Project 3 Overview

CS162, 7/27–28/2015

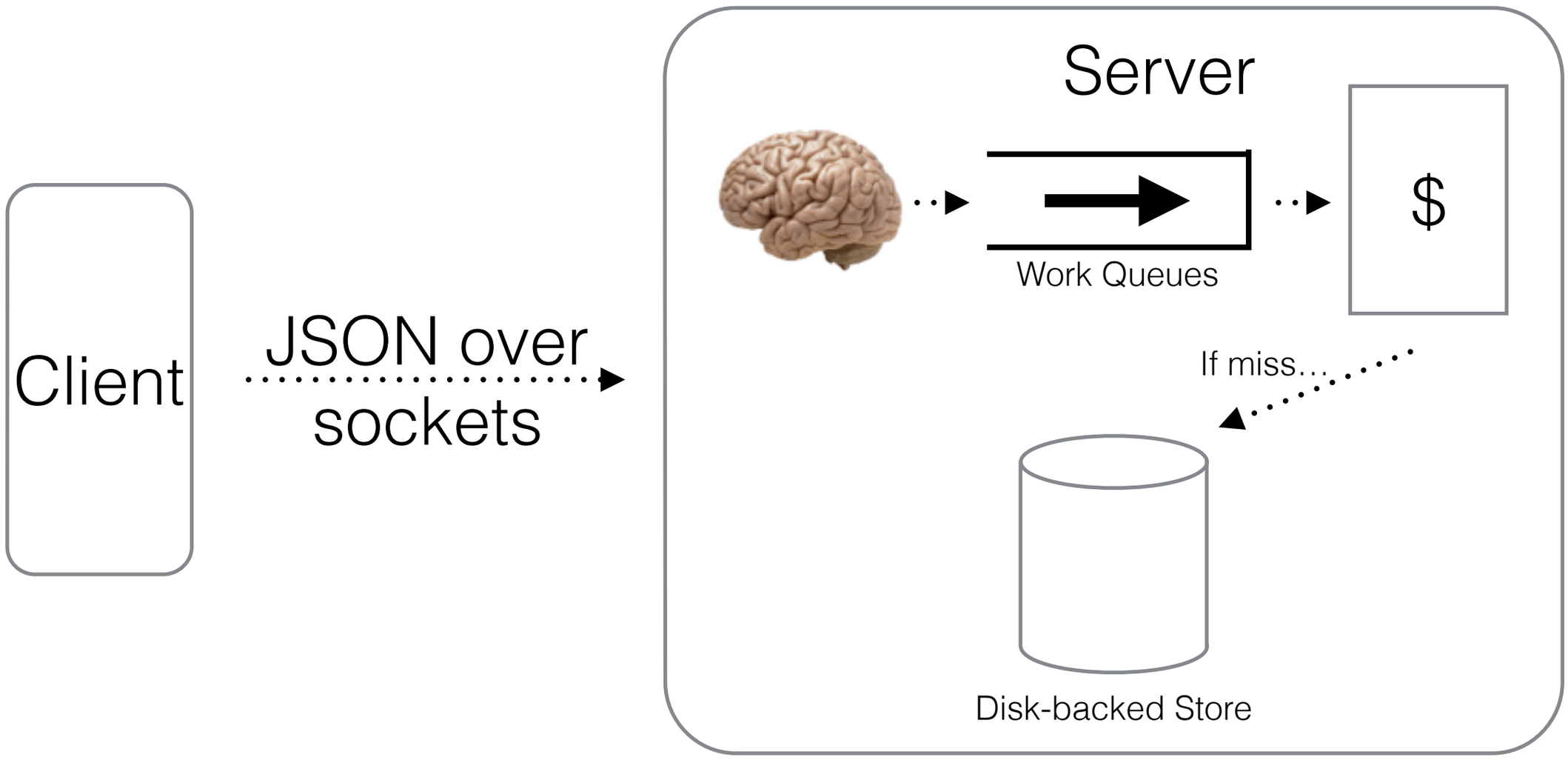
Deadlines

- **Design document:** 7/30/15 *before class*
- **Design review:** 7/31/15, sign up by 7/29/15
- **Checkpoint 1:** 8/3/15
- **Project deadline:** 8/10/15

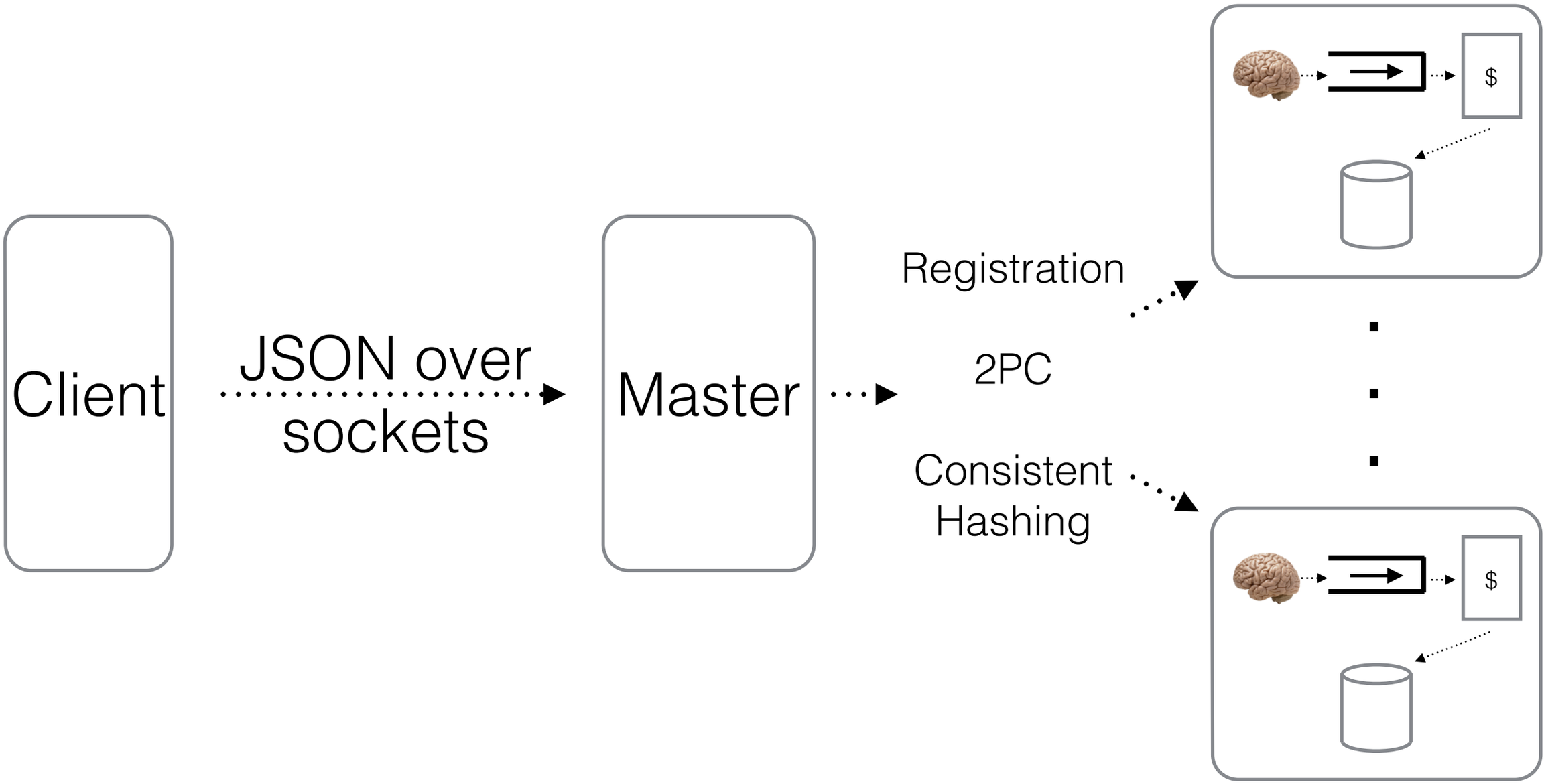
What are *you* building?

A key-value store that uses a
two-phase commit protocol!

KVS Architecture: Checkpoint 1



KVS Architecture: Checkpoint 2



What are you given?

- KVMessages: sends JSON messages over sockets. Fully implemented.

- KVConstants

- KVCache: In-memory cache for KV pairs. Mostly implemented, **need to implement call to correct cache set.**

- KVStore: Disk backed binary dump of KV pair. Fully implemented.

What are you given? (con't)

- Work Queue: queue for storing unprocessed jobs. Fully implemented, **except for synchronization (is not thread safe and does not wait when empty).**

- Socket server: General interface for communicating over sockets. Mostly implemented; **`server_run` needs to be modified to support multiple async requests.**

- UTHash, UTList

Starter code: *Important Note!*

- In PintOS, you were generally allowed to remove and change fields in internal structs, functions, etc.
    - Wasn't always a good idea, but was OK
- In this project, modifications to starter code are more restrictive:
    - Do not change/remove struct fields, typedefs
    - Do not change *any* provided function signatures
    - Adding fields to existing structs is OK
    - Adding structs, functions, typedefs is OK
    - Do not modify the provided hash functions!!!!!

What are you *not* given?

- Tests!
    - Actually, we give you a *few* tests.
        - But, unlike projects 1 and 2, we are holding tests back from you, which we will use to evaluate your work.
- What is our assumption here?
    - You will write lots of tests to make up for this!

What are you *not* given? (con't)

- Design doc stub!
- What is our message here?
    - For this project, we expect you to identify which portions are difficult.
- Your design doc should outline:
    - What new functionality are you adding?
    - What is difficult about this functionality?
    - How will you test this new code?

Checkpoint 1

Due 8/3/15

What do you need to do?

Make single node key-value store work!

A correct implementation will…

- Support concurrent operations. How?
    - Cache is partitioned —> allows parallel access.
    - Thus, need to use synchronized queues to pass work between caches and main control logic.
- Generally, make the caches work.

Cache structure

- A KVCache contains multiple sub-KVCacheSets.
- KVCache is fully implemented **except for** the get_cache_set method.
- Each cache set contains a subset of the key space. Thus, how can we figure out which key maps to which cache set?

What do the subsets look like?

- Cache subsets do *not* allow concurrent modification —> synchronize properly!

- Cache subsets are limited in size —> defined at init

- What is the eviction policy?

    - We specify a FIFO + second-chance policy.

    - I.e., pick the first KV pair that was added to the cache. If this was accessed since the last eviction, try to evict a different KV pair.

Important note:

If you do not make checkpoint 1 fully functional, the final checkpoint will be *unpleasant.*

Final Checkpoint

Due 8/10/15

What do you need to do?

Go single node to distributed!

Specifically…

- Add a master node that will allow you to:
    - Register worker nodes
    - Cache `get` results
    - On cache misses/`set`, coordinate between workers for 2PC
- Add persistent commit logging on workers

Registration

- Probably the simplest part of this assignment…
- Master node will listen on a specific port
- When a worker pings on that port, assign that worker a globally unique 64-bit ID

Consistent hashing

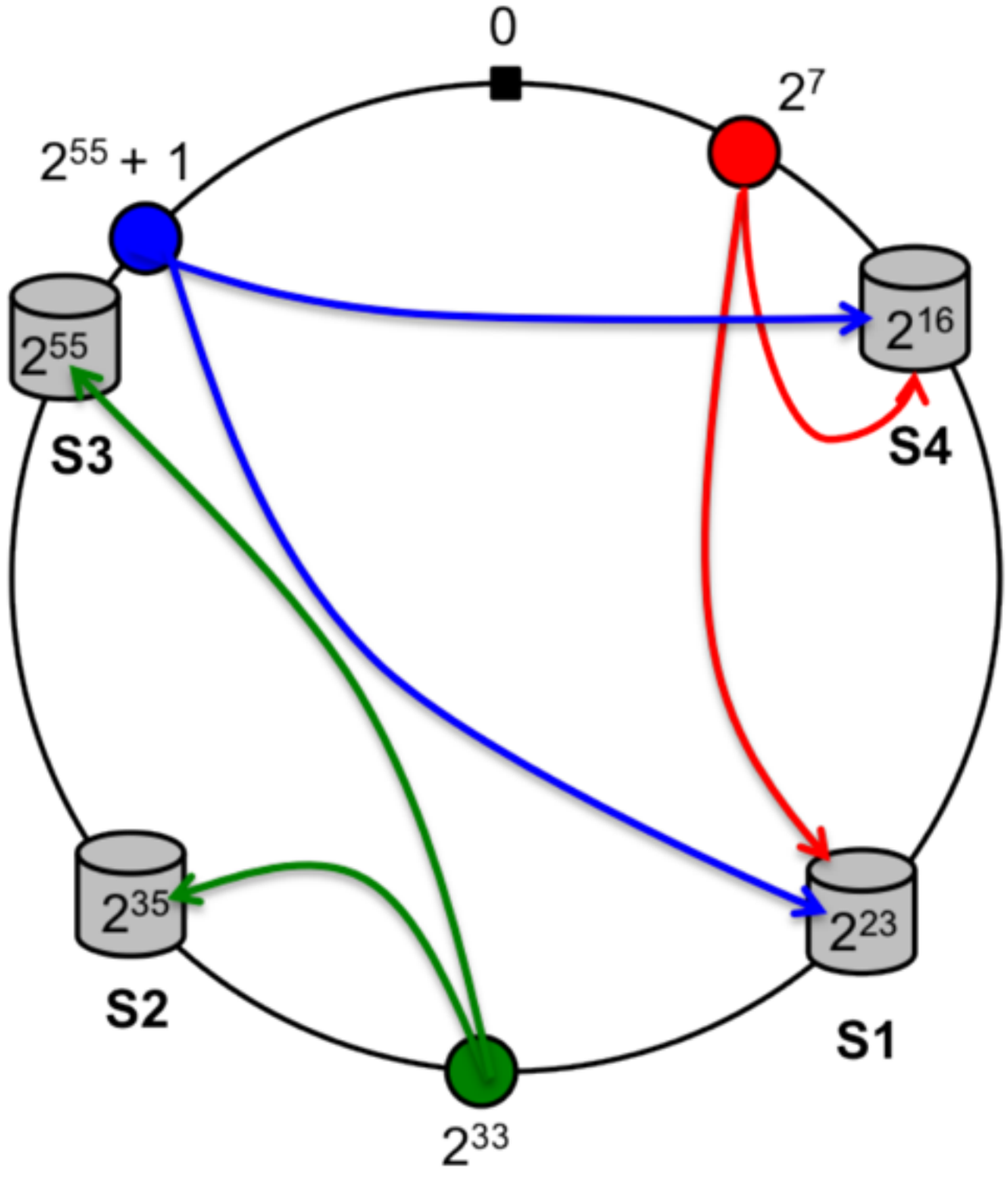
- Keys are replicated across nodes (replication factor is set on init)

- The master node is responsible for hashing keys to worker nodes

- How is this done?

Each slave stores:

- All keys with ID between immediate predecessor and own ID
- Duplicates of keys stored in *n* predecessors



Suggested reading: Stoica, Ion, et al. "Chord: A scalable peer-to-peer lookup service for internet applications." ACM SIGCOMM Computer Communication Review 31.4 (2001): 149-160.

2PC

- The cluster can execute a single 2PC at once
- When does this occur?
- What happens?
    - Master will identify all nodes that have a copy of the KV pair
    - The master will ask all server nodes to vote
        - When do you vote to abort?
    - If there is a quorum, the master will proceed
    - When done, all nodes should ACK. Else, retry after timeout.