Distributed Systems 1

CUCS Course 4113
https://systems.cs.columbia.edu/ds1-class/

Instructor: Roxana Geambasu

Agreement in Distributed Systems

Agreement

- *A set of nodes in a DS often need to agree on something: a decision, the value of a variable, order of events,...*
- Examples?

Agreement

- *A set of nodes in a DS often need to agree on something: a decision, the value of a variable, order of events,...*
- Examples:
  - Lamport's distributed mutual exclusion protocol: nodes agree on who has the lock at any time.
  - ATM example from RPC: ATM front-end and banking service need to agree on whether to commit or abort my cash withdrawal.
  - In HW2, you design a replication protocol where primary and secondary replicas of a key-value store agree on the sequence of values written at each key.

Two Types of Agreement

1. **Consensus:** participants need to agree on a value, but they are willing and capable to accept *any value*.
2. **Atomic commitment:** participants need to agree on a value, but they have specific constraints on whether they can accept any particular value.

- Examples:
  - Decision of when to meet is likely a ??? problem.
  - Decision of which zoom link to meet at is likely a ??? problem.

Two Types of Agreement

1. **Consensus:** participants need to agree on a value, but they are willing and capable to accept *any value*.
2. **Atomic commitment:** participants need to agree on a value, but they have specific constraints on whether they can accept any particular value.

- Examples:
  - Decision of when to meet is likely **atomic commitment**.
  - Decision of which zoom link to meet at is likely **consensus**.

Examples – Which Type is Each?

- Lamport's distributed mutual exclusion protocol: nodes agree on who has the lock at any time. ← ???
- ATM example from RPC lecture: ATM front-end and banking service need to agree on whether to commit or abort my cash withdrawal. ← ???
- In HW2, you will design a replication protocol where primary and secondary replicas of a key-value store agree on the sequence of values written at each key. ← ???

Examples – Which Type is Each?

- Lamport's distributed mutual exclusion protocol: nodes agree on who has the lock at any time. ← **Consensus**
- ATM example from RPC lecture: ATM front-end and banking service need to agree on whether to commit or abort my cash withdrawal. ← **Atomic commitment**
- In HW2, you will design a replication protocol where primary and secondary replicas of a key-value store agree on the sequence of values written at each key. ← **Consensus**

Agreement Is “Hard”

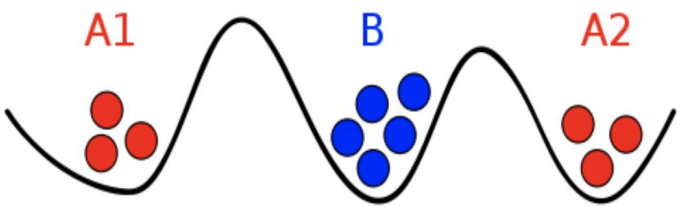
- In the asynchronous system model, it is impossible to guarantee agreement in finite time under *all failure scenarios*.
- The consensus problem can be approached in practice: there exist protocols to solve consensus under vast majority of plausible failure scenarios.
- That’s not the case for atomic commitment: if each participant has their own constraints, then you can’t tolerate any one participant’s failure.
- In that sense, atomic commitment is “even harder” than consensus.

Remainder of This Class

Focus on **consensus**:

- Justify impossibility intuitively with a standard example: the two generals problem.
- Formulate the consensus problem more abstractly.
- Discuss the 1985 impossibility result by Fischer, Lynch, and Paterson (FLP) that showed that consensus is fundamentally impossible to guarantee in asynchronous systems.

The Two-Generals Problem

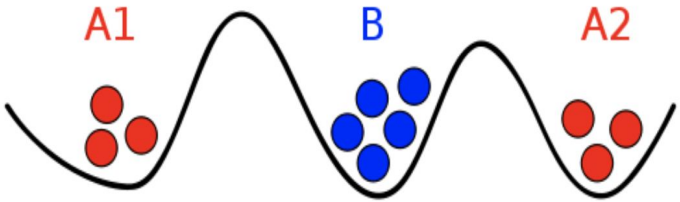


- Two armies, A1 and A2, want to attack a fortified city, B.
- Both armies must attack at the same time to succeed.
- The armies can communicate through messengers, but those can be captured or delayed, so msg. delivery is unreliable.

The Two-Generals Problem

Three requirements for a solution:

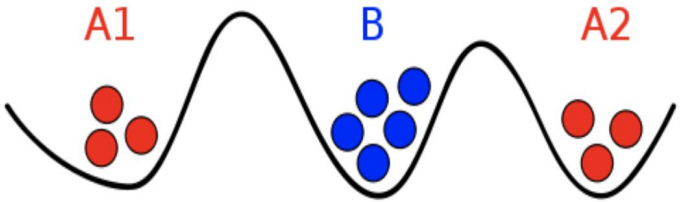
1. **Consistency**: both armies decide to attack at the same time.
2. **Termination:** each army decides to attack after a finite number of messages.
3. **Validity:** the time to attack was proposed by one of the armies.



The Two-Generals Problem

Three requirements for a solution:

1. **Consistency**: both armies decide to attack at the same time.
2. **Termination:** each army decides to attack after a finite number of messages.
3. **Validity:** the time to attack was proposed by one of the armies.

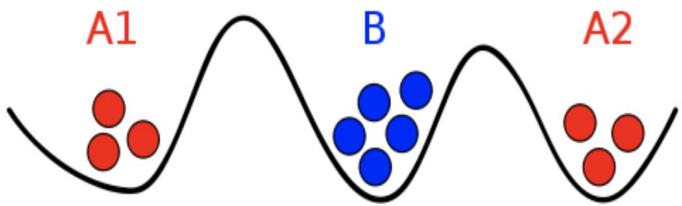


Question: What should be the protocol to achieve these properties?

Case 1: Known Delays, Reliable Delivery (synchronous system model)

Protocol:

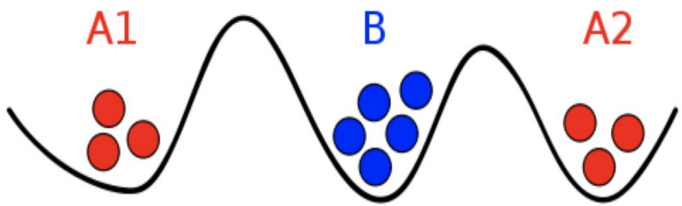
- Pre-agree on either A1 or A2 generals proposing the time to attack. Say A1 is the one to propose. A2 will be the one to accept.
- A1 sets the time of attack to communication delay + some extra time to account for A2's preparation for response.



Case 1: Known Delays, Reliable Delivery (synchronous system model)

Protocol:

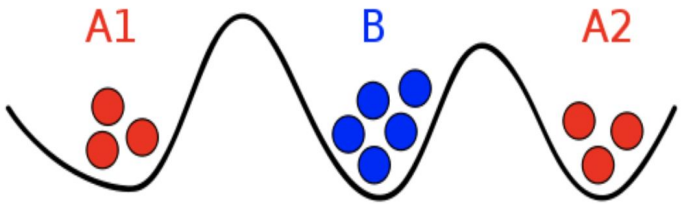
- Pre-agree on either A1 or A2 generals proposing the time to attack. Say A1 is the one to propose. A2 will be the one to accept.
- A1 sets the time of attack to communication delay + some extra time to account for A2's preparation for response.



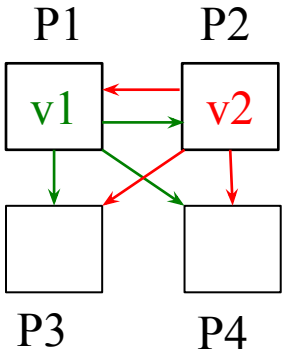
So problem is solvable in synchronous networks.

Case 2: Unknown Delays / Unreliable Delivery (asynchronous system model)

- Achieving consistency, termination, and validity in the asynchronous model is provably impossible.
- Sketch:
  - Need Acks in the protocol.
  - But Acks can be delayed/lost too.
  - Therefore I need more Acks.
  - Therefore, one general can never be sure the other will attack.
  - So they can't be guaranteed to reach agreement.

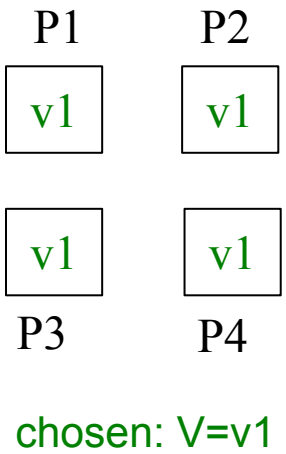


Consensus Problem Formulation



- A collection of processes, P_i.
- They propose values V_i (e.g., time to attack, client update, lock requests, …), and send messages to others to exchange proposals.
- Different processes may propose different values, and they can all accept any of the proposed values.
- Only one of the proposed values, *V*, will be "chosen" and eventually all processes learn that *one chosen value*.

Consensus Problem Formulation



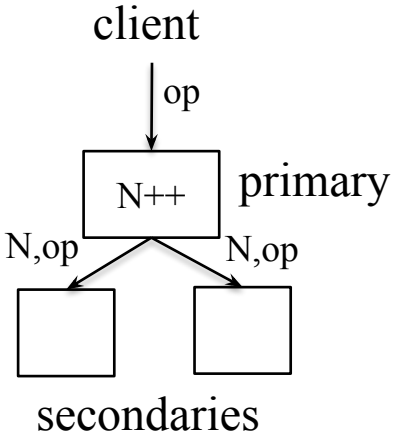
- A collection of processes, P_i.
- They propose values V_i (e.g., time to attack, client update, lock requests, …), and send messages to others to exchange proposals.
- Different processes may propose different values, and they can all accept any of the proposed values.
- Only one of the proposed values, *V*, will be "chosen" and eventually all processes learn that *one chosen value*.

Consensus Problem Formulation

- Three requirements for a solution:
  - **consistency:** once a value is chosen, the chosen value of all working processes is the same.
  - **termination:** eventually they agree on a value (a.k.a., a value is “chosen”).
  - **validity:** the chosen value was proposed by one of the nodes.

Consensus Is Pervasive in DS

- Agreeing on order of updates to replicated DB.
  - One solution is primary/secondaries replication (like you have in HW2).
    - There are several replicas, one is *primary*.
    - Reads and writes are accepted only by primary, which establishes an order for all operations before forwarding them to secondaries.
    - Multiple variants exist, but they all reduce to one core consensus question: **how to choose the primary**? A.k.a. *leader election*.



Consensus Is Pervasive in DS

- Grabbing a lock for mutual exclusion.
  - Previously we looked at a specific algorithm for distributed mutual exclusion. At its core, it's a consensus problem, which can be addressed with a generic consensus algorithm.
- Reliable and ordered multicast: all members of a group agree on a set and order of messages to receive.

- … Many other examples. **One protocol that solves consensus can solve them all!**

The FLP Impossibility Result

- **In an asynchronous system (unordered messages, unbounded communication delays, unbounded processing delays), no protocol can guarantee consensus within a finite amount of time if even a single process can fail by stopping.** [FLP-1985]

- But, there are approximate solutions to the problem that solve consensus in all but the exceedingly rare events. We'll look at such a protocol, Paxos, in future lectures.

When Can Consensus Be Guaranteed?

- **Determining factors:**
  - Processors: synchronous vs. asynchronous.
  - Communication: bounded vs. unbounded.
  - Messages: ordered vs. unordered.
  - Transmission: broadcast vs. point-to-point.

Can's and Cannot's

Processors	Message Order: Unordered		Message Order: Ordered		Communication
	Transmission: Point-to-point	Transmission: Broadcast	Transmission: Broadcast	Transmission: Point-to-point	
Asynchronous	No	No	Yes	No	Unbounded
Synchronous	No	No	Yes	No	Bounded
	Yes	Yes	Yes	Yes	
	No	No	Yes	Yes	Unbounded

Table credit: [Turek-Shasha-1992]

Can's and Cannot's

Processors	Message Order: Unordered		Message Order: Ordered		Communication
	Transmission: Point-to-point	Transmission: Broadcast	Transmission: Point-to-point		
Asynchronous	No	No	Yes	No	Unbounded
	No	No	Yes	No	Bounded
Synchronous	Yes	Yes	Yes	Yes	
	No	No	Yes	Yes	Unbounded

asynchronous system (realistic): mostly no.

synchronous system (not realistic): yes.

Table credit: [Turek-Shasha-1992]

Next Classes

Today: A parenthesis on local transactions

Then: **Atomic commitment protocols**

Then: **Consensus protocols**

Finally: Case studies from industry

Key Papers

- [FLP-1985] Michael Fischer, Nancy Lynch, and Michael Paterson. *Impossibility of Distributed Consensus with One Faulty Process.* In Journal of the ACM, 1985.
- [Turek-Shasha-1992] John Turek, Dennis Shasha. *The many faces of consensus in distributed systems.* In *IEEE Computer*, 1992.