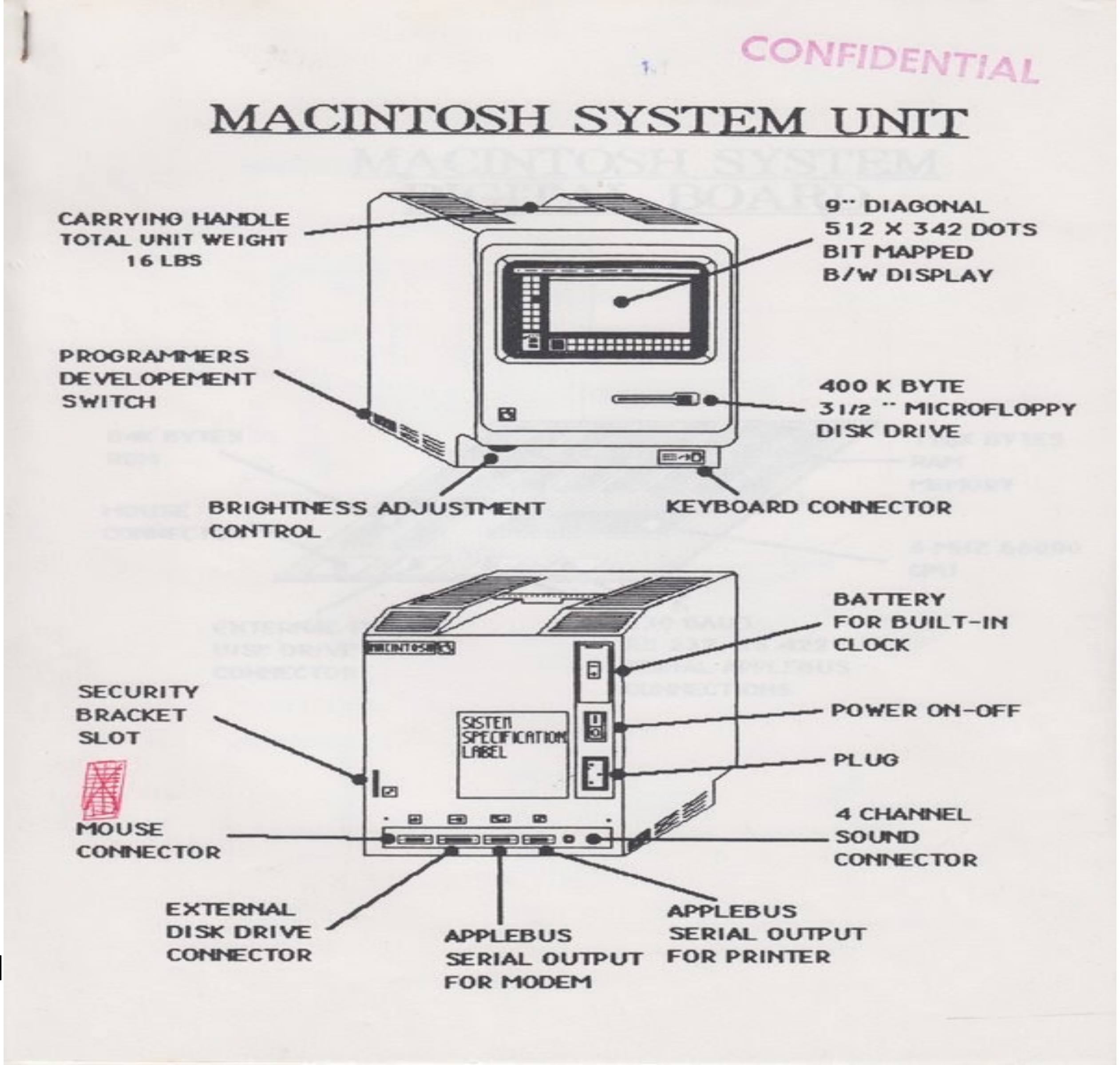

CONFIDENTIAL
MACINTOSH SYSTEM UNIT
CARRYING HANDLE TOTAL UNIT WEIGHT 16 LBS
9" DIAGONAL 512 X 342 DOTS BIT MAPPED B/W DISPLAY
PROGRAMMERS DEVELOPEMENT SWITCH
400 K BYTE 3 1/2 " MICROFLOPPY DISK DRIVE
BRIGHTNESS ADJUSTMENT CONTROL
KEYBOARD CONNECTOR
BATTERY FOR BUILT-IN CLOCK
MACINTOSH
SECURITY BRACKET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

CPEN 321

Architecture and Design

Agenda

- Top project ideas!
- Announcements and Reminders
- Architecture and Design

Top Ideas!

	Team	% student votes	% instructor votes
1st	**The Last Brain Cell**; Idea2: PlotPal	58%	100%
2nd	**SCAM-mers**; Idea1: RunIO	76%	66%
3rd	**Karate Kids**; Idea2: Bartender Simulator	53%	83%

Announcements and Reminders

1. MVP (12%) - Friday Oct 27, 9pm
   – Fully functional product. **Start now!**
2. Azure – individual allocations are deactivated;
   see Piazza for instructions for how to ask for group allocations
3. Discord – join voice and text channel for your group
   – Will be made private over the weekend
4. If sick: do not come
   – Send Instructors a private Piazza message and we will arrange a solution
5. Reminder: we do not read notes on assignment in Canvas
   – All notes should be in the submission pdfs

Agenda

- Top project ideas!
- Announcements and Reminders
- **Architecture and Design**

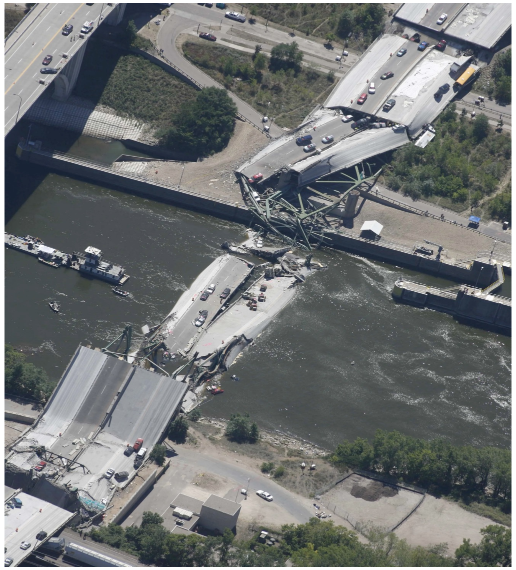
Good Design

There is no single correct design (or answer to any real SE question)

Good Design

There is no single correct design (or answer to any real SE question)

... but there are many incorrect designs (answers)



About Good Design (before you build)

Design Step #1: Identify main modules (components)

Design Step #2: Identify interfaces between modules

Terminology

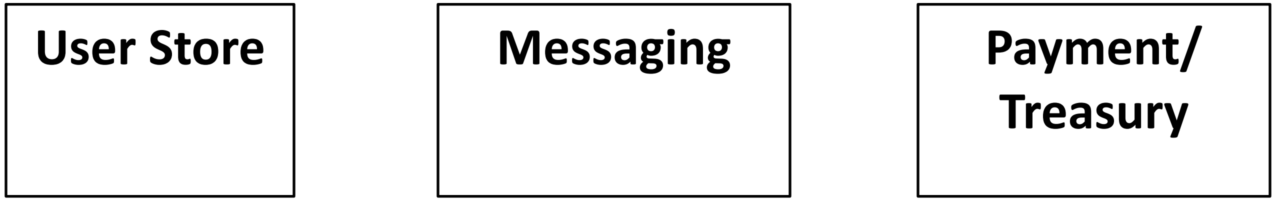
- “Architecture” and “High-level Design” are terms that are often used interchangeably
- The term “low-level design” is often used to describe the detailed design of individual modules

- Modules, subsystems, components, etc. are terms that are often used interchangeably

Our Online Dating System from Last Week

- You are building an online dating system.
- The client will attract customers by providing free browsing and matching functionality, but charging for allowing users to contact other users.
  - For example, a user should be able to register, create a profile, and search for “soul mates”, all without paying.
  - Then, if they want to send a message to another user, or receive a message from another user, they need to upgrade their membership by making a payment.
- The client should be able to find and ban “offensive” users.

Identify main modules...



Agenda

- Top project ideas!
- Announcements
- **Architecture and Design**
  - **What is a good module**
  - How to define interfaces
  - REST interfaces (next week)
  - Some popular architectural patterns (next week)

Single Responsibility Principle

- Every module should have a single responsibility
- The responsibility should be entirely encapsulated by the module
- All module functions should be aligned with that responsibility

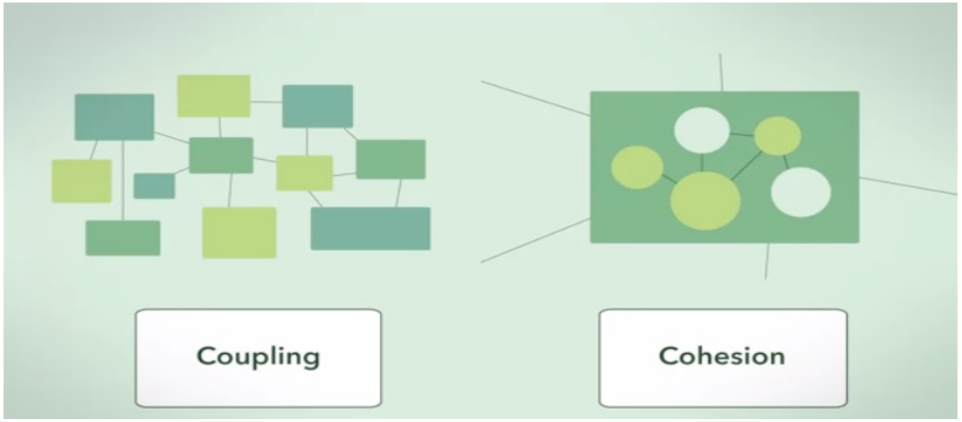
- Why?
  - easier to understand
  - easier to test
  - easier to maintain
  - easier to replace

Test: Can you easily name it?

Low Coupling / High Cohesion Principle

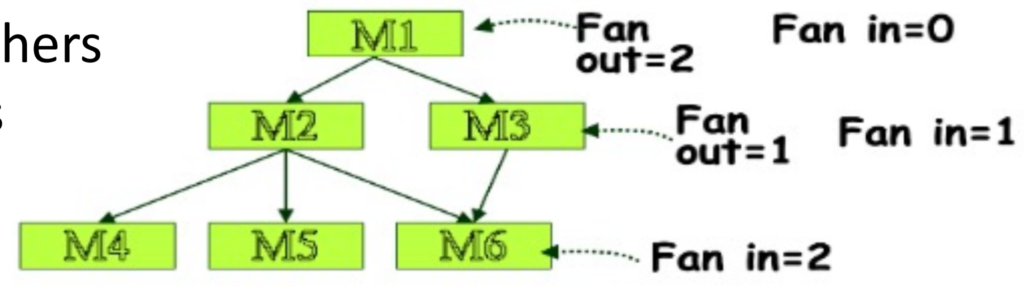
- **Cohesion**: the degree to which the elements of a module belong together (related code should be close to each other)
- **Coupling**: the degree to which the different modules depend on each other (modules should be independent as much as possible)
  - Data coupling
  - Control coupling

Test: How often do modules interact with each other?

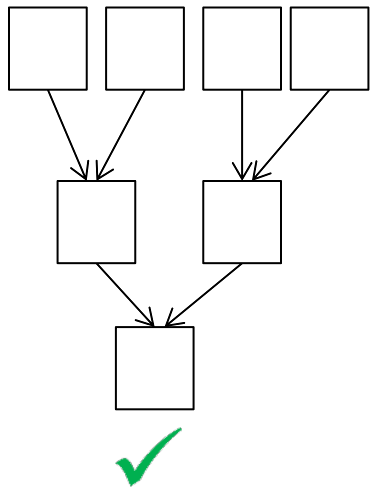


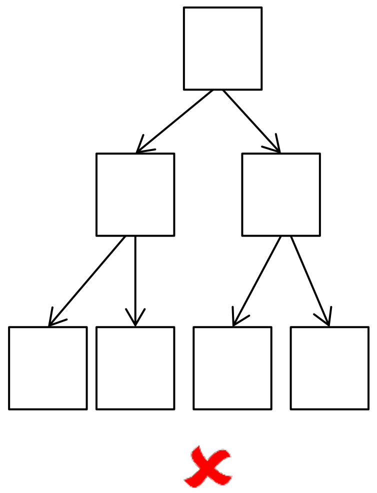
High Fan-in / Low Fan-out Principle

- Have a module used by many others
- Do not use many other modules
  - A module with high fan-out lacks cohesion



- Why?
  - complexity management
  - understandability
  - maintenance
  - reuse
  - …





Principle of Least Knowledge

- Keep only the information and resources absolutely necessary for the module
- The module should assume as little as possible (better: nothing) about the structure or properties of any other modules
  - Pass all the needed info as parameters

Do Not Repeat Yourself (DRY)

- Implement all functions once and only once
  - Duplications and clones make the system susceptible to errors

- If functionality is duplicated, think why
  - Usually a bad design and requires refactoring

KISS

- Make is simple and easy to understand
- In SE: simple is good!

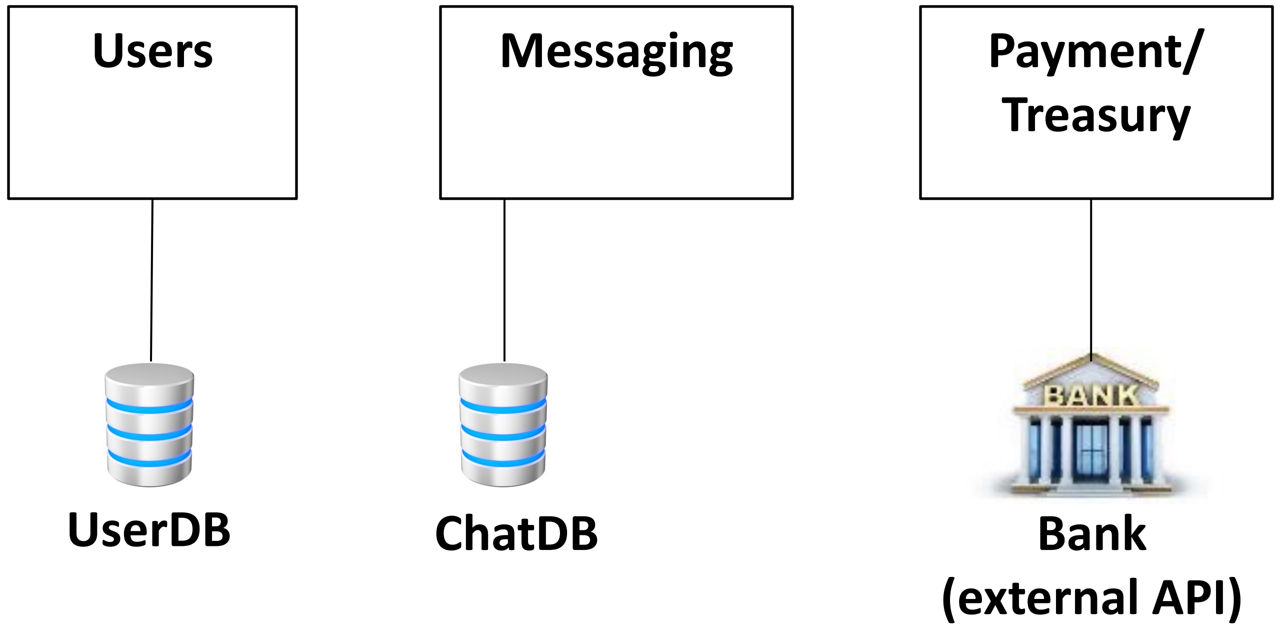
Why?

- Understandability
- Maintainability
- Testability

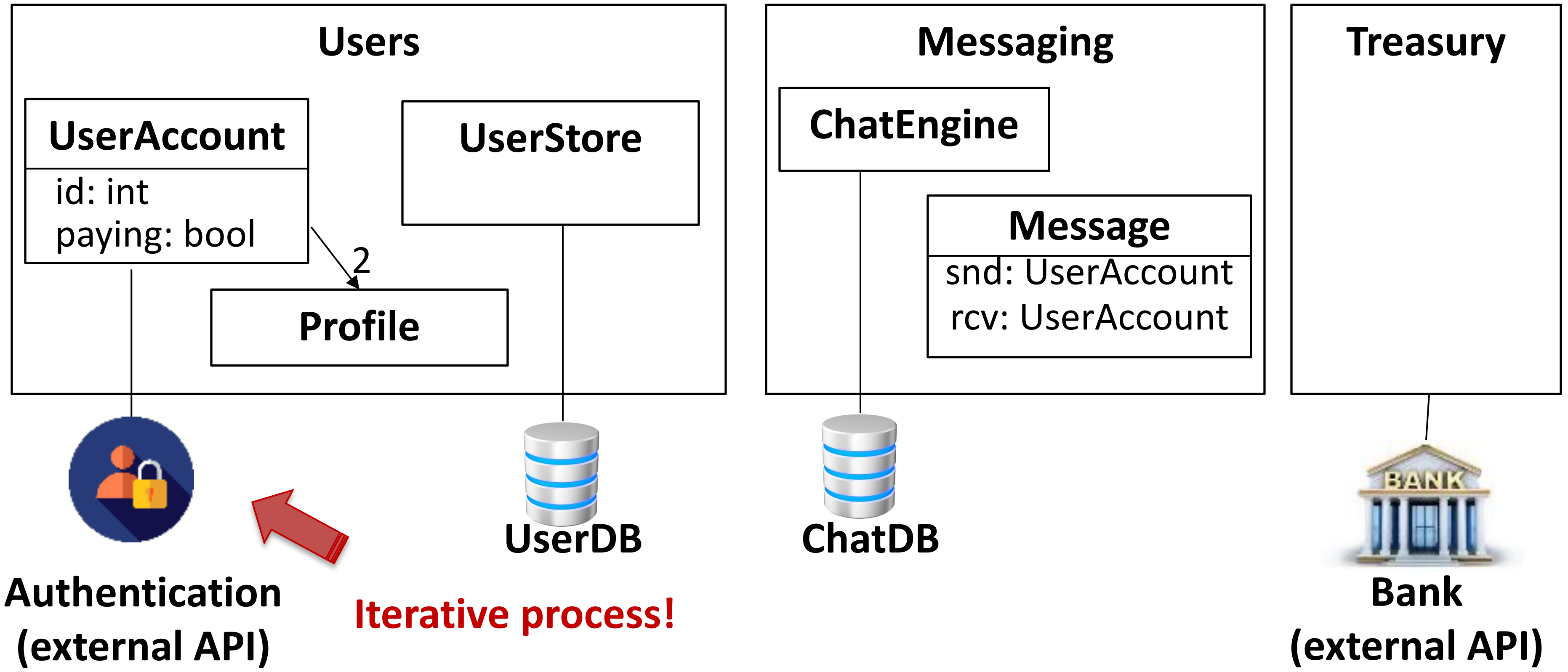
Summary: Core Architectural Principles (Checklist)

- **Single Responsibility Principle:** Each module should be responsible for only a specific feature or functionality, or aggregation of cohesive functionalities.
- **Separation of Concerns:** Minimize interaction points to achieve high cohesion and low coupling.
- **Independence**: Have modules that are highly used but do not use many other modules.
- **Principle of Least Knowledge:** A module should not know about internal details of other modules.
- **Don’t Repeat Yourself (DRY)**: Do not duplicate functionality.
- **KISS**: Make it simple. Only focus on what is needed.

Our Online Dating System

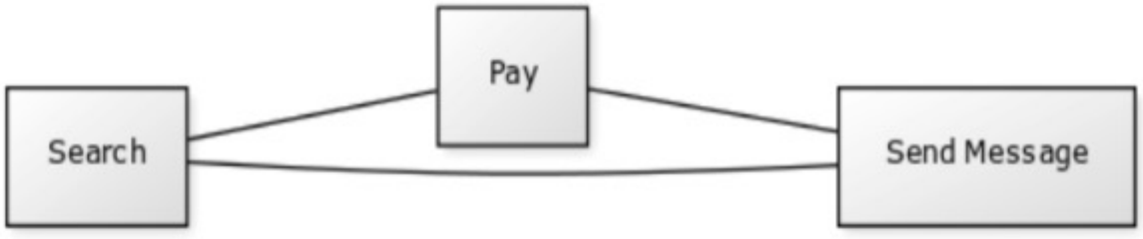


Components have internal classes…



Iterative process!

Notes



Wrong!

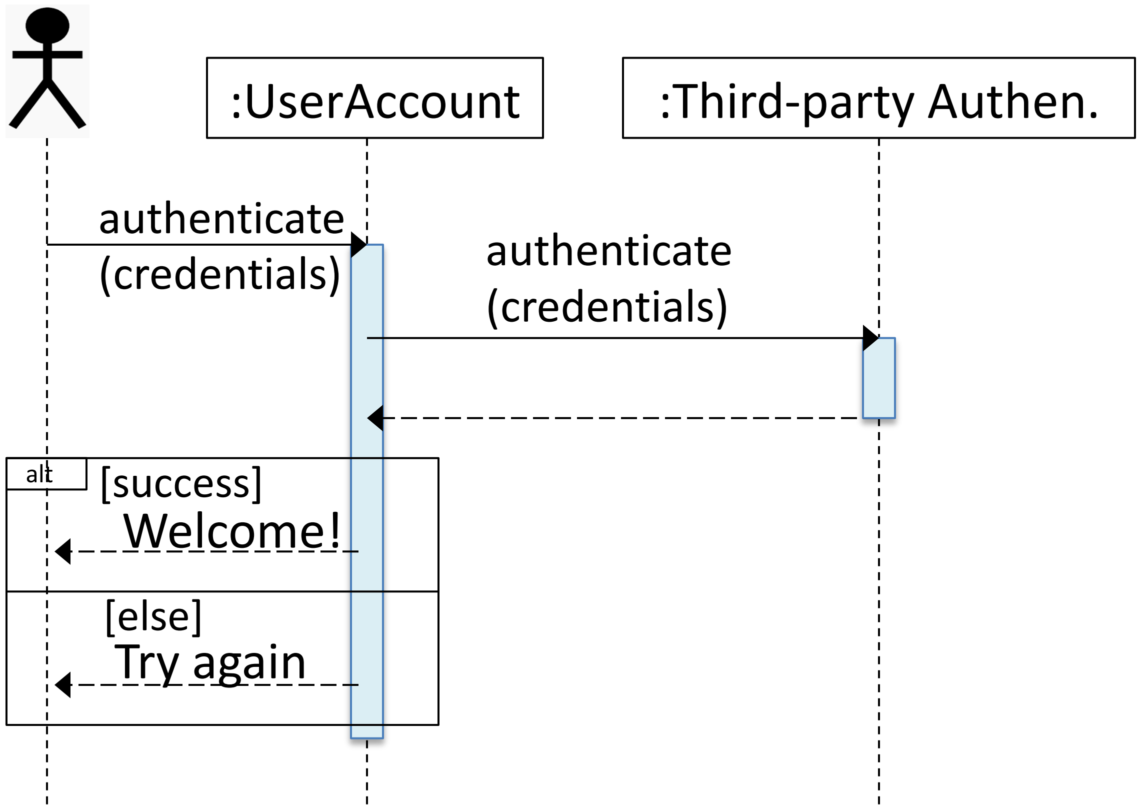
- Look for nouns, not verbs

- More formally: break a large system down into progressively smaller components or classes that are responsible for some part of the problem domain

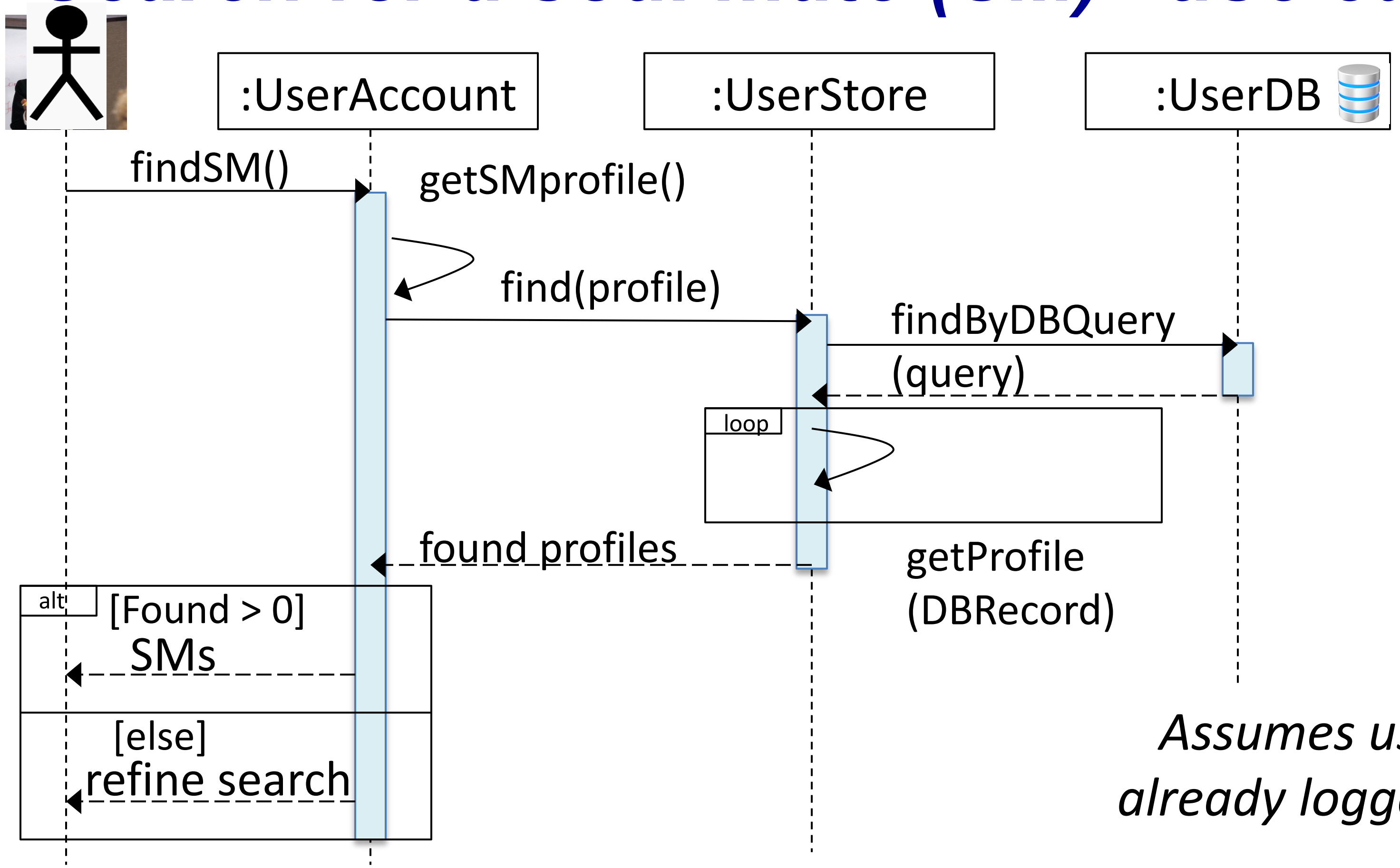
Are the Identified Modules Good?

- Check if they are sufficient to satisfy all requirements
  - Define sequence diagrams to elaborate on each use case
  - Derive interfaces
  - Specify ways to satisfy non-functional requirements
  - Refine main modules and <u>repeat</u>!

A sequence diagram for the “Login” use case

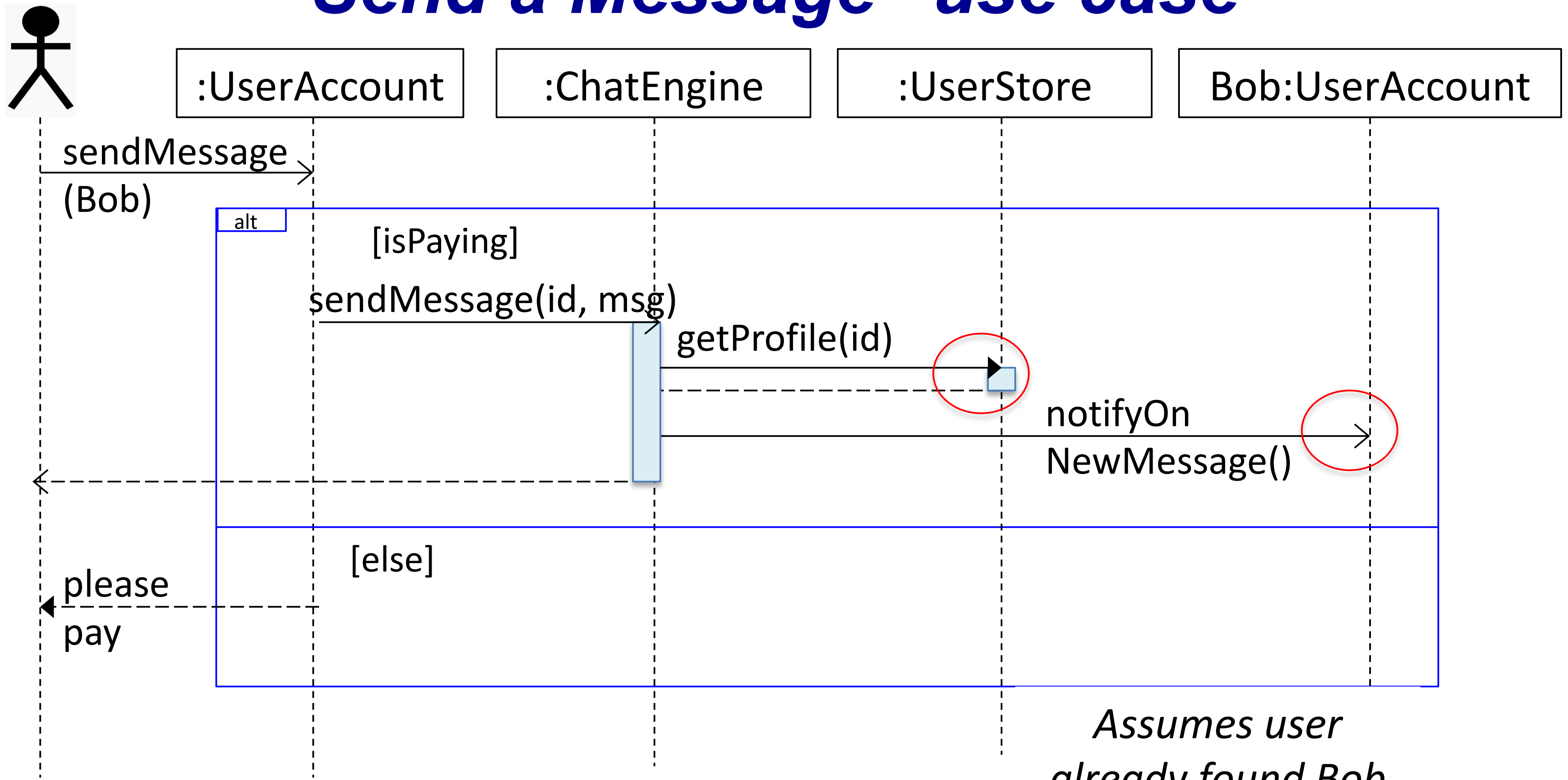


A sequence diagram for the "Search for a soul mate (SM)" use case



Assumes user already logged in

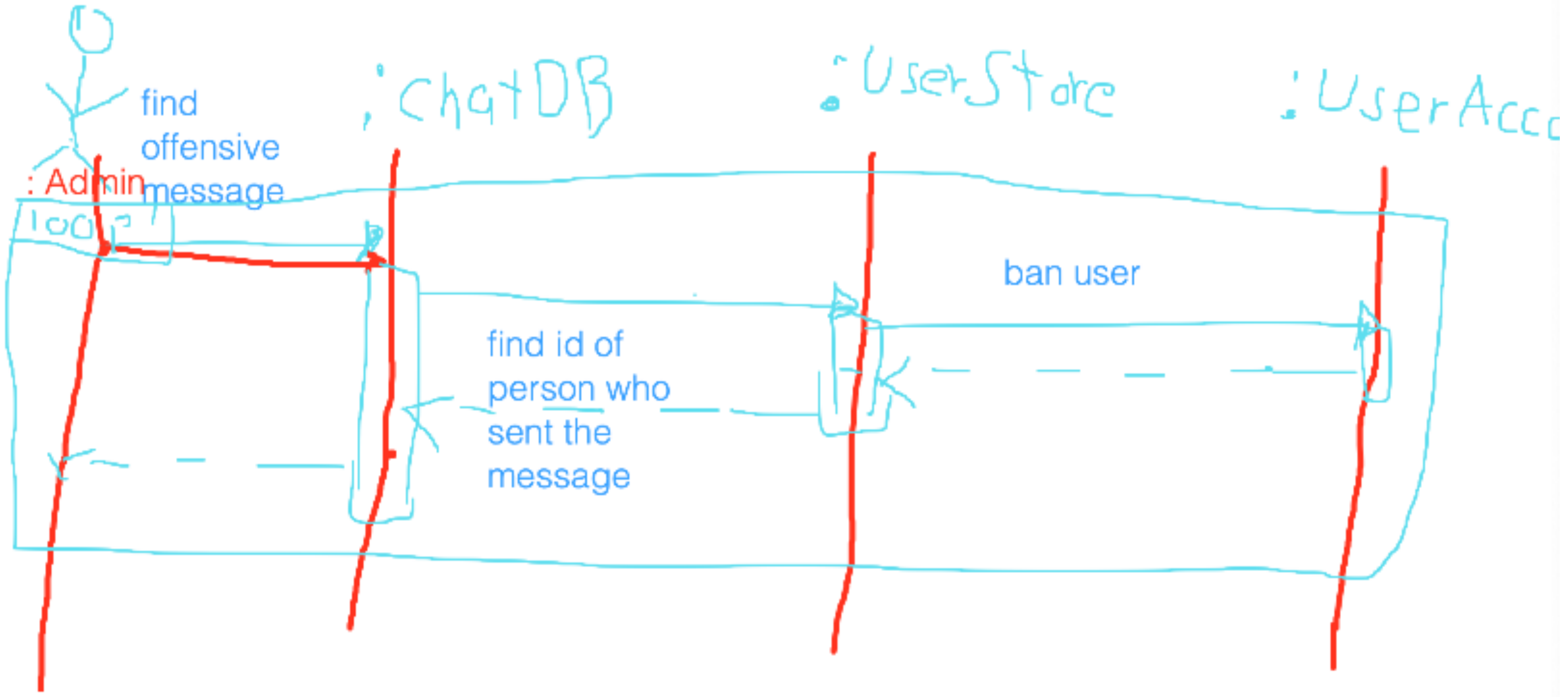
A sequence diagram for the "Send a Message" use case



Assumes user already found Bob and has their id

Now your turn:
A sequence diagram for the "Find and Ban Offensive Users" use case

Now your turn: A sequence diagram for the "Find and Ban Offensive Users" use case



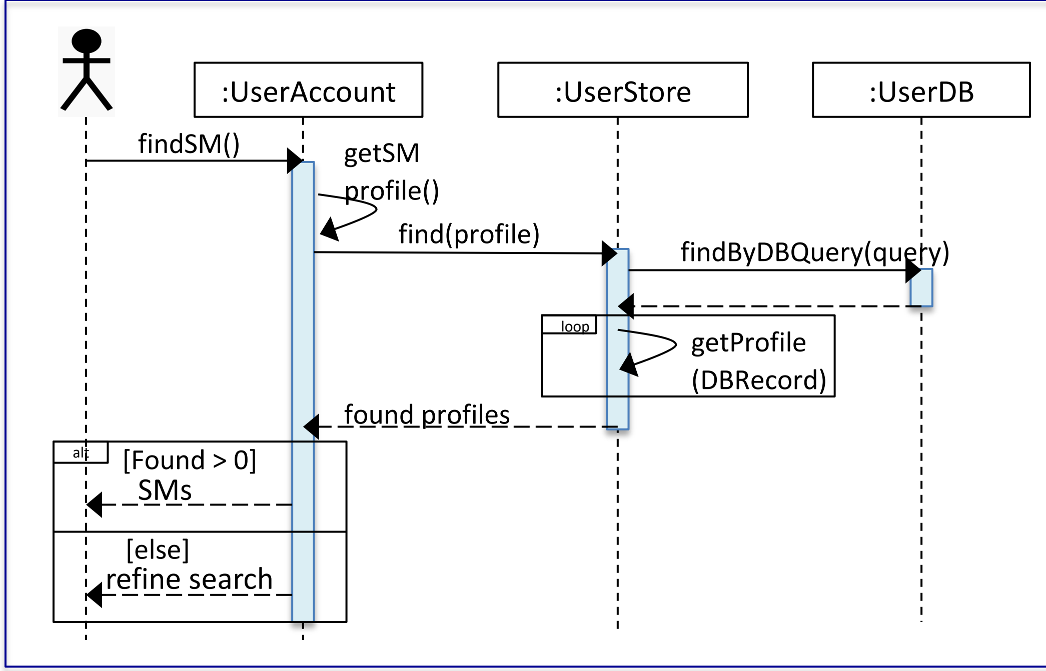
Common Mistakes

1. Be descriptive: “ban user” is not descriptive enough, need to elaborate how
   - E.g., find in the database, update status to inactive, update profile to not searchable, etc.
2. Messages should be labeled with appropriate interfaces: include both input parameters and return values
3. Consider success and fail paths
4. Make sure the information flows between players rather than coming out of nowhere
   - If you need to search by ID, the ID should be retrieved first

Agenda

- Top project ideas!
- Announcements
- Architecture and Design
  - What is a good module
  - **How to define interfaces**
  - Rest interfaces (next week)
  - Some popular architectural patterns

“Search for a soul mate (SM)” use case



Profile

UserAccount

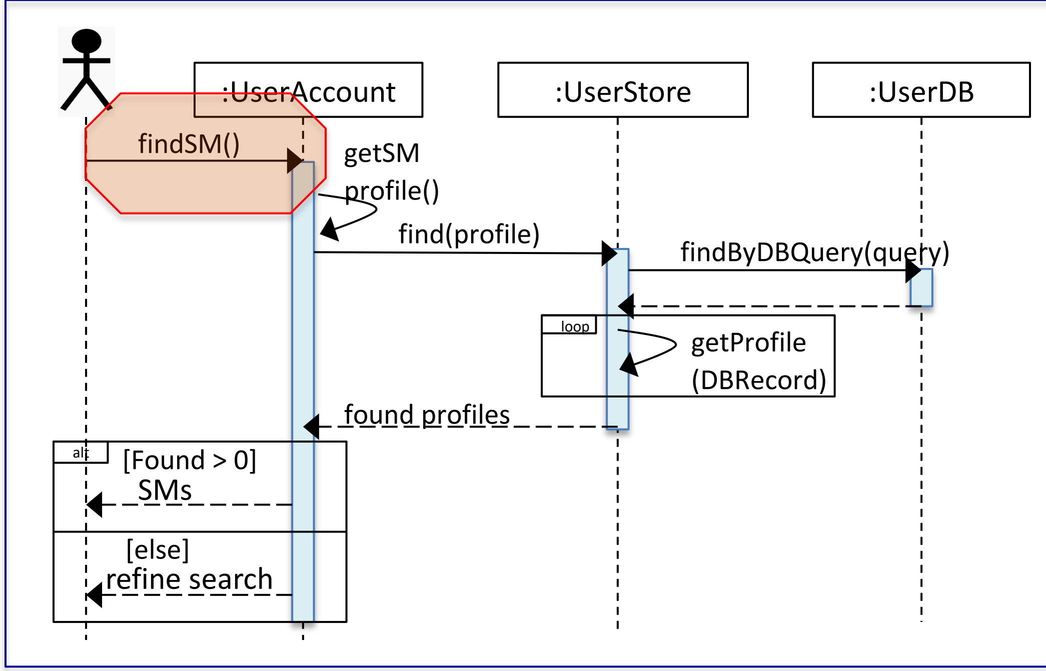
UserStore

ChatEngine

Message

Treasury

"Search for a soul mate (SM)" use case



Profile

UserAccount

UserAccount[] findSM()

UserStore

ChatEngine

Message

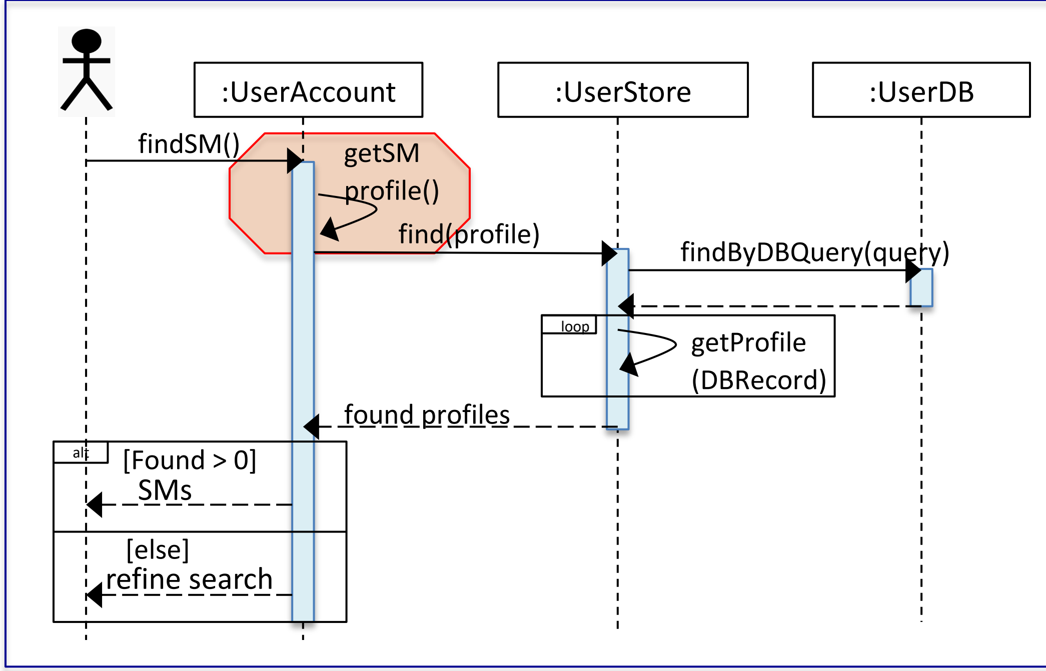
Treasury

“Search for a soul mate (SM)” use case

Profile

UserAccount
UserAccount[] findSM()
Profile getSMProfile()

UserStore

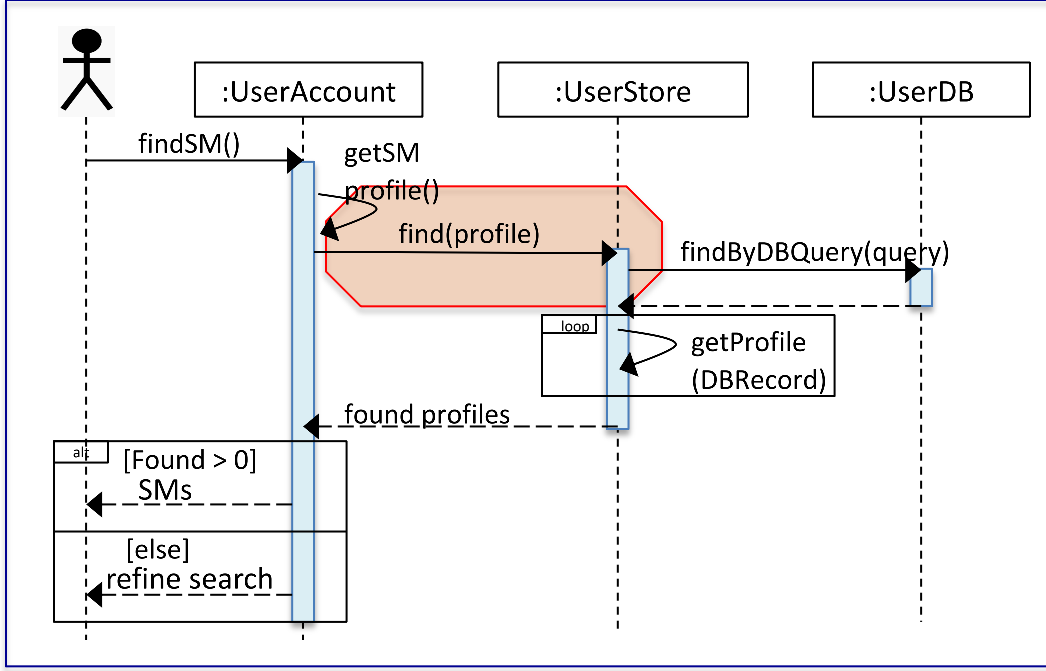


ChatEngine

Message

Treasury

“Search for a soul mate (SM)” use case



Profile

UserAccount
UserAccount[] findSM()
Profile getSMProfile()

UserStore
UserAccount[] find(Profile)

ChatEngine

Message

Treasury

“Search for a soul mate (SM)” use case

Profile

UserAccount

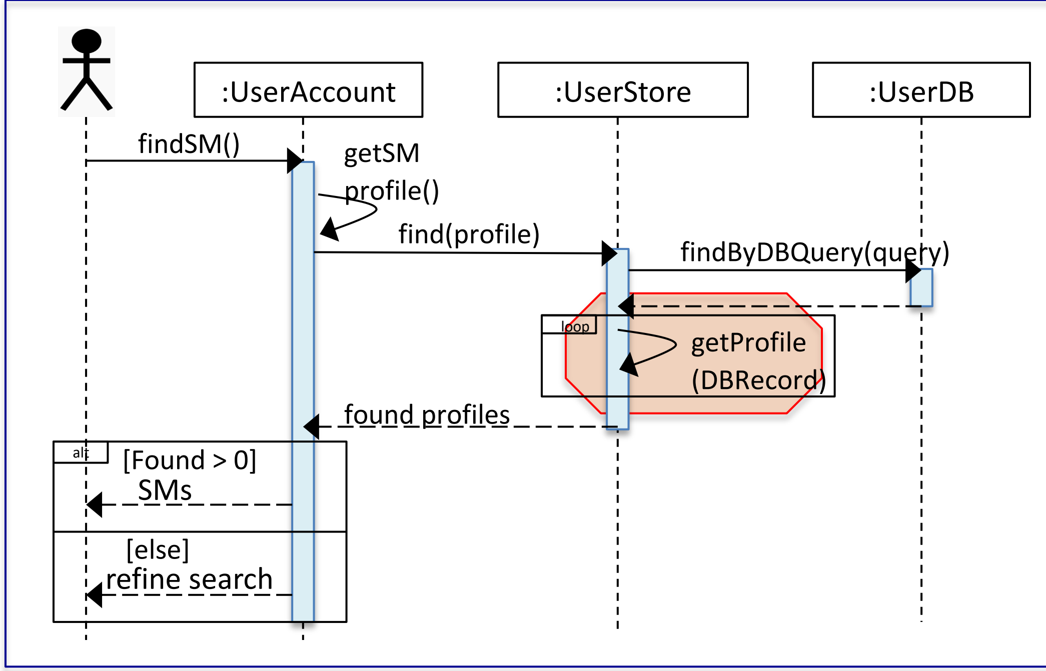
UserAccount[] findSM()

Profile getSMProfile()

UserStore

UserAccount[] find(Profile)

Profile getProfile(DBRecord)

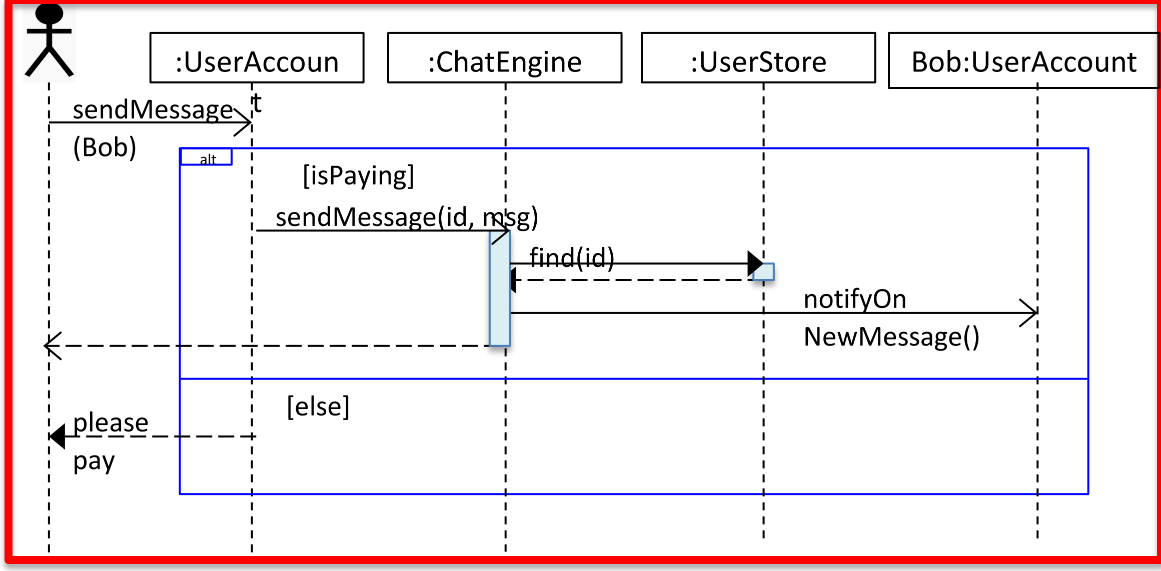


ChatEngine

Message

Treasury

"Send a Message" use case



Profile

UserAccount

UserAccount[] findSM()
Profile getSMProfile()

UserStore

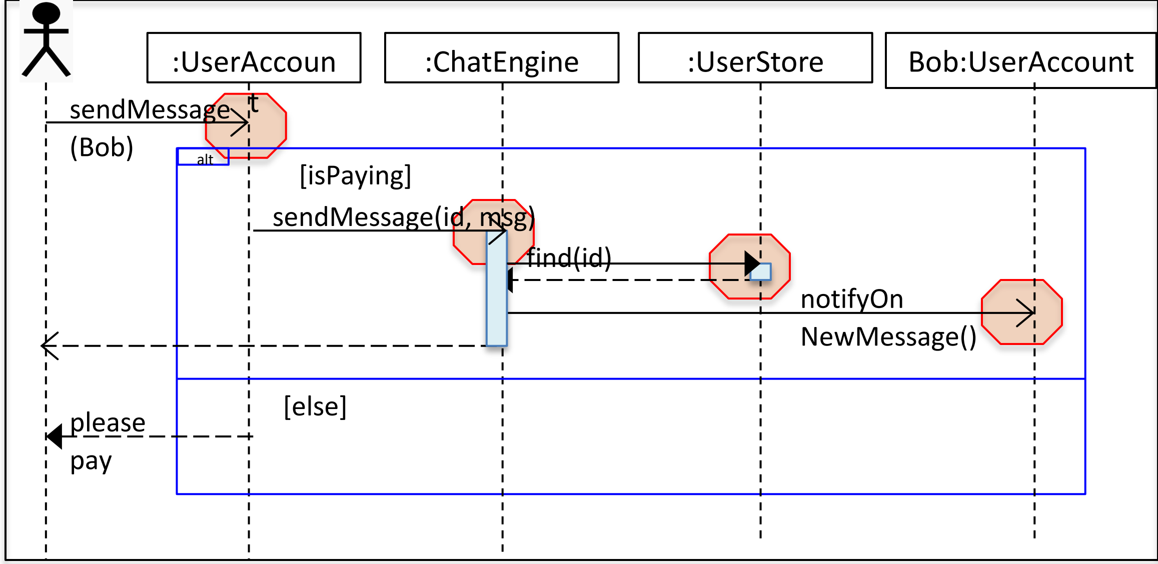
UserAccount[] find(Profile)
Profile getProfile(DBRecord)

ChatEngine

Message

Treasury

"Send a Message" use case



Profile

UserAccount
- UserAccount[] findSM()
- Profile getSMProfile()
- bool sendMessage(int id, msg)
- notifyOnNewMessage()

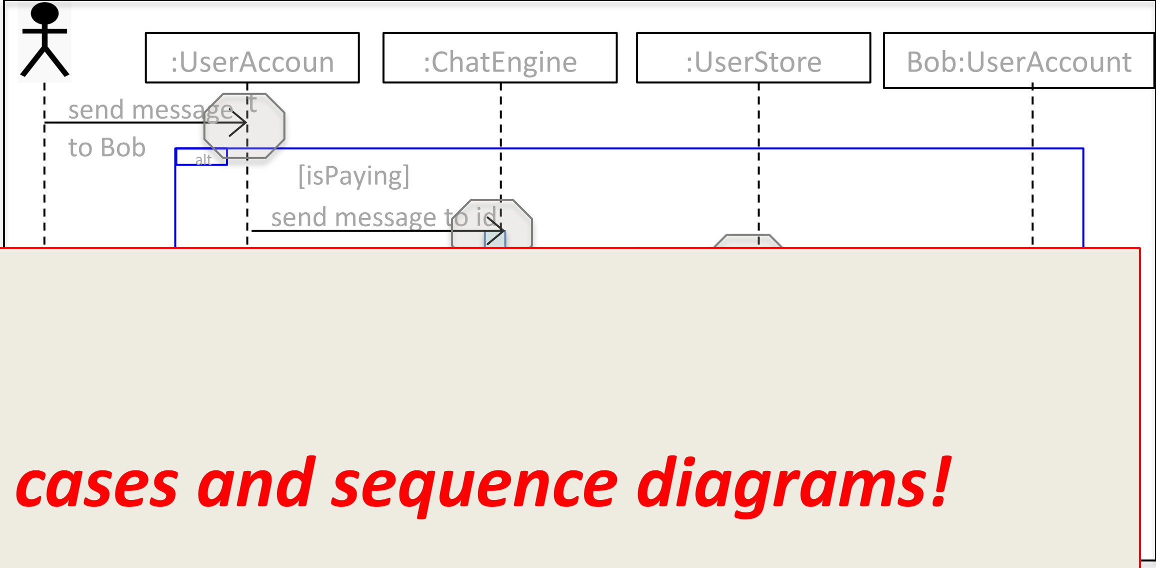
UserStore
- UserAccount[] find(Profile)
- UserAccount find(id)
- Profile getProfile(DBRecord)

ChatEngine
- bool sendMsg(id, msg)

Message

Treasury

"Send a Message" use case



Repeat for all use cases and sequence diagrams!

notifyOnNewMessage()

UserStore

UserAccount[] find(Profile)
UserAccount find(id)
Profile getProfile(DBRecord)

sendMsg(id, msg)

Message

Treasury

Main Points

- Collect info from multiple use cases
- Cannot meet requirements? Identified new components? Repeat!

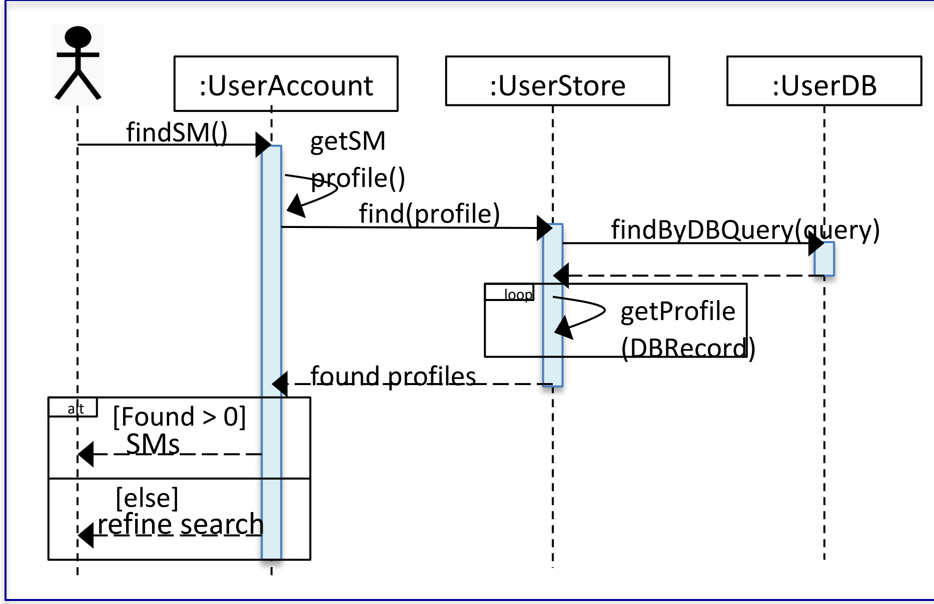
Is there a correct answer?

- *There are many correct answers (think about building a bridge)*
- *There are many incorrect answers (think about a bridge that collapses)*

Common Mistakes

- All objects (use cases, classes, interfaces) must have descriptive names (+ you will need to include textual explanations when their role is not obvious from the context)
- Meaningful and consistent names (methods and parameters)
  - Either *remove* or *delete*, everywhere
- Focus on interactions between the main players for accomplishing each task (rather than internal implementation details of each player)
  - KISS: focus on 2-5 interacting objects and include sub-procedures when needed
- Aim to be complete, e.g., if you have *add*, you should have *remove*

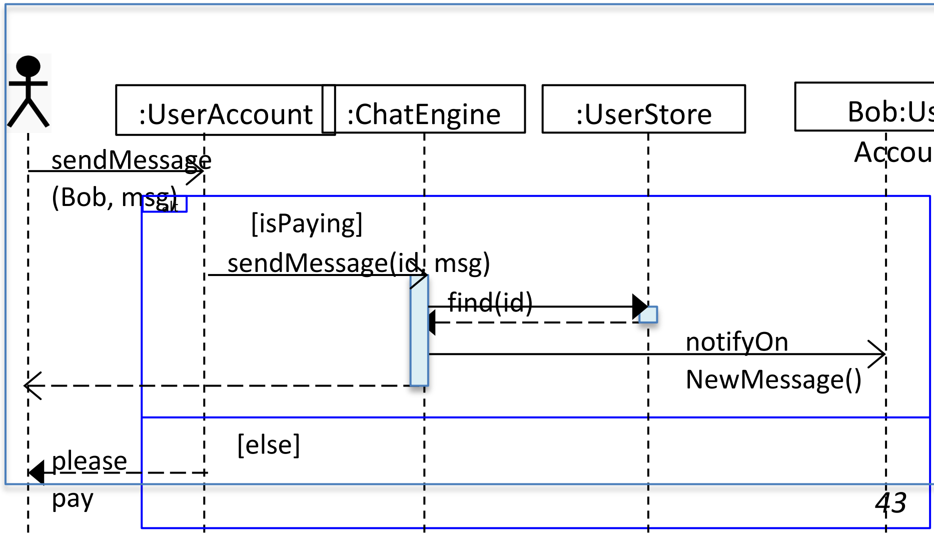
Now: External Interfaces!



<u>Users</u>:

<u>Messages:</u>

<u>Payment:</u>



CPEN 321 | Software Engineering | Winter 2023 | UBC

Now: External Interfaces!

Users:

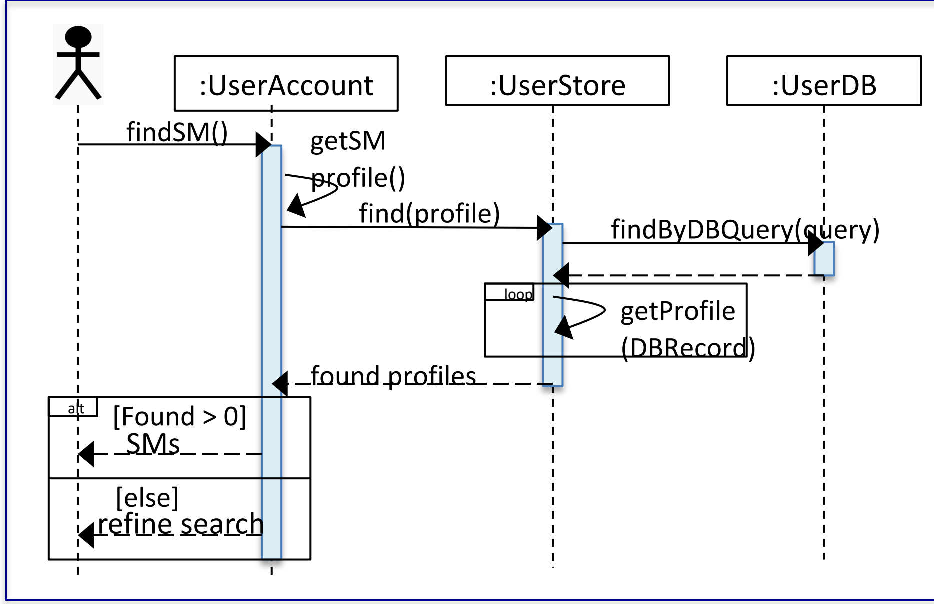
- UserAccount[] findSM()
- bool sendMessage(String id, String msg)
- notifyOnNewMessage()

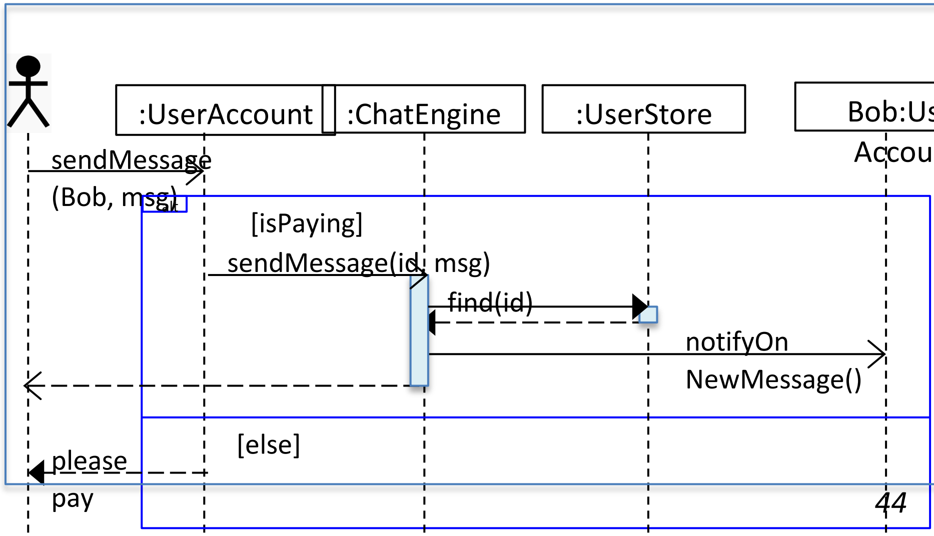
Messages:

- bool sendMessage(Int id, String msg)

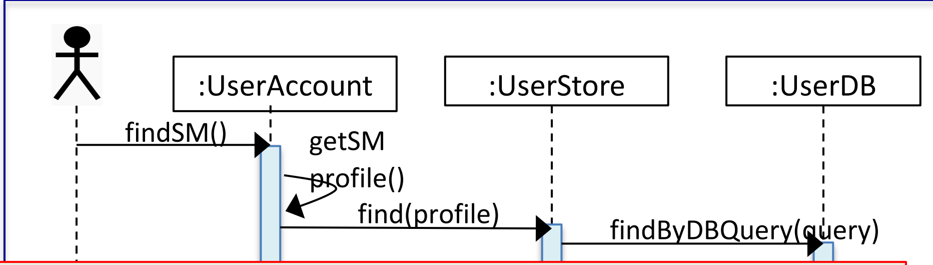
Payment:

- pay(String id, String ccInfo)



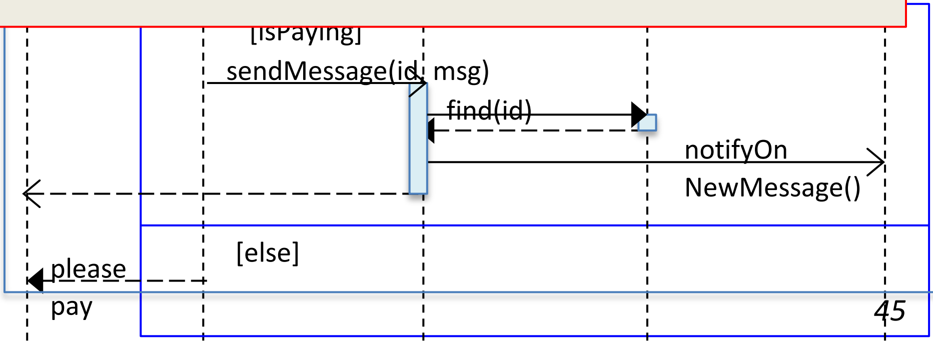


Now: External Interfaces!



Repeat for all main modules, focusing on interfaces <u>between</u> modules

Iterative process!



<u>Payment:</u>

- pay(String id, String ccInfo)

Agenda

- Top project ideas!
- Announcements
- **Architecture and Design**
  - What is a good module
  - How to define interfaces
  - REST interfaces (next week)
  - Some popular architectural patterns (next week)

See You Next Wednesday:

REST, Architectural Patterns, Microservices