ASL – Fall 2016

Project Presentation

Overview

- Tutorials on Tuesdays
- Project Q&A sessions on Thursdays
- Mapping Student -> TA is published on course web page
  - http://www.systems.ethz.ch/courses/fall2016/asl
- Questions about the project can be sent to
  - sg-asl@lists.inf.ethz.ch

Project rules

- No Team-Work
- Use
  - Java
  - Memcached + Memaslap
  - Ant (no Eclipse/Netbeans/IntelliJ project files for submission)
  - Microsoft Azure Cloud
  - Gitlab to submit code and reports
- Not allowed:
  - Use of external libraries (except log4j)
  - Using features such as load balancers, pre-made virtual machine images, etc. on Azure

Testing

- Your end product is not allowed to use anything but JDK, memcache and memaslap
- BUT: you are free to use any other library/framework for testing
  - E.g. you can use JUnit for testing and bash scripts for automation

Infrastructure

- Your **private machine** (or ETH machines in the PC rooms) for development and simple testing
- The **Azure Cloud** for distributed testing and benchmarking
- **Gitlab** for storing source code, experiment results and reports

Microsoft Azure Cloud

- All experiments will be ran in the cloud
  - Each of you will receive a "sponsorship" from us for the duration of the course, with a spending cap
  - All experiments shall be run on multiple machines
  - Use Ubuntu Virtual Machines, connect with ssh

- Your TA will sign you up with out sponsorship program during the next weeks – will have to have a **gitlab** repository to be eligible.

Problem statement

"The project consists of the
(i) design and development of a middleware platform for key-value stores as well as the
(ii) evaluation and analysis of its performance,
(iii) development of an analytical model of the system, and
(iv) an analysis of how the model predictions compare to the actual system behavior."

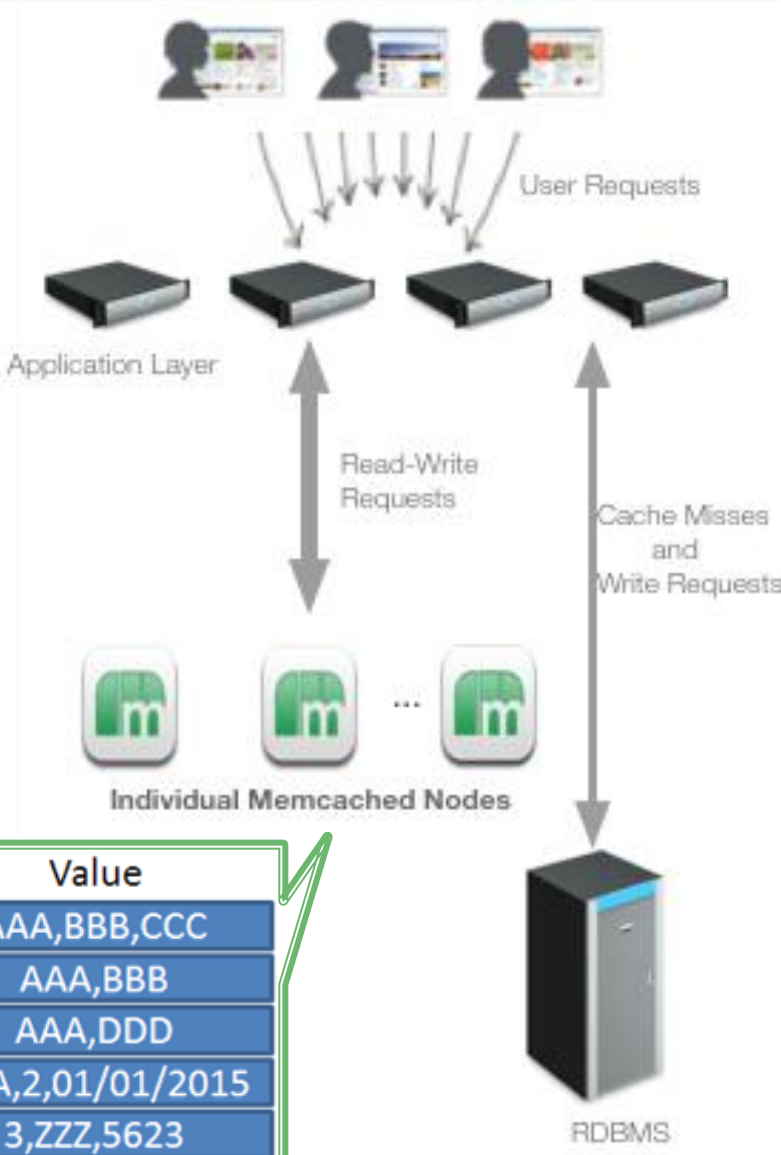
- Three milestones:
  - M1 = (I), 4 weeks
  - M2 = (ii), 5 weeks
  - M3 = (iii) and (iv), 4 weeks

Key-value stores

- Traditional relational databases (RDBMs) allow users to store data and query it (SQL) in a consistent and durable fashion.
  - Rigid schema
  - Relatively large overhead
- In the cloud-era there is more unstructured data, and the ability to scale out load is important
  - NoSQL data stores emerged: simple interface, simple schemas, almost no query processing
  - Many of them store data only in memory (no durability)
- memcache: cache for web applications

memcache

- Stores key-value pairs in main memory
  - Key is a string, value can be anything
  - No disk access
- Protocol consists of simple commands:
  - get somekey
  - set somekey <exp> <flag> <valuesize> somevalue
  - delete somekey
  - ….
- Is used mostly as a cache, items have expiration (ignore this for now)



Key	Value
K1	AAA,BBB,CCC
K2	AAA,BBB
K3	AAA,DDD
K4	AAA,2,01/01/2015
K5	3,ZZZ,5623

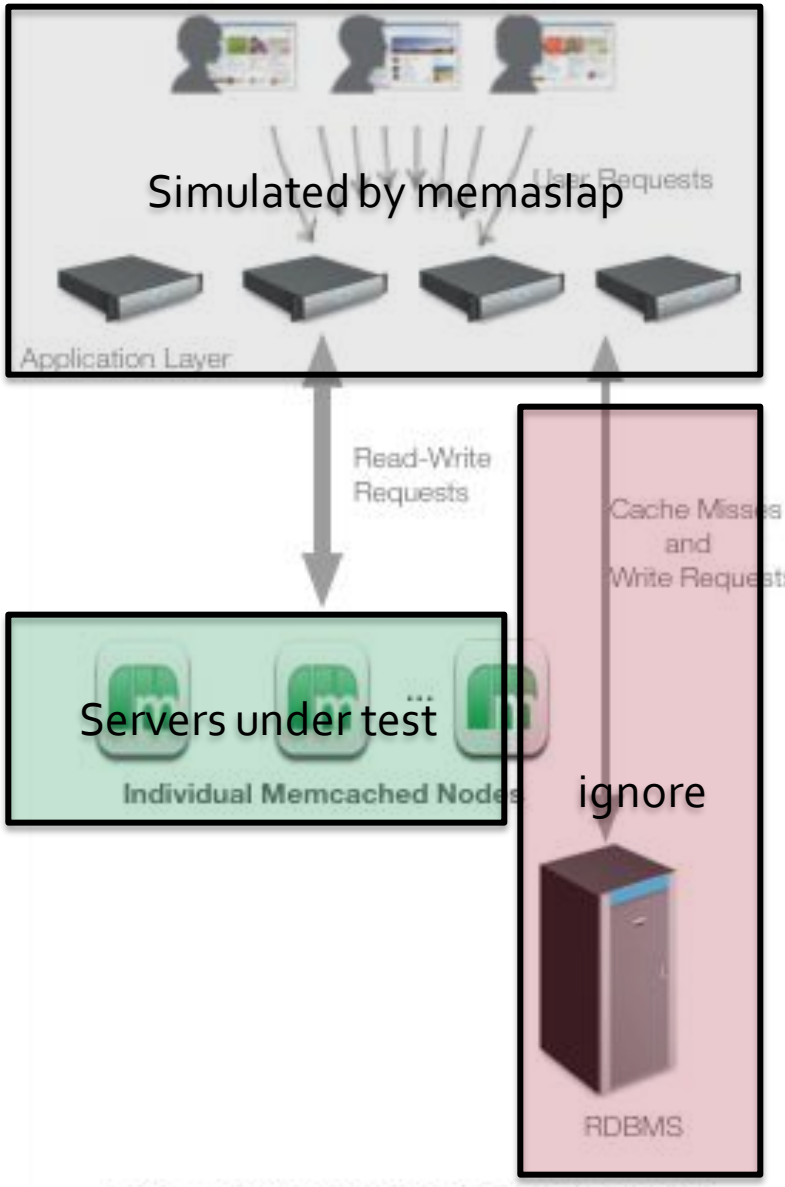
Using memcached as a caching layer

memcache (2)

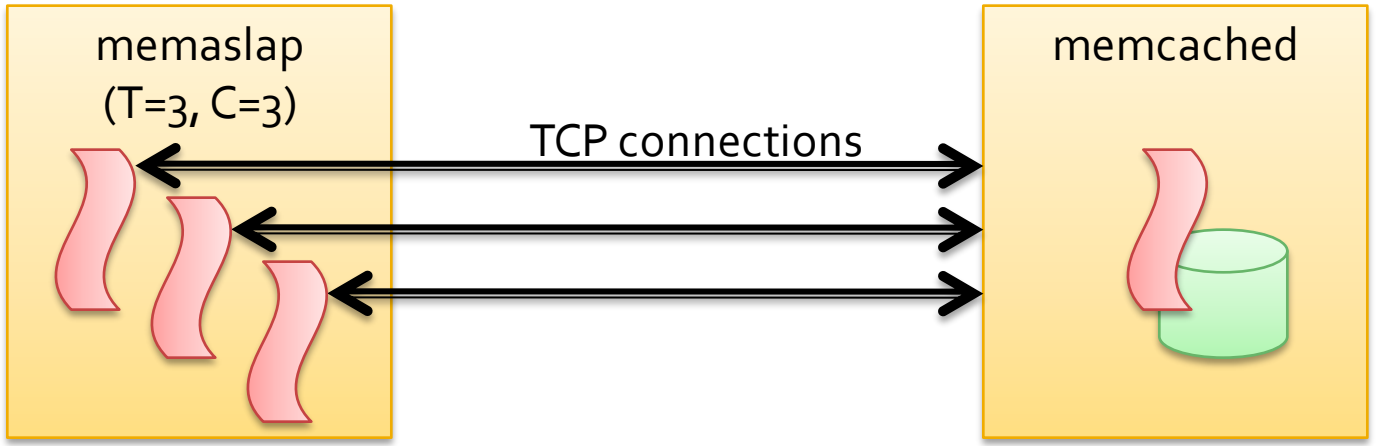
- It is a simple design, essentially single threaded
  - It does not reorder requests from the same client.
- Workloads:
  - Key size should be couple of bytes (16B for this course)
  - Value size can reach up to 1MB, but we are interested in 128B-1KB range in this course
  - For very large values the network bandwidth will be the bottleneck. In other cases processing will be the bottleneck.

memaslap

- Load generator for memcached for testing
  - Generates "gibberish" keys and values
  - Requests created according to workload (%read, %write, key and value size)
- Measures and reports throughput and response time
- Part of the *libmemcached* library, will provide steps on how to build it



Simple benchmarking



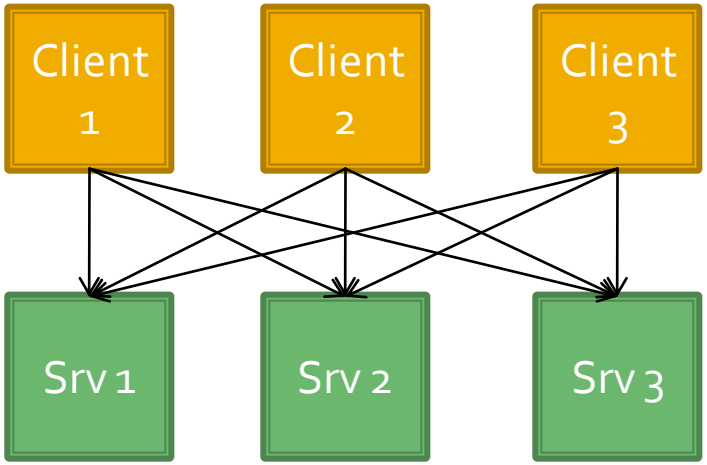
Important parameters: Threads and Clients:

- C connections will be opened to memcached
- If T>C multiple virtual clients will share a single thread and response times will go up
- Use it with T=C=32 or 64.

No setup overhead. Just specify number of threads (1) and port (e.g. 11212). If stopped, loses data.

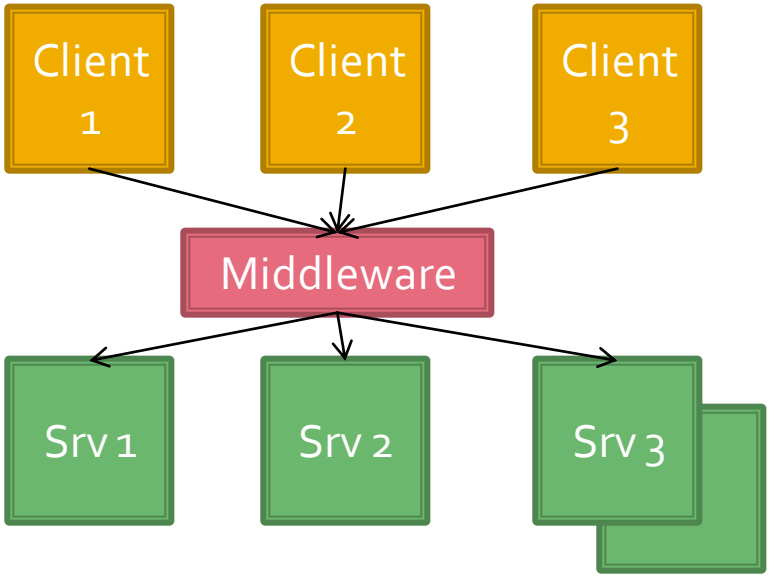
What is a middleware?

NO MIDDLEWARE



- How to add/remove servers?
- What if we want to load balance?
- How about replication without changing the clients?

WITH MIDDLEWARE



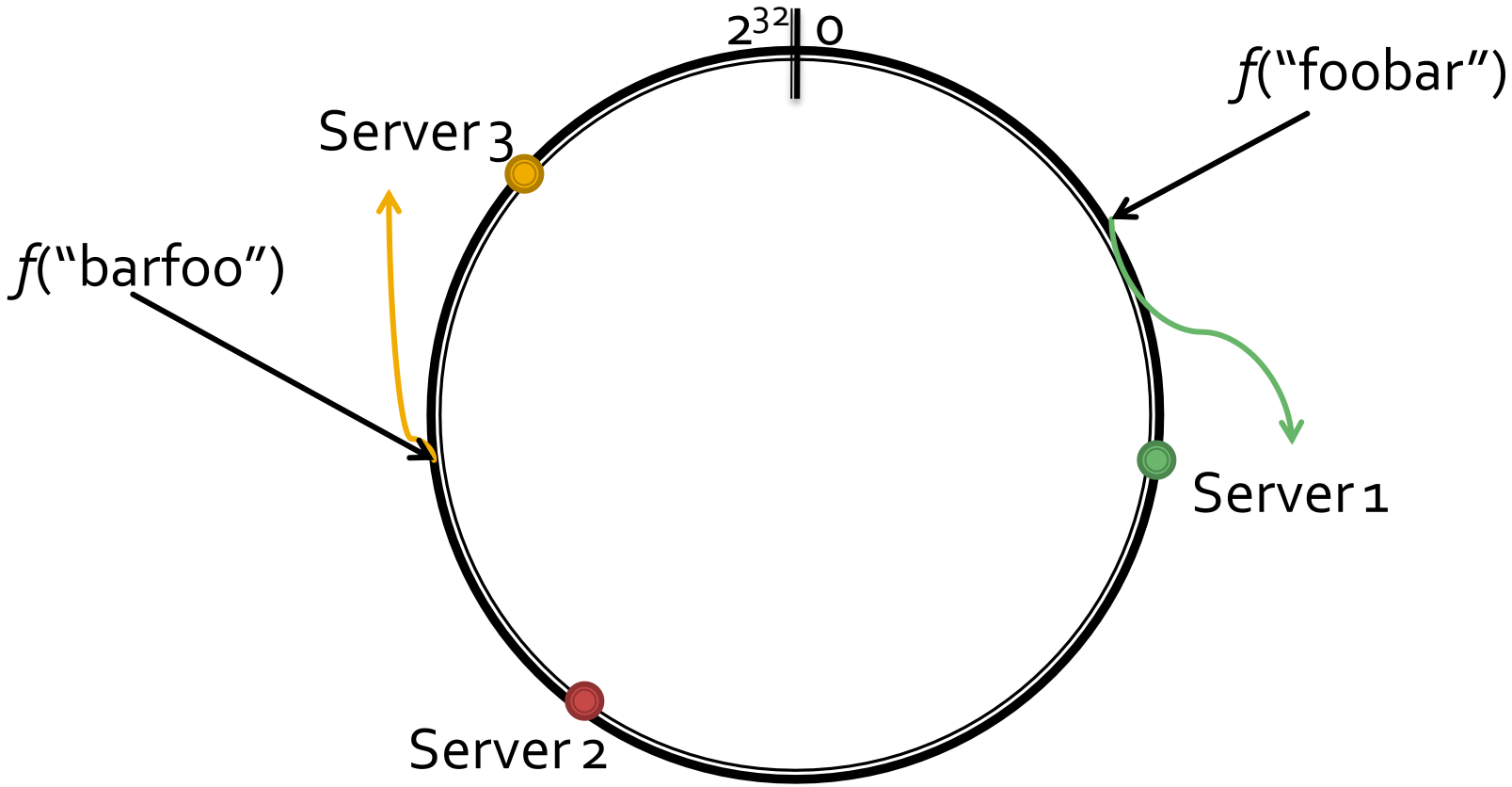
- Decouples server layout from clients
- Allows for load balancing
- Transparent replication
- Etc.

Your middleware

- Act as a relay
  - Relay messages from clients to memcached servers and then do the same for answers
- + Load balance
  - Understand what is in a message (set? get?) and find the key to route the message based on it
- + Replicate
  - Perform write operations in more than a single location to make sure the data is safe – will have to wait for ALL answers before relaying to client the final response

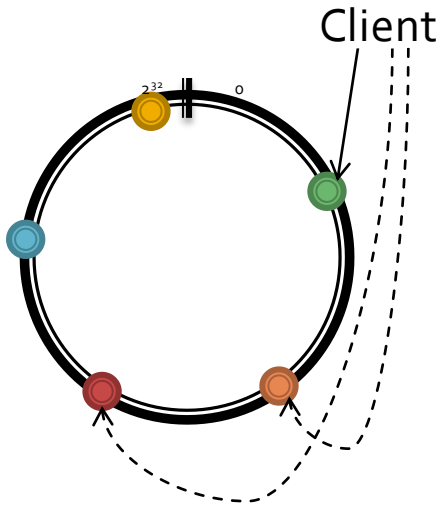
Consistent hashing for LB

- Hash function: *f*(key) → integer

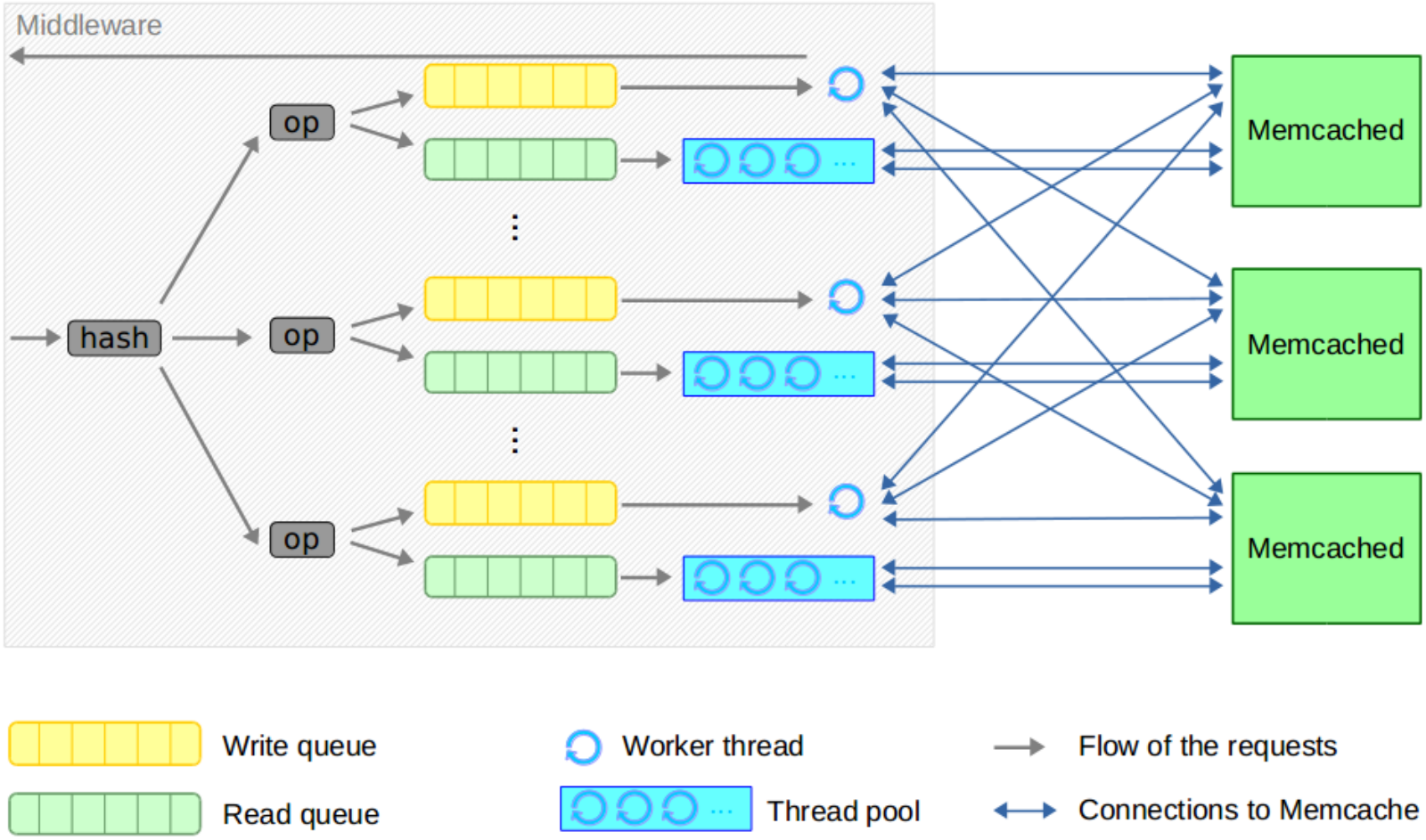


1+r Replication

- Consistent hashing determines the "home" of a key
  - Write it to *r* replicas, read it from its main copy
  - Choose replicas as right-hand neighbors on the ring
- A write is successful if all replicas succeed

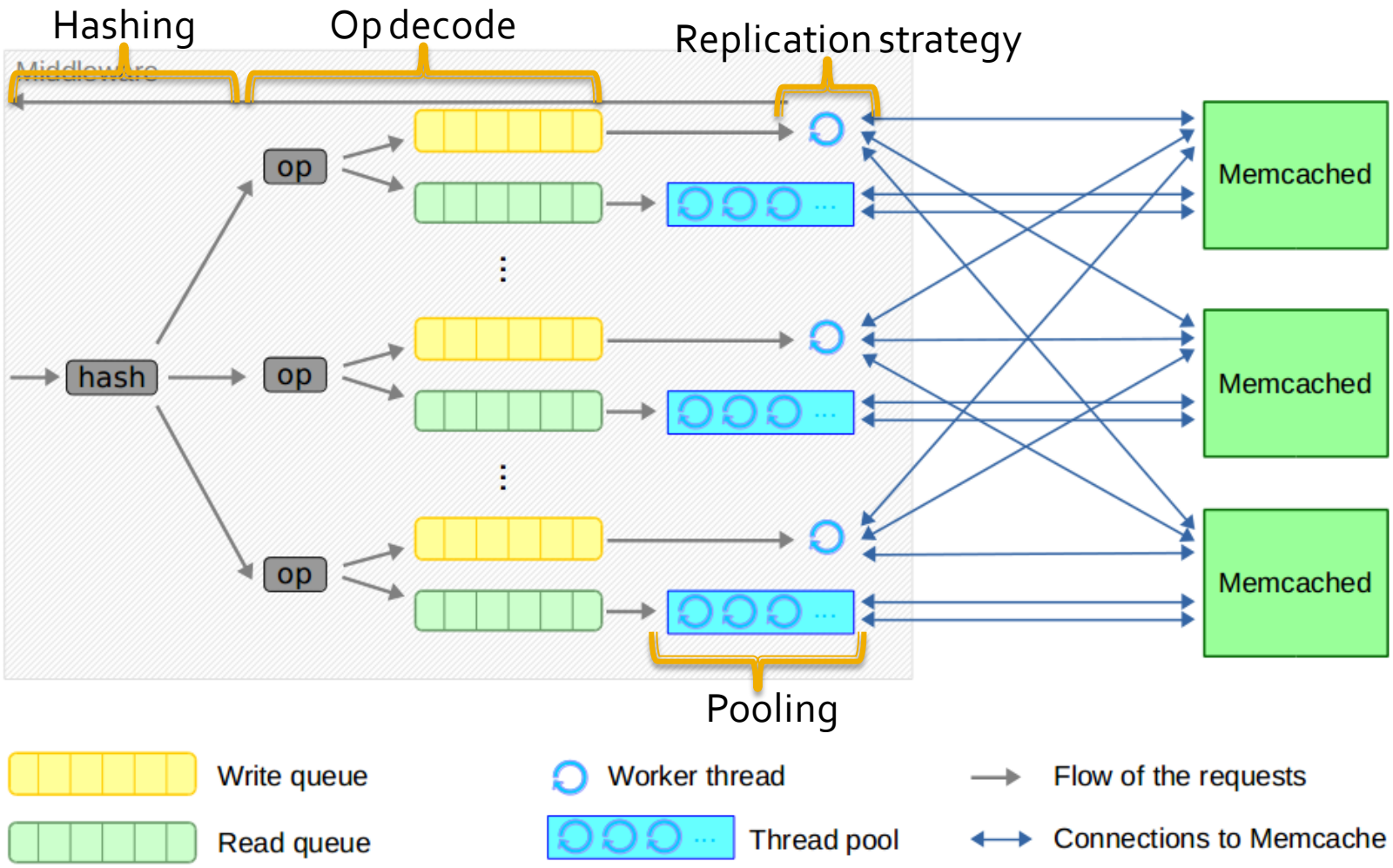


Internal architecture



You also have to instrument the code to measure time spent in different parts of the system per request

Internal architecture



You also have to instrument the code to measure time spent in different parts of the system per request

A word on synchronous/asynchronous networking

- **Synchronous** networking on a thread:
  1. send data
  2. wait idle until the response is available
  3. process response

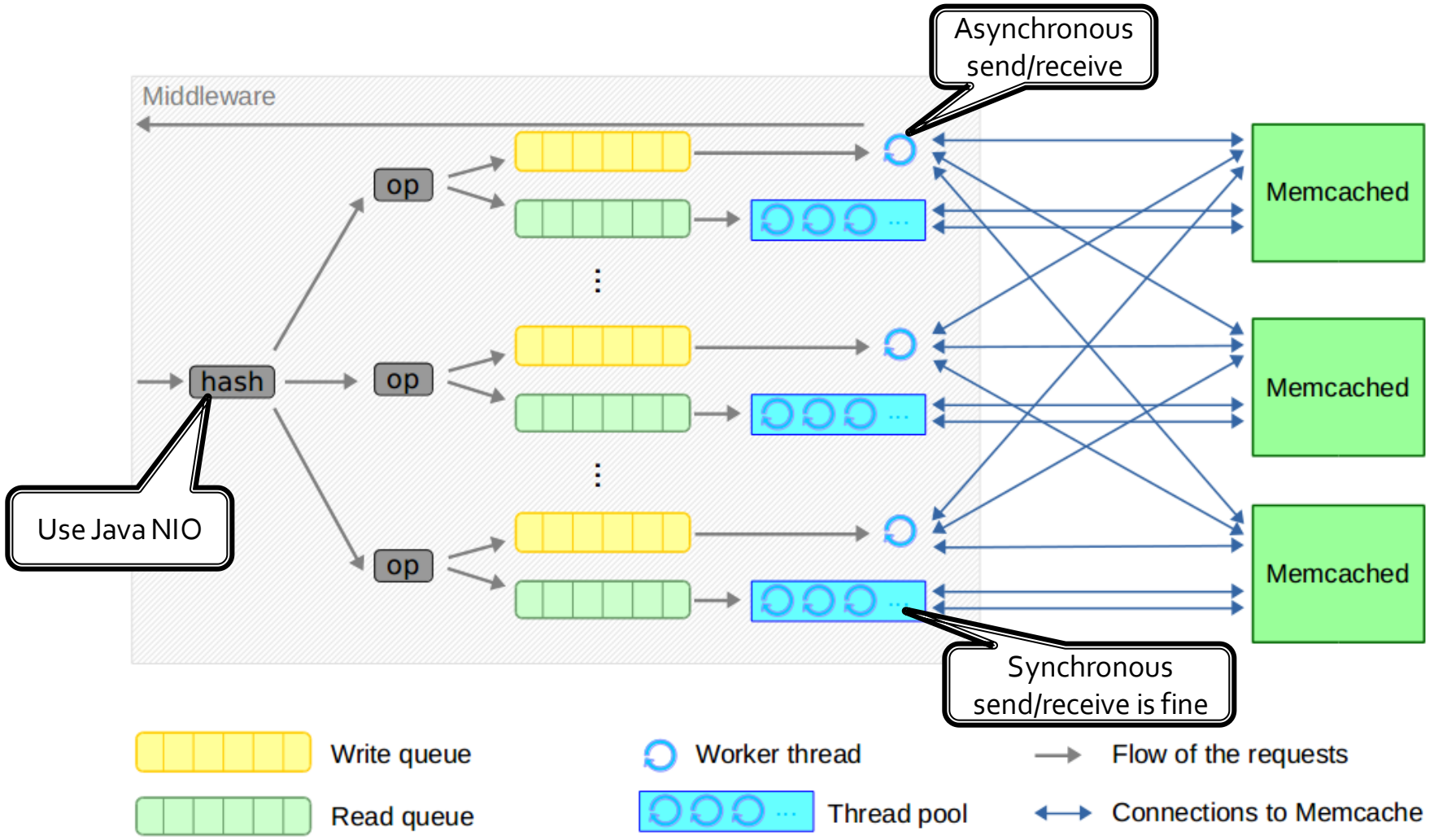
- One thread can only perform **1/RTT** requests even if processing on the other end is immediate
- Can mitigate this issue by using a pool of connections and multiple threads – out of order processing?

A word on synchronous/asynchronous networking (2)

- **Asynchronous** networking on a thread:
  1. send data
  2. if no data available, goto 1.
  3. process responses

- This hides the network round-trip time
- Keeps order of requests (important for writes!)
- But, more challenging to code

Internal architecture



You also have to instrument the code to measure time spent in different parts of the system per request

First Milestone

- Describe the design and implementation
  - Documentation, not "story"
  - Thread pool, connection management, etc.
- Baselines for memcached without middleware
  - Hint: do it in parallel with the implementation
- Trace of the middleware connected to multiple client machines and servers

- We will provide a **template** for each milestone.