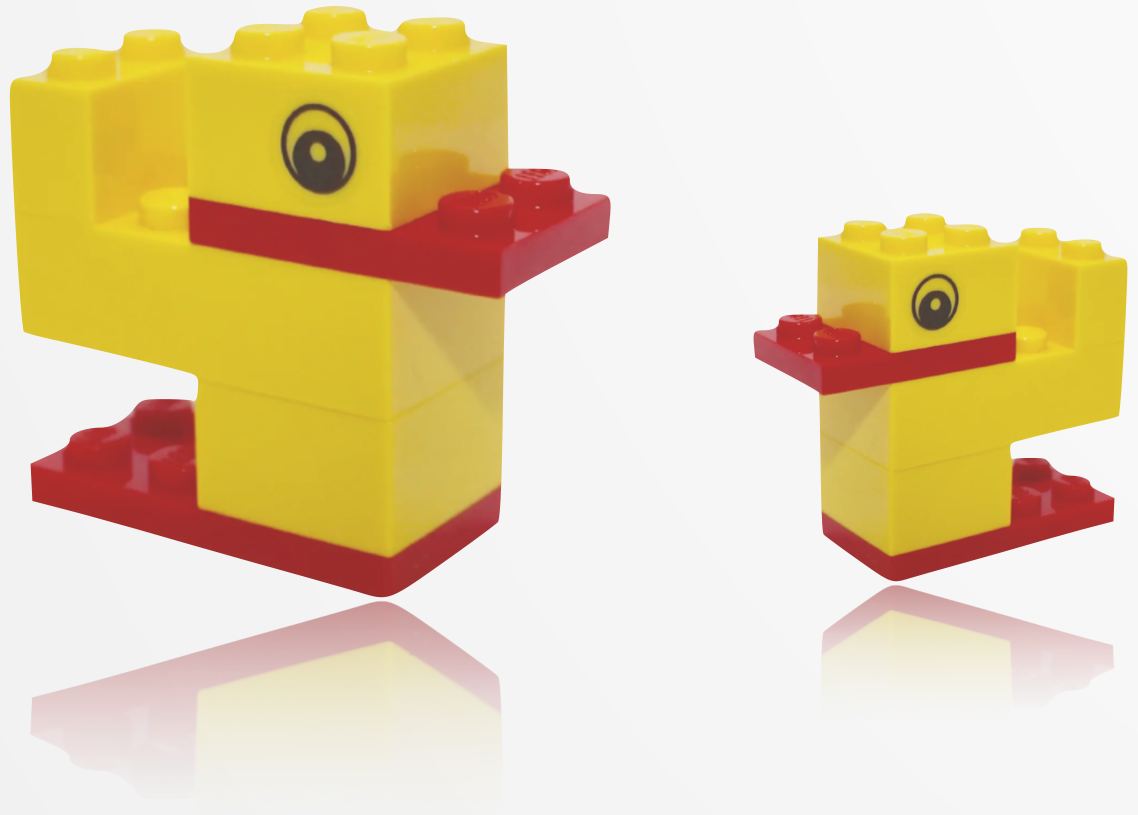
Advanced Course
Distributed Systems

Introduction to Distributed Systems



Paris Carbone

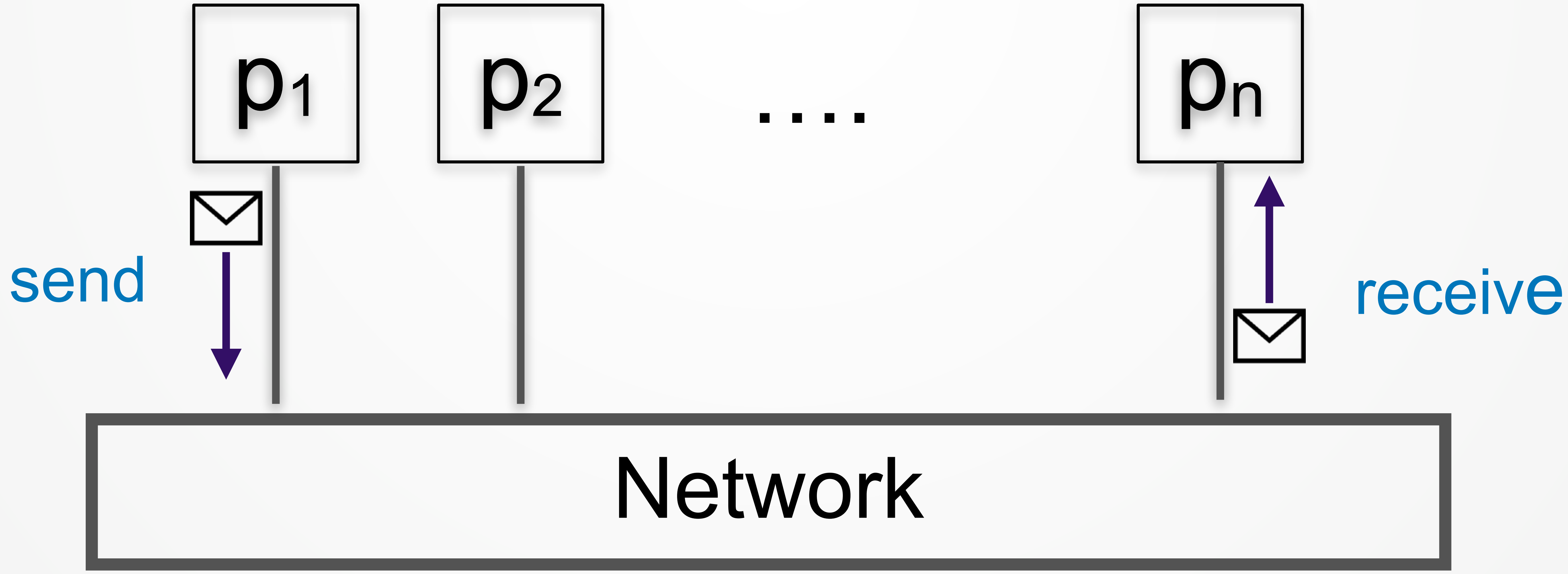
Course Topics

- Intro to Distributed Systems
- Fundamental Abstractions and Failure Detectors
- Reliable and Causal Order Broadcast
- Distributed Shared Memory-CRDTs
- Consensus (Paxos)
- Replicated State Machines (OmniPaxos, Raft, Zab etc.)
- Time Abstractions and Interval Clocks (Spanner etc.)
- Consistent Snapshotting (Stream 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"

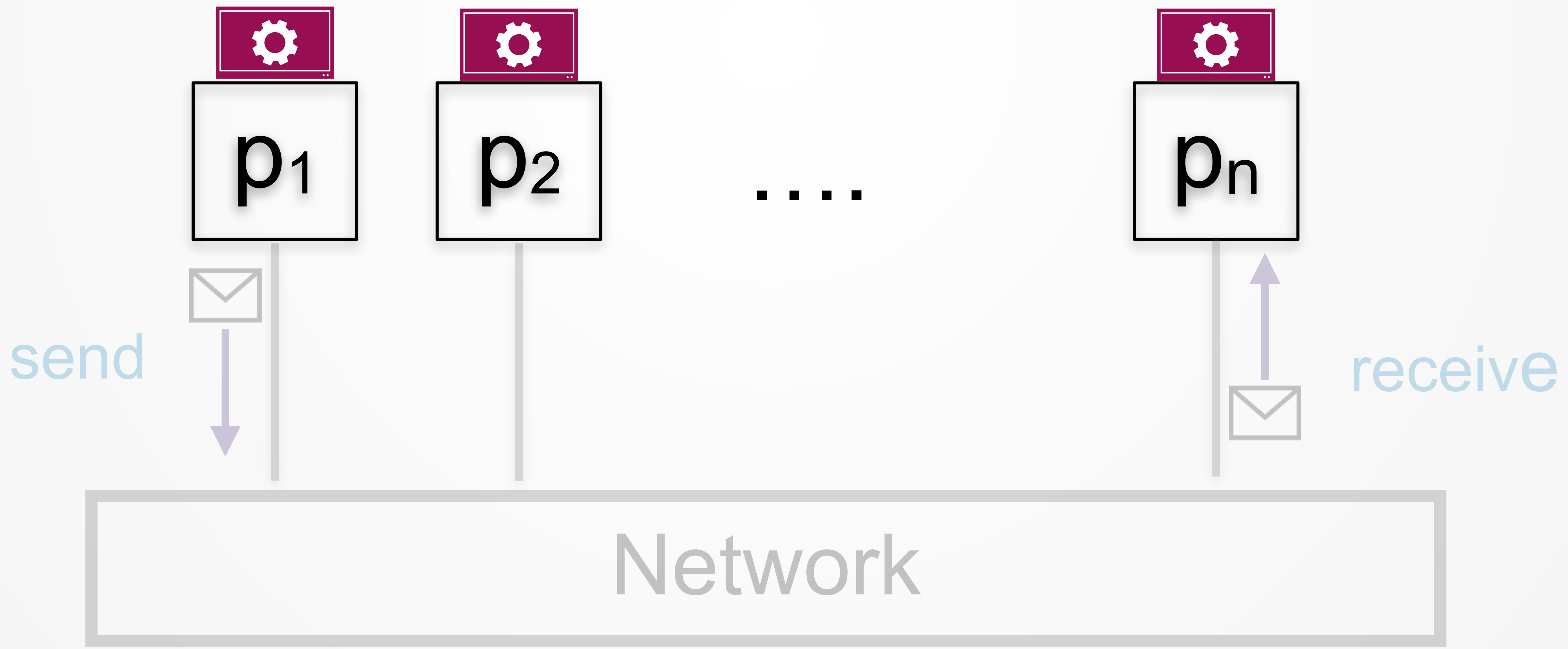


ID2203

KTH-2023

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?

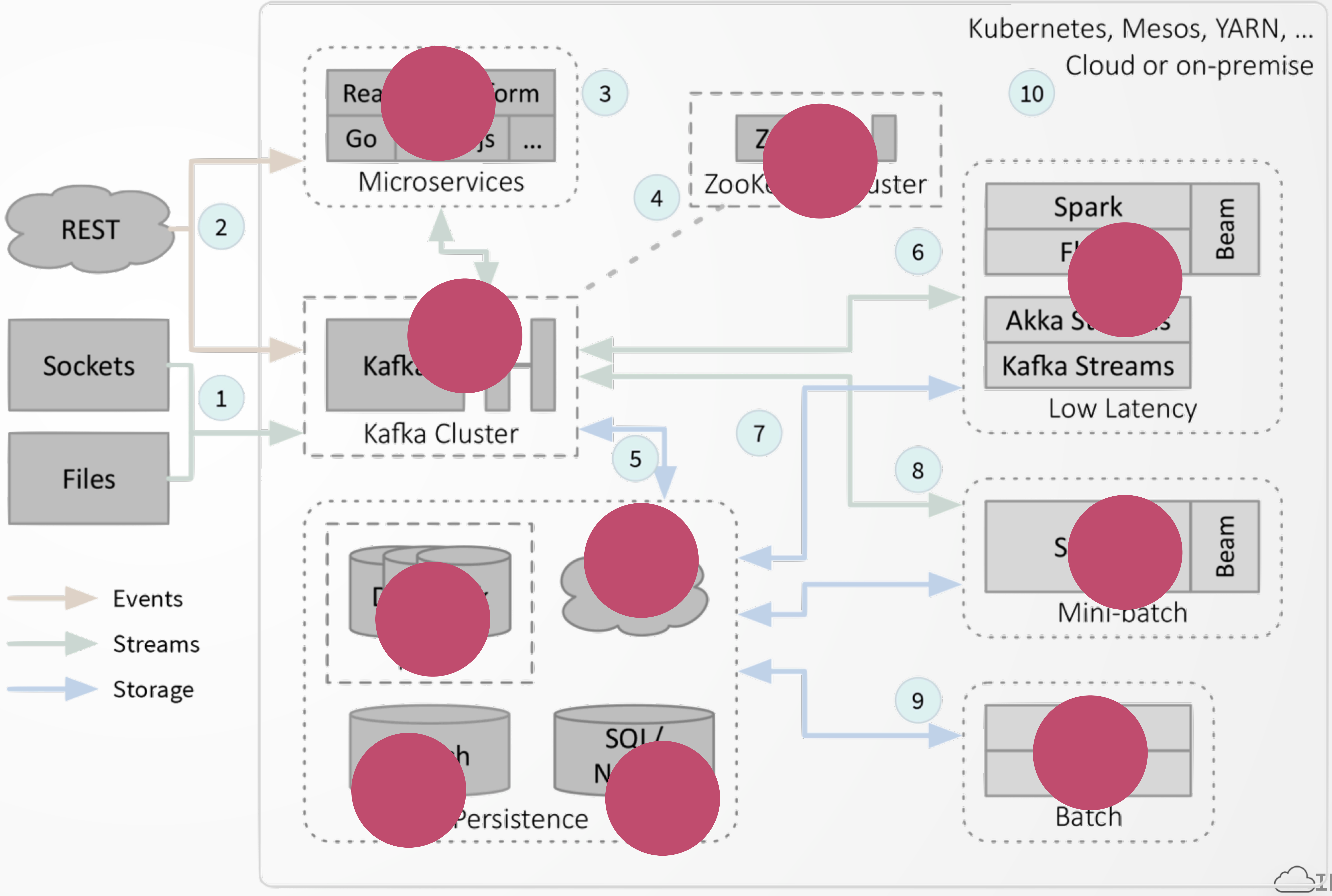
It is important, useful and interesting

Societal importance

- Internet
- WWW
- Cloud computing (e.g., Google, Amazon)
- Edge computing
- Small devices (mobiles, sensors)

WEB SERVICES ARE DISTRIBUTED SYSTEMS OF DISTRIBUTED SYSTEMS

Each subsystem is a point of critical failure



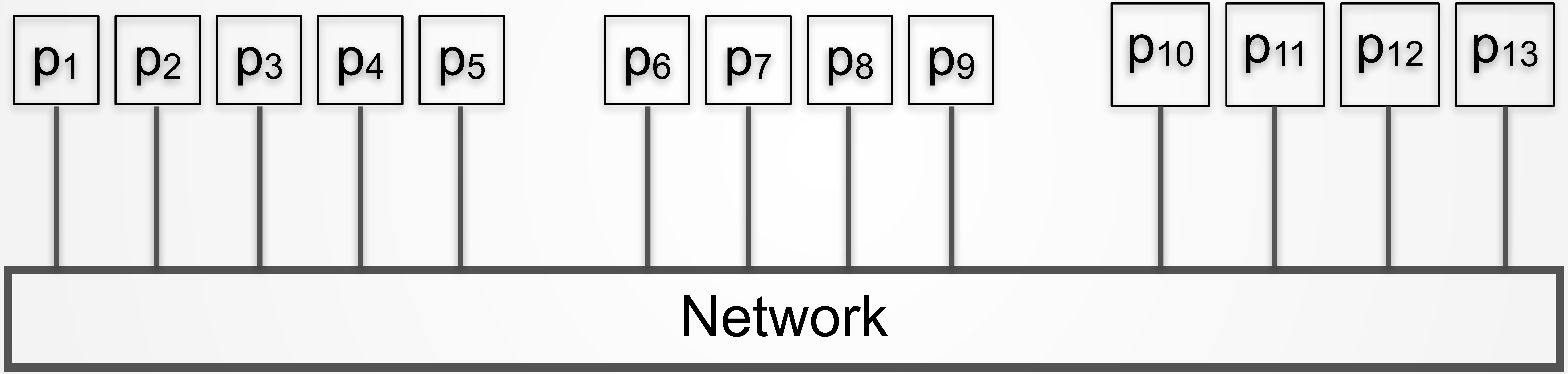
Source: Portals Presentation: Carbone, Haller

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?

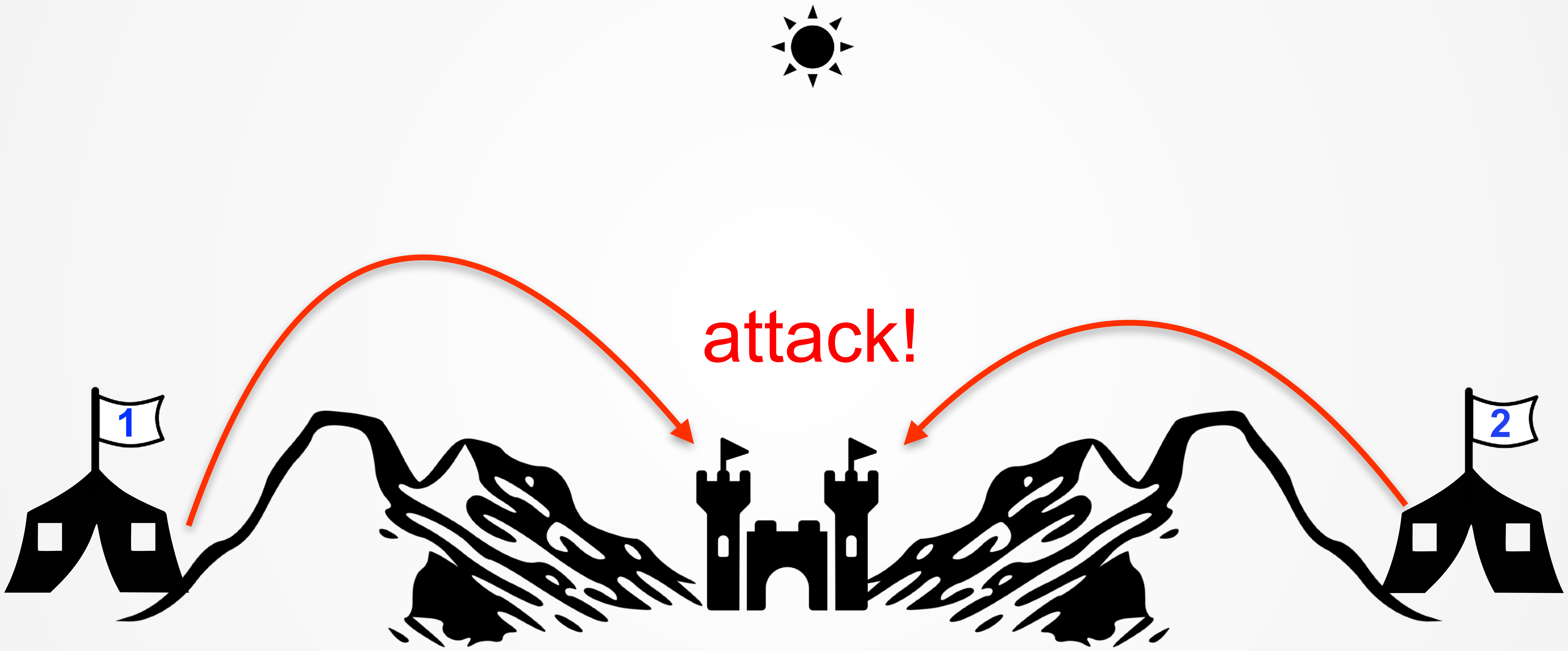
TEASER: TWO GENERALS' PROBLEM

"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

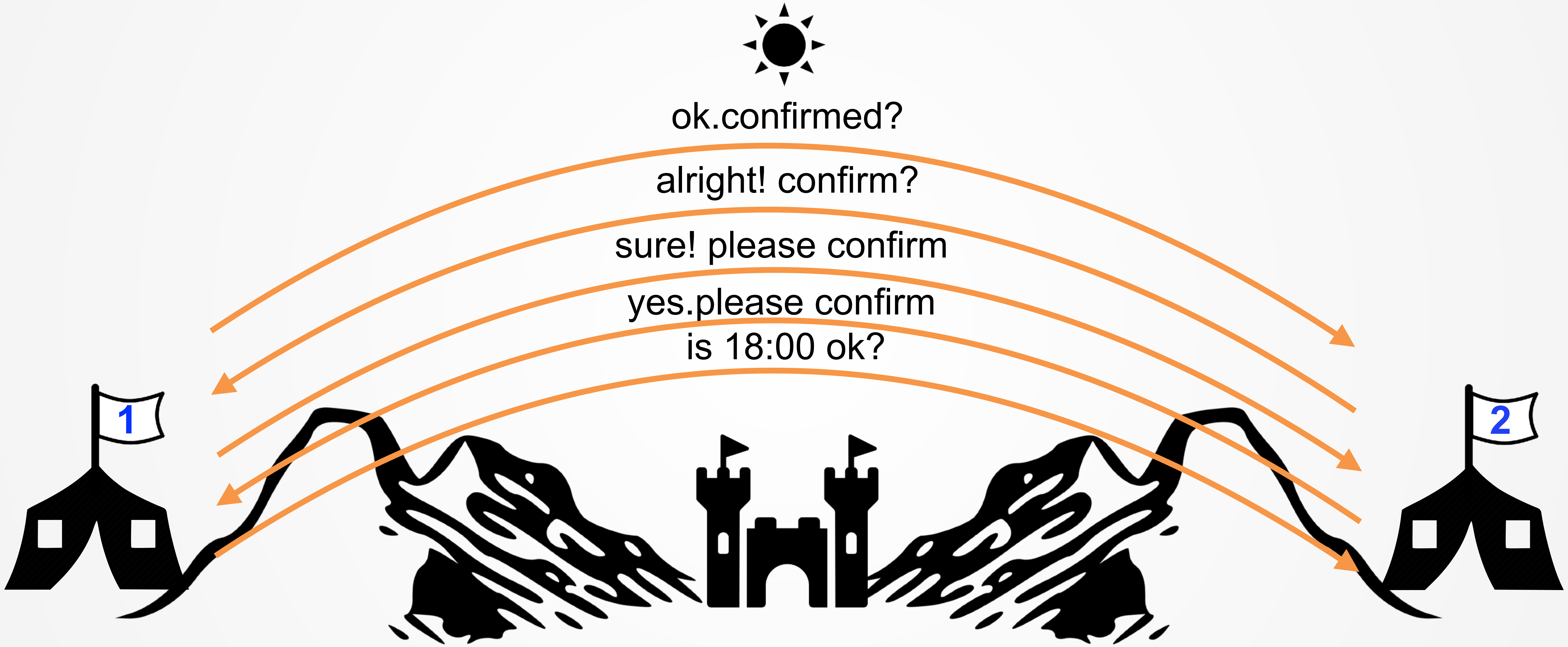
Teaser: Two Generals' Problem

Teaser: Two Generals' Problem

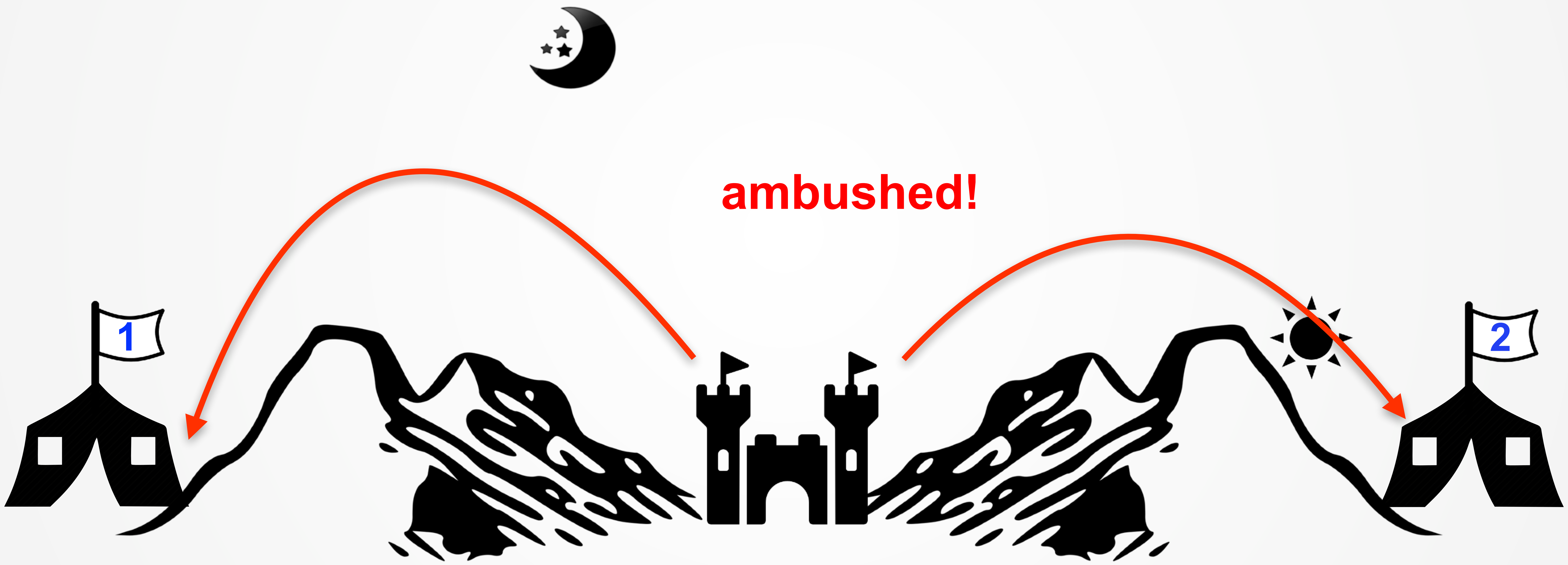


Teaser: Two Generals' Problem

Teaser: Two Generals' Problem



Teaser: Two Generals' Problem



Impossible to solve!

Teaser: Two Generals' Problem

Applicability to distributed systems

- Two processes need to agree on a value before a specific time-bound
- Communicate by messages using an unreliable channel

Agreement is a core problem…

CONSENSUS: AGREEING ON A NUMBER

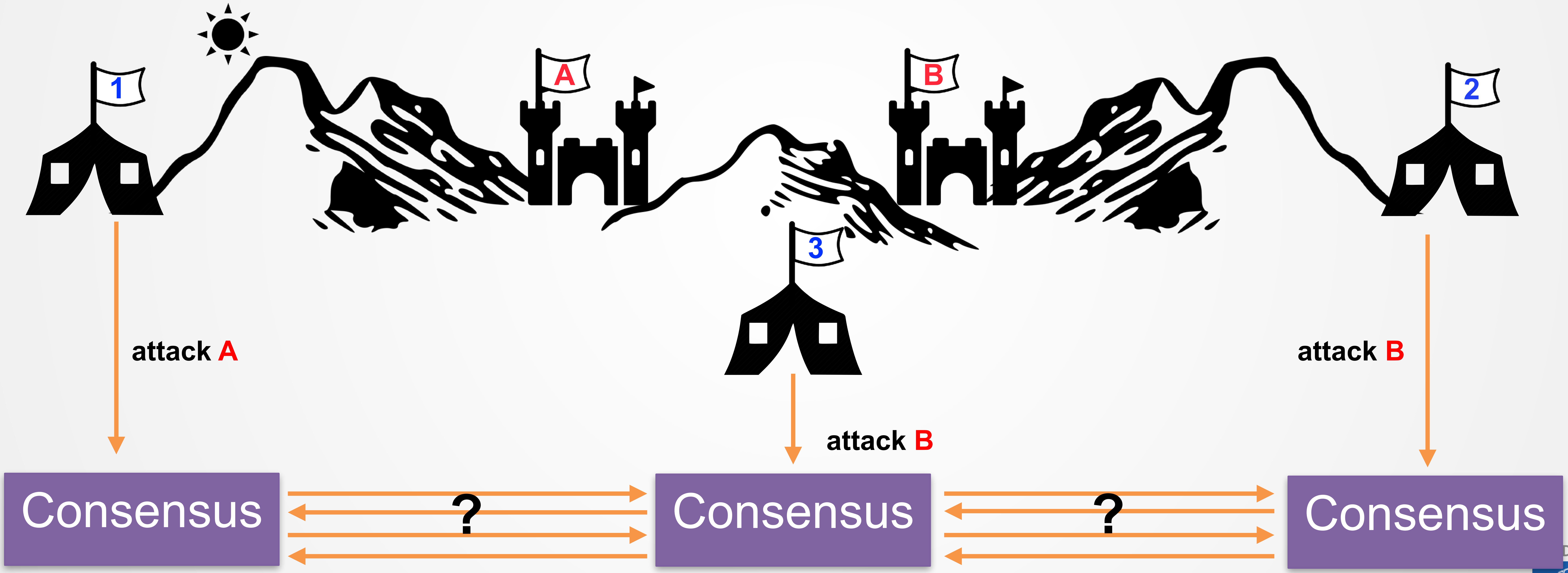
Consensus problem

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

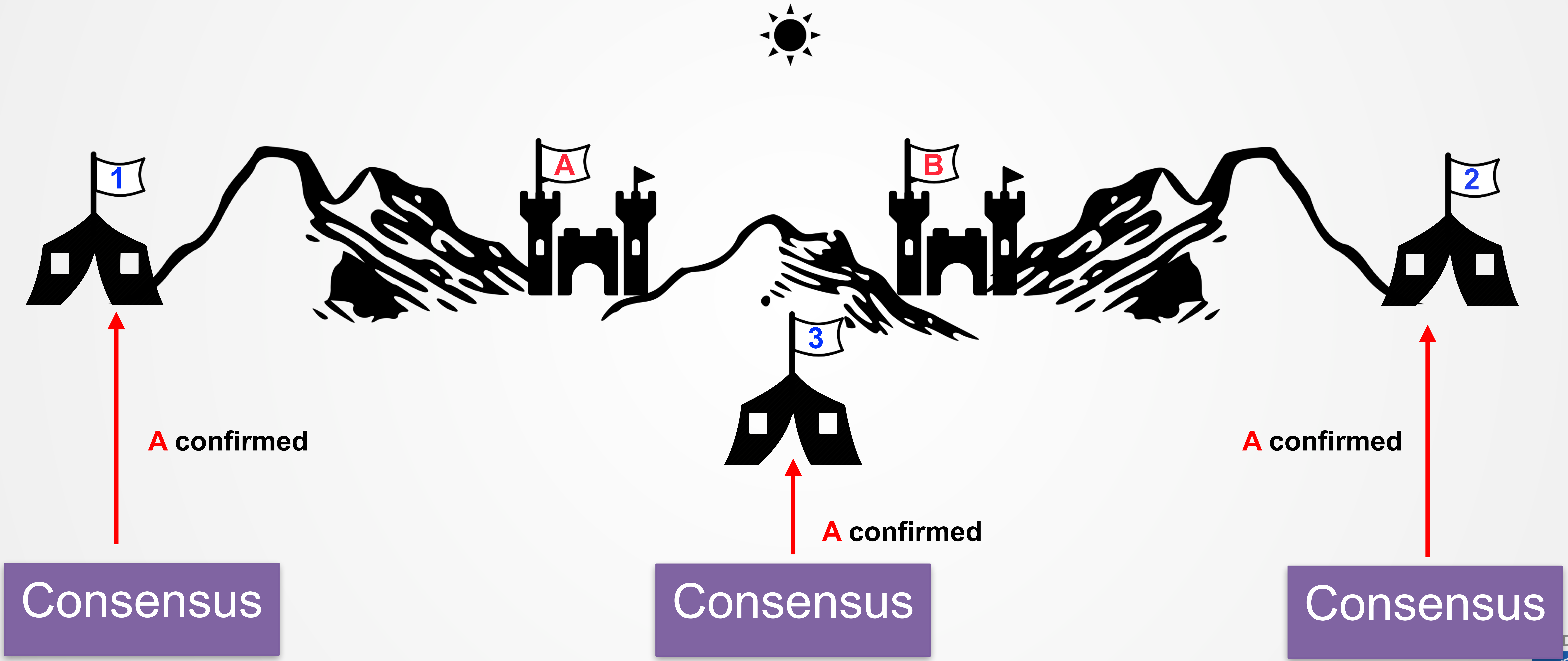
The algorithm must ensure a set of properties (specification):

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

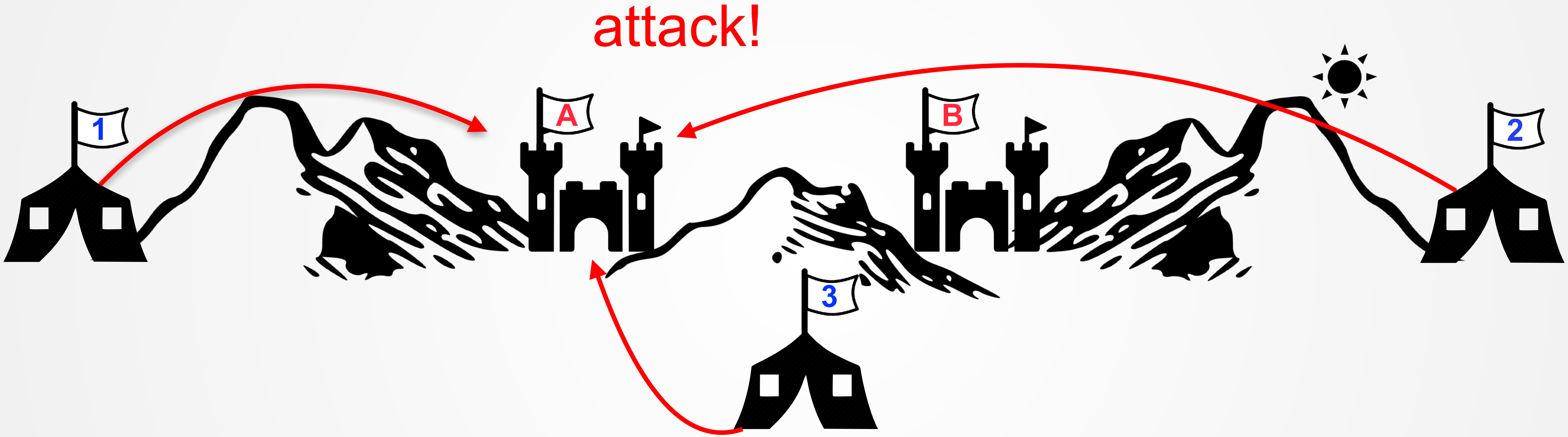
EXAMPLE: AGREEING ON A TARGET



Example: Agreeing on a Target



Example: Agreeing on a Target

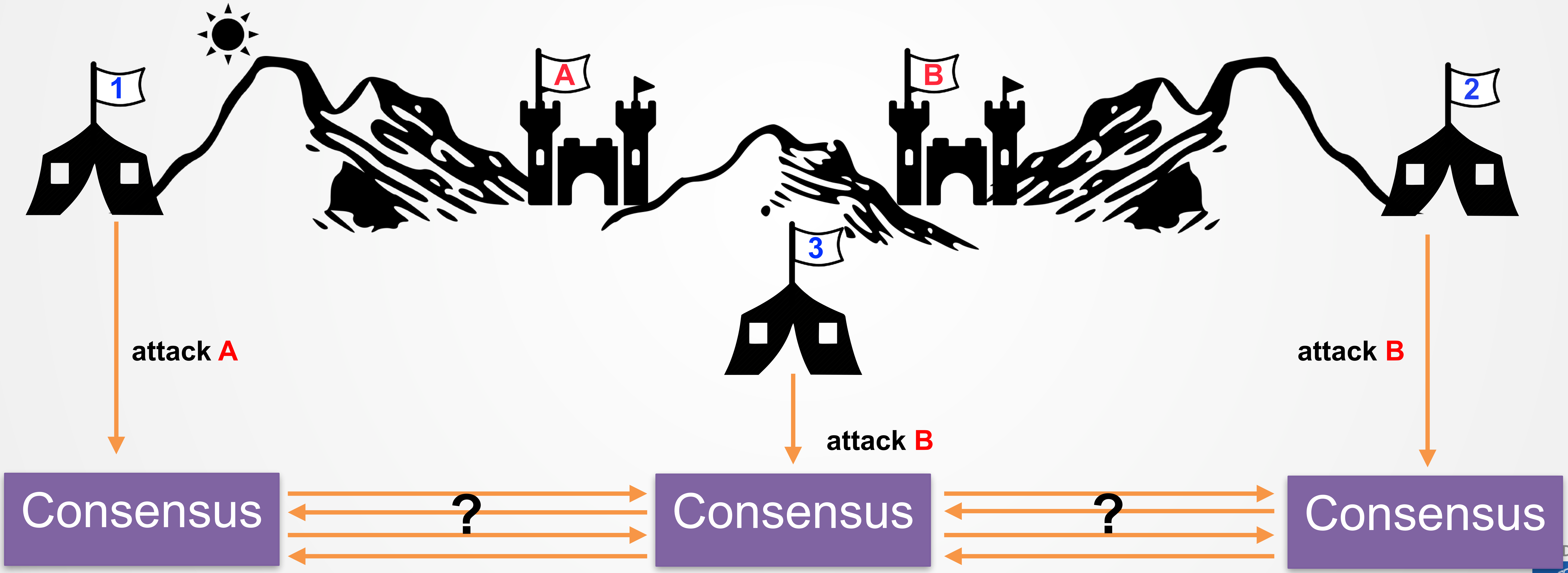


Consensus

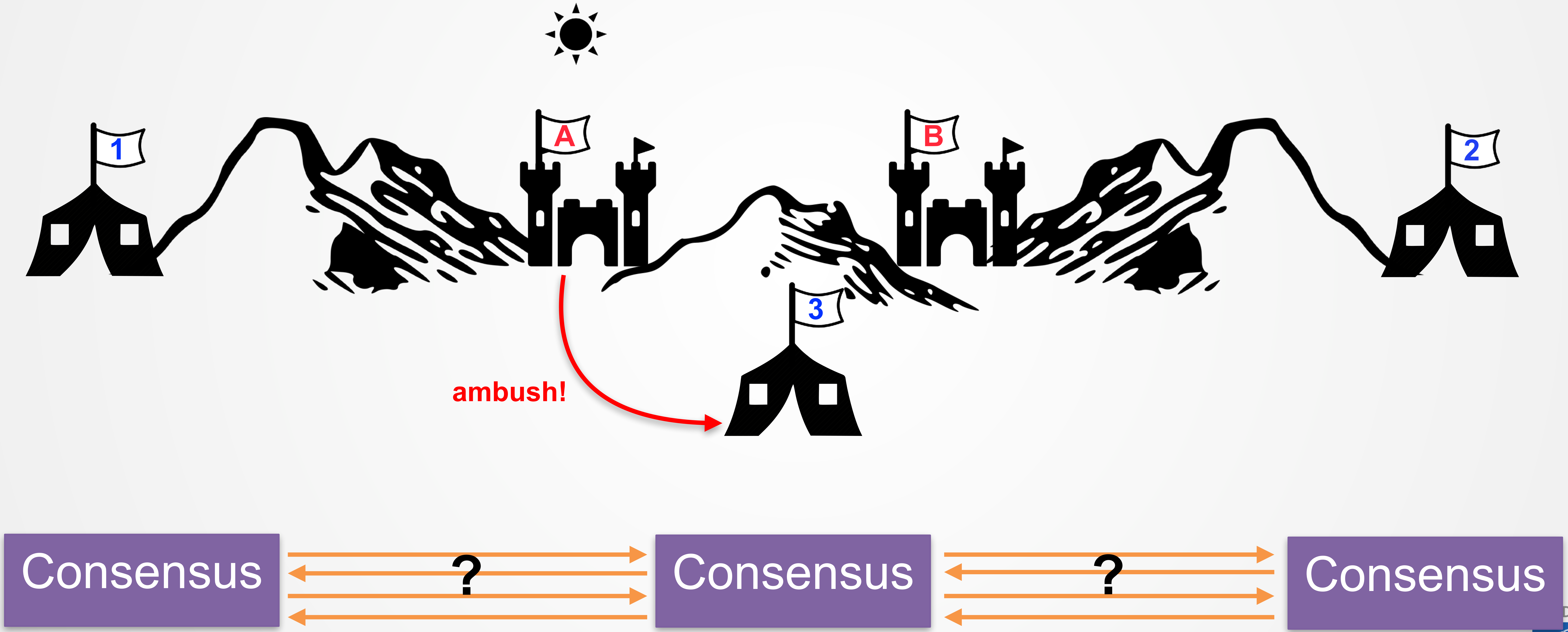
Consensus

Consensus

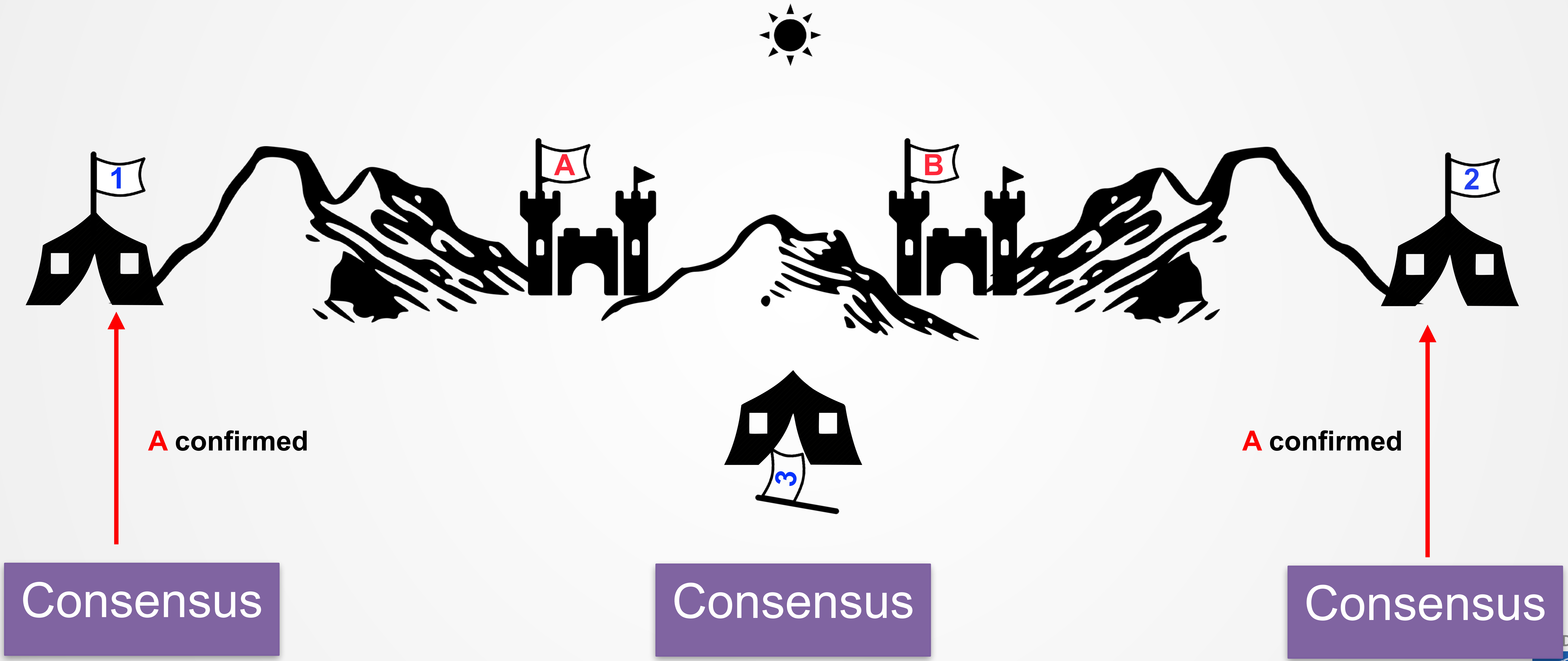
EXAMPLE: AGREEING ON A TARGET



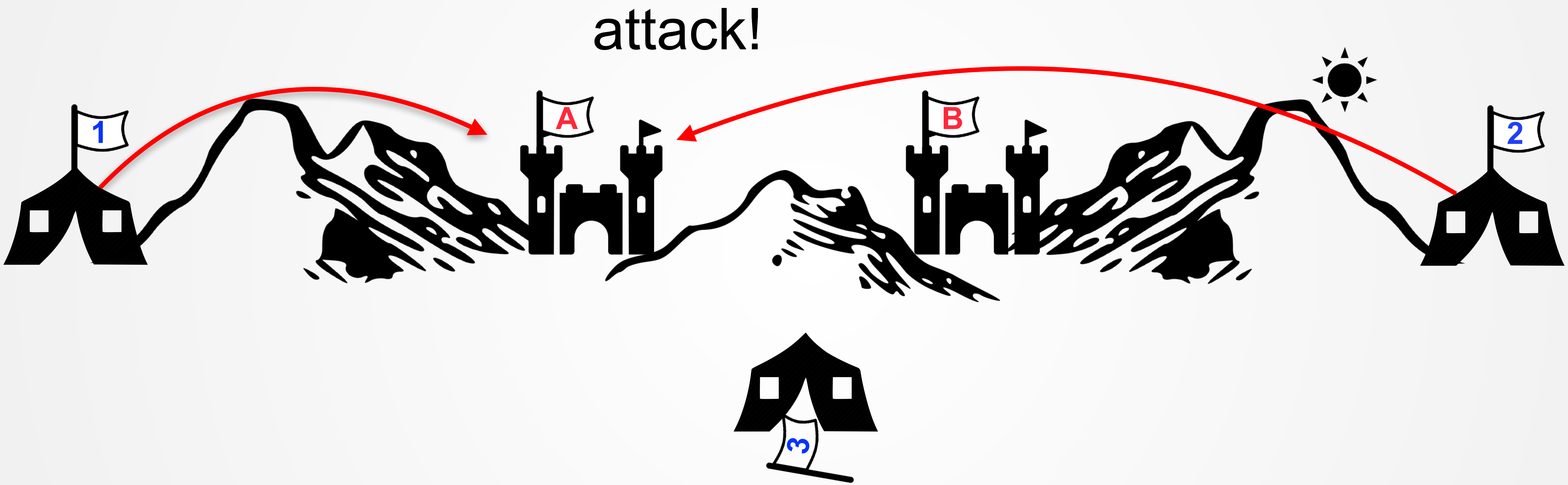
Example: Agreeing on a Target



Example: Agreeing on a Target



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

Consensus is Important

Distributed Databases / Cloud Stores

Concurrent changes/transactions to same data

{commit}: If **every** node agrees to commit

{abort}: If at least one node aborts

Use a form of consensus: atomic commit

Only two proposal values {commit, abort}

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

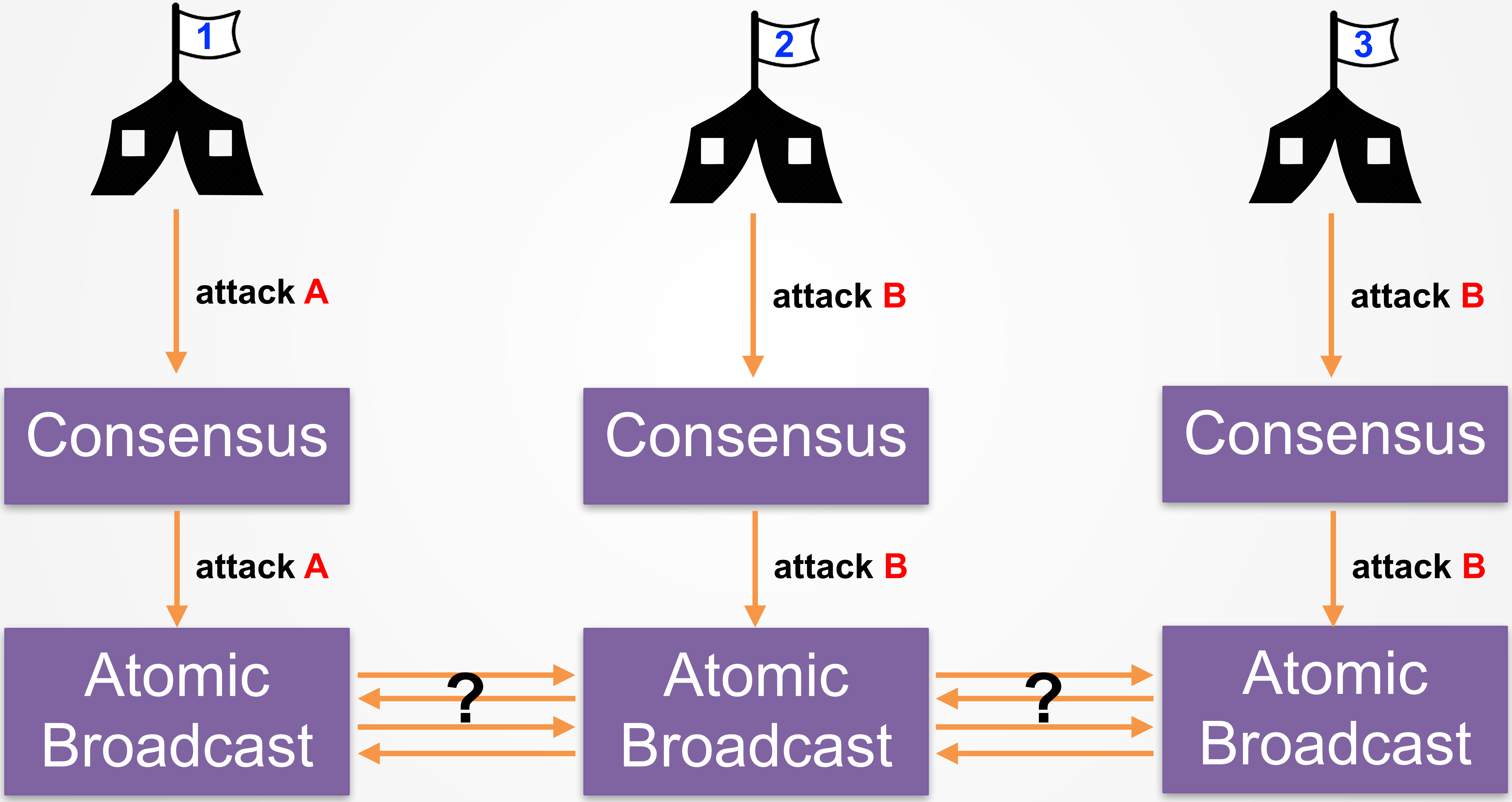
- Decide on the **first** received proposal
- Messages received in same order
  - Thus, all nodes will decide the same value.

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

(more on that later in the course)

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

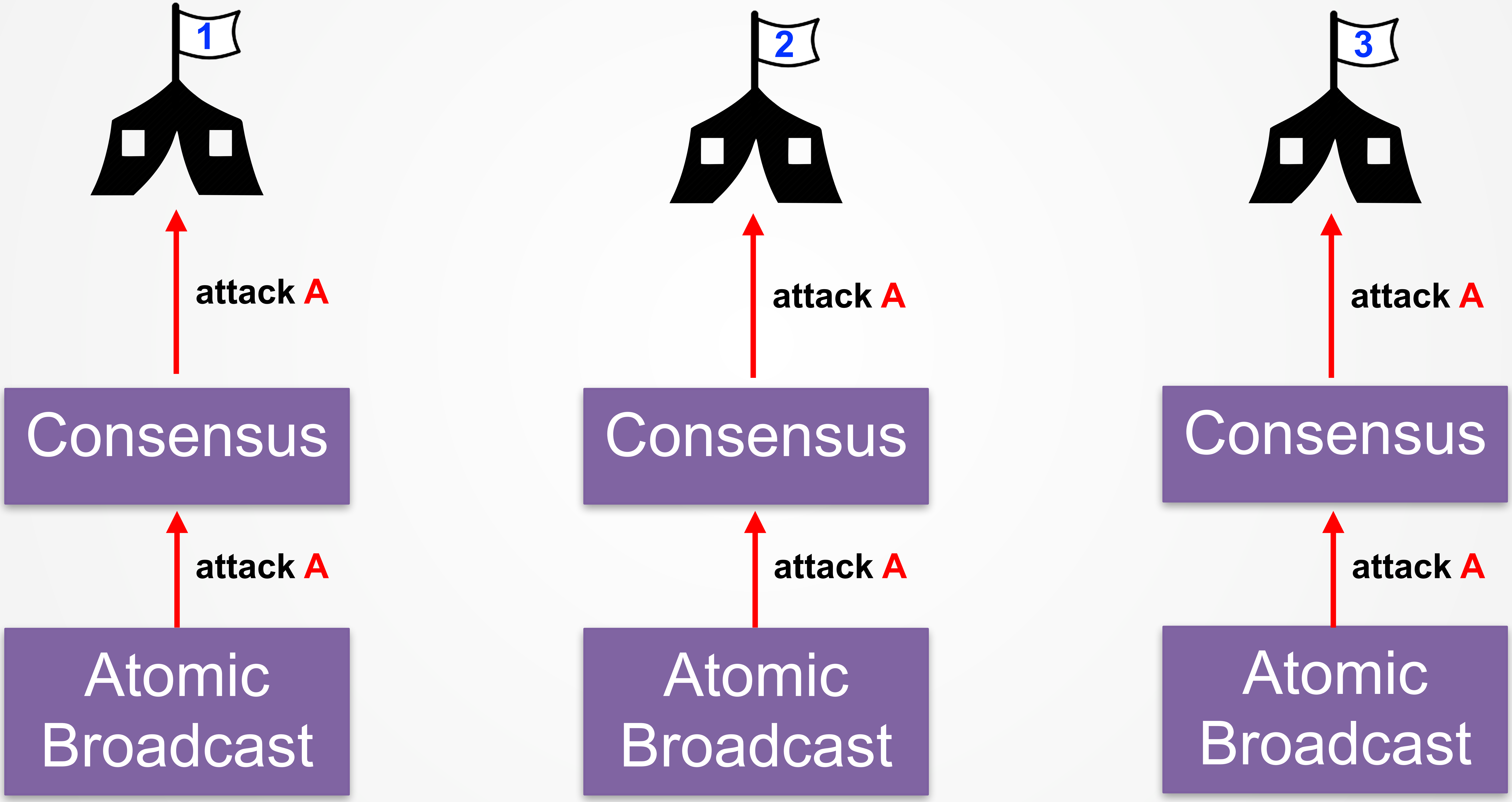
Atomic Broadcast ↔ Consensus



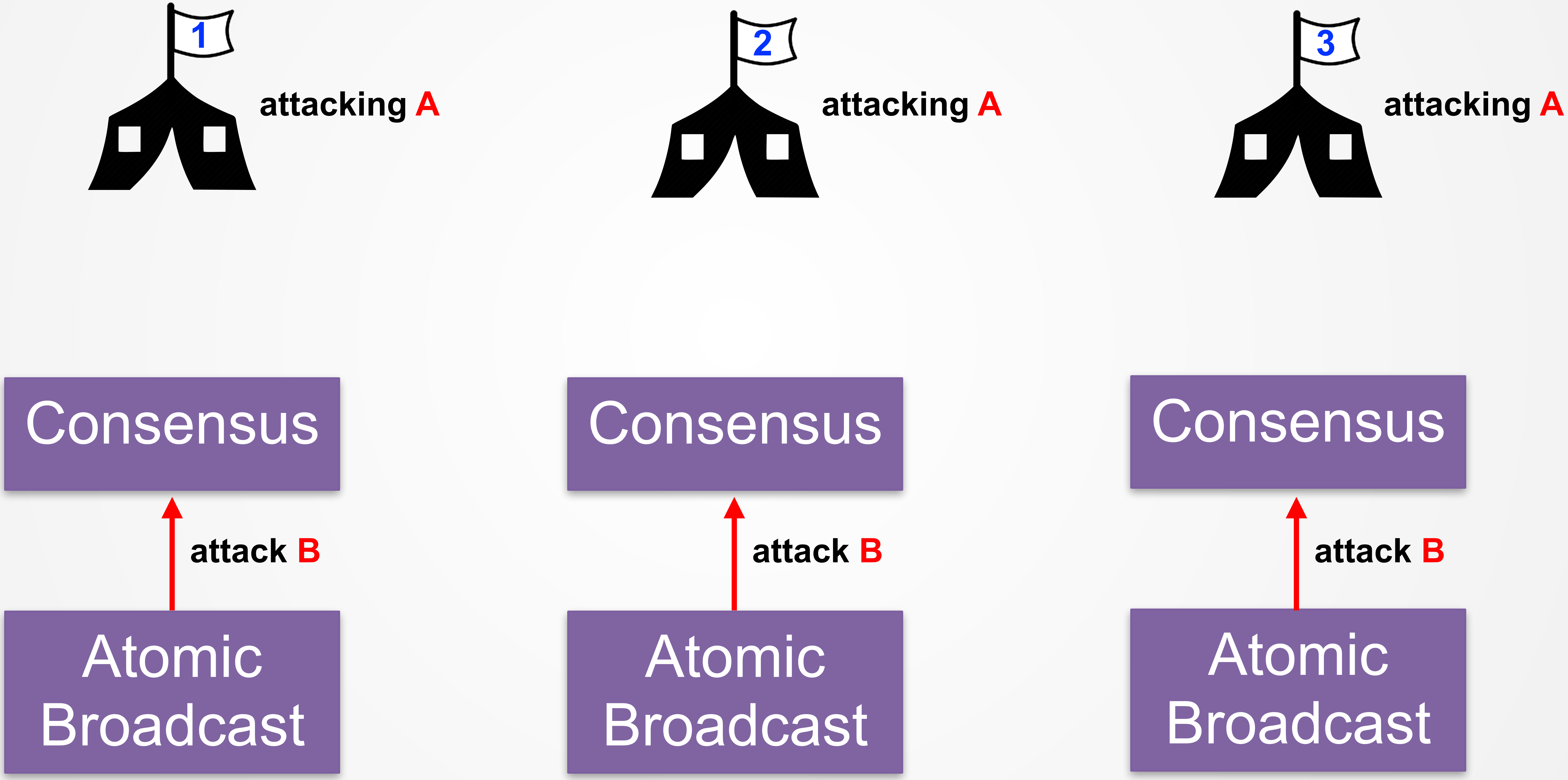
ID2203

KTH-2023

Atomic Broadcast ↔ Consensus



ATOMIC BROADCAST ↔ CONSENSUS



Models of Distributed Systems

How to reason about them?

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

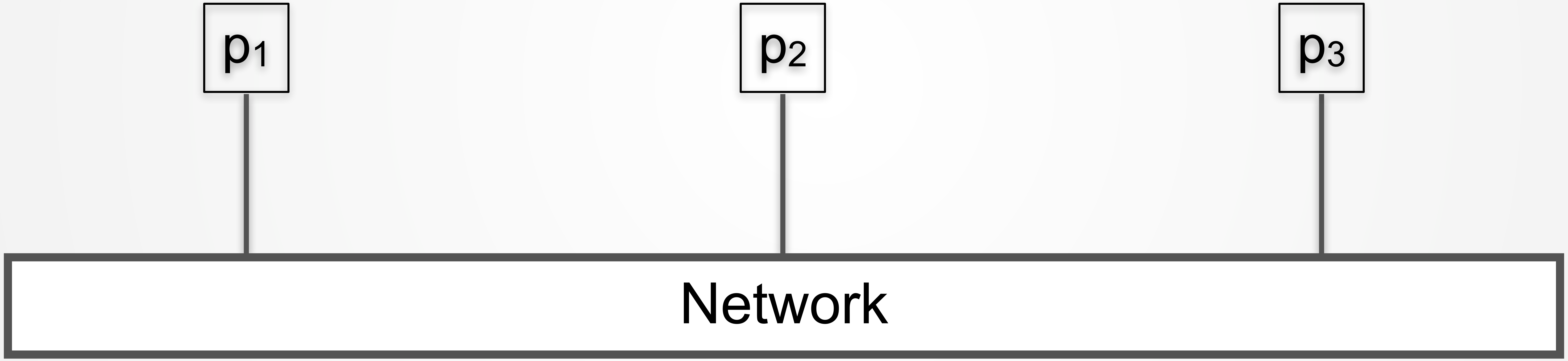
Processes

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

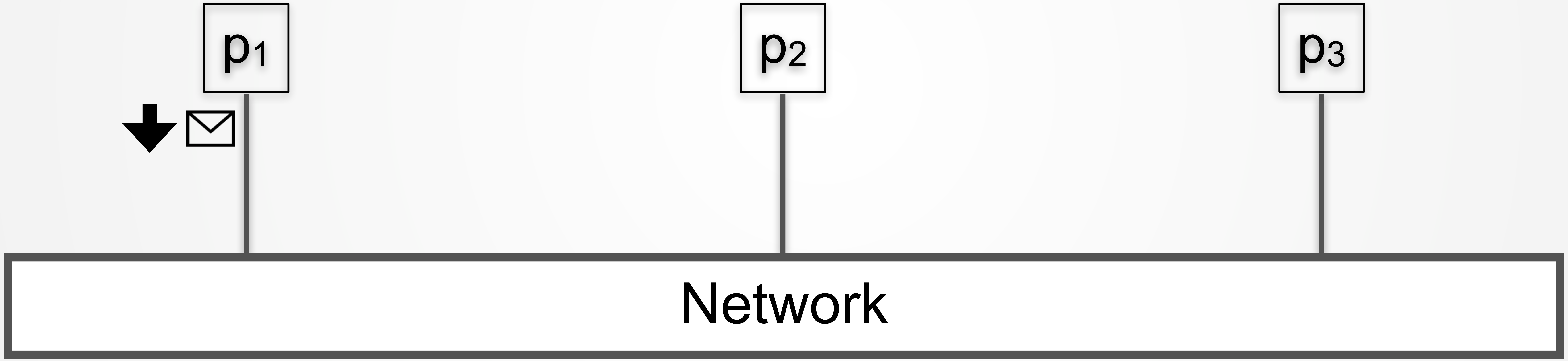
Network

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

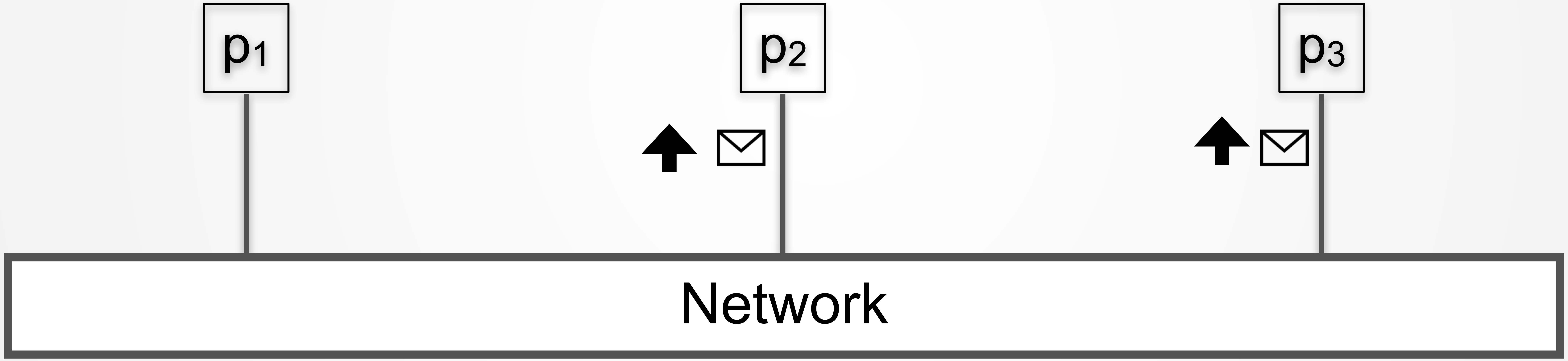
Modeling a Distributed System



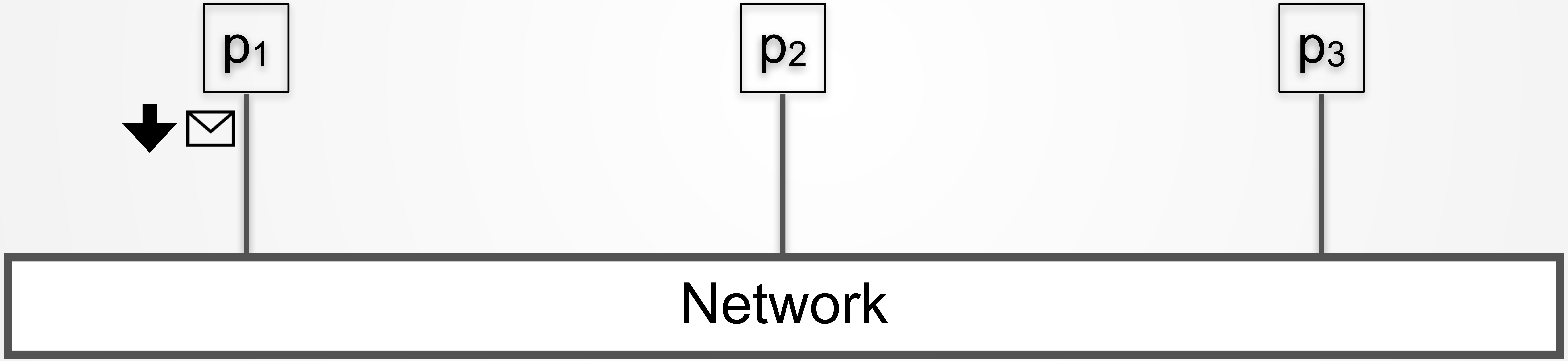
Modeling a Distributed System



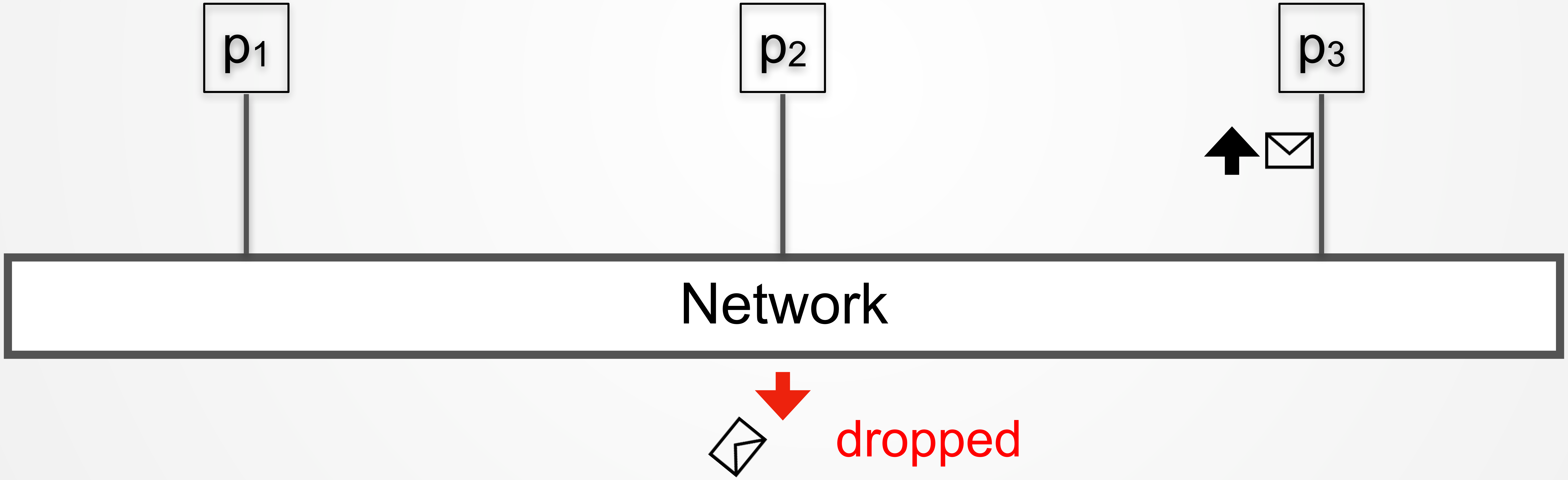
Modeling a Distributed System



Network Failures

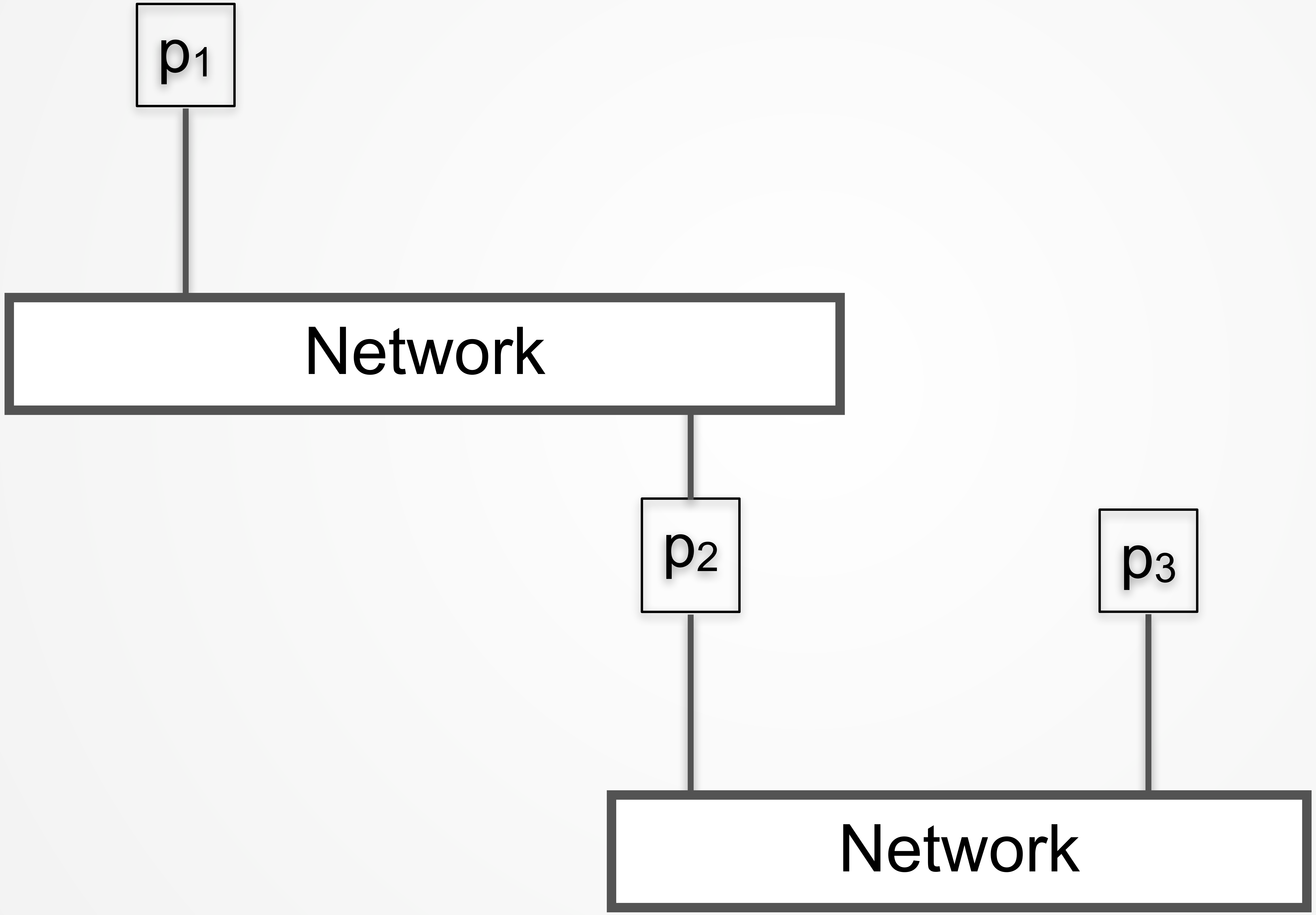


Network Failures

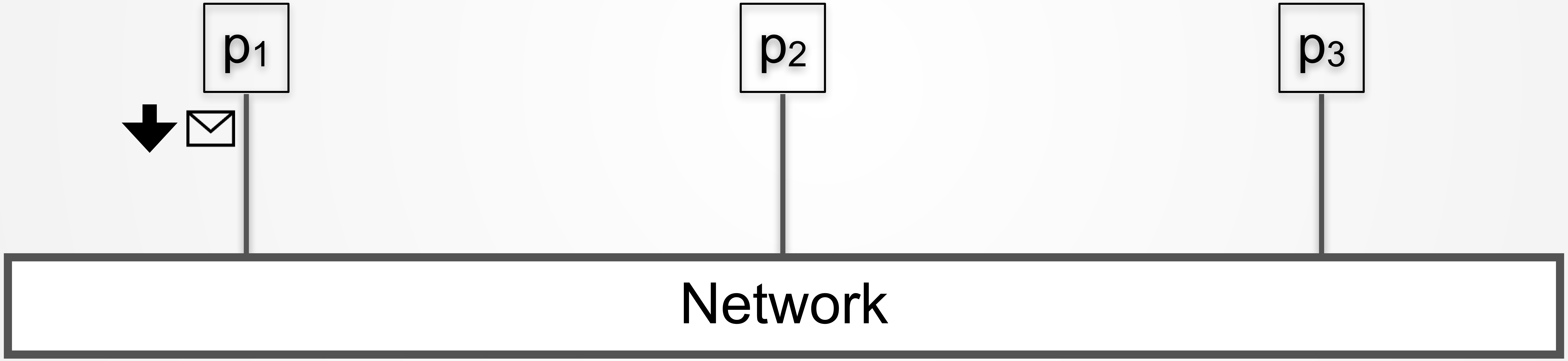


Network Partitions

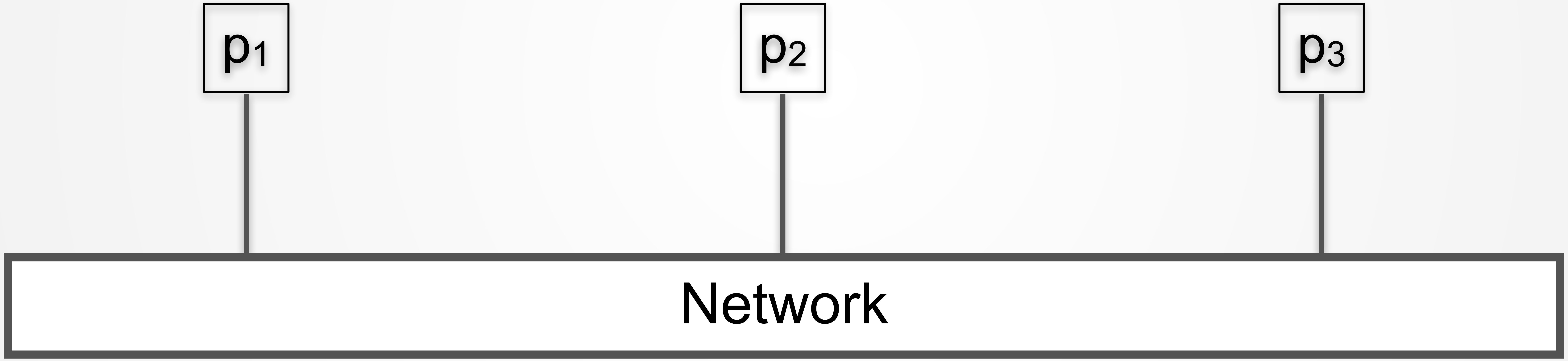
Partial Network Connectivity



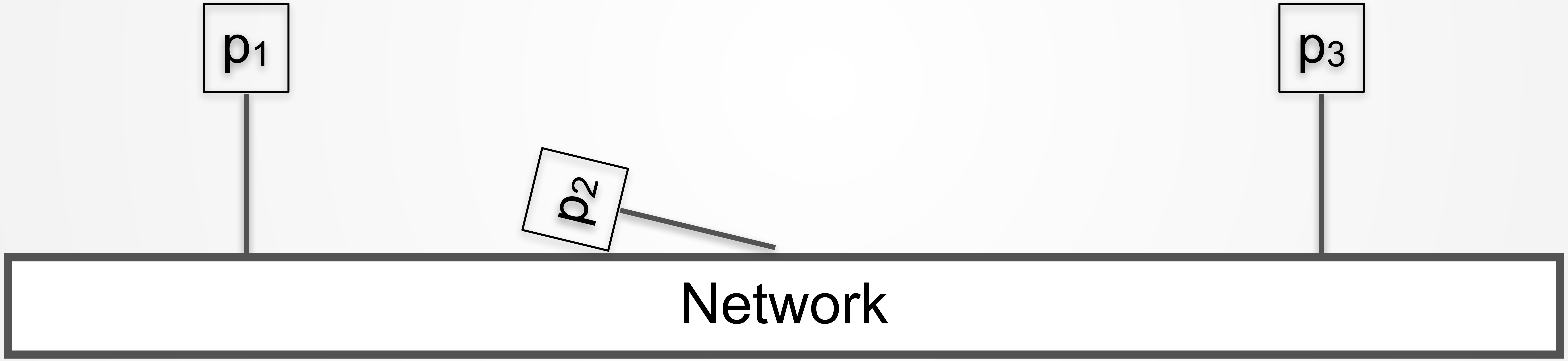
Process Failures



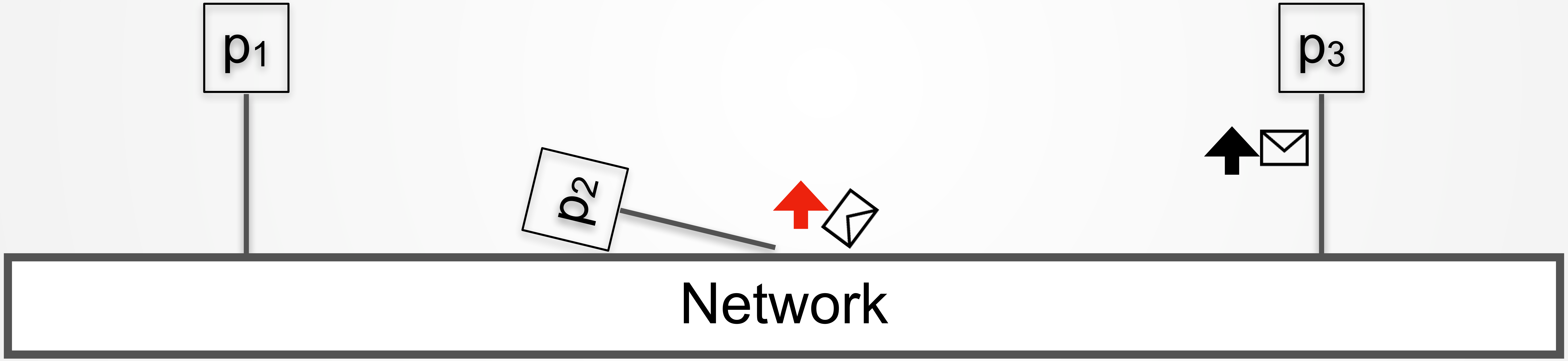
Process Failures



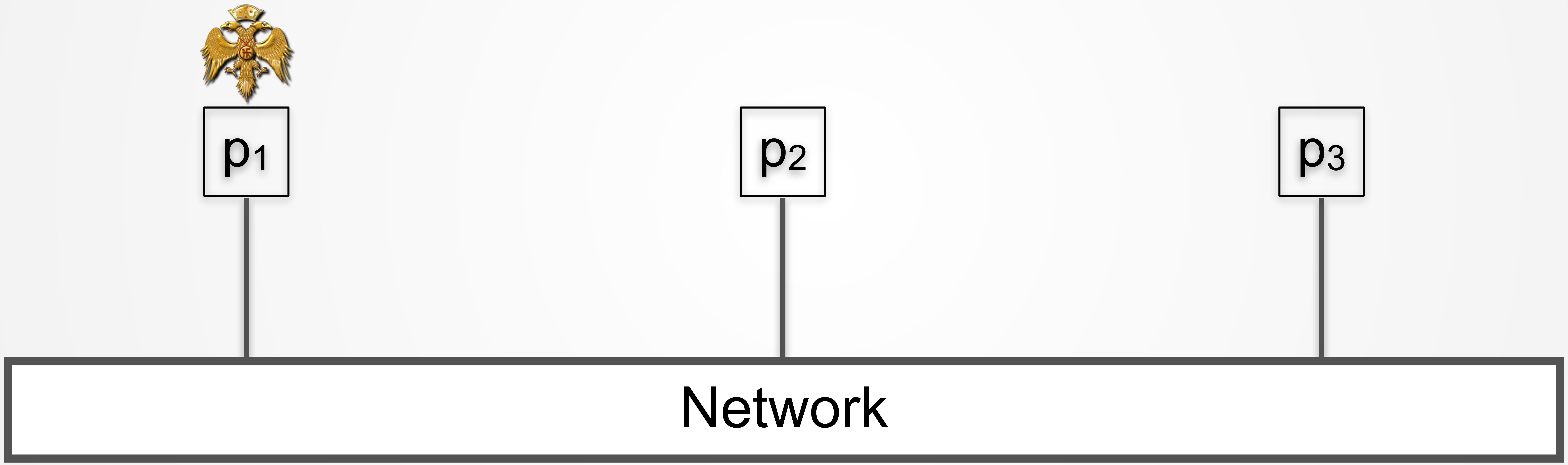
PROCESS FAILURES



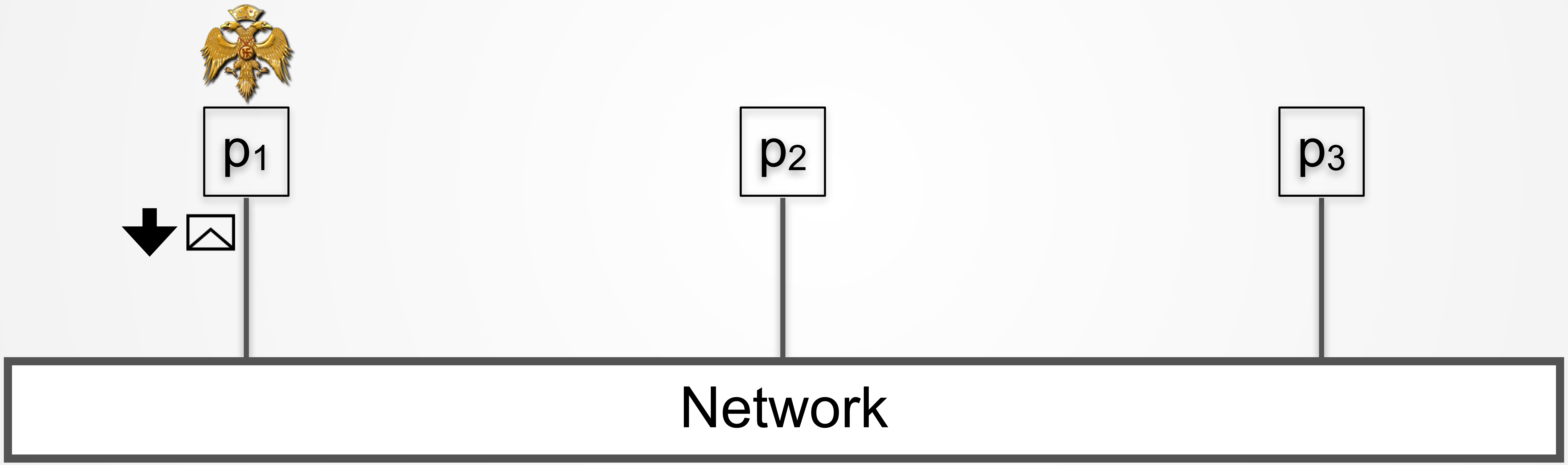
Process Failures



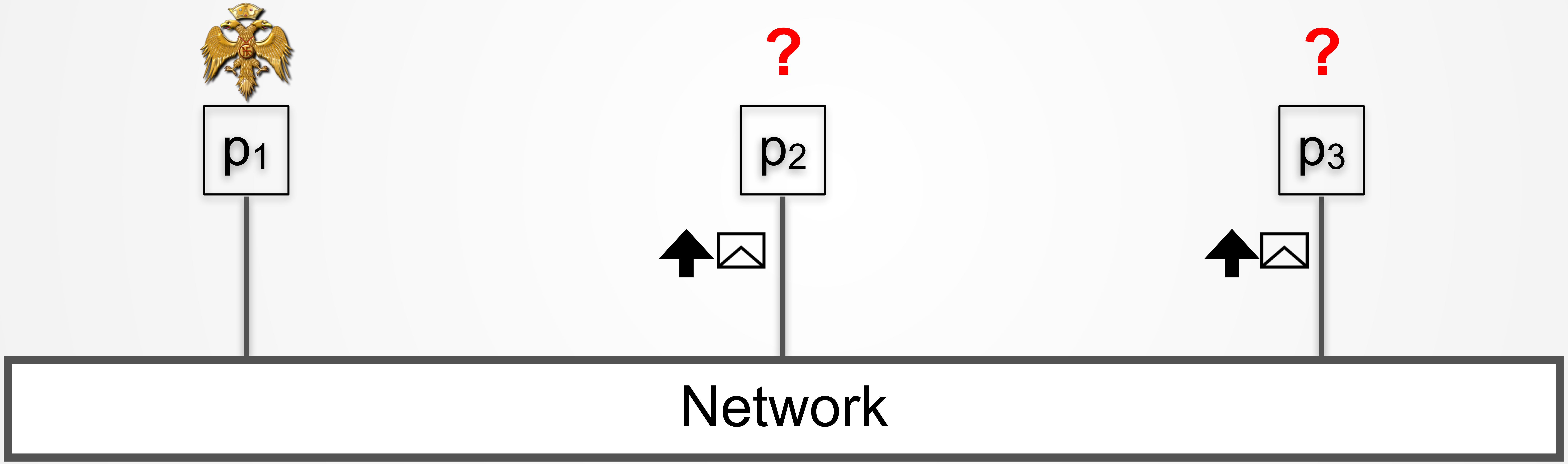
Byzantine Processes



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 essentially asynchronous

Impossibility of Consensus

Consensus is non-solvable in asynchronous systems if node 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

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!

Modeling a Distributed System

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

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

Self-Stabilizing Algorithms

Wont be covered in the course but cool to know.

- Robust algorithms that run forever
  - System might temporarily be incorrect
  - But eventually always becomes correct

- System can either be in a legitimate state or an illegitimate state (invariant)

  Self-stabilizing algorithm iff
  - Convergence
    - Given any illegitimate state, system eventually goes to a legitimate state
  - Closure
    - If system in a legitimate state, it remains in a legitimate state

SELF-STABILIZING EXAMPLE

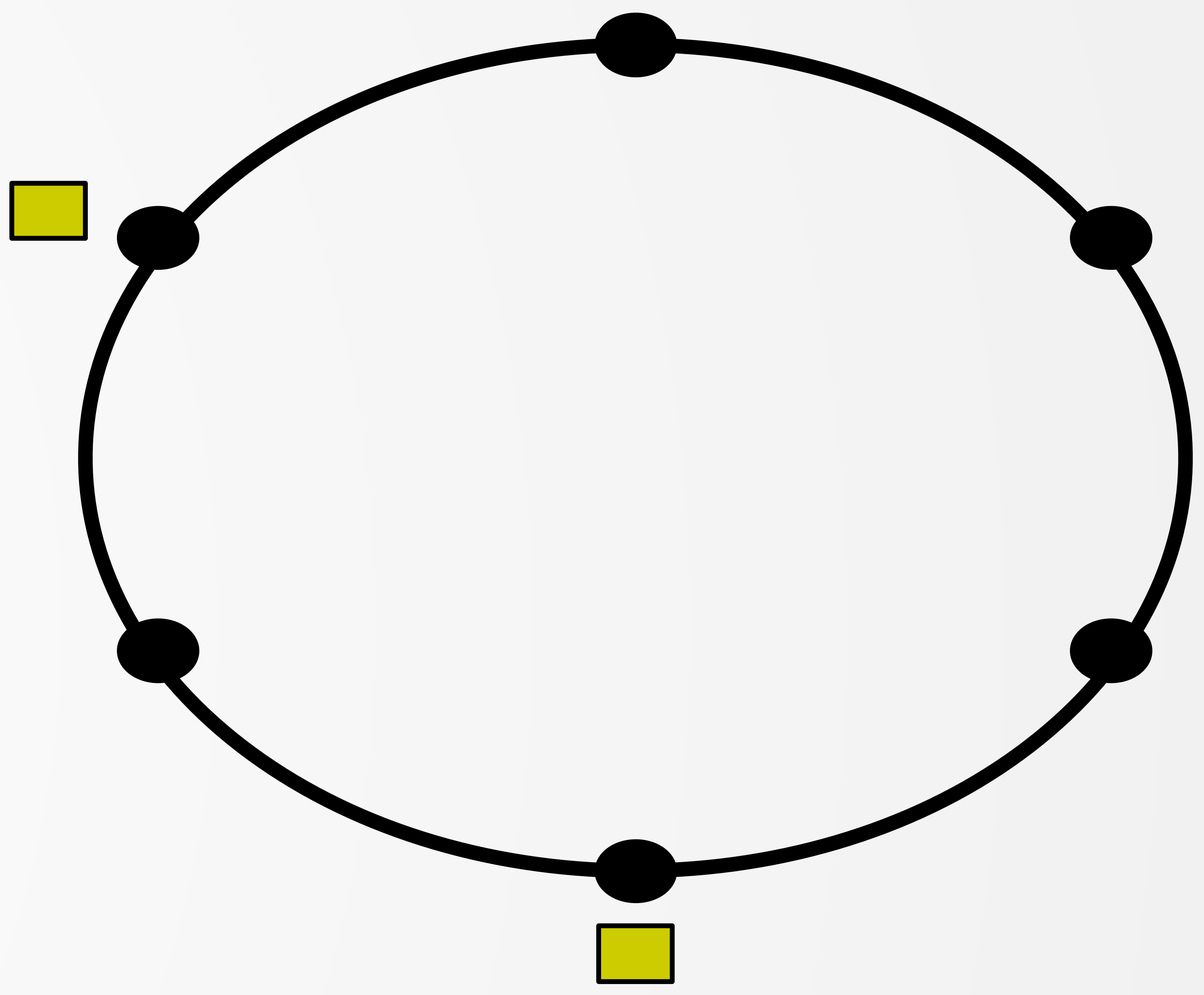
Token ring algorithm

Wish to have one token at all times circulating among processes

Self-Stabilization

Error leads to 2,3,… tokens

Ensure always 1 token eventually



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, Byzantine, self-stabilizing algorithms

Interesting **research** directions

- Large scale dynamic distributed systems