ECE444: Software Engineering

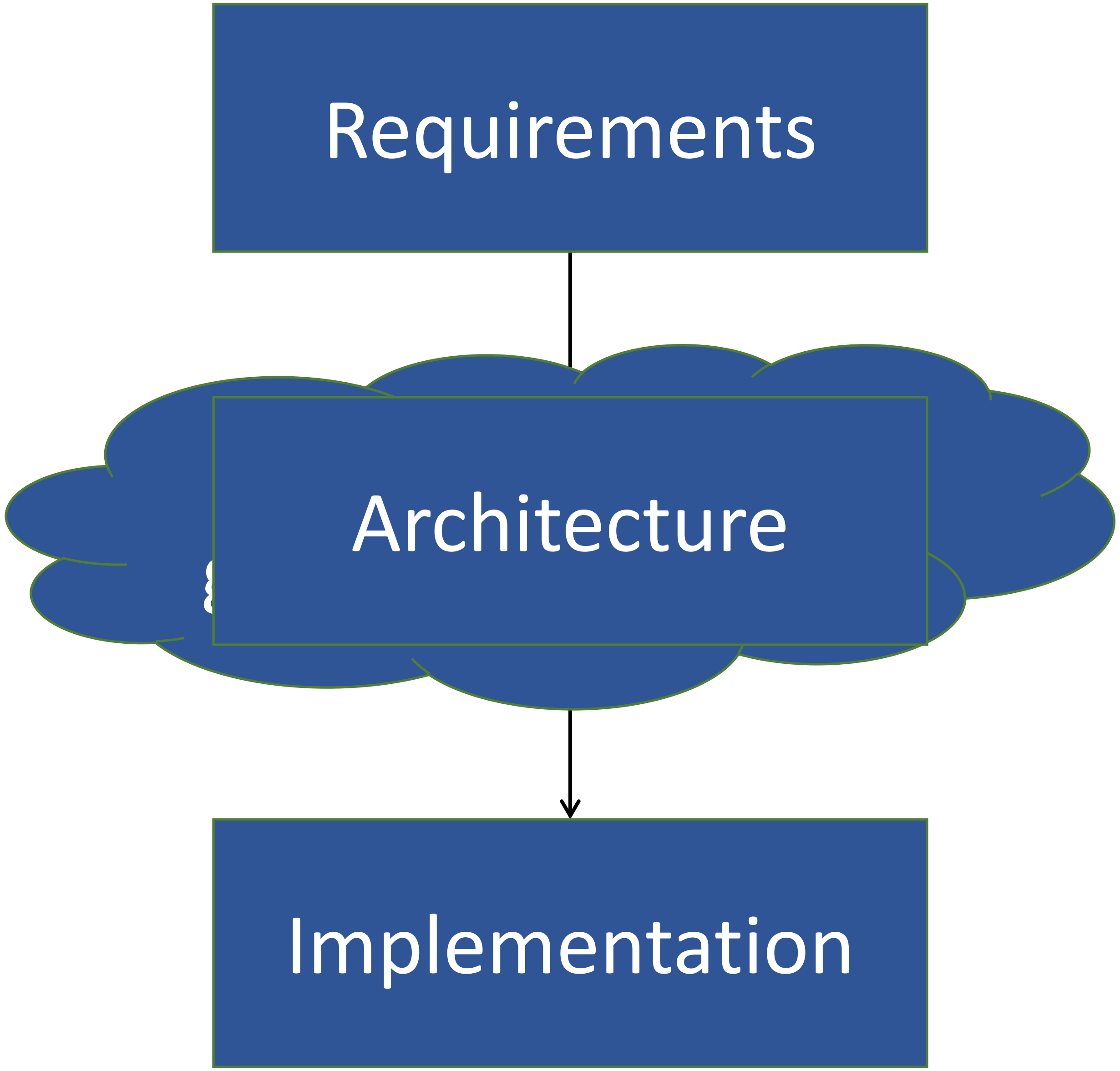
Software Architecture

Shurui Zhou

The Edward S. Rogers Sr. Department
of Electrical & Computer Engineering
UNIVERSITY OF TORONTO

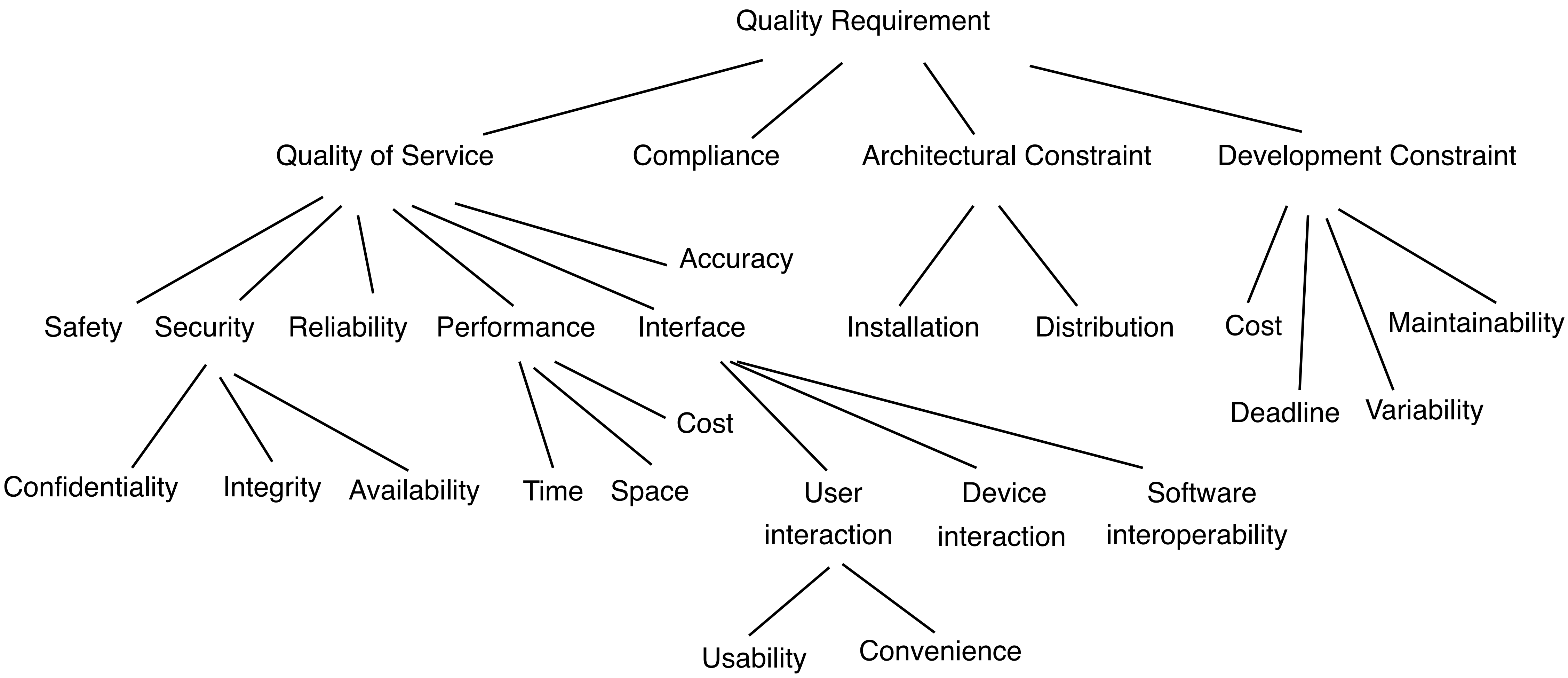
Learning Goals

- Understand what drives design
- Understand information hiding
- Understand the abstraction level of architectural reasoning
- Approach software architecture with quality attributes in mind
- Use notation and views to describe the architecture suitable to the purpose
- Understand a few common architecture patterns



Requirements
Architecture
Implementation

Quality Requirements, now what?



Quality Requirements, now what?

- "should be highly available"
- "should answer quickly, accuracy is less relevant"
- "needs to be extensible"
- "should efficiently use hardware resources"

Introduction to Software Design

A typical Intro of CS design process

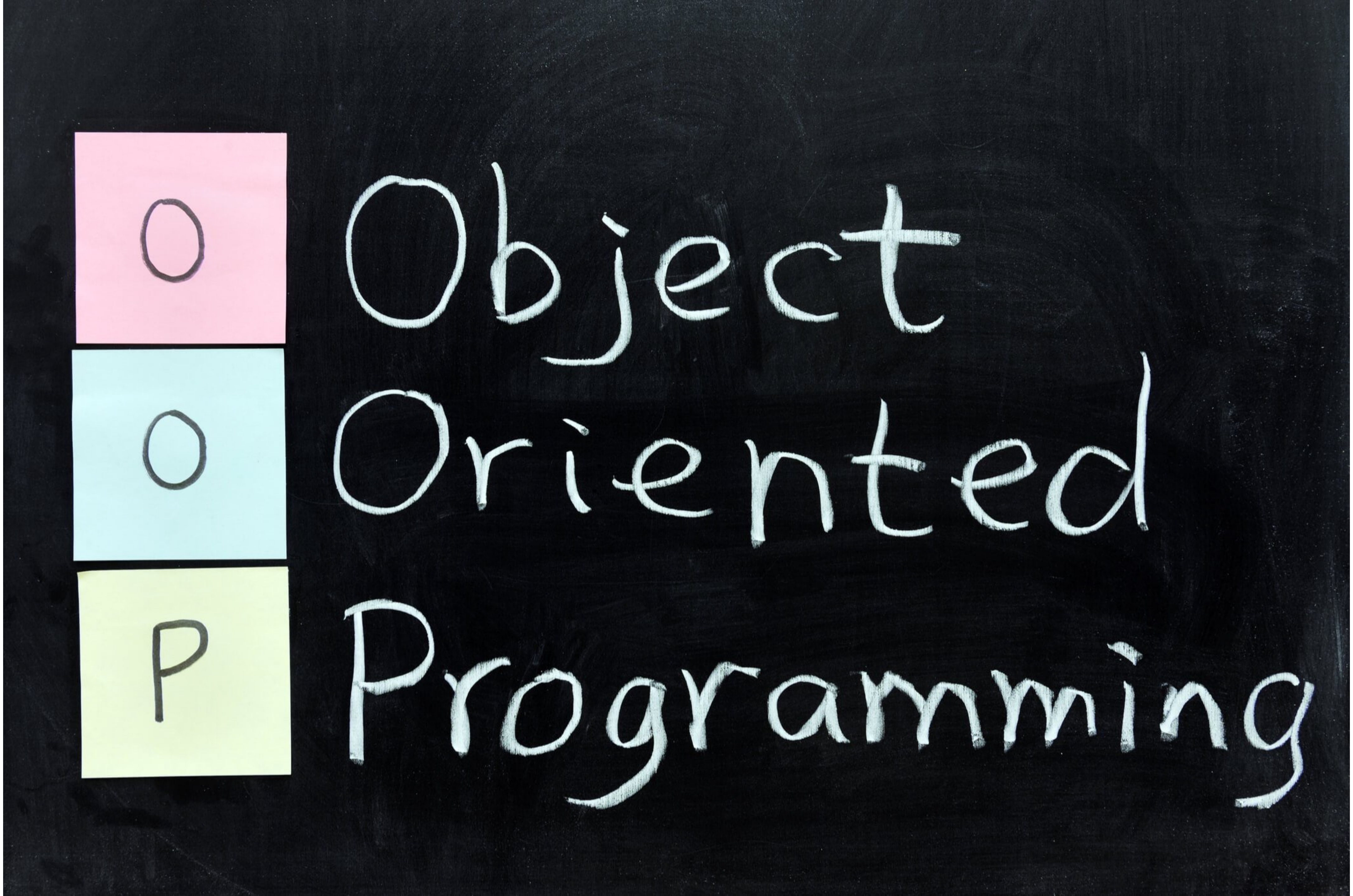
1. Discuss software that needs to be written
2. Write some code
3. Test the code to identify the defects
4. Debug to find causes of defects
5. Fix the defects
6. If not done, return to step 1

A Better Software Design

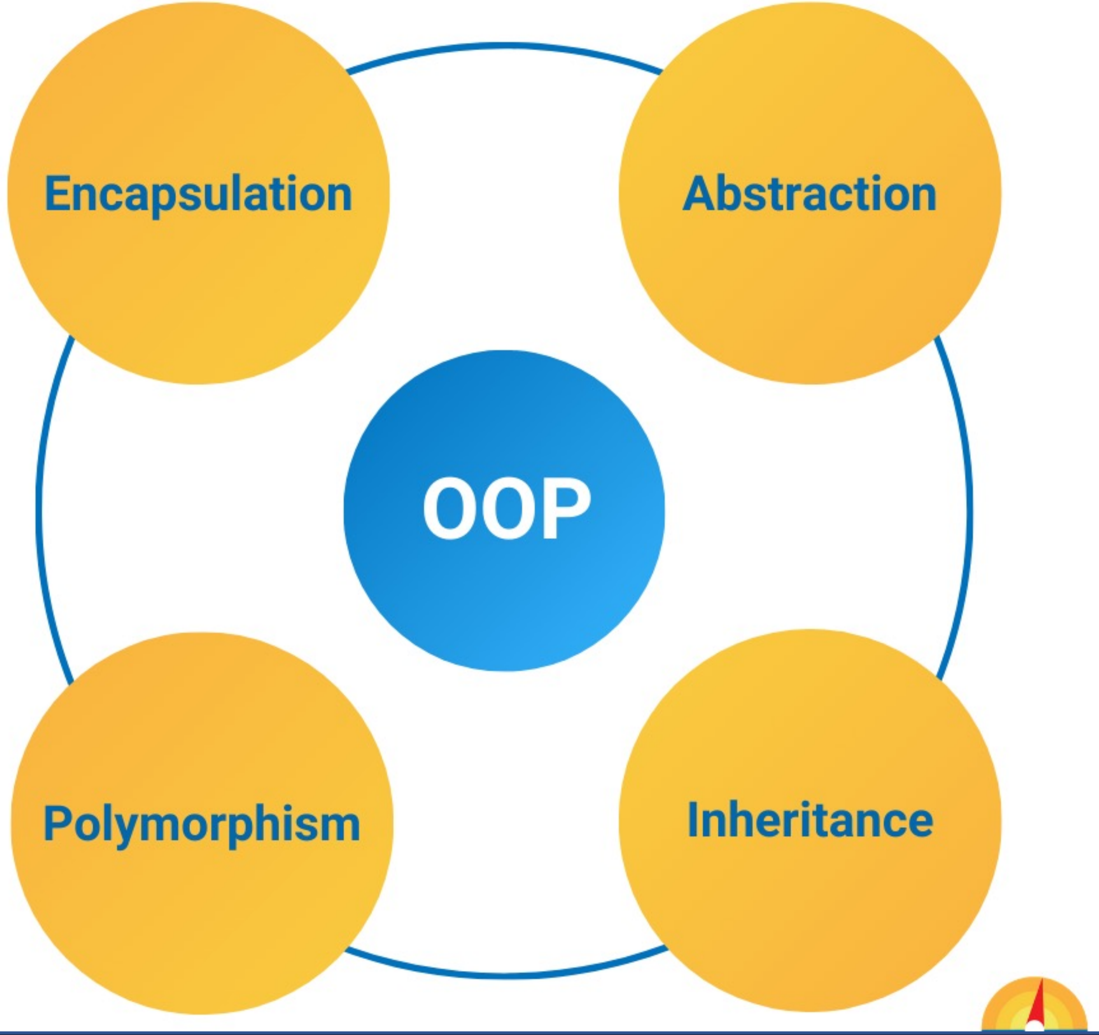
- Think before coding: broadly consider quality attributes
  - – Maintainability, extensibility, performance, …
- Propose, consider design alternatives
  - – Make explicit design decision

Using a Design Process

- A design process organizes your work
- A design process structures your understanding
- A design process facilitates communication

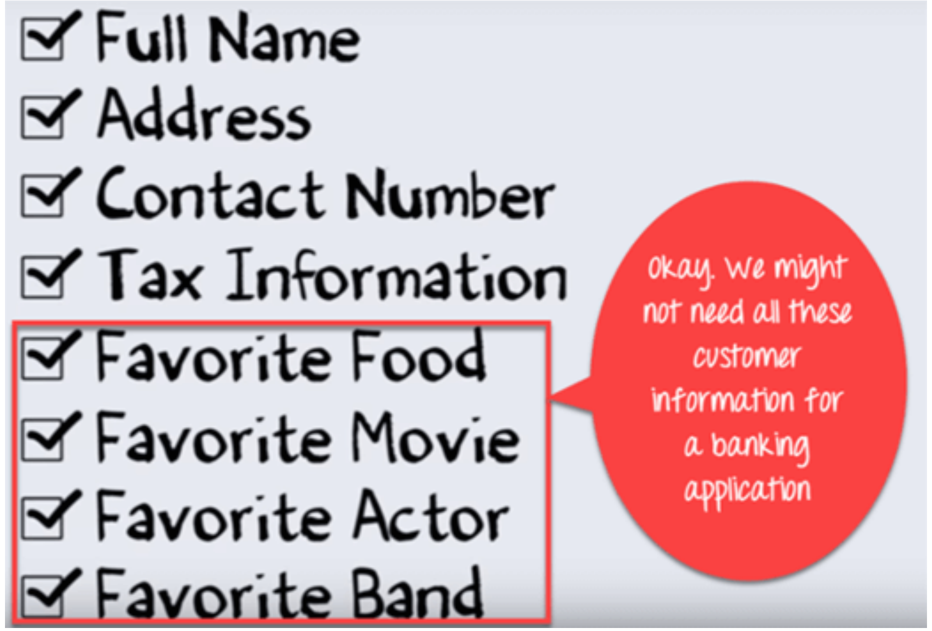

O Object
O Oriented
P Programming

Fundamental Object-Oriented Design Principle



OOP - Abstraction

- "shows" only essential attributes and "hides" unnecessary information.
- Think about a banking application, you are asked to collect all the information about your customer.



☑ Full Name

☑ Address

☑ Contact Number

☑ Tax Information

☑ Favorite Food

☑ Favorite Movie

☑ Favorite Actor

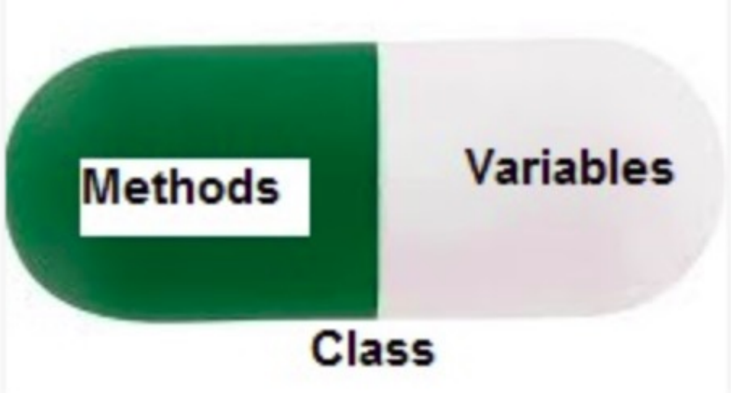
☑ Favorite Band

Okay. We might not need all these customer information for a banking application

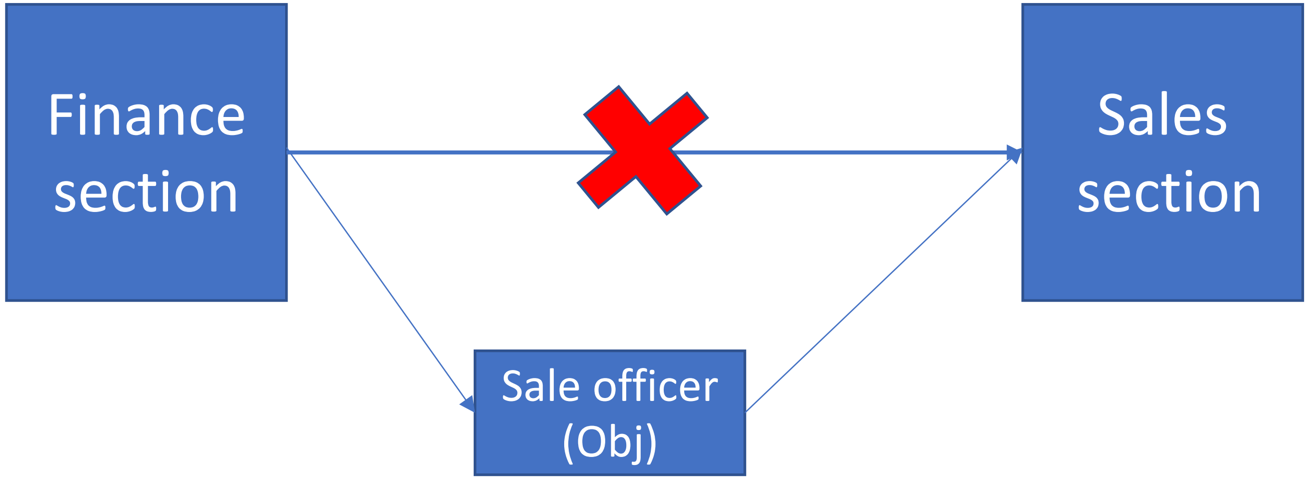
Fundamental Object-Oriented Design Principle

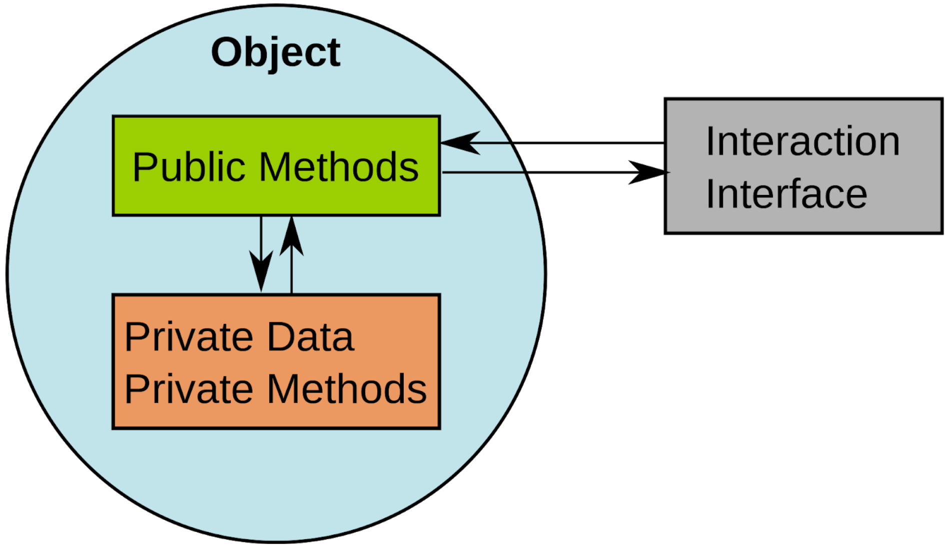
- **Abstraction**
- **Encapsulation** bundling data and methods that work on that data within one unit, e.g., a class in Java.
- Modularity
- Hierarchy

OOP - Encapsulation

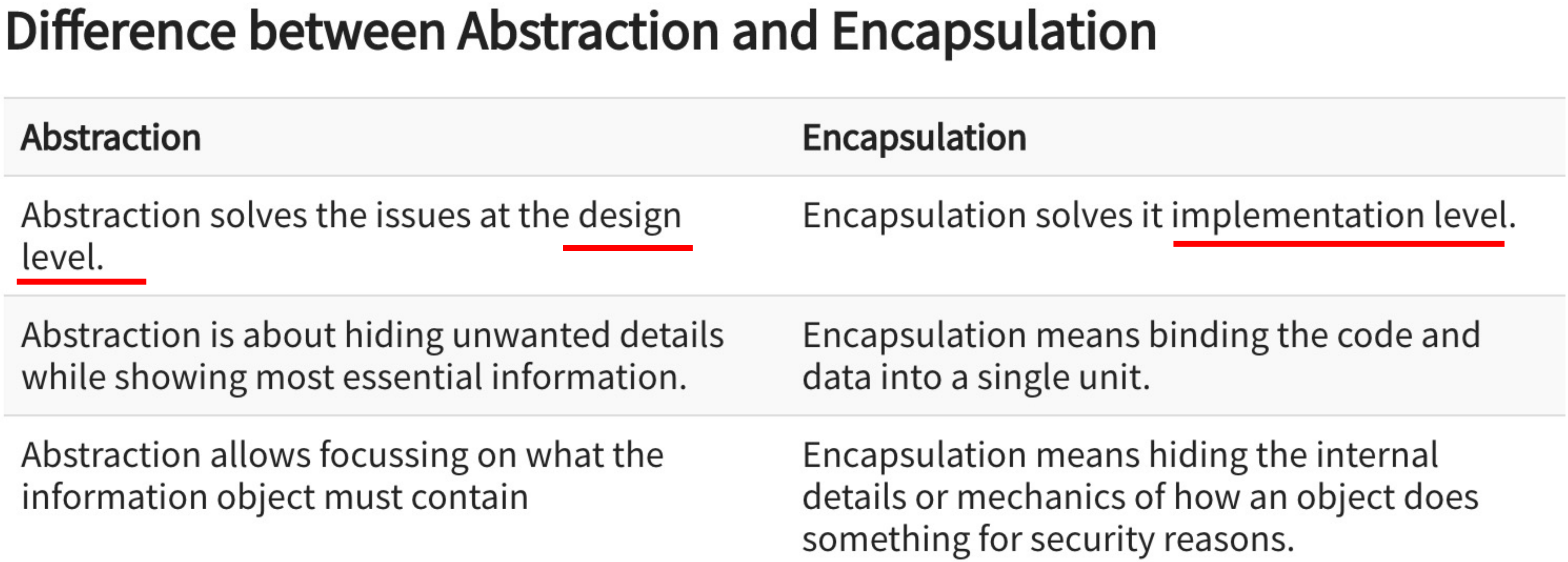


- A class is an example of encapsulation as it encapsulates all the data that is member functions, variables, etc.
- Consider a real-life example, in a company:





Fundamental Object-Oriented Design Principle



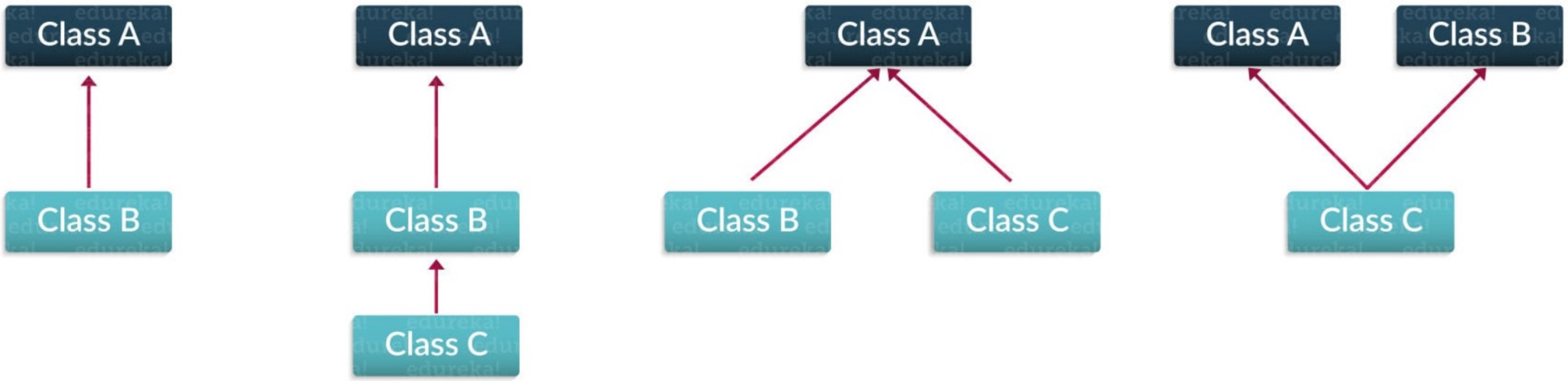
Difference between Abstraction and Encapsulation

Abstraction	Encapsulation
Abstraction solves the issues at the design level.	Encapsulation solves it implementation level.
Abstraction is about hiding unwanted details while showing most essential information.	Encapsulation means binding the code and data into a single unit.
Abstraction allows focussing on what the information object must contain	Encapsulation means hiding the internal details or mechanics of how an object does something for security reasons.

Fundamental Object-Oriented Design Principle

- **Abstraction** "shows" only essential attributes and "hides" unnecessary information.
- **Encapsulation** bundling data and methods that work on that data within one unit, e.g., a class in Java.
- **Inheritance** inheriting or transfer of characteristics from parent to child class without any modification”
- Polymorphism

Types Of Inheritance



Single Inheritance

Multilevel Inheritance

Hierarchical Inheritance

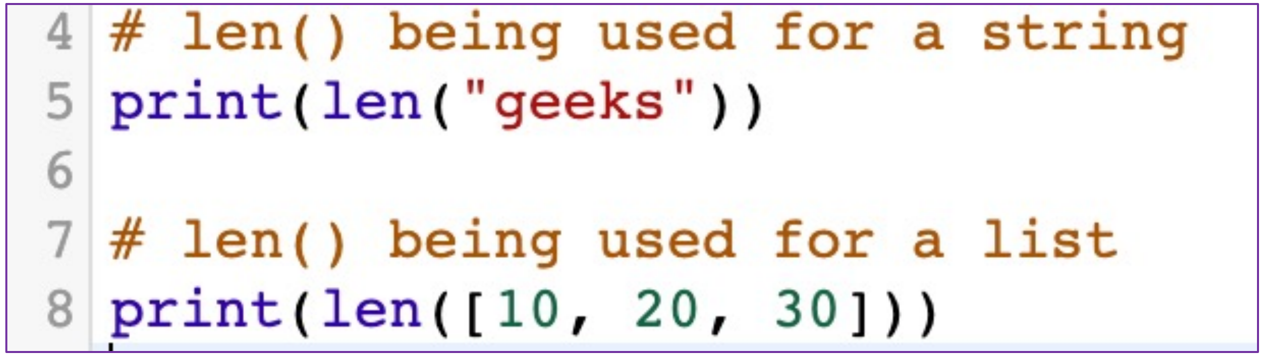
Multiple Inheritance

Fundamental Object-Oriented Design Principle

- **Abstraction** "shows" only essential attributes and "hides" unnecessary information.
- **Encapsulation** bundling data and methods that work on that data within one unit, e.g., a class in Java.
- **Inheritance** inheriting or transfer of characteristics from parent to child class without any modification"
- **Polymorphism** a property of an object which allows it to take multiple forms.

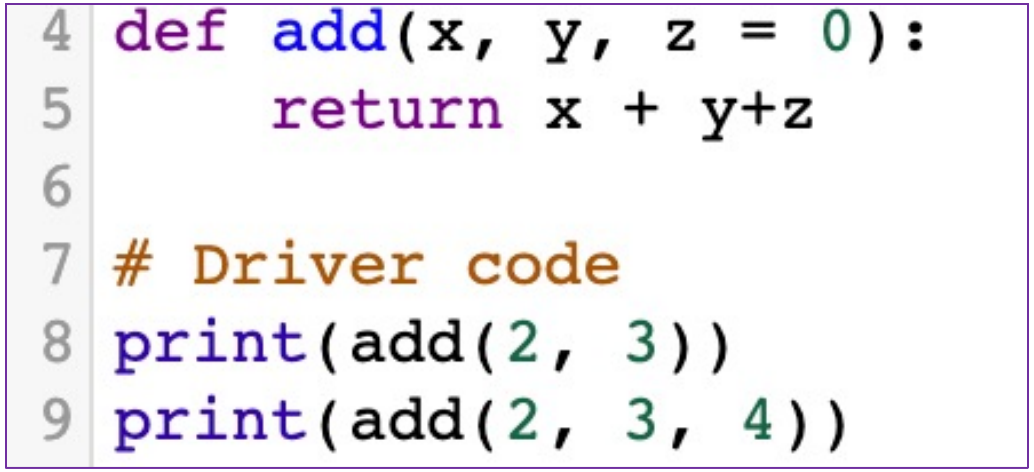
OOP - Polymorphism

- a property of an object which allows it to take multiple forms.



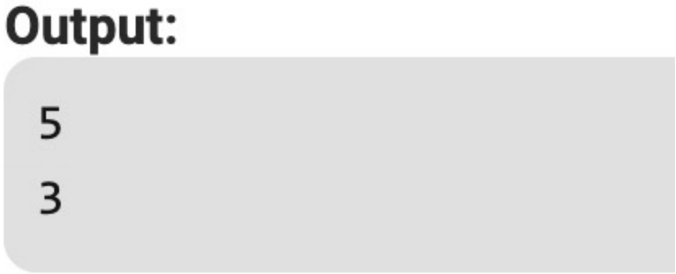
```python
# len() being used for a string
print(len("geeks"))

# len() being used for a list
print(len([10, 20, 30]))
```



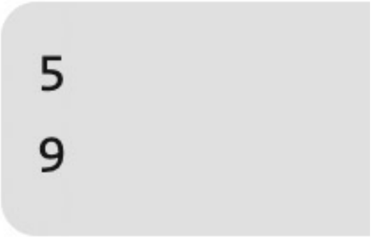
```python
def add(x, y, z = 0):
    return x + y+z

# Driver code
print(add(2, 3))
print(add(2, 3, 4))
```



Output:

```
5
3
```

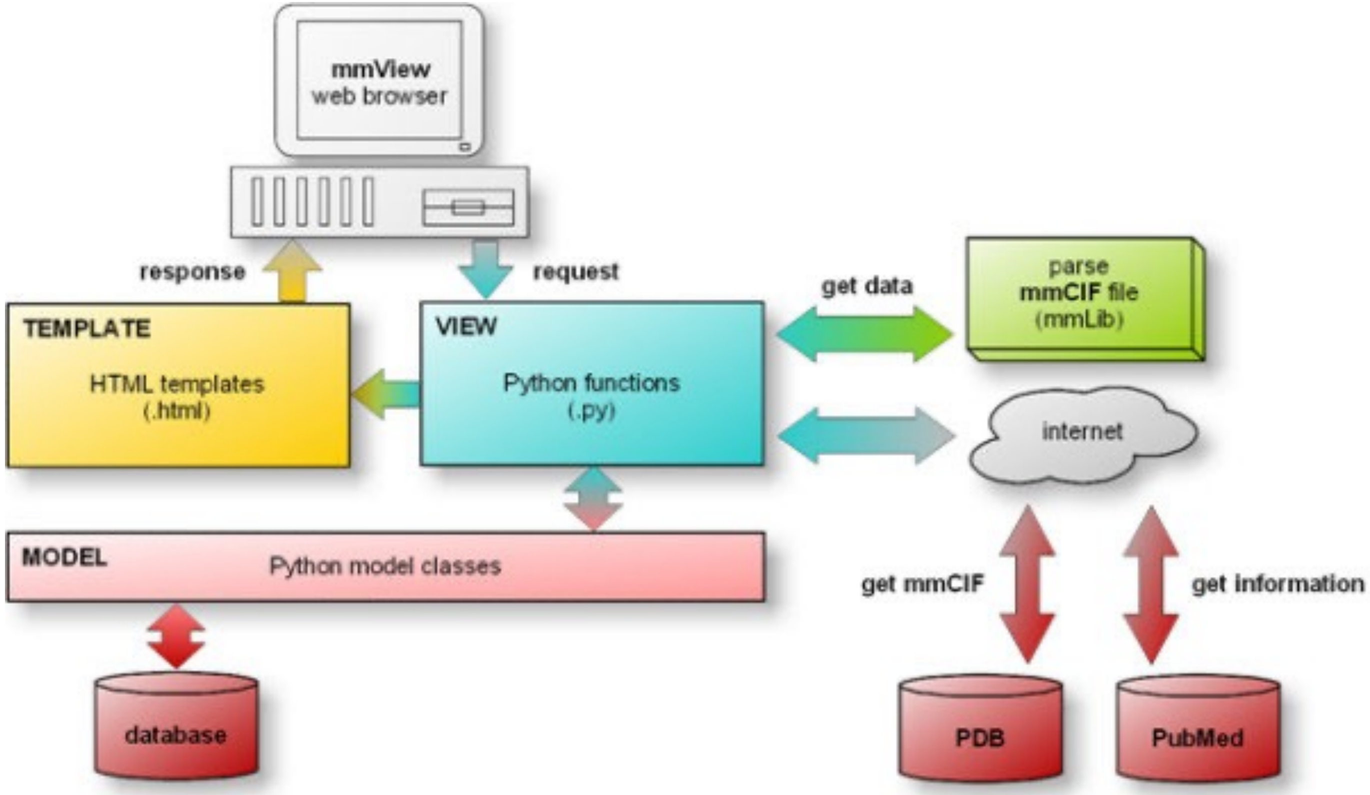


```
5
9
```

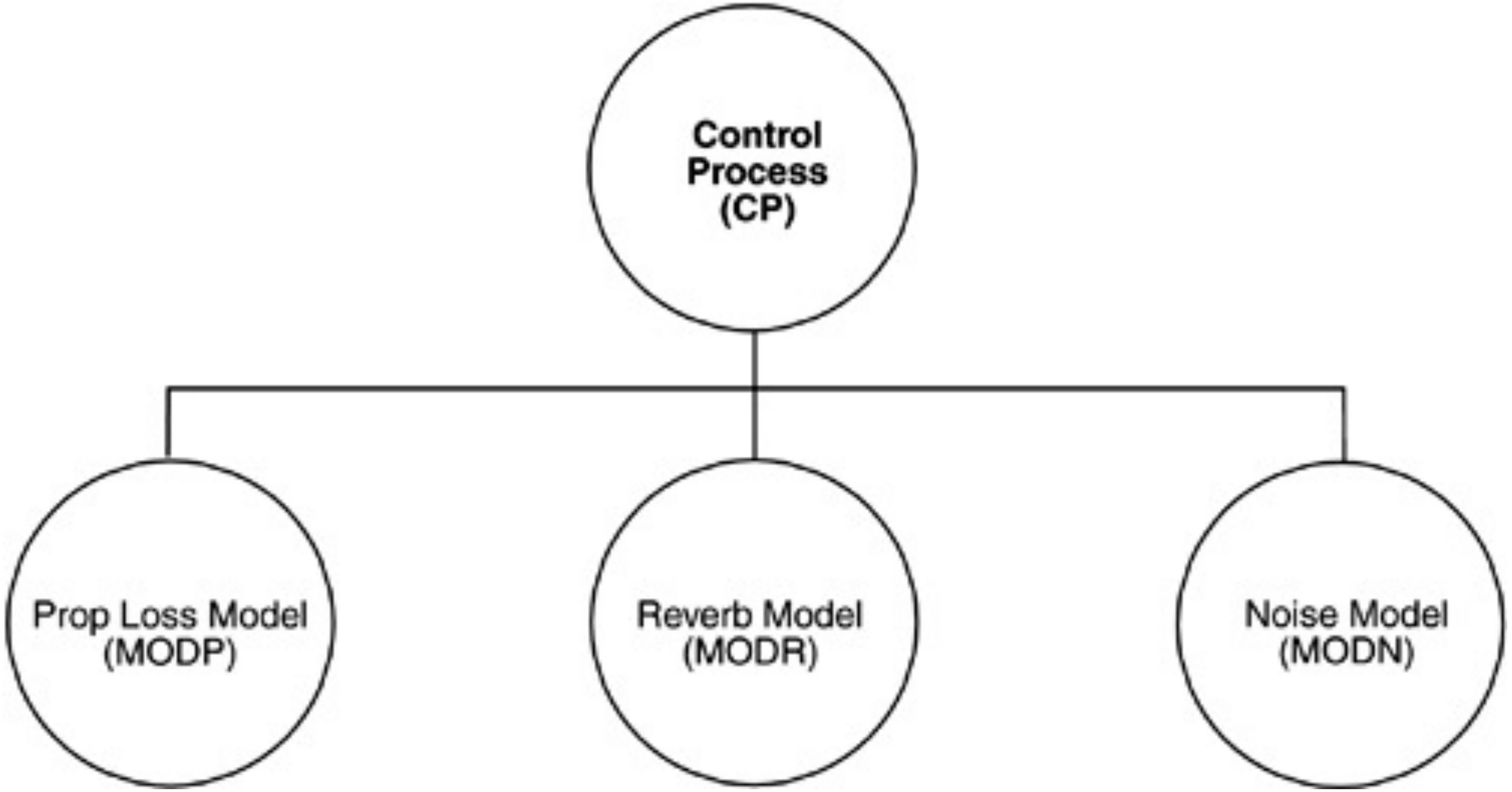
OOP

- **Abstraction** "shows" only essential attributes and "hides" unnecessary information.
- **Encapsulation** bundling data and methods that work on that data within one unit, e.g., a class in Java.
- **Inheritance** inheriting or transfer of characteristics from parent to child class without any modification”
- **Polymorphism** a property of an object which allows it to take multiple forms.

Software Architecture



Typical, but uninformative, presentation of a software architecture



From Bass et al. Software Architecture in Practice, 2nd ed.

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]

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]

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]

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]

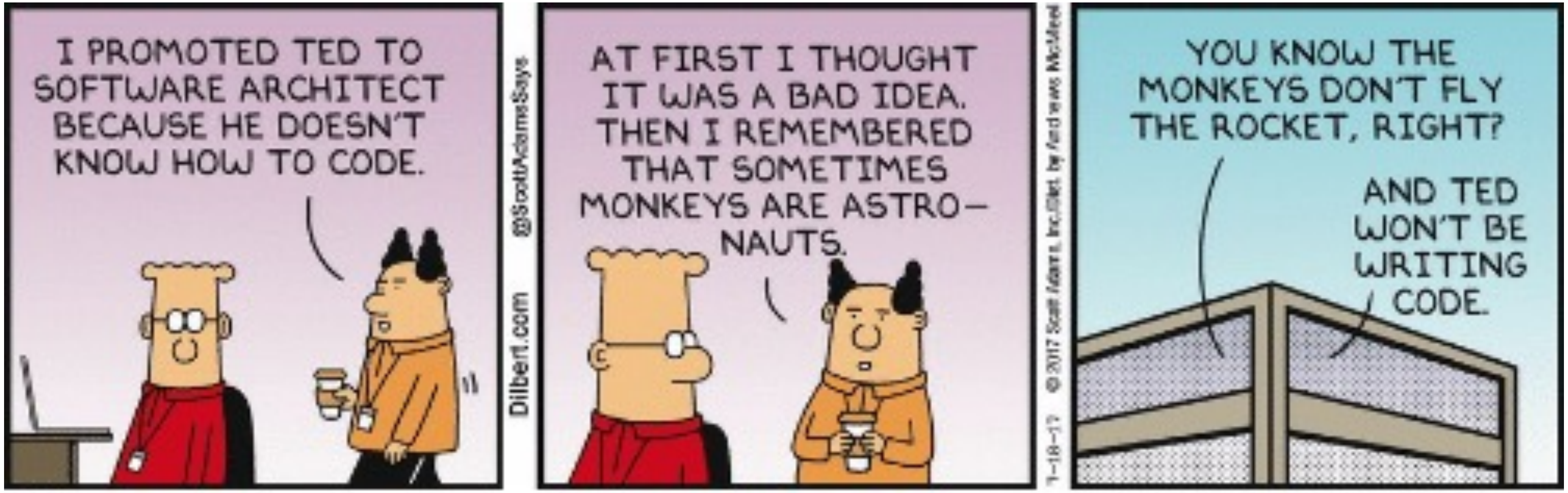
Software Architecture

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!

Why is software architecture important?


I PROMOTED TED TO SOFTWARE ARCHITECT BECAUSE HE DOESN'T KNOW HOW TO CODE.
@ScottAdamsSays
Dilbert.com
AT FIRST I THOUGHT IT WAS A BAD IDEA. THEN I REMEMBERED THAT SOMETIMES MONKEYS ARE ASTRONAUTS.
7-18-17 © 2017 Scott Adams, Inc./Dist. by Andrews McMeel
YOU KNOW THE MONKEYS DON'T FLY THE ROCKET, RIGHT?
AND TED WON'T BE WRITING CODE.

Alright, fair enough

Why is software architecture important?

1. inhibit or enable a system’s driving quality attributes.

2. to reason about and manage change as the system evolves.

3. enables early prediction of a system’s qualities.

4. enhances communication among stakeholders.

5. a carrier of the earliest and hence most fundamental, hardest-to-change design decisions.

6. defines a set of constraints on subsequent implementation.

7. Influencing the organizational structure

8. provide the basis for evolutionary prototyping.

9. the key artifact that allows the architect and project manager to reason about cost and schedule.

10. can be created as a transferable, reusable model that forms the heart of a product line.

11. Architecture-based development focuses attention on the assembly of components.

12. architecture channels the creativity of developers, reducing design and system complexity.

13. can be foundation for training a new team member.

[Bass et al. 2013]

Beyond functional correctness

- Quality matters, eg.,
  - Performance
  - Availability
  - Modifiability, portability
  - Scalability
  - Security
  - Testability
  - Usability
  - Cost to build, cost to operate

Why is software architecture important?

1. inhibit or enable a system’s driving quality attributes.
2. to reason about and manage change as the system evolves.
3. enables early prediction of a system’s qualities.
4. enhances communication among stakeholders.
5. a carrier of the earliest and hence most fundamental, hardest-to-change design decisions.
6. defines a set of constraints on subsequent implementation.
7. Influencing the organizational structure
8. provide the basis for evolutionary prototyping.
9. the key artifact that allows the architect and project manager to reason about cost and schedule.
10. can be created as a transferable, reusable model that forms the heart of a product line.
11. Architecture-based development focuses attention on the assembly of components.
12. architecture channels the creativity of developers, reducing design and system complexity.
13. can be foundation for training a new team member.

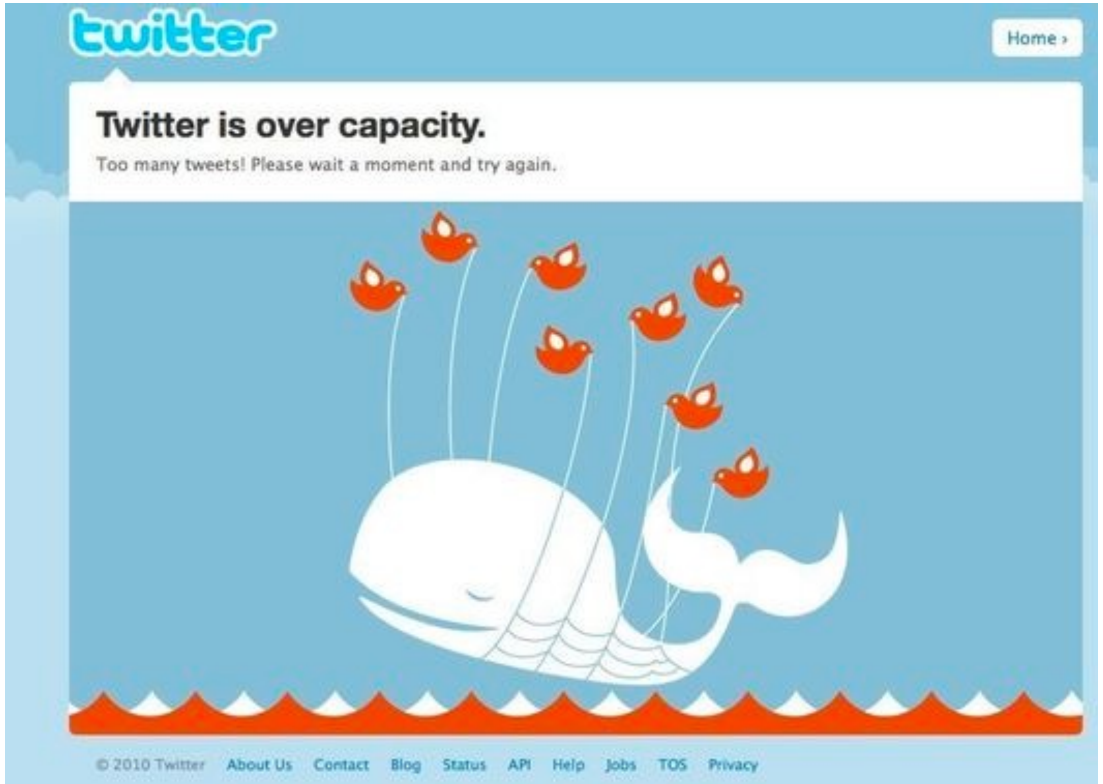
[Bass et al. 2013]

Why is software architecture important?

1. inhibit or enable a system’s driving quality attributes.
2. to reason about and manage change as the system evolves.
3. enables early prediction of a system’s qualities.
4. enhances communication among stakeholders.
5. a carrier of the earliest and hence most fundamental, hardest-to-change design decisions.
6. defines a set of constraints on subsequent implementation.
7. Influencing the organizational structure
8. provide the basis for evolutionary prototyping.
9. the key artifact that allows the architect and project manager to reason about cost and schedule.
10. can be created as a transferable, reusable model that forms the heart of a product line.
11. Architecture-based development focuses attention on the assembly of components.
12. architecture channels the creativity of developers, reducing design and system complexity.
13. can be foundation for training a new team member.

[Bass et al. 2013]

Case Study:
Architecture and Quality at Twitter



Twitter is over capacity.

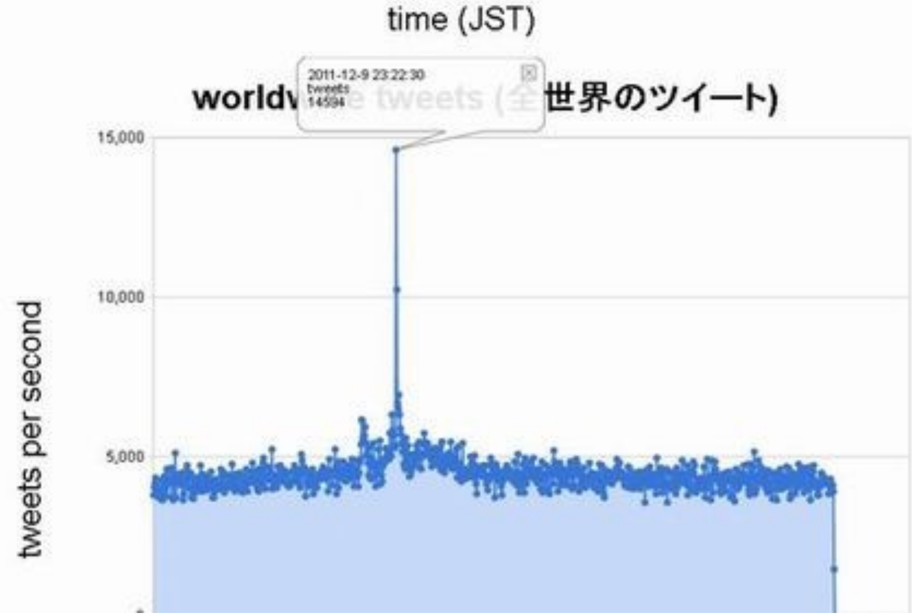
Too many tweets! Please wait a moment and try again.

"After that experience, we determined we **needed to step back**. We then determined we needed to **re-architect** the site to support the continued growth of Twitter and to keep it running smoothly."

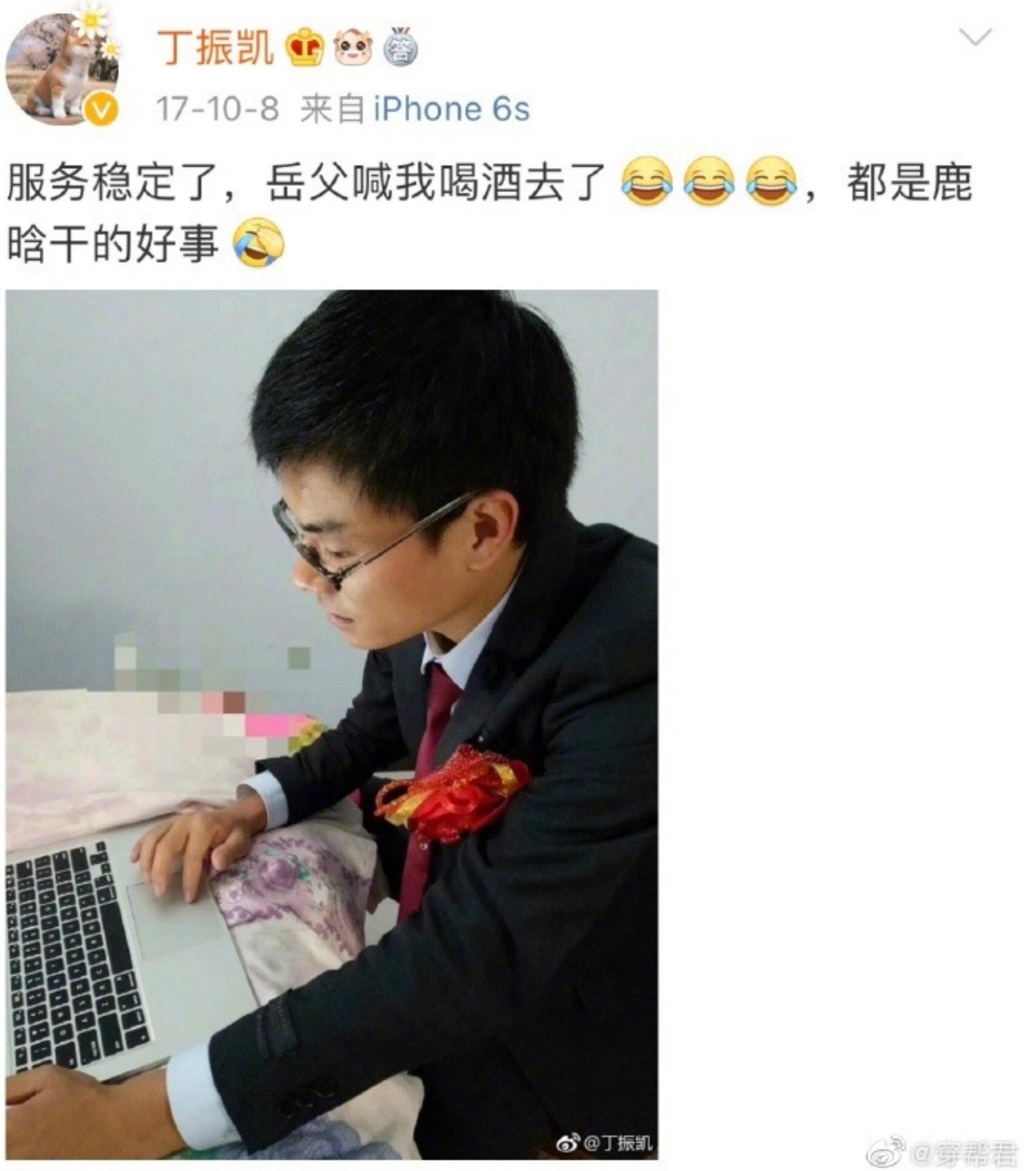
Balse!

Toei Company

A scene from *Castle in the Sky*, the classic 1986 film by Hayao Miyazaki.



Celebrities‘ personal revelation caused a sudden breakdown of Weibo’s server due to the traffic. Ding Zhenkai, a Weibo programmer at the site, said he got called to work on the breakdown during his wedding.



丁振凯

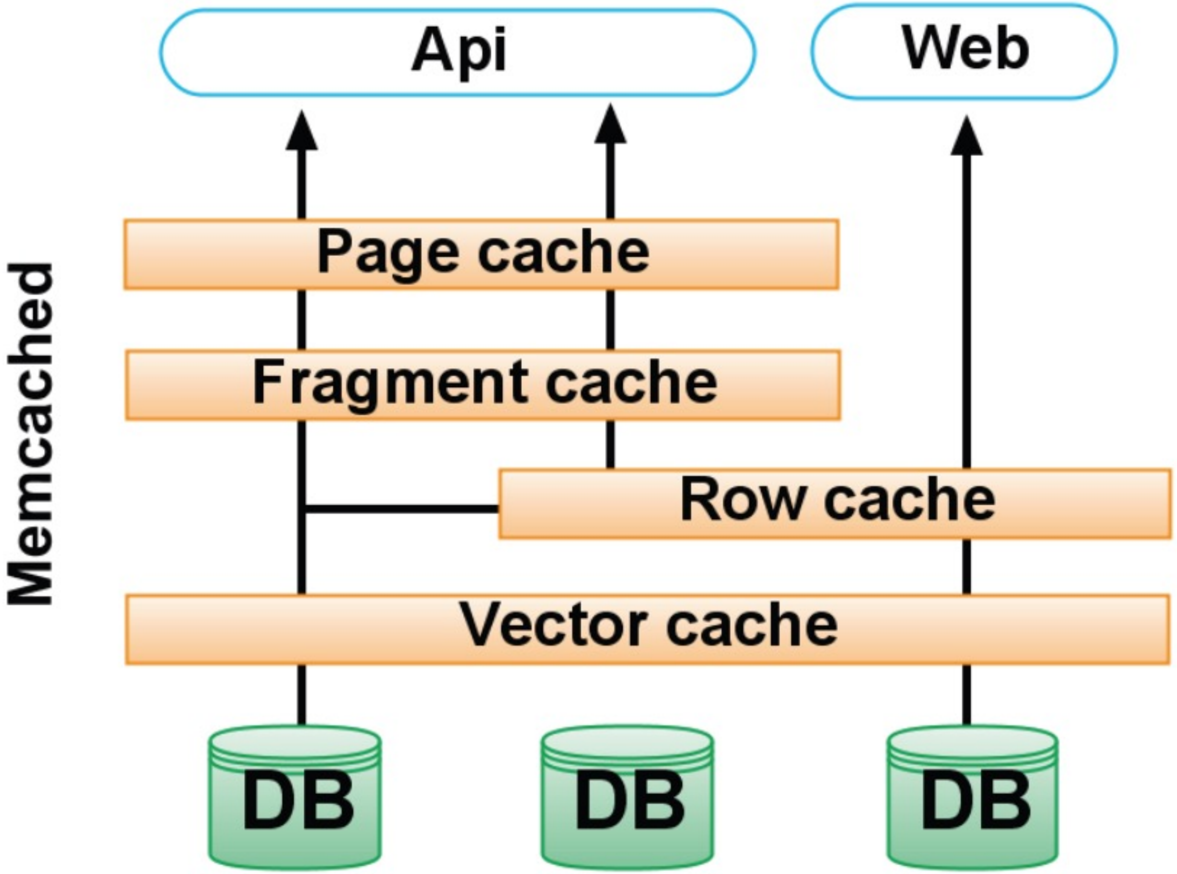
17-10-8 来自iPhone 6s

服务稳定了，岳父喊我喝酒去了😂😂😂，都是鹿晗干的好事😭

Inspecting the State of Engineering

- Running one of the world's largest Ruby on Rails installations
- 200 engineers
- Monolithic: managing raw database, memcache, rendering the site, and presenting the public APIs in one codebase

Caching



What is Memcached?

Free & open source, high-performance, distributed memory object caching system, generic in nature, but intended for use in speeding up dynamic web applications by alleviating database load.

Memcached is an in-memory key-value store for small chunks of arbitrary data (strings, objects) from results of database calls, API calls, or page rendering.

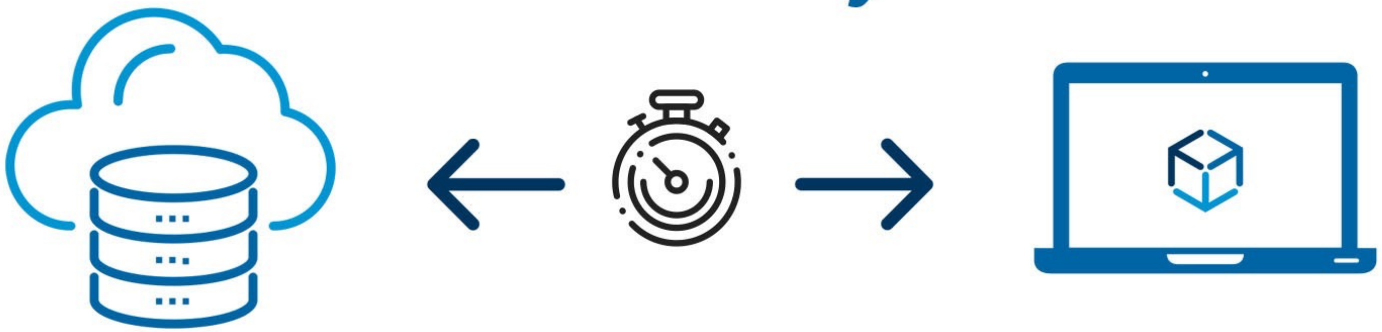
Inspecting the State of Engineering (Cont.)

- Increasingly difficult to understand system; organizationally challenging to manage and parallelize engineering teams
- Reached the limit of throughput on our storage systems (MySQL); read and write hot spots throughout our databases
- Throwing machines at the problem; low throughput per machine (CPU + RAM limit, network not saturated)
- Optimization corner: trading off code readability vs performance

Twitter's Quality Requirements/Redesign goals

- Improve median latency; lower outliers

Latency



Time it takes for a request to go from the client to the server and back to the client

Twitter's Quality Requirements/Redesign goals

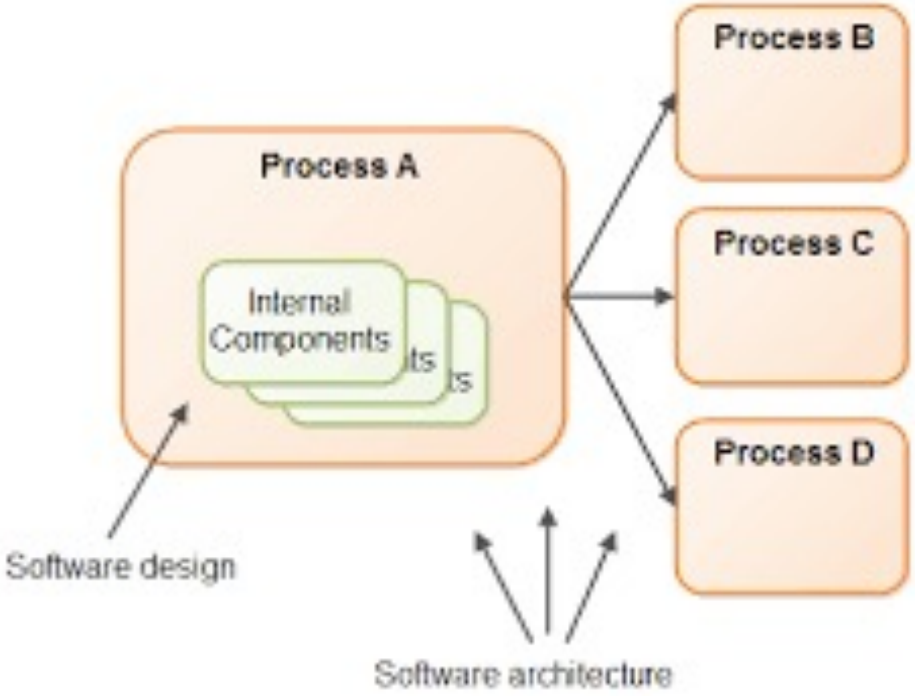
- Reduce number of machines 10x

Twitter's Quality Requirements/Redesign goals

- **Isolate failures** -- the failure does not propagate or cause a deterioration of other services within the platform. The “blast radius” of failure is contained.

Twitter's Quality Requirements/Redesign goals

- "We wanted **cleaner boundaries** with “related” logic being in one place"
  - encapsulation and modularity at the systems level (rather than at the class, module, or package level)
  - best practices of encapsulation and modularity



Twitter's Quality Requirements/Redesign goals

- Quicker release of new features
  - "run small and empowered engineering teams that could make local decisions and ship user-facing changes, independent of other teams"

Twitter's Quality Requirements/Redesign goals

- Improve median latency; lower outliers

performance

- Reduce number of machines 10x
- Isolate failures

reliability

- "We wanted cleaner boundaries with “related” logic being in one place"
  - encapsulation and modularity at the systems level (rather than at the class, module, or package level)

maintainability

- Quicker release of new features
  - "run small and empowered engineering teams that could make local decisions and ship user-facing changes, independent of other teams"

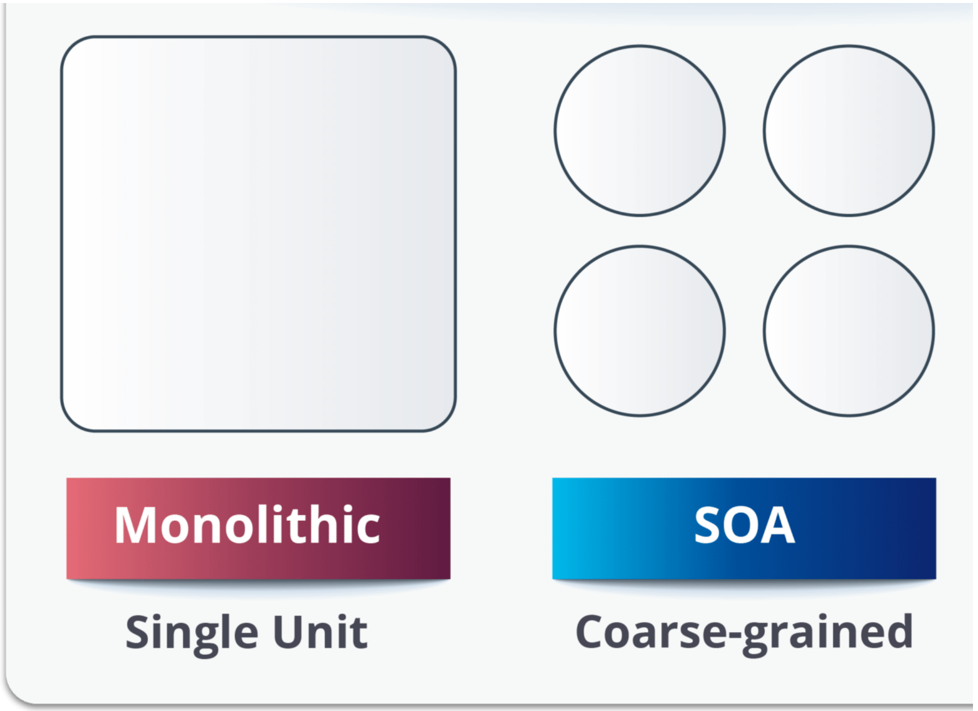
modifiability

JVM vs Ruby VM

- Rails servers capable of 200-300 requests / sec / host
- Experience with Scala on the JVM; level of trust
- Rewrite for JVM allowed 10-20k requests / sec / host

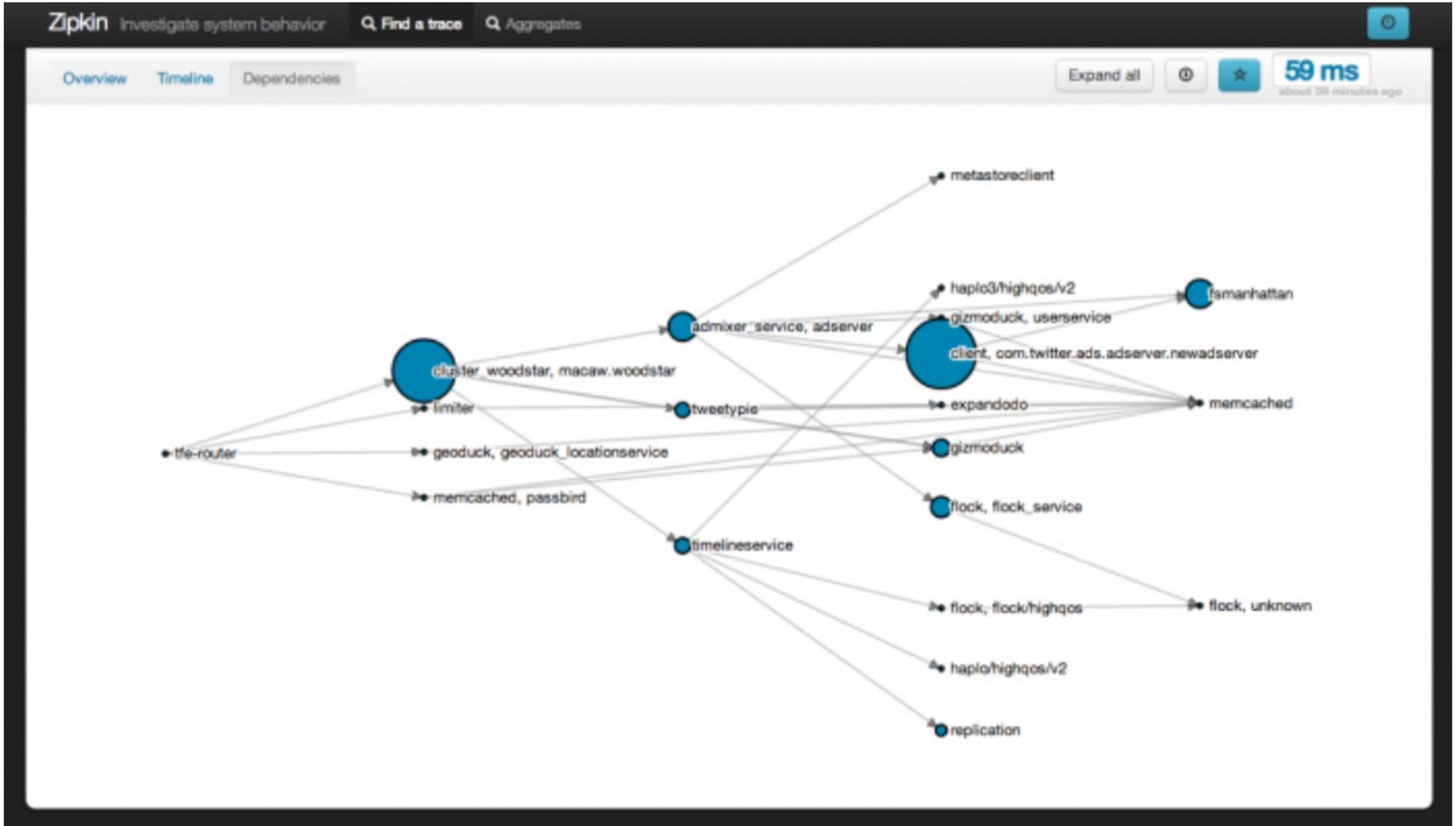
move from monolithic Ruby application to one that is more services oriented.

Either needed experts who understood the entire codebase or clear owners at the module or class level.



- develop the system in parallel
- logic for each system was self-contained within itself.
- need coordination

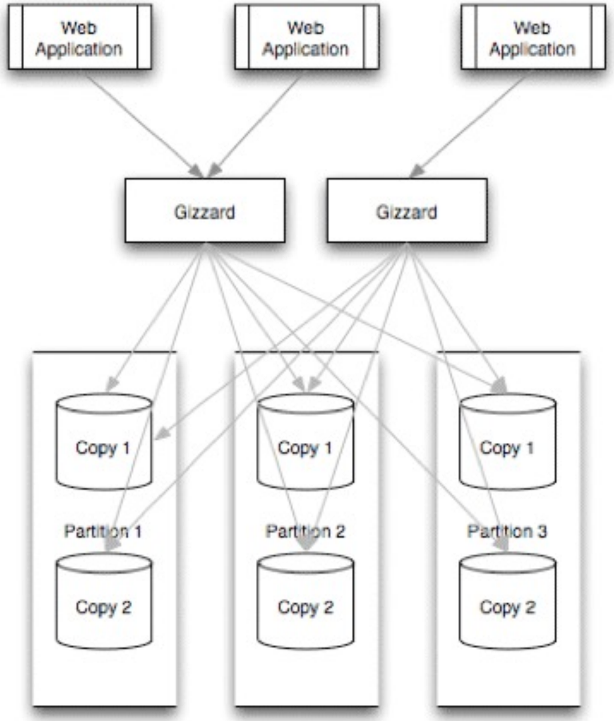
Software architecture Influencing the organizational structure



“. .. we’d organize “whale hunting expeditions” to try to understand large scale failures that occurred. At the end of the day, we’d spend more time on this than on shipping features, which we weren’t happy with.”

Storage

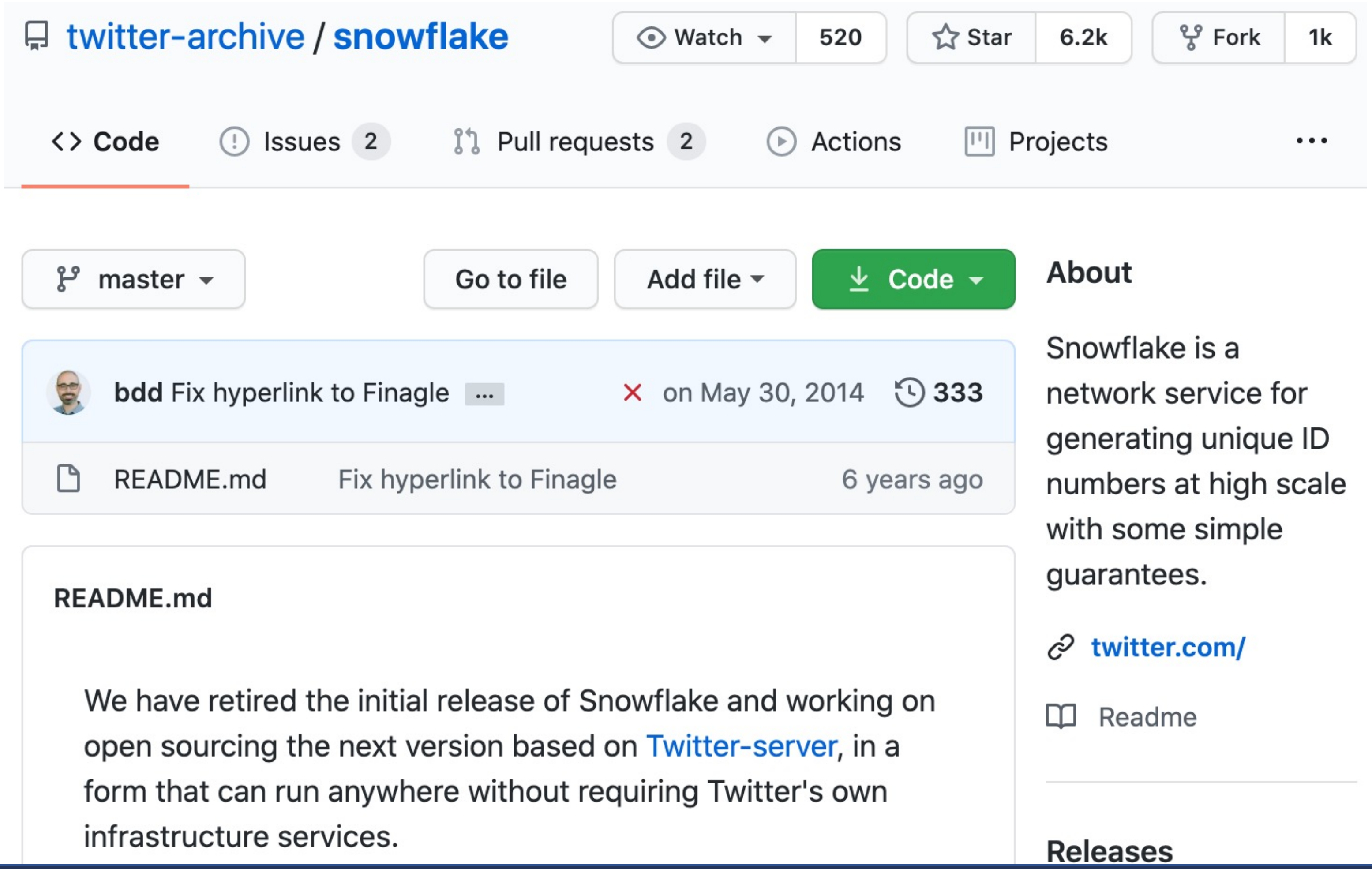
- "We stored the tweets in order in the database, and when the database filled up we spun up another one "
- Single-master MySQL → Distributed DB
  - every time a tweet comes into the system, we hashes it, and then chooses an appropriate database.



Q: potential problem ?

A: lose the ability to rely on MySQL for unique ID generation.

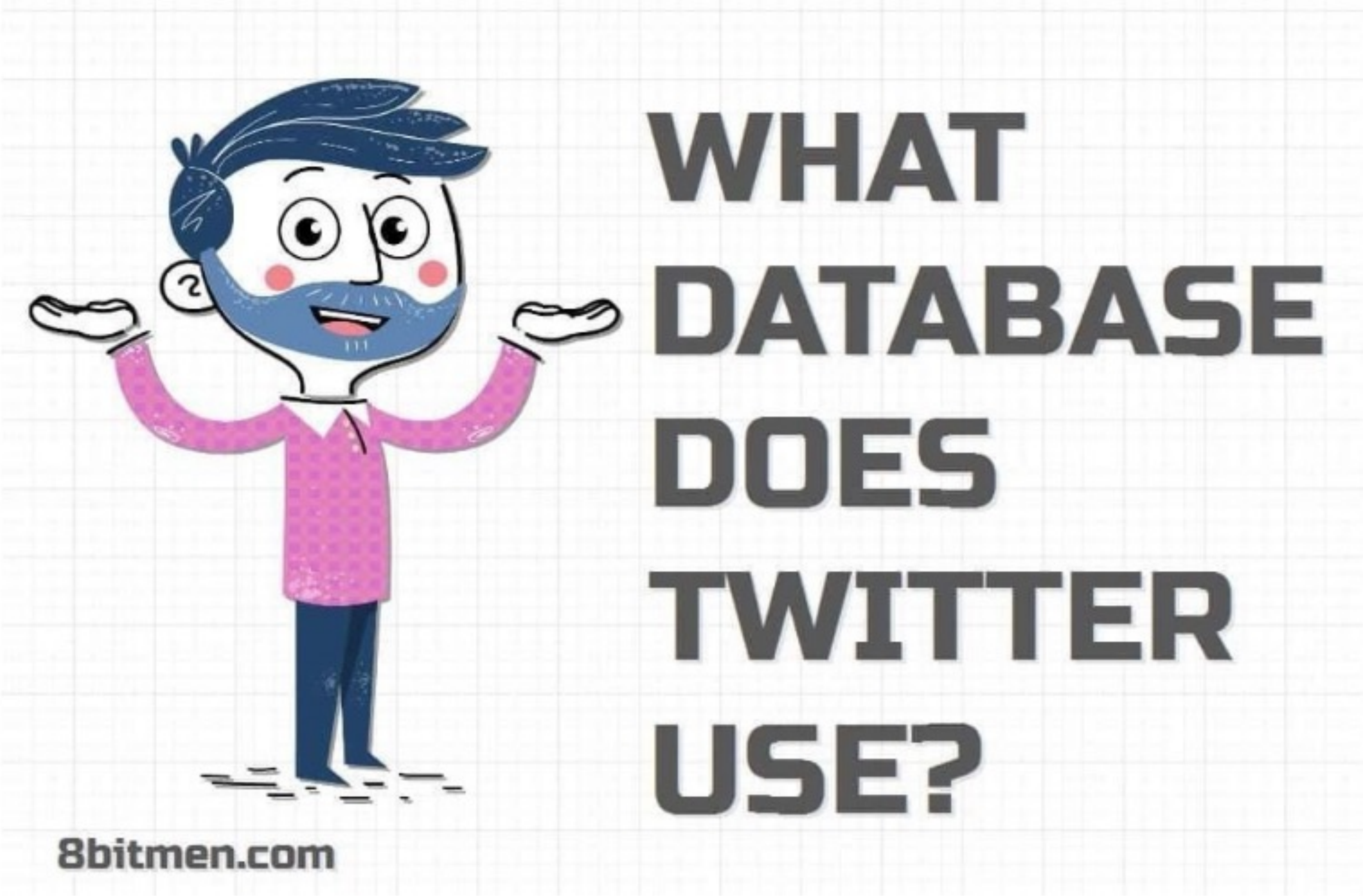
Snowflake: Create an almost-guaranteed globally unique identifier



About

Snowflake is a network service for generating unique ID numbers at high scale with some simple guarantees.

We have retired the initial release of Snowflake and working on open sourcing the next version based on Twitter-server, in a form that can run anywhere without requiring Twitter's own infrastructure services.

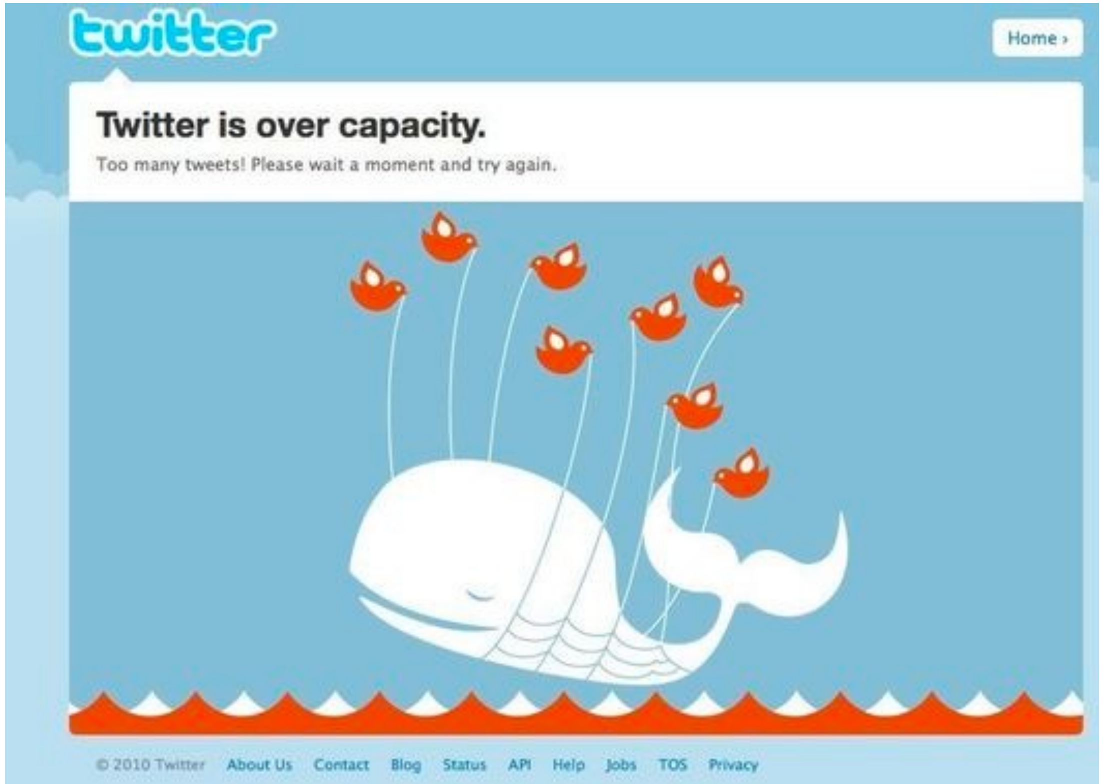


https://www.scaleyourapp.com/what-database-does-twitter-use-a-deep-dive/

Key Insights: Twitter Case Study

- Architectural decisions affect entire systems, not only individual modules
- Abstract, different abstractions for different scenarios
- Reason about quality attributes early
- Make architectural decisions explicit

Question: *Did the original architect make poor decisions?*



Twitter is over capacity.

Too many tweets! Please wait a moment and try again.

(UML) Unified Modeling Language

The Edward S. Rogers Sr. Department
of Electrical & Computer Engineering
UNIVERSITY OF TORONTO

Modeling Notations

- Used for both requirements analysis and for specification and design
  - Useful for technical people
  - Provide a high-level view
  - Descendent of Entity-Relationship Diagrams
  - Describes data and operations
  - Require training
  - Many notations
    - each good for something
    - none good for everything

Origin of UML

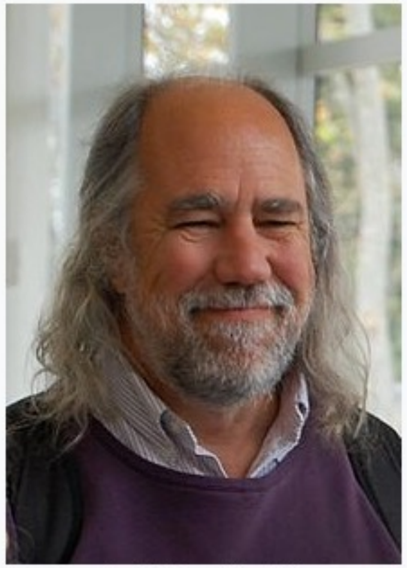
Grady Booch Diagrams +

Jim Rumbaugh (OMT) Object Diagrams +

Ivar Jacobson use case diagrams

Three Amigos

Grady Booch



Grady Booch in 2011

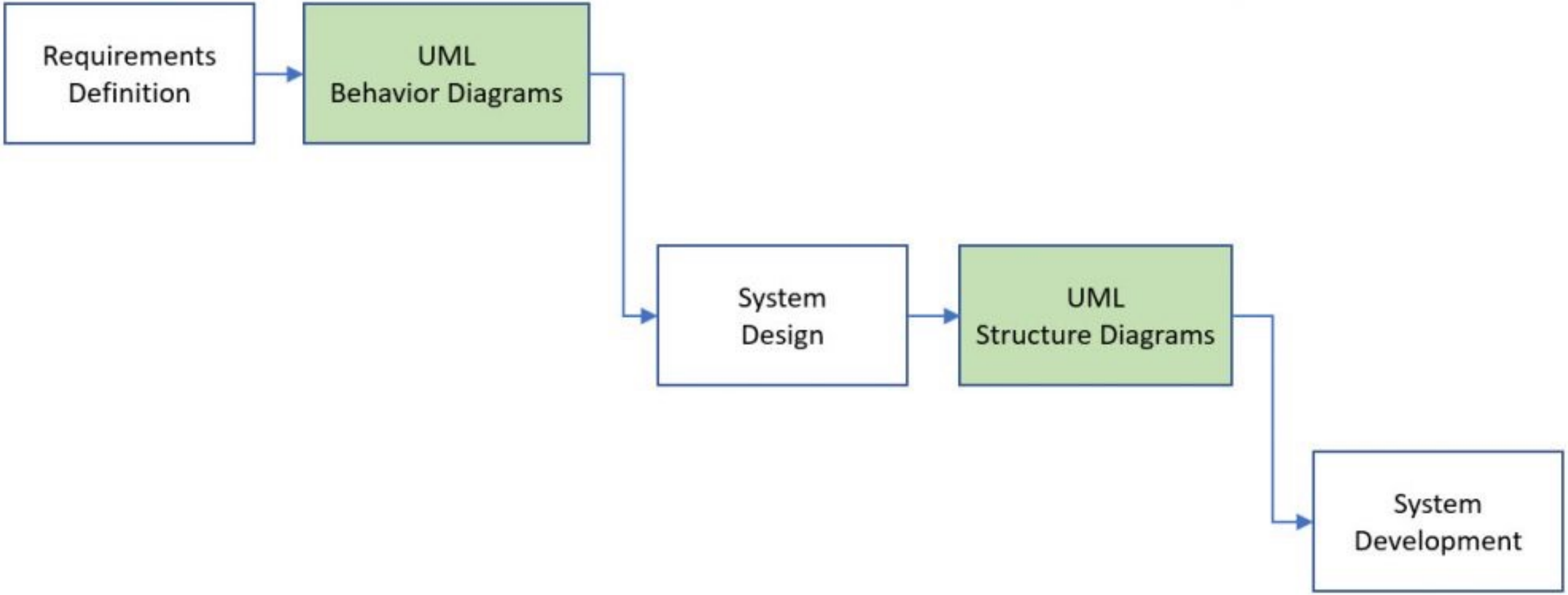
Jim Rumbaugh

Ivar Jacobson

Usage of UML

- Help developers communicate
- Provide documentation
- Help find errors (tools check for consistency)
- Generate code (with tools)
- Drawing Tools: ArgoUML, Visio (Microsoft), OmniGraffle

UML works best in a sequential Waterfall process



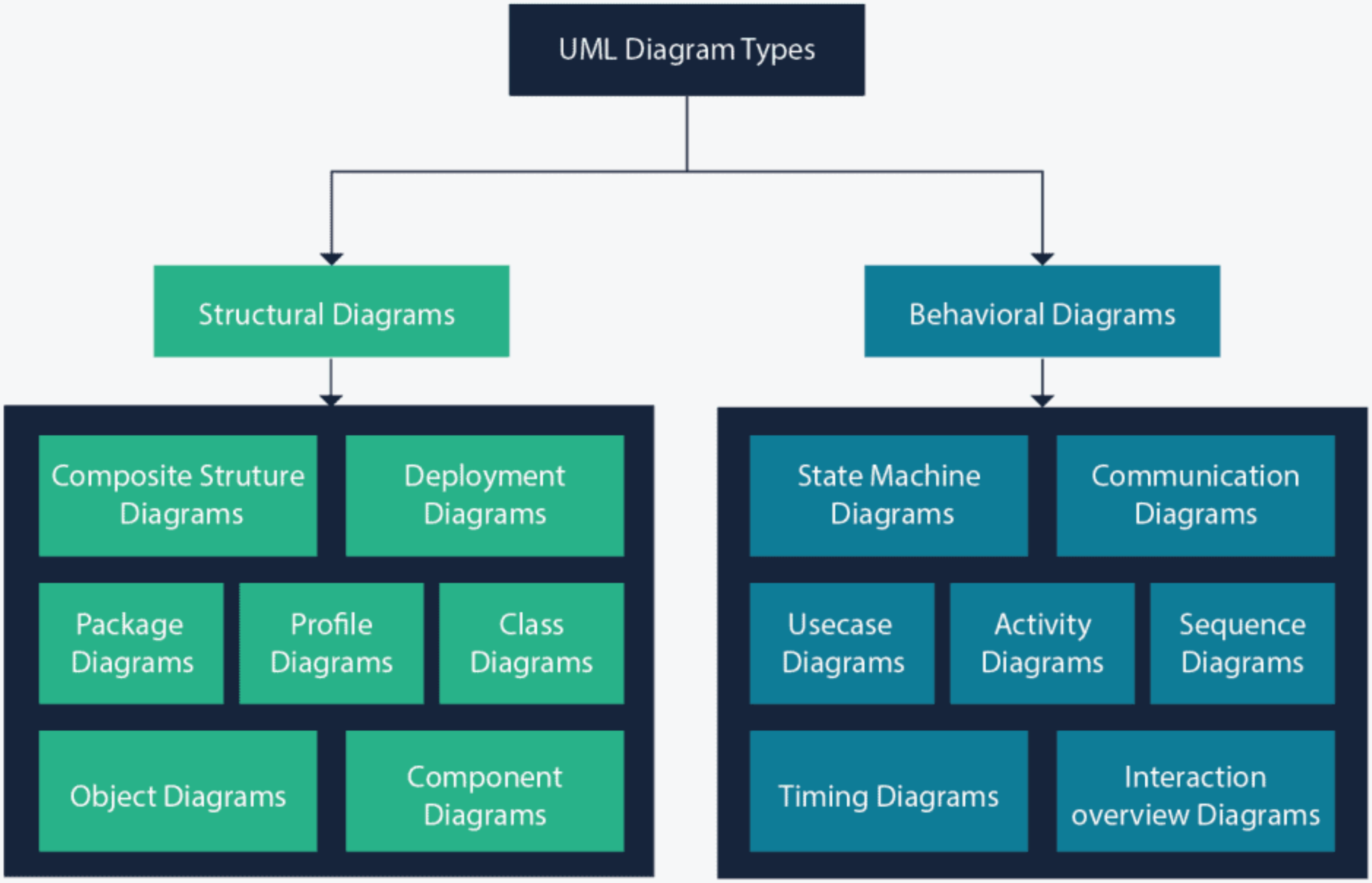
Requirements Definition

UML Behavior Diagrams

System Design

UML Structure Diagrams

System Development



UML Diagram Types

Structural Diagrams

Composite Struture Diagrams

Deployment Diagrams

Package Diagrams

Profile Diagrams

Class Diagrams

Object Diagrams

Component Diagrams

Behavioral Diagrams

State Machine Diagrams

Communication Diagrams

Usecase Diagrams

Activity Diagrams

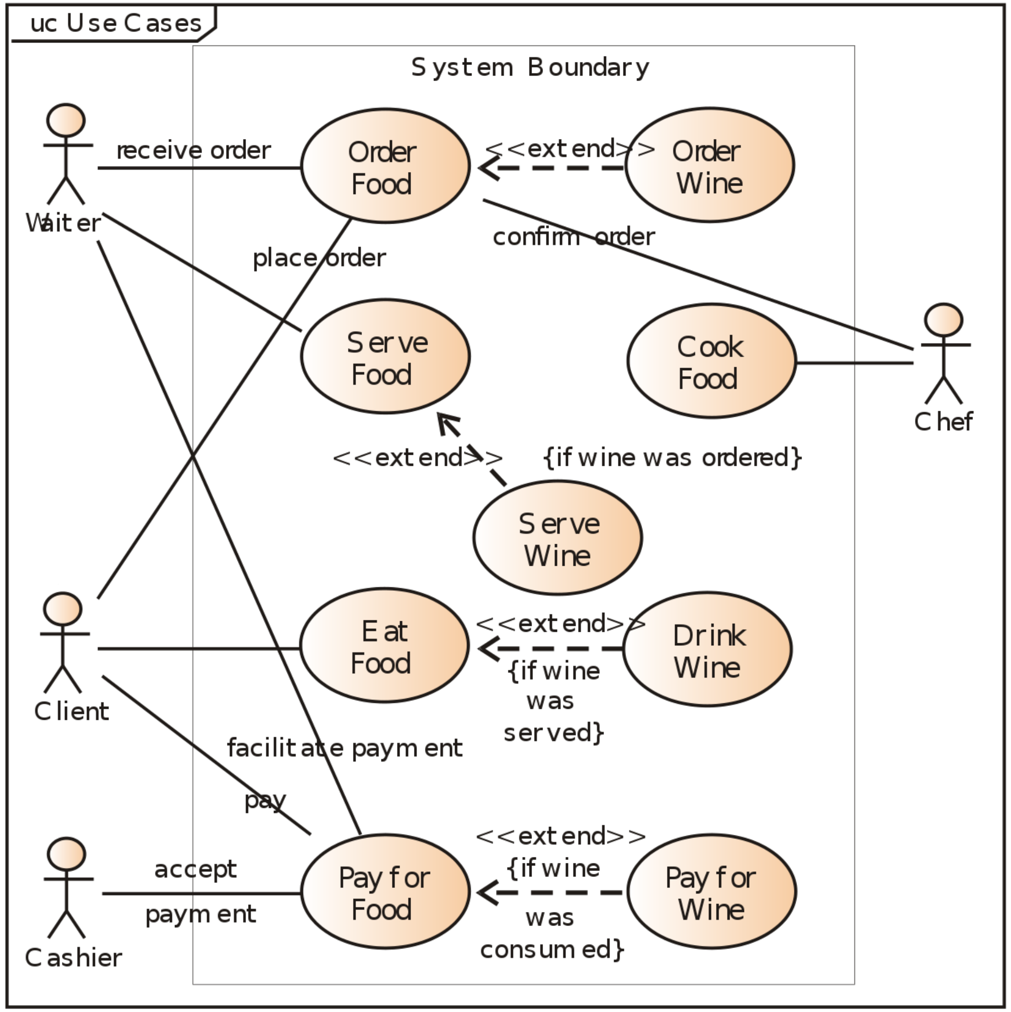
Sequence Diagrams

Timing Diagrams

Interaction overview Diagrams

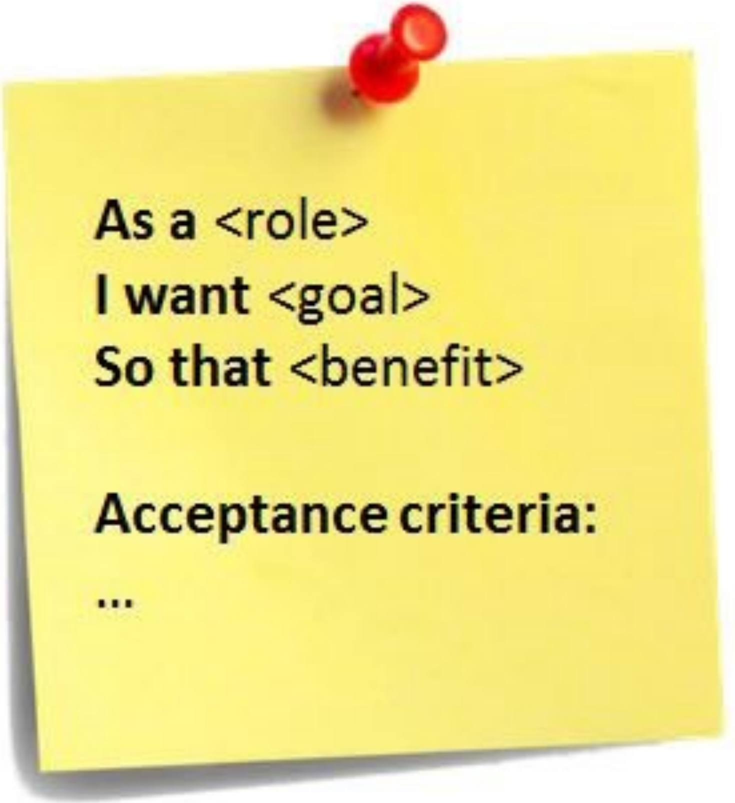
Behavioral UML Diagram – Use Case Diagram

- Actor + Action



User Stories

- Informal descriptions of user-valued features scheduled for implementation
- Details left for negotiation with customer later or pointer to real requirements
- Common agile development practice



As a <role>
I want <goal>
So that <benefit>

Acceptance criteria:
…

Use cases

Use Case Name	(Title)
Scope	System under design
Level	User level, subprocess level
Primary actor	(actors can be primary, supporting, or offstage)
Stakeholders, interests	Important! A use case should include everything necessary to satisfy the stakeholders’ interests.
Preconditions	What must always be true before a scenario begins. Not tested; assumed. Don’t fill with pointless noise.
Success guarantees.	Aka post conditions
Main success scenario	Basic flow, “happy path”, typical flow. Defer all conditions to the extensions. Records steps: interaction between actors, a validation, a state change by the system.
Extensions	Aka alternate flows. Usually the majority of the text. Sometimes branches off into another use case.
Special requirements	Where the non-functional/quality requirements live.
Technology and data variations list	Unavoidable technology constraints; try to keep to I/O technologies.
Frequency of occurrence	
Miscellaneous	

Defining actors/agents

- An actor is an entity that interacts with the system for the purpose of completing an event [Jacobson, 1992].
  - Not as broad as stakeholders.
- Actors can be a user, an organization, a device, or an external system.

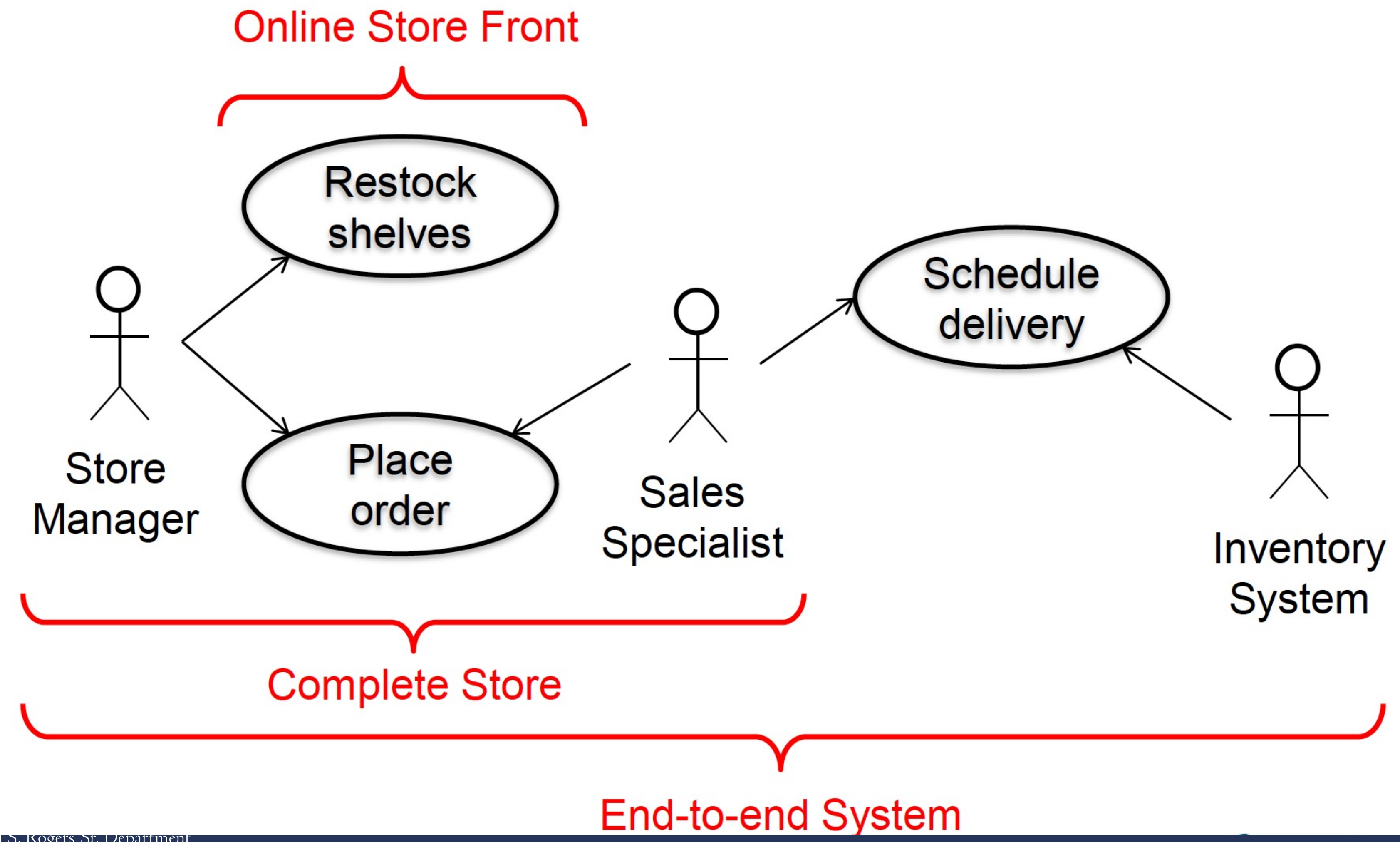
Sales Specialist

Marketing

GPS Receiver

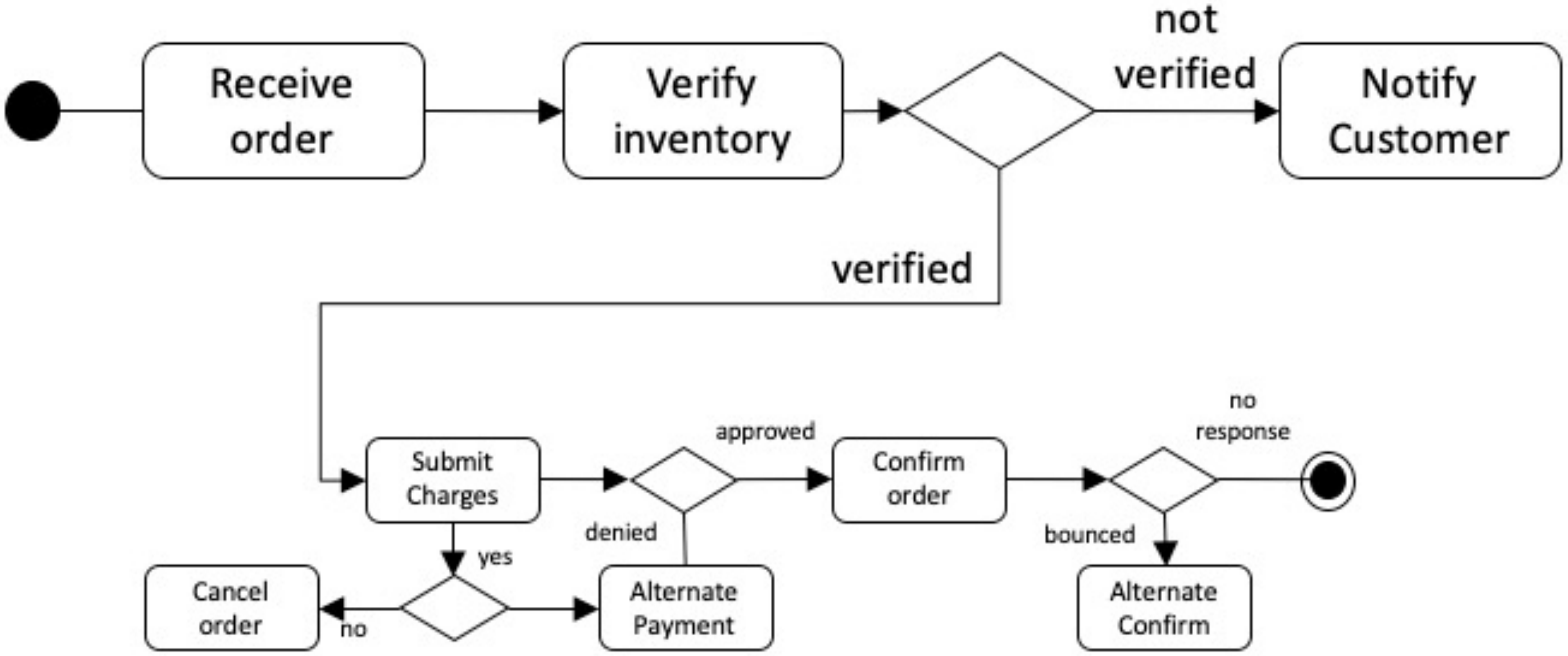
Inventory System

Defining the system boundary



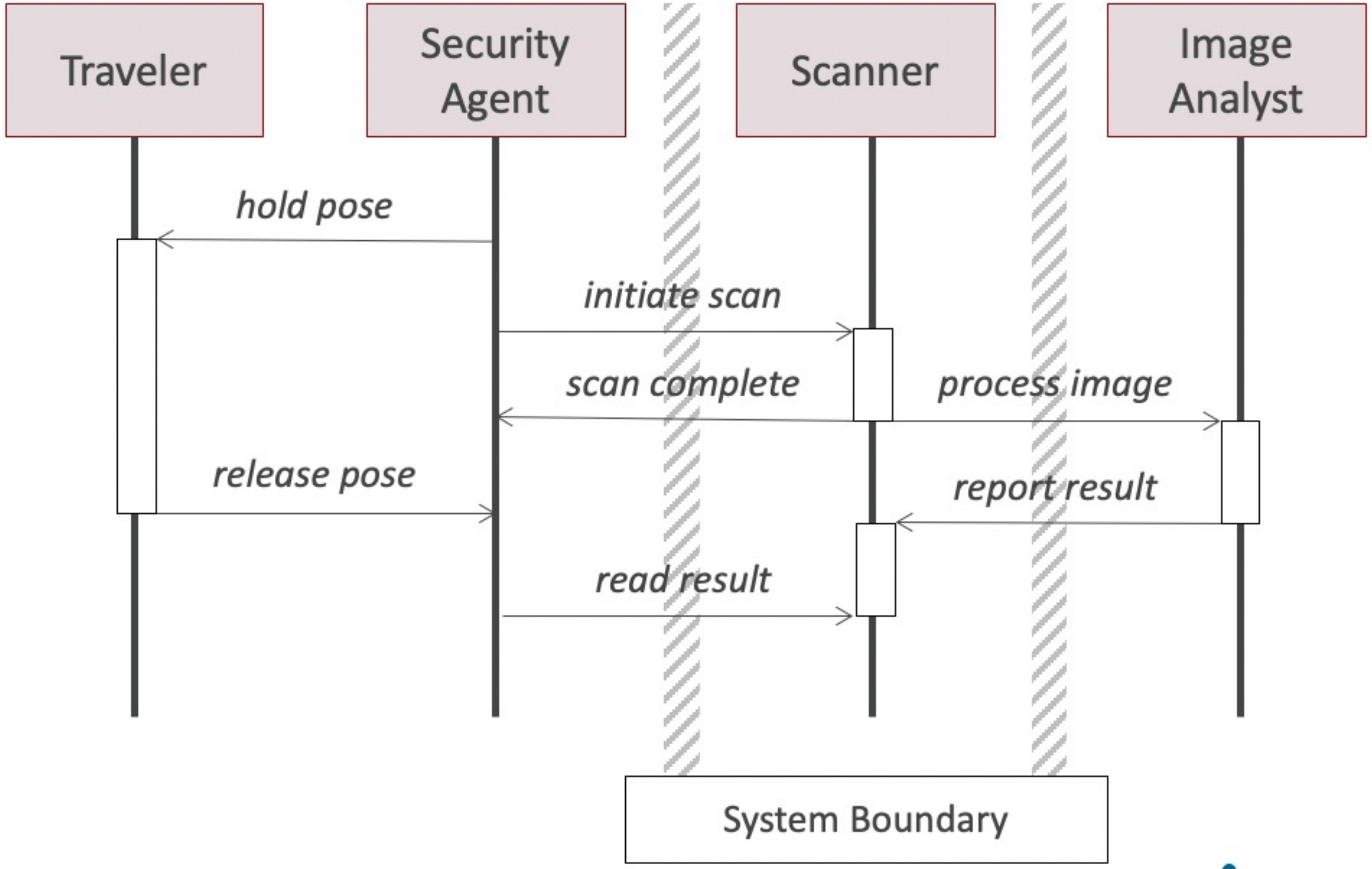
Behavioral UML Diagram - Activity Diagram

- The dynamic nature of a system by forming the flow of control from activity to activity
- model a business process, workflow, and internal operation



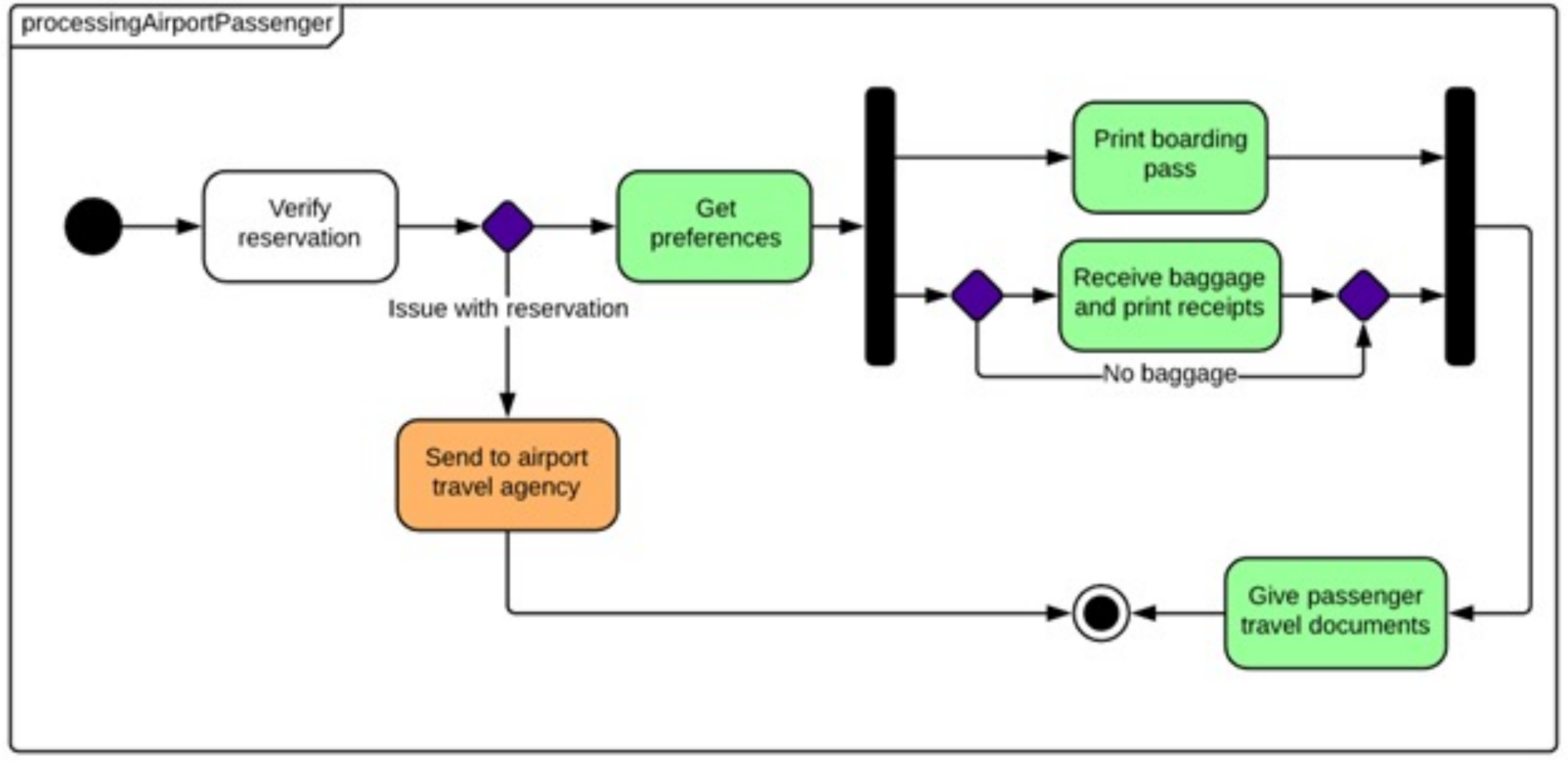
Behavioral UML Diagram - Sequence Diagram

- The time sequence of the objects participating in the interaction

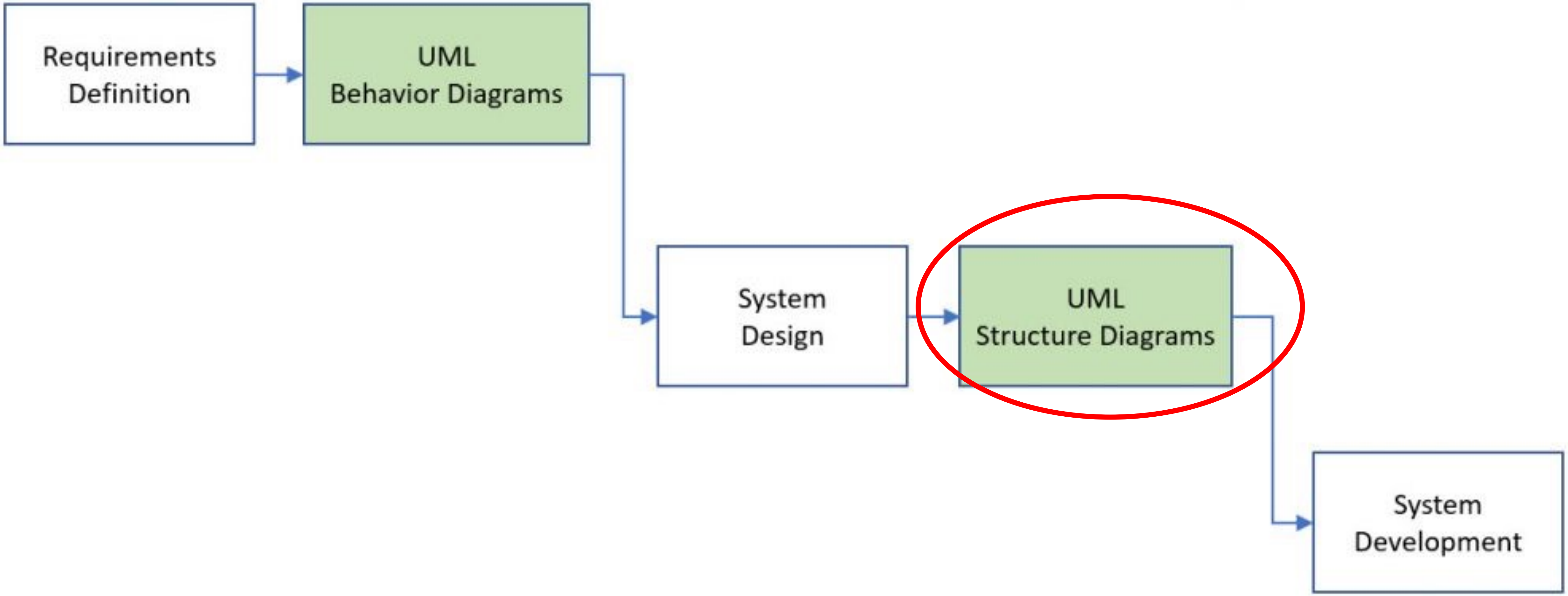


Behavioral UML Diagram – State Diagram

- possible states that an object of interaction goes through when an event occurs.


processingAirportPassenger
Verify reservation
Get preferences
Issue with reservation
Send to airport travel agency
Print boarding pass
Receive baggage and print receipts
No baggage
Give passenger travel documents

UML works best in a sequential Waterfall process



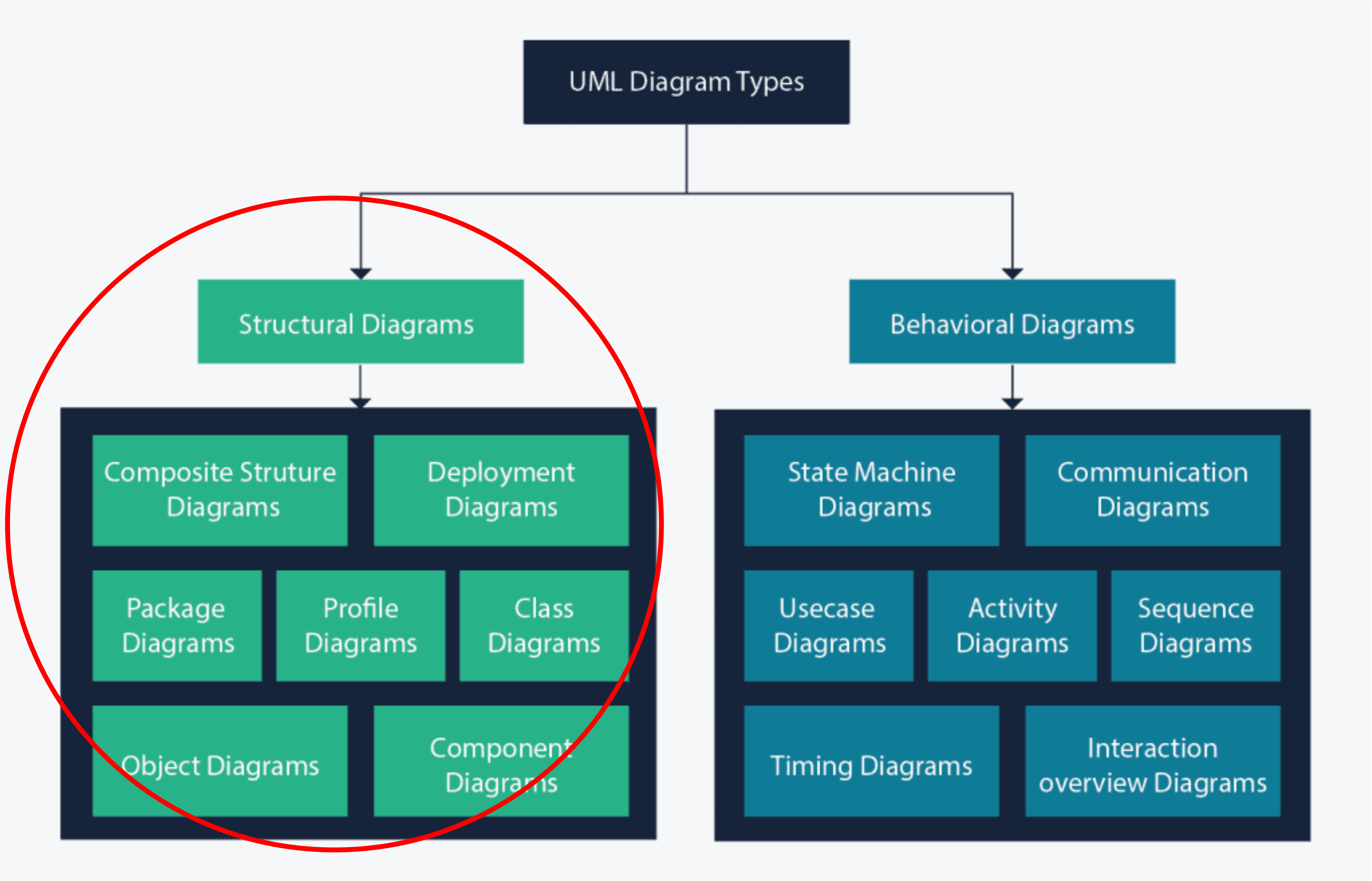
Requirements Definition

UML Behavior Diagrams

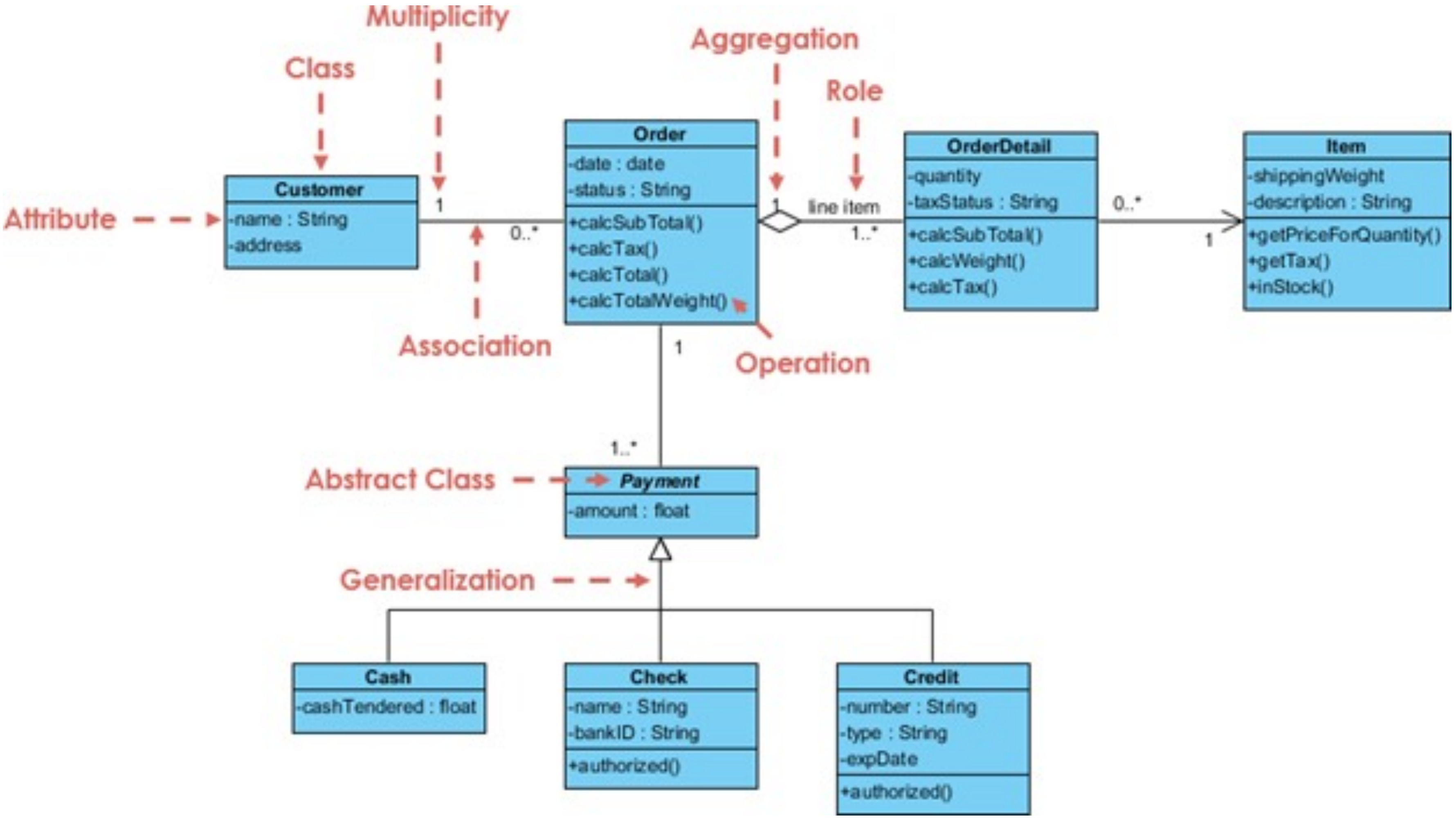
System Design

UML Structure Diagrams

System Development

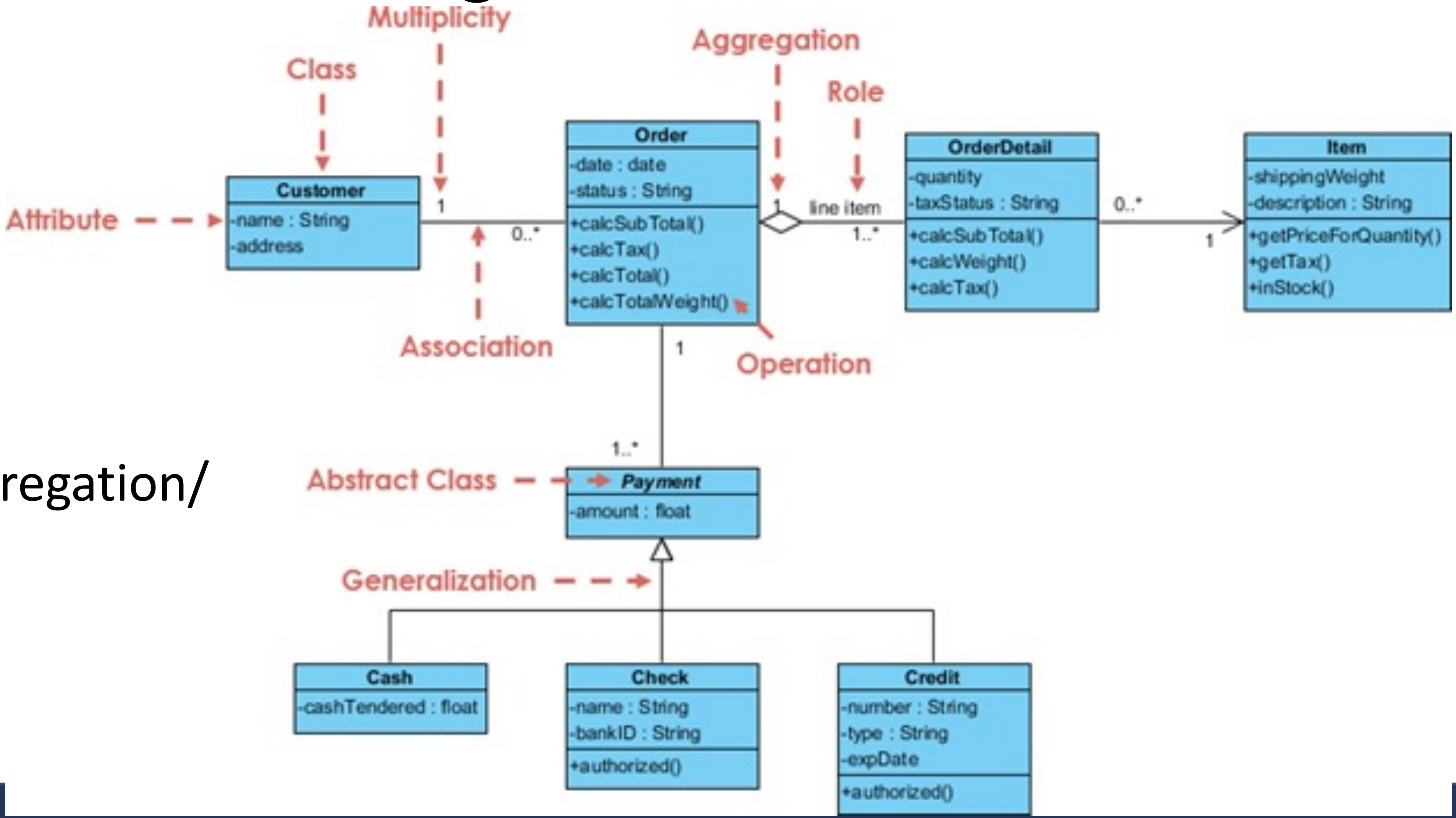

UML Diagram Types
Structural Diagrams
Behavioral Diagrams
Composite Struture Diagrams
Deployment Diagrams
Package Diagrams
Profile Diagrams
Class Diagrams
Object Diagrams
Component Diagrams
State Machine Diagrams
Communication Diagrams
Usecase Diagrams
Activity Diagrams
Sequence Diagrams
Timing Diagrams
Interaction overview Diagrams

Structural UML Diagram - Class Diagram



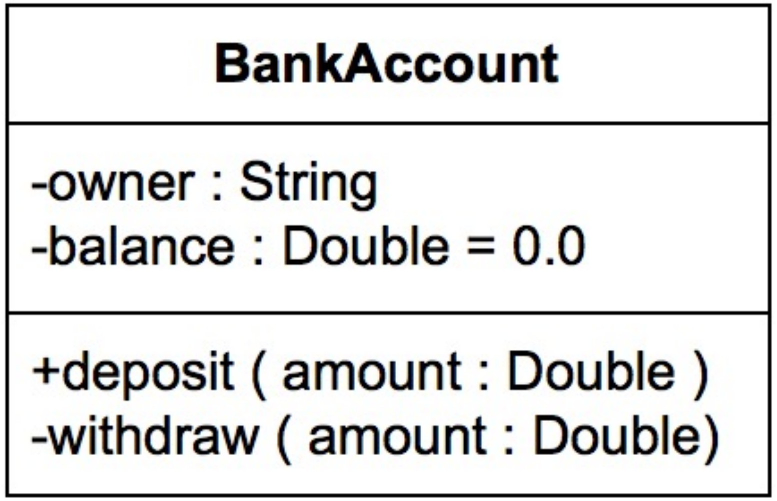
Elements of Class Diagram

- Class
  - attributes
  - operations
- Associations
  - multiplicity
  - direction/aggregation/
- Generalization



Class

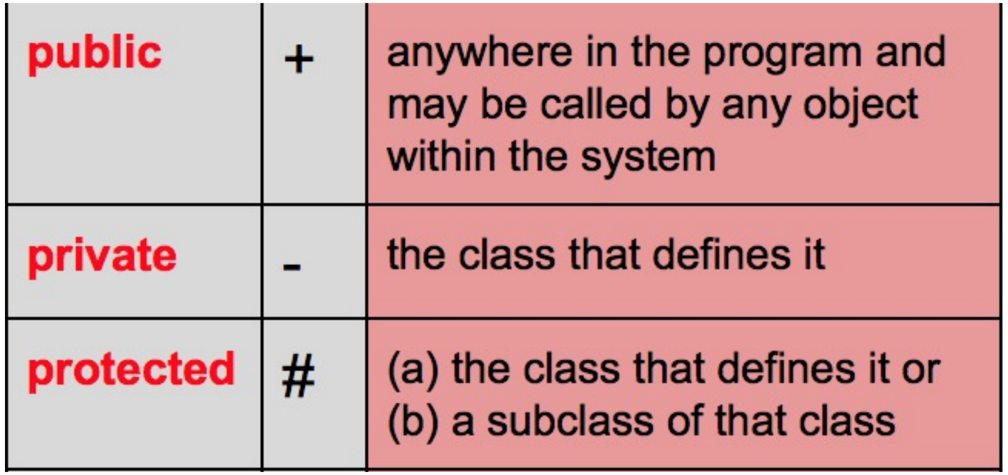
- class name
- class attributes [attribute name : type]
- class methods [parameter: type]



BankAccount

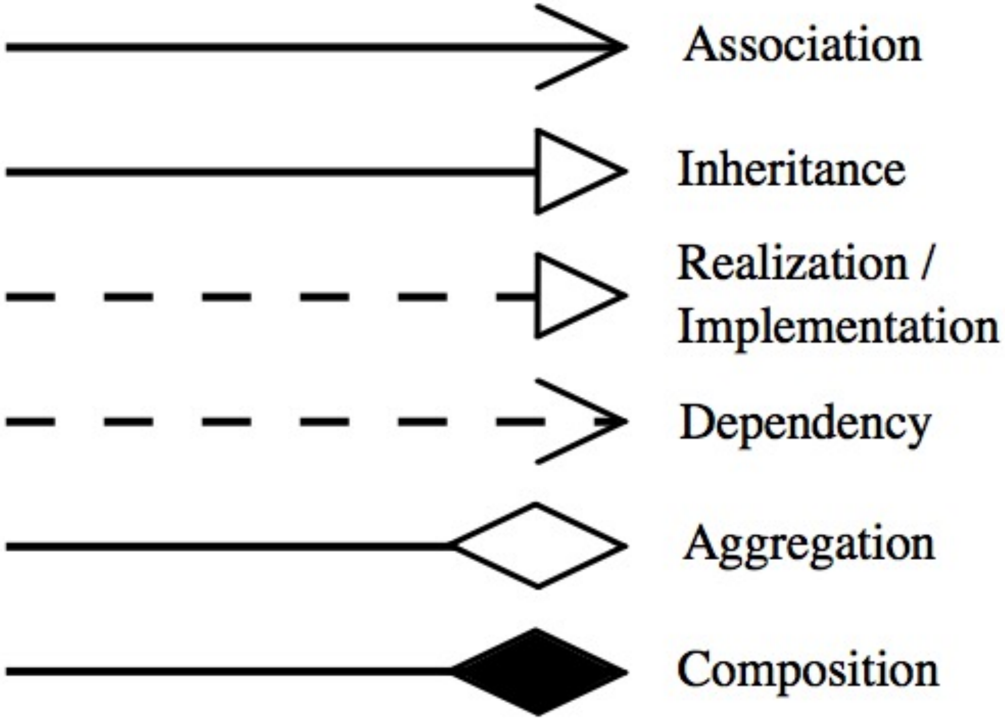
-owner : String
-balance : Double = 0.0

+deposit (amount : Double)
-withdraw (amount : Double)

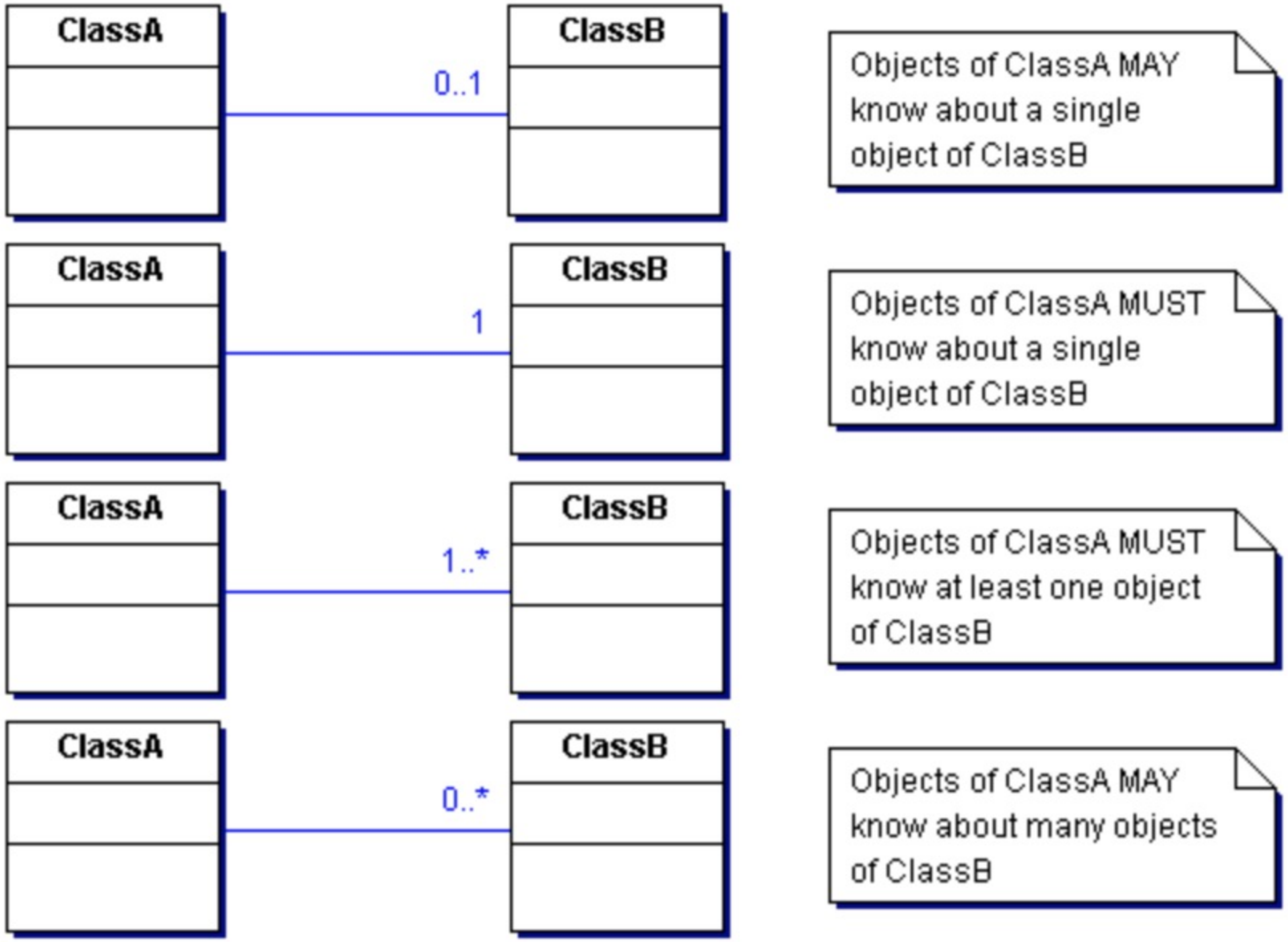


public	+	anywhere in the program and may be called by any object within the system
private	-	the class that defines it
protected	#	(a) the class that defines it or (b) a subclass of that class

Relationships



Association

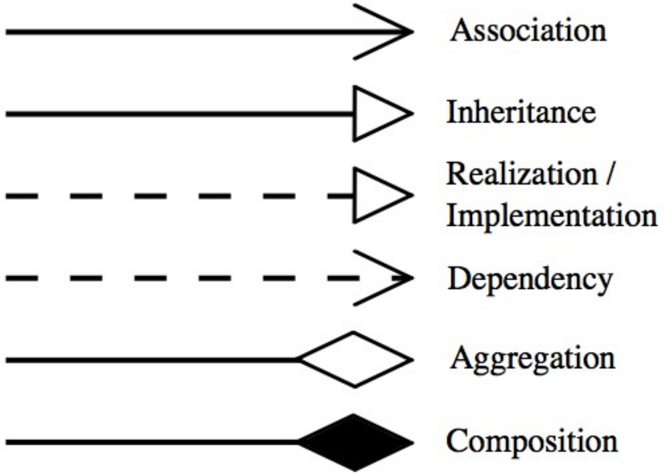


Objects of ClassA MAY know about a single object of ClassB

Objects of ClassA MUST know about a single object of ClassB

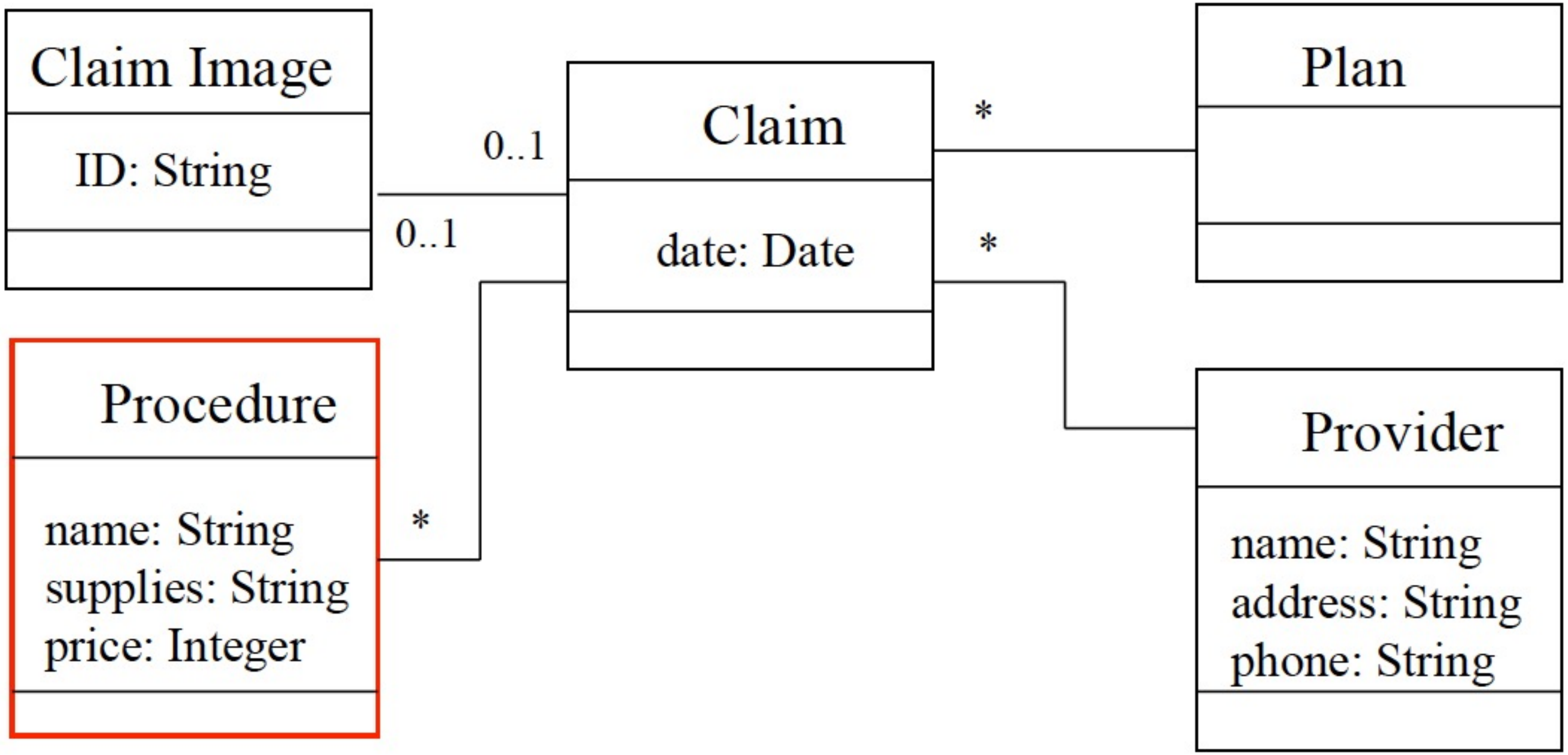
Objects of ClassA MUST know at least one object of ClassB

Objects of ClassA MAY know about many objects of ClassB

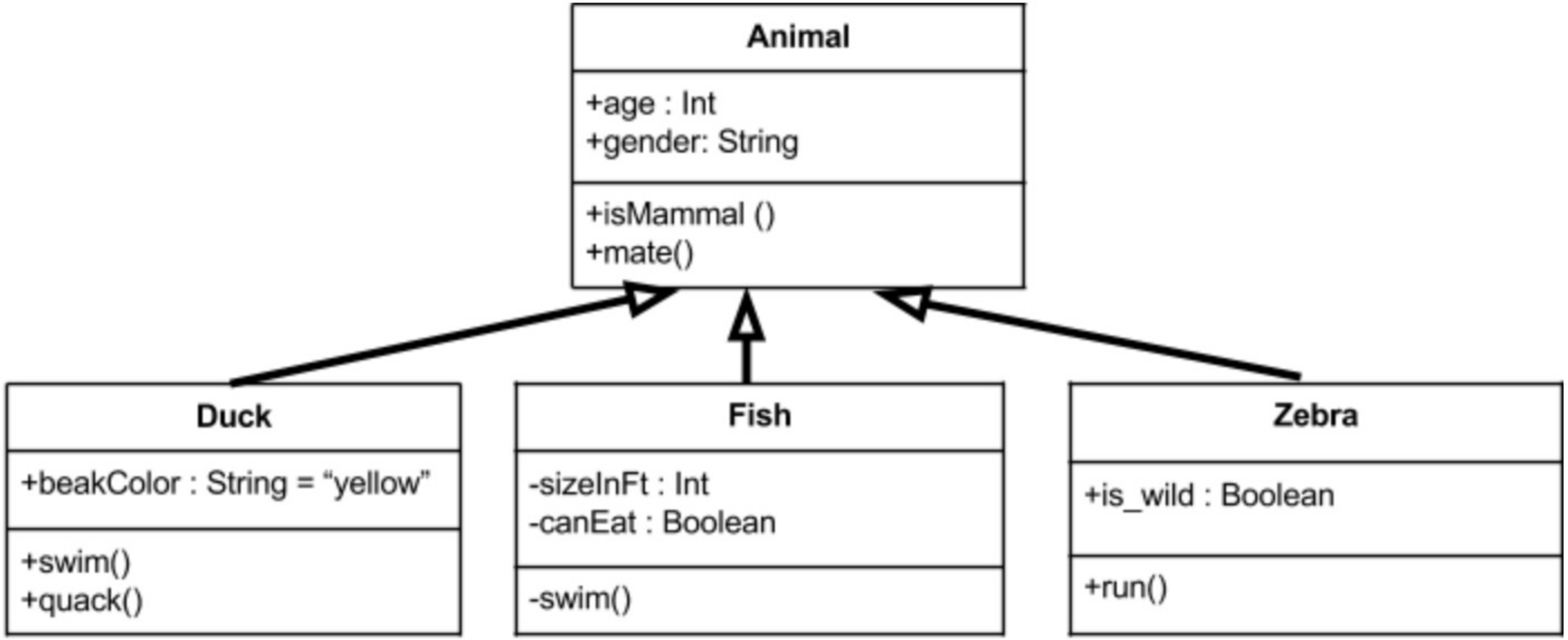


Multiplicity

Attributes vs Associations



Inheritance



Animal

+age : Int
+gender: String

+isMammal ()
+mate()

Duck

+beakColor : String = “yellow”

+swim()
+quack()

Fish

-sizeInFt : Int
-canEat : Boolean

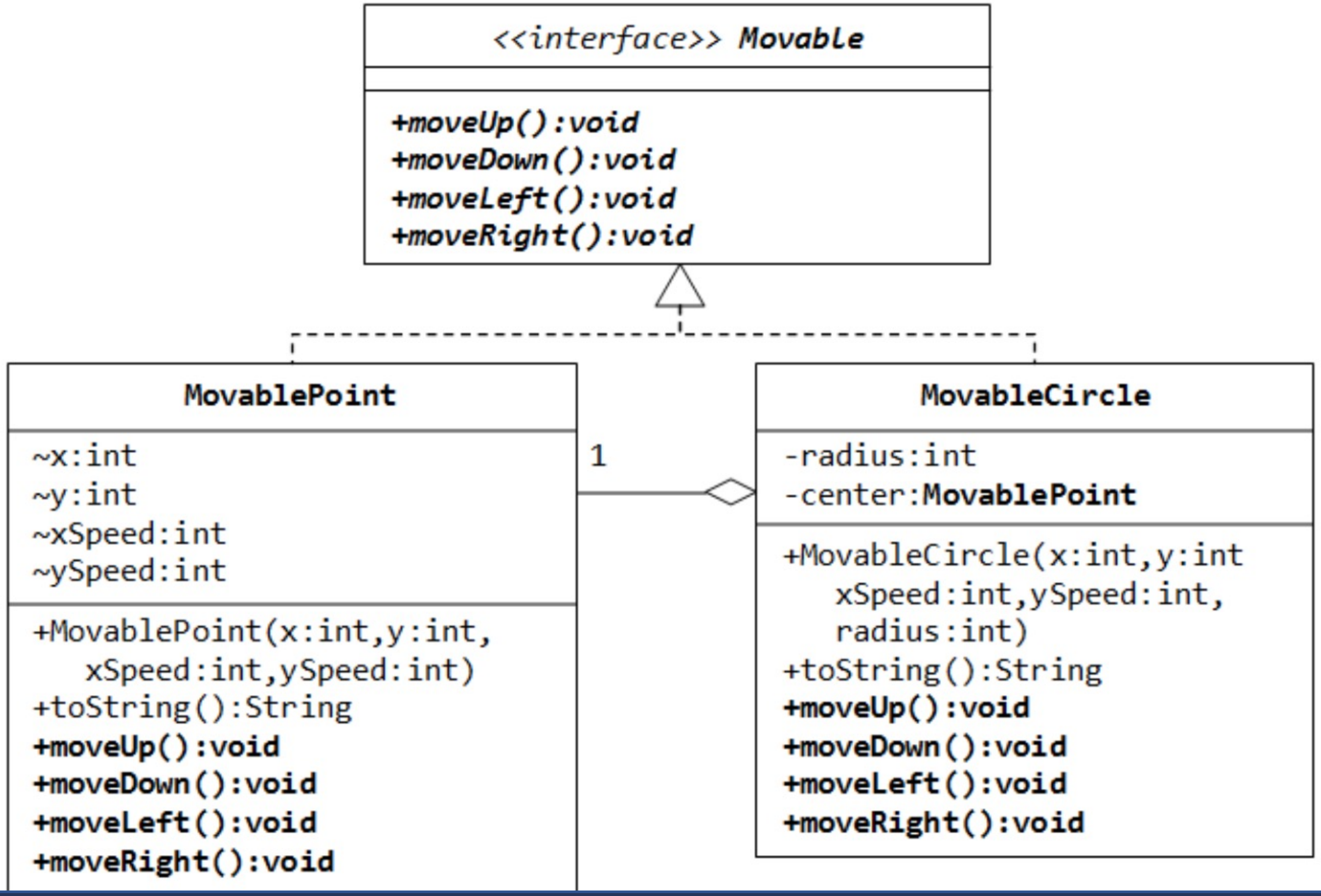
-swim()

Zebra

+is_wild : Boolean

+run()

Realization/Implementation



<<interface>> ***Movable***

+moveUp():void
+moveDown():void
+moveLeft():void
+moveRight():void

MovablePoint

~x:int
~y:int
~xSpeed:int
~ySpeed:int

+MovablePoint(x:int,y:int,
xSpeed:int,ySpeed:int)
+toString():String
+moveUp():void
+moveDown():void
+moveLeft():void
+moveRight():void

1

MovableCircle

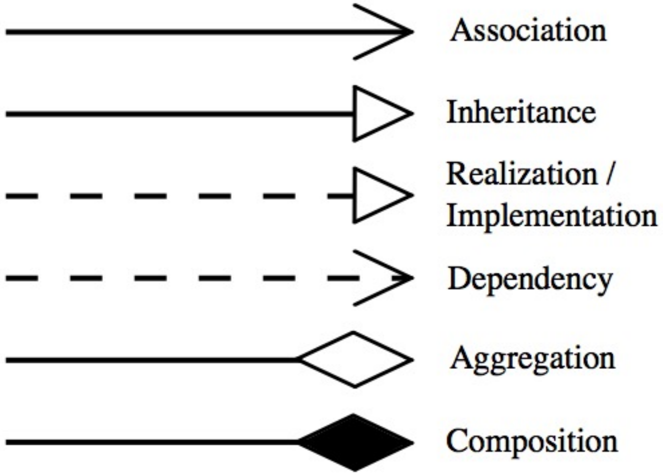
-radius:int
-center:**MovablePoint**

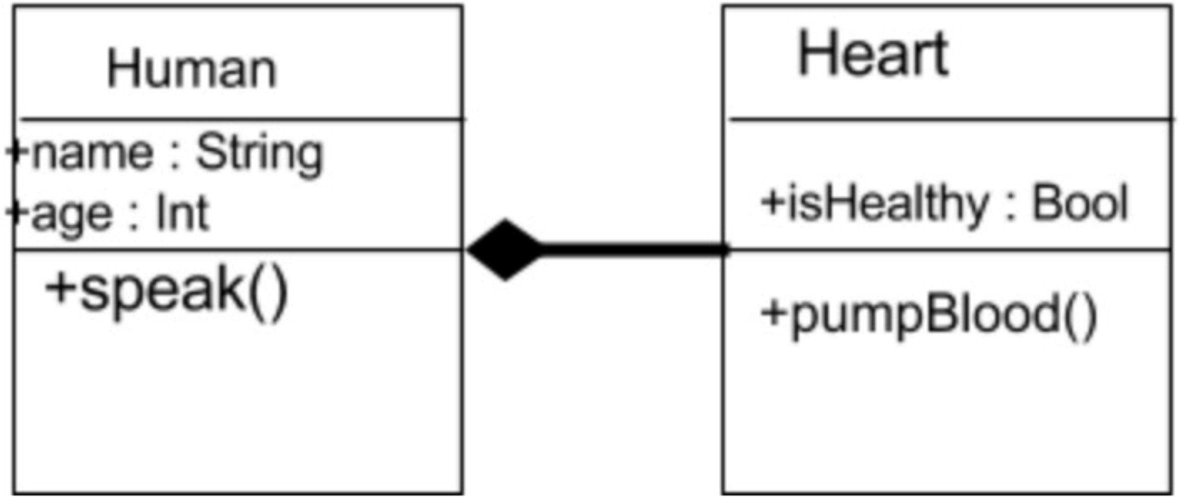
+MovableCircle(x:int,y:int
xSpeed:int,ySpeed:int,
radius:int)
+toString():String
+moveUp():void
+moveDown():void
+moveLeft():void
+moveRight():void

Aggregation

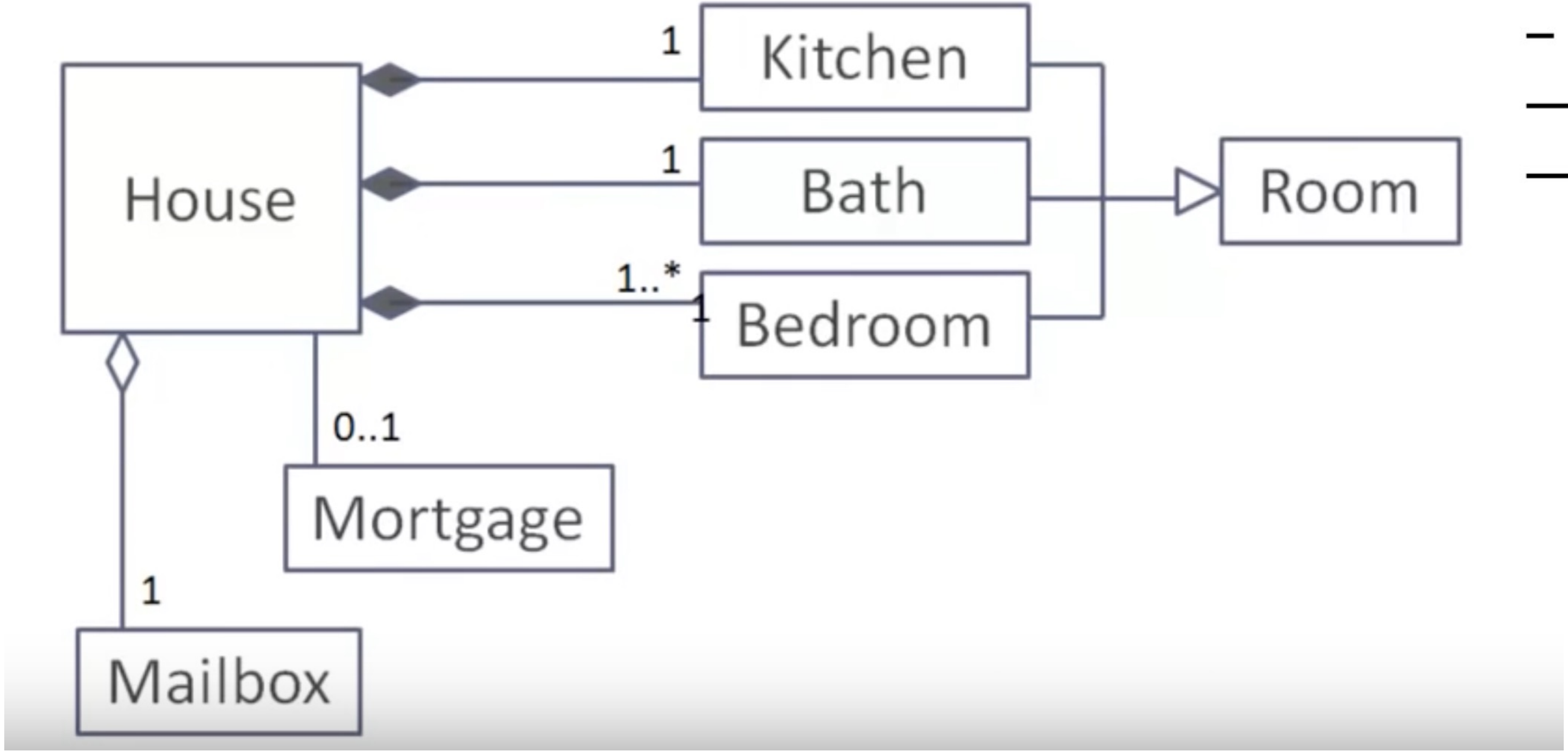
- “has a”
- “is part of”

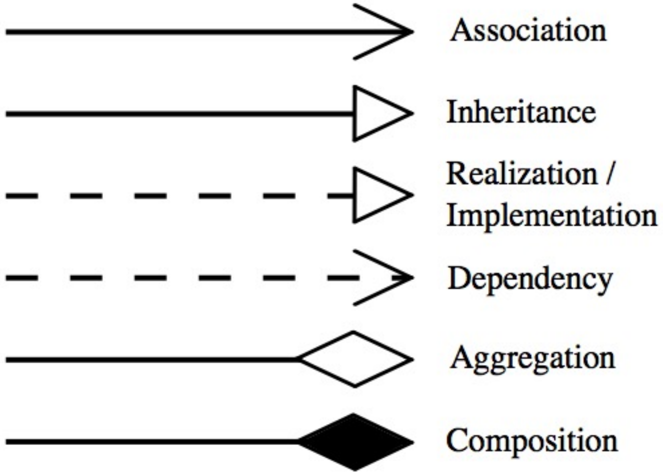
Composition





Multiplicity



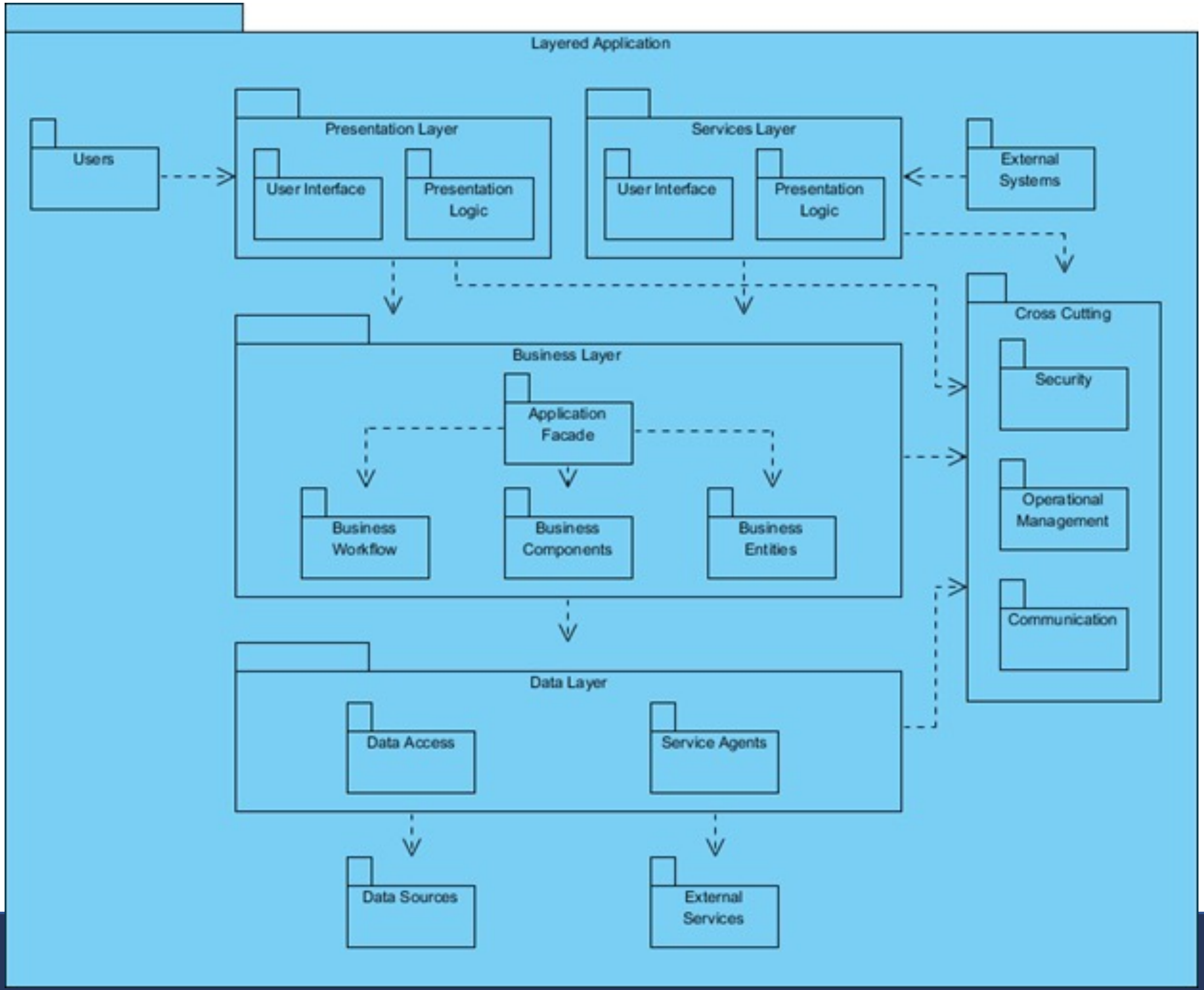


Analysis vs Design

- Class diagrams are used in both analysis and design
- Analysis - conceptual
  - model problem, not software solution
  - can include actors outside system
- Design - specification
  - tells how the system should act
- Design – implementation
  - actual classes of implementation

Structural UML Diagram - Package Diagram

- Package UML diagrams bring together the elements of a system into related groups to reduce dependencies between sets.

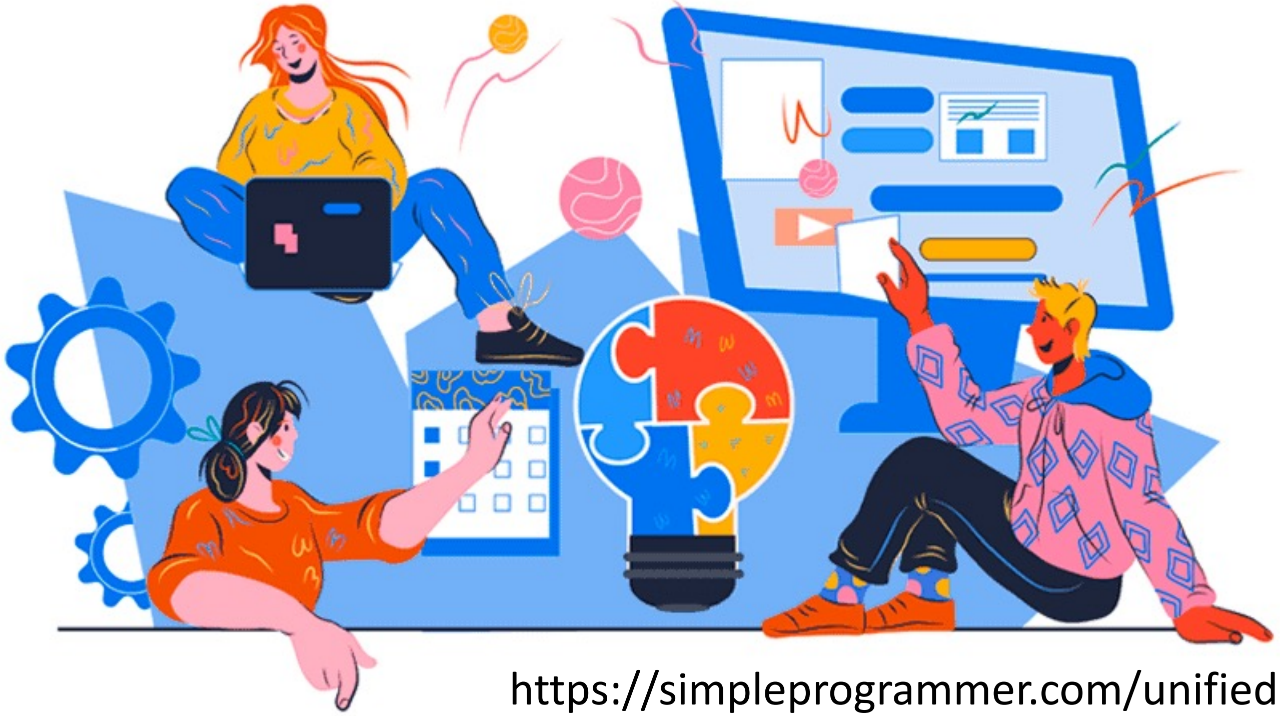


Readings

- More UML resources
- http://dn.codegear.com/article/31863
- http://www.sparxsystems.com.au/
- UML_Tutorial.htm

  http://www.gnome.org/projects/dia/umltut/index.html

THE UML IN THE AGE OF AGILE: WHY IT'S STILL RELEVANT



https://simpleprogrammer.com/unified-modeling-language-age-of-agile/

Architecture Patterns, and Tactics

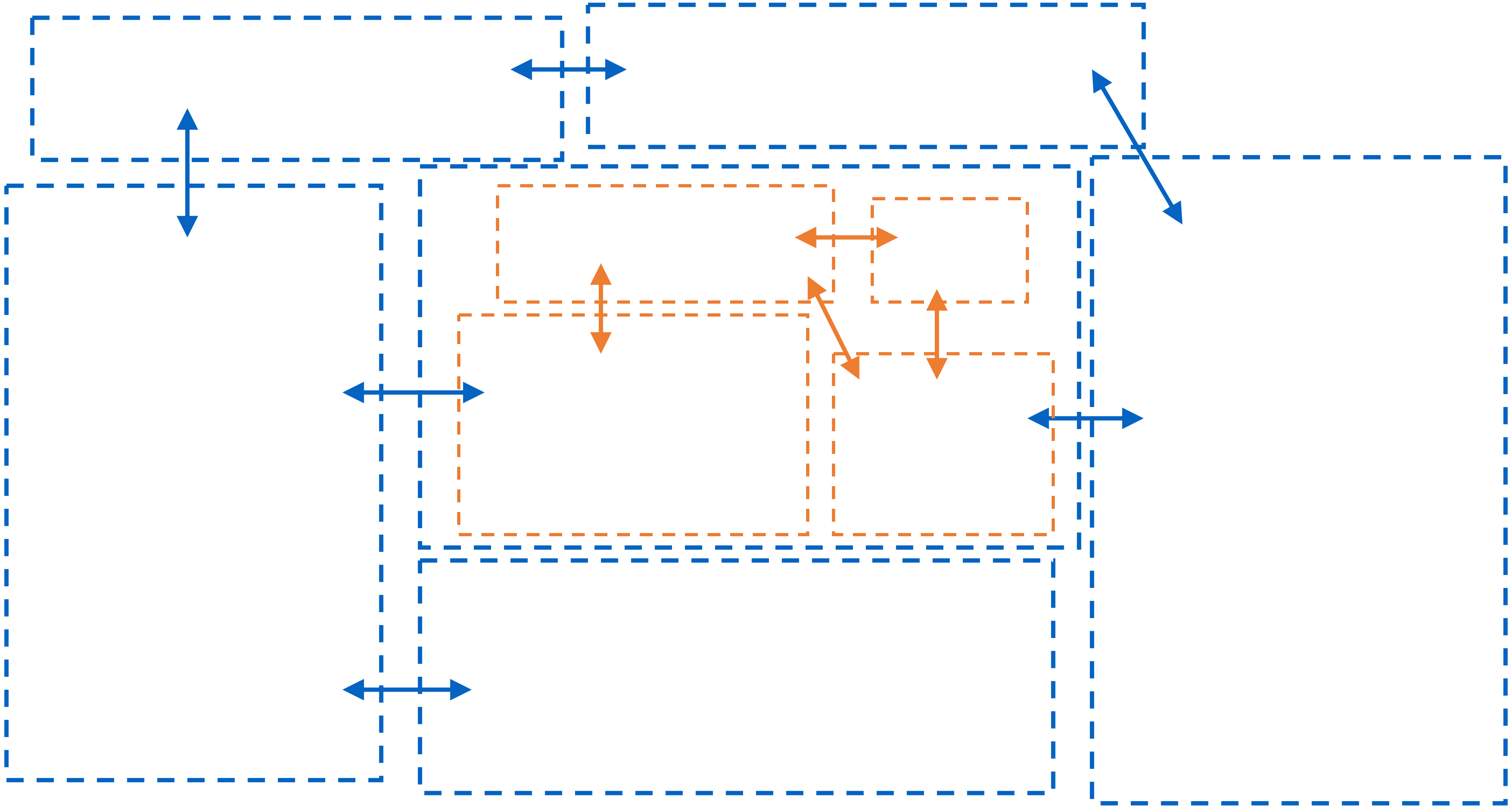
The Edward S. Rogers Sr. Department
of Electrical & Computer Engineering
UNIVERSITY OF TORONTO

Architecture vs Object-level Design

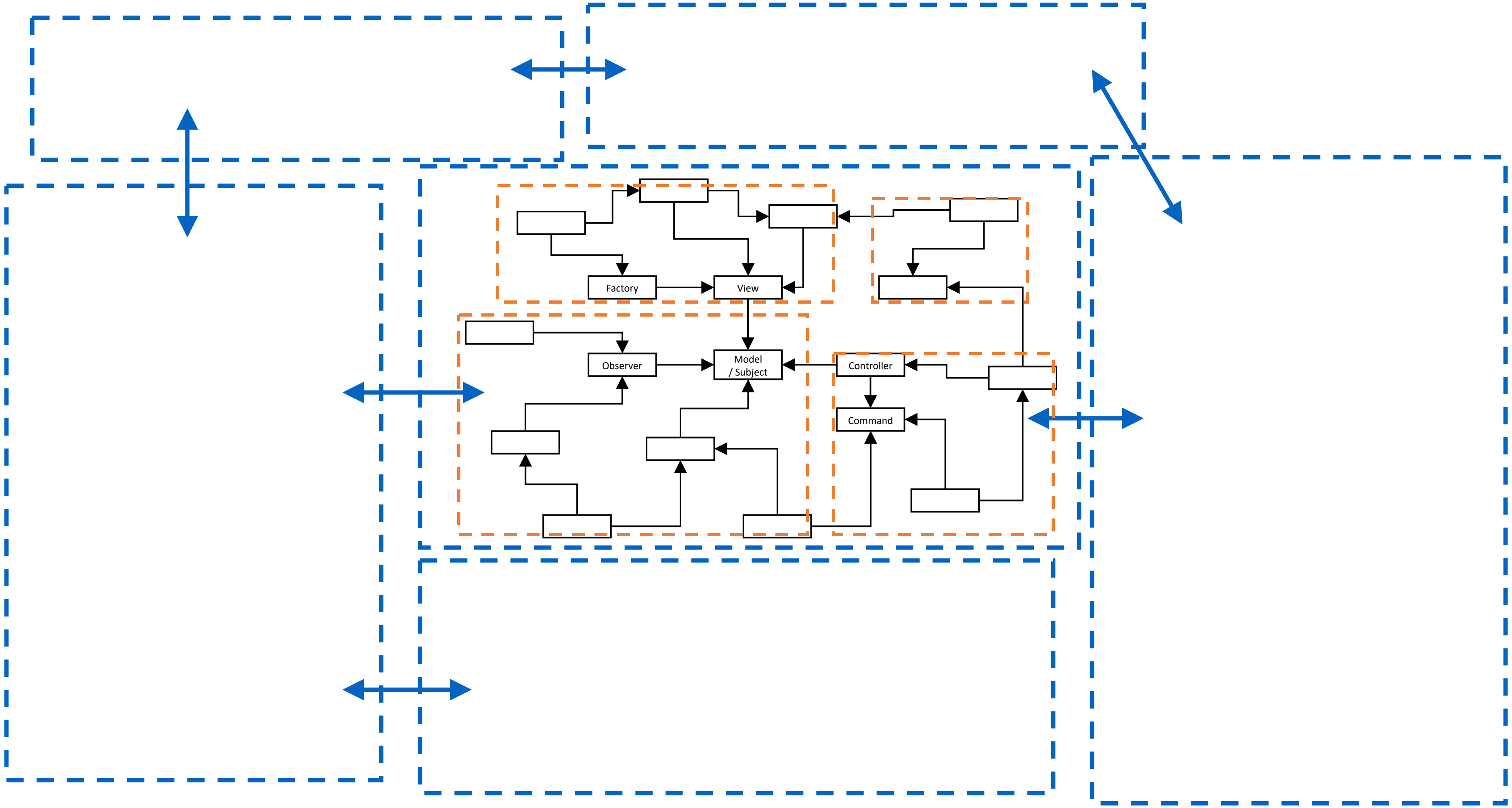
Levels of Abstraction

- 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”

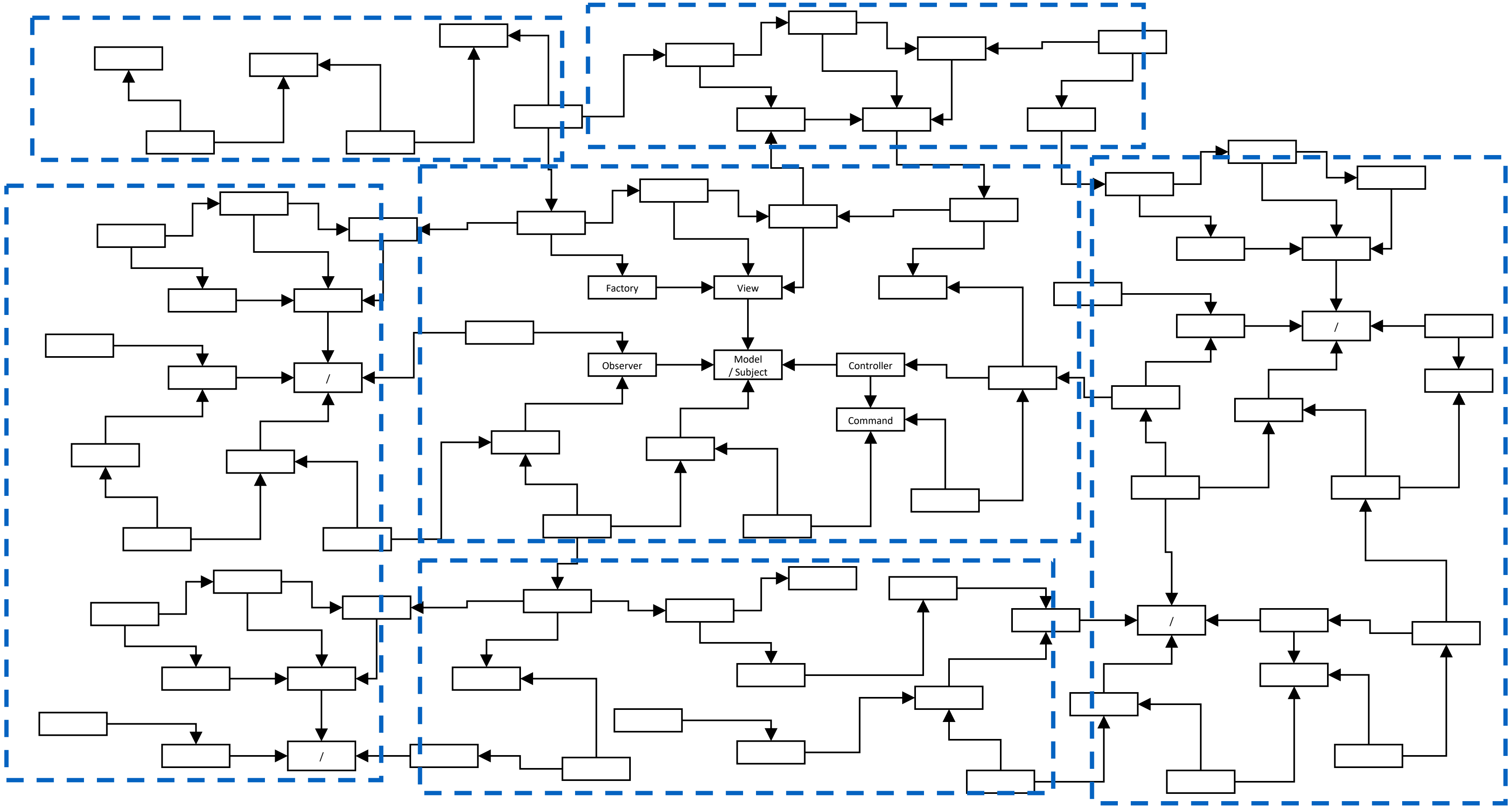
Architecture



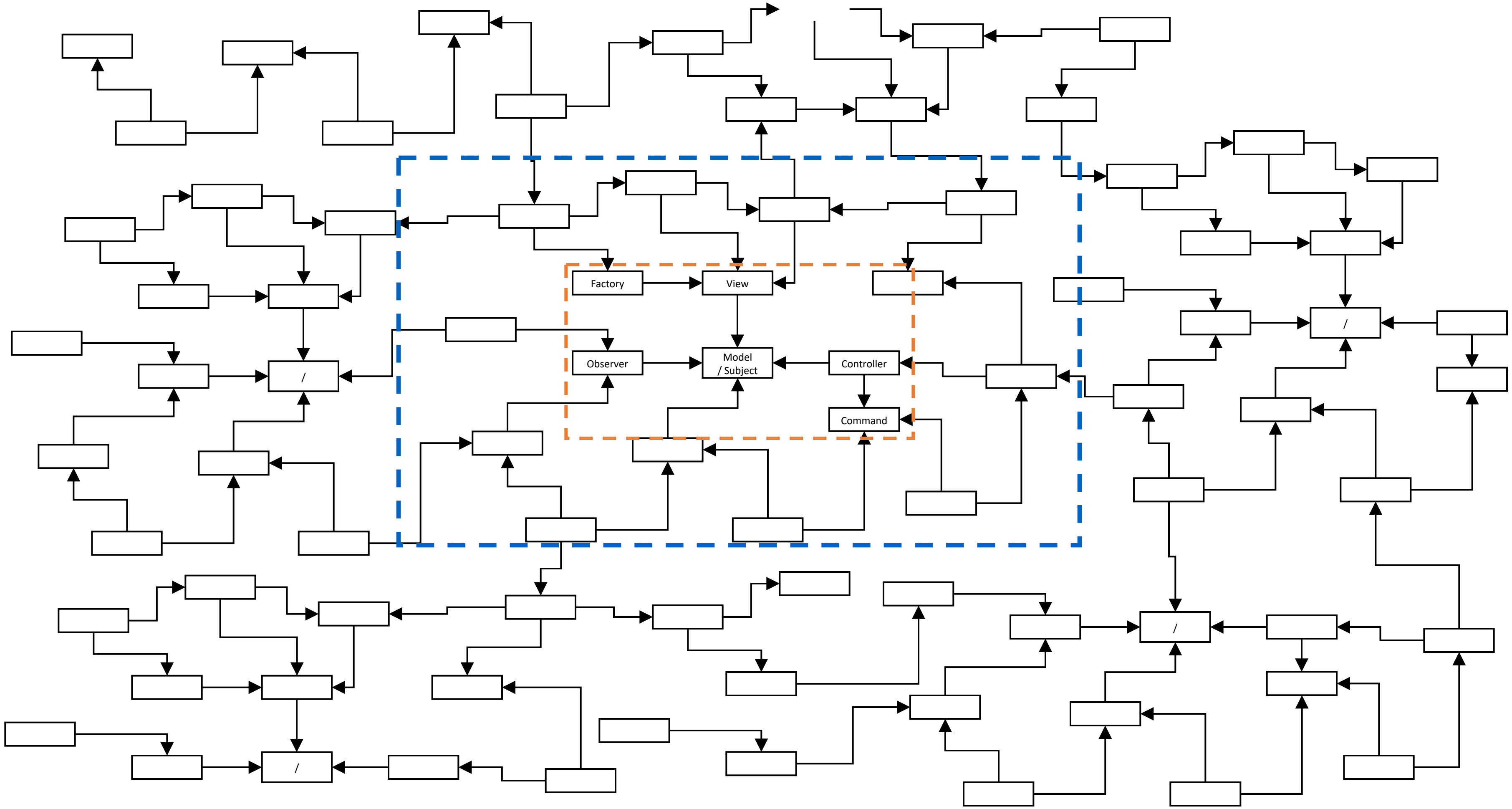
Architecture



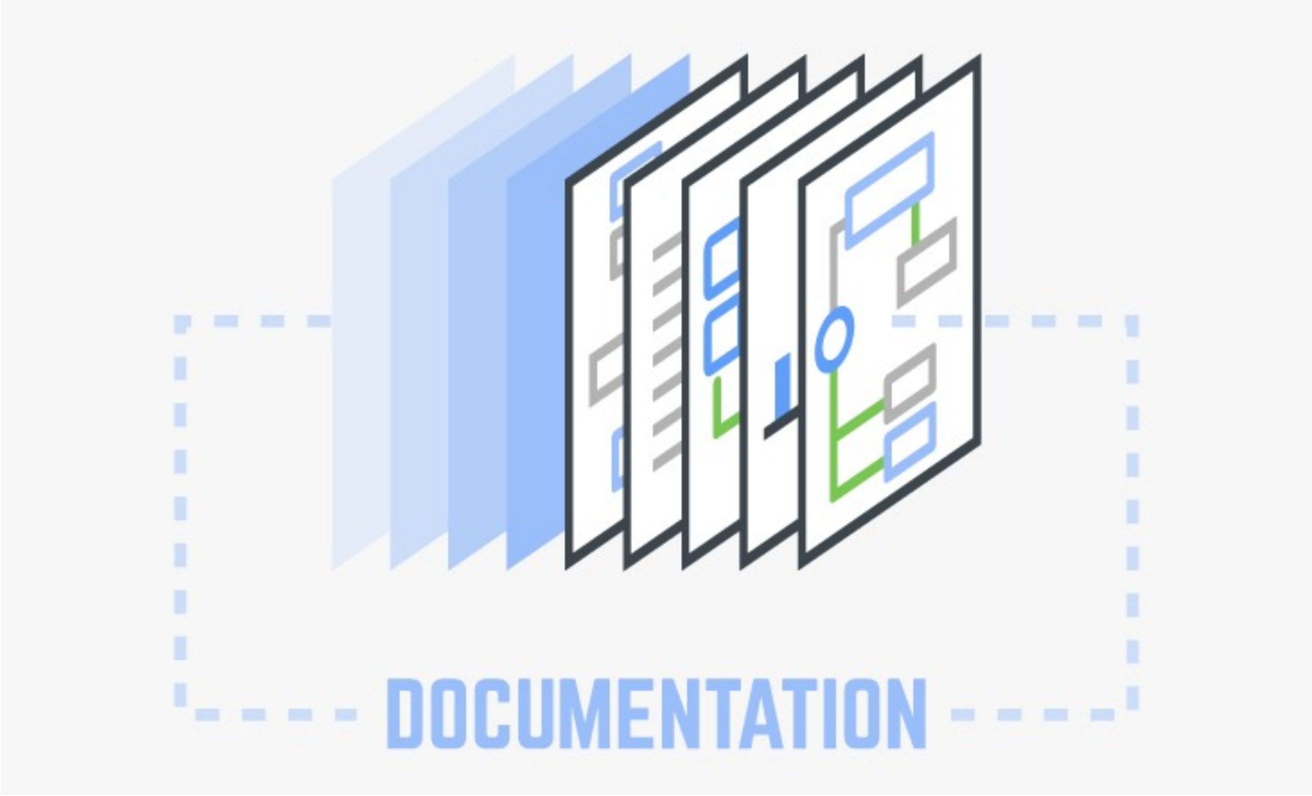
Architecture



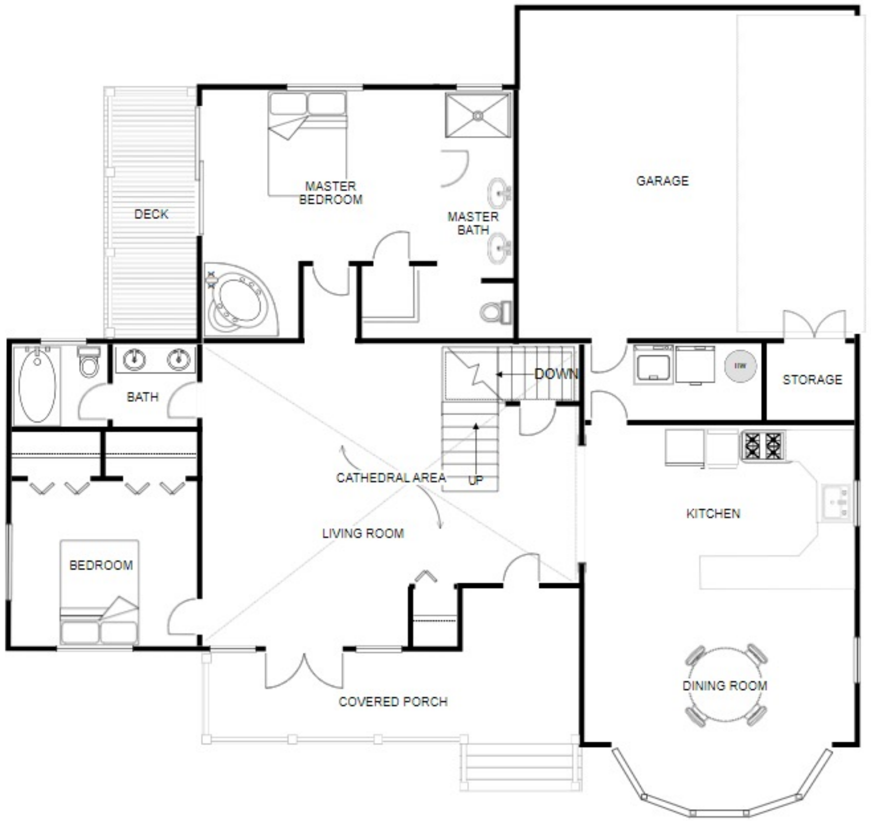
Design Patterns



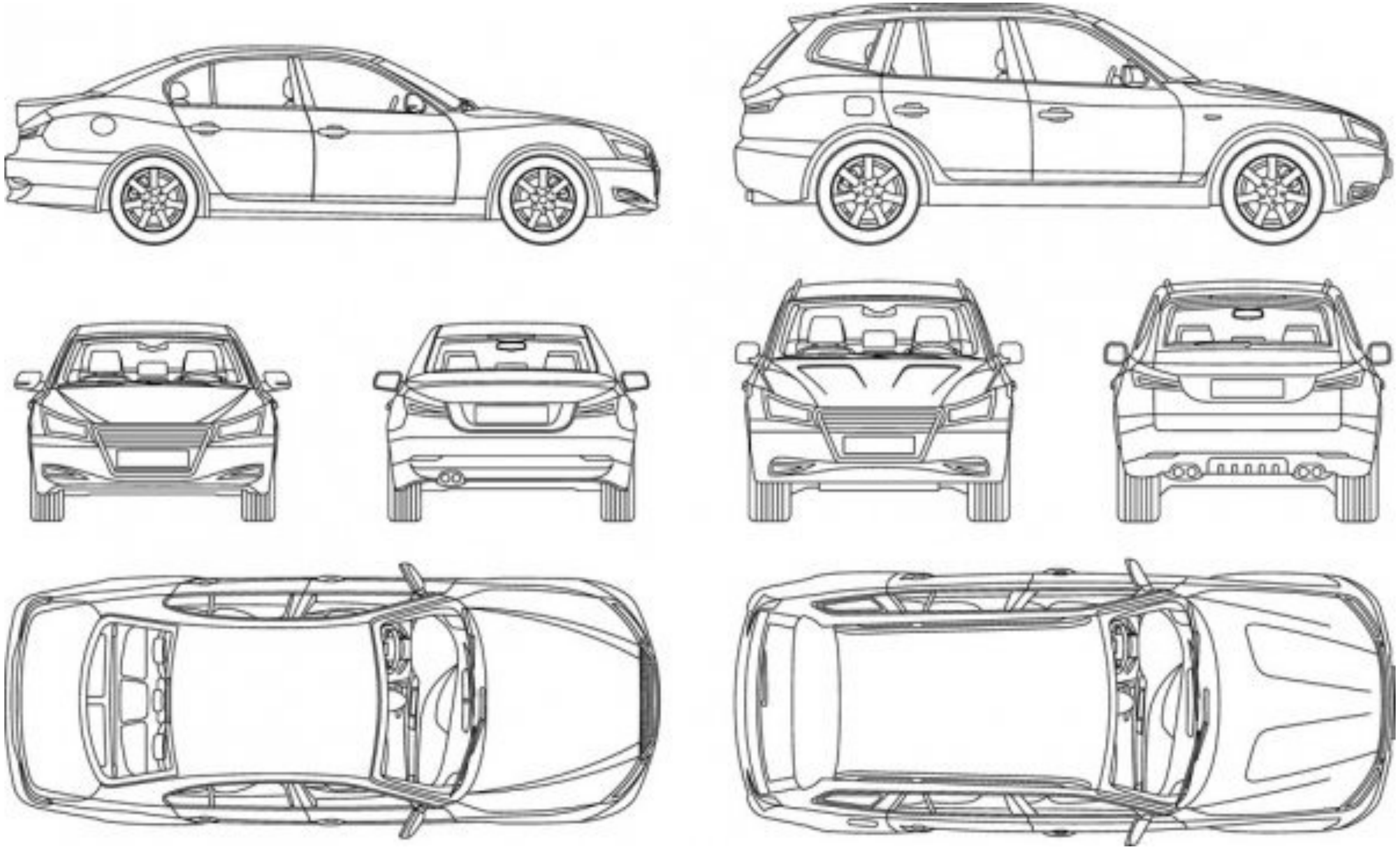
Architecture Documentation & Views



Every engineered artifact has an architecture



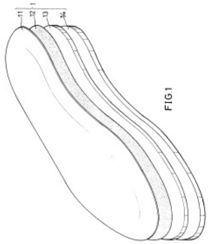
Blueprint



Architecture Disentangled

Architecture as
structures and relations
(the actual system)

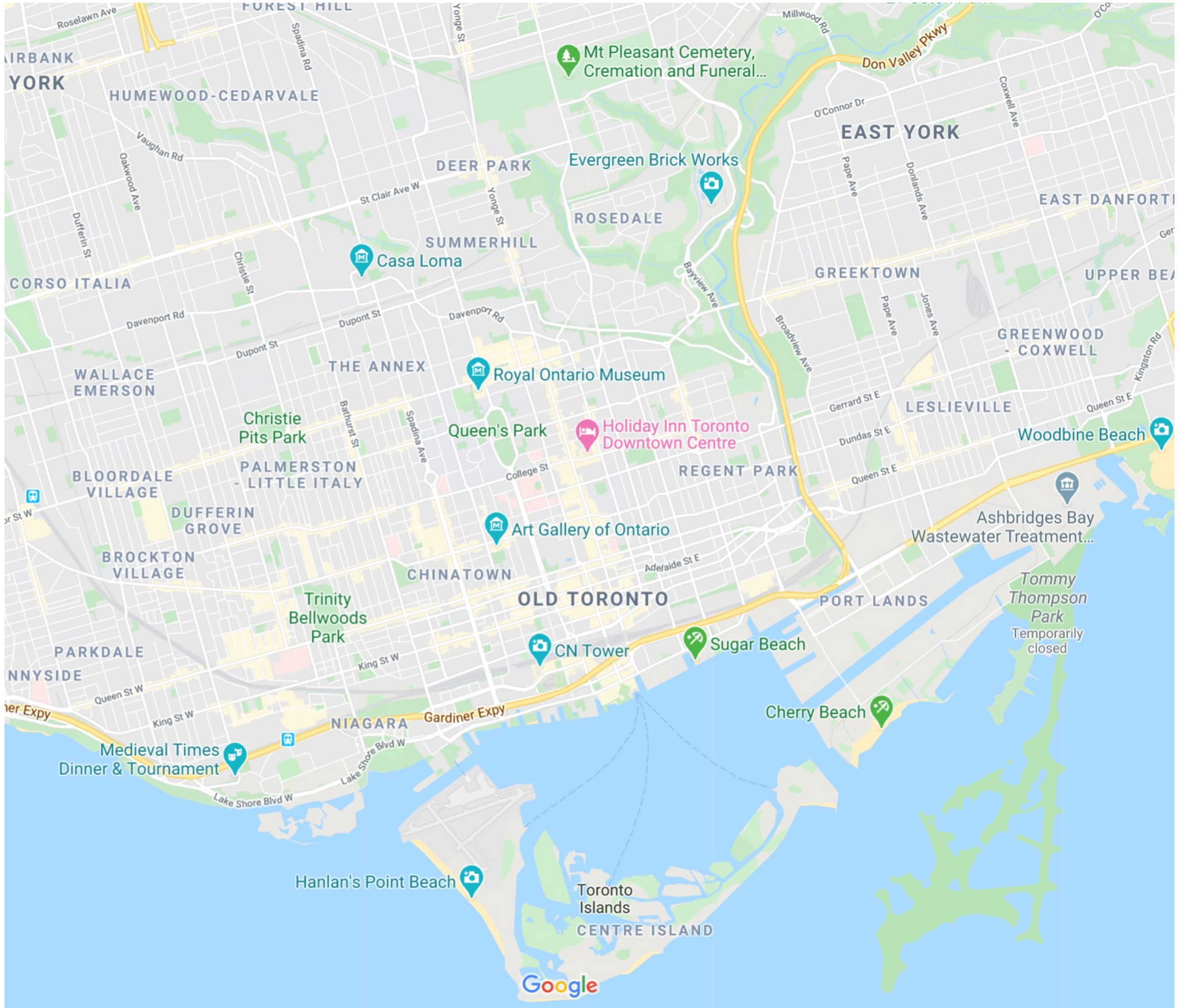
Architecture as
documentation
(representations of the system)

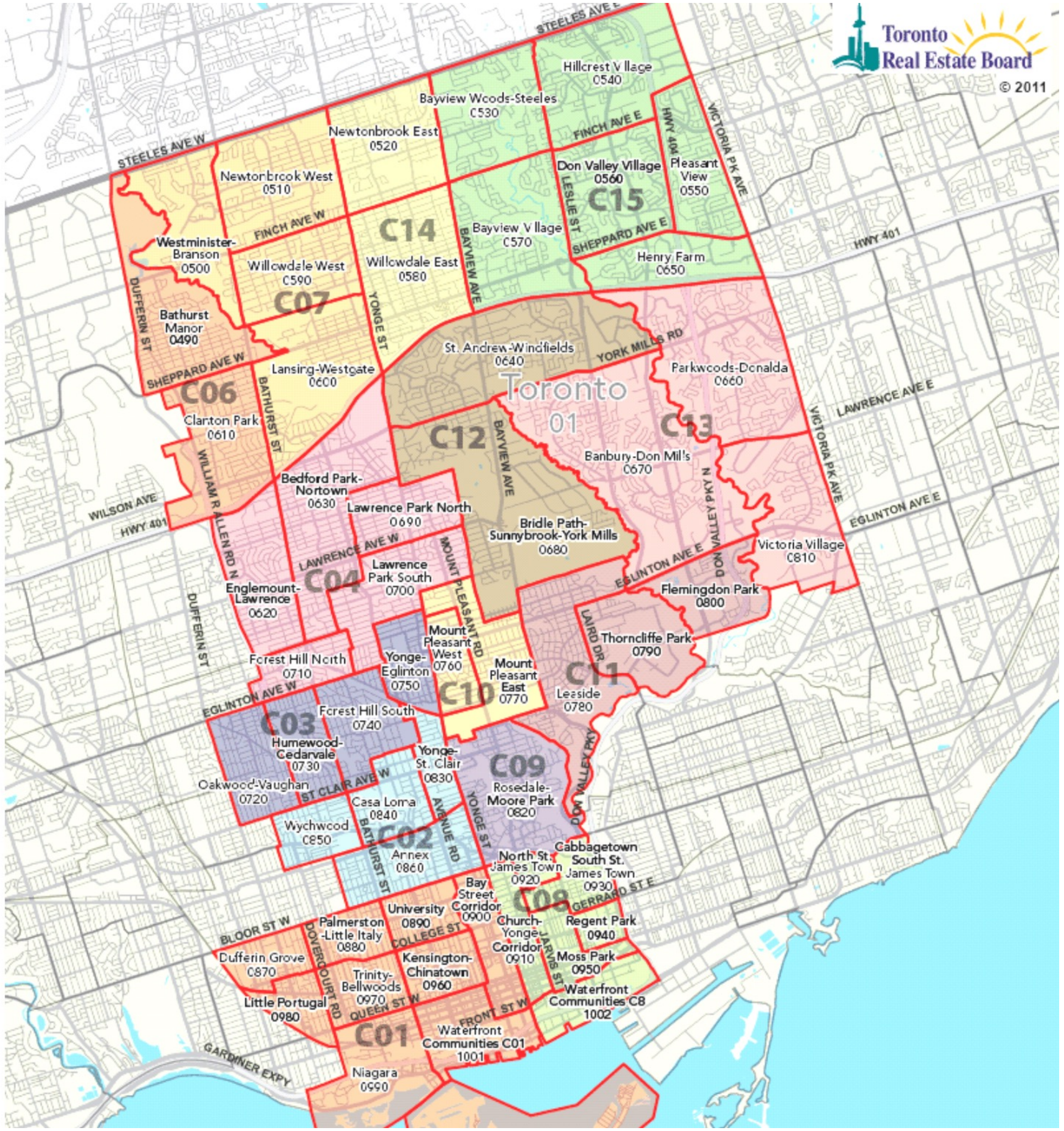


Architecture as (design) process
(activities around the other two)

Why Document Architecture?

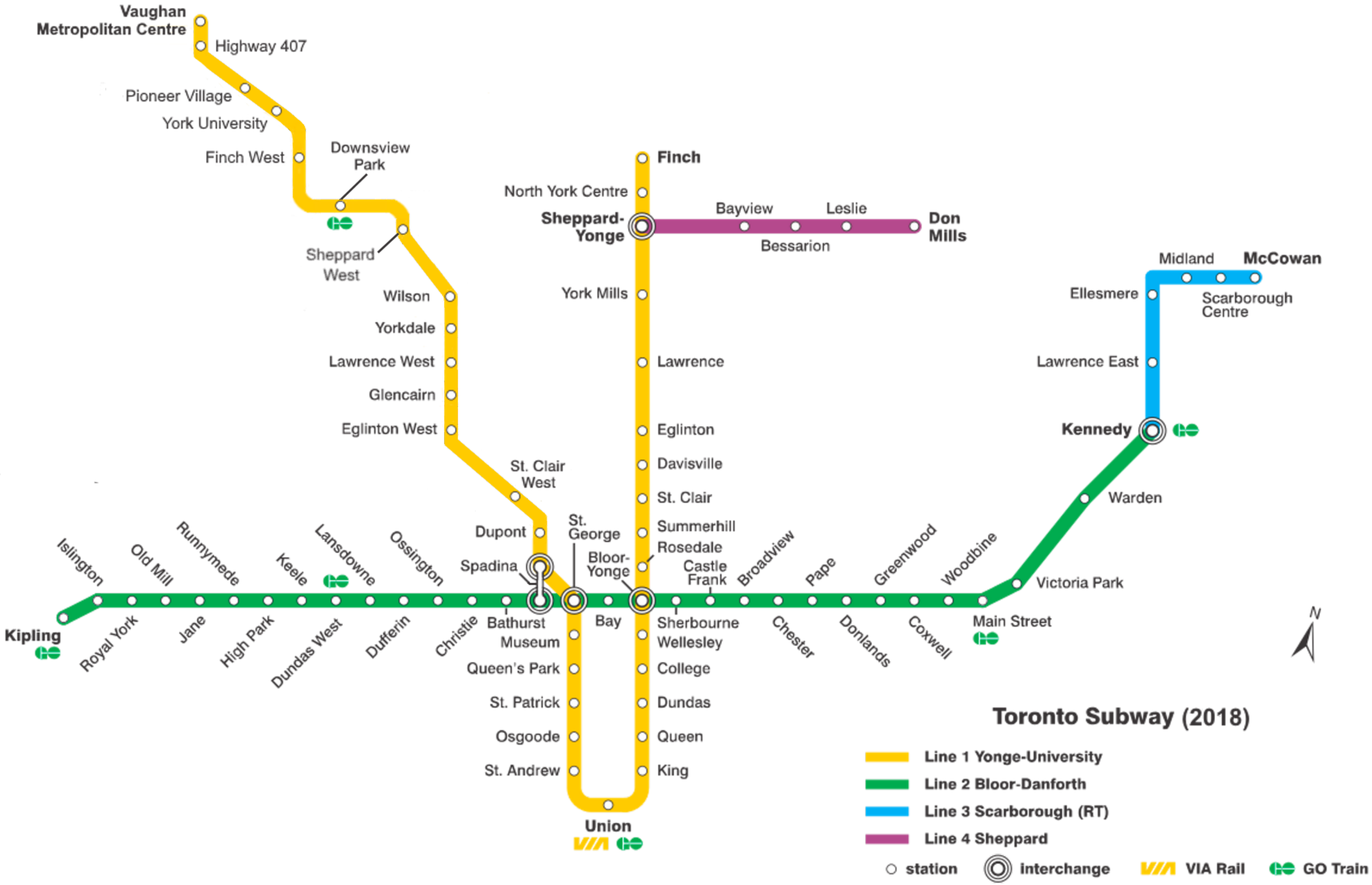
- Blueprint for the system
- Documentation speaks for the architect, today and 20 years from today
- Support traceability.


FOREST HILL
Roselawn Ave
AIRBANK
YORK
HUMEWOOD-CEDARVALE
Spadina Rd
Yonge St
Millwood Rd
Mt Pleasant Cemetery, Cremation and Funeral...
Don Valley Pkwy
O'Connor Dr
EAST YORK
Coxwell Ave
Vaughan Rd
Oakwood Ave
DEER PARK
St Clair Ave W
Evergreen Brick Works
Pape Ave
Donlands Ave
EAST DANFORTH
ROSEDALE
Dufferin St
SUMMERHILL
Casa Loma
Christie St
Bayview Ave
GREEKTOWN
UPPER BEACHES
CORSO ITALIA
Davenport Rd
Dupont St
Davenport Rd
Jones Ave
Broadview Ave
GREENWOOD - COXWELL
Dupont St
THE ANNEX
Royal Ontario Museum
Kingston Rd
WALLACE EMERSON
Gerrard St E
LESLIEVILLE
Queen St E
Christie Pits Park
Bathurst St
Spadina Ave
Queen's Park
Holiday Inn Toronto Downtown Centre
Dundas St E
Woodbine Beach
BLOORDALE VILLAGE
PALMERSTON - LITTLE ITALY
College St
REGENT PARK
Queen St E
DUFFERIN GROVE
Art Gallery of Ontario
Ashbridges Bay Wastewater Treatment...
BROCKTON VILLAGE
CHINATOWN
Adelaide St E
Tommy Thompson Park
Temporarily closed
Trinity Bellwoods Park
OLD TORONTO
PORT LANDS
PARKDALE
CN Tower
Sugar Beach
King St W
NNYSIDE
Queen St W
Gardiner Expy
Cherry Beach
King St W
NIAGARA
Gardiner Expy
Medieval Times Dinner & Tournament
Lake Shore Blvd W
Lake Shore Blvd W
Hanlan's Point Beach
Toronto Islands
CENTRE ISLAND
Google



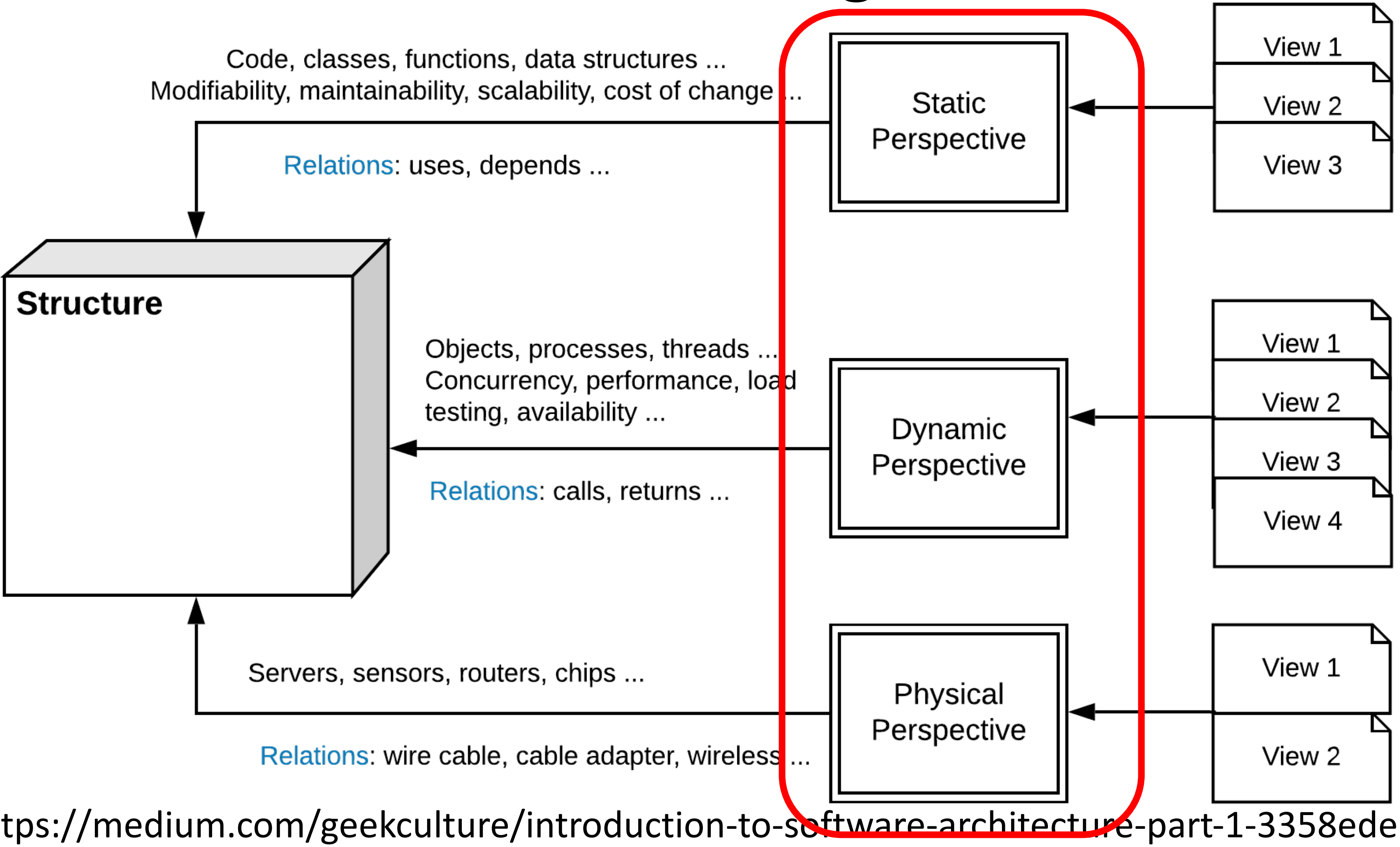
http://jackiecarron.com/real_estate/toronto/buyers_zone_maps.shtml

TORONTO
The Annex
The Junction
Harbord Village
8
3
Little Italy
1
6
5
Brockton Village
2
Little Portugal
Chinatown
High Park
Fashion District
7
4
9
LAKE ONTARIO



Toronto Subway (2018)

Common Views in Documenting Software Architecture


Code, classes, functions, data structures ...
Modifiability, maintainability, scalability, cost of change ...
Relations: uses, depends ...
Static Perspective
View 1
View 2
View 3
Structure
Objects, processes, threads ...
Concurrency, performance, load testing, availability ...
Relations: calls, returns ...
Dynamic Perspective
View 1
View 2
View 3
View 4
Servers, sensors, routers, chips ...
Relations: wire cable, cable adapter, wireless ...
Physical Perspective
View 1
View 2

https://medium.com/geekculture/introduction-to-software-architecture-part-1-3358ede31af9

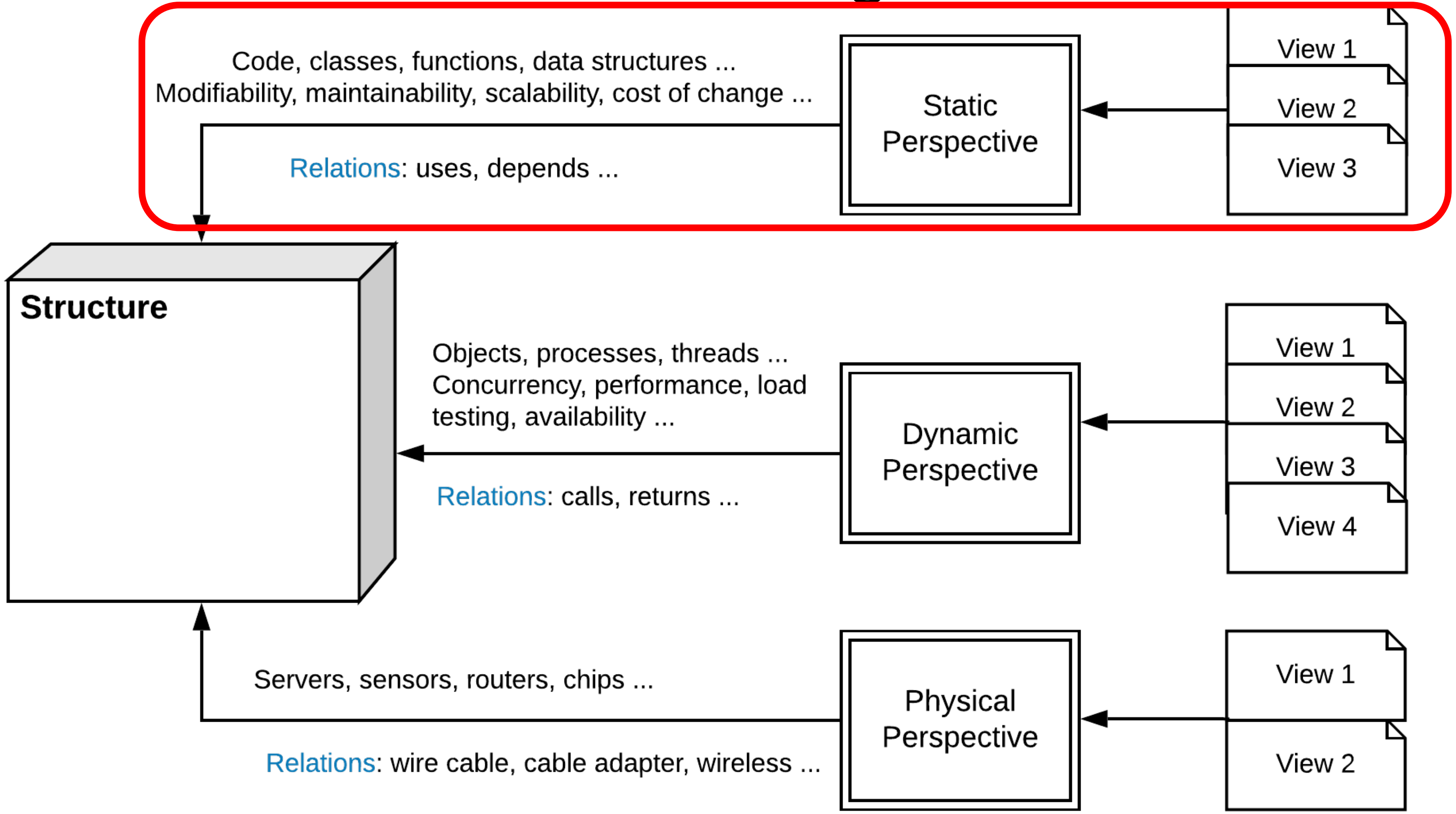
Analyze a car engine

Static

Dynamic

Physical

Common Views in Documenting Software Architecture

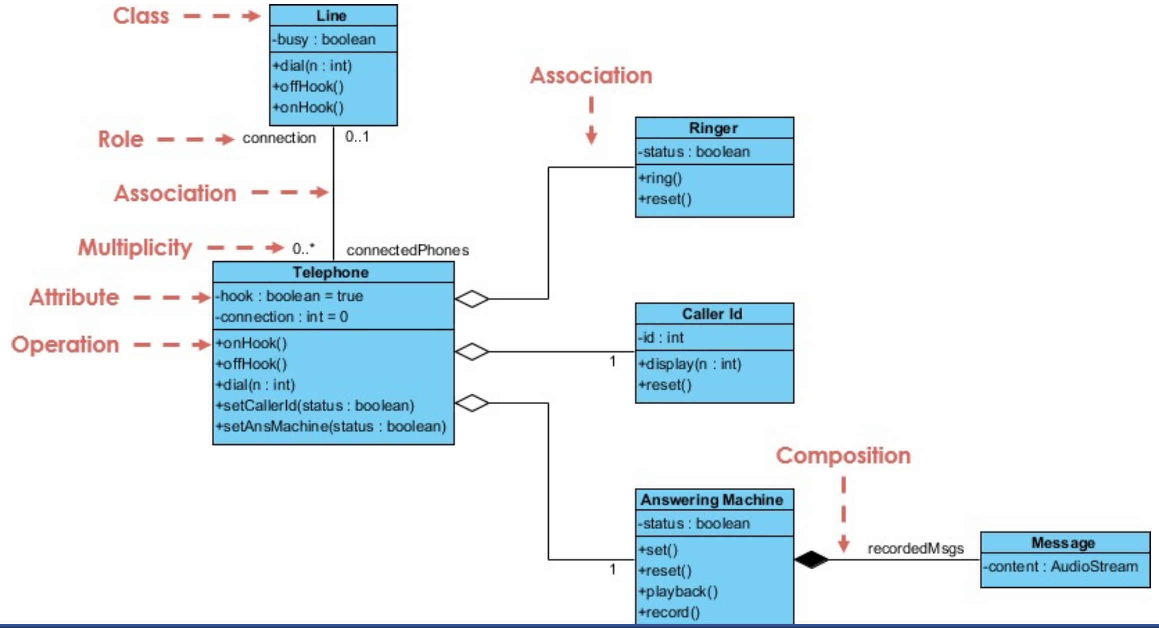

Code, classes, functions, data structures ...
Modifiability, maintainability, scalability, cost of change ...
Relations: uses, depends ...
Static Perspective
View 1
View 2
View 3
Structure
Objects, processes, threads ...
Concurrency, performance, load testing, availability ...
Relations: calls, returns ...
Dynamic Perspective
View 1
View 2
View 3
View 4
Servers, sensors, routers, chips ...
Relations: wire cable, cable adapter, wireless ...
Physical Perspective
View 1
View 2

https://medium.com/geekculture/introduction-to-software-architecture-part-1-3358ede31af9

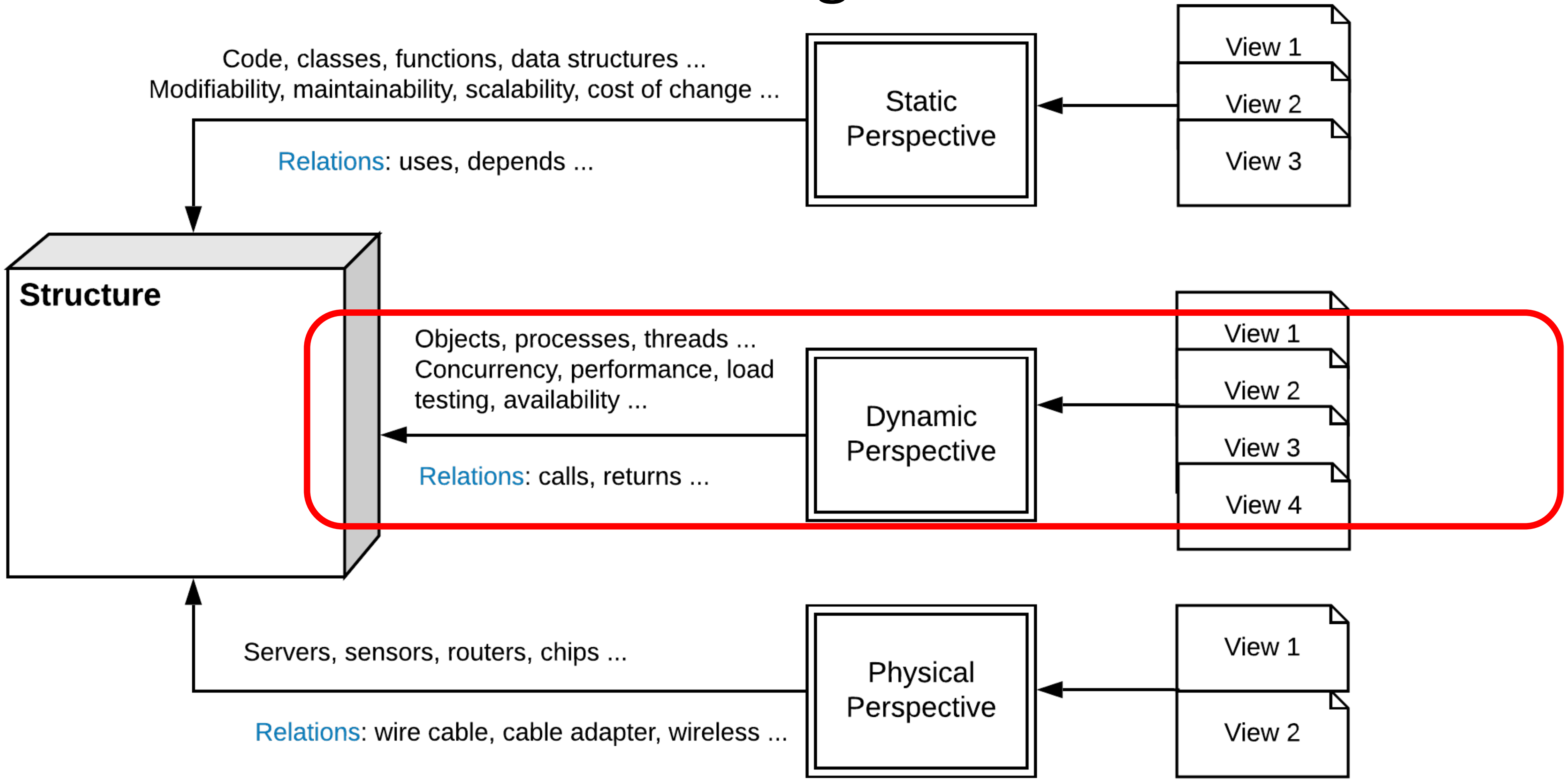
Common Views in Documenting Software Architecture

- **Modules** (Static)

Modules are assigned specific computational responsibilities, and are the basis of work assignments for programming teams



Common Views in Documenting Software Architecture


Code, classes, functions, data structures ...
Modifiability, maintainability, scalability, cost of change ...
Relations: uses, depends ...
Static Perspective
View 1
View 2
View 3
Structure
Objects, processes, threads ...
Concurrency, performance, load testing, availability ...
Relations: calls, returns ...
Dynamic Perspective
View 1
View 2
View 3
View 4
Servers, sensors, routers, chips ...
Relations: wire cable, cable adapter, wireless ...
Physical Perspective
View 1
View 2

https://medium.com/geekculture/introduction-to-software-architecture-part-1-3358ede31af9

Architecture Is a Set of Software Structures

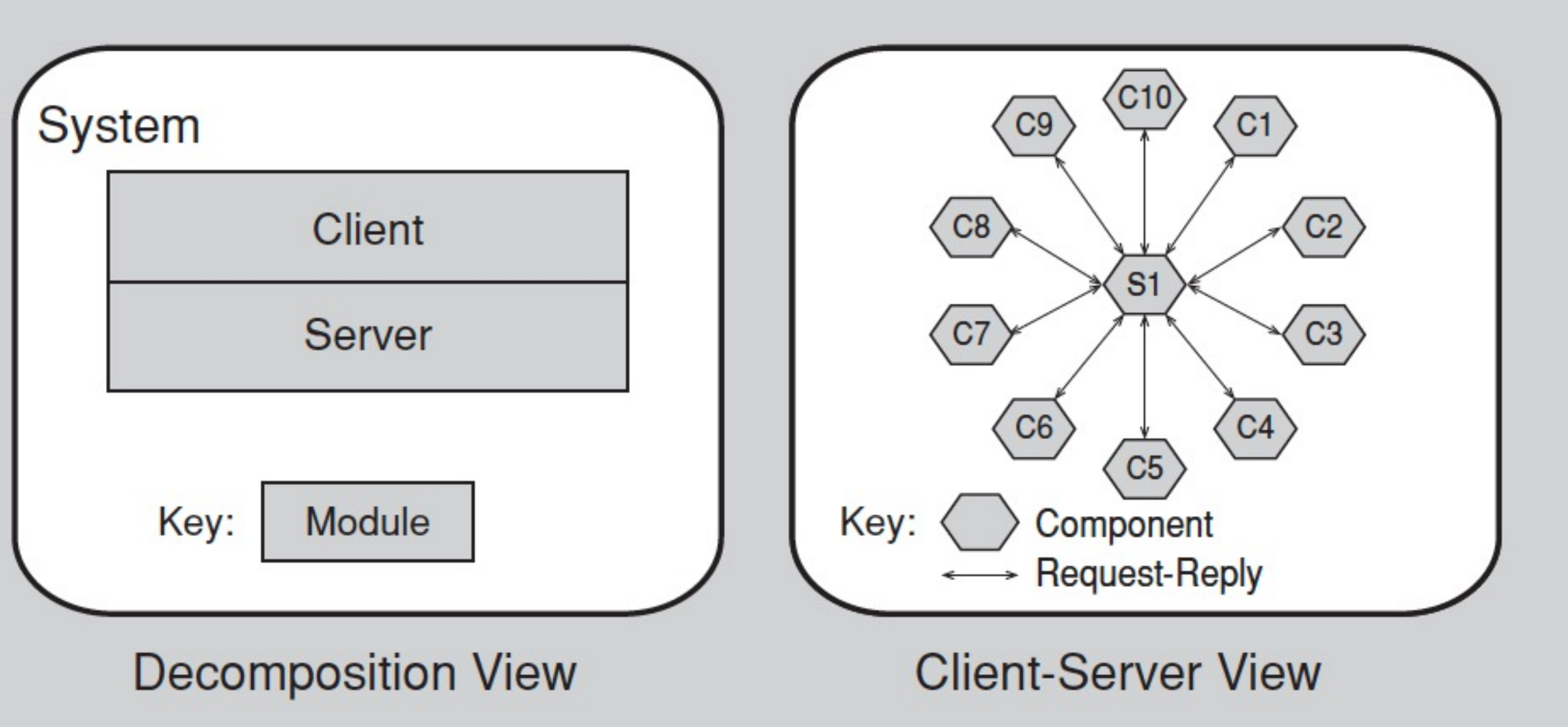
- **Modules** (Static)

Modules are assigned specific computational responsibilities, and are the basis of work assignments for programming teams

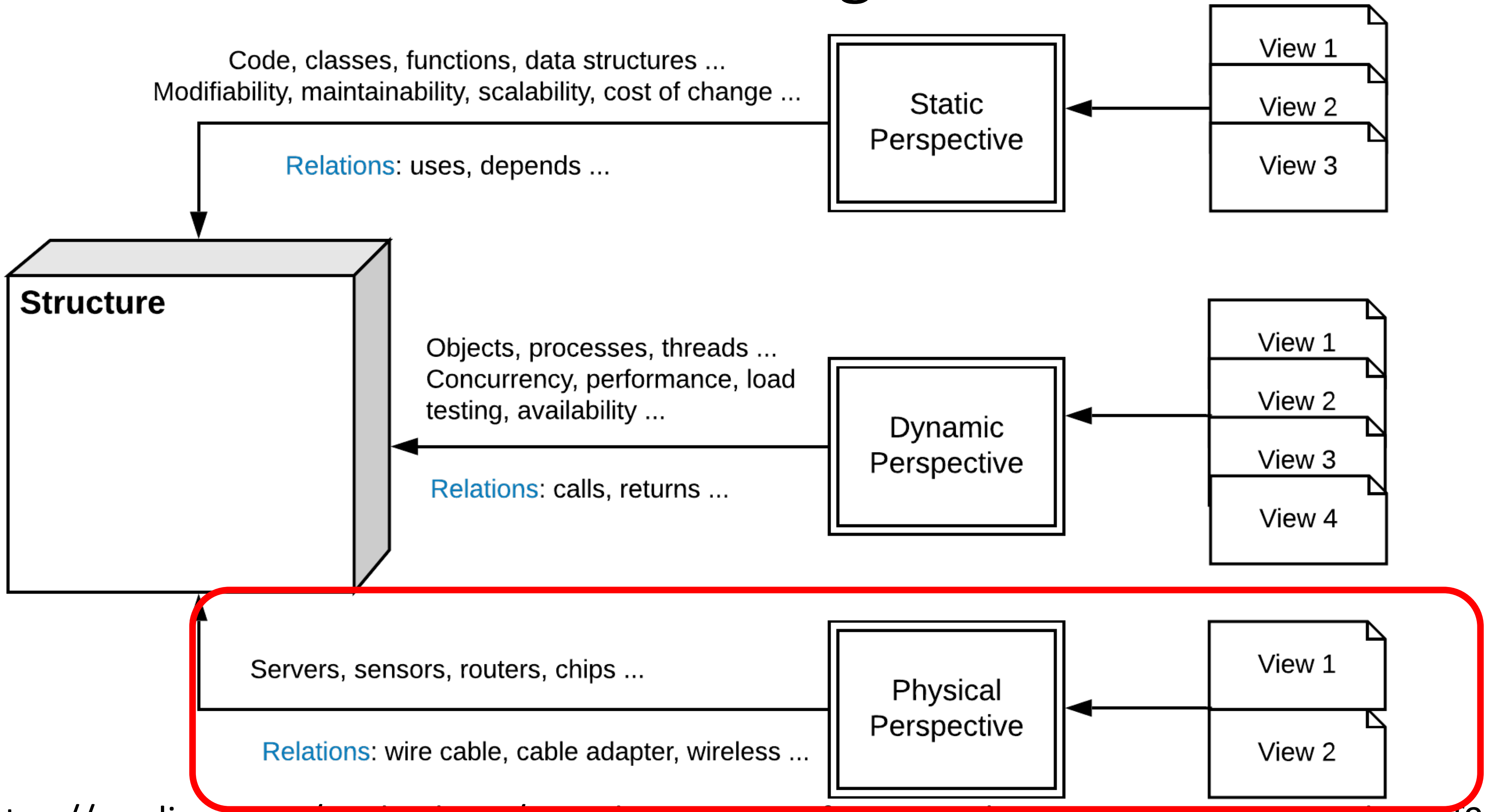
- **Dynamic** (Component-and-Connector **C&C**)

Focus on the way the elements interact with each other at runtime to carry out the system's functions.

Two views of a client-server system



Common Views in Documenting Software Architecture



Code, classes, functions, data structures ...
Modifiability, maintainability, scalability, cost of change ...

Relations: uses, depends ...

Static Perspective

View 1
View 2
View 3

Structure

Objects, processes, threads ...
Concurrency, performance, load testing, availability ...

Relations: calls, returns ...

Dynamic Perspective

View 1
View 2
View 3
View 4

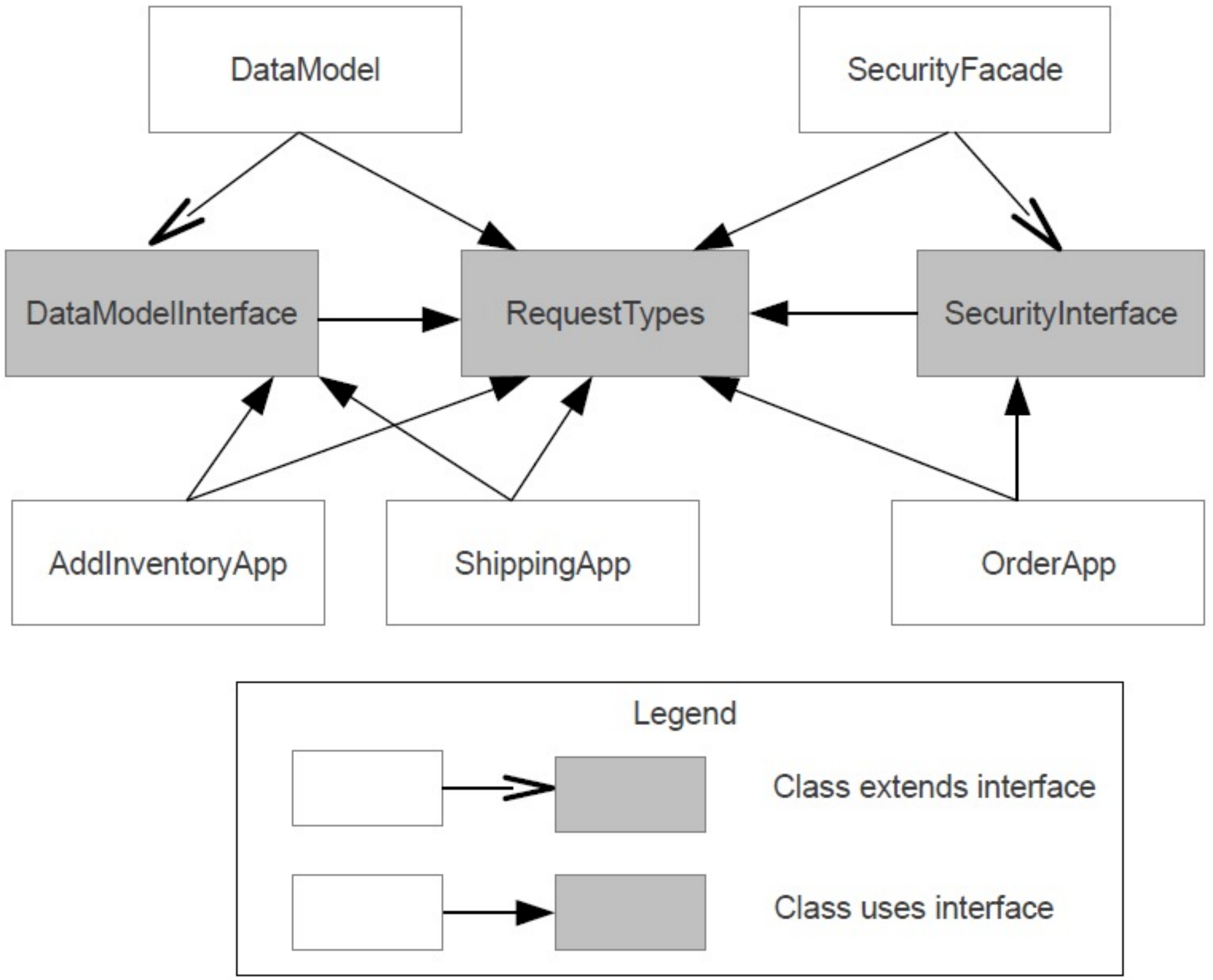
Servers, sensors, routers, chips ...

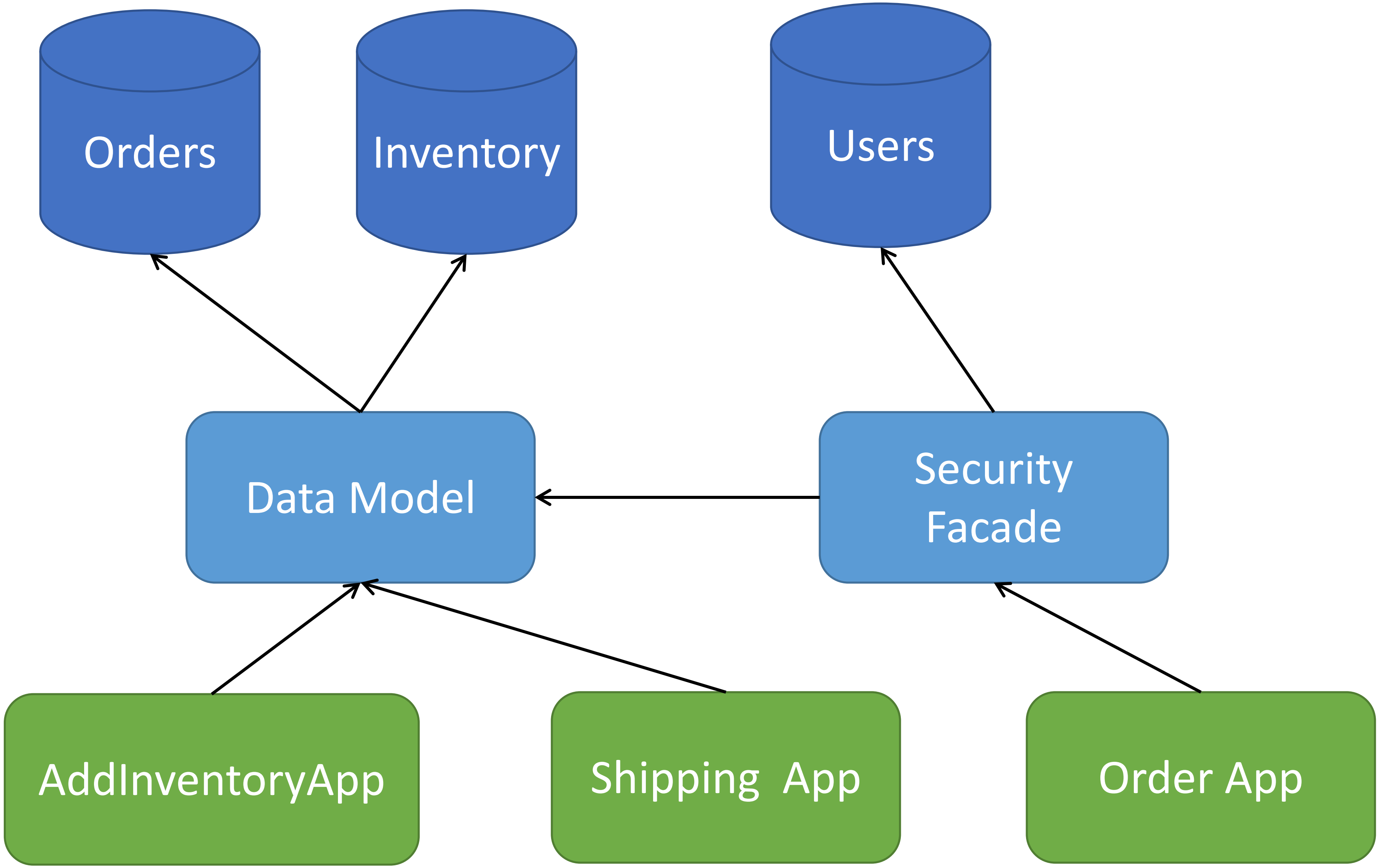
Relations: wire cable, cable adapter, wireless ...

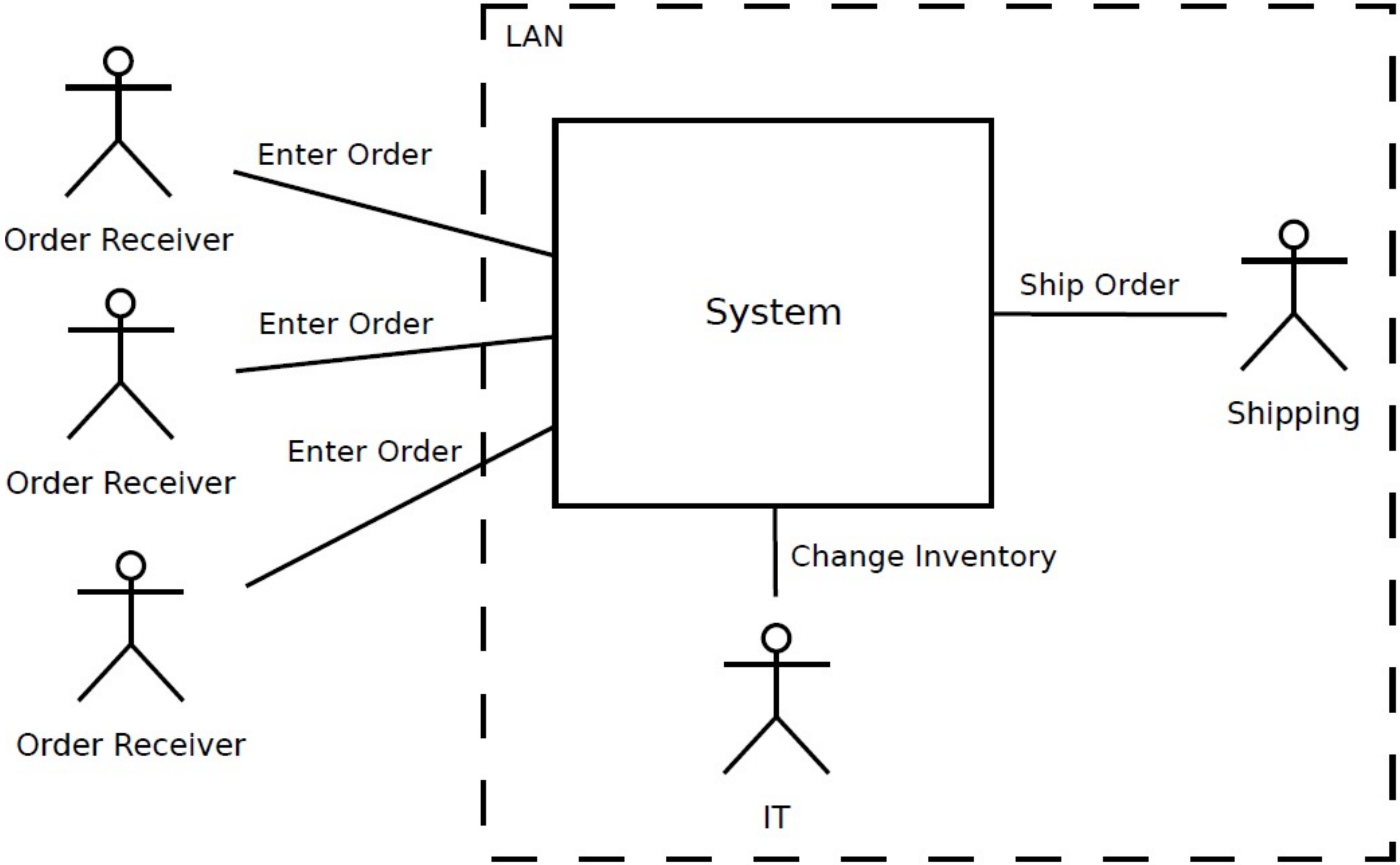
Physical Perspective

View 1
View 2

https://medium.com/geekculture/introduction-to-software-architecture-part-1-3358ede31af9


DataModel
SecurityFacade
DataModelInterface
RequestTypes
SecurityInterface
AddInventoryApp
ShippingApp
OrderApp
Legend
Class extends interface
Class uses interface


Orders
Inventory
Users
Data Model
Security Facade
AddInventoryApp
Shipping App
Order App


LAN
Enter Order
Order Receiver
Enter Order
Order Receiver
Enter Order
Order Receiver
System
Ship Order
Shipping
Change Inventory
IT

Selecting a Notation

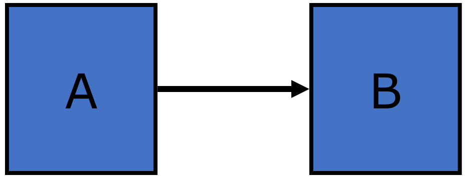
- Suitable for purpose
- Often visual for compact representation
- Usually boxes and arrows
- UML possible (semi-formal), but possibly constraining
    - Note the different abstraction level – Subsystems or processes, not classes or objects
- Formal notations available
- Decompose diagrams hierarchically and in views

Guidelines: Avoiding Ambiguity

- Always include a legend
- Define precisely what the boxes mean
- Define precisely what the lines mean
- Supplement graphics with explanation
    - Very important: rationale (architectural intent)
- Do not try to do too much in one diagram
    - Each view of architecture should fit on a page
    - Use hierarchy

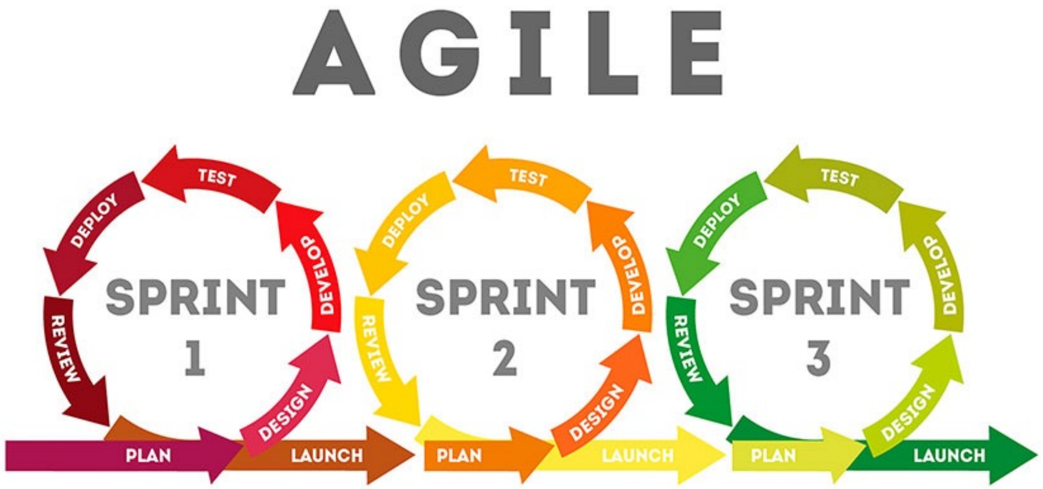
What could the arrow mean?

- Many possibilities
  - A passes control to B
  - A passes data to B
  - A gets a value from B
  - A streams data to B
  - A sends a message to B
  - A creates B
  - A occurs before B
  - B gets its electricity from A
  - …



Future Readings

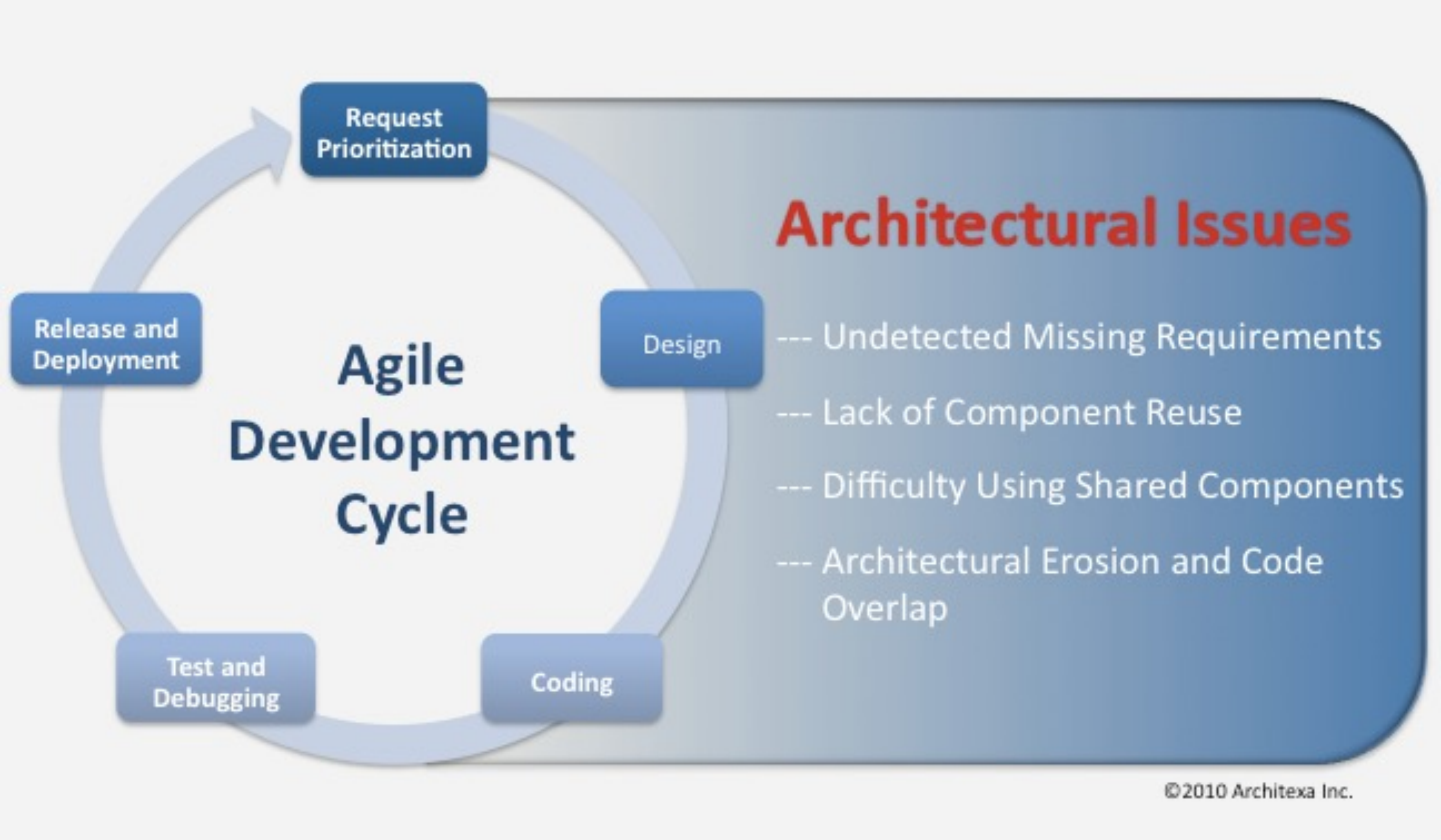
- Bass, Clements, and Kazman. Software Architecture in Practice. Addison-Wesley, 2003.
- Boehm and Turner. Balancing Agility and Discipline: A Guide for the Perplexed, 2003.
- Clements, Bachmann, Bass, Garlan, Ivers, Little, Merson, Nord, Stafford. Documenting Software Architectures: Views and Beyond, 2010.
- Fairbanks. Just Enough Software Architecture. Marshall & Brainerd, 2010.
- Jansen and Bosch. Software Architecture as a Set of Architectural Design Decisions, WICSA 2005.
- Lattanze. Architecting Software Intensive Systems: a Practitioner's Guide, 2009.
- Sommerville. Software Engineering. Edition 7/8, Chapters 11-13
- Taylor, Medvidovic, and Dashofy. Software Architecture: Foundations, Theory, and Practice. Wiley, 2009.



Architecture in Agile Project

Architecture in Agile Project

- "How much architecture should I do up front versus how much should I defer until the project's requirements have solidified somewhat?",
- "When and how should I refactor?"
- "How much of the architecture should I formally document, and when?"



Architectural Issues

--- Undetected Missing Requirements

--- Lack of Component Reuse

--- Difficulty Using Shared Components

--- Architectural Erosion and Code Overlap

©2010 Architexa Inc.

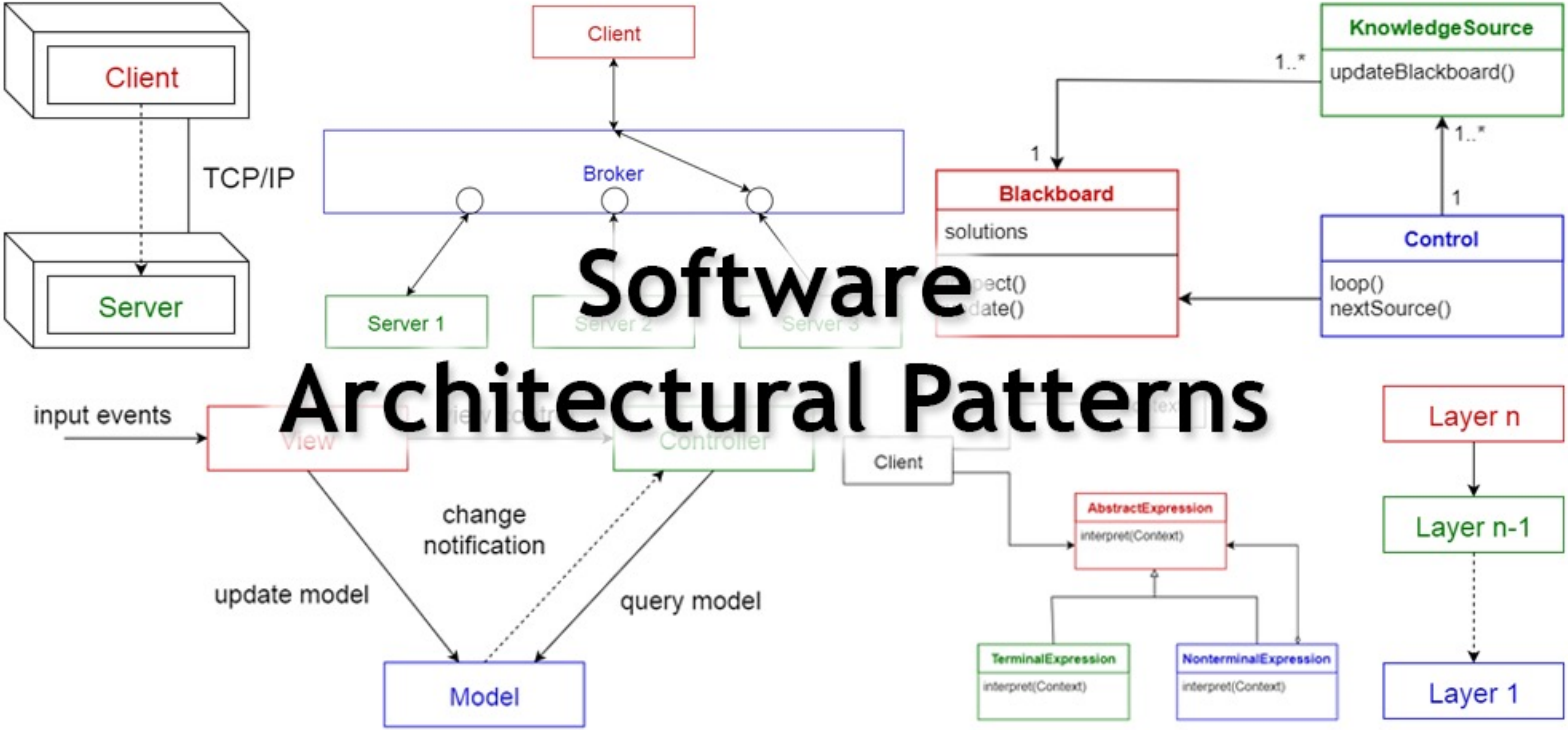
https://www.architexa.com/learn-more/architecture

Architecture Design in Agile

- Understanding code architecture and preventing boundaries erosion
- Maintaining well-defined architecture and module boundaries
- Having a consistent architecture shared with the entire team

Overview! Diagram! Mitigate Technical Debt!

Software Architectural Patterns



Architectural Patterns

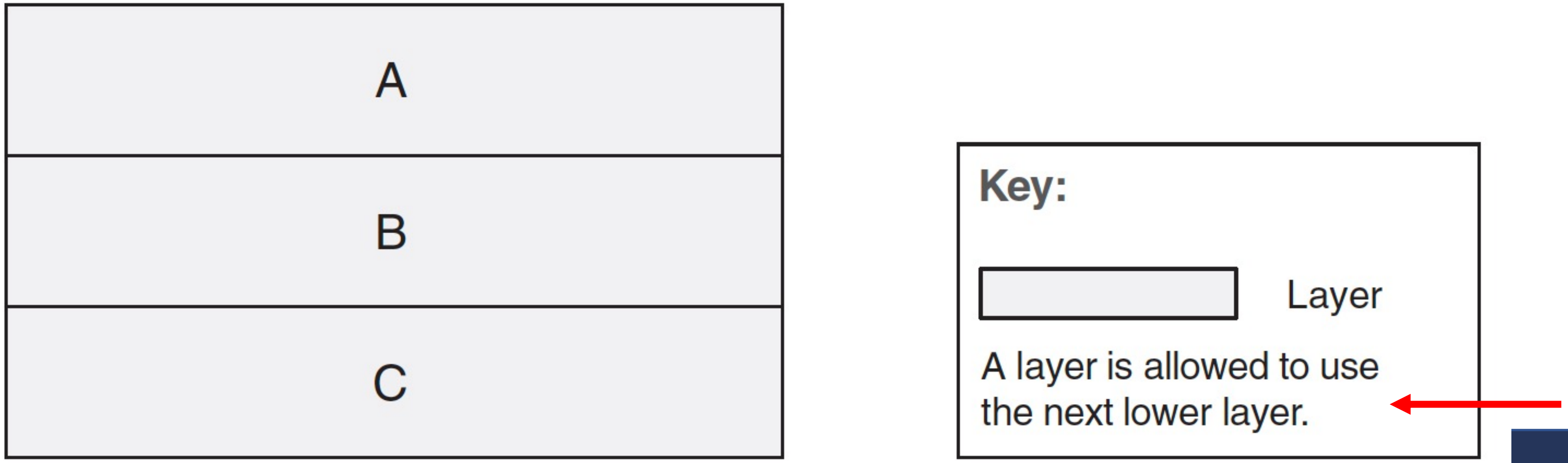
- Context + Problem + Solution
- Related to one of common view types
    - Static, dynamic, physical

Example Architectural Patterns

- **Modules (Static)**
  - Layered Pattern
- **Dynamic** (Component-and-connector **C&C**)
- **Allocation** (Physical, Deployment)

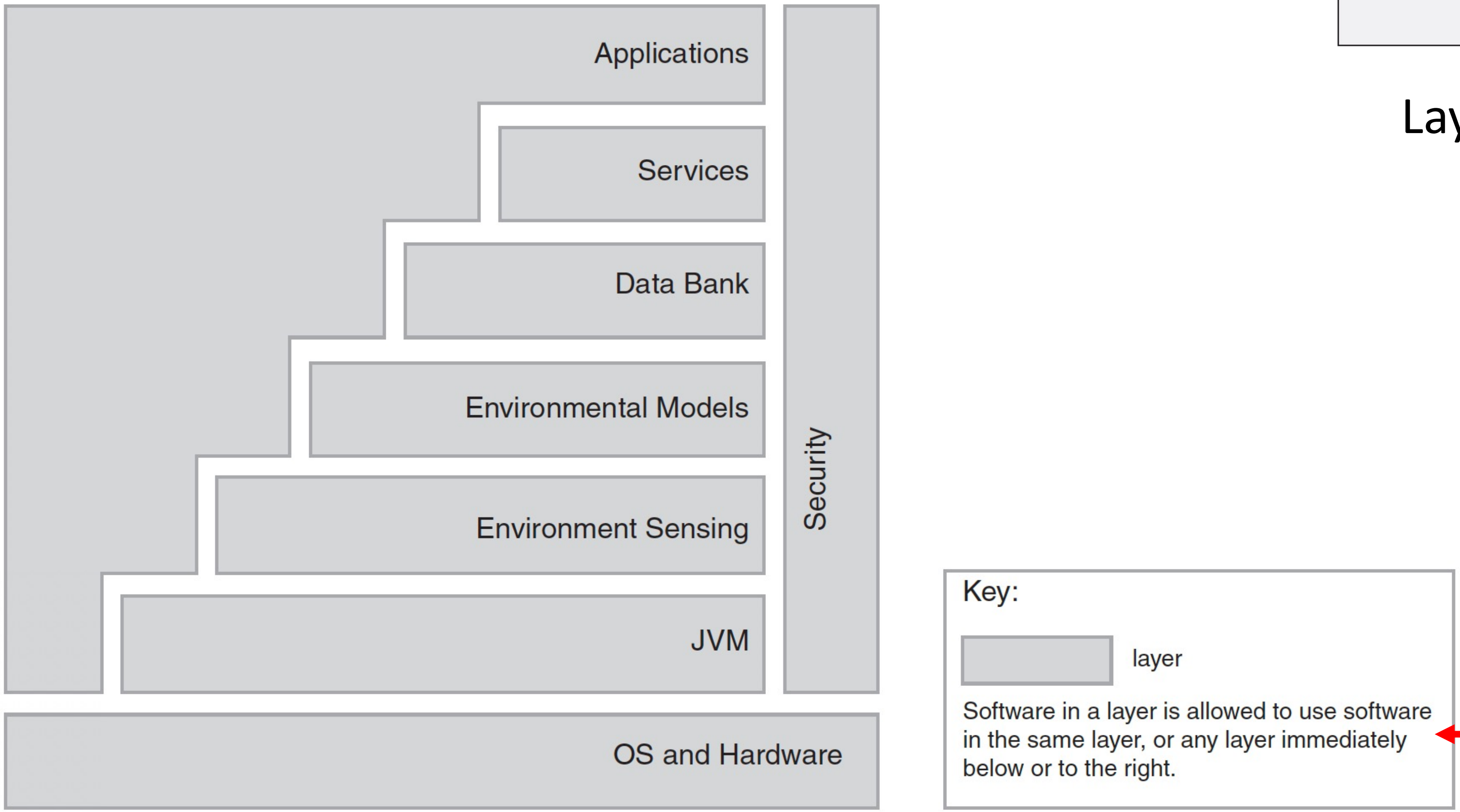
Layered Pattern

- Separation of concerns
- Constraints on the allowed-to-use relationship among the layers, the relations must be unidirectional
- Normally only next-lower-layer uses are allowed
- “above” and “below” matter

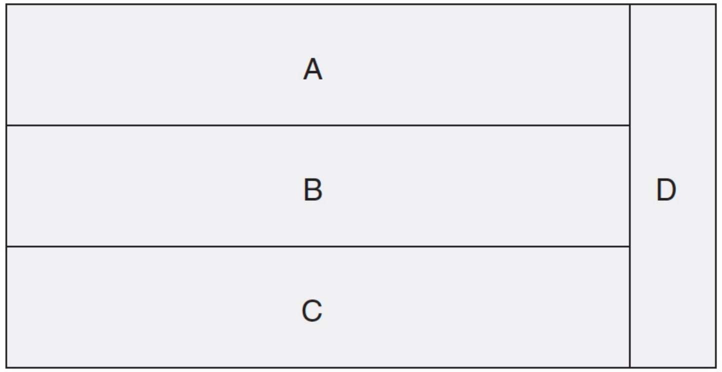


A layer is allowed to use the next lower layer.

Layered Pattern



Software in a layer is allowed to use software in the same layer, or any layer immediately below or to the right.



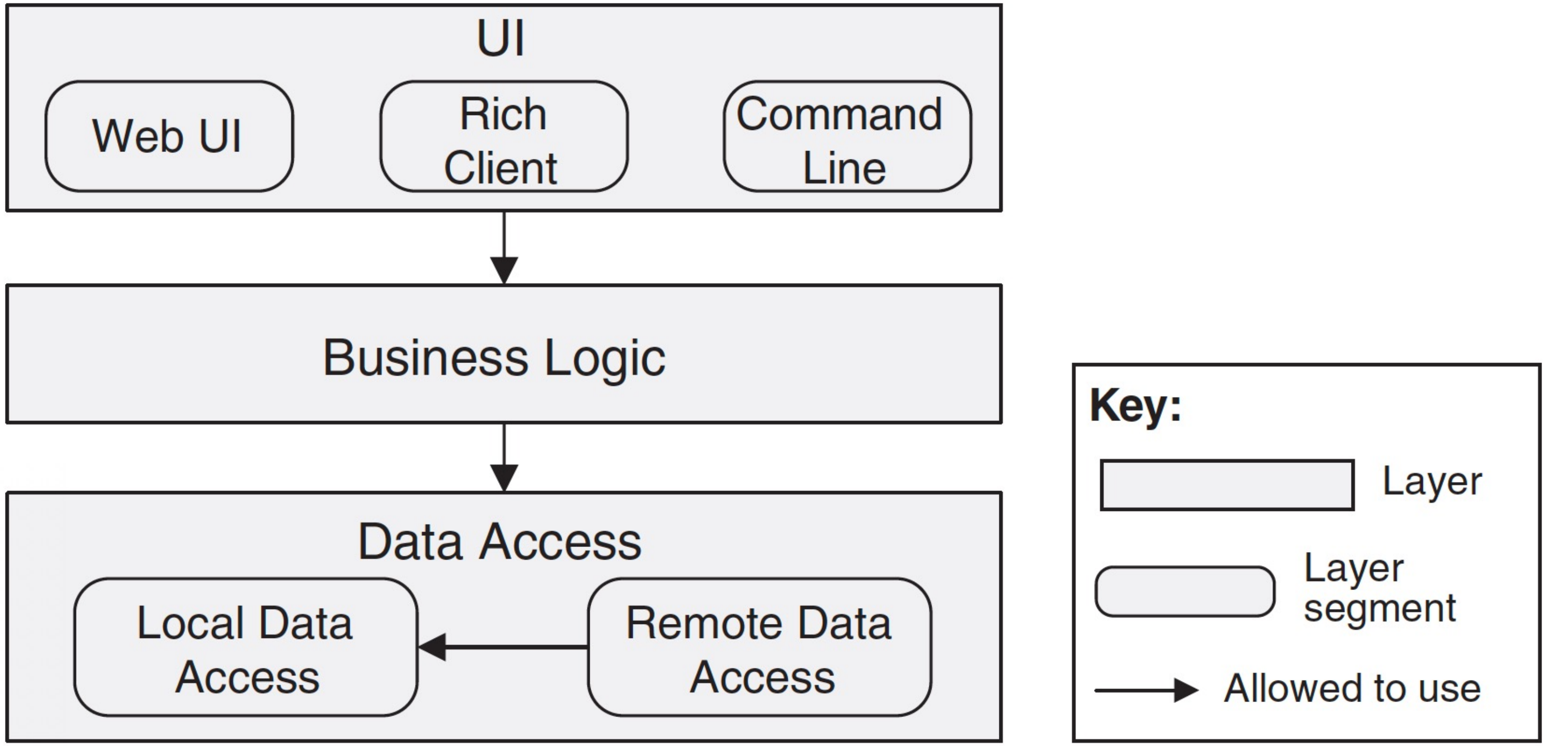
Layers with a “sidebar”

Layered Pattern

Usage:

General desktop applications.

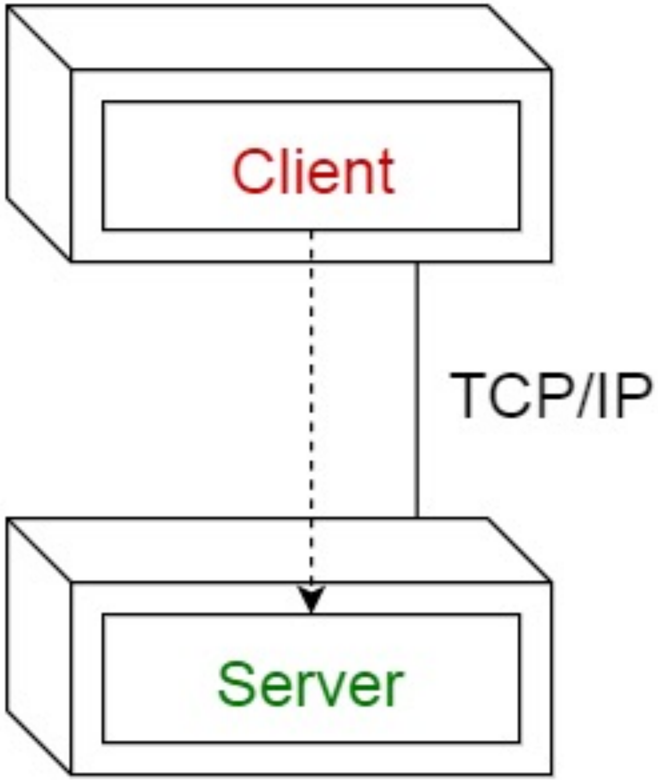
E commerce web applications.



Layered design with segmented layers

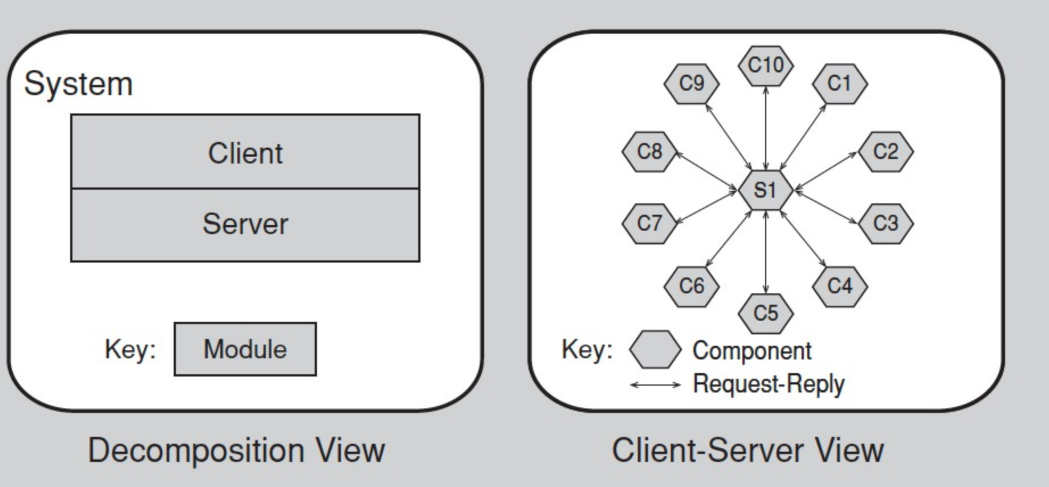
Example Architectural Patterns

- **Modules (Static)**
  - Layered Pattern
- **Dynamic** (Component-and-connector **C&C**)
  - Client-Server Pattern
  - MVC (Model-View-Controller) Pattern
- **Allocation** (Physical, Deployment)



Client-Server Pattern

- Context:
  - shared resources and services
  - large numbers of distributed clients wish to access,
  - control access or quality of service.
- Modifiability, Reuse, Scalability, Availability
- Asymmetric or Synchronous



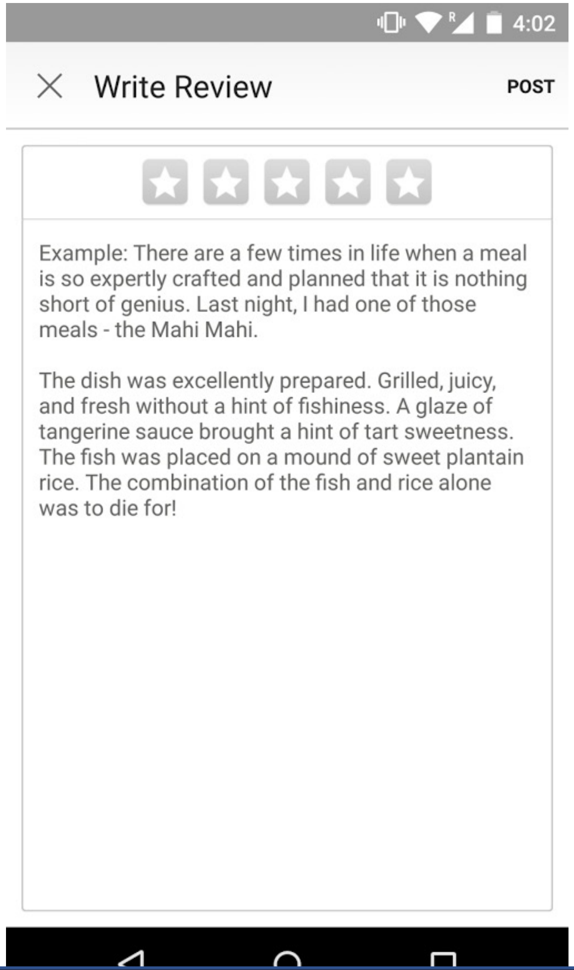
Client-Server Pattern

Disadvantages:

- the server can be a performance bottleneck and it can be a single point of failure
- decisions about where to locate functionality (in the client or in the server) are often complex and costly to change after a system has been built.

Where to validate user input?

Example: Yelp App

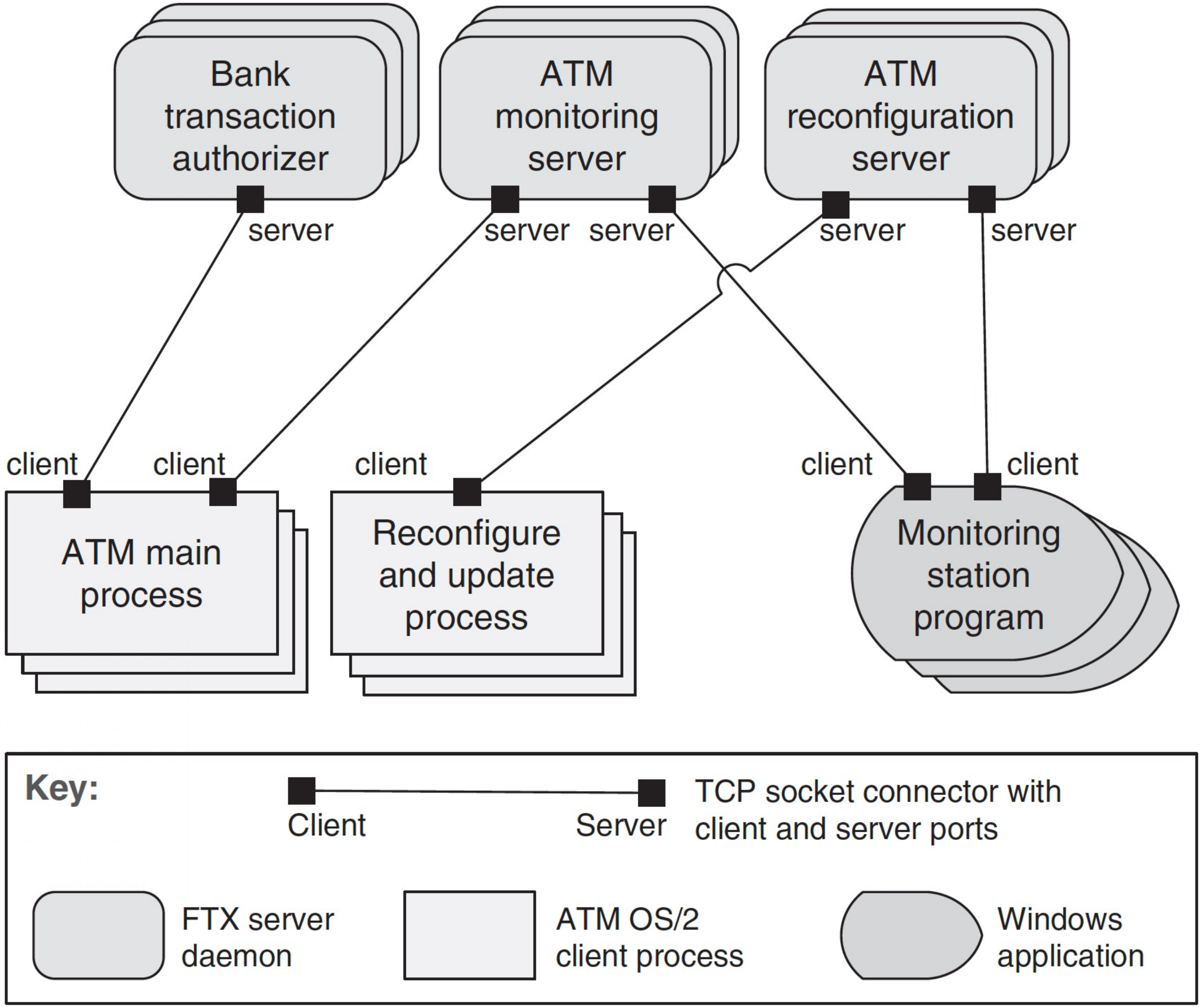


Example: There are a few times in life when a meal is so expertly crafted and planned that it is nothing short of genius. Last night, I had one of those meals - the Mahi Mahi.

The dish was excellently prepared. Grilled, juicy, and fresh without a hint of fishiness. A glaze of tangerine sauce brought a hint of tart sweetness. The fish was placed on a mound of sweet plantain rice. The combination of the fish and rice alone was to die for!

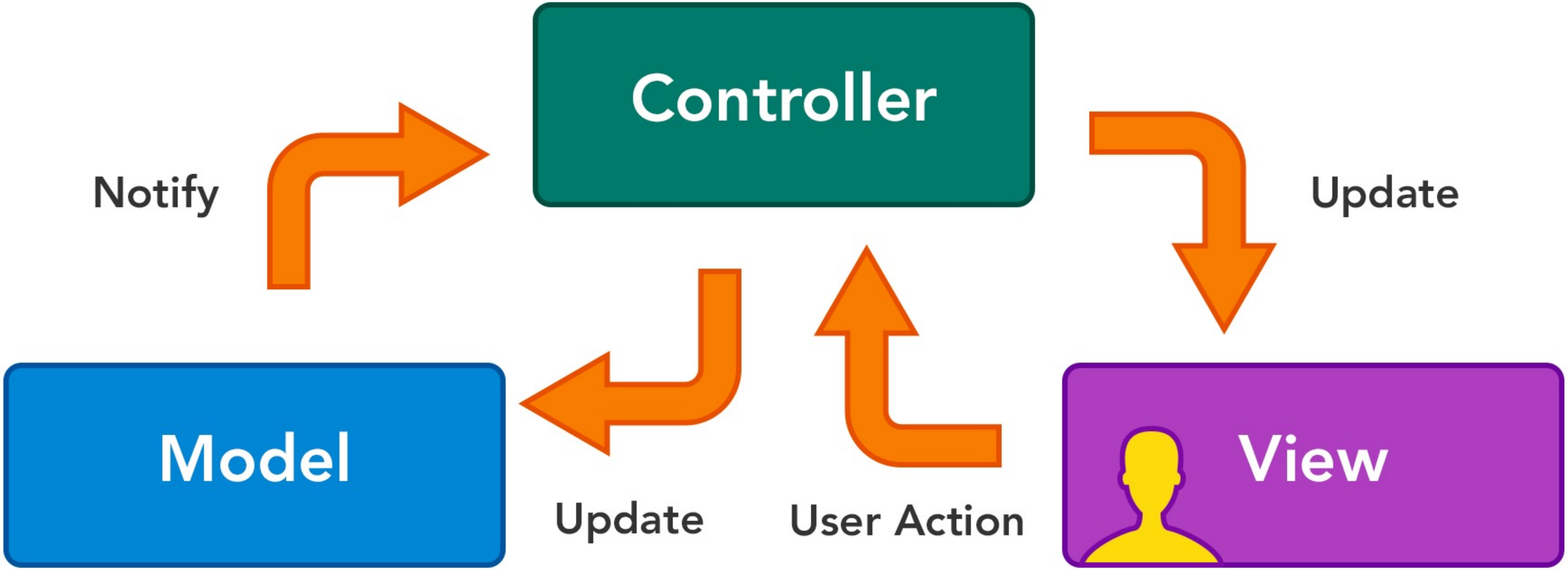
Real-world Example

Online applications (email, document sharing and banking)



Example Architectural Patterns

- **Modules (Static)**
  - Layered Pattern
- **Dynamic** (Component-and-connector **C&C**)
  - Client-Server Pattern
  - MVC (Model-View-Controller) Pattern



Controller

Notify

Update

Model

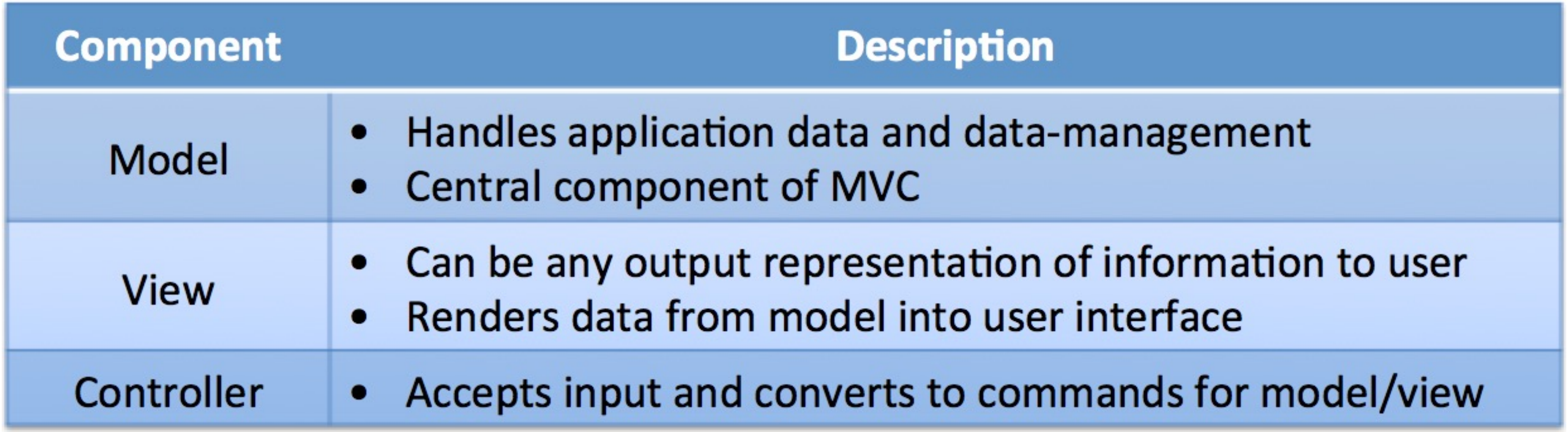
Update

User Action

View

MVC (Model-View-Controller) Pattern

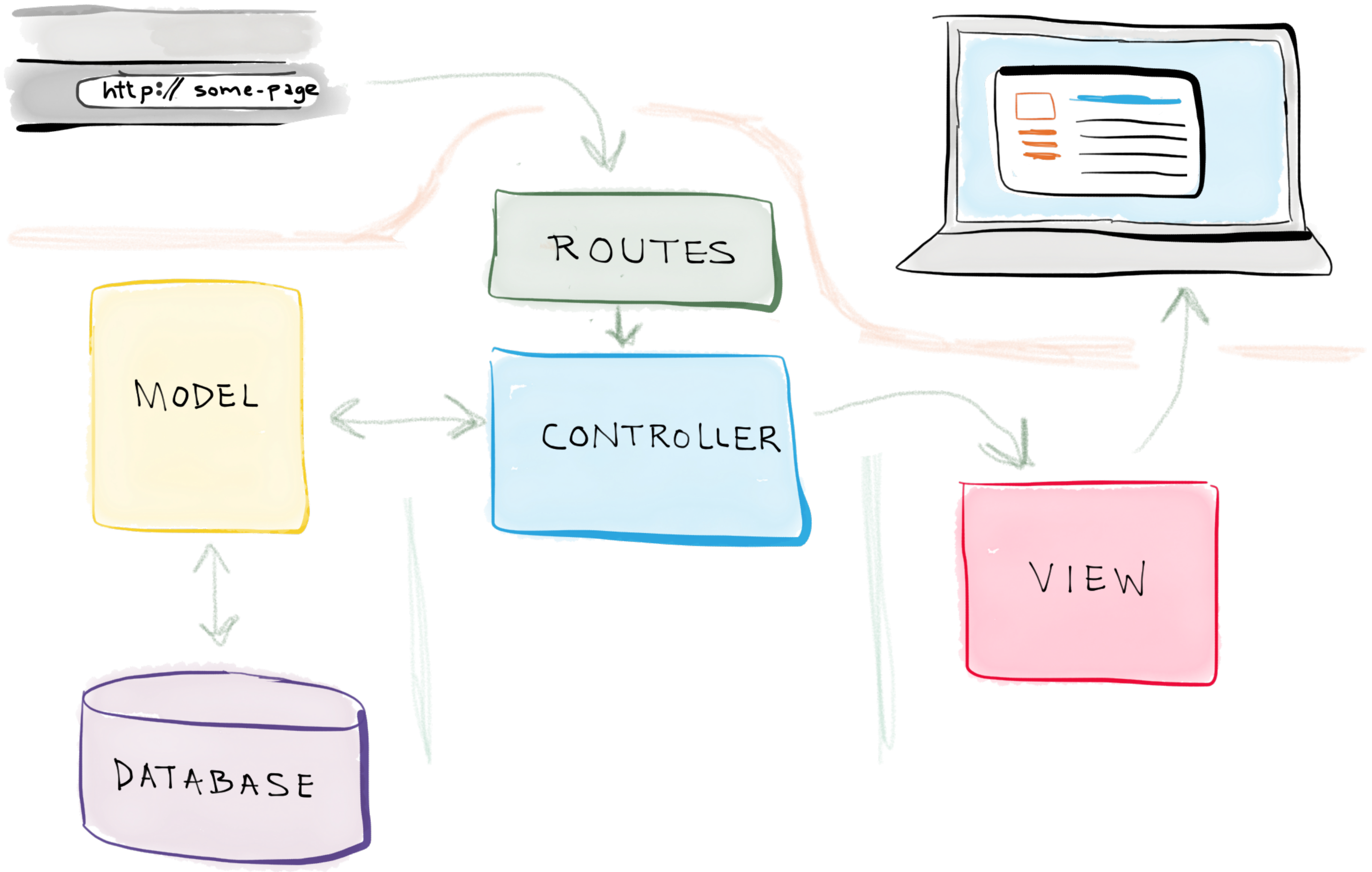
- Separate UI functionality from the application functionality
- Multiple views of the user interface can be created, maintained, and coordinated when the underlying application data changes



Component	Description
Model	• Handles application data and data-management • Central component of MVC
View	• Can be any output representation of information to user • Renders data from model into user interface
Controller	• Accepts input and converts to commands for model/view

https://realpython.com/the-model-view-controller-mvc-paradigm-summarized-with-legos/

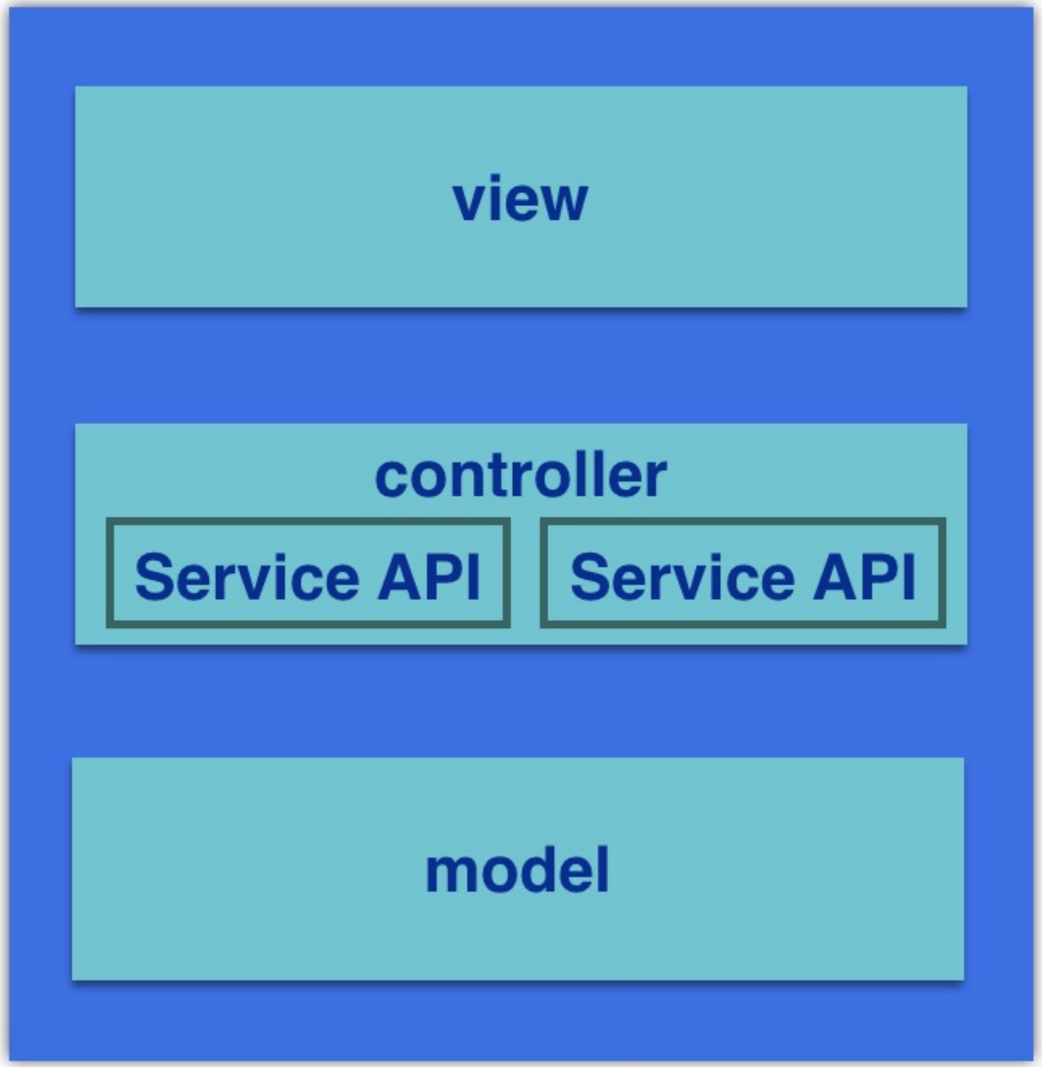
MVC and the Web



MVC (Model-View-Controller) Pattern

- Weaknesses: The complexity may not be worth it for simple user interfaces.

Monolithic



view

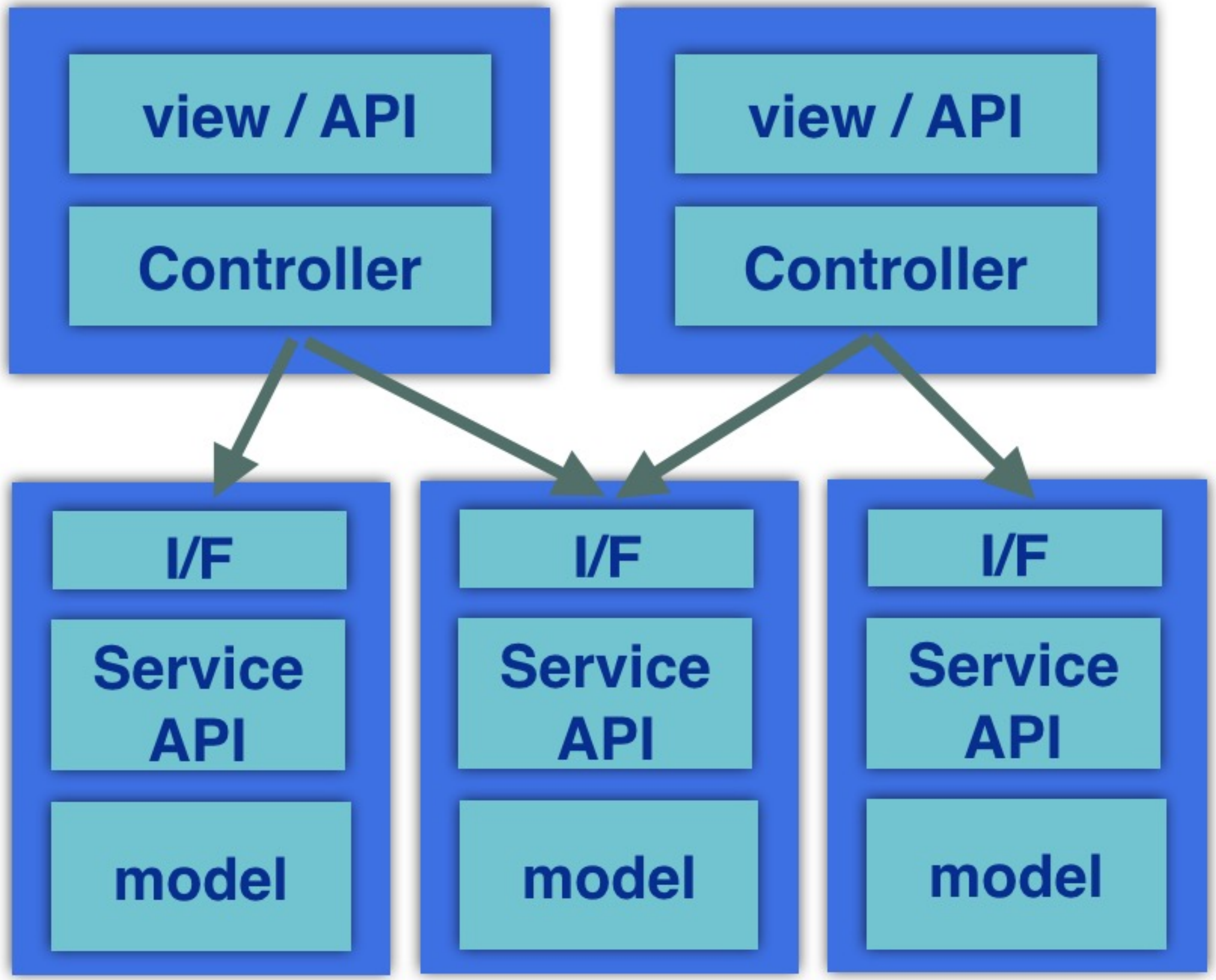
controller

Service API

Service API

model

MicroServices



view / API

Controller

view / API

Controller

I/F

Service API

model

I/F

Service API

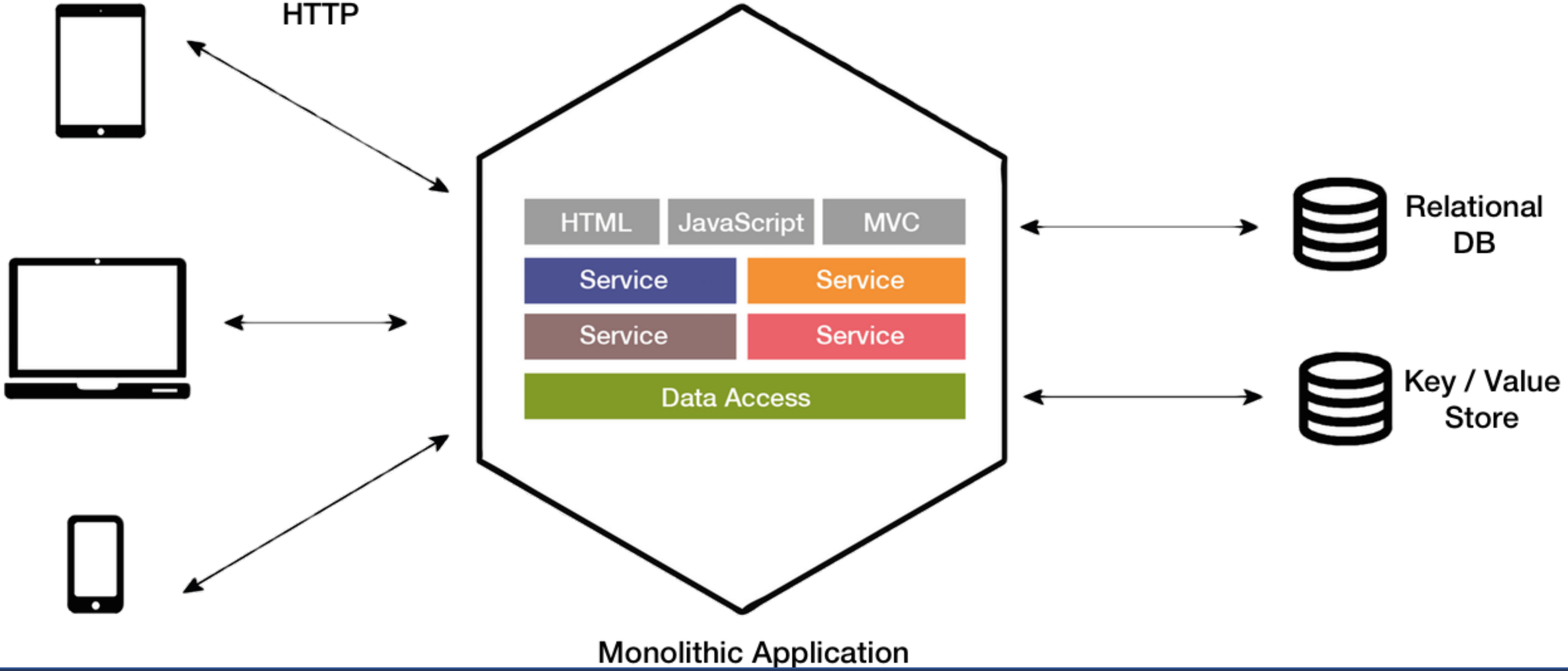
model

I/F

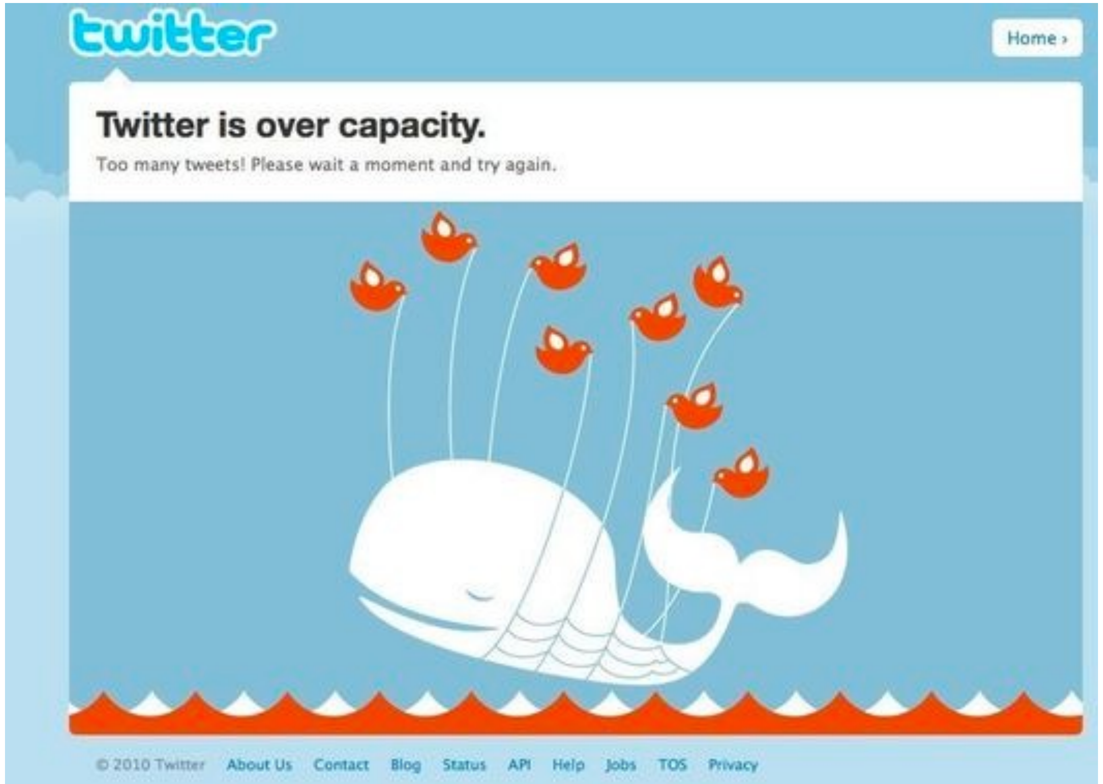
Service API

model

Monolithic Architecture



© 2005 FIFA TM

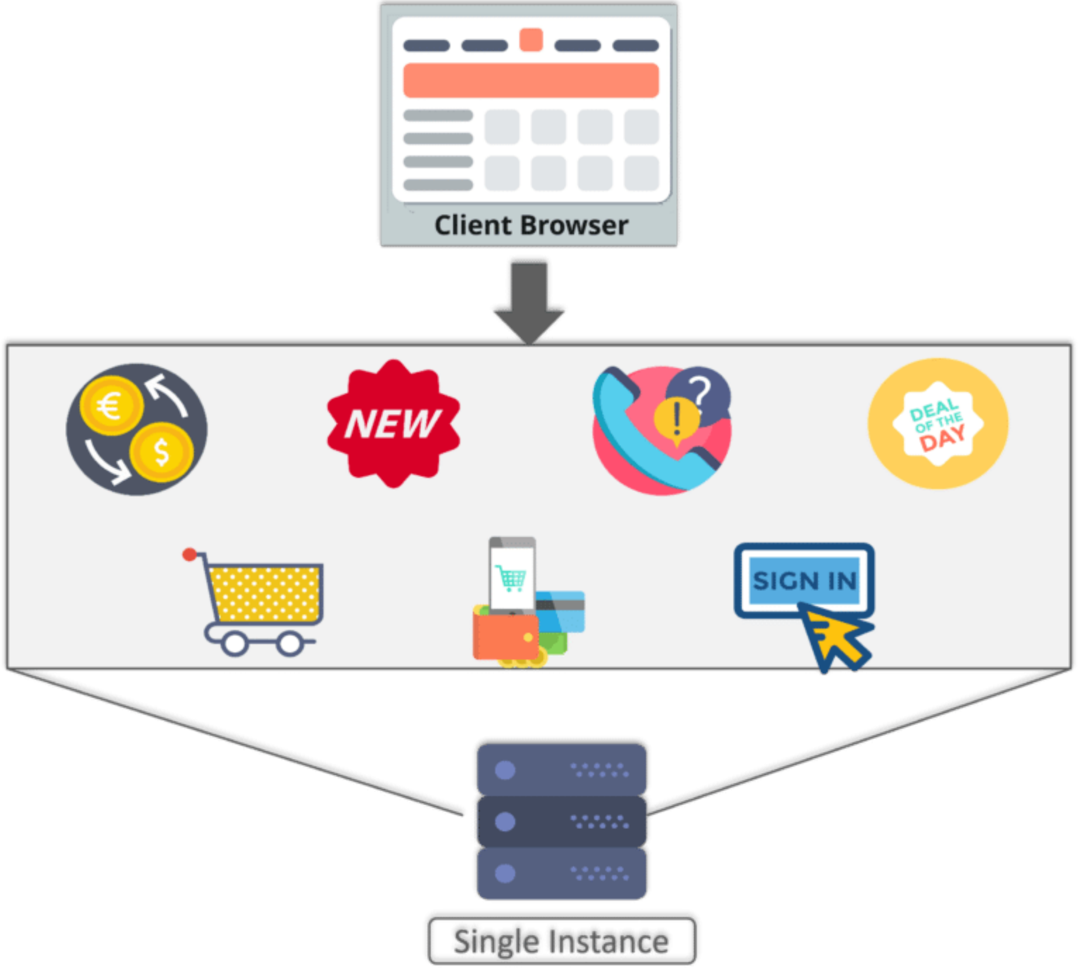


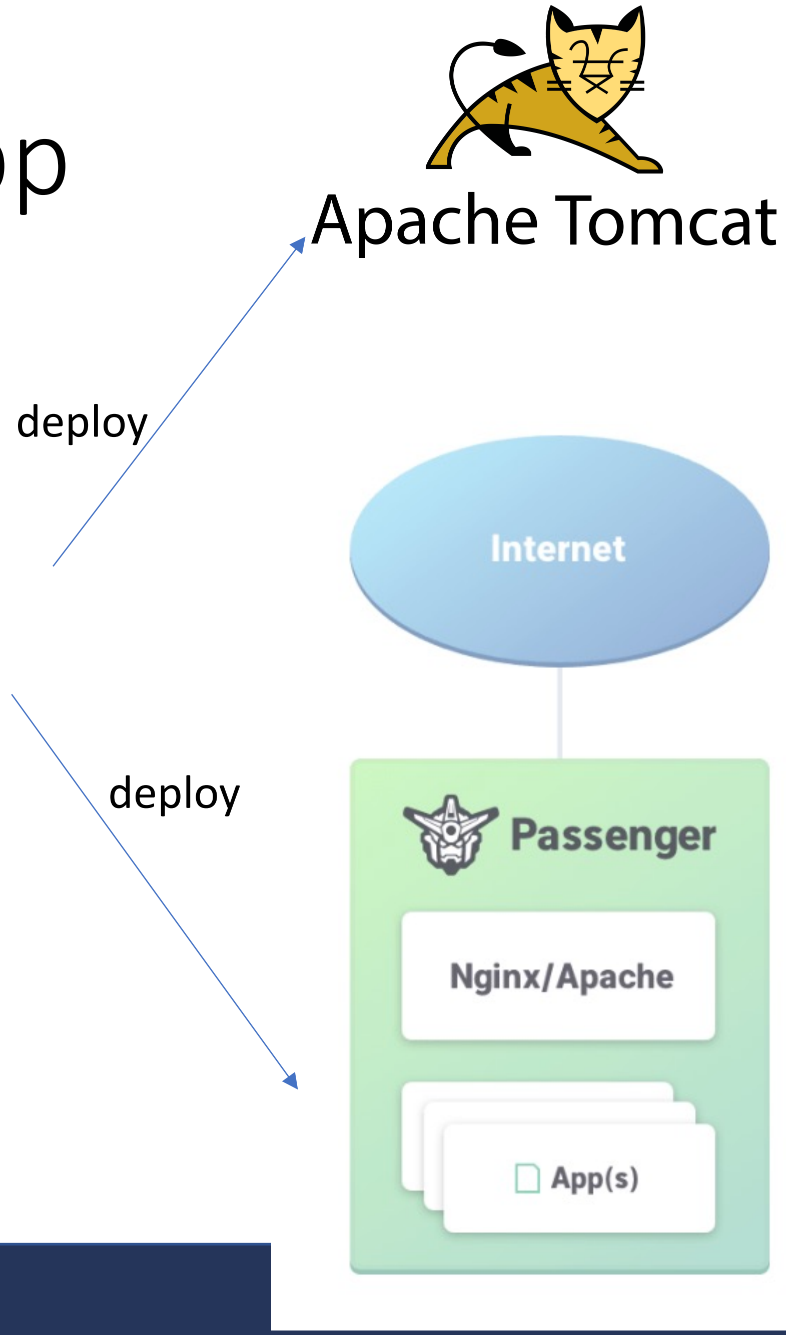
Twitter is over capacity.

Too many tweets! Please wait a moment and try again.

"After that experience, we determined we **needed to step back**. We then determined we needed to **re-architect** the site to support the continued growth of Twitter and to keep it running smoothly."

Example: a shopping cart app




Apache Tomcat

deploy

deploy

Internet

Passenger

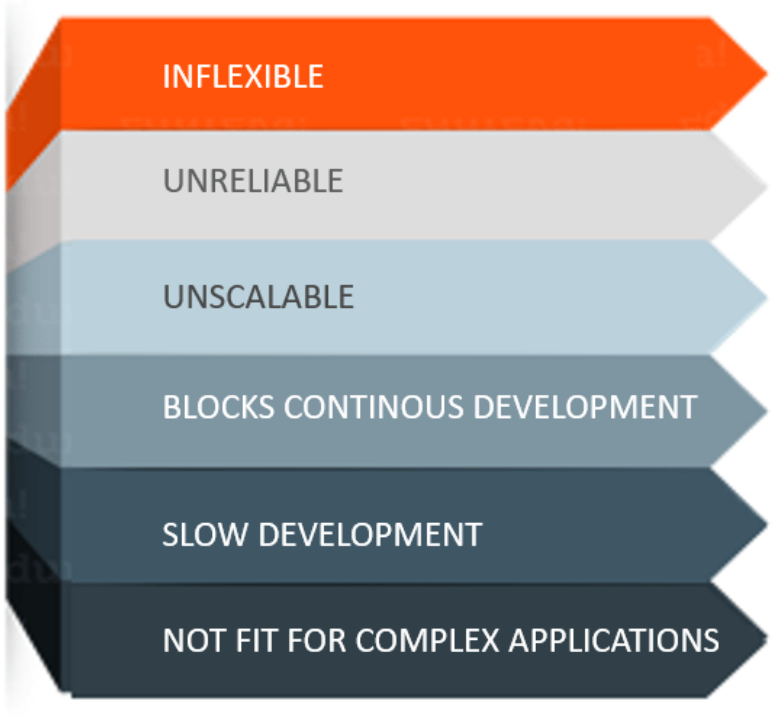
Nginx/Apache

App(s)

Monolithic Architecture Benefits

- Simple to develop
- Simple to deploy
- Simple to scale

Challenges of Monolithic Architecture


INFLEXIBLE
UNRELIABLE
UNSCALABLE
BLOCKS CONTINOUS DEVELOPMENT
SLOW DEVELOPMENT
NOT FIT FOR COMPLEX APPLICATIONS

- ***Inflexible*** *— Monolithic applications cannot be built using different technologies*
- ***Unreliable*** *— Even if one feature of the system does not work, then the entire system does not work*
- ***Unscalable*** *— Applications cannot be scaled easily since each time the application needs to be updated, the complete system has to be rebuilt*
- ***Blocks Continuous Development*** *— Many features of the applications cannot be built and deployed at the same time*
- ***Slow Development*** *— Development in monolithic applications take lot of time to be built since each and every feature has to be built one after the other*
- ***Not Fit For Complex Applications —*** *Features of complex applications have tightly coupled dependencies*

Microservices

Microservices
Microservices E
WHAT ARE MICROSERVICES?
OR ANTS???!
makeameme.org
MICROSERVICES...
...SO HOT RIGHT NOW
memegenerator.net

https://microservices.io/articles/whoisusingmicroservices.html

shopify

COMCAST

NETFLIX

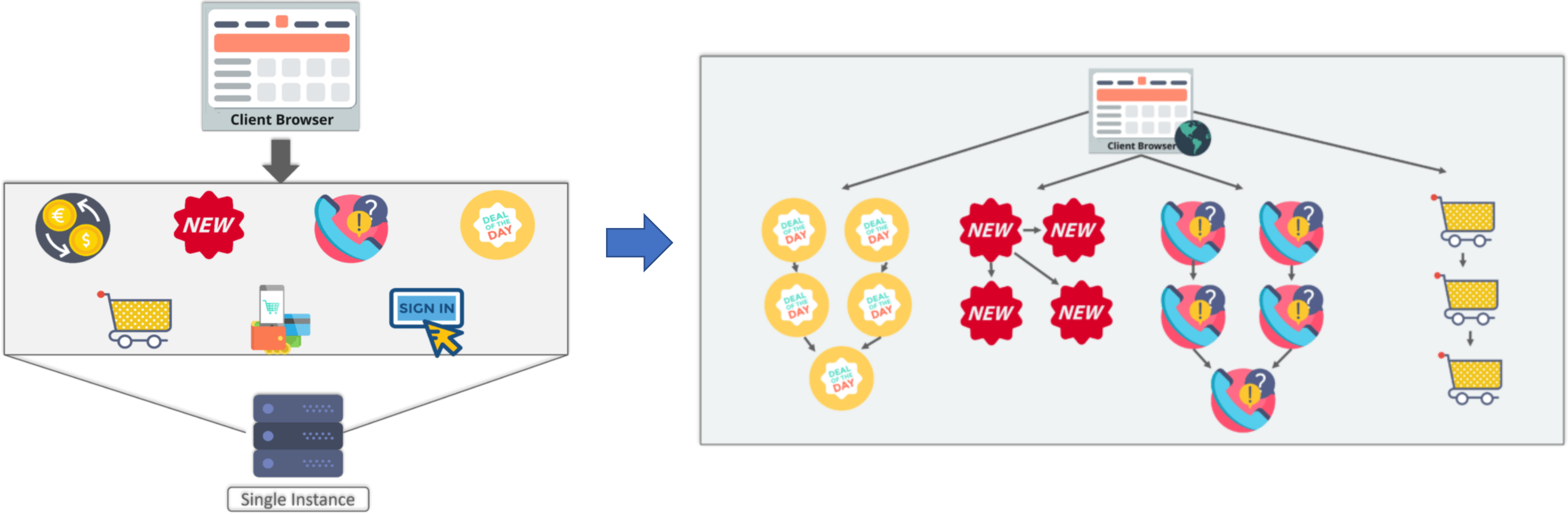
ebay

a

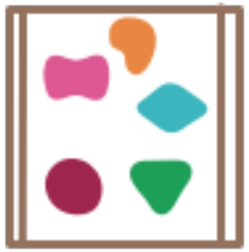
UBER

GROUPON®

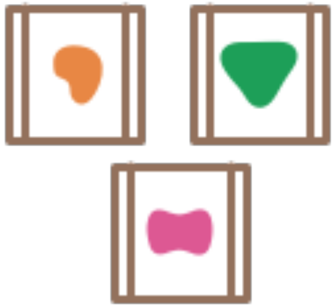
Use case: Shopping Cart Application



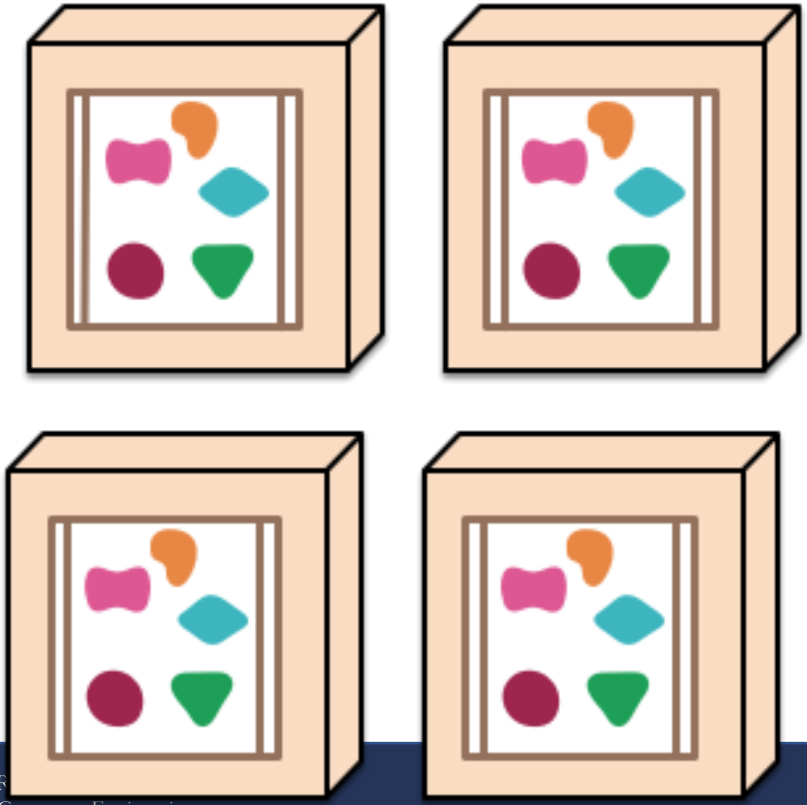
A monolithic application puts all its functionality into a single process...



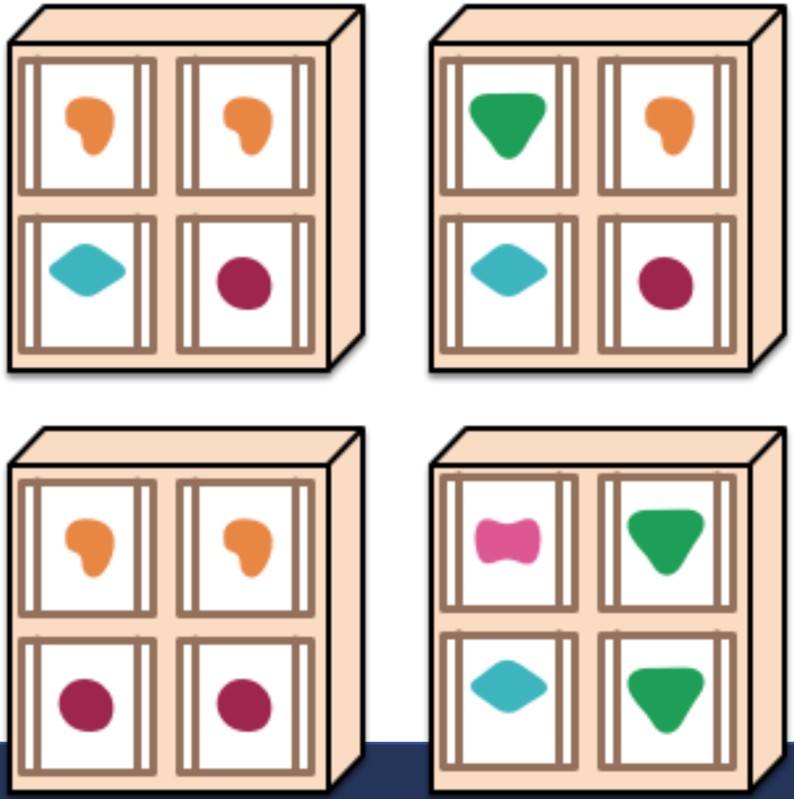
A microservices architecture puts each element of functionality into a separate service...

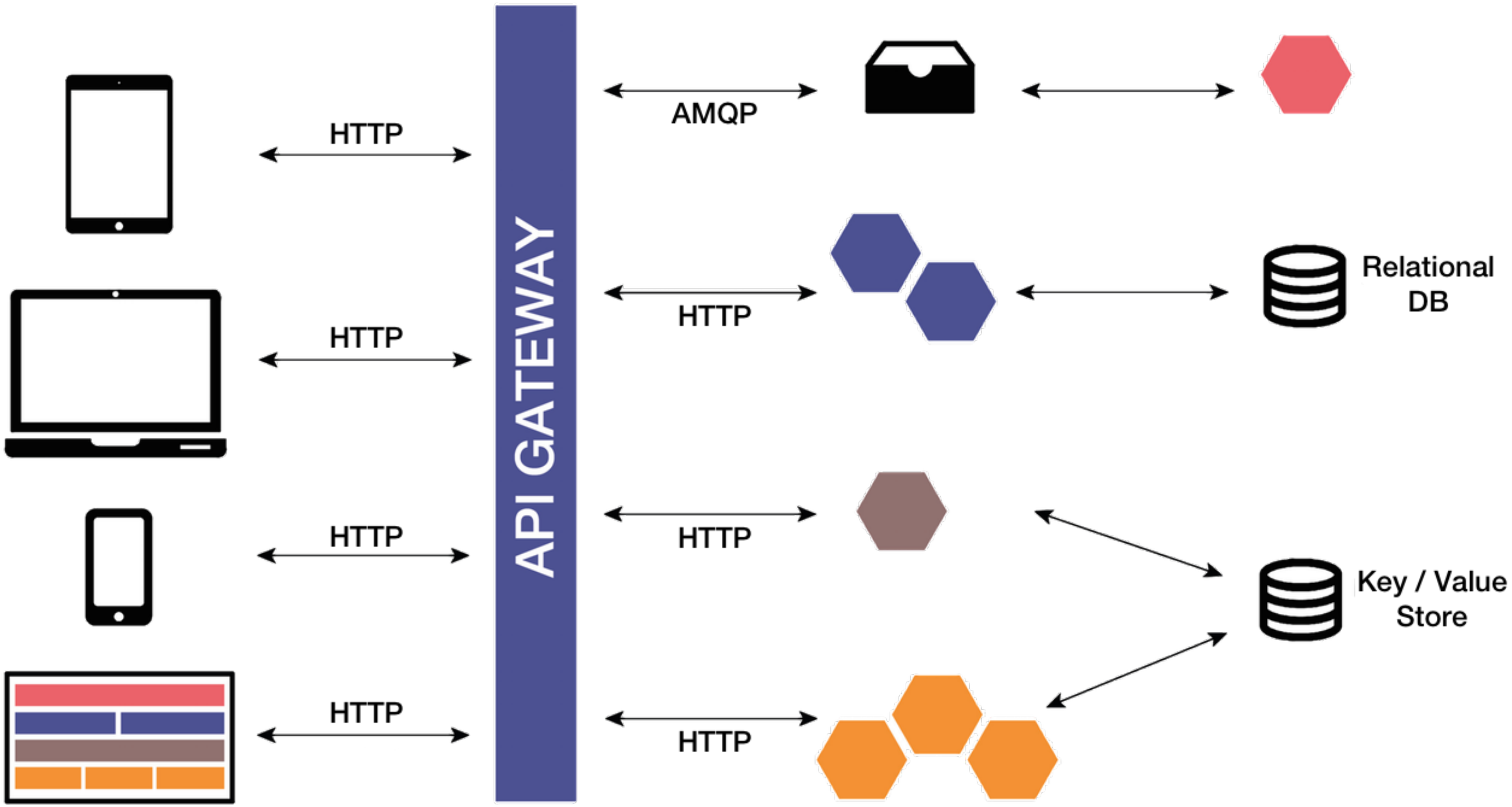


... and scales by replicating the monolith on multiple servers

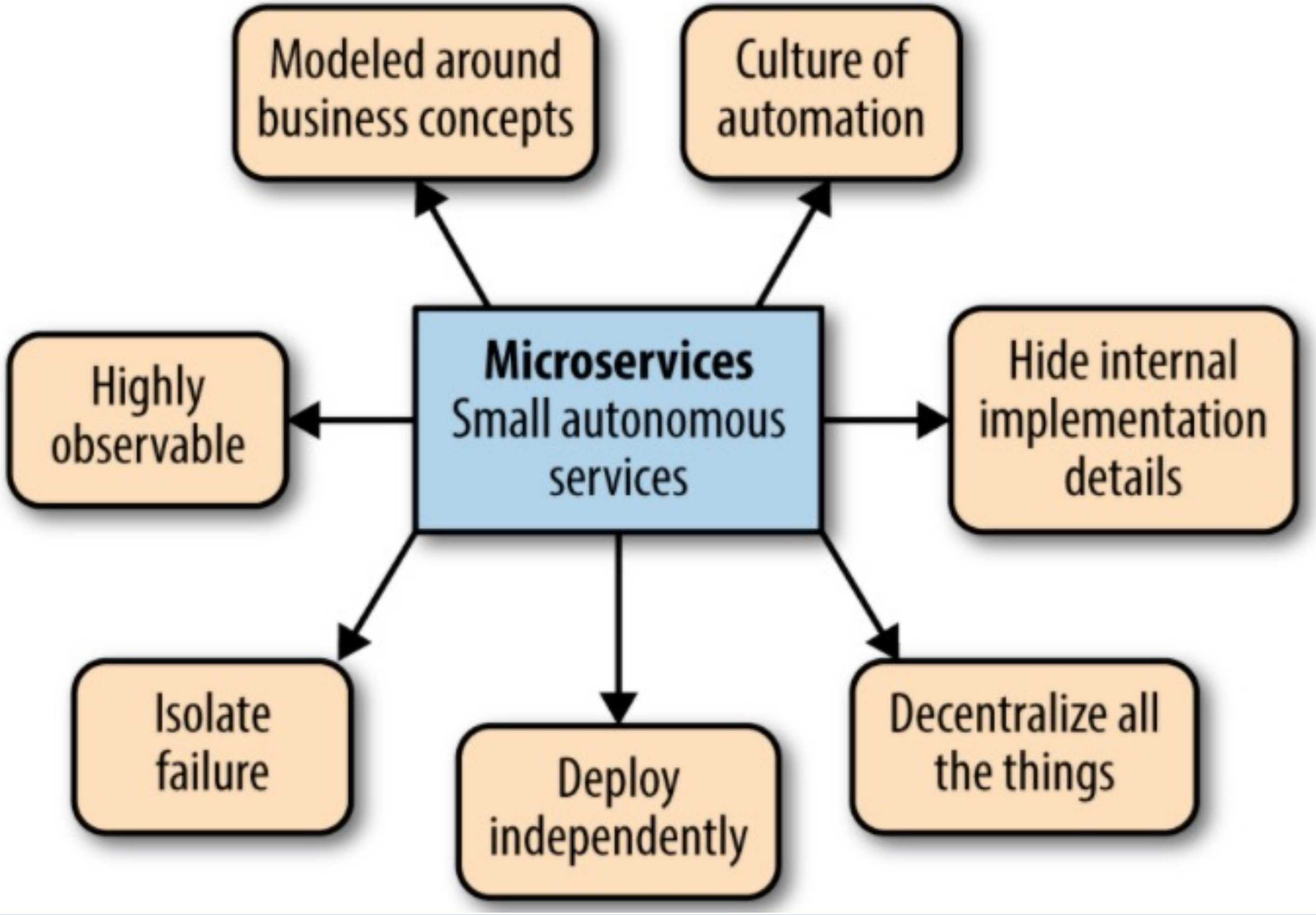


... and scales by distributing these services across servers, replicating as needed.



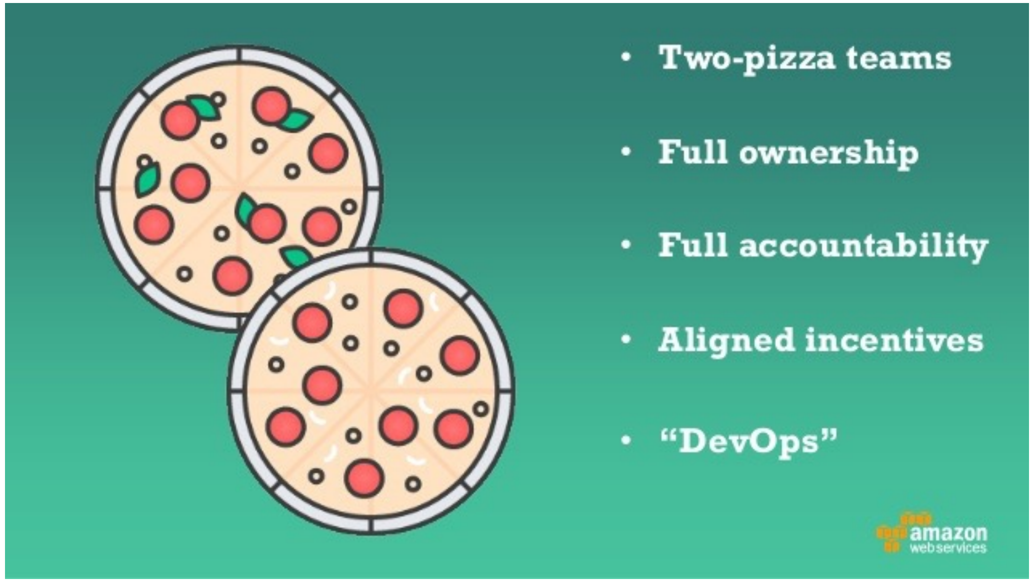

HTTP
HTTP
HTTP
HTTP
API GATEWAY
AMQP
HTTP
HTTP
HTTP
Relational DB
Key / Value Store

Principle of Microservices


Microservices
Small autonomous services
Modeled around business concepts
Culture of automation
Hide internal implementation details
Decentralize all the things
Deploy independently
Isolate failure
Highly observable

Benefits of Microservices

- Faster and simpler deployments and rollbacks
- Elimination of long-term commitment to a single technology stack
- Improved fault isolation
- Independently scalable services
- Technology diversity
- Ability to write new features as plugins

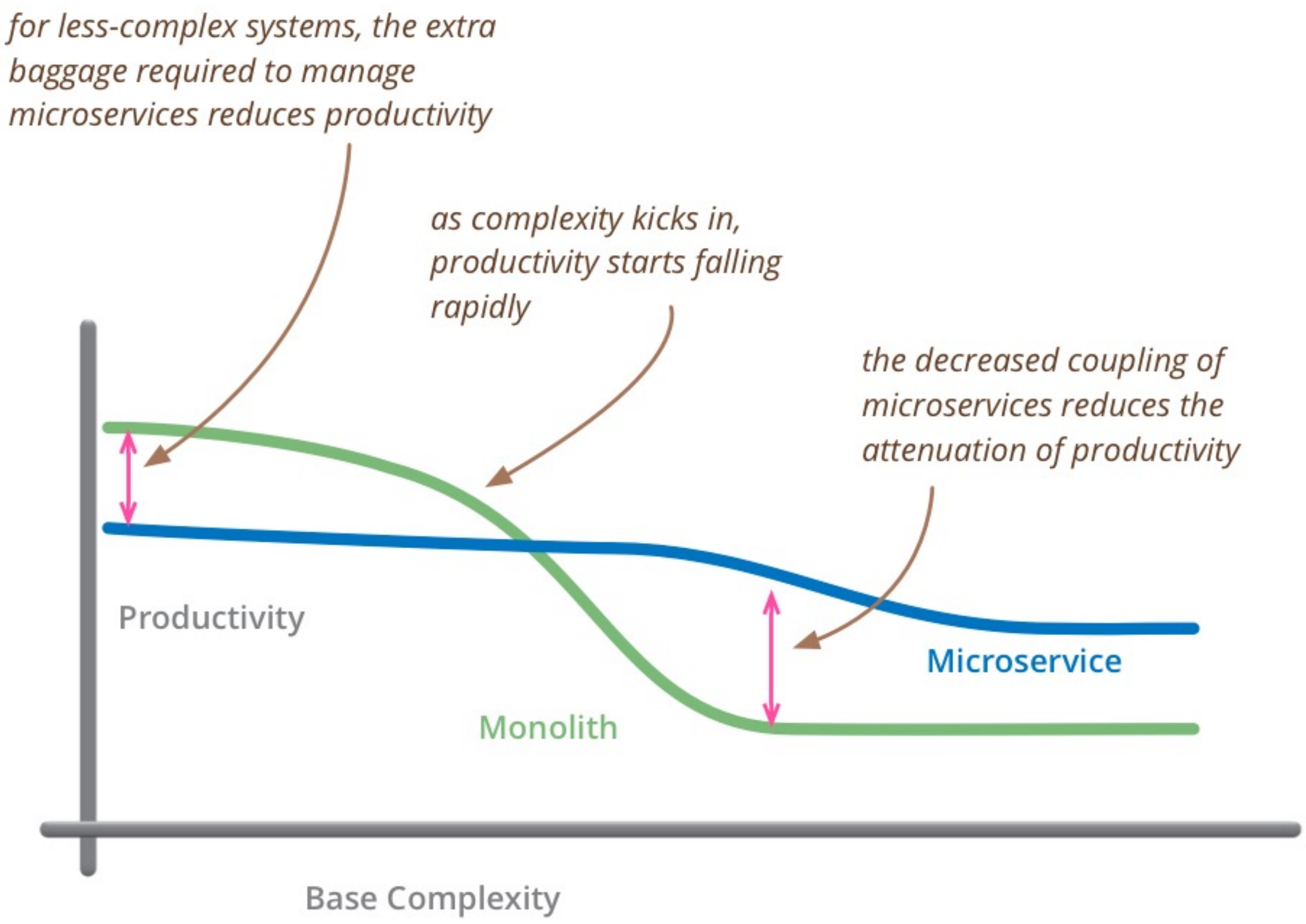


- Two-pizza teams
- Full ownership
- Full accountability
- Aligned incentives
- "DevOps"

Drawbacks of Microservices

- Increased network communication
- Serialization between microservices
- Additional complexity in testing a distributed system
- Increased complexity in deployment

Microservies overhead



for less-complex systems, the extra baggage required to manage microservices reduces productivity

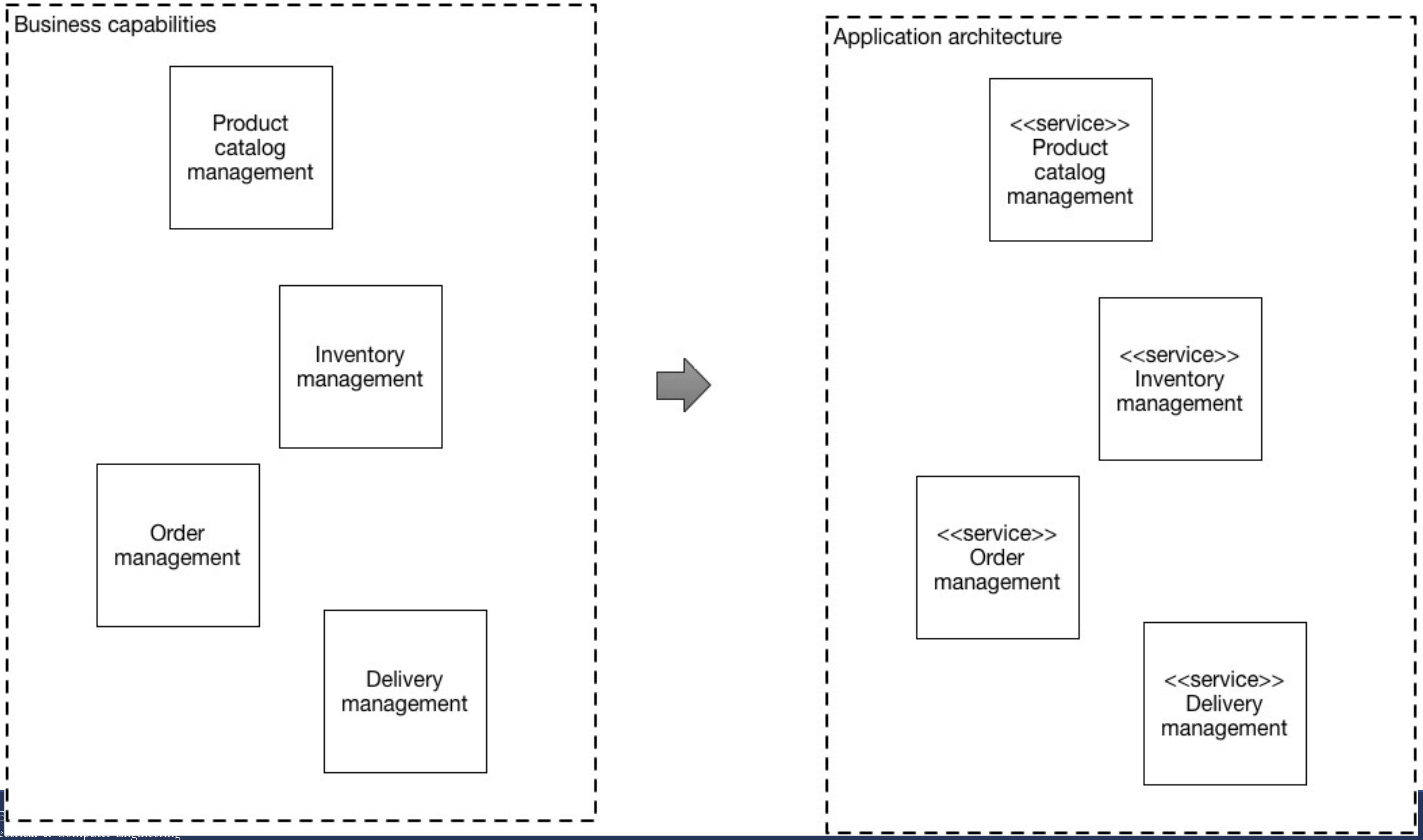
as complexity kicks in, productivity starts falling rapidly

the decreased coupling of microservices reduces the attenuation of productivity

but remember the skill of the team will outweigh any monolith/microservice choice

How to decompose the application into services?

- Decompose by business capability
- Decompose by verb or use case
- Decompose by by nouns or resources



Business capabilities

Product catalog management

Inventory management

Order management

Delivery management

Application architecture

<<service>>
Product catalog management

<<service>>
Inventory management

<<service>>
Order management

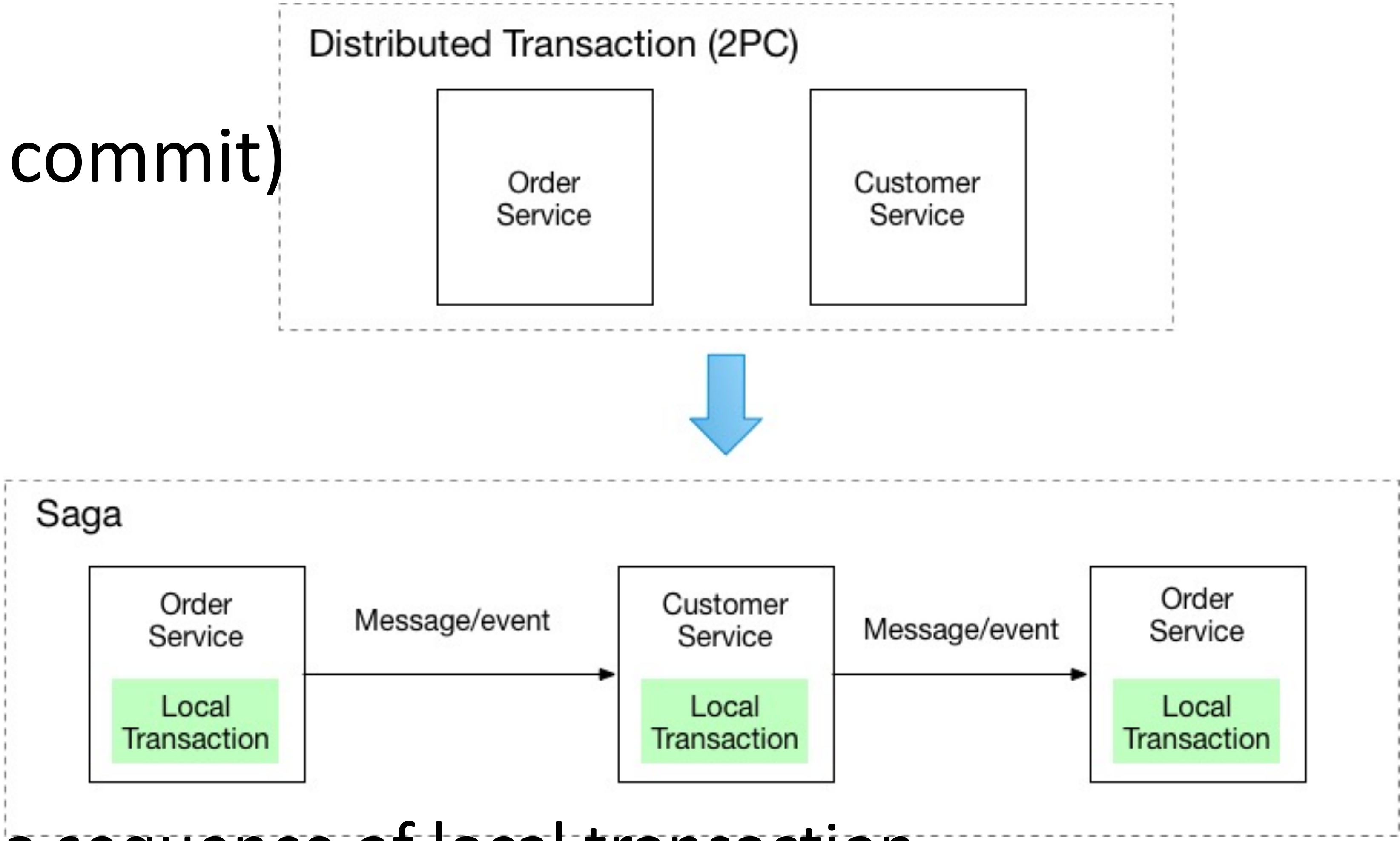
<<service>>
Delivery management

How to decompose the application into services?

- Decompose by business capability
- Decompose by verb or use case
- Decompose by by nouns or resources

How to maintain data consistency?

- ✗ 2PC (Two-phase commit)
- ✓ Saga Pattern



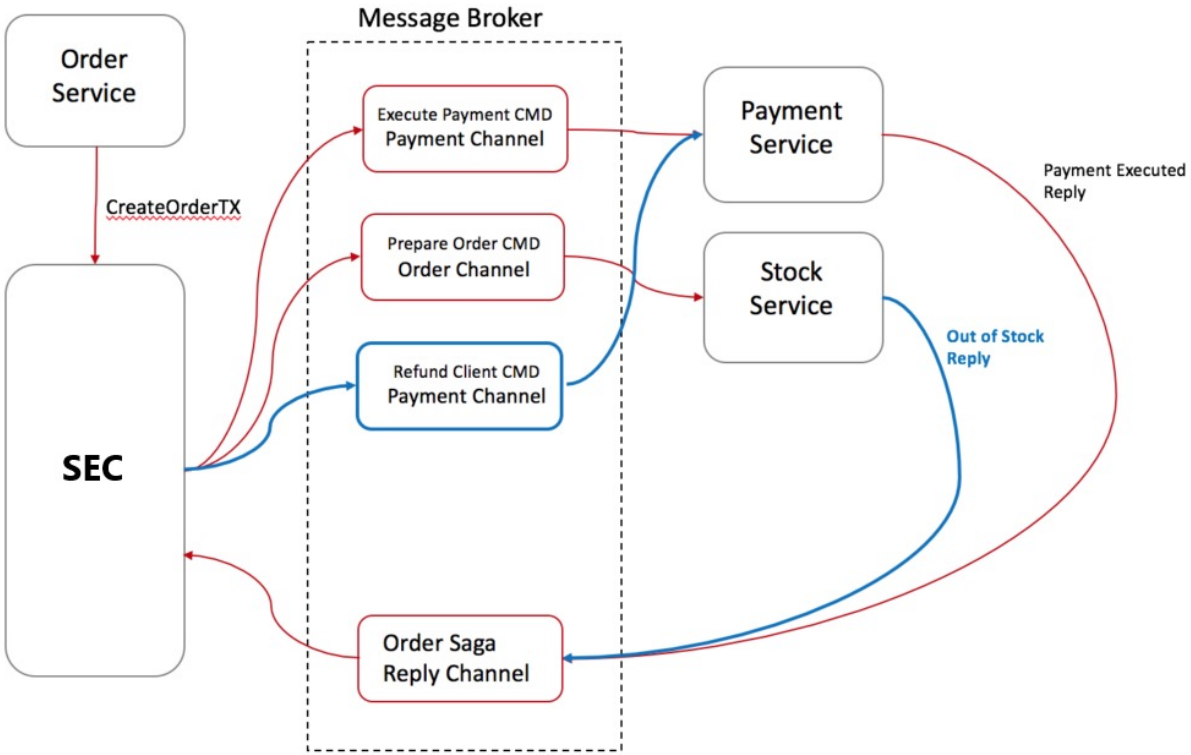
saga – a sequence of local transaction

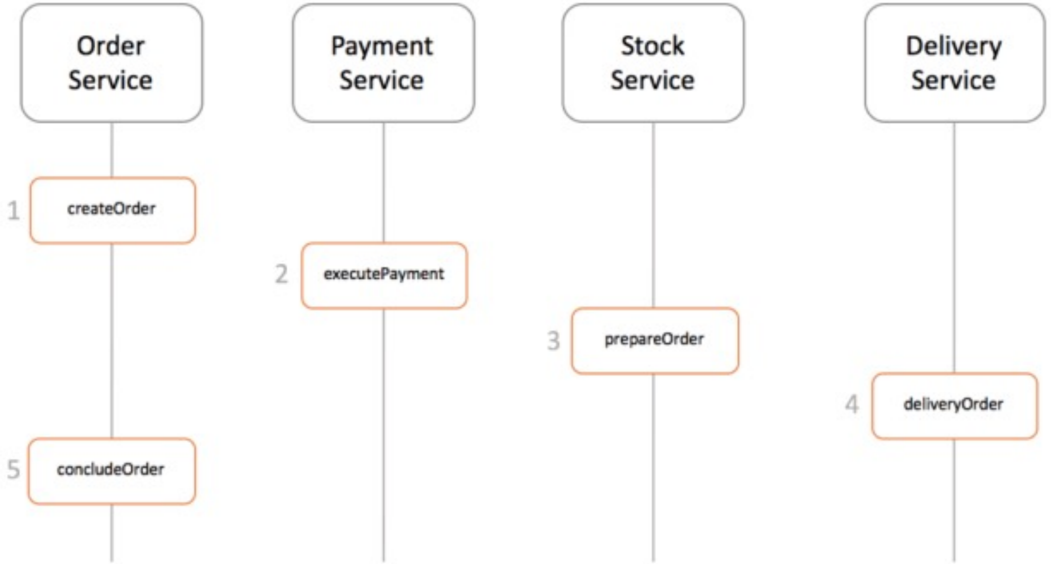
Saga Pattern

The master process called “**Saga Execution Coordinator**” or SEC.

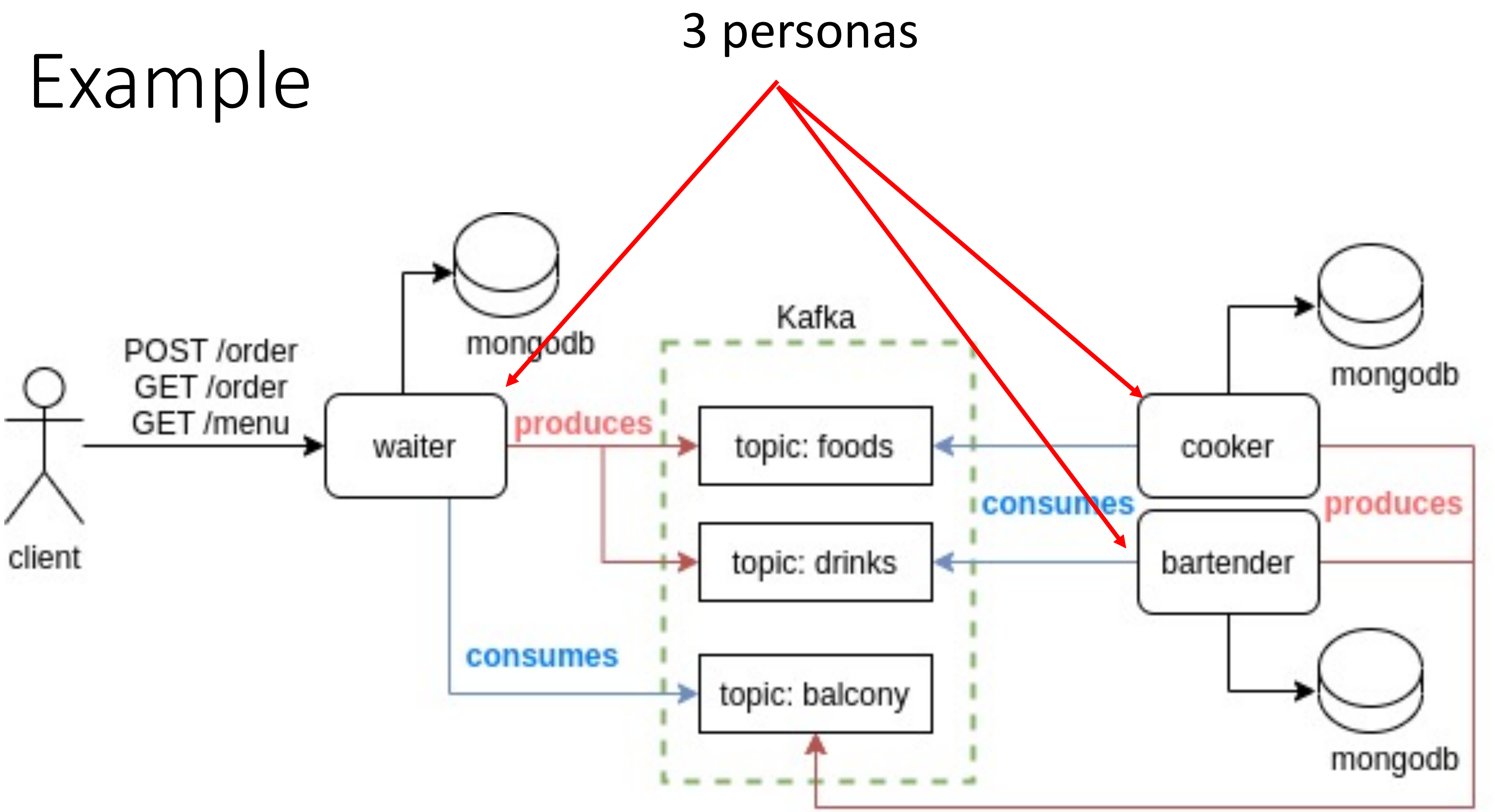
- two ways to achieve sagas
  - Choreography : each local transaction publishes domain events that trigger local transactions in other services.
  - Orchestration : an orchestrator (object) tells the participants what local transactions to execute.

Orchestration





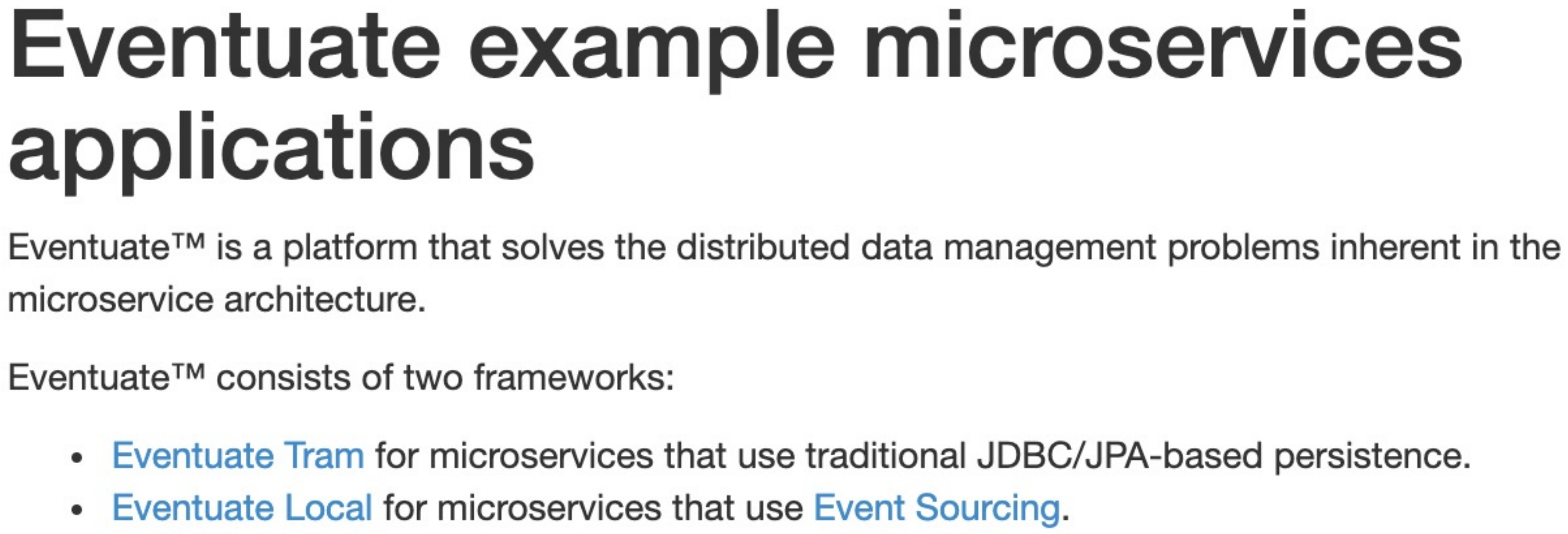
Example



https://github.com/victoramsantos/saga-pattern-example

Other examples and platforms

eventuate



Eventuate example microservices applications

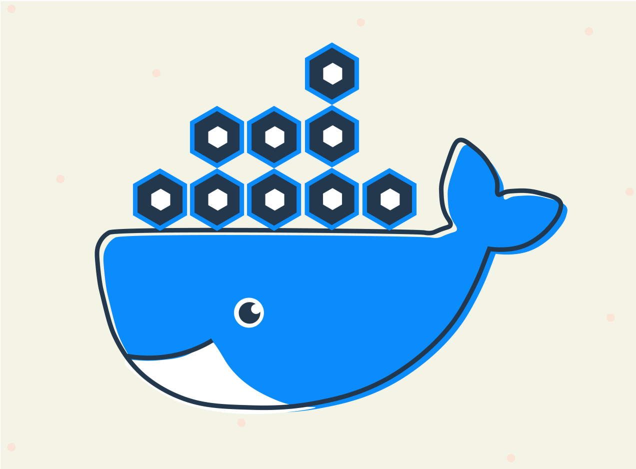
Eventuate™ is a platform that solves the distributed data management problems inherent in the microservice architecture.

Eventuate™ consists of two frameworks:

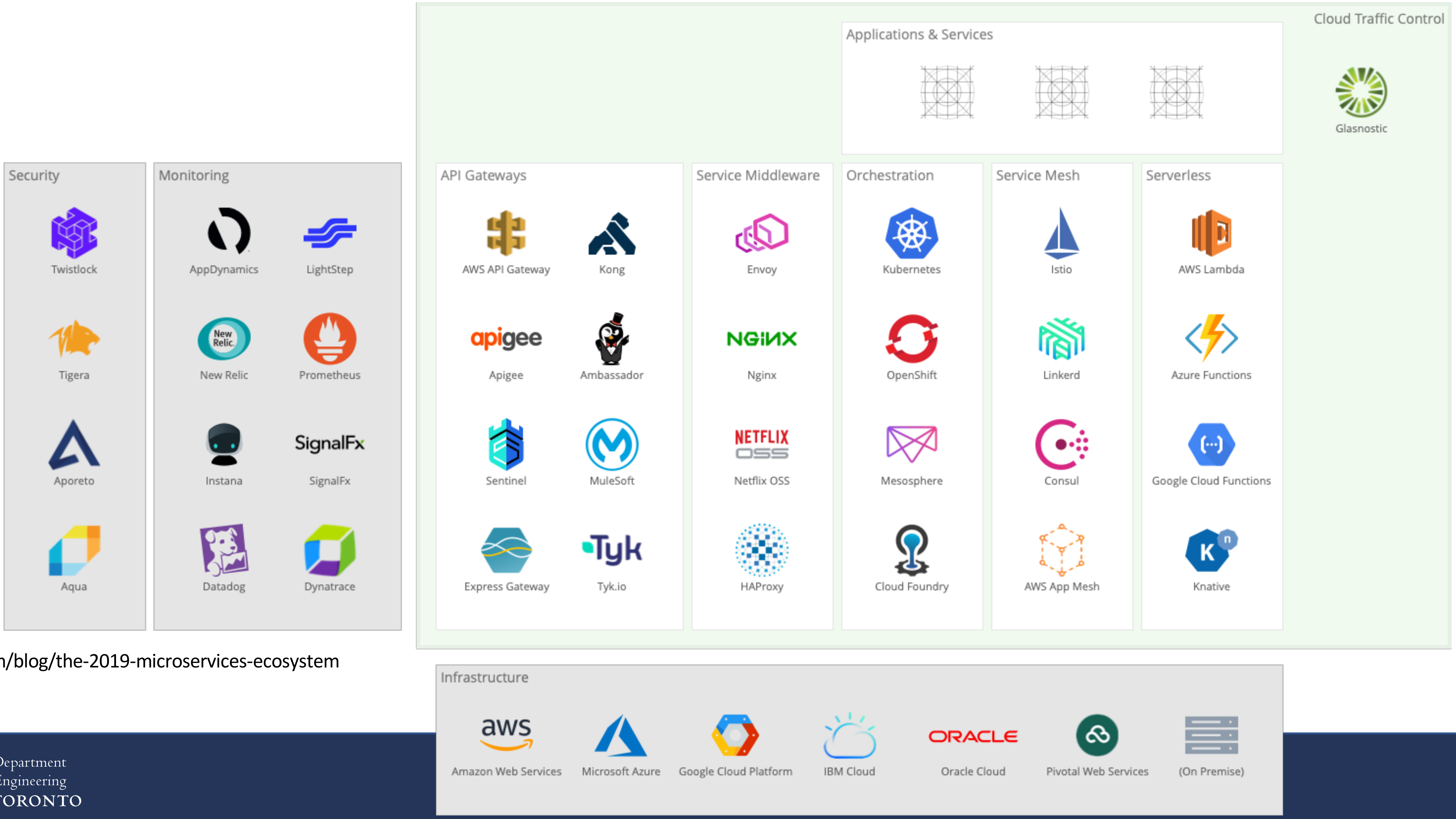
- Eventuate Tram for microservices that use traditional JDBC/JPA-based persistence.
- Eventuate Local for microservices that use Event Sourcing.

How are services packaged and deployed?

- Container
- Serverless deployment
- Platform as a Service (PaaS)

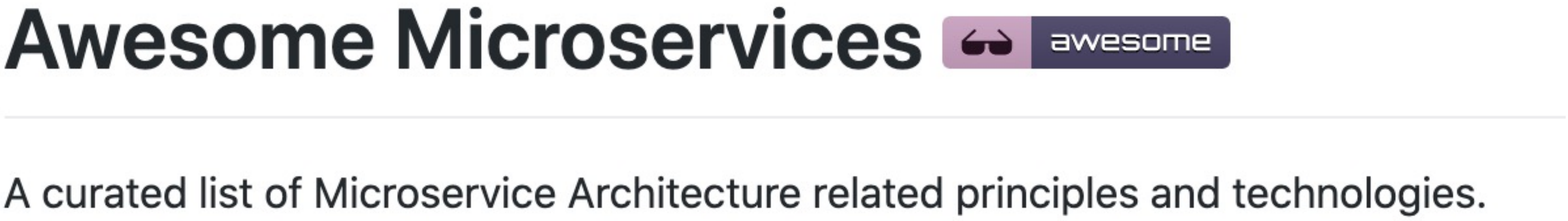


The 2019 Microservices Ecosystem



https://glasnostic.com/blog/the-2019-microservices-ecosystem

Technology Stacks



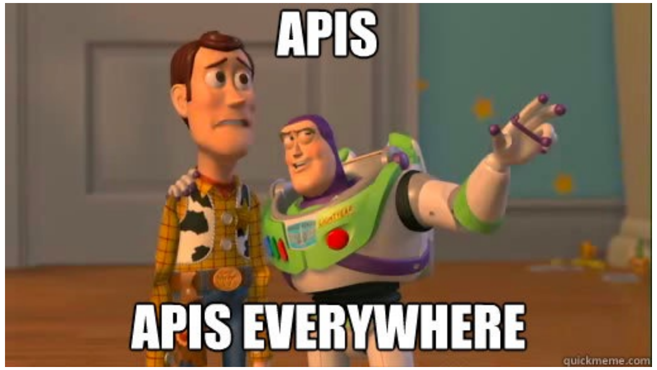
Awesome Microservices

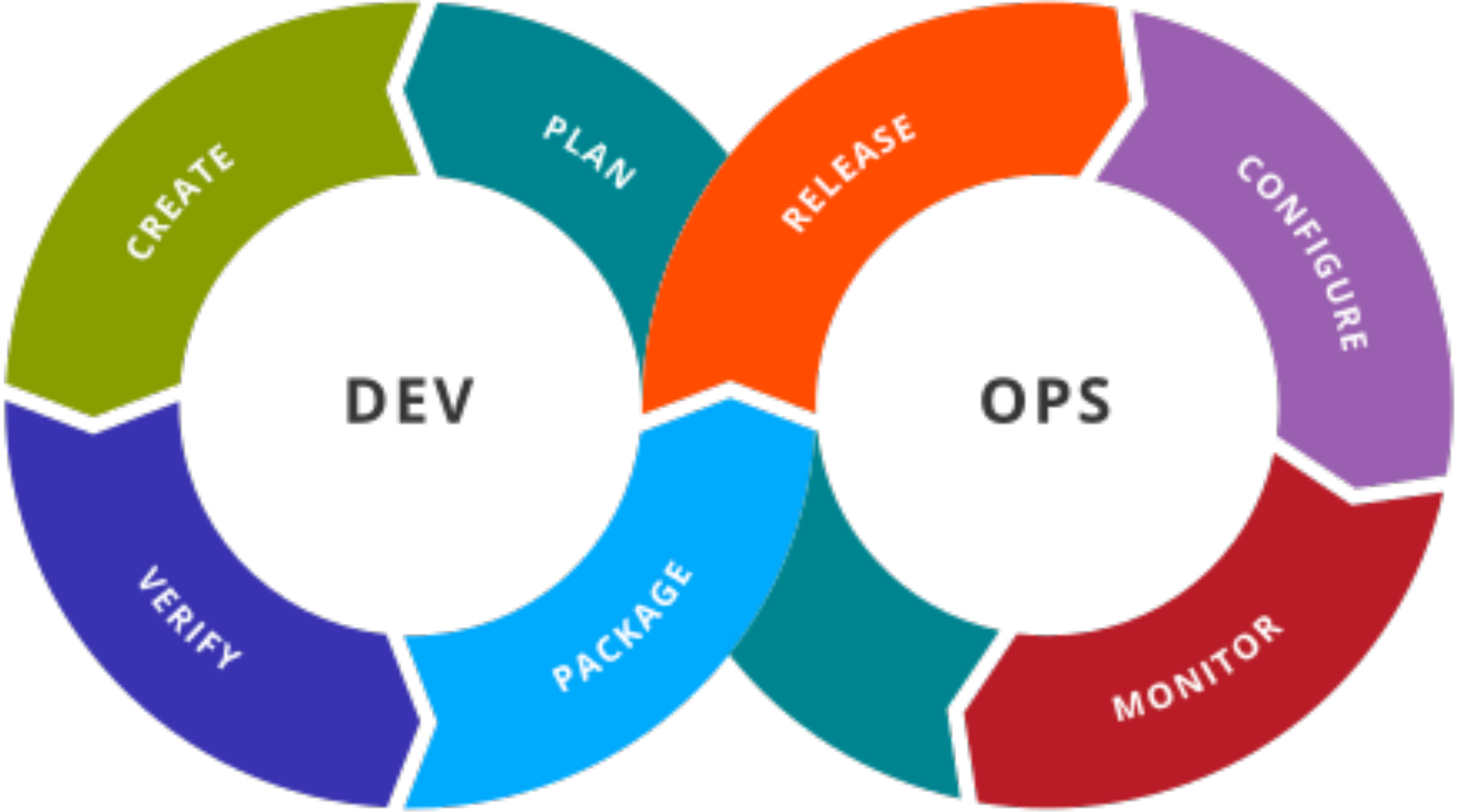
A curated list of Microservice Architecture related principles and technologies.

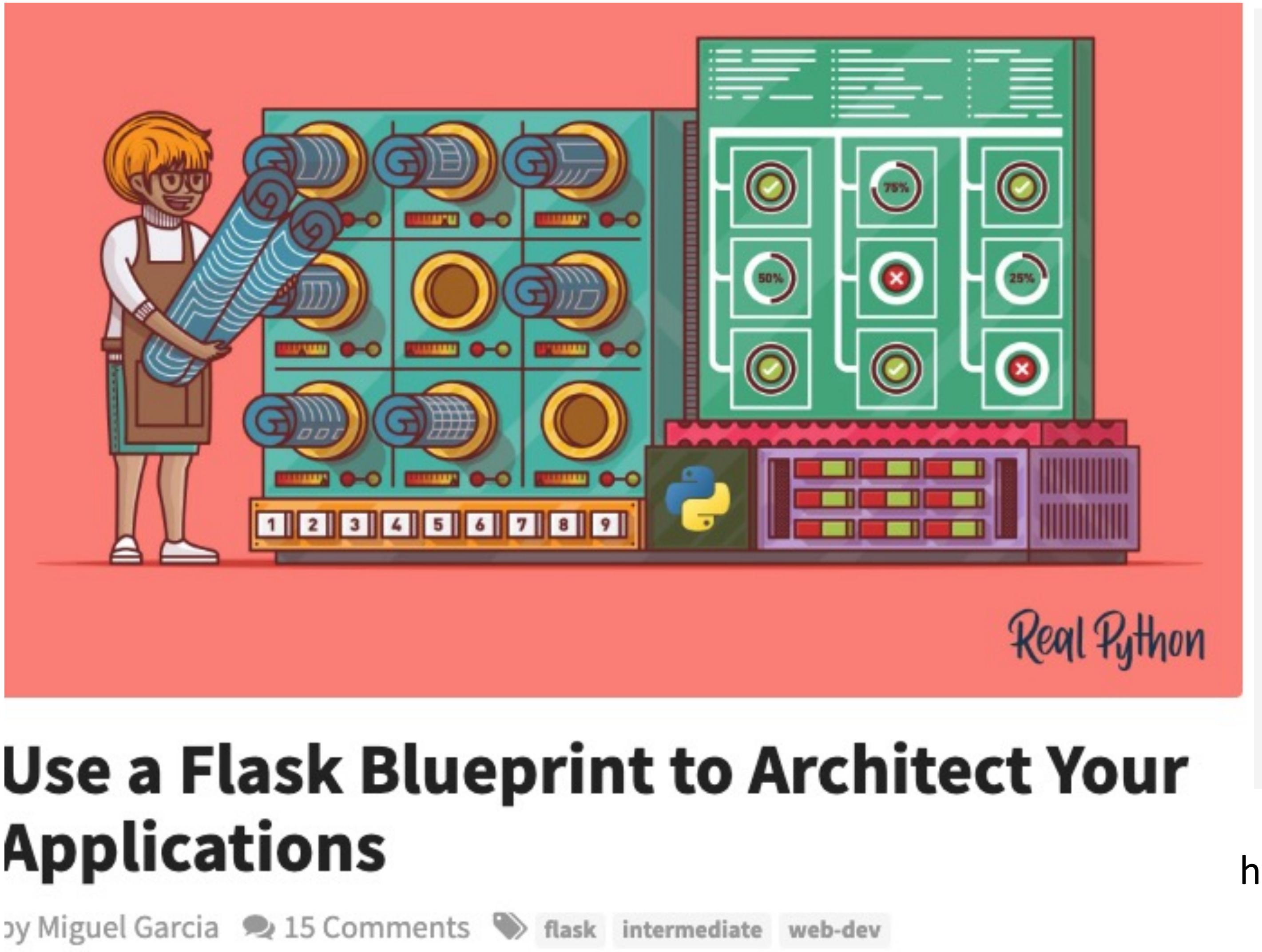
https://github.com/mfornos/awesome-microservices

Discussion of Microservices

- Are they really “new”?
- Do microservices solve problems, or push them down the line?
- What are the impacts of the added flexibility?
- Beware “cargo cult”
- “If you can’t build a well-structured monolith, what makes you think microservices is the answer?” – Simon Brown
- Leads to more API design decisions




CREATE
PLAN
RELEASE
CONFIGURE
DEV
OPS
VERIFY
PACKAGE
MONITOR



Use a Flask Blueprint to Architect Your Applications

by Miguel Garcia 15 Comments flask intermediate web-dev

https://realpython.com/flask-blueprint/

An introduction to the Flask Python web app framework

In the first part in a series comparing Python frameworks, learn about Flask.

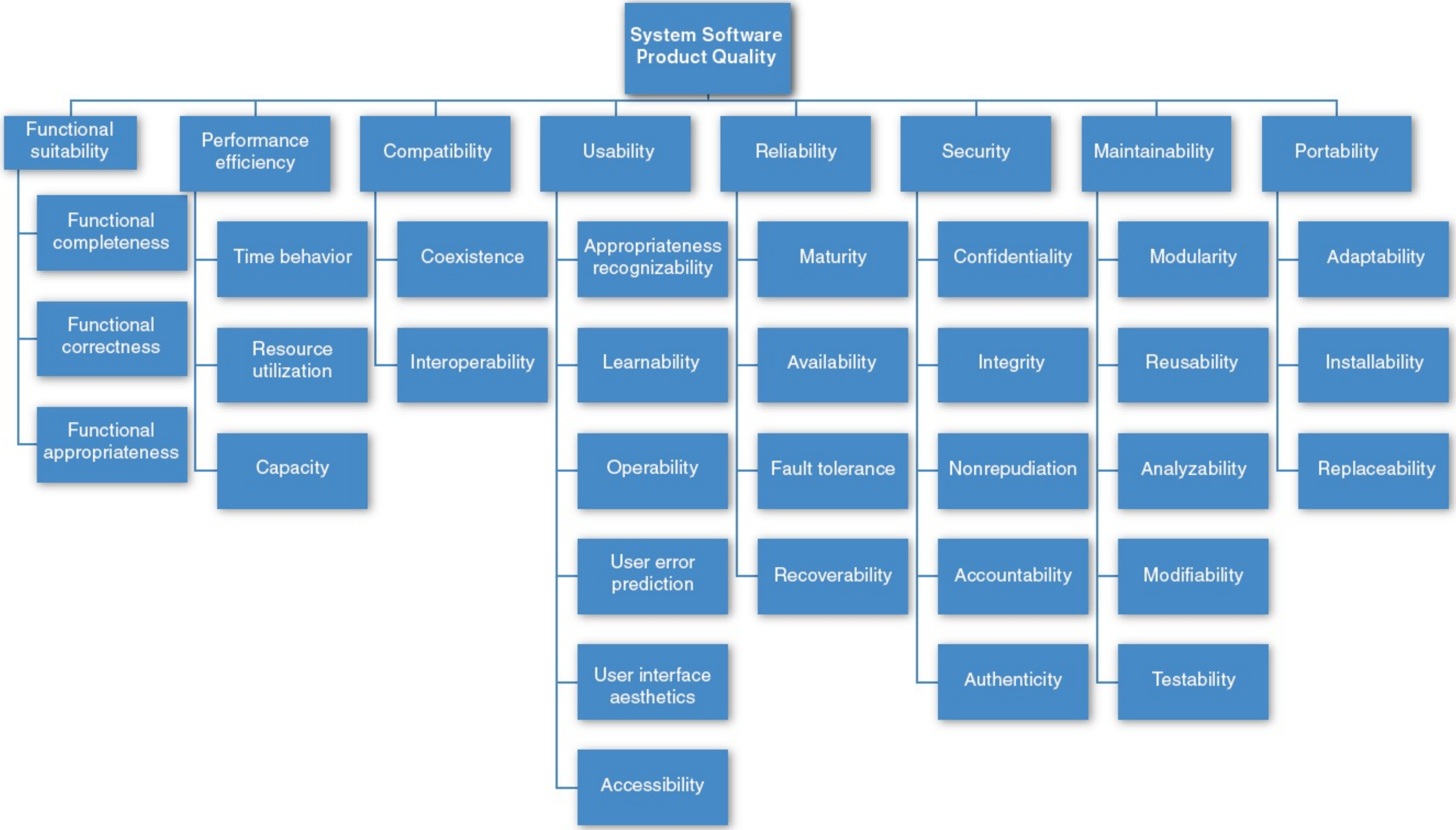
02 Apr 2018 | Nicholas Hunt-Walker | 390 | 3 comments

https://opensource.com/article/18/4/flask

Tactics

- Architectural techniques to achieve qualities
  - More tied to specific context and quality
- Smaller scope than architectural patterns
  - Problem solved by patterns: “How do I structure my (sub)system?”
  - Problem solved by tactics: “How do I get better at quality X?”
- Collection of common strategies and known solutions
  - Resemble OO design patterns

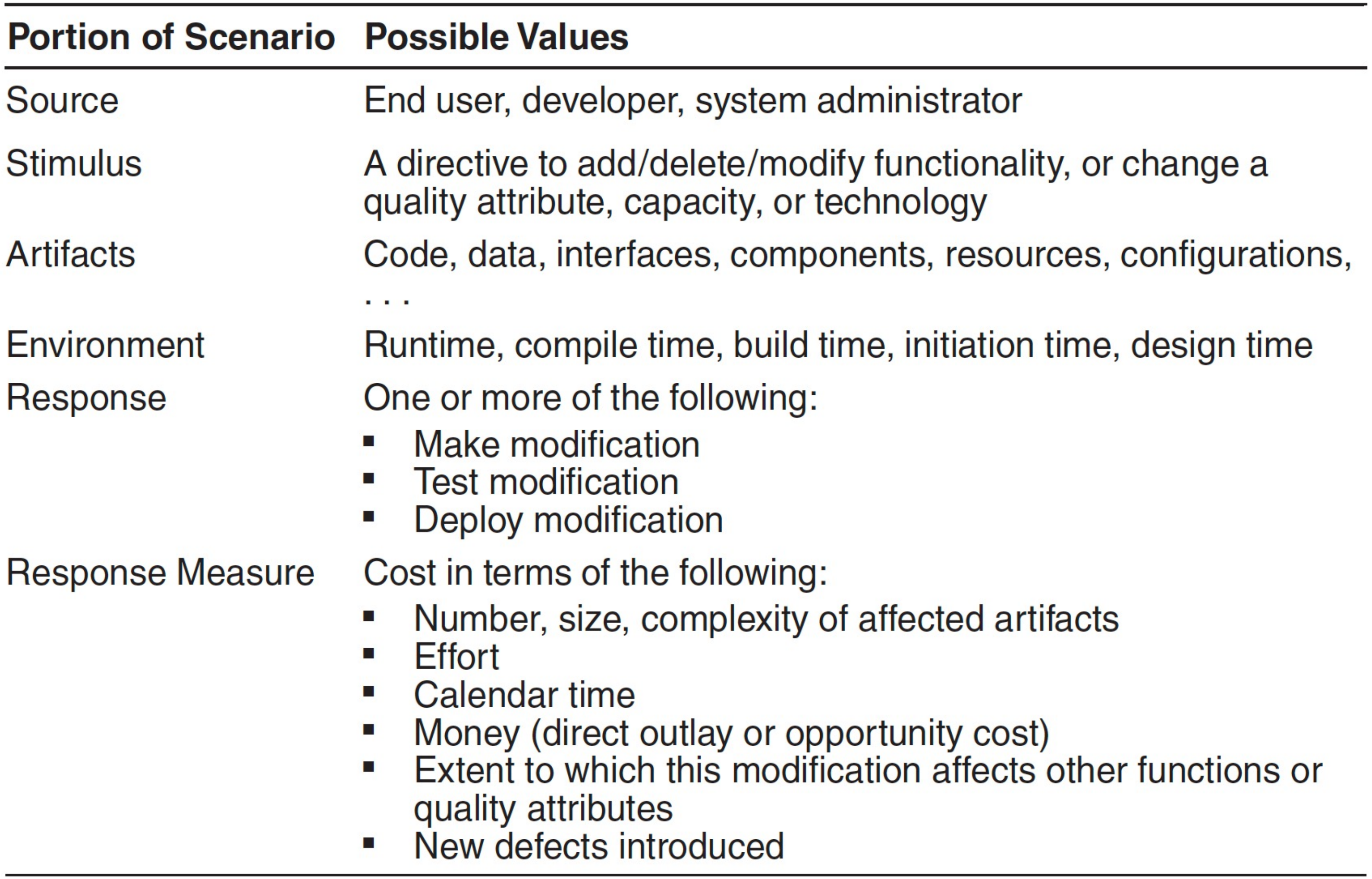
Achieving Quality Attributes through Tactics



System Software Product Quality

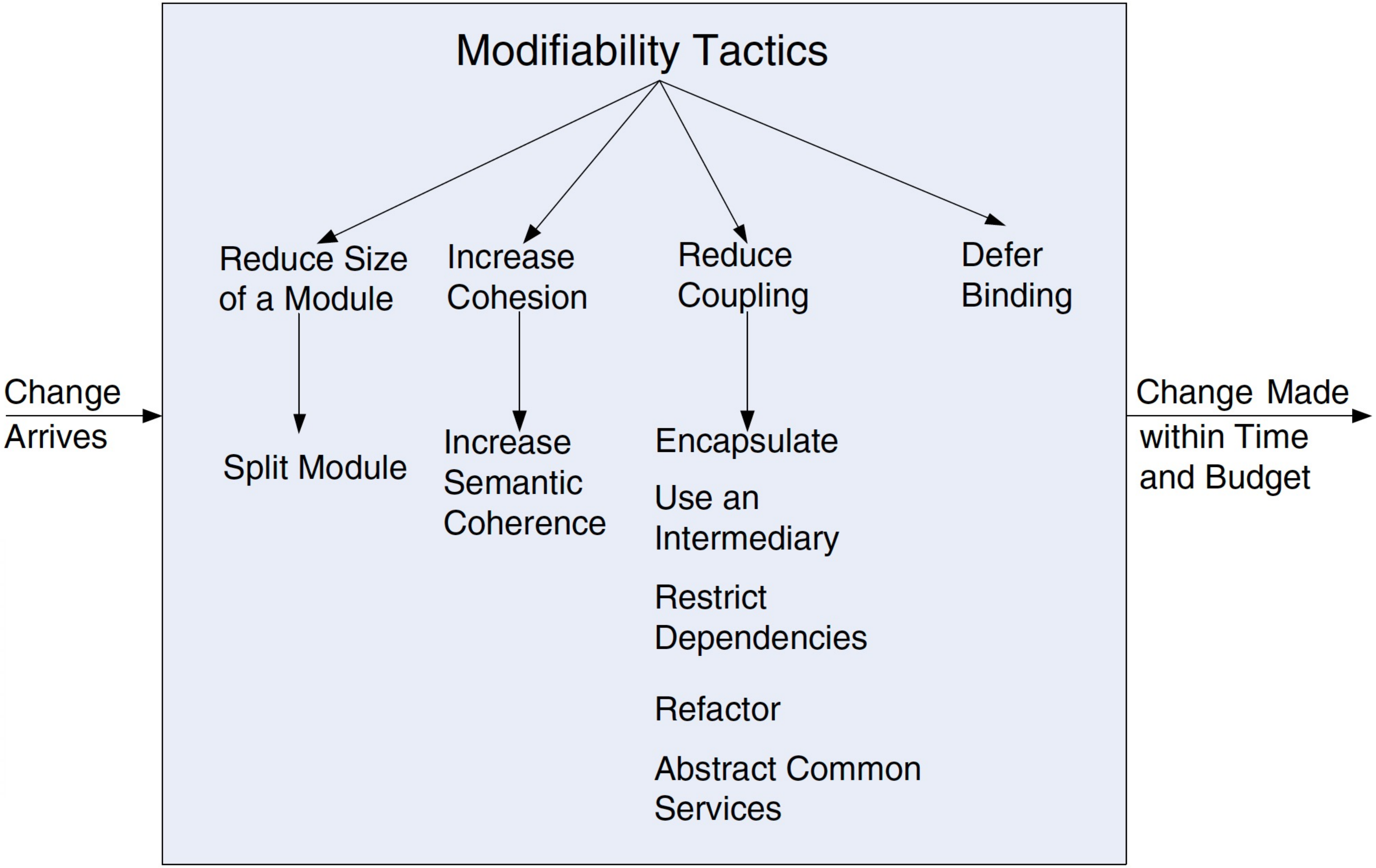
- Functional suitability
  - Functional completeness
  - Functional correctness
  - Functional appropriateness
- Performance efficiency
  - Time behavior
  - Resource utilization
  - Capacity
- Compatibility
  - Coexistence
  - Interoperability
- Usability
  - Appropriateness recognizability
  - Learnability
  - Operability
  - User error prediction
  - User interface aesthetics
  - Accessibility
- Reliability
  - Maturity
  - Availability
  - Fault tolerance
  - Recoverability
- Security
  - Confidentiality
  - Integrity
  - Nonrepudiation
  - Accountability
  - Authenticity
- Maintainability
  - Modularity
  - Reusability
  - Analyzability
  - Modifiability
  - Testability
- Portability
  - Adaptability
  - Installability
  - Replaceability

Modifiability



Portion of Scenario	Possible Values
Source	End user, developer, system administrator
Stimulus	A directive to add/delete/modify functionality, or change a quality attribute, capacity, or technology
Artifacts	Code, data, interfaces, components, resources, configurations, . . .
Environment	Runtime, compile time, build time, initiation time, design time
Response	One or more of the following: ▪ Make modification ▪ Test modification ▪ Deploy modification
Response Measure	Cost in terms of the following: ▪ Number, size, complexity of affected artifacts ▪ Effort ▪ Calendar time ▪ Money (direct outlay or opportunity cost) ▪ Extent to which this modification affects other functions or quality attributes ▪ New defects introduced

Modifiability



- coupling - probability that a modification to one module will propagate to the other
- cohesion - how strongly the responsibilities of a module are related

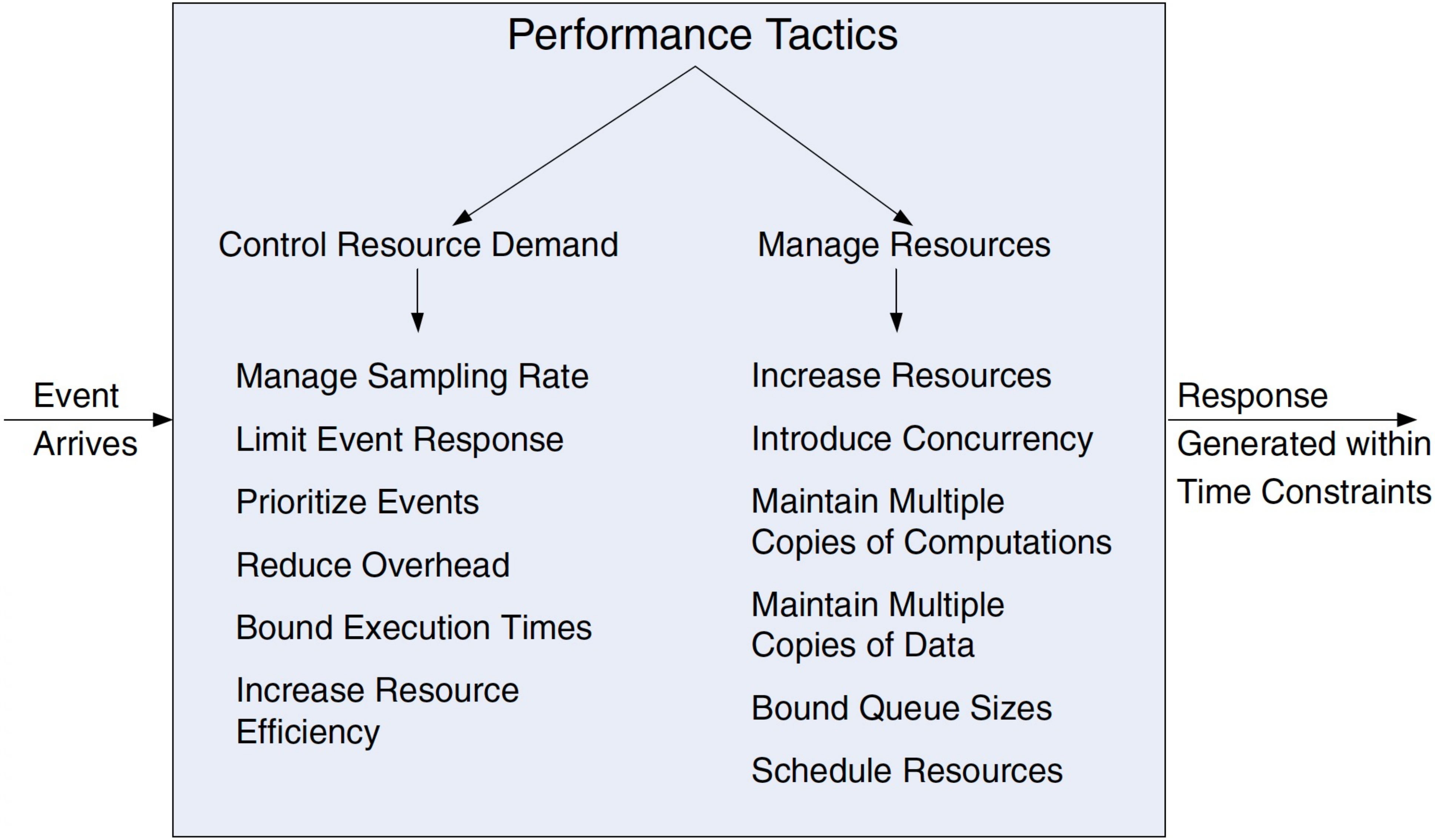
<u>Low coupling, high cohesion, better modifiability</u>

Performance

- about time and the software system’s ability to meet timing requirements
- Event arrival patterns: Periodic, Stochastic, Sporadic
- Measurements:
  - Latency
  - Deadlines in processing
  - Throughput
  - jitter of the respsonse
  - number of events not processed

Performance

response time = processing time + blocked time



Performance Tactics

Control Resource Demand

- Manage Sampling Rate
- Limit Event Response
- Prioritize Events
- Reduce Overhead
- Bound Execution Times
- Increase Resource Efficiency

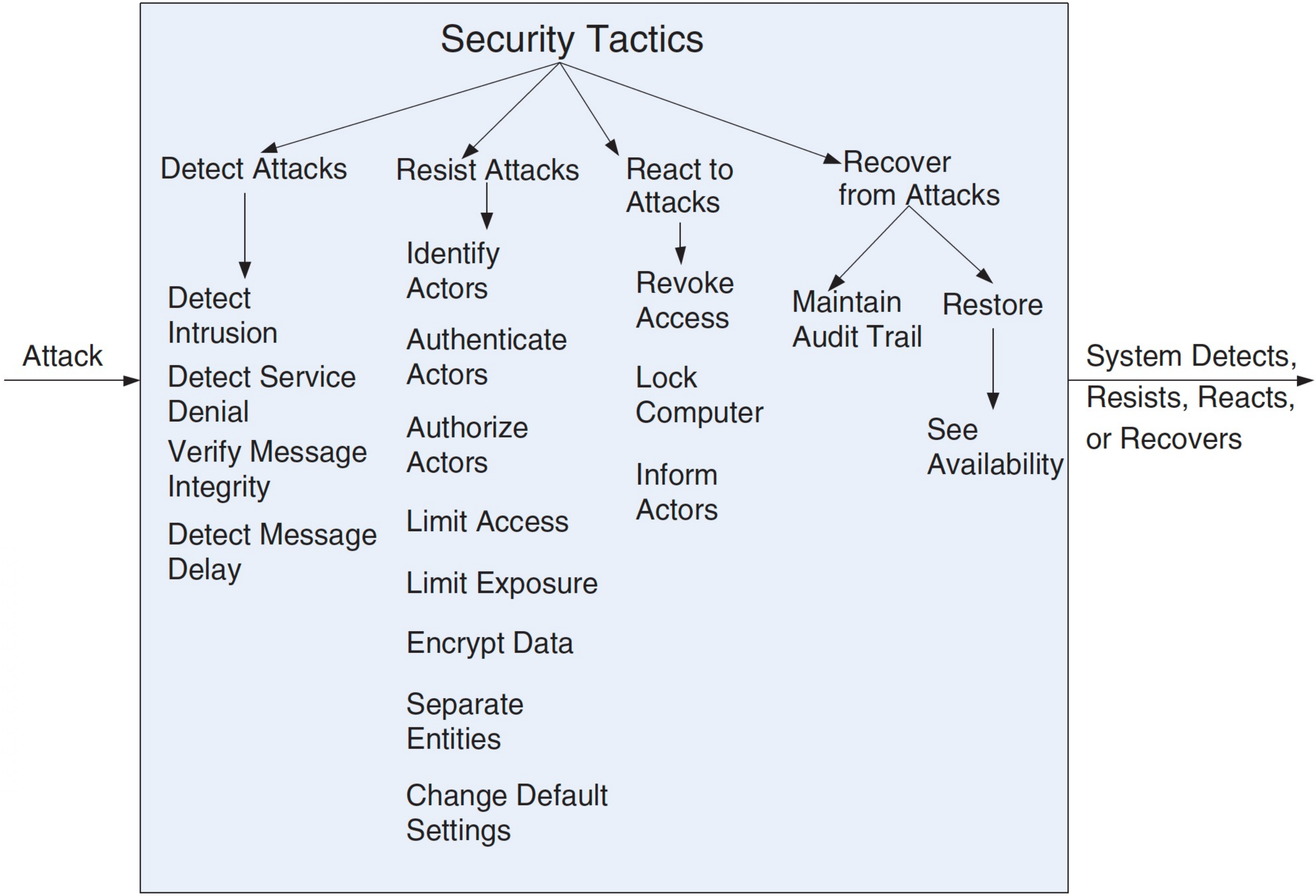
Manage Resources

- Increase Resources
- Introduce Concurrency
- Maintain Multiple Copies of Computations
- Maintain Multiple Copies of Data
- Bound Queue Sizes
- Schedule Resources

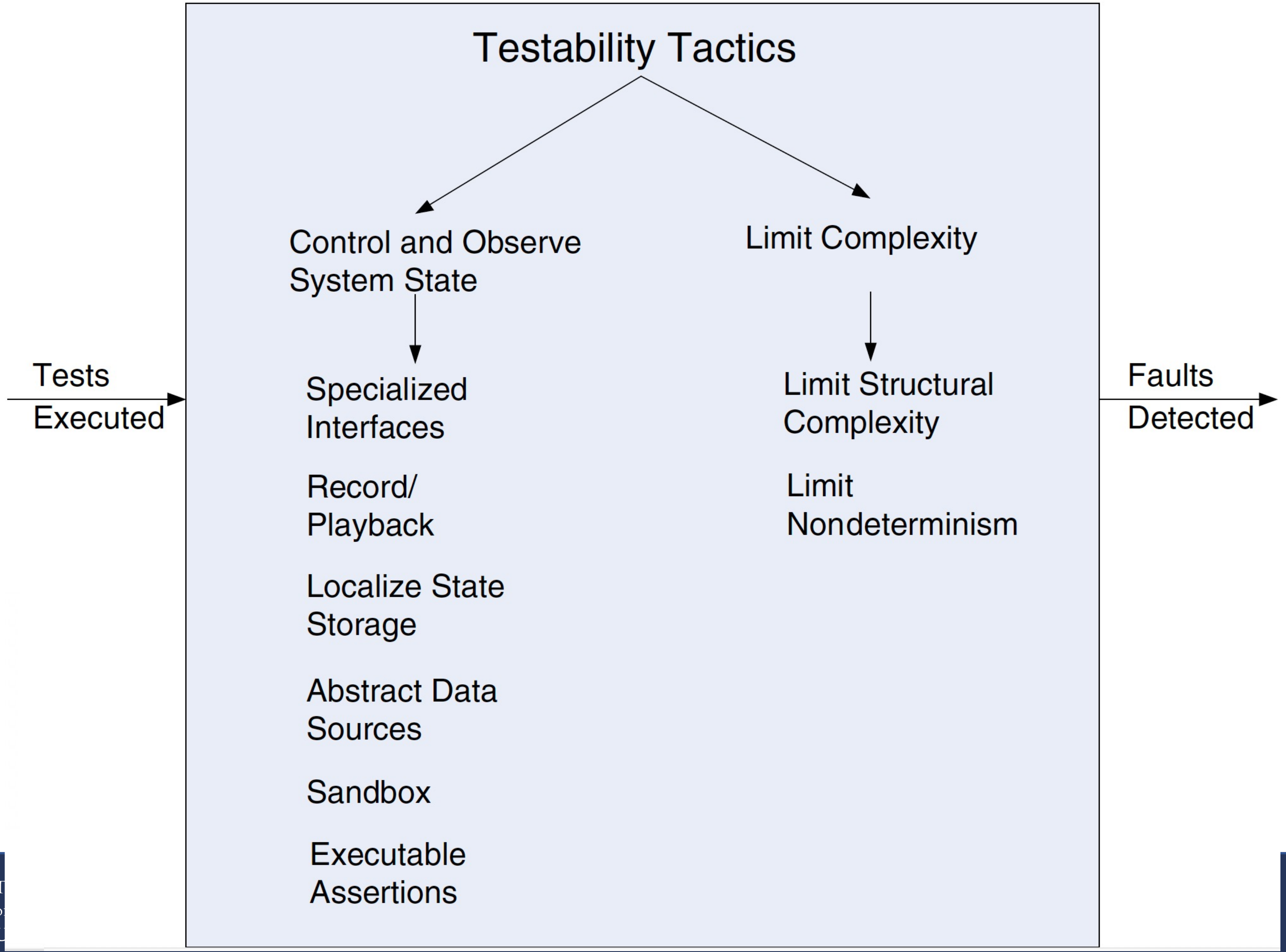
Event Arrives

Response Generated within Time Constraints

Security


Security Tactics
Detect Attacks
Resist Attacks
React to Attacks
Recover from Attacks
Detect Intrusion
Detect Service Denial
Verify Message Integrity
Detect Message Delay
Identify Actors
Authenticate Actors
Authorize Actors
Limit Access
Limit Exposure
Encrypt Data
Separate Entities
Change Default Settings
Revoke Access
Lock Computer
Inform Actors
Maintain Audit Trail
Restore
See Availability
Attack
System Detects, Resists, Reacts, or Recovers

Testability



Testability Tactics

Control and Observe System State

- Specialized Interfaces
- Record/ Playback
- Localize State Storage
- Abstract Data Sources
- Sandbox
- Executable Assertions

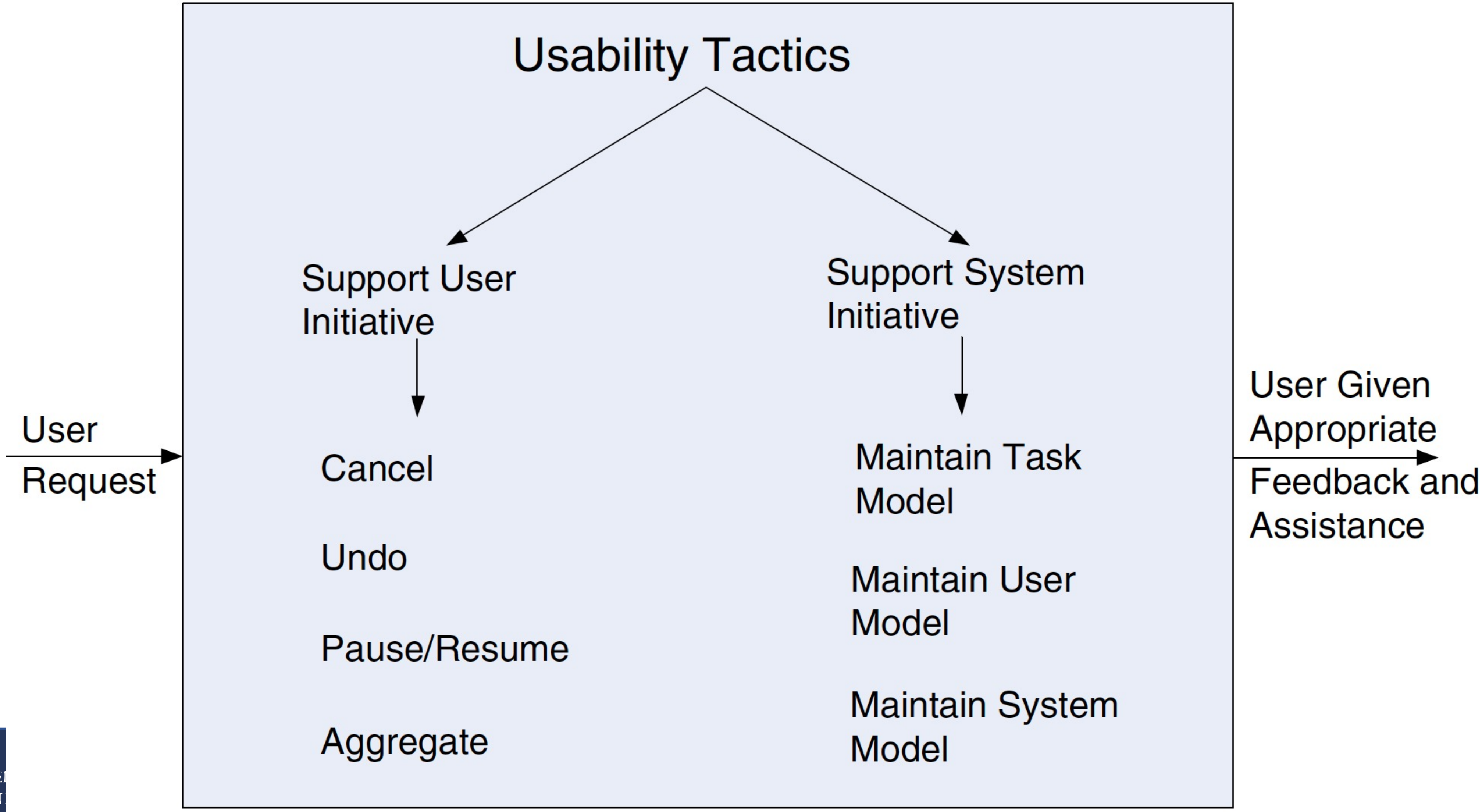
Limit Complexity

- Limit Structural Complexity
- Limit Nondeterminism

Tests Executed

Faults Detected

Usability



User Request

Usability Tactics

Support User Initiative

- Cancel
- Undo
- Pause/Resume
- Aggregate

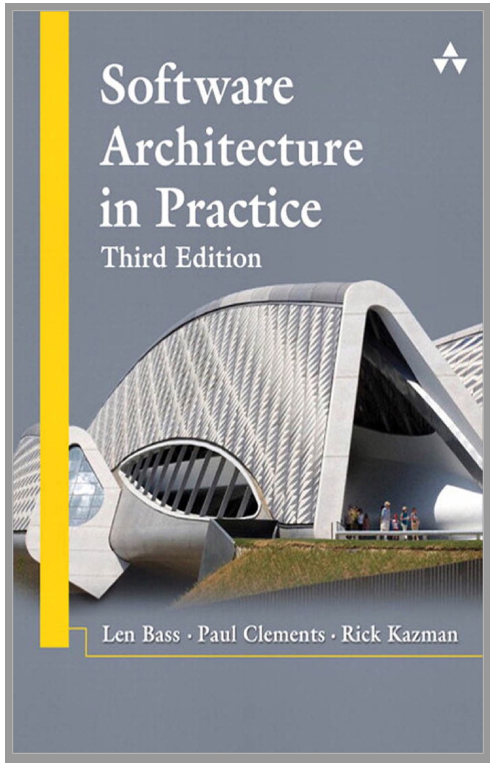
Support System Initiative

- Maintain Task Model
- Maintain User Model
- Maintain System Model

User Given Appropriate Feedback and Assistance

Summary of Tactics and Patterns

Tactics are the “building blocks” of design, from which architectural patterns are created. Tactics are atoms and patterns are molecules. Most patterns consist of several different tactics.


Software Architecture in Practice
Third Edition
Len Bass · Paul Clements · Rick Kazman

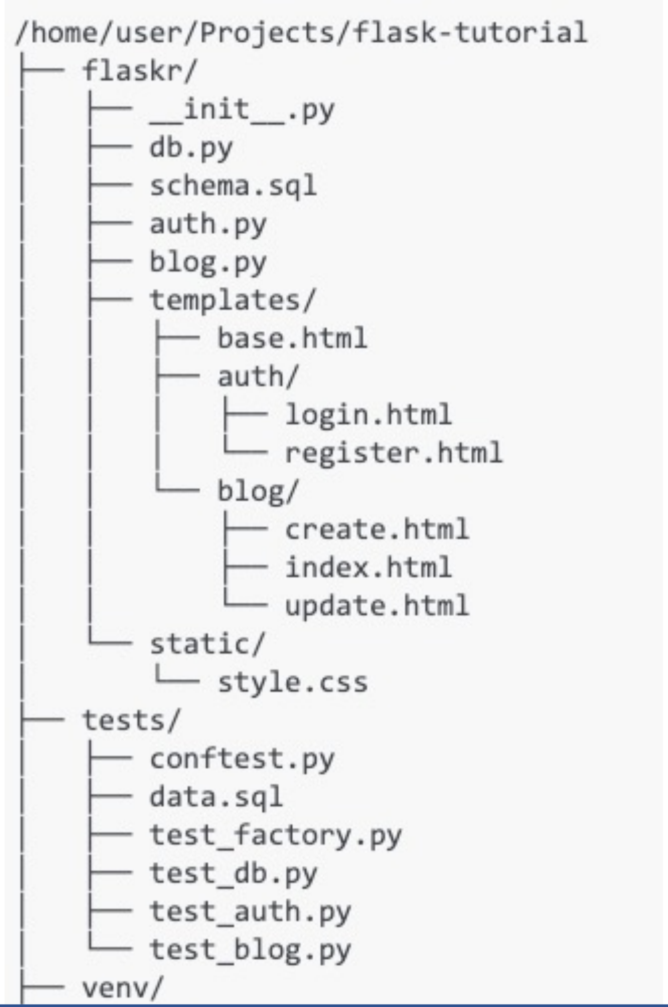
Many tactics described in Chapter 4-10

- Brief high-level descriptions (about 1 paragraph per tactic)
- Checklist available

Future Readings

- Bass, Clements, and Kazman. Software Architecture in Practice. Addison-Wesley, 2013.
- Boehm and Turner. Balancing Agility and Discipline: A Guide for the Perplexed, 2003.
- Clements, Bachmann, Bass, Garlan, Ivers, Little, Merson, Nord, Stafford. Documenting Software Architectures: Views and Beyond, 2010.
- Fairbanks. Just Enough Software Architecture. Marshall & Brainerd, 2010.
- Jansen and Bosch. Software Architecture as a Set of Architectural Design Decisions, WICSA 2005.
- Lattanze. Architecting Software Intensive Systems: a Practitioner's Guide, 2009.
- Sommerville. Software Engineering. Edition 7/8, Chapters 11-13
- Taylor, Medvidovic, and Dashofy. Software Architecture: Foundations, Theory, and Practice. Wiley, 2009.

Flask Web Application Layout



```
/home/user/Projects/flask-tutorial
├── flaskr/
│   ├── __init__.py
│   ├── db.py
│   ├── schema.sql
│   ├── auth.py
│   ├── blog.py
│   ├── templates/
│   │   ├── base.html
│   │   ├── auth/
│   │   │   ├── login.html
│   │   │   └── register.html
│   │   └── blog/
│   │       ├── create.html
│   │       ├── index.html
│   │       └── update.html
│   └── static/
│       └── style.css
├── tests/
│   ├── conftest.py
│   ├── data.sql
│   ├── test_factory.py
│   ├── test_db.py
│   ├── test_auth.py
│   └── test_blog.py
├── venv/
```