Assignment #1

Implementing a network service that can handle large scale network requests

* Co-worked with Jaehyun Nam
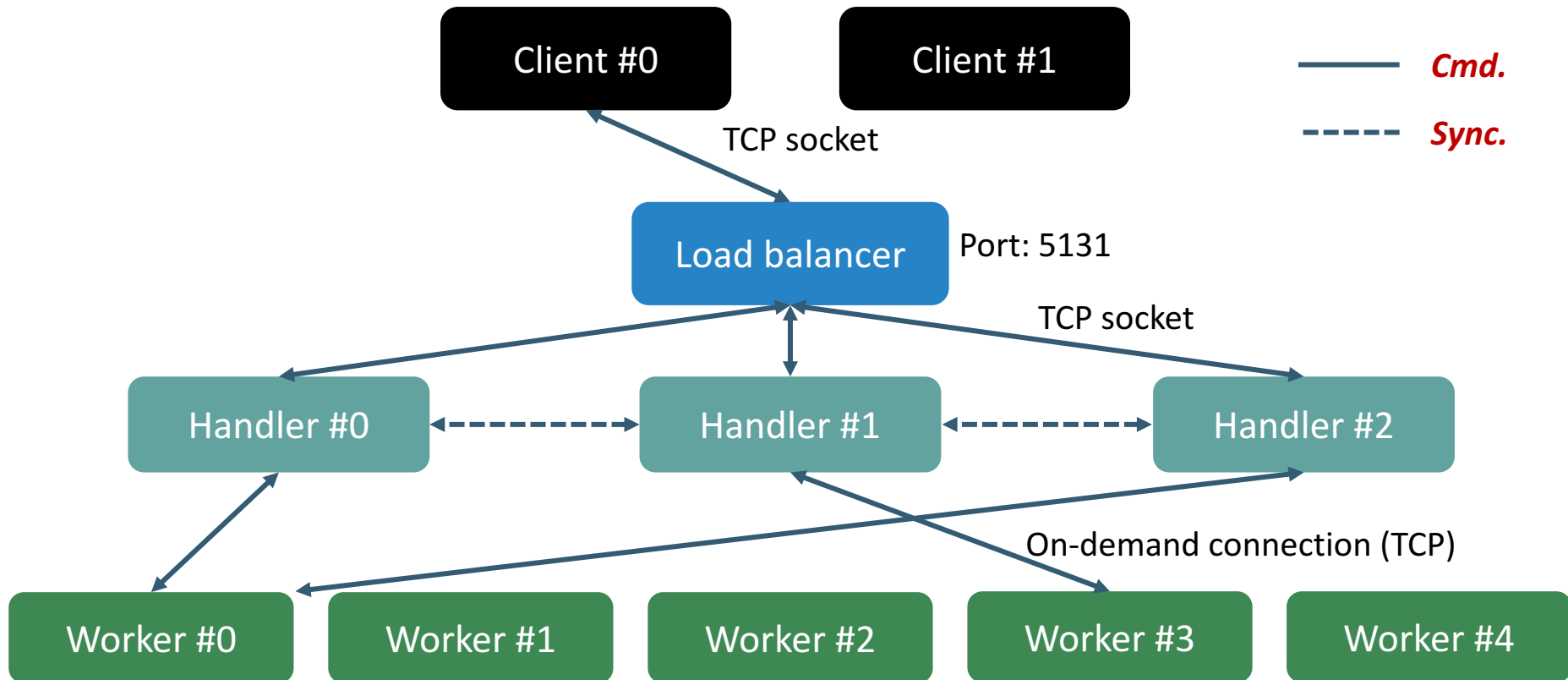
High-level functions

- ***Ultimate Goal: Build a distributed key-value storage system***

- Store/retrieve data as a ***key-value pair***
- Handle ***bulk requests*** with I/O multiplexing and load balancing
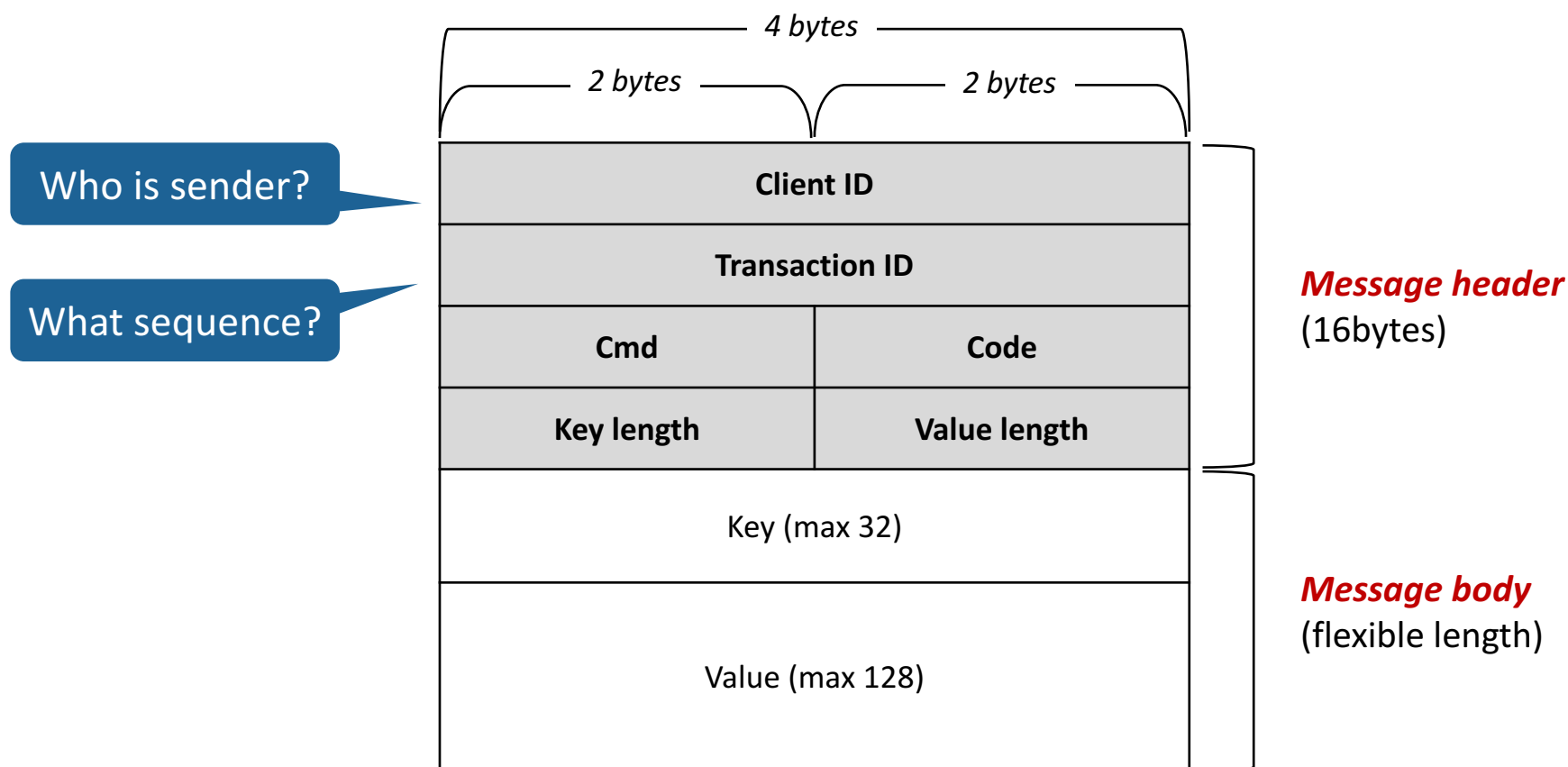- Write ***logs*** and provide ***CLIs*** to interact with users

Overall architecture

- All components = processes

Client #0   Client #1   ——— **Cmd.**

TCP socket   - - - - **Sync.**

Load balancer   Port: 5131

TCP socket

Handler #0 ⟵ - - - ⟶ Handler #1 ⟵ - - - ⟶ Handler #2

On-demand connection (TCP)

Worker #0   Worker #1   Worker #2   Worker #3   Worker #4

Protocol

- Data format

	4 bytes	
2 bytes		2 bytes

Who is sender? → Client ID

What sequence? → Transaction ID

Cmd	Code
Key length	Value length

Key (max 32)

Value (max 128)
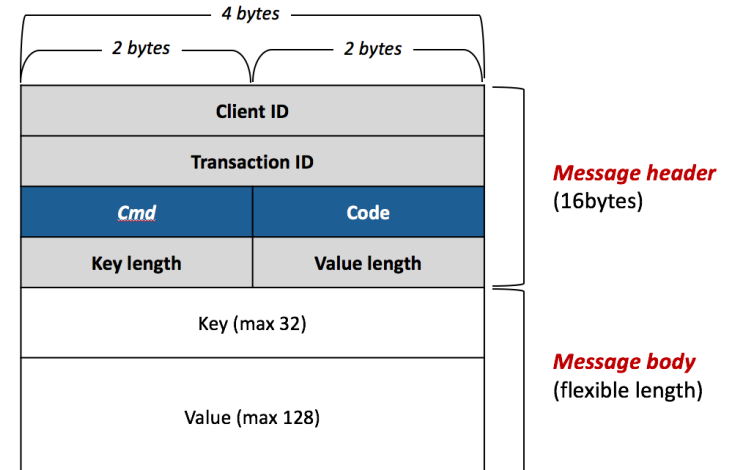
Message header (16bytes)

Message body (flexible length)

Protocol(Cont'd)

- Cmd
  - Identify **commands**
  - **put**: 1, *put-ack: 2*, **get**: 3, *get-ack: 4*, **del**: 5, *del-ack: 6*
- Code
  - Represent **status** for *put-ack, get-ack,* and *del-ack*
  - **none***: 0, *success: 1, not exist: 2, already exist: 3*
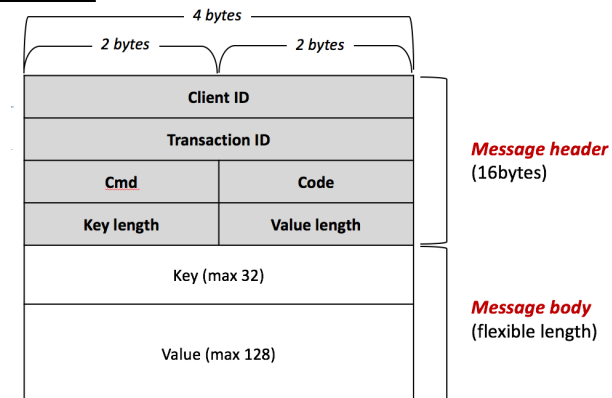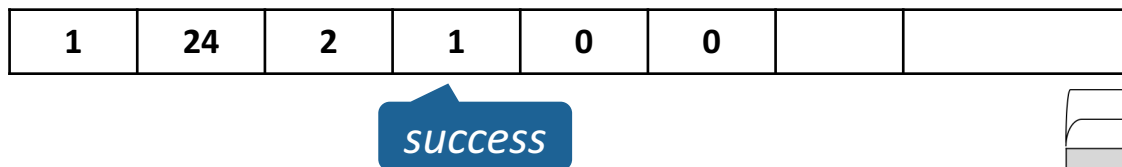
  * Only used for non-ack messages

4 bytes	
2 bytes	2 bytes
Client ID	
Transaction ID	
Cmd	Code
Key length	Value length
Key (max 32)	
Value (max 128)	

Message header
(16bytes)

Message body
(flexible length)

Examples

- Client #1 requests **"*put*"** a *{hello, helloworld}* pair into the system **(sequence #24)**

1	24	1	0	5	10	hello	helloworld

put   *none*
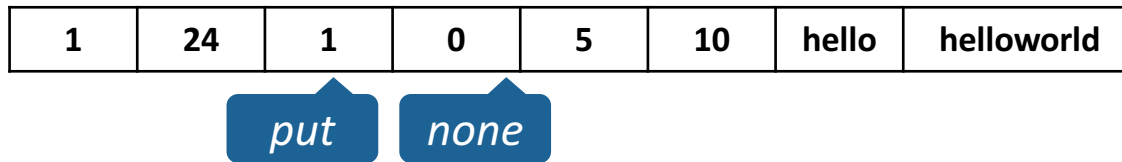
- The **"*put*"** *message* requested by Client #1 was successfully stored on the system**(sequence #24)**

1	24	2	1	0	0		

success

4 bytes	
2 bytes	2 bytes
Client ID	
Transaction ID	
Cmd	Code
Key length	Value length
Key (max 32)	
Value (max 128)	

Message header (16bytes)

Message body (flexible length)

Examples(Cont'd)

- Client #2 requests **"get"** a *{hello}* key to the system **(sequence #25)**

1	25	3	0	5	0	hello	

get *none*
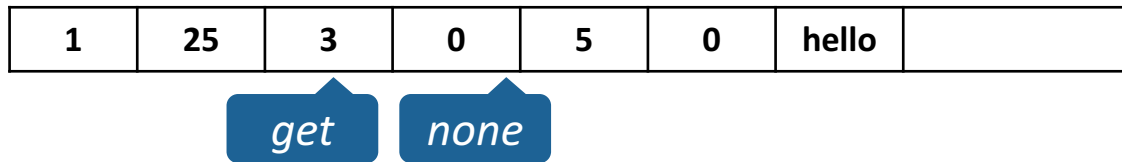
- The **"get"** *message* requested by Client #2 was successfully found on the system**(sequence #25)**

1	25	4	1	5	10	hello	helloworld

success

4 bytes	
2 bytes	2 bytes
Client ID	
Transaction ID	
Cmd	Code
Key length	Value length
Key (max 32)	
Value (max 128)	

Message header (16bytes)

Message body (flexible length)
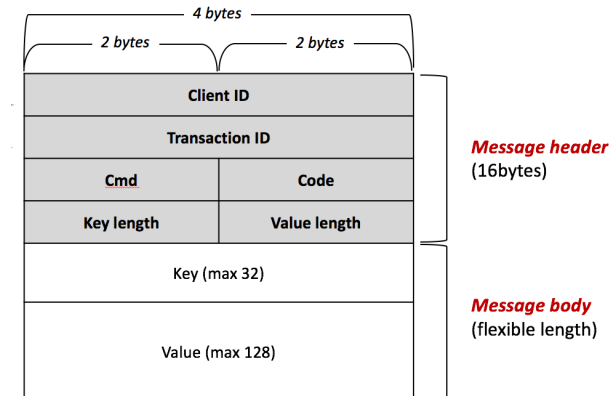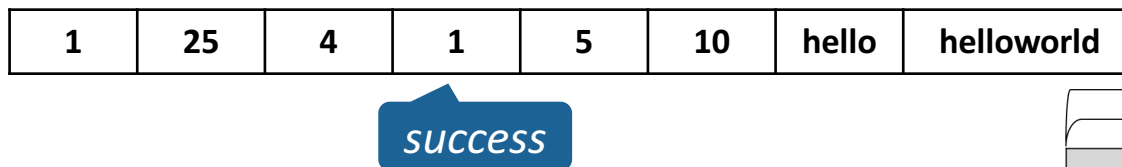
Operations – Load balancer

- Request delivery to *a specific handler* according to a LB strategy
  - ➔ *Round-robin*
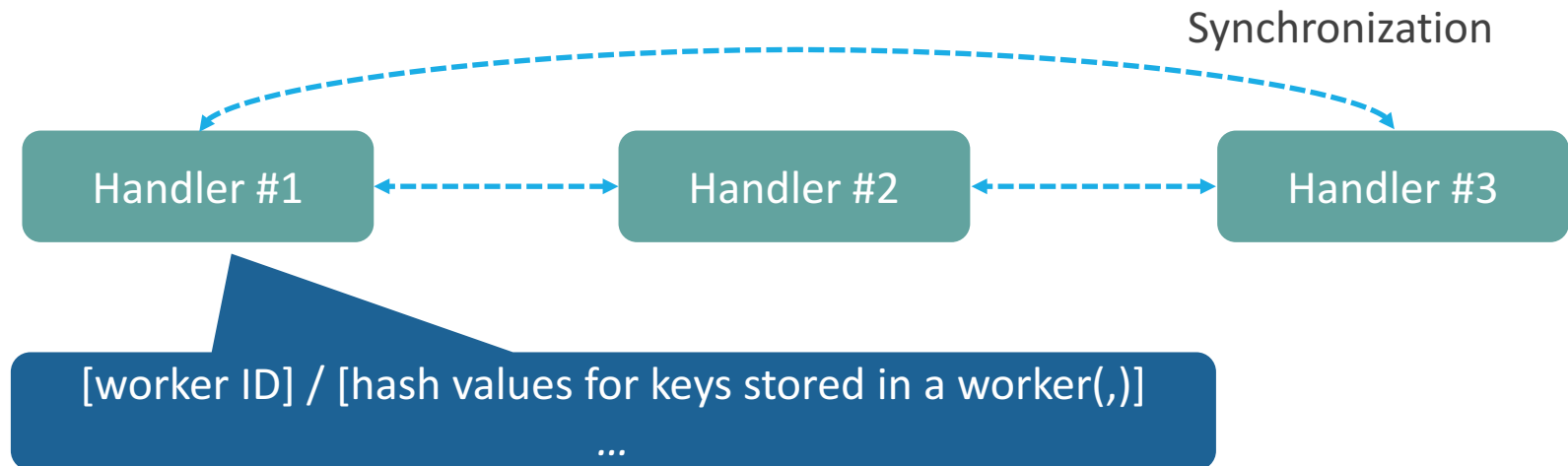
- Response delivery to the corresponding client
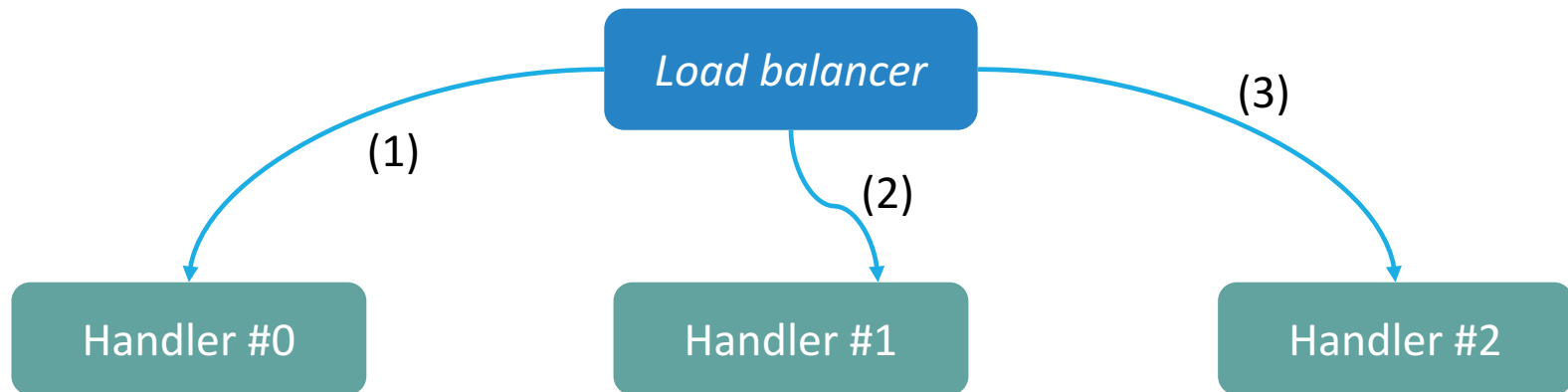
Operations - Handler

- Request delivery to a specific worker and response delivery to the load balancer

  - *Target worker = Hash(key) % # of workers*
  - You should use *Jenkins hash function(one-at-a-time)*
    - See https://en.wikipedia.org/wiki/Jenkins_hash_function

Handler #1

0 = Hash("hello") % 5

Worker #0   Worker #1   Worker #2   Worker #3   Worker #4
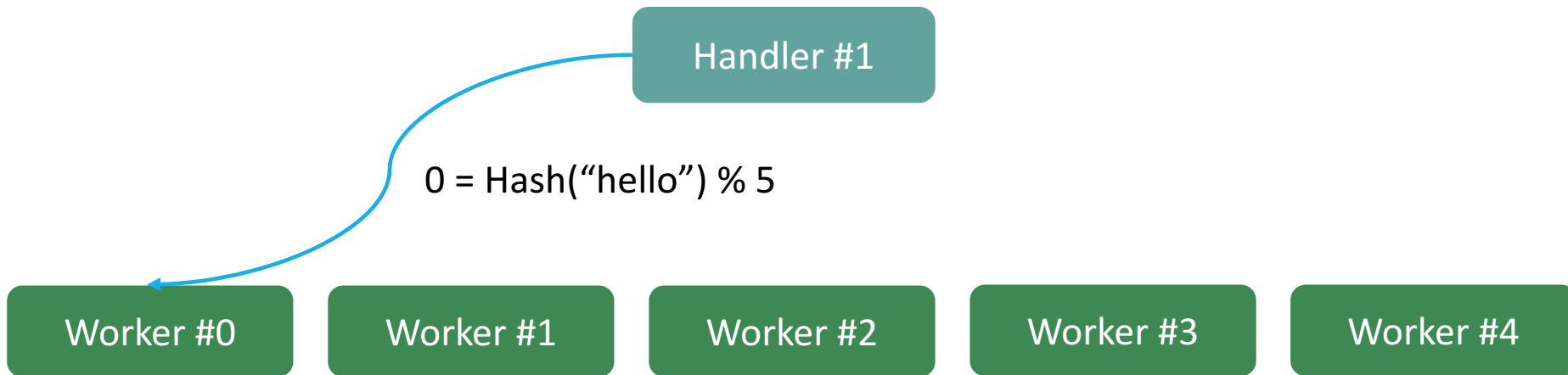
Operations – Handler(Cont'd)

- Management of **key-value distributions**

- **Synchronization** of key-value distributions among handlers
  - All handlers should eventually know all key-value distributions

  * ignore the strong consistency and failure issues

Synchronization

| Handler #1 | Handler #2 | Handler #3 |

[worker ID] / [hash values for keys stored in a worker(,)]
…

NS² Network and System Security Laboratory KAIST

Operations - Worker

- **Put** a pair of a key and a value into its data table
- **Get/Del** a value of a key from its data table

  ignore the failure issue

Worker #1	Worker #2	Worker #3	Worker #4	Worker #5

[hash value of a key] / [key] / [value]
...

[hash value of a key] / [key] / [value]
...

[hash value of a key] / [key] / [value]
...

Requirement #1 - Message logging

- All received messages *should be written* in log files

- Format: *[component_name].log* (e.g., handler_2.log, worker_1.log)

- Contents:
  - *[time] [client id] [transaction id] [cmd] [code] [key length] [value length] [key] [value]*
  - *[time] [client id] [transaction id] [cmd] [code]*

Requirement #2 - CLI

- Client's CLI(Example)

```
Client> connect [LB IP]

Client> set 1
Client #1>
```

```
Client #1> put [key] [value]
Success
    or
Fail (reason: already exist)
```

```
Client #1> get [key]
[value]
    or
Fail (reason: not exist)
```

```
Client #1> del [key]
Success
    or
Fail (reason: not exist)
```

Requirement #2 – CLI(Cont'd)

- Load-balancer's CLI

```
LB> list
[handler ID] / [handler IP address / port] / [# of requests]
```

- Handler's CLI

```
HD# [handler ID]> list
[worker ID] / [hash values for keys stored in a worker(,)]
HD# [handler ID]> show [worker ID]
[hash values for keys stored in the worker]
```

- Worker's CLI

```
WK# [worker ID]> list
[hash value of a key] / [key] / [value]
WK# [worker ID]> show [key]
[hash value of the key] / [key] / [value]
```

Notice

- Languages
  - C / Java
  - ***Please DO write comments in your code (be concise)***

- Tests
  - Your system MUST be implemented as ***separate modules***(Clients, Load Balancer, Handlers, Workers)
  - TAs will conduct ***module tests*** to evaluate your system
  - TAs will validate your system ***with other students' clients***

- Environments
  - Ubuntu 14.04.5 LTS (Trusty Tahr) 64-bit

Notice(Cont'd)

- We **highly** recommend that you use **vagrant**
  - Install VirtualBox: https://www.virtualbox.org/
  - Install Vagrant: https://www.vagrantup.com/downloads.html
  - Type the following commands
    - $ mkdir PATH_TO_BOX; cd PATH_TO_BOX
    - $ vagrant init ubuntu/trusty64
    - $ vagrant up
    - $ vagrant ssh   } You just need to use this after installation
  - *Borrowed by Sang Kil Cha's slide*

Grade

- Total *200pts*

  - **40pts** Documentation (Including README)

  - **20pts** Request and response in a client

  - **20pts** Load-balance requests to handlers in a load balancer

  - **20pts** Maintain key-value distributions in a handler

  - **20pts** Synchronize the distributions among handlers

  - **20pts** Manage a key-value table in a worker

Grade(Cont'd)

- **60pts *handle burst requests***
  - 10,000 (10pts) ~ 100,000 (60pts) requests per second
  - 100,000 *random unique* key-value requests
    - *Put* 50,000 pairs
    - *Get* 25,000 pairs (1/2 pairs exist), and *Del* 25,000 pairs (1/2 pairs exist)