SE 464
Week 1

Design Patterns & Architecture

Design Patterns

Note: The following slides were taken from Professor Werner Dietl (UWaterloo) with permission. These slides were from the Fall 2022 oering of SE 464.

Why design patterns?

- Ease communication by using a shared **vocabulary**
- **Leverage** existing design knowledge
- Enhance **flexibility** for future change
- Increase **reusability** of developed code

“Gang of Four” (GoF) Design Patterns

By Purpose				
		Creational	Structural	Behavioral
By Scope	Class	• Factory Method	• Adapter (class)	• Interpreter • Template Method
	Object	• Abstract Factory • Builder • Prototype • Singleton	• Adapter (object) • Bridge • Composite • Decorator • Façade • Flyweight • Proxy	• Chain of Responsibility • Command • Iterator • Mediator • Memento • Observer • State • Strategy • Visitor

Design Patterns: Elements of Reusable Object-oriented Software by Gamma, Helm, Johnson, and Vlissides, 1994

http://www.gofpatterns.com/

Design Patterns

Common solutions to recurring design problems.

Abstract recurring structures.

Comprised of class and/or object:

- Dependencies
- Structures
- Interactions
- Conventions

Names the design structure explicitly.

Distills design experience.

Design Patterns

Design patterns have four main parts:

1. Name
2. Problem
3. Solution
4. Consequences / trade-offs

Are language-independent.

Are “micro-architectures”.

Cannot be mechanically applied.

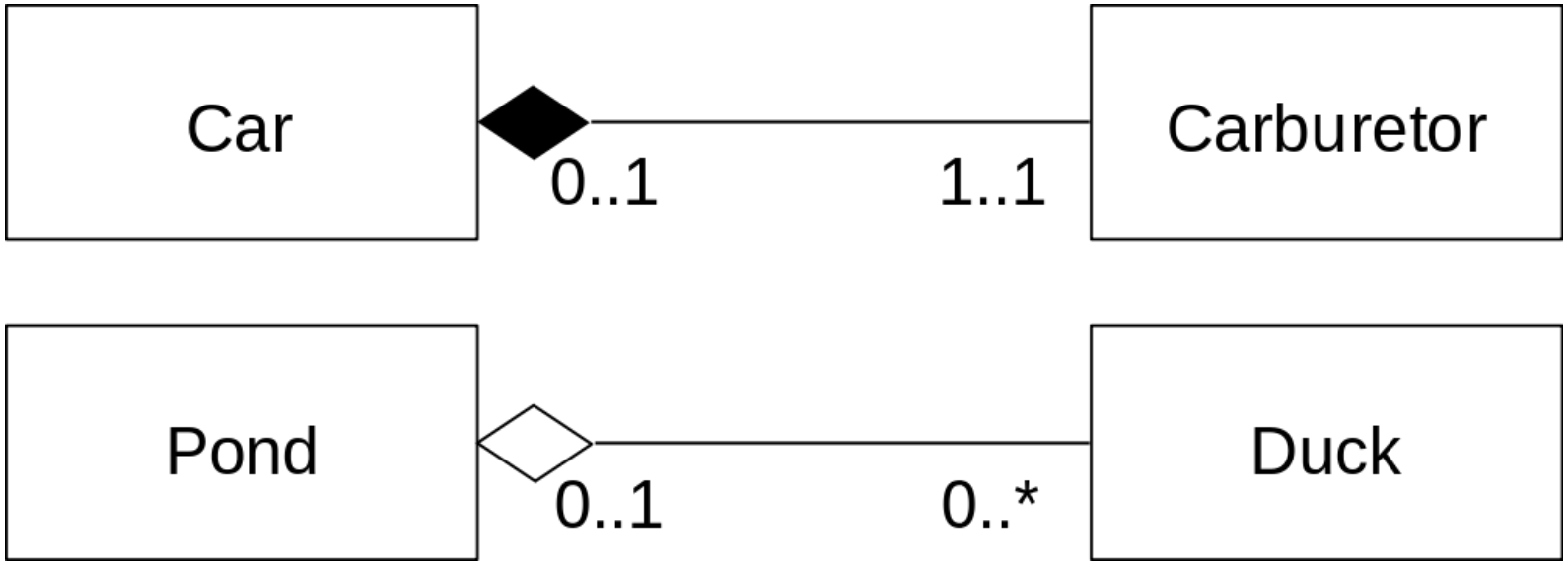
Must be translated to a context by the developer.

Composition >> Inheritance

Composition: Creating objects with other objects as members. Composition should be used when a **has-a**-relationship applies.

Inheritance: The concept of classes automatically containing the variables and methods defined in their supertypes.

Composition vs. Aggregation



https://en.wikipedia.org/wiki/Object_composition#UML_notation

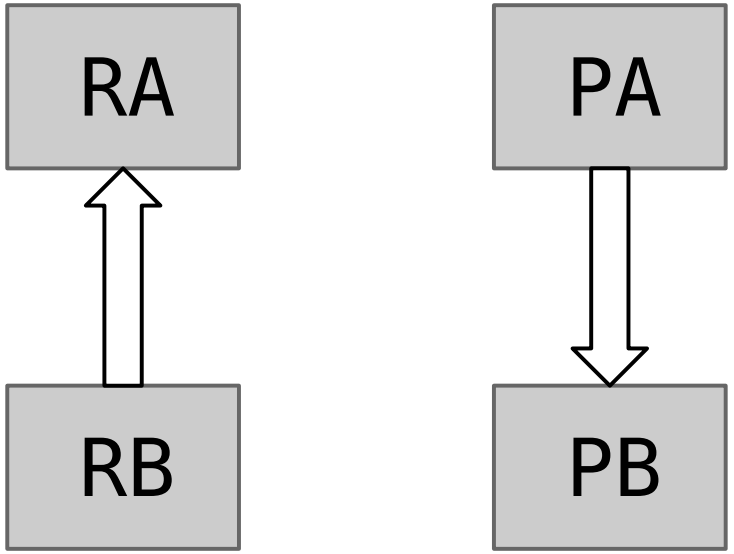
Liskov substitution principle

Subtypes should *behave* as their parent types.

Types just one aspect!

```
class A {
  RA m(PA p)…
}

class B extends A {
  RB m(PB p)…
}
```



Software Architecture

What is Software Architecture?

The conceptual fabric that defines a system

> **All architecture is design but not all design is architecture.**

Architecture: parts of a system that would be difficult to change once the system is built.

Architectures capture three primary dimensions:

- Structure
- Communication
- Nonfunctional requirements

Architecture

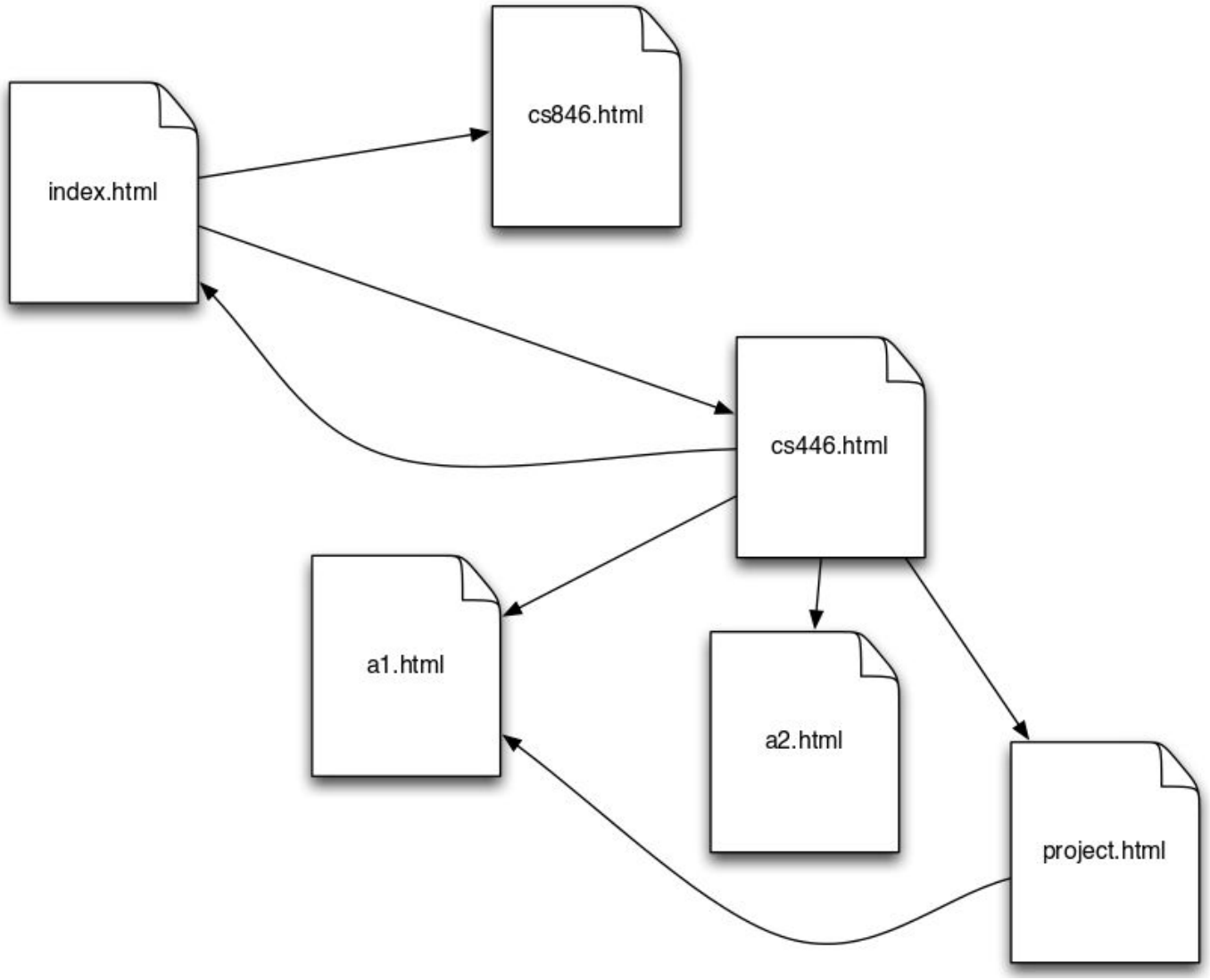
Architecture is:

- All about communication.
- What ‘parts’ are there?
- How do the ‘parts’ fit together?

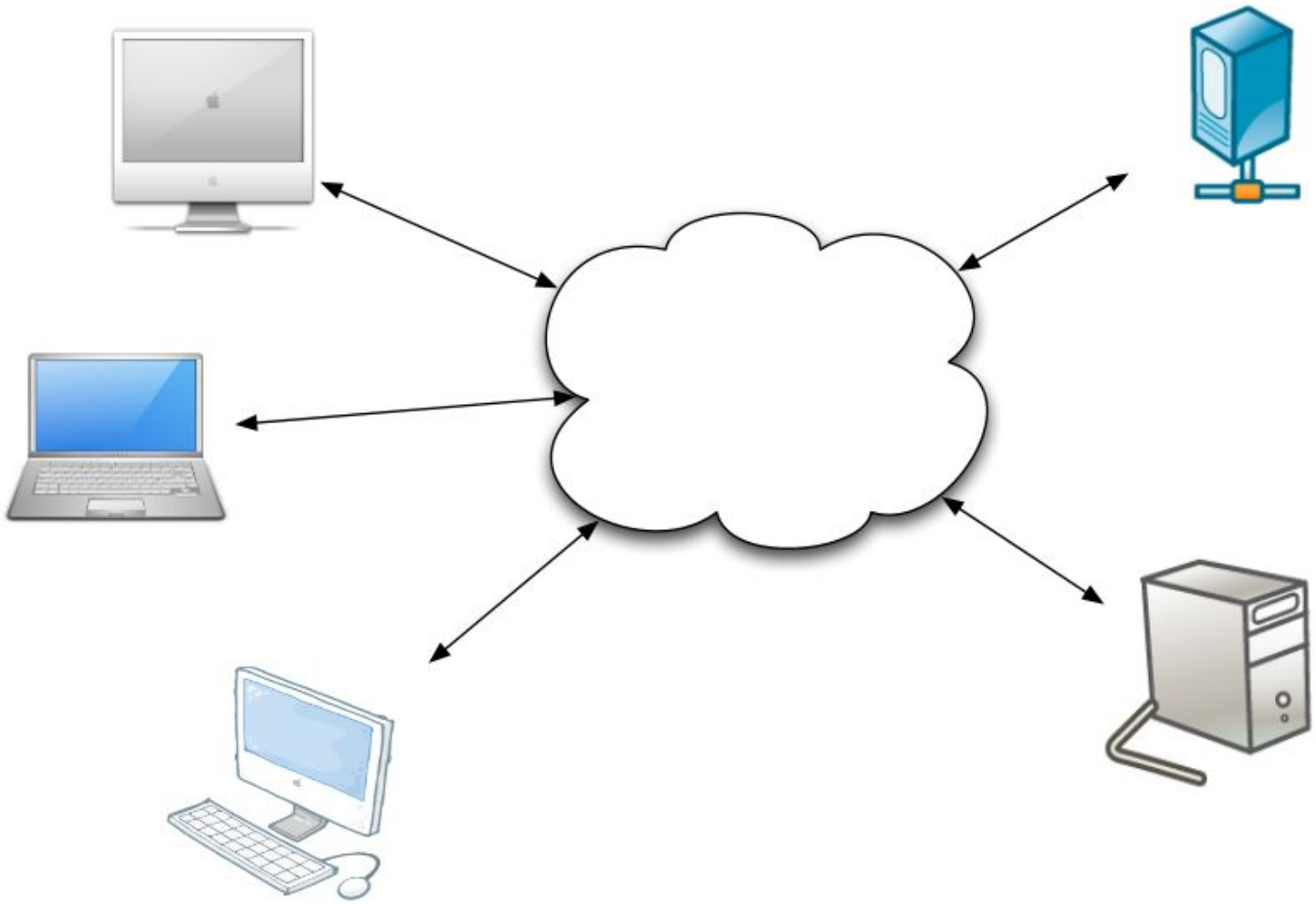
Architecture is not:

- About development
- About algorithms
- About data structures

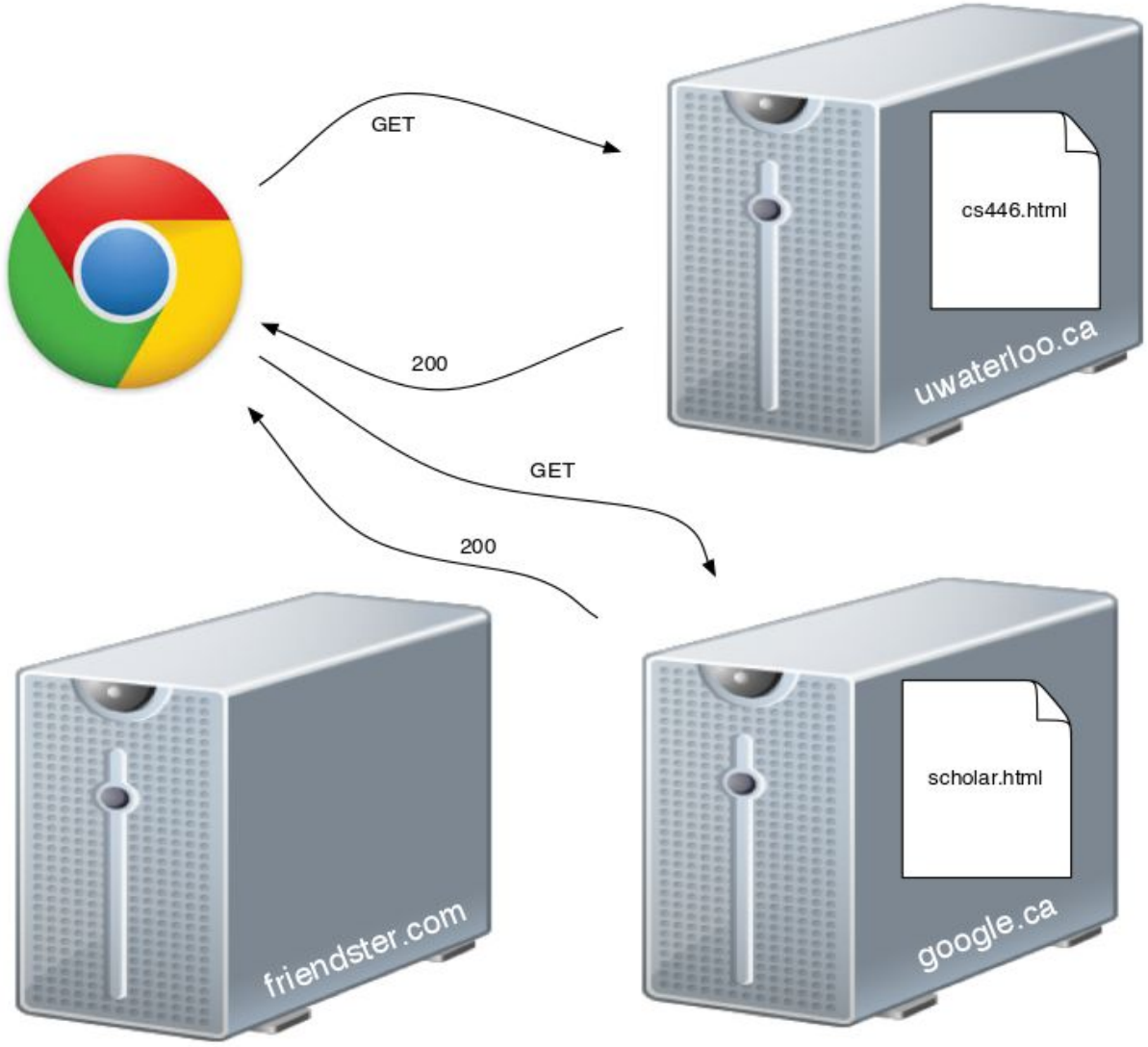
Logical Web Architecture



Physical Web Architecture



Dynamic Web Architecture



What is Software Architecture?

- **Definition**

  A software system's architecture is the set of *principal design decisions* about the system

- Software architecture is the blueprint for a software system's construction and evolution
- Design decisions encompass every facet of the system under development
  - Structure
  - Behavior
  - Interaction
  - Non-functional properties

Prescriptive vs. Descriptive Architecture

- A system's *prescriptive architecture* captures the design decisions made prior to the system's construction
  - It is the *as-conceived* or *as-intended* architecture

- A system's *descriptive architecture* describes how the system has been built
  - It is the *as-implemented* or *as-realized* architecture

Architectural Evolution

- When a system evolves, ideally its prescriptive architecture is modified first
- In practice, the system – and thus its descriptive architecture – is often directly modified
- This happens because of
  - Developer sloppiness
  - Perception of short deadlines which prevent thinking through and documenting
  - Lack of documented prescriptive architecture
  - Need or desire for code optimizations
  - Inadequate techniques or tool support

Architectural Degradation

- *Architectural drift* is the introduction of principal design decisions into a system's descriptive architecture that
  - are not included in, encompassed by, or implied by the prescriptive architecture
  - but which do not violate any of the prescriptive architecture's design decisions

- *Architectural erosion* is the introduction of architectural design decisions into a system's descriptive architecture that violate its prescriptive architecture

Architectural Recovery

- If architectural degradation is allowed to occur, one will be forced to *recover* the system's architecture sooner or later
- *Architectural recovery* is the process of determining a software system's architecture from its implementation-level artifacts
- Implementation-level artifacts can be
  - Source code
  - Executable files
  - Java .class files

Architecture Views

4 + 1 View Model

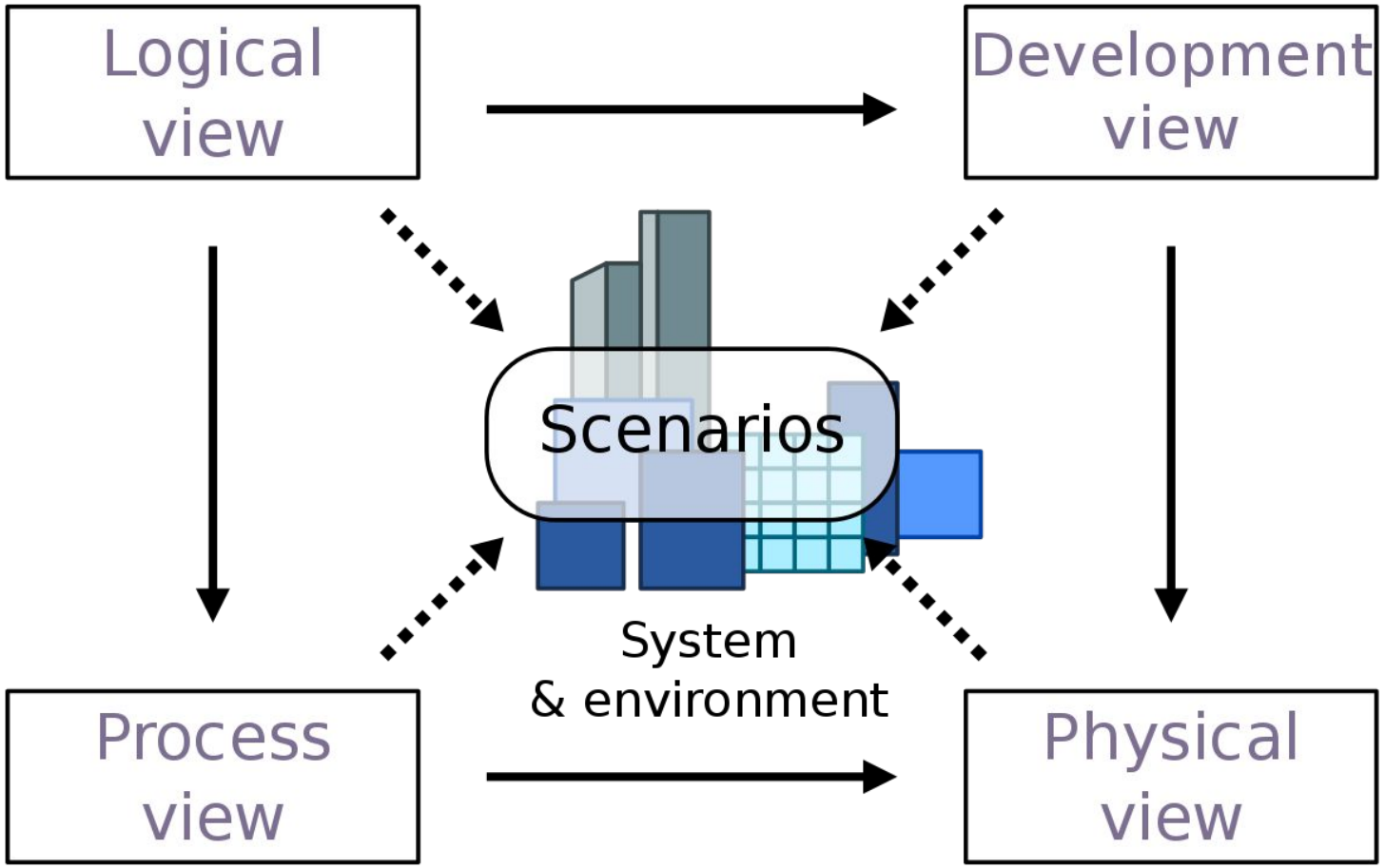
"Architectural Blueprints—The "4+1" View Model of Software Architecture"

Philippe Kruchten

Read:

http://www.cs.ubc.ca/~gregor/teaching/papers/4+1view-architecture.pdf

Ignore the concrete notation. Mostly replaced by UML.



https://en.wikipedia.org/wiki/4%2B1_architectural_view_model

Architectural Models, Views, and Visualizations

- Architecture Model

  An artifact documenting some or all of the architectural design decisions about a system

- Architecture Visualization

  A way of depicting some or all of the architectural design decisions about a system to a stakeholder

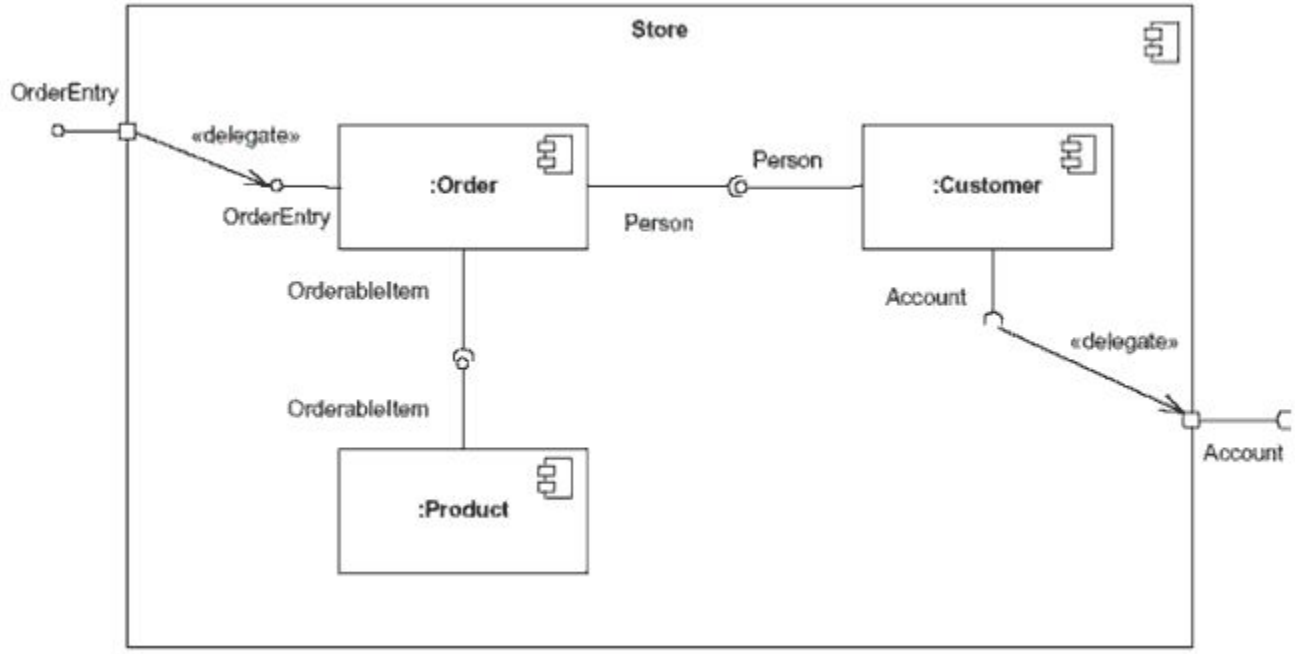
- Architecture View

  A subset of related architectural design decisions

Component diagram

Captures components and relationships.

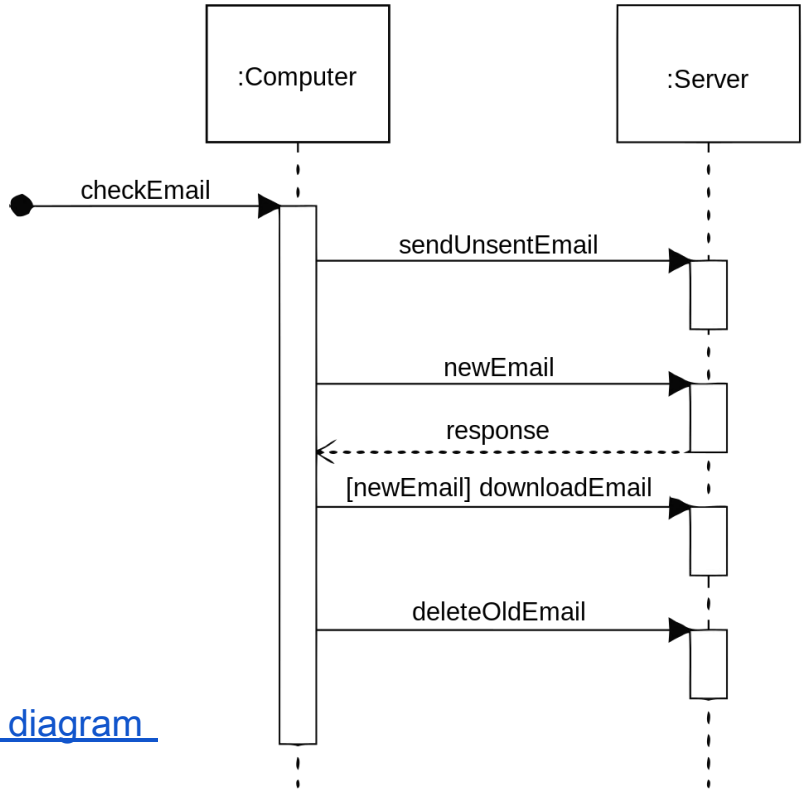
Required and provided APIs explicitly recorded.



IBM: http://www.ibm.com/developerworks/rational/library/dec04/bell/

Sequence diagram

Focus on inter-component collaboration.
Capture behaviour for specific runtime scenarios.

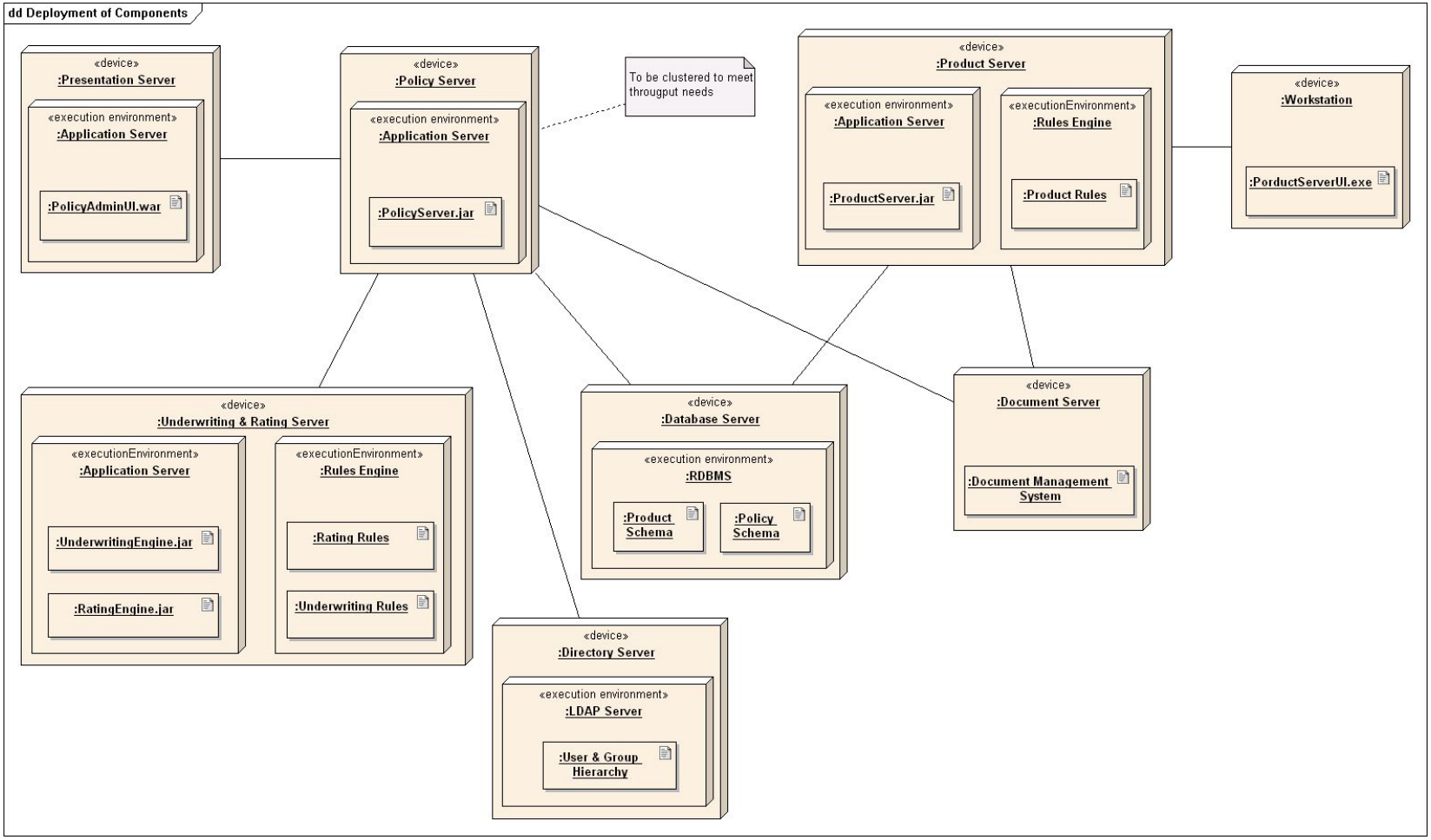


https://en.wikipedia.org/wiki/Sequence_diagram

Deployment diagram

Provide mapping between components and physical devices.

Deployment diagram



https://en.wikipedia.org/wiki/Deployment_diagram

Read & Watch

"Architectural Blueprints—The "4+1" View Model of Software Architecture", Philippe Kruchten

"The C4 Model for Software Architecture", Simon Brown. More at https://c4model.com/

Watch a video summary.

Architecture Patterns

Architectural Patterns

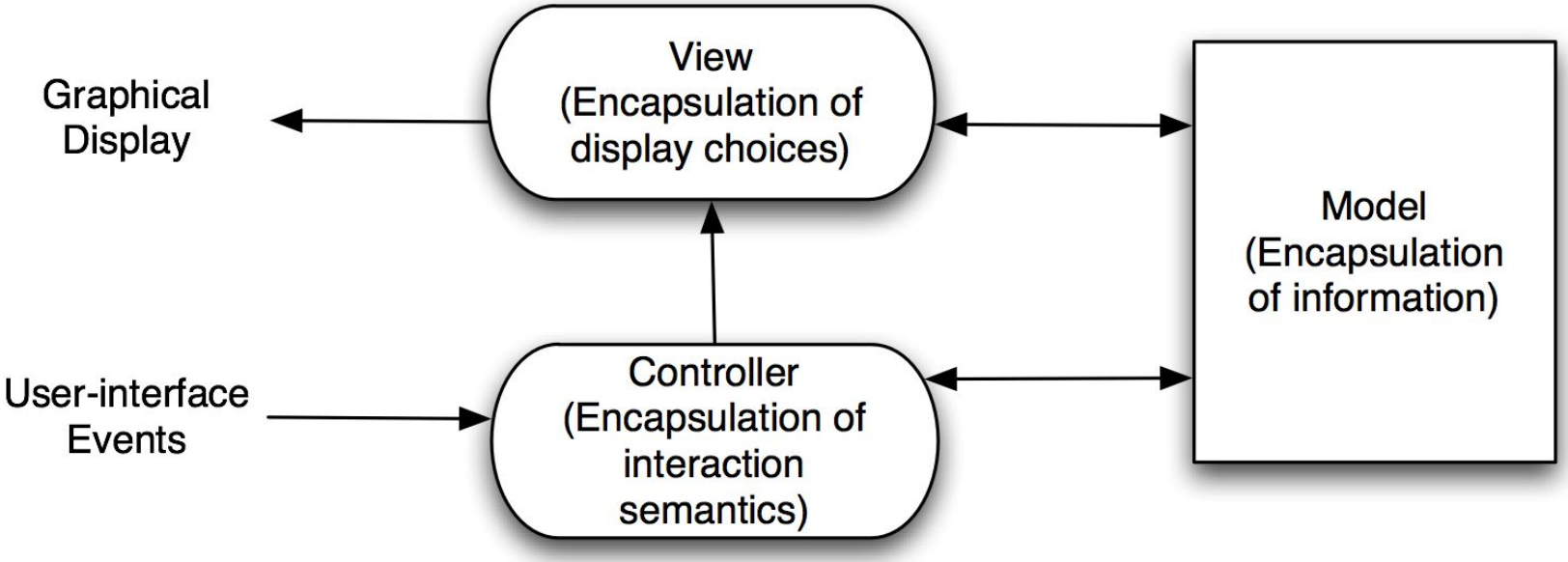
- **Definition**: An *architectural pattern* is a set of architectural design decisions that are applicable to a recurring design problem, and parameterized to account for different software development contexts in which that problem appears

- A widely used pattern in modern distributed systems is the *three-tiered system* pattern
  - Science
  - Banking
  - E-commerce
  - Reservation systems

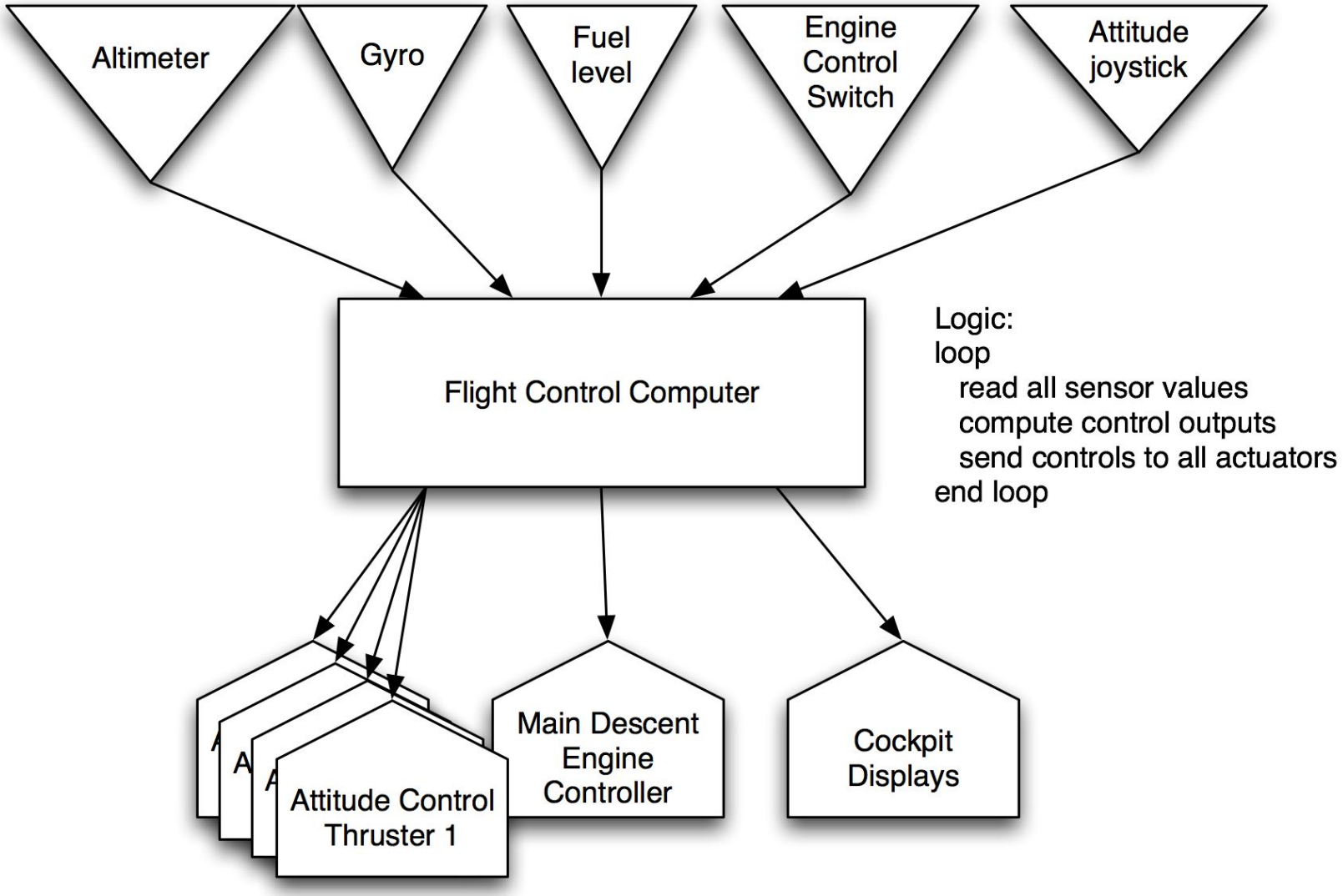
Three-Tiered Pattern

- Front Tier
  - Contains the user interface functionality to access the system's services
- Middle Tier
  - Contains the application's major functionality
- Back Tier
  - Contains the application's data access and storage functionality

Model-View-Controller



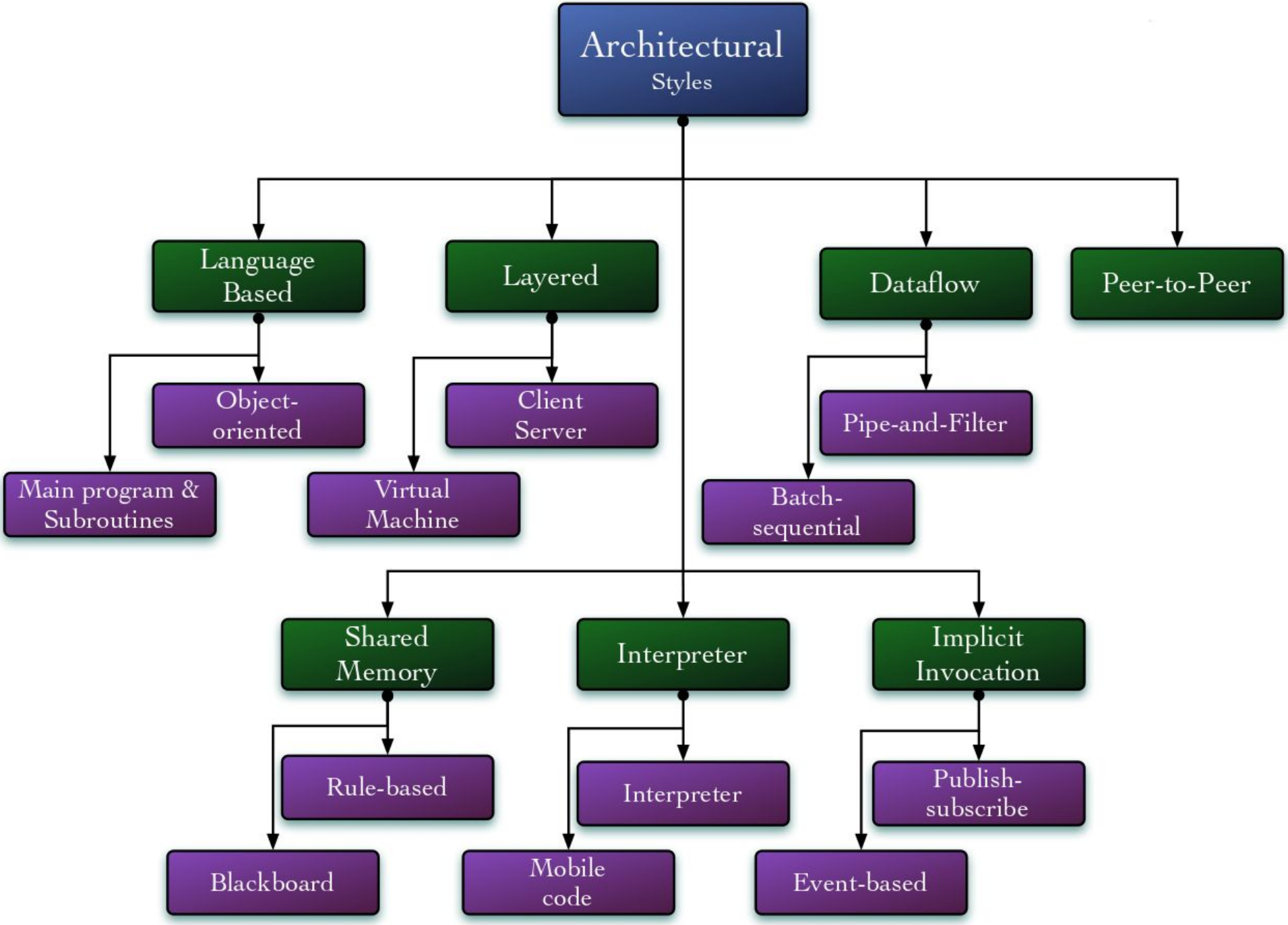
Sense-Compute-Control



Architectural Styles

- Certain design choices regularly result in solutions with superior properties
  - Compared to other possible alternatives, solutions such as this are more elegant, effective, efficient, dependable, evolvable, scalable, and so on

- **Definition**: An *architectural style* is a named collection of architectural design decisions that
  - are applicable in a given development context
  - constrain architectural design decisions that are specific to a particular system within that context
  - elicit beneficial qualities in each resulting system


Architectural Styles
Language Based
Layered
Dataflow
Peer-to-Peer
Object-oriented
Client Server
Pipe-and-Filter
Main program & Subroutines
Virtual Machine
Batch-sequential
Shared Memory
Interpreter
Implicit Invocation
Rule-based
Interpreter
Publish-subscribe
Blackboard
Mobile code
Event-based