15-319 / 15-619
Cloud Computing

Recitation 6

February 17th, 2016

Overview

- **Administrative issues**
  Office Hours, Piazza guidelines
- **Last week's reflection**
  Project 2.2, OLI unit 3 module 7, 8 and 9
- **This week's schedule**
  - Project 2.3 - Sunday, February 21st
  - Unit 3 - Modules 10, 11 and 12
  - Quiz 5 - Friday, February 19th
- **Demo**

Project 2.2

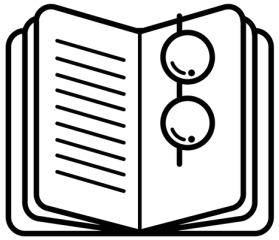
- Manual Grading: 20% for code
  - Always make sure that your code is readable
    - Follow style presented in Recitation 2

Project Grading Penalties

Besides the penalties mentioned in recitation and/or on Piazza, penalties accrue for the following:

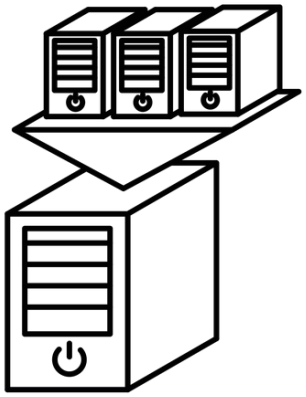
Violation	Penalty of the project grade
Spending more than $5 for this project phase on AWS	-10%
Spending more than $10 for this project phase on AWS	-100%
Failing to tag all your AWS resources in either parts (EC2 instances, etc) for this project	-10%
Submitting your AWS credentials in your code for grading	-100%
Submitting your Azure credentials in your code for grading	-100%
Submitting the Azure part with AWS instances or the AWS part with Azure VMs.	-100%
Attempting to hack/tamper the autograder in any way	-100%
Using virtual machines other than Standard_A1(DC) and Standard_D1(LG and LB) in the Azure part	-100%

This Week: Content



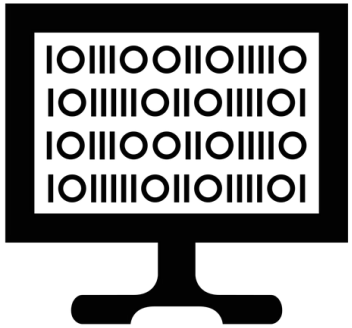
- UNIT 3: Virtualizing Resources for the Cloud
  - Module 7: Introduction and Motivation
  - Module 8: Virtualization
  - Module 9: Resource Virtualization - CPU
  - Module 10: Resource Virtualization - Memory
  - Module 11: Resource Virtualization – I/O
  - Module 12: Case Study
  - Module 13: Storage and Network Virtualization

OLI, Unit 3: Modules 10, 11, 12



- Understand two-level page mappings from virtual to real to physical
- Learn how memory is overcommitted and reclaimed using ballooning
- Study how I/O requests are intercepted at different interfaces
- Map these concepts into your practical exploration with AWS

This Week: Project



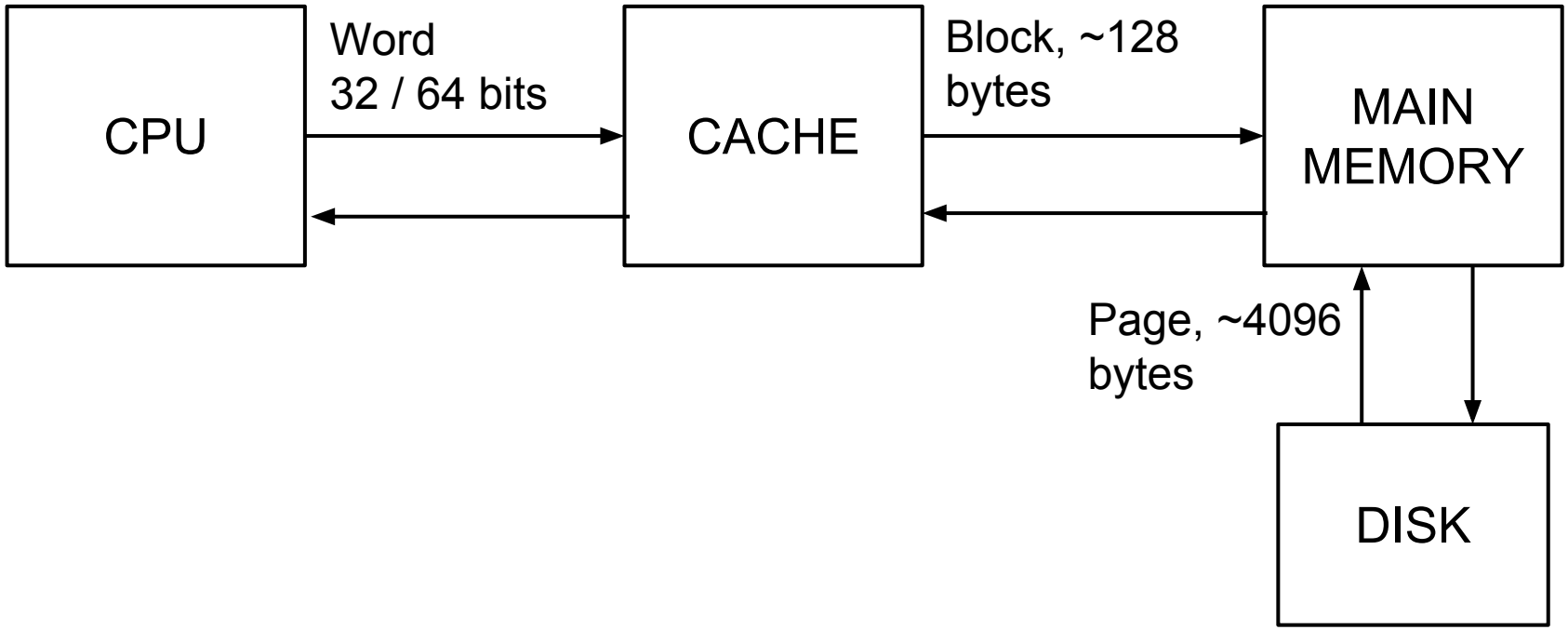
- P2.1: Horizontal Scaling and Advanced Resource Scaling
  - MSB Recruitment Exam
- P2.2: The Internals and Strategies of a Load Balancer
  - Junior System Architect at the MSB
- P2.3: Advanced Scaling Concepts: Caching
  - Speed up a web-service using caching

Caching

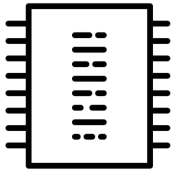
- Technique for hiding latency
- Move data closer (faster) to client
- Leverages hierarchies of access speeds
- Benefits from bias towards locality of access
- Generally a transparent abstraction layer
- Introduces many different inconsistencies
  - Page replacement and invalidation
  - Cache coherence
- Highly beneficial, but needs careful tuning

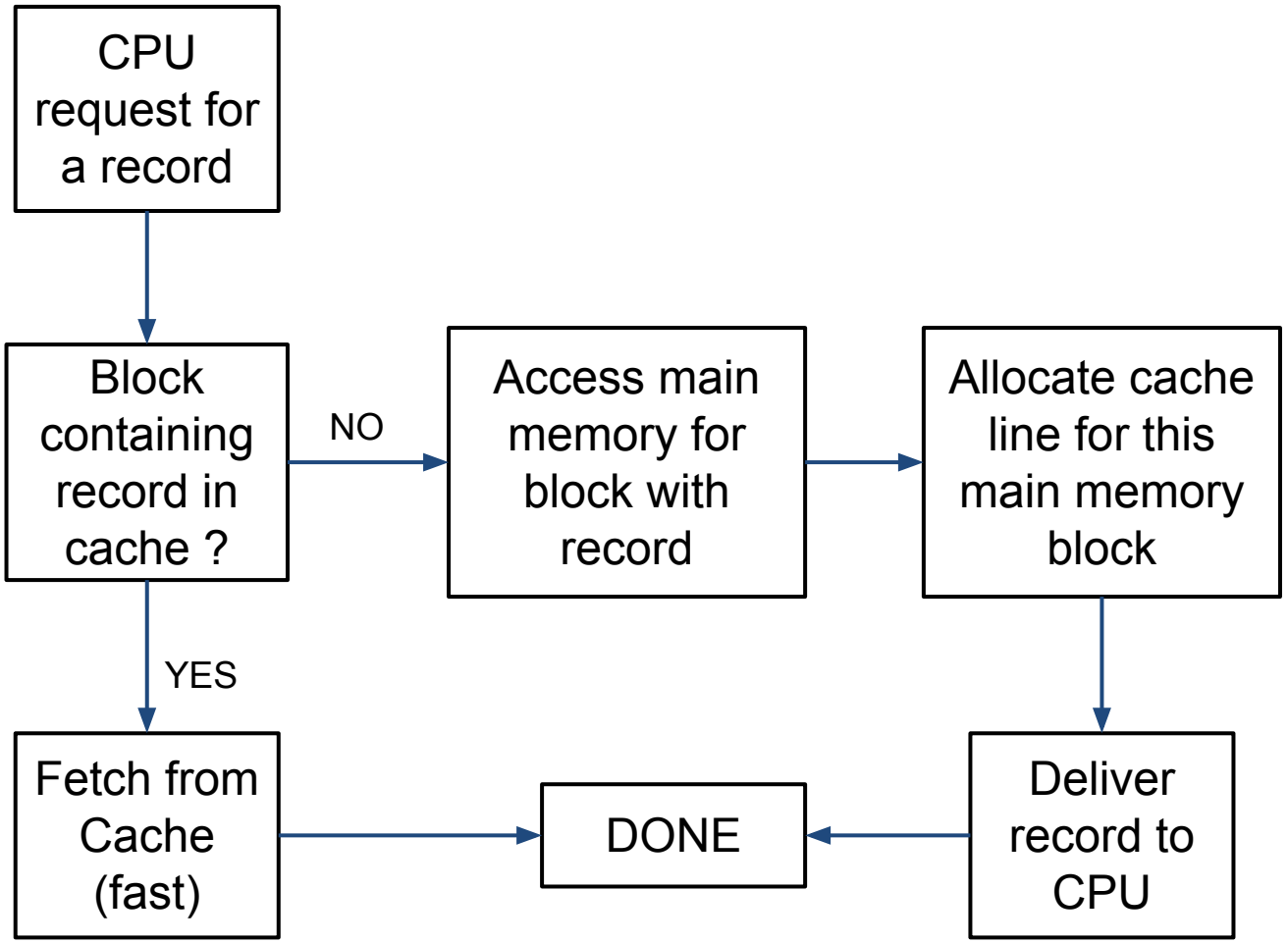
Caching - I

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



Caching - II



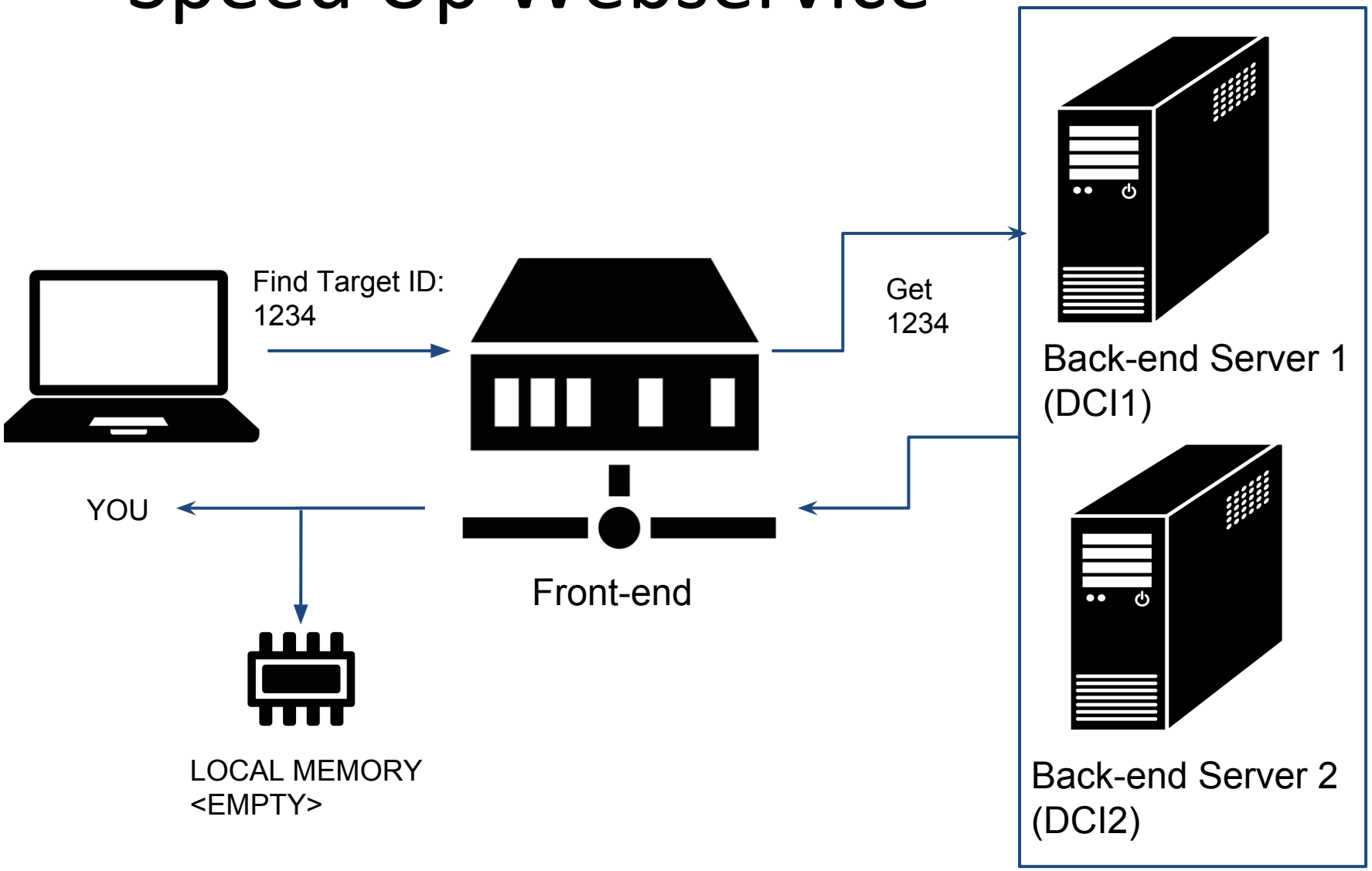

CPU request for a record
Block containing record in cache ?
NO
Access main memory for block with record
Allocate cache line for this main memory block
YES
Fetch from Cache (fast)
DONE
Deliver record to CPU

Caching - III

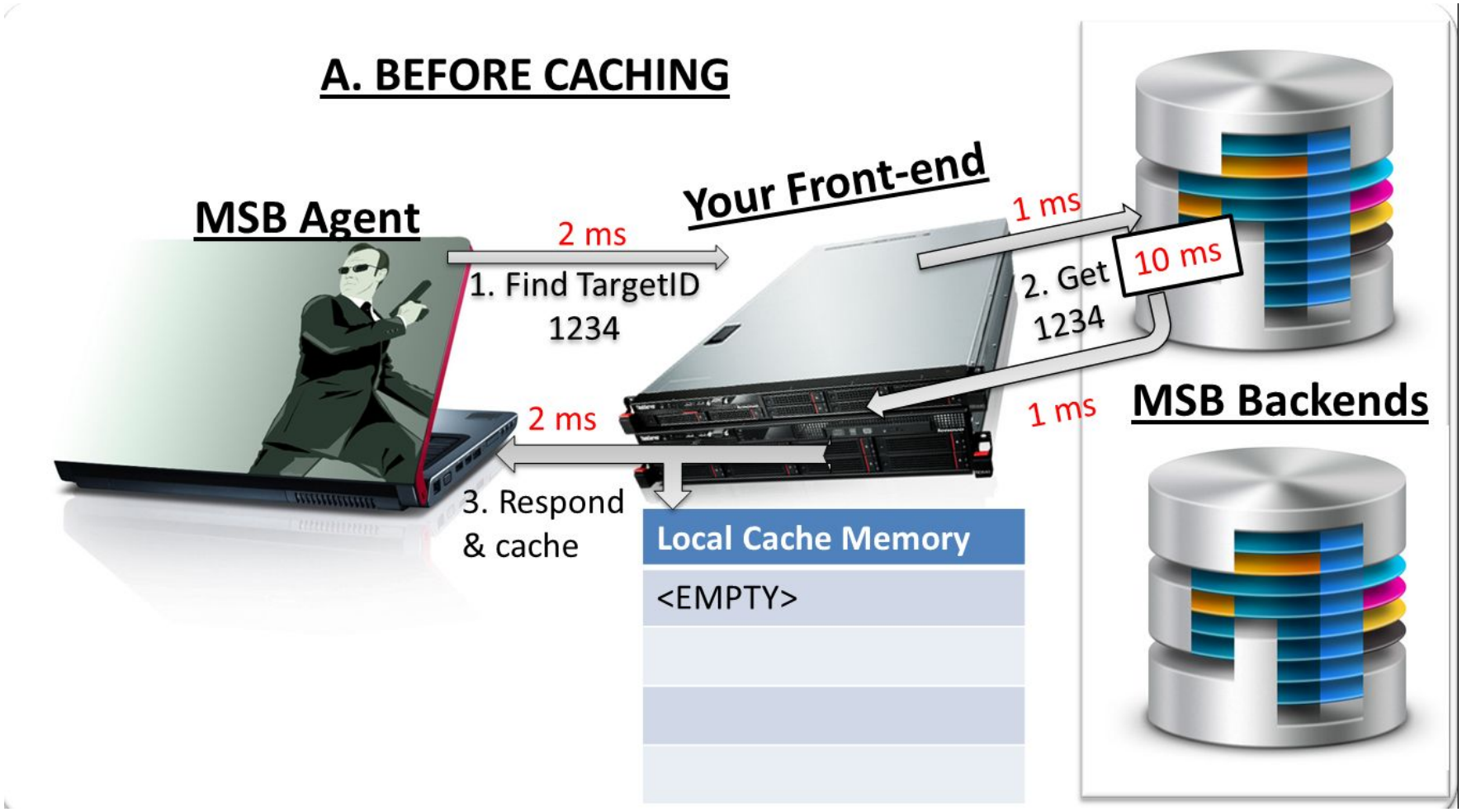
- Exploiting locality
  - Temporal locality
  - Spatial locality

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

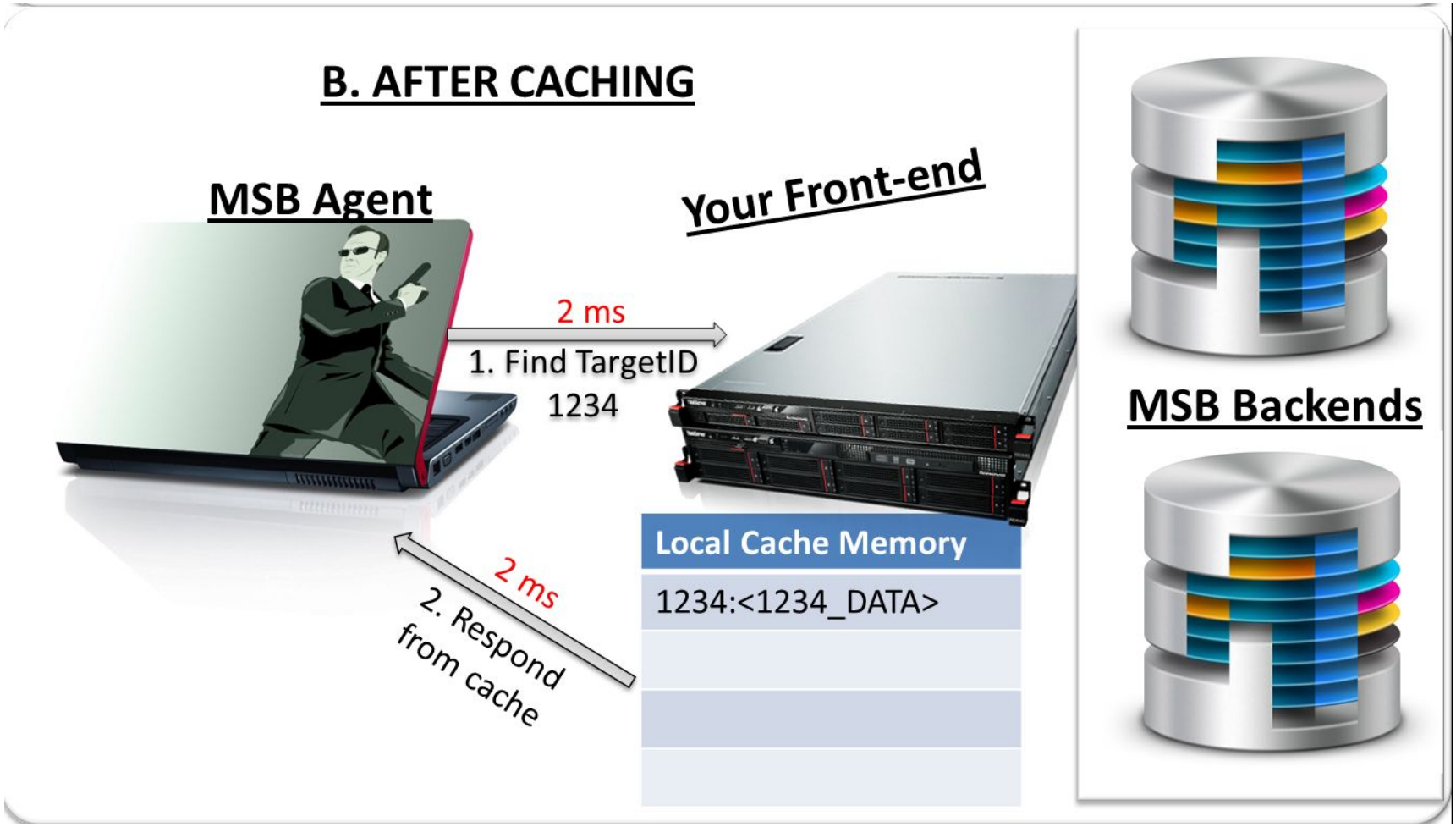
P2.3 - This Week
Speed Up Webservice



Web Service: Before Caching



Web Service: After Caching



P2.3 - what you have to do

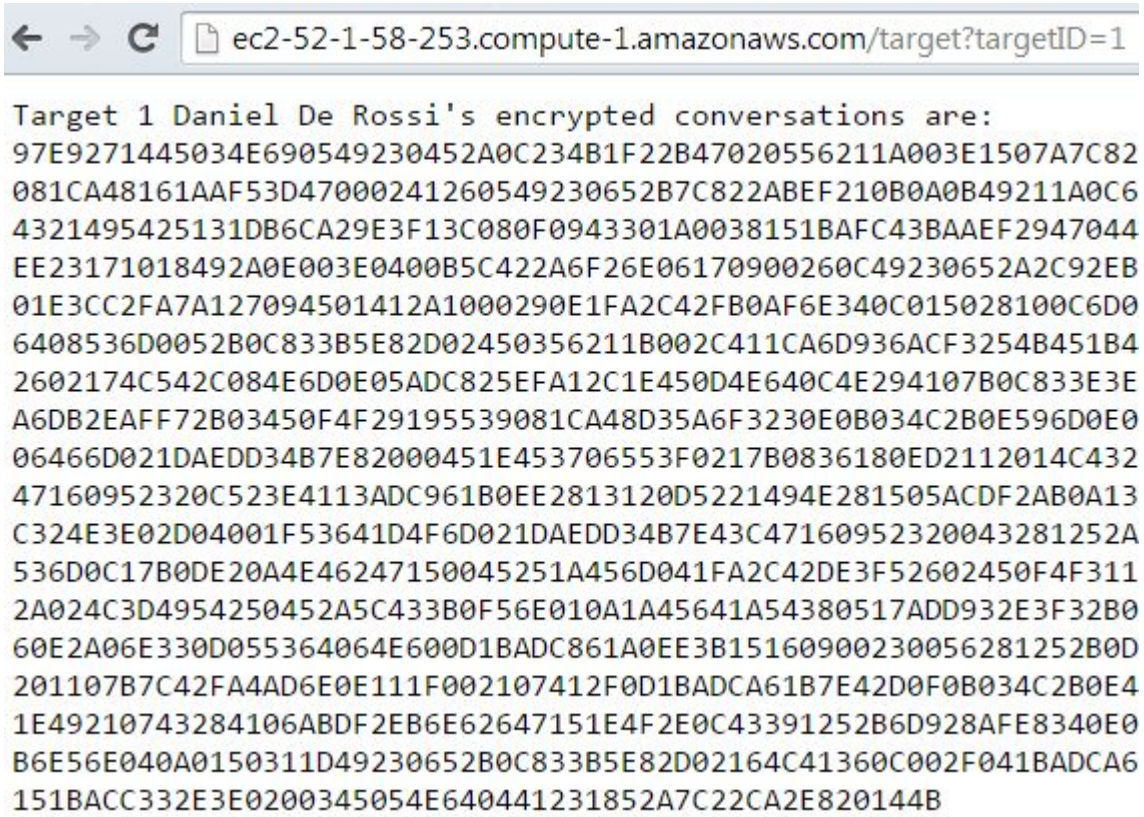
- Serve more web requests in the same time period using a cache to save on the number of hops
  - Size of cache on backend and frontend is predetermined
  - Learn the characteristics of the traces
  - Skeleton code provided gives low RPS
    - Trace 1: Skeleton code RPS ~ 25
      Target RPS ~ 145

Note: You are allowed to 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`

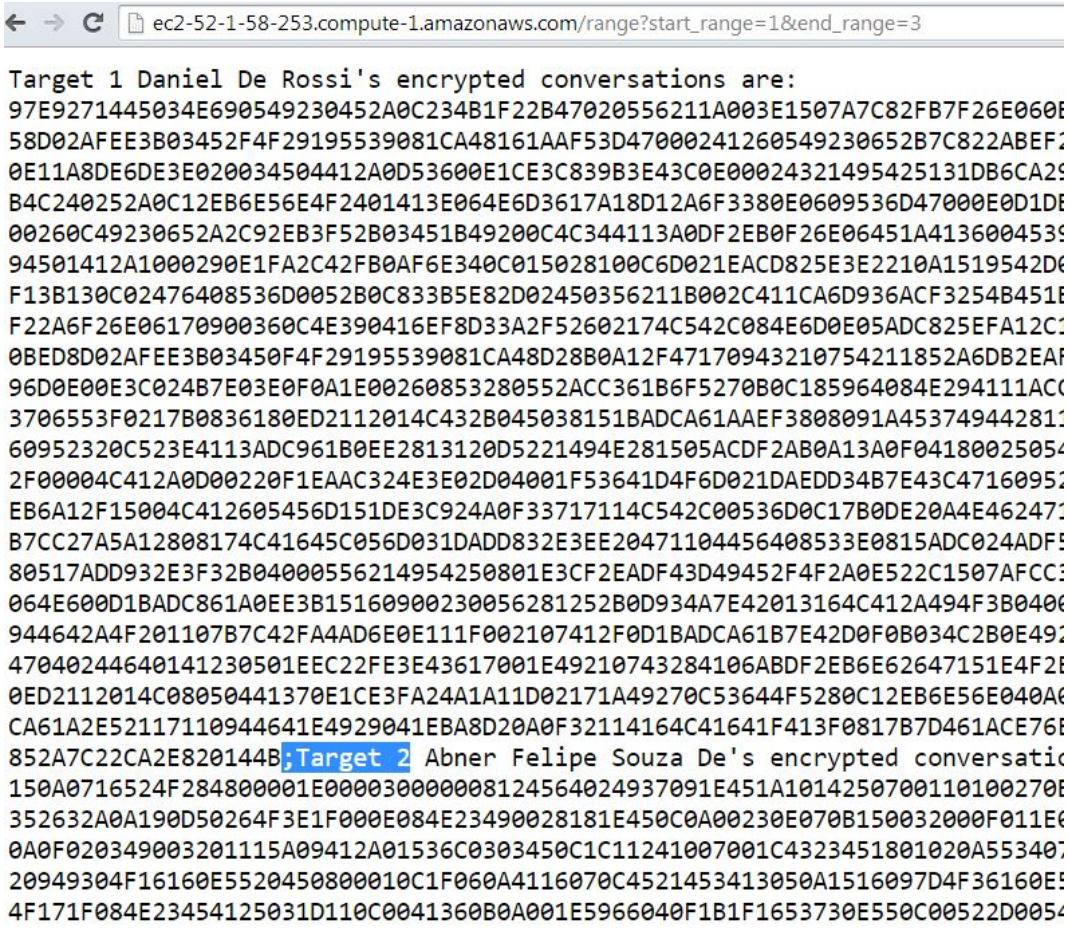
Response:



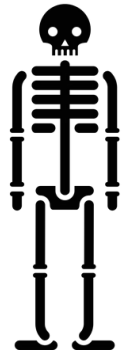
Querying multiple records from the DCI

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

Response:



Skeleton Code Provided



Skeleton code in java (**`MSB.java`**)

- You can get a 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

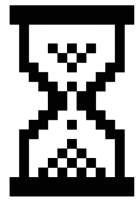
1. Brewer, Eric A. "Lessons from giant-scale services." IC, IEEE 5.4 (2001): 46-55.

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 front-end web-service:

Get record for targetID = 2000

Without the record cached:	0.135s
With the record cached at back-end:	0.030s
With the record cached at front-end:	0.011s

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 the project description more than once.
- Think about the 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

Project Grading Penalties

Besides the penalties mentioned in recitation and/or on Piazza, penalties accrue for the following:

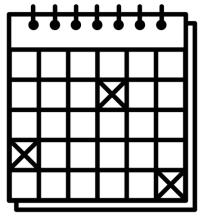
Violation	Penalty of the project grade
Spending more than $5 for this project phase on AWS	-10%
Spending more than $10 for this project phase on AWS	-100%
Failing to tag all your AWS resources in either parts (EC2 instances, etc) for this project	-10%
Submitting your AWS credentials in your code for grading	-100%
Submitting your Azure credentials in your code for grading	-100%
Submitting the Azure part with AWS instances or the AWS part with Azure VMs.	-100%
Attempting to hack/tamper the autograder in any way	-100%
Using virtual machines other than Standard_D1(DC and LG) and Standard_A0(FE) in the Azure part	-100%
Caching more than 1000 records in the front end.	-100%

Upcoming Deadlines

- Project 2.3: Advanced Scaling Concepts - Caching
  Due: 02/21/16 11:59PM Pittsburgh

- Quiz 5: Unit 3 - Virtualizing IO, Memory; Cases
  Due: 02/19/16 11:59PM Pittsburgh

15619 Project Time Table



Phase (and query due)	Start	Deadline	Code and Report Due
Phase 1 Part 1 • Q1, Q2	Thursday 02/25/2016 00:00:01 EST	Wednesday 03/16/2016 23:59:59 E**D**T	Thursday 03/17/2016 23:59:59 E**D**T
Phase 2 • Q1, Q2, Q3	Thursday 03/17/2016 00:00:01 E**D**T	Wednesday 03/30/2016 15:59:59 E**D**T	
Phase 2 Live Test (Hbase/MySQL) • Q1, Q2, Q3	Wednesday 03/30/2016 18:00:01 E**D**T	Wednesday 03/30/2016 23:59:59 E**D**T	Thursday 03/31/2016 23:59:59 E**D**T
Phase 3 • Q1, Q2, Q3, Q4	Thursday 03/31/2016 00:00:01 E**D**T	Wednesday 04/13/2016 15:59:59 E**D**T	
Phase 3 Live Test • Q1, Q2, Q3, Q4	Wednesday 04/13/2016 18:00:01 E**D**T	Wednesday 04/13/2016 23:59:59 E**D**T	Thursday 04/13/2016 23:59:59 E**D**T

Note:

- There will also be a report due at the end of each phase, where you are expected to discuss optimizations
- **WARNING: Check your AWS instance limits on the new account (should be > 10 instances)**

Demo