



ARCHITECTING SOFTWARE SOLUTIONS

INTRODUCTION TO SOLUTION ARCHITECTURE

Instructor: Heng Boon Kui

Email: boonkui.heng@nus.edu.sg

Total slides: 40

- Understand what **architecture** is
- Differentiate between **enterprise** architecture and **solution** architecture
- Understand the **role** and **responsibilities** of an architect
- Understand the **quality attributes** of a software system

- Introduction to Architecture
 - Definition of Architecture
 - Importance of the architectural review
 - Enterprise Architecture and Solution Architecture
 - Architect's role and responsibilities
- Introduction to software quality attributes
 - The ISO 25010:2011 model



Definition of Architecture

The fundamental organization of a system embodied in its **components**, their **relationships to each other**, and to the **environment**, and the **principles** guiding its design and evolution.



Definition adapted from ANSI/IEEE Standard 1471-2000 (*IEEE Recommended Practice for Architectural Description for Software-Intensive Systems*) and ISO/IEC/IEEE 42010:2011 (*Systems and software engineering — Architecture description*)



Influences on Architecture.

Stakeholders
Organization
Technical Environment
Architect's Experience



Solution Architect

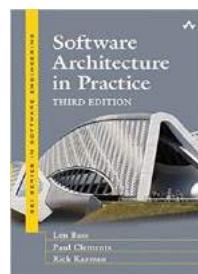


Architecture



System

Adapted from
the book
**'Software
Architecture in
Practice'** by Len
Bass, Paul
Clements and
Rick Kazman





Benefits of the Architecture Review.

Benefits derived from the architecture review:

- a) Identifying potential risks in the proposed architecture (88%)
- b) Assessing quality attributes (for example, scalability, performance) (77%)
- c) Identifying opportunities for reuse of artifacts and components (72%)
- d) Promoting good architecture design and evaluation practices (64%)
- e) Reducing project cost caused by undetected design problems (63%)
- f) Capturing the rationale for important design decisions (59%)
- g) Uncovering problems and conflicts in requirements (59%)
- h) Conforming to organization's quality assurance process (55%)
- i) Assisting stakeholders in negotiating conflicting requirements (43%)
- j) Partitioning architectural design responsibilities (40%)
- k) Identifying skills required to implement the proposed architecture (40%)
- l) Improving architecture documentation quality (40%)
- m) Facilitating clear articulation of nonfunctional requirements (31%)
- n) Opening new communication channels among stakeholders (31%)

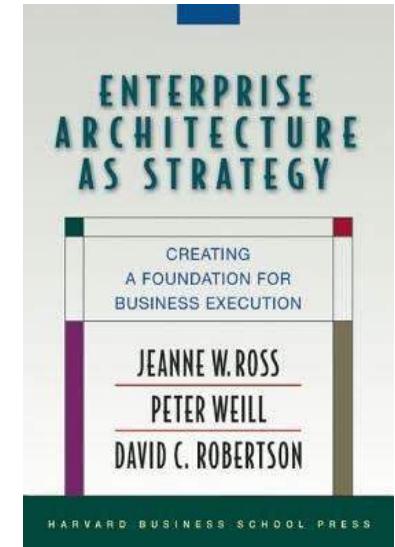
Survey of benefits from systems architecture reviews for 86 responding organizations, from 'Software Architecture Review: The State of Practice' by Muhammad Ali Babar, Lero - the Irish Software Engineering Research Centre and Ian Gorton, Pacific Northwest National Laboratory, IEEE Computer July 2009 © IEEE 2009.



Definition of Enterprise Architecture

Enterprise Architecture is:

- The organizing logic for **business processes** and **IT infrastructure** reflecting the **integration** and **standardization** requirements of the firm's operating model.
- A **conceptual blueprint** that defines the structure and operation of an organization. The intent of an enterprise architecture is to determine how an **organization** can most effectively achieve its current and future objectives.
- The source of **strategic direction** and **guidance** to solution architecture.



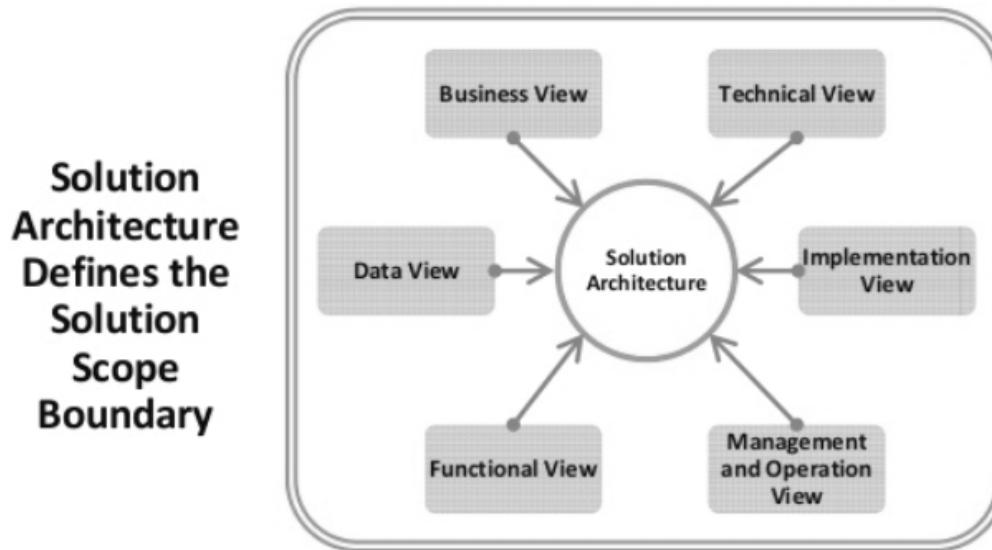
Adapted from MIT's Center for Information Systems and the book above

<https://cisl.mit.edu/research-overview/classic-topics/enterprise-architecture/>



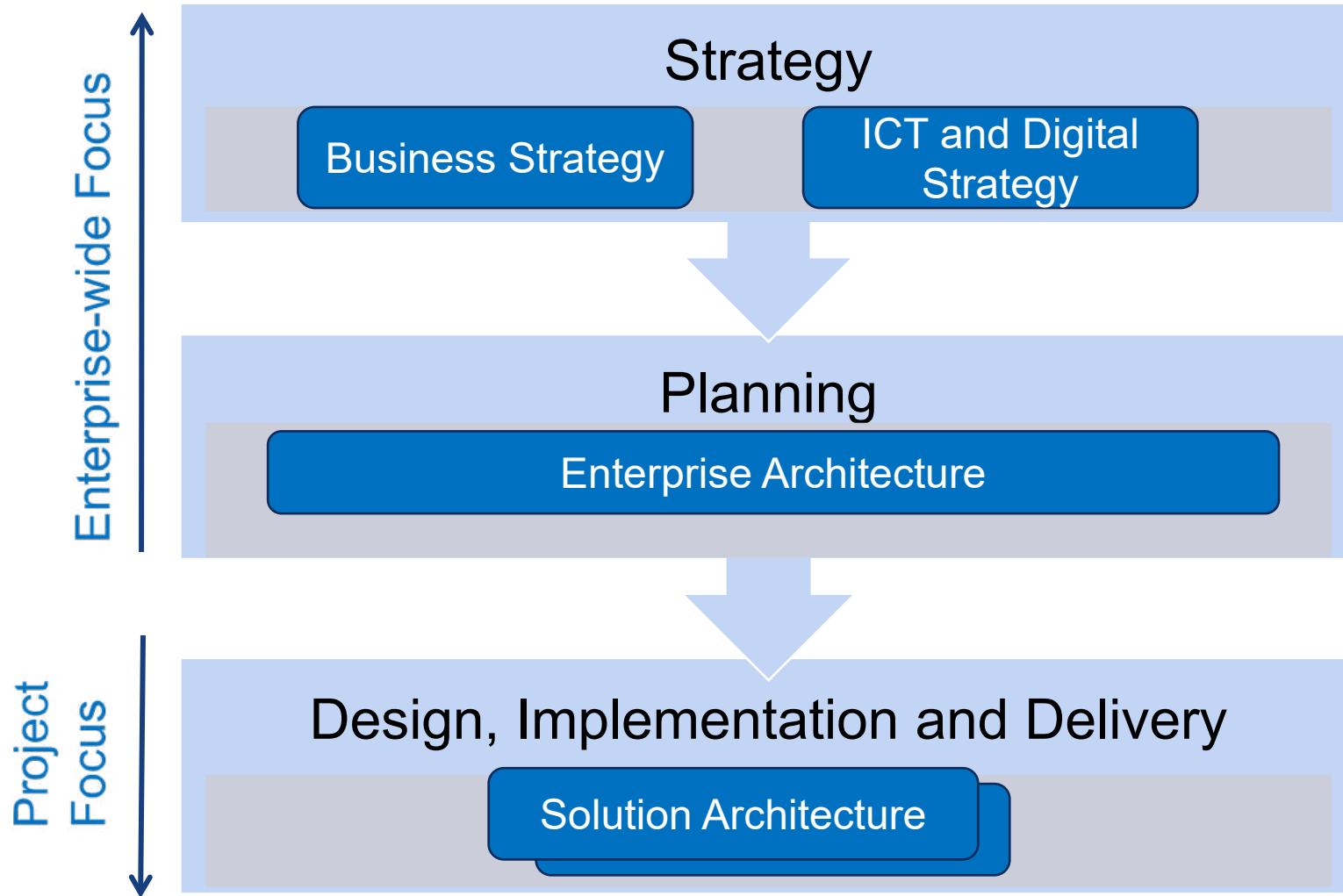
Solution Architecture

- Implementing ICT projects requires **analysis, design and construction** – the product is ‘the solution’.
- A solution’s architecture is influenced by its business requirements, project constraints and available technology options.
- ‘Solution Architecture’ describes the **end-to-end solution**, bringing together multiple views.





Relationship between Enterprise Architecture and Solution Architecture



Enterprise Architecture is the source of strategic direction and guidance to the Solution Architecture.



Relationship between Enterprise Architecture and Solution Architecture

Enterprise Architecture

Solution Architecture

Software
Architecture

Data
Architecture

Integration
Architecture

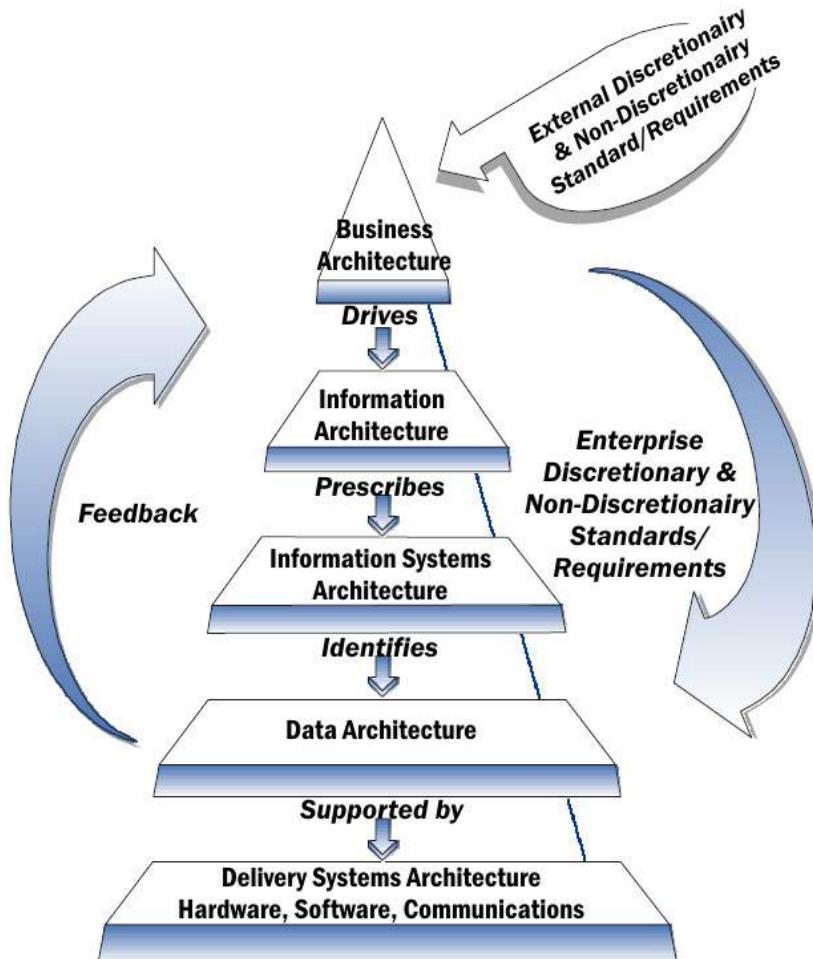
Technology
Architecture

... and
more

Enterprise Architecture is the source of strategic direction and guidance to the Solution Architecture.



Enterprise Architecture and Solution Architecture: Focus and Deliverables



The Open Group Architecture Framework (TOGAF) provides guidance on architecture

<http://www.opengroup.org/subjectareas/enterprise/togaf>

http://www.bestpractices.ca.gov/sysacq/documents/solution_architecture_framework_toolkit.pdf

https://en.wikipedia.org/wiki/Enterprise_architecture_framework

- The Enterprise Architect has the responsibility for architectural design and documentation at a landscape and technical reference model level. The Enterprise Architect often leads a group of the Segment Architects and/or Solution Architects related to a given program. The focus of the Enterprise Architect is on enterprise-level business functions required.
- The Solution Architect has the responsibility for architectural design and documentation at a system or subsystem level, such as management or security. A Solution Architect may shield the Enterprise/Segment Architect from the unnecessary details of the systems, products, and/or technologies. The focus of the Solution Architect is on system technology solutions.

Enterprise Architecture's deliverables:

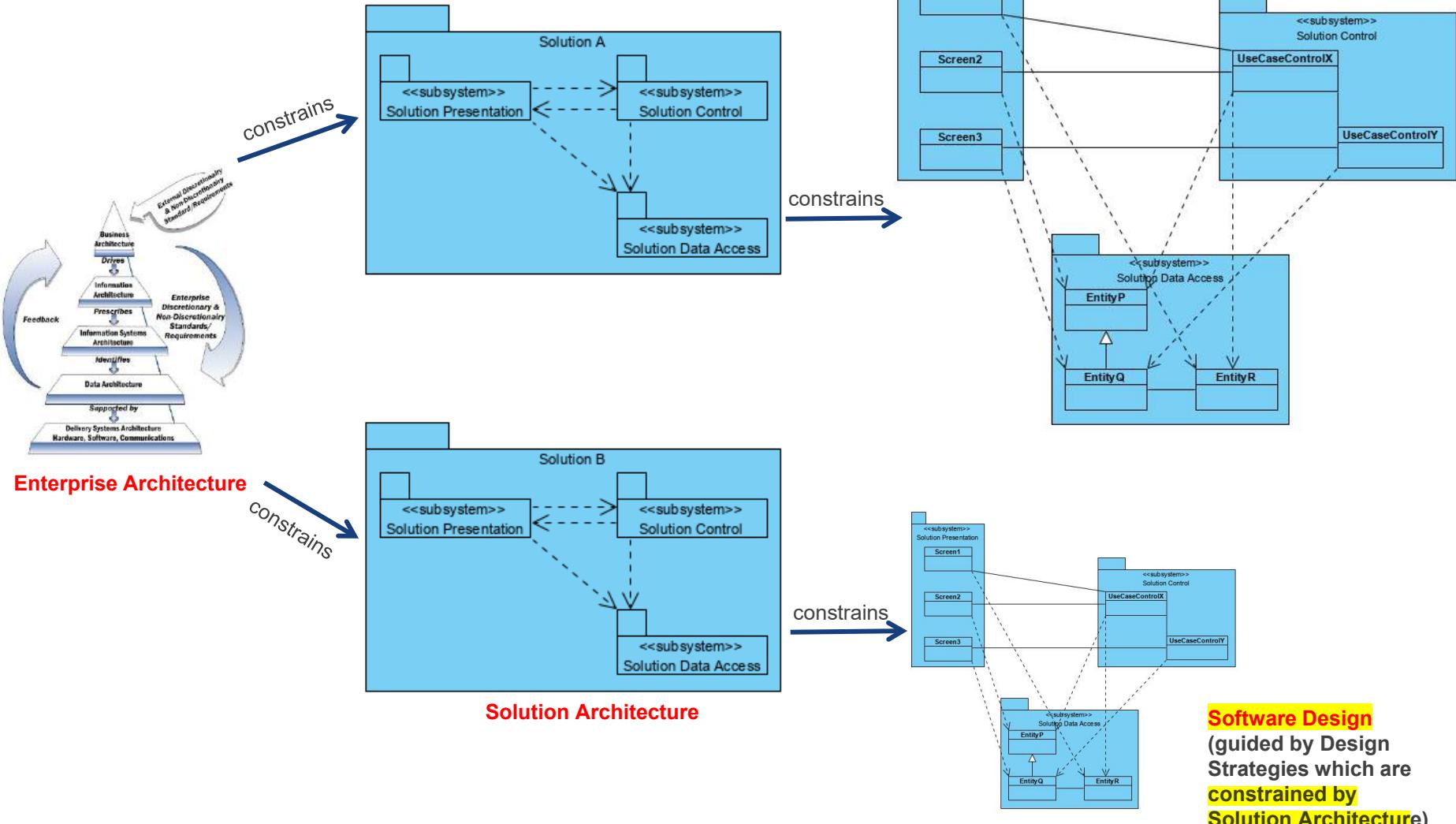
- Business Architecture model
- Data Architecture model
- Application Architecture model
- Technology Architecture model
- Gap analysis
- Impact assessment
- Transition Architecture

Solution Architecture's deliverables:

- Conceptual Solution Architecture Model
- Logical Data Model
- Logical Service Model
- Logical Technology Model
- Physical Data Model
- Physical Service Model
- Physical Technology Model



EA vs. Soln Arch vs. S/W Design





Relationship between Solution Architecture and Detailed Design

Solution Architecture (includes High-Level Design)	Detailed Design
Defines how components work together at a logical or high-level	Defines how components work together at a detailed level , sufficient for implementing the solution
Incorporates guidelines, standards, patterns, policies and constraints	Defines the methods and tools to be used for the implementation of the solution to the defined architecture
Addresses how the solution's components will work together to meet business requirements and the expected qualities	Addresses how solution-level components will be implemented and operated within the defined architecture



Common Characteristics of Architecture and Design.

Cohesion

Characteristic	Benefit to Architecture and Design
Abstraction	Helps us to focus on significant detail only
Encapsulation	Helps us to hide detail that is unnecessary
Cohesion (high is good)	Degree to which components belong together <u>High cohesion</u> supports good abstraction and encapsulation, reduces complexity, and enhances reusability and maintainability
Coupling (low is good)	Degree of interdependence between components <u>Low coupling</u> supports good abstraction and encapsulation, reduces complexity, and enhances reusability and maintainability



The Solution Architect

The Solution Architect:

- Is the person, team or organization responsible for designing an IT system's architecture.

[Definition from ANSI/IEEE Standard 1471-2000](#)

- Designs (or refines) a solution blueprint or structure to guide the **development** of IT solutions in **hardware, software, processes** or related components, to meet current and future business needs.

The solution architecture developed may lead to broad or specific changes to **IT services, operating models and processes**, and should provide a framework to guide the **development and modification** of solutions.



[Definition from SkillsFuture Singapore - Skills Framework for Infocomm Technology](#)

Responsibilities:

- Responsible for the solution's **viability**: implement and operate
- Provides solution-related **leadership**
- **Supports** the Project Manager in delivering the solution

Activities:

- Assess the **business requirements** for the solution
- Evaluate feasible **technical solutions** and select the optimum choice
- Align the solution's architecture with the **Enterprise Architecture**
- Define and document the **architecture** of the optimum solution
- Define the integration of the solution with the **rest of the enterprise**
- Incorporate the required **security, reliability and maintainability**
- Support the **detailed design and construction** of the solution
- Deliver the required **functional and non-functional** requirements

What makes a ‘Good’ Architect:

- The stakeholders and project team have **confidence** in the Architect
- The Architect is able to correctly **define and document** the architecture of an optimum solution that meets needs
- The project team is able to **design, implement and deliver** the architected solution
- **The solution performs as expected**
- **The Architect is dispensable**

What makes a ‘Not So Good’ Architect:

- Creating a perfect architecture for the **wrong system**
- Creating a perfect architecture which **cannot be implemented** within the constraints
- **Ivory tower** syndrome
- The **absent architect** syndrome

- Introduction to Architecture
 - Architecture definition
 - Importance of the architecture review
 - Enterprise Architecture and Solution Architecture
 - Architect role and responsibilities
- Introduction to **software quality attributes**
 - ISO 25010:2011 model



What is a Software Quality Attribute?

Software Quality Attribute

- Measures **how well** the software is designed
- Measures **how well** the software conforms to the design

Relationship among Quality Attributes

- **Direct Relationship**
If a software system is good at one attribute it should also be good at the other attribute
- **Neutral Relationship**
The two attributes are independent of each other
- **Inverse Relationship**
A software system that is good at one attribute will not be good at the other attribute



ISO 25010 Product Quality Model

- Basis of a product quality **evaluation** system
- Defines **quality attributes** of a software product
- Documented as a **standard** by ISO in ISO/IEC 25010 – Systems and software Quality Requirements and Evaluation (SQuaRE)



<http://iso25000.com/index.php/en/iso-25000-standards/iso-25010>

Refer to Appendix A for the ISO25010 Quality Model.



Discussion

1. Select two qualities from the ISO 25010 Product Quality model and discuss their relationship (either direct or inverse).
2. Give a real-world example to illustrate the relationship.

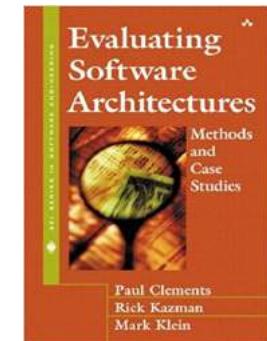
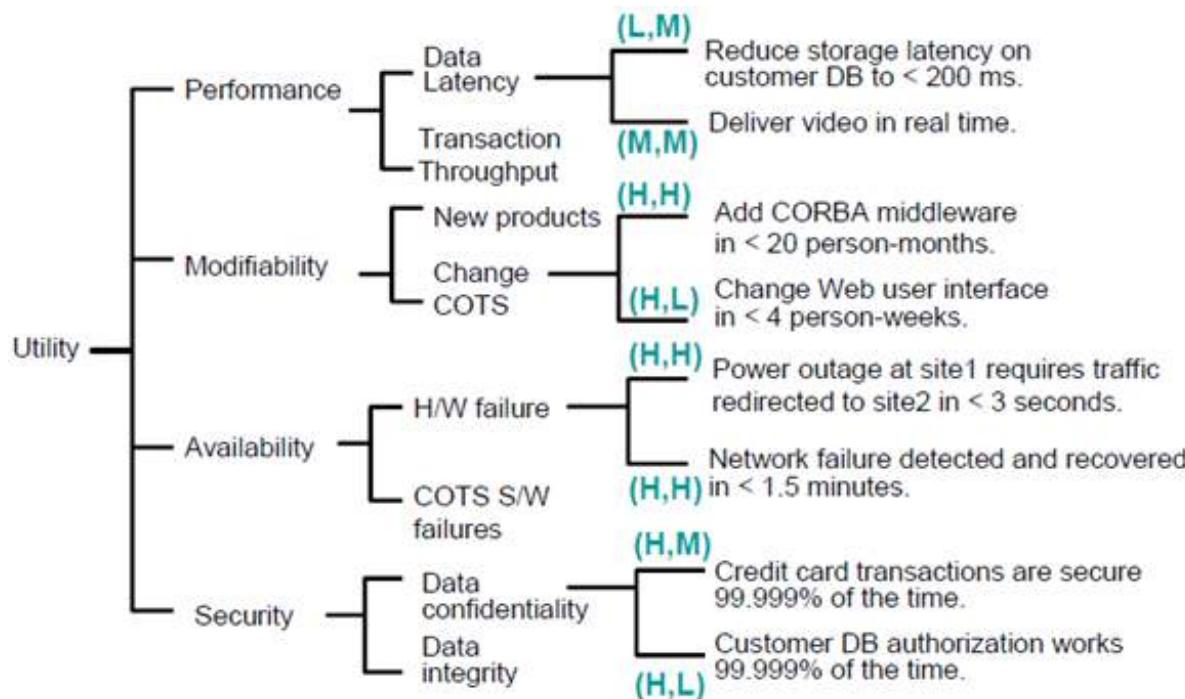
The selected quality attributes should be specific. For example, for user input via a graphical user interface versus command-line, discuss **Operability** (Usability) versus **Resource Utilization** (Efficiency).



Quality Attribute Utility Tree.

- A ‘utility tree’ represents the **overall usefulness** of the system
- The **scenarios** serve as the **leafs** of the utility tree
- The architecture is evaluated by considering how it makes the scenarios possible

To rank each leaf: (Importance of the node, Risk of achieving)



From '*Evaluating Software Architectures: Methods and Case Studies*' by Paul Clements, Rick Kazman and Mark Klein.



Summary.

- Architecture is the **organization** of a system's components, their **relationship** to each other and to the environment, governed by principles and standards
- Enterprise Architecture's focus is on **enterprise-wide** concerns and Solution Architecture's focus is on **projects**
- A Solution Architect is responsible for the solution's **viability** and technical **leadership** in its implementation



APPENDIX A

(ISO25010 Product Quality Model)



Degree to which a product or system provides functions that meet stated and implied needs when used under specified conditions

- **Functional Completeness**

Degree to which the set of functions covers all the specified tasks and user objectives

- **Functional Correctness**

Degree to which a product or system provides the correct results with the needed degree of precision

- **Functional Appropriateness**

Degree to which the functions facilitate the accomplishment of specified tasks and objectives



Represents the performance relative to the amount of resources used under stated conditions

- **Time Behaviour**

Degree to which the response and processing times and throughput rates of a product or system, when performing its functions, meet requirements

- **Resource Utilization**

Degree to which the amounts and types of resources used by a product or system, when performing its functions, meet requirements

- **Capacity**

Degree to which the maximum limits of a product or system parameter meet requirements



ISO 25010 - Compatibility

Degree to which a product, system or component can exchange information with other products, systems or components, and/or perform its required functions, while sharing the same hardware or software environment

- **Co-existence**

Degree to which a product can perform its required functions efficiently while sharing a common environment and resources with other products, without detrimental impact on any other product

- **Interoperability**

Degree to which two or more systems, products or components can exchange information and use the information that has been exchanged



ISO 25010 - Usability

Degree to which a product or system can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use

- **Appropriateness recognisability**

Degree to which users can recognize whether a product or system is appropriate for their needs

- **Learnability**

Degree to which a product or system can be used by specified users to achieve specified goals of learning to use the product or system with effectiveness, efficiency, freedom from risk and satisfaction in a specified context of use



ISO 25010 - Usability

- **Operability**
Degree to which a product or system has attributes that make it easy to operate and control
- **User error protection**
Degree to which a system protects users against making errors
- **User Interface Aesthetics**
Degree to which a user interface enables pleasing and satisfying interaction for the user
- **Accessibility**
Degree to which a product or system can be used by people with the widest range of characteristics and capabilities to achieve a specified goal in a specified context of use



ISO 25010 - Reliability

Degree to which a system, product or component performs specified functions under specified conditions for a specified period of time

- **Maturity**

Degree to which a system, product or component meets needs for reliability under normal operation

- **Availability**

Degree to which a system, product or component is operational and accessible when required for use

- **Fault Tolerance** ✓

Degree to which a system, product or component operates as intended despite the presence of hardware or software faults



ISO 25010 - Reliability

- **Recoverability**

Degree to which, in the event of an interruption or a failure, a product or system can recover the data directly affected and re-establish the desired state of the system



ISO 25010 - Security

Degree to which a product or system protects information and data so that persons or other products or systems have the degree of data access appropriate to their types and levels of authorization

- **Confidentiality**

Degree to which a product or system ensures that data are accessible only to those authorized to have access

- **Integrity**

Degree to which a system, product or component prevents unauthorized access to, or modification of, computer programs or data



ISO 25010 - Security

- **Non-Repudiation**

Degree to which actions or events can be proven to have taken place, so that the events or actions cannot be repudiated later

- **Accountability**

Degree to which the actions of an entity can be traced uniquely to the entity

- **Authenticity**

Degree to which the identity of a subject or resource can be proved to be the one claimed

Degree of effectiveness and efficiency with which a product or system can be maintained

- **Modularity**

Degree to which a system is composed of discrete components such that a change to one component has minimal impact on other components

- **Reusability**

Degree to which an asset can be used in more than one system, or in building other assets

- **Analyzability**

Degree of effectiveness and efficiency with which it is possible to assess the impact on a product or system of an intended change to one or more of its parts, or to diagnose a product for deficiencies or causes of failures, or to identify parts to be modified

- **Modifiability**

Degree to which a product or system can be effectively and efficiently modified without introducing defects or degrading existing product quality

- **Testability**

Degree of effectiveness and efficiency with which test criteria can be established for a system, product or component and tests can be performed to determine whether those criteria have been met



ISO 25010 - Portability

Degree of effectiveness and efficiency with which a system, product or component can be transferred from one environment to another

- **Adaptability**

Degree to which a product or system can effectively and efficiently be adapted for different or evolving hardware, software or other operational or usage environments

- **Installability**

Degree of effectiveness and efficiency with which a product or system can be successfully installed and/or uninstalled in a specified environment.



ISO 25010 - Portability

- Replaceability

Degree to which a product can replace another specified software product for the same purpose in the same environment



APPENDIX B

(Other Quality Models)



Business Qualities

- Time to Market ✓
Competitive Pressure and Short window of opportunity
- Cost and Benefit
Development Budget that cannot be exceeded
- Projected Lifetime ✓
System lifetime affecting the modifiability, scalability and portability
- Targeted Market
Mass market versus specialized market



Quality in Use Model

- Effectiveness
Accuracy and completeness with which users achieve specified goals
- Efficiency
Resources expended in relation to the accuracy and completeness with which users achieve goals
- Satisfaction
Degree to which user needs are satisfied when a product or system is used in a specified context of use
- Freedom from Risk
Degree to which a product or system mitigates the potential risk to economic status, human life, health, or the environment
- Context Coverage
Degree to which a product or system can be used with effectiveness, efficiency, freedom from risk and satisfaction in both specified contexts of use and in contexts beyond those initially explicitly identified



ARCHITECTING SOFTWARE SOLUTIONS

ARCHITECT THE SOLUTION ARCHITECTURE (ASSET)

Instructor: Heng Boon Kui

Email: boonkui.heng@nus.edu.sg

Total slides: 101



Objectives

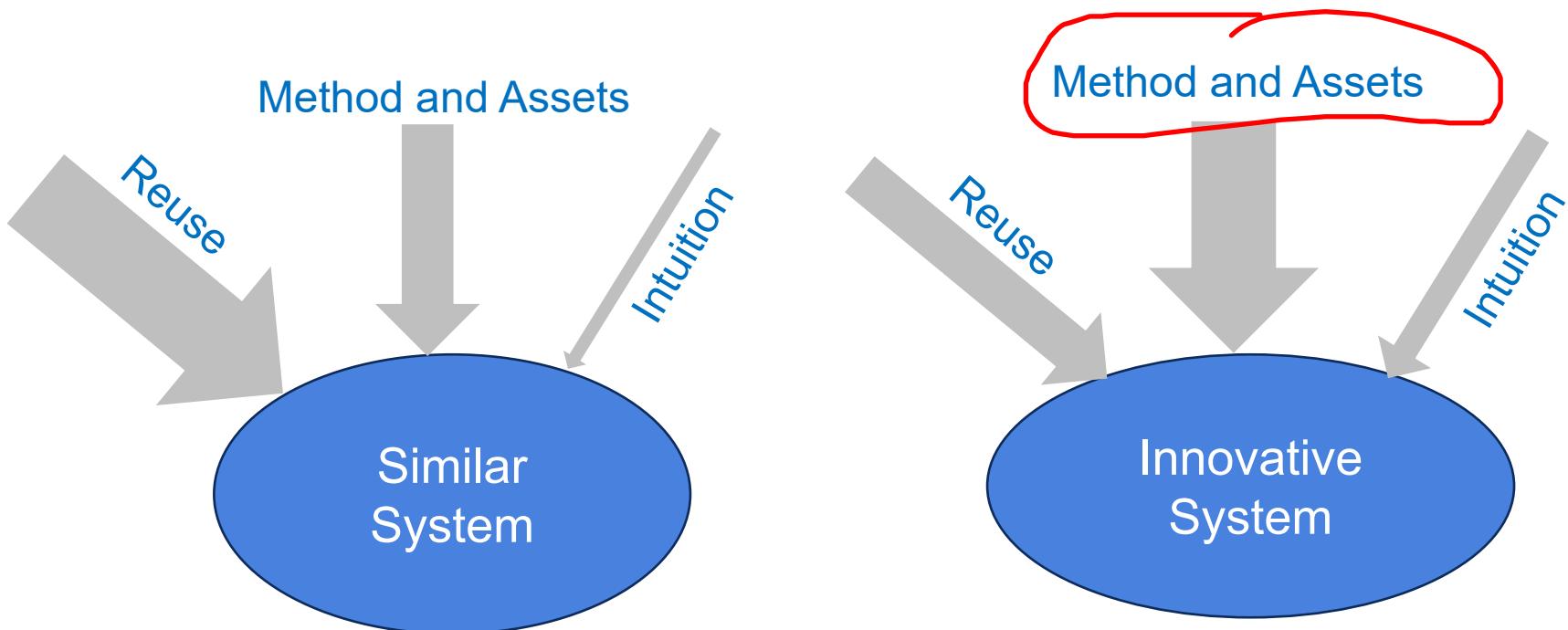
- Understand what **reusable** architectural assets are
- Understand the architecture styles, components and options to **integrate** components
- Understand how to **benefit** from the assets in defining the architecture

- Metamodel of Architectural Assets
- Architecture Styles
- Reference Architectures
- Architectural Description



Deriving the architecture

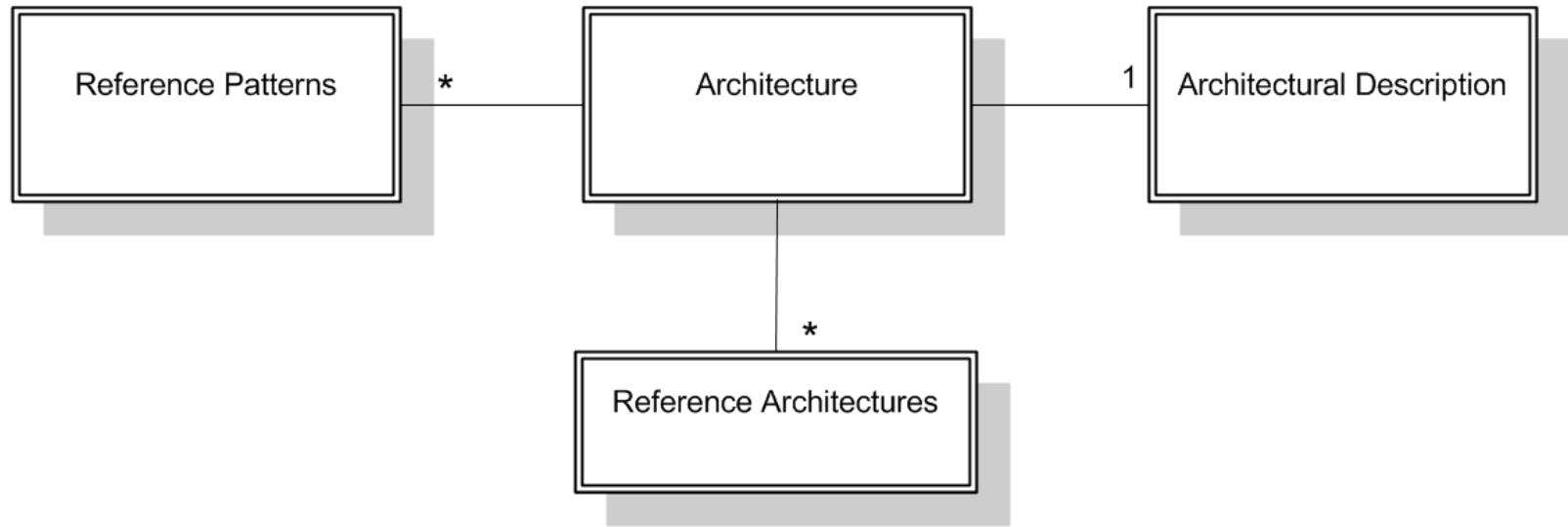
Deriving the architecture can be **different** for a system that is similar to one **implemented previously**, compared to a **new innovative** system



The size of the arrows indicates relative importance.



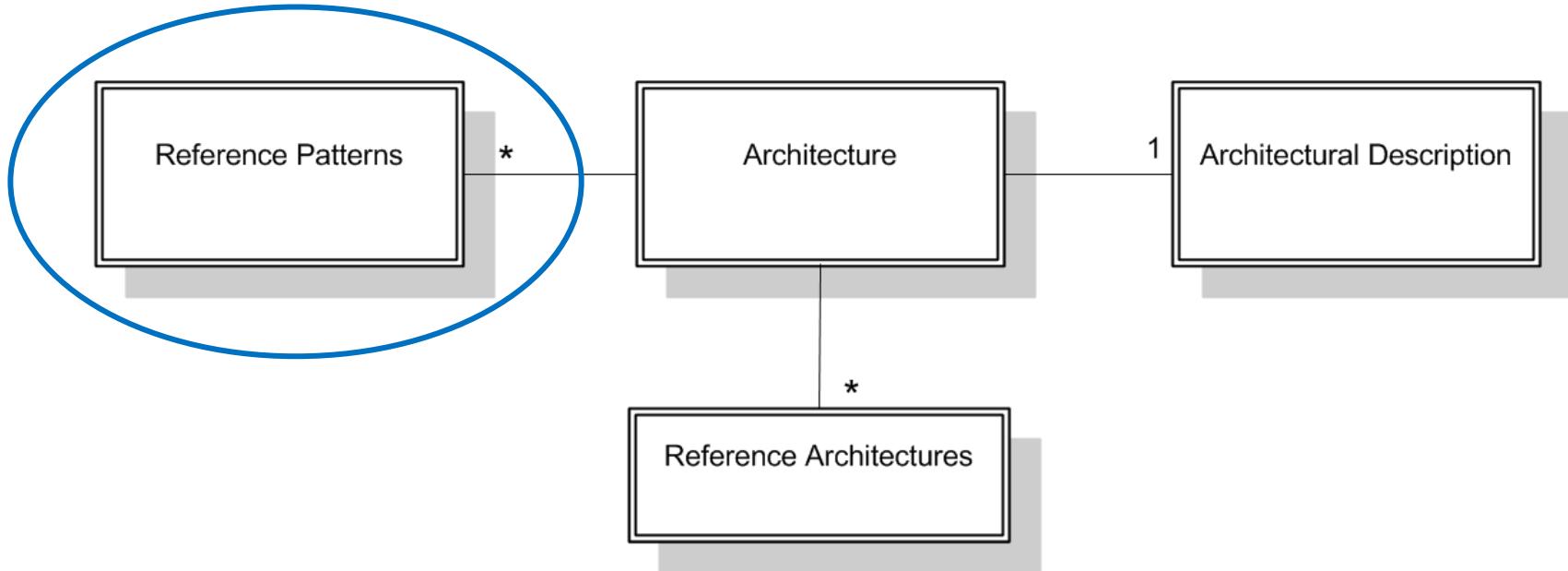
Metamodel of Architectural Assets



Reference patterns and reference architectures capture proven successes of the past in architecting a system.

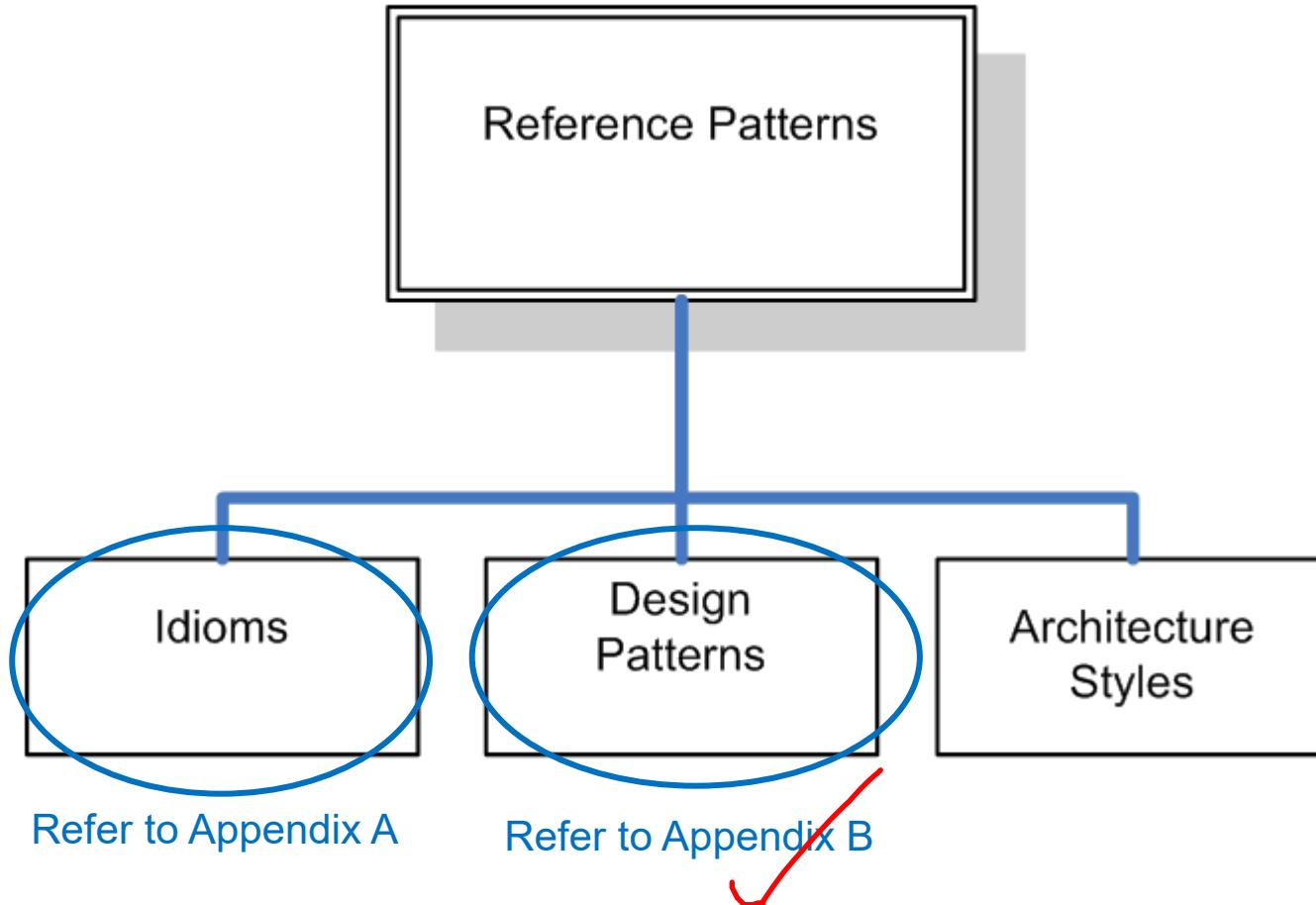


Metamodel of Architectural Assets





Reference Patterns.



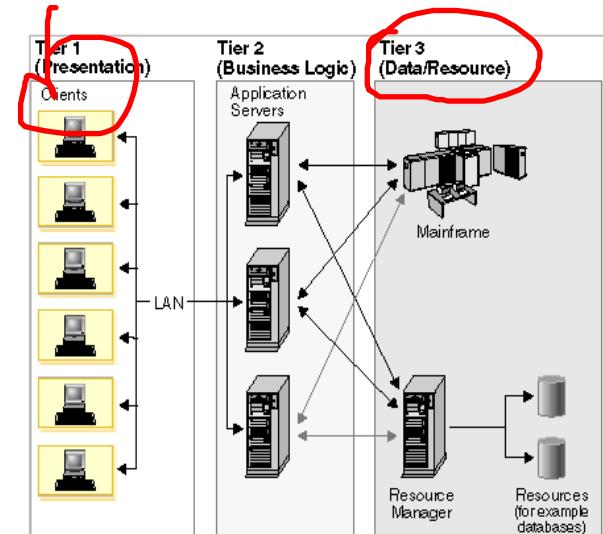
- Metamodel of Architectural Assets
- **Architecture Styles**
- Reference Architectures
- Architectural Description

Architectural Styles (Structural)



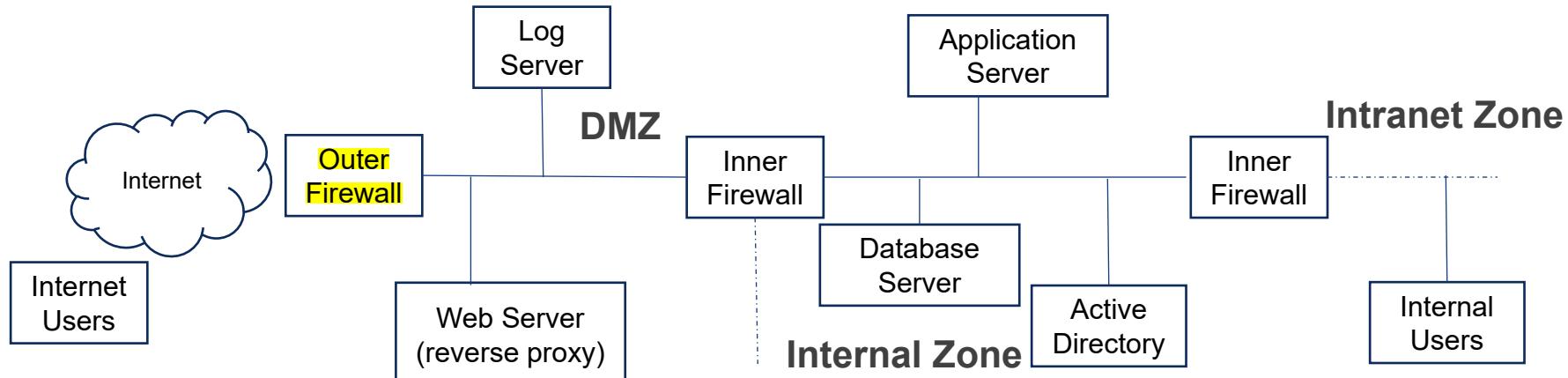
Architecture Styles (Structural)

- Client Server
 - Has a **client component and server component**. **Server** provides one or more **services** via well defined **interfaces** and **client** uses the services, e.g., web browser, applet and mobile app with server-side services
 - Thick or thin clients? Application layer design? State management? Authentication and authorization design?
- Tiered Computing
 - Contains a number of **computational tiers** which together offer a service to the consumer. **One tier acts as a server for its caller and client to the next tier in the architecture**, e.g., 3-tier and n-tier models.
*3
Tier*
 - Segregate by **layers / features / organization structure**? State management? Authentication and authorization design?



Network Components

- **Demilitarized Zone (DMZ)** is the portion of a network that separates a purely **internal** network from an **external** network.
- **Reverse Proxy** is an intermediate server that sits between the **internet users** and an **internal server** typically used to provide **internet users** access to the **internal server** that is behind a **firewall**. Hides identity of the servers.
- **Forward Proxy** is typically used to provide internet access to **internal clients** that are otherwise restricted by a **firewall**. Hides identity of the clients.





Server Components.

- **Service execution**
E.g., Application Servers
- **B2C integration**
E.g., Portal Servers
- **Service and B2B integration**
E.g., API Gateway, Enterprise Service Bus (ESB) suites, Business Process Management (BPM) suites and B2B gateways
- **Service management and governance**
E.g., Mobile Device Management (MDM), Application Management (MAM), Service Registry
- **Data Persistence**
E.g., SQL, NoSQL



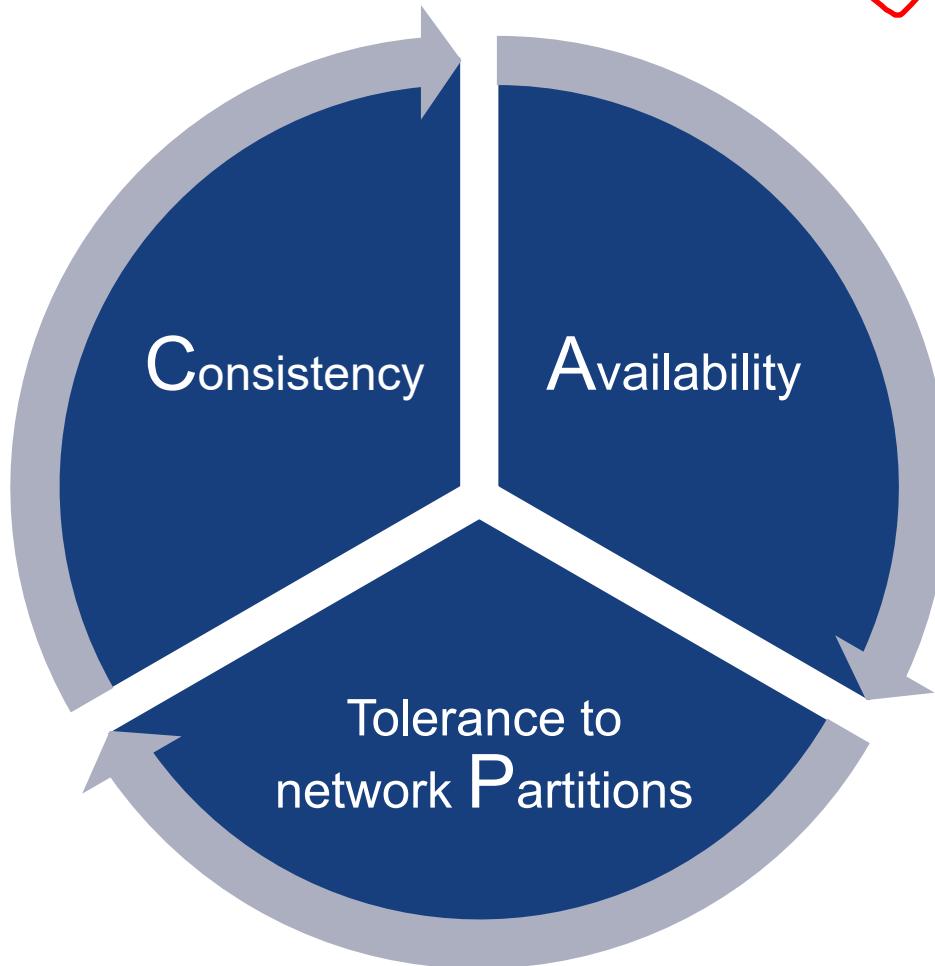
SQL vs. NoSQL Databases



Key differences	SQL	NoSQL
Types	One type with minor variations	Many types, e.g., key-value stores, document databases, wide-column stores and graph databases
Examples	MySQL, Postgres, Oracle Database, etc.	MongoDB, Cassandra, HBase, Neo4j, Amazon DynamoDB, etc.
Data Storage Model	Rows and Columns in tables	Varies based on database type
Schemas	Structure and data types are fixed in advance	Typically dynamic
Scaling	Typically vertically	Horizontally
Consistency	Can be configured for strong consistency	Depends on product. Can be strong or eventual consistency



Brewer's CAP Theorem



Theorem: We can have at most two of these properties for any shared data system.



ACID vs. BASE

- ACID
 - Atomicity – All of the operations in the transaction will complete, or none will.
 - Consistency - The database will be in a consistent state when the transaction begins and ends.
 - Isolation - The transaction will behave as if it is the only operation being performed upon the database.
 - Durability - Upon completion of the transaction, the operation will not be reversed.
- BASE
 - Basically Available - The database “appears” to work most of the time.
 - Soft-state - Stores don’t have to be write-consistent, nor do different replicas have to be mutually consistent all the time.
 - Eventual consistency - Stores exhibit consistency at a later point



Discussion.

Assume a large e-commerce web application with the following three features:

1. Select item(s) from product catalog
2. Place item(s) into shopping cart
3. Check out and make payment



How will you apply the CAP theorem, assuming that tolerance to network partitions (P) is always required?

Specifically, for each of the 3 items above, consider if:

- The item should be persisted?
- If so, whether C or A should be prioritized?



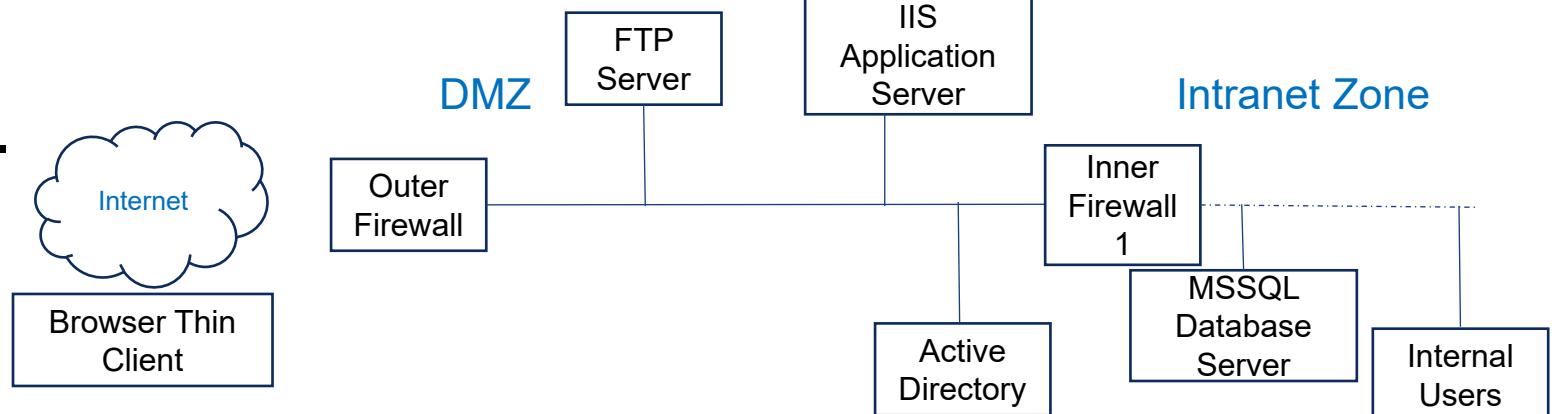
Architecture Styles (Structural).

- Peer to Peer
 - Each peer is capable of acting **as both client and server.**
 - Any peer is free to communicate directly with another peer **without a central server.**
 - E.g., Napster, BitTorrent, Clustering Agents, Massively Multiplayer Online Games, Bitcoin network, etc.
 - Strict consistency concern? Authentication and authorization design?

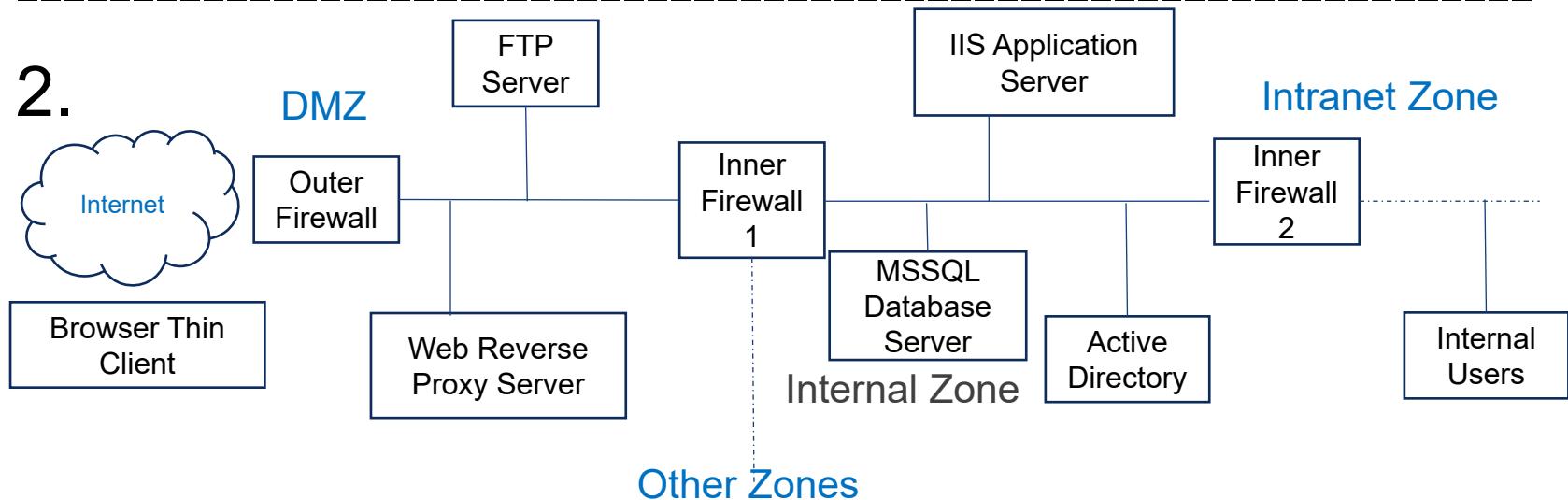
Discussion.

Which is a better option for structural deployment? Why?

1.



2.



Architecture Styles (Communication)



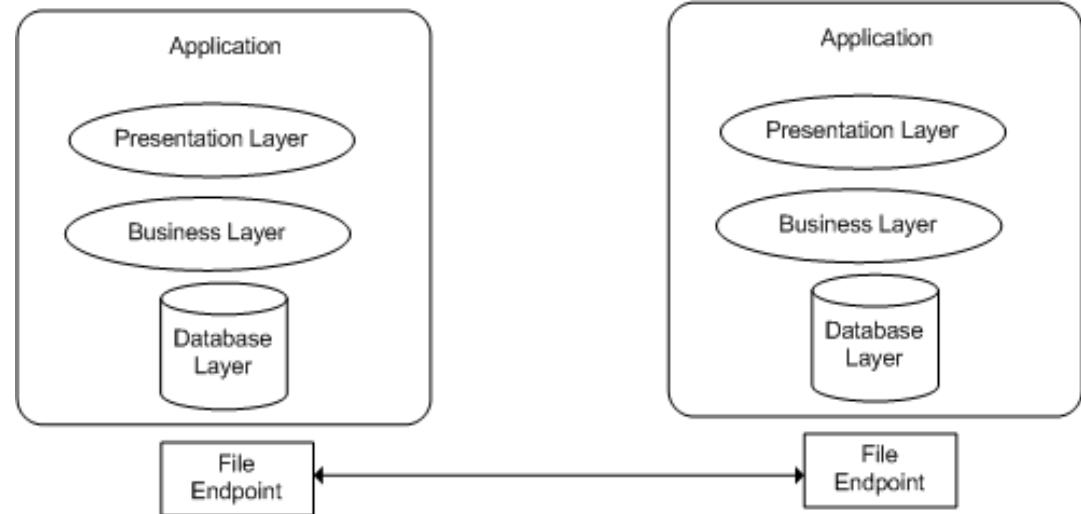
Architecture Styles (Communication)

- Data Flow
 - Data **enters** the system and then **flows through** the components one at a time until it is assigned to some **final destination**. Suitable for **batch data integration**.
 - E.g., Extract-Transform-Load
 - Error handling design? Acceptable processing time? One vs two way replication?



Data Flow - File Level

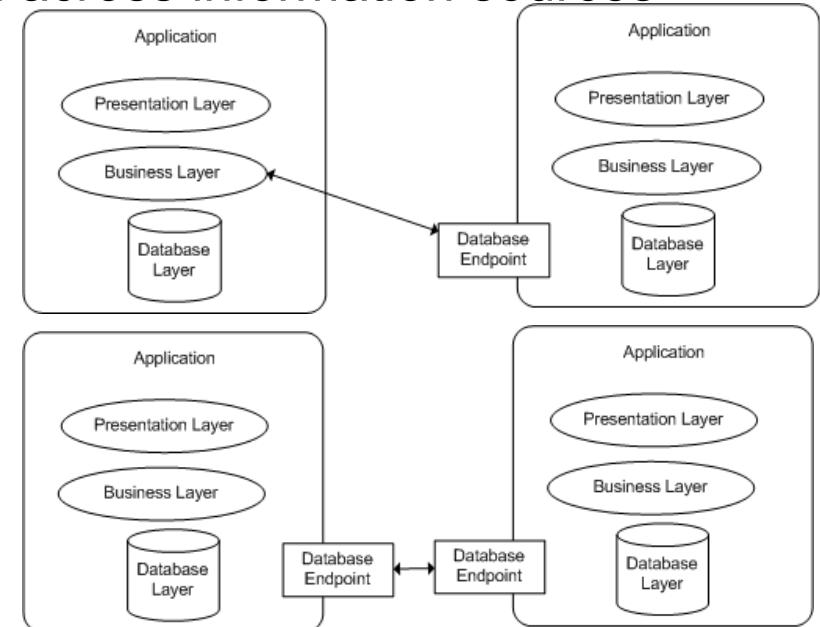
- Usage Scenarios
 - Transmitting **large batch files** for **scheduled jobs**
 - **Legacy** Integration with limited technology choices
- Integration Technology
 - Sending and receiving with **FTP**





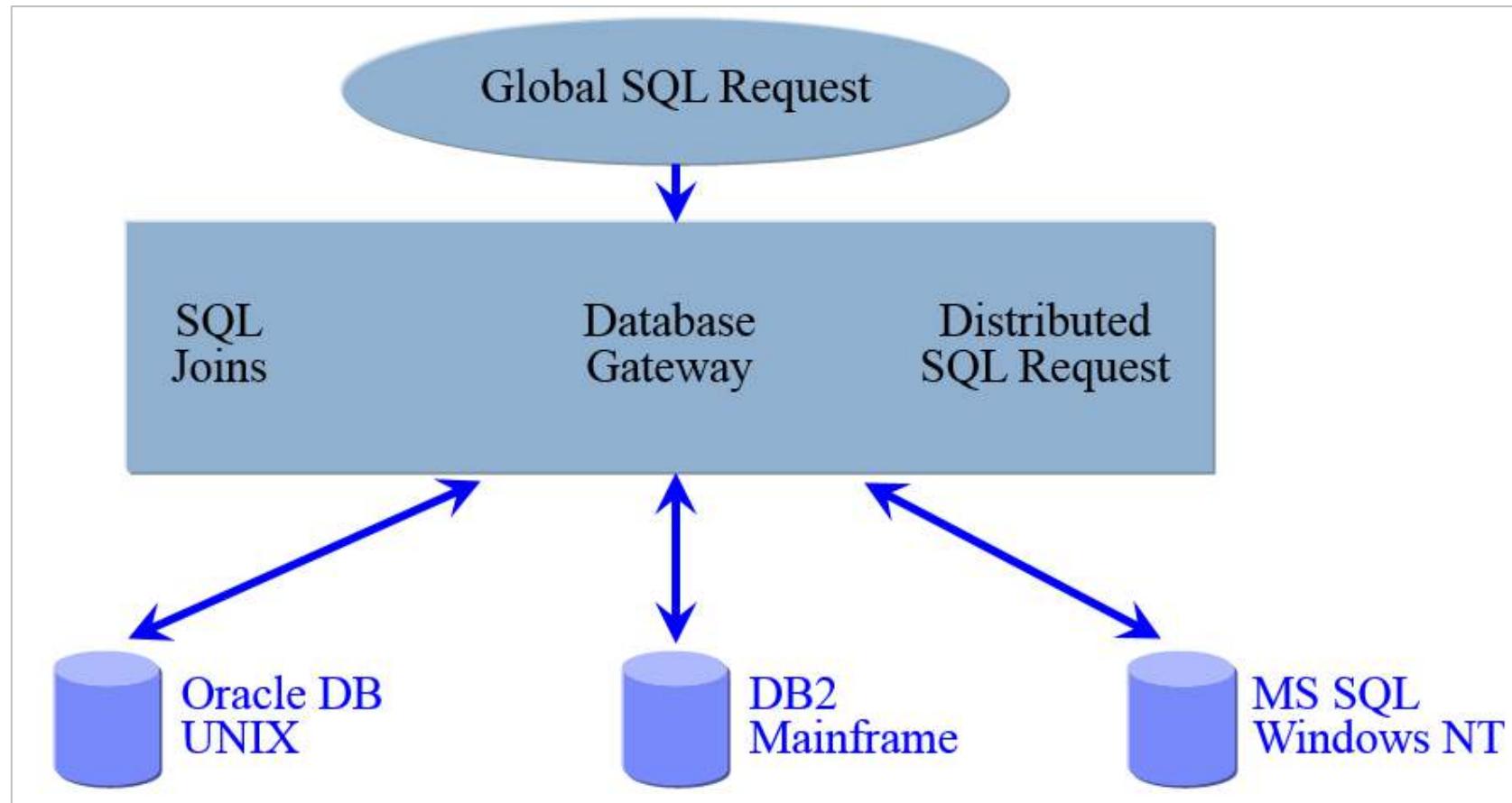
Data Flow - Database Level

- Usage Scenarios
 - Creating a **single view** of a customer or other business entity
 - **Enterprise data** inventory and management
 - Real time **reporting** and analysis and creating **dashboards**
 - Updating **common information** across information sources
- Integration Technology
 - **Database Views**
 - Database **Federation**
 - Database Replication



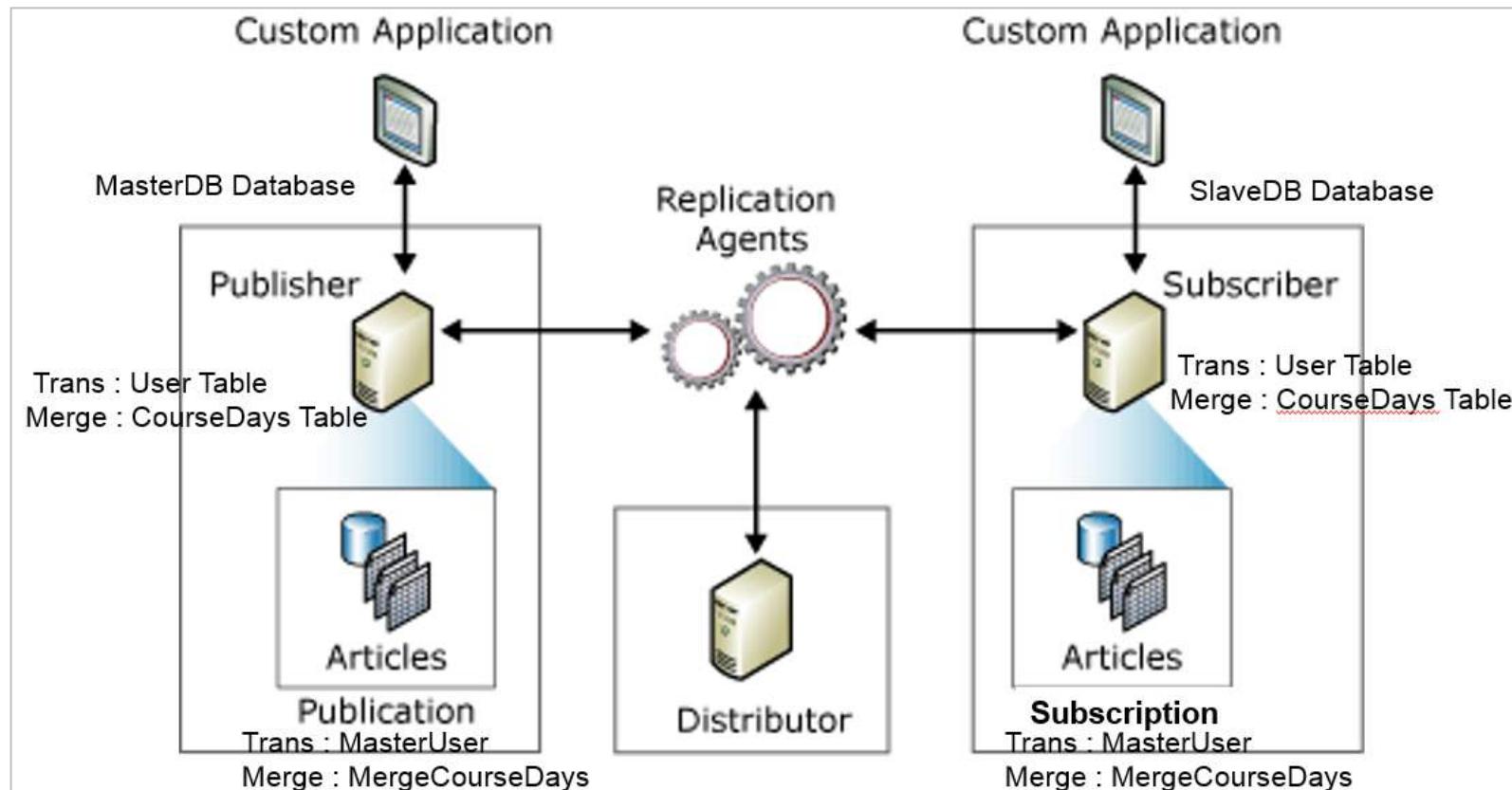


Database Federation





Database Replication.



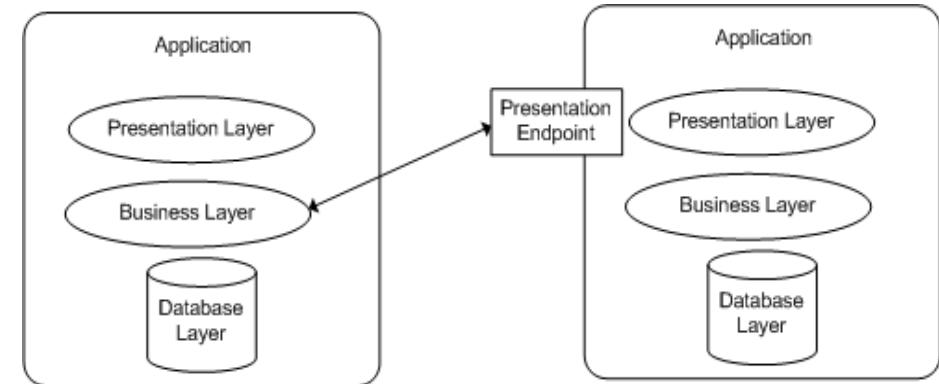
Architecture Styles (Communication)

- Call and Return
 - Each component **executes** only when it **gets the control** from another component and **returns** the control to that component when it **terminates execution**.
 - E.g., Open Systems Interconnection (OSI) layers, Operating System layers.
 - How to discover target? Error handling and timeout design?



Call and Return - Presentation Level

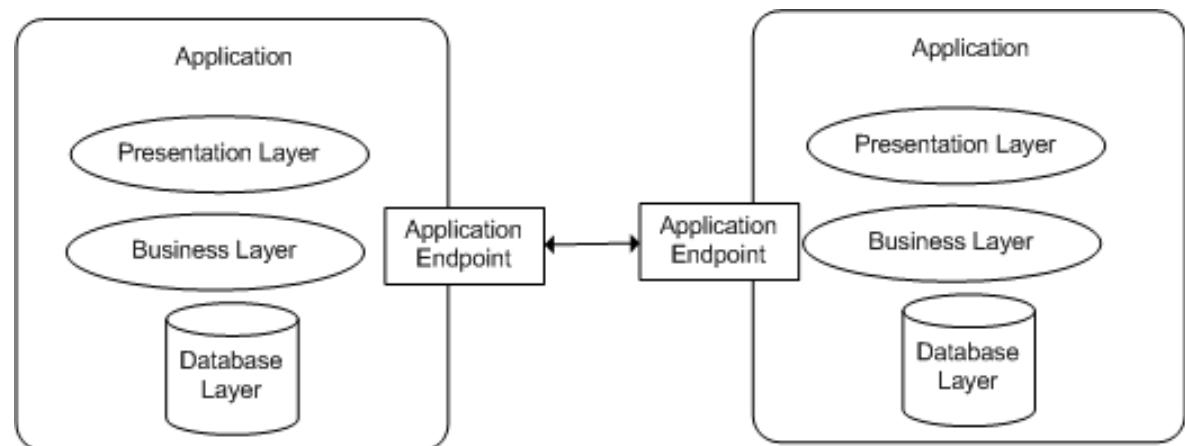
- Usage Scenarios
 - Extending **legacy** systems to the Web
 - Access to **public information** without compromising security
 - Absence of **source code**
 - Typically used for communication across **different technology domains**
- Integration Technology
 - Accessing Web Page via URL
 - Screen scraping





Call and Return - Application Level

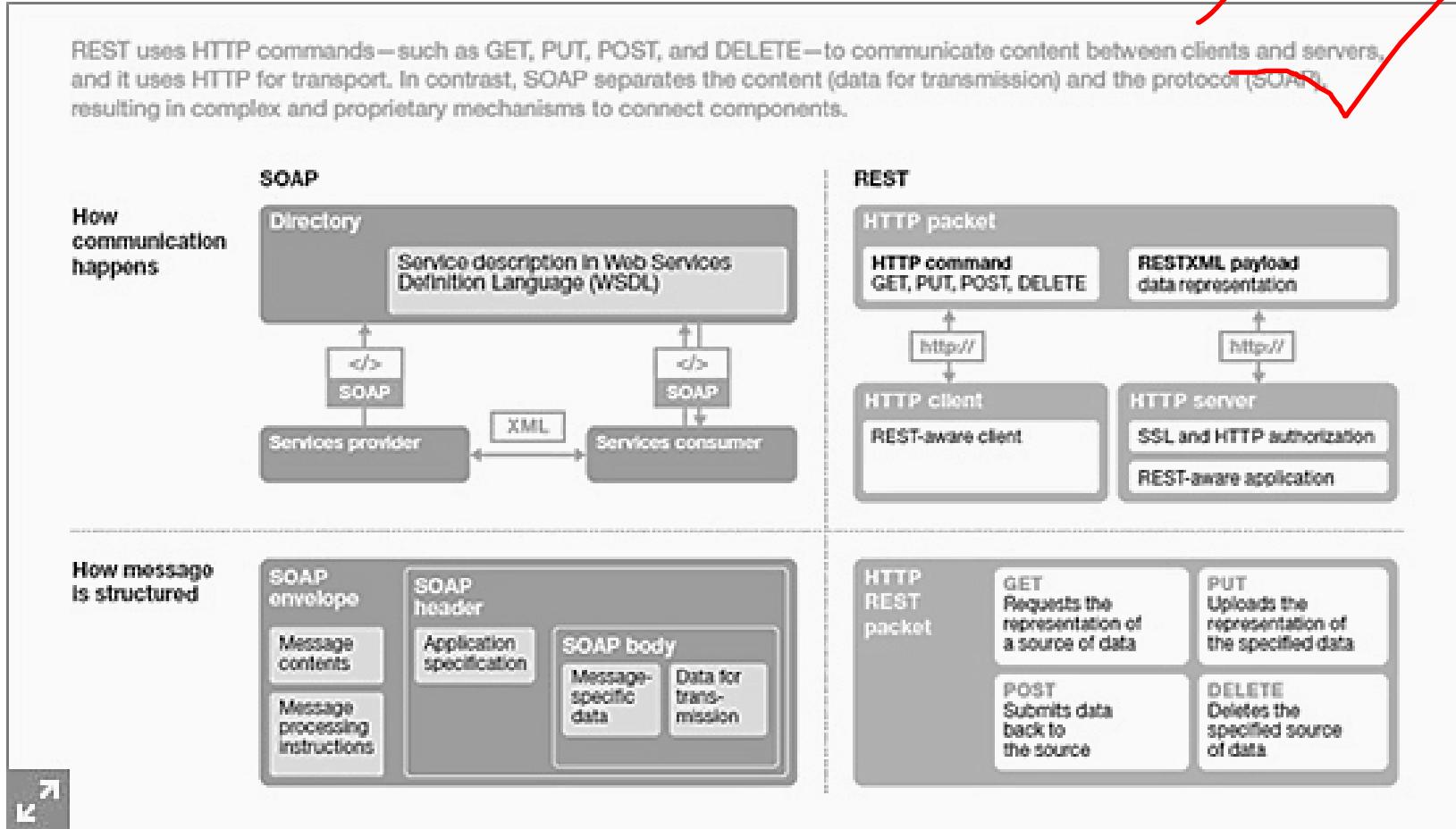
- Usage Scenario
 - **Business-to-business** transactions across the Web
 - **Integration** with existing information systems participating in a process
- Integration Technology
 - Remote Procedure Call (RPC)
 - **SOAP / REST**
Web Service



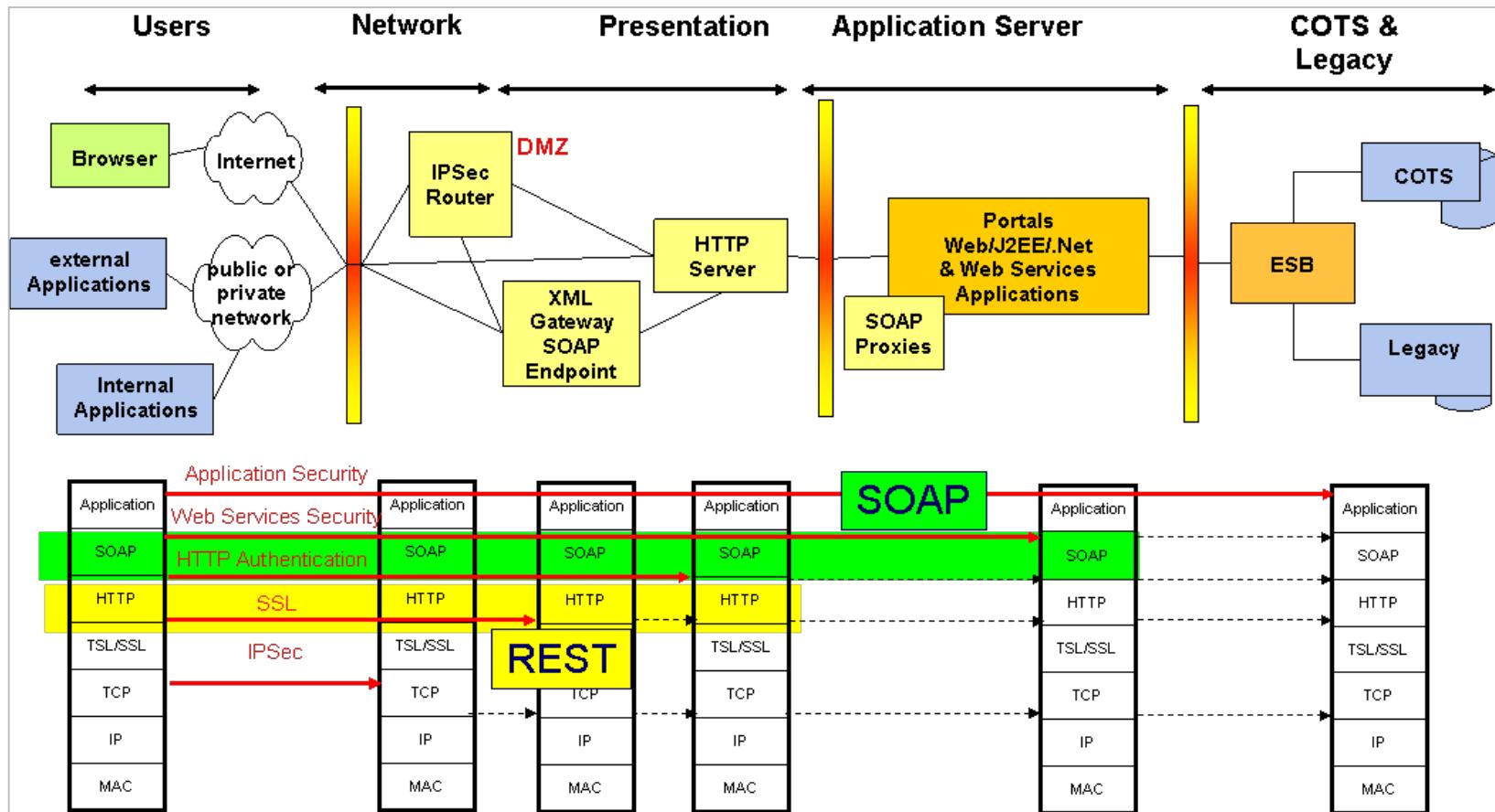
SOAP and REST Web Services

SOAP

REST uses HTTP commands—such as GET, PUT, POST, and DELETE—to communicate content between clients and servers, and it uses HTTP for transport. In contrast, SOAP separates the content (data for transmission) and the protocol (SOAP), resulting in complex and proprietary mechanisms to connect components.



SOAP and REST Web Services.





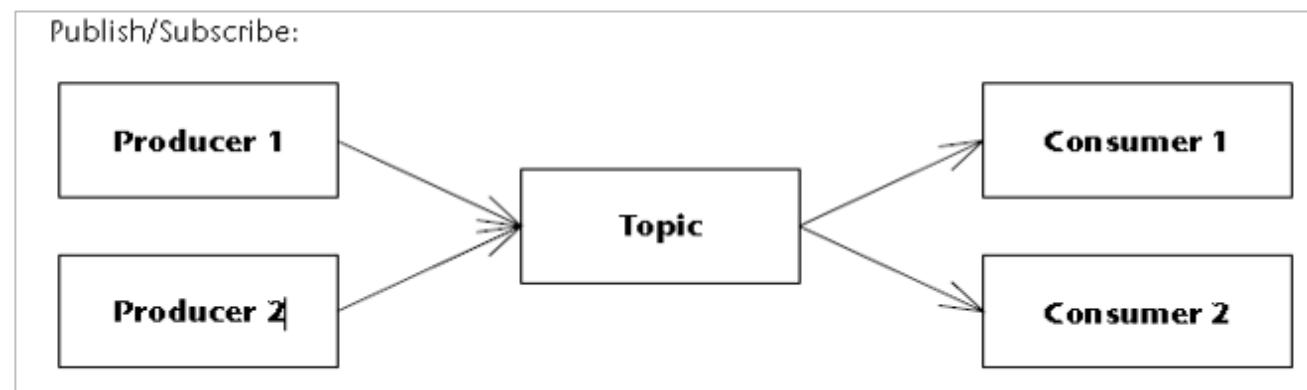
Architecture Styles (Communication)

- Event Based (**Point to Point**)
 - **Producer** (component) produces **message** for the **consumer** (component) to consume. Once the message is consumed, it will not be available for another consumer. The producer does not wait for the consumption of the message.
 - E.g., Event action handlers (Java Swing, iOS / Android **event handlers**), backend messaging integration.
 - Asynchronous complexity. Persistence or non persistence? Single point of failure? Pull or Push? Message sequencing concern?



Architecture Styles (Communication)

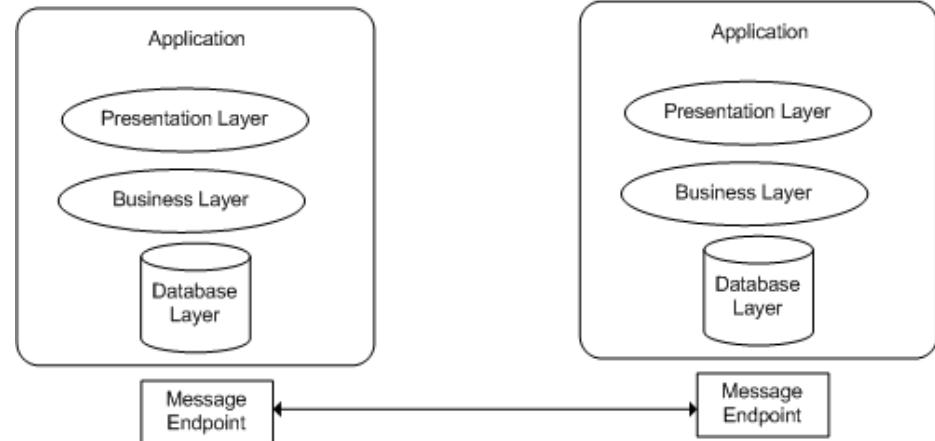
- Event Based (**Publish / Subscribe**)
 - **Subscriber** registers interest in **events** announced by the **publisher**. The publisher creates the information of interest and publishes it to **any number** of registered subscribers.
 - E.g., forum thread subscription, iOS / Android remote (push) notifications.
 - Asynchronous complexity. Durable or non-durable? Single point of failure?





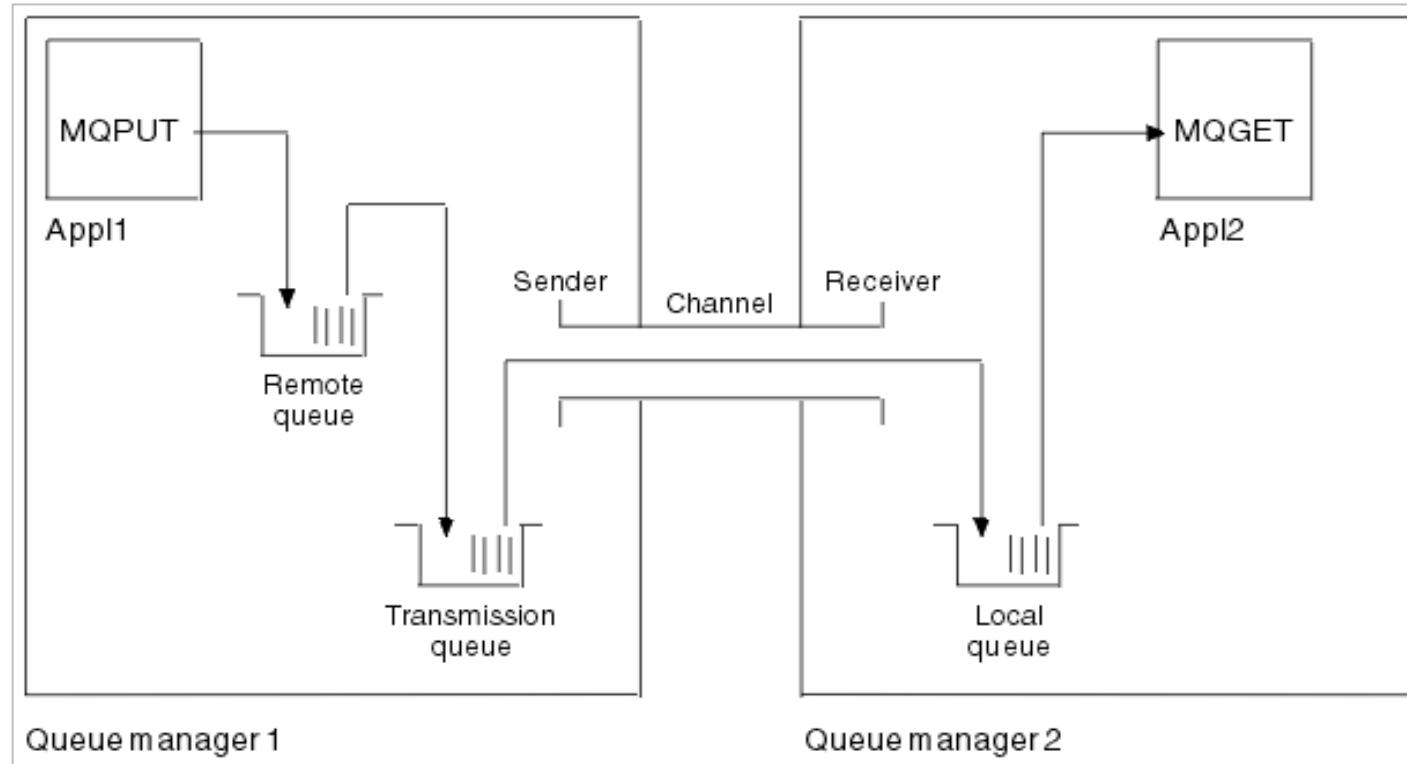
Event Based - Message Level

- Usage Scenarios
 - **Asynchronous** update of information
 - **Legacy** Integration with limited technology choices
 - Sending **notification** to customer after transaction
 - **Business to Business** transactions
- Integration Technology
 - Message Queue





Message Queues.





Discussion.

'An agency needs to enable the general public and tourists to access haze information with extensive visual graphics and illustrations. The haze data is detected by haze sensors with readings currently already stored in a haze information system.'

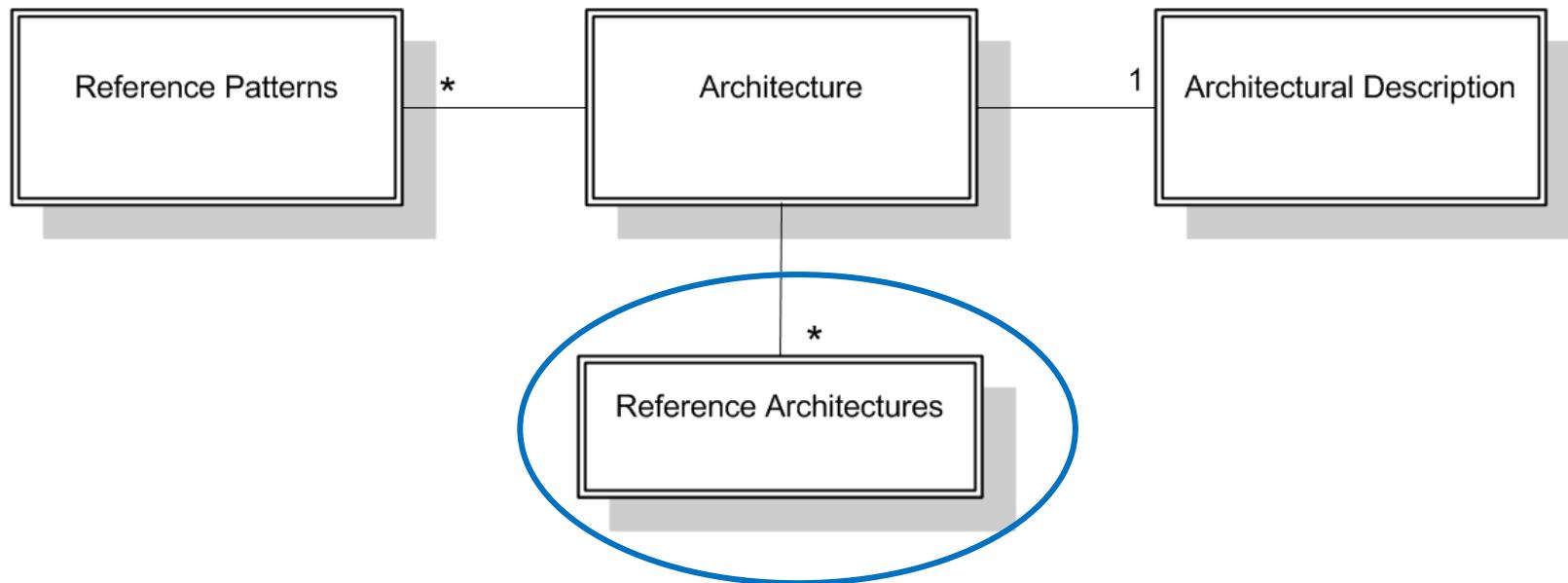
1. What do you think are the required quality attributes?
2. What is the appropriate communication style(s) for this scenario?



- Metamodel of Architectural Assets
- Architecture Styles
- **Reference Architectures**
- Architectural Description



Metamodel of Architectural Assets

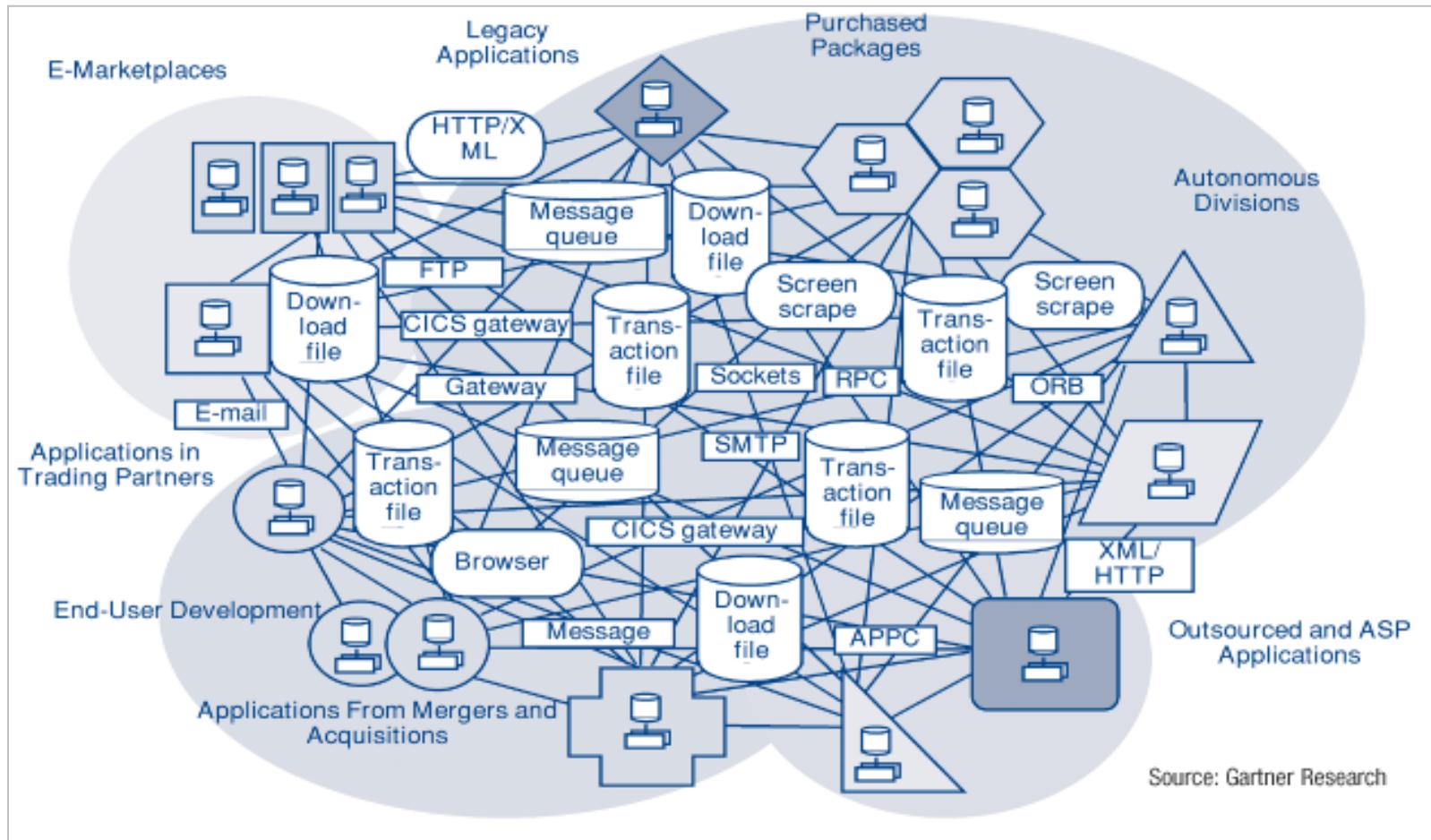


Reference Architectures

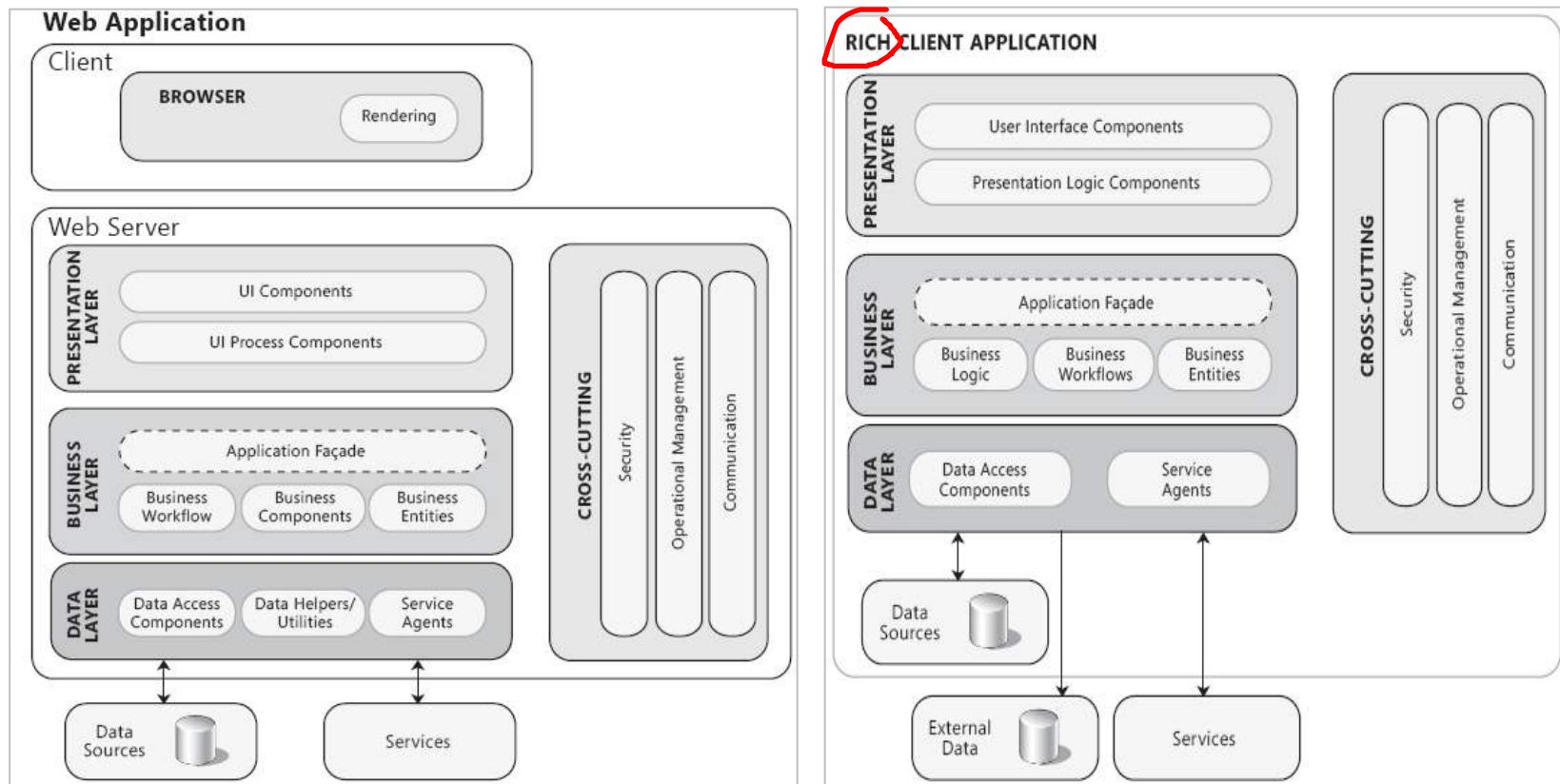
- A reference architecture is associated with **a particular domain of interest** and typically includes many different **architectural patterns**
- Subsequent slides focus on:
 - Web Architecture
 - Mobile Architecture
 - Broker Architecture
 - **SOA and Microservice Architecture**
 - Cloud Architecture
 - Serverless Architecture



Spaghetti 'Architecture' – not desirable!



Web Architecture



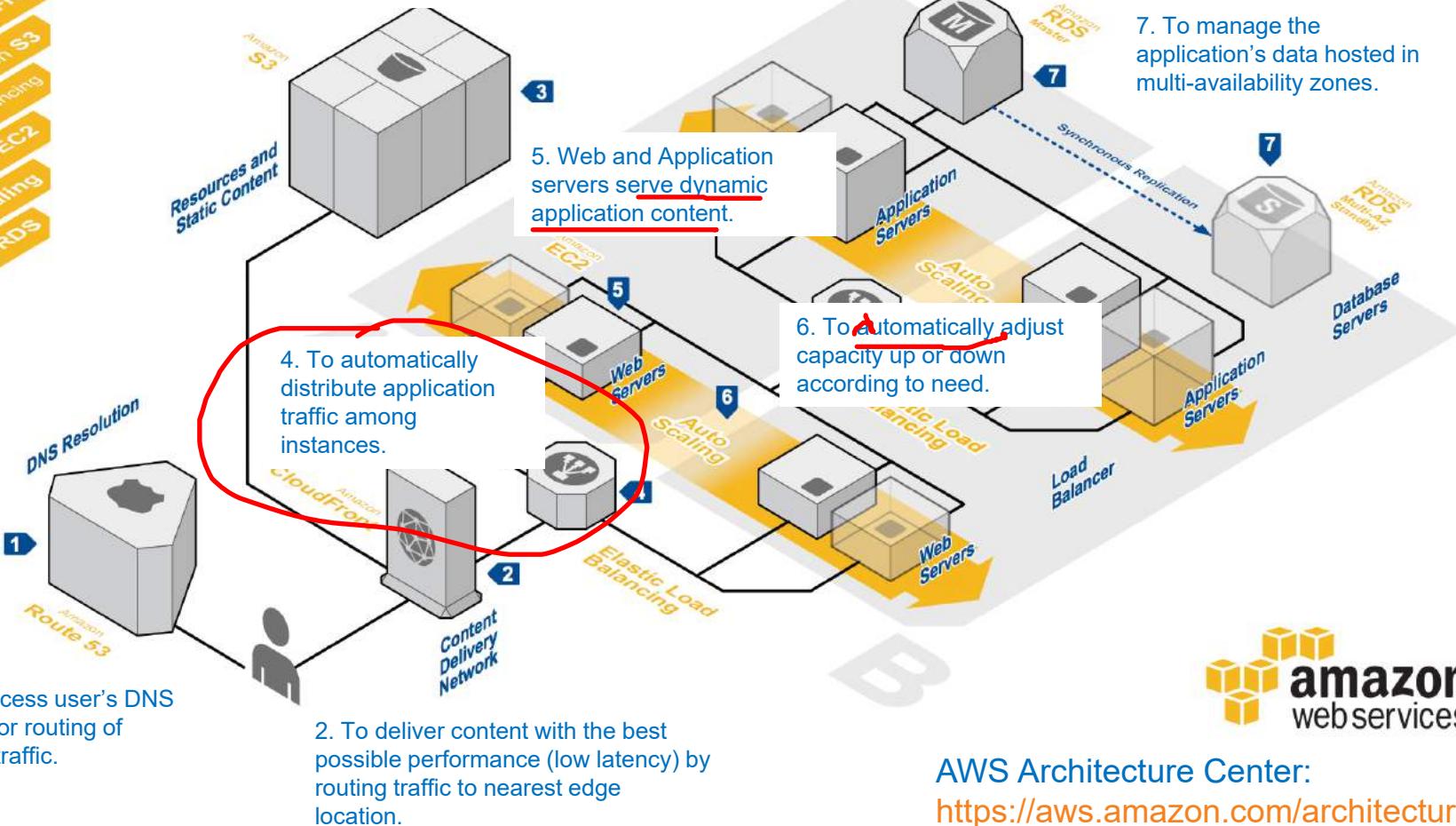
Application Architecture Knowledge Base: <http://apparch.codeplex.com/>

Amazon Web Services (AWS)

WEB APPLICATION HOSTING

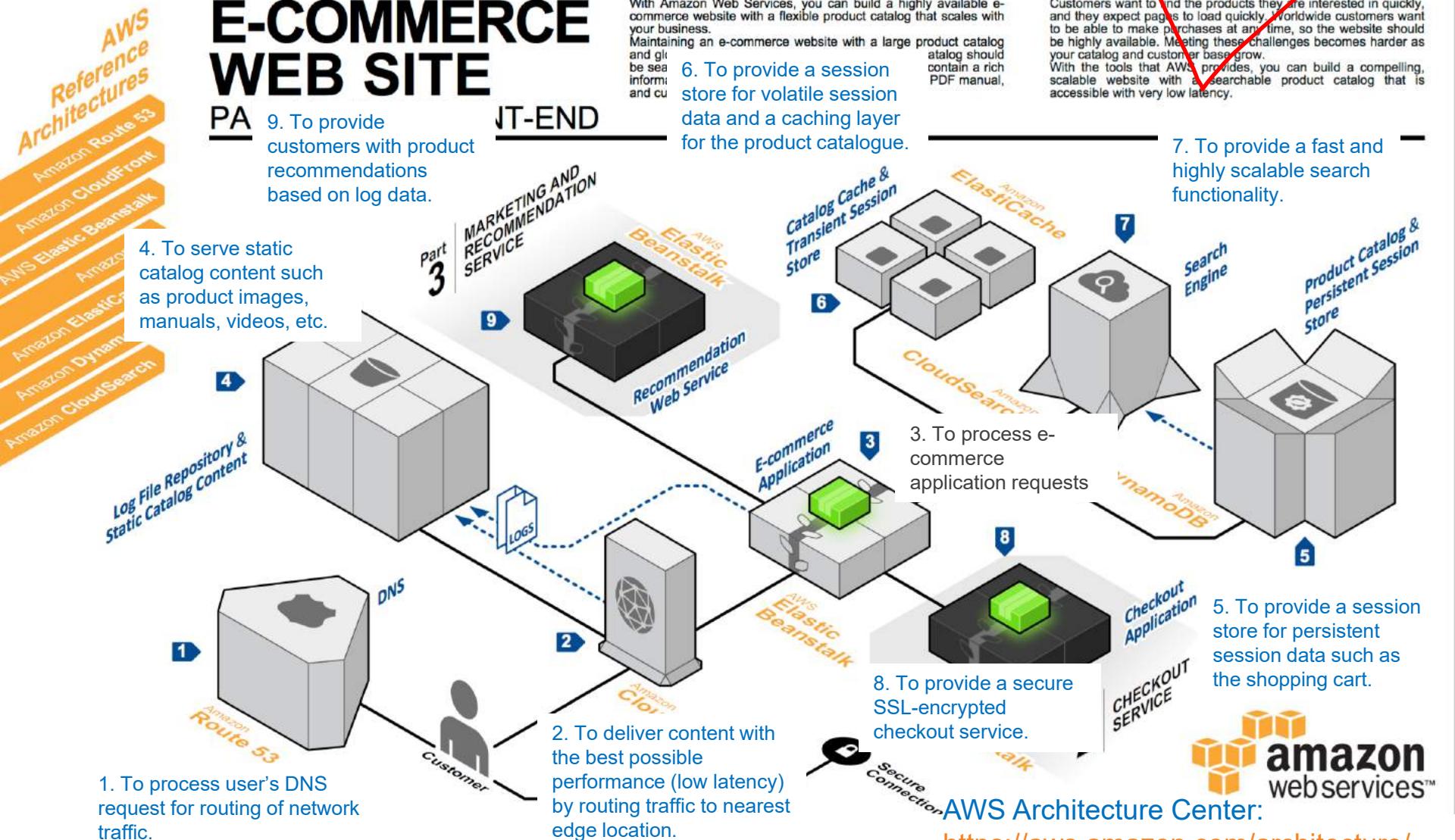
- AWS Reference Architectures
- Amazon Route 53
 - Amazon CloudFront
 - Amazon S3
 - Amazon Load Balancing
 - Elastic EC2
 - Auto Scaling
 - Amazon RDS

3. To serve resources and static content for web application.



AWS Architecture Center:
<https://aws.amazon.com/architecture/>

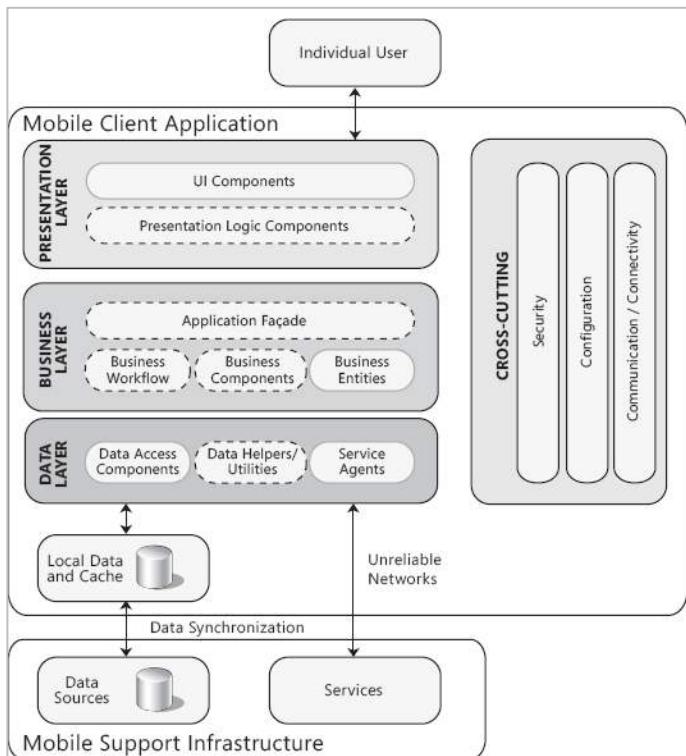
Amazon Web Services (AWS).



AWS Architecture Center:
<https://aws.amazon.com/architecture/>



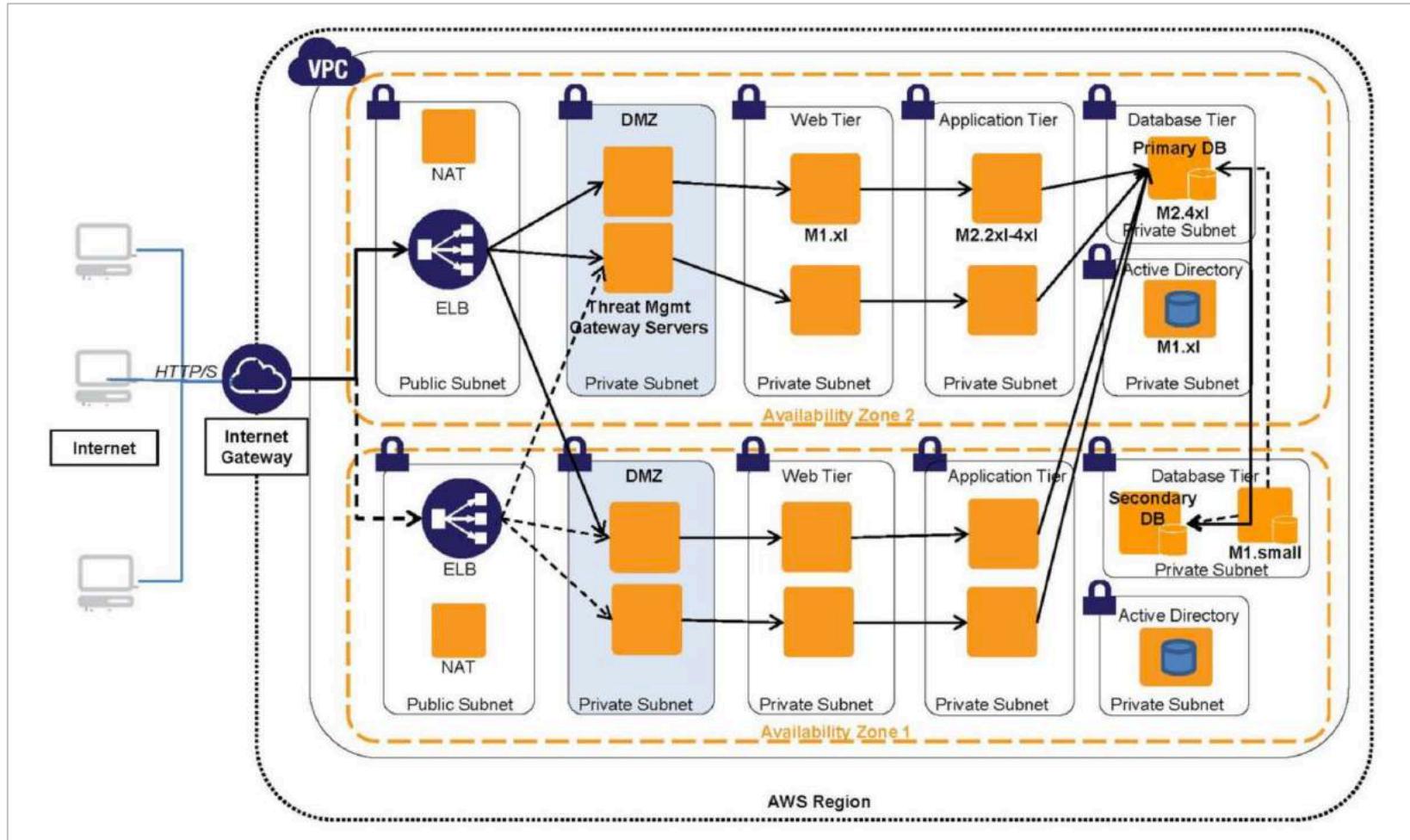
Mobile Architecture



Application Architecture Knowledge Base: <http://apparch.codeplex.com/>

IBM: http://www.ibm.com/developerworks/websphere/techjournal/1310_amrhein/1310_amrhein.html

Complex Hosted Mobile App on AWS

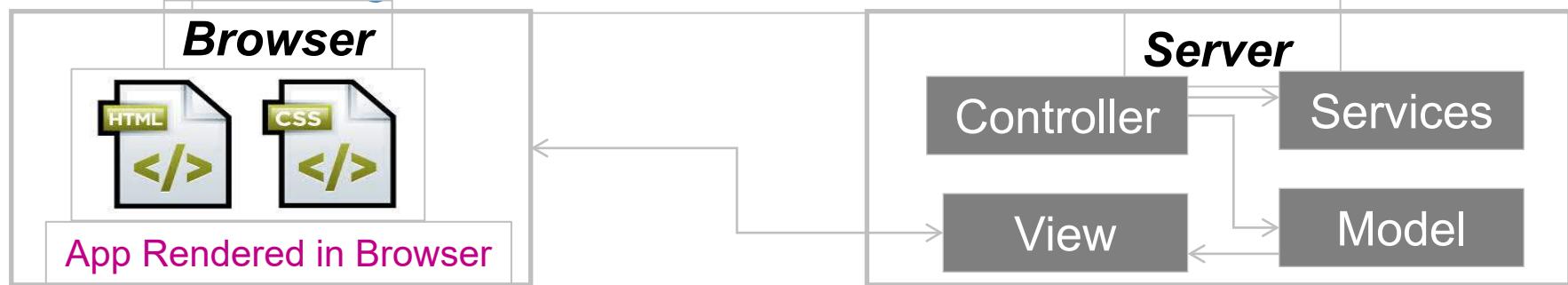


AWS Architecture Center: <https://aws.amazon.com/architecture/>

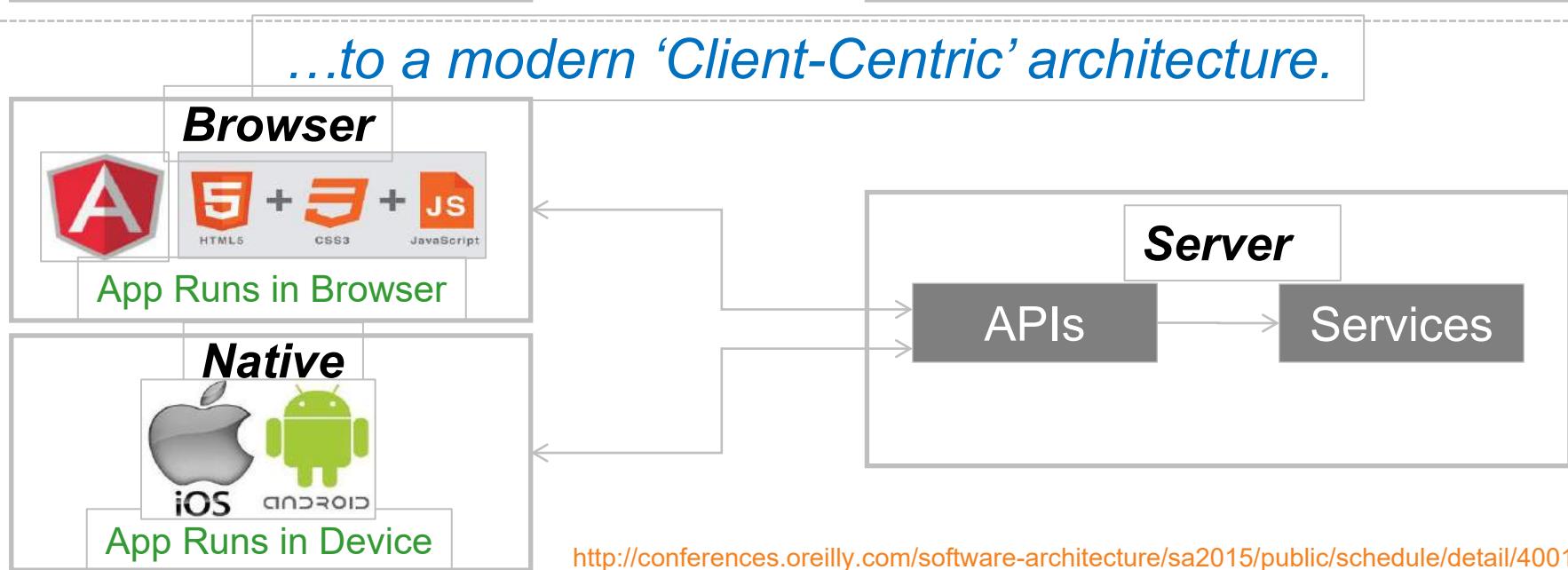


Client-Centric Architecture

Moving from a traditional Web architecture...



...to a modern 'Client-Centric' architecture.



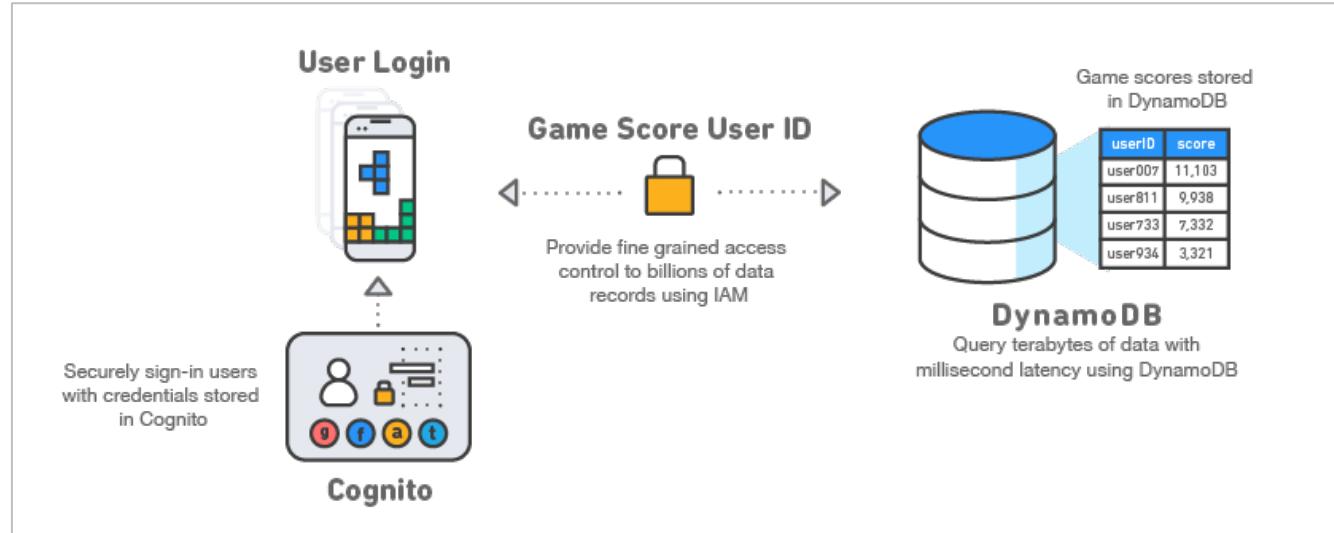
<http://conferences.oreilly.com/software-architecture/sa2015/public/schedule/detail/40012>



Mobile App Backend on AWS

Example:

Build a backend to update game scores:



Example:

Build a social messaging app:

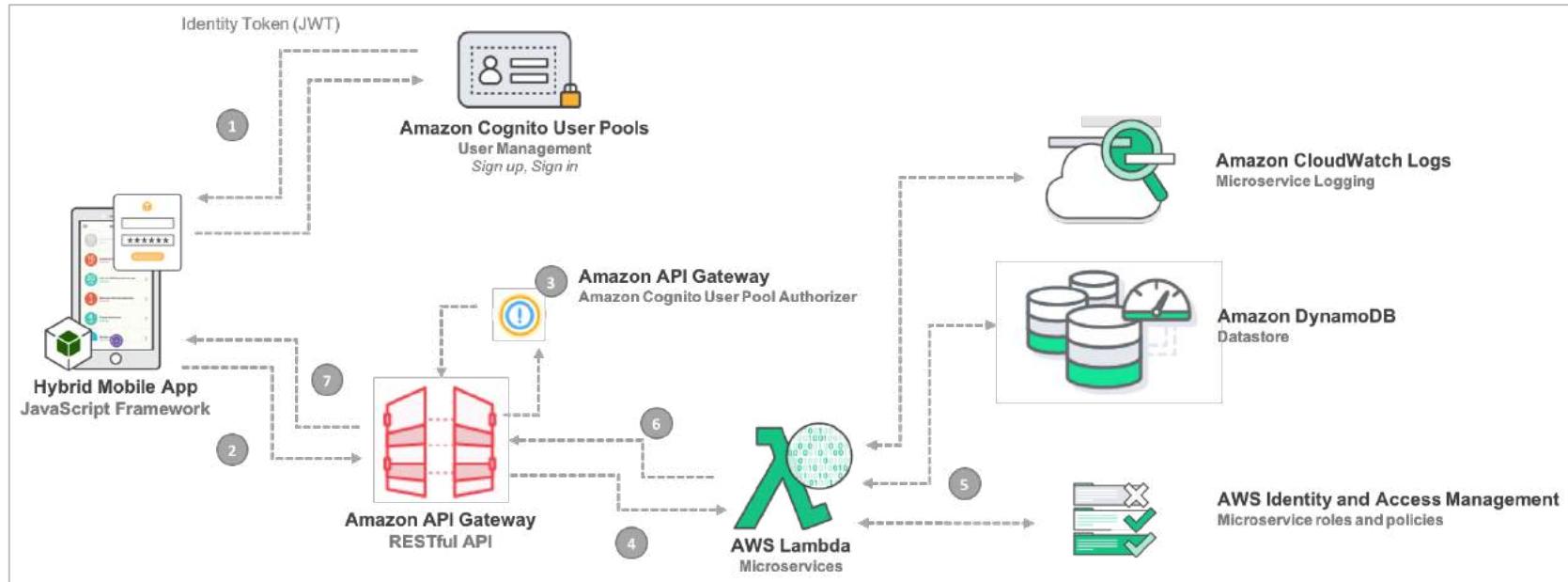


AWS Architecture Center: <https://aws.amazon.com/architecture/>

© 2011-23 National University of Singapore. All Rights Reserved.



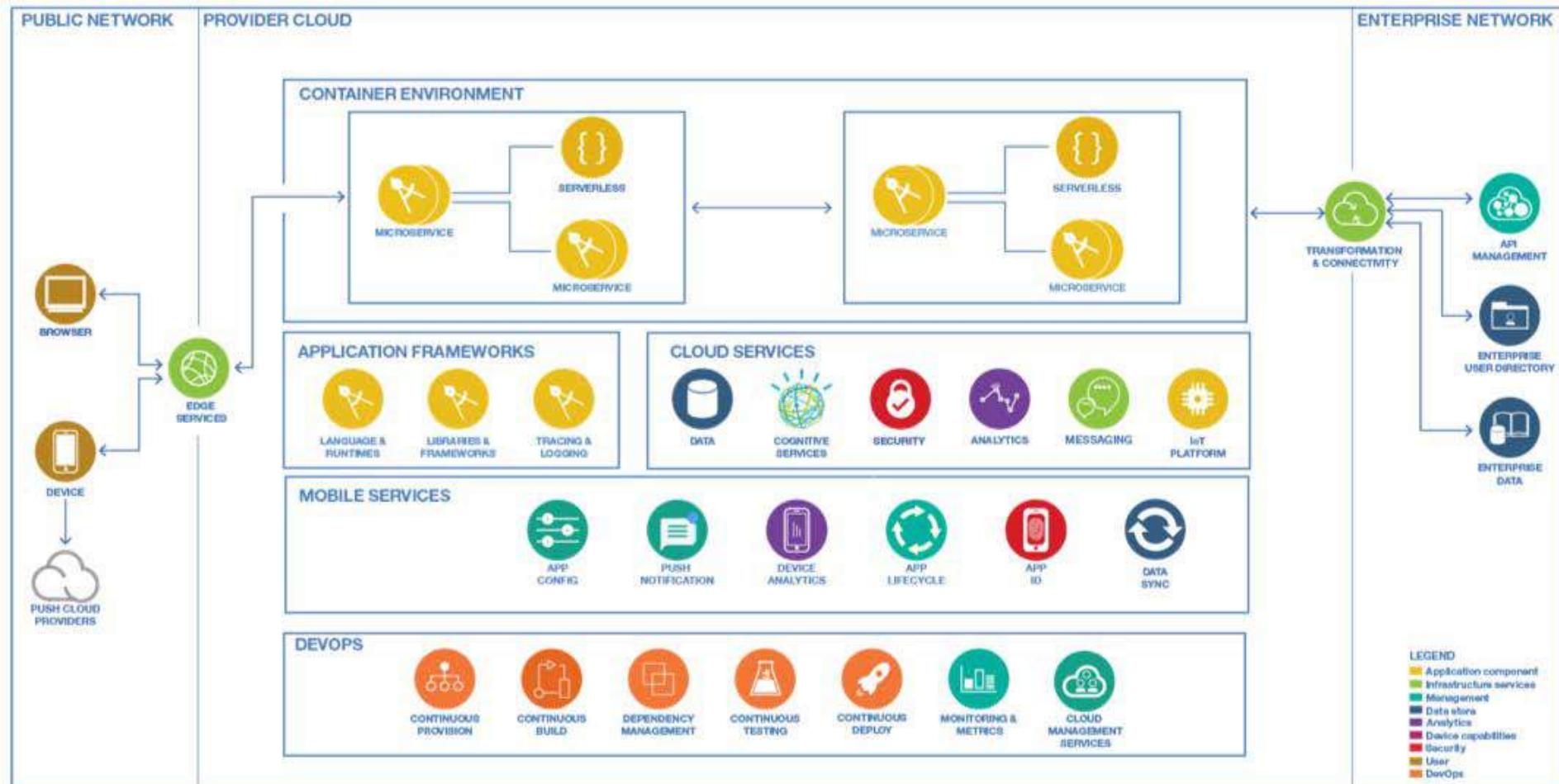
AWS Mobile Reference Architecture



AWS Architecture Center: <https://aws.amazon.com/architecture/>



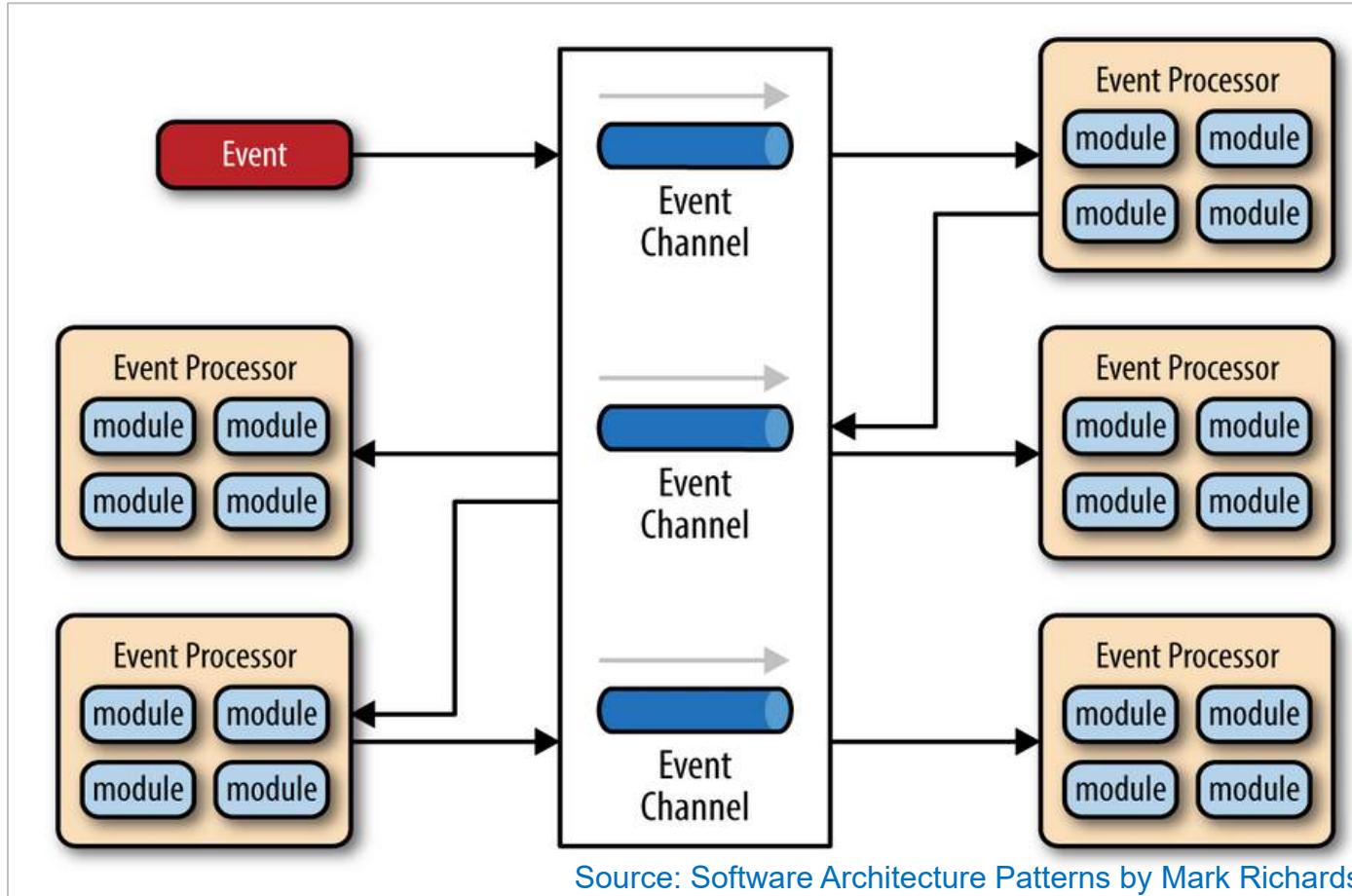
IBM Mobile Reference Architecture.



IBM: <https://www.ibm.com/cloud/architecture/architectures/mobileArchitecture/reference-architecture>



Broker Architecture

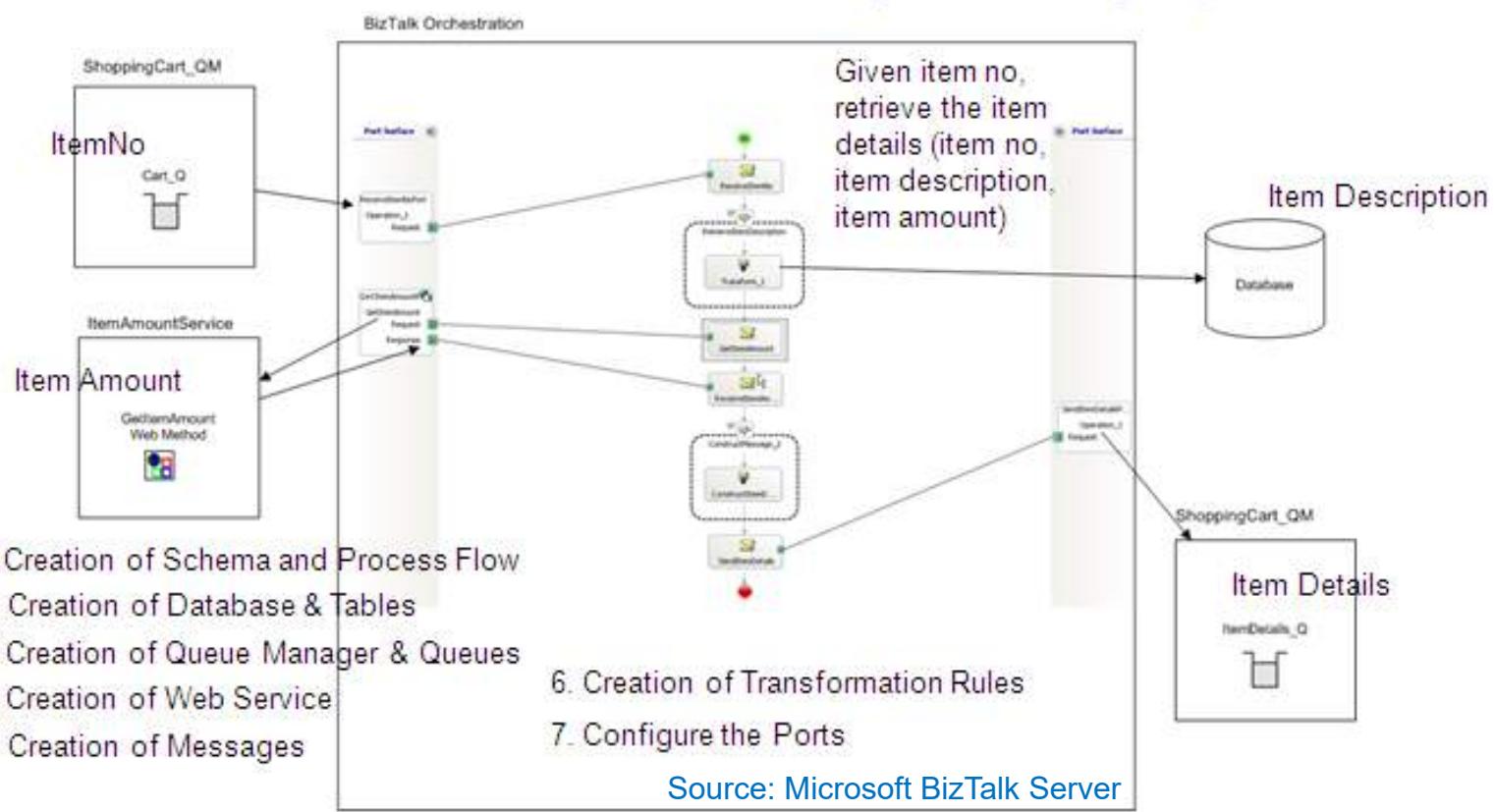


Examples: WebSphere ESB, Apache Camel, Mule ESB, etc.

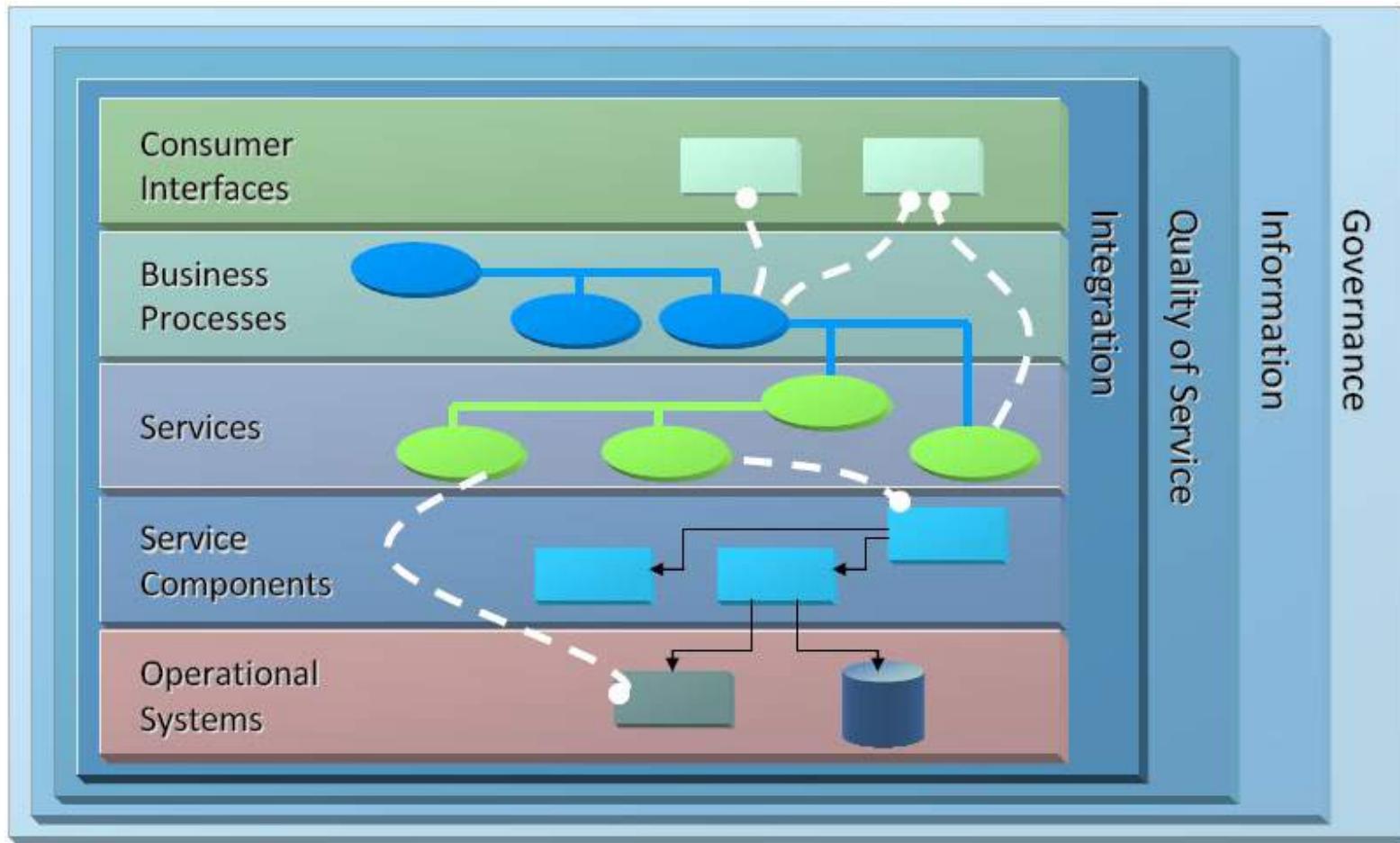
Broker Architecture. - Process Orchestration



What are the considerations to decide on this broker architecture?



SOA Architecture (Open Group)



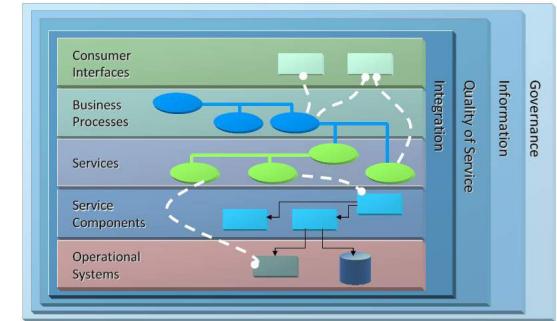
From The Open Group: <http://www.opengroup.org/subjectareas/soa>



SOA Architecture (Open Group).

An example:

- Consumer Layer: Portal to loan processing application.
- Integration and Business Process Layers: Processing of a loan processing application.
- Services Layer: Services that are required to support the loan processing application.
- Component Layer: Software components that support the realization and implementation of the services.
- Operational Systems Layer: Actual runtime environment in which the components, legacy systems, all back-end applications, and packaged applications reside and run.

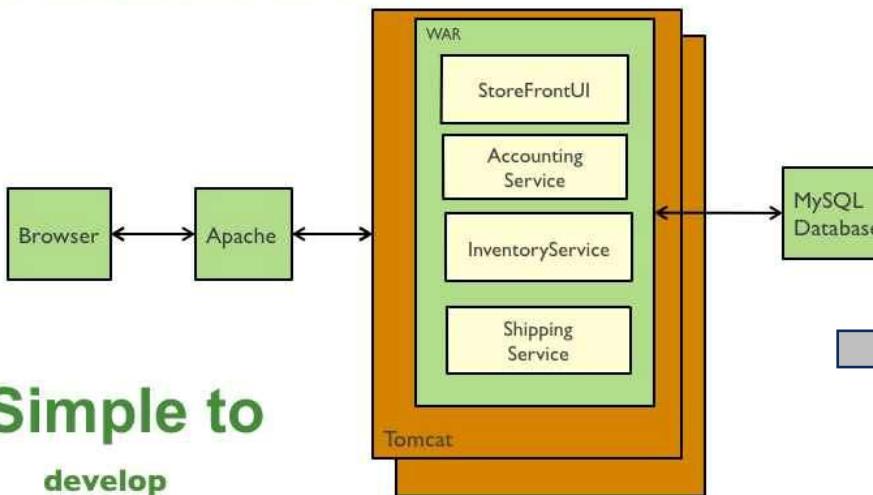


From The Open Group:
<http://www.opengroup.org/subjects/soa>



Microservices Architecture

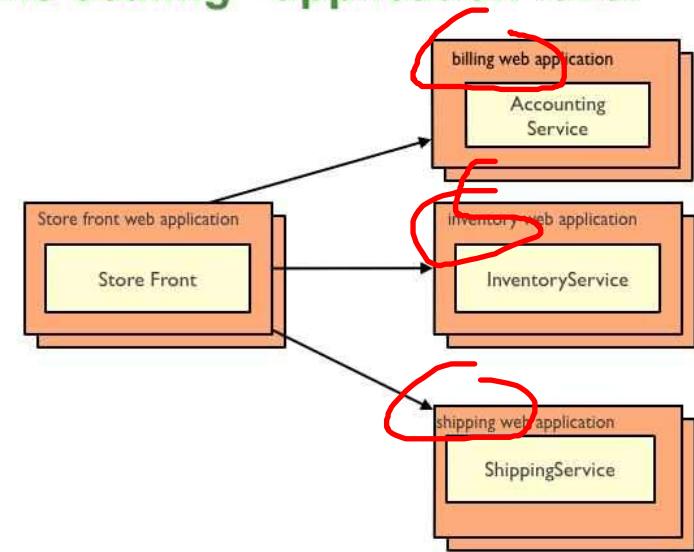
Traditional web application architecture



Simple to

develop
test
deploy
scale

Y axis scaling - application level



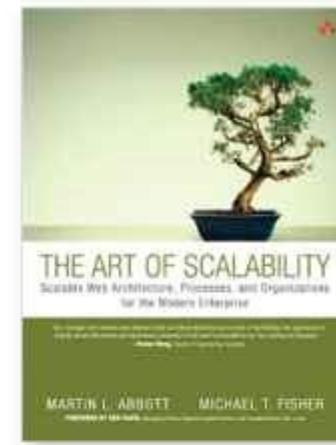
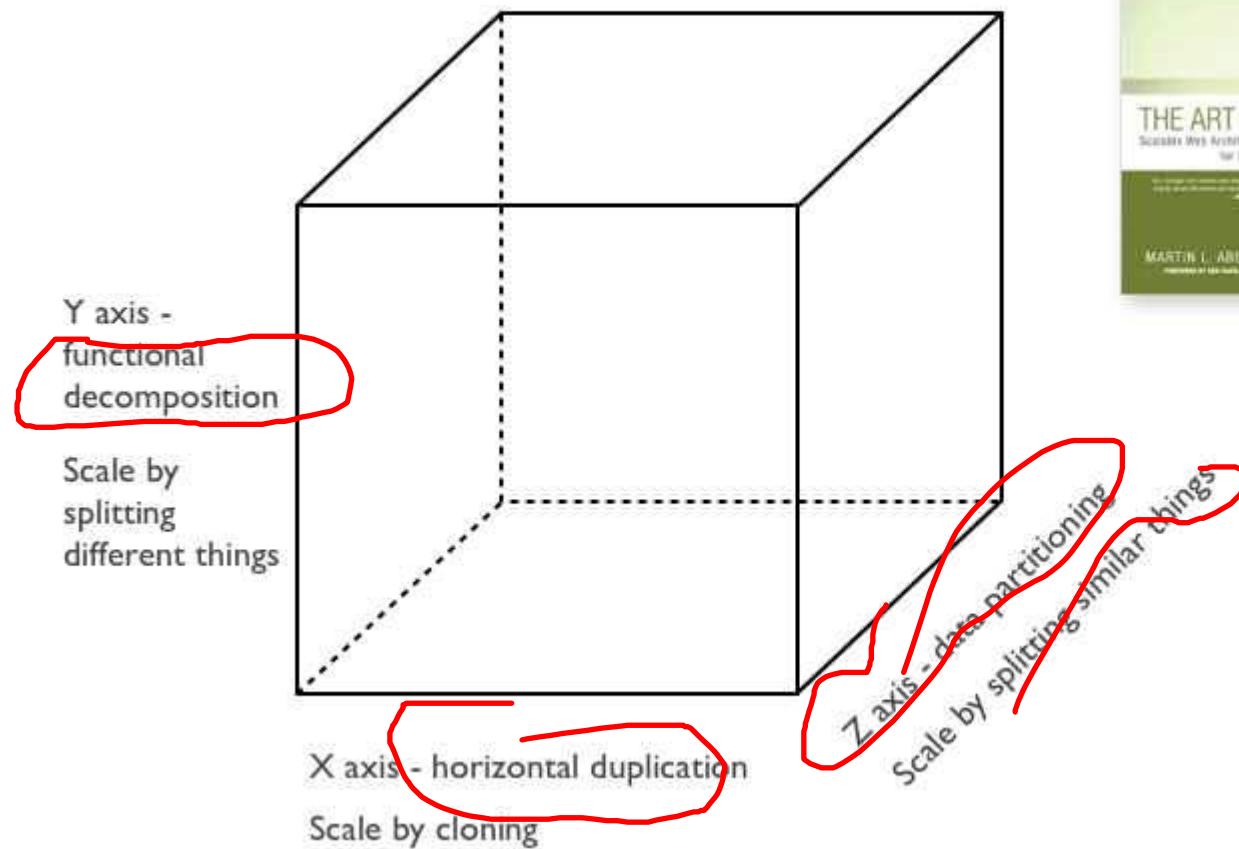
Apply X axis cloning and/or Z axis partitioning to each service

<http://microservices.io/patterns>



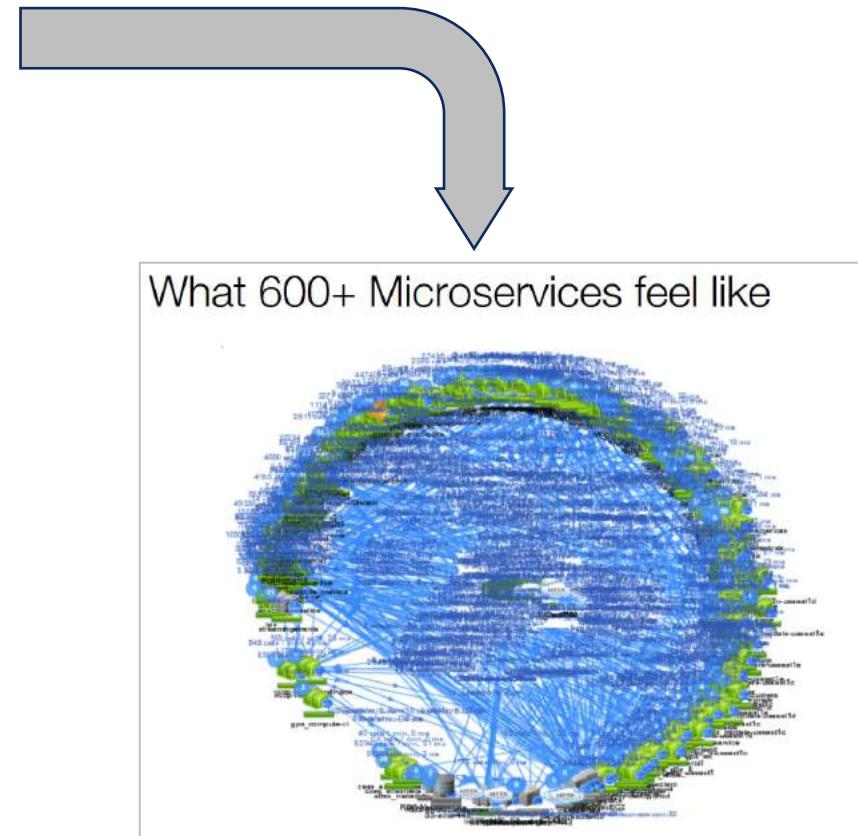
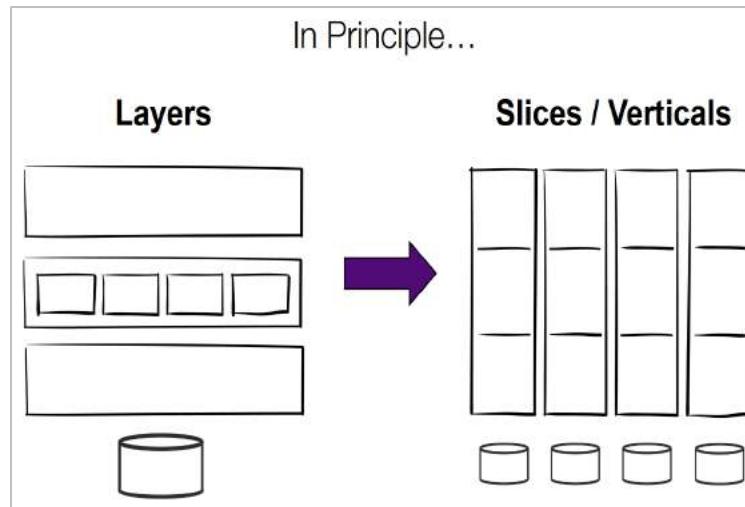
Microservices Architecture – the Scale Cube

3 dimensions to scaling





Microservices Architecture



<https://martinfowler.com/microservices/>

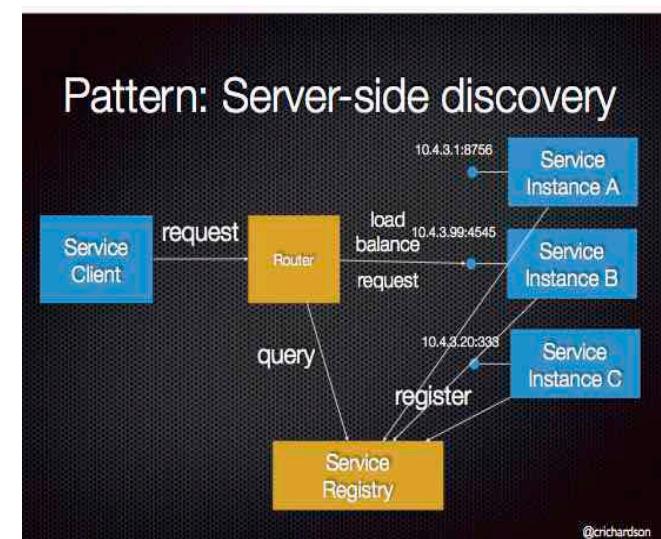
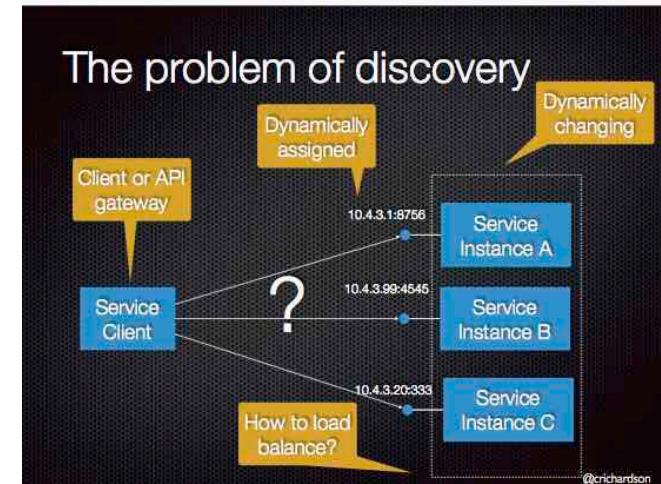
http://www.embarc.de/wp-content/uploads/2016/04/SANY16_TOH.pdf



Microservices Architecture - Service Discovery

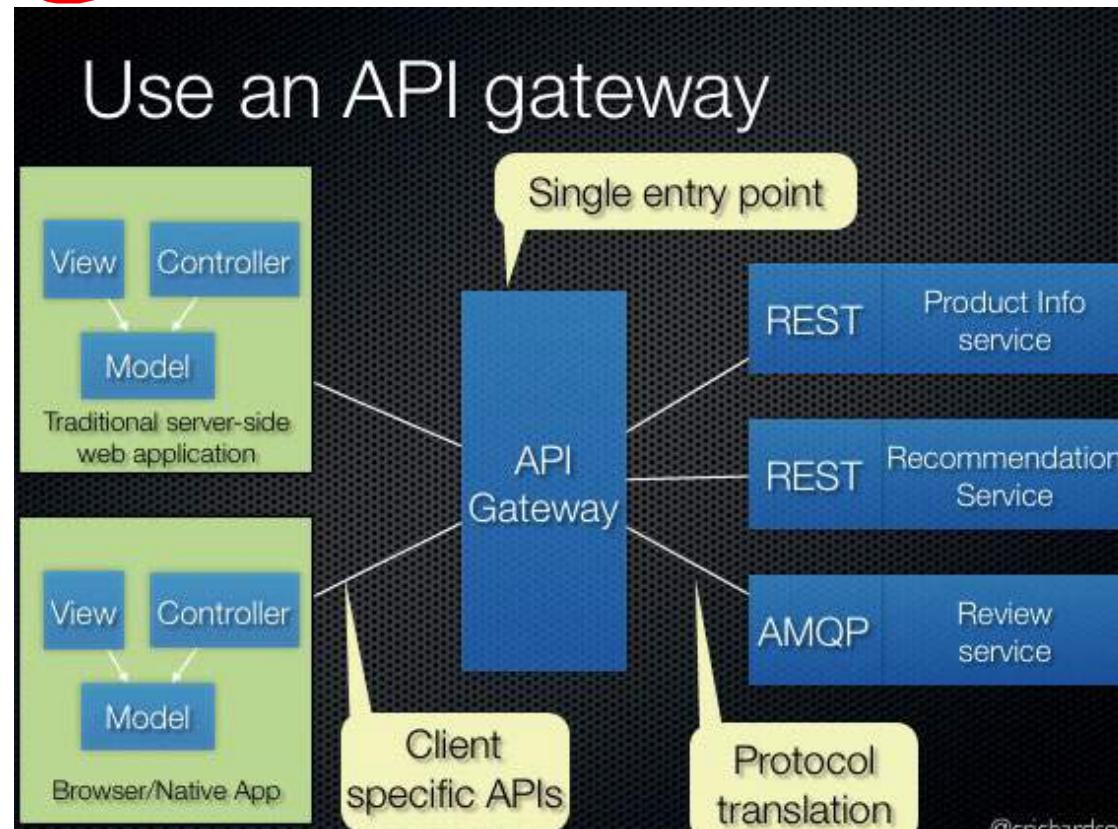
We must implement a mechanism for that enables the clients of services to make requests to a **dynamically changing** set of ephemeral service instances.

The client makes a request via a **router** that runs at a well known location. The router queries a **service registry** and forwards the request to an available service instance.



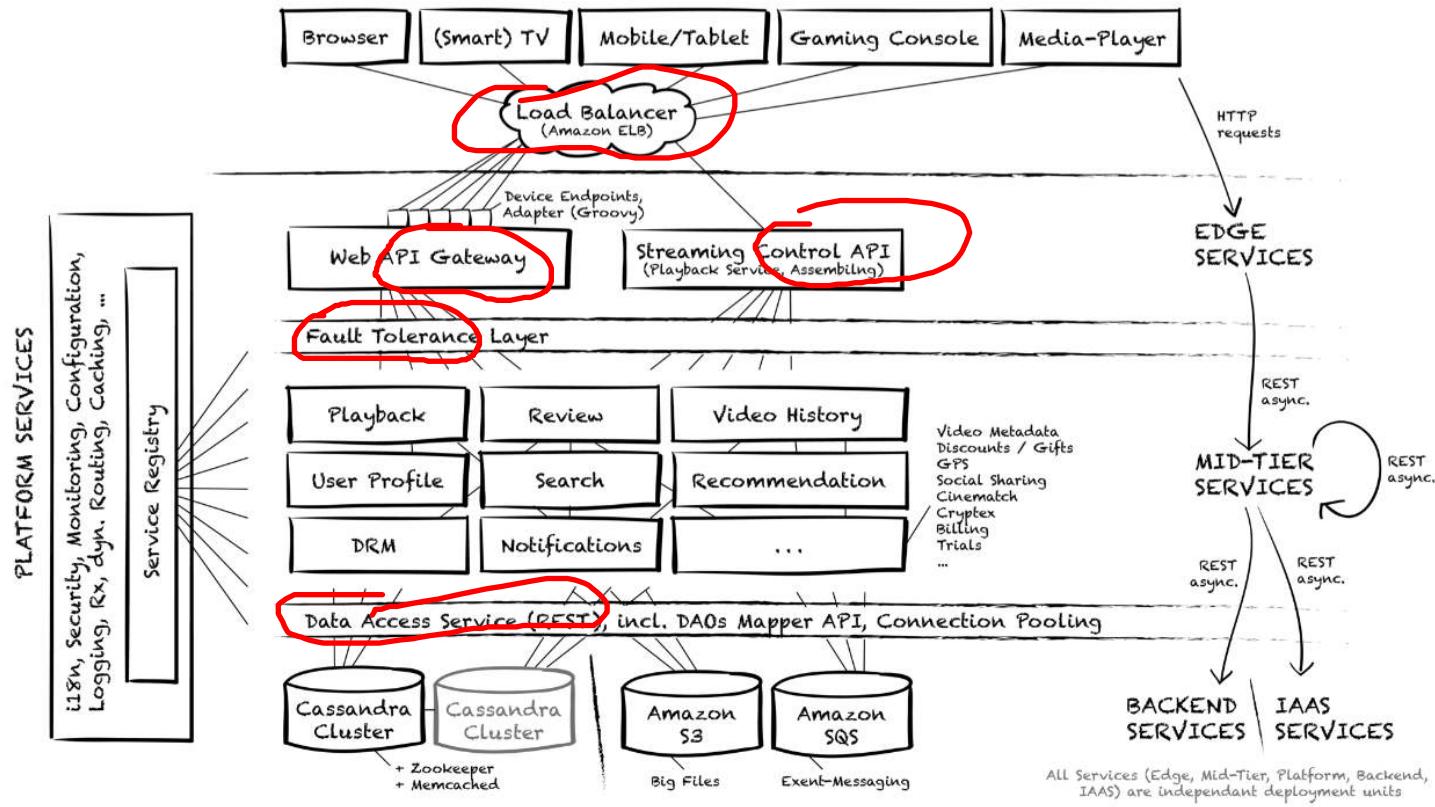
<http://microservices.io/patterns>

Microservices Architecture - role of an API Gateway



Microservices Architecture

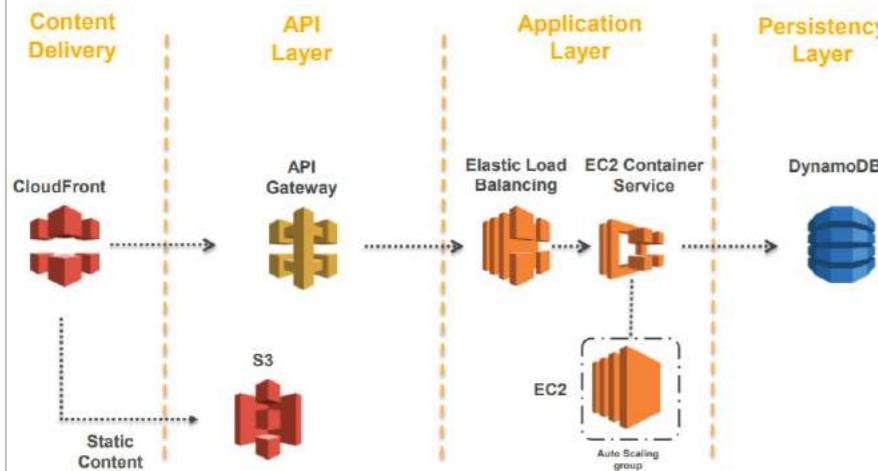
Netflix Architectural Overview (simplified)



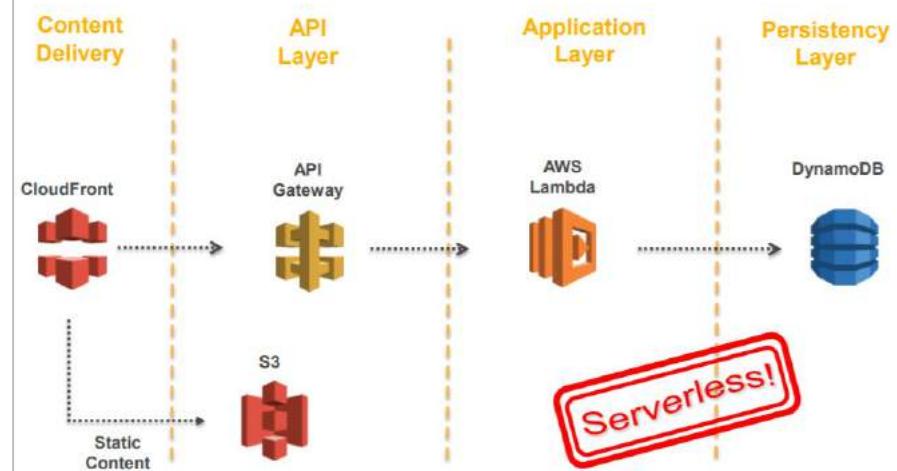
http://www.embarc.de/wp-content/uploads/2016/04/SANY16_TOOTH.pdf

Typical Microservices Architectures on AWS.

A Typical Microservice Architecture on AWS



A Typical Microservice Architecture on AWS



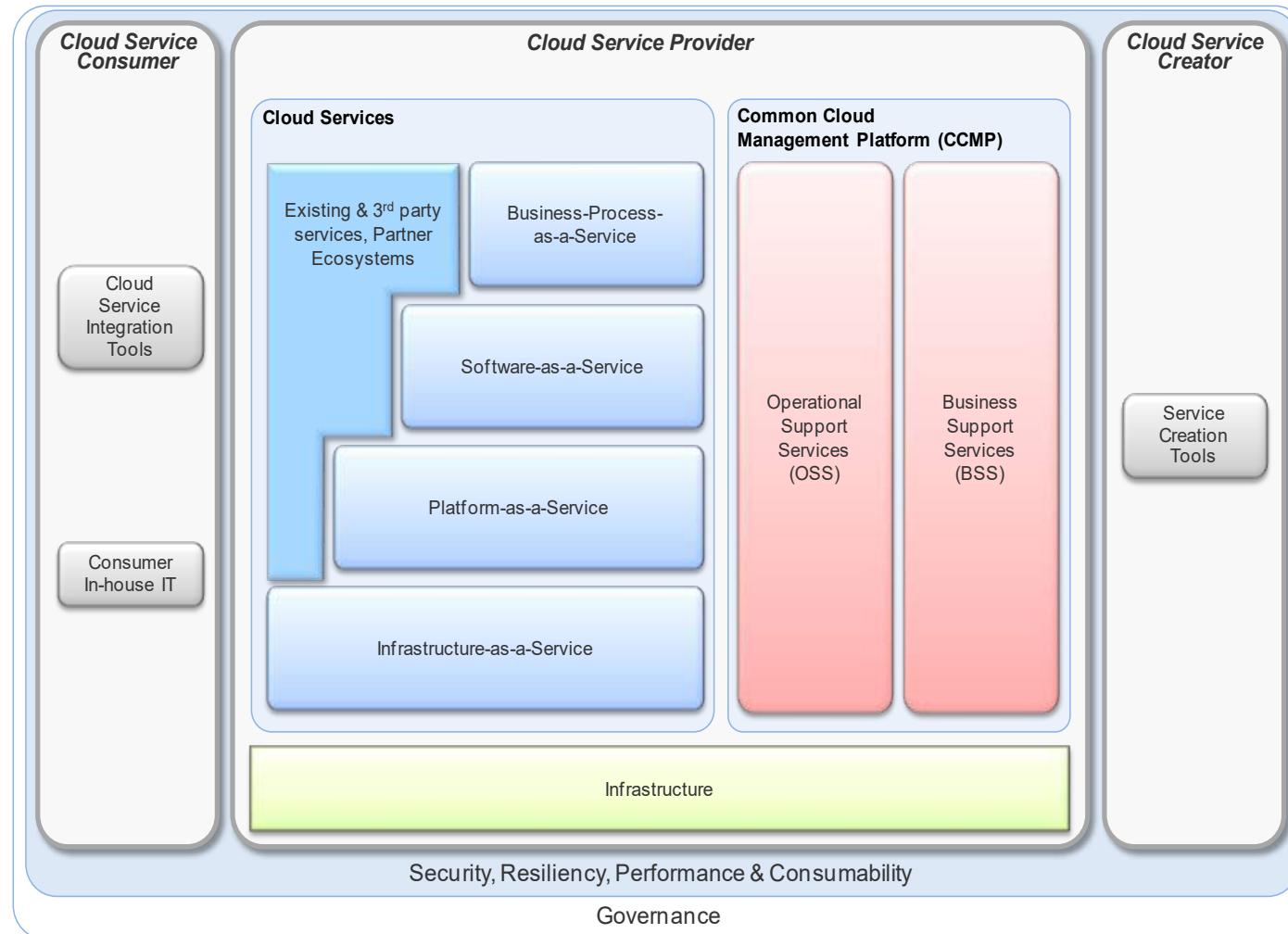
Traditional – with application servers running on EC2 instances

Serverless – with AWS Lambda

https://aws-de-media.s3.amazonaws.com/images/AWS_Summit_Berlin_2016/sessions/pushing_the_boundaries_1300_microservices_on_aws.pdf



Cloud Computing Reference Architecture (IBM)

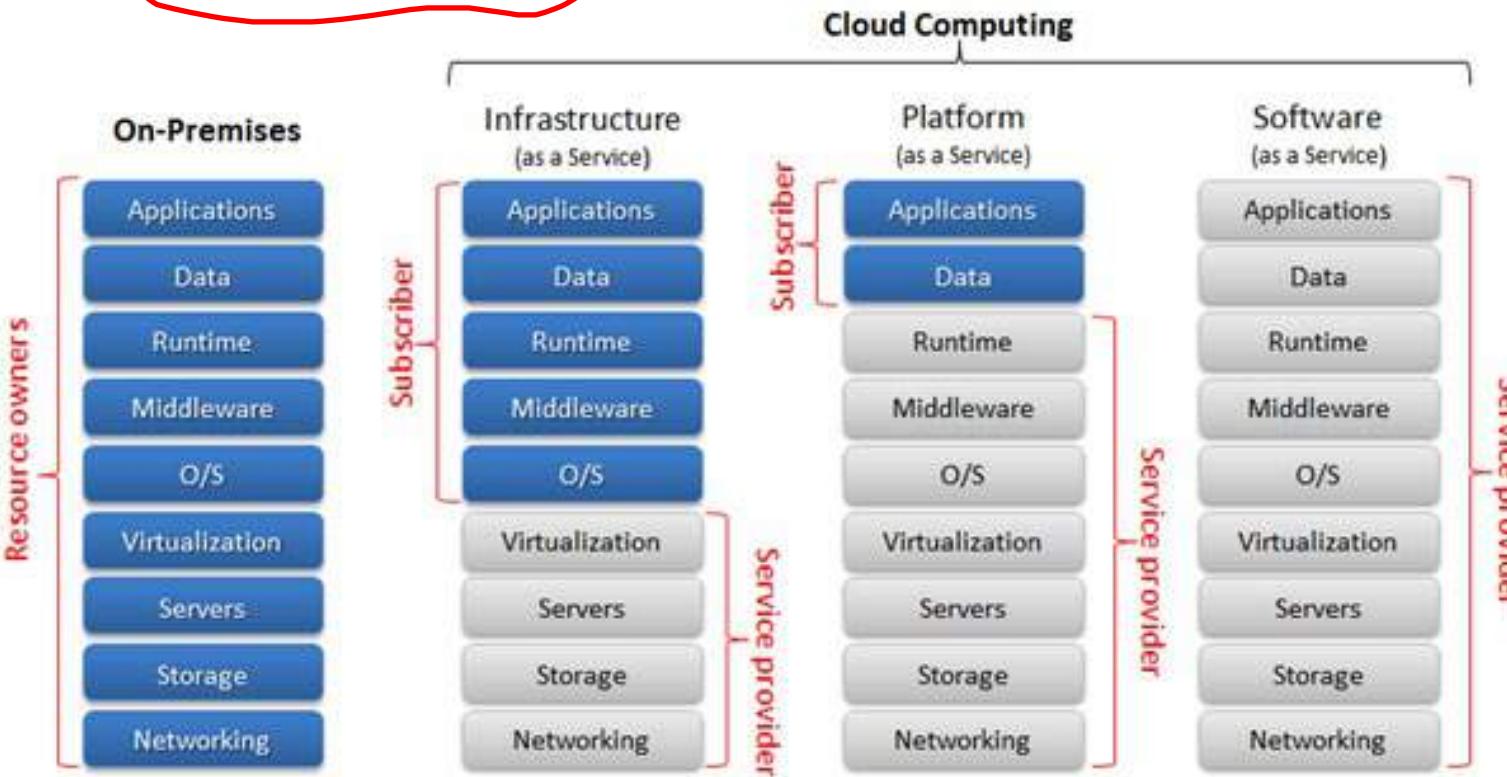


IBM: https://www-05.ibm.com/it/cloud/downloads/Cloud_Computing.pdf

IBM: <https://www.ibm.com/cloud/garage/architectures>

Cloud Architecture - Service Models

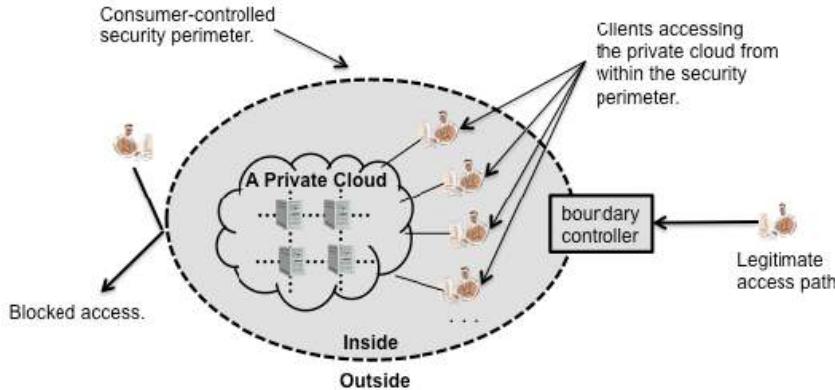
Separation of Responsibilities



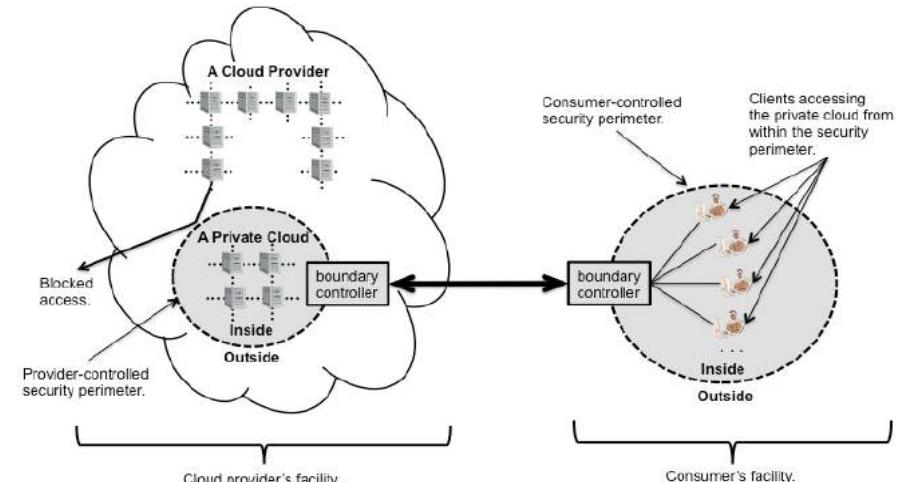
Microsoft: <https://blogs.technet.microsoft.com/yungchou/2010/11/15/cloud-computing-primer-for-it-pros/>

Cloud Architecture - Deployment Models

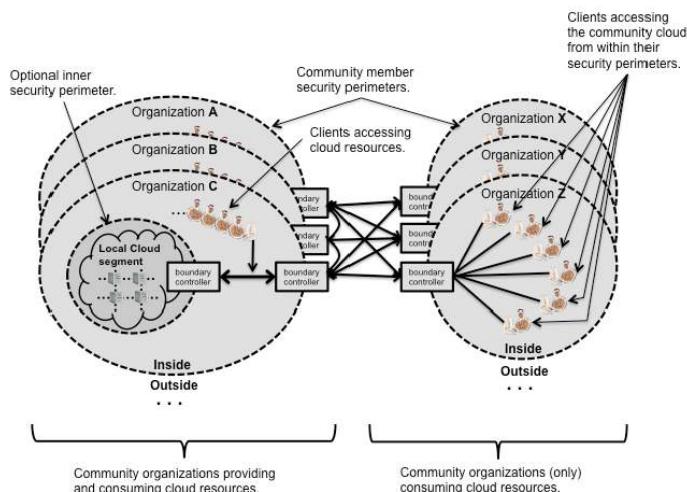
- On-site Private Cloud



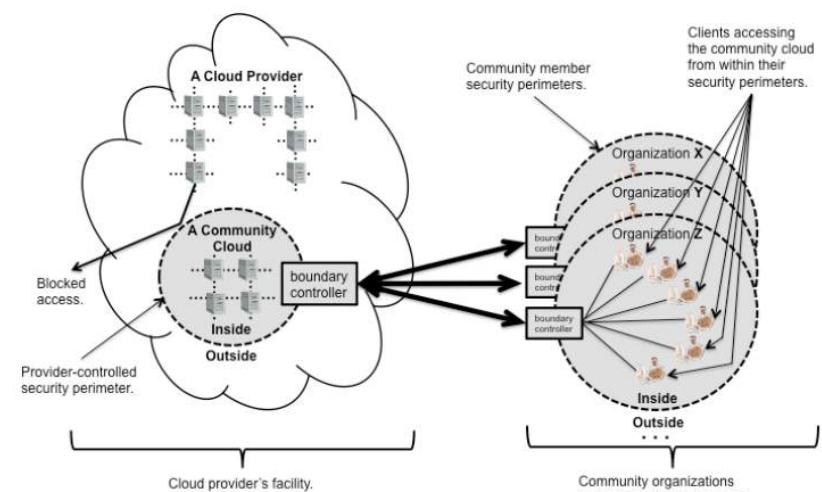
- Outsourced Private Cloud



- On-Site Community Cloud



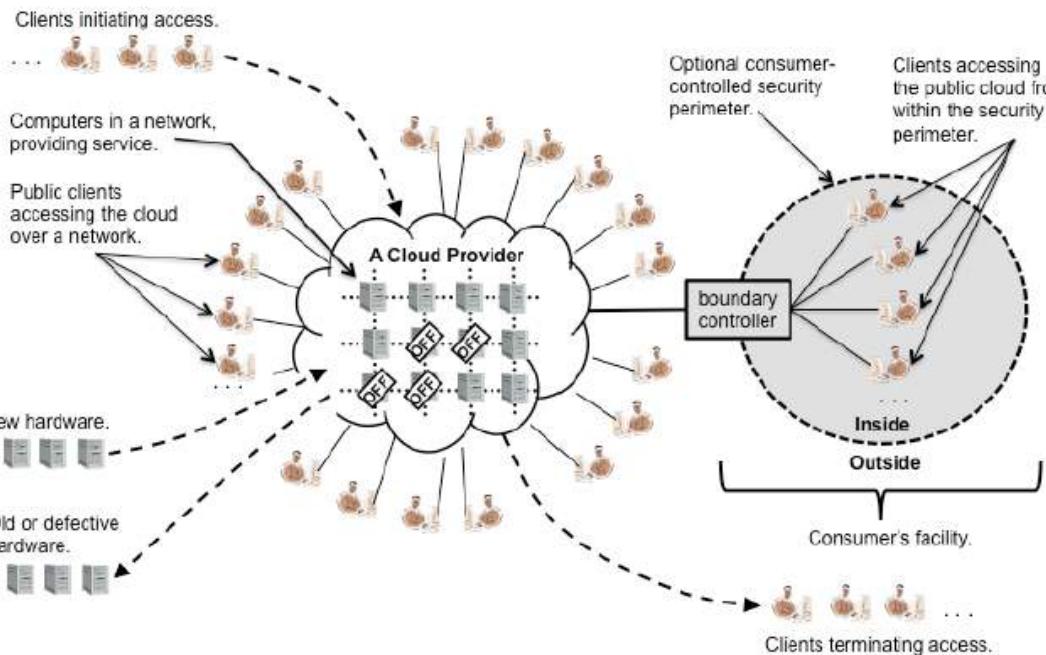
- Outsourced Community Cloud



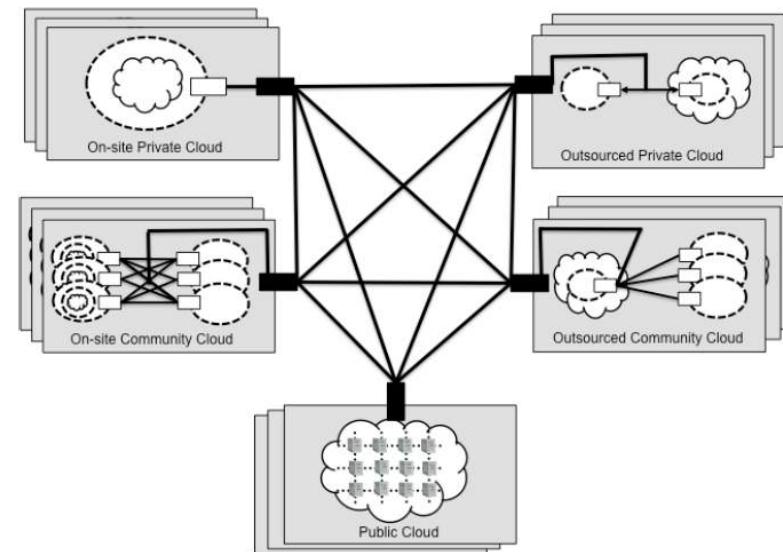


Cloud Architecture - Deployment Models

- Public Cloud

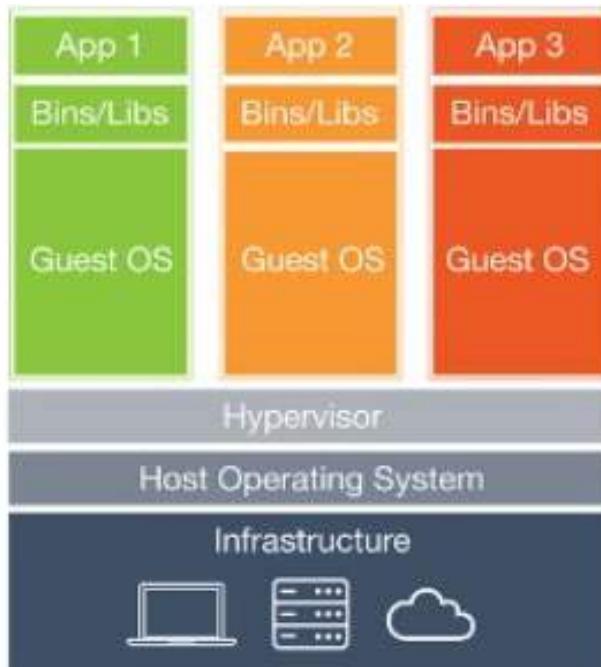


- Hybrid Cloud

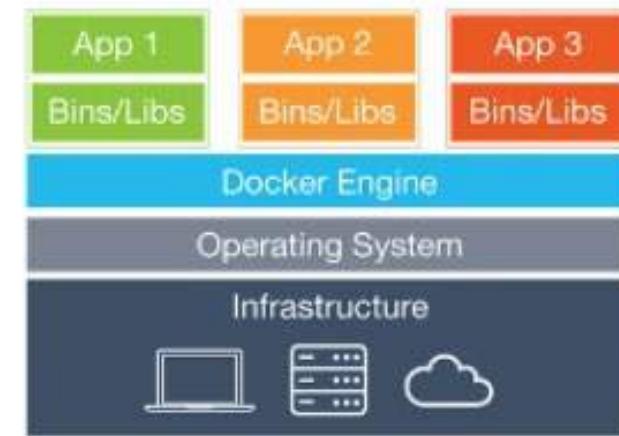


<http://www.nist.gov/itl/csd/cloud-102511.cfm>

Virtual Machines compared to Containers



Virtual Machine



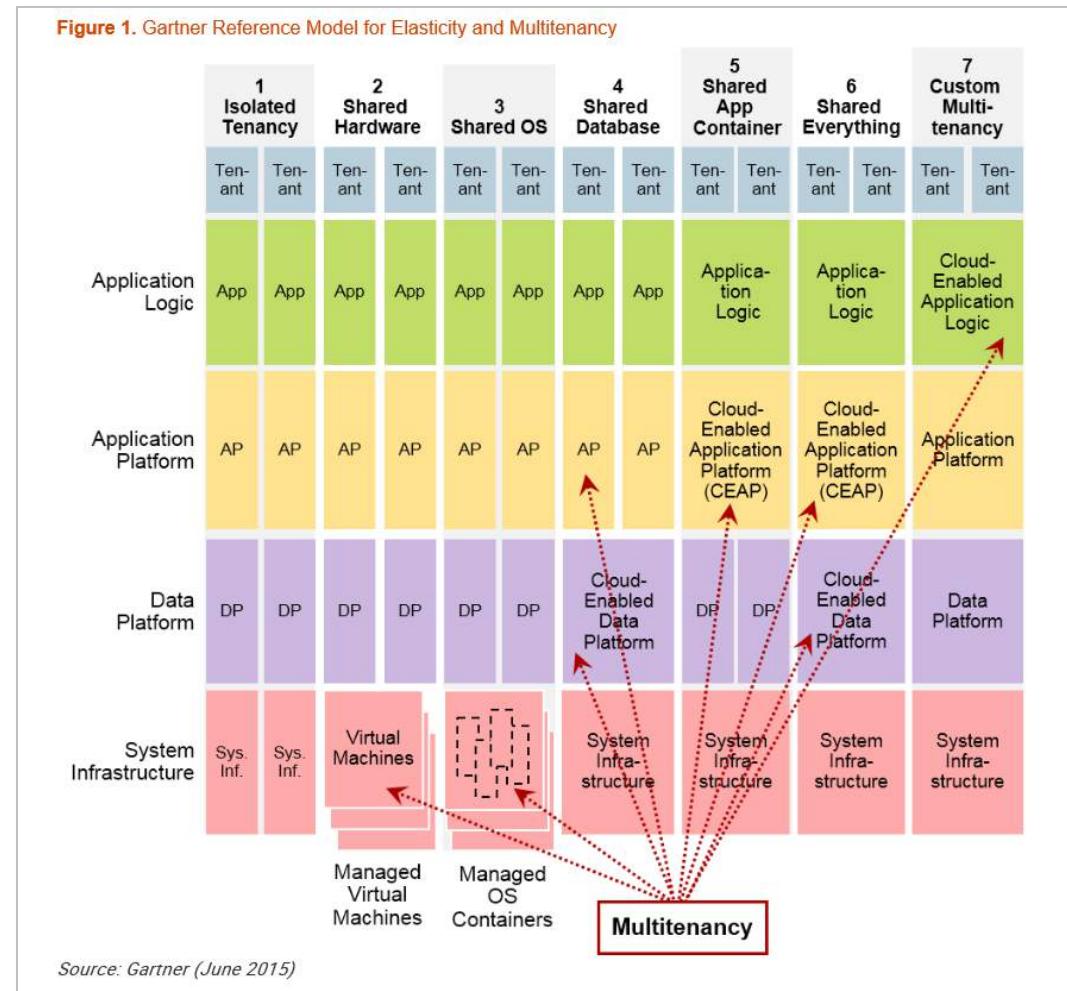
Container



Cloud Architecture - Multitenancy

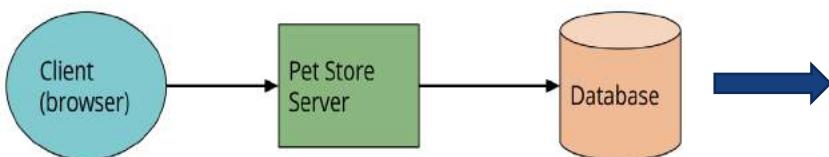
Multitenancy refers to the **isolation of independent organization workloads** in a **shared environment**.

Elasticity refers to the characteristic of the software environment where the underlying IT resources are provided in **proportion to changing needs** as they increase or decrease.

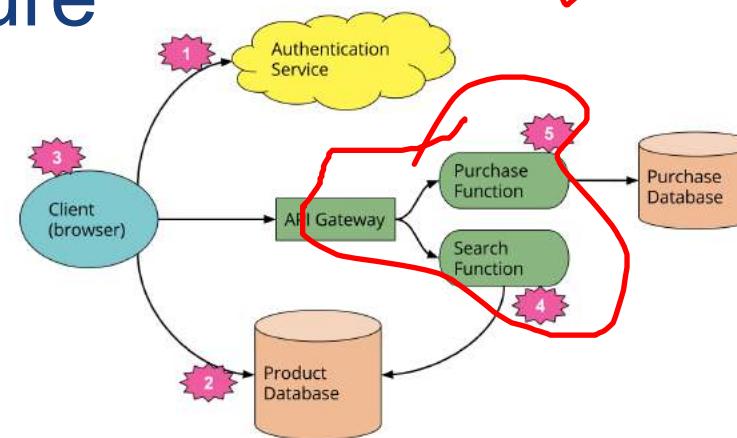


Gartner Reference Model for Elasticity and Multitenancy, ID: G00275010

Serverless Architecture



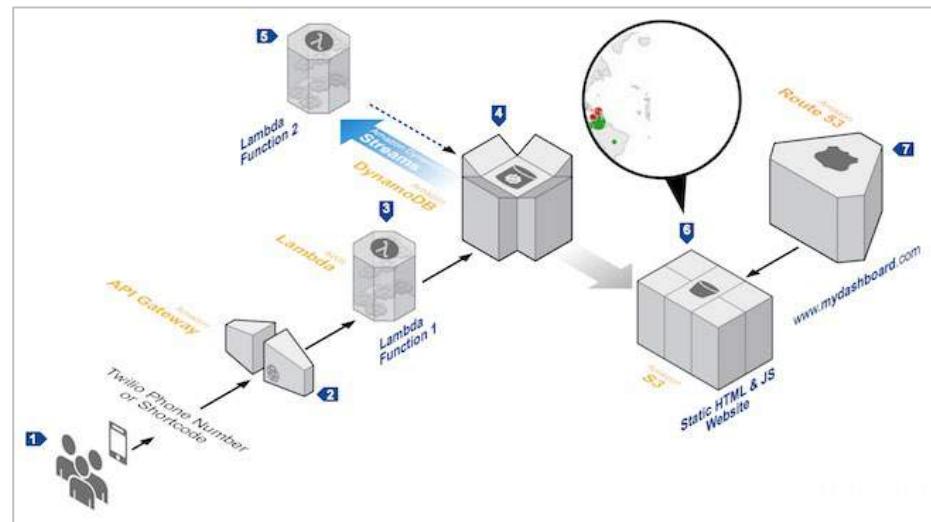
<http://martinfowler.com/articles/serverless.html>



Serverless Microservice Architecture



<https://aws.amazon.com/blogs/compute/microservices-without-the-servers/>

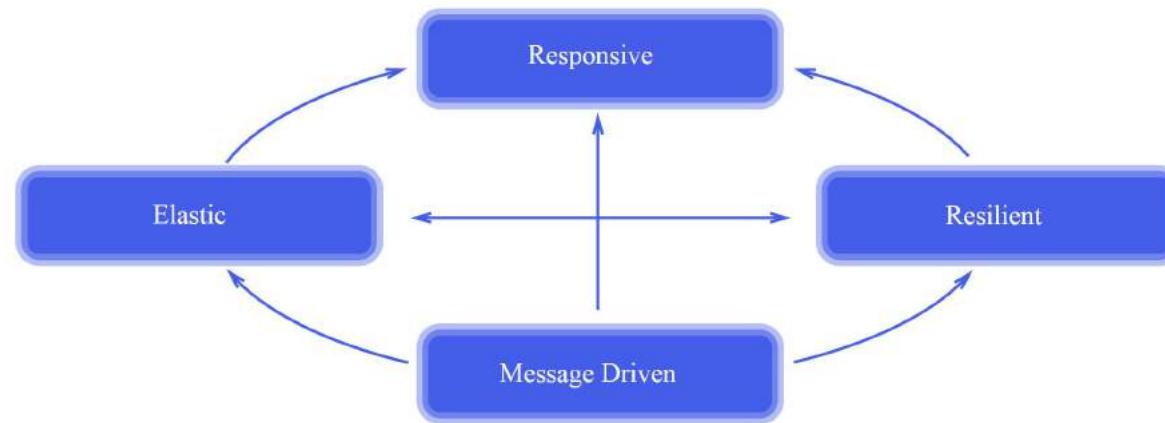


<http://www.allthingsdistributed.com/2016/06/aws-lambda-serverless-reference-architectures.html>



The Reactive Manifesto. – Guidance for Architectures

Reactive systems focus on providing **rapid** and **consistent** response times, establishing reliable upper bounds so they deliver a consistent quality of service.



The system stays responsive **under varying workload**.

The system stays responsive **in the face of failure**.

Reactive Systems rely on asynchronous message-passing to establish a boundary between components that ensures **loose coupling, isolation and location transparency**.

<http://www.reactivemanifesto.org/>



Discussion.



Discuss the following open-ended questions:

1. Mobile Architecture

What are the key decision drivers to determine your choice of mobile application development?

2. SOA / Microservices Architecture / Serverless Architecture

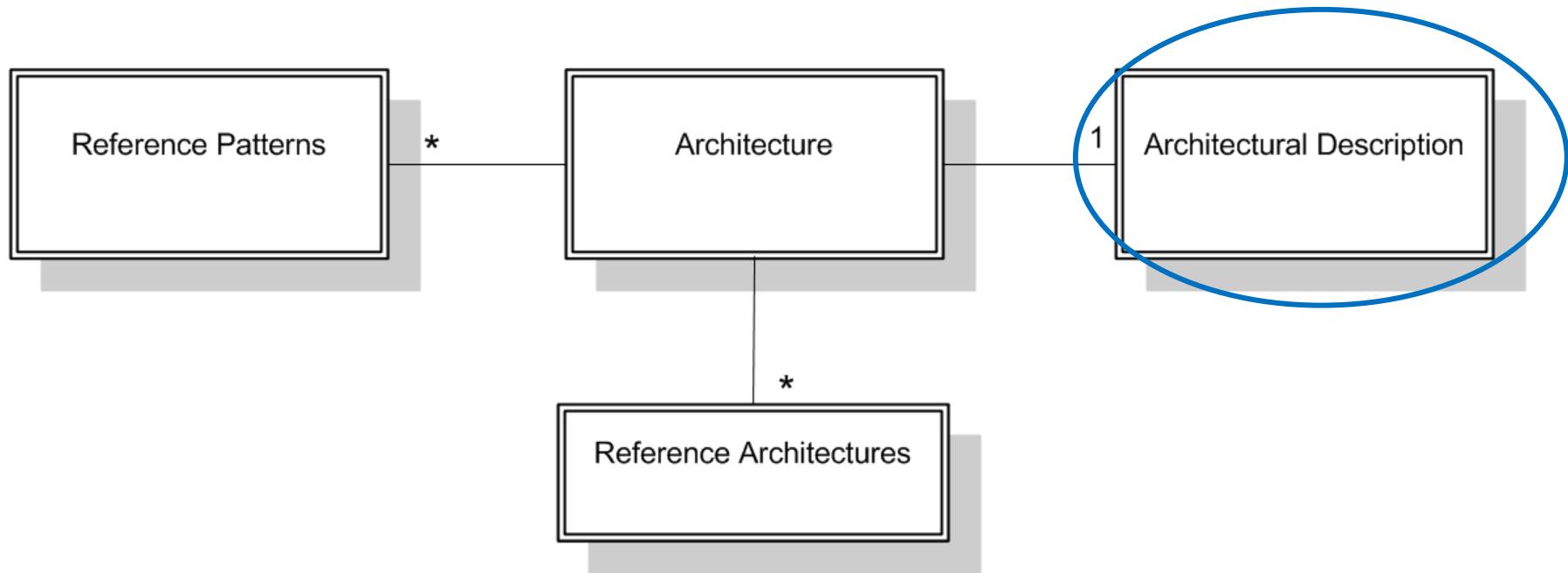
Assume you will likely adopt the SOA or Microservices architecture for the enterprise. What are the key concerns / factors / issues to be considered. If you have already adopted, have you achieved the benefits and what are the lessons learnt?

3. Cloud Architecture

Assume you are tasked to evaluate cloud adoption, what type of applications are suitable to be deployed to the cloud? What type of applications are not? What are the key considerations to determine the choice of the cloud deployment?

- Metamodel of Architectural Assets
- Architecture Styles
- Reference Architectures
- Architectural Description

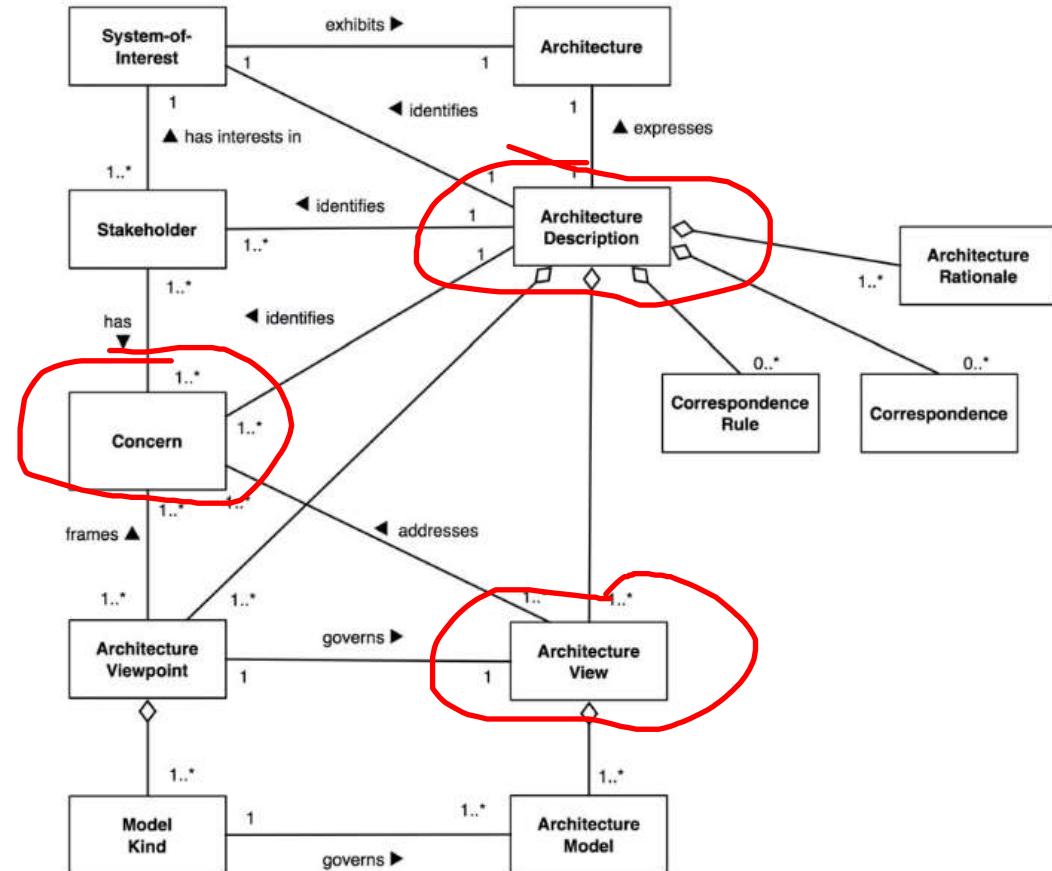
Metamodel of Architectural Assets



IEEE 42010 standard for Systems and Software Engineering: Architecture description - metamodel

Standard for describing the architecture of a software-intensive system.

Builds upon IEEE **1471-2000** - recommended practice for architectural description for software-intensive systems



<http://eam-initiative.org/pages/10kg0yek2601n/ISO-IEC-IEEE-42010-Systems-and-software-engineering-Architecture-description>

Rationale - Architectural Decision

- Records the **significant decisions** made in shaping the overall architecture, and describes the **rationale** used in making the decision, the **options** considered, and the **implications** of the decision
- Focus is on **high impact, high priority** areas that are in strong alignment with the business strategy

Architecture Decision – Adopting active passive database clustered architecture

Issue	The data tier should not be a single point of failure for the system.
Architecture Decision	Adopting active-passive database clustered architecture
Assumptions	Time (in minutes) to failover to the passive database is acceptable. The storage space is accessible from both database instances
Alternatives	1. Data replication
Justification	Data replication might still result in potential data loss.



Architectural Viewpoint Catalogue

- Architects describe the architecture by using a set of viewpoints that have been selected from a **viewpoint catalogue** and tailored to their needs.
- A **viewpoint** is a specification of the conventions for constructing and using a view
 - Pattern or template for representing one set of concerns relative to an architecture
- A **view** is a representation of a whole system from the perspective of a related set of concerns
 - Actual representation of a particular system
 - List of descriptions, matrix, diagrams

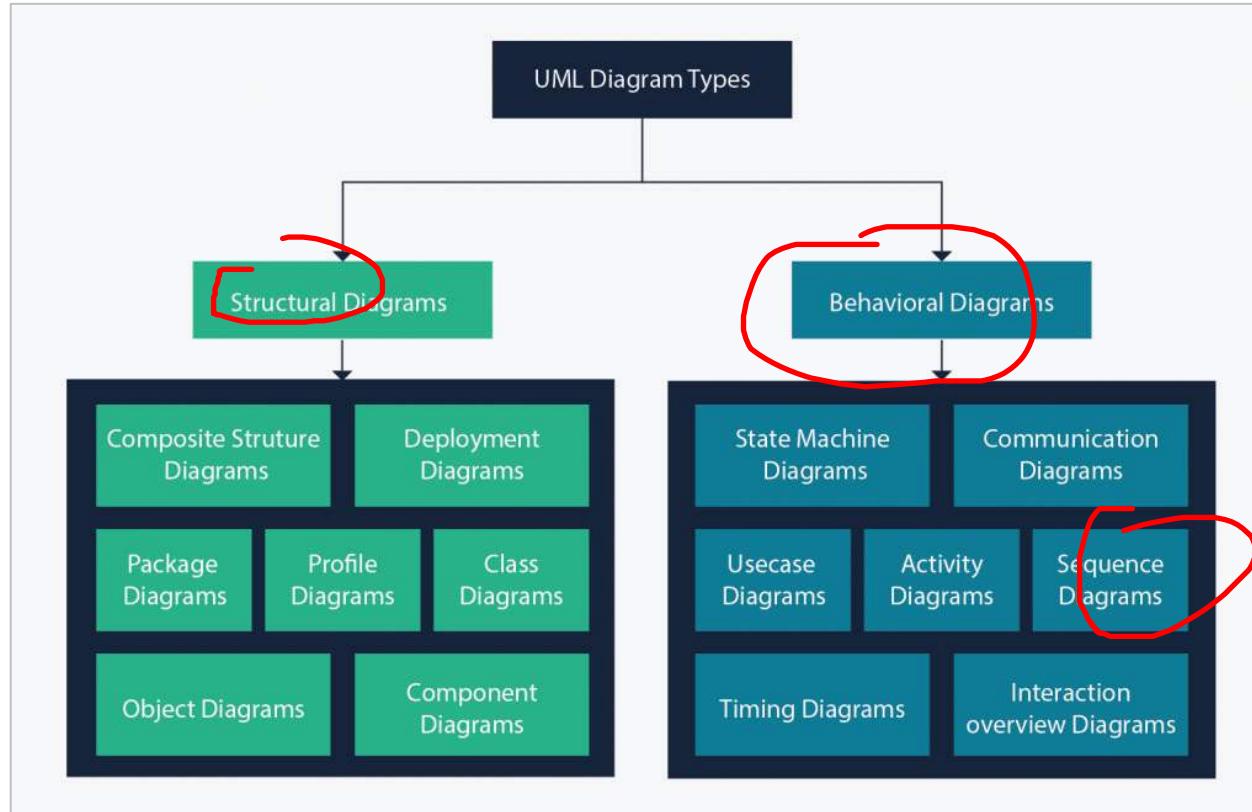


Architectural Modeling Methods

- Hand-drawn sketches
 - Whiteboard, paper and pen, ...
- Drawing Tools
 - Microsoft Visio, OmniGraffle, ...
- Modeling Tools
 - Architecture Description Languages - Architecture Analysis and Design Language(AADL), ArchiMate, ...
 - Using code - Structurizr
 - UML Diagrams – SparxSystem Enterprise Architect, WhiteStarUML, ...



UML - taxonomy of diagram types



The Unified Modeling Language (UML) is a general-purpose, developmental, modelling language for software engineering, providing a standard way to visualize the design of a system.

OMG: <https://www.omg.org/cgi-bin/doc?formal/2010-05-05>

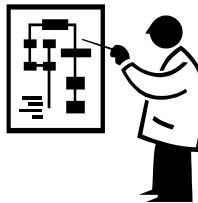
4+1 architectural view model -

for describing the architecture of software-intensive systems, based on the use of multiple, concurrent views"



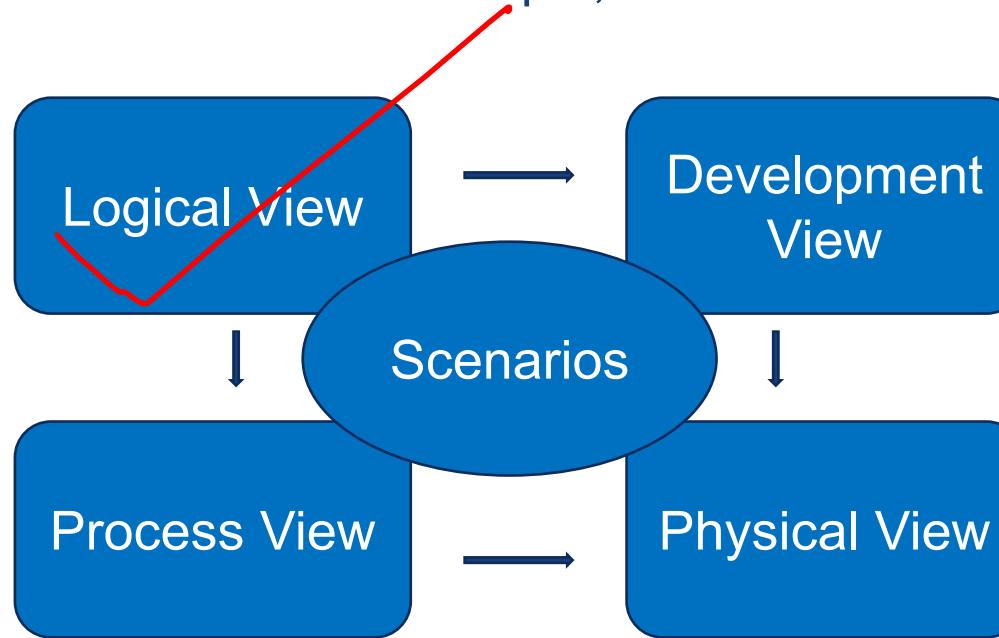
End User

- Functionality



Integrator

- Performance ✓
- Security
- Throughput ✓



Programmer

- Coding ✓
- Testing



System Engineer

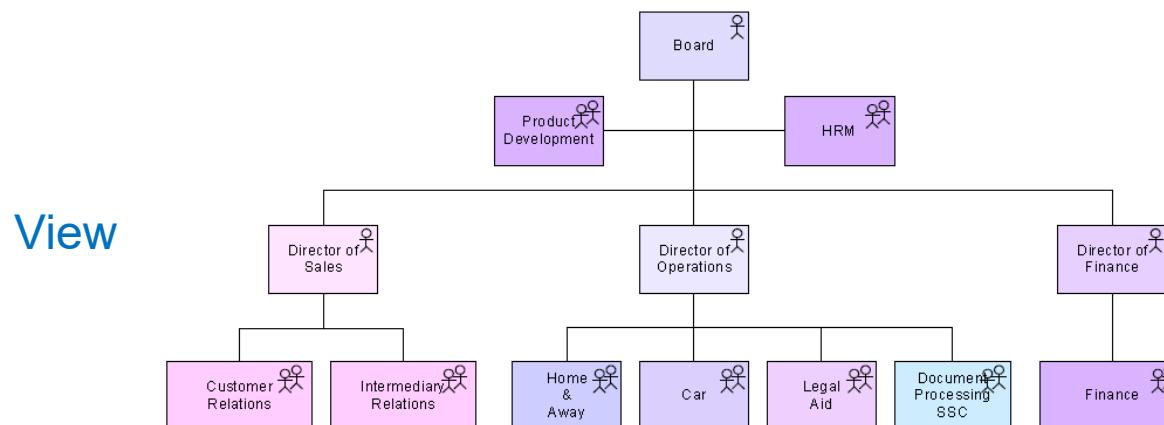
- System Topology
- Delivery
- Installation
- Telecommunication

Refer to Appendix C for examples.



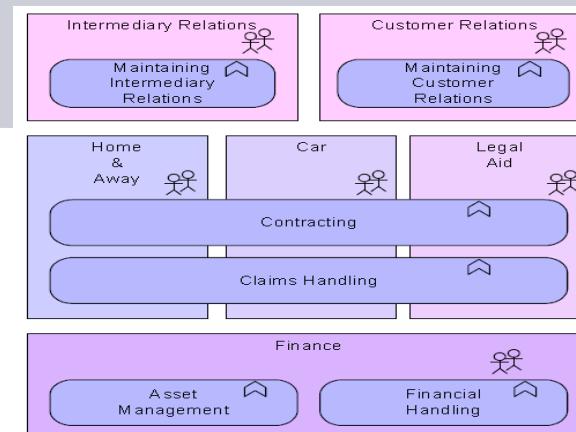
Viewpoint and View

Viewpoint Name	Organization Viewpoint
Stakeholders	Customer Representatives (end users, management) Managers
Concerns	Structure of the organization, department or other organizational entity. Shows its competencies, responsibilities and authority
Modeling techniques	Block diagram



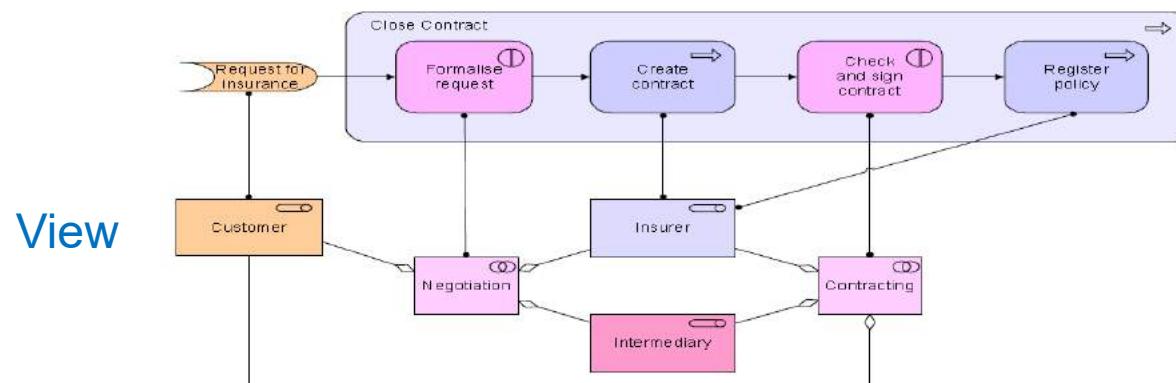


Viewpoint and View

Viewpoint Name	Business Function Viewpoint
Stakeholders	Customer Representatives (end users, management) Managers Designers of constituent parts (business architect)
Concerns	Main business functions of the system with relations to the flows of information between them. Identification of essential services and competencies.
Modeling techniques	Block diagram  <p>View</p>

Viewpoint and View

Viewpoint Name	Business Process Viewpoint
Stakeholders	Customer Representatives (end users, management) Managers Designers of constituent parts (business architect) Quality attribute specialists
Concerns	Show the high level structure and composition of one or more business processes. Emphasis on completeness and consistency.
Modeling techniques	Block diagram, BPMN, BPEL, UML Activity diagram

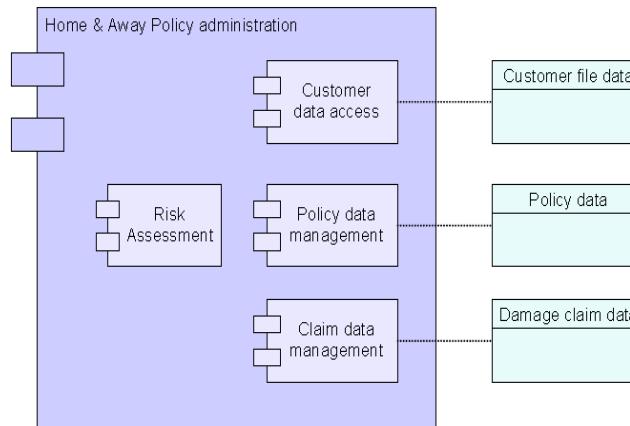




Viewpoint and View

Viewpoint Name	Application Structure Viewpoint
Stakeholders	Customer Representatives (customer architect) Designers of constituent parts (application architect) Quality attribute specialists Developers, Testers and integrators, Maintainers
Concerns	Structure of application, focus on the completeness and consistency
Modeling techniques	Layering, UML package, component diagrams

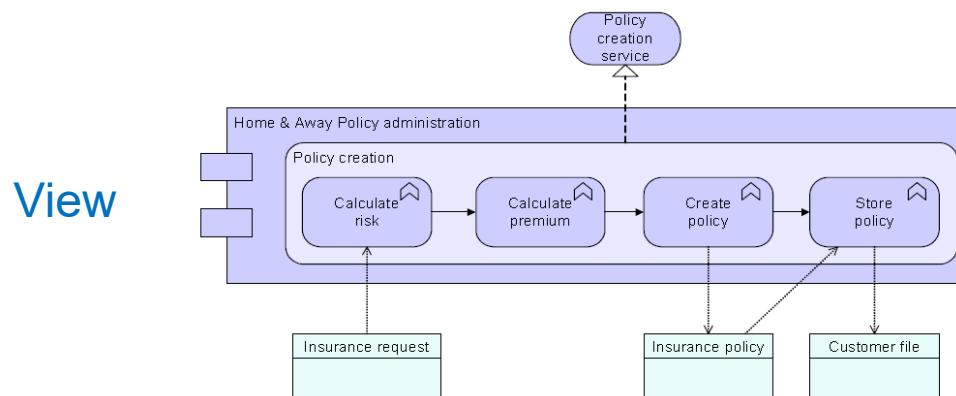
View





Viewpoint and View

Viewpoint Name	Application Behaviour Viewpoint
Stakeholders	Customer Representatives (customer architect) Designers of constituent parts (application architect) Quality attribute specialists Developers, Testers and integrators, Maintainers
Concerns	Internal behavior of the application or component as it realizes one or more requirements. Showing the structure, relations and dependencies of components
Modeling techniques	UML Package, sequence, collaboration, state diagrams

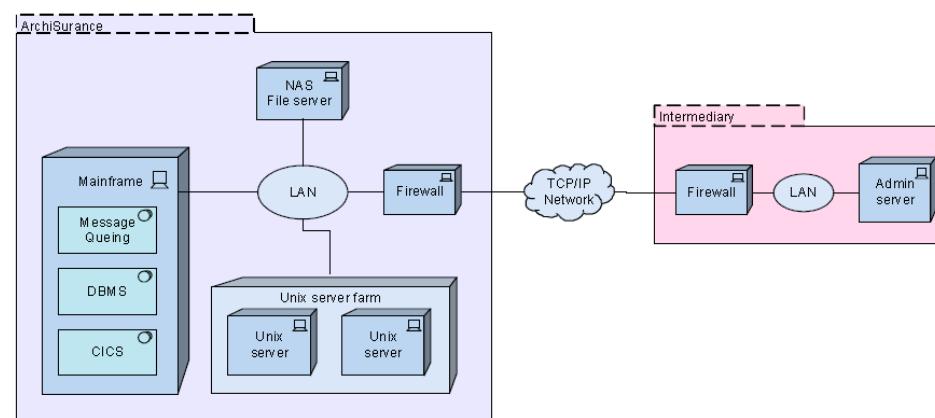




Viewpoint and View

Viewpoint Name	Infrastructure Viewpoint
Stakeholders	Customer Representatives (customer architect) Designers of constituent parts (infrastructure architects) Quality attribute specialists Maintainers
Concerns	Comprises of the hardware and software infrastructure upon which the application depends. Focus on the security, stability, dependencies and costs of infrastructure
Modeling techniques	Network diagram, UML package diagram, Block diagram

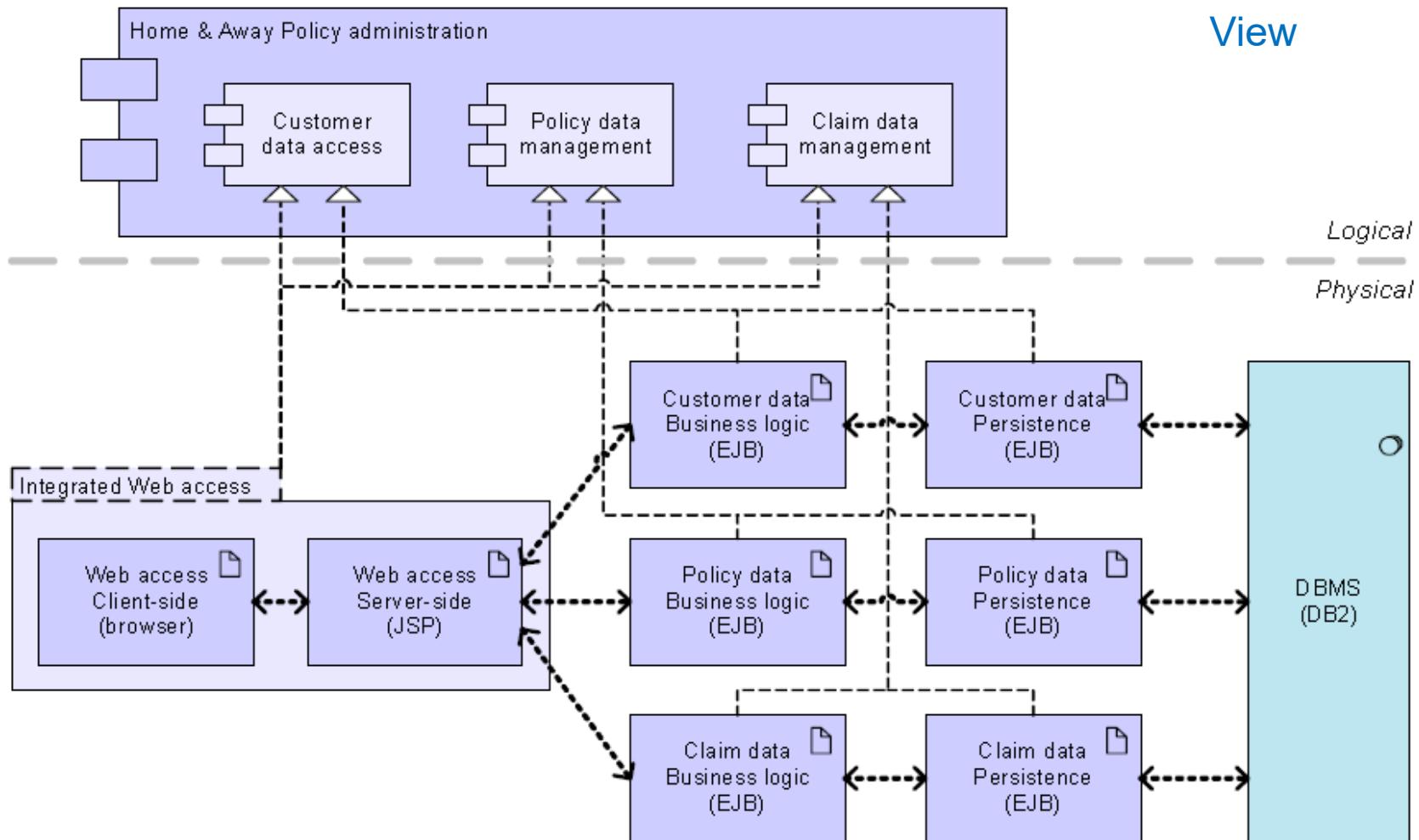
View



Viewpoint and View

Viewpoint Name	Deployment Viewpoint
Stakeholders	Customer Representatives (customer architect) Designers of constituent parts (application architects) Quality attribute specialists Developers, Testers and integrators, Maintainers
Concerns	Shows the mapping of the logical components into physical artifacts. Focus on the security, performance and other dependencies
Modeling techniques	UML package, deployment diagrams

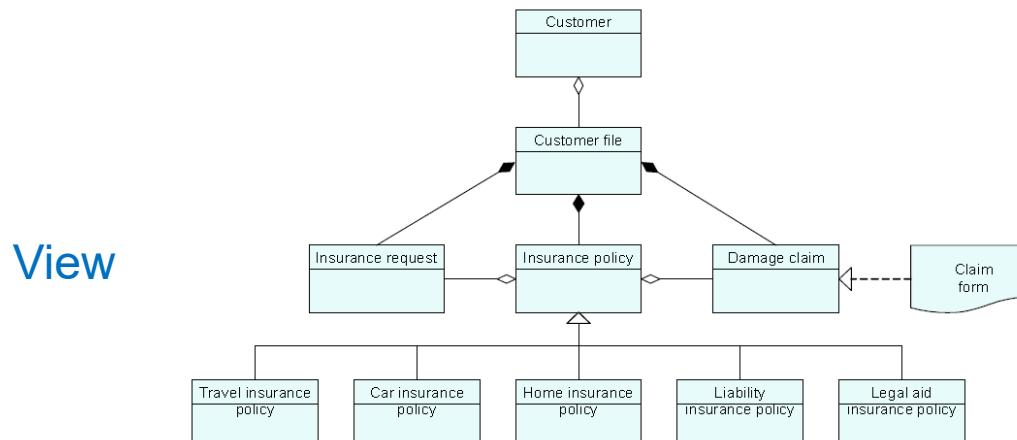
Viewpoint and View





Viewpoint and View.

Viewpoint Name	Information Structure Viewpoint
Stakeholders	Customer Representatives (end users, customer architect) Designers of constituent parts (data architect) Quality attribute specialists Developers, Testers and integrators, Maintainers
Concerns	Structure and dependencies of information and data used
Modeling techniques	UML class diagram, entity relationship diagrams





Summary.

- The architectural metamodel consists of **reference patterns**, **reference architectures** and **architectural descriptions** that describe the architecture
- The reference patterns are **architectural styles**, analysis patterns, design patterns and idioms
- Determining the **right style and architecture** (which depends on the architect's knowledge and experience) is critical to achieving the required qualities in the system



APPENDIX A

(Idioms)

- Java Idioms - common code-based solutions to Java coding problems. E.g., includes naming conventions, source code formatting, memory management, etc.)
 - Java naming and coding conventions

Packages	Prefix of a unique package name is always written in all-lowercase ASCII letters and should be one of the top-level domain names,	com.apple.quicktime.v2 edu.cmu.cs.bovik.cheese
Classes	should be nouns, in mixed case with the first letter of each internal word capitalized.	class Raster; class ImageSprite;
Interfaces	Interface names should be capitalized like class names.	Interface RasterDelegate; interface Storing;



- Check Don't Catch - more efficient in Java to check to see if an exceptional condition exists than to let an exception be raised and have to catch it afterwards

```
try {  
    for (int i=0; ; i++) { // do something with someArray here  
}  
} catch (ArrayIndexOutOfBoundsException ex) {  
    // ignore the exception  
}
```

can be more efficiently written as

```
for (int i=0; i < someArray.length; i++) {  
    // do something with someArray here  
}  
// In cases when upperbound is invariant, better to calculate once
```



Idioms

- Throwaway Object - Don't always need to keep the classes we create.
- Don't create garbage

```
new MyClass (arg1, arg2).go();
```

```
String data[100] = { /* 100 strings */ };
String create_report(String[] data) {
    String result = "";
    for(int i = 0; i < data.length, i++) { result += data[i]; }
    return result;
} // result += data[i] creates a new String instance every time the code is executed.
```



```
String data[100] = { /* 100 strings */ };
String create_report(String[] data) {
    StringBuffer result = new StringBuffer (totalNumOfChars(data));
    for(int i = 0; i < data.length, i++) { result.append(data[i]); }
    return new result.toString();
} // intermediate objects can be avoided by using StringBuffer
```



- Java naming and coding conventions

Methods	Methods should be verbs, in mixed case with the first letter lowercase, with the first letter of each internal word capitalized.	run(); runFast(); getBackground();
Variables	All instance, class, and class constants are in mixed case with a lowercase first letter. Variable names should not start with underscore _ or dollar sign \$ characters. Variable names should be short yet meaningful.	float myWidth;
Constants	should be all uppercase with words separated by underscores ("_").	static final int MIN_WIDTH = 4; static final int MAX_WIDTH = 999;

- Java naming and coding conventions

Line Length	Avoid lines longer than 80 characters, since they're not handled well by many terminals and tools
Wrapping Lines	Break after a comma, break before an operator, align the new line with the beginning of the expression at the same level on the previous line.
Simple Statement	Each line should contain at most one statement.
Compound Statements	The enclosed statements should be indented one more level than the compound statement
Blank Lines	<p>Two blank lines should always be used in the following circumstances:</p> <p style="padding-left: 2em;">Between sections of a source file. Between class and interface definitions</p> <p>One blank line should always be used in the following circumstances:</p> <p style="padding-left: 2em;">Between methods. Between the local variables in a method and its first statement. Before a block</p>

- Java naming and coding conventions

Referring to Class Variables and Methods	Avoid using an object to access a class (static) variable or method.	AClass.classMethod(); //OK anObject.classMethod(); //AVOID!
Constants	Numerical constants (literals) should not be coded directly except for -1, 0, and 1, which can appear in a for loop as counter values.	for (int size=0; size < 15; size++) {} for (int size=0; size < totalsize; size ++)
Variable Assignments	Avoid assigning several variables to the same value in a single statement	objectA.size = objectB.size = 10;
Parentheses	Use parentheses liberally in expressions involving mixed operators to avoid operator precedence problems	if (a == b && c == d) // AVOID! if ((a == b) && (c == d)) // RIGHT



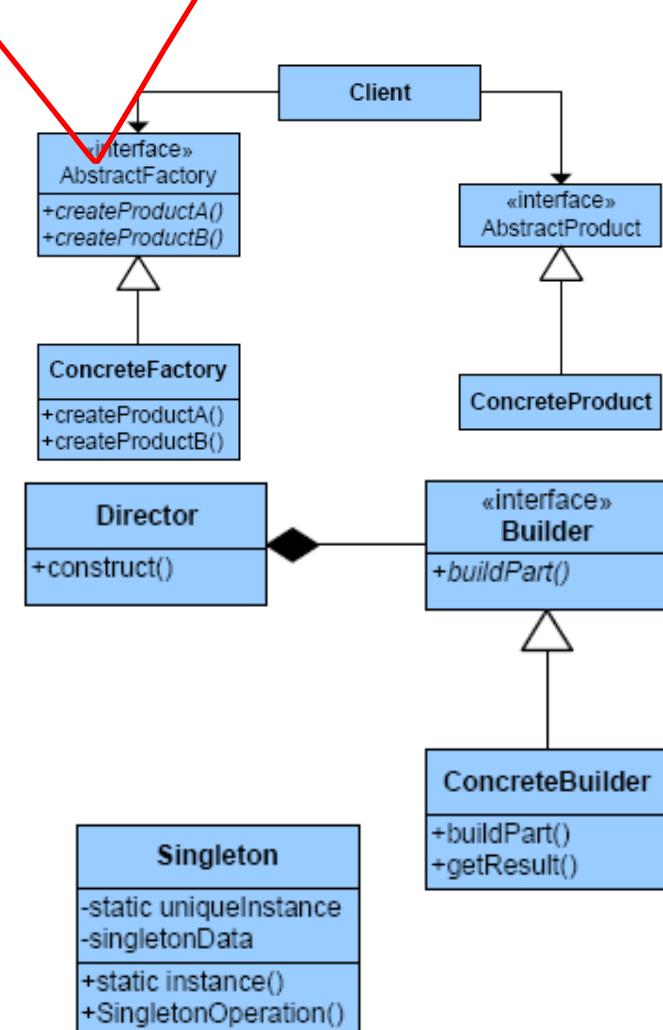
APPENDIX B

(Design Patterns)



Gang of Four Design Pattern

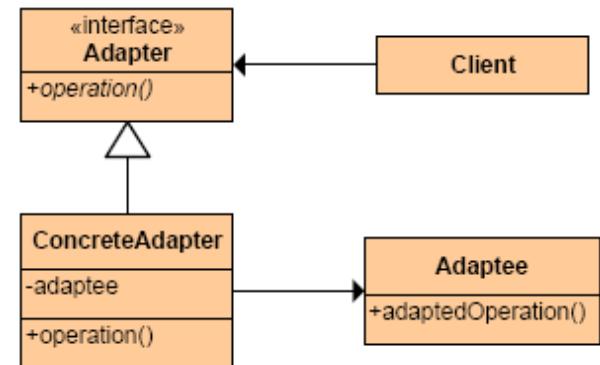
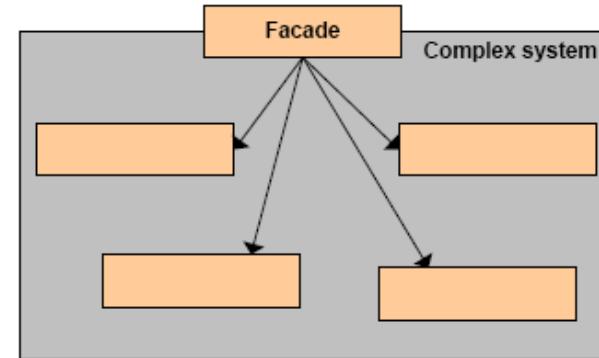
- Creational - For class instantiation
 - Abstract Factory - Provides an interface for creating families of related or dependent objects without specifying their concrete class.
 - Builder - Separate the construction of a complex object from its representing so that the same construction process can create different representations.
 - Singleton - Ensure a class only has one instance and provide a global point of access to it.
 - Others - Factory Method and Prototype





Gang of Four Design Pattern

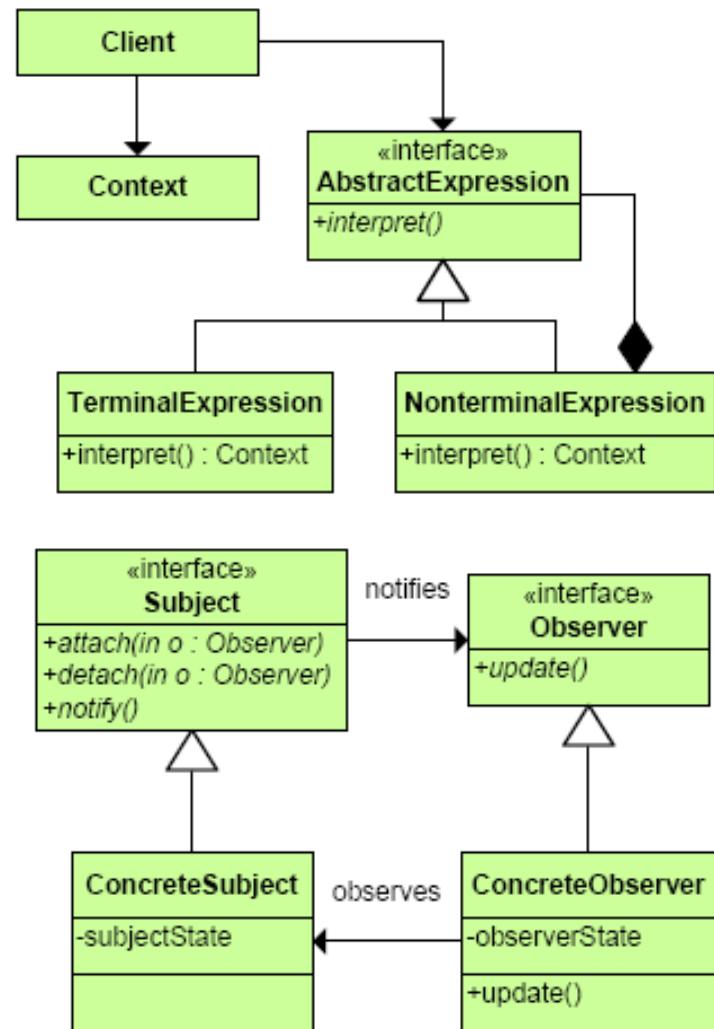
- Structural - Class and Object composition
 - Façade - Provide a unified interface to a set of interfaces in a subsystem. Defines a high level interface that makes the subsystem easier to use.
 - Adapter - Convert the interface of a class into another interface clients expect. Lets classes work together that couldn't otherwise because of incompatible interfaces.
 - Others – Bridge, Composite, Decorator, Flyweight, Proxy





Gang of Four Design Pattern

- Behaviour – Objects communication
 - Iterator - Provide a way to access the elements of an aggregate object sequentially without exposing its underlying representation.
 - Observer - Define a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically.
 - Others - Chain of Responsibility, Command, Interpreter, Mediator, Memento, State, Strategy, Template Method, Visitor



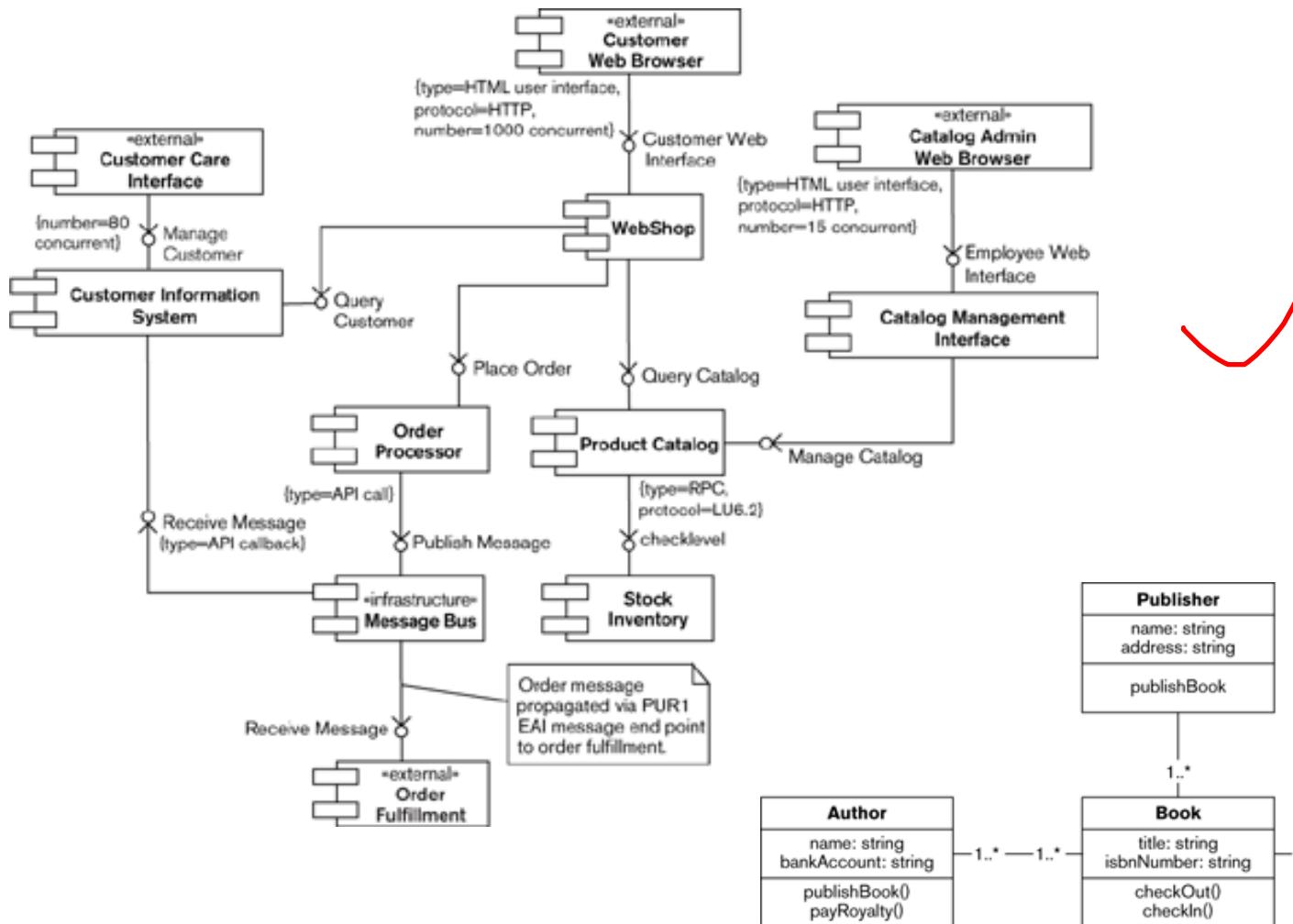


APPENDIX C

(4+1 Views)



4+1 View - Logical View



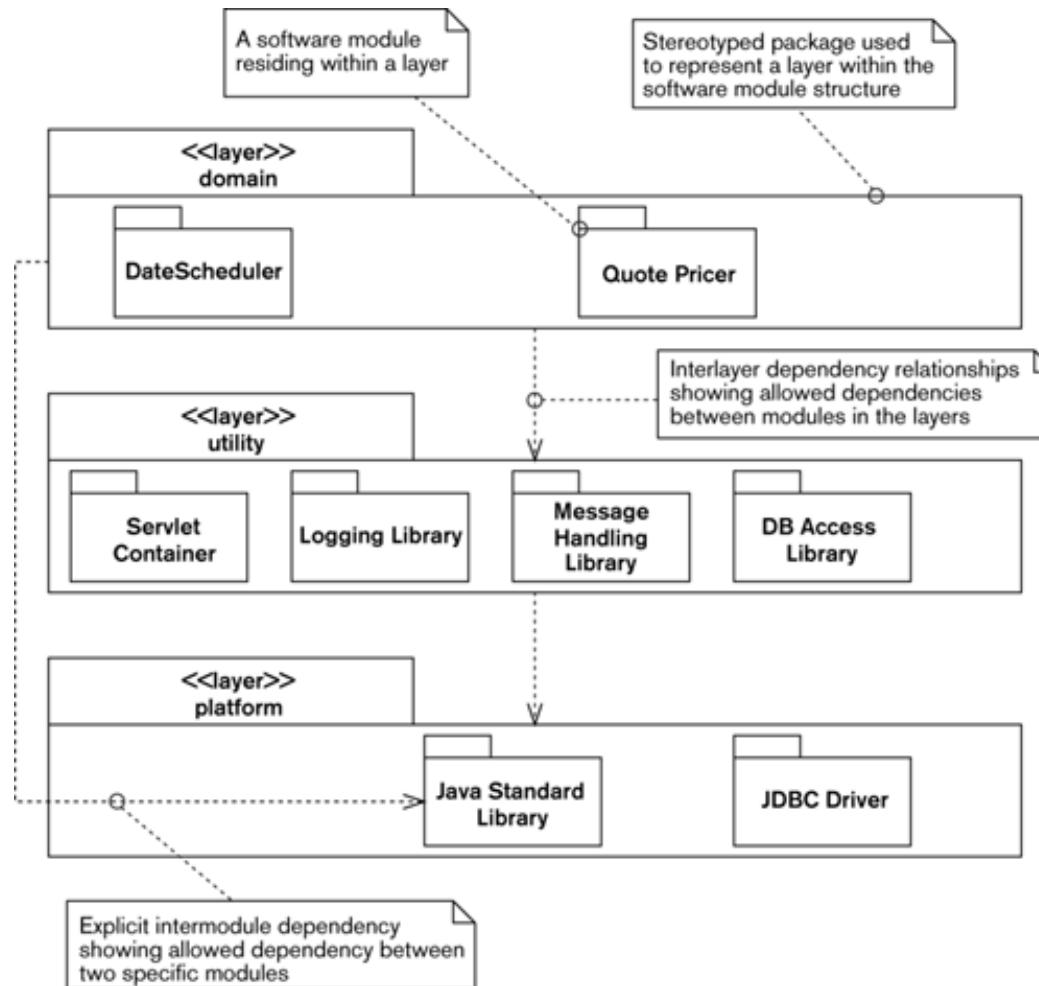


4+1 View - Process View



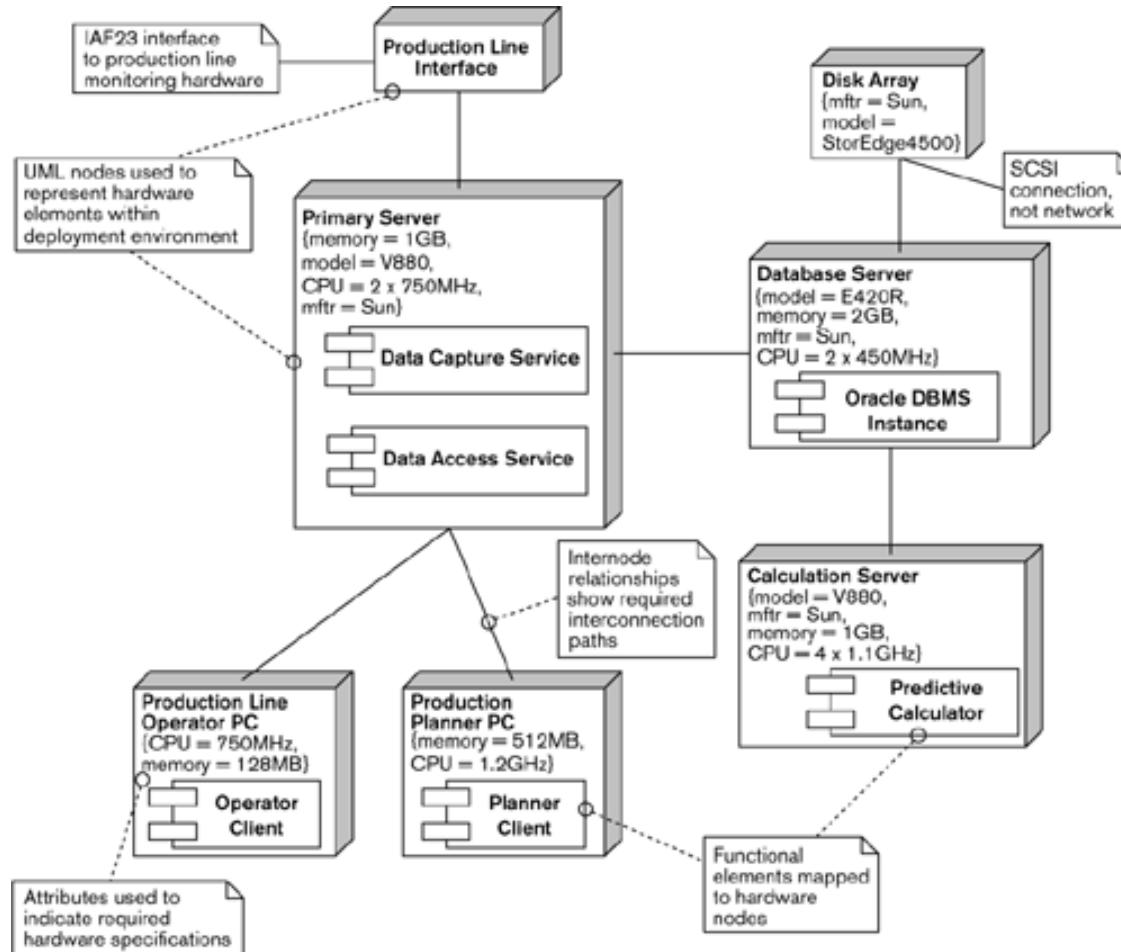


4+1 View - Development View





4+1 View - Physical View





ARCHITECTING SOFTWARE SOLUTIONS

ARCHITECT THE SOLUTION ARCHITECTURE (METHOD)

Instructor: Heng Boon Kui

Email: boonkui.heng@nus.edu.sg

Total slides: 84

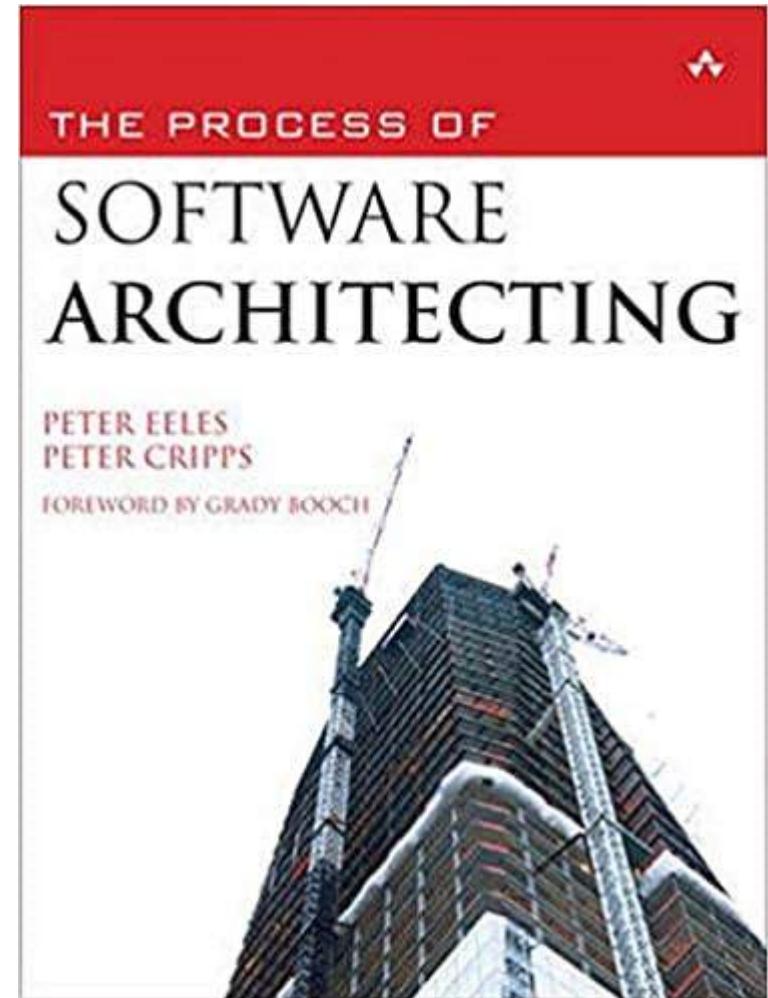


Objectives

- Understand the fundamentals of a method for architecting solutions
- Understand the requirements definition process
- Define the logical and physical solution architectures

- Fundamentals of the Method
- Defining Requirements
- Defining the Logical Solution Architecture
- Defining the Physical Solution Architecture

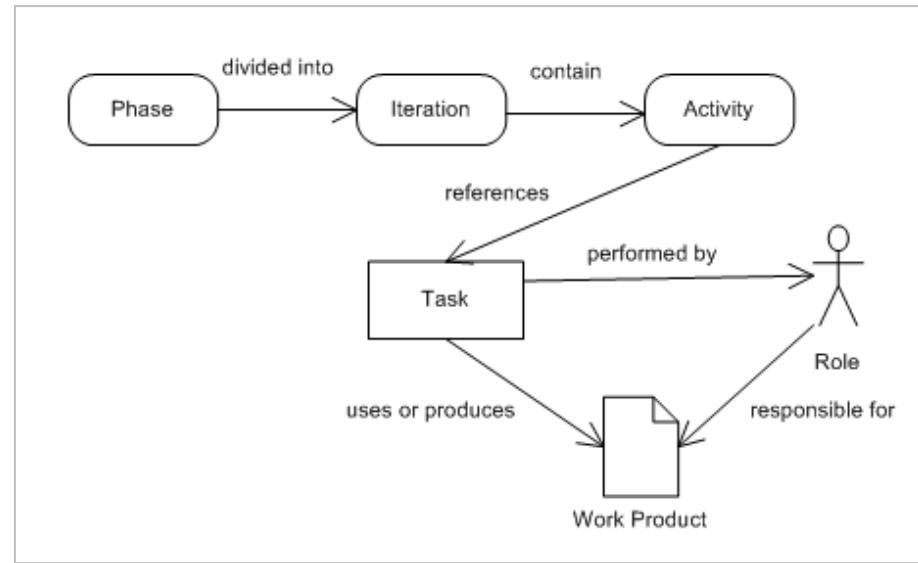
- Deep examination of the **method of architecting**
 - Not focusing on the concepts of software architecture
- Covering **requirements**, logical & physical **architecture** and role of **architect**
 - Compatible with variants of **Unified Process**
- Drawing from **experiences** at IBM, Rational Software and the wider software community





Fundamentals of the Method

- **When** : Phases and iterations
- **Who** : Roles defines the responsibilities of an individual or a set of individuals working together as a team within the context of a software development organization
- **What** : Work product is the information or entity that is produced and/or used
- **How** : Activity represents a grouping of tasks. Task is a unit of work that provides a meaningful result in the context of the project



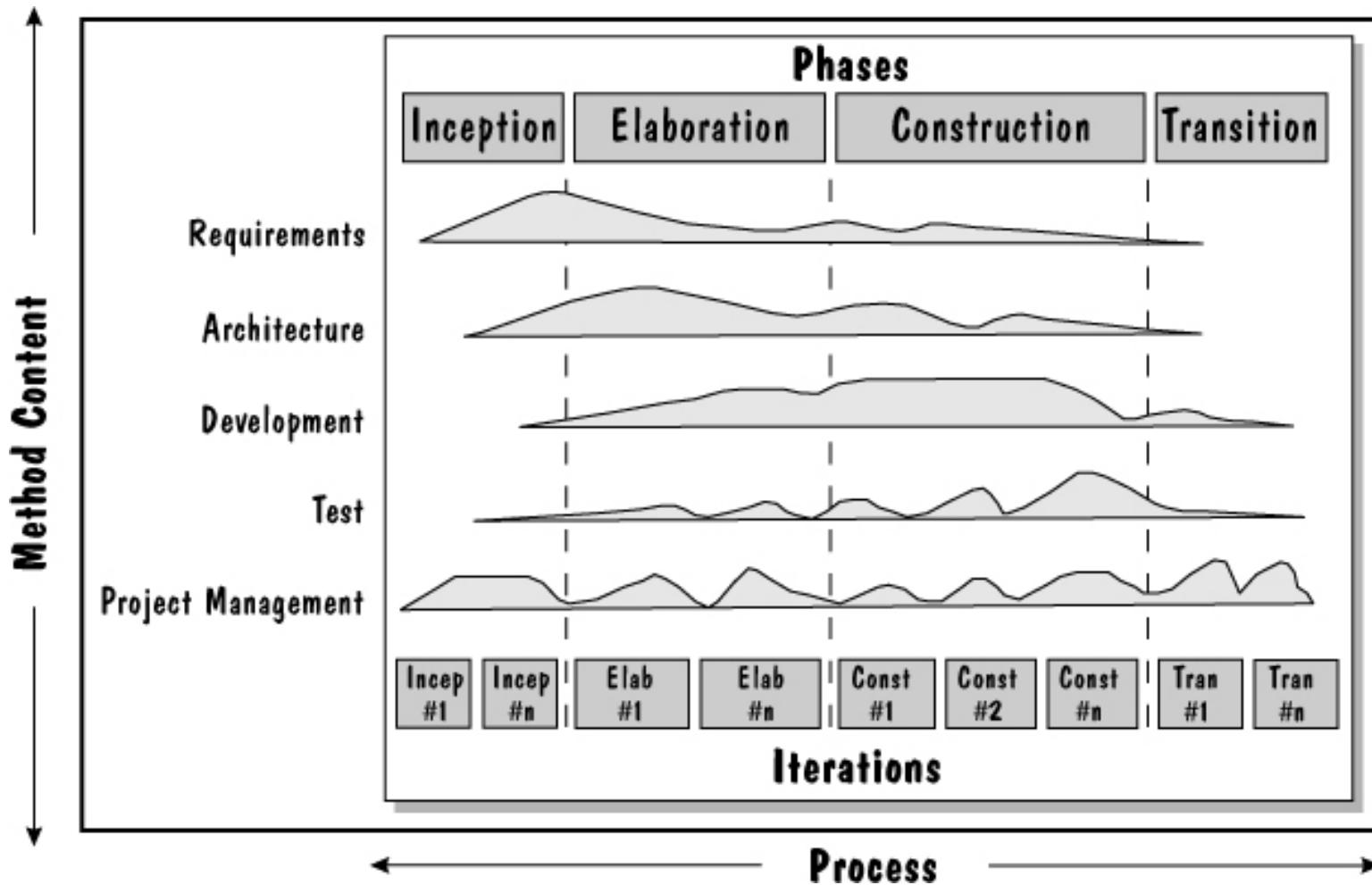
Unified Software Development Process or Unified Process is a popular framework for software solutions

Different flavours:

- Rational Unified Process (RUP), from IBM
- Open Unified Process (OpenUP), the Eclipse Process Framework version
- Agile Unified Process (AUP)
- Enterprise Unified Process (EUP), an extension of RUP

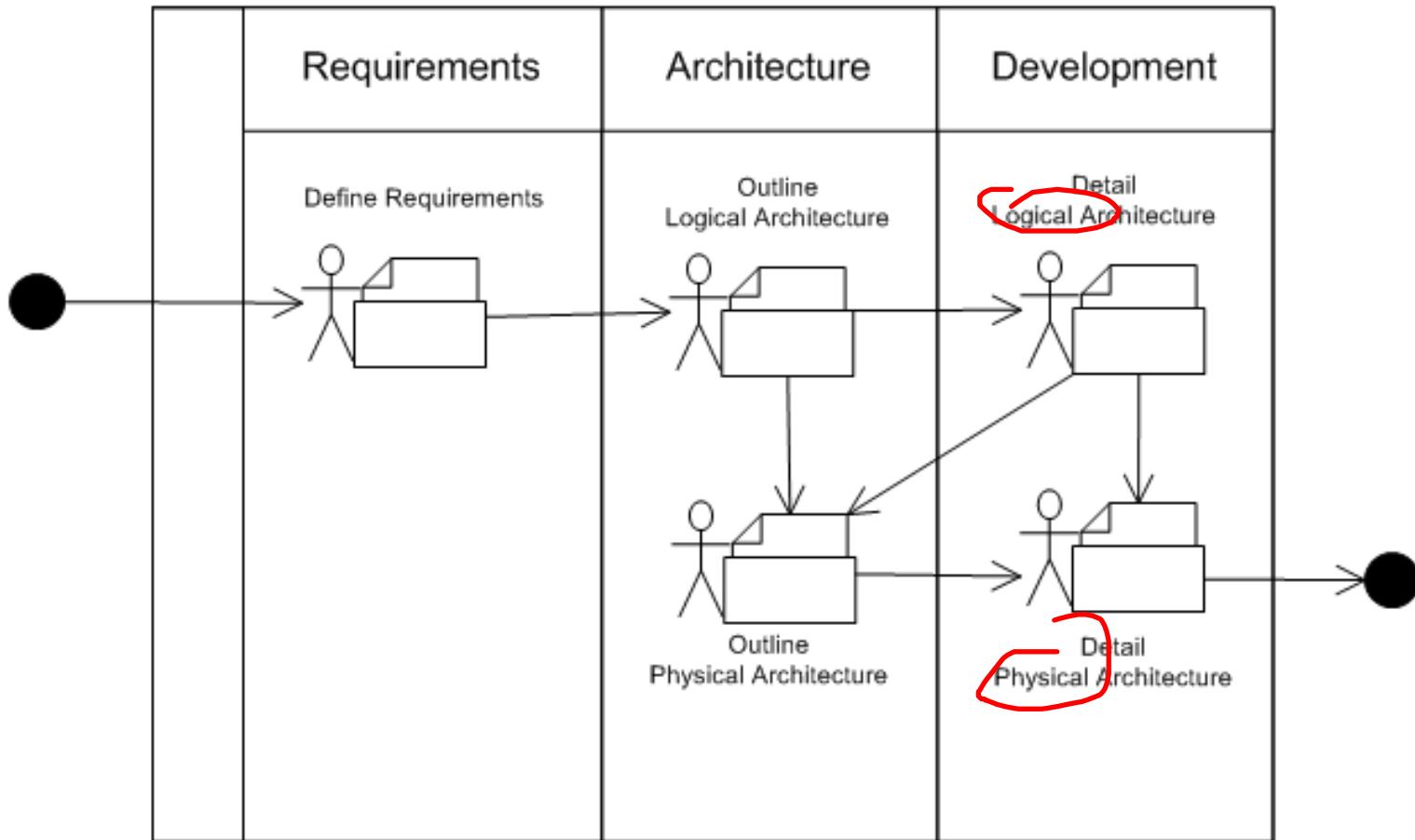


Fundamentals of the Method





Summary of Activities.



- Fundamentals of the Method
- **Defining Requirements**
- Defining the Logical Solution Architecture
- Defining the Physical Solution Architecture

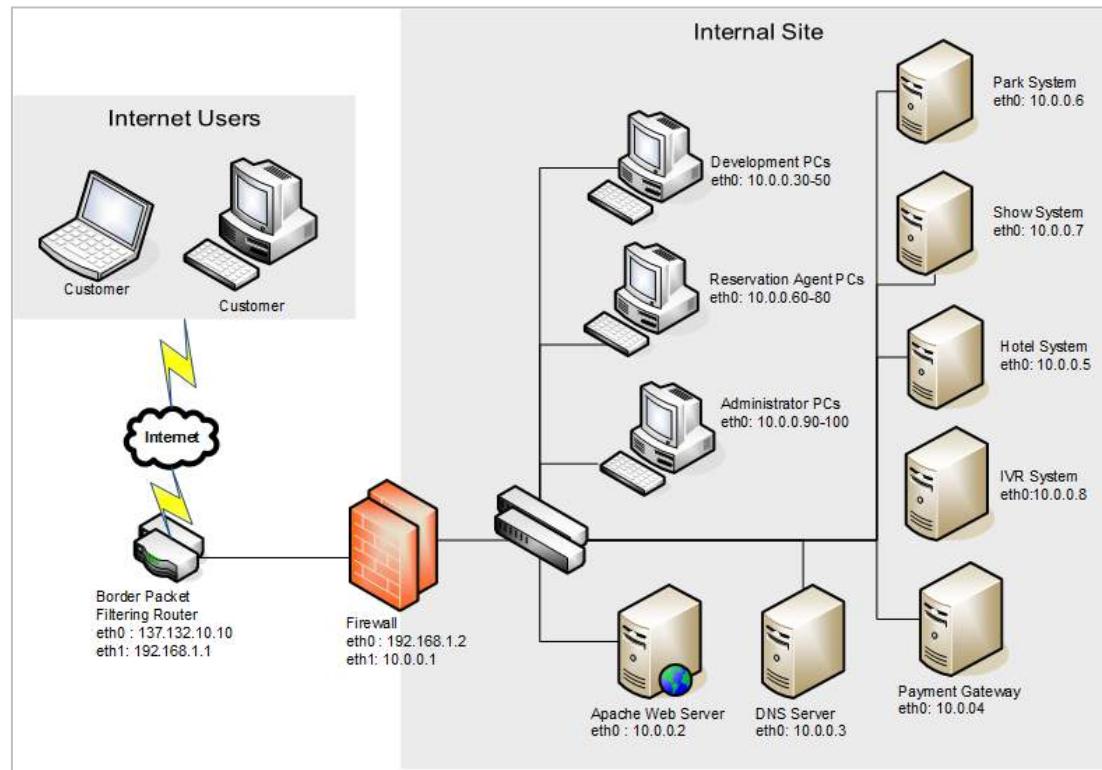


Example Scenario: IROne

Self Service Resort System

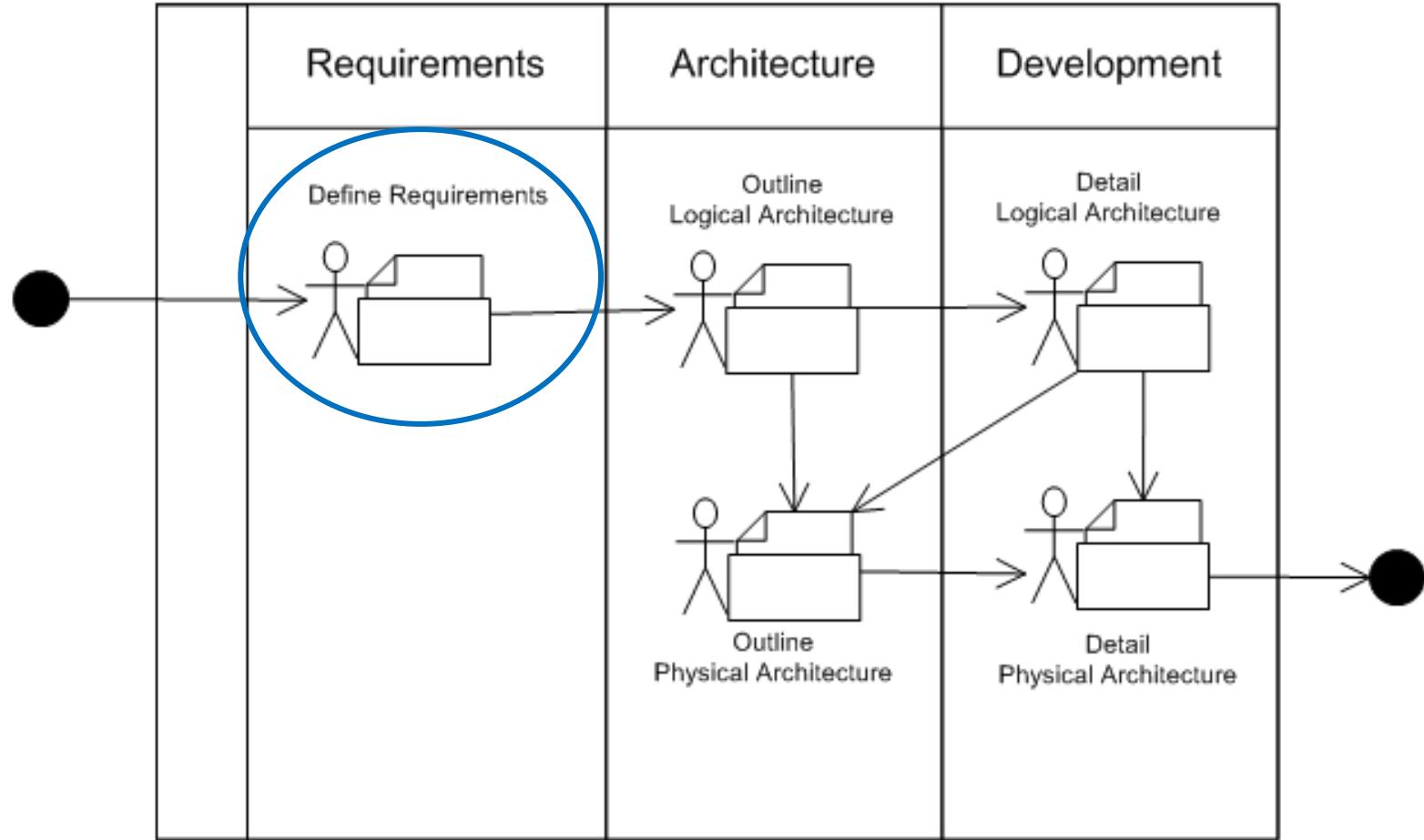
This project, by company SSOne, will automate the reservation system through a self-service web channel. The solution will be referred to as '**IROne**'.

Current Environment



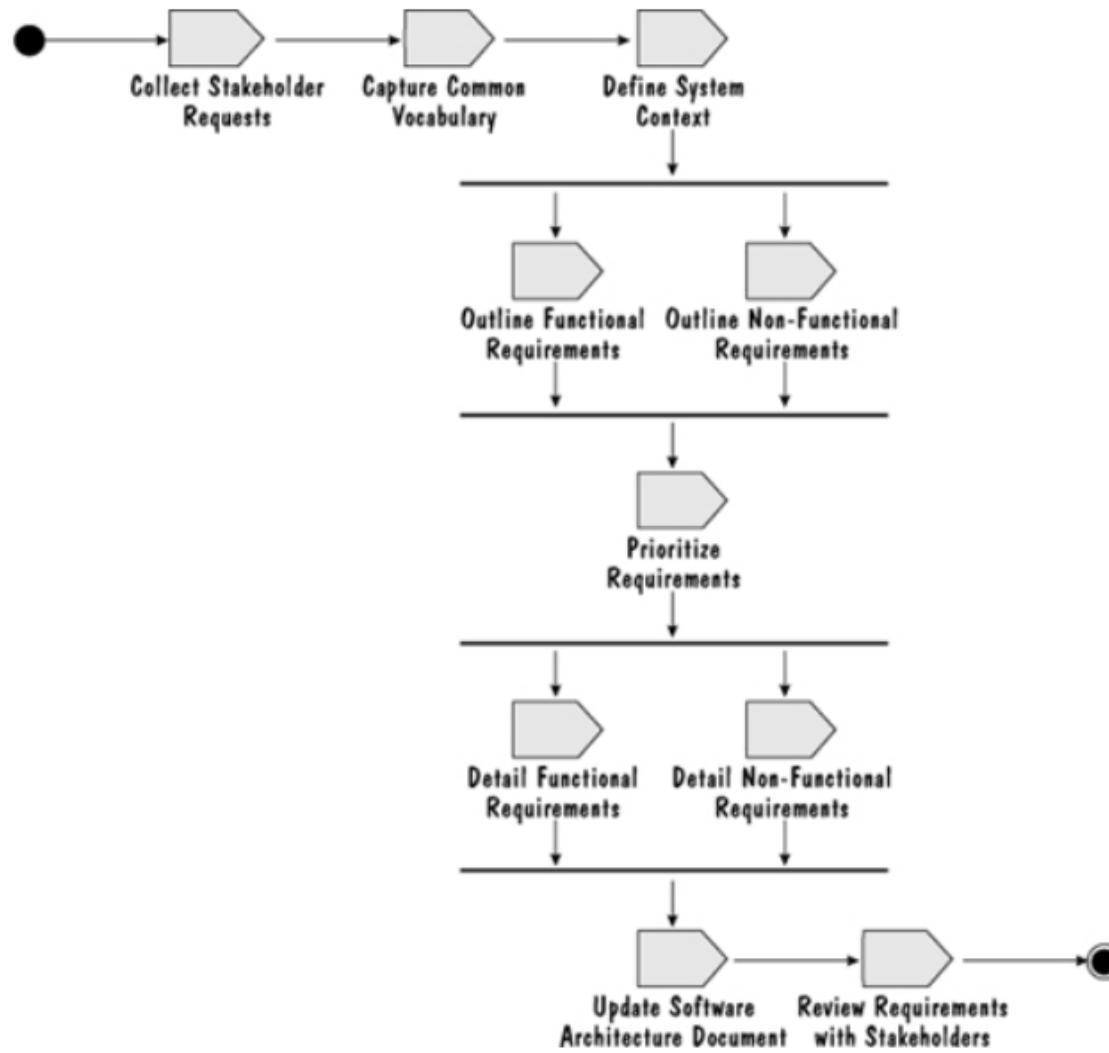


Summary of Activities





Define Requirements Activity – Task Summary





Define Requirements Activity – Task Summary

Task
1. Collect Stakeholder Requests
2. Capture Common Vocabulary
3. Define System Context
4. Outline Functional Requirements
5. Outline Non-Functional Requirements
6. Prioritize Requirements
7. Detail Functional Requirements
8. Detail Non-Functional Requirements
9. Update Solution Architecture Document
10. Review Requirements with Stakeholders





Task 1 – Collect Stakeholder Requests

Roles	Task Inputs	Task Outputs
Stakeholder Business Analyst Solution Architect Infrastructure Architect	Vision Business Process Model Change Request	Stakeholder Requests

No	Steps	Things to note (as an Architect)
1	Identify Stakeholders	Ensure more <u>technical roles</u> (e.g. System Administrator, Owner of external systems) are considered.
2	Collect & prioritize Stakeholder requests	<p>Ensure that the <u>more technical requirements such as qualities and constraints</u> are captured.</p> <p><u>Assist in prioritization</u> by explaining the impact of one request over another to the business such as cost and time to market.</p>



Task 1 – Collect Stakeholder Requests (Identify Stakeholders)

Stakeholders	Stakeholder Description
System Owner	Person who is responsible for the overall system.
Customer	Person who use the system to perform transaction
System Administrator	Person who administer the system and ensures it runs smoothly
Show Operator	Person who is responsible for the show system which is integrated with IROne
Park Operator	Person who is responsible for the park system which is integrated with IROne
Hotel Operator	Person who is responsible for the hotel system which is integrated with IROne
Reservation Agent	Person who handle customer calls and interact with the system
SSOne Architect	Person who is responsible for the enterprise archtitecture of SSOne.



Task 2 – Capture Common Vocabulary

Roles	Task Inputs	Task Outputs
Stakeholder Business Analyst	Vision Stakeholder Requests	Glossary
Solution Architect Infrastructure Architect	Business Entity Model Business Process Model Enterprise Architecture Principles	

No	Steps	Things to note (as an Architect)
1	Identify common terms	Ensure that the definition of technical terms are consistent



Task 2 – Capture Common Vocabulary (Identify common terms)

Terms	Description
Reservation	Represent a booking of a particular amenity or amenities
Amenity	Represent a service to be provided to the customer
Reservation Agent	Represent the call service operator who handle customer calls on reservation
Amenity Operator	Represent the amenity service operator who manage the actual amenities.



Task 3 – Define System Context

Roles	Task Inputs	Task Outputs
Stakeholder Business Analyst Solution Architect	Vision Stakeholder Requests Business Entity Model Business Process Model Enterprise Architecture Principles Existing IT Environment	System Context

No	Steps	Things to note (as an Architect)
1	Identify actors	Ensure external systems to be integrated (e.g. payment gateway) are captured.
2	Identify actor locations	Ensure that the location of actors are defined properly. Introduce new actor if the existing actor is found to be at multiple locations
3	Identify data flows	Ensure that the flows of data between actors (including external systems) and the system itself are considered.



Task 3 – Define System Context (Identify actors)

Actor	Description
Customer	User of the system
System Owner	Owner of the system
Reservation Agent	User of the system who handles reservation on behalf of the customer
System Administrator	User of the system who ensures the smooth running of the system
Hotel System	External system that manages the hotel reservations
Show System	External system that manages the show reservations
Park System	External system that manages the park reservations
Payment System	External system for verification and payment making
Sales Person	Internal user interested on the sales

Task 3 – Define System Context (identify actor locations)

Location	Description	Actors
Show Office	Location where the show system resides	Show System
Hotel Office	Location where the hotel system resides	Hotel System
Park Office	Location where the park system resides	Park System
Central Office	Location where the main office resides	System Owner System Administrator Reservation Agent Payment System



Task 4 – Outline Functional Requirements

Roles	Task Inputs	Task Outputs
Stakeholder Business Analyst Solution Architect	Vision Stakeholder Requests System Context Business Process Model Enterprise Architecture Principles Glossary	Functional Requirements

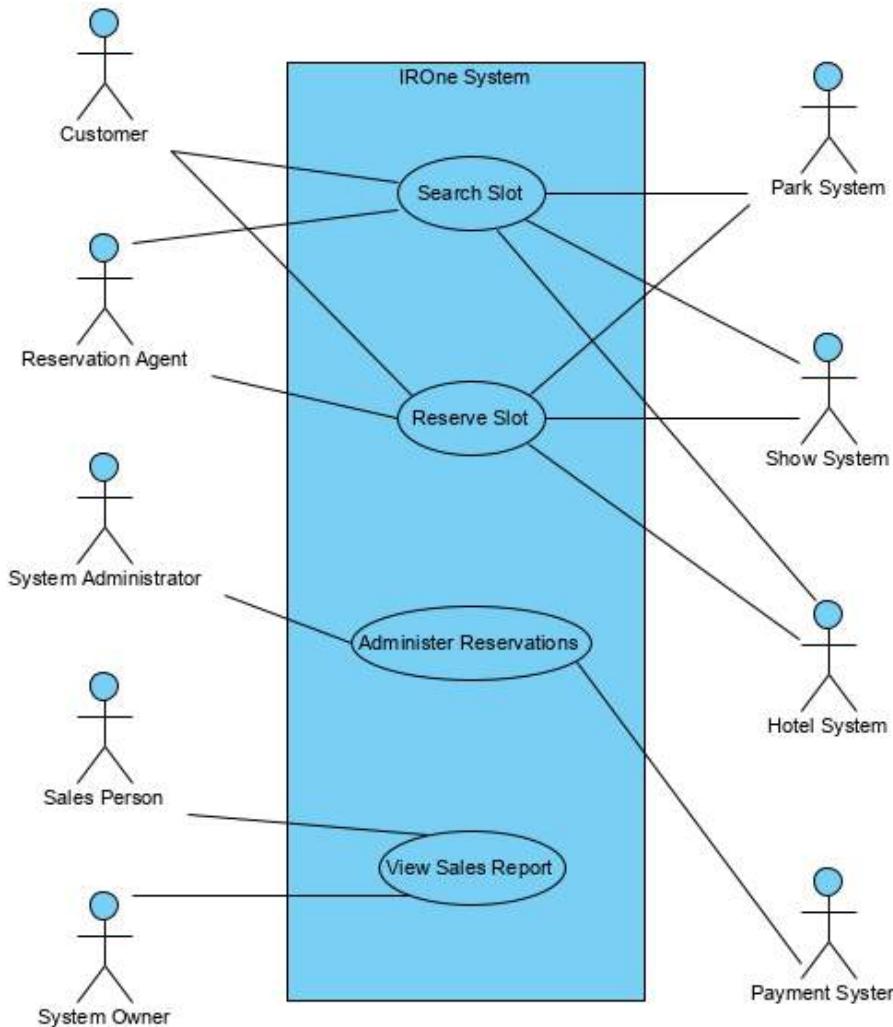
No	Steps	Things to note (as an Architect)
1	Identify and outline functional requirements	Ensure that the functional requirements of the technical users (system operator, system administrator) are considered Ensure the understanding of what is not required for functional requirements



Task 4 – Outline Functional Requirements (Identify and outline functional requirements)

Functional Requirement	Description	Primary Actors involved
Search Slot	Allows the user to search for hotel, park or show slots based on pre-defined criteria	Customer, Reservation Agent
Reserve Slot	Allows the user to reserve for hotel, park or show slots	Customer, Reservation Agent
Administer Reservations	Allows the user to view, change and modify existing reservations	System Administrator
View Sales Report	Allows the user to view the current sales report	Sales Person

Task 4 – Outline Functional Requirements (identify and capture functional requirements)



System Context



Task 5 – Outline Non-Functional Requirements

Roles	Task Inputs	Task Outputs
Stakeholder Business Analyst Solution Architect Infrastructure Architect	Vision Stakeholder Requests System Context Business Rules Enterprise Architecture Principles	Non-Functional Requirements

No	Steps	Things to note (as an Architect)
1	Identify and outline non-functional requirements	<p>Ensure that the appropriate non-functional requirements are discussed.</p> <p>Ensure that the non-functional requirements are clear and concise</p> <p>Ensure the understand of what is not required for non functional requirements</p>

Task 5 – Outline Non-Functional Requirements (identify and outline non-functional requirements)

Non-Functional Requirement	Description
Accessibility ✓	The system will support people with visual, hearing, motor, or cognitive impairments
Availability	The system will be 99.9% available for search functions and 99% for the rest of the functionalities
Performance	The system response time will be below 3 seconds for search functions and below 5 seconds for the rest of the functions.
Scalability ✓	The system will support <u>1000 concurrent users</u>
Legacy Integration	The system will integrate to the external systems through Web Service



Task 6 – Prioritize Requirements

Roles	Task Inputs	Task Outputs
Solution Architect	Vision	Prioritized Requirements
Business Analyst	Stakeholder Requests	
Stakeholder	Functional Requirements	
Project Manager	Non-Functional Requirements	
	RAID Log	

No	Steps	Things to note (as an Architect)
1	Identify architectural significance requirements	Ensure the tradeoffs are articulated during prioritization.

Task 6 – Prioritize Requirements (identify architecturally significant requirements)

- Associated with some critical functionality of the system, without which you do not have a viable system. Use the 80/20 rule.
- Associated with some critical quality of the system, such as performance, without which you do not have a viable system.
- Associated with a critical constraint on the solution, such as the need to integrate with a specific external system.
- The element incurs a particular technical or architectural risk.



Task 7 – Detail Functional Requirements

Roles	Task Inputs	Task Outputs
Stakeholder Business Analyst Solution Architect	Functional Requirements Prioritized Requirements Stakeholder Requests System Context Glossary	Detailed Functional Requirements

No	Steps	Things to note (as an architect)
1	Detail use cases	<p>Consider the architecturally significant use cases</p> <p>Ensure that alternative flows are considered</p> <p>Assist with the definition of the data items</p> <p>Assist in identifying and definition of the system wide functionality, e.g., Online Help</p>

Task 7 – Capture Functional Requirements (detailed use cases)

Reserve Slot Use Case

Description	Allows the user to reserve for hotel, park or show slots
Primary Actors	Customer, Reservation Agent
Secondary Actors	Hotel System, Park System, Show System, Payment System
Main Flow of Events	<ol style="list-style-type: none">1. The user opts to reserve the slot by selecting the Reserve option on the Search Slot screen2. The system displays the Slot Reservation screen3. The user enters his/her personal particulars and payment details4. The system validates the entered details5. The system requests the appropriate amenity system to perform the reservation and stores the reservation details6. The system requests the payment system to effect the payment and stores the payment details7. The system notifies the user on the successful reservation by displaying the details of reservation and payment on the Slot Reservation screen8. The user acknowledges the notification and the system displays the Search Slot screen

Task 7 – Capture Functional Requirements (detailed use cases)

Reserve Slot Use Case (continued)

Alternative Flow of Events	<p>Fail to validate details at step 4: The system prompts the user to enter the details again at step 3</p> <p>Fail to reserve slot at step 5: The system notifies the user of the unsuccessful reservation and displays the Search Slot screen</p> <p>Fail to make payment at step 6: The system notifies the user of the unsuccessful payment and prompts the user to enter the details again at step 3</p>
Pre-conditions	<ul style="list-style-type: none">The user has searched for the slotIROne system has indicated the availability of the slot
Post-conditions	<ul style="list-style-type: none">The reservation is successful and the slot is not available for useThe reservation is unsuccessful and the slot is available for use



Task 8 – Detail Non-Functional Requirements

Roles	Task Inputs	Task Outputs
Stakeholder Business Analyst Solution Architect Infrastructure Architect	Non-Functional Requirements Prioritized Requirements Stakeholder Requests Glossary	Detailed Non-Functional Requirements

No	Steps	Things to note (as an Architect)
1	Detail non-functional scenario	Ensure that the detailing of non-functional requirements are clear and concise



Task 8 – Capture Non-Functional Requirements (details of the non-functional scenario)

Availability Requirements

Stimulus	Bug in coding
Source of the stimulus	Application Programs
Artifact stimulated	IROne System
Environment	Normal Environment, Degraded Environment
Response	<p>System should detect events and do one or more of the following:</p> <ul style="list-style-type: none">• Record it• Notify appropriate parties• Be unavailable for a pre-specified interval• Continue to operate in normal or degraded mode
Response Measure	<ul style="list-style-type: none">• Time interval when the system must be available• Time interval in which system can be in degraded mode

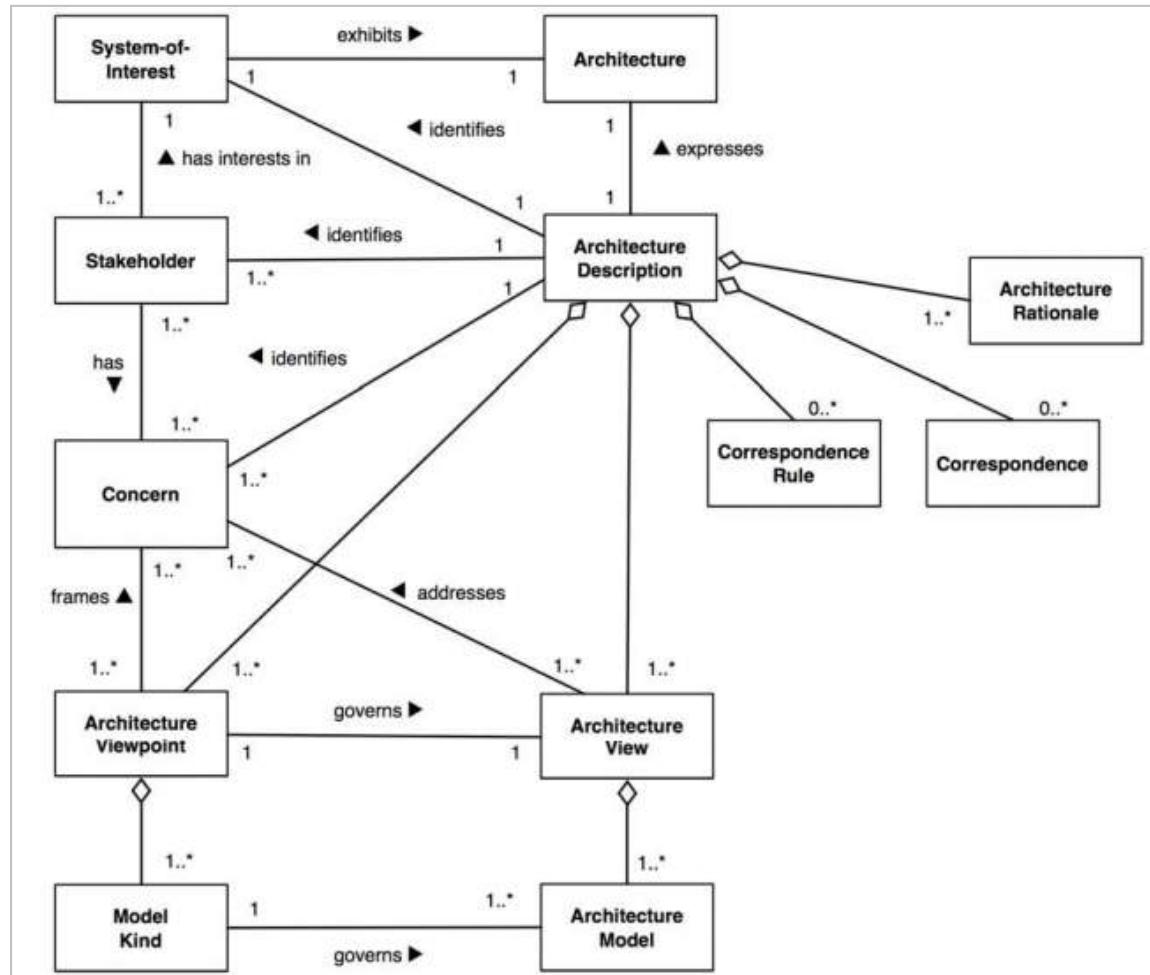


Task 9 – Update Solution Architecture Document

Roles	Task Inputs	Task Outputs
Solution Architect	Business Process Model Non-Functional Requirements Functional Requirements RAID Log	Solution Architecture Document

No	Steps	Things to note (as an Architect)
1	Update document	The Architect is responsible for this task

Task 9 – Update Solution Architecture Document



Conceptual
model of an
architectural
description.

Adapted from ANSI/IEEE Standard 1471-2000 - *Recommended Practice for Architecture Description of Software-Intensive Systems*. Updated and confirmed as ISO/IEC/IEEE standard 42010:2011 in 2017.



Task 9 – Update Solution Architecture Document

Structure of the Document:

1. Introduction

Provide the overview of this document

2. Scope

Provide the list of identified stakeholders, their relationship and requests

3. Architectural Decisions

Describe the requirements and decisions that have some significant impact on the architecture

Task 9 – Update Solution Architecture Document

4. Architectural Mechanisms

Describe the architectural mechanisms that are significant for the architecture. This can include the selection of the architectural assets and the impact to the architecture.

5. Architectural Representation

Describes the solution architecture for the system and how it is represented.

6. Traceability

Describes the linkage from the requirements to the elements of the solution.



Task 10 – Review Requirements with Stakeholders.

Roles	Task Inputs	Task Outputs
Stakeholder	All	Change Request
Business Analyst		Request Log
Solution Architect		Review Record
Infrastructure Architect		
Architect		

No	Steps	Things to note (as an Architect)
1	Review work Products	Ensure that the artefacts are baselined Ensure that any questions of technical nature have been answered

Architecting the Student Health System (Exercise 1)

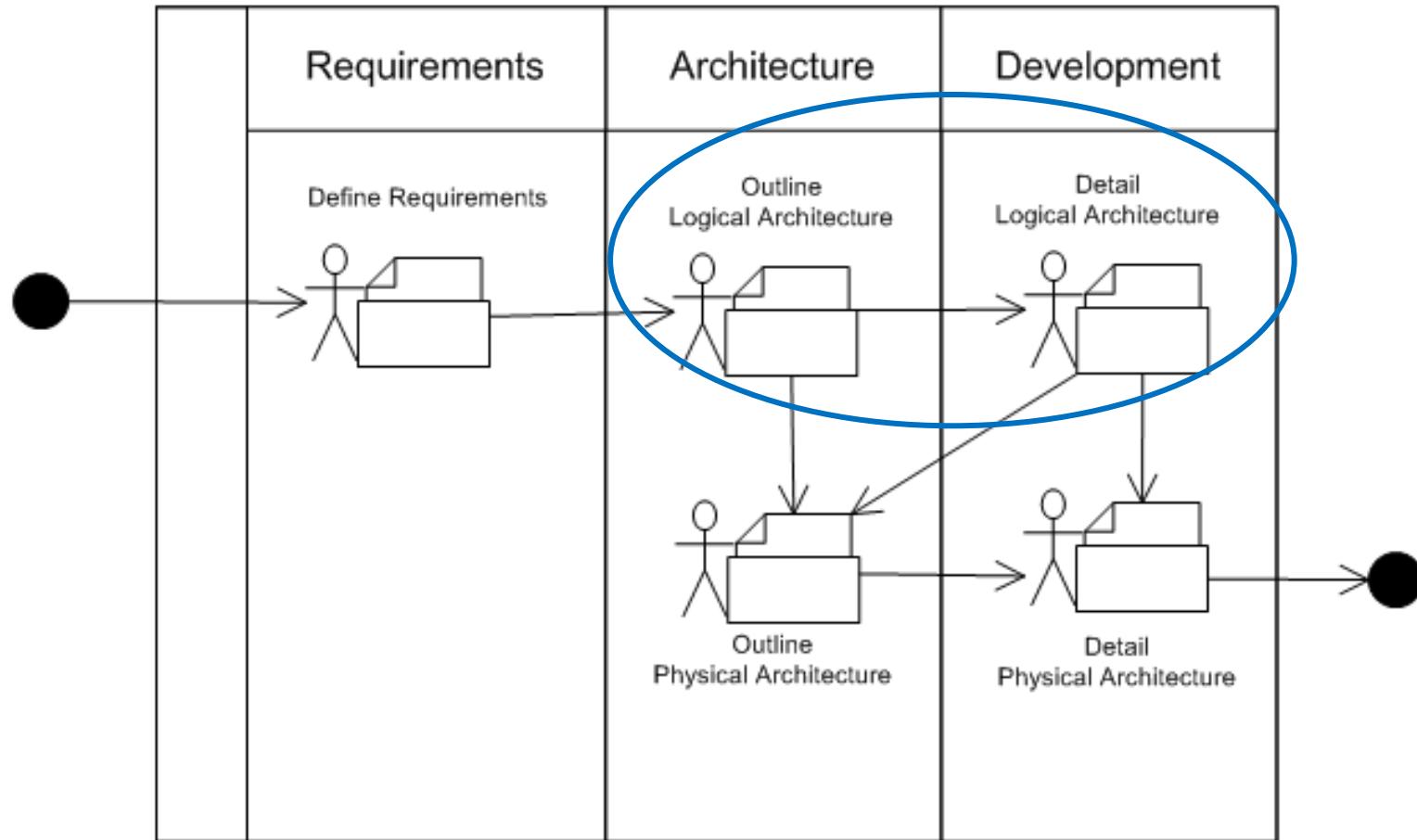
Guidelines

- Focus on **describing** the architecturally significant use cases in detail than the **completeness** of the list of use cases
- Take the whole system (not including the Mainframe) as a **black box** (do not split it into tiers)
- Do not consider the use of **technologies** and **implementation**
- Apart from the **interactions** of the system with the actors, describe also the **inputs and outputs of information**, **flow of information** between the system and actors as well as the **accesses (reading and writing) of information** by the system

- Fundamentals of the Method
- Defining Requirements
- **Defining the Logical Solution Architecture**
- Defining the Physical Solution Architecture



Summary of Activities



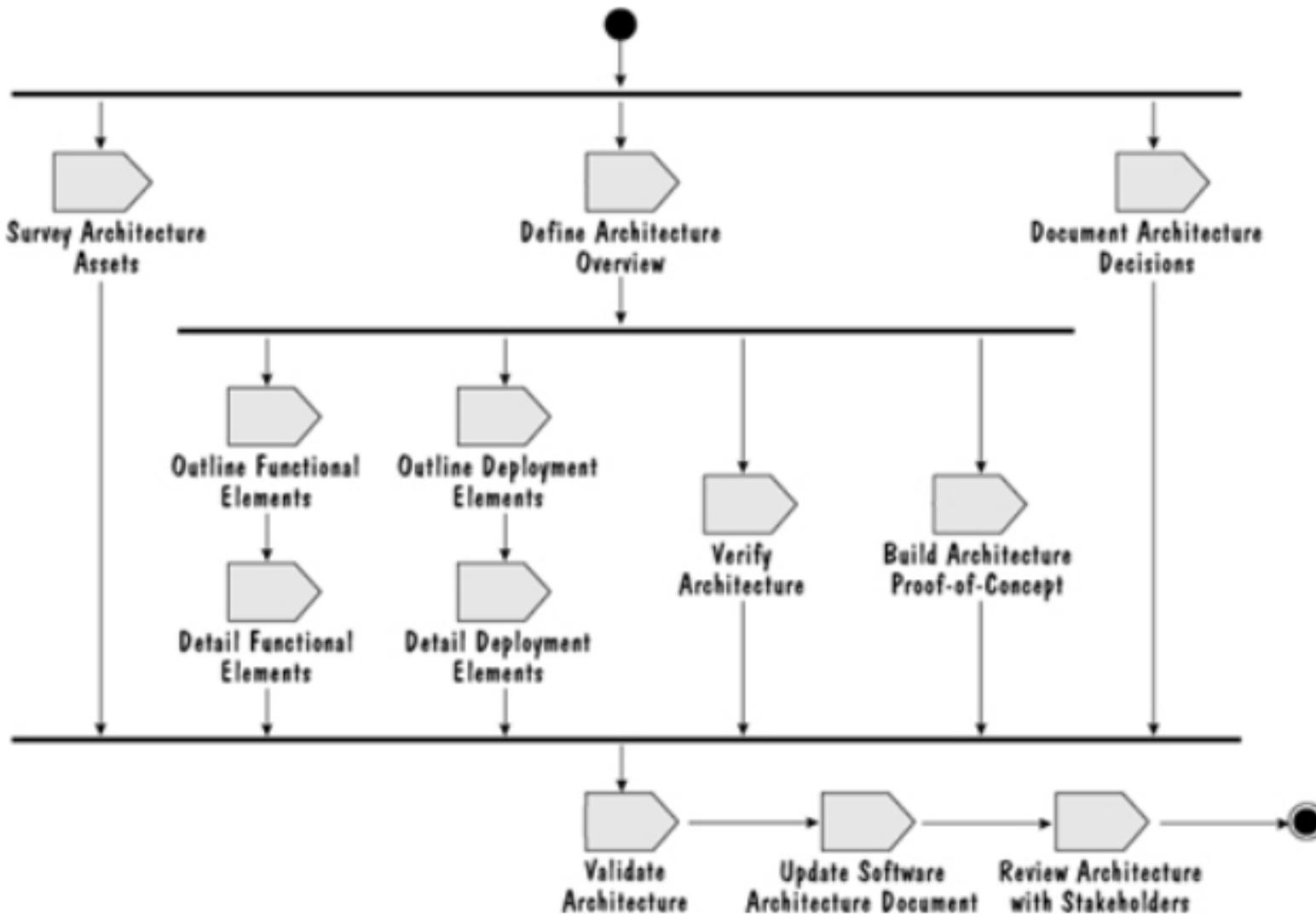


Logical Architecture (Outline versus Detailed)

Element	Outline	Detailed
Functional	Subsystem Component	Interface Operation Operation Signature Mapping to data Pre conditions and post conditions
Deployment	Location Node	Deployment Unit Connection
Role	Solution Architect	Software Architect Data Architect Infrastructure Architect

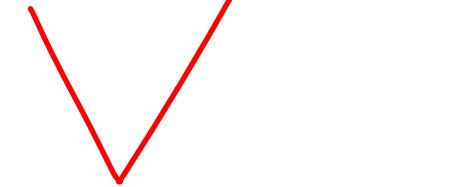


Logical Architecture Activity (Task Summary)





Logical Architecture Activity (Task Summary)



Task
1. Survey Architecture Assets
2. Define Architecture Overview
3. Outline Functional Elements
4. Outline Deployment Elements
5. Verify Architecture
6. Detail Functional Elements
7. Detail Deployment Elements
8. Validate Architecture
9. Update Solution Architecture Document
10. Review Architecture with Stakeholders



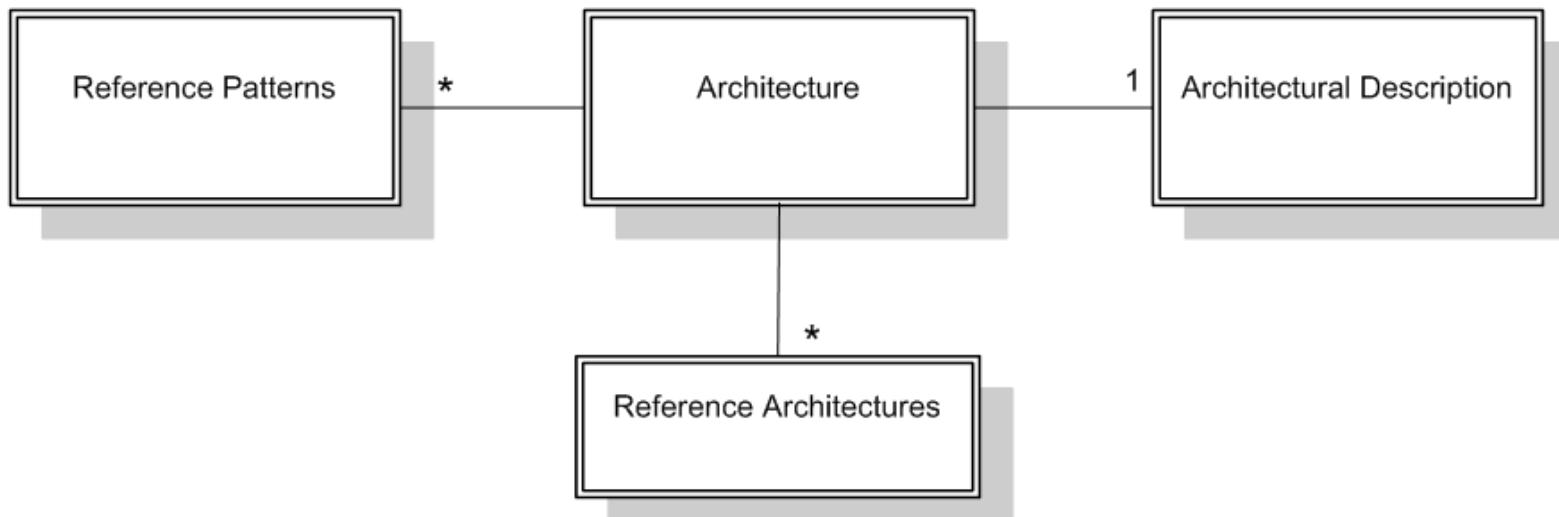
Task 1 – Survey Architecture Assets

Roles	Task Inputs	Task Outputs
Solution Architect Software Architect Infrastructure Architect Data Architect	Architecture Overview Deployment Model Existing IT Environment Enterprise Architecture Principles Functional Model	Architecture Decisions

No	Steps	Things to note (as an Architect)
1	Survey reusable assets	Depending on the project, the architect might focus less on logical and more on physical. (e.g. using reference architecture or packaged solutions)
2	Document architectural decisions & constraints	These constraints and decisions will be referenced in subsequent phases. It is important to document these constraints and decisions as justifications to the architecture.



Task 1 – Survey Architecture Assets (survey reusable assets)





Task 1 – Survey Architectural Assets (document architectural decisions)

Architecture Decision – Develop the custom rules subsystem

Issue	IROne system will use a small number of business rules (about 30) that needs to be modifiable
Architecture Decision	Develop a custom rules engine
Assumptions	The number of rules to be managed is kept at 30, at most 40
Alternatives	<ol style="list-style-type: none">1. Buy a commercial rule engine package, e.g., ILOG JRules2. Embed rules in code
Justification	<p>Given the number of rules is small purchasing the engine might be an overkill and not cost effective</p> <p>Embedding rules in code goes against the best practices of separation of concerns and decoupling, making the system harder to maintain</p>



Task 2 – Define Architecture Overview

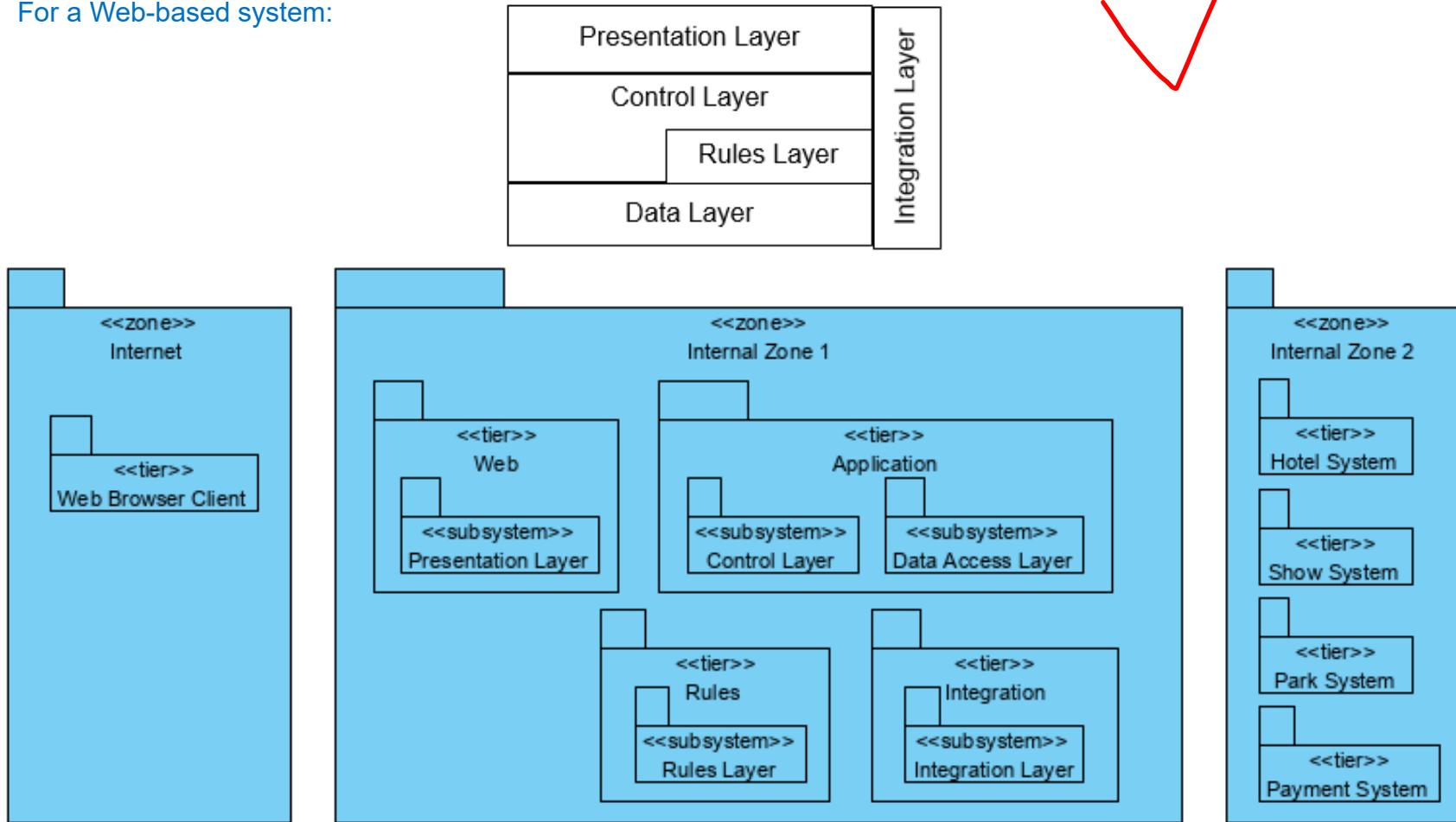
Roles	Task Inputs	Task Outputs
Solution Architect Infrastructure Architect Architect Data Architect	Functional Requirements Enterprise Architecture Principles Glossary Non-Functional Requirements System Context	Architecture Overview

No	Steps	Things to note (as an architect)
1	Create architecture overview	<p>Ensure that the definition of technical terms are consistent.</p> <p>Ensure that all actors and subsystems identified are shown. Use the system context as the starting point, add in the identified subsystems.</p> <p>Ensure that the overview covers the key logical structure (e.g. layers, locations and their relationships) of your architecture.</p>



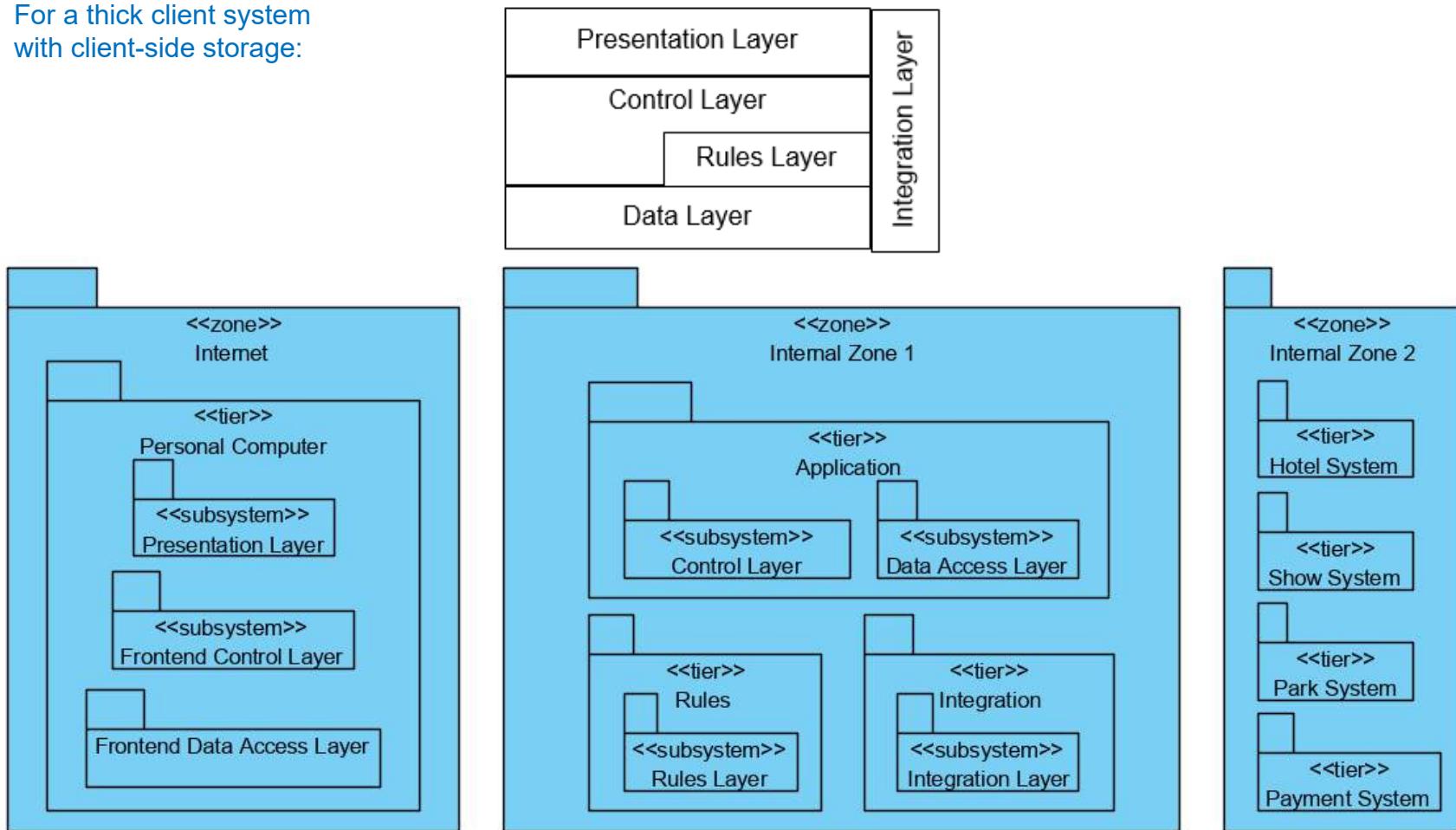
Task 2 – Define Architecture Overview (create the architecture overview)

For a Web-based system:



Task 2 – Define Architecture Overview (create the architecture overview)

For a thick client system
with client-side storage:





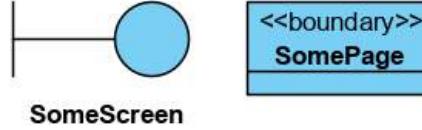
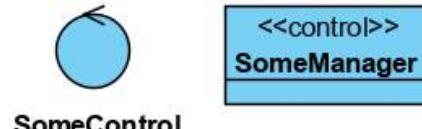
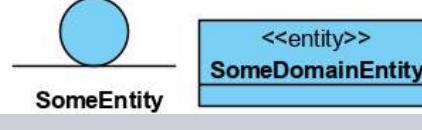
Task 3 – Outline Functional Elements

Roles	Task Inputs	Task Outputs
Solution Architect Infrastructure Architect Data Architect	Architecture Decisions, Architecture Overview, Business Entity Model Business Rules Enterprise Architecture Principles, Existing IT Environment Functional Requirements Glossary, Non-Functional Requirements Prioritized Requirements List Architecture Proof-of-Concept	Functional Model

No	Steps	Things to note (as an Architect)
1	Identify functional components	Existing components identified for other use cases should be reviewed and reused.



Task 3 – Outline Functional Elements (identify functional components)

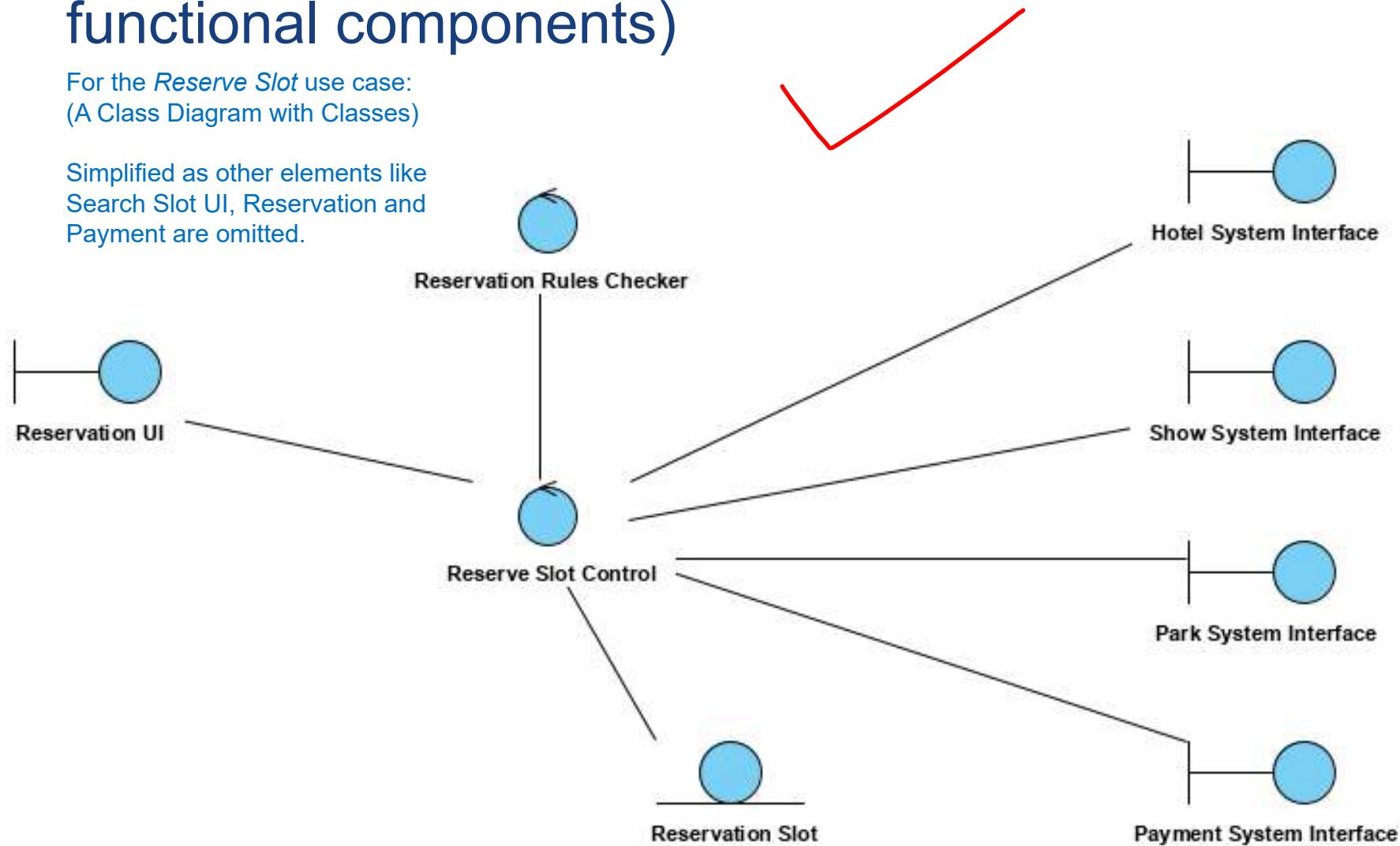
Type	UML Representation	Description
Boundary		<p>Boundary components represent the boundary between the system and its environment</p> <ol style="list-style-type: none">1. Interface to the <u>primary actor</u> to handle requests and responses to the actor2. Interface to <u>external system</u> to get or send data
Control		<p>Control components represent the control and coordination logic of the system.</p> <ol style="list-style-type: none">1. Controller to handle <u>overall control</u> of the use case2. For <u>each use case</u>, assign <u>a controller</u> to manage that interaction; break it into sub-controllers only if it is too complex
Entity		<p>Entity components encapsulate the information represented in the system.</p> <ol style="list-style-type: none">1. Define the entity component to <u>manage the use of data</u>



Task 3 – Outline Functional Elements (identify functional components)

For the *Reserve Slot* use case:
(A Class Diagram with Classes)

Simplified as other elements like
Search Slot UI, Reservation and
Payment are omitted.





Task 4 – Outline Deployment Elements

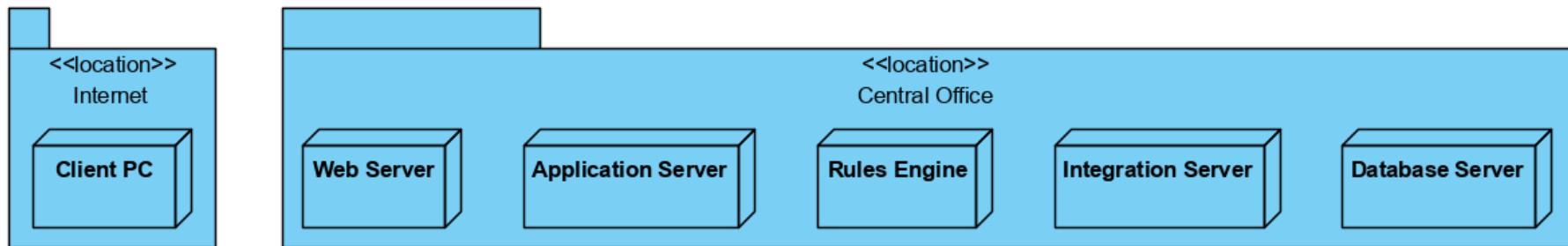
Roles	Task Inputs	Task Outputs
Solution Architect Infrastructure Architect Data Architect	Architecture Decisions (optional), Architecture Overview, Architecture Proof-of-Concept (optional), Enterprise Architecture Principles, Existing IT Environment (optional), Functional Model, Functional Requirements, Glossary, Non-Functional Requirements, System Context	Deployment Model

No	Steps	Things to note (as an Architect)
1	Identify locations & nodes	Identification of the initial locations can be based on the existing it environment, architectural decisions, non functional requirements and the system context. At this moment, the nodes are purely logical containers.



Task 4 – Outline Deployment Elements (identify locations & nodes)

(A Deployment Class with Nodes)





Task 5 – Verify Architecture

Roles	Task Inputs	Task Outputs
Solution Architect Infrastructure Architect Data Architect	Architecture Decisions, Architecture Overview, Architecture Proof-of-Concept, Deployment Model, Functional Model, Functional Requirements, Non-Functional Requirements, System Context	Review Record
No	Steps	Things to note (as an Architect)
1	Review Architecture	<p>The architect needs to ensure that any cross cutting concerns have been addressed consistently. Architectural decisions that come out of the verification needs to be captured.</p> <p>This step can involve planning verification, holding kickoff meeting, performing individual verification and holding verification meeting. Any rework identified needs to be captured and follow-up.</p>



Task 6 – Detail Functional Elements

Roles	Task Inputs	Task Outputs
Software Architect Infrastructure Architect Data Architect	Architecture Decisions, Business Entity Model, Functional Model, Glossary	Data Model, Functional Model

No	Steps	Things to note (as an Architect)
1	Detail functional components	<p>For this step, the Architect needs to ensure the component interfaces are defined with the operation and operation signatures.</p> <p>Contracts between components (preconditions and postconditions) should be defined (if any)</p> <p>The Architect will also ensure that the logical data model is created.</p>



Task 7 – Detail Deployment Elements

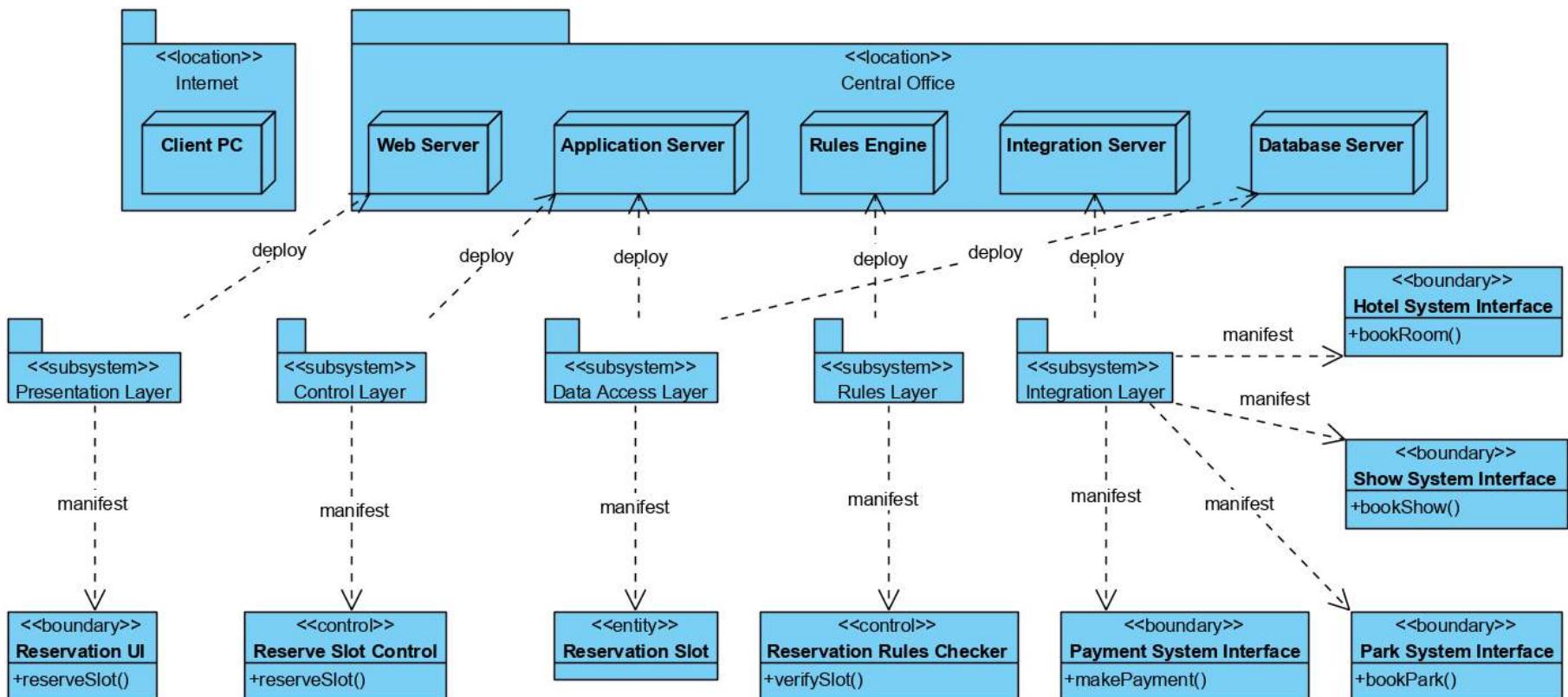
Roles	Task Inputs	Task Outputs
Software Architect Infrastructure Architect Data Architect	Architecture Decisions, Deployment Model, Functional Model, Glossary, Non- Functional Requirements	Deployment Model

No	Steps	Things to note (as an Architect)
1	Detail deployment components	For this step, the Architect will need to assign components to nodes and define connections between nodes and locations.



Task 7 – Detail Deployment Elements (detail deployment components)

For the *Reserve Slot* use case:
(A Deployment Diagram with Nodes, Packages and Classes)





Task 8 – Validate Architecture

Roles	Task Inputs	Task Outputs
Solution Architect Infrastructure Architect Data Architect Project Manager	Architecture Assessment, Change Request, RAID Log	All architecture-related work products

No	Steps	Things to note (as an Architect)
1	Plan validation	The architect should lead this planning and validation session.
2	Create architectural proof-of-concept	The purpose is to reduce risk. It can be a simple sketch of the conceptual model of the solution to a workable prototype.
3	Review architecture	Findings should be well-documented. Assess the risks and make recommendations.



Task 9 – Update Solution Architecture Document

Roles	Task Inputs	Task Outputs
Solution Architect	Architecture Assessment, Architecture Decisions, Architecture Overview, Architecture Proof-of-Concept, Data Model, Deployment Model, Functional Model	Solution Architecture Document

No	Steps	Things to note (as an Architect)
1	Update Solution Architecture Document	The architect is responsible for this task



Task 10 – Review Architecture with Stakeholders.

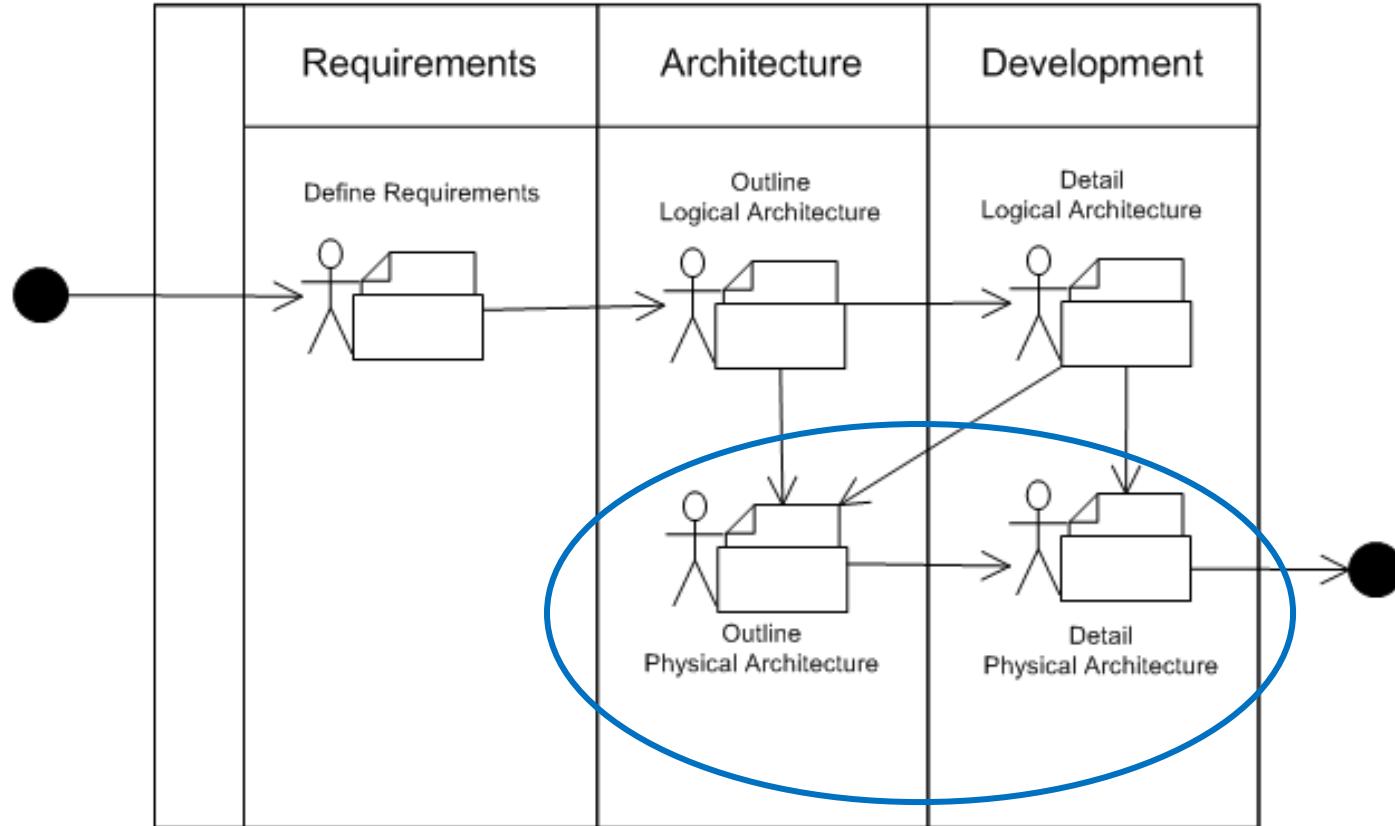
Roles	Task Inputs	Task Outputs
Stakeholder Solution Architect Infrastructure Architect Data Architect Project Manager	All	Change Request Request Log Review Record

No	Steps	Things to note (as an Architect)
1	Review work Products	Ensure the products are baselined Ensure that any questions of technical nature have been answered

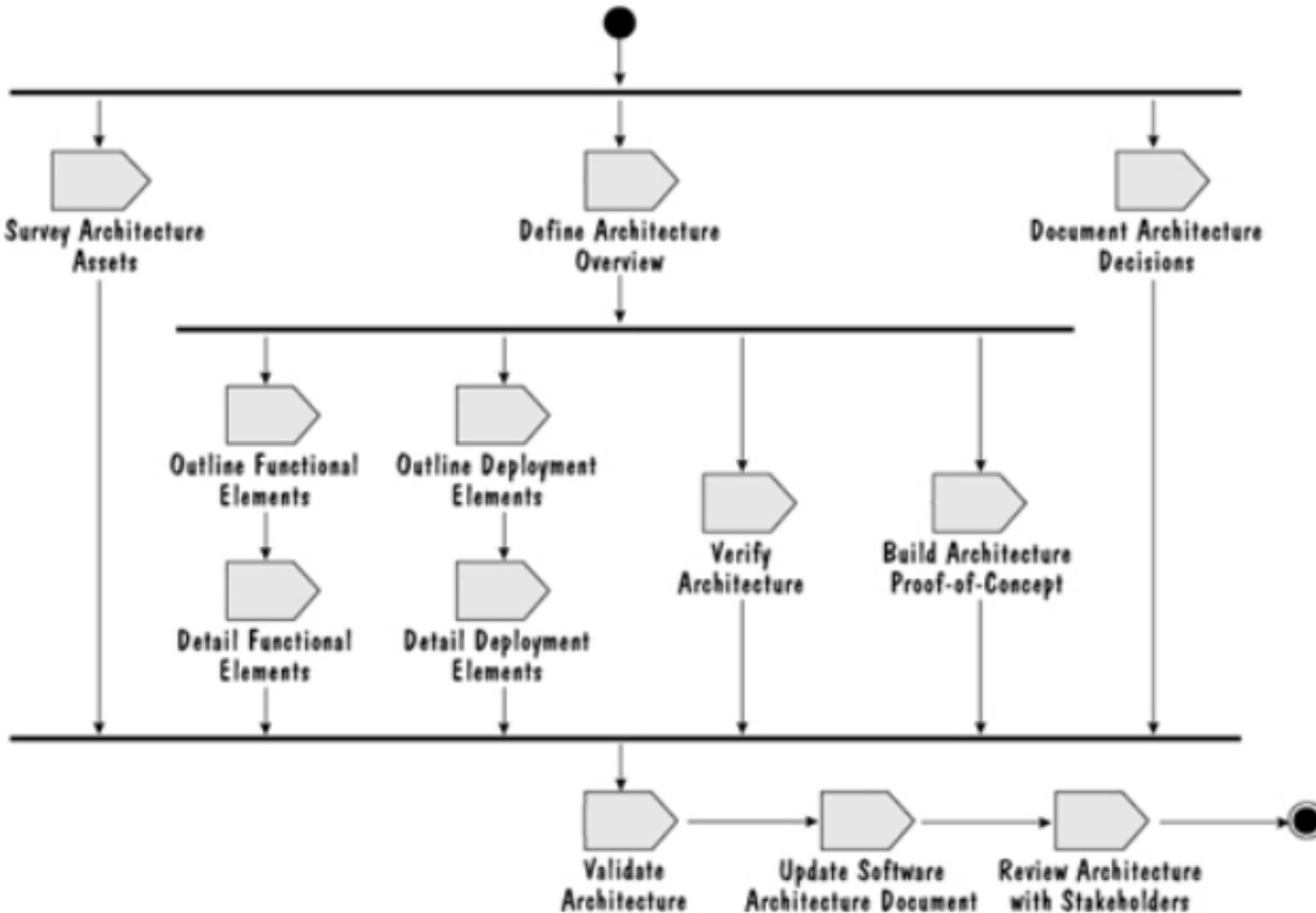
- Fundamentals of the Method
- Defining Requirements
- Defining the Logical Solution Architecture
- **Defining the Physical Solution Architecture**



Summary of Activities



Physical Architecture Activity (Task Summary)





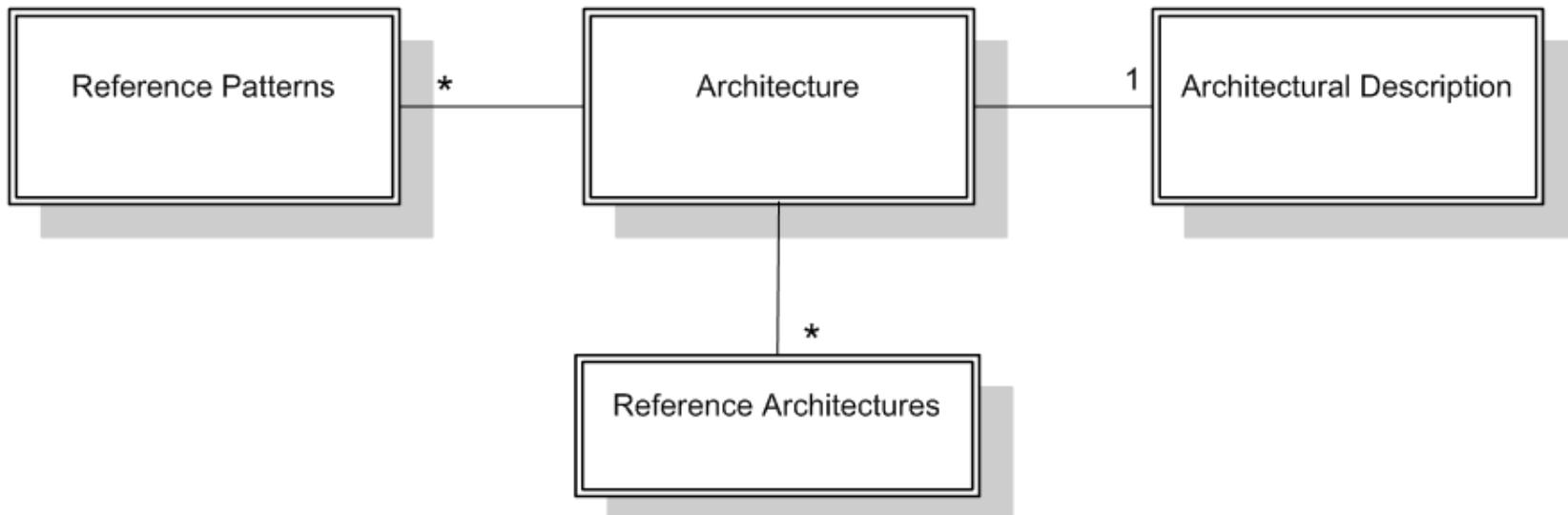
Physical Architecture Activity (Task Summary)

Task

1. Survey Architecture Assets
2. Define Architecture Overview
3. Outline Functional Elements
4. Outline Deployment Elements
5. Verify Architecture
6. Detail Functional Elements
7. Detail Deployment Elements
8. Validate Architecture
9. Update Solution Architecture Document
10. Review Architecture with Stakeholders



Task 1 – Survey Architecture Assets (survey reusable assets)



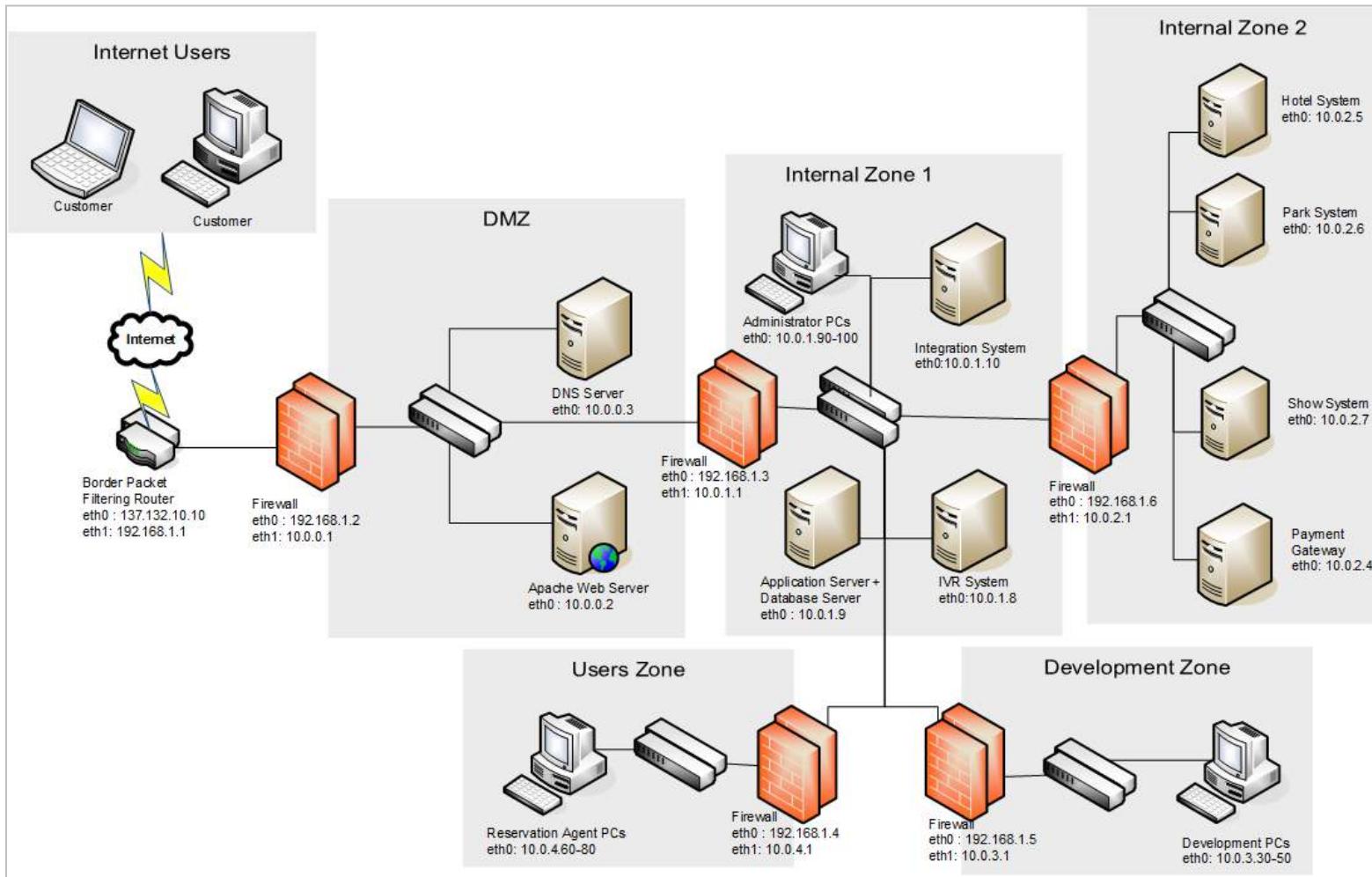


Task 1 – Survey Architecture Assets (document architectural decisions & constraints)

Architectural Decision – Use IBM WebSphere Application Server

Issue	The IROne system will run in a Java EE server
Architecture Decision	Use IBM WebSphere Application Server
Assumptions	N. A.
Alternatives	<ol style="list-style-type: none">1. Oracle WebLogic2. Microsoft .NET Server
Justification	There are existing skills in developing and deploying on the IBM WebSphere Application Server

Task 2 – Define Architecture Overview (create the architecture overview)



Task 3 – Outline Functional Elements (identify functional components)

Moving from logical to physical components:

- One to One

Realize one logical element with one physical element if the physical element can satisfy the full specification

- One to many

Realize one logical element into multiple physical elements,
e.g., for availability

- Many to One

Realize many logical elements into one physical element, e.g., a packaged application

Pack



Task 3 – Outline Functional Elements (Identify functional components)

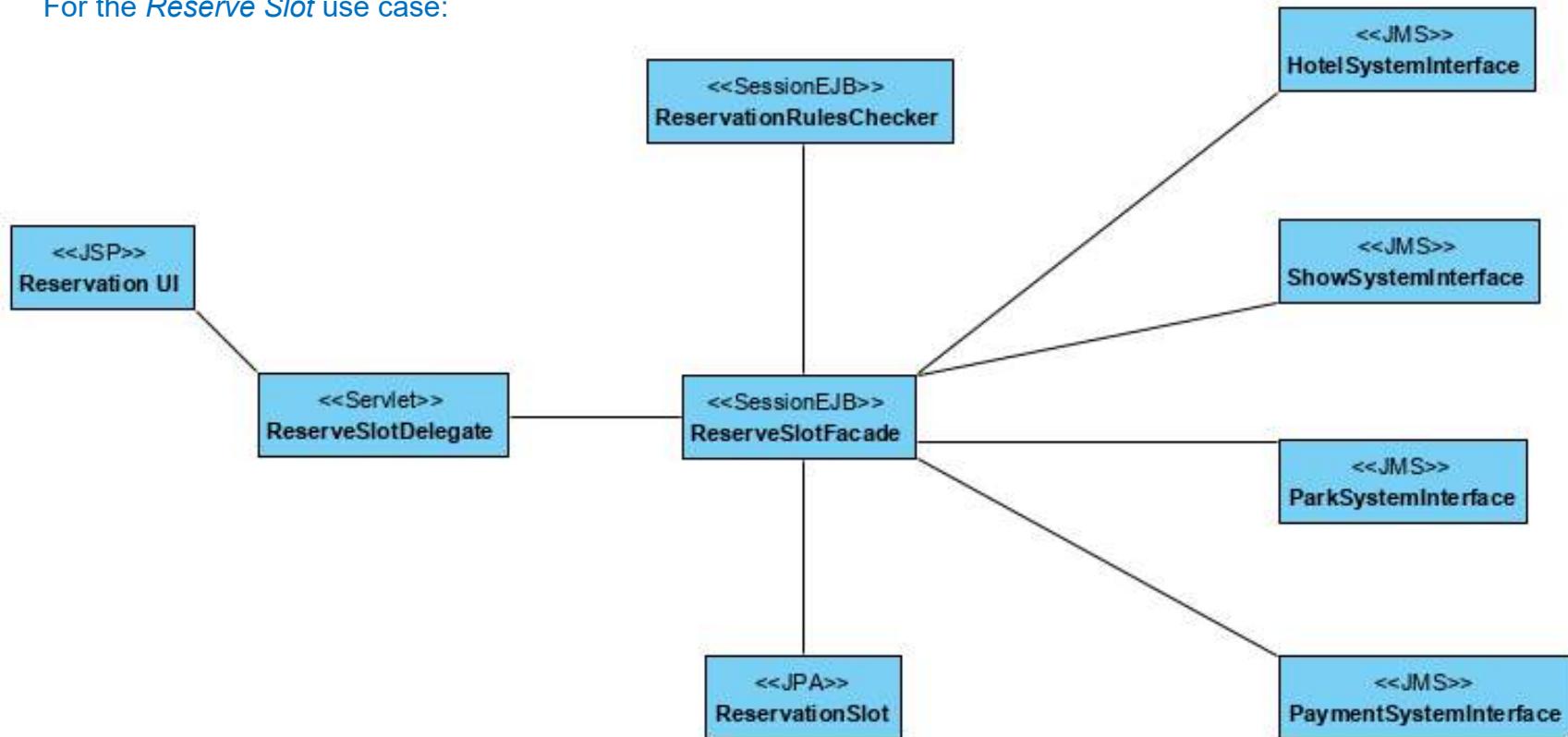
Outline functional elements using Java EE:

- For each boundary component representing interaction with persons, implement with JSPs and servlets
- For each boundary component representing interaction with systems, implement with JMS
- For each controller component, implement with Session EJB
- For each entity component, implement with JPA

Task 3 – Outline Functional Elements (identify functional components)

Outline functional elements using Java EE

For the *Reserve Slot* use case:



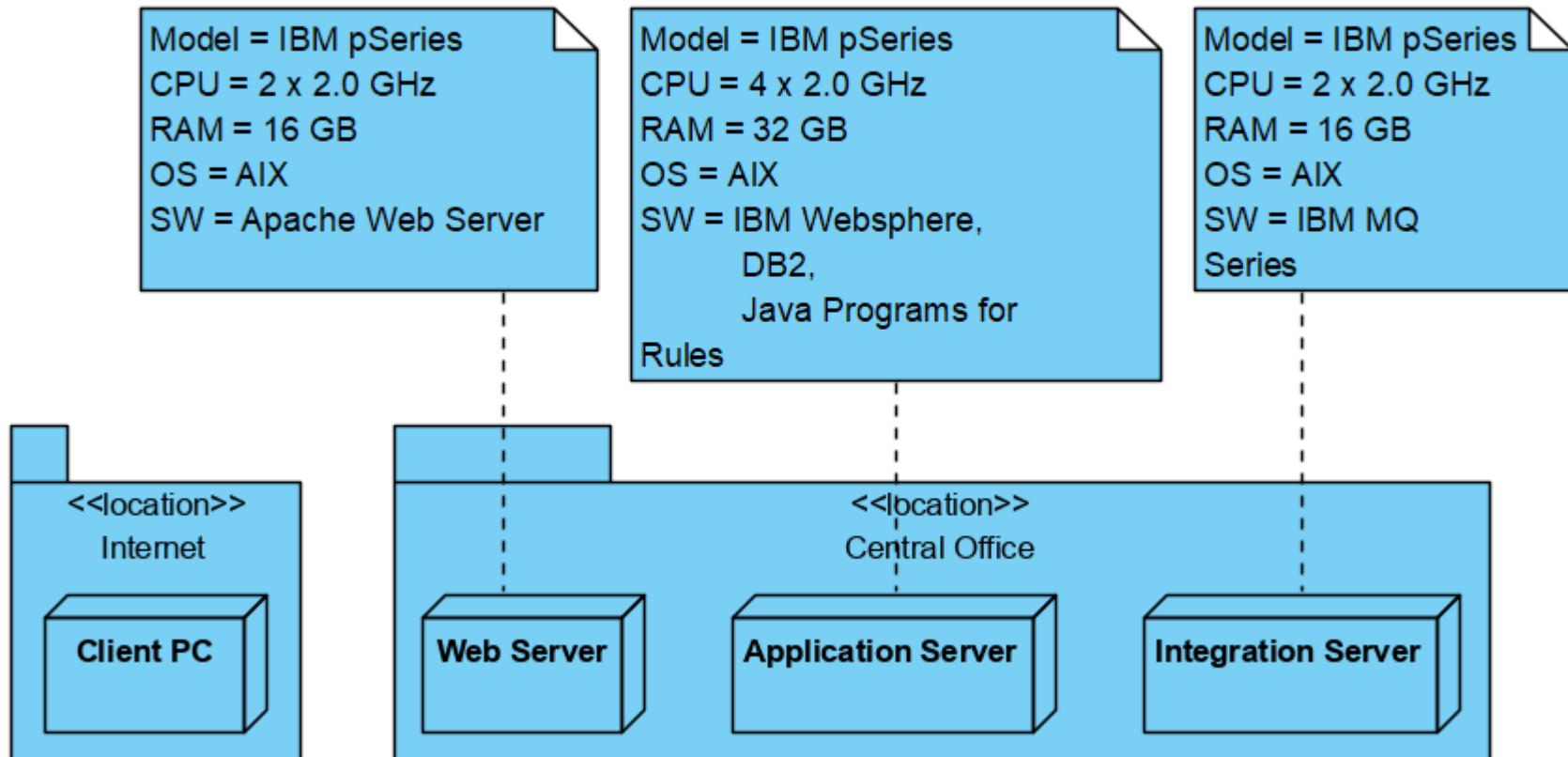


Task 4 – Outline Deployment Elements (identify locations & nodes)

Identifying physical deployment elements:

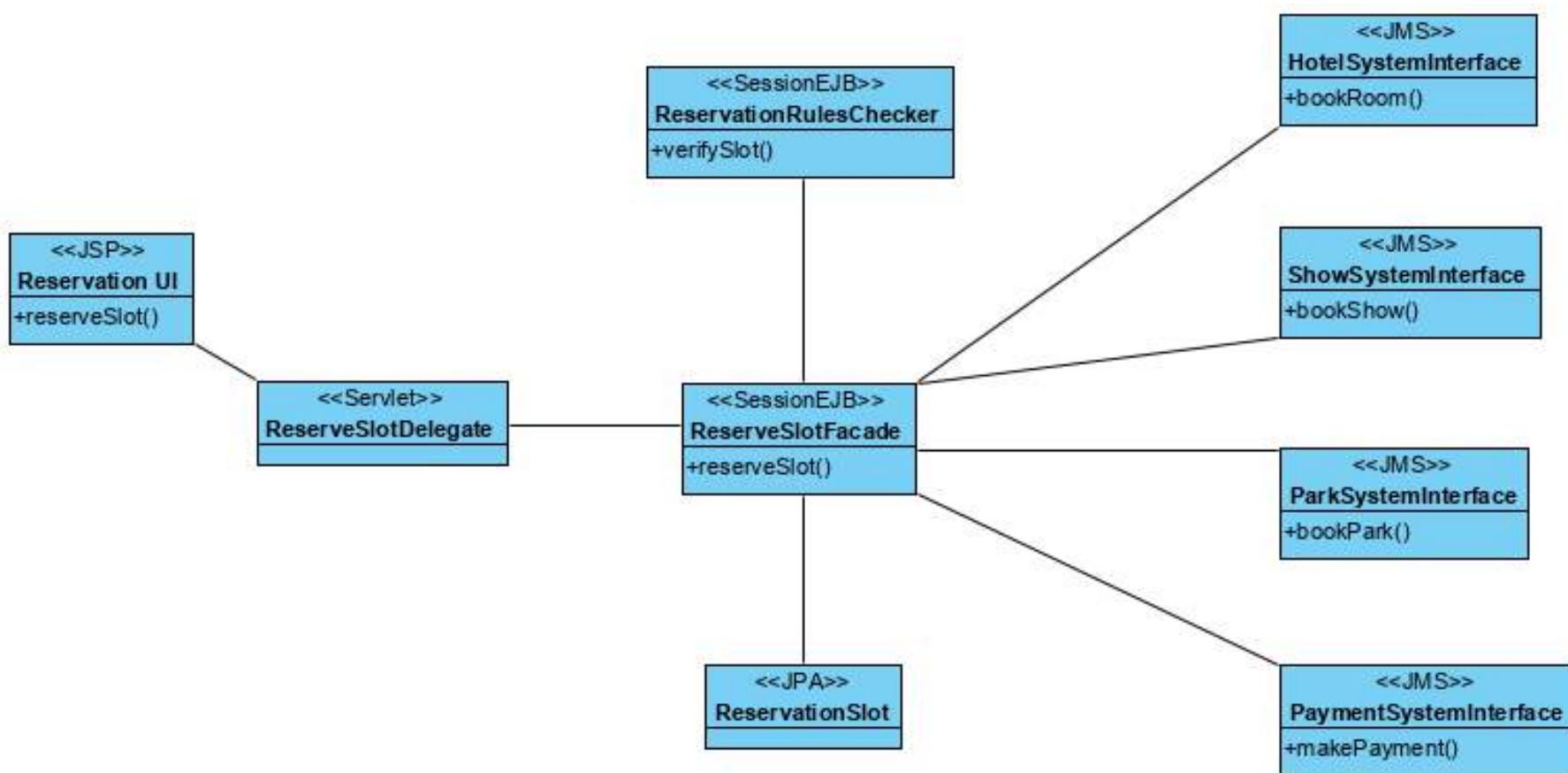
- For each logical location, define how many physical instances
- For each node in each logical location, define the corresponding physical node or nodes
- Within each physical node, define the hardware and software specification
- Size the physical nodes to meet the non-functional requirements

Task 4 – Outline Deployment Elements (identify locations and nodes)



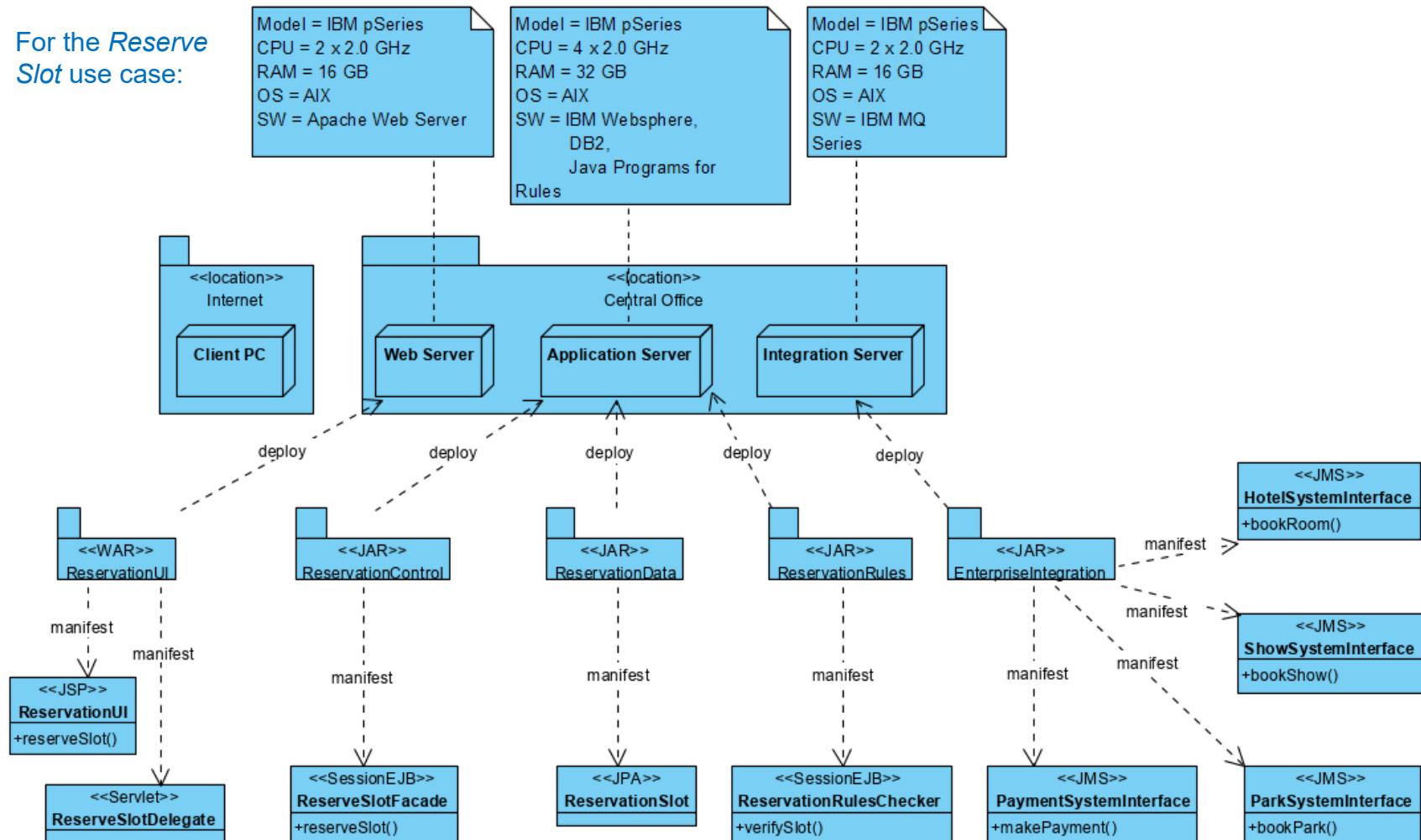
Task 6 – Detail Functional Elements (detail functional components)

For the *Reserve Slot* use case:



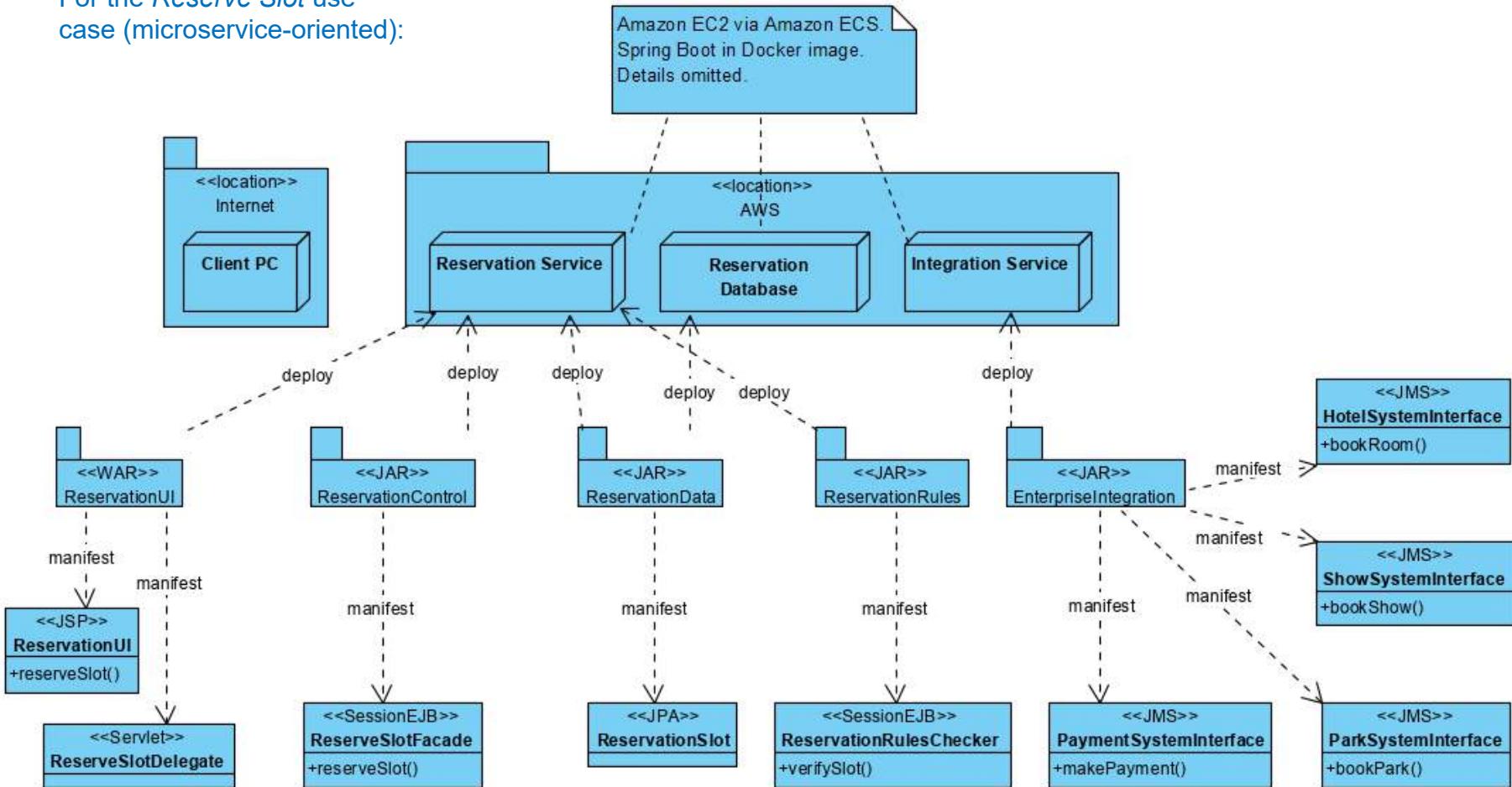
Task 7 – Detail Deployment Elements (detail deployment components)

For the *Reserve Slot* use case:



Task 7 – Detail Deployment Elements (detail deployment components)

For the *Reserve Slot* use case (microservice-oriented):





Task 7 – Detail Deployment Elements (Detail deployment components).

Integration Endpoints					
Source	Destination	Protocol	Format	One/Two way	Transformation Needed?
Client System	Web Server	HTTP / HTTPS	HTML / JSP	Two way	No
Web Server	Application Server	HTTP / HTTPS	HTML / JSP	Two way	No
Application Server	Show System	JMS	XML	Two way	Yes
Application Server	Hotel System	JMS	XML	Two way	Yes
Application Server	Park System	JMS	XML	Two way	Yes

Architecting the Student Health System (Exercise 2)



Summary.

- The architectural method consists of phases, iterations and activities
- Each activity consists of tasks performed by roles responsible for a list of work products
- These deliverables are produced:
 - System Context
 - Functional and non-functional requirements
 - Logical Architecture Overview
 - Logical Functional Model
 - Logical Deployment Model
 - Physical Architecture Overview
 - Physical Functional Model
 - Physical Deployment Model



APPENDIX A

(Sample Scenario)



Example Scenario

Current Business Environment

- Currently the general information on the amenities is available online through the web channel
- The booking services are currently offered through phone calls
- A reservation agent will pick up the call and process the reservation



Example Scenario

Current IT Environment

- Currently, there is an existing web channel for SSOne to provide static web content. To book the amenities, the customer can review the information on the web channel and call up the reservation agent through the Integrated Voice Reservation (IVR) system.
- The reservation agent's PC is linked to the amenity systems and common payment gateway through each of the specific system web user interface .
- The web channel is powered by the Apache Web Server running on Microsoft Windows Server. A DNS server responds to DNS queries to resolve the name of the web server. The administrator uses Telnet to remotely administer the servers (DNS, Web Server and IVR System).



Example Scenario

System Requirements for IROne

- The system must allow the customer to make amenity bookings online by selection of the amenity and the time slot, to making payment in an integrated process.
- Payment for the bookings will be through credit card only. Payment details will need to be transmitted securely to the common payment gateway system for processing.
- Once the transaction is done, email notifications must be sent to the customer. The customers can also send emails to enquire.
- There should be a rule engine in the architecture to handle business rules. 30 rules are estimated to be incorporated in the application.



Example Scenario

Other Requirements for IROne

- The Enterprise Architecture team of SSOne will review the IROne solution architecture to ensure compliance with the architectural principles defined.
- The amenity systems and the payment gateway can be integrated through Web Services using SOAP protocol over HTTP or HTTPS. The operators of the amenity systems are external vendors and expects no extensive modifications required.
- The IROne system will be built using Java EE technology. The software architecture will be based on three tiers (Presentation, Business Logic and Data). Servlets and JSPs will be used in the web tier for the presentation, enterprise java beans (EJBs) for the application logic and an Oracle database for the data tier.



Example Scenario

Other Requirements for IROne

- There are other successful projects being implemented based on Java EE technology. The technology being used is the IBM WebSphere Application Server.
- The availability of each of the proposed three tiers should be at least 99.9%.
- Currently all the systems are located at a single site in Singapore for ease of maintenance. The web server, payment gateway, DNS server, IVR system and other PCs are located at the Central Office. The amenity systems are located at their respective offices (Show, Hotel & Park offices).



ARCHITECTING SOFTWARE SOLUTIONS

DESIGN SECURE ARCHITECTURE

Instructor: Angela HUANG

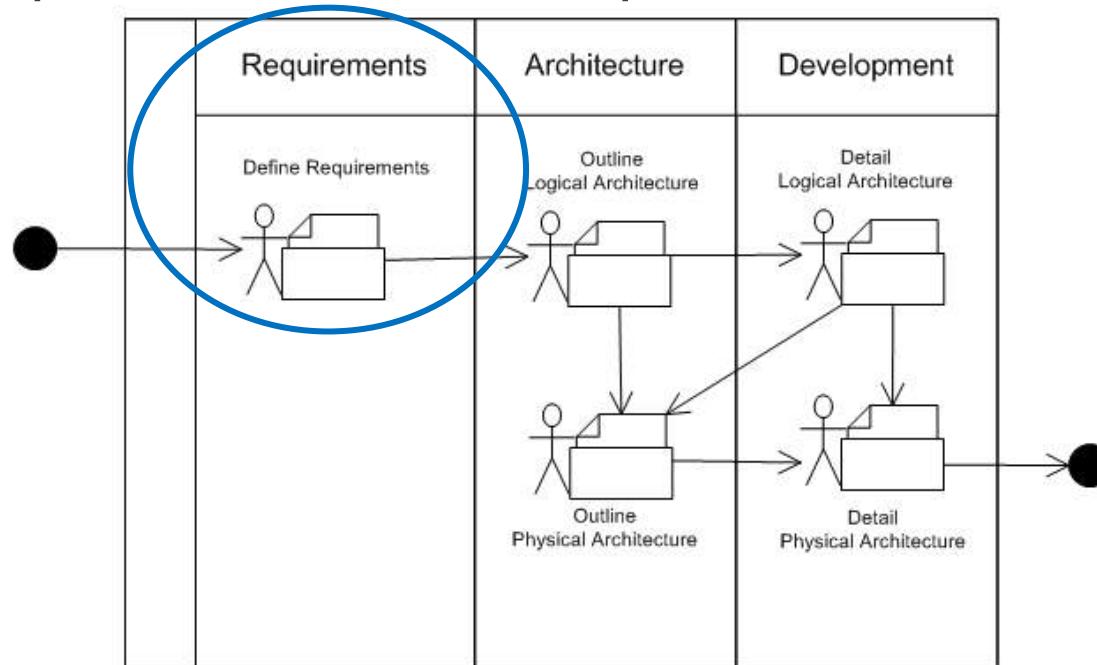
Email: ah88@nus.edu.sg

Total slides: 114



Introduction

- To design a secure architecture,
 - Security requirements must be included during requirements definition phase



- Solution Architecture must be reviewed from a security perspective.
 - Risk Management
 - Risk assessment
 - Risk Treatment

Threat Modelling



Learning Outcome

- Be able to:
 - Explain security concepts
 - Conduct risk assessment
 - Apply the appropriate **security controls** and secure design principles to mitigate risks

- Security Concepts
- Risk Management
 - Risk Assessment
 - Risk Treatment
- Security Controls
- Secure Design Principles
- Reference Architectures (security)

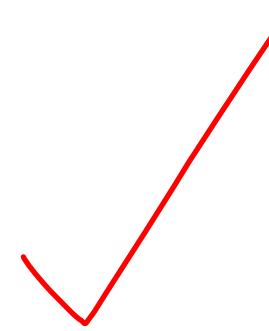


SECURITY CONCEPTS



Information Security

Preservation of the **CIA** of information



ISO27000

Protect the accuracy of information processing and data from improper modification

Integrity

Protect against **unauthorized disclosure** of information

Confidentiality

Availability

Ensure that the systems and data **can be accessed when required**



Recall ISO 25010



CIA/AAA/NR

Authentication, Authorization, and Accounting

Non-repudiation is the assurance that someone cannot deny the validity of something



RISK MANAGEMENT



Risk Management Process (on-going)

Determine Scope



Risk Assessment

- Risk Identification
- Risk Analysis
- Risk Evaluation



- **Identify:**
 - Assets
 - Threats
 - Vulnerabilities
 - Consequence
- **Construct risk scenarios.**



Assets

- **Crown Jewels**

- Assets **critical** to achieving the business objectives of the organization.
- Typically critical data e.g. customer account data of a bank, patient medical records of a hospital



- **Stepping Stones**

- Assets attackers would want to take control of and leverage to pivot across network segments before reaching the crown jewels

- Any potential **danger** to **information** or the **IT system**.
- **STRIDE Threat taxonomy**
 - Threats to CIA I4A Identification 4 authentication
 - Threats to components of access control, AAA
 - Threat to **non-repudiation**
- **Think like a hacker.**



Threats

- STRIDE Threat taxonomy

Security Objective	Threat
Authenticity	Spoofing
Integrity	Tampering
Non-Repudiation	Repudiation
Confidentiality	Info Disclosure
Availability	Denial of Service (DoS)
Authorization	Elevation of Privilege





Vulnerability

- A **flaw** or **weakness** in the system which can be exploited by the threat to harm an asset.





Vulnerability (cont'd)

In other words, vulnerabilities occur when the software solution you have architected **does not** perform one or more of the following:

- Protect CIA of data
- Carry out AAA properly
- Provide NR (where required)

For more examples of vulnerabilities, refer to [Common Weakness Enumeration \(CWE\)](#)

- The following **quadruple** constitutes a **risk scenario**:
 - **Assets**
 - **Threats**
 - Start with STRIDE, then think like a hacker.
 - Focus on threats to:
 - Data Store, Data Flow, Processes, Actors, Trust Boundaries
 - **Vulnerabilities**
 - **Consequence** (direct result of threat event)
 - E.g. loss of PII, disruption of service
- **Risk Scenario**
 - <*threat*> exploits <*vulnerability*> to harm <*asset*> resulting in <*consequence*>.
 - E.g.
 - Unauthorized access (*threat*) to cleartext medical records (*asset*) because they are not stored encrypted (*vulnerability*) resulting in leakage of sensitive information (*consequence*).
 - Ransomware (*threat*) exploiting an unpatched (*vulnerability*) system to encrypt business data (*asset*) resulting in stoppage in business operations (*consequence*).



Risk Analysis

Risk Assessment

- Risk Identification
- Risk Analysis ➔
- Risk Evaluation

- For **each** risk identified, determine its **Likelihood & Impact**.
 - **Likelihood** that the **threat** will **exploit** the **vulnerability** to harm the asset.
 - **Impact**: magnitude of harm (resulting from the occurrence of the threat).
- For **qualitative** risk analysis, we can use the **DREAD** framework to determine likelihood and impact.

DREAD

- **Discoverability.** How easy is it to discover this vulnerability?
 - 1 = very hard to impossible; requires source code or administrative access.
 - 5 = can figure it out by guessing or monitoring network traces.
 - 9 = such details of faults are already in the public domain and can be easily discovered using a search engine.
 - 10 = the information is visible in the Web browser address bar or in a form.
- **Reproducibility.** How easy is it to reproduce the attack?
 - 1 = very hard or impossible, even for administrators of the application.
 - 5 = one or two steps required, just need to be an authorized user.
 - 10 = just a Web browser and the address bar is sufficient, without authentication
- **Exploitability.** How much effort is required to exploit this vulnerability?
 - 1 = advanced programming and networking knowledge, with custom or advanced attack tools. Need superuser access.
 - 5 = malware exists on the Internet, or an exploit is easily performed, using available attack tools.
 - 10 = just a Web browser.

- **Affected users.** How many users will be affected?
 - 1 = none.
 - 5 = some users, but not all.
 - 10 = all users.
- **Damage potential.** If a threat exploit actually occurs, how much damage will it cause?
 - 1 = nothing.
 - 5 = individual user data are compromised or affected.
 - 10 = complete system or data are destruction



Risk Evaluation

- Determine risk

Risk Assessment

- Risk Identification
- Risk Analysis
- Risk Evaluation



Risk = function (Likelihood, Impact)

Risk is the **likelihood** of a **threat** exploiting a particular **vulnerability** to harm an **asset**, resulting in a business **impact**.

- DREAD (Qualitative):

DIscoverability
RReproducibility
EExploitability

AAffected Users
DDamage Potential

Risk = Probability of Occurrence × Business Impact

$$\text{Risk} = (R_{\text{value}} + E_{\text{value}} + DI_{\text{value}}) \times (D_{\text{value}} + A_{\text{value}})$$



Risk Management Process (on-going)

Determine Scope

- **Risk treatment options** (for each risk scenario)
 - **Mitigate**
 - Implement Security Controls
 - Refer to “Security Controls” and “Secure Design Principles” section
 - **Avoid**
 - Do not carry out the activity that gives rise to the risk
 - **Transfer**
 - **Accept** spend 100 dollar to save 100 dollar?



Security Controls

- **Safeguards or Countermeasures** that act to prevent or minimize losses associated with occurrences of threats
 - E.g. **encryption** of confidential data



Security Controls

- Security Control Categories
 - Resisting Attacks (Preventive Controls)
 - Detecting Attacks (Detective Controls)
 - Reacting to Attacks (Response Controls)
 - Recovering from Attacks (**Recovery** Controls)
- Secure Design Principles



SECURITY CONTROLS



Resisting Attacks

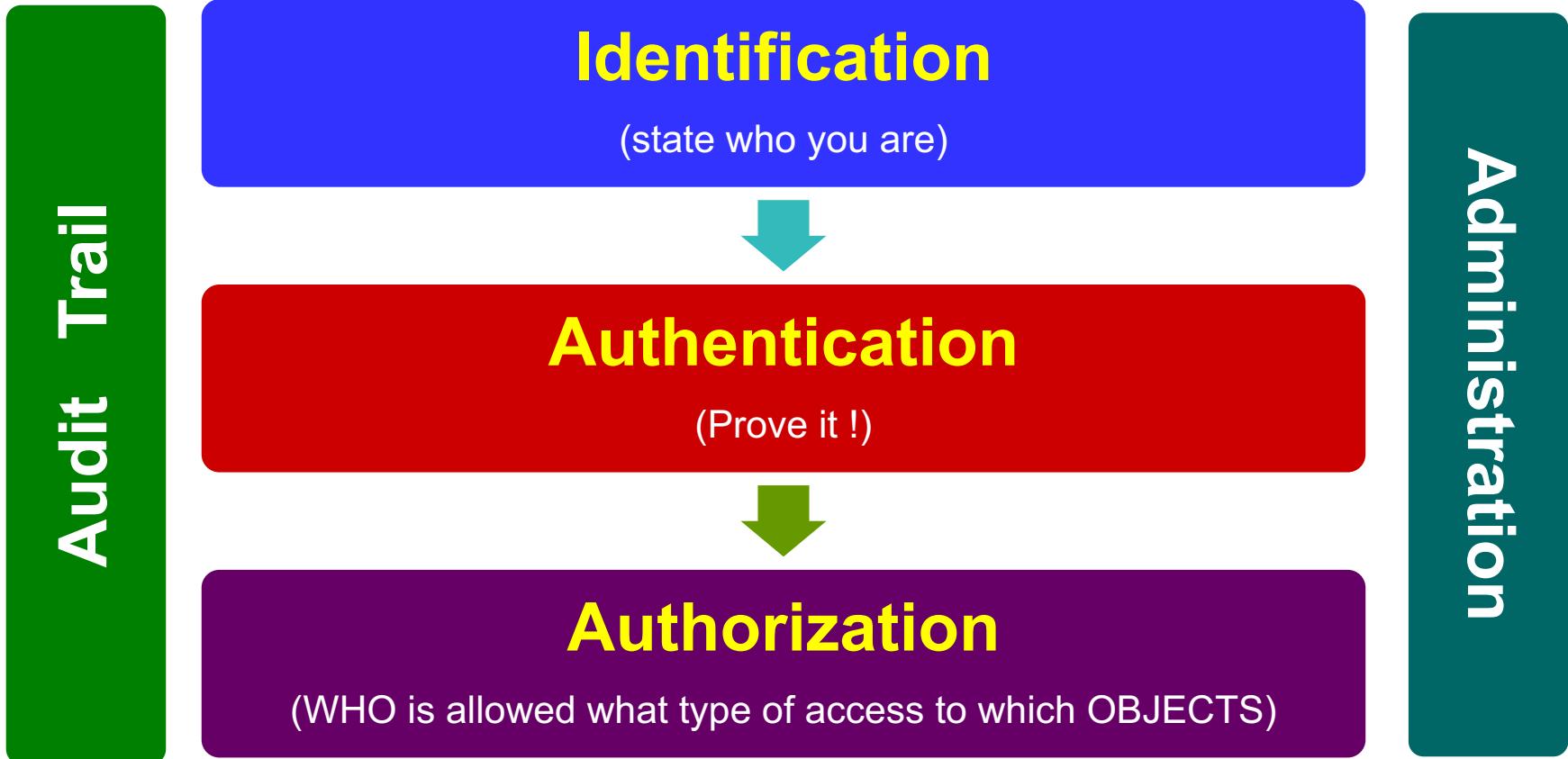
- **Access Control**
- **Cryptography**
 - Symmetric vs Asymmetric Keys
 - Encryption
 - Hashing
 - Digital Signatures
 - Digital Certificates
 - HMAC
- **Data-Centric safeguards**
- **Network Segregation**



Access Control

I4A

NRIC = PII so fall into PDPA - but audit trail will log everything.
Singpass will be able to transalte IC --> arbitrary string

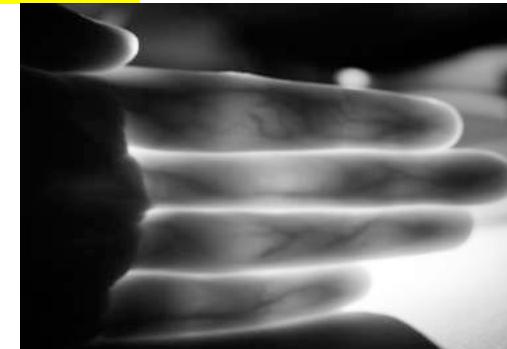


Baseline security control



Authentication Methods

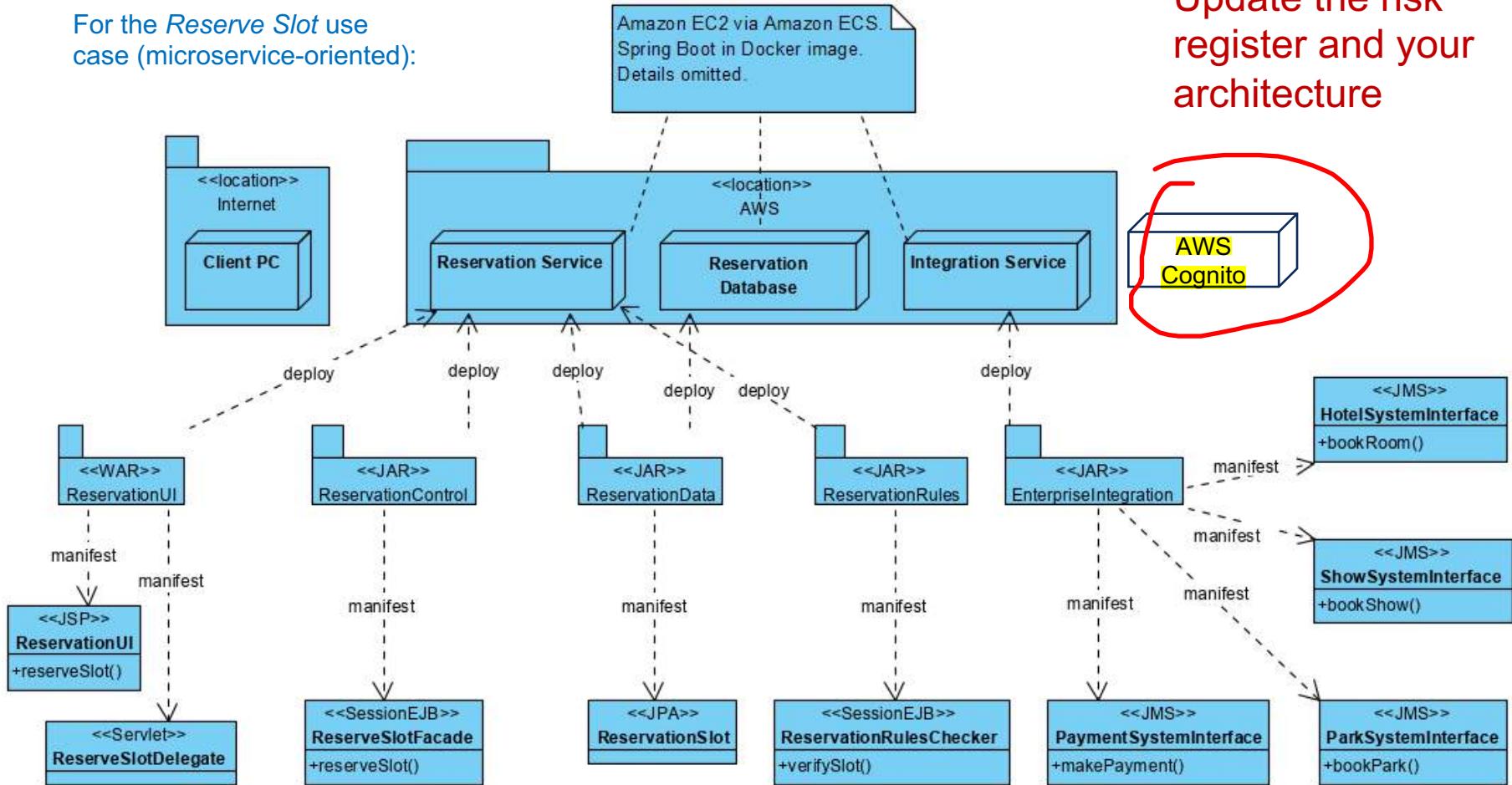
- **What you Know.**
 - E.g. password, passphrase
- **What you Have.**
 - E.g. OTP generator token
- **What you Are.** ↴
 - Biometrics
 - Physiological e.g. iris, fingerprint, **palm vein**
 - Behavioural e.g. handwriting, gait





Task 7 – Detail Deployment Elements (detail deployment components)

For the *Reserve Slot* use case (microservice-oriented):



Update the risk register and your architecture



Resisting Attacks

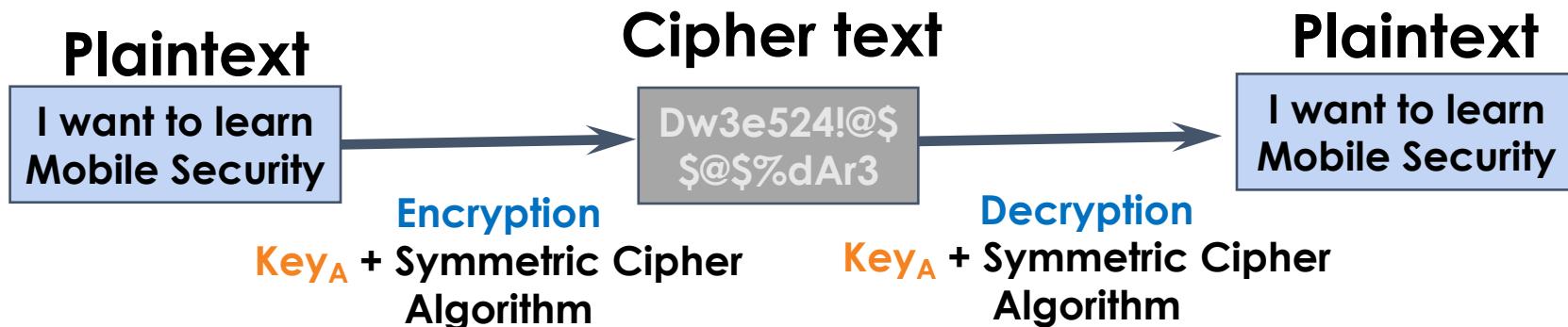
- Access Control
- Cryptography
 - Symmetric vs Asymmetric Keys
 - Encryption
 - Hashing
 - Digital Signatures
 - Digital Certificates
 - HMAC
- Data-Centric safeguards
- Network Segregation



Cryptography

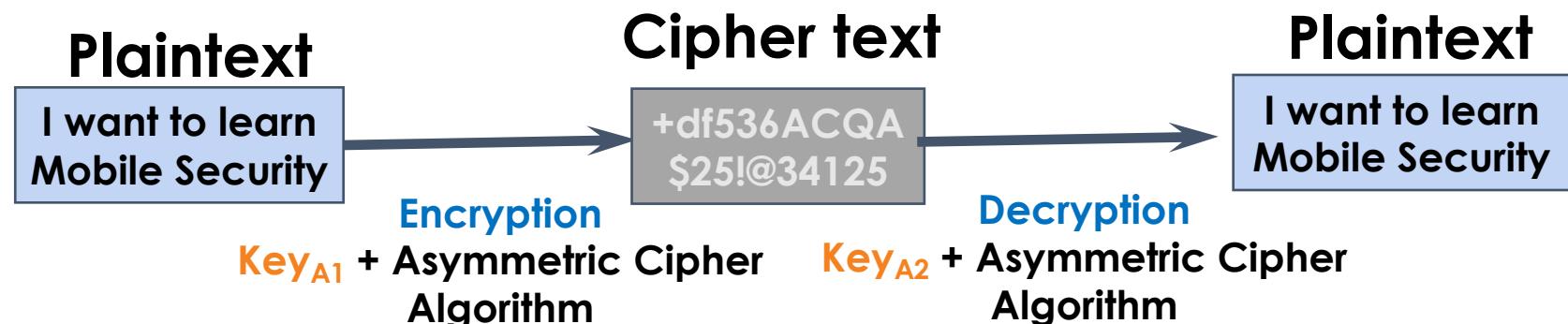
The science of encryption and decryption of messages

- Using Symmetric Key (Key_A)



- Using Asymmetric Keys ($Key_{A1} + Key_{A2}$)

Key pair:
public key,
private key





Asymmetric Encryption (Confidentiality Protection)

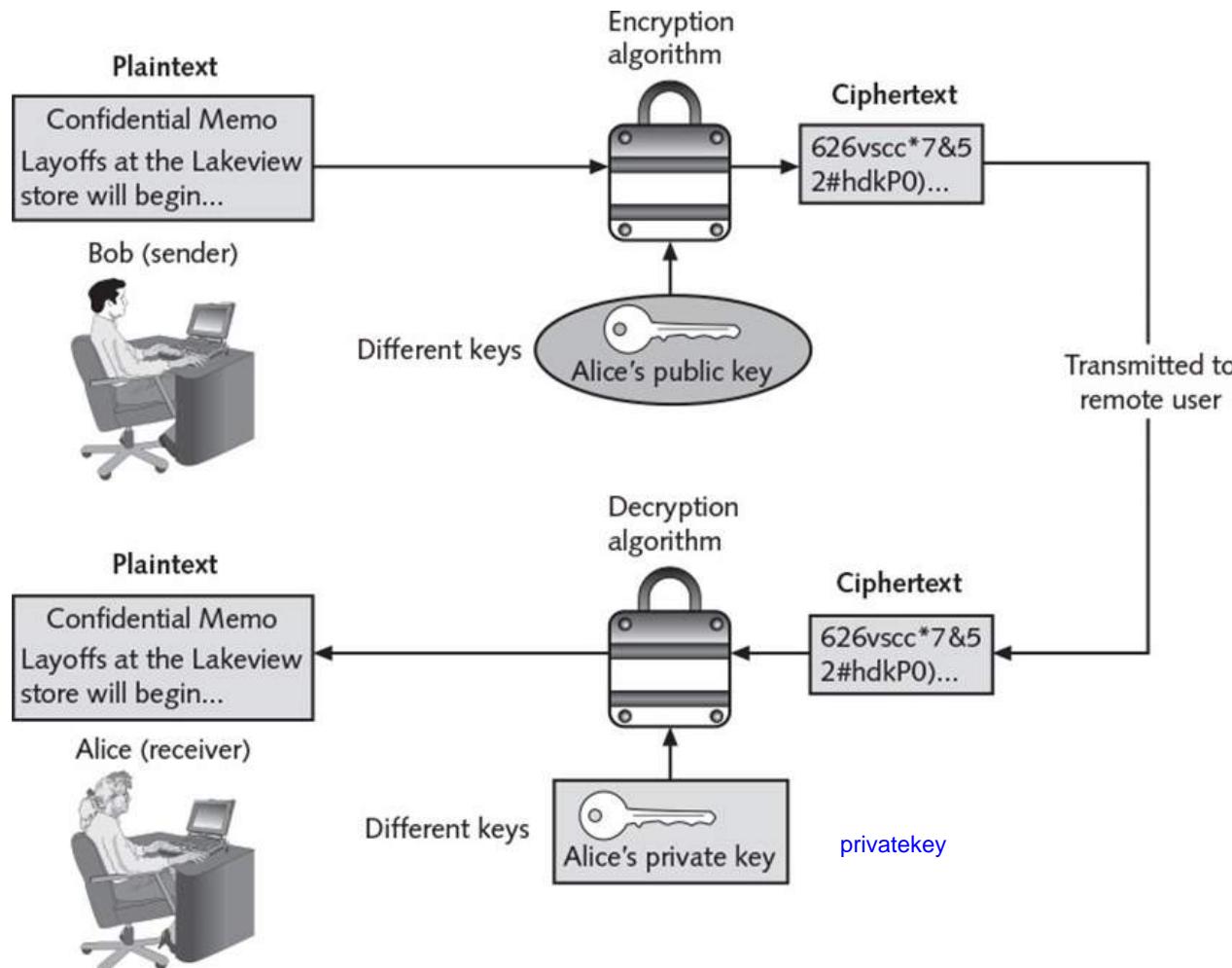


Figure 5-8 Asymmetric (public key) cryptography

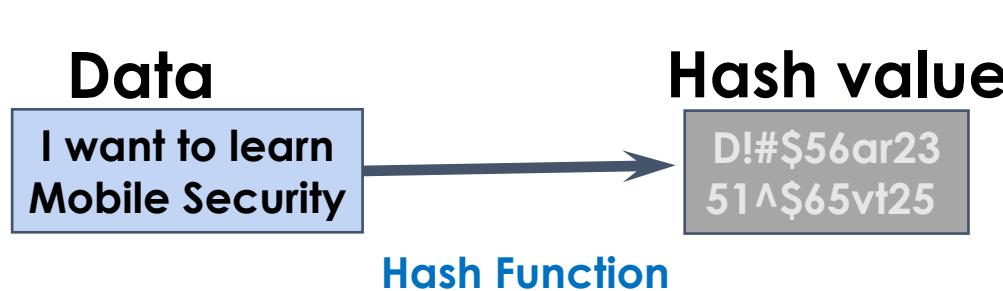
Source: Guide to Network Security Fundamentals by Mark Ciampa



Hash Function

A function that can be used to map data of arbitrary size to **data of fixed size**. The values returned by a hash function are called hash values, hash codes, digests or simply hashes.

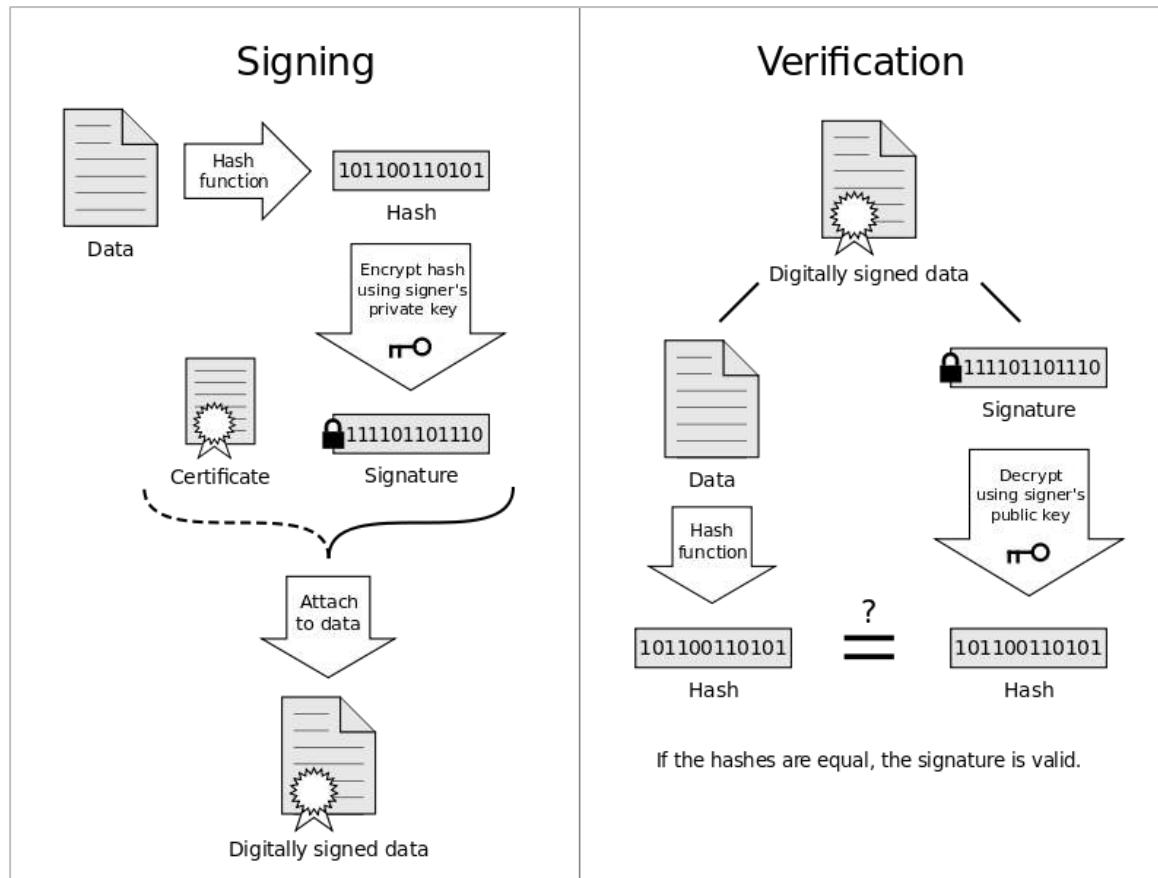
- Provides a **non-reversible** calculation
- Produces a different hash value if there is any change in the input data
- Used to verify the **integrity** of data.





Digital Signature

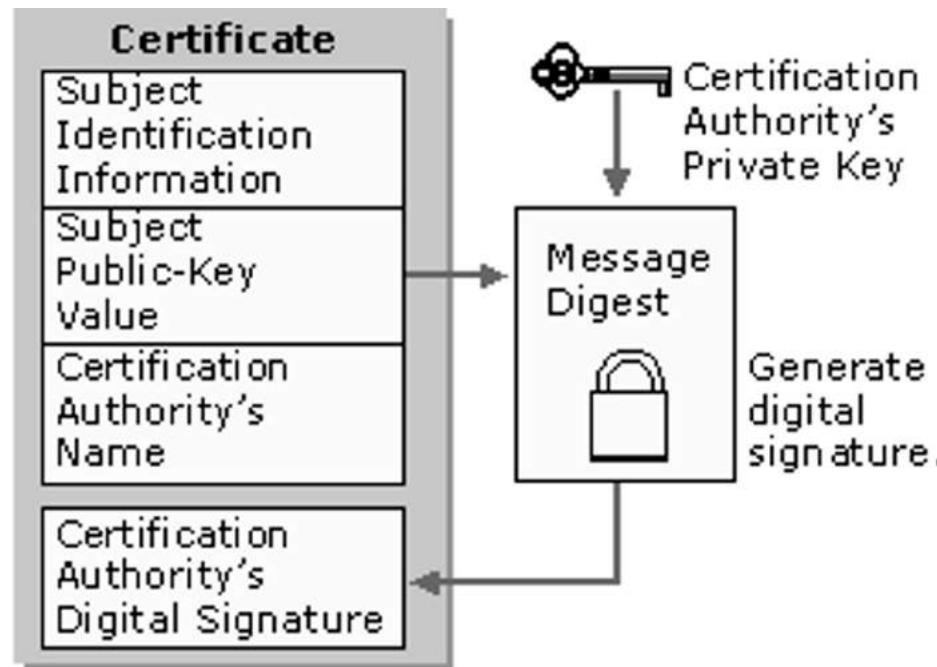
A mathematical scheme for demonstrating the authenticity of digital messages or documents. A valid digital signature gives a recipient reason to believe that the message was created by a known sender (**authentication**), that the sender **cannot deny** having sent the message (**non-repudiation**), and that the message was **not altered in transit** (**integrity**).





Digital Certificate

Also known as **public key certificate**. An electronic document used to prove the **ownership of a public key**. Information includes the **key**, information about the **identity of its owner (called the subject)**, and the **digital signature of an entity that has verified the certificate's contents (called the issuer)**.





Resisting Attacks

- **Access Control**
- **Cryptography**
 - Symmetric vs Asymmetric Keys
 - Encryption
 - Hashing
 - Digital Signatures
 - Digital Certificates
 - HMAC
- **Data-Centric safeguards**
- **Network Segregation**



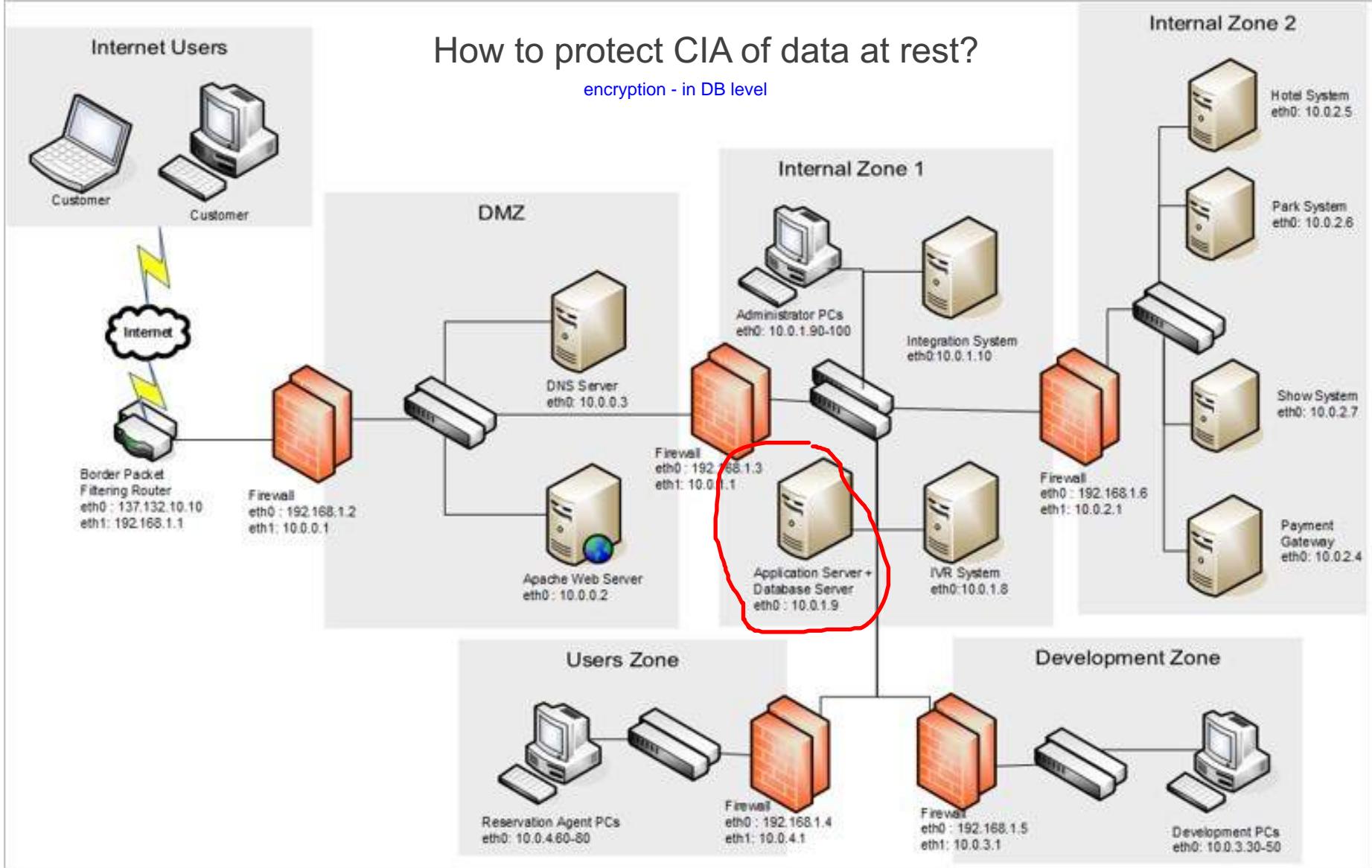
Data-Centric Safeguards

- Protect the CIA of data at **all times**, throughout its lifecycle
 - Data at rest (in storage)
 - Data in motion (being transmitted)
 - Data in use





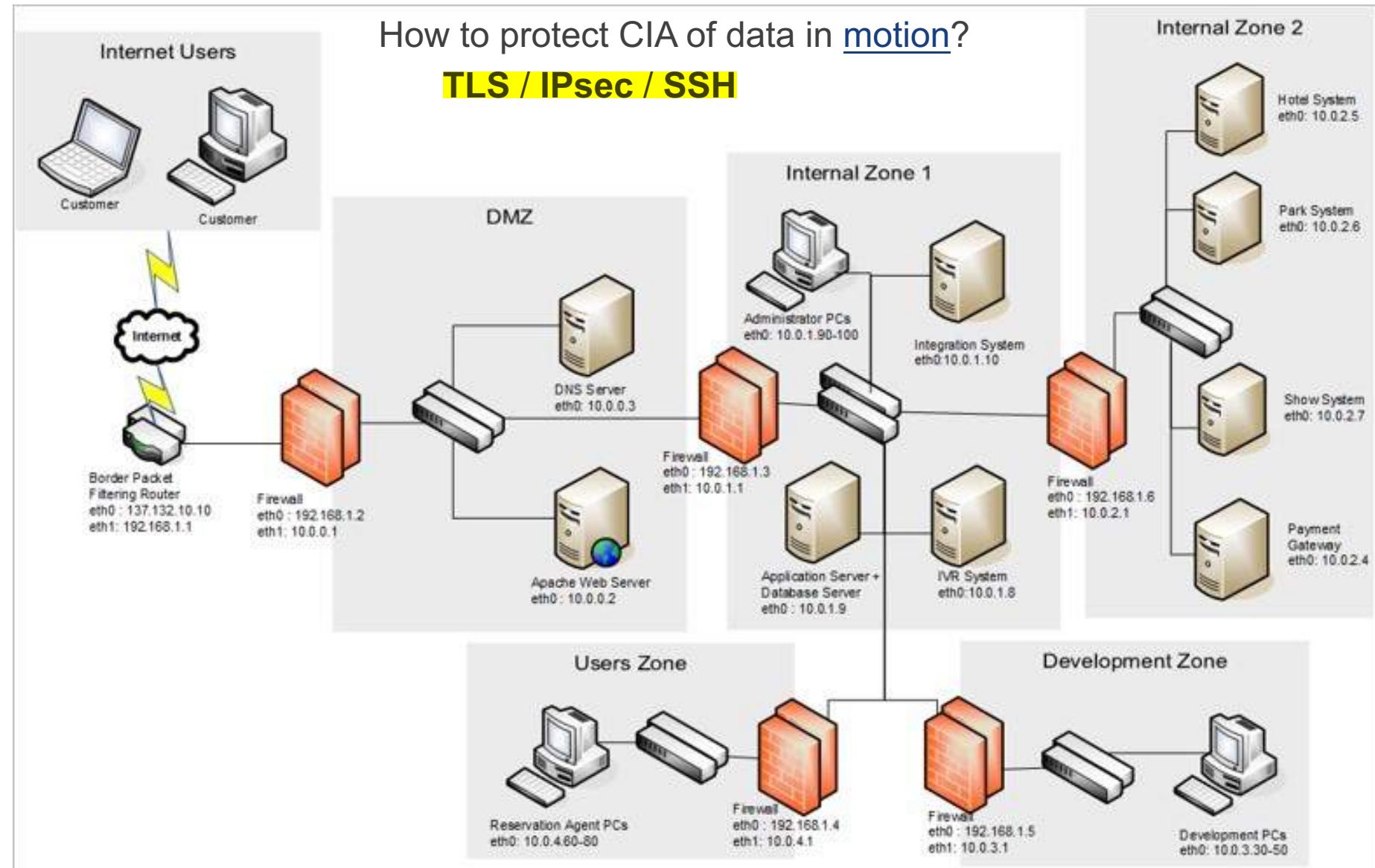
Where is data at rest?



Where is data in motion?

How to protect CIA of data in motion?

TLS / IPsec / SSH

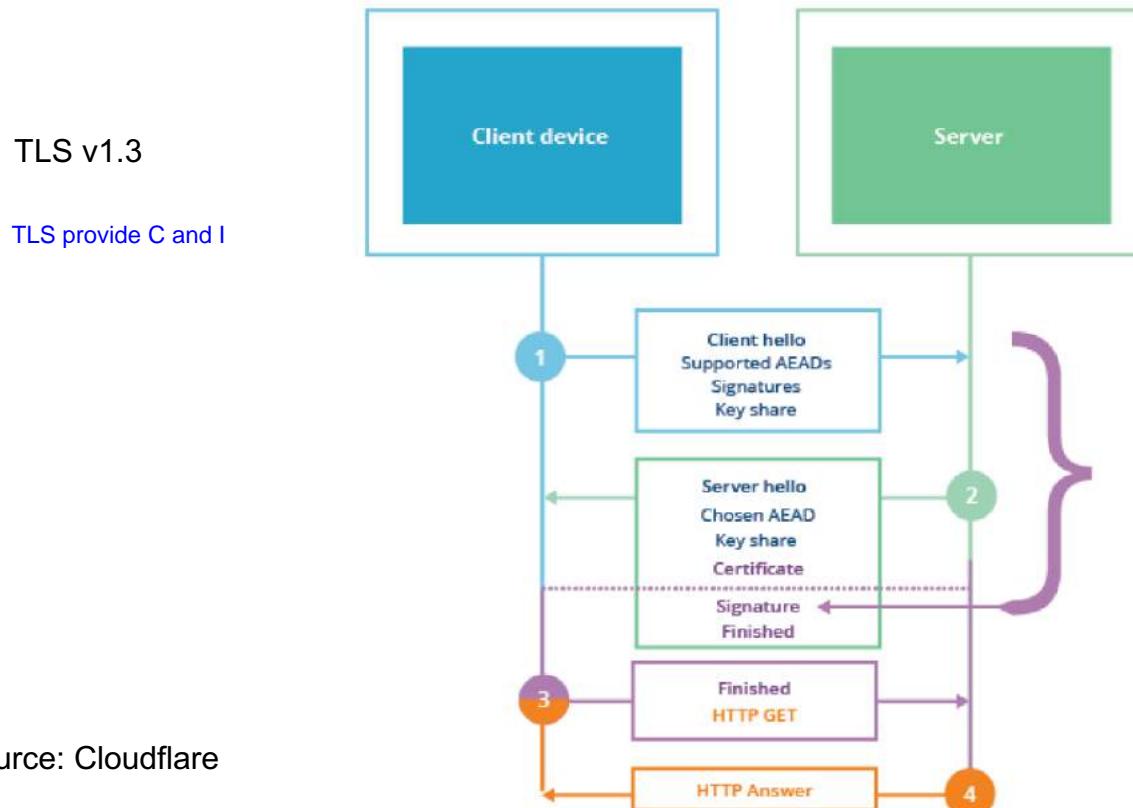




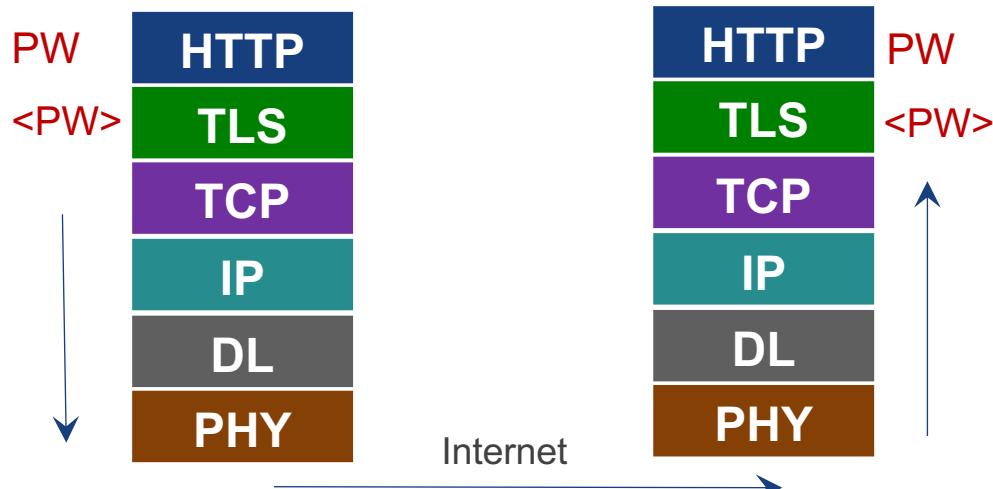
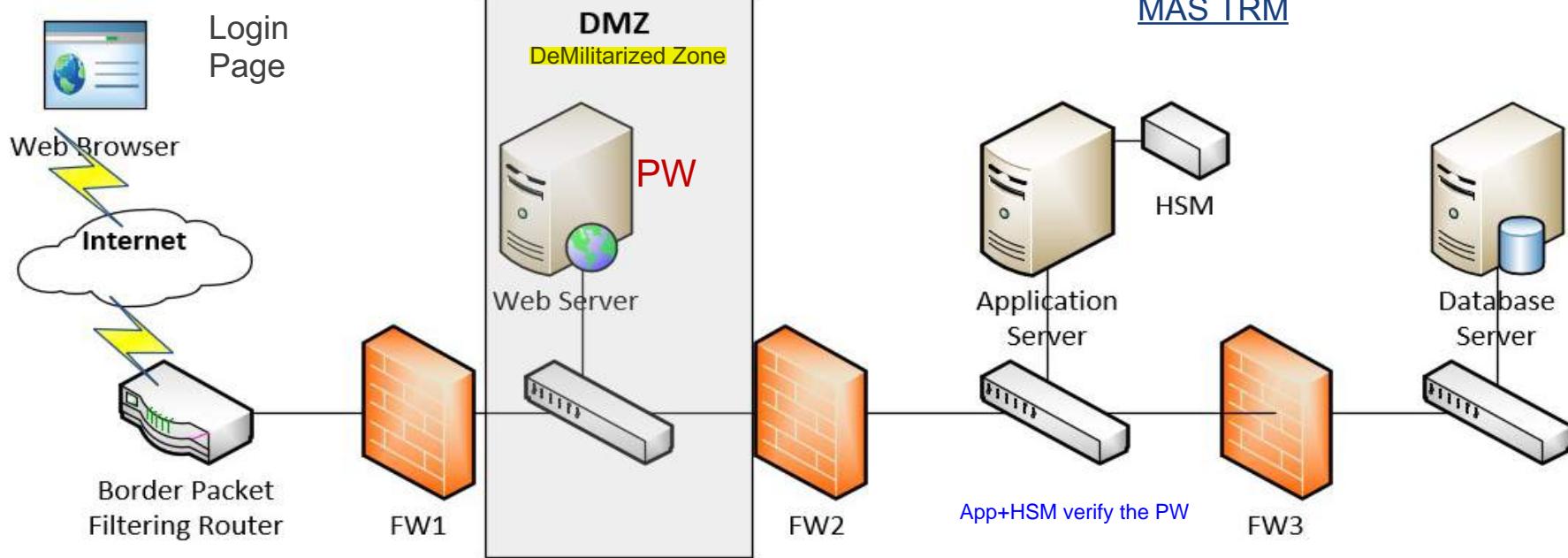
Transport Layer Security (TLS)

Transport Layer Security (TLS) and its predecessor, **Secure Sockets Layer (SSL)**, both referred to as 'SSL', are cryptographic protocols that provide communication security over a computer network. Protects data in transit.

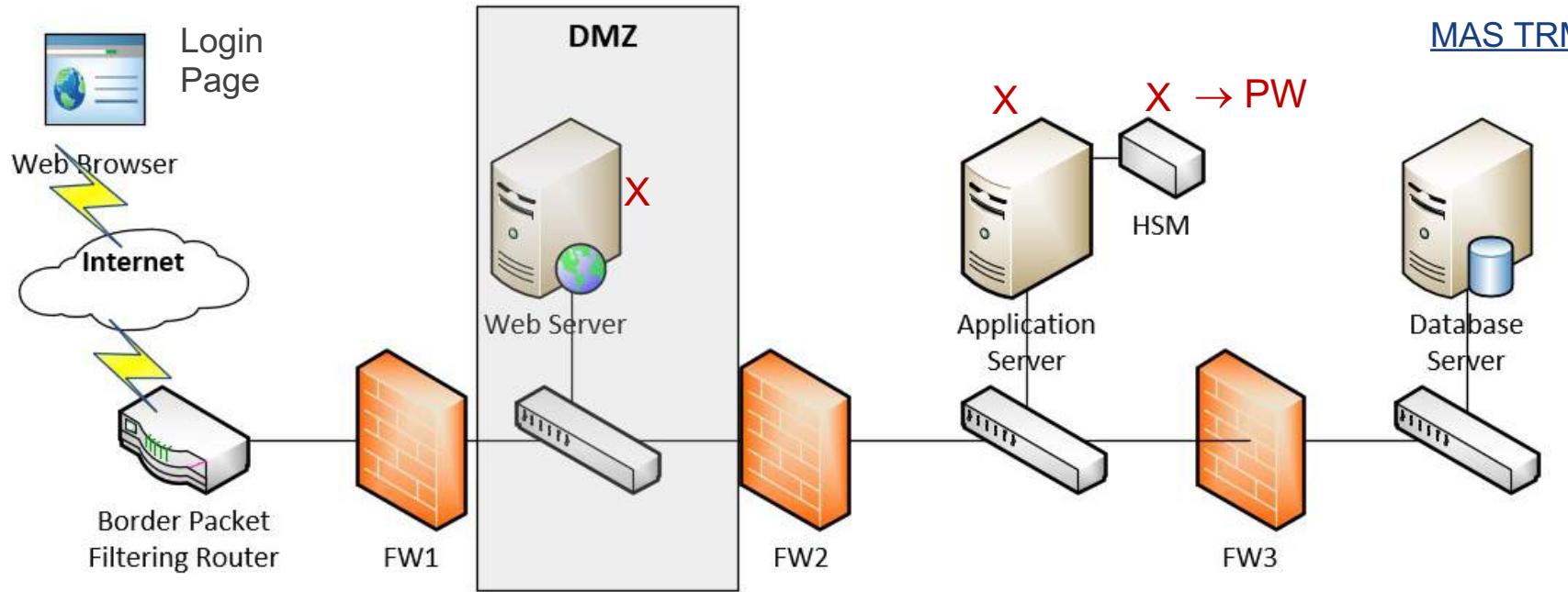
Protection (confidentiality, integrity) is only between TLS client and TLS server, may not be end-to-end.



TLS – Scope of Protection



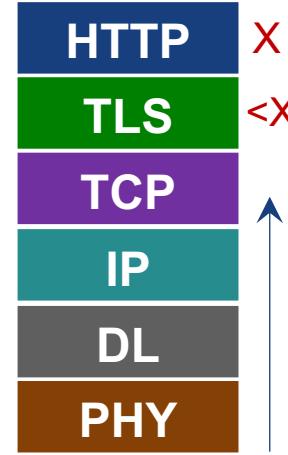
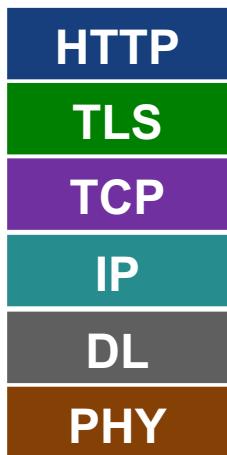
End to End Encryption – possible solution



**X=RSA(PW, Nonce)
with bank's pub key**

TLS is symmetric

<X>



Internet

Sensitive data (PW) remains encrypted throughout, from the point of "production" to the point of "consumption".

end to end data confidentiality: encryption in app level: asymmetric cipher

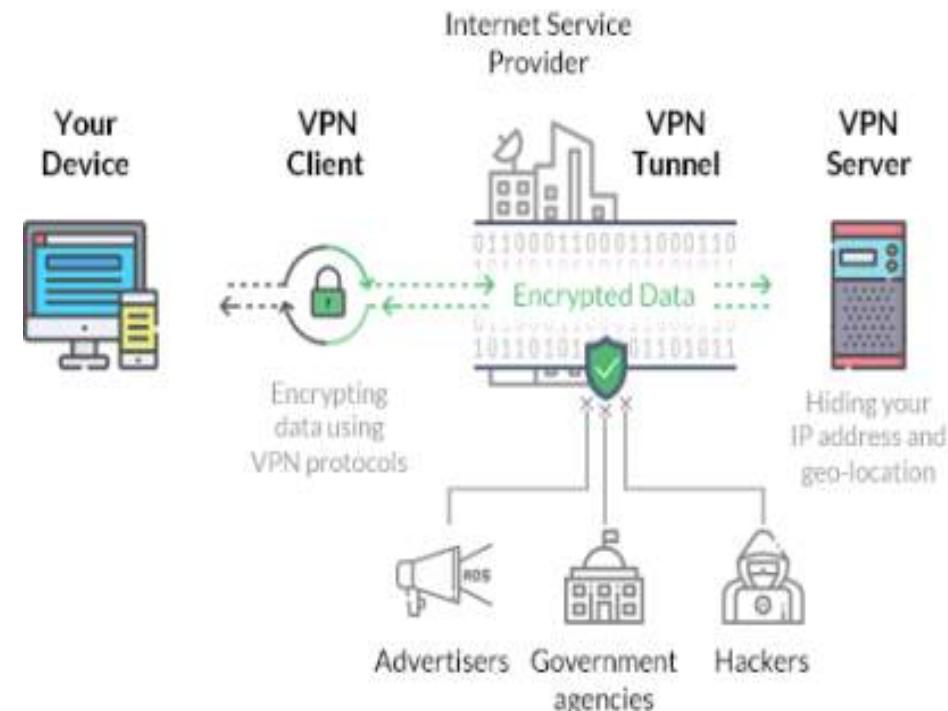
- Refer to the Activity-TLS sheet provided





Virtual Private Network (VPN)

- A secure connection (“tunnel”) is established between **the VPN client** and **the VPN server**, over public networks e.g. Internet
- Any data going through that tunnel is encrypted, and – therefore – unreadable to anyone who is outside the tunnel. **Data Confidentiality is preserved in transit.**
- Data **integrity** is also verified.
- Traditionally, **IPsec standard is used to protect the communications.**
- In SSL VPN, **TLS is the standard used** to protect the communications.



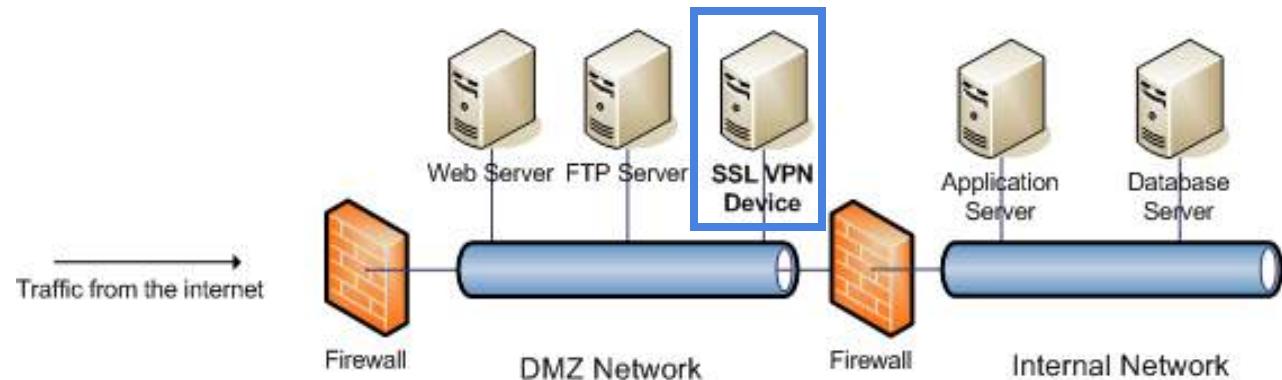
<https://www.cactusvpn.com/beginners-guide-to-vpn/what-is-a-vpn-server-how-does-a-vpn-server-work/>



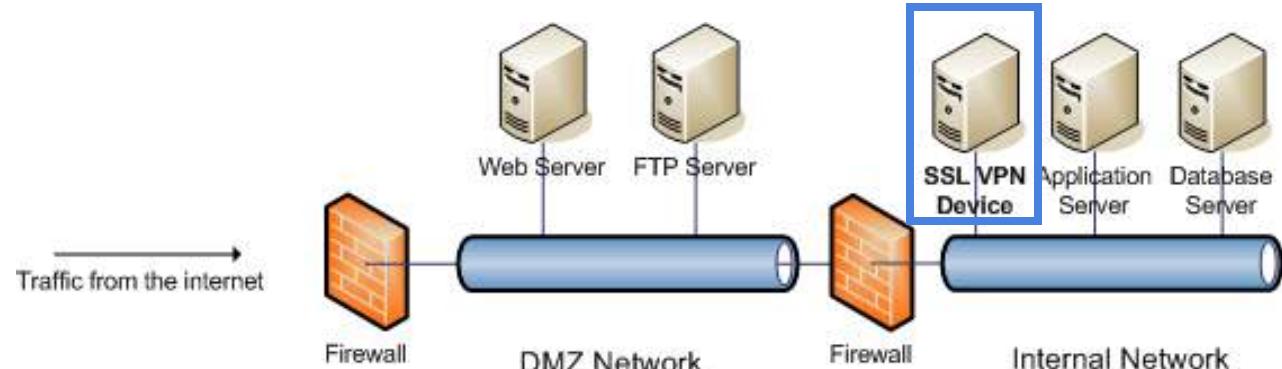
Discussion

Discuss the **pros and cons** of placement of the VPN Server:

Scenario A (DMZ)

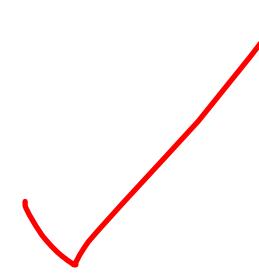


Scenario B (Internal)





Placing the SSL VPN in a DMZ



- **Pros**

- All untrusted parties are stopped in the DMZ
- Using a DMZ offers a layered approach to security

- **Cons**

- **SSL Decryption keys maintained in an unsafe environment**
- Decryption performed in an insecure area



Placing the SSL VPN onto an Internal Network

- **Pros:**

- Decryption of SSL-encrypted traffic is performed in the secure back office
- SSL keys are stored on a secure network

- **Cons:**

- Undermines firewall infrastructure
- Network based IDSs on DMZ rendered ineffective
- Unfettered access to the internal network



Quick Quiz

- To protect confidentiality of data at rest use:
 - Encryption
- To protect integrity of data at rest use:
 - HMAC digital Signature
- To protect confidentiality of data in transit use:
 - TLS/IPSEC/SSH
- To protect integrity of data in transit use:
 - TLS/IPSEC/SSH
- To protect confidentiality of data in transit, **end-to-end** use:
 - ENCRYPTION AT APP LEVEL



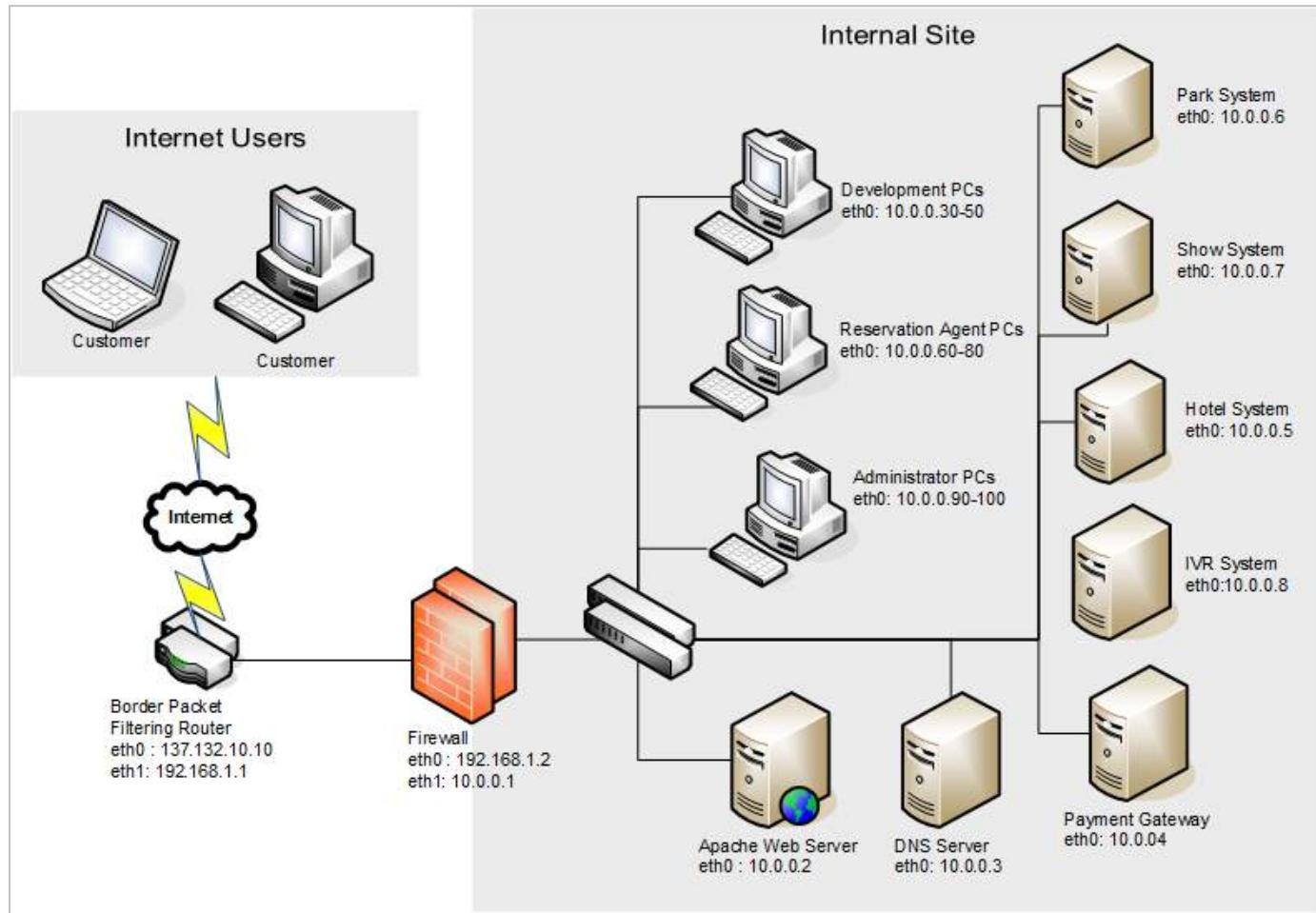
Resisting Attacks

- **Access Control**
- **Cryptography**
 - Symmetric vs Asymmetric Keys
 - Encryption
 - Hashing
 - Digital Signatures
 - Digital Certificates
 - HMAC
- **Data-Centric safeguards**
- **Network Segregation**



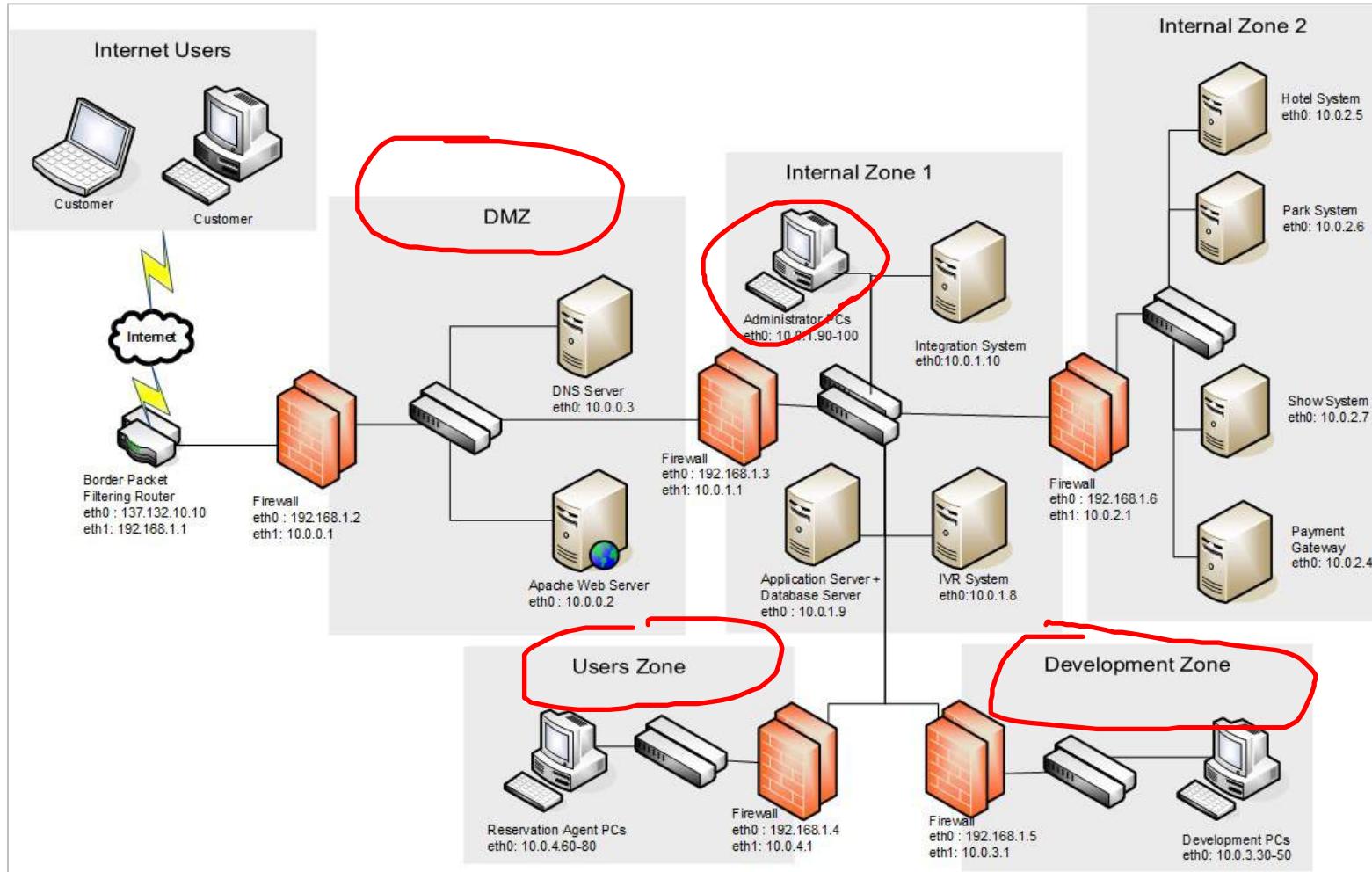
Network Segregation

No Network Segregation (flat network)





Network Segregation – 1st Attempt



How can it be improved?



Out of Band Network

- Segregated LAN for logging
- Dedicated device and segregated network for system administration

Out of Band Network != BAU network = need speareted network card



Detecting Attacks

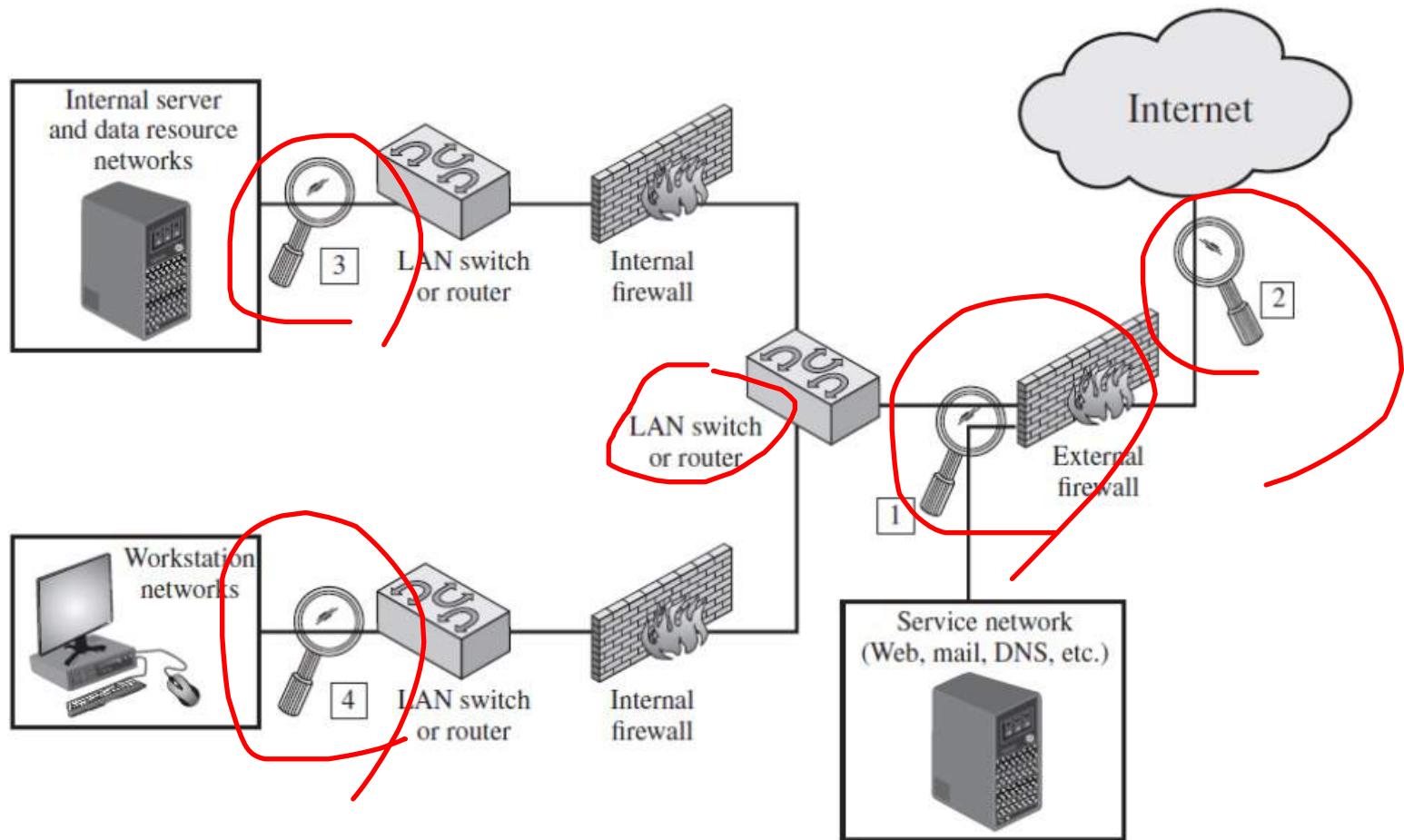
- Detecting Data Tampering
 - Digital Signature, HMAC
- Intrusion Detection
 - Can detect attack as it occurs
 - IDS systems use different methodologies for monitoring for attacks
 - Can be installed on either local hosts or networks
 - An extension of IDS is an intrusion prevention system (IPS)



Intrusion Detection Systems

- Types of IDS - two basic types of IDS exist
- **Host intrusion detection system (HIDS)**
 - Installed on each system needing protection
 - Monitors:
 - System calls and file system access
 - Can recognize unauthorized Registry modification
 - Host input and output communications
 - Detects anomalous activity
- **Network intrusion detection system (NIDS)**
 - Watches for attacks on the network
 - NIDS sensors installed on firewalls and routers

NIDS Sensor Deployment



Computer Security: Principles and Practice: William Stallings, Lawrie Brown



Reacting to Attacks

- **Revoke Access**
- **IDPS**
- **Block data exfiltration**



Recovering from Attacks

- Restoration
 - Maintain data backups
 - Tested
 - Offline
 - WORM
 - Golden Images
 - Securely stored

immutable infrastructure - all running instance - vm/containers



Mitigation Controls – Another Exercise

Threat	Vulnerability	Mitigation controls (illustrative)
Spoofing	<ul style="list-style-type: none">Lack of/Broken authen.Weak PW.1FA	MFA
Tampering Preserve Integrity	<ul style="list-style-type: none">Lack of integrity “protection”.Poor AC	Digital Signature/HMAC
		Access control
Repudiation	Lack of digital sig or any other non-rep mech	Digital signature
Information Disclosure	Poor AC Cleartext (i.e. Lack of encryption).	Encryption for data - static TLS/IPsec - data - transition Access Control - 1st layer
Denial of Service (DoS)	<ul style="list-style-type: none">No throttling.Low capacity comms pipe	rate limiting 3rd party dos service: distribute the traffic to other servers
Elevation of Privilege	Process running with root privileges or excessive privileges	<ul style="list-style-type: none">Run processes with least privilegeSegregation of duties, Need to Know



SECURE DESIGN PRINCIPLES



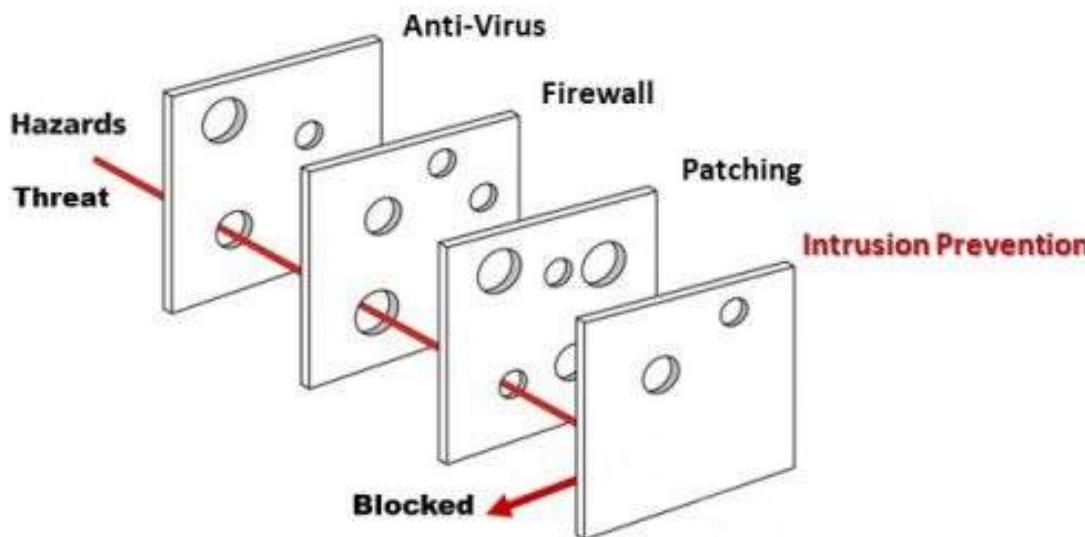
Secure Design Principles

- **Defence-In-Depth**
- **Least Common Mechanism**
- **Least Privilege**
- **Complete Mediation**



Defence-in-Depth

- **Multiple layers** of protection
 - If one layer is breached, other layers will provide protection
- E.g. Access control and encryption to protect **confidential data**



Graphic source: <https://tedangevaare.nl/useful-standards/>



Least Common Mechanism



- **Minimize the number of protection mechanisms common to multiple users**
 - Because shared access paths can be sources of unauthorized information exchange
- **e.g.**
 - **Segregated LAN** for logging
 - **Dedicated device** and segregated network for system administration
 - **Different crypto keys** for different purposes
 - Compartmentalization: Different functions based on user roles instead of one function used by all roles



Least Privilege

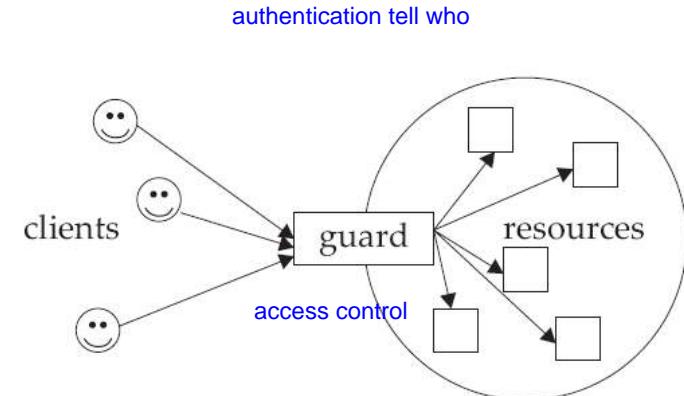
- **Minimum permissions** for the **minimum period of time** required to complete a task.
- When vulnerabilities are exploited, least privilege help limit the magnitude of damage that can be inflicted.
- E.g.
 - Application server requires access to the network, read access to a database table and ability to write to a log file.
 - Grant only these permissions.
 - Never grant superuser permissions.



Complete Mediation

- **Every request by a subject to access an object in a computer system must undergo a valid and effective authorization procedure**
 - ==> **do not cache authorization decisions**

- Intent
 - Design a system so that all access by clients to resources is mediated by a guard which enforces a security policy [complete mediation principle]
- Consequences
 - Isolates resources
 - Loosens coupling between security policy and resources
 - Degrades performance
- Implementations
 - Guard must be non-bypassable
 - Guard must be incorruptible
 - Assurance must be provided



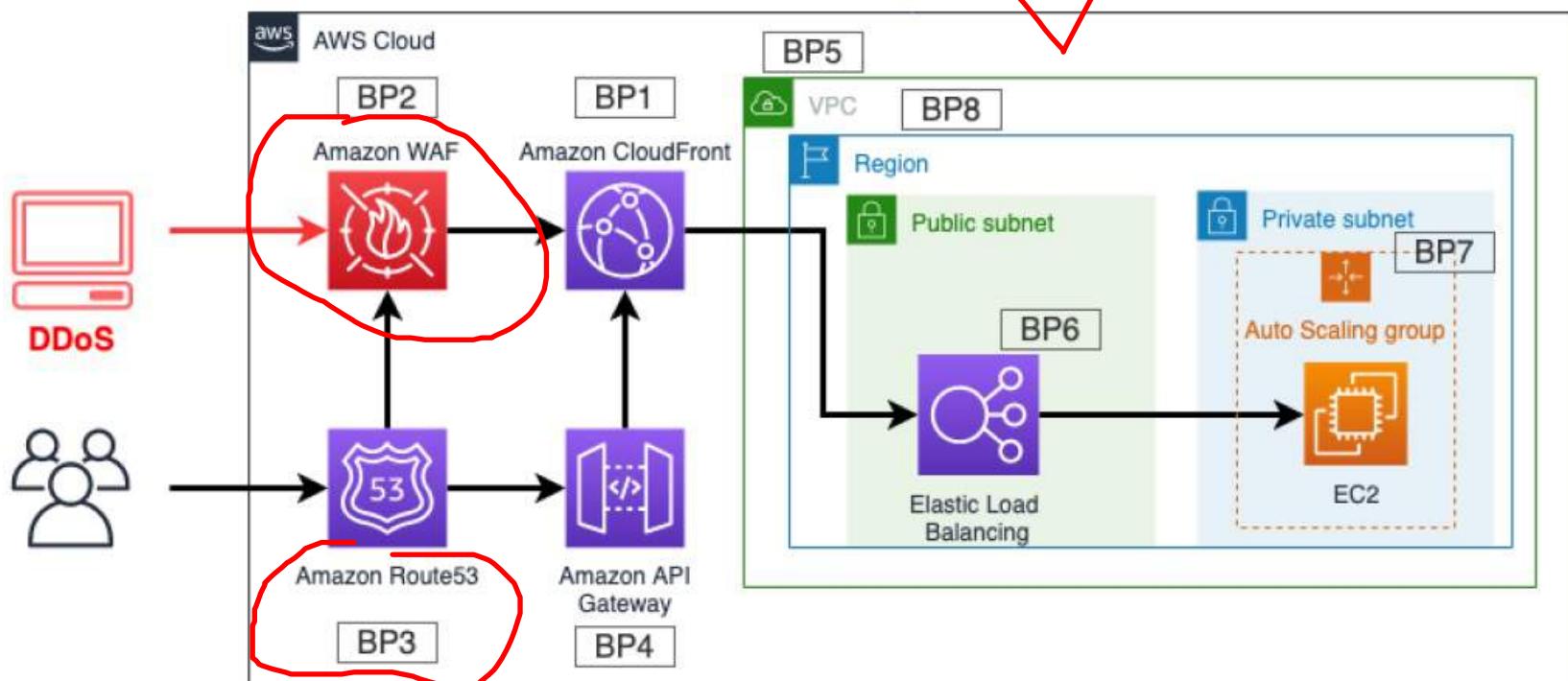
Reference: Security Design Patterns, Open Group,
<https://www2.opengroup.org/ogsys/catalog/g031>



REFERENCE ARCHITECTURES

AWS: DDoS-Resilient Reference Architecture

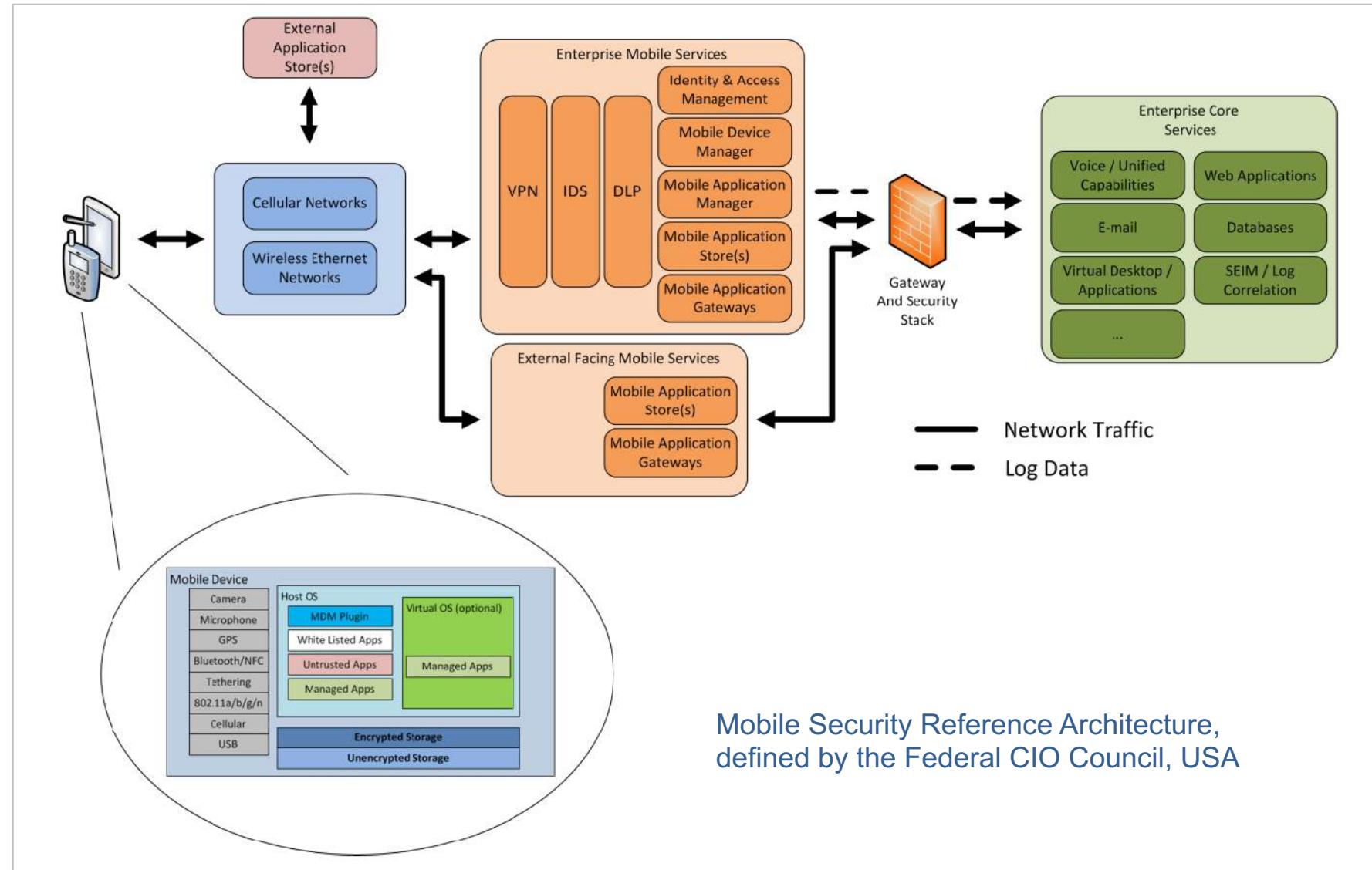
- AWS infrastructure is DDoS-resilient and has automatic mitigation
- A reference architecture and guidance are provided for user applications



Source: <https://docs.aws.amazon.com/whitepapers/latest/aws-best-practices-ddos-resiliency/aws-best-practices-ddos-resiliency.pdf>



Mobile Security - Conceptual Architecture



Source: <https://s3.amazonaws.com/sitesusa/wp-content/uploads/sites/1151/2016/10/Mobile-Security-Reference-Architecture.pdf>



Mobile Infrastructure : Some Architecture Components

- Mobile Device Management (MDM)
 - primary mechanism for technical enforcement of the organization's policies and procedures
- Mobile Application Management (MAM)
 - provides in-depth distribution, configuration, data control, and life-cycle management for specific applications installed on a mobile device



Security standard for Cloud Computing

- **Multi-Tier Cloud Security**
 - World's first cloud security standard that covers multiple tiers
 - to help businesses understand cloud computing certifications provided by different cloud service providers (CSPs),
 - mandatory for CSPs participating in bulk tenders from the Government.

Security standard for Cloud Computing

- Multi-Tier Cloud Security (MTCS) standard has been defined by IMDA in Singapore (SS 584)

MTCS Levels

Level	Overview	Security Control Focus	Typical Usage
1	Designed to be low cost with a minimum of required controls	Baseline security controls – “security 101”	<ul style="list-style-type: none">• Hosting web site• Test & Development• Simulation• Non-critical biz apps
2	Address the needs of most organizations that are concerned about data security	A set of more stringent security controls required to address security risks & threats to data	<ul style="list-style-type: none">• The majority of cloud usages.• More critical biz apps
3	Designed for regulated organizations with specific requirements & are willing to pay for more stringent security requirements	Additional set of security controls are necessary to supplement & address security risks & threats in high-impact information systems using cloud services	<ul style="list-style-type: none">• Hosting applications & systems with sensitive information & regulated systems

Source: <https://www.imda.gov.sg/industry-development/infrastructure/ict-standards-and-frameworks/mtcs-certification-scheme/MTCS-Certification-Scheme>



Activity



- **RM Practice Exercise.**
 - Read the IROne Example Scenario and perform risk management.
 - Record your work in the format shown in the table below.

Asset/Asset Group	Potential Threat/Vulnerability pair	Possible Mitigation Controls



Summary

- Security is about ensuring Confidentiality, Integrity and Availability of data and systems
 - Additional security requirements
 - Non-repudiation
 - Authenticity
 - Authorization
 - Accountability
- Risk Management
 - Risk assessment
 - Asset, Threat, Vulnerability, Consequence
 - Risk = function (Likelihood, Impact)
 - Risk Treatment



Project Assignment

Architecting the **Student Health System** **(Exercise 3)**

Architecting Software Solutions - Project Assignment PAGE 1 OF 12

Architecting Software Solutions
Project Assignment

Architecting Student Healthcare System



©2011-20 NUS. The contents contained in this document may not be reproduced in any form or by any means, without the written permission of ISS and NUS, other than for the purpose for which it has been supplied.

ATA/S-ARCHSS/OM/PROJECT ASSIGNMENT V1.0 © 2011-20 NUS. ALL RIGHTS RESERVED NUS National University of Singapore ISS Institute of Software Systems

S-ARCHSS Project Assignment (Case Study Templates Exercise 3)

Architecting Software Solutions Project Assignment

Exercise 3. Design Secure Architecture

Name: _____

Date: _____

1. Risk Assessment

Note: Include at least 2 distinct "crown jewel" assets and 2 distinct "stepping stone" assets in your risk assessment.

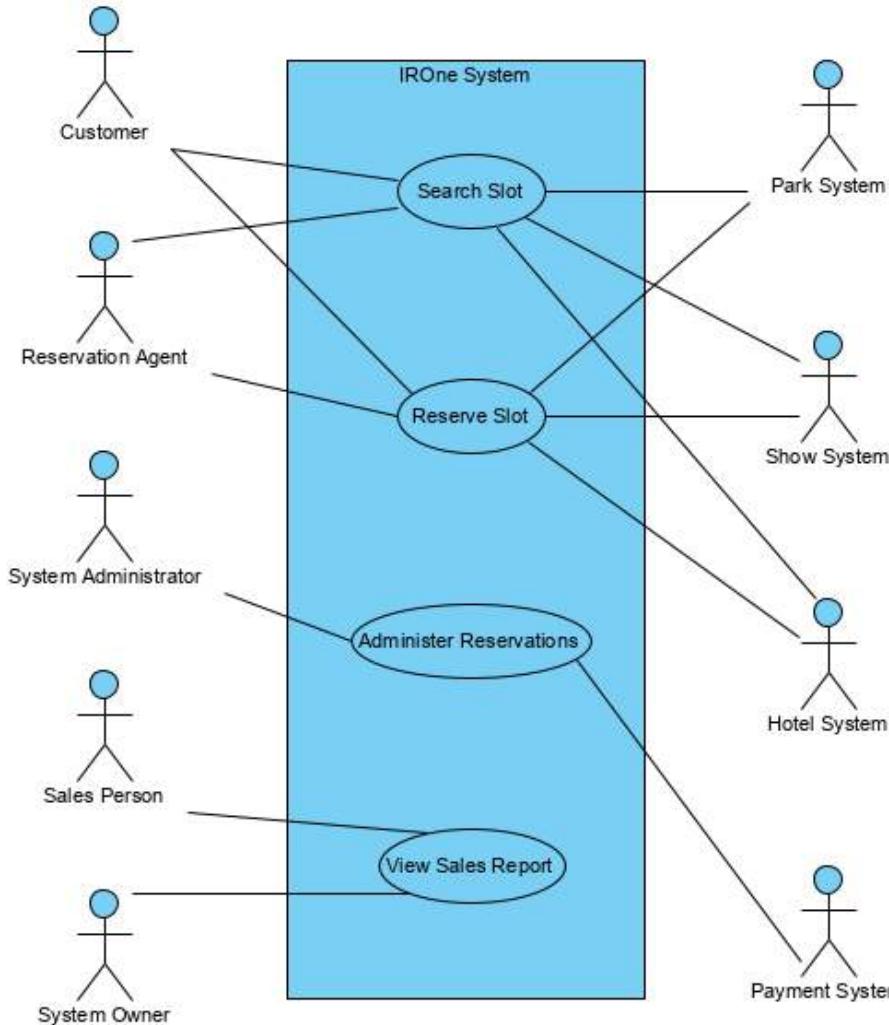
Asset/Asset Group	Potential Threat/Vulnerability pair	Possible Mitigation Controls



APPENDIX A

(Slides from earlier modules)

Task 4 – Outline Functional Requirements (identify and capture functional requirements)



System Context





Task 7 – Capture Functional Requirements (detailed use cases)

Reserve Slot Use Case

Description	Allows the user to reserve for hotel, park or show slots
Primary Actors	Customer, Reservation Agent
Secondary Actors	Hotel System, Park System, Show System, Payment System
Main Flow of Events	<ol style="list-style-type: none">1. The user opts to reserve the slot by selecting the Reserve option on the Search Slot screen2. The system displays the Slot Reservation screen3. The user enters his/her personal particulars and payment details4. The system validates the entered details5. The system requests the appropriate amenity system to perform the reservation and stores the reservation details6. The system requests the payment system to effect the payment and stores the payment details7. The system notifies the user on the successful reservation by displaying the details of reservation and payment on the Slot Reservation screen8. The user acknowledges the notification and the system displays the Search Slot screen



Task 7 – Capture Functional Requirements (detailed use cases)

Reserve Slot Use Case (continued)

Alternative Flow of Events	<p>Fail to validate details at step 4: The system prompts the user to enter the details again at step 3</p> <p>Fail to reserve slot at step 5: The system notifies the user of the unsuccessful reservation and displays the Search Slot screen</p> <p>Fail to make payment at step 6: The system notifies the user of the unsuccessful payment and prompts the user to enter the details again at step 3</p>
Pre-conditions	<ul style="list-style-type: none">The user has searched for the slotIROne system has indicated the availability of the slot
Post-conditions	<ul style="list-style-type: none">The reservation is successful and the slot is not available for useThe reservation is unsuccessful and the slot is available for use



Integration Endpoints

Integration Endpoints					
Source	Destination	Protocol	Format	One/Two way	Transformation Needed?
<u>Client System</u>	<u>Web Server</u>	HTTP / HTTPS	HTML / JSP	Two way	No
Web Server	<u>Application Server</u>	HTTP / HTTPS	HTML / JSP	Two way	No
Int Svr IS 15	Show System	JMS REST	XML JSON	Two way	Yes
Application Server	Hotel System	JMS	XML	Two way	Yes
Application Server	Park System	JMS	XML	Two way	Yes





APPENDIX B

(Web Security Best Practices)



Web Security - Best Practices

1. Use Stateful firewalls
 - Keep track of all web tier transmission and protocol sessions
2. Put Web Servers in a demilitarized zone (DMZ)
 - Secure the internet facing web server / reverse proxy in a DMZ using an exterior firewall
3. Drop non-HTTP packets
 - Configure to allow HTTP and HTTP over SSL only to prevent malicious packets intended for back end systems
4. Use HTTP POST rather than HTTP GET
 - HTTP GET reveals URL appended information, allowing sensitive information to be revealed in the URL string
5. Destroy HTTP sessions upon logout and set timeouts
 - Invalidate the session and remove all state upon logout - stale sessions can lead to session hijacking, client side Trojan horses and eavesdropping



Web Security - Best Practices

6. Disallow direct access to the Application Tier
 - Ensure that the application is only accessible via legitimate clients
7. Validate request data
 - Validate all requests, responses and both inbound and outbound data
8. Error reporting
 - Always return an error page or exception specific to the application error and user request.
 - Do not expose remote, system level and naming service specific exceptions which expose the weakness in the application, allowing hackers to design potential attacks



Web Security - Best Practices

9. Secure the Pipe

- Make sure that the session and data exchanged between server and client are confidential and tamper proof

10. Secure Administrative Communications

- Make sure all administration tasks are done using encrypted communication

11. Harden the Server Operating System

- Must not run any unneeded service that may provide an avenue for attacker to compromise

12. Obfuscate Code and Encrypt Application Properties

- Avoid code misuse and reverse reengineering
- Ensure that application specific properties stored on local disks are encrypted to protect against unauthorized access



Web Security - Best Practices

13. Set up an Intrusion Detection System (IDS)

- Use IDS to detect suspicious acts, abuses and unsolicited use

14. Deny outbound traffic directly from Web Servers

- Disallow outbound traffic directly generated out from the IP address of the Web Servers running in the DMZ

15. Audit all security-relevant use

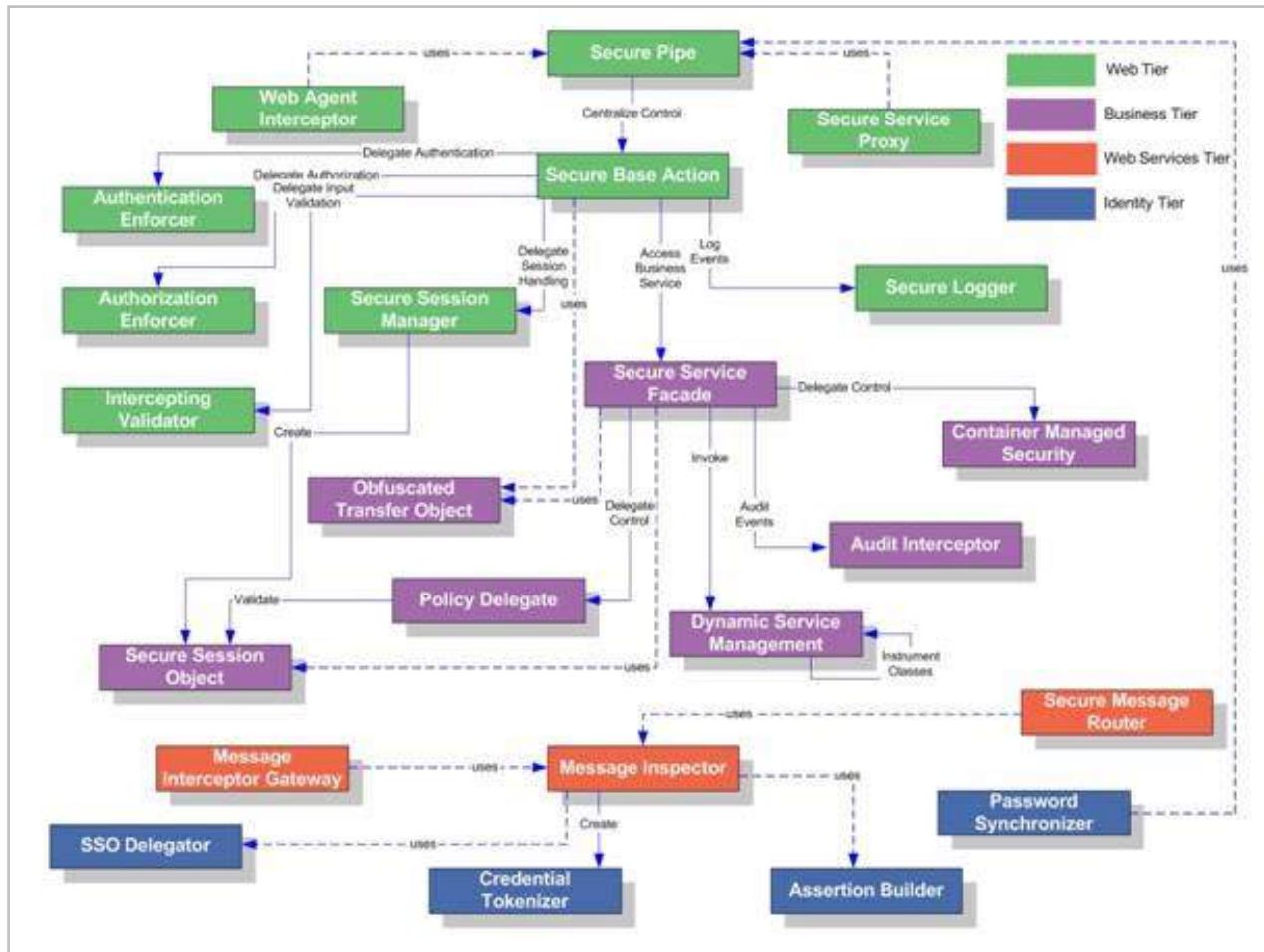
- Securely log, audit and timestamp all relevant application level events
- Audit trails for all identified user level sessions and actions should be captured for non-repudiation of user actions, if digital signature or other more robust non-repudiation mechanisms are not feasible.



APPENDIX C

(Core Web Security Patterns)

Core Web Security Patterns



core SECURITY PATTERNS

CHRIS STEEL • RAMESH NAGAPPAN • RAY LAI

- Authentication Enforcer
- Authorization Enforcer
- Interceptor Validator
- Secure Base Action
- Secure Logger
- Secure Pipe
- Secure Service Proxy
- Secure Session Manager
- Intercepting Web Agent



Authentication Enforcer

- Problem
 - Verify that each request is from an authenticated entity - since different classes handle different requests, authentication code is replicated in many places and the authentication mechanism cannot easily be changed
- Solution
 - Container Managed Strategy
 - Declarative Security
 - Deployment Descriptors and Annotations
 - Authentication Mode
 - HTTP Basic Authentication
 - Form Based Authentication
 - Digest Based Authentication
 - Client Based Authentication

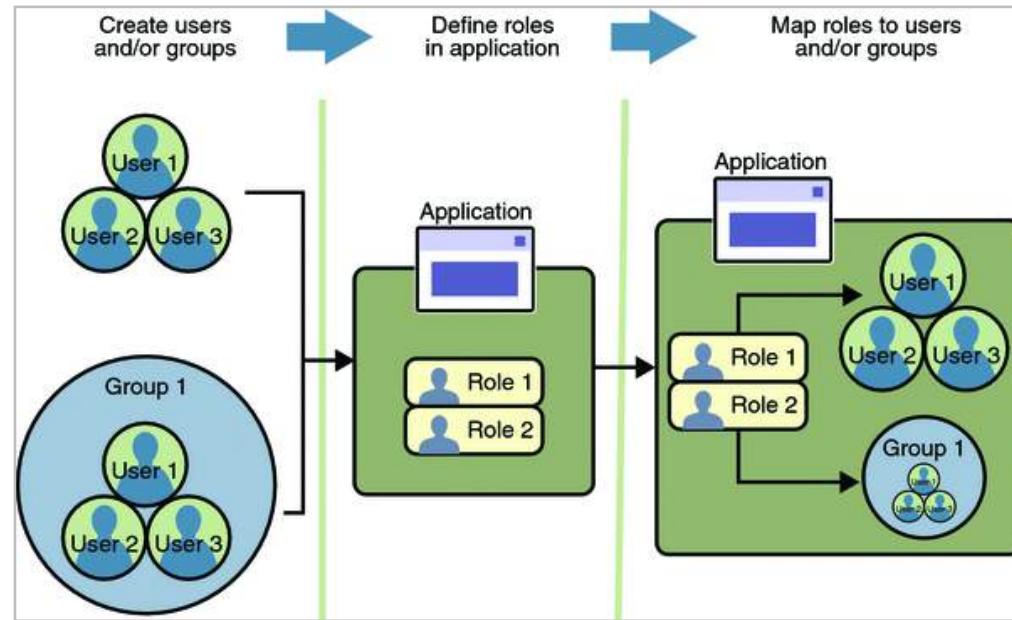


Authorization Enforcer

- Problem
 - Many components need to verify that each request is properly authorized at the method and link level
 - For applications that cannot take advantage of container-managed security, this custom code has the potential to be replicated
- Solution
 - Create an Access Controller that will perform authorization checks using standard Java security API classes

Working with Java Security

- Principal / User
- Group
- Role
- Security Domain or Realm



```
<sun-web-app>
  <security-role-mapping>
    <role-name>DIRECTOR</role-name>
    <principal-name>mcneely</principal-name>
  </security-role-mapping>
  <security-role-mapping>
    <role-name>MANAGER</role-name>
    <group-name>manager</group-name>
  </security-role-mapping>
</sun-web-app>
```



Interceptor Validator

- Problem
 - You need a simple and flexible mechanism to scan and validate data passed in from the client for malicious code or malformed content
 - The data could be form-based, queries or XML content.
- Solution
 - Use an Intercepting Validator to cleanse and validate data prior to its use within the application, using dynamically loadable validation logic
 - Apache Struts - Interceptors
 - Aspect Oriented Programming



Secure Base Action

- Problem
 - You want to consolidate interaction of all security-related components in the Web tier into a single point of entry for enforcing security and integration of security mechanisms
- Solution
 - Use a Secure Base Action to coordinate security components and to provide Web tier components with a central access point for administering security-related functionality



Secure Logger

- Problem
 - All application events and related data must be securely logged for debugging and forensic purposes.
 - This can lead to redundant code and complex logic
- Solution
 - Use a Secure Logger to log messages in a secure manner so that they cannot be easily altered or deleted and so that events cannot be lost
 - Balance between performance and analytical purposes
 - Remote Logger, Secure Data Logger and Secure Log Store

- Problem
 - You need to provide privacy and prevent eavesdropping and tampering of client transactions caused by man-in-the-middle attacks
- Solution
 - Use a Secure Pipe to guarantee the integrity and privacy of data sent over the wire
 - Web Based SSL, Hardware Based Cryptographic Card / SSL Accelerators, SSL/TLS Appliances



Securing the Application Tier

- Audit Interceptor
- Container Managed Security
- Dynamic Service Management
- Obfuscated Transfer Object
- Policy Delegate
- Secure Service Façade
- Secure Session Object

- Problem
 - You want to intercept and audit requests and responses to and from the Business tier
- Solution
 - Use an Audit Interceptor to centralize auditing functionality and define audit events declaratively, independent of the Business tier services
 - Java Messaging Service, EJB Interceptors

- Problem
 - You need a simple, standard way to enforce authentication and authorization in your J2EE applications and don't want to reinvent the wheel or write home-grown security code
- Solution
 - Use Container Managed Security to define application-level roles at development time and perform user-role mappings at deployment time or thereafter.
 - Declarative Authorization `@RolesAllowed ("admin")`
 - Programmatic Authorization:
`EJBContext.getCallerPrincipal()` method



Obfuscated Transfer Object

- Problem
 - You need a way to protect critical data as it is passed within application and between tiers
- Solution
 - Use an Obfuscated Transfer Object to protect access to data passed within and between tiers

- Problem
 - You want to shield clients from discovery and invocation details of security services and to control client interactions by intercepting and administering policy on client requests
- Solution
 - Use Policy Delegate to mediate requests between clients and security services, and to reduce the dependency of client code on implementation specifics of the service framework



APPENDIX D

(Sources of Security Requirements)



Sources of Security Requirements

- **Laws**
 - E.g. PDPA
- **Regulations**
 - E.g. PCI DSS, MAS TRM
- **Organizational Security Policies, Standards, Guidelines**
 - E.g. classified data handling policy
- **Misuse case analysis of functional requirements**





Recall ISO 25010 - Security

Degree to which a product or system protects information and **data** so that persons or other products or systems have the degree of data access **appropriate** to their types and levels of **authorization**

- **Confidentiality**

Degree to which a product or system ensures that **data** are accessible only to those authorized to have access

- **Integrity**

Degree to which a system, product or component prevents unauthorized access to, or modification of, computer **programs** or **data**



Recall ISO 25010 - Security

- Non-Repudiation

Degree to which actions or events can be proven to have taken place, so that the events or actions cannot be repudiated later

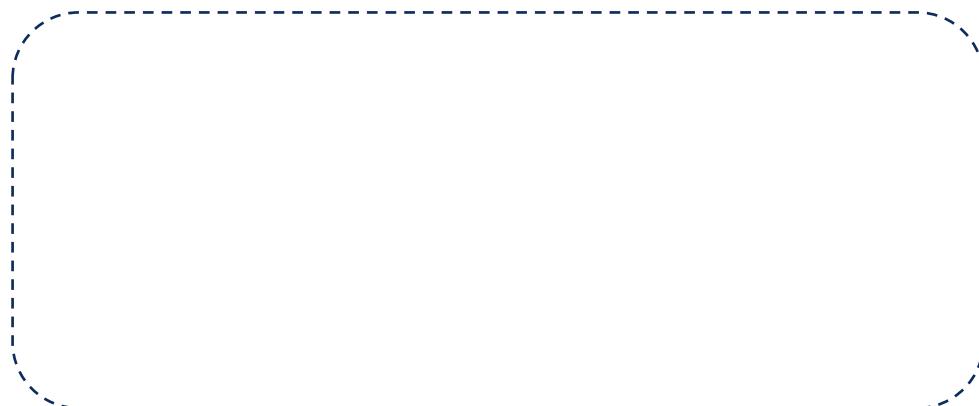
- Authenticity

Degree to which the **identity** of a subject or resource can be **proved** to be the one claimed

- Accountability

Degree to which the actions of an entity can be traced uniquely to the entity

Authorization?





APPENDIX E

(Risk Management)



What is Cybersecurity Risk?

Risk = function (Likelihood, Impact)

Risk is the **Likelihood** of
a **threat** exploiting a particular **vulnerability**
to harm an **asset**
resulting in a business **impact**.



Vulnerability - Examples

Security Objective	Threat	Vulnerability
Authentication	Spoofing	<ul style="list-style-type: none">• Broken authentication.• Weak PW.• 1FA
Integrity	Tampering	<ul style="list-style-type: none">• Lack of integrity assurance.• Poor AC
Non-repudiation	Repudiation	<ul style="list-style-type: none">• Lack of digital sig or any other non-rep mechanism• Insufficient logging & monitoring
Confidentiality	Information Disclosure	Cleartext (i.e. Lack of encryption). Poor AC
Availability	Denial of Service (DoS)	<ul style="list-style-type: none">• No throttling.• Low capacity comms pipe
Authorization	Elevation of Privilege	Process running with root privileges or excessive privileges

For more examples of vulnerabilities, refer to [Common Weakness Enumeration \(CWE\)](#)



Likelihood and Impact

- $\text{Risk} \approx \text{Asset} \times \boxed{\text{Threat} \times \text{Vulnerability}}$

↓
Impact

↓
Likelihood



Documenting the identified risks

Asset/Asset Group	Threat/Vulnerability pair
Customer PII	<p>Threat: sniffing/unauthorized disclosure</p> <p>Vulnerability: transmission in cleartext or stored in cleartext</p>
Web Server	<p>Threat: hacking / unauthorized access</p> <p>Vulnerability: insecure configuration, patching not done in timely manner</p>
.....

Risk Register (partial)



Update the risk register (with risk level)

Asset/Asset Group	Threat/Vulnerability pair	Risk Level
Customer PII	Threat: sniffing/unauthorized disclosure Vulnerability: transmission in cleartext or stored in cleartext	xx
Web Server	Threat: hacking / unauthorized access Vulnerability: insecure configuration, patching not done in timely manner	Yy
.....	



Security Controls

Asset/Asset Group	Threat/Vulnerability pair	Risk Level	Risk Mitigation / Security Controls
Customer PII	Threat: sniffing/unauthorized disclosure Vulnerability: transmission in cleartext or stored in cleartext	xx	
Web Server	Threat: hacking / unauthorized access Vulnerability: insecure configuration, patching not done in timely manner	yy	
.....	



Security Controls – Exercise

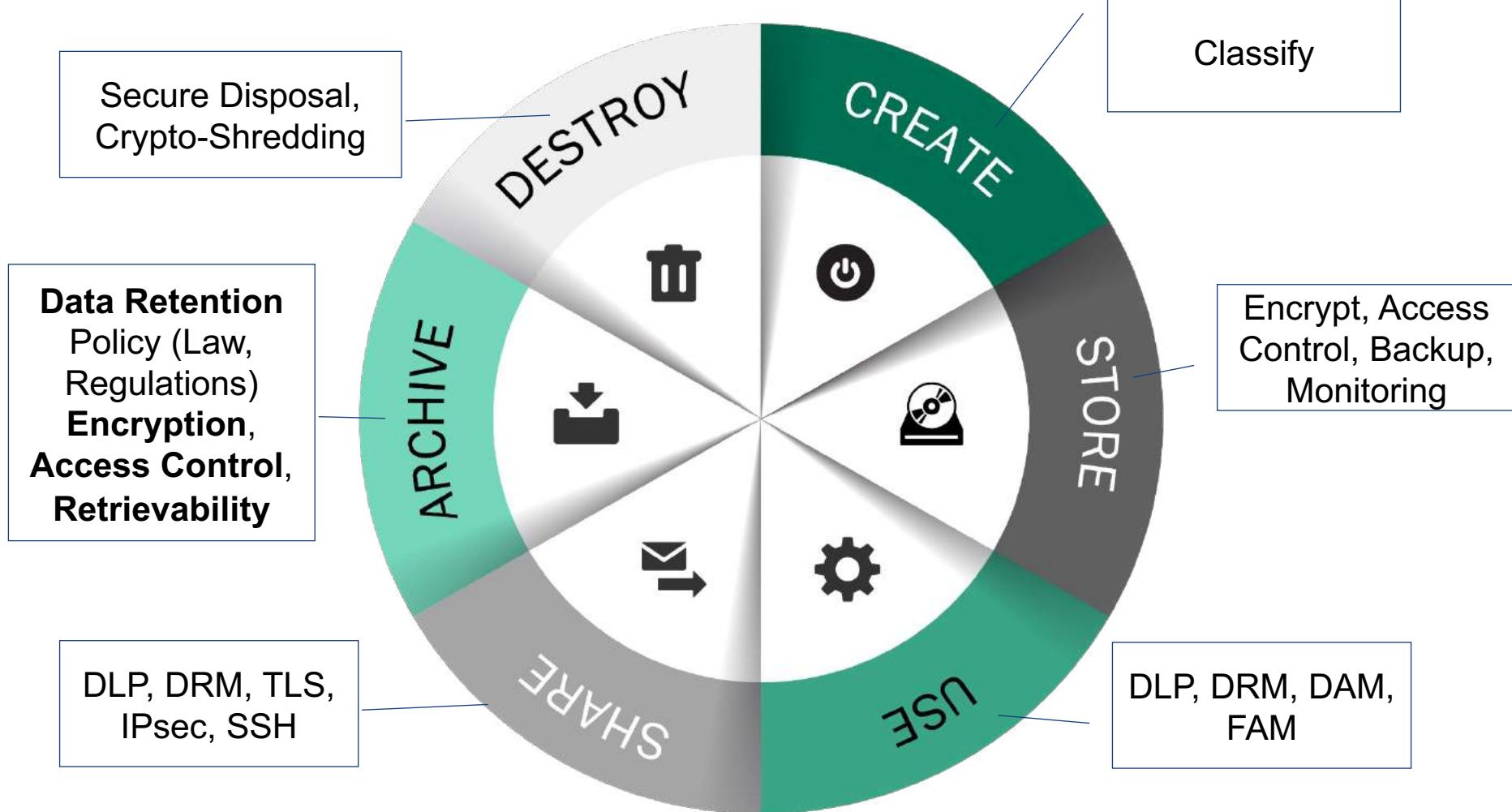
Asset/Asset Group	Threat/Vulnerability pair	Risk Level	Risk Mitigation / Security Controls
Customer PII	Threat: sniffing/unauthorized disclosure Vulnerability: transmission in cleartext or stored in cleartext	xx	
Web Server	Threat: hacking / unauthorized access Vulnerability: insecure configuration, patching not done in timely manner	yy	
.....	



APPENDIX F

(Data Lifecycle)

Data Life Cycle Phases



DLP: Data Loss Prevention
DRM: Digital Rights Management
DAM: DB Activity Monitoring
FAM: File Activity Monitoring

The End



ARCHITECTING SOFTWARE SOLUTIONS

DESIGN HIGHLY AVAILABLE ARCHITECTURE

Instructor: Darryl Ng

Email: darryl.ng@nus.edu.sg

Total slides: 104



Objectives

- Understand
 - Availability concepts
 - Availability patterns
 - Technologies & Tactics
 - Reference architectures



Topics

- Availability:
 - Definitions, Measures & Metrics
 - Causes for Low Availability
 - Architectural Strategies
 - Tactics
 - Reference Architectures



Overview of Availability

- Availability
 - Measure of the readiness for usage of a system or component
 - Probability that the system will work as required.
- Reliability
 - Probability that the system will perform its intended function without failure for a specified period of time under stated conditions.
- Fault Tolerance
 - Property of a component, subsystem, or system that enables normal service to continue even though a fault has occurred within the system.



Availability – Definition and Measure

Availability %	Downtime per year	Downtime per month	Downtime per week
90% (1 nine)	36.5 days	72 hours	16.8 hours
99% (2 nines)	3.65 days	7.20 hours	1.68 hours
99.5%	1.83 days	3.60 hours	50.4 minutes
99.9% (3 nines)	8.76 hours	43.8 minutes	10.1 minutes
99.95%	4.38 hours	21.56 minutes	5.04 minutes
99.99% (4 nines)	52.56 minutes	4.32 minutes	1.01 minutes
99.999% (5 nines)	5.26 minutes	25.9 seconds	6.05 seconds
99.9999% (6 nines)	31.5 seconds	2.59 seconds	0.605 seconds
99.99999% (7 nines)	3.15 seconds	0.259 seconds	0.0605 seconds



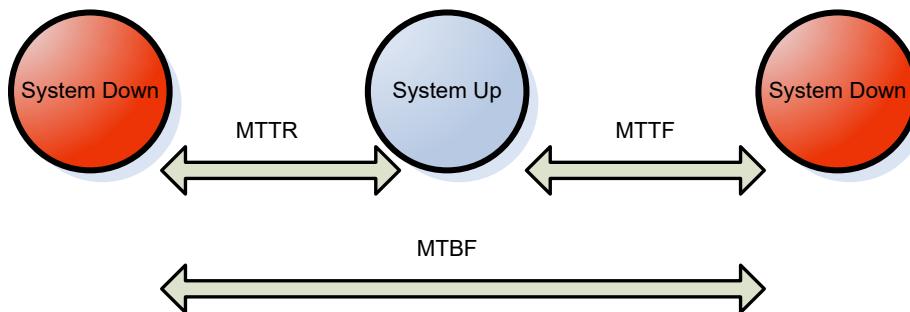
Reliability

- probability that the system will meet certain **performance standards** in generating correct output for a desired time duration
- used to determine how well the service will be available in context of different real-world conditions



Related Availability Metrics

- **Mean Time to Recover (MTTR)**
 - Time that it takes to recover (includes repair) a component, subsystem or system.
- **Mean Time to Failure (MTTF)**
 - Average amount of time from when a system becomes available to the time it becomes unavailable.
- **Mean Time between Failure (MTBF)**
 - Average time between successive failures of a given component, subsystem or system.





MTTR example

- Formula
 - $\text{Total maintenance time} / \text{Total number of repairs}$
- Example
 - A app server fails three times throughout a workday. The first repair & recovery lasted for 30 minutes, while the other two recovery lasted only 15 minutes. What is the MTTR?
- Diagnosis
 - Measure and improve average time taken to repair assets
 - Understand how much time to schedule for maintenance and repairs
 - Help reduce downtime in areas of business that are constantly impaired
 - Pick out anomalies in incident management



MTTF example

- Formula
 - Total hours in operation/ Total number of units
- Example
 - We tested 3 IoT environment sensors. The first one failed after 500,000 hours, the second one failed after 600 000 hours, and the third sensor failed after 700,000 hours in use.
- Diagnosis
 - Inventory lead time
 - Quality control
 - Misuse of equipment



MTBF example

- Formula
 - Total operation time/ **Total number of failures**
- Example
 - App server expected runtime of ten hours (a day), it ran for nine hours. It failed for one hour spread over three occasions. What is the MTBF?
 $9/3=3$
- Diagnosis
 - **Cost** of breakdowns
 - **Frequency** of failures



Availability Computations

- Let A be an index of system availability expressed as a percentage or fraction

$$A = \text{MTBF} / (\text{MTBF} + \text{MTTR})$$

- To calculate downtime:

$$\text{Downtime Per Year (min)} = (1 - \text{Uptime Ratio}) \times 365 \times 24 \times 60$$

Availability	Downtime
90% (1 nine)	36.5 days/year
99% (2 nines)	3.65 days/year
99.9% (3 nines)	8.76 hours/year
99.99% (4 nines)	52.5 minutes/year
99.999% (5 nines)	5.4 minutes/year
99.9999% (6 nines)	31.5 seconds/year

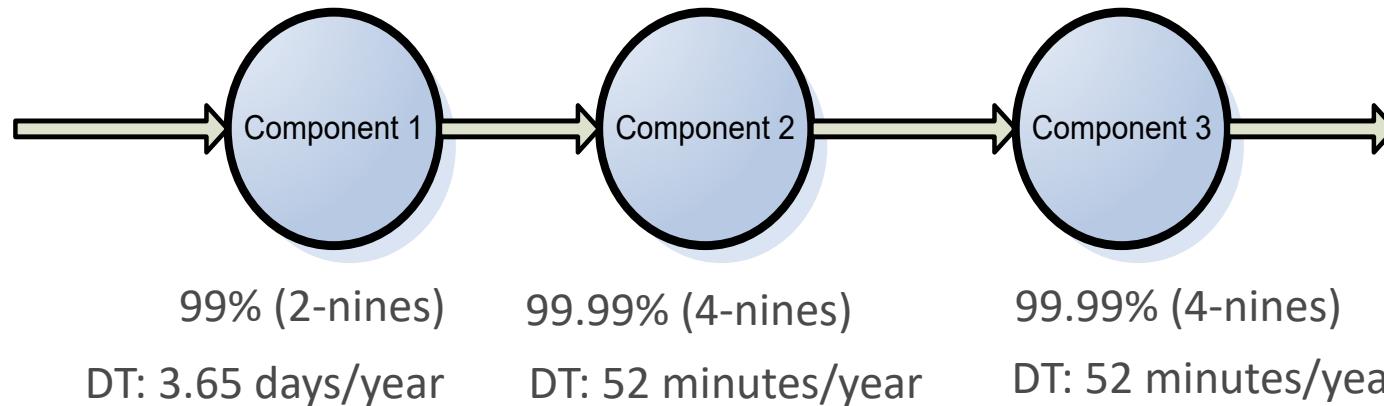


Availability Concepts and Principles

- Overall System Availability
 - Is a function of individual component Availability.

- Overall Availability of System having *components in series*.

$$\text{Availability} = A_1 \times A_2 \times A_3$$



What is the availability of the system (in series)?

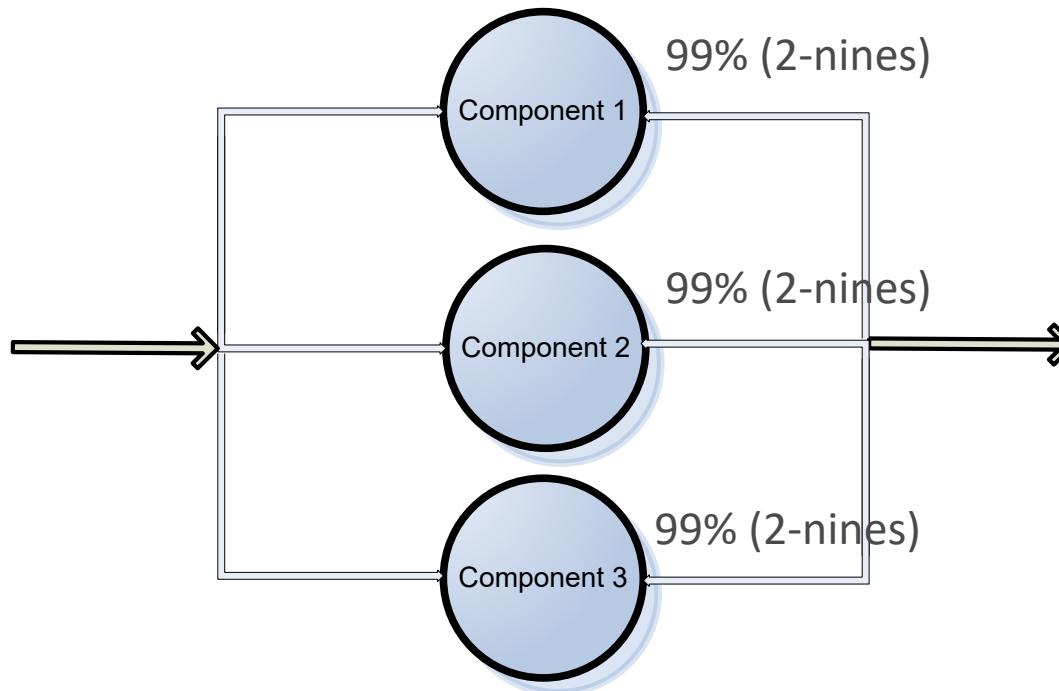
Note: The overall availability is always lower than the availability of any component in the series.



Availability Concepts and Principles

- Overall Availability of System having *components in parallel*.

$$\text{Availability} = 1 - ((1-A_1) \times (1-A_2) \times (1-A_3))$$



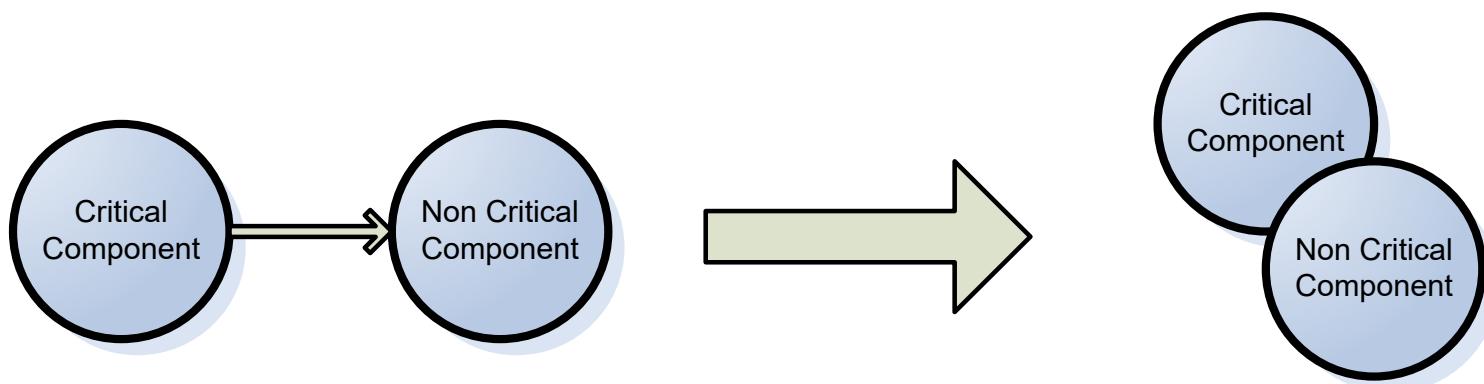
What is the overall availability of the system (in parallel)?

Note: Overall availability is always higher than the availability of the individual links (components).



Availability Concepts and Principles

- Separation of Concerns
 - Technique that can be used to enable loose coupling of the components that provide critical services

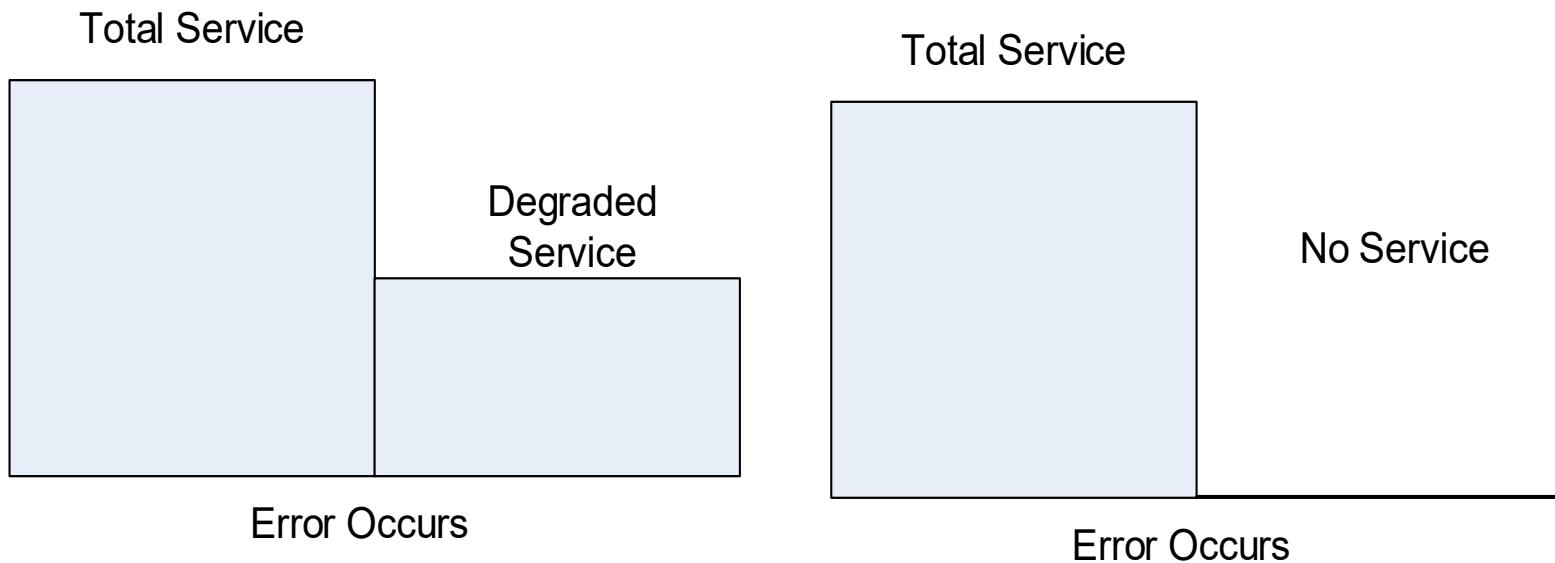




Availability Concepts and Principles

- Fault Tolerance

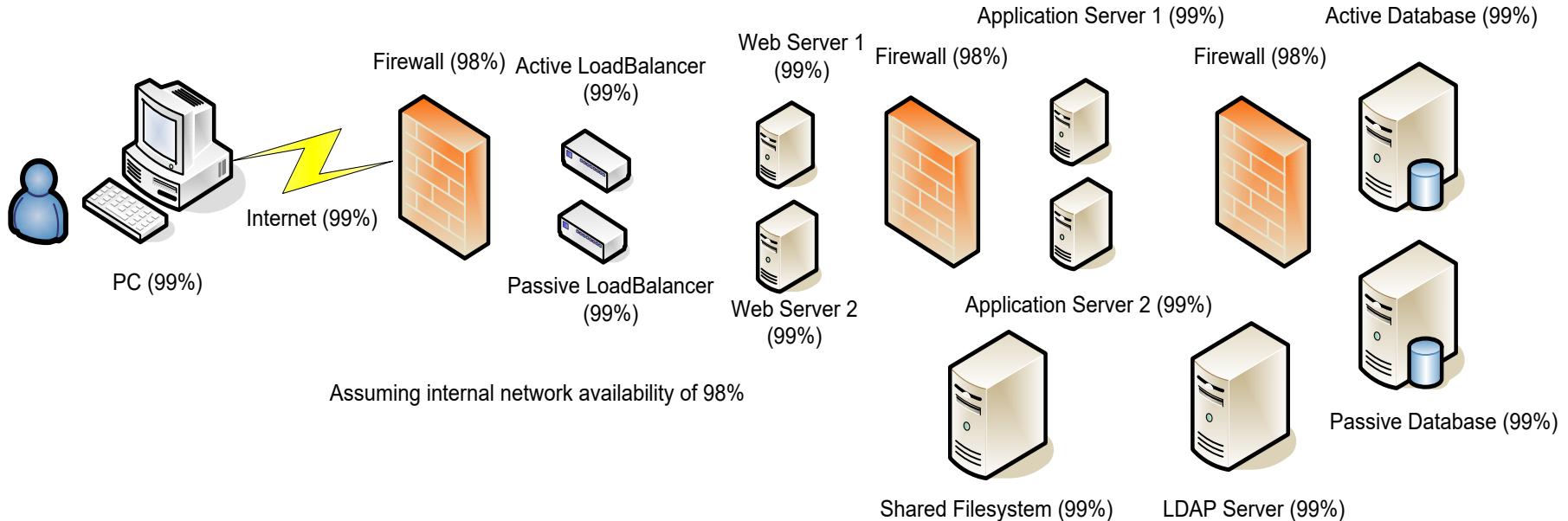
- Technique that can be used to enable the detection and correction of latent errors before they take effect



<https://s3.amazonaws.com/gitbook/Server-Admin-2019/ServerAdmin1Pre-InstallationConsiderations/1.04.PlanningForFaultTolerance.html>



Discussion



Can the above architecture achieve at least 95% availability?



Some questions to ponder...

- What is High Availability?
- How does High Availability mitigate failures?
- How is High Availability implemented?
- Examples of Applications/Platform in High Availability systems.
- How to decide whether your business needs High Availability?



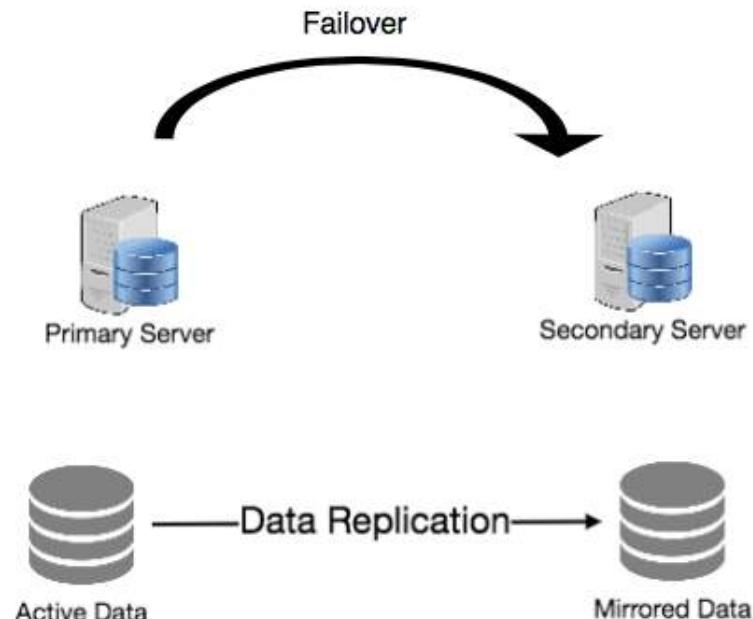


- Architectural Principles for Highly Available Systems
 - *High availability architecture is an approach of defining the components, modules or implementation of services of a system which ensures optimal operational performance, even at times of high loads.*
- Implementation of Highly Available Systems
 - No fixed rules ☹
 - **Follow best practices**
 - Obtain maximum out of minimum resources! ☺



Fundamental - High Availability Principles

- **Component redundancy**
 - Every component has at least one backup.
- **No single points of failure**
 - There is **no single component** whose malfunction can cause complete system failure.
- **Failure detection and response**
 - Component failures are detected on time, and dealt with accordingly.
 - Health checks



<https://medium.com/must-know-computer-science/system-design-redundancy-and-replication-e9946aa335ba>



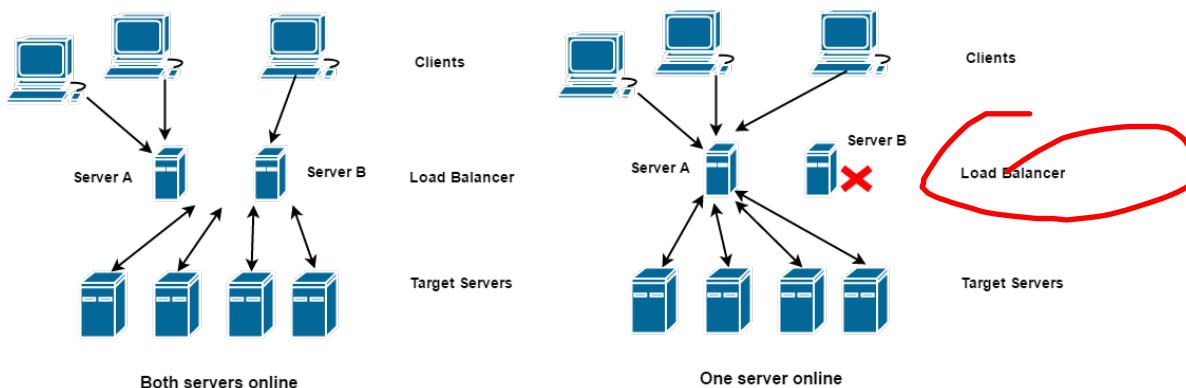
Improving Availability

- **Objective**
 - Assess ways and means of reducing Down Time
- **Approach**
 - Chart the various application systems and IT resources in the organisational environment.
 - Gather past statistics of downtime and assess the Availability.
 - Identify the **Causes of Downtime** *in general* and those *specific to the organisation's system* in particular.
 - Classify **causes** that are **controllable and non-controllable**.
 - Classify systems / functions into **critical and non-critical**
 - Derive Infrastructure & Architectural plans to *reduce Down Time* thereby *improve Availability*.
 - Consider alternatives to optimise costs (essential trade-off).

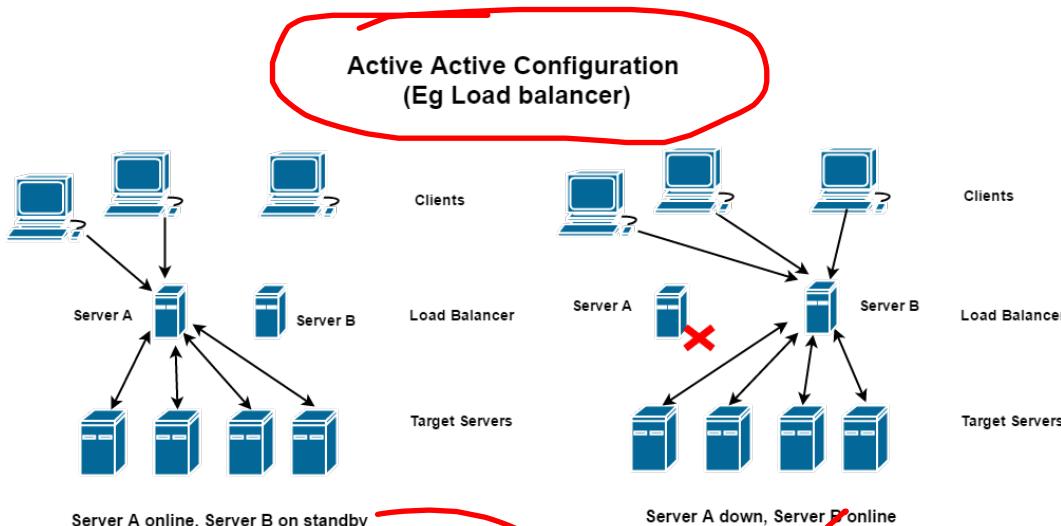
Key Message: Reduce Down Time

Ways of improving availability

✓



LB



Active Passive Configuration
 (Eg Load balancer)

https://raghumb.gitbooks.io/a-guide-to-software-architecture/content/infrastructure_concepts/active_active_passive.html



Planned vs Unplanned Downtime

- Downtime can be planned and unplanned.
 - Planned downtime usually is caused by deployment.
 - Unplanned downtime is downtime caused by failure of the system.

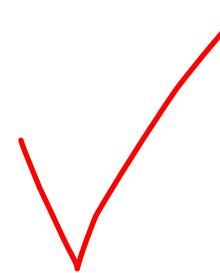
We focus more on unplanned downtime.

Reducing/removing planned downtime is the focus of continuous delivery/deployment.



Down Time Causes (non-Availability)

- Causes from a Software Application Perspective
 - Heterogeneous environments
 - Multiple single points of failure
 - Multiple application interfaces
 - Inadequate monitoring
 - Resource bottlenecks
 - Team Silos
 - Job failures
 - Network Issues
 - Password expiry or locked accounts
 - Employee attrition



Source: <https://foglogic.com/top-10-reasons-for-application-downtime/>



Down Time Causes (non-Availability)

- Causes from a **Hosting Server Perspective**
 - Human Error
 - Security Flaws
 - Bugs in a Server's OS
 - **Understaffed IT Departments**
 - Outdated hardware
 - Instability of server hardware
 - **Server OS Too Old for New Computers**

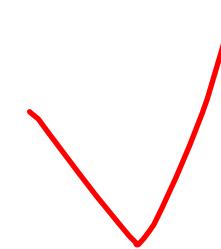
Source: <https://www.extnoc.com/blog/top-7-major-causes-of-it-downtime/>



Down Time Causes (non-Availability)

- Causes from a **Cloud Hosting** Perspective

- Overload
- Noisy neighbor
- Retry spikes
- Bad dependency
- Scaling boundaries
- Uneven sharding
- **Pets** Risk of Downtime: Since each server is unique and its configuration might not be easily replicated, any hardware failure or other issues could lead to extended downtime while waiting for repairs or replacements.
- Bad deployment
- Monitoring gaps
- Failure domains



Source: Google Conference:
<https://www.techrepublic.com/article/10-common-causes-of-downtime-and-how-to-avoid-them/>



Addressing the Causes

Cause	Architect's approach
Heterogeneous Environment	<ul style="list-style-type: none">- Adopt Architectural best practices derived from Solutions Integration and Service Oriented Architectures
Multiple Application Interfaces	<ul style="list-style-type: none">- Adopt EA and standardisation efforts to control variety. <p>"EA" typically stands for "Elastic Architecture" or "Elasticity Architecture." Elasticity refers to the ability of a system to automatically and dynamically scale its resources up or down based on the changing demand and workload.</p>
Multiple Single Point Failure	<ul style="list-style-type: none">- Adopt Hardware, Data, Server, Network redundancy- Evolve Backup strategies *
Inadequate Monitoring	<ul style="list-style-type: none">- Adopt manual and automated Monitoring *
Resource Bottlenecks	<ul style="list-style-type: none">- Perform component level criticality analysis and build redundancies for critical sets.
Team Silos, Job failures, Password expiry or locked accounts, Employee attrition	<ul style="list-style-type: none">- Non Technical.- Human relations approach may need to be employed for corporate welfare

* Further discussion & expansion in later slides



Addressing the Causes

Cause	Architect's approach
Human Error	<ul style="list-style-type: none">- May need both Technical and Non-Technical approach.
Understaffed IT Dept	<ul style="list-style-type: none">- Propose proper/periodic training plan and audits for crucial functions.
Security Flaws	<ul style="list-style-type: none">- Devise secure infrastructure to minimise threats to attacks such as DOS, malwares, etc.
Security Threats	<ul style="list-style-type: none">- Architecture the infrastructure to identify critical components to be placed behind Firewalls with DMZ and public domains reviewed. *- Backup strategies & DRPs.
Bugs in Server OS	<ul style="list-style-type: none">- Regular patching, upgrades and maintenance support
Outdated Hardware	<ul style="list-style-type: none">- Hmm!
Instability of Hardware	
Server OS too Old	

* Further discussion & expansion in other lectures topics



Addressing the Causes

Cause	Architect's approach
Overload	<ul style="list-style-type: none">- Build redundancy, separation of concerns where feasible & required *- Load Shedding averting crash- Define threshold loads for server based on client calls and balk the further calls
Noisy Neighbour	<ul style="list-style-type: none">- Limiting calls per user per unit time to avoid spams or greedy clients.- Alternate to "user" is enforcement of limits by IP address.- Derive framework to identify what will constitute a noisy neighbour for purpose of limits.
Retry spikes	<ul style="list-style-type: none">- Rejection (above) may increase re-tries. Derive policy for controlling retries.
Bad dependency	<ul style="list-style-type: none">- Load testing to identify critical components.

* Further discussion & expansion in later slides



Addressing the Causes

Cause	Architect's approach
Scaling boundaries	<ul style="list-style-type: none">- Employ sharding and clustered environment *
Uneven Sharding	<ul style="list-style-type: none">- Engage splitting of problematic shards.
Pets	<ul style="list-style-type: none">- Identify human intensive components and engage in more training and encouragement for avoiding downtime.
Bad deployment	<ul style="list-style-type: none">- Devise phased roll out to avoid big bang deployment problem.- Architect for rollback systematically.
Monitoring Gaps	<ul style="list-style-type: none">- Monitoring user experience and resolving issues.
Failure Domains	<ul style="list-style-type: none">- Geographical redundancy and off-site backups.

* Further discussion & expansion in later slides



Discussion

- If a system is unavailable for 15 minutes in a particular 30-day month, what is the availability of the system in that month?



Online SLA and Uptime Calculator:
<https://www.hostingmanual.net/uptime-calculator/>



SOURCES OF UNPLANNED DOWNTIME



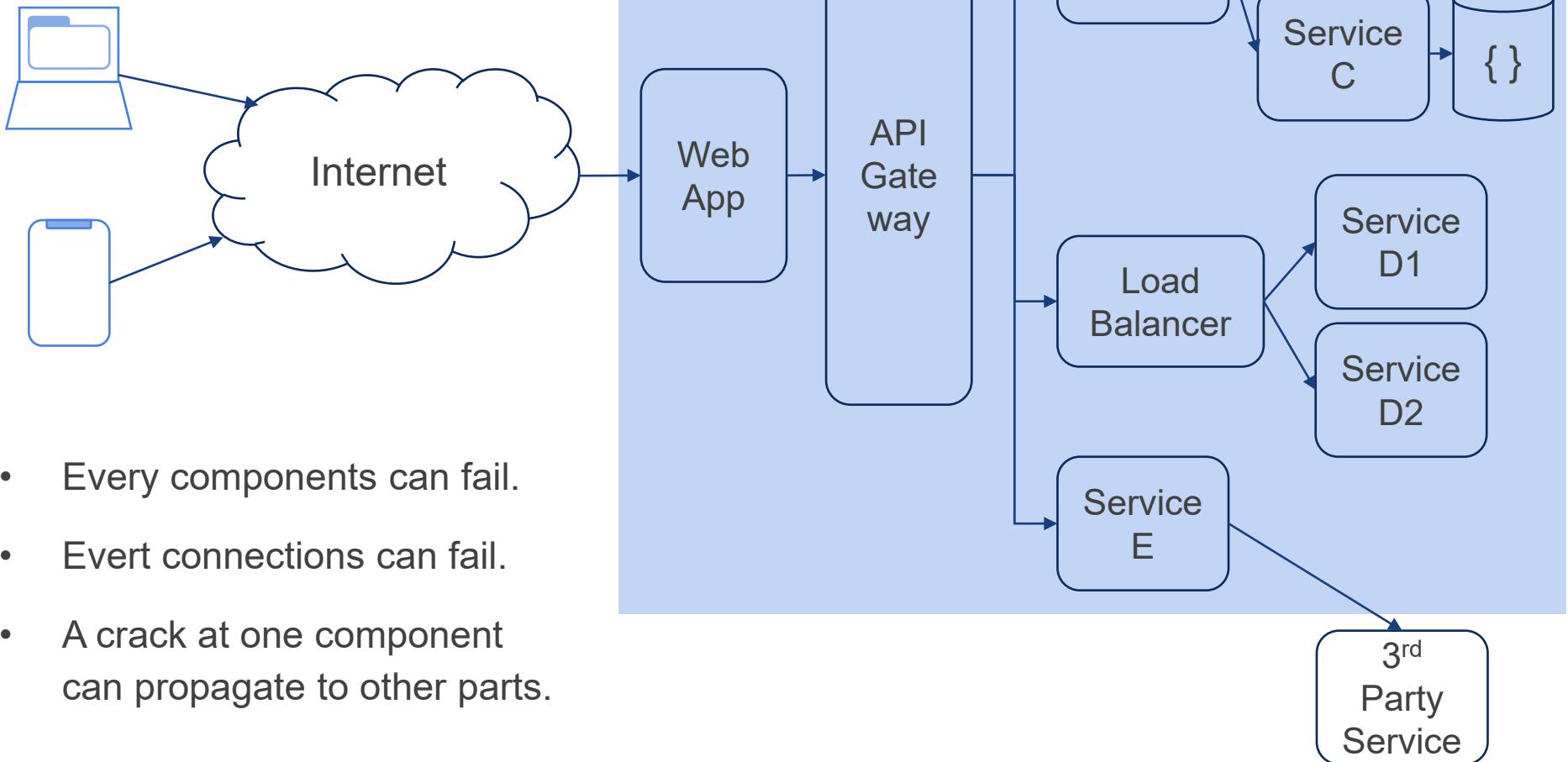


Failures

- Hardware failures
- Network failures
- Software failures CICD

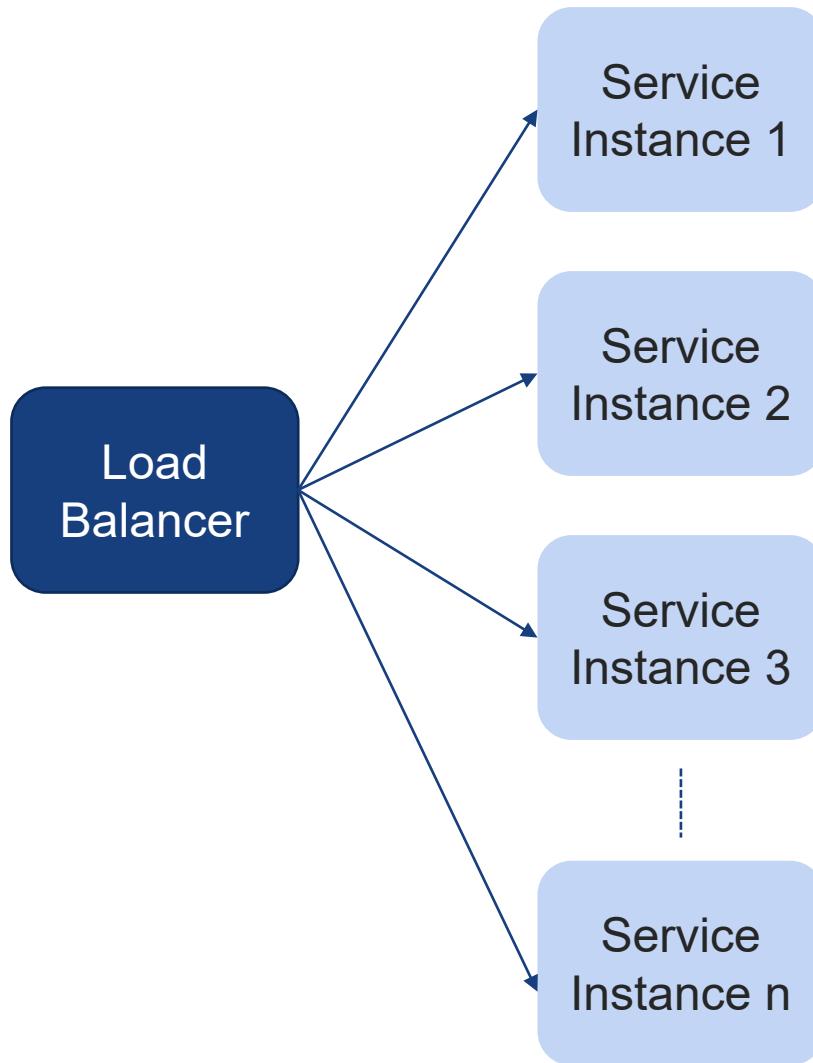


Distributed System





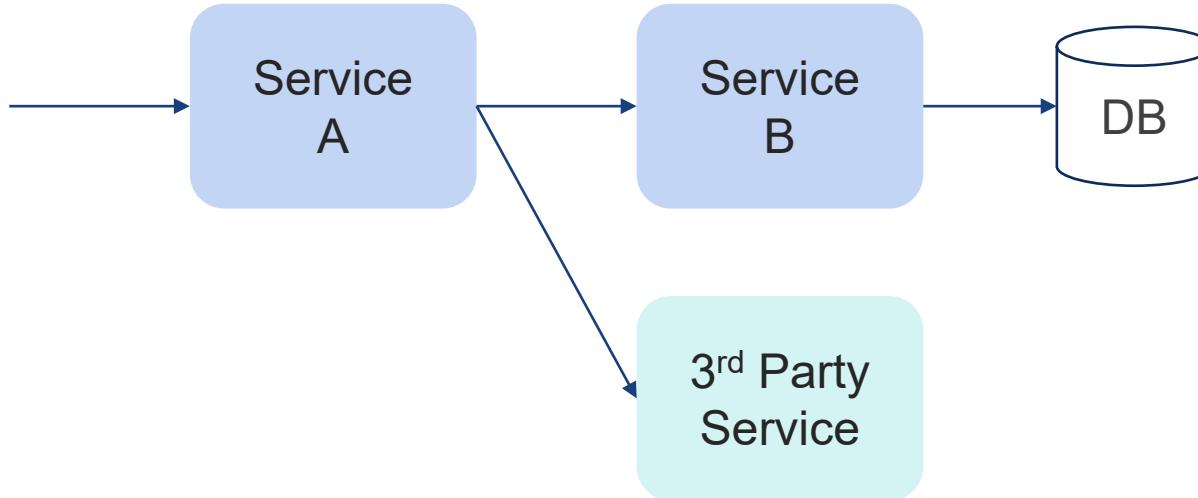
Chain reaction



- All the service instances are **serving the same logic**.
- Assuming there is a bug (e.g. causing memory leak), and it is more prevalent for a specific operations.
- If service instance 1 handles many of such operation, it will be the first one to crash.
- Subsequently, the remaining instances needs to handle higher loads.
- **This triggers a chain reaction where all the instances will fail faster.**



Cascading Failures

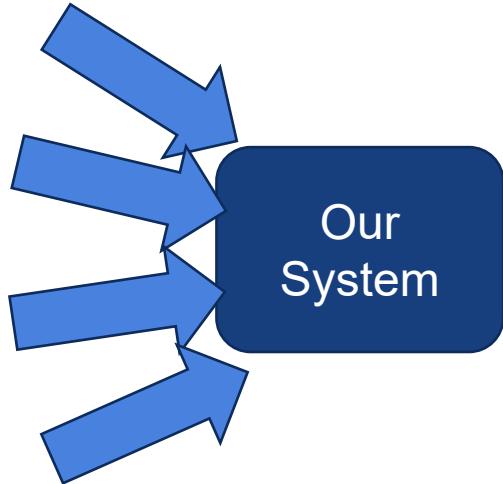


- Upstream service needs to handle any error from service providers.
- Slow responses can propagate problem.
- Speculative retry may exacerbates the problem.
- Pay attention to resource pools

<https://www.datadoghq.com/videos/the-anatomy-of-a-cascading-failure-n26/#throttling-circuit-breaking>

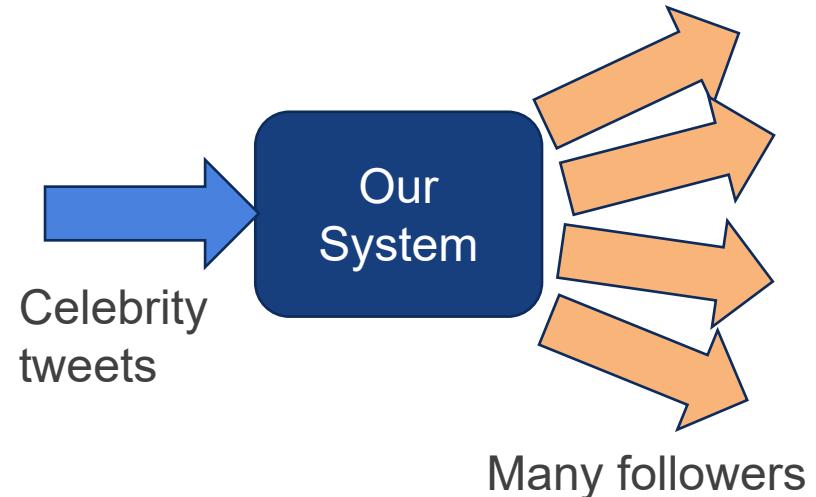


Impact of Users



Heavy incoming request

- **Malicious users**
- Marketing campaign not well communicated

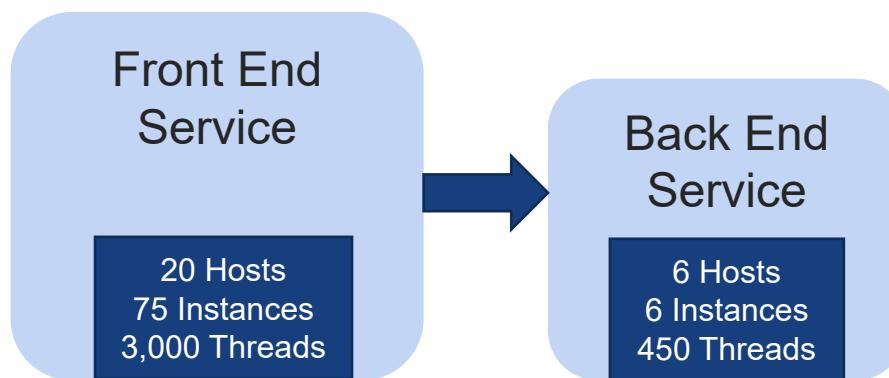
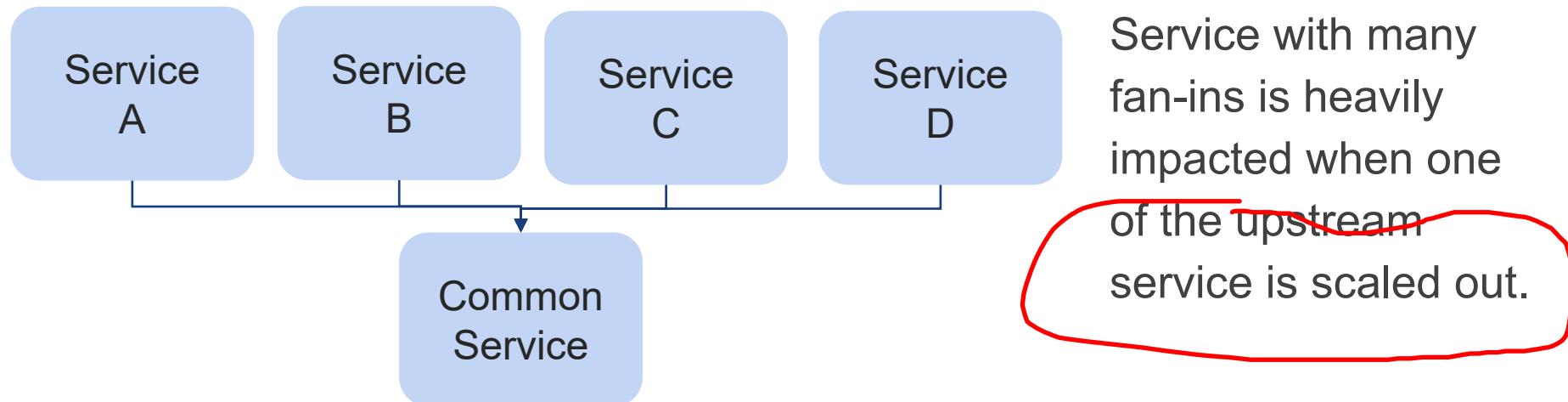


Heavy operation

- Special/power user (e.g. **famous people tweets** a message)

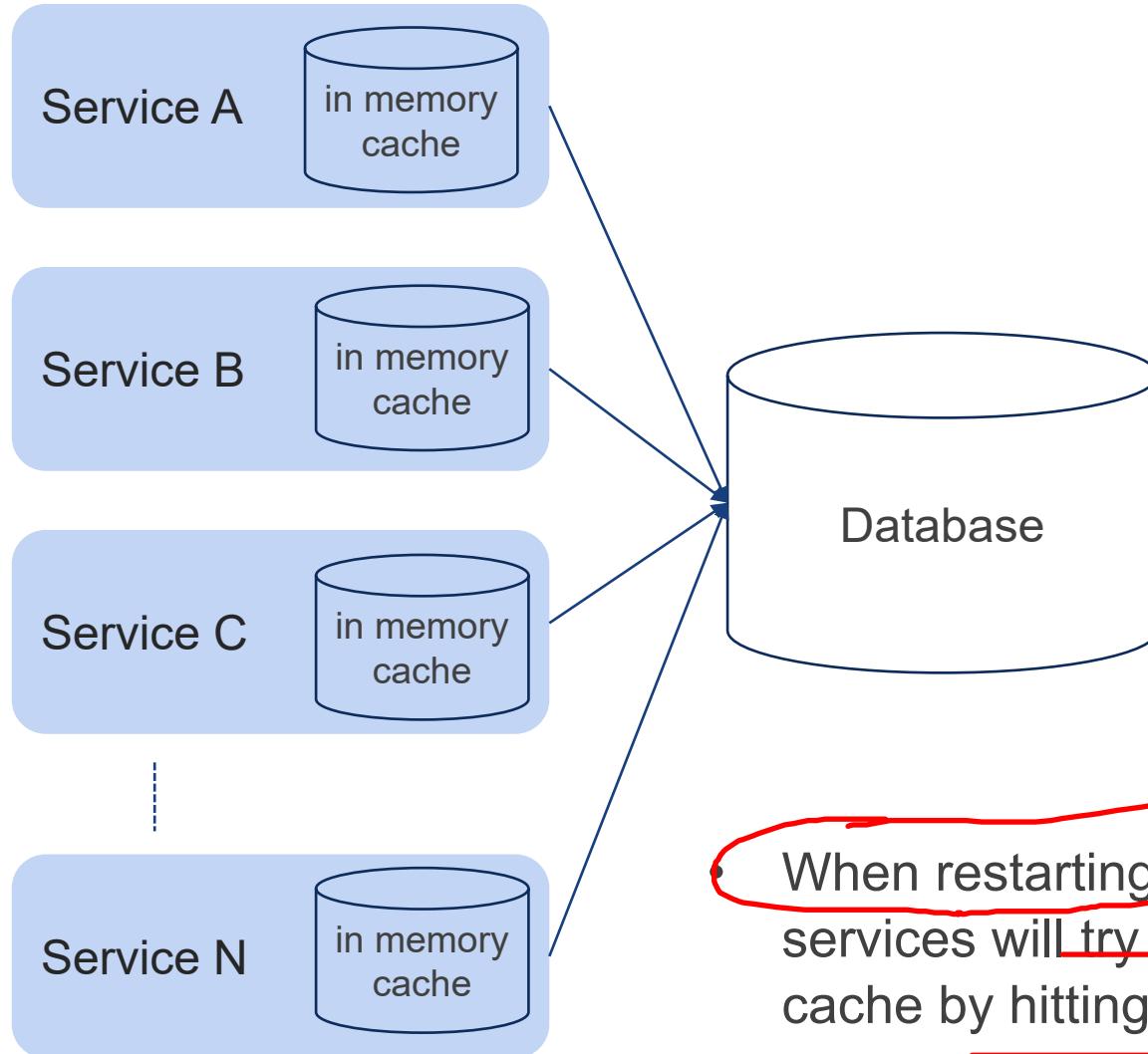


Scaling Effect and Unbounded Capacities



Observe the number of instances and the number of threads.

Dogpile



When restarting the system, all services will try to warm up their cache by hitting the database.



AVAILABILITY TACTICS



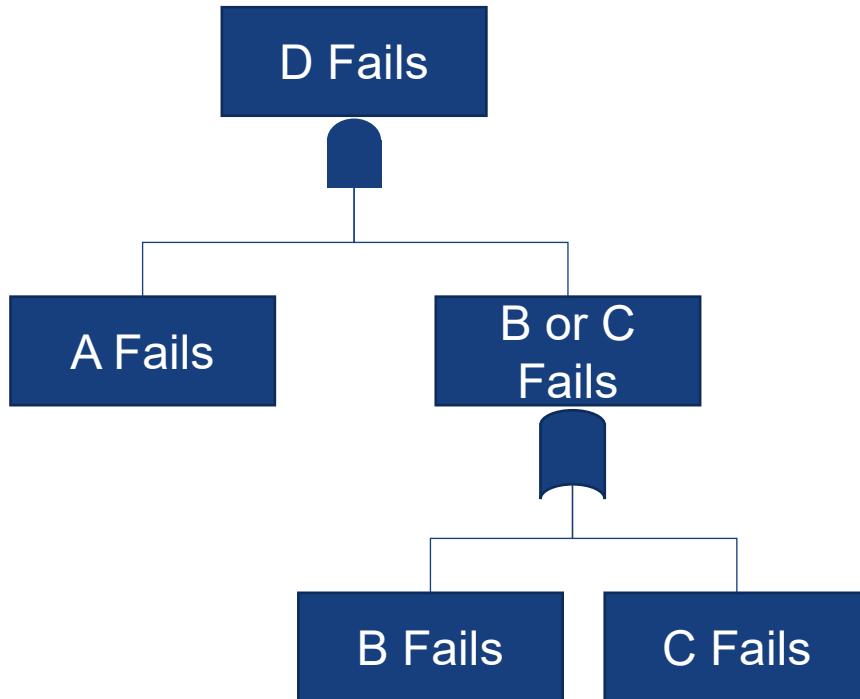
Availability Tactics



- Static Analysis
- Fault Detection
- Fault Recovery
 - Preparation and Repair
 - Reintroduction
- Fault Prevention
- Dynamic Analysis



Static Analysis



- “Minimal cut set” is the smallest combination of events along the bottom of the tree that together can cause the top event.
- A singleton reveals a single point of failure
- Can show if a critical component depends on non-critical component.



Availability Tactics - Fault Detection

- **Fault Detection** Tactics & Techniques
 - Ping / Echo
 - Heartbeat & Monitor
 - **Exceptions Tracking**
 - Timestamp
 - Sanity Checking
 - Condition Monitoring
 - Voting
 - **Exception Detection**
 - Self Test



Availability Tactics - Fault Recovery

- Preparation & Repair for recovery
 - Active Redundancy
 - Passive Redundancy
 - Spare Server
 - Exception Handling
 - Rollback
 - Software Upgrade
 - Retry
 - Ignore Faulty Behavior
 - Degradation
 - Reconfiguration



Availability Tactics - Fault Recovery

- Reintroduction Techniques for recovery
 - Shadow
 - State Resynchronization
 - Escalating Restart
 - Non-stop Forwarding

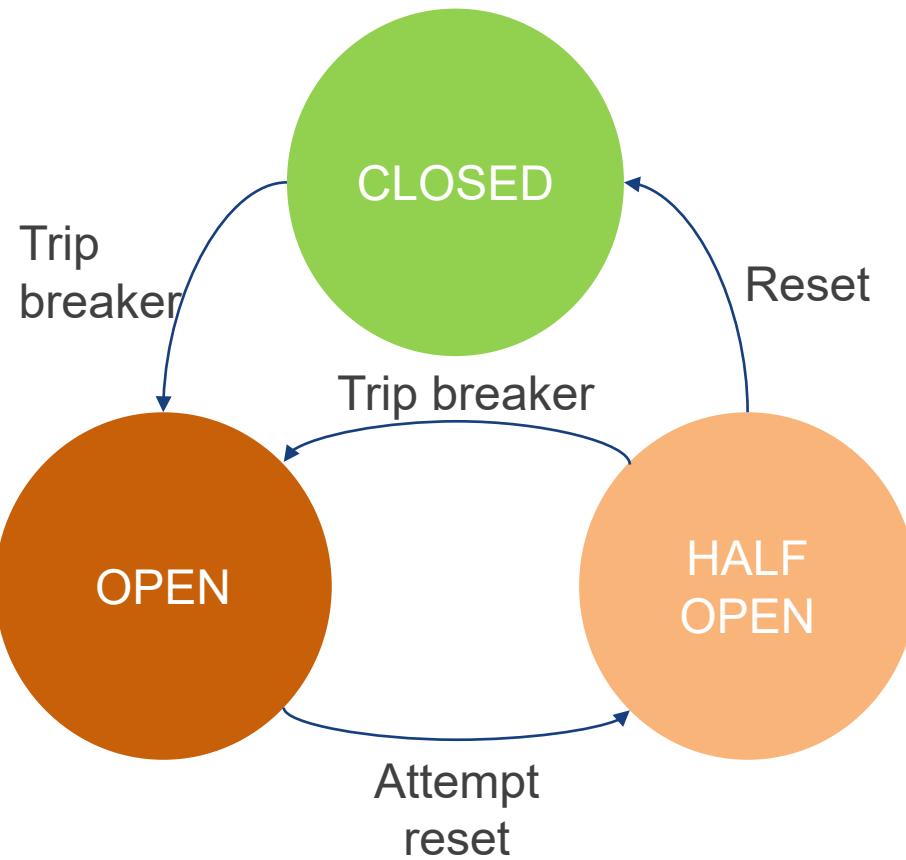


Availability Tactics – Fault Prevention

- Tactics and Techniques to avert Faults
 - Removal from service
 - Transactions
 - Process Monitor
 - Exception Prevention through handling
 - Increase competence sets



Timeout, Retry, and Circuit Breaker



- Protect a call to an external system or an internal operations that are subject to timeout or other execution failure.
- In **closed state**, the call would proceed.
- Once the number of failures exceeds a threshold, the circuit breaker trips and enter the open state.
- After a certain period, it will try to return to close state.



Circuit Breaker

What to do in open state:

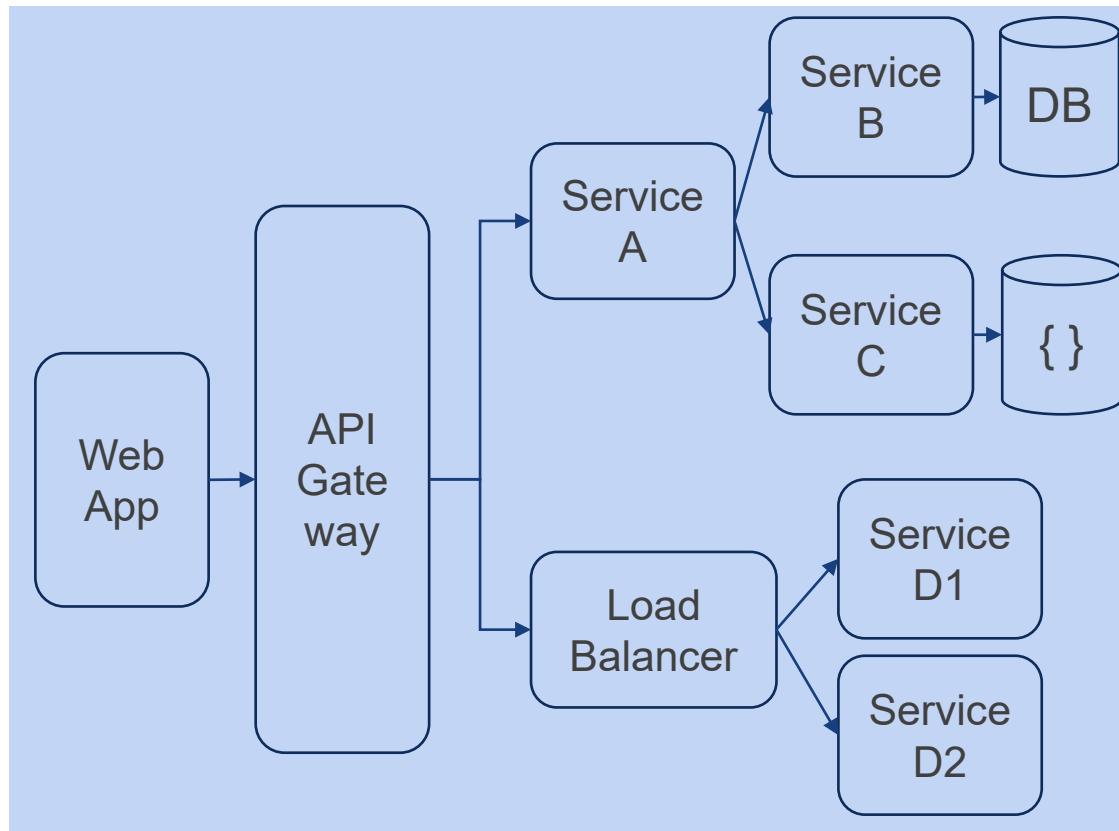
- Throw a different exception.
- Return a last good value (cached value).
- Return a generic instead of personalized answer.
- This is a good opportunity to discuss with the stakeholder on what would be a fallback strategy.



Circuit breaker should be visible to the operations team, and any change of state should be logged.

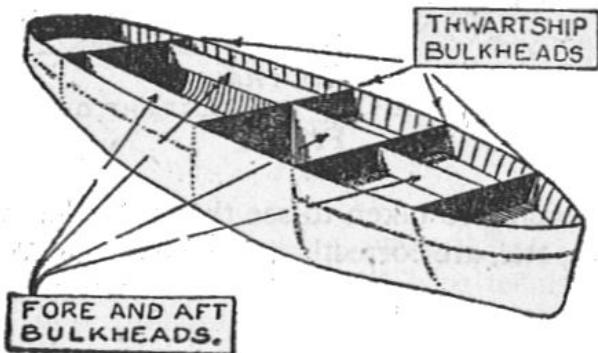


Steady State



- Database accumulates data.
- Services that uses in memory cache potentially accumulates caches in memory.
- Services and database accumulate logs.
- **It is important to have a strategy and implementation to purge them.**

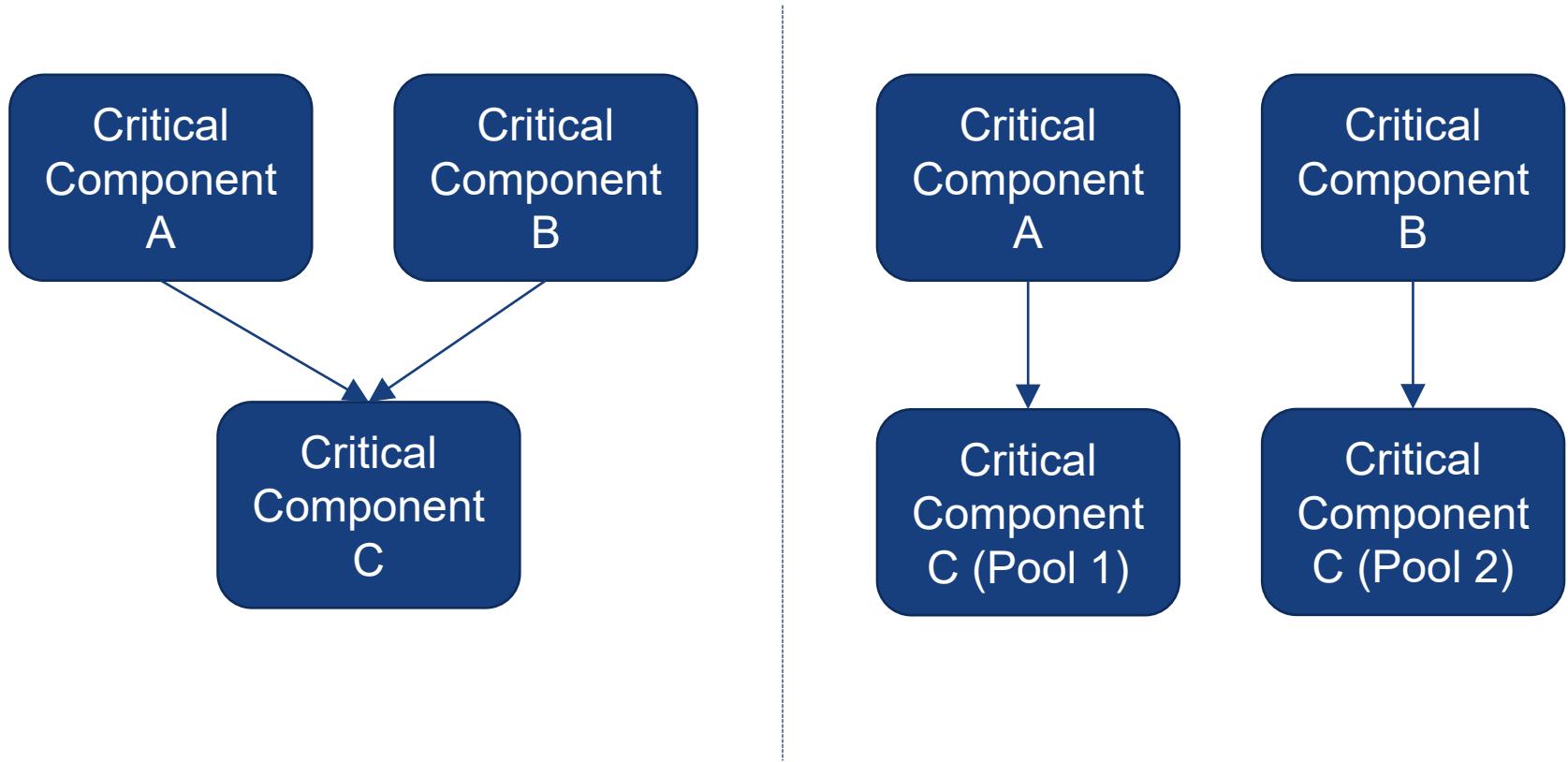
Bulk heads



- Partition the system.
- Redundant copy of services, middlewares, and databases.
- Physical separations.
- Availability zones separations.



Bulk heads



- Make sure critical systems are properly segregated.



Shed Loads, Handshake, and Backpressure

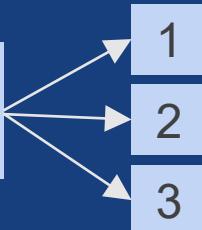
Shed
Loads

Shed loads by throttling the external requests to the system boundary.

E.g. **throttling by NGINX**, DDOS protection from Cloudflare

Throttling refers to the intentional limiting or restricting of resources

Load
Balancer



Handshake is an approach for caller to decide if the downstream service can handle the load.

E.g. Load balancer stop sending traffic to loaded instance.

A

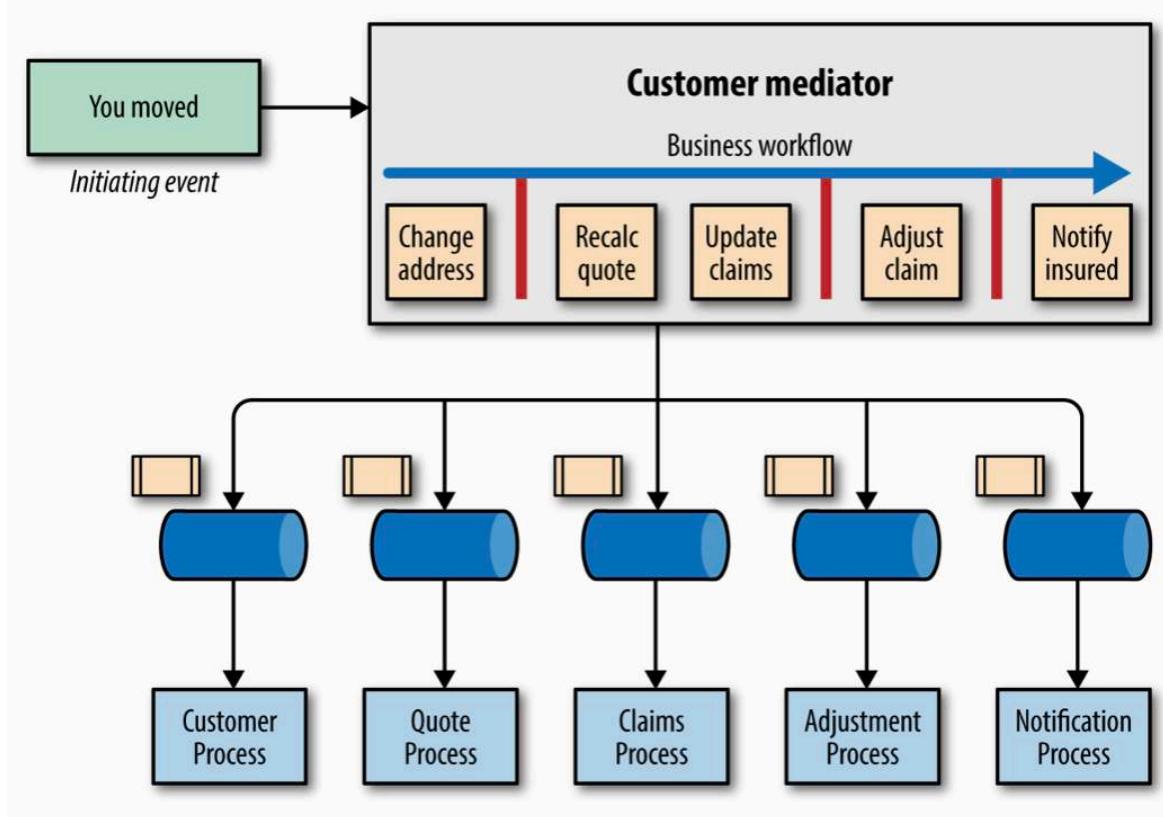
B

Backpressure is an approach to move the pressure upstream.

E.g. The use of circuit breaker approach is one example.



Decouple Middleware



- The use of publish-subscribe queue to handle events asynchronously.
- Event driven architecture.



Dynamic Analysis (Chaos Engineering)

Principles of chaos

Chaos Engineering is the discipline of experimenting on a system in order to build confidence in the system's capability to withstand turbulent conditions in production.

- Popularized by Netflix with their Simian Army tool.
 - Chaos Monkey
 - Latency Monkey
 - Conformity Monkey
 - Chaos Gorilla



Chaos Engineering Steps

1. Prepare the system
 - Ensure tracing and monitoring is available
2. Design the experiment
 - Clustered services should be unaffected by instance failures.
 - Application is responsive under high latency conditions.
3. Inject the chaos
 - Use chaos engineering tool (e.g. chaos monkey, failure injection tool)
4. Targeting chaos
 - Once we run a randomized experiment, we can run a targeted experiment.
5. Automate and repeat

<https://www.harness.io/blog/chaos-engineering-tools#:~:text=Netflix%E2%80%99s%20Chaos%20Monkey%20is%20an%20open-source%20chaos%20engineering,The%20software%20functions%20by%20implementing%20continuous%20unpredictable%20attacks.>



REDUNDANCY AND BACKUP STRATEGIES



Redundancies Strategies

- Avoiding a Single Point of Failure:
 - Hardware Redundancy
 - Data Redundancy
 - Server Redundancy
 - Network Redundancy
 - Backup Strategies



Apply Redundancy - Components

- Hardware component redundancy:
 - Redundant Power: We can replace a failed power supply without taking the system down - known as ‘hot-swapping’
 - Redundant Cooling: Failure of a cooling fan can lead to failure of temperature-sensitive components such as the processor or memory
 - Processors: Multiple central processing units (CPUs) can also provide load balancing and improve scalability



Apply Redundancy - Components

- Hardware Component Redundancy
 - Memory: Hard errors are when the component has failed catastrophically while soft errors are random electromagnetic events which may flip a single bit in memory.
 - Network Adapters: Implementing **failover pairs** to provide fault tolerance increases the reliability of servers.
 - RAID adapters: Using two RAID adapters as a failover pair.



Apply Redundancy - Server

- Server Redundancy

Refers to the amount and intensity of backup, failover or redundant servers in a computing environment.
- How is it Achieved?
 - Typically, a server replica is created with the same computing power, storage, applications and other operational parameters.
- Benefits
 - Server redundancy with clustering mitigates hardware- and operating system-level failures.
 - Simplistically, the redundant server may be in a power off mode or in-active mode and is turned on when failure to the production server happens.



Server Terminologies for Architects

- Primary Server
 - A primary server is a server that acts as the first source for Domain Name System (DNS) data and responds to queries
- Secondary Server
 - A secondary server is a type of server that serves as an addition to the primary server and is used for a variety of services. It has the same features and capabilities as the primary server and acts as a second or substitutive point of contact in case the primary server is unavailable, busy or overloaded.
- Backup Server
 - A backup server is a type of server that enables the backup of data, files, applications and/or databases on a specialized in-house or remote server.



Server Terminologies for Architects

- Server Cluster
 - A cluster, in the context of servers, is a group of computers that are connected with each other and operate closely to act as a single computer.
- Failover
 - **Failover** is the constant capability to automatically and seamlessly switch to a highly reliable backup. This can be operated in a redundant manner or in a standby operational mode upon the failure of a primary server, application, system or other primary system component. Sometimes synonymously referred as Switchover.
- Fallback
 - **Fallback** is the second stage of a two-part system for safeguarding information in a crisis mode during natural disasters or other events that can compromise an IT operation.



Apply Redundancy - Server

- Active / Passive

Highly Available (HA) clustering uses heartbeats to monitor the health and status of each computer node in the cluster.

When a failure occurs, the application is restarted on the backup redundant node automatically, a process known as failover.

- Active / Active

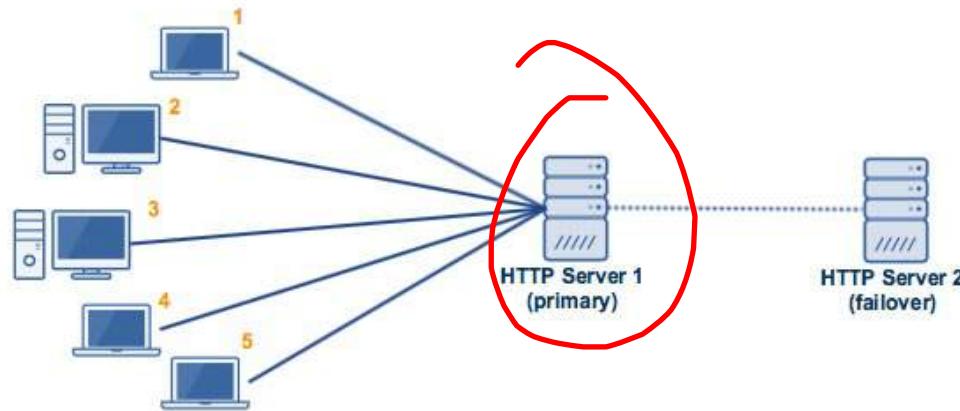
Multiple homogeneous computer nodes are linked to share processing and provide a logical view of a single virtual computer.

- Floating IP address

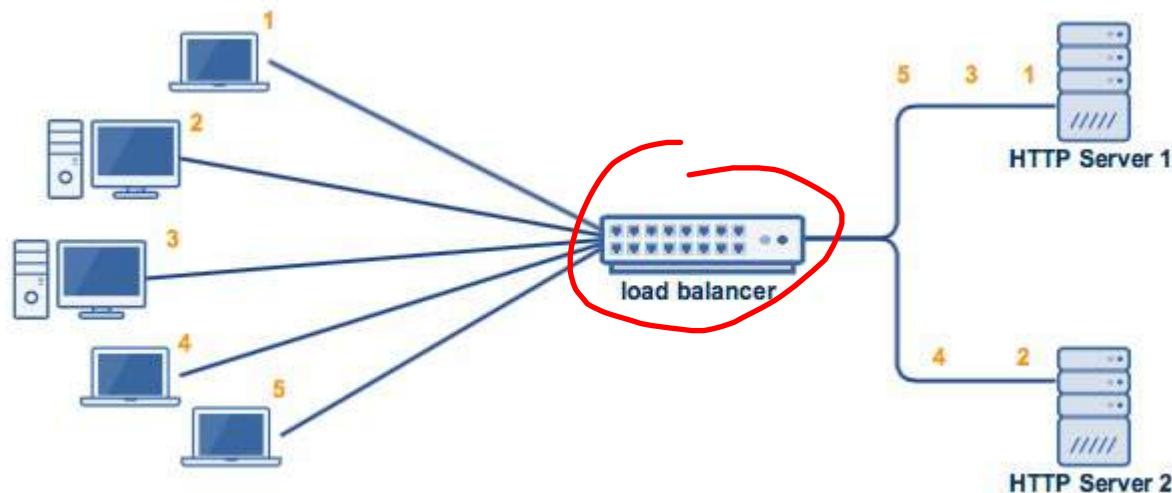
IP address can be moved from a one droplet to another droplet within the same cluster in an instance. This enables switching to standby at a moments notice.

Server Redundancy

- Active / Passive



- Active / Active



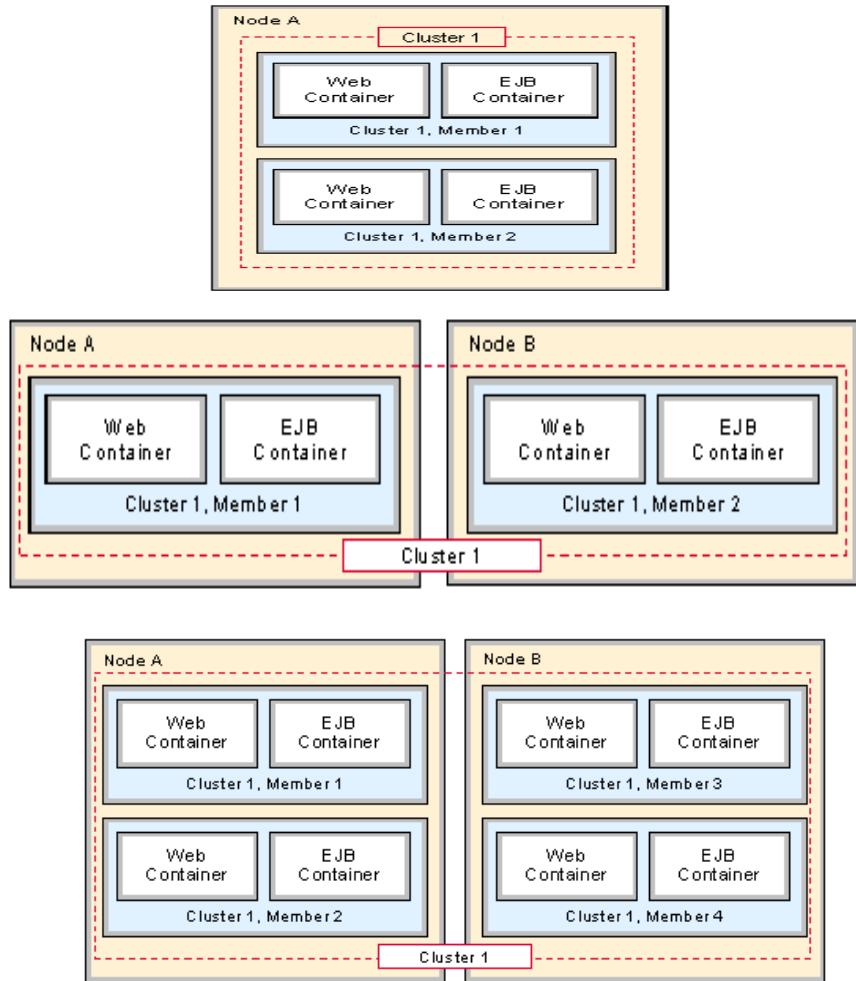


Apply Redundancy - Software

Software clustering

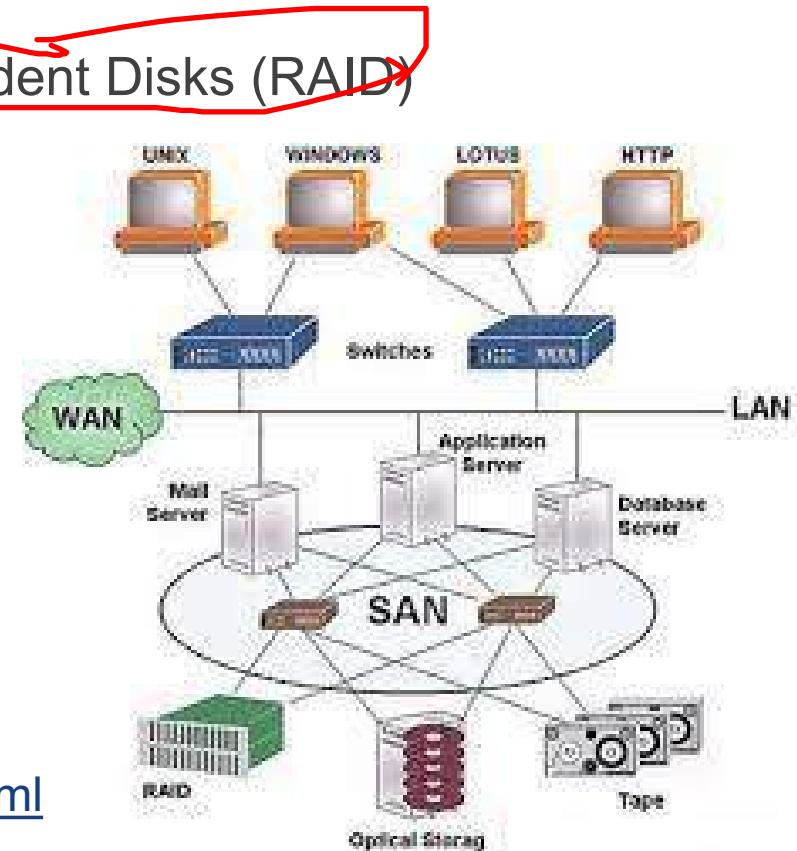
Technique to create multiple copies of software components running actively and collaborating seamlessly to present a single, unified system

- Strategies
 - Vertical Scaling
 - Horizontal Scaling
 - Vertical & Horizontal Scaling



Apply Redundancy - Data

- Data Redundancy
 - ~~Redundant Arrays of Independent Disks (RAID)~~
 - Storage Area Network (SAN)
 - Backup solution
 - Duplicate data centres

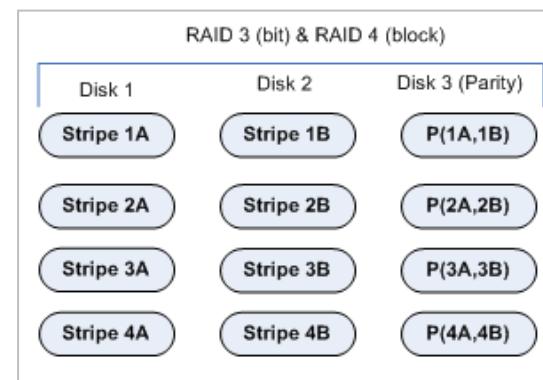
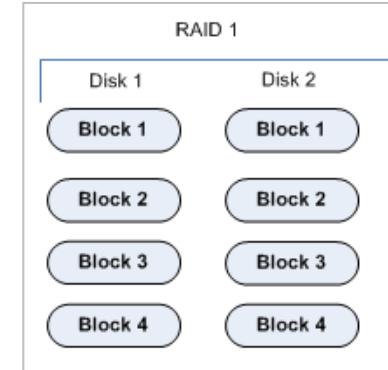
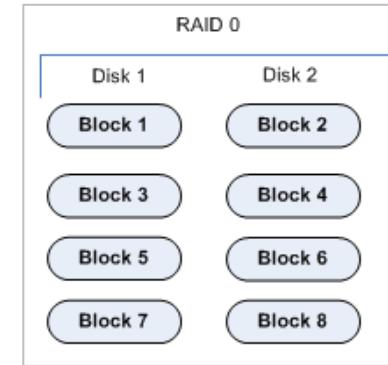


<http://terabitcom.com/m/case/san/9.html>



Apply Redundancy - Data: RAID

- Levels of RAID:
 - Level 0 – Striped Disk Array without Fault Tolerance
 - Level 1 – Mirroring
 - Level 2 – Striping with Error Correction Code (not used)
 - Level 3 – Bit Interleaved Parity
 - Level 4 – Block Interleaved Parity

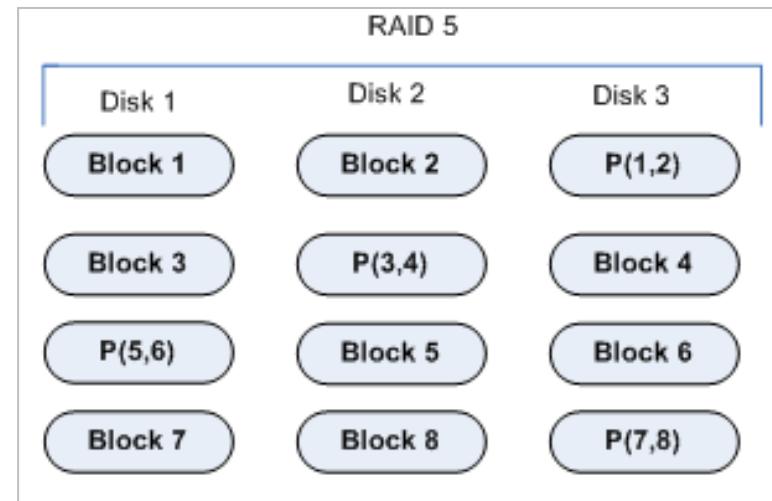


https://docs.oracle.com/cd/E19247-01/817-3337-18/appa_raid_basic.html

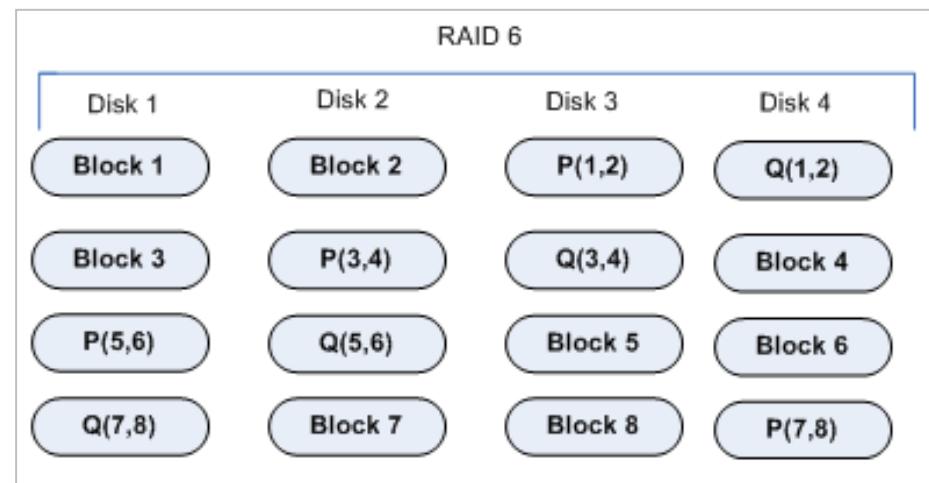


Apply Redundancy - Data: RAID

Level 5 – Block Interleaved
Distributed Parity



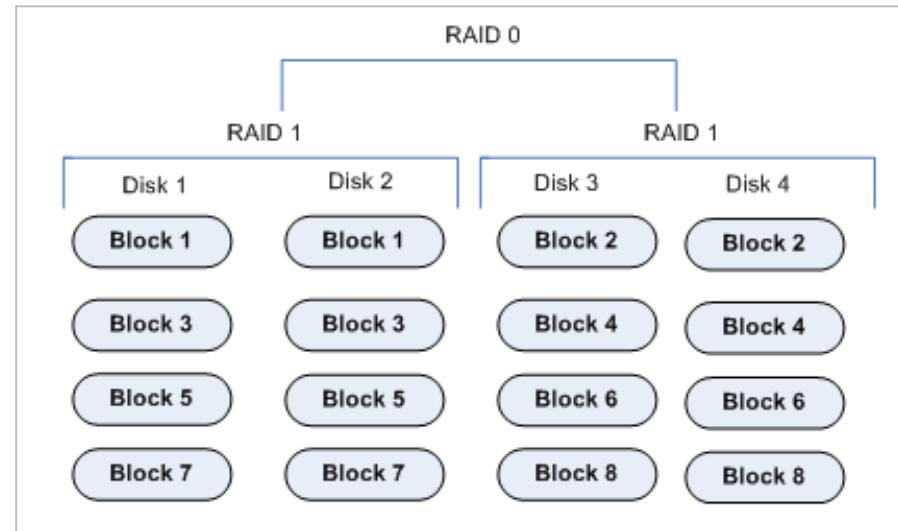
Level 6 – Block Interleaved
with two independent
Distributed Parity



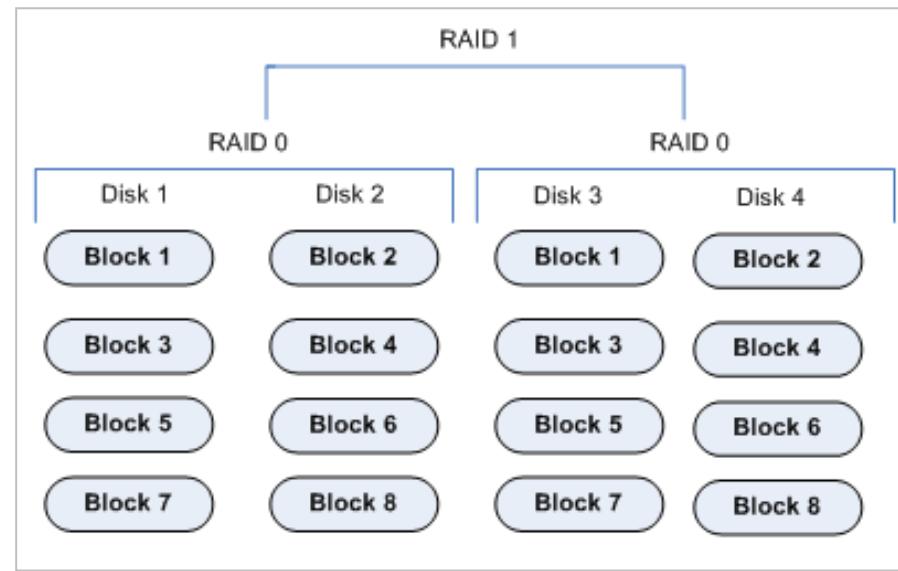


Apply Redundancy - Data: RAID

Level 1+0 – Mirroring
with Striping



Level 0+1 – Striping
with Mirroring





Comparison of RAID models

	RAID 0	RAID 1	RAID 5	RAID 6	RAID 10
Min number of disks	2	2	3	4	4
Fault tolerance	None	1 disk	1 disk	2 disks	1 disk
Disk space overhead	None	50%	1 disk	2 disks	50%
Read speed	Fast	Fast	Slow	Slow	Fast
Write speed	Fast	Fair	Slow	Slow	Fair
Hardware cost	Cheap	High (disks)	High	Very high	High (disks)



Discussion

Please spend 10 minutes thinking through the following questions applicable various RAID levels; confine to the ones in use such as 0,1,5,6,10 only:

- Assume that you have 3 TB hard disks and are required to configure an array of disks in various RAID patterns.
- You may assume that you have N (any suitable number) hard disks of 3TB each for your computations. If possible take two options for N if possible.
- Compute the Speed Gain & Fault Tolerance (i.e., tolerance for number of disks failed simultaneously)



<http://www.raid-calculator.com/default.aspx>



FAULT RECOVERY BACKUP STRATEGIES





Backup Strategies (Full)

Full Backup:

All the files and folders in the file set are backed up every time

- Advantages:
 - All files from the selected drives and folders are backed up to one backup set
 - In the event you need to restore files, they are easily restored from the single backup set.
- Disadvantages:
 - A full backup is more time consuming than other backup options
 - Full backups require more disk, tape, or network drive space than other backup options



Backup Strategies (Differential)

Differential Backup:

Backup of files that have changed or new (based on archive bits) since a full backup was performed. Together, a full backup and a differential backup include all the files on your computer, changed and unchanged.

- Advantages:
 - Differential backups require less disk, tape, or network drive space than full backups
 - Backup time is faster than full backups as only files that have changed require backup
- Disadvantages:
 - Restoring all your files may take considerably longer than full backup since you may have to restore both the last differential and full backup.
 - Restoring an individual file may take longer since you have to locate the file on either the differential or full backup.



Backup Strategies (Differential)

- Example:
 - **Monday:** *Perform a full backup*
 - **Tuesday:** *Perform a differential backup (files changed since Monday's full backup are backed up)*
 - **Wednesday:** *Perform a differential backup (files changed since Monday's full backup are backed up)*



Backup Strategies (Incremental)

Incremental Backup:

Backup of files that have changed or are new since the last incremental backup. For the first incremental backup, all files in the file set are backed up. If the same file set is used to perform a incremental backup only the files that have changed are backed up.

- Advantages:
 - Backup time is fastest than full or differential backups.
 - Incremental backups require less disk, tape, or network drive space than full or differential backups.
- Disadvantages:
 - In order to restore all the files, you must have all of the incremental backups available. Restoring time is slowest.
 - It may take longer to restore a specific file since you must search more than one backup set to find the latest version of a file.



Backup Strategies (Incremental)

- Example
 - **Monday:** *Perform first incremental backup (same as full backup)*
 - **Tuesday:** *Perform second incremental backup (only those files changed since Monday's incremental backup are backed up)*
 - **Wednesday:** *Perform third incremental backup (only those files changed since Tuesday's incremental backup are backed up)*

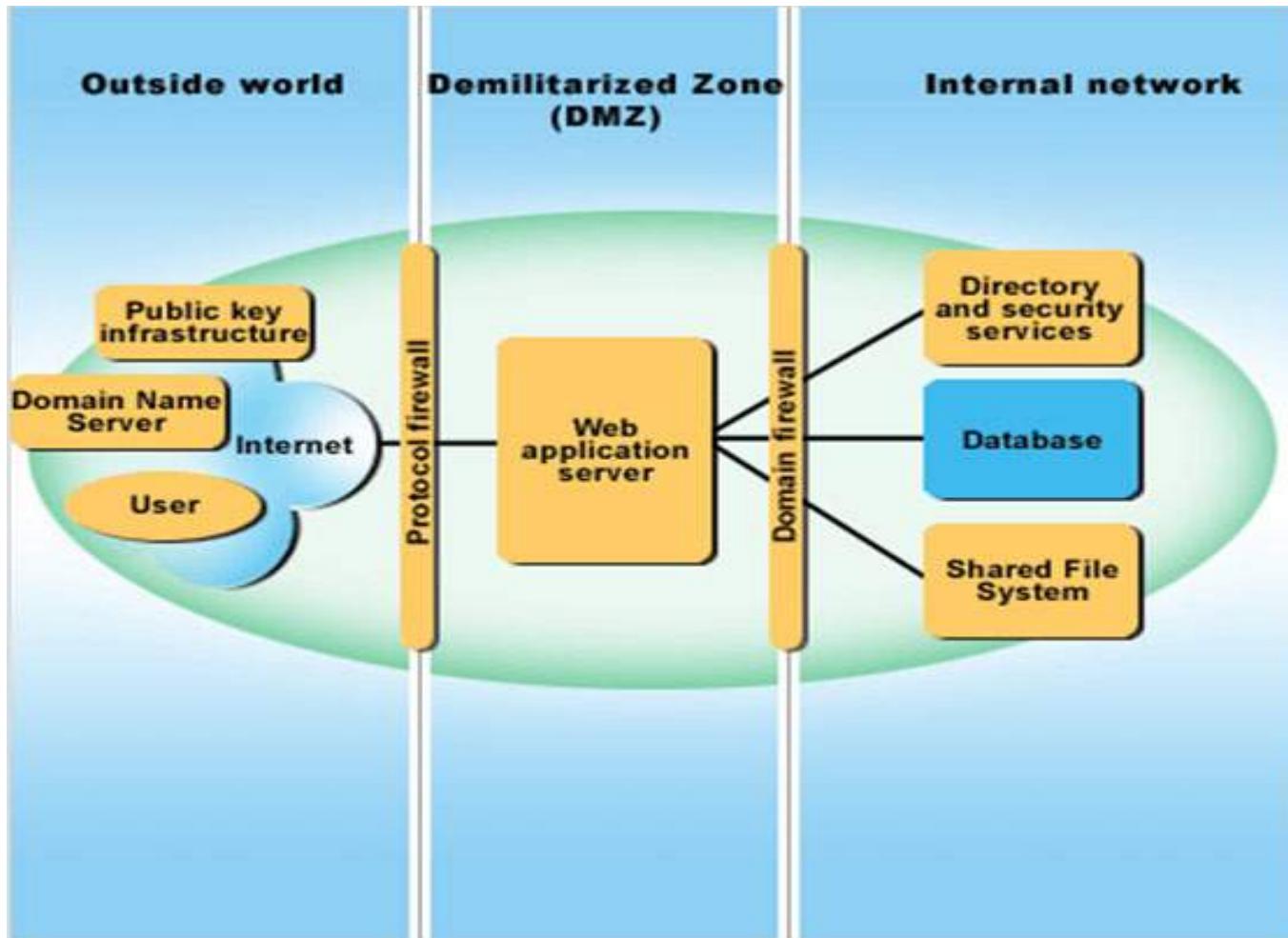
<https://securityboulevard.com/2020/03/types-of-backup-understanding-full-differential-and-incremental-backup/>



AVAILABILITY PATTERNS

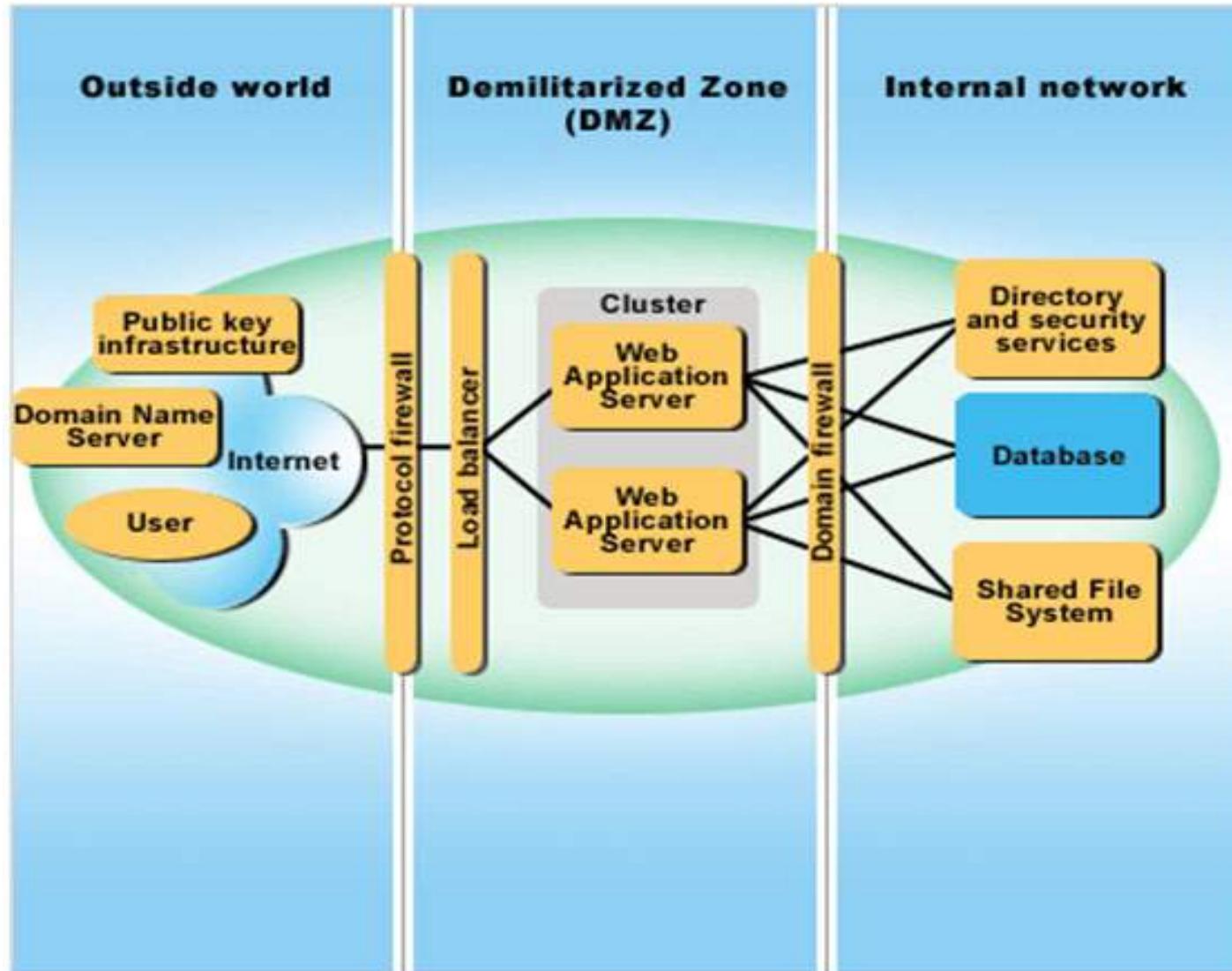


Availability Patterns (Basic Web app)



Source: <https://www.redbooks.ibm.com/redbooks/pdfs/sg246303.pdf>

Availability Patterns (Load Balanced Web app)

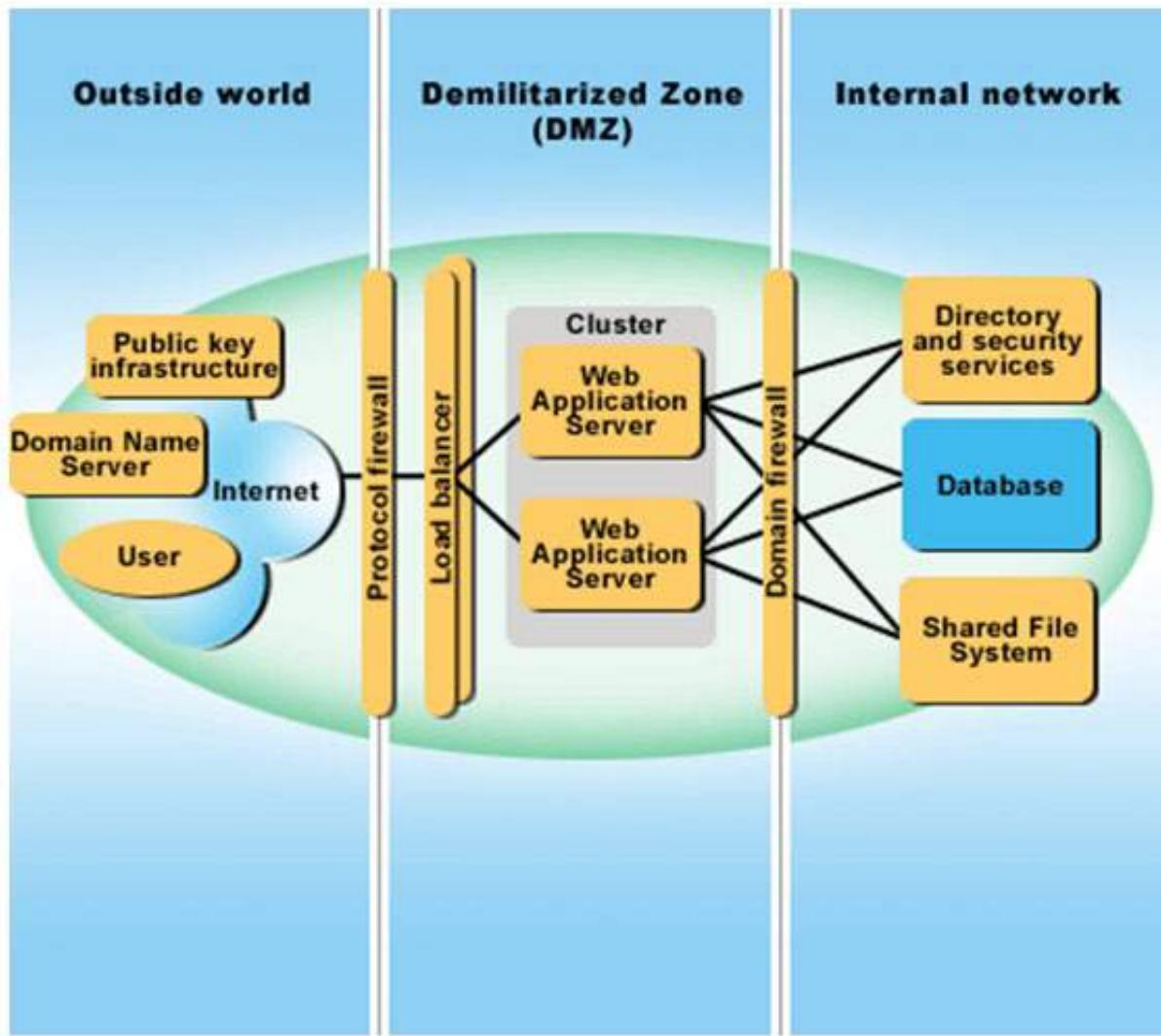


Source: IBM



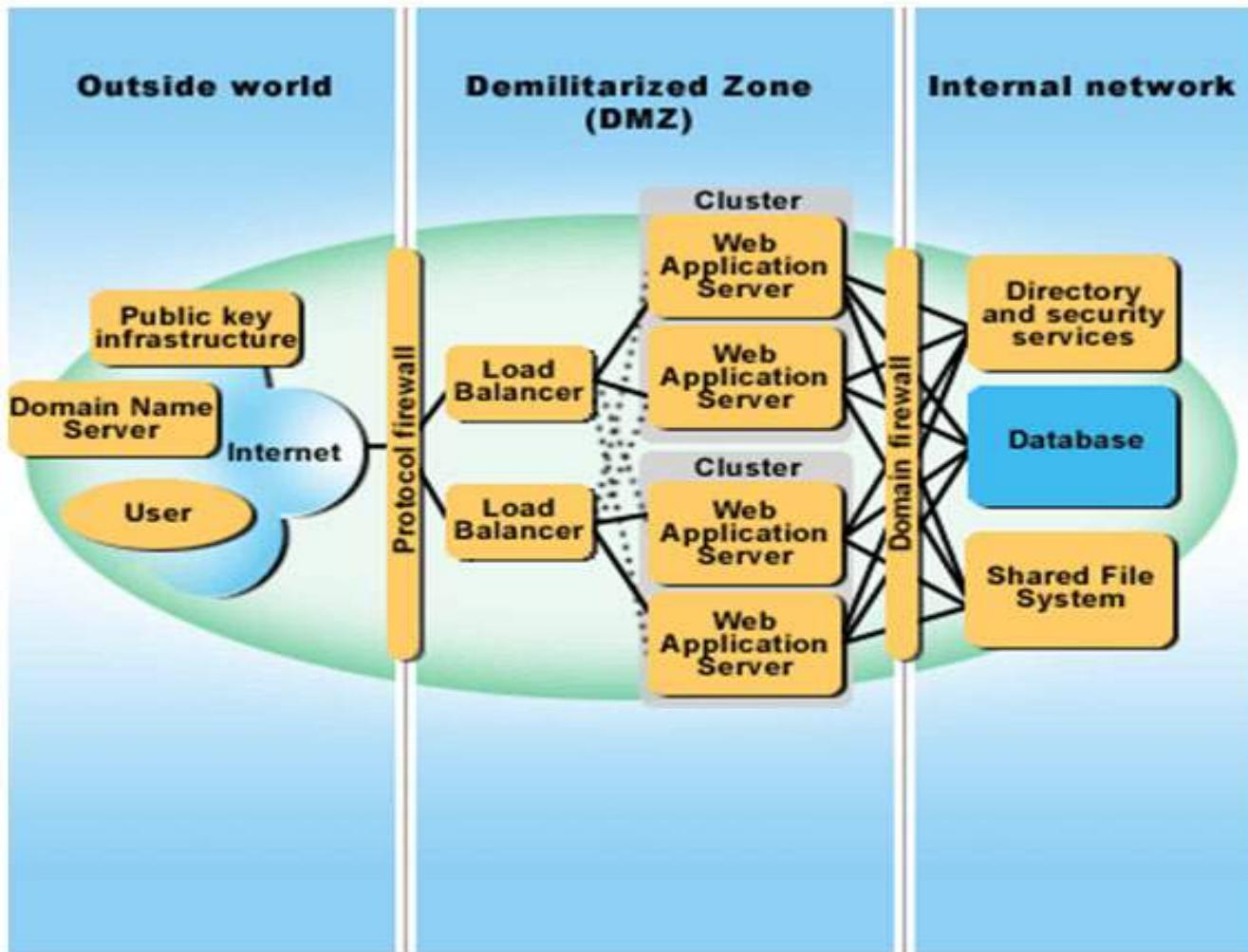
Availability Patterns

(Load Balanced with Hot Standby)





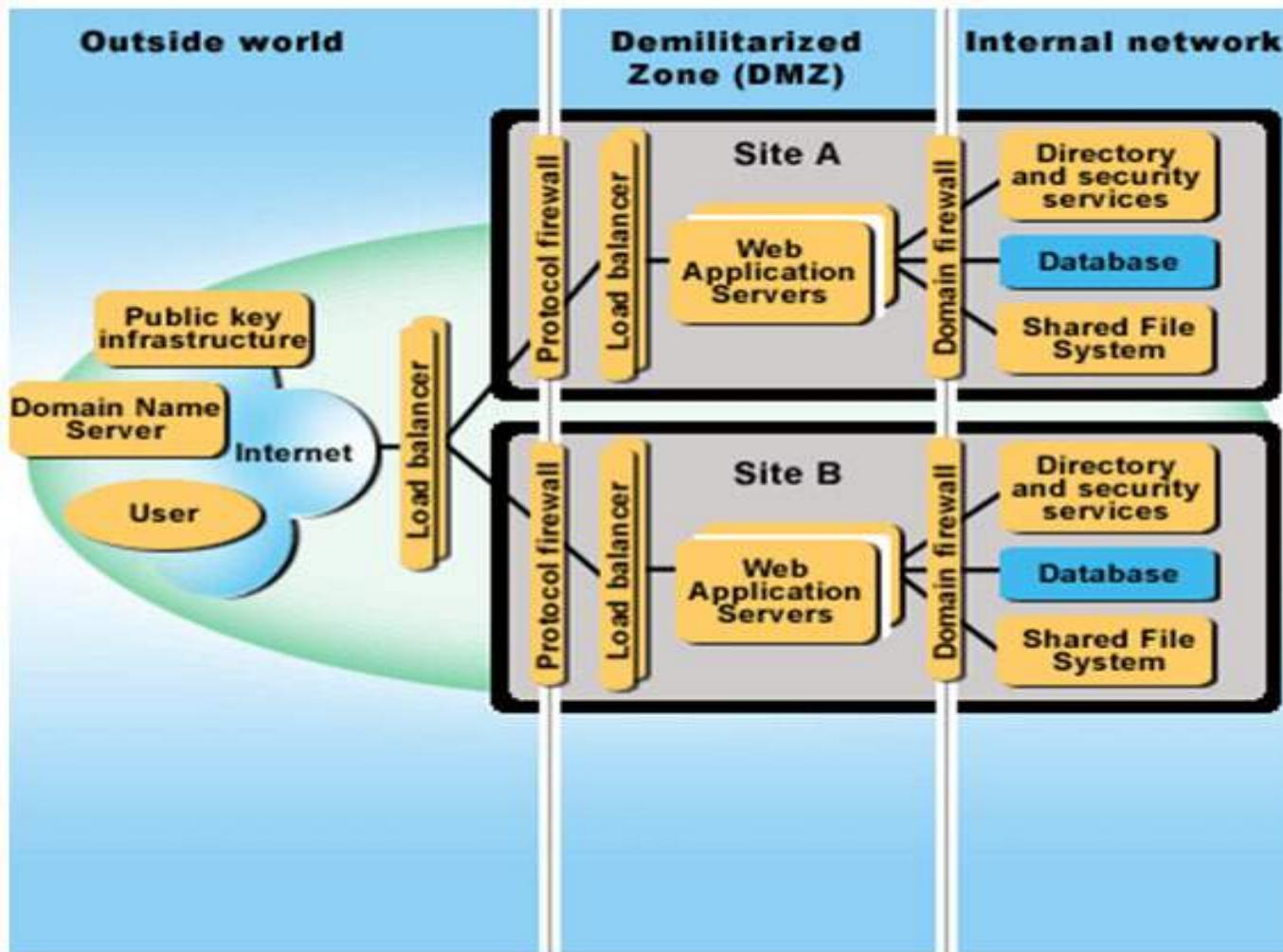
Availability Patterns (High Availability)



Source: IBM



Availability Patterns (Wide Area Load Balancing)



Source: IBM



AVAILABILITY:

PRODUCT REFERENCE ARCHITECTURE





Kubernetes

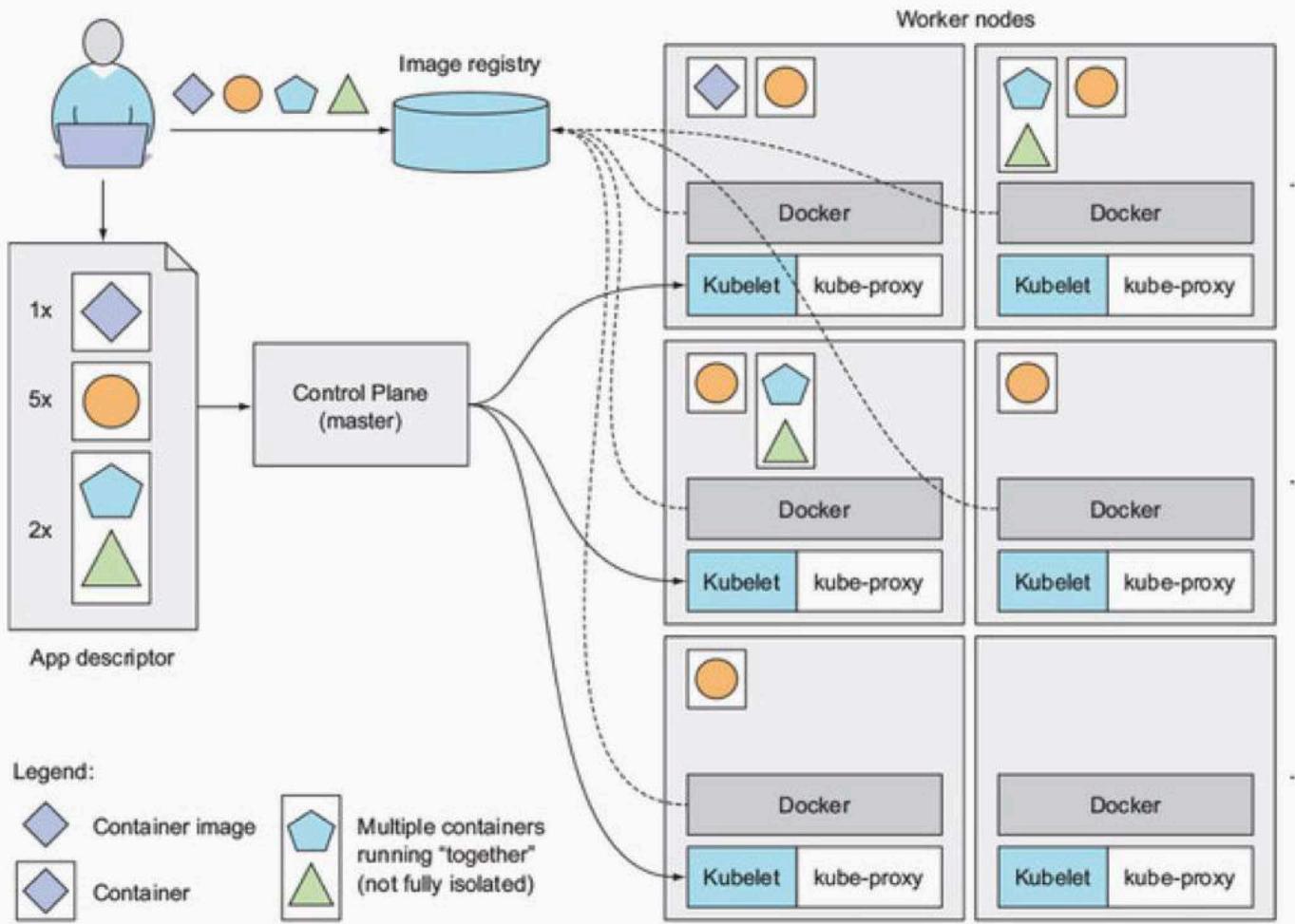
Kubernetes is a portable, extensible, open-source platform for managing containerized workloads and services, that facilitates both declarative configuration and automation.

Advantages:

- Declarative
- Simplifying application deployment
- Achieving better utilization of hardware
- Auto scaling
- Health check and auto healing

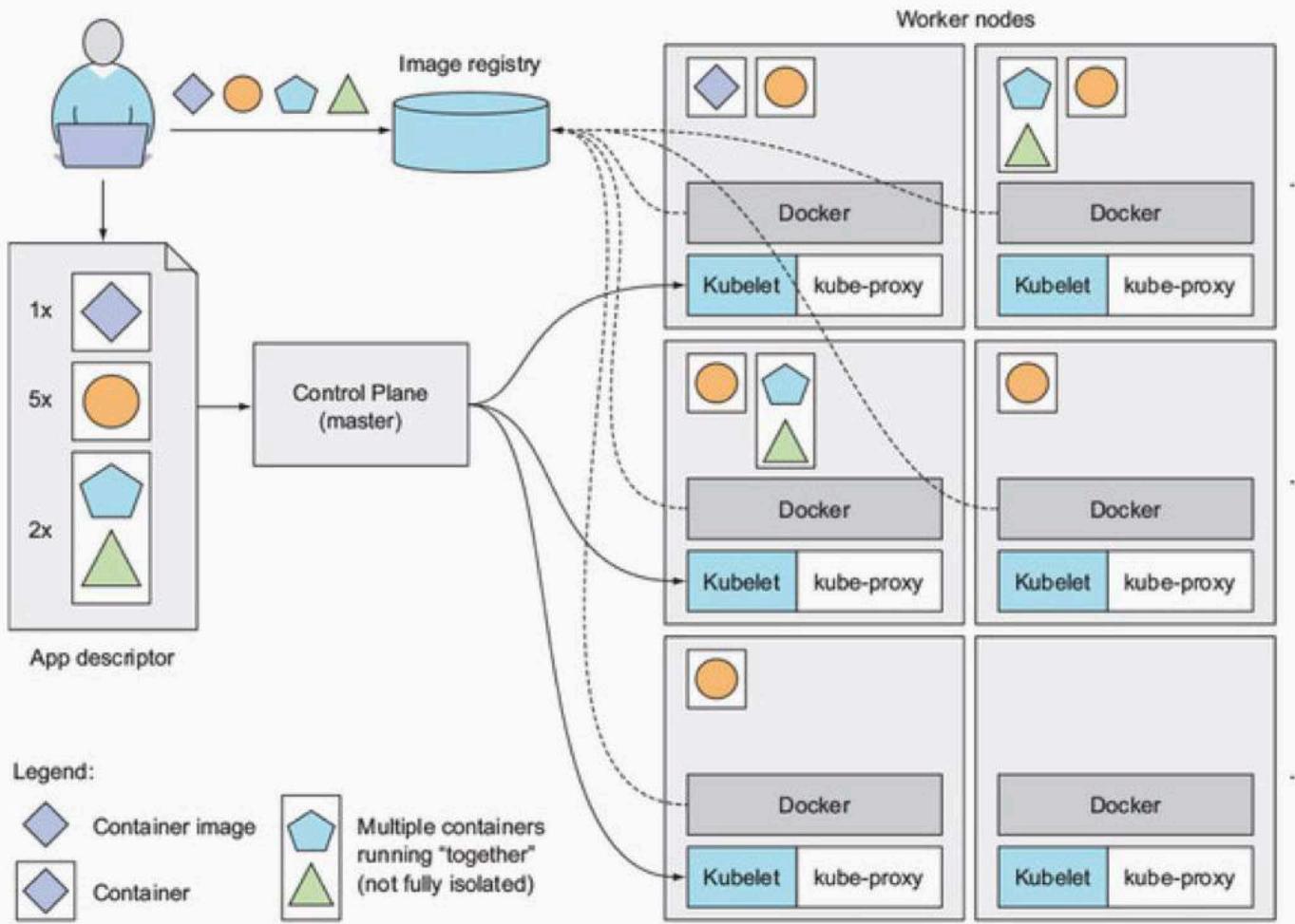


Kubernetes



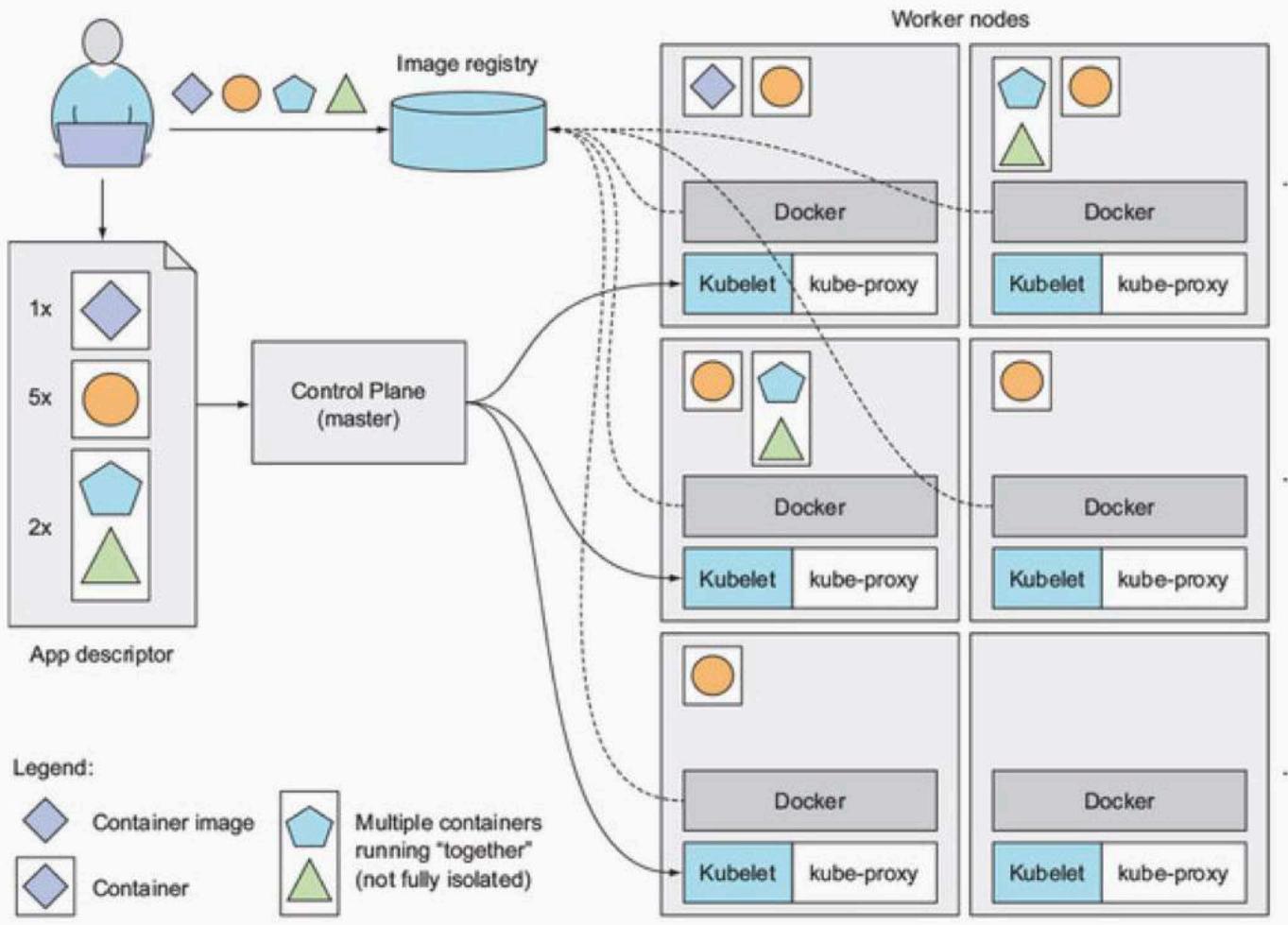


Kubernetes



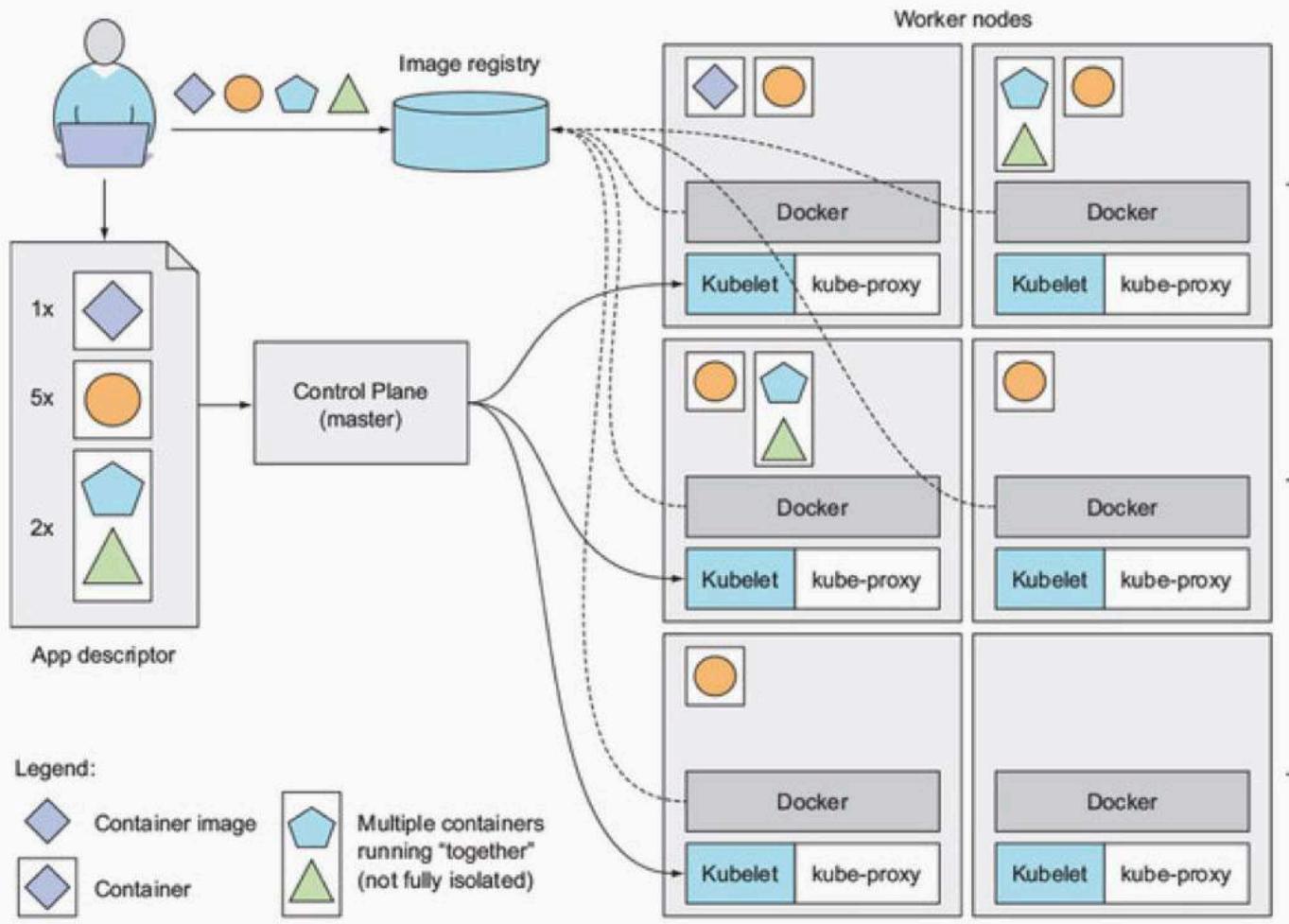


Kubernetes



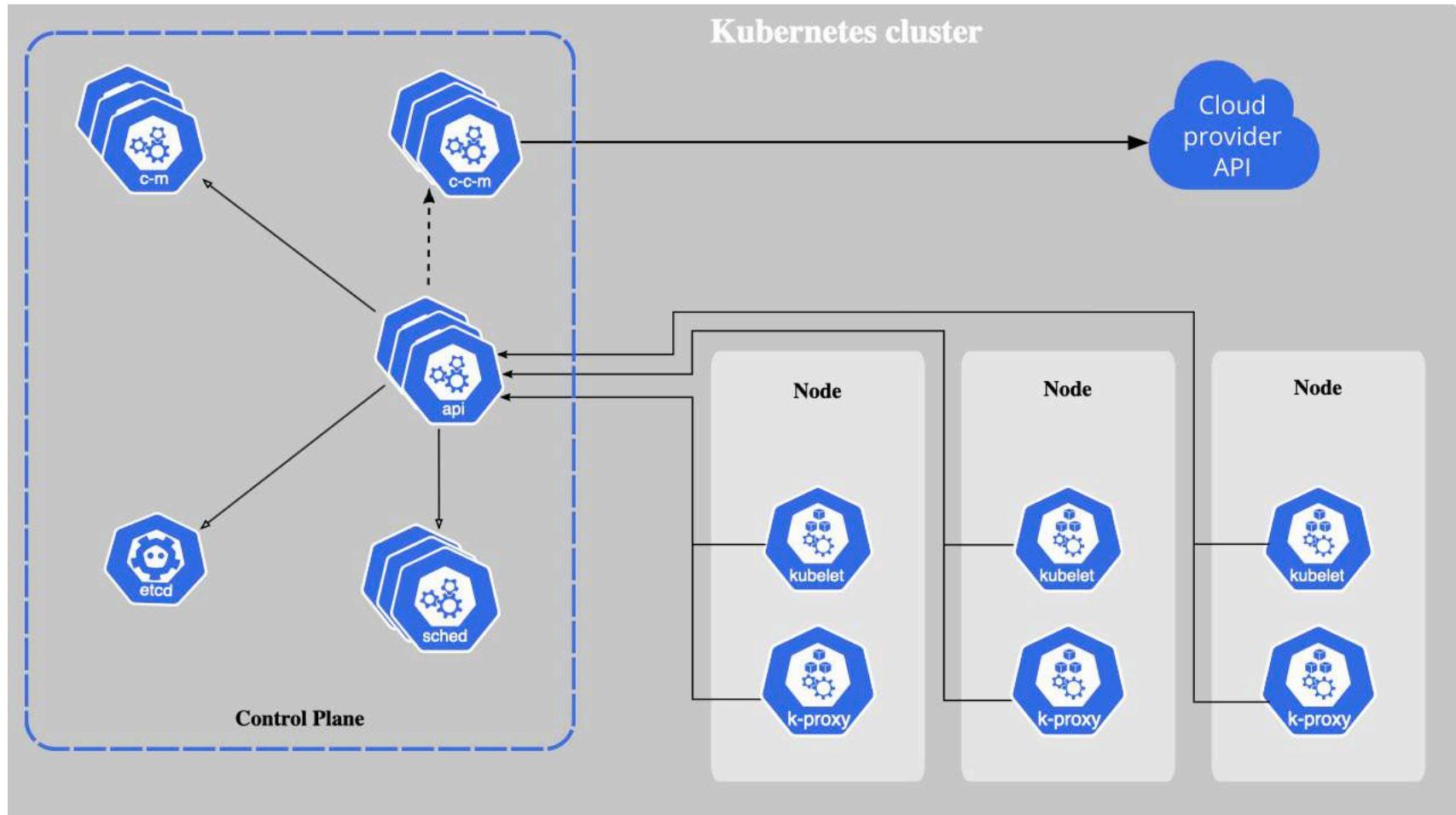


Kubernetes





Kubernetes





Kubernetes

- Node controller
 - Responsible for noticing and responding when nodes go down.
- Job controller
 - Watches for Job objects that represent one-off tasks, then creates Pods to run those tasks to completion.
- Endpoints controller
 - Populates the Endpoints object.
- Service Account & Token controllers
 - Create default accounts and API access tokens for new namespaces.
- Cloud controllers
 - Node controller
 - For checking the cloud provider to determine if a node has been deleted in the cloud after it stops responding.
 - Route controller
 - For setting up routes in the underlying cloud infrastructure.
 - Service controller
 - For creating, updating and deleting cloud provider load balancers.



Kubernetes (Example of declaration)

```
apiVersion: v1
kind: Pod
metadata:
  name: kubia-manual
spec:
  containers:
    - image: luksa/kubia
      name: kubia
      ports:
        - containerPort: 8080
          protocol: TCP
```

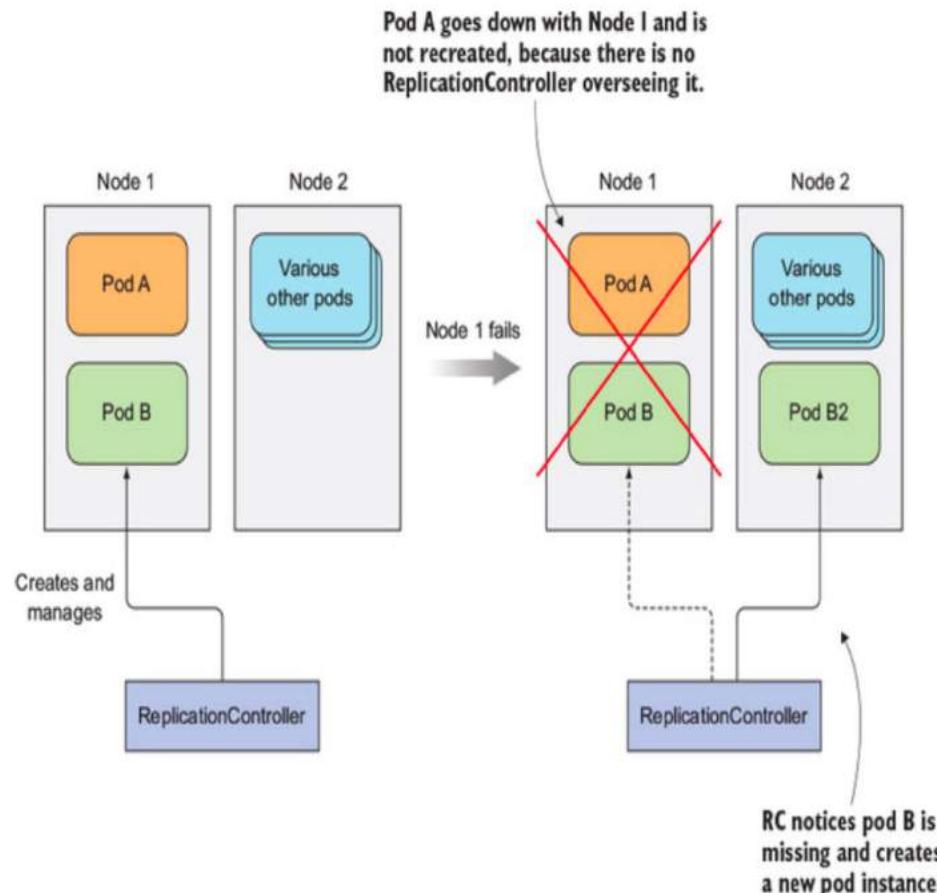
An example of a very basic pod resource declaration.

```
apiVersion: v1
kind: pod
metadata:
  name: kubia-liveness
spec:
  containers:
    - image: luksa/kubia-unhealthy
      name: kubia
      livenessProbe:
        httpGet:
          path: /
          port: 8080
```

An example of a very basic pod resource declaration with liveness probe.



Kubernetes (ReplicaSet/Controller)



```
apiVersion: apps/v1beta2
kind: ReplicaSet
metadata:
  name: kubia
spec:
  replicas: 3
  selector:
    matchLabels:
      app: kubia
  template:
    metadata:
      labels:
        app: kubia
    spec:
      containers:
        - name: kubia
          image: luksa/kubia
```



Kubernetes (ReplicaSet)

- With ReplicaSet resource, if one of the pod is terminated, the ReplicaSet will automatically start a new one.

\$ kubectl get pods				
NAME	READY	STATUS	RESTARTS	AGE
kubia-53thy	1/1	Terminating	0	3m
kubia-oini2	0/1	ContainerCreating	0	2s
kubia-k0xz6	1/1	Running	0	3m
kubia-q3vkg	1/1	Running	0	3m



Future of Kubernetes

- <https://acloudguru.com/blog/engineering/kubernetes-is-deprecating-docker-what-you-need-to-know>
- <https://blog.sighup.io/how-to-run-kubernetes-without-docker/>



Redundancy in Database Servers

- Database server has two main components
 - Database server process
 - Handle requests and queries to access the data
 - Database storage
 - The data stored in some kind of storage
- Database redundancy has to cater to both components
- The models:
 - Active/Active vs. Active/Passive
 - Shared Disk vs. Shared Nothing



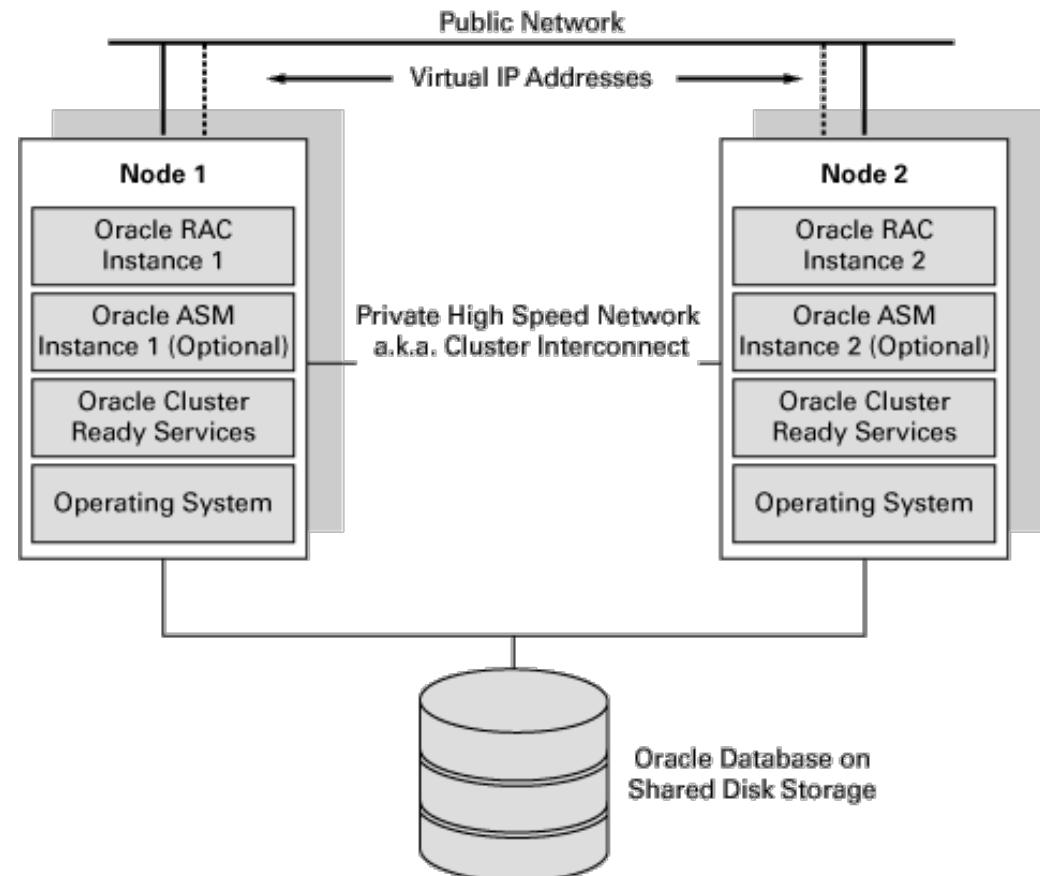
Redundancy in Database Servers

- Database server has two main components
 - Database server process
 - Handle requests and queries to access the data
 - Database storage
 - The data stored in some kind of storage
- Database redundancy has to cater to both components
- The models:
 - Active/Active vs. Active/Passive
 - Shared Disk vs. Shared Nothing



Oracle Real Application Cluster

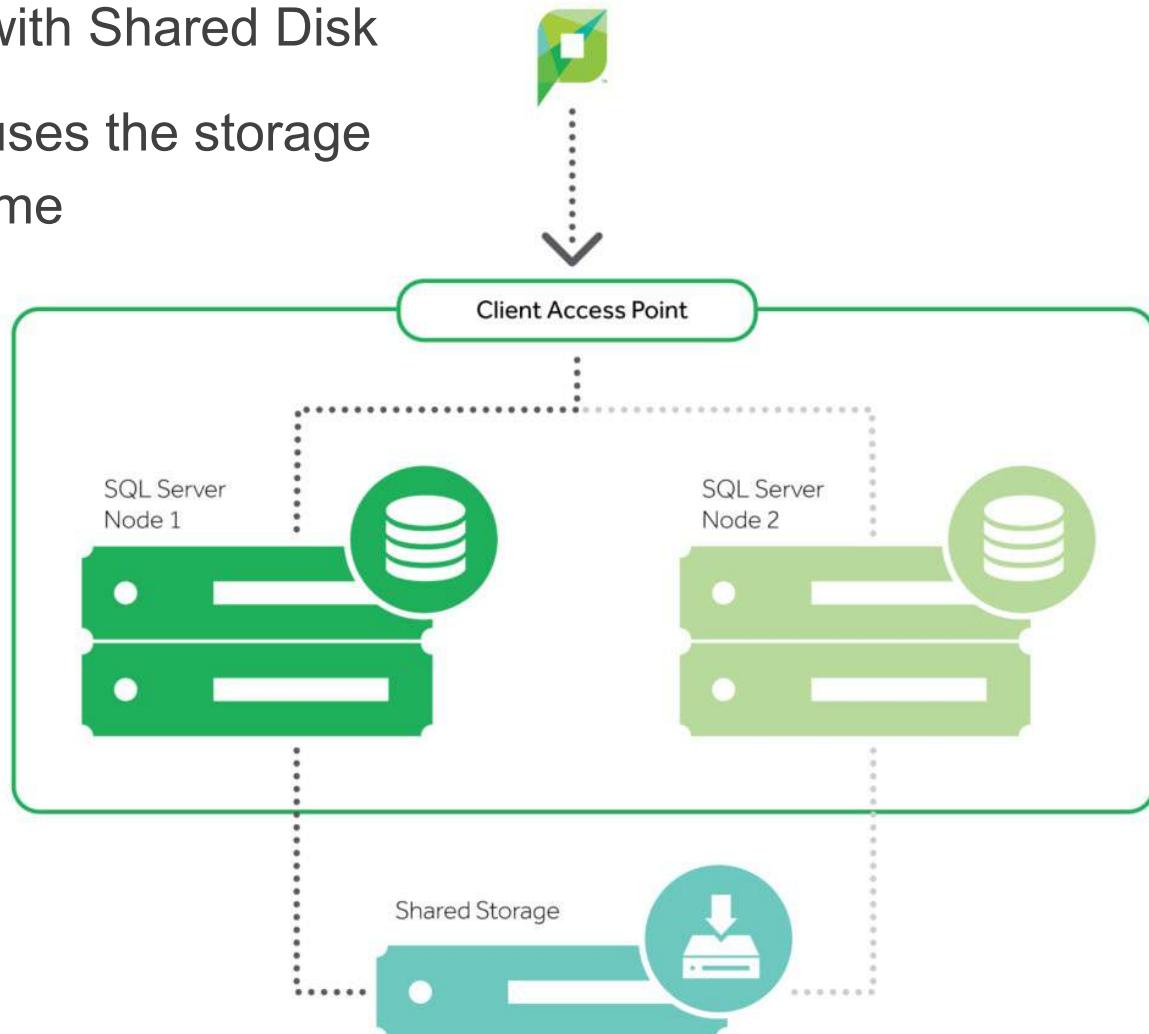
- Active/Active with Shared Disk
- Handles server redundancy by having redundant nodes for database process
- Shared Disk – so disk redundancy is delegated to the hardware





SQL Server Always On Failover Clustering

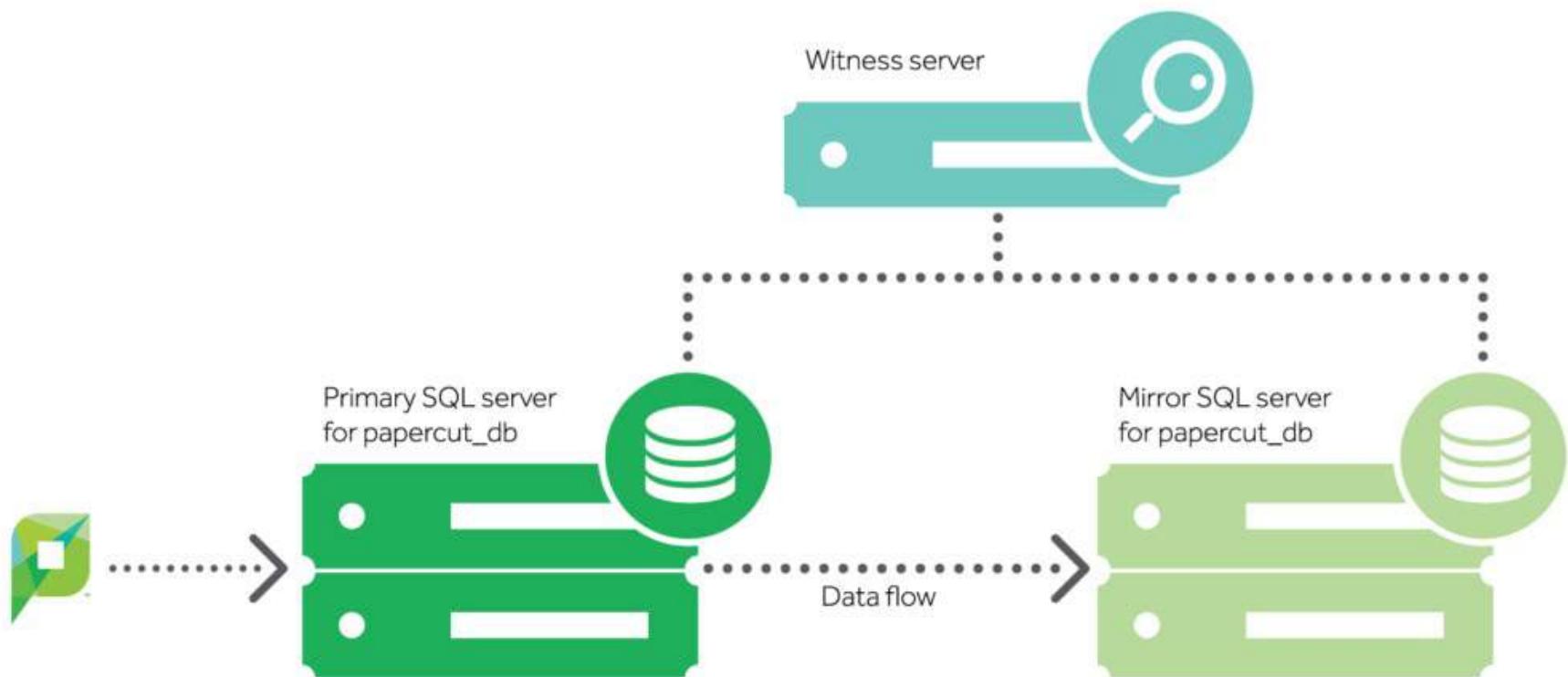
- Active/Passive with Shared Disk
- Only one node uses the storage at any point in time





SQL Server Database Mirroring

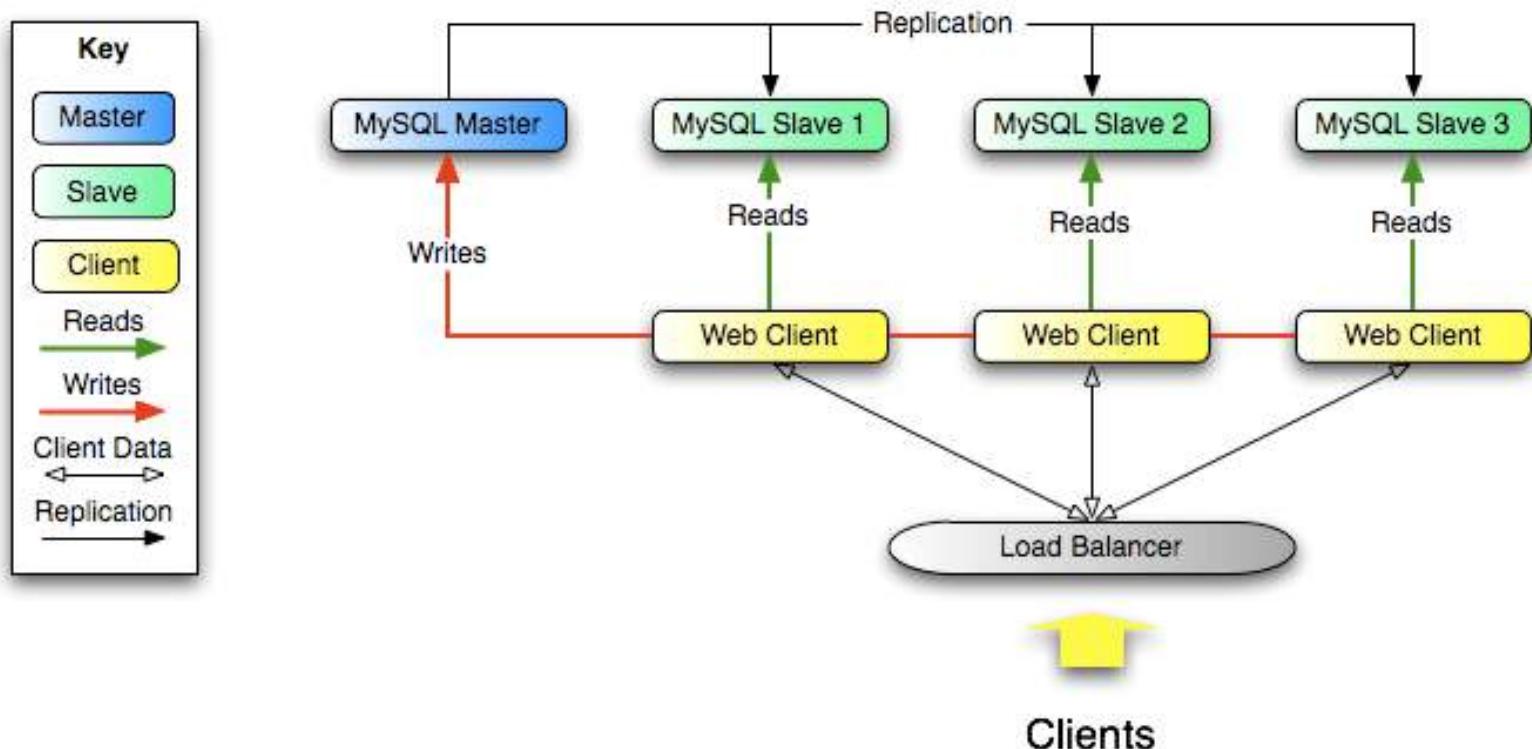
- Active/Passive with Shared Nothing





MySQL Master – Slave Replication

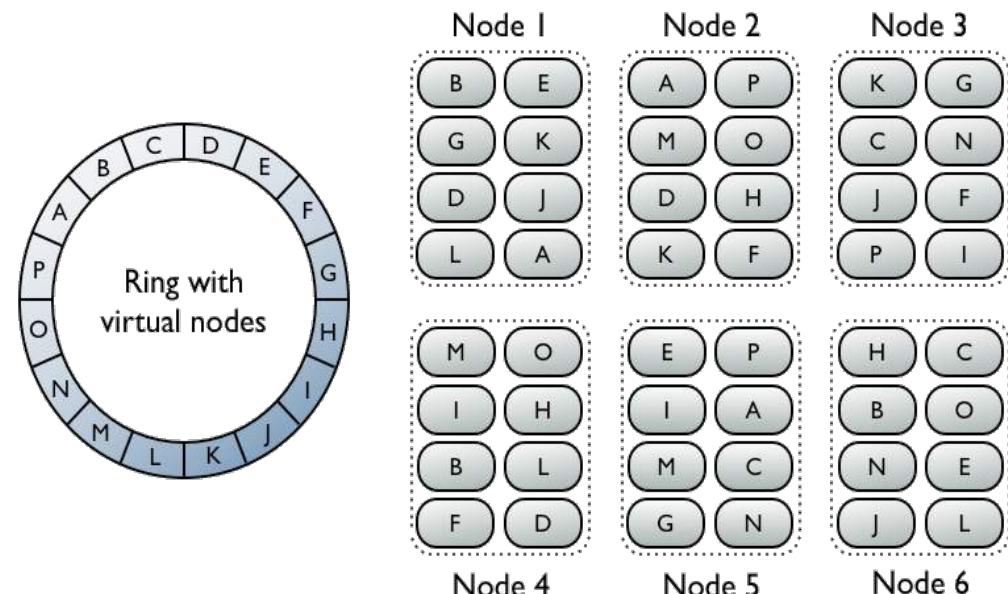
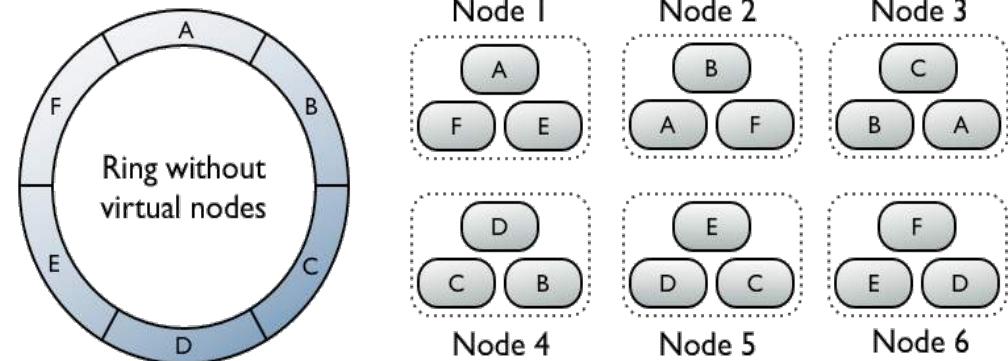
- Active/Active with Shared Nothing
- Master handles write, and slaves are redundant to handle reads
- Slaves have to be "promoted" to master for master redundancy





Cassandra Ring Based Replication

- Active/Active with Shared Nothing
- The number of replica can be configured
- Virtual nodes are automatically rebalanced when new node is added or removed.





Reference Architecture: AWS



NUS
National University
of Singapore

ISC

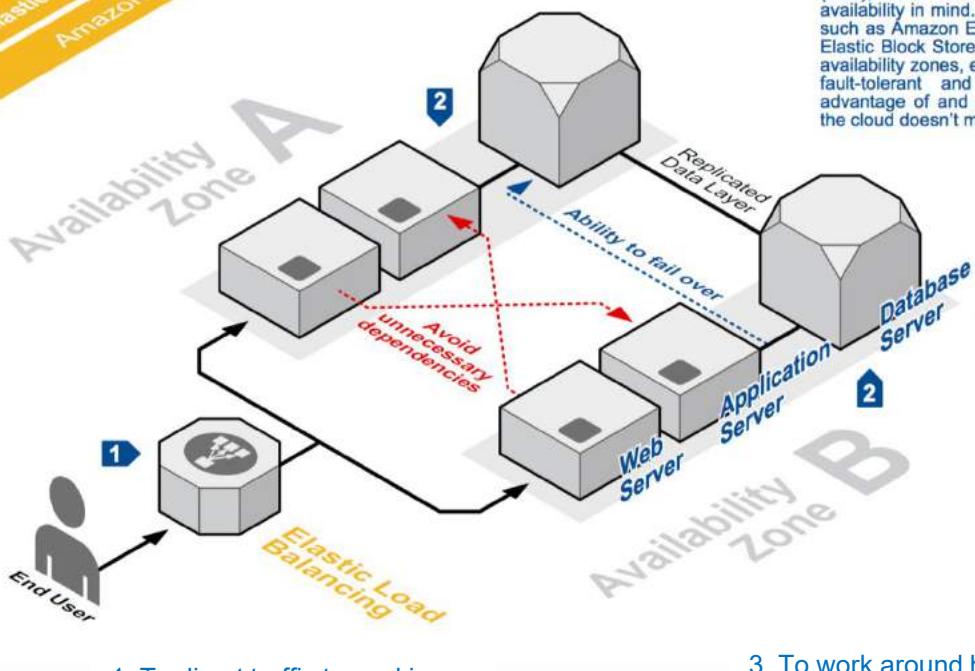
AWS Reference Architectures
Amazon EC2
Amazon EBS
Elastic Load Balancing
Amazon S3

FAULT TOLERANCE & HIGH AVAILABILITY

Amazon Web Services provides services and infrastructure to build reliable, fault-tolerant, and highly available systems in the cloud. These qualities have been designed into our services both by handling such aspects without any special action by you and by providing features that must be used explicitly and correctly.

Amazon EC2 provides infrastructure building blocks that, by themselves, may not be fault-tolerant. Hard drives may fail, power supplies may fail, and racks may fail. It is important to use combinations of the features presented in this document to achieve fault tolerance and high availability.

2. To protect the application from single location failure with different distinct geographical zones.

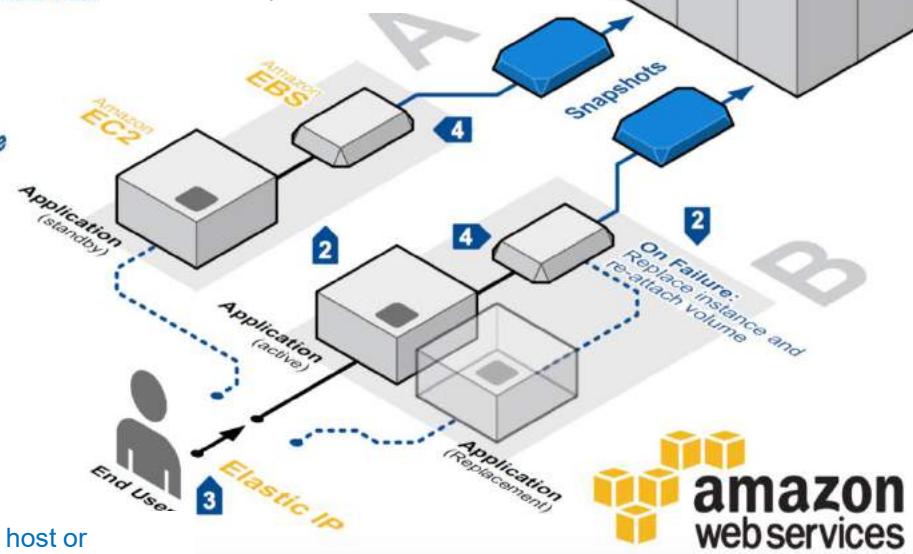


1. To direct traffic to working instances. Instances that fail can be replaced seamlessly behind the load balancer

Fault Tolerance and High Availability of Amazon Web Services

Most of the higher-level services, such as Amazon Simple Storage Service (S3), Amazon SimpleDB, Amazon Simple Queue Service (SQS), and Amazon Elastic Load Balancing (ELB), have been built with fault tolerance and high availability in mind. Services like Amazon Elastic Compute Cloud (Amazon EC2), Amazon Elastic Block Store (EBS), provide availability zones, elastic IP addresses, and are fault-tolerant and highly available. The advantage of using these services in the cloud is that the cloud doesn't make it fault-prone.

4. Data is replicated within the same availability zone for redundancy and point-in-time snapshots can be made and replicated to other zones.



Source: <https://aws.amazon.com/architecture/>



Cloud Redundancy Strategies

- **Hardware-Level Redundancy**
- **Process Redundancy**
 - Do you map out each of the processes within a company and determine which require the highest availability and which are less critical to the business?
 - How processes use and share resources?
- **Network Redundancy**
 - Are there multiple routes to the internet?
 - If one carrier becomes unavailable for some reason, can another carrier pick up the load and handle the traffic?
- **Geographic Redundancy**
 - Replicates data between two (or more) physically disparate locations?
 - Network traffic should be split amongst locations for geo-redundancy?



Summary

- The need to understand the system components and failure causes to provide an architectural plan.
- Key concepts learned include: Availability, Reliability and Fault Tolerance
- Key architectural decisions including ideas for redundancy, back up and patterns.



ARCHITECTING SOFTWARE SOLUTIONS

DESIGN FOR PERFORMANCE IN ARCHITECTURE

Instructor: Darryl Ng

Email: darryl.ng@nus.edu.sg

Total slides: 108



Objectives

- Understand key performance metrics and their relationship with each other
- Able to analyze a software architecture based on the understanding of performance concepts and metrics
- Understand best practices in designing an architecture with good performance
- Understand key principles in documenting performance characteristics of an architecture



Topics

- Introduction
- Performance Metrics
 - Response Time
 - Throughput
 - Utilization and Load
 - Exercises



Performance

Definition:

Performance in IT Solution context is the amount of useful work accomplished by a computer system. In general, computer performance is estimated in terms of accuracy, efficiency and speed of executing computer program instructions.

Indicators of good performance:

- Short response time for a given piece of work.
- High throughput (rate of processing work).
- Low utilization of computing resource(s).
- High availability of the computing system or application.
- Fast (or highly compact) data compression and decompression.
- High bandwidth.
- Short data transmission time.



Performance Metrics

- Response Time (speed)

Response time is the elapsed time between an inquiry on a system and the response to that inquiry.

Typically consists of:

- Service Time
 - *How long it takes to do the work requested.*
- Waiting Time
 - *How long the request has to wait for requests queued ahead of it before it gets to run.*
- Transmission Time
 - *How long it takes to move the request to the computer doing the work and the response back to the requestor.*

Note: For further detailed analysis the above three *Time Measures* may further be fragmented to items such as network time, rendering time, processing time etc.



Performance Metrics

- Throughput
 - Definition

Measure of a computer system's overall performance in sending data through all its components, such as the processor, buses, storage devices, network. Throughput is more meaningful indicator of system performance than raw clock speed or disk seek time or other advertised parameters.

- Measures:
 - Number of events happening within a time duration
 - Arrival Rate / Input Rate / Load

The number of request coming into the system within a period of time

- Response Rate/ Output Rate/ Throughput

The number of response going out of the system within a period of time.



Performance Metrics - Questions

- *A cafe owner wants to know their throughput rate for coffee drinks over the course of one 12-hour day. They count the number of drinks made over the course of that period of time and determine that 300 coffee drinks were made and sold.*
- *A factory manager wants to know the throughput rate for bolts manufactured per second. They know that every minute, 3000 bolts are in production and in stock combined.*
- *The manager of a cell phone store knows that over the course of four hours, 20 customers enter the store, wait for assistance and receive service before they leave. What is their throughput rate?*



Performance Metrics

- **Utilization (Resource Usage)**

- Definition

Utilization is the percentage of time that a component is actually occupied, as compared with the total time that the component is available for use.

- Measure

- Used capacity / total capacity.

- Significance

- Whenever a system resource, such as a CPU or a particular disk, is occupied by a transaction or query, the resource is unavailable for processing other requests. Pending requests must wait for the resources to become available before they can complete.



Performance Metrics

- Load

Can loosely mean one of these things:

- Request rate
- Utilization
- Throughput
 - Because throughput = request rate if the system is not saturated and assuming 1 request – 1 response

- Latency

- The minimum amount of time to send data from one component to another component.
- Some design decision may trade off latency with bandwidth.



Performance Metrics

- Think Time

The time taken by user between two successive actions such as navigation from one page to another, providing data inputs etc.

This is typically the time users use to “think” (not use the system) in between requests.

Uses:

- Though not directly related to performance assessment, think time impacts system utilization and throughput calculations
- Quite useful in context of Performance Testing and Capacity Planning – rather than in performance assessment per se.



Response Time

- Serial



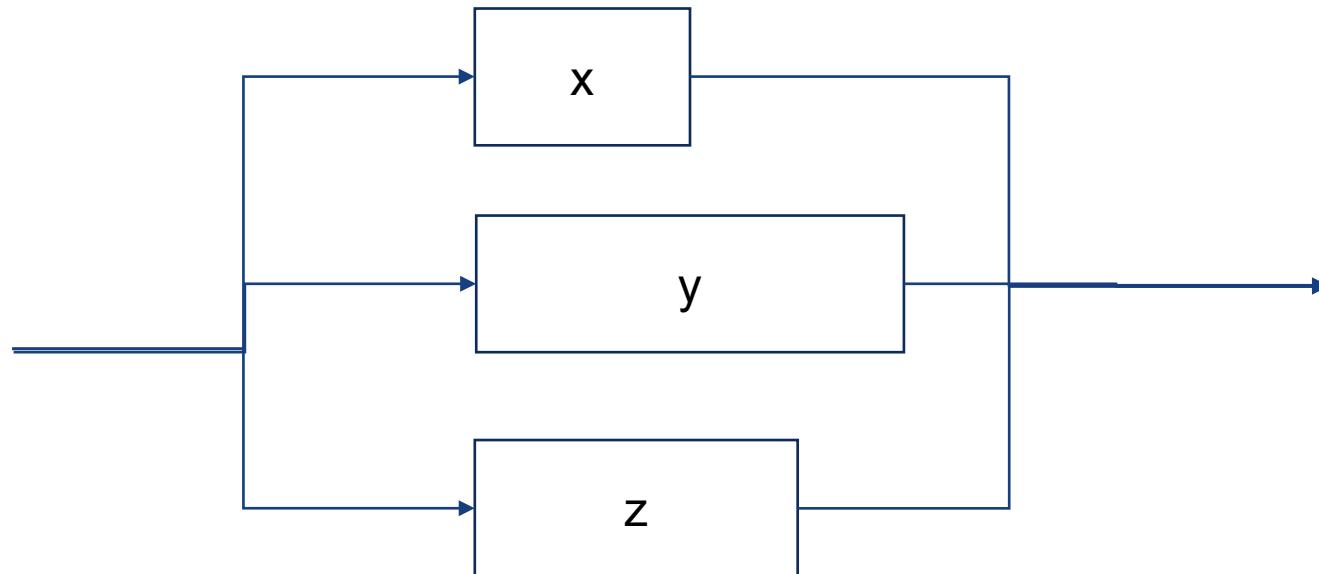
A request has to be processed by 3 components organized using a multi tier architecture with average response time of x, y and z

Average total response time = ?



Response Time

- Parallel



A request has to be processed by 3 components that can process the request simultaneously.

Average total response time = ?



Throughput

- Serial



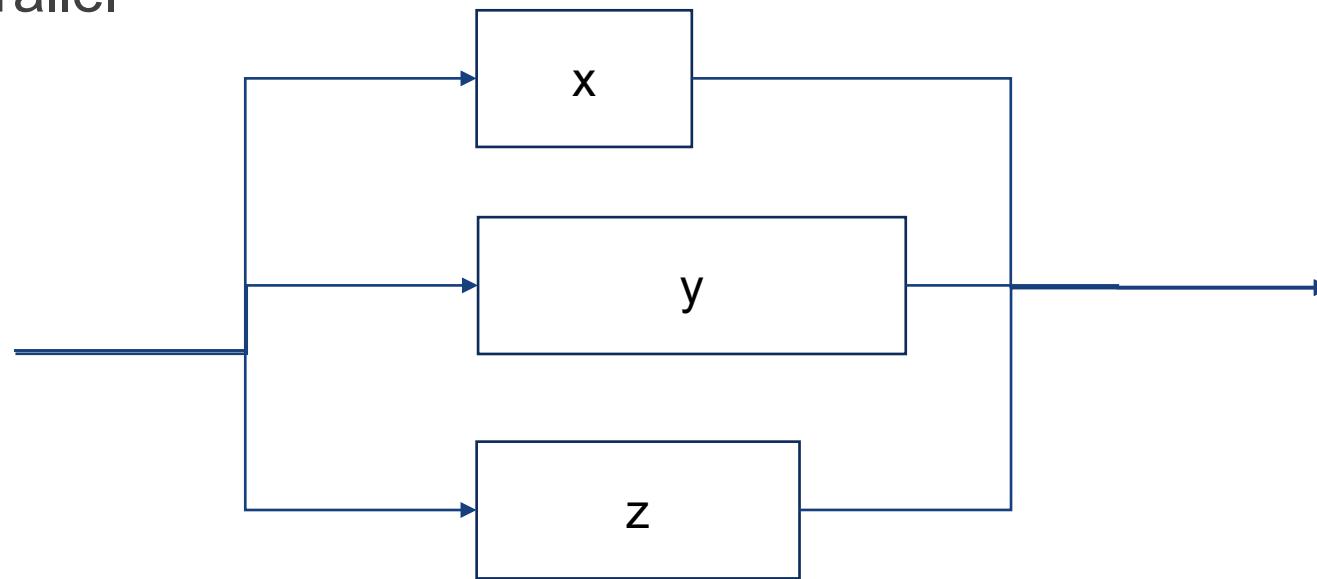
Three components work in a pipeline each with throughput of x, y, and z.

Total throughput = ?



Throughput

- Parallel



When requests are load balanced between 3 instances of components with throughput of x, y and z

Total throughput = ?



Throughput Example

- You work part time to prepare burger. It takes you 1 minute to prepare a burger, and another 30 seconds to put it in the packaging box
- How many burger can you prepare in an hour?
- What is your maximum throughput?
- What is your minimum throughput?



Response Time vs Throughput

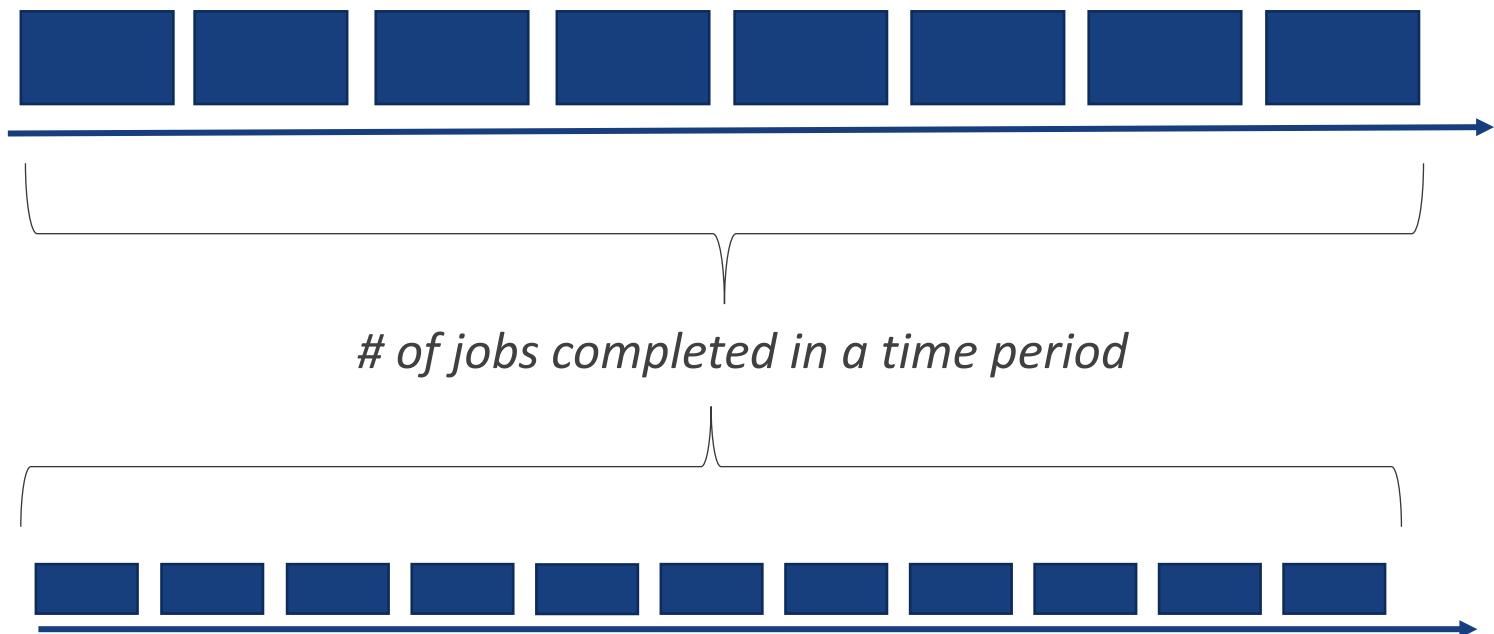
- You are given a training by the company and now you are twice as fast.
- What is your maximum throughput now?
- What is your minimum throughput now?

What can you conclude regarding the relationship between speed (response time) and throughput?



Time and Throughput

- Faster response time generally leads to better throughput





Exercise

- You are developing the backend service of a multiplayer mobile game. The game app will send events to the service (e.g. hit the monster, open the treasure chest) frequently and the server should respond with some information back
- You assume that the round trip network latency between the app and the server is about 200 ms on average, and the server target service time is 50 ms.
- You assume that the think time of the app is 3s which means that the player will encounter some kind of events every 3s.
- Calculate the average load that your server have to handle if there are 1000 online players behaving regularly as described above.

*See if you can do it without referring to the formula.
The formula can be found at the end.*



Request Rate and Throughput

- If a system has a continuous load of 2 requests/sec and it's within the system capacity, what is the throughput of the system (how many responses would the system produce every second)?
- Assume 1 request – 1 response



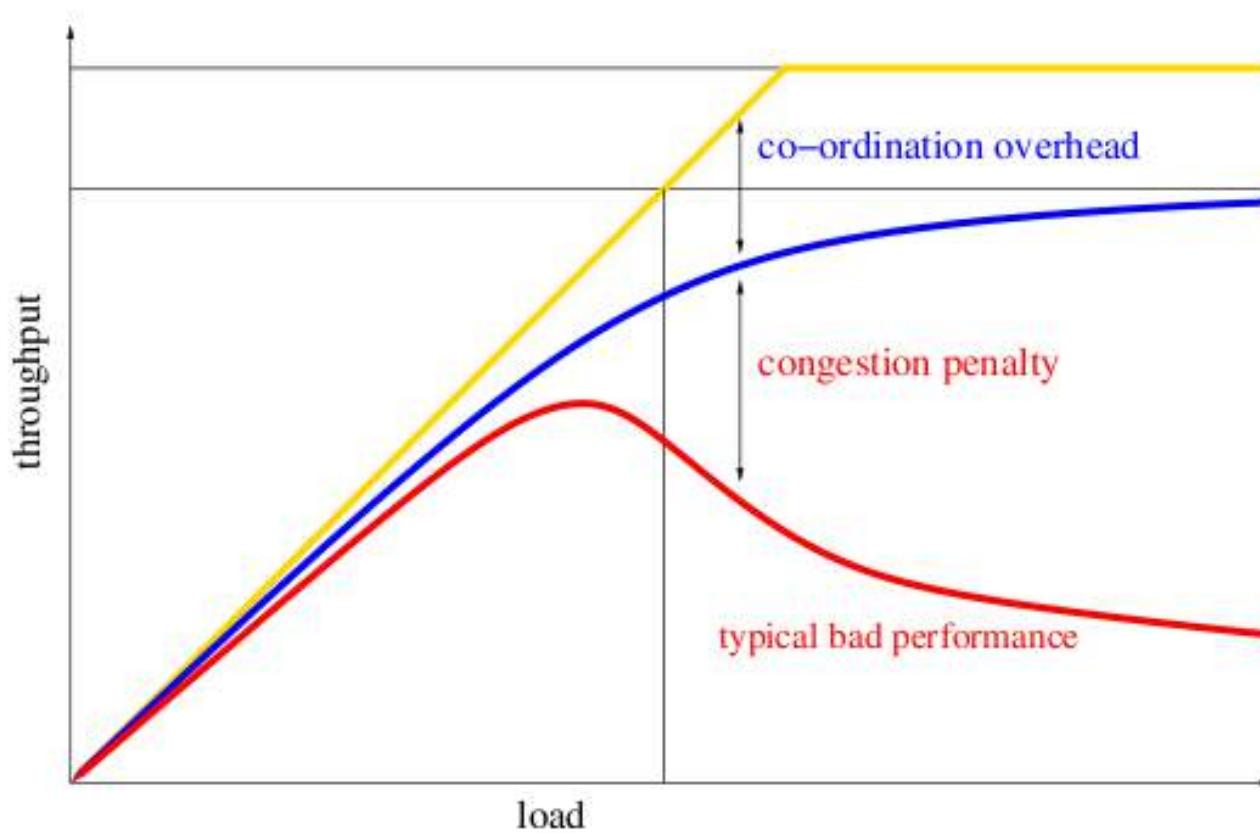
Load and Throughput

- If the system capacity is only able to compute 20 responses every second – what will happen if the request rate goes up to 22 requests/second. What will be the throughput?



Throughput for interactive systems

Throughput vs. load





Utilization, Load and Throughput

- If a system is fully loaded, what should the utilization be?
- If a system is being idle, what should the utilization be?
- If a system has 50% load, what is the system utilization?
 - What is the throughput (compared to the max throughput) ?



Web Application related speeds

- Response Time

Time between the time the user/system send the request and the time when they receive the response

- **Time To First Byte (TTFB)**

- Measured from the time request is sent until the time the first byte of the response is received

- **Time To Last Byte (TTLB)**

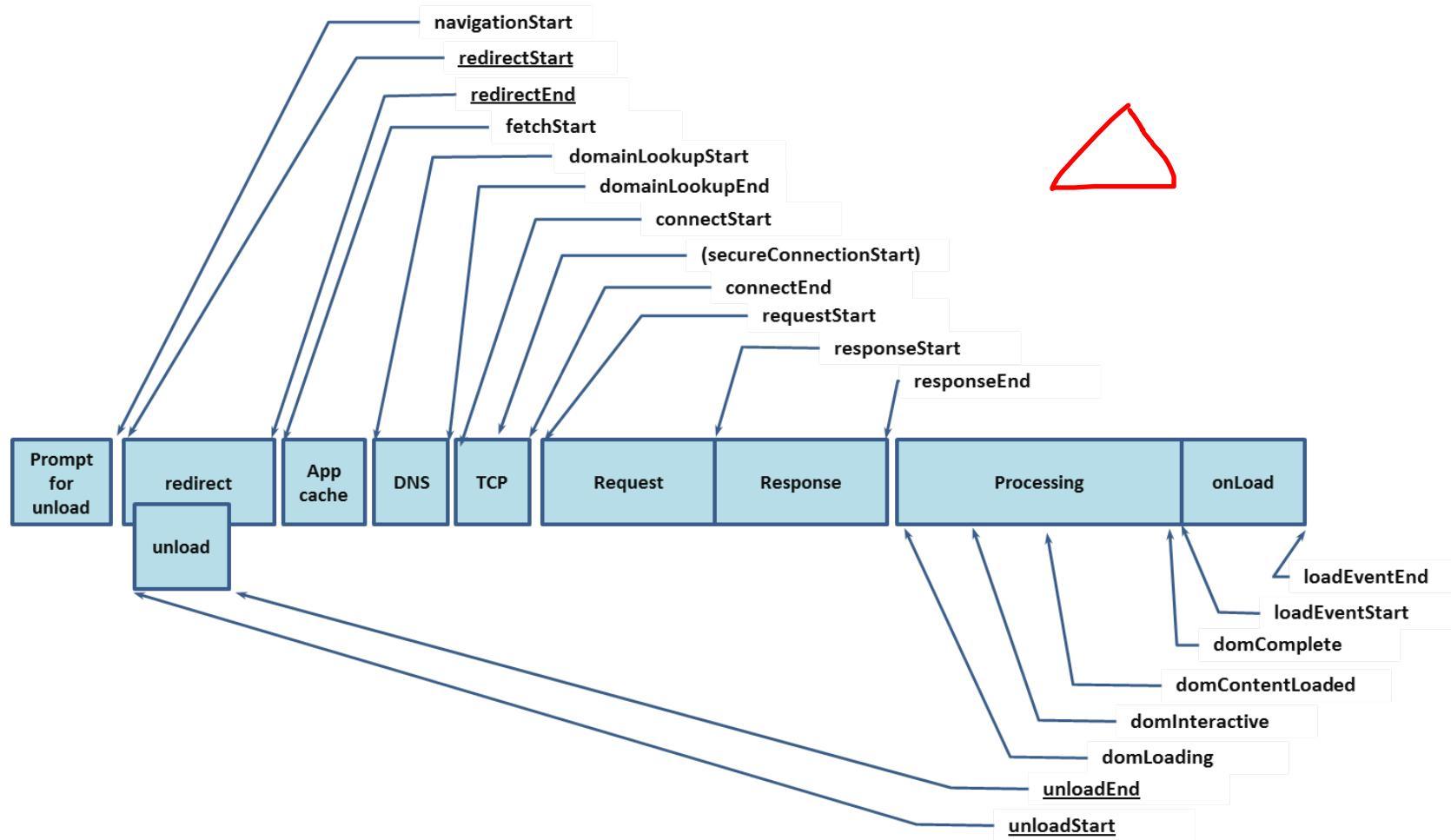
- Measured from the time request is sent until the time the last byte of the response is received

- Think Time

- **How long the user think (not use the system) between requests?**



Web Request-Response Cycle



<https://www.w3.org/TR/navigation-timing>



Simplified Web-Request Response

- **Response Time**
 - Viewed from the perspective of the client (requestor)
 - Time duration from submitting a request until receiving a response
 - Sum of wait time and service time
- **Service Time**
 - Viewed from the perspective of the server
 - Time taken by a server/worker to perform a service
 - Service = processing = computation
- **Wait Time/ Queue Time**
 - Time taken for a request to wait/queue before it is served
 - The difference between response time and service time
- In performing simple analysis, sometimes we ignore latency
 - We can add it into the formula, or include it into service time



Exercise

- Consider an architecture where every request will be inspected by a Level 7 firewall and then going into a reverse proxy before hitting the application server and then the database.
- If we assume that the average performance profile for each of the layers are as follows and a network latency of 200ms, answer the questions in the next slide.

	Wait Time (ms)	Service Time (ms)	Throughput (tps)
Firewall	5 ms	45 ms	10 tps
Reverse Proxy	10 ms	90 ms	16 tps
App server	50 ms	550 ms	6 tps
DB server	30 ms	300 ms	20 tps



Exercise

- How long is the average response time for a browser request – if we assume that the roundtrip network latency from the browser to the server is 200ms?
- What is the expected throughput of the system if we only have one server per type?

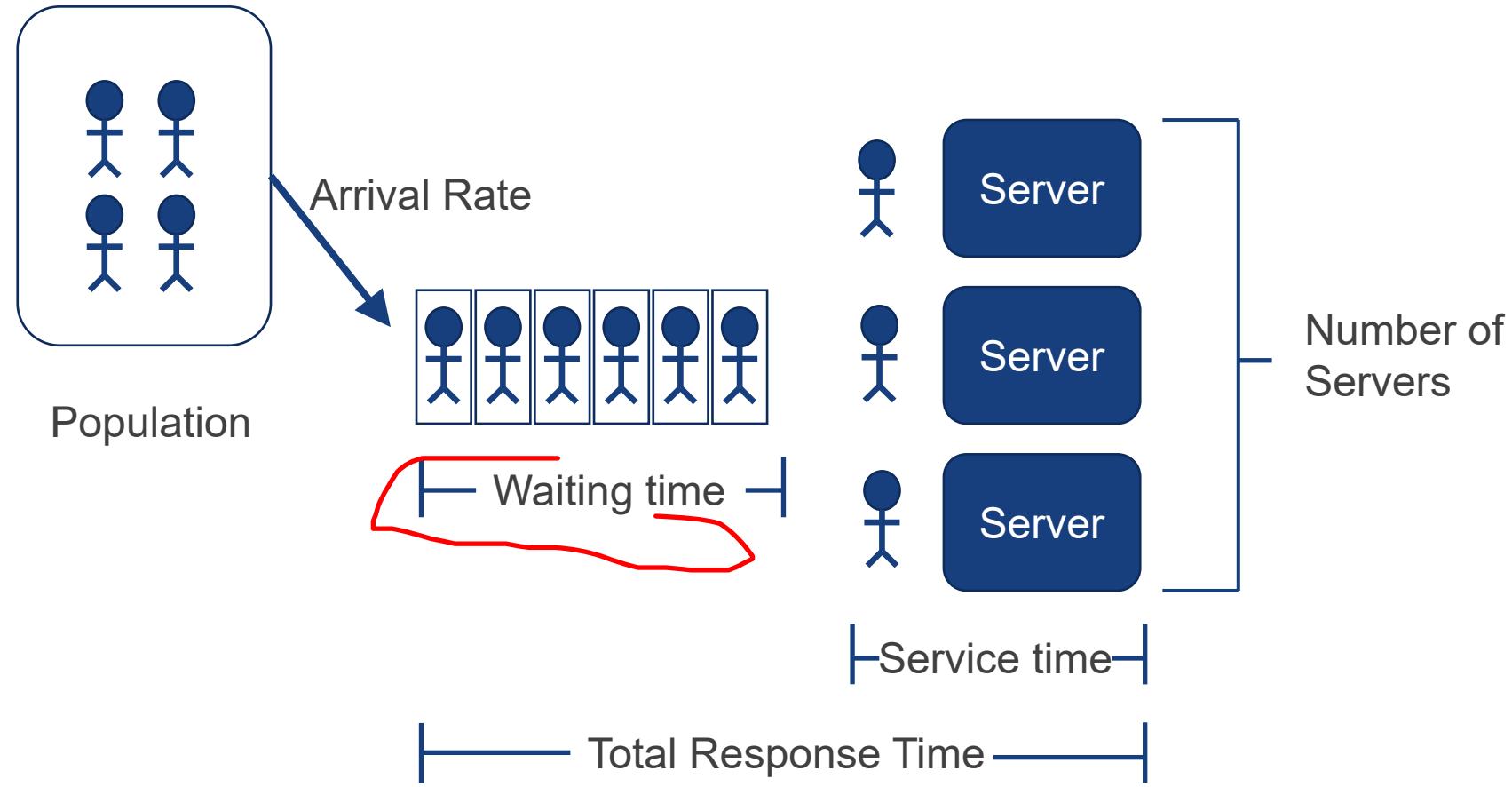


Queuing Theory

- Formal treatment of jobs being processed by a system
- Forms mathematical basis for determining key measures such as:
 - Response Time
 - Throughput
 - Utilization
- Helps determining the system performance and capacity planning to define Solution Architecture.



Queuing Theory



Number of jobs in the system = Number of jobs in queue + Number of jobs in service



Queuing Theory Notations

Symbol	Meaning	Example / Remarks
λ	Mean rate of arrival. <i>(also 1/inter-arrival time)</i>	User requests to a server.
S	Service Time	The time taken by a Server to process the user request
μ	Mean Service Rate <i>Note: $\mu = 1/S$</i>	The throughput of a server
c	The number of parallel servers.	The number of application servers.
ρ	Utilization of server <i>Note: $\rho = \lambda/(c\mu)$</i>	Also the probability that the server is busy (or proportion of time busy)
K	Capacity of queue	Message queue or buffer limits.



Queuing Theory Notations

Symbol	Meaning / Example
P_n	Probability that there are n requests in the system.
L	Mean number of requests in the system.
L_q	Mean number of requests in the queue.
W	Mean wait in the system.
W_q	Mean wait in the queue.



The 2 Sides

- Operational Law

Formulas that are correct regardless of the arrival time distribution and the service time distribution.

Formulas are to be applied to observed values.

The only assumption is that there is no job loss in the process.

- Mathematical

Formulas and derivations that depends on the arrival time distribution and service time distribution.

Most of the analysis assumes exponential (Memoryless or Markovian) distribution due to the ease of analysis.



QUEUE OPERATIONAL LAW



Utilization

- Formula:

$$\frac{\text{Used Resource}}{\text{Total Available Resources}}$$

where $\text{Total Available Resource} = \text{Used Resource} + \text{Unused Resource}$

- Resource:
 - Time: processor time
 - Space: memory, storage
 - Others: power, cooling, money



Utilization

- You work part time to prepare burger. It takes you 1 minute to prepare a burger, and another 30 seconds to put it in the packaging box.
- If you only get 20 order in an hour, what is your utilization?



Utilization Law

$$U = \lambda S \quad or$$

$$U = \lambda / \mu$$



U: Utilization

μ : Mean Service Rate (*rate of processing*)

λ : Mean Arrival Rate (*rate of requests*)

S: Mean Service Time (*i.e.: S is $1/\mu$*)

Utilization depends on:

- a) how many job the worker has to perform in a period of time, *and*
- b) how long it takes to perform the job.



Utilization Law

- Consider a network gateway at which the packets arrive at a rate of 125 packets per second and the gateway takes an average of 2 milliseconds to forward them.
- Calculate the Utilization.



Forced Flow Law

$$\lambda_i = \lambda_0 V_i$$

λ_i : Load for component i

λ_0 : Load for the system

V_i : Visit Ratio - Average number of call (visit) to component i for each request to the system

Throughput of a component depends on average throughput of the system and how many times the component is called on an average for every requests.



Bottleneck Analysis

Combining Utilization Law and Forced Flow Law:

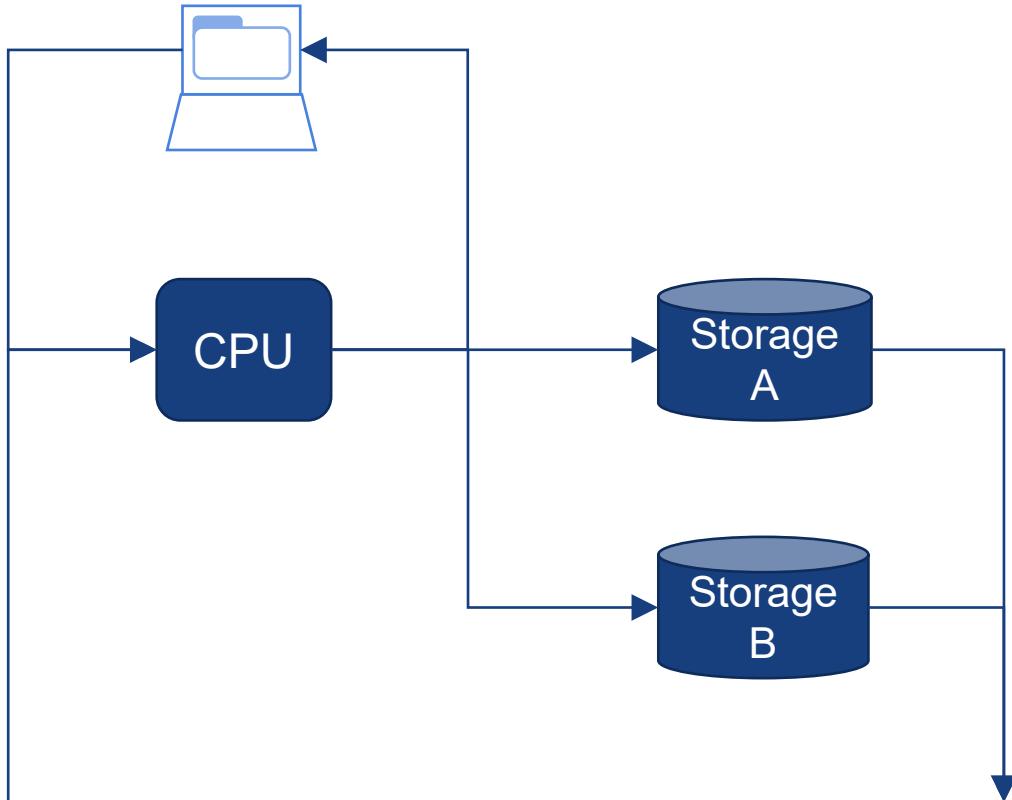
$$\begin{aligned}U_i &= \lambda_i S_i \\&= \lambda_0 V_i S_i \\&= \lambda_0 D_i\end{aligned}$$

Where D_i is the total service demand for component i .

The component with the highest D_i is the bottleneck of the system



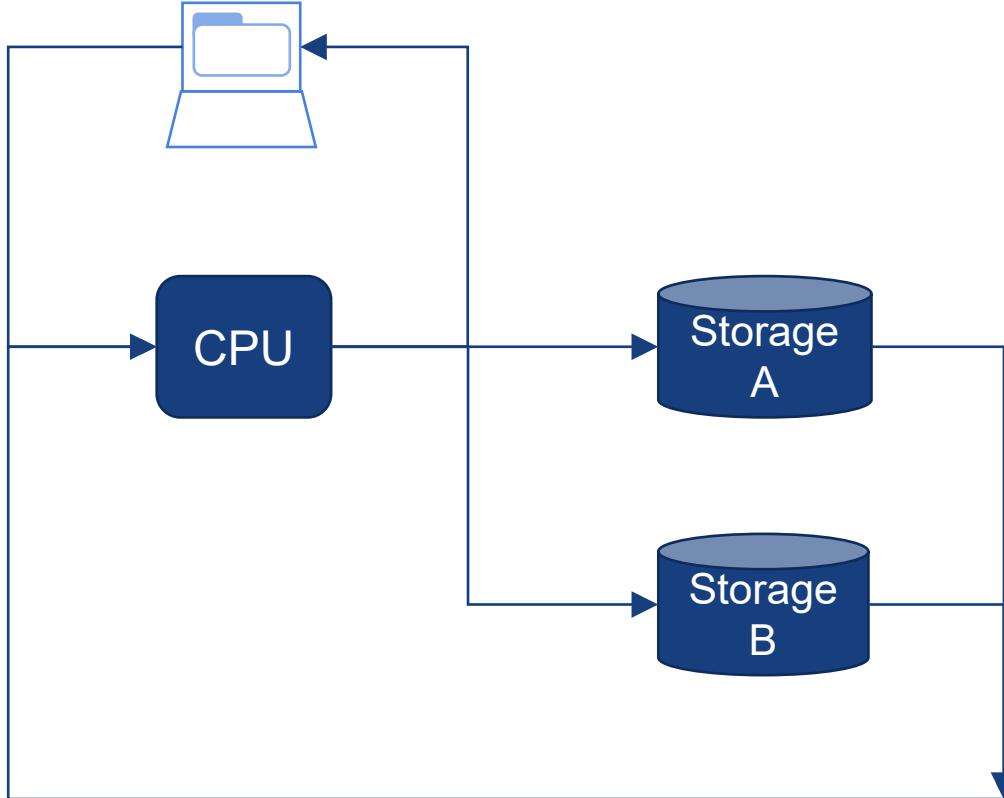
Forced Flow Law and Bottleneck Analysis



- A closed queue network system.
- The terminal sends an instruction to the CPU.
- The CPU would make several calls to both storage A and B.
- The storage system will return the result to CPU, which may make further calls to the storage systems.
- Finally, the CPU will return the instruction results to the terminal.



Forced Flow Law and Bottleneck Analysis



Observations:

- Each instruction requires 5 seconds of CPU time, makes 80 I/O request to storage A and 100 I/O request to storage B.
- From device specifications, storage A takes 50 ms and storage B takes 30 ms to satisfy an I/O request.
- Throughout the experiment, it was observed that storage A throughput was 15.70 requests per second.
- Calculate the system throughput and the device utilizations.



Forced Flow Law and Bottleneck Analysis

What we have:

- $D_{\text{cpu}} = 5 \text{ sec}$
- $V_A = 80, S_A = 0.05 \text{ sec}, \lambda_A = 15.70$
- $V_B = 100, S_B = 0.03 \text{ sec}$

Applying forced flow law:

- $\lambda_0 = \lambda_A / V_A = 0.196 \text{ job/sec}$

Obtaining all the total service demand:

- $D_A = V_A S_A = 4 \text{ sec}$
- $D_B = V_B S_B = 3 \text{ sec}$
- Since D_{CPU} is the highest, CPU is the bottleneck.

Utilization:

- $U_{\text{CPU}} = \lambda_0 D_{\text{CPU}} = 98\%$
- $U_A = \lambda_0 D_A = 78.4\%$
- $U_B = \lambda_0 D_B = 58.8\%$



Little's Law

$$n = \lambda R$$

n: Number of jobs in the system (requests)

λ : Rate of requests

R: Response time

Average number of jobs in the system is equal to average arrival time multiplied by average response/residence time.



Little's Law Example

- The shop gets 15 customers each hour.
- The average size of each customer order is 1.6 burgers.
- It takes 1 minute to make a burger and 0.5 minutes to pack (roughly the response time)
- What is average number of customers in the shop.



Little's Law (extra example)

- Continuing the previous example, if the number of job in the system is observed to be 8.88, 3.19, and 1.40 jobs at the CPU, storage A, and storage B, respectively. What were the response times of these devices?
- What we have:
 - Number of jobs in the system
 - Total rate request (0.196 job/sec)
- Using Forced Flow Law:
 - $\lambda_B = \lambda_0 V_B = 19.6 \text{ job/sec}$
 - $\lambda_{CPU} = \lambda_0 V_{CPU} = 35.47 \text{ job/sec}$ (number of visit = visit to 2 storages + 1)
- Using Little's Law:
 - $R_{CPU} = n_{CPU} / \lambda_{CPU} = 0.250 \text{ sec}$
 - $R_A = n_A / \lambda_A = 0.203 \text{ sec}$
 - $R_B = n_B / \lambda_B = 0.071 \text{ sec}$



General Response Time Law

- General Response Time Law

$$R = \sum_{i=1}^M R_i V_i$$

- R is the total response time of the system
- R_i is the response time of component i in the system
- V_i is the visit rate of component i in the system
- M is the number of component in the system



General Response Time Law (Example)

- Continuing the previous example, since we know the response time and the visit rate of all the components, the total response time can be easily calculated.
 - $R = 68.6$ seconds



QUEUE MATHEMATICAL



Queue Example

- You work part time to prepare burger. It takes you 1 minute to prepare a burger, and another 30 seconds to put it in the packaging box
- On average, there will be one customer coming to the shop every 2 minutes. Each customer order 1 burger
- Will there ever be a queue in the shop?



Queue

- Imagine that 30 customers come in the last 5 minutes of the hour after 55 minutes of idleness
- In many scenarios, the rate of request is not uniform, but we can find out the average. How?



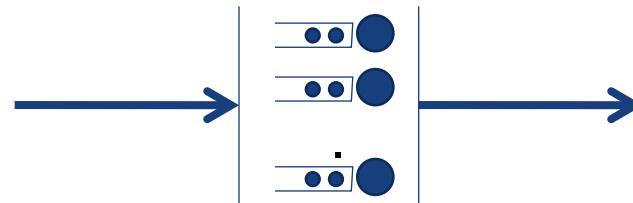
Queuing Theory

- Theory that study queues.
 - We are going to use some simple ones.

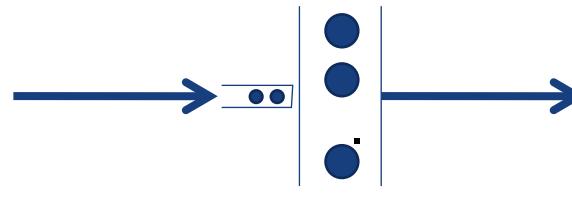
- Single Queue
 - $(M/M/1)$



- Parallel Queue
 - $c(M/M/1)$

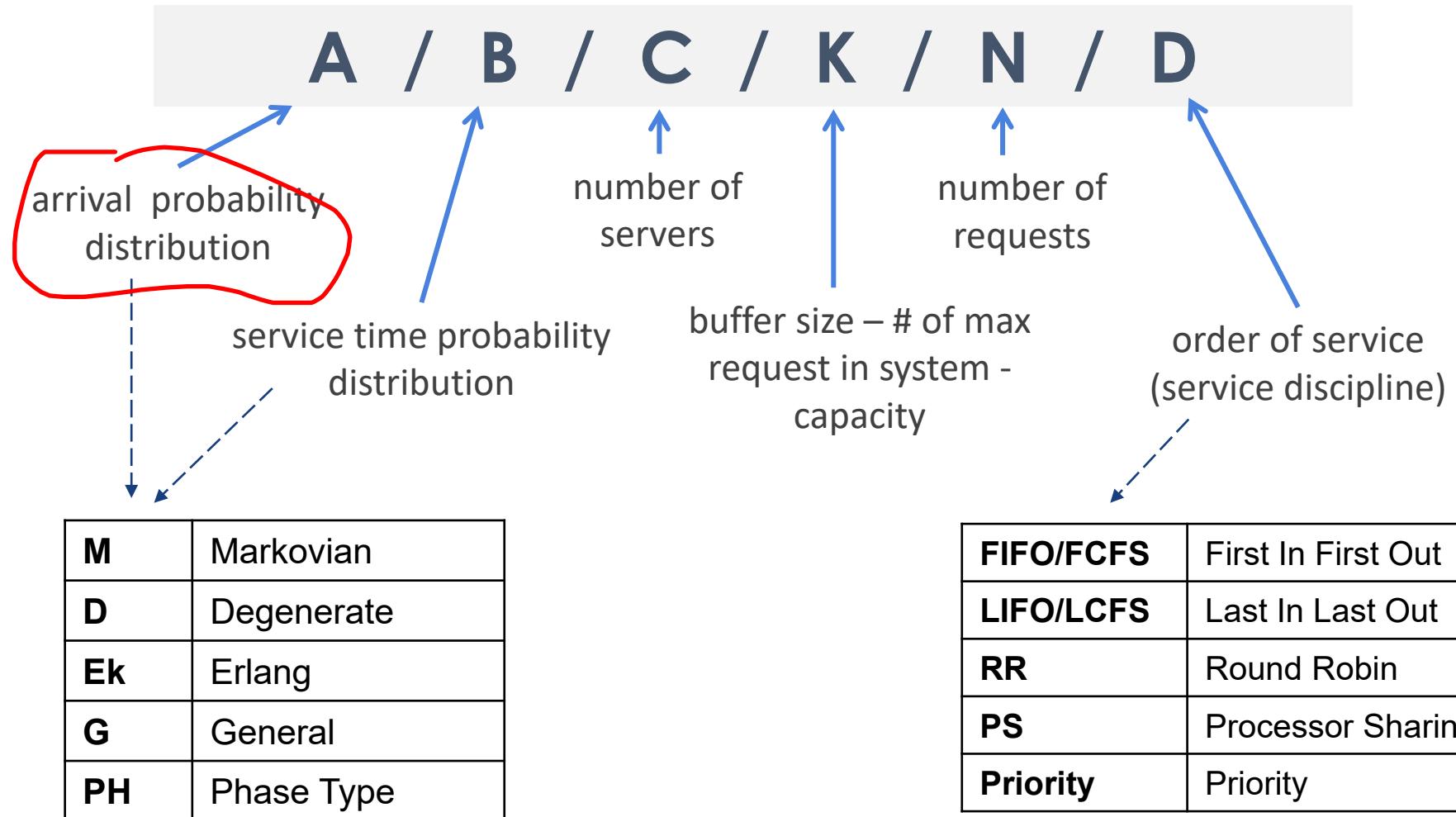


- Single Queue Multiple Server
 - $(M/M/c)$





Kendall Notation (Queuing Theory)





Response Time (Single Queue)

- Queue Type:
Single Server Single queue -- (M/M/1)
- Formula:

$$R = \frac{S}{1 - U} \quad OR \quad \frac{1}{\mu - \lambda}$$

where U is utilization
 S is service time
 μ is service rate (i.e., $1/S$)
 λ is arrival rate
 $U = \lambda/\mu$ is between 0 and 1



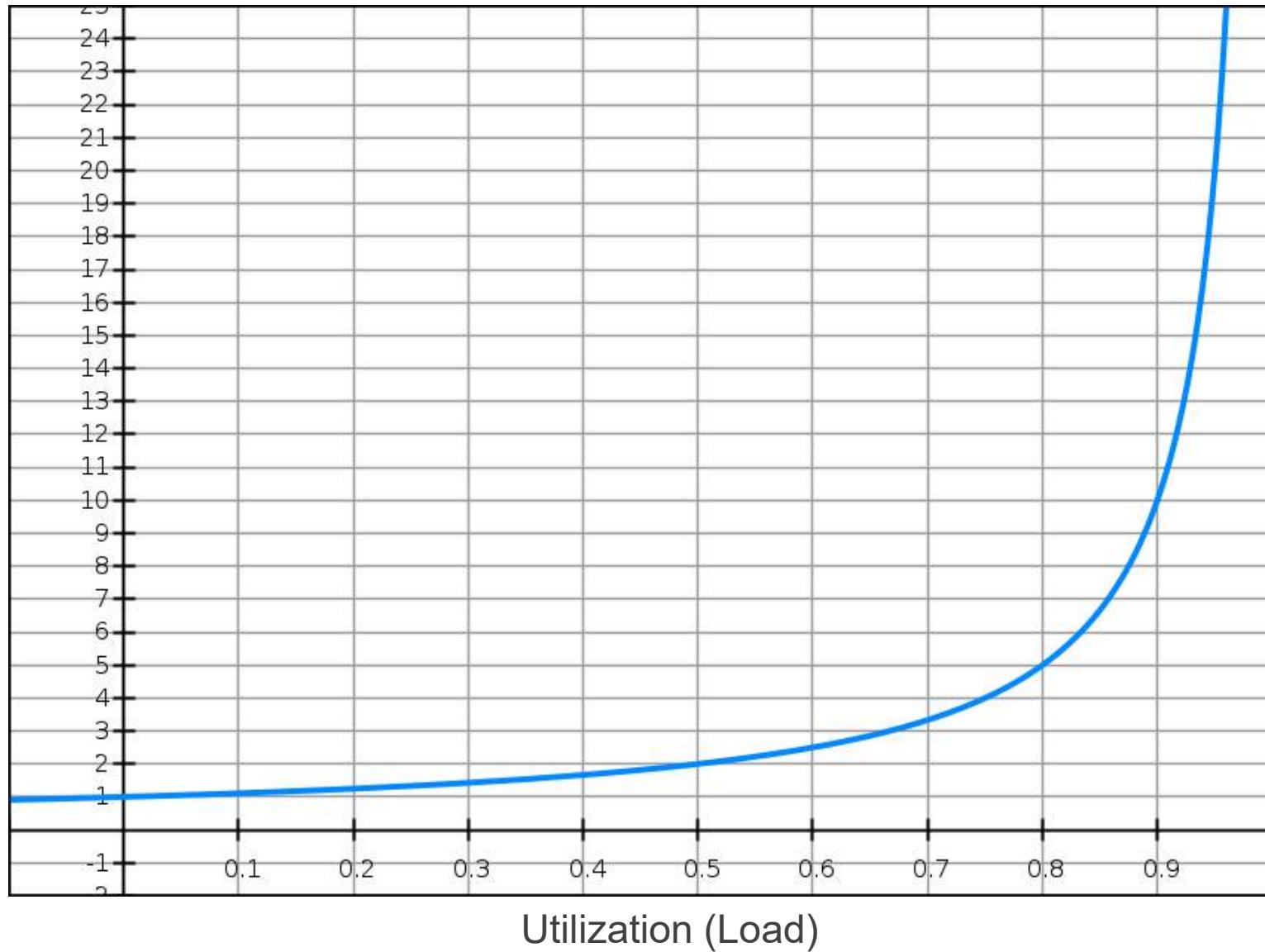
Queue

- If customer comes in randomly, but we know that there are 20 orders of burgers in an hour and the shop (assume one server) can service 30 orders an hour. What is the average amount of time a customer would stay in the shop, assuming customer would come, order and go?



Response Time vs. Utilization

Response Time





Response Time vs Utilization

- Assume service time is 1s
- Average response time when the load is 50% is twice the service time
- Average response time when the load is 70% is more than 3 times the service time
- Limiting our utilization would limit the average response time from growing as well.
 - What is the maximum “healthy” average utilization rate?



Revisit Operational Law

- Response time has an inverse relationship to Utilization.
- Recall that utilization of a component is equals to overall system arrival rate multiplied by the demand of the component.

$$\begin{aligned}U_i &= \lambda_i S_i \\&= \lambda_0 V_i S_i \\&= \lambda_0 D_i\end{aligned}$$

- If the arrival rate of the system remains the same,
 - Reducing S_i (service time) will reduce utilization.
 - Reducing V_i (visit rate) will reduce utilization.
- Without changing the service time or visit rate, we can reduce utilization by reducing the arrival rate to the system.



Response Time (Multiple Queue)

- Queue Type:

Multiple Server, each with separate Queue -- c(M/M/1)

- Formula:

$$R = \frac{S}{1 - \rho}$$

where S is service time

$$\rho = U/c ,$$

U being utilization and $0 < \rho < 1$



Response Time (Single Queue – Multiple Servers)

- Queue Type:

Single Queue Multiple Servers -- (M/M/c)

- Formula:

$$R = \frac{S}{1 - \rho^c}$$

where S is service time

$$\rho = U/c,$$

U being utilization and $0 < \rho < 1$

Note: The formula is an approximation, the exact formula is provided in appendix (for completeness)



Ex: Parallel Queue, Multiple Servers

- The burger shop configures parallel queues
- In a busy day, they expect 2 customers/minute average coming to the shop.
- As before each customer service required 1.5 minutes (1 minute for the burger and 0.5 minutes for packing)
- Target average queueing time is less than 3 minutes

How many pairs of burger flipper and cashiers are needed?



Ex: Single Queue, Multiple Servers

- The burger shop configures a single queue in the previous example (restated below)
 - In a busy day, they expect 2 customers/minute average coming to the shop.
 - As before each customer service required 1.5 minutes (1 minute for the burger and 0.5 minutes for packing)
 - Target average queueing time is less than 3 minutes

How many pairs of burger flipper and cashiers are needed?



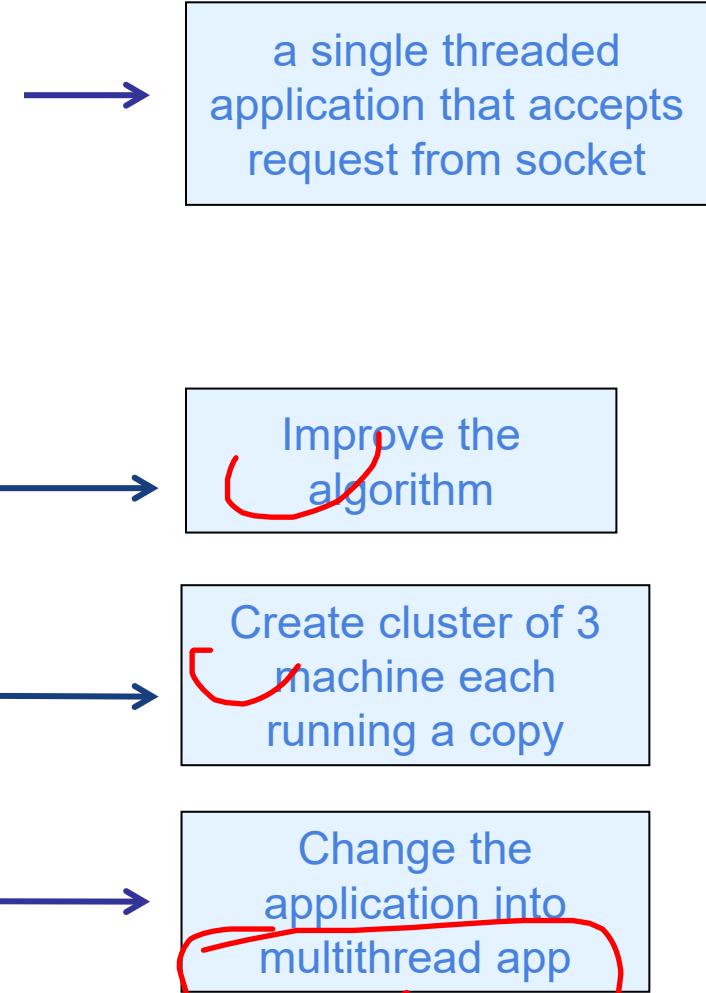
Parallel vs Single Queue

- If you are the burger shop owner, which kind of queue would you prefer?
- Why is one of the queue model is better than the other?



Performance of various Queue Types

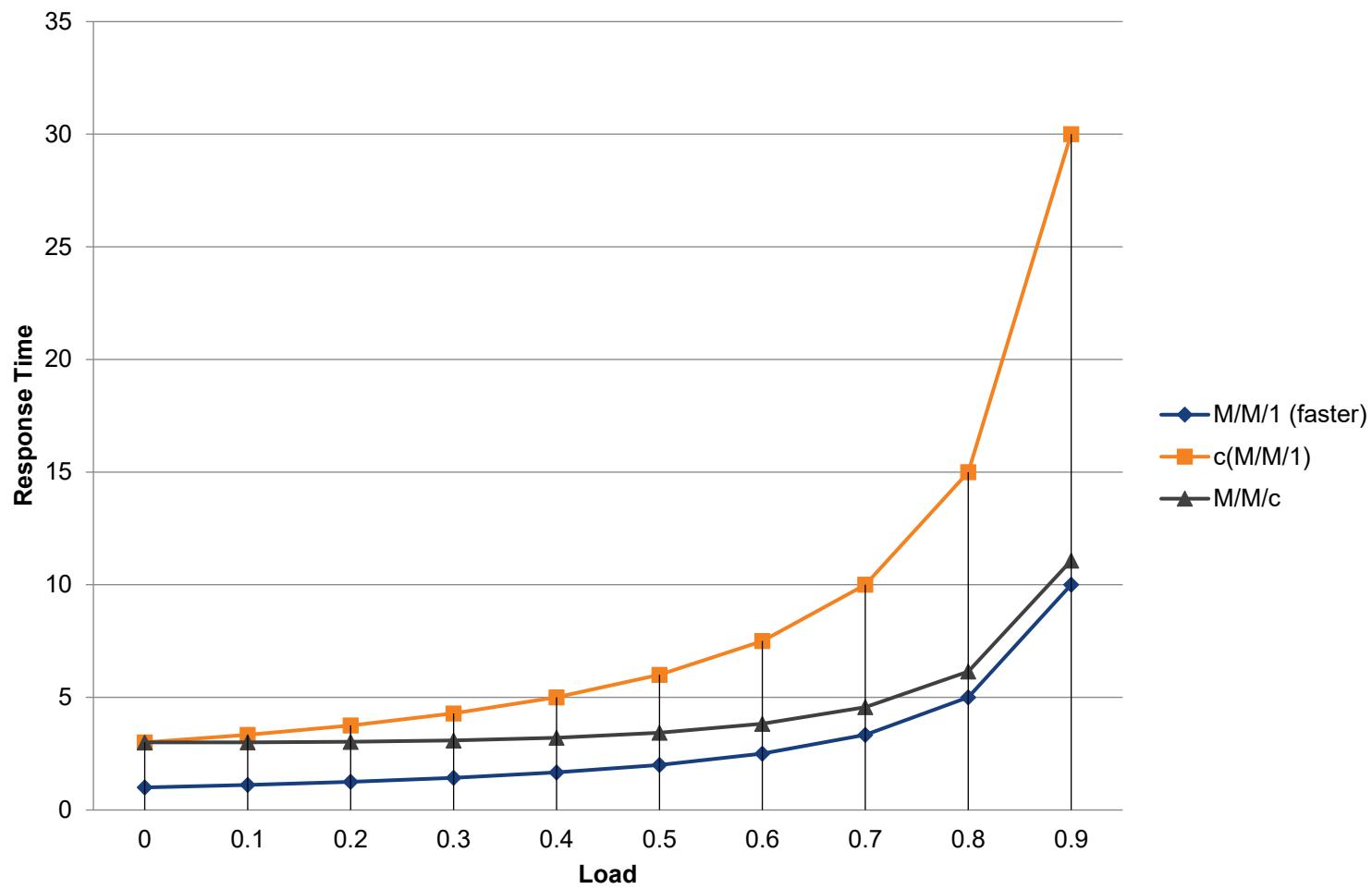
- Assume we have a single server with service time S
- We want to improve the response time by changing it into one of the below options
 - Improve the service time to 3 times faster (faster M/M/1)
 - Create 3 parallel server with its own queue ($c(M/M/1)$)
 - Create 3 server with single queue ($M/M/c$)



Which approach is better?



Comparison Result



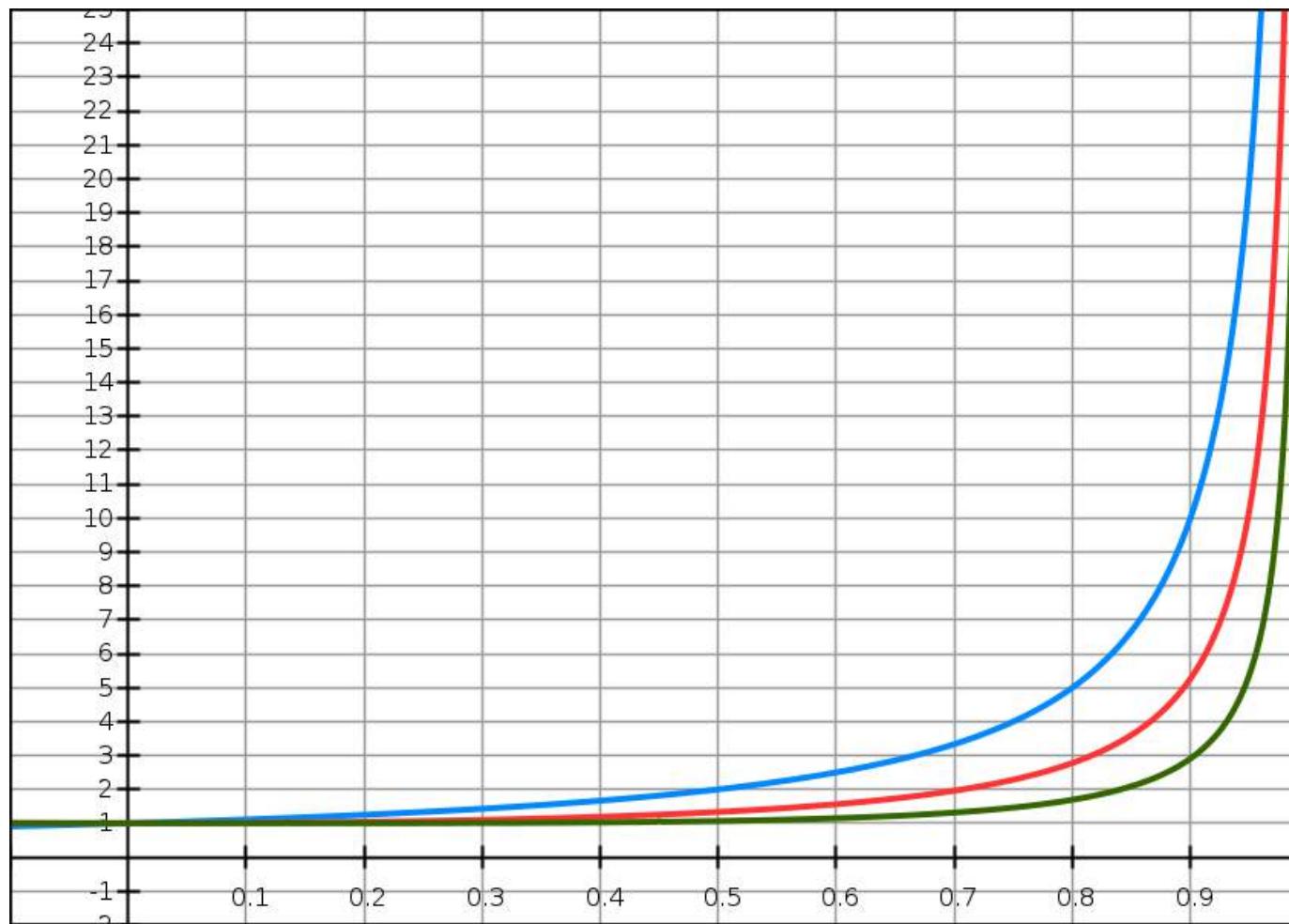


Effect of multiple worker for single queue



NUS
National University
of Singapore

ISYE



1 worker

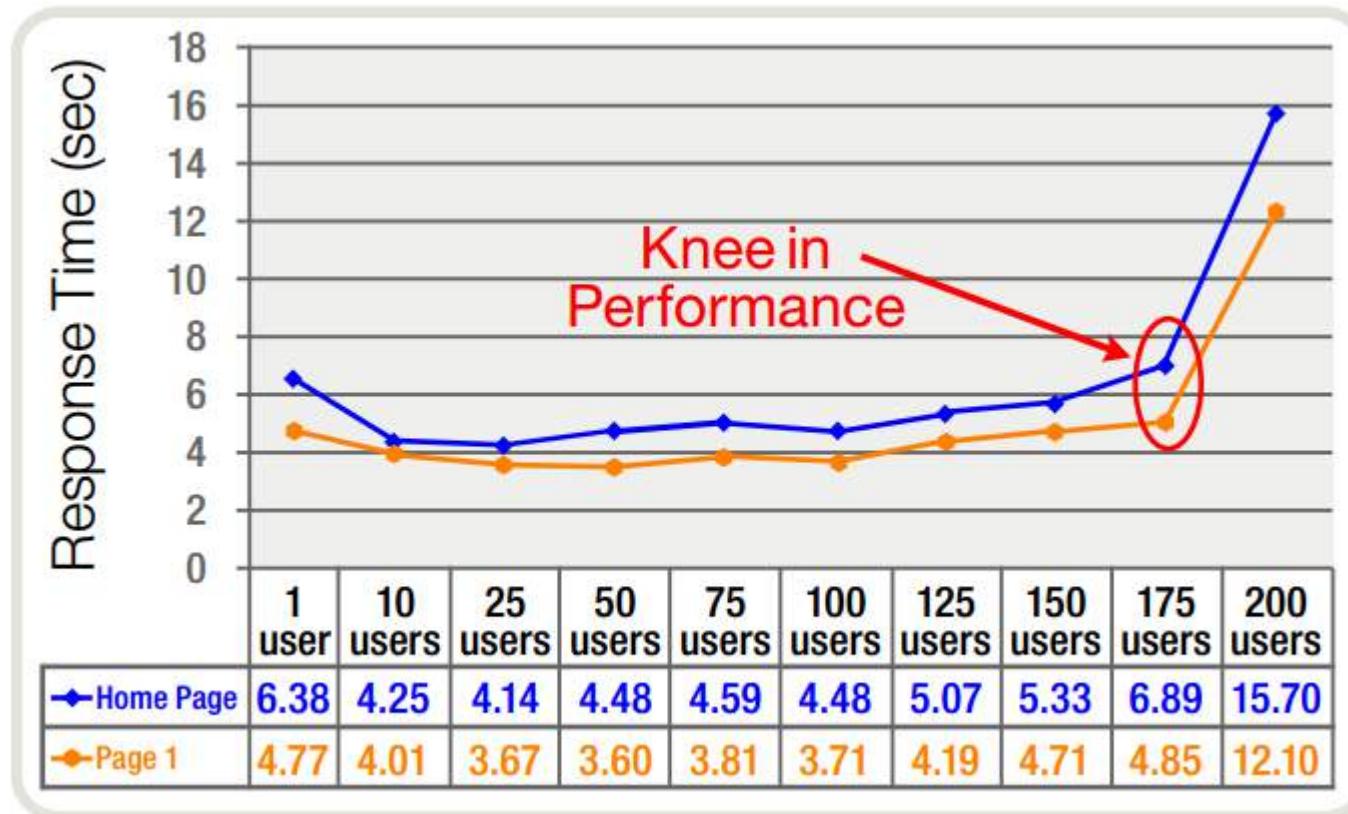
2 workers

4 workers



Response Time vs Load

Look familiar?



http://www.perftestplus.com/resources/requirements_with_compuware.pdf



Price/Performance

- Can be calculated by simply dividing the cost by the throughput.

- Example

http://www.tpc.org/tpce/results/tpce_price_perf_results.asp

- \$/tps

TPC E - Top Ten Price/Performance Results

Version 1 Results As of 11-Sept-2014 10:55 AM [GMT]

Note 1: The TPC believes it is not valid to compare prices or price/performance of results in different currencies.

All Results Clustered Results Non-Clustered Results Currency United States - Dollar (USD) ▾

Rank	Company	System	Performance (tpsE)	Price/tpsE	Watts/tpsE	System Availability	Database
1	FUJITSU	PRIMERGY RX300 S8	2,472.58	135.14 USD	NR	09/10/13	Microsoft SQL Server 2012 Enterprise Edition SP1
2	IBM	IBM System x3650 M4	2,590.93	150.00 USD	NR	11/29/13	Microsoft SQL Server 2012 Enterprise Edition
3	FUJITSU	PRIMERGY RX500 S7	2,651.27	161.95 USD	.68	11/05/12	Microsoft SQL Server 2012 Enterprise Edition
4	hp invent	HP ProLiant DL380p Gen8	1,881.76	173.00 USD	NR	11/21/12	Microsoft SQL Server 2012 Enterprise Edition



Case Study: Commodity Hardware

- Web Search for a Planet: Google Cluster Architecture
 - <http://static.googleusercontent.com/media/research.google.com/en//archive/googlecluster-ieee.pdf>

“Our focus in price/performance favors servers that resemble mid-range desktop PCs

Our ultimate selection criterion is cost per query, expressed as the sum of capital expense (with depreciation) and operating costs (hosting, system administration, and repairs) divided by performance.

Realistically, a server will not last beyond two or three years because of its disparity in performance when compared with newer machines.”



Price/Performance

- What is the measure of price and measure of performance for
 - TPC case study
 - Google case study



Summary of Performance Metrics

- Response Time = Service Time + Queue Time
- Faster response time leads to better throughput
- Throughput is directly proportional with load when the load is less than saturation point (max throughput)
- Ratio between the load/max throughput is more or less directly proportional with the utilization



Architecting Software Performance

- Applying the performance tactics
- Validate the design using performance modeling technique
- Validate the design using prototyping technique



Performance Design Tactics

- Control Resource Demand
 - Manage arrival rate
 - Prioritize input
- Manage Resources
 - Increase resources
 - Increase resource efficiencies
 - Multiple copies of compute
 - Multiple copies of data
 - Reduce overhead
 - Schedule resource



Performance Design Tactics

- Consider and apply design heuristic in your architecture:
 - Load balancing
 - Caching
 - Pre-computation
 - Reduce algorithmic complexity
 - Avoid/reduce bottleneck
 - May tradeoff with cost



Load Balancing

- Definition

Load balancing is the process of distributing network traffic (or client requests) across multiple servers.

This ensures no single server bears too much demand. By spreading the work evenly, load balancing improves application responsiveness. It also increases availability of applications and websites for users.

- Can improve:

- Throughput ✓
- Speed ✓

- How?

- Remember queuing theory. What happens when you add more servers?



Improving Throughput

Single Worker



Two Workers – same job



Two Workers - specialization



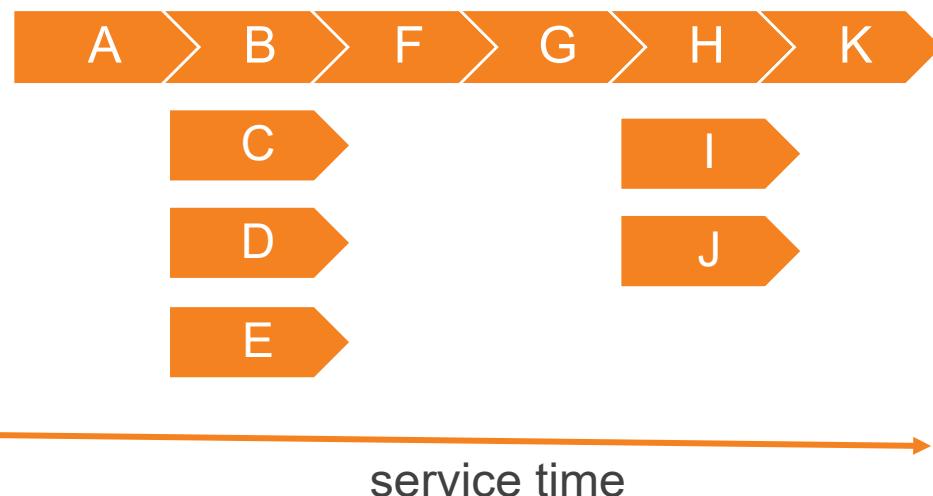


Improving Speed

Sequential Execution



Parallel Execution





Distributing the load

- Round robin
- Random
- Server load
- Least response time
- Traffic served
- Number of active connections
- Geographical locations
- Capabilities

Why bother?



Load Balancer and State

- If the server holds some state for the client, then the client may need to always connect to the same server.
 - E.g. browser need to connect to application server that holds its session state
 - What happens otherwise?
- What are the concerns?
 - Availability
 - Uneven distribution



Cache

- Definition:

A cache is a data storing technique that provides the ability to access data or files at a higher speed.

- Types of Cache:

- Memory (typically data from database server gets cached)
- Browser
- Webserver
- Remote Servers

D S N

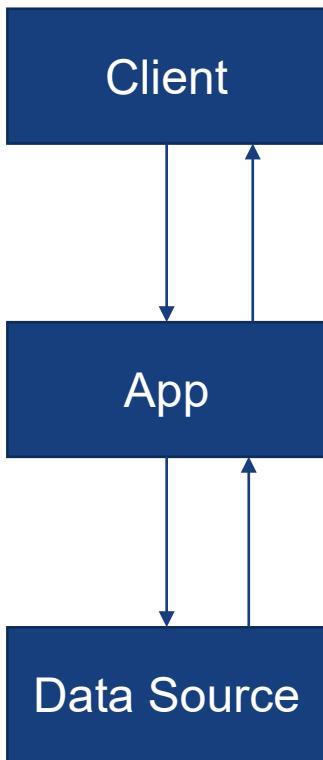
- Two events associated with caching.

- Cache hit: *Data requested is in the cache*
- Cache miss: *Data requested is not in the cache*

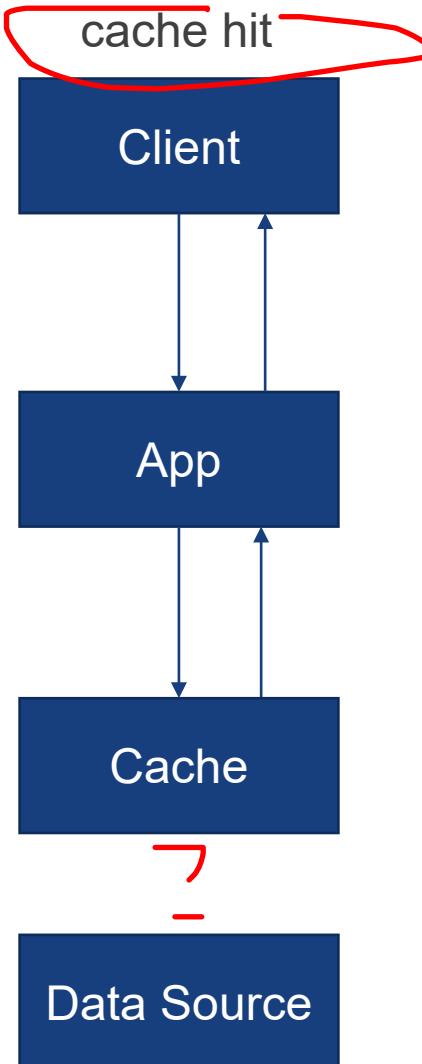


Comparison

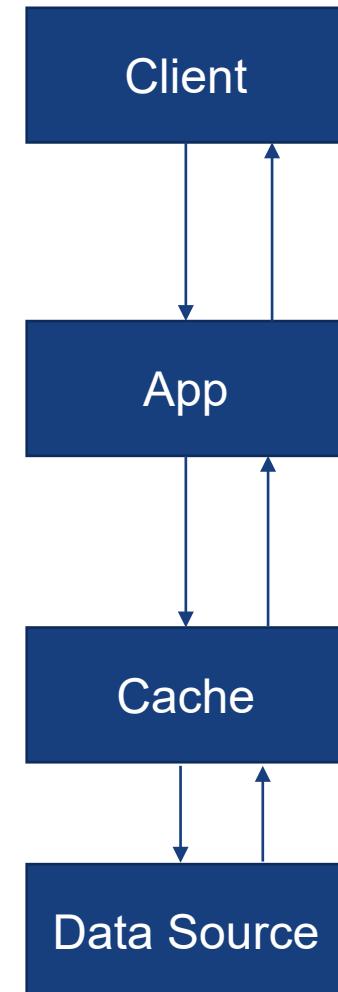
no cache



cache hit



cache miss



How does it even make sense?



Calculating Cache Efficiency

Assumption

Location	Access Time
Cache	5 ms
Data Source	50 ms

No cache

Scenario	Access Time
No cache	50 ms

With Cache

Scenario	Access Time	Ratio	Ratio * Access Time
Cache Hit	5 ms	20%	1 ms
Cache Miss	55 ms	80%	44 ms
Weighted Average			45 ms

In this case, is it worth having the cache?

Can you change cache access time so that it's not worth having the cache?



Why the different access time?

- Fast storage tends to be:
 - More expensive
 - Thus smaller in size
- Location
 - Local machine
 - Local network
 - Remote network (near vs. far)
- Different operation
 - Simple lookup (e.g. with primary key)
 - Complex query



Cache cost/performance

- Cost measurement
 - Storage size
 - buying storage need money
 - Bandwidth
 - CDN may charge by bandwidth consumed instead of storage used
- Performance measurement
 - Improvement of response time
 - Improvement of throughput
- Example:
 - For every 1ms improvement of response time, we need 1GB of storage
 - For every 1 tps throughput improvement, we need to pay for 1 GB of CDN bandwidth



Cache Strategy

Assume that we get this data from testing

Data Type	Cache Size	Speedup	Speedup/Size
A	10 GB	10 ms	
B	20 GB	30 ms	
C	30 GB	10 ms	

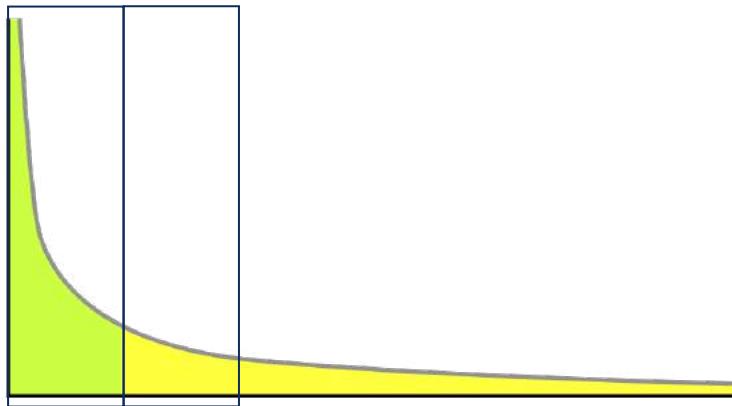
What is the priority of the data type to be cached?

If we can only allocate 15 GB of RAM to be used for cache, how should this 15 GB be used?

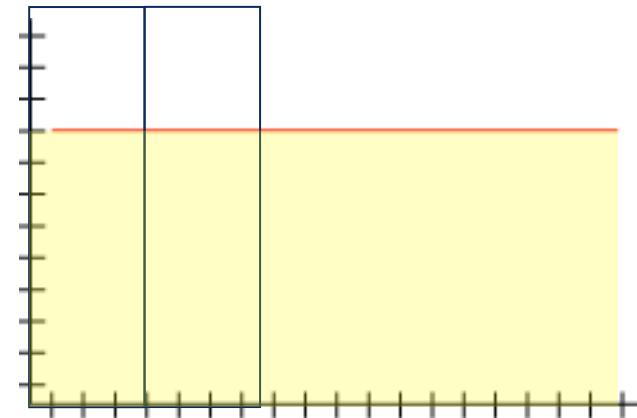
If we have the hit ratio, we can probably extrapolate how much speedup we can get with less than cache size.



Effect of Cache Size to Hit Ratio



Power Law Distribution



Uniform Distribution

- Imagine that the curve above represent the relationship between frequency of access (y-axis) vs. data ID (ordered from the most popular to least popular)
- The hit ratio is represented by the area under the curve
- The box represent the size of cache used to keep the data (assuming the data size for each item is roughly uniform)
- We can easily see that if we halve the cache size, the hit ratio will be at least half of the original hit ratio



Precomputation

- The idea is similar to cache
 - Pre-compute some value
 - Store the result
 - Use the result when needed
- Trade-off between:
Single precomputation cost + multiple access cost

vs.

Multiple computation cost



Twitter Example

- Two main operations:
 - Posting of tweet (4.6k rps to 12k rps)
 - Viewing tweets posted by people that the user follows. (300k rps)

Approach 1

- When a user tweets, it will be just an insertion to the database
- When a user enters the timeline, the following query will be done

```
SELECT tweets.*, users.* FROM tweets
    JOIN users ON tweets.sender_id = users.id
    JOIN follows ON follows.followee_id = users.id
    WHERE follows.follower_id = current_user
```

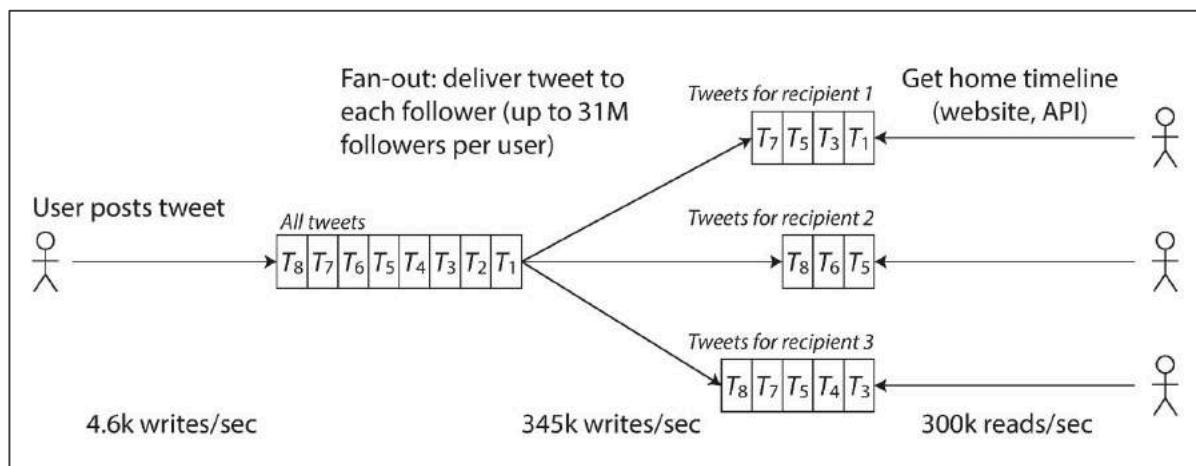
- Twitter originally used this approach, but the query becomes the bottleneck to the system.



Twitter Example

Approach 2

- Maintain a cache for each user's timeline. When a user tweets, look up all the followers, and insert the new tweet into each follower's cache.
- This way, read operations becomes very quick.

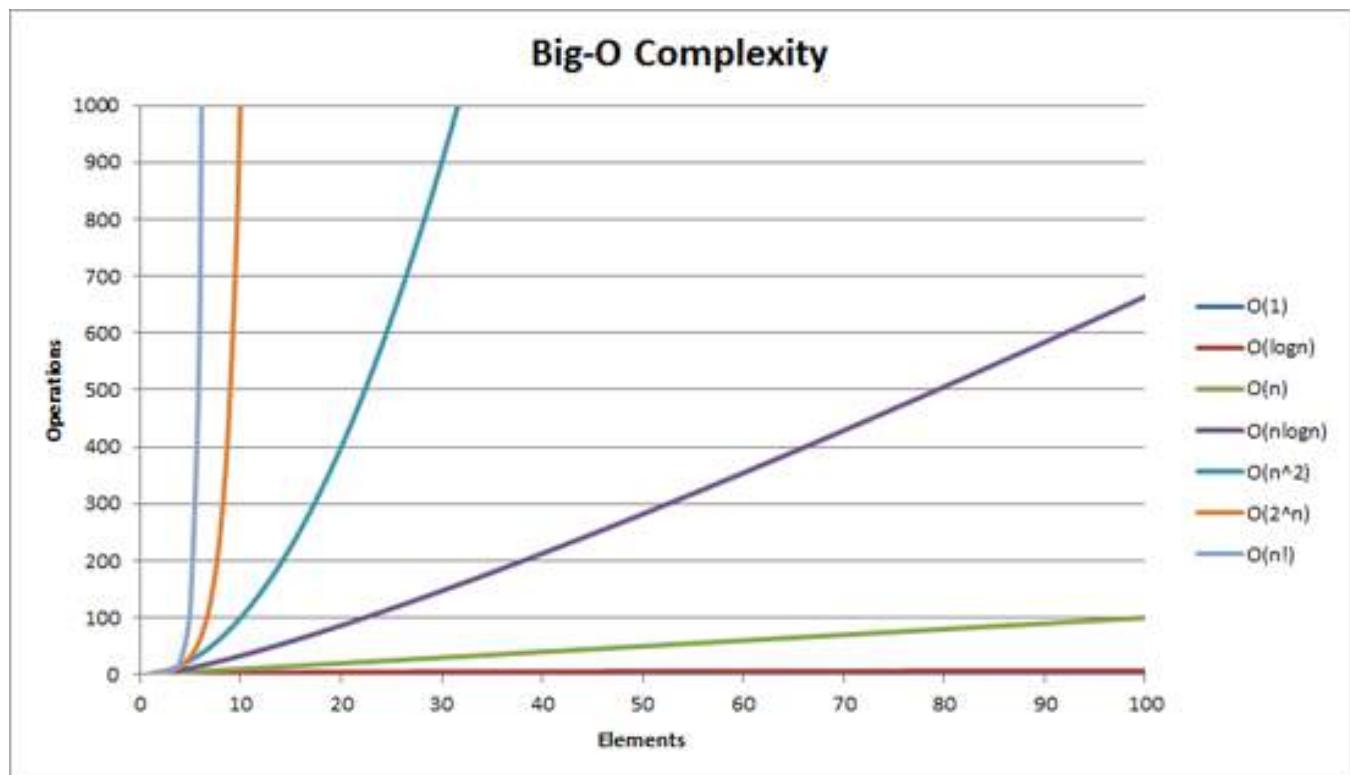


- Currently Twitter combines approach 1 and 2. Approach 1 is used for people who has many followers (e.g. celebrities).



Algorithmic Complexity

- Big O notation is used to tell you how fast the function (time/space) grows or decline given different sizes of input
- You can apply this to other measurable factors



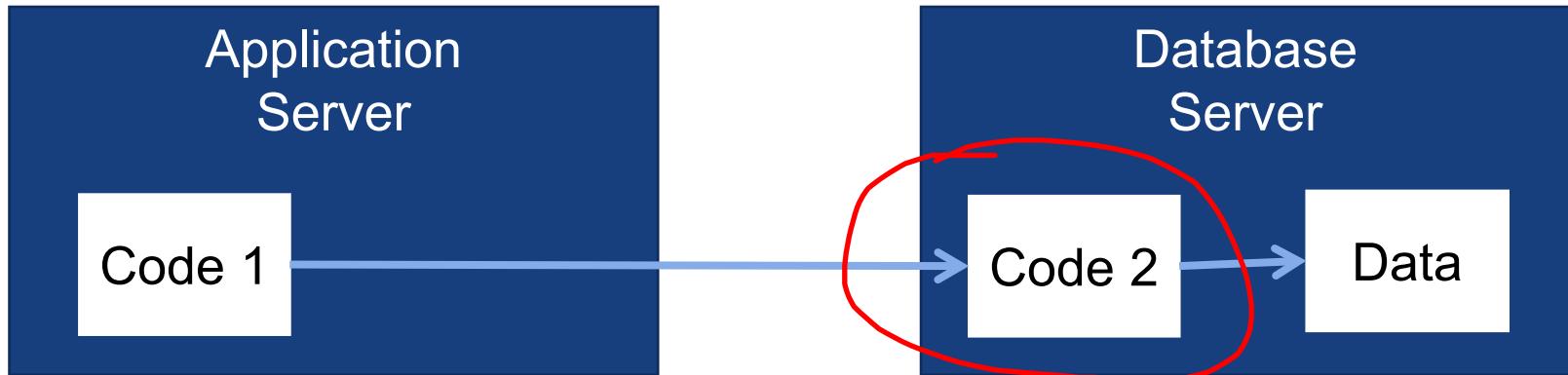
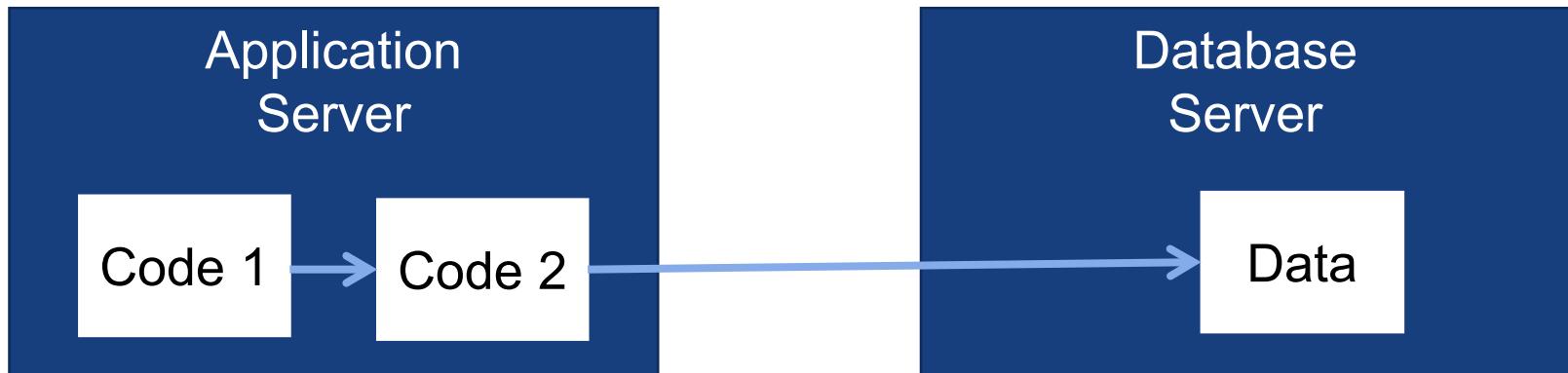


Algorithmic Complexity

	$O(n^2)$	$O(n)$	$O(1)$
1 item	1 second	1 second	1 second
10 items	100 seconds	10 second	1 second
100 items	10000 seconds	100 seconds	1 second



Example: Code in Apps vs. DB





Example: Code in Apps vs. DB

- Case:
 - You want to calculate the score between jobs vs. candidate in a job portal site and return the top 1000 match
 - I know it's unrealistic but ...
 - Consider:
 - The growth in the number of scoring operation (affect the time complexity)
 - The growth in number of network transfer (network complexity)



Optimizing Data Storage

- In many applications, the write or read operations from the data storage is the bottleneck of the system.
- There are many resources on improving performance of relational database (out of scope of this course), but utilizing the right index goes a long way.
- Choosing the right data storage technology also affects the performance of reading/writing into the data store.
- For example, graph database is very optimized for storing information that has many relationship such as social network.



Flow Control

- Case 1: Spam calls
 - What would you do if you get a lot of spam calls everyday?
- Case 2: Overly eager student
 - Suppose there's a student who asks too many questions and reduce the chance for other students to ask questions.
What can be done to help the situation?



Flow Control

- Case 3: Busy food stall
 - You are queueing for your lunch. You are 2nd in line. The person in front of you is ordering 25 packets for his office lunch meeting.
 - How can the queue in the stall be improved?
- Case 4 Burger Shop:
 - A burger shop need cashier and cook.
 - Rule:
 - One person can only have one role at a time
 - When the person play the role, they must complete the current order first before consider changing role
 - A person should grab whatever job available
 - There are 5 person in the shop and suddenly 20 people come in to the shop. What will happen?



Software Simulation

- Simulation software can also be used to perform simulation on how the design is going to work
- Open Source Queue Modeling Tool
 - <http://jmt.sourceforge.net/>



Prototyping and Benchmarking

- Prototyping and benchmarking can be done to verify that the architecture can meet the expected performance before the whole system is built
- If you're going to fail, fail early
 - ~~Unknown technology~~
 - Experimental design
- Trade-off between
 - Level of confidence
 - Effort and cost



Steps

- Define the goals
- Identify the extent of prototyping to meet the goals
- Build the prototype
 - Possibly with some instrumentation
- Perform benchmarking
- Repeat until the performance risk is contained



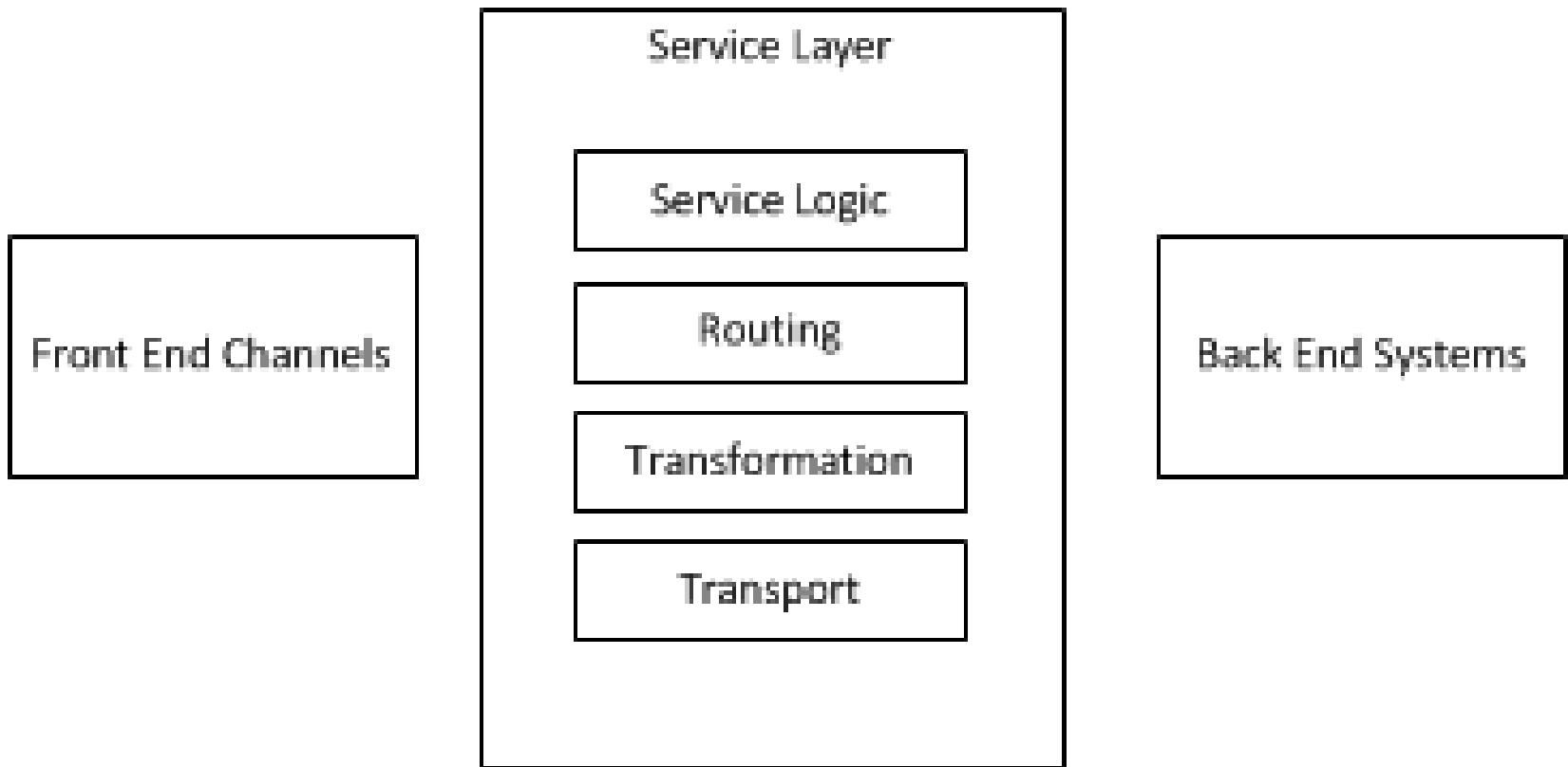
Documenting for Performance

How should architecture be documented from performance perspective?



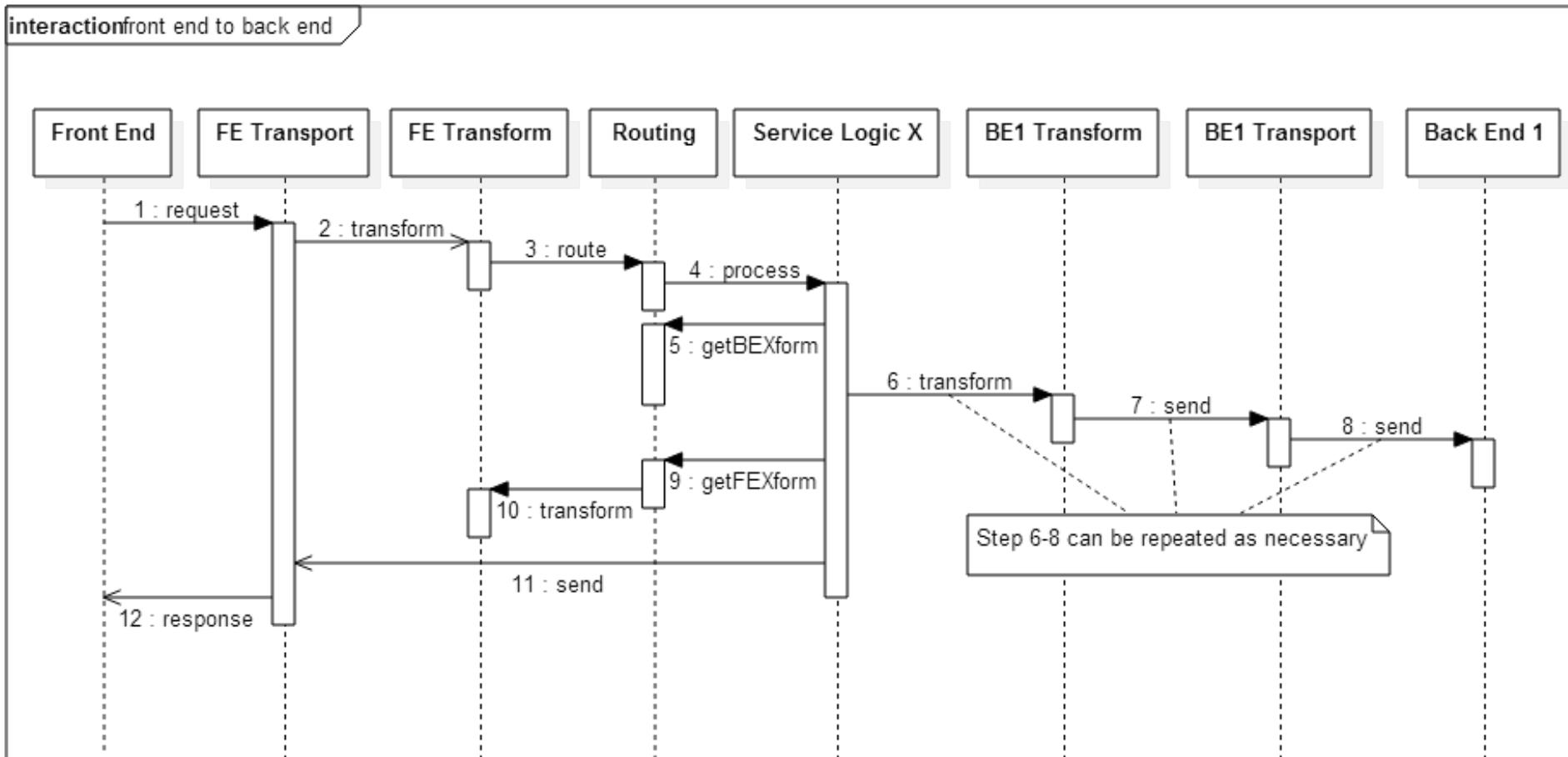
Example: Block Diagram

How do you understand this diagram?



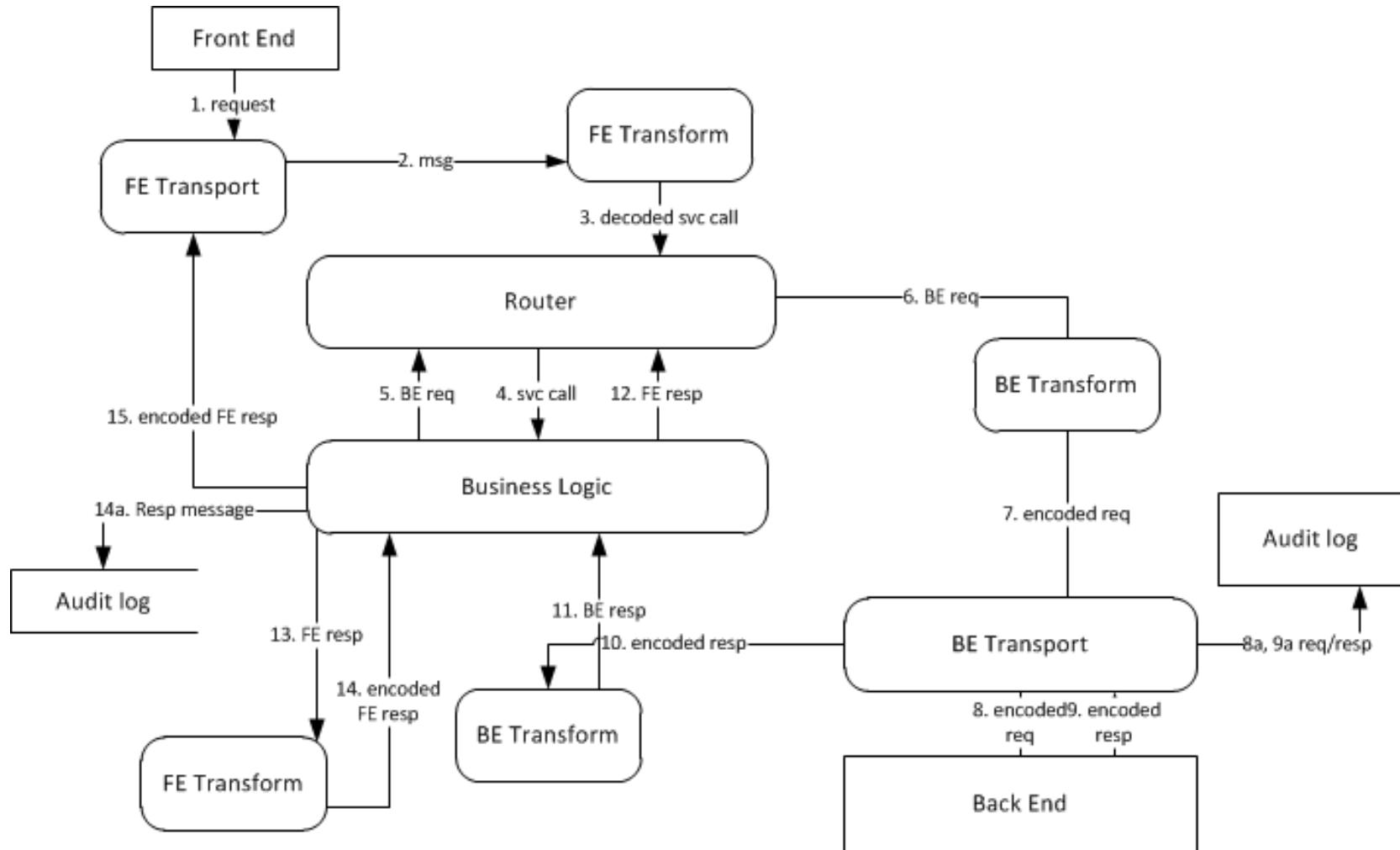


Example: Sequence





Example: Data Flow Diagram





Documentation Tips

- Focus on the major components and items
 - Major components
 - Large size messages
 - Calls that contains many records/items
 - High frequency interaction
- Find the sweet spot. Not too high level, not too low level.
- Perform overall analysis that works on related data. You may find additional insights
 - The architecture may need to handle high load of online transaction as well as a timely complex reporting requirement



Low Level vs High Level

- You need a viewpoint as the basis for thinking about architecture
 - Draw and iterate
- It can be as coarse grained or fine grained as necessary
- Some viewpoint can be more complicated than others
 - Deployment diagram for facial detection system can be very simple, but the logical component diagram can be very complex



Reference

- Len Bass, et al, Software Architecture in Practice, 2nd Edition, 2003, Pearson Education Inc
- Neil J. Gunther, The Practical Performance Analyst, 1998, McGraw-Hill
- Neil J. Gunther, Guerrilla Capacity Planning, 2007, Springer
- Daniel A. Menasce, et al, Capacity Planning for Web Services, 2002, Prentice Hall
- <https://www.ibm.com/developerworks/patterns/>
- <http://www.alexanderpodelko.com/PerfManagement.html>
- Lectures from Prof Raj Jain -
<https://www.cse.wustl.edu/~jain/queue/index.html>



APPENDIX



Average Load and Concurrent Client

- Average load = concurrent client / (think time + response time + latency)
- The intuition:
 - If think time = 0 and response time = 1s, how many requests would one client send in 100 seconds
 - If there are 50 clients with similar behavior, how many requests end within 100 seconds
 - What would be the average load (request/sec) for this 100 seconds period?



M/M/c Queue Formulae

- The formula in the notes is a common approximation that is simpler to calculate.

- Exact formula:

$$R = S \left[1 + \frac{C(m, \rho)}{m(1-\rho)} \right]$$

- $C(m, \rho)$ is probability that all servers are busy and the request must be queued. It is known as Erlang's C function

$$C(m, \rho) = \frac{\frac{(m\rho)^m}{m!}}{(1-\rho) \sum_{k=0}^{m-1} \frac{(m\rho)^k}{k!} + \frac{(m\rho)^m}{m!}}$$



ARCHITECTING SOFTWARE SOLUTIONS

PERFORM CAPACITY PLANNING FOR ARCHITECTURE

Instructor: Darryl Ng

Email: darryl.ng@nus.edu.sg

Total slides: 54



Objectives

- Perform capacity planning
- Perform trend analysis based on historical data
- Plan for capacity procurement



Topics

- Approach
- Predicting server requirement
- Predicting hard disk usage
- Trend Analysis
- Non-linearity and Amdahl Law
- Using benchmark for sizing
- Metrics
- Procurement Lead Time



Why Capacity Planning

Goal:

Ensure sufficient computing resources are available for the expected workload to run with acceptable service level.

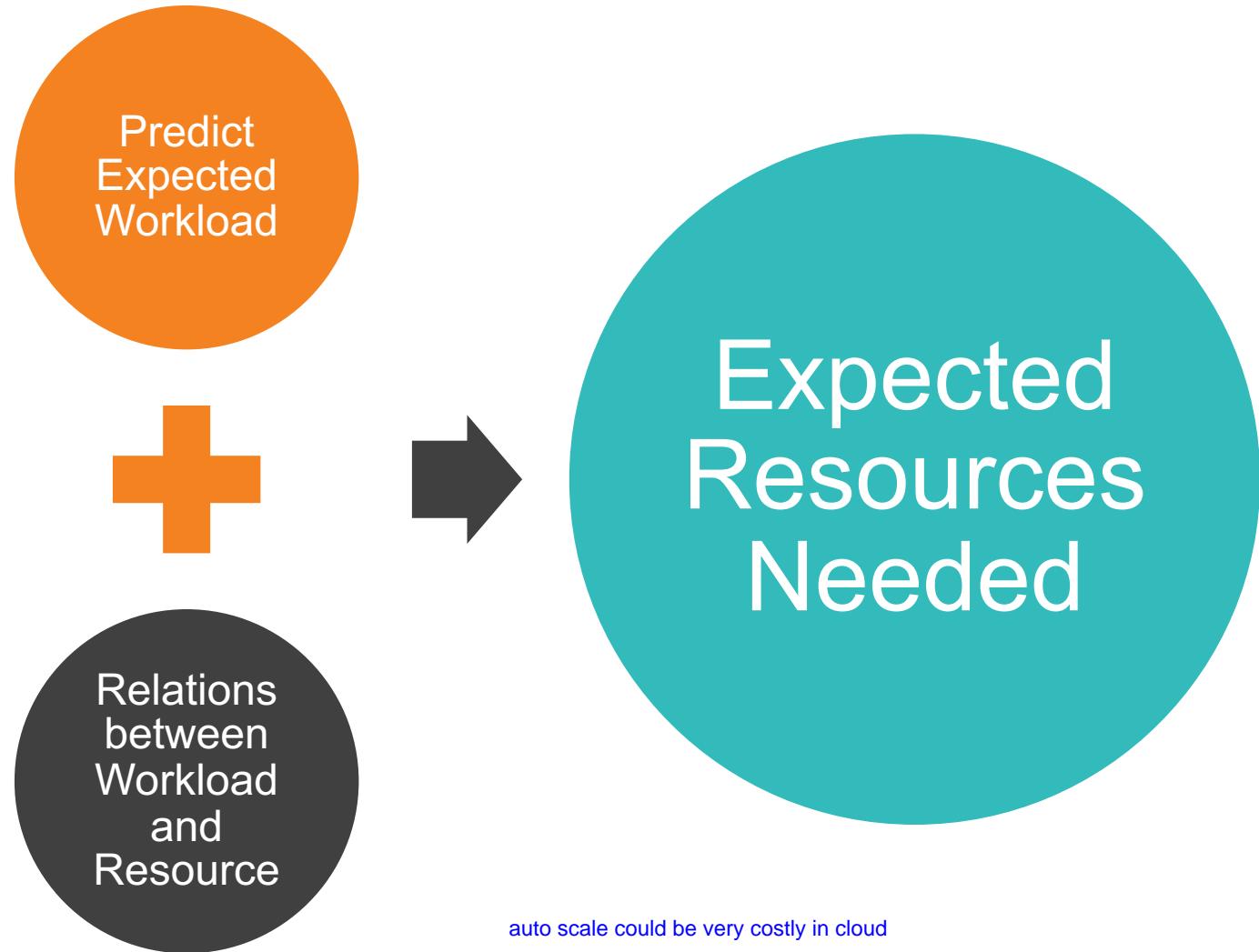
Why is it necessary?

- New System
- Existing System:
 - Growth in the number of users
 - Cumulative growth due to usage
 - New features



Capacity Planning Tasks

- Predict the required computing resources based on the expected workload
- Purchase of new servers and storage
 - How much **extra capacity?**
 - When is the extra capacity needed?
- Allow the system to scale automatically based on the workload (elasticity)





Source of Data

- Published benchmark from vendors
- Self benchmarking
- Historical monitoring of metrics



Collecting Metrics

- **System Metrics**
 - Disk Utilization
 - I/O wait
 - RAM usage
 - CPU usage
 - Used storage
- **Database**
 - Queries/sec
 - Connections open
 - Cache hit rate
 - Master/Slave Lag Time
- **Web/Application Server**
 - Request/sec
 - Active sessions
 - Busy threads
- **Application Metrics**
 - Example is on the next page

magic number 70%



Case: Flickr Application Metrics

- Application Level Metrics
 - Photo uploaded (daily, cumulative)
 - Photos uploaded per hour
 - Average photo size (daily, cumulative)
 - Processing time to segregate photos based on their different sizes (hourly)
 - User registration (daily, cumulative)
 - Pro account signups (daily, cumulative)
 - Number of photos tagged (daily, cumulative)
 - API traffic (API keys in use, request/sec, request/key)
 - Number of unique tags (daily, cumulative)
 - Number of geotagged photos (daily, cumulative)

Source: *The Art of Capacity Planning*



HARD DISK USAGE

PLANNING HARD DISK CAPACITY



Hard disk usage

- Analysis is cumulative driven
- Measurement approach
 - Back of the envelope measurement
 - Monitoring
 - Logging



Hard disk usage

- EXAMPLE:

Predict hard disk usage for a database system that is mainly responsible for keeping customer transaction information



Data feeds

<p>Customer</p> <pre>graph LR; A((Predict Expected Workload)) --- B((Relations between Workload and Resource)); B --- C((Expected Resources Needed))</pre>	<p>Transaction</p> <pre>graph LR; A((Predict Expected Workload)) --- B((Relations between Workload and Resource)); B --- C((Expected Resources Needed))</pre>
<p><other type of data></p> <pre>graph LR; A((Predict Expected Workload)) --- B((Relations between Workload and Resource)); B --- C((Expected Resources Needed))</pre>	<p><other type of data></p> <pre>graph LR; A((Predict Expected Workload)) --- B((Relations between Workload and Resource)); B --- C((Expected Resources Needed))</pre>
<p>Fixed overhead Operating System, Database Application, Log Files</p>	



Example

- Forecast:
 - Number of customer transactions stored will grow from 1 million to 20 millions
 - Average size of transaction records is 1 MB
- Task:
 - Plan for storage capacity

Predict
Expected
Workload

Relations
between
Workload
and
Resource

Expected
Resources
Needed



Example

- What if the parameters are:
 - Currently number of transaction stored is 1 million per month
 - We need to keep the data for 2 years before archival
 - Each month the number of transaction increase by 10%
- Estimate the storage needed to last the next 2 years

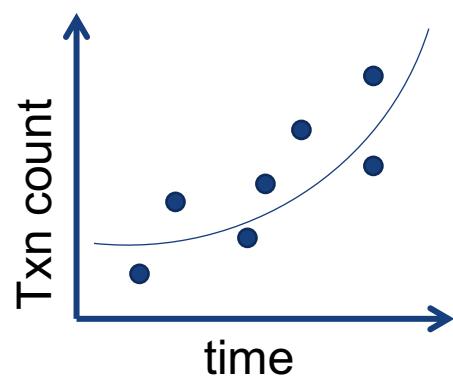
Predict
Expected
Workload

Expected
Resources
Needed



Example

- What if you only know:

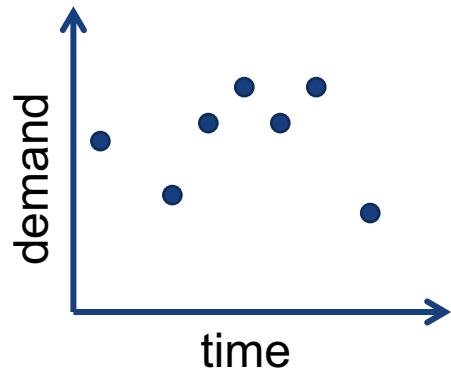


of transaction/month

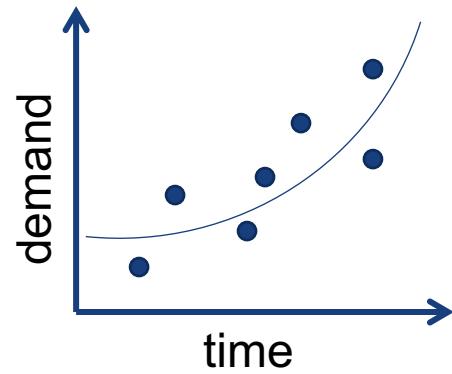
- How would you estimate the storage needed to last the next 2 years



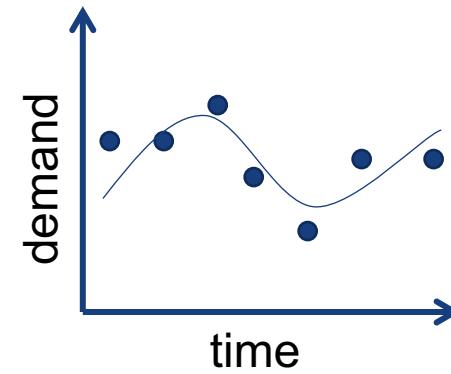
Historical Data Patterns



random



trending



seasonal



Statistical Approach

- Linear Regression

$$Y = a + bX + \text{error term}$$

- Or, simply

$$Y = a + bX$$

where:

$$a = \frac{(\Sigma y)(\Sigma x^2) - (\Sigma x)(\Sigma xy)}{n(\Sigma x^2) - (\Sigma x)^2}$$

$$b = \frac{n(\Sigma xy) - (\Sigma x)(\Sigma y)}{n(\Sigma x^2) - (\Sigma x)^2}$$



Statistical Approach

- Time series analysis where variations due to season or other factors may play a role in the future forecast.
- Moving Average

$$M_t = \frac{X_t + X_{t-1} + \dots + X_{t-N+1}}{N}$$

where N is the number of periods averaged

- Exponential Smoothing

$$s_0 = x_0$$

$$s_t = \alpha x_{t-1} + (1 - \alpha)s_{t-1}, \quad t > 0$$

where α is the *smoothing factor*, and $0 < \alpha < 1$.



Using historical data to predict

- Use statistics and predictive analytics
- In this class, we will do a simple regression with Excel



Example

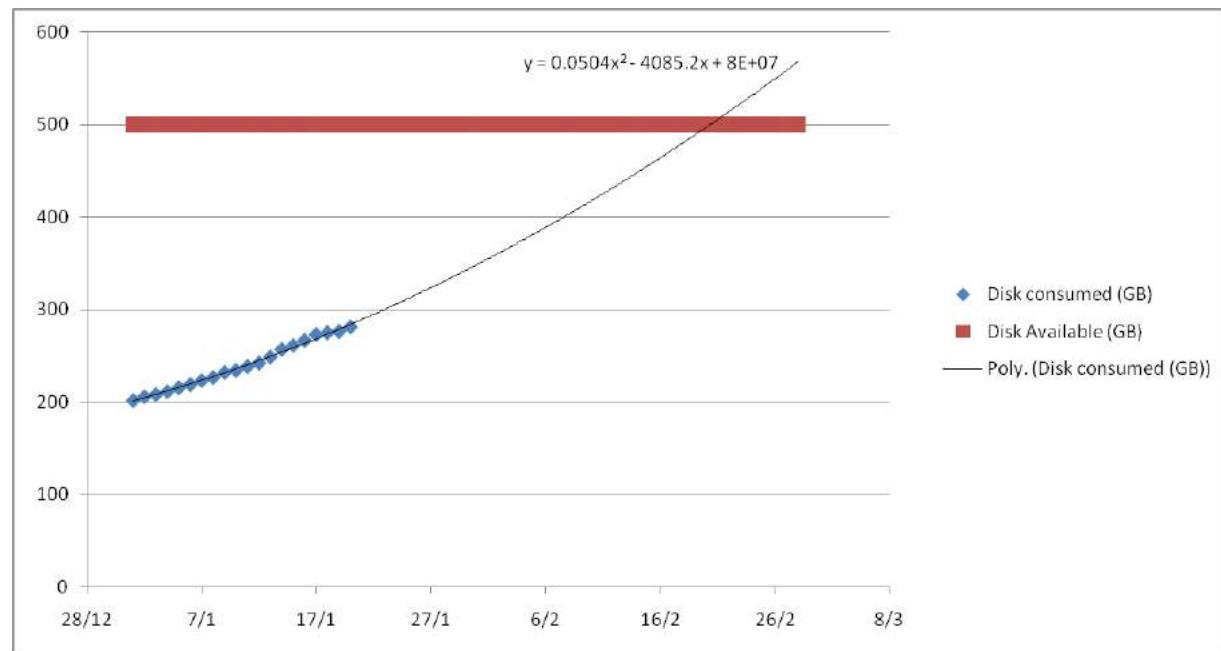
- Based on the data predict when we will run out of space

No	Date	Disk consumed (GB)	Disk Available (GB)
1	01/01/2011	202	500
2	02/01/2011	206	500
3	03/01/2011	208.3	500
4	04/01/2011	211.4	500
5	05/01/2011	215.6	500
6	06/01/2011	219	500
7	07/01/2011	223.4	500
8	08/01/2011	226.6	500
9	09/01/2011	231.9	500
10	10/01/2011	234.2	500
11	11/01/2011	238.5	500
12	12/01/2011	242.2	500
13	13/01/2011	248.5	500
14	14/01/2011	256.7	500
15	15/01/2011	261	500
16	16/01/2011	266.3	500
17	17/01/2011	272.6	500
18	18/01/2011	274.7	500
19	19/01/2011	276	500
20	20/01/2011	280.7	500



Using Excel for Trend Analysis

- Chart



- Formula
 - FORECAST/TREND – Linear
 - GROWTH - Exponential



SERVER (PROCESS) USAGE

PLANNING SERVER CAPACITY



Server (Process) Usage

- Analysis is peak driven
- Measurement approach
 - Benchmark
 - Load testing
 - Monitoring
 - Logging



Benchmark

- Definition

A benchmark is the act of running a computer program, a set of programs, or other operations, in order to assess the relative performance of a server (or any hardware), normally by running a number of standard tests and trials against it.

- Key Parameters

The performance is usually measured in terms of:

- Number of requests that can be served per second (depending on the type of request, etc.);
- Latency response time in milliseconds for each new connection or request;
- Throughput in bytes per second (depending on file size, cached or not cached content, available network bandwidth, etc.).
- The measurements must be performed under a varying load of clients and requests per client.



Reference Benchmark

- Ideally, you have reference benchmark for each of your significant scenarios for the type of machine that you're interested on
 - Perform benchmark on the same machine
 - Relatively easier to do using cloud computing compared to buying in-premise machines
- On the other hand, if this is not possible, estimate the slowness factor. If my scenario is twice as slow as the reference, then my scenarios throughput would be roughly half the reference throughput



Reference Benchmark

- The industry standard for benchmarking system is TPC (<http://www.tpc.org>).
- TPC publishes their benchmark result on their website. Their measurements are for different hardware systems and specifications.
- For an online transactional processing (OLTP) system, TPC maintains the TPC-C specification.

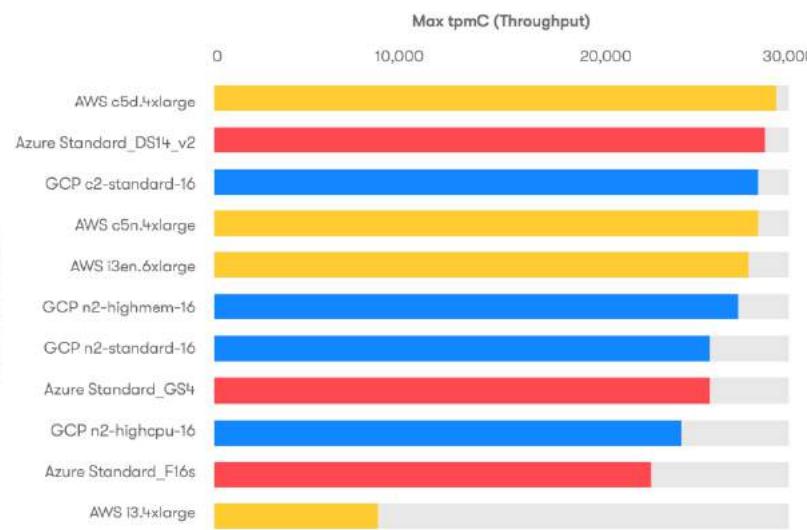
TPC Benchmark C is an on-line transaction processing (OLTP) benchmark. TPC-C involves a mix of five concurrent transactions of different types and complexity either executed on-line or queued for deferred execution. The database is comprised of nine types of tables with a wide range of record and population sizes. TPC-C is measured in transactions per minute (tpmC).



TPC-C Benchmark for Cloud

- For the past few years, Cockroach Lab has performed TPC-C benchmark against different cloud provider.
<https://www.cockroachlabs.com/blog/2020-cloud-report/>

2020 TPC-C Results by Cloud



Price per tpmC by Cloud Machine Type





Benchmark

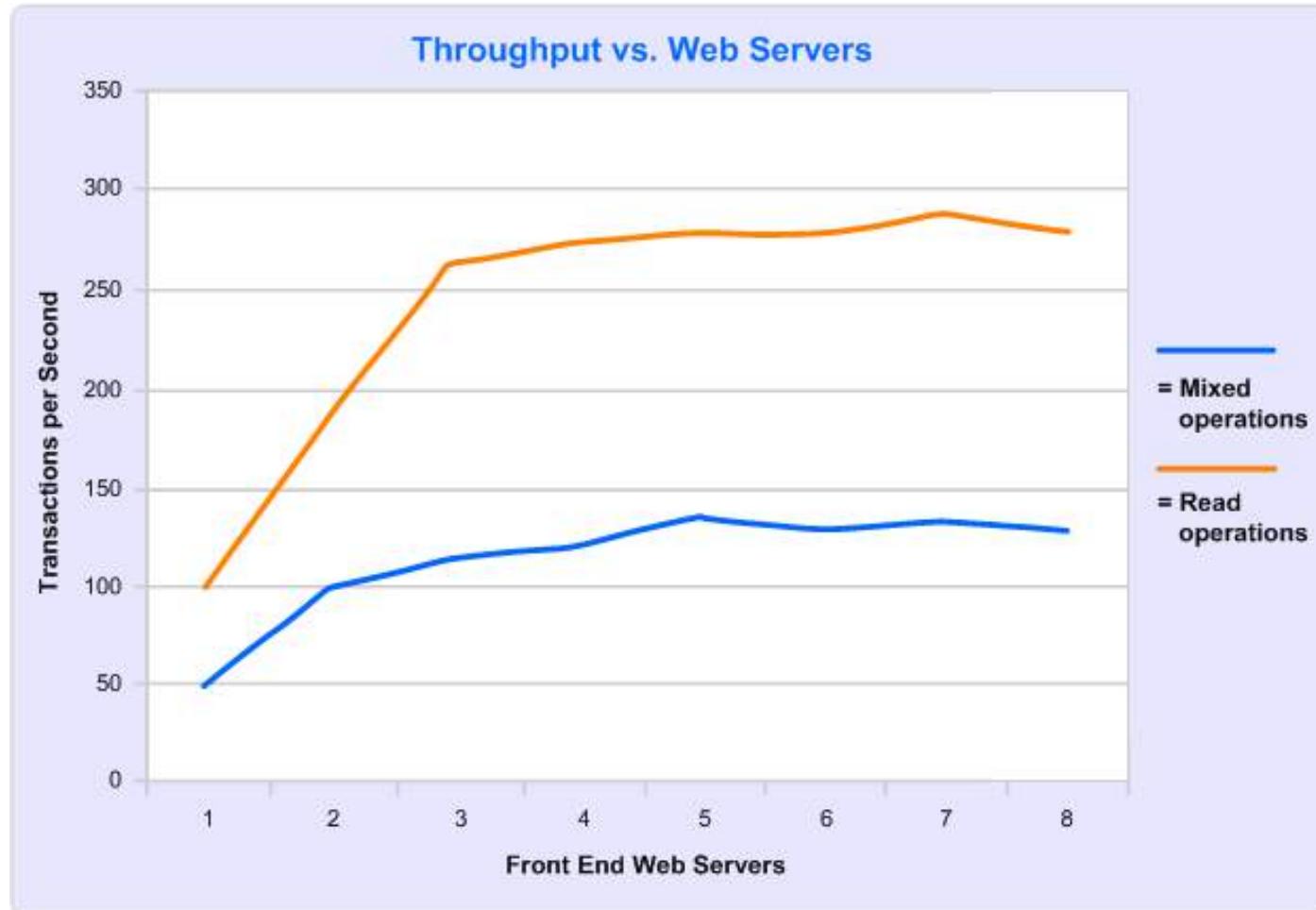
Relations
between
Workload
and
Resource

	1 machine	2 machines	4 machines
10 clients	16.74	30.12	53.87
20 clients	17.04	30.54	55.76
40 clients	16.72	30.81	54.02
80 clients	16.83	30.71	54.33
Max TPS	17.04	30.81	55.76
Appserver CPU Utilization for Max TPS	98%	97%	95%
DB Server CPU Utilization	<1%	1.3%	2.25%
DB Server Disk Utilization	6%	12%	19%

Observe the Non-linearity



Non-linearity



[http://technet.microsoft.com/en-us/library/cc261795\(v=office.12\).aspx](http://technet.microsoft.com/en-us/library/cc261795(v=office.12).aspx)



Amdahl Law

- Amdahl Law
 - Used in parallel computing to predict the theoretical maximum speedup using multiple processors

$$S(N) = \frac{1}{(1-P) + \frac{P}{N}}$$

where

- $S(N)$ is speedup given N number of processor
 - P is the proportion of the program that can be made parallel. $0 < P < 1$
 - N is the number of processor
-
- It can also be applied to other scalability problem. E.g. : machine scalability, etc.

Source: https://en.wikipedia.org/wiki/Amdahl%27s_law



Interpreting Amdahl Law

- If application is 100% parallel, it will get linear scalability
- If application is 100% serial, it can't scale
- Increasing parallelism in application improves the scalability
- Two ways to use Amdahl Law to predict resource requirement
 - Calculate the parallel proportion (P) by conducting load test using increasing number of server and use P to predict the scalability
 - Guesstimate P and use it to predict and fine tune P after the fact.



Interpreting Amdahl Law

- Conducting load test to determine the parallelism would be very sensitive to the workload mix
 - Make sure it mimics the real usage pattern for accurate result
- Some source of serial processing
 - Accessing/updating same data
 - Quota on resources
 - Inefficient algorithm



Universal Scalability Law (USL)

- Universal Scalability Law
 - Used in parallel computing to predict the theoretical maximum speedup using multiple processors (N)

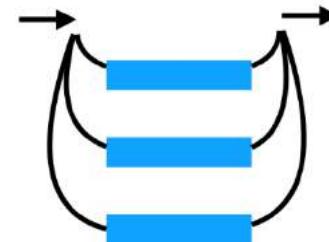
$$X(N) = \frac{\lambda N}{1 + \sigma(N - 1) + \kappa N(N - 1)}$$

where

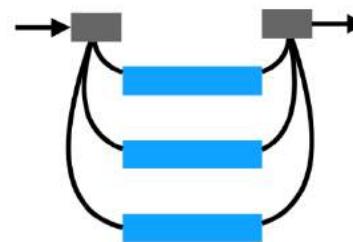
- γ is the coefficient performance (**slope of a line**)
- σ is the coefficient of serialization (defining the limit)
- κ is the coefficient of crosstalk (retrograde)



USL Explanation



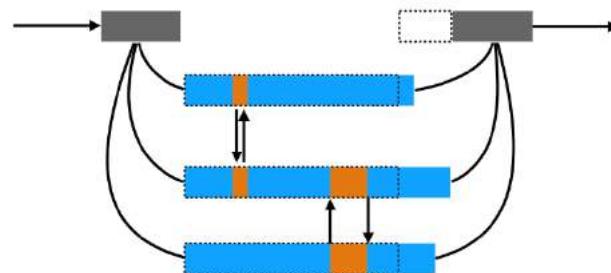
Perfect parallelization



Presence of serial portion

— Linear/Serial —

— Parallel! —

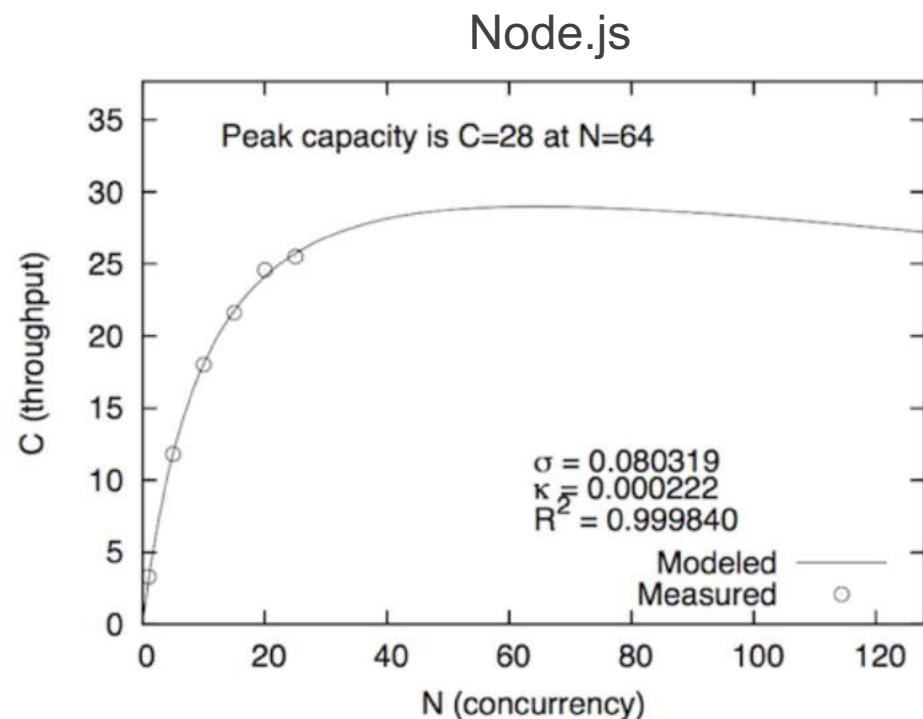
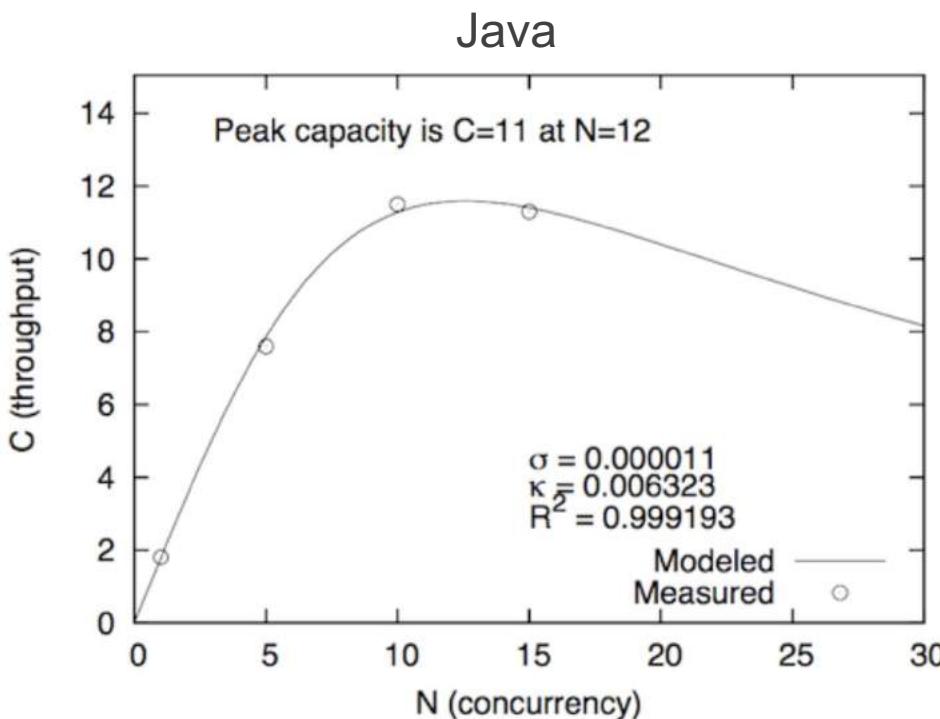


Presence of crosstalk



USL Examples

- Analysis of PayPal's switch from Java to Node.js
(<https://www.vividcortex.com/blog/2013/12/09/analysis-of-paysals-node-vs-java-benchmarks/>)



- In Java, the serialization parameter is lower and the crosstalk parameter is higher. The reverse is true for Node.js.
- Possible explanation: Node.js is a single threaded system and thus it has lower crosstalk.
- It could also be that the new system is just written better.



USL Demo using R

- The coefficient of USL can be obtained by measuring data points and applying regression using the following R package
- <https://cran.r-project.org/web/packages/usl/>

```
benchmark <- read.csv("/path/to/benchmark.txt", sep="")  
  
usl <- nls(tput ~ lambda*size/(1 + sigma * (size-1) + kappa * size *  
           (size-1)), benchmark, start=c(sigma=0.1, kappa=0.01, lambda=1000))  
  
summary(usl)  
  
sigma <- coef(usl)['sigma']  
kappa <- coef(usl)['kappa']  
lambda <- coef(usl)['lambda']  
  
u=function(x){y=x*lambda/(1+sigma*(x-1)+kappa*x*(x-1))}  
  
plot(u, 0, max(benchmark$size)*2, xlab="Size", ylab="Throughput", lty="dashed")  
points(benchmark$size, benchmark$tput)
```



Benchmark

Relations
between
Workload
and
Resource

	1 machine	2 machines	4 machines
10 clients	16.74	30.12	53.87
20 clients	17.04	30.54	55.76
40 clients	16.72	30.81	54.02
80 clients	16.83	30.71	54.33
Max TPS	17.04	30.81	55.76
Appserver CPU Utilization for Max TPS	98%	97%	95%
DB Server CPU Utilization	<1%	1.3%	2.25%
DB Server Disk Utilization	6%	12%	19%

Observe the Non-linearity



Mapping benchmark to our scenarios

Scenario	Slowness Factor	Estimated Throughput (tps)
Reference	1.0	55.76
Login	1.0	55.76
Search	2.0	27.88
Item Details	1.5	37.17
Add to shopping cart	1.5	37.17
Checkout	2	27.88
Payment	4	13.94



Example

Scenario	Mix	Estimated Throughput (tps)	Weighted
Reference	0%	55.76	0
Login	30%	55.76	16.73
Search	20%	27.88	5.576
Item Details	15%	37.17	5.576
Add to shopping cart	20%	37.17	7.434
Checkout	10%	27.88	2.788
Payment	5%	13.94	0.697
Overall			38.8



Putting it all together

- We can get 38.8 tps with the assumptions
 - We use the 4 machines with the same spec as the benchmark in slide 41
 - The workload mix in production is like the one stated in slide 43
 - The slowness factors of the scenario is as stated in slide 42
 - Slowness factor is uniform between app server and DB server
 - Simplification – will not be 100% accurate





When we don't have benchmark

- Obtain the data ourselves.
 - Without any production server → perform simulated load testing.
 - With production server → analyze data from our monitoring tools.



Load Testing

- Many available tools.
- Most would provide ability to define scenarios and rate of incoming requests.
- From the result, we can obtain the response time and/or the throughput.

Jmeter (using UI to configure)

```
config:  
  target: "https://staging1.local"  
phases:  
  - duration: 60  
    arrivalRate: 5  
  - duration: 120  
    arrivalRate: 5  
    rampTo: 50  
  - duration: 600  
    arrivalRate: 50  
payload:  
  path: "keywords.csv"  
  fields:  
    - "keywords"  
scenarios:  
  - name: "Search and buy"  
    flow:  
      - post:  
          url: "/search"  
          body: "kw={{ keywords }}"  
          capture:  
            json: "$.results[0].id"  
            as: "id"  
      - get:  
          url: "/details/{{ id }}"  
      - think: 3  
      - post:  
          url: "/cart"  
          json:  
            productId: "{{ id }}"
```

Artillery.io (declarative)



Using Existing Data

- If we already have a running system, using the existing metrics provides the best data.
- Measure and track relevant metrics over a period of time.

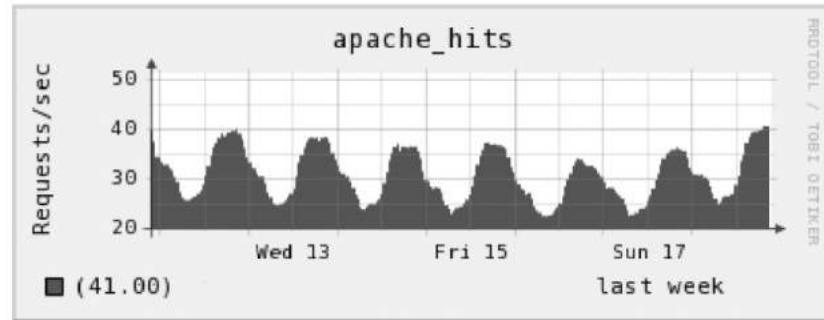
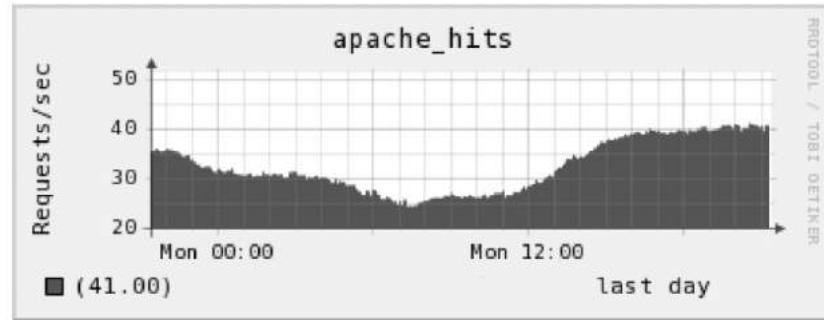
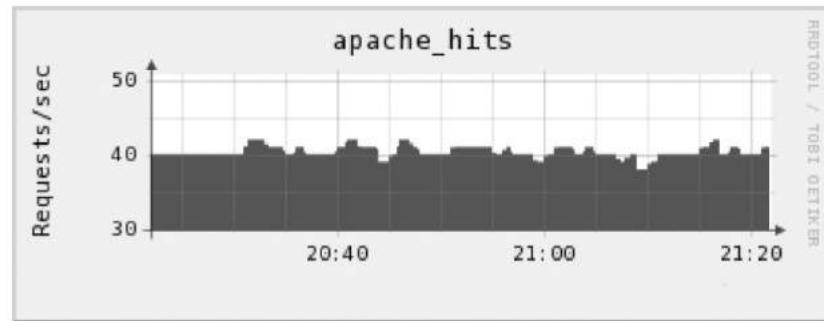


The Importance of Time

- Patterns
- Seasonality
- Time zones

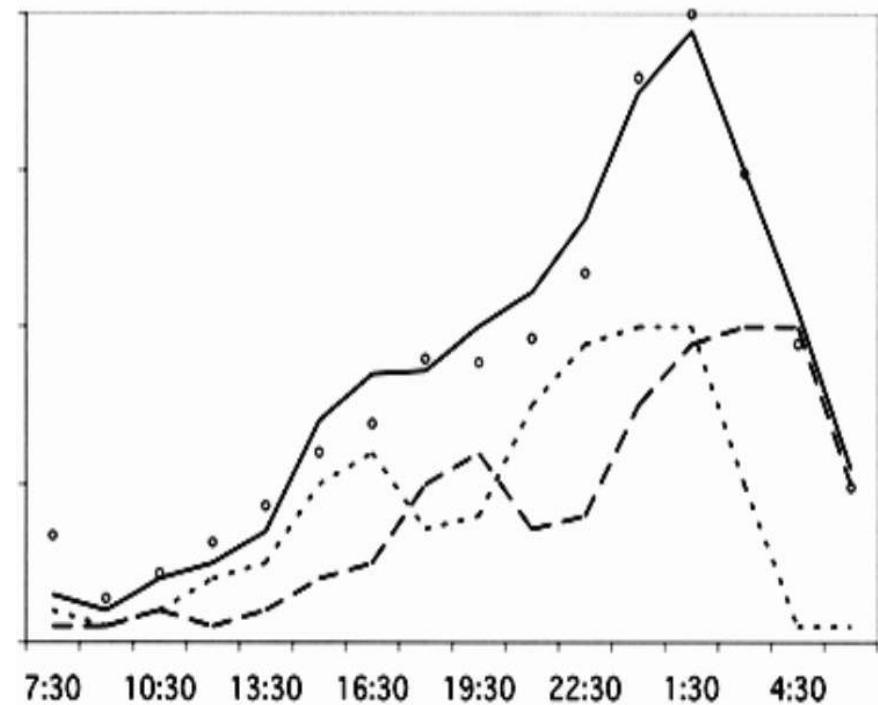
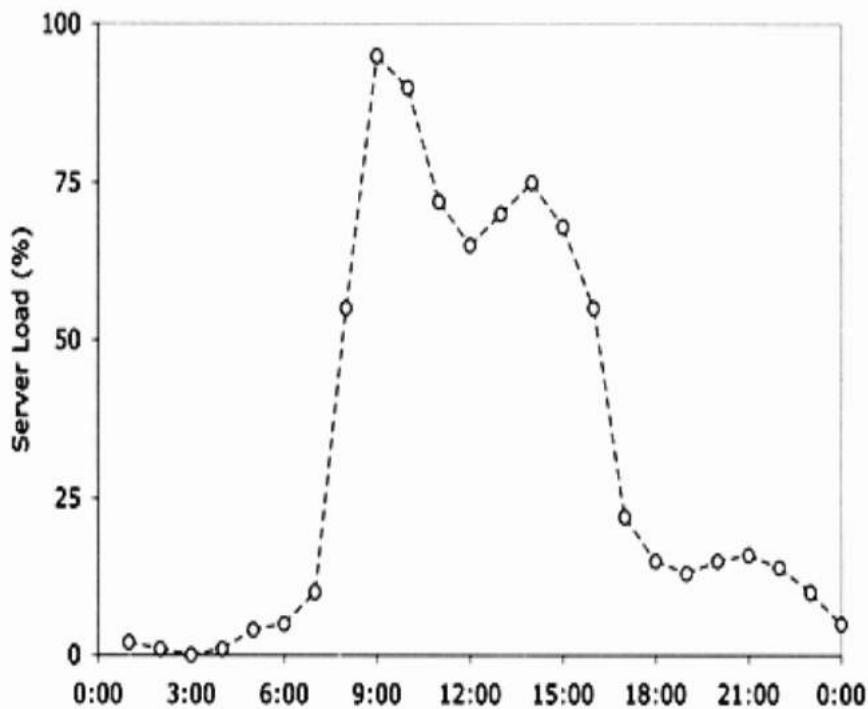


Measurement Time



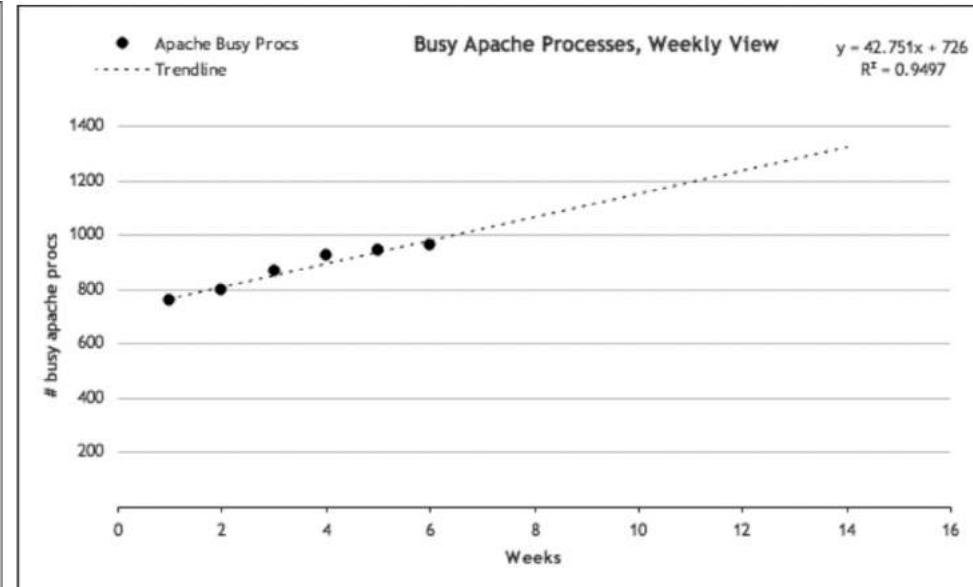
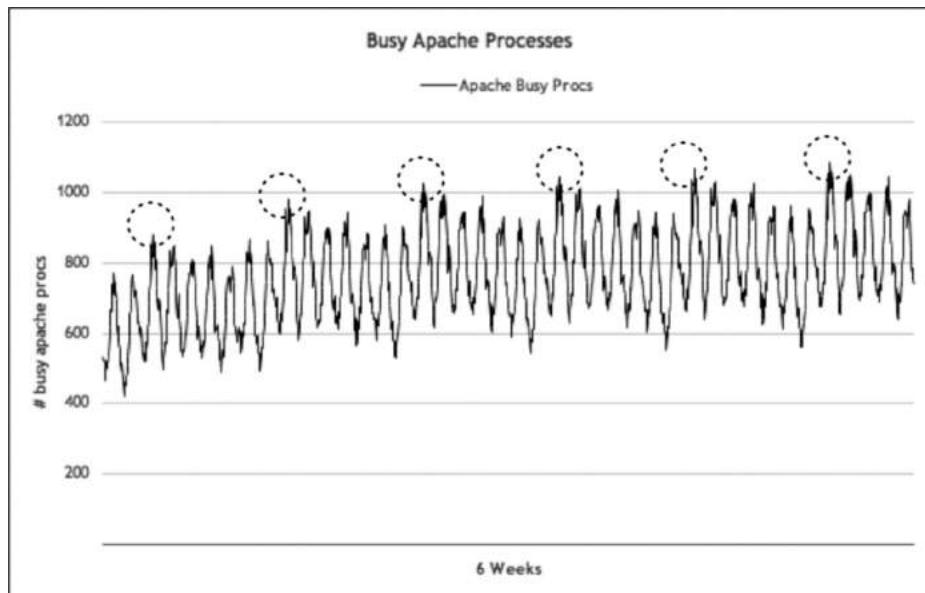


Time Zones (US West – East)





Using Existing Data



- An example from Flickr.
- Peak usage is on Monday every week.
- The trend slowly goes up.
- We can apply the same trend analysis as storage analysis.



Procurement

- Procuring extra capacity involves planning to account for
 - Internal procurement procedure
 - Lead time to get hardware
 - Hardware installation
 - Software installation
 - Integrate into production environment
 - Testing
 - Downtime required
 - Lead time to inform/get approval for downtime
- All this need to be included in the project plan
- With cloud computing technology and proper automation, the time may be drastically reduced for some of the activity



Sample Procurement Planning Template

Procurement Plan for	Extra application server
----------------------	--------------------------

Process	Source	
Internal procurement process (raise request to PO)	Internal organization	(3 working days)
Lead time from PO to delivery	Hardware vendor	(4 weeks)
Hardware installation time	Hardware vendor	(3-5 days)
Software installation time	Software vendor/in house IT	(0.5 day)
Production setup time	In house IT	(1 day)
Testing	In house IT/user	(0.5 day)
Planning buffer		1 week
TOTAL		7 weeks

Estimated downtime/reduced SLA	4 hours
Lead time for downtime approval	1 week



Procurement Timing

- Too late → Not enough capacity
- Too early → But the price of computing usually goes down



Elasticity

- All cloud providers provide the capability to auto-scale instances.
- The auto-scale is controlled by a policy:
 - Scaling adjustment
 - Adjustment type
 - **Cooldown period**
- An alternative approach is to schedule the scaling (if we have historical data).



Elasticity Design Guidelines

- Be proactive – we need to consider the application startup time.
- Be aggressive in scaling upwards, and conservative in scaling downwards.
- Perform scalability analysis to determine threshold.
- Avoid the ping-pong effect.
 - RPS per node after scale up should be more than the scale down threshold.
 - RPS per node after scale down should be less than the scale up threshold.



Summary

- Capacity Planning involves a lot of assumptions
 - Simplify whenever possible
 - Add buffer to compensate for errors
- Do it systematically
 - Easier to communicate to stakeholders
 - Easier to change when assumption changes
 - Use the techniques to get some accuracy



ARCHITECTING SOFTWARE SOLUTIONS

ENSURING MAINTAINABLE ARCHITECTURE

Instructor: Darryl Ng

Email: darryl.ng@nus.edu.sg

Total slide: 41



Objectives

- Understand the characteristics of **maintainable system**
- Understand and be able to apply the different concepts and techniques to make the design more maintainable



Topics

- Definition
- Changeability
 - Changeability Tactics
- Analyzability
- Code Quality Tools



Maintainability

“The ease with which software can be maintained, enhanced, adapted, or corrected to satisfy requirement”

“The ease with which a software system or component can be modified to correct faults, improve performance or other attributes, or adapt to a changed environment.”

IEEE [IEEE610.12-90]



Why it's important

“Maintenance can consume 80% to 90% of the total life cycle cost of software”

- *The Software Project Manager’s Handbook*



Maintainable Software Motives

- Maintainable software allows you to quickly and easily:
 - Fix a bug, without introducing a new bug as you do so.
 - Add new features, without altering existing features.
 - Improve usability
 - Increase performance
 - Make a fix that prevents a bug from occurring in future
 - Make changes to support new environments, operating systems or tools
 - Bring new developers on board your project



Change

“In software system, change is not just inevitable, it is ubiquitous.” - Software Architecture in Practice

Source of Change

- Functional
- Platform
- Integration
- Growth

Likelihood of change

- Deferred functionality
- Requirement gaps
- Vague requirements
- Open ended requirement



Incorporating Change



	Code change	Configuration change	Customizable application	Customizable application
Who	Developer	Developer / Operation	Implementation team	End user
Cost of change	High	Medium	Medium	Low
Cost of development	Low	Medium	High	High
Remarks	Any kind of change is possible. Cost depends on how modifiable the architecture is.	Needs to be incorporated in the original implementation.	This is applicable mostly for B2B system. Different tenant has a dedicated implementation team.	Difficult to achieve, but provide the best value when done correctly



Sub-characteristics

- Modifiability (Changeability)
- Analyzability
- How to achieve reliable change



Modifiability Tactics

1. Split modules
2. Increase Cohesion
3. Reduce Coupling
4. Defer Binding

We will look at the first 3 tactics from 2 perspectives

- High level architecture
- Low level coding/implementation

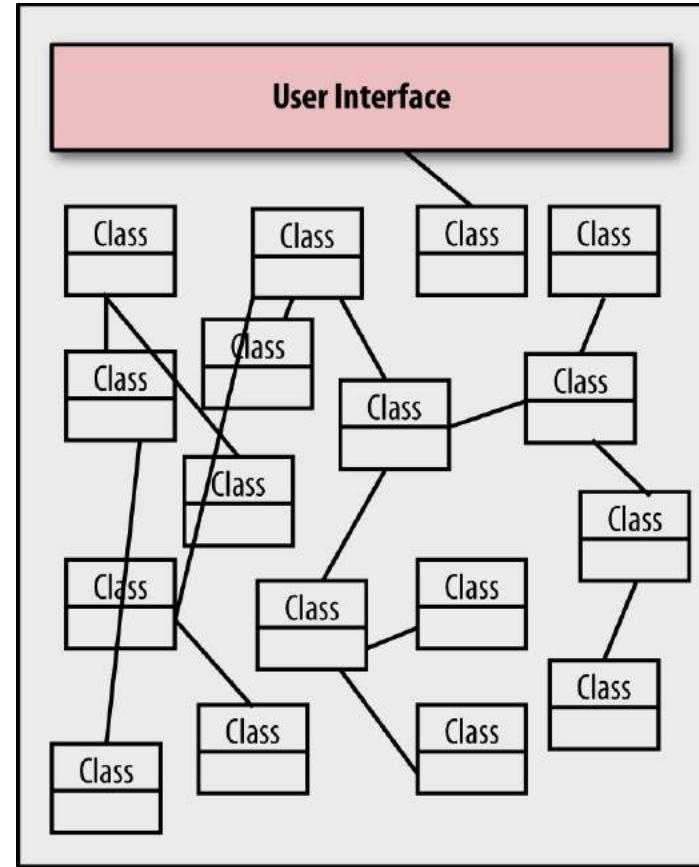


HIGH LEVEL ARCHITECTURE

HOW DIFFERENT ARCHITECTURES SUPPORTS MODIFIABILITY

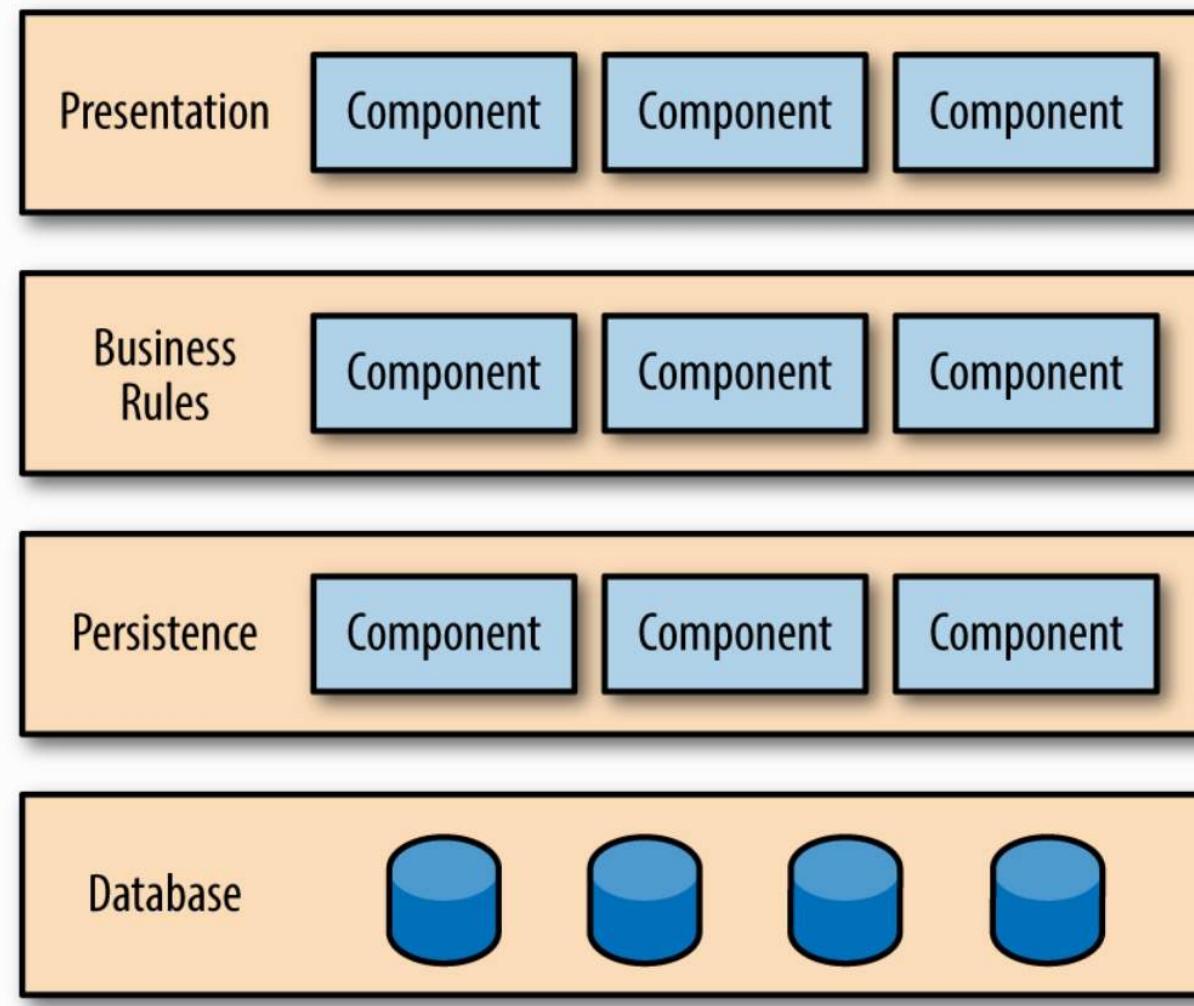


Monolith



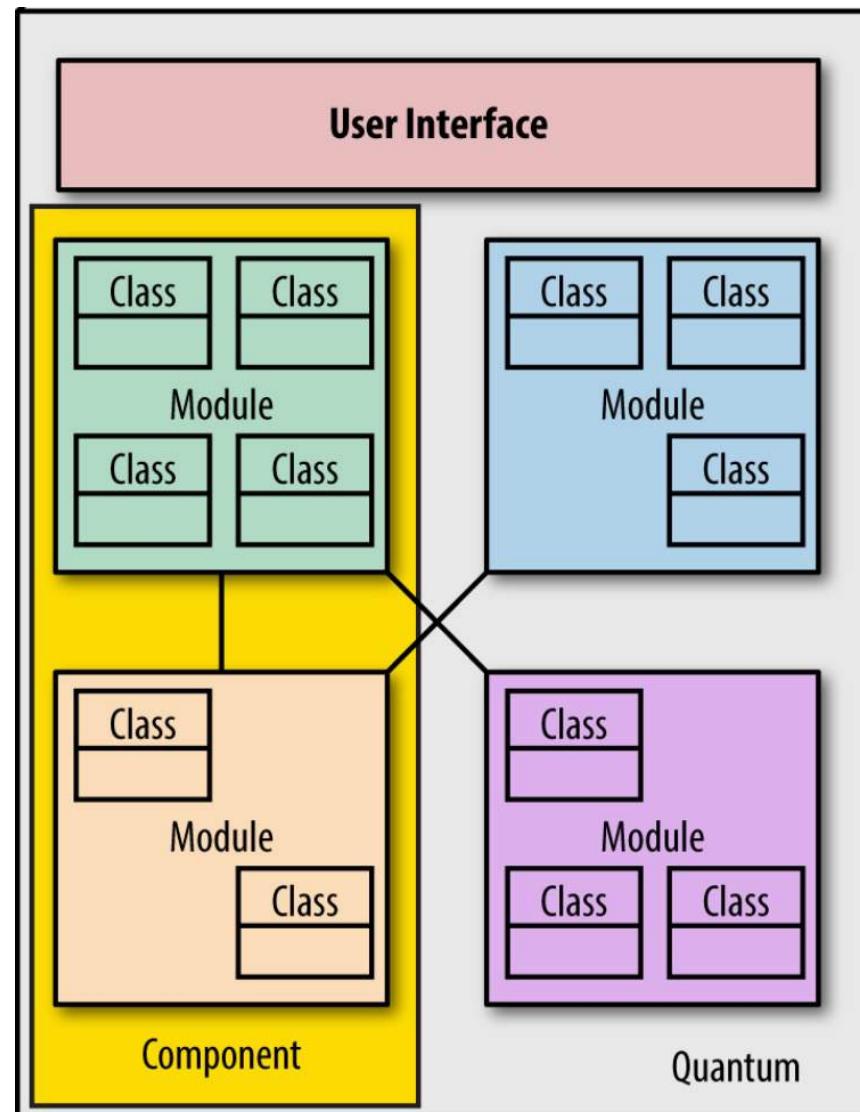


Monolith



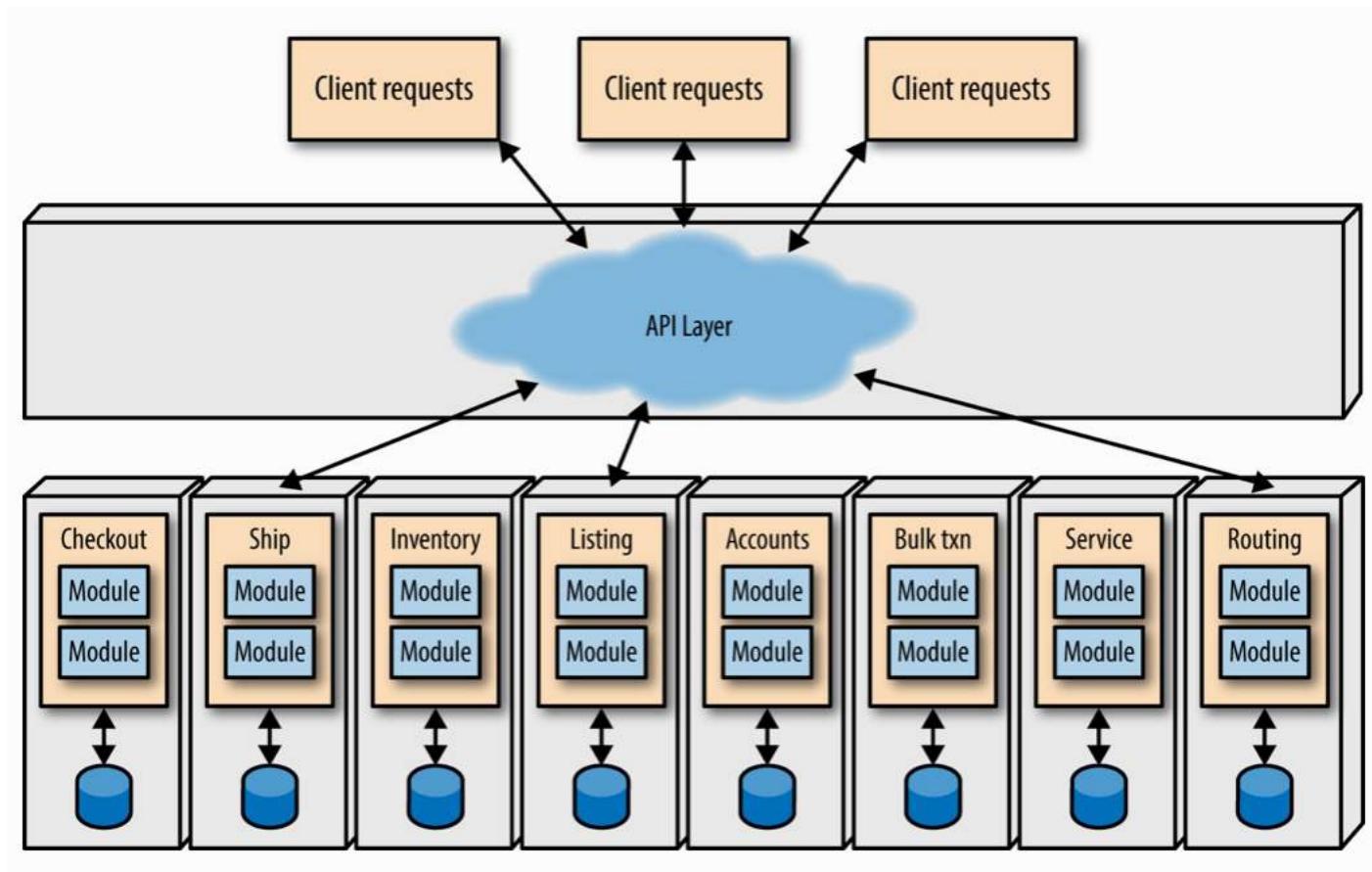


Monolith



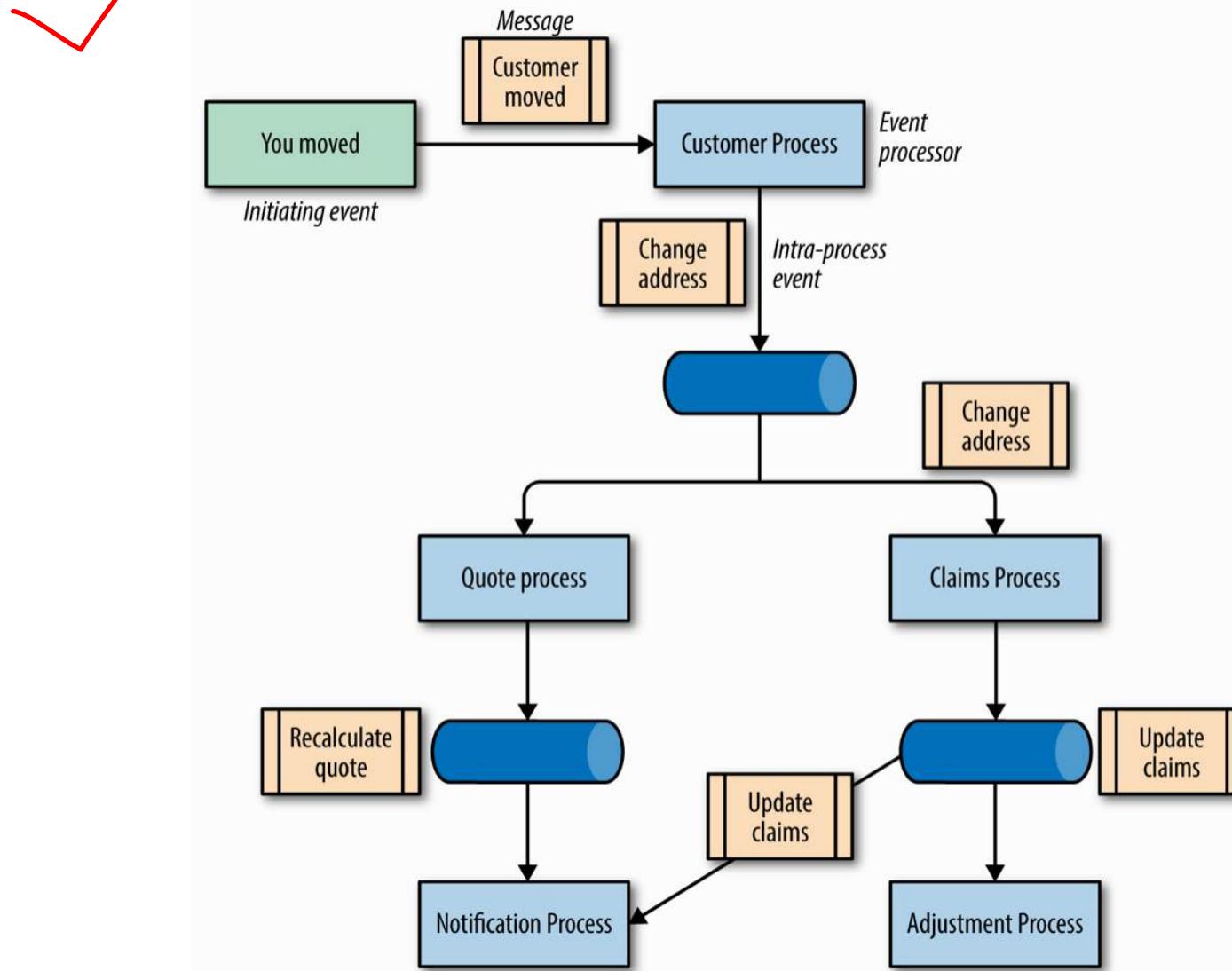


Microservices



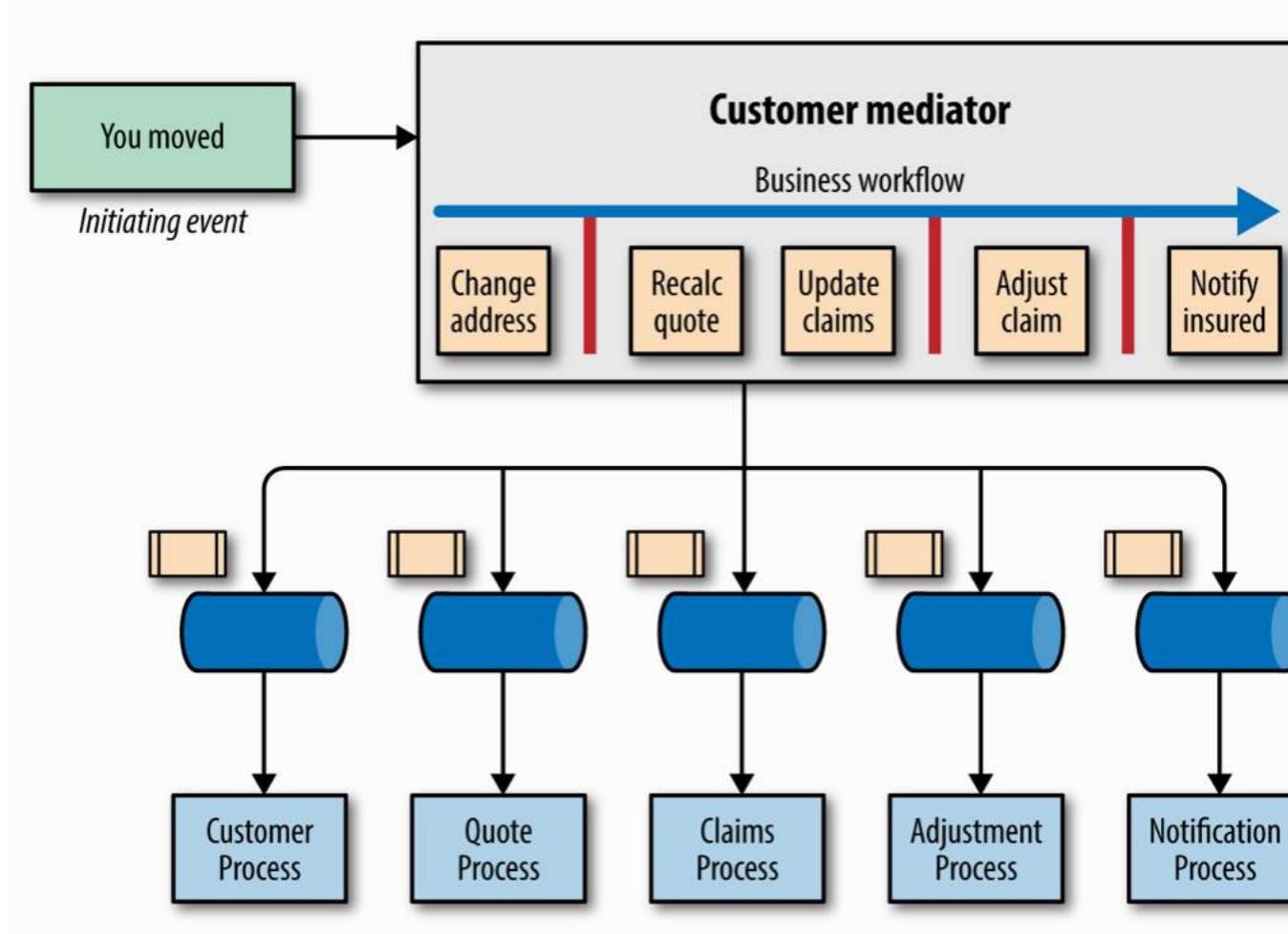


Event Driven Architectures





Event Driven Architectures



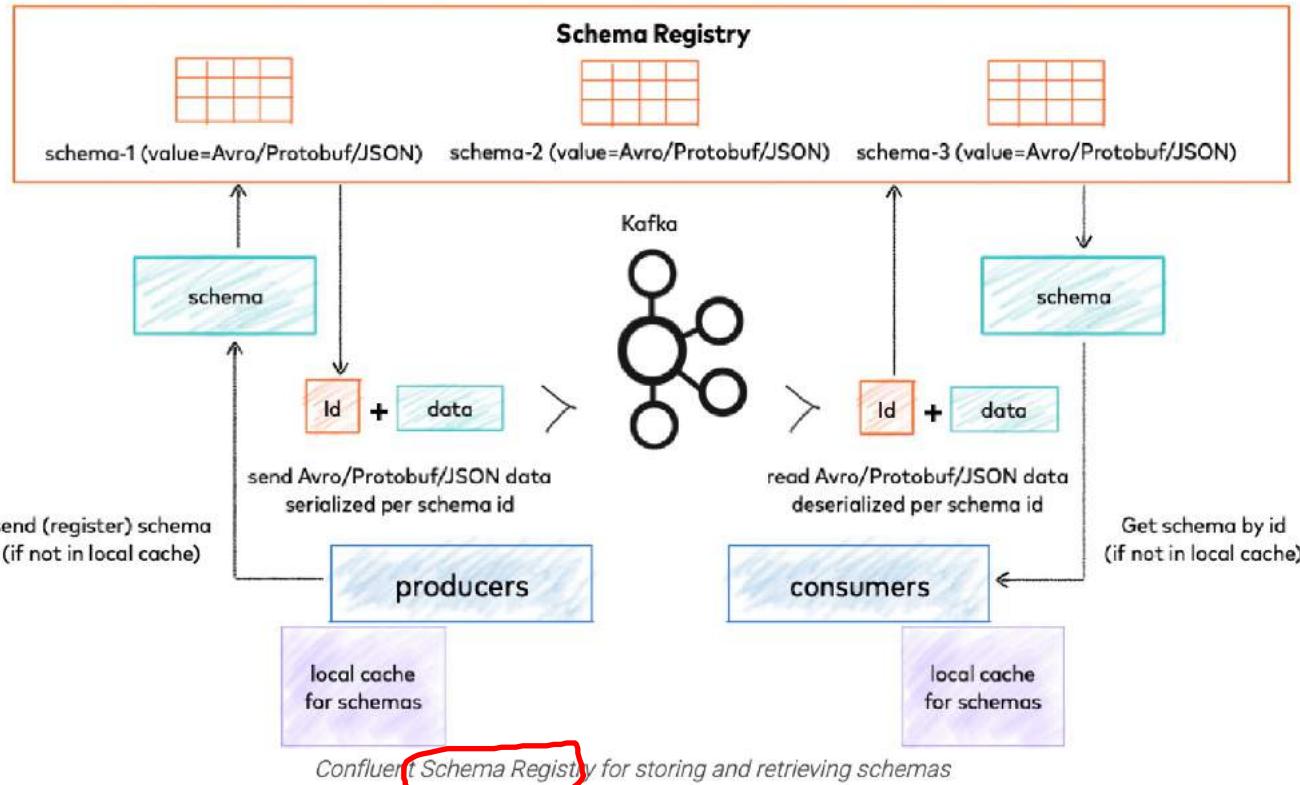


Event driven

- In event driven system, adding a new functionality based on a certain event is easy.
- E.g. A new analytic service can start consuming existing events. None of the producer needs to know about the new service.



Kafka schema registries



- This increases the flexibility and evolvability even further.
- A schema registry stores the schema of the serialized payload together with the version number.
- Every message sent by the producer will be accompanied by the id and the version of the schema.



CODE LEVEL MODIFIABILITY

BEST PRACTICES TO IMPROVE MAINTAINABILITY



Maintainability Tactics (Code Level)

- Contain change
- Create extensible interfaces
- Apply design techniques that facilitate change
- Build variation points into the software
- Use standard extension points
- Achieve reliable change



Contain Change

- Encapsulation
 - Keep the change contained within a component.
 - Callers need not concern about the details.
 - E.g.: Discount Engine, Strategy Design Pattern.
- Separation of concerns
 - Clear and distinct set of responsibility for each of the component
 - Attempt so contain changes to one element.
- Single point of definition
 - Implement all data types, algorithms, configuration, data schemas, etc only once.
 - Prevent the need to “synchronize” the change in various parts of the system.



Create Extensible Interfaces

- Design API with object as parameters
 - Definition of the object can be extended without changing the API method signature
 - Object parameters can be made optional for backward compatibility
- Using flexible data format
 - XML, JSON
- Interface versioning
 - Explicitly state the version of the API
 - End of support for a particular version can be easily communicated



Apply Design Technique That Facilitates Change

- Abstraction and Layering Patterns
 - Intention: changes in one parts should have minimal impact on others
- Generalization
 - Group similar scenarios into general patterns and create architectural components to handle the overall group
- Inversion of Control
 - Allow reconfiguration of components



Build Variation Points into the Software

- Make elements replaceable
- Parameterize changeable property in configuration
- Separate physical and logical processing
 - Physically things changes, but logically they are still the same
 - E.g. convert all the different physical data format into a standard logical data structure and use this logical structure on the rest of the systems
- Break processes into steps
 - Changes may be contained in some steps



Using Standard Extension Point

- Leverage extension points that are built into standard technologies
 - ODBC, JDBC: support various databases
 - JSON, SOAP, REST: calling services
 - JCA: standard adapter standard for Java



Defer Binding

- Making an application fully customizable by the user is one way to provide modifiability.
- This requires a good user experience expertise and good understanding of the domain.

Demo

- A good example is how Amplitude analytics enable full customization of cohort analysis.
- Context for the demo:
 - The user would like to analyze the retention of users from different cohorts.
 - Amplitude provides a customizable cohort analyzer that does not require the users to write data munging logic.



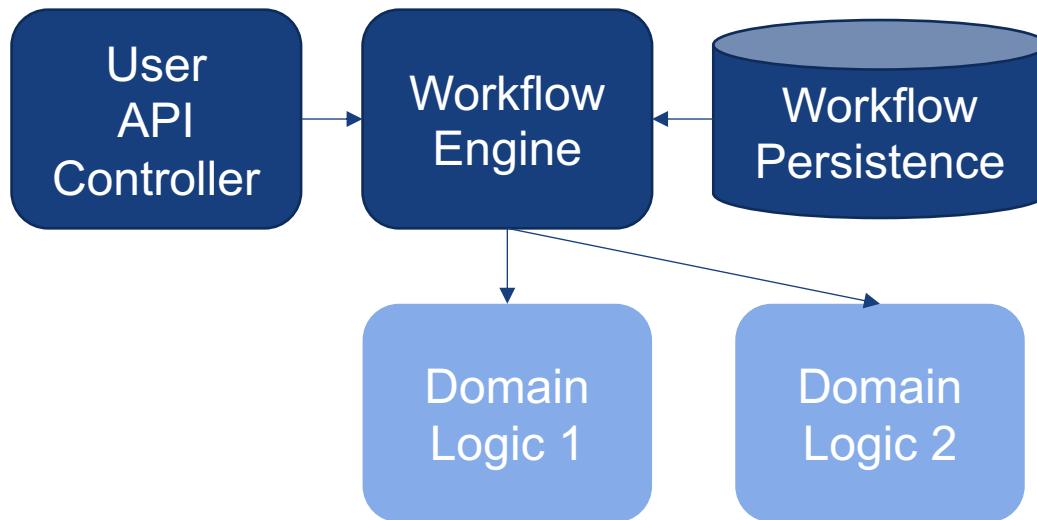
Defer Binding (Use of Workflow Engine)

Designing an application that utilizes workflow engine allows logic customization without recompiling the application.

Example: Supporting different approval workflow for different client.



Admin/implementer can design a workflow for a specific tenant or specific roles. The workflow logic will be persisted.

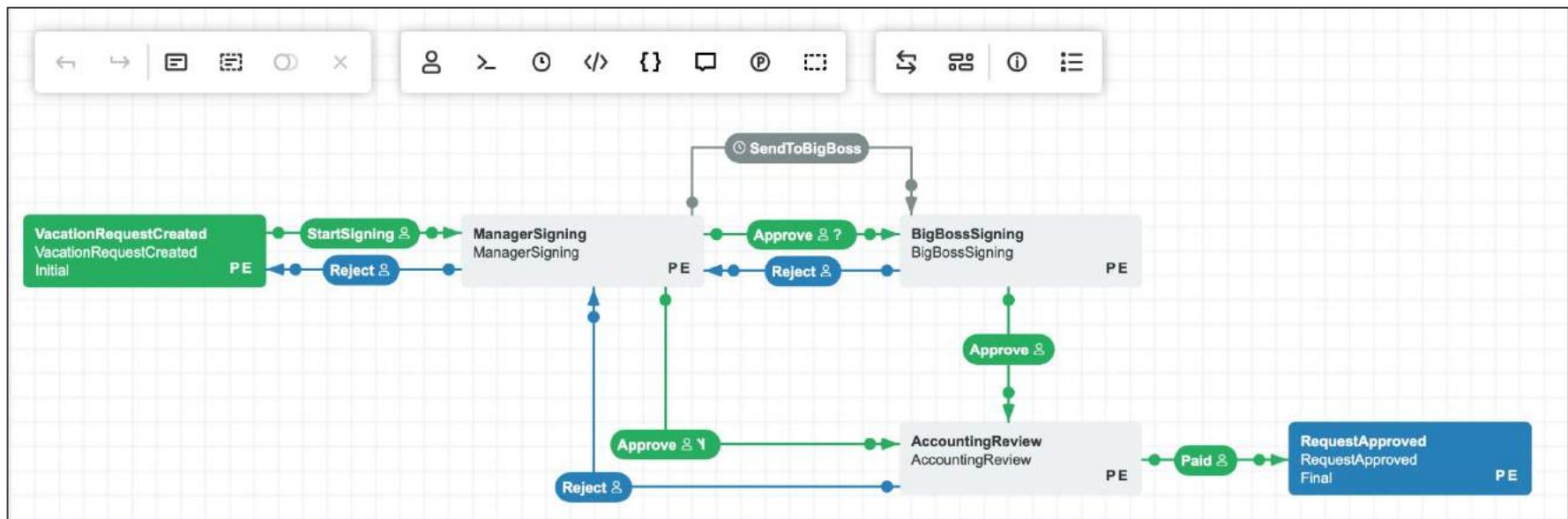


During actual run time, when a user from a specific tenant/role uses the system, the workflow logic will be deserialized from the persistence and executed.



Defer Binding (Use of Workflow Engine)

There are many available workflow engine tools that provides visual designer.





Analyzability

- How do we detect problems?
- How do we identify the root cause of the problem?
- How to make necessary change and deploy them
 - Similar to other type of changes



Analyzability Tactics

- Automated error detection and reporting
 - Some platform may provide features on crash detection and reporting
 - UNIX core dump
 - Log file monitoring
- Good logging practice
 - Enough information to inform the investigation
- Allow system monitoring
 - Some standard protocols available



JMX/SNMP

J2SE 5.0 Monitoring & Management Console: service:jmx:rmi:///jndi/rmi://localhost:7001/jmxrmi

Connection

Summary Memory Threads Classes MBeans VM

MBeans

Tree

- JMImplementation
- org.apache.activemq
 - localhost
 - Broker
 - Connection
 - ID_wireless.hiram.chirino-49454
 - Connector
 - tcp://wireless.hiram.chirino_616
 - tcp://wireless.hiram.chirino_616
 - NetworkConnector
 - Queue
 - TEST.FOO
 - Subscription
 - Topic
 - ActiveMQ.Advisory.Connection
 - ActiveMQ.Advisory.Consumer.Queue
 - ActiveMQ.Advisory.Producer.Queue
 - ActiveMQ.Advisory.Queue
 - ActiveMQ.Advisory.Topic

Attributes Operations

Name	Value
ConsumerCount	1
DequeueCount	10
EnqueueCount	10
QueueSize	0

Refresh

Paessler SNMP Tester 3.0

File Help

1. Set SNMP Settings

Local IP: Any
Device IP: 10.0.0.1
Port: 161
SNMP Version: SNMP V1
Community: public
V3 Authentication: MD5 SHA
V3 Password:
V3 Encryption Key:
Advanced Settings Force 32bit "Slow" Tweak Signed

2. Select Request Type

32 bit Traffic Counter (V1/2/3): 1
64 bit Traffic Counter (V2/3): 1
Custom OID:
Read Device Uptime
Scan Available Standard Interfaces
Scan Available OIDs from OIDLIB:

3. Run Test Repeat every 5 s

Save Log to File Clear Log

New Test

13.11.2008 15:14:22 (1 ms) : Start using SNMP V1
13.11.2008 15:14:22 (3 ms) : -----
13.11.2008 15:14:22 (4 ms) : Value: 52143100
13.11.2008 15:14:22 (4 ms) : Done

New Test

13.11.2008 15:14:31 (1 ms) : Start using SNMP V1
13.11.2008 15:14:31 (4 ms) : -----
13.11.2008 15:14:31 (4 ms) : Value: 2323555943
13.11.2008 15:14:31 (5 ms) : Done

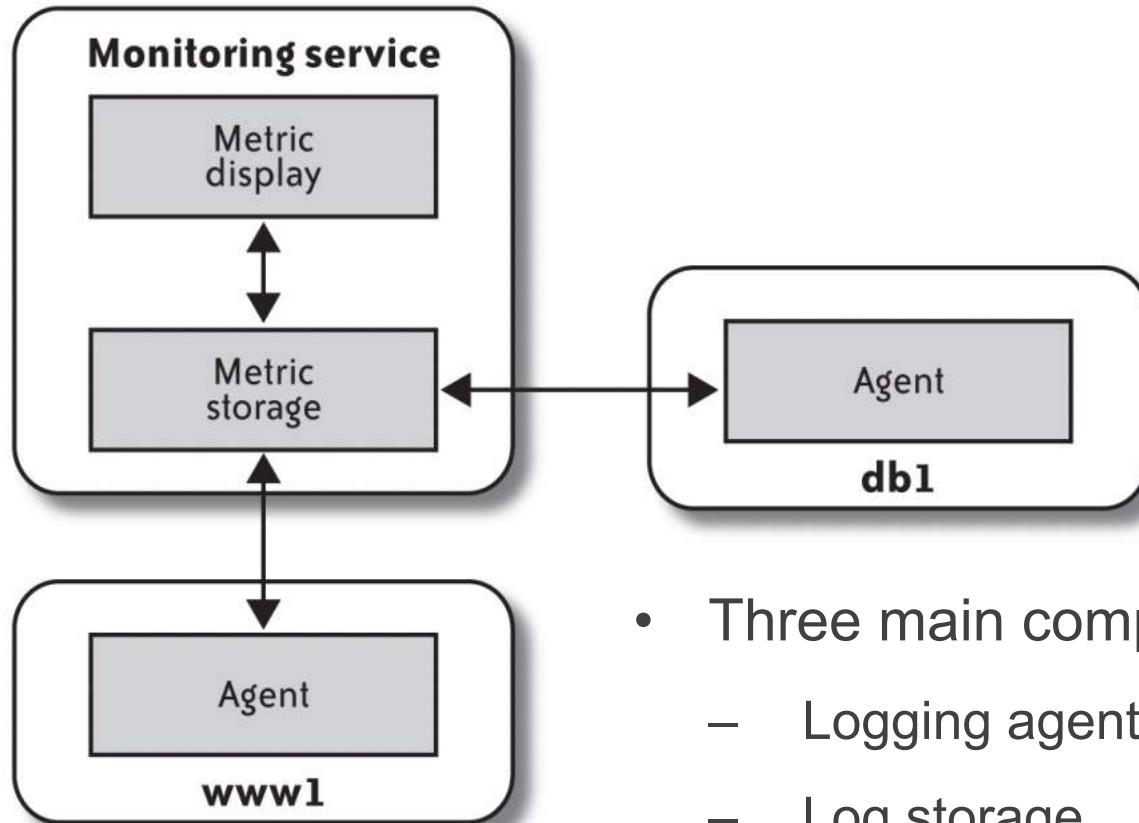
New Test

Scanning Standard Interfaces...
Start Scan
Current: 1.3.6.1.2.1.2.2.1.2.1
Description: Adaptive Security Appliance 'EXTERN' interface Testing...
Ok
GetNext=1.3.6.1.2.1.2.2.1.2.2
Current: 1.3.6.1.2.1.2.2.1.2.2
Description: Adaptive Security Appliance 'INTERN' interface Testing...
Ok
GetNext=1.3.6.1.2.1.2.2.1.2.3
Current: 1.3.6.1.2.1.2.2.1.2.3
Description: Adaptive Security Appliance 'management' interface Testing...
Ok
GetNext=1.3.6.1.2.1.2.2.1.3.1

Found standard interfaces:
1: (001) Adaptive Security Appliance 'EXTERN' interface,Connected,100 MBit/s,Ethernet,
2: (002) Adaptive Security Appliance 'INTERN' interface,Connected,100 MBit/s,Ethernet,
3: (003) Adaptive Security Appliance 'management' interface,Not Connected,10 MBit/s,Ethernet



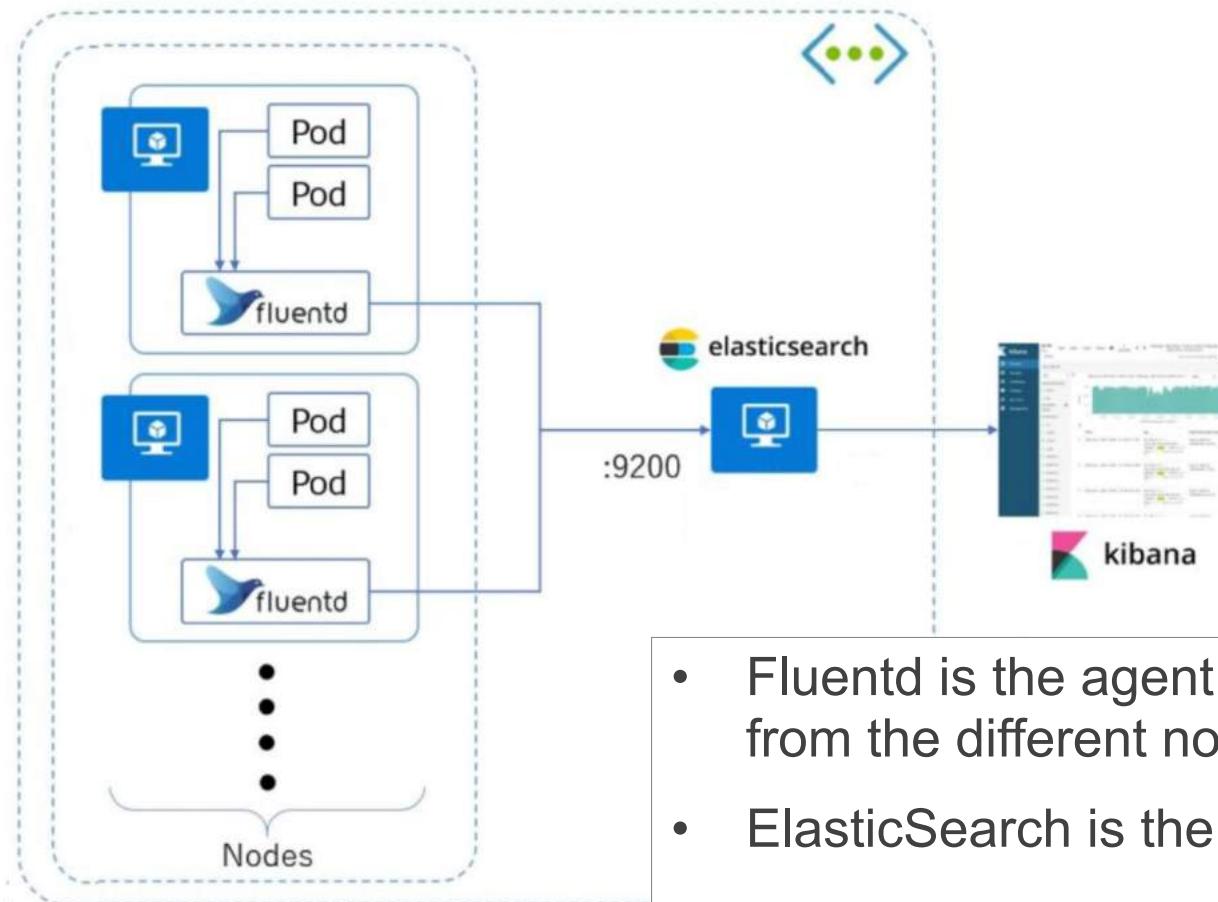
Logging in Microservices



- Three main components of logging
 - Logging agent
 - Log storage
 - Log dashboard (with search capabilities)



EFK Stack in A Kubernetes Cluster



- Fluentd is the agent collecting the logs from the different nodes.
- ElasticSearch is the storage.
- Kibana is the dashboard.
- In Microservices, it is important to provide a correlation id to trace the logs.

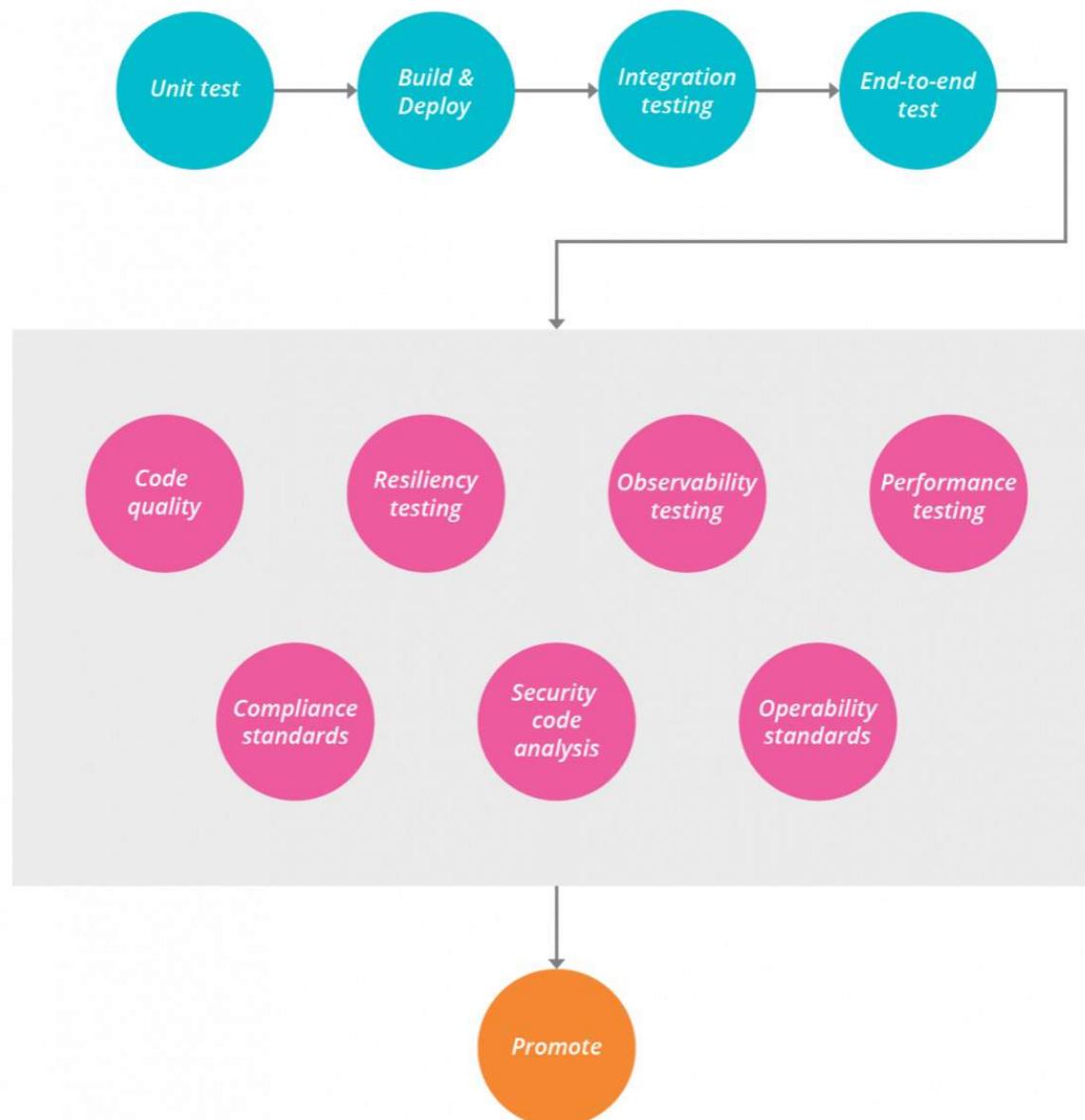


Achieve Reliable Change

- Software configuration management
 - All changes are tracked and versioned in a central place
- Automated testing
- Continuous integration
 - Automate the build-test-release process
- Ability to rollback unsuccessful deployments
- Infrastructure as code
 - Environment configuration managements
 - Secrets Management
- Hypothesis driven testing
- Fitness Function



Fitness Function





An Example of Fitness Function

```
public void testMatch(){
    DependencyConstraint constraint = new DependencyConstraint();

    JavaPackage persistence = constraint.addPackage("com.xyz.persistence");
    JavaPackage web = constraint.addPackage("com.xyz.web");
    JavaPackage util = constraint.addPackage("com.xyz.util");

    persistence.dependsUpon(util);
    web.dependsUpon(util);

    jdepend.analyze();

    assertEquals("Dependency mismatch", true, jdepend.dependencyMatch(constraint));
}
```

- This is to ensure that in a layered architecture, no code breaks the layer dependencies.



Code Quality Tools

- There are tools that analyze the source code and try to assess the quality and highlight bad coding practices
 - FindBugs
 - Checkstyle
 - PMD
 - SonarQube
 - CodeCity
 - and many more
- Need to understand what the metrics mean
- Useful to get the relative quality progression, but not for absolute comparison



Example (SonarQube)

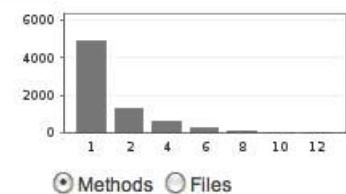
Lines of code
63,281 ▲
112,731 lines ▲
23,739 statements ▲
1,230 files ▲

Classes
1,325 ▲
171 packages ▲
7,147 methods ▲
1,409 accessors ▲

Duplications
0.8%
1,023 lines ↘
45 blocks
27 files

Duplicated lines that can be reduced
121 lines ↗

Complexity
1.9 /method
10.3 /class
11.1 /file
Total: 13,617 ▲



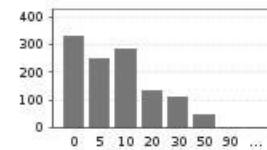
Issues
2,120 ↗
Technical Debt
179.2 days ↗



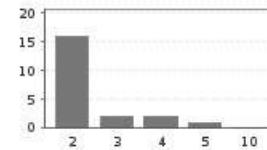
Package tangle index
11.9%
> 55 cycles ▼

Dependencies to cut
35 between packages
58 between files ▲

Response for Class
15 /class



LCOM4
1.0 /class
1.8% files having LCOM4>1



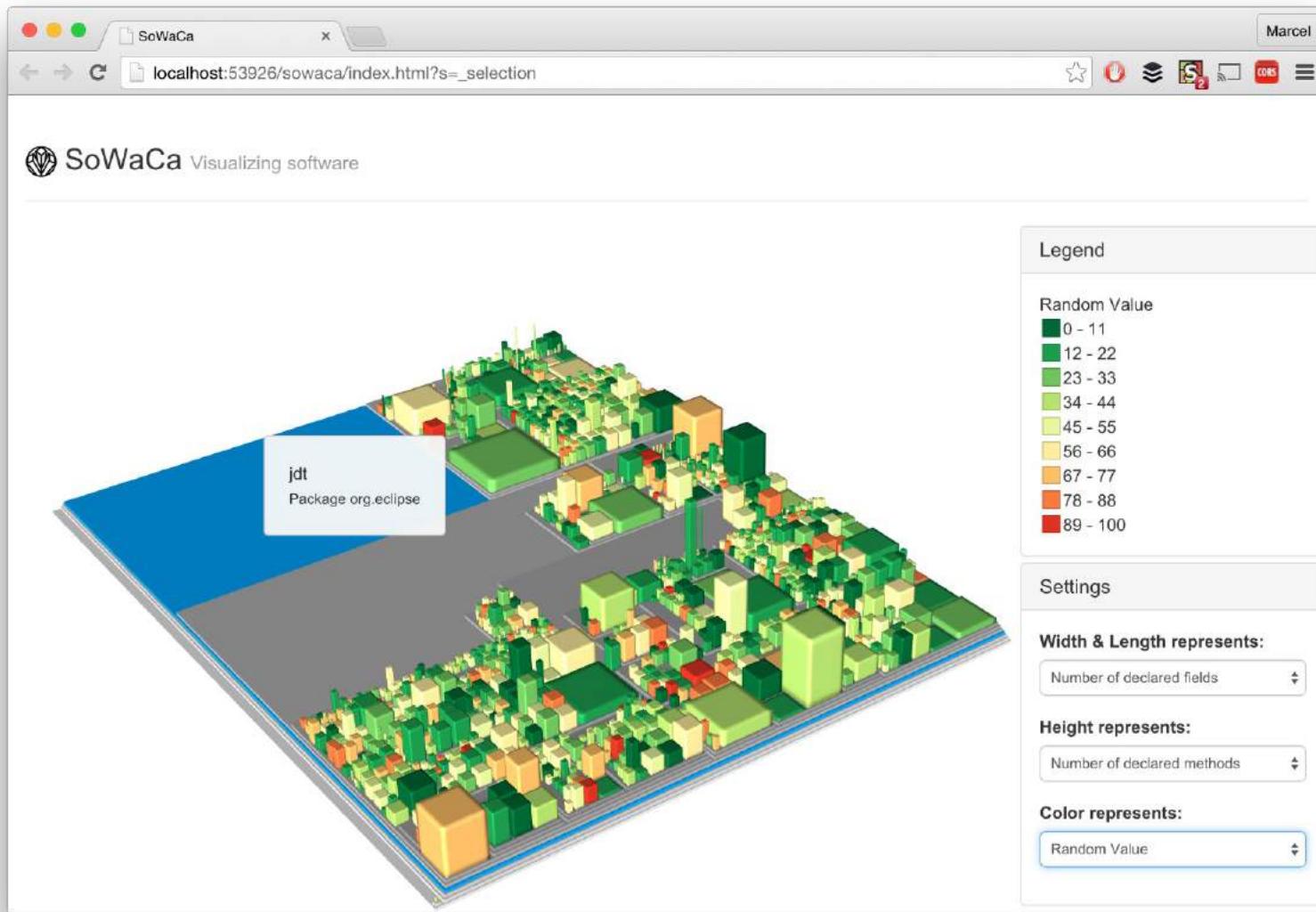
Code coverage
70.6% (+1.0)
71.2% line coverage (+0.9)
68.9% branch coverage (+1.2)

On new code
76.6%
2,418 lines to cover
78.2% line coverage
72.5% branch coverage

Unit test success
100.0% (+0.0)
0 failures (+0)
0 errors (+0)
2,762 tests (+253)
2 skipped (+2)
13:57 min (-5:08 min)



Example (CodeCity)





Summary

- Maintainability is a critical aspect that address concerns from many stakeholder
- Maintainability involves analyzability, changeability, stability and testability