Two Phase Commit (2PC)

Institute of Parallel and Distributed Systems

Rong Chen

*Some slides adapted from Jinyang Li

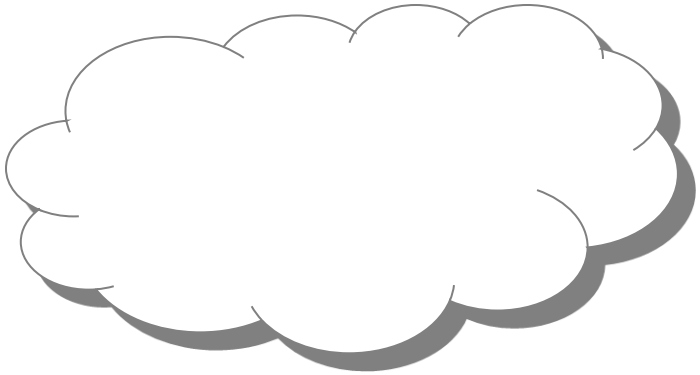
What we've learnt so far

- Sequential consistency
  - All nodes agree on a total order of ops on a single object
- Crash recovery
  - An operation writing to many objects is atomic w.r.t. failures
- Concurrency control
  - Serializability of multi-object operations (transactions)
  - 2-phase-locking, snapshot isolation
- This class:
  - Atomicity and concurrency control across multiple nodes

Example

client

Transfer $1000
From A:$3000
To B:$2000



Bank A

Bank B

- Clients desire:
  1. Atomicity: transfer either happens or not at all
  2. Concurrency control: maintain serializability

Strawman solution

client

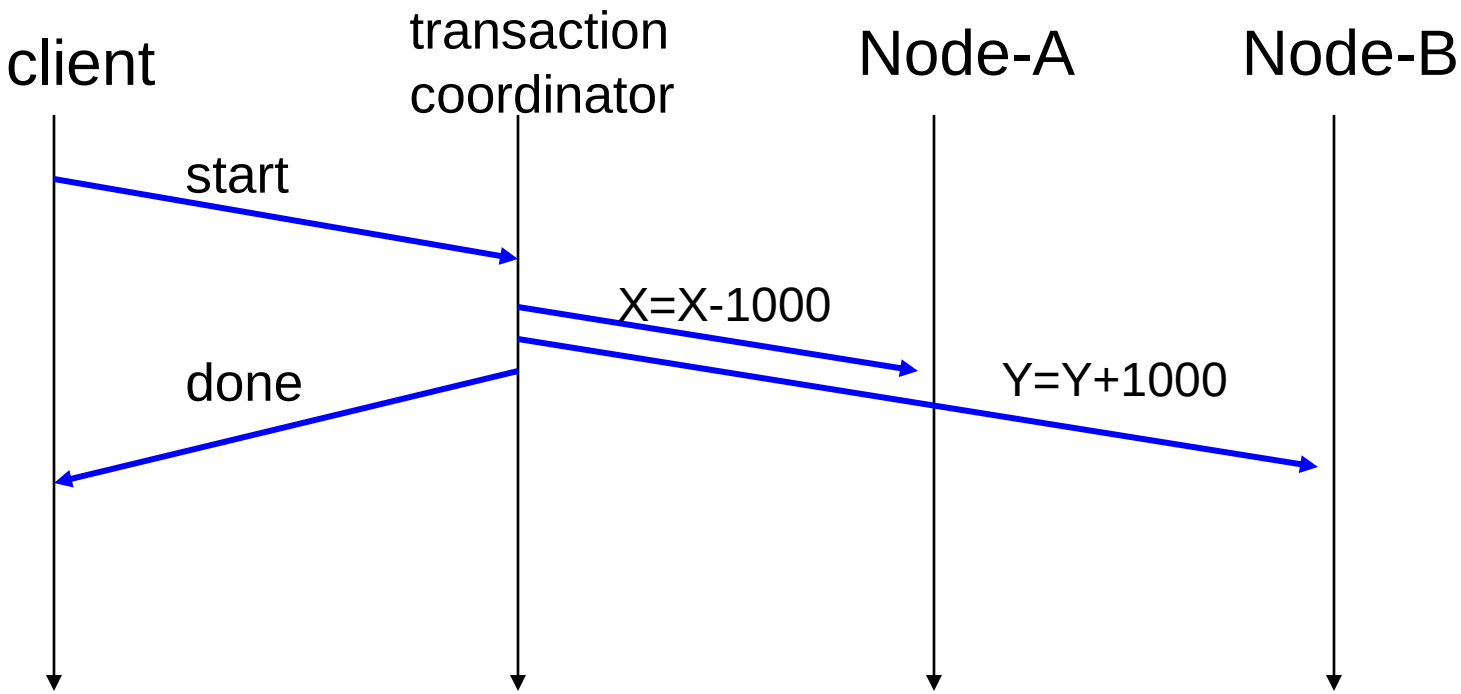
Transfer $1000
From X:$3000
To Y:$2000

Transaction coordinator

Node A

Node B

Strawman solution

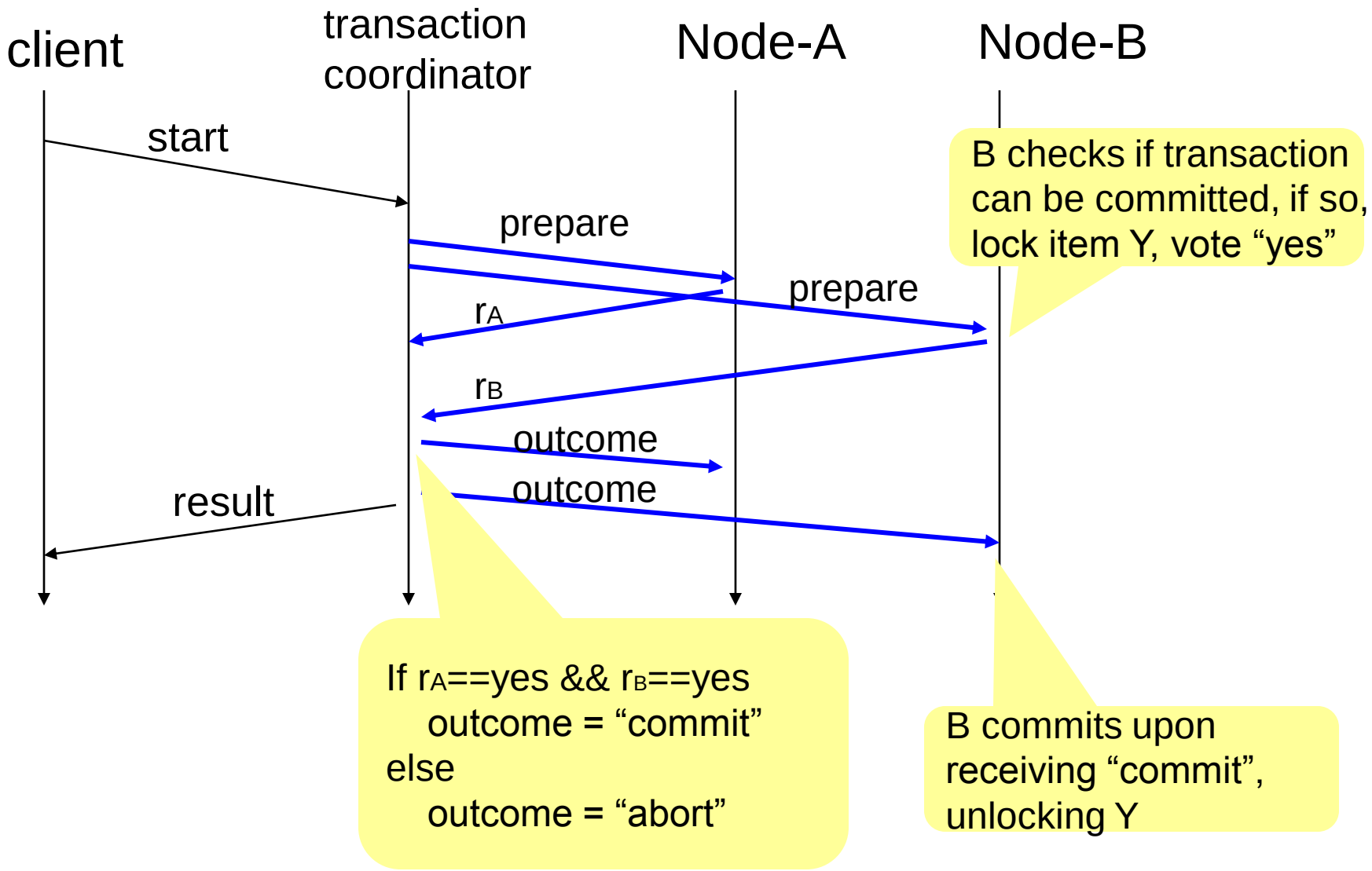


- What can go wrong?
  - X does not have enough money
  - Node B has crashed
  - Coordinator crashes
  - Some other client is reading or writing to X or Y

Reasoning about correctness

- TC, A, B each has a notion of committing
- Correctness:
  - If one commits, no one aborts
  - If one aborts, no one commits
- Performance:
  - If no failures, A and B can commit, then commit
  - If failures happen, find out outcome soon

Correctness first



Performance Issues

- What about timeouts?
  - TC times out waiting for A's response
  - A times out waiting for TC's outcome message
- What about reboots?
  - How does a participant clean up?

Handling timeout on A/B

- TC times out waiting for A (or B)'s "yes/no" response
  - Can TC unilaterally decide to commit?
  - Can TC unilaterally decide to abort?

Handling timeout on TC

- If B responded with “no” …
  - Can it unilaterally abort?
- If B responded with “yes” …
  - Can it unilaterally abort?
  - Can it unilaterally commit?

Possible termination protocol

- Execute termination protocol if B times out on TC and has voted “yes”
- B sends “status” message to A
  - If A has received “commit”/”abort” from TC …
  - If A has not responded to TC, …
  - If A has responded with “no”, …
  - If A has responded with “yes”, …

Resolves most failure cases except sometimes when TC fails

Handling crash and reboot

- Nodes cannot back out if commit is decided
- TC crashes just after deciding “commit”
  - Cannot forget about its decision after reboot
- A/B crashes after sending “yes”
  - Cannot forget about their response after reboot

Handling crash and reboot

- All nodes must log protocol progress
- What and when does TC log to disk?
  - Before send “Commit”
- What and when does A/B log to disk?
  - Before send “Yes”

Recovery upon reboot

- If TC finds no “commit” on disk
  - abort
- If TC finds “commit”
  - commit
- If A/B finds no “yes” on disk
  - abort
- If A/B finds “yes”
  - run termination protocol to decide

Summary: two-phase commit

1. All nodes that decide reach the same decision
2. No commit unless everyone says "yes".
3. No failures and all "yes", then commit.
4. If failures, then repair, wait long enough for recovery, then some decision.

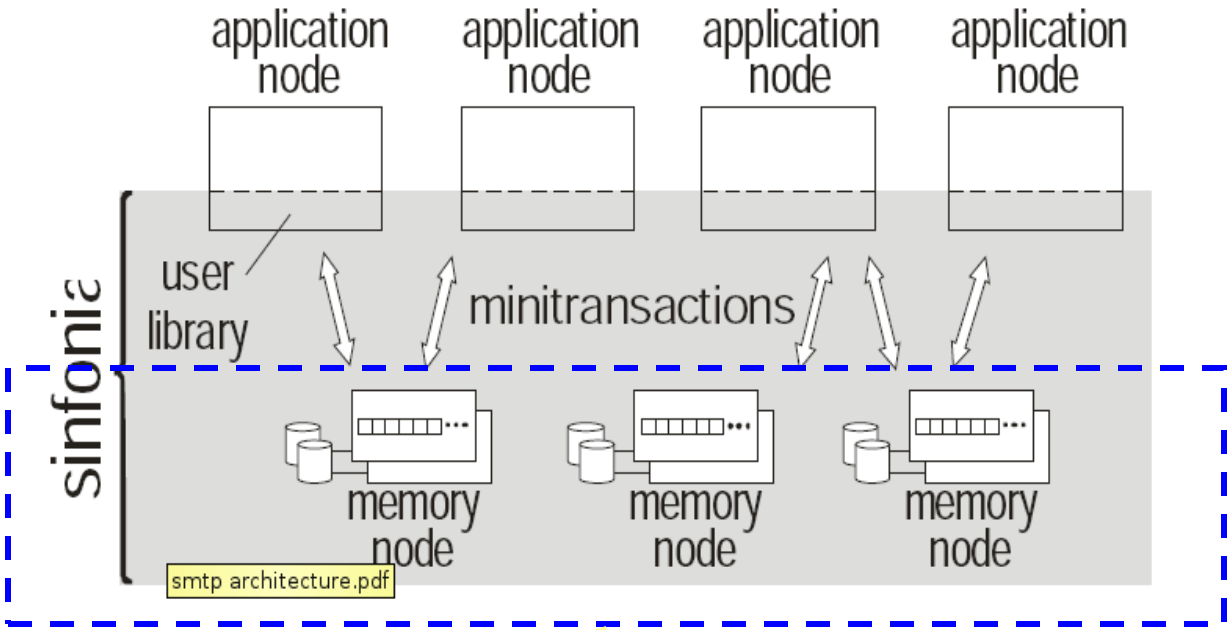
A Case study of 2P commit in real systems

Sinfonia (SOSP'07)

What problem is Sinfonia addressing?

- Targeted uses
  - Systems or infrastructural apps within a data center
- Sinfonia: a shared data service
  - Span multiple nodes
  - Replicated with consistency guarantees
- Goal: reduce development efforts for system programmers

Sinfonia architecture

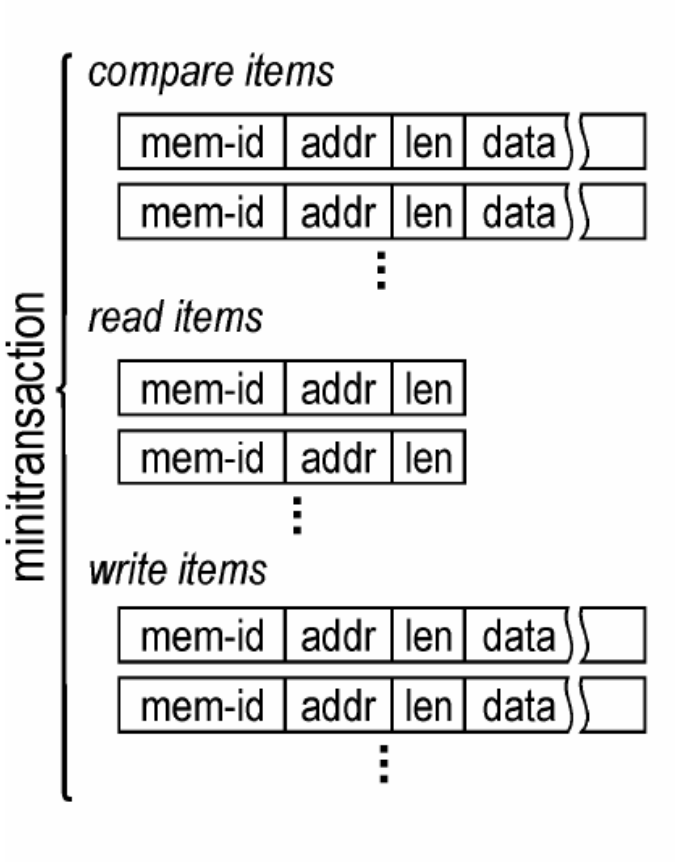


Each memory node provides a shared address space with name (node-id, address)

Sinfonia mini-transactions

- Provide atomicity and concurrency control
- Trade off expressiveness for efficiency
  - Fewer network round trips to execute
  - Less flexible, general-purpose than traditional transactions
- Result
  - a lightweight, short-lived type of transaction
  - over unstructured data

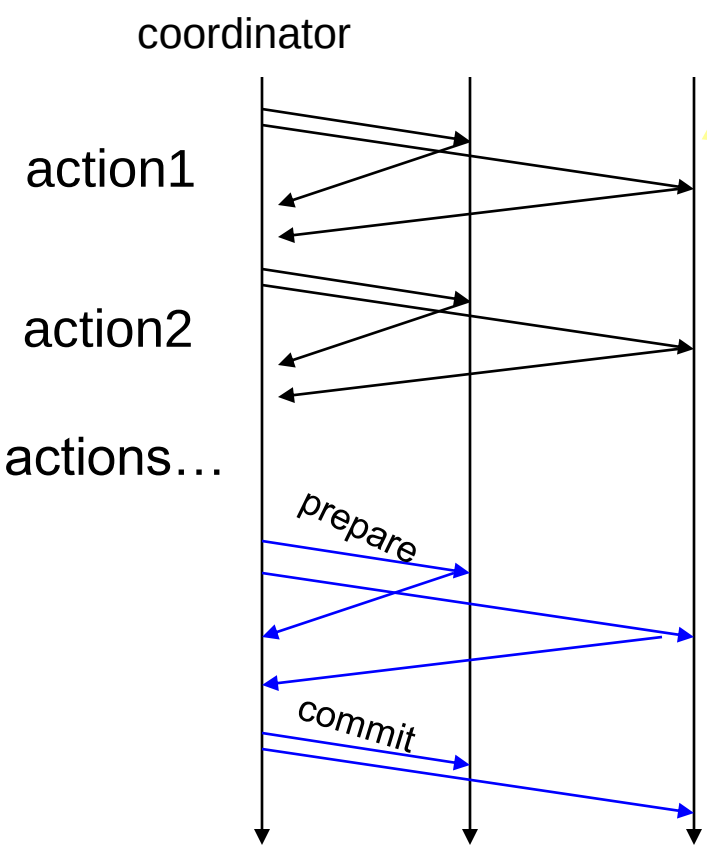
Mini-transaction details



- Mini-transaction
  - Check *compare* items
  - If all match
    - retrieve data in *read* items
    - modify data in *write* items
- Example:

```
t = new Minitransaction()
t->cmp(node-X:0x000, 4, 3000)
t->cmp(node-Y:0x100, 4, 2000)
t->write(node-X:0x000, 4, 2000)
t->write(node-Y:0x100, 4, 3000)
Status = t->exec_and_commit()
```

Sinfonia uses 2P commit



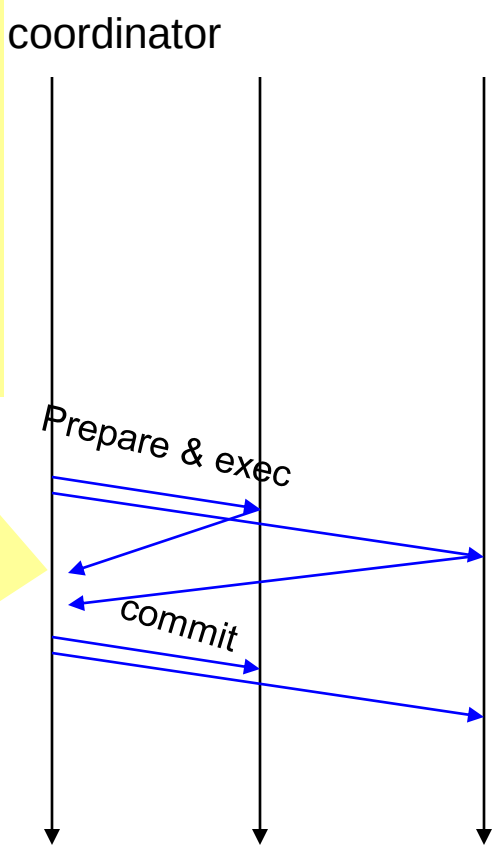
Traditional transactions

Traditional transactions:
general but expensive

```
BEGIN tx
If (a > 0 && b== 0)
   b = a * a
for (i = 0; i < a; i++)
   b += i
END tx
```

Mini-transaction:
less general but efficient

```
BEGIN tx
If (a == 3000 && b==2000)
{
   a=2000
   b=3000
}
END tx
```



Mini-transactions

Potential uses of mini-transactions

1. atomic swap operation
2. atomic read of many data
3. try to acquire a lease
4. try to acquire multiple leases atomically
5. change data if lease is held
6. validate cache then change data

Sinfonia's 2P protocol

- Transaction coordinator is at application node instead of memory node
  - Saves one RTT
- Problems: crashed TC blocks transaction progress
  - App nodes are less reliable than memory nodes

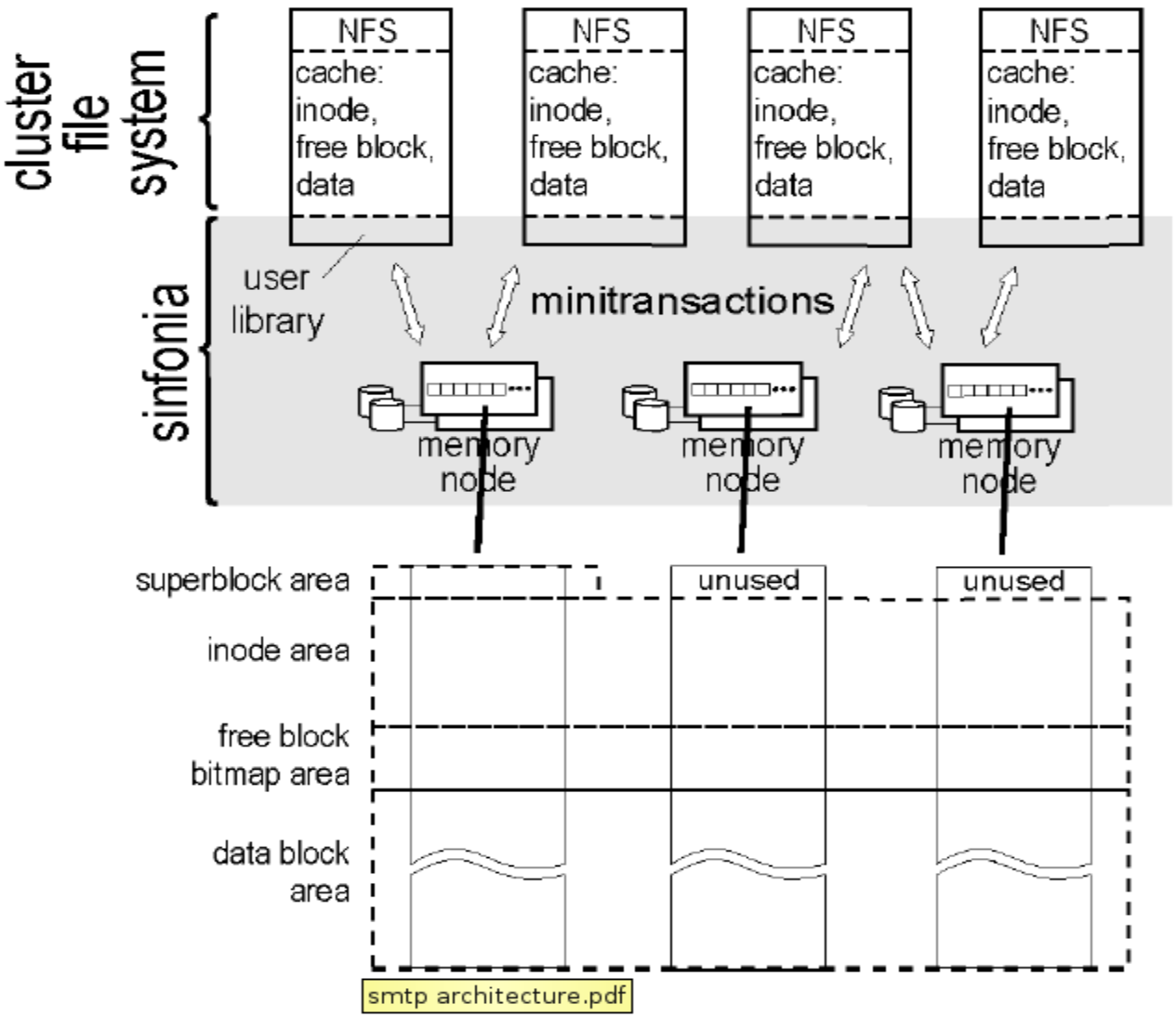
Sinfonia's 2P protocol

- TC keeps no log
- A transaction is committed iff all participants have "yes" in their logs
- Recovery coordinator cleans up
  - Ask all participants for existing vote (or vote "no" if not voted yet)
  - Commit iff all vote "yes"
- Transaction blocks if a memory node crashes
  - Must wait for memory node to recovery from disk

Sinfonia applications

- SinfoniaFS
  - hosts share the same set of files, files stored in Sinfonia
  - scalable: performance improves with more memory nodes
  - fault tolerant
- SinfoniaFS exports a NFS interface
  - Each NFS op corresponds to 1 mini-transaction

SinfoniaFS architecture



Example use of mini-transaction

```
setattr(ino_t inum, sattr_t newattr) {
  do {
       addr = address of inode
       curr_version = inode->version
       t = new Minitransaction;
       t->cmp(addr, 4, curr_version)
       t->write(addr, 4, curr_version+1)
       t->write(addr, 20, newattr);
  }while (t->status == fail);
}
```

General use of mini-transaction in SinfoniaFS

1. If local cache is empty, load it
2. Make modifications to local cache
3. Issue a mini-transaction to check the validity of cache, apply modification
4. If mini-transaction fails, reload cached item and try again

More examples: append to file

- Find a free block in cached freemap
- Issue mini-transaction with
  - Compare items: cached inode, free status of the block
  - Write items: inode, append new block, freemap, new block
- If mini-transaction fails, reload cache

Next Classes

- Fault Tolerance
  - Paxos

Thank you !

Any questions?