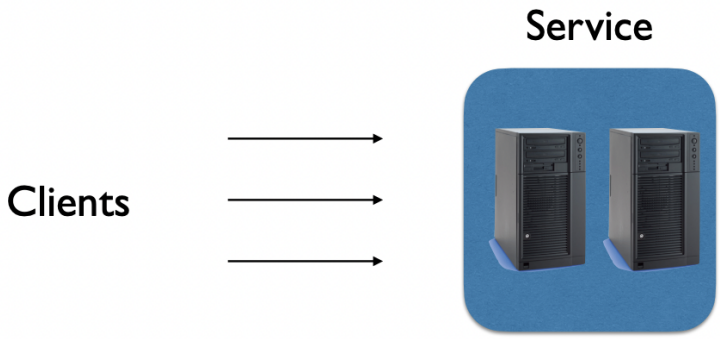
CAS CS 350
Lec 10

1. Fault-Tolerance
    a. Replication
        i. When network goes down, we want it to still work
        ii. End users should not notice any behavior in case of failures
        iii. We can balance the workload and keep latency as low as possible
        iv. The cost of the replication is to keep track of the same data within all replicas, which is challenging (needs to take care of consistency)
        v. Another cost is redundancy
        vi. As a developer, developing a replicated system is very difficult

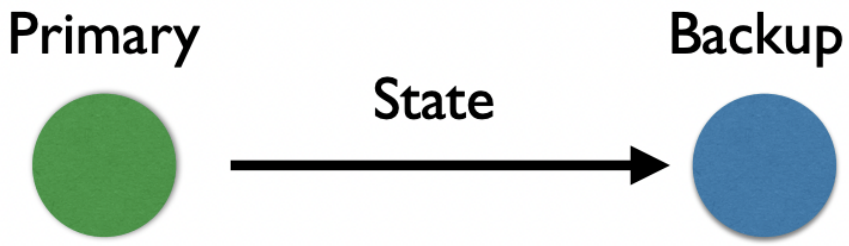


Two computers run your service

        vii.
        viii. Somehow arrange for the second computer to take over from the first if it ever suffers a failure
        ix. Two things → availability (if the first system fails would like the second to take over and pickup immediately while it may take hours to fix the first), correct in the face of replication (lots of difficulties around correctness)
    b. Correctness
        i. Create a copy when we replicate
        ii. Very hard to avoid a window in which they do not match - ensure same all the time
        iii. So very hard to ensure that clients never notice that failure has happened and cut-over occurred
        iv. Total transparency is very difficult - more often than weaken interface to clients to make it easier
    c. Types of failures interested in backup

i. At some point, the machine goes down and goes up again (in general, we assume independent failures) → in real life, failures are not independent

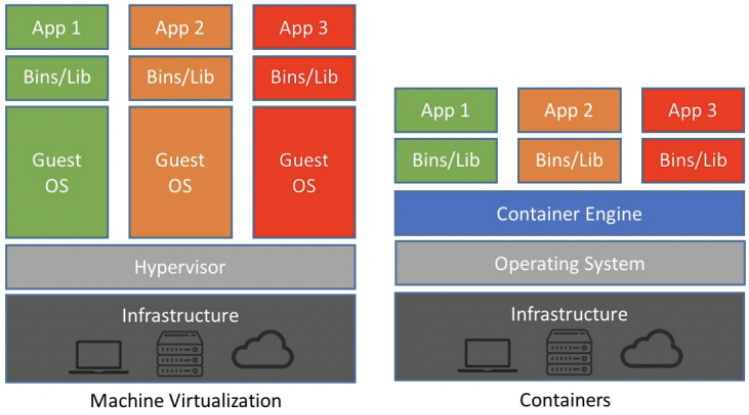
d. What state is replicated?
   i. State transfer → send the state to the backup as a whole (simplest to implement)



   ii.
   iii. Replicated state machine → send the individual changes and instructions to update the state (beneficial when instructions are deterministic)
   iv. Which one is better → it depends
      1. If the state is huge → want to avoid sending the whole data (since it takes huge time)
      2. Replicated state machine → cannot easily handle non-deterministic states

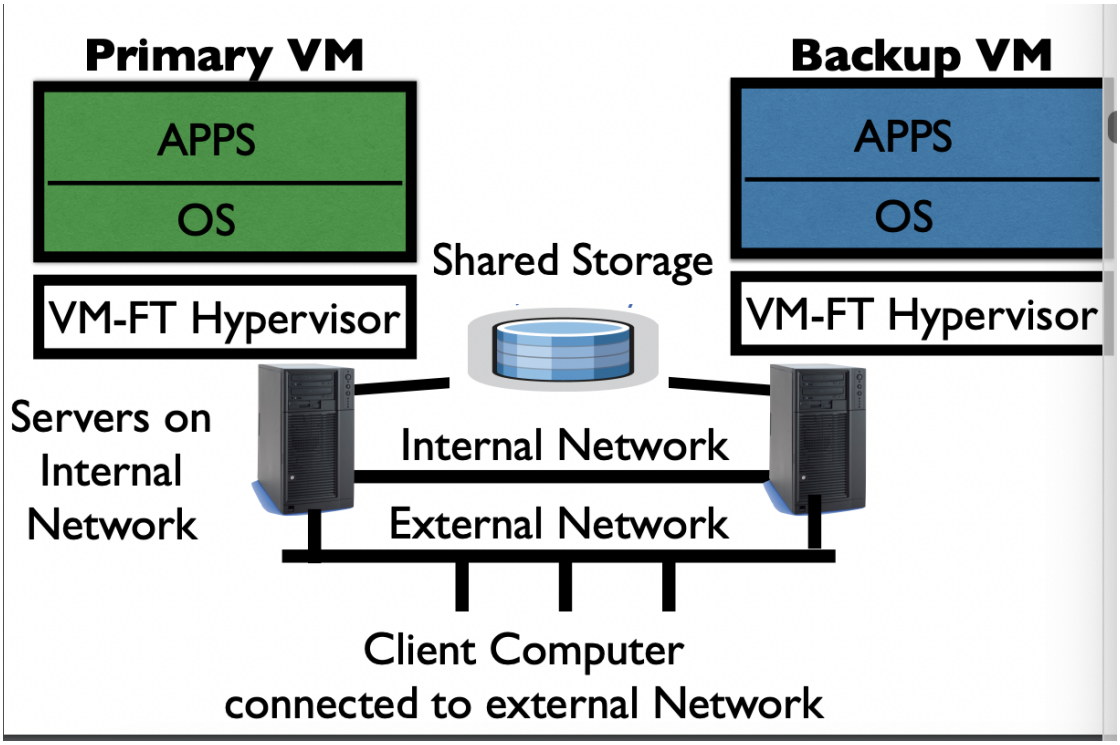
2. PB replication approaches
   a. State transfer systems
      i. Primary execute and updates its state and periodically transfers to backups
3. Virtual machine - Fault Tolerant (VM-FT)
   a. RSM-based approach
   b. Super ambitious
      i. Take any software (OS and app) and make it fault tolerant
      ii. Like a magic wand
   c. Costs are high but very cool given what it's goals are

4. VMs vs containers



   a.
   b. VM is a software that runs on top of physical machine and emulates particular hardware (computer system of abstraction)
   c. Software manages VMs (Hypervisor is supervisor of virtual machine)

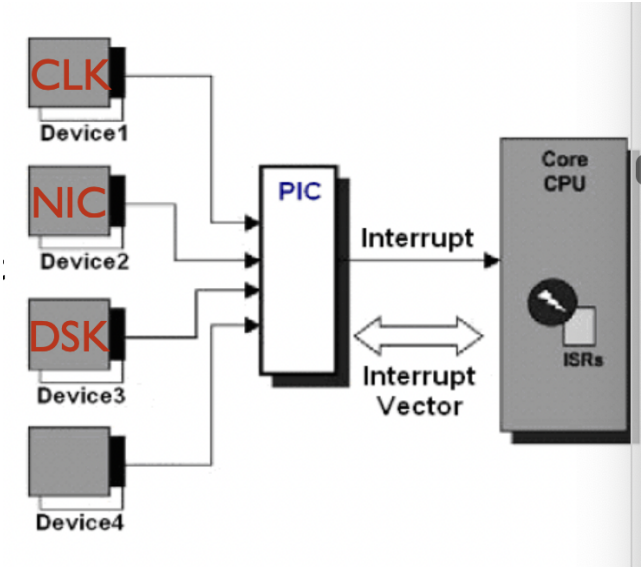
5. Overview

   a. 

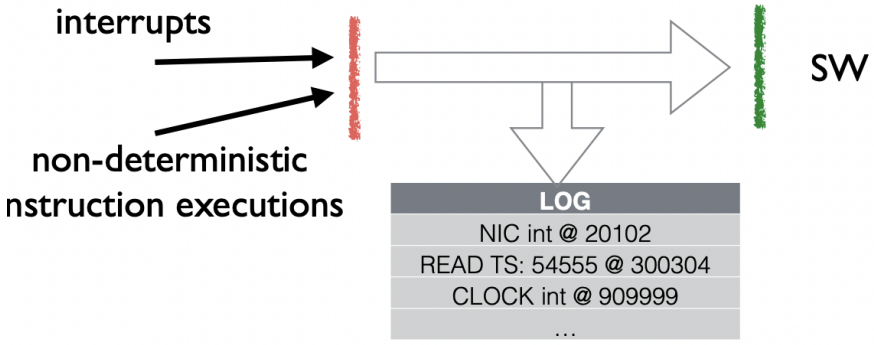
   b. Two VMs (two servers or multiple servers) → main and backup and have clients talking to them
   c. Primary and backup are connected (these two share a disk that run on different nodes)
   d. Both VMs run VM-FT Hypervisor (implement primary backup protocol)
   e. The goal is to make the whole stack fault tolerant

f. Backup in “lock step” with primary
    i. Primary sends all input to backup through a login channel
    ii. Output of backup is dropped (backup doesn’t interact with the clients until the primary fails)
    iii. Heart beats between primary and backup
        1. If primary fails start backup executing

6. Inputs
    a. Clock interrupts/ Network interrupt/ Disk interrupts

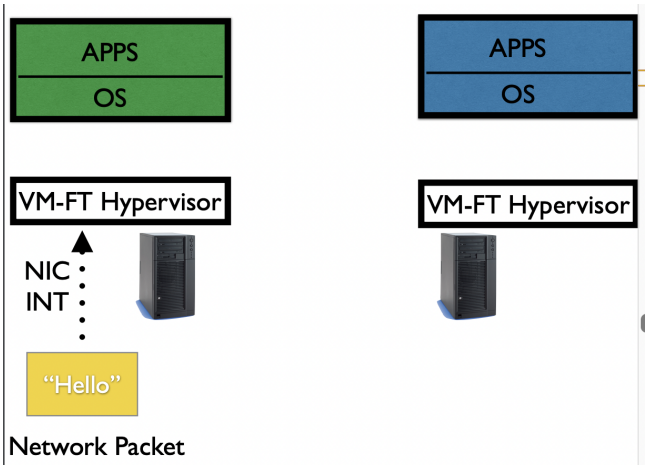


    b.
    c. Inputs come from the hardware of the primary
7. Logging

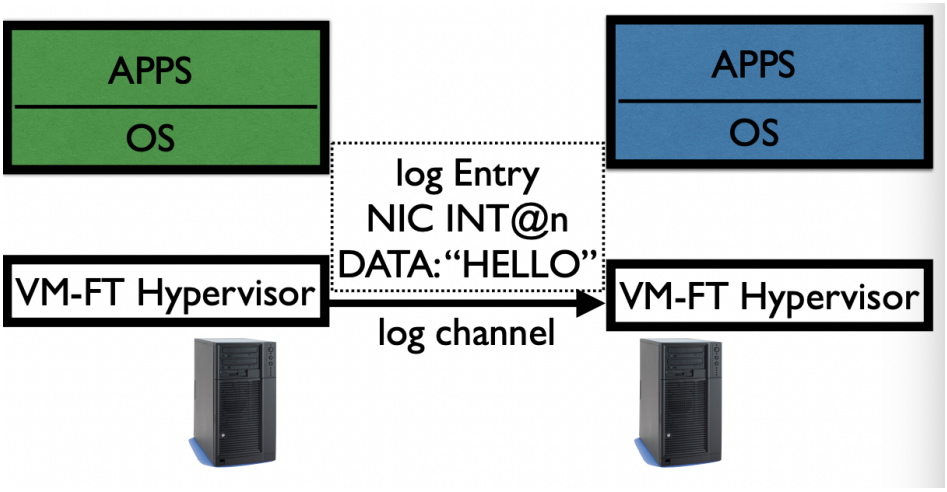


    a.
    b. Primary stores all hardware events and sends it to the backup using logging channel (primary does not send the copy, just sends the commands) → backup executes the events
    c. Log all hardware events into a log
    d. Hypervisor takes care of the non-deterministic instruction executions (it makes sure that the value of the variable will be moved to the backup so that backup gets the same value)

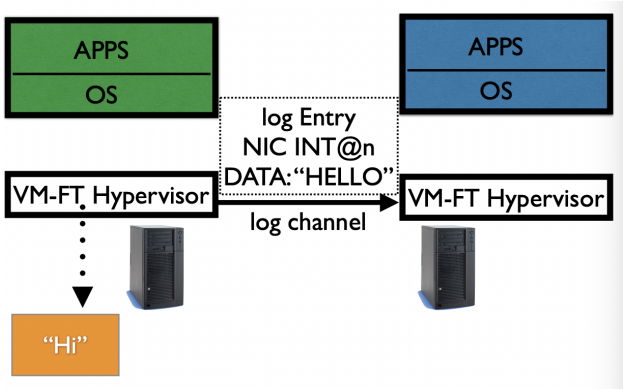
8. Example
    a. Client sends message to primary



    b. Primary sends the data to the backup through the logging channel

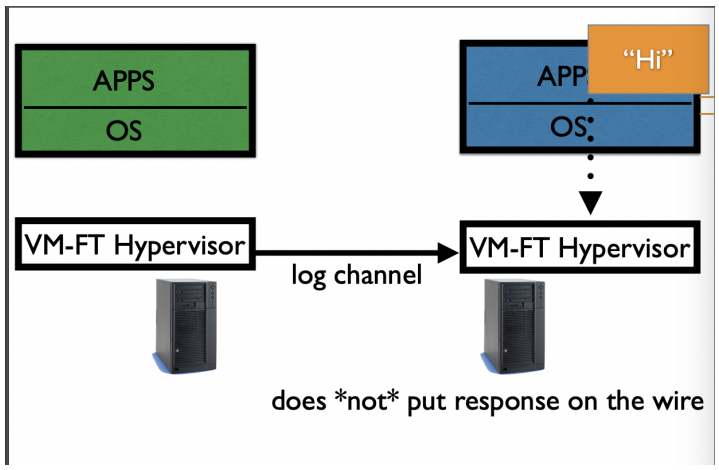


    c. Operation does whatever it has to and sends it to app
    d. Hypervisor responds back to the user on the wire

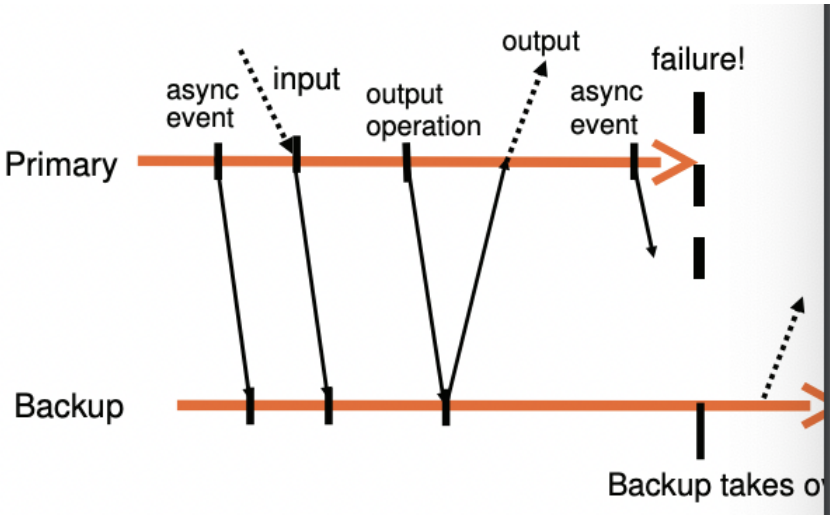


    e. In the meantime, the backup that received the data, and follows the same request as primary (but does not send the response on the wire) → these steps occur asynchronously

i. Ignores local clock gets from primary
ii. Primary and backup same input and execute same instruction and end up with the same state

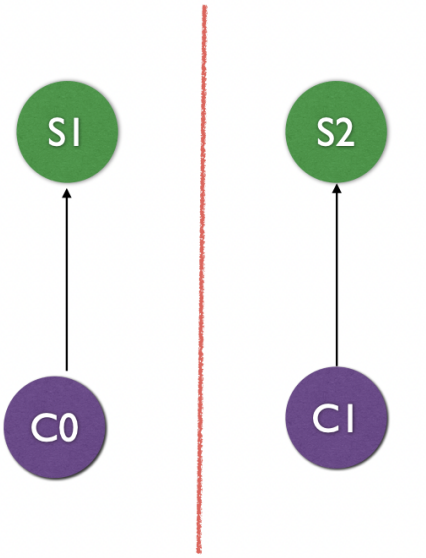


9. Challenges
    a. Making it look like a single reliable server from the outside (the end user should not know about the system)
    b. How to avoid two primaries?
    c. How to make sure backup keeps up with primary
    d. If the backup is slow and cannot keep up with the primary
10. Challenge 1



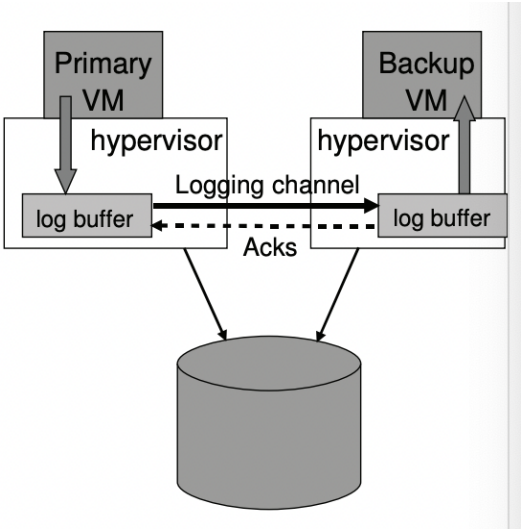
    a.
    b. Making it look like a single reliable server from the outside
    c. Primary delays output until backup acks
        i. Log each output op
        ii. Only send after backup acked receiving output op
    d. Perfect opt: primary continues to execute after output
    e. Offers output till backup ack

11. Challenge 2



a.
b. How to avoid two primaries?
c. Partition: split brain
    i. Network partition both primary and back believe they are the primary
d. VM-FT's solution to the Split Brain problem
    i. Hard problem with only unreliable network
    ii. Assume shared disk
        1. Backup replays through last log entry
        2. Backup atomically(can only change the flag that exists in the shared storage one at a time) test-and-set "live" variable on disk
            a. If set, primary is alive. Commit suicide
            b. If not set, primary are dead. Become primary
            c. If primary, create new backup from checkpoint
        3. If this node dies, and if one the VMs fail, you may end up with two primaries (system might become unavailable)

12. Challenge 3: How do we deal with as low backup?

a. 

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