ID2203

KTH-2026

Distributed Systems
Advanced Course

1. Introduction

Paris Carbone

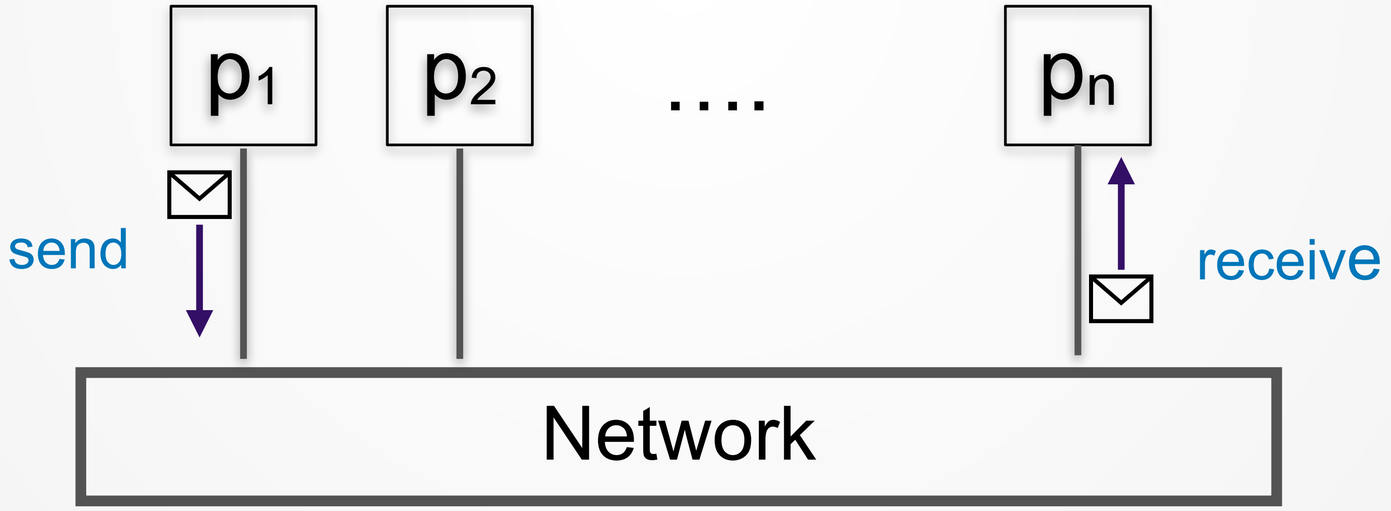
Course Topics

- Intro to Distributed Systems
- Fundamental Abstractions and Failure Detectors
- Reliable and Causal Order Broadcast
- Consensus (Paxos)
- Replicated State Machines (OmniPaxos, Raft, Zab etc.)
- Distributed Shared Memory & CRDTs
- Real-Time Abstractions (Spanner, Atomic, Quantum Clocks)
- Consistent Snapshotting (Data Management)
- Distributed ACID Transactions (Cloud DBs)

What is a distributed system?

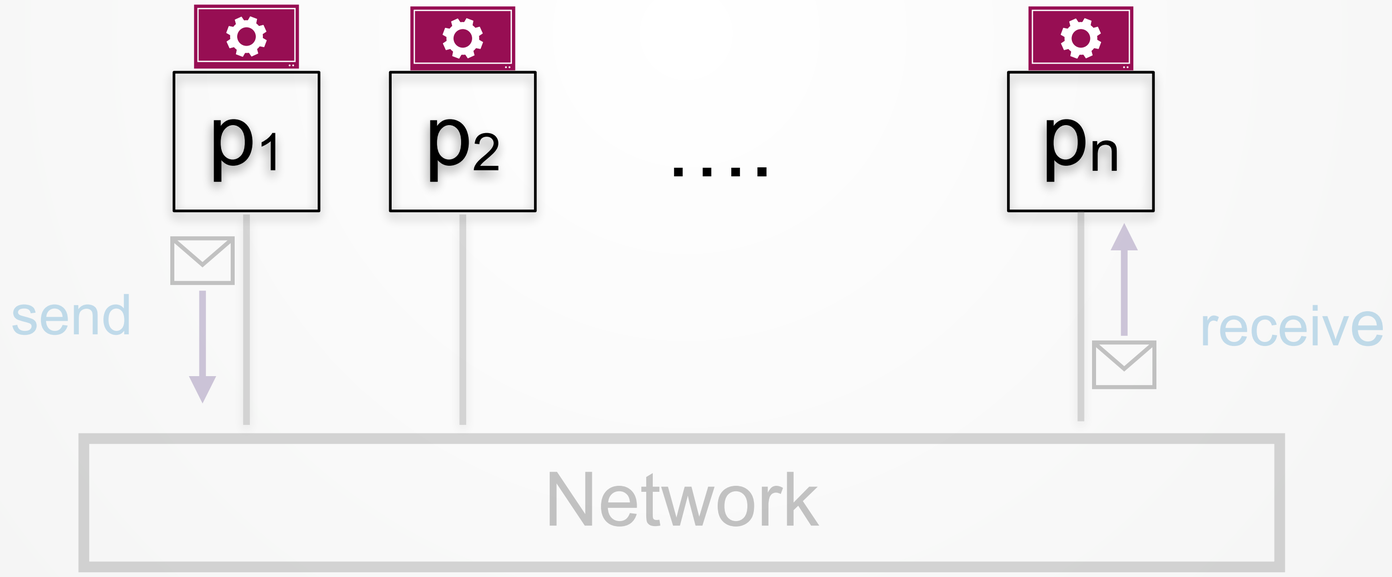
WHAT IS A DISTRIBUTED SYSTEM?

"A set of **nodes**, connected by a **network**, which appear to its users as a **single** coherent system"



What is a Distributed Algorithm

"A copy of a program running in each process"



Our focus in this course

- Concepts (Processes, Messages, Failures)
- Models (assumptions about system)
- Given the model…
  - Which problems are solvable / not solvable
  - What are the core problems in distributed systems
  - What are the algorithms
  - How to reason about correctness

WHY STUDY DISTRIBUTED SYSTEMS?

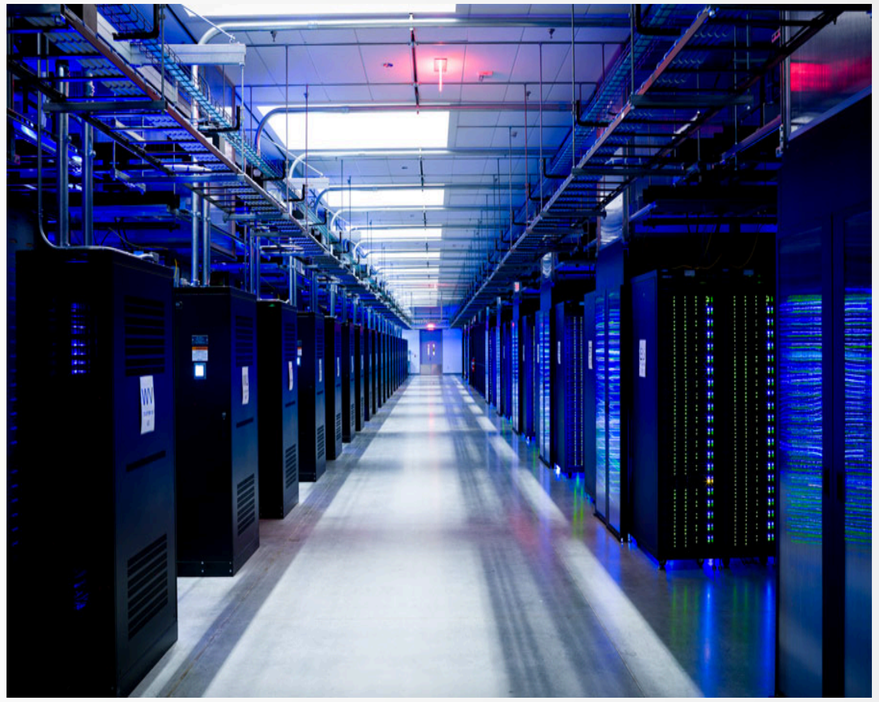
It is important, useful and interesting

Societal importance

Internet, WWW
Cloud computing
Edge - Small devices (mobiles, sensors)

Too Complex for AI

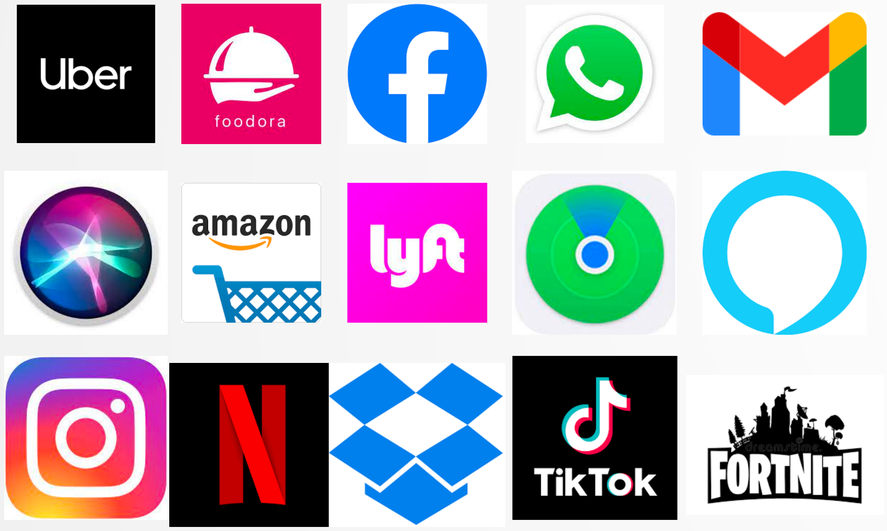
Protocol vs App logic
Rare Domain Expertise
High Neuro-Symbolic Complexity



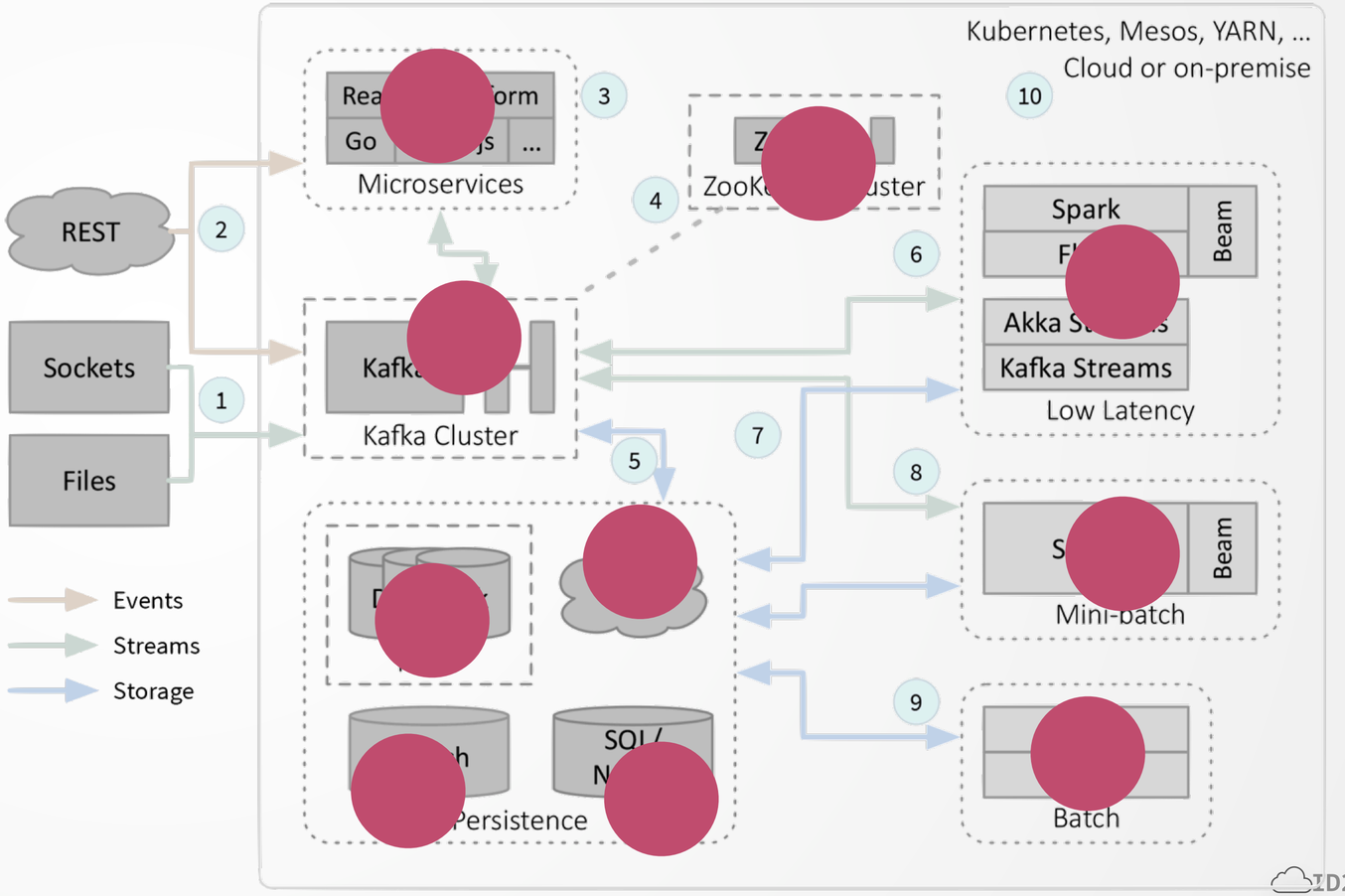
Disruptive Nature: P2P (00s) → Cloud (10s)→ Space (30s) → ?

Governing principles behind all our Data Services + Systems

DATA SERVICES : DISTRIBUTED SYSTEMS OF DISTRIBUTED SYSTEMS

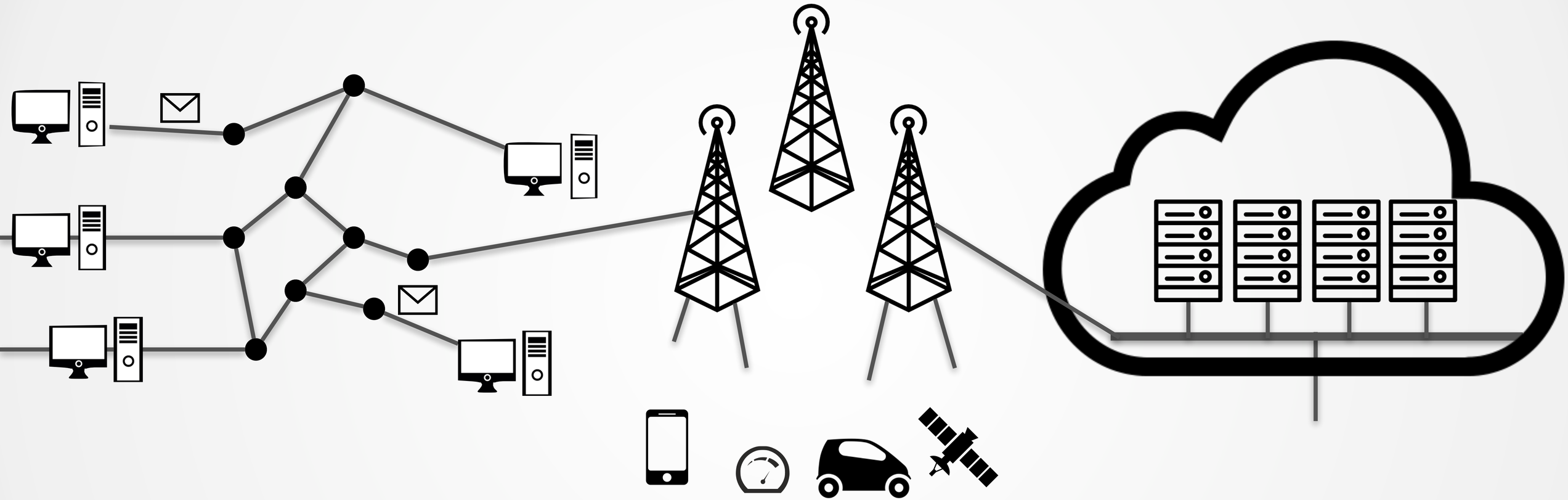


Each subsystem is a point of critical failure



Source: Portals Presentation: Carbone, Haller

WHY STUDY DISTRIBUTED SYSTEMS?

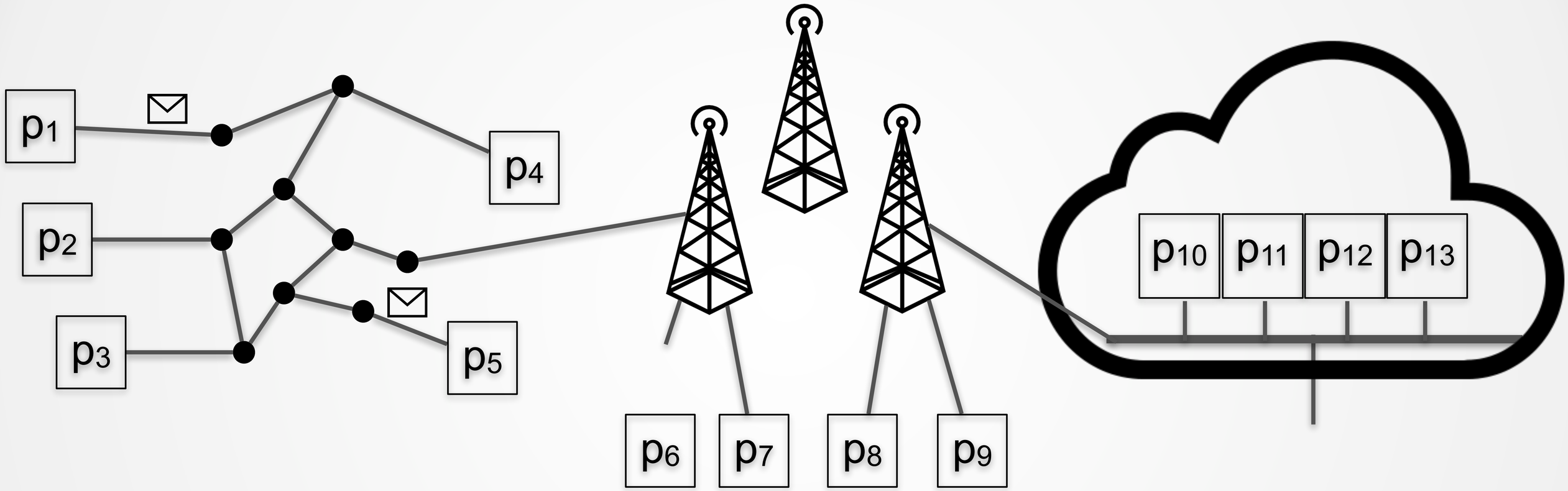


Internet

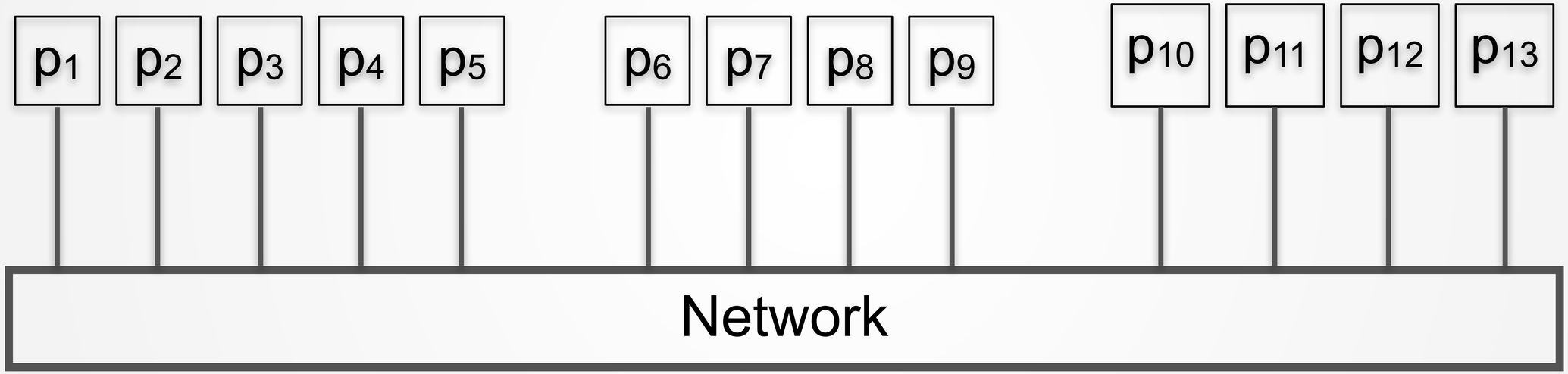
Edge Computing

Cloud Computing

WHY STUDY DISTRIBUTED SYSTEMS?



WHY STUDY DISTRIBUTED SYSTEMS?



WHY STUDY DISTRIBUTED SYSTEMS?

It is important and useful

- Technical importance
  - Improve scalability
  - Improve reliability
  - Inherent distribution

WHY STUDY DISTRIBUTED SYSTEMS?

It is very challenging!

Partial Failures

- Network (dropped messages, partitions)
- Node failures

Concurrency

- Nodes execute in parallel
- Messages travel asynchronously

} Parallel computing

Recurring core problems

Core Problems in Distributed Systems

What types of problems are there?

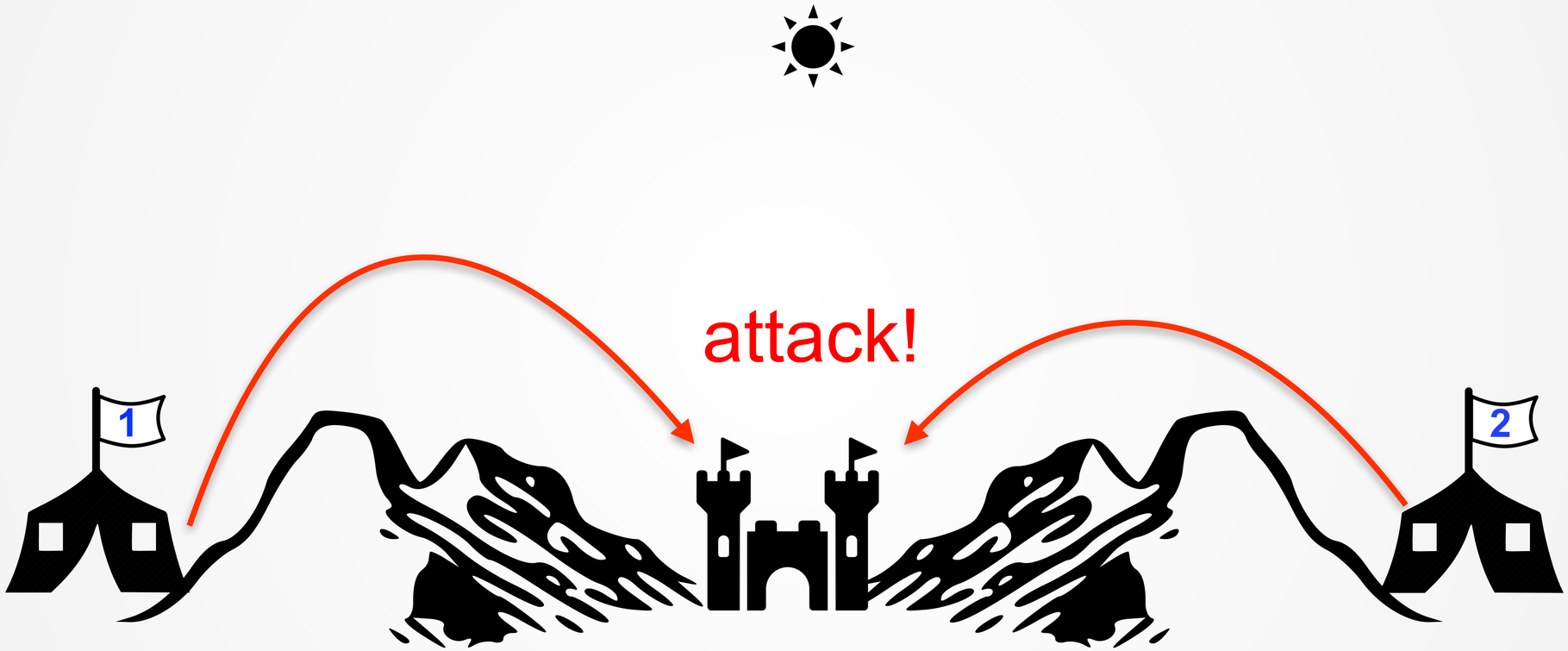
TEASER

“Two generals need to coordinate an attack”

- Must agree on time to attack
- They’ll win only if they attack simultaneously
- Communicate through messengers
- Messengers may be killed on their way

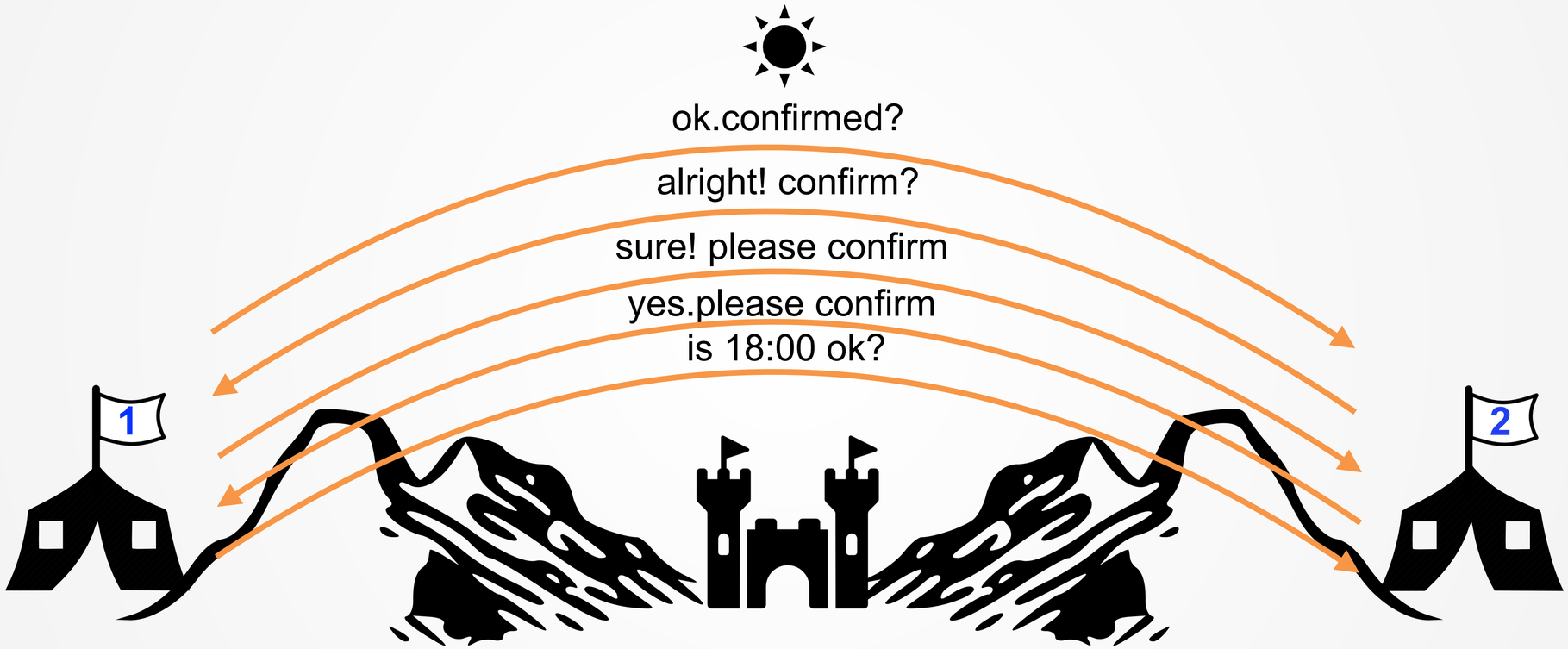
Teaser: Two Generals' Problem

Teaser: Two Generals' Problem

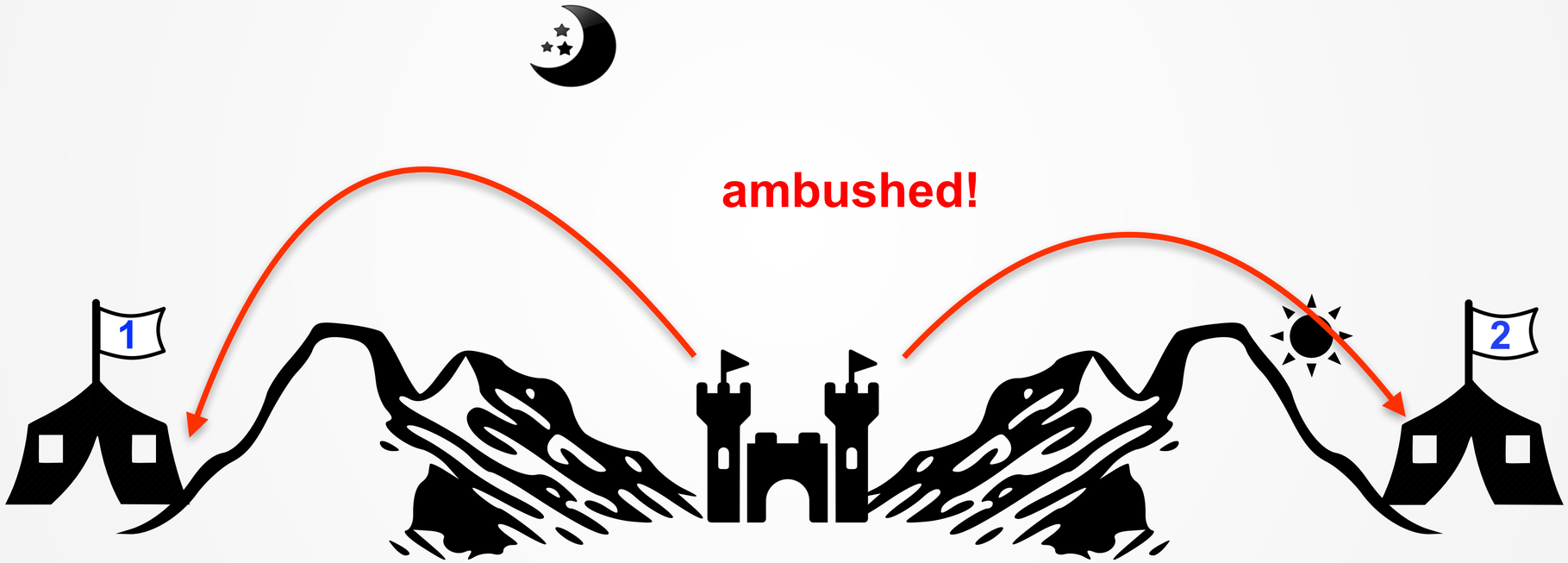


Teaser: Two Generals' Problem

Teaser: Two Generals' Problem



Teaser: Two Generals' Problem



Impossible to offer guaranteed solution!

Teaser: Two Generals' Problem

- Two processes need to agree on a value before a timeout
- Communicate by messages using unreliable network

Agreement is a core problem…

CONSENSUS: AGREEING ON A NUMBER

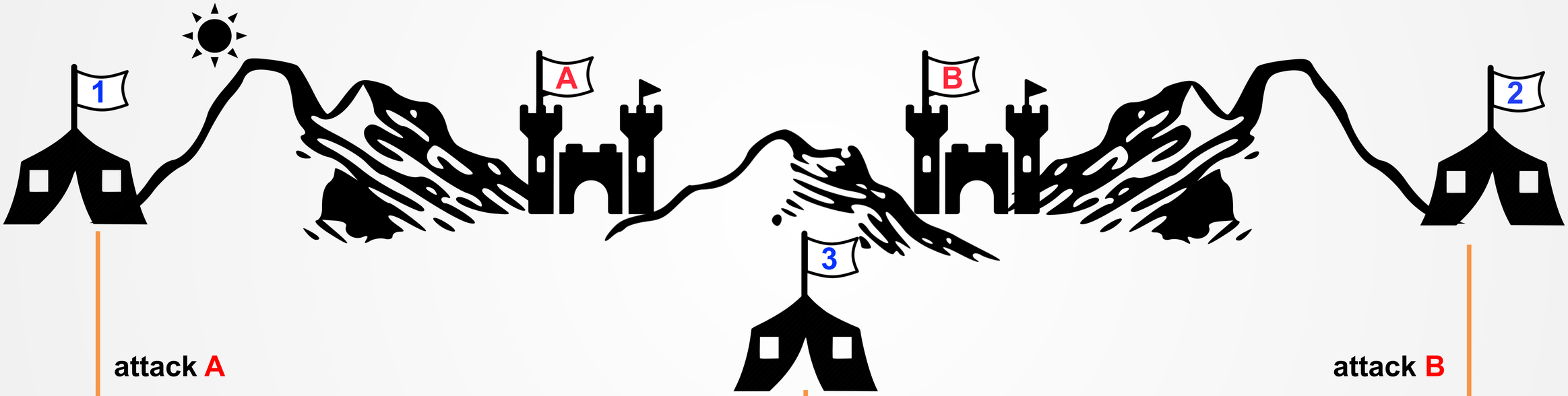
Consensus problem

All nodes/processes propose a value
Some nodes (non correct nodes) might crash & stop responding

The algorithm must ensure a set of properties (specification):

- All correct nodes eventually decide
- Every node decides the same
- Only decide on proposed values

Example: Agreeing on a Target



attack A

attack B

attack B

Consensus

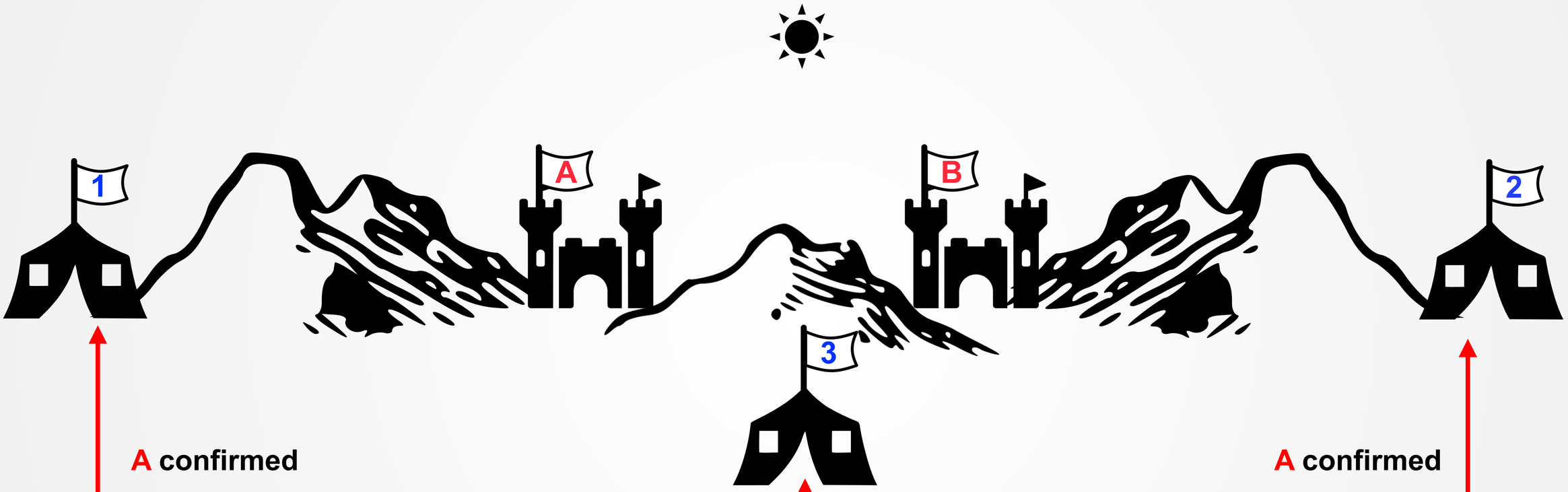
?

Consensus

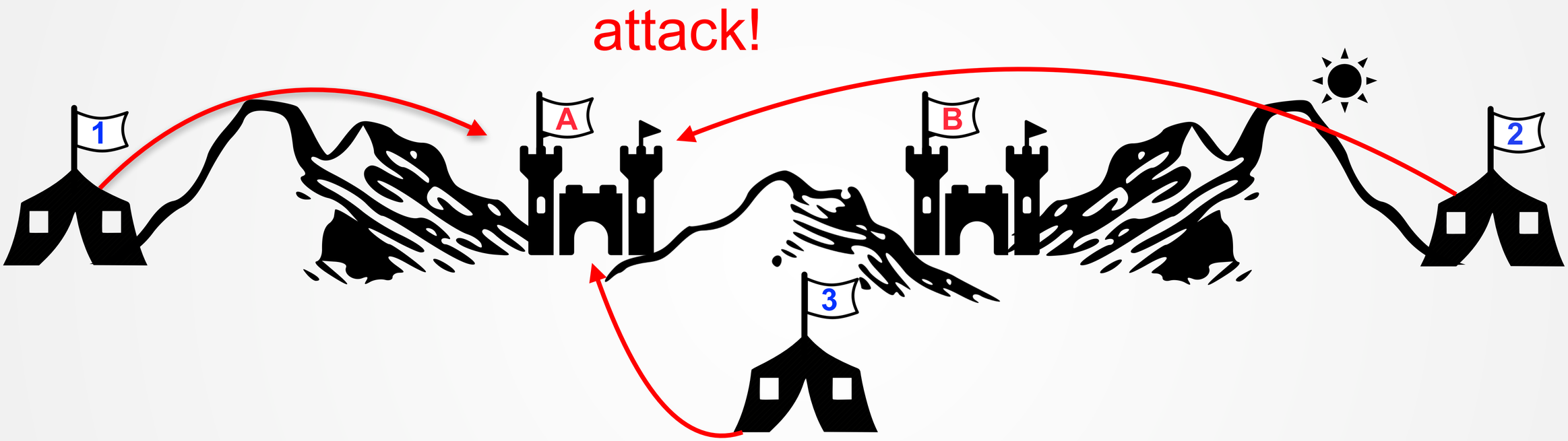
?

Consensus

Example: Agreeing on a Target



Example: Agreeing on a Target

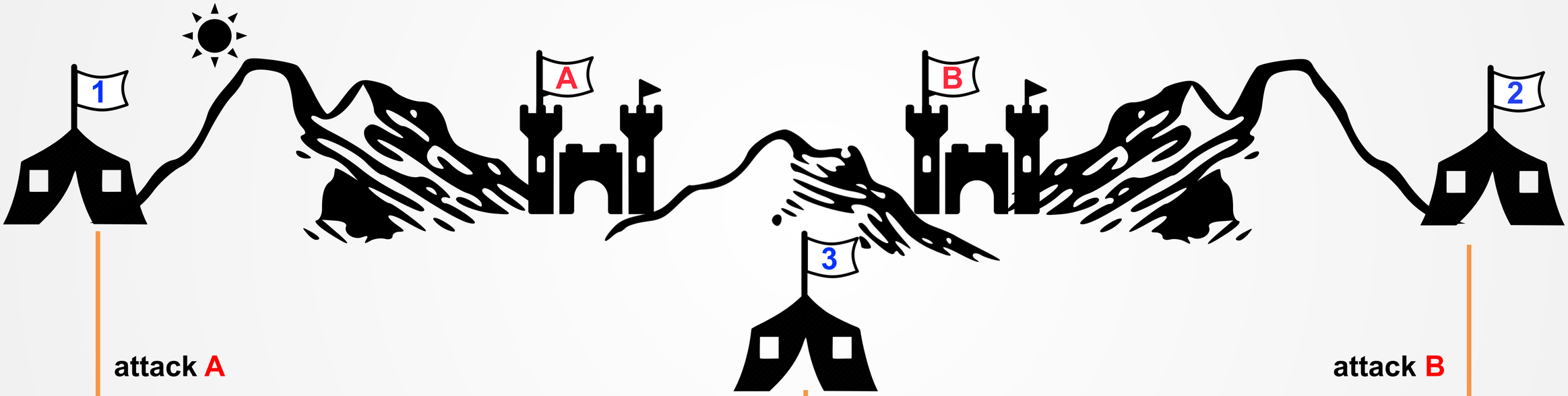


Consensus

Consensus

Consensus

Example: Agreeing on a Target



attack A

attack B

attack B

Consensus

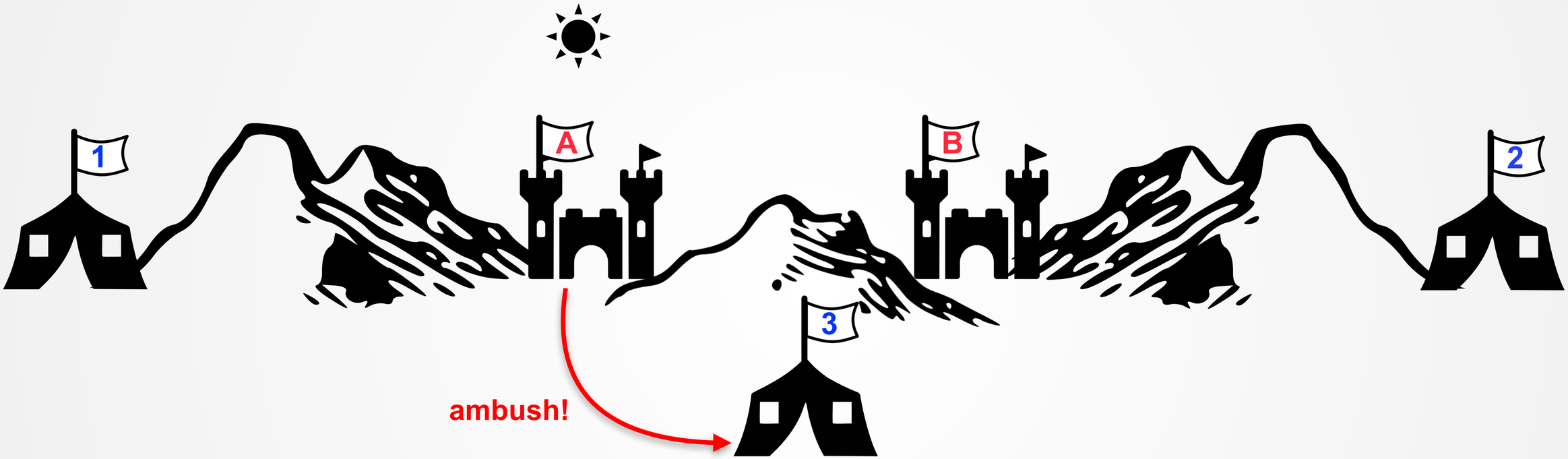
?

Consensus

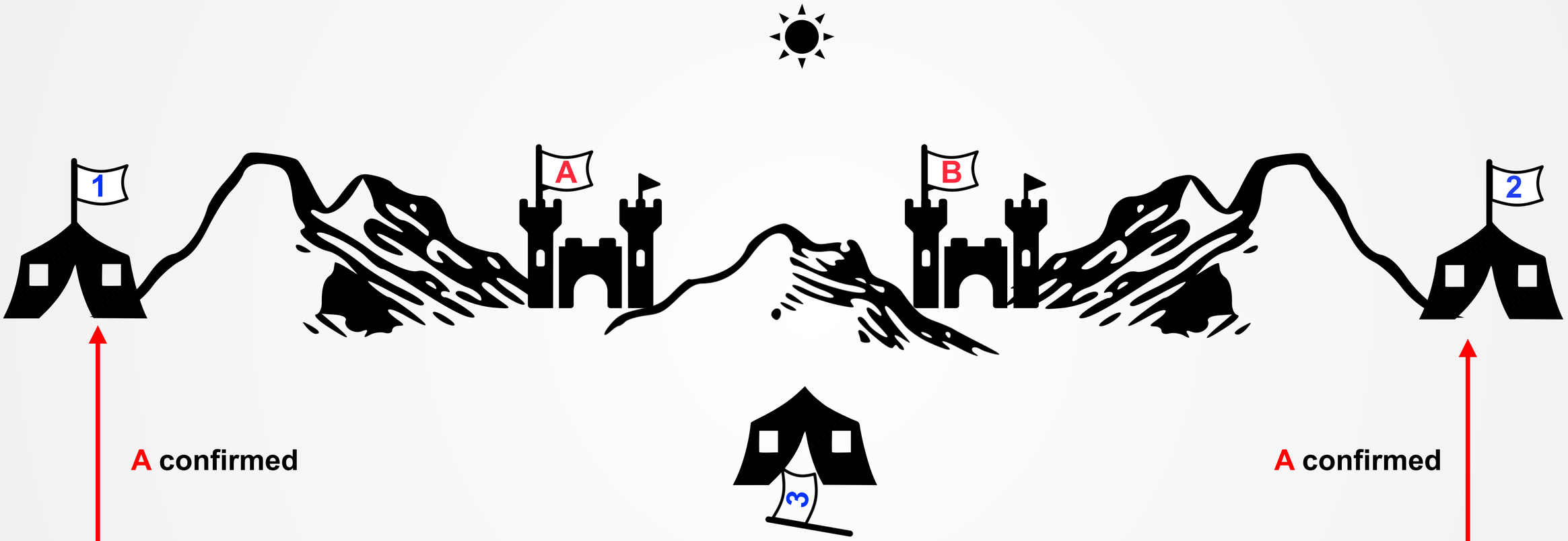
?

Consensus

Example: Agreeing on a Target



Example: Agreeing on a Target

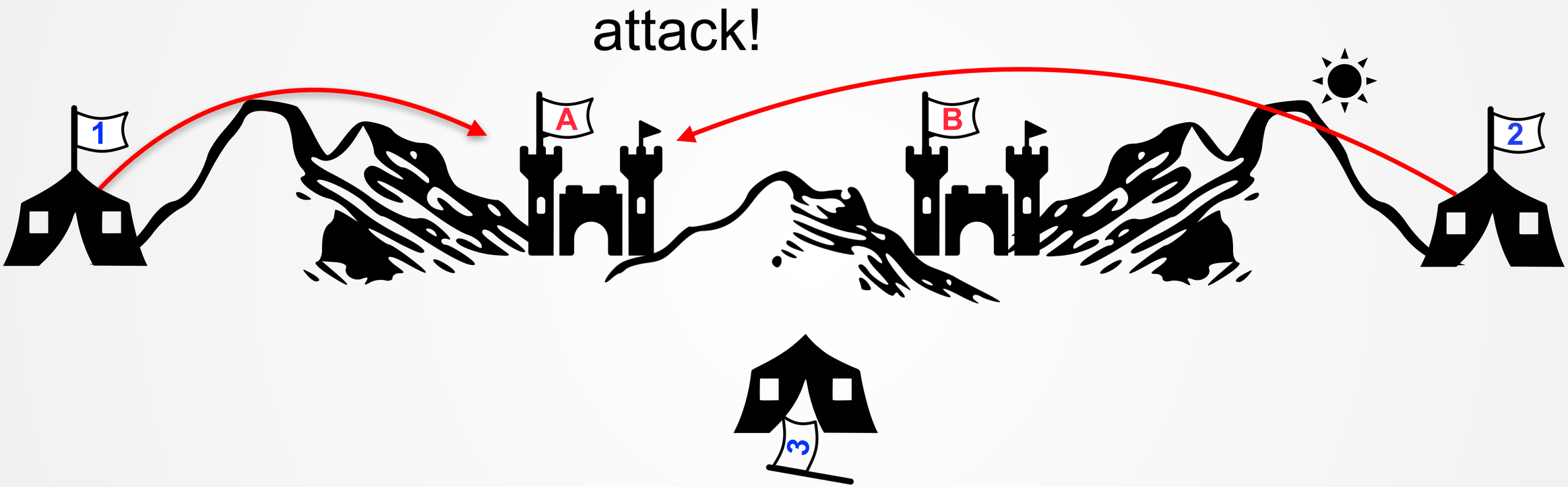


Consensus

Consensus

Consensus

Example: Agreeing on a Target



Consensus

Consensus

Consensus

Is Consensus Solvable?

Consensus problem

All nodes propose a value

Some nodes might crash & stop responding

The algorithm must ensure:

- All correct nodes eventually decide
- Every node decides the same
- Only decide on proposed values

Broadcast Problem

Atomic Broadcast

- A node broadcasts a message
- If sender correct, all correct nodes deliver msg
- All correct nodes deliver the same messages
- Messages delivered in the same order

Atomic broadcast is Important

Replicated services

- Multiple servers (processes)
- Execute the same **sequence** of commands
- Replicated State Machines RSM

Use atomic broadcast

Provide fault tolerance

Can we use atomic broadcast to solve consensus?

Atomic Broadcast ↔ Consensus

I. **Atomic broadcast** can be used to solve **Consensus**!

i.e., Every node broadcasts its proposal

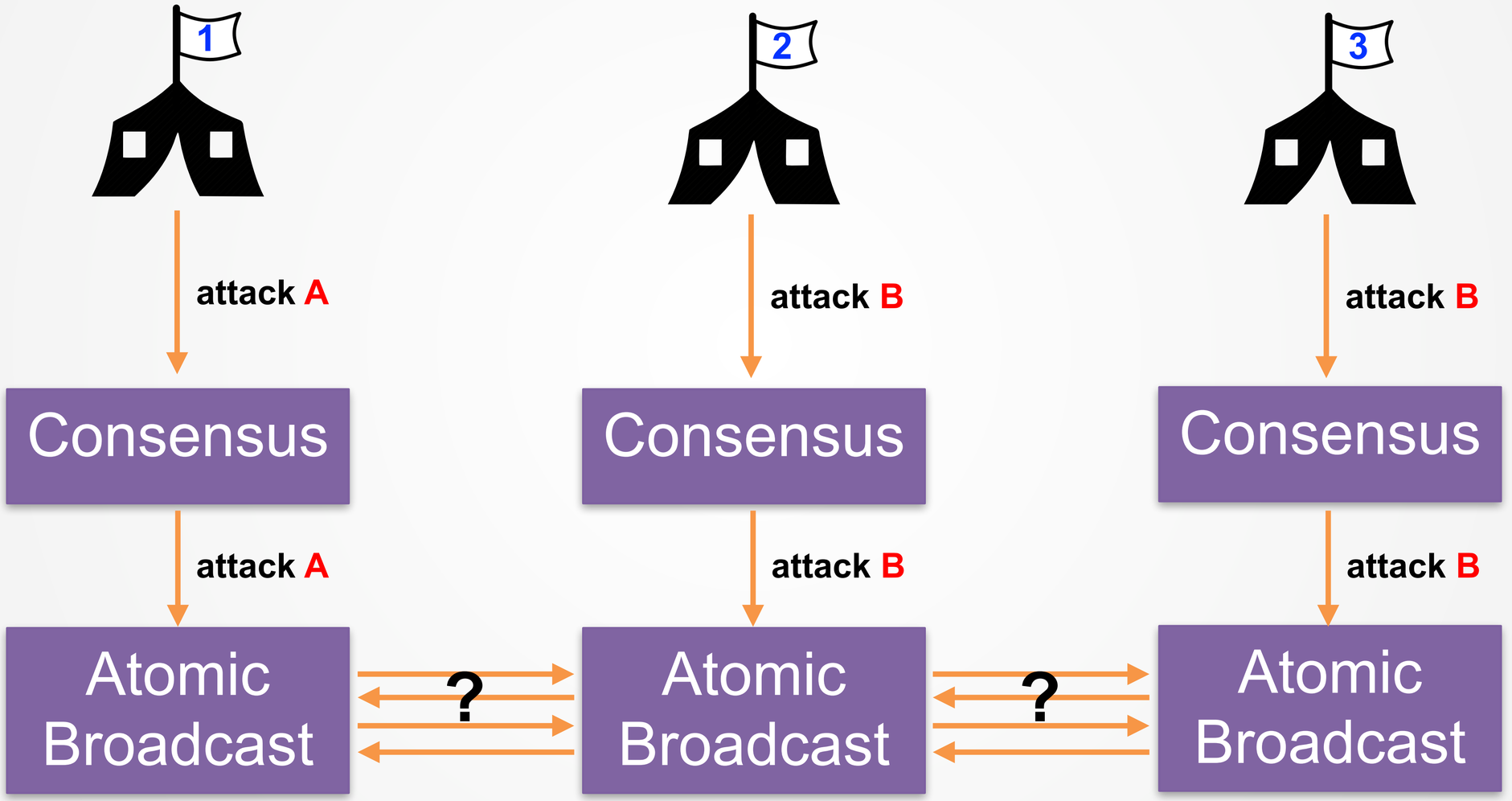
- Decide on the **first** received proposal
- Messages received in same order
  - Thus, all nodes will decide the same value.

II. **Consensus** can be used to solve **Atomic broadcast**

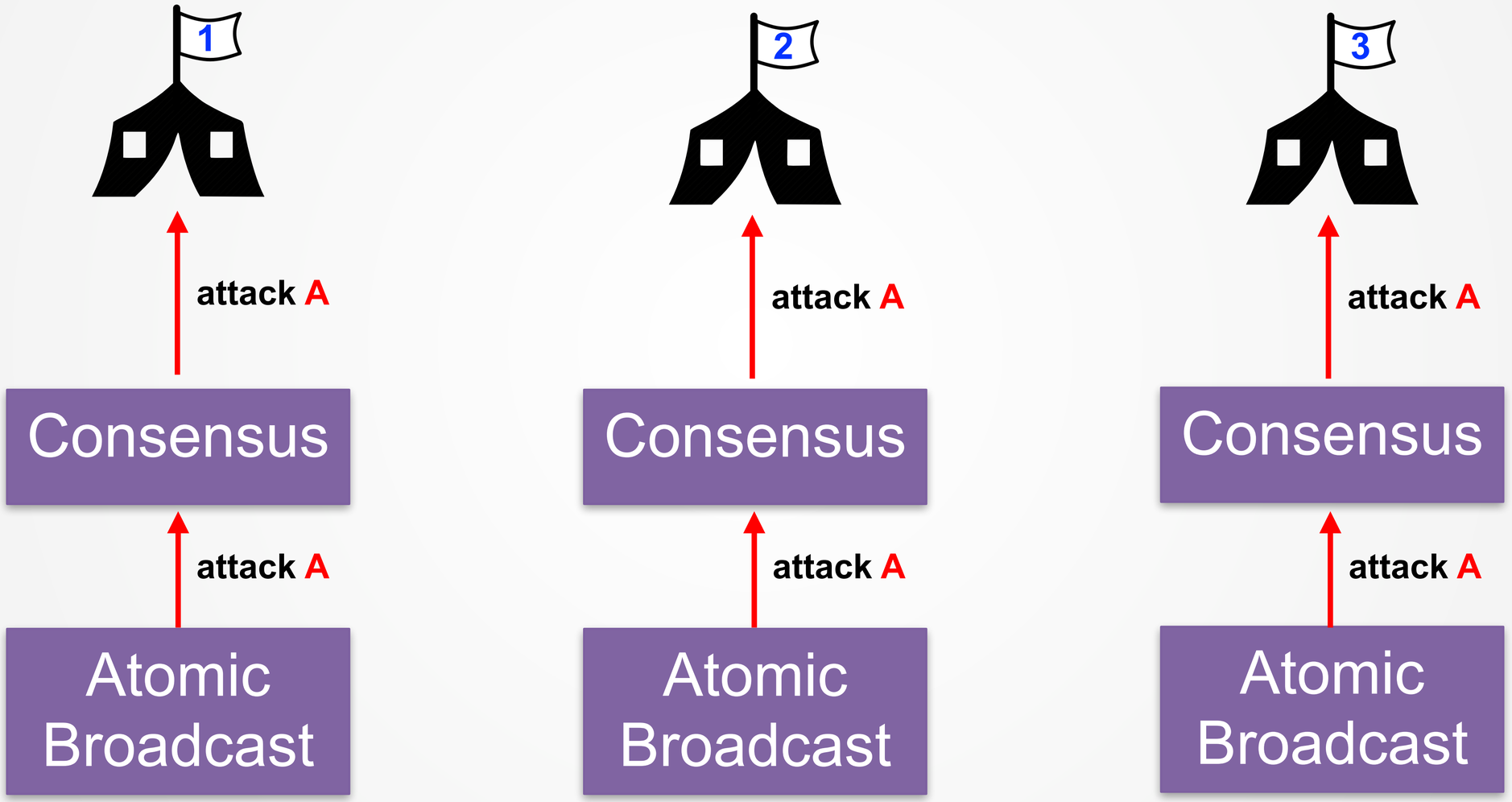
(more on that later in the course)

I+II: Atomic Broadcast **equivalent** to Consensus

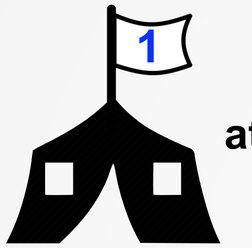
Atomic Broadcast ↔ Consensus



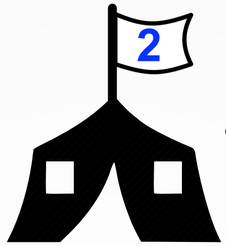
Atomic Broadcast ↔ Consensus



Atomic Broadcast ↔ Consensus

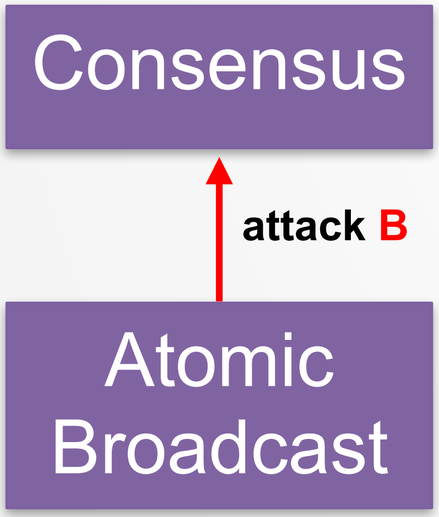


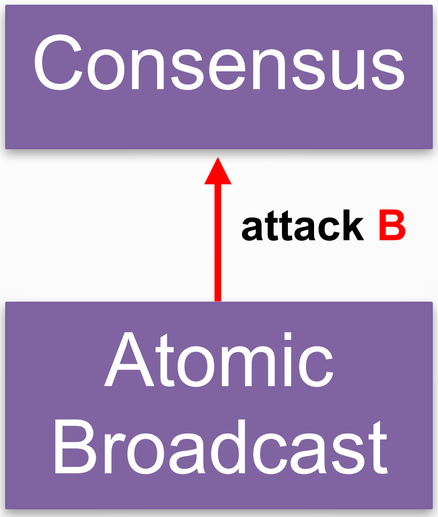
attacking **A**

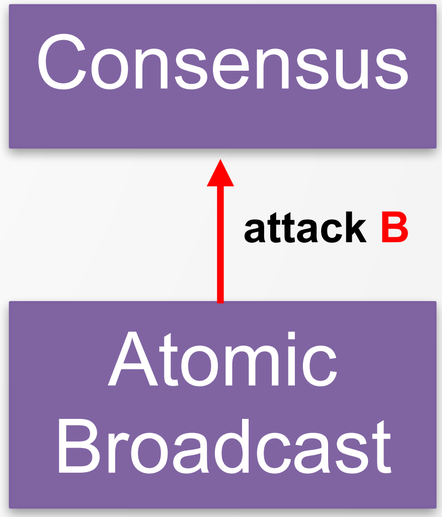


attacking **A**

attacking **A**







Models of Distributed Systems

How to reason about them?

Model = Assumptions

Assumptions → Timing + Failures

Modelling a Distributed System

Timing assumptions

Processes

- Bounds on time to make a computation step

Network

- Bounds on time to transmit a message between a sender and a receiver

Clocks

- Lower and upper bounds on clock drift rate

Modelling a Distributed System

Failure assumptions

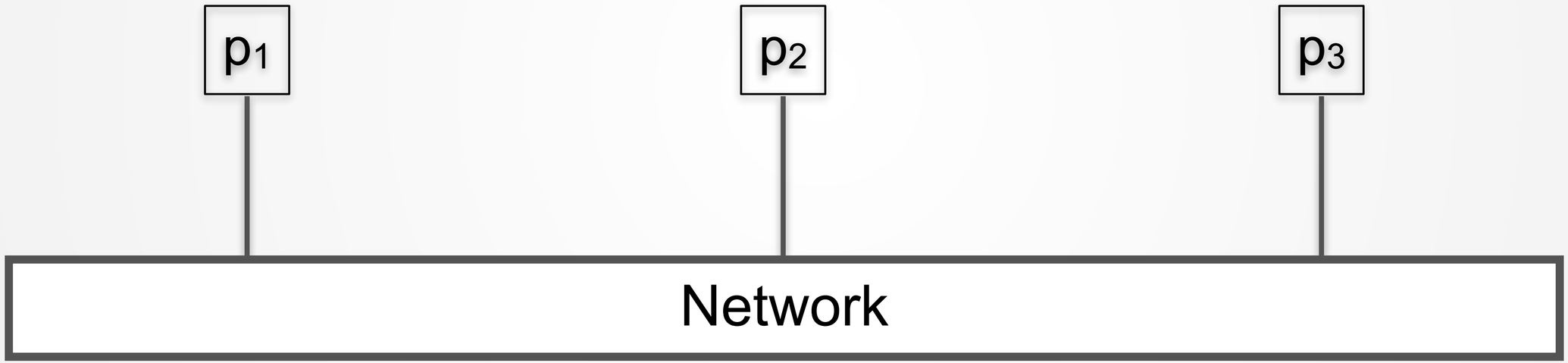
Processes

- What kind of failure can a process exhibit?
- Crashes and stops
- Behaves arbitrary (Byzantine)

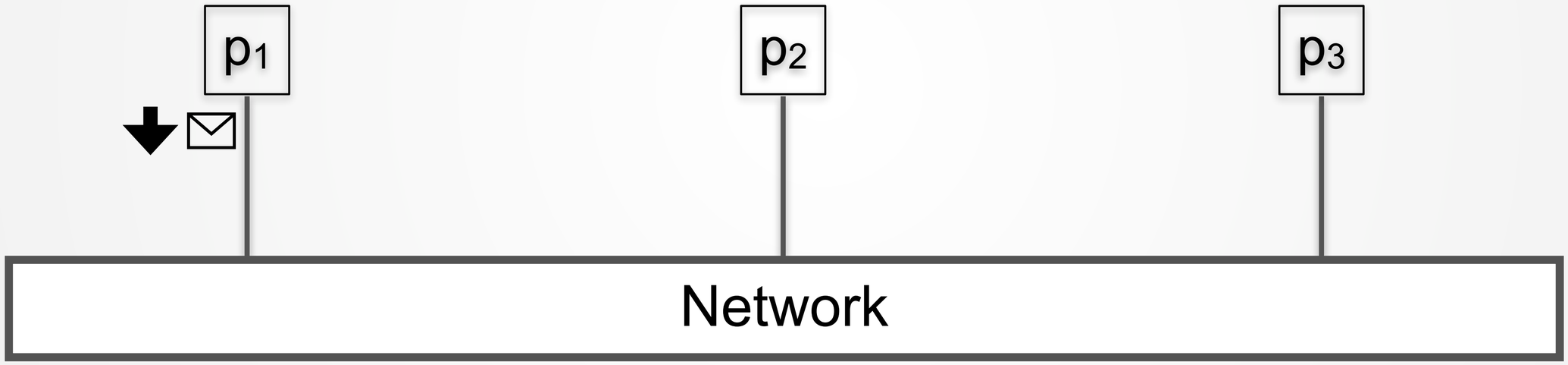
Network

- Can a network channel drop messages?
- Can certain channels temporarily disconnect? (partitions)

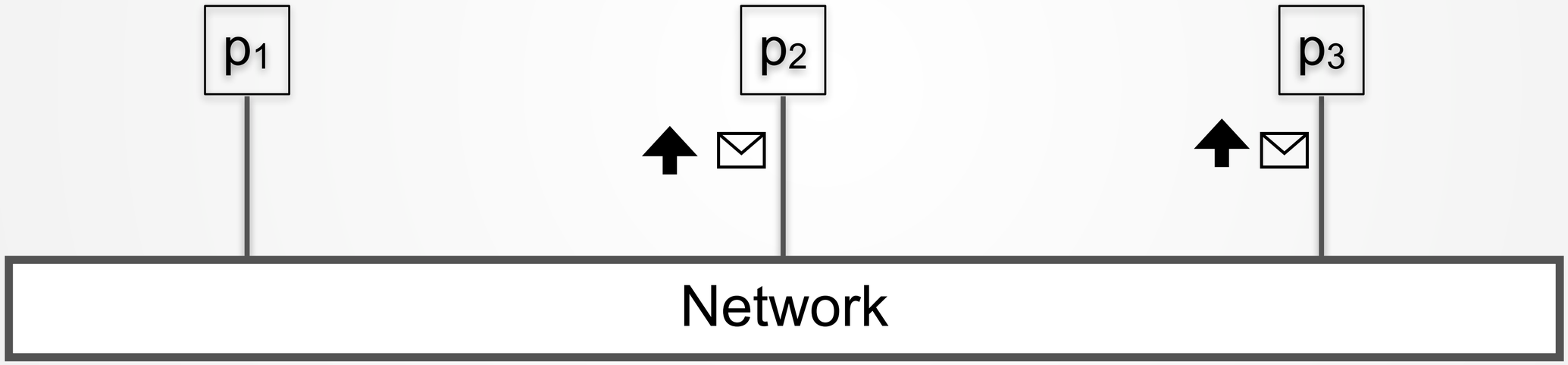
Modeling a Distributed System



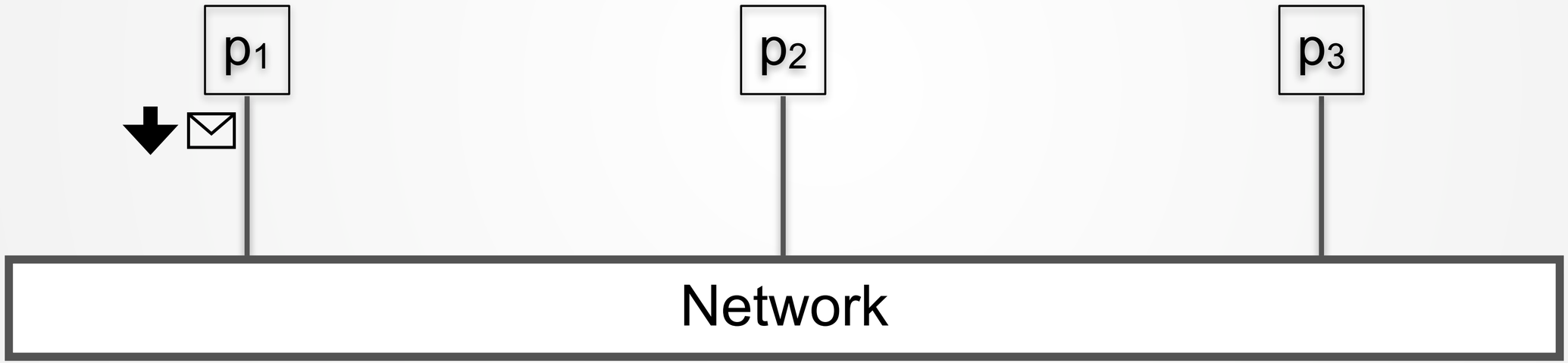
Modeling a Distributed System



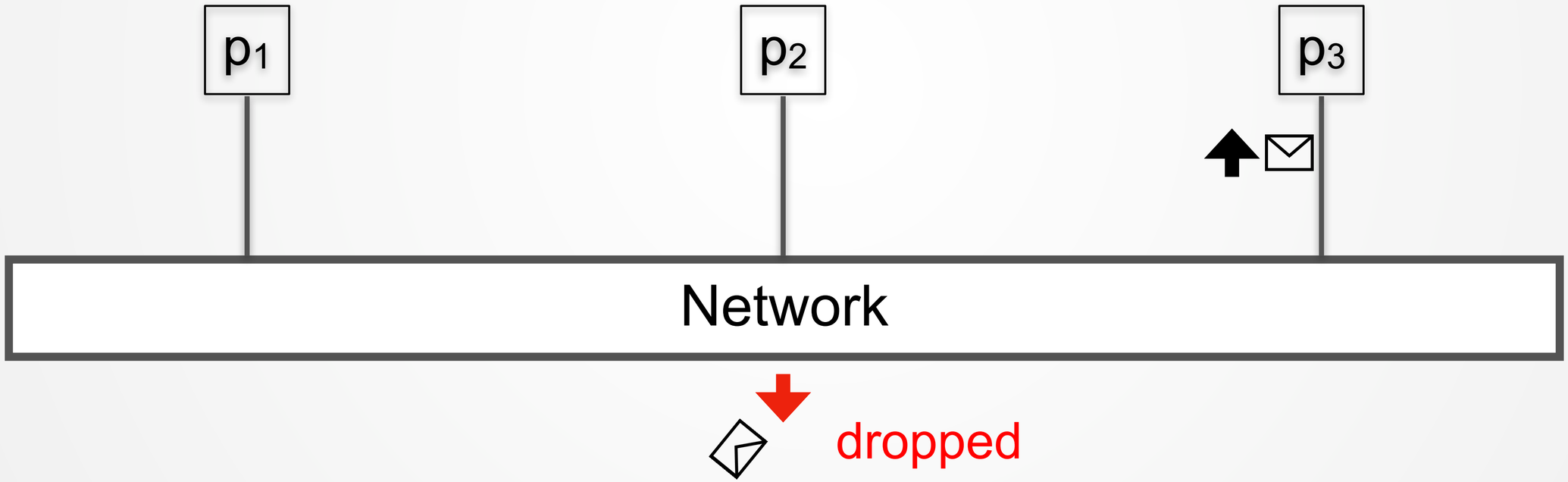
Modeling a Distributed System



Network Failures

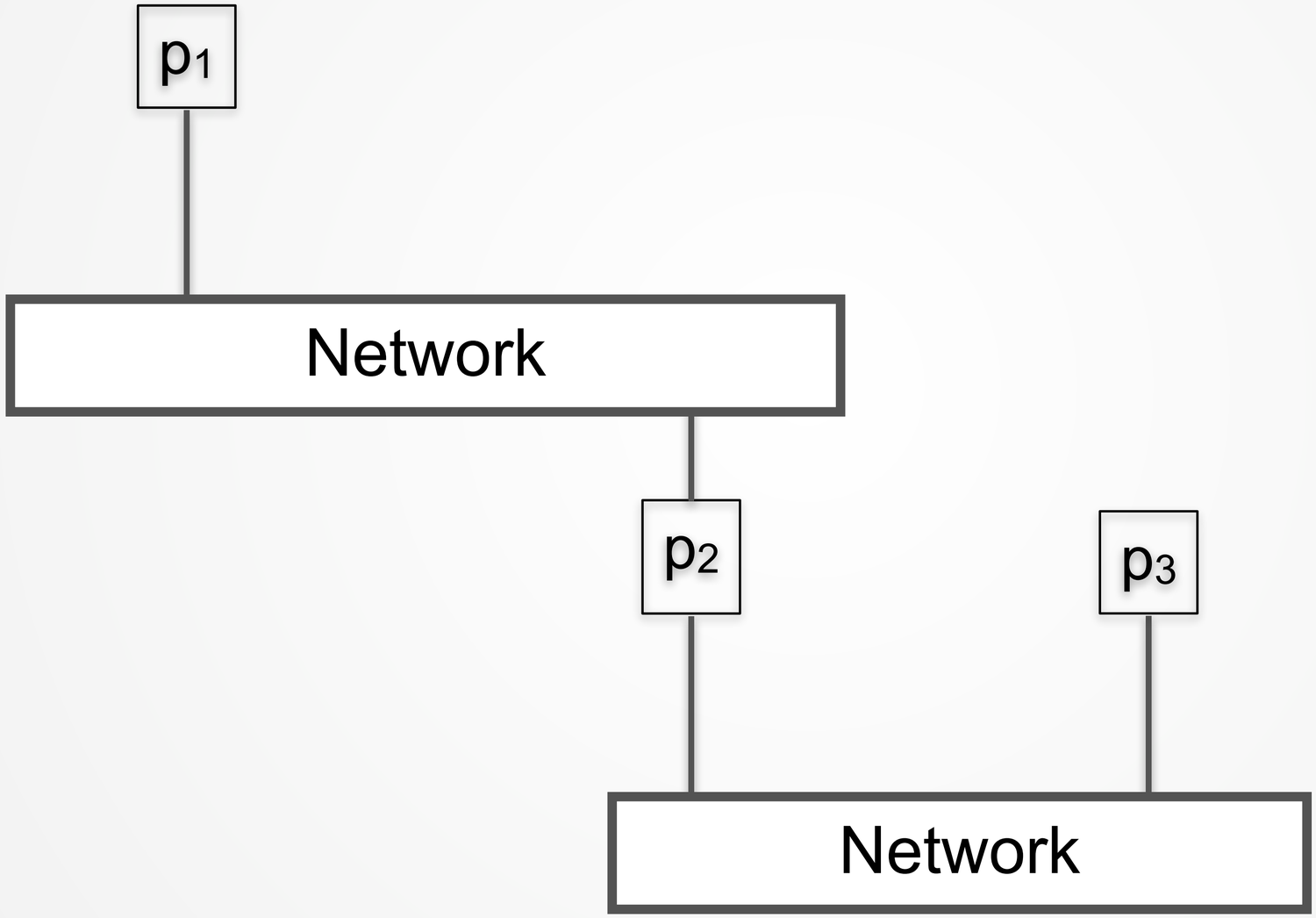


Network Failures

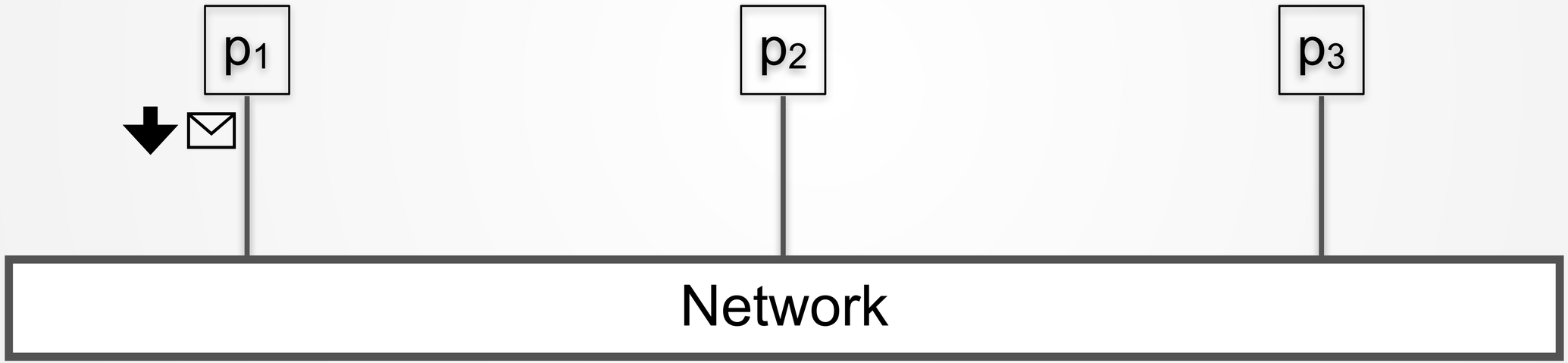


Network Partitions

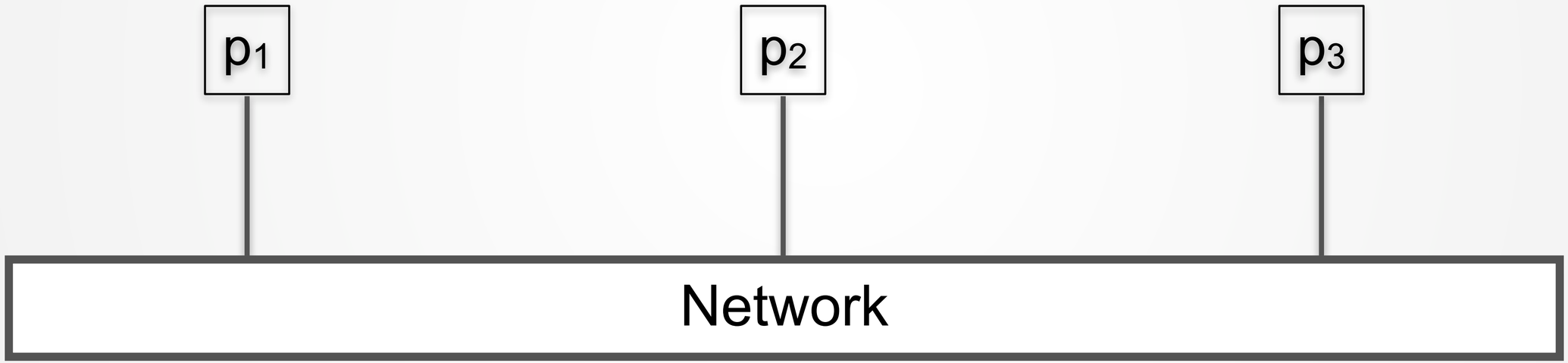
Partial Network Connectivity



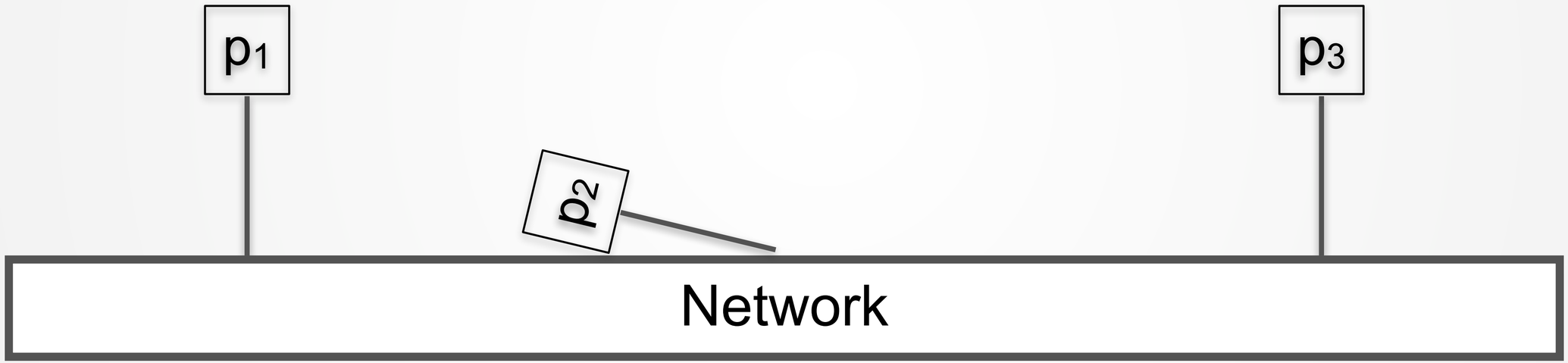
Process Failures



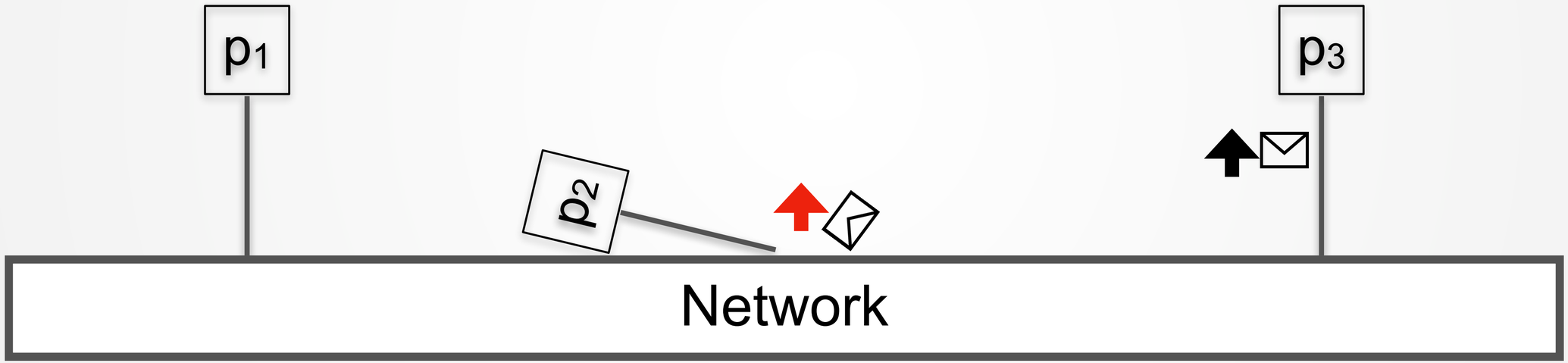
Process Failures



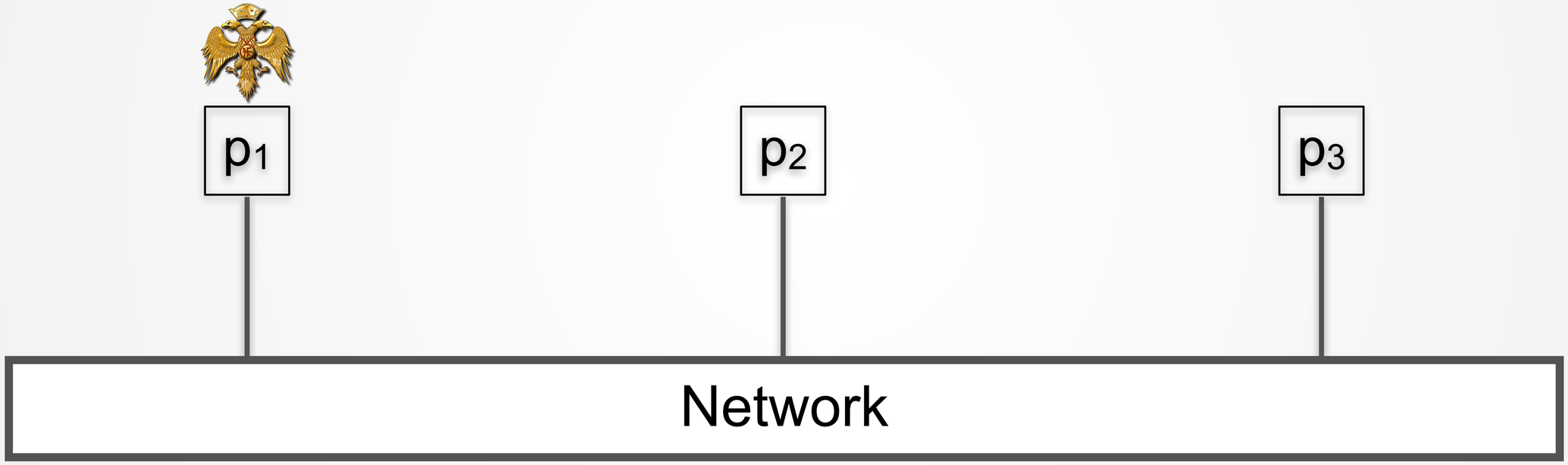
Process Failures



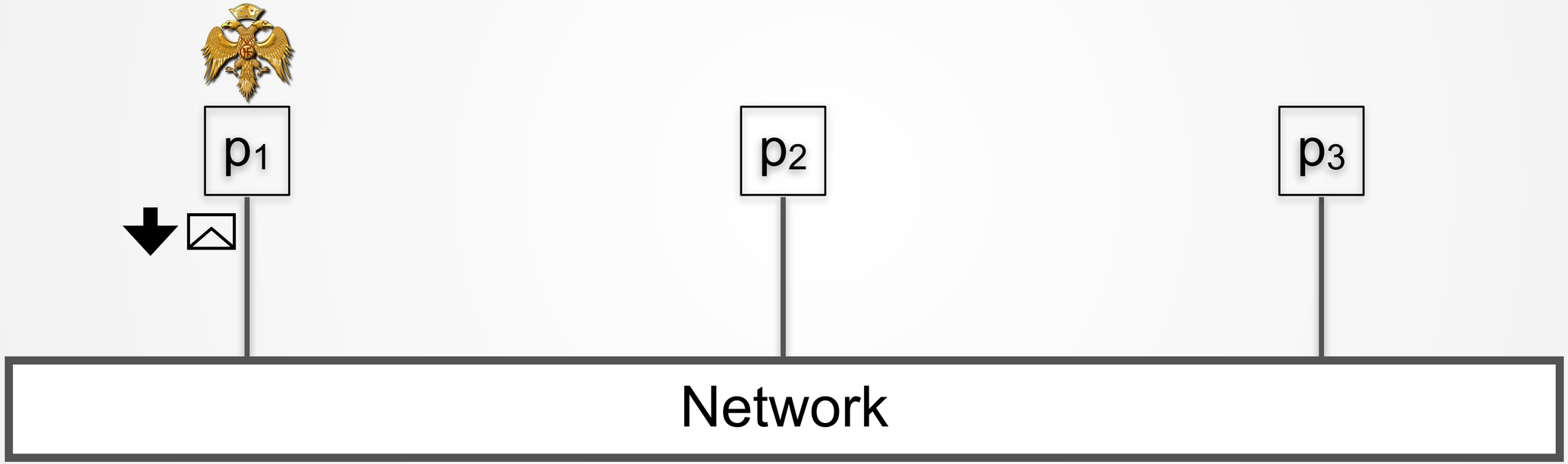
Process Failures



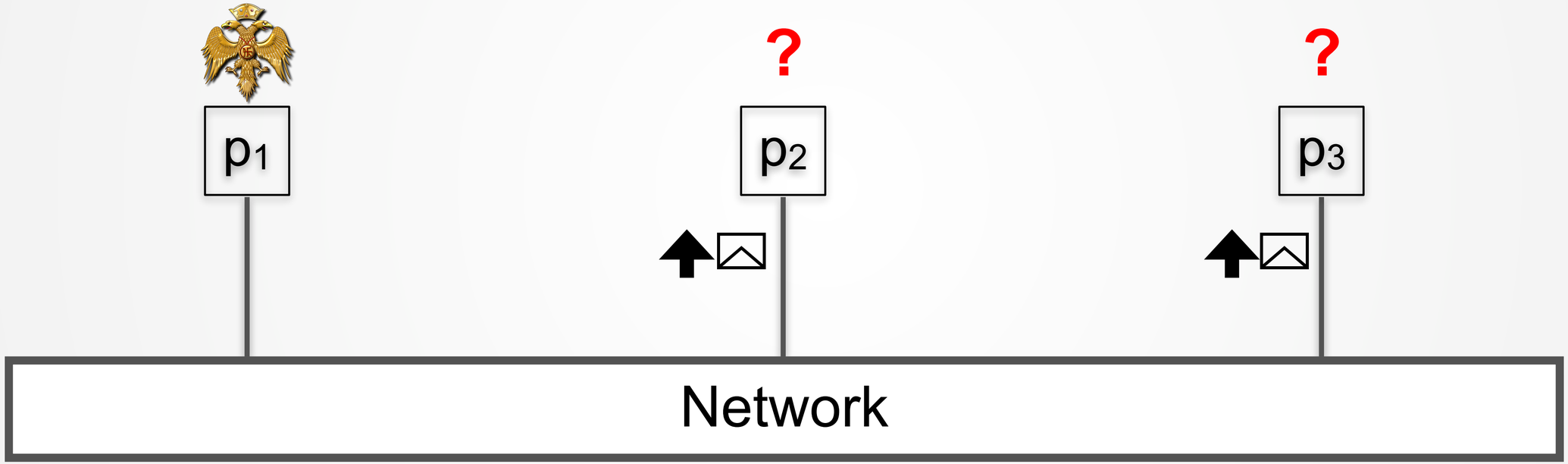
Byzantine Processes



Byzantine Processes



Byzantine Processes



Modelling a Distributed System

The Asynchronous System Model

- No bound on time to deliver a message
- No bound on time to compute
- Clocks are not synchronized

Internet is “pessimistically” asynchronous

Impossibility of Consensus

Consensus (agreement) is non-solvable in asynchronous systems if process crashes can happen. 😢😭😭

Implications on

- Atomic broadcast
- Atomic commit
- Leader election

…

Modelling a Distributed System

Synchronous system

- Known bound on time to deliver a message (latency)
- Known bound on time to compute
- Known lower and upper bounds in physical clock drift rate

Examples:

- Embedded systems (shared clock)
- Multicore computers

Possibility of Consensus

Consensus is **solvable** in synchronous system with up to N-1 crashes 🥳

Intuition behind solution

- Accurate crash detection
  - Every node sends a message to every other node
  - If no msg from a node within bound, node has crashed

Not useful for Internet, how to proceed?

Modelling a Distributed System

A more realistic view of most systems (e.g., over internet)

- Bounds respected mostly
- Occasionally violate bounds (congestion/failures)

How do we model this?

Partially synchronous system

- Initially system is asynchronous
- Eventually the system becomes synchronous (for a while)

Possibility of Consensus

Consensus **solvable** in any partially synchronous system with up to N/2 crashes 🎉🎉

FAILURE DETECTORS

Let each node use a **failure detector**

- Detects crashes
- Implemented by heartbeats and waiting
- Might be initially wrong, but eventually correct

Consensus and Atomic Broadcast solvable with failure detectors

How? Attend rest of course!

Specialized Models

Timed Asynchronous system

- No bound on time to deliver a message
- No bound on time to compute
- **Clocks have known clock-drift rate**

Another realistic model for the Internet

Specialized Models

Eventually Consistent System

- Processes can be partially/fully partitioned or fail temporarily.
- Eventually processes reconnect or recover (for a while)
- Computation is monotonic.
- Favors IoT networks

This model inspired **Conflict-Free Replicated Data Types** which we will explore in the advanced lectures!

Byzantine Faults

Some processes might behave arbitrarily

- Sending wrong information
- Omit messages…

Byzantine algorithms that tolerate such faults

- Only tolerate up to 1/3 Byzantine processes
- Non-Byzantine algorithms can often tolerate ½ nodes in the asynchronous model

Summary

Distributed systems everywhere

- Set of processes (nodes) cooperating over a network

Few core problems reoccur

- Consensus, Broadcast, Leader election, Shared Memory

Different failure scenarios important

- Crash stop, network partitions, Byzantine processes

Interesting research directions

- Large scale dynamic distributed systems