01 - Introduction

Distributed Systems

Introduction and Key Concepts

❖ ***Topics***

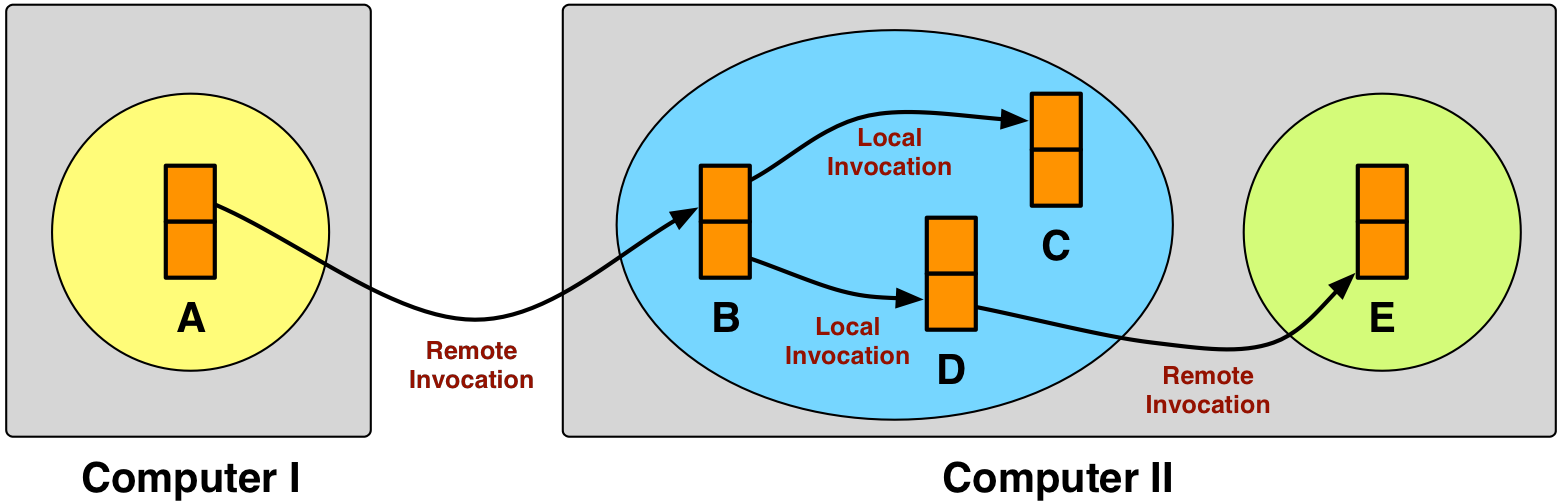
- Definitions
- Key Attributes
- Programming Models
- Architectural Models
- Cloud Models

Distributed Systems

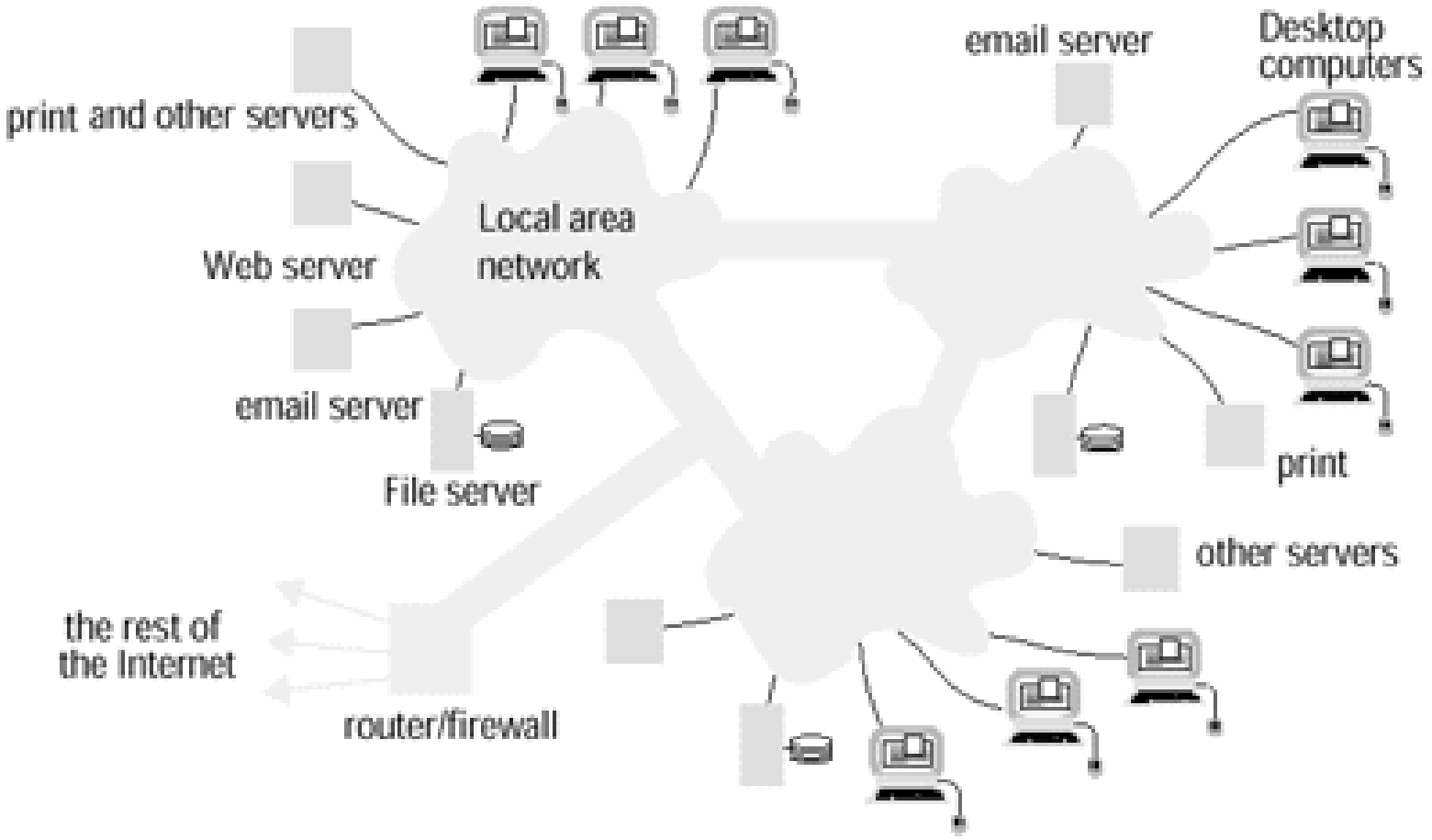
- *Uni-processor, or monolithic computing makes use of a single central processing unit (CPU) to execute one or more programs for each application.*
  - Computer is not connected to any network & only uses resources within it's immediate access.
- A distributed system is a collection of *independent computers* / processing elements, interconnected via a network, that are capable of collaborating on a task.
  - Computers are considered ***independent if they do not share memory or program execution space***.
  - Hardware or software components on network nodes communicate and coordinate actions only by ***passing messages***.
  - Designed to allow resources to be ***shared*** across the network.
  - Users of a well-designed distributed system ***should perceive a single integrated facility*** even though it may be implemented by many computers in different locations.

Distributed Systems

- *Not the same as parallel computing!*
  - Each process in distributed computing has its own private / distributed memory where data for computations is exchanged by passing messages between the distributed processes.
- Different models of passing messages (inter-process communication).



Simple Distributed System



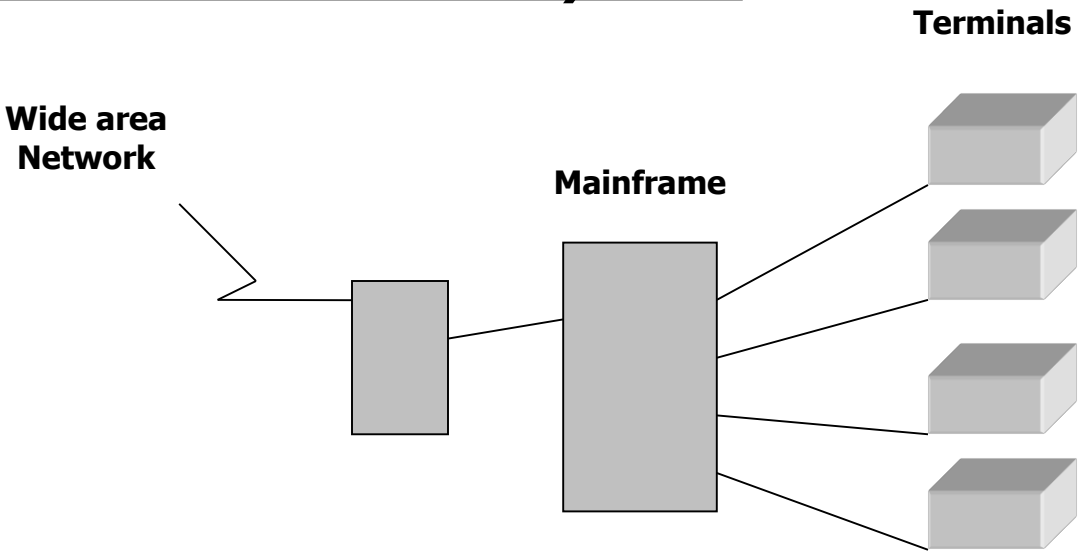
Distributed Systems

- *Computers connected by a network may be spatially separated by any distance.*
- The networked nature of distributed systems has the following consequences:
  1. **Concurrency:** In a network of computers, concurrent program execution is the norm.
  2. **No global clock:** When programs need to cooperate, they coordinate their actions by exchanging messages. Close coordination often depends on the shared idea of the time at which the program's actions occur. However, there are limits to the accuracy with which computers in a network can synchronise their clocks – thus there is no single notion of the correct time.
  3. **Independent failures:** Each component of the system can fail, leaving the others still running. Unexpected termination of a program on one host may or may not be picked up by other hosts on a distributed system.

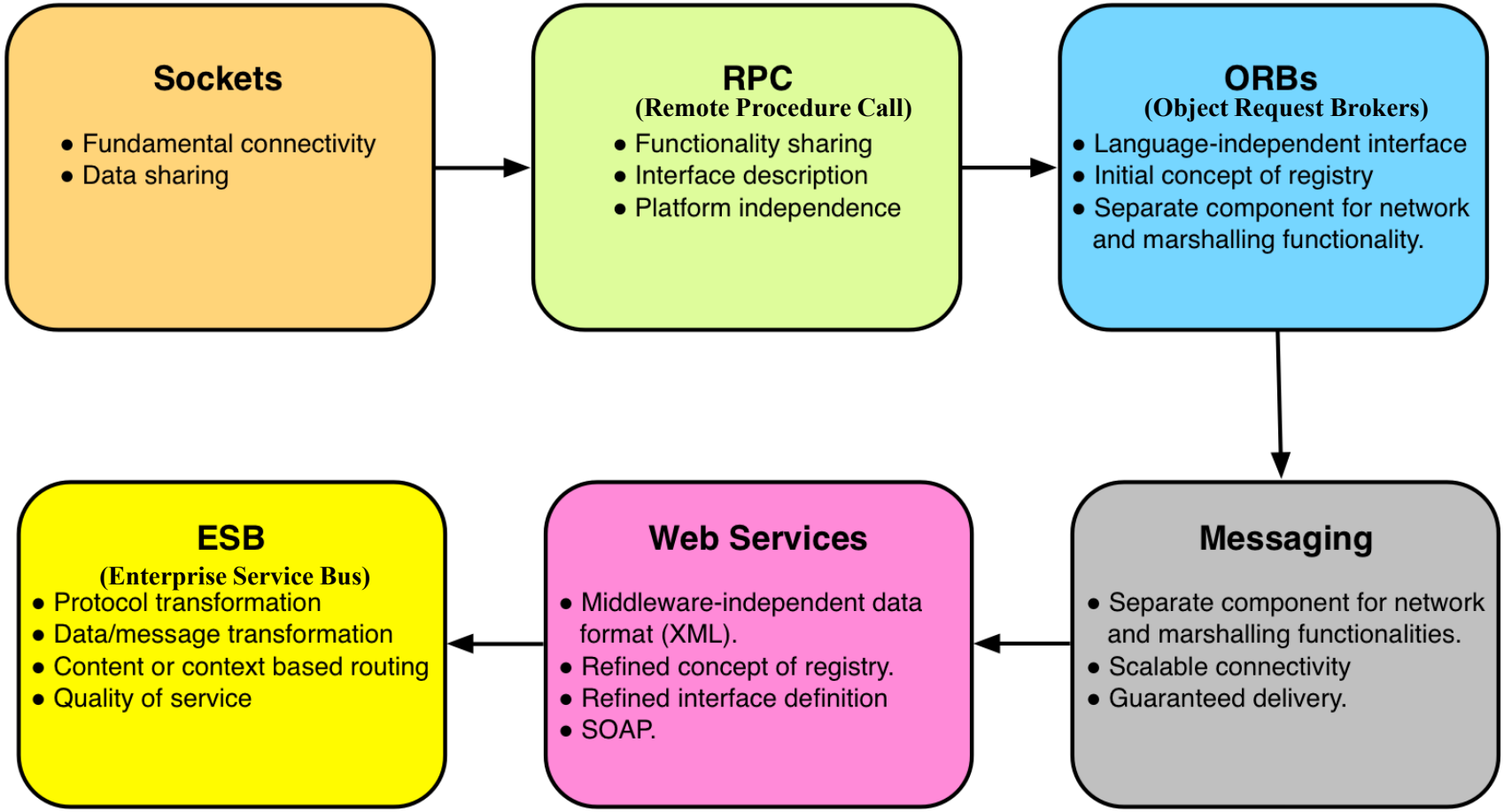
Clients and Servers

- *Servers / services provide access to a variety of shared system facilities and resources, e.g. printing, wide area communication, user authentication.*
  - Programs running on users workstations act as clients of the servers - obtain access to servers resources.
  - Servers may be multiplied as required to support the workload.

A centralized multi-user system:



Evolution of Distributed Systems



Key Attributes - Resource Sharing

- *Resources in a distributed system are physically encapsulated within computers and can only be accesses from other computers by communication.*
  - The pattern of sharing and the spatial distributiion of users determines what mechanisms the system must supply.
  - For effective sharing, each resource must be managed by a program that ***offers a communication interface*** enabling the resource to be accessed and updated reliably and consistently.
  - The term ***service*** is used for a distinct part of a computer system that manages a collection of related resources and presents their functionality to users and applications.
  - The term ***server*** refers to a running program (*process*) on a networked computer (not the actual computer itself) that accepts requests from programs running on other computers to perform a service and respond accordingly.
  - The requesting processes are called ***clients***. The same process may be both a client and server if it makes requests to other servers.

Key Attributes - Heterogenity

- *Heterogenity (variety and difference) applies to the following:*
  - Networks (IP, IPX, DECnet, SNA). IP is standard.
  - Computer Hardware (32 v 64 bit). Big/little endian.
  - Operating Systems (Win v Unix/Linux/OsX, Android, IOS)
  - Programming languages (Assembler, C, C++, Java, C#)
  - Software (Oracle v MySQL v NoSQL, IE v Chrome v Mozilla)
- *Middleware helps solve the problems of heterogeneity.*
  - Term *middleware* applies to ***a software layer that provides a programming abstraction as well as masking the heterogenity*** of the underlying networks, hardware, operating systems and programming languages.
  - Aka ***application server***. Abstracton decouples high level components from low level operations (DiP)
  - Also provides a ***uniform computational model*** for use by the programmers of distributed applications, e.g. J2EE (JBoss, WebSphere…), .NET, CORBA, ODBC. Standards-based.

Key Attributes - Openness

- *The openness of a computer system determines if the system can be extended.*
- System will be open/closed with respect to:
  - Hardware extensions e.g. addition of peripherals, memory or communication interfaces.
  - Software extensions e.g. addition of operating system features, communication protocols.
- Openness achieved by publishing the *key software interfaces* of the components of a system.
  - Open systems are based on the provision of a uniform communication mechanism and published interfaces for access to shared resources. ***Usually developed using open standards***.
- Open distributed systems can be constructed from heterogeneous hardware and software, possibly from different vendors.
  - The conformance of each component to a published standard must be carefully tested and verified if the system is to work correctly.

Key Attributes - Security

- *Security for information resources has three components:*
  1. **Confidentiality** (protection against unauthorised disclosure)
  2. **Integrity** (protection against alteration/corruption)
  3. **Availability** (protection against access interference)
- Information encoded and sent across a network in a secure manner, e.g. SSL, Secure Hash Algorithm (SHA-0...SHA-3).
  - Security also involves identifying the user who is sending the message. Encryption is used to solve this issue.
- Any component of a distributed system archirecture can compromise security. Can use the ***dark web***, e.g. Tor / Onion.
  - **Network**: Even if a message is encrypted, knowledge of recepient and sender can be used to profile. Can also suffer a DoS attack.
  - **Middleware**: Open ports, SQL injections, OS vulnerabilities, leaky firewalls, known (un-patched / updated) bugs.
  - **Database**: User access, fragmentation, person at the next desk…

Key Attributes - Scalability

- *Scalability refers to the ability of a system to remain effective when there is a significant increase in the number of resources and the number of users.*
  - Increasing loads expose limitations with centralised services, data structures and algorithms.
- ***Can scale horizontally or vertically:***
  - ***Horizontal scalability (scale out):*** system can accommodate additional network nodes, e.g. adding new DB servers to a NoSQL cluster. Allows distributed system to grow.
  - ***Vertical scalability (scale up):*** Add additional resources to a node, e.g. memory, processors. Allows increased run-time performance.
- Each component of system architecture contributes to scalability.
  - Communication latencies, i.e. synchronous v asynchronous services.
  - Middleware clustering / visualisation / elastic cloud.
  - Database replication, mirroring, schema-less, CAP Theorem.

Key Attributes - Fault Tolerance

- *Fault tolerance is a measure of how the system is able to cope with hardware or software faults.*
  - Failures in a distributed system are partial. The system will continue to operate (at a reduced level or performance / service).
  - ***Availability*** of a system is a measure of the proportion of time that it is available for use (up-time).
- Not just handled at a programming level!
  - Fault tolerance provided by fail-over support of middleware / clustering or virtualisation.
- *Fault tolerance relates to each component of an architecture.*
  - Messaging (TCP), method invocation (exceptions), time-outs on remote nodes, database / rollbacks / storage errors.
- Difficult to determine if a remote host is still working without constant polling / communication overheads.
  - Challenge is to manage in the presence of failures that cannot be detected but are suspected and to mask failures when they arise.

Key Attributes - Concurrency

- *In a distributed system, multiple simultaneous access to shared resources is assumed.*
  - Implies concurrent access to these shared resources, i.e. the system should be threaded to support concurrency.
  - ***Has a major impact on both performance / scalability***.
- Applies to all components of the distributed system.
  - Mostly (but not always) implemented and managed by middleware, not by the programmer.
- System is only as fast as its worst algorithm / bottleneck!
  - Balanced and extensible system architecture. Bottlenecks can occur at any poorly designed / implemented component.
  - Balance user / work load with the resources available.
  - Fine tune number of processes (Apache), indices / locks (Oracle), synchronisation of methods or code blocks.
  - Asynchronous communication to queue user requests.

Key Attributes - Transparency

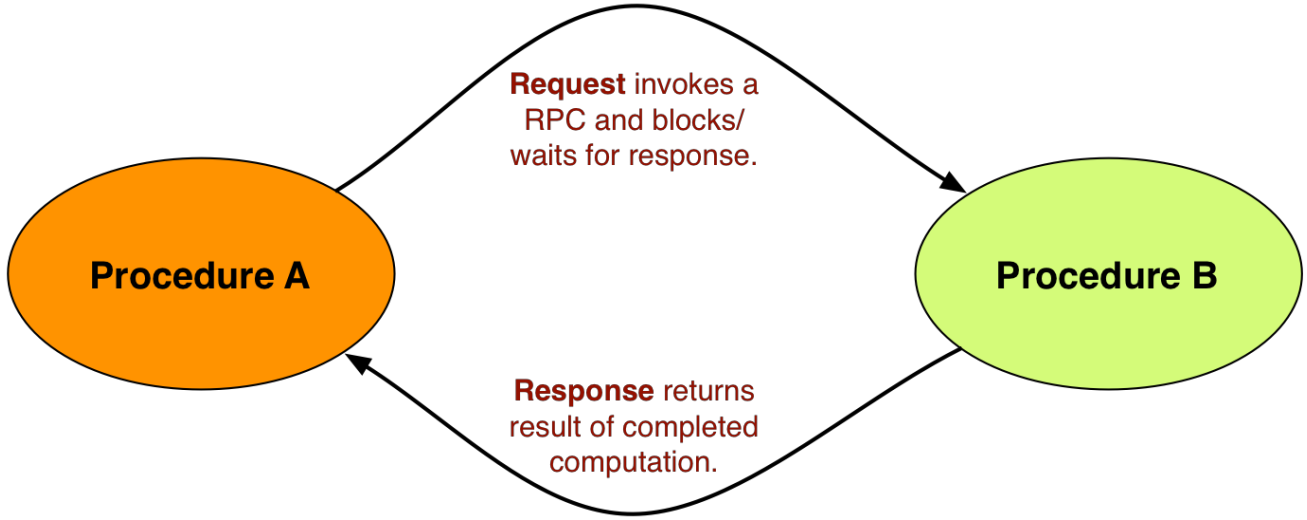
- *The concealment from the user and the application programmer of the separation of components in a distributed system so that the system is perceived as a whole rather than as a collection of independent components.*
  - The implications of transparency are a major influence on the design of the system software.
- There are a number of different forms of transparency:
  - **Access transparency**: local and remote resources accessed using identical operations, e.g. file manager, APIs.
  - **Location transpency**: resources located without knowledge of their location, e.g. URLs refer to a computer domain rather than an IP address.
  - **Concurrency transparency**: several users or application programs can concurrently use shared data without mutual interference, e.g. threading, locking.

Key Attributes - Transparency

- **Replication transparency**: multiple copies of resources can be used to increase reliability and performance without users or applications being aware of the replicas, e.g. load balancing.
- **Failure transparency**: concealment of faults, allowing users and applications to complete their work despite the presence of faults, e.g. email retransmission.
- **Migration transparency**: allows the movement of resources within a system (load share) without affecting operation of users or applications, e.g. mobile phones, mobile IP?? (maybe some day).
- ***Performance transparency:*** allows the system to be reconfigured to improve performances as loads vary.
- **Scaling transparency**: allows the system to grow without changing the overall system structure.

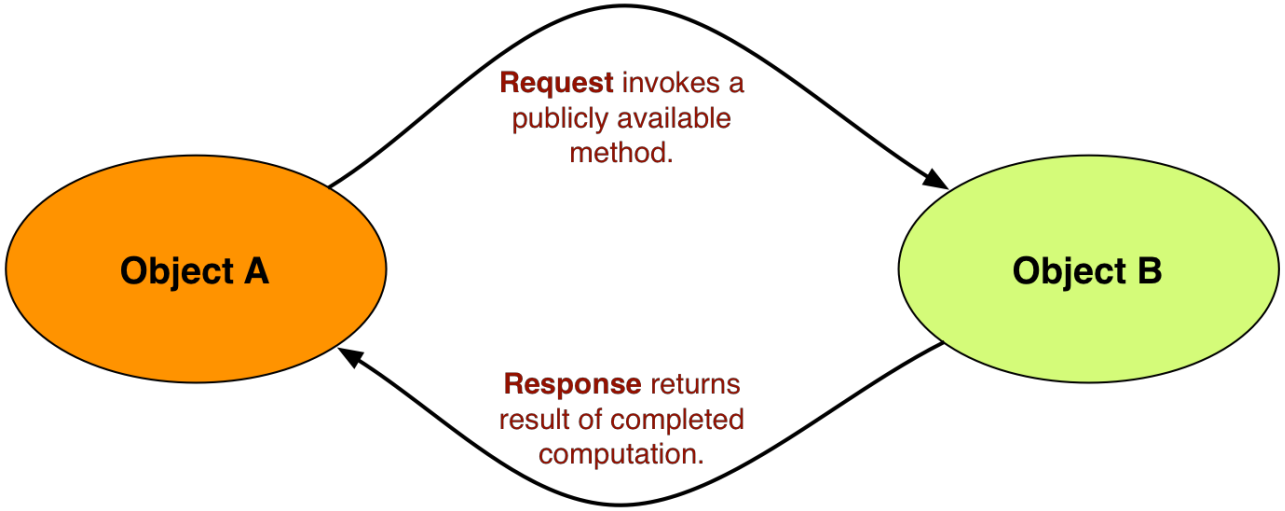
Remote Procedure Call Model

- *The RPC programming model is a first generation approach to sharing computing services between distributed applications.*
  - Model relies on a request/reply communication protocol, resulting in a fully-coupled communication between applications.
  - Underlying RPC support varies with the programming language & therefore not amenable to ***heterogeneous*** computing.



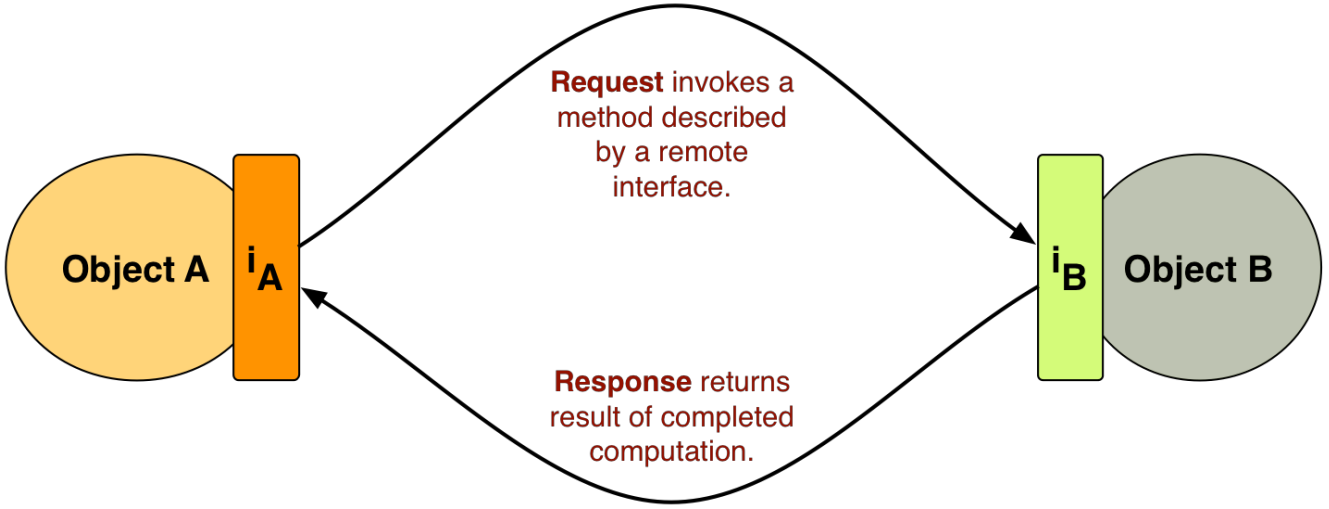
Object-Oriented Model

- *OOP model emerged as a result of the birth of OO programming languages.*
  - Shares a lot of commonality with RPC model, e.g. still uses the request-reply communication protocol & frequently results in a fully-coupled communication between applications.
  - Methods are grouped together using class definitions & not all methods in a given class are remotely accessible.



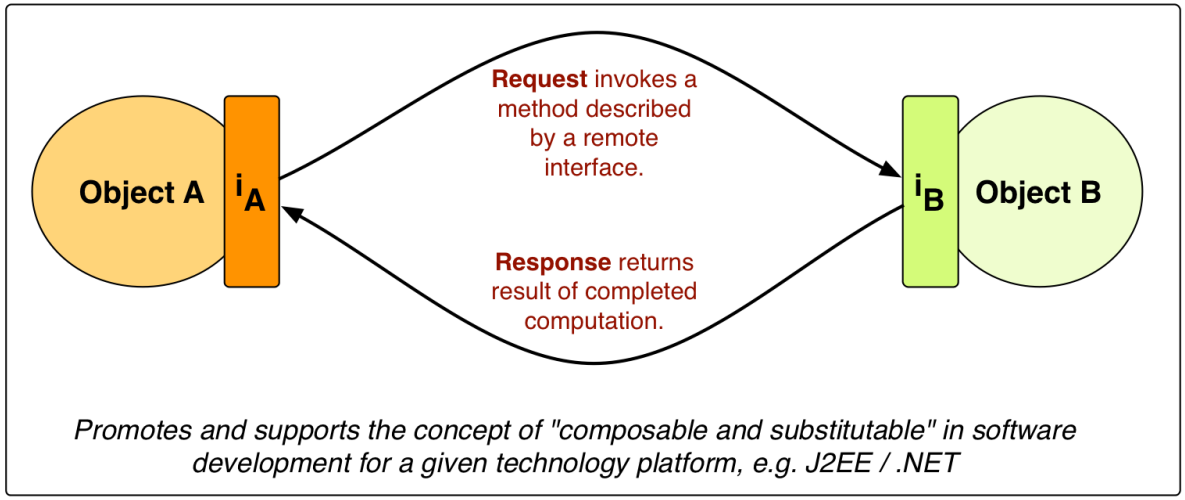
Interface-based Model

- *Interfaces are abstraction layers in programming.*
  - Interface model takes advantage of the emergence of abstraction & corresponding programming technologies to improve the interoperability, robustness & maintainability of enterprise applications.
  - Change from OOP to interface-based programming is not substantial but the addition of an abstraction layer improves interoperability by encapsulating the implementation details.



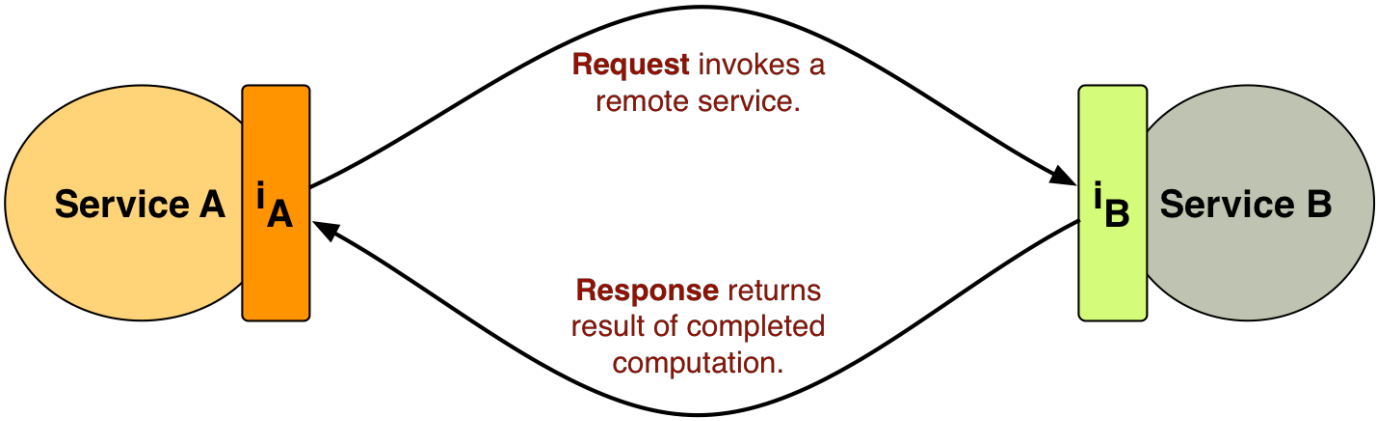
Component-based Model

- *Develop software applications by assembling software modules, often built by other providers, when a preset specification is commonly compiled by such providers (Object Nirvana).*
  - Greatly improves reliability within enterprise systems.
  - Components that become instable can be swapped with another.
  - Enables reusability of components internally within an enterprise where multiple internally delivered applications require the same type of component level functionality.



Service-based Model

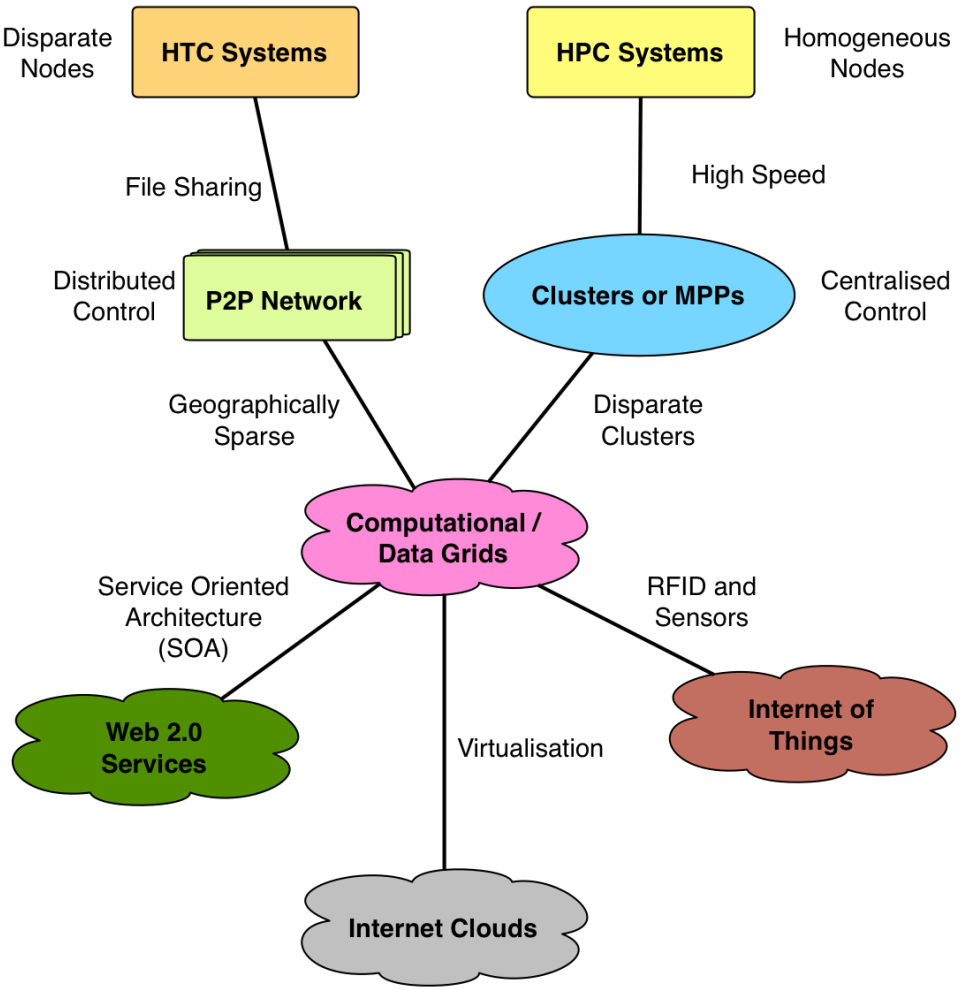
- *Aka Service-Oriented and allows enabled services to be programming-language, executing platform & integration / middleware independent.*
  - These characteristics are key to facilitating the interconnection of devices in a heterogeneous network.
  - A service is essentially encapsulated to represent a unit of business work.



Focuses on the delivery of platform, language and middleware-independent (information) services.

Architectural Evolution

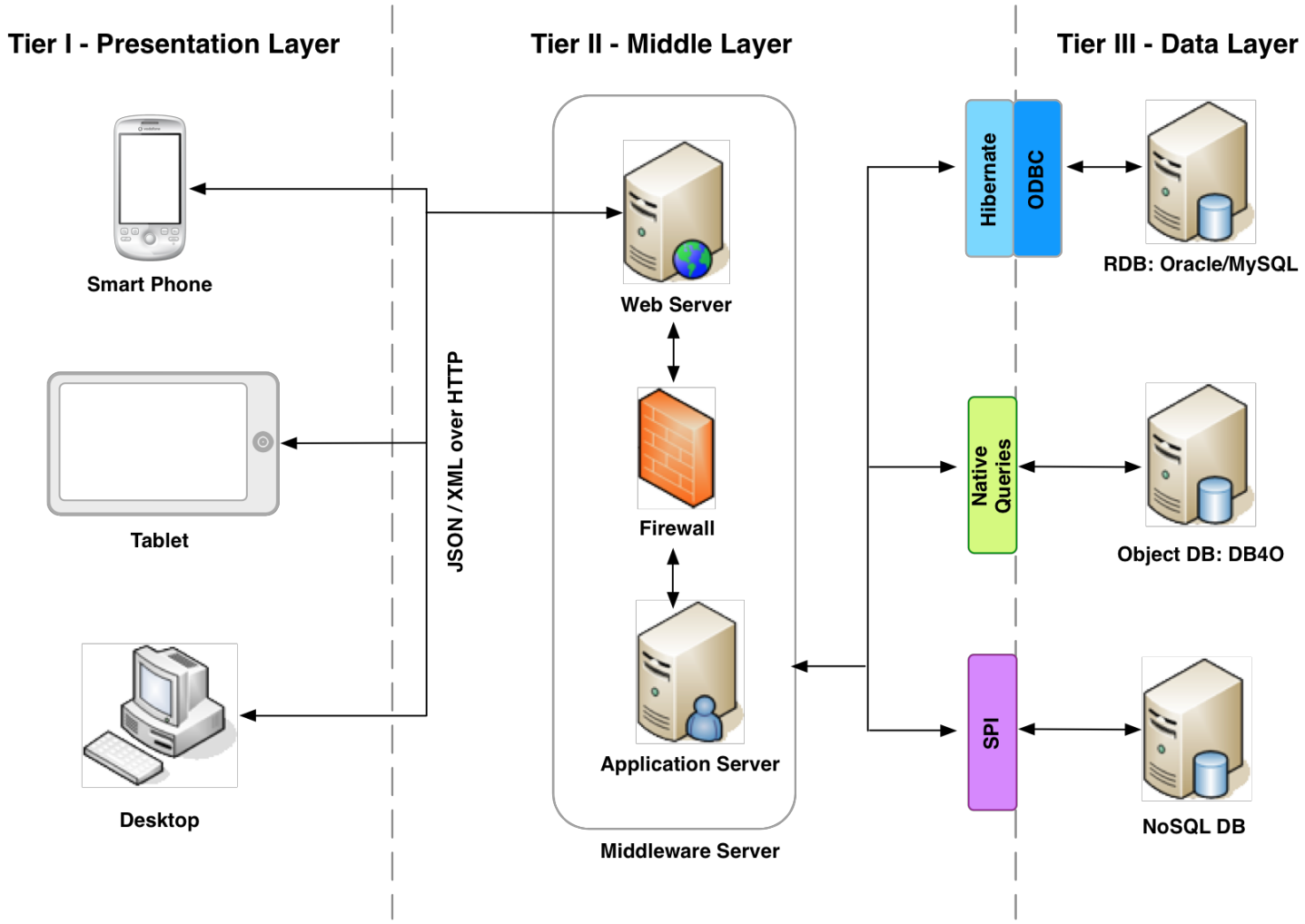
- High Throughput Computing (HTC) and High Performance Computing (HPC) evolve into P2P networks or clusters.
- These evolve into grids and then into services, clouds and part of the IoT.



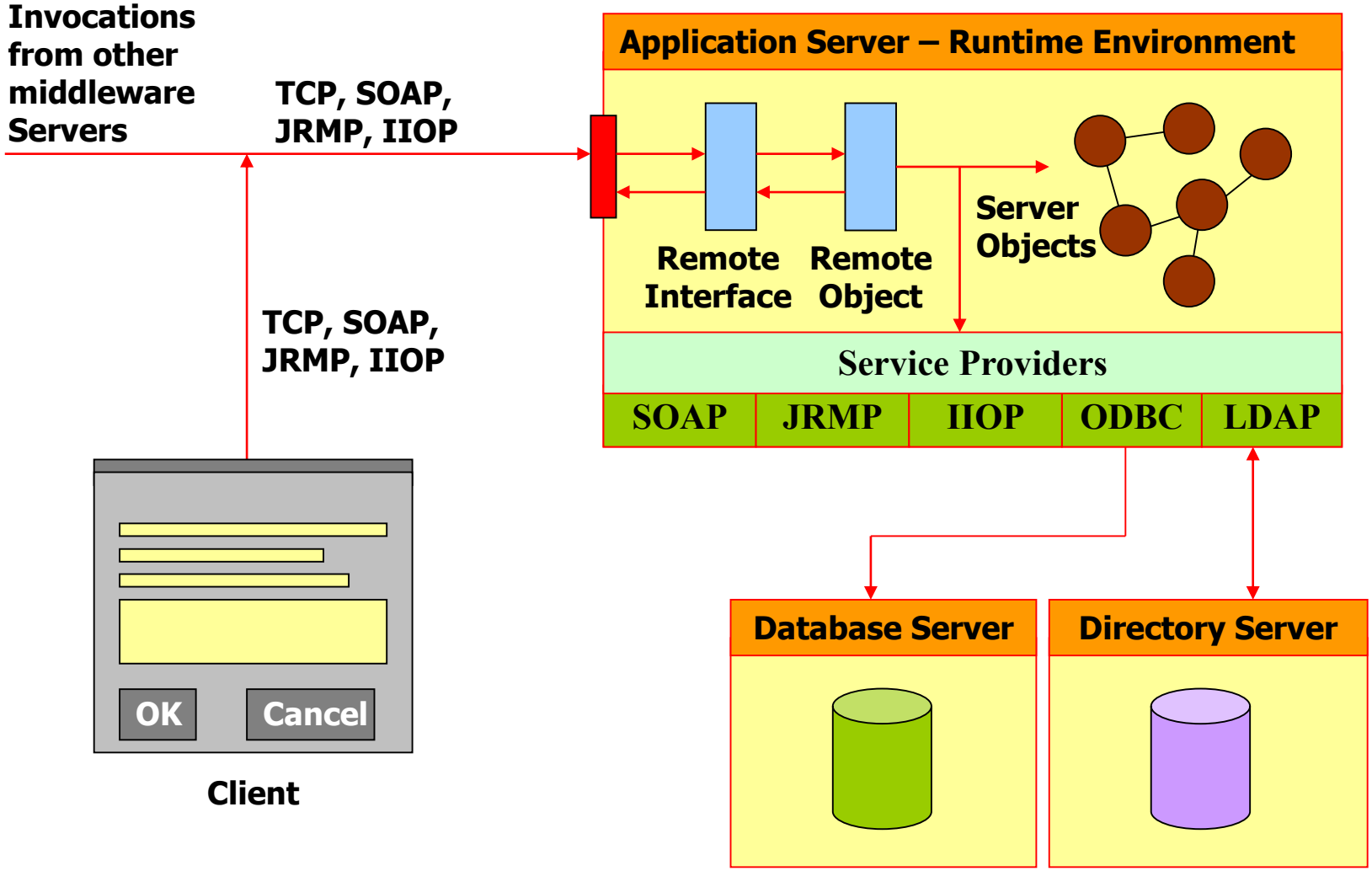
Architectural Models

- *Trend is to leverage shared web resources and massive amounts of data over the Internet.*
- ***High Performance Computing (HPC)***
  - Supercomputers (massively parallel processors / *MPPs*) are *gradually replaced by clusters* of cooperative computers out of a desire to share computing resources.
  - Cluster is often a collection of homogeneous compute nodes that are physically connected in close range to one another.
- ***High Throughput Computing (HTC)***
  - Peer-to-peer (P2P) networks are formed for distributed file sharing and content delivery applications. Built over many client machines and are *globally distributed in nature*.
  - P2P, cloud computing, and web service platforms are more focused on HTC applications than on HPC applications.
  - Clustering and P2P technologies lead to the development of computational grids or data grids.

Basic n-Tier Client/Server Model



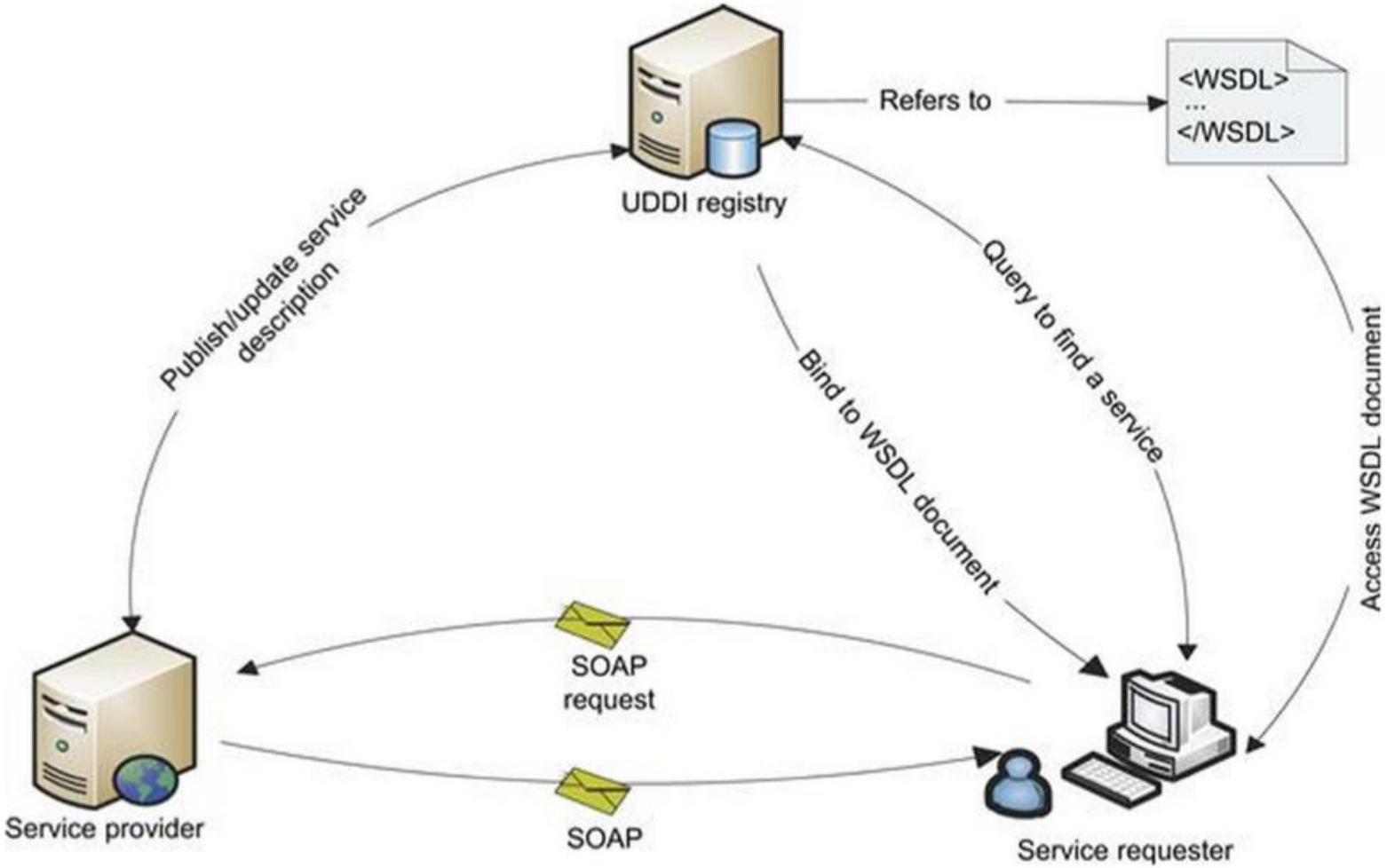
Generic IPC Model



Web 2.0 Services

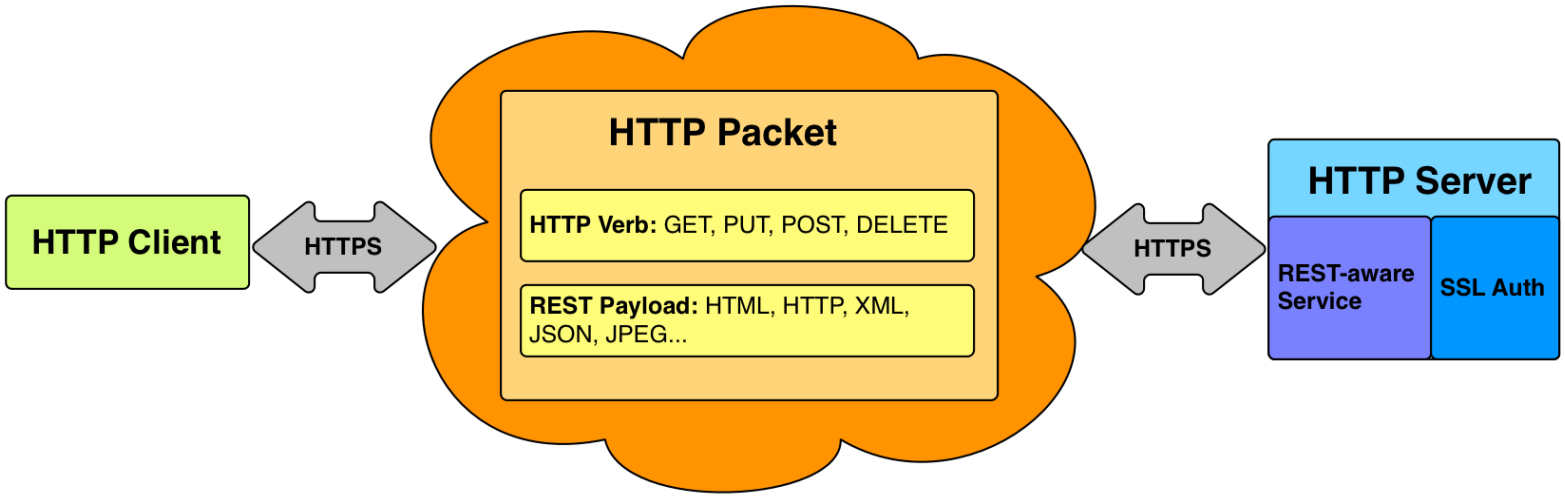
- *In a service oriented architecture (SOA), software capabilities are delivered and consumed via loosely coupled, reusable, coarse-grained, discoverable, and self-contained services interacting via a message-based communication model.*
  - A ***web service*** is often referred to as a self-contained, self-describing, modular application designed to be used and accessible by other software applications across the web.
    - A software system designed to support interoperable machine-to-machine interaction over a network.
  - Once a web service is deployed, other applications and other web services can discover and invoke the deployed service.
- *Described by an interface in a machine-executable format (WSDL).*
  - Other systems interact with the web service in a manner prescribed by its description using SOAP messages.
  - Messages typically dispatched over HTTP using XML serialization in conjunction with other web-related standards.

Web 2.0 Services



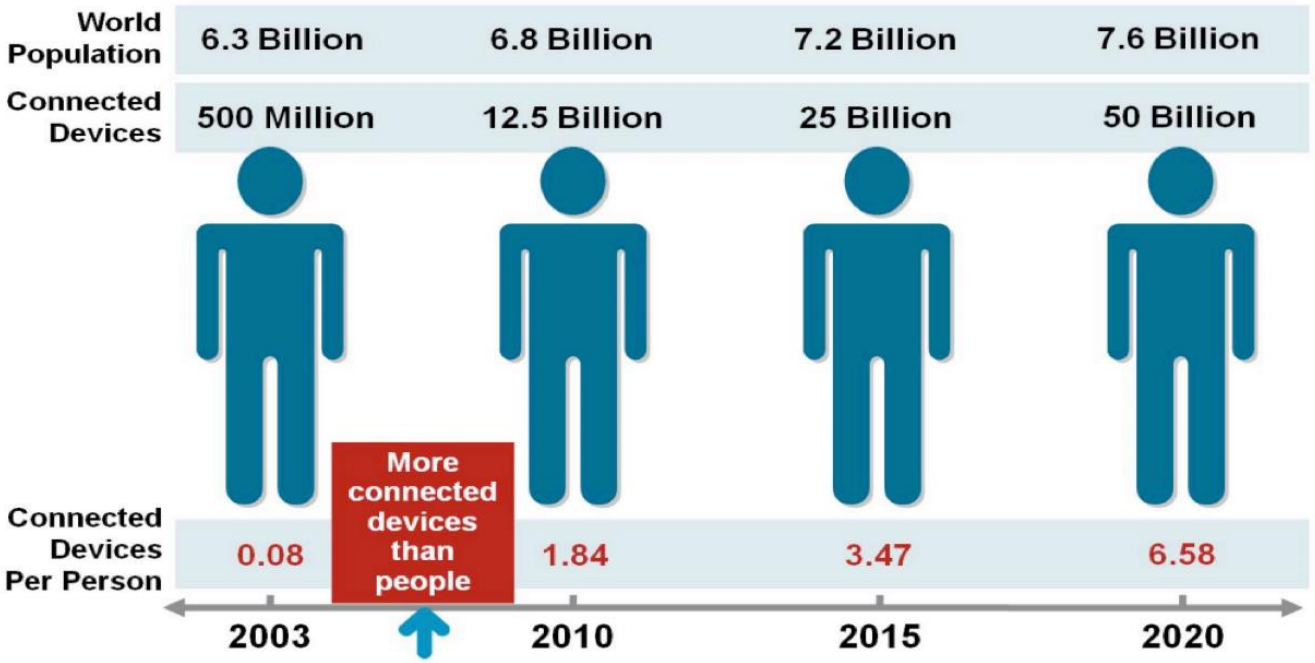
Web 2.0 RESTful Services

- *Common approach is to use a RESTful architecture.*
  - Use HTTP/S over TCP/IP for communication.
  - Supports multiple MIME types, e.g. text, XML, JSON etc..
  - Easy to implement and deploy.
    - Annotate RESTful methods to describe service interface.



Internet of Things (IoT)

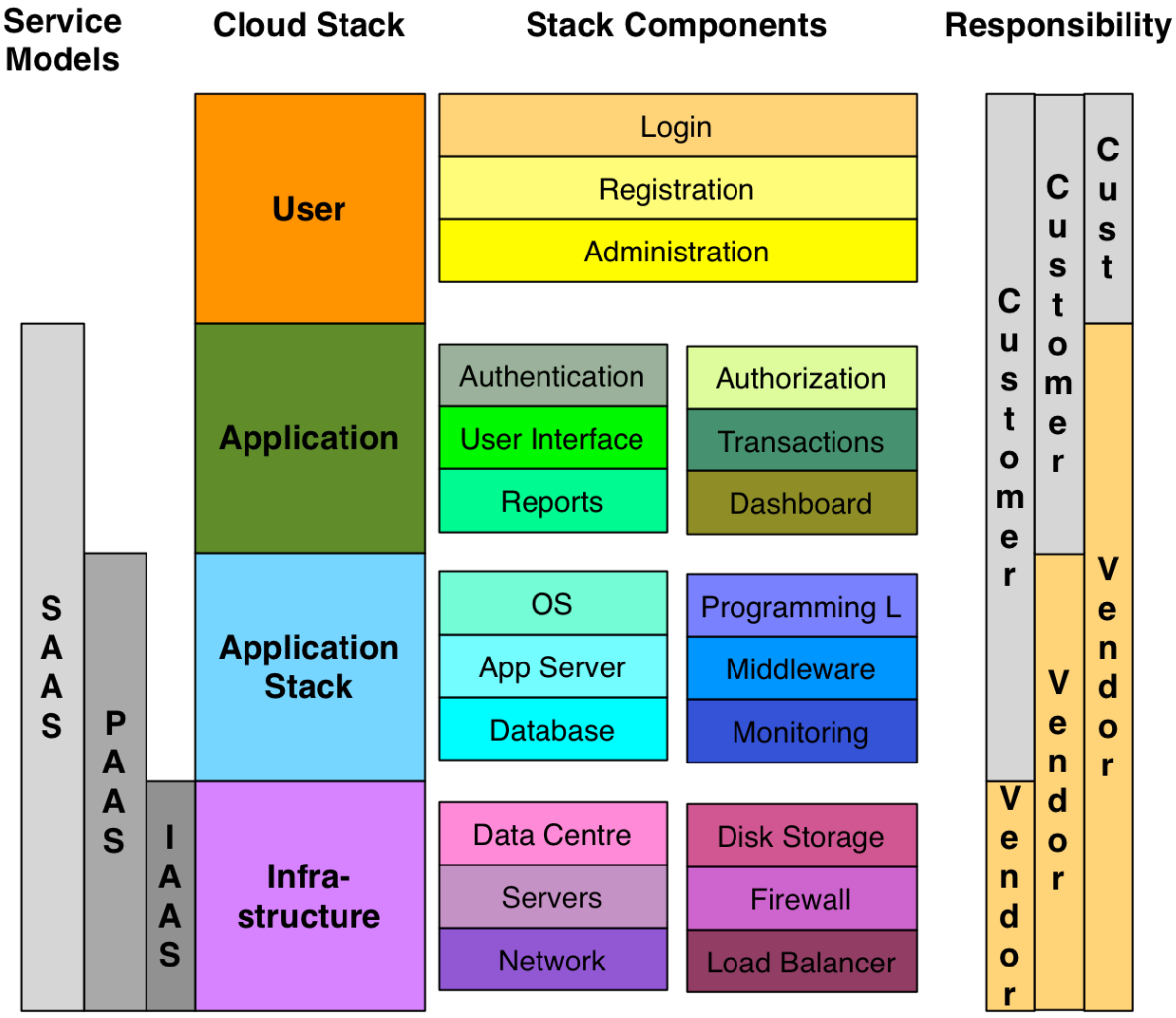
- *Internet of Things (IoT): the point in time when more "things or objects" were connected to the Internet than people.*
  - Reached this point circa 2007. By 2020 there will be 50 x 10^9 devices connected to the Internet.
  - Major implication / opportunities for future distributed systems.



Cloud Service Models

- *A cloud is presented to a user as a computing environment, accessed via the Internet, with apparent infinite h/ware resources.*
  - Large data centers owned by specific companies who make them available through pay-as-you-go schemes described above.
  - Most use virtualisation for both access provision and flexible hardware configurations.
- Popular public clouds include Google's AppEngine, Amazon's AWS, IBM's Blue cloud & Microsoft's Azure.
  - Each provider organises its cloud in a different way but utilises a basic layered architecture.
- ***Three main cloud service models:***
  - ***Software as a Service*** (SaaS), ***Platform as a service*** (PaaS), and ***Infrastructure as a Service*** (IaaS).
  - Each cloud model provides a level of abstraction that reduces the efforts required by the service consumer to build & deploy systems.

Cloud Service Stack



Infrastructure as a Service (IaaS)

- *The capability to provide and provision processing, storage, networks, and other fundamental computing resources where the consumer is able to deploy / run arbitrary software.*
  - Arbitrary software includes OSs & applications.
  - Consumer doesn't manage /control underlying cloud infrastructure.
  - Has control over OS, storage, deployed applications and limited control of select networking components e.g. host firewalls.
- Several IaaS vendors in the marketplace.
  - Amazon (AWS) and Rackspace most mature & widely used.
  - OpenStack is another key player.
    - Open source project that provides IaaS capabilities for consumers who want to avoid vendor lock-in & want the control to build their own IaaS capabilities in house, referred to as private cloud.

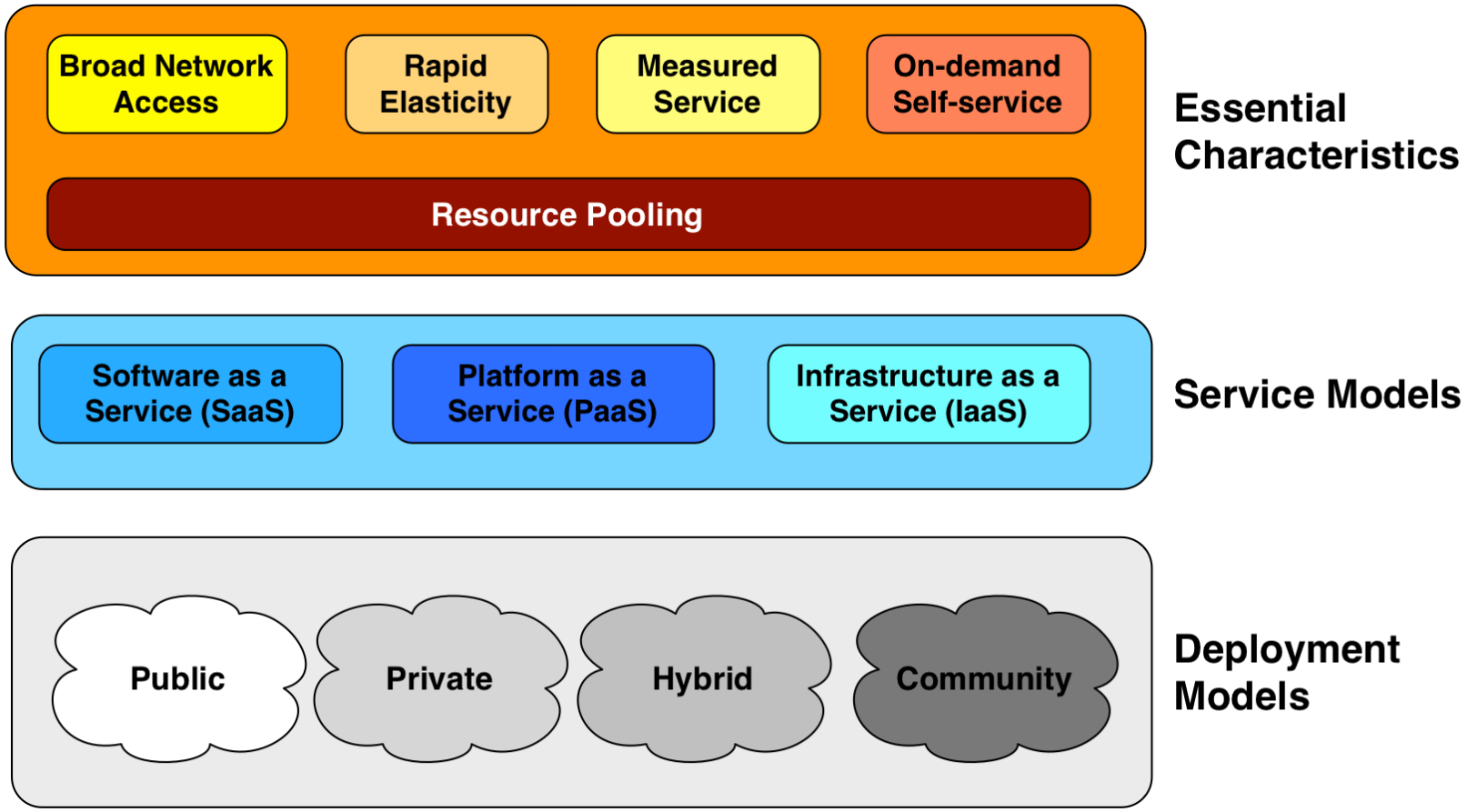
Platform as a Service (PaaS)

- *The delivery of a computing platform and solution stack as a service.*
  - PaaS offerings facilitate deployment of applications without the cost and complexity of buying / managing the underlying hardware / software and provisioning hosting capabilities.
  - PaaS sits on top of IaaS and abstracts much of the application stack-level functions and provides those functions as a service.

Software as a Service (SaaS)

- *Top of the stack. A complete application delivered as a service to the service consumer.*
  - ***Service consumer*** has only to configure some application-specific parameters and manage users.
  - ***Service provider*** handles all infrastructure, all application logic, all deployments and everything pertaining to the delivery of the product or services.
- Implications for next generation of software developers (you!).
  - Your role and responsibility for the delivery of such services and guaranteeing up-time.
- Companies use SaaS solutions for non-core-competency functions obviating the need to support and provide infrastructure and maintenance.
  - ***Pay a subscription fee*** and simply use the service over the Internet as a browser-based service. Constant revenue stream!

Cloud Deployment Models



Cloud Deployment Models

- *Four key models for cloud computing:*
  - ***Public Cloud:*** infrastructure provisioned for use by the general public. Located on premises of the cloud provider.
    - May be owned, managed and operated by a business, academic, or government organization or combination.
  - ***Private Cloud:*** infrastructure provisioned for exclusive use by a single organization comprising multiple consumers (e.g. business units). May exist off premises.
    - May be owned, managed and operated by the organization, a 3rd party or combination.
  - ***Hybrid and Community Cloud:*** Composition of two or more distinct cloud infrastructures that remain unique entities, but are bound together by standardized or proprietary technology that enables data and application portability.
    - e.g., cloud bursting for load balancing between clouds.