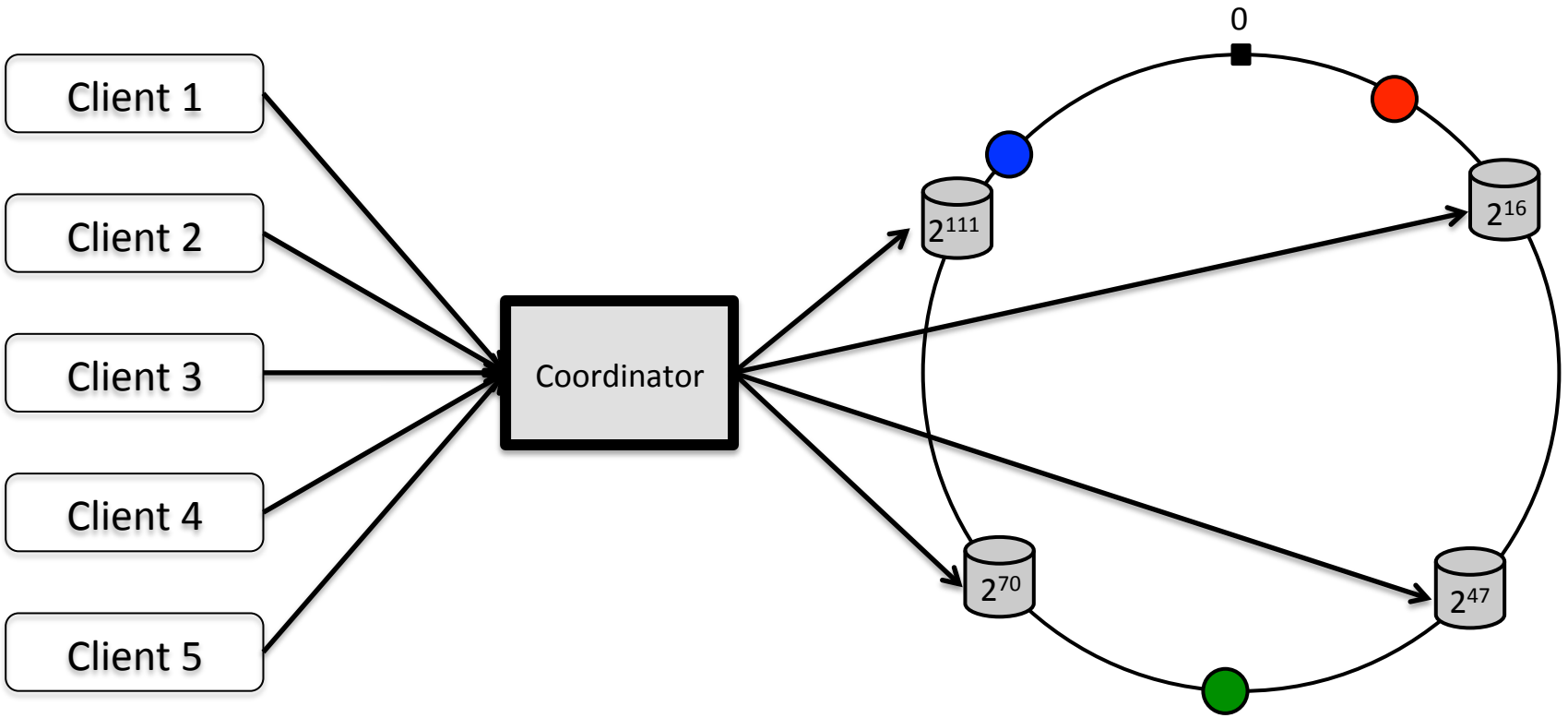
# CS162 Section

Lecture 11

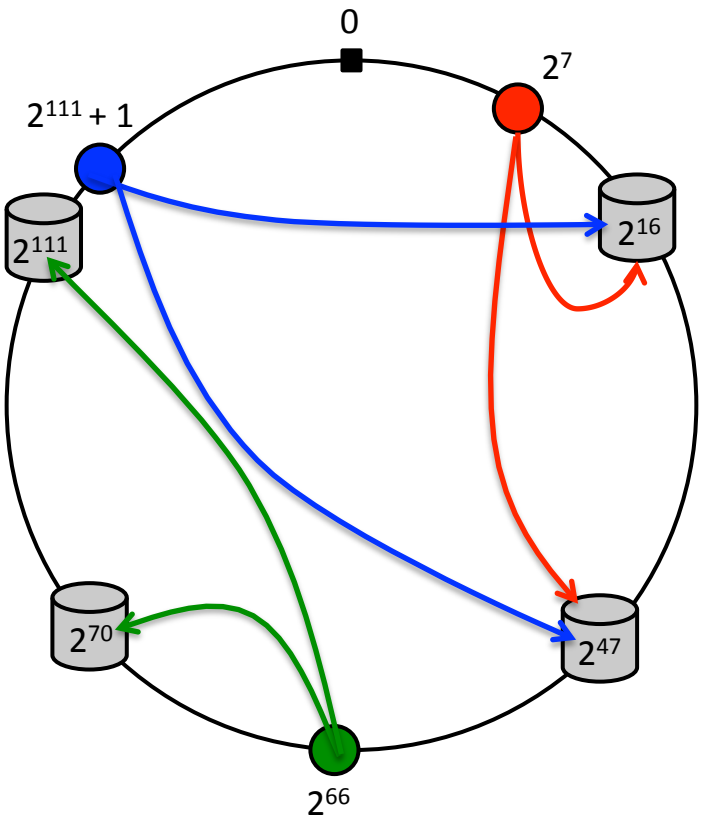
# Project 4

- Implement a distributed key-value store that uses
  - Two-Phase Commit for atomic operations,
  - Replication for performance and fault-tolerance,
  - Encryption for security

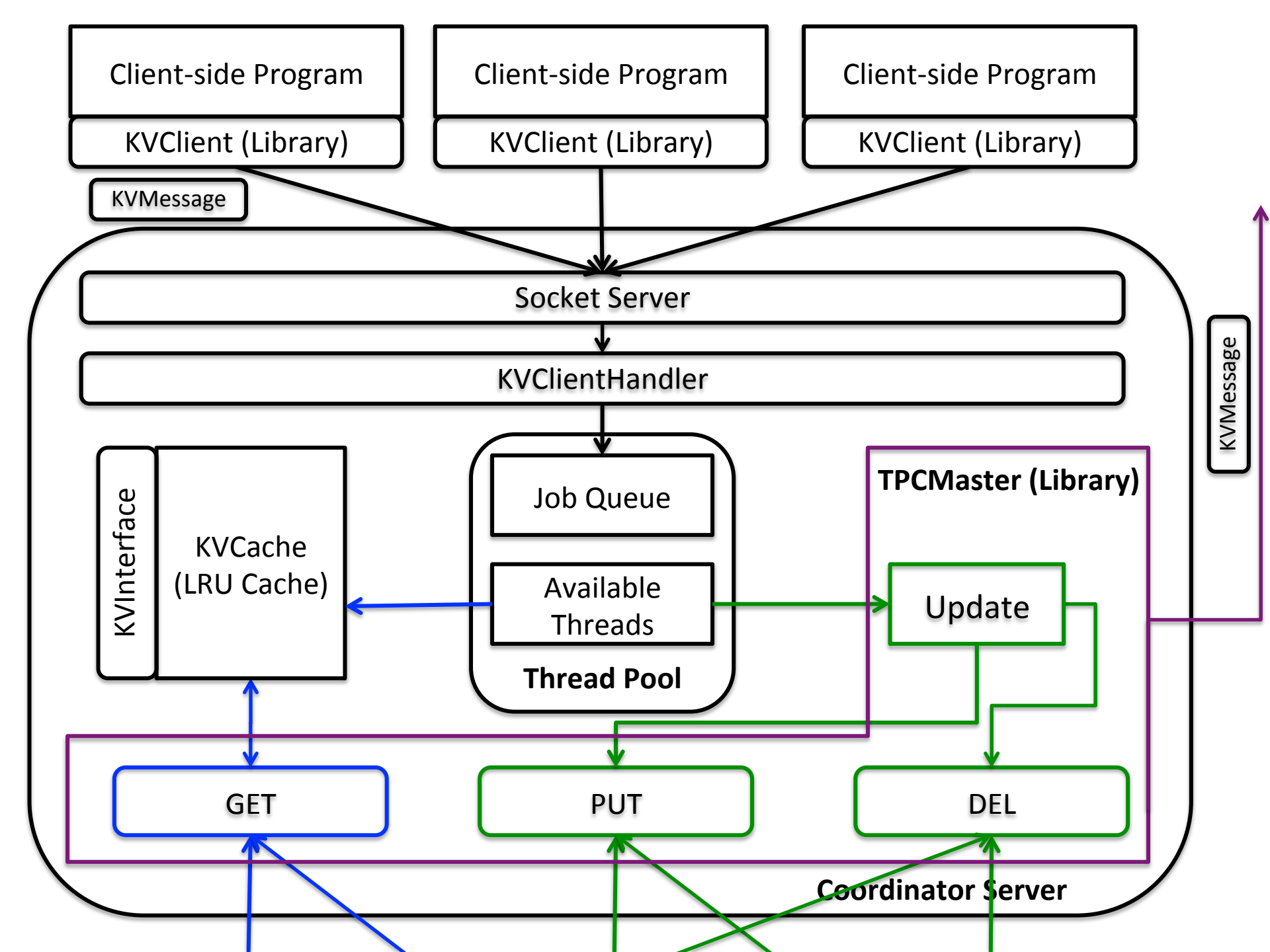
# Distributed Key-Value Store

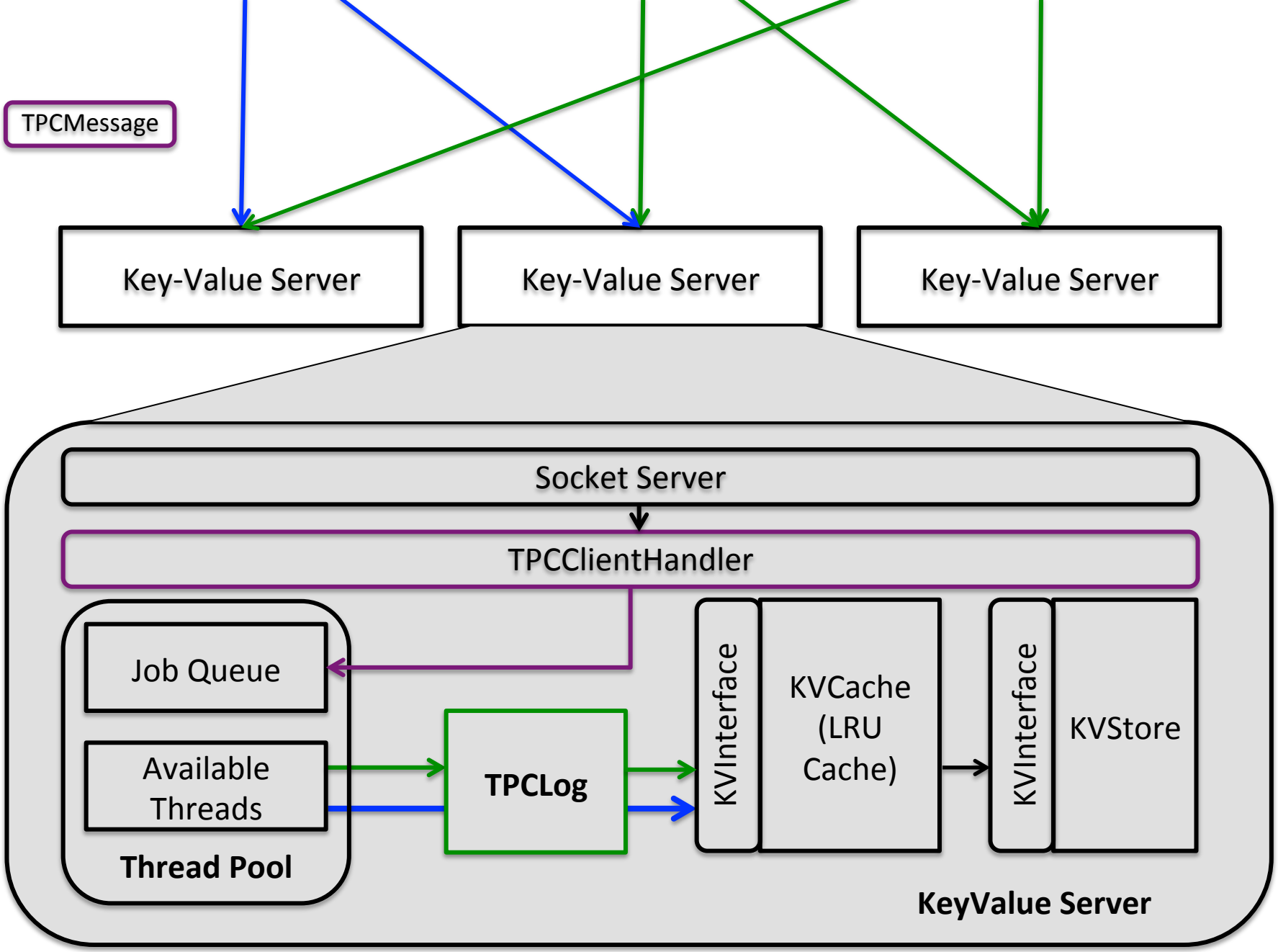


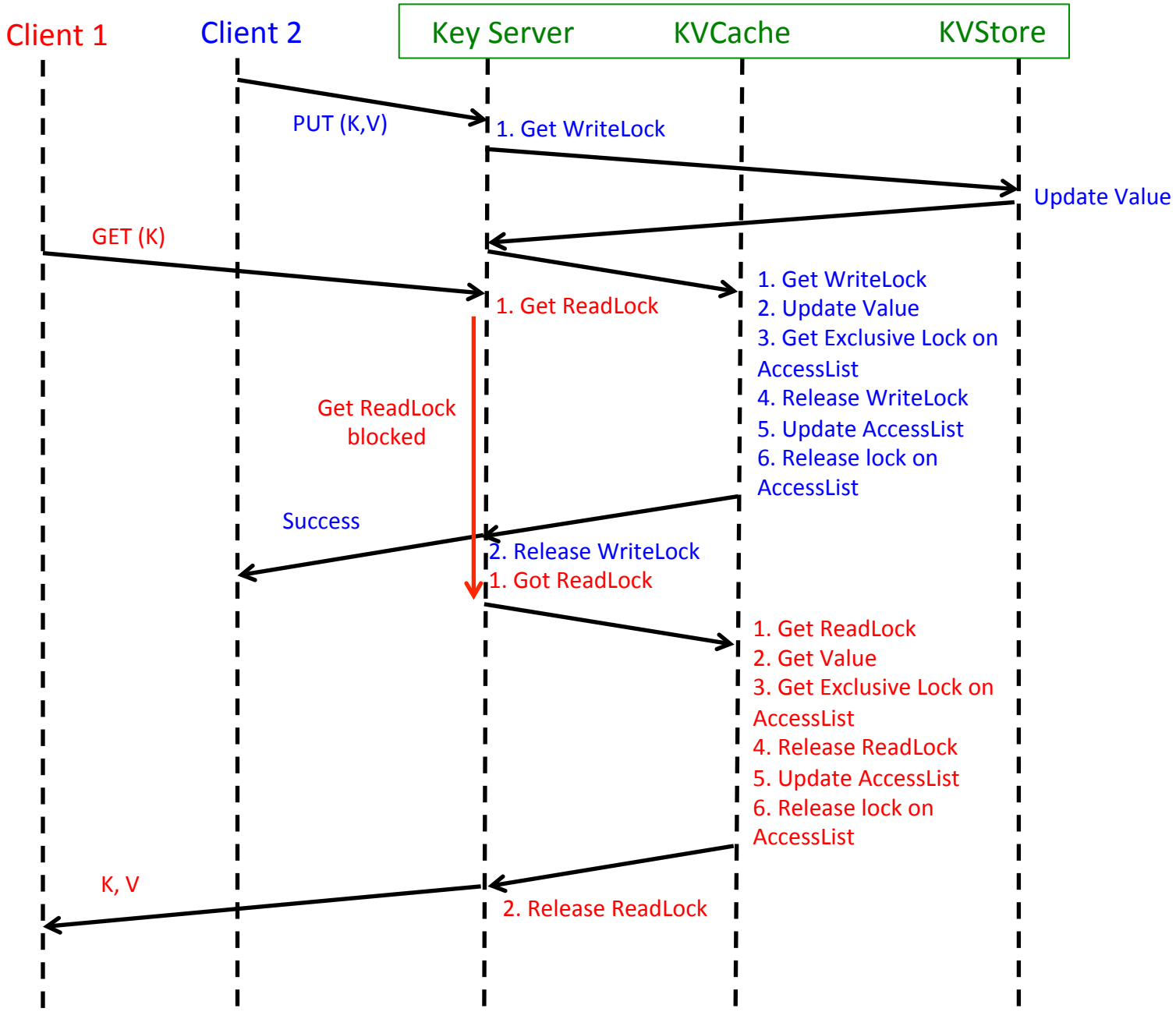
# Consistent Hashing

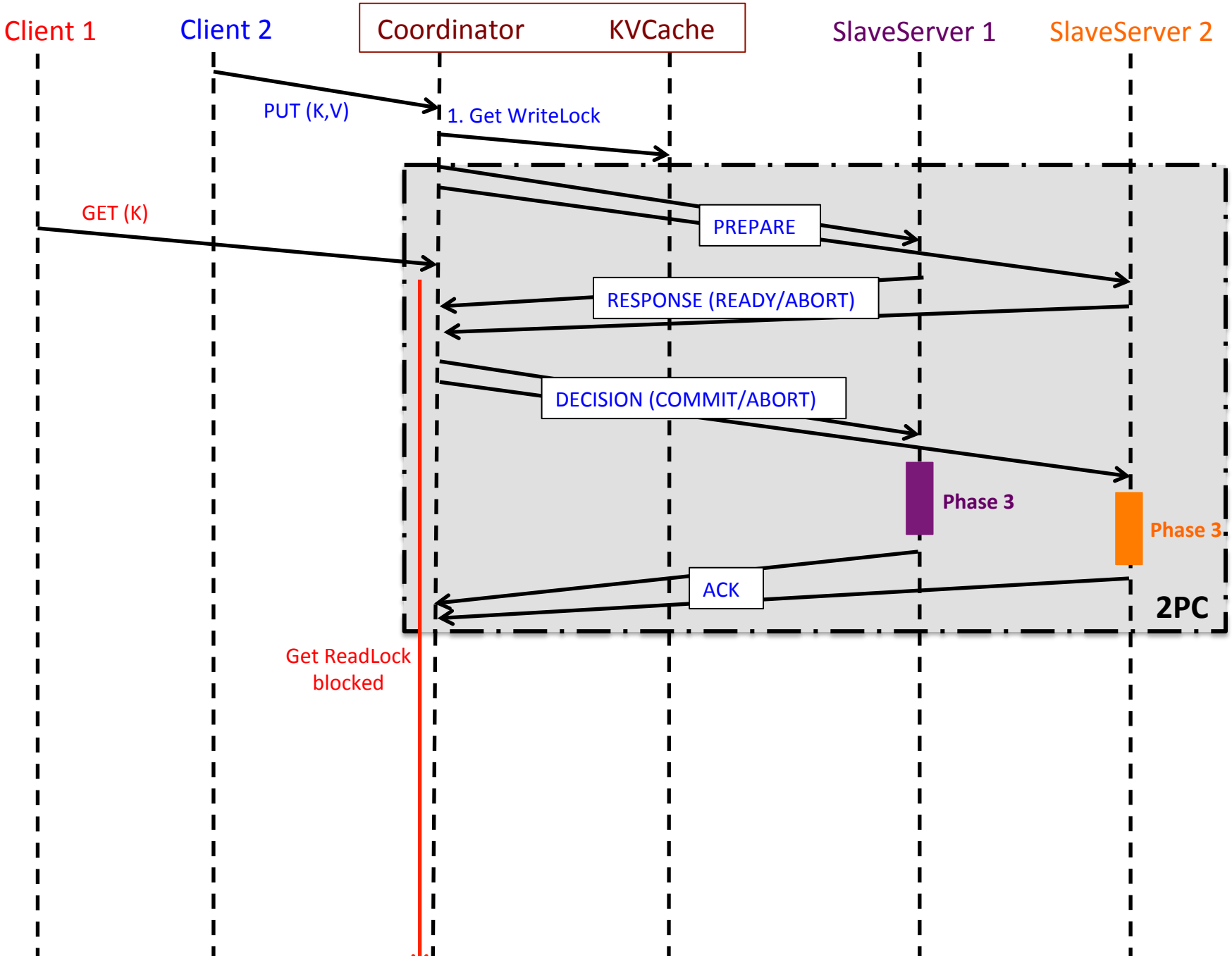


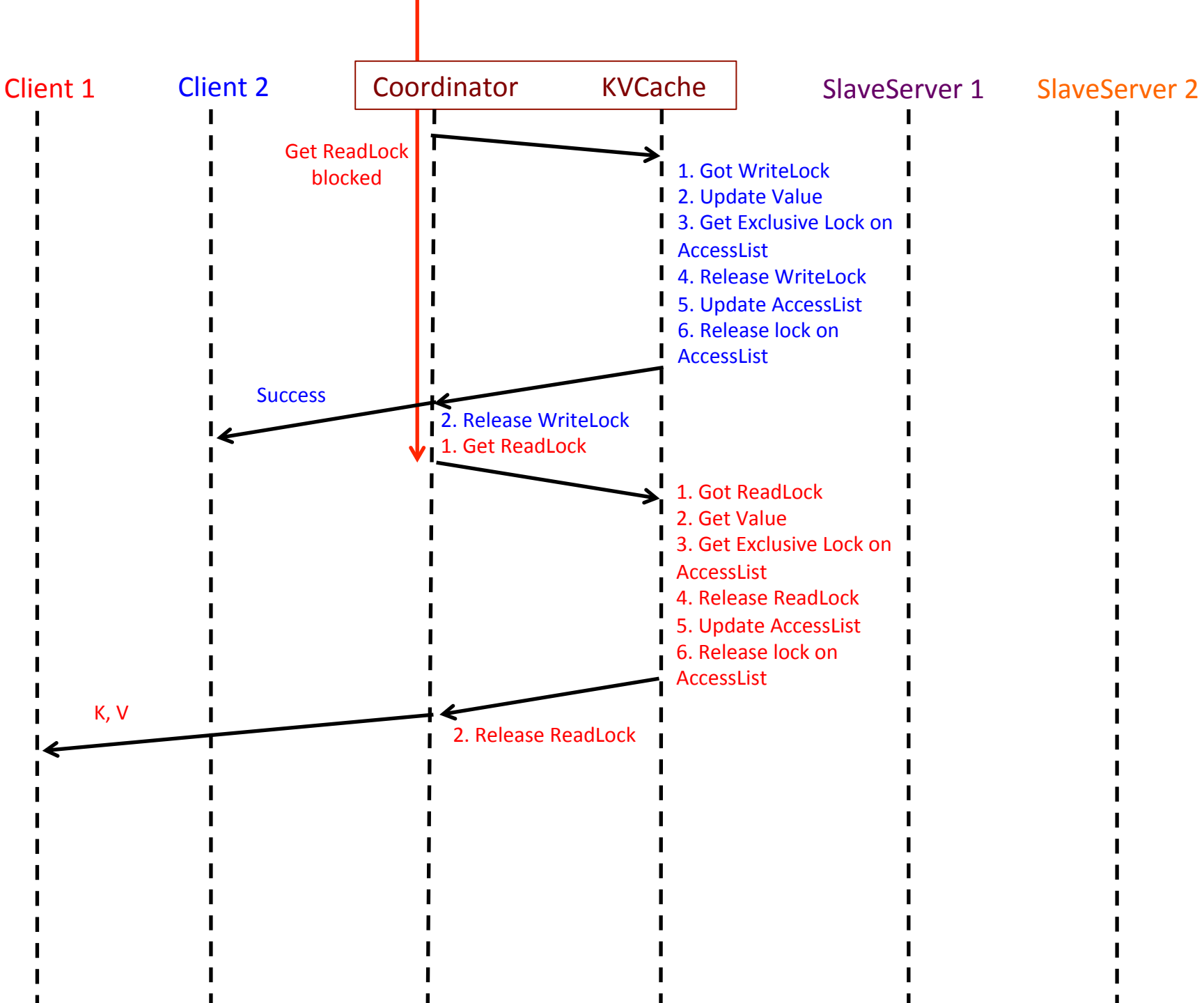

- SlaveServer 128-bit ID using `java.util.GUID`
  - Slave count known in advance
  - Nodes will come back if fails (no permanent failure)
- Hash Keys to 128-bit numbers


Client-side Program
KVClient (Library)
Client-side Program
KVClient (Library)
Client-side Program
KVClient (Library)
KVMessage
Socket Server
KVClientHandler
KVMessage
KVInterface
KVCache
(LRU Cache)
Job Queue
Available
Threads
Thread Pool
TPCMaster (Library)
Update
GET
PUT
DEL
Coordinator Server

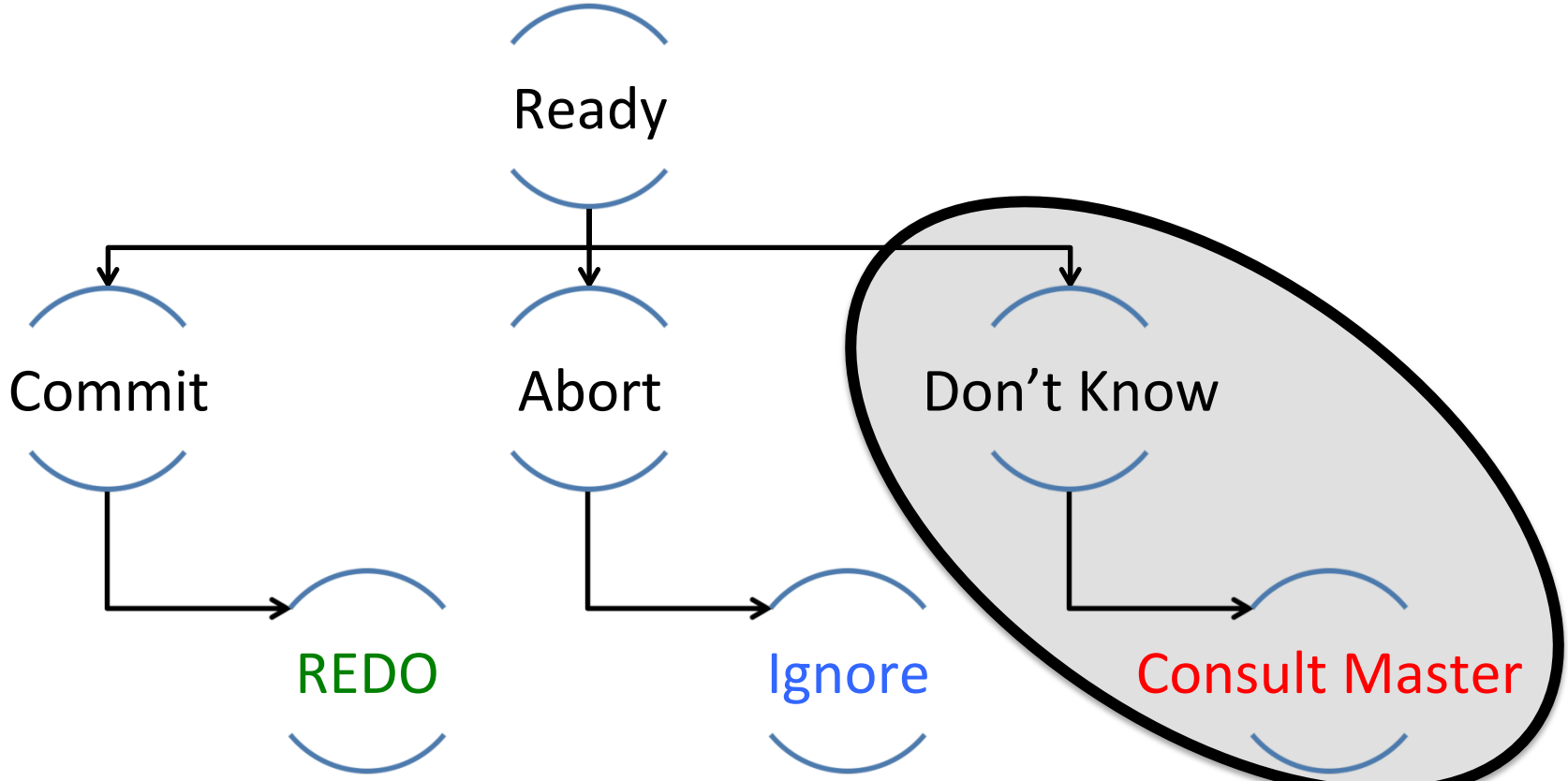

TPCMessage
Key-Value Server
Key-Value Server
Key-Value Server
Socket Server
TPCClientHandler
Job Queue
Available Threads
Thread Pool
TPCLog
KVInterface
KVCache (LRU Cache)
KVInterface
KVStore
KeyValue Server


Client 1
Client 2
Key Server
KVCache
KVStore
PUT (K,V)
1. Get WriteLock
Update Value
GET (K)
1. Get ReadLock
1. Get WriteLock
2. Update Value
3. Get Exclusive Lock on AccessList
4. Release WriteLock
5. Update AccessList
6. Release lock on AccessList
Get ReadLock blocked
Success
2. Release WriteLock
1. Got ReadLock
1. Get ReadLock
2. Get Value
3. Get Exclusive Lock on AccessList
4. Release ReadLock
5. Update AccessList
6. Release lock on AccessList
K, V
2. Release ReadLock


Client 1
Client 2
Coordinator
KVCache
SlaveServer 1
SlaveServer 2
PUT (K,V)
1. Get WriteLock
GET (K)
PREPARE
RESPONSE (READY/ABORT)
DECISION (COMMIT/ABORT)
Phase 3
Phase 3
ACK
2PC
Get ReadLock
blocked


Client 1
Client 2
Coordinator
KVCache
SlaveServer 1
SlaveServer 2
Get ReadLock
blocked
1. Got WriteLock
2. Update Value
3. Get Exclusive Lock on AccessList
4. Release WriteLock
5. Update AccessList
6. Release lock on AccessList
Success
2. Release WriteLock
1. Get ReadLock
1. Got ReadLock
2. Get Value
3. Get Exclusive Lock on AccessList
4. Release ReadLock
5. Update AccessList
6. Release lock on AccessList
K, V
2. Release ReadLock

# Recovery from Site Log Entries



What is MTBF?
What is MTTR?

# Fault Models

- Failures are independent*
  So, single fault tolerance is a big win
- Hardware fails fast (blue-screen, panic, …)
- Software fails-fast (or stops responding/hangs)
- Software often repaired by reboot:
  - Heisenbugs – Works On Retry
  - (Bohrbugs – Faults Again On Retry)

- Operations tasks: major source of outage
  - Utility operations – UPS/generator maintenance
  - Software upgrades, configuration changes

# Fault Tolerance Techniques

- Fail fast modules: work or stop
- Spare modules: yield instant  repair time
- Process/Server pairs: Mask HW and SW faults
- Transactions:  yields ACID semantics (simple fault model)

# Fault-tolerance in MapReduce

- What if
  - A task crashes
  - A node crashes
    - Worker node
    - Master node
  - A task is going slow

# Key Security Properties

- Authentication
- Data integrity
- Confidentiality
- Non-repudiation

# Lxvnkbgz Vhffngbvtmbhg pbma Vkrimhzktiar

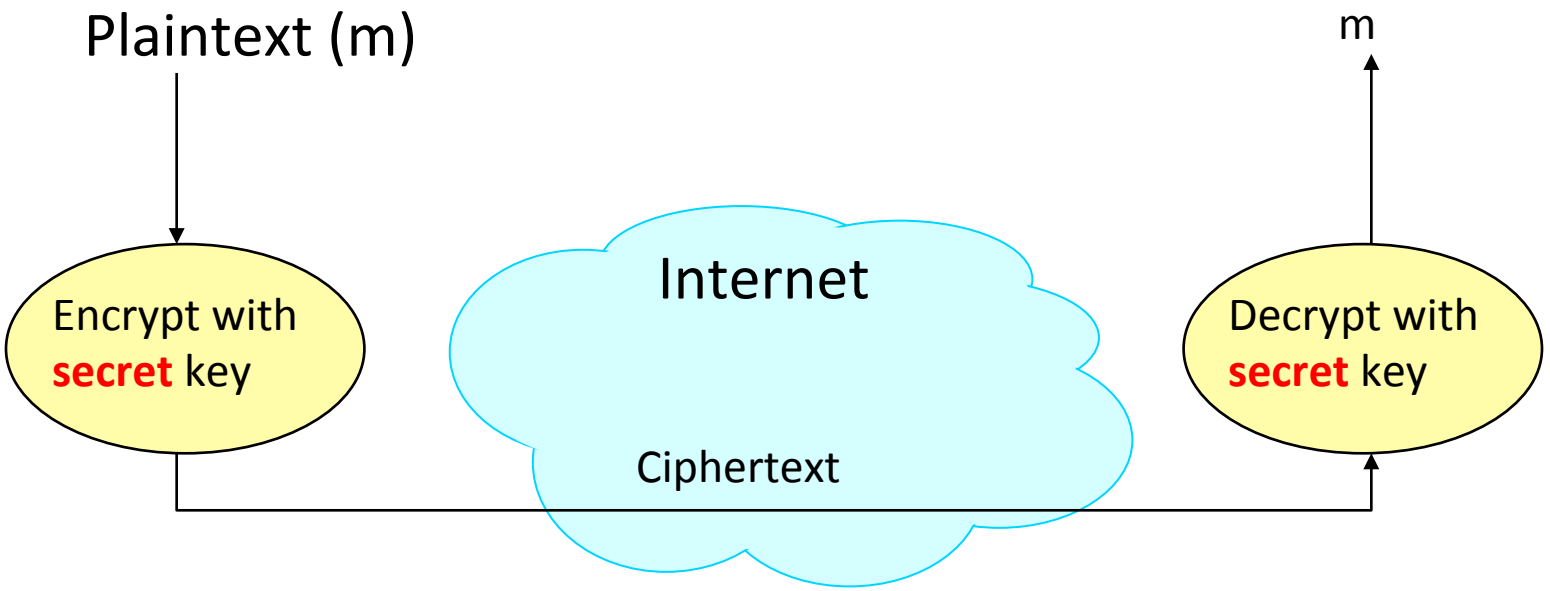
- Vhffngbvtmbhg bg ikxlxgvx hy twoxkltkbxl
- Maxkx bl t dxr, ihllxllbhg hy pabva teehpl wxvhwbgz, unm pbmahnm pabva wxvhwbgz bl bgyxtlbuex
  - Manl, dxr fnlm ux dxim lxvkxm tgw ghm znxlltuex

# Securing Communication with Cryptography

- Communication in presence of adversaries
- There is a key, possession of which allows decoding, but without which decoding is infeasible
  - Thus, key must be kept secret and not guessable

# Using Symmetric Keys

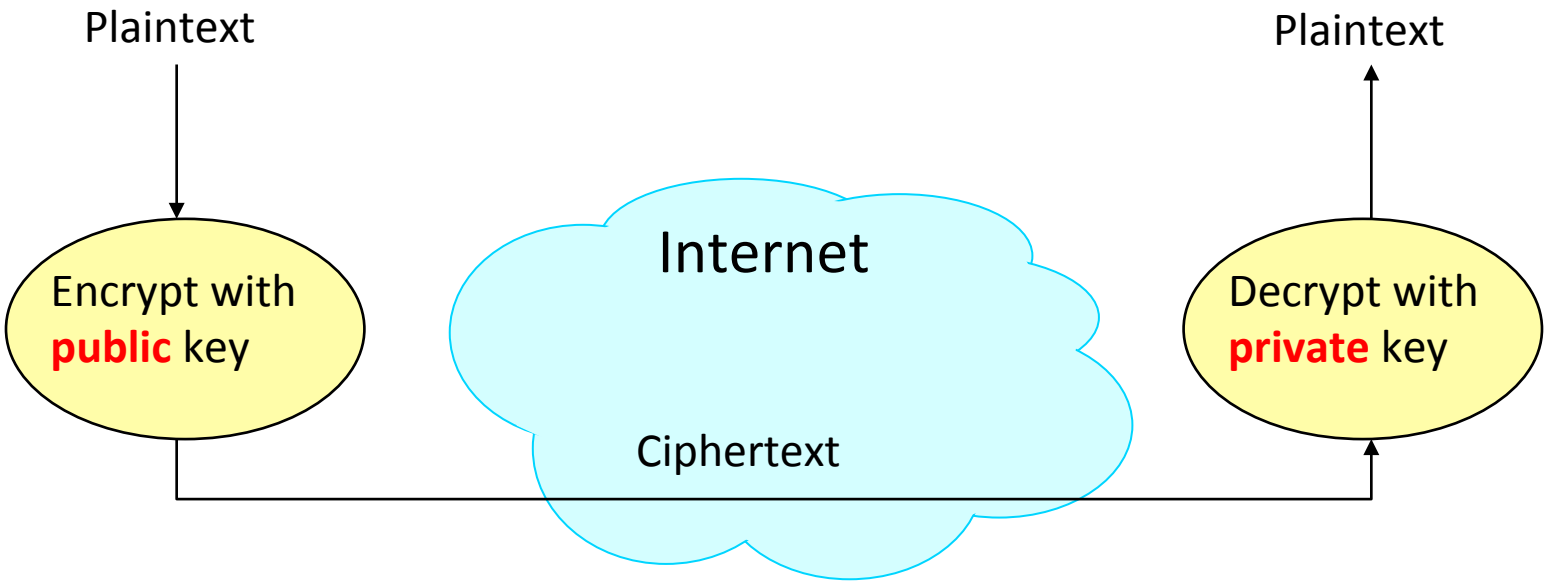
- Same key for encryption and decryption




- Transferring keys over the network is problematic

# Public Key / Asymmetric Encryption

- Sender uses receiver's public key
  - Advertised to everyone
- Receiver uses complementary private key
  - Must be kept secret



# Properties of RSA

- Requires generating large, random prime numbers
  - Algorithms exist for quickly finding these (probabilistic!)
- Requires exponentiation of very large numbers
  - Again, fairly fast algorithms exist
- Overall, much slower than symmetric key crypto
  - One general strategy: use public key crypto to exchange a (short) symmetric session key
    - Use that key then with AES or such
- How difficult is recovering *d*, the private key?
  - Equivalent to finding prime factors of a large number
    - Many have tried - believed to be very hard (= brute force only)
    - (Though *quantum computers* can do so in polynomial time!)