FAULT TOLERANCE

Distributed Systems

4. Sem BSc Informatics

IMC FH Krems

LECTURE OUTLINE

- Introduction
- Process resilience
- Reliable communication
- Recovery

fault -> Error -> Failure -> Total|Partial

INTRODUCTION

focus on replication but iwth different perspective

- Failures in centralized systems are **total**: the application stops working at all.
- Failures in distributed systems can be **partial**: a part of the application experiences a failure.
  - The system can't provide all of its services to the user.
- An **error** is a part of the system's state that can lead to a **failure.**
- The cause of an error is called a **fault.**
- Types of faults:
  - **Transient:** occurrs only once and then disappears (i.e. beam interferences).
  - **Intermittent:** occurrs and reappears. Difficult to diagnose (i.e. faulty contact).
  - **Permanent:** occurrs and remains until faulty component is replaced (i.e. software bug).

INTRODUCTION

Failures in distributed systems (classification by behaviour)

1. **Crash failure**: The server halts execution after it worked correctly for a given amount of time.
2. **Omission failure** (receive/send omission): Server fails to receive or to send a message.
3. **Timing failure**: Server responds, but outside of some limited time interval.
4. **Response failure**: The response sent from the server is not correct.
   - Value incorrect.
   - Type of response incorrect.
5. **Arbitrary failure** (Byzantine failures). doesnt fit in the 4 previous types

INTRODUCTION

failyre free doesnt exist

- We can mask failures by using **redundancy.** process redundancy
- Types of redundancies:

1. Information redundancy: Add information to correct missing/incorrect bits.
   - Example: redundancy codes (Hamming codes).
2. Time redundancy: Actions are replayed if needed.
   - Example: retry request.
3. Physical redundancy: Add resources (hardware/software) to increase tolerance against failures of individual components.

LECTURE OUTLINE

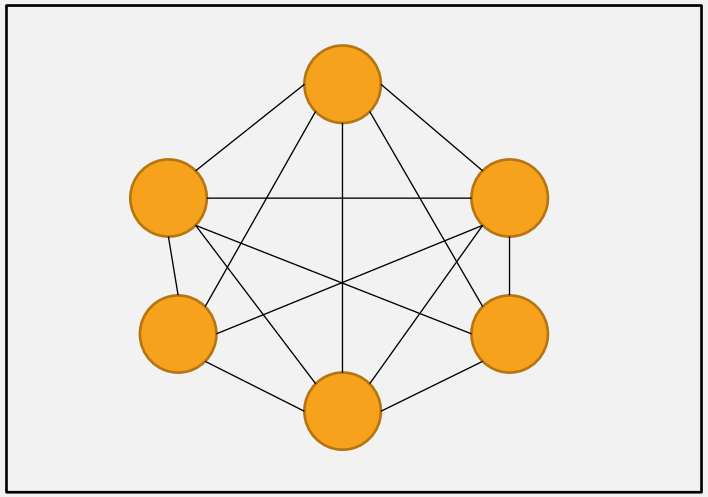
- Introduction
- **Process resilience**
- Reliable communication
- Recovery

PROCESS RESILIENCE

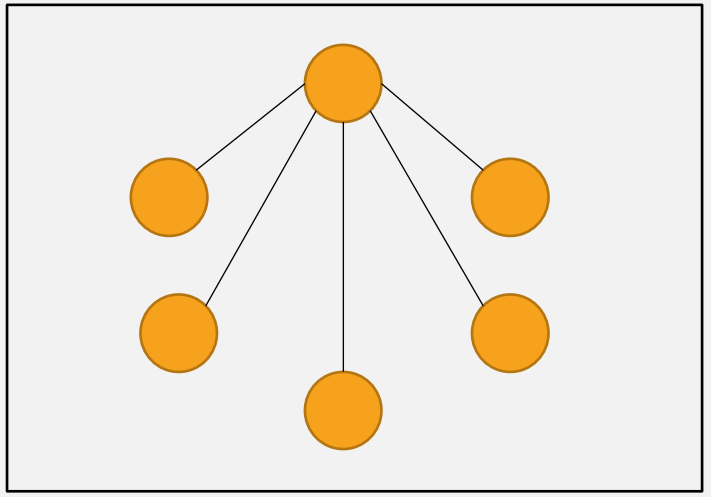
- How to achieve fault tolerance for processes?
- Key approach: substitute single processes by **process groups.**
- A process group appears to an external (client) process as a single process.
- Main idea: if a process fails, other processes can take over and mask the failure.
- All members of the group receive all messages sent to the group.
- Main challenges:
  1. How to organize groups?
  2. How to reach consensus?

PROCESS RESILIENCE

How to organize groups?



Flat group



Hierarchical group

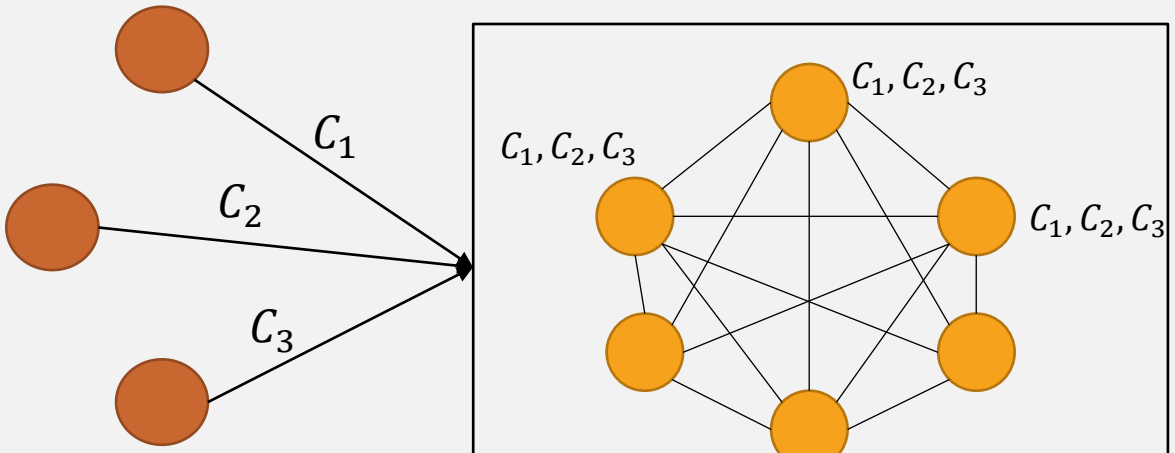
advantages, disadvantages of each???

PROCESS RESILIENCE

- Definition: we say that a system is ***k*-fault tolerant** if it can survive failures in k components and still meet its specifications.
- Q1: If processes fail silently (i.e. not responding), what is the minimum number of processes needed to support k-fault tolerance? k+1
- Q2: If processes exhibit arbitrary failures, what is the minimum number of processes needed to support k-fault tolerance? 2K+1

PROCESS RESILIENCE

How to reach consensus in process groups?



Client processes send requests (commands) to the process group

All processes in the process group have to agree on:

- Which command to execute next
- Order of commands to execute

→ **Consensus** algorithm needed

PROCESS RESILIENCE

Paxos algorithm

- Lamport, 1998.
- Most popular consensus algorithm.
- Key assumptions:
  - We assume that processes *eventually* reliably detect that another process has **crashed.**
  - Communication may be **unreliable**: messages may be lost, duplicated or reordered.
  - Processes do not exhibit **arbitrary** failures.
  - All operations are **deterministic.**

→ These assumptions are realistic for many systems.

how is ssystem suposed to behave when multiple services crash

PROCESS RESILIENCE

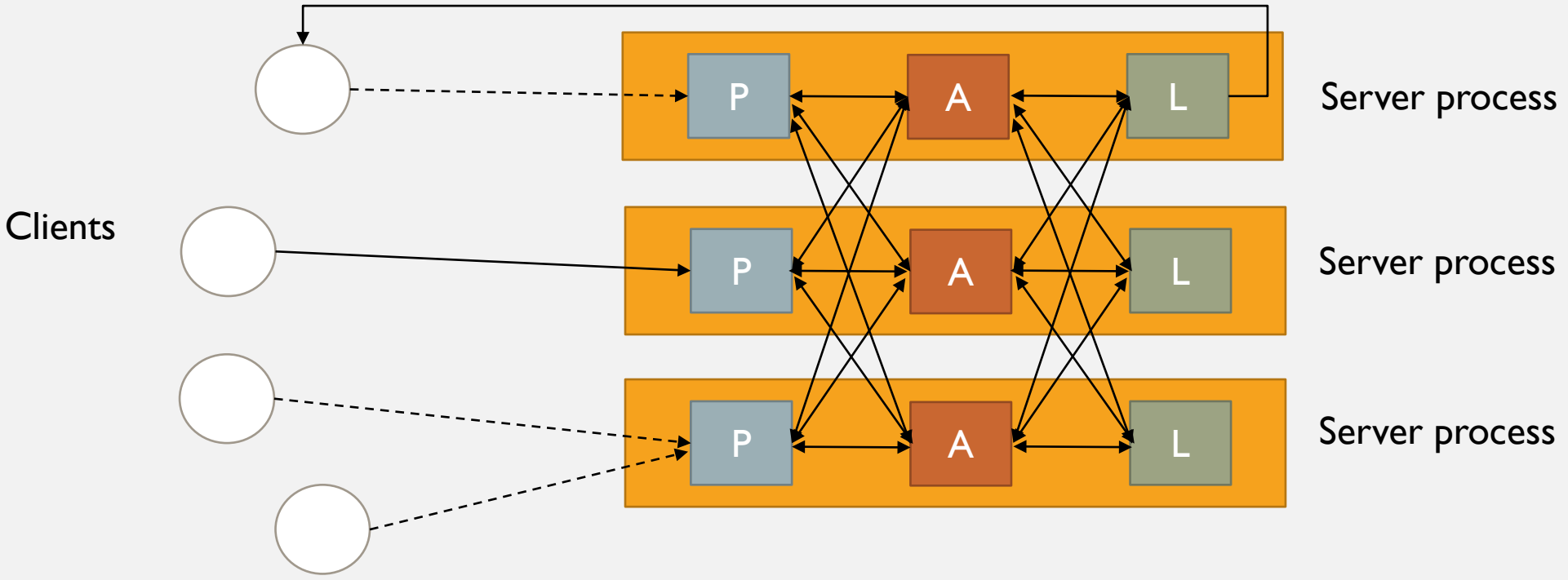
Paxos algorithm

- Processes are split into **proposers, acceptors** and **learners.**
- **Proposer:** Receives commands directly from the clients and attempts to have them executed. There might be an elected **leader** that forwards proposals to the acceptors. In this case, all proposers forward their proposals to the leader.
- **Acceptor:** Accepts proposals sent from the leader. A proposal is **chosen** if it is accepted by a majority of acceptors.
- **Learner:** Will execute a proposal (operation) once informed by a majority of acceptors.

PROCESS RESILIENCE

all the proposer talks with all acceptors
all acceptors communicate with all learners and proposers

Paxos algorithm



PROCESS RESILIENCE

Paxos algorithm

Goal: Reach consensus on which value to write.

Three phases:

Prepare	Accept	Commit
Establish the latest logical clock and gather accepted values	Propose a value to all replicas to accept	Communicate the accepted value to all replicas

PROCESS RESILIENCE

Paxos algorithm

Goal: Reach consensus on which value to write.

High-level overview:

1. **Prepare:** The **proposer** contacts all the nodes (**acceptors**) and asks them to promise to consider to accept a value.

   → If there is a **quorum** of acceptors, the proposer moves to the next phase.

2. **Accept:** The **proposer** sends out a proposed value.

   → If there is a **quorum** of acceptors, the value gets **chosen.**

3. **Commit:** The **proposer** commits the value to all other nodes.

PROCESS RESILIENCE

Paxos algorithm

1. Proposer asks if there is a pending proposal (value) and if not, it makes one
   a) **(prepare)** The proposer P sends a "prepare" message to all the acceptors, sending its current logical timestamp t. The proposer intends that
      (1) the acceptors do not accept any proposal with a lower timestamp and
      (2) the acceptors inform it of any accepted proposal with the highest timestamp less than t.

PROCESS RESILIENCE

Paxos algorithm

2. If a majority of acceptors agree, consensus was reached and the operation can be executed
   a) **(promise)** An acceptor A receives proposal from P with timestamp t. Either:
      i. Timestamp t is the highest so far. A sends a "promise" message to P stating that it will ignore any proposals with lower timestamp.
      ii. Timestamp t is the highest so far, but another proposal was accepted: A returns that proposal with a "promise" message to P.
      iii. Otherwise, do nothing.
   b) **(accept)** P waits until it receives "promise" from a majority of acceptors. Either (1) P did not receive any other proposal from the acceptors and therefore forwards its confirmation to all of them by means of an "accept" message, or (2) adopt the new proposal and forward its confirmation for acceptance with its own timestamp.

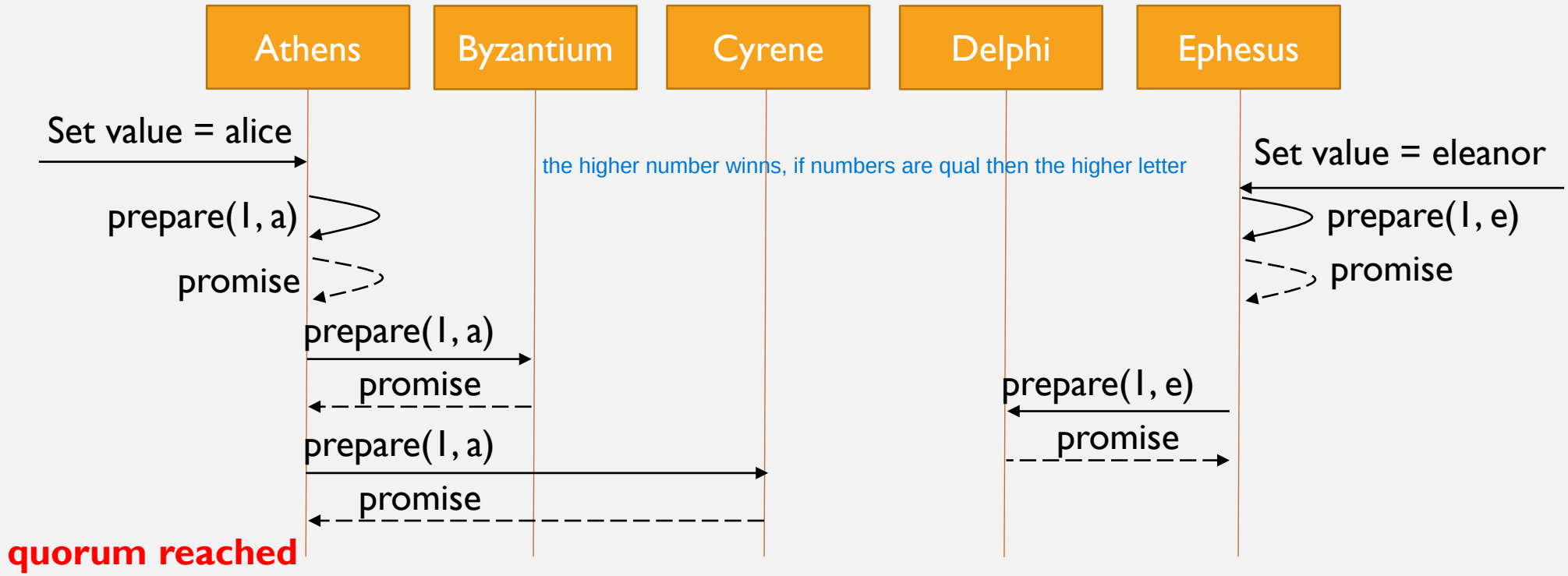
PROCESS RESILIENCE

Paxos algorithm

3. The chosen value is communicated to the other replicas
    a) **(commit/learn)** If the acceptor receives the "accept" message and did not make a promise with a higher timestamp, will tell all learners to execute the operation by sending a "learn" message. The acceptor is then finished with that operation. A learner receiving a learn command from a majority of acceptors will execute the operation.

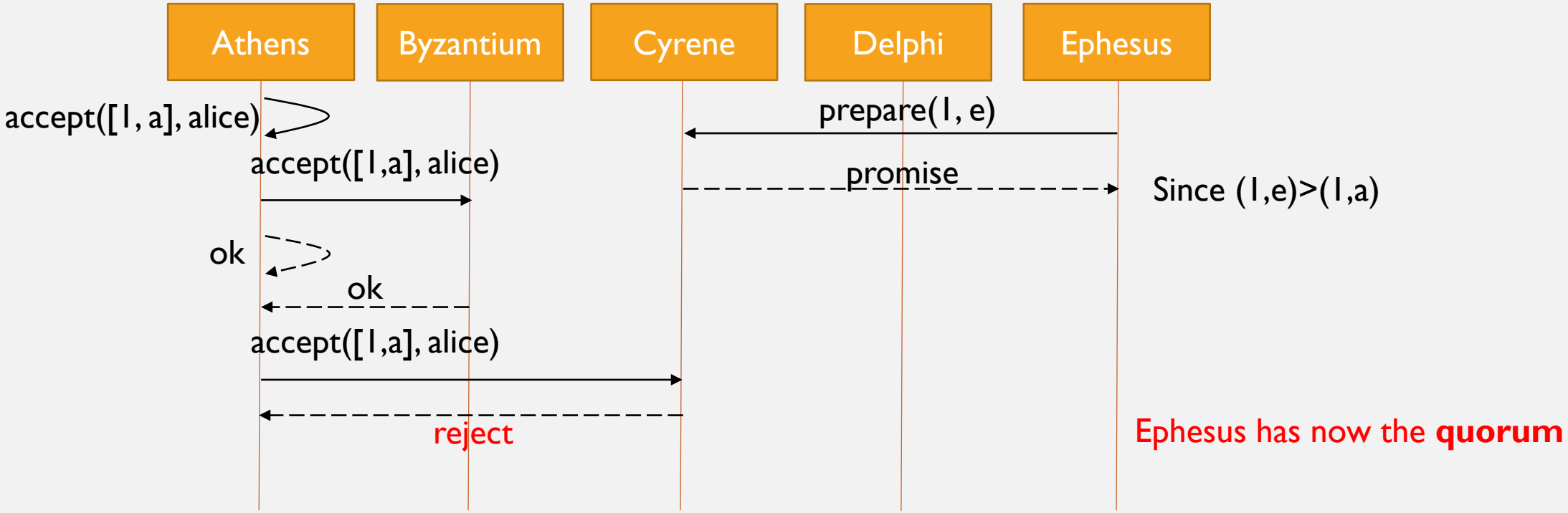
PROCESS RESILIENCE

Paxos algorithm – Example



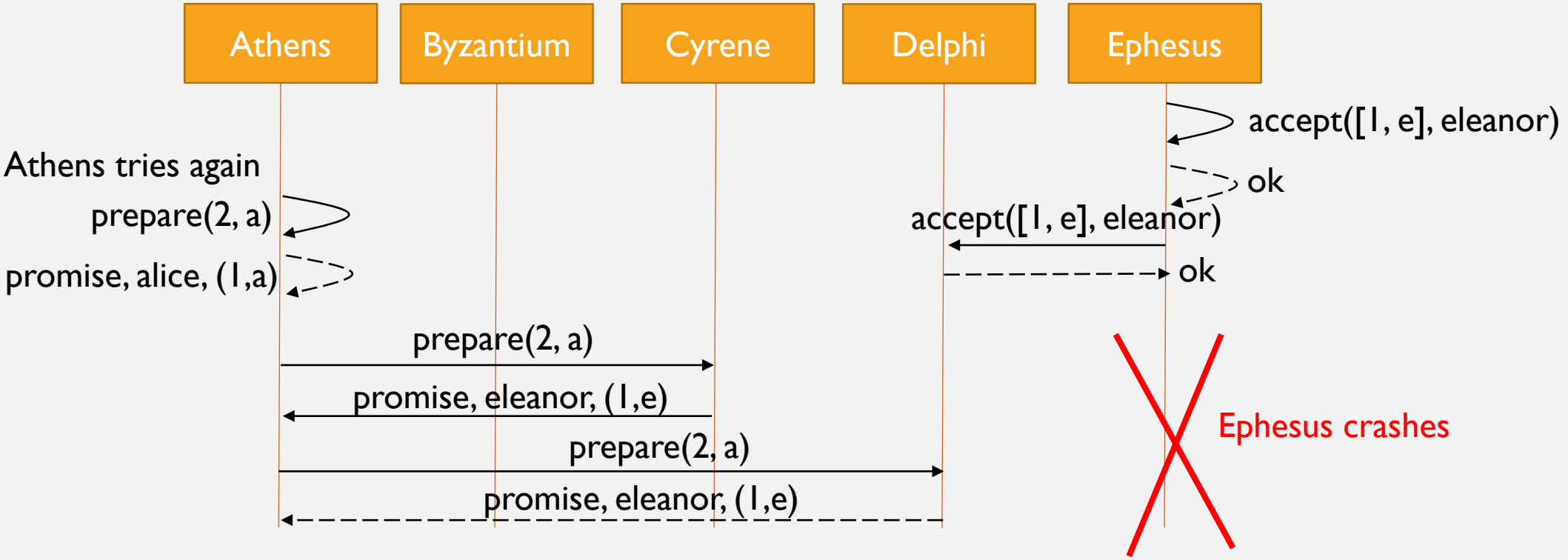
PROCESS RESILIENCE

Paxos algorithm – Example



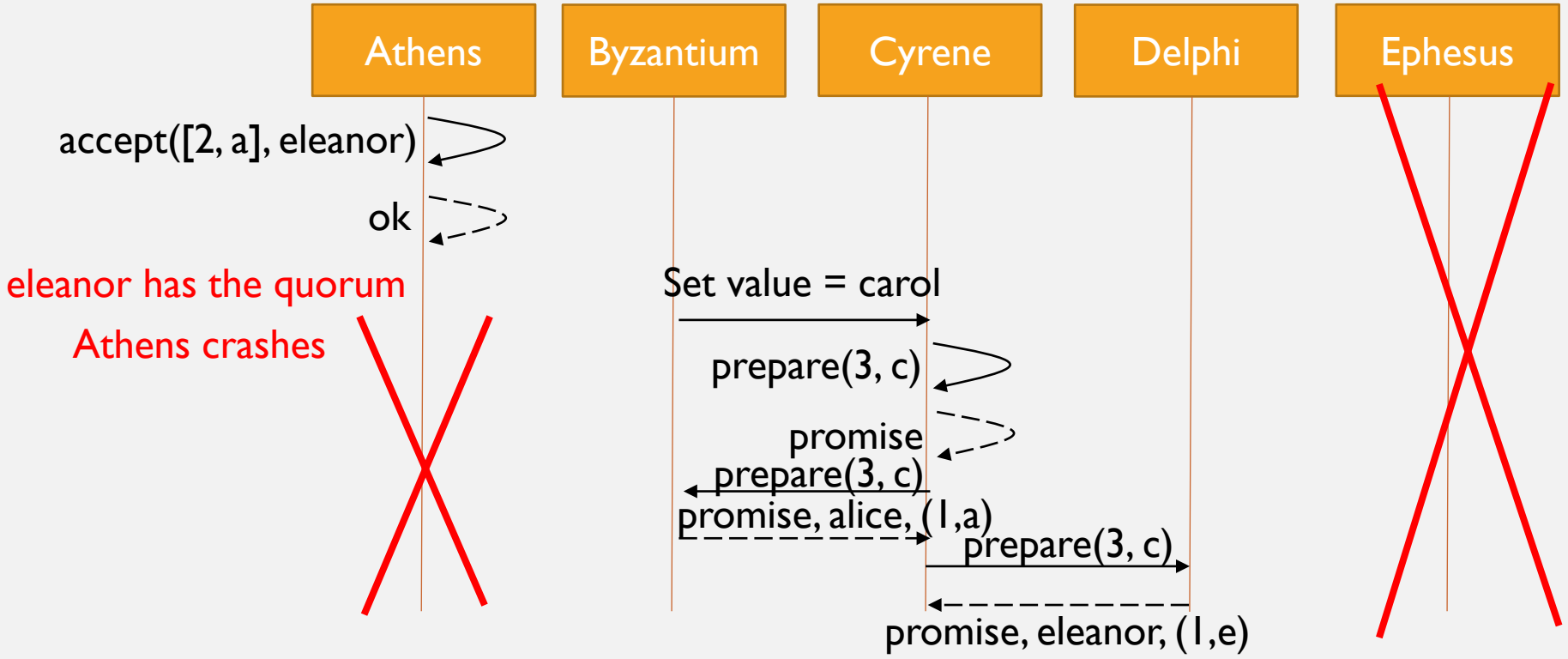
PROCESS RESILIENCE

Paxos algorithm – Example



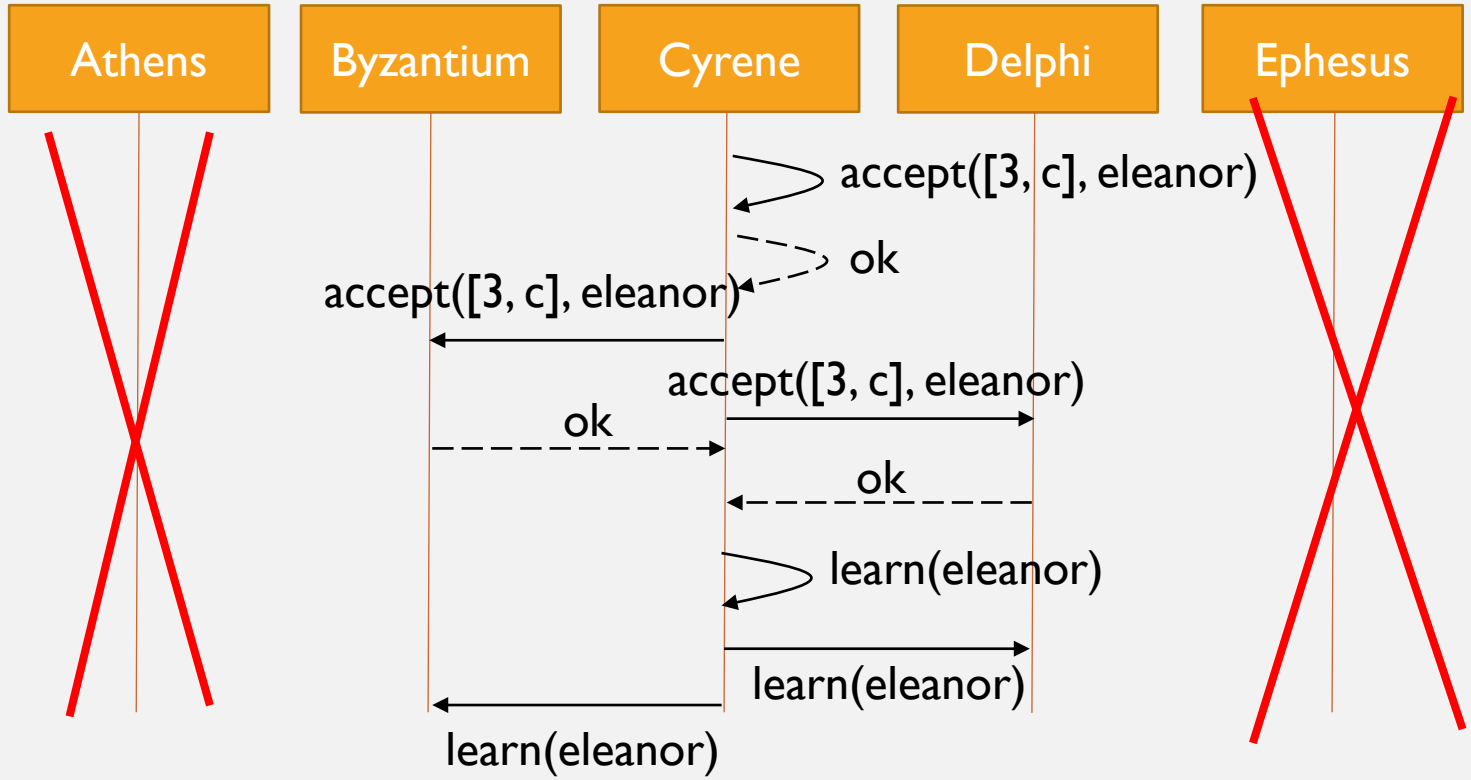
PROCESS RESILIENCE

Paxos algorithm – Example



PROCESS RESILIENCE

Paxos algorithm – Example



PROCESS RESILIENCE

Paxos algorithm

- What could possibly go wrong? Some examples

1. Acceptor fails during the promise phase
   - As long as proposer gets a reply from majority of acceptors, the algorithm can continue.
2. Proposer fails during the accept phase
   - Another proposer sends a “prepare“ message with a higher timestamp.
   - Acceptor responds that a promise was already accepted with a “promise“ message.
   - Proposer takes the value with the highest timestamp and finishes the job of the failed proposer (sending an “accept“ message).

PROCESS RESILIENCE

Paxos algorithm

- In practice, a sequence of consensus decisions has to be made (i.e. replicating a log). For that, we run Paxos multiple times (**multi-Paxos**).
- Byzantine failures are not covered (e.g. defense against malicious processes).
  - Variant of Paxos known as Byzantine Paxos.
- Production uses:
  - Google uses Paxos with Chubby and Bigtable.
  - Amazon Elastic Container Service (ECS).
  - Apache Cassandra.

LECTURE OUTLINE

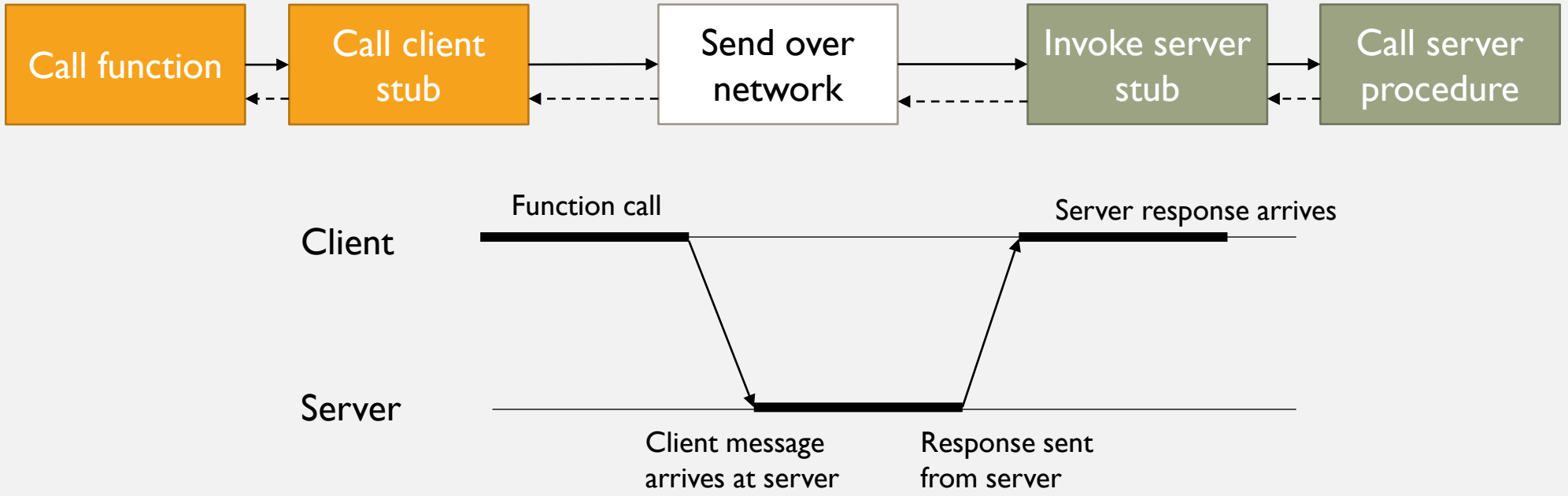
- Introduction
- Process resilience
- **Reliable communication**
- Recovery

RELIABLE COMMUNICATION

- How to handle communication failures?
- Communication channels can fail by omission, timing, crash and arbitrary failures.
- In point-to-point communication, a reliable transport protocol like TCP is used.
  - TCP can mask all failures except crashing.
  - When a crash (abrupt connection termination) is detected, a new connection is attempted.
- What types of failures can happen when using a **RPC protocol**?

REMOTE PROCEDURE CALL

Basic RPC operation



REMOTE PROCEDURE CALL

Steps involved in calling a procedure c = foo(a, b)

1. Client procedure calls the client stub
2. Client stub marshalls parameters (a, b) and builds network message
3. Network message is passed to client OS
4. Client OS sends message over the network
5. Message arrives at server OS
6. Server OS gives message to server stub
7. Server stub unmarshalls parameters and calls the server procedure
8. Server executes call and passes result to server stub
9. Server stub marshalls result (c) and builds network message
10. Server OS send message over the network to client OS
11. Client OS passes message to client stub
12. Client stub unmarshalls result (c) and passes it to client procedure

RPC MESSAGE

- Stubs can be generated automatically or manually by the developer.
- Serialized message: void exampleFunction(char x; float y; int z[5]).

- Passing arguments by reference:
  - Makes only sense locally.
  - Use global references (i.e. file system).
  - Otherwise: copy-by-value/restore.

RELIABLE COMMUNICATION

Failures in RPC communication

1. Cannot connect to server: no server available/no appropriate endpoint found.
   - Solution: raise and handle exceptions.
2. Lost requests or replies
   - After a timeout, request or reply might be retransmitted.
   - How to deal with repeated messages?
     a) Use **idempotent** requests: the result is the same regardless how many request we send → preferred way.
     b) If not possible, use sequence numbers for retransmissions: receiving end can detect that a message is a retransmission of a former one.

RELIABLE COMMUNICATION

Failures in RPC communication

3. Server crashes: several possible strategies
   a) At-least-one semantics: client keeps trying until a reply has been received. Drawback: operation may have been performed more than once.
   b) At-most-one semantics: client gives up immediately and does not retry. Request has possibly not been performed.
4. Client crashes: worst case if work is being performed on the server side and there is no client to return the result to → this is called orphan.
   - Orphan elimination: try to clean up orphans so that no resources are wasted.
   - Example: reincarnation → as soon as client boots again, it broadcasts a message so that all remote computations are reexamined. Computations without clients are killed.

LECTURE OUTLINE

- Introduction
- Process resilience
- Reliable communication
- **Recovery**

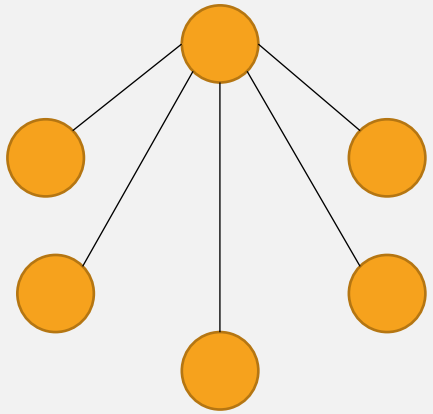
RECOVERY

- How to recover from failures to resume normal operation?
- Two fundamental approaches: **backward** and **forward** recovery.
- Backward recovery:
  - Bring the system from its current erroneous state to a previous correct state.
  - Main techniques are **checkpoints** and **message logging.**
  - **Chekpoint:** recording of the state of the system than can be replayed.
  - **Message logging:** sender or receiver logs all messages.
  - Best approach often to **combine** both techniques
    - For example: restoring an older checkpoint + replaying all messages from message logs until correct state is reached.

RECOVERY

Types of checkpoints

- **Coordinated checkpointing:** All processes synchronize to jointly write their state to local storage.
  - Needs coordinator process
  - Guarantees a global consistent state



1. Coordinator sends checkpoint request
2. Processes take local checkpoint and acknowledge to the coordinator

- **Independent checkpointing:** All processes record checkpoints independently from each other.
  - Calculation of recovery line (= order or restoring checkpoints) becomes very complicated.
  - Necessary to record dependencies between processes.

RECOVERY

- Forward recovery:
  - Attempt to move the system to a correct new state.
  - Example: try to reconstruct a lost message (bits) by means of a **error correction code**.
  - A message of k symbols is extended to a longer message with n symbols so that the original message can be recovered from a subset of those symbols.