

Design Abstractions

17-423/723 Software System Design

Lecture 3
Jan 21, 2026

Learning Goals

- Describe the role of abstraction in communicating designs
- Describe common types of design abstractions
- Select and apply a suitable notation to model an aspect of a design
- Generate questions to explore further design decisions given a model

Logistics

- Homework 1 is out on Canvas; due next Wednesday, Jan 28
- Practice on building a context model (last lecture) & design models (today)
- Submission through Gradescope
- Reminder: Project team formation survey due the end of **today**

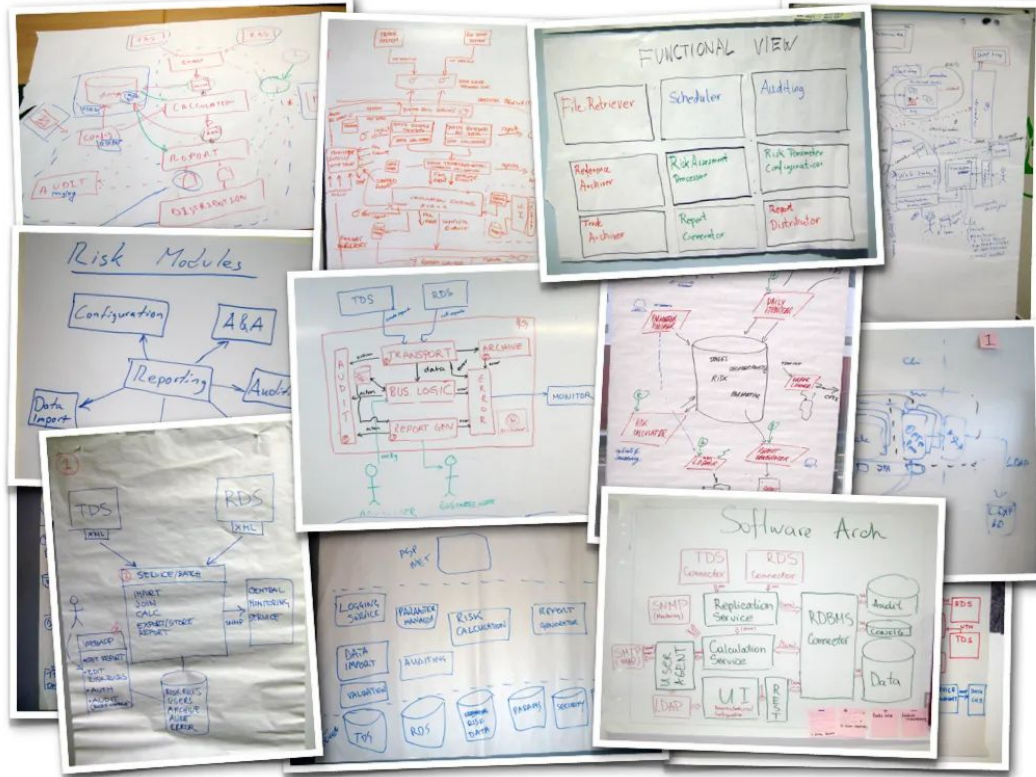
Documenting & Communicating Designs

Last class: Use of a context model to understand the **problem space** (domain entities & assumptions)

This class: Models for communicating design ideas in the **solution space**

How do software designers communicate ideas?

Designers use sketches

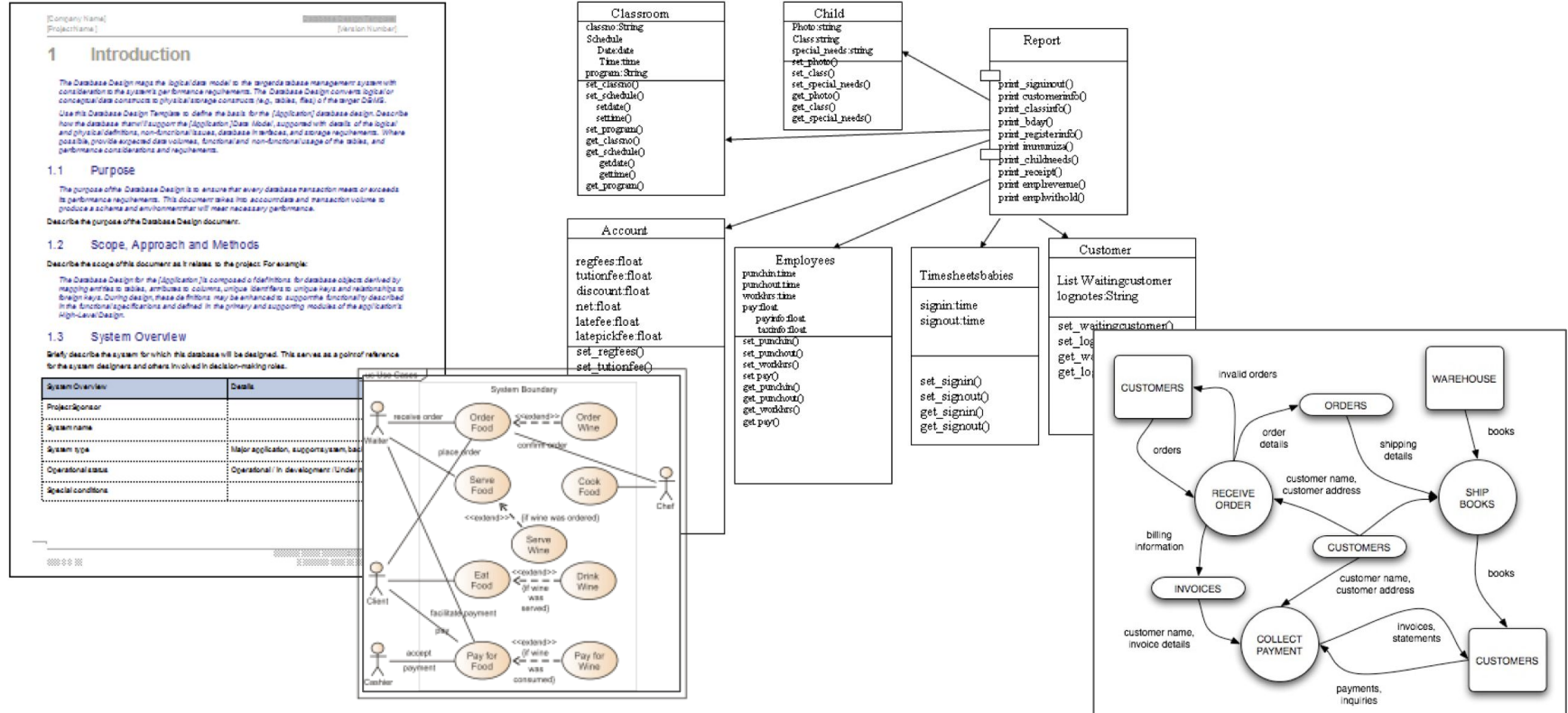


To brainstorm ideas

To explain how a design works

To ask questions about designs

Designers document their work



Viewpoint: Code as Design

Code is the final design and the only source of the truth

Source code listing is the only design documentation that we need

Q. What do you think? Arguments for & against?

Jack W. Reeves “Code as Design: Three Essays”

https://www.developerdotstar.com/mag/articles/reeves_design_main.html

Designers use abstractions

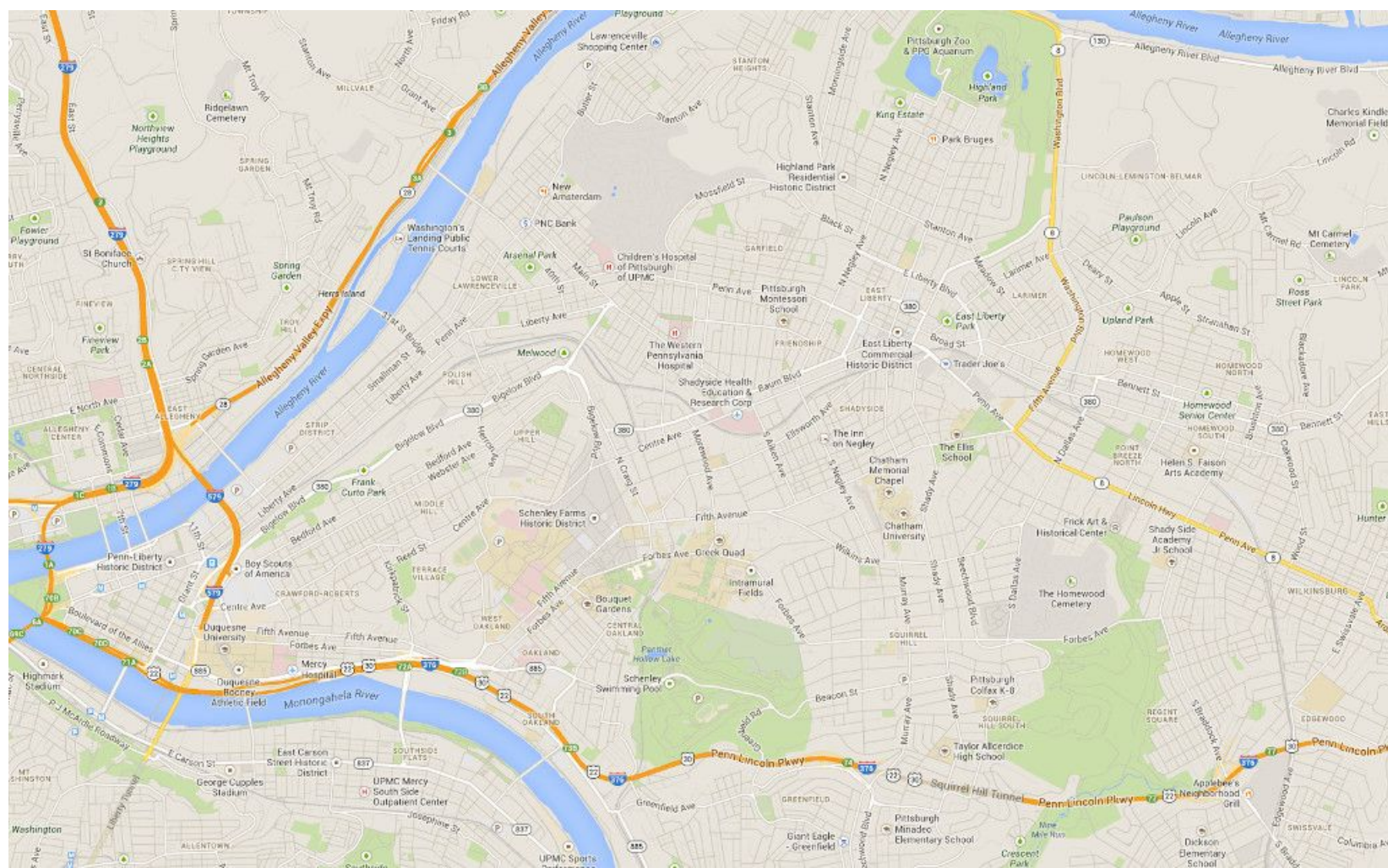
For most systems, code is too complex for a single person to understand

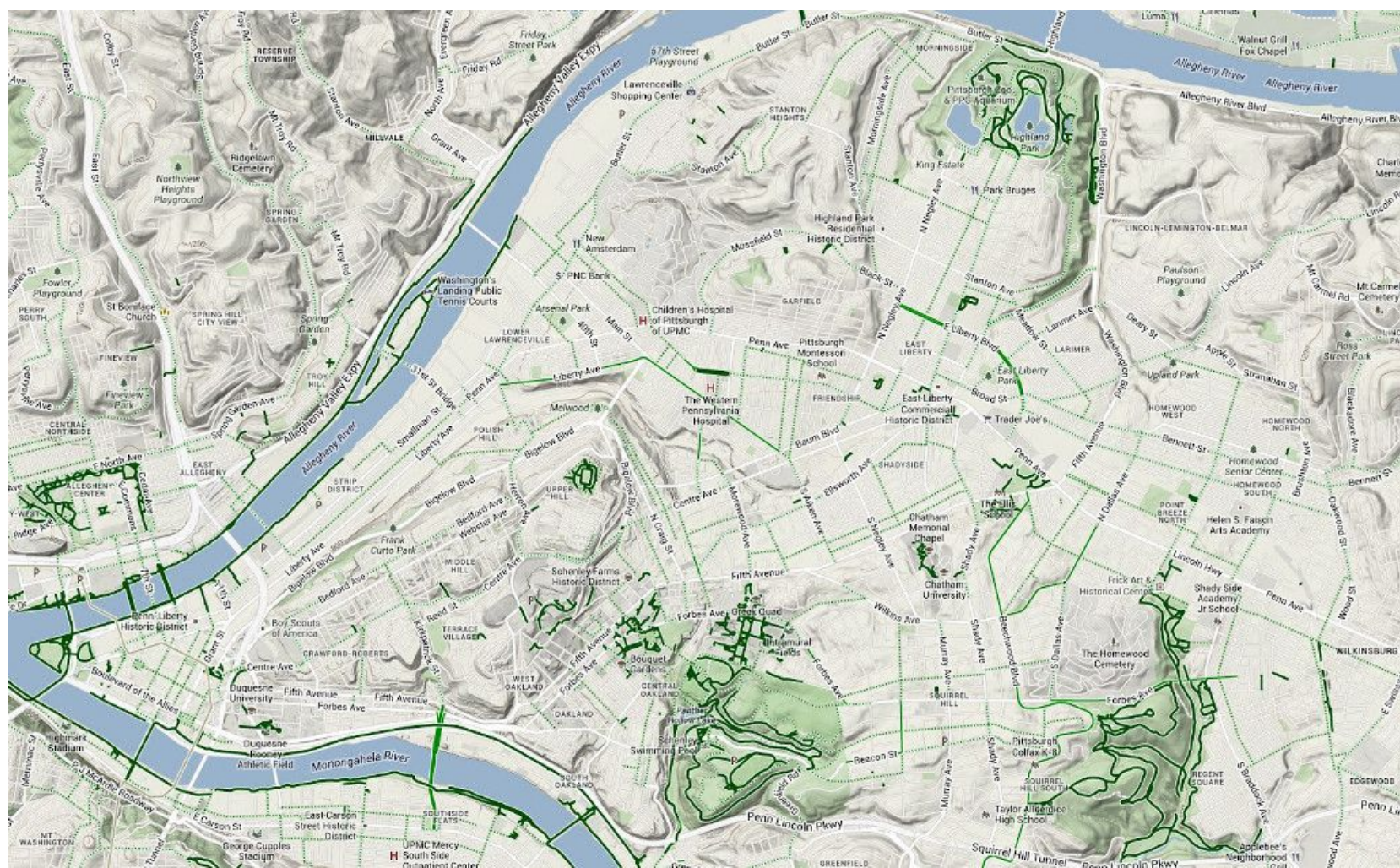
Code is not an ideal medium of communicating ideas

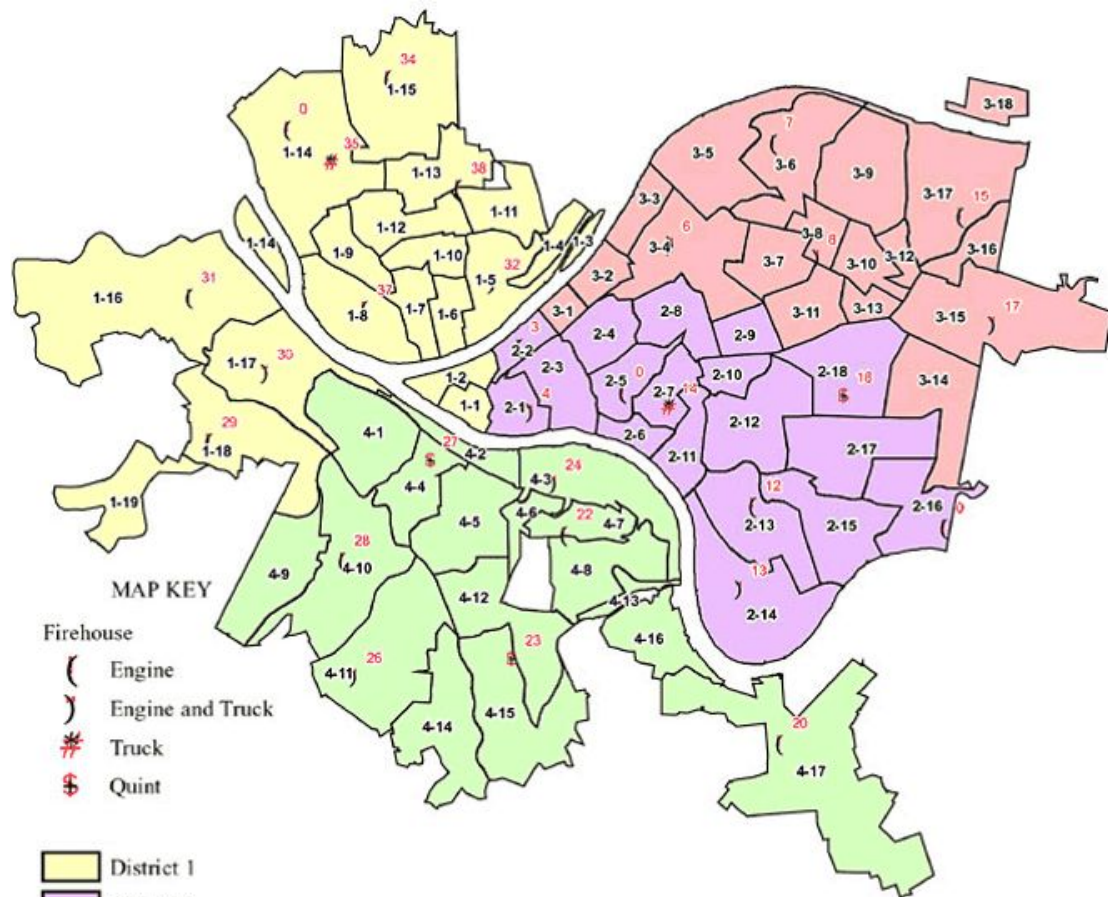
Code does not capture everything about design

- Q. What are some information that is not captured in the code?

Instead, designers communicate through **an abstraction of a system** – a description that focuses on a particular aspect & ignores other details







Purpose of an Abstraction

Abstractions aid in understanding

Each abstraction highlights particular aspects of a system and deliberately hides other details

Abstractions facilitate reasoning

Each abstraction answers certain types of questions about the system

Abstractions are reusable

Each abstraction captures a commonality across multiple systems (within a single domain or sometimes across multiple domains)

What can we reason about?

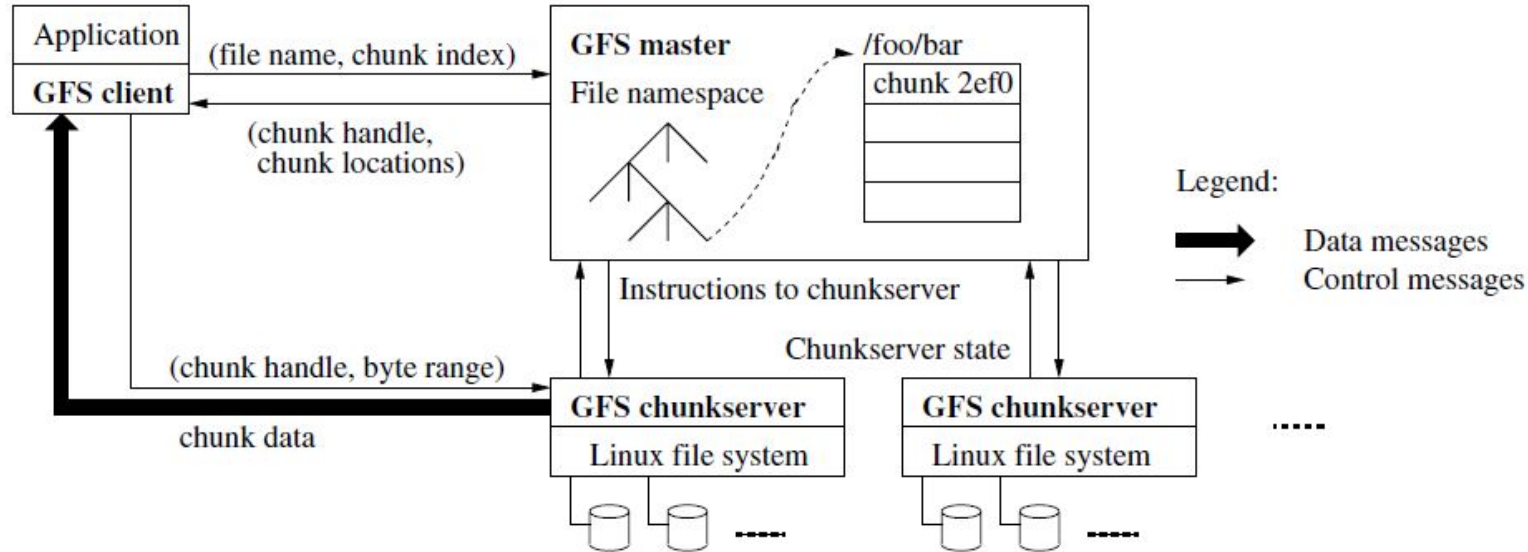
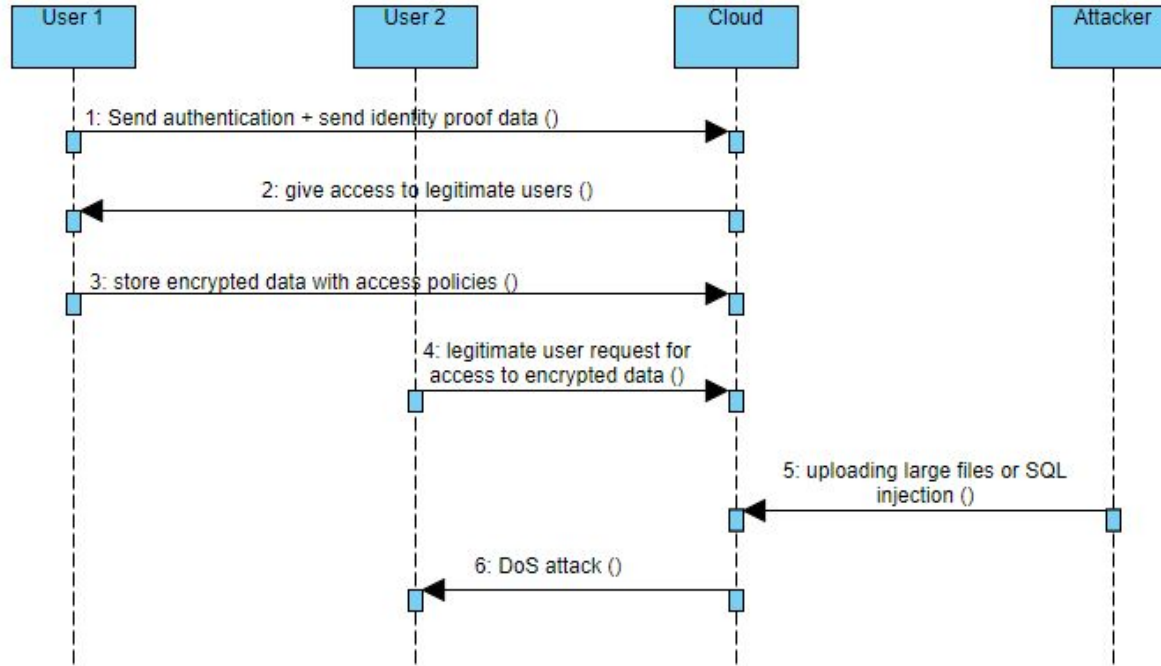


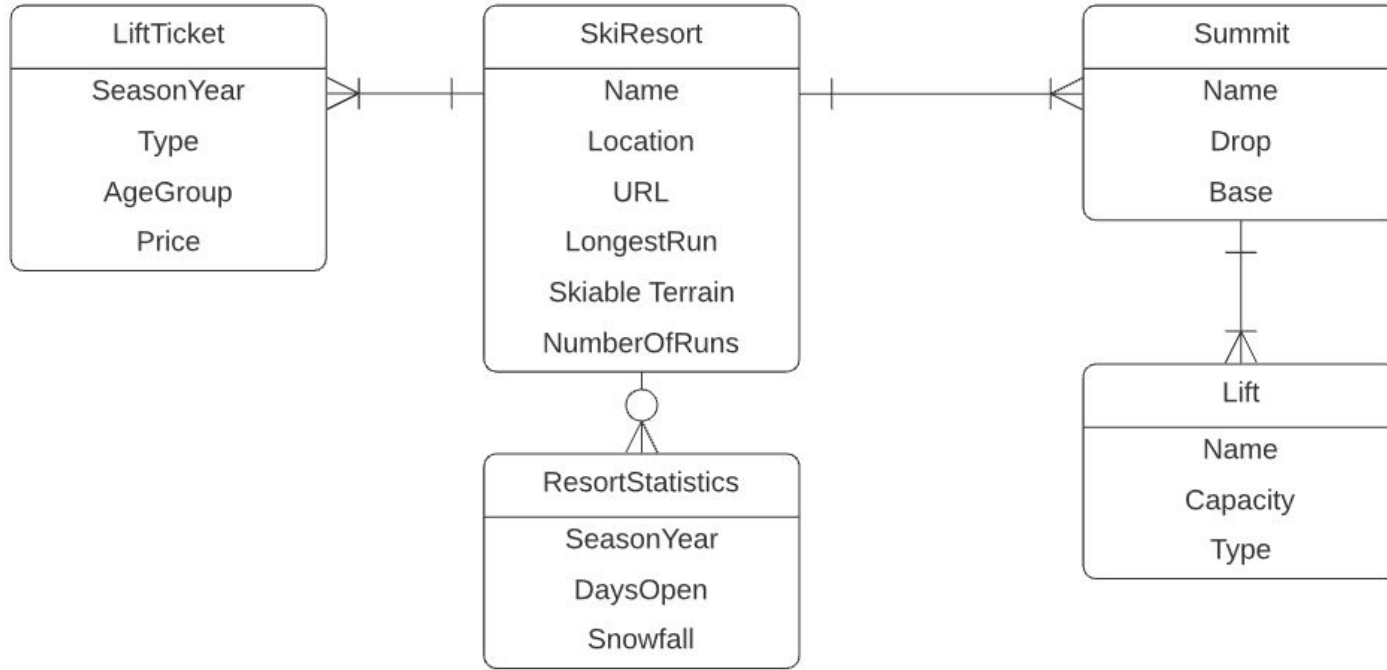
Figure 1: GFS Architecture

Ghemawat, Sanjay, Howard Gobioff, and Shun-Tak Leung. "The Google file system." ACM SIGOPS operating systems review. Vol. 37. No. 5. ACM, 2003.

What can we reason about?



What can we reason about?



Notations for Common Design Abstractions

Component diagrams: What are major components in the system and how do they communicate with each other?

State machines: What states can the system be in, and what events cause it to change its state?

Data models: What types of data does the system store and what are their relationships?

Sequence diagrams: How different actors (domain entities & system components) collaborate to carry out some functionality?

Modeling in this class

Sometimes, models are themselves developed as a formal, first-class artifact

- To rigorously specify & verify properties about the system
- Completeness is an important property of the model in this case

In this class, the goal is mainly to communicate

- To explain a design to someone who will work on the system later (sometimes, yourself)
- To articulate an aspect of the system that is particularly complex
- To guide an AI coding agent towards a desirable implementation
- Clarity, rather than completeness, is more important

Example: Electronic Voting System

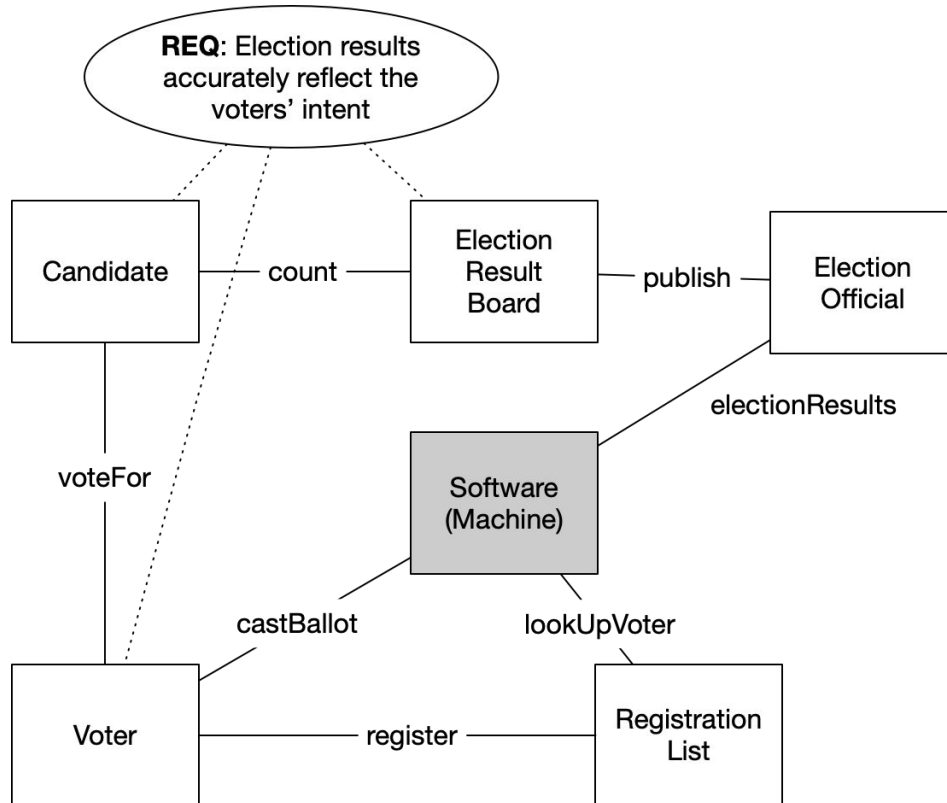


Voting System: Workflow

- A voter walks to an electronic voting machine and enters their voter ID information.
- The voting software checks whether the voter is registered to vote.
- The voter is presented with an electronic ballot with a list of candidates.
- The voter selects a candidate and casts their ballot by confirming their choice.
- The voting software collects all of the ballots and produces the final count for each candidate.
- The election officials retrieve and post the election results on a public board.

Q. What are some potential risks of failure with this system? What type of quality attributes should we design for?

Context Model for the Voting System



Component Diagram

Component Diagram

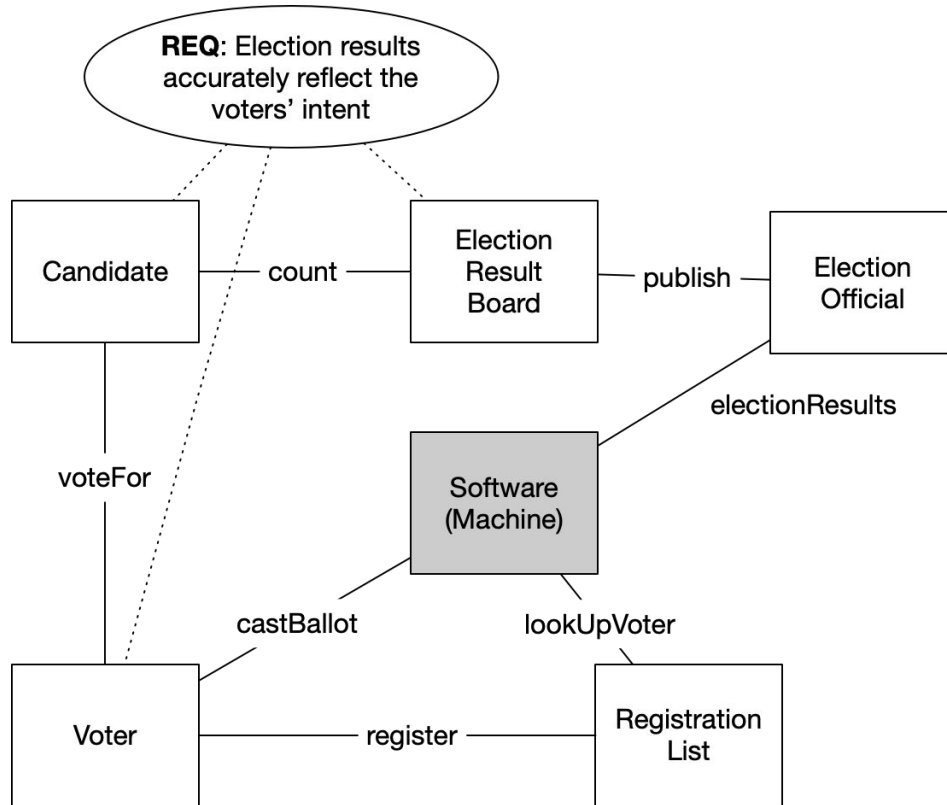
Purpose

Describe the major components in a software system, which components communicate with each other, and how they communicate.

Building blocks

- **Component:** A software component, responsible for carrying out a distinct unit of functionality
- **Connection:** A **directed** connection between a pair of components, labeled with an event (e.g., an API call) or data flow
- **Component responsibilities** (in text annotations): Describe the responsibilities of each component.

Designing the Voting Software: Decisions to Make



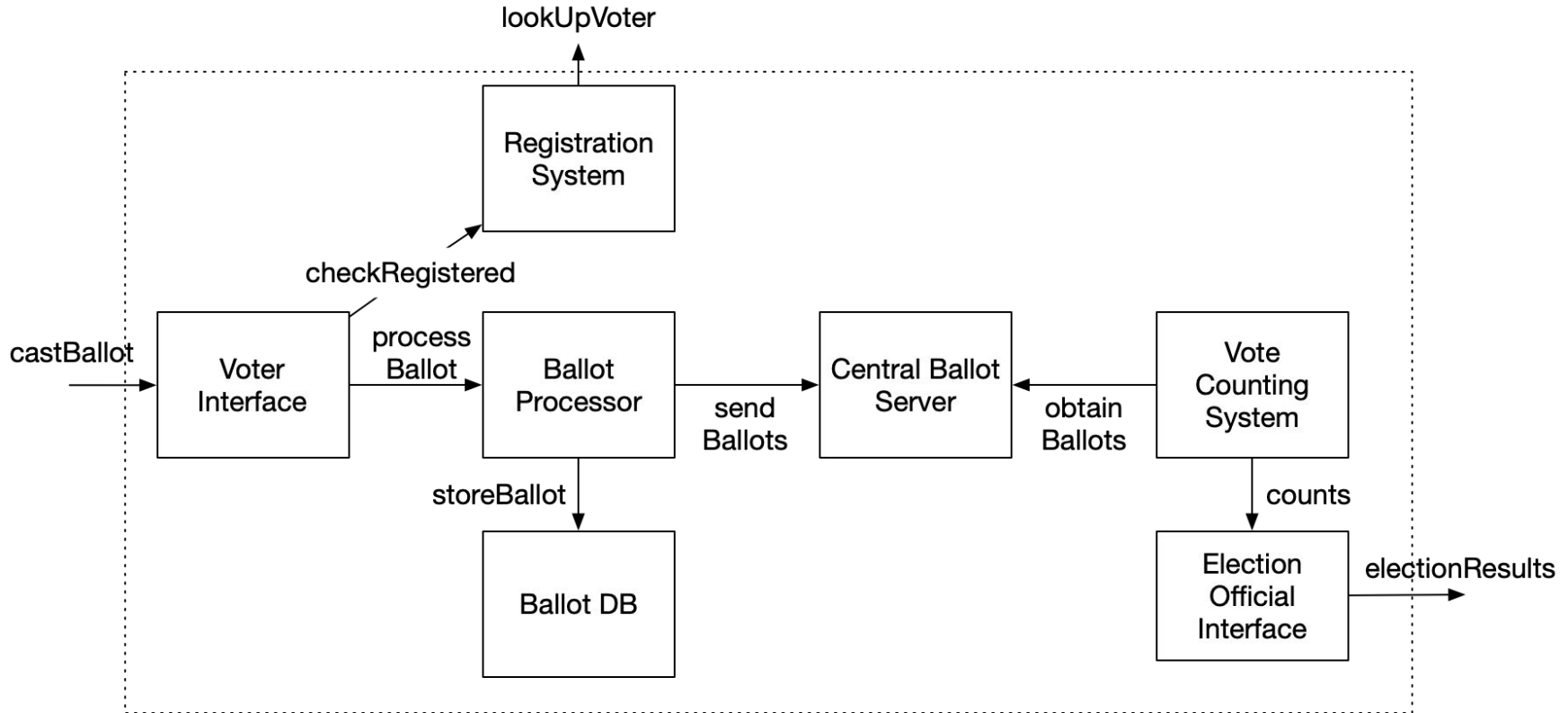
What is the specification of the voting software?

What components does the voting software consist of?

How do the components interact with the domain entities?

How do these components interact with each other?

Component Diagram: Example



Component Responsibilities

Voter Interface: Receives inputs from the voter & forwards ballot information to the Ballot Processor.

Registration System: Checks whether a voter is registered by looking up an external registration list.

Ballot Processor: Processes & stores each ballot on a local database. Uploads all of the ballots to the central server.

BallotDB: Stores the ballots cast so far for a local precinct.

Central Ballot Server: Stores all of the ballots across multiple precincts.

Vote Counting System: Compute the final tally of votes for each candidate.

Election Official Interface: Serves requests from an election official to retrieve election results.

Component Diagram: Questions to Ask

Each model is a starting point for asking & discussing further design questions

Different types of models encourage different types of questions

Component Diagram: Questions to Ask

What information is passed between one component to another? Is there a return value?

Which components interact with the entities in the problem space (e.g., users)?

Where is each component deployed? Which components are deployed on the same device?

What type of communication mechanism is used for each connection? (e.g., HTTP/S, RPC, Bluetooth) Is the communication synchronous or asynchronous?

What if a component changes or fails? What other components does it affect?

What if a connection fails? How does it affect the overall system?

State Machine

State Machine

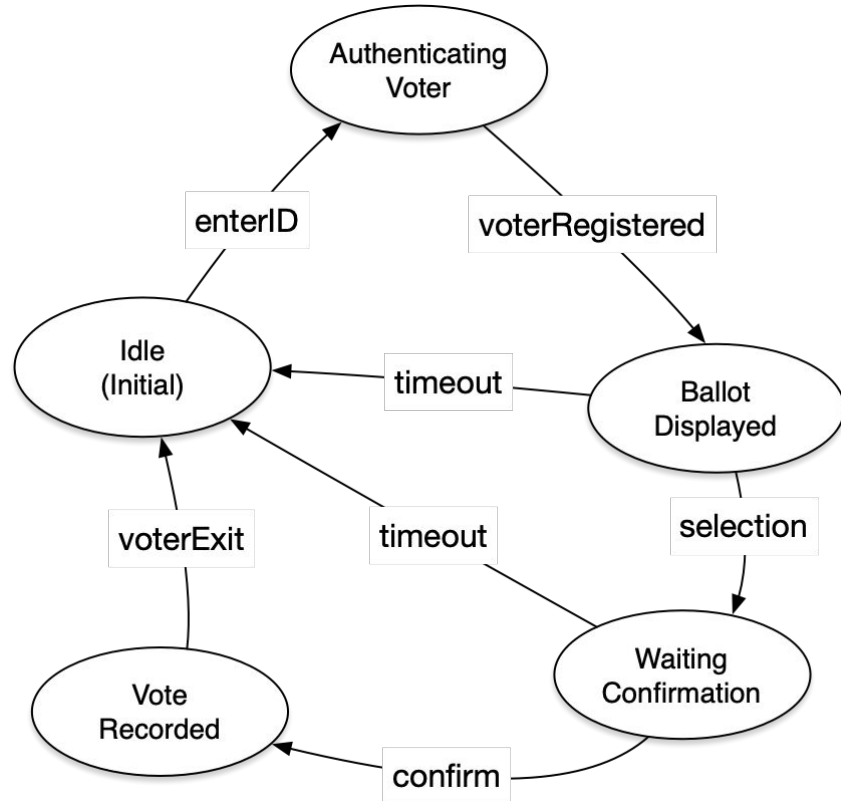
Purpose

- Describe possible states of a system (or its component) and which event triggers its state to change from one to another
- Often used to illustrate complex business logic or protocols

Building blocks

- **State:** A state of a system or a component
 - **Initial state:** A special state where the system/component begins its execution
- **Transition:** A change from one state to another based on an event

State Machine: Voter Interface



State Machine: Questions to Ask

What are possible error states? How can the machine reach those states?

Given a state, which events can occur? Does the state have a transition for each such event?

Can the machine get stuck in a state due to an event that never takes place?

Can it get into a cycle, and is that acceptable?

Is the machine deterministic; i.e., given a state and an event, is there at most one outgoing transition?

Data Model

Examples by Daniel Jackson

Data Model

Purpose

- Describe different types of data that the system needs to remember to fulfill its specification
- Serve as the conceptual schema for designing a database

Building blocks

- **Data type:** A collection of data elements or objects
- **Relation:** A **directed** relation between a pair of data types
- **Multiplicity constraints:** A constraint on a relation, specifying how many instances of the two data types can be related to each other

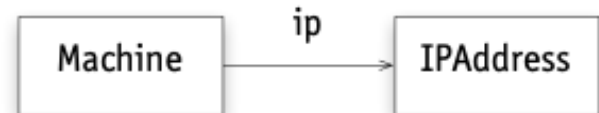
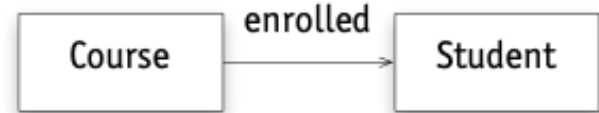
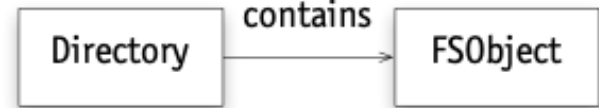
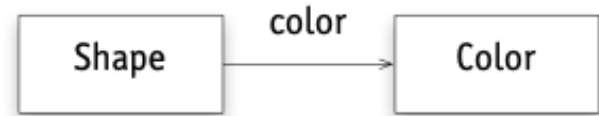
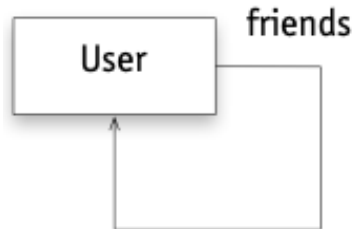
Data Model: Relations

A relationship between different data types

Kinds of relations:

- Property
- Containment
- Association
- Naming

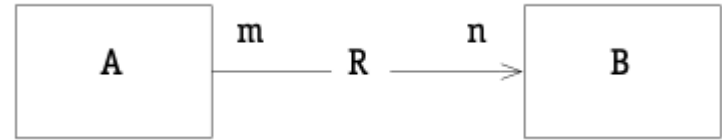
Q. What kinds of relations are these?



Data Model: Multiplicity Constraints

Constraints on relations

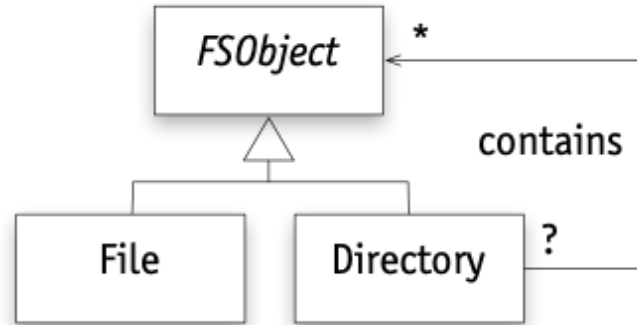
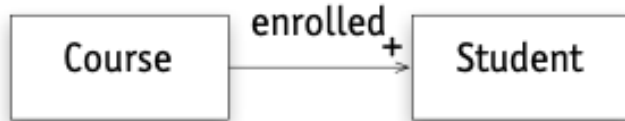
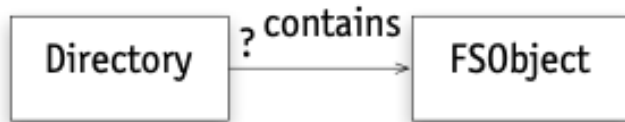
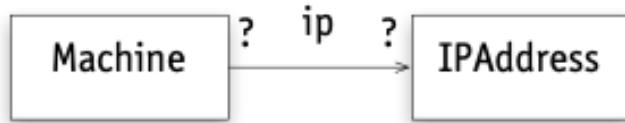
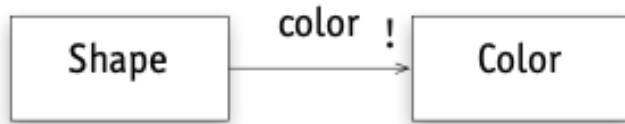
- R is a relation of type A to B
- R maps **m** A's to each B
- R maps each A to **n** B's



These constraints will eventually need to be enforced in the database design

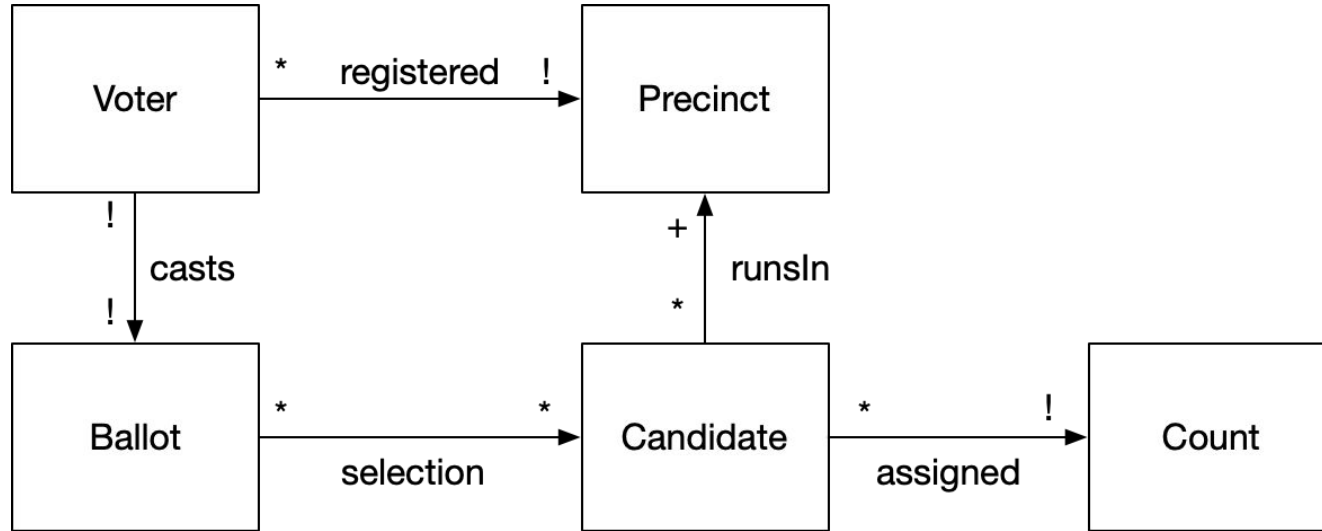
+ one or more
* zero or more
! exactly one
? at most one
omitted = *

Data Model: Multiplicity Examples



+ one or more
* zero or more
! exactly one
? at most one
omitted = *

Data Model: Voting System



Data Model: Questions to Ask

Are we capturing all information that the system needs to function and achieve its quality attributes (e.g., security, availability)?

Do the multiplicity constraints reflect the real world scenarios? Are there any missing constraints? Are some constraints too strong?

Will all relations be stored on the same database, or be distributed across multiple databases? If distributed, do we need to consider consistency issues?

Is some of the data potentially sensitive? Do we need additional security or privacy mechanisms?

Sequence Diagram

Sequence Diagram

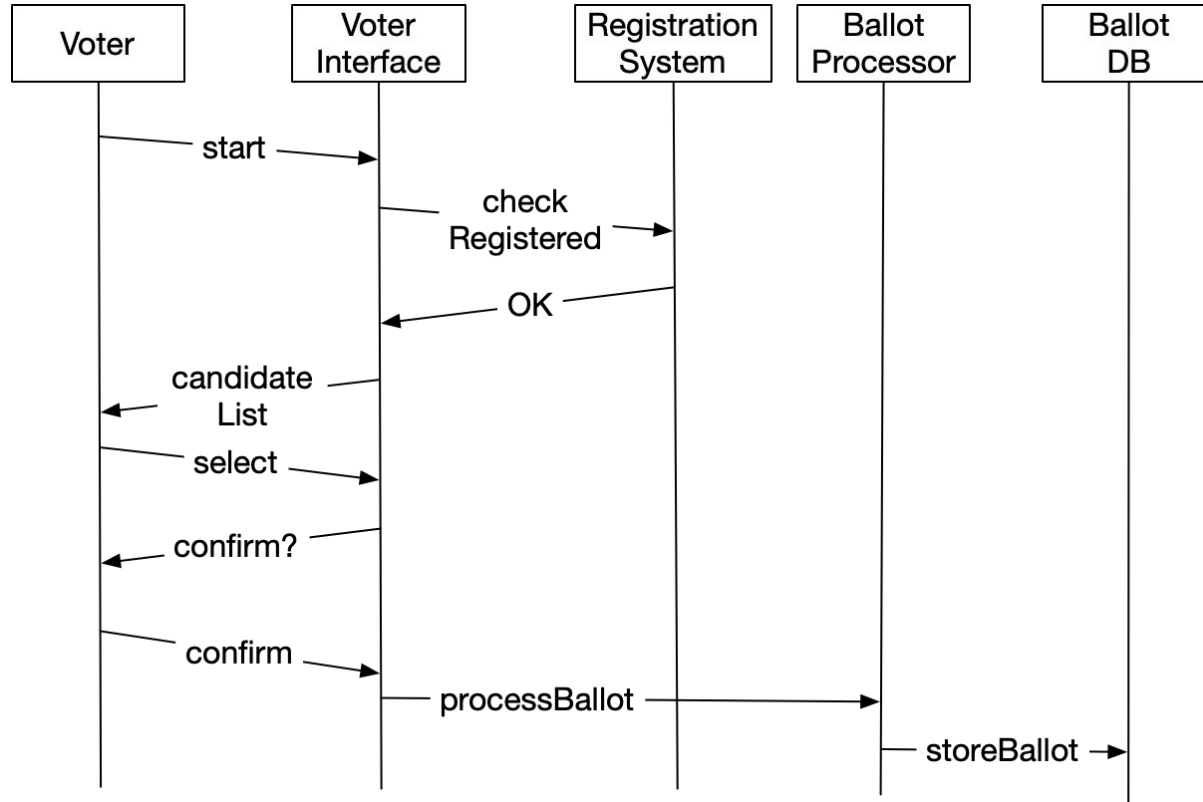
Purpose

Describe how a set of domain entities and system components collaborate in sequence to achieve a piece of functionality

Building blocks

- **Process:** A domain entity or system component
- **Message:** A message passed from one process to another

Sequence Diagram: Example



Sequence Diagram: Questions to Ask

What happens if a process terminates its activity early?

Is it possible for a message to be lost, and how does the system handle this?

What if it arrives late?

What if a process receives two messages out of order? Does the order of execution matter for functionality or a quality attribute?

Are there any other processes that we are missing in this scenario?

Voter Fraud



FRANKFORT — A former Clay County precinct worker testified Friday that top election officers in the county taught her how to change people's choices on voting machines to steal votes in the May 2006 primary.

Voters walk away from the machine before pressing “confirm”
Election officials enter booth, press “back” & modify the vote

Tips for Building Diagrams

Aim for simplicity & clarity!

Keep diagrams to a reasonable size. If a diagram gets too big, break it into multiple ones.

Annotate a diagram with text to explain a concept (e.g., the responsibility of a component) if its meaning is not obvious from its label

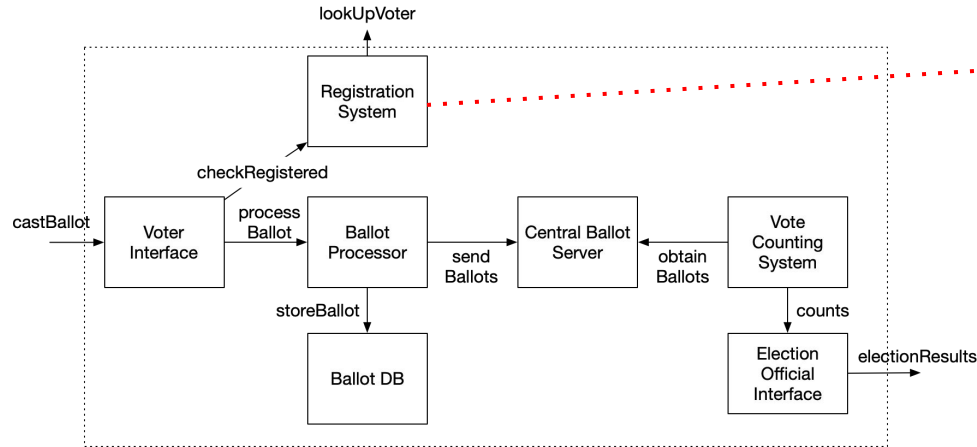
Use intuitive names! Avoid meaningless names (e.g., component named “Service”).

Consistency between Diagrams

Try to use consistent names for the same concepts (components, events, etc.,) across multiple diagrams

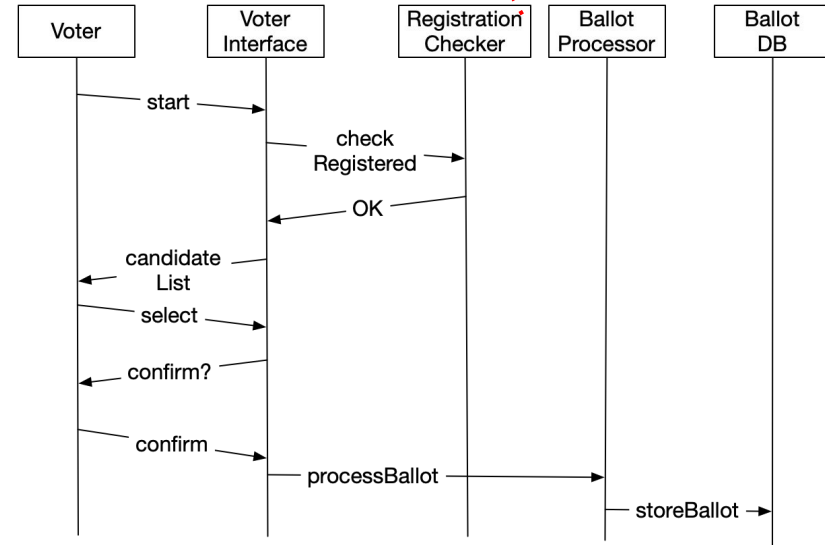
A concept in one diagram may be represented by multiple concepts in another diagram; if so, explain how they are related

Consistency between Diagrams



Are these two the same?

Voter input “castBallot” is implemented by Voter Interface as a sequence of events “start”, “select”, and “confirm”



Tips for Building Diagrams

We will ask you document your design using some of these models throughout project milestones

Use these models in your team discussions! We will ask for your reflections on how they helped (or not) with designing your system

Treat these models as a tool for brainstorming & communicating, not for documenting everything perfectly

- Focus on aspects that are most important for your system (recall “risk-driven” approach to design)
- Use models to ask further questions about the system!

Summary

Exit ticket!