EE324 Assignment 2

<Implementing Redis Web Server>

2020.09.29

Goal of the Assignment 2

- Implement Web Server that supports one of the widely used applications, Redis

- Improve the performance of your Web Server via multi-threads and event-based IO multiplexing

Assignment Due

- Part 1: Implement Redis Web Server **(Due ~10/13 11:59 PM)**
- Part 2: Improve Part 1 w/ multi-threads **(Due ~10/27 11:59 PM)**
- Part 3: Improve Part 1 w/ libevent **(Due ~10/27 11:59 PM)**

- **10% late penalty per day**
- Hard deadline: **23:59, Oct. 30. 2020 (Friday)**

Assignment Due

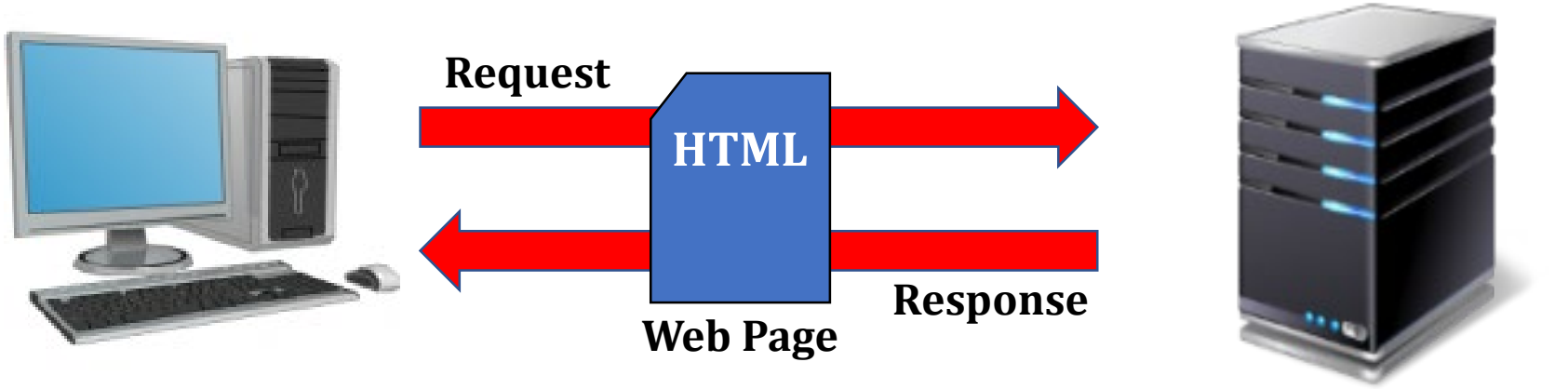
OCTOBER 2020						
Sun	Mon	Tue	Wed	Thu	Fri	Sat
27	28	29 HW2 Release	30	1	2	3
4	5	6	7	8	9	10
11	12	13 Part 1 Due	14	15	16	17
18	19	20 Mid-term Exam	21	22	23	24
25	26	27 Part 2, 3 Due	28	29	30	31

Background

- What is Web Server ?
- HTTP protocol
- What is Redis ?
- Redis Protocol

What is Web Server ?

- Stores, processes and delivers web pages to clients.
- Processes incoming network requests from clients over HTTP protocol.

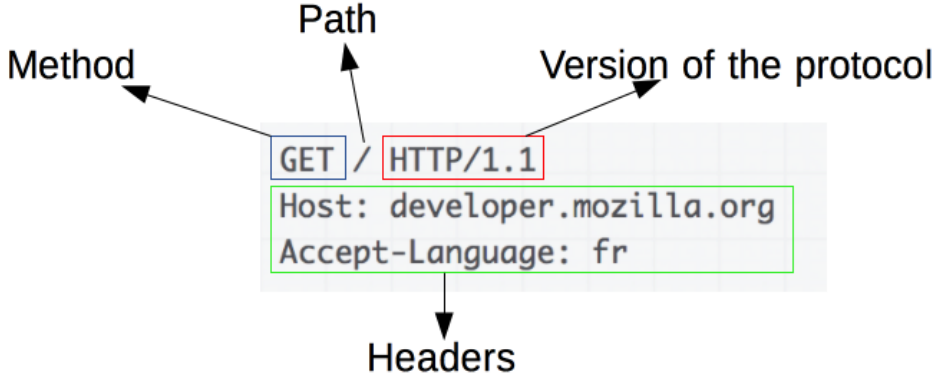


Client **Web Server**

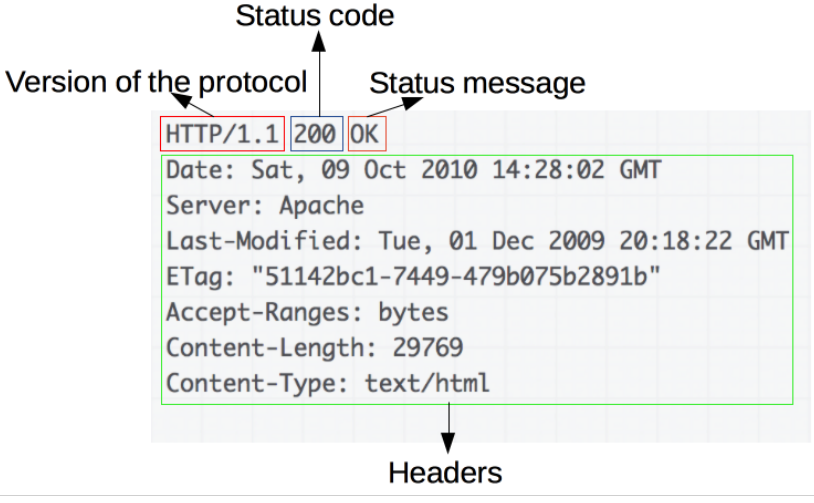
HTTP Protocol

- Defines how your client requests resources from a web server and how the server responds.
- Methods
  - GET : retrieve information identified by the request-url
  - POST : submit data to the server for updates

<HTTP GET Request Example>

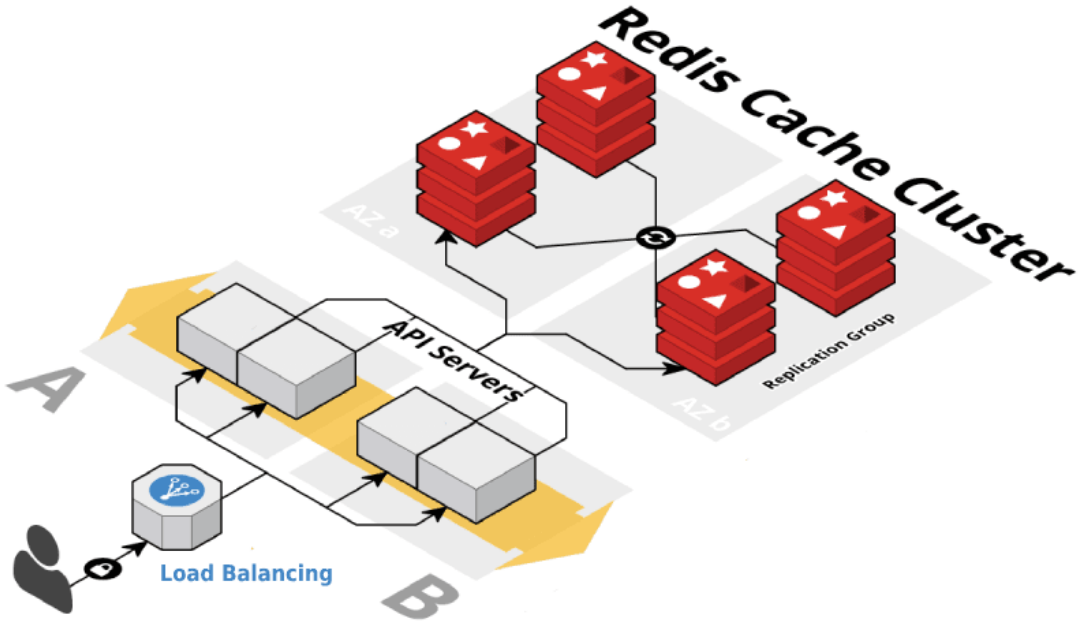


<HTTP GET Response Example>



What is Redis ?

- In-memory key-value data store, widely used as database and cache.
- Redis avoids seek time delays and access data in micro-seconds.
- Enables real-time apps such as Gaming, IOT, …



Redis Example Usage

- Commands
  - SET [key] [value] : set the string value of a key
  - GET [key] : get the value of a key
  - FLUSHALL : remove all keys from database

```
jaykim305@iris1:~/redis-6.0.6/src$ ./redis-server --port 6398
6:C 28 Sep 2020 10:46:20.151 # oO0OoO0OoO0Oo Redis is starting
6:C 28 Sep 2020 10:46:20.151 # Redis version=6.0.6, bits=64, c

6:C 28 Sep 2020 10:46:20.151 # Configuration loaded
6:M 28 Sep 2020 10:46:20.151 * Increased maximum number of ope
                Redis 6.0.6 (00000000/0) 64
                Running in standalone mode
                Port: 6398
                PID: 3717646
```

<Redis server>

```
(base) jaykim305@iris1:~/redis-6.0.6/src$ ./redis-cli -p 6398
127.0.0.1:6398> ping
PONG
127.0.0.1:6398> GET course
(nil)
127.0.0.1:6398> SET course EE324
OK
127.0.0.1:6398> SEt assign2 Redis_Web_Server
OK
127.0.0.1:6398> GET course
"EE324"
127.0.0.1:6398> GET assign2
"Redis_Web_Server"
127.0.0.1:6398> FLUSHALL
OK
127.0.0.1:6398> GET course
(nil)
127.0.0.1:6398>
```

<Redis command line interface>

Redis Protocol

For more info, refer to : https://redis.io/topics/protocol

<Rules>

- *: followed by the number of elements in the command array
- $: followed by the number of bytes composing the string, terminated by CRLF(\r\n)

<Example>

GET course

```
*2\r\n
$3\r\n
GET\r\n
$6\r\n
course\r\n
```

SET course EE324

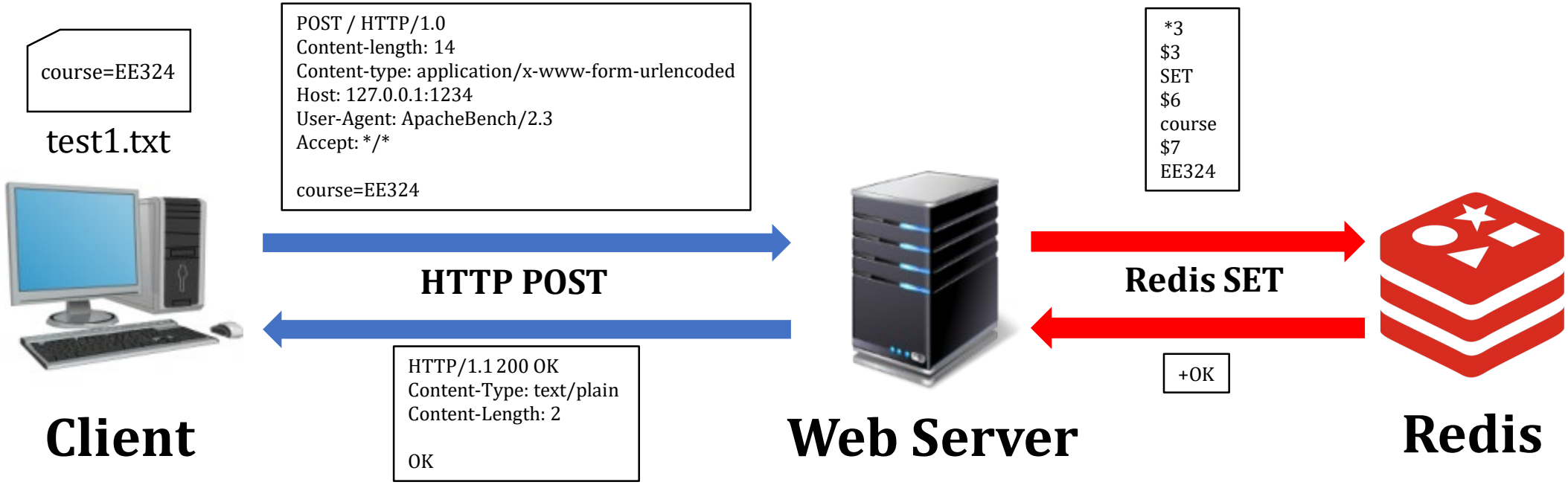
```
*3\r\n
$3\r\n
SET\r\n
$6\r\n
course\r\n
$5\r\n
EE324\r\n
```

Part 1: Implement Redis Web Server

- You need to implement Web Server that communicates with Redis.
  - Client <---- HTTP Protocol (GET, POST) ----> Web Server
  - Web Server <---- Redis Protocol (GET, SET) ---> Redis
  - Should handle multiple concurrent clients : use fork ( ) as used in HW1

Scenario1 : SET key-value via HTTP POST

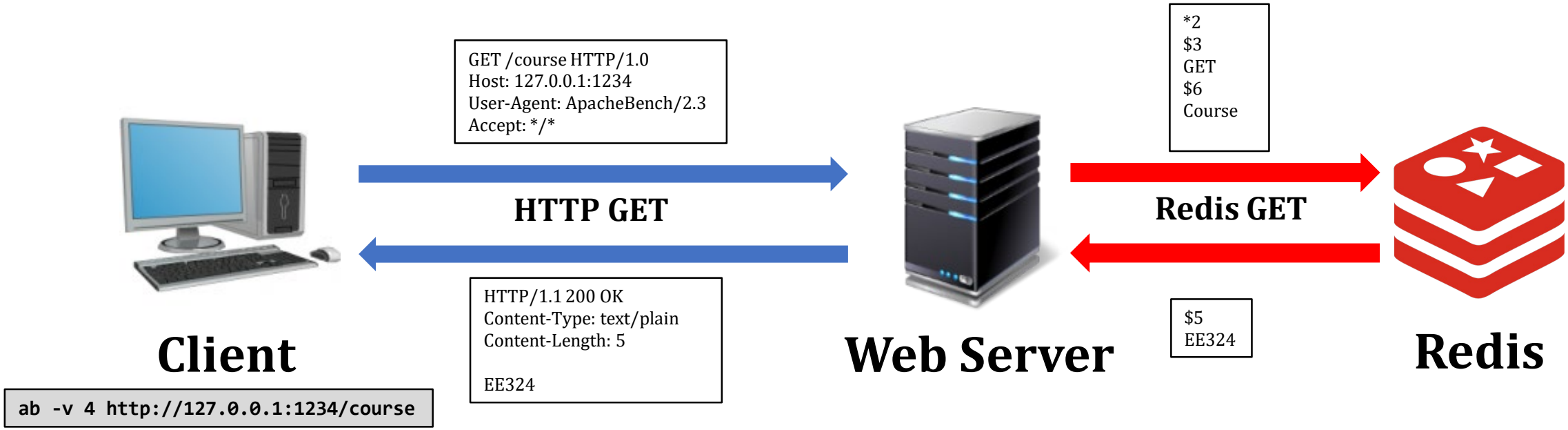
- Client wants to store “course” key as a value “EE324“ to the server




```
ab -v 4 -p test1.txt http://127.0.0.1:1234/
```

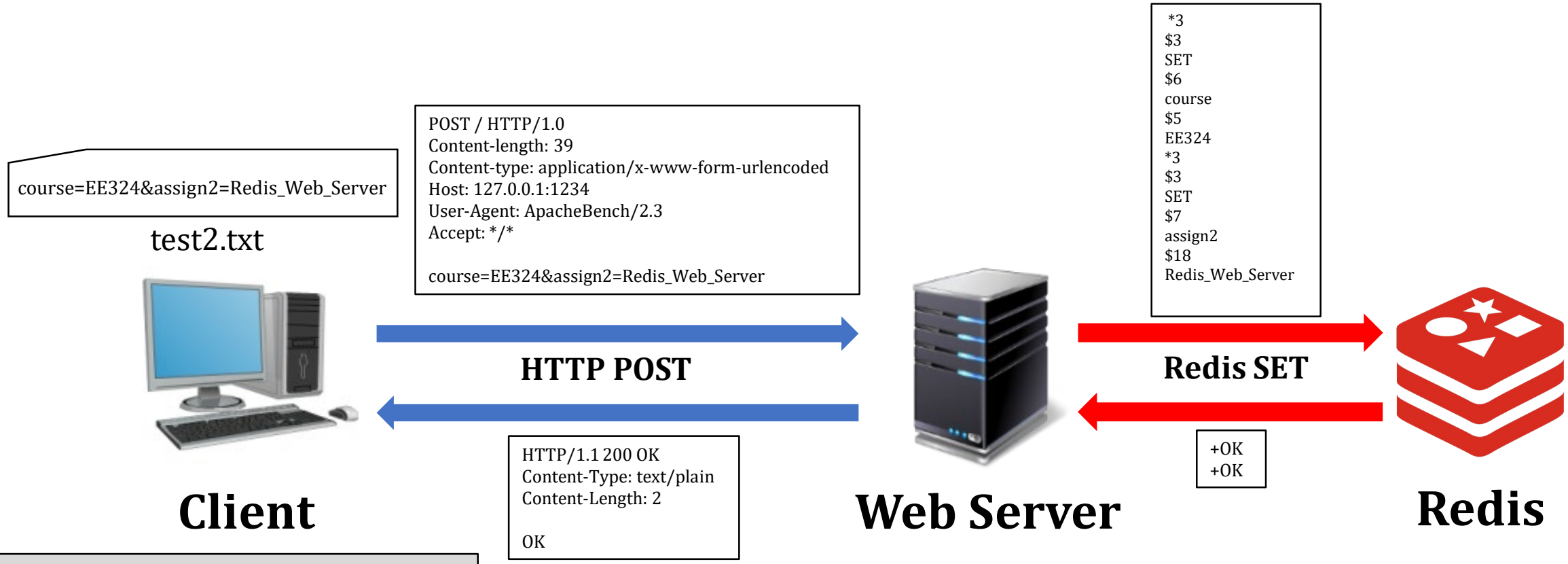
Scenario2 : GET key-value via HTTP GET

- Client wants to get the value of "course" from the server



Scenario3: Multiple SET key-value

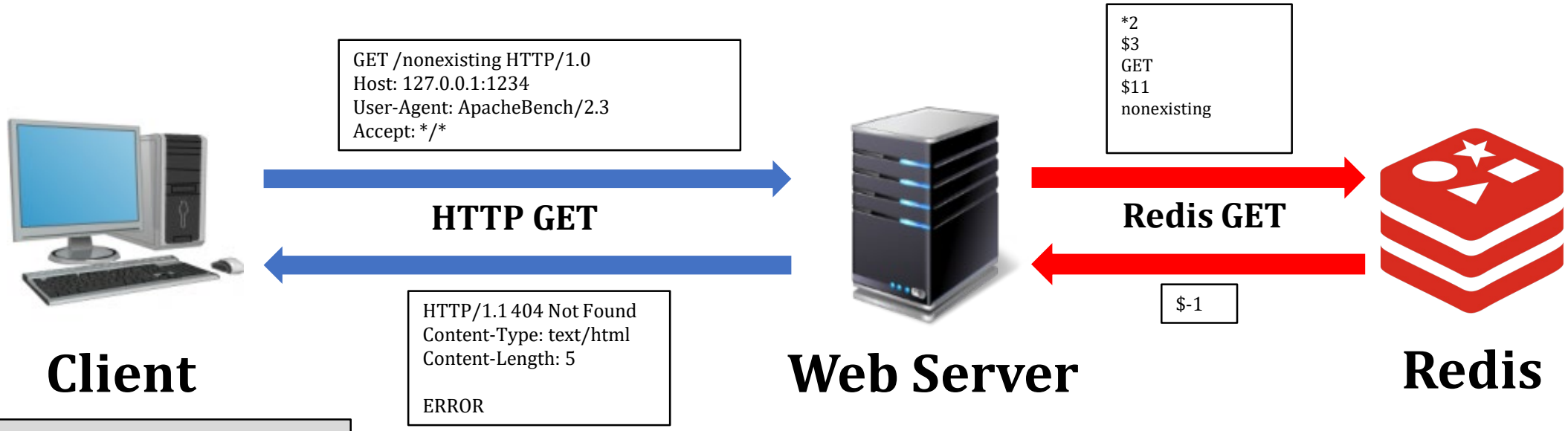
- Client wants to store multiple key-value pairs to the server {course=EE324} and {assign2=Redis_Web_Server}




```
ab -v 4 -p test2.txt http://127.0.0.1:1234/
```

Scenario4: GET non-existing key

- Client tries to get value of a key that is not stored




```
ab -v 4 http://127.0.0.1:1234/nonexisting
```

Required Functionality (1/2)

- Your Web Server should
  1. parse HTTP request (GET, POST).
  2. get the corresponding Redis command (i.e., GET, SET) and its key-value arguments.
  3. build Redis request following Redis protocol.
  4. retrieve Redis response from Redis Server.
  5. send response to client via HTTP response body.
  - handle error cases (non-existing keys, invalid Redis protocol)
  - single(multiple) key-value SET and single key-value GET

- Your Web Server should support multiple concurrent clients (up to 1K).

  - part 1 : fork, part 2: multi-threads, part 3: libevent

- Your server must at least handle key and value of any size.

Required Functionality (2/2)

- Your server binaries (webserver_fork, webserver_thread, webserver_libevent) should get
  [Port # of your Web Server], [IP address of your Redis Server], [Port # of your Redis Server] from stdin.
- Example)

```
bin/webserver_fork 1234 127.0.0.1 6398
```

Assignment Due

- Part 1: Implement Redis Web Server (fork based)
  (Due ~10/13 11:59 PM)

- Part 2: Improve Part 1 w/ multi-threads **(Due ~10/27 11:59 PM)**

- Part 3: Improve Part 1 w/ libevent **(Due ~10/27 11:59 PM)**

- **10% late penalty per day**
- Hard deadline: **23:59, Oct. 30. 2020 (Friday)**

Part 2: Improve Part 1 w/ multi-threads

- Threads are light-weight process and have less overhead than fork.
- Use `pthread.h`
- thread pool or using select is not required (this is for HW3)
- Simply spawn threads in a similar way as `fork()`
- Use `pthread_create(), pthread_join(), pthread_detach()`
- Compare the performance of Part 2 and 1 using Apache Bench (ab).

Part 3: Improve Part 1 w/ libevent

- Callback based event interface based on event mechanisms such as epoll, kqueue, select and poll.
- Use `event2/event.h`
- The whole process runs on single main thread
- Use `event_base_new()`, `event_base_dispatch()`, `event_new()`, `event_del()`, `event_add()`
- Compare the performance of Part 3 and 1 using Apache Bench (ab).

Assignment Due

- Part 1: Implement Redis Web Server **(Due ~10/13 11:59 PM)**
- Part 2: Improve Part 1 w/ multi-threads **(Due ~10/27 11:59 PM)**
- Part 3: Improve Part 1 w/ libevent **(Due ~10/27 11:59 PM)**

- **10% late penalty per day**
- Hard deadline: **23:59, Oct. 30. 2020 (Friday)**

Before you start

- Install Redis (https://redis.io/download)

```
$ wget http://download.redis.io/releases/redis-6.0.8.tar.gz
$ tar xzf redis-6.0.8.tar.gz
$ cd redis-6.0.8
$ make
```

  - Your `redis-server, redis-cli` binaries are located in src.
- Install libevent (this is not required for eelab machines)

```
sudo apt-get install libevent-dev
```

- Download template from gitlab (Clone repository)

```
https://gitlab.ee324.kaist.ac.kr/assignments/assignment-2.git
```

Grading

1. Part 1 : (50 pts)
   - Functionality – 25 pts
   - Robustness – 25 pts
2. Part 2 : (10 pts)
   - Functionality – 10 pts
3. Part 3: (40 pts)
   - Functionality – 10 pts
   - Robustness – 10 pts
   - Performance – 20 pts
4. Bonus points (5 pts)

(+5pt) Bonus: Percent-encoding

- Key and value sent from client is URL encoded (percent-encoding)
  - Content-type: application/x-www-form-urlencoded
  - https://en.wikipedia.org/wiki/Percent-encoding

`hello, world!` ⇄ `hello%2C%20world%21`

🤮 ⇄ `%F0%9F%8D%BA`

- Decode the request when you GET/SET value on Redis
- Example (SET)
  - Input from HTTP client: `R%25D=staff%40ee324`
  - Parsed key and value: {R&D, staff@ee324}
  - You should not store "R%25D" as is!
- **Note decoded data can be binary!**
- You can use helper function in `src/urlencode.c` (if you want.)

Test Cases

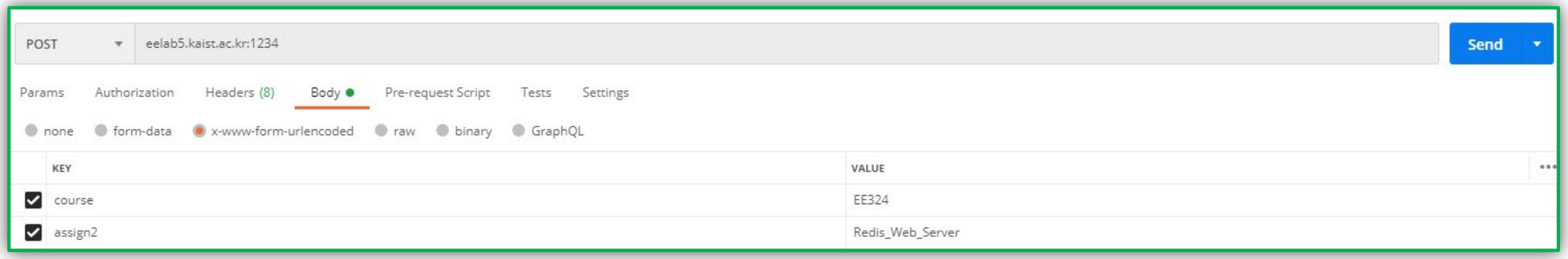
- We provide `test.py`
- We will use more tricky test cases not included in the `test.py`

- Usage:
  - `test/test.py --redis-port [port] --test [test case] [binary]`
  - `test case = {func_get, func_set_simple, func_set_multiple}`
  - `binary = {bin/webserver_fork, bin/webserver_thread, bin/webserver_libevent}`

Useful Tools for Developing

- ApacheBench (ab)
  - We will use ApacheBench (ab) as a benchmark to test the performance of your webservers.

- Postman (https://www.postman.com/downloads/)



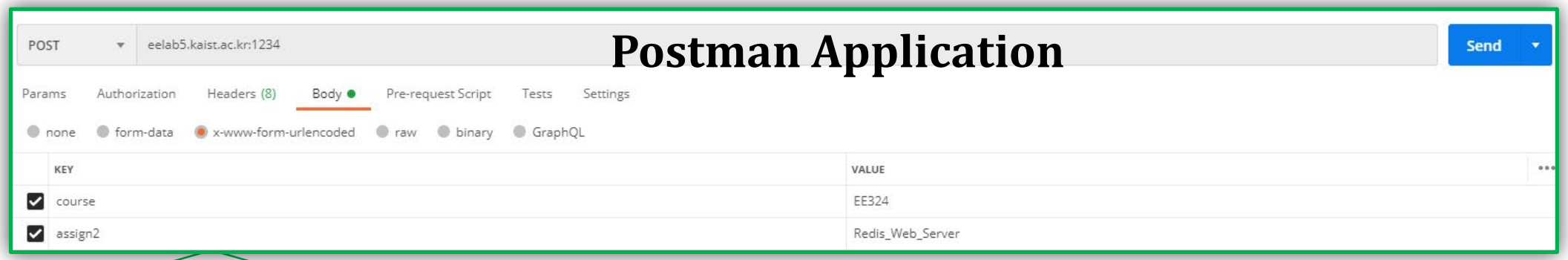
Performance Test using Apache Bench (ab)

1. ab with 1 concurrency 1,000 requests
   - `$ ab –n 1000 server_ip:server_port`

2. Ab with 100 concurrency, 10,000 requests
   - `$ ab –c 100 -n 10000 server_ip:server_port`

- We will test your program up to 1K requests with 1K concurrency.
- Other options (type ab for more options)
  - p : postfile
  - v : verbosity
  - T : content-type
  (Note for Bonus: you should set this to application/x-www-form-urlencoded)

Running Example



Postman Application

```
ab -v 4 -p test2.txt http://eelab5.kaist.ac.kr:1234/
```

Apache Bench

```
./webserver_fork 1234 127.0.0.1 6398
```

HTTP Protocol

Redis Protocol

Client

Web Server (eelab5)

Redis

```
./redis-server --port 6398
```

References

- libevent Document: http://www.wangafu.net/~nickm/libevent-book/, http://www.wangafu.net/~nickm/libevent-2.0/doxygen/html/dir_db160b4728e6067cf5f9cc14ec42c79d.html

- EE324 Assignment 2 Document : https://www.notion.so/jaykim305/EE324-Assignment-2-Implementing-Redis-Web-Server-db47f65642454e749b26d7c0e174272f

Submission (1/2)

- You should upload your code on GitLab
- We will download your project when the due is over
- In the project, there should be following files
  - README.md
  - Makefile
  - src/webserver_fork.c
  - src/webserver_thread.c
  - src/webserver_libevent.c
  - and some test scripts…

Submission (2/2)

- You must include the following contents in the README.md
  - the date and time of your submission in README.md

  - Performance measurements from Part2 (not Due on Part1)
  - Performance measurements from Part3 (not Due on Part1)

  - Brief explanation and reasonings about your results

- Feel free to change Makefile, test.py, or gitlab.ci.yml
  (As long as your Makefile generates binaries with designated names)

Performance measurements example

```
Document Path:          /ee324
Document Length:        8002 bytes

Concurrency Level:      1000
Time taken for tests:   8.780 seconds
Complete requests:      1000
Failed requests:        0
Total transferred:      8068000 bytes
HTML transferred:       8002000 bytes
Requests per second:    113.89 [#/sec] (mean)
Time per request:       8780.228 [ms] (mean)
Time per request:       8.780 [ms] (mean, across all concurrent requests)
Transfer rate:          897.35 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0  297 695.7     20    3004
Processing:     2  630 879.5    220    6416
Waiting:        2  629 879.5    220    6416
Total:          2  926 1198.1    622    7417

Percentage of the requests served within a certain time (ms)
  50%    622
  66%   1100
  75%   1315
  80%   1349
  90%   2601
  95%   3414
  98%   4205
  99%   7411
 100%   7417 (longest request)
```

payload size (8kB)

Time taken for tests

Completed requests

99%-ile completion time

- Fill README.md

Performance measurements

Set 8kB-sized value into a specific key. Measure the time for running 1,000 concur

- Part 1
  - Completed requests: ____
  - Time taken for test: ____ ms
  - 99%-ile completion time: ____ ms
- Part 2
  - Completed requests: ____
  - Time taken for test: ____ ms
  - 99%-ile completion time: ____ ms
- Part 3
  - Completed requests: ____
  - Time taken for test: ____ ms
  - 99%-ile completion time: ____ ms

Briefly compare the performance of part 1 through 3 and explain the results.

Test Environment

- Language: C or C++
- Test O/S: Ubuntu 16.04 LTS (Xenial Xerus), Ubuntu 18.04.5 LTS (Bionic Beaver), 20.04 LTS (focal fossa) 64bit
- **We will not consider your compilation and execution problems due to the different OS versions.**

Plagiarism

- You can discuss with your colleagues, but you should turn in your own programs
- Copy and Paste
  - Will run plagiarism detection on source code
  - “Copy and paste” codes will get severely penalized
  - If detected, 0 point for all assignments (both providers and consumers)
  - But you will have a chance to defend yourself

Questions

- Please use Piazza for asking any questions.
- Please read the documents and the slide carefully before you ask any questions.