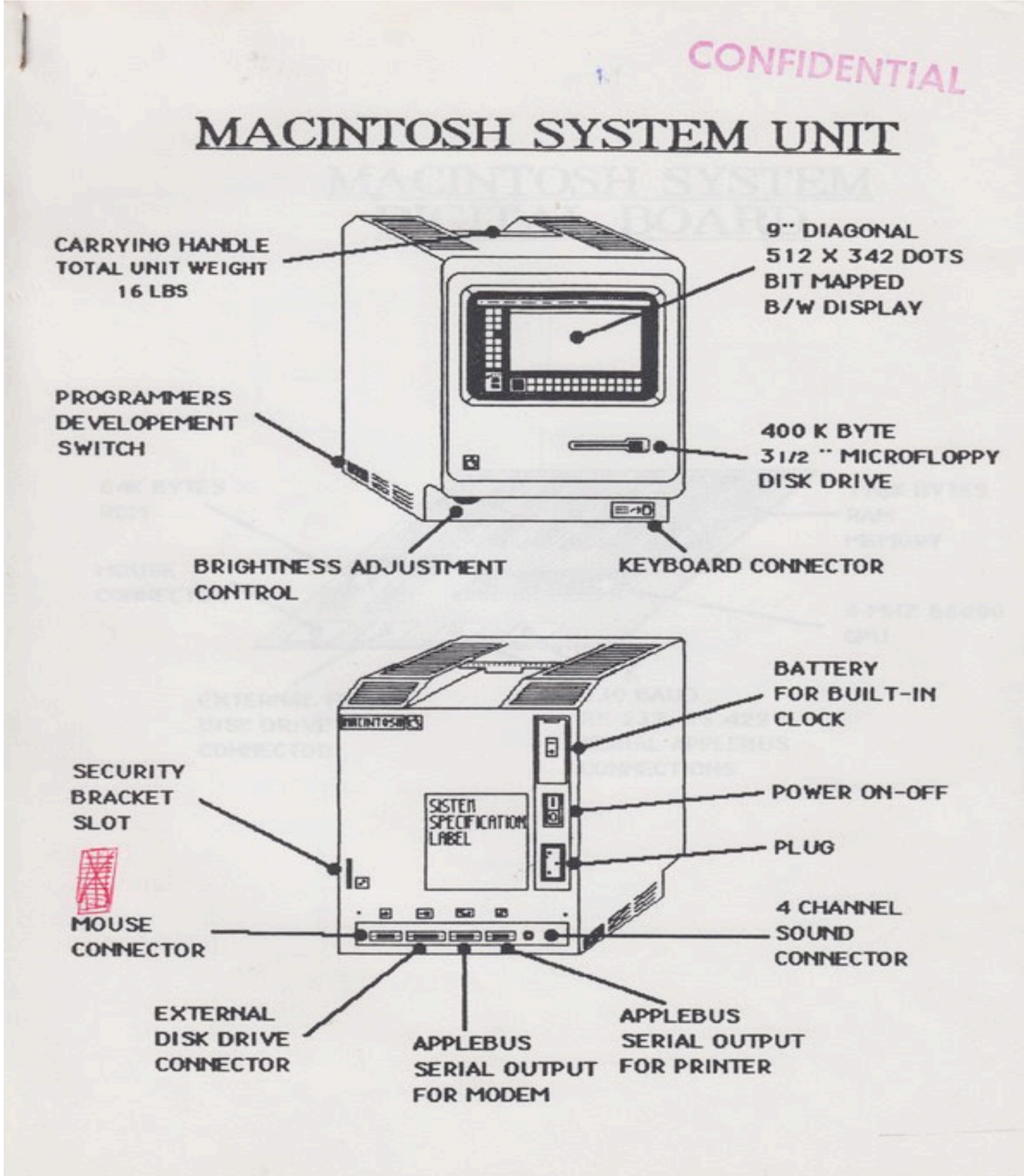

CONFIDENTIAL
MACINTOSH SYSTEM UNIT
CARRYING HANDLE
TOTAL UNIT WEIGHT
16 LBS
9" DIAGONAL
512 X 342 DOTS
BIT MAPPED
B/W DISPLAY
PROGRAMMERS
DEVELOPEMENT
SWITCH
400 K BYTE
3 1/2 " MICROFLOPPY
DISK DRIVE
BRIGHTNESS ADJUSTMENT
CONTROL
KEYBOARD CONNECTOR
BATTERY
FOR BUILT-IN
CLOCK
SECURITY
BRACKET
SLOT
SYSTEM
SPECIFICATION
LABEL
POWER ON-OFF
PLUG
MOUSE
CONNECTOR
4 CHANNEL
SOUND
CONNECTOR
EXTERNAL
DISK DRIVE
CONNECTOR
APPLEBUS
SERIAL OUTPUT
FOR MODEM
APPLEBUS
SERIAL OUTPUT
FOR PRINTER

CPEN 321

REST, Microservices,
Midterm 1 Recap

Recap

- Architectural principles
- Architectural patterns
  - Layered architecture
  - Client-server architecture
  - Pipe-and-filter architecture
  - Model-View-Controller (MVC)
  - Model-View-ViewModel
  - Message bus
  - **REST, Microservices**

What is REST?

- REST (Representational State Transfer) is a *design guideline* for communication in networked systems
  - not a protocol, not a specification

- Three main parts to remember:
  - Resource Identification: a URI, e.g., my.domain.ca/cars/bmw
  - Resource representation: any format, e.g., XML, a web page, comma-separated-values, printer-friendly-format, JSON,…)
    - Can flow to and from the server
  - Unified interface to retrieve, create, delete or update resources

Uniform Interface

- Similar to the CRUD (Create, Read, Update, Delete) databases operations

- REST *Uniform Interface Principle* uses 4 main HTTP methods
  - GET: Retrieve a representation of a resource.
  - POST*: Create a new resource.
  - PUT*: Update a resource (existing URI).
  - DELETE: Clear a resource, afterwards the URI is no longer valid

* Some claim you can use PUT to create resources or POST to update

Do Not Misuse HTTP Interface Conventions

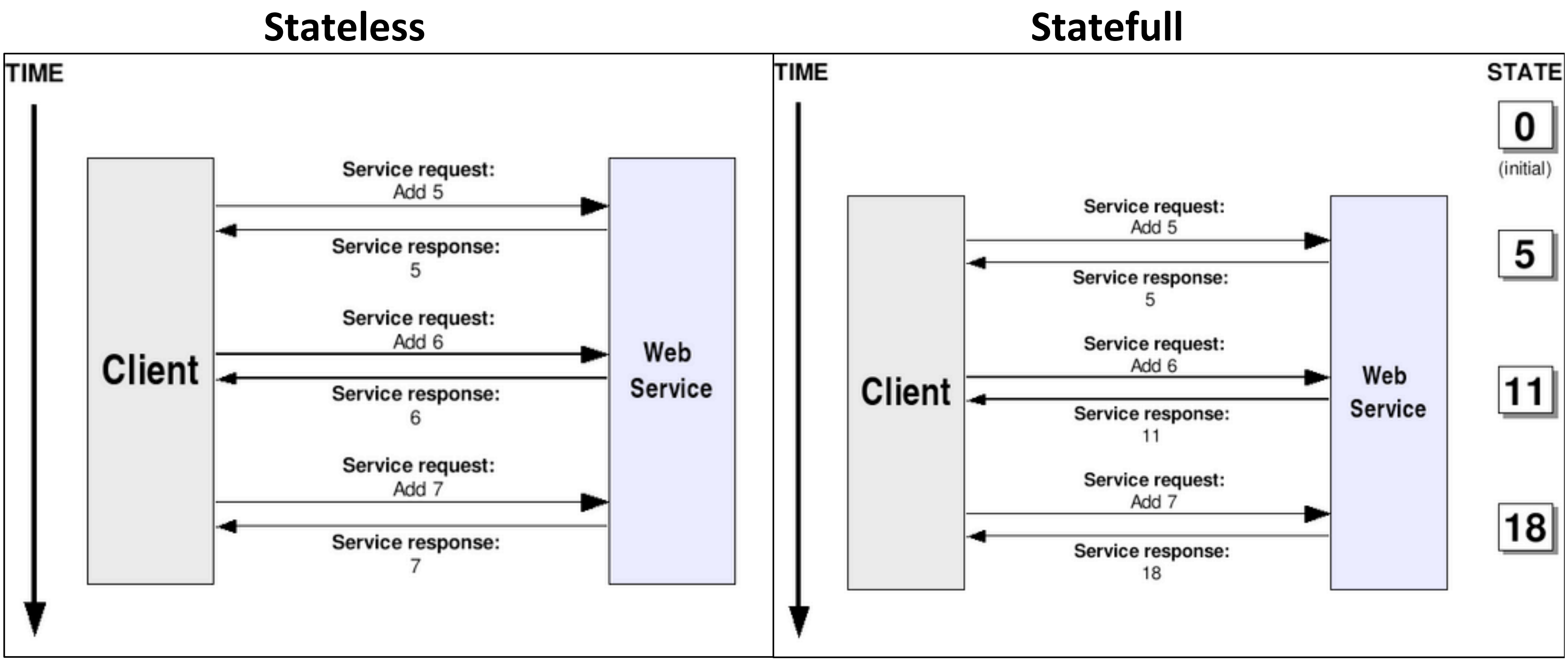
- GET https://api.del.icio.us/posts/delete
- GET www.example.com/registration?update=true&name=aaa&ph=123

- Let a client make reliable requests over an unreliable network:
  - Your GET request gets no response? Retry, it's ok
  - Your PUT request gets no response? Retry, it's ok
  - If you use POST, a new resource will be created each time, not safe to retry

Stateless Server

- RESTful services suggest that the state stay on the client side
- Server does not keep track of the state
- When a client makes a request, it includes all necessary information for the server to fulfill the request.

Stateless Servers



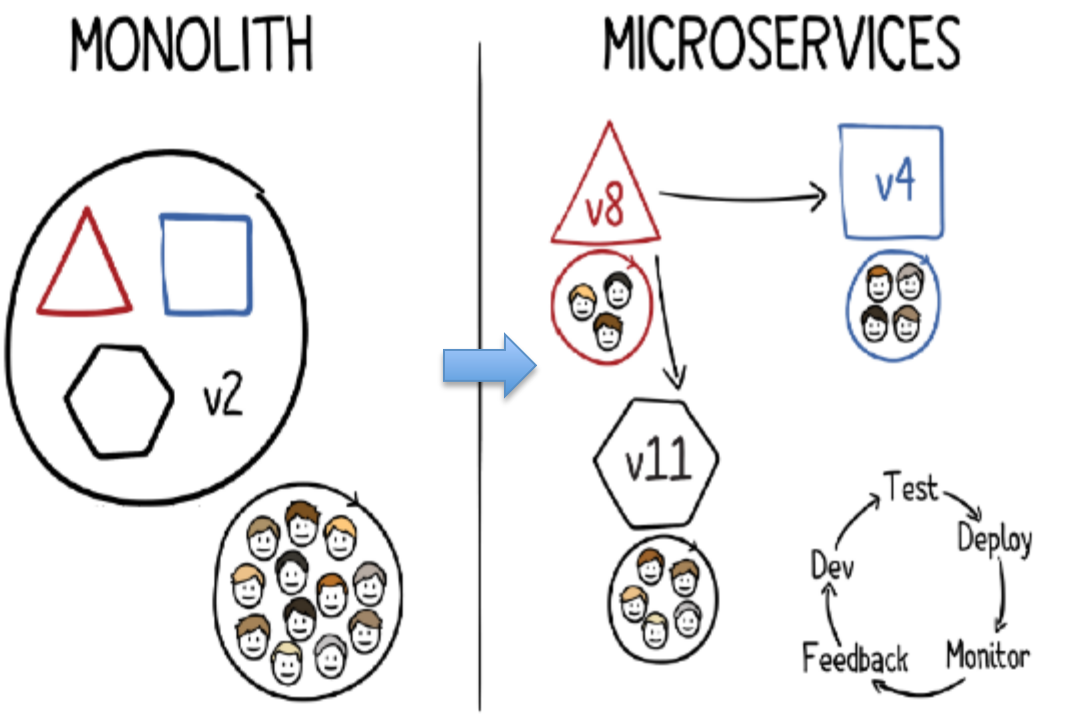
What about a database on the server side?

Microservices

Microservice-Based Architecture

- A modern interpretation of service-oriented architectures
  - A service is a component wrapped behind a standardized interface
  - Interactions between services is based on message exchange rather than direct calls, e.g., via HTTP/REST
- The term was coined at a workshop of software architects in May 2011
  - Described what the participants saw as a common architectural style that many of them had been exploring.
- Microservice-based architectures became really popular around 2014, with Martin Fowler's blog (assigned reading)
- (Mostly) used to build cloud-native systems

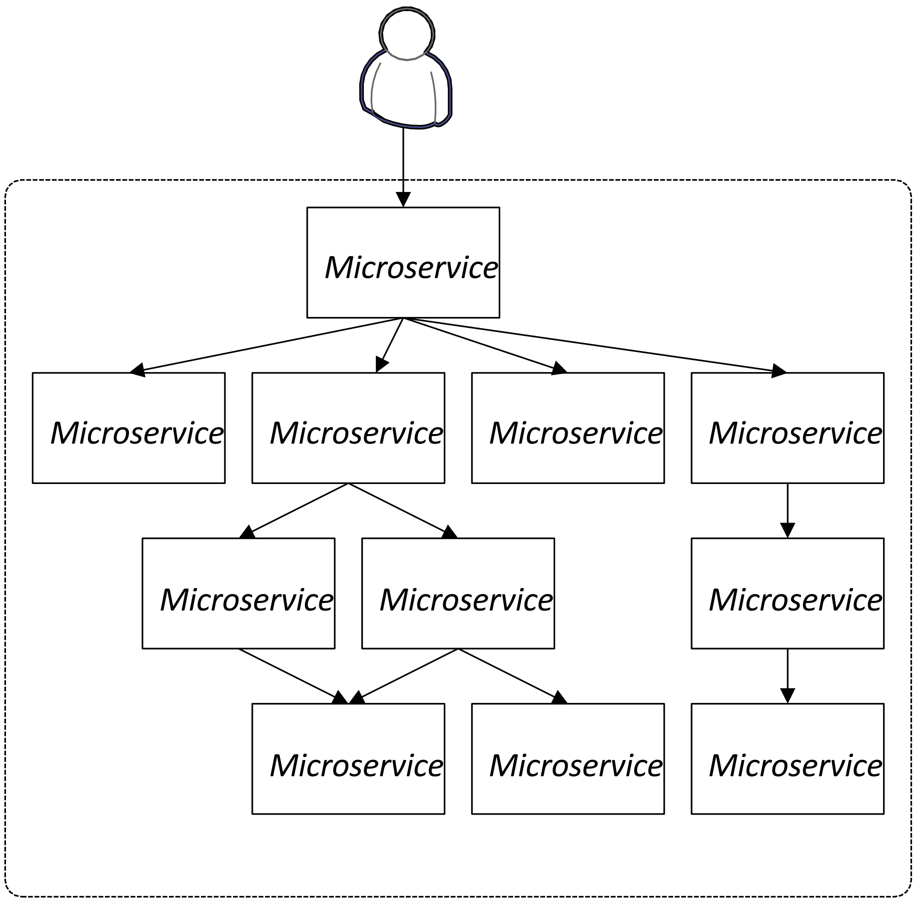
From monoliths to microservices



- *Developed independently*
- *Multilingual and multi-technology*
- *Communicate over lightweight interfaces (REST)*
- *Easy to deploy*
- *Scaled independently*

amazon ebay IBM HUAWEI UBER NETFLIX

Microservice Architecture – Core Idea



- Total system functionality comes from composing multiple services
- Each user request is satisfied by some sequence of services
- Most services are not
- externally available
- Each service communicates with other services through service interfaces
- Service depth may be 70, e.g., LinkedIn

Amazon Design Rules

- Each microservice provides a concrete functionality
- All teams will expose their data and functionality through service interfaces.
  - Teams must communicate with each other through these interfaces.
  - There will be no other form of inter-process communication allowed: no direct linking, no direct reads of another team's data store, no shared-memory model, no back-doors whatsoever.
- It doesn't matter what technology the services use.

Advantages of Microservices: Developers View

- Shortens development lifecycle
- Reduces communication and coordination effort of synchronizing on delivery cycles
- Reinforces the component abstraction
- Makes it easier to maintain clear boundaries between components
- Improves maintainability and dealing with legacy by focusing on a small part of an application
- Gives freedom to choose different languages, frameworks, and tools (e.g., python for ML, JavaScript for front-end, Scala for the backend)

At Runtime

- Usually deployed inside containers, e.g., Docker
- Can be individually scaled by adding more instances
  - For microservices that experience increasing traffic
  - Improve the availability and scalability of applications at runtime
- Managed by container-orchestration system, e.g., Kubernetes
- Easy blue-green deployments
- …

The Illustrated Children's Guide to Kubernetes:
https://www.youtube.com/watch?v=4ht22ReBjno

Best Practices

Concept	# Participants
Best Practices	
Clear sense of ownership	13
Strict API management	12
Automated setup processes	7
Robust logging and monitoring	9
Distributed tracing	4

More in following lectures

Microservice Granularity (19)	
Business capabilities	19
Data access	3
Team structure	5
Dependencies	3
Resource utilization	3
Delivery cycles	1

More Info on REST / Microservices

- https://en.wikipedia.org/wiki/Microservices
- https://martinfowler.com/articles/microservices.html
- https://microservices.io/
- RESTful Web Services, L. Richardson and S. Ruby, O'Reilly
- Web Services: Concepts, Architectures and Applications

Exercise: Restaurant Management System

You are building a food delivery system, where customers can browse and select food items, pay, and order delivery.

1. **How many microservices will be part of your backend, what is their role, and what is their name?**

Exercise: Restaurant Management System

You are building a food delivery system, where customers can browse and select food items, pay, and order delivery.

1. **Which microservices will be part of your backend?**

- Menu
- Orders
- Payments
- Delivery

Exercise:
Restaurant Management System

Consider the following scenario:

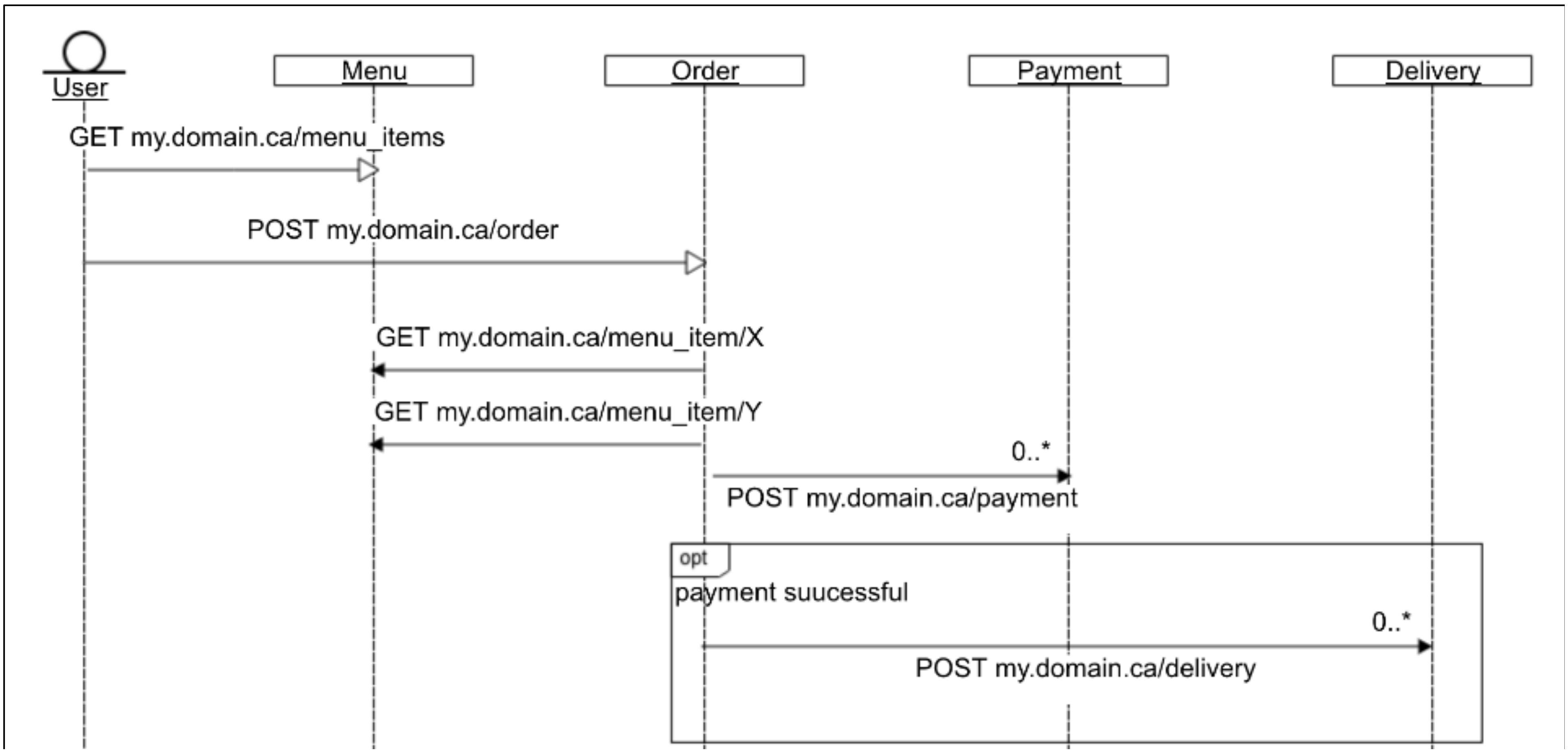
- *a customer browses the menu,*
- *picks items X and Y,*
- *pays with their credit card C,*
- *and orders delivery to the home address A.*

2. Identify main REST APIs needed to implement this scenario and draw a sequence diagram showing their interactions

Reminder:

GET: Retrieve a representation of a resource.
POST: Create a new resource.*
PUT: Update a resource (existing URI).*
DELETE: Clear a resource

- Menu
  GET my.domain.ca/menu_items
  GET my.domain.ca/menu_item/id
- Order
  POST my.domain.ca/order
  items: X, Y; credit info: C;
  delivery address: A
- Payments
  POST my.domain.ca/payment
  amount: XXX; credit info: C
- Delivery
  POST my.domain.ca/delivery
  items: X, Y;
  delivery address: A



Quiz

Here are four REST APIs:

1. List all cars in a database: GET my.domain.ca/cars
2. List all BMW cars: GET my.domain.ca/cars/bmw
3. Delete all cars: GET my.domain.ca/cars/delete
4. Delete all BMW cars: GET my.domain.ca/delete/bmw

Which APIs are inadequate:

A: 2 and 4 **B**: 3 **C**: 4 **D**: 3 and 4 **E**: 2, 3 and 4

Recap so Far

- Architectural principles
  - Single responsibility, clear interfaces, high cohesion and low coupling, high fan-in low fan-out, DRY, KISS, ...
- Patterns
  - Best practices for a particular problem
- REST
  - Conventions for service communication
  - Uniform interface to resources
- Microservices
  - Service orientation
  - Split application into small, well-scoped components
  - Communication only through standard interfaces

Combining Architectural Patterns

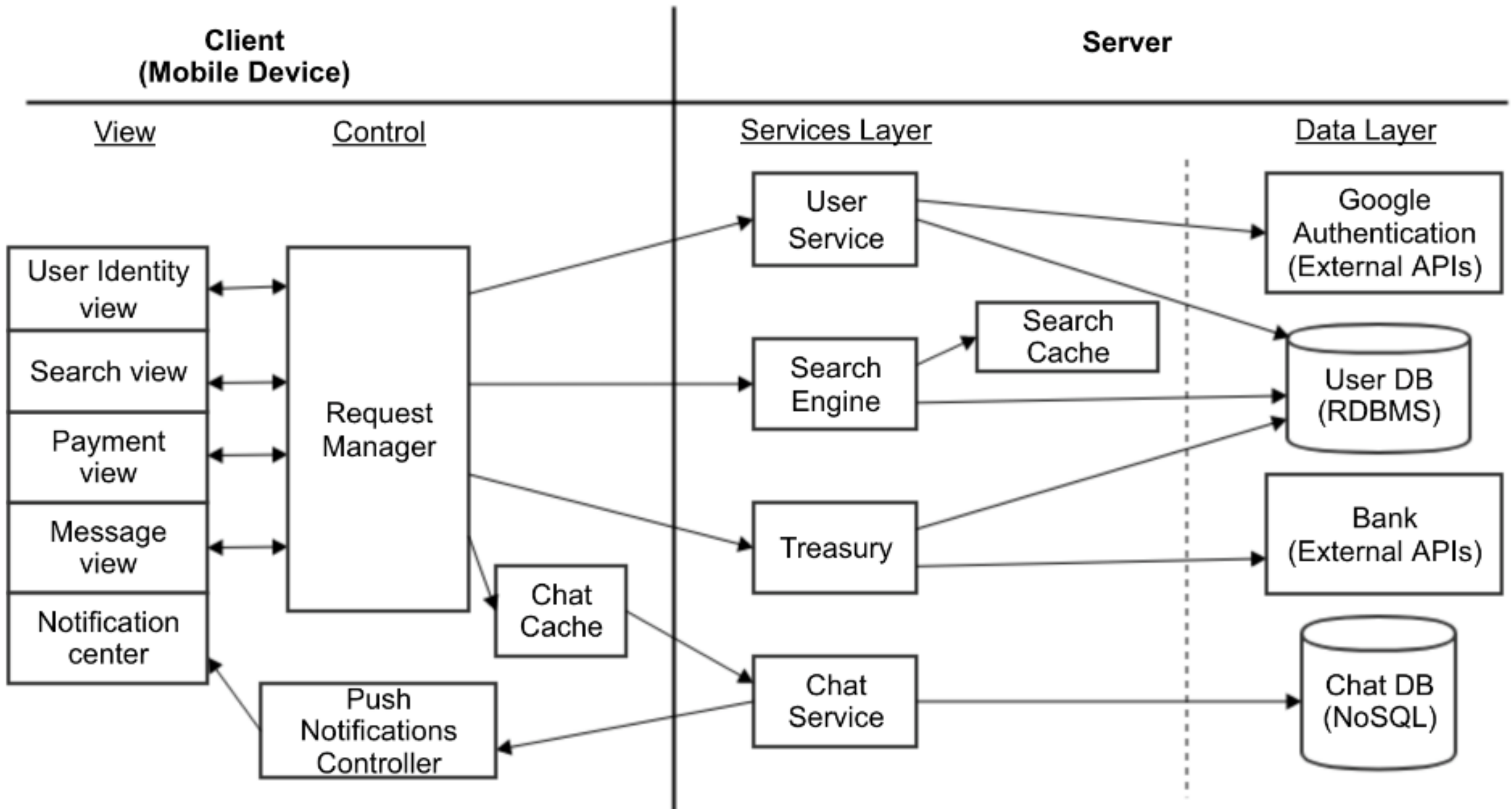
- The architecture is almost never limited to a single architectural pattern
  - Often a combination of architectural styles that make up the complete system

- For example:
  - Overall: client-server
  - Client: MVC
  - Server: microservices-based architecture
  - Communication pattern: message bus
  - A particular microservice: layered application

Expectations for the Next Milestone (Again)

- Main components, roles, and responsibilities
- Which architectural patterns are used and why (for both client and server)
- Interaction protocols (which and why)
- Data store -> which data, type, why and what were the alternatives
- Main frameworks and tools used, why, what were the alternatives
- How your non-functional requirements are realized
- Main algorithms

Example: High-Level Architecture of the Dating App



+ All explanations in text

Next two weeks

- Monday, October 8: Thanksgiving Day
- Wednesday, October 10: Midterm 1
  - Processes
  - Requirements
  - Architecture and Design
  - REST, Microservices
  - Assigned reading
  - …
- No community meetings during the week of October 15 (W7); classes as usual

Following Milestones

- W4: Development team and the customer discuss the requirements.
- W5: M1 – Requirements (both customer and development teams).
- W6: M2 – Design (development team).
- W8: M3 – MVP (development team). ← Getting close!
- W9: M4 – Code review (development teams).
- W10: M5 – Test plan and results (development team).
- W11: M6 – Refined specifications (both customer and development teams).
- W12: M7 – Customer acceptance test (customer team).


HAPPY
THANKSGIVING