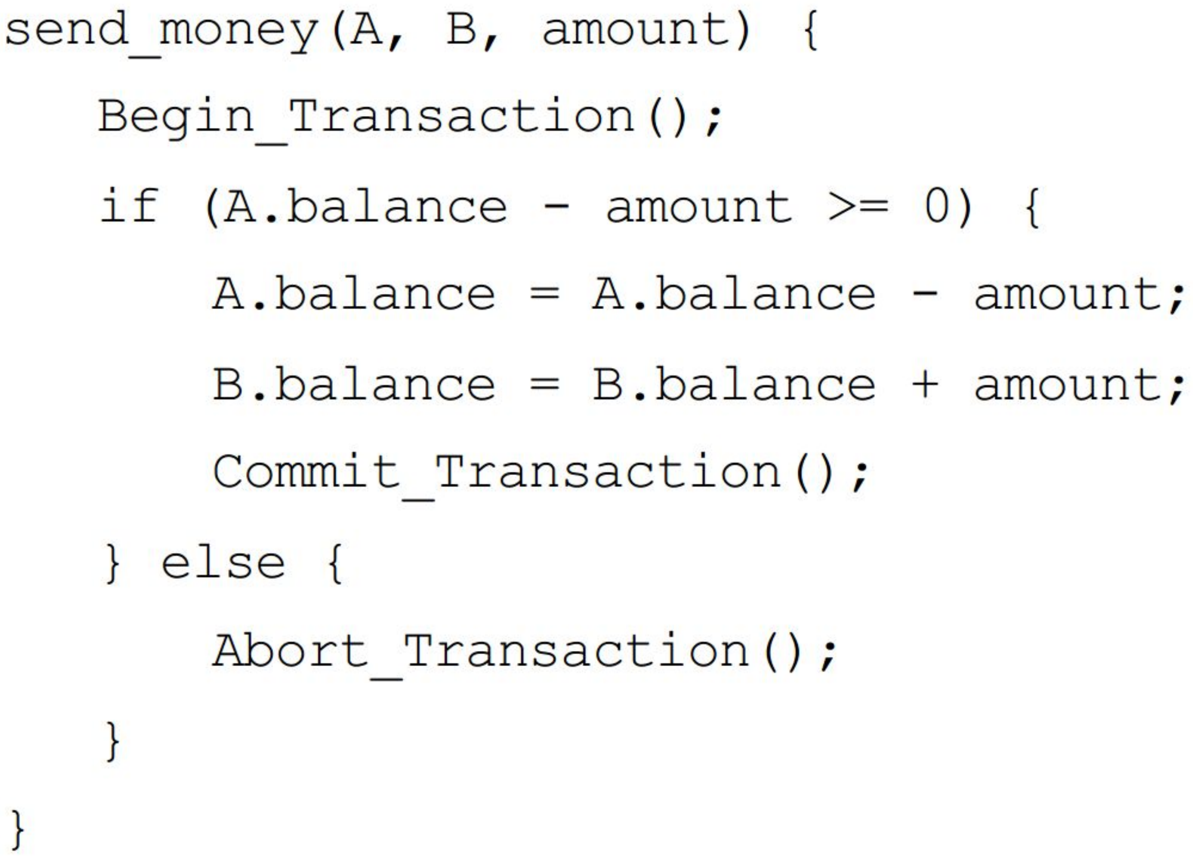
Distributed Two-Phase Commit Protocol with fault tolerance

Anmol Agarwal
Shrey Gupta
Ashwin Mittal

Distributed Systems Course Project

April 2023

Motivation: sending money



```
send_money(A, B, amount) {
    Begin_Transaction();
    if (A.balance - amount >= 0) {
        A.balance = A.balance - amount;
        B.balance = B.balance + amount;
        Commit_Transaction();
    } else {
        Abort_Transaction();
    }
}
```

Single-server: ACID

- **Atomicity:** all parts of the transaction execute or none
  - (A’s decreases and B’s balance increases)
- **Consistency:** the transaction only commits if it preserves invariants
  - (A’s balance never goes below 0)
- **Isolation:** the transaction executes as if it executed by itself
  - (even if C is accessing A’s account, that will not interfere with this transaction)
- **Durability:** the transaction’s effects are not lost after it executes (updates to the balances will remain forever)

Distributed transactions?

Partition databases across multiple machines for scalability (customer “A” and customer “B” might not share a server)

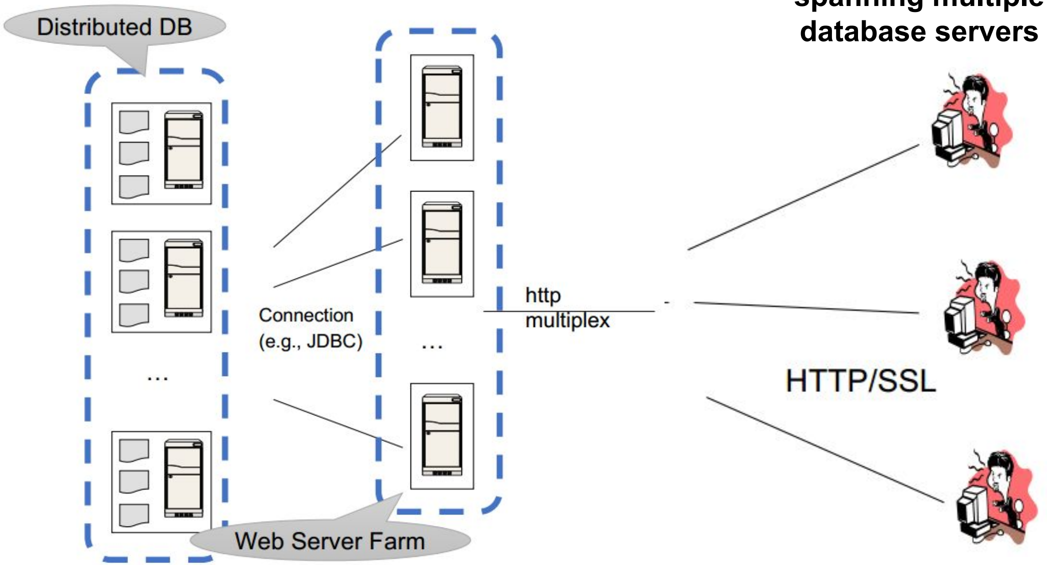
Why the need for distributed databases ?

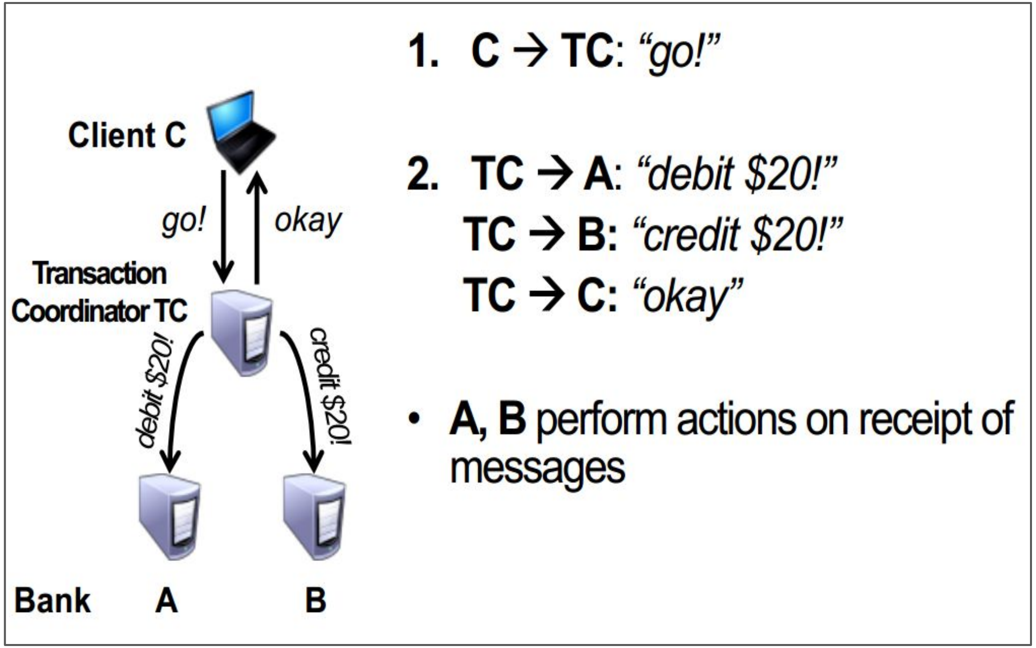
- There is a limit on transactions/sec on one server
- Need to partition the database across multiple servers
- If a transaction touches one machine, life is good!
- If a transaction touches multiple machines, ACID becomes extremely expensive!

Different <u>sites</u> each having a database server

Different <u>Transaction Coordinators</u>

Different clients initiating transactions spanning multiple database servers





1. C → TC: *"go!"*
2. TC → A: *"debit $20!"*
   TC → B: *"credit $20!"*
   TC → C: *"okay"*

- A, B perform actions on receipt of messages

What can go wrong ?

1) Not enough money in **A's** bank account?
2) **B's** bank account no longer exists?
3) **A** or **B** **crashes** before receiving message?
4) **TC** **crashes** after it sends debit to A but before sending to B?
5) **Network failure** between A and TC

Goals

Multiple servers agree on some action despite failures with the following properties:

1) **Safety**
- If one **commits**, no one **aborts**
- If one **aborts**, no one **commits**

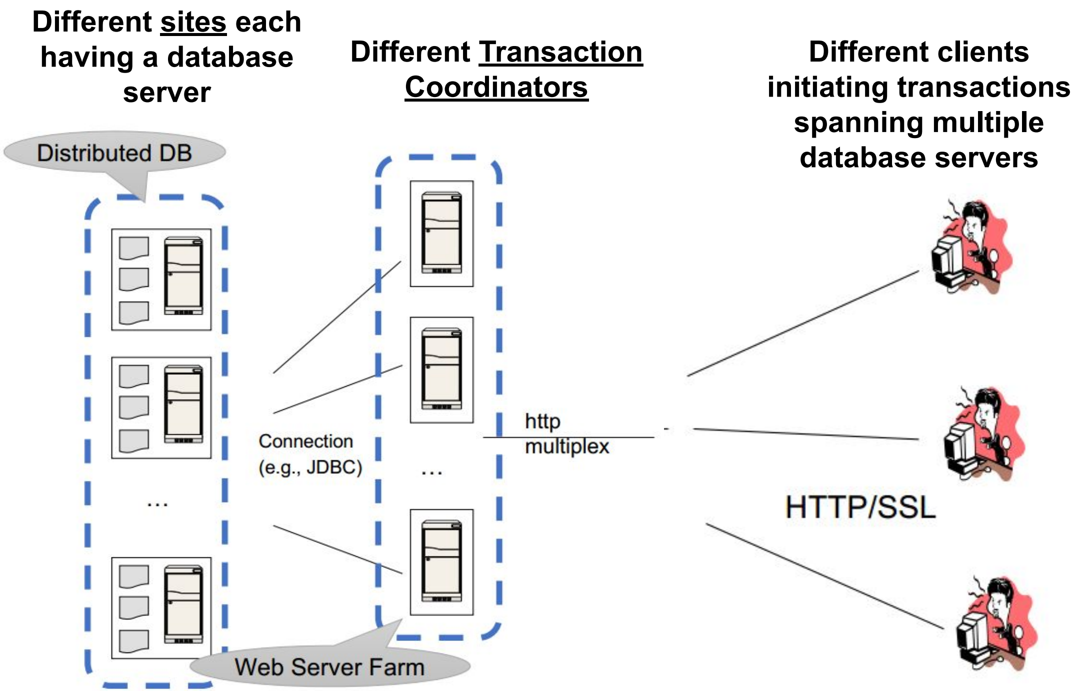
2) **Liveness**
- If **no failures** and A and B can commit, **action commits**
- If **failures**, reach a conclusion ASAP

The components

- **Client**: the machine requesting some transaction (whose updates spans multiple databases) to be taken

- **Transaction Coordinator**: coordinates transaction feasibility/final status at the differer sites via the 2PC protocol

- **Database server (site)**: machine that takes the action

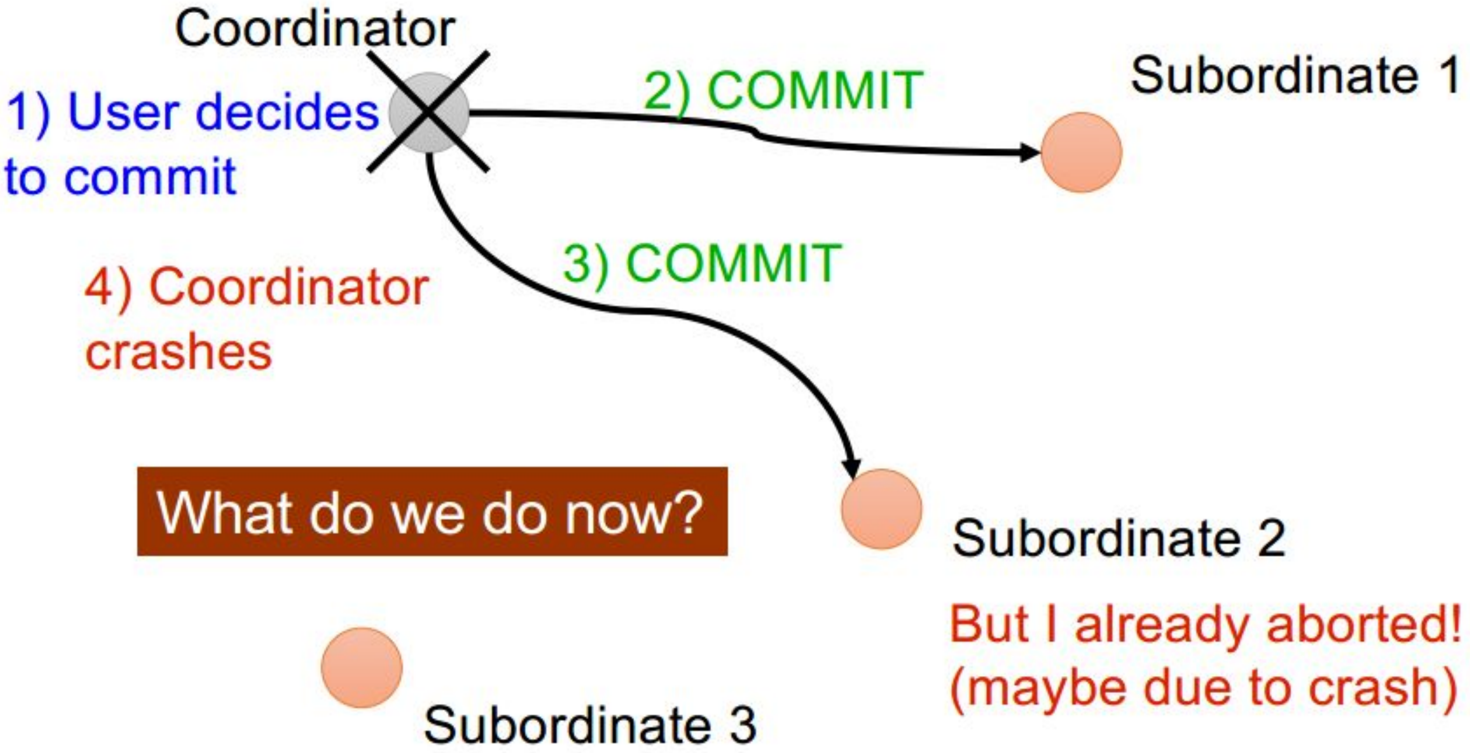


Different sites each having a database server

Different Transaction Coordinators

Different clients initiating transactions spanning multiple database servers

Motivation figure



What do we do now?

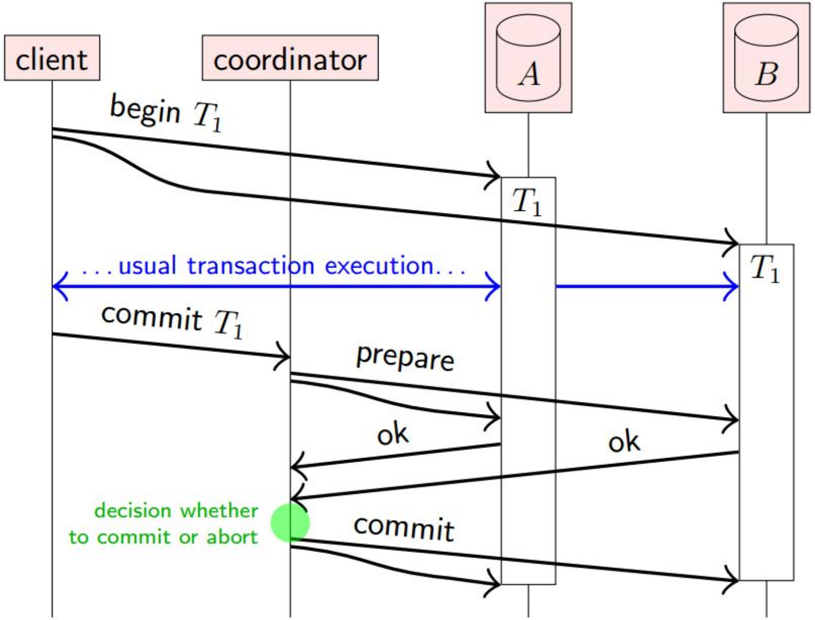
But I already aborted! (maybe due to crash)

Two-Phase Commit (2PC)

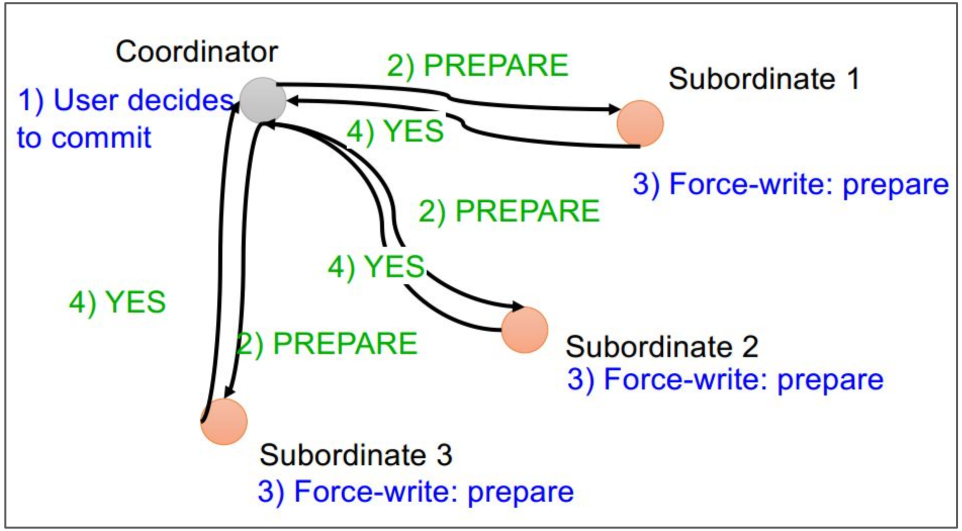
Goal: General purpose, distributed agreement on some action, with failures – Different entities play different roles in the action

The 2 phases

- **Prepare:** master asks if all nodes can commit to an action or not
- **Commit:** if all nodes respond yes during the prepare phase, the master tells all nodes to commit



2 PC with Global COMMIT



Coordinator

1) User decides to commit

2) PREPARE

4) YES

Subordinate 1

3) Force-write: prepare

2) PREPARE

4) YES

4) YES

2) PREPARE

Subordinate 2

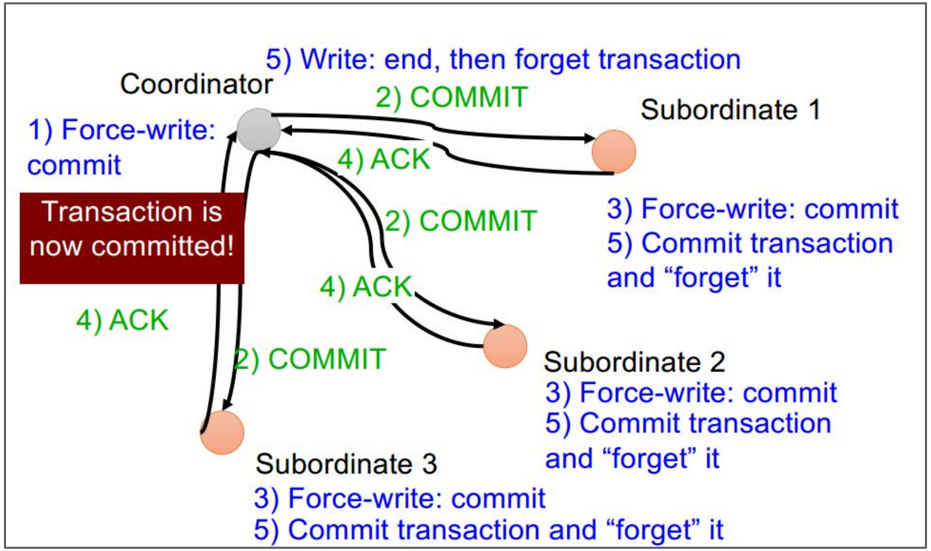
3) Force-write: prepare

Subordinate 3

3) Force-write: prepare

Phase 1: PREPARE/VOTING PHASE

- Collect votes from everyone



5) Write: end, then forget transaction

Coordinator

2) COMMIT

Subordinate 1

1) Force-write: commit

4) ACK

Transaction is now committed!

2) COMMIT

3) Force-write: commit
5) Commit transaction and “forget” it

4) ACK

4) ACK

2) COMMIT

Subordinate 2

3) Force-write: commit
5) Commit transaction and “forget” it

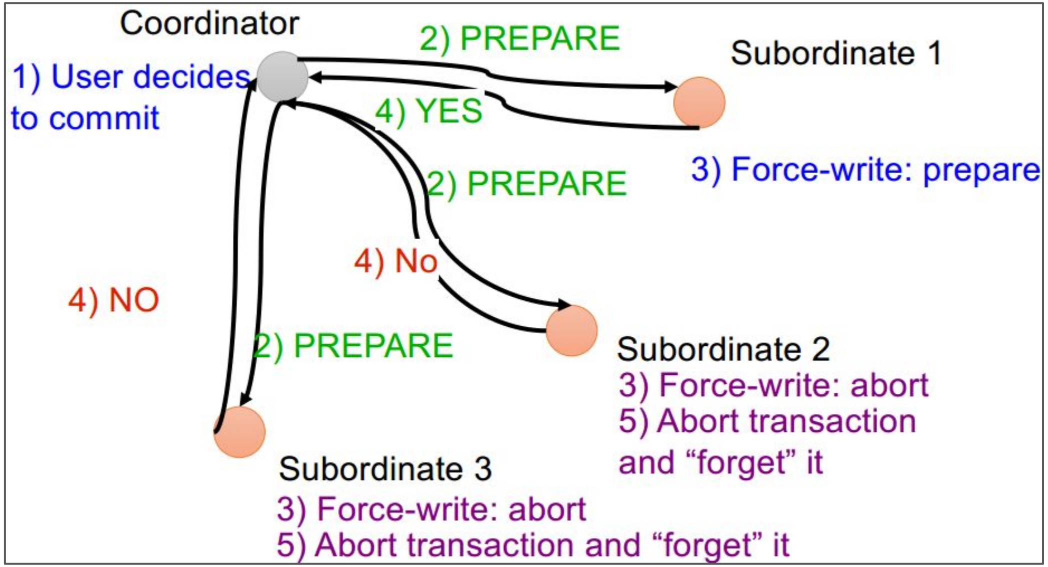
Subordinate 3

3) Force-write: commit
5) Commit transaction and “forget” it

Phase 2: COMMIT/ABORT phase

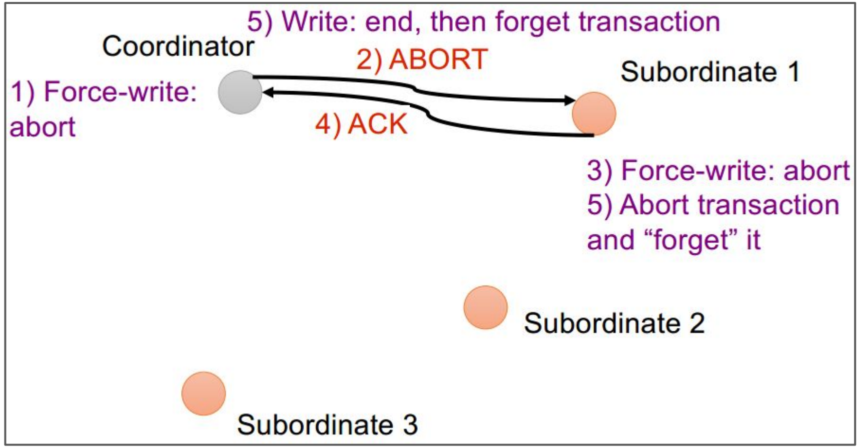
- Communicate decision to everyone based on the votes (in this example, decision is **COMMIT**)

2 PC with Global ABORT



Phase 1: PREPARE/VOTING PHASE

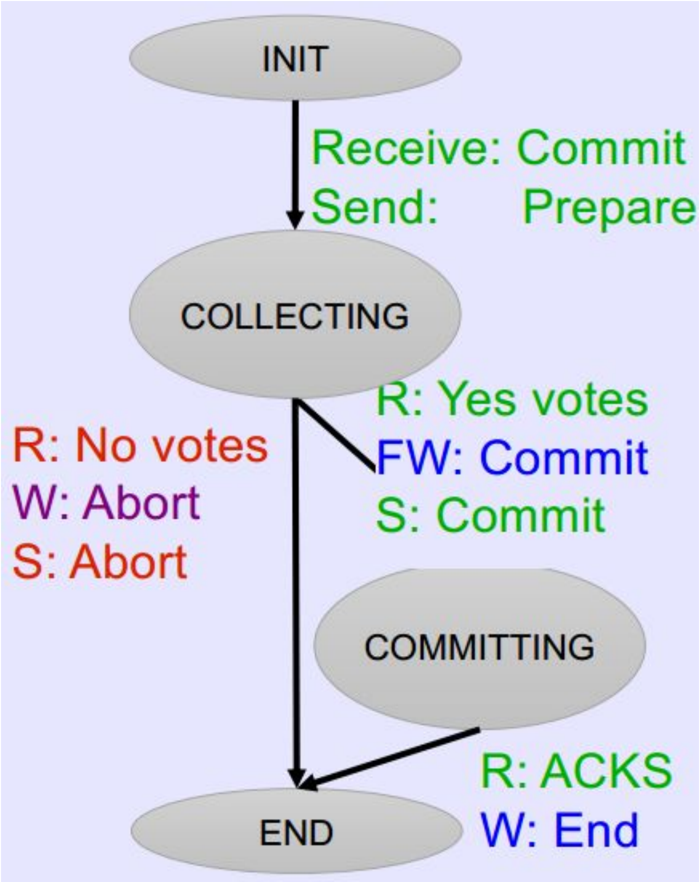
- Collect votes from everyone



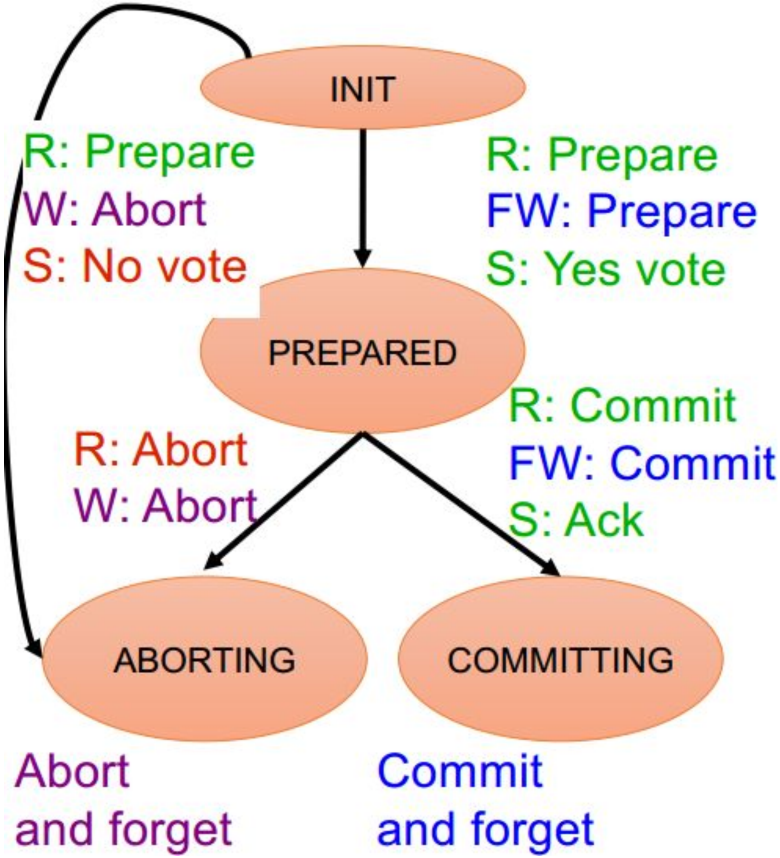
Phase 2: COMMIT/ABORT phase

- Communicate decision to everyone based on the votes (in this example, decision is **ABORT**)

STATE DIAGRAMS | Failure can happen after any state



Coordinator



Database Site Server

Transactions in our implementation

- Each transaction is composed of:
    - <PERSON NAME>: [List of items ordered by the person]
- Either orders get delivered to all people or to NONE AT ALL
- Do not allow a person to order same thing twice

Transaction 1: 1_anmol 4 5 9 0_shrey 4 5 0_pratyush 1 7	**Transaction 2:** 1_anmol 1 2 0_shrey 0 1 0_pratyush 1 6 1_gurkirat 3	**Transaction 3:** 1_anmol 1 2 0_pratyush 2 6
Happens	**Does not happen as pratyush already has item 1**	**Happens**

SETUP/STACK:

- All 3 sites coded in python
- Exchange messages via gRPC (Remote Procedure Call) sharing common data structures (with help of protocol buffers)
- FLAGS encoded in code to simulate CRASHES.

Till how long to wait for messages ?

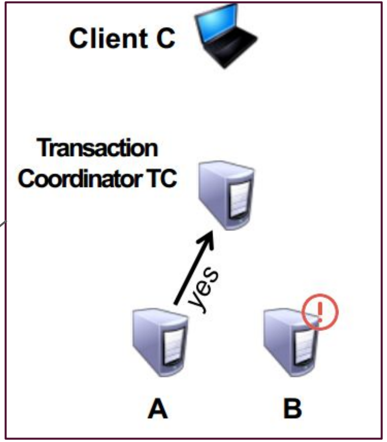
The wait for receiver may be infinite it the sender has crashed. Hence, put a **timeout**.

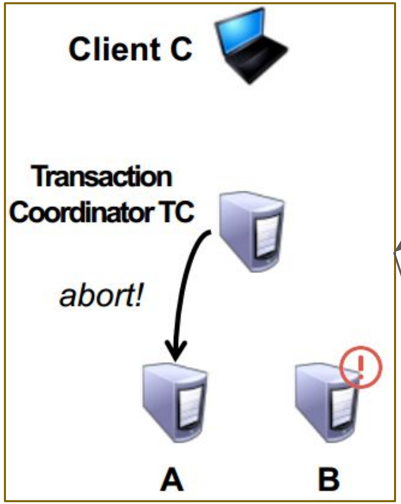
Example:

TC waits for “yes” or “no” from A and B

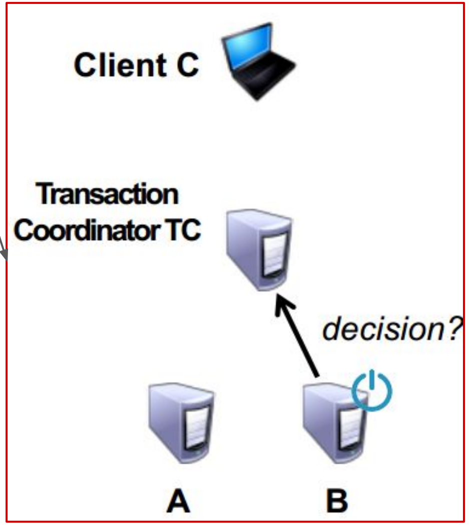
- TC hasn’t yet sent any commit messages, so can safely abort after a timeout
- But this is conservative: might be network problem
- We’ve preserved correctness, sacrificed performance

Node B crashes and only Node A responds





- TC waits on B until some timeout period but then aborts.



- B decided <YES>, crashed before it could send <YES> but logs indicate it has sent <YES>
- Queries coordinator.

DEMO

Extensive Proofs here:
https://docs.google.com/document/d/1QNwa9DeXqCmhqfwgDnK-Xg3-0jze2x8Av08yKp5T5e0/edit#

DEMO 1

- Which sites involved ?
  - Server Site 0 and 1 are involved
  - Client Test Case 1
- Who fails and when ?
  - Server 0 fails before sending local verdict to Coordinator
- How do the non-failed processes cope and infer verdicts?
  - Coordinator doesn’t verdicts from everyone and hence decides to Abort
  - Coordinator send GLOBAL ABORT to all active server sites
- How does the failed process recover ?
  - Wakes up and sees that it didn’t send local verdict and hence everyone would have aborted
  - Recovering Site Aborts
- What is the final verdict the transaction has ?
  - ABORT (Failure)

Environment variable to set: 1 at server_0

DEMO 2

- Which sites involved ?
  - Server Site 0 and 1 are involved
  - Client Test Case 1
- Who fails and when ?
  - Server 0 fails before receiving global verdict from Coordinator
- How do the non-failed processes cope and infer verdicts?
  - All other sites get global verdict and they act accordingly
- How does the failed process recover ?
  - Recovering Site asks Coordinator for verdict and after receiving acts on that verdict
- What is the final verdict the transaction has ?
  - COMMIT (Succes)

Environment variable to set: 2 at server_0

DEMO 3

- Which sites involved ?
  - Server 0 and 1
  - Client Test Case 1
- Who fails and when ?
  - Server 0 fails after receiving global verdict from Coordinator
- How do the non-failed processes cope and infer verdicts?
  - All other sites get global verdict and they act accordingly
- How does the failed process recover ?
  - Recovering Site has global verdict in its log and just reads log and acts accordingly
- What is the final verdict the transaction has ?
  - COMMIT (Succes)

Environment variable to set: 3 at server_0

DEMO 4

- Which sites involved ?
  - Server 0 and 1
  - Client Test Case 1
- Who fails and when ?
  - Server 0 fails before receiving PREP from Coordinator
- How do the non-failed processes cope and infer verdicts?
  - As some site failed before sending PREP, Coordinator infers ABORT and send ABORT to all remaining active sites
- How does the failed process recover ?
  - Recovering site sees that it didn’t send local verdict to Coordinator and hence can abort
- What is the final verdict the transaction has ?
  - ABORT (Failure)

Environment variable to set: 7 at server_0

Coordinator Failure

DEMO 5

- Which sites involved ?
  - Server 0, 1 and 2
  - Client Test Case 5
- Who fails and when ?
  - Coordinator fails after receiving local verdicts from all sites
- How do the non-failed processes cope and infer verdicts?
  - All Sites have ready T (Local Commit) and hence wait for Coordinator to come online and tell decision
- How does the failed process recover ?
  - Coordinator recovers and sees no global verdict has been set by it and hence decides to abort.
  - GLOBAL ABORT is also conveyed to other sites
- What is the final verdict the transaction has ?
  - ABORT (Failure)

Environment variable to set: 9 at txn_coord

DEMO 6

- Which sites involved ?
  - Server 0, 1 and 2
  - Client Test Case 5
- Who fails and when ?
  - Coordinator fails after sending global verdicts to a few sites
- How do the non-failed processes cope and infer verdicts?
  - Some processes have global verdict so they convey that verdict to other server sites and act on the global verdict
- How does the failed process recover ?
  - Coordinator wakes up and and sees global verdict so it just conveys success to client after all servers finish
- What is the final verdict the transaction has ?
  - COMMIT (Success)

Environment variable to set: 10 at txn_coord

DEMO 7

- Which sites involved ?
  - Server 0, 1 and 2
  - Client Test Case 5
- Who fails and when ?
  - Coordinator fails after sending prepare to a few sites
- How do the non-failed processes cope and infer verdicts?
  - Some processes didn’t receive PREP so they know that coordinator couldn’t have decided COMMIT.
  - Hence, they all ABORT
- How does the failed process recover ?
  - Coordinator sees that it didn’t take any global decision and hence decides to ABORT
- What is the final verdict the transaction has ?
  - ABORT (Failure)

Environment variable to set: 11 at txn_coord

DEMO 8

- Which sites involved ?
  - Server 0, 1 and 2
  - Client Test Case 5
- Who fails and when ?
  - Coordinator fails before sending global verdicts to any site
- How do the non-failed processes cope and infer verdicts?
  - All servers have LOCAL_COMMIT and hence they wait for the coordinator to recover
  - On recovery of the coordinator, they get global verdict and act on it
- How does the failed process recover ?
  - Coordinator recovers and active sites get global decision
- What is the final verdict the transaction has ?
  - COMMIT (Success)

Environment variable to set: 12 at txn_coord