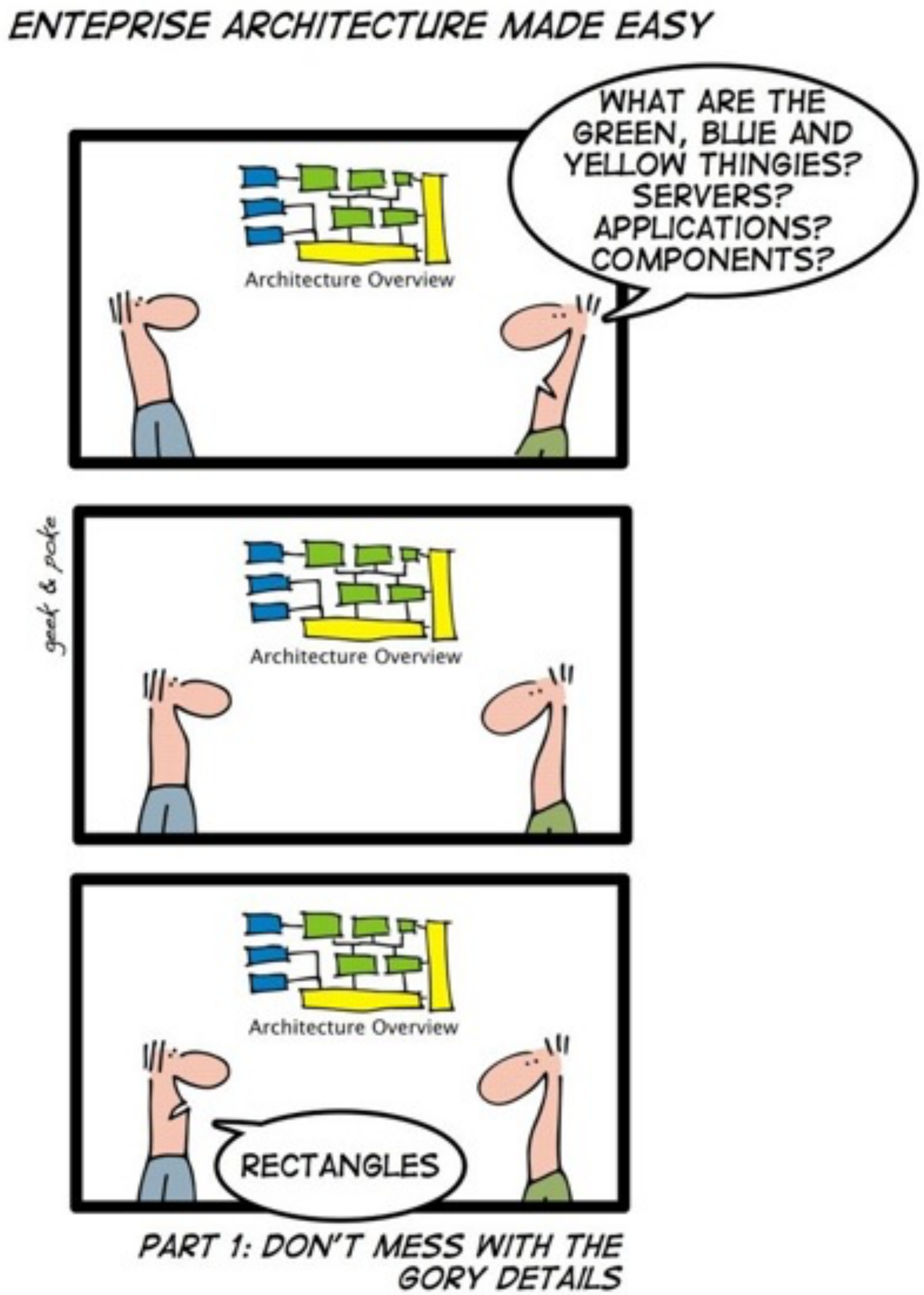
Introduction to Software Architecture: Views

Michael Coblenz


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

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]

Note: this definition is ambivalent to whether the architecture is known, or whether it's any good!

Reminder: 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.

Abstraction

- Goal: Reason without understanding implementation details
- Approach:
  - Divide enormous system into smaller pieces
  - Define what those pieces do and how they relate to each other

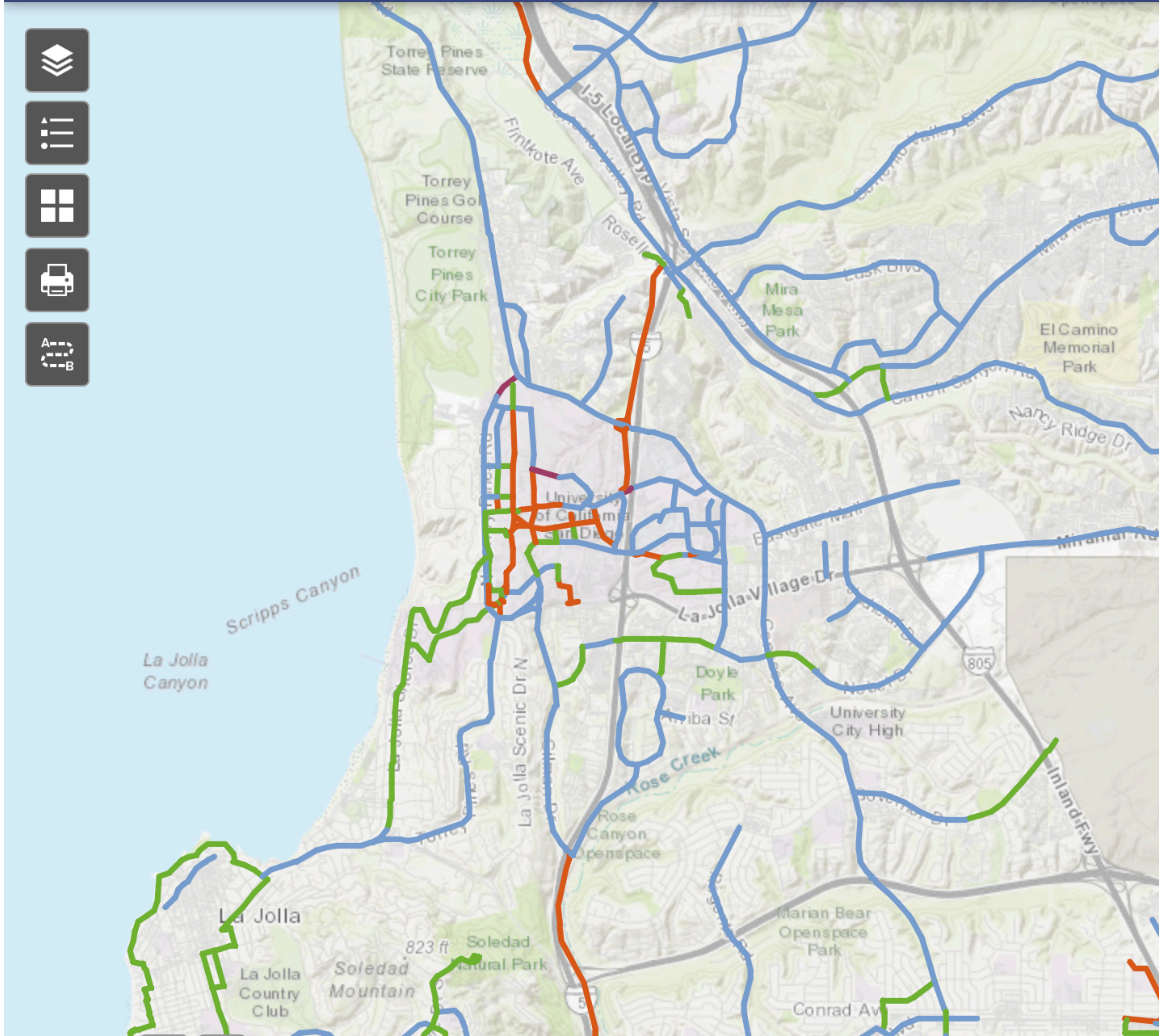
Considerations for Decomposition

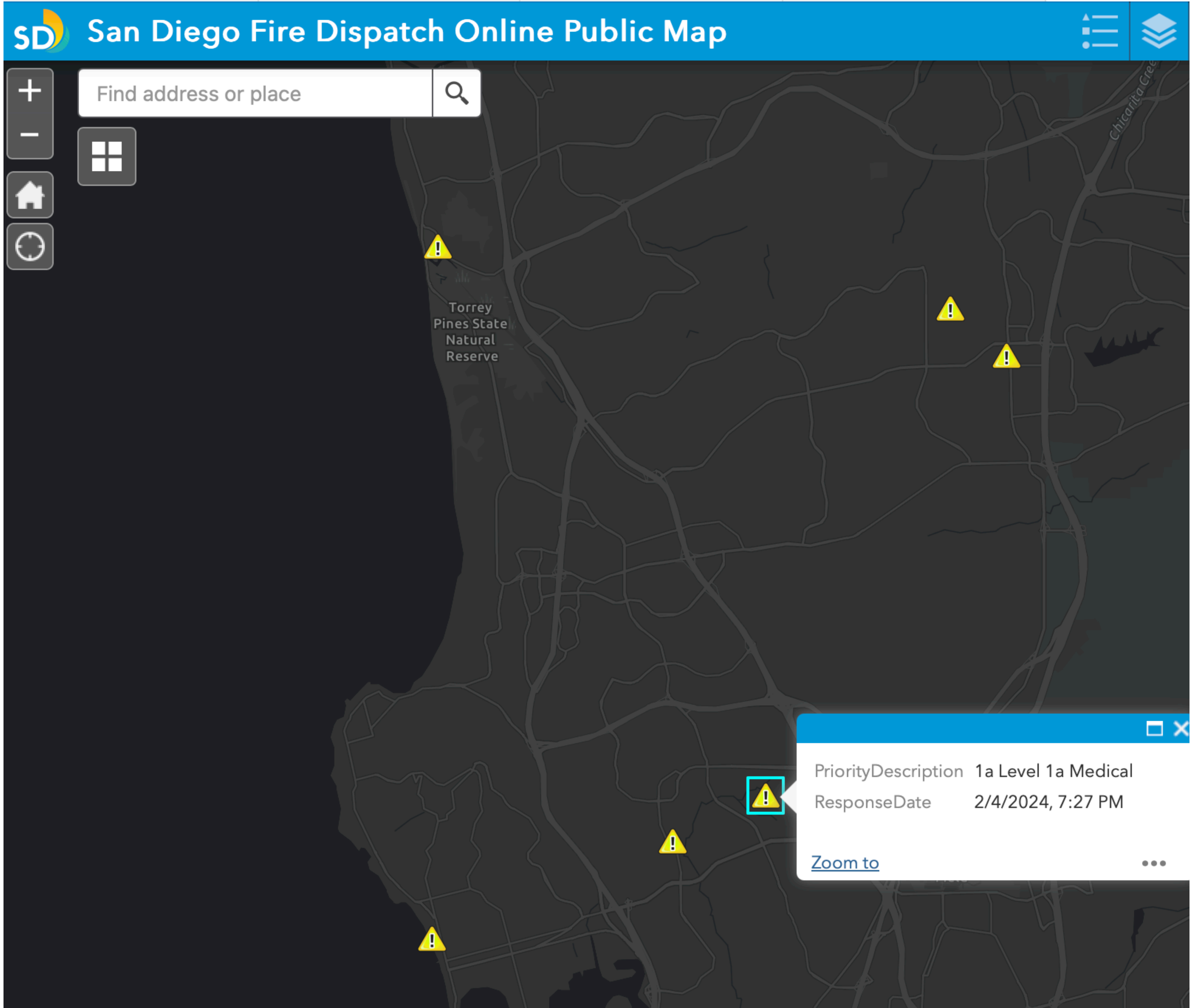
- Conceptual integrity (each piece is responsible for one "thing")
- Conway's Law (organizations inevitably produce copies of their org charts)
- Minimize coupling (avoid entangling separate modules)
- Maximize cohesion (everything in a module fits the theme)
- Use known-good solutions for prioritized quality attributes (see book)

Views

- Once you have a design, how do you draw it?

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

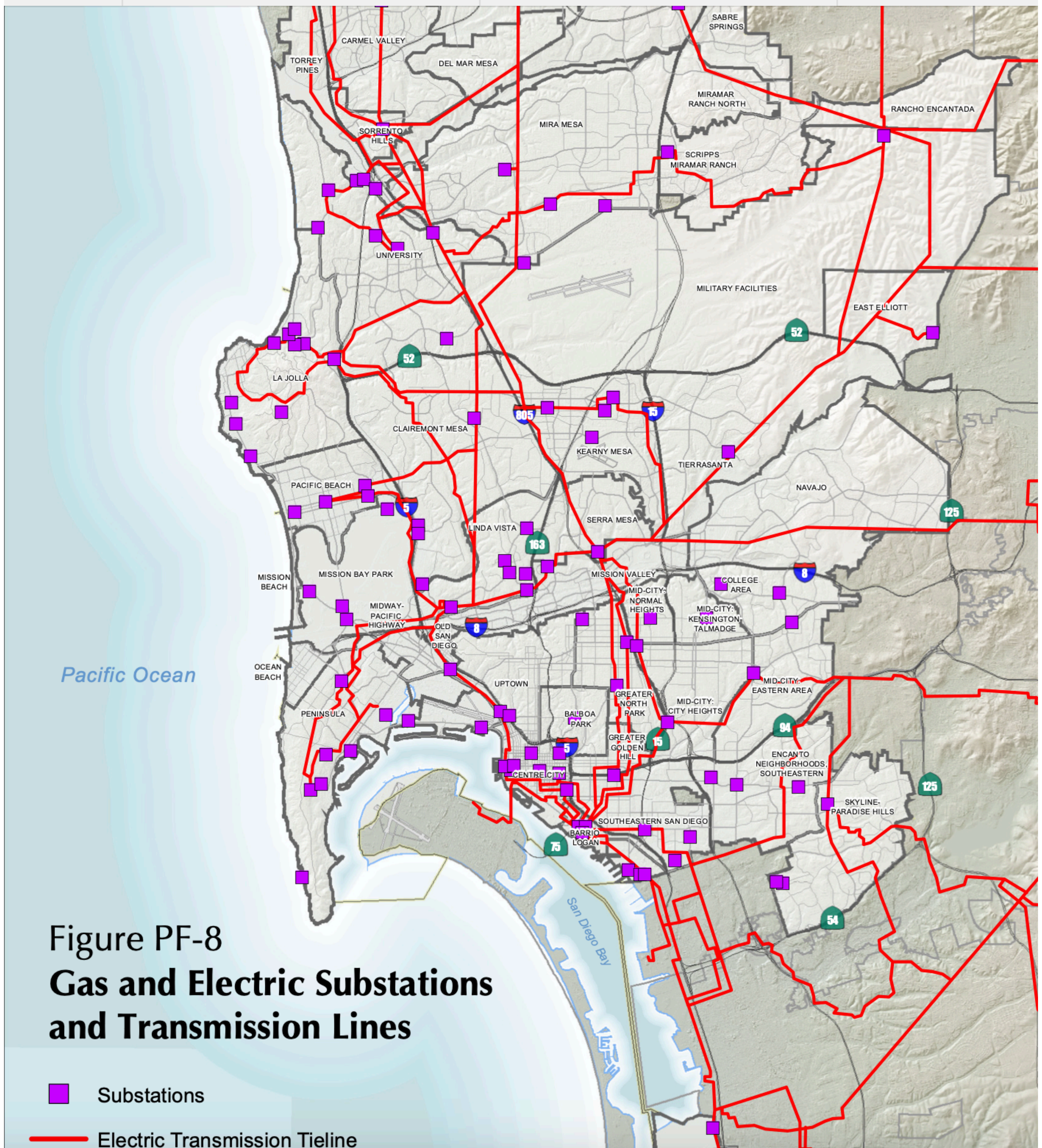


Figure PF-8
Gas and Electric Substations and Transmission Lines

Views and Purposes

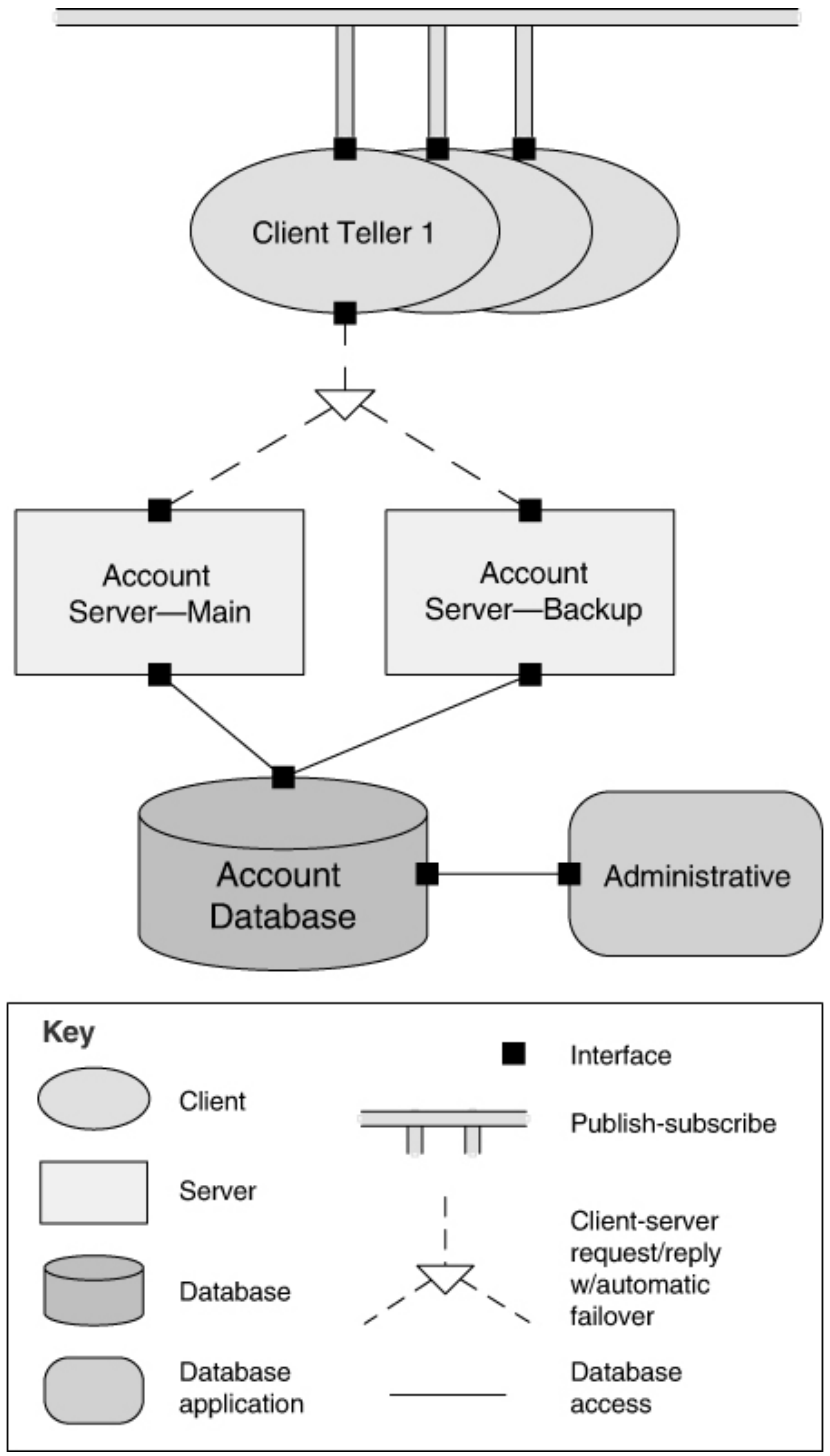
- Every view should align with a purpose
- Views should only represent information relevant to that purpose
  - Abstract away other details
  - Annotate view to guide understanding where needed
- Different views are suitable for different reasoning aspects (different quality goals)
  - Performance
 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

Architectural Structures

- Three kinds of structures:
    - Components and connectors (runtime entities)
    - Modules (static entities)
    - Allocations (mapping of software to the real world)
- Each type has its own kind of *view*

Components and Connectors (for *run time* entities)

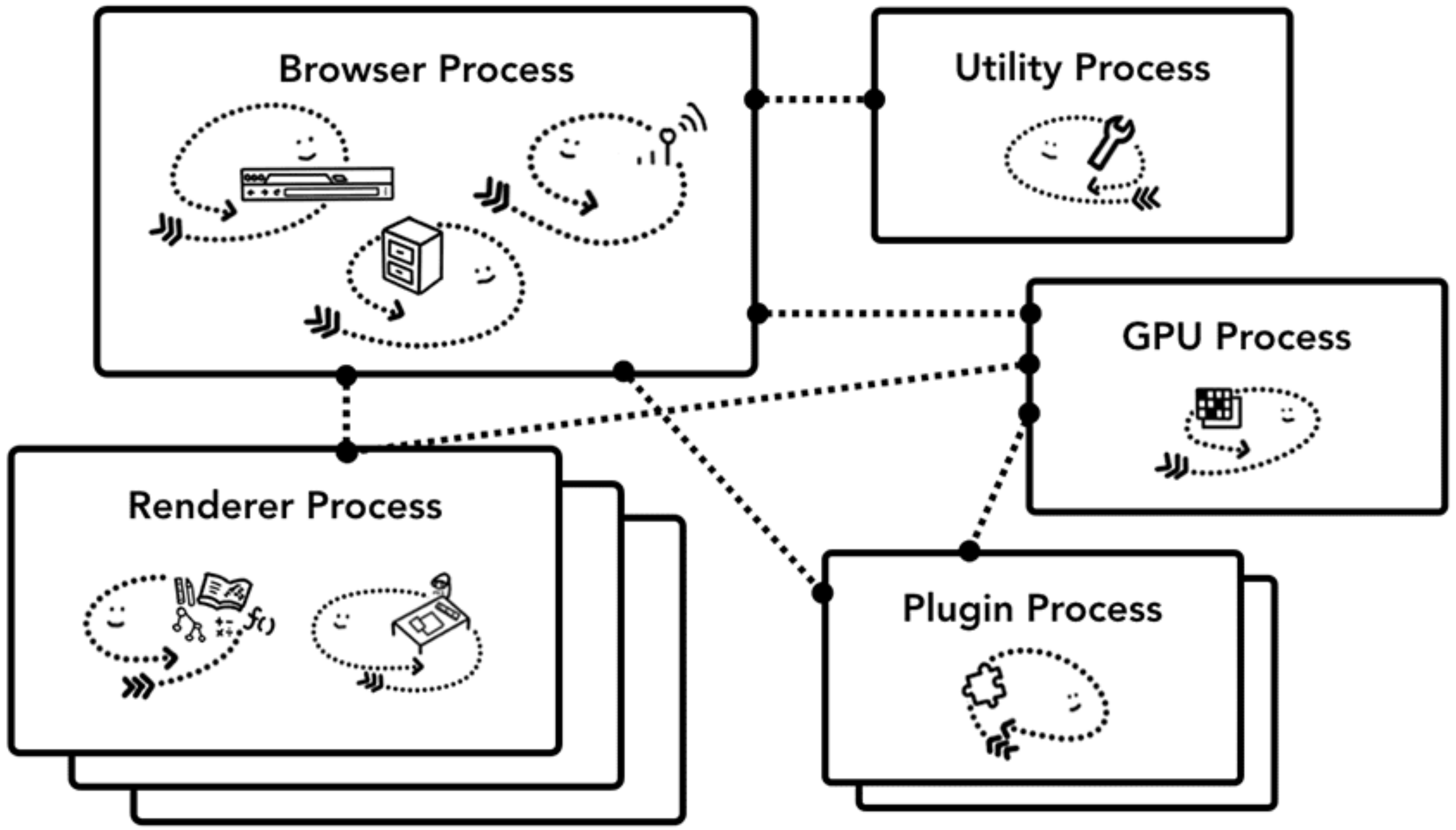
- These show:
  - Major executing components and interactions
  - Major shared data stores
  - Replicas
  - How data progresses through system
  - Which parts run in parallel
  - How structure can change at run time



Credit: *Software Architecture in Practice*

Component views (dynamic)

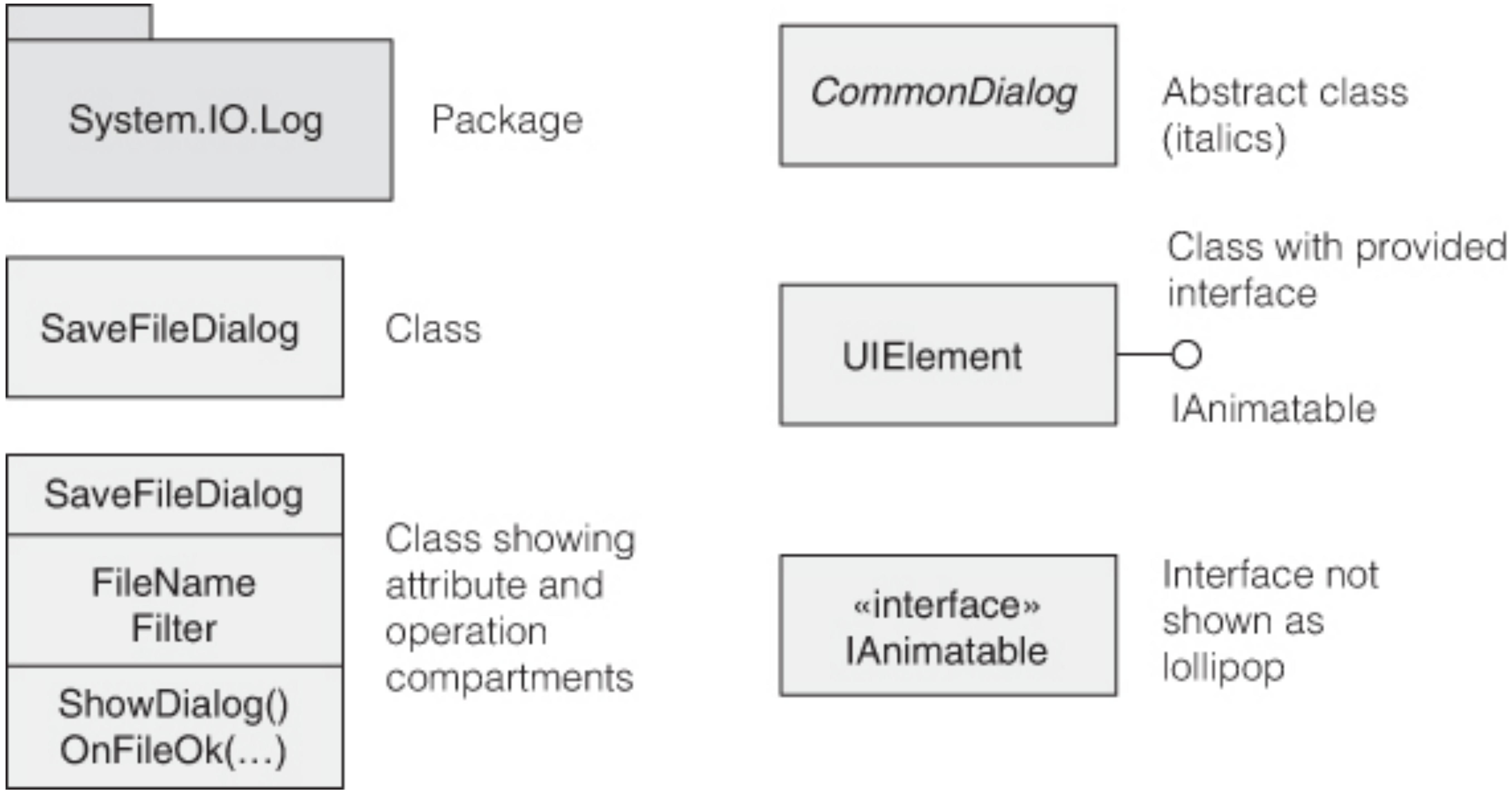
- Shows entities that exist at *run time*
- Components (processes, runnable entities) and connectors (messages, data flow, ...)
- These do not exist until the program runs; cannot be shown in a static view



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

Module Structures (for *static* entities)

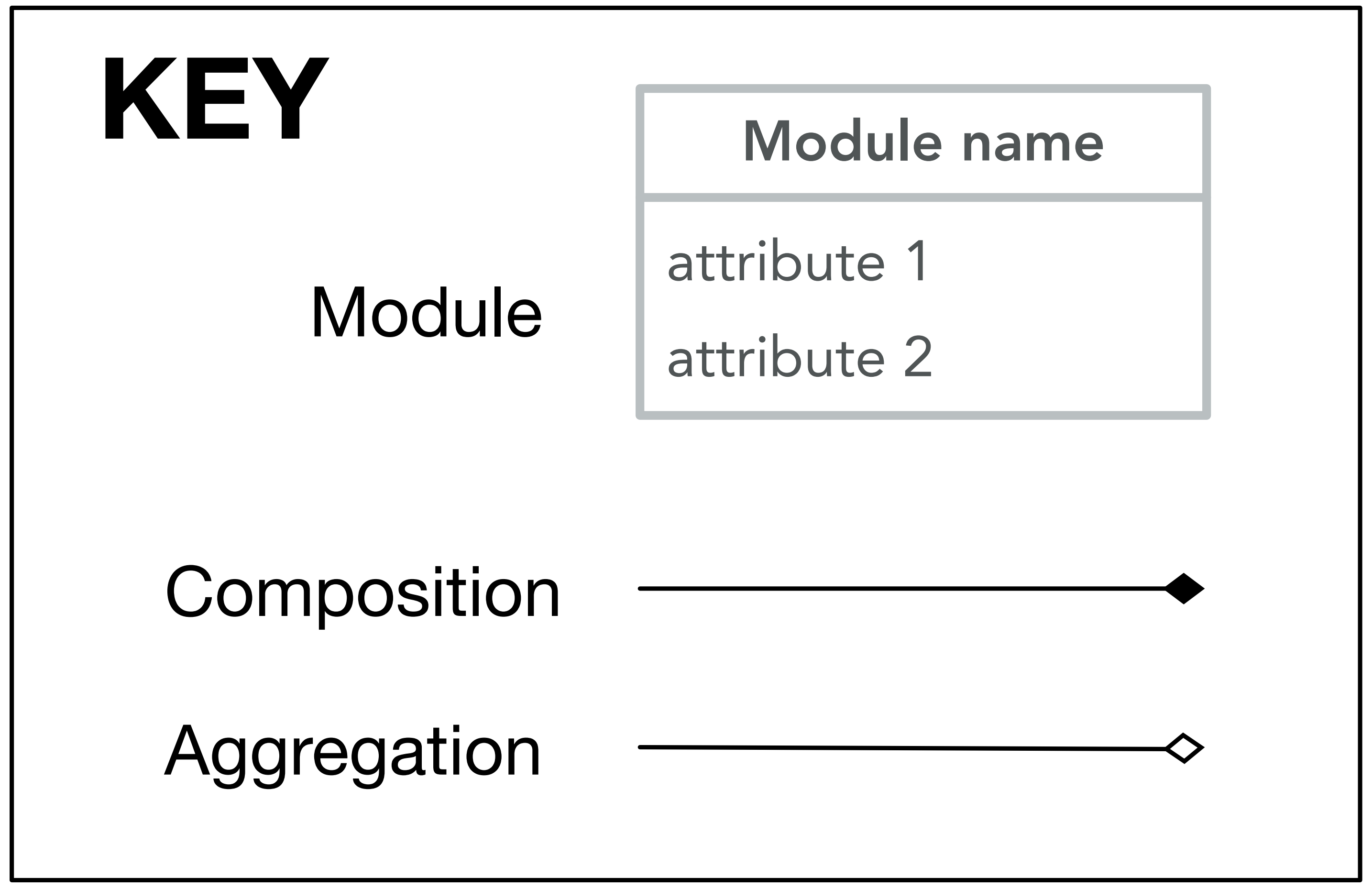
- Show how responsibilities are held by *code* structures
- Packages, classes, layers

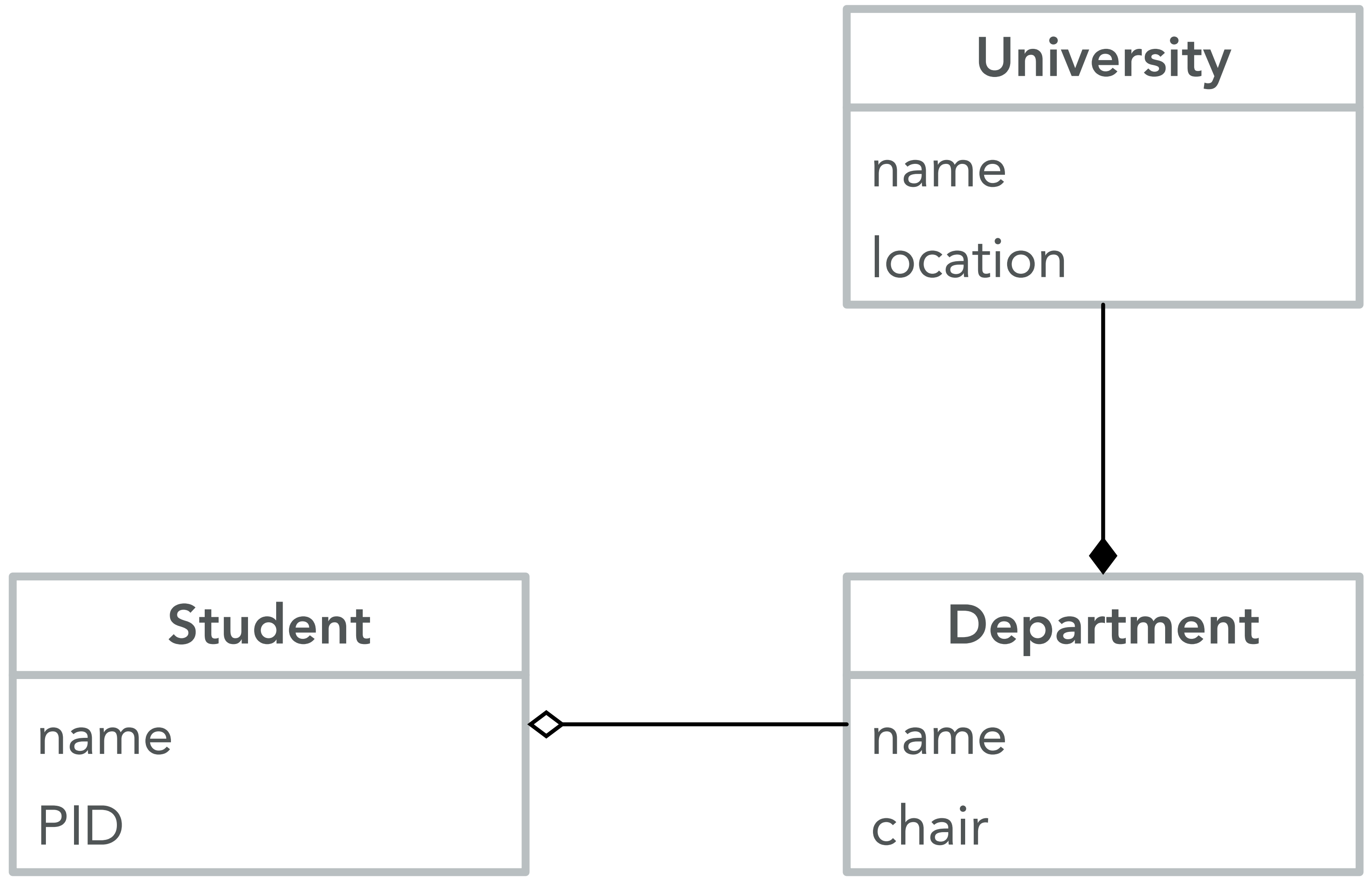


Credit: *Software Architecture in Practice*

Module views (*static*)

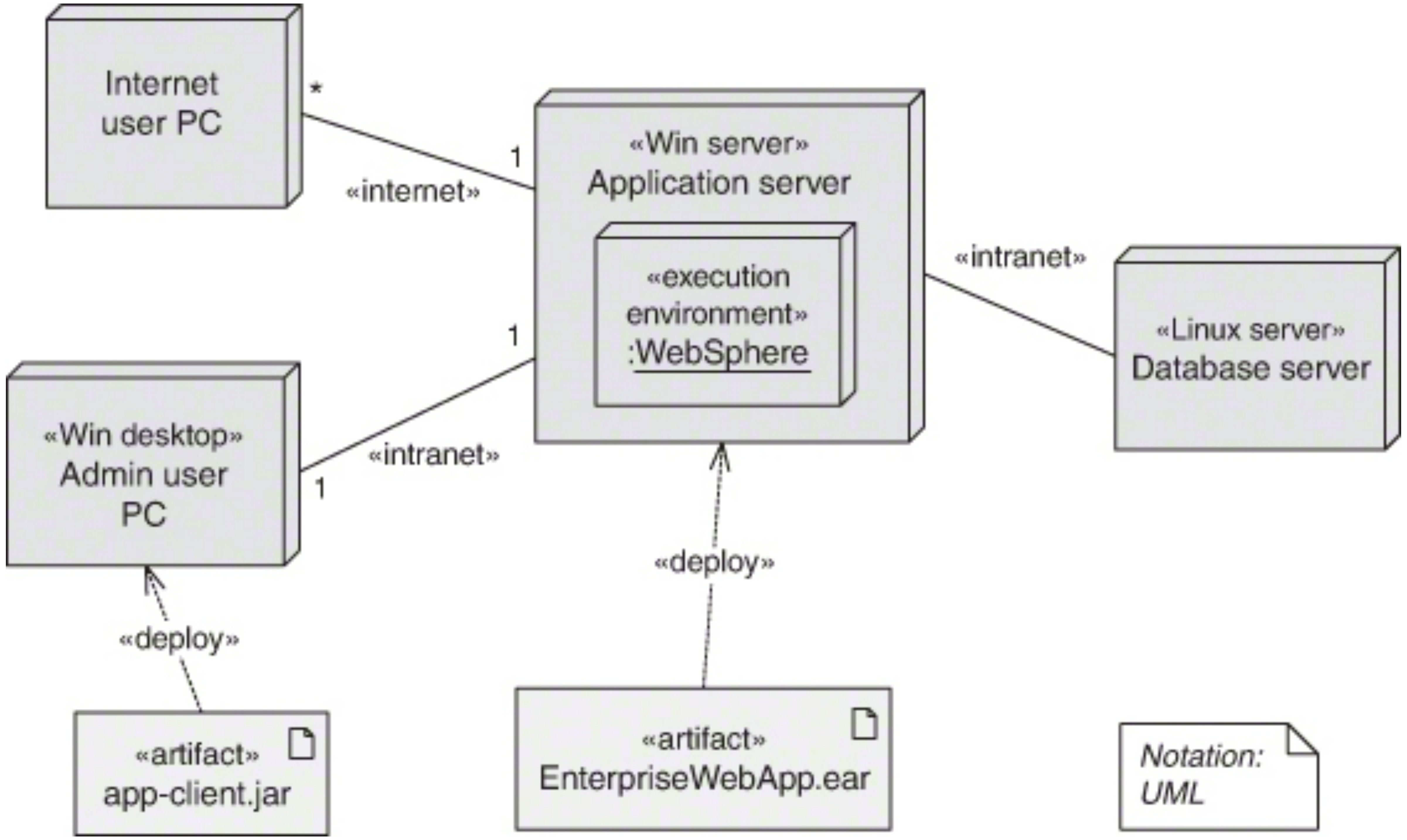
- Shows structures that are defined by the code
- Modules (subsystems, structures) and their relations (dependencies, …)
- Often shows *decompositions* (a University consists of Departments) and *uses* (a Course uses a Classroom)





Allocation Views (relate different kinds of dynamic components)

- Example: deployment view shows how software artifacts are deployed on servers



Credit: *Software Architecture in Practice*

Physical view (deployment)

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

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.

Styles

Software Architectural Styles

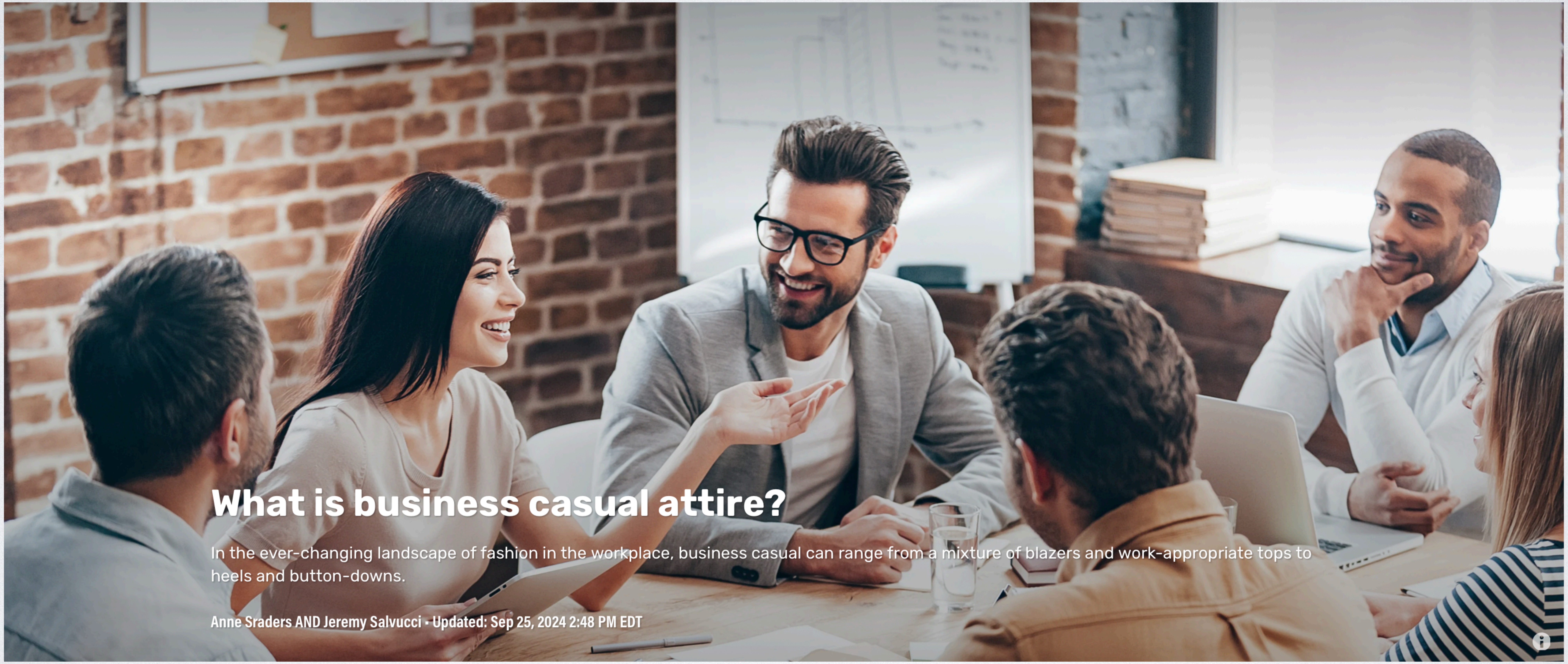
- A style describes a family of architectures
- Each style promotes some quality attributes and inhibits others
- Learning these patterns can enable you to make good architectural choices
- *Pure* styles rarely occur in practice
- Each style includes:
  - **Components** or **modules**
  - **Connectors** that describe relationships between components or modules

Compare: Clothing Styles

- "Business casual is typically defined as no jeans, no shorts, no short dresses or skirts for women, optional ties for men, and a rotation of button-downs or blouses. Business casual dressing is more about avoiding a list of "don'ts" than following a list of "dos" and can vary slightly depending on style, preference, and gender presentation."

https://www.thestreet.com/lifestyle/what-is-business-casual-14629494

Compare: Clothing Styles

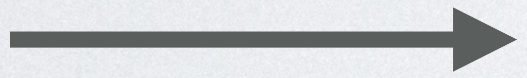


https://www.thestreet.com/lifestyle/what-is-business-casual-14629494

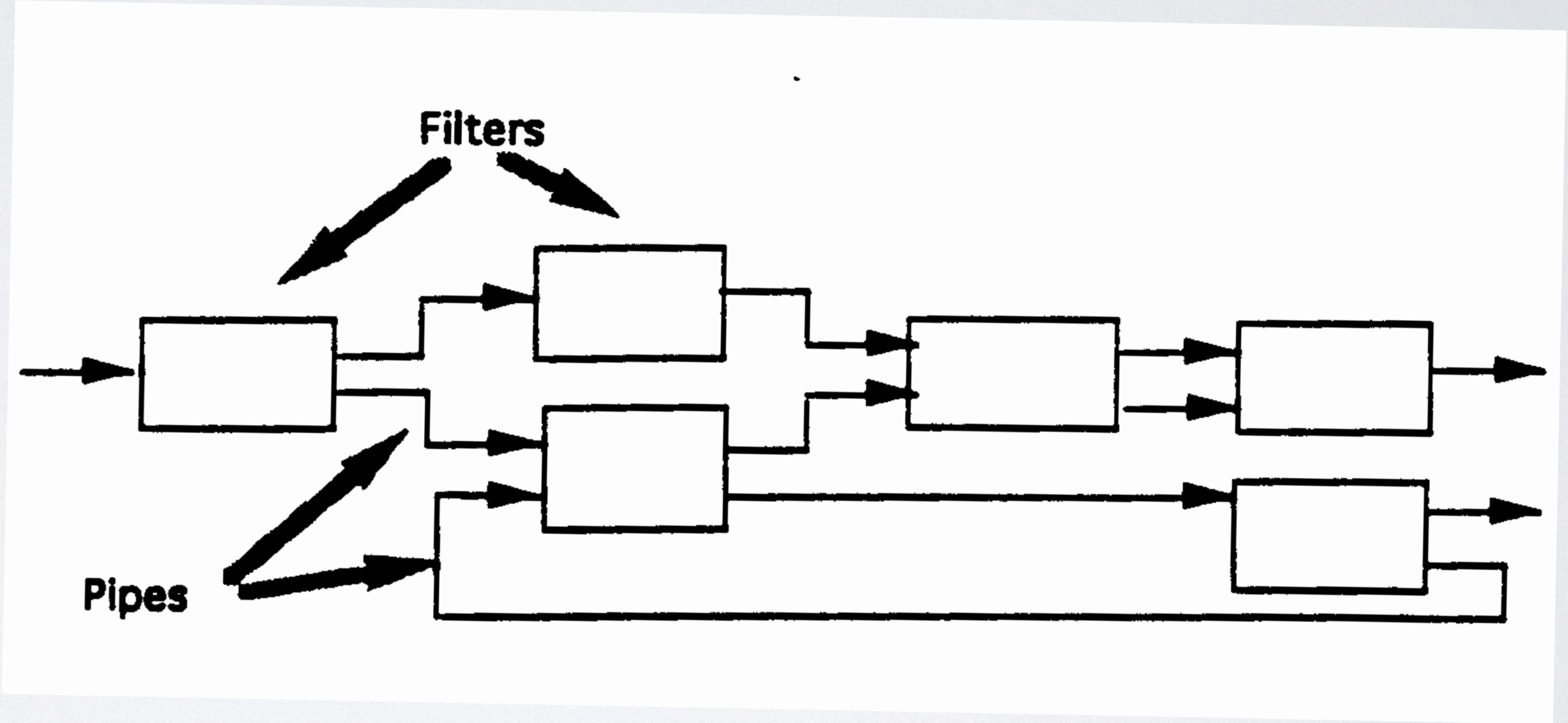
1. Pipes and Filters (One Style in the "Data Flow" Family of Styles)

Key

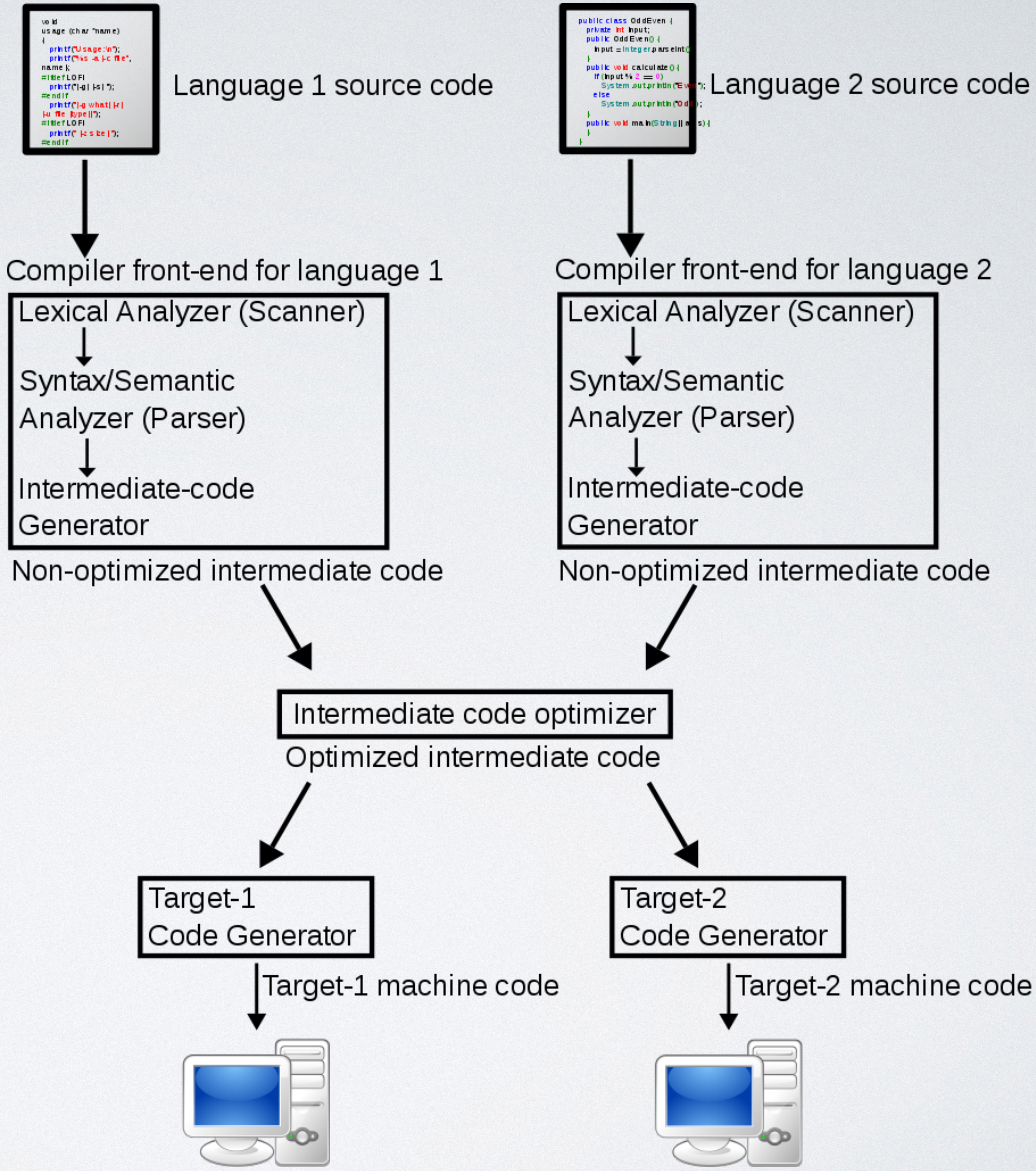
Component



Data flow



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

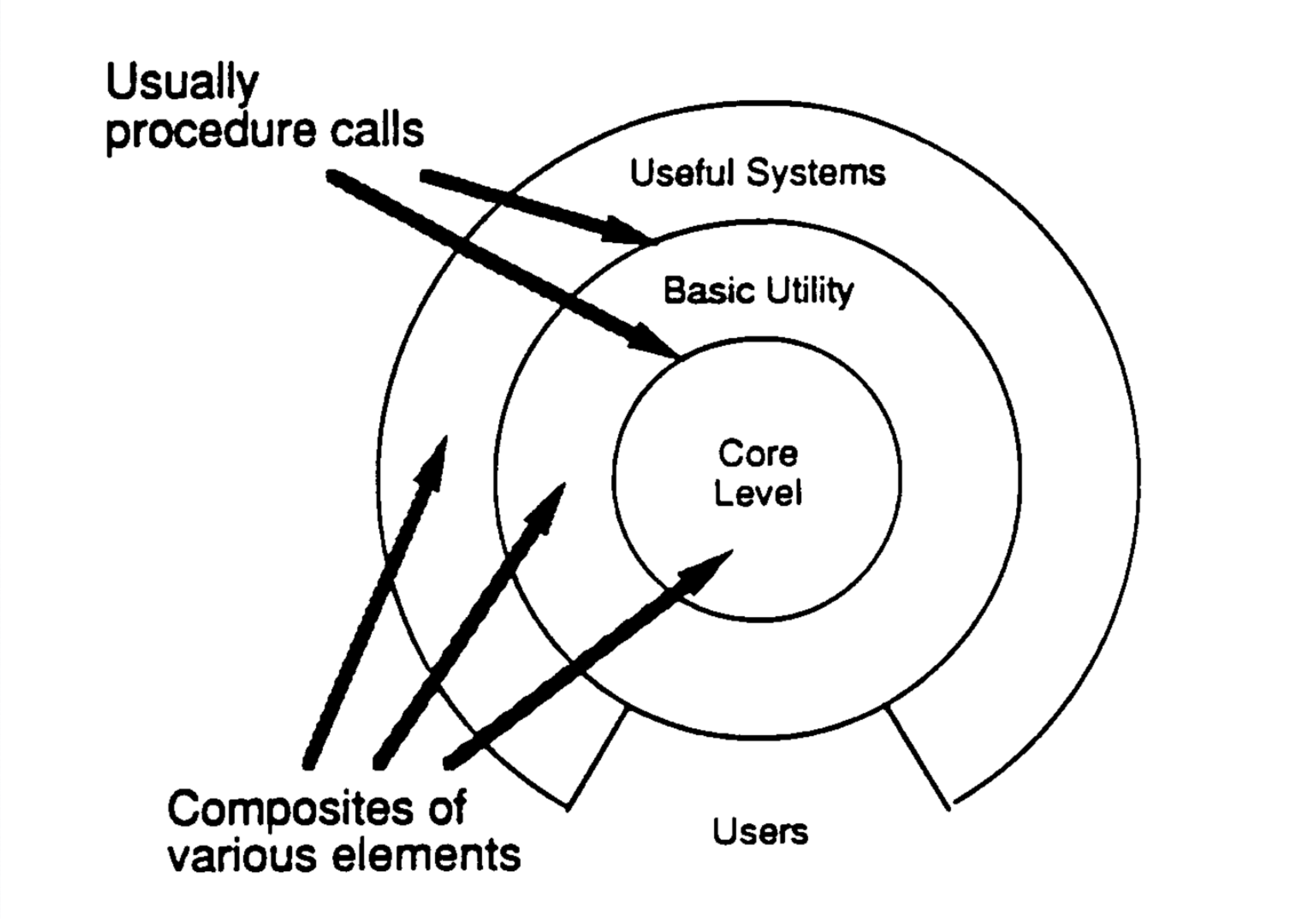
	Pipes	Procedures
Arity	Binary	Binary
Control	Asynchronous, data-driven	Synchronous, blocking
Semantics	Functional	Hierarchical
Data	Streamed	Parameter/return value
Variations	Buffering, end-of-file behavior	Binding time, exception handling, polymorphism

Table from David Garlan

Analysis

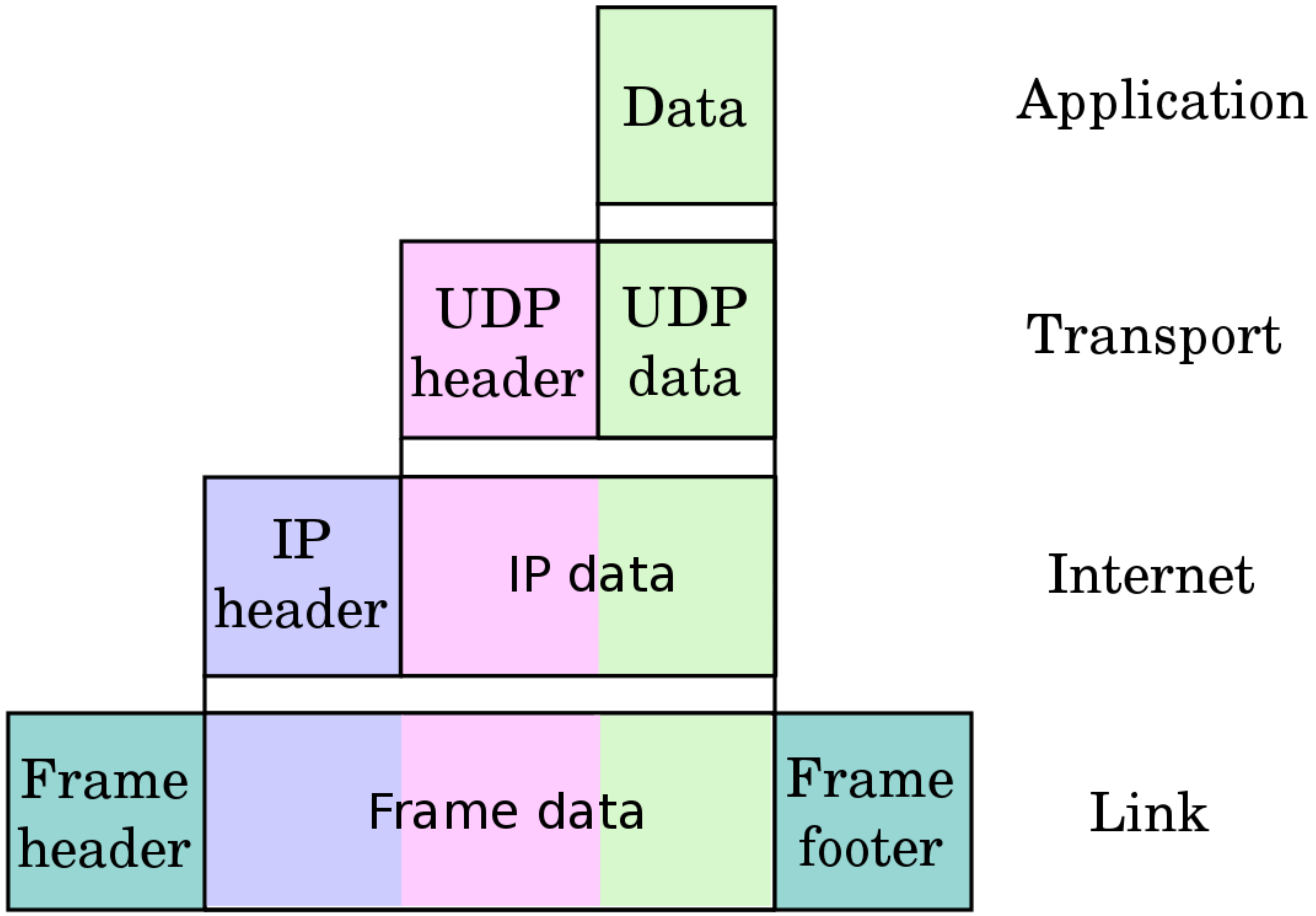
- Promotes:
  - Modifiability: can insert or remove filters
  - Modifiability: can redirect pipes
  - Reuse
  - Performance: enables parallel computation
- Inhibits:
  - Usability: hard to build interactive applications this way
  - Performance: may have to translate data to be sent on pipes
  - Cost: writing filters may be complex due to common pipe data format
  - Correctness, if need to synchronize across pipes

Layered Styles



© David Garlan and Mary Shaw, CMU/SEI-94-TR-021

Example: Internet Protocol Suite

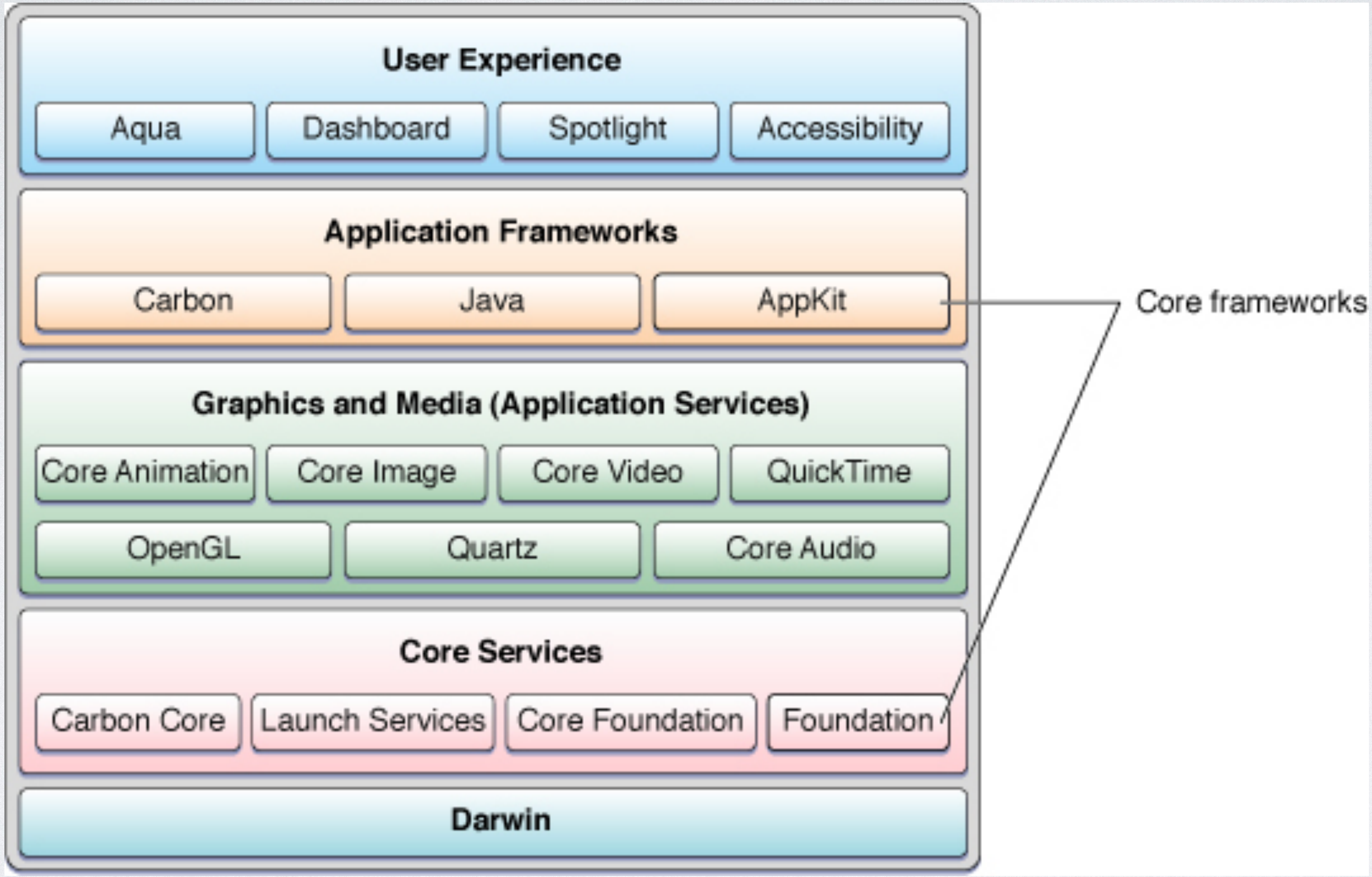


Slide credit: Michael Hilton

Layered Styles

- Note: we're talking about **static** entities here (classes, modules, etc.)
- Constraint: only invoke code at lower levels
  - Variation: only the next level down
- Benefits:
  - Changes only affect layer(s) above (not the whole system)
  - Reuse (swap out implementation of a layer)
- Considerations:
  - Hard to choose right layers
  - Which layer does this code go in?

Example: macOS



https://developer.apple.com/library/archive/documentation/Cocoa/Conceptual/CocoaFundamentals/WhatIsCocoa/WhatIsCocoa.html#//apple_ref/doc/uid/TP40002974-CH3-SW16

Tiers

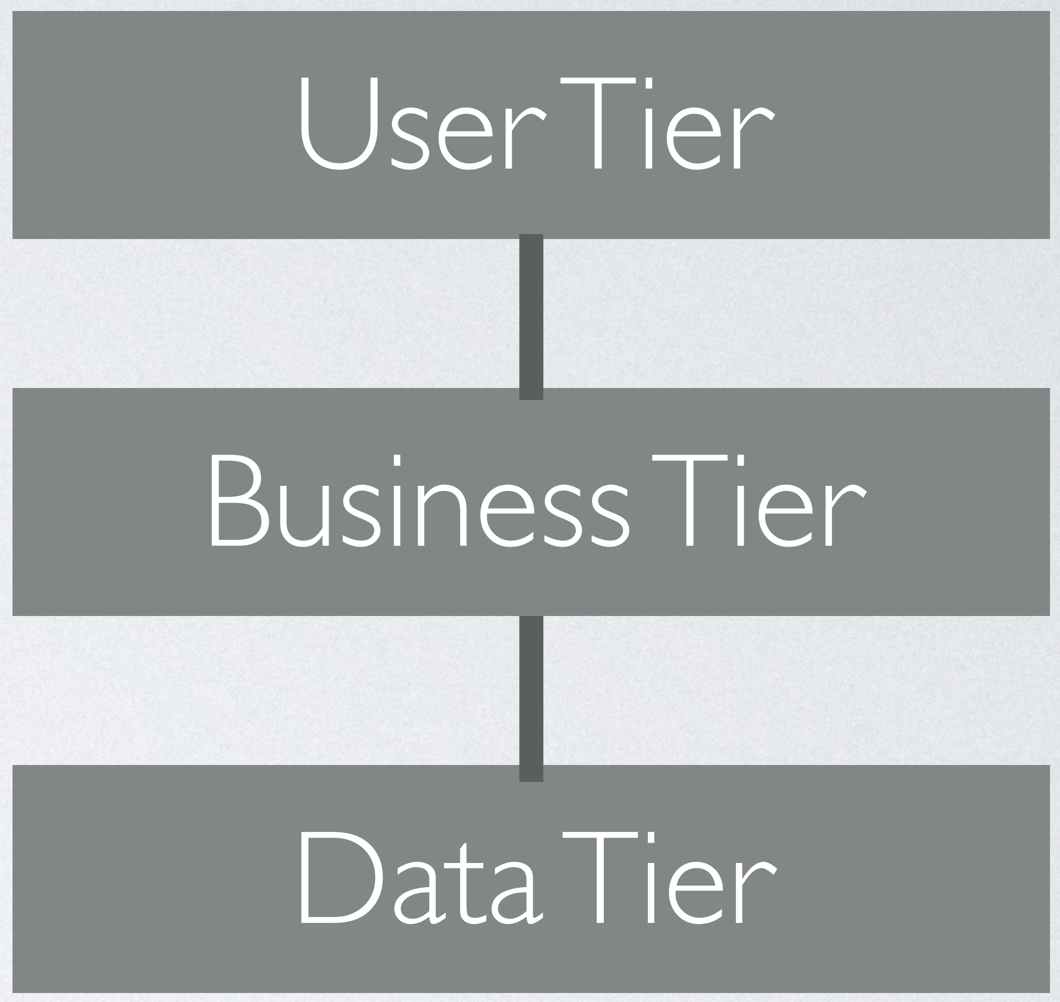
- Organize clients and servers into tiers
- IMPORTANT: tiers can be seen in a RUNTIME view
- Tiers provide services above, rely on services below

Constrast: Layers

- Layers appear in a module (static) view

3-Tiered Client-Server

- Promotes:
  - security (user can't access data directly)
  - performance (separate tiers can run on separate hardware)
  - availability (replicate tiers)



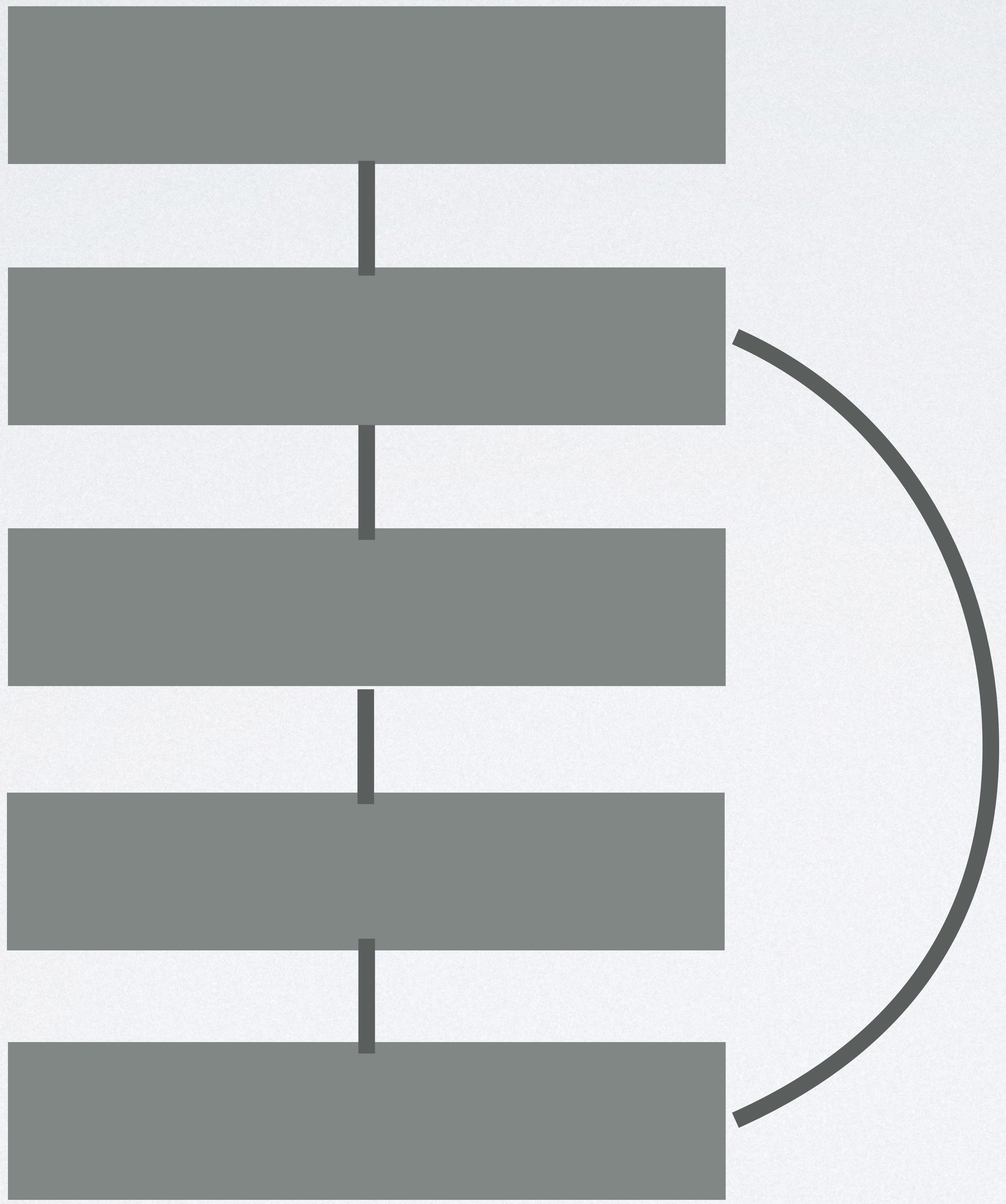
Tiered Style Rules

- Each component is in exactly one tier
- Each component can use services in:
  - Any lower tier; or
  - Next tier down
- Components {can or cannot} use components in same tier

Tiered Style Tradeoffs

- Advantages:
  - Tiers reflect clean abstractions
  - Promotes reuse
- Disadvantages:
  - Unclear which tier a component belongs in
  - What if a computation fits in multiple layers?
  - Performance implications motivate inappropriate connections around layers (tunneling)

Tunneling



Violates layering architecture…
but sure is convenient!
Maybe also improves performance.

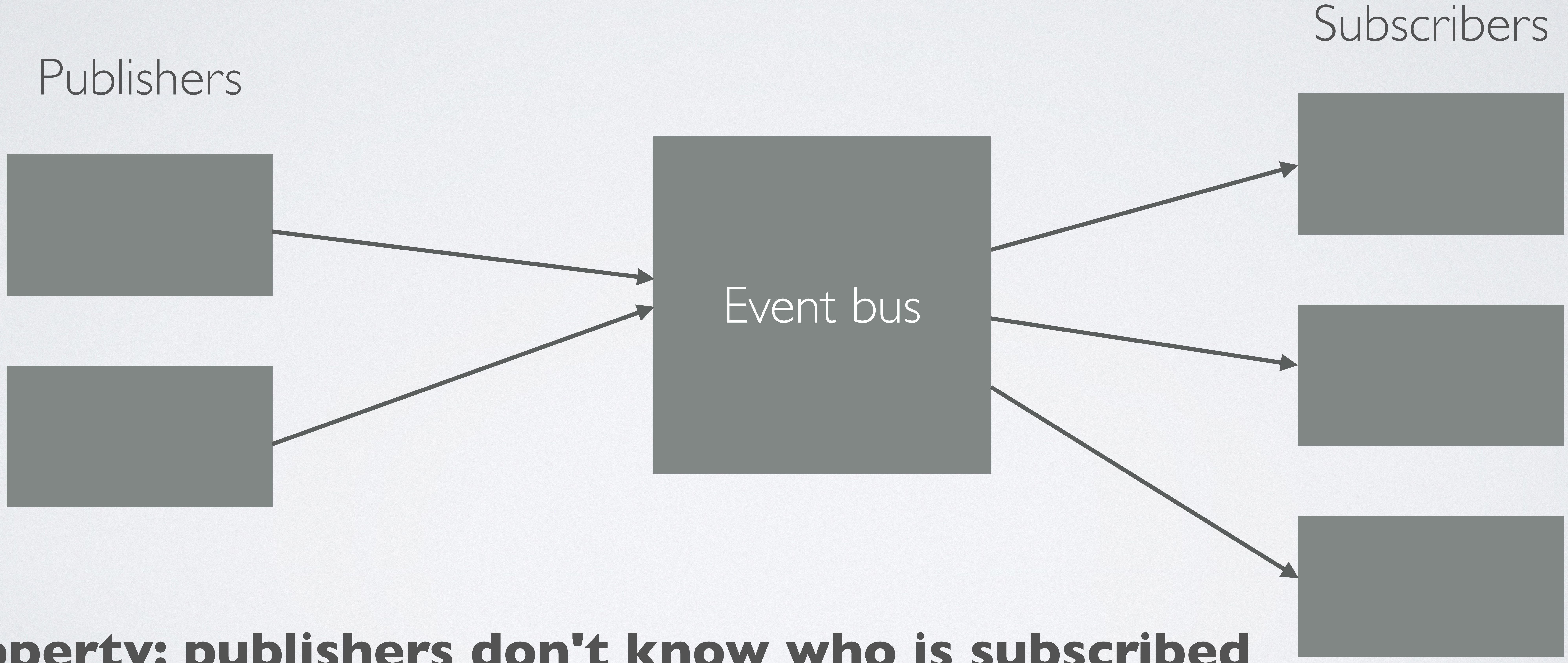
Client-Server Architecture

- Clients know who the server is
- Server knows little about the clients (number, identity)
- Agree on protocol in advance

Client/Server Tradeoffs

- Promotes:
  - Scalability: easy to add more clients, servers
  - Modifiability: can swap out clients and servers separately
- Inhibits:
  - Reliability (server/network may be down)
  - Performance (network bandwidth, latency)
  - Security (open ports)
  - Simplicity (more failure modes to test)

Publish-Subscribe Style (Also Called "Implicit Invocation"



Key property: publishers don't know who is subscribed

Implicit Invocation

- Benefits:
  - Decouples publishers from subscribers
  - Promotes reuse: add a component by registering it for events
- Potential problems:
  - Order of event delivery is not guaranteed
    - Warning: bugs will result from accidentally depending on this order
- Choose: synchronous or asynchronous event processing