

# CEN 5016: Software Engineering

Fall 2025

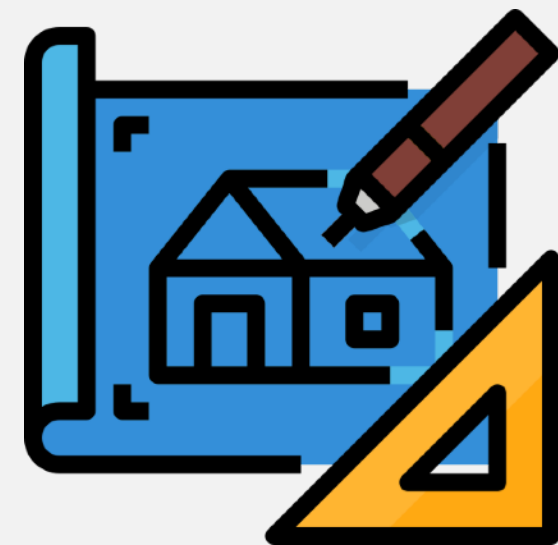


University of  
Central Florida

---

Dr. Kevin Moran

## *Week 5 - Class 1:* Introduction to Software Architecture





- *SDE Project Part 1*
  - Will be Posted to Course Webpage today
  - Due on Tuesday, Sept. 23rd
- *Assignment 3*
  - Posted to Course Webpage
  - Also Due on Tuesday Sept. 23rd
- Course webpage will be fully up later this evening, watch out for announcements on Ed Discussions

# Intro to Software Architecture





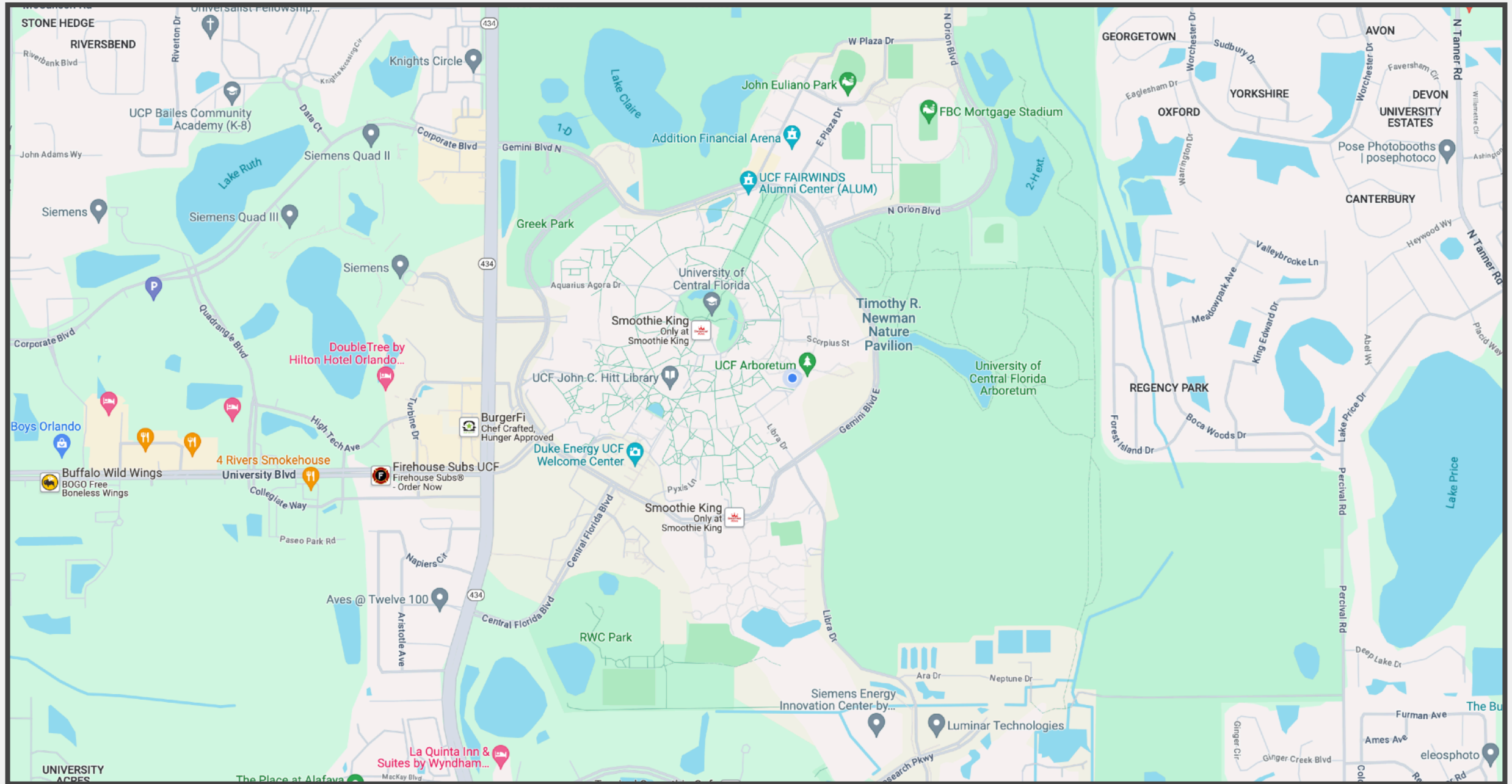
- Understand the abstraction level of architectural reasoning
- Appreciate how software systems can be viewed at different abstraction levels
- Distinguish software architecture from (object-oriented) software design
- Use notation and views to describe the architecture suitable to the purpose
- Document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



# Views and Abstraction

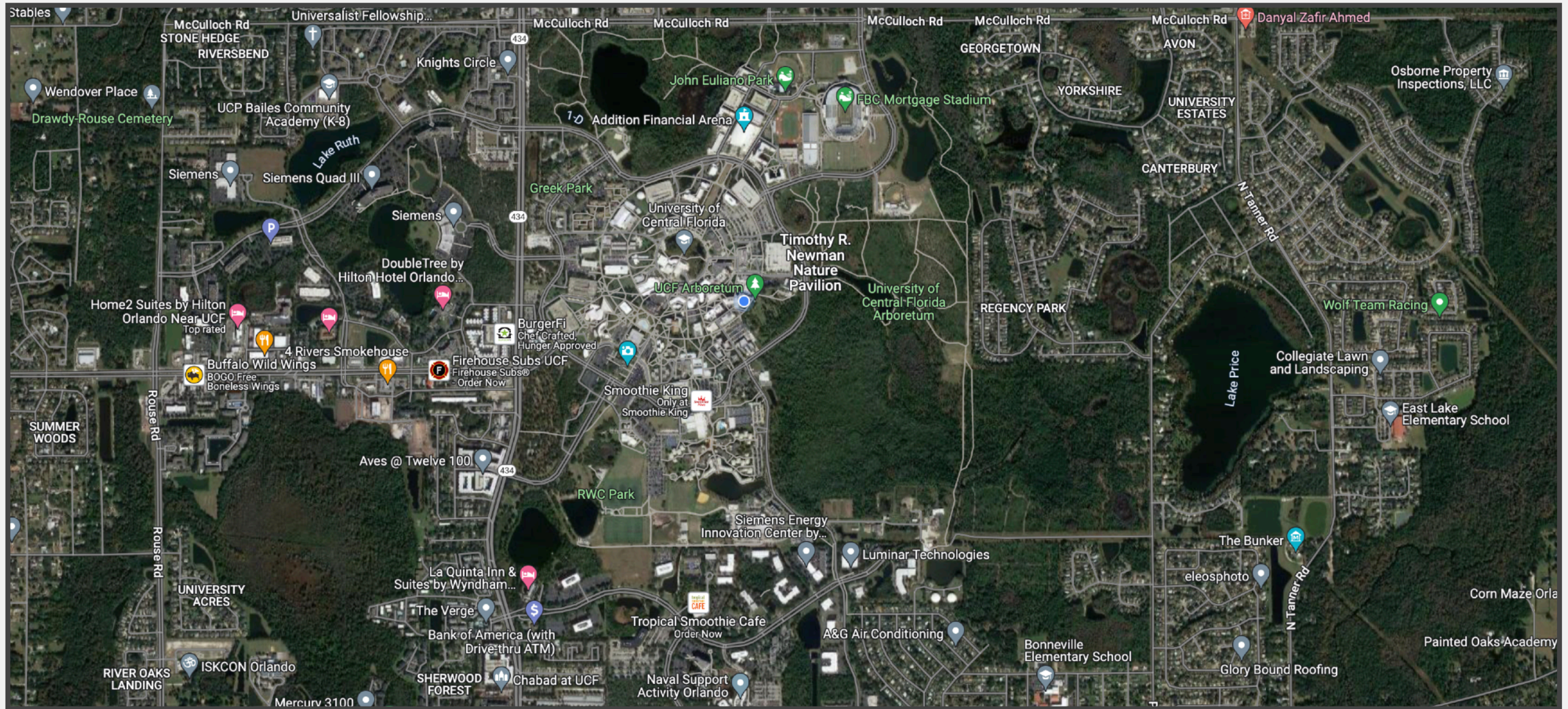


# Views & Abstraction



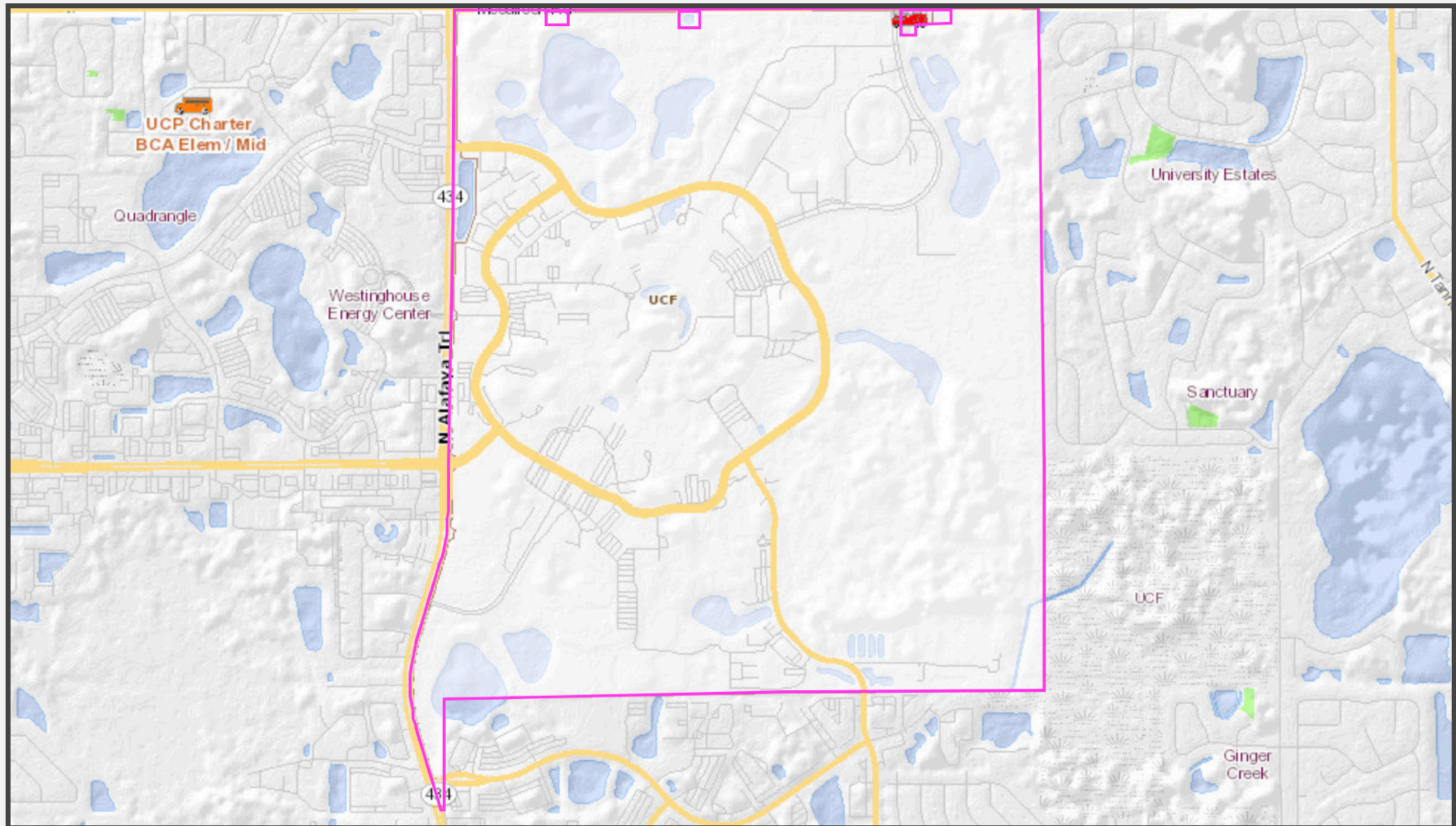


# Views & Abstraction





# Views & Abstraction



# Abstracted Views Focus on Conveying Information



- They have a well-defined purpose
- Show only necessary information
- Abstract away unnecessary details
- Use legends/annotations to remove ambiguity
- Multiple views of the same object tell a larger story



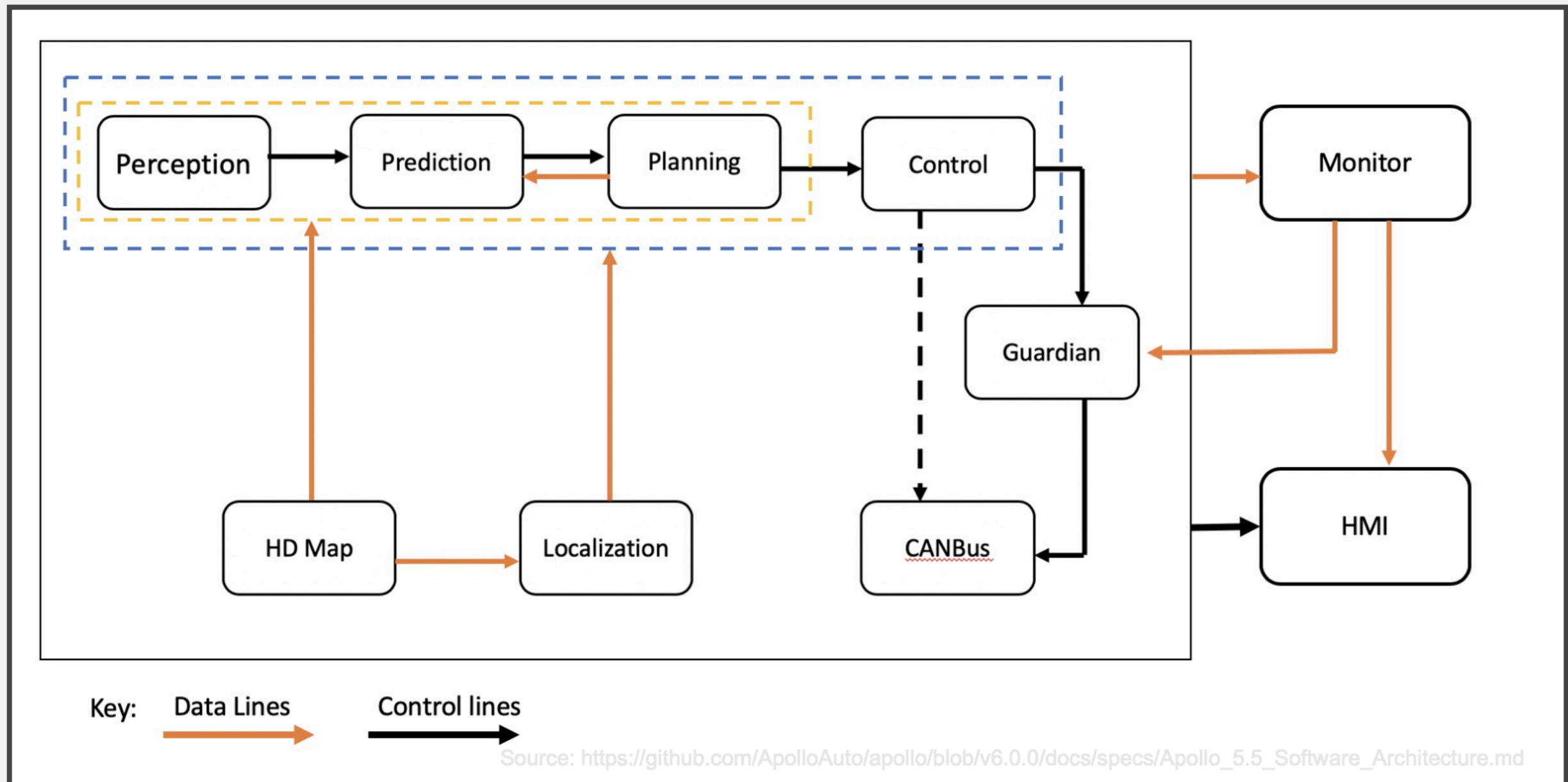
# Software Architecture Case Study: Autonomous Vehicles





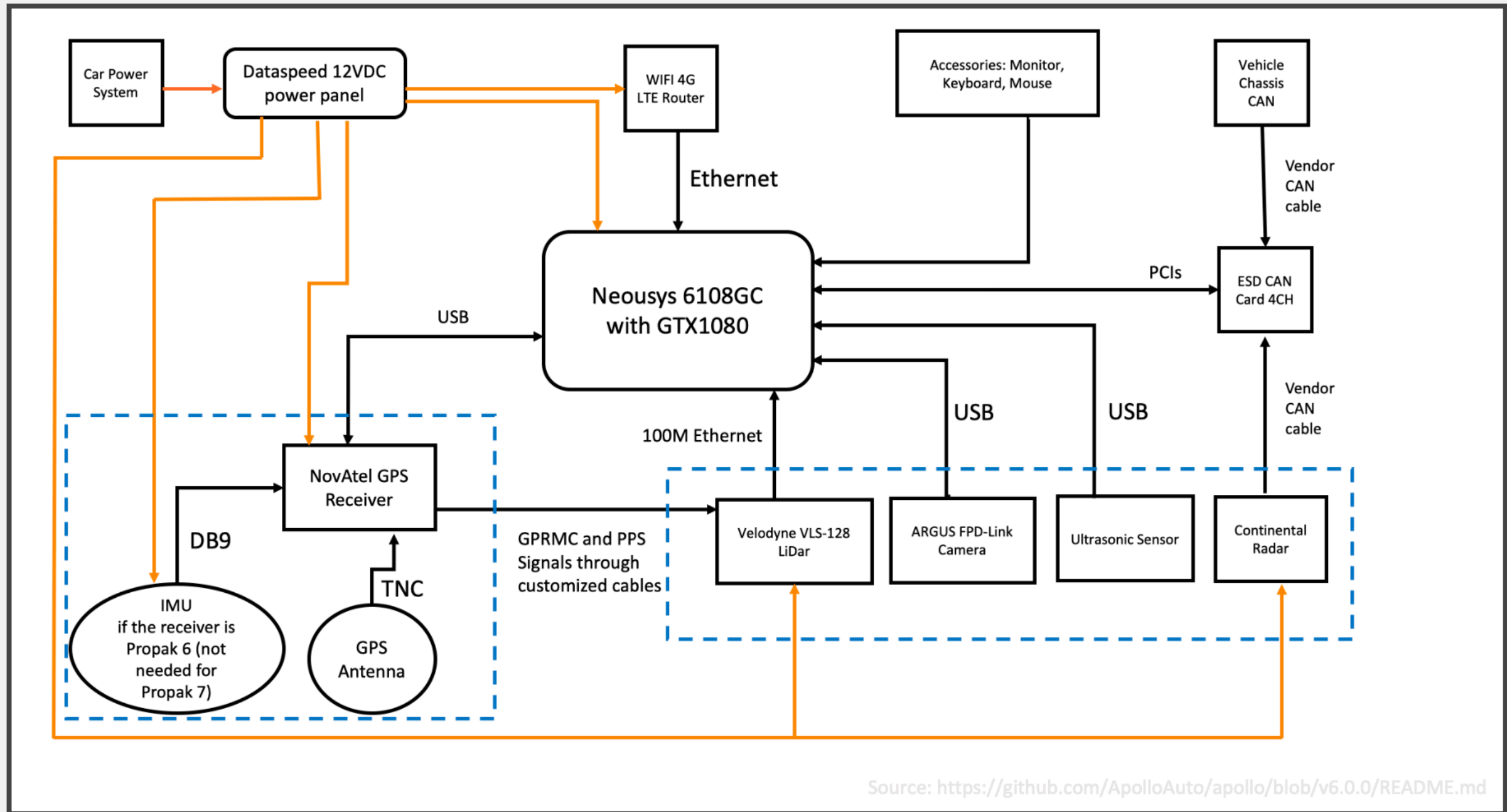
- Check out the “side pass” feature from the video:
  - <http://tinyurl.com/cen24-vid>
- **Source:** <https://github.com/ApolloAuto/apollo>
- **Doxygen:** <https://hidetoshi-furukawa.github.io/apollo/doxygen/index.html>

# Apollo Software Architecture

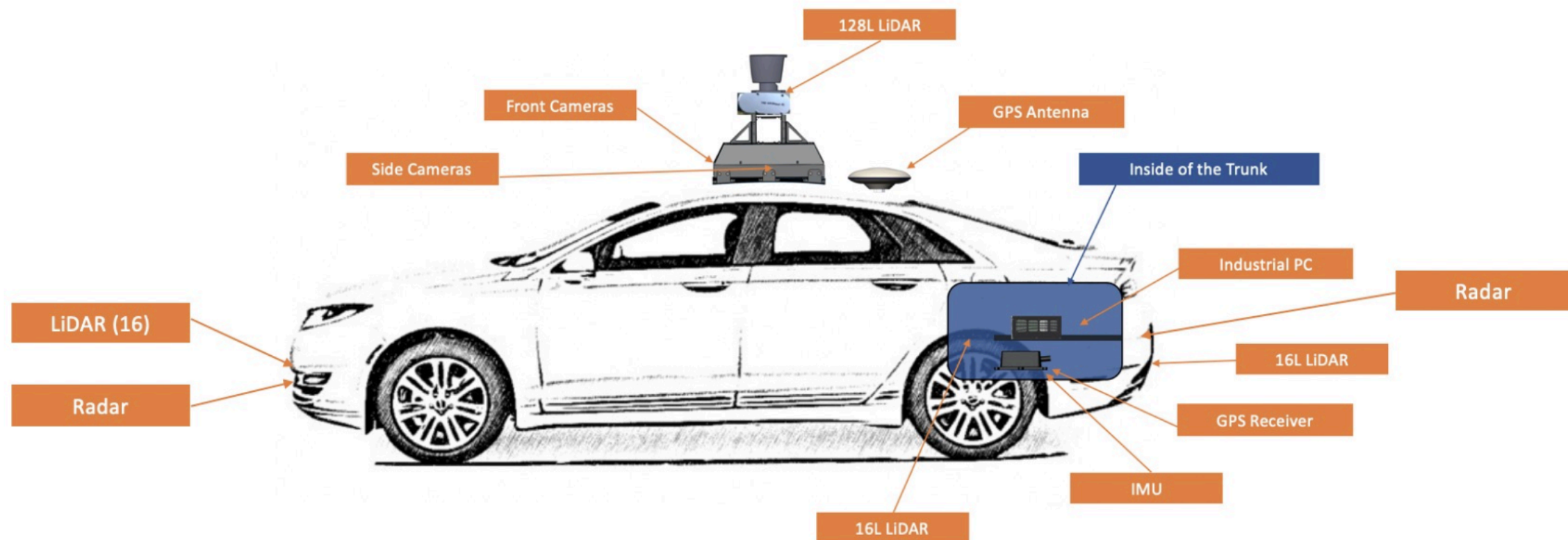




# Apollo Hardware Architecture

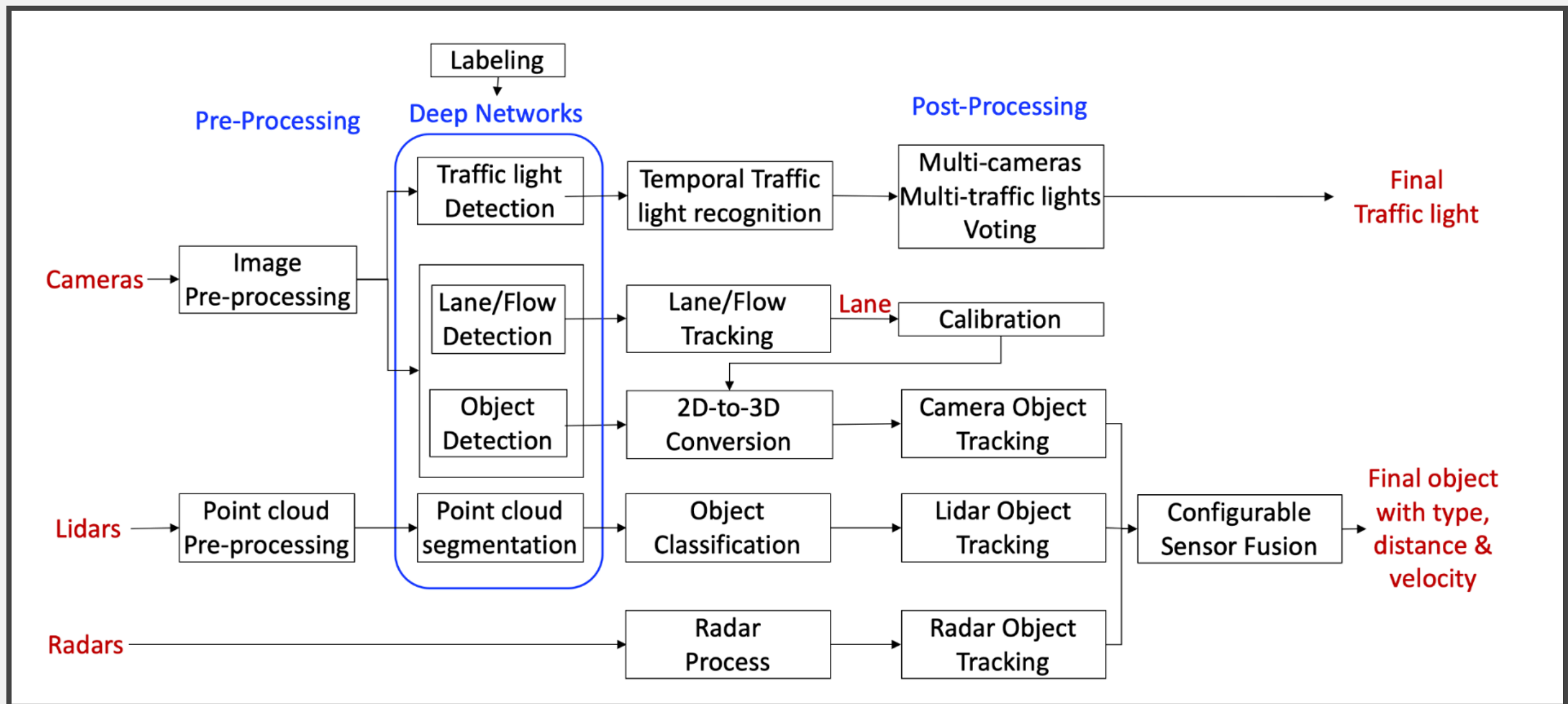


# Apollo Hardware/Vehicle Overview

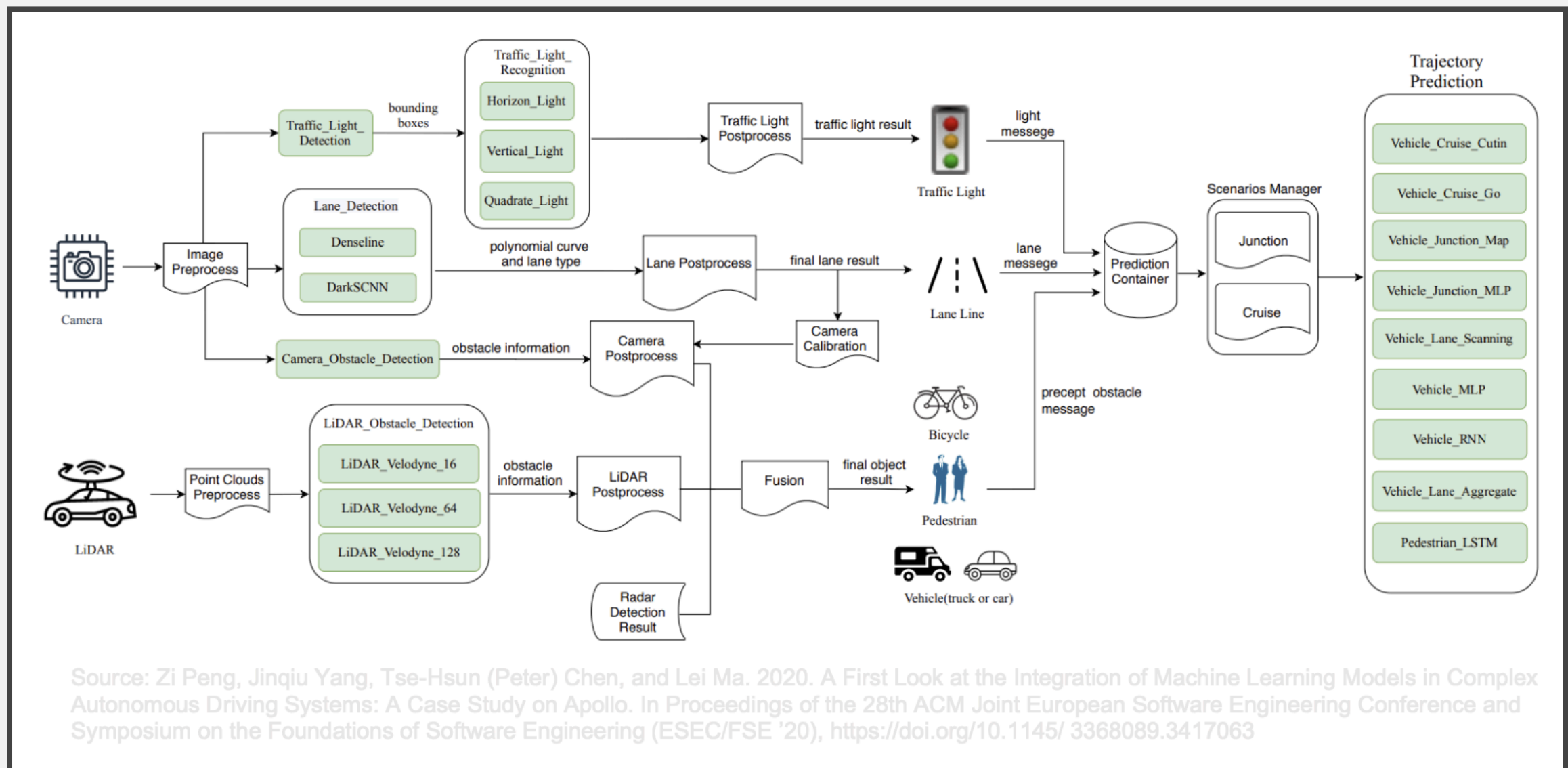


Source: <https://github.com/ApolloAuto/apollo/blob/v6.0.0/README.md>

# Apollo Perception Module



# Apollo ML Models



Source: Zi Peng, Jinqiu Yang, Tse-Hsun (Peter) Chen, and Lei Ma. 2020. A First Look at the Integration of Machine Learning Models in Complex Autonomous Driving Systems: A Case Study on Apollo. In Proceedings of the 28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '20), <https://doi.org/10.1145/3368089.3417063>

# Apollo Software Stack

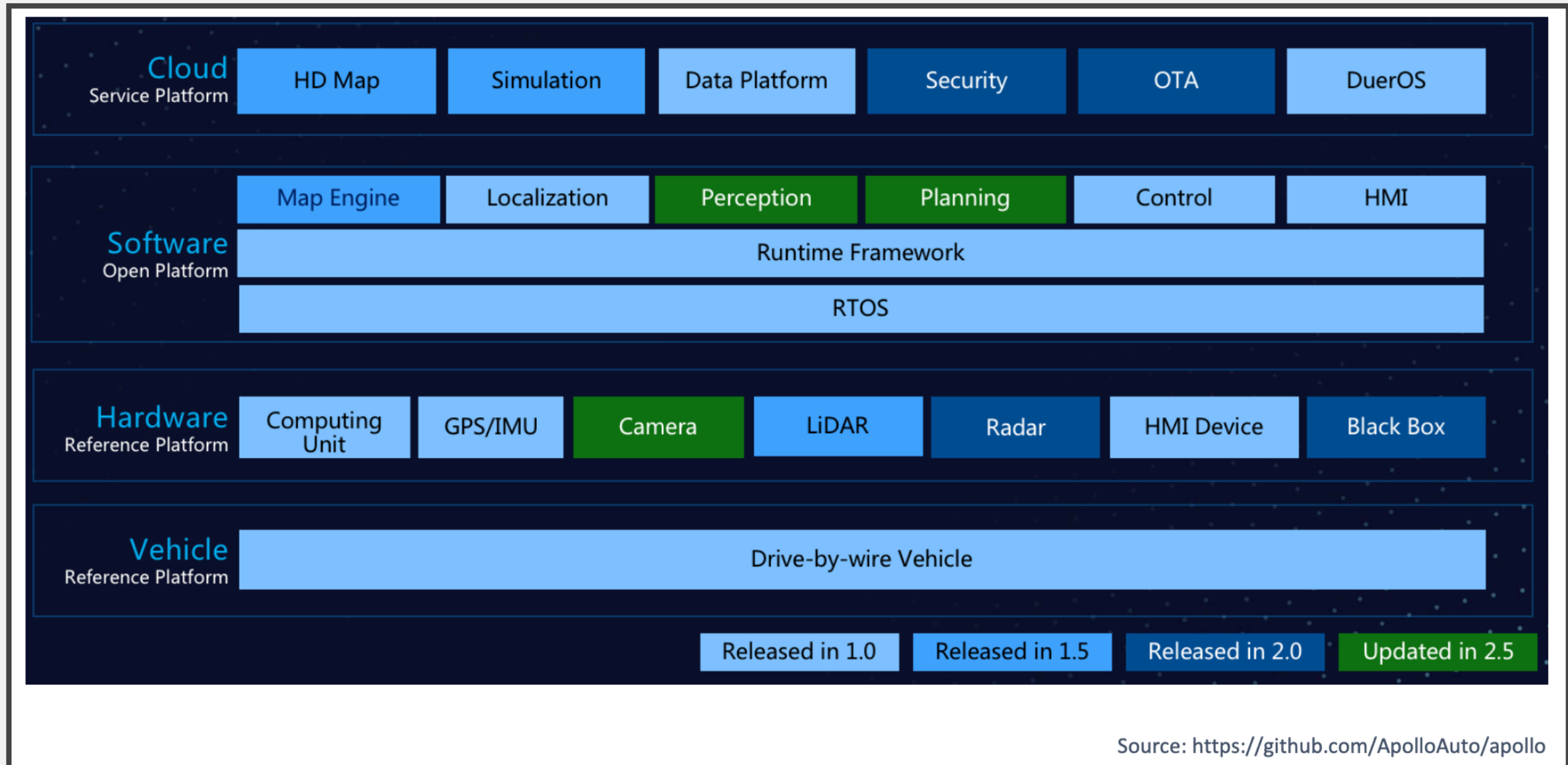


Cloud Service Platform	HD Map	Simulation		Data Platform		Security		OTA		DuerOS		Volume Production Service Components	V2X Roadside Service
Open Software Platform	Map Engine	Localization		Perception		Planning		Control		End-to-End		HMI	V2X Adapter
	Apollo Cyber RT Framework												
	RTOS												
Hardware Development Platform	Computing Unit	GPS/IMU	Camera	LiDAR	Radar	Ultrasonic Sensor	HMI Device	Black Box	Apollo Sensor Unit	Apollo Extension Unit		V2X OBU	
Open Vehicle Certificate Platform	Certified Apollo Compatible Drive-by-wire Vehicle								Open Vehicle Interface Standard				

Major Updates in Apollo 3.5

Source: <https://github.com/ApolloAuto/>

# Feature Evolution (Software Stack View)





*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!

# 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

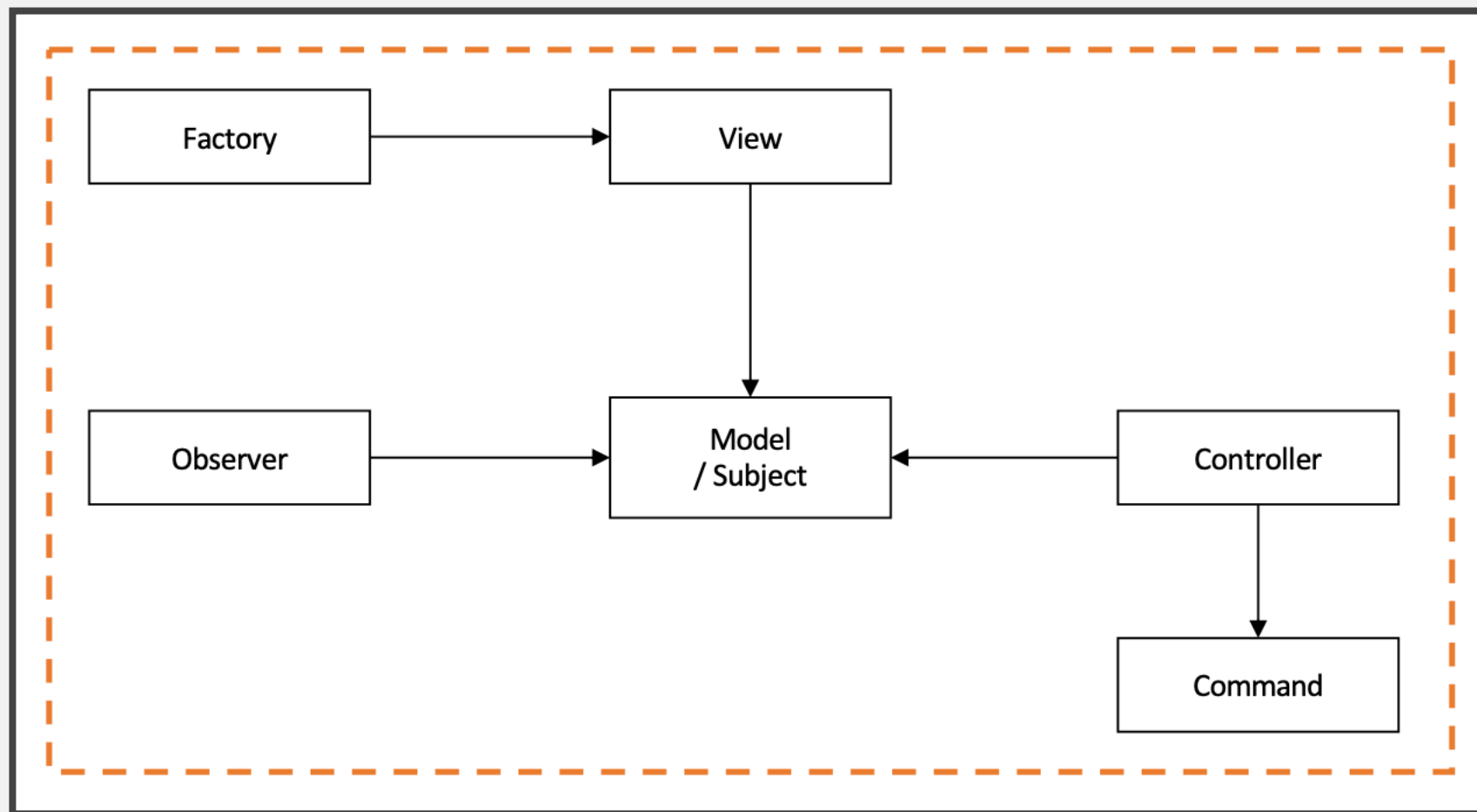
- *How do I extend VSCode with a plugin?*
- *What threads exist and how do they coordinate?*
- *How does Google scale to billions of hits per day?*
- *Where should I put my firewalls?*
- *What is the interface between subsystems?*

# Objects

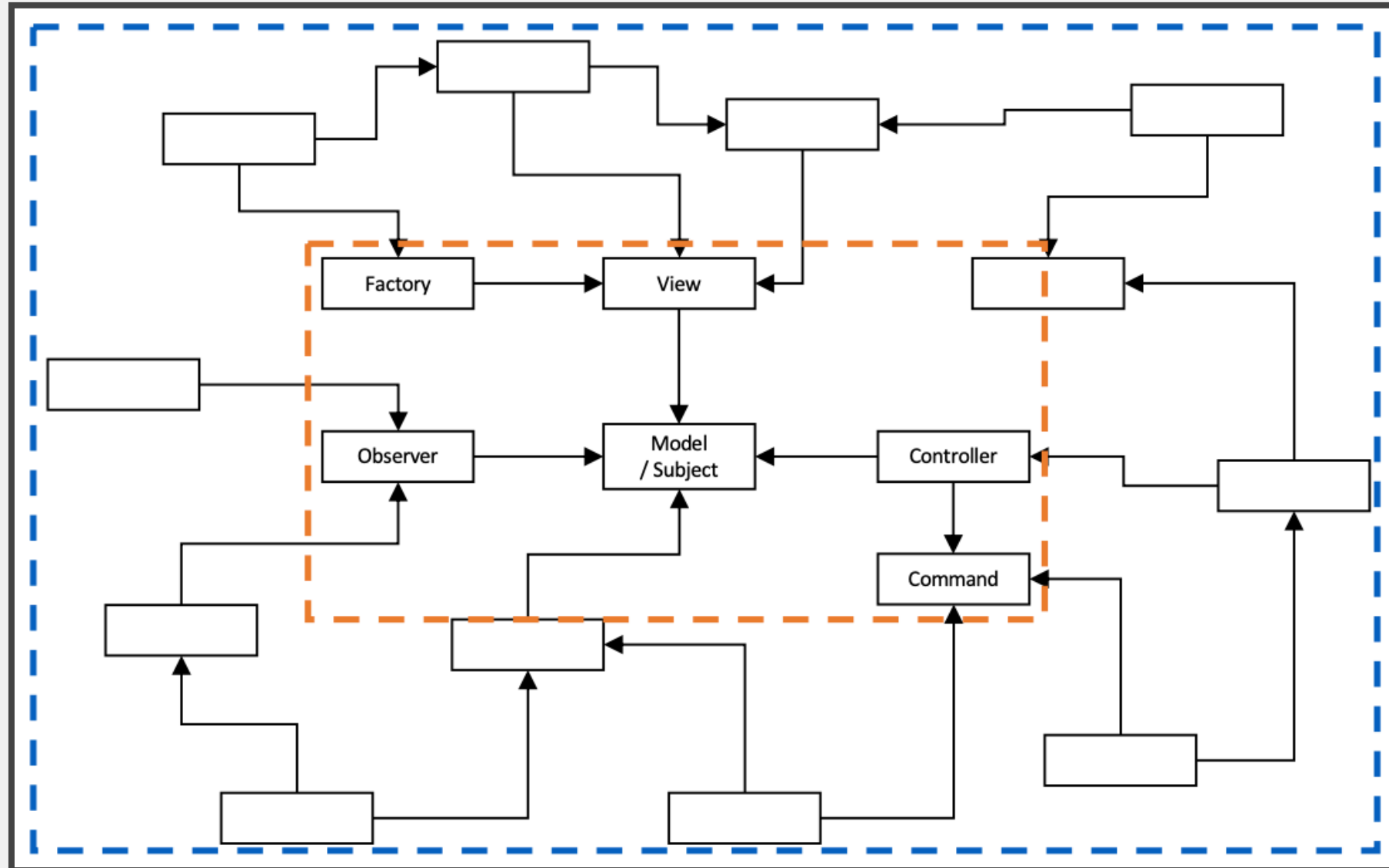


Model

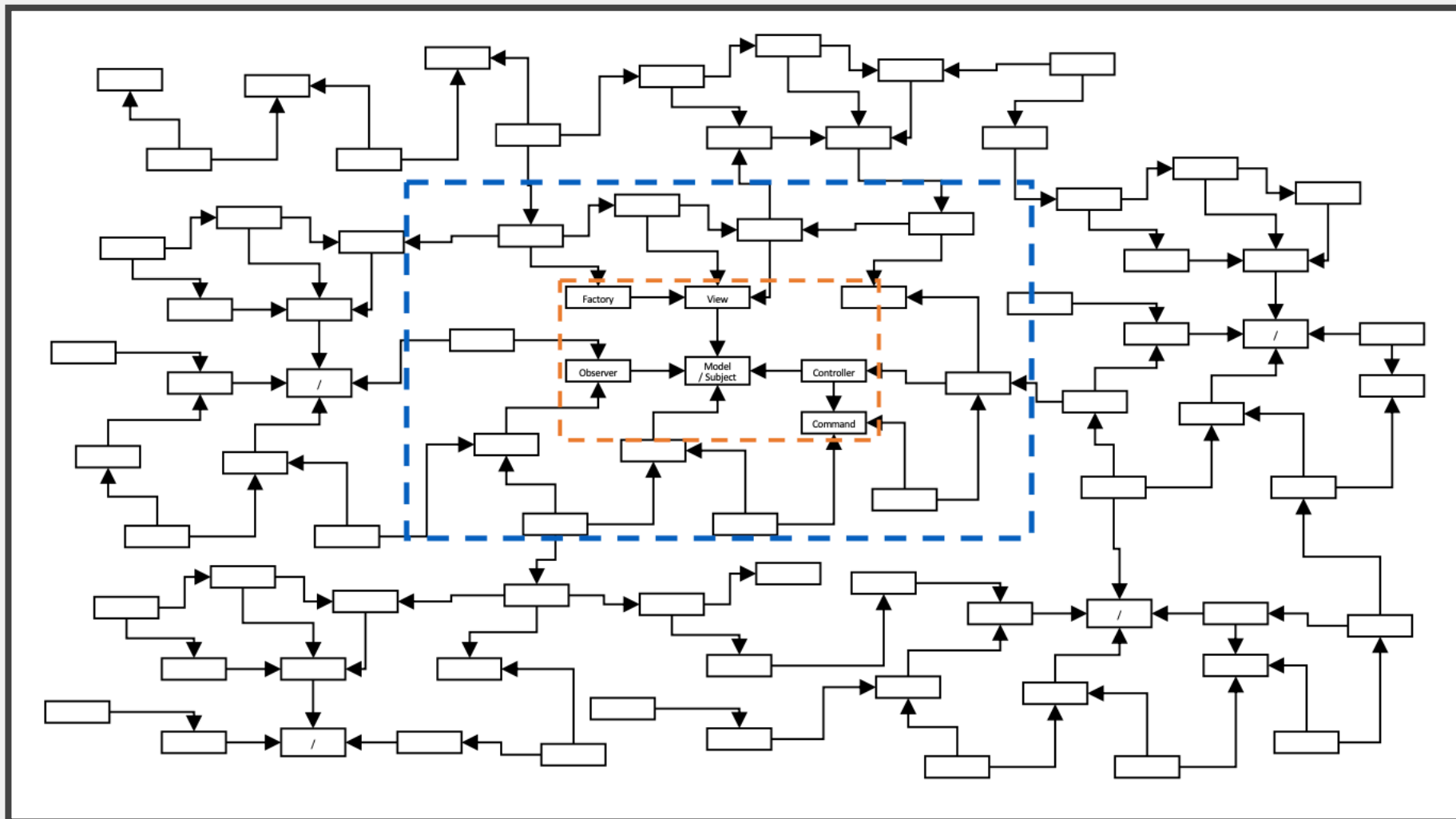
# Design Patterns

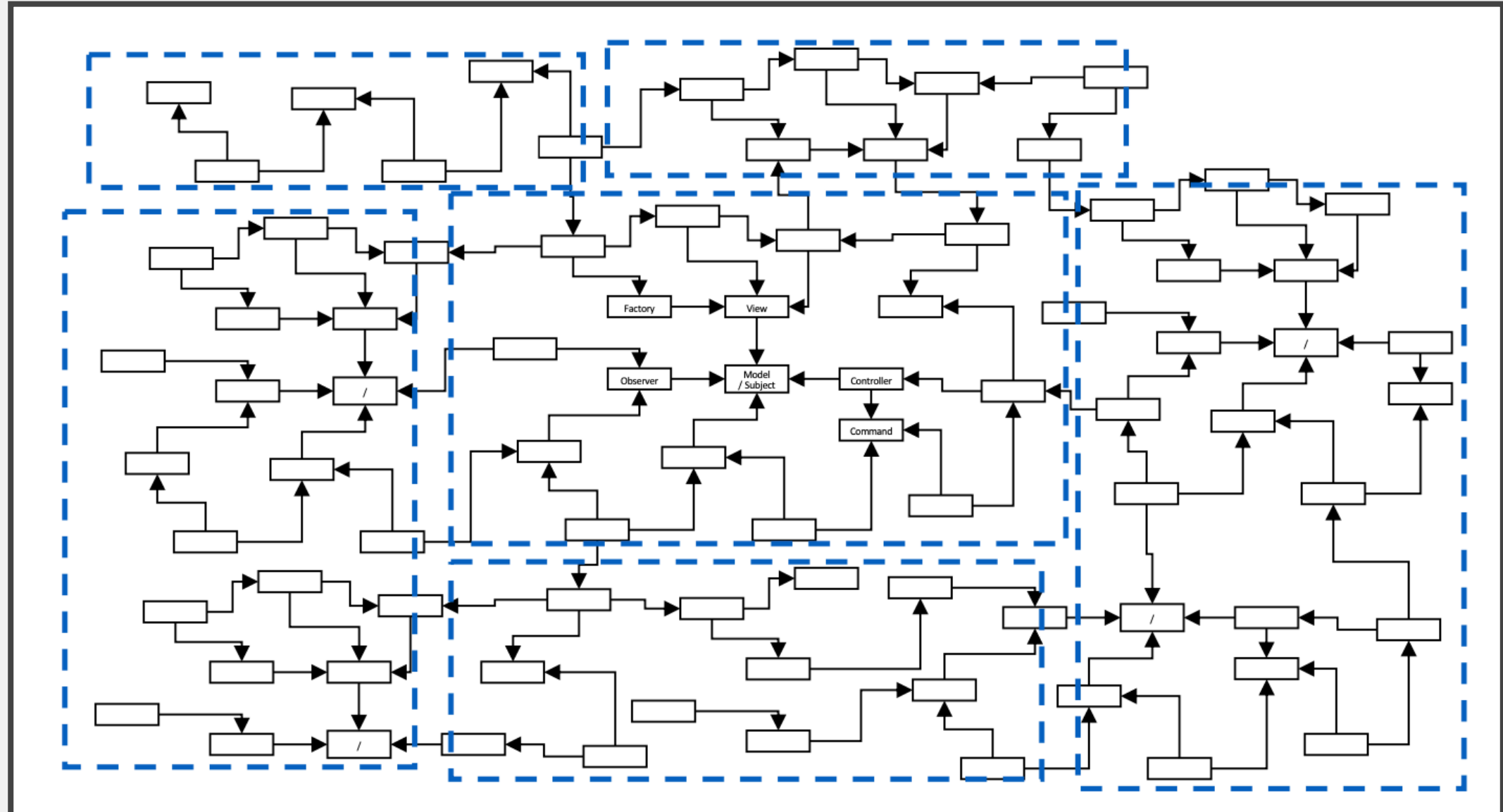


# Design Patterns

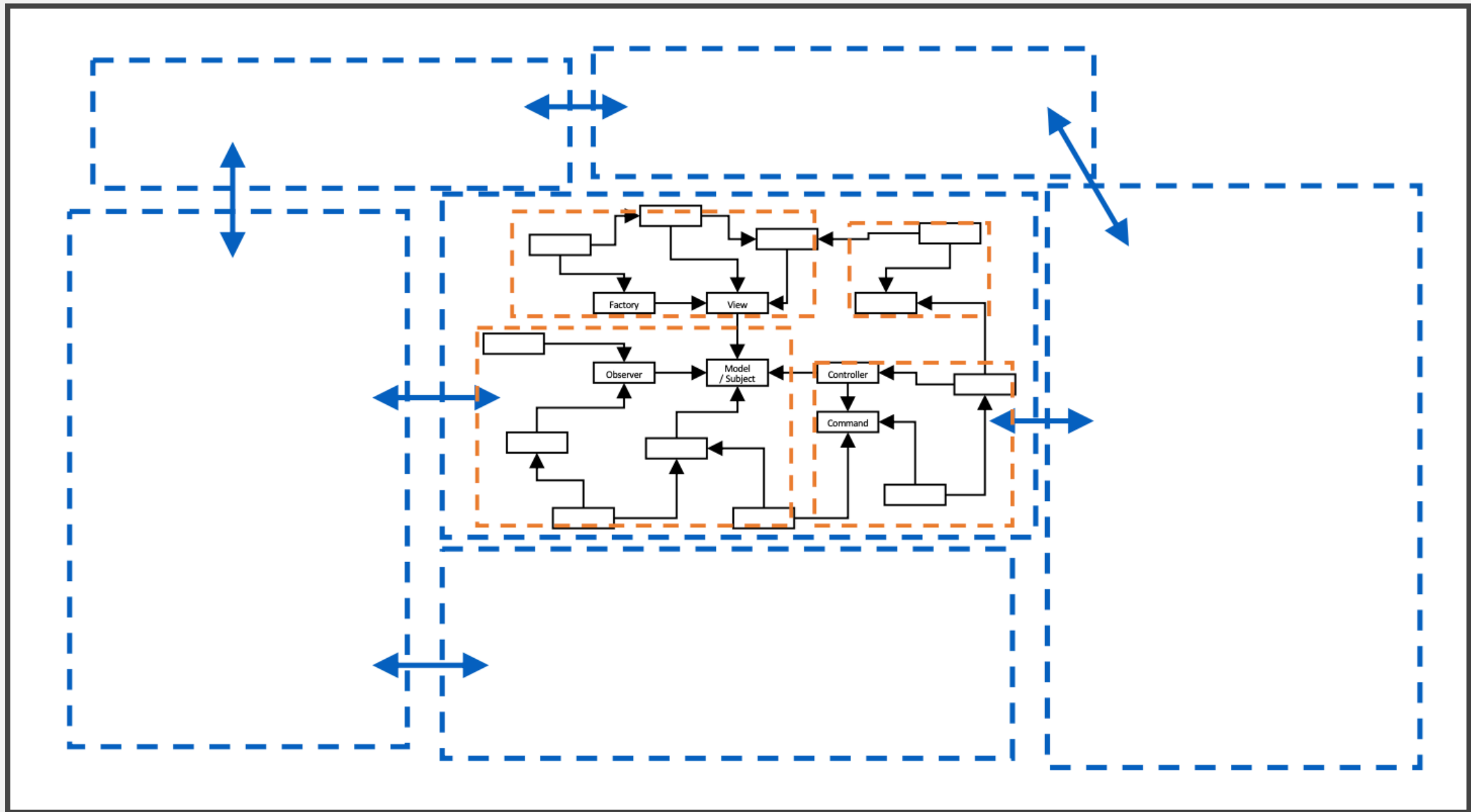


# Design Patterns

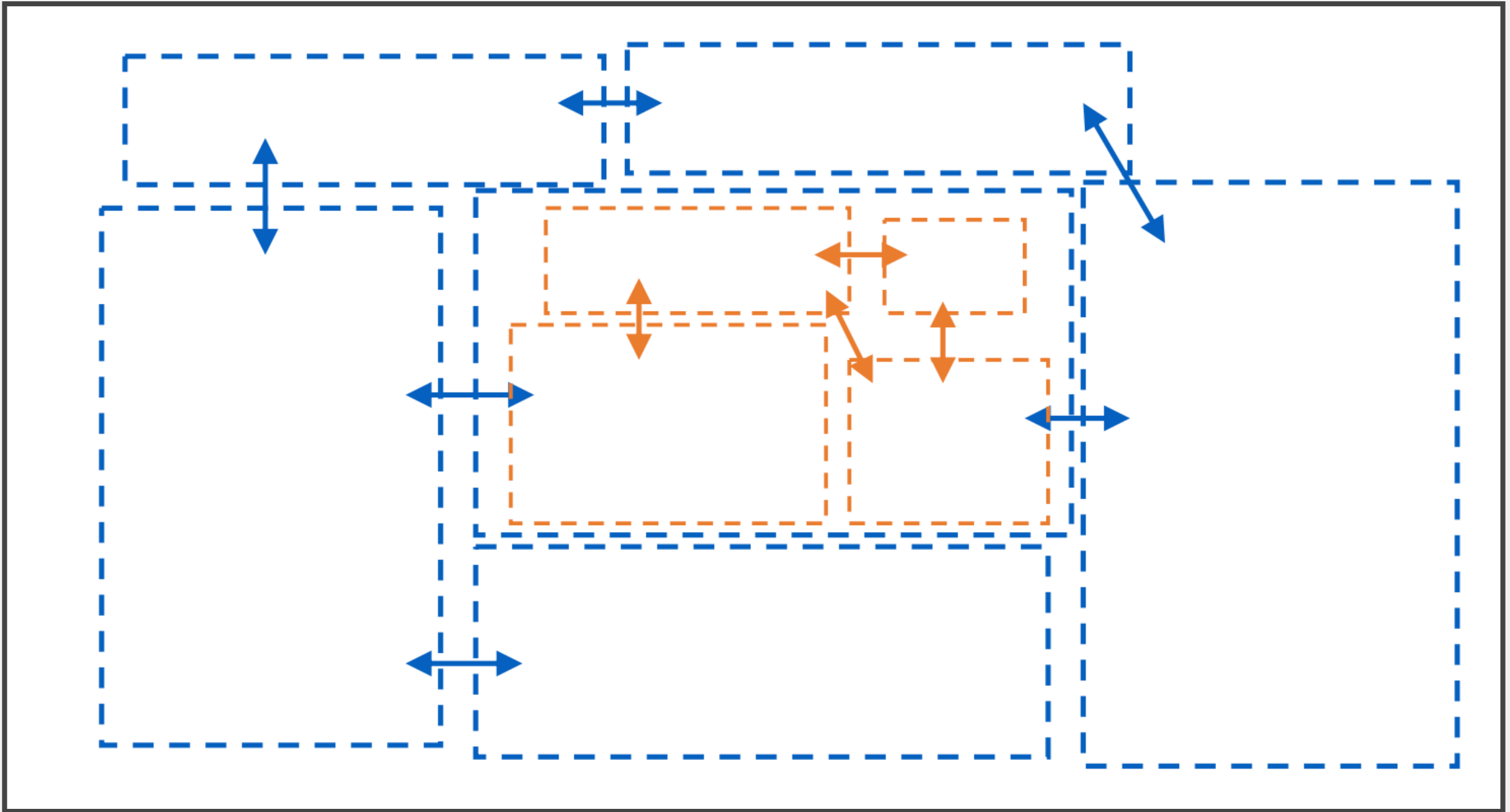




# Architecture







# 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 & 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
  - ...

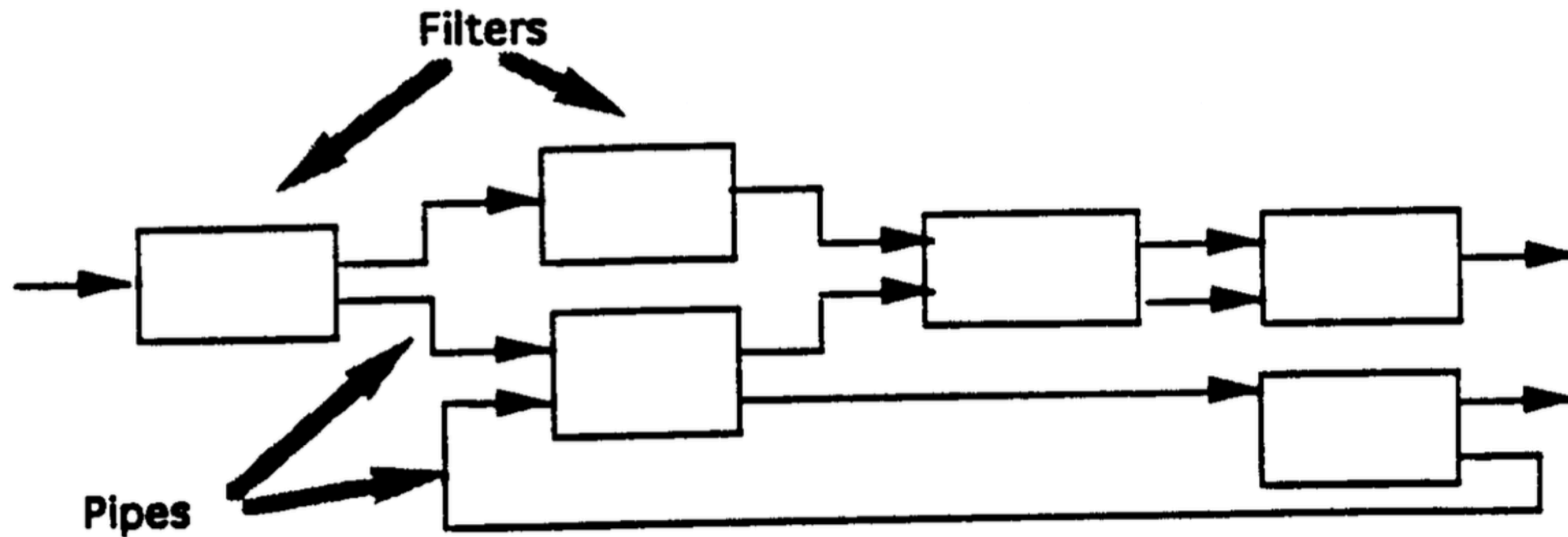


- Static View
  - Modules (subsystems, structures) and their relations (dependencies, ...)
- Dynamic View
  - Components (processes, runnable entities) and connectors (messages, data flow, ...)
- Physical View (Deployment)
  - Hardware structures and their connections

# Common Software Architectures

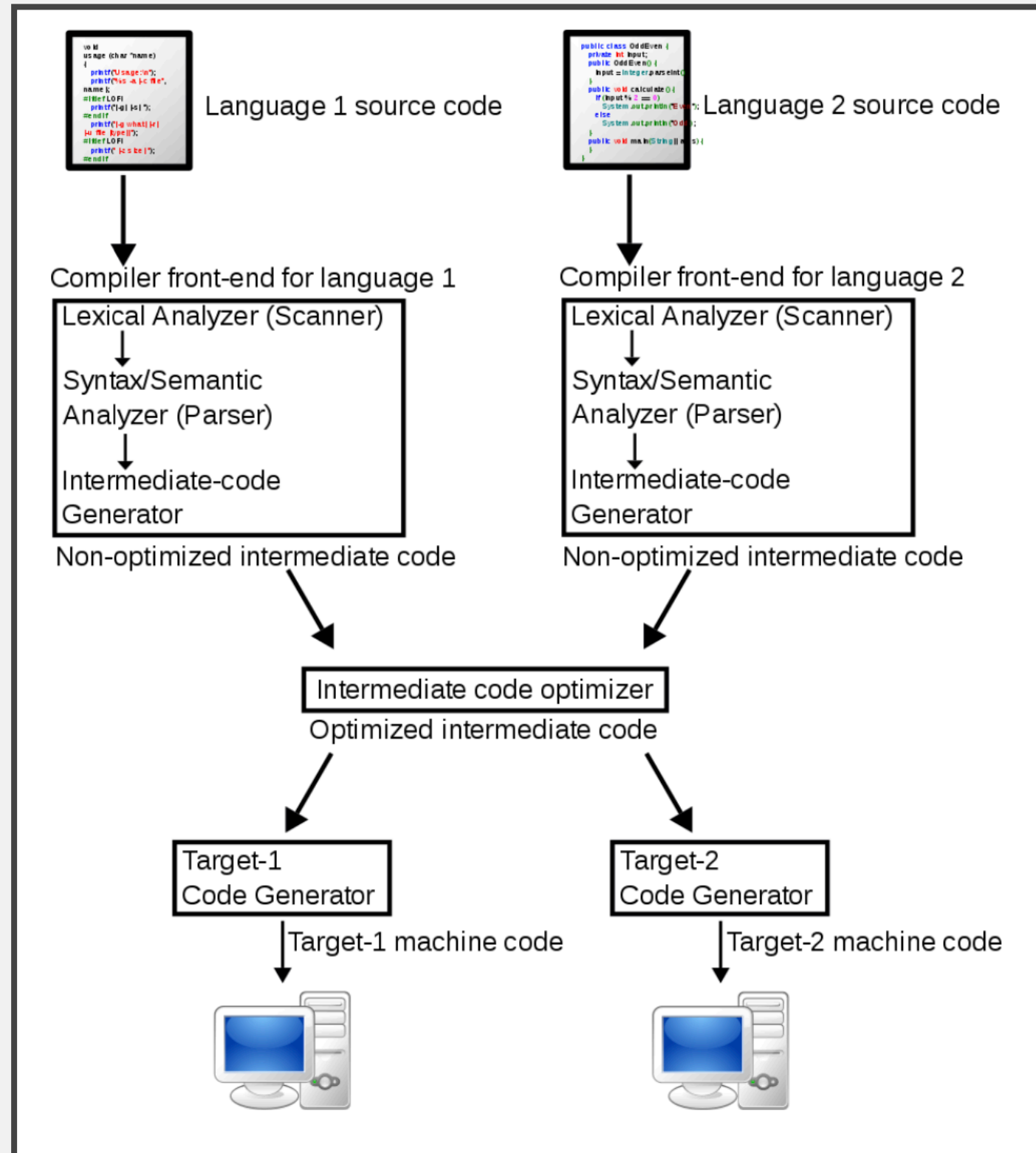


# I. Pipes & Filters

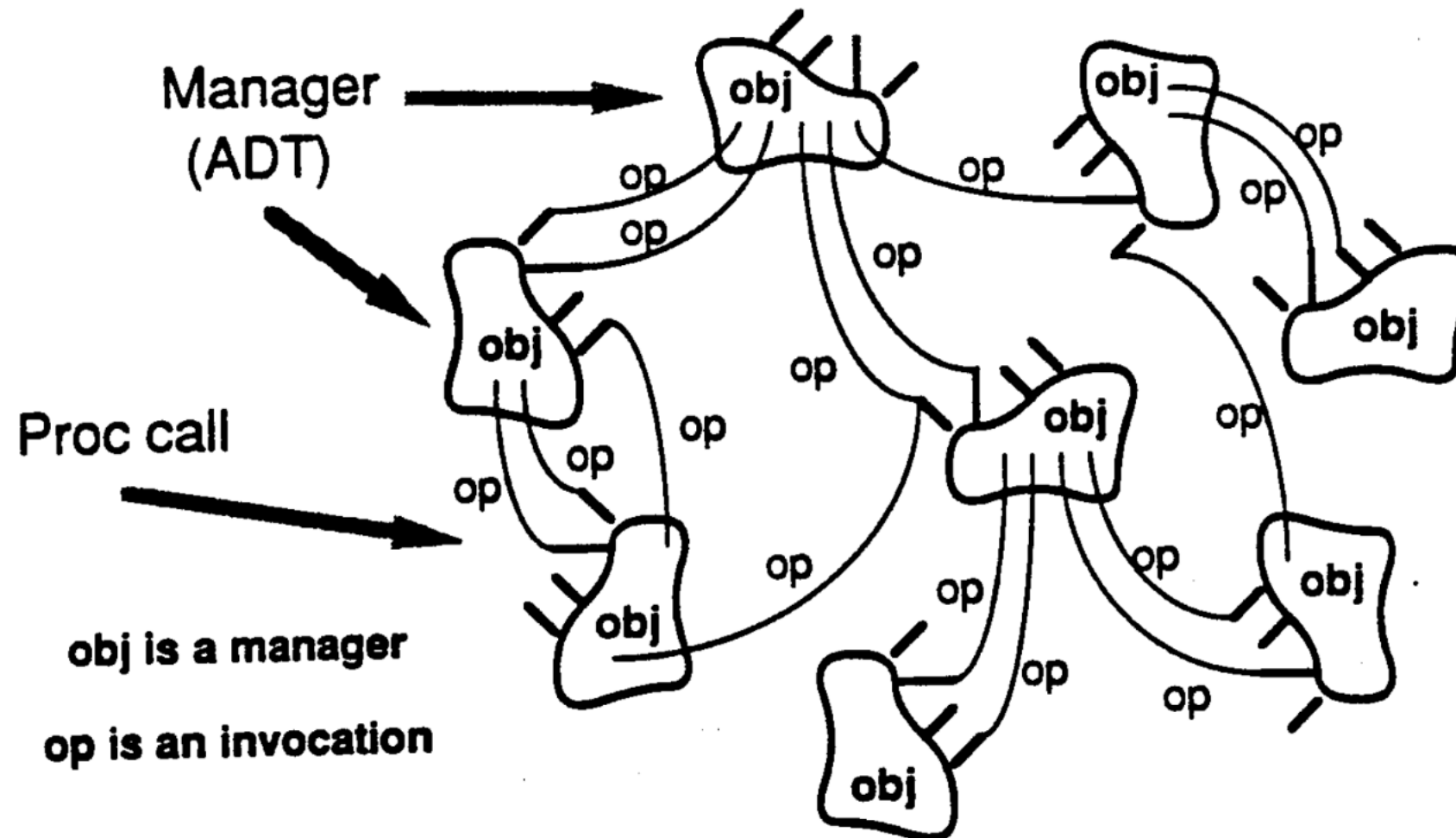


© David Garlan and Mary Shaw, CMU/SEI-94-TR-021

# Pipes & Filters Example: Compilers



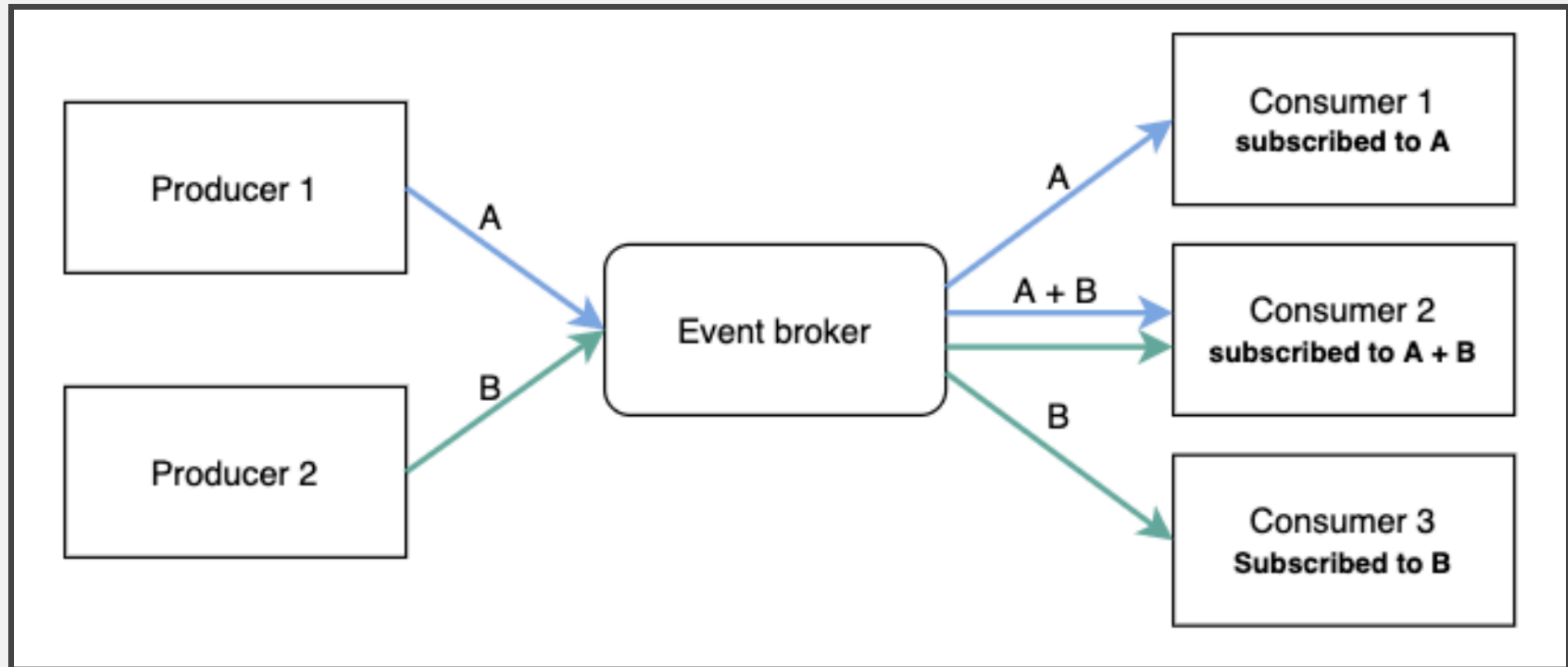
## 2. Object Oriented Organization



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



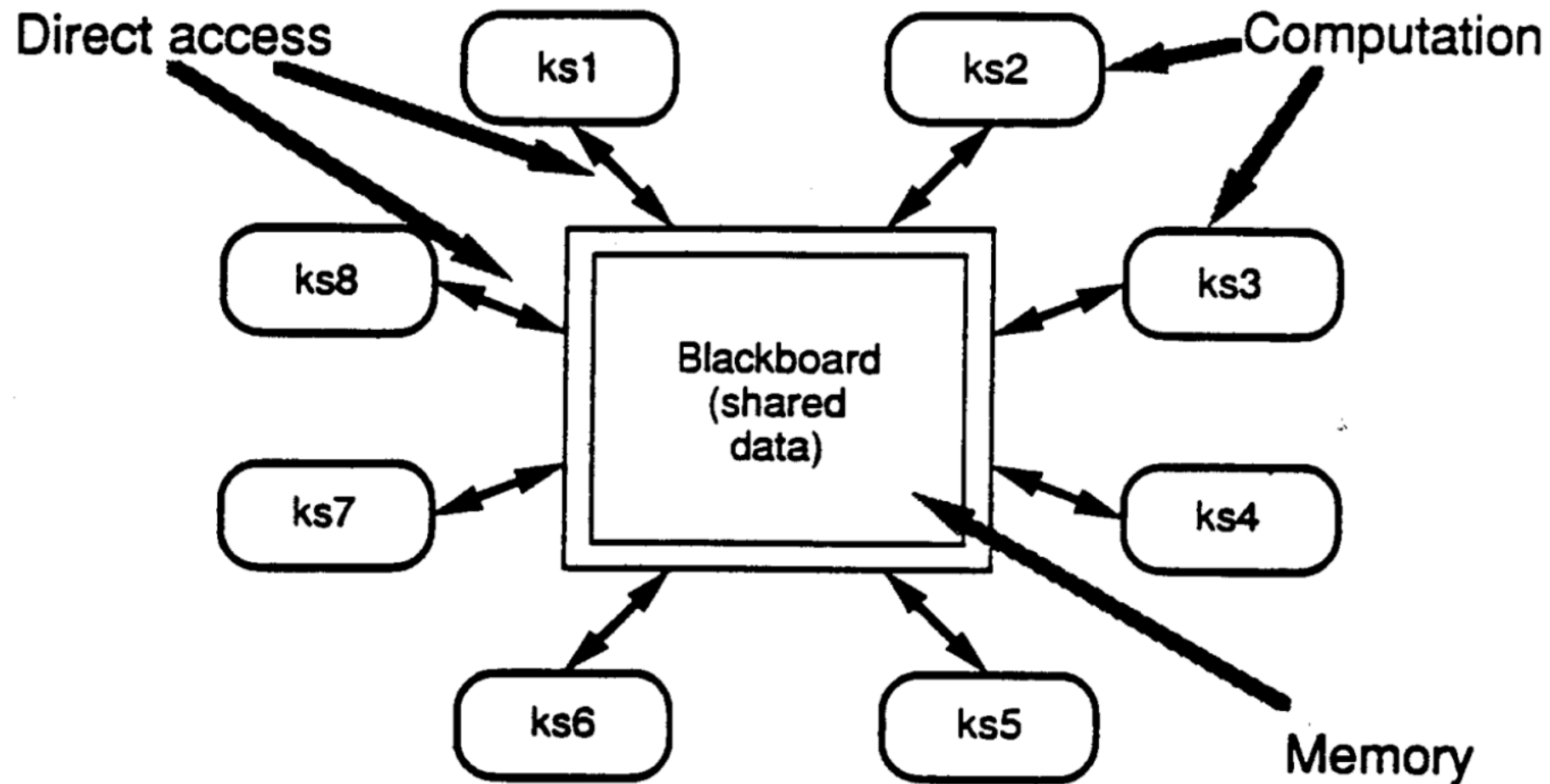
### 3. Event-Driven Architecture



# Example: HTML DOM + Javascript

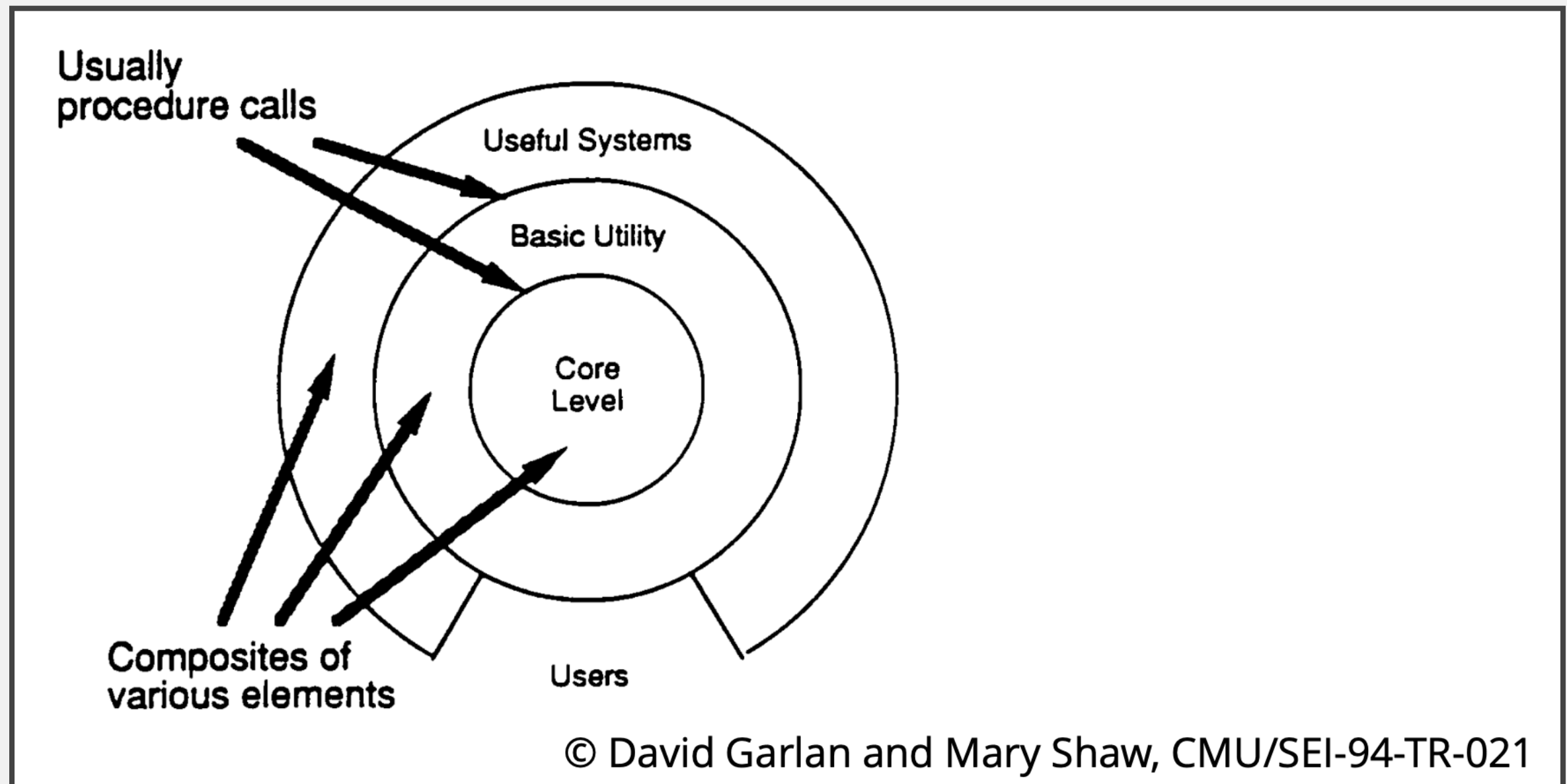


## 4. Blackboard Architecture

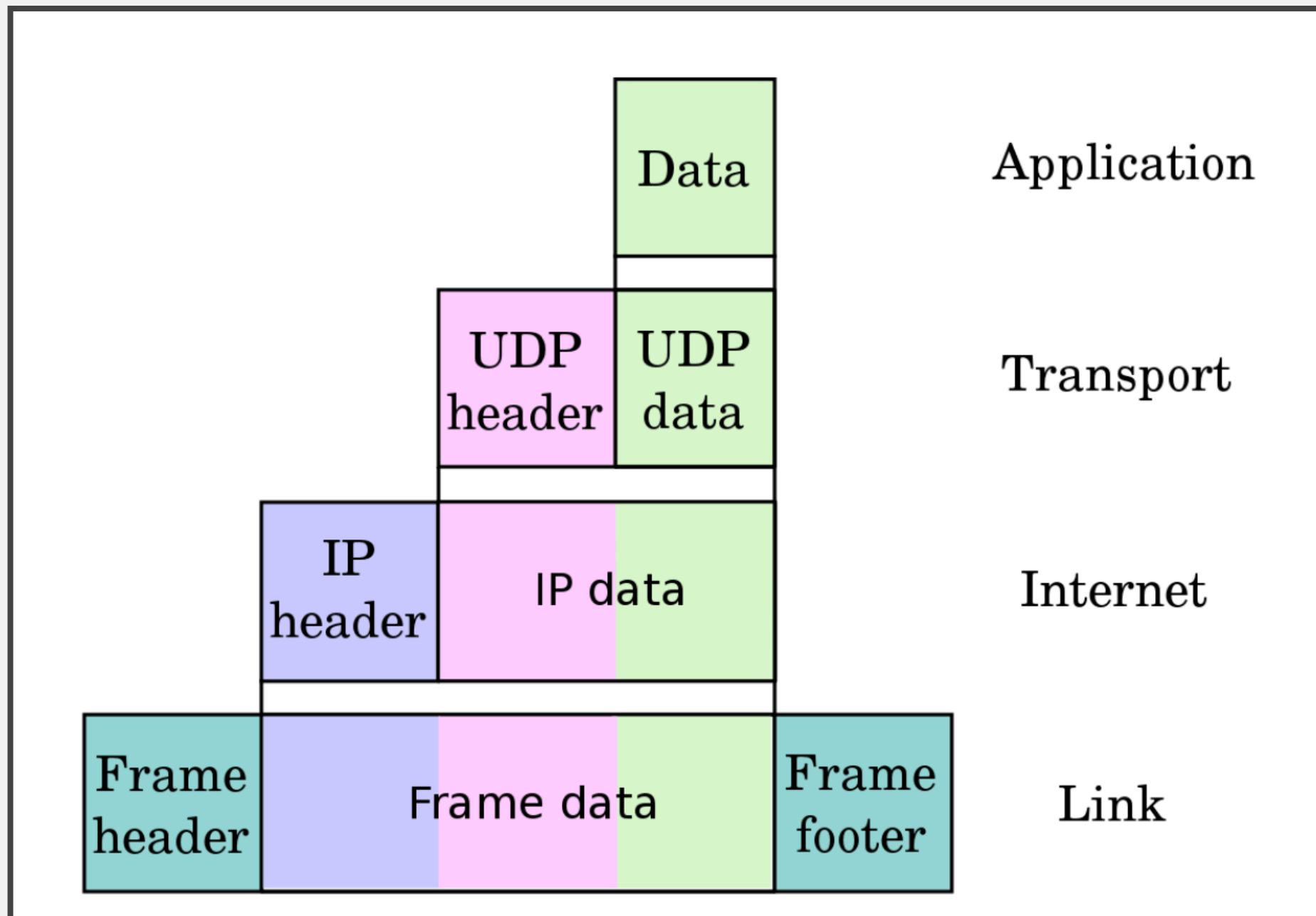


© David Garlan and Mary Shaw, CMU/SEI-94-TR-021

# 5. Layered Systems



# Example Internet Protocol Suite



# Guidelines for Selecting a Notation



- Suitable for purpose
- Often visual for compact representation
- Usually boxes and arrows
- UML possible (semi-formal), but possibly constraining
  - Note the different abstraction level – Subsystems or processes, not classes or objects
- Formal notations available
- Decompose diagrams hierarchically and in views
- Always include a legend
- Define precisely what the boxes mean
- Define precisely what the lines mean
- Do not try to do too much in one diagram
  - Each view of architecture should fit on a page
  - Use hierarchy