Architectural Design

CSCI 4448/5448: Object-Oriented Analysis & Design

Lecture 38

Before we start, Next Steps

- Project 6 **due today Wed 11/17 at 8 PM**
  - Project 6 includes Zoom or in-person code demonstration sign-ups
  - Sign up for a Project 6 demo slot here: https://docs.google.com/document/d/1tsO0Tf5EIEs3elwQgh3n1RyiRY3dduhfuZ0YIhEVz2g/edit?usp=sharing
- Graduate Pecha Kucha **due Mon 11/29 at 10 AM**
  - **You must sign up for a Pecha Kucha class presentation slot here:** https://docs.google.com/document/d/1EGvA3ZnKVhheJqdRUp7obqG7n3sLQPqCtV8mxwJhB0U/edit?usp=sharing
- Project 7 due Wed 12/8 at 8 PM
  - Includes report, code, and recorded demonstration
- Graduate Final Research Presentation due Wed 12/8 at 8 PM
  - Details on assignments in Canvas Files/Class Files
- New Quiz **due today Wed 11/17**
  - Last quiz will open next weekend, will be due 12/1
- No OOAD classes the week of 11/22, back 11/29
- No office hours for Bruce this week or next; please contact me if you need any support
- Coming soon…
  - APIs
  - Anti-/Other Patterns
  - Graduate Pecha Kuchas
  - Wrapping up
- Posted extra bonus points from coding exercise
- New Piazza topic this week for your comments for Participation Grade
- Piazza article posts now available for extra bonus points (if you're not getting points in class); will add those bonus points in after 12/8
- Find a class staff member if you need anything!

Acknowledgement & Materials Copyright

- I'd like to start by acknowledging Dr. Ken Anderson
- Ken is a Professor and the Chair of the Department of Computer Science
- Ken taught OOAD on several occasions, and has graciously allowed me to use his copyrighted material for this instance of the class
- Although I will modify the materials to update and personalize this class, the original materials this class is based on are all copyrighted © Kenneth M. Anderson; the materials are used with his consent; and this use in no way challenges his copyright

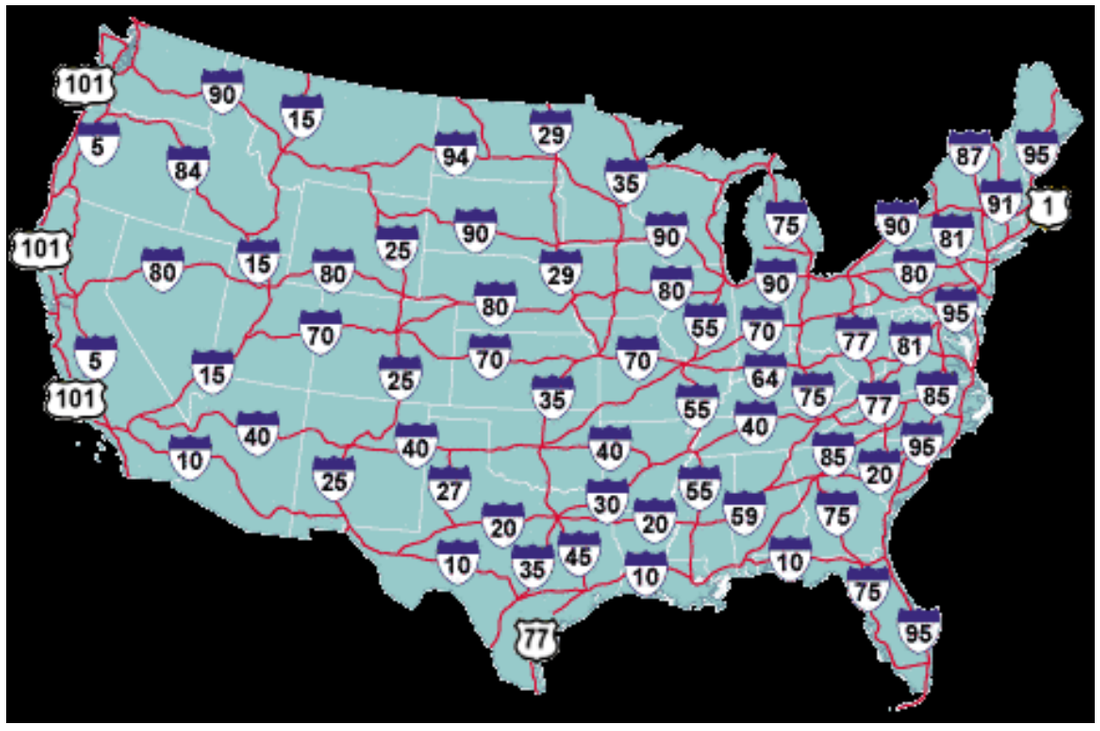
Also some bits taken from graduate projects class architecture presentation by Trevor DiMartino, 2018

Goals of the Lecture

- Review best practices for architectural design
  - Definitions
  - Goals
  - Approaches
  - Methods

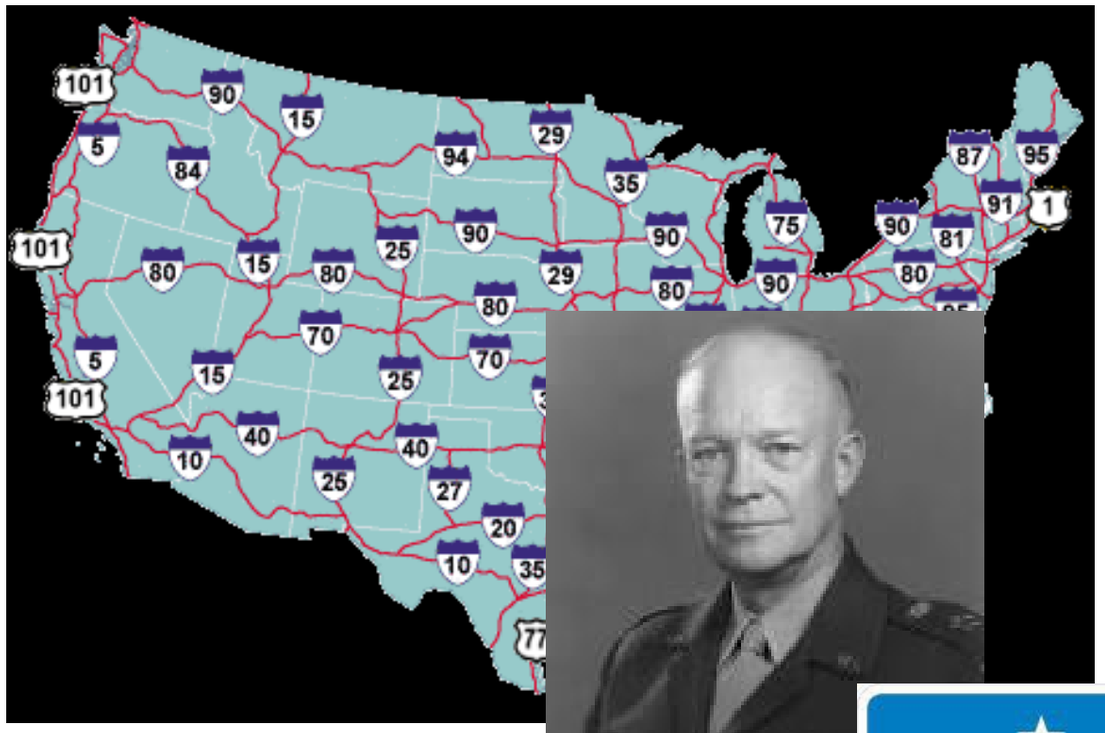
Consider…

- The CU Campus
- The US Interstate System



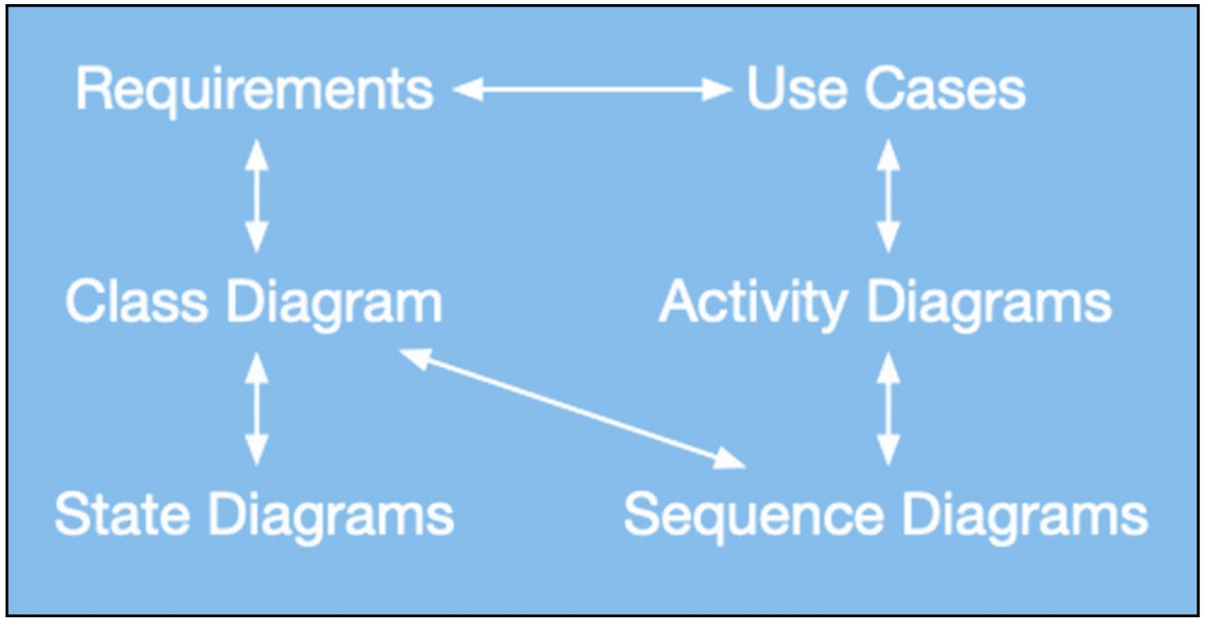
Consider…

- The CU Campus
  - 1919 Campus Development Plan
- The US Interstate System
  - Eisenhower - Federal Aid Highway Act of 1956
  - 2018 25% of all vehicle traffic on Interstate



Consider our design approaches so far…

- Approaches we've discussed
  - The OO A&D/UML Iteration Approach
    - Requirements to use cases to UML diagrams in iterative cycles
  - Design Pattern-Driven Design
    - aka Thinking in Patterns
  - Commonality and Variability Analysis
  - Analysis Matrix
  - CRC Cards
- Fairly fine grained…
  - Developing individual programs or small systems with objects and relationships



Class Name	
Responsibilities	Collaborators

System-Level Architecture

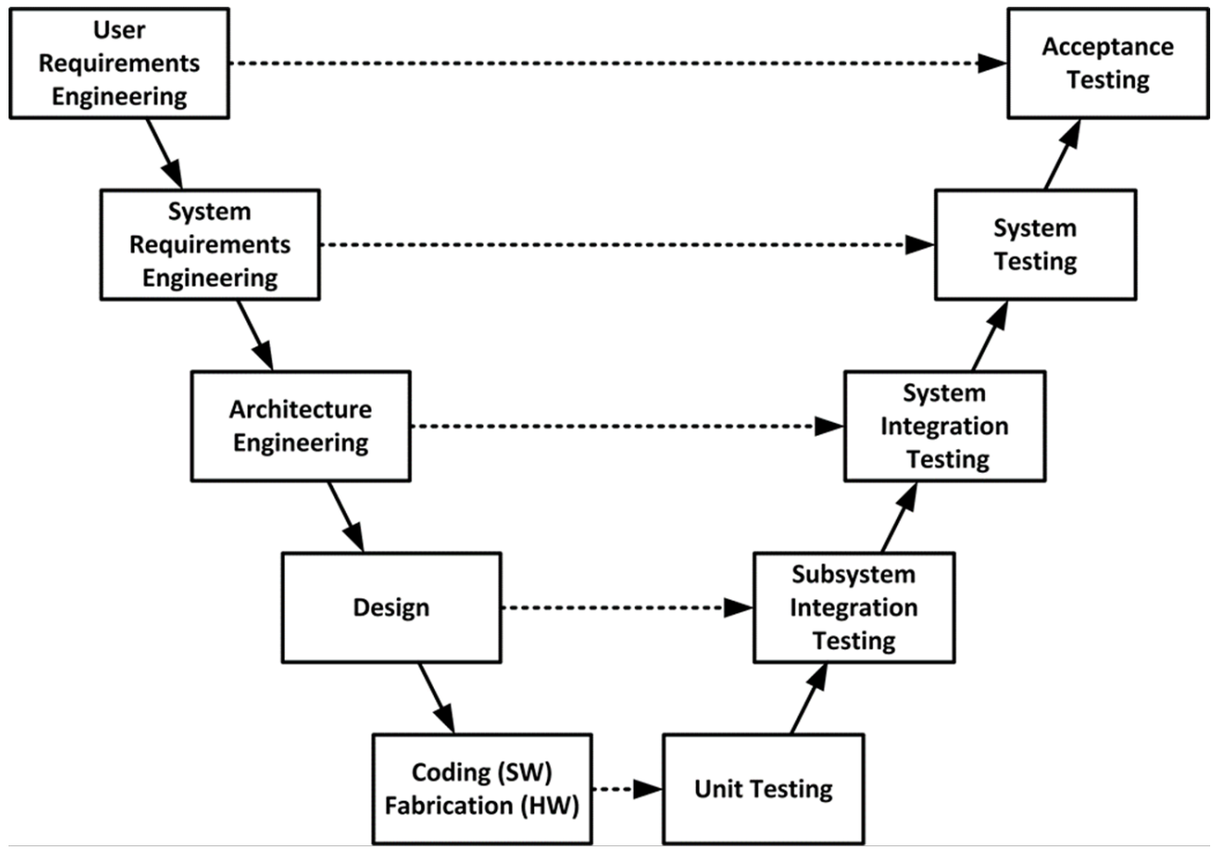
- Larger scale problems
- Set of global project decisions/constraints
- Structures and patterns to be realized with modules
- Structure and flow of components in use
- Views of the system from multiple perspectives

- From Martin Fowler:
  - The shared understanding that the expert developers have of the system design
  - The decisions you wish you could get right early in a project
  - Architecture is about the “important stuff; whatever that is”
  - https://martinfowler.com/architecture/

- Your approach to architecture may vary by the complexity of the project

Architecture in Software

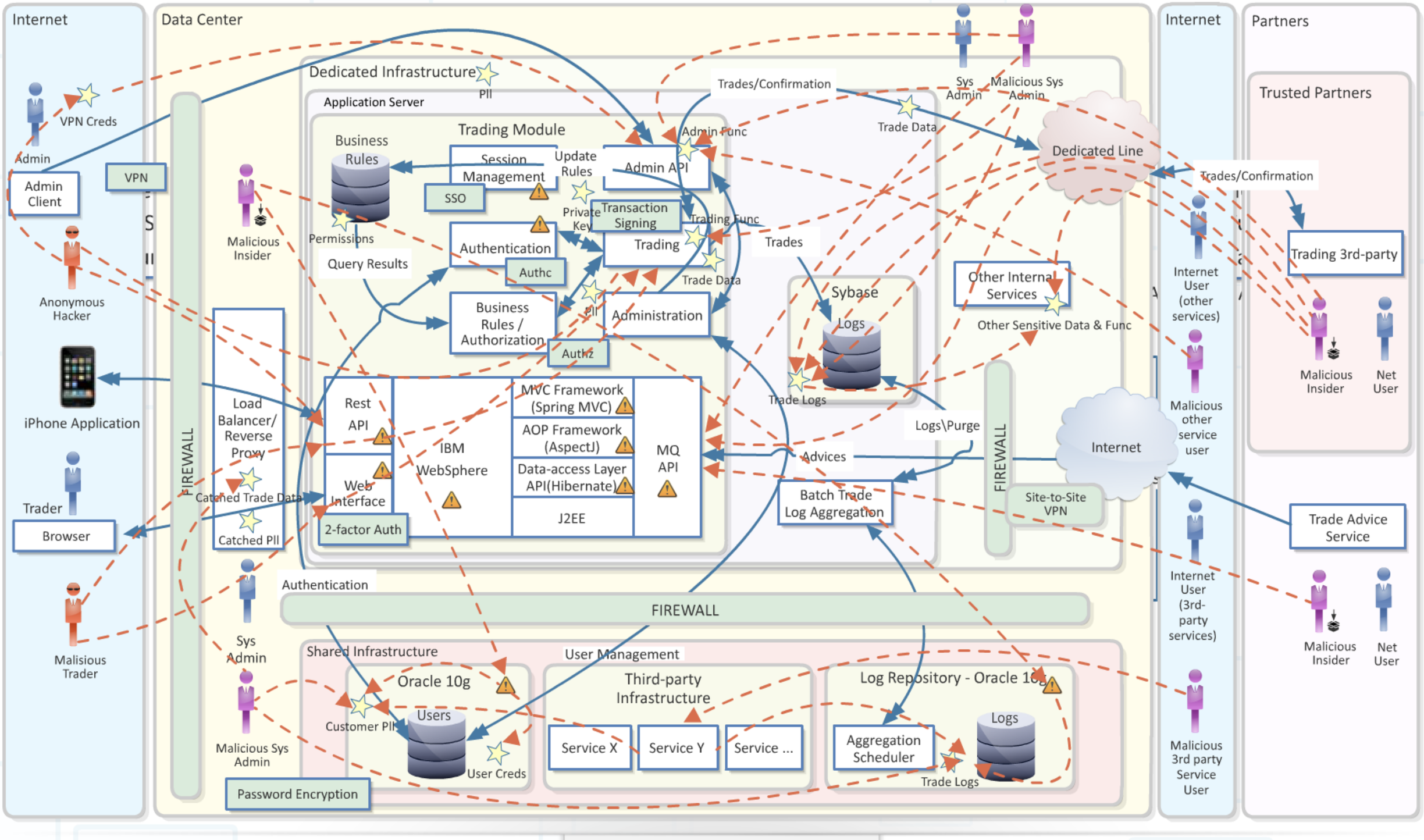
- Architecture: A phase in the software lifecycle, between user/system requirements and detailed design work
- System Architecture: The form and structure of the system
  - Verified with system integration and overall system tests
- Architectural Patterns: Repeatable solution to common software system problems



https://insights.sei.cmu.edu/sei_blog/2013/11/using-v-models-for-testing.html

Planning an Architecture

- In building architecture, blueprints are drafted showing the system to be built from different perspectives
  - Floor plan
  - Electrical
  - HVAC
  - Plumbing
- In software, these might be:
  - Code view
  - Run time behavior view
  - Data flow view
  - Etc.
- In fact, Project 4 forced you to look at multiple views of your design
- Why multiple views?

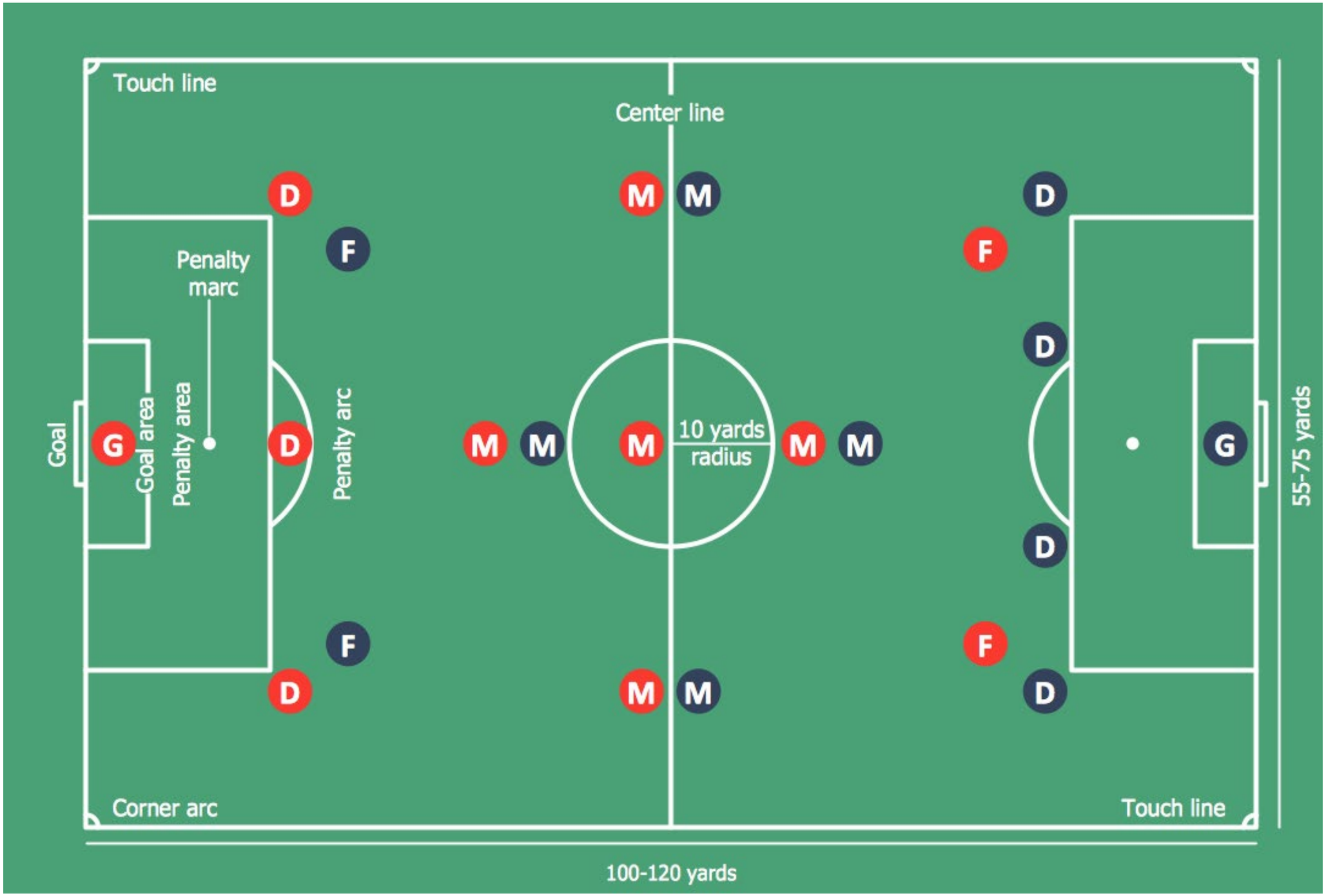


Usually too complicated for a single view…

Keys: Simplicity and Consistency

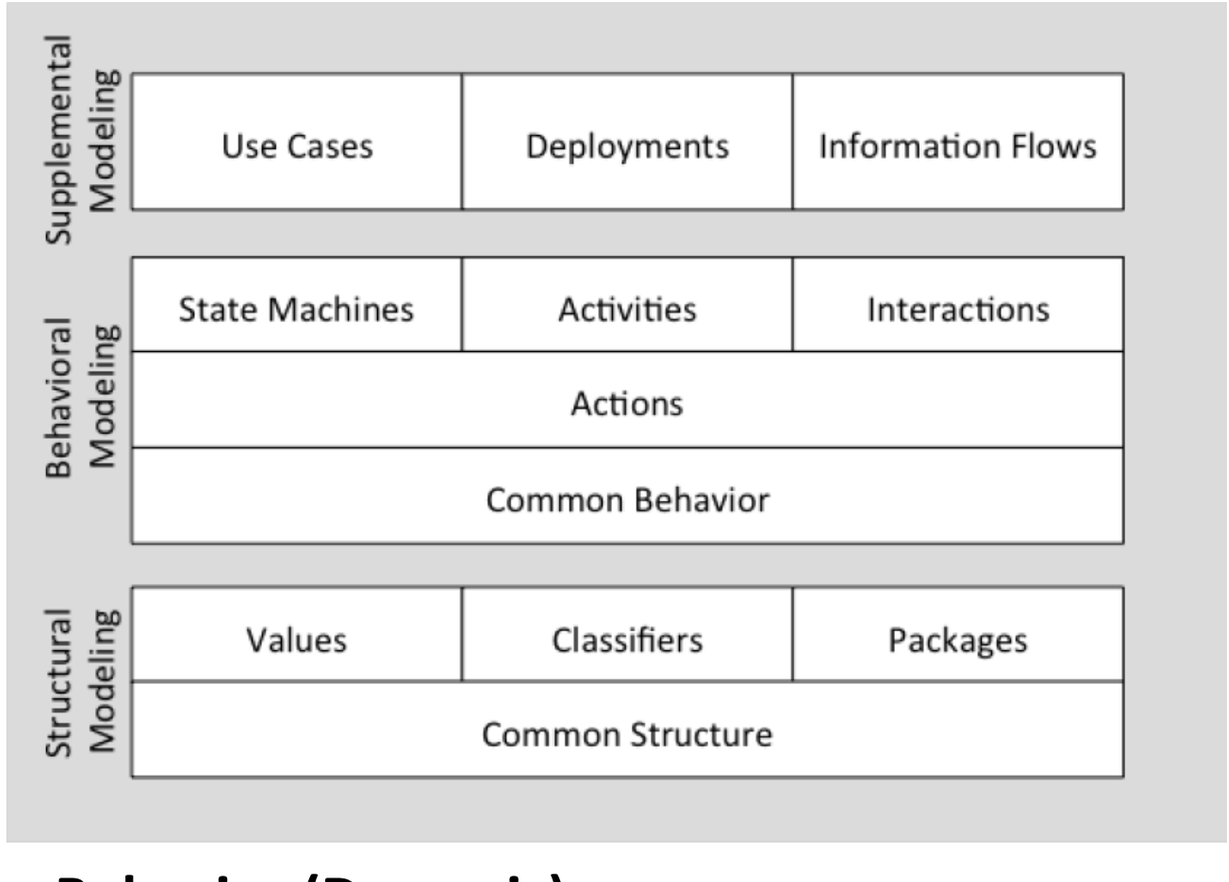
- Prior examples – clear system goals
  - CU = Campus infrastructure that is visually consistent and supports University operations
  - Interstate = Transportation system to support interstate transport for defense and commerce
- Simplicity
  - **Use multiple viewpoints or perspectives to specify architecture**
  - Separation of concerns for local consideration and optimization
  - Complexity will mask underlying issues and mistakes
- Consistency
  - Enhances system understanding
  - Discoveries of commonality and behavior
  - Unnecessary diversity in system may lead to issues

Does this define a soccer game?



Why do UML Diagrams Provide Multiple Views

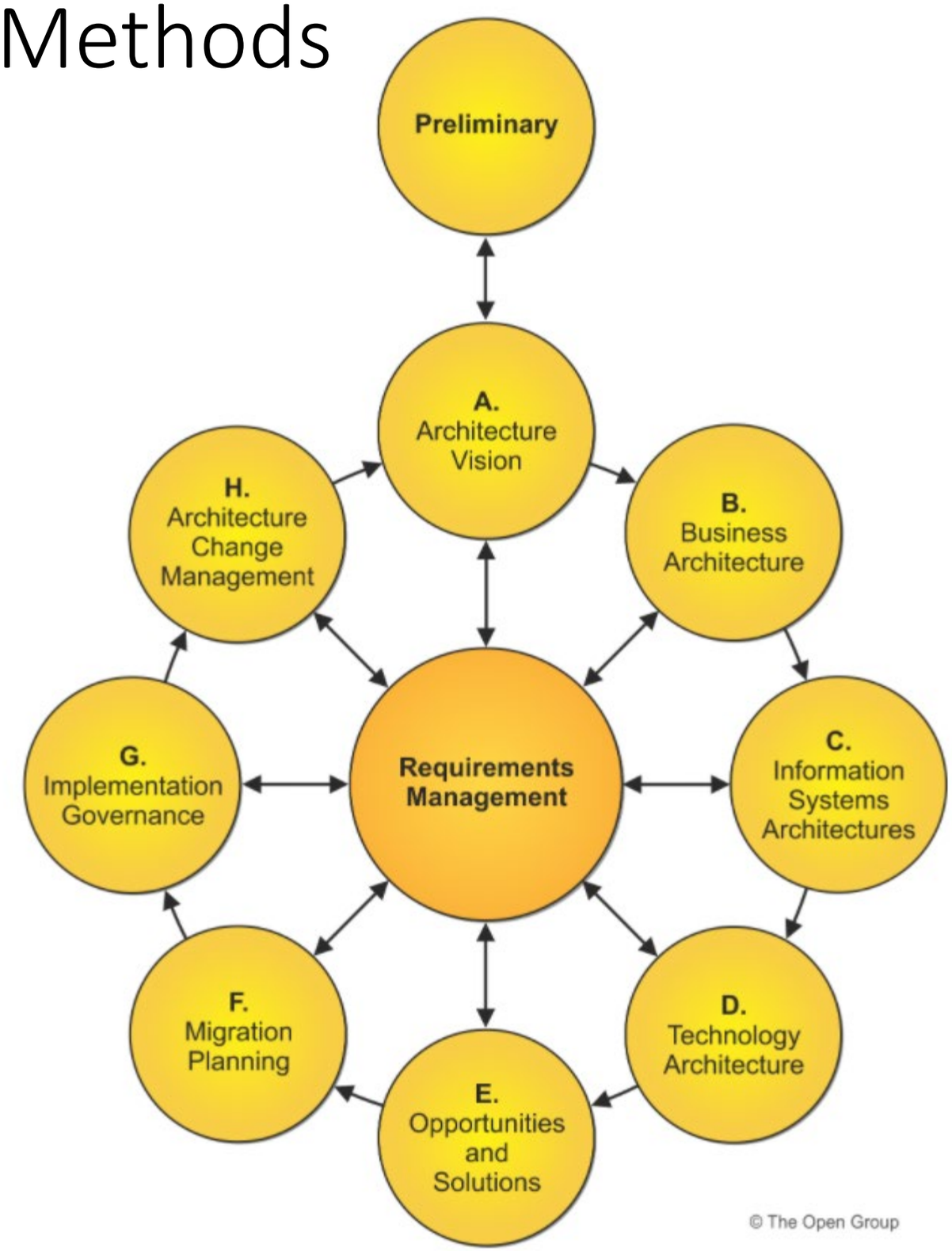
- Diagrams from the current UML release (https://www.omg.org/spec/UML/2.5.1/PDF)
- **Structural (Static)**
  - Class
  - Object
  - Package
  - Model
  - Composite Structure
  - Internal Structure
  - Collaboration Use
  - Component
  - Manifestation
  - Network Architecture
  - Profile
- **Supplemental (both structural and behavioral elements)**
  - Use Case
  - Information Flow
  - Deployment



- **Behavior (Dynamic)**
  - Activity
  - Sequence
  - State (Machine)
  - Behavioral State Machine
  - Protocol State Machine
  - Interaction
  - Communication (was Collaboration)
  - Timing
  - Interaction Overview

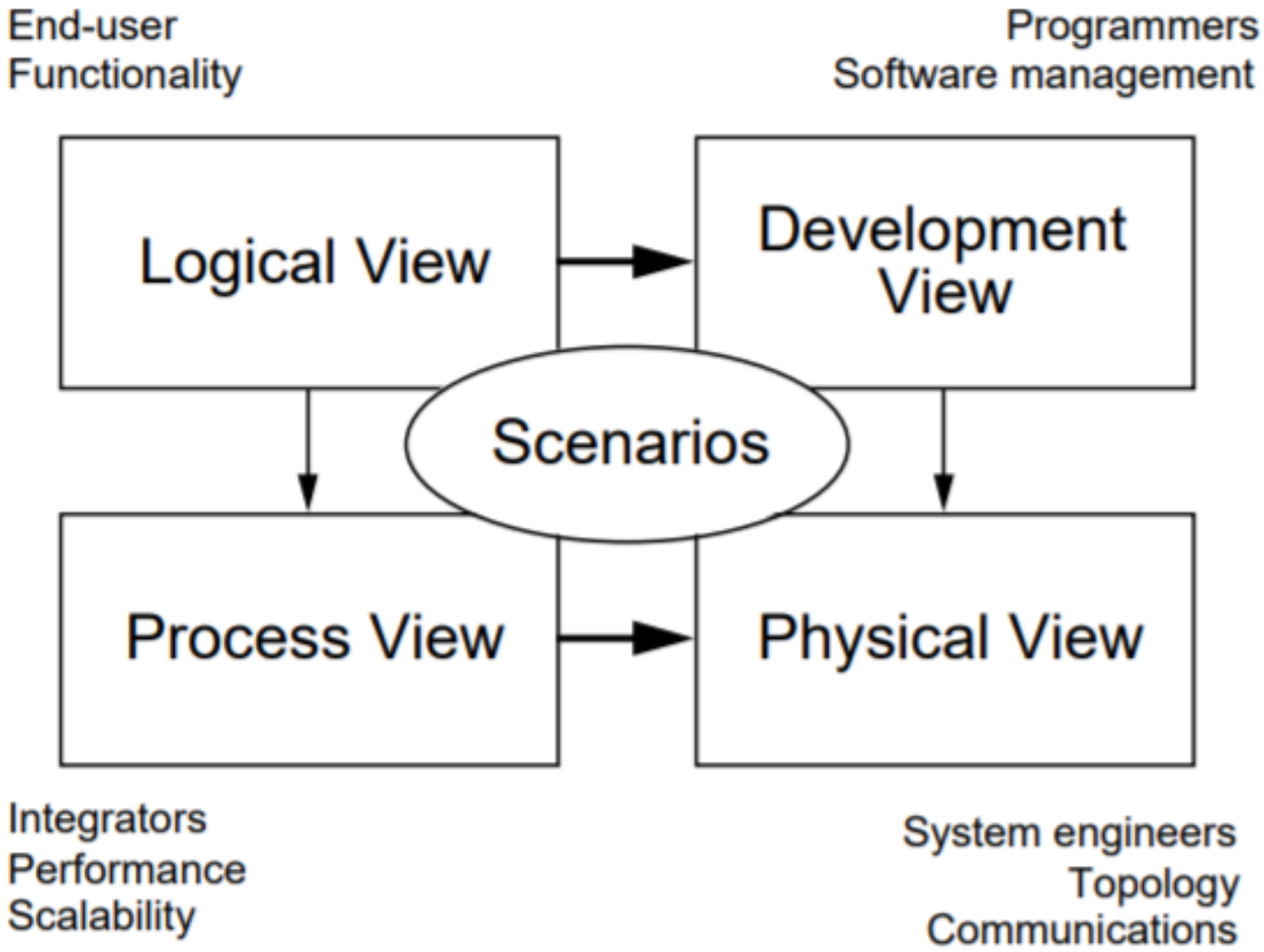
Architecture Modeling Methods

- Structuring our approach to defining our views of the system
- **Provide different perspectives**
- Common Methods
  - Iterative UML-based Design
  - 4+1
  - C4
  - Others
    - TOGAF →
      - The Open Group Architecture Framework – process for enterprise level architecture
      - https://www.opengroup.org/togaf
    - Arc42
      - Template-based with multiple views for architecture description
      - https://arc42.org/overview/



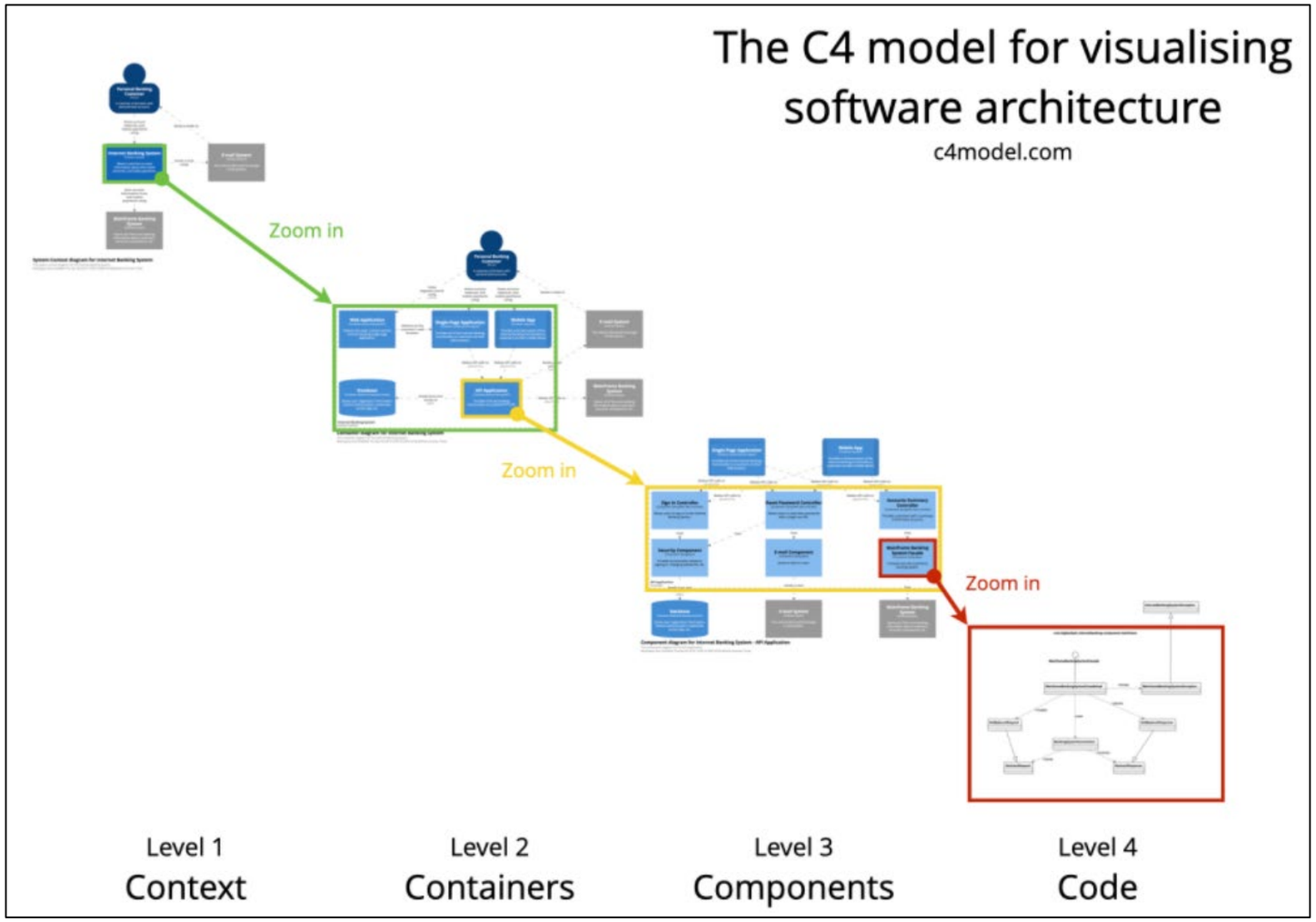
4+1 Model

- By Phillipe Kruchten, 1995, in development at Rational around the time of UML
- Use cases at the core, different perspectives on the system elements in the four views
- Still a useful way to breakdown a complex design into different views
- https://www.cs.ubc.ca/~gregor/teaching/papers/4+1view-architecture.pdf



C4 Model

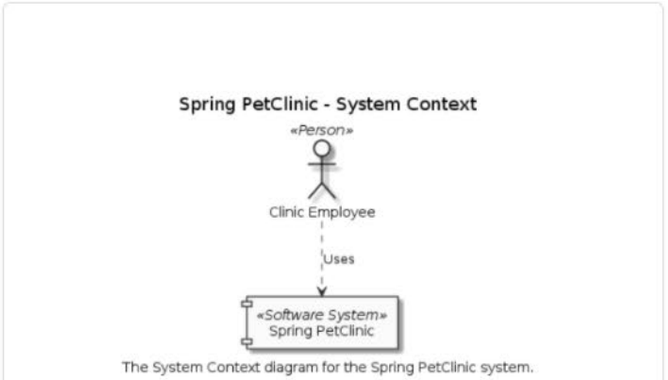
- A response to agile vs. architecture
- More modern view of architectural and progressive design –
  - “Maps of your code”
  - abstraction first
- 4 Levels
  1. System Context – how the system fits into its environment
  2. Containers – high level technical building blocks (not Docker, but different types of apps or data stores)
  3. Components – details of components in containers
  4. Code – UML Class Diagram
- https://c4model.com/



C4 & UML

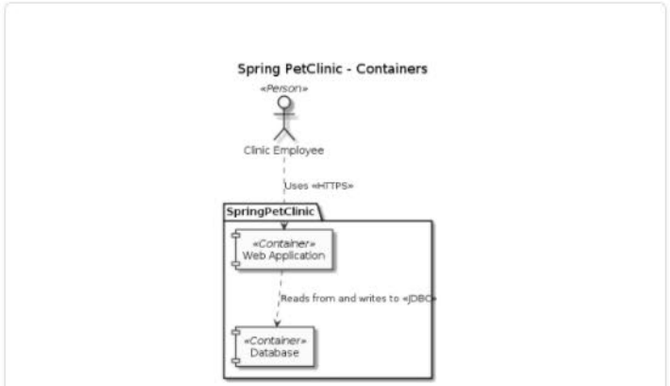
- Common to replace the boxes/arrow models in C4 with appropriate UML diagrams (https://c4model.com/):

System Context diagram



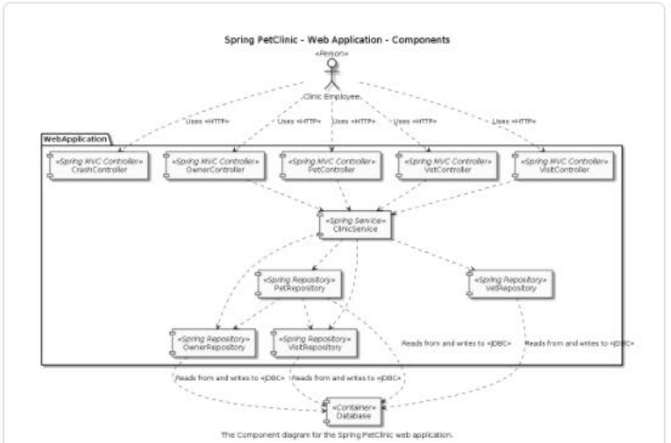
The System Context diagram for the Spring PetClinic system.

Container diagram

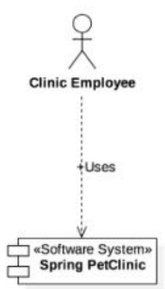


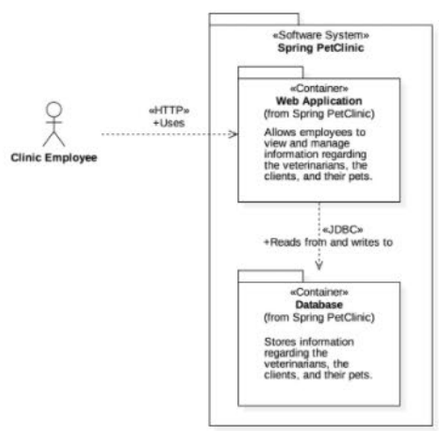
The Container diagram for the Spring PetClinic system.

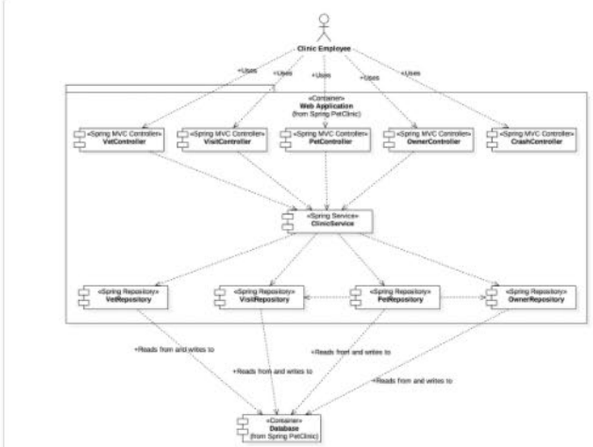
Component diagram



The Component diagram for the Spring PetClinic web application.







Other concerns

- Usability?
- Security?
- Reliability?
- Robustness?
- Scalability?
- Performance?
- Others?
- Often represented in requirements by non-functional entries
- Use different architectural views to consider these questions and your response in the design

Issues in distributed computing:

- Reliable networks
- Latency
- Bandwidth
- Security
- Topology changes
- External administrators
- Cost of transport
- Heterogeneous network elements
- Maintenance/versioning
- Transaction models
- Logging

From Fundamentals of Software Architecture – Richards/Ford

Architecture in Agile

- Likely more iterative than all up-front
  - Doesn't mean we don't design
- Right-size the effort to the application
- Have an Architecture Owner
- Focus on facilitating the evolution of the architecture over time
- Facilitates creation of the architecture
- Plans architectural "spikes" and refactors
- Mentors team members in
  - Coding guidelines (automated standards)
  - Database and data model guidelines
  - Security guidelines
  - Documentation principles

- Architect Skills
  - Architecture direction/decisions
  - Continual analysis
  - Keep current on tools/trends
  - Ensure architectural compliance
  - Diverse exposure/experience
  - Business domain knowledge
  - Interpersonal/communication skills
  - Navigating politics
  - Balance architecture and other concern
  - Engineering practices
  - Operations/Dev(Sec)Ops
  - Process
  - Data Modeling

- From Fundamentals of Software Architecture – Richards/Ford

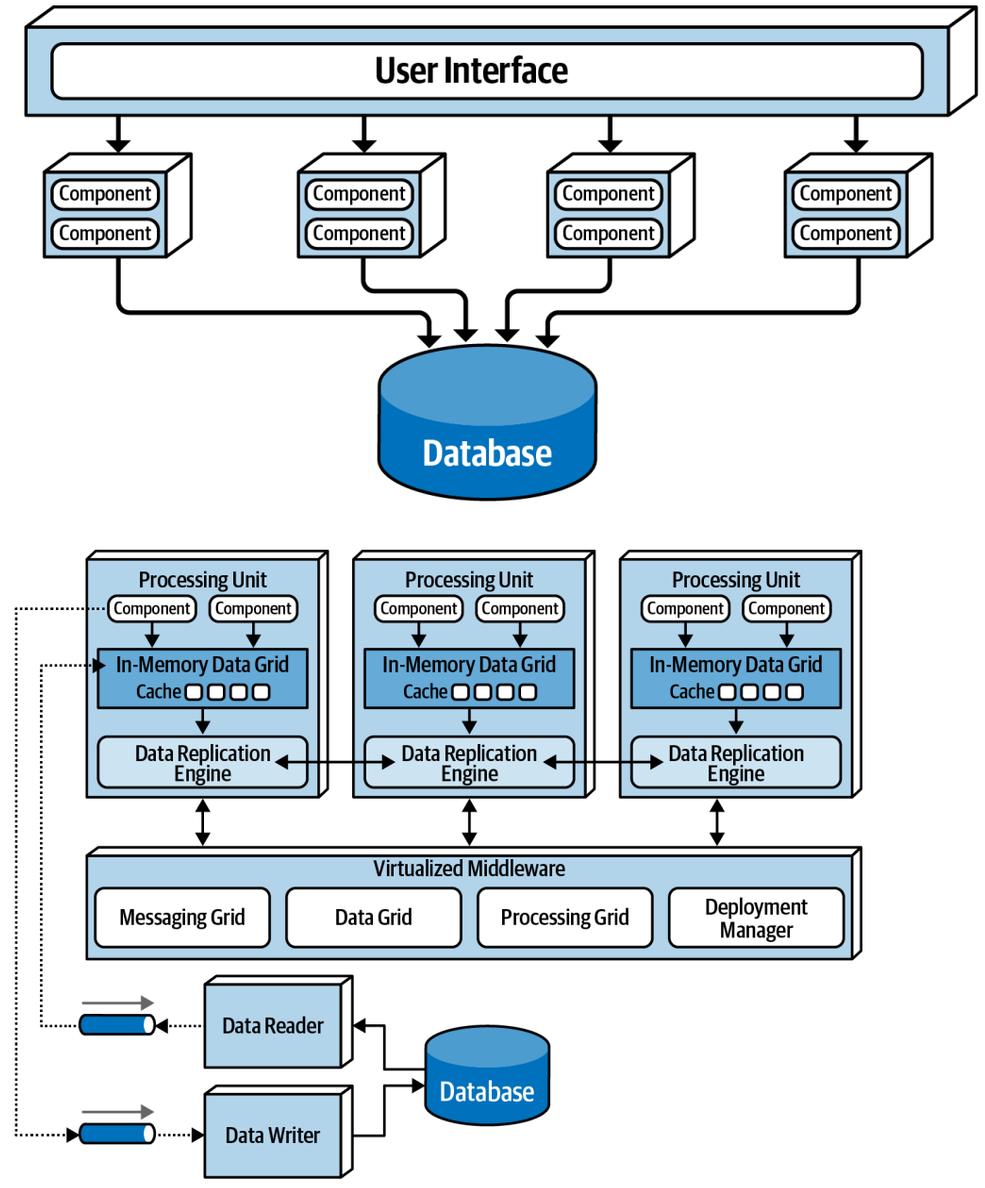
Don’t over-engineer

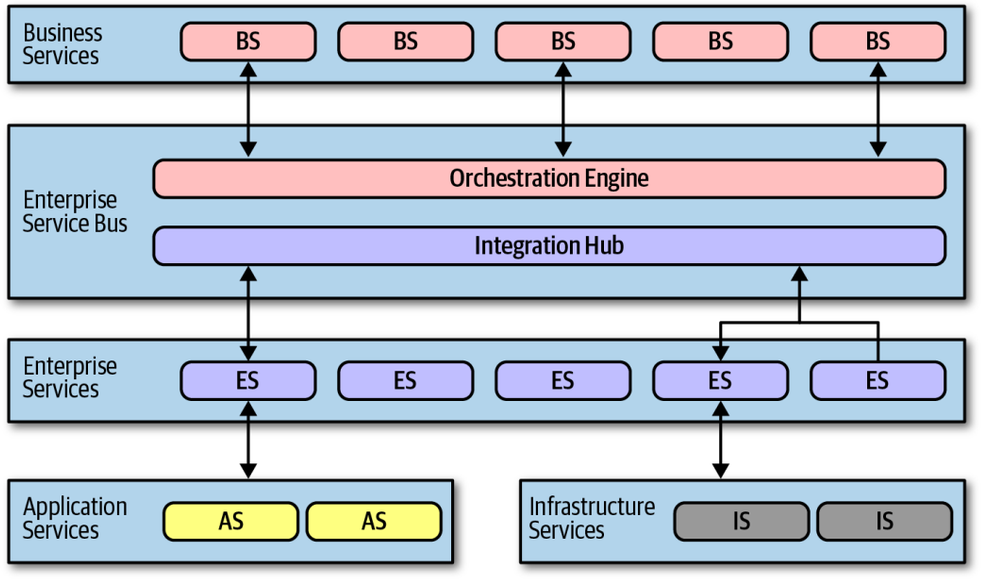
- Many systems we develop are relatively small and don’t require a full architectural plan
- …but they can grow, which often requires re-architecting
- It’s good to think of the future
  - How will the use of your system differ in 10 years?
  - What if your system becomes quickly popular and user numbers multiply by 10?
  - What components of your system would need to change for a new client?
- …but it’s also important to start with something achievable that can be envisioned by the team

Common Architecture Styles

- Big Ball of Mud
- Unitary (typically single embedded systems)
- Client/Server (two-tier, front-end/back-end)
- N-tier (MVC, etc.)
- Monolithic
  - Layered
  - Pipeline
  - Microkernel
- Distributed
  - Service-based (1st figure)
  - Event-driven
  - Space-based (2nd figure)
  - Service-oriented (3rd figure)
  - Microservices
- Styles have their related patterns

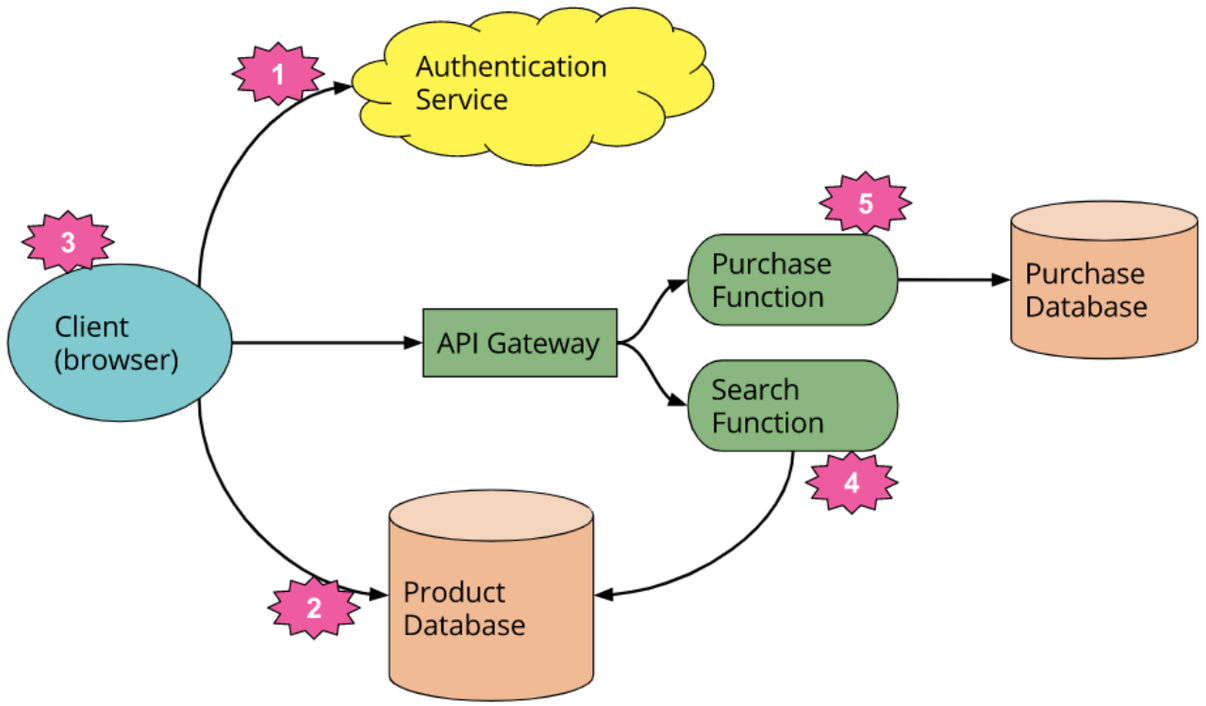
From Fundamentals of Software Architecture – Richards/Ford





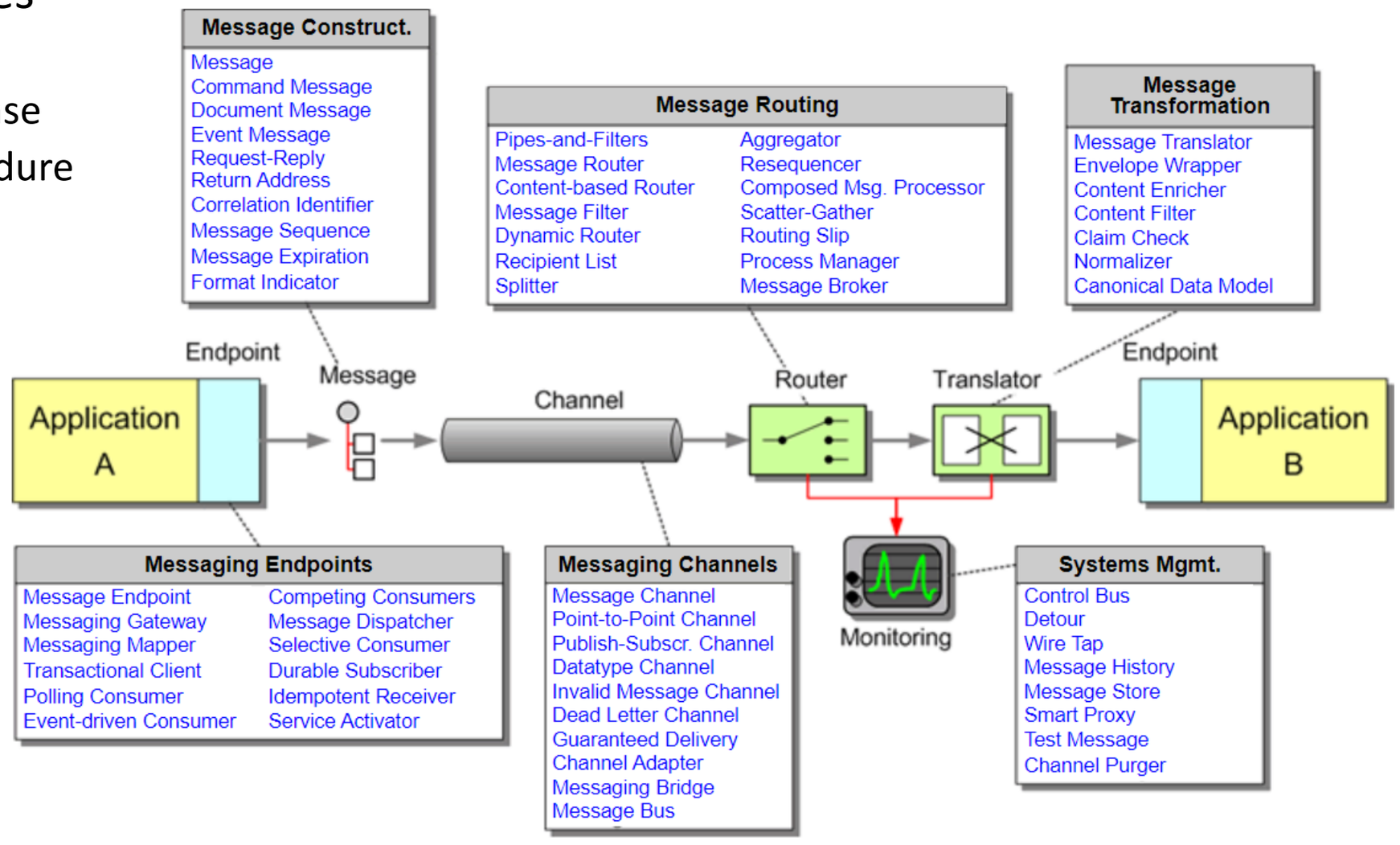
Architecture Patterns

- Patterns – structured solutions to identified common problems
  - Experience reuse (not code)
- High Level
  - Application Boundaries
  - Microservices (next)
  - Serverless Architectures →
  - Micro Frontends
  - GUIs and MVC
  - Presentation/Domain/Data Layering
  - https://martinfowler.com/architecture/
- Pattern Libraries
  - https://www.enterpriseintegrationpatterns.com/
- Cloud-based Architectures
  - AWS, Azure, GCP, etc.
  - Vendor support for solution patterns is generally strong



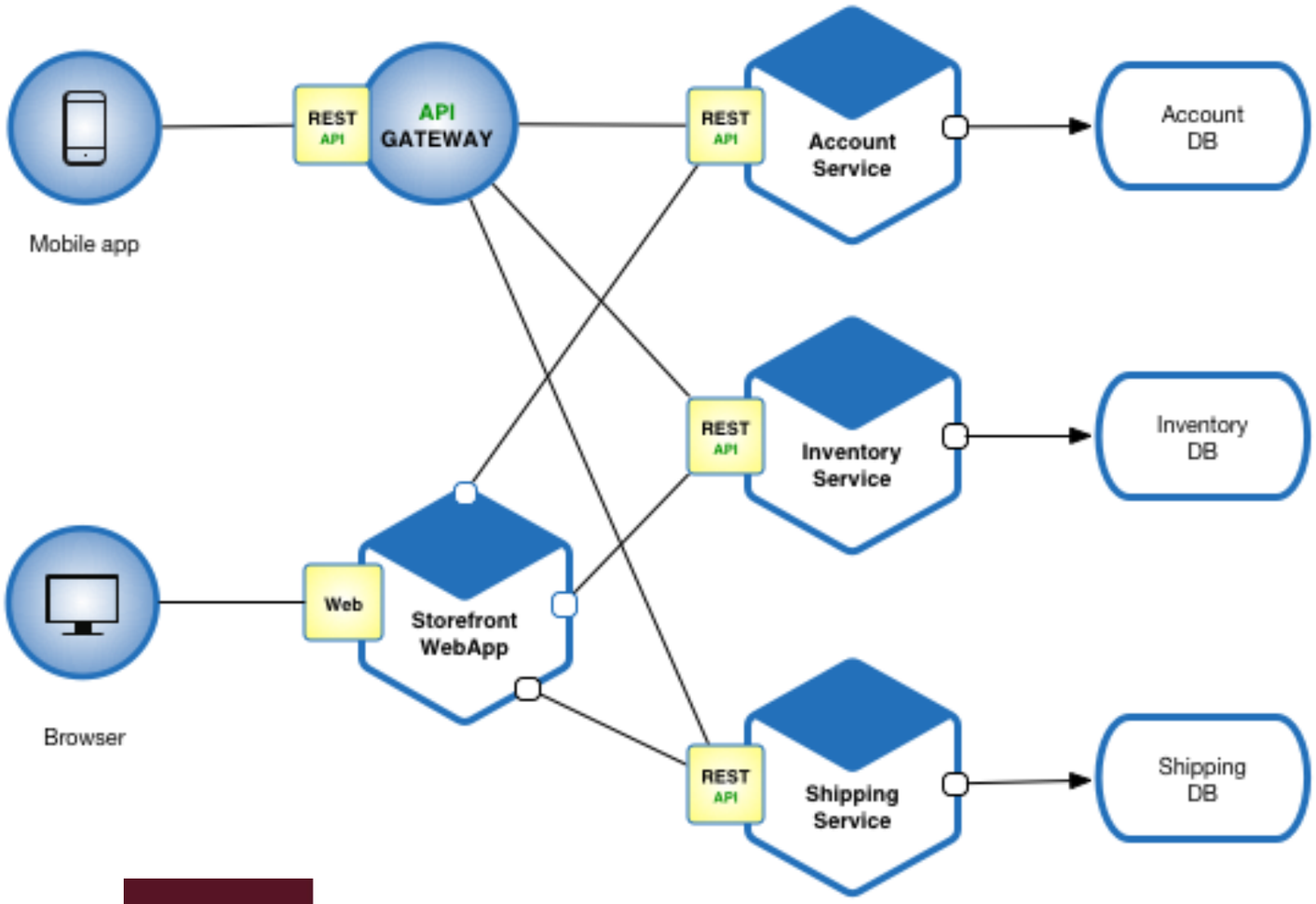
Enterprise Integration Patterns

- https://www.enterpriseintegrationpatterns.com/
  - From the book by Hohpe & Woolf
  - Focuses on integration via messaging (65 messaging patterns)
  - Integration Styles
    - File Transfer
    - Shared Database
    - Remote Procedure Invocation
    - Messaging



Architectural Pattern Example: Microservices

- Microservice – an independently deployable component of bounded scope that supports interoperability through message-based communication
- Microservice architecture - a style of engineering highly automated, evolvable software systems made up of capability-aligned microservices
- Focus on being **Replaceable vs. Maintainable**
- Usually targeted at big systems and their performance
- Goal-oriented architecture
- Drawn from service-oriented architectures (SOA) for modularity and messaging, DevOps for automated deployment, and Agile for responding to identified needs as they arise
- From: Microservice Architecture, Nadareishvili et al., 2016, O'Reilly



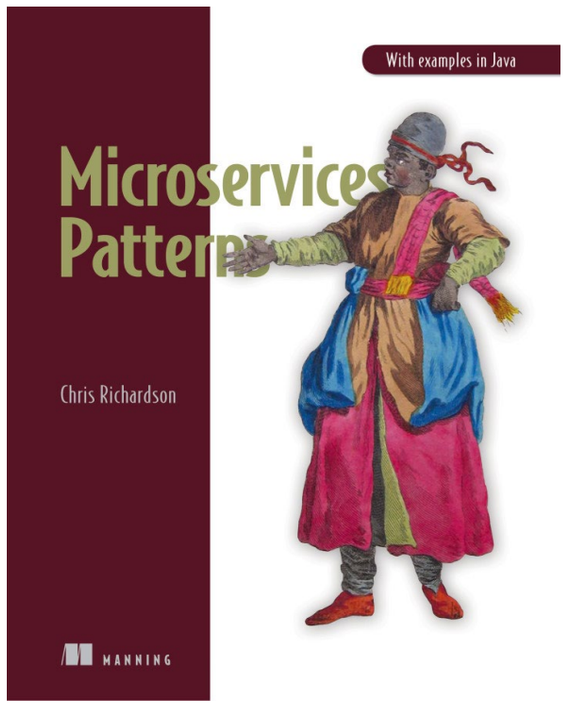
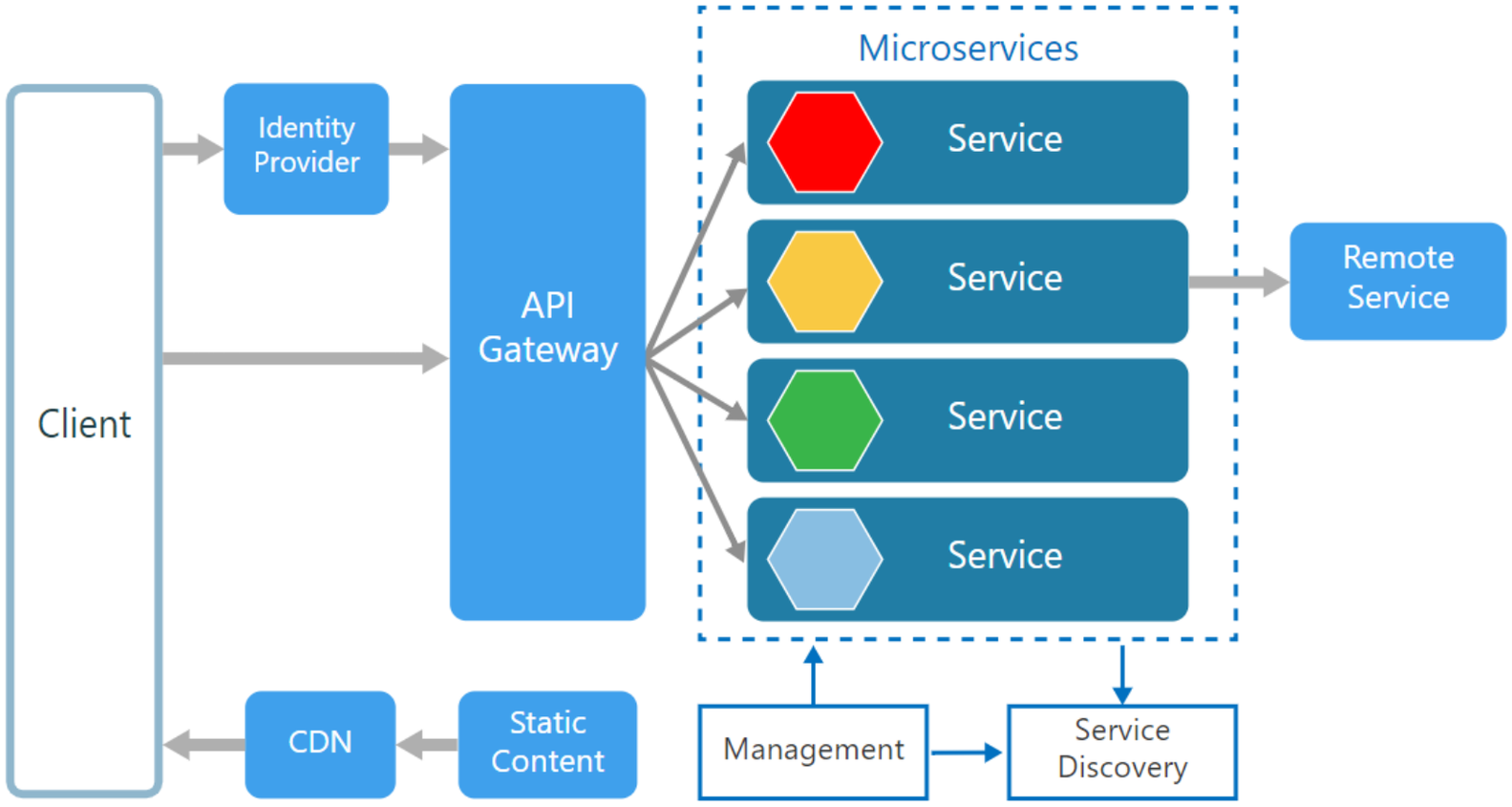


Image from https://microservices.io/patterns/microservices.html

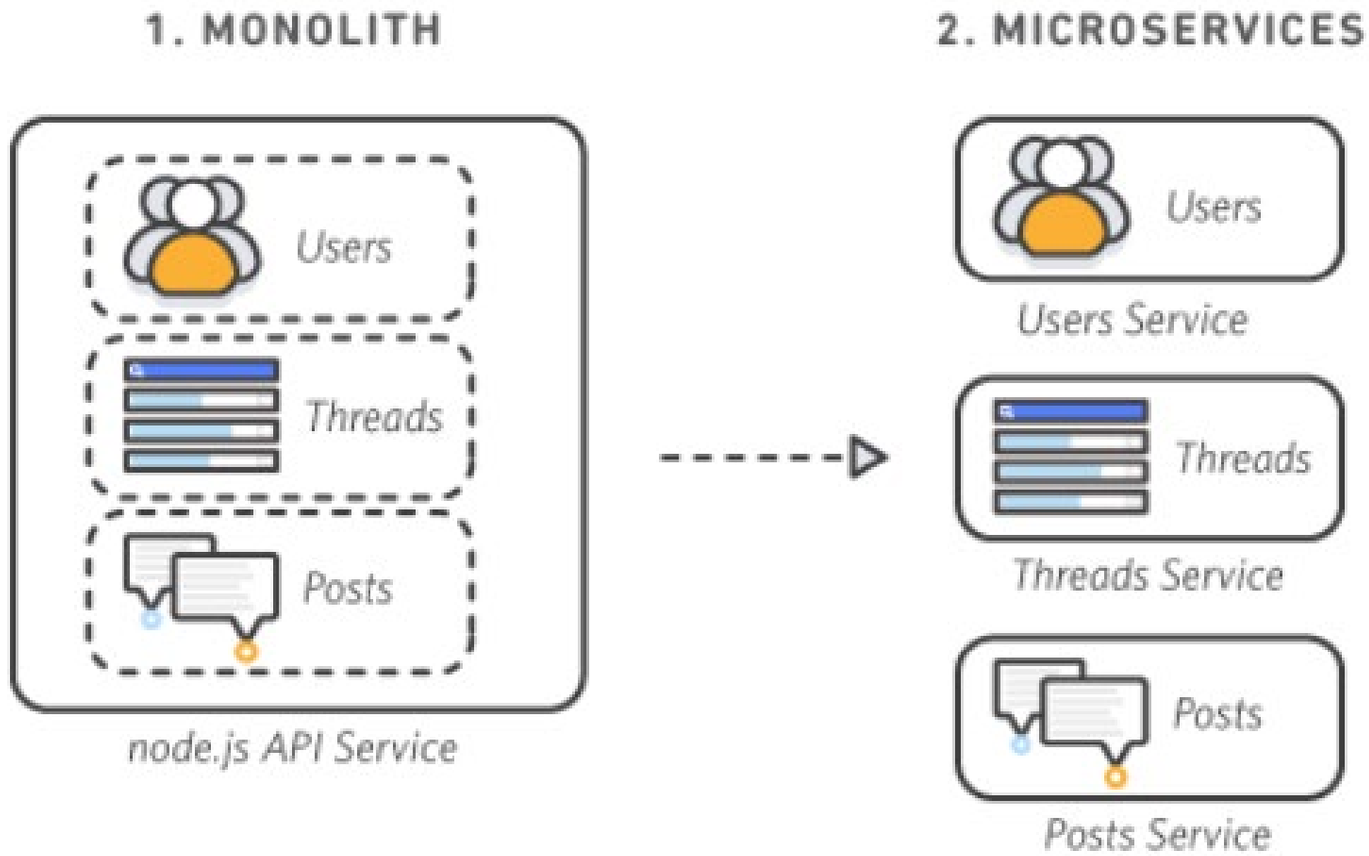
Microservices

- Characteristics
  - Small in size
  - Messaging enabled
  - Bounded by contexts
  - Autonomously developed
  - Independently deployable
  - Decentralized
  - Built and released with automated processes
- Goal – balance speed, safety, and scale
- Focus on API Design
  - Standard API Gateway
  - Containers (e.g. Docker)
  - Service Discovery
  - Standard Security
  - Routing
  - Monitoring and Alerting
- Image from https://docs.microsoft.com/en-us/azure/architecture/guide/architecture-styles/microservices

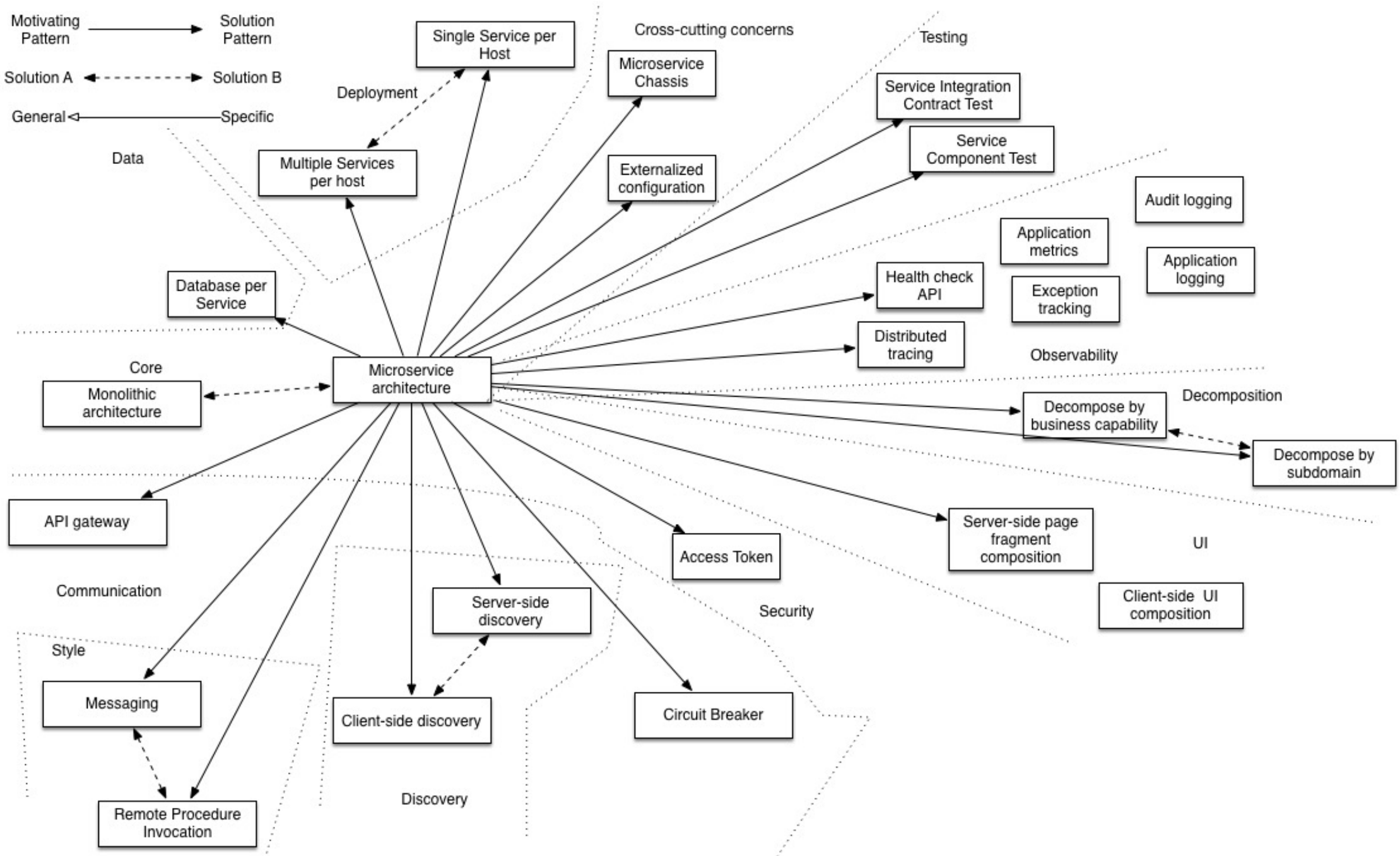


Microservices in the Cloud

- AWS View of Microservices
  - Autonomous
  - Specialized
  - Agile
  - Flexible scaling
  - Easy deployment
  - Technology freedom
  - Reusable code
  - Resilient
- Typical AWS Components
  - ECS – Elastic Container Service
  - Lambda
  - Databases
  - API Gateway
  - SNS
  - SQS
  - Cloudwatch
- https://aws.amazon.com/microservices/



Architectural Patterns from Microservices



- https://microservices.io/patterns/microservices.html

Architecture References

- Martin Fowler's Books
  - Refactoring
  - Patterns of Enterprise Application Architecture
- Richards & Ford
  - Fundamentals of Software Architecture: An Engineering Approach (O'Reilly)
- Hoope & Woolf
  - Enterprise Integration Patterns
    - https://www.enterpriseintegrationpatterns.com/
- Michael Keeling
  - Design It!
    - Process of doing architecture, basic architecture patterns, good introductory book
- Simon Brown
  - Software Architecture for Developers
    - 2 volume e-books on architecture and the C4 method
    - https://leanpub.com/b/software-architecture
- Brown & Wilson
  - The Architecture of Open Source Applications →
    - http://aosabook.org/en/index.html
    - See how popular open source programs were architected (or not)
    - Example: Life cycle of Eclipse architecture