LECTURE 7

Consistency and Replication

Replication and why is it hard

- Data needs to be replicated for reliability or to improve performance
  - If a replica crashes, possible to continue to have access.
  - Performance improves by accessing nearer data stores
  - Aids scaling
- However, a key challenge is ensuring the consistency of data across replicas.
  - Hard to realize in large distributed systems.

`

Handling Failures

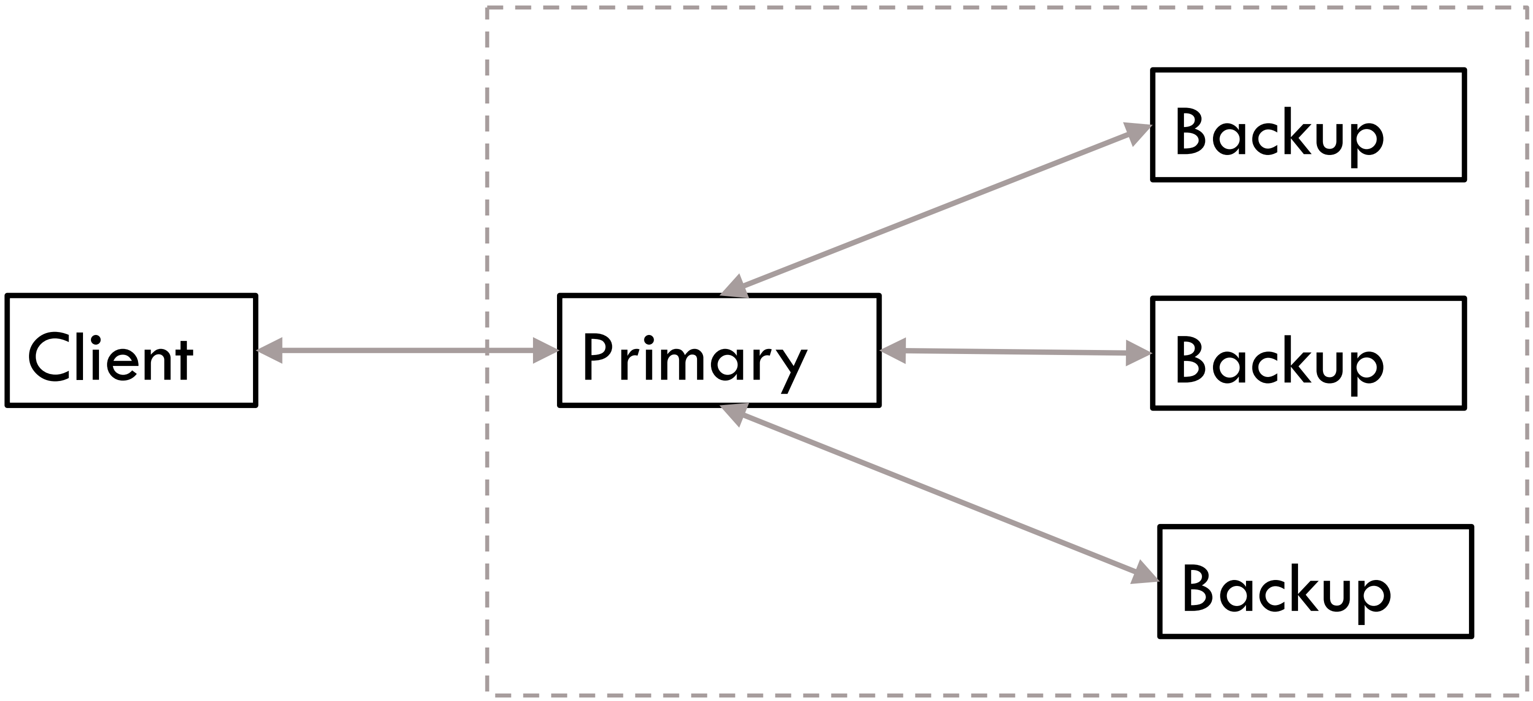
- Logical clocks enable all replicas to execute updates in same order
  - But, cannot handle failure of any replica!

- To deal with crash failures we saw:
  - Chandy-Lamport snapshot for coordinated checkpoint
  - Log sources of non-determinism between checkpoints

Types of failures

- Crash failures
  - Can resume with previously saved state
- Fail stop
  - All state is lost upon failure
  - Need to replicate state

Primary Backup Replication



Primary Backup Replication

- How to handle primary failure?
  - Promote one of the backups as the primary
- How to handle backup failure?
  - Add another machine as a backup

Primary Backup Sync

- When should primary sync up with backups?
- What should be transferred when syncing?

Example: MapReduce Master

Primary & Backups in sync →

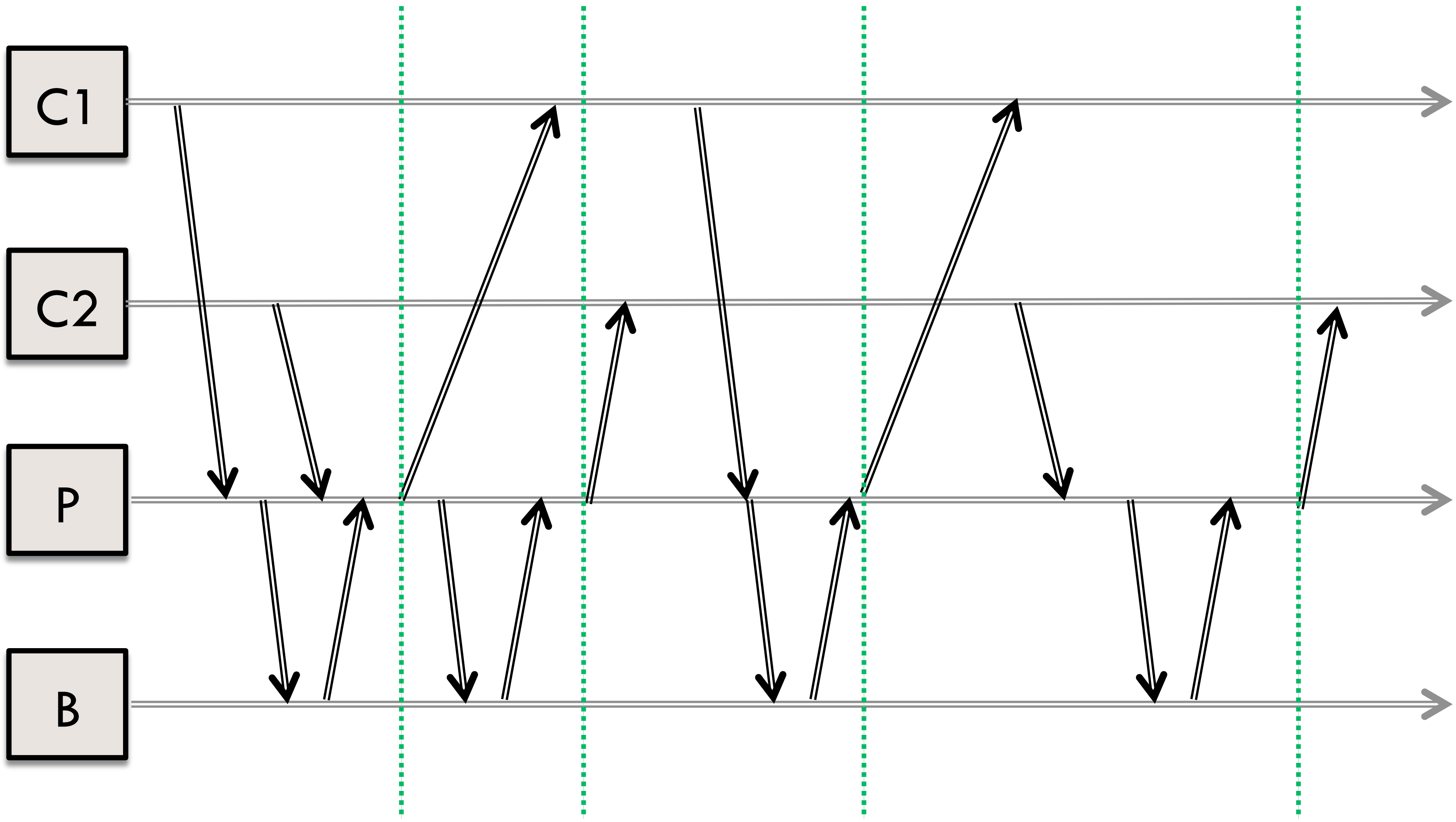
When to sync again? →

```
RegisterServer() {
  while (1) {
    receive msg and parse addr
    pick task to assign
    mark task as assigned
    respond with task assignment
  }
}
```

Takeaways

- Cannot tolerate primary failure after update to its state is externally visible
- Corollary: Okay for primary to be out of sync with backup until change is externally visible
  - External consistency
- Implications for *when* primary should sync with backups?

Primary-Backup Sync



Reads vs. Updates

- For operations that do not update state, primary need not consult backups
  - When is this true?
- If backup is externally consistent with primary
  - → If backup takes over as primary, it will generate identical response as primary may have

What to transfer in sync?

- Snapshot of primary's state
  - Slow
  - When is this necessary?
  - Necessary when bootstrapping new backup

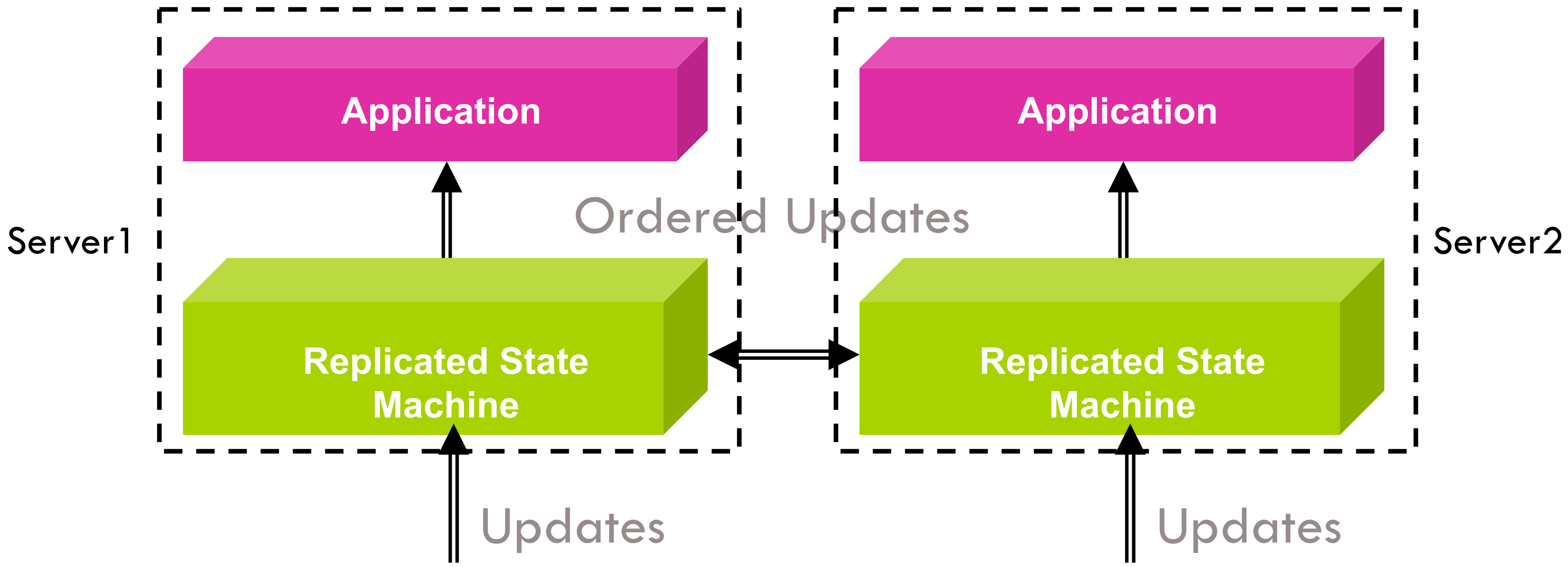
- Every operation
  - Why is this okay?
  - Leverage determinism of state machine

Service Development

- Getting coordination right between primary and backups is tricky
  - Easy to mess up

- Must make replication transparent to developer

RSM with Logical Clocks

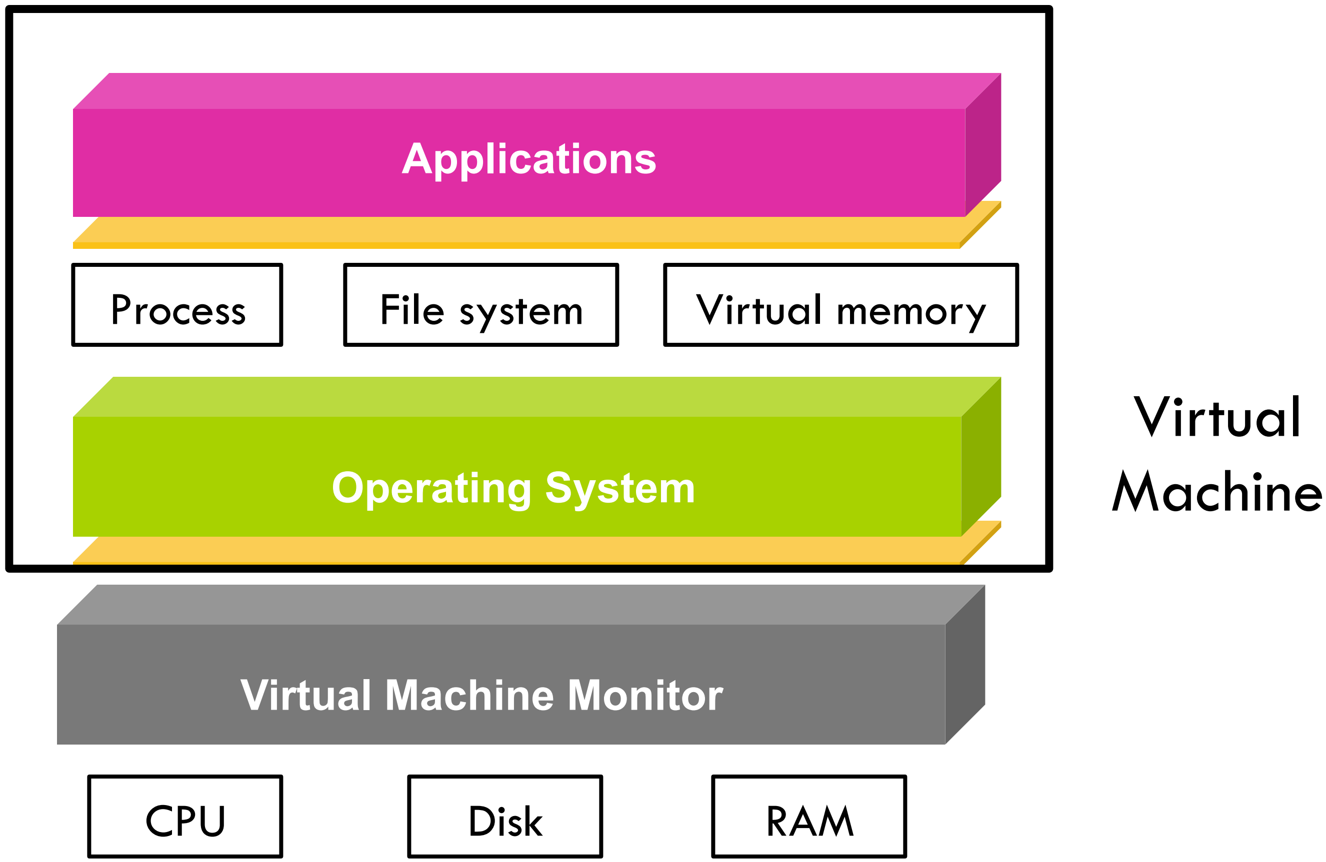


Transparent Primary Backup

- Application relies on library to keep primary and backups in sync
  - Receive message from client
  - Sync with backups before sending response to client

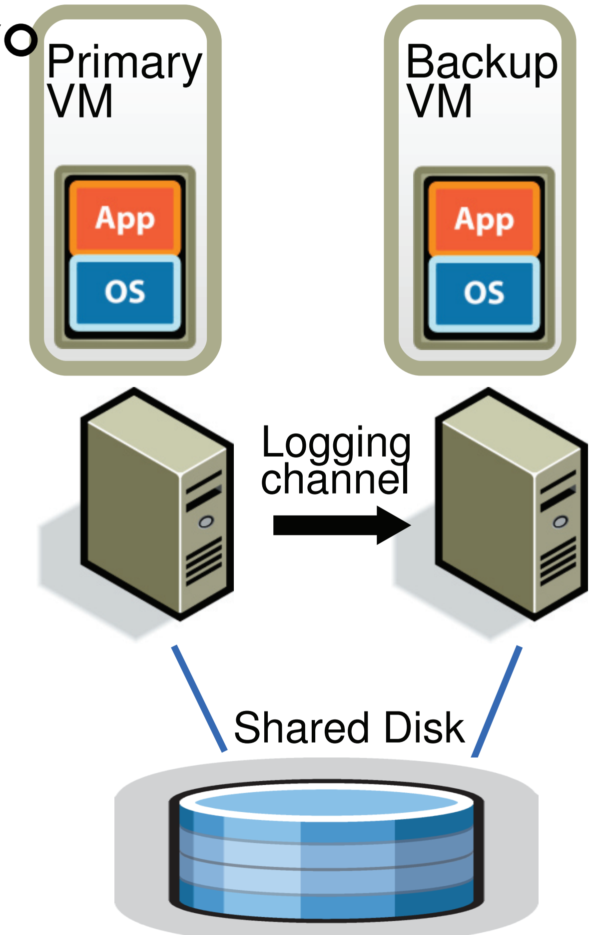
- Will this solution work?
- Does not capture non-determinism in execution

Virtual Machines



VMM-based Primary Backup

- Primary and backup execute on two virtual machines



- Primary logs inputs and outputs
- Backup applies inputs from log
- Primary waits for backup output

- Primary-backup monitor each other
  - If primary fails, backup takes over

Log-based VM replication

- VMM at primary logs external inputs and causes of non-determinism
- Example: Log results of non-deterministic instructions
  - *e.g.*, timestamp counter read (RDTSC)
- Random number generation?

Log-based VM replication

- VMM at primary sends log entries to VMM at backup over the logging channel

- Backup hypervisor replays log entries
  - Stops backup VM at next input event or non-deterministic instruction
  - Delivers same input as primary
  - Delivers same result to non-deterministic instruction as primary

Virtual Machine I/O

- VM inputs
  - Incoming network packets
  - Disk reads
  - Keyboard and mouse events
  - Clock timer interrupt events
  - Results of non-deterministic instructions
- VM outputs
  - Outgoing network packets
  - Disk writes
- Why log outputs?

Example Scenario

Primary	Backup

Primary

- Write input to log
- Apply input
- Produce output
- Fail!

Backup

- Take over as primary
- Read from log and apply input
- Timer interrupt fires
- Produce output

Backup does not know timing of output relative to timer interrupt

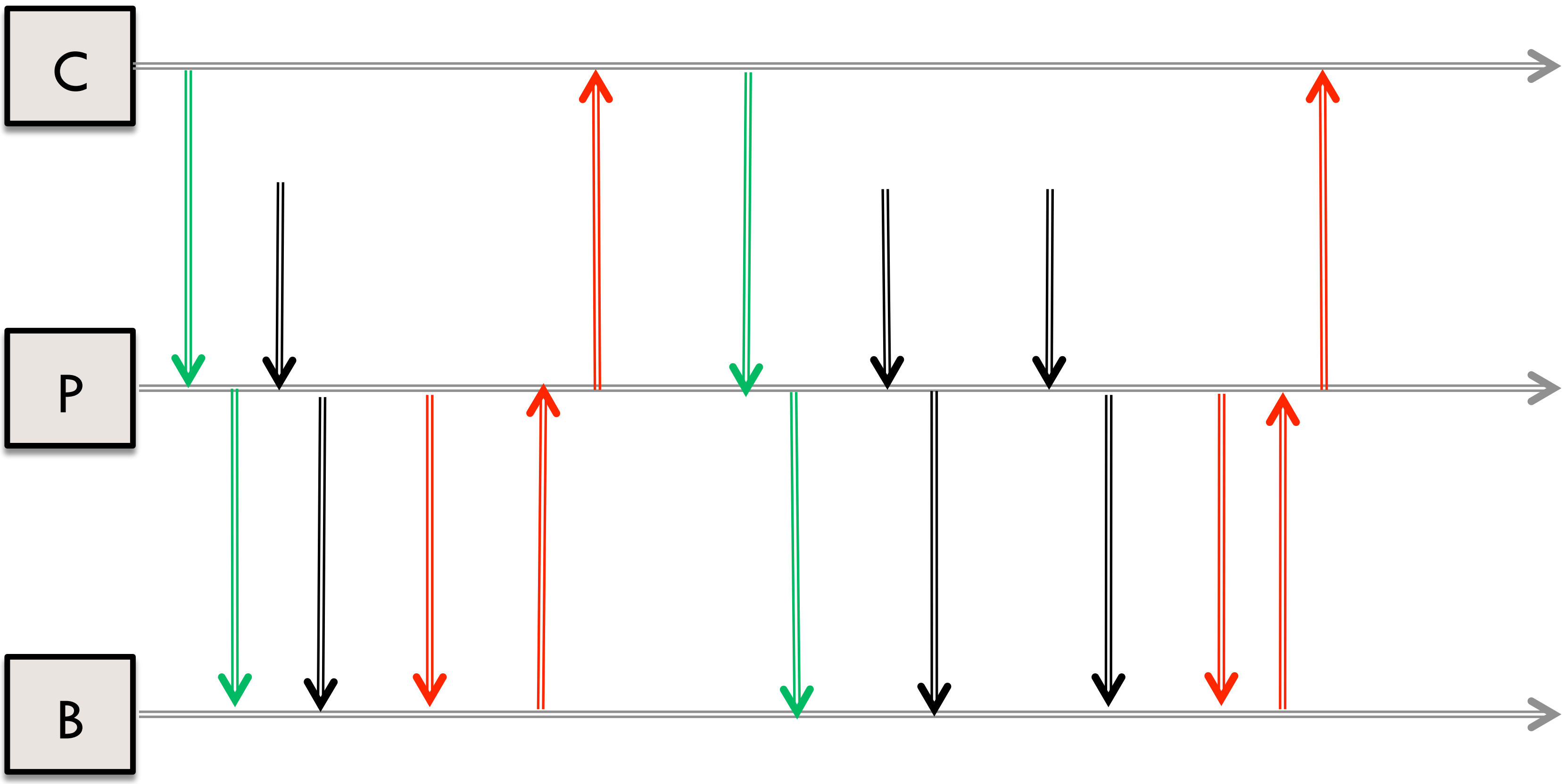
Execution of interrupt handler may affect output

Optimizing Performance

- Primary must buffer output until ACK from backup
  - Why?
  - Slow from client perspective!
    - If replicas are not in close proximity.

- How to optimize performance?
- Pipeline sync with backup and execution at primary

VMM-based Replication



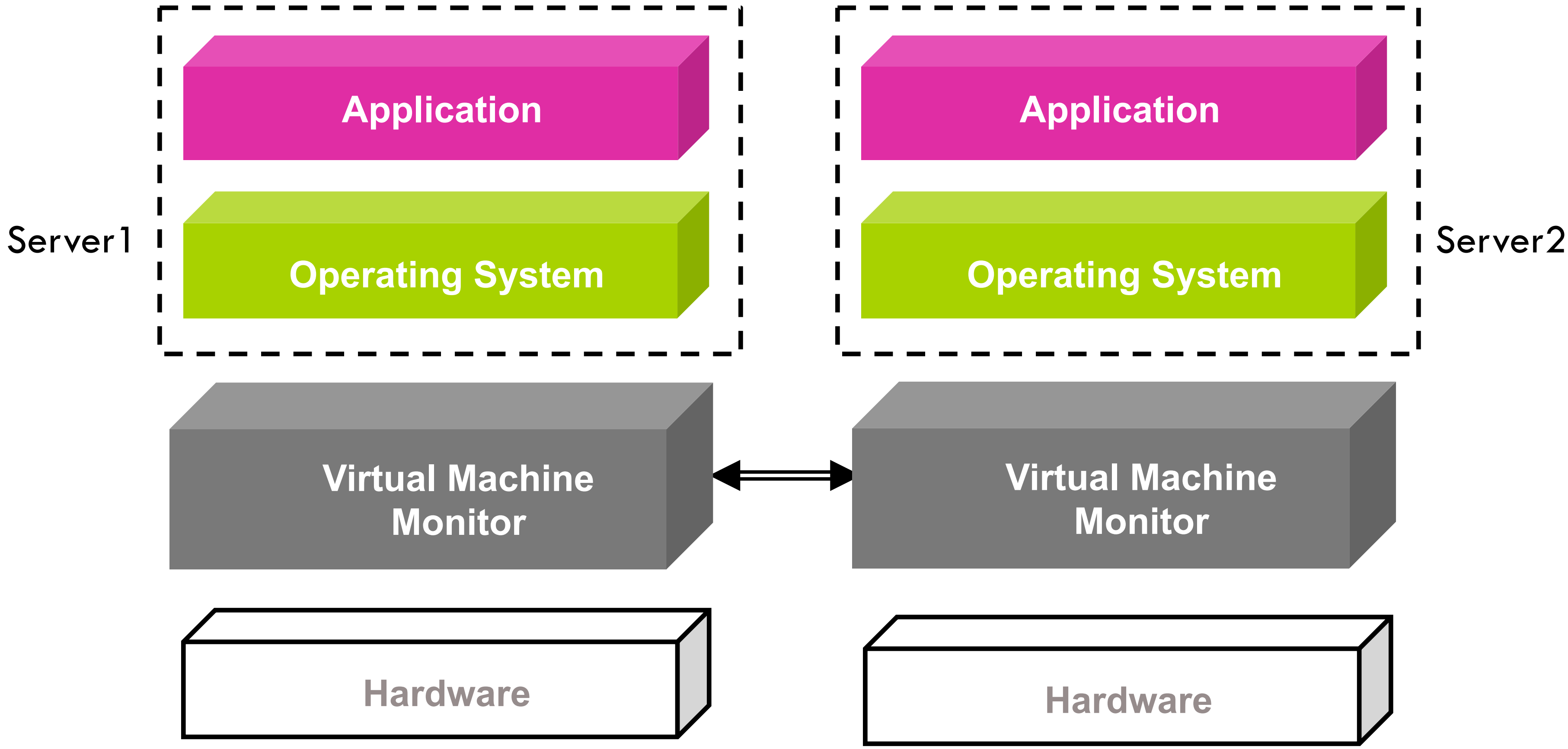
Primary Backup Replication

- Promote one of the backups if primary fails
- Replace any failed backup

- When should primary sync with backups?
  - Before making state change externally visible
  - Primary and backups must be externally consistent

- What to sync?
  - Entire state when bootstrapping new backup
  - Thereafter, all sources of non-determinism

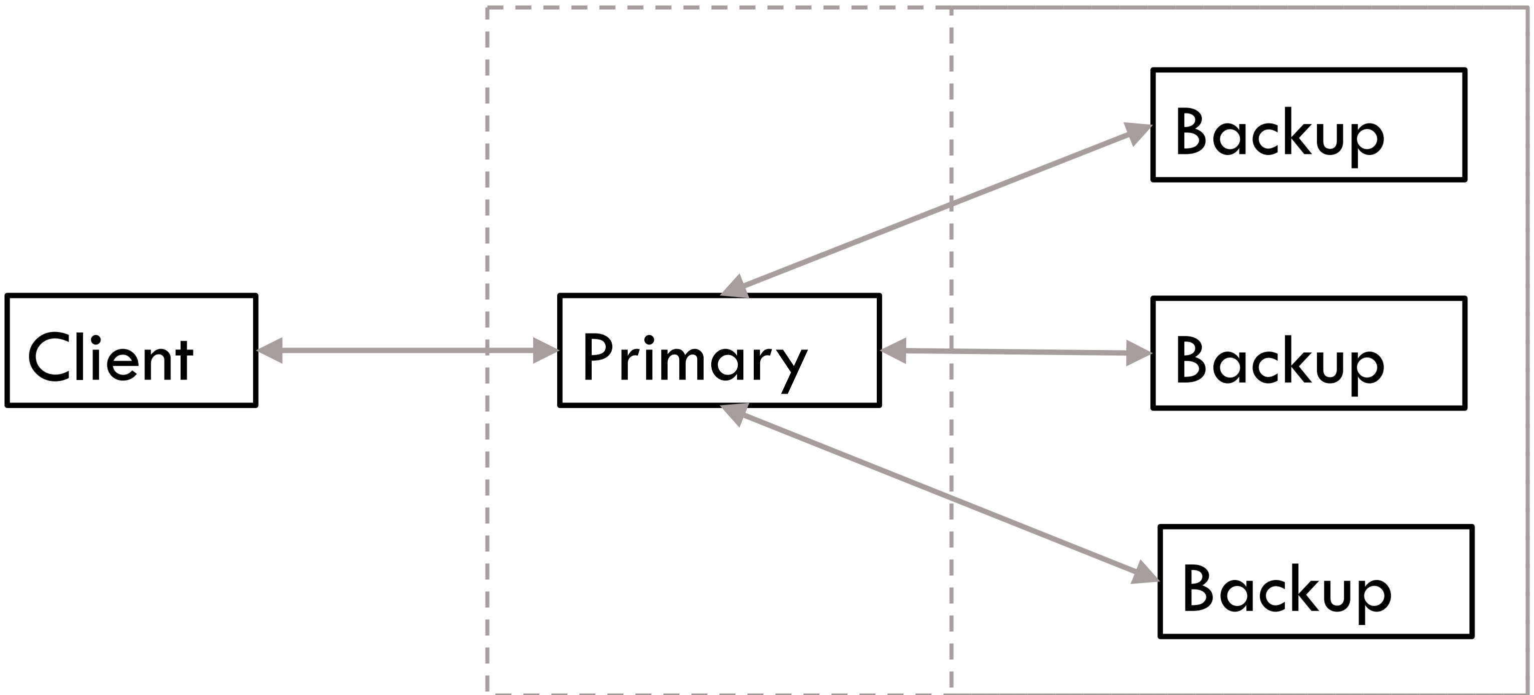
RSM with Primary Backup



Client perspective

- What does a worker need to know in order to register itself with replicated master?
- Needs to know which machine is primary

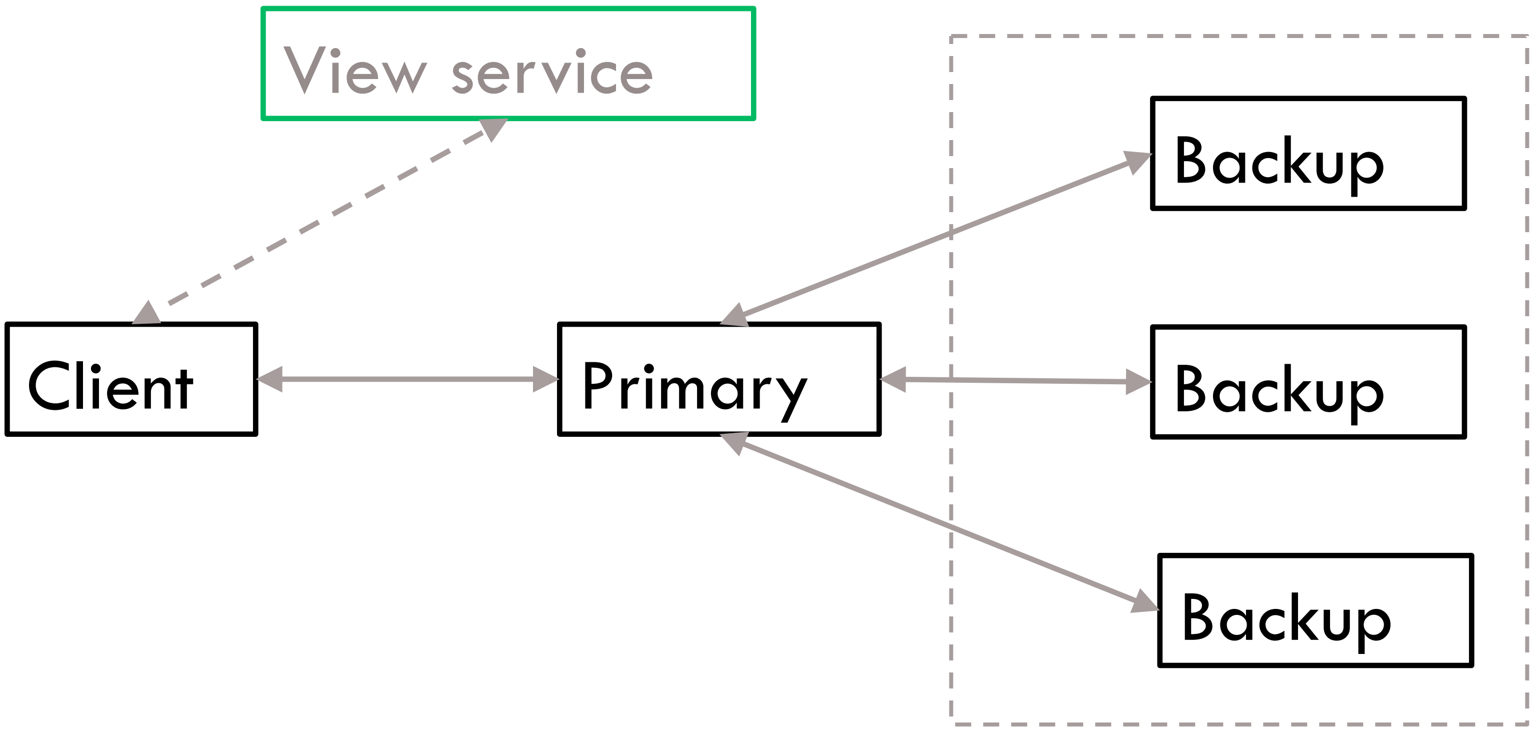
Primary Backup Replication



Client perspective

- What does a worker need to know in order to register itself with replicated master?
- Needs to know which machine is primary
- Can primary be hard coded into client code?
- No, primary gets replaced when it fails
- How does client discover current primary?

Primary Backup Replication



View service

- Maintains current membership of primary-backup service (called view)
  - View number, primary, backup

- When does view service change view?
- When primary or any backup fails
- Periodically exchange heartbeat messages to detect failures

- What if view service is down or not reachable?

Transitioning between views

- How to ensure new primary is up-to-date?
  - Only promote a previous backup
  - This is why view service needs to pick backups

- How does view service know if a backup is up-to-date?
- Two scenarios for ill-timed primary failure:
  - Primary applies operation but fails before syncing with backup
  - Primary fails before new backup is initialized

Transitioning between views

- View service broadcasts view change to all
- Primary must ACK new view once backup is up-to-date
- Two implications:
  - Liveness detection timeout > State transfer time
  - Cannot change view if primary fails during sync

View service

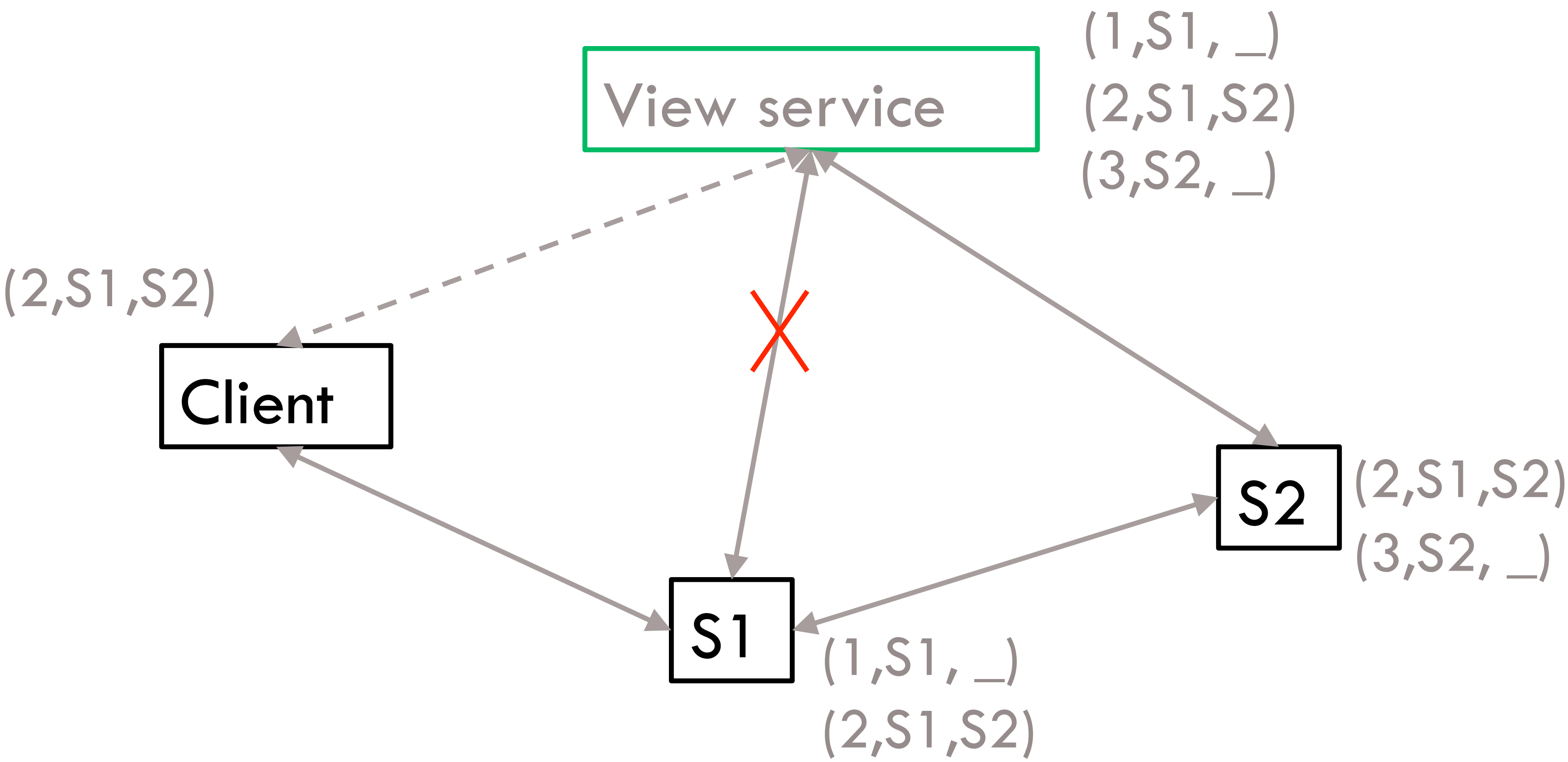
- View change has three steps:
  - View service announces new view
  - Primary syncs with new backup if there is one
  - Primary acknowledges new view

- Stuck if primary fails in the midst of this process

Scalability of View service

- Does every client need to contact view service before any operation?
- Clients can cache view across operations
- When to invalidate cached view?
- Client invalidates cache when no response or negative response from primary

Split Brain



(1,S1, _)
(2,S1,S2)
(3,S2, _)
View service
(2,S1,S2)
Client
S2
(2,S1,S2)
(3,S2, _)
S1
(1,S1, _)
(2,S1,S2)

Avoiding Split Brain

- Primary must forward all operations to backups
  - Goal: Get ACKs from backups that they too recognize primary
- Why can't backups be mistaken about who is primary?
  - Only a backup can be promoted as primary

View service

- Valid sequence of views:
  - (1, S1, _) → (2, S1, S2) → (3, S1, S3) → (4, S3, S4) → (5, S4, _)
- Examples of invalid transitions between views?
  - (1, S1, S2) → (2, S3, S4)
  - (1, S1, S2) → (2, _, S2)
  - (1, S1, _) → (2, S2, S1)

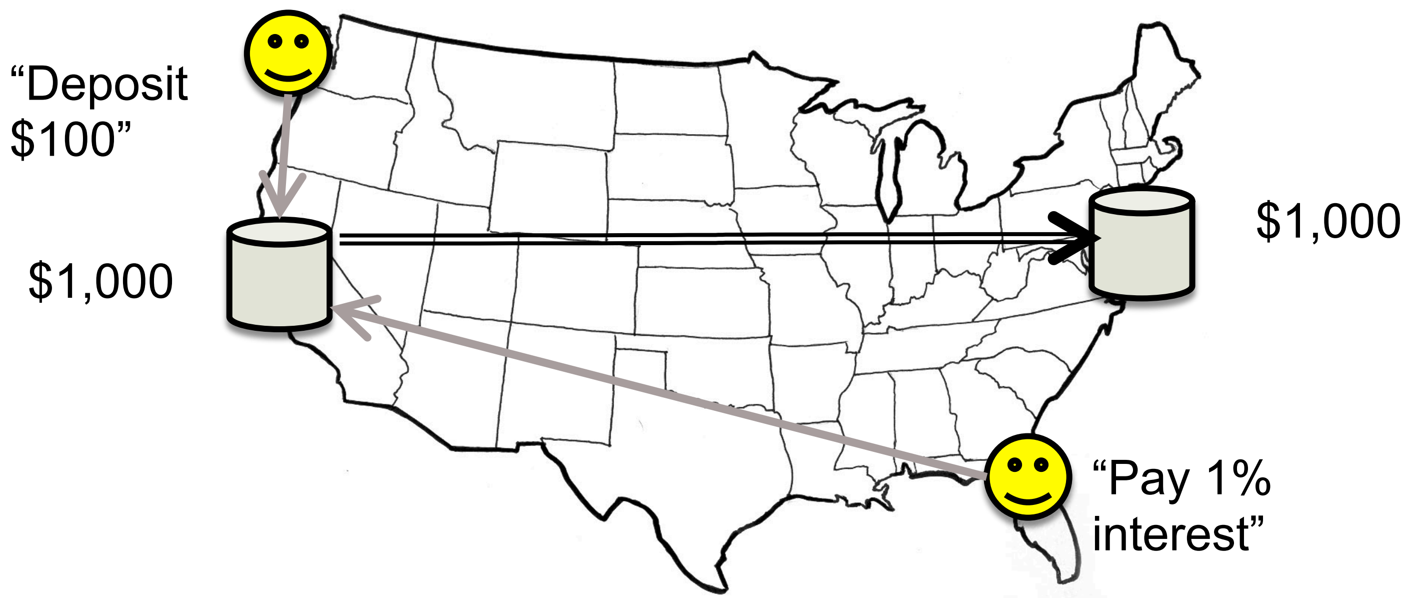
Summary of view service

- Monitors primary and backups to detect when to change view
  - Can change only after primary has ACKed view
  - Primary ACKs only after syncing with backups

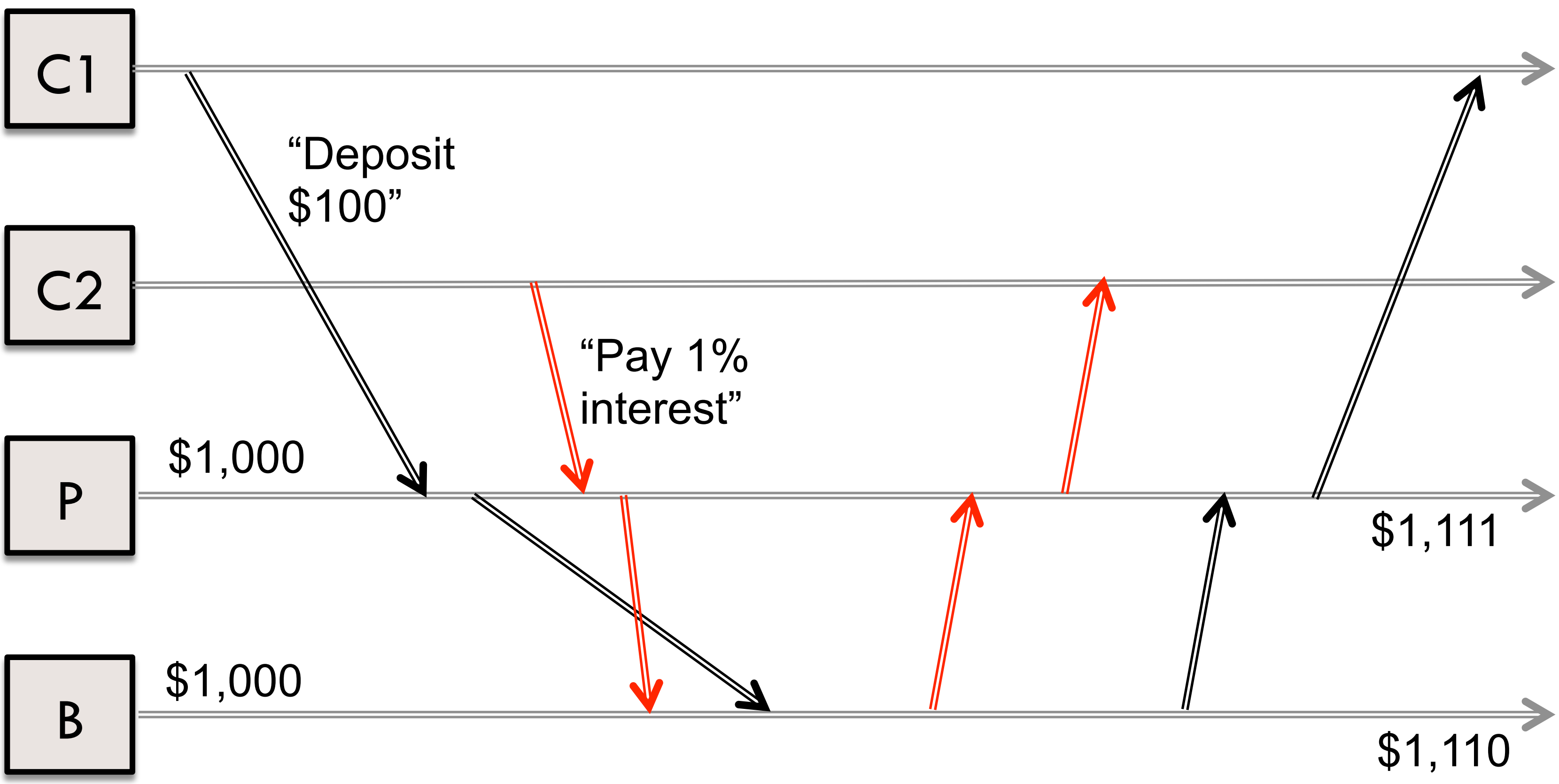
- Clients cache view for scalability
- To address split brain, primary must check with backup before serving client

Replicating Bank Database

- One copy in SF (primary), one in NY (backup)



Primary-Backup Sync



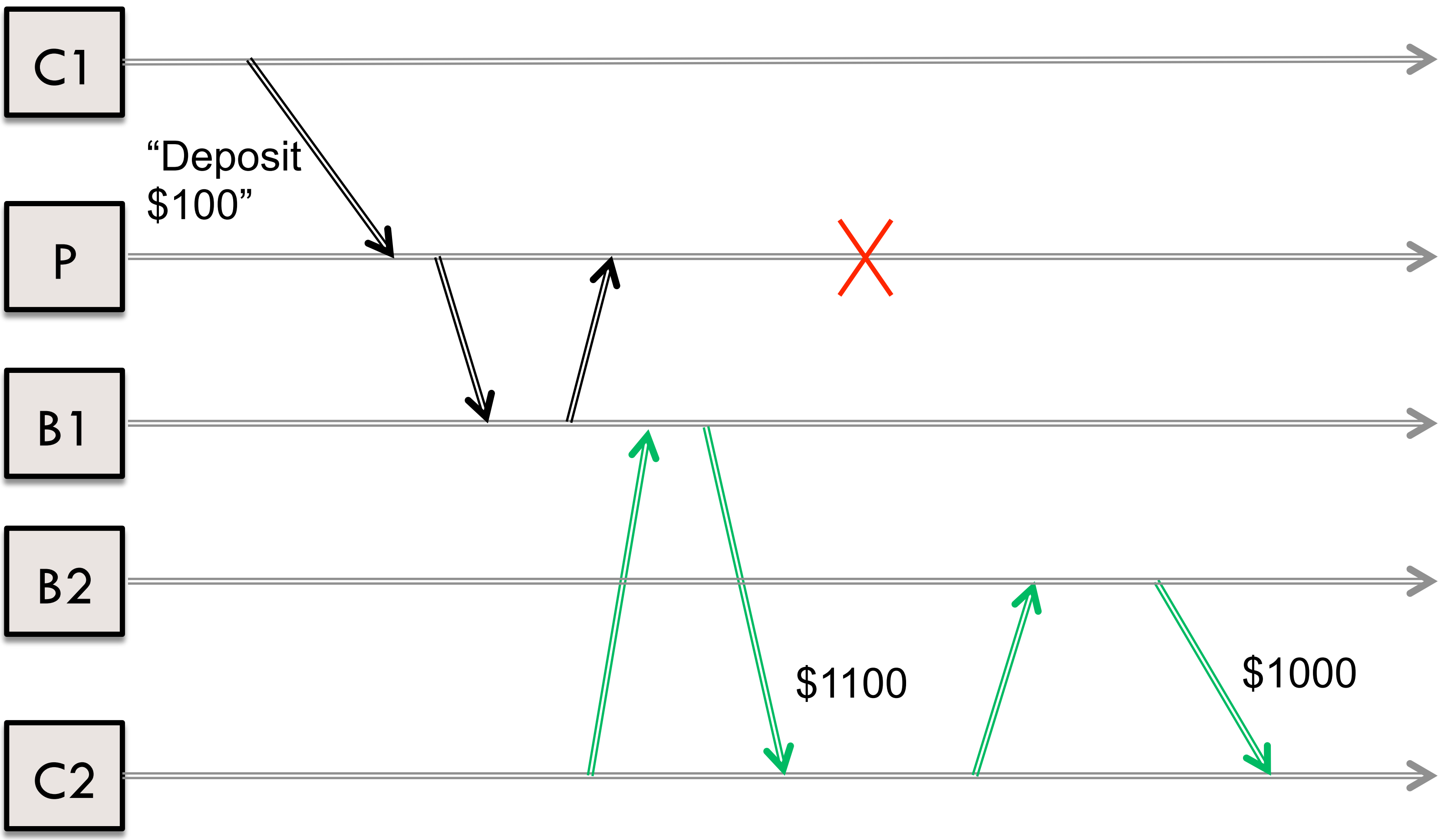
Ordering of Updates

- All updates must be applied in the same order at all replicas
- External view: Total ordering of writes
- Primary effectively serializes all writes
  - Order of events made known to replicas

Serving Reads

- Can backups serve reads?
  - Assume no split brain
- What if primary's state is ahead of backup?
  - Updates to primary not yet externally visible
  - Effect of read equivalent to if primary fails at this point
- What if backup's state is ahead of primary?
  - Different backups may not be in sync
  - Primary may get replaced before it applies update

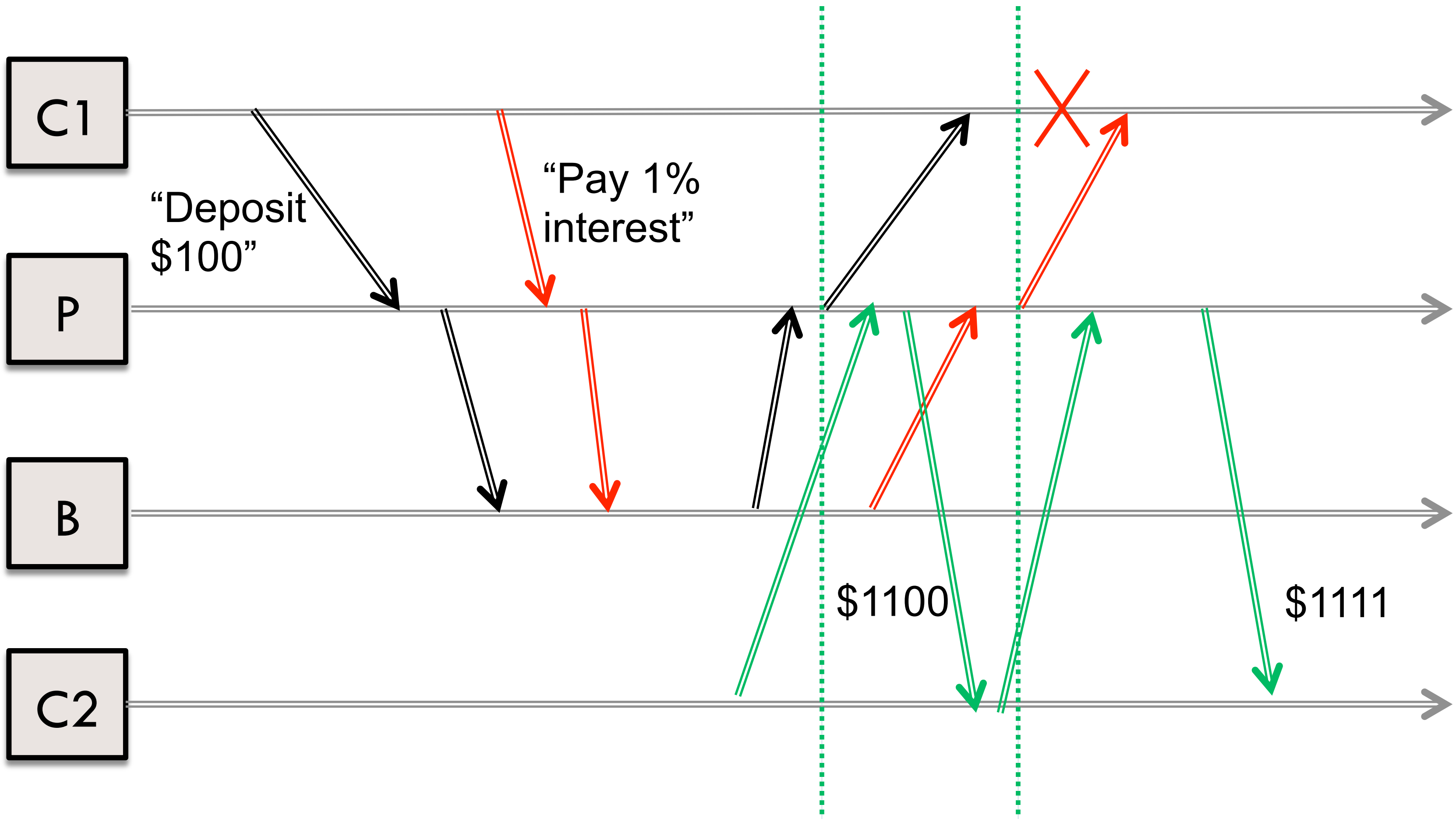
Reads: Primary vs. Backup



Desired Properties

- All writes are totally ordered
- Once read returns particular value, all later reads should return that value or value of later write
- Once a write completes, all later reads should return value of that write or value of later write

Reads relative to Writes



Consistency Spectrum

Eventual — Read-after-write — Causal — Sequential — Linearizability

Consistency →

Ease of programming →

- Consistency: What are the properties of externally visible effects?

Linearizability

- Total ordering of writes
- Read returns last completed write

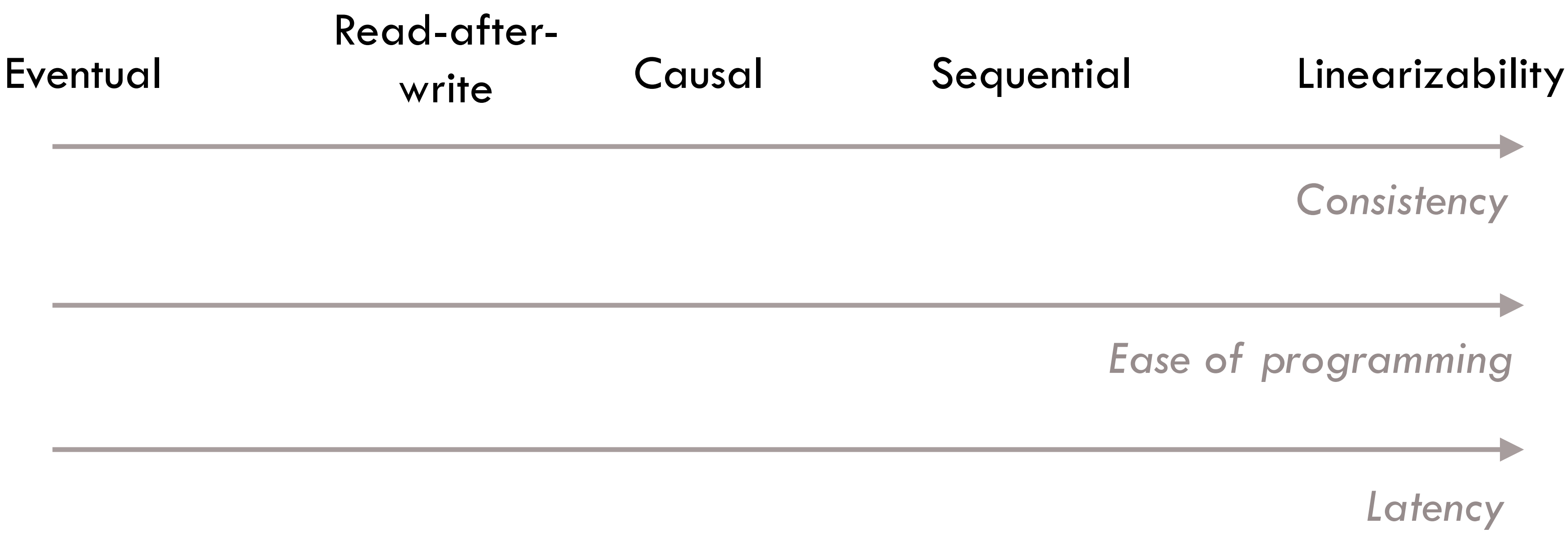
- Single copy semantics
  - Externally visible effects of writes and reads are equivalent to if there existed a single copy
- Users oblivious to replication

Why weaken consistency?

- Shouldn’t we always strive for single copy semantics?
  - Comes at the expense of lower performance

- Latency vs. consistency tradeoff

Consistency Spectrum



Sequential Consistency

- Results of any execution same as that when the operations by all processes on the data store were executed in some sequential order.
- And … the operations of each individual process appear in this sequence in the order specified by its program.

Example

(a) (b)

- (a) is sequentially consistent. All processes see the same interleaving of the operations (x takes value b before a)
- (b) is not sequentially consistent. P3 sees x taking on value b before a while P4 sees the opposite.

Causal Consistency

- Order of causally related writes must be preserved in values returned to reads
  - If **W1** → **W2**, then if a read sees effect of **W2**, it must see effect of **W1**

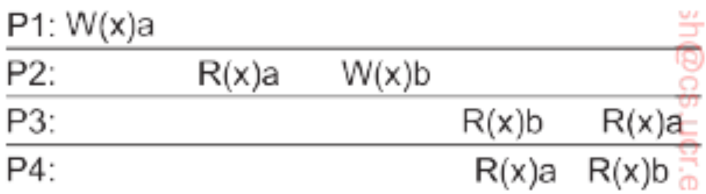
- Example: Facebook News Feed
  - Okay to not see all completed posts
  - But, if you see a comment, you must see the post on which the comment is made
- Main utility: Lazy sync between replicas

Example

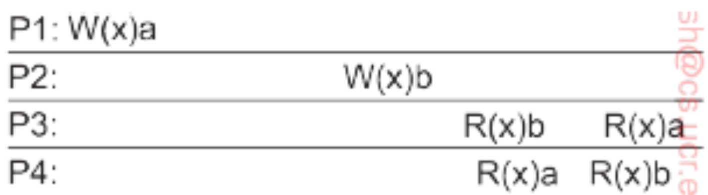
P1:	W(x)a					W(x)c				
P2:			R(x)a		W(x)b					
P3:				R(x)a			R(x)c			R(x)b
P4:		R(x)a						R(x)b	R(x)c	

- The operations above do not result in sequential consistency (see reads of P3 and P4)
- But there is causal consistency. This is because the writes W1(x)c and W2(x)b are concurrent (there are no dependencies and thus no causal relationship).
- But note that W1(x)a and W2(x)b are causally related since R2(x)a may be the reason for W2(x)b.

Example



(a)

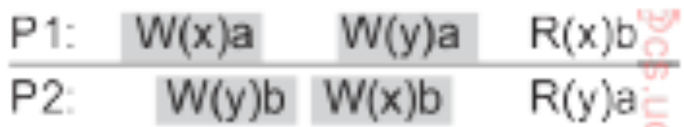


(b)

- (a) is not causally consistent. R3(x)b cannot follow R3(x)a since the the write of b to x is dependent on P2 reading x's value to be a.
- (b) however is causally consistent since W1(x) a and W2(x) b are concurrent.

Linearizability example

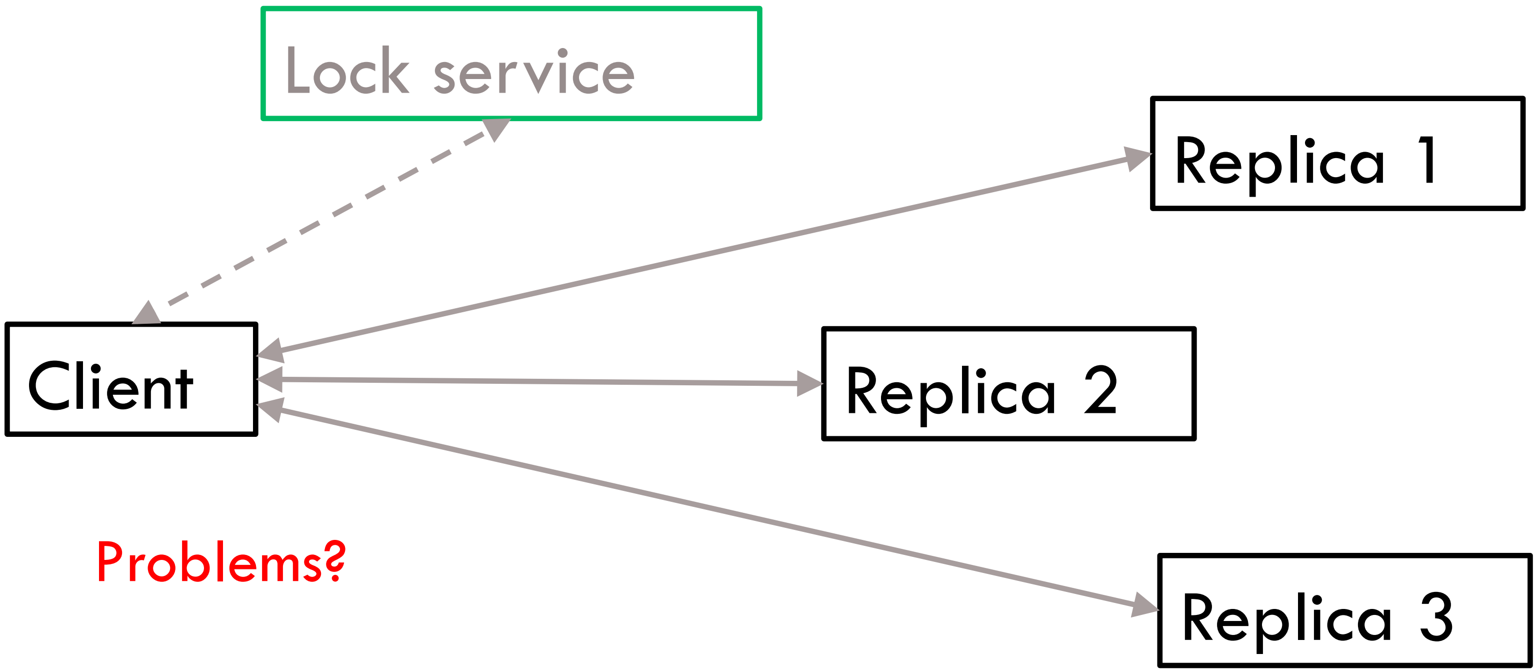
- Every operation should appear to take effect instantaneously at some moment before its start and completion (sometime in the shaded area)



- Thus, the result is consistent regardless of ordering of operations

Ordering of operations	Result	
$W_1(x)a; W_2(y)b; W_1(y)a; W_2(x)b$	$R_1(x)b$	$R_2(y)a$
$W_1(x)a; W_2(y)b; W_2(x)b; W_1(y)a$	$R_1(x)b$	$R_2(y)a$
$W_2(y)b; W_1(x)a; W_1(y)a; W_2(x)b$	$R_1(x)b$	$R_2(y)a$
$W_2(y)b; W_1(x)a; W_2(x)b; W_1(y)a$	$R_1(x)b$	$R_2(y)a$

Linearizability with Locks



Problems?

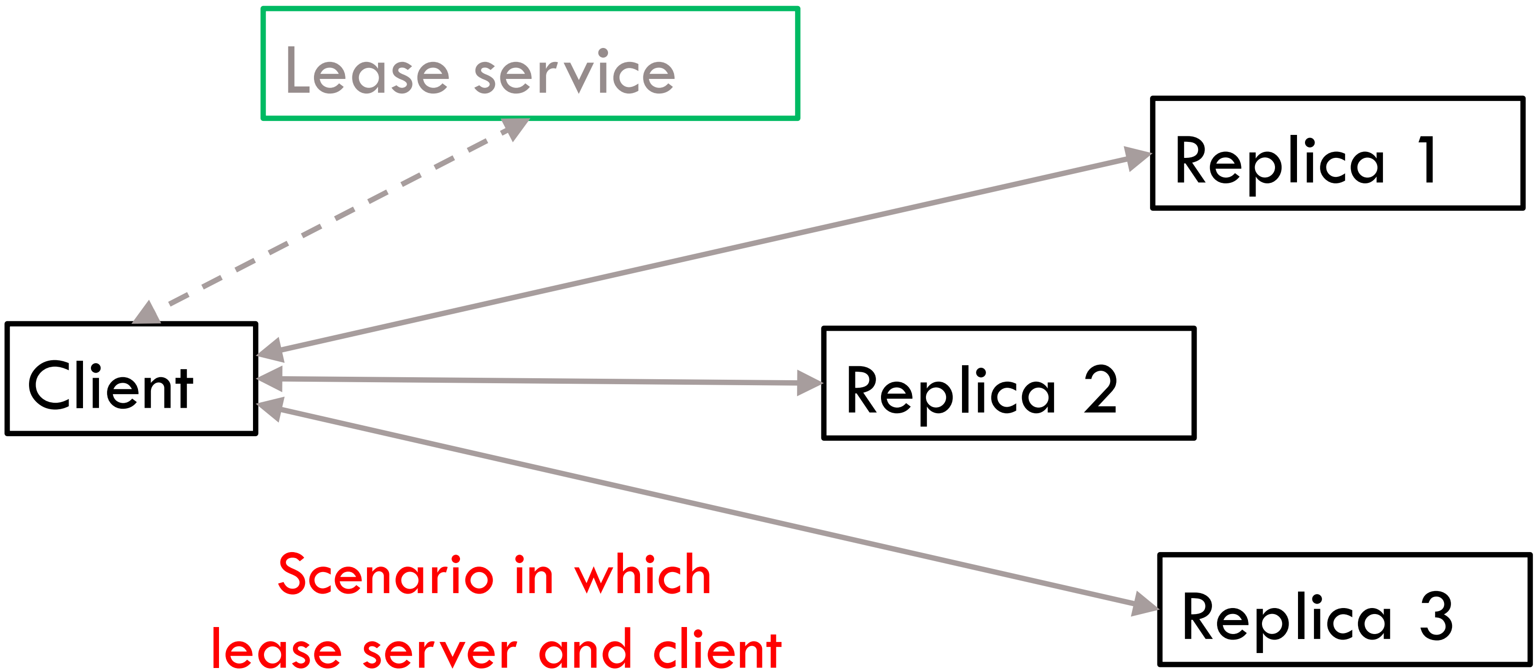
Client failures!

Lease

- Lock with timeout
- If lease holder fails, not a problem because lease will expire

- How to pick lease timeout value?
  - Short timeout → Client needs to renew lease
  - Long timeout → Unnecessarily block operations

Discrepancy in Lease Validity

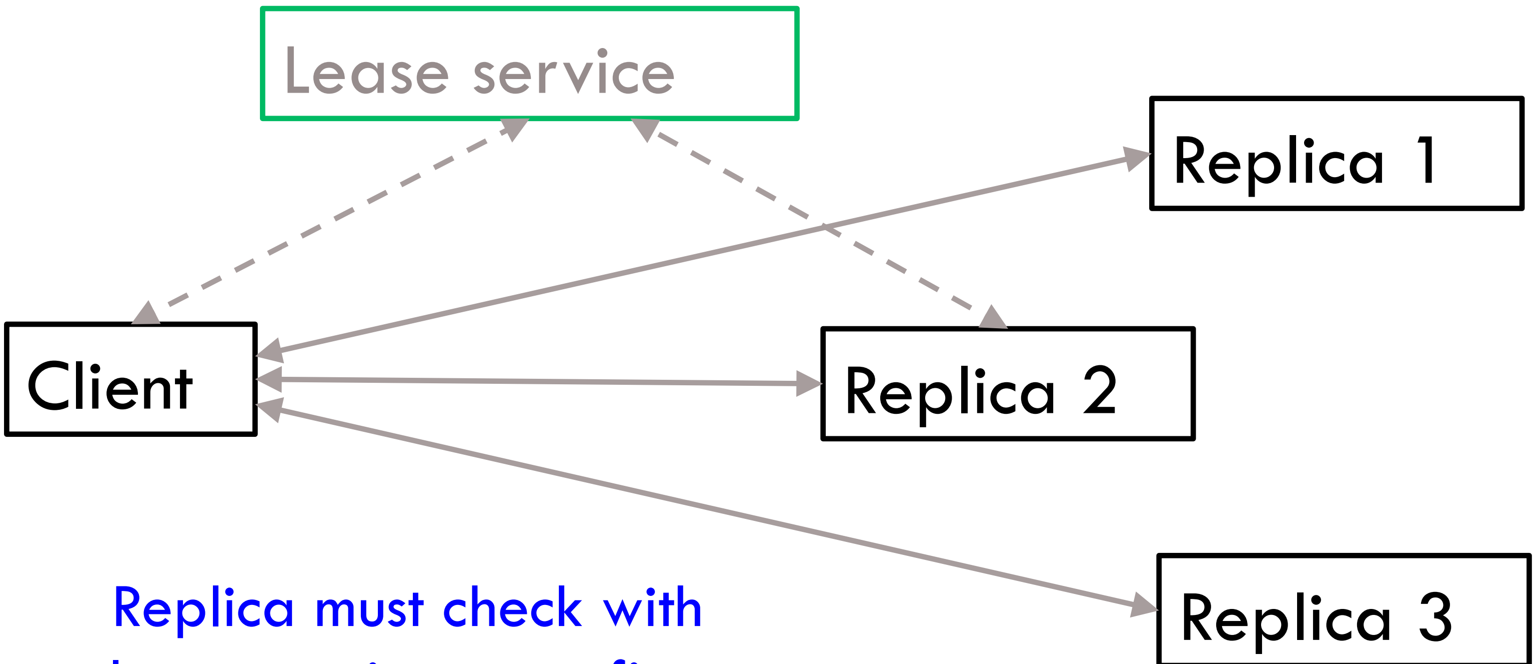


Scenario in which
lease server and client
differ about lease validity?

Discrepancy in Lease Validity

- Message that grants lease may have high delay
- Clock at lease holder and lease service may have different skew
- How to account for potential discrepancy?

Discrepancy in Lease Validity



Replica must check with lease service to confirm lease validity