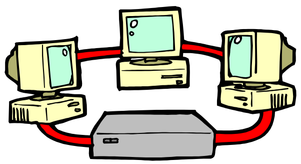
Distributed Algorithms

Jeff Magee & Jeff Kramer

With grateful acknowledgement to **Christos Karamanolis** for much of the material



Course Outline

- Models of distributed computing
- Synchronous message-passing distributed systems
  - Algorithms in systems with no failures
  - The commit problem
  - Consensus problems
- Asynchronous message-passing distributed systems
  - Logical time and global system snapshots
  - Impossibility of consensus
  - Fault-tolerant broadcasts
- Partially synchronous message-passing distributed systems
  - Failure detectors

Distributed Algorithms

Bibliography

- "Distributed Algorithms", Nancy Lynch, Morgan Kaufmann, 1996.
- "Distributed Computing", Hagit Attiya and Jennifer Welch, McGraw-Hill, 1998.
- "Distributed Systems", S. Mullender (Ed.), 2nd ed., Addison-Wesley, 1993.
- "Concurrency Control and Recovery in Database Systems", Philip Bernstein, Vassos Hadzilacos, Nathan Goodman, Addison-Wesley, 1987.

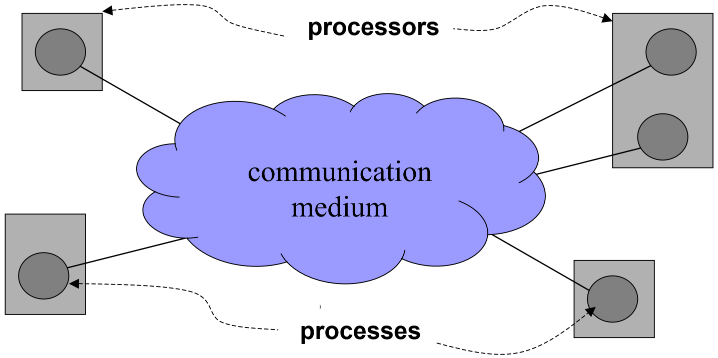
Distributed Systems

Distributed Systems:

- **provide the means for performance, scalability, dependability…**
  - loosely coupled computers, modular design
- **introduce special problems regarding correctness, complexity, failures…**
  - Lamport: *"A distributed system is one in which the failure of a computer you didn't even know existed can render your own computer unusable."*

Fault-Tolerance: the ability of a system to provide useful service (possibly degraded in functionality and/or performance), despite the fact that some of its components malfunction.

Models of Distributed Computing



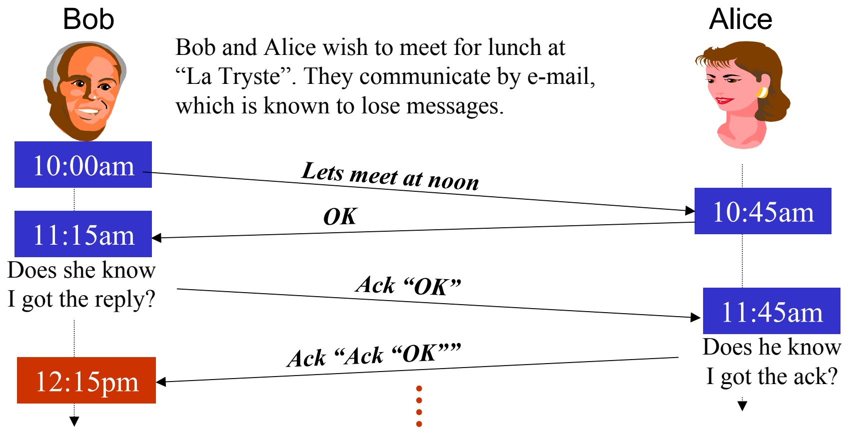
What kind of ***computational problems*** can one solve in a system?

Depends on the ***system model***:

execution and interaction timeliness, failure behaviour of software and hardware components, ...

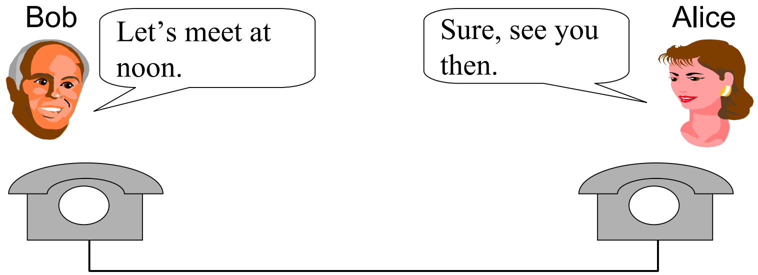
Models of Distributed Computing
Example: *Consenting adults*

Bob and Alice wish to meet for lunch at "La Tryste". They communicate by e-mail, which is known to lose messages.



System model:
- Unreliable message transmission
- No known bound on transmission time
- No commonly accessible medium (eg. Whiteboard)

Models of Distributed Computing
Example: *Consenting adults*

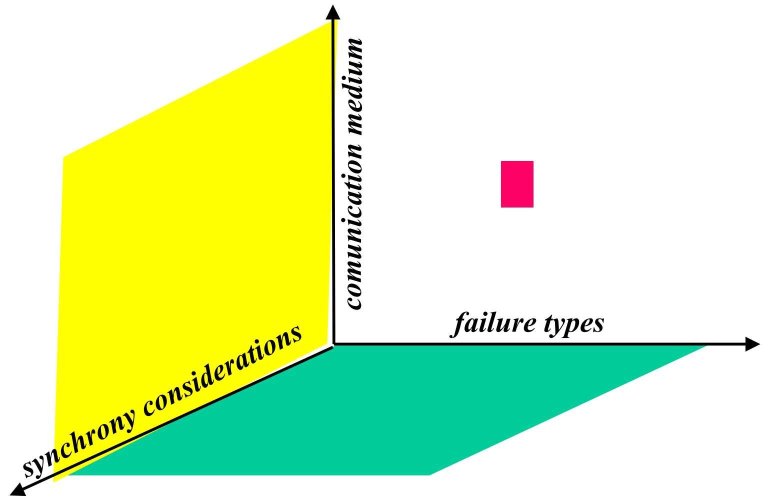


System model:
- one party hears what the other says within a bounded delay, or
- the existence of problem is known within a bounded delay

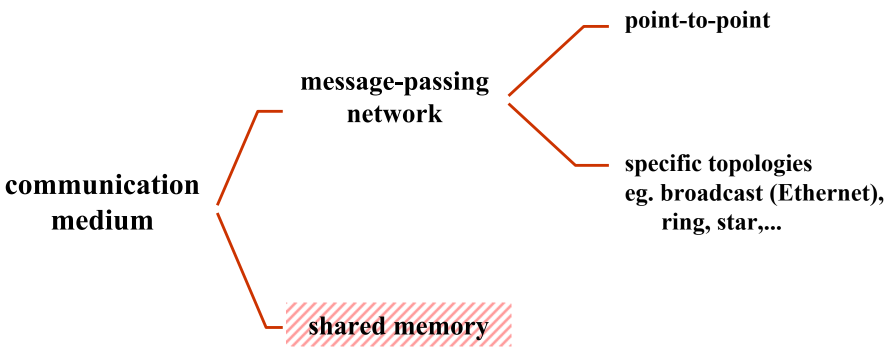
Ref: *"The Many Faces of Consensus in Distributed Systems", J. Turek and D. Shasha, IEEE Computer, June 1992.*

Models of Distributed Computing

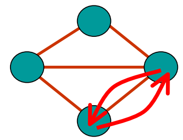
Three dimensions to consider ….



Models of Distributed Computing
Communication medium



Models of Distributed Computing
Point-to-point networks



point-to-point network : modelled as a graph

processes : graph nodes

communication links : graph edges (uni- or bi-directional)

Processes connected by a link communicate via send/recv primitives.

- send:**non-blocking**; necessary for fault-tolerance!
- Often assume **complete** communication graph.
- Links are **virtual**, not necessarily direct physical connections.

Models of Distributed Computing
Point-to-point networks

Properties of failure-free point-to-point networks

- ***Process specifications:***
  If a process has not reached a final state, eventually it will execute another **step**. (Liveness)

- ***Communication specifications:***
  - Process q receives message m from p **at most once** and only if p has **previously sent** m to q. (Safety)
  - If p sends m to q and q takes **infinitely many steps**, then q **eventually receives** m from p. (Liveness)

<u>**Note:**</u> In general, do not assume FIFO links; easy to implement, if needed.
Exercise: How?

Models of Distributed Computing
Types of failures

<u>Process failures</u>: "**crash**"
...a process stops taking steps before reaching a final state.

faulty process: violates *process specifications*

correct process: satisfies *process specifications*

<u>Link (communication) failures</u>: "**message loss**"
...a message sent from p to q is never received by q, even though q takes infinitely many steps.

faulty link: violates *liveness* of *communication specifications*

correct link: satisfies *liveness* of *communication specifications*

"Benign" failures - other types of failures introduced later on.

Models of Distributed Computing
Synchrony considerations

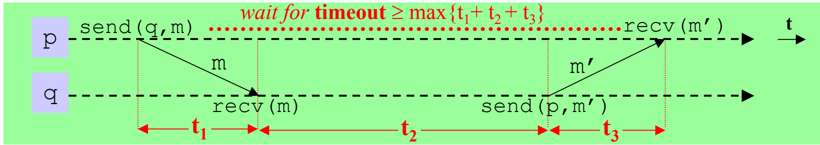
A. Synchronous network model:

- Known upper bound on time required for a process to execute a **local step**.
- Known upper bound on message **transmission delay**.
- Can assume that processes have perfectly synchronised physical clocks. In practice, when the two previous properties hold, approximately synchronised (with a known bounded drift $\varepsilon > 0$, from each other or from real time) clocks can be implemented -- they are more realistic; perfectly synchronised clocks are simpler for models.

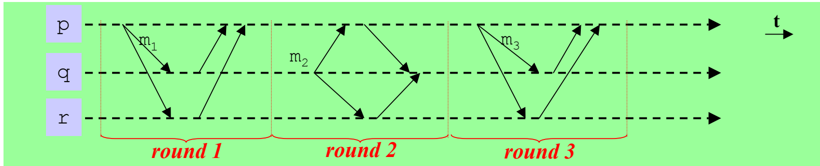
Models of Distributed Computing
Synchrony considerations

Consequences:

- can use **timeouts** to detect process or link failures



- can organise computation in **rounds**…
  - send messages to a set of processes P
  - recv message of that round from all processes in P
  - change state



Models of Distributed Computing
Synchrony considerations

B. Asynchronous network model:

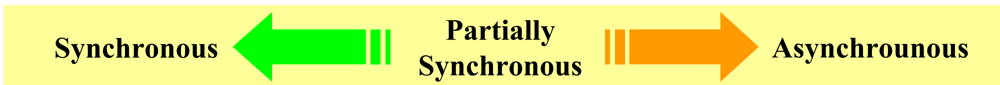
- No bound on time to execute a local process step; however, time to execute a local step is **finite**.
- No bound on message transmission delay.
- Cannot assume the existence of perfectly or approximately synchronised physical clocks (that measure real time).
  Note: may have logical clocks.

> the most general model -- an algorithm designed for asynchronous systems also works in synchronous systems.

Models of Distributed Computing
Synchrony considerations

Unfortunately,
some very basic computational problems **cannot** be solved in **asynchronous systems**…
✗ in a fault-tolerant manner (in the presence of failures)
and
✗ with a deterministic algorithm

Thus,
for certain problems we have to resort to…
✓ synchronous systems
or
✓ randomised (probabilistic) algorithms
 - not discussed in this course



Models of Distributed Computing
Summary

To describe a distributed system, must specify:

- **communication graph** (often: complete)
- **process failures** (e.g. crash failures)
- **link failures** (e.g. message loss)
- assumptions on the **number** (usually max) of process or link failures
- degree of **synchrony** for processes and communication

It is crucial to be clear and precise about these matters as they affect whether:

- an algorithm works in a given system
- a computational problem is solvable in a given system