Haahoos of Transactions
(2PC synchronous)

Outline

- 5 Faces of 2PC in the synchronous world
  - **2PC** blocking transactions
  - **3PC** non-blocking transactions
  - **2PC** non-blocking with single-object replication (aka atomic consistency).
  - **2PC** non-blocking transactions with active replication (all primaries)
  - **2PC** non-blocking with primary backups replication? (we need additional RTs)
- Asynchrony and CAP (Brewer's Conjecture)
- Consensus

*replication is when each shard is replicated in more than one replica for availability

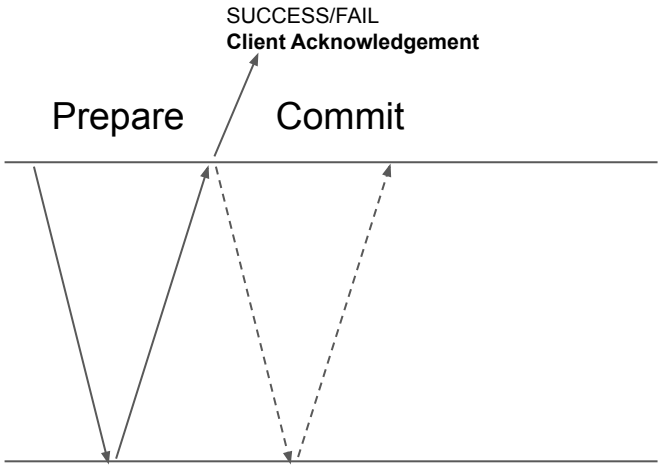
History, Assumptions, Pathology

- 2PC was proposed for fail-restart machines.
- Transaction status is always persistent. No replication.
- Participants decide individually.
- Use timeouts for recovery. (Autonomous recovery)

Two-Phase Commit (2PC)

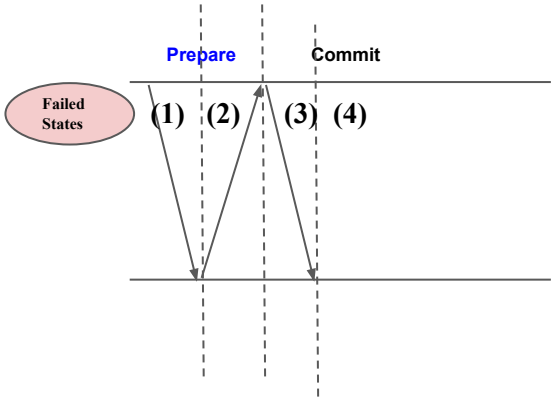
- Phases: prepare and commit.
- Prepare RPC : Vote/Lock + Log
- Commit RPC: Update + Unlock/Release
- Blocking

Two-Phase Commit (2PC)



Prepare → Vote/Lock + Log
Commit → Update + Unlock

Initials {X=0, Y=0, Z=0}, Transaction → {X=1, Y=1, Z=1}



2PC

Coordinator

Participant 1			
X	Lock	Value	Log
(1)	**1**	**0**	**X=1**
(2)			
(3)			
(4)			

Failed States

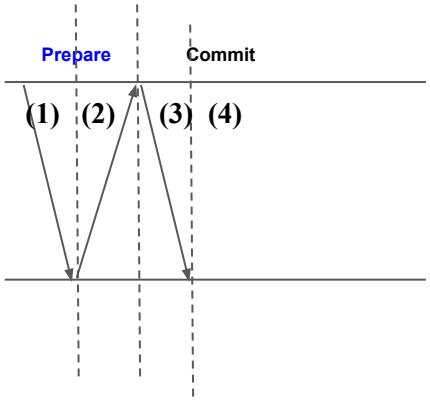
Participant 2			
Y	Lock	Value	Log
(1)	**1**	**0**	**Y=1**
(2)			
(3)			
(4)			

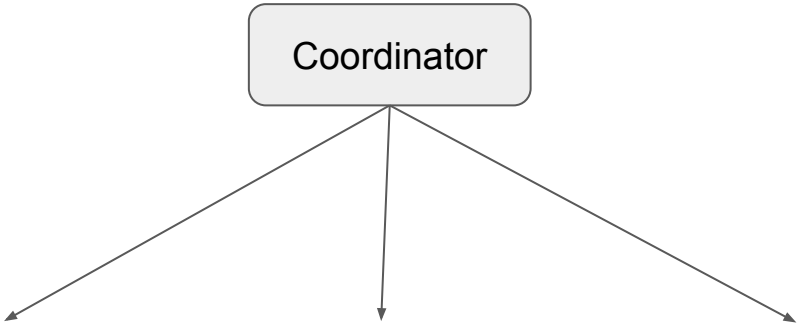
Participant 3			
Z	Lock	Value	Log
(1)	**0**	**0**	-
(2)			
(3)			
(4)			

Failures and Recovery at (1)

- Coordinator fails
  - Problem - (1) Participants doesn’t know others. (2) or the Tx Success?
  - Simple FIx - Log everything on all participants. (or replicate them separately)
- Participants fails
  - Coordinator abort or commit the transaction.
- Coordinator + Participant(s) fails
  - Hold on

2PC



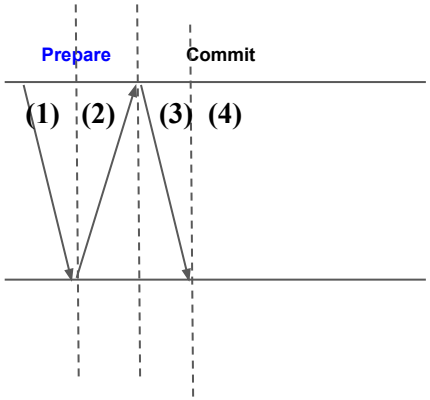


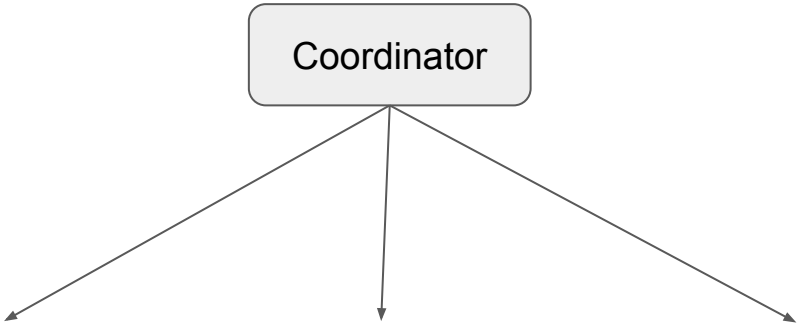
Participant 1			
X	Lock	Value	Log
(1)	**1**	**0**	**X=1,Y=1,Z=1**
(2)	**1**	**0**	**X=1,Y=1,Z=1**
(3)			
(4)			

Participant 2			
Y	Lock	Value	Log
(1)	**1**	**0**	**X=1,Y=1,Z=1**
(2)	**1**	**0**	**X=1,Y=1,Z=1**
(3)			
(4)			

Participant 3			
Z	Lock	Value	Log
(1)	**0**	**0**	-
(2)	**1**	**0**	**X=1,Y=1,Z=1**
(3)			
(4)			

2PC



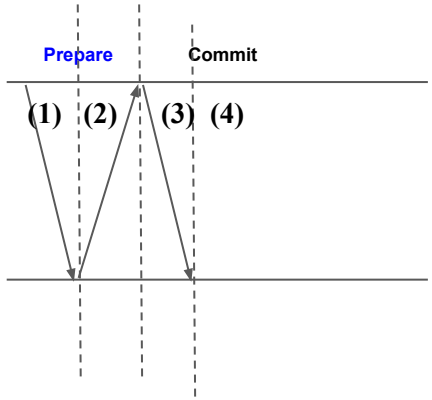


Participant 1			
X	Lock	Value	Log
(1)	**1**	**0**	**X=1,Y=1,Z=1**
(2)	**1**	**0**	**X=1,Y=1,Z=1**
(3)	**0**	**1**	**X=1,Y=1,Z=1**
(4)			

Participant 2			
Y	Lock	Value	Log
(1)	**1**	**0**	**X=1,Y=1,Z=1**
(2)	**1**	**0**	**X=1,Y=1,Z=1**
(3)	**1**	**0**	**X=1,Y=1,Z=1**
(4)			

Participant 3			
Z	Lock	Value	Log
(1)	**0**	**0**	-
(2)	**1**	**0**	**X=1,Y=1,Z=1**
(3)	**1**	**0**	**X=1,Y=1,Z=1**
(4)			

Adverse Case



Another Coordinator

Read X=1
(cannot rollback)

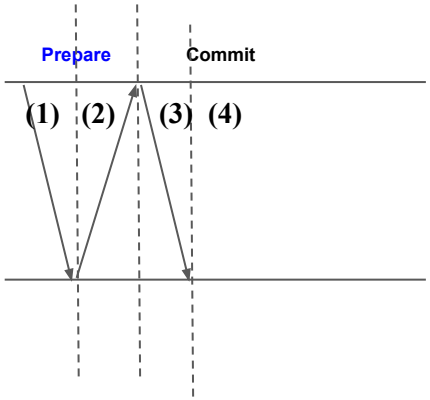
Coordinator

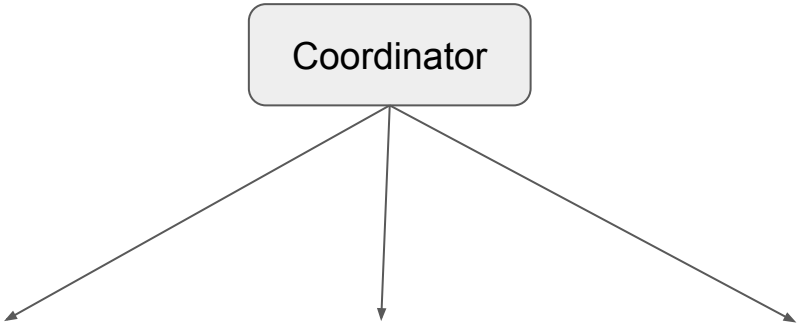
Participant 1			
X	Lock	Value	Log
(1)	1	0	X=1,Y=1,Z=1
(2)	1	0	X=1,Y=1,Z=1
(3)	0	1	X=1,Y=1,Z=1
(4)			

Participant 2			
Y	Lock	Value	Log
(1)	1	0	X=1,Y=1,Z=1
(2)	1	0	X=1,Y=1,Z=1
(3)	1	0	X=1,Y=1,Z=1
(4)			

Participant 3			
Z	Lock	Value	Log
(1)	0	0	-
(2)	1	0	X=1,Y=1,Z=1
(3)	1	0	X=1,Y=1,Z=1
(4)			

2PC





Participant 1			
X	Lock	Value	Log
(1)	1	0	X=1,Y=1,Z=1
(2)	1	0	X=1,Y=1,Z=1
(3)	0	1	X=1,Y=1,Z=1
(4)	0	1	X=1,Y=1,Z=1

Participant 2			
Y	Lock	Value	Log
(1)	1	0	X=1,Y=1,Z=1
(2)	1	0	X=1,Y=1,Z=1
(3)	1	0	X=1,Y=1,Z=1
(4)	0	1	X=1,Y=1,Z=1

Participant 3			
Z	Lock	Value	Log
(1)	0	0	-
(2)	1	0	X=1,Y=1,Z=1
(3)	1	0	X=1,Y=1,Z=1
(4)	0	1	X=1,Y=1,Z=1

2PC Failures and Recovery at (2)/(3)

- Coordinator fails
  - Recovery check logs and all values; commit or abort.
- Participants fails
  - Coordinator abort or commit the transaction.
- Coordinator + Participant (Let's say X) fail simultaneously?
  - Status of X is ambiguous. Cannot distinguish (2) and (3).
  - If (2), cannot **commit**; X might have not voted and locked by someone else.
  - If (3), cannot **abort** because another coordinator has already seen X=1.

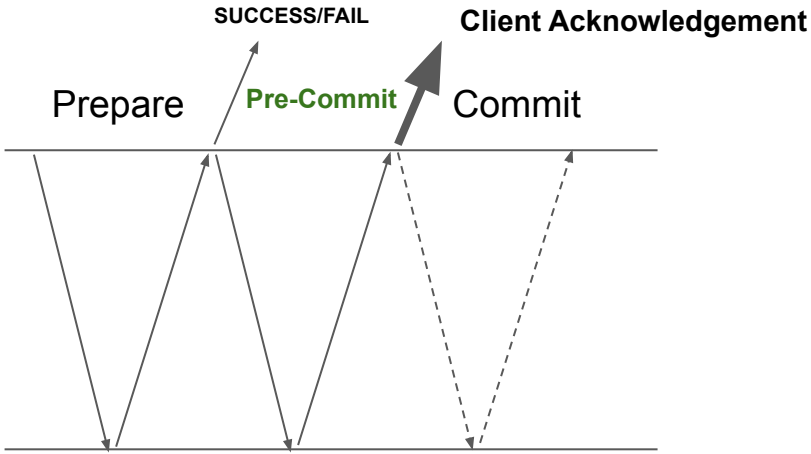
2PC Problems

- If the coordinator and a participant fails, cannot recover.
- Need ***at least*** one commit on any active participants side to recover. If not,
- Either break the atomicity (*Safety*) or *blocking* (*Liveness*).

Three-Phase Commit (2PC)

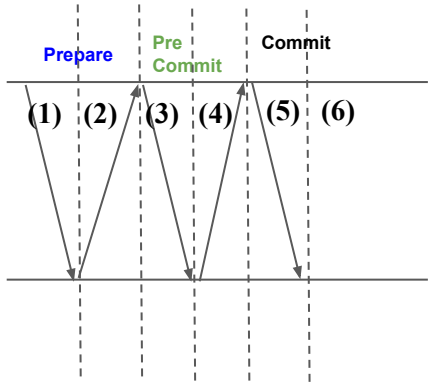
- Phases: Prepare, **Pre-commit** and commit.
- Prepare RPC: Lock + Log using.
- Pre-commit RPC: Log the decision in all participants.
- Commit RPC: Update + Unlock.
- Advantage: Non-blocking.

Three-Phase Commit (3PC)



Prepare → Vote/Lock + Log
Pre-commit → Log the coordinator's decision
Commit → Update + Unlock

3PC



Coordinator

Participant 1			
X	Lock	Value	Log
(1)	**1**	**0**	**X=1,Y=1,Z=1**
(2)	**1**	**0**	**X=1,Y=1,Z=1**
(3)	**1**	**0**	**X=1,Y=1,Z=1 Pre-Commit**
(4)	**1**	**0**	**X=1,Y=1,Z=1 Pre-Commit**
(5)	**0**	**1**	**X=1,Y=1,Z=1**
(6)	**0**	**1**	**X=1,Y=1,Z=1**

Participant 2			
Y	Lock	Value	Log
(1)	**1**	**0**	**X=1,Y=1,Z=1**
(2)	**1**	**0**	**X=1,Y=1,Z=1**
(3)	**1**	**0**	**X=1,Y=1,Z=1**
(4)	**1**	**0**	**X=1,Y=1,Z=1 Pre-Commit**
(5)	**1**	**0**	**X=1,Y=1,Z=1**
(6)	**0**	**1**	**X=1,Y=1,Z=1**

Participant 3			
Z	Lock	Value	Log
(1)	**0**	**0**	-
(2)	**1**	**0**	**X=1,Y=1,Z=1**
(3)	**1**	**0**	**X=1,Y=1,Z=1**
(4)	**1**	**0**	**X=1,Y=1,Z=1 Pre-Commit**
(5)	**1**	**0**	**X=1,Y=1,Z=1**
(6)	**0**	**1**	**X=1,Y=1,Z=1**

Observation - 2PC does not block when **participants' status survive**

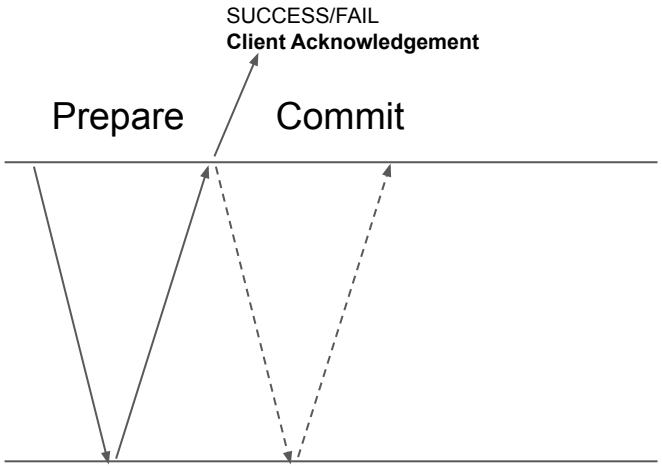
So, when is 2PC non-blocking?

- Without 3PC :).
- trivially, when there are no failures.
- **Can you think of a case(s) for non-blocking 2PC?**

So, when is 2PC non-blocking?

- without 3PC :).
- when there are no failures. Trivial :):).
- **Can you think of a case(s) for non-blocking 2PC?**
  - **Single object replication** **(when all participants are replicas of the same object)**
  - **Replicated transactions** **(when every participant is replicated)**

Recall Two-Phase Commit (2PC)

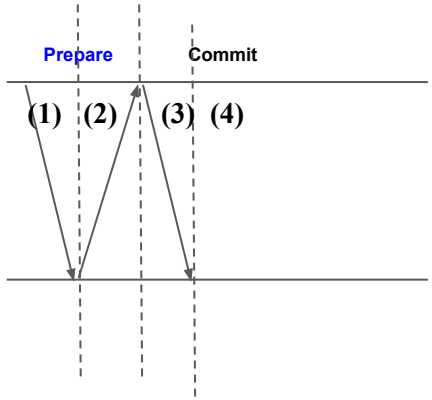


Prepare → Vote/Lock + Log
Commit → Update + Unlock

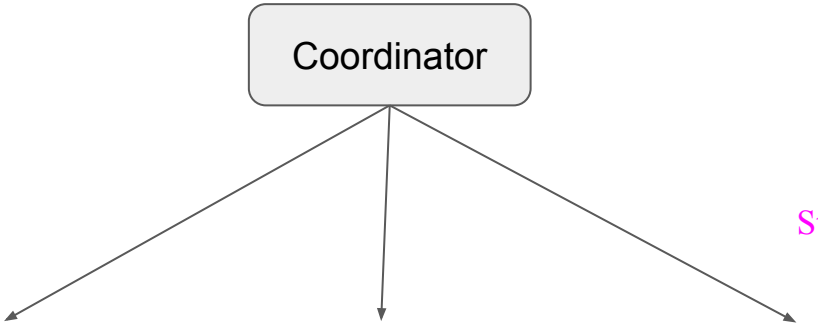
2PC - Single Object Replication

- We need to atomically update the value in all replicas.
- Similar to Transactions
  - Tx = {X=1, X’ =1, X’’=1}

2PC non-blocking



Status Survive. Commit or abort



Replica 1			
X	Lock	Value	Log
(1)	**1**	**0**	**X=1**
(2)	**1**	**0**	**X=1**
(3)	**0**	**1**	**X=1**
(4)	**0**	**1**	**X=1**

Replica 2			
X	Lock	Value	Log
(1)	**0**	**0**	-
(2)	**1**	**0**	**X=1**
(3)	**1**	**0**	**X=1**
(4)	**0**	**1**	**X=1**

Replica 3			
X	Lock	Value	Log
(1)	**0**	**0**	-
(2)	**1**	**0**	**X=1**
(3)	**1**	**0**	**X=1**
(4)	**0**	**1**	**X=1**

2PC non-blocking Recovery

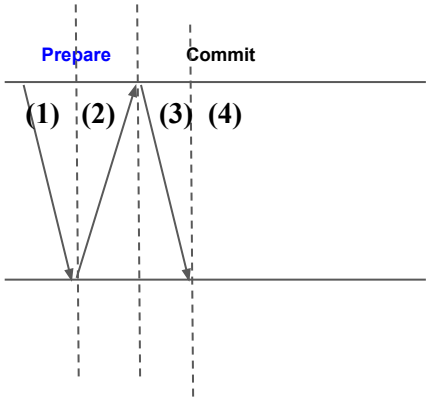
- If at least one replica survives then the decision survives.
- Failed replicas have commited → all active ones are locked.
    - Recovery commit and unlock other replicas.
    - Dead ones can be removed or repair later.
- Failed replicas have not committed →
    - Partially locked. Can commit/abort later.
    - Nothing is locked. Abort or nothing to do.

2PC non-blocking Recovery

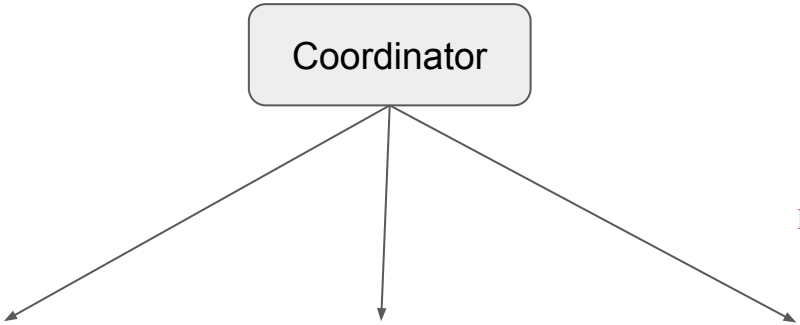
- All active replicas are clean → nothing (Tx is not successful. abort)
- All/some active replicas are locked → commit
- All/Some replicas are dirty → commit
- (always forward progress if the prepare survives in at least one replica)

Opt: We can directly update the values if wants. Conflict must be resolved separately.

2PC non-blocking without logging



Logging can be avoided (versions or locks needed)

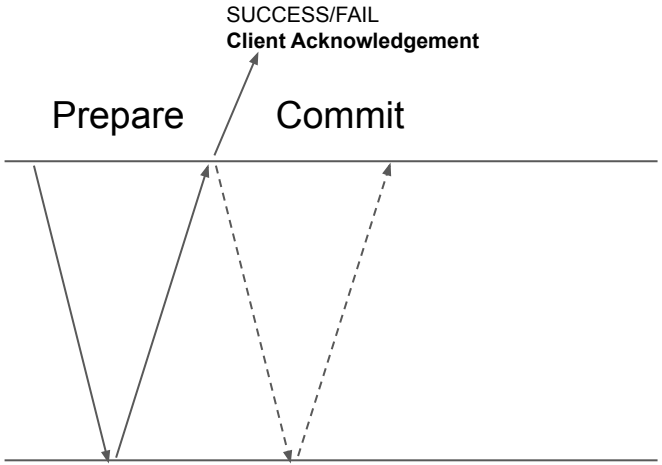


Replica 1		
X	Lock	Value
(1)	**1**	**0**
(2)	**1**	**1**
(3)	**0**	**1**
(4)	**0**	**1**

Replica 2		
X	Lock	Value
(1)	**0**	**0**
(2)	**1**	**1**
(3)	**1**	**1**
(4)	**0**	**1**

Replica 3		
X	Lock	Value
(1)	**0**	**0**
(2)	**1**	**1**
(3)	**1**	**1**
(4)	**0**	**1**

Two-Phase Commit (2PC) Single Object Replication



Prepare → Vote/Lock+ **Update**
Commit → Unlock

No Logging is needed

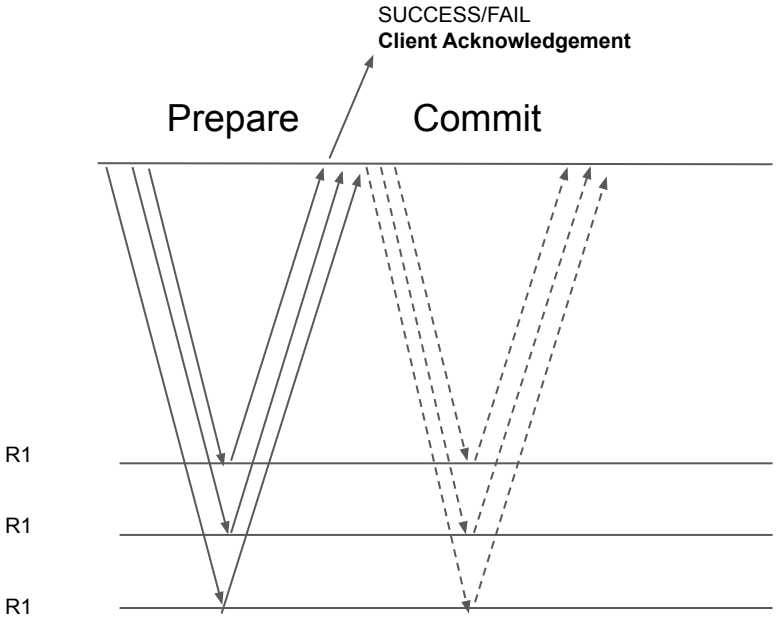
Recall, when is 2PC non-blocking?

- I mean without 3PC :).
- when there are no failures. Trivial :).
- **Another case(s) for non-blocking 2PC?**
  - **Single object replication** **(when all participants are replicas of the same object)**
  - **Replicated transactions** **(when every participant is replicated)**

Replicated Transactions (2PC with active replication)

- Lets replicate everything: locking, logging, update.
- Prepare RPC: Lock all replicas + Log all replicas
- Commit RPC: Update all replicas + Unlock all replicas

Replicated Transactions (2PC with active replication)



Prepare → (Vote/Lock + Log) all replicas
Commit → (Update + Unlock) all replicas

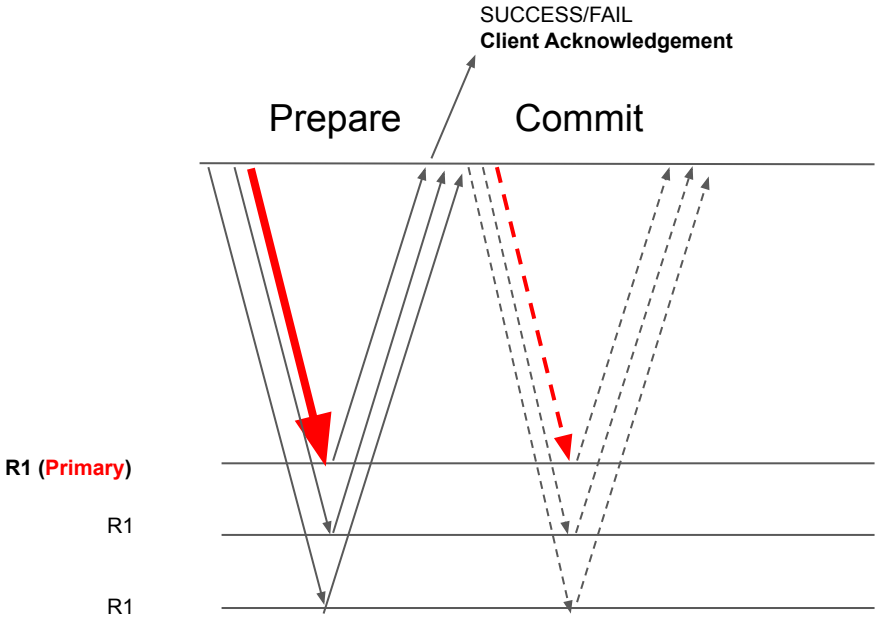
Failures and Recovery

- Before commit - wait for all lock and log.
- If a participant + coordinator fail
  - construct logs of all participants. If all logs have received
  - If at least one update on any replica survives, transaction commits.
  - If a replica has failed after committing, we cannot rollback.
  - *Check all active locks*, if all of the have been locked, then commit (regardless of failed replicas)

Questions?

- Can we use primary backup to avoid locking all replicas overhead?
- Can we use 2PC with primary-backup?.
- Bonus 5

Replicated Transactions (2PC with Primary Backup replication)



Prepare → Vote/Lock Primary + Log all replicas
Commit → Update all replicas + Unlock Primary

Commit is not an **atomic** action? Any problems.
Issues - Transactions can be reordered in backups?

Primary Backups Problem 1

- Atomicity violations in backups. Need update→unlock ordering ordering.
- How to achieve ordering in backups?.
- Simple fix:
    - 2 RTTS for update and unlock. No longer just 2 phases.
    - Lock all replicas. Fall back to 4.

Primary Backups Problem 2

- Only wait for the *primary locks*
- Primary failures. But log before lock. problem?
- without locking all replicas, we cannot reckon if the transaction is successful or not from replica logs.
- We cannot check if the transactions is successful by comparing their values without metadata. ambiguity - FAIL/SUCCESS/COMMITTED
- Simple Fixes -
    a. log after locking. Or additional log message.
    b. Fall back to 4. Lock all replicas.
    c. Could metadata solves the problem?.

Example

- T1 = {X=1, Y=1}, T2 = {X=2, Z=2}
- T1 fails after logging and coordinator dies. T2 succeeds.
- Primary of X also fails.
- potential issues
    - Failed coordinator has not acquired locks. But logged.
    - Coordinator unlatched locks after an abort.

Replica 1			
X	Lock	Value	Log
(1)	**1 (T2)**	**0**	**X=2**
(2)	**1 (T2)**	**0**	**X=2**
(3)	**0 (T2)**	**2**	**X=2**
(4)	**0**	**2**	**X=2**

Replica 2		
X	Value	Log
(1)	**0**	-
(2)	**0**	**X=2/X=1,Y=1**
(3)	**0**	**X=2/X=1,Y=1**
(4)	**2**	**X=2/X=1,Y=1**

Replica 3		
X	Value	Log
(1)	**0**	-
(2)	**0**	**X=2/X=1,Y=1**
(3)	**0**	**X=2/X=1,Y=1**
(4)	**2**	**X=2/X=1,Y=1**

Issue:
Log before locking

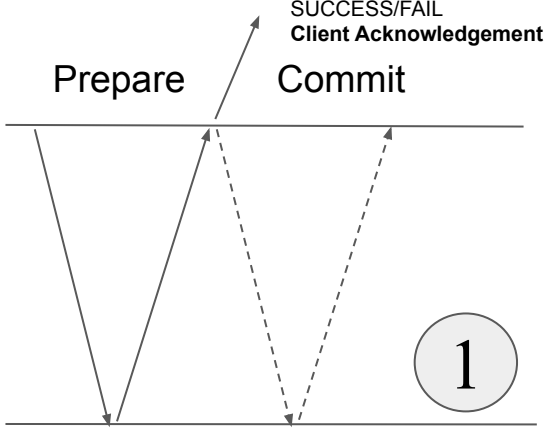
Replica 1			
Y	Lock	Value	Log
(1)	**1 (T1)**	**0**	**X=1,Y=1**
(2)	**1 (T1)**	**0**	**X=1,Y=1**
(3)			
(4)			

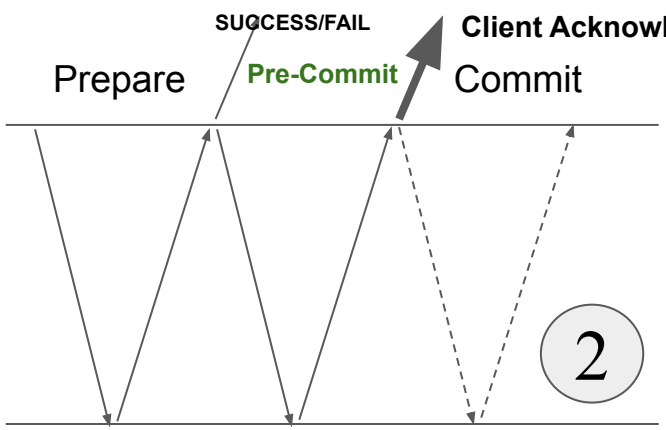
Replica 2		
Y	Value	Log
(1)	**0**	-
(2)	**0**	**X=1,Y=1**
(3)		
(4)		

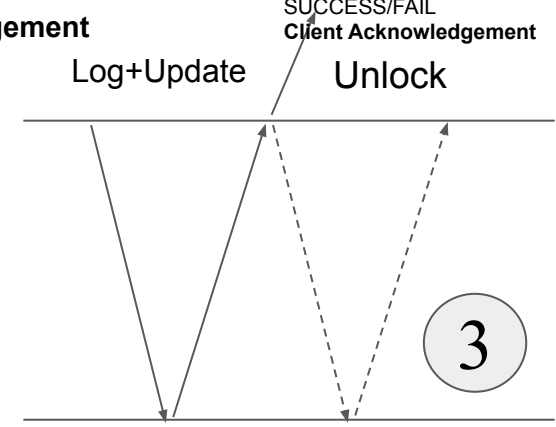
Replica 3		
Y	Value	Log
(1)	**0**	-
(2)	**0**	**X=1,Y=1**
(3)		
(4)		

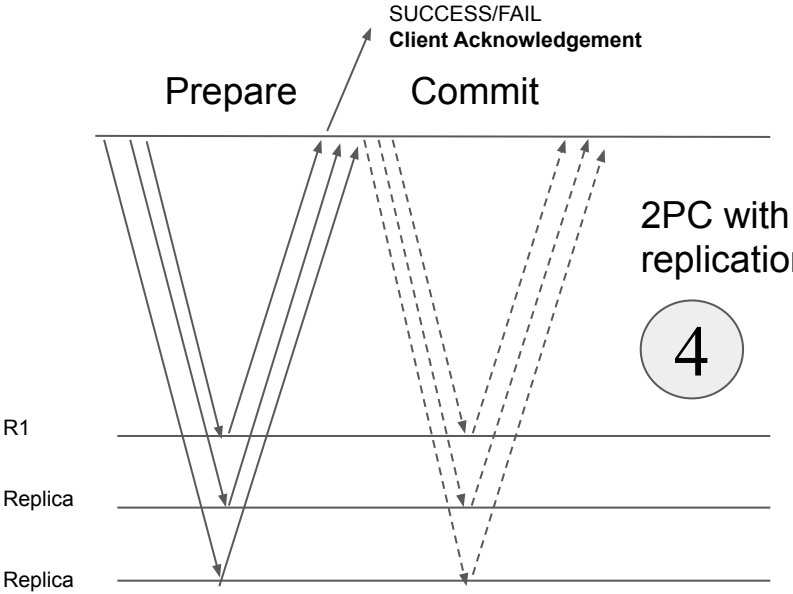
Summary

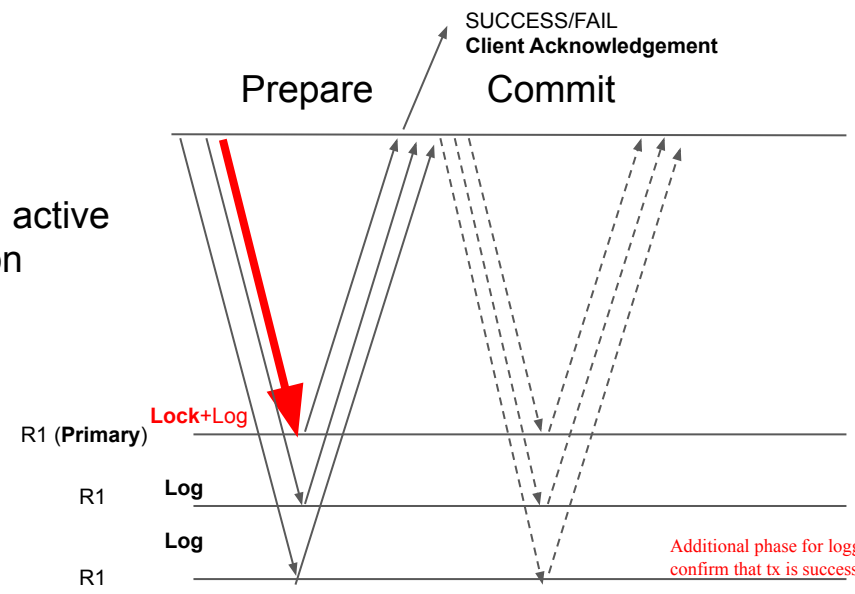
- There are two problems with primary-backup. (atomicity and recovery)
- Recovery problem with primary-backup. (recovery races)
  - Log before locking all replicas
  - Log before the transactions is successful.
- Overlap in pessimistic transactions.
- In optimistic , Lock → Log → Succ. (careful in one-sided)
- Let’s discuss more in one-sided.
- Need more metadata or lock in all replicas.









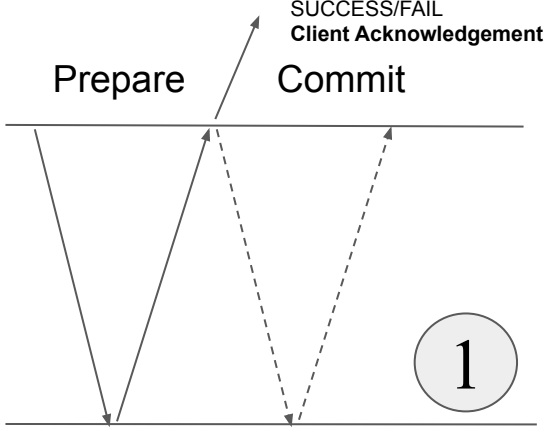


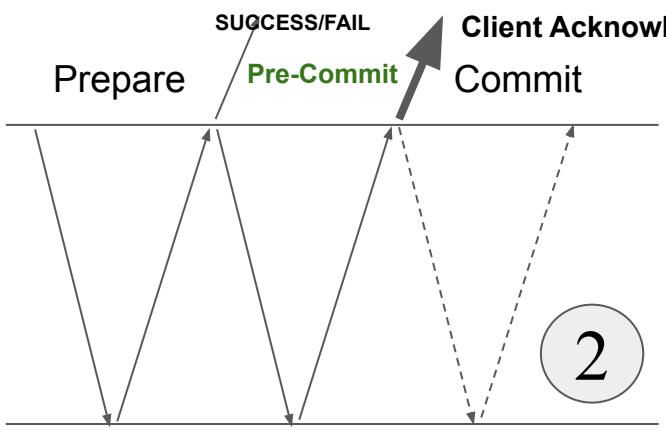
2PC Summary so far

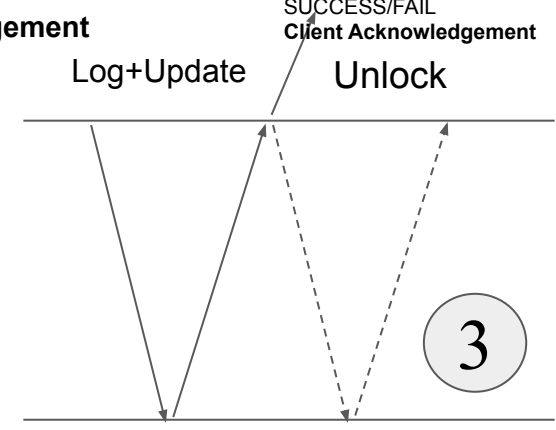
Primary-Backup
Need more RTTS?

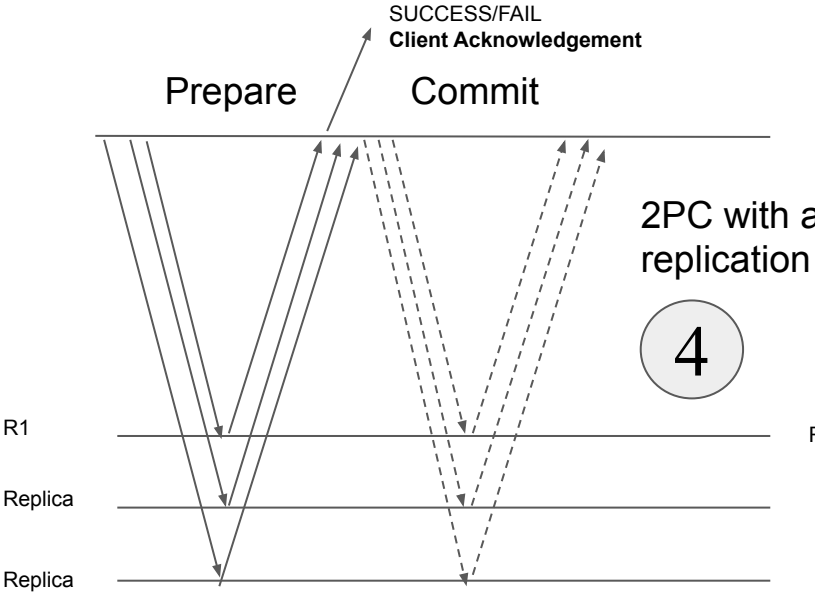
5

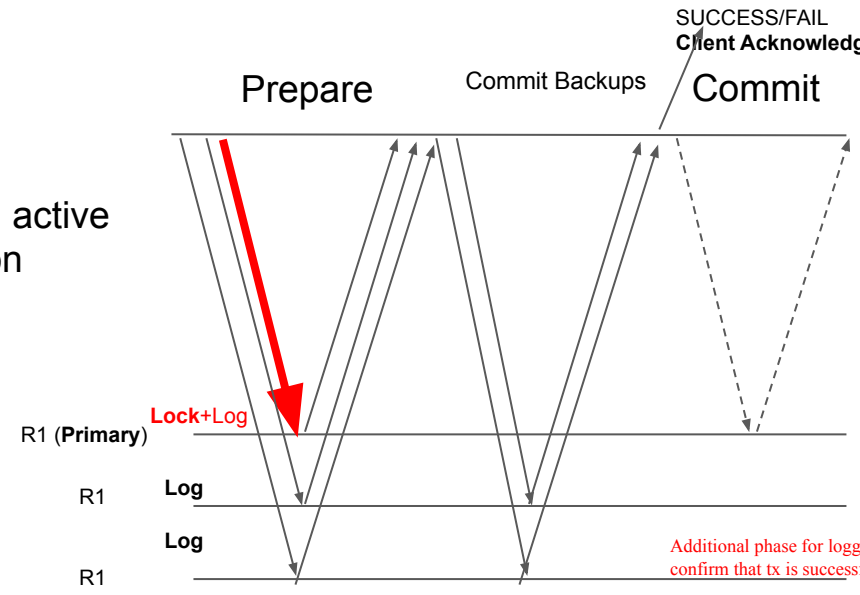
Additional phase for logging backup updating backups for confirm that tx is successful. Update primary later.


SUCCESS/FAIL
Client Acknowledgement
Prepare
Commit
1


SUCCESS/FAIL
Client Acknowledgement
Prepare
Pre-Commit
Commit
2


SUCCESS/FAIL
Client Acknowledgement
Log+Update
Unlock
3


SUCCESS/FAIL
Client Acknowledgement
Prepare
Commit
2PC with active replication
4
R1
Replica
Replica


SUCCESS/FAIL
Client Acknowledgement
Prepare
Commit Backups
Commit
RPC Summary so far
Primary-Backup
Need more RTTS
5
R1 (Primary)
Lock+Log
R1
Log
Log
R1
Additional phase for logging backup updating backups for confirm that tx is successful. Update primary later.

With READS

Recall: assumptions so far (only writes commit)

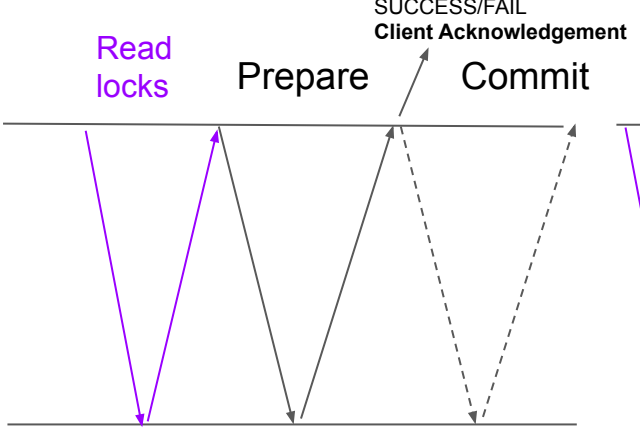
- **Only writes**. No read-writes or reads. Reads make the logging complicated.
- Reads need locking (pessimistic) or validation (optimistic).
- This affect coordinator failures as well.
- Pessimistic
    - Either we need to log reads as well. Treat everything read-write. (hard to take redo at once. Reads need an additional round trip anyway.)
    - Additional round trips.

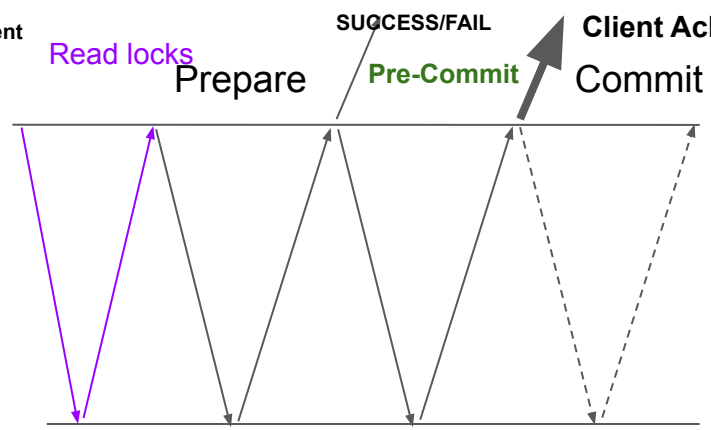
Solutions with read/read-writes.

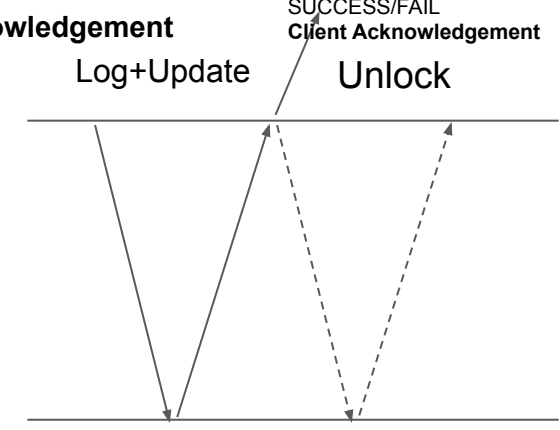
- **In any case, an additional read round trip would solve the problem.**
  - By the time we take logs, we know all reads are valid
  - Not possible with OCC (reads are validated after locking)
- **The other solution is to log after locking or use pre-commit (3PC)**
  - Latter is only when reads and writes can be locked together.

example

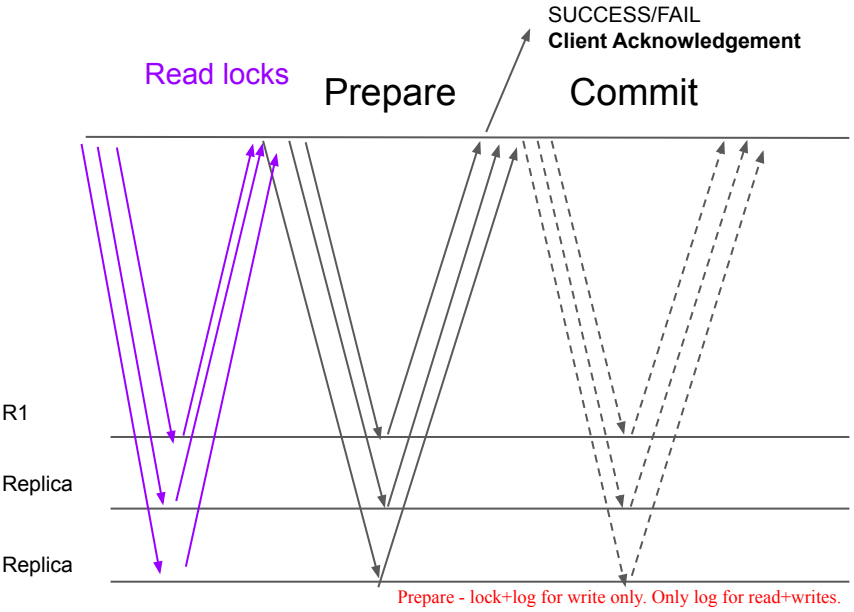
- T1 = {Read X=1, Write Y=2}, T2 = {Read Y =1, Write X=2}
- Both transactions fails after locking X (T2) and Y(T1).
- Recovery cannot just look at redo logs and replay the transactions.



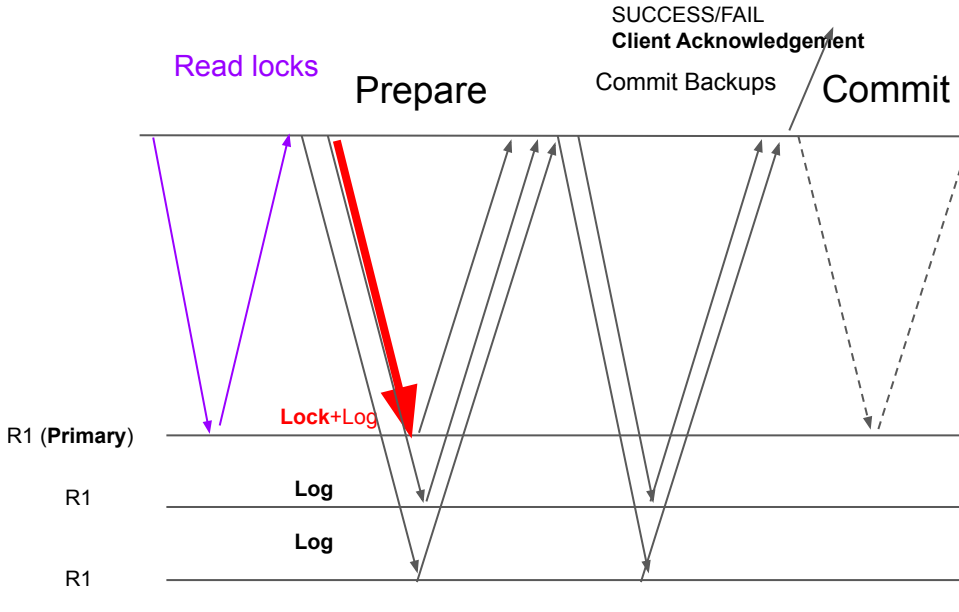




Prepare - lock+log for write only. Only log for read+writes.

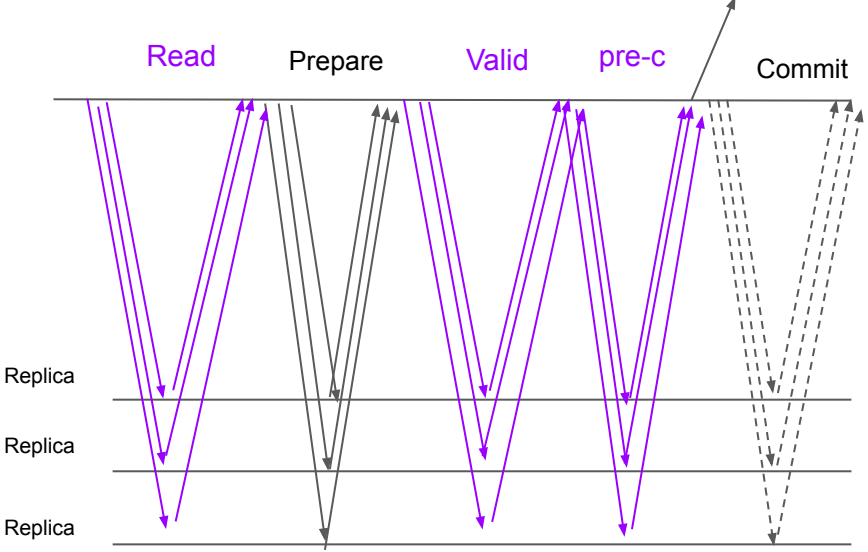


Prepare - lock+log for write only. Only log for read+writes.



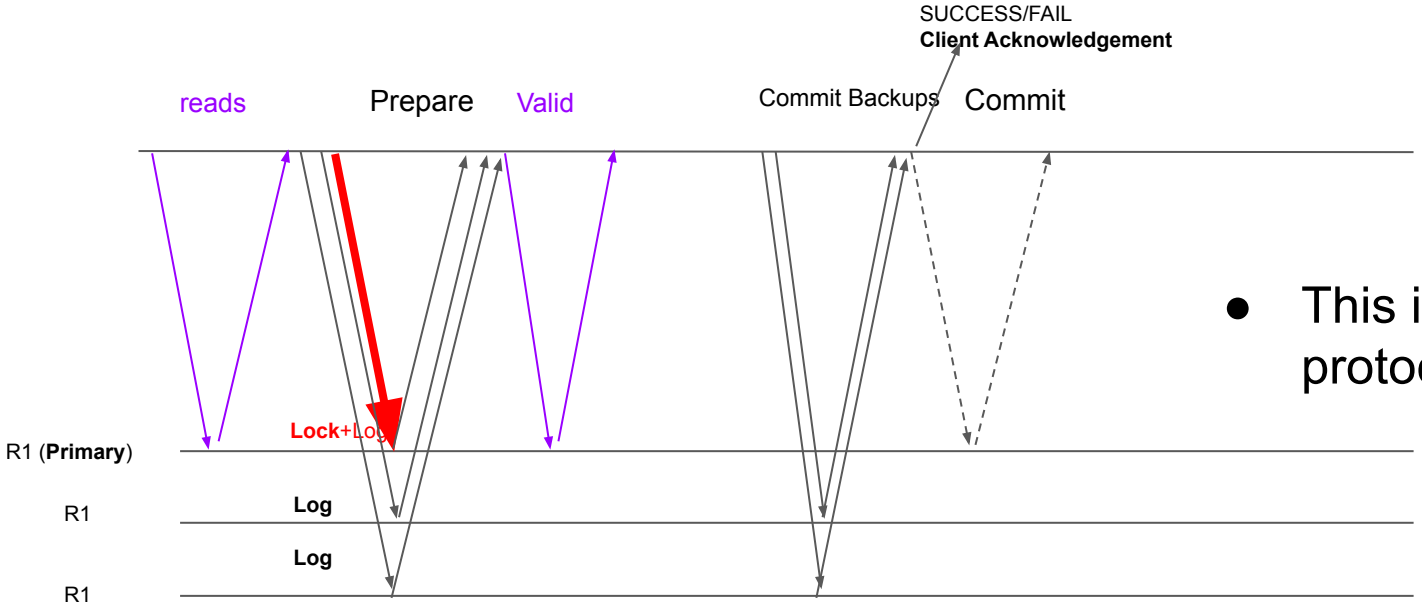
OCC (with replication)

- OCC does read validation later.
- So can we take logs before validation?
- Answer is yes. But need a sort of pre-commit to confirm that tx is successful.
- Or simple - we take redo logs after the validation.
- Undo logging?



OCC

- Active - too many messages.
- Primary-backup - additional round trip.



- This is FaRM protocol

Tricky? Can I use commit backup here for TX confirmation.

- Validation → log or confirm must be ordered. If logs are taken before we need an additional round trip to confirm the transactions
- Question is can we use backup commit as the confirmation.
- Two things. Everyone takes logs or only primary. FORD does there former.

Simple solution
Send commit backup message to all replicas including primary. But only backups update.

OCC

- Primary failure?
- Backup failure?

Recovery

- Primary failure?. How to confirm if the tx is SUCC.
- Backup failure?