CIS 5050
Software Systems

Linh Thi Xuan Phan

Department of Computer and Information Science
University of Pennsylvania

Lecture 12: Bigtable + Project Overview

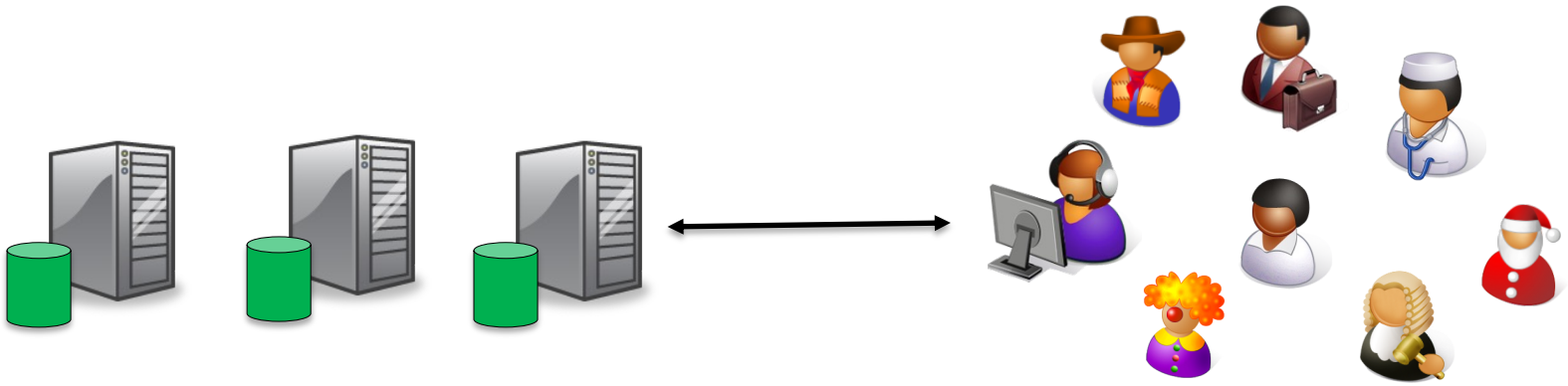
March 21, 2023

Plan for today

- Distributed storage systems 

- Bigtable
- A few words about the project

Why distributed storage systems?



- Goal: Make a storage system available to remote users
  - Example: Maintain shared data for a group
  - There could be many users
  - ... or the system itself could be spread across many nodes

- Key goal: Transparency
  - Users should not have to know (or care) where the data is located

How hard can this be?

- This sounds like a relatively simple problem
- And yet there are many many different solutions!
  - Examples: NFS, Coda, Andrew, GFS, Colossus, xFS, Swift, Frangipani, Intermezzo, GPFS, RGFS, Sprite, SMB, Locus, ...

- Why so many?
  - Lots of different design decisions!
  - Examples:
    - How much do we care about scalability? (Data center? Single node?)
    - What kind of consistency do we want?
    - What is the expected workload like?
    - What kind of network connectivity is available?
    - Does fault tolerance matter? If so, how strong do we want it to be?
    - What kind of security guarantees do we need?

Distributed storage: Examples

- Two 'classics'
  - Sun's Network File System (NFS)
    - Widely used! This is the file system you're using if you log into eniac.
  - CMU's Coda file system
    - A research system with interesting features

- Some more scalable examples
  - Google File System (GFS)
    - A file system for data centers; used, e.g., in conjunction with MapReduce/Hadoop
  - Cooperative File System (CFS)
    - A highly scalable file system that is based on a distributed hashtable
  - **Bigtable**
    - A flexible general-purpose storage system for structured data

Plan for today

- Distributed storage systems
- Bigtable 

- A few words about the project

Why Bigtable?

- An interesting example of scalable data storage systems with replication
  - Petabytes of data across thousands of machines
- Used extensively in many projects at Google
  - Web indexing, Google Earth, Google Finance, Google Analytics, …
- Designed for flexibility and performance
  - Varying data sizes (URLs, webpages, satellite images)
  - Often smaller pieces (e.g., compared to GFS); random accesses
  - Structured data (e.g., organized by user/by site/...)
  - Diverse latency requirements (bulk processing, real-time demands)
- Inspiration for the storage system in our project!

Bigtable

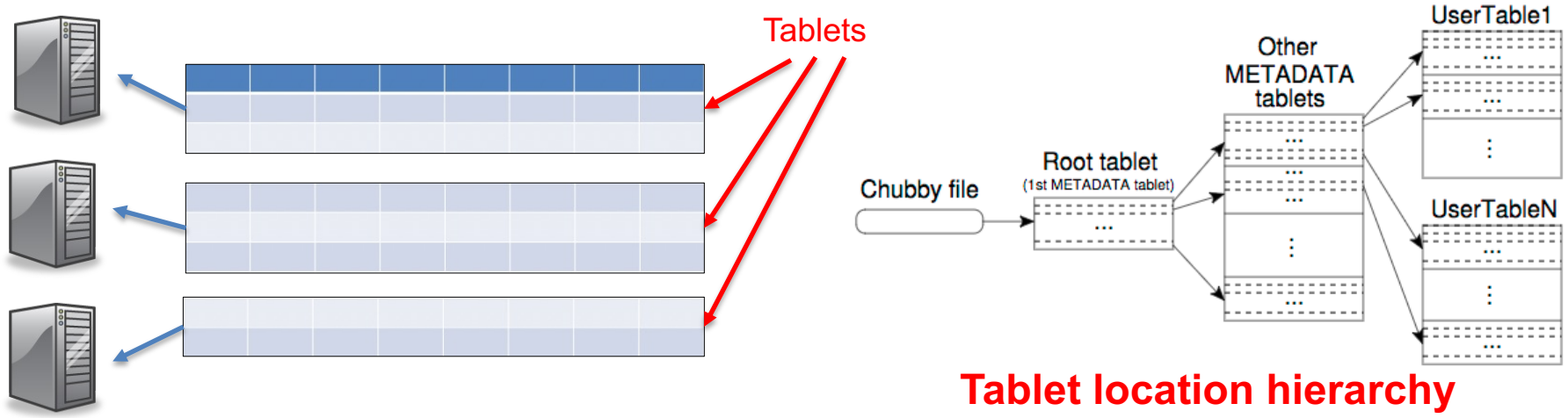
- Bigtable: A storage system for structured data
  - Designed by Google; widely used by over sixty different projects and products
- Common abstraction: Key-value pairs
  - Very general: tuple (k,v); k and v can typically be arbitrary bitstrings
    - Example: Key could be an email ID, and value the text of that email
  - Widely used: S3, memcached, MapReduce, DHTs, ...
  - Common interface of key-value stores: **PUT(k,v)**, **GET(k)** → **v**
- This is the abstraction we will use for our project

Bigtable

Row key	A	B	C	M	Q
...					
bar	1	2	3		
foo	7	9		8	12
...					

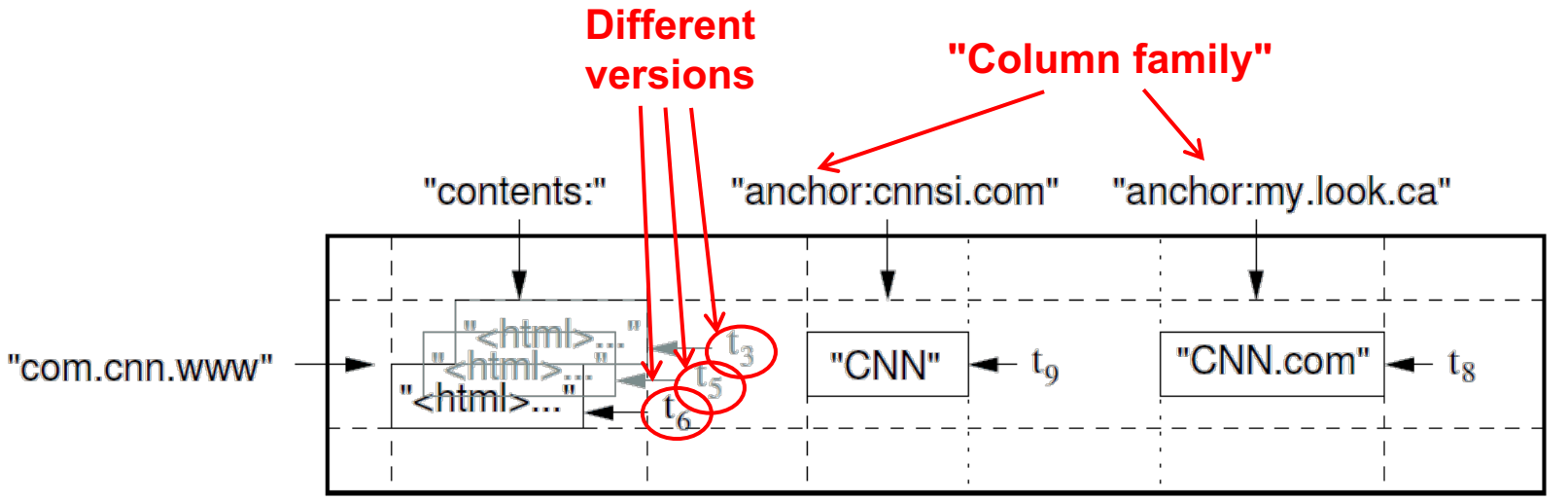
- Provides the abstraction of a giant table
  - Special column ("row key") contains a unique value for each row
    - Each row key is an arbitrary string (up to 64KB, but typical usage 10-100 bytes)
  - Each row is basically a little key-value store of its own – the key identifies a "column" in the row identified by the row key
  - No explicit schema: Different rows can have different columns
    - For instance, row ”bar" can contain (A,1), (B,2), (C,3), and row ”foo" can contain (A,7), (B,9), (M,8), (Q,12)
  - Each read (write) of data under a row key is atomic. (Why?)

Tablets in Bigtable



- The table is partitioned into row ranges ("tablets")
  - Tablets are bite-size pieces of data (~100-200MB by default)
- Tablets are distributed across many tablet servers
  - Each tablet server has many different tablets at any given time
  - Tablet state is stored in GFS for persistence
- Bigtable maintains an index by row key
  - Can be used to find rows by row key

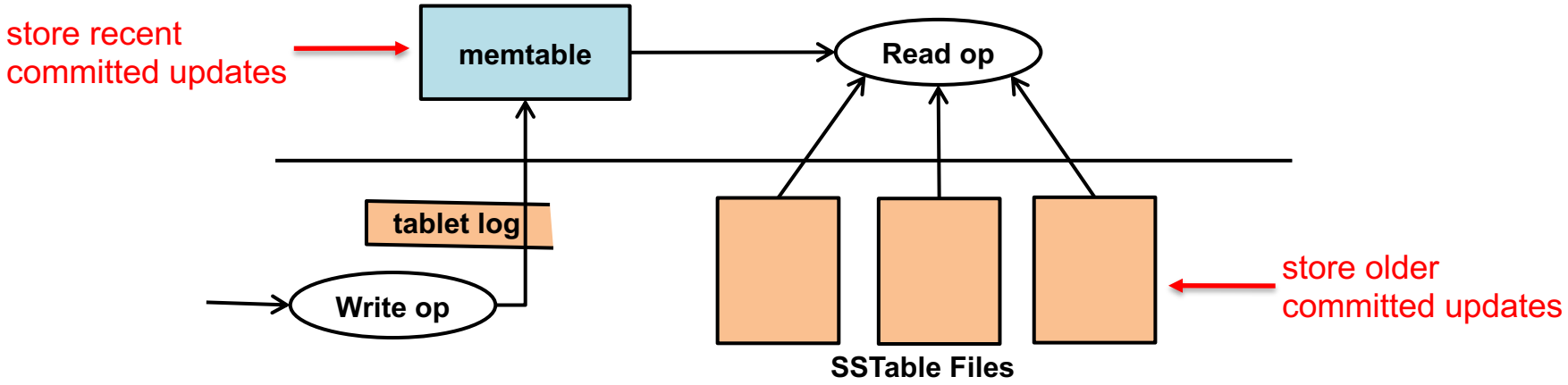
Versioning and column families



Source: Bigtable paper (OSDI2006)

- Data in Bigtable is versioned
  - Each cell can 'remember' values it has held previously (by timestamp)
    - Example: Row U, column "contents:" contains crawled web page at URL U
    - Remembering earlier crawls is useful, e.g., to see how a page changes over time
- Columns are grouped into column families
  - Example: "anchor:XYZ" columns could contain references from other pages (URL XYZ) to the page in the current row
  - Basic unit of access control

Bigtable implementation



- A single-master system, similar to GFS (later lecture)
  - The master assigns tablets to tablet servers, load balancing, etc.
- State is kept in a memtable and various SSTables
  - Writes are appended to a commit log and then kept in the memtable
    - After a crash, server can restore the state of its memtable by replaying the log
    - The memtable and SSTables are lexicographical sorted data structures
  - When the memtable fills up, it is 'frozen' (made read-only) and written to GFS as a SSTable ("minor compaction")
  - Reads are handled from the memtable and the SSTables
  - A background process occasionally merges SSTables and gets rid of old/deleted data ("major compaction")

Some services that use Bigtable

Project name	Table size (TB)	Compression ratio	# Cells (billions)	# Column Families	# Locality Groups	% in memory	Latency-sensitive?
Crawl	800	11%	1000	16	8	0%	No
Crawl	50	33%	200	2	2	0%	No
Google Analytics	20	29%	10	1	1	0%	Yes
Google Analytics	200	14%	80	1	1	0%	Yes
Google Base	2	31%	10	29	3	15%	Yes
Google Earth	0.5	64%	8	7	2	33%	Yes
Google Earth	70	–	9	8	3	0%	No
Orkut	9	–	0.9	8	5	1%	Yes
Personalized Search	4	47%	6	93	11	5%	Yes

Source: Bigtable paper (OSDI 2006)

- In 2006, there were 388 non-test Bigtable clusters at Google
  - Combined total: 24,500 tablet servers
- Example: Google Analytics
  - Raw click table (~200TB): 1 row for each end-user session
  - Summary table (~20TB): Predefined summaries per website

How does this relate to the class?

- Bigtable uses many of the technologies we will look at later in this course:
  - Lock service is made fault-tolerant with Paxos
  - Clients can run per-row transactions
  - Data is persisted in a scalable file system, GFS
  - Bigtable can be used as source or target for MapReduce jobs
- More details are in the paper:
  - F. Chang, J. Dean, S. Ghemawat, W. Hsieh, D. Wallach, M. Burrows, T. Chandra, A. Fikes, R. Gruber: "Bigtable: A Distributed Storage System for Structured Data", OSDI 2006
  - http://research.google.com/archive/bigtable-osdi06.pdf
- Inspiration for Apache HBase (open-source)

How does this relate to the project?

- The "key-value store" is modeled after Bigtable
- However, you don't need to implement all features!
  - You can leave out versioning, access control, and the index
  - You can store data directly on the storage node (instead of in GFS)
  - You don't have to use the SSTable mechanism (unless you want to)

- A basic implementation could work like this:
  - Single master that knows which tablets are on which node
  - Several storage nodes that keep their tablets in memory
  - When a PUT happens, the node appends the change to a log file on disk, so it is persistent if the node should crash
    - If a crash does happen, the node can 'replay' the changes in the log file
  - Once in a while, each node writes snapshots of its tablets to a file on disk and then empties its change log

Plan for today

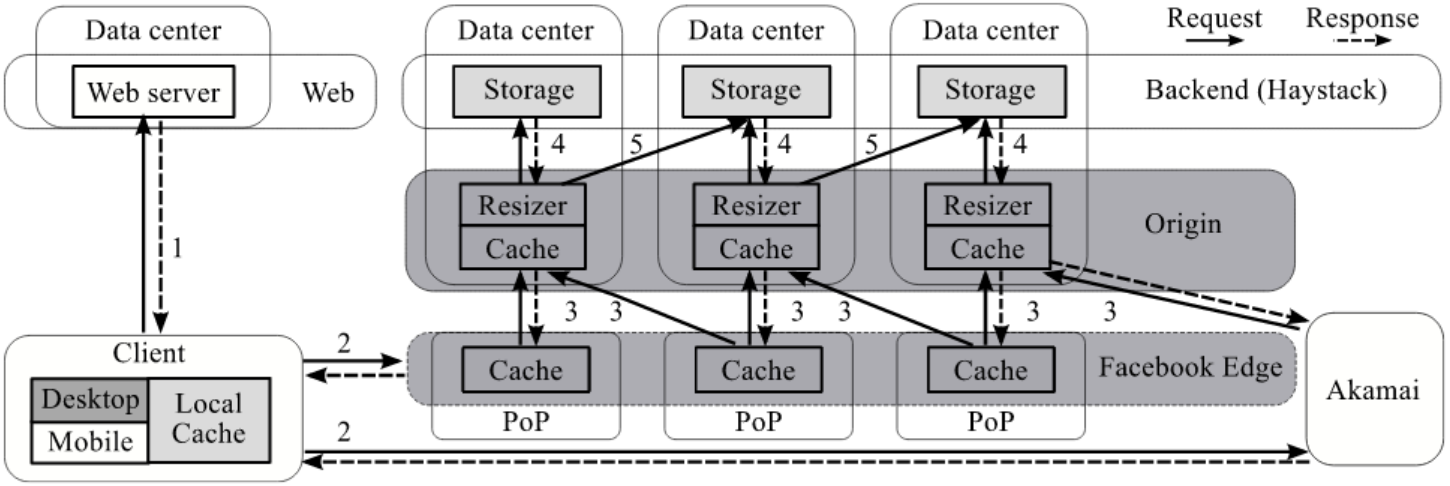
- Distributed storage systems
- Bigtable
- A few words about the project

Why a Final Project?

- So far, homeworks have been single components:
  - HW1: Sorting program
    - Opportunity to get experience with processes, threads, and IPC
    - Learn about concurrency, synchronization, coordination
  - HW2: SMTP and POP3
    - Get experience with client/server (very common paradigm!), socket programming
    - Learn to work with, and comply with, simple protocol specifications (RFCs)
    - Understand how simple real-world systems work
      - Why not IMAP? Because it is far more complicated than POP3!
  - HW3: Chat server
    - Get experience with distributed coordination, group communication
    - Work with different consistency models; implement standard ordering protocols

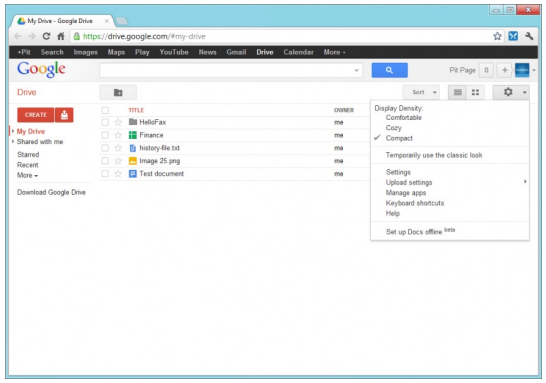
- But the class is about software systems!
  - Don't these have multiple interacting components?

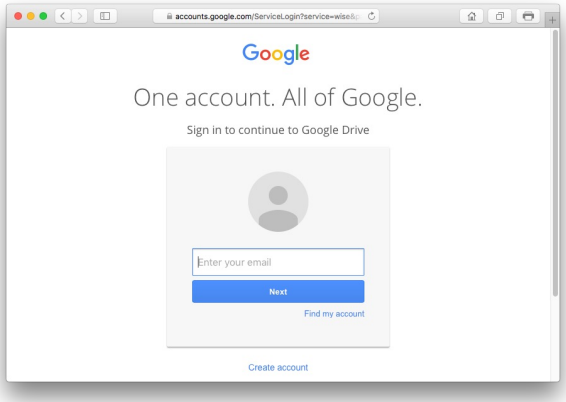
Real-world systems

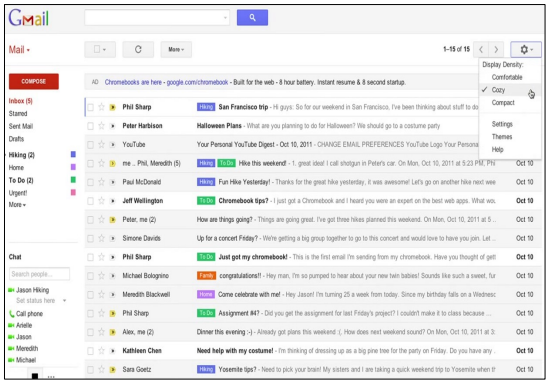


- Example: Facebook's photo serving stack [SOSP'13]
  - Widely distributed; several different layers
    - Usually a user-facing 'frontend' (via web protocols) and an internal 'backend' (e.g., for storage)
  - Multiple components with complex interactions
- This is too much work to build by yourself!
  - Hence, I've saved this for the final project, which we'll do in teams

Final project: PennCloud



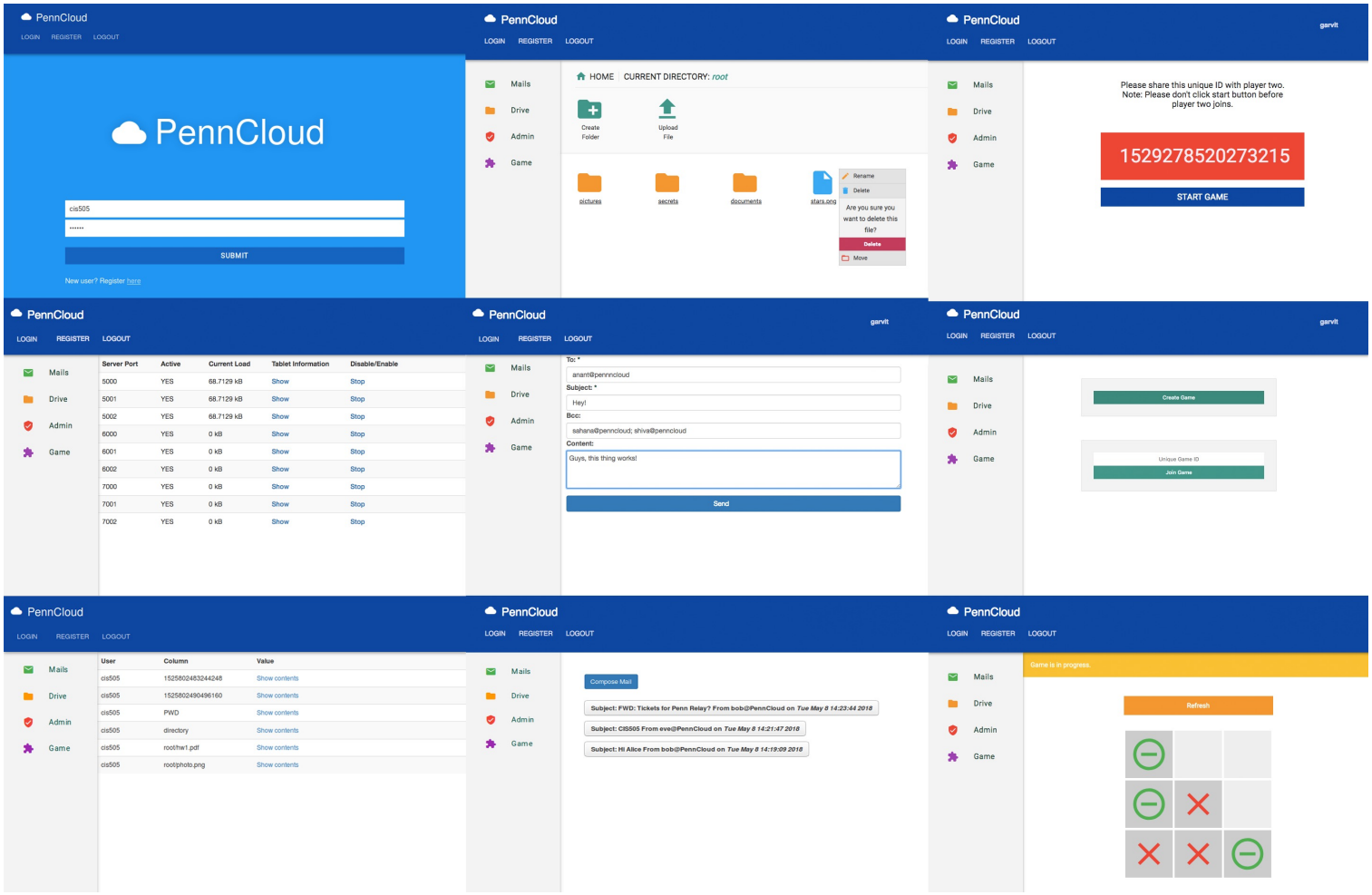




- Goal: Implement a small clone of 'Google Apps'
  - Just two apps: Webmail and storage
  - No need to replicate the fancy optics (just some of the key features)
  - Should include a web frontend and a key-value store backend
  - Should be done in teams of four
- Can apply many of the things we've discussed!
  - RPCs, fault tolerance, replication, client/server, synchronization, socket programming, distributed coordination, ...
  - Also: Nice addition to your portfolio! (e.g., for interviews)

Examples from previous years

Storage

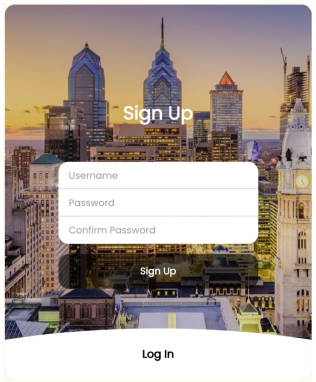


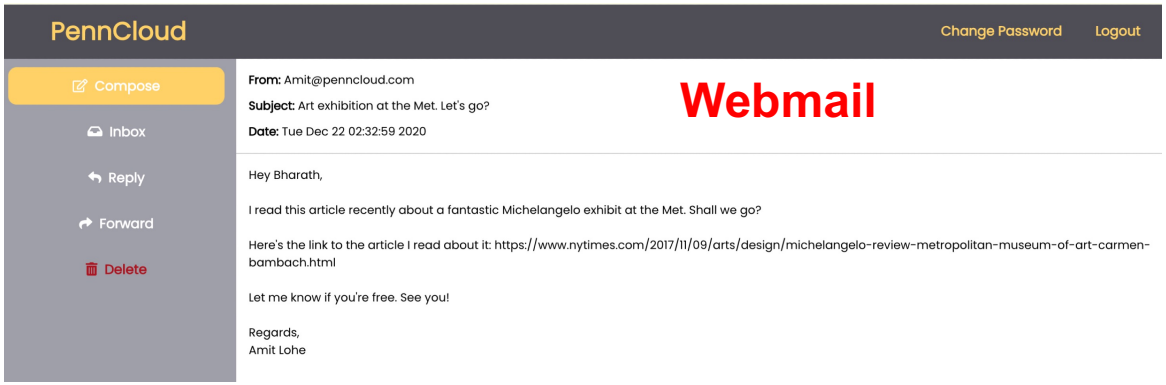
Admin

Game

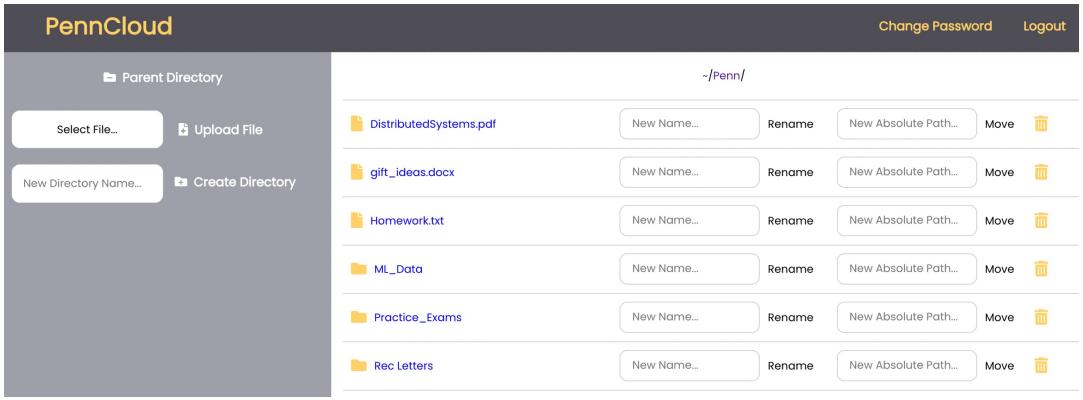
Webmail

Examples from previous years

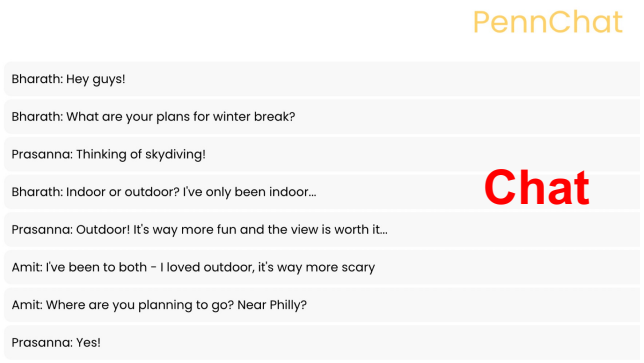




Webmail



Storage

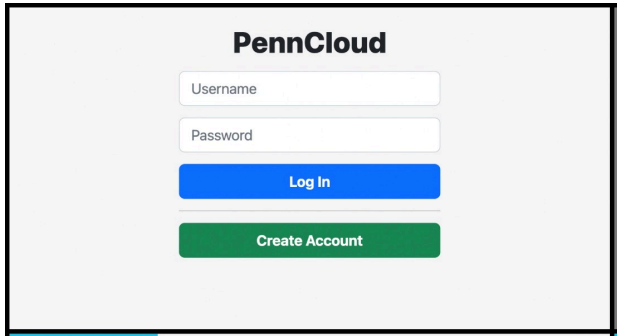


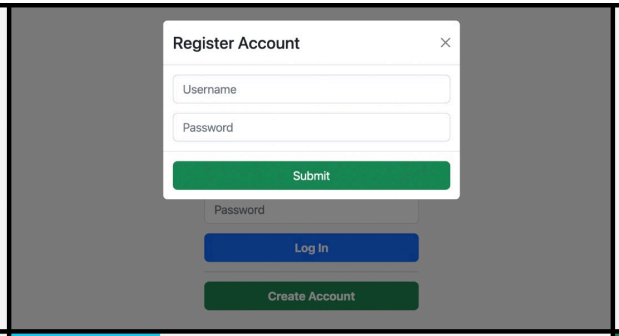
Chat

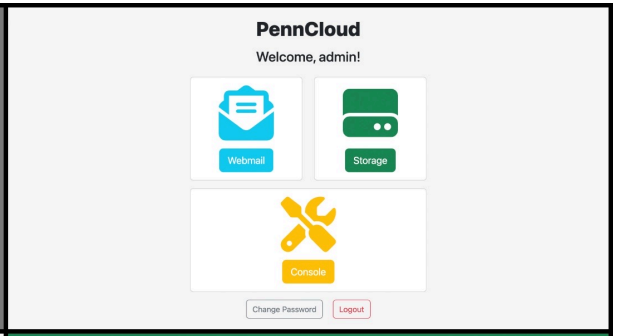
Discussion board

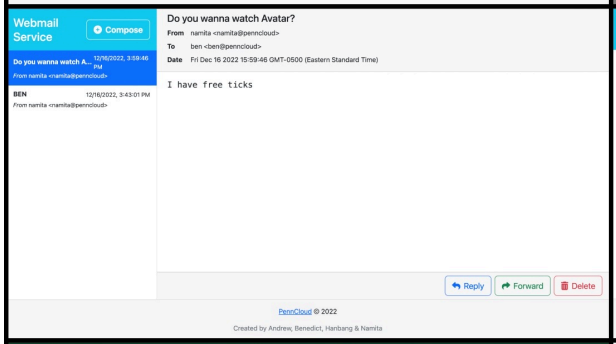
Examples from previous years

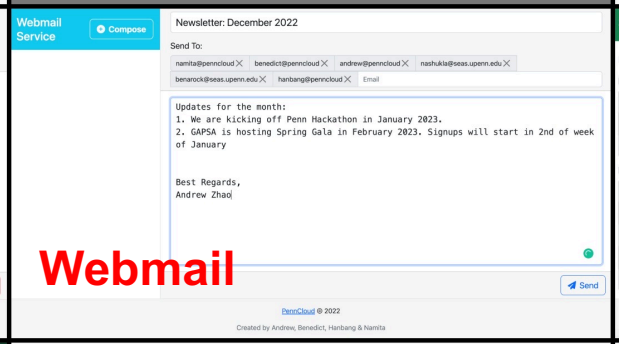
Log in



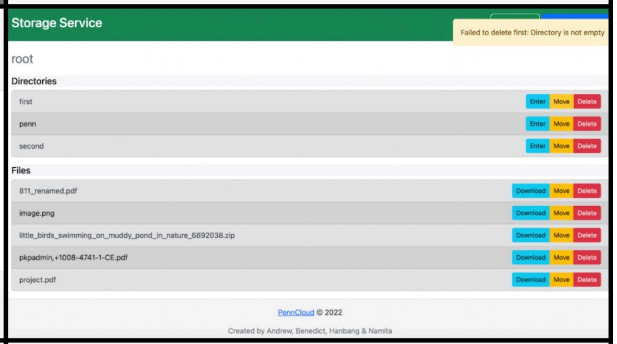


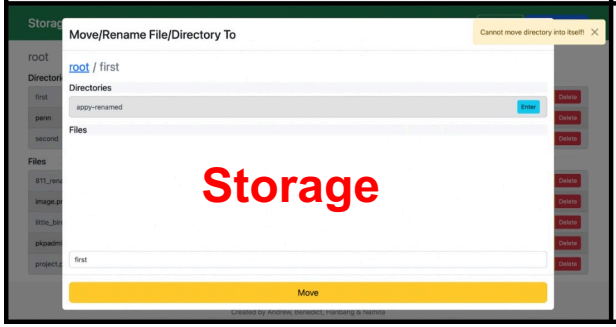




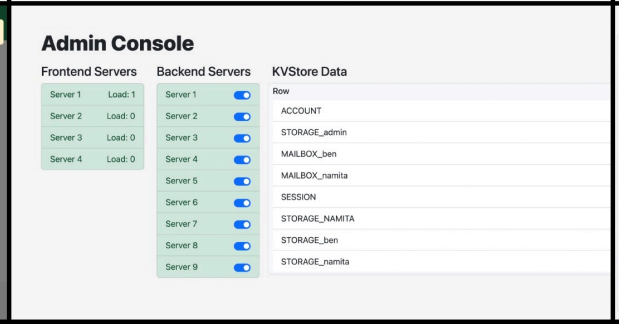


Webmail



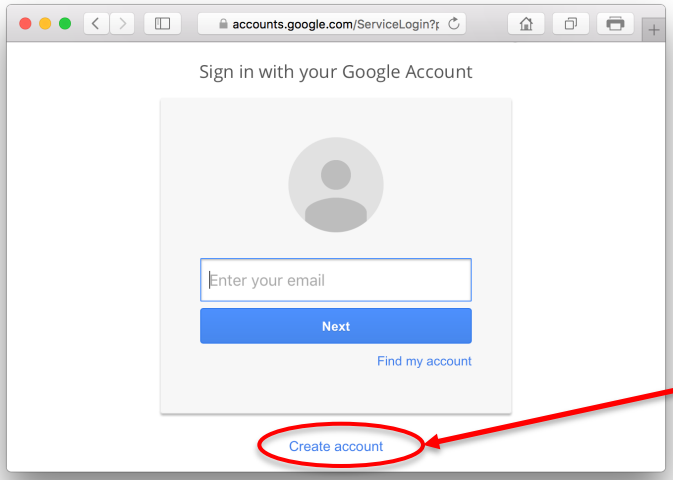


Storage



Admin

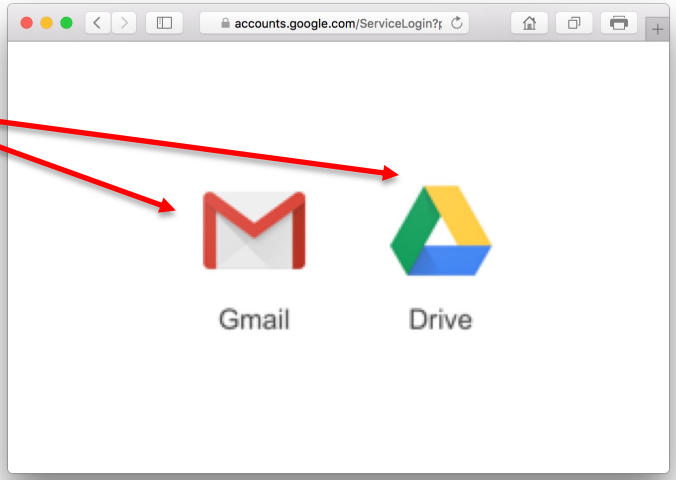
How users interact with the project



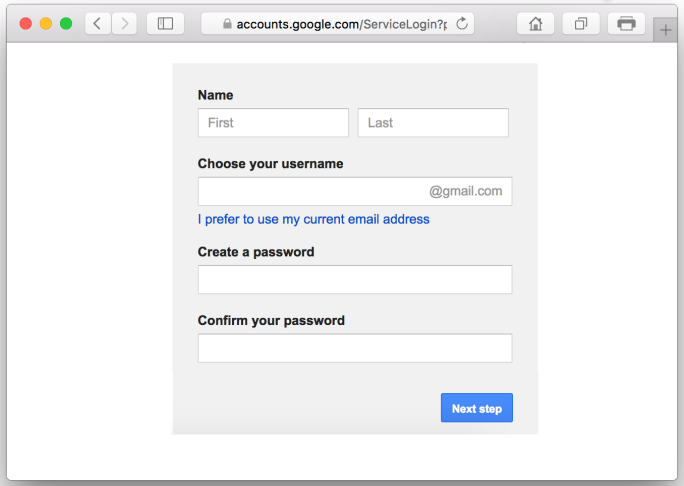
Sign up link

Login page: Enter username and password, or sign up for new account

Links to webmail & storage



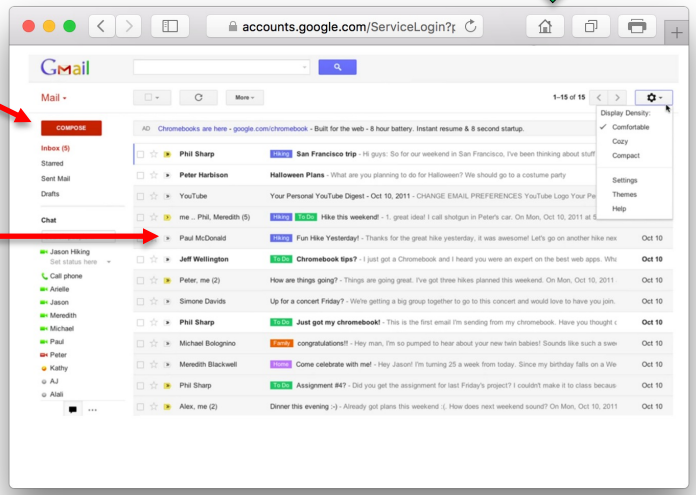
Menu page: Go to the webmail service or to the storage service



Signup page: Enter information for a new account

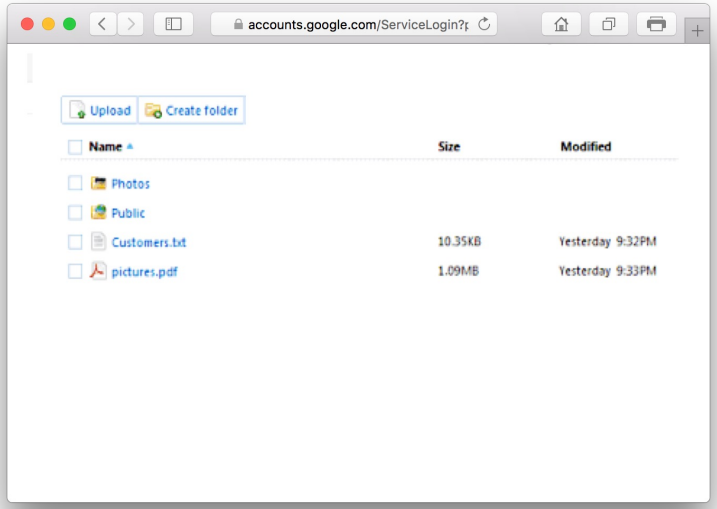
Compose button

List of emails

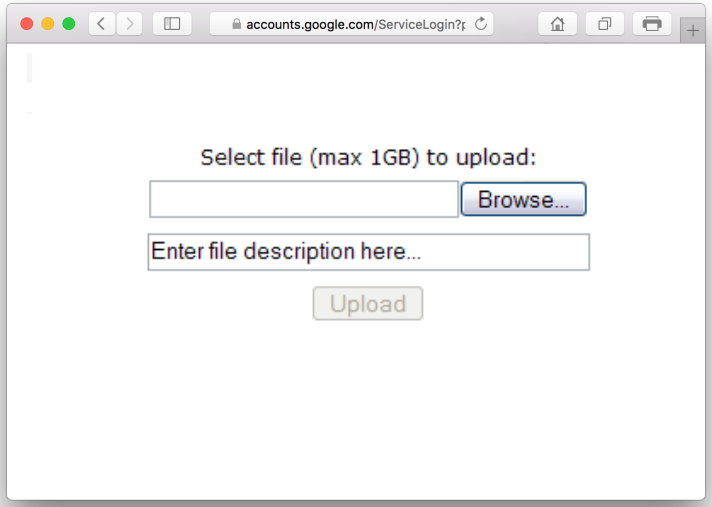


Webmail page: Shows list of emails in inbox & a "compose" button

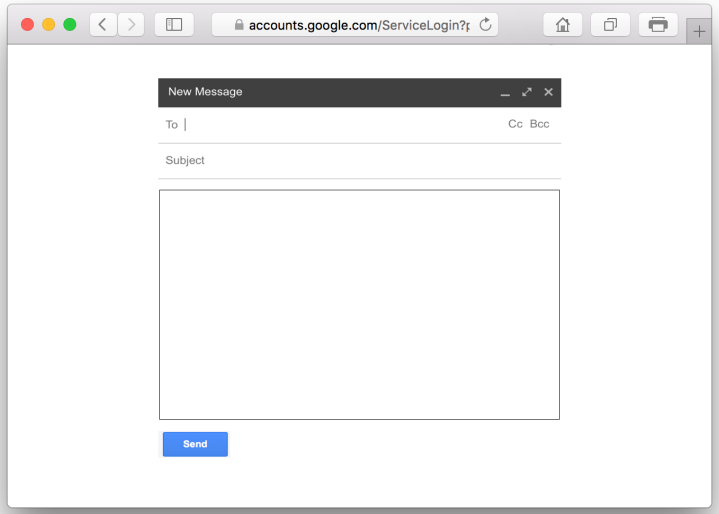
How users interact with the project



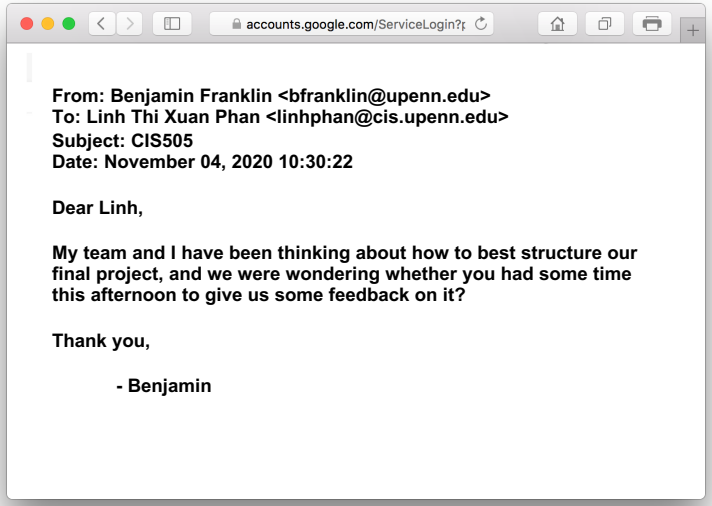
Directory page: Displays the files the user has currently stored in the system



Upload page: Lets the user upload a new file into their storage

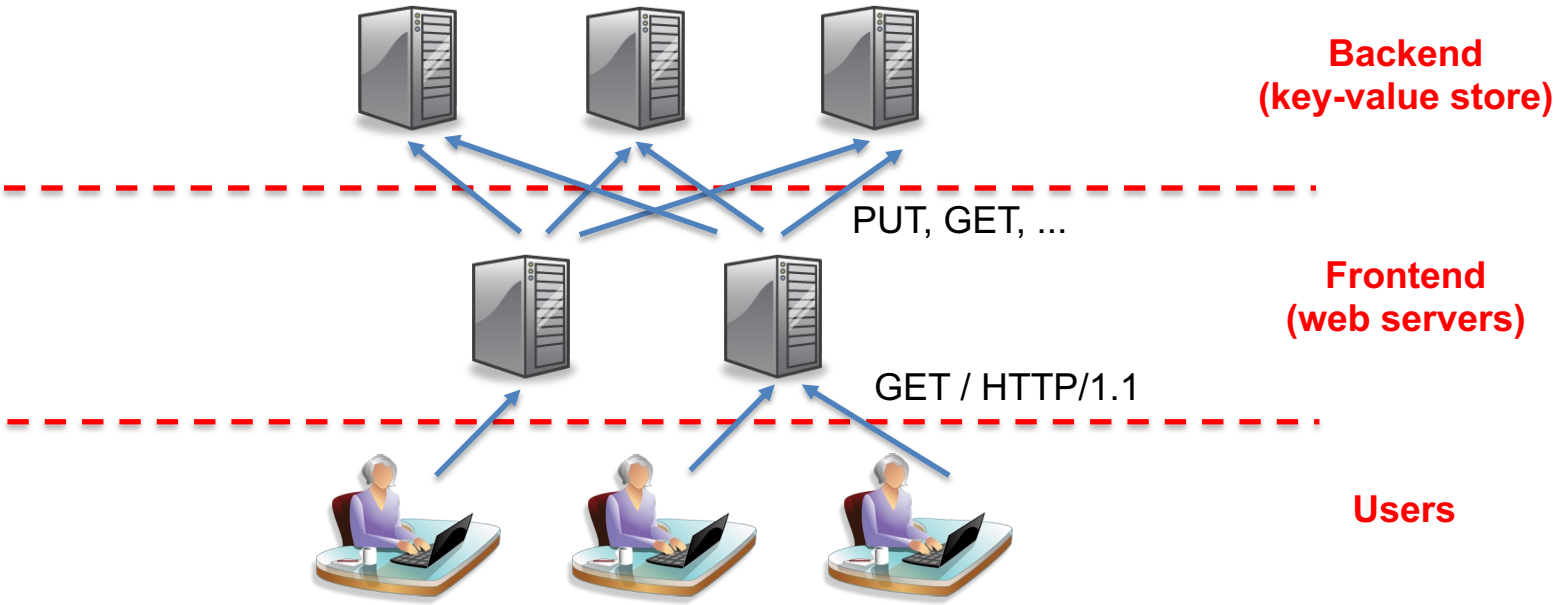


Compose page: Lets the user enter a new email to be sent



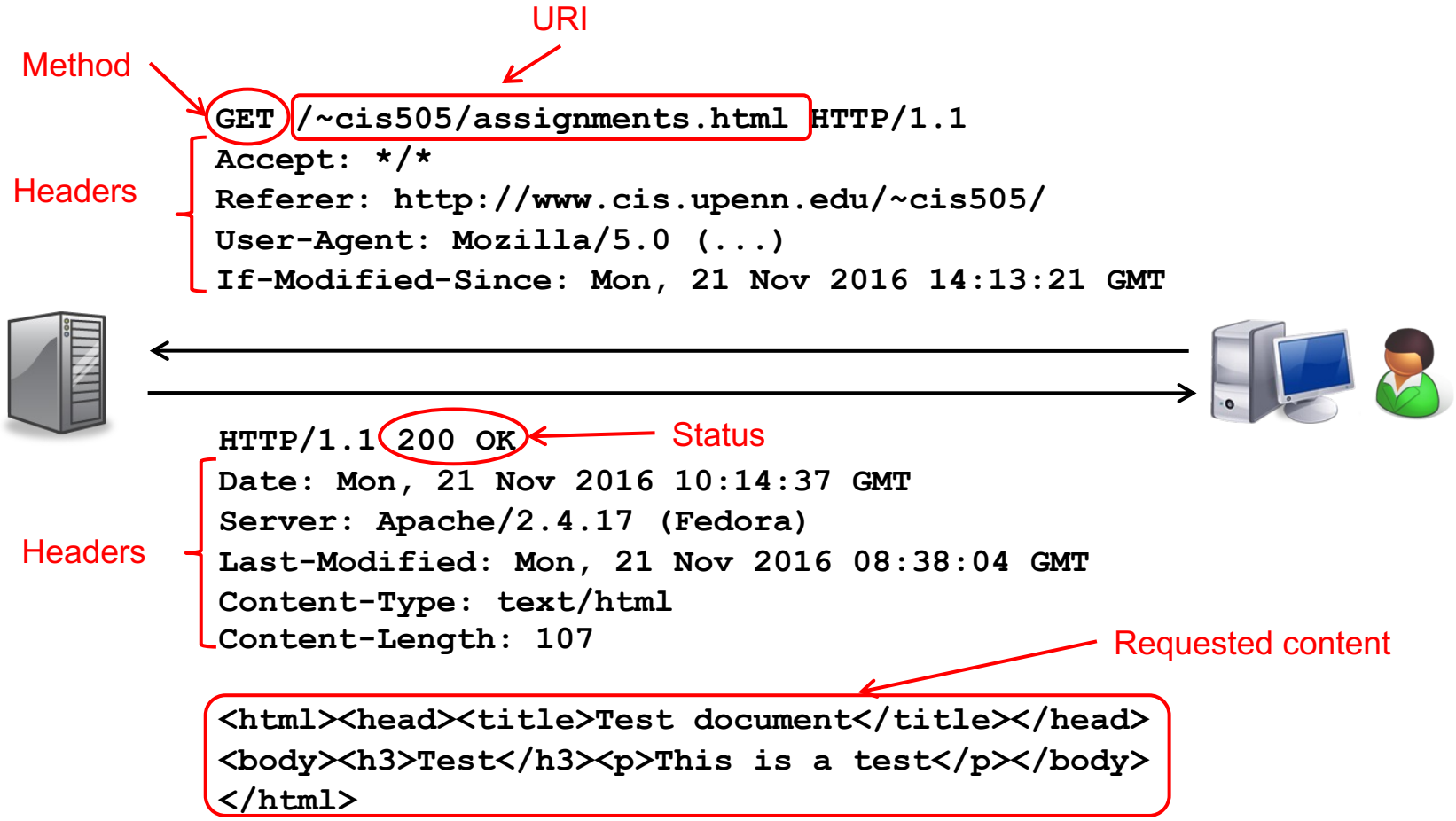
Email page: Shows an individual email

Major components



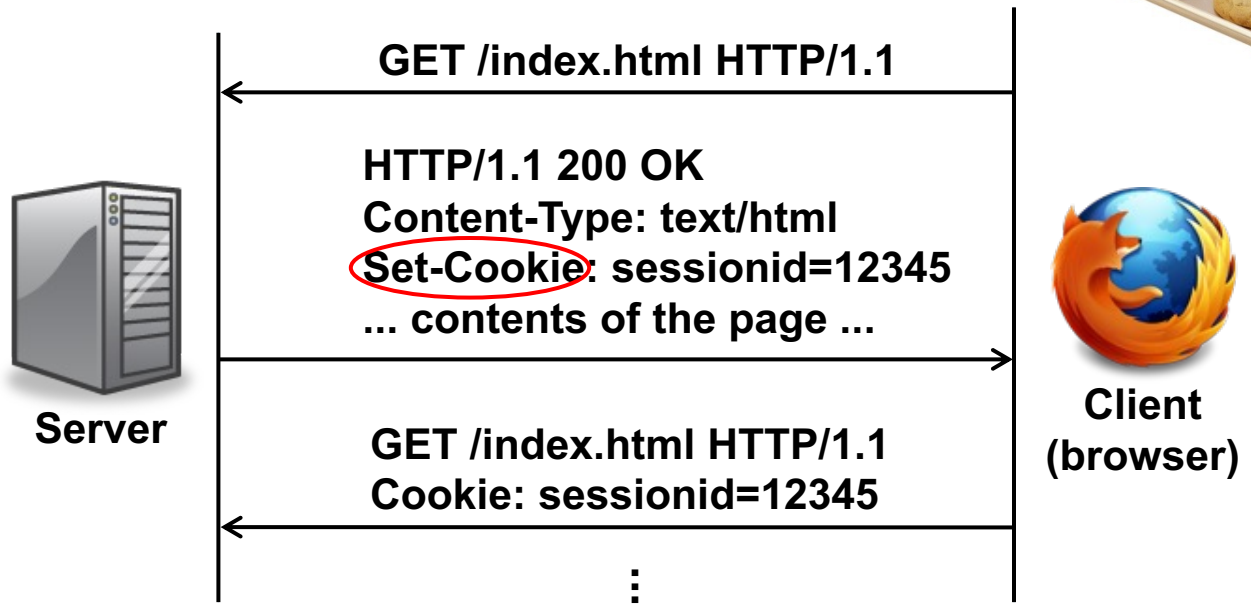
- Frontend tier: Several web server instances
- Backend tier: Several key-value store nodes
- User can connect to any front-end server
  - All the state is supposed to be stored in the key-value store!

HTTP in one slide



- Very similar to the servers you wrote for HW2
  - Text-based requests and responses; lots of headers with extra information, but simple implementations can omit most of them
  - For more details: https://www.jmarshall.com/easy/http/

HTTP cookies



- What is a cookie?
  - A set of key-value pairs that a web site can store in your browser (example: 'sessionid=12345')
  - Created with a Set-Cookie header in the HTTP response
  - Browser sends the cookie in all subsequent requests to the same web site until it expires

Example session



GET / HTTP/1.1

HTTP/1.1 200 OK
...HTML for login page...

POST /login HTTP/1.1
user=linh&pass=foo

Server for row 'linh'?

A

GET("linh","pwd")

"foo"

PUT("linh","sid", 123)

OK

HTTP/1.1 200 OK
Set-Cookie: sessionid=123
Set-Cookie: user=linh
...HTML for menu page...

GET /emails HTTP/1.1
Cookie: user=linh; sid=123

GET("linh","sid")

123

GET("linh","emails")

1, 2, 5, 9, 12, 15, ...

GET("linh","header-1")

...

Request for the login page; user has no cookies yet

Web server needs to lookup data in row "linh" in the storage service; contacts storage master to see which node is responsible for this row

Web server looks up the password

Password matches, so server sets a session ID

Server returns the menu page along with a cookie that contains the user name and the session ID

User requests the list of emails; cookies are included. Server checks whether session ID is valid

Web server requests the list of email IDs

Web server starts downloading email headers using email IDs

How fancy does it have to be?

- There is no need to replicate Gmail visuals etc.
  - Very basic HTML (unordered lists etc.) are sufficient
- However, the project is open-ended!
  - We will liberally award extra credit for extra features
  - Please get the basics working first and only then start on EC tasks!
- The project will be graded on
  - Functionality: Are all the required features there? Do they work?
  - Design: Is the system well-designed? (Bottlenecks? Fault tolerance? Recovery? Scalability? Etc.)
  - Usability: Good user interface, etc.

Resources

- Storage backend
  - Bigtable paper (research.google.com/archive/bigtable-osdi06.pdf)
- Web frontend
  - HTTP Made Really Easy (https://www.jmarshall.com/easy/http/)
  - RFC 2616 (https://www.ietf.org/rfc/rfc2616.txt) – basic HTTP
  - RFC 2965 (https://tools.ietf.org/html/rfc2965) – cookies
- HTML
  - http://www.w3schools.com/html/