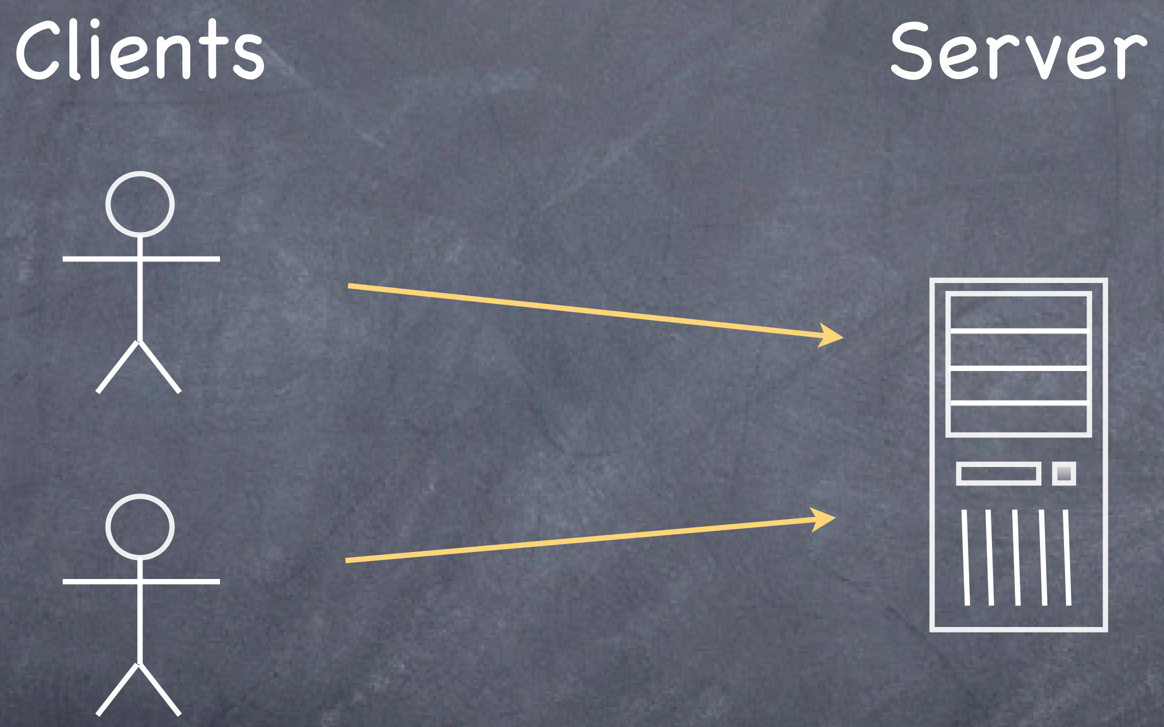
State-Machine Replication

The Problem

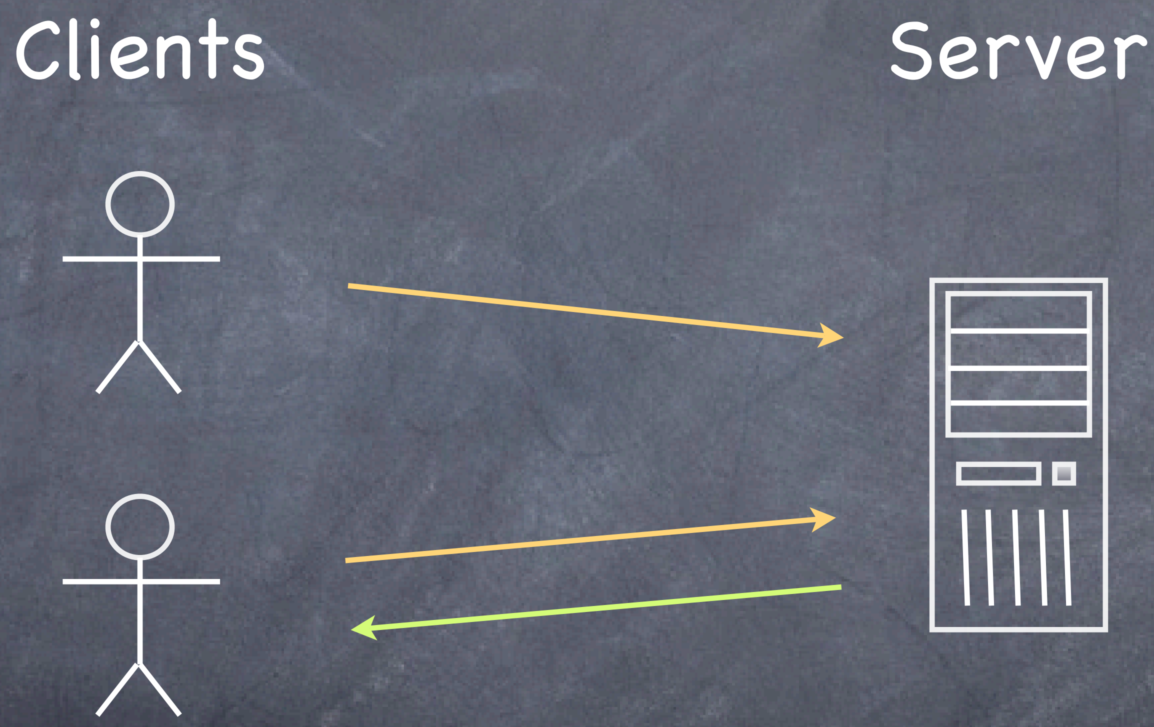
Clients

Server

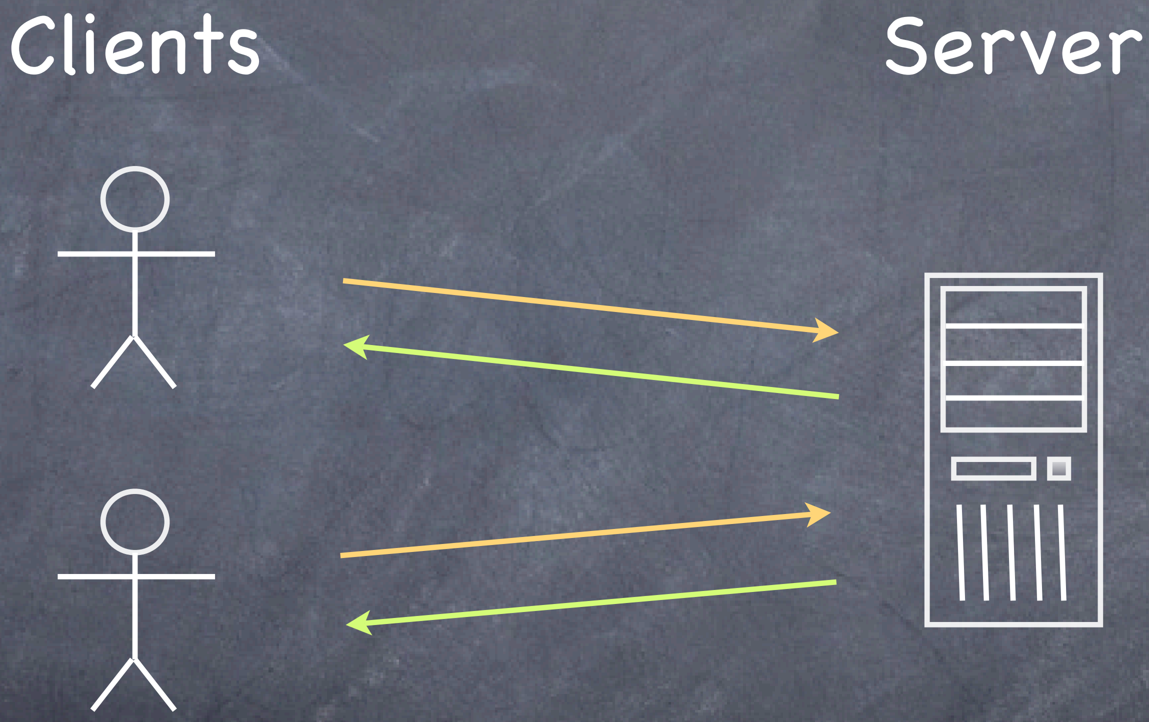
The Problem



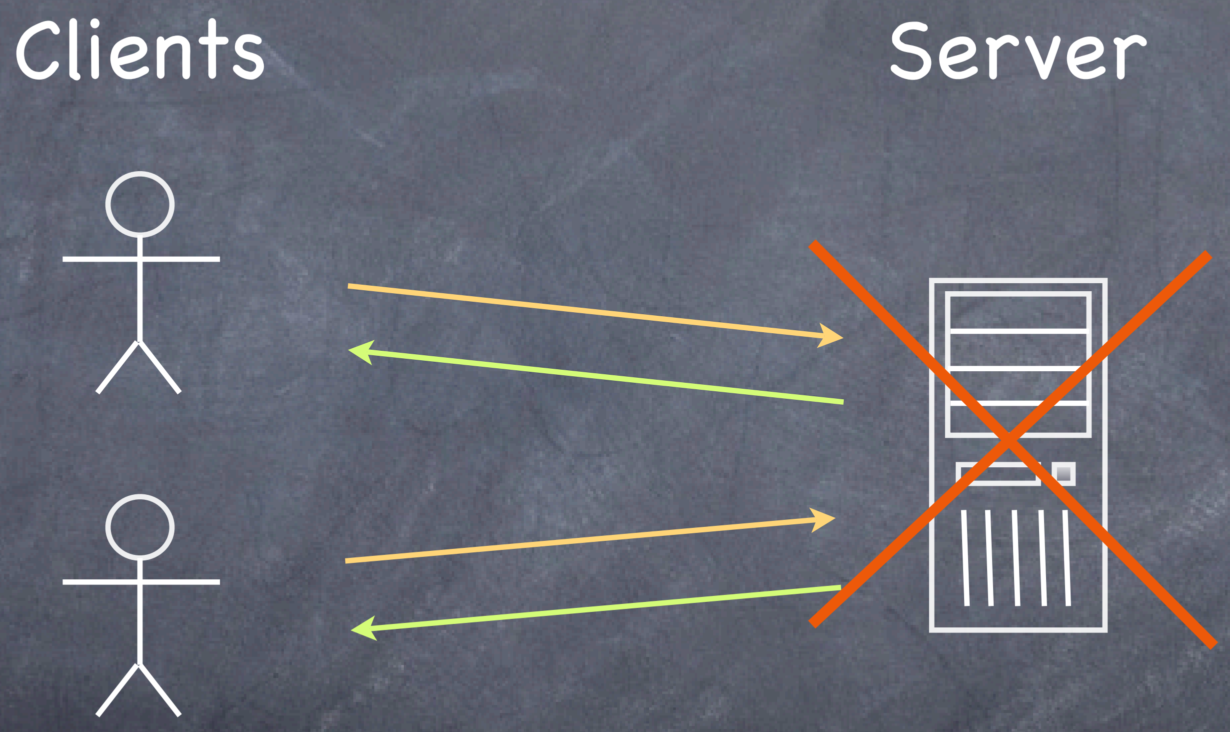
The Problem



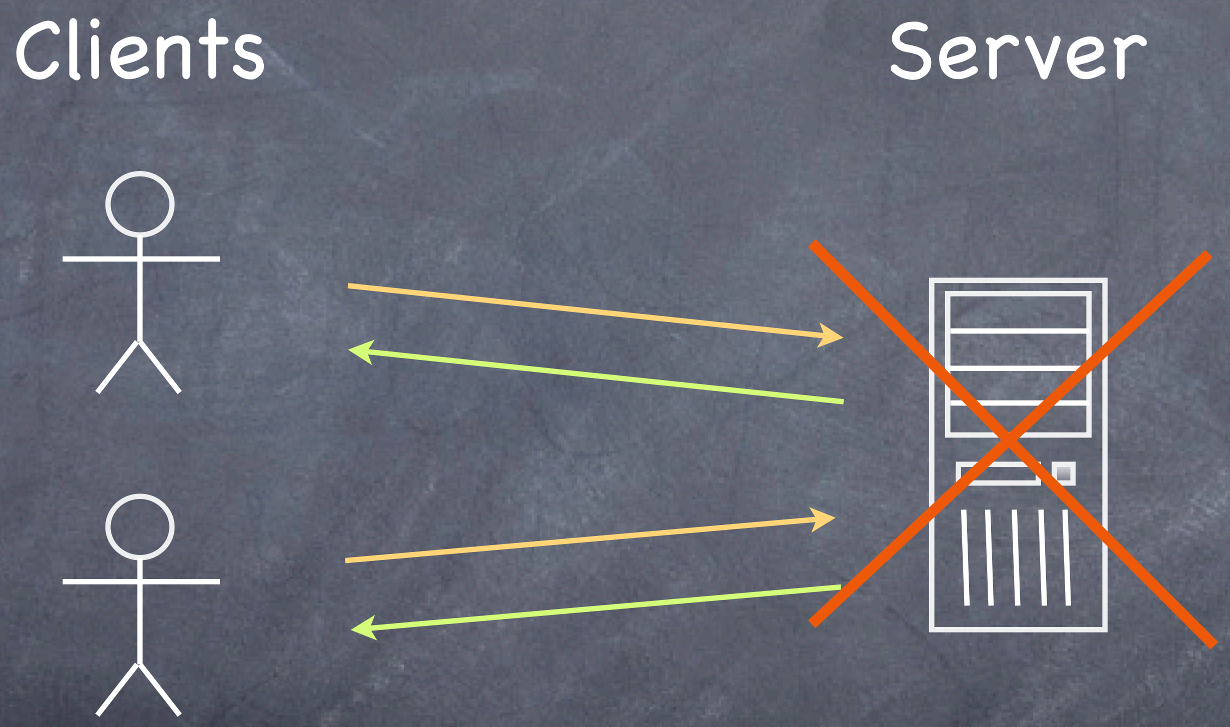
The Problem



The Problem



The Problem



Solution: replicate server!

The Solution

The Solution

1. Make server deterministic (state machine)

The Solution

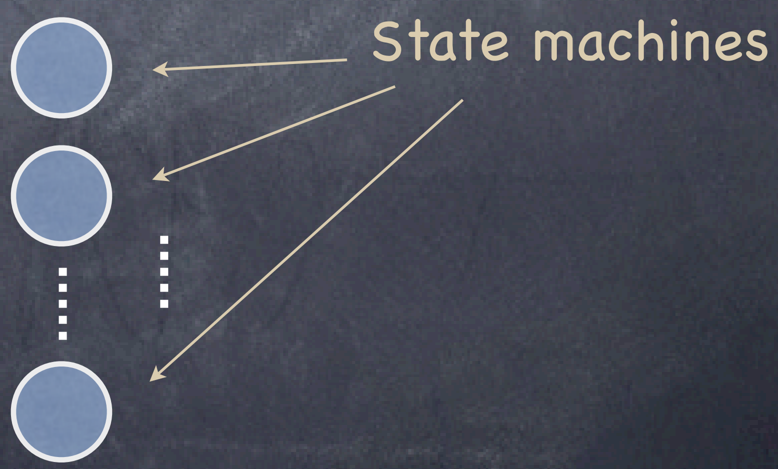
1. Make server deterministic (state machine)

The Solution

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

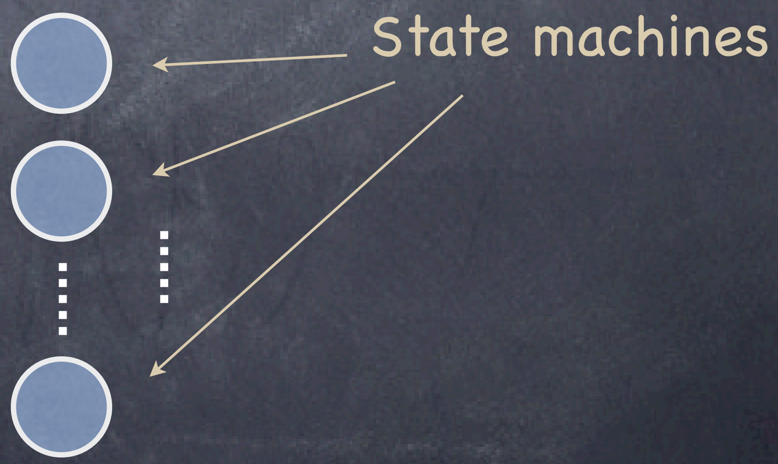
The Solution

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



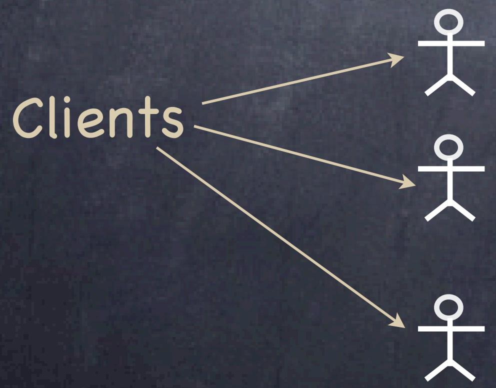
The Solution

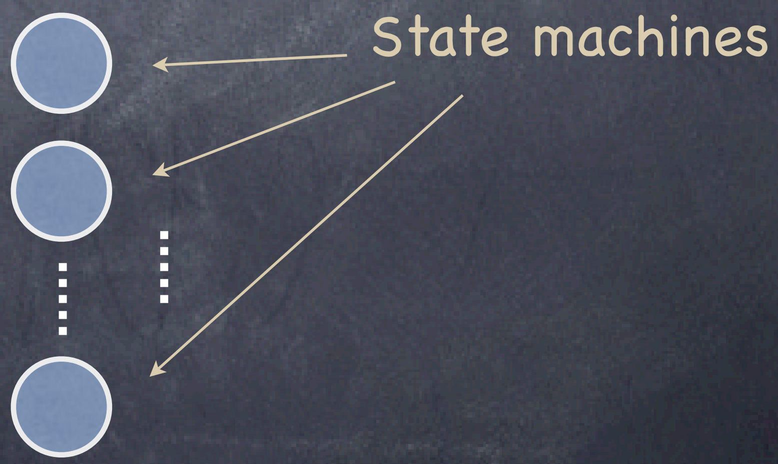
1. Make server deterministic (state machine)
2. Replicate server
3. Ensure correct replicas step through the same sequence of state transitions



The Solution

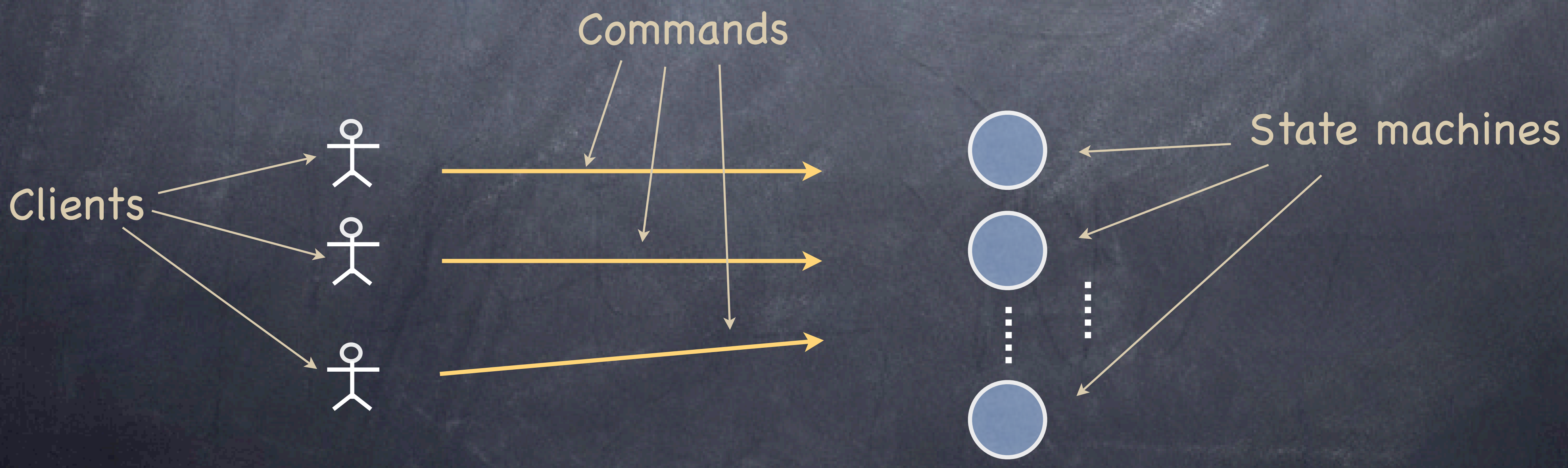
1. Make server deterministic (state machine)
2. Replicate server
3. Ensure correct replicas step through the same sequence of state transitions





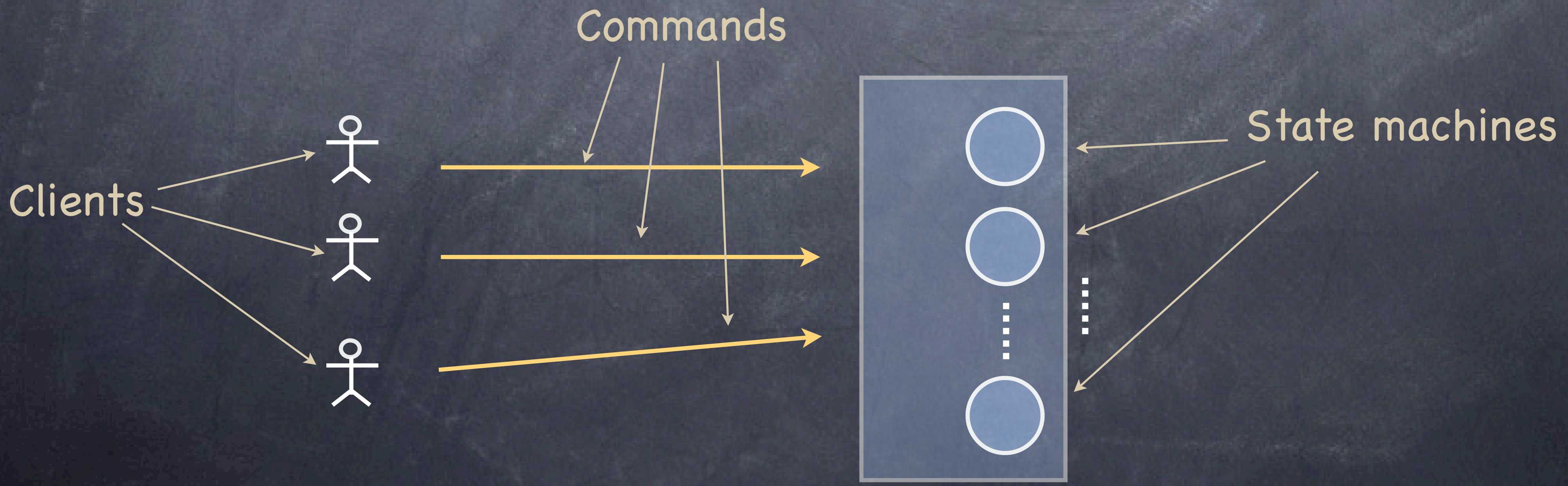
The Solution

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



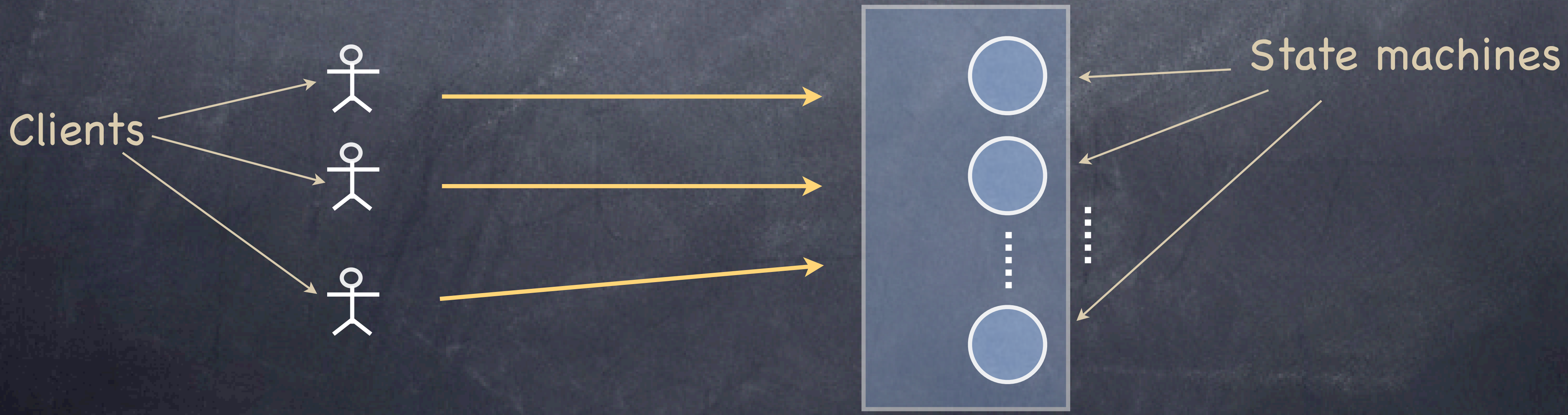
The Solution

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



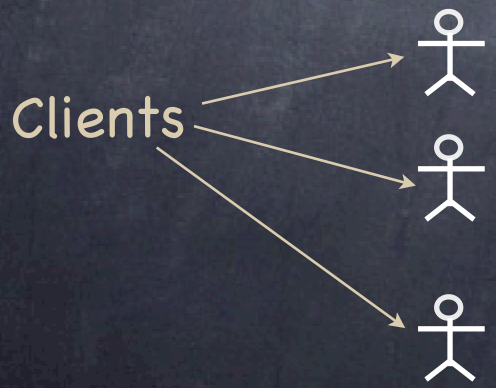
The Solution

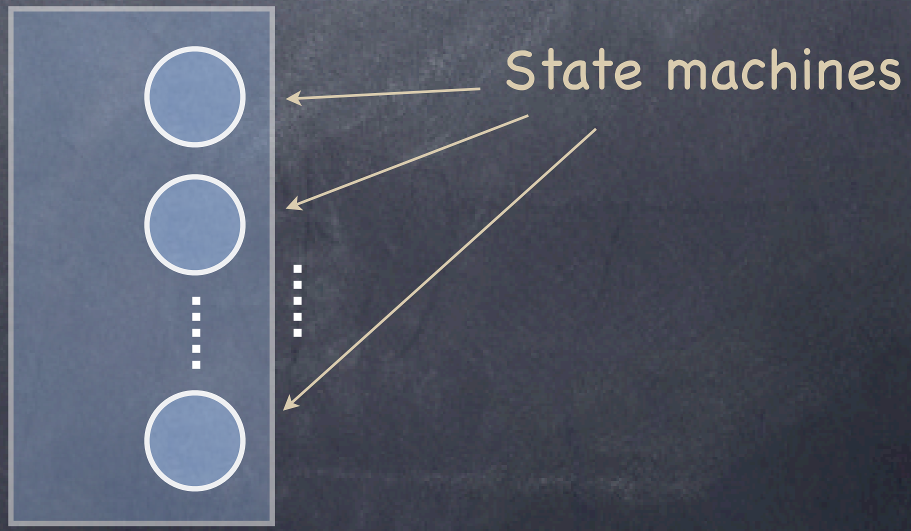
1. Make server deterministic (state machine)
2. Replicate server
3. Ensure correct replicas step through the same sequence of state transitions
4. Vote on replica outputs for fault-tolerance



The Solution

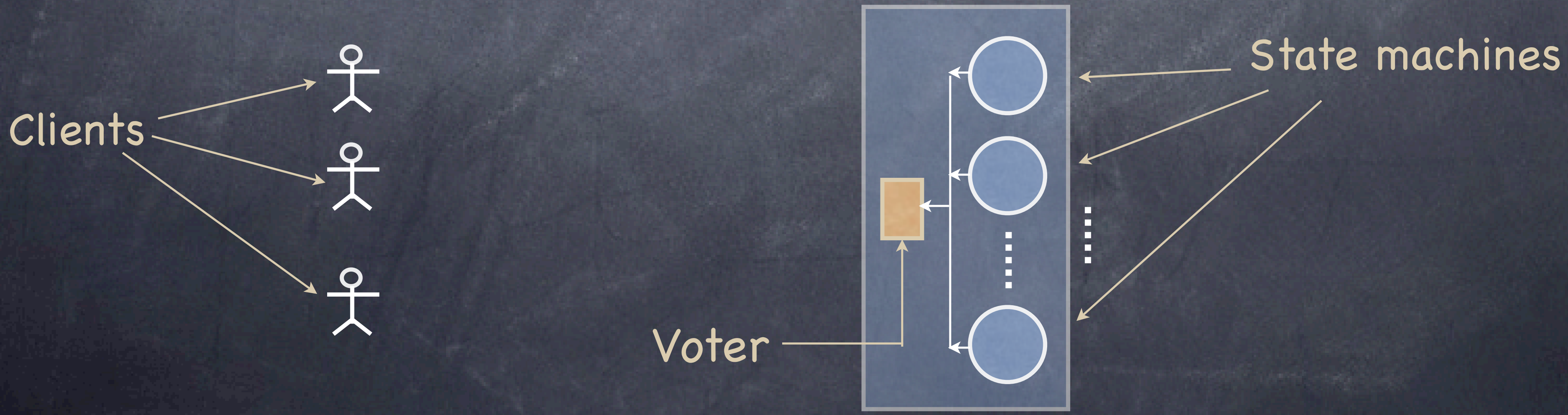
1. Make server deterministic (state machine)
2. Replicate server
3. Ensure correct replicas step through the same sequence of state transitions
4. Vote on replica outputs for fault-tolerance





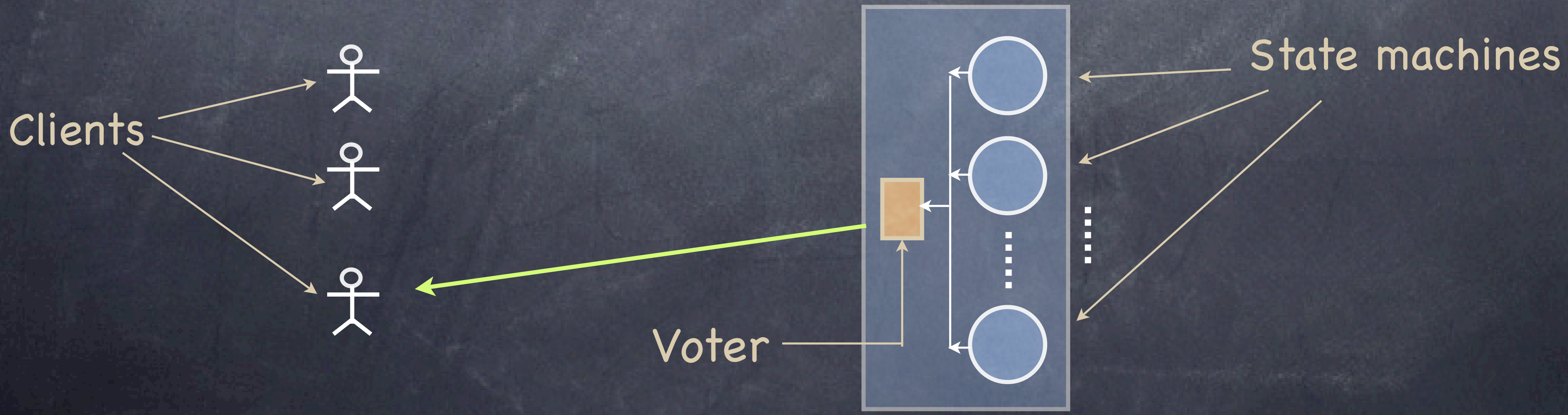
The Solution

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



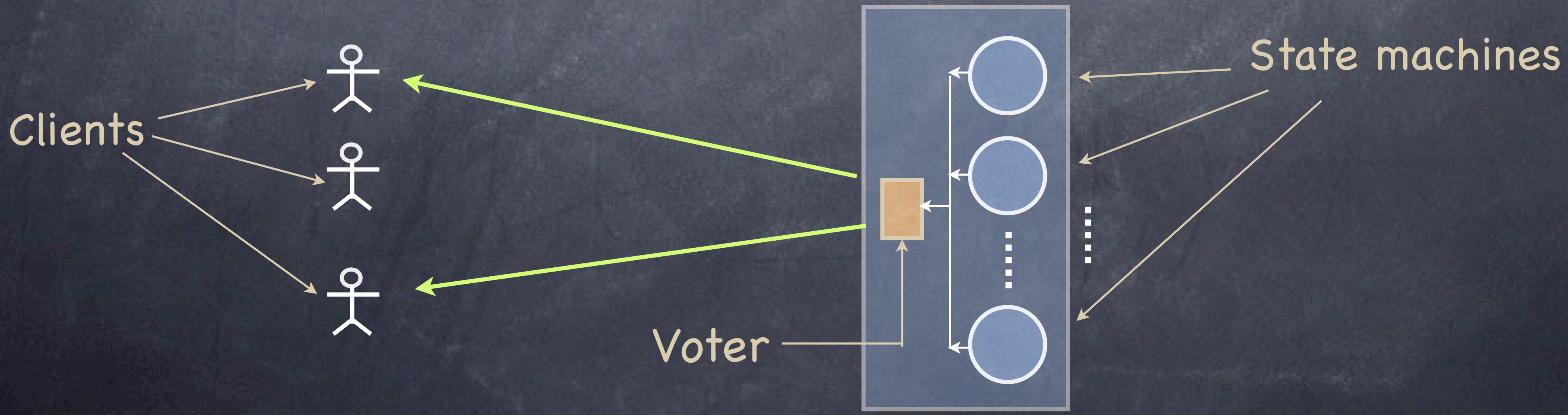
The Solution

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



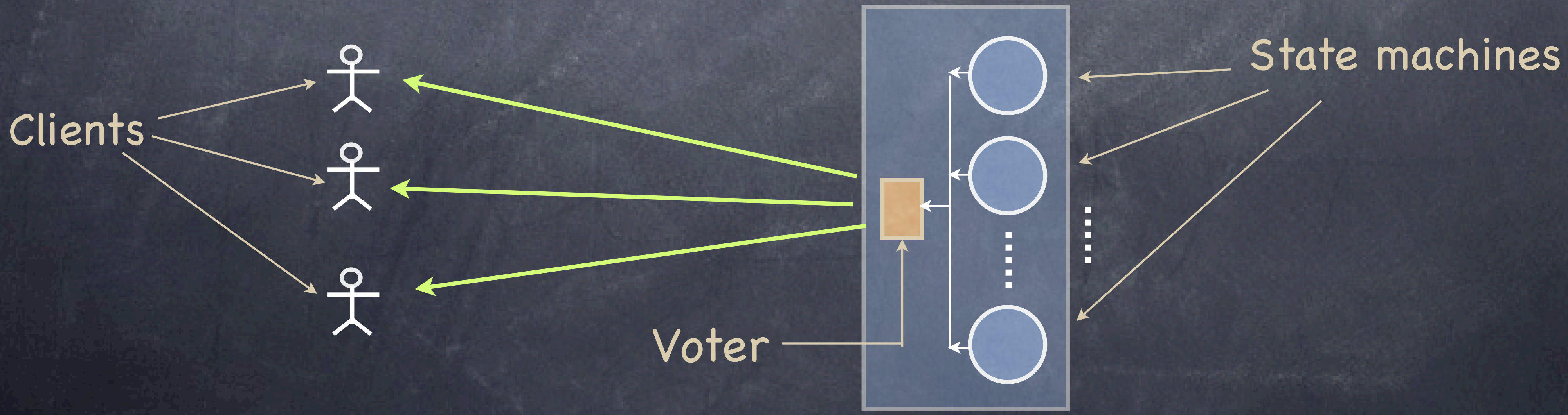
The Solution

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

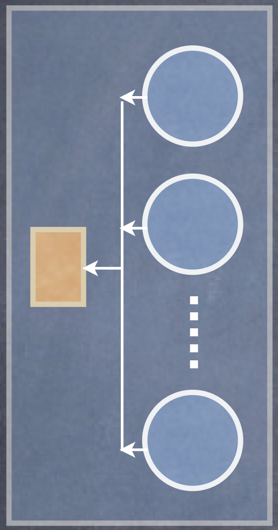


The Solution

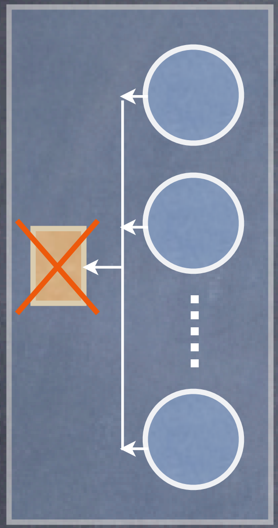
1. Make server deterministic (state machine)
2. Replicate server
3. Ensure correct replicas step through the same sequence of state transitions
4. Vote on replica outputs for fault-tolerance



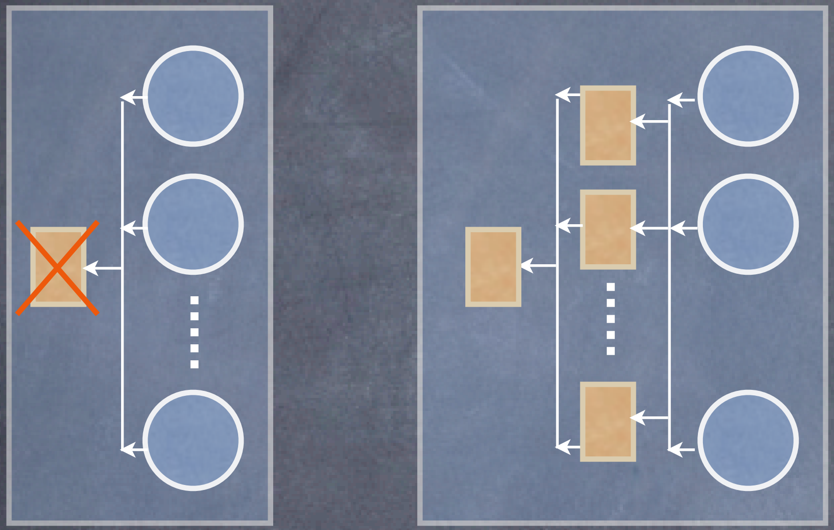
A conundrum



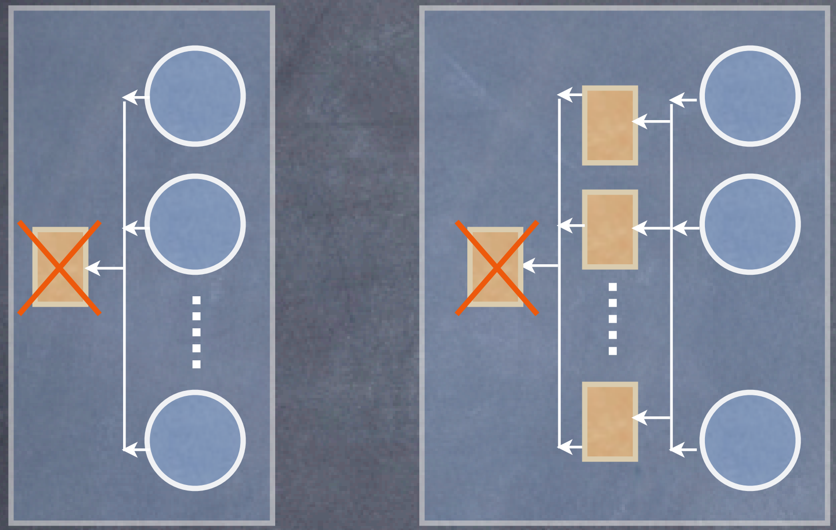
A conundrum



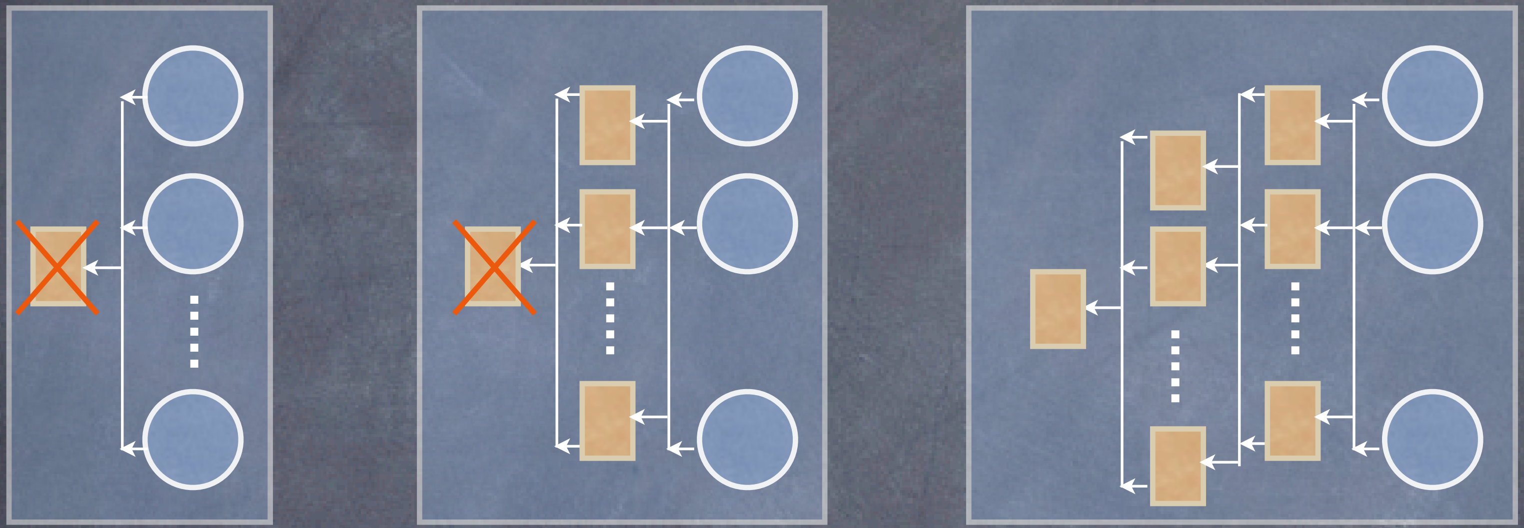
A conundrum



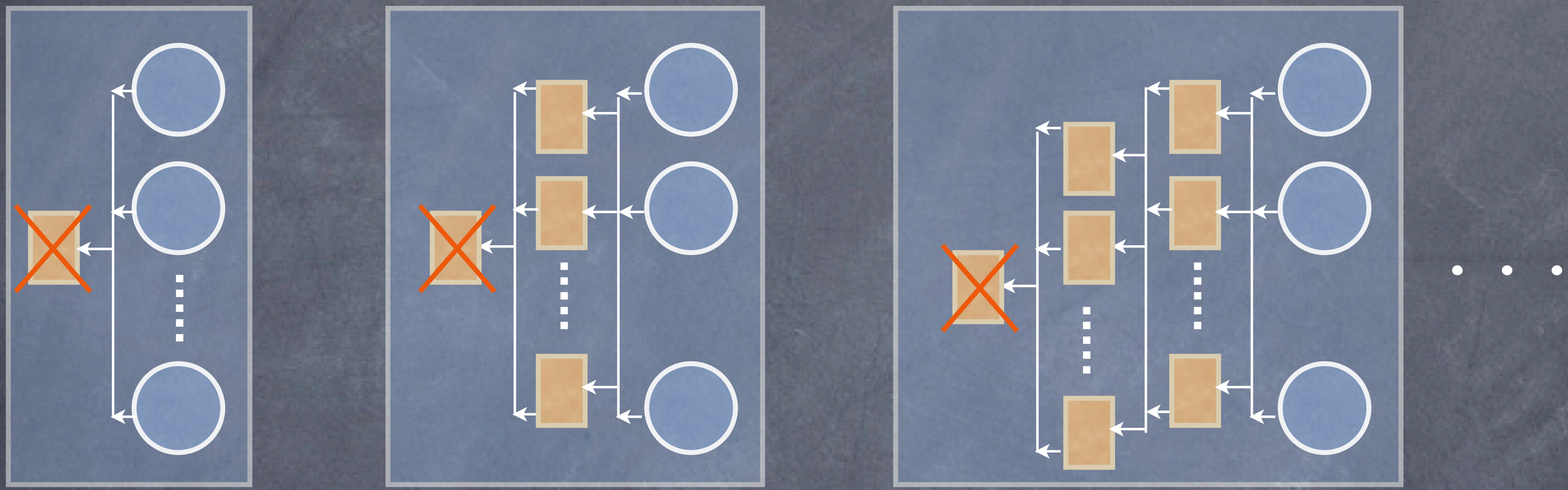
A conundrum



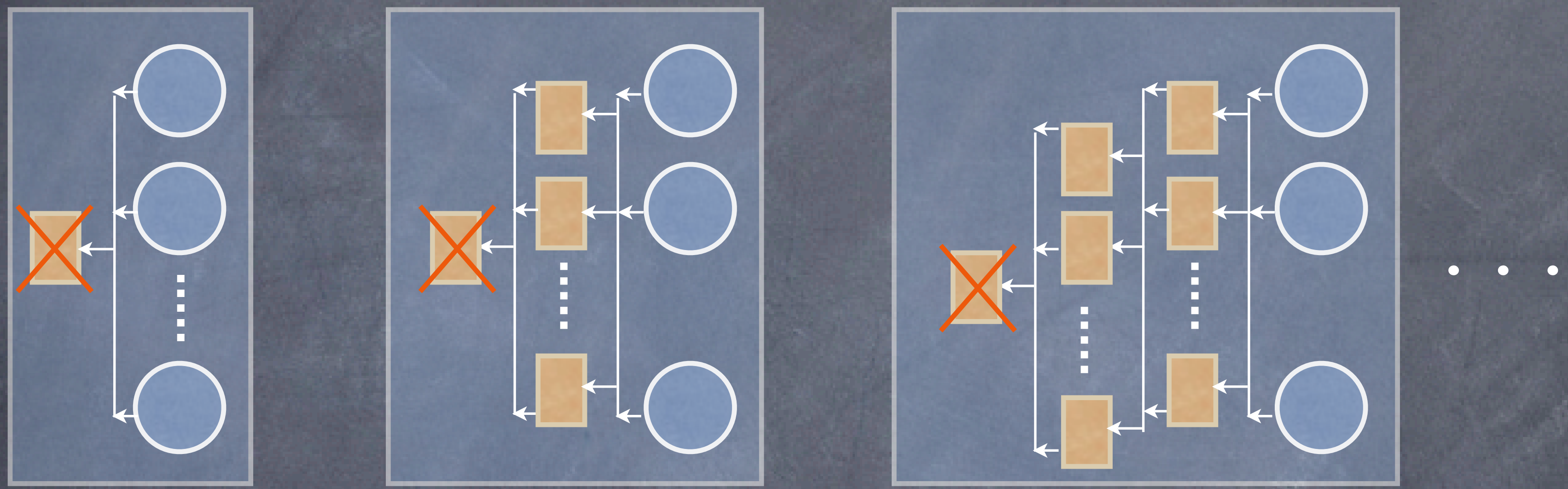
A conundrum



A conundrum

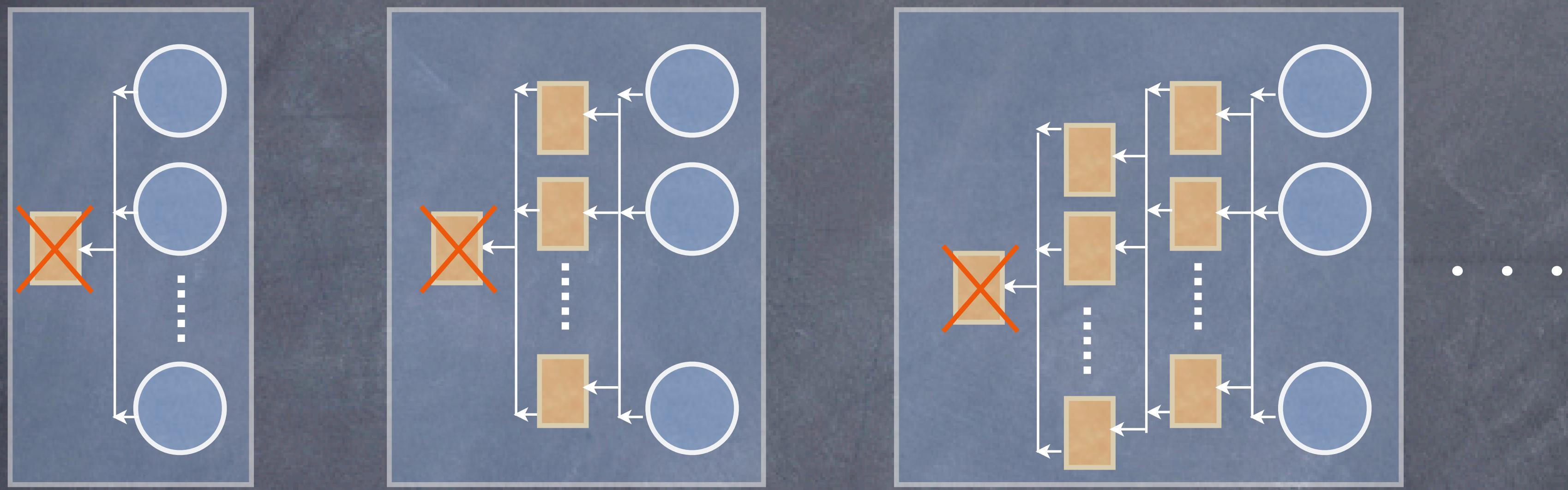


A conundrum

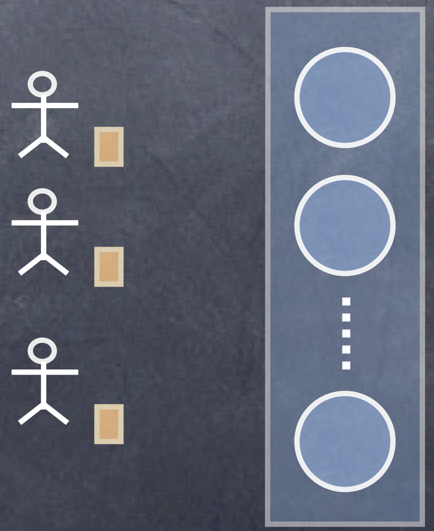


A: voter
and client
share fate!

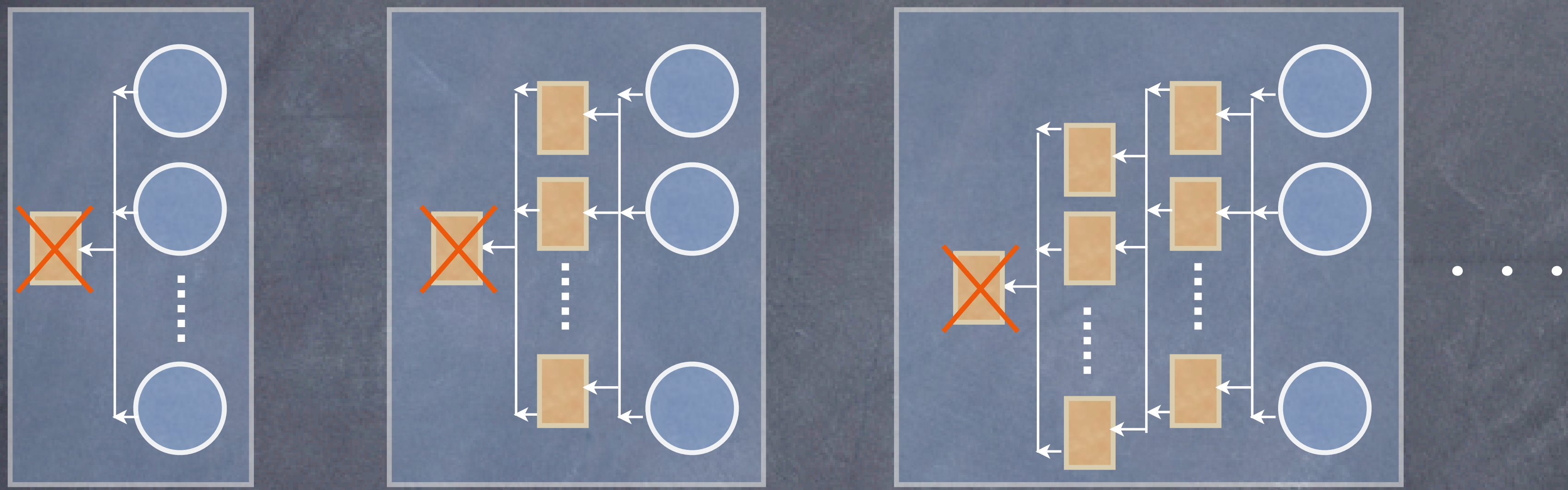
A conundrum



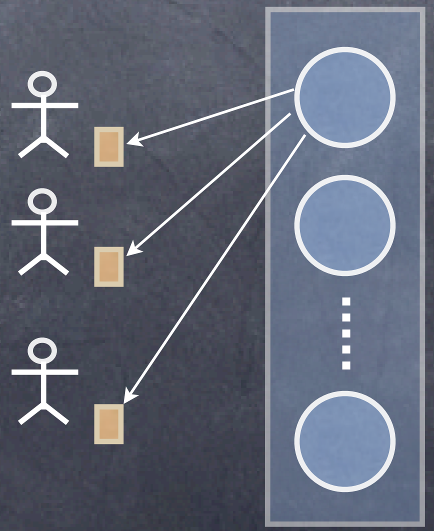
A: voter
and client
share fate!



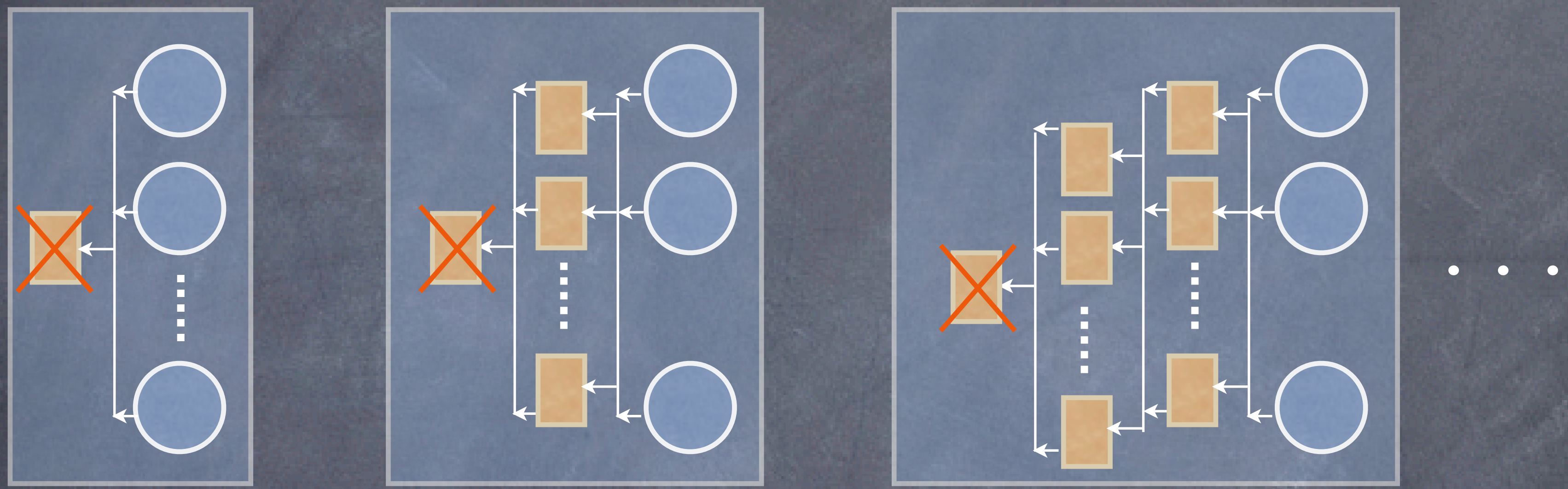
A conundrum



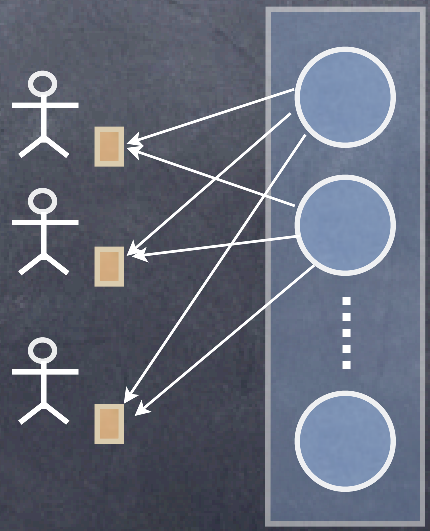
A: voter
and client
share fate!



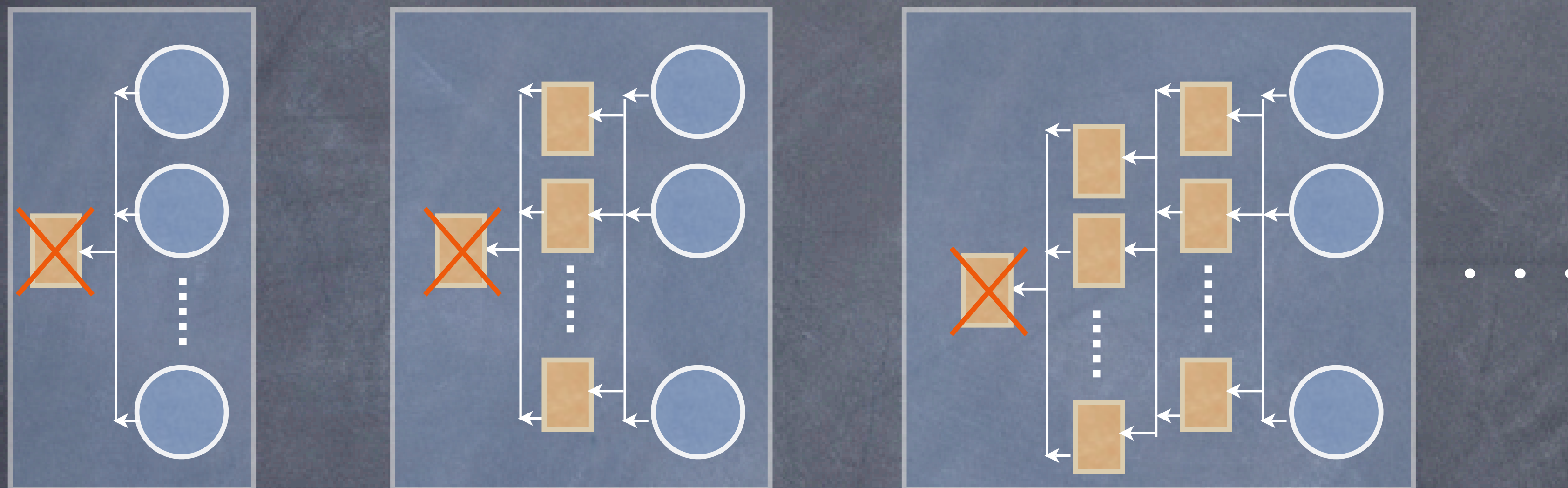
A conundrum



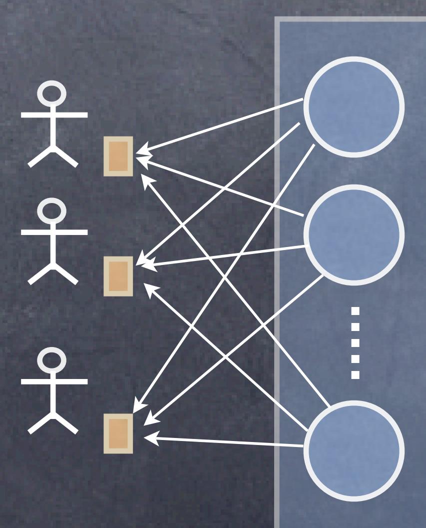
A: voter
and client
share fate!



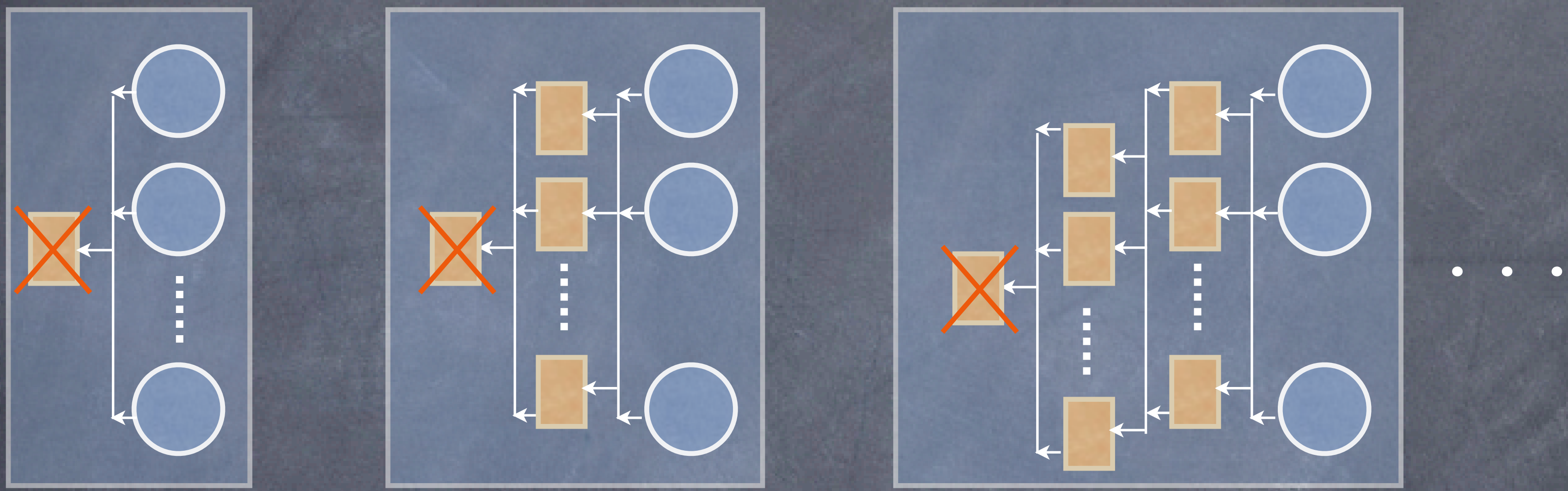
A conundrum



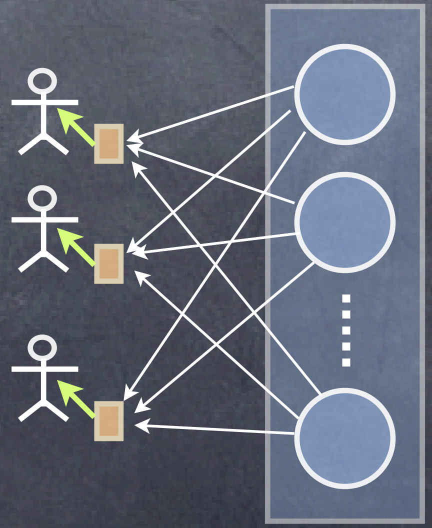
A: voter
and client
share fate!



A conundrum



A: voter
and client
share fate!



State Machines

- Set of state variables + Sequence of commands
- A command
  - Reads its read set values (opt. environment)
  - Writes to its write set values (opt. environment)
- A deterministic command
  - Produces deterministic wsvs and outputs on given rsv
- A deterministic state machine
  - Reads a fixed sequence of deterministic commands

Semantic Characterization of a State Machine

Outputs of a state machine are completely determined by the sequence of commands it processes, independent of time and any other activity in a system

Replica Coordination

All non-faulty state machines receive all commands in the same order

- Agreement: Every non-faulty state machine receives every command
- Order: Every non-faulty state machine processes the commands it receives in the same order

Where should RC be implemented?

- In hardware
  - sensitive to architecture changes
- At the OS level
  - state transitions hard to track and coordinate
- At the application level
  - requires sophisticated application programmers

Hypervisor-based Fault-tolerance

- Implement RC at a virtual machine running on the same instruction-set as underlying hardware
- Undetectable by higher layers of software
- One of the great come-backs in systems research!
  - CP-67 for IBM 369 [1970]
  - Xen [SOSP 2003], VMware

The Hypervisor as a State Machine

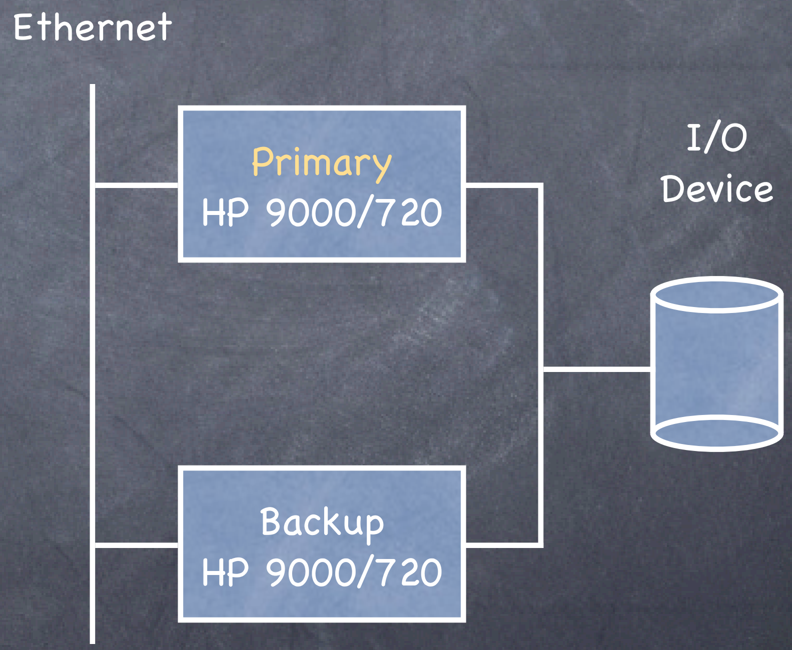
- Two types of commands
  - virtual-machine instructions
  - virtual-machine interrupts (with DMA input)
- State transition must be deterministic
  - ...but some VM instructions are not (e.g. time-of-day)
  - interrupts must be delivered at the same point in command sequence

The Architecture

- Good-ol' Primary-Backup
- Primary makes all non-deterministic choices

I/O Accessibility Assumption

Primary and backup have access to same I/O operations



Ensuring identical command sequences

- Ordinary (deterministic) instructions
- Environment (nondeterministic) instructions

Ensuring identical command sequences

- Ordinary (deterministic) instructions
- Environment (nondeterministic) instructions
- Environment Instruction Assumption

  Hypervisor captures all environmental instructions, simulates them, and ensures they have the same effect at all state machines

Ensuring identical command sequences

- Ordinary (deterministic) instructions
- Environment (nondeterministic) instructions
- Environment Instruction Assumption
- VM interrupts must be delivered at same point in instruction sequence at all replicas

Ensuring identical command sequences

- Ordinary (deterministic) instructions
- Environment (nondeterministic) instructions
- Environment Instruction Assumption
- VM interrupts must be delivered at same point in instruction sequence at all replicas
- Instruction Stream Interrupt Assumption
  - Hypervisor can be invoked at specific point in the instruction stream

Ensuring identical command sequences

- Ordinary (deterministic) instructions
- Environment (nondeterministic) instructions
- Environment Instruction Assumption
- VM interrupts must be delivered at same point in instruction sequence at all replicas
- Instruction Stream Interrupt Assumption
  - implemented via recovery register
  - interrupts at backup are ignored

The failure-free protocol

P0: On processing environment instruction i at pc, HV of primary p :

sends $[e_p, pc, Val_i]$ to backup b
waits for ack

P1: If p'HV receives Int from its VM:

p buffers Int

P2: If epoch ends at p:

p sends to b all buffered Int in e_p
p waits for ack
p delivers all VM Int in e_p
$e_p := e_p + 1$
p starts e_p

P3: If b'HV processes environment instruction i at pc:

b waits for $[e_b, pc, Val_i]$ from p
returns Val_i

If b receives $[E, pc, Val]$ from p:

b sends ack to p
b buffers Val for delivery at E, pc

P4: If b'HV receives Int from its VM

b ignores Int

P5: If epoch ends at b:

b waits from p for interrupts for e_b
b sends ack to p
b delivers all VM Int buffered in e_b
$e_b := e_b + 1$
b starts e_b

If the primary fails…

P6: If b receives a failure notification instead of $[e_b, pc, Val_i]$, b executes i

If in P5 b receives failure notification instead of Int:

$e_b := e_b + 1$

b starts e_b <--- failover epoch

b is promoted primary for epoch $e_b + 1$

If p crashes before sending Int to b, Int is lost!

SMR and the environment

- On outputs, no exactly-once guarantee on outputs
- On primary failure, avoid input inconsistencies
  - time must increase monotonically
    - at epoch, primary informs backup of value of its clock
  - interrupts must be delivered as a fault-free processor would
    - but interrupts can be lost...
    - weaken constraints on I/O interrupts

On I/O device drivers

IO1: If an I/O instruction is executed and the I/O operation performed, the processor issuing the instruction delivers a completion interrupt, unless it fails. Either way, the I/O device is unaffected.

IO2: An I/O device may cause an uncertain interrupt (indicating the operation has been terminated) to be delivered by the processor issuing the I/O instruction. The instruction could have been in progress, completed, or not even started.

On an uncertain interrupt, the device driver reissues the corresponding I/O instruction–not all devices though are idempotent or testable

Backup promotion and uncertain interrupts

P7: The backup's VM generates an uncertain interrupt for each I/O operation that is outstanding right before the backup is promoted primary (at the end of the failover epoch)

The Hypervisor prototype

- Supports only one VM to eliminate issues of address translation
- Exploits unused privileged levels in HP's PA-RISC architecture (HV runs at level 1)
- To prevent software to detect HV, hacks one assembly HP-UX boot instruction

RC in the Hypervisor

- Nondeterministic ordinary instructions (Surprise!)

RC in the Hypervisor

- Nondeterministic ordinary instructions (Surprise!)
  - TLB replacement policy non-deterministic
  - TLB misses handled by software
  - Primary and backup may execute a different number of instructions!

HV takes over TLB replacement

RC in the Hypervisor

- Nondeterministic ordinary instructions (Surprise!)
  - TLB replacement policy non-deterministic
  - TLB misses handled by software
  - Primary and backup may execute a different number of instructions!

HV takes over TLB replacement

- Optimizations
  - p sends Int immediately
  - p blocks for acks only before output commit

The JVM as a State Machine

- Asynchronous commands
  - interrupts
- Non-deterministic commands
  - read time-of-day
- Non-deterministic read set values
  - multi-threaded access to shared data
- Output to the environment
  - simulate a single, fault-tolerant state machine

Non-deterministic Commands

Only invoked through Java Native Interface (JNI)

- direct access to OS and other libraries
- implement windowing, I/O, read HW clock...

Executes outside the JVM:
can't agree on inputs!

Non-deterministic Commands

Only invoked through Java Native Interface (JNI)

- direct access to OS and other libraries
- implement windowing, I/O, read HW clock...

Executes outside the JVM:
can't agree on inputs!

Non-deterministic Commands

Only invoked through Java Native Interface (JNI)

- direct access to OS and other libraries
- implement windowing, I/O, read HW clock...

Force agreement on the wsvs

Non-deterministic Commands

Only invoked through Java Native Interface (JNI)

- direct access to OS and other libraries
- implement windowing, I/O, read HW clock...

Force agreement on the wsvs

Not out of the woods:

- Non-deterministic output to the environment
- Non-deterministic method invocation