Don't Wait For Sync To Achieve Strong Consistency!

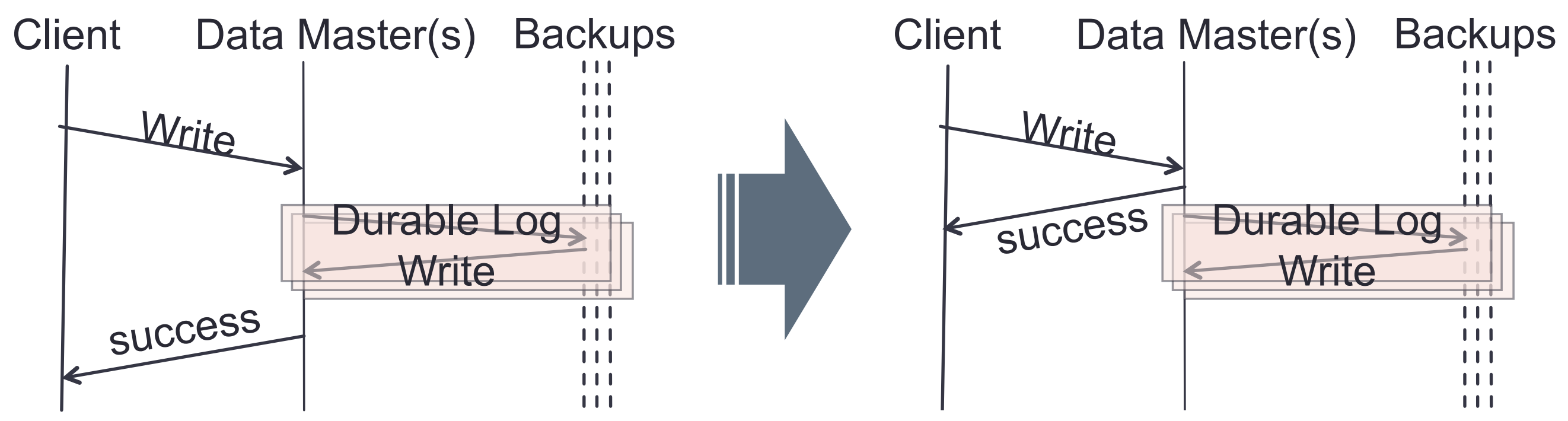
Seo Jin Park and John Ousterhout

OVERVIEW

Deterministic updates **durable** and **consistent** with **asynchronous** backup

- Server returns to clients before making updates durable.
- Use "retries" of RPCs to recover from server crash.
- Two ways to recover from crash.
  - Client-driven retries.
  - 3rd party (witness) driven retries.
- Better Performance
  - Write latency: 15 µs → 7.5 µs
  - Better throughput: 3x server throughput

LET'S BACKUP ASYNC



asynchrony = weak consistency??

State of art: Consistency **OR** Performance

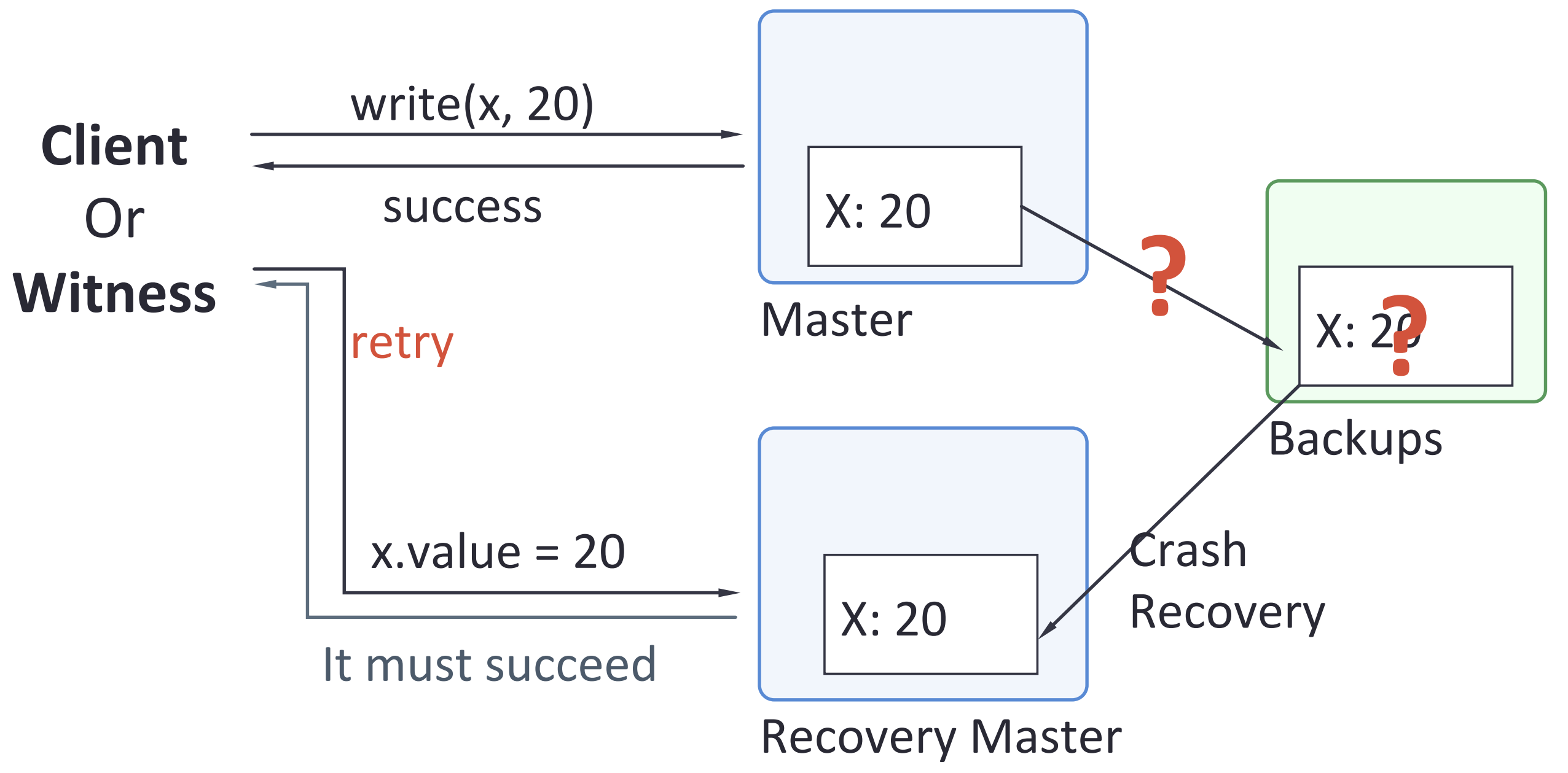
- Async replication + eventual consistency (eg. TAO, Redis)
- Sync replication + strong consistency (eg. RAMCloud)

*We can get: **Performace & Strong Consistency & simplicity of Primary-Backup***

CHALLENGES IN RECOVER BY RETRY

Issue 1. Retry may re-execute

- If a server crash, an update may or may not be recovered
- **RIFL** (Reusable Infrastructure for Linearizability) [SOSP15] **will let server ignore retries for already completed updates**



Issue 2. Out-of-order retries

- Allow 1 not-replicated update per key (overwrites wait for sync of previous value)
  - Pro: Any deterministic operations can be recovered by retries.
  - Con: continuously overwritten object can be slow.

CLIENT-DRIVEN RECOVERY

New Consistency Model: consistency w/o durability

1. **All reads are consistent**
   - Reads are <u>blocked</u> until data become durable
2. **Client written data should never experience anomaly as long as it is alive.**
   - When a server crashes, client retries previously returned writes.

- **Write (last two) is lost only if <u>both client and server</u> crash**
- **Client may <u>wait for durability</u> before <u>externalization</u>**
- **Conditional write is still consistent**
- **Non-durable atomic (ACI) transaction**

Example

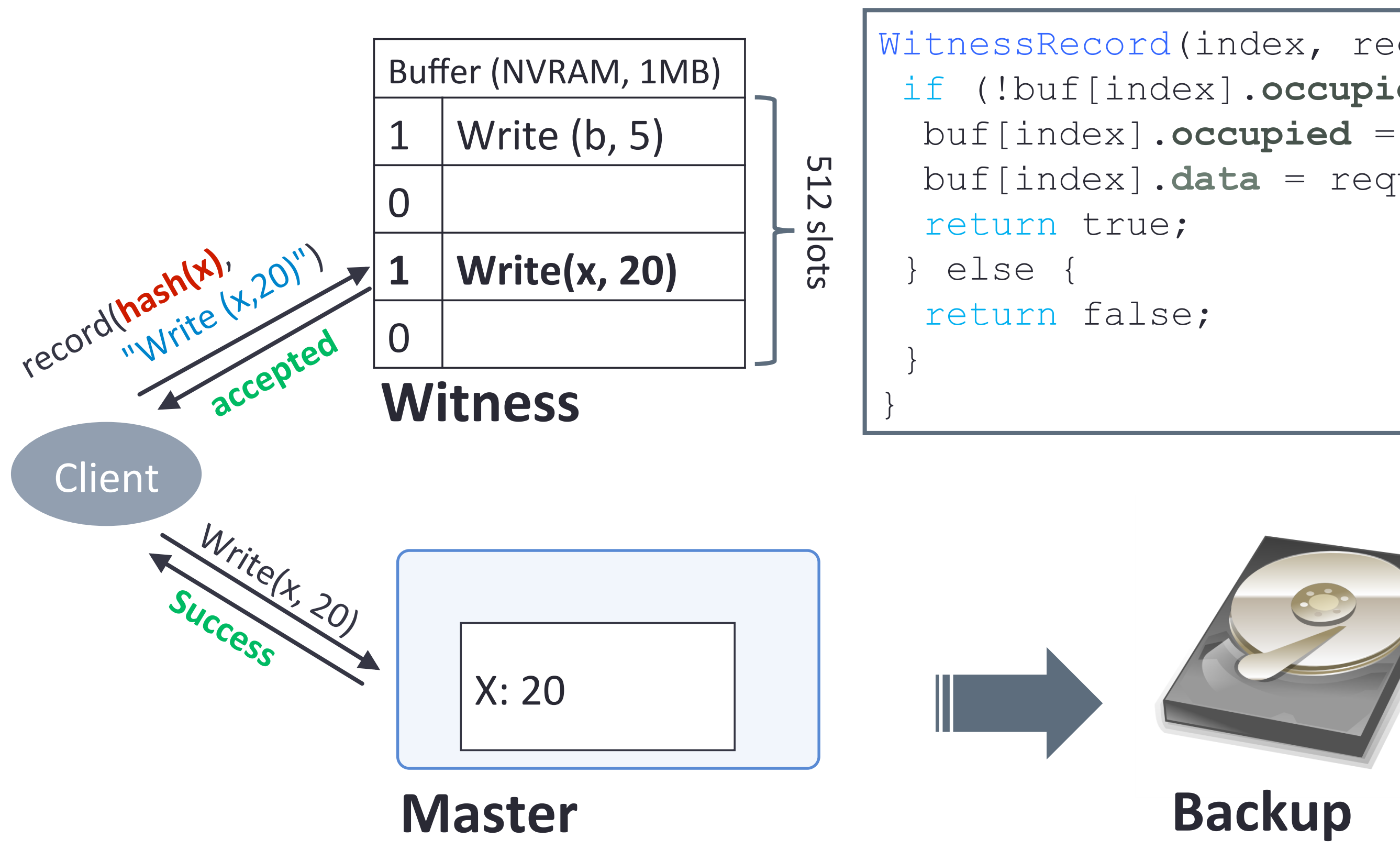
```
ramcloud.write(1, “Bob”, “2”);
ramcloud.write(1, “Bill”, “2”);
ramcloud.sync();
printf(“Updated Bob and Bill”);
```

Applications

- Things don't care durability
- Split of update / validate client
- Many updates before externalization.

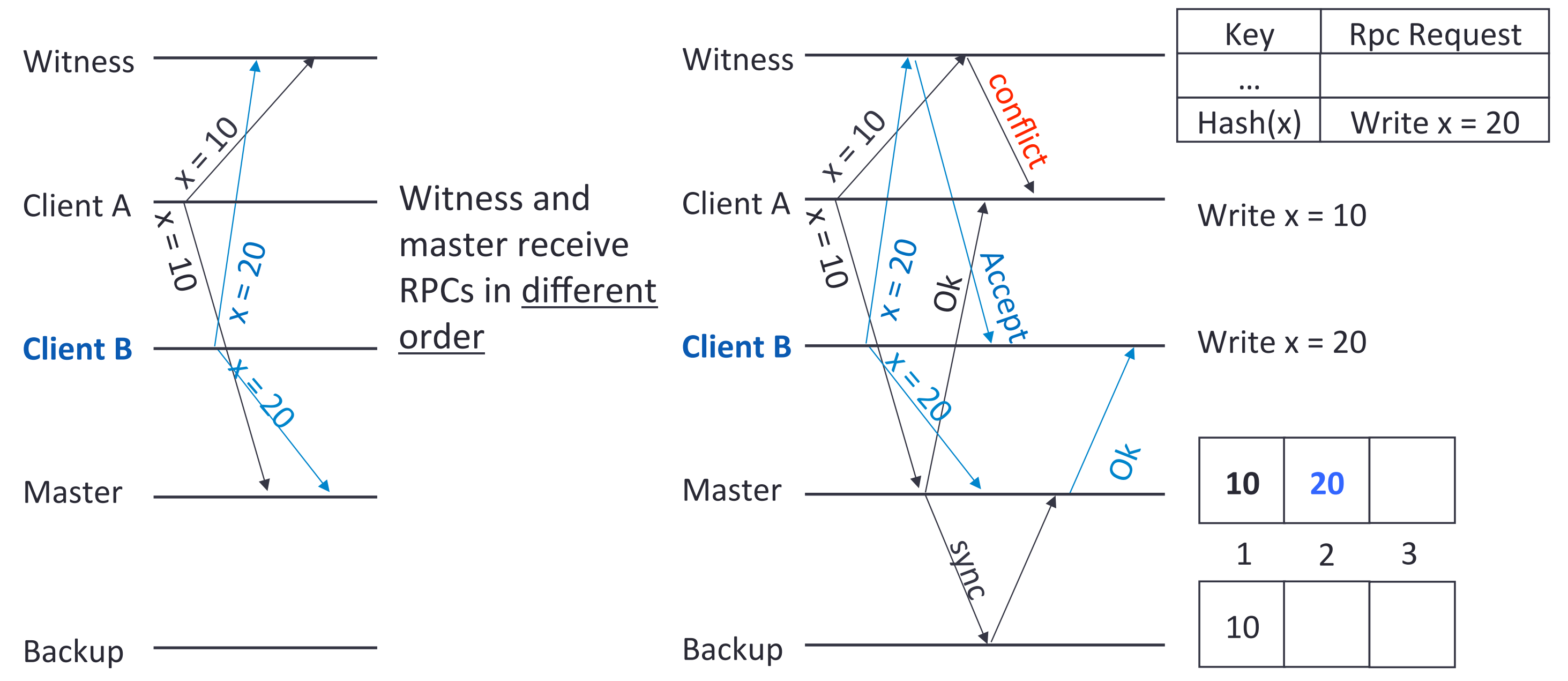
Don't guarantee invariants. App still need to clean up for crashed clients.

3RD PARTY-DRIVEN RECOVERY



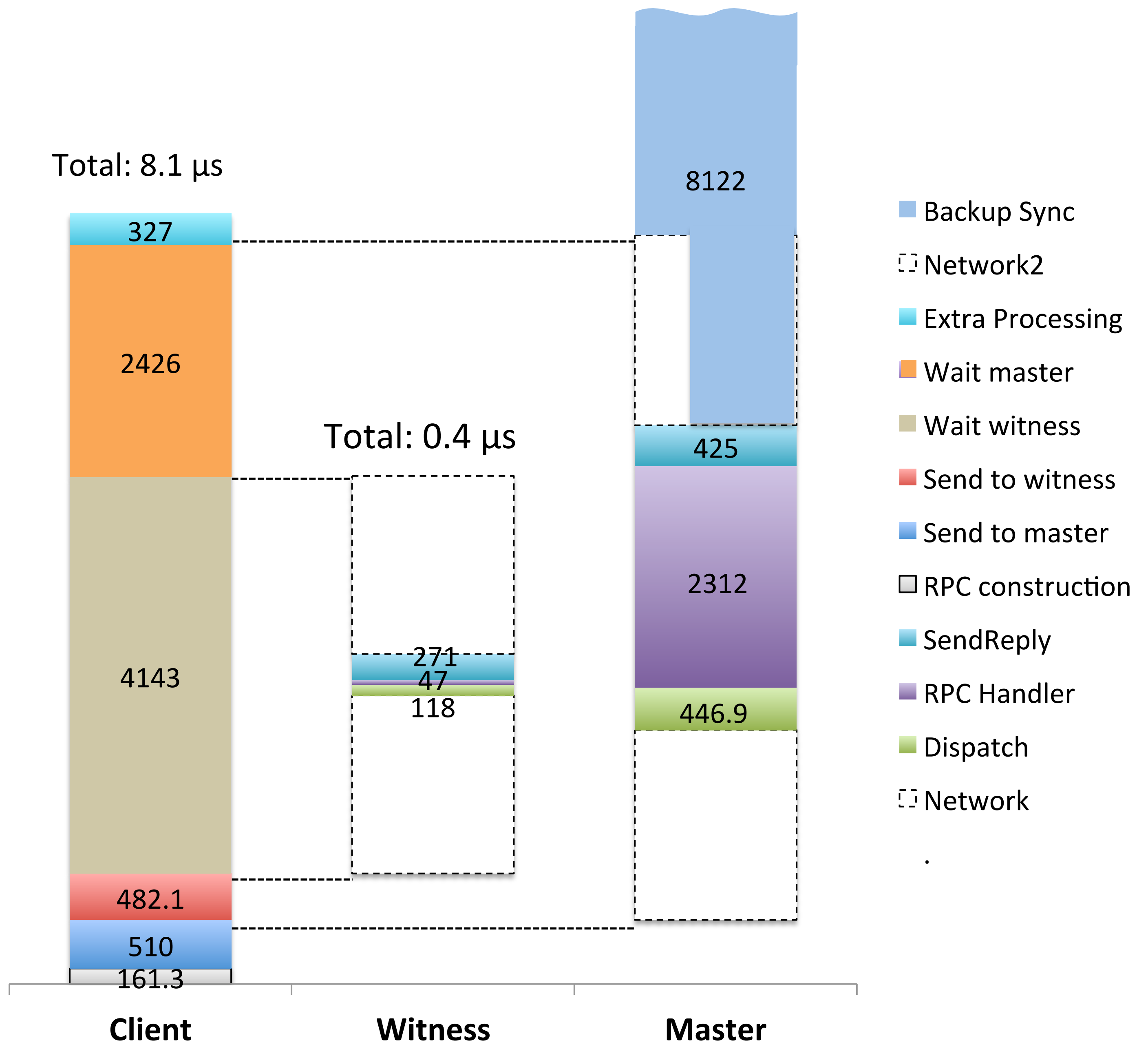
```
WitnessRecord(index, request){
 if (!buf[index].occupied) {
  buf[index].occupied = true;
  buf[index].data = request;
  return true;
 } else {
  return false;
 }
}
```

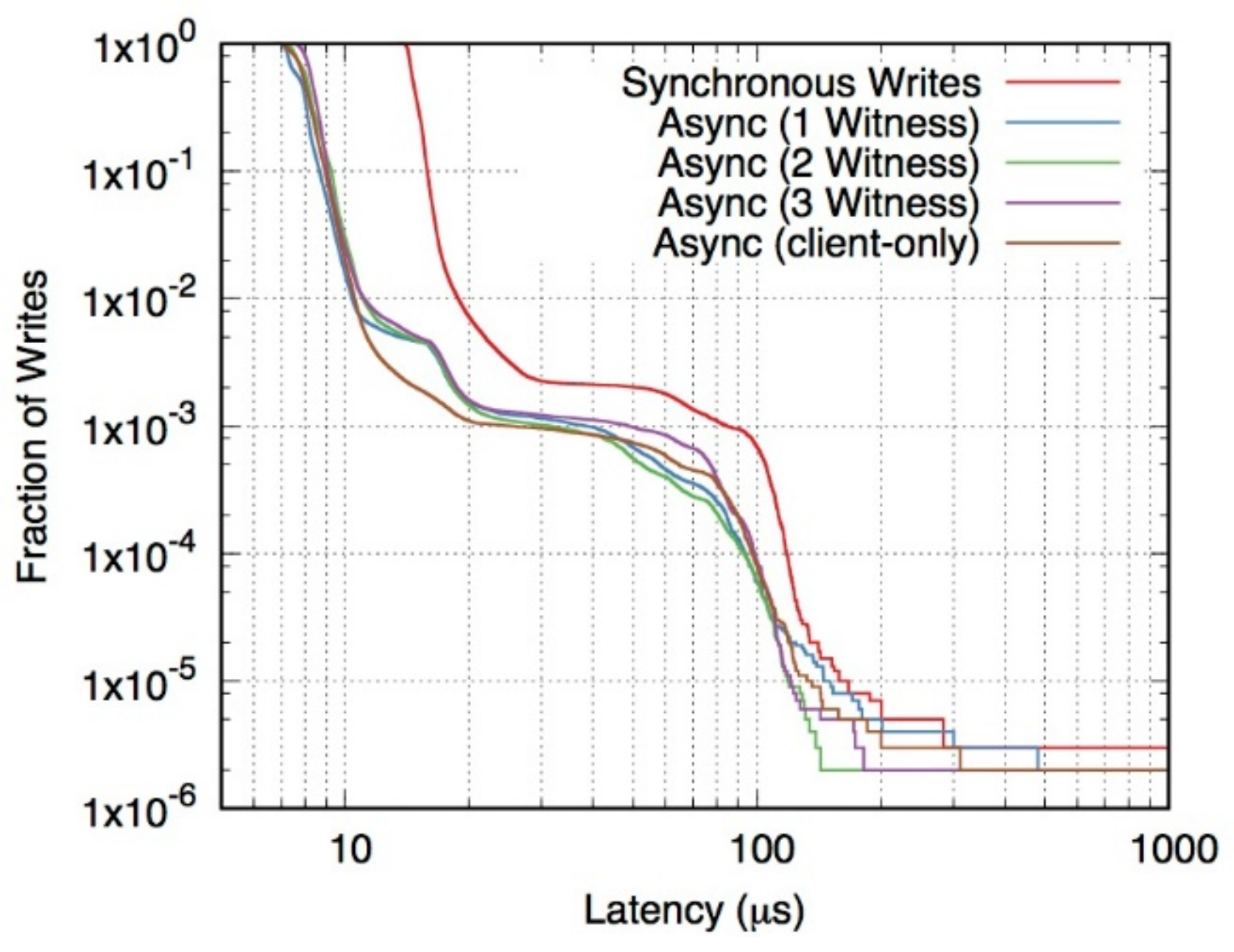
Witness detects out-of-order recording and reject.

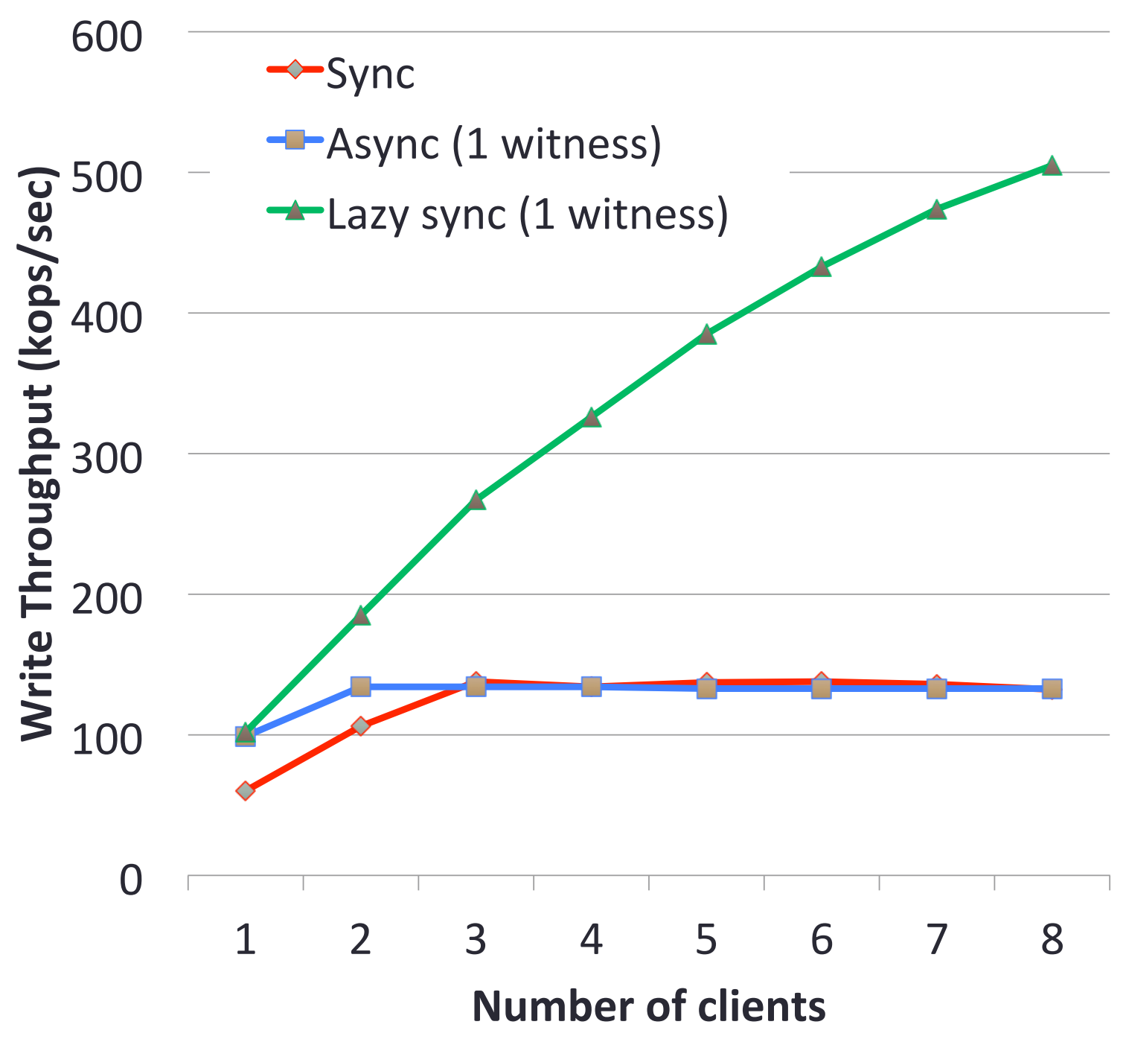


Key	Rpc Request
…	
Hash(x)	Write x = 20

FASTER + HIGHER THROUGHPUT







CONCLUSIONS

- Retry RPC requests if a server crashes → strong consistency & durability (for witness) without synchronous backup.
- Decoupling durability process from critical path improved performance
  - Lower latency: 7.5 µs regardless of backup media
  - Higher throughput: 137kops → 500 kops
- RIFL (Reusable Infrastructure for Linearizability) eases design and reasoning of consistency