Architectural Patterns & Styles

Software Architecture
3rd Year – Semester 1
Lecture 10

What are Architectural Styles?

- An architectural style, sometimes called an architectural pattern, is a set of principles that shapes an application, a system or a system of systems.
- An architectural style improves partitioning and promotes design reuse by providing solutions to frequently recurring problems.
- Provides a common language to understand systems – often not coupled with specific technologies/frameworks (Java, .NET, etc.) thus facilitates higher-level conversations.

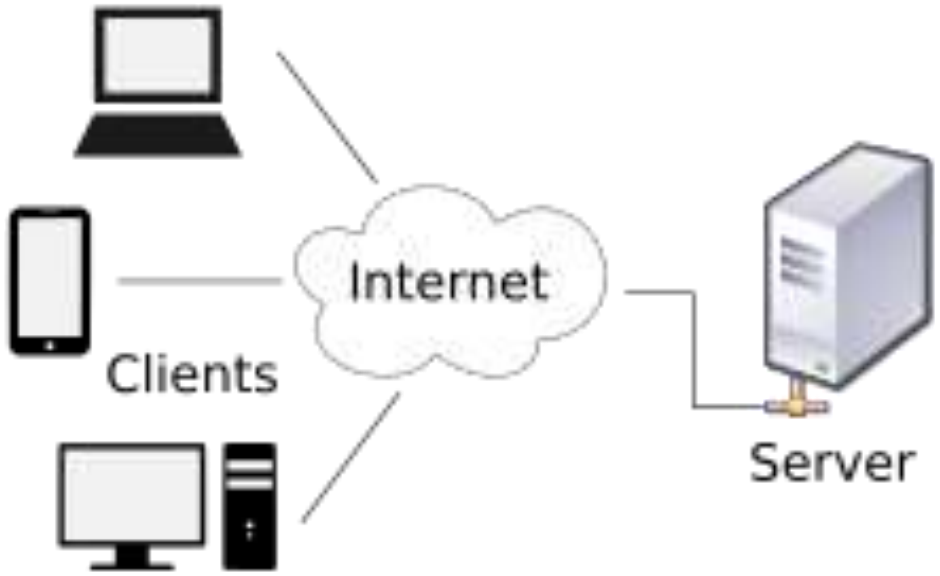
Common Architectural Styles

- Monolithic
- Client-Server
- Component-based
- Layered
- N-Tier
- Object Oriented
- Blackboard
- Event Driven
- Domain Driven
- Plugin
- Microservice
- Peer-to-Peer
- Rule-based
- Service Oriented
- Message Bus
- Pipe and Filter
- REST
- Publish-Subscribe
- Cloud

block chain

Client-Server Architecture

- Server can server multiple clients at a time
- Server usually provides a function, data, content, etc... to the client
- Centralized
- Client knows how to locate the server
- Connection could be HTTP, RPC, etc....



Client-Server Architecture in practice

- Examples:
  - Outlook Email - Outlook [Thick Client] connects to Microsoft Exchange Server via SMTP/POP
  - Gmail - Web Browser [Thin Client] connects to Google Mail Server via HTTP

use general purpose software

- Advantages:
  - High Security (centralized)
- Disadvantages:
  - Single point of failure (server centric)

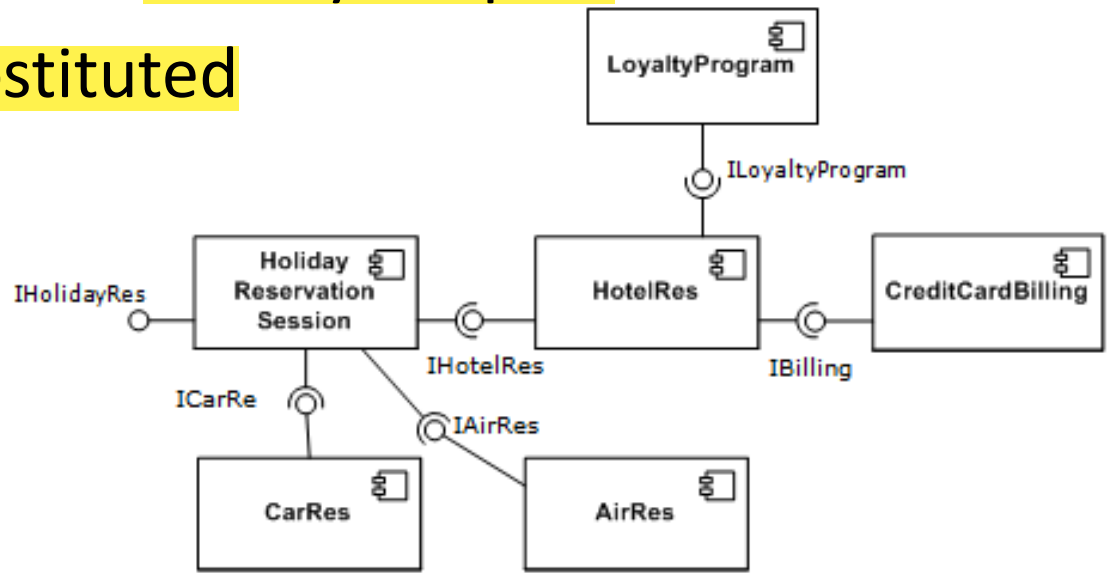
so no body can use system

  - Maintenance & Downtime issues

Component-based Architecture

- Emphasize on Separation of Concerns
- Components are reusable
- Components are highly cohesive and loosely coupled
- Components are made to be substituted

one component use one thing only

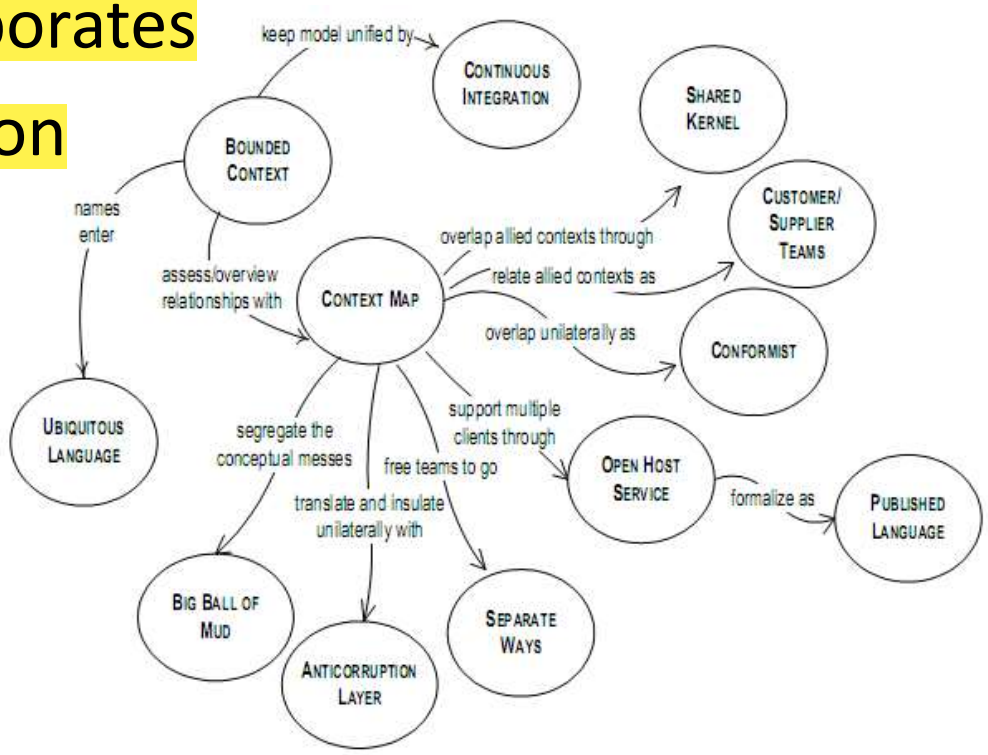


Component-based Architecture in practice

- Examples:
  - Java libraries (.jar files)
  - Windows OS .dll files
- Advantages:
  - Reusability (reduce development cost)
  - Extendibility – each component can be further adjusted
- Disadvantages:
  - Managing a large component base may be harder

Domain Driven Architecture

- Focus mainly on Domain & Logic around it
- Technical and Domain experts collaborates
- Ontology – Knowledge Representation

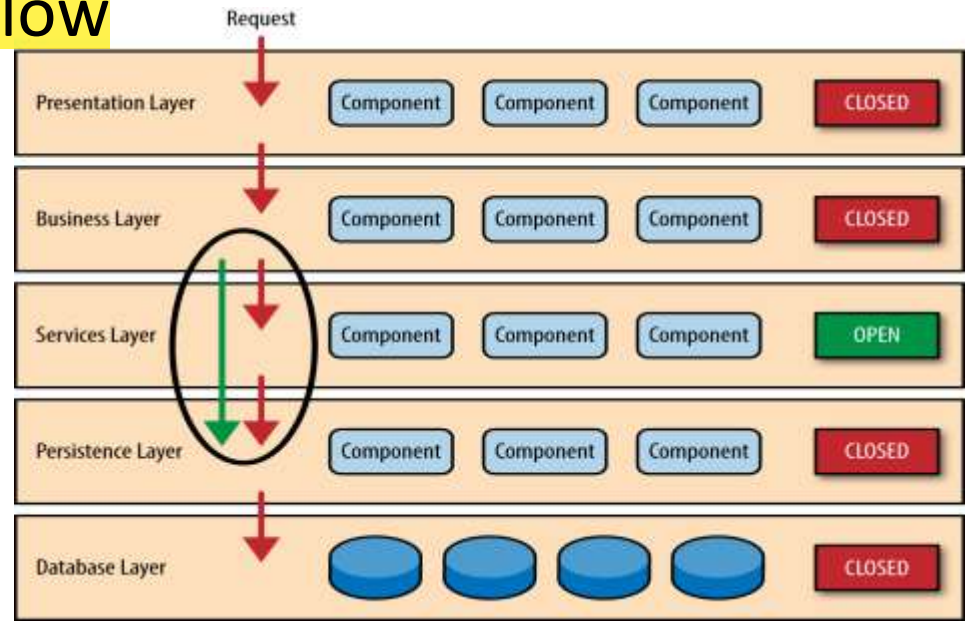


Domain Driven Architecture in practice

- Examples: wiki pedia, forams
  - Web Content Management Systems
- Advantages:
  - Easy for the Domain experts
- Disadvantages:
  - When there is a larger team the system gets very complex and disorganized

Layered Architecture

- Groups related functions in to layers
- Layers are stacked on top of each other
- Typically components in one Layer can communicate with components in same layer or Layers below
- Layering helps Separation of Concerns

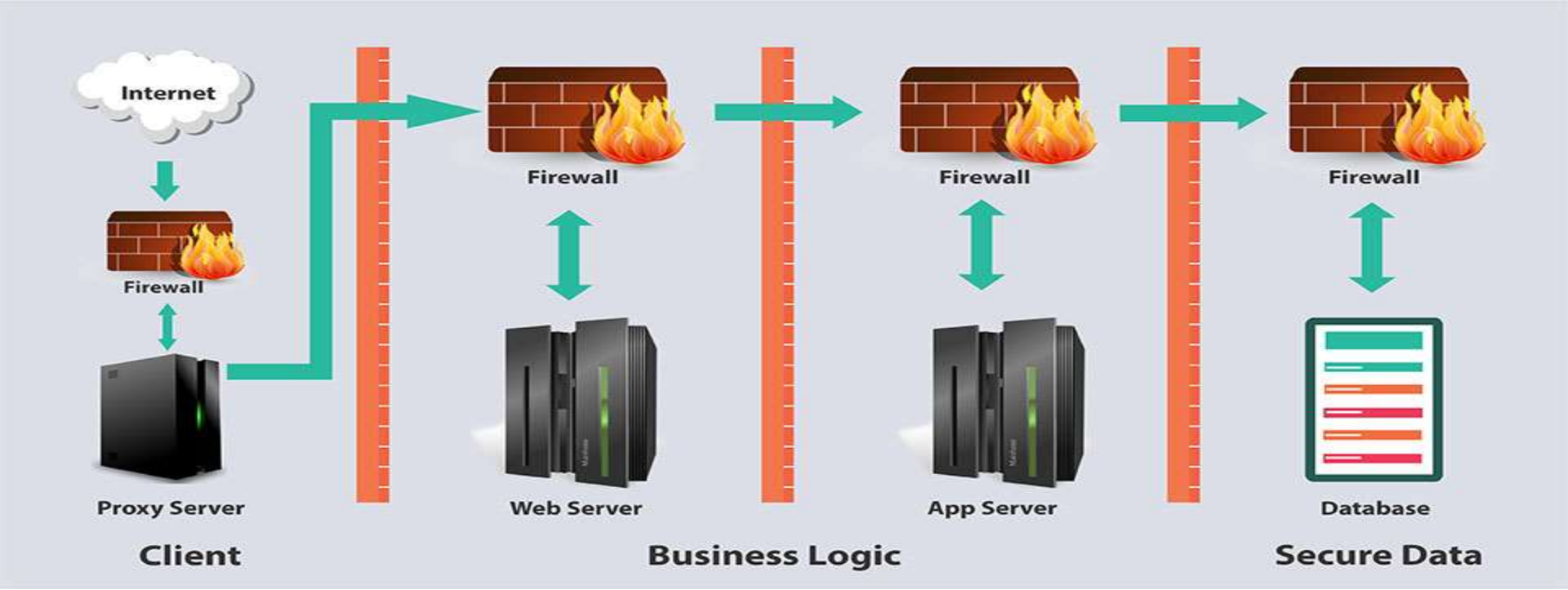


Layered Architecture in practice

- Examples:
  - Applications with Presentation, Service and Data Layers
  - TCP/IP
- Advantages:
  - Shares many advantages similar to Component Based Architecture
  - Layered Architecture can extend to N-Tier Model
- Disadvantages:
  - Components in bottom layers cannot communicate with top layers without Cyclic dependencies

N-Tier Architecture

- Can be considered as an extension to Layered Architecture with each layer having the ability of executing on different physical locations

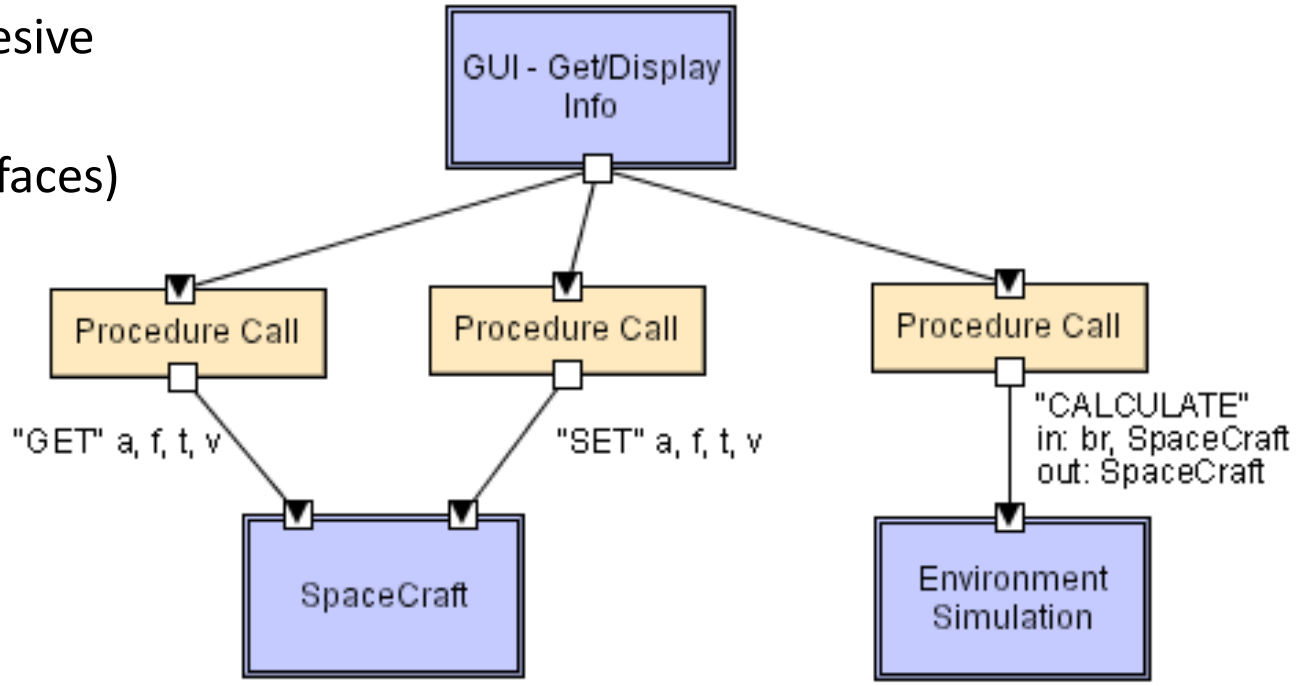


N-Tier Architecture in practice

- Examples:
  - Commercial Web Applications
- Advantages:
  - Shares many advantages similar to Layered Architecture
  - Can be scaled up to support increasing demand
  - Multiple nodes can be allocated to a tier that requires more resources
- Disadvantages:
  - Maintenance of multiple nodes
  - Data communication cost

Object Oriented Architecture

- Views the system as a set of cooperating objects
  - Components are Objects:
    - Objects contain data and behaviors
    - Objects are reusable and cohesive
  - Connectors are messages:
    - Method invocations (via interfaces)
- Based on Key Principles:
  - Abstraction
  - Encapsulation
  - Inheritance
  - Polymorphism

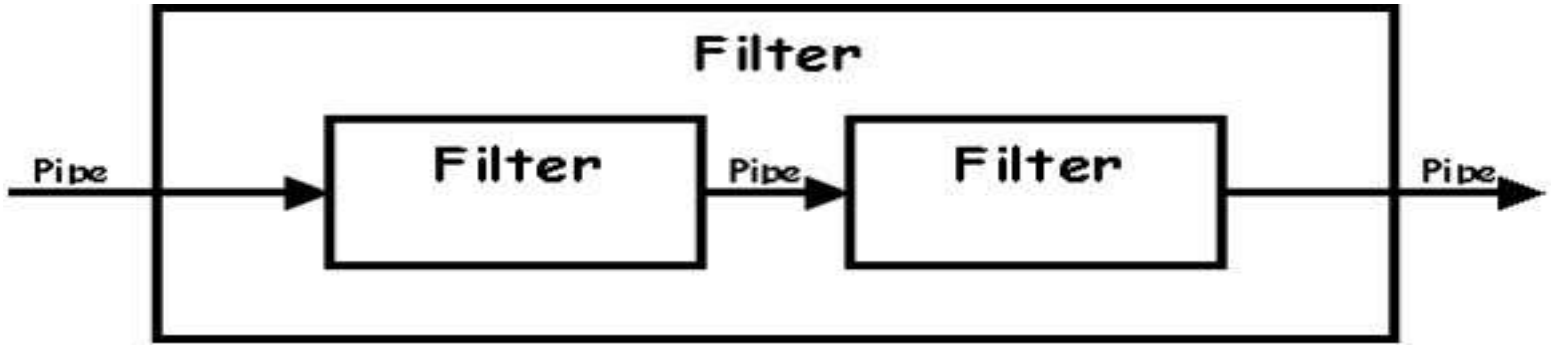


Object Oriented Architecture in practice

- Examples:
  - Most of modern applications
- Advantages:
  - Reusability
  - Extensibility
  - Highly Cohesive
  - Support of many Design/Development tools (e.g. UML)
- Disadvantages:
  - Speed
  - Effort (short term Cost implications)

Pipe and Filter Architecture

- Components are filters
  - Transform input data streams into output data streams
  - Possibly incremental production of output
  - Filters are independent (no shared state)
  - Filter has no knowledge of up- or down-stream filters
- Connectors are pipes
  - Pass data (output) of one filter to another (input)

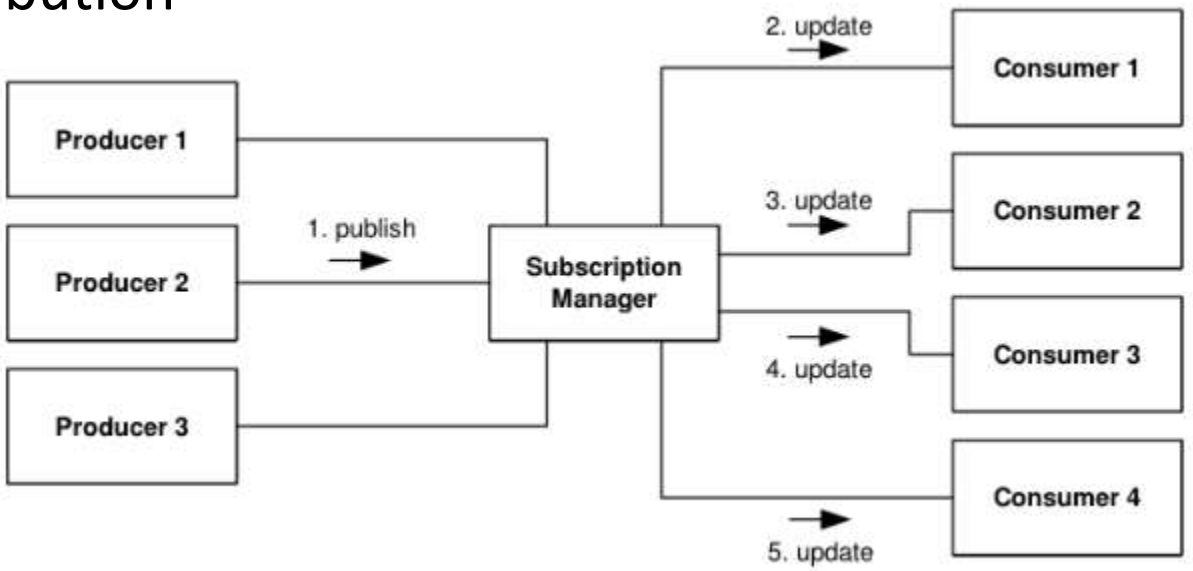


Pipe and Filter Architecture in practice

- Examples:
  - Unix/Linux Shell (Single Processing)
  - Compilers: consecutive filters perform lexical analysis, parsing, semantic analysis, and code generation
- Advantages:
  - Can add/remove filters easily
  - Concurrent Execution: each filter can be implemented as a separate task and be executed in parallel with other filters
- Disadvantages:
  - Performance – may force a lowest common denominator on data transmission
  - No filter cooperation

Publish-Subscribe Architecture

- Subscribers register/deregister to receive specific messages or specific content
- Publishers broadcast messages to subscribers
  - May use Proxies to manage distribution
  - Topic based or Content based

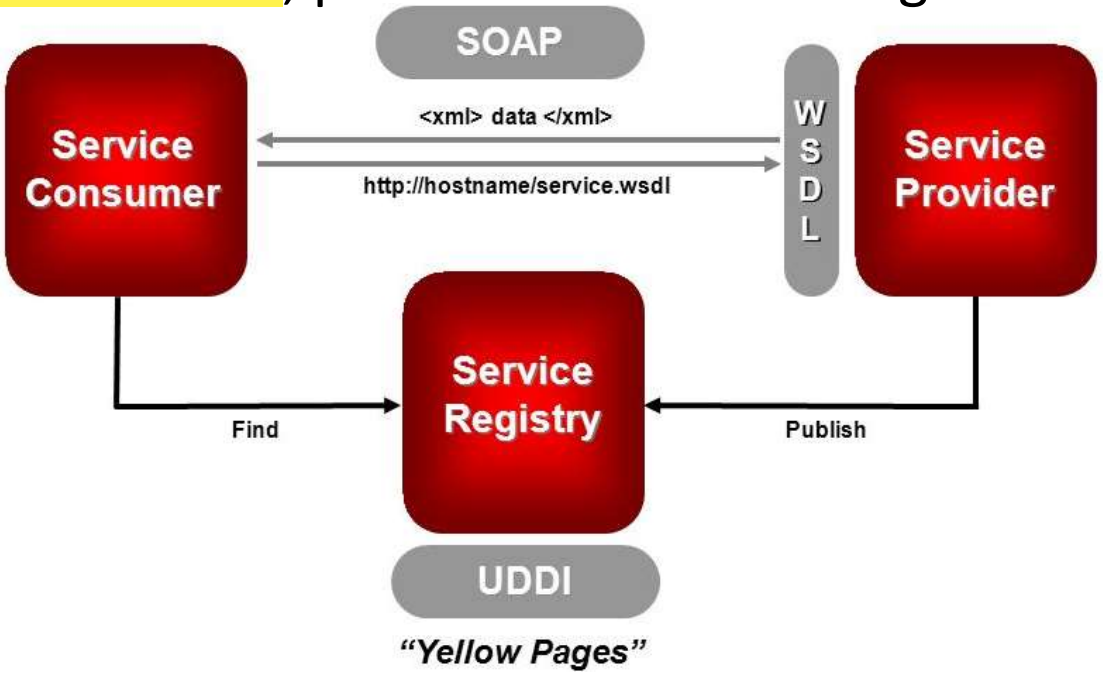


Publish-Subscribe Architecture in practice

- Examples:
  - Mobile News Alerts, Mobile App Push Notifications (e.g. GCM)
- Advantages:
  - Can avoid Polling for new messages (save bandwidth / power)
  - Can use queues / message bus to manage
  - Highly scalable
- Disadvantages:
  - Focus mainly on one-way communication
  - Decoupling of Subscriber from Publisher

Service Oriented Architecture

- Application functionality is provided as a set of remote services
  - Uses standard communication protocols
  - Service and Clients are independent of vendors, products and technologies
- Based on key principles:
  - Autonomous
  - Standard Service Contract
  - Abstracted & Encapsulated
  - Distributable & Discoverable
  - Etc.

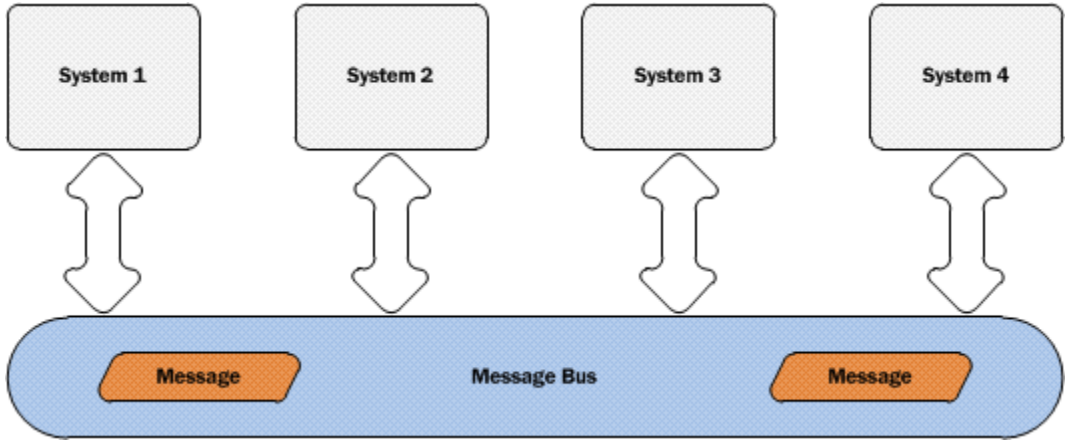


Service Oriented Architecture in practice

- Examples:
  - Many SOAP based Web Services
- Advantages:
  - Interoperability – can integrate products built with different technologies
  - Reusability
  - Widely used – well defined standards & tools
- Disadvantages:
  - Requires high availability

Message Bus Architecture

- Systems communicate with each other by passing messages [Asynchronous] via a common intermediator [Bus]
- Widely used for Enterprise Application Integration (EAI)
  - Many SOA systems use message oriented middleware

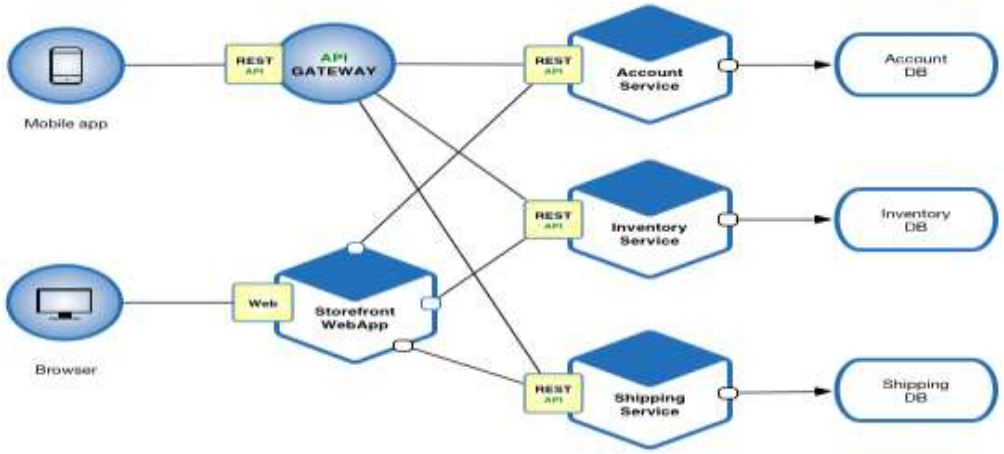


Message Bus Architecture in practice

- Examples:
  - Enterprise Service Bus (JBoss, Mule, WSO2, …)
- Advantages:
  - Extensibility - Can easily add/remove applications from the bus
  - Can integrate with different technologies (via standard communication protocols)
  - Highly Scalable
- Disadvantages:
  - Requires middleware

Microservices Architecture

- Similar to Service Oriented Architecture (SOA)
  - Structures the system as a collection of loosely coupled services
- Decomposes services in to much smaller but more cohesive computation units
- Uses lightweight protocols for communication

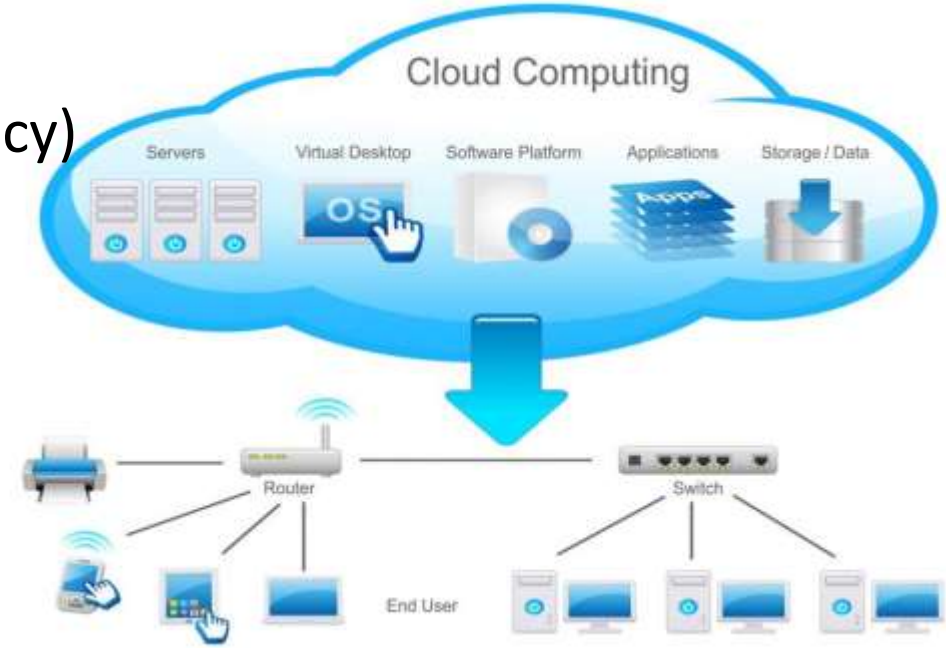


Microservices Architecture in practice

- Examples:
  - Netflix, Twitter, Amazon
- Advantages:
  - Having light weight communication protocols allow thin clients to connect
  - Supports better Continuous Integration & Delivery CI/CD
  - Easy to deploy and scale services independently
- Disadvantages:
  - Maintenance require special [Dev Ops] skills
  - Increase Network Communication within the System

Cloud Architecture

- Enables access to shared pool of resources
  - Can be rapidly provisioned to a new consumer
- Let the business focus on its core business instead of infrastructure
- Basic models of Could Computing:
  - IaaS, PaaS, SaaS (can achieve multi tenancy)



Cloud Architecture in practice

- Examples:
  - Amazon AWS based systems, SalesForce
- Advantages:
  - Elasticity – can scale up and down on-demand
  - Pay as you grow
- Disadvantages:
  - Security Concerns – All information with third parties

Combining Different Architecture Styles

- The overall Architecture of a System is most of often a combination of multiple Architectural Styles
  - E.g. Layered combined with Object-Oriented with a Component-based deployment
- Factors involved:
  - Knowledge/Experience/Capabilities of the Development Team
  - Organizational Constraints (i.e. Data Security Vs. Cloud / SaaS)

References

- Software Architecture Patterns; by Mark Richard
- https://msdn.microsoft.com/en-us/library/ee658117.aspx
- Software Architecture: Foundations, Theory, and Practice; by Taylor & Medvidovic