Introduction to Distributed Systems

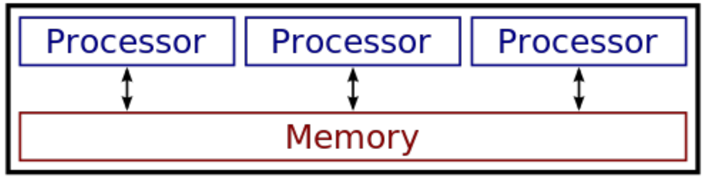
CS4405 – Analysis of Concurrent and Distributed Programs

Burcu Kulahcioglu Ozkan

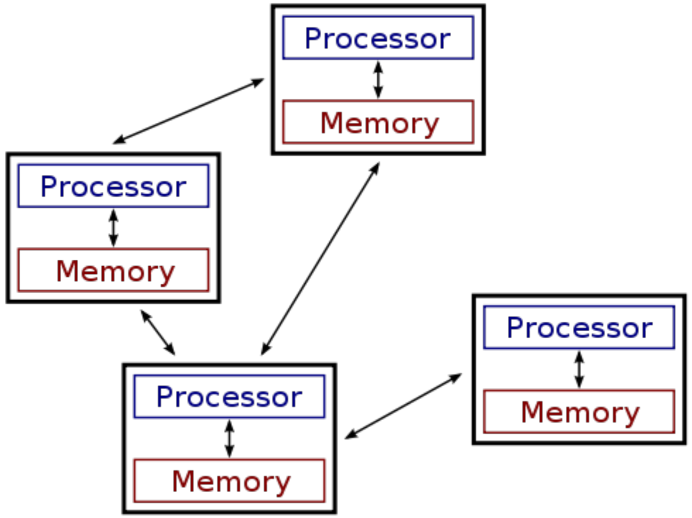
TU Delft

Shared-memory vs distributed systems

What concurrency problems do you think are common or different in these systems?



Shared-memory



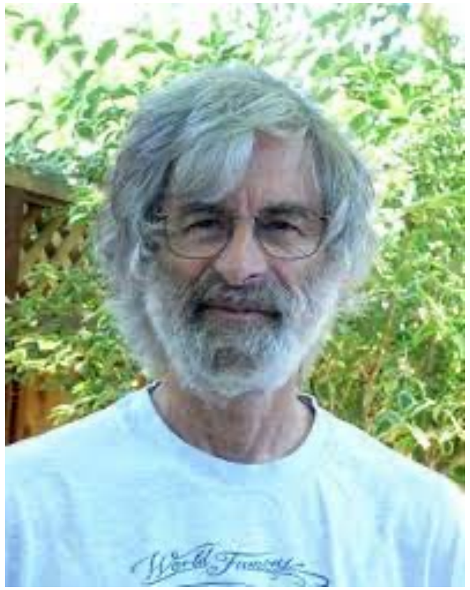
Distributed

Image credit: Wikipedia

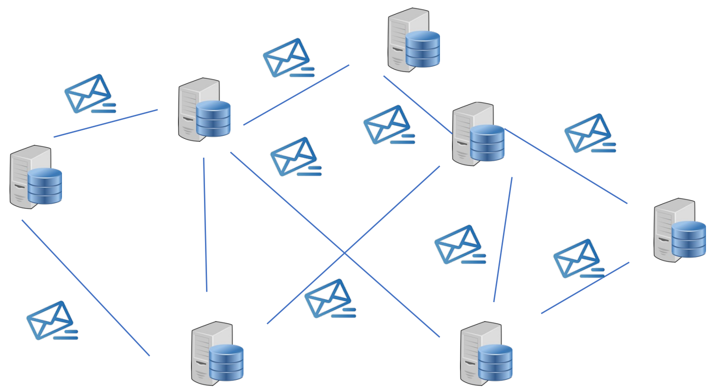
What is a distributed system?

- “. . . a system in which the failure of a computer you didn't even know existed can render your own computer unusable.”

  — Leslie Lamport



- The computers in the system:
  - Are connected over network
  - Communicate by exchanging messages



Why distributed systems?

Inherent Distribution:

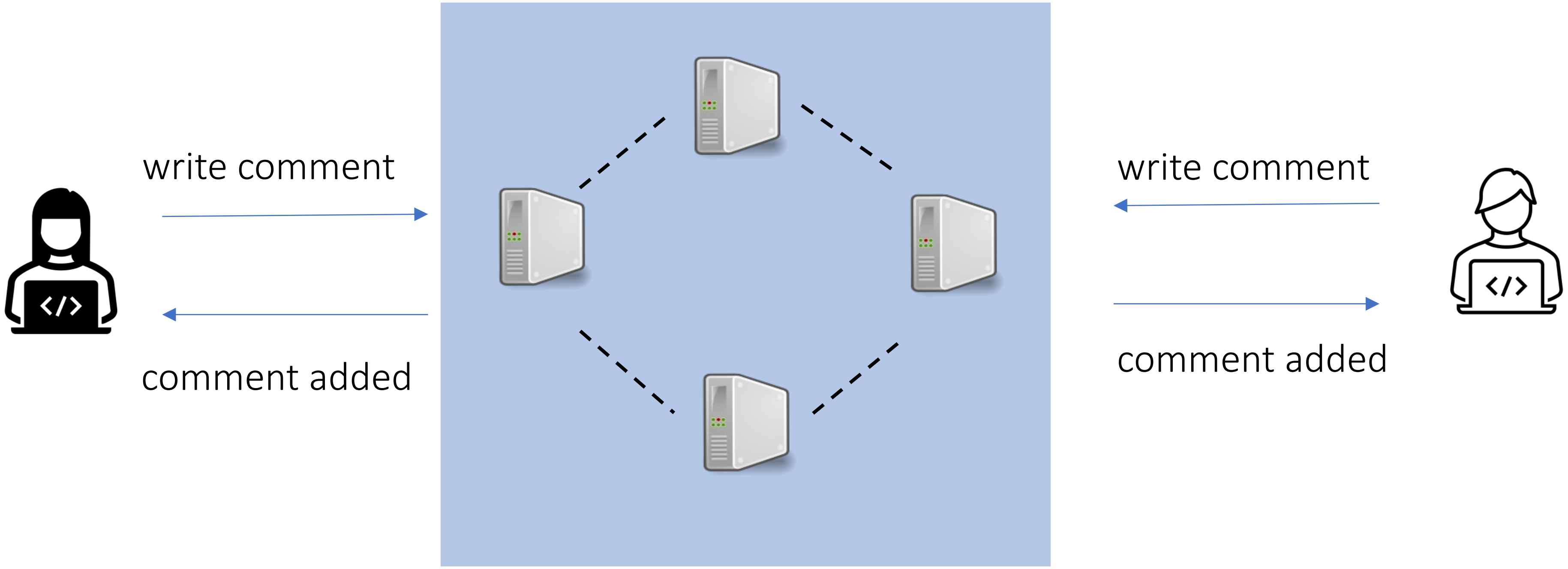
- Information dissemination (publishers/subscribers)
- Distributed process control
- Cooperative work (different nodes on a network read/write)
- Distributed storage

Distribution as an Artifact:

- Performance
- Scalability (data, geographical, functional)
  - Moore's law: The number of transistors *on a single chip* doubles about every two year.
  - The advancement has slowed since around 2010.
  - Distribution provides massive performance.
- Availability
- Fault tolerance

Introduction to reliable and secure distributed programming. Cachin, Christian, Rachid Guerraoui. Springer, 2011

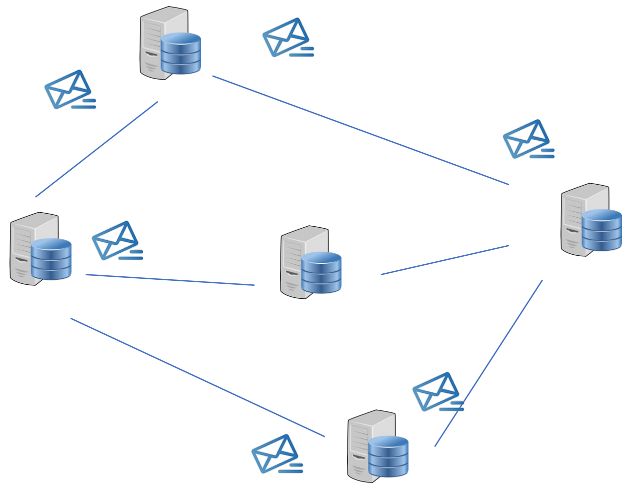
A simple distributed system example



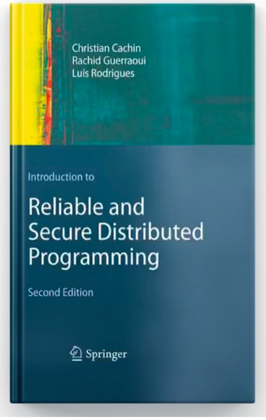
Fallacies of distributed systems

By Peter Deutsch

- The network is reliable
- Latency is zero
- Bandwidth is infinite
- The network is secure
- Topology does not change
- Transport cost is zero
- The network is homogeneous



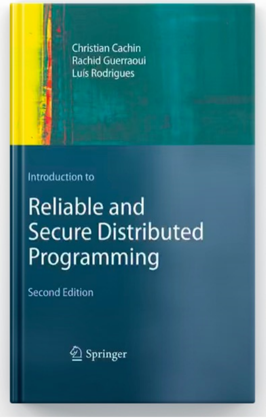
Distributed system abstractions



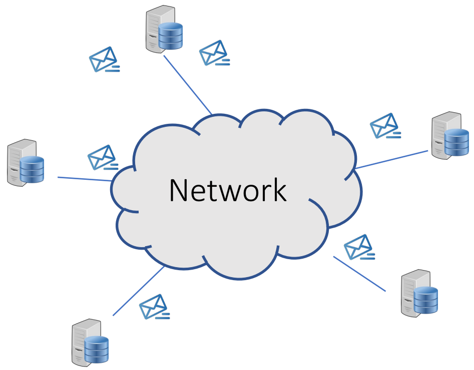
- Abstracting the underlying physical system into a system model:
  - Processes and messages
  - Communication network
- We start with a primitive link abstraction (e.g. from point A to point B), and layer on abstractions to build stronger guarantees.
- Abstractions for common interaction patterns (incremental):
  - Process identities (processes agree on who they are)
  - Consensus (processes agree on a common plan)
  - Atomic commitment (processes take a step only if all processes agree)
  - Total order broadcast (processes agree on a total order of actions)

Introduction to reliable and secure distributed programming. Cachin, Christian, Rachid Guerraoui. Springer, 2011

Processes and messages



- Processes: “units that are able to perform computations in a distributed system”
- A distributed system consists of a collection of automata, one per process
- The execution of a distributed algorithm is represented by a sequence of steps executed by the processes:
  - Receiving a message
  - Executing a local computation
  - Sending a message to a process



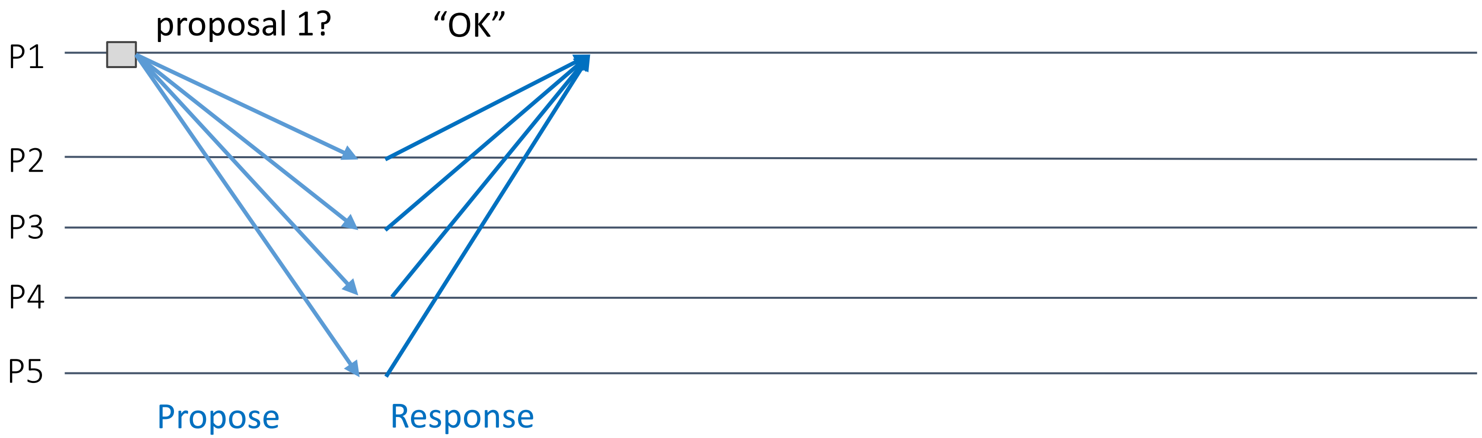
Abstracting Processes: Process failures

- Crash fault
- Omission fault
- Crash-recovery fault
- Arbitrary/Byzantine fault

System synchrony model: **Synchronous**

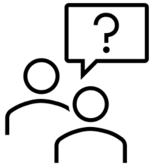
- Process execution speeds or message delivery times are bounded.

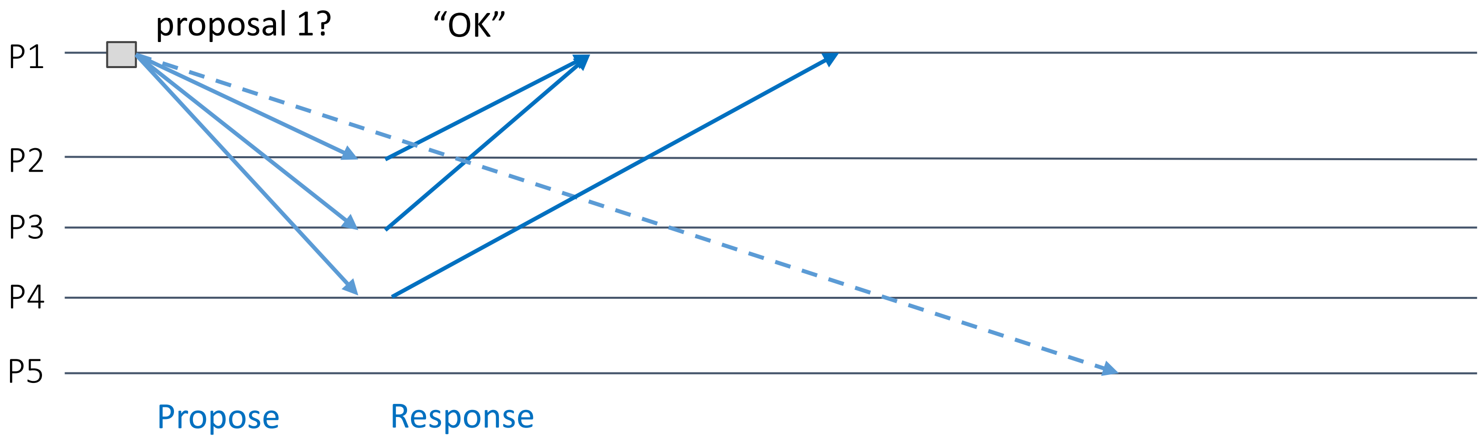
  In a synchronous system, we can have:
  - Timed failure detection
  - Time based coordination
  - Worst-case performance



System synchrony model: **Asynchronous**

- No assumptions about process execution or message delivery times are made
- Upon waiting for a response to a requests, it is **not possible to distinguish** whether:
  - the request was lost
  - the remote node is down
  - the response was lost





System synchrony model: **Partially-synchronous**

The retransmission of the messages may help ensure the reliability of the communication links but introduce unpredictable delays.

Set timeouts and retry the request until it succeeds

In this sense, practical systems are partially synchronous:

- **Partially-synchronous system**:
  - There exist upper bounds on the network delay but the programmer does not know them.
  - Eventually synchronous: The network may behave asynchronously for some duration, but eventually behaves synchronously.