Introduction to Software Architecture (II)

17-313 Fall 2026

Foundations of Software Engineering

https://cmu-17313q.github.io

Eduardo Feo Flushing

Administrivia

- P3B deadline tomorrow.
- Team surveys running as usual.
- P4 out tomorrow.
- Remember the no-laptop policy and classroom etiquette.

Learning Goals

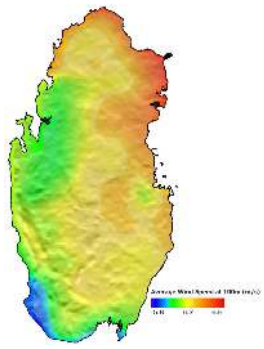
- Understand the abstraction level of architectural reasoning.
- Appreciate how software systems can be viewed at different abstraction levels.
- Distinguish software architecture from (object-oriented) software design.
- Explain the importance of architectural decisions.
- Integrate architectural decisions into the software development process.
- Document architectures clearly, without ambiguity.

Outline

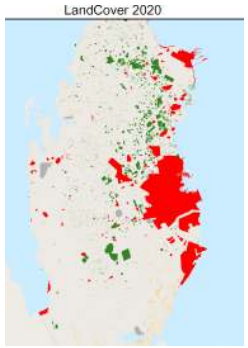
- Views and Abstraction
- Case Study: Autonomous Vehicles
- Definitions and Importance
- Software Design vs. Architecture
- Architecting Software
- Integrating Architectural Decisions
- Common Architectural Styles
- Documentation

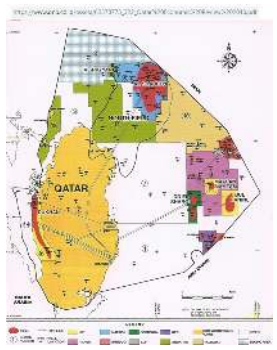
Many Views of One Place













NOTES

A road map, a wind-speed map, a municipal-boundary map, a land-cover map, a zoning map, a navigation map: the same place, documented for very different purposes.

What Makes a Good View?

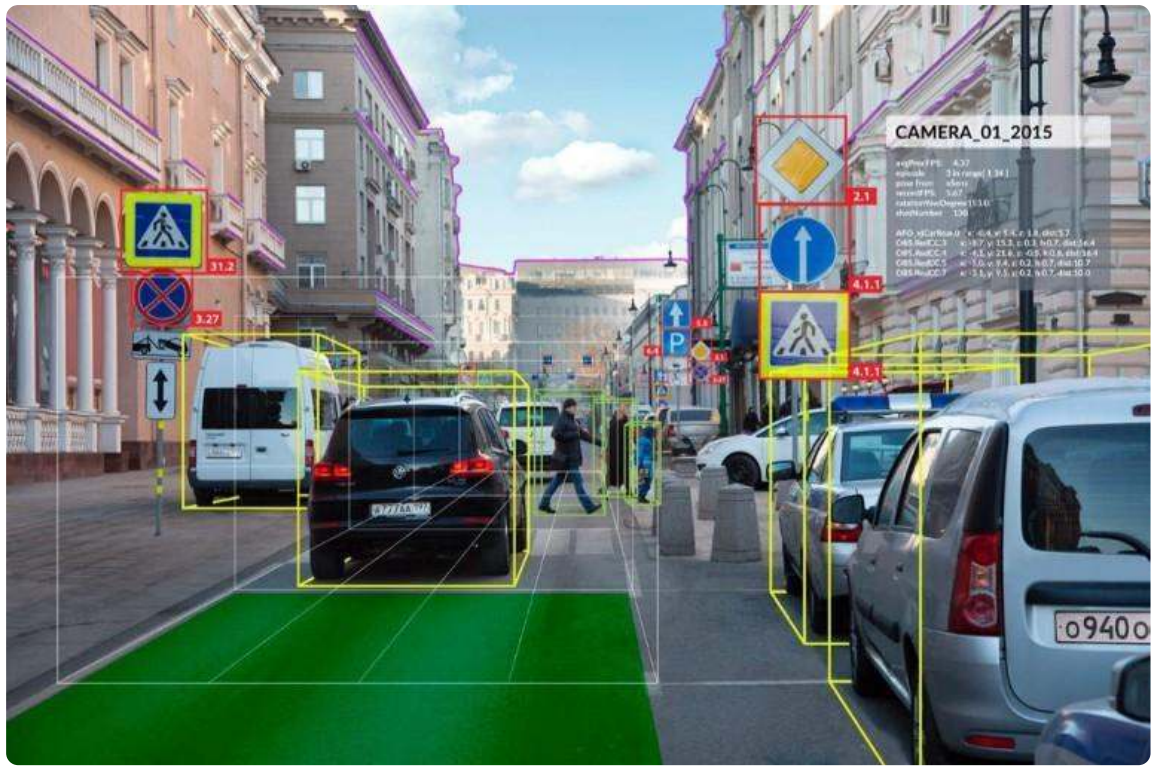
- Abstracted views focus on conveying specific information.
- They have a well-defined purpose.
- Show only necessary information; abstract away unnecessary details.
- Use legends and annotations to remove ambiguity.
- Multiple views of the same object tell a larger story.

An abstraction shows the important details and hides the unimportant ones, relative to its purpose.

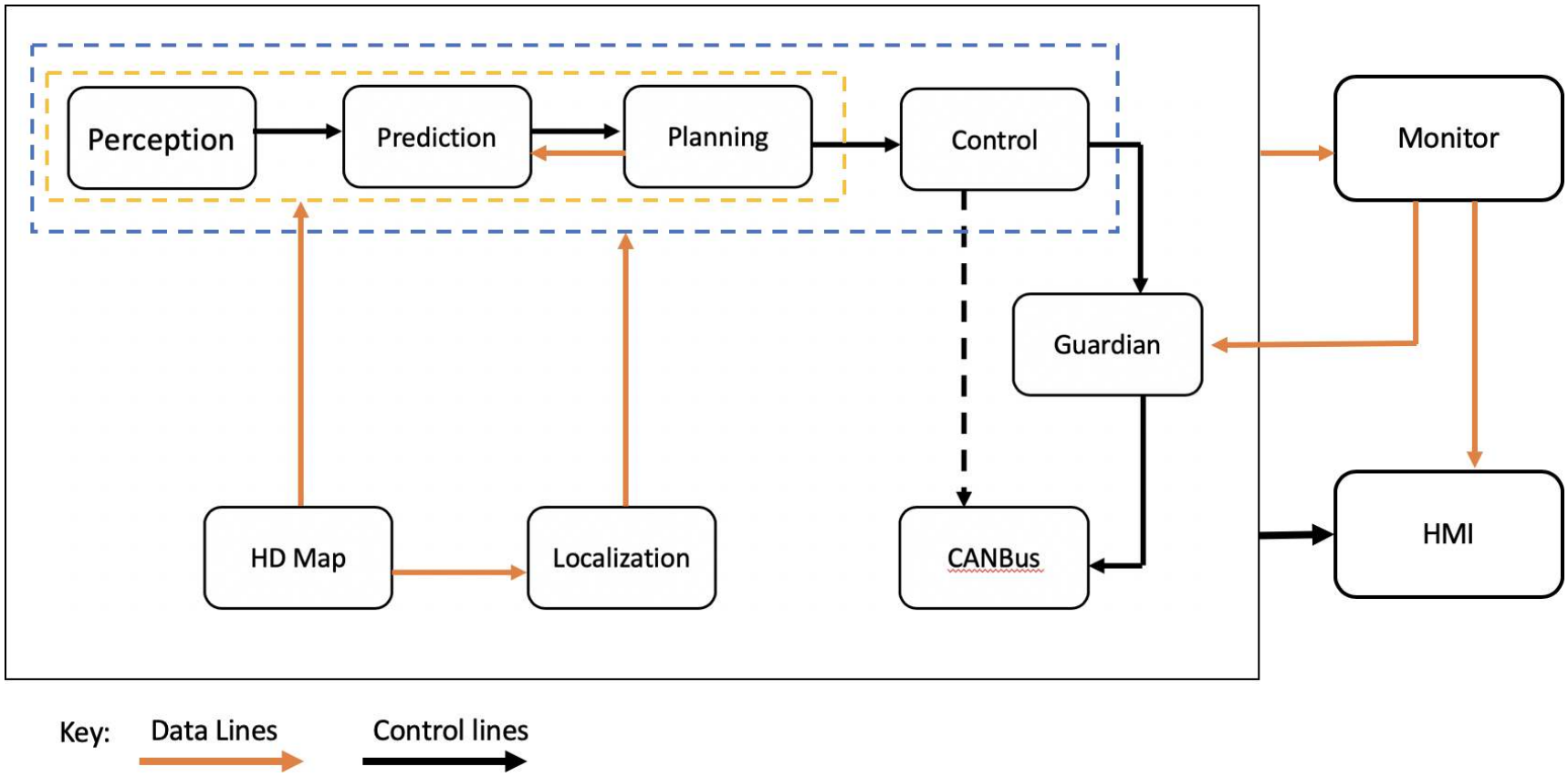
Outline

- Views and Abstraction
- **Case Study: Autonomous Vehicles**
- Definitions and Importance
- Software Design vs. Architecture
- Architecting Software
- Integrating Architectural Decisions
- Common Architectural Styles
- Documentation

Case Study: Autonomous Vehicle Software

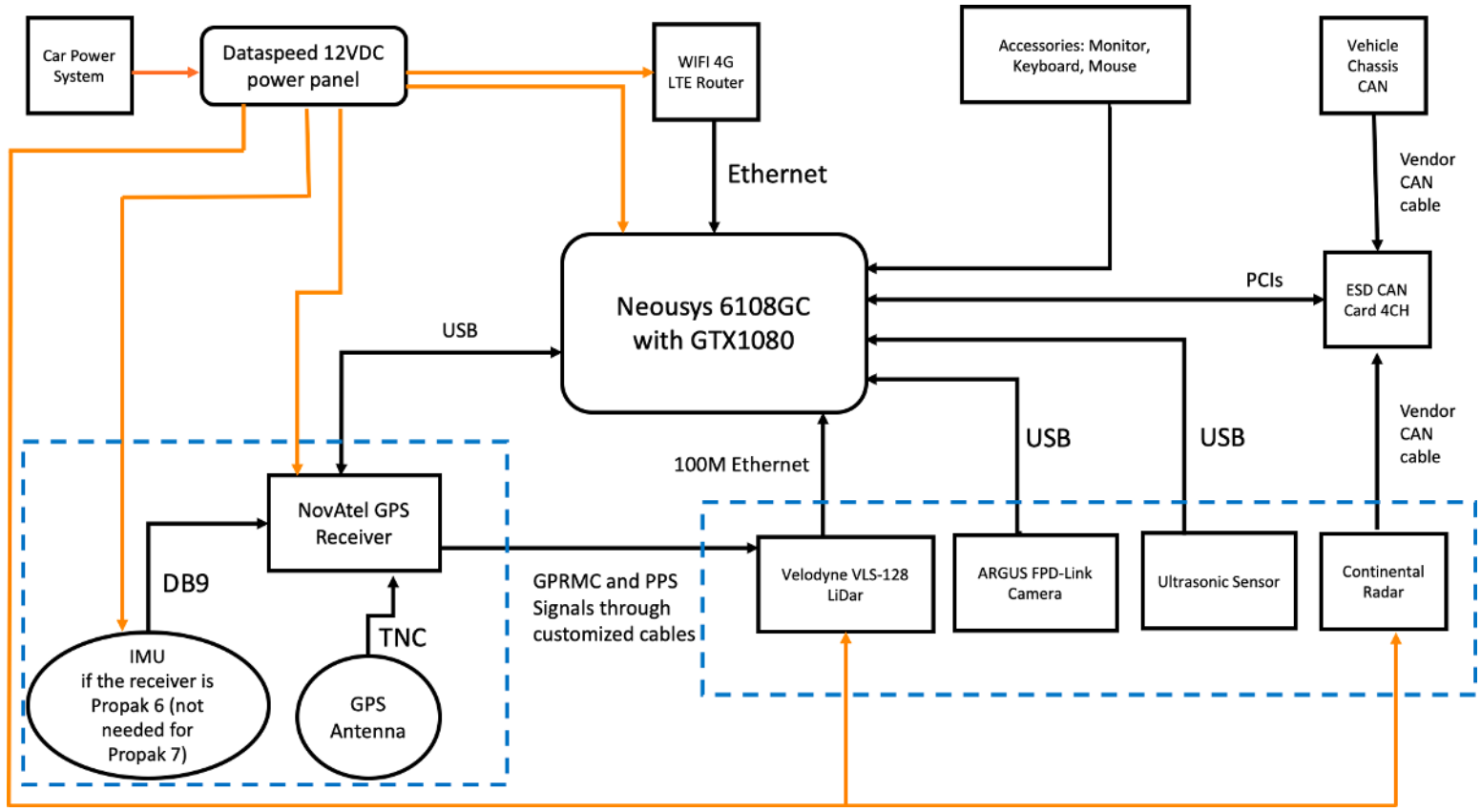


Apollo Software Architecture



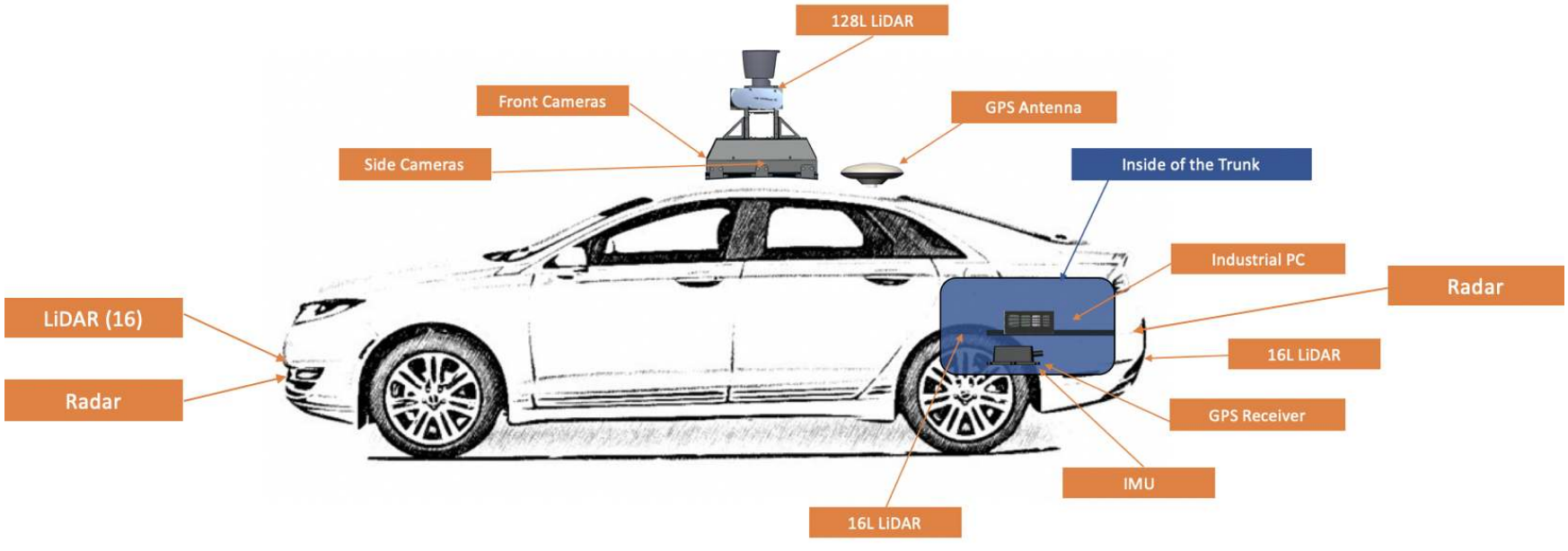
Source: github.com/ApolloAuto/apollo/blob/v6.0.0/docs/specs/Apollo_5.5_Software_Architecture.md

Apollo Hardware Architecture



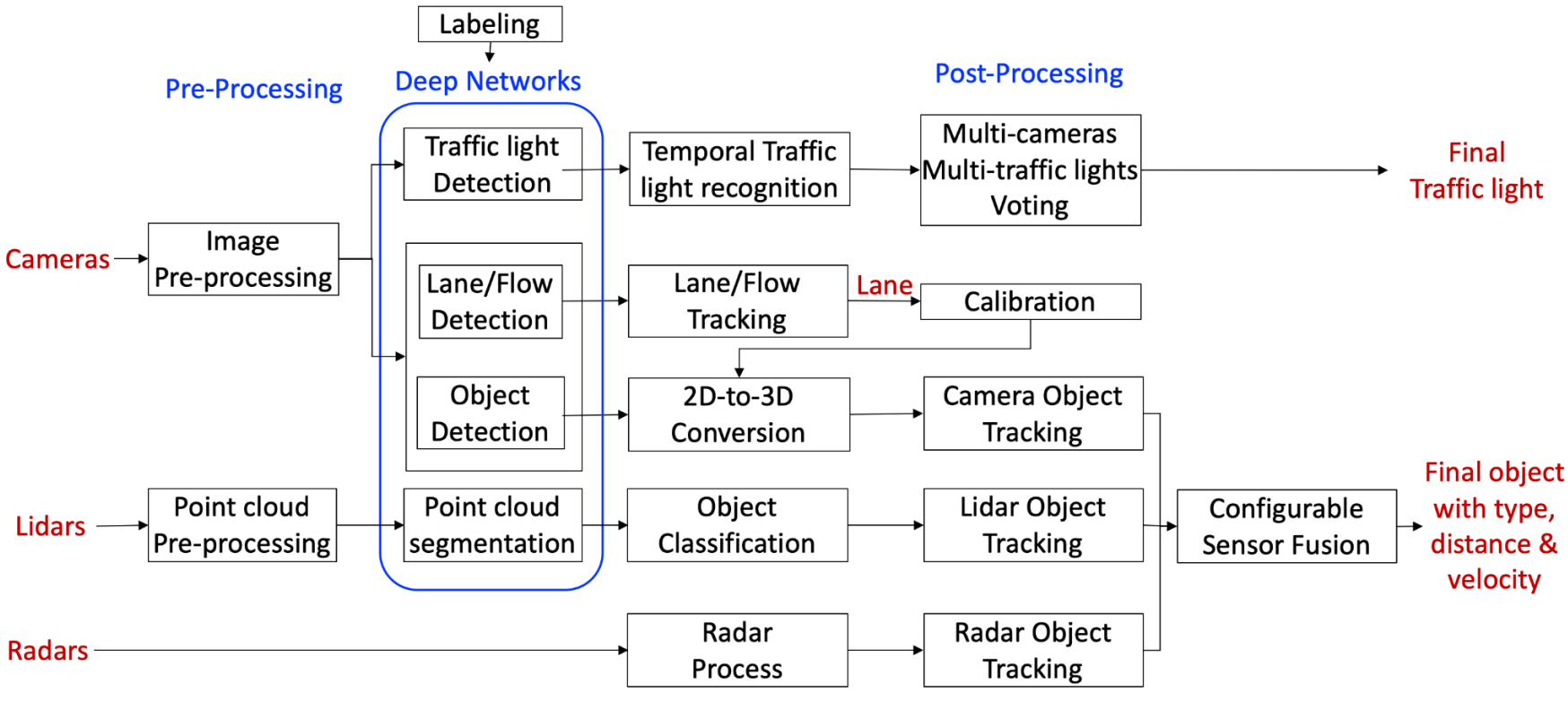
Source: github.com/ApolloAuto/apollo/blob/v6.0.0/README.md

Apollo Hardware/Vehicle Overview

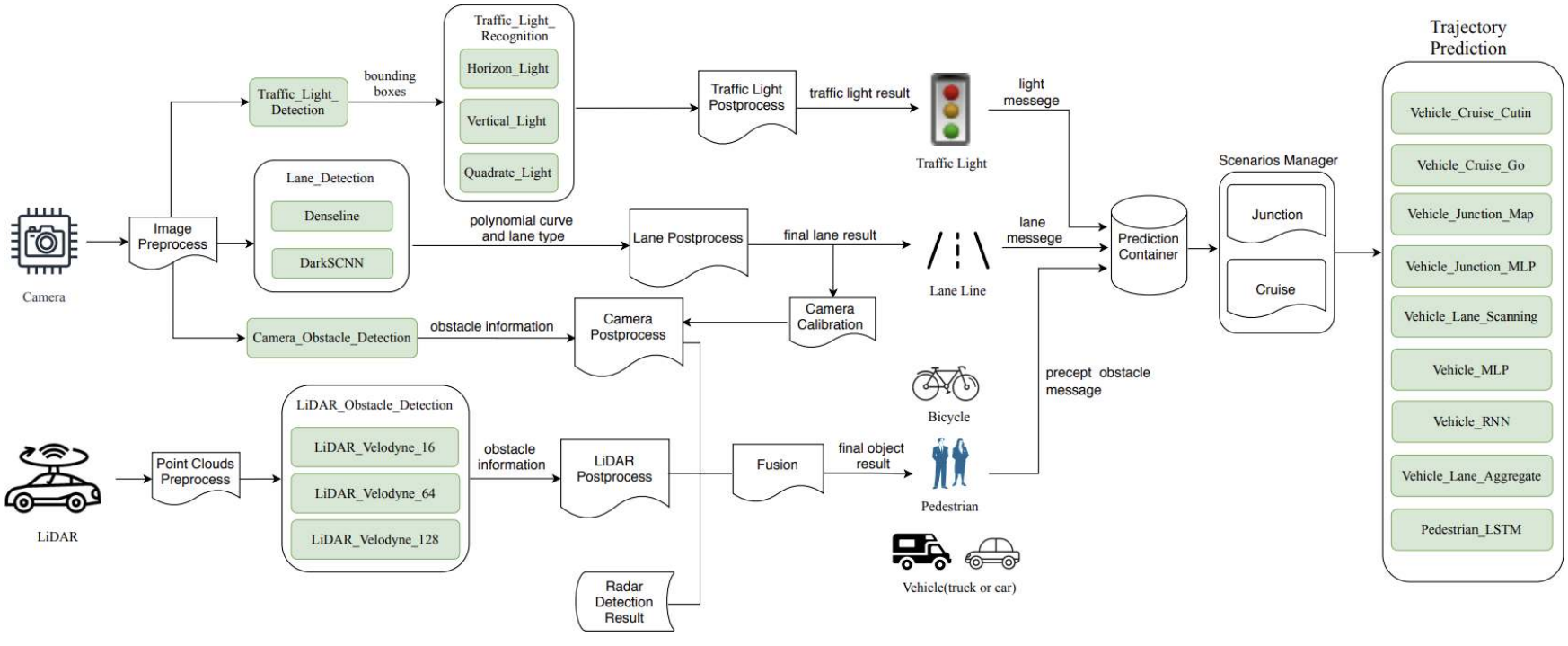


Source: github.com/ApolloAuto/apollo/blob/v6.0.0/README.md

Apollo Perception Module



Apollo ML Models



Source: Peng, Yang, Chen, Ma. A First Look at the Integration of Machine Learning Models in Complex Autonomous Driving Systems: A Case Study on Apollo. ESEC/FSE 2020.

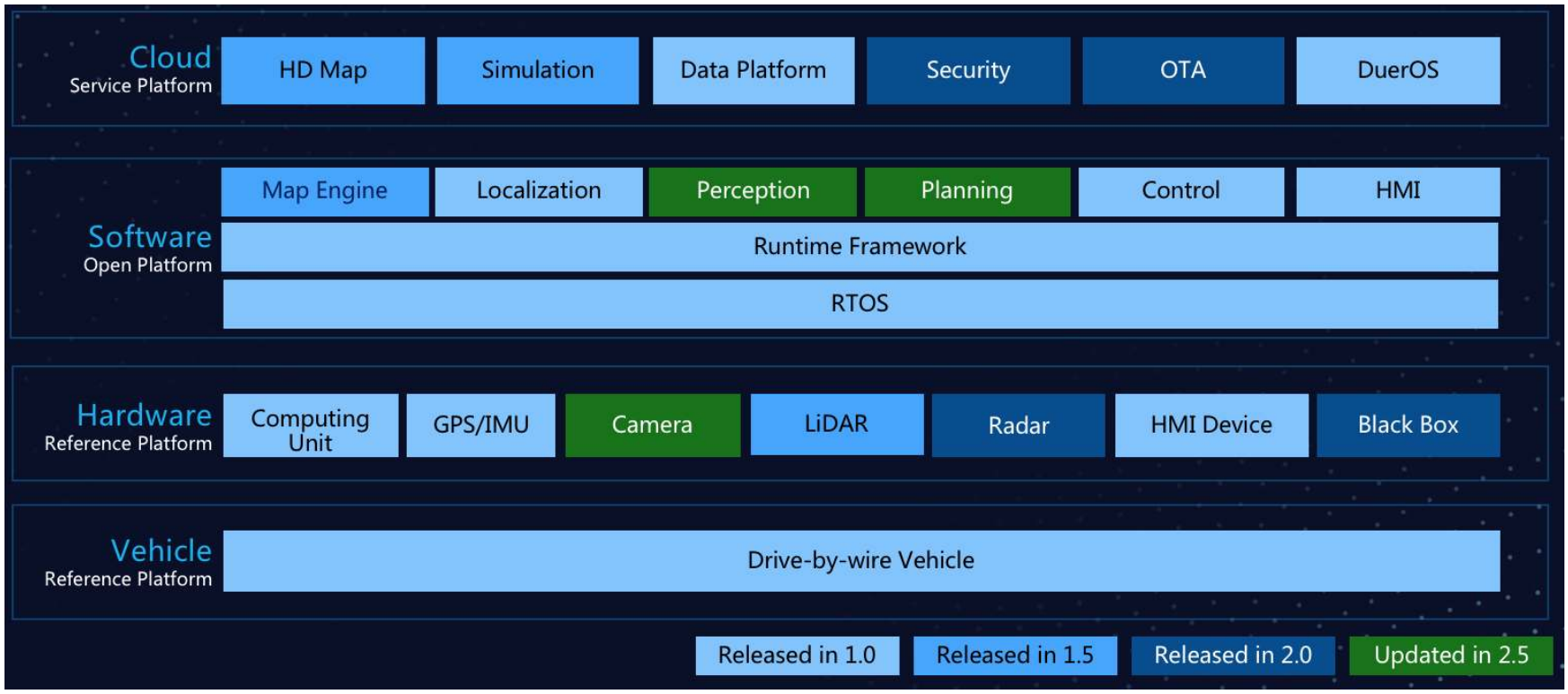
Apollo Software Stack

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: github.com/ApolloAuto/apollo

Feature Evolution (Software Stack View)



Every view serves a purpose: this one tracks which layer of the stack each released feature touched.

Source: github.com/ApolloAuto/apollo

Activity: The "Side Pass" Feature

- Watch the "side pass" feature in action: youtube.com/watch?v=BXNDUtNZdM4
- In groups of 3–4, using the architecture diagrams from the last several slides: which modules or components are involved in enabling this feature?
- Other resources: source, Doxygen

Outline

- Views and Abstraction
- Case Study: Autonomous Vehicles
- **Definitions and Importance**
- Software Design vs. Architecture
- Architecting Software
- Integrating Architectural Decisions
- Common Architectural Styles
- Documentation

"Architecture Is About the Important Stuff"



design

Editor: Martin Fowler ▫ ThoughtWorks ▫ fowler@acm.org

Who Needs an Architect?

Martin Fowler

Wandering down our corridor a while ago, I saw my colleague Dave Rice in a particularly grumpy mood. My brief question caused a violent statement, "We shouldn't interview anyone who has 'architect' on his resume." At first blush, this was an odd turn of phrase, because we usually introduce Dave as one of our leading architects.

The reason for his title schizophrenia is the fact that, even by our industry's standards, "architect" and "architecture" are terribly overloaded words. For many, the term "software architect" fits perfectly with the smug controlling image at the end of *Matrix Reloaded*. Yet even in firms that have the greatest contempt for that image, there's a vital role for the technical leadership that an architect such as Dave plays.

What is architecture?

When I was fretting over the title for *Patterns of Enterprise Application Architecture* (Addison-Wesley, 2002), everyone who reviewed it agreed that "architecture" bothered them. Most definitions of architecture emphasize the importance of high-level decisions, but I've worked on projects where the team has collectively regretted decisions that the architect (or architects) made early on. I've also seen successful projects where important decisions were made later in development by programmers who weren't formally designated as architects.

Ultimately, I decided that I couldn't find a better word than "architecture" to describe the important stuff, whatever that is. So I left the word in the book's title, but I had to agree with the reviewers that the current definitions of the word aren't very good.

Recently, I've been trying to get a definition that I can live with. I'm not there yet, but I'm getting closer. However, as so often occurs, inside the blighted cynicism is a pinch of truth. Understanding came to me after reading a posting from Ralph Johnson on the Extreme Programming mailing list. It's so good I'll quote it all.

A previous posting said

> The RUP, working off the IEEE definition, defines architecture as "the highest level concept of a system in its environment. The architecture of a software system (at a given point in time) is its organization or structure of significant components interacting through interfaces, those components being composed of successively smaller components and interfaces."

Johnson responded:

> I was a reviewer on the IEEE standard that used that, and I argued uselessly that this was clearly a completely bogus definition. There is no highest level concept of a system. Customers have a different concept than developers. Customers do not care at all about the structure of significant components. So, perhaps an architecture is the highest level concept that developers have of a system in its environment. Let's forget the dead

"Architecture is about the important stuff. Whatever that is." – Ralph Johnson

A Formal Definition

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]

This definition is ambivalent to whether the architecture is known, or whether it's any good.

NOTES

A software architecture description names the major elements, their responsibilities and properties, and how those elements relate to each other, with enough detail to understand the design decisions behind it. It focuses on externally visible properties, not implementation detail.

Elements of Software Architecture

- **Abstraction.**
- Elements: roles, responsibilities, behaviors, properties.
- Relationships between elements.
- Relationships to non-software elements: hardware, external systems.
- Described from many different "external" perspectives.
- Hides "internal" details.

NOTES

Architecture is the level of abstraction where you focus on elements, their responsibilities, and their relationships, without getting lost in implementation. The important details are the design decisions that affect properties only achievable through structure.

Why Software Architecture Matters

- Facilitates internal and external communication.
- Describes design decisions and prescribes implementation constraints.
- Relates to organizational structure.
  - Conway's Law: the technical architecture of a software system mirrors the communication structure of the organization that builds it.
- Permits or precludes achieving non-functional requirements.
- Allows control of complexity, managing change, and (better) estimating effort.

NOTES

Architecture is the vehicle for communication among stakeholders: it provides the vocabulary used to discuss requirements and schedule. Once a design decision is made, it constrains how the rest of the system can be implemented.

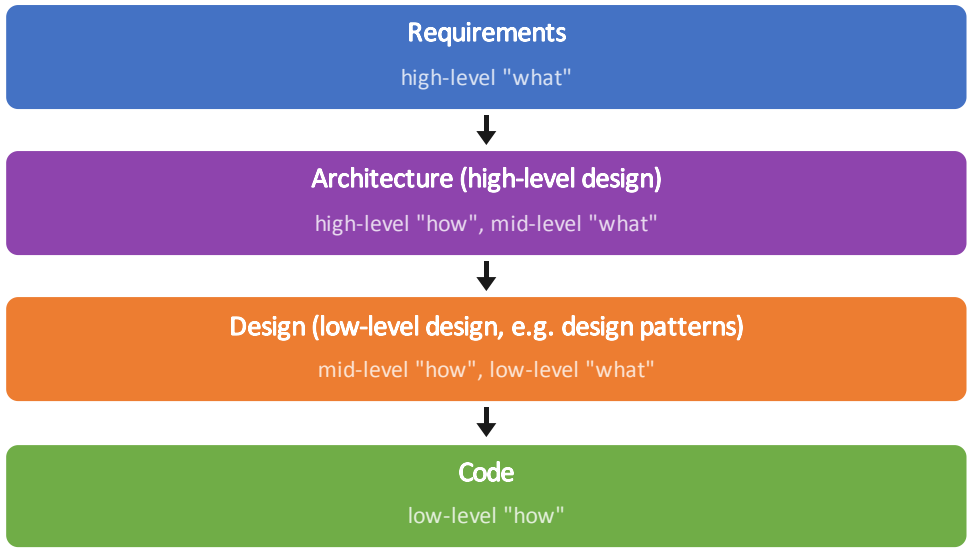
Wojcik, R. Architecting Software the SEI Way: Software Architecture Fundamentals: Technical, Business, and Social Influences. 2012.

Outline

- Views and Abstraction
- Case Study: Autonomous Vehicles
- Definitions and Importance
- **Software Design vs. Architecture**
- Architecting Software
- Integrating Architectural Decisions
- Common Architectural Styles
- Documentation

Software Design vs. Architecture

Levels of Abstraction



NOTES

Architectural decisions sit between requirements and design: architecture is the high-level "how," while design (including design patterns) is the mid-level "how."

Design Questions vs. Architectural Questions

Design Questions

- How do I add a new slash command to OpenCode?
- How can I make it easy to render a tool-call result?
- 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

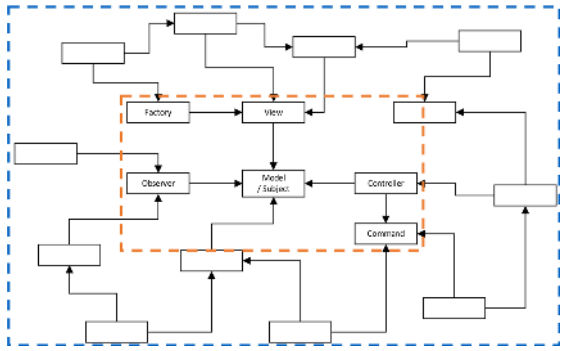
- How do I extend OpenCode with a new provider integration?
- What threads exist, and how do they coordinate?
- How does Google scale to billions of hits per day?
- Where should I put my firewalls?
- What is the interface between subsystems?

NOTES

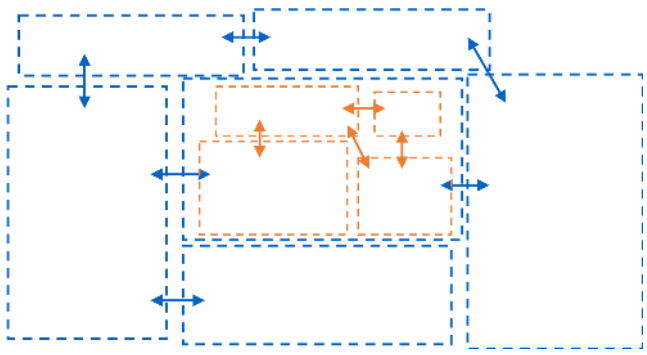
No single definition pins down what architecture is, so the distinction is easiest to see through examples: design questions are about one module or class, while architectural questions are about subsystems and how they are organized.

From Software Elements to Architecture

- Start with objects/modules and design patterns (Model/View/Controller, Factory, Observer, Command, …): a local, pairwise view of how a few classes interact.
- Zoom out to architecture: the same system, organized into subsystems with clear boundaries and responsibilities.
- Same underlying system, two very different levels of abstraction.



Design patterns: local structure



Architecture: organized subsystems

NOTES

Outline

- Views and Abstraction
- Case Study: Autonomous Vehicles
- Definitions and Importance
- Software Design vs. Architecture
- **Architecting Software**
- Integrating Architectural Decisions
- Common Architectural Styles
- Documentation

Every System Has an Architecture

- Whether you know it or not.
- Whether you like it or not.
- Whether it's documented or not.
- If you don't consciously elaborate the architecture, it will evolve by itself.

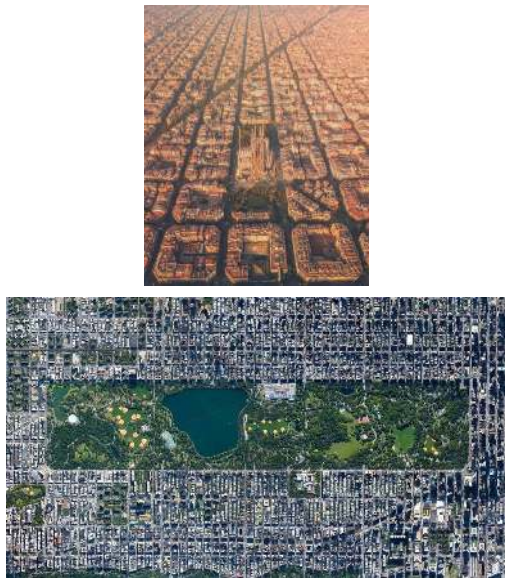


Every software system has an architecture, so don't complain later.

Wojcik, R. Architecting Software the SEI Way: Software Architecture Fundamentals: Technical, Business, and Social Influences. 2012.

Two Ways to Grow a City

Planned



Barcelona, Manhattan

Organic

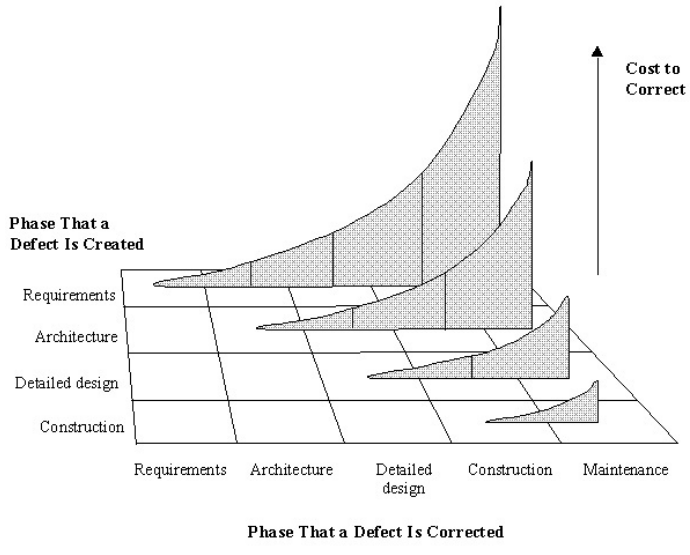


Barrios, Caracas, Venezuela

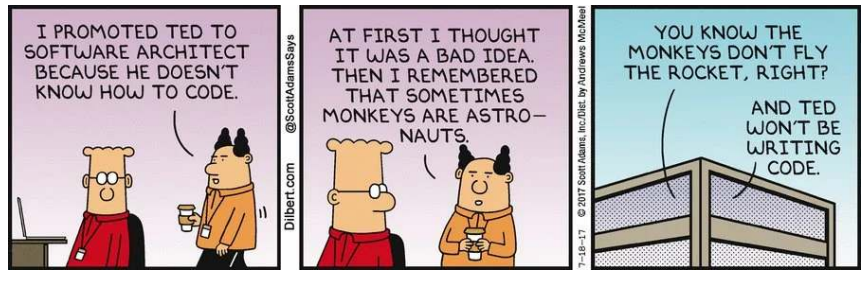
NOTES

A grid plan (Barcelona, Manhattan) is an intentional architecture, visible from above. An organic settlement, such as the barrios in Caracas, Venezuela, also has structure, but it emerged gradually instead of being designed up front. Organic does not mean bad: it can be functional and support certain quality attributes, even if it is not the style most people would call aesthetically appealing.

The Cost of a Wrong Architecture



Copyright 1998 Steven C. McConnell. Reprinted with permission from *Software Project Survival Guide* (Microsoft Press, 1998).



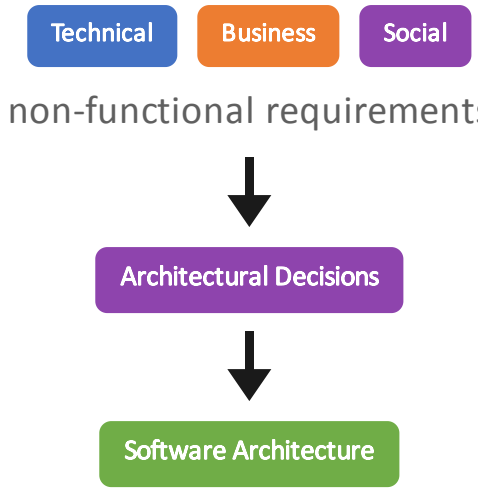
Outline

- Views and Abstraction
- Case Study: Autonomous Vehicles
- Definitions and Importance
- Software Design vs. Architecture
- Architecting Software
- **Integrating Architectural Decisions**
- Common Architectural Styles
- Documentation

How to Make Architectural Decisions

There is more than one answer.

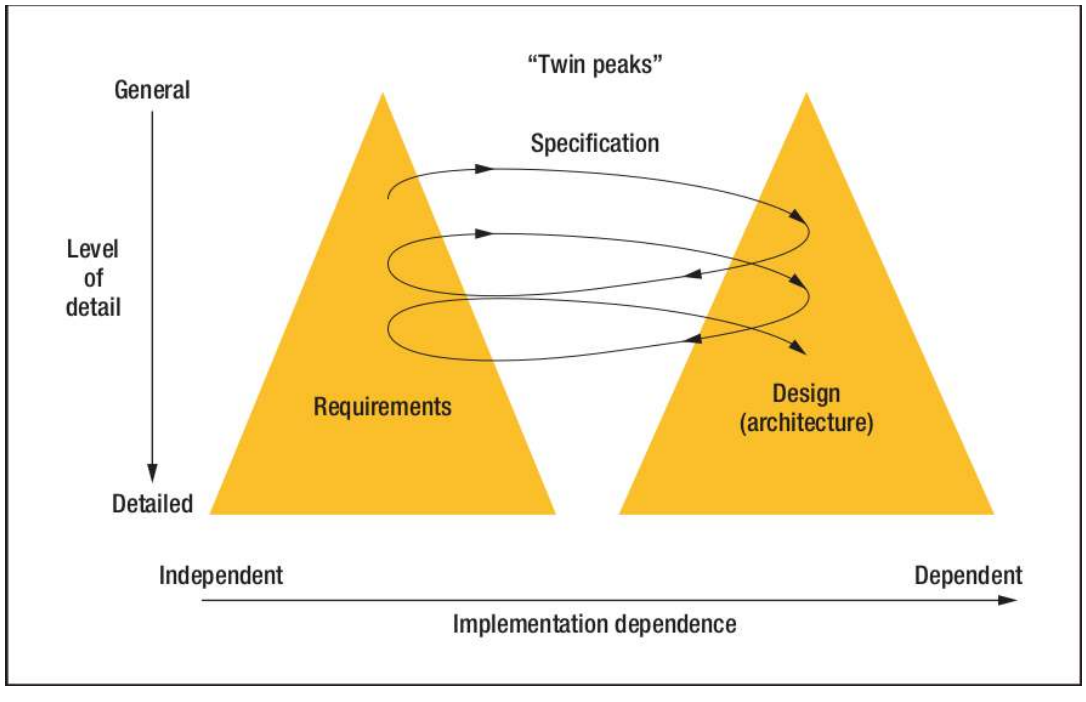
The Ecosystem of Architectural Decisions



Architecture is deeply entangled with people, goals, and systems.

Wojcik, R. Architecting Software the SEI Way: Software Architecture Fundamentals: Technical, Business, and Social Influences. 2012.

Requirements and Architecture Co-Evolve



The "twin peaks" model: requirements and architecture are elicited and specified together, not in strict sequence.

NOTES

Emphasizes the iterative nature in which requirements are elicited and specified, intertwined with architectural design. Early prototypes and architectural evaluations are needed to demonstrate that the delivered system will actually meet the requirements.

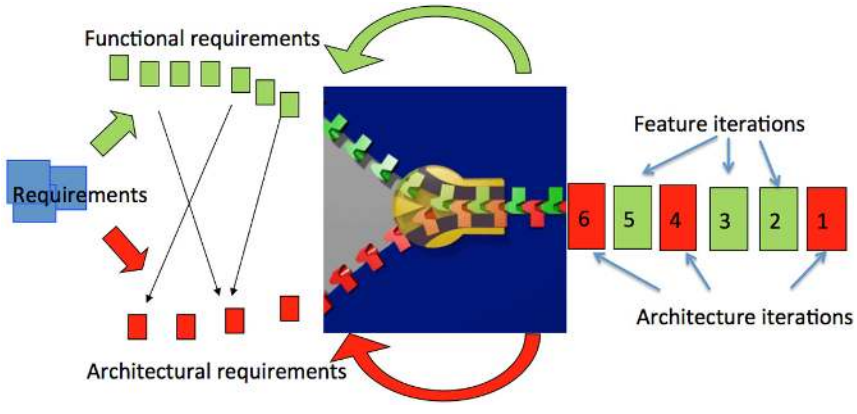
Nuseibeh, B. Weaving Together Requirements and Architectures. 2001.

Agile and Architecture



"The best architectures, requirements, and designs emerge from self-organizing teams."

The Zipper Model



How to Agilely Architect an Agile Architecture

by Stephany Bellomo, Philippe Kruchten, Robert L. Nord, and Ipek Ozkaya

Elicit architecturally significant user stories in early iterations.

Bellomo, Kruchten, Nord, Ozkaya. How to Agilely Architect an Agile Architecture.

Outline

- Views and Abstraction
- Case Study: Autonomous Vehicles
- Definitions and Importance
- Software Design vs. Architecture
- Architecting Software
- Integrating Architectural Decisions
- **Common Architectural Styles**
- Documentation

Common Architectural Styles



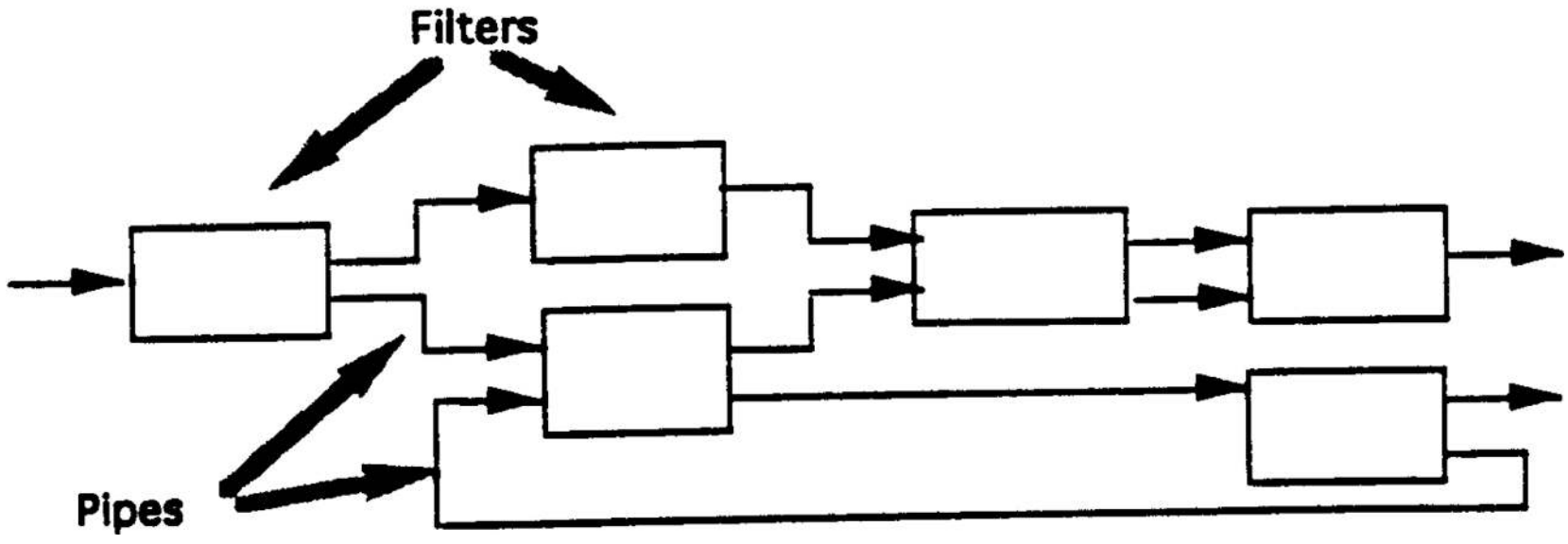





Buildings have recognizable styles. So does software: a recurring pattern of elements and relationships, reused across many systems.

thespruce.com/top-architectural-styles-4802083

1. Pipes and Filters



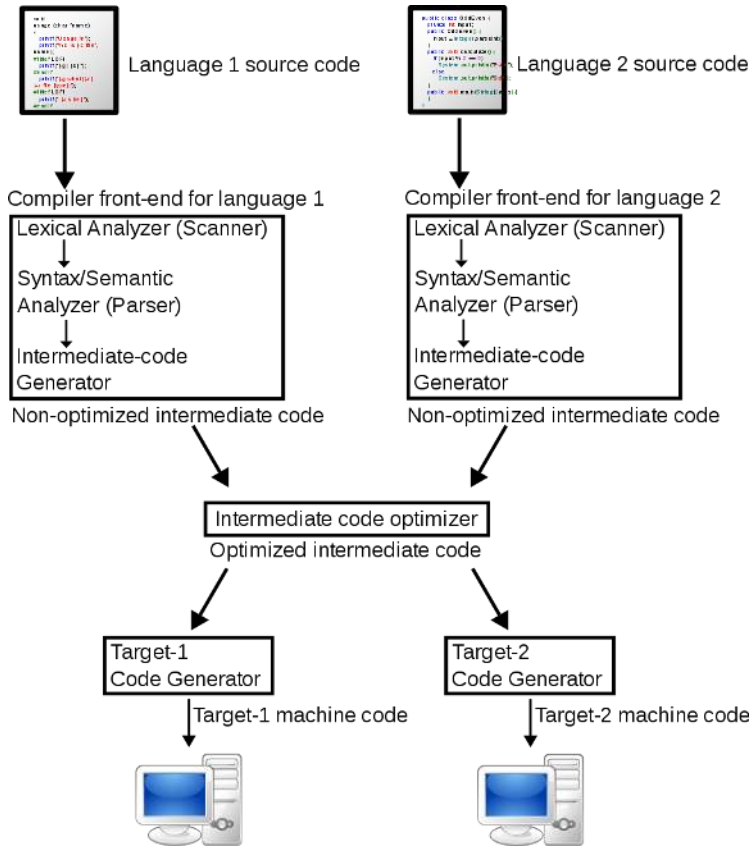
Example: compilers.

NOTES

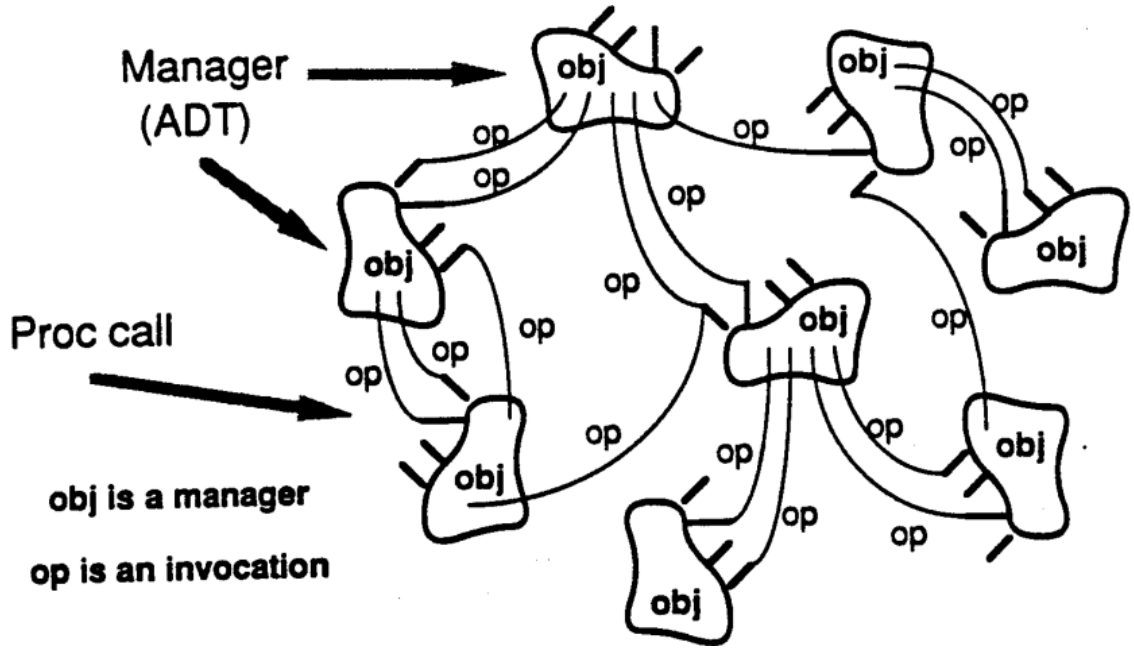
Data flows along pipes between stateless filters that transform input into output. Filters are independent and can run incrementally; there are usually no cycles unless offset by a time step. The Unix shell is a common example.

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

Example: Compilers

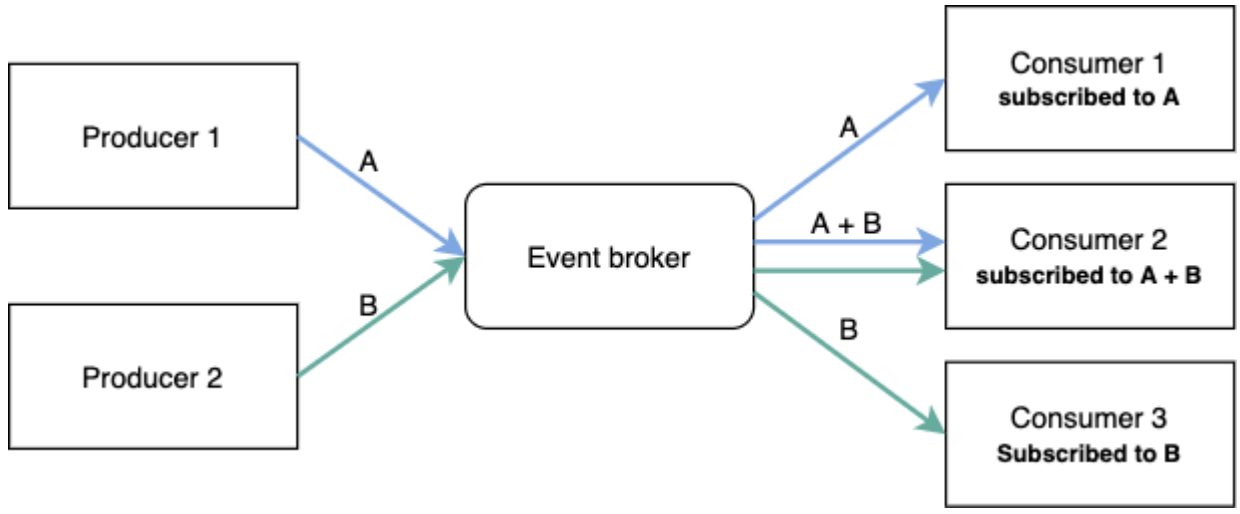


2. Object-Oriented Organization

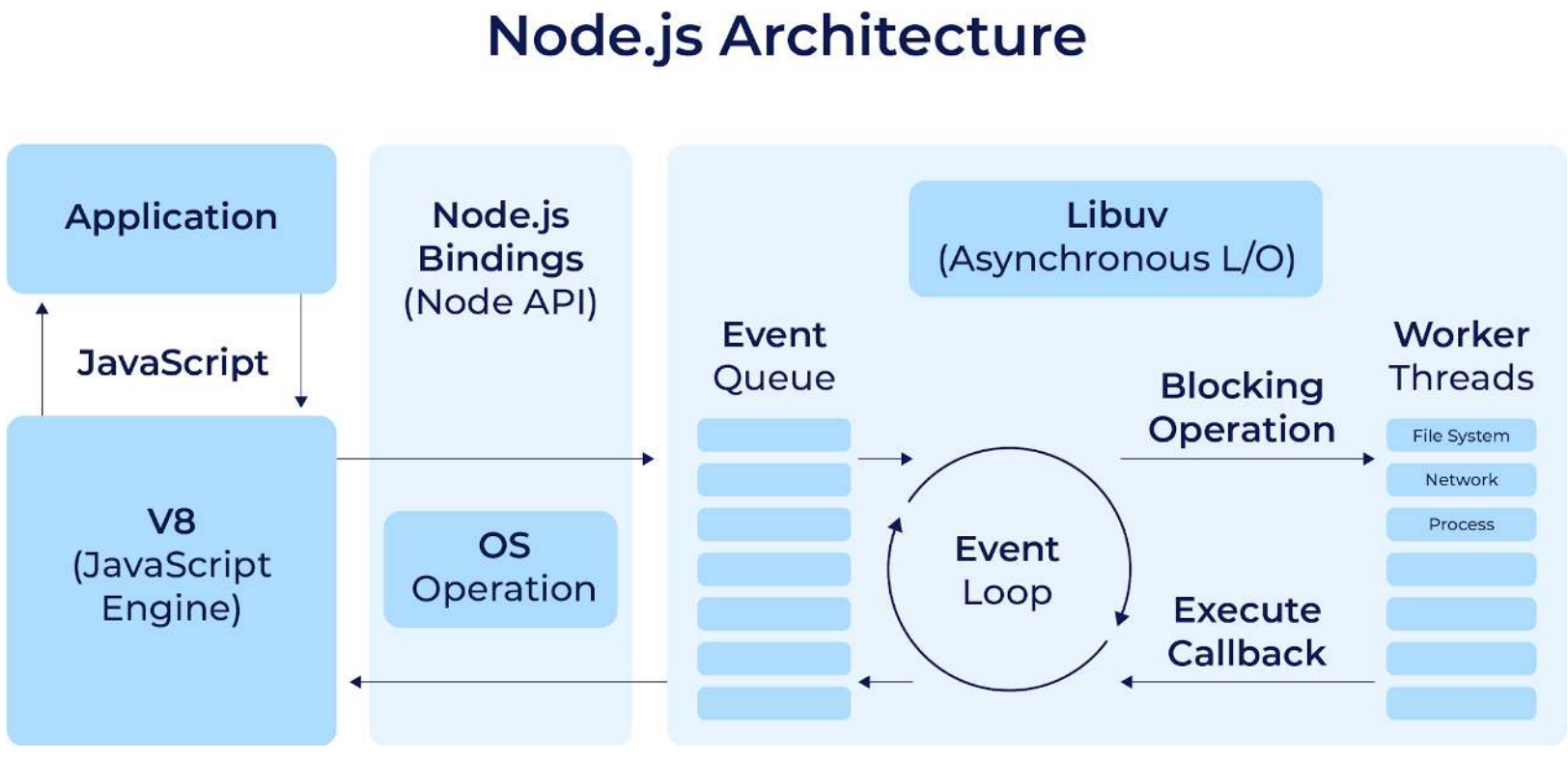


© David Garlan and Mary Shaw, CMU/SEI-94-TR-021

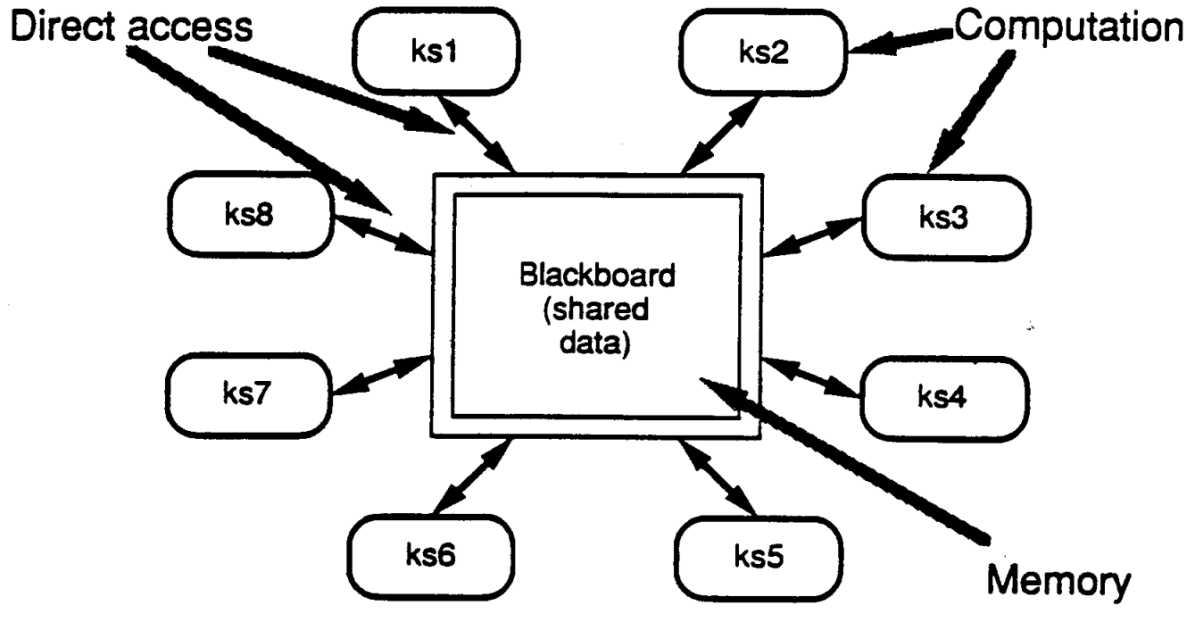
3. Event-Driven Architecture



Example: Node.js



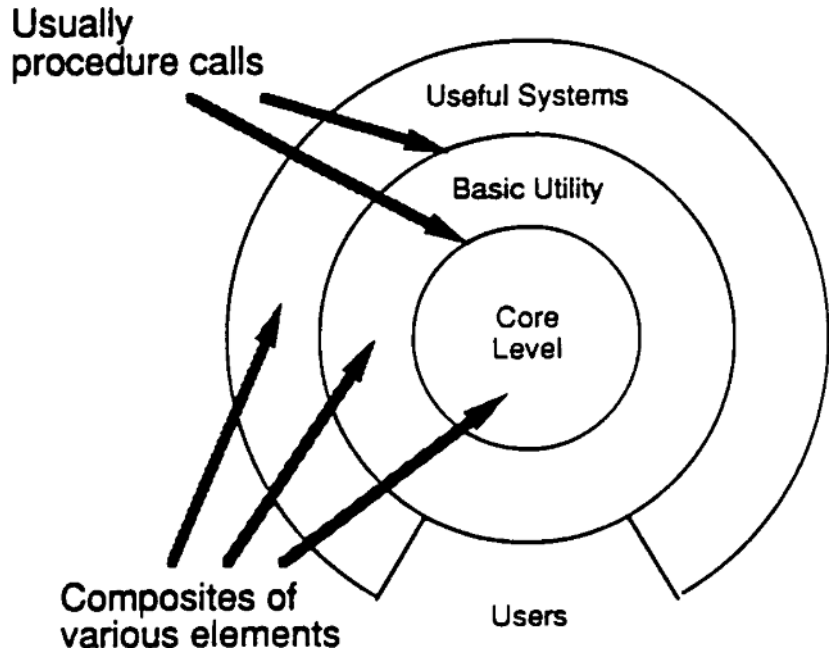
4. Blackboard Architecture



Example: stock exchanges, Photoshop.

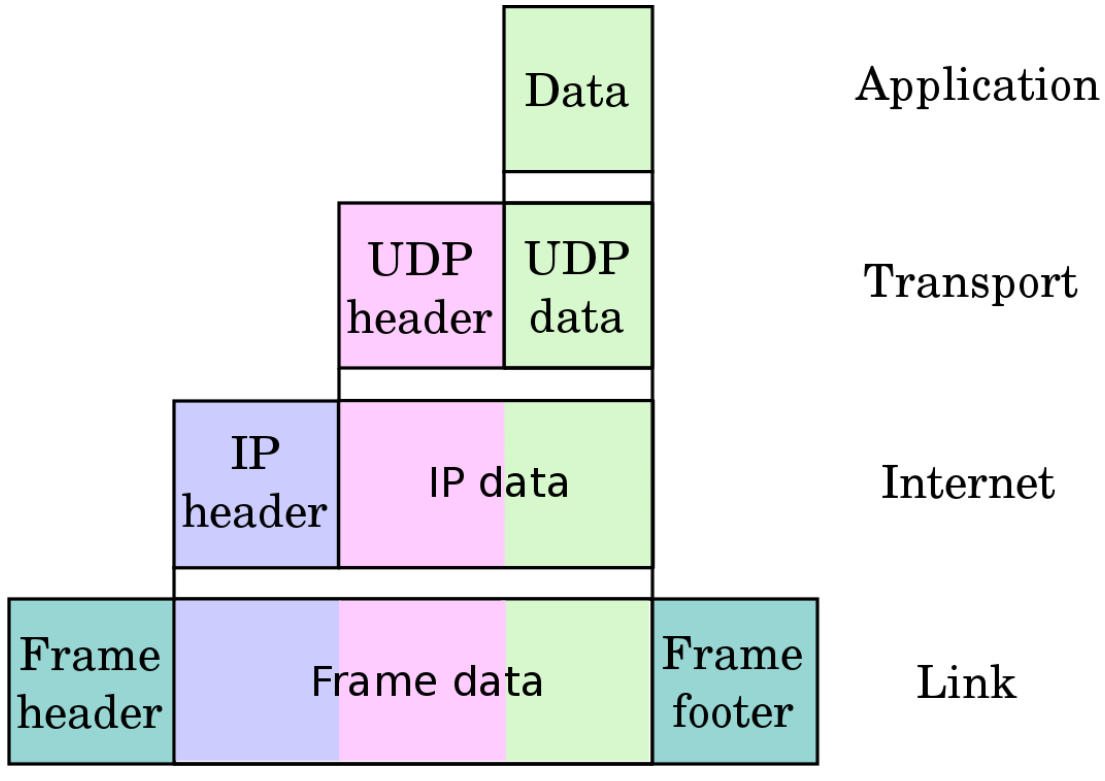
© David Garlan and Mary Shaw, CMU/SEI-94-TR-021

5. Layered Systems



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

Example: Internet Protocol Suite



Outline

- Views and Abstraction
- Case Study: Autonomous Vehicles
- Definitions and Importance
- Software Design vs. Architecture
- Architecting Software
- Integrating Architectural Decisions
- Common Architectural Styles
- **Documentation**

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 it.
- Supports traceability.

NOTES

Documentation is the vocabulary used to discuss requirements and the manifestation of the earliest, most important design decisions. Good architecture documentation is prescriptive, organized around specific quality attributes, not merely descriptive.

Architecture Documentation Is Not New



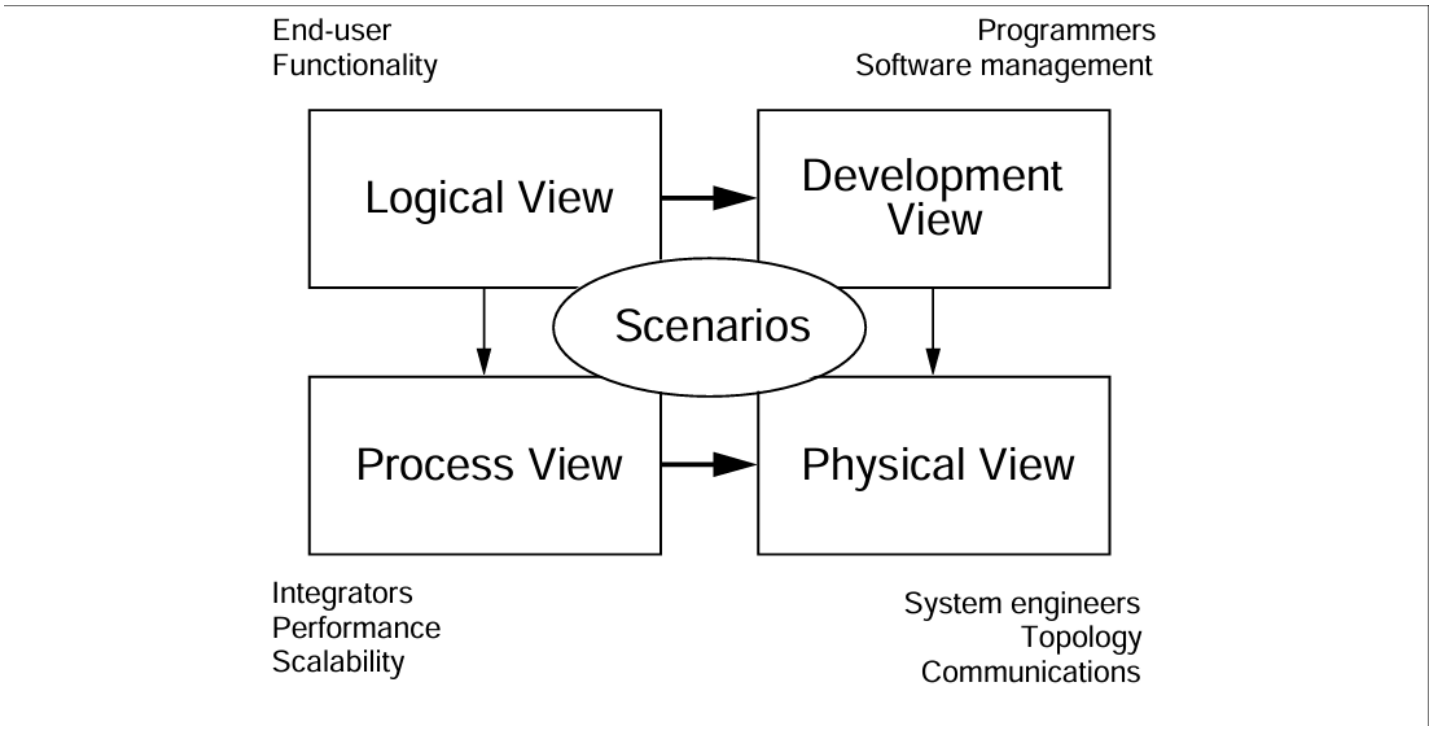
A floorplan carved into a drawing tablet excavated in Umma, dated to the Ur III period (c. 2100–2000 BC). Nearly 5,000 years old.

cdli.mpiwg-berlin.mpg.de/artifacts/125392

Views and Purposes

- Every view should align with a purpose.
- Views should only represent information relevant to that purpose; abstract away other details.
- Annotate a view to guide understanding where needed.
- Different views are suitable for different reasoning aspects (different quality goals), e.g. performance, extensibility, security, scalability, …

The "4+1" View Model

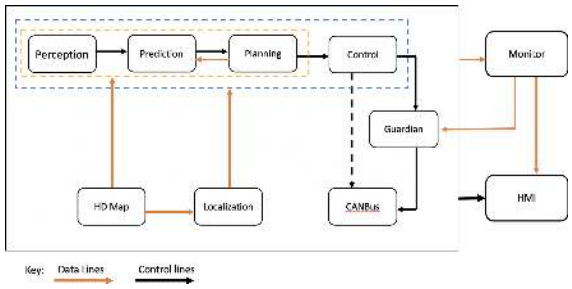


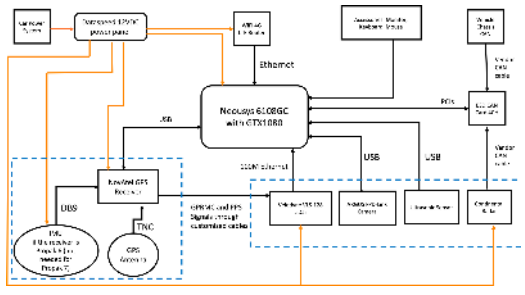
Kruchten, P. Architectural Blueprints: The '4+1' View Model of Software Architecture. 1995.

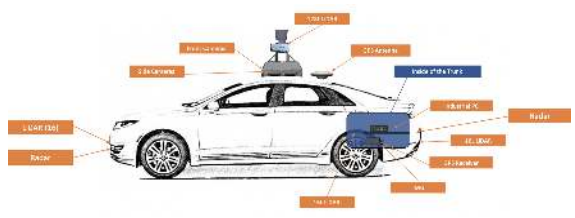
Common Views in Documenting Software Architecture

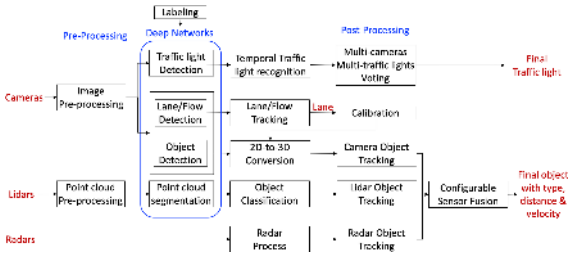
View	Audience	Covers
Logical	End user	Functionality: subsystems, structures, and their relations (dependencies, …).
Process	System integration	Non-functional aspects: components (processes, runnable entities) and connectors (messages, data flow, …).
Development	Developers	Software modularity and decomposition.
Physical / Deployment	System engineer / DevOps	Hardware structures and their connections.
Scenarios	All	Use cases; sequences of interactions between objects and processes.

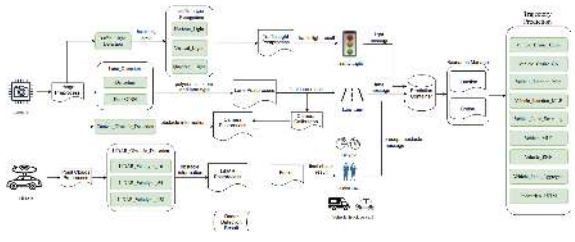
Revisit: Which View Is This?

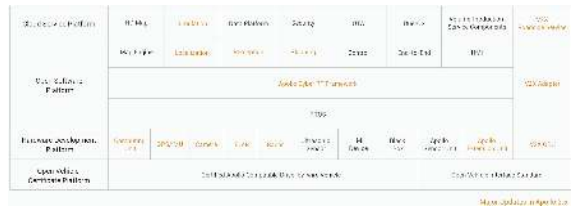








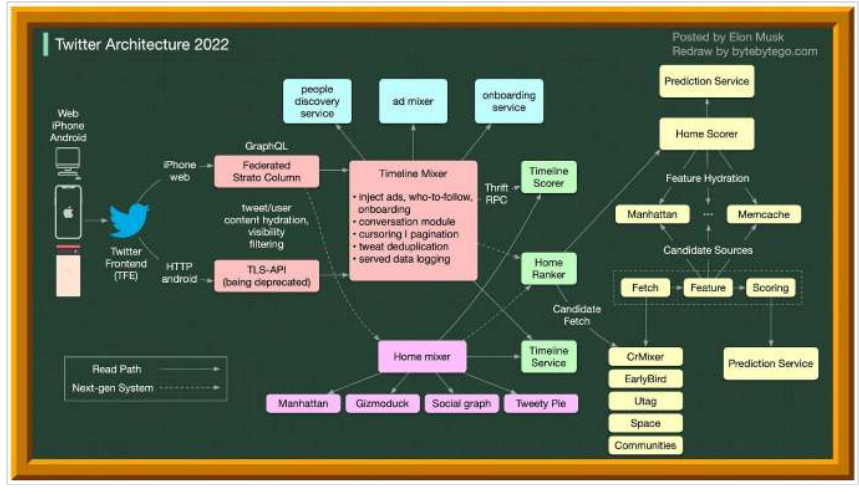




Now that we have a vocabulary: which of logical, process, development, or physical is each Apollo diagram?

Even Elon Musk Reaches for an Architecture Diagram

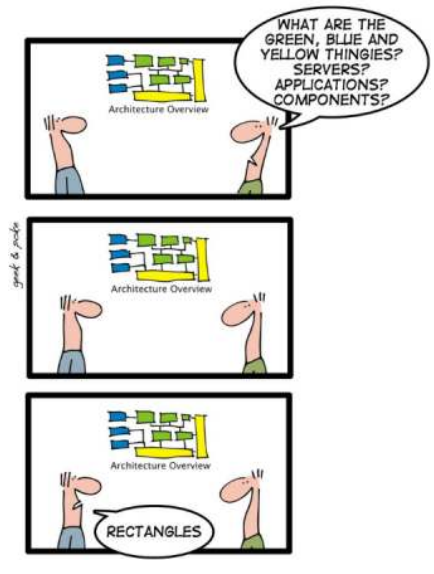




"Just leaving Twitter HQ code review." Days later: "Btw, I'd like to apologize for Twitter being super slow in many countries. App is doing >1000 poorly batched RPCs just to render a home timeline!"

Guidelines for Selecting a Notation

- Suitable for purpose; often visual, for compact representation.
- Usually boxes and arrows.
- UML is possible (semi-formal), but can be constraining.
- Note the abstraction level: subsystems or processes, not classes or objects.
- Formal notations are available.
- Decompose diagrams hierarchically and into views.
- Always include a legend; define precisely what the boxes mean and what the lines mean.
- Do not try to do too much in one diagram: each view should fit on a page. Use hierarchy.



Learning Goals, Revisited

- 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.
- Explain the importance of architectural decisions.
- Integrate architectural decisions into the software development process.
- Document architectures clearly, without ambiguity.