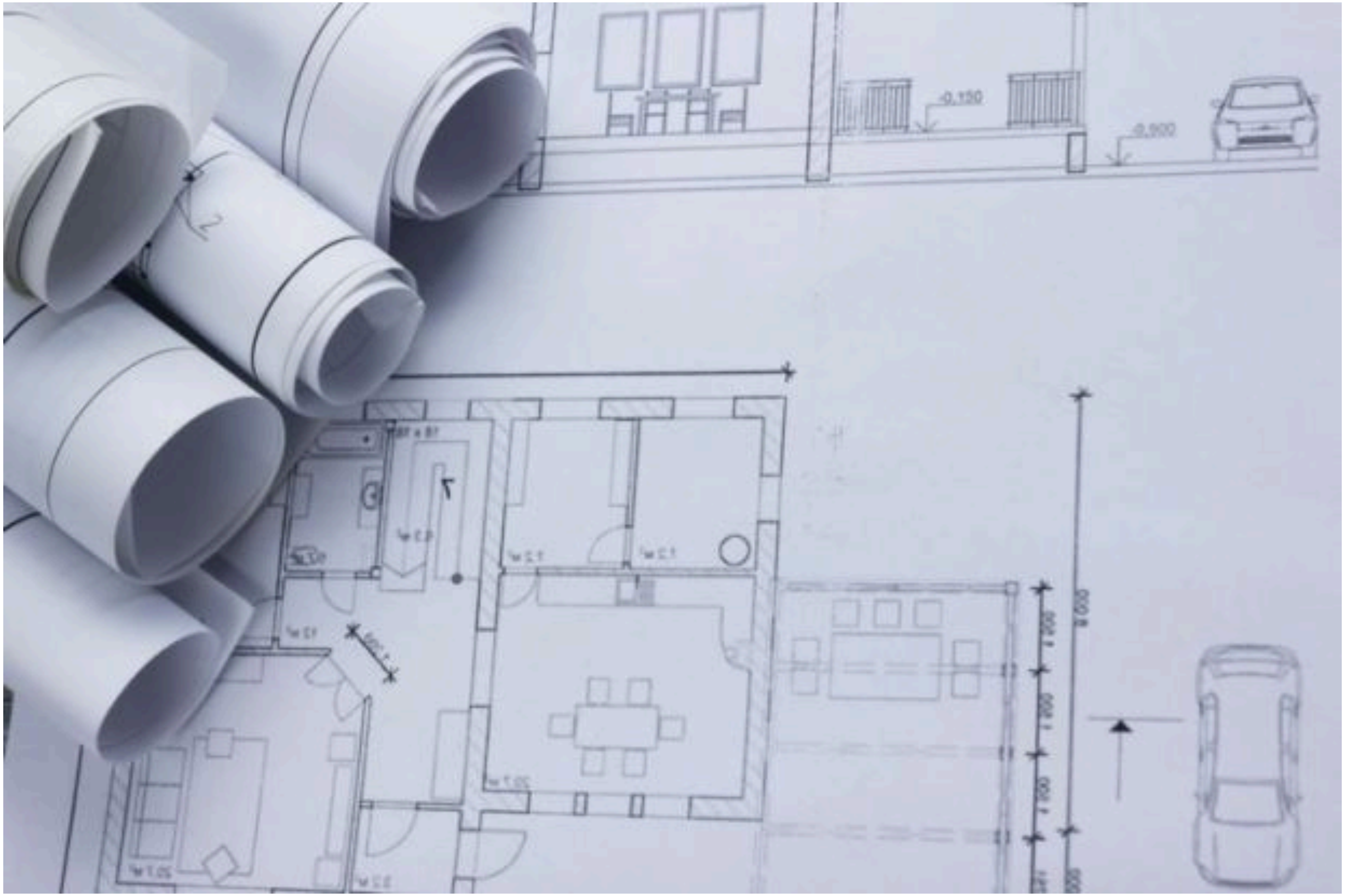


Thinking and Designing Architecture

A LITTLE BIT ABOUT me...

- Skan.ai - chief Architect
- Ai.robotics - chief Architect
- Genpact - solution Architect
- Welldoc - chief Architect
- Microsoft
- Mercedes
- Siemens
- Honeywell

Mubarak

- Arch Requirements (QAW)
    - Context, key fun, quality, constraints, assumption
- Arch Decisions
    - Logical, Deployment, Security, …
- Arch Eval (ATAM, ARID, SAAM)
    - Justification

Part I

Architecture nut and bolts
What is a Architecture Blue print?
How is Architecture different from design ?
What is difference between Application Architecture and Enterprise Architecture ?
What is Role of a Application Architect ?
Application Architect vs Solution Architect
Application Architect vs Vertical Architect

Architecture vs Design

Quality

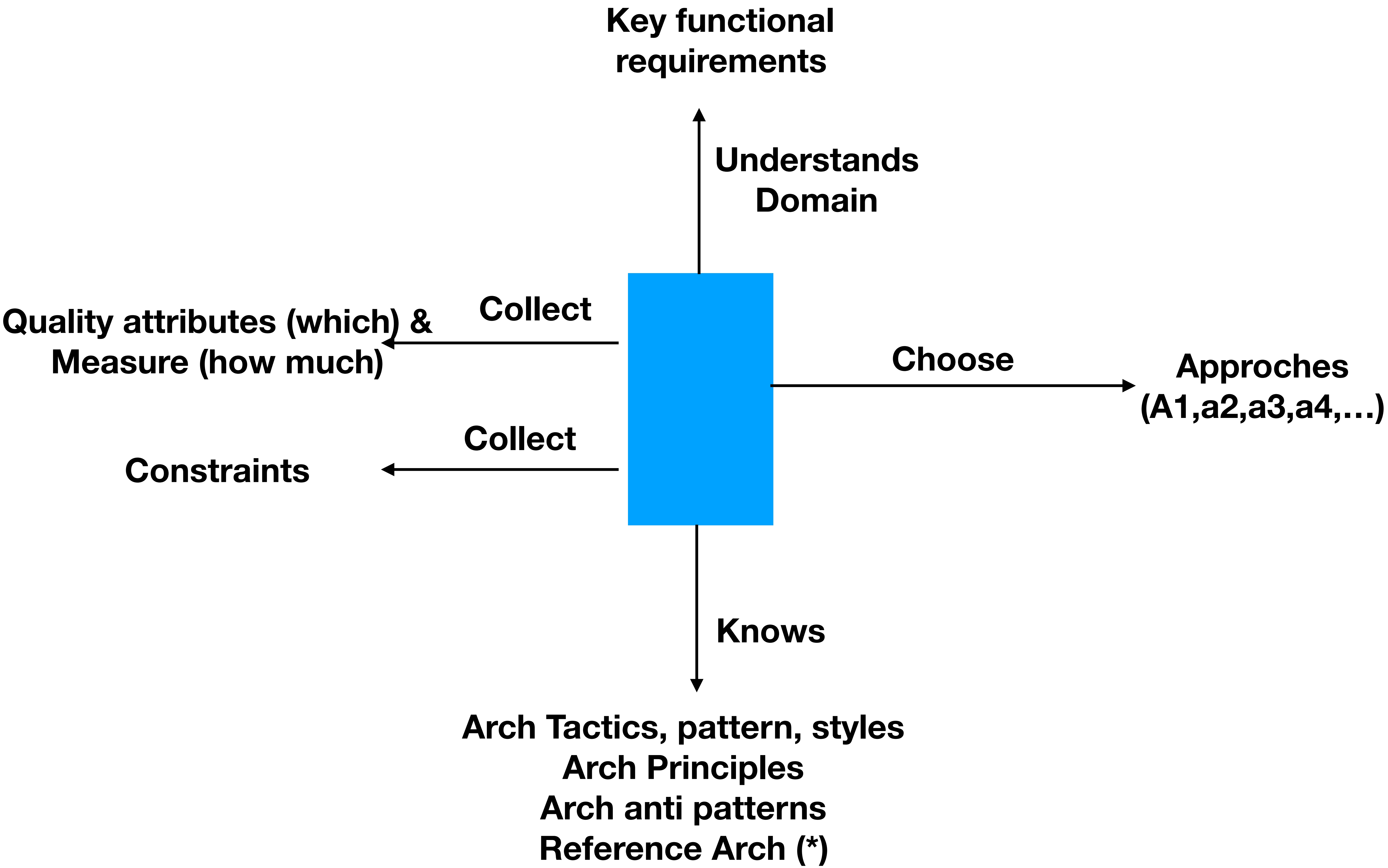
- Performance (Resource)
- Maintainability
- Reliability (Trust)
- Robustness (Rugud)
- Scalability (volume)
- Availability
- Security (Trust)
- Usability

Measure

- tps
- Response time
- Latency
- % uptime
- % downtime
- No of clicks
- Probability
-

Approach

- Versioning
- Modularity
- Concurrency
- Caching
- Lazy loading
- ACID
-



Key functional requirements

Understands Domain

Collect

Quality attributes (which) & Measure (how much)

Collect

Constraints

Choose

Approches (A1,a2,a3,a4,…)

Knows

Arch Tactics, pattern, styles
Arch Principles
Arch anti patterns
Reference Arch (*)

Engineering vs Tuning

Attribute

- Performance <—
- Maintainability <—
- Security (Trust) <—
- Scalability (Volume - I/O, CPU, Memory) <—
- Usability x
- Availability <—
- Reliability (Trust) <—
- Robustness (Rugud)
-

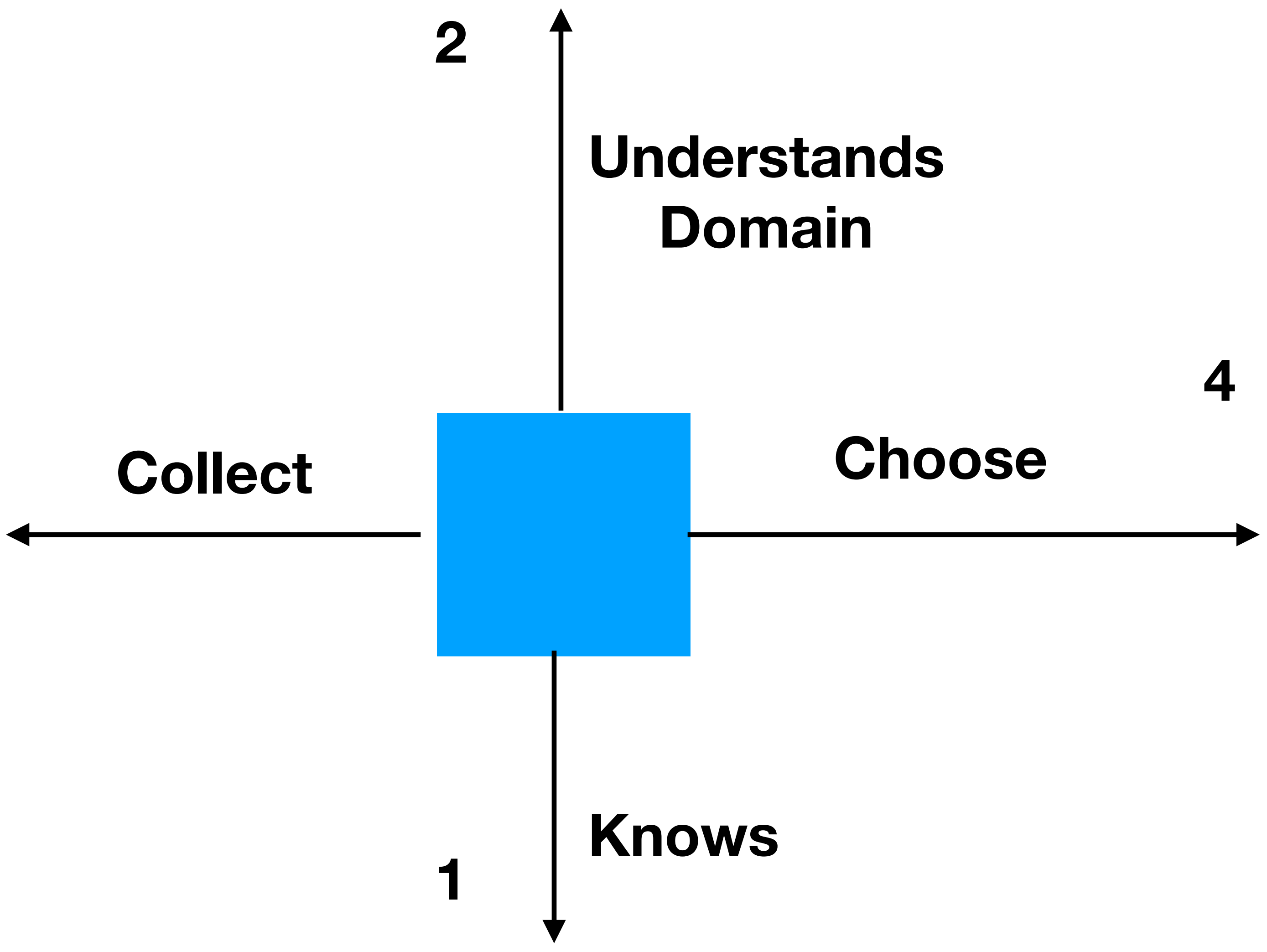
Measure

- Response time
- Memory
- CPU
- I/O
- Latency
- TPS
- …

Approach

- Caching
- Parallel
- Pooling
- Lazy Loading
- Compression
- Chunking
- Reusability/ Extensible
- Modular /Component
- Low Coupling
- EDA
- ACID - transaction
- Input Validation

5 Communicate



Key functional requirements (80:20)

2

Understands Domain

NFR

3

Quality attributes, Measure (Architectural Requirements)

Collect

Choose

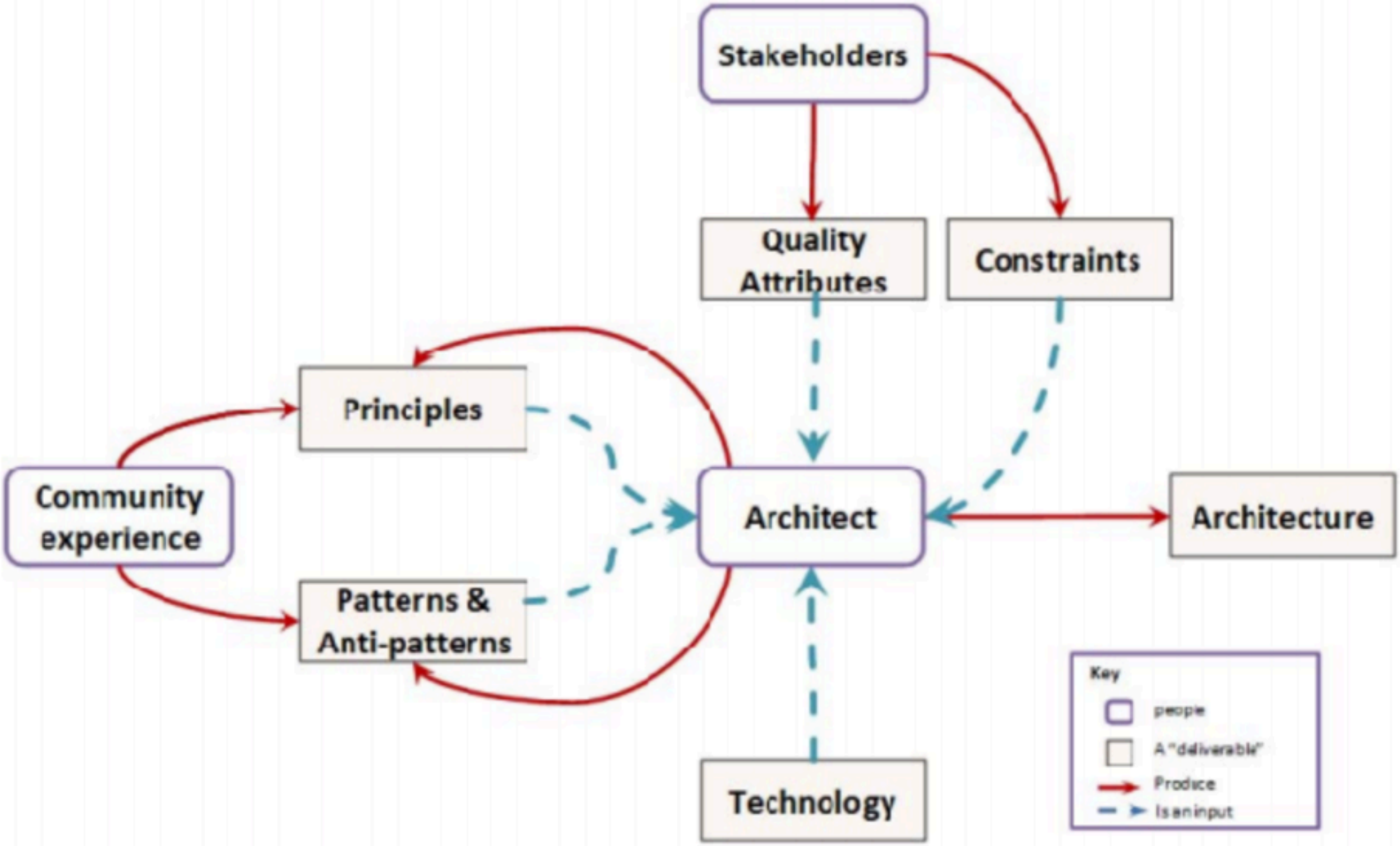
4

Approaches (Architectural Decisions)

Knows

1

Arch Tactics, pattern, styles
Arch Principles
Arch anti patterns
Reference Arch (*)



Stakeholders

Quality Attributes

Constraints

Principles

Community experience

Architect

Architecture

Patterns & Anti-patterns

Technology

Key

people

A "deliverable"

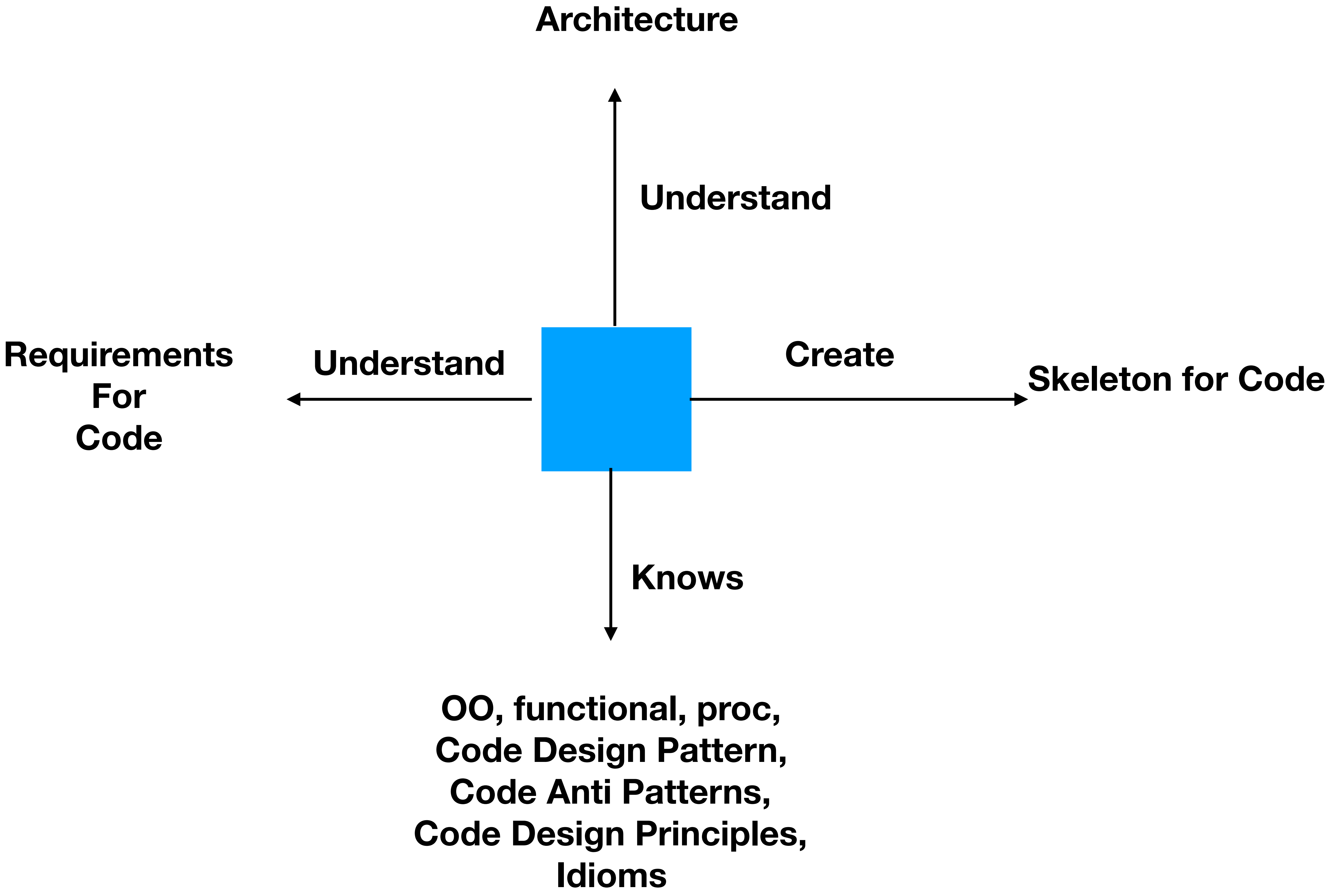
Produce

Is an input

Pre

Engineering vs Tuning

Post



Architecture

Understand

Requirements
For
Code

Understand

Create

Skeleton for Code

Knows

OO, functional, proc,
Code Design Pattern,
Code Anti Patterns,
Code Design Principles,
Idioms

Architecture Design
UI Design
Test Design
Module Design
Class Design
Code Design

Architecture vs Design

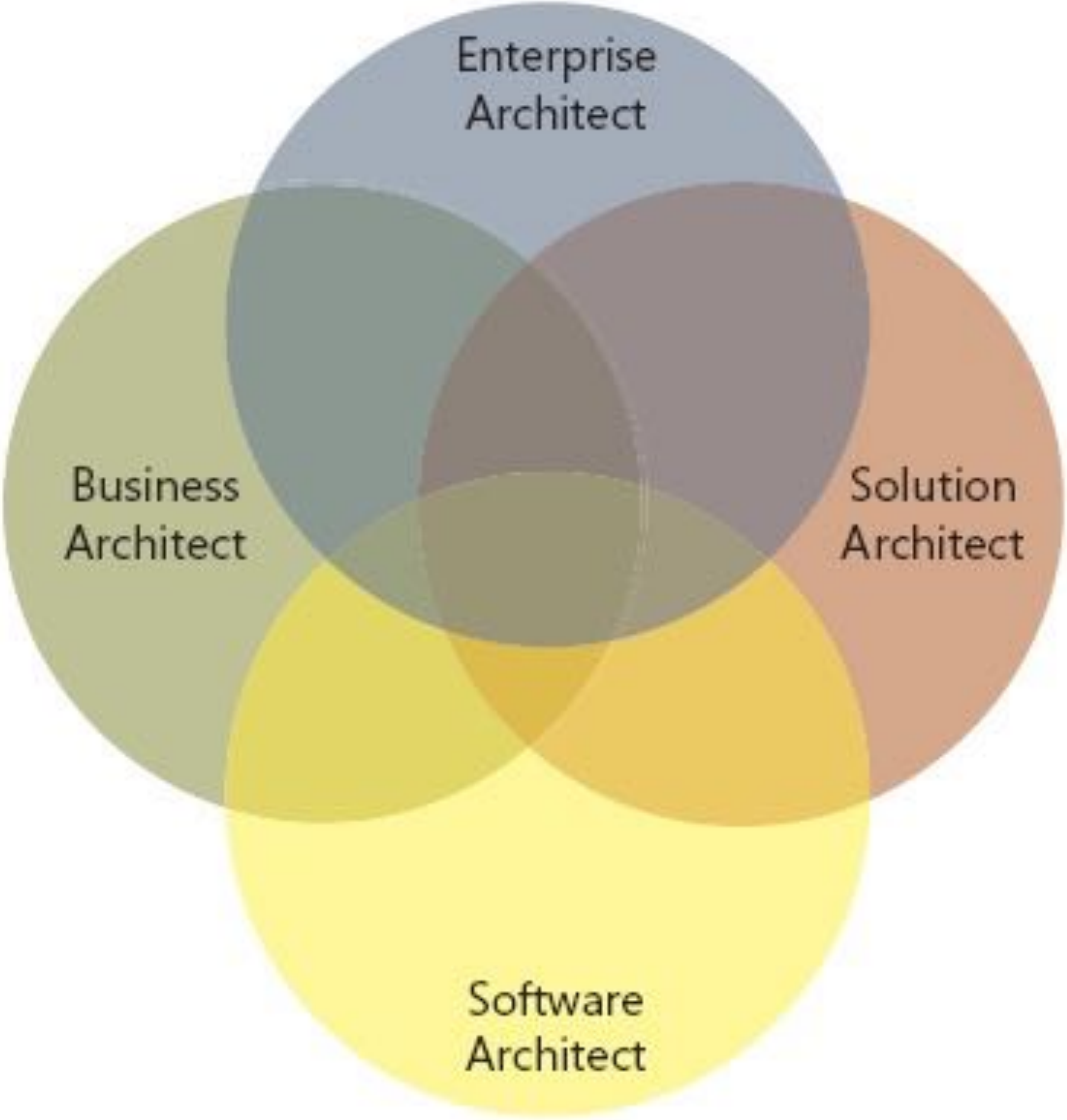
Code Maintainability

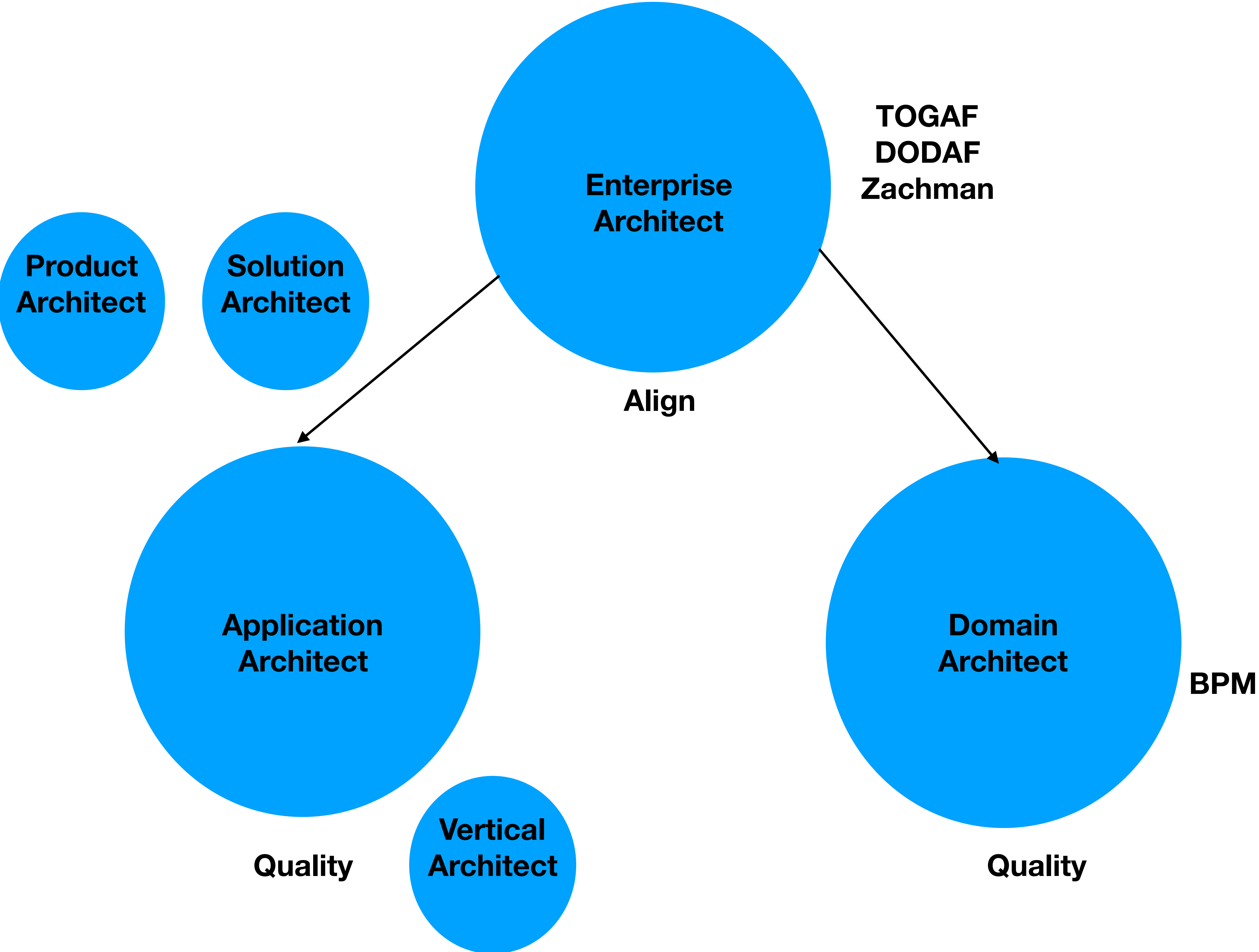
System quality

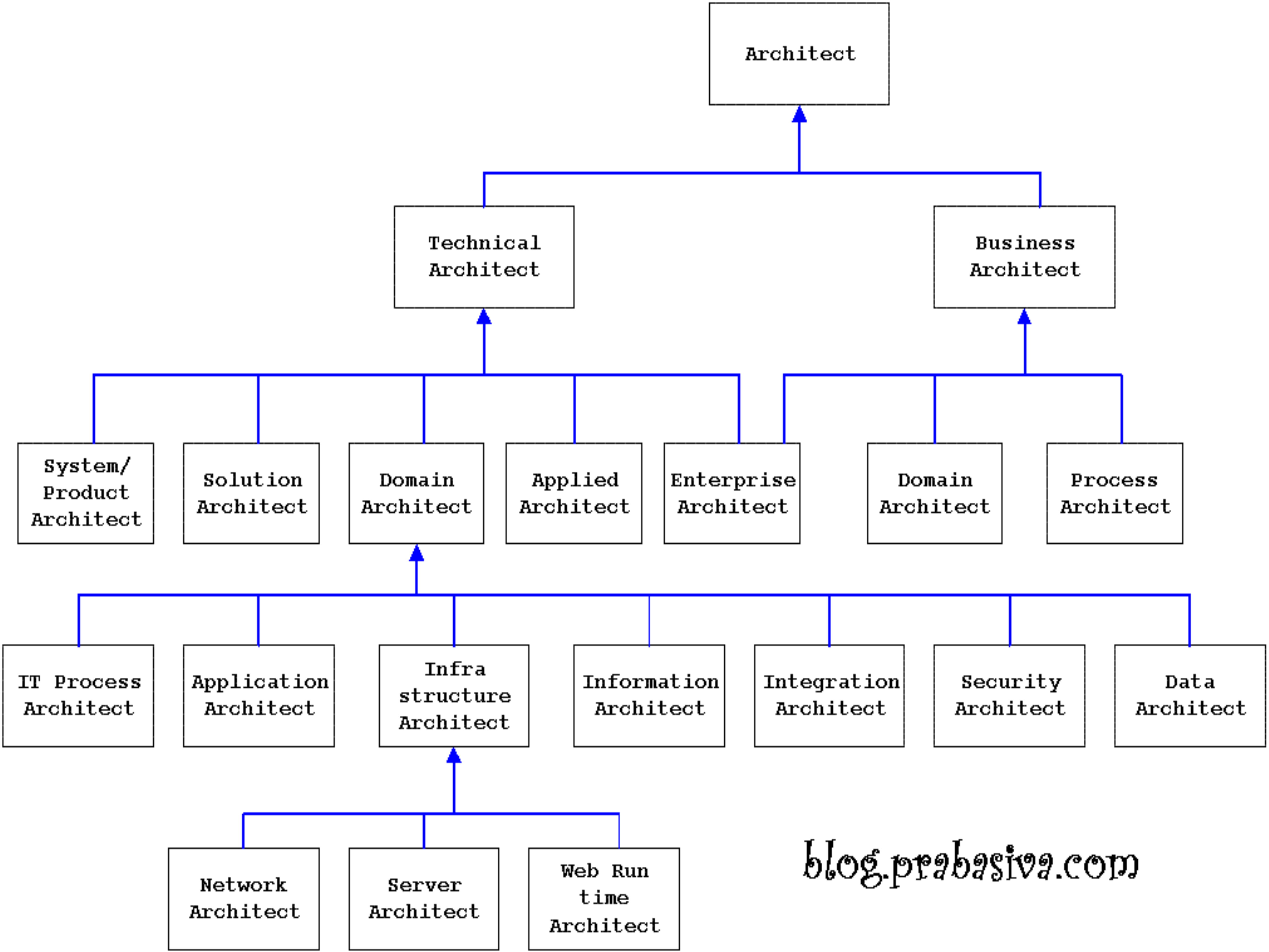
Detail Design
Module Design
Class Design
Low Level Design
Code design
Implementation Design

Types of Architect

Architect Roles

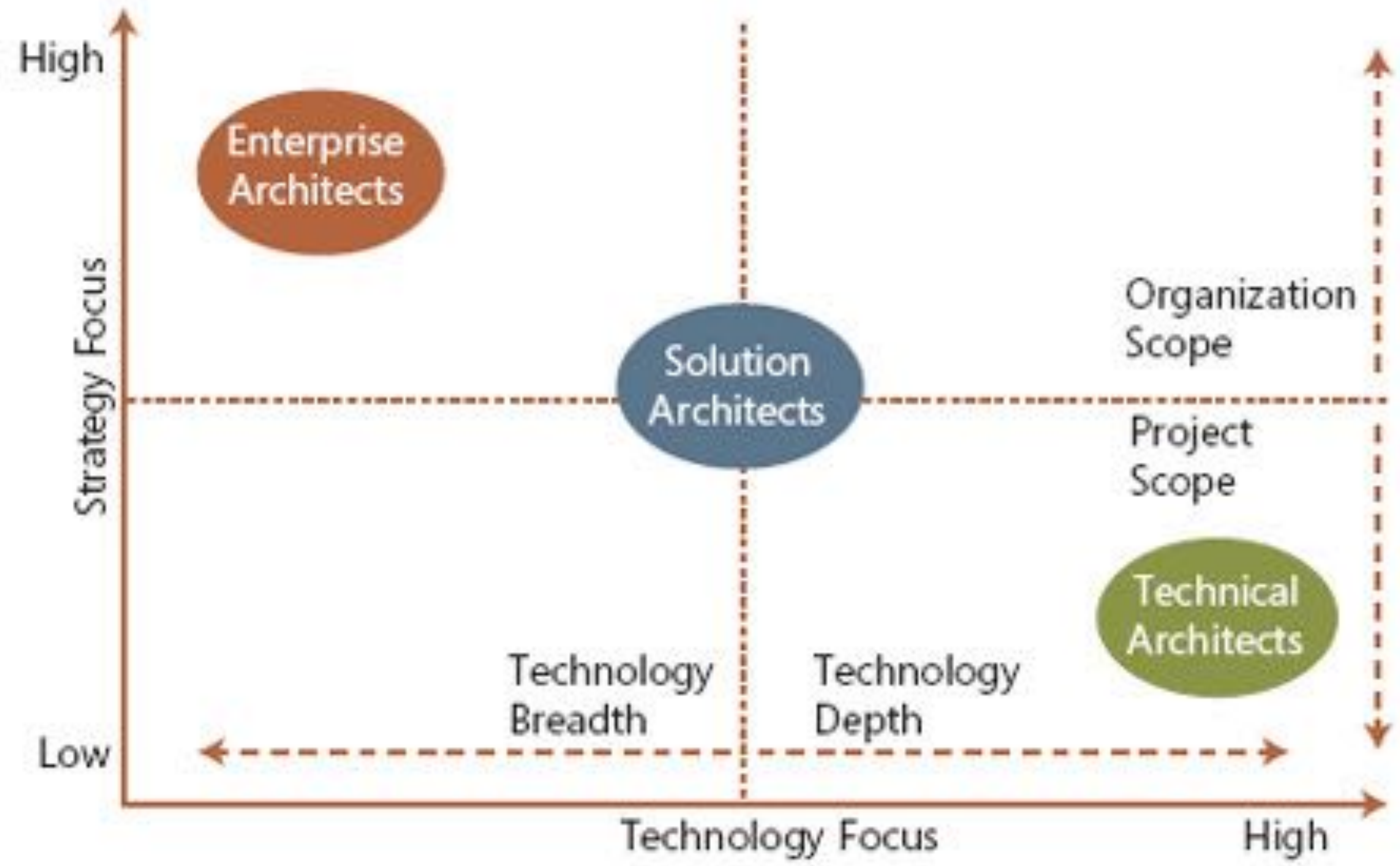



Enterprise Architect
TOGAF
DODAF
Zachman
Product Architect
Solution Architect
Align
Application Architect
Domain Architect
BPM
Quality
Vertical Architect
Quality


Architect
Technical Architect
Business Architect
System/ Product Architect
Solution Architect
Domain Architect
Applied Architect
Enterprise Architect
Domain Architect
Process Architect
IT Process Architect
Application Architect
Infra structure Architect
Information Architect
Integration Architect
Security Architect
Data Architect
Network Architect
Server Architect
Web Run time Architect

	Business	Enterprise	Solution	Application
Soft Skills				
Communicates well with key stakeholders (C-level officers)	#	*	#	#
Communicates well with stakeholders (Director / Manager level)	*	*	*	#
Public Speaking	~	*	~	#
Writing Skills	*	*	~	#
Hard Skills				
Business Process Engineering	*	*	~	#
Programming	#	#	*	*
Requirements Analysis	*	~	~	#
Software Design	#	~	*	*
Scope	Process	Organization	Single Solution	Single Project

Solution Architect



Part II

Collect Architectural Requirements

Understanding Quality of Service -

What is Quality Model

Quality Attribute scenarios

Quality Attribute Workshop (QAW)

Identification of architectural drivers

Implicit and explicit characteristics

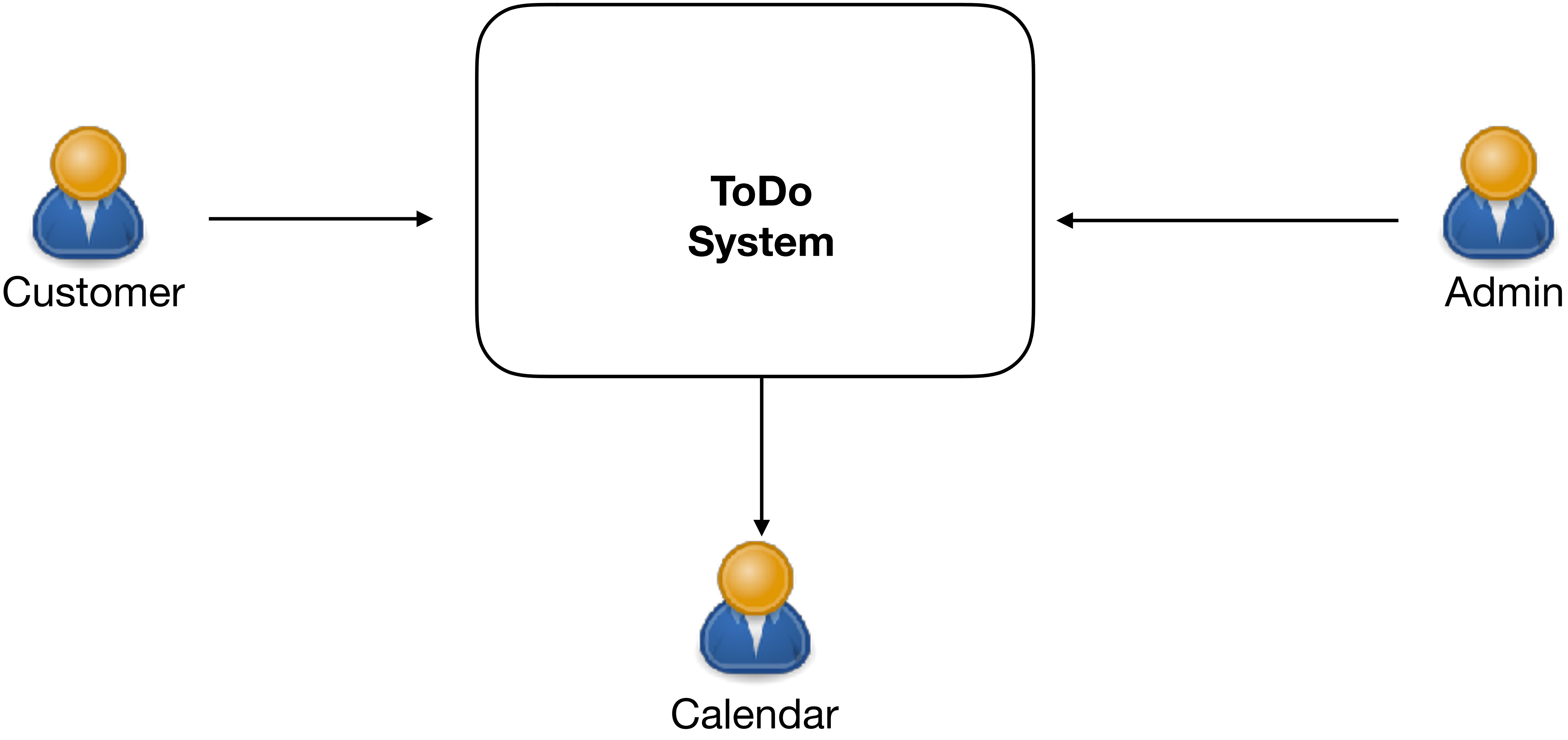
Gathering Performance scenarios

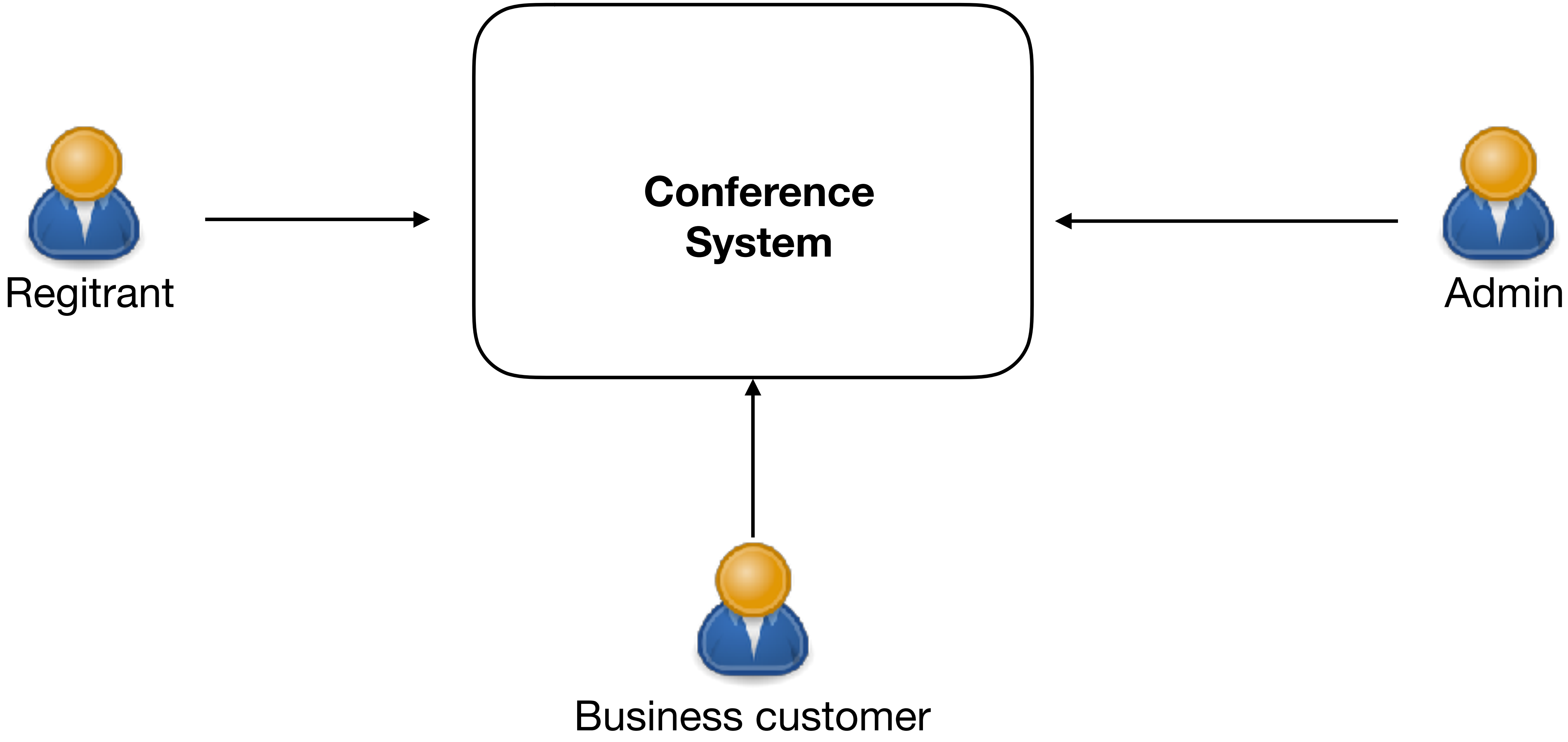
Gathering Maintainability scenarios

Lab : Architecture driven requirements engineering using QAW

Arch Req

1. Context view



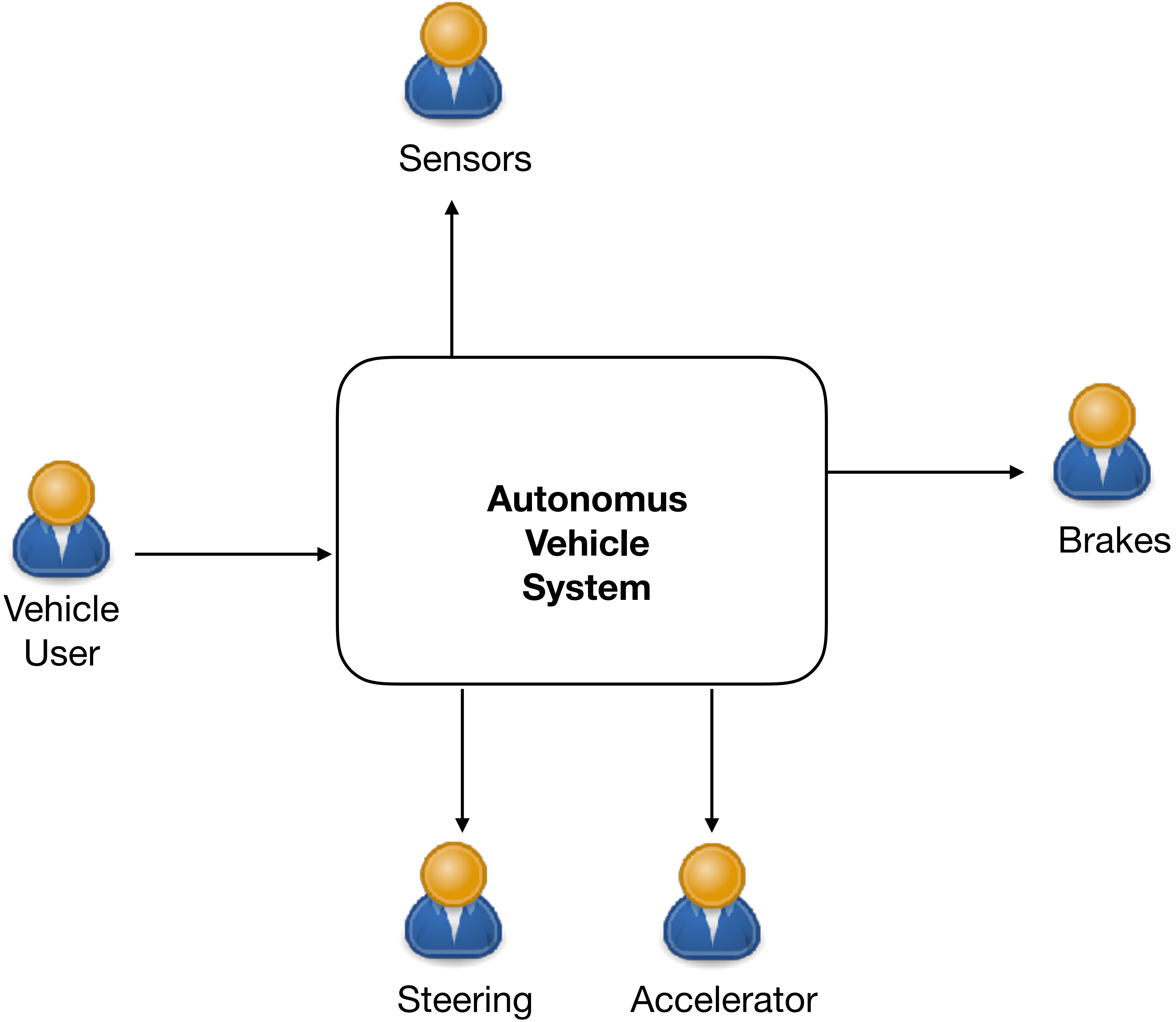


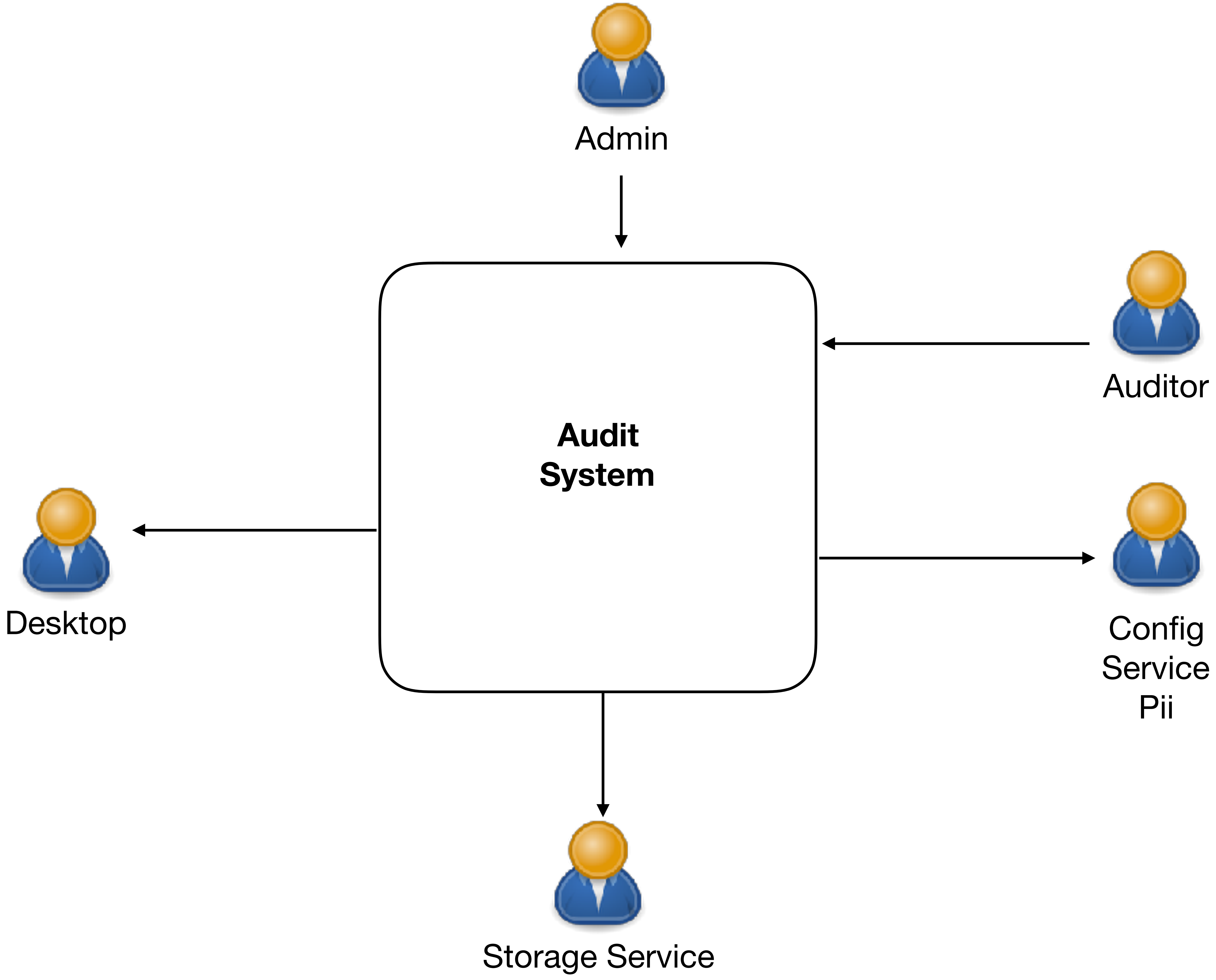
Conference System

Regitrant

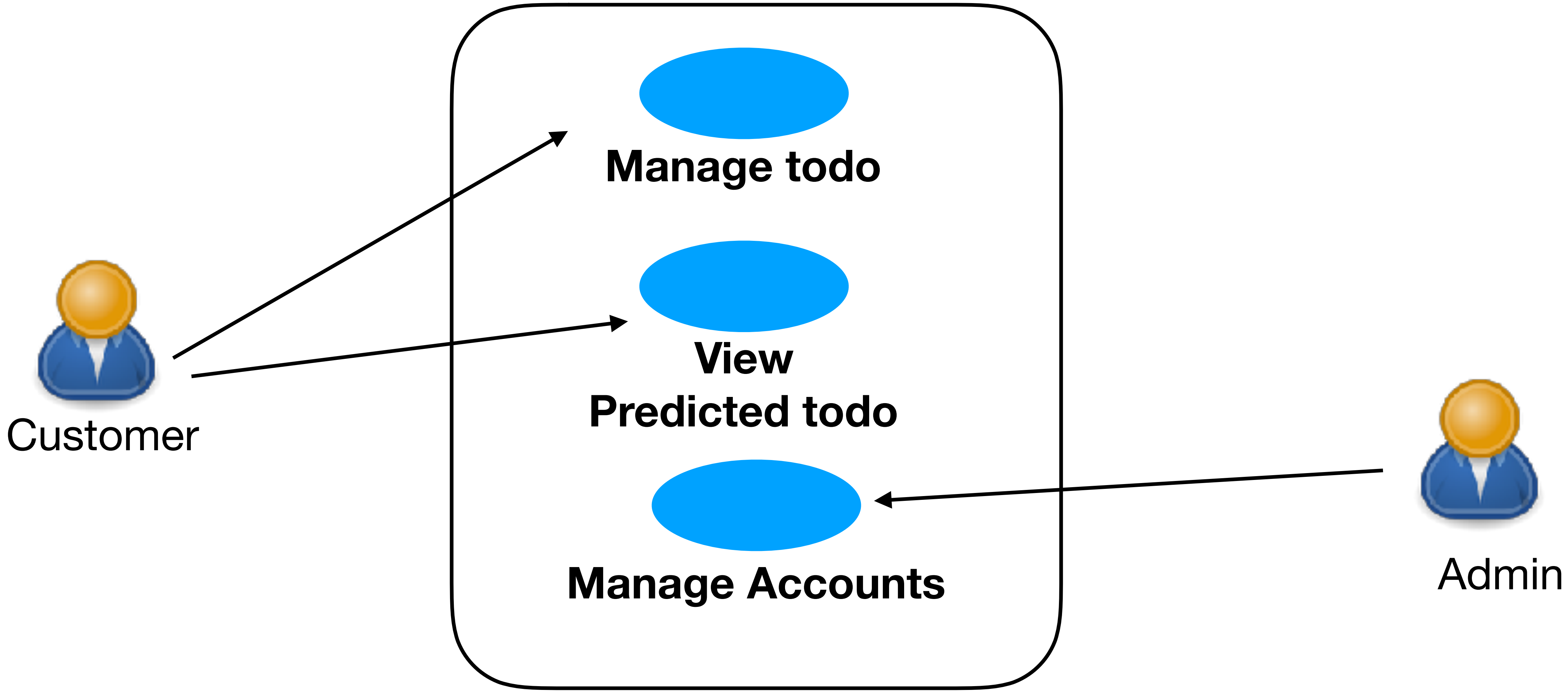
Admin

Business customer


Sensors
Vehicle User
Autonomus Vehicle System
Brakes
Steering
Accelerator


Admin
Auditor
Audit
System
Desktop
Config
Service
Pii
Storage Service

2. Functional view



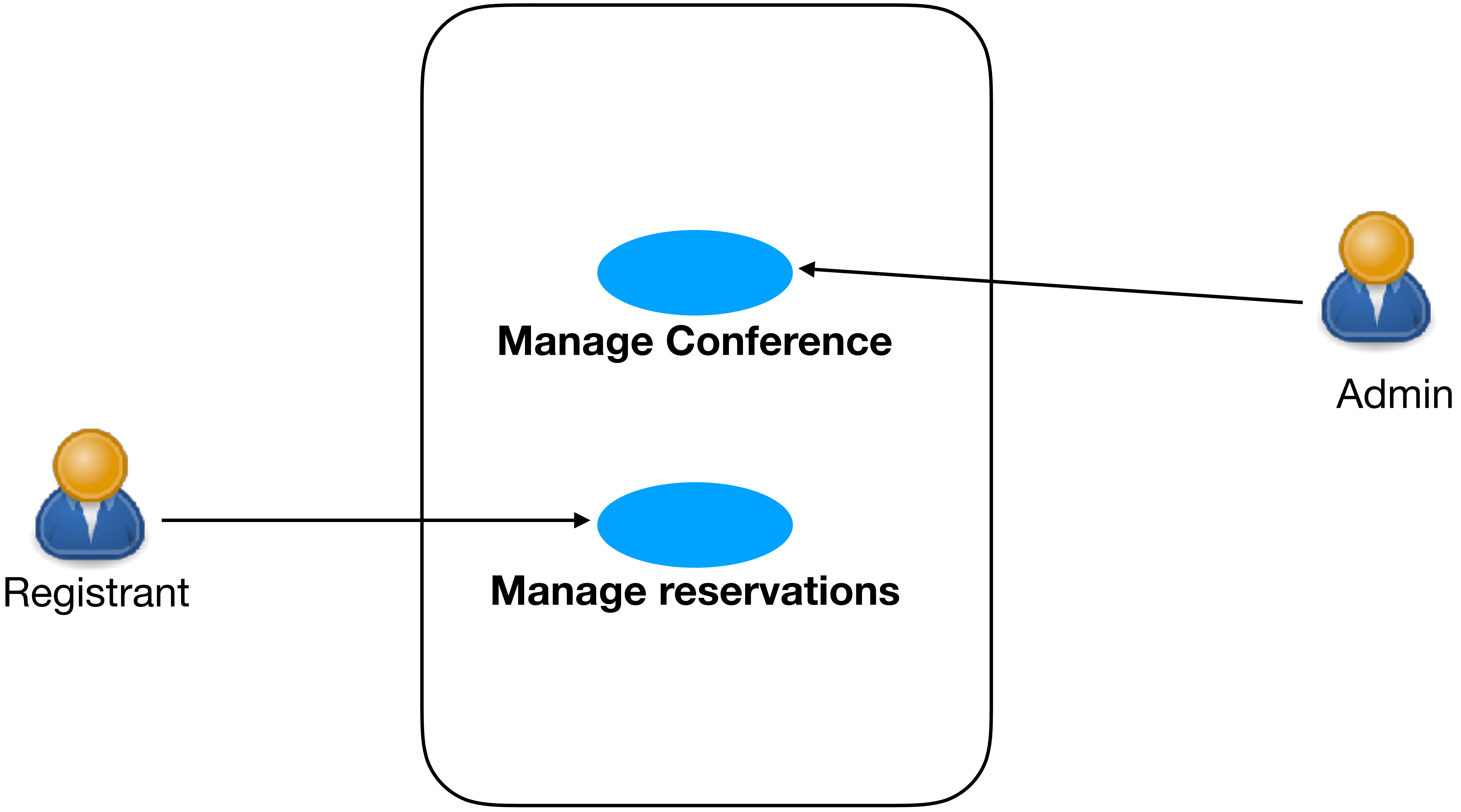
Manage todo

View
Predicted todo

Manage Accounts

Customer

Admin

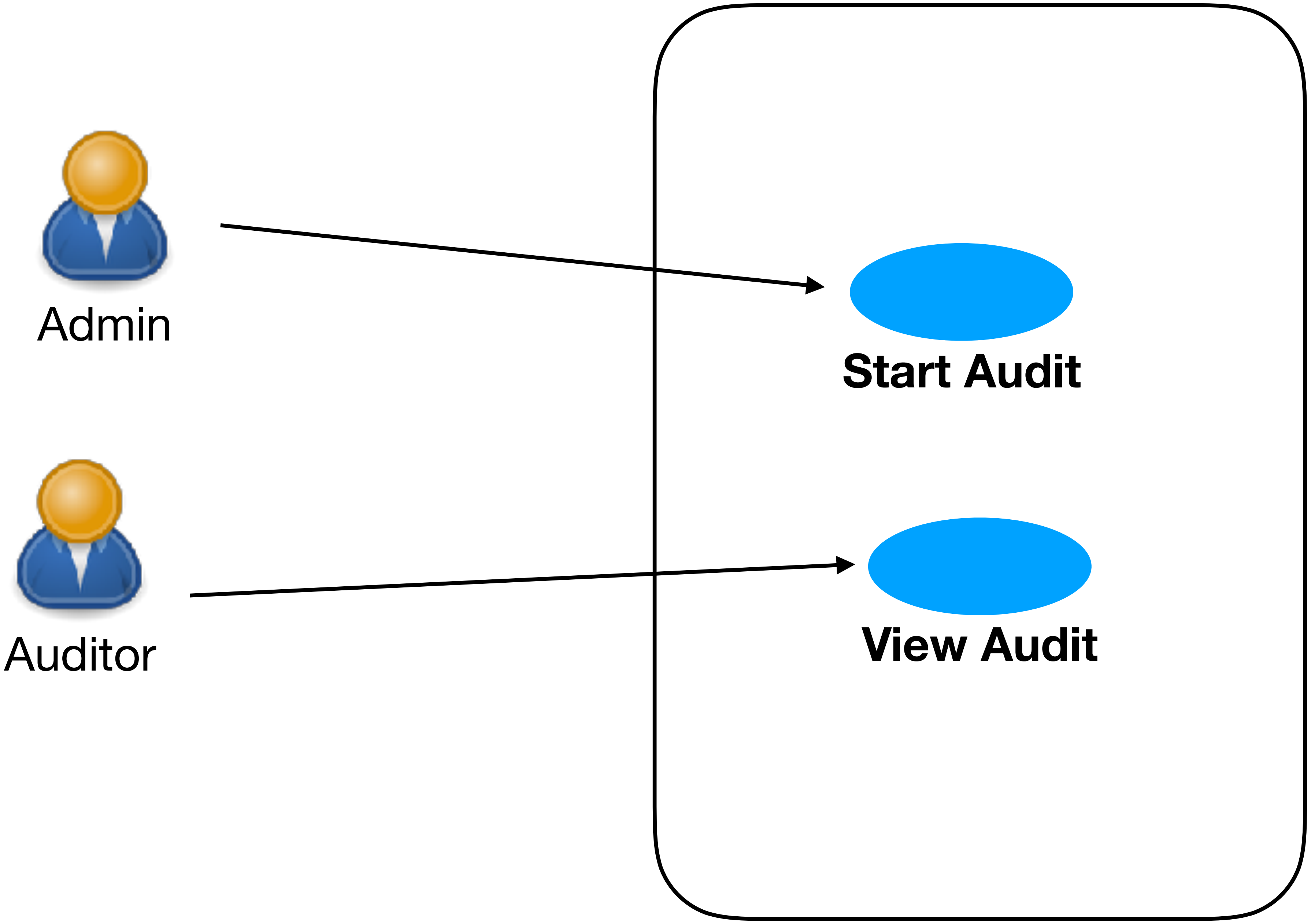


Manage Conference

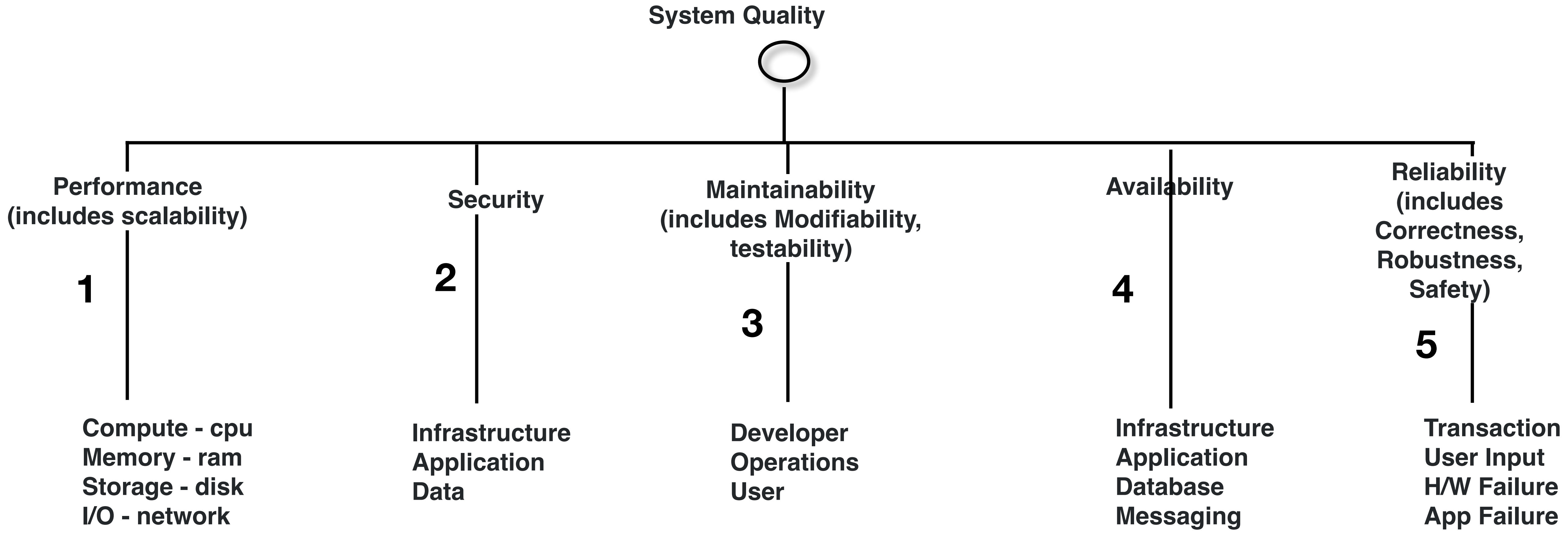
Admin

Registrant

Manage reservations


Admin
Start Audit
Auditor
View Audit

3. Quality View



System Quality

Performance (includes scalability)

1

Compute - cpu
Memory - ram
Storage - disk
I/O - network

Security

2

Infrastructure
Application
Data

Maintainability (includes Modifiability, testability)

3

Developer
Operations
User

Availability

4

Infrastructure
Application
Database
Messaging

Reliability (includes Correctness, Robustness, Safety)

5

Transaction
User Input
H/W Failure
App Failure

# Usability
# Portability
# Interoperability

Quality Model

- IEEE
- ISO
- MacCal
- Bohem
- ..

As a user I want to **add a todo** In the app

As a user I want to **add a todo** In the app . The todo is added In **< 3 secs**

SEI - Software Engineering Institute

Quality Attribute Scenarios

Quality	Source (who)	Stimulus (action)	Artifact (which)	Environment (context)	Response (output)	Measure (scale)
Performance	As a user	I want to add a todo	In the portal	During peak load.	The todo is added	In < 3 secs
Reliability	As a user	I want to add a todo	From the portal	Duplicate request	Duplicate entry is detected	In 100% of cases
Maintainability	Developer	Change the Configuration engine	In the Frame work	During maintenance	The configuration mechanism is replaced	In < 1 week
Security	unknown identity	requests to add a new todo	In the portal	During Normal load.	block access to the data and record the access attempts	100% probability of detecting the attack, 100% probability of denying access
Availability	The processor	Failed	In the db server	During Operational Hours	Secondary is made Primary	In < 5 minutes

BA, product owners, …
Dev, QA, Ops

Seed scenarios
List of scenarios (Today | growth | exploratory)

Source: Processor

Stimulas: Stop working during Peak Hours

Artifact: in the central system

environment: during Peak Hours (8 am - 12 pm)

Response : Notification to operator, system Administrator

….

Measure: not more than 5 minutes to get a new processor running

A processor stops working during Peak Hours in the central system

during Peak Hours, Notification to operator, system Administrator

….. Should … not more than 5 minutes to get a new processor running

Performance

Scenario Portion	Possible Values
Source	Internal to the system, External to the system
Stimulus	Periodic events, sporadic events, Bursty events, stochastic events
Artifact	System, component
Environment	Normal mode; overload mode, Reduced capacity mode, Emergency mode, Peak mode
Response	Process stimuli; change level of service
RespMeasure	Latency, deadline, throughput, jitter, miss rate, data loss

source - “Software architecture in Practice

1. What is the expected response time for each use case?
2. What is the average/max/min expected response time?
3. What resources are being used (CPU, LAN, etc.)?
4. What is the resource consumption?
5. What is the resource arbitration policy?
6. What is the expected number of concurrent sessions?
7. Are there any particularly long computations that occur when a certain state is true?
8. Are server processes single or multithreaded?
9. Is there sufficient network bandwidth to all nodes on the network?
10. Are there multiple threads or processes accessing a shared resource? How is access managed?
11. Will bad performance dramatically affect usability?
12. Is the response time synchronous or asynchronous?
13. What is the expected batch cycle time?
14. How much can performance vary based on the time of day, week, month, or system load?
15. What is the expected growth of system load?
16. Increased network bandwidth consumption?
17. Increased database server processing, resulting in reduced throughput.
18. Increased memory consumption, excessive cache misses
19. Increased client response time, reduced throughput, and server resource over utilization.

source - msdn

Modifiability

Portion of Scenario	Possible Values
Source	End user, developer, system administrator
Stimulus	Wishes to add/delete/modify/vary functionality, quality attribute, capacity or technology
Artifact	Code, Data, Components, Configurations, Interfaces, Resources
Environment	At runtime, compile time, build time, design time
Response	Locates places in architecture to be modified; makes modification without affecting other functionality; tests modification; deploys modification
Response Measure	Cost in effort, Cost in money, Cost in time,Cost in number, size, complexity of affected artifacts, Extent affects other system functions or qualities, New defects introduce

source - “Software architecture in Practice

1. Is this system the start of a new product line?
2. Will other systems be built that more or less match the characteristics of the system under construction? If so, what components will be reused in those systems?
3. What Changes could impact components, services, features, and interfaces when adding or changing the functionality, fixing errors, and meeting new business requirements ?
4. Does business rule values change frequently ?
5. Does Business process change frequently ?
6. What existing components are available for reuse?
7. Are there existing frameworks or other code assets that can be reused?
8. Will other applications reuse the technical infrastructure that is created for this application?
9. What are the associated costs, risks, and benefits of building reusable components?
10. Excessive dependencies between components and layers
11. The use of direct communication prevents changes to the physical deployment of components and layers.
12. Reliance on custom implementations of features such as authentication and authorization prevents reuse.
13. The logic code of components and segments is not cohesive.
14. The code base is large, unmanageable, fragile, or over complex.
15. The existing code does not have an automated regression test suite.
16. Lack of documentation may hinder usage, management, and future upgrades.

source - msdn

Testability

Scenario Portion	Possible Values
Source	Unit tester, Integration tester, System tester, Acceptance tester, End user, Automated testing tools
Stimulus	Analysis, architecture, design, class, subsystem integration, system delivered
Artifact	Portion of the system being tested(Piece of design, piece of code, complete system)
Environment	Design time, Development time, Compile time, Integration time, Deployment time, Run time
Response	Execute test suite & capture results, Capture cause of fault, Control & monitor state of the system
RespMeasure	Effort to find fault, Effort to achieve coverage %,Probability of fault being revealed by next test, Time to perform tests, Effort to detect faults, Length of longest dependency chain, Time to prepare test environment, Reduction in risk exposure

source - “Software architecture in Practice

1. Are there tools, processes, and techniques in place to test language classes, components, and services?
2. Are there hooks in the frameworks to perform unit tests?
3. Can automated testing tools be used to test the system?
4. Can the system run in a debugger?
5. Complex applications with many processing permutations are not tested consistently, perhaps because automated or granular testing cannot be performed if the application has a monolithic design.
6. Lack of test planning.
7. Poor test coverage, for both manual and automated tests.
8. Input and output inconsistencies; for the same input, the output is not the same and the output does not fully cover the output domain even when all known variations of input are provided.

source - msdn

4. Constraints view

It should work on IE 11, Chrome and Firefox

Should work on windows, unix, Mac platforms

Should deploy on Azure Cloud

5. Assumptions View

1 Will use open source Technologies only

2

3

Part III

Building Architecture

Understanding Types of Applications

# Client, Native, Web, background

# Server, API, Backend

# Brokered

Decomposing system into Applications

# layered, Hexagonal

# pipes and filters

# Micro kernel

# Component based

# Micro service

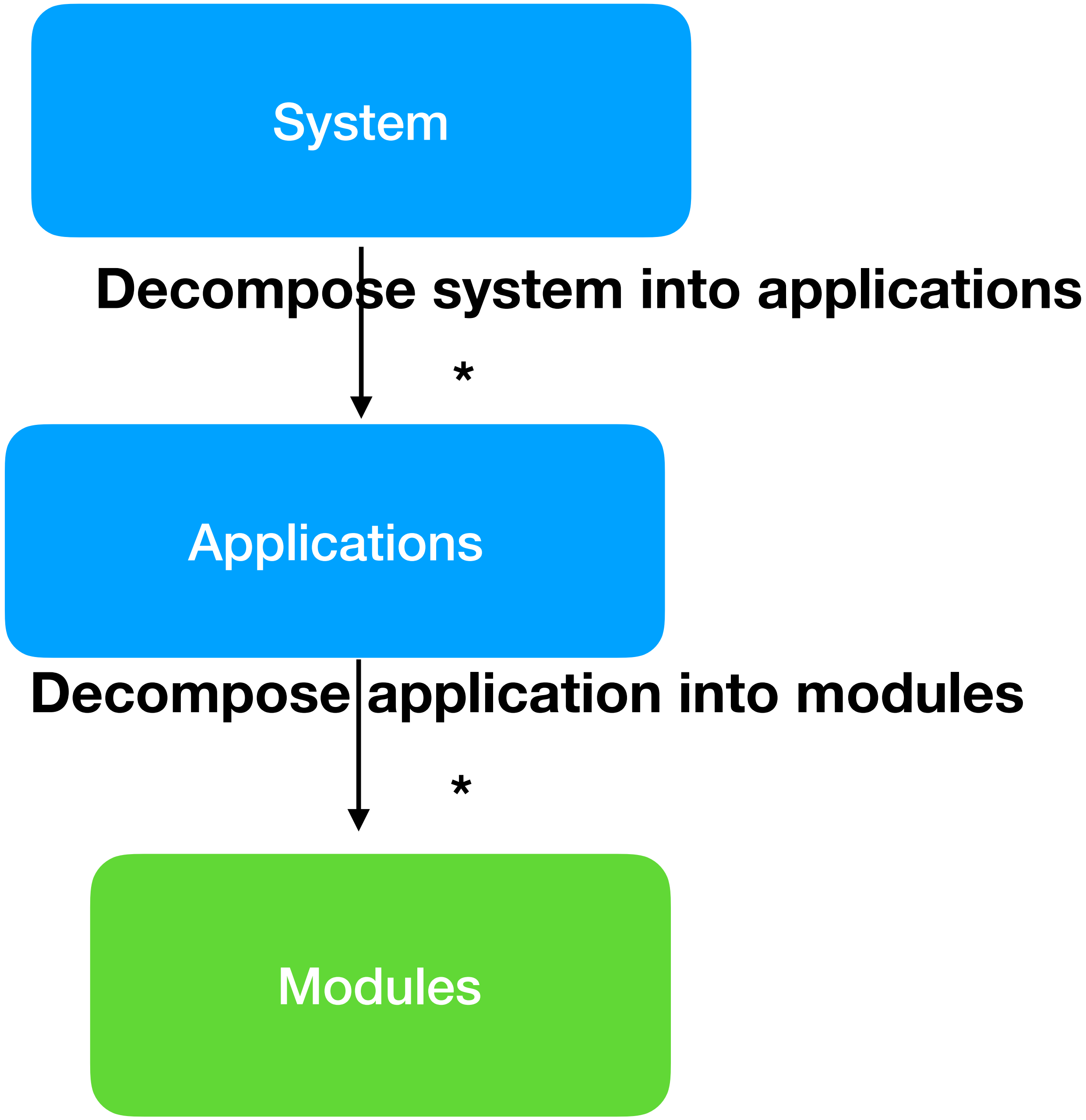
# Event driven Architecture

# MVC, MVP, MVVM

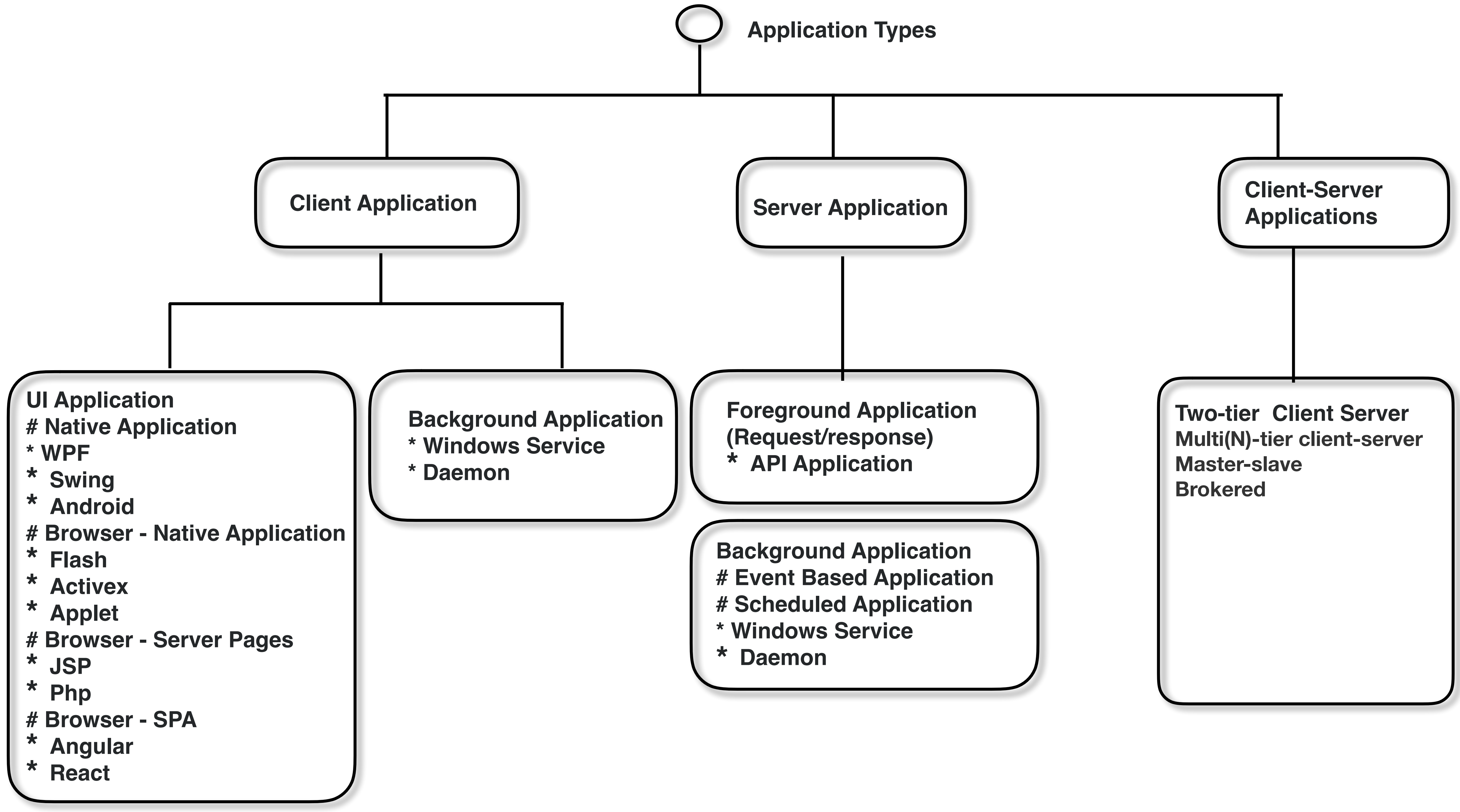
Arch Design

Decisions

Logical View


System
Decompose system into applications
*
Applications
Decompose application into modules
*
Modules

Step 1. Decompose the system into Applications/Tiers



Application Types

Client Application

Server Application

Client-Server Applications

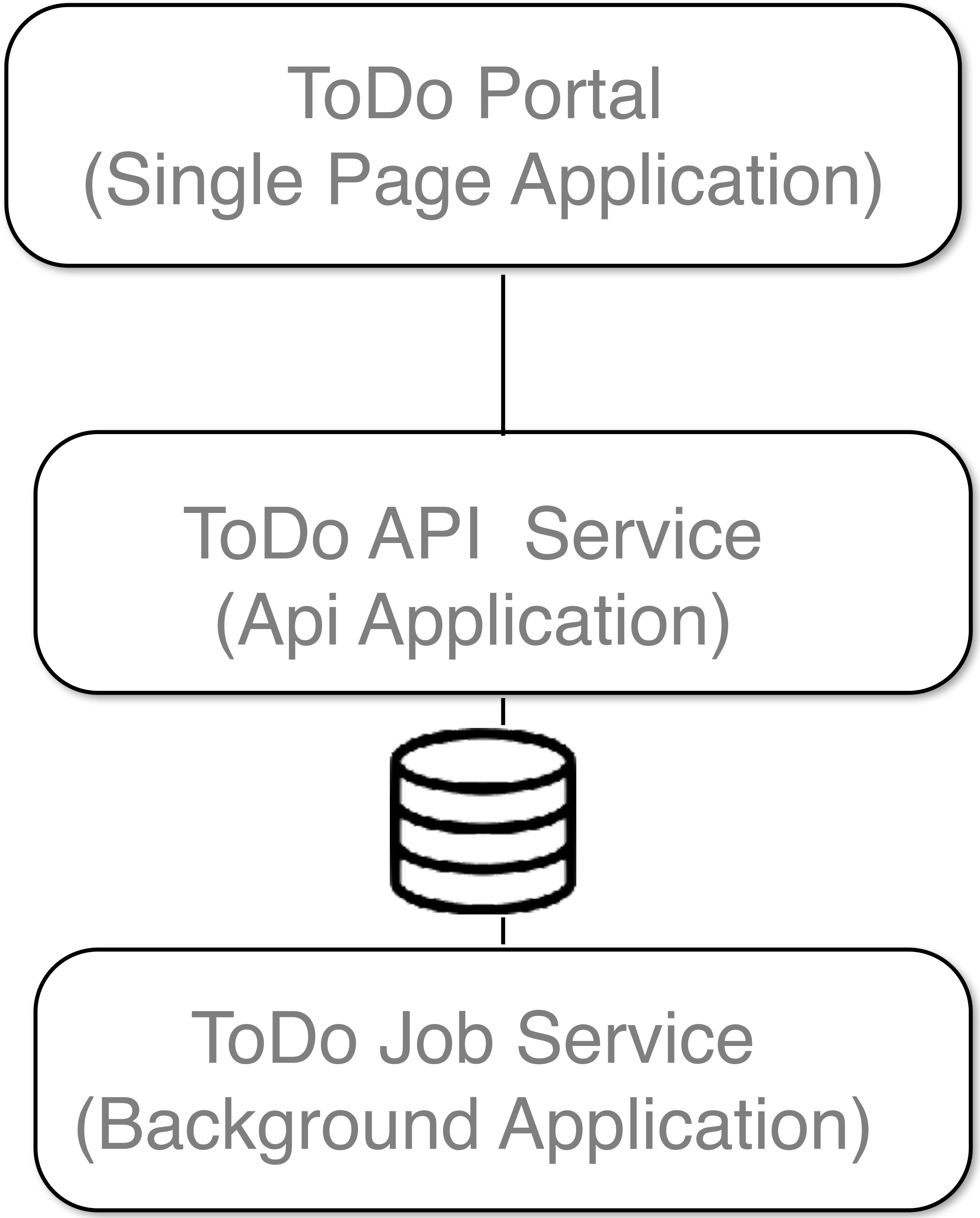
UI Application
# Native Application
* WPF
* Swing
* Android
# Browser - Native Application
* Flash
* Activex
* Applet
# Browser - Server Pages
* JSP
* Php
# Browser - SPA
* Angular
* React

Background Application
* Windows Service
* Daemon

Foreground Application
(Request/response)
* API Application

Background Application
# Event Based Application
# Scheduled Application
* Windows Service
* Daemon

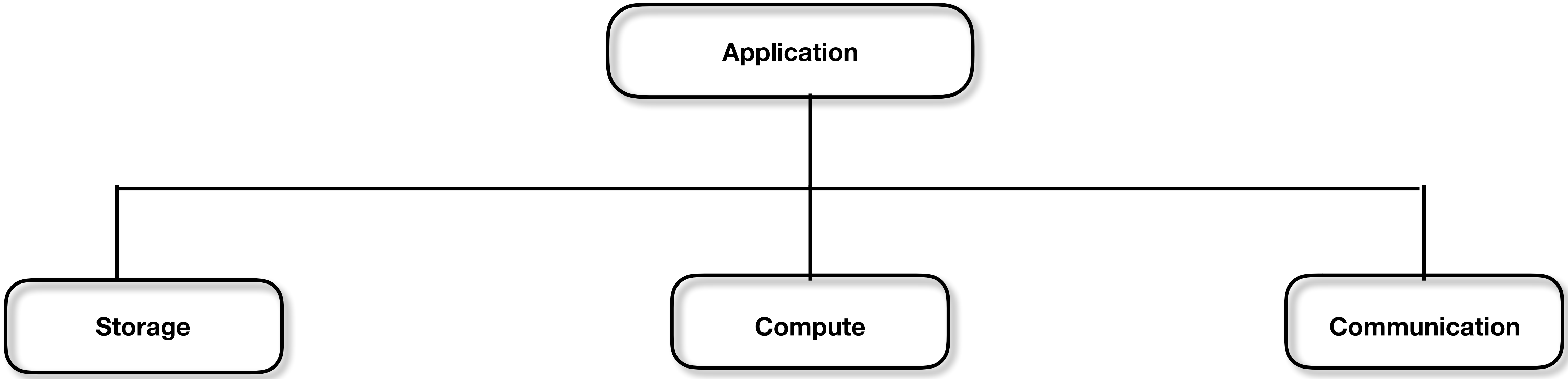
Two-tier Client Server
Multi(N)-tier client-server
Master-slave
Brokered



ToDo Portal
(Single Page Application)

ToDo API Service
(Api Application)

ToDo Job Service
(Background Application)

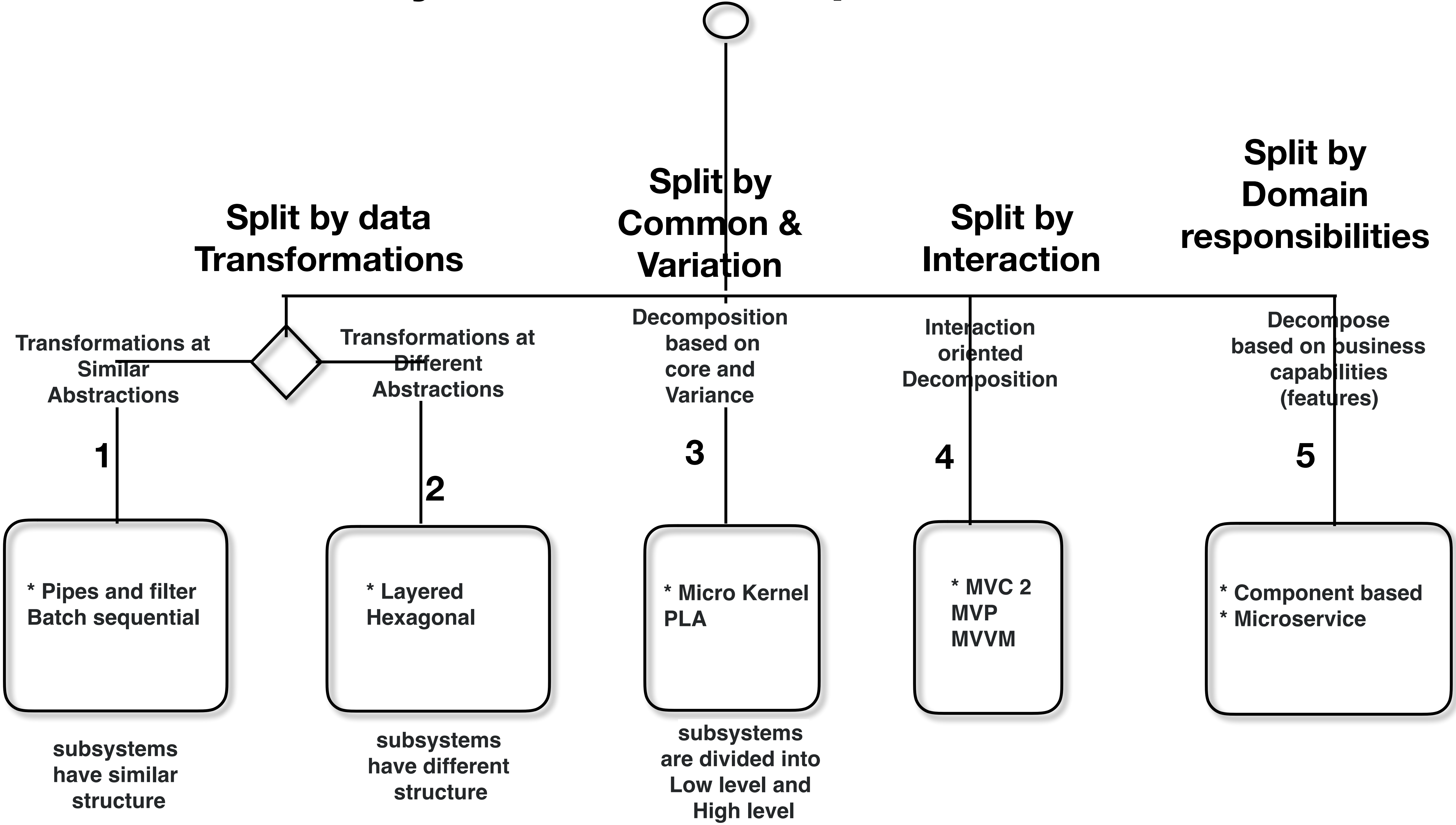

Application

Storage

Compute

Communication

Choose System Decomposition Patterns



Split by data Transformations

Transformations at Similar Abstractions

1

* Pipes and filter
Batch sequential

subsystems have similar structure

Transformations at Different Abstractions

2

* Layered
Hexagonal

subsystems have different structure

Split by Common & Variation

Decomposition based on core and Variance

3

* Micro Kernel
PLA

subsystems are divided into Low level and High level

Split by Interaction

Interaction oriented Decomposition

4

* MVC 2
MVP
MVVM

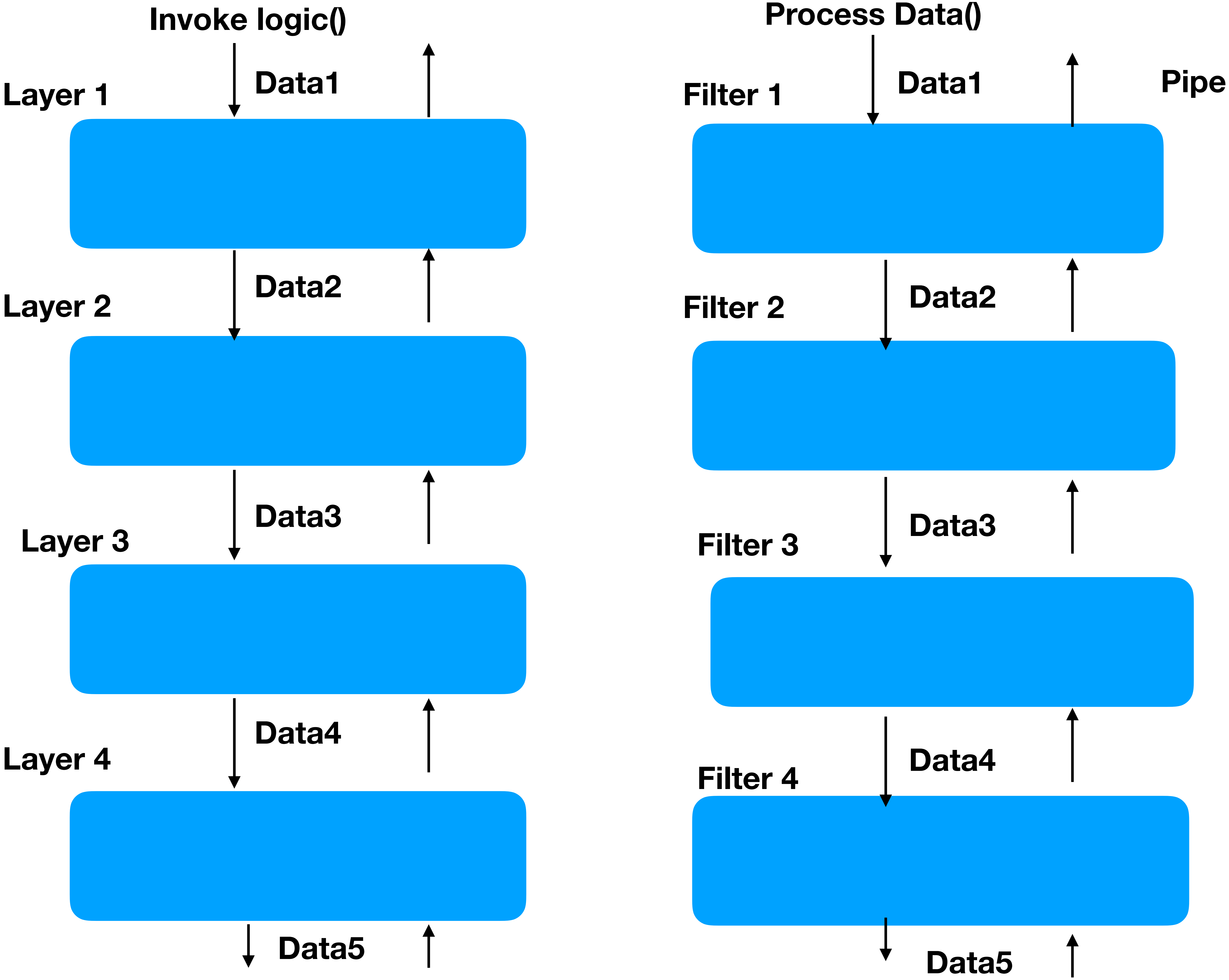
Split by Domain responsibilities

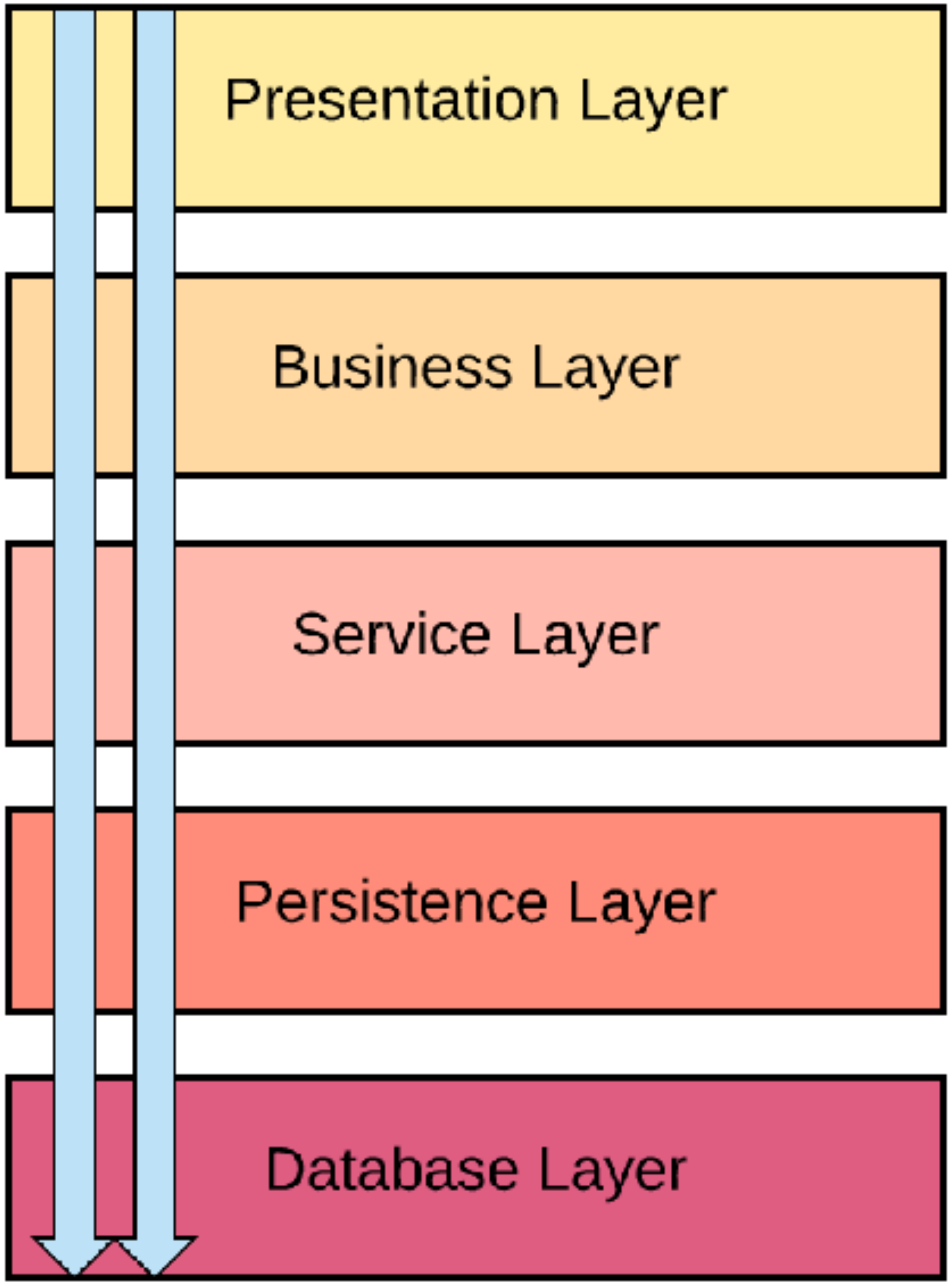
Decompose based on business capabilities (features)

5

* Component based
* Microservice

Actor *


Invoke logic()
Layer 1
Data1
Layer 2
Data2
Layer 3
Data3
Layer 4
Data4
Data5
Process Data()
Filter 1
Data1
Pipe
Filter 2
Data2
Filter 3
Data3
Filter 4
Data4
Data5



Presentation Layer

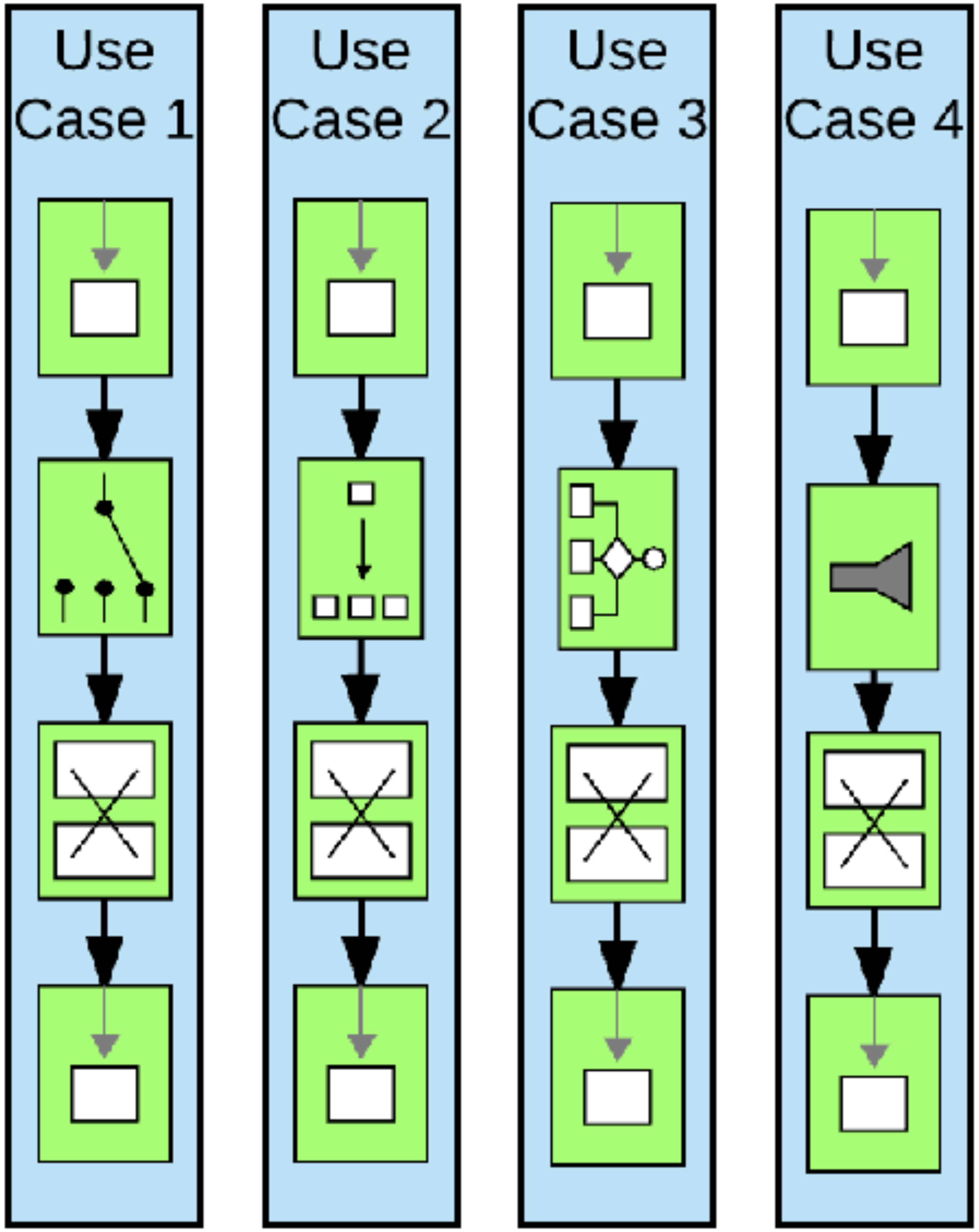
Business Layer

Service Layer

Persistence Layer

Database Layer

Layered Architecture



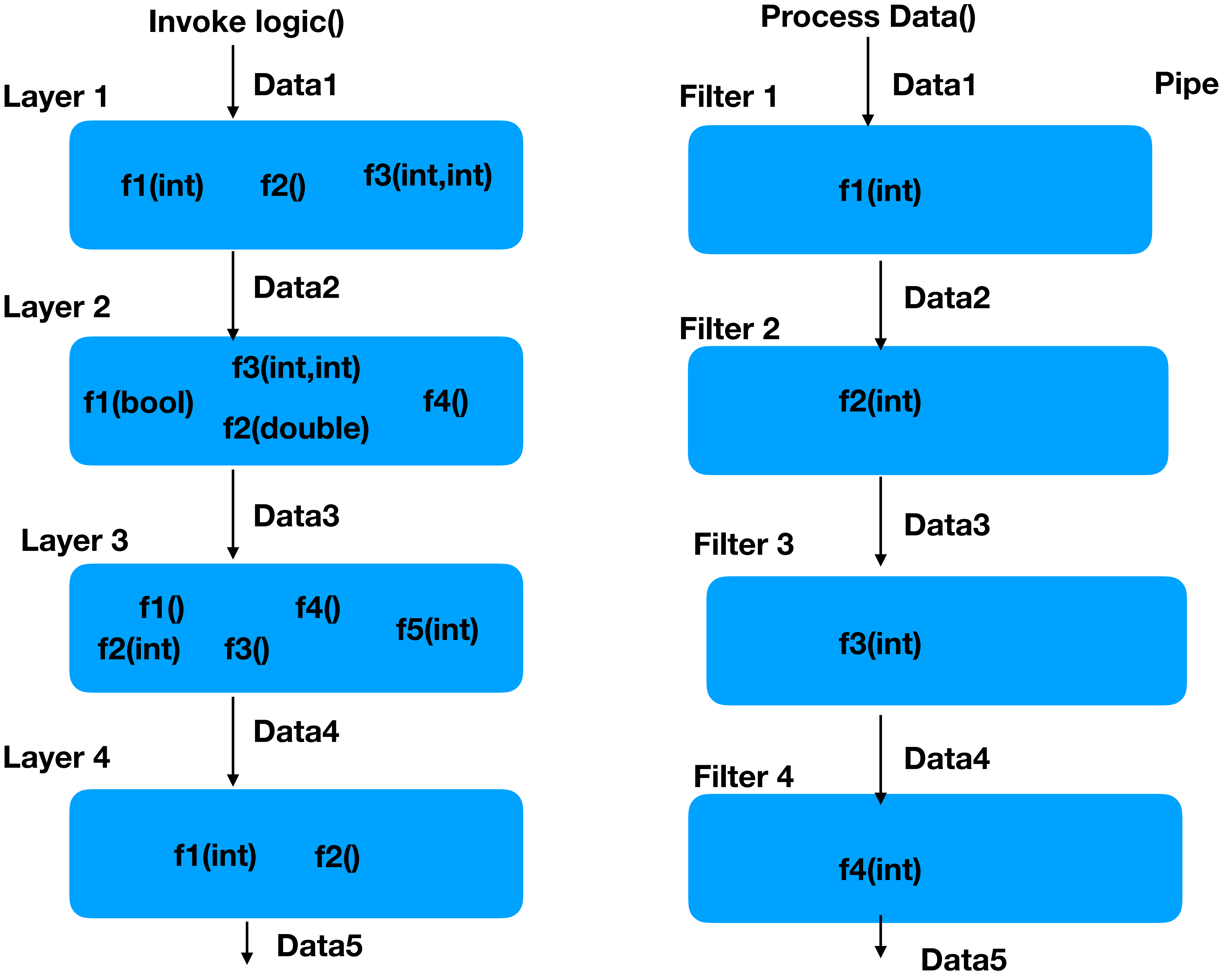
Use Case 1

Use Case 2

Use Case 3

Use Case 4

Pipes and Filters



Invoke logic()

Layer 1

Data1

f1(int) f2() f3(int,int)

Data2

Layer 2

f3(int,int)

f1(bool) f4()

f2(double)

Data3

Layer 3

f1() f4()

f2(int) f3() f5(int)

Data4

Layer 4

f1(int) f2()

Data5

Process Data()

Filter 1

Data1

Pipe

f1(int)

Data2

Filter 2

f2(int)

Data3

Filter 3

f3(int)

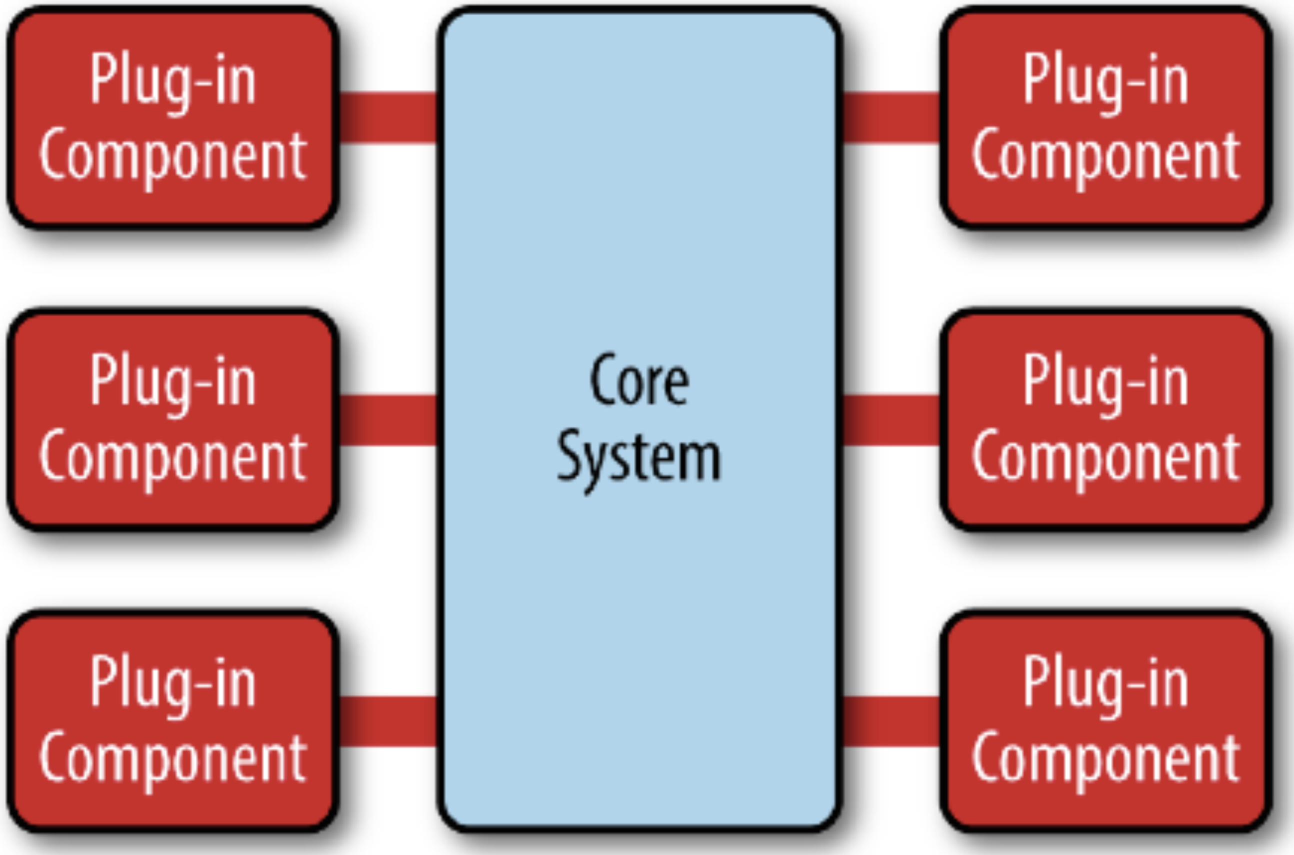
Data4

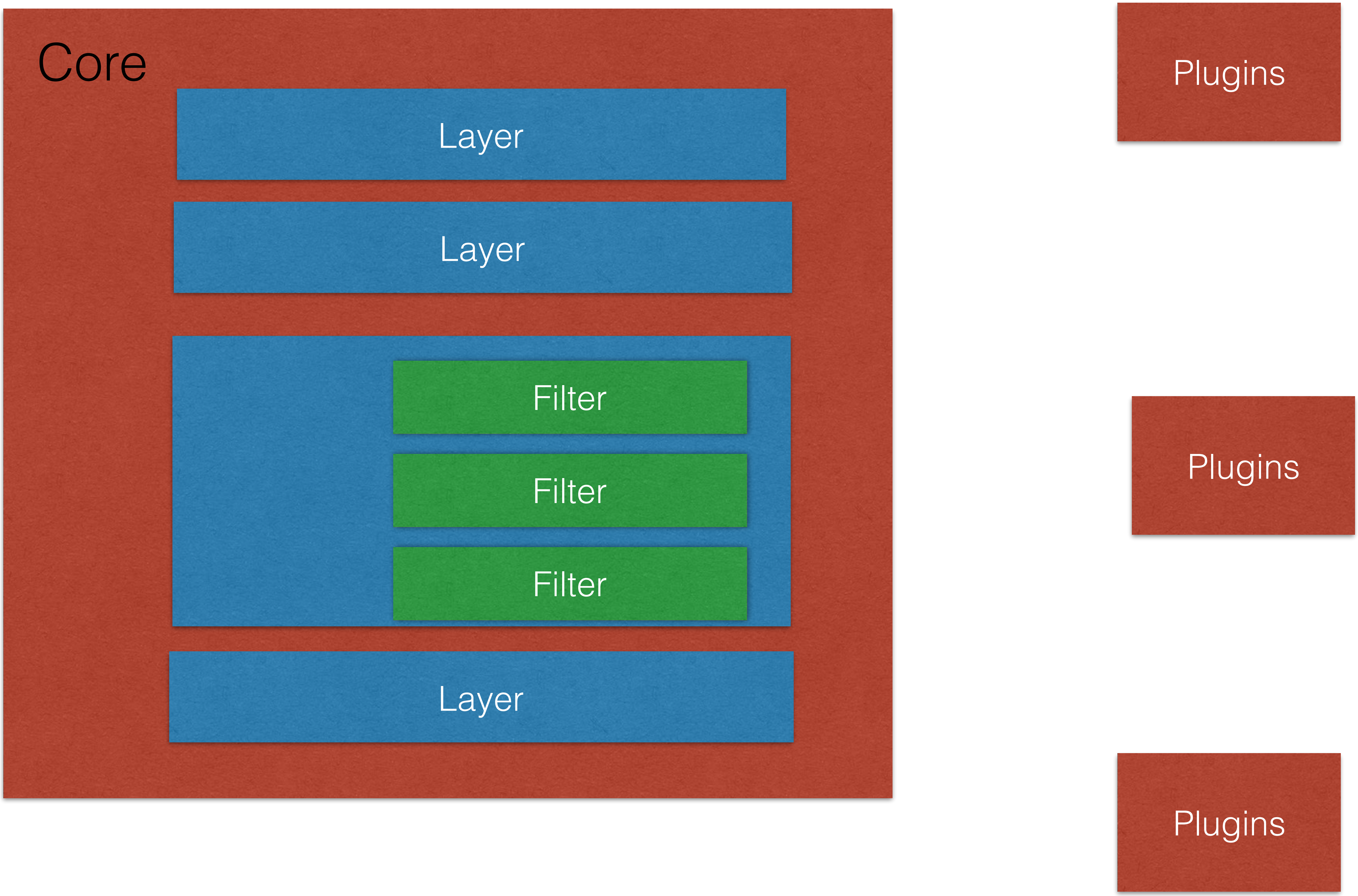
Filter 4

f4(int)

Data5

Microkernel Architecture





Core

Layer

Layer

Filter

Filter

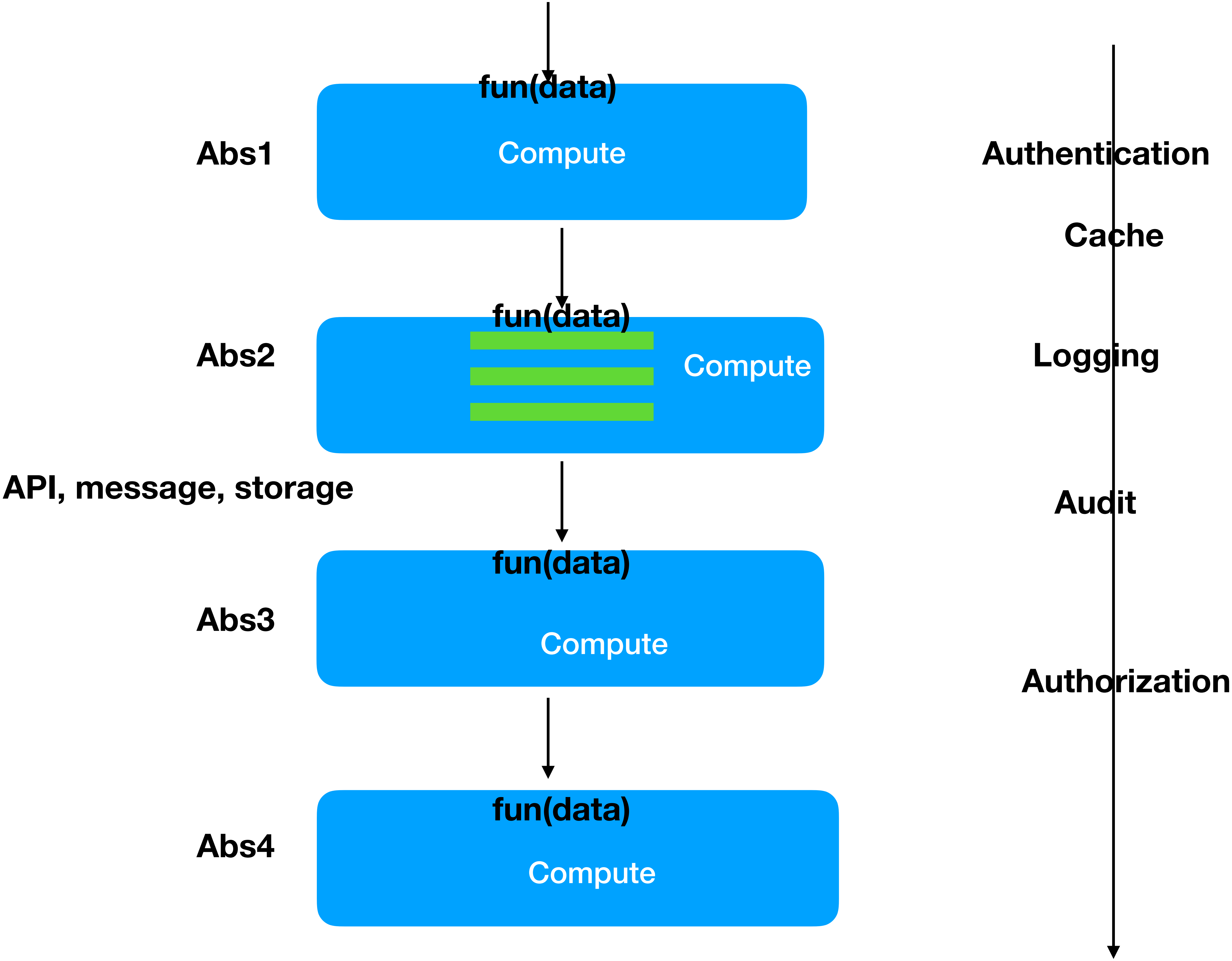
Filter

Layer

Plugins

Plugins

Plugins



fun(data)

Abs1

Compute

Authentication

Cache

fun(data)

Abs2

Compute

Logging

API, message, storage

Audit

fun(data)

Abs3

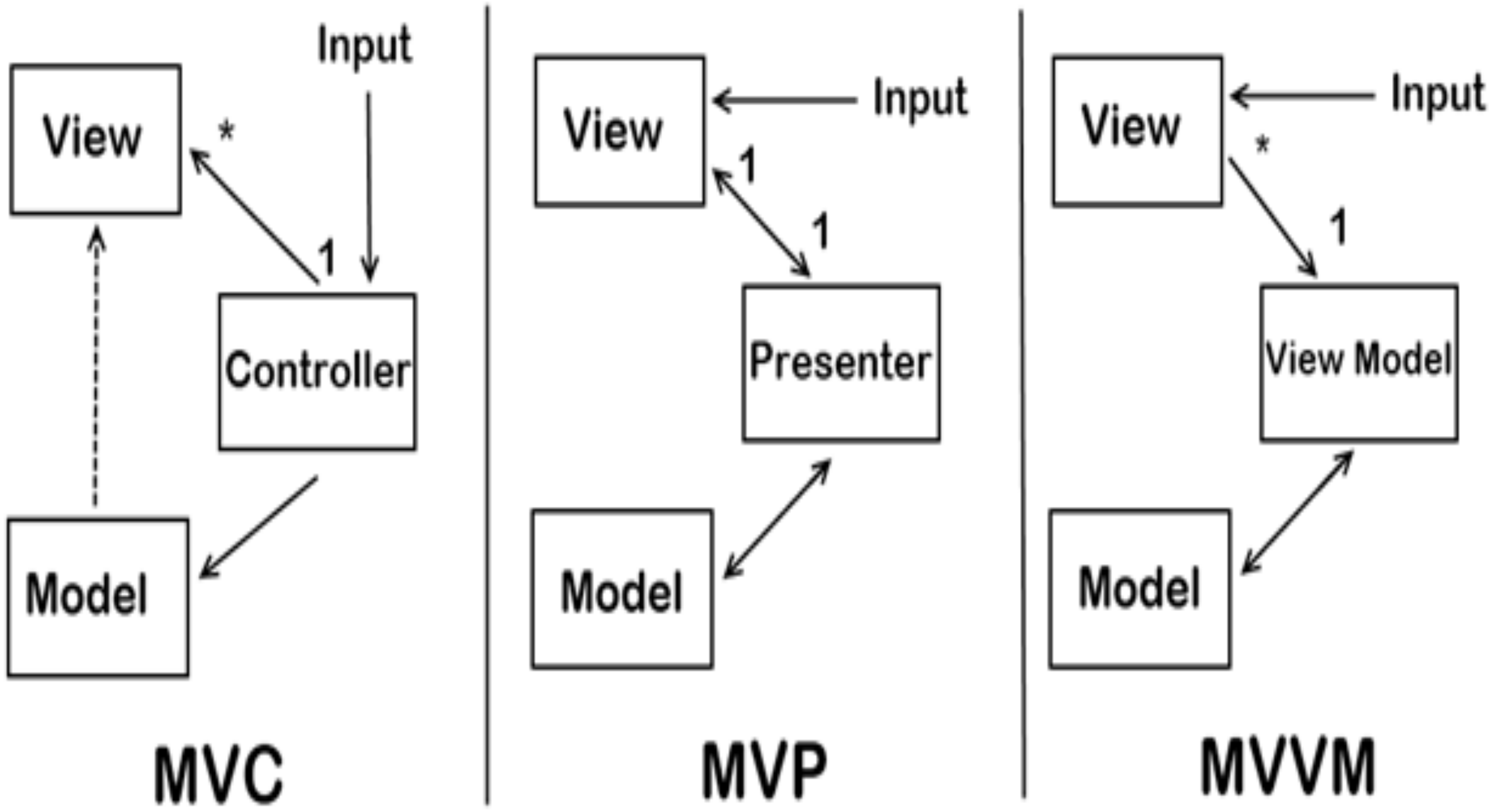
Compute

Authorization

fun(data)

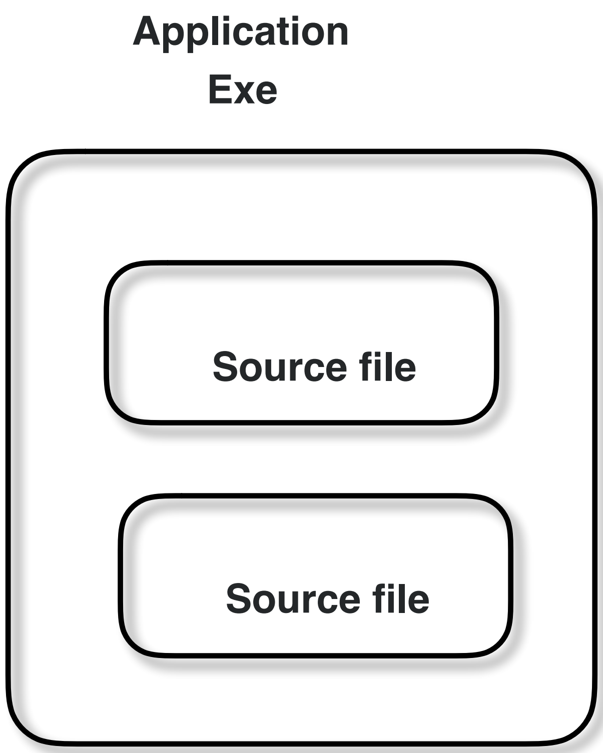
Abs4

Compute


View
*
1
Input
Controller
Model
MVC
View
Input
1
1
Presenter
Model
MVP
View
Input
*
1
View Model
Model
MVVM

1

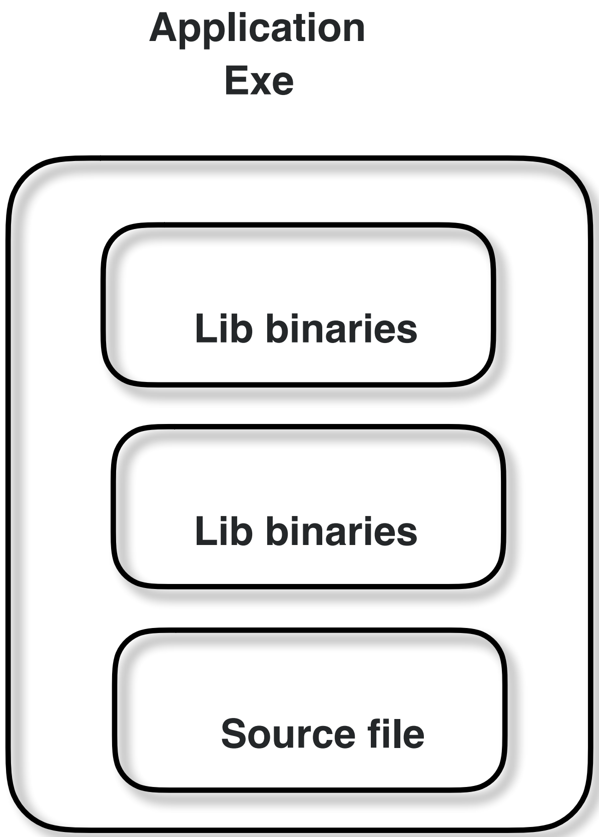
Ship one binary



Compile time monolithic

2

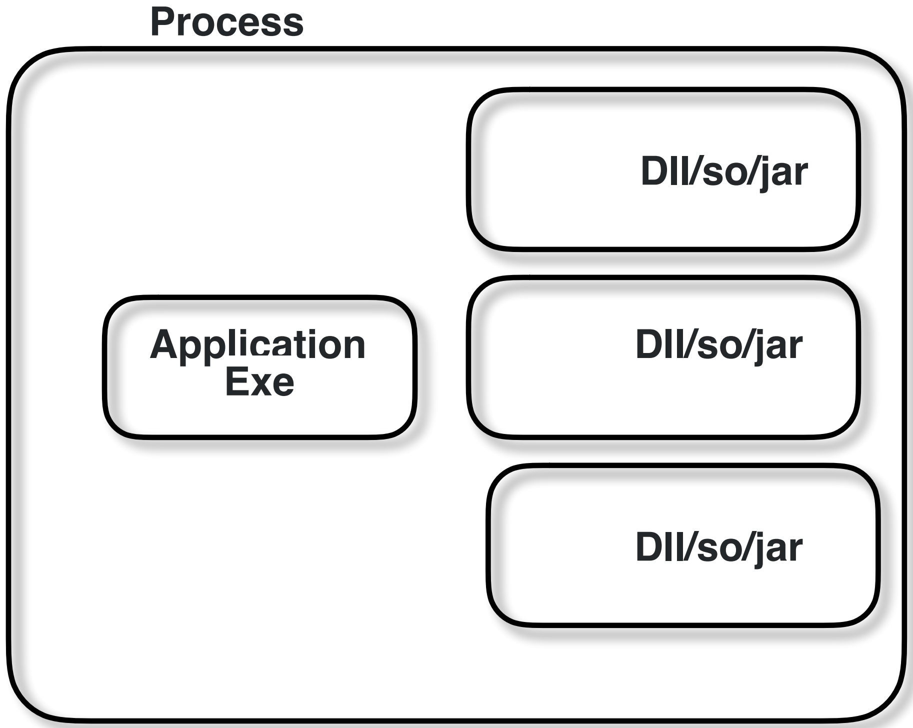
Ship one binary



Link time monolithic

3

Ship multiple binary

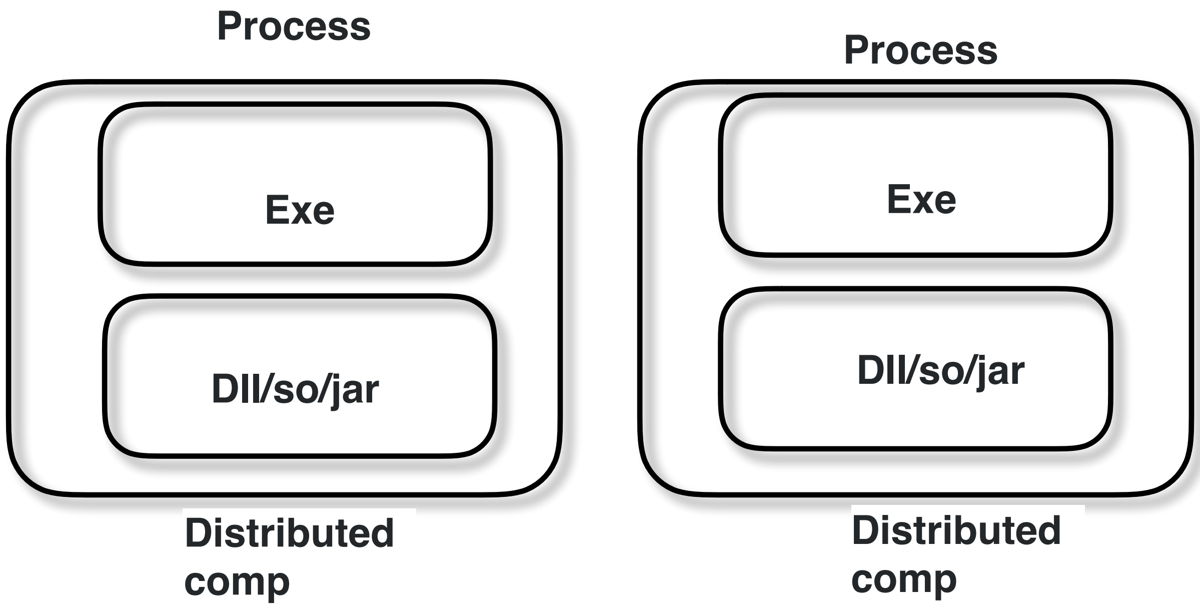


Runtime monolithic

In process comp

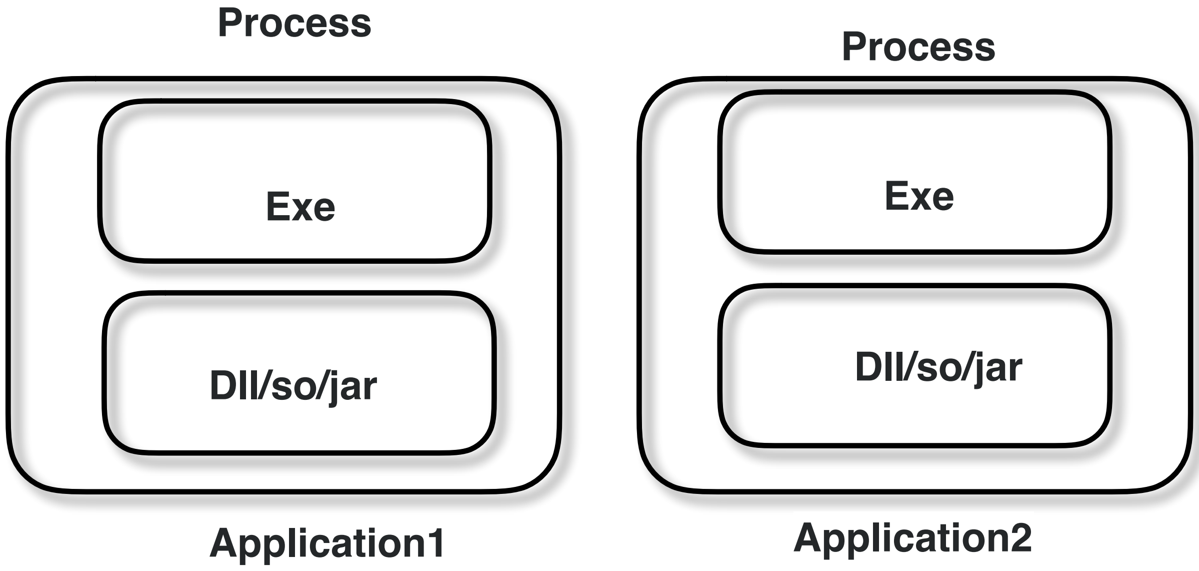
Ship multiple binary & host in multiple process

4



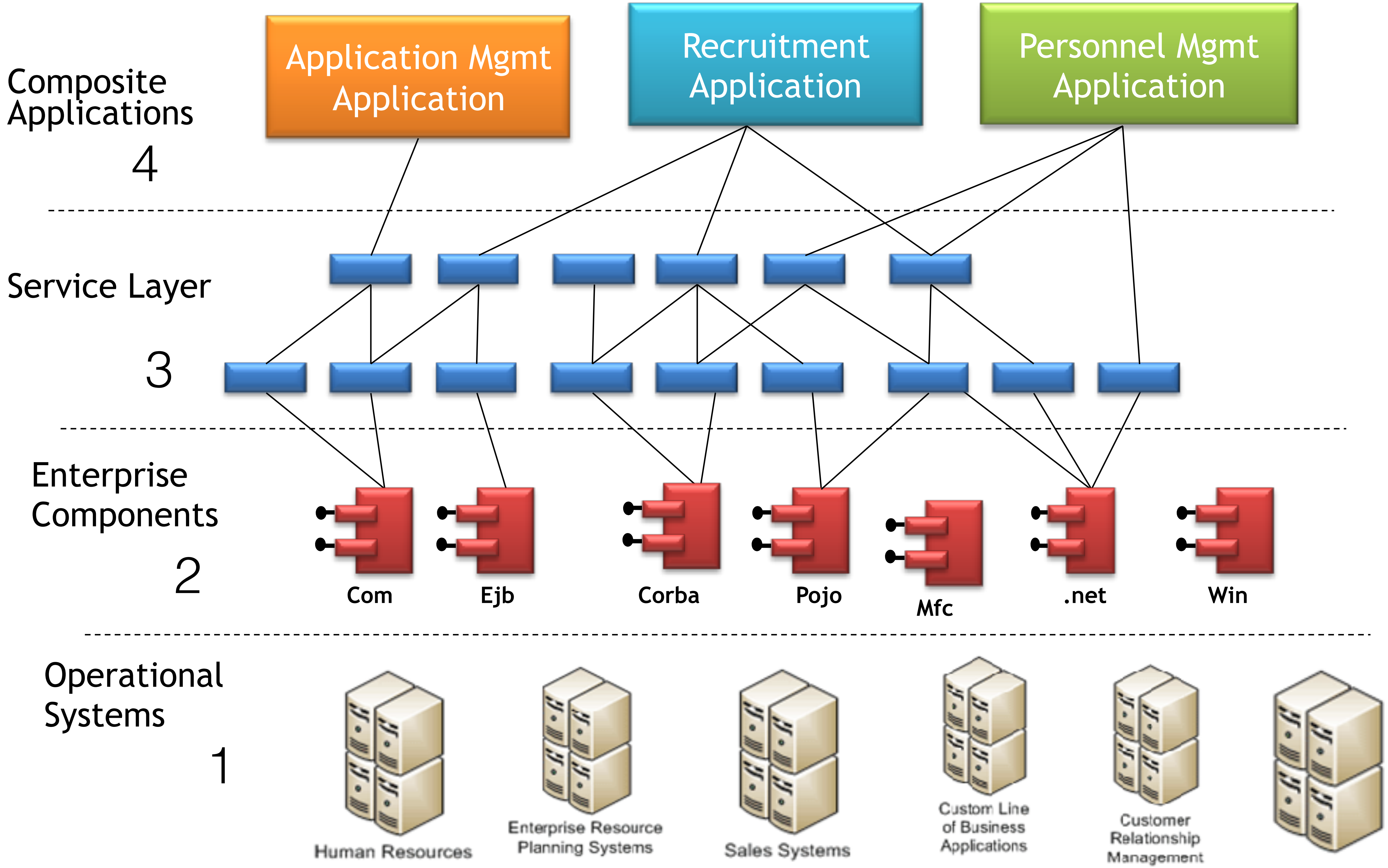
Distributed Application

5



Micro Application

Service Oriented Architecture



Composite Applications
4

Application Mgmt Application

Recruitment Application

Personnel Mgmt Application

Service Layer

3

Enterprise Components
2

Com

Ejb

Corba

Pojo

Mfc

.net

Win

Operational Systems
1

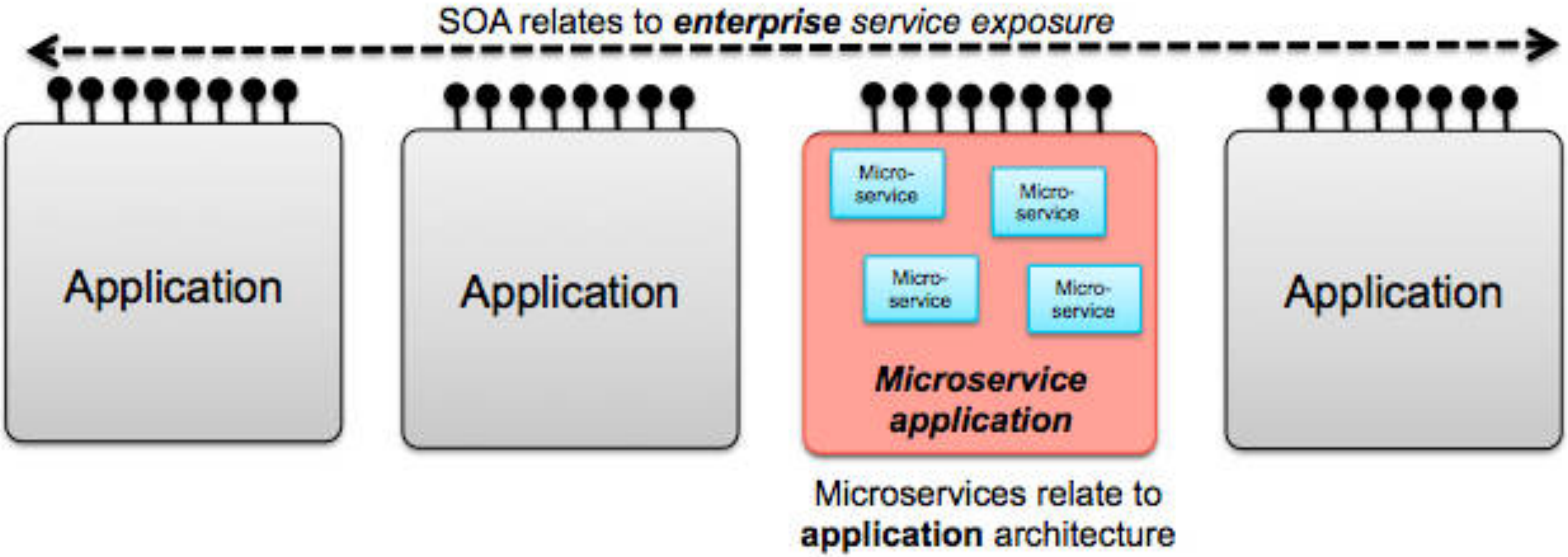
Human Resources

Enterprise Resource Planning Systems

Sales Systems

Custom Line of Business Applications

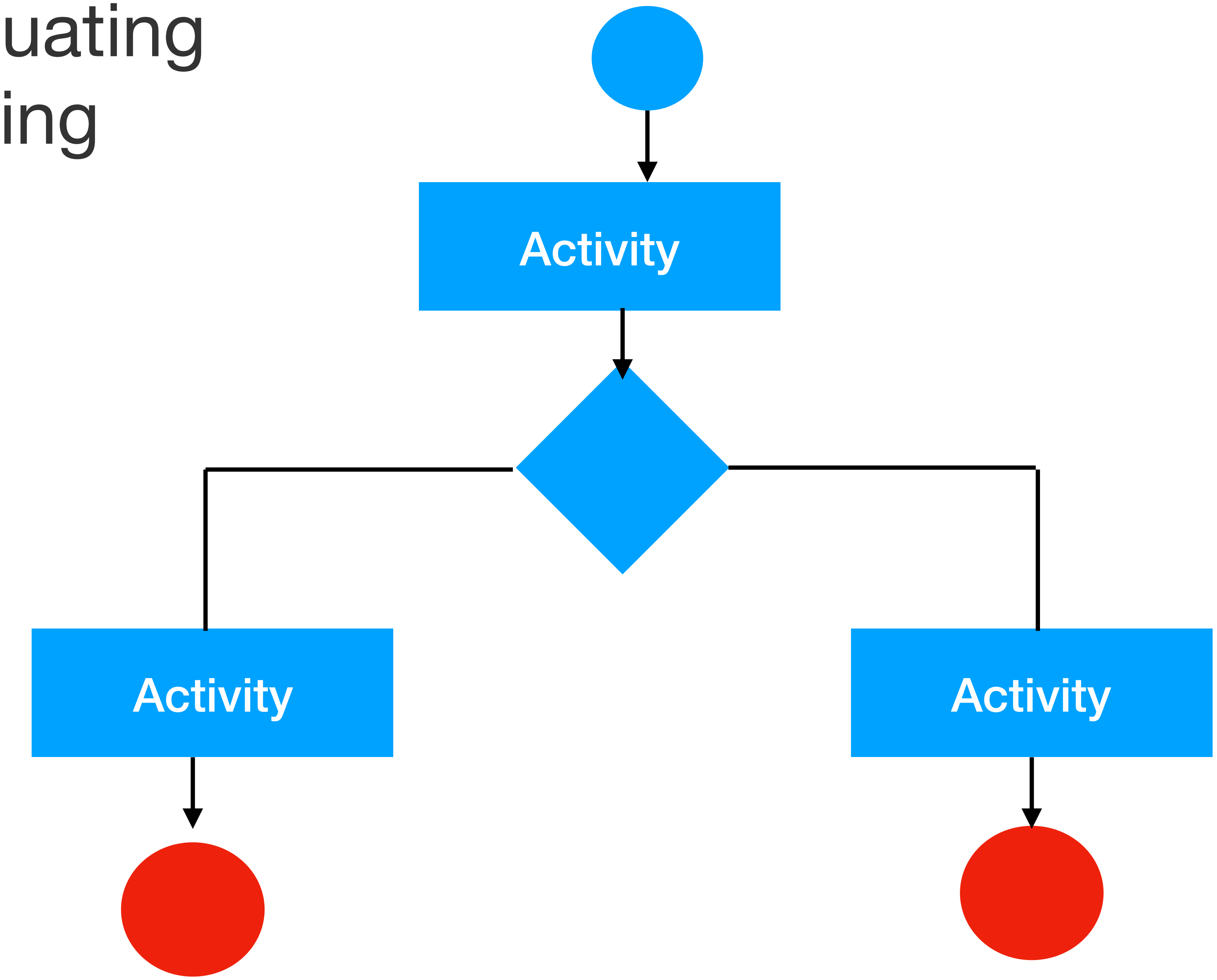
Customer Relationship Management

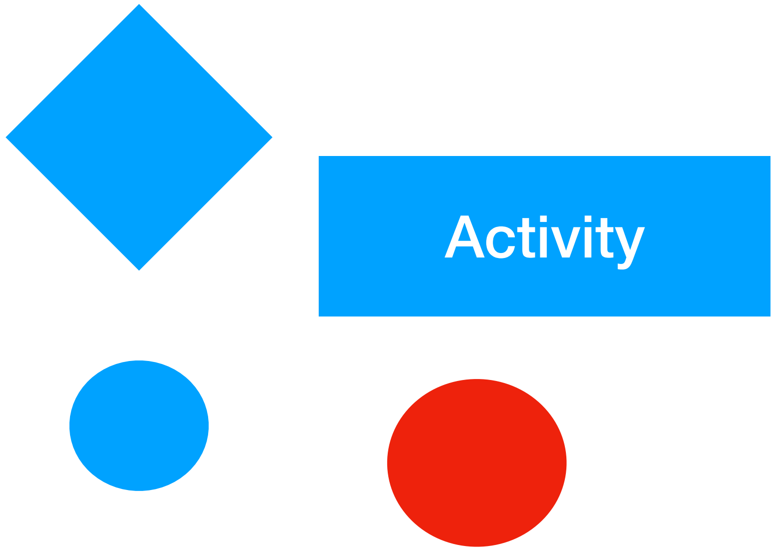

SOA relates to ***enterprise*** *service exposure*
Application
Application
Micro-service
Micro-service
Micro-service
Micro-service
Microservice application
Application
Microservices relate to **application** architecture

Eclipse IDE. Downloading the basic Eclipse product provides you little more than a fancy editor. However, once you start adding plug-ins, it becomes a highly customizable and useful product.

An application is required to perform a variety of tasks of varying complexity on the information that it processes. The processing tasks performed by each module, or the deployment requirements for each task, could change as business requirements are updated. Also, additional processing might be required in the future, or the order in which the tasks performed by the processing could change. A solution is required that addresses these issues, and increases the possibilities for code reuse.

A workflow implementation. The implementation of a workflow contains concepts like the order of the different steps, evaluating the results of steps, deciding what the next step is, etc.





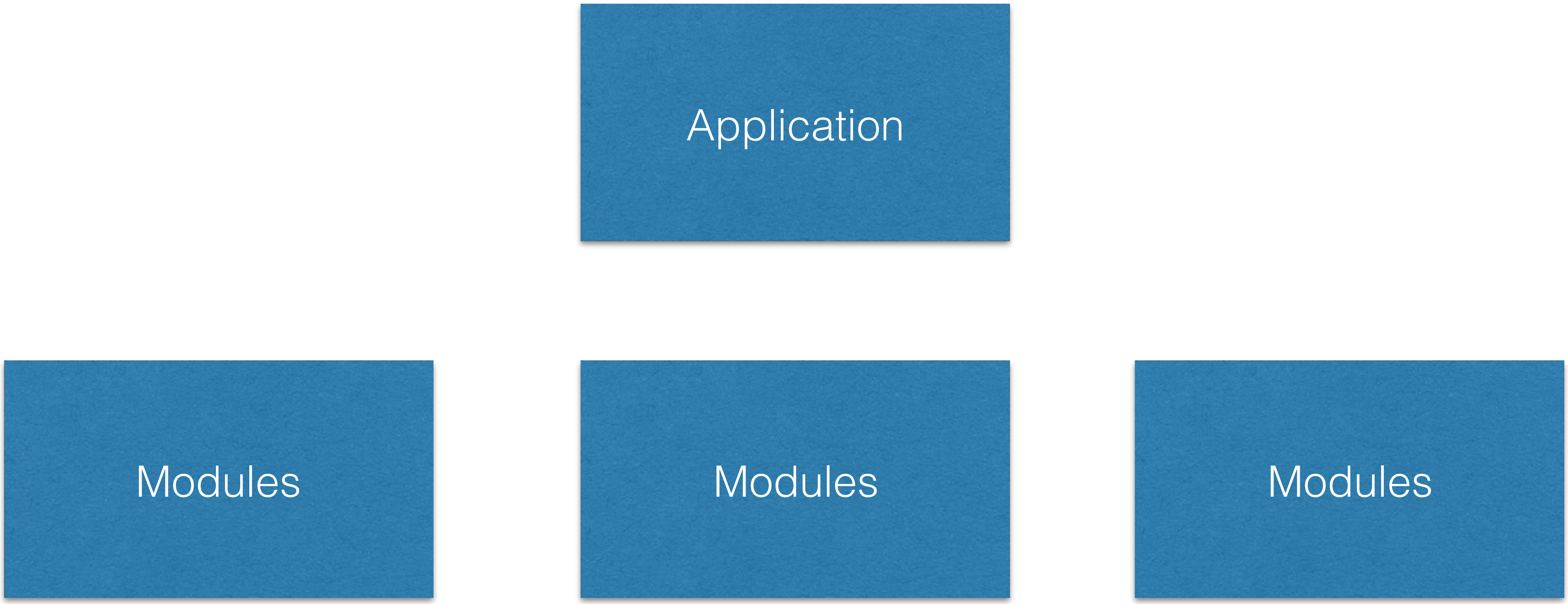
A task scheduler. A scheduler contains all the logic for scheduling and triggering tasks

Internet browses plug-ins add additional capabilities that are not otherwise found in the basic browser

Networking engineering is a complicated task, which involves software, firmware, chip level engineering, hardware, and electric pulses. To ease network engineering, the whole networking concept is decomposed into more manageable parts.

Break an app into smaller manageable piece

	2 Modules	**2 Applications**	**W**	**Score**
Share Database / Storage	Yes	No	2	
Share Infra (Hosting)	Yes	No	3	
Share Sorce Control	Yes	No	2	
Share CI/CD (Build Server)	Yes or No	No	3	
Share functional logic (same feature)		No		
Fun Requirements	Shared	Its own	1	
SCRUM Team / Sprint	Shared	Its own	1	
Test Cases	Shared	Its own	1	
Architecture	Shared	Its own	1	
Technology Stack / Fwks	Shared	Its own	1	



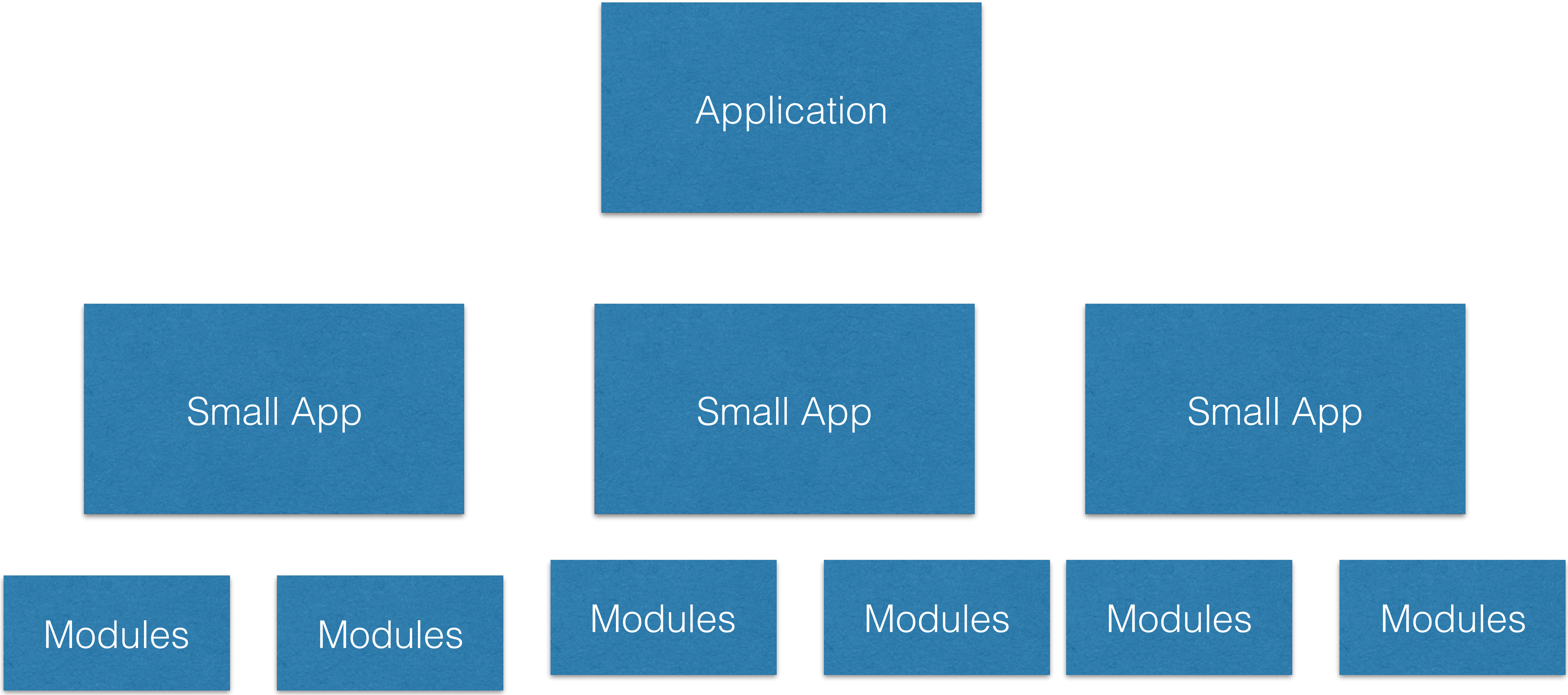
Application

Modules

Modules

Modules

3 or 4



Application

Small App

Small App

Small App

Modules

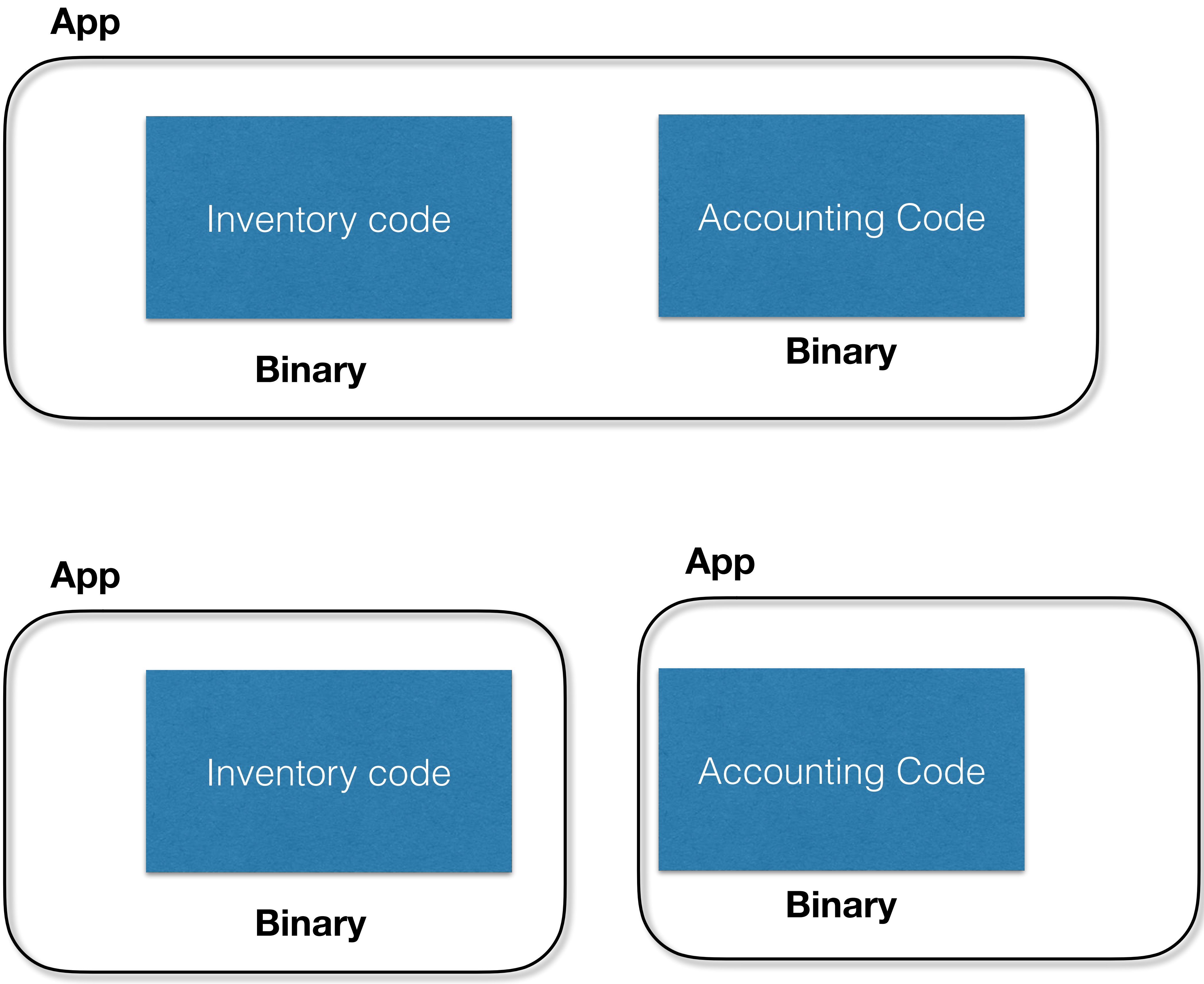
Modules

Modules

Modules

Modules

Modules



App

Inventory code

Binary

Accounting Code

Binary

App

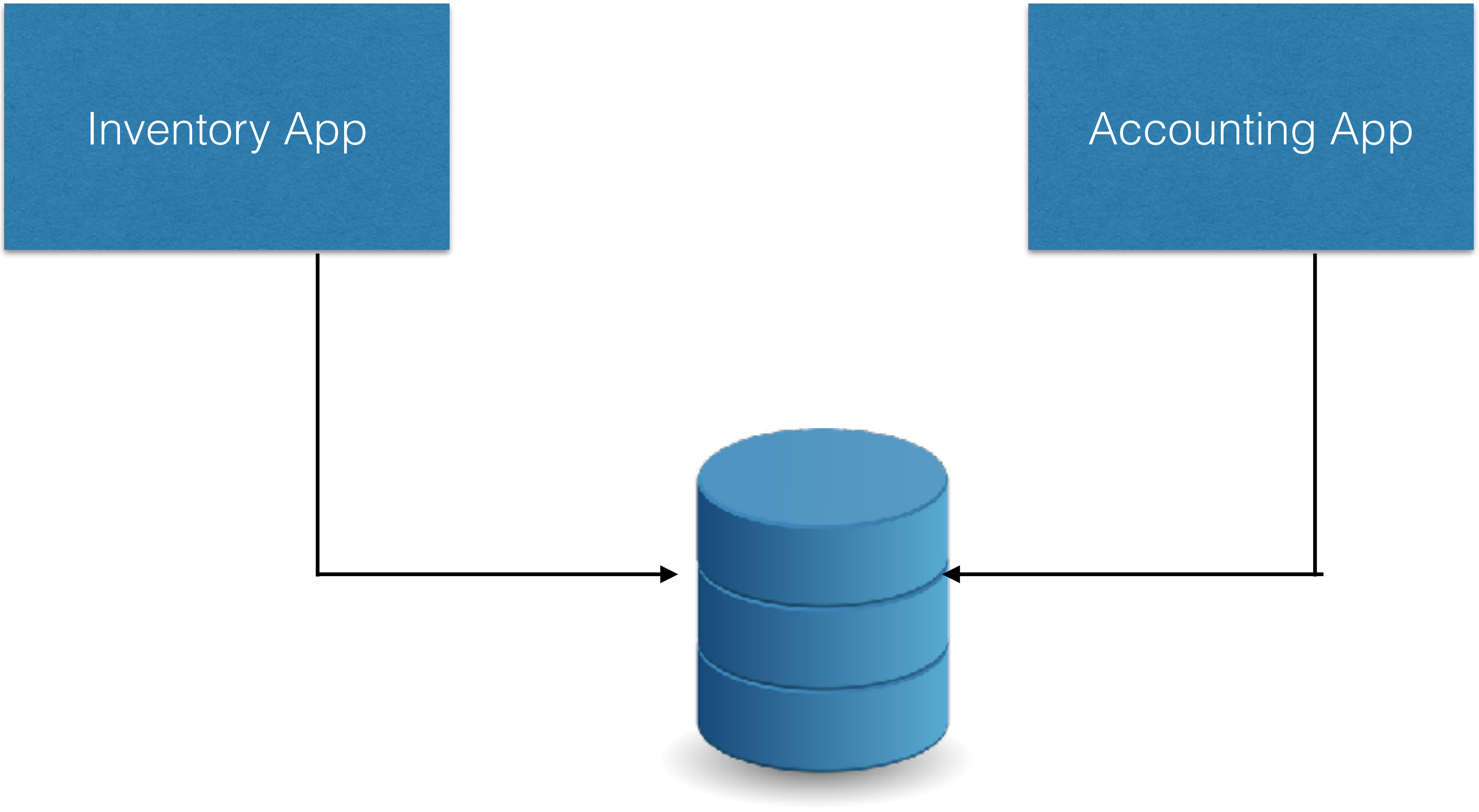
Inventory code

Binary

App

Accounting Code

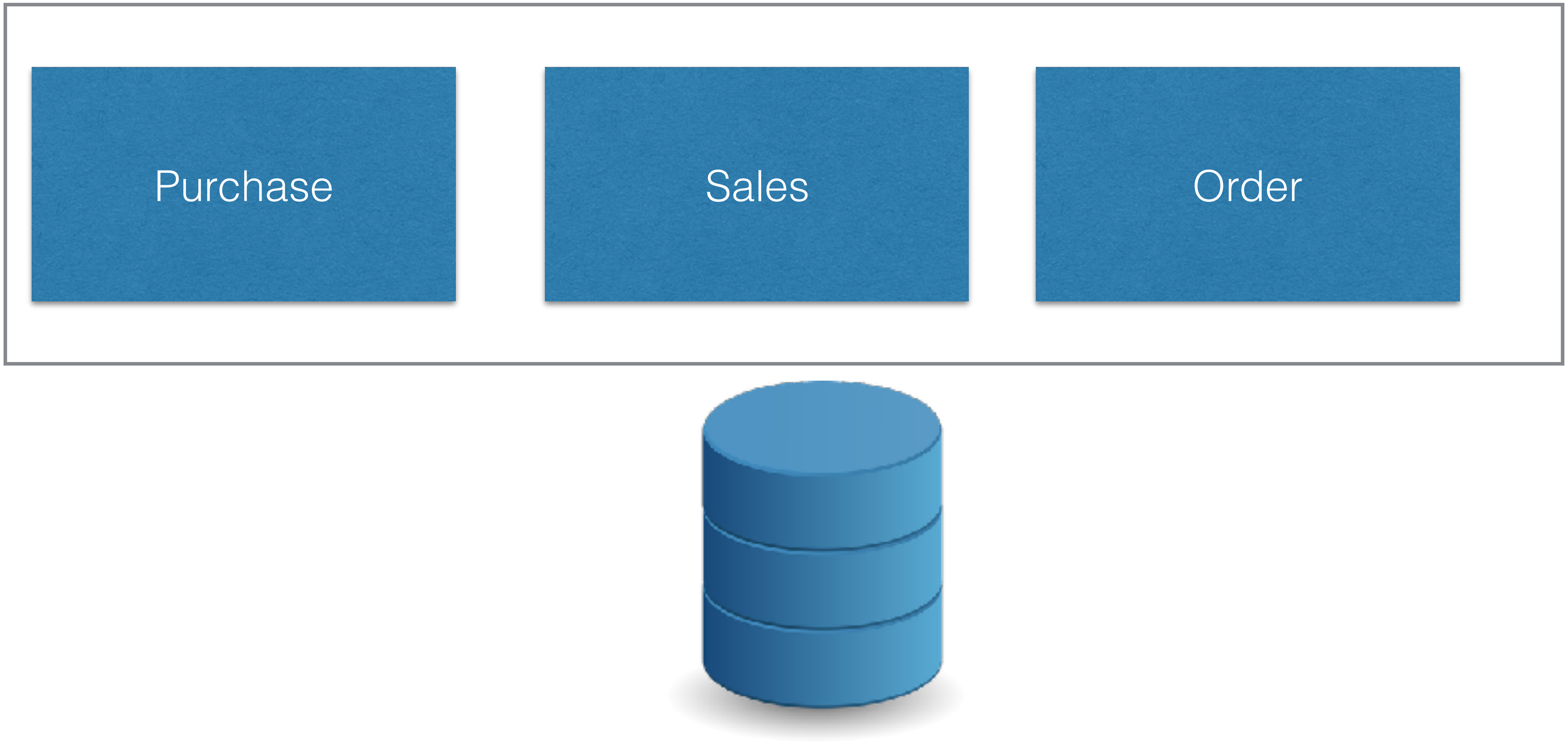
Binary



Inventory App

Accounting App

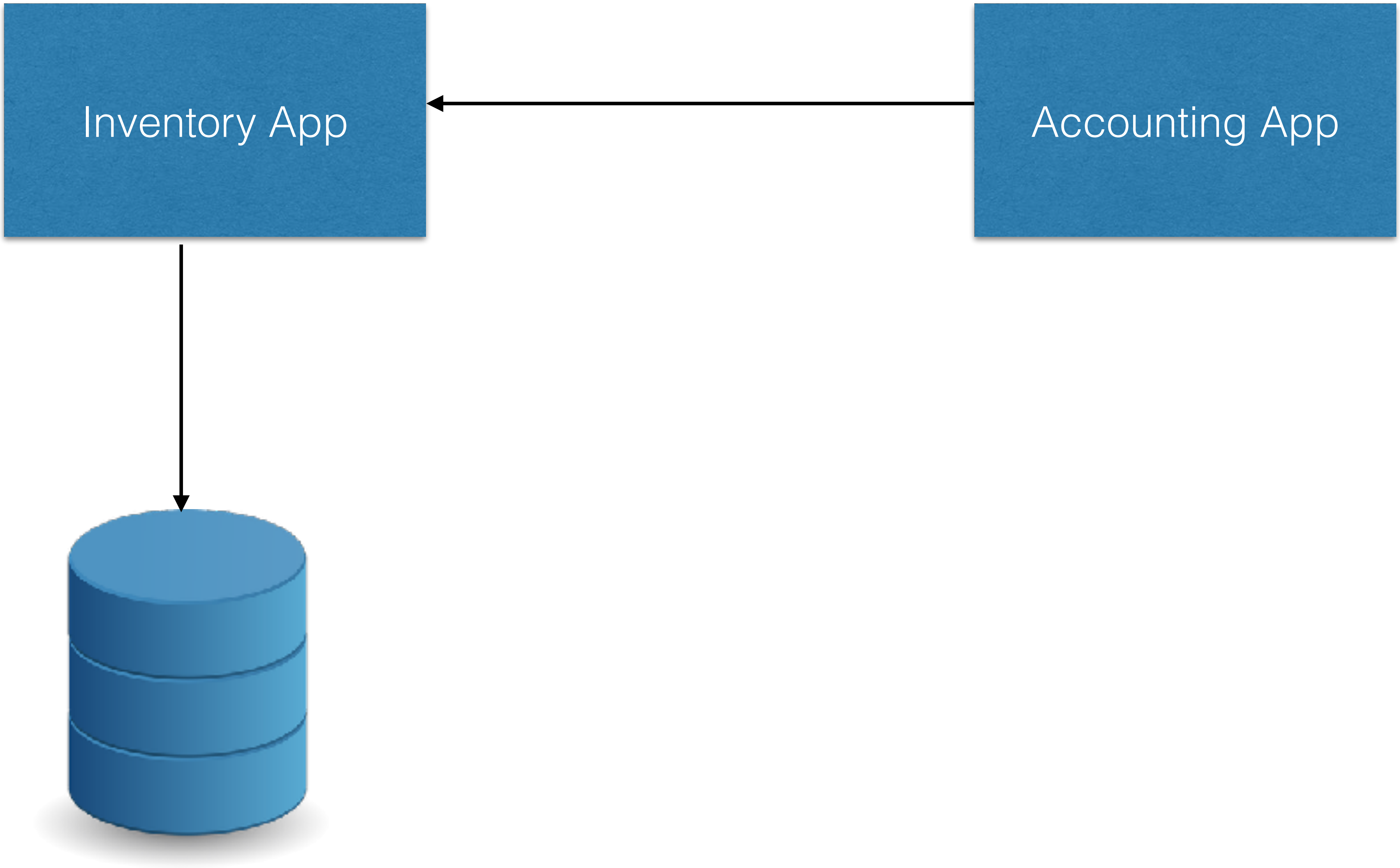
Inventory Application

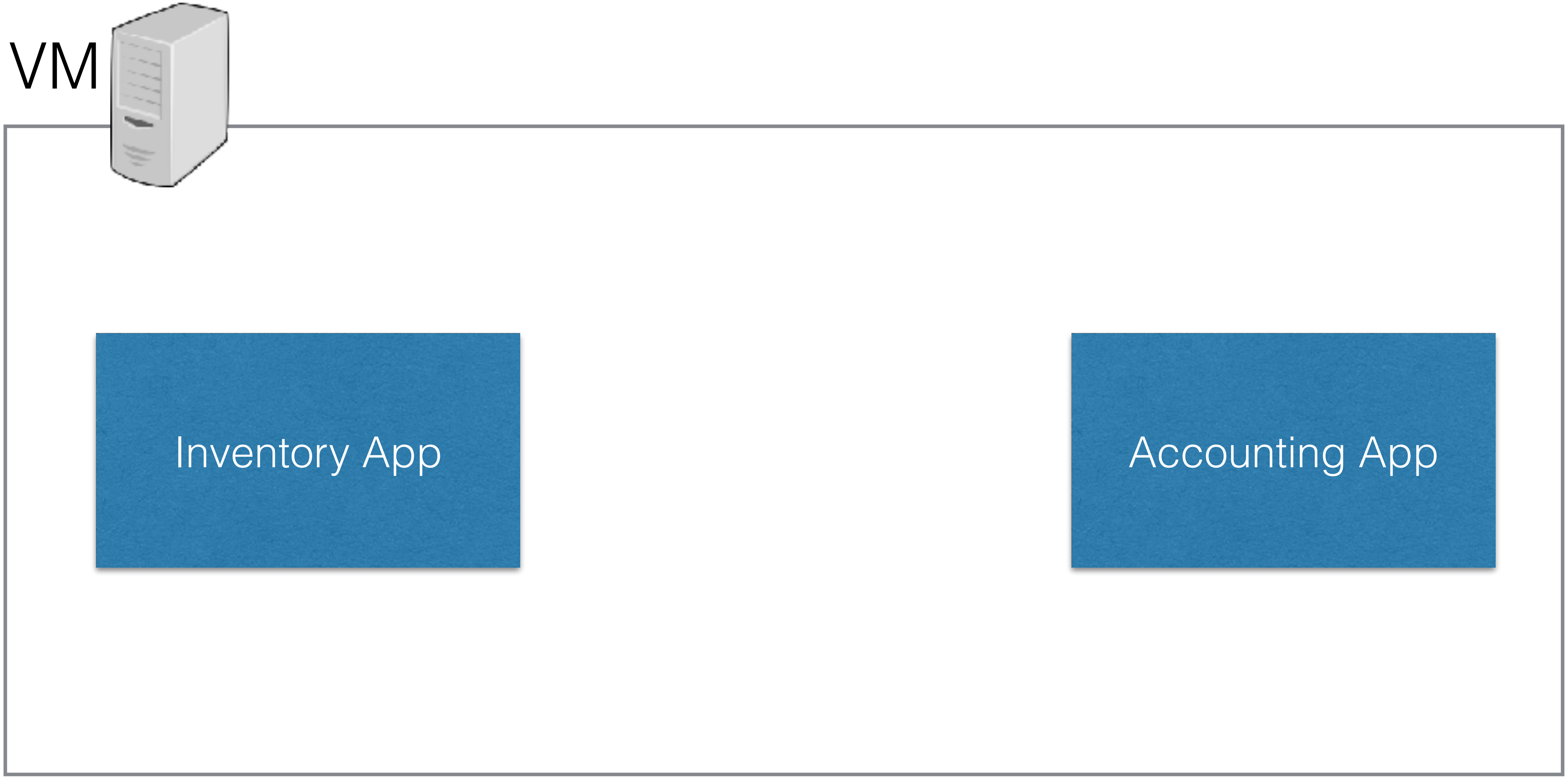


Purchase

Sales

Order

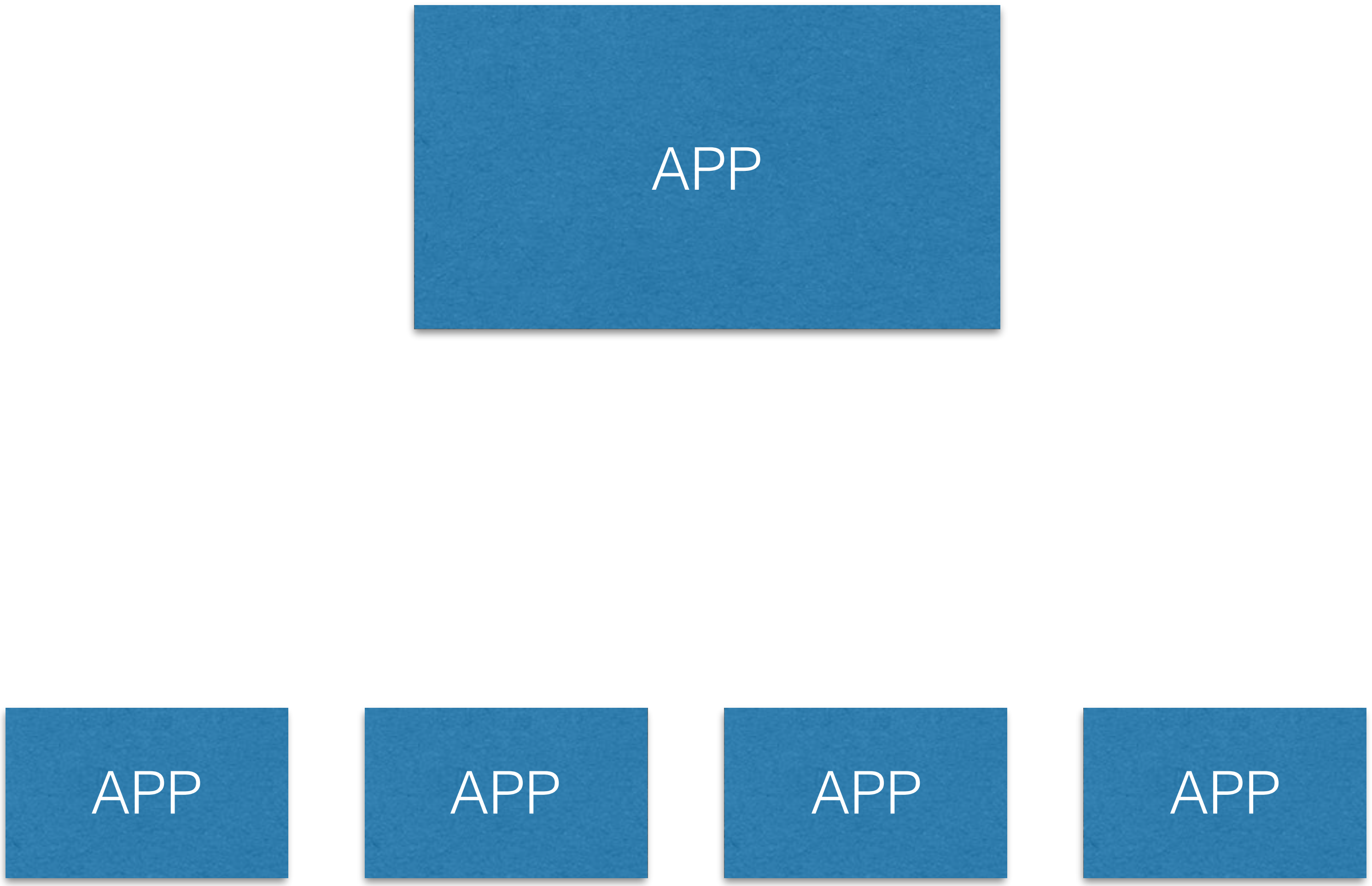

Inventory App
Accounting App



VM

Inventory App

Accounting App



APP

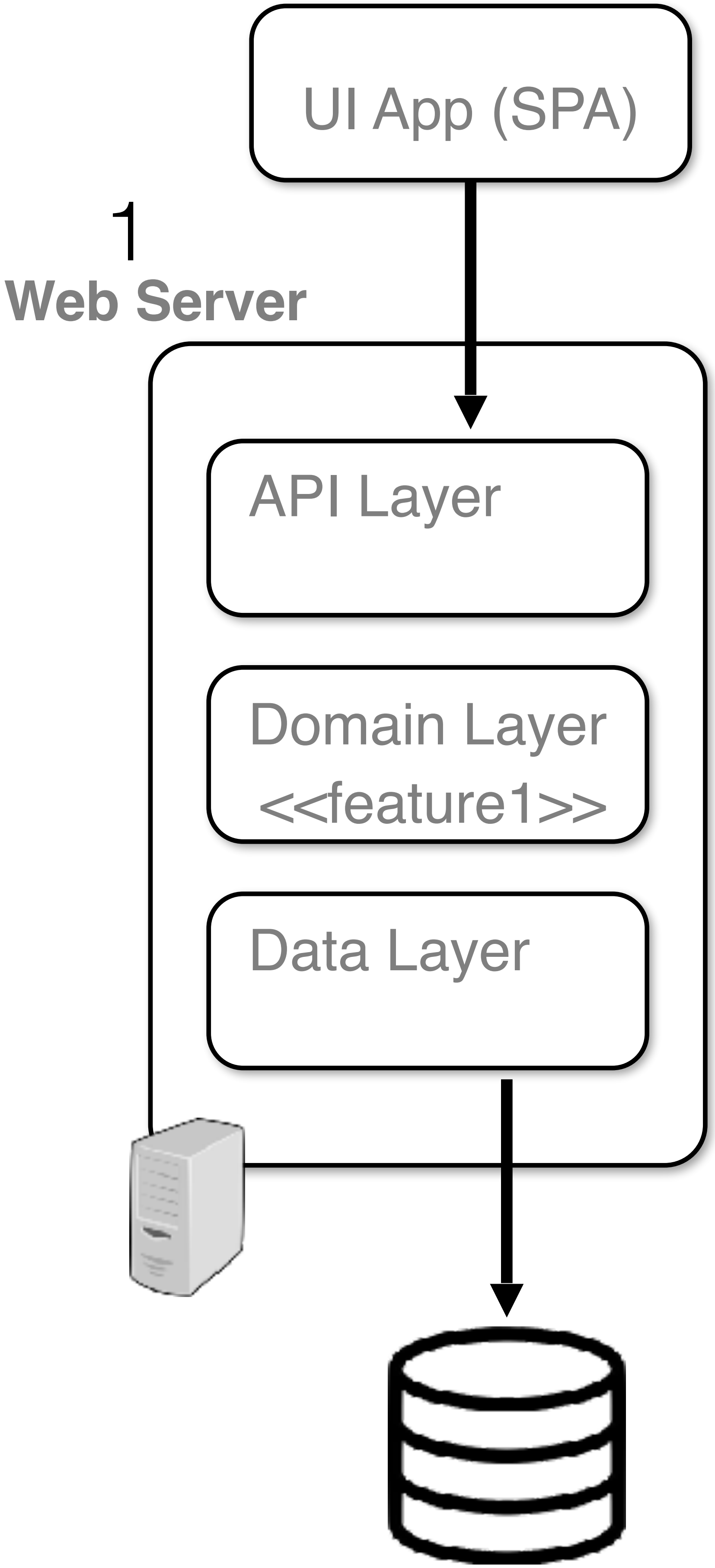
APP

APP

APP

APP

Types of Microservice



1

Web Server

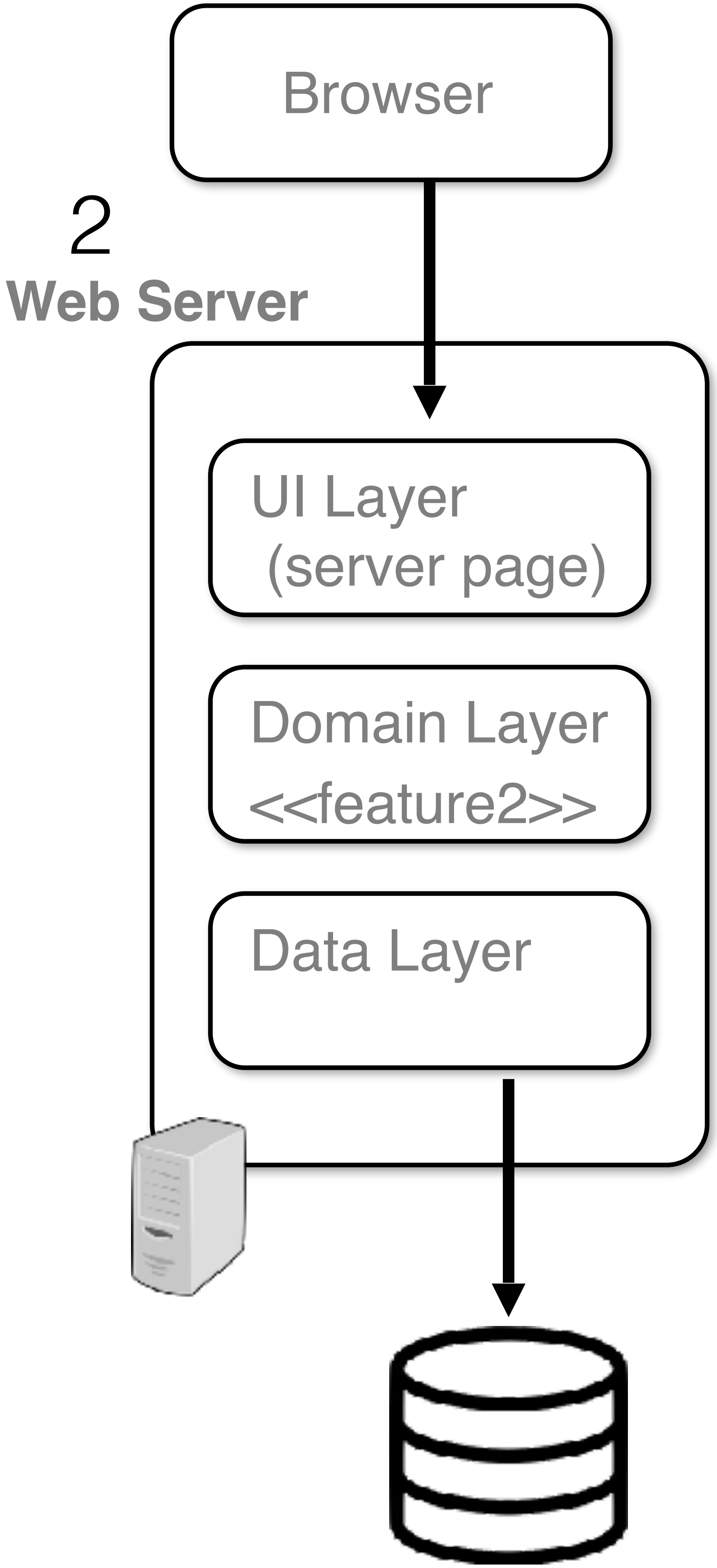
UI App (SPA)

API Layer

Domain Layer <<feature1>>

Data Layer

Angular (SPA)



2

Web Server

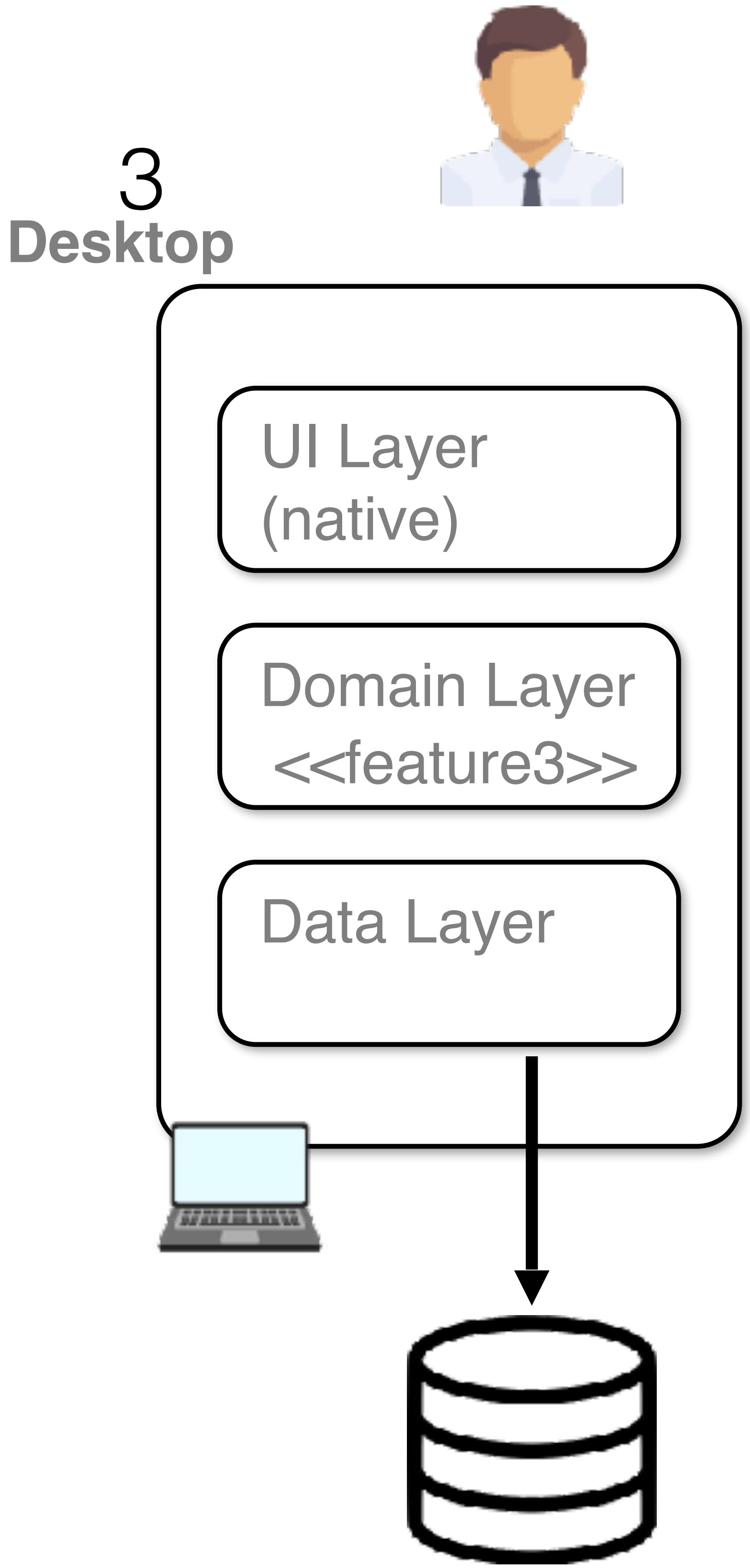
Browser

UI Layer (server page)

Domain Layer <<feature2>>

Data Layer

JSP, Servlet, JSF (Server pages)



3

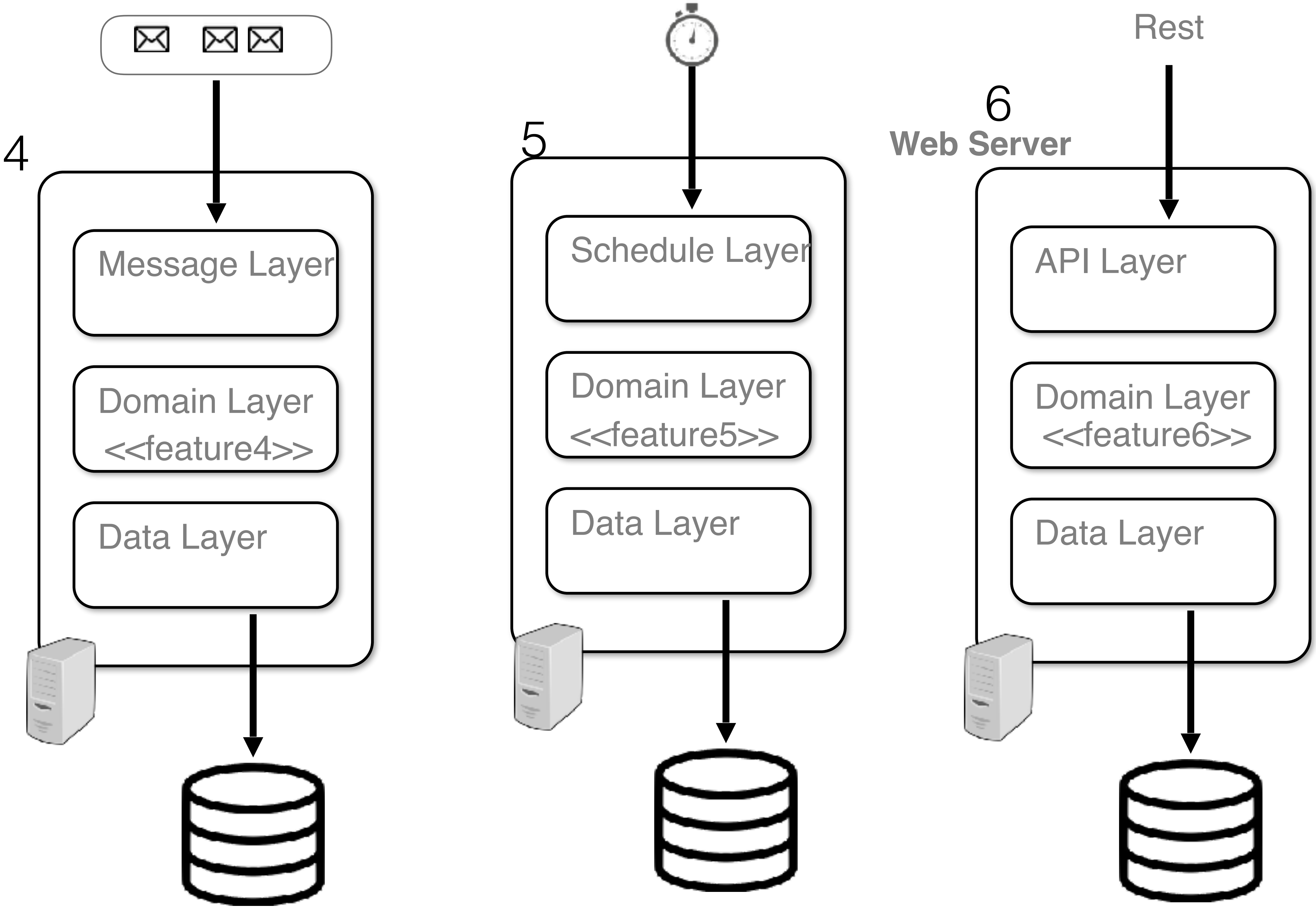
Desktop

UI Layer (native)

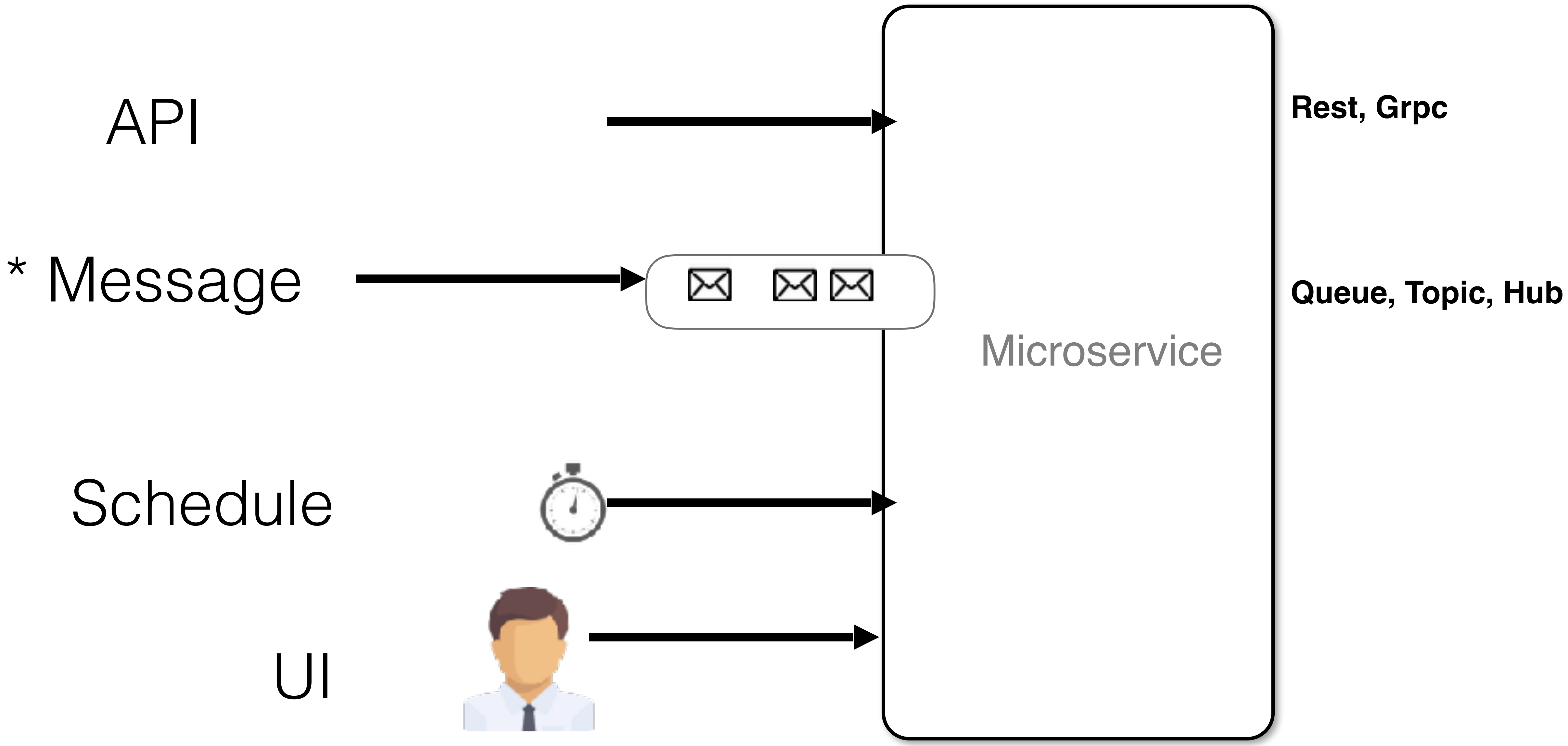
Domain Layer <<feature3>>

Data Layer

Swing, Android, Ios (Native)



4
Message Layer
Domain Layer
<<feature4>>
Data Layer
5
Schedule Layer
Domain Layer
<<feature5>>
Data Layer
Rest
6
Web Server
API Layer
Domain Layer
<<feature6>>
Data Layer



API

* Message

Schedule

UI

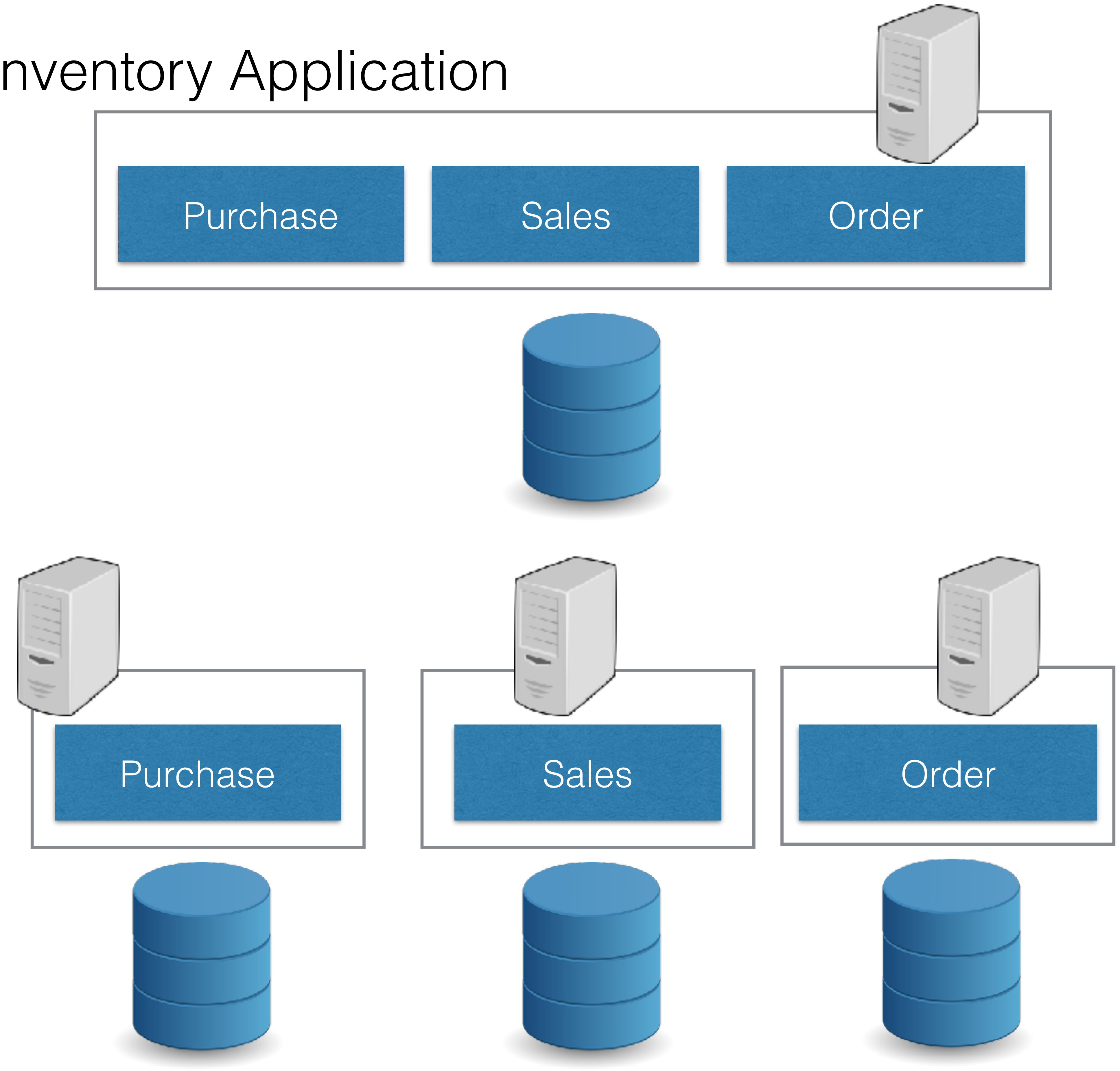
Microservice

Rest, Grpc

Queue, Topic, Hub

	Pros/ Cons	Solution
Development time		
Micro-service practices Learning Curve		
Resource Performance (CPU, Memory, I/O)		
Db Transaction Management		
Views / Report / Dash board/ join		
Infra Cost		
First time Deployment effort		
Debugging, Error Handling (End to End)		
Integration Test		
Log Mgmt (debug/ error)		
Config Mgmt		
Authentication (who)		
Authorization (what can they do)		
Audit Log mgmt (who accessed what)		
Monitoring / Alerting		
Data Security and Privacy (transit, storage)		
Build Pipeline (CI)		
Agile Architecture (Agility to change)		
Feature Shipping (Agility to ship)/ CD		
Scalability (volume - request, data,		
Availability		
Ability to do Polygot		

Inventory Application



Purchase

Sales

Order

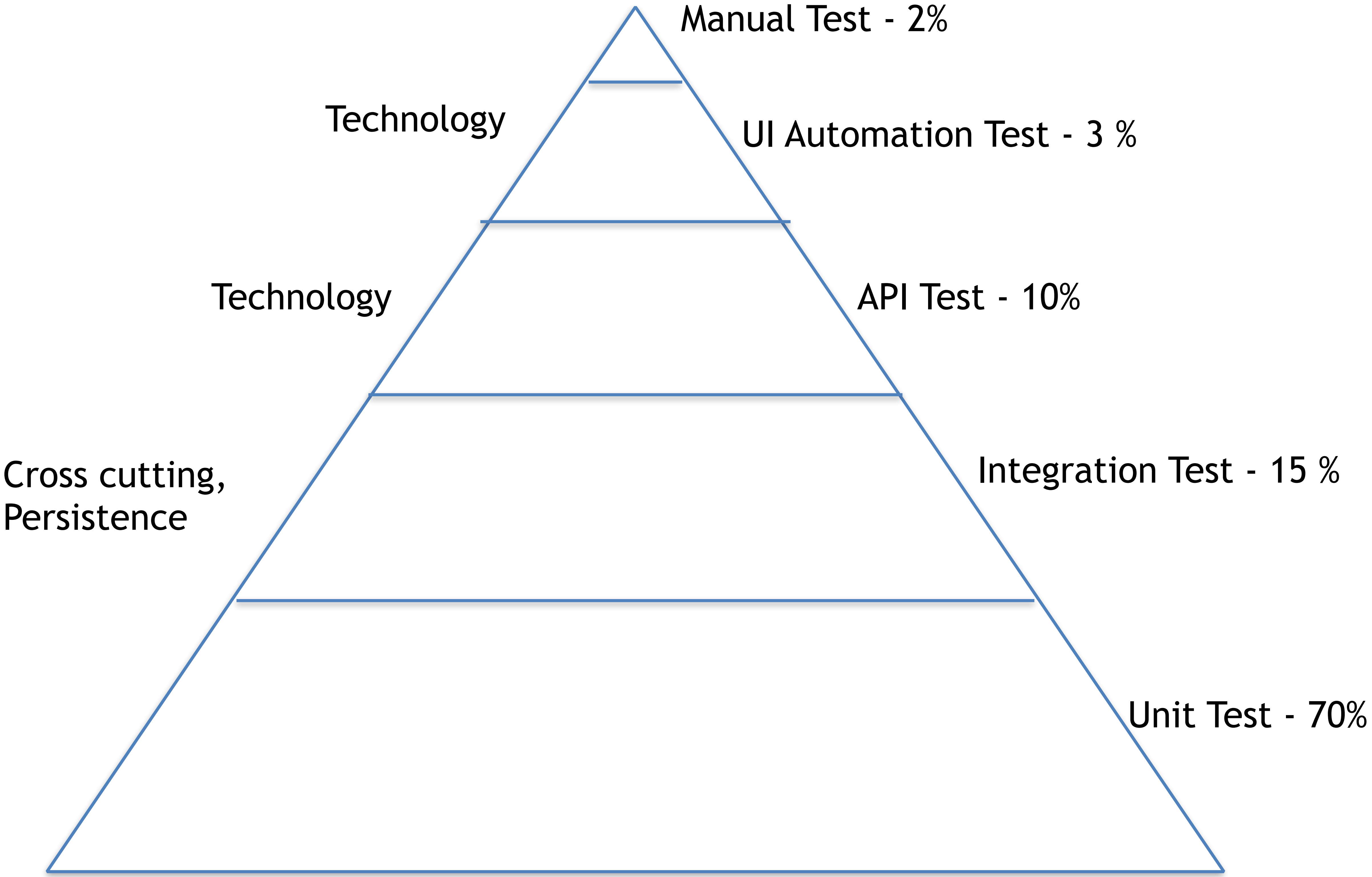
Purchase

Sales

Order

Decentrailze Domain
Centrailze Tech

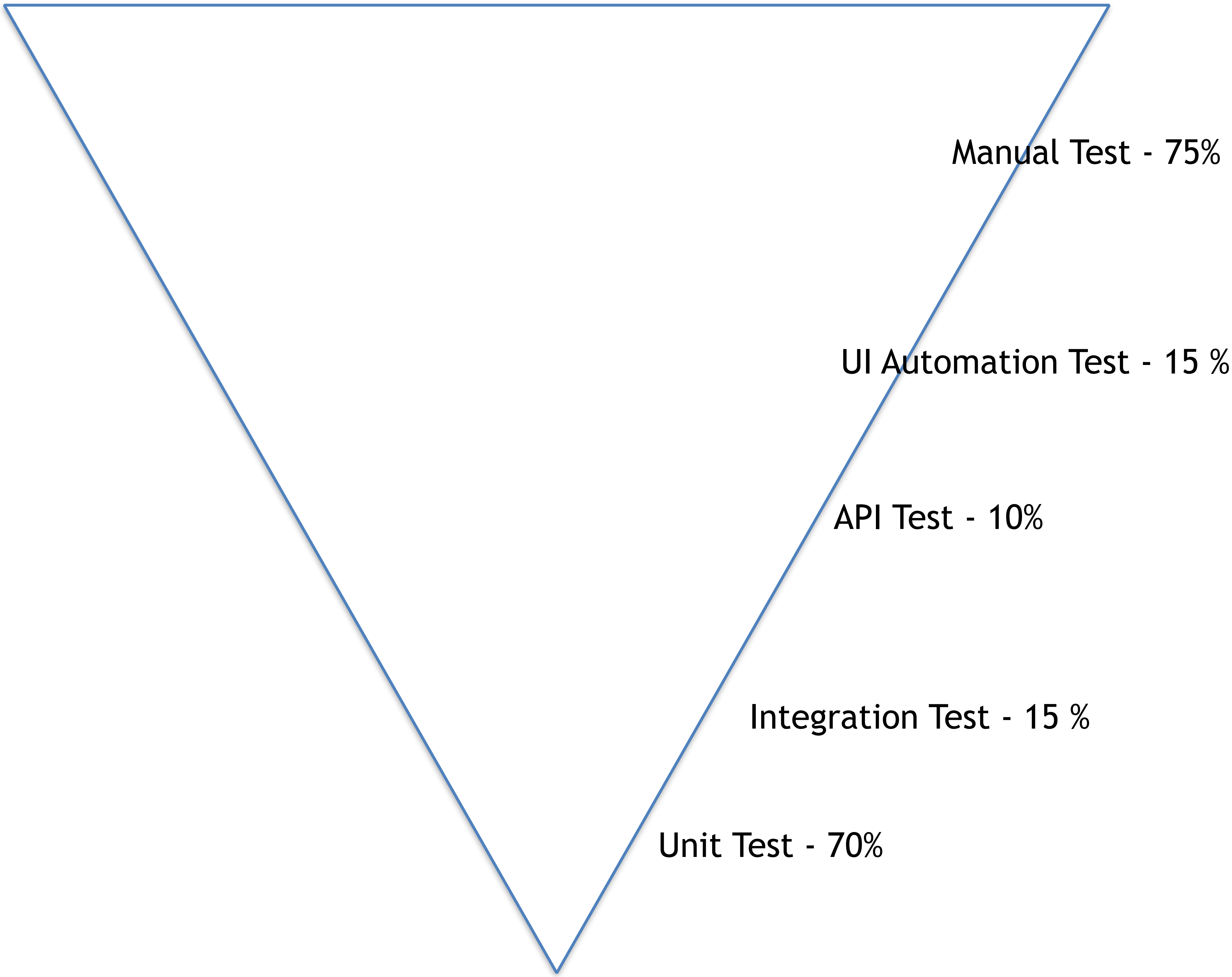
Pyramid test



Unit Test

- Documentation for Code
- Design Code
- Regression
- Find Bugs

Its working don’t touch it



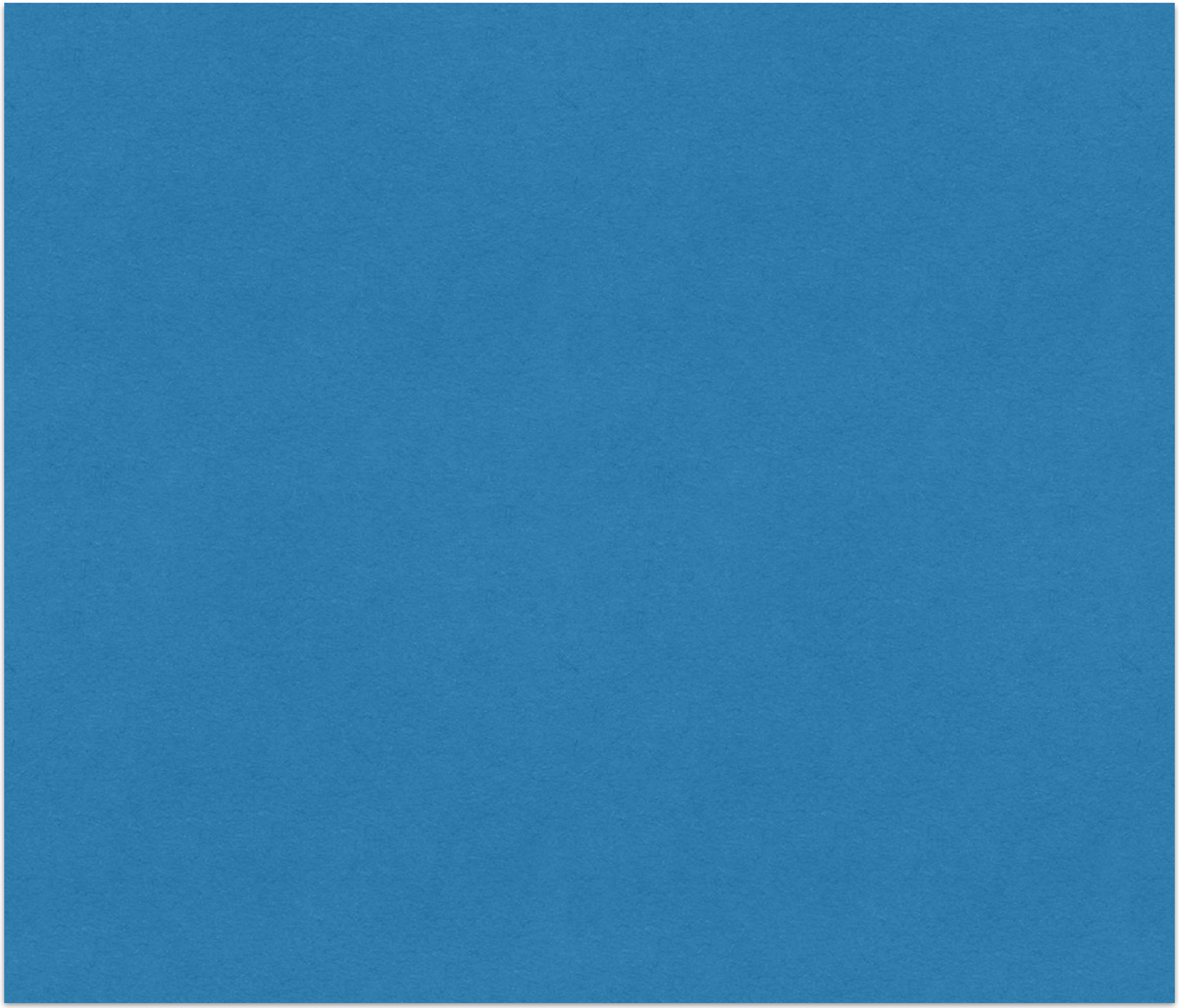
Manual Test - 75%

UI Automation Test - 15 %

API Test - 10%

Integration Test - 15 %

Unit Test - 70%



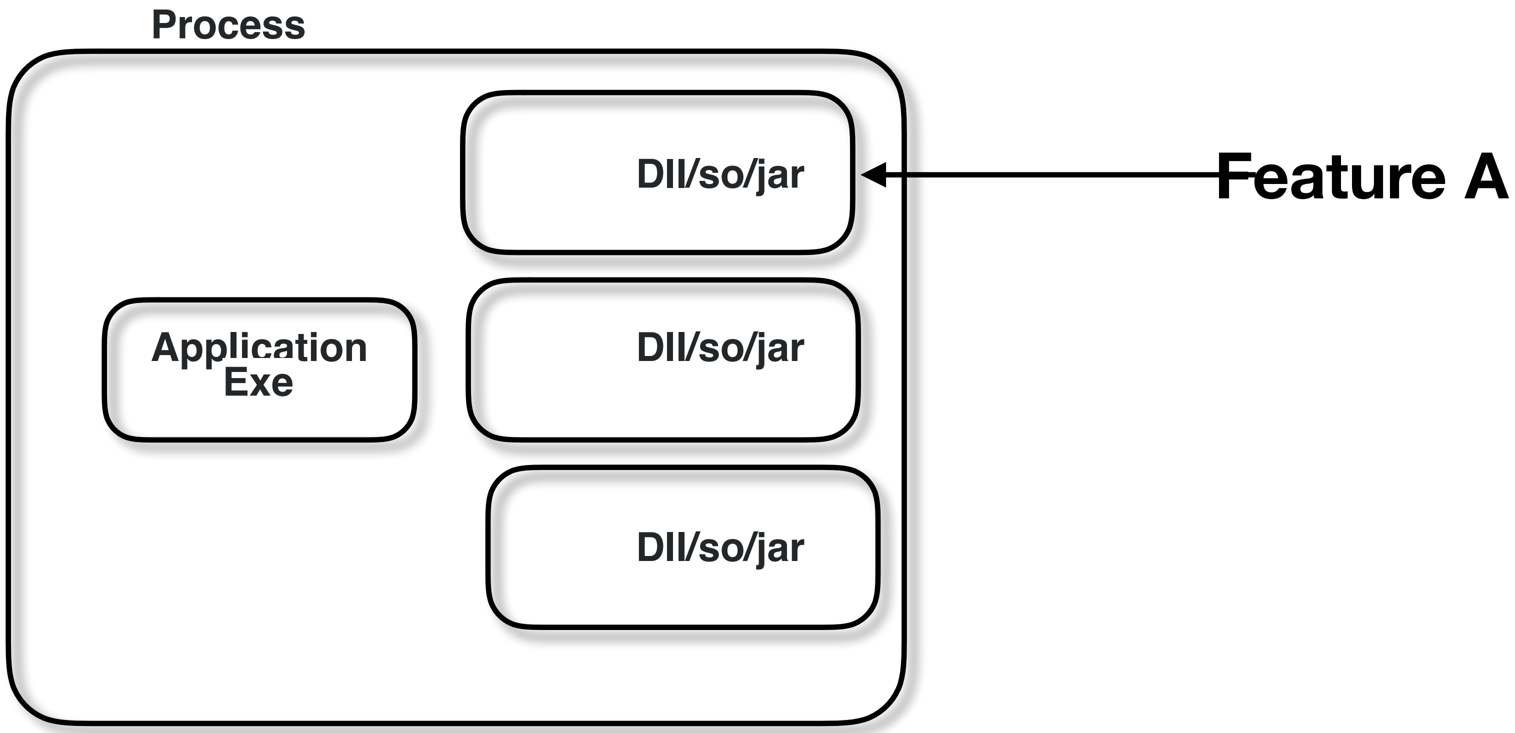
Manual Test - 2%

UI Automation Test - 3 %

API Test - 10%

Integration Test - 15 %

Unit Test - 70%



Process

Application Exe

Dll/so/jar

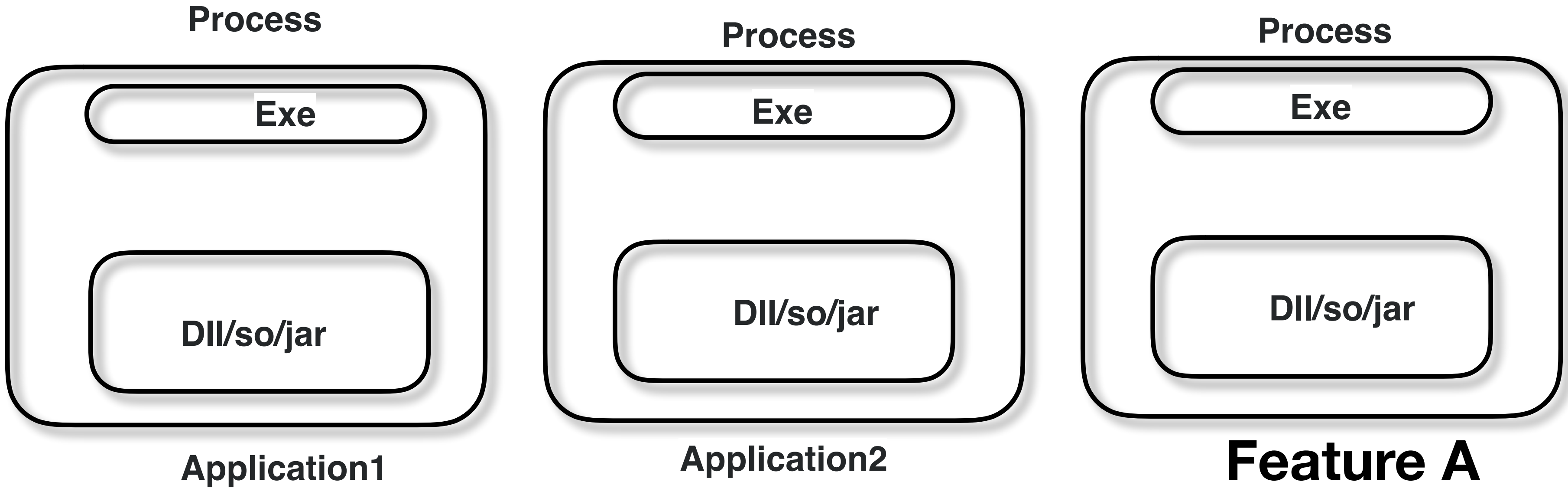
Dll/so/jar

Dll/so/jar

Feature A

Runtime monolithic

In process comp



Process

Exe

Dll/so/jar

Application1

Process

Exe

Dll/so/jar

Application2

Process

Exe

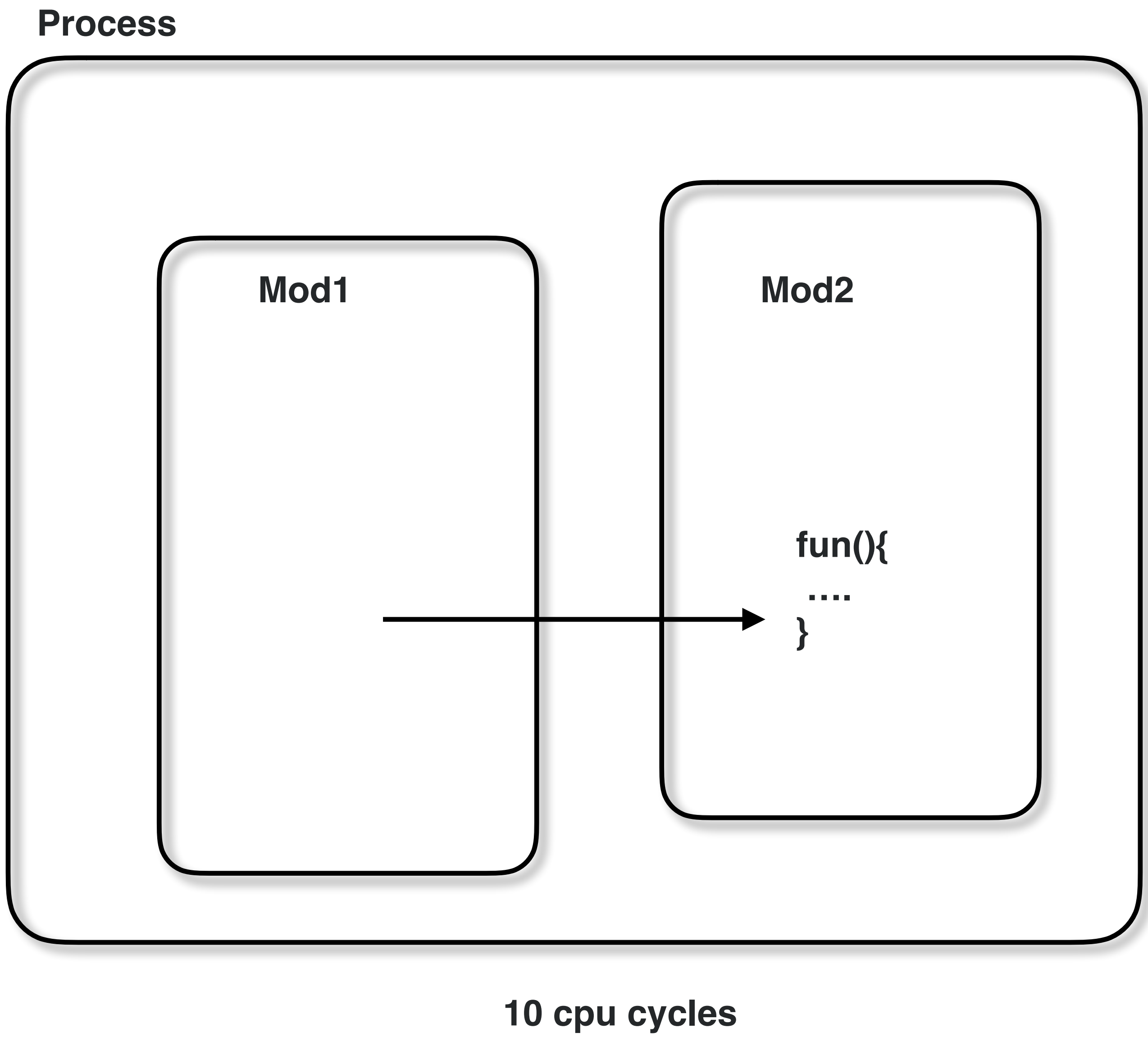
Dll/so/jar

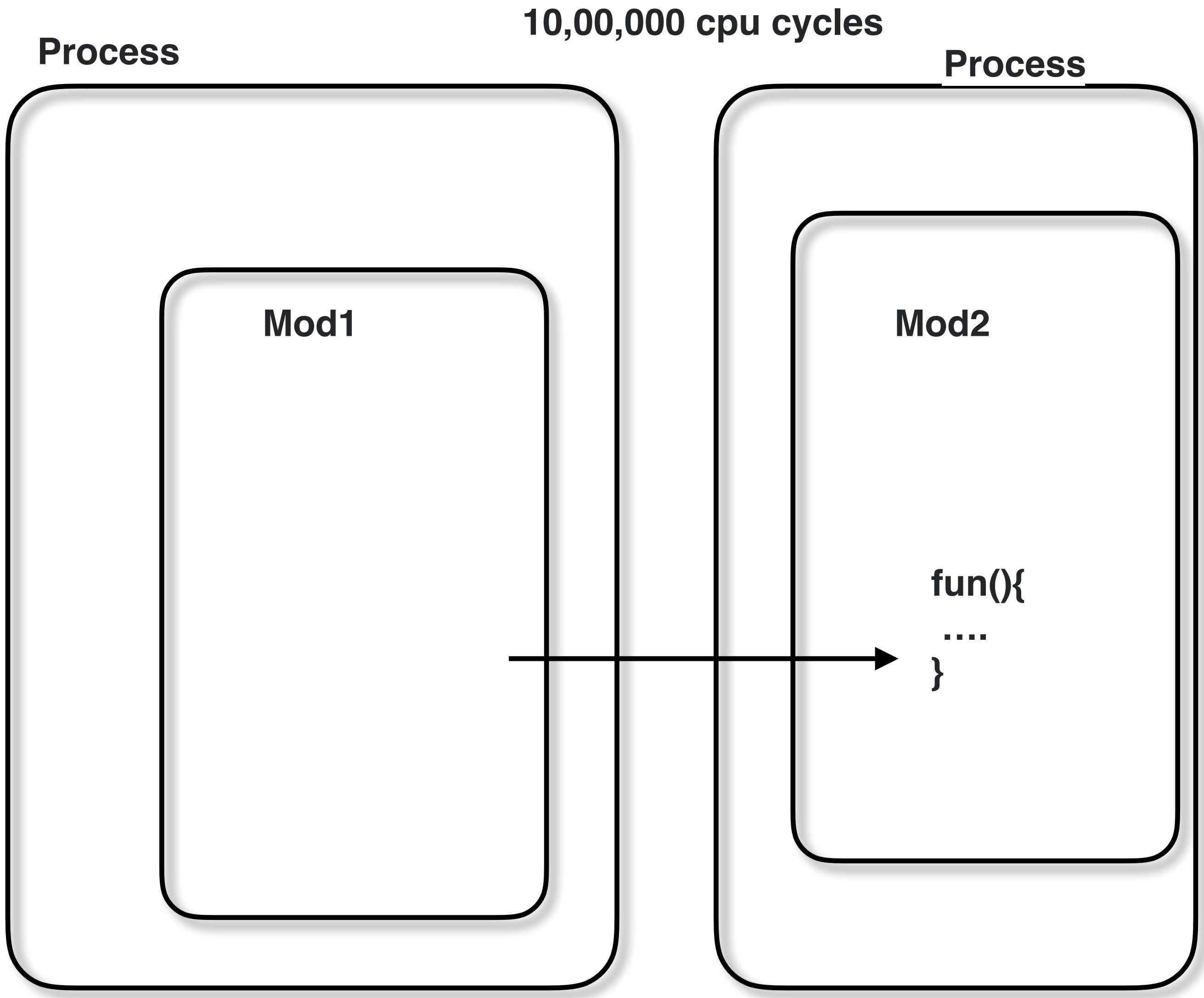
Feature A

Micro Application

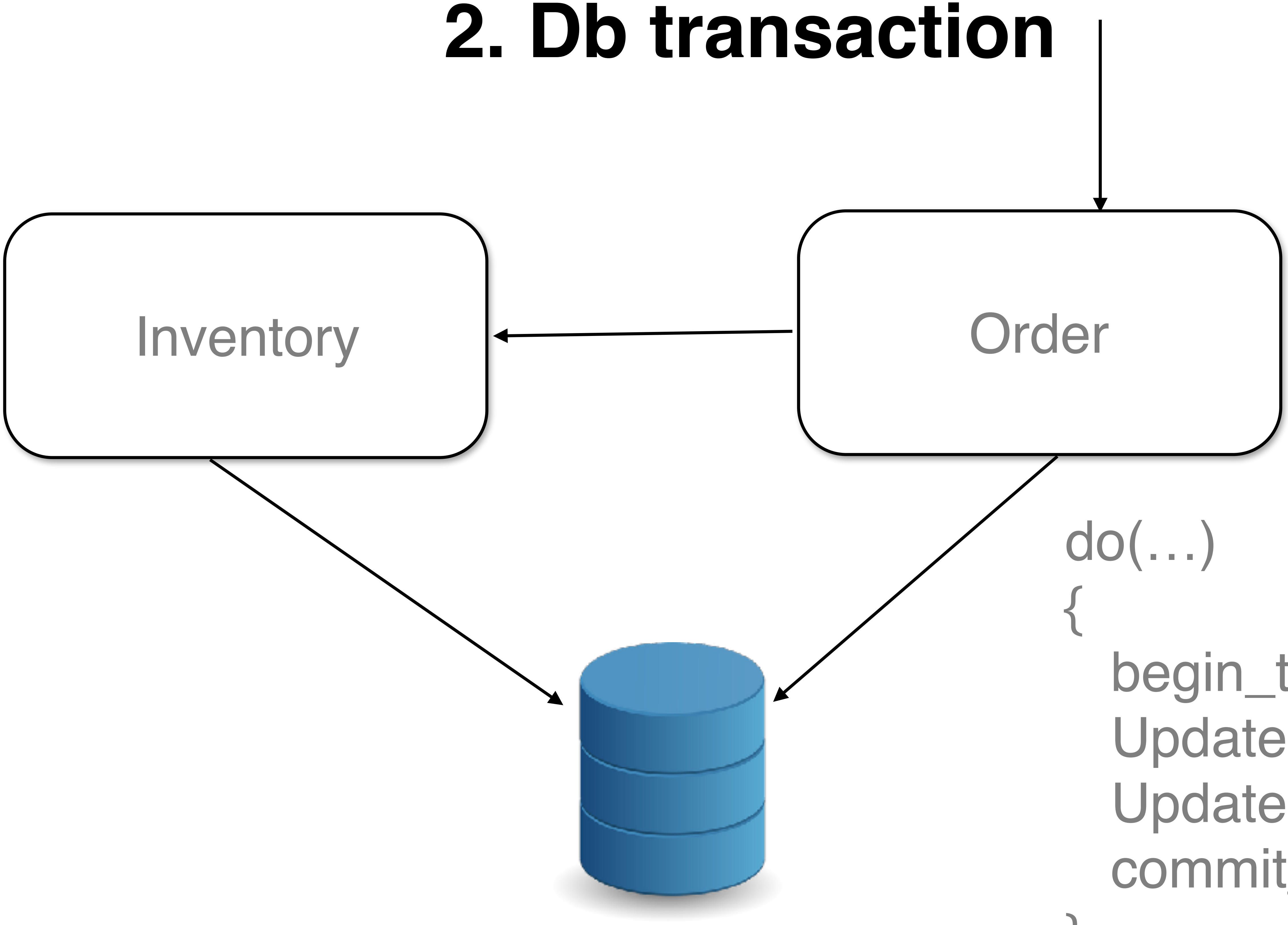
1. Performance

Operation	Cpu Cycles
• 10 + 12	3
• Calling a in-memory Method	10
• Create Thread	2,00,000
• Destroy Thread	1,00,000
• Database Call	40,00,000
• Distributed Fun Call	20,00,000
Write to disk	10,00,000


Process
Mod1
Mod2
fun(){
....
}
10 cpu cycles

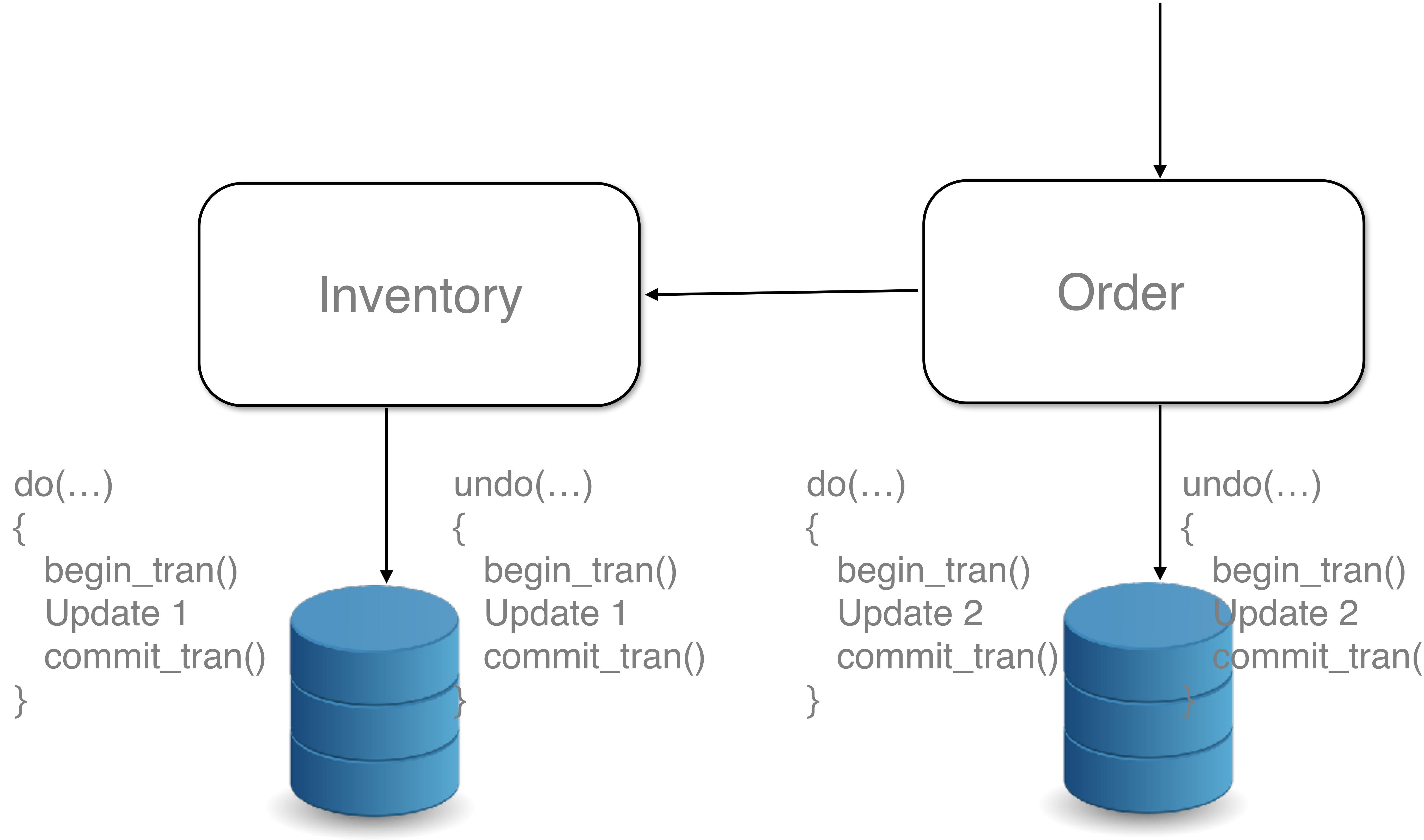

10,00,000 cpu cycles
Process
Process
Mod1
Mod2
fun(){
....
}

2. Db transaction



```
do(…)
{
   begin_tran()
   Update 1
   Update 2
   commit_tran()
}
```

2. Db transaction

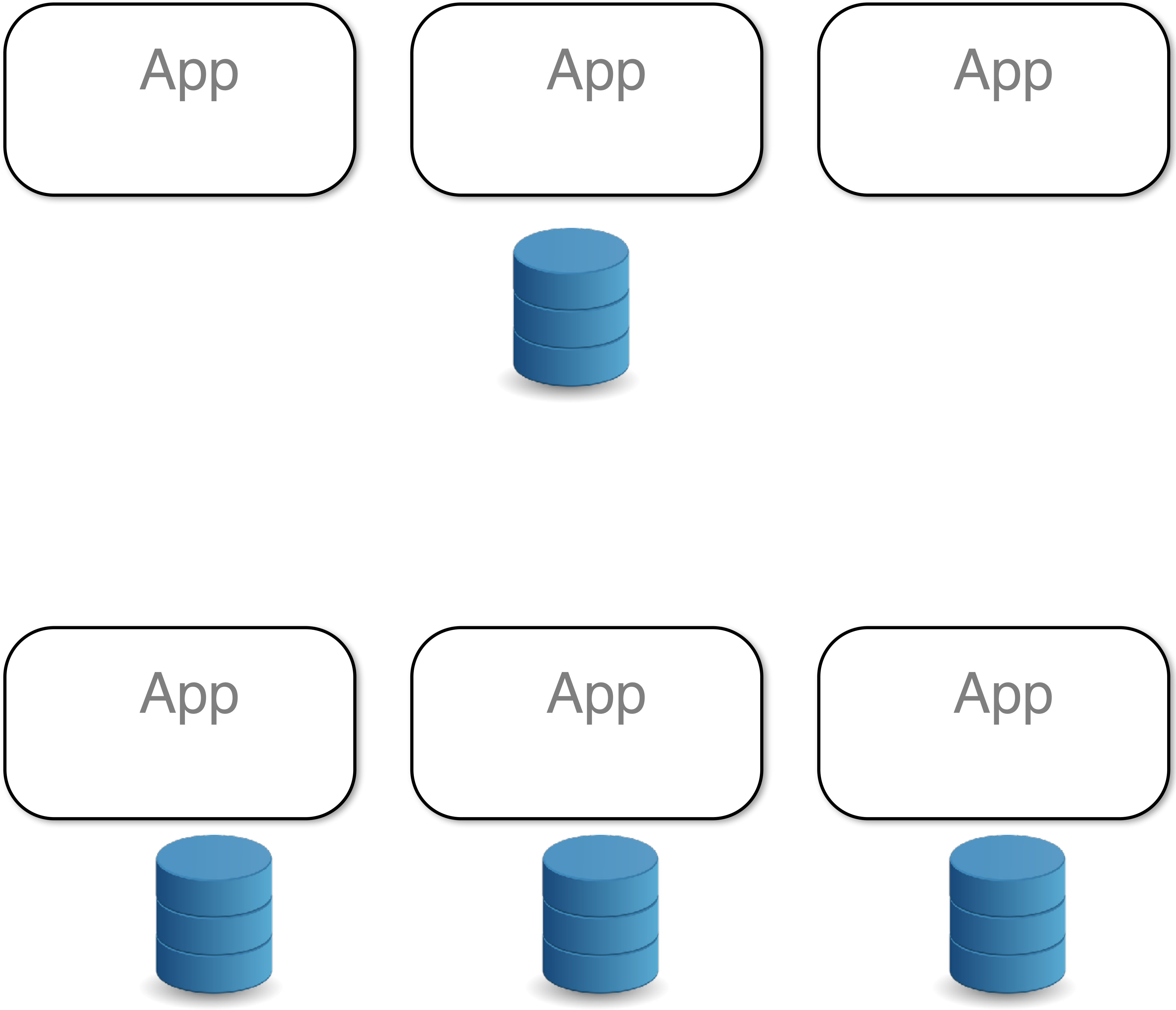


```
do(…)
{
  begin_tran()
  Update 1
  commit_tran()
}
```

```
undo(…)
{
  begin_tran()
  Update 1
  commit_tran()
}
```

```
do(…)
{
  begin_tran()
  Update 2
  commit_tran()
}
```

```
undo(…)
{
  begin_tran()
  Update 2
  commit_tran(
}
```



App

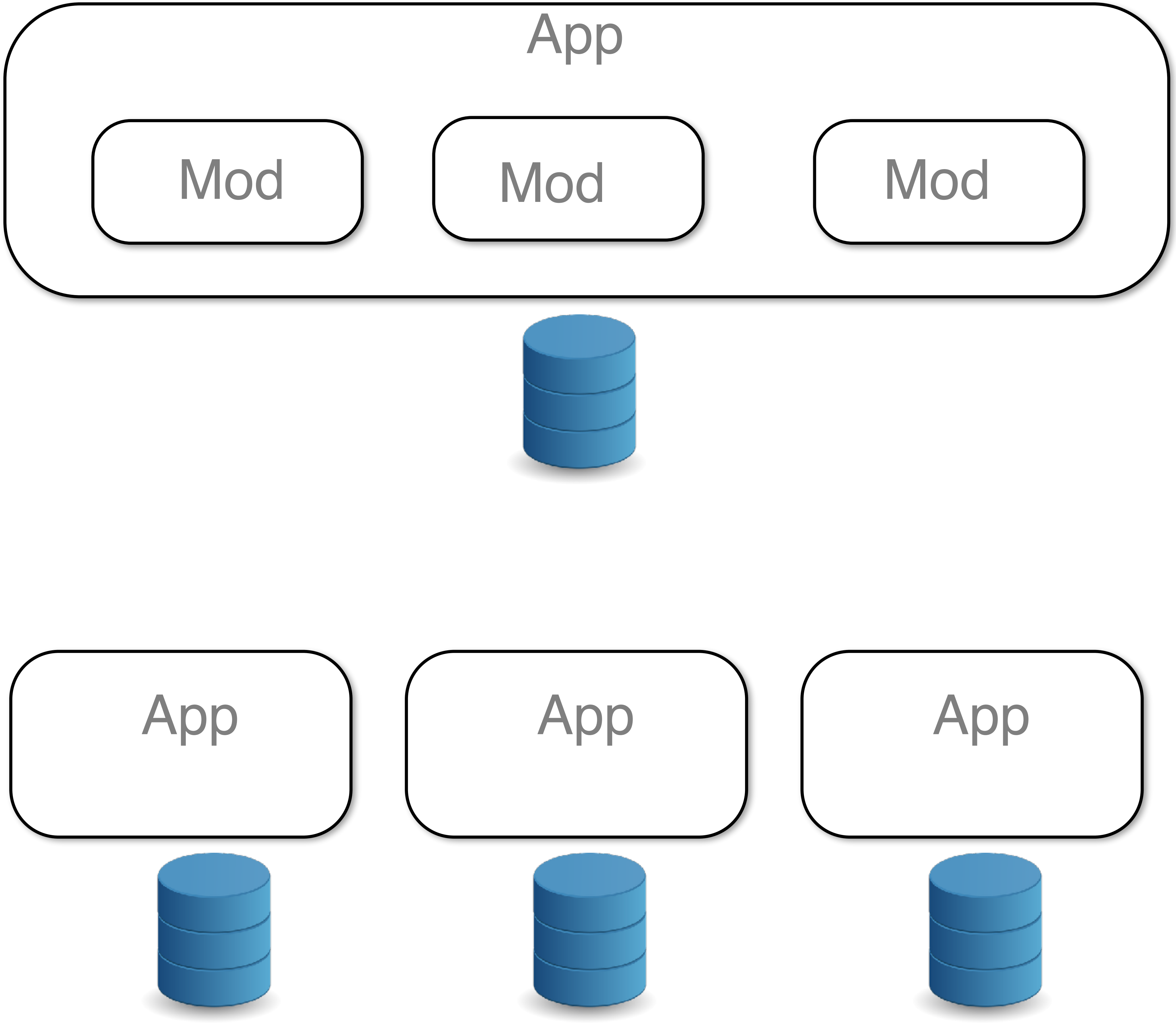
App

App

App

App

App



App

Mod

Mod

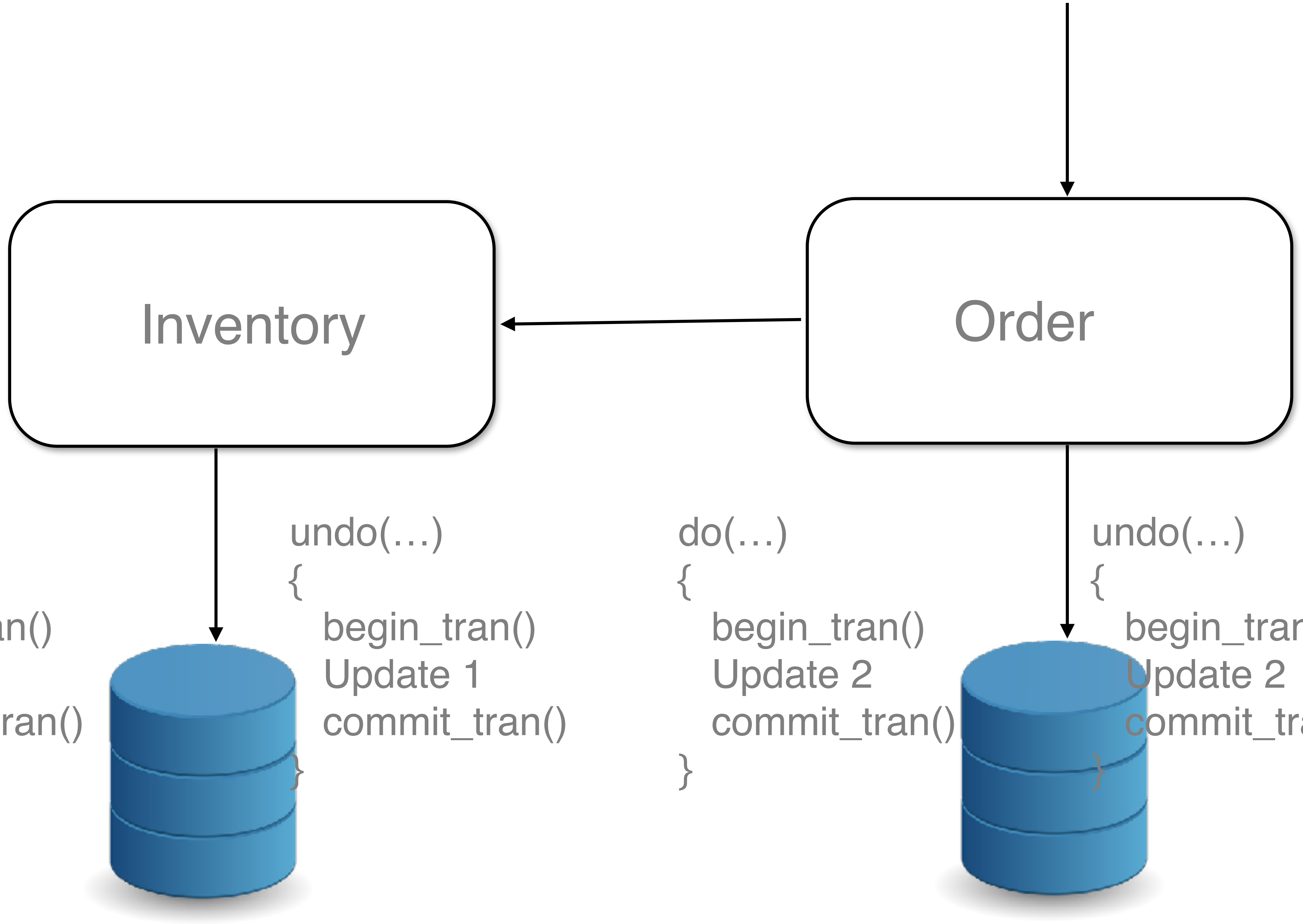
Mod

App

App

App

2. Db transaction



```
do(…)
{
  begin_tran()
  Update 1
  commit_tran()
}
```

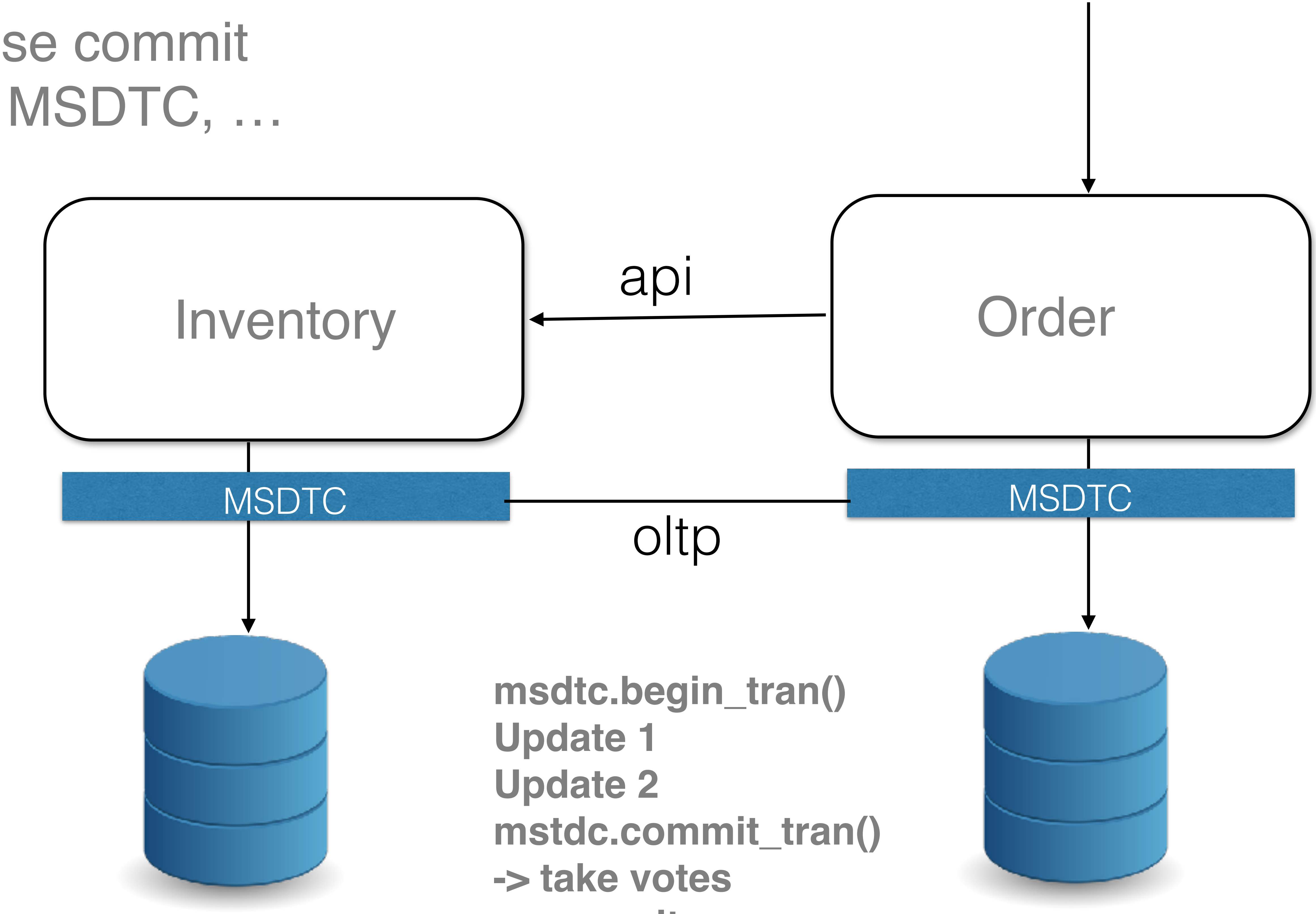
```
undo(…)
{
  begin_tran()
  Update 1
  commit_tran()
}
```

```
do(…)
{
  begin_tran()
  Update 2
  commit_tran()
}
```

```
undo(…)
{
  begin_tran()
  Update 2
  commit_tran(
}
```

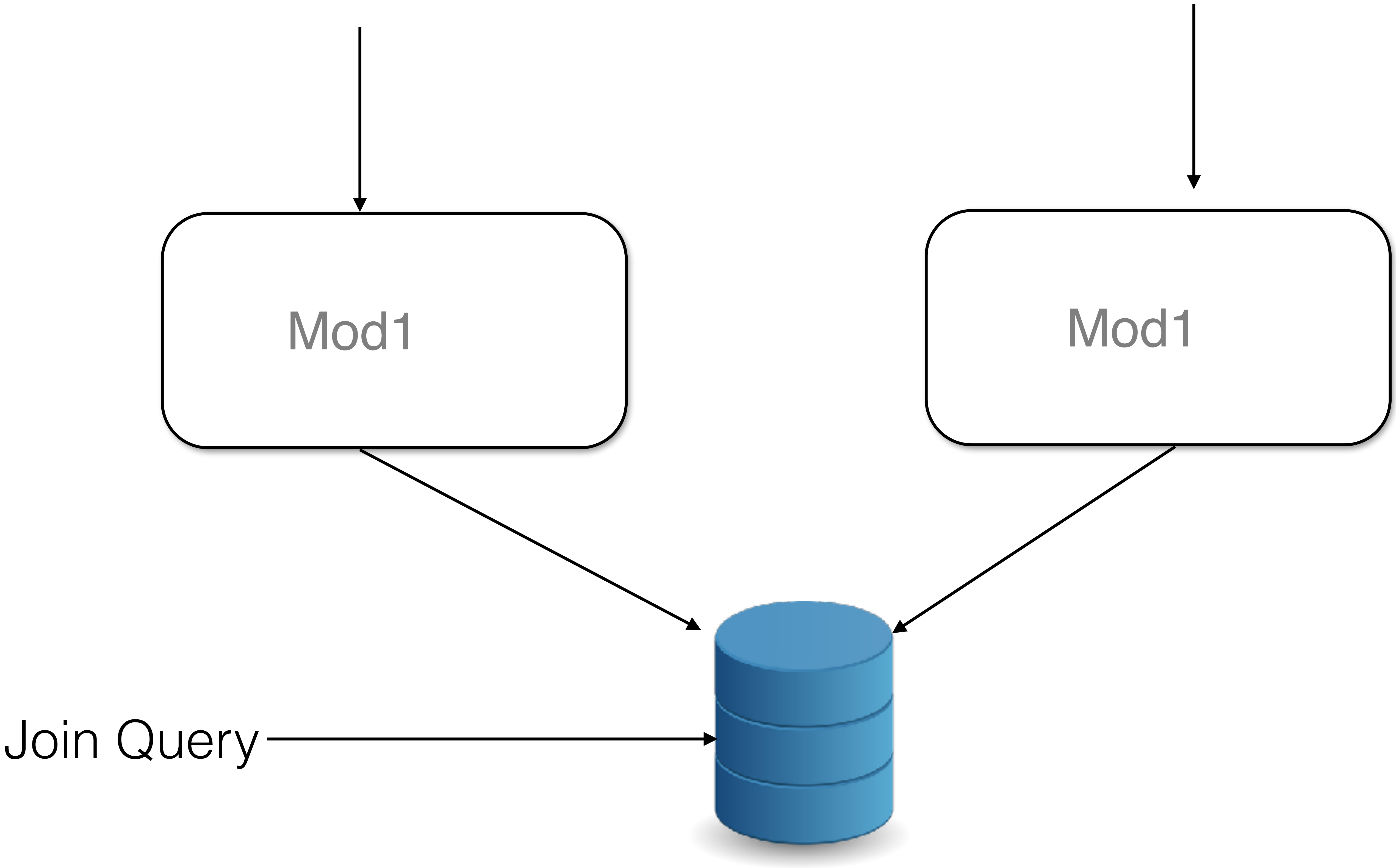
2. Db transaction

2 phase commit
JTX , MSDTC, …



```
msdtc.begin_tran()
Update 1
Update 2
mstdc.commit_tran()
-> take votes
-> commit
```

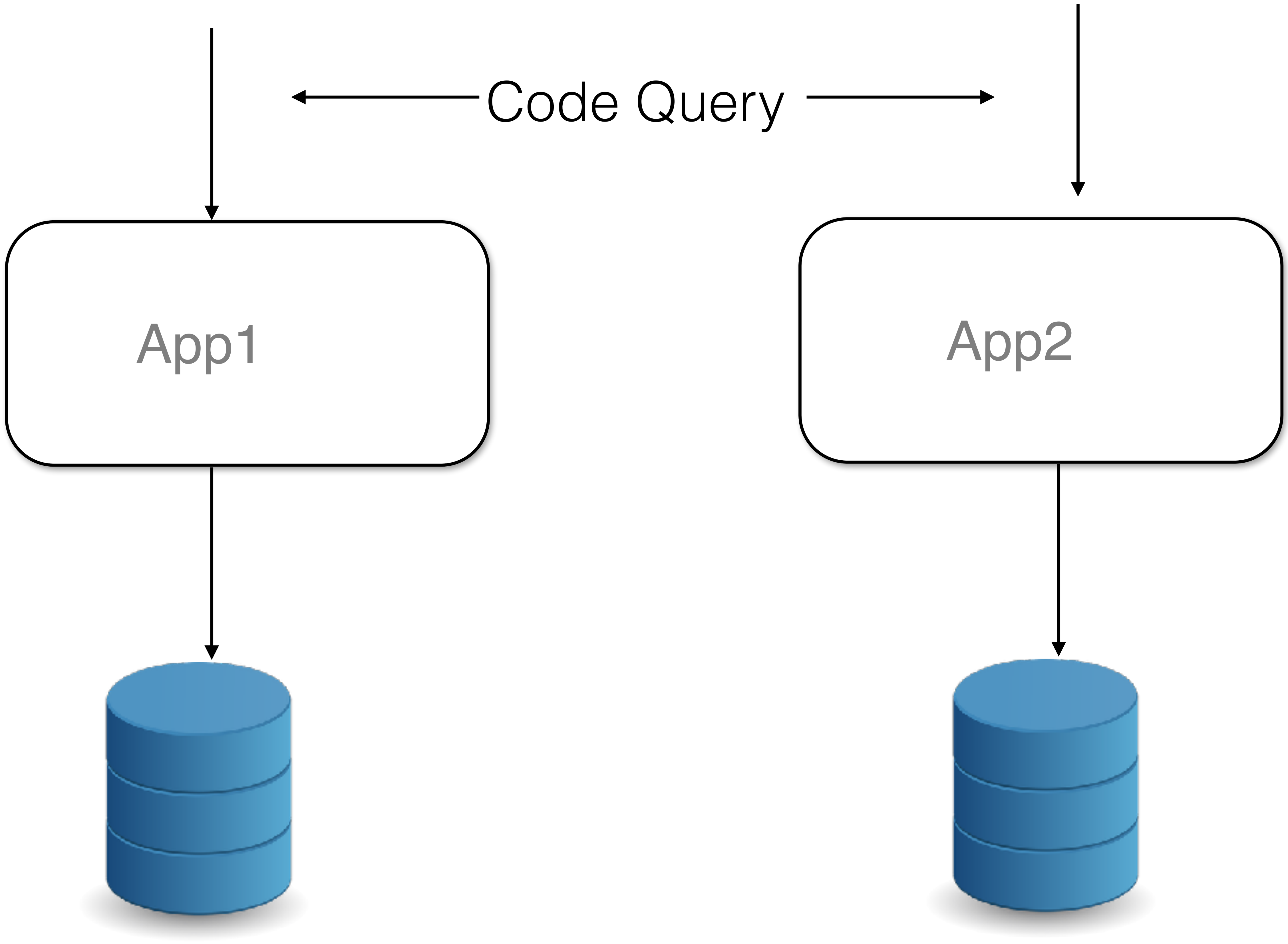
3. Db query


Mod1

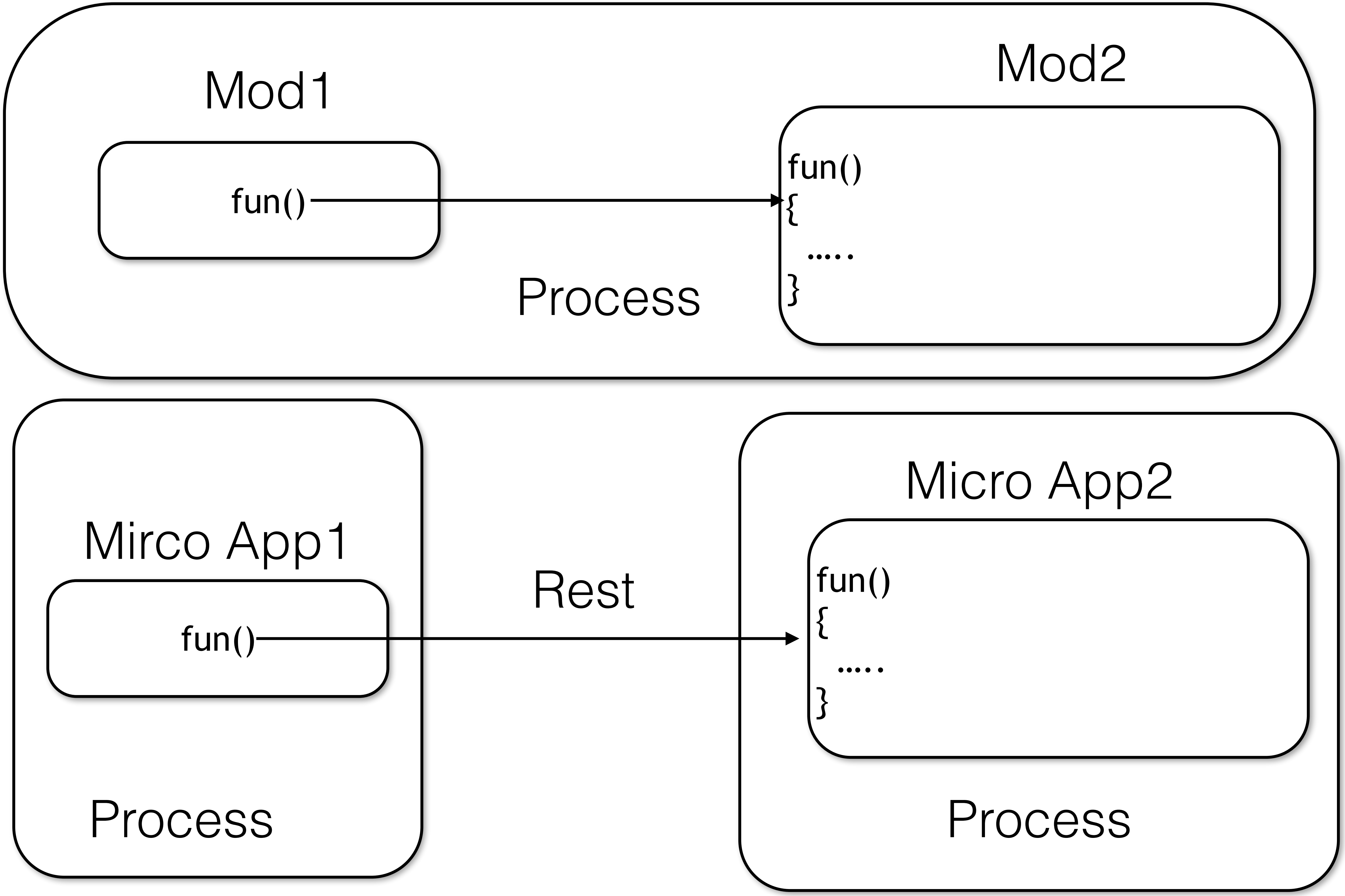
Mod1

Join Query

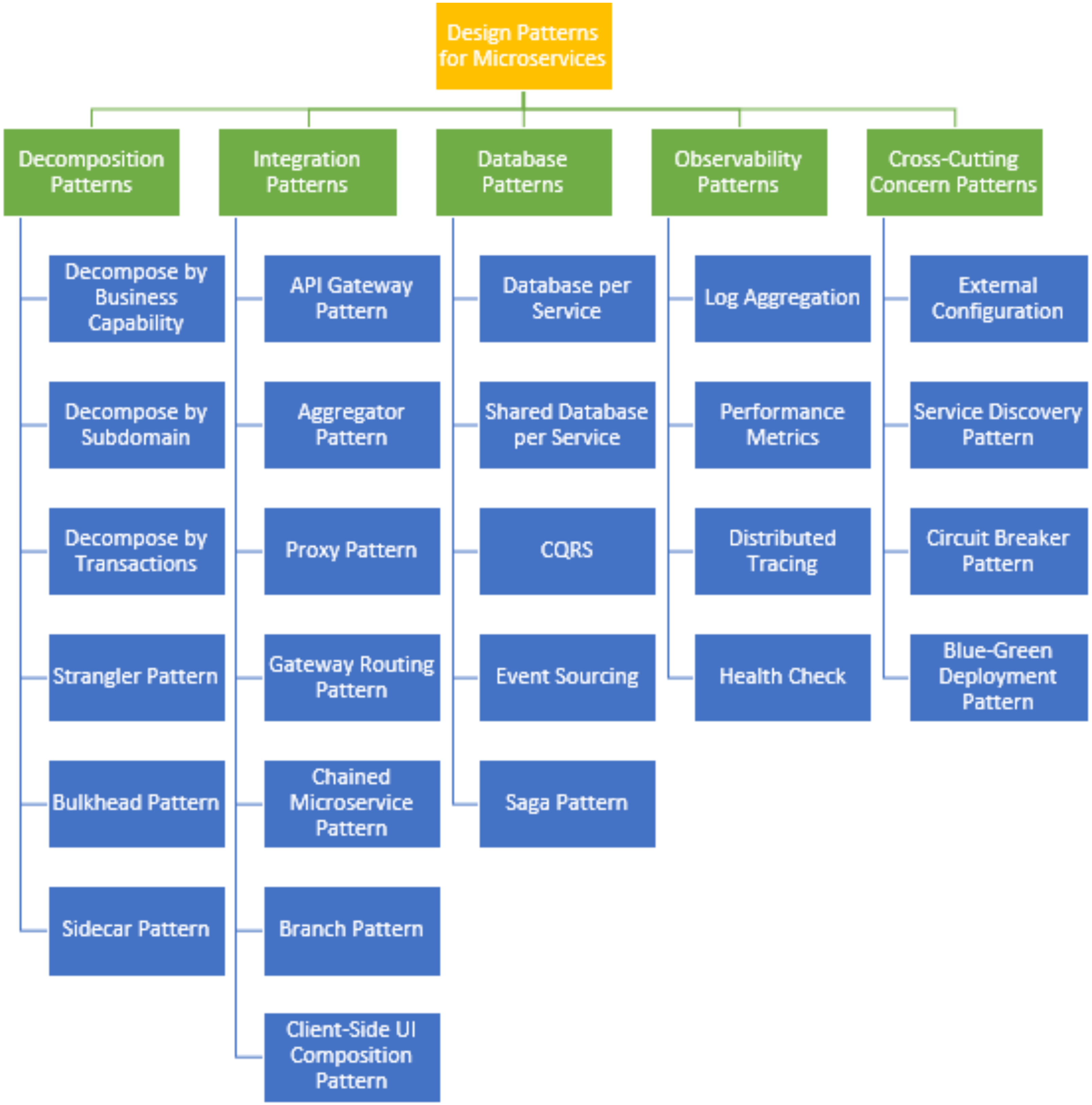
3. Query



4. Development time



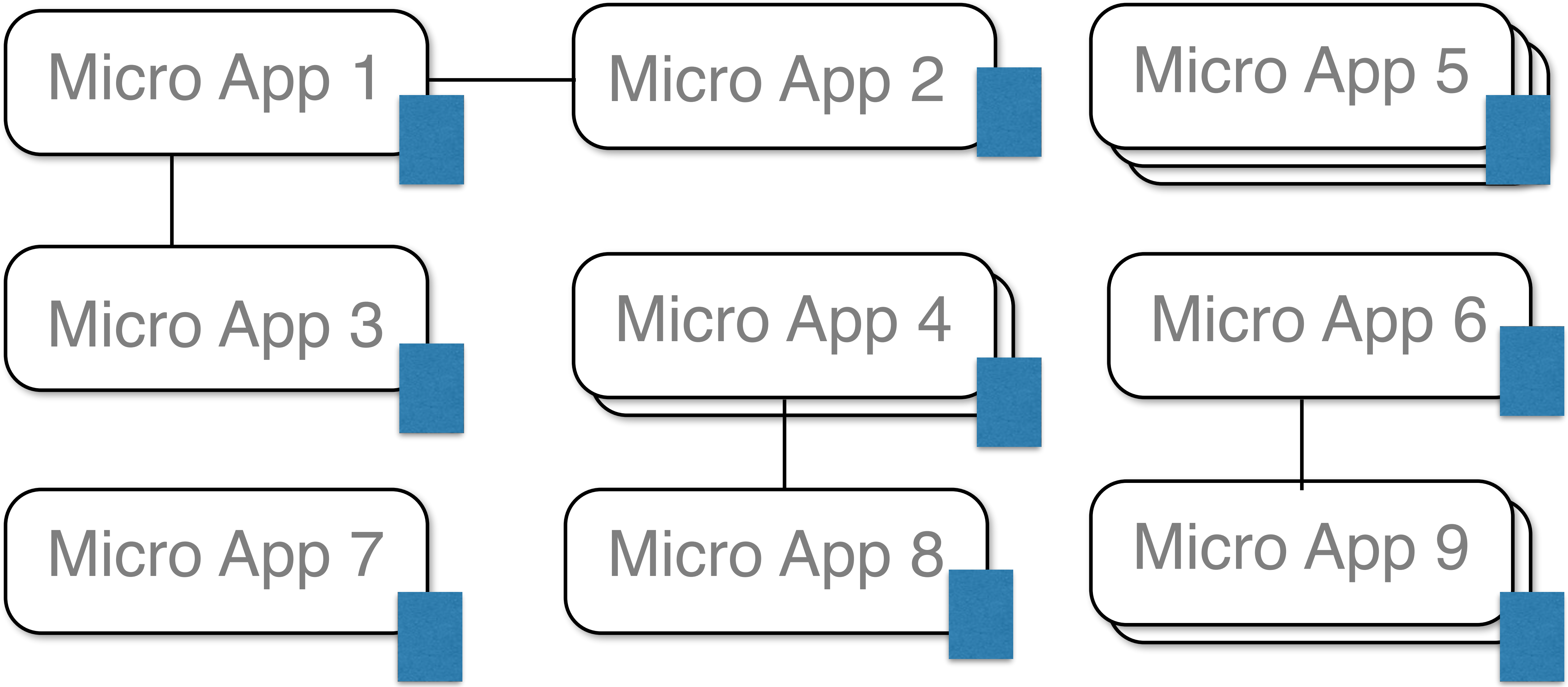
5. Learning Curve



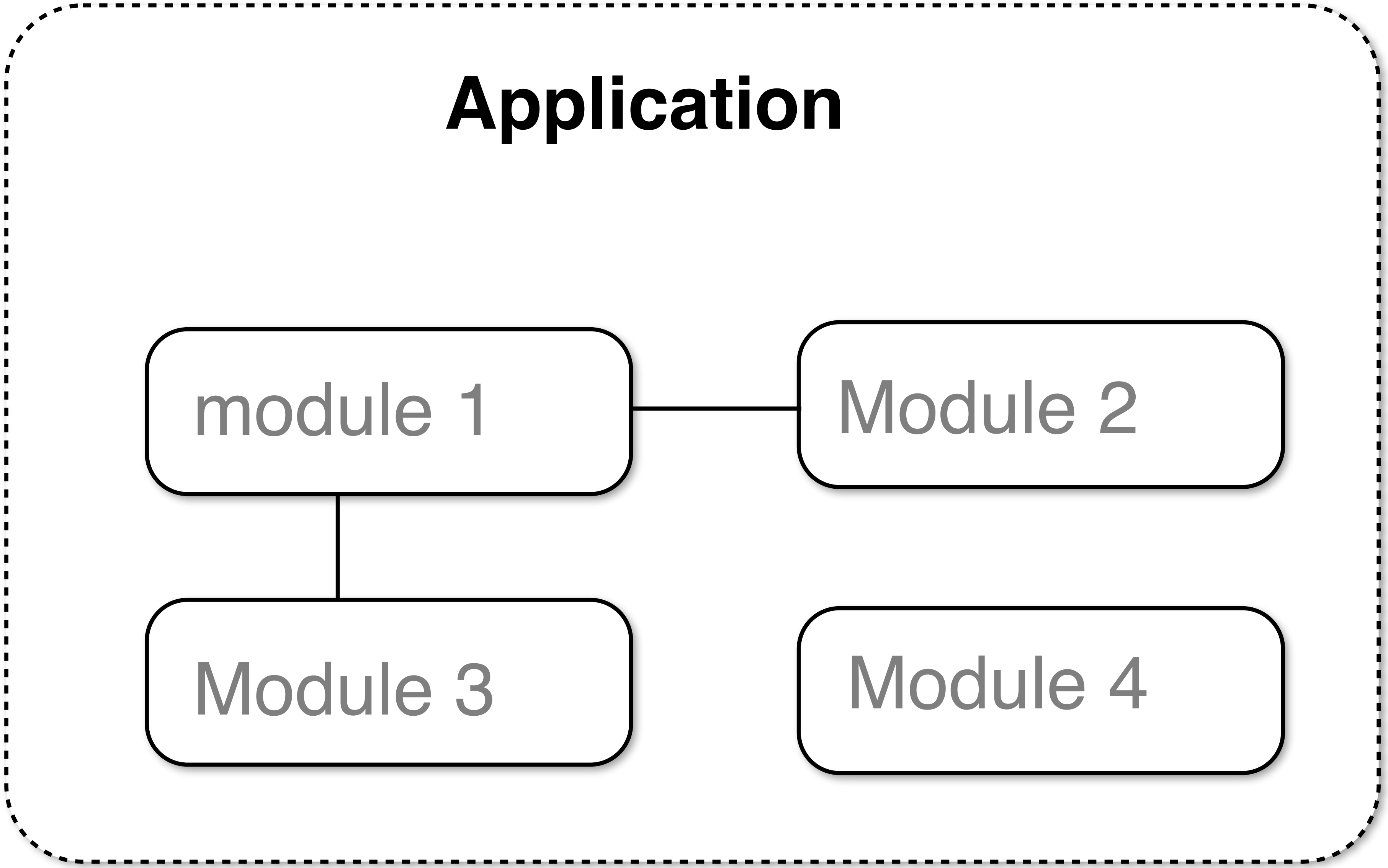
Design Patterns for Microservices

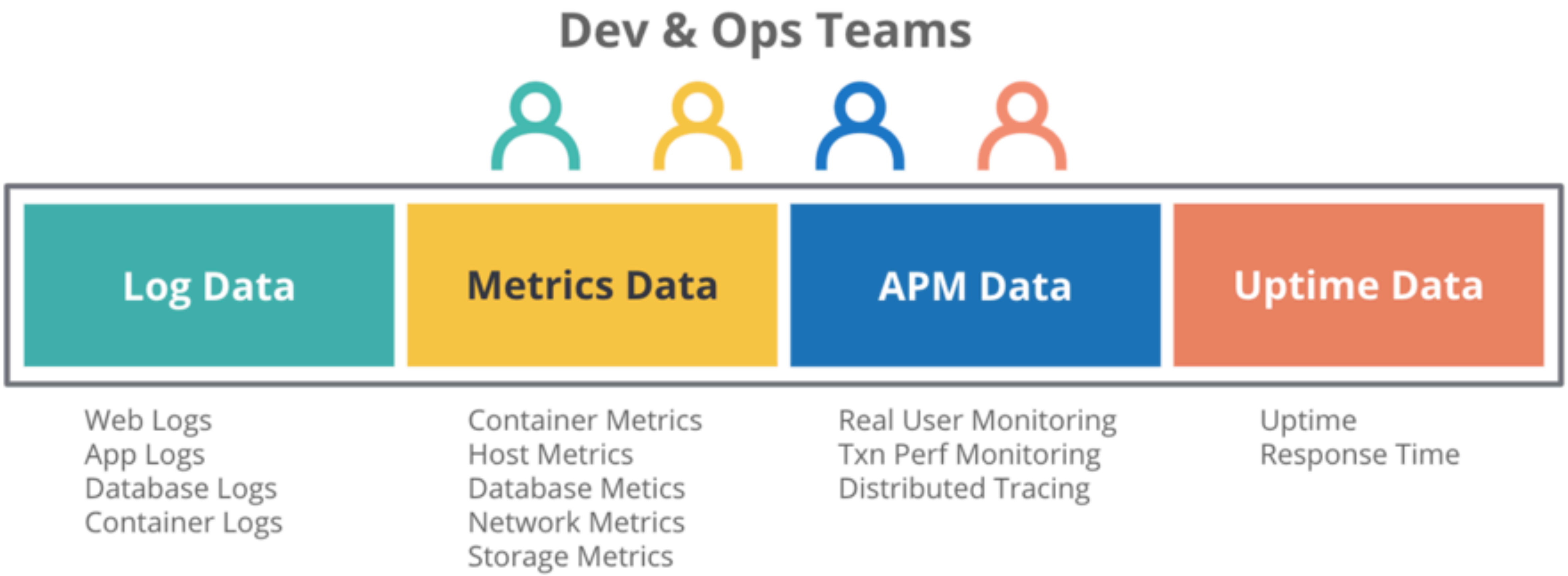
- Decomposition Patterns
  - Decompose by Business Capability
  - Decompose by Subdomain
  - Decompose by Transactions
  - Strangler Pattern
  - Bulkhead Pattern
  - Sidecar Pattern
- Integration Patterns
  - API Gateway Pattern
  - Aggregator Pattern
  - Proxy Pattern
  - Gateway Routing Pattern
  - Chained Microservice Pattern
  - Branch Pattern
  - Client-Side UI Composition Pattern
- Database Patterns
  - Database per Service
  - Shared Database per Service
  - CQRS
  - Event Sourcing
  - Saga Pattern
- Observability Patterns
  - Log Aggregation
  - Performance Metrics
  - Distributed Tracing
  - Health Check
- Cross-Cutting Concern Patterns
  - External Configuration
  - Service Discovery Pattern
  - Circuit Breaker Pattern
  - Blue-Green Deployment Pattern

6. Infra Cost


Micro App 1
Micro App 2
Micro App 5
Micro App 3
Micro App 4
Micro App 6
Micro App 7
Micro App 8
Micro App 9

7. Debugging





Dev & Ops Teams

Log Data

- Web Logs
- App Logs
- Database Logs
- Container Logs

Metrics Data

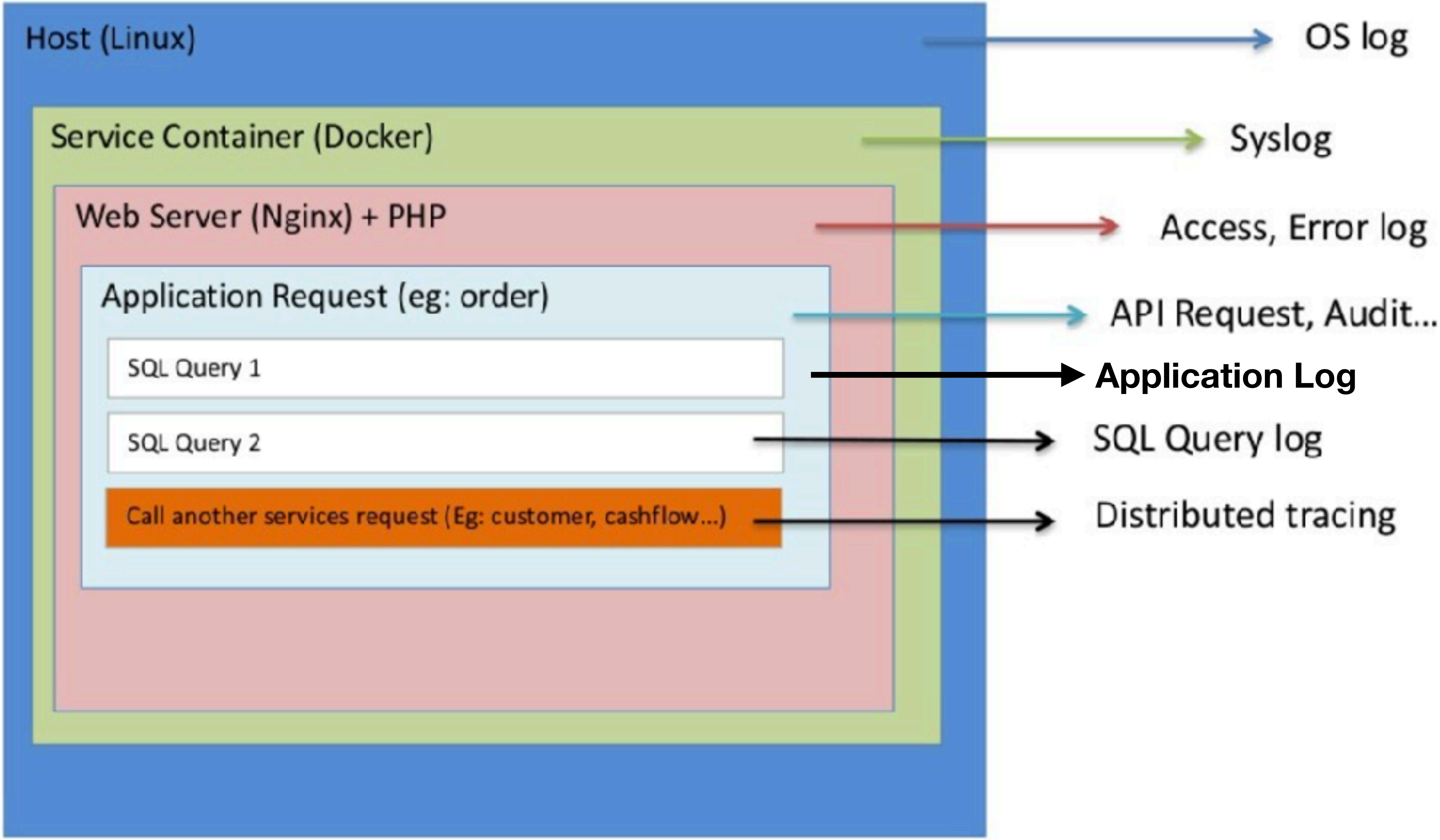
- Container Metrics
- Host Metrics
- Database Metics
- Network Metrics
- Storage Metrics

APM Data

- Real User Monitoring
- Txn Perf Monitoring
- Distributed Tracing

Uptime Data

- Uptime
- Response Time



Host (Linux)

Service Container (Docker)

Web Server (Nginx) + PHP

Application Request (eg: order)

SQL Query 1

SQL Query 2

Call another services request (Eg: customer, cashflow...)

OS log

Syslog

Access, Error log

API Request, Audit...

Application Log

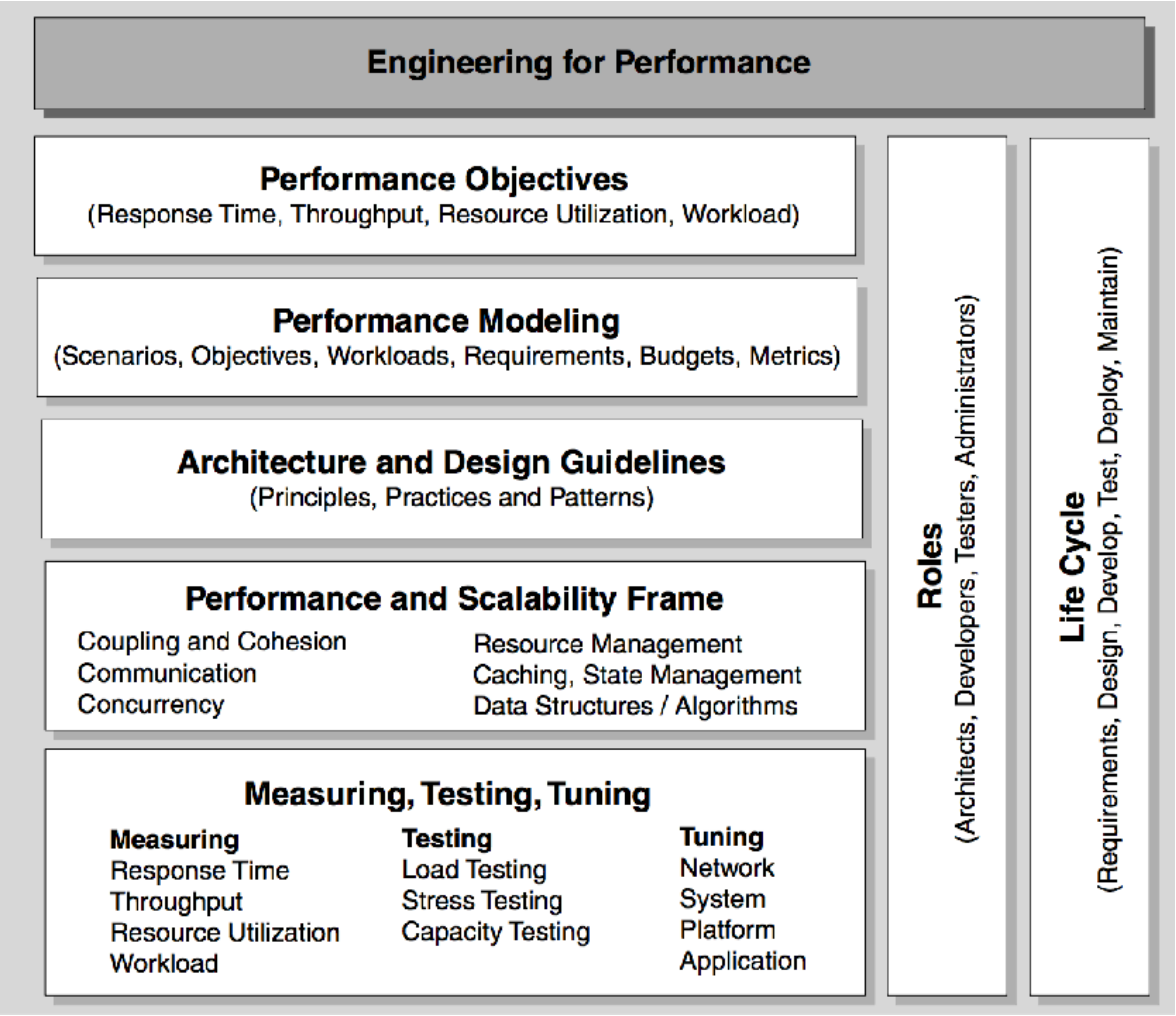
SQL Query log

Distributed tracing

Part IV

Addressing Performance Scenarios
# Performance Engineering
# CPU Performance patterns
# I/O Performance patterns
# Memory performance patterns
# Performance anti patterns
Lab : Building Architecture for a Desktop Application

Performance View



Engineering for Performance

Performance Objectives
(Response Time, Throughput, Resource Utilization, Workload)

Performance Modeling
(Scenarios, Objectives, Workloads, Requirements, Budgets, Metrics)

Architecture and Design Guidelines
(Principles, Practices and Patterns)

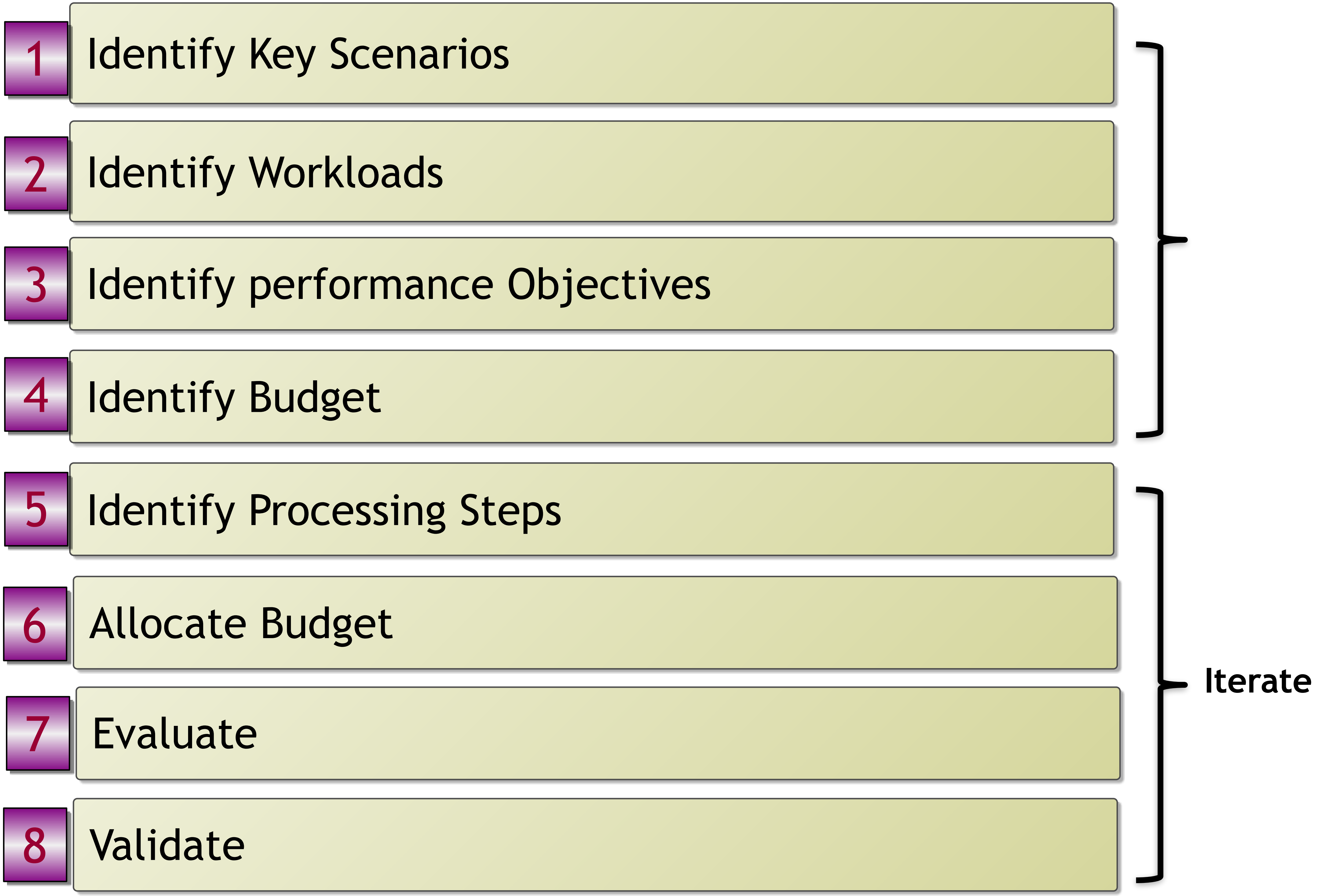
Performance and Scalability Frame
Coupling and Cohesion
Communication
Concurrency
Resource Management
Caching, State Management
Data Structures / Algorithms

Measuring, Testing, Tuning
Measuring
Response Time
Throughput
Resource Utilization
Workload
Testing
Load Testing
Stress Testing
Capacity Testing
Tuning
Network
System
Platform
Application

Roles
(Architects, Developers, Testers, Administrators)

Life Cycle
(Requirements, Design, Develop, Test, Deploy, Maintain)

Performance Modeling

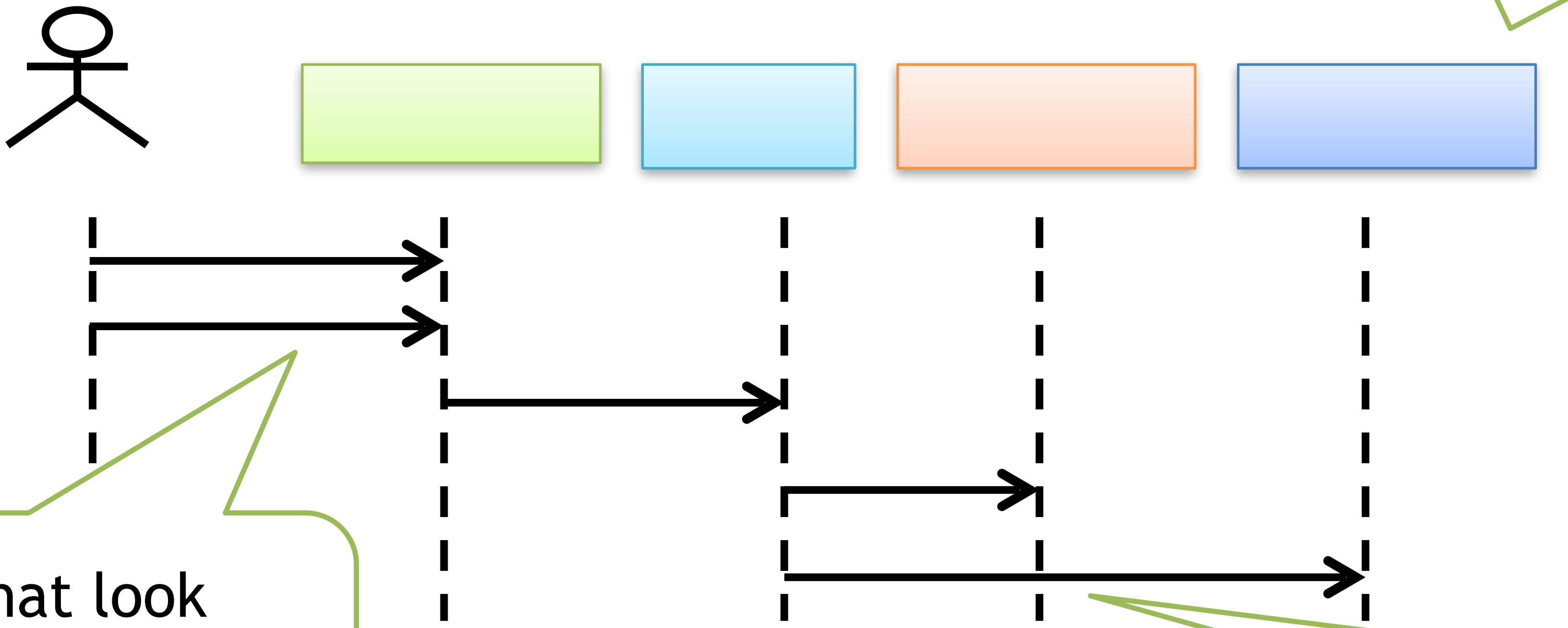


1. Identify Key Scenarios
2. Identify Workloads
3. Identify performance Objectives
4. Identify Budget
5. Identify Processing Steps
6. Allocate Budget
7. Evaluate
8. Validate

Iterate

Allocate Budget

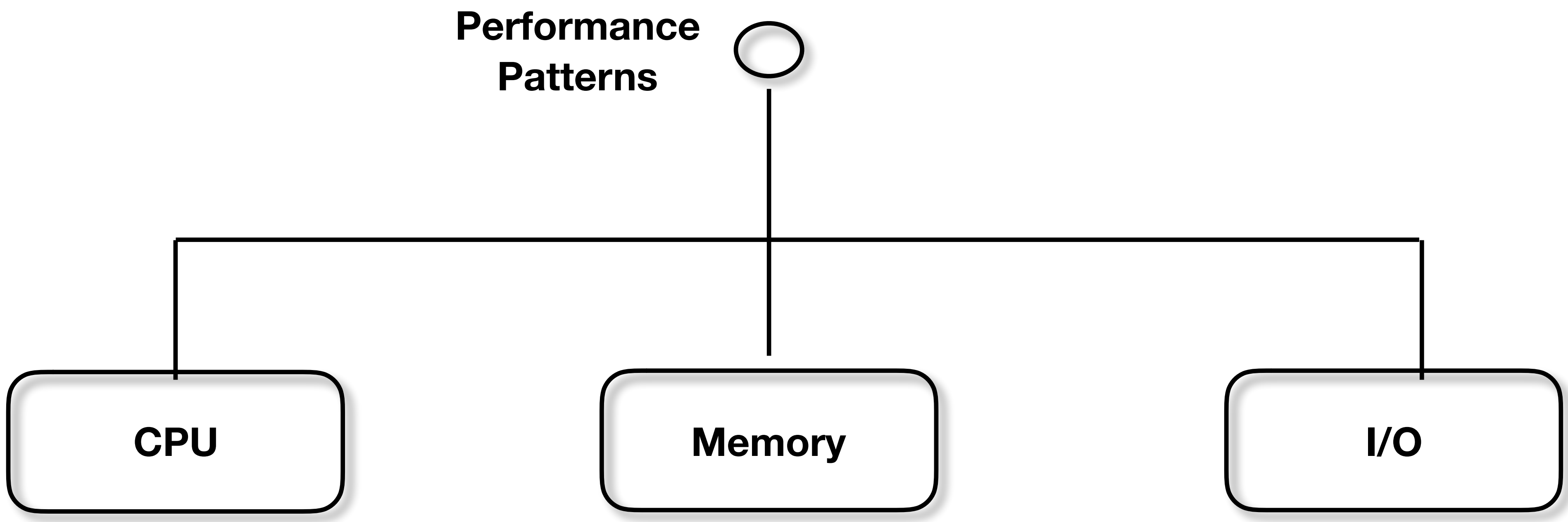
Know the cost of your materials.



For the ones that look risky, conduct some experiments (prototypes)

if you do not know how much time to assign, simply divide the total time equally between the steps.

Spread your budget across your processing steps



Performance Patterns

CPU

Memory

I/O

CPU

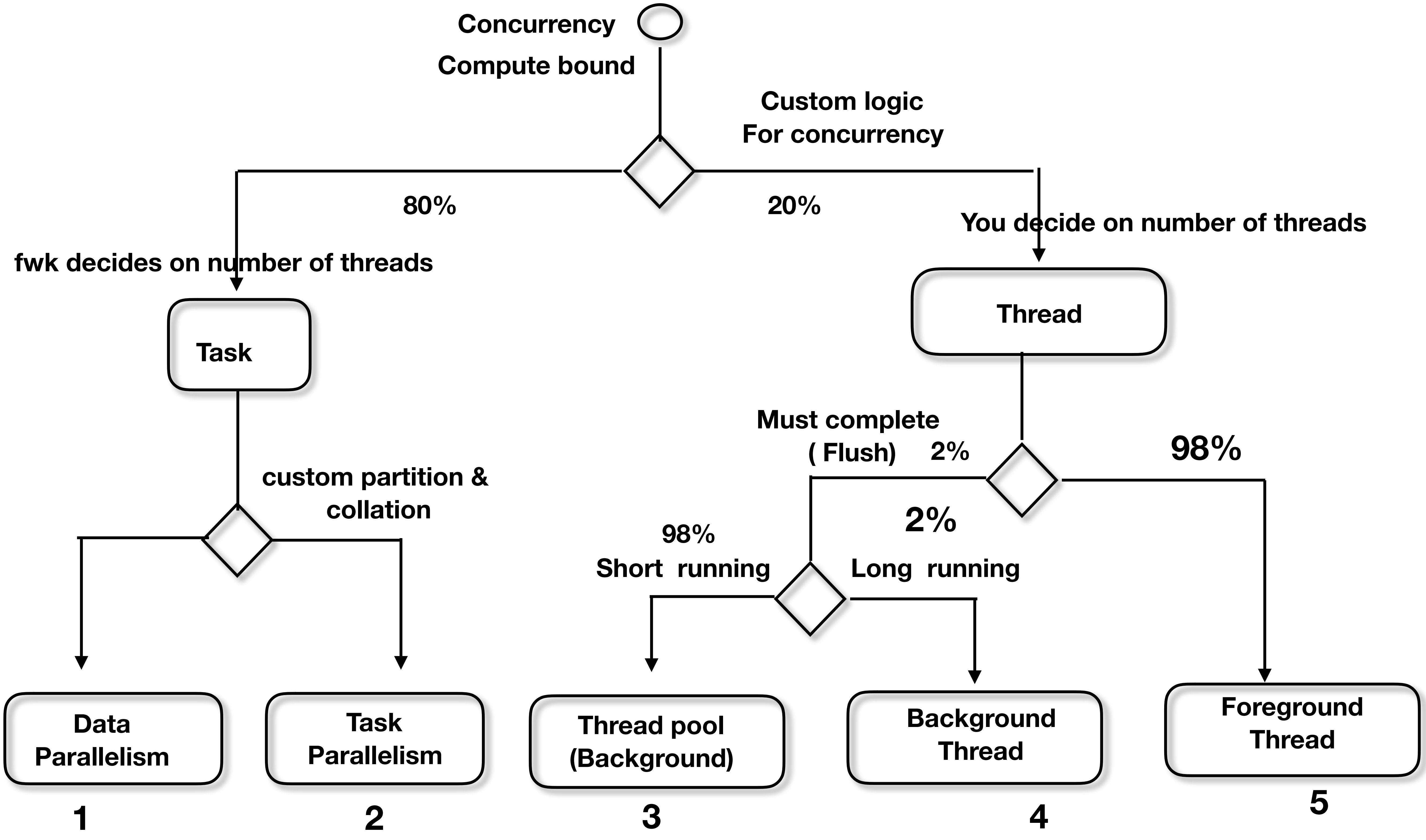
Eager Loading
Caching (Data, Output, Code)
Concurrency
Pooling (Object, Memory)
Queuing (Load Leveling)

Memory

Lazy Loading
GC
Fixed Size Memory Pool
Virtualization (Data, UI)
Singleton
Fly weight

I/O

Chunky / Batch
Compression
Async I/O
Pooling (Connection)
Acquire Late and Release Early



Concurrency

Compute bound

Custom logic
For concurrency

80%

20%

fwk decides on number of threads

You decide on number of threads

Task

Thread

custom partition &
collation

Must complete
(Flush) 2%

98%

2%

98%
Short running

Long running

Data
Parallelism

Task
Parallelism

Thread pool
(Background)

Background
Thread

Foreground
Thread

1

2

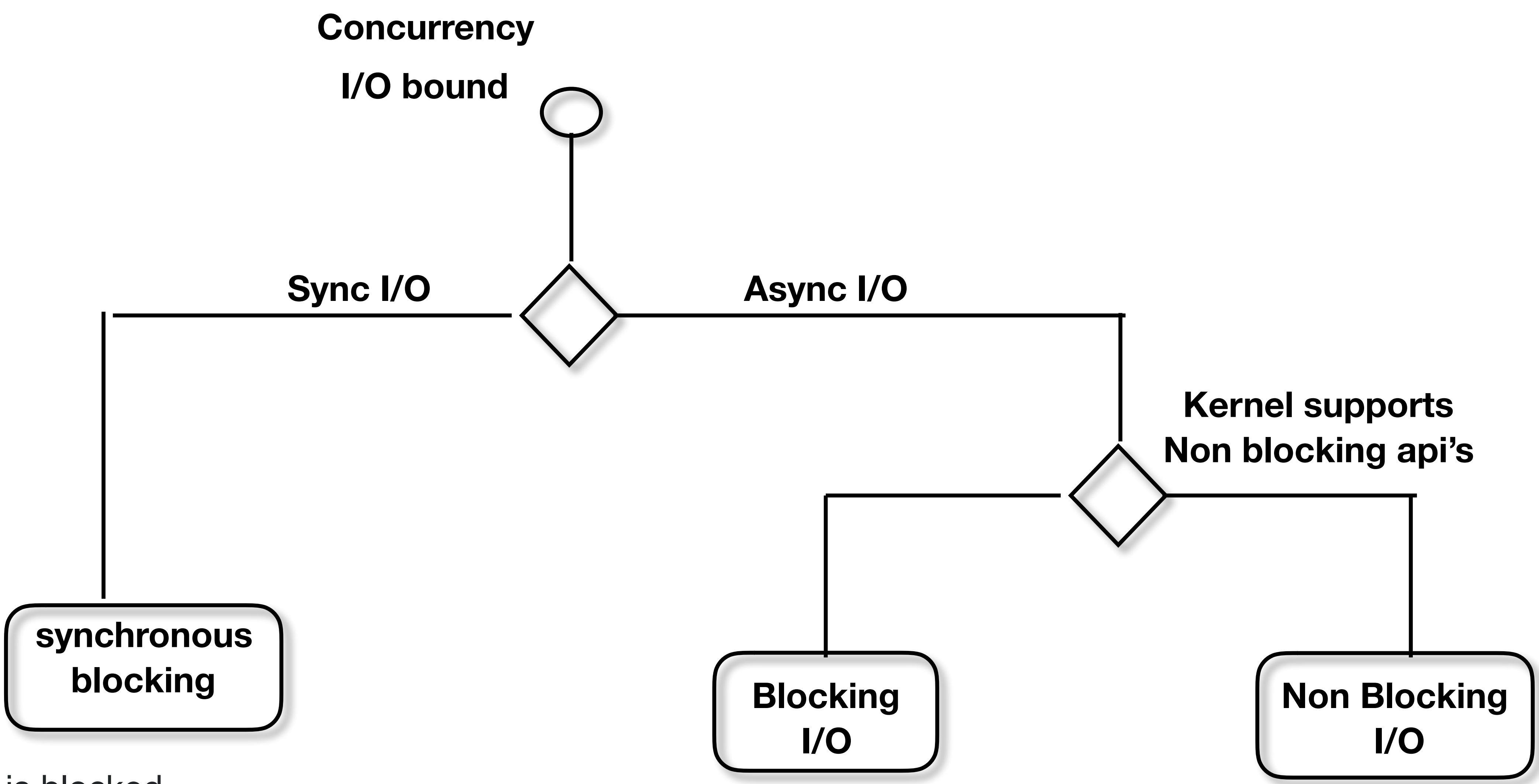
3

4

5

Imperative

Declarative



Concurrency

I/O bound

Sync I/O

Async I/O

Kernel supports Non blocking api's

synchronous blocking

Thread is blocked while hardware performs the work.

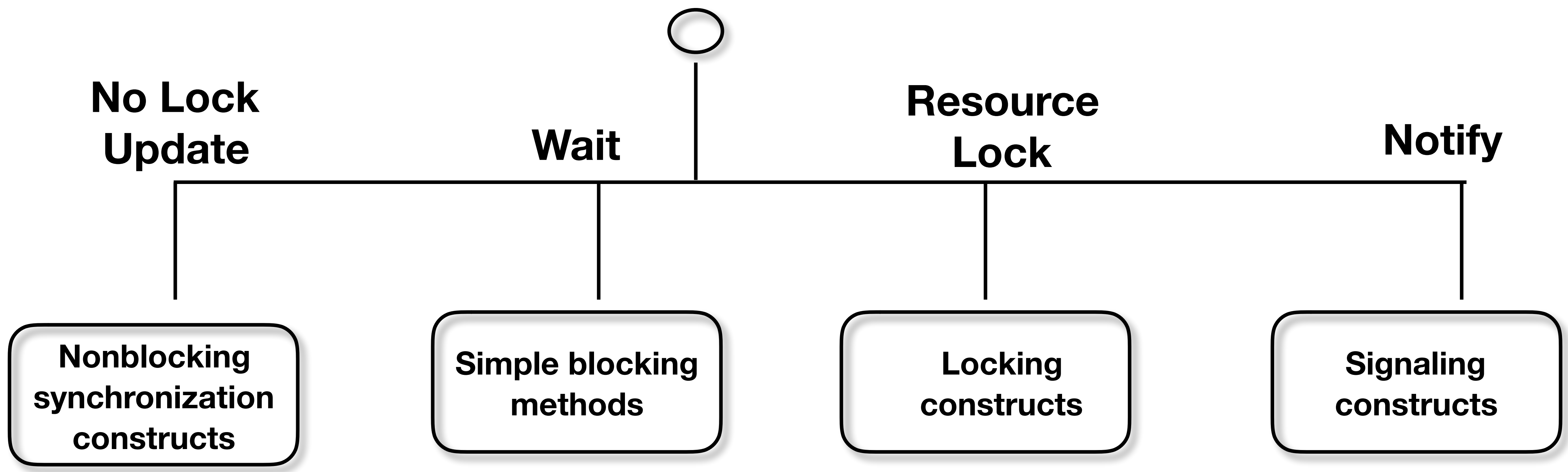
Blocking I/O

Thread is not blocked, But there is still a thread lower down the stack which is blocked on the IO

Non Blocking I/O

The kernel doesn't block your thread and returns from the IO call immediately.

Syncronization



No Lock Update

Nonblocking synchronization constructs

- volatile
- Atomic.

Wait

Simple blocking methods

- Sleep (bad)
- Join
- Spin

Resource Lock

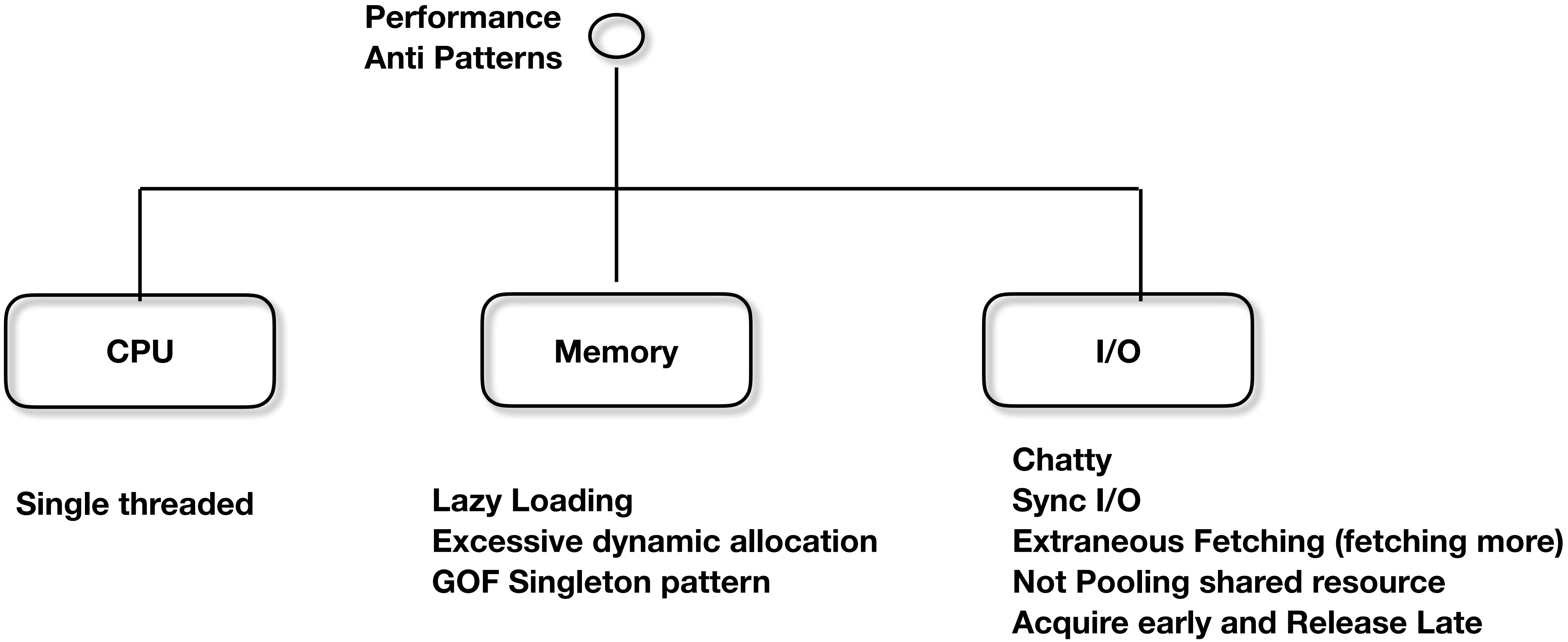
Locking constructs

- *Exclusive* locking
- nonexclusive locking
  - Semaphore
  - reader/writer locks.

Notify

Signaling constructs

- event wait
- Latch
- Barrier



Performance
Anti Patterns

CPU

Memory

I/O

Single threaded

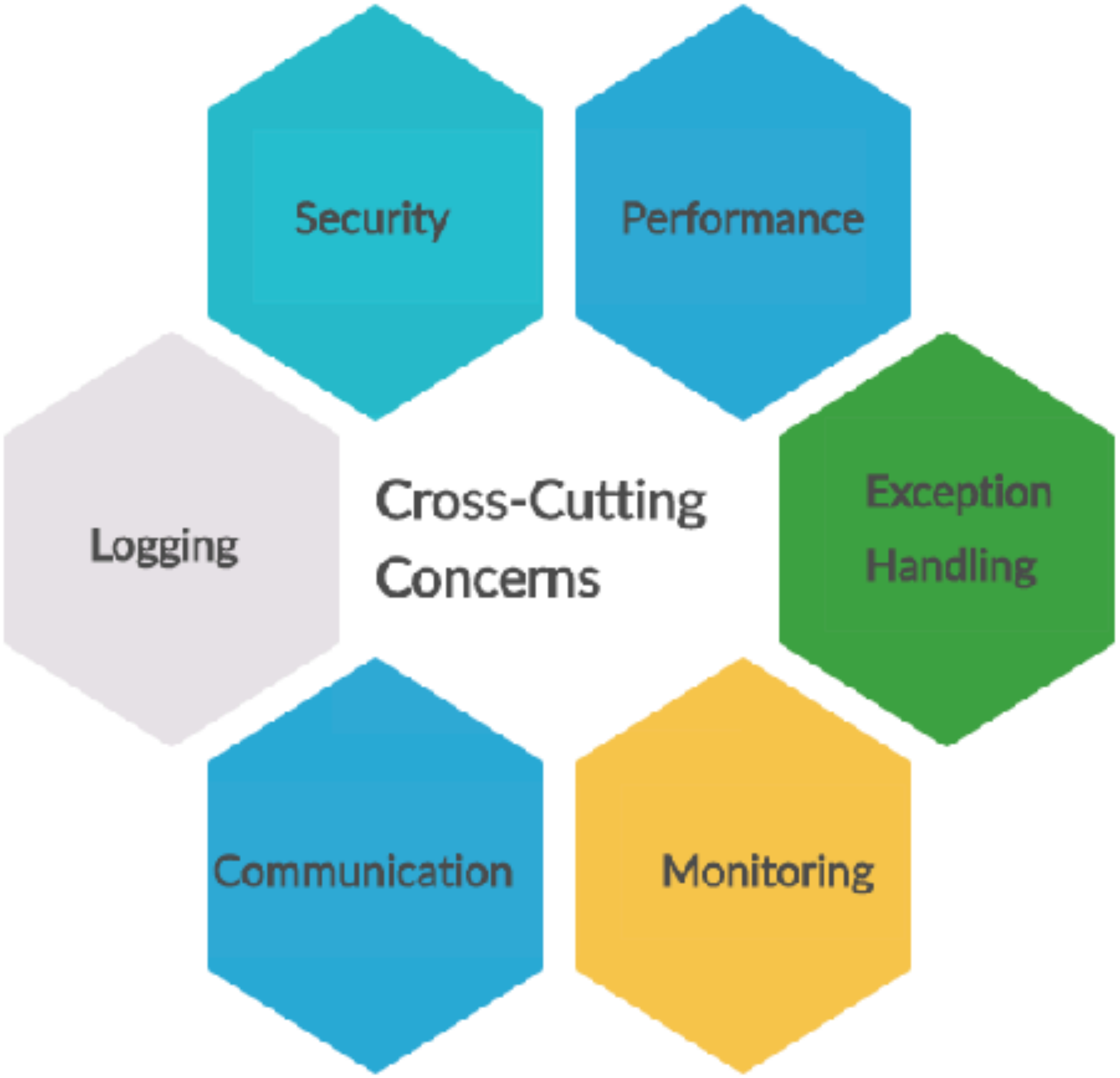
Lazy Loading
Excessive dynamic allocation
GOF Singleton pattern

Chatty
Sync I/O
Extraneous Fetching (fetching more)
Not Pooling shared resource
Acquire early and Release Late

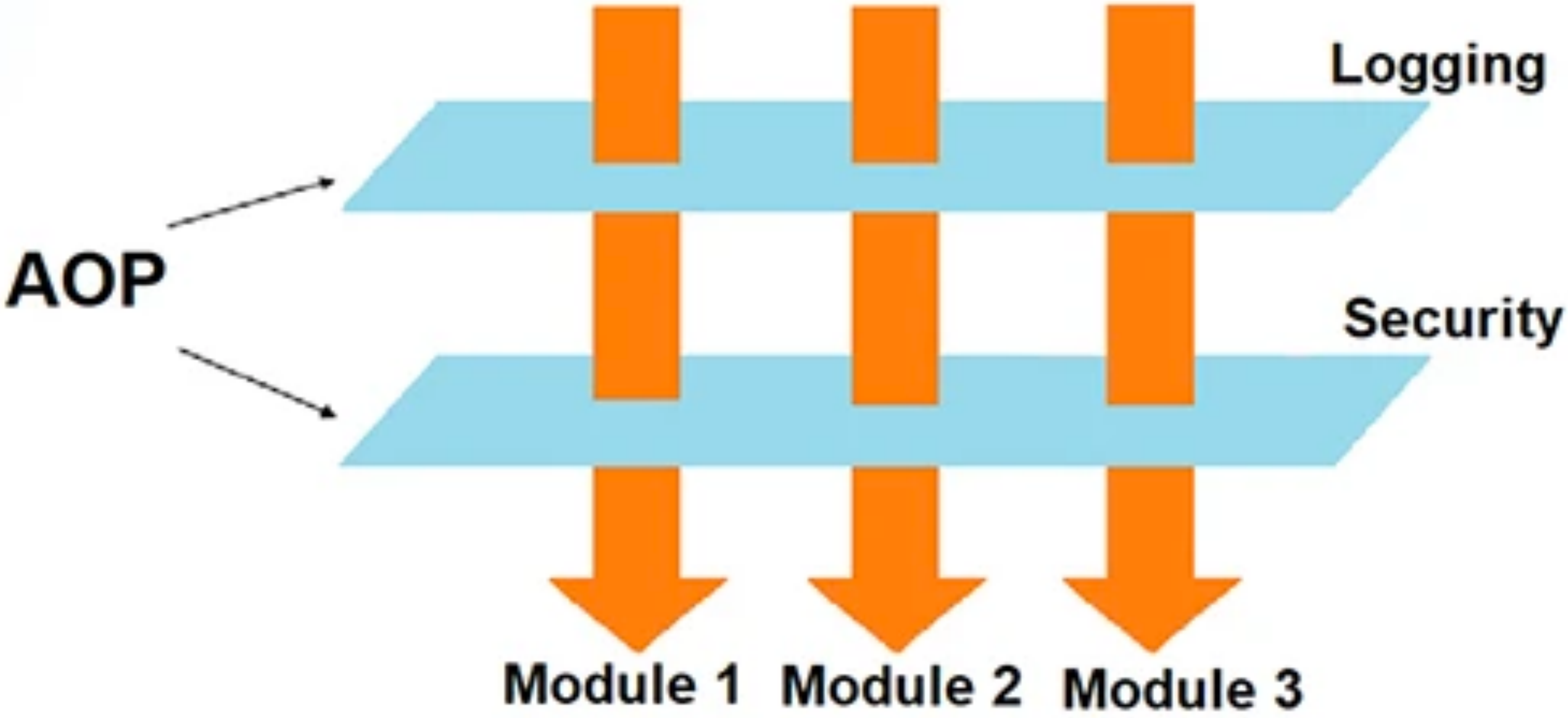
Part V

Addressing Cross cutting concerns -
# logging
# exception handling
Addressing Operational concerns -
# Health monitoring Patterns
# Alerts

3. Cross cutting concerns



The **crosscutting concern** is a concern which is needed in almost every module of an application.


Logging
AOP
Security
Module 1
Module 2
Module 3

Evaluate Architecture

Architecture Trade of Analysis Method (ATAM)

Scenario generation using Utility Tree

Mapping Quality attributes with Architectural Decisions

Documenting Risks, Non Risks and sensitivity points

Lab : Evaluate Architecture using ATAM

Part 3
Arch Risks

	Risk	I	L	R	C	Notes
1	Performance of the View Todo is significantly impacted by large data volumes.	8	4	32	High	Testing to be undertaken to gauge impact.
2	Limitations of JPA API result in complex workarounds in repository implementation or workarounds in other architectural layers.	6	4	24	Medium	Can fall back to Hibernate which provides richer ORM functionality.
3	The Open Source tools and components selected lack the capabilities of equivalent commercial products limiting the features that can be implemented (e.g. GIS components).	6	3	18	High	Investigate alternative products.

- Risk – The description of the architectural risk.
- Impact to Project – The impact the risk will have on the project
    - (1 = Minor impact, 10 = Showstopper)
- Likelihood – The likelihood of the risk eventuating (1 = Low, 953 = Highly likely)
- Rating – The overall rating for the risk based on Impact * Likelihood (1 = Minor, 90 = Critical)
- Confidence – The confidence in resolving the issue should it eventuate (Low, Medium, High)

- ATAM
- ARID
- SAAM
-

Evaluate Architecture (ATAM)

# identify all Architectural approaches (design)

A1 (CQRS)
# A2(Cube)
# A3 (Caching)
…

# identify all quality requirements (requirements)

# s1 (< 5 sec)
# s2 (99.99%)
s3 (…)
…

# analyse Scenario -> Approach

S1 -> A1, A2
S2 -> A6,A8, A9
S3-> ?
S4 -> A6, ?

=> Risk & trade off's

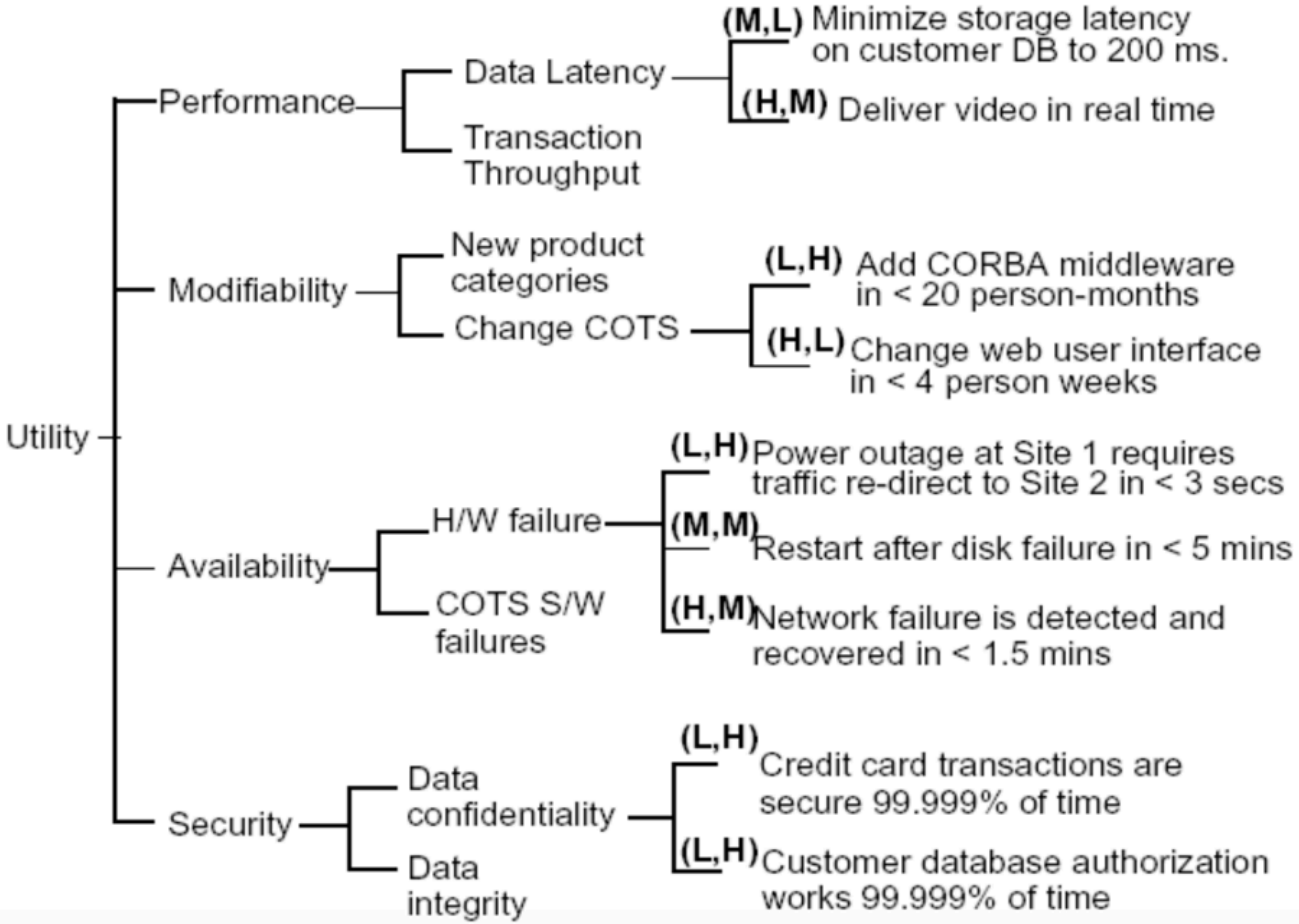
# brainstorm for scenarios

s8
s9
# s10
…

# analyse Scenario -> Approach

S8 -> A1, A2
S9 -> A6,A8, A9
S10-> ?

=> Risk & trade off's



- Utility
  - Performance
    - Data Latency
      - (M,L) Minimize storage latency on customer DB to 200 ms.
      - (H,M) Deliver video in real time
    - Transaction Throughput
  - Modifiability
    - New product categories
    - Change COTS
      - (L,H) Add CORBA middleware in < 20 person-months
      - (H,L) Change web user interface in < 4 person weeks
  - Availability
    - H/W failure
      - (L,H) Power outage at Site 1 requires traffic re-direct to Site 2 in < 3 secs
      - (M,M) Restart after disk failure in < 5 mins
      - (H,M) Network failure is detected and recovered in < 1.5 mins
    - COTS S/W failures
  - Security
    - Data confidentiality
      - (L,H) Credit card transactions are secure 99.999% of time
      - (L,H) Customer database authorization works 99.999% of time
    - Data integrity