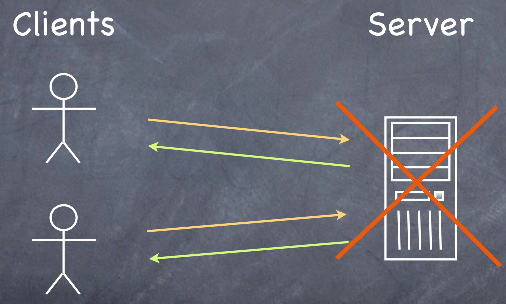
State-Machine Replication

The Problem



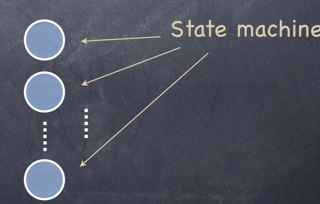
Solution: replicate server!

The Solution

1. Make server deterministic (state machine)

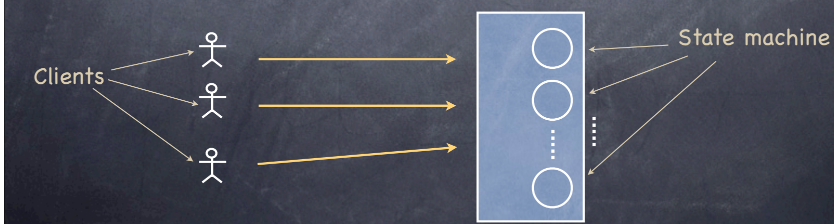
The Solution

1. Make server deterministic (state machine)
2. Replicate server



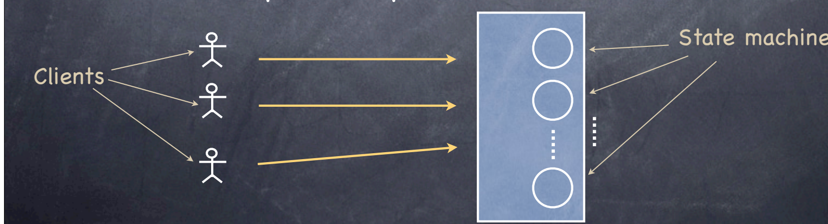
The Solution

1. Make server deterministic (state machine)
2. Replicate server
3. Ensure correct replicas step through the same sequence of state transitions



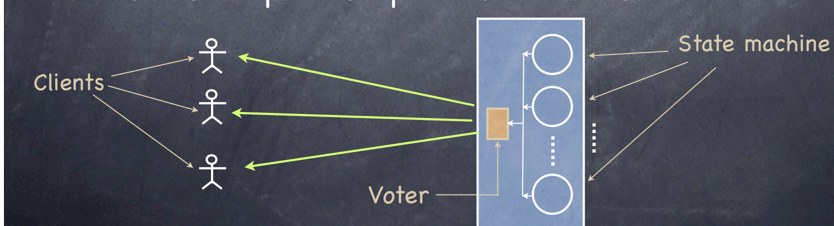
The Solution

1. Make server deterministic (state machine)
2. Replicate server
3. Ensure correct replicas step through the same sequence of state transitions
4. Vote on replica outputs for fault-tolerance

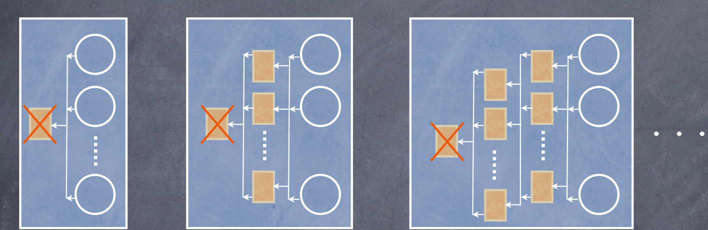


The Solution

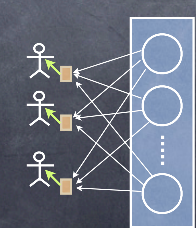
1. Make server deterministic (state machine)
2. Replicate server
3. Ensure correct replicas step through the same sequence of state transitions
4. Vote on replica outputs for fault-tolerance



A conundrum



A: voter and client share fate!



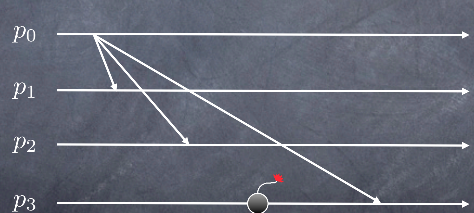
Consensus and Reliable Broadcast

Broadcast

- If a process sends a message m, then every process eventually delivers m

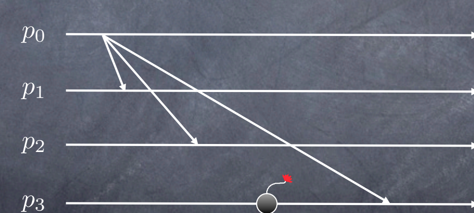
Broadcast

- If a process sends a message m, then every process eventually delivers m



Broadcast

- If a process sends a message m, then every process eventually delivers m



- How can we adapt the spec for an environment where processes can fail?

Reliable Broadcast

Validity If the sender is correct and broadcasts a message m, then all correct processes eventually deliver m

Agreement If a correct process delivers a message m, then all correct processes eventually deliver m

Integrity Every correct process delivers at most one message, and if it delivers m, then some process must have broadcast m

Terminating Reliable Broadcast

Termination Every correct process eventually delivers some message

Validity If the sender is correct and broadcasts a message m, then all correct processes eventually deliver m

Agreement If a correct process delivers a message m, then all correct processes eventually deliver m

Integrity Every correct process delivers at most one message, and if it delivers m, then some process must have broadcast m

Terminating Reliable Broadcast

Termination Every correct process eventually delivers some message

Validity If the sender is correct and broadcasts a message m, then all correct processes eventually deliver m

Agreement If a correct process delivers a message m, then all correct processes eventually deliver m

Integrity Every correct process delivers at most one message, and if it delivers $m \neq$ SF, then some process must have broadcast m

Consensus

Termination Every correct process eventually decides some value

Validity If all processes that propose a value propose v, then all correct processes eventually decide v

Agreement If a correct process decides v, then all correct processes eventually decide v

Integrity Every correct process decides at most one value, and if it decides $v \neq$ NU, then some process must have proposed v

Properties of send(m) and receive(m)

Benign failures:

Validity If p sends m to q, and p, q, and the link between them are correct, then q eventually receives m

Uniform* Integrity For any message m, q receives m at most once from p, and only if p sent m to q

* A property is uniform if it applies to both correct and faulty processes

Properties of send(m) and receive(m)

Arbitrary failures:

Integrity For any message m, if p and q are correct then q receives m at most once from p, and only if p sent m to q

Questions, Questions...

- Are these problems solvable at all?
- Can they be solved independent of the failure model?
- Does solvability depend on the ratio between faulty and correct processes?
- Does solvability depend on assumptions about the reliability of the network?
- Are the problems solvable in both synchronous and asynchronous systems?
- If a solution exists, how expensive is it?

Plan

- Synchronous Systems
  - Consensus for synchronous systems with crash failures
  - Lower bound on the number of rounds
  - Early stopping protocols for Reliable Broadcast
  - Reliable Broadcast for arbitrary failures with message authentication
  - Lower bound on the ratio of faulty processes for Consensus with arbitrary failures
  - Reliable Broadcast for arbitrary failures
- Asynchronous Systems
  - Impossibility of Consensus for crash failures

Model

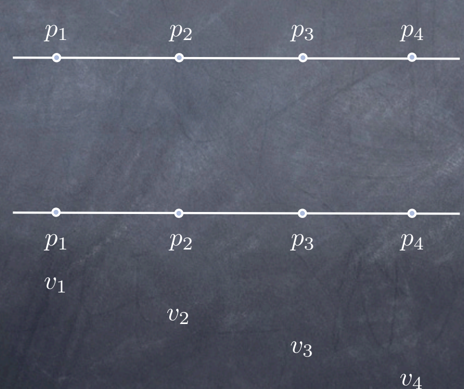
- Synchronous Message Passing
  - Execution is a sequence of rounds
  - In each round every process takes a step
    - sends messages to neighbors
    - receives messages sent in that round
    - changes its state
- Network is fully connected (an n-clique)
- No communication failures

A simple Consensus algorithm

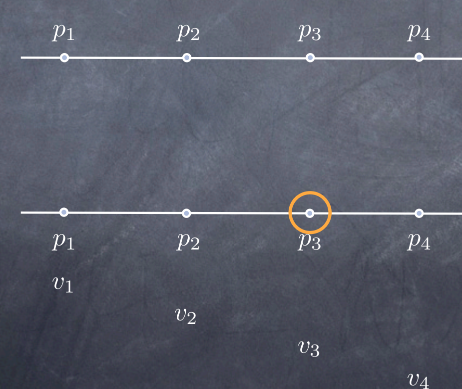
Process p_i:

```
Initially V={v_i}
To execute propose(v_i)
    send {v_i} to all
decide(x) occurs as follows:
    for all j, 0 ≤ j ≤ n-1, j ≠ i do
      receive S_j from p_j
      V:= V U S_j
    decide min(V)
```

An execution

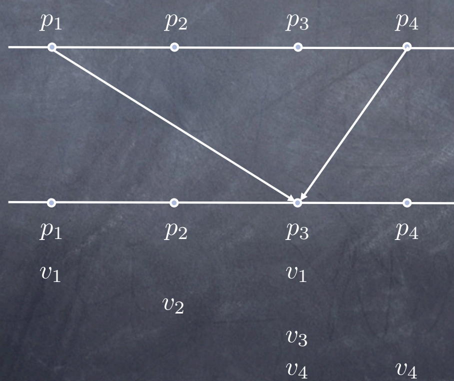


An execution



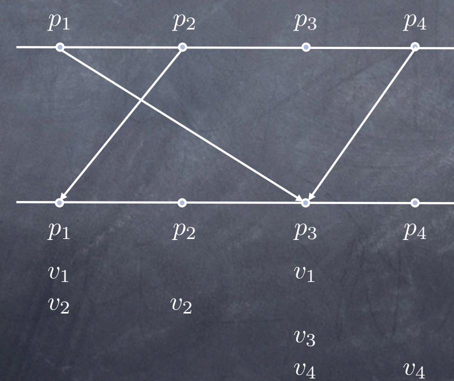
An execution

Suppose $v_1 = v_3 = v_4$ at the end of round 1
Can p_3 decide?



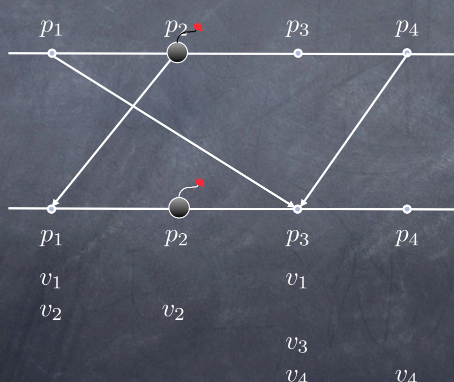
An execution

Suppose $v_1 = v_3 = v_4$ at the end of round 1
Can p_3 decide?



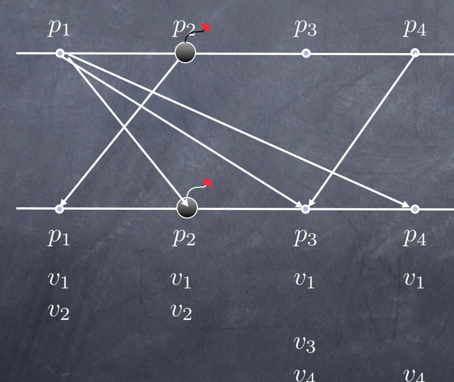
An execution

Suppose $v_1 = v_3 = v_4$ at the end of round 1
Can p_3 decide?



An execution

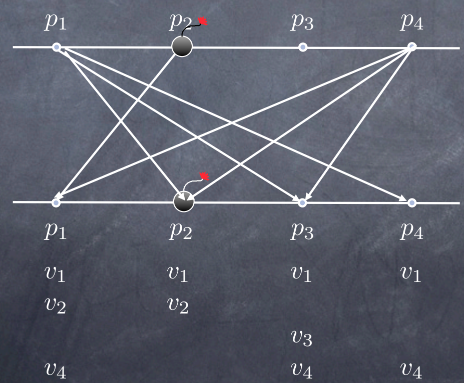
Suppose $v_1 = v_3 = v_4$ at the end of round 1
Can p_3 decide?



An execution

Suppose $v_1 = v_3 = v_4$ at the end of round 1

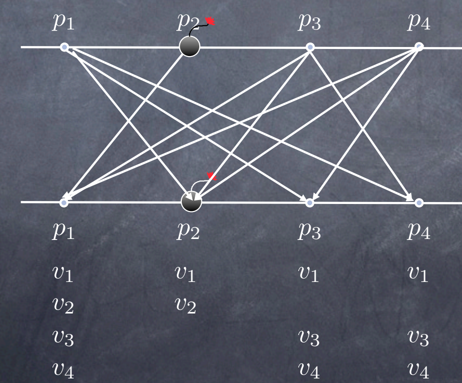
Can p_3 decide?



An execution

Suppose $v_1 = v_3 = v_4$ at the end of round 1

Can p_3 decide?



Echoing values

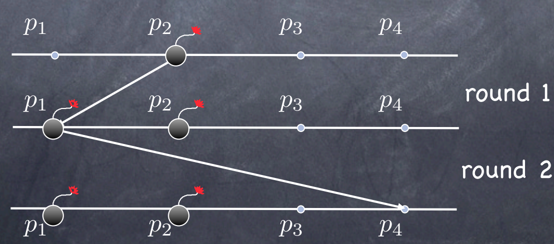
- A process that receives a proposal in round 1, relays it to others during round 2.

Echoing values

- A process that receives a proposal in round 1, relays it to others during round 2.
- Suppose p_3 hasn't heard from p_2 at the end of round 2. Can p_3 decide?

Echoing values

- A process that receives a proposal in round 1, relays it to others during round 2.
- Suppose p_3 hasn't heard from p_2 at the end of round 2. Can p_3 decide?



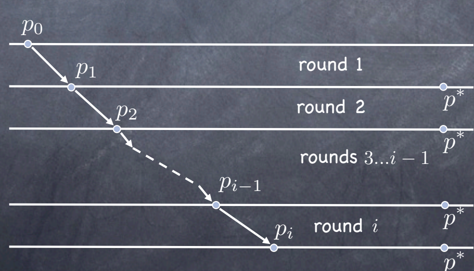
What is going on

- A correct process p^* has not received all proposals by the end of round i. Can p^* decide?
- Another process may have received the missing proposal at the end of round i and be ready to relay it in round $i+1$

Dangerous Chains

Dangerous chain

The last process in the chain is correct, all others are faulty



Living dangerously

How many rounds can a dangerous chain span?

- f faulty processes
- at most $f+1$ nodes in the chain
- spans at most f rounds

It is safe to decide by the end of round $f+1$!