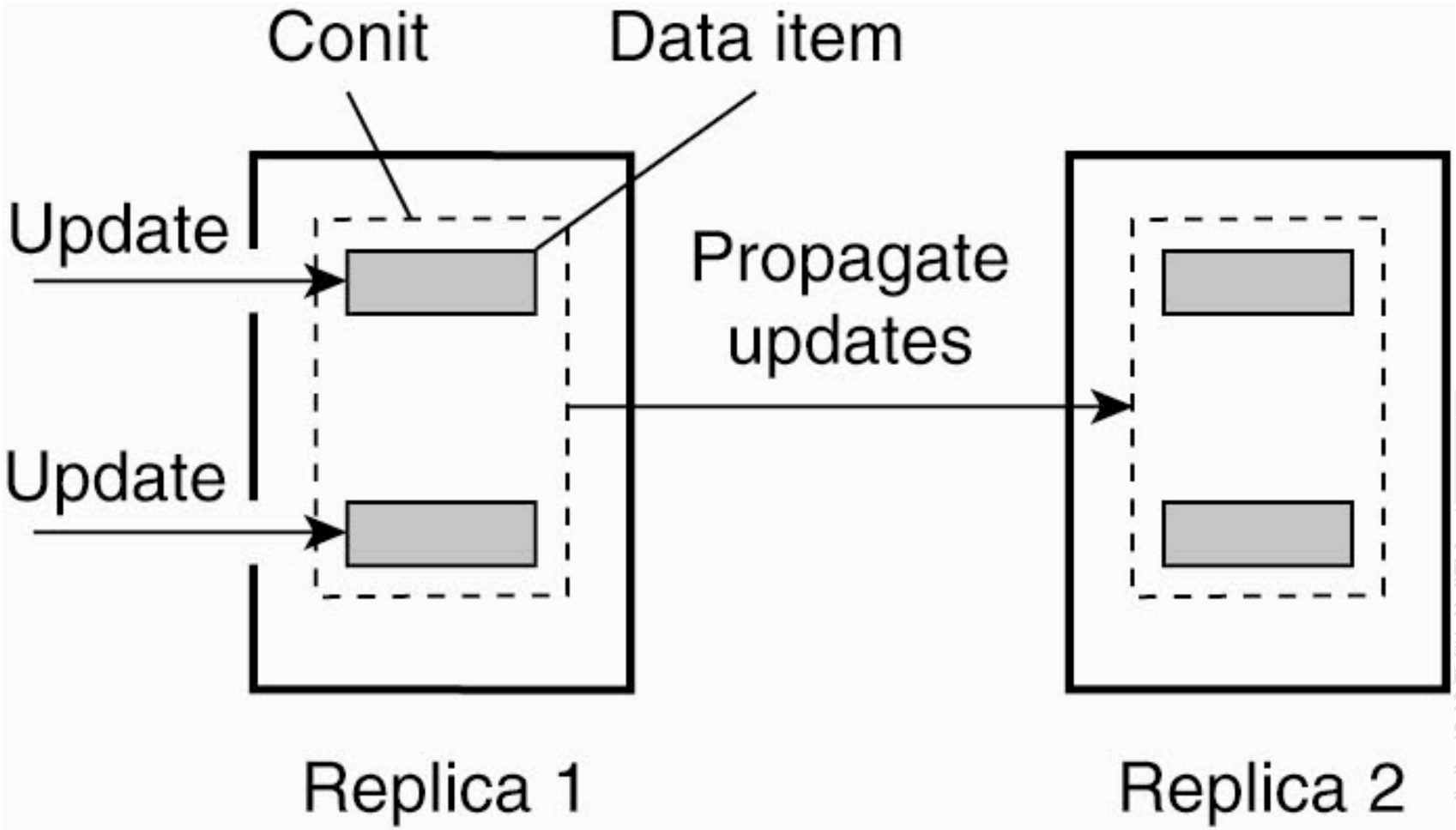
This defines the notion of continuous consistency

Inconsistency

Conit

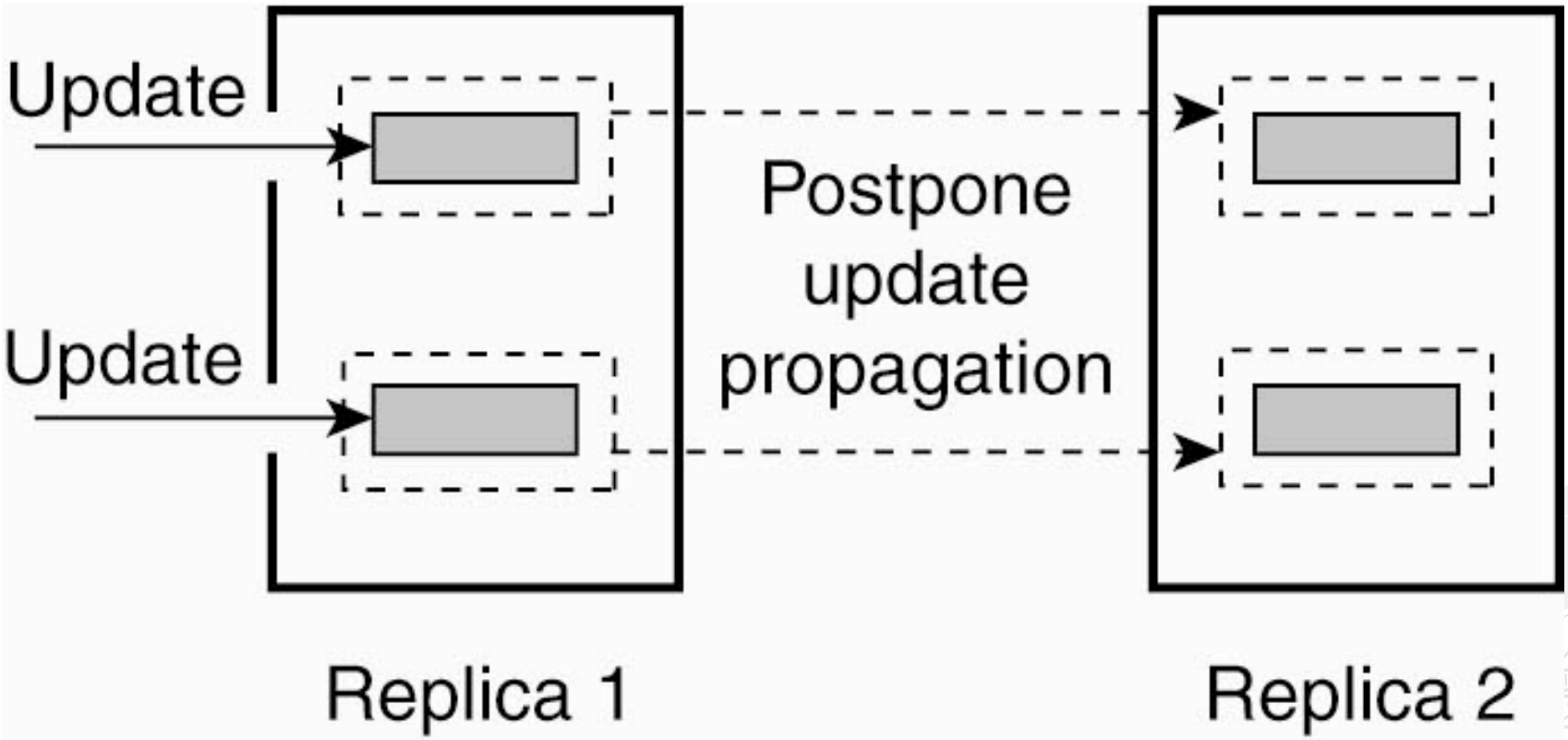
- notion of unit of consistency, called *conit*
- each data store either implicitly or explicitly suggests its conit
- however, a consistency model (and replication) is defined around a suitably-designed notion of conit
- deviation is measured in terms of *differences of conits*

The Choice of Granularity of Conit I



Larger conit [Tanenbaum and van Steen, 2007]

The Choice of Granularity of Conit II



Smaller conit, minor need for propagation [Tanenbaum and van Steen, 2007]

Consistent Operations Ordering

Main issue

- from parallel and concurrent environments, where several processes share resources, and have to access them simultaneously
- new models conceptually extending data-centric ones: when committing on a state for replicas, an agreement has to be reached among processes upon the global ordering of updates

Sequential Consistency

Main idea

- all update operations are seen by all processes in the same order

Definition

- a data store is sequentially consistent when the result of any execution is the same as if the (read and write) operations by all processes on the data store were executed in some sequential order, and the operations of each individual process appear in this sequence in the order specified by its program

Causal Consistency

Main idea

- weakening sequential consistency
- based on the notion of cause/effect relation
- unrelated operations are *concurrent* ones
- ordering is limited to operations in cause/effect relation

Definition

- a data store is causally consistent when all processes see write operations in cause/effect relation in the same order

Focus on...

Switching Perspective

Sharing data in mobile computing scenario

- a client connects with different replicas over time
- differences between replicas should be made transparent
- no particular problems of simultaneous updates, here

Client-centric consistency models

- in essence, they ensure that whenever a client connects to a new replica, that replica is up to date according to the previous accesses of the client to the same data in the other replicas on different sites
- consistency is no longer referred directly to the resource – its nature, its dynamics, . . . –, but rather on the view that clients have of the resource

Eventual Consistency

Scenario

- large, distributed data store with almost no update conflicts
- typically, a single authority updating, with many processes simply reading
- the only conflict is *read-write conflict* where one process wants to update a data item while another concurrently attempts to read the same data
- examples: DNS changes, Web content

Issues

- non-updated data may be provided to readers
- in most cases, such an inconsistency might be acceptable to readers
- typically, if no update takes place for a while, gradually all replicas will become consistent
- this sort of consistency is called eventual consistency

Monotonic Reads

Definition

A data store is said to provide monotonic-read consistency if the following condition holds

- if a process reads the value of a data item x, any successive read operation on x by the process will always return that same value or a more recent value

Example

- distributed e-mail database

Monotonic Writes

Definition

A data store is said to provide monotonic-write consistency if the following condition holds

- a write operation by a process on a data item x is completed before any successive operation on x by the same process

Idea

- the order of updates is maintained over distributed replicas

Example

- software library under development

Read Your Writes

Definition

A data store is said to provide read-your-writes consistency if the following condition holds

- the effect of a write operation by a process on data item x will always be seen by a successive read operation on x by the same process

Idea

- avoid the "web page failed update" effect

Example

- password updating

Writes Follow Reads

Definition

A data store is said to provide writes-follow-reads consistency if the following condition holds

- a write operation by a process on data item x following a previous read operation on x by the same process is guaranteed to take place on the same or a more recent value of x that was read

Idea

- writes affect only up-to-date data items

Example

- comments to posts on FaceBook

Focus on...

Key Issue of Replication

Supporting replication in a distributed system

- means deciding where, when and by whom replicas should be placed
- and which mechanisms should be adopted to keep replicas consistent

Two subproblems

- placing *replica servers*
- placing *content*

! not the same problem indeed

Focus on...

Replicating Services

Replication does not mean replicating data—not merely

- services could be replicated as well in a distributed setting
- for the same reasons of data stores
- this essentially means replicating functions, which may / may not insist on the same data store
- two layers for replications, with two consistency & replication models

Replicating Processes

Mobility & cloning for replication

- processes could also be replicated in a distributed mobile setting
- again, for the same reasons of data stores
- this might require cloning...
- ... but also higher-level mechanisms, like goal-passing

Next in Line...

Failure in Distributed Systems

Partial failure

- a typical feature of distributed systems is the notion of partial failure
- one component may fail, while the rest of the systems keeps running
- while the functionality guaranteed by the failed component is compromised, this does not necessarily holds for the other components, as well as for the overall system

Engineering distributed systems with failure

- when engineering a distributed systems, a twofold goal is possible
  - reducing the impact of failure of a single component on the others, and on the overall system performance
  - exploiting partial failure to recover from failure

Focus on...

Dependable Systems

Main features of dependable systems

- availability
- reliability
- safety
- maintainability

Dependability is closely related to fault tolerance

Availability

Definition

Availability refers to the property that a system is ready for immediate use

This means...

- ... that availability refers to the probability that a system is operating correctly at any given moment, ready to provide users with its functions
- so, a highly-available system is a system that is most likely to be ready and working at any given instant of time

Reliability

Definition

Reliability refers to the property that a system can run continuously without failure

This means...

- ... that reliability is defined in terms of a time interval, rather than of a instant – as in the case of availability
- so, a highly-reliable system is a system that is most likely to keep on running for a long period of time

Safety

Definition

Safety refers to the situation that when a system temporarily fails to operate correctly, nothing catastrophic happens

This is...

- ... a very difficult property to be defined, and to be ensured as well

Maintainability

Definition

Maintainability refers to how easily a failed systems can be repaired

This means...

- ... that maintainability is closely related to availability
- so, a highly-maintainable system may also show a high degree of availability

Focus on...

Faults I

Failure

- a system is said to *fail* when does not behave as promised
- an *error* is a part of a system state that might have caused a failure
- the cause of an error is a *fault*

Faults II

Fault tolerance and dependable systems

- building a dependable system closely relates to controlling faults
- one may distinguish between
  - preventing faults
  - removing faults
  - forecasting faults
- in distributed system, the most important issue is *fault tolerance*
- as the property of a system to provide its function even in the presence of faults

Faults III

Sorts of faults

- *Transient faults* occur once then disappear
- *Intermittent faults* occur, vanishes of its own accord, then reappears, and so on
- *Permanent faults* keep on existing until the faulty component is replaced /fixed

Failure Models

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

Different types of failures [Tanenbaum and van Steen, 2007]

Failure Masking by Redundancy

Idea

- hiding failures from other processes
- the key technique for masking faults is *redundancy*

Three kinds of redundancy

- information redundancy
  - e.g., extra bits
- time redundancy
  - e.g., redos after transaction aborts
- physical redundancy
  - typical in biological systems

References

Tanenbaum, A. S. and van Steen, M. (2007).
Distributed Systems. Principles and Paradigms.
Pearson Prentice Hall, Upper Saddle River, NJ, USA, 2nd edition.

Features of Distribution

Distributed Systems

Sistemi Distribuiti

Andrea Omicini

andrea.omicini@unibo.it

Dipartimento di Informatica – Scienza e Ingegneria (DISI)

Alma Mater Studiorum – Università di Bologna a Cesena

Academic Year 2017/2018