Distributed Systems

Spring Semester 2020

Lecture 8: Primary-Backup Replication

John Liagouris
liagos@bu.edu

Replication

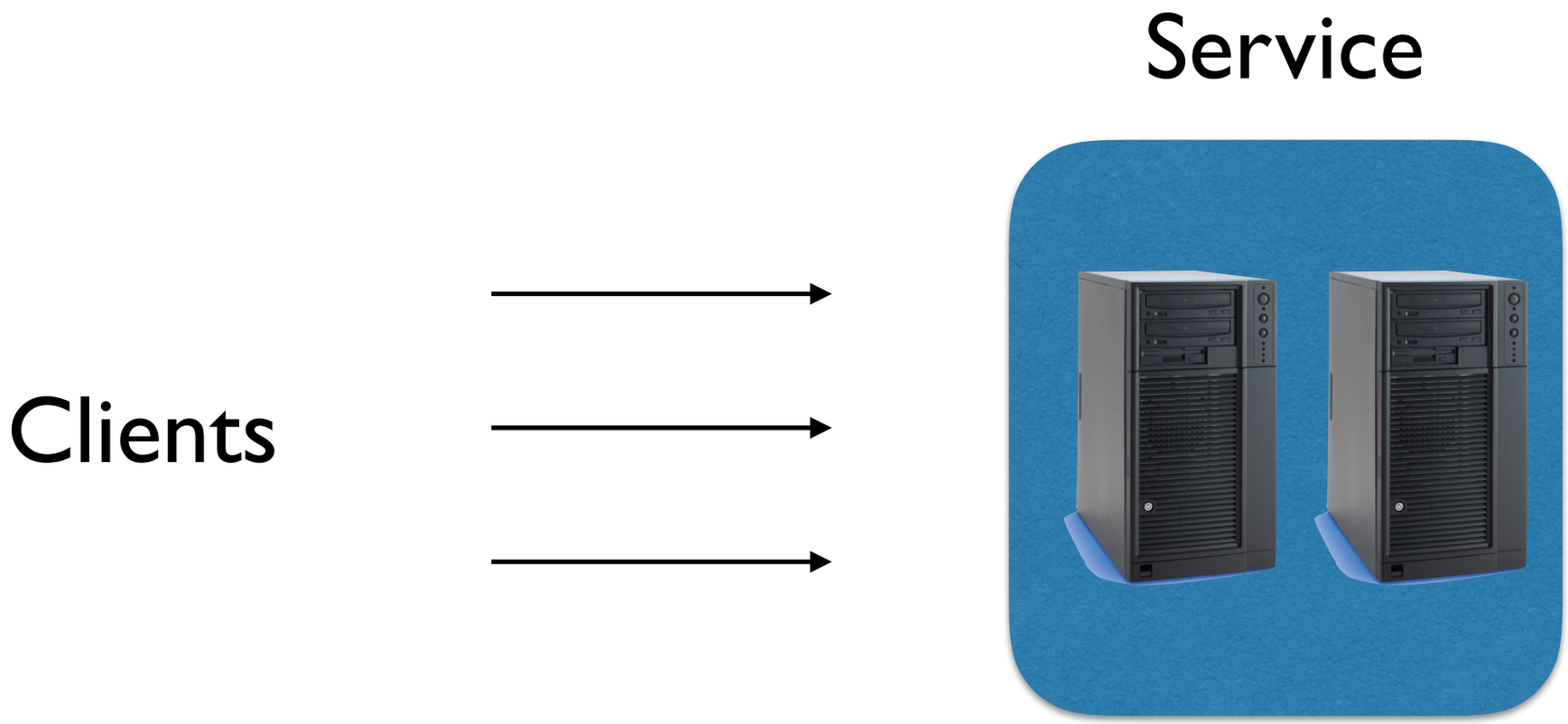
- Replication for Fault Tolerance
  - Ideally : System keeps work even when things fail
  - Consider Master of Map Reduce
    - Single point of Failure — 100's of workers useless if Master fails
    - Like to build systems without single points of failure
- Several approaches but perhaps easiest to consider — general purpose — replicated service

Basic idea

Service

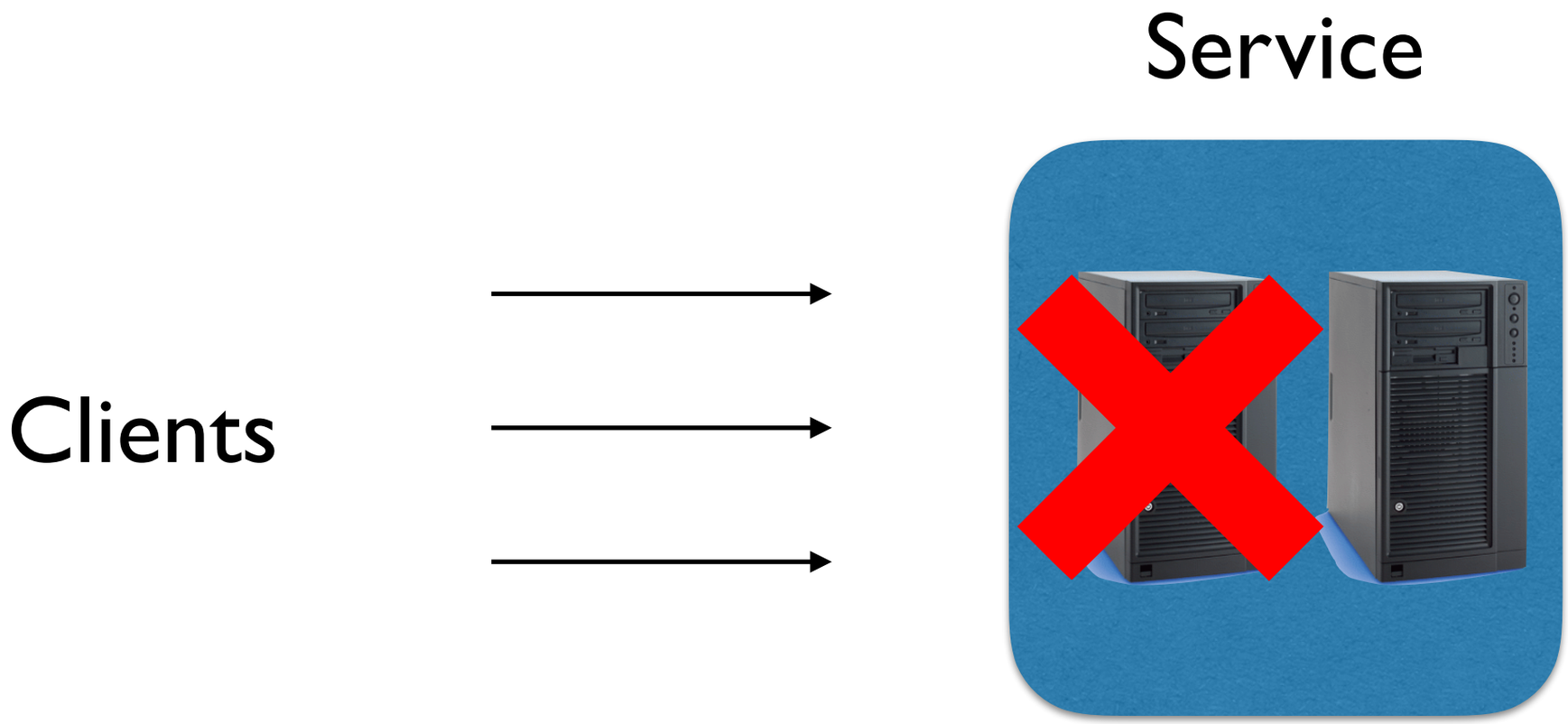
Clients

Basic Idea



Two computers run your service

Replication



Some how arrange for the second computer to take over from the first if it ever suffers a failure

High Level — Two things we are after

- Available — if first system fails would like the second to take over and pickup immediately while it make take hours to fix the first
- Correct in the face of replication — lots of difficulties around correctness

Correctness

- Basic Problem:
  - When we replicate — create a copy
  - very hard to avoid a window in which they do not match — ensure same ALL the time
  - So very hard to ensure that clients never notice that a failure has happened and cut-over occurred

Total transparency is very difficult — more often than not weaken interface to clients to make it easier

Failure Model

Never going to build a system that will run no matter what
Rather it will survive this particular set of faults and not this other set

- Independent Fail Stop
- Fail Stop: Assume computers execute correctly for awhile then halt — very different than considering incorrect execution and non-halting
- Independent: The failure of one computer does not change the likelihood of the failure of another — necessary assumption for replication to be sensible

Are failures independent?

Perhaps the most common correlate failure

— Site wide power failure —

Failure Model

The two we focus on

- Independent Fail Stop
- Site-wide power failures

Most systems have one story for Independent failures and another for Site-wide power failures usually involving disk drives

Big Questions to consider as we look at systems

1. What state is replicated? — back up has to have some state so it can take over.

2. How do the replicas get state?

3. When to cut-over? can't directly inspect computers can only use network messages — failure detection — fraught with difficulty

4. What Anomalies are visible to the clients if cut-over occurs? — lost requests, duplicate requests, …

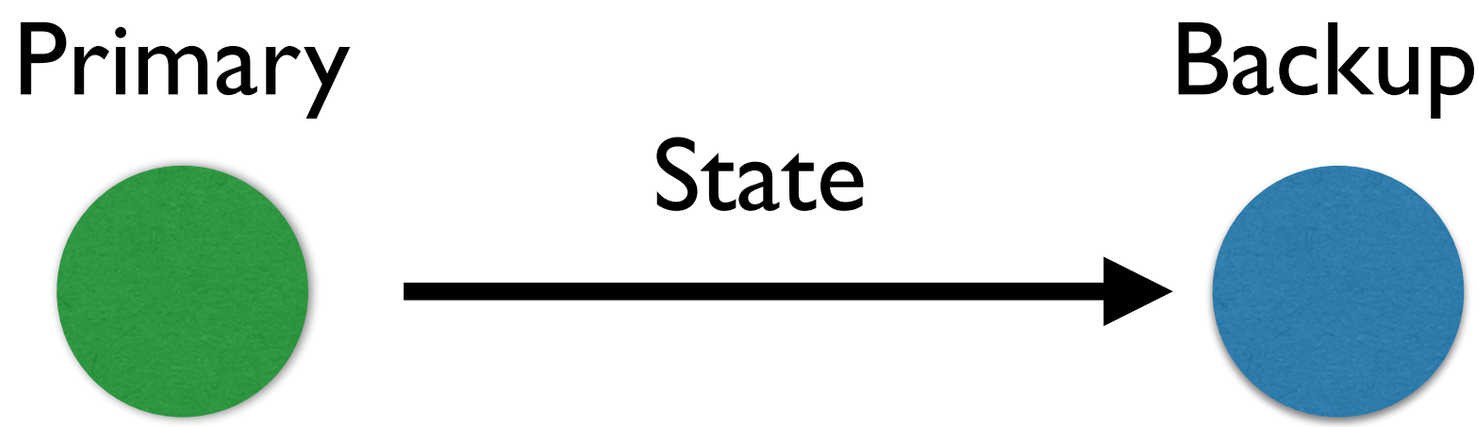
5. How is re-integration handled? — restore a failed system after repair

We are going to organize systems into two main types

Types

1. State Transfer Systems

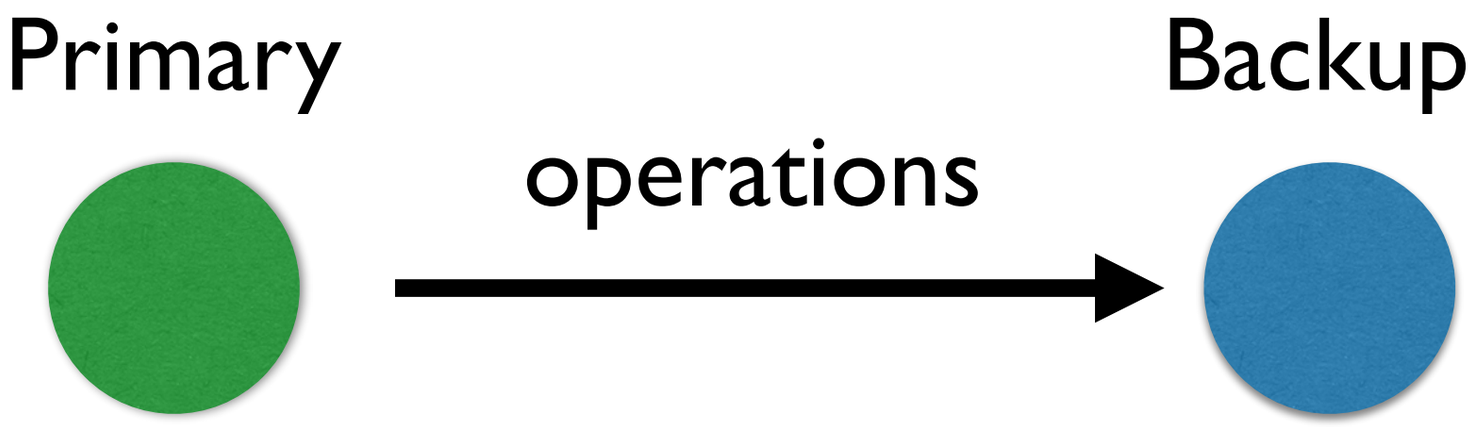
- Primary execute and updates its state and periodically transfers to backups



Types

2. Replicated State Machines

- Both Primary and Backup execute the requests/operations and their state evolves consistently



Clearly some assumptions : deterministic operations, operations are done in the same order, etc

Why might we prefer Replicated State Machines (RSM) at times?

State transfer is simpler

But state maybe large

— slow to transfer

RSM can be more efficient
— if operations are small
compared to data
But complex to get right

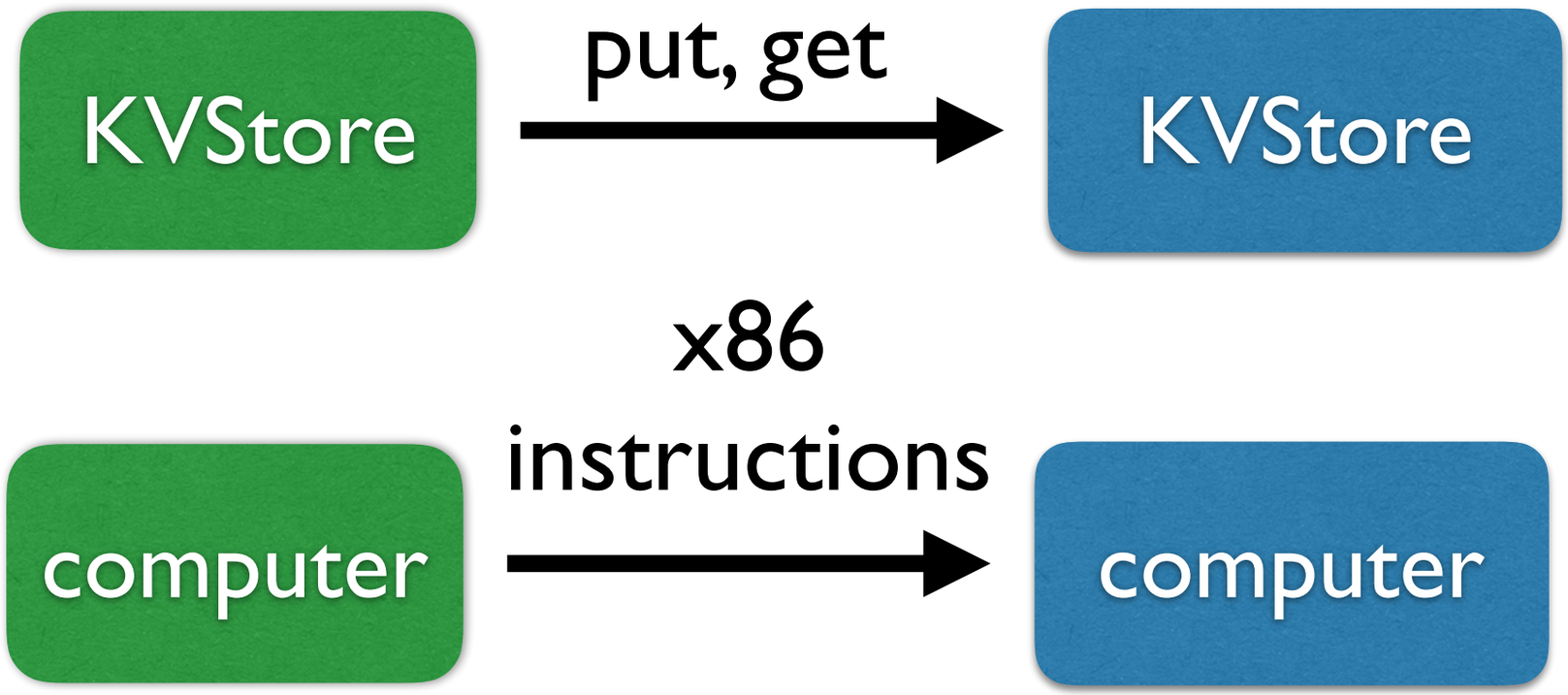
Lots of computations is **NOT** Deterministic
Must take special care to restrict and or ensure
determinism

RSM can be more efficient
— if operations are small
compared to data
But complex to get right

Lots of computations is **NOT** Deterministic
Must take special care to restrict and or ensure
determinism

In Practice Replicated State Machines are more common approach — lighter weight vm-ft and Labs uses RSM

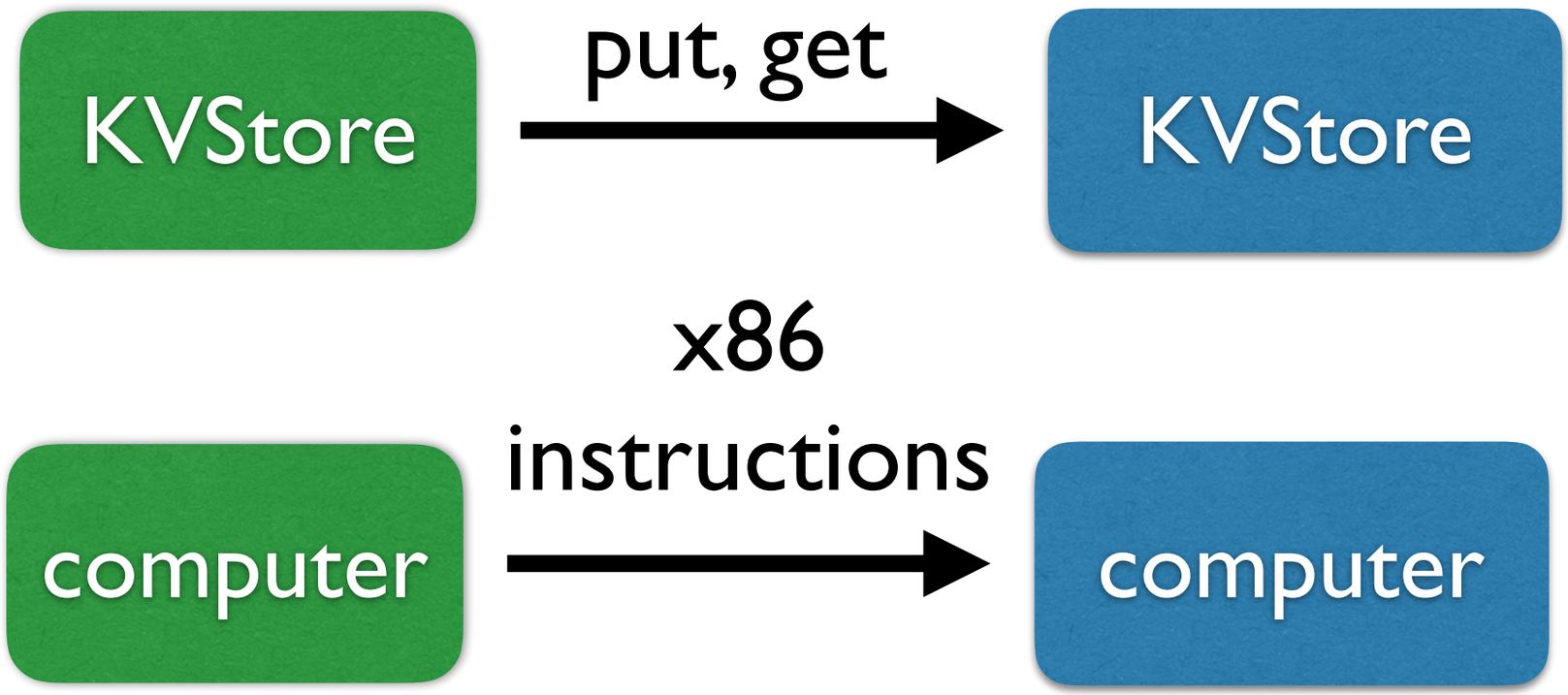
RSM



Choices impact 1) performance and 2) ease of implementation

- How much interaction is necessary
- Dealing with non-determinism is hard on x86 level but easier on put/get level

RSM



Choices impact 1) performance and 2) ease of implementation

- How much interaction is necessary
- Dealing with non-determinism is hard on x86 level but easier on put/get level

BUT higher-level RSM works only for apps written to higher-level interface

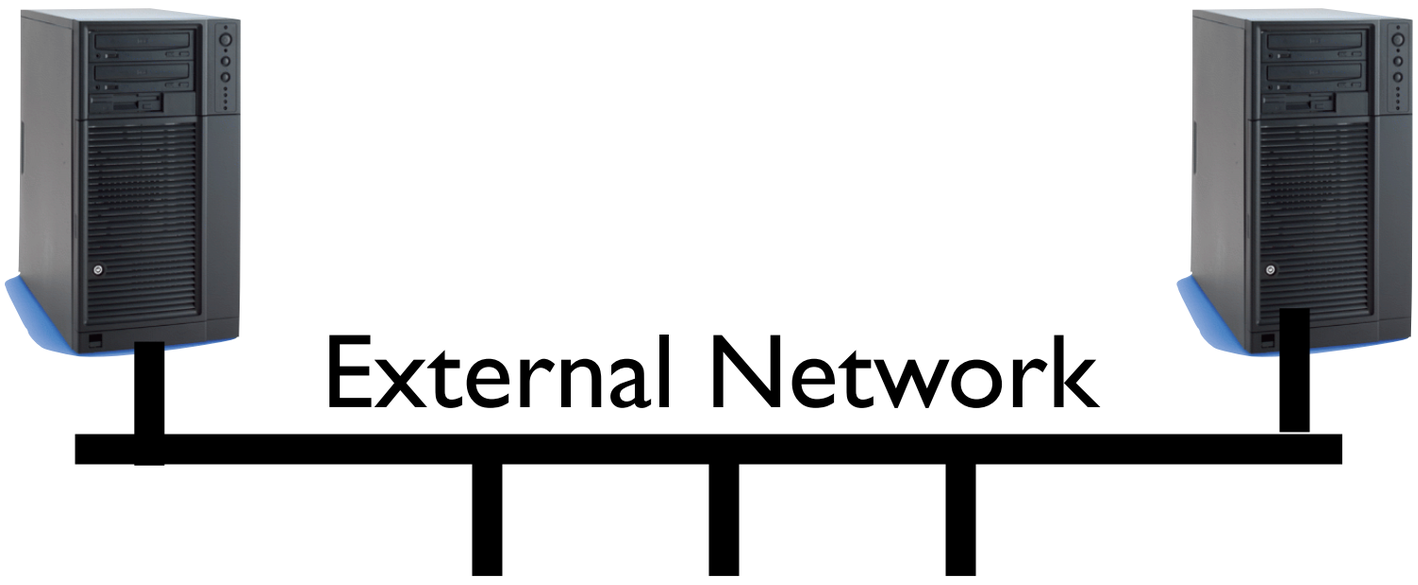
x86 RSM can make any x86 program FT

VM-FT

- Super ambitious :
  - Take any software (OS and APP) and make it fault tolerant
  - Like a Magic Wand
- Costs are high but very cool given what it's goals are

Overview

Servers on Internal Network

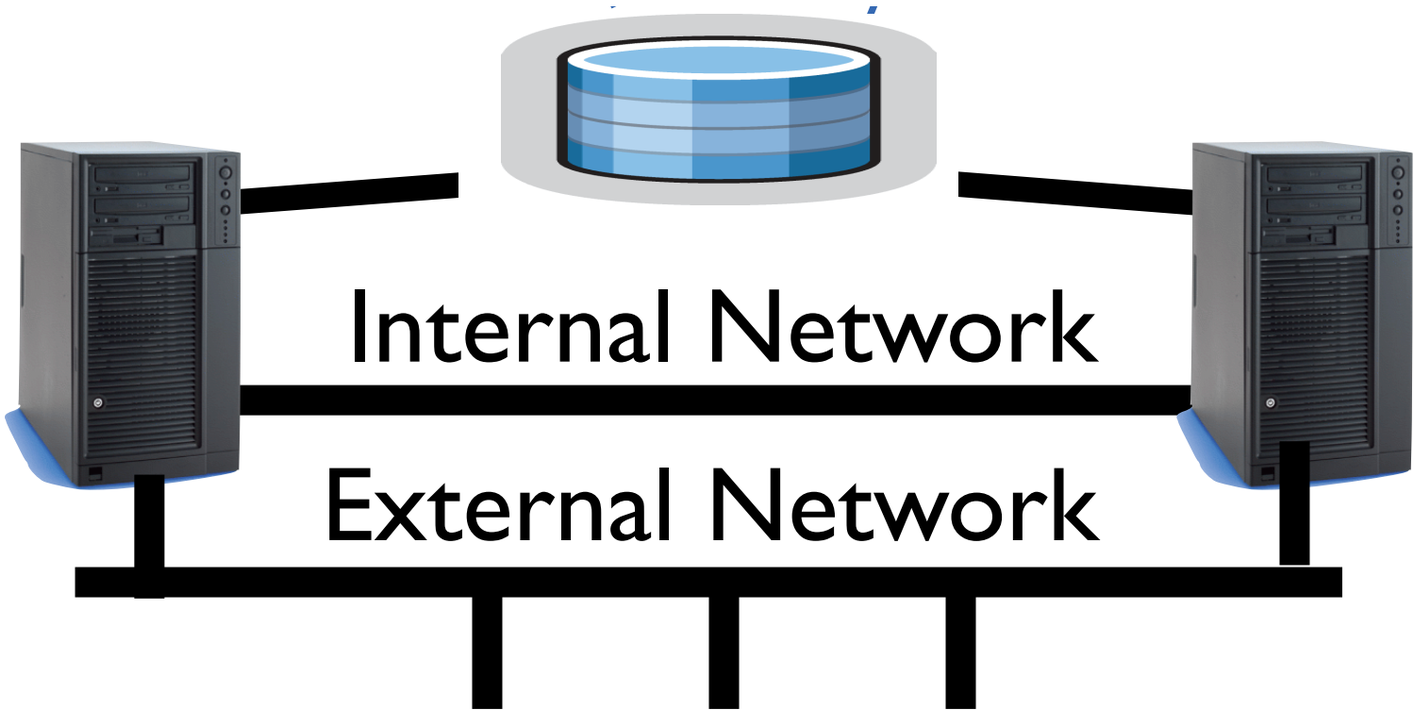


Client Computer
connected to external Network

Overview

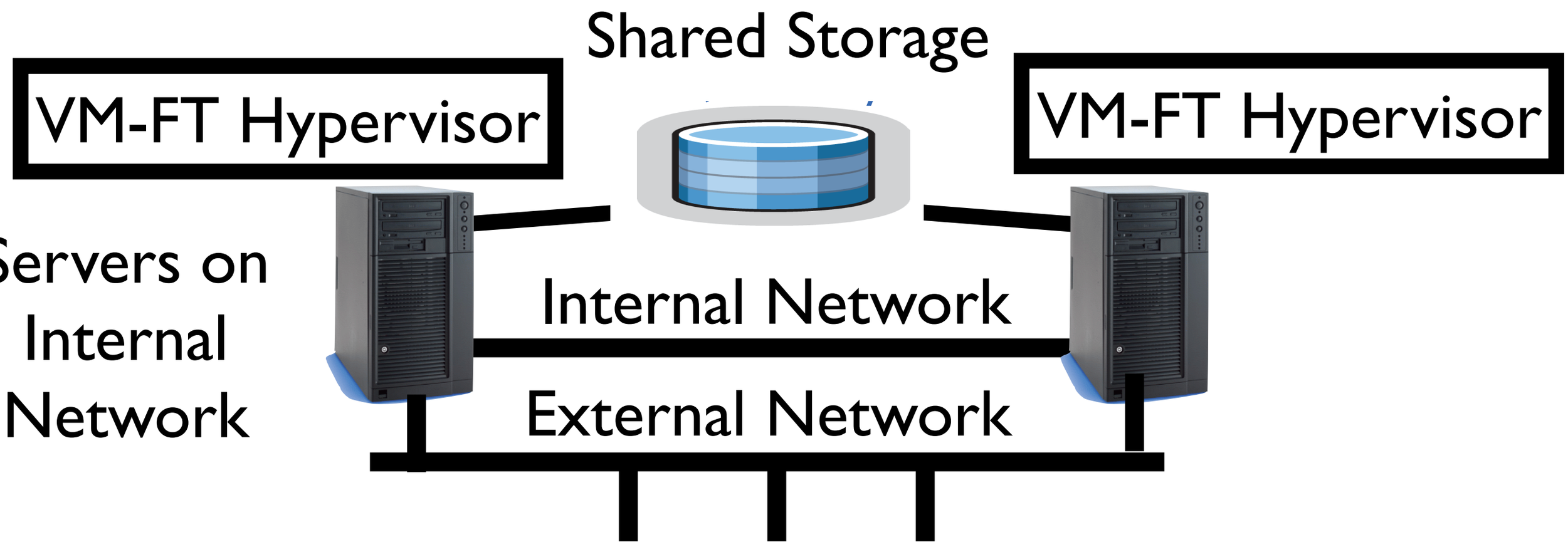
Shared Storage

Servers on Internal Network



Client Computer
connected to external Network

Overview


Shared Storage
VM-FT Hypervisor
VM-FT Hypervisor
Servers on Internal Network
Internal Network
External Network
Client Computer
connected to external Network

Overview

Primary VM

APPS

OS

VM-FT Hypervisor

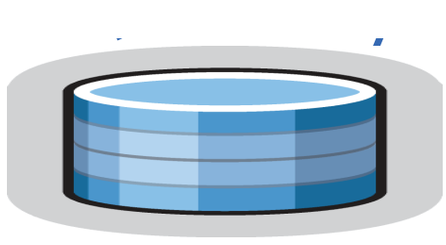
Backup VM

APPS

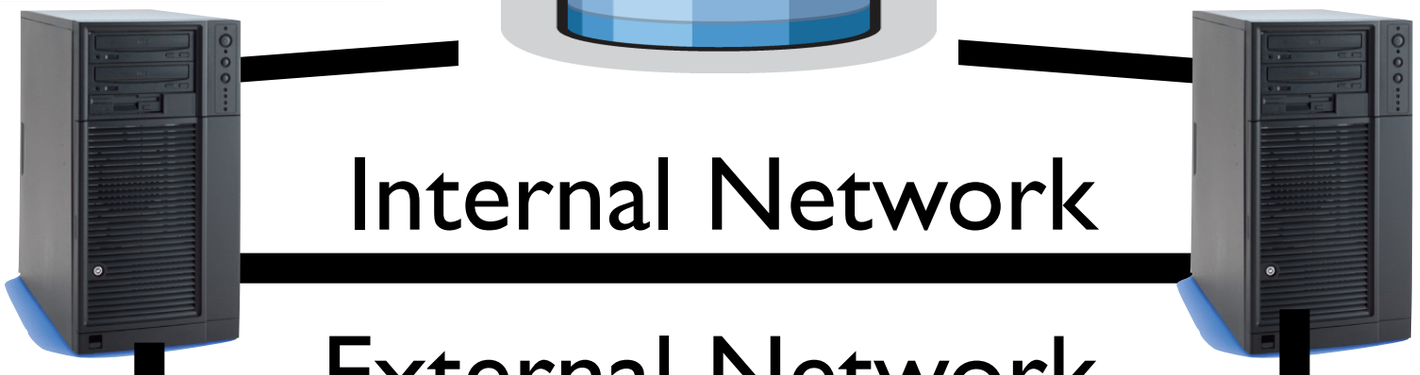
OS

VM-FT Hypervisor

Shared Storage



Servers on Internal Network



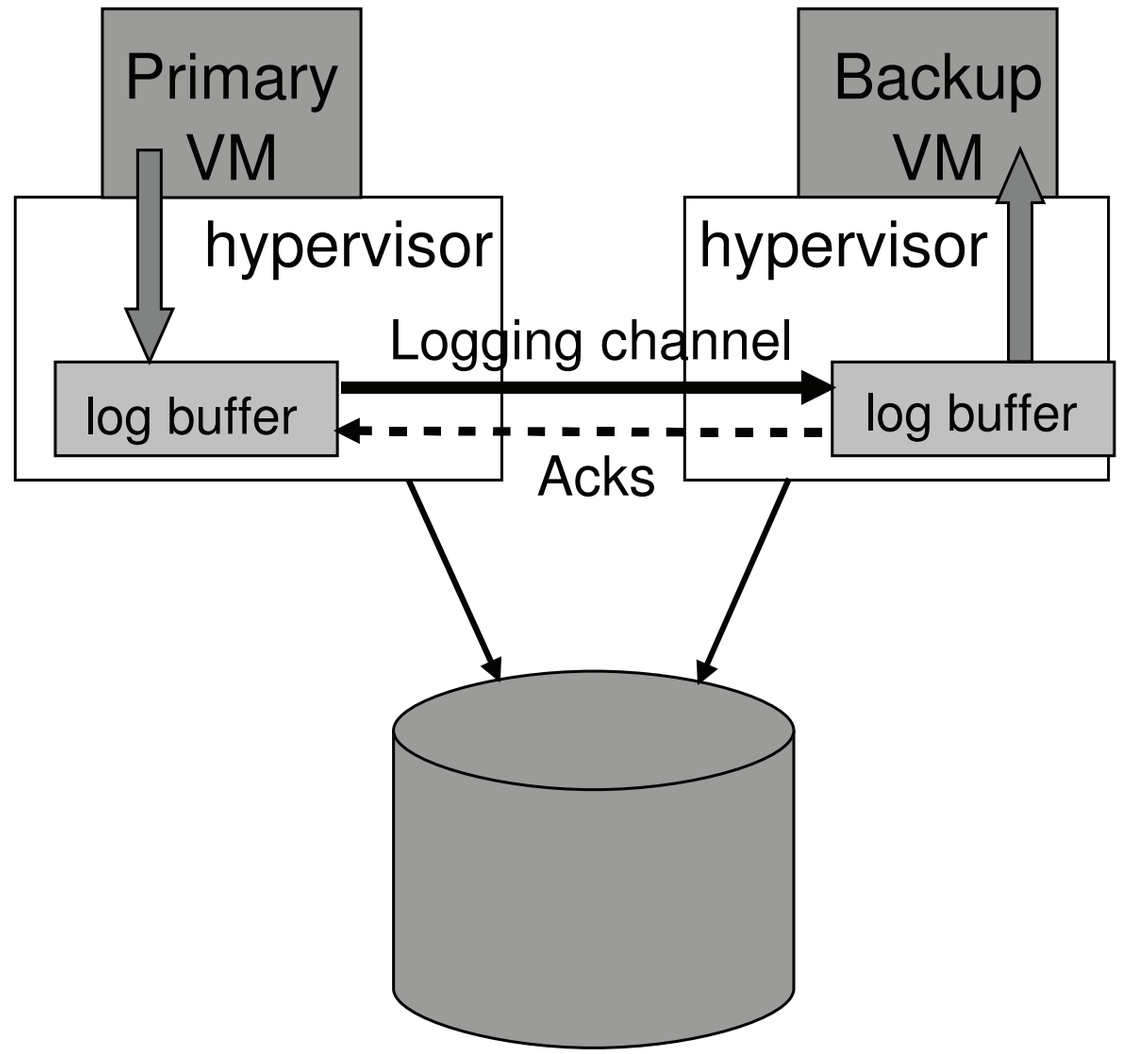
Internal Network

External Network

Client Computer
connected to external Network

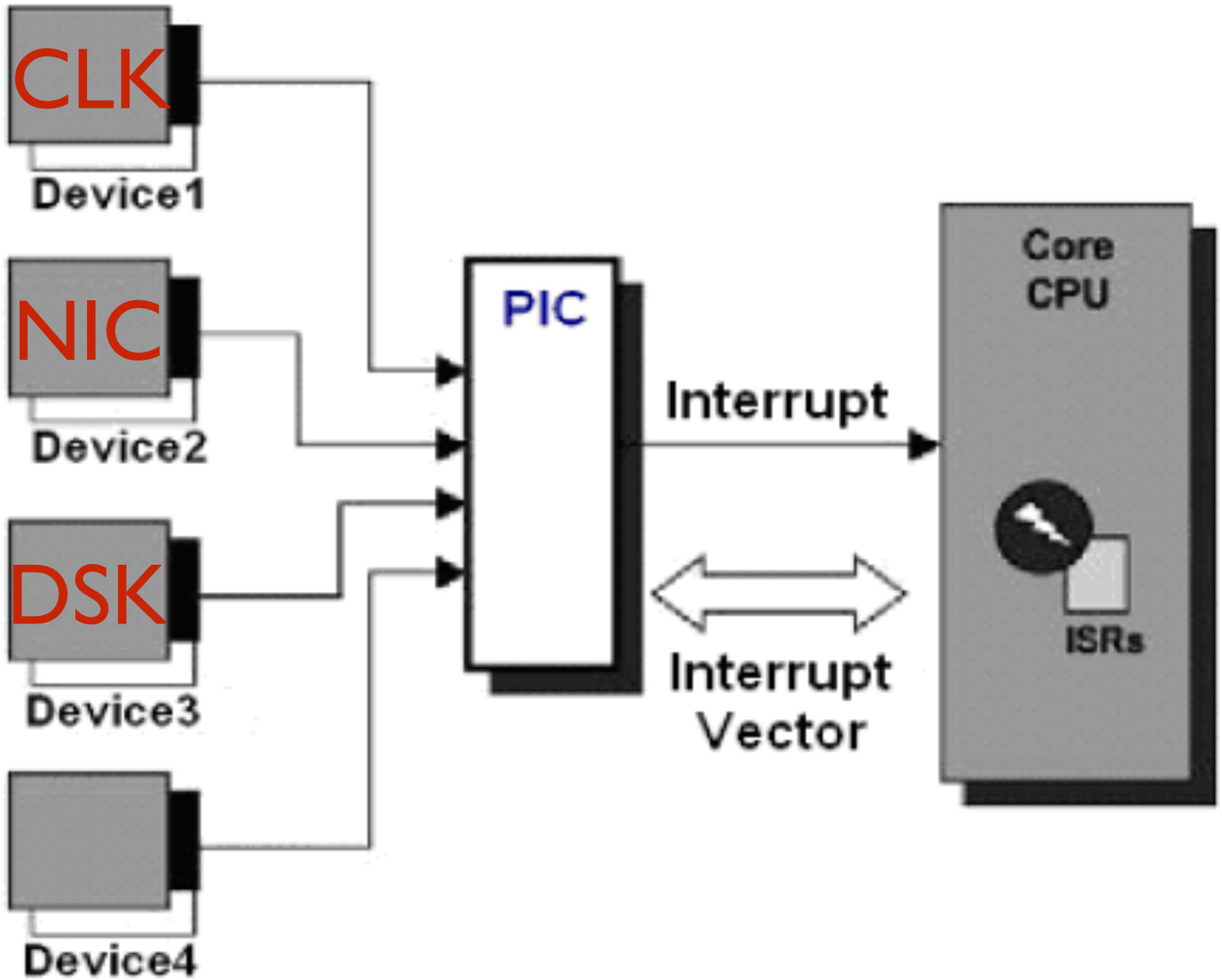
Overview

- Backup in “lock step” with primary
  - Primary sends all input to backup
  - output of backup is dropped
  - heart beats between primary and backup
    - if primary fails start backup executing



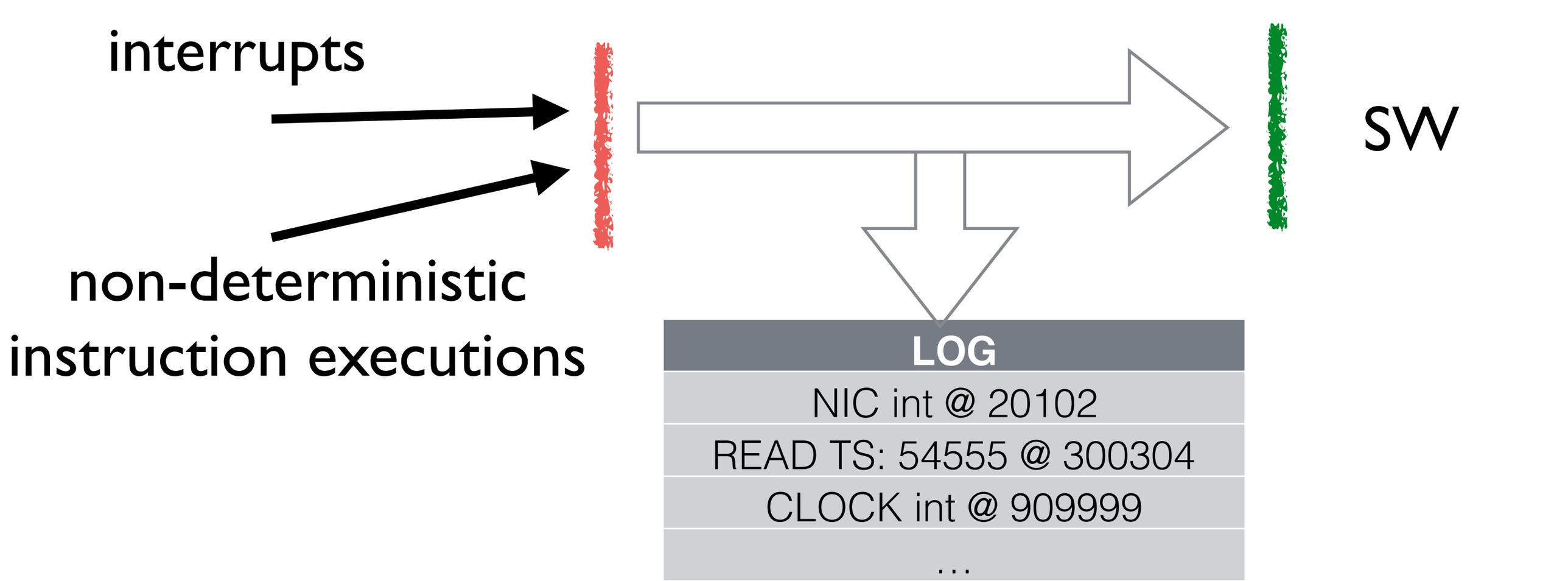
Inputs

- Clock interrupts
- Network interrupt
- Disk interrupts



Logging

- Log all hardware events into a log



EXAMPLE

APPS

OS

APPS

OS

VM-FT Hypervisor

VM-FT Hypervisor

NIC

INT

"Hello"

Network Packet

APPS

OS

APPS

OS

log Entry
NIC INT@n
DATA:"HELLO"

VM-FT Hypervisor

log channel

VM-FT Hypervisor

APPS

OS

APPS

OS

NIC:

INT:

"Hello"

log Entry
NIC INT@n
DATA:"HELLO"

VM-FT Hypervisor

VM-FT Hypervisor

log channel

APPS

"Hello"

OS

APPS

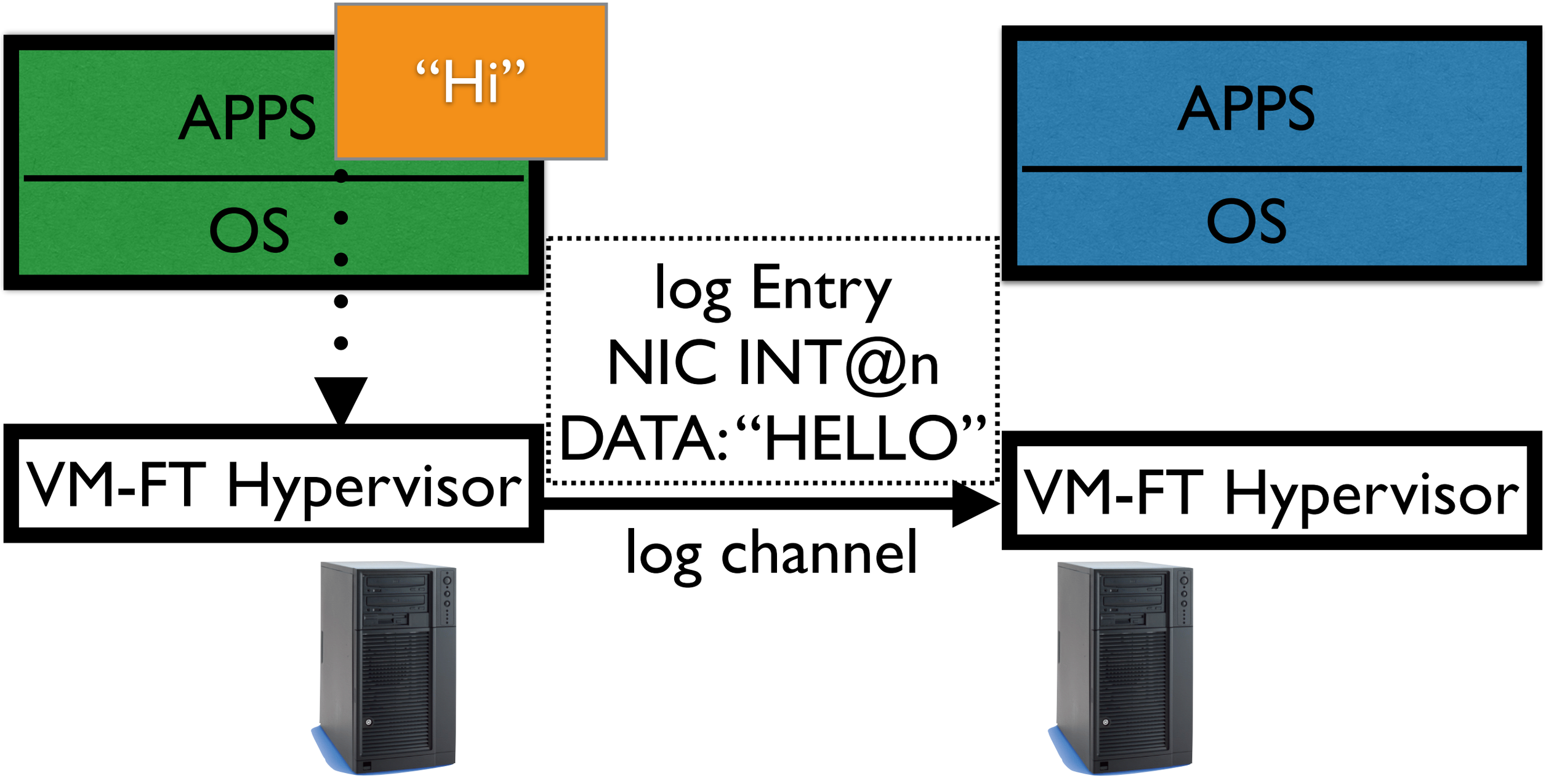
OS

log Entry
NIC INT@n
DATA:"HELLO"

VM-FT Hypervisor

log channel

VM-FT Hypervisor


APPS
OS
"Hi"
log Entry
NIC INT@n
DATA:"HELLO"
APPS
OS
VM-FT Hypervisor
log channel
VM-FT Hypervisor

APPS

OS

APPS

OS

log Entry
NIC INT@n
DATA:"HELLO"

VM-FT Hypervisor

log channel

VM-FT Hypervisor

"Hi"

APPS

OS

VM-FT Hypervisor

log Entry
NIC INT@n
DATA:"HELLO"

log channel

APPS

OS

"Hello"

NIC:
INT:

VM-FT Hypervisor

On Backup

APPS

OS

"Hello"

APPS

OS

VM-FT Hypervisor

log channel

VM-FT Hypervisor

On Backup

APPS

OS

APP

"Hi"

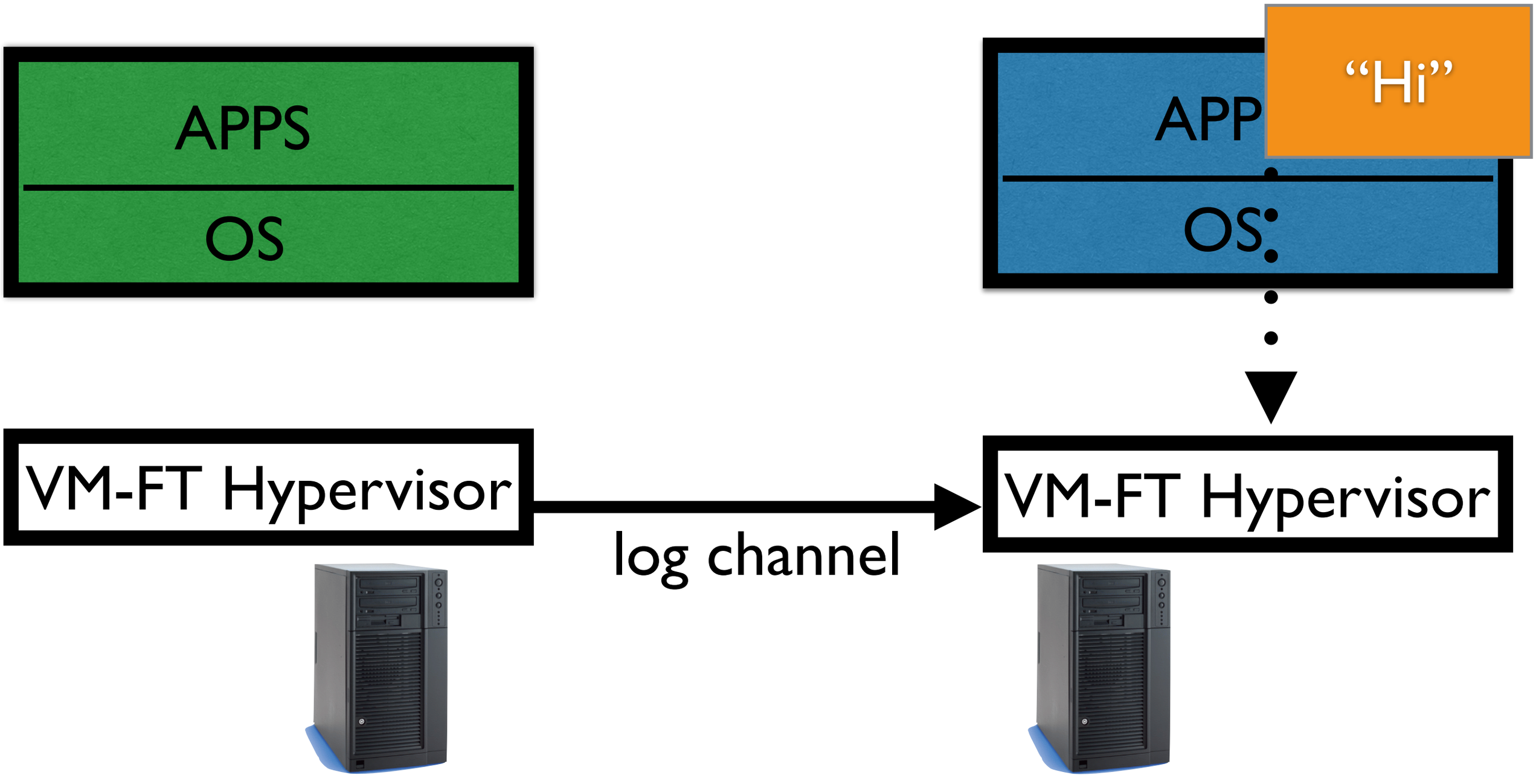
OS

VM-FT Hypervisor

log channel

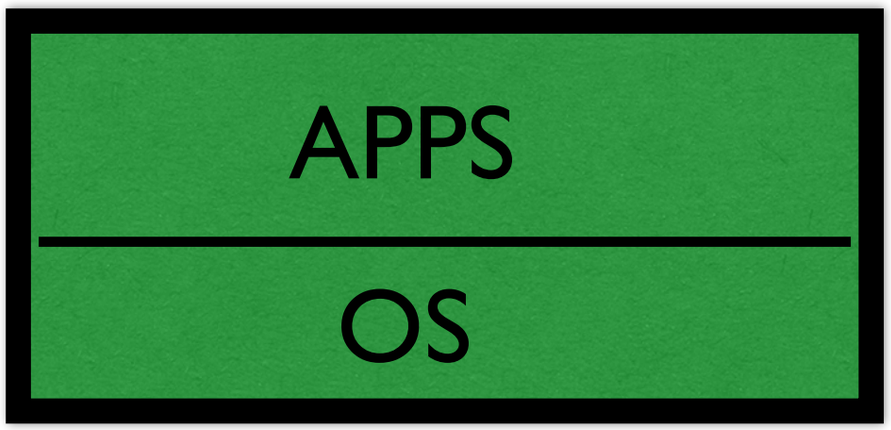
VM-FT Hypervisor

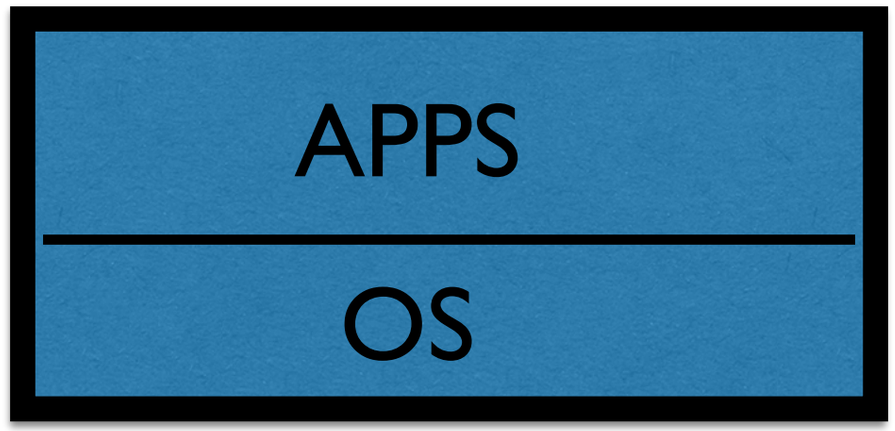
On Backup



does *not* put response on the wire

On Backup





VM-FT Hypervisor

log channel

VM-FT Hypervisor

does NOT put response on the wire

— ignores local clock gets from primary

— primary and backup same input end up in same state

Challenges

1. Making it look like a single reliable server from the outside

2. How to avoid two primaries?

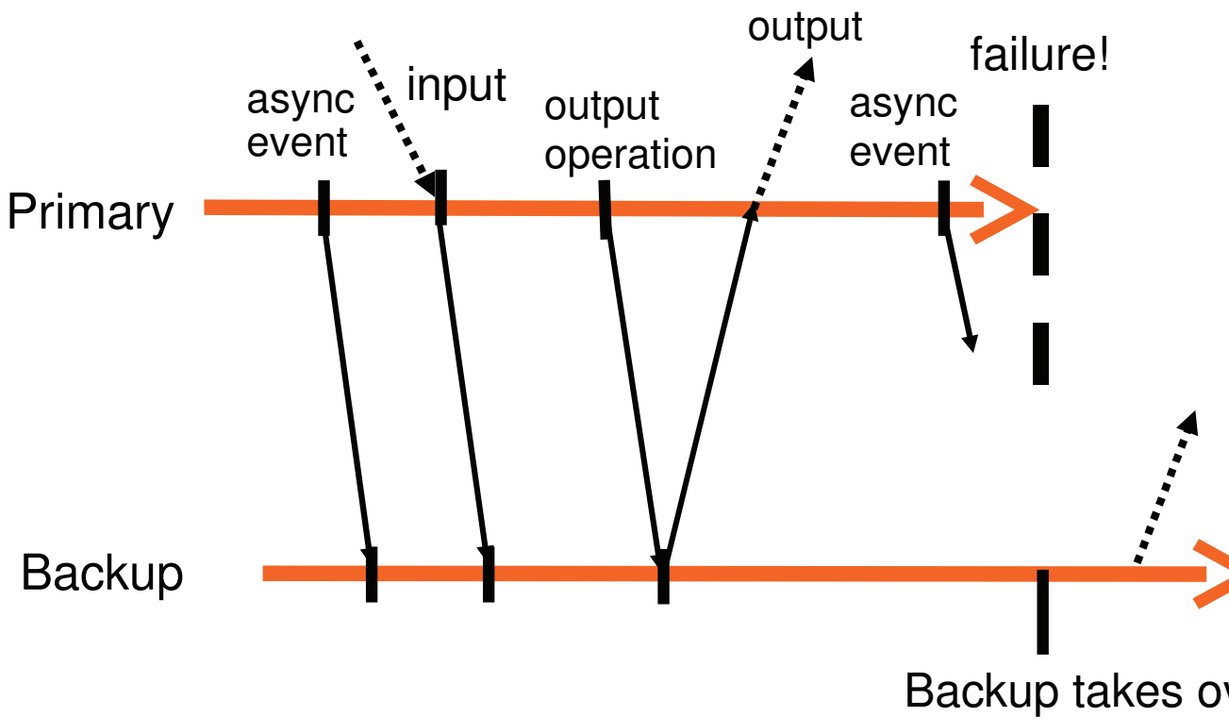
3. How to make sure backup keeps up with primary?

Challenge 1:

Making it look like a single reliable server from the outside

Primary delays output until backup acks

- log each output op
- only send after backup acked receiving output op

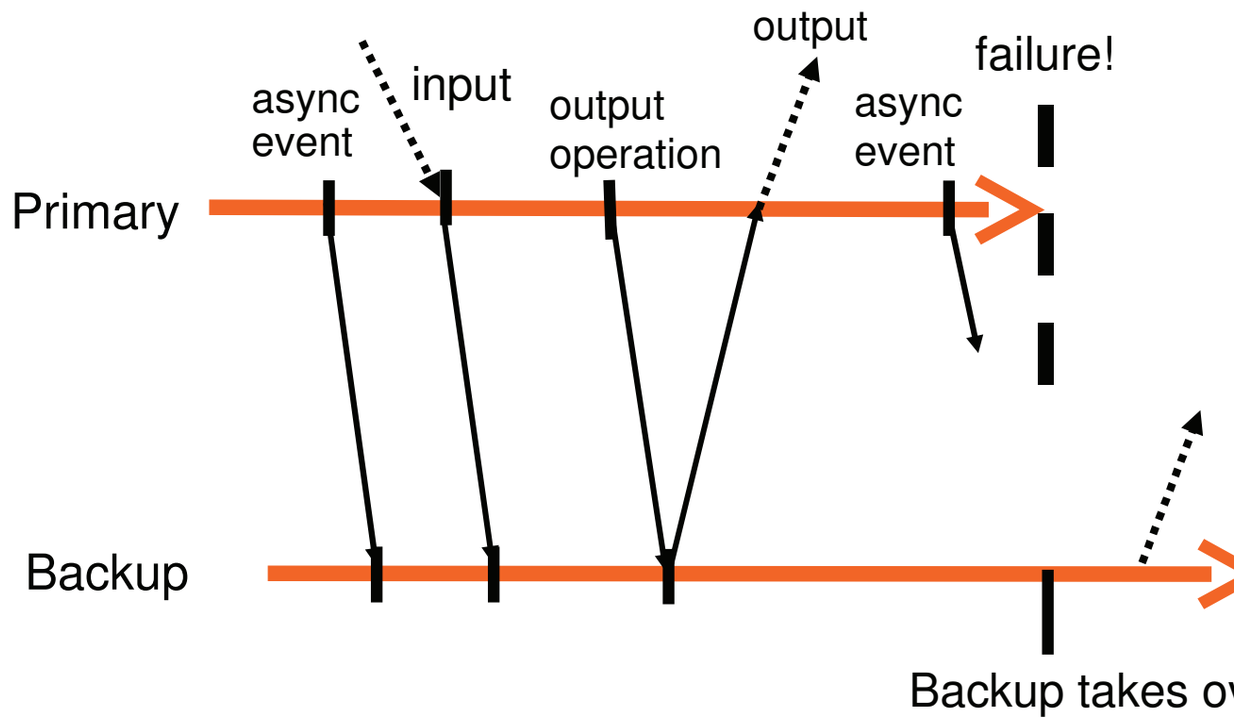


Challenge 1:

Making it look like a single reliable server from the outside

Primary delays output until backup acks

- log each output op
- only send after backup asked receiving output op



perf opt: primary continues to execute after output buffers output till backup ack

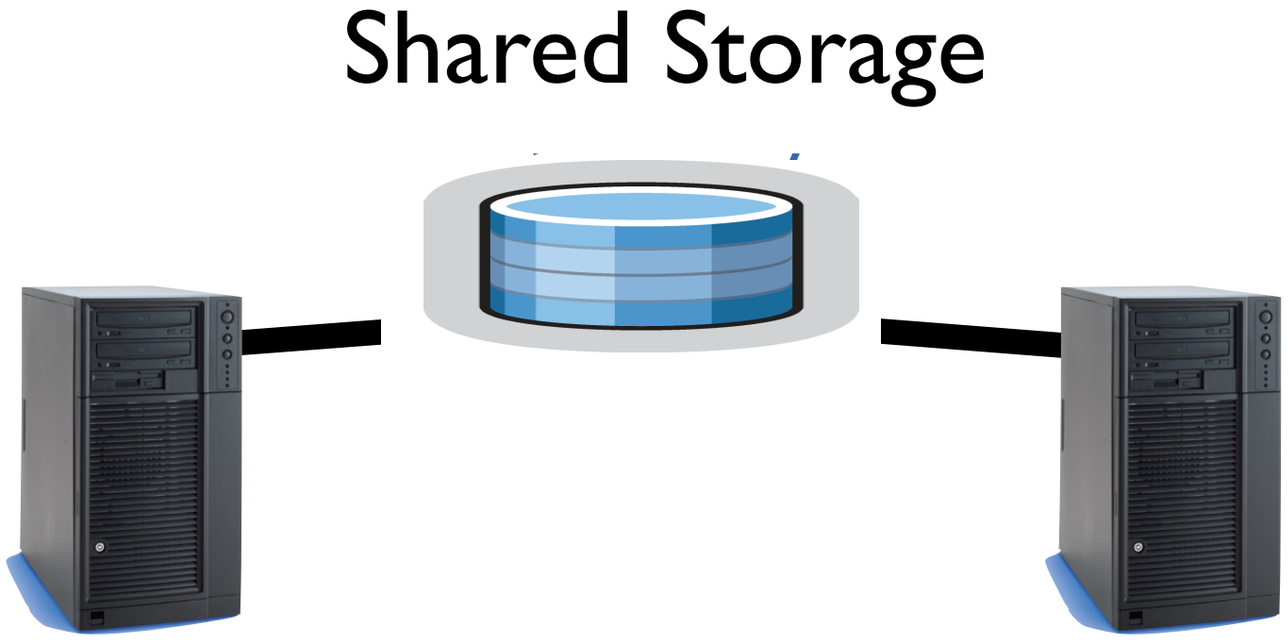
What happens when primary fails right after sending the reply to the client and the backup takes over?

The backup doesn't know if client did receive a reply, so it will send the reply again when it becomes a primary

What happens when primary fails after receiving network input but before sending a corresponding log entry to backup?

Challenge 2: How to avoid two primaries? (Split Brain)

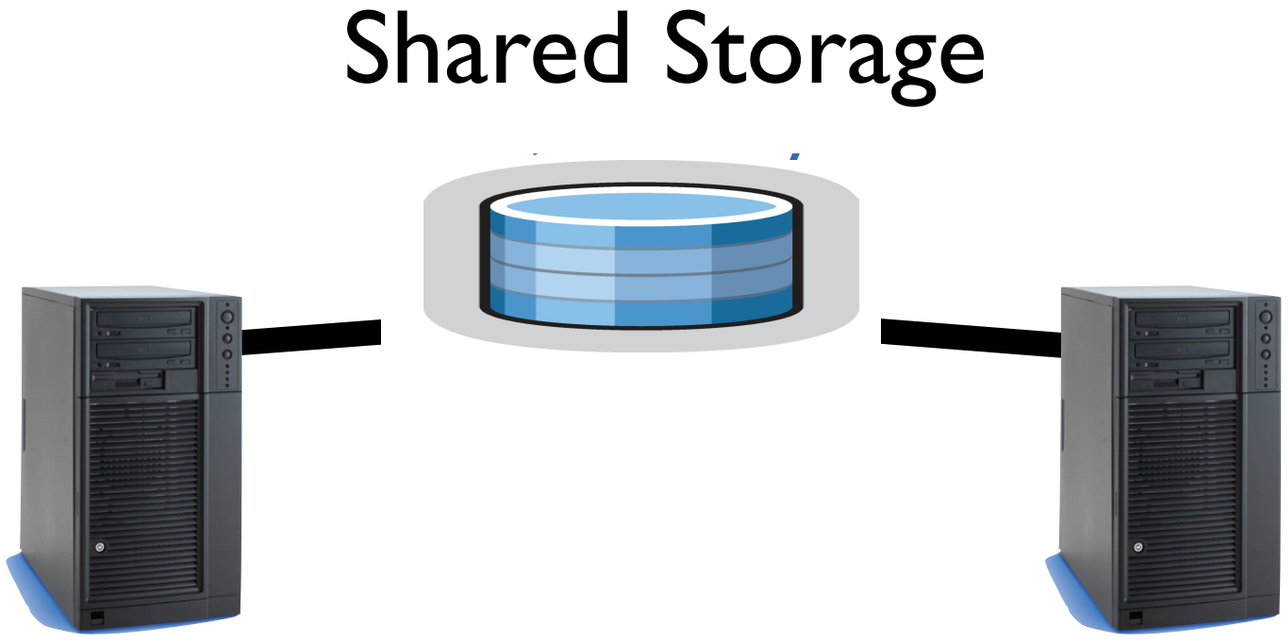
Split Brain



- Hard problem with only unreliable network!
- This is the point of things like Raft

Assume shared disk
Backup replays through last log entry
Backup atomically test-and-set "live" variable on disk
If set, primary is alive. Commit suicide
If not set, primary is dead. Become primary
If primary, create new backup from checkpoint
Using VMotion

Split Brain



- Hard problem with only unreliable network!
- This is the point of things like Raft

Assume shared disk
Backup replays through last log entry
Backup atomically test-and-set "live" variable on disk
If set, primary is alive. Commit suicide
If not set, primary is dead. Become primary
If primary, create new backup from checkpoint
Using VMotion

Challenge 3: Slow backup keeps up

- If Primary is much faster than backup
  - Log buffer will be full
  - Primary will block until there is free space in the buffer
  - VM FT can also slow down Primary proactively, until backup catches up