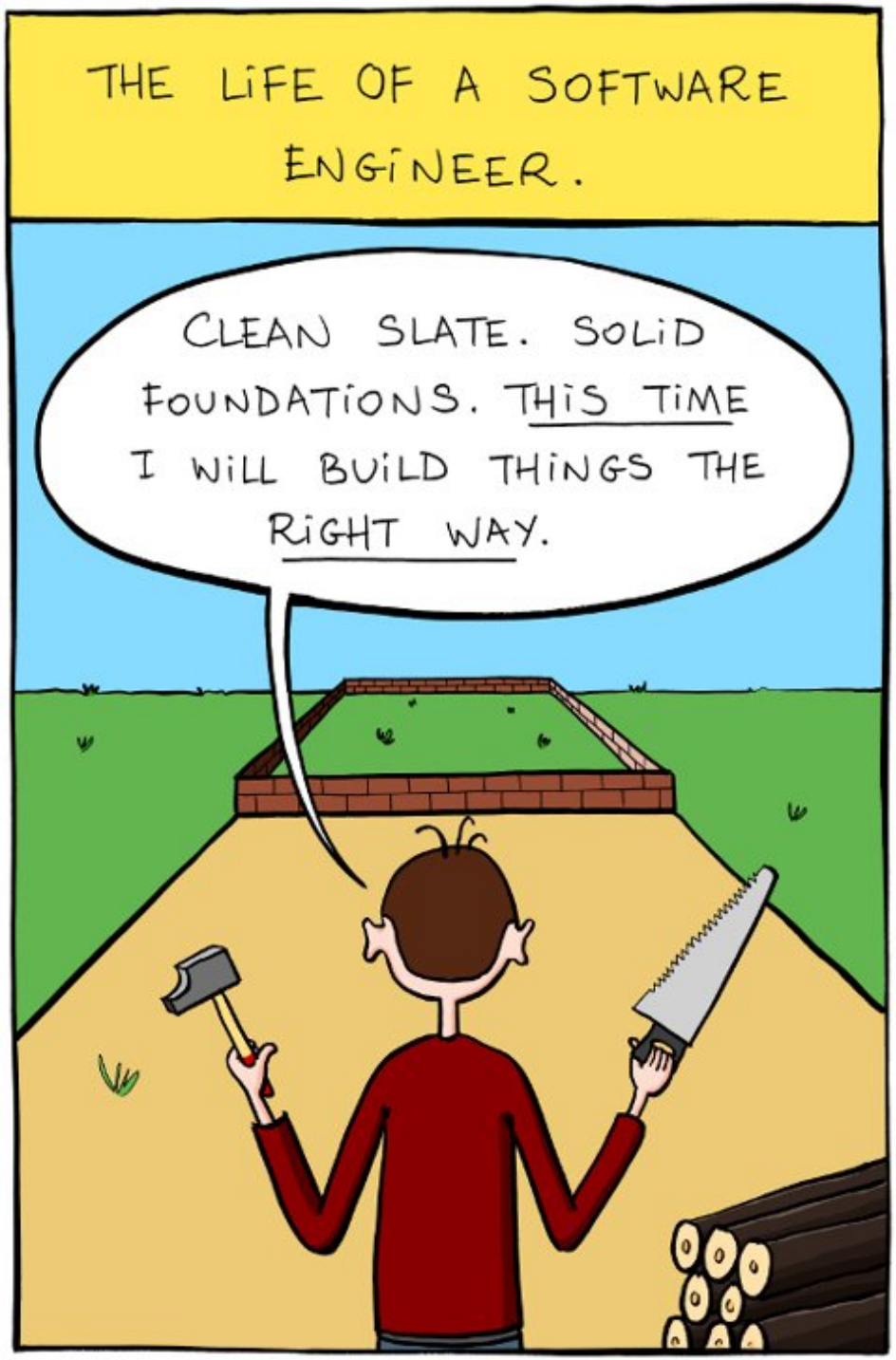
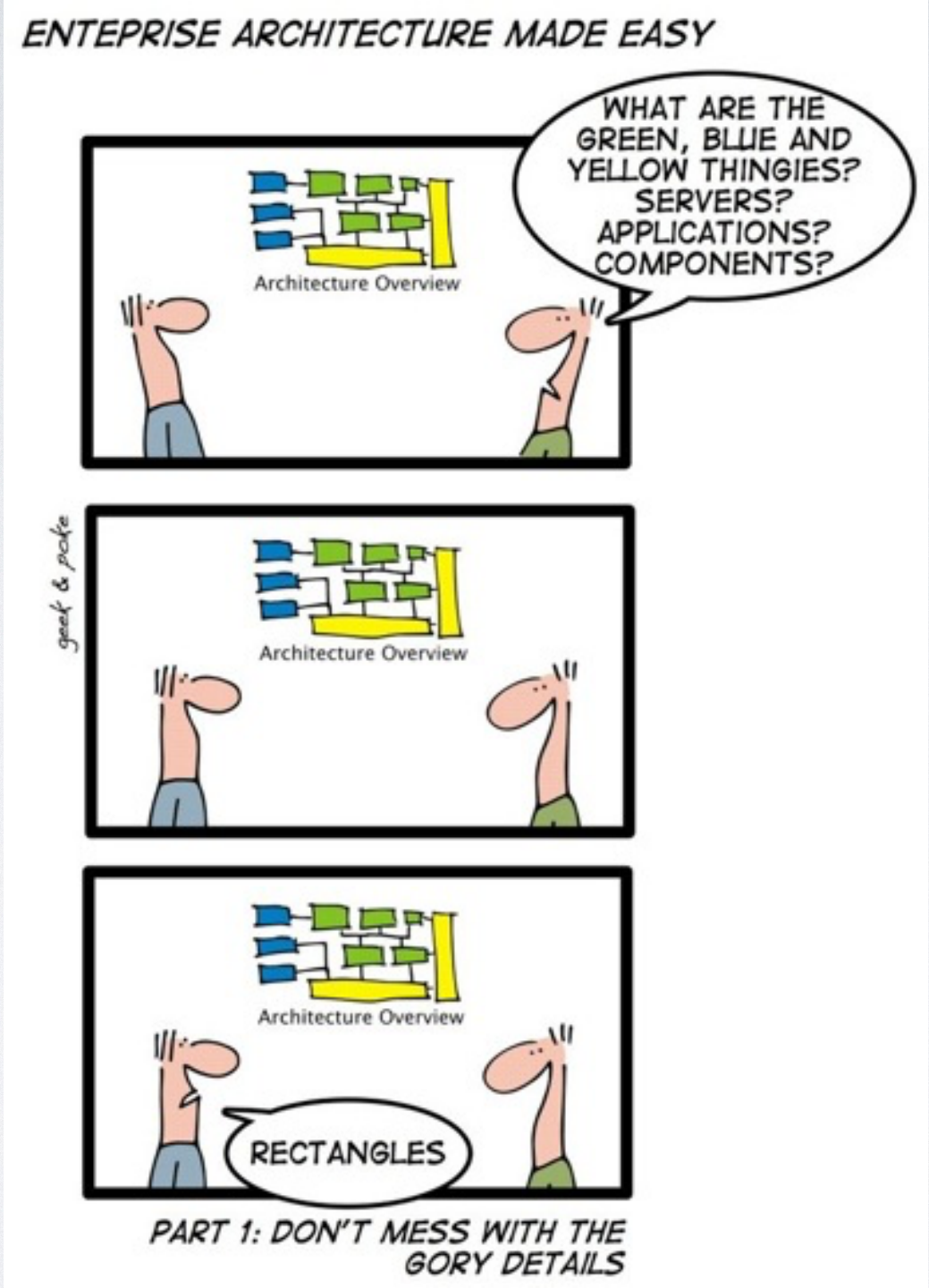

THE LIFE OF A SOFTWARE ENGINEER.
CLEAN SLATE. SOLID FOUNDATIONS. THIS TIME I WILL BUILD THINGS THE RIGHT WAY.


MUCH LATER...
OH MY. I'VE DONE IT AGAIN, HAVEN'T I ?
MANU


ENTEPRISE ARCHITECTURE MADE EASY
WHAT ARE THE GREEN, BLUE AND YELLOW THINGIES? SERVERS? APPLICATIONS? COMPONENTS?
Architecture Overview
geek & poke
Architecture Overview
Architecture Overview
RECTANGLES
PART 1: DON'T MESS WITH THE GORY DETAILS

Introduction to Software Architecture

Michael Coblenz

Slide credit: Michael Hilton at CMU

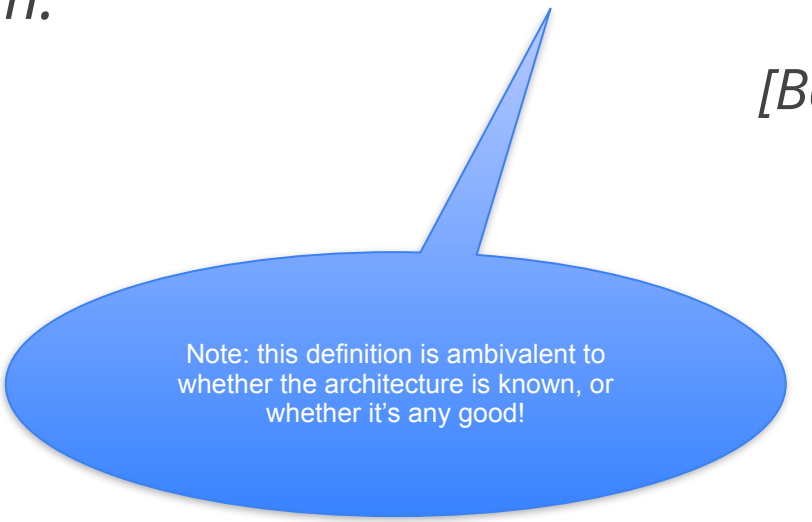
Learning Goals

- Understand the abstraction level of architectural reasoning
- Appreciate how software systems can be viewed at different abstraction levels
- Distinguish software architecture from (object-oriented) software design
- Use notation and views to describe the architecture suitable to the purpose
- Document architectures clearly, without ambiguity

Software Architecture

The software architecture of a program or computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them.

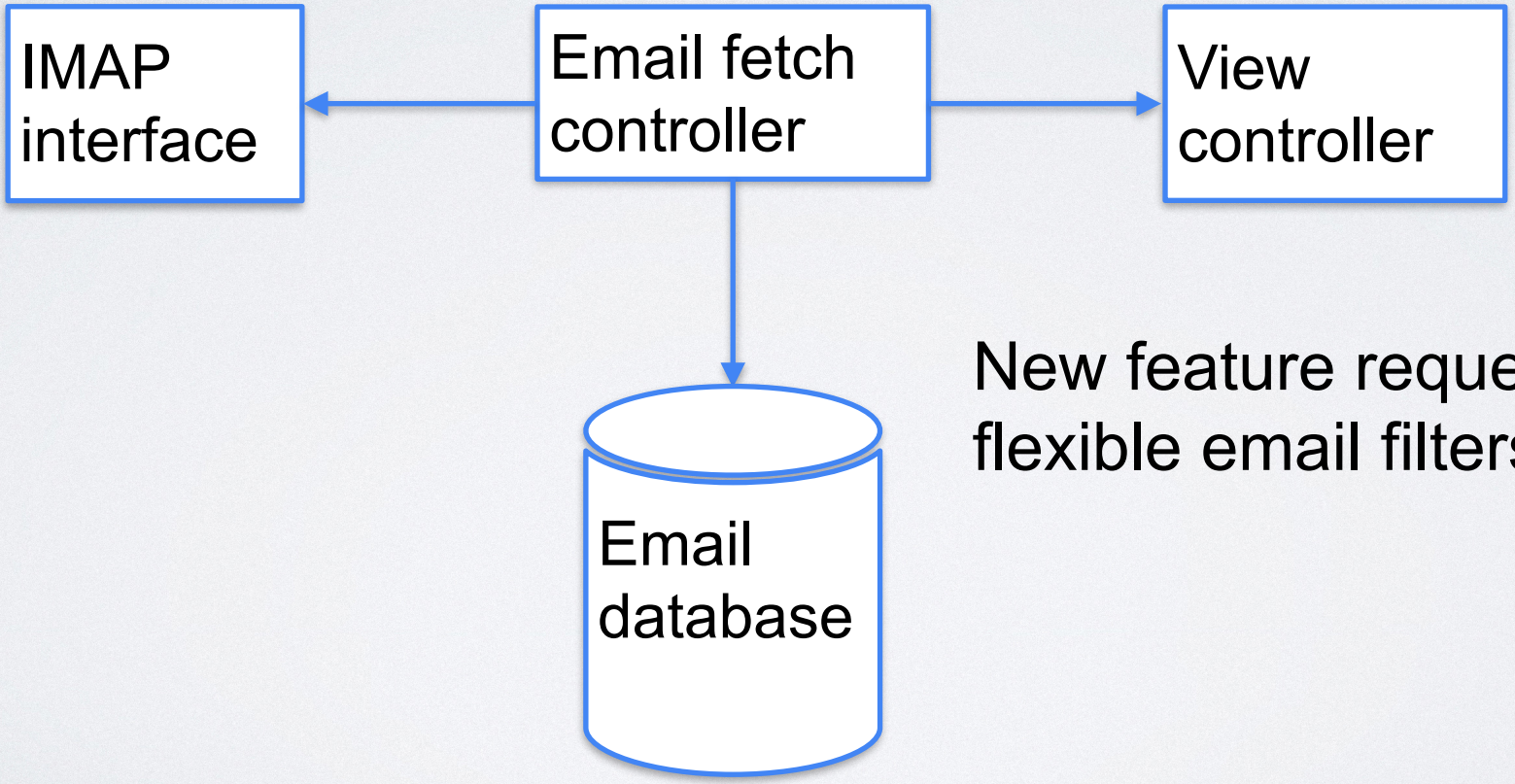
[Bass et al. 2003]



Why Understand Architecture?

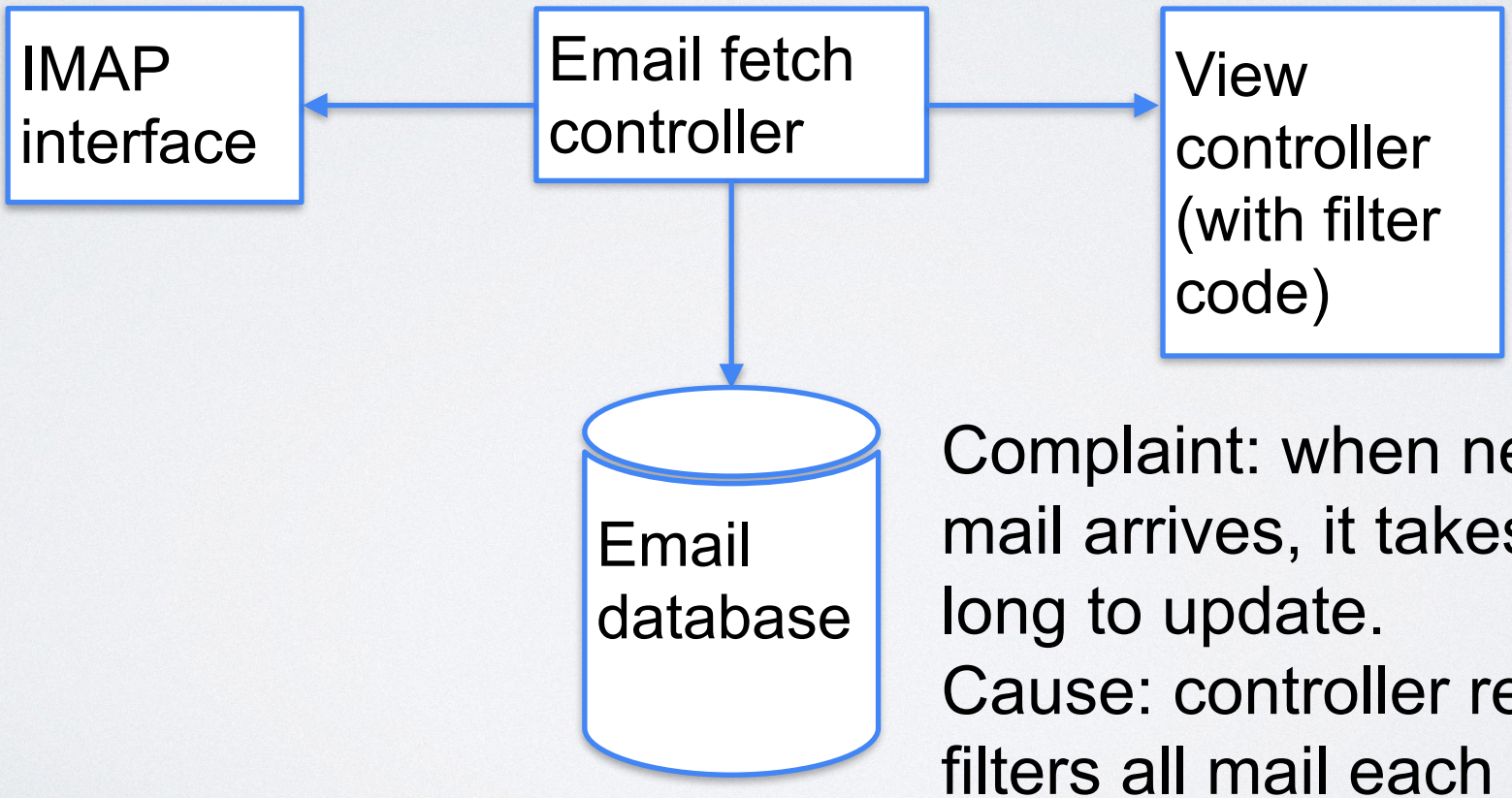
- Every system has an architecture
- But if you design the architecture intentionally, it's likely to be better!
- Let's look at an example

Example: Email Client



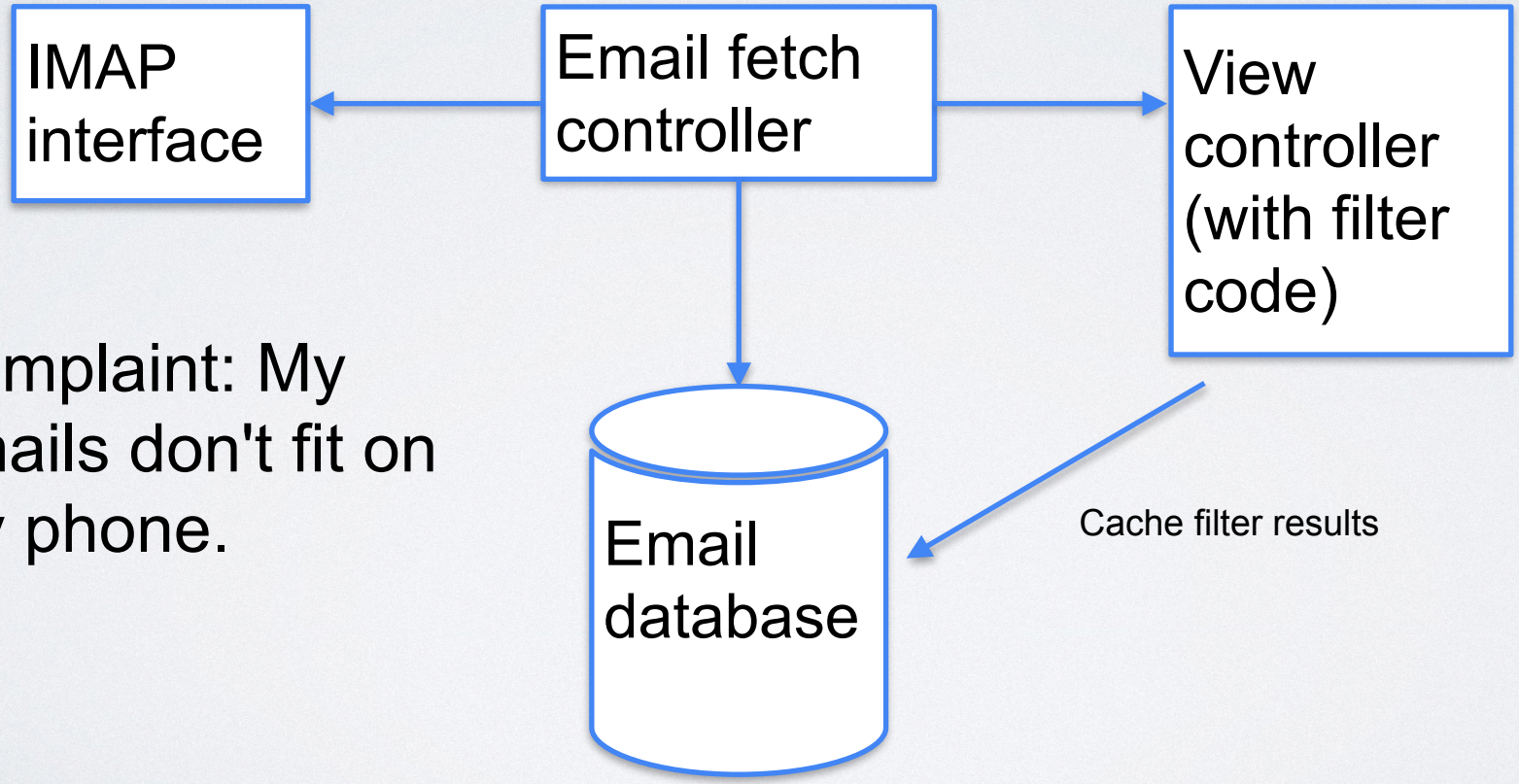
New feature request: flexible email filters.

Example: Email Client



Complaint: when new mail arrives, it takes too long to update.
Cause: controller re-filters all mail each time.

Example: Email Client



Complaint: My emails don't fit on my phone.

Two Kinds of Requirements

- Functional requirements: what the system should do
    - "The system shall enable the user to read email."
    - Generally, these are either met or not met (if not met, the system is unacceptable)
- Quality attributes: the degree to which the software works as needed
    - "The system shall fetch 1 GB of email in under 1 minute."
    - Sometimes called "non-functional requirements"
    - Maintainability, modifiability, performance, reliability, security
    - Generally, these can be achieved in degrees

Goal: Meet Quality Requirements

- Maintainability / Modifiability
- Performance
- Scalability
- Availability
- Usability

Key lesson: software architecture is about selecting a design that meets the desired quality attributes.

Another Perspective

- Quality requirements help designers choose from among many different designs that all meet the functional requirements.

Software Design vs. Architecture

Levels of Abstraction

- Requirements
  - high-level “what” needs to be done
- Architecture (High-level design)
  - high-level “how”, mid-level “what”
- OO-Design (Low-level design, e.g. design patterns)
  - mid-level “how”, low-level “what”
- Code
  - low-level “how”

Design vs. Architecture

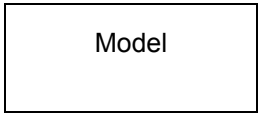
Design Questions

- How do I add a menu item in VSCode?
- How can I make it easy to add menu items in VSCode?
- What lock protects this data?
- How does Google rank pages?
- What encoder should I use for secure communication?
- What is the interface between objects?

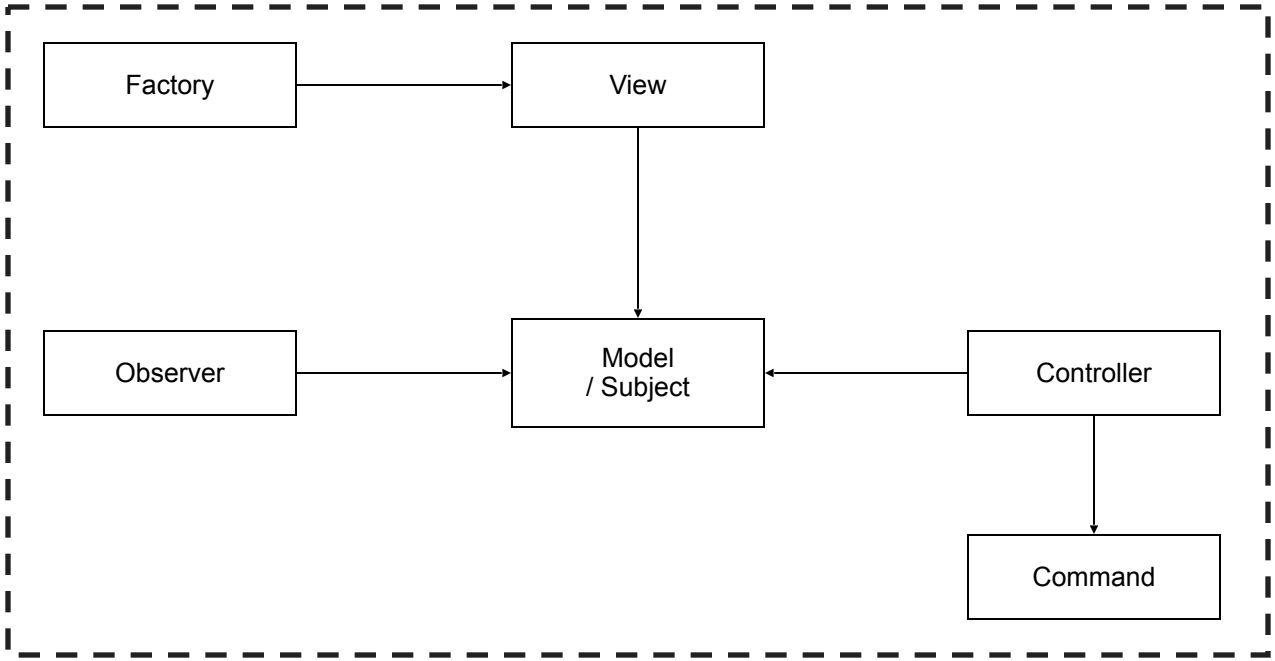
Architectural Questions

- How do I extend VSCode with a plugin?
- What threads exist and how do they coordinate?
- How does Google scale to billions of hits per day?
- Where should I put my firewalls?
- What is the interface between subsystems?

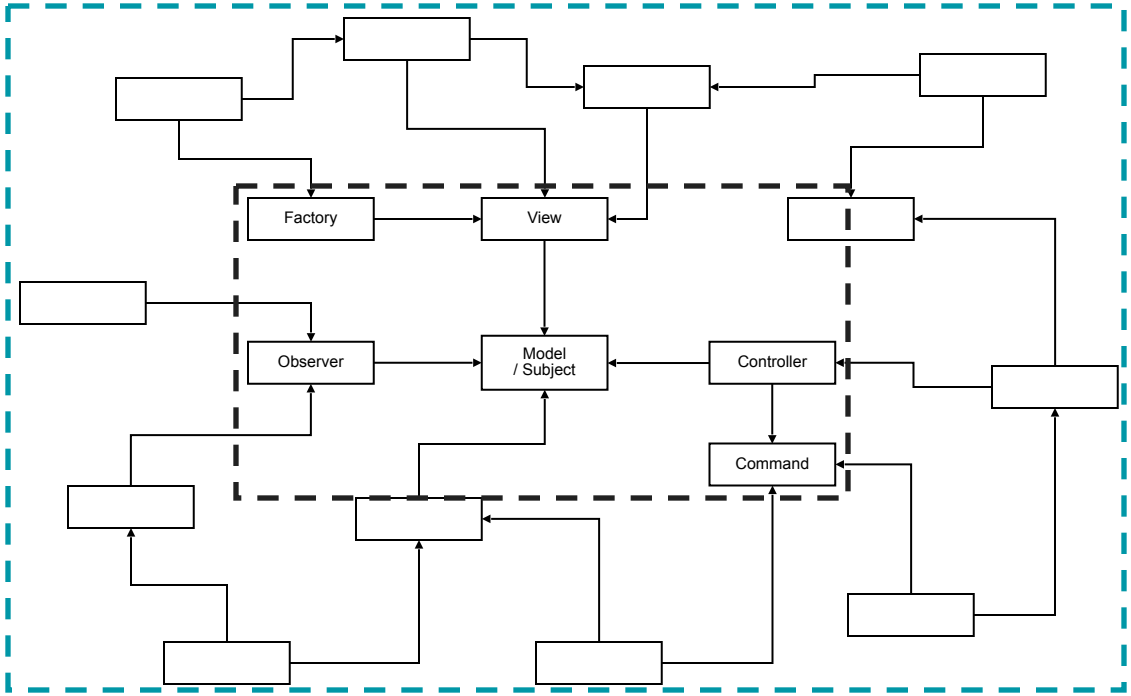
Objects



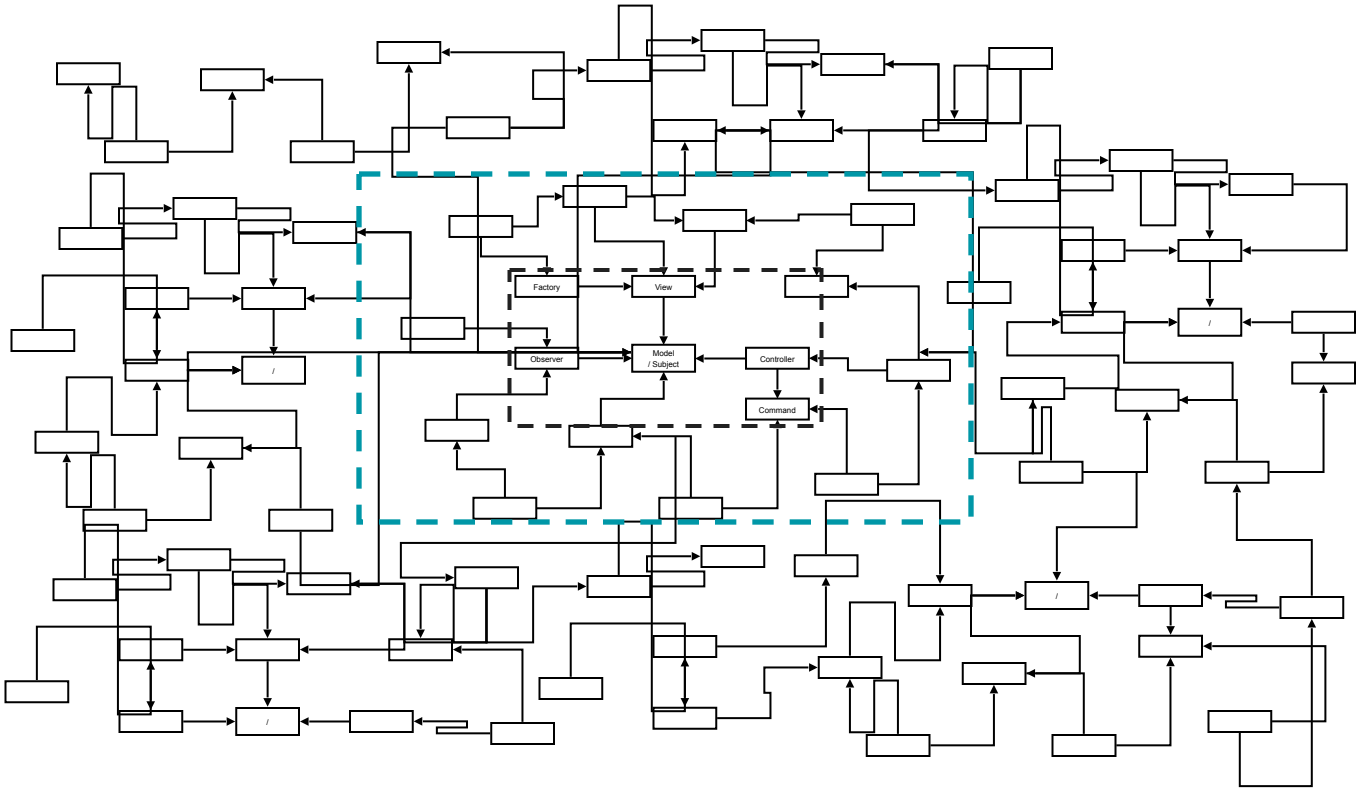
Design Patterns



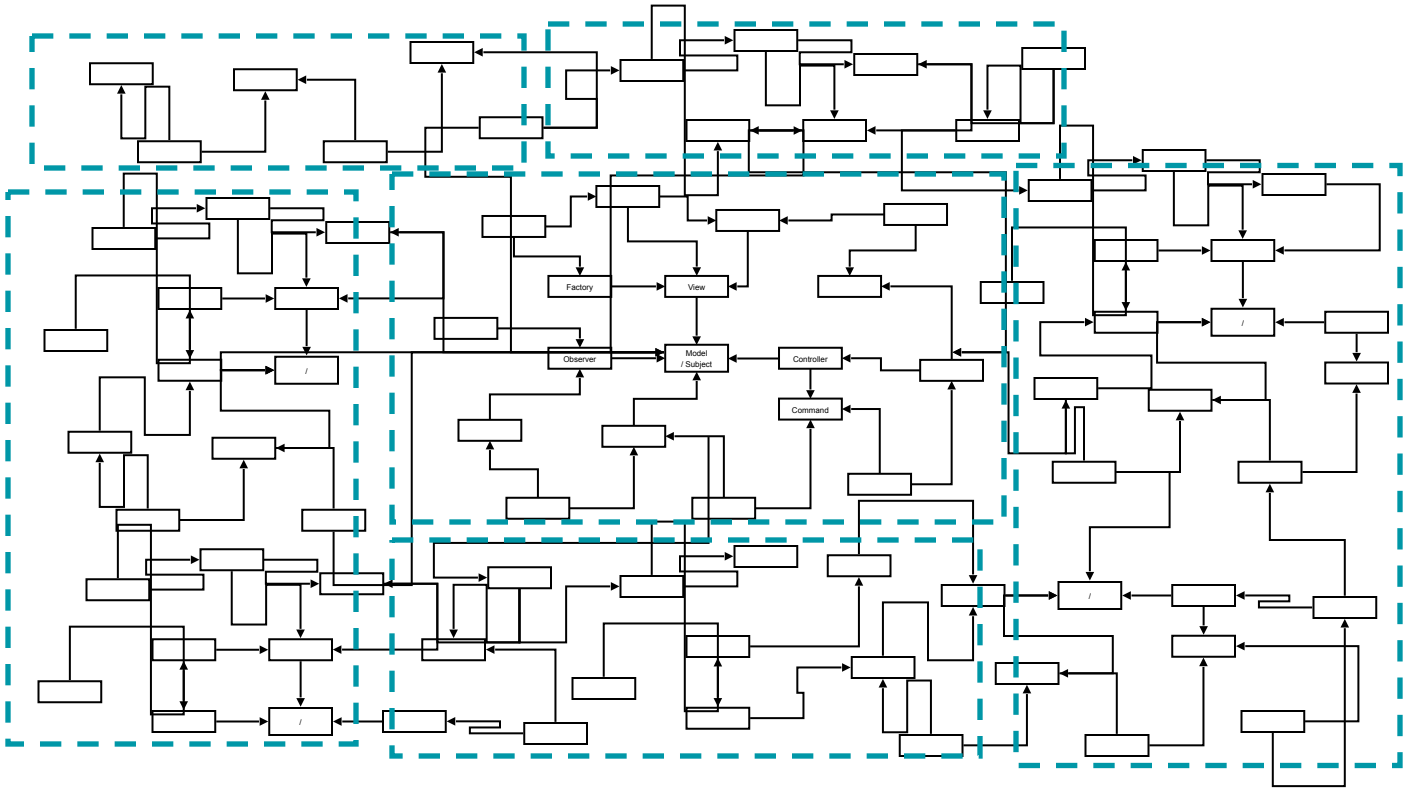
Design Patterns



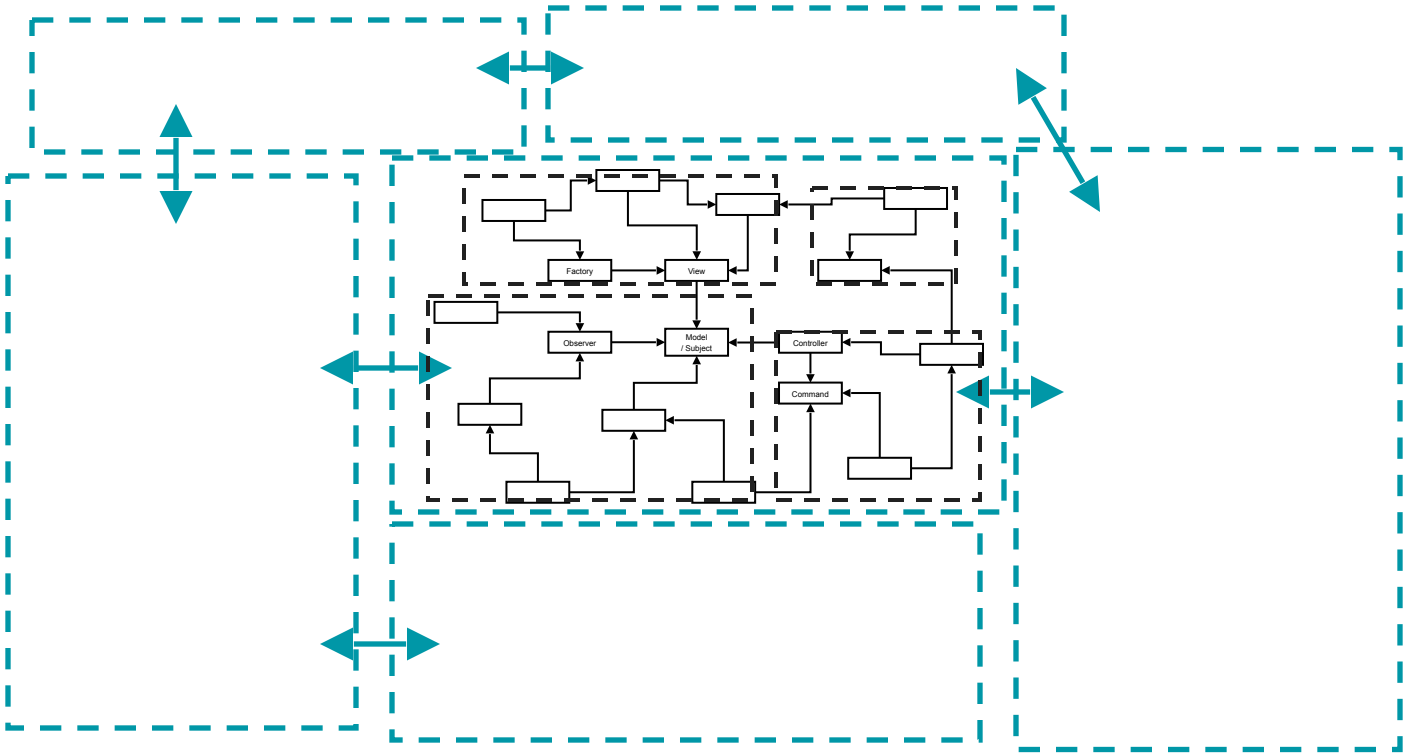
Design Patterns



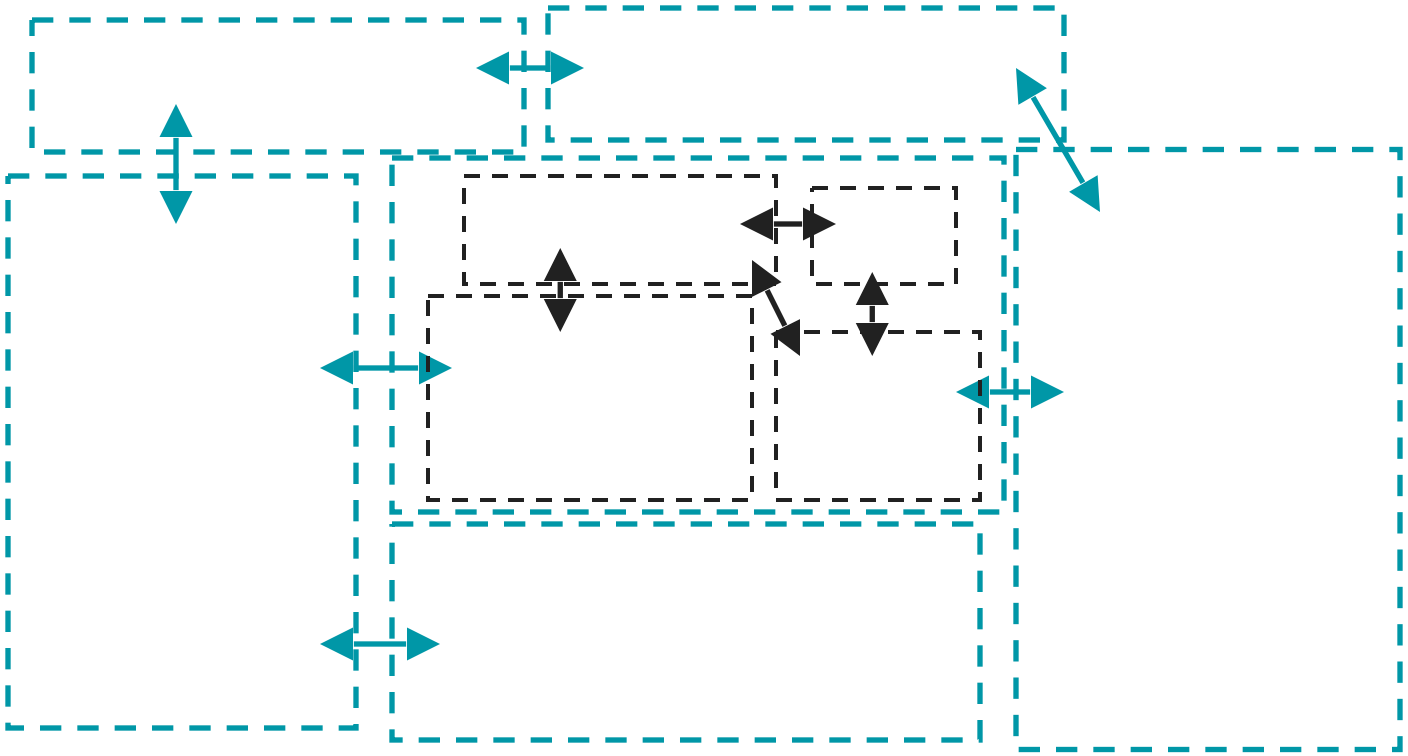
Architecture



Architecture



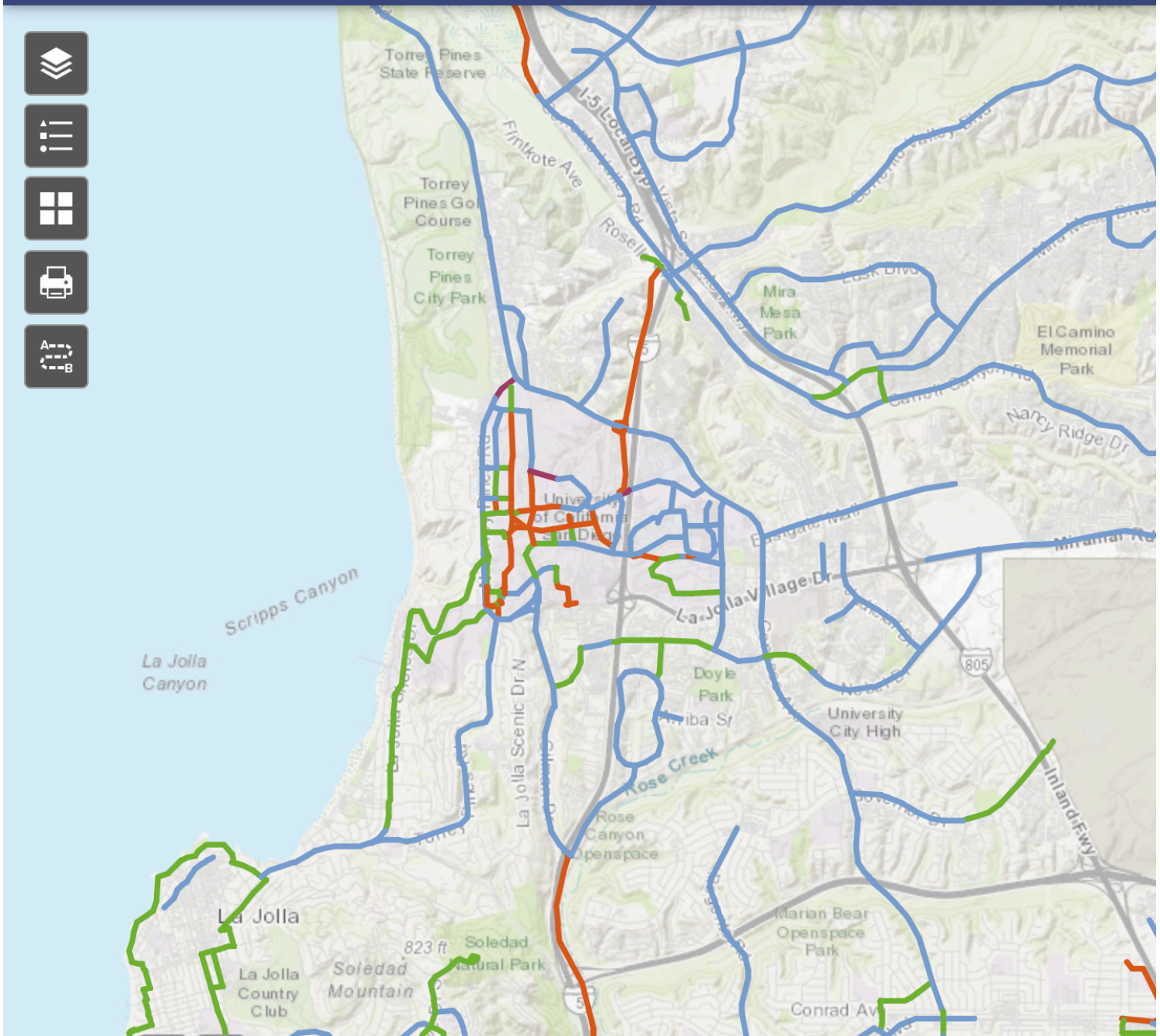
Architecture

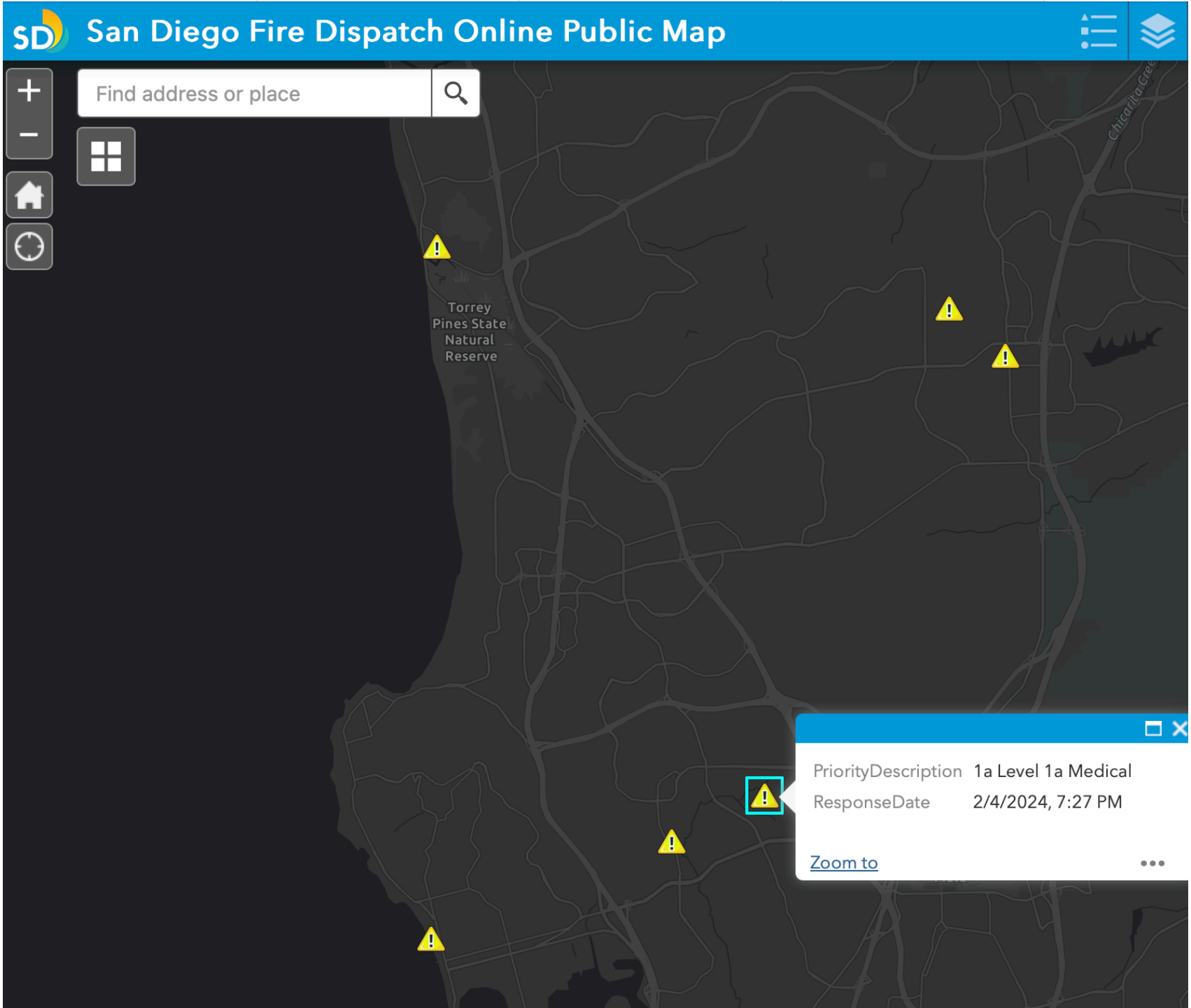


Next concept: *views*

- Often, there's too much information for you to show it all at once.

SANDAG San Diego Regional Bike Map




San Diego Fire Dispatch Online Public Map
Find address or place
Torrey Pines State Natural Reserve
PriorityDescription 1a Level 1a Medical
ResponseDate 2/4/2024, 7:27 PM
Zoom to

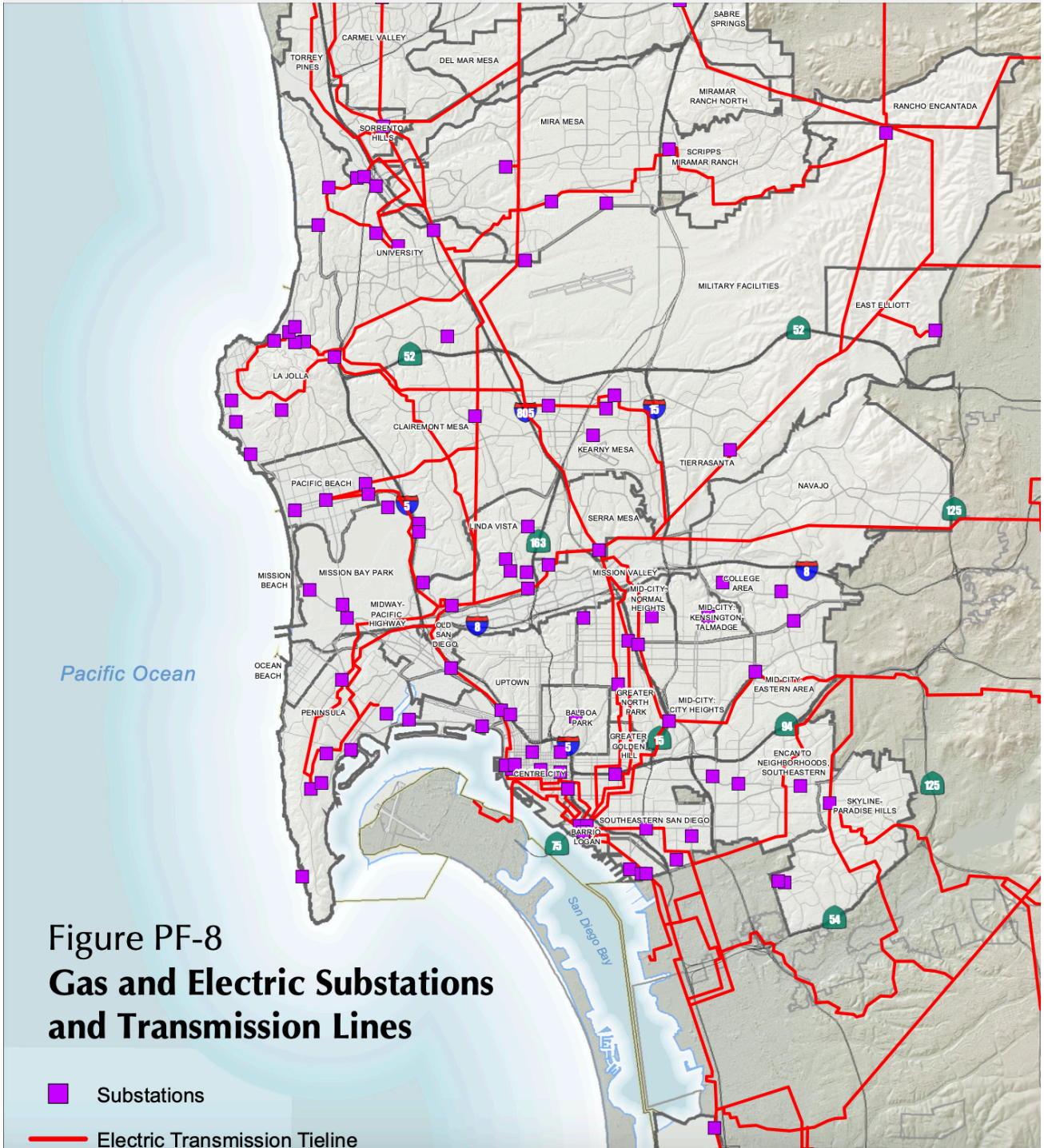


Figure PF-8

Gas and Electric Substations and Transmission Lines

Why Document Architecture?

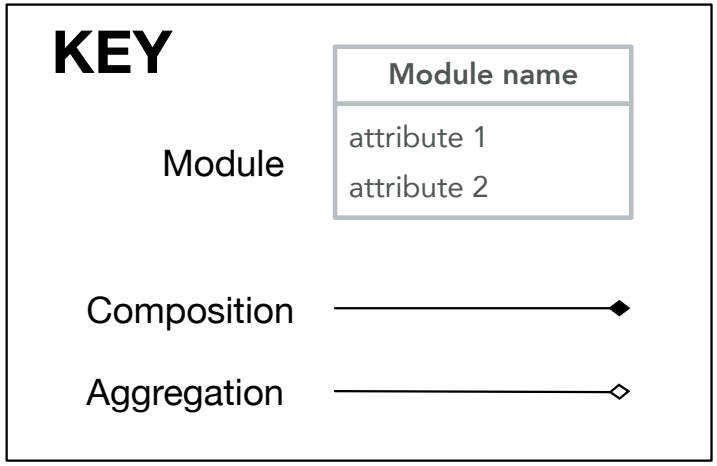
- Blueprint for the system
  - Artifact for early analysis
  - Primary carrier of quality attributes
  - Key to post-deployment maintenance and enhancement
- Documentation speaks for the architect, today and 20 years from today
  - As long as the system is built, maintained, and evolved according to its documented architecture
- Support traceability.

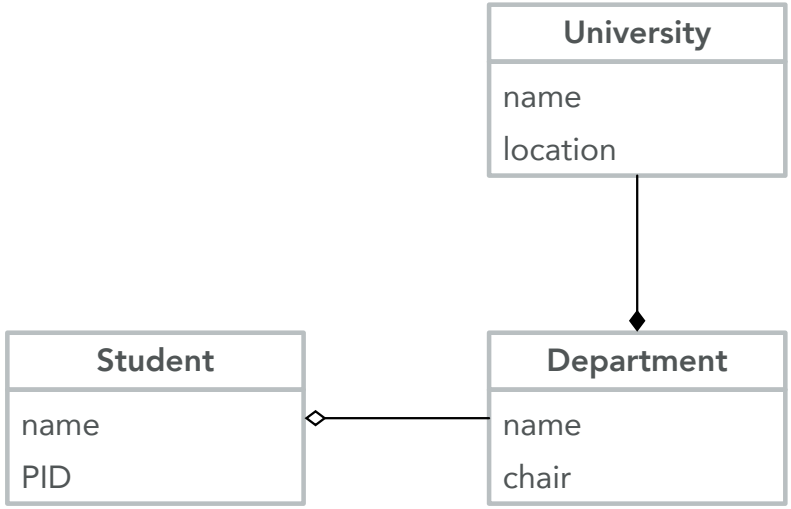
Views and Purposes

- Every view should align with a purpose
- Views should only represent information relevant to that purpose
  - Abstract away other details
  - Annotate view to guide understanding where needed
- Different views are suitable for different reasoning aspects (different quality goals), e.g.,
  - Performance
  - Extensibility
  - Security
  - Scalability
  - ...

Module views (*static*)

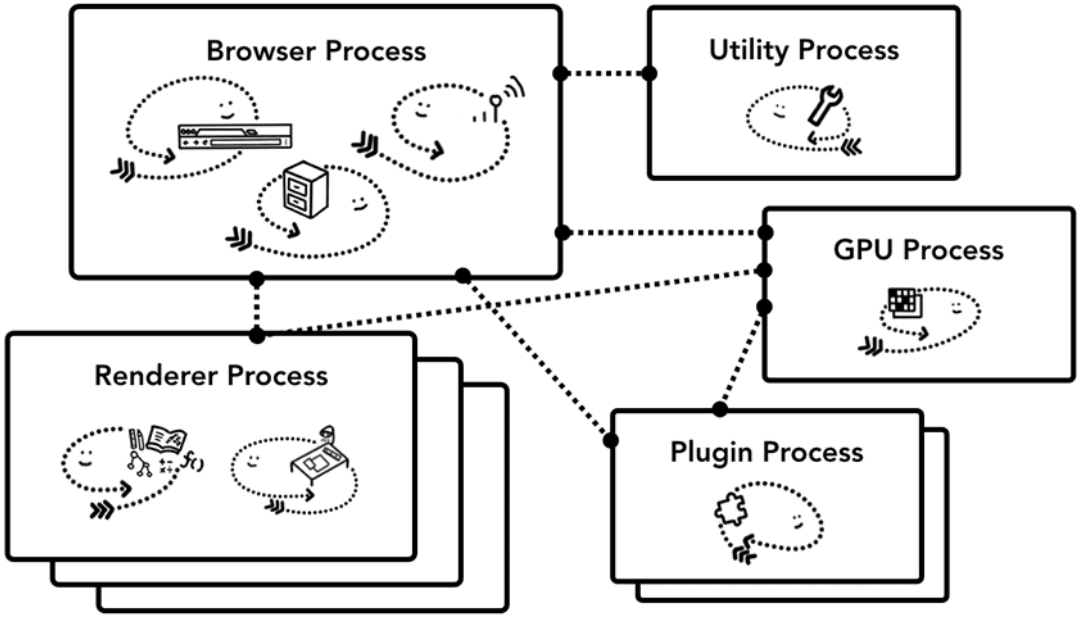
- Shows structures that are defined by the code
- Modules (subsystems, structures) and their relations (dependencies, ...)
- Often shows *decompositions* (a University consists of Departments) and *uses* (a Course uses a Classroom)





Component views (dynamic)

- Shows entities that exist at *run time*
- Components (processes, runnable entities) and connectors (messages, data flow, ...)
- These do not exist until the program runs; cannot be shown in a static view



https://developer.chrome.com/blog/inside-browser-part1

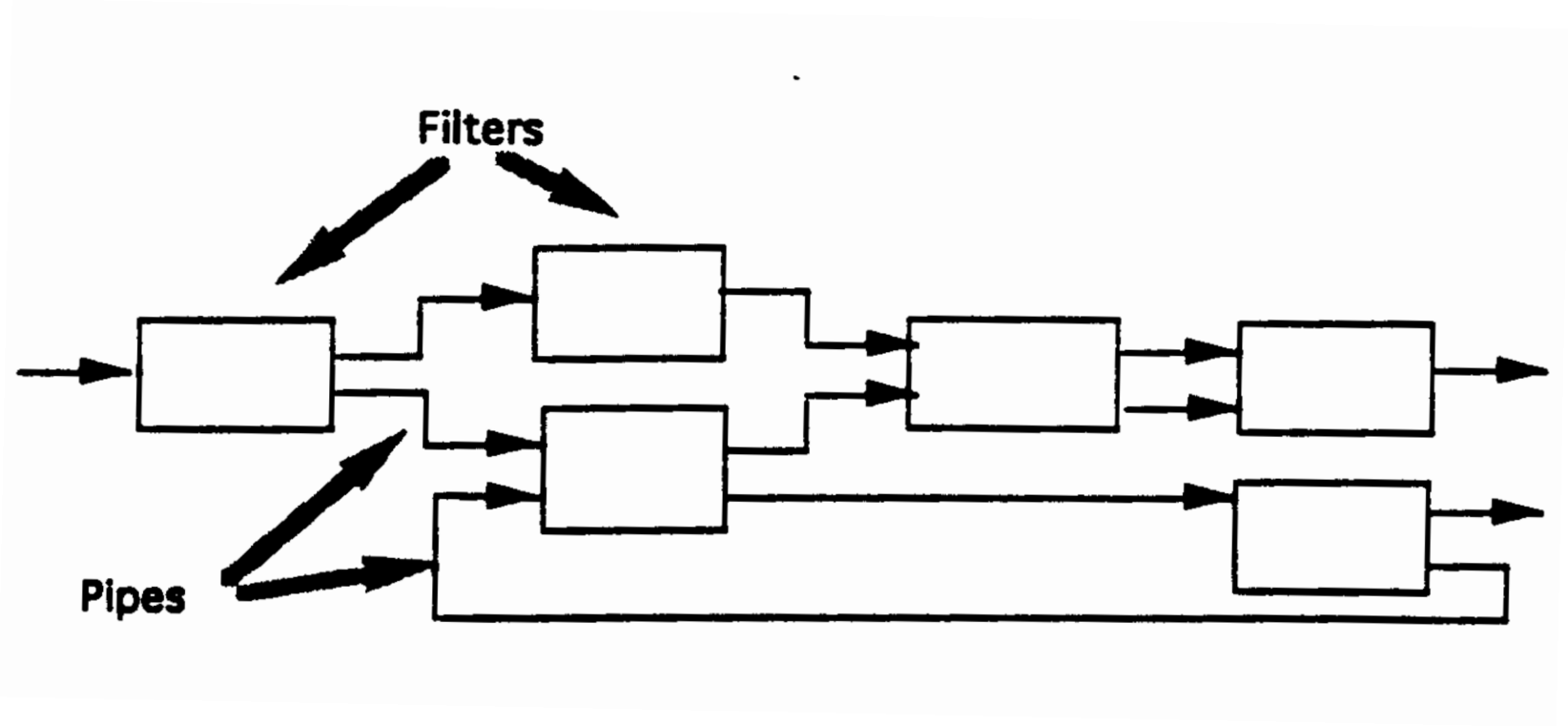
Physical view (deployment)

- Hardware structures and their connections
- Which parts of the system run on which physical machines?
- How do those machines connect?
- Example (you choose)

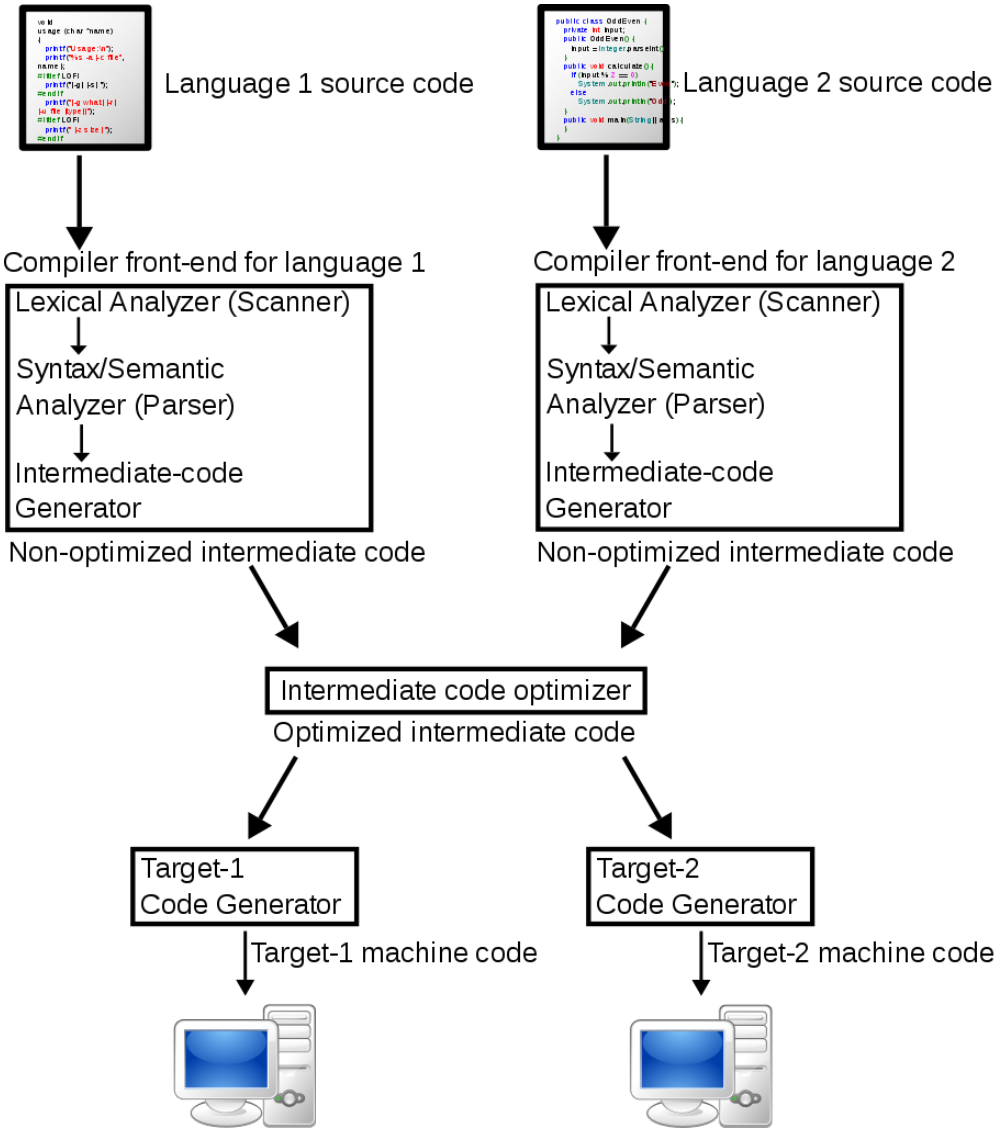
Software Architectural Styles

- A style describes a family of architectures
- Each style promotes some quality attributes and inhibits others
- Learning these patterns can enable you to make good architectural choices
- Important: "pure" styles rarely occur in practice
- But I will teach them as pure so we can study them individually
- Each style includes:
  - **Components** or **modules**
  - **Connectors** that describe relationships between components or modules

1. Pipes and Filters (one style in the "data flow" family of styles)



Example: Compilers



Example: UNIX pipes

- Filters: processes
  - Ports: stdin, stdout, stderr
- Pipes: buffered streams
  - Pipes carry byte streams (usually assume: UTF-8 strings)

Pipes vs. Procedures

	Pipes	Procedures
Arity	Binary	Binary
Control	Asynchronous, data-driven	Synchronous, blocking
Semantics	Functional	Hierarchical
Data	Streamed	Parameter/return value
Variations	Buffering, end-of-file behavior	Binding time, exception handling, polymorphism

Table from David Garlan

Analysis

- Quality attributes promoted:
  - Modifiability: can insert or remove filters
  - Modifiability: can redirect pipes
  - Reuse
  - Performance: enables parallel computation
- Quality attributes inhibited:
  - Usability: hard to build interactive applications this way
  - Performance: may have to translate data to be sent on pipes
  - Cost: writing filters may be complex due to common pipe data format
  - In some cases, correctness, if need to synchronize across pipes