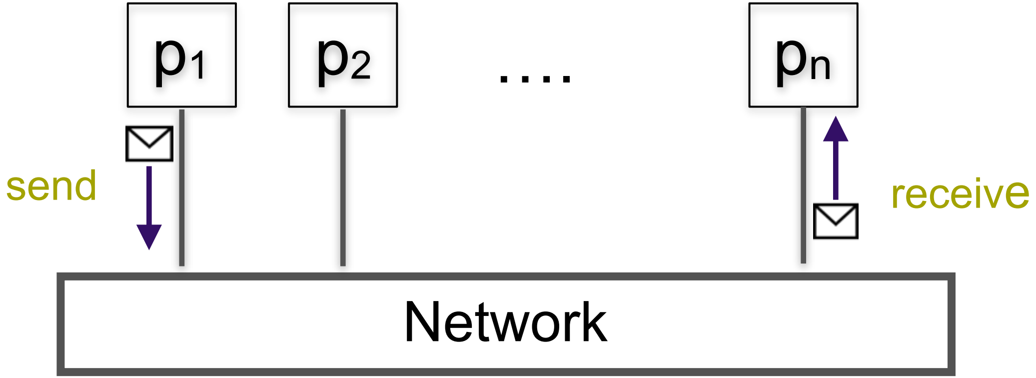
Introduction to Distributed Systems

Seif Haridi

haridi@kth.se

What is a distributed system?

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



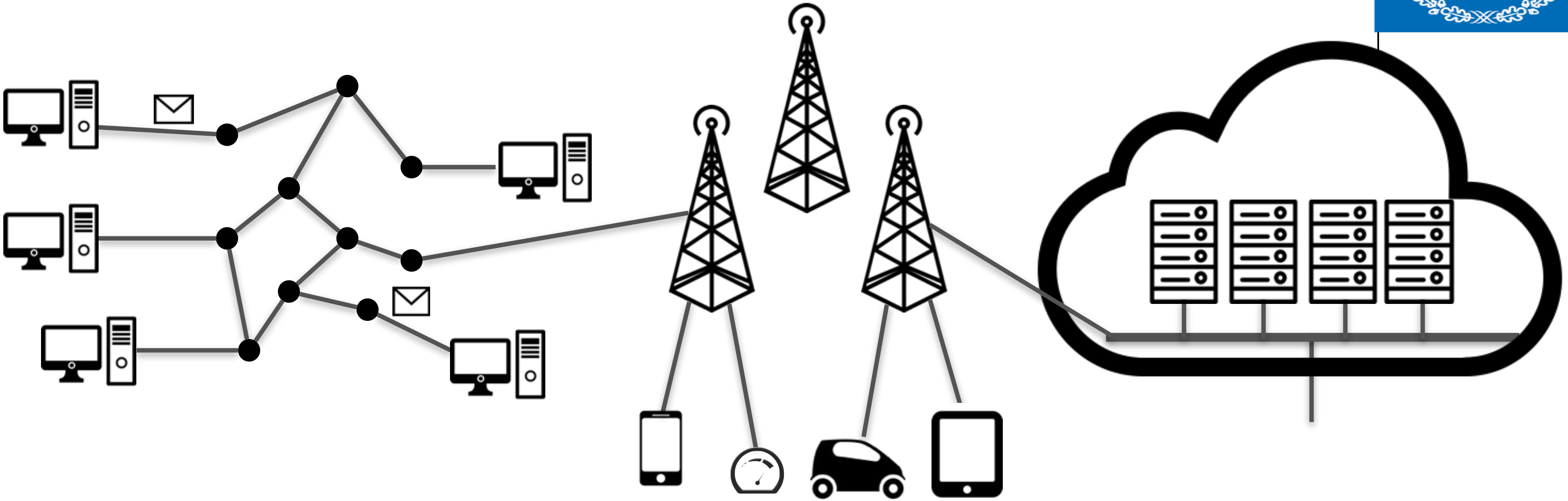
Our focus in this course

- Concepts
- Models
- 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?

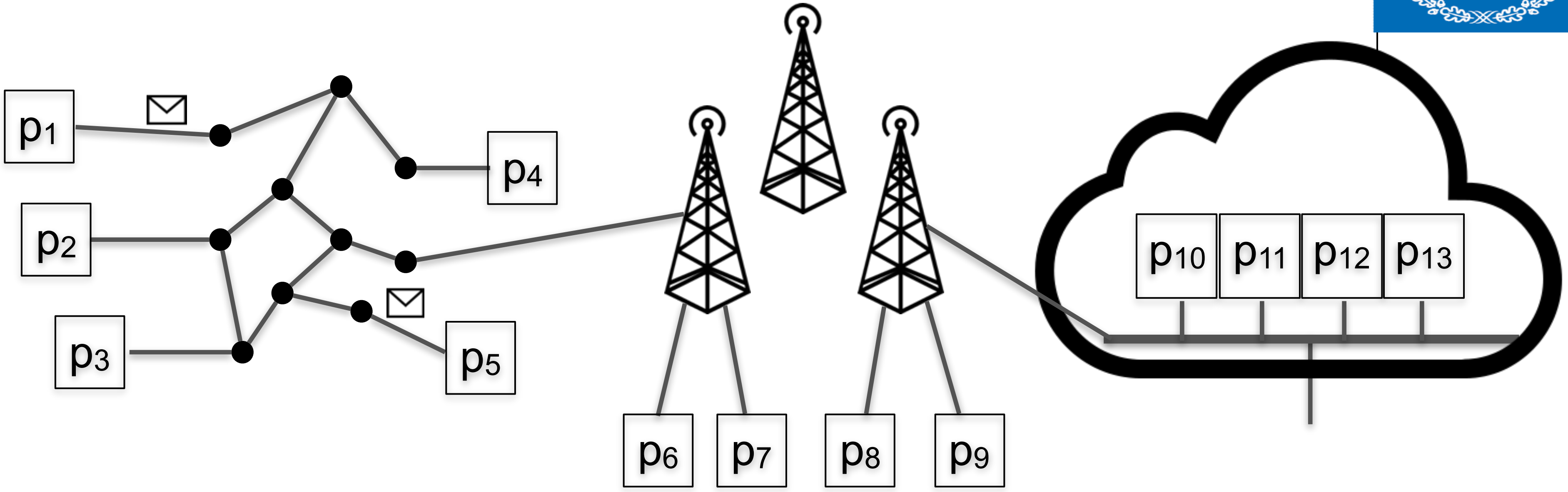
- It is important and useful
  - Societal importance
    - Internet
    - WWW
    - Cloud computing
    - Edge computing
    - Small devices (mobiles, sensors)

Why study distributed systems?



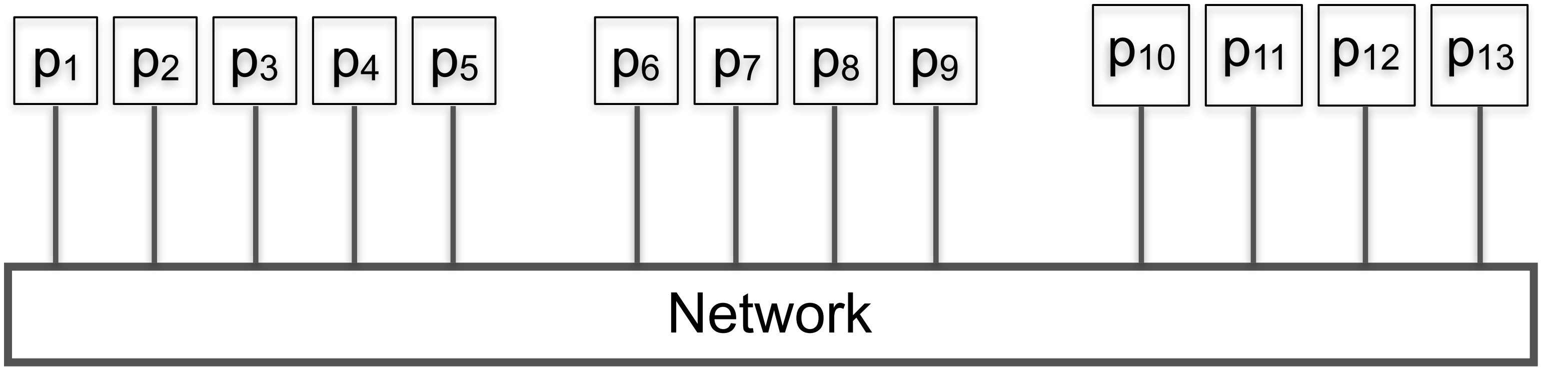
Internet Edge Computing Cloud Computing

Why study distributed systems?



Internet Edge Computing Cloud Computing

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

What types of problems are there?

Teaser: Two Generals' Problem

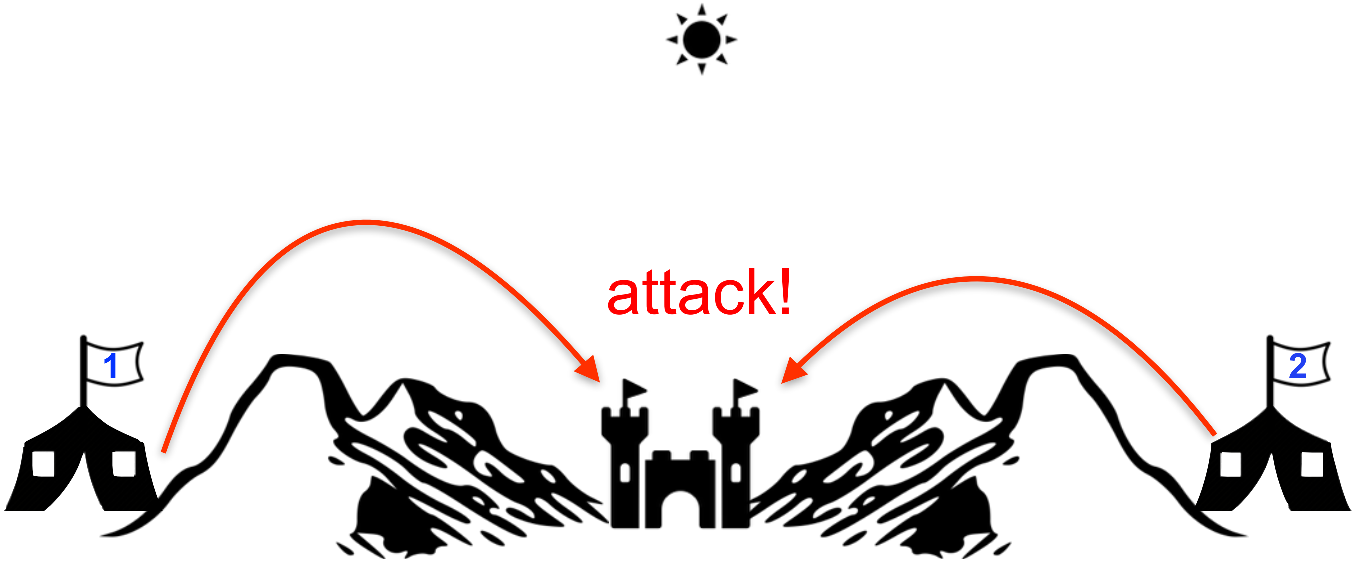
- Two generals need to coordinate an attack
  - Must agree on time to attack
  - They'll win only if they attack simultaneously
  - Communicate through messengers
  - Messengers may be killed on their way

Teaser: Two Generals' Problem

- Lets try to solve it for general g1 and g2
- g1 sends time of attack to g2
  - Problem: how to ensure g2 received msg?
  - Solution: let g2 ack receipt of msg
  - Problem: how to ensure g1 received ack
  - Solution: let g1 ack the receipt of the ack…
  - …
- This problem is impossible to solve!

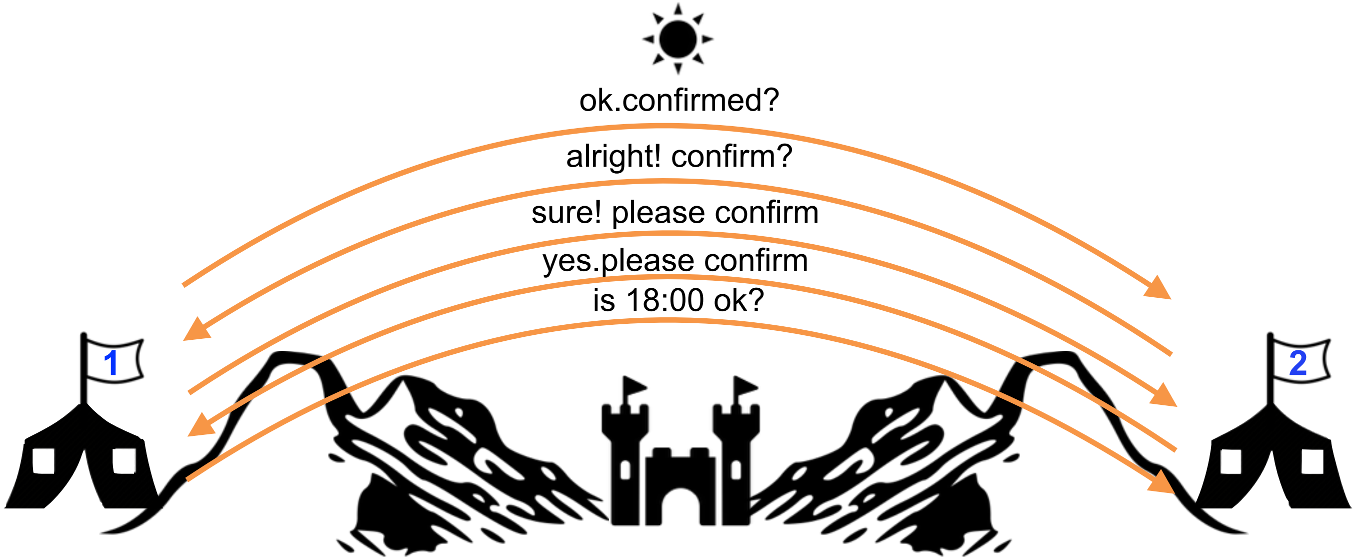
Teaser: Two Generals' Problem

Teaser: Two Generals' Problem

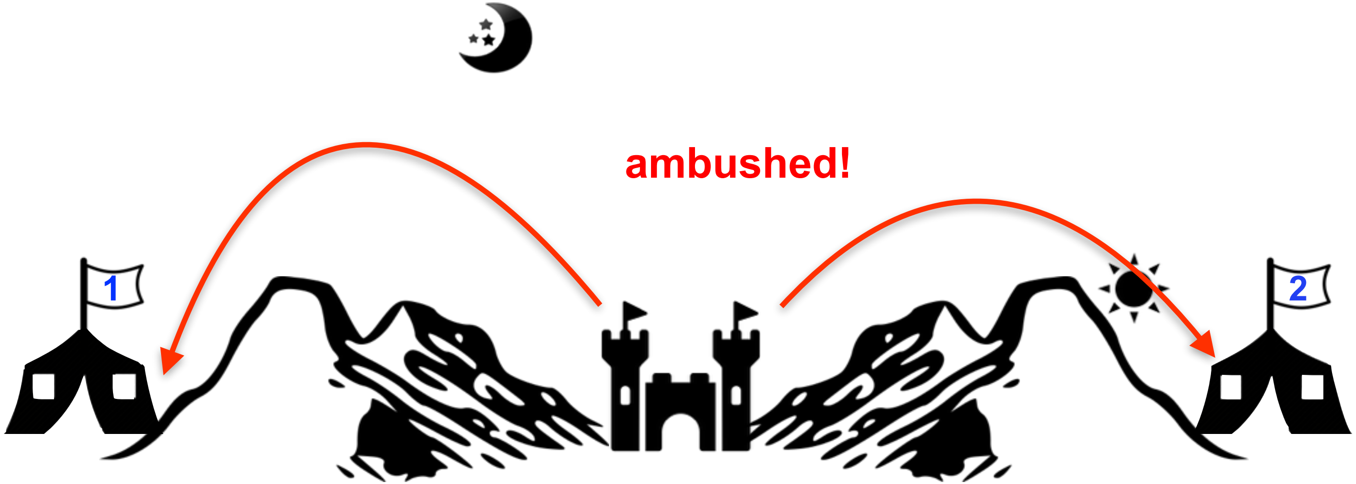


Teaser: Two Generals' Problem

Teaser: Two Generals' Problem



Teaser: Two Generals' Problem



Impossible to solve!

Teaser: Two Generals' Problem

- Applicability to distributed systems
  - Two nodes need to agree on a value before a specific time-bound
  - Communicate by messages using an unreliable channel

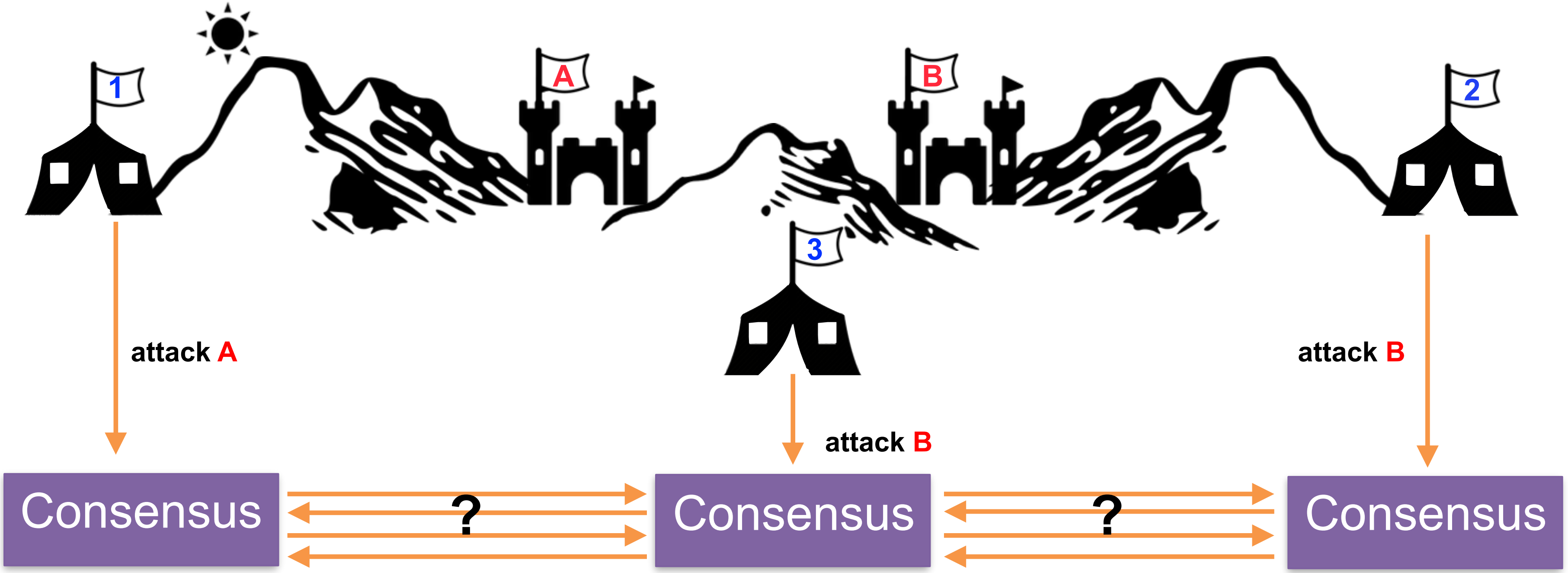
- Agreement is a core problem…

Consensus: agreeing on a number

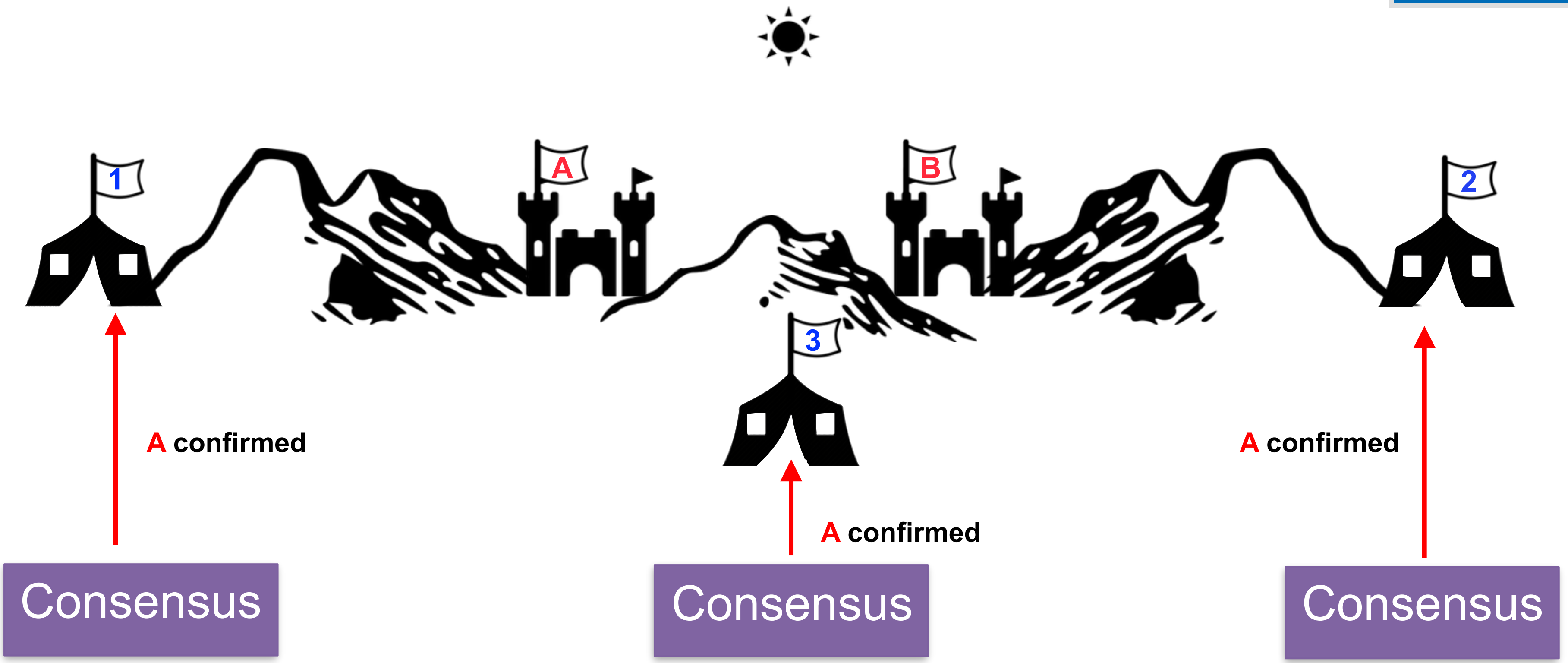
- Consensus problem
  - All nodes propose a value
  - Some nodes might crash & stop responding

- The algorithm must ensure:
  - All correct nodes eventually decide
  - Every node decides the same
  - Only decide on proposed values

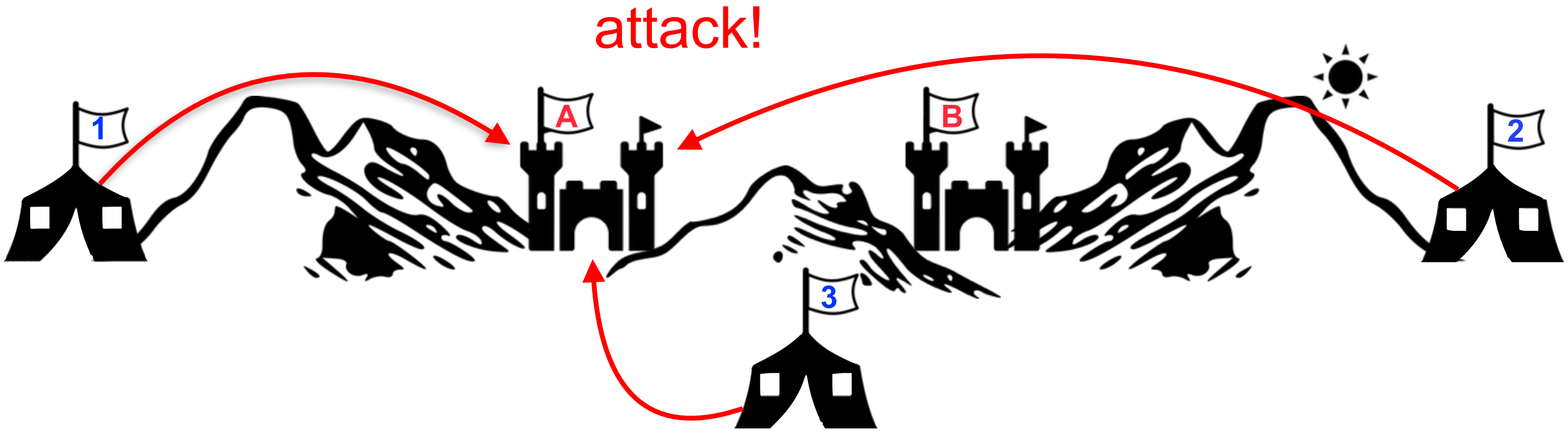
Example: Agreeing on a Target



Example: Agreeing on a Target



Example: Agreeing on a Target

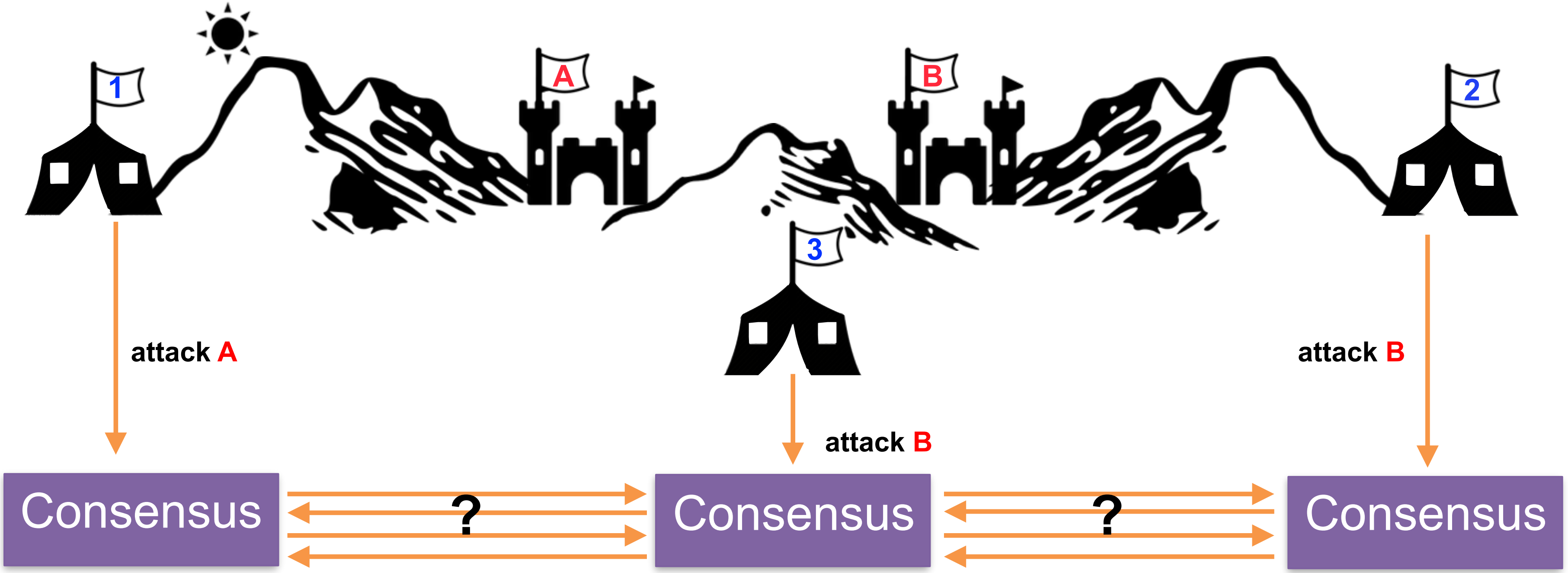


Consensus

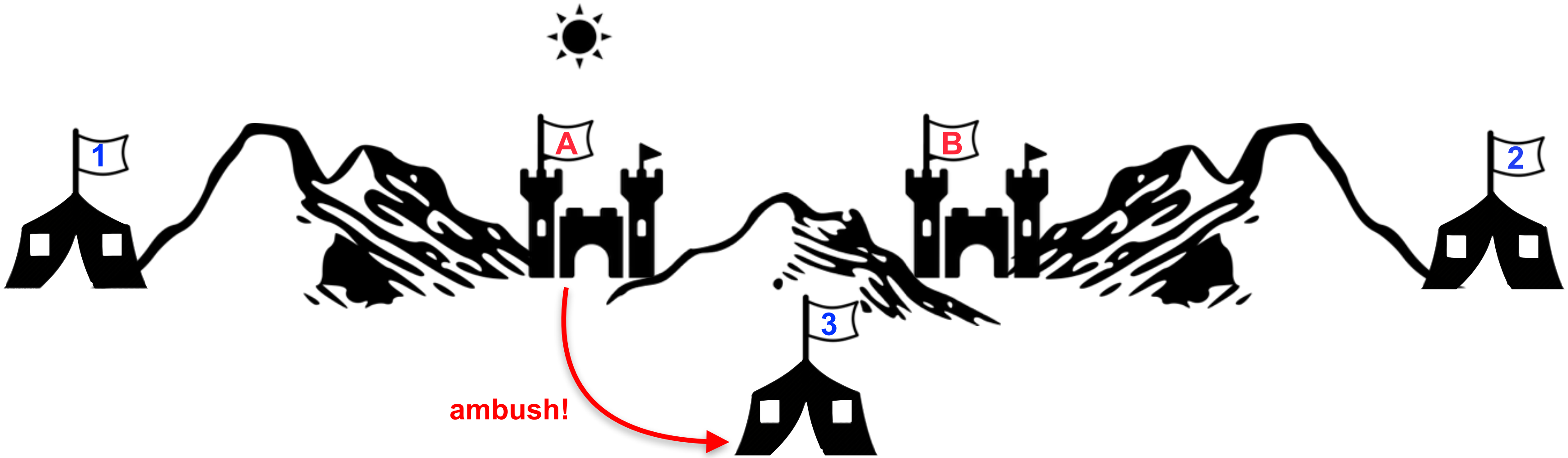
Consensus

Consensus

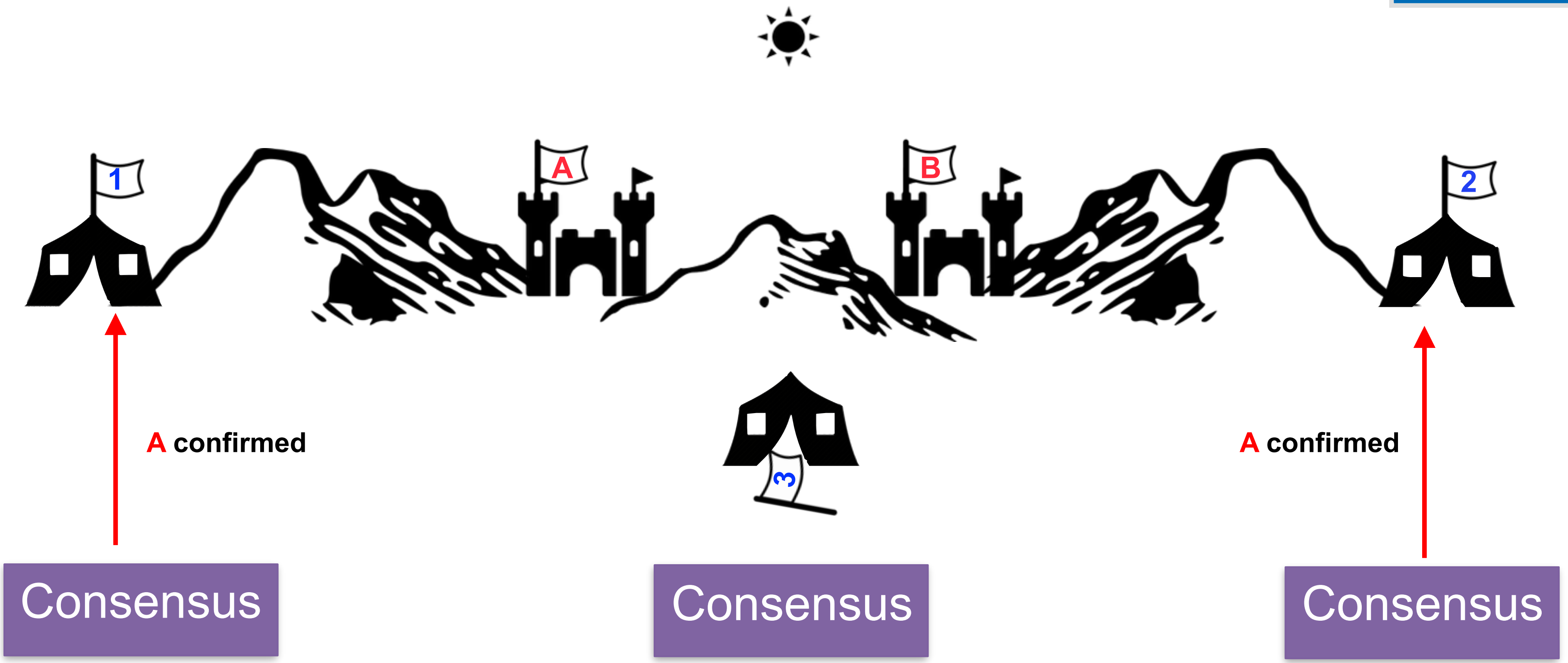
Example: Agreeing on a Target



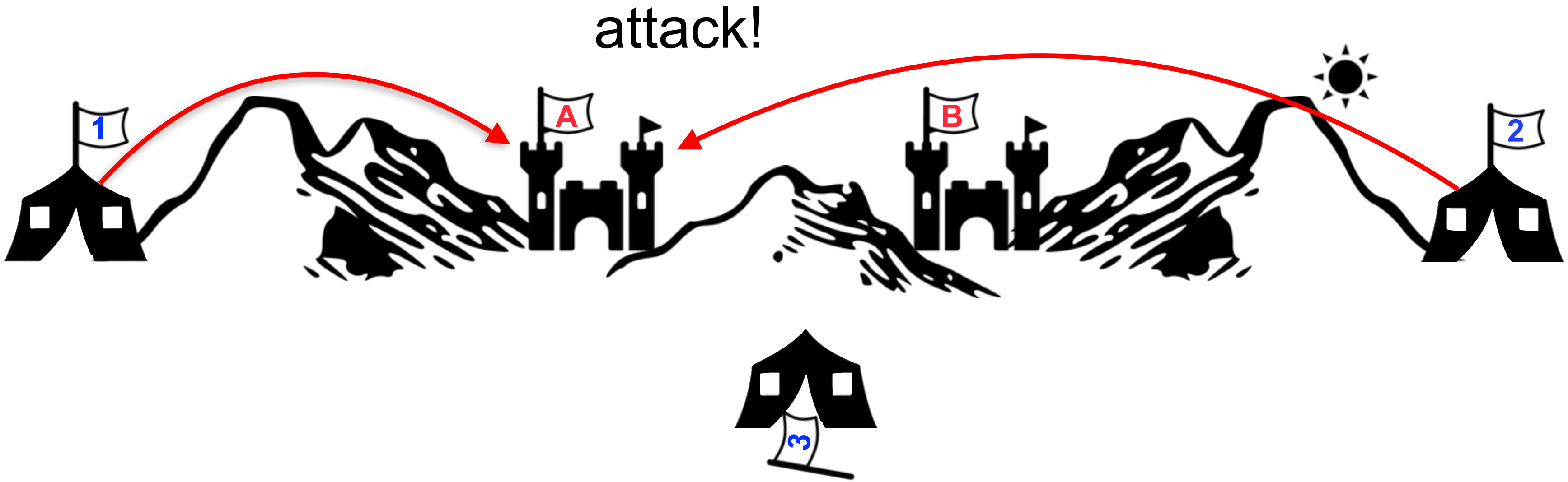
Example: Agreeing on a Target



Example: Agreeing on a Target



Example: Agreeing on a Target



Consensus

Consensus

Consensus

Is Consensus is 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

- Databases
  - Concurrent changes to same data
  - Nodes should agree on changes
- Use a kind 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 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

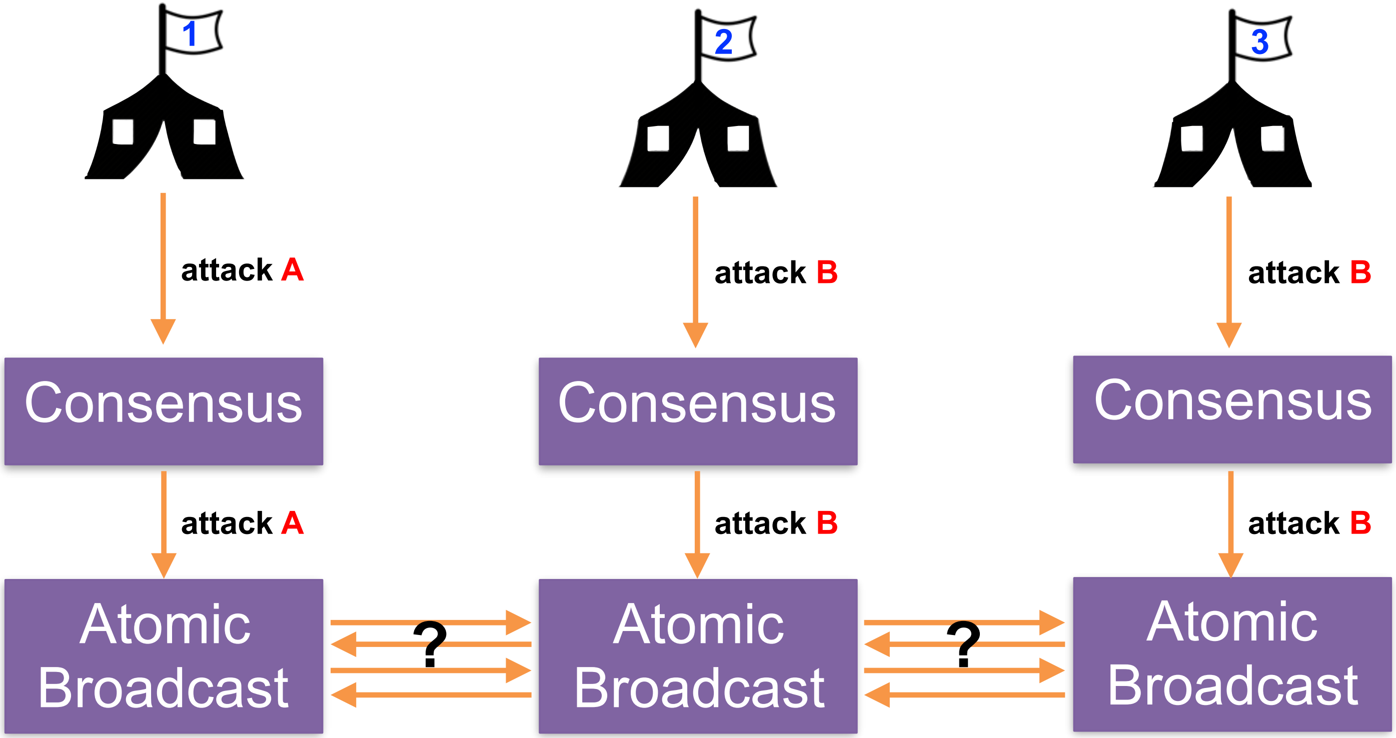
Atomic Broadcast ↔ Consensus

- Given Atomic broadcast
  - Can use it to solve Consensus

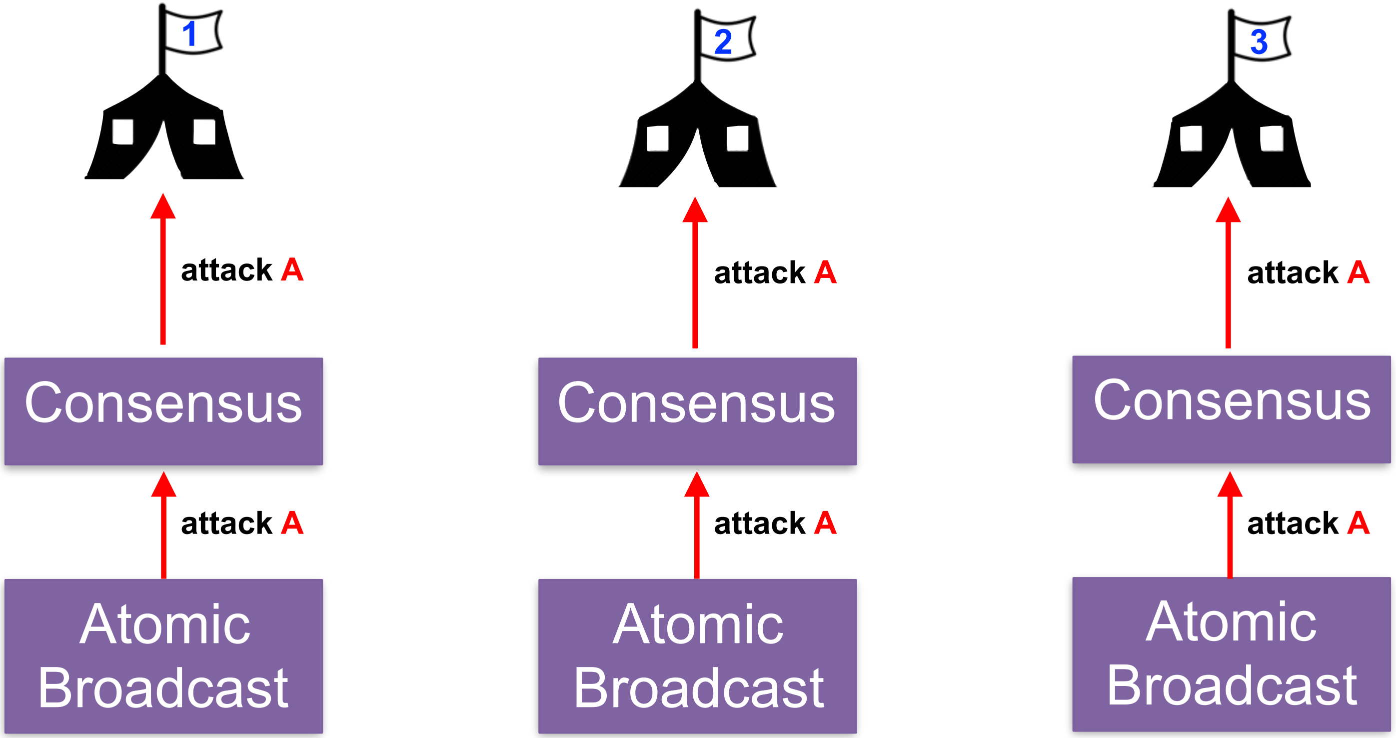
- Every node broadcasts its proposal
  - Decide on the first received proposal
  - Messages received in same order
    - All nodes will decide the same
- Given Consensus
  - Can use it to solve Atomic broadcast **[d]**

- Atomic Broadcast equivalent to Consensus

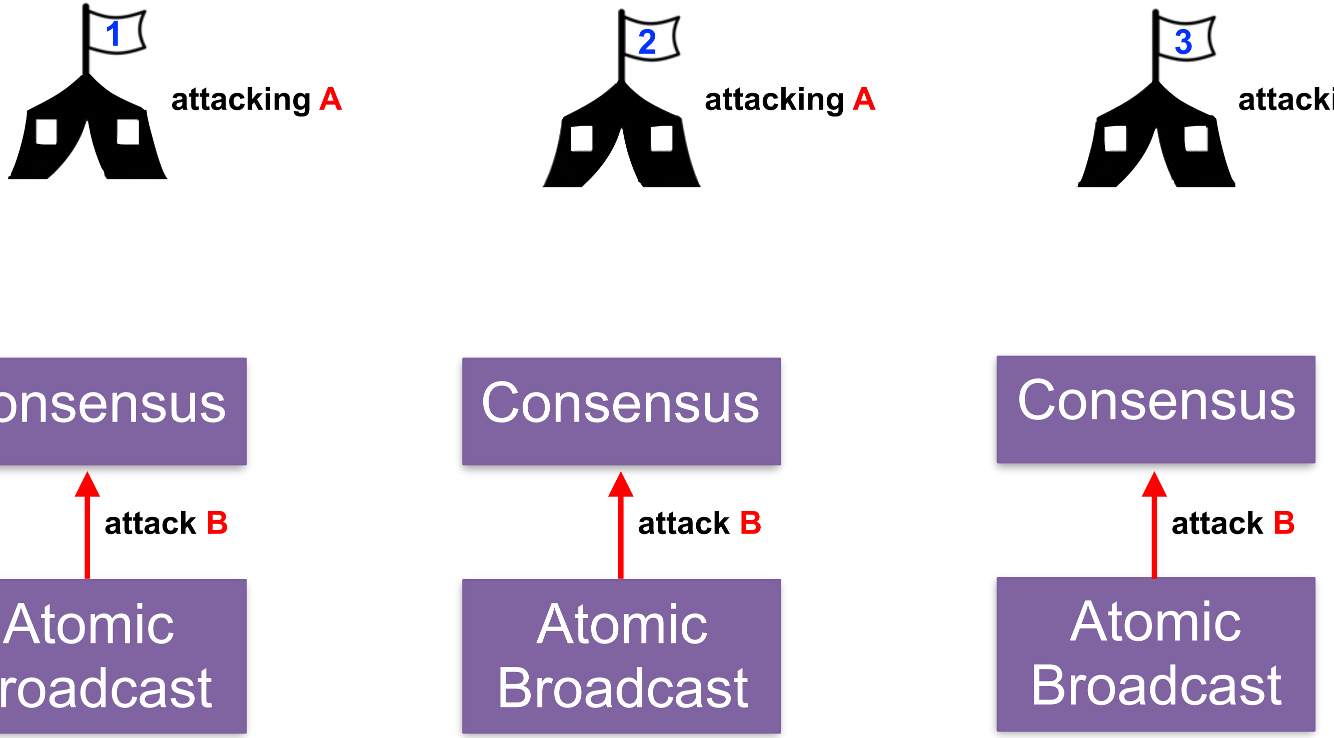
Atomic Broadcast ↔ Consensus



Atomic Broadcast ↔ Consensus



Atomic Broadcast ↔ Consensus



Models

How to reason about them?

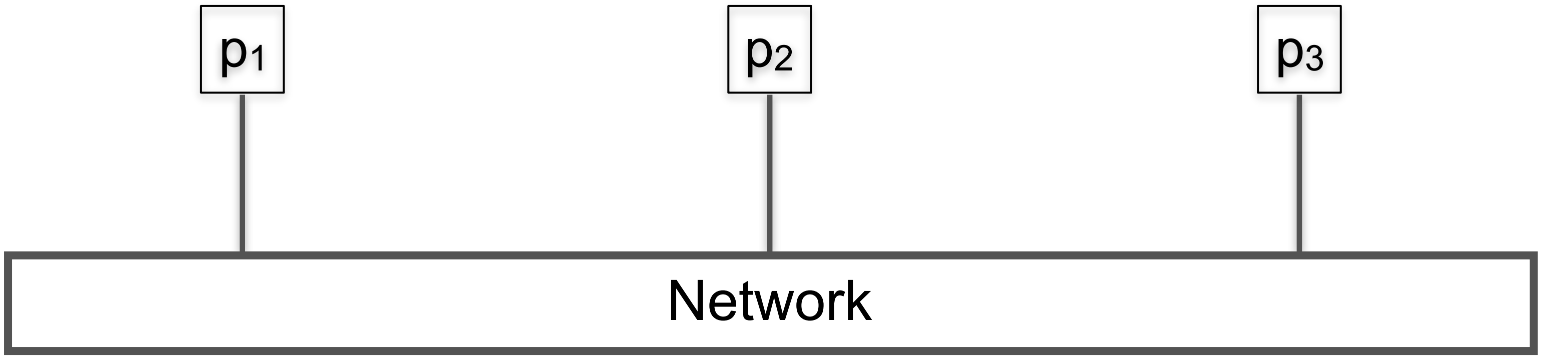
Modeling a Distributed System

- 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

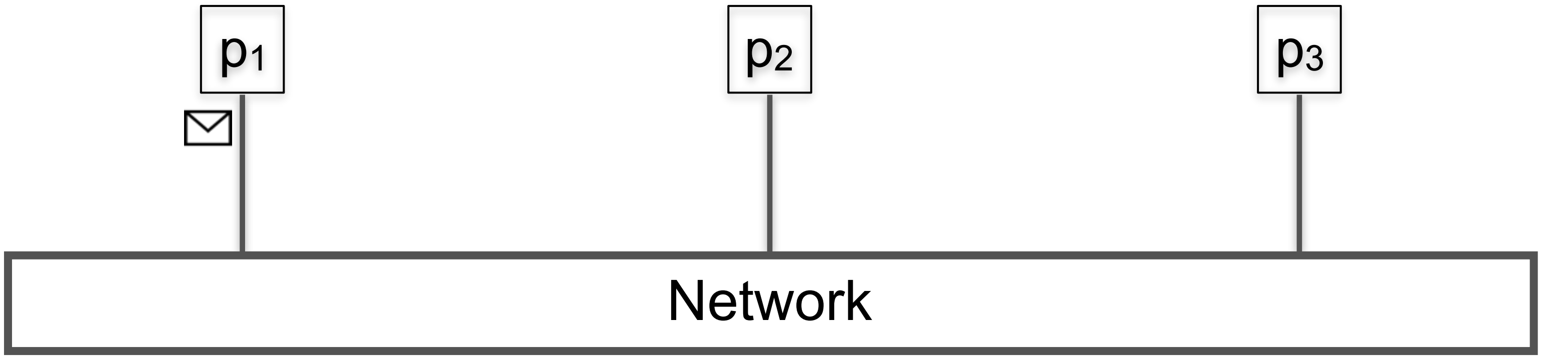
Modeling a Distributed System

- Failure assumptions
  - Processes
    - What kind of failure a process can exhibit?
    - Crashes and stops
    - Behaves arbitrary (Byzantine)
  - Network
    - Can a network channel drop messages?

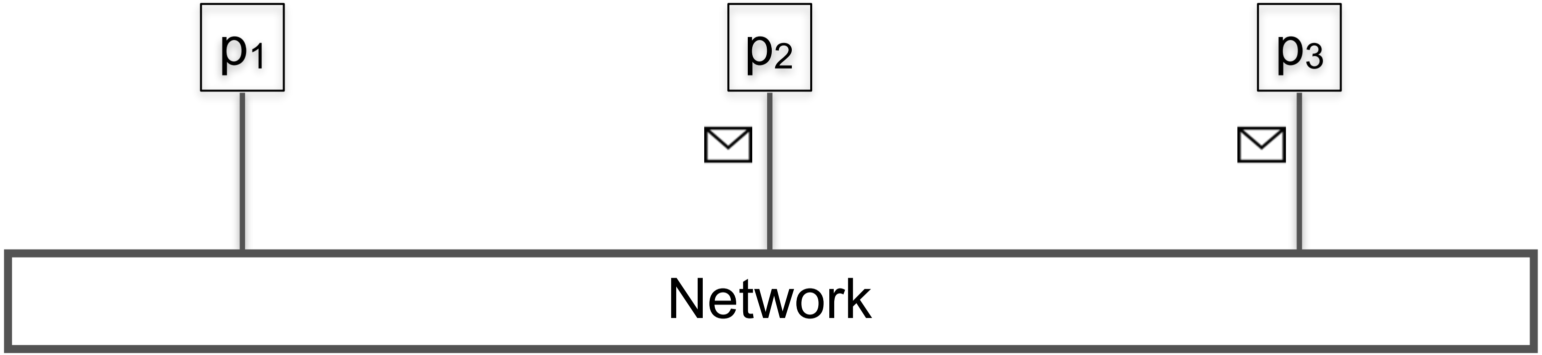
Modeling a Distributed System



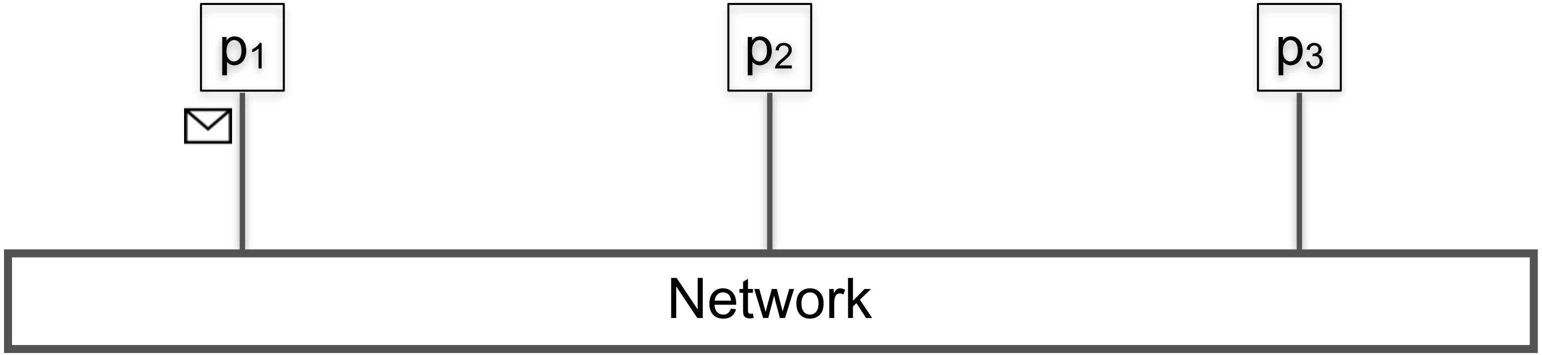
Modeling a Distributed System



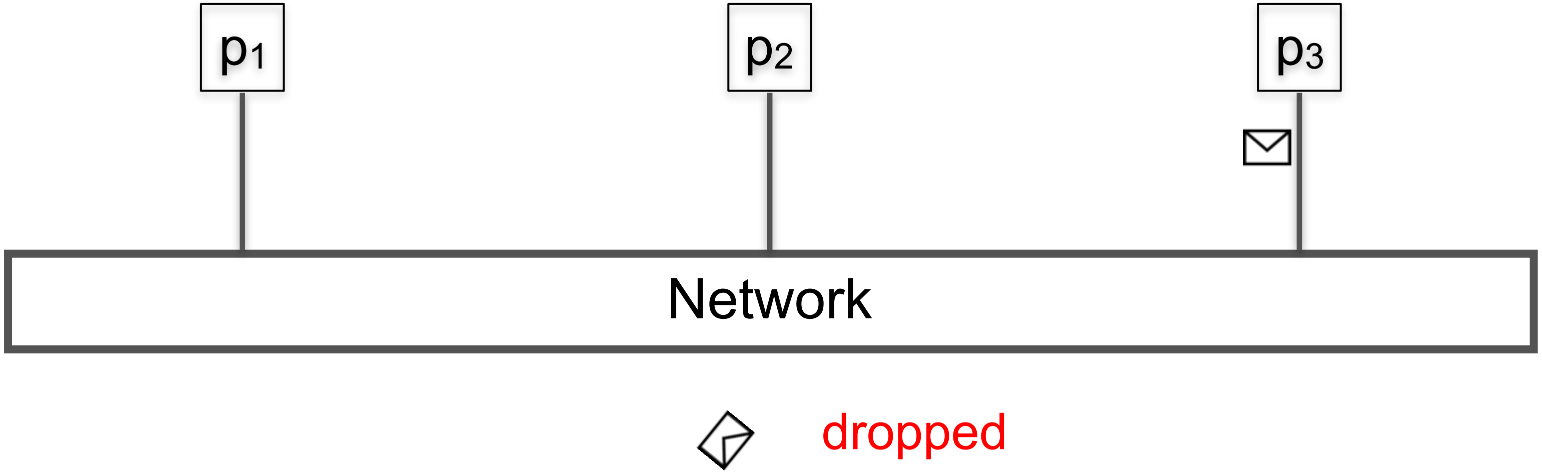
Modeling a Distributed System



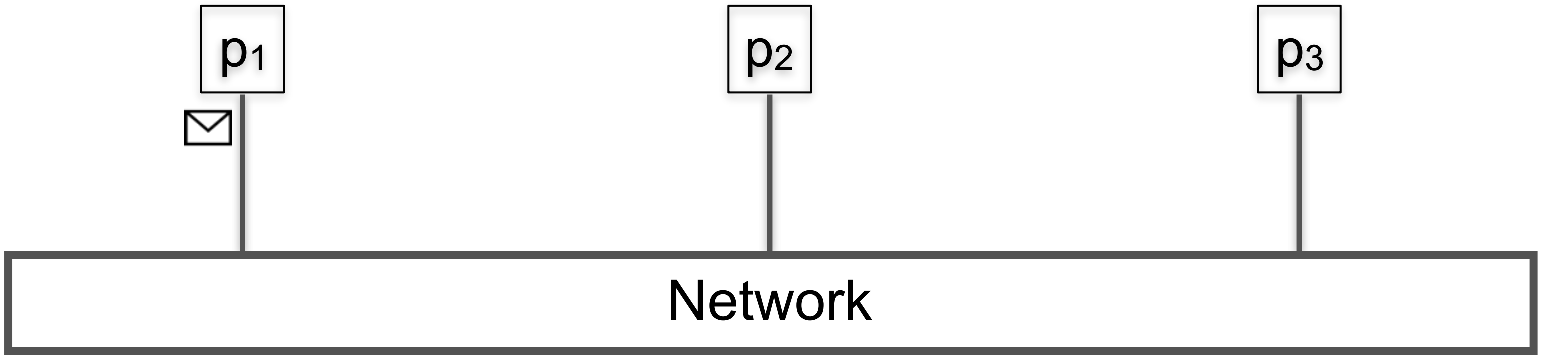
Network Failures



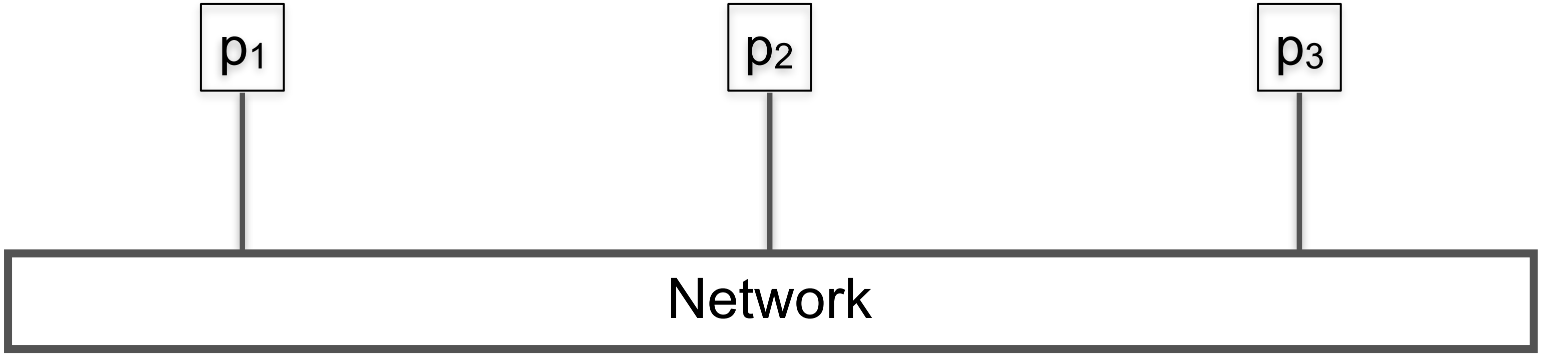
Network Failures



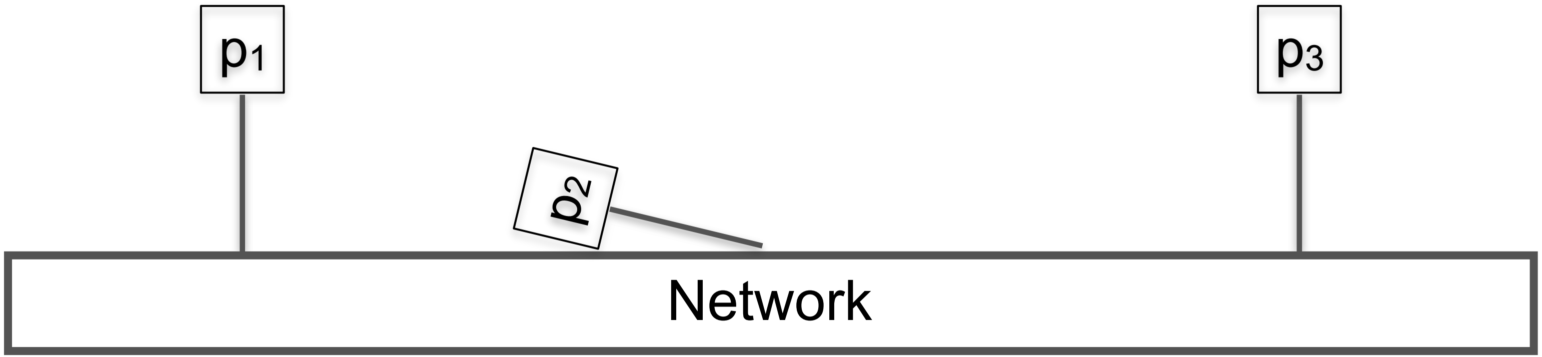
Process Failures



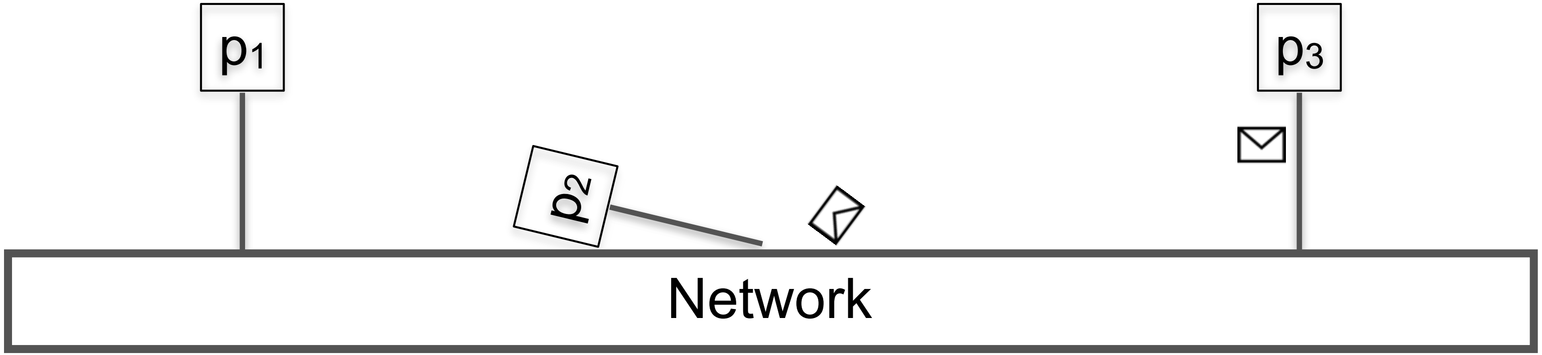
Process Failures



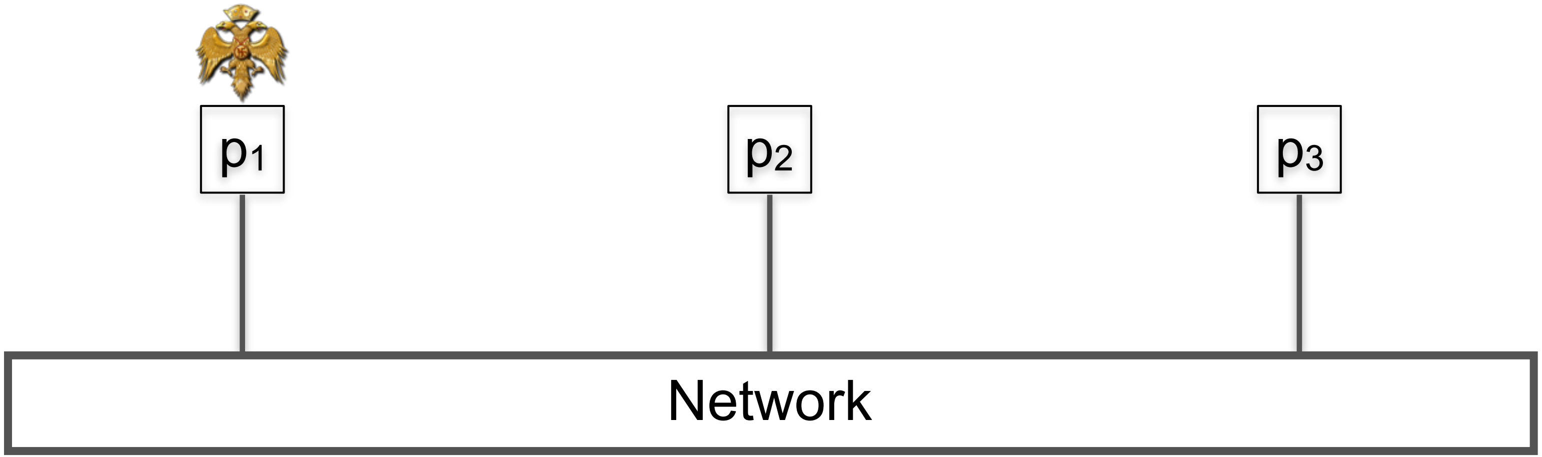
Process Failures



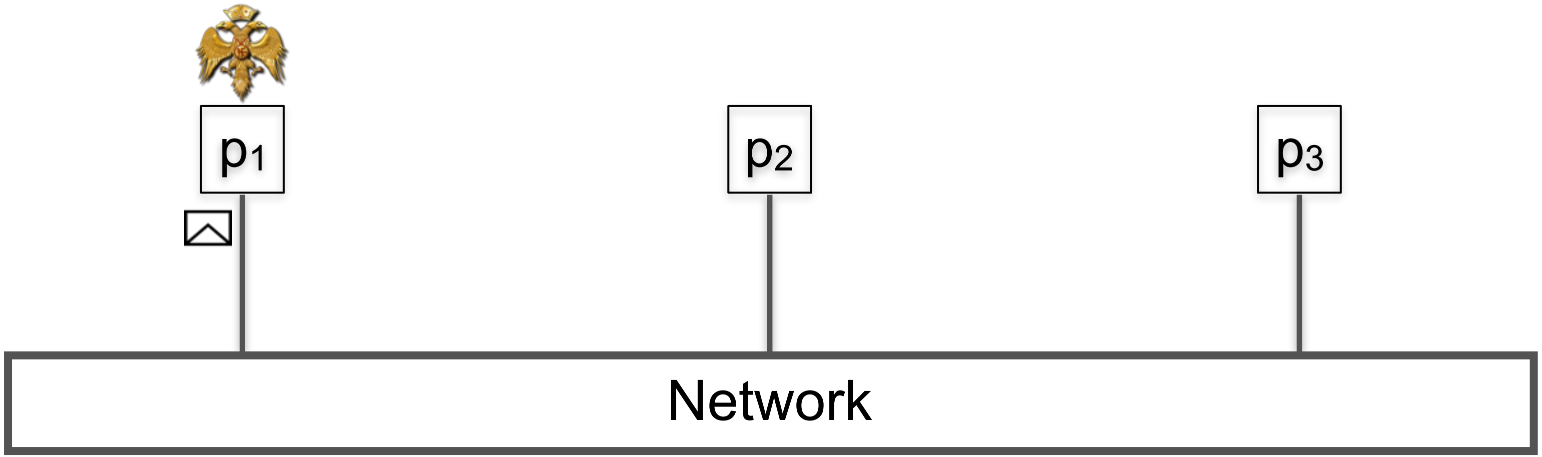
Process Failures



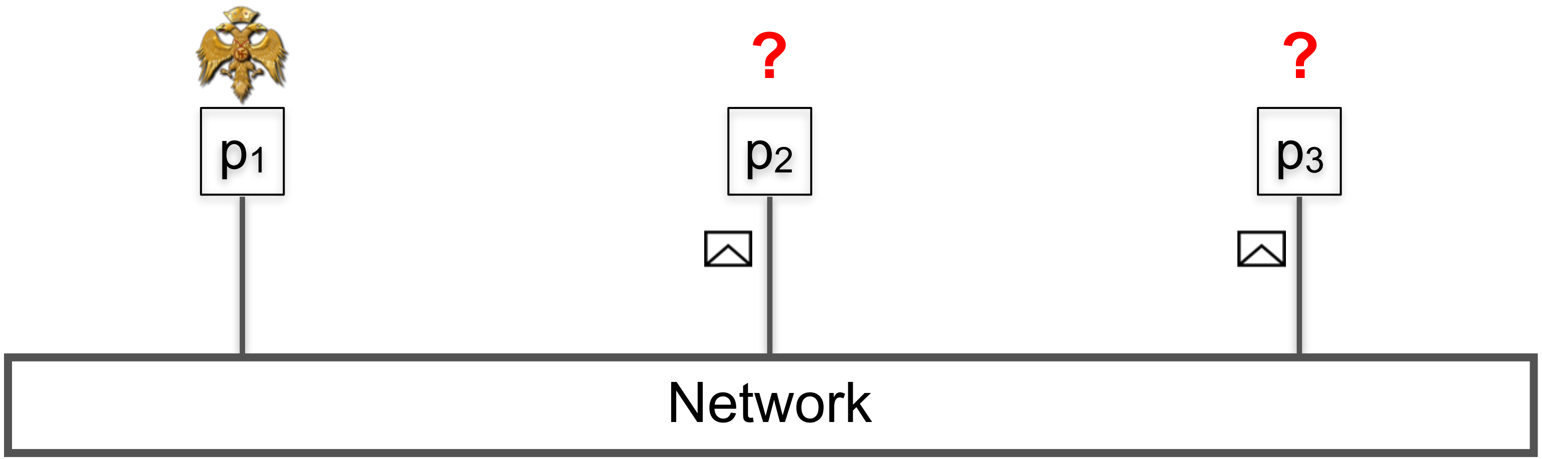
Byzantine Processes



Byzantine Processes



Byzantine Processes



Modeling a Distributed System

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

- Internet essentially asynchronous

Impossibility of Consensus

- Consensus cannot be solved in asynchronous system
  - If a single node may crash
- Implications on
  - Atomic broadcast
  - Atomic commit
  - Leader election
  - …

Modeling 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
  - Multicore computers

Possibility of Consensus

- Consensus 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?

Modeling a Distributed System

- But Internet is mostly synchronous
  - 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 partially synchronous system
  - with up to N/2 crashes
- Useful for Internet?

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

- Realistic model Internet

Conclusions

Topics not covered

Processes always crash?

- Other types of failures
  - Not just crash stops

- Byzantine faults
- Self-stabilizing algorithms

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

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

- System can either by in a legitimate state or an illegitimate state
- 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 Algorithms

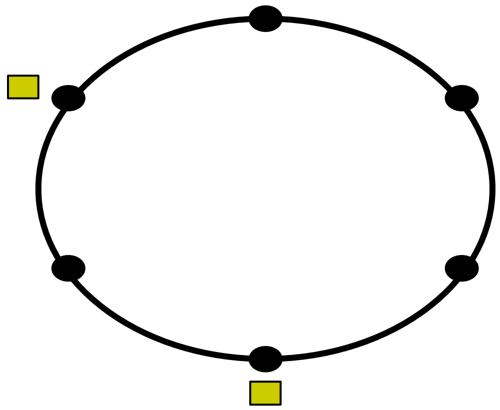
- System can either by in a legitimate state or an illegitimate state
- Self-stabilizing algorithm iff
  - Convergence
  - Closure

Self-stabilizing Algorithms

- Advantages
  - Robust to transient failures
  - Don’t need initialization
  - Can be easily composed
    - A service composed of two self-stabilizing services is self-stabilizing service

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



Content of the Course

Content I

- Formal Models of Asynchronous Systems
- Basic Abstractions
- Reliable Broadcast Algorithms
- Distributed Shared Store and Consistency Models

Content II

- Single Value Consensus
  - Paxos algorithm
- Sequence Consensus
  - Multi-Paxos
  - Replicated State Machines (RSM)
- Dynamic Reconfiguration
- Physical Clocks
  - Leader election (timed asynchronous model)
  - More efficient RSM)
  - Shared stores with Strong Consistency
- Relaxed consistency models
  - CAP theorem

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

Let’s start