

Primary/Backup

CS6450: Distributed Systems

Lecture 3/4

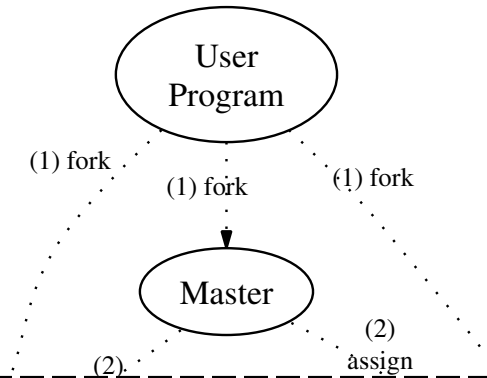
Ryan Stutsman

Material taken/derived from Princeton COS-418 materials created by Michael Freedman and Kyle Jamieson at Princeton University.

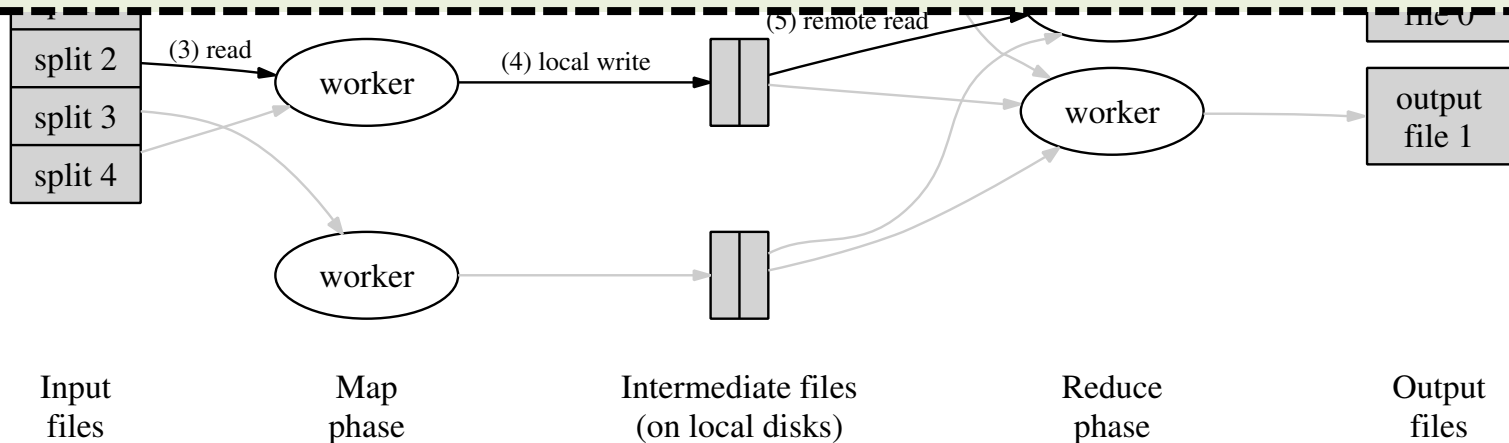
Licensed for use under a Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Unported License.

Some material taken/derived from MIT 6.824 by Robert Morris, Franz Kaashoek, and Nickolai Zeldovich.

Simplified Fault Tolerance in MapReduce



MapReduce used GFS, stateless workers, and clients themselves to achieve fault tolerance



Plan

1. Introduction to Primary-Backup replication
2. Case study: VMWare's fault-tolerant virtual machine

Primary-Backup: Goals

- **Mechanism:** Replicate and separate servers
- **Goal #1:** Provide a highly reliable service
 - Despite some server and network failures
 - **Continue operation** after failure
- **Goal #2:** Servers should behave just like a single, more reliable server

State machine replication

- Any server is essentially a **state machine**
 - Set of (key, value) pairs is **state**
 - Operations **transition** between states
- Need an op to be executed on all replicas, or none at all
 - *i.e.*, we need **distributed all-or-nothing atomicity**
 - If op is deterministic, replicas will end in same state
- **Key assumption:** Operations are deterministic
 - This assumption can be (carefully) relaxed

Primary-Backup (P-B) approach

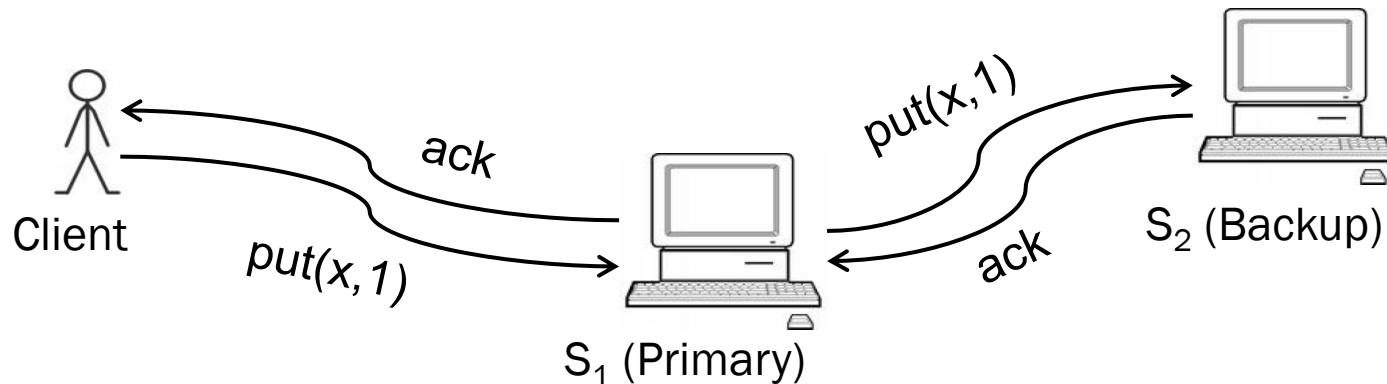
- Nominate one server the *primary*, call the other the *backup*
 - Clients send all operations (get, put) to current primary
 - The primary **orders** clients' operations
- Should be only **one primary at a time**

Need to keep clients, primary, and backup in sync: who is primary and who is backup

Challenges

- Network and server **failures**
- Network **partitions**
 - Within each network partition, near-perfect communication between servers
 - Between network partitions, **no communication between servers**

Primary-Backup (P-B) approach



1. Primary logs the operation locally
2. Primary sends operation to backup and waits for ack
 - Backup performs or just adds it to its *log*
3. Primary performs op and **acks** to the client
 - After backup has applied the operation and ack'ed



View server

- A **view server** decides who is primary, who is backup
 - Clients and servers depend on view server
 - Don't decide on their own (might not agree)
- Challenge in designing the view service:
 - Only want one primary at a time
 - Careful protocol design needed
- For now, **assume** view server **never fails**


Monitoring server liveness

- Each replica periodically *pings* view server
 - View server declares replica *dead* if it missed N pings in a row
 - Considers replica *alive* after single ping
- *Can a replica **be alive but declared “dead”** by view server?*
 - Yes, in the case of network failure or partition

The view server decides the current view

- **View** = (view #, primary server, backup server)

Challenge: All parties make their own local decision of the current view number


Client

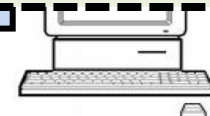
(1, S_1 , S_2)

(2, S_2 , -)

(3, S_2 , S_3)



S_3 (Backup)



S_2 (Primary)

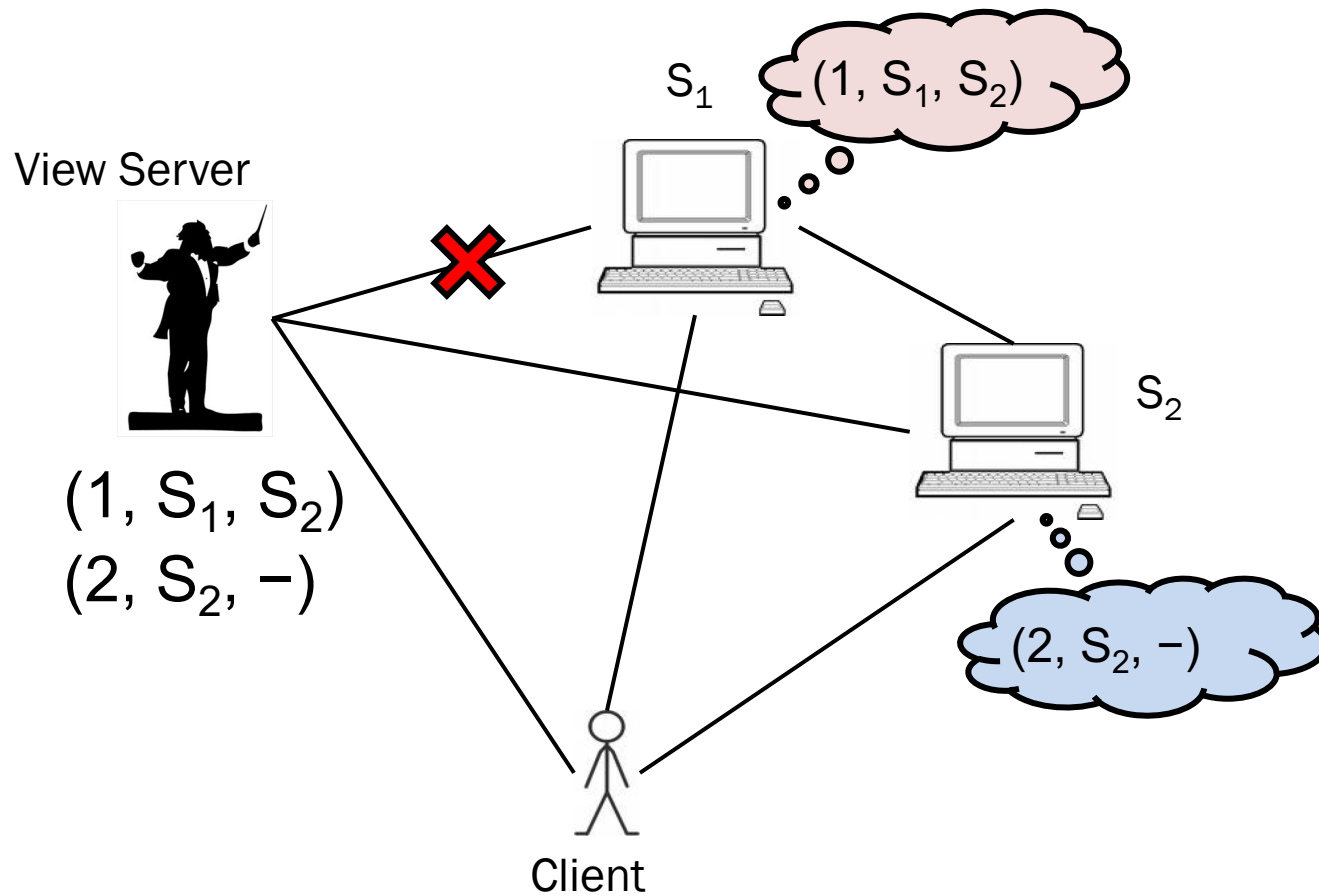
Agreeing on the current view

- In general, any number of servers can ping view server
- Okay to have a view with a primary and **no backup**
- Want everyone to **agree** on the **view number**
 - **Include** the view # in RPCs between all parties

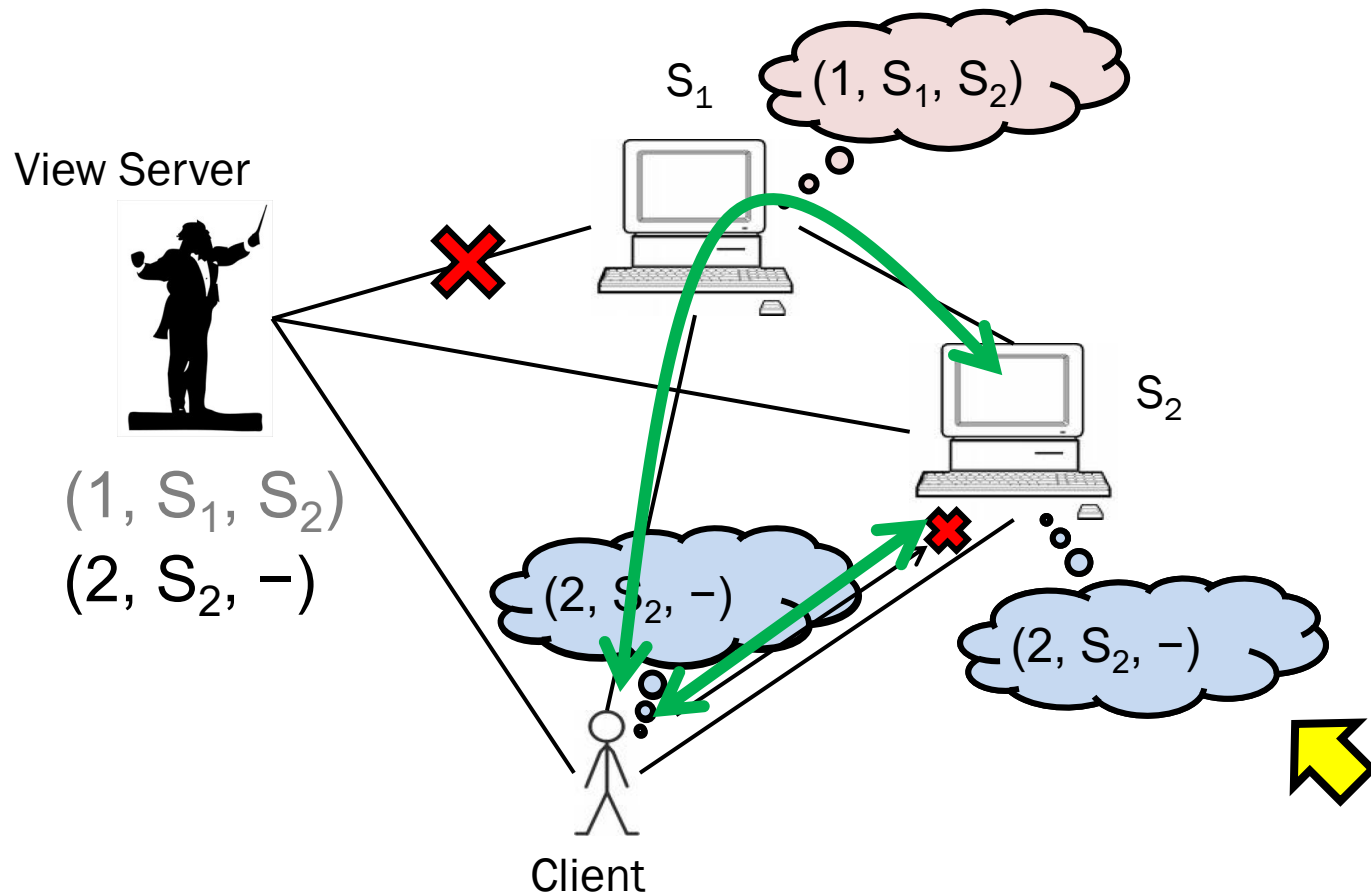
Transitioning between views

- *How do we ensure new primary has up-to-date state?*
 - Only promote a previous backup
 - *i.e.*, don't make a previously-idle server primary
 - State transfer can take awhile, so may take time for backup to take up role
- *How does view server know whether backup is up to date?*
 - View server sends **view-change** message to all
 - Primary **must ack new view** once backup is up-to-date
- View server stays with current view until ack
 - Even if primary has or appears to have failed

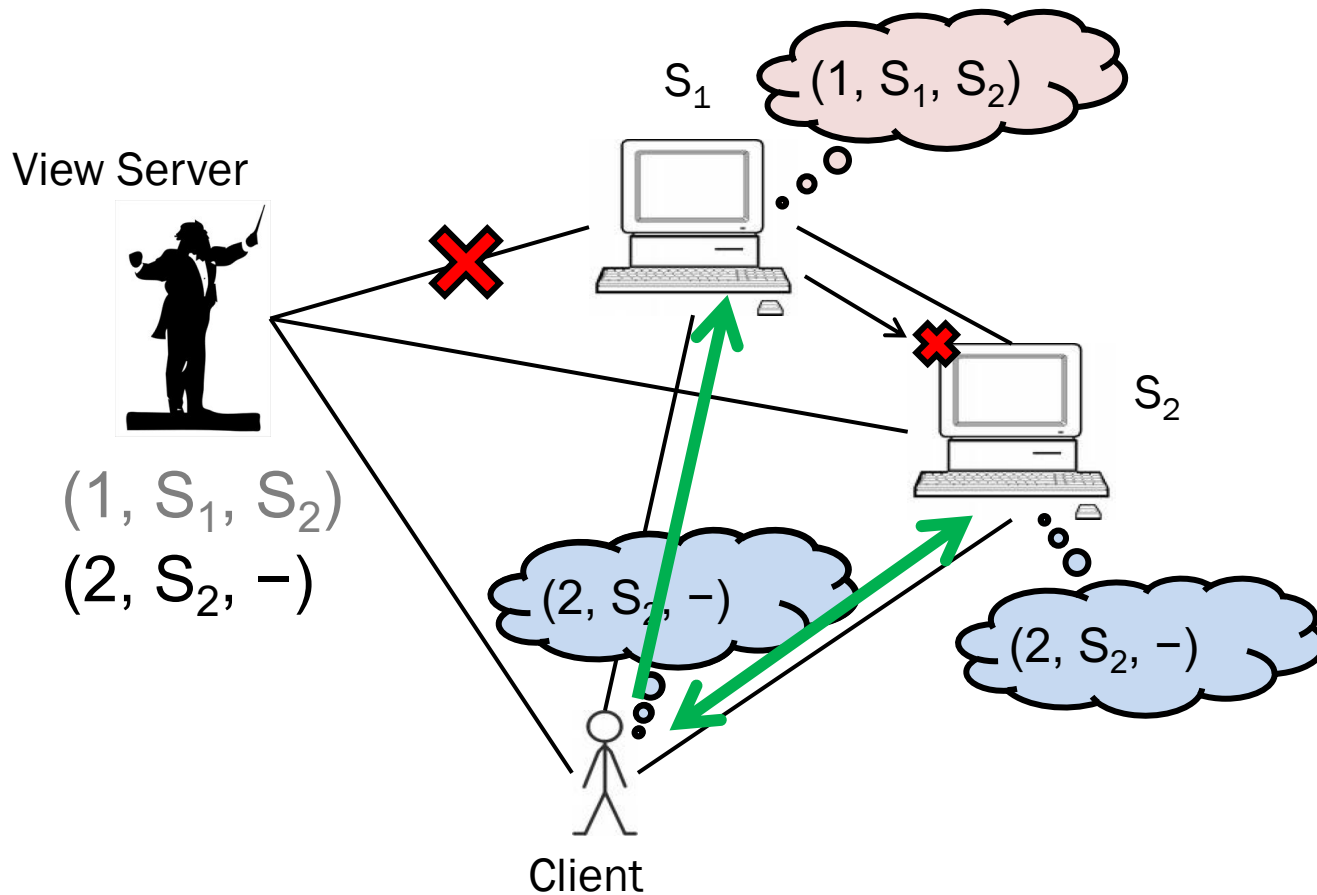
Split Brain



Server S_2 in the old view



Server S_2 in the new view



State transfer via operation log

- *How does a new backup get the current state?*
 - If S_2 is backup in view i but was not in view $i-1$
 - S_2 asks primary to transfer the state
- One alternative: transfer the **entire operation log**

[Simple, but **inefficient** (operation log is long)]

State transfer via snapshot

- Every op must be either **before** or **after** state transfer
 - If op **before** transfer, transfer must **reflect** op
 - If op **after** transfer, primary **forwards the op** to the backup after the state transfer finishes
- If each client has only one RPC outstanding at a time, state = map + result of the last RPC from each client
 - (Had to save this anyway for “at most once” RPC)

Summary of rules

1. View i 's **primary** must have been primary/backup in view $i-1$
2. A **non-backup** must reject forwarded requests
 - Backup accepts forwarded requests only if they are in its idea of the current view
3. A **non-primary** must reject direct client requests
4. Every operation must be **before or after** state transfer

Primary-Backup: Summary

- First step in our goal of making **stateful** replicas **fault-tolerant**
- Allows replicas to provide **continuous service** despite **persistent net and machine failures**
- Finds repeated application in **practical systems (next)**

Question

- Suppose view server V moves from $(1, A, B)$ to $(2, B, C)$, but A comes back and B goes unresponsive
 - What then?
 - Don't know if either of A or C is up-to-date
 - Have to wait for B to come back
- What happens if view service is down/partitioned?
 - View service could use $P-B$, but then...

Plan

1. Introduction to Primary-Backup replication
- 2. Case study: VMWare's fault-tolerant virtual machine**

Scales *et al.*, SIGOPS Operating Systems Review 44(4), Dec. 2010

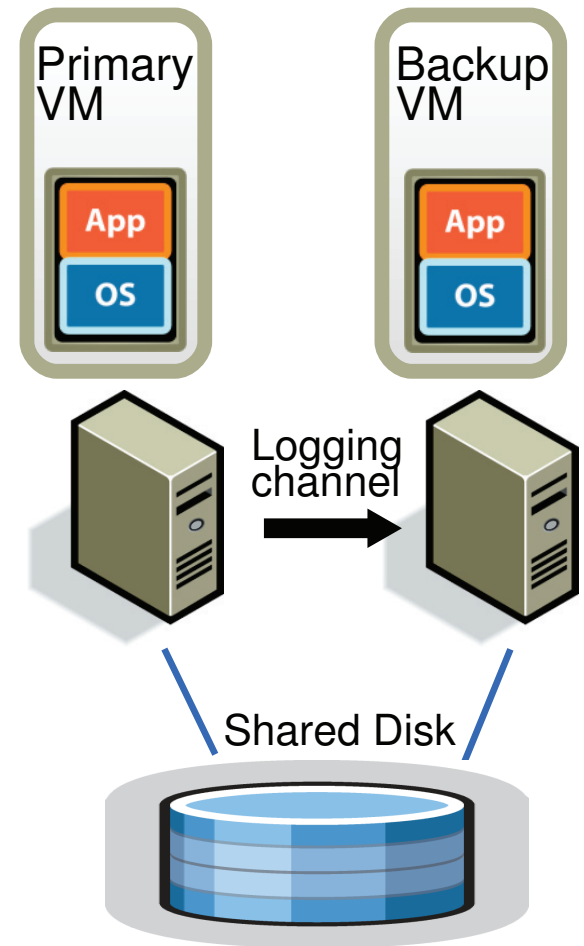
VMware vSphere Fault Tolerance (VM-FT)

- Goals:

1. Replication of the **whole virtual machine**
2. **Completely transparent** to applications and clients
3. **High availability** for any existing software

Overview

- Two virtual machines (*primary, backup*) on different bare metal
- *Logging channel* runs over network
- Fiber channel-attached *shared disk*

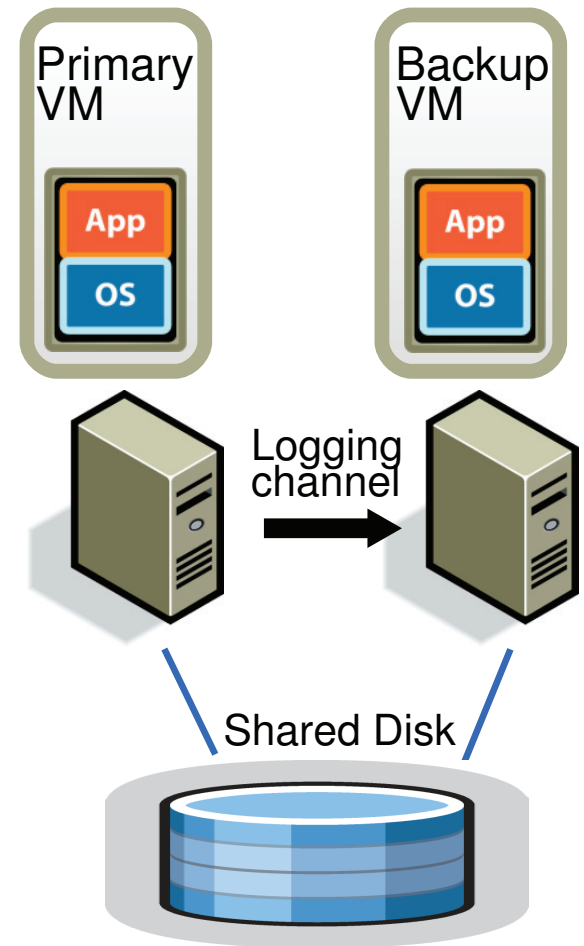


Virtual Machine I/O

- **VM inputs**
 - Incoming network packets
 - Disk reads
 - Keyboard and mouse events
 - Clock timer interrupt events
- **VM outputs**
 - Outgoing network packets
 - Disk writes

Overview

- **Primary** sends **inputs** to backup
- **Backup outputs** dropped
- Primary-backup **heartbeats**
 - If primary fails, backup takes over



VM-FT: Challenges

1. Making the backup an exact replica of primary
2. Making the system behave like a single server
3. Avoiding two primaries (Split Brain)

Log-based VM replication

- **Step 1:** Hypervisor at the primary **logs the causes of non-determinism:**
 1. Log results of **input events**
 - Including current program counter value for each
 2. Log results of **non-deterministic instructions**
 - e.g. log **result** of timestamp counter read (RDTSC)

Log-based VM replication

- **Step 2:** Primary hypervisor **sends log entries to backup hypervisor** over the logging channel
- Backup hypervisor **replays** the log entries
 - **Stops backup VM** at next input event or non-deterministic instruction
 - Delivers **same input** as primary
 - Delivers **same non-deterministic instruction result** as primary

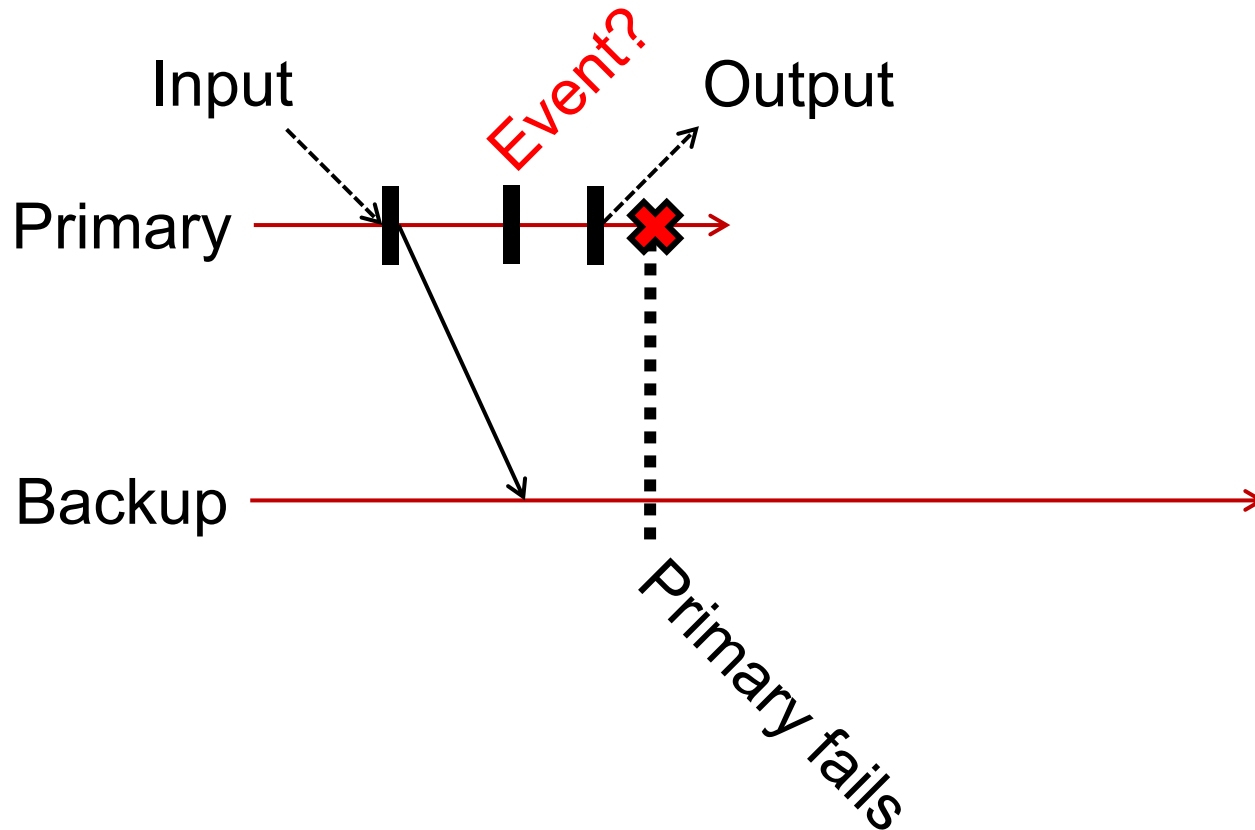
VM-FT Challenges

1. Making the backup an exact replica of primary
2. Making the system behave like a single server
 - FT Protocol
3. Avoiding two primaries (Split Brain)

Primary to backup failover

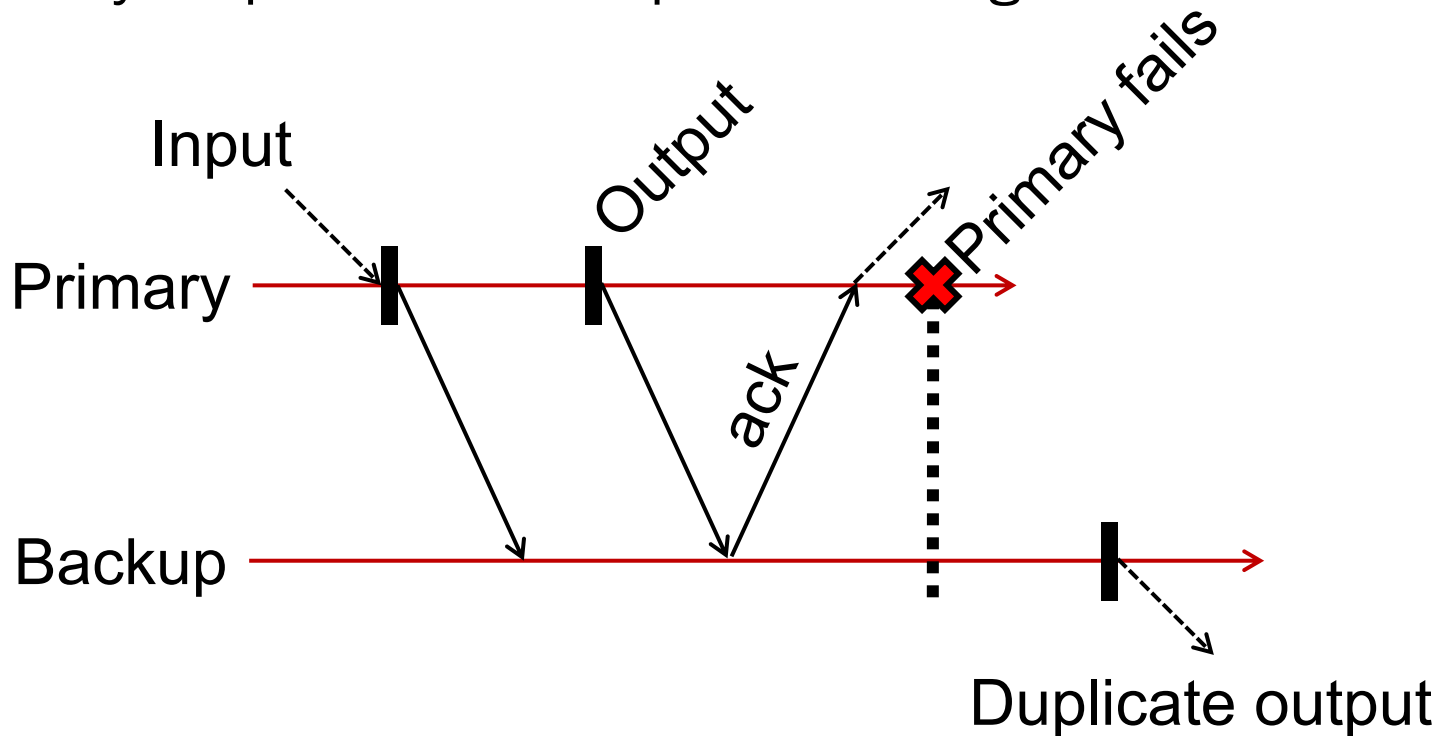
- When backup takes over, non-determinism will make it **execute differently** than primary would have done
 - **This is okay!**
- **Output requirement:** When backup VM takes over, its execution is **consistent** with **outputs** the primary VM has already sent

The problem of inconsistency



FT protocol

- Primary **logs each output** operation
 - **Delays** any output until Backup acknowledges it



Can restart execution at an output event

VM-FT: Challenges

1. Making the backup an exact replica of primary
2. Making the system behave like a single server
3. **Avoiding two primaries (Split Brain)**
 - Logging channel may **break**

Detecting and responding to failures

- Primary and backup each run UDP heartbeats, monitor logging traffic from their peer
- Before “going live” (backup) or finding new backup (primary), execute an **atomic test-and-set** on a variable in shared storage
 - Note the “lease” assumption; on successful CAS assume old primary has “stopped”
 - Why is this guaranteed here?
- If the replica finds variable already set, it **aborts**

VM-FT: Conclusion

- Challenging application of primary-backup replication
- Design for correctness and consistency of replicated VM outputs despite failures
- Performance results show generally **high performance, low logging bandwidth overhead**