Group Agreement

- Servers need to agree on a common value/state
  - The state is distributed across servers
  - The value(s) may be proposed by servers
- Examples?
- Hard problem, depends on:
  - Reliability of communication channel
  - Behavior of faulty servers

Group Agreement: Feasibility

- Factors:
  - Process behavior: Synchronous or asynchronous
  - Communication delay: bounded or unbounded
  - Message ordering: Ordered or unordered
  - Message transmission: Unicast or multicast
- Can achieve agreement if:
  - Synchronous: Bounded delay or ordered messages
  - Asynchronous: Not possible in general
Two-Army Problem
- Simple scenario:
  - Two armies (perfect servers)
  - Communicate through a messenger that can be caught (unreliable communication channel)
  - Both need to agree on a common value
- Question: Can they agree?
- Example: TCP connection termination

Byzantine Generals Problem
- N generals, M traitors
- Reliable communication channel
- Problem: Traitors can lie, others don't know who the traitors are
- Question: Can trusted generals agree on their army sizes?

Requirement for Byzantine Fault Tolerance
- Must have: $N \geq 3M+1$ for agreement
- Why?

Lamport’s Agreement Algorithm
- Round-based. Each round:
  - Step 1: Each general sends everyone else its army size
    - Loyal generals tell the truth, traitors can lie
  - Step 2: Each general collects received info as a vector
  - Step 3: Each general sends its vector to others
    - Loyal generals send what they have
    - Traitors can change the vectors
  - Step 4: Each general determines vector elements by voting among received vectors
- Multiple rounds depending on $N$ and $M$
Lamport’s Agreement Algorithm

N=4, M=1

1 2
[1,2,3,5] [1,2,3,6]
1 2
[1,2,3,5] [e,f,g,h]

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>[1,2,3,6]</td>
<td>[1,2,3,5]</td>
<td>[1,2,3,5]</td>
</tr>
<tr>
<td>[1,2,3,7]</td>
<td>[1,2,3,7]</td>
<td>[1,2,3,6]</td>
</tr>
<tr>
<td>[a,b,c,d]</td>
<td>[e,f,g,h]</td>
<td>[l,j,k,l]</td>
</tr>
</tbody>
</table>

Paxos

- A general agreement algorithm
  - Nodes can propose different (and multiple) values
- Goal:
  - All nodes must agree on the same value
  - An agreed value must have been proposed
- Assumptions:
  - Unreliable network with unbounded delays
  - Nodes can crash, but have stable storage (so they can resume from the pre-crash state)
- Used in Google’s Chubby, Apache Zookeeper coordination services

Paxos: Entities

- 3 types of nodes:
  - Proposers: Propose values
  - Acceptors: Accept (or reject) values
  - Learners: Learn the eventually accepted value
- Different processes can have different roles, or the roles can be overlapping
**Paxos Algorithm: Overview**

- Each proposer can propose a value $v$ with a sequence number $s$
- All proposals have unique sequence numbers
- Goal: Pick a value $v$ from all proposals
- Insights:
  - Only need a quorum of acceptors to agree on a value
  - Once a value is picked, then older proposals can be rejected
  - Once a value is picked by a quorum, then subsequent proposers must agree to this value

**Paxos Algorithm**

- Phase 1 (Prepare Phase): Proposer sends a proposal $<n,v>$ to a quorum of acceptors
  - An acceptor can reply with a:
    - Promise (not to accept a lower number proposal, sends the value of previous highest number accepted proposal)
    - Reject (Won't accept this proposal)
- Phase 2 (Accept Phase): Proposer getting Promises from quorum sends Accept
  - With a value (highest-numbered proposal's value seen so far, or $v$ otherwise)
- Phase 3 (Commit Phase): Quorum of acceptors send ACK back. Learners are notified

**Paxos: Benefits**

- Works if machines crash and resume:
  - If proposer sends low sequence number, it can be rejected
  - If an acceptor comes back with an old Promise, this can be superseded by a newer proposal
  - The final value can be propagated by any learner
  - Robust to network partitions
    - If one partition has a majority of acceptors