# 15-319 / 15-619
# Cloud Computing

Recitation 6

February 17th & 19th, 2015

# Overview

- **Administrative issues**
  Office Hours, Piazza guidelines
- **Last week's reflection**
  Project 2.2, OLI unit 3 module 6 and 7
- **This week's schedule**
  - Project 2.3 - Feb 22nd
  - Unit 3 - Module 8 - Resource Virtualization - CPU
- **Demo**

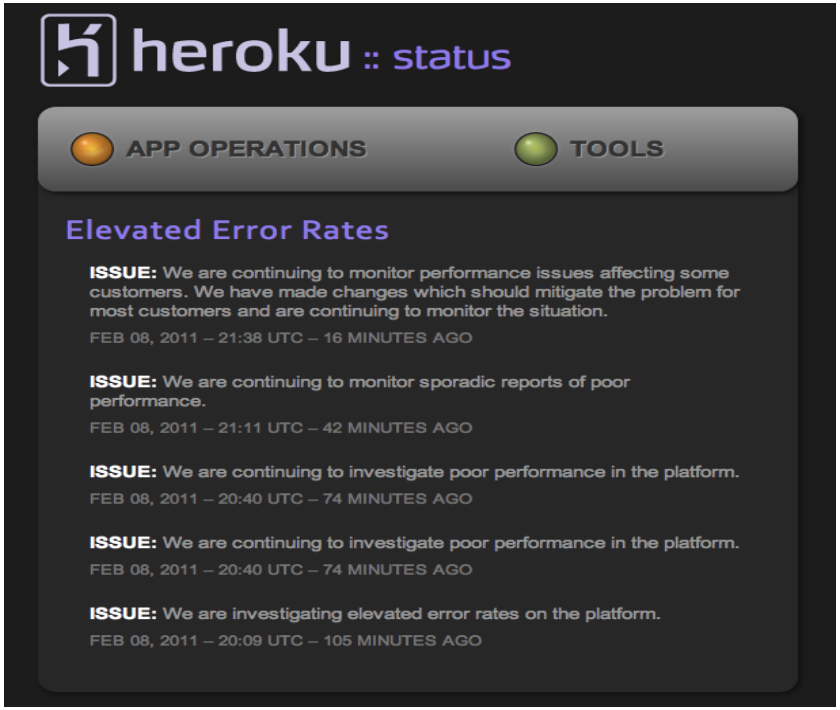
# Announcements

- Monitor AWS expenses regularly
  - Check your bill frequently (use Cost Explorer and filter by tags).
- Terminate your resources when not in use
  - Stop still costs EBS money ($0.1/GB/Month)
  - Amazon EC2 and Amazon Cloudwatch fees for monitoring, ELB
  - Autoscaling group - no additional fees
- Use spot instances

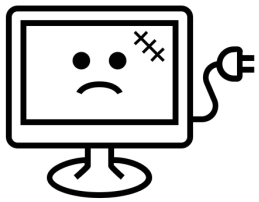
# Last Week's Reflection

- Content
  - Unit 3 - Modules 6 and 7  Virtualizing Resources of the cloud
  - Quiz 2 completed
- EC2 APIs
  - Amazon CLI, Java, Python
- Load Balancing and Autoscaling
  - Experience ASG (horizontal scaling) on AWS
  - Manage cloud resources and deal with failures using programs.

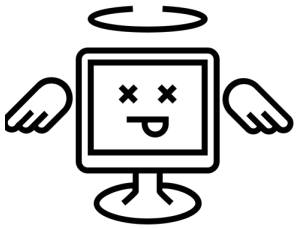
# Types of Failures




Transient Failure



Permanent Failure

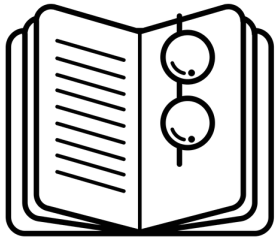


# Project 2.2

- Manual Grading: 20 Points are for the code
  - Always make sure that your code is readable
    - Follow style presented in Recitation 2

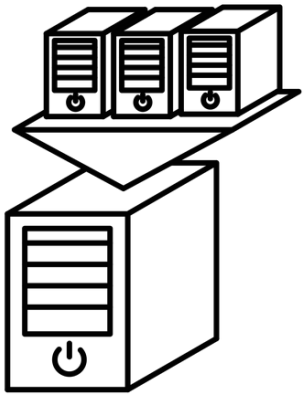
| Violation | Penalty of the project grade |
|---|---|
| Spending more than $10 for this project phase | -10% |
| Spending more than $20 for this project phase | -100% |
| Failing to tag all your resources(EC2 instances, ELB, ASG) for this project | -10% |
| Submitting your AWS credentials in your code for grading | -10% |
| Using instances other than m1/m3.small/medium/large | -10% |

# This Week: Content



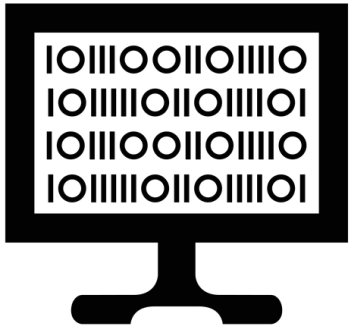
- UNIT 3: Virtualizing Resources for the Cloud
  - Module 6: Introduction and Motivation
  - Module 7: Virtualization
  - **Module 8: Resource Virtualization - CPU**
  - Module 9: Resource Virtualization - Memory
  - Module 10: Resource Virtualization – I/O
  - Module 11: Case Study

# Resource Virtualization - CPU



- Virtualizing an ISA
- Full Virtualization and Paravirtualization
- Emulation
- Virtual CPU

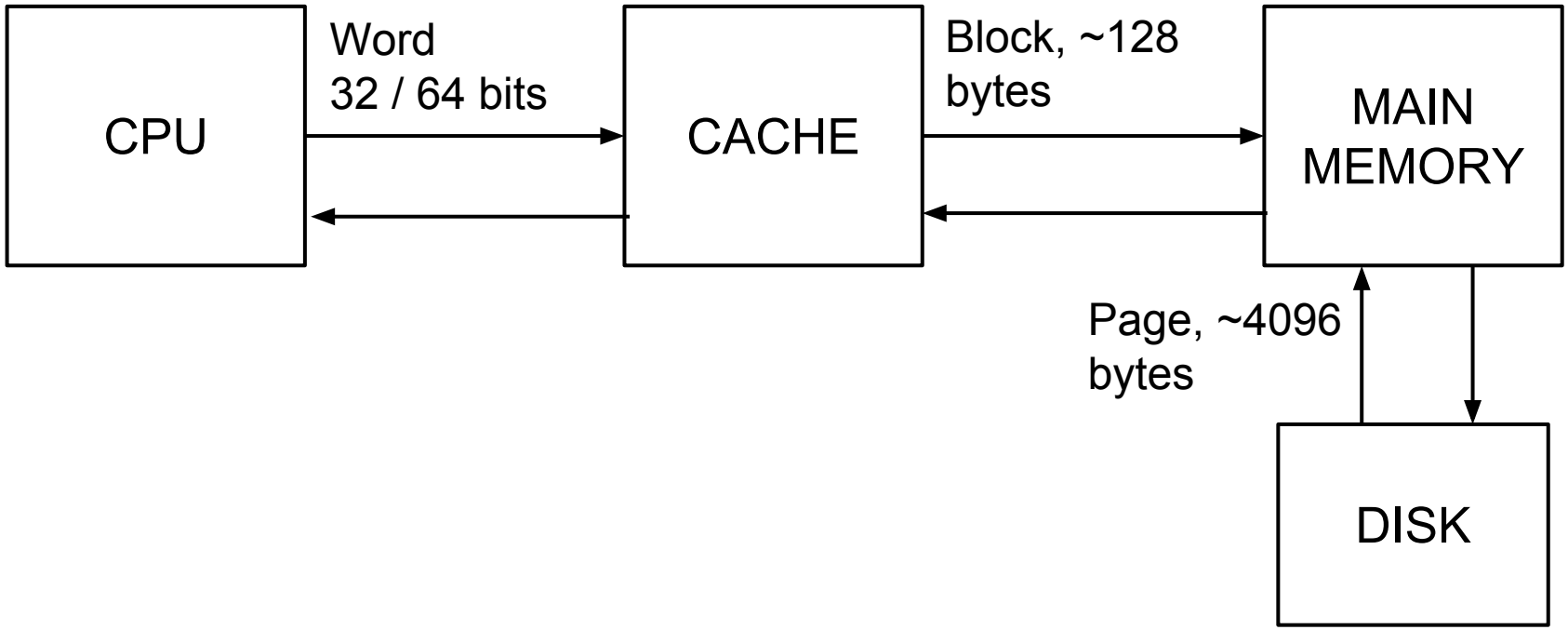
# This Week: Project



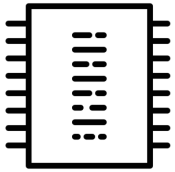
- P2.1: Introduction and APIs
  - MSB Recruitment Exam
- P2.2: Autoscaling and Elastic Load Balancing
  - Junior System Architect at the MSB
- P2.3: Advanced Scaling Concepts: Caching
  - Speed up a web-service using caching

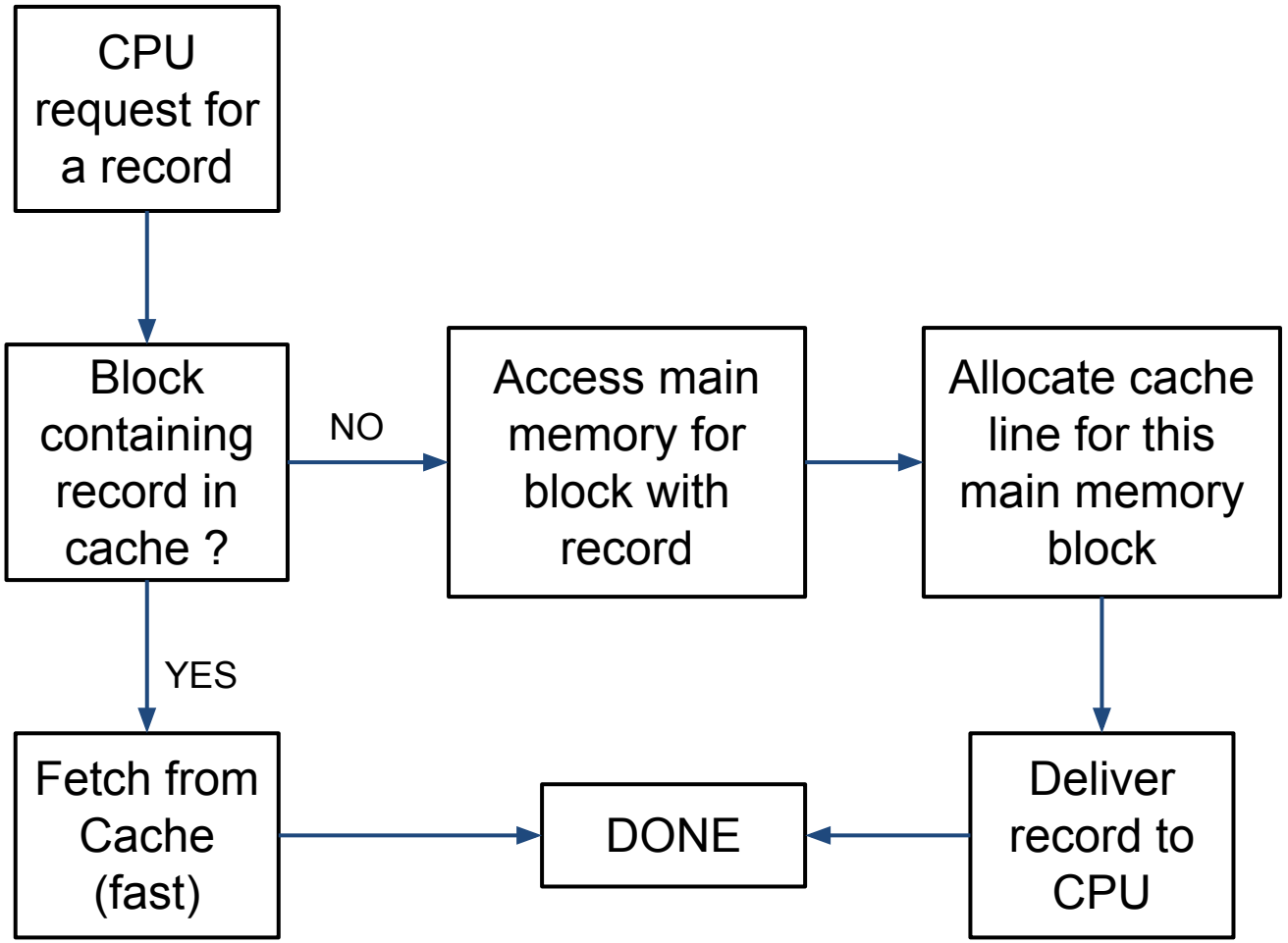
# Caching - I

Old idea, used across many layers within a single machine and across machines. An example of caching between CPU and MM:



# Caching - II



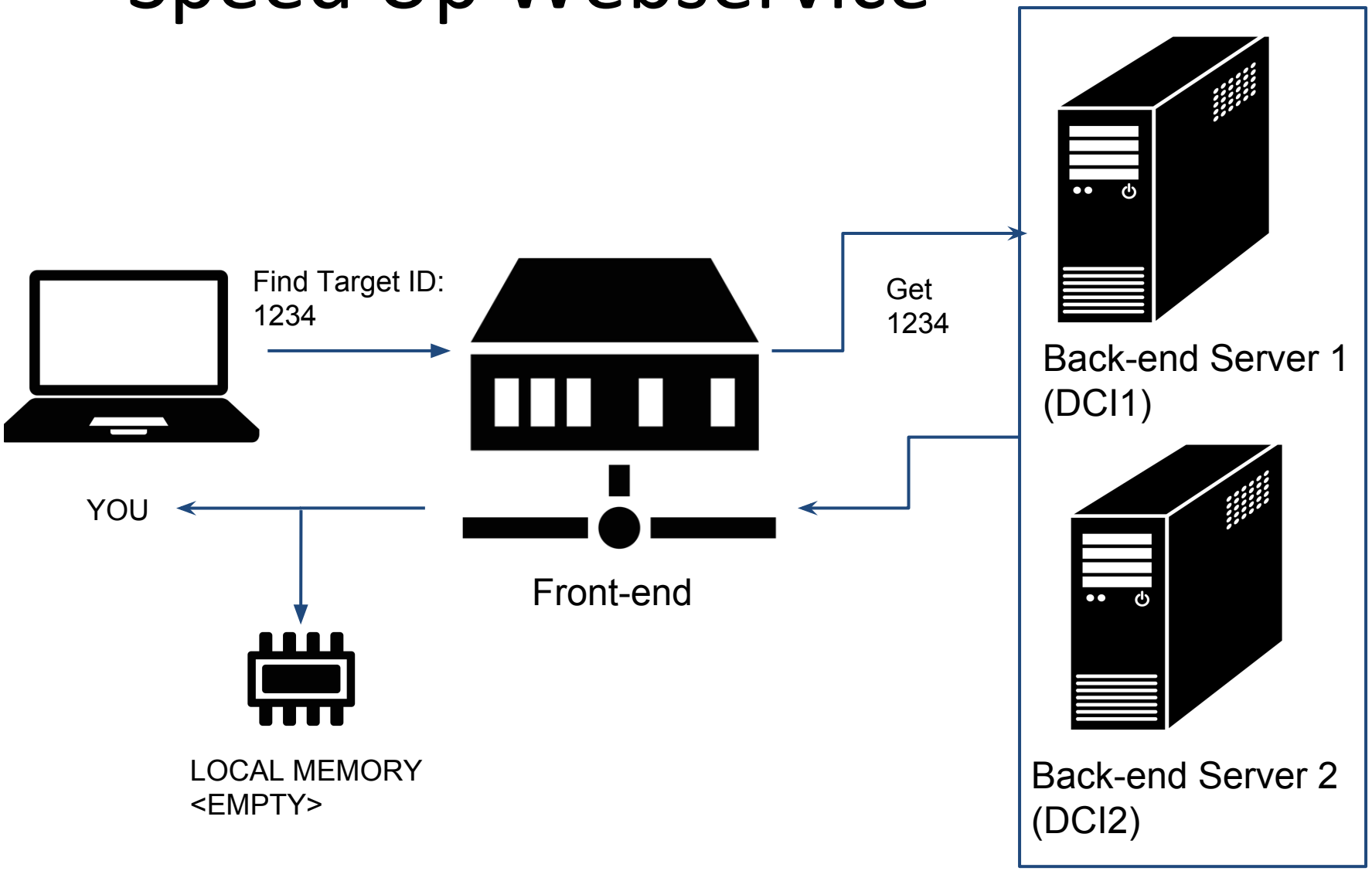

CPU request for a record
Block containing record in cache ?
NO
Access main memory for block with record
Allocate cache line for this main memory block
YES
Fetch from Cache (fast)
DONE
Deliver record to CPU

# Caching - III

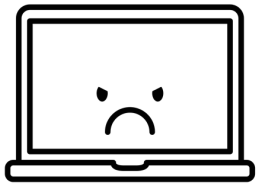
- Exploiting locality
  - Temporal locality
  - Spatial locality

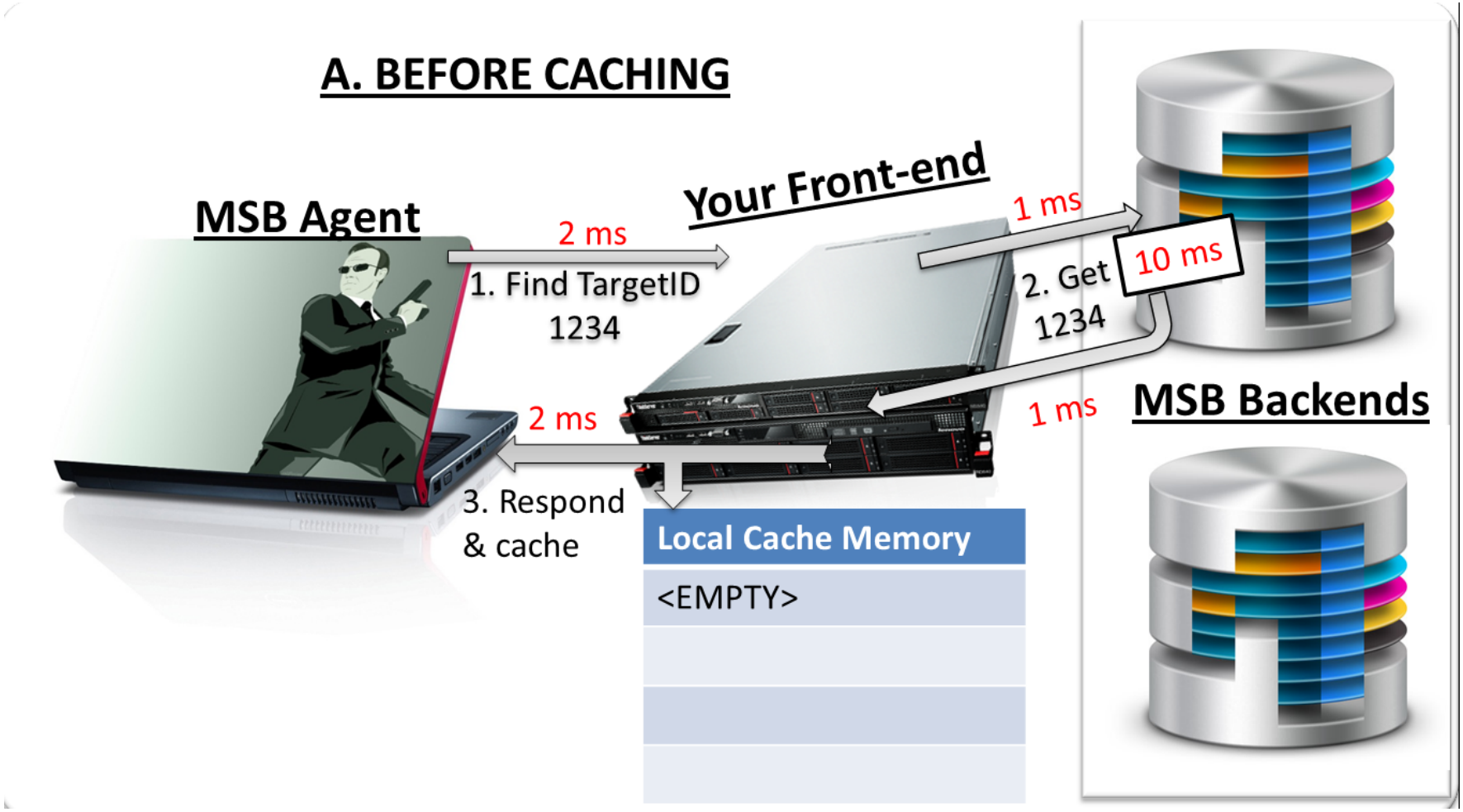
- Things to consider in caching
  - Which data to fetch?
  - Which block to invalidate during writing?
  - Eviction policy, when you run out of space?

# P2.3 - This Week
# Speed Up Webservice

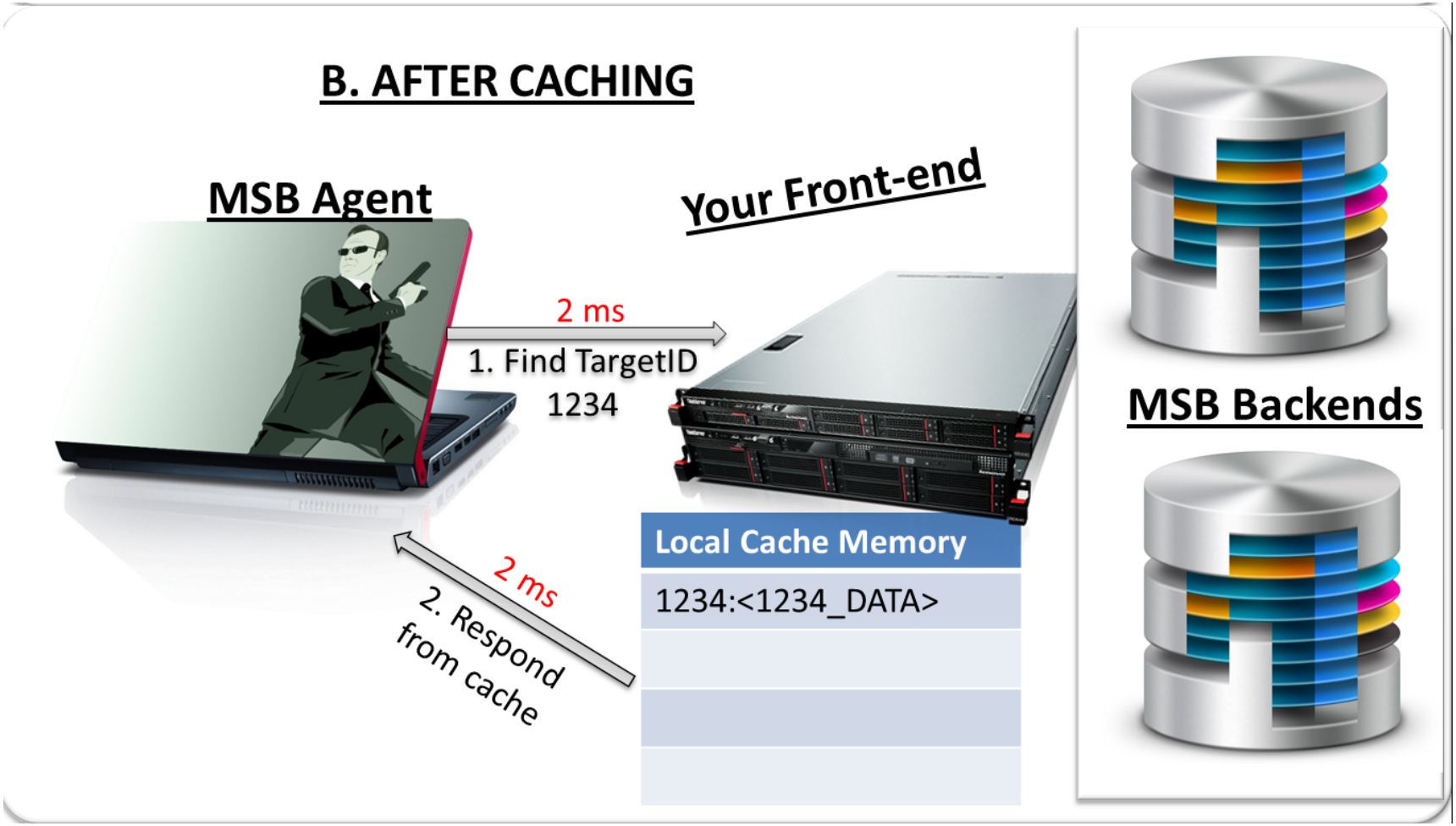


# Web Service: Before Caching





# Web Service: After Caching



# P2.3 - what you have to do

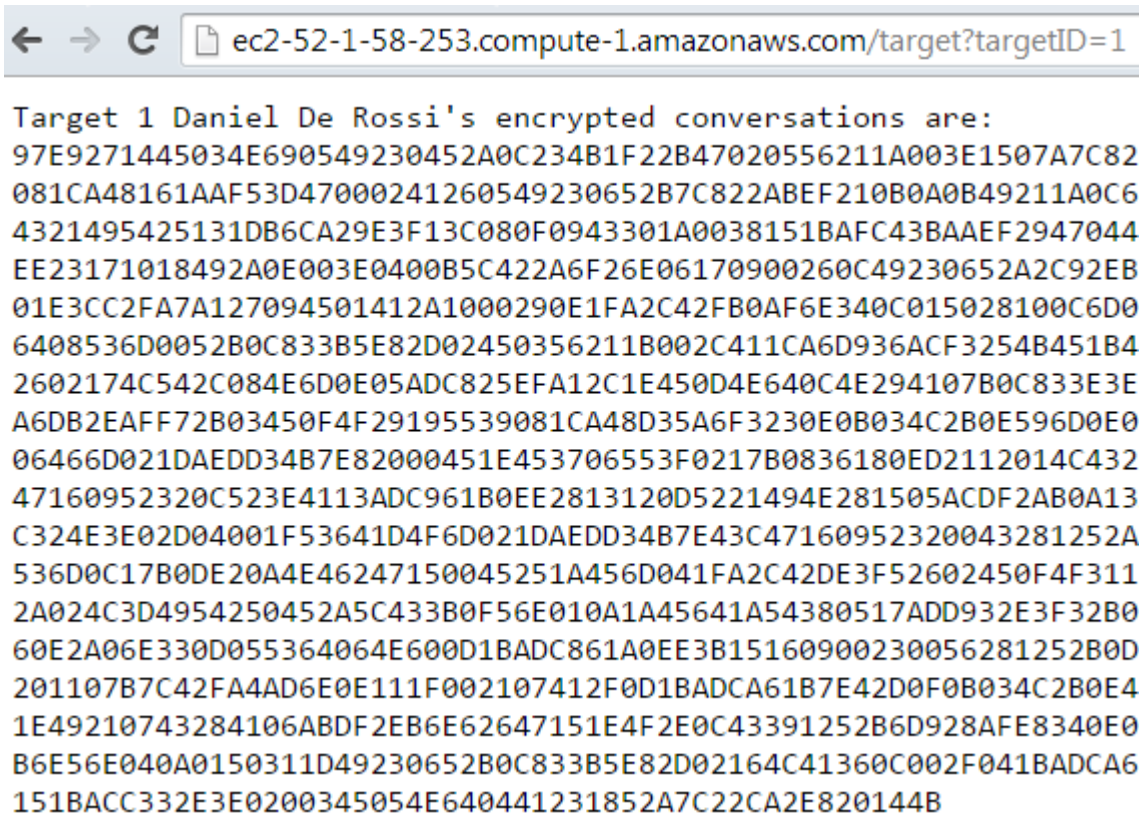
- Speed up the web-service using caching
  - Size of cache on backend and frontend is predetermined
  - Learn the characteristics of the traces
  - Skeleton code provided gives low RPS
    - Trace 1: Skeleton code RPS ~ 65
      Target RPS ~ 1300

Note: You may only cache up to 1000 records at a time on the front end.

# Querying a single record from the DCI

URL: `<DCI-DNS>/target?targetID=1`

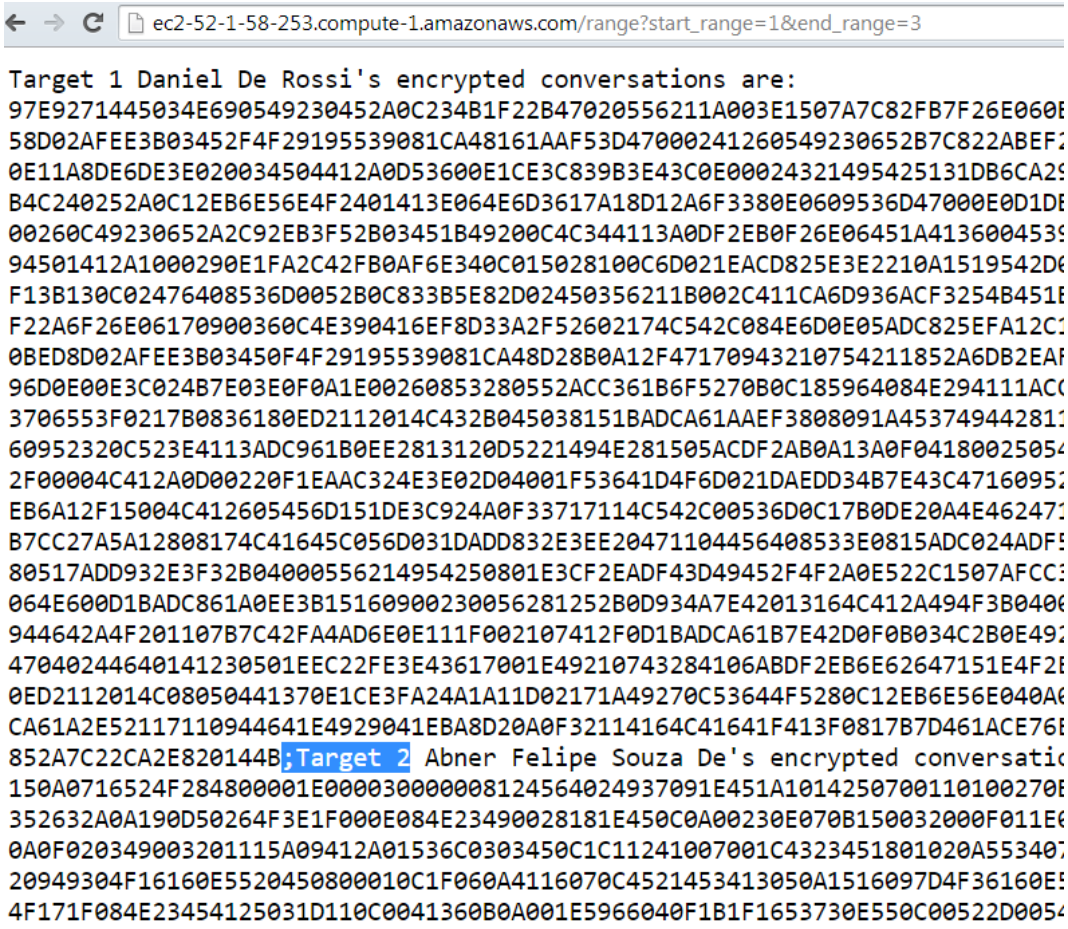
Response:



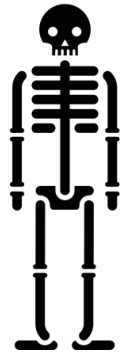
# Querying multiple records from the DCI

URL: `<DCI-DNS>/range?start_range=1&end_range=3`

Response:



# Skeleton Code Provided



Skeleton code in java (**`MSB.java`**) and python (**`MSB.py`**)

- You can get full score by optimizing this function.
- Feel free to optimize other parts too!

```
/*
 * retrieveDetails - you have to modify this function to achieve a higher RPS value
 * Input: the targetID
 * Returns: The result from querying the database instance
 */
private String retrieveDetails(String targetID) {
        try{
                return sendRequest(generateURL(0, targetID));
        } catch (Exception ex){
                System.out.println(ex);
                return null;
        }
}
```

# Resources

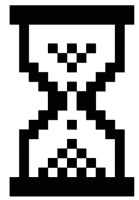

1. Brewer, Eric A. "Lessons from giant-scale services." Internet Computing, IEEE 5.4 (2001): 46-55.

2. Sivasubramanian, Swaminathan, et al. "Analysis of caching and replication strategies for web applications." Internet Computing, IEEE 11.1 (2007): 60-66.

3. Karger, David, et al. "Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the World Wide Web." Proceedings of the twenty-ninth annual ACM symposium on Theory of computing. ACM, 1997.

4. Fan, Bin, et al. "Small cache, big effect: Provable load balancing for randomly partitioned cluster services." Proceedings of the 2nd ACM Symposium on Cloud Computing. ACM, 2011.

# Latency comparison with Caching



Query the frontend web-service:

Get record for targetID = 2000

| | |
|---|---|
| Without the record cached: | 0.035s |
| With the record cached at backend: | 0.020s |
| With the record cached at front-end: | 0.004s |

# Project 2.3 Hints - 1

- Caches will be effective if the memory accesses exhibit good locality of references

- Temporal locality - resources accessed in close points in time.

- Spatial Locality - likelihood that a resource will be accessed that is near a resource that was just accessed

# Project 2.3 Hints - 2

- Read project description more than once
- Think about workflow before starting
- Read the Hints section! It *might* be useful
- When you are working on the caching code, stop the load generator and data center instances to save costs.
- Start early.

# Project 2.3 Penalties

| Violation | Penalty of the project grade |
|---|---|
| Spending more than $10 for this project phase | -10% |
| Spending more than $20 for this project phase | -100% |
| Failing to tag all your resources for this project | -10% |
| Submitting your AWS credentials in your code for grading | -10% |
| Using instances other than the ones specified | -100% |
| Caching more than 1000 records in the front end | -100% |

# Upcoming Deadlines

- Project 2.3: Advanced Scaling Concepts - Caching
  Due: 02/22/15 11:59PM Pittsburgh
- Unit 3: Virtualization Resource for the Cloud Module 8: Resource Virtualization - CPU
- 15619Project Team Formation Deadline
  Due: Friday 02/20/15 11:59PM Pittsburgh
- Quiz 3: Unit 3 - Virtualizing Resources
  Due: Next Week (Start reading, long unit)

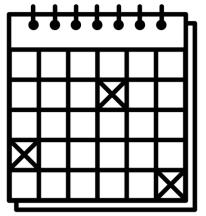
# Upcoming Deadline

Form your 15619Project teams and submit your team info to:

http://bit.ly/1EFAsdO

- Once submitted, teams are final and binding

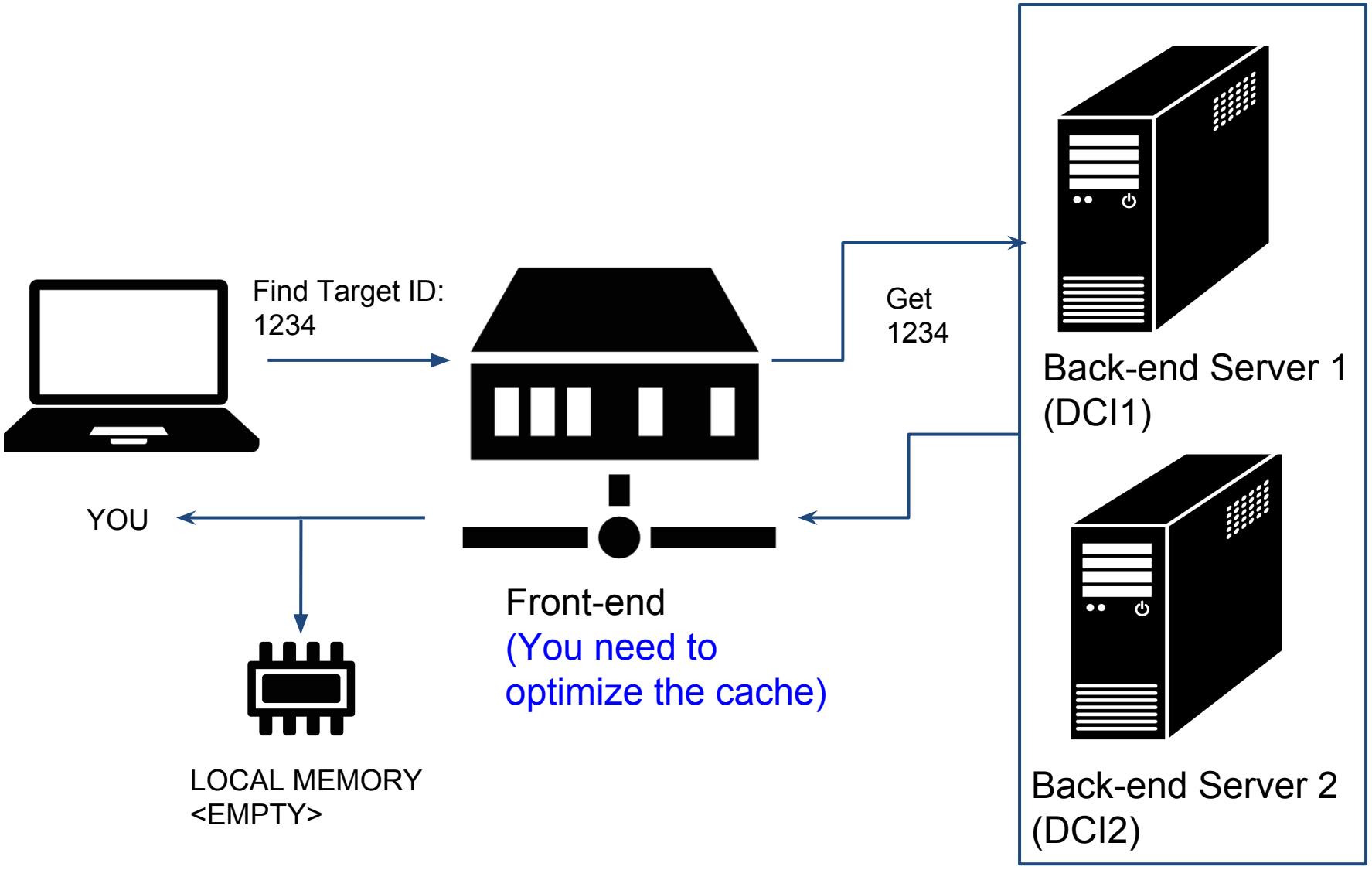
# 15619 Project Time Table



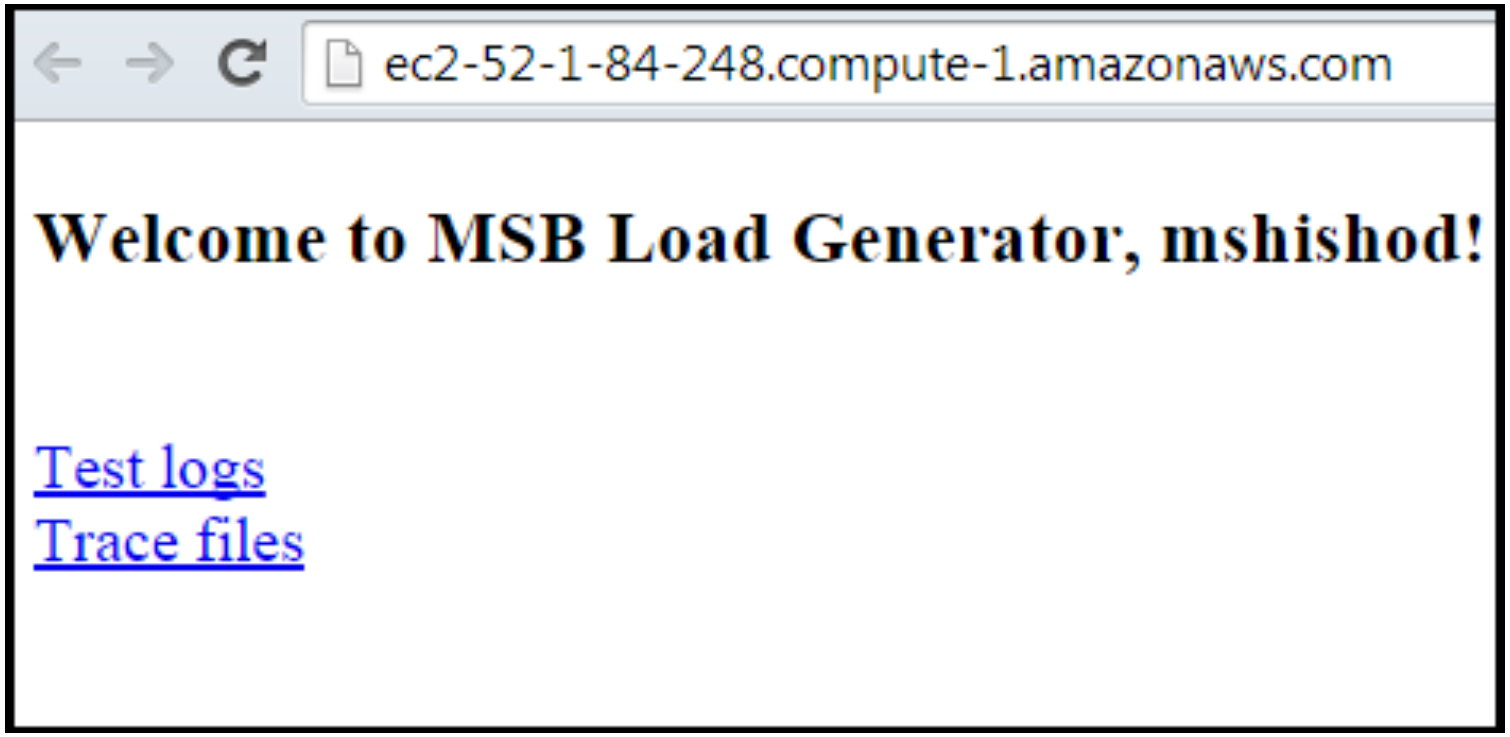
| Phase (and query due) | Start | End |
|---|---|---|
| Phase 1 Part 1<br>● Q1 (due), Q2 (not yet due) | Thursday 2/26/2015<br>00:00:01 EST | Wednesday 3/4/2015<br>23:59:59 EST |
| Phase 1 Part 2<br>● Q1, Q2 (due) | Thursday 3/26/2015<br>00:00:01 EST | Wednesday 3/18/2015<br>23:59:59 E**D**T |
| Phase 2<br>● Q1, Q2, Q3, Q4 | Thursday 3/19/2015<br>00:00:01 E**D**T | Wednesday 3/25/2015<br>16:59:59 EDT |
| Phase 2 Live Test<br>● Q1, Q2, Q3, Q4 | Wednesday 3/25/2015<br>18:00:01 EDT | Wednesday 3/25/2015<br>23:59:59 EDT |
| Phase 3<br>● Q1, Q2, Q3, Q4, Q5, Q6 | Thursday 3/26/2015<br>00:00:01 EDT | Wednesday 4/8/2015<br>18:59:59 EDT |
| Phase 3 Live Test<br>● Q1, Q2, Q3, Q4, Q5, Q6 | Wednesday 4/8/2015<br>20:00:01 EDT | Wednesday 4/8/2015<br>23:59:59 EDT |

There will also be a report due at the end of each phase, where you are expected to discuss optimizations

# P2.3 - This Week



# Project 2.3: Load Generator UI



# Project 2.3: Data Center UI