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

Primary-Backup

The Idea

- Clients communicate with a single replica (primary)
- Primary:
  - sequences clients' requests
  - updates as needed other replicas (backups) with sequence of client requests or state updates
  - waits for acks from all non-faulty clients
- Backups use timeouts to detect failure of primary
- On primary failure, a backup is elected as new primary

Primary-backup and non-determinism

- Non-deterministic commands executed only at the primary
- Backups receive either
  - state updates (non-determinism?)
  - command sequence (non-determinism?)

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

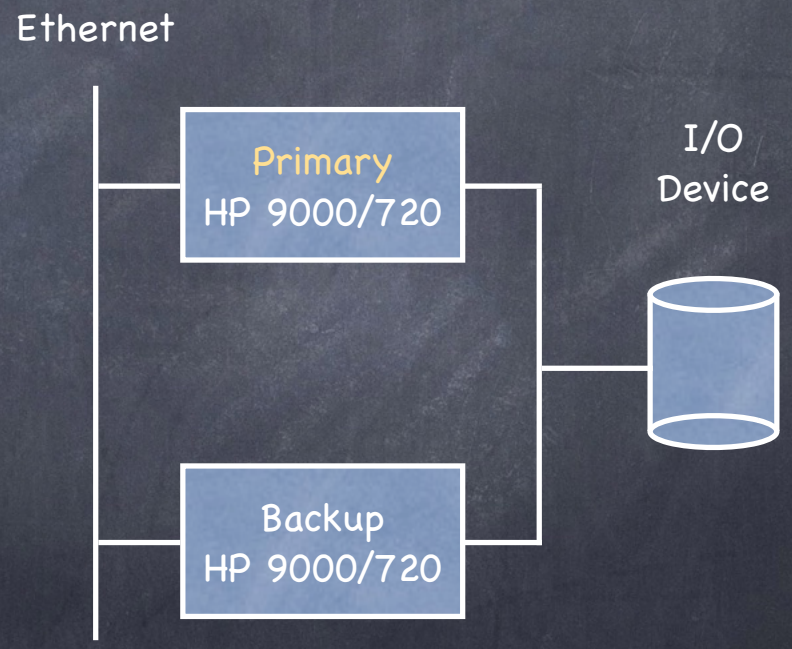
- Two types of commands
  - virtual-machine instructions
  - virtual-machine interrupts (with DMA input)
- State transition must be deterministic
  - ...but some VM instructions are not (e.g. time-of-day)
  - interrupts must be delivered at the same point in command sequence

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 environment 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 until epoch ends

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 is a failover epoch):

$e_b := e_b + 1$

b is promoted primary for epoch e_b

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

Failures and the environment

- No exactly-once guarantee on outputs
- On primary failure, avoid input inconsistencies
  - time must increase monotonically
    - at epoch boundaries, 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 issuing processor delivers a completion interrupt, unless it fails. If the processor fails, the I/O device continues as if the interrupt had been delivered.

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 outstanding I/O operation right before the backup is promoted primary (at the end of the failover epoch)

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

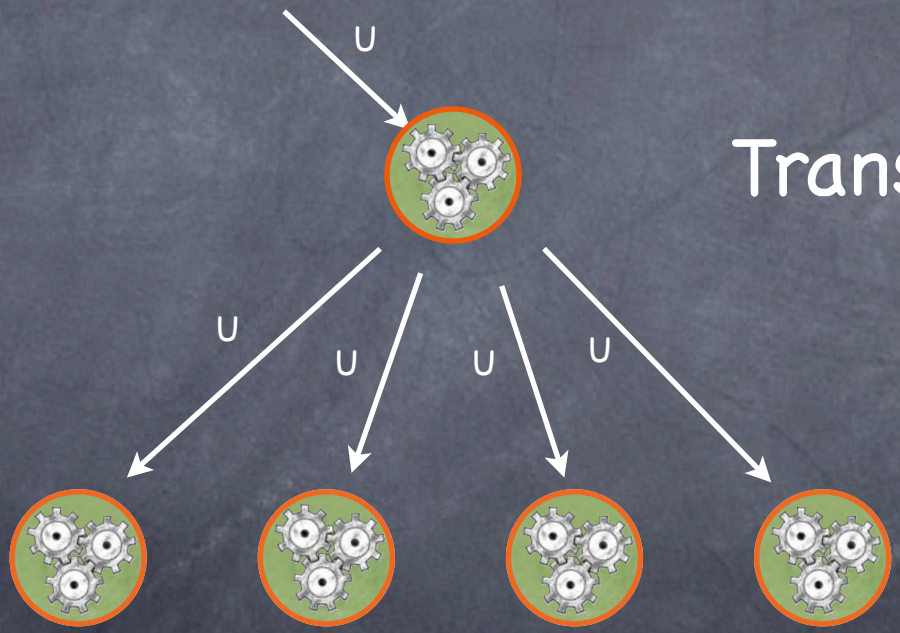
RC in the Hypervisor

- 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

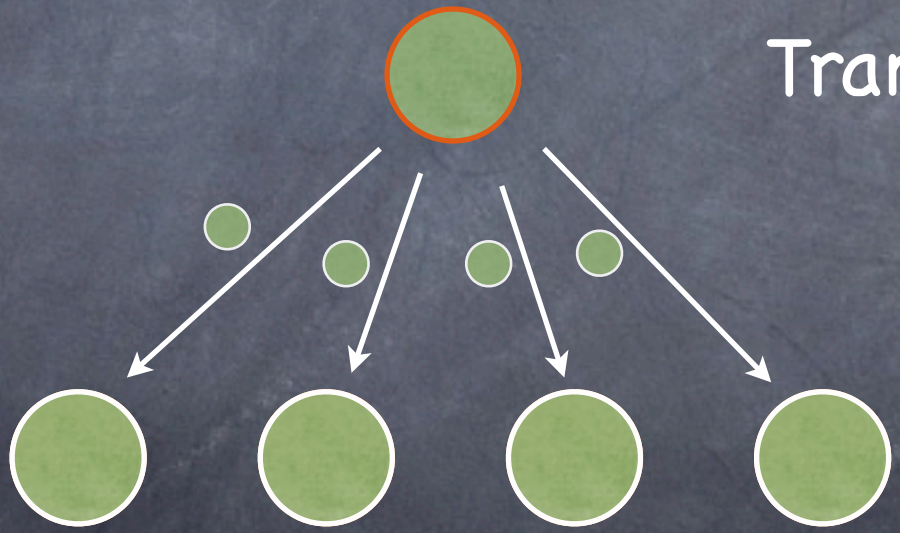
- Optimizations
  - p sends Int immediately
  - p blocks for acks only before output commit

Primary-backup: Updates

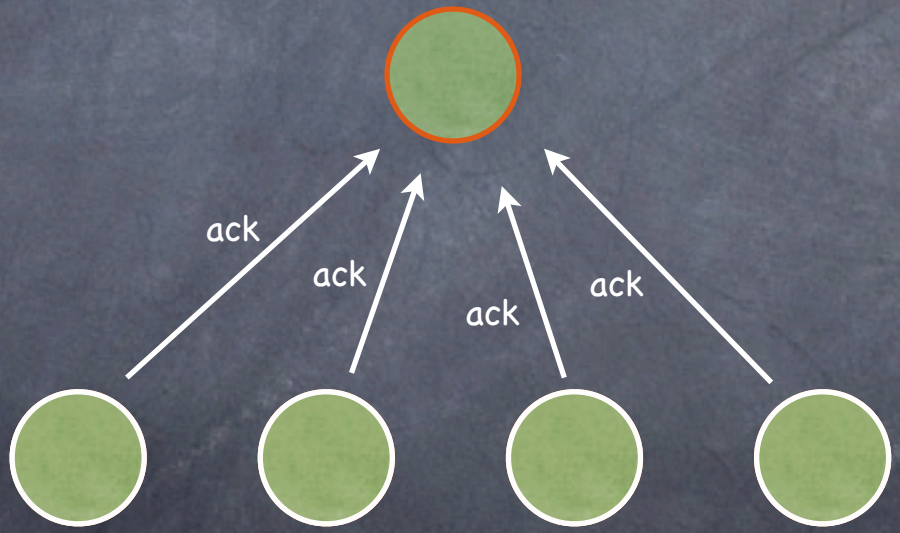


Transfer update

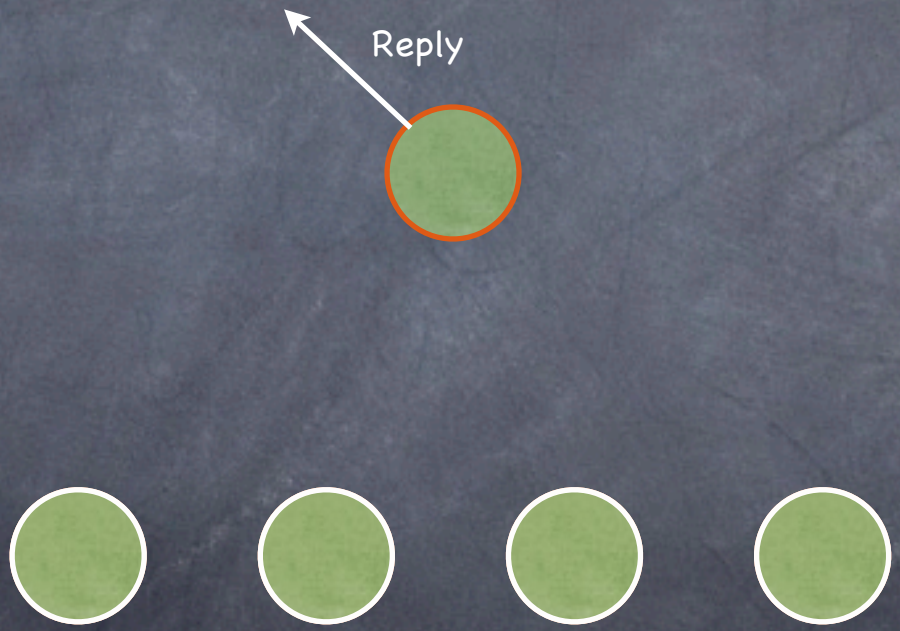
Primary-backup: Updates



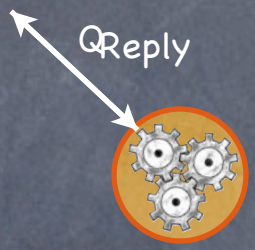
Primary-backup: Updates



Primary-backup: Updates



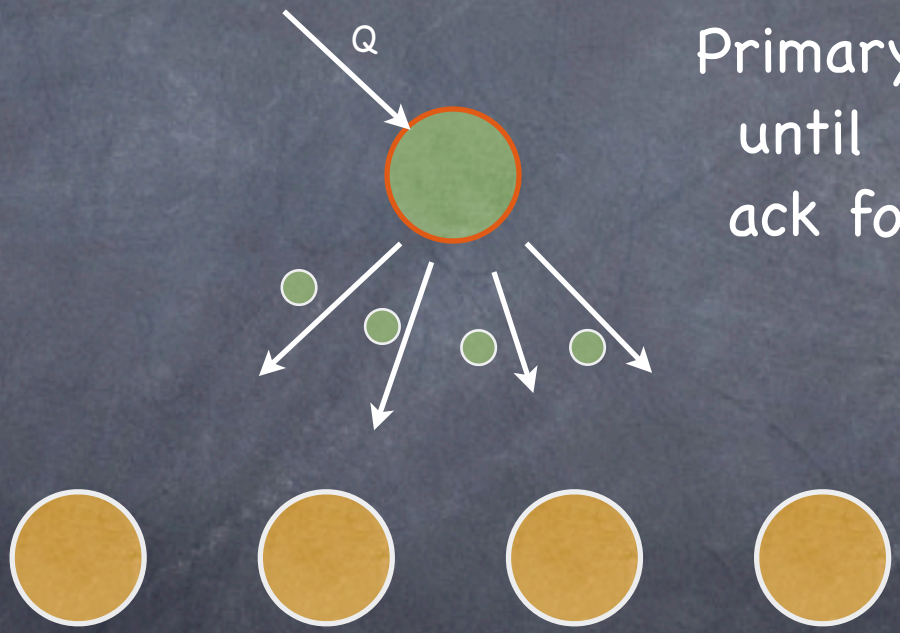
Primary-backup: Queries



However...



Primary-backup: Queries



Primary cannot respond until it has received ack for prior updates!

Chain replication

Van Renesse, Schneider, OSDI '04



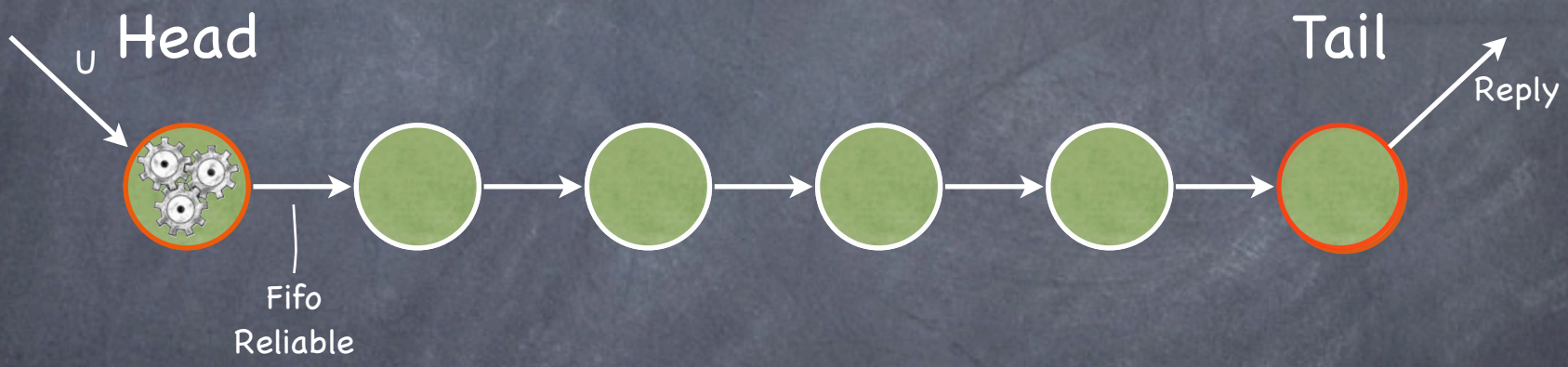
Chain replication

Van Renesse, Schneider, OSDI '04



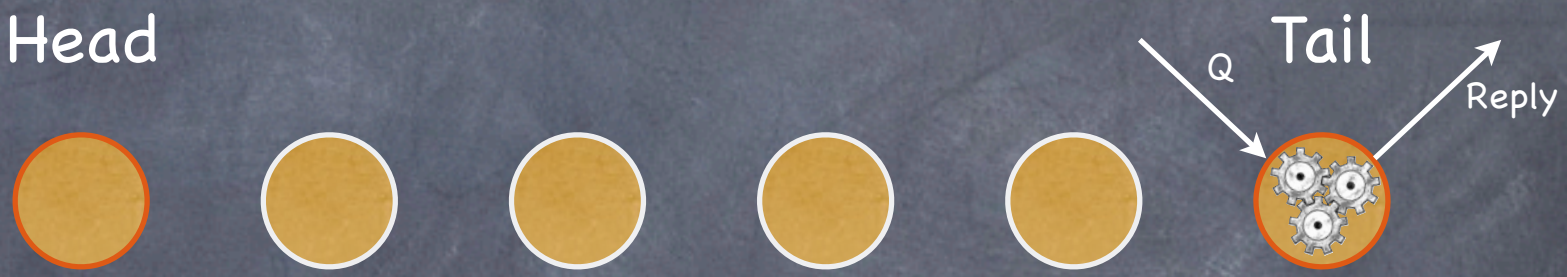
Chain replication: Updates

Van Renesse, Schneider, OSDI '04



Chain replication: Queries

Van Renesse, Schneider, OSDI '04



Furthermore...

Chain replication: Queries

Van Renesse, Schneider, OSDI '04



Tail can respond immediately, without waiting for the ongoing update...

since the tail always reflects the state of the latest completed update

Some like it hot

- Hot Backups process information from the primary as soon as they receive it
- Cold Backups log information received from primary, and process it only if primary fails
- Rollback Recovery implements cold backups cheaply:
  - the primary logs directly to stable storage the information needed by backups
  - if the primary crashes, a newly initialized process is given content of logs—backups are generated "on demand"