

ESCOLA D'ENGINYERIA

Parallel and Distributed Computing (calculation) Systems

Introduction to distributed Systems &
Architectures.

R. Suppi (Remo.Suppi@uab.cat)

Syllabus

Course Content: distributed computing systems (hands on course). Topics include introduction to DS (architectures, communication, process synchronization, consistency, replication, fault-tolerance, security, distributed file systems), distributed web-based systems, cloud computing, mobile computing, big data & distributed computing.

Prerequisite: Computer Science skills (Hw/Sw, networking, programming)

Course Requirements:

All students should not only learn basic theoretical principles but also accumulate practice experience. Each student will do some lab assignments (group, personal) and finally present a report including the results and conclusion of these assignments & the code.

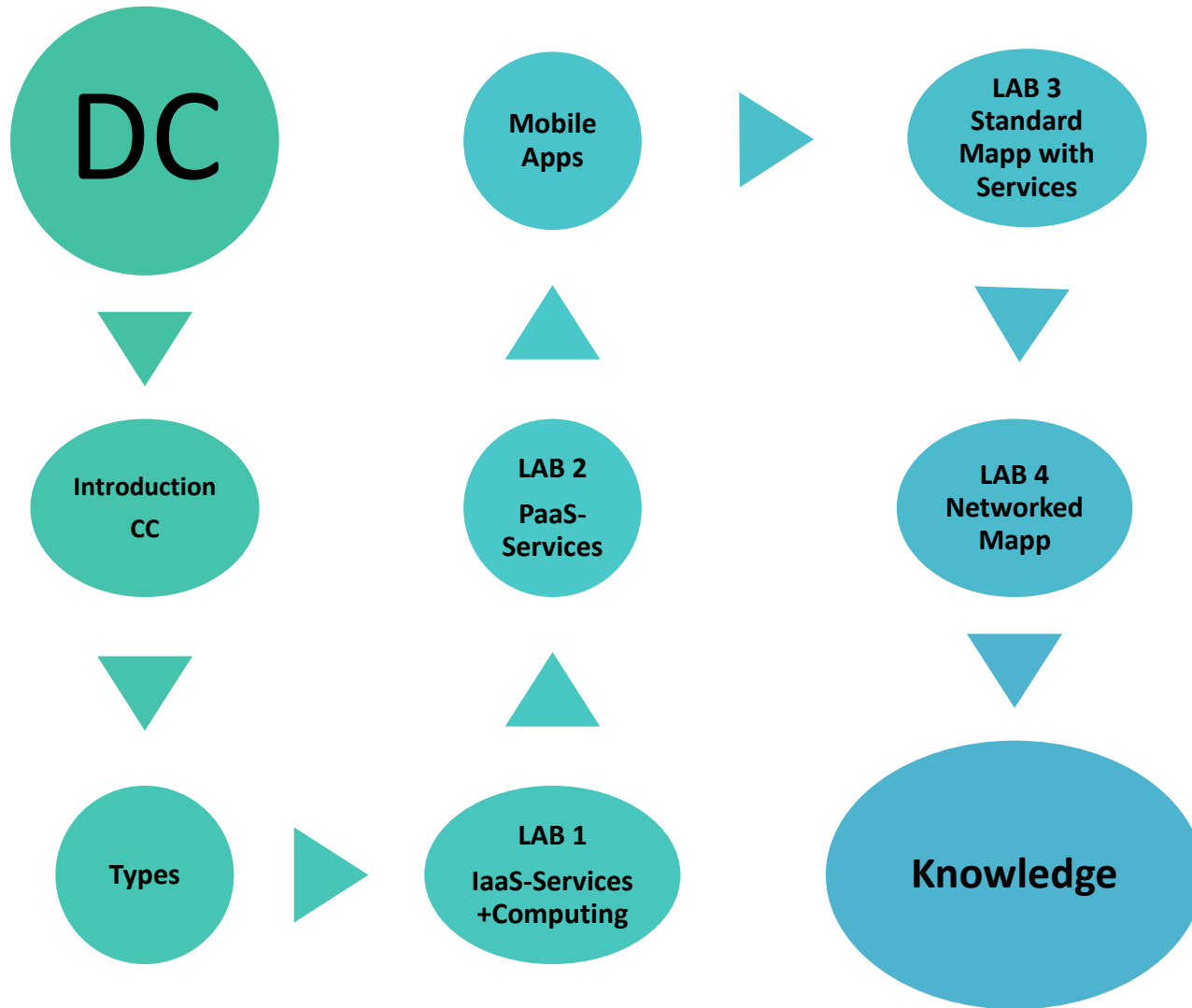
Class Policy:

Attendance: Students are required to attend all classes. Role will be taken.

Web Site: class information, documents and all info will be published in the <http://cv.uab.cat>.

Academic honesty: Plagiarism will result in a score of zero on any work, assignment or code. The instructor has the right to evaluate if students are cheating and make a decision.

Conceptual map



Definicions de un sistema d'informació (ordenador, dades, processament, ...)

Una mica d'història

Evolució i dependència

- Introduction and concepts DCP

- Data Intensive Applications

Informació i dades

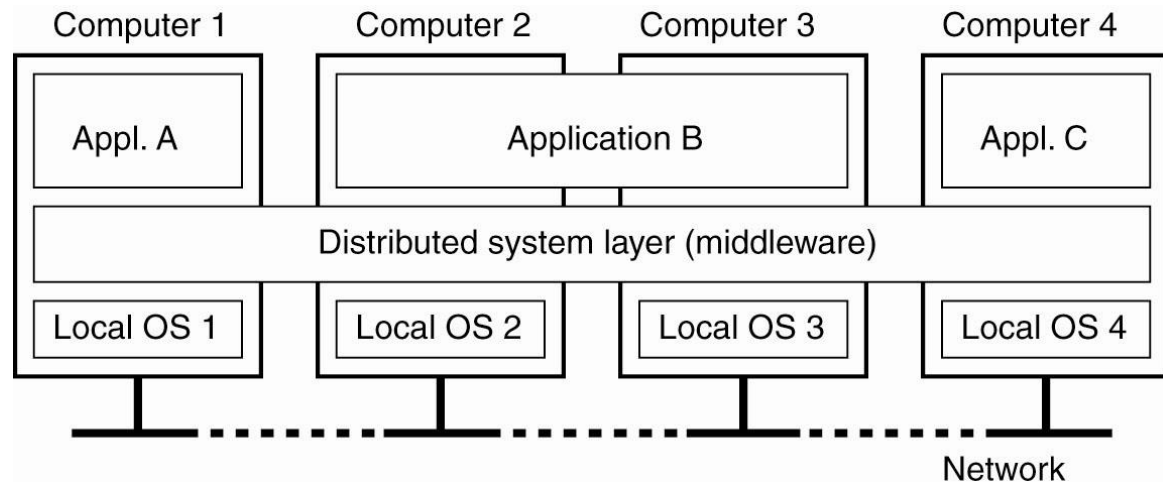
- Cluster computing

- Distributed computing environments: Apache Hadoop and Spark

What is a distributed system?

Concept

1st. A collection of independent computers that appears to its users as a single coherent system.



A distributed system can be organized as middleware. The middleware layer extends over multiple machines, and offers each application the same interface.

Why is necessary?

Transparency?

Access	Hide differences in data representation and how a resource is accessed
Location	Hide where a resource is located
Migration	Hide that a resource may move to another location
Relocation	Hide that a resource may be moved to another location while in use
Replication	Hide that a resource is replicated
Concurrency	Hide that a resource may be shared by several competitive users
Failure	Hide the failure and recovery of a resource

Scalability?

Concept	Example
Centralized services	A single server for all users
Centralized data	A single on-line telephone book
Centralized algorithms	Doing routing based on complete information

Problems?

- Unpredictable failures of components: lack of response may be due to either failure of a network component, network path being down, or a computer crash (failure-prone, unreliable)
- Large numbers of hosts: 2 to several million (One Adm?)
- No machine has complete information about the system state.
- There is no implicit assumption that a global clock exists
- The network (reliable, secure, homogeneous, latency = 0, Bw = ∞ , transport cost = 0).
- The topology (does not change).

Common Goals?

- **Heterogeneity:** can the system handle a large variety of types of PCs and devices?
- **Robustness:** is the system resilient to host crashes and failures, and to the network dropping messages?
- **Availability:** are data+services always there for clients?
- **Transparency:** can the system hide its internal working from the users?
- **Concurrency:** can the server handle multiple clients simultaneously?
- **Efficiency:** is the service fast enough? Does it utilize 100% of all resources?
- **Scalability:** can it handle 100 million nodes without degrading service?(nodes=clients and/or servers) How about 6 B? More?
- **Security:** can the system withstand hacker attacks?
- **Openness:** is the system extensible?

Attributes?

The degree to which a system is distributed can be determined by answering four questions:

Where is the processing done?

How are the processors and other devices interconnected?

Where is the information stored?

What rules or standards are used?

Attributes?

Where is the processing done?

How are the processors and other devices interconnected?

Where is the information stored?

What rules or standards are used?

Distributed Processing is the ability for more than one interconnected processor to be operating at the same time, typically for processing an application on more than one computer at a time

The goal is move the appropriate processing as close to the user as possible and to let other machines handle the work they do best

Permits interoperability-capability of different computers using different OS on different networks to work together on tasks

Two forms of interoperability:

- Communication between systems
- Two-way flow between user applications

Attributes?

Where is the processing done?

How are the processors and other devices interconnected?

Where is the information stored?

What rules or standards are used?

Connectivity Among Processors means that each processor in a distributed system can send data and messages to any other processor through electronic communication

Desirable to have at least two independent paths between two nodes to provide automatic alternate routing (Planned Redundancy)

Distributed Databases either:

Divide a database and distribute its portions throughout a system without duplicating the data

Users do not need to know where a piece of data is located to access it, because the system knows where all the data is stored

Store the same data at several different locations, with one site containing the master file

Synchronization of data is a significant problem

Where is the processing done?

How are the processors and other devices interconnected?

Where is the information stored?

What rules or standards are used?

Attributes?

Where is the processing done?

How are the processors and other devices interconnected?

Where is the information stored?

What rules or standards are used?

System wide Rules mean that an operating discipline for the distributed system has been developed and is enforced at all times

These rules govern communication between nodes, security, data accessibility, program and file transfers, and common operating procedures

Since the 1990s, Open systems concept-mix products from vendors using open standards. Based on open-systems - standardized interfaces that allow products to inter-operate across multi-vendor networks, operating systems and databases

Now API define the way to present data to another system component. Makes writing distributed systems much easier

Components

Processing

Single user, stand-alone and connected to LAN; clients

Multiple user, serve local groups of users; server. Also heavy duty computation for single users, backups, program libraries, and database management.

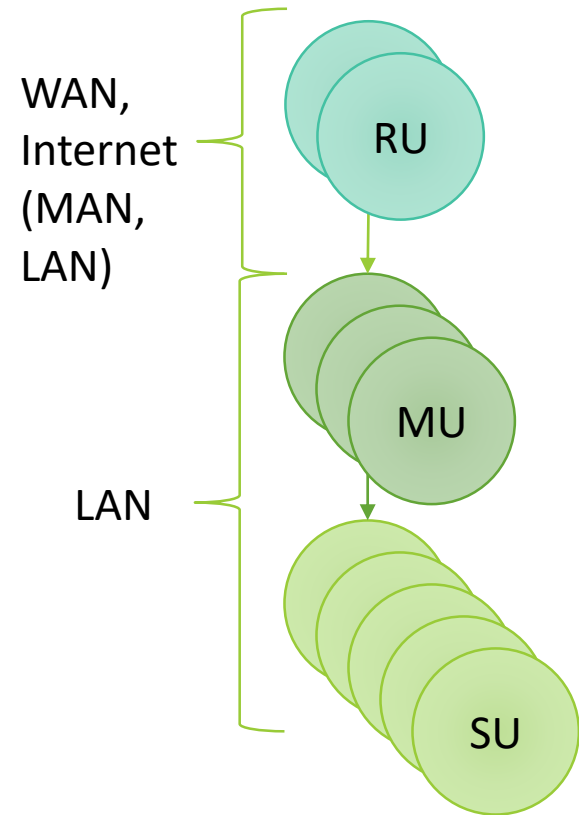
Remote utility, heavy-duty computing, corporate DB management, corporation mainframes and value-added network services

Services:

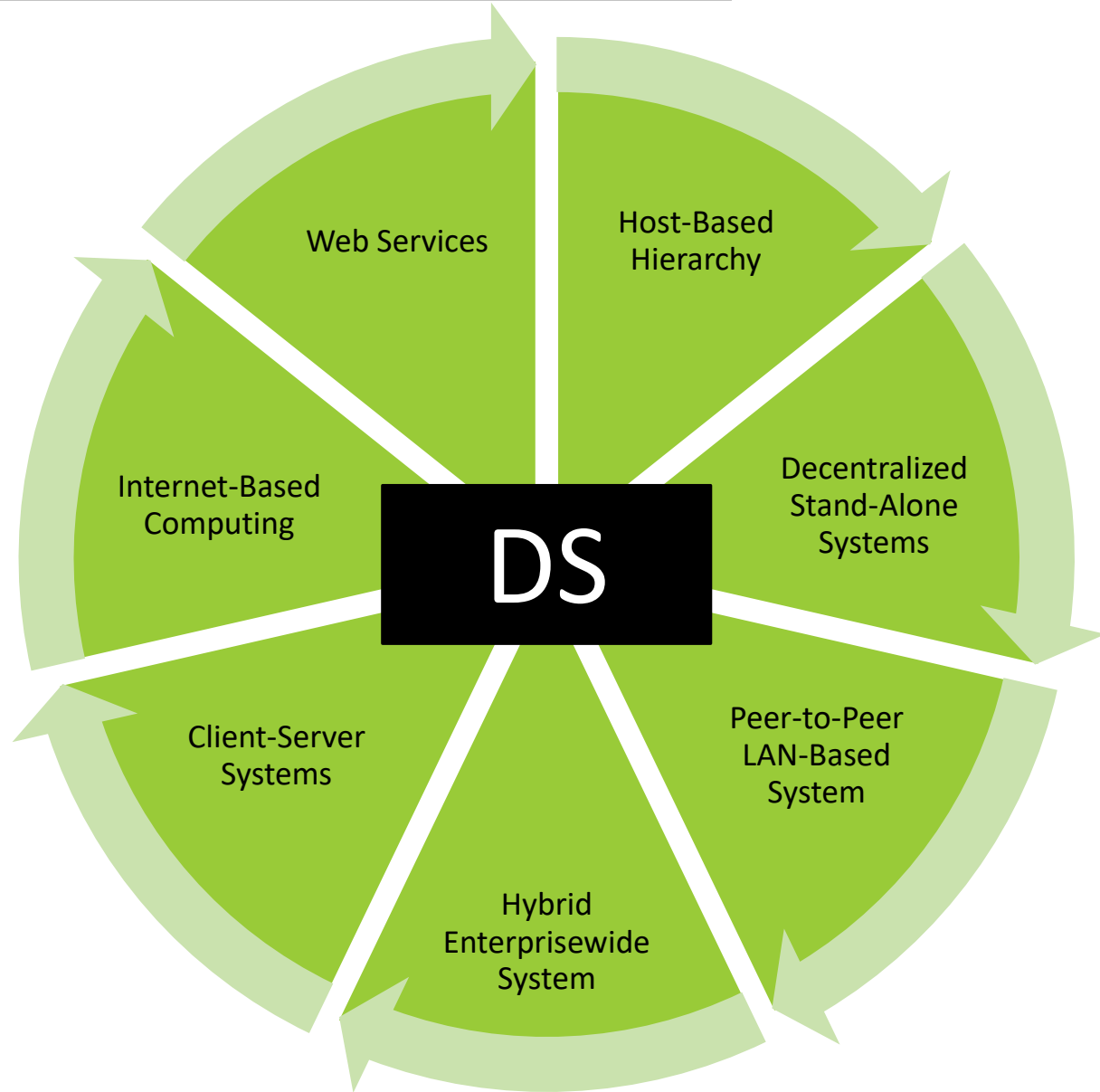
Access, Files, distributed apps (C-S model), ...

Standards:

OS, Protocols (TCP/IP), SQL, HTML, XML, ...

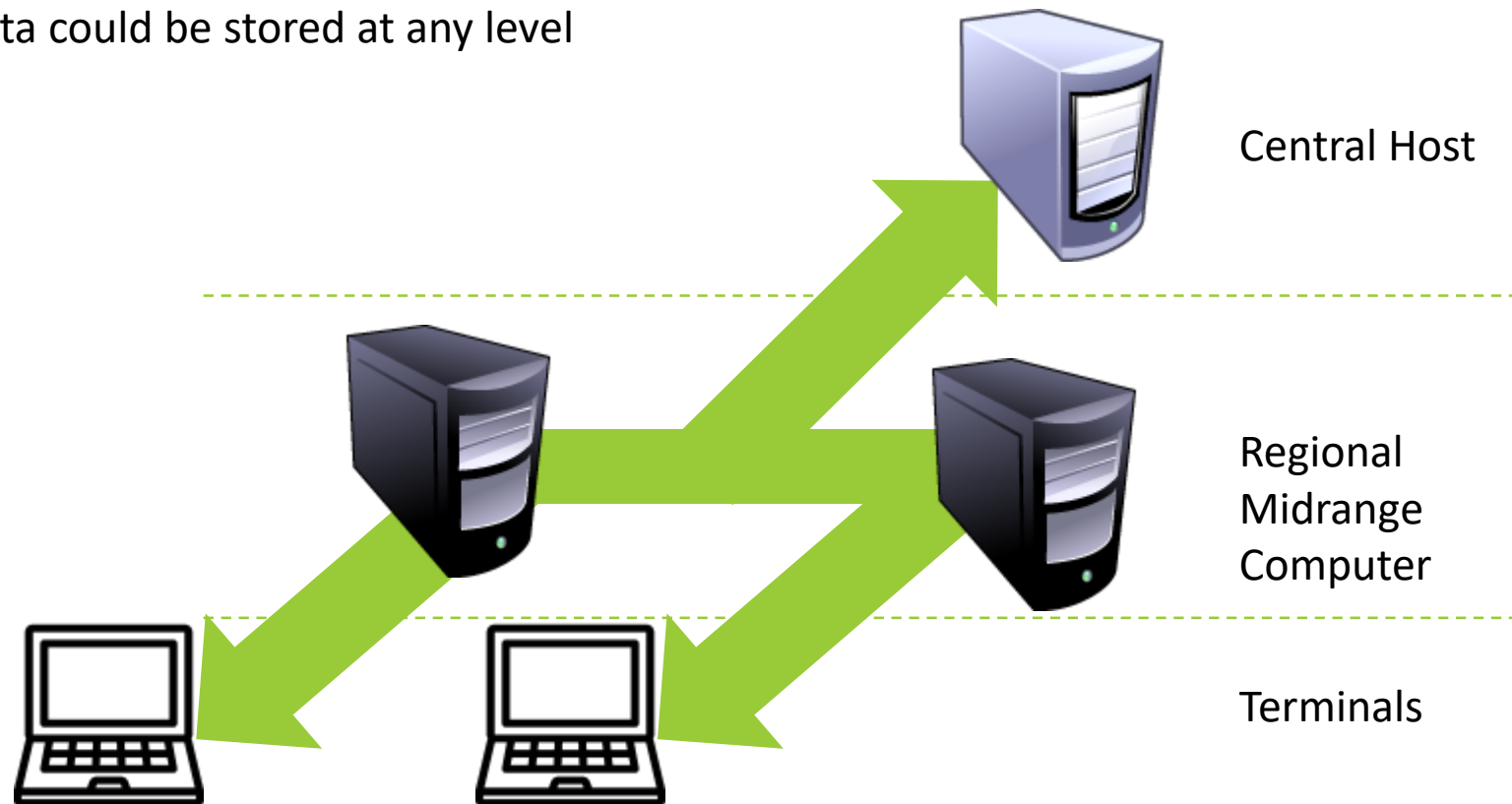


Taxonomy (1 of them): System Architectures



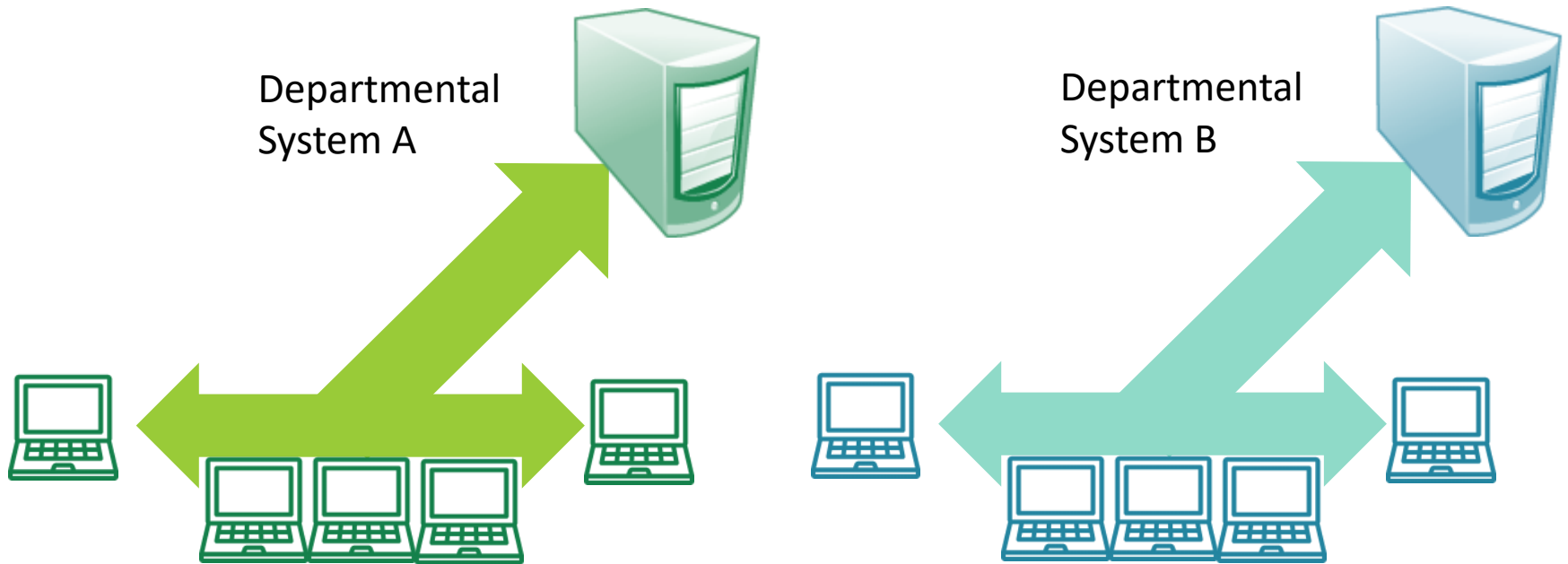
Host based Hierarchy

- First data processing distributed system. Host computer central, controlling component; terminals are access systems
- In Master-slave relationship: a central mainframe at the top, PCs at the bottom, minicomputers in between
- Data could be stored at any level



Decentralized Standalone Systems

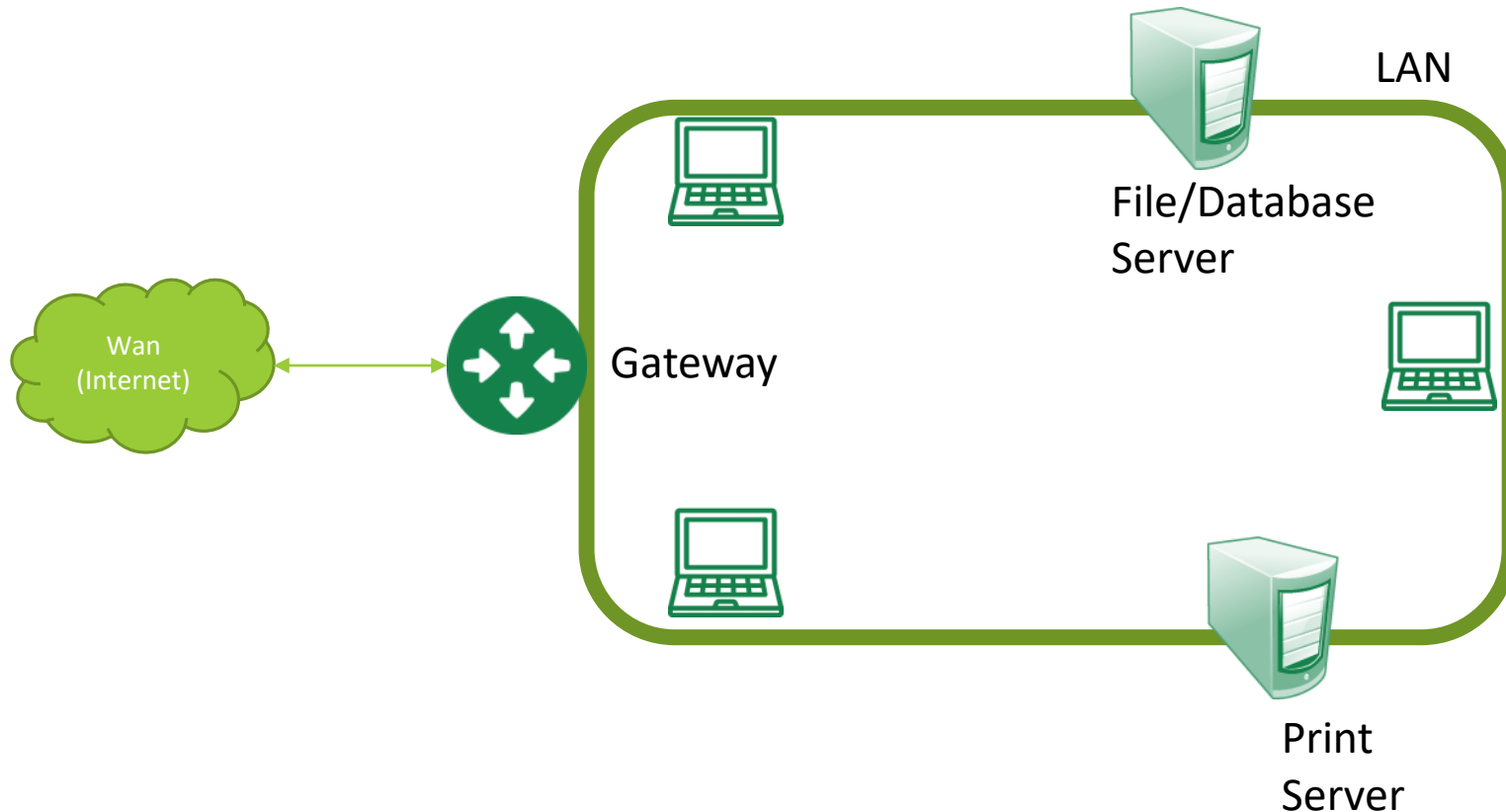
- Decentralized but does not really form a distributed system
- Holdover from the past where departments put in their own minicomputers with no intention of connecting them to the corporate host or to other departmental systems



A major goal in introducing ERP systems was to replace such disparate systems with a single platform of inter-connectable modules to serve these various functions

Peer-to-Peer/LAN Based Systems

- No hierarchy
 - “Peer-to-peer” communications
- Interconnecting LANs rather than hierarchical communications through a central hub
 - No “superior” computer



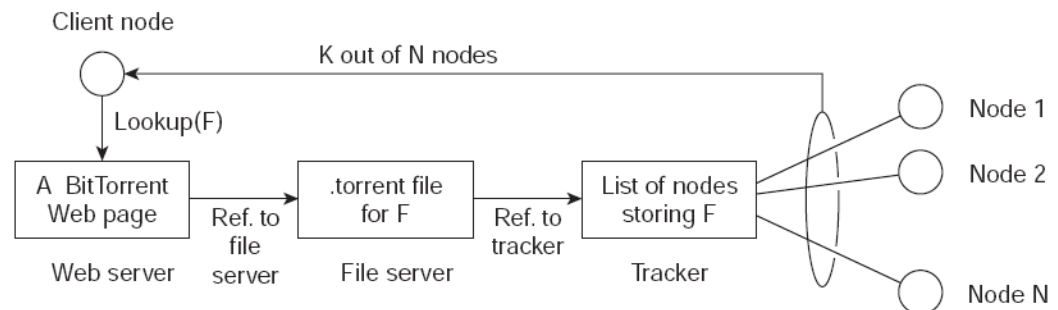
Peer-to-Peer

- Organize the nodes in a structured overlay network such as a logical ring, and make specific nodes responsible for services based only on their ID.
- A common approach is to use a distributed hash table (DHT) to organize the nodes.
- Traditional hash functions convert a key to a hash value, which can be used as an index into a hash table:
 - Keys are unique -- each represents an object to store in the table;
 - The hash function value is used to insert an object in the hash table and to retrieve it.
- In a DHT, data objects and nodes are each assigned a key which hashes to a random number from a very large identifier space (to ensure uniqueness).
- A mapping function assigns objects to nodes, based on the hash function value.
- A lookup, also based on hash function value, returns the network address of the node that stores the requested object.

Hybrid : C/S with P2P : BitTorrent

Users cooperate in file distribution

Once a node has identified where to download a file from, it joins a swarm of downloaders who in parallel get file chunks from the source, but also distribute these chunks amongst each other.



Hybrid Enterprisewide Systems

Combination two types of distributed system via network (e.g. WAN)

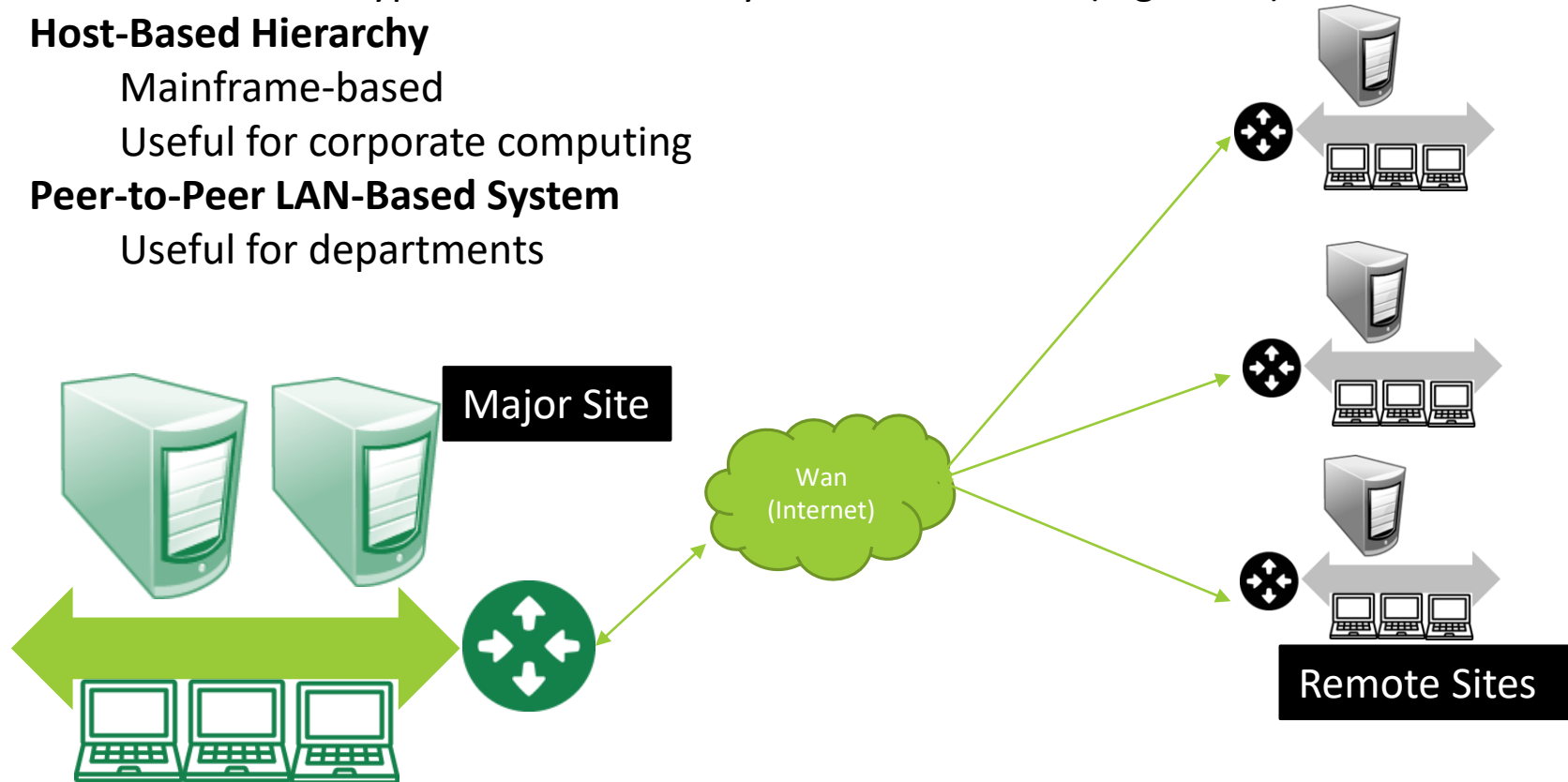
Host-Based Hierarchy

Mainframe-based

Useful for corporate computing

Peer-to-Peer LAN-Based System

Useful for departments



Allows company to link “Automation Islands” and retain IT investments, begin to automate business processes

Such cooperating processes allow to take advantage of specialized computer programs, while at the same time extending the usefulness of some legacy systems

- The process of pulling together such individual applications or components is called Systems Integration

Hybrid Enterprise-wide Systems

Combination two types of distributed system via network (e.g. WAN)

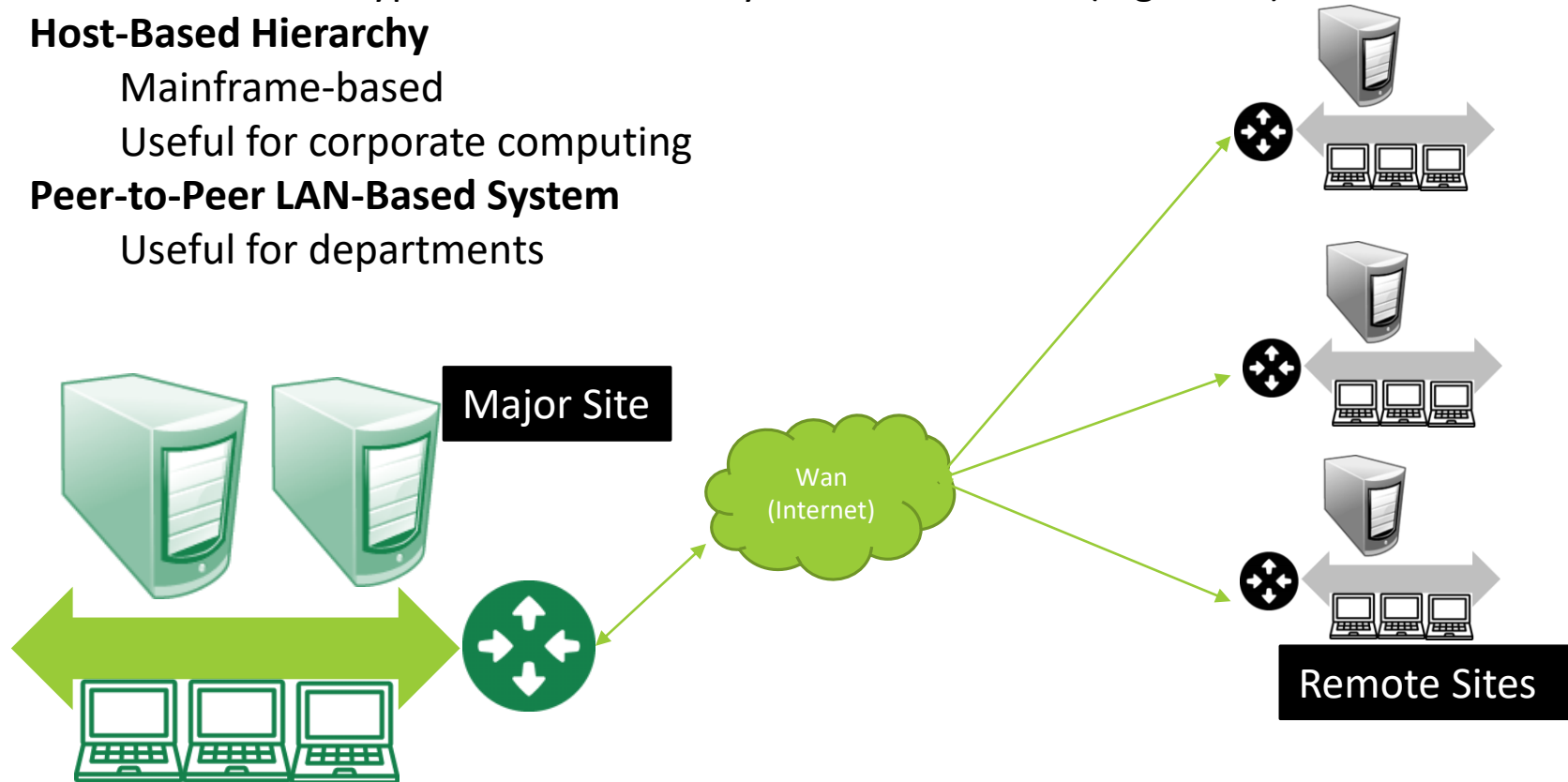
Host-Based Hierarchy

Mainframe-based

Useful for corporate computing

Peer-to-Peer LAN-Based System

Useful for departments



Allows company to link “Automation Islands” and retain IT investments, begin to automate business processes

Such cooperating processes allow to take advantage of specialized computer programs, while at the same time extending the usefulness of some legacy systems

- The process of pulling together such individual applications or components is called Systems Integration

Client Server Systems

Arose to take advantage of the processing capabilities of both host machines and PCs in the same system

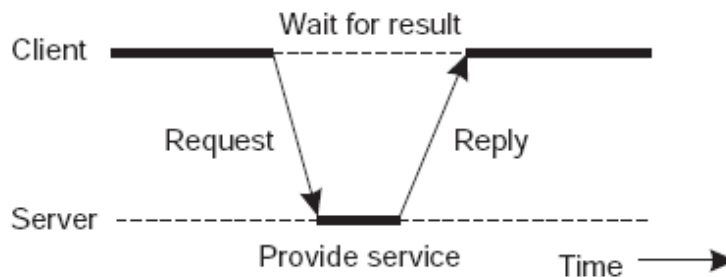
The computational flow of pure client-server systems is asymmetric.

Splitting work between clients and servers.

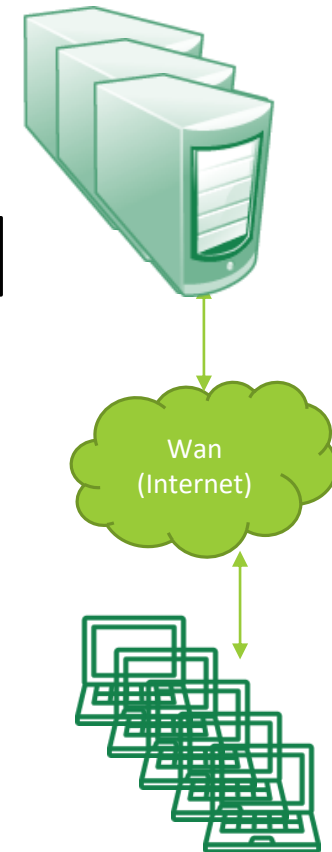
Constraints on the use of the client-server style

Limit the number of clients that can be connected to a Server.

Impose a restriction that servers cannot interact with other servers



Servers



Client Server Systems

splitting work between clients and servers

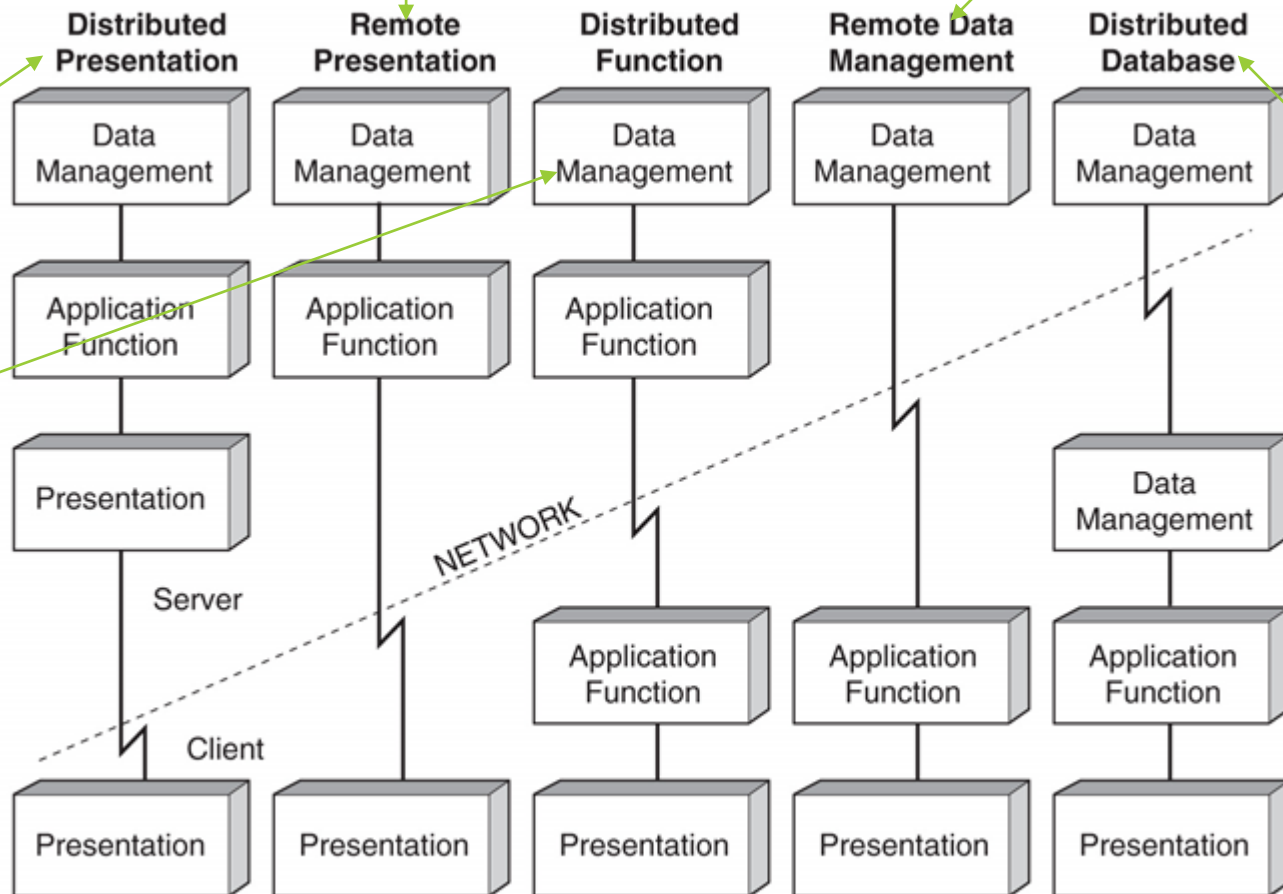
Most of the Web-based content systems Today. Database sitting at the back. Web servers provide contents. Browsers display contents

Traditional Client-Server database applications (Fat Client). DBMS stores and retrieves data. Client applications process and display data. Client maintenance may be challenging

Put all the data, all the application software, and some of the presentation software on server

Some new Web-GIS systems (image rendering moved to the Client, reduce network flow, load balancing)
Most of the online-gaming systems

Mobile computing.
Less dynamic data stored locally.
Data synchronize when connected
Some applications in retail chains. Local data copy to improve service availability



Source: Roger Woolfe, *Managing the Move to Client-Server*, Wentworth Research Program

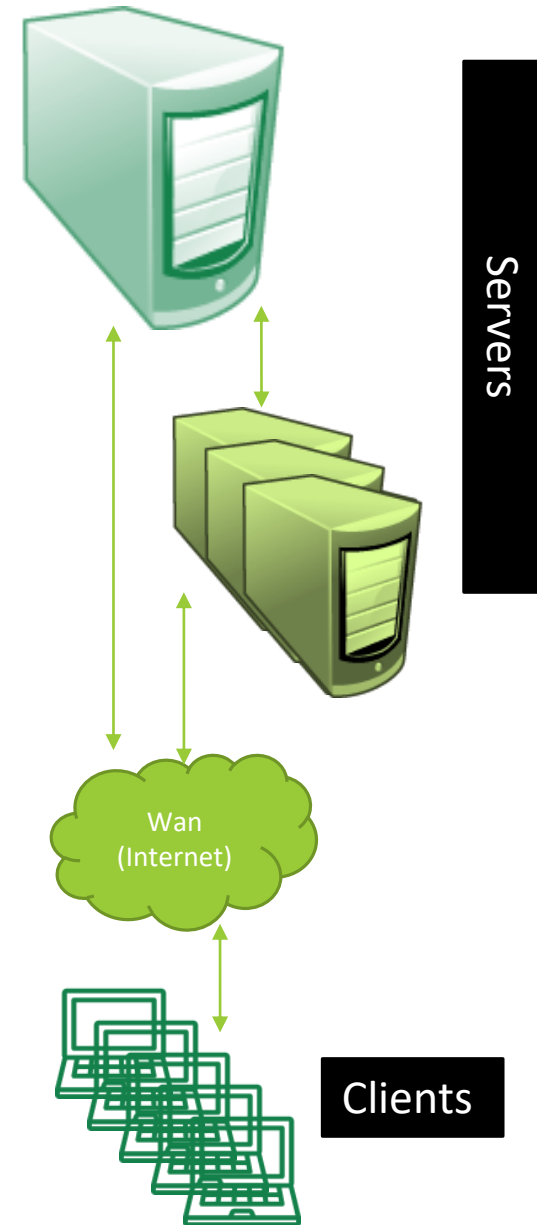
Client Server Systems

Preferred: **three-tier architecture**

Tier 3: the superserver/mainframe. Allows inclusion of legacy applications, short-lived and fast-changing data, and integrity rules

Tier 2: specialized servers, dedicated to housing databases or middleware-software to ease connection between client and server. Also, department-specific data that does not change often

Tier 1: clients (some of which could be portable) connected through network



Thin and fat clients

Thin client model

In a thin client model, all of the application processing and data management is carried out on the server. The client is simply

responsible for running the presentation software.

Used when legacy systems are migrated to client server architectures.

The legacy system acts as a server in its own right with a graphical interface implemented on a client.

A major disadvantage is that it places a heavy processing load on both the server and the network.

Fat client model

In this model, the server is only responsible for data management.

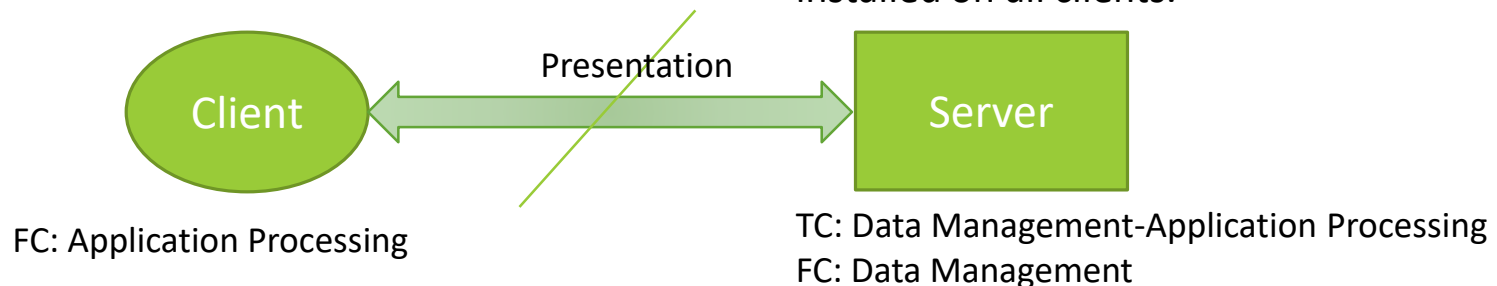
The software on the client implements the application logic and the interactions with the system user.

More processing is delegated to the client as the application processing is locally executed.

Most suitable for new C/S systems where the capabilities of the client system are known in advance.

More complex than a thin client model especially for management.

New versions of the application have to be installed on all clients.

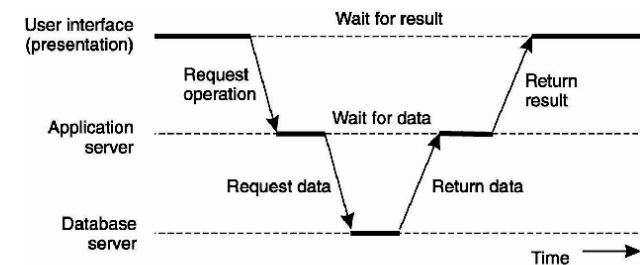
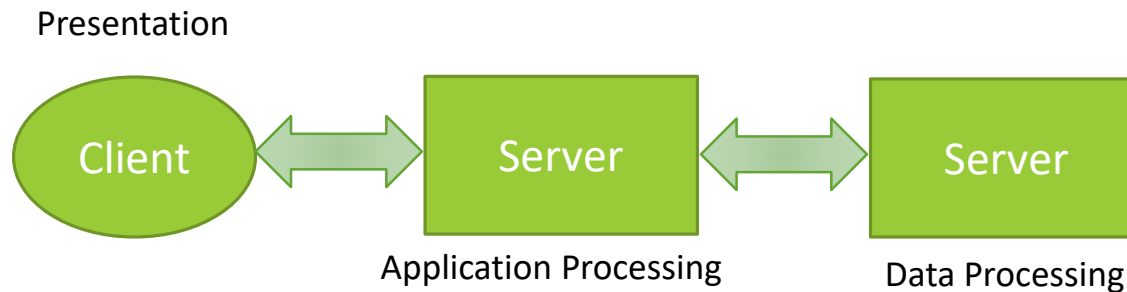


Three tier architectures

In a three-tier architecture, each of the application architecture layers may execute on a separate processor.

Allows for better performance than a thin client approach and is simpler to manage than a fat client approach.

A more scalable architecture as demands increase, extra servers can be added.



Architecture

Two-tier C/S architecture with thin clients

Two-tier C/S architecture with fat clients

Three-tier or multi-tier C/S architecture

Applications

Legacy system applications where separating application processing and data management is impractical. Computationally-intensive applications such as compilers with little or no data management. Data-intensive applications (browsing and querying) with little or no application processing.

Applications where application processing is provided by off-the-shelf software (e.g. Microsoft Excel) on the client. Applications where computationally-intensive processing of data (e.g. data visualisation) is required. Applications with relatively stable end-user functionality used in an environment with well-established system management.

Large scale applications with hundreds or thousands of clients. Applications where both the data and the application are volatile. Applications where data from multiple sources are integrated.

Client Server Systems

Advantages/Drawbacks

Better access to information

Shift the focus of computing to user and empower
Employees

Increases organizational flexibility:

- Allows new technology to be added more easily without affecting rest of system

- Streamlines work flow between functional areas

- Encourages people to work together via networks

- Supports new organizational structures via its connectivity



Expensive

Easier for users, but complex for IT,
Information System

What looked like simple connections have
turned into large, often fragile, complex
systems



Internet-Based Computing

In the late 1990s, the client-server trend was augmented by the Internet

The tenets of client-server remain

With Internet sitting at the heart

Ubiquitous computing

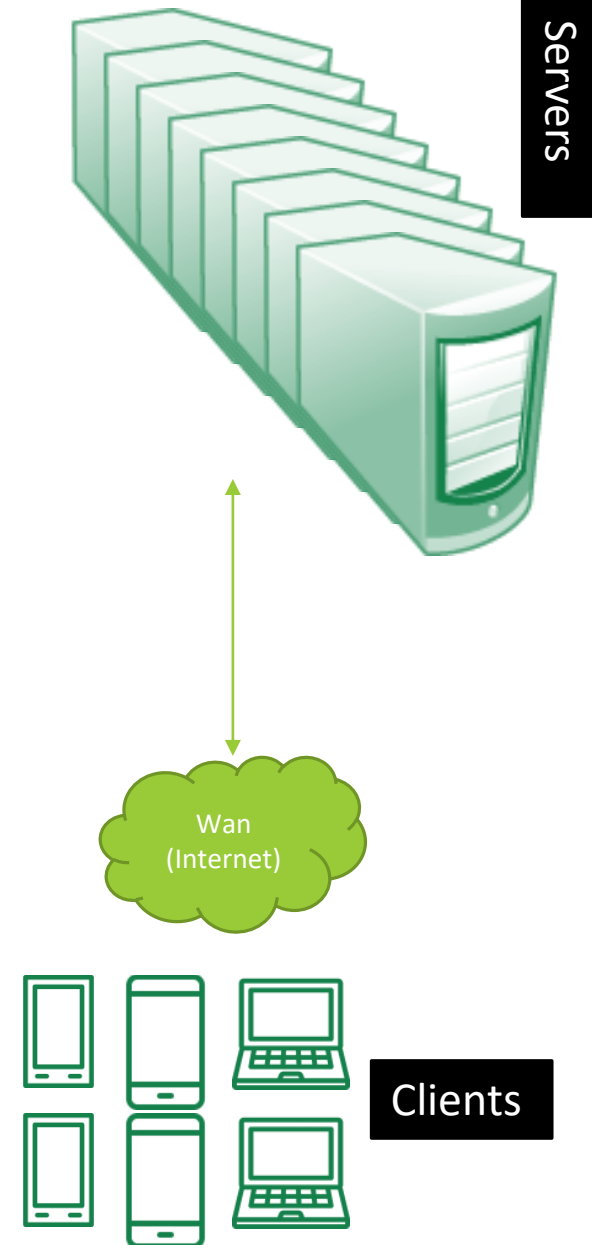
Various different platforms: Servers, PC, handhelds

A new computing environment

Irregular request arrival patterns

Unpredictable user numbers

Network-centric computing = a computer and a cloud



Internet-Based Computing

Network computers have not taken off desktops, idea of computing over Internet

Evolution of Network Computers (such as thin clients)

Thin clients = logical for hand held but now = increasingly popular for the desktop

Server-Base Computing

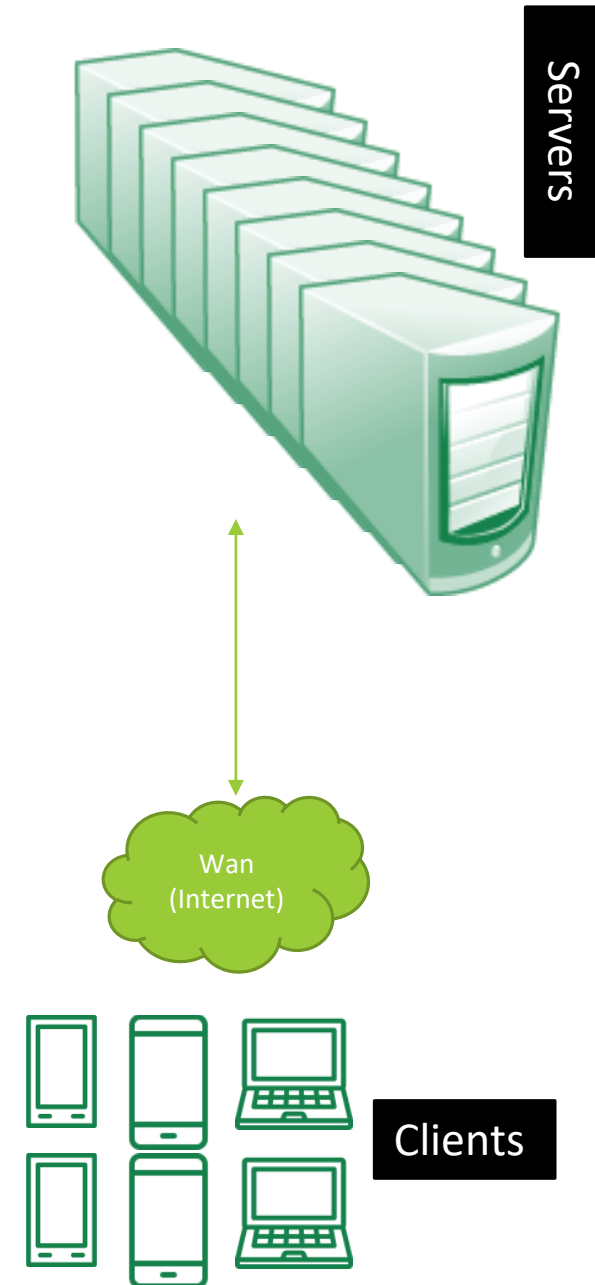
With more use of laptops which do not have strong security features

Updating in masse is not easy

Even individual downloads can require helpdesk support

Applications reside on corporate servers rather than on laptops

Applications can be securely accessed by any device, they can be updated directly on the server, and they do not have to be tailored to run on specific machines



Distributed Pervasive Systems

Small nodes, mobile, usually embedded, battery-powered

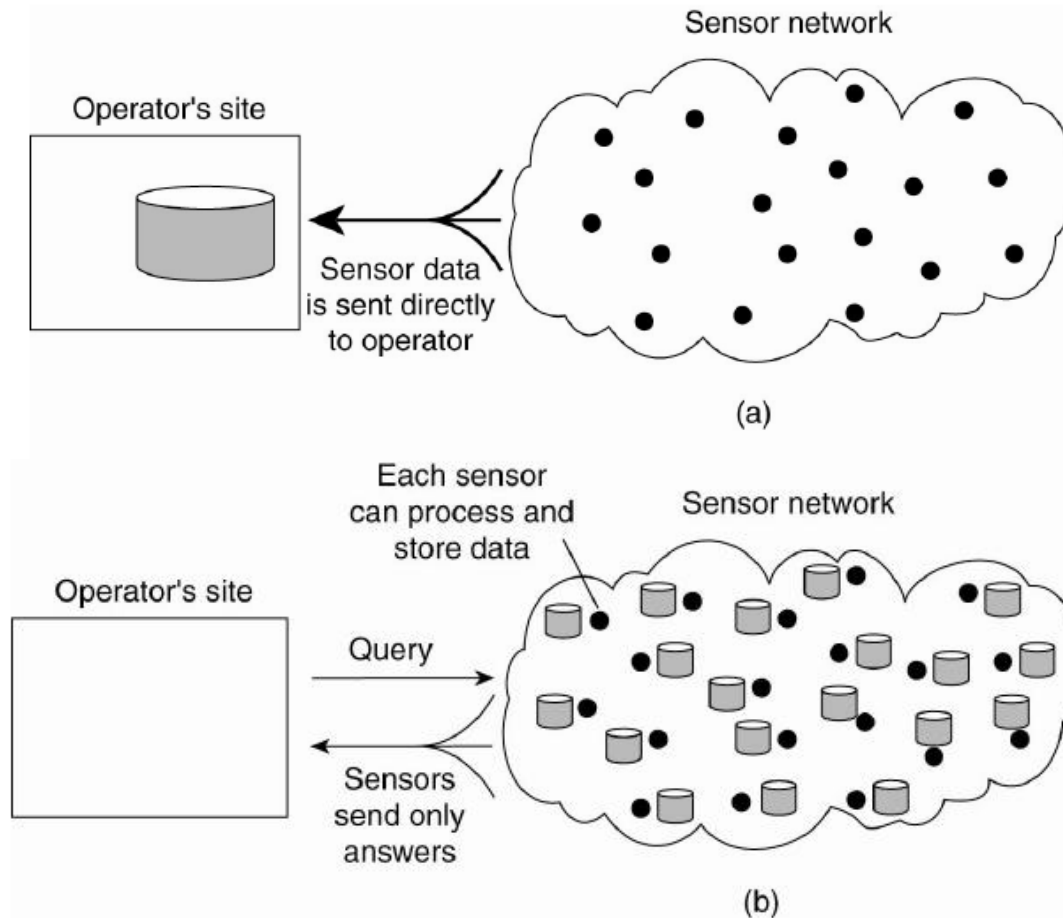
Requirements for pervasive systems

Embrace contextual changes.

Encourage ad hoc composition.

Recognize sharing as the default.

New tendencies (Dew, Fog, Edge)



Enterprise Architecture Framework

FIGURE 5-13 Enterprise Architecture—A Framework

Information

system

components:

Represent physical manifestations of the system

Data models (what it is made of)

Functional models (how it works)







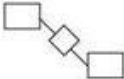

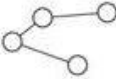
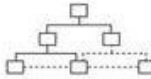

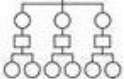
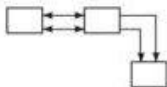

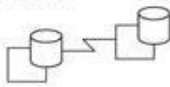
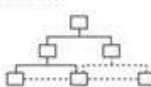

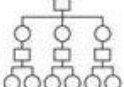
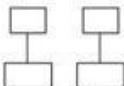

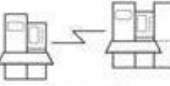
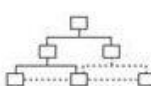

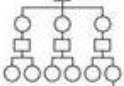






Network models (where the components are located)

And:

People (who)

Time (when)

Motivation (why)

	Data <i>What</i>	Function <i>How</i>	Network <i>Where</i>	People <i>Who</i>	Time <i>When</i>	Motivation <i>Why</i>
Objectives/ Scope (Contextual)	List of things important to the business 	List of processes the business performs 	List of locations in which the business operates 	List of Organizations/agents important to the business 	List of events significant to the business 	List of business goals/strategies 
<i>Planner</i>	Entity = Class of business thing	Function = Class of business process	Node = Major business location	Agent = Class of agent	Time = Major business event	Ends/Means = Major bus goal/critical success factor
Enterprise Model (Conceptual)	e.g., Semantic model 	e.g., Business process model 	e.g., Logistics network 	e.g., Organization chart 	e.g., Master schedule 	e.g., Business plan 
<i>Owner</i>	Ent = Business entity ReIn = Business relationship	Proc = Business process I/O = Business resources	Node = Business location Link = Business linkage	Agent = Organization unit Work = Work product	Time = Business event Cycle = Business cycle	End = Business objective Means = Business strategy
System Model (Logical)	e.g., Data model 	e.g., "Application architecture" 	e.g., Distributed system architecture 	e.g., Human interface architecture 	e.g., Processing structure 	e.g., Knowledge architecture 
<i>Designer</i>	Ent = Data entry ReIn = Data relationship	Proc = Application function I/O = User views	Node = I/S function (processor, storage, etc.) Link = Line characteristics	Agent = Role Work = Deliverable	Time = System event Cycle = Processing cycle	Ends = Criterion Means = Business rules
Technology Model (Physical)	e.g., Data design 	e.g., System design 	e.g., System architecture 	e.g., Human/technology interface 	e.g., Control structure 	e.g., Knowledge design 
<i>Builder</i>	Ent = Segment/Row/etc ReIn = Pointer/Key/etc	Proc = Computer function I/O = Screen/Device formats	Node = Hardware/System software Link = Line specifications	Agent = User Work = Job	Time = Execute Cycle = Component cycle	Ends = Condition Means = Action
Detailed Representations (out-of-context)	e.g., Data definition 	e.g., Program 	e.g., Network architecture 	e.g., Security architecture 	e.g., Timing definition 	e.g., Knowledge definition 
<i>Sub-Contractor</i>	Ent = Field ReIn = Address	Proc = Language stmt I/O = Control block	Node = Addresses Link = Protocols	Agent = Identity Work = "Transaction"	Time = Interrupt Cycle = Machine cycle	End = Subcondition Means = Step
Functioning System	e.g., Data	e.g., Function	e.g., Network	e.g., Organization	e.g., Schedule	e.g., Strategy

Source: Adapted from John Zachman, Zachman International, 2222 Foothill Blvd., Suite 337, LaCanada, CA 91011.

Software Architecture

SA serves as a blueprint for a system. It provides an abstraction to manage the system complexity and establish a **communication and coordination mechanism among components**. It defines a structured solution to meet all the technical and operational requirements, while optimizing the common quality attributes like performance and security.

Significant decisions about the organization related to software development and each of these decisions can have a considerable impact on quality, maintainability, performance, and the overall success of the final product.

- Selection of structural elements and their interfaces by which the system is composed.

- Behaviour as specified in collaborations among those elements.

- Composition of these structural and behavioural elements into large subsystem.

- Architectural decisions align with business objectives.

- Architectural styles guide the organization.

The software that is built for computer-based systems exhibit one of many architectural styles. Each style describes a system category that encompasses:

- A set of component types which perform a required function by the system.

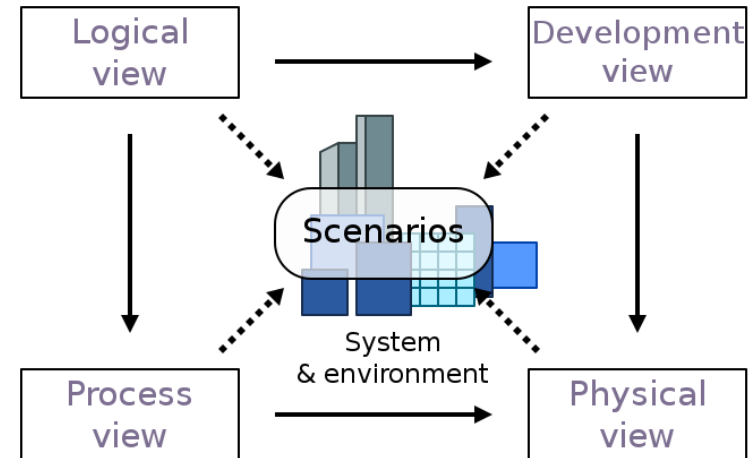
- A set of connectors (subroutine call, remote procedure call, data stream, and socket) that enable communication, coordination, and cooperation among different components.

- Semantic constraints which define how components can be integrated to form the system.

- A topological layout of the components indicating their runtime interrelationships.

Architectural Design

Category	Architectural Design	Description
Communication	Message bus	Prescribes use of a software system that can receive and send messages using one or more communication channels.
	Service-Oriented Architecture (SOA)	Defines the applications that expose and consume functionality as a service using contracts and messages.
Deployment	Client/server	Separate the system into two applications, where the client makes requests to the server.
	3-tier or N-tier	Separates the functionality into separate segments with each segment being a tier located on a physically separate computer.
Domain	Domain Driven Design	Focused on modeling a business domain and defining business objects based on entities within the business domain.
Structure	Component Based	Breakdown the application design into reusable functional or logical components that expose well-defined communication interfaces.
	Layered	Divide the concerns of the application into stacked groups (layers).
	Object oriented	Based on the division of responsibilities of an application or system into objects, each containing the data and the behavior relevant to the object.



Web-Services

This second-generation Internet-based distributed system gives software modules URLs (Internet addresses) so they can be called upon to perform their function as a service via the Internet

This development will permit widespread computer-to-computer use of the Internet. One computer program or Web Service makes a request of another Web Service to perform its task (or set of tasks) and pass back the answer

Wrapping: Encapsulate functionality from an existing application in an XML envelope

Exposing: For use by others

Web Services Standards:

Three software standards:

- XML (eXtensible Markup Language)
- WSDL (Web Services Definition Standard)
- UDDI (Universal Discovery, Description, and Integration)

Three communication standards

- SOAP (Simple Object Access Protocol)
- HTTP (HyperText Transfer Protocol)
- TCP/IP (Transmission Control Protocol / Internet Protocol)

Service–Oriented Architecture

A service has four properties according to one of many definitions of SOA:

- It logically represents a business activity with a specified outcome.

- It is self-contained.

- It is a black box for its consumers.

- It may consist of other underlying services.

Different services can be used in conjunction to provide the functionality of a large software application.

The definition could be a definition of modular programming in the 1970s.

SOA is less about how to modularize an application, and more about how to compose an application by integration of distributed, separately-maintained and deployed software components.

It is enabled by technologies and standards that make it easier for components to communicate and cooperate over a network, especially an IP network.

SOA: building blocks can play any of the three roles

Service provider: It creates a web service and provides its information to the service registry. Each provider debates upon a lot of hows and whys like which service to expose, whom to give more importance: security or easy availability, what price to offer the service for and many more.

Service broker, service registry or service repository: Its main functionality is to make the information regarding the web service available to any potential requester. Whoever implements the broker decides the scope of the broker. Public brokers are available anywhere and everywhere but private brokers are only available to a limited amount of public.

Service requester/consumer: It locates entries in the broker registry using various find operations and then binds to the service provider in order to invoke one of its web services. Whichever service the service-consumers need, they have to take it into the brokers, bind it with respective service and then use it.

Implementation approaches

SOA can be implemented with Web services. This is done to make the functional building-blocks accessible over standard Internet protocols that are independent of platforms and programming languages. These services can represent either new applications or just wrappers around existing legacy systems to make them network-enabled.

Implementers commonly build SOAs using web services standards. One example is SOAP, which has gained broad industry acceptance (also referred to as web service specifications) also provide greater interoperability and some protection from lock-in to proprietary vendor software. One can, however, also implement SOA using any other service-based technology, such as Jini, CORBA or REST.

Architectures can operate independently of specific technologies:

- Web services based on WSDL and SOAP

- Messaging, e.g., with ActiveMQ, JMS, RabbitMQ

- RESTful HTTP, OPC-UA, WCF (Microsoft's Web services)

- Apache Thrift

- SORCER

Service-Oriented Architecture

Move component interactions from hard-coded to dynamically discovered and invoked

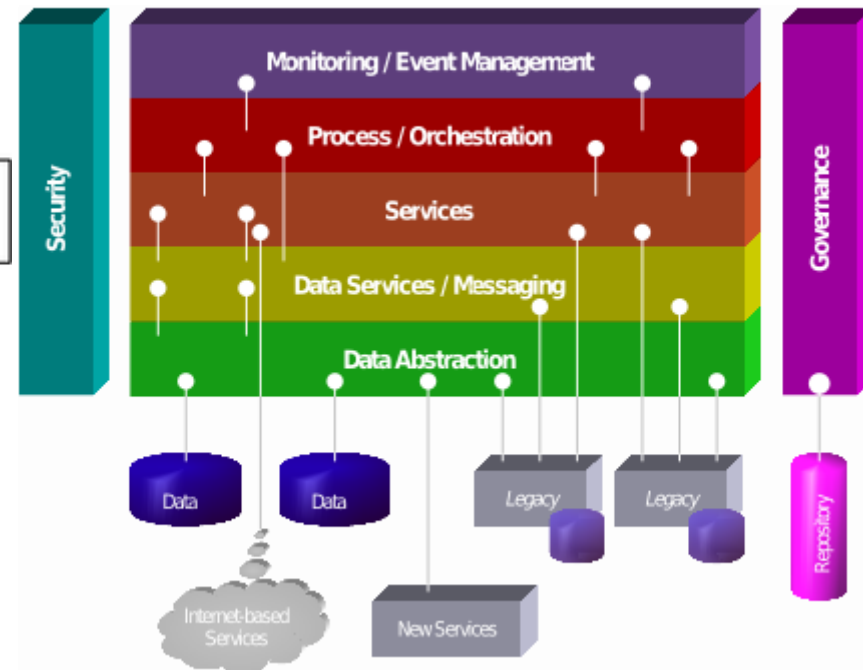
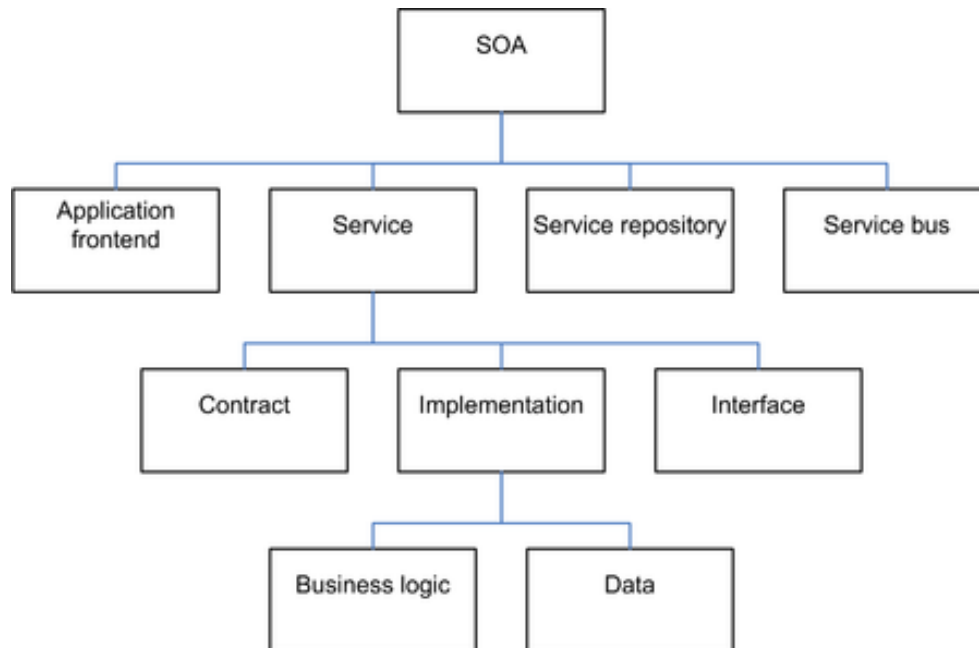
By exposing the data and functions in a way that other systems can easily use

Service-Oriented Architecture (SOA)

An **Architecture** is a formal description of a system, defining its components, their interrelationships, and the principles and guidelines governing their design and evolution over time

A **Service** is a software component that can be accessed via a network to provide functionality to a service requester

SOA is a style of building reliable distributed systems that deliver functionality as services, with the additional emphasis on loose coupling between interacting services



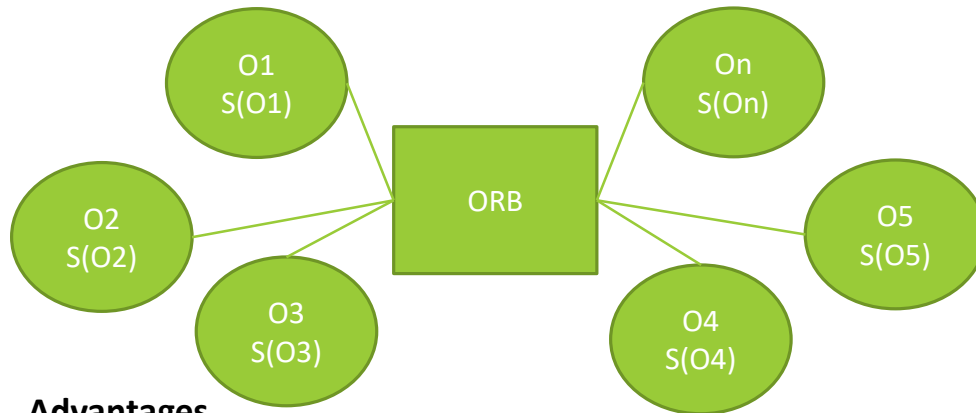
Distributed object architectures

There is no distinction in a distributed object architectures between clients and servers.

Each distributable entity is an object that provides services to other objects and receives services from other objects.

Object communication is through a middleware system called an object request broker.

However, distributed object architectures are more complex to design than C/S systems.



Advantages

It allows the system designer to delay decisions on where and how services should be provided.

It is a very open system architecture that allows new resources to be added to it as required.

The system is flexible and scalable.

It is possible to reconfigure the system dynamically with objects migrating across the network as required.

Uses

As a logical model that allows you to structure and organise the system. In this case, you think about how to provide application functionality solely in terms of services and combinations of services.

As a flexible approach to the implementation of client-server systems. The logical model of the system is a client-server model but both clients and servers are realised as distributed objects communicating through a common communication framework

The **Common Object Request Broker Architecture (CORBA)** standard by the Object Management Group (OMG). CORBA enables collaboration between systems on different operating systems, programming languages, and computing hardware & uses an object-oriented model. Is an example of the distributed object paradigm.

Microservices

Microservices is a variant of the service-oriented architecture (SOA) architectural style that structures an application as a collection of loosely coupled services.

In a microservices architecture, services should be fine-grained and the protocols should be lightweight.

The benefit of decomposing an application into different smaller services is that it improves modularity and makes the application easier to understand, develop and test.

It also parallelizes development by enabling small autonomous teams to develop, deploy and scale their respective services independently.

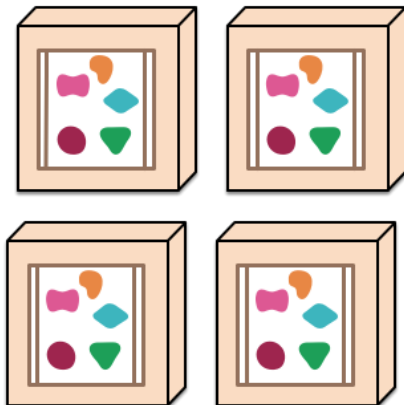
It also allows the architecture of an individual service to emerge through continuous refactoring.

Microservices-based architectures enable continuous delivery and deployment.

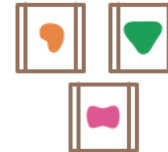
A monolithic application puts all its functionality into a single process...



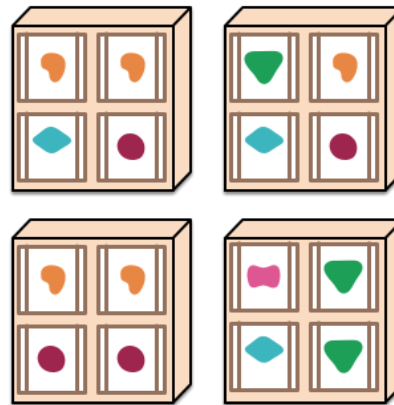
... and scales by replicating the monolith on multiple servers



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



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



Microservices

Some of the defining characteristics that are frequently cited include:

Services in a microservice architecture (MSA) are often processes that communicate with each other over a network in order to fulfil a goal using technology-agnostic protocols such as HTTP.

However, services might also use other kinds of inter-process communication mechanisms such as shared memory. Services might also run within the same process as, for example, OSGI bundles.

Services in a microservice architecture should be independently deployable.

The services are easy to replace.

Services are organized around capabilities, e.g., user interface front-end, recommendation, logistics, billing, etc.

Services can be implemented using different programming languages, databases, hardware and software environment, depending on what fits best.

Services are small in size, messaging enabled, bounded by contexts, autonomously developed, independently deployable, decentralized and built and released with automated processes.

A microservices-based architecture:

Naturally enforces a modular structure.

Lends itself to a continuous delivery software development process. A change to a small part of the application only requires one or a small number of services to be rebuilt and redeployed.

Adheres to principles such as fine-grained interfaces (to independently deployable services), business-driven development (e.g. domain-driven design), IDEAL cloud application architectures, polyglot programming and persistence, lightweight container deployment, decentralized continuous delivery, and DevOps with holistic service monitoring.

Provides characteristics that are beneficial to scalability.

Shared nothing architecture

A SN is a distributed computing architecture in which each node is independent and self-sufficient, and there is no single point of contention across the system. More specifically, none of the nodes share memory or disk storage. People typically contrast SN with systems that keep a large amount of centrally-stored state information, whether in a database, an application server, or any other similar single point of contention.

Advantages of SN architecture versus a central entity that controls the network (a controller-based architecture) include eliminating any single point of failure, allowing self-healing capabilities and providing an advantage with offering non-disruptive upgrade.

Applications

Shared nothing is popular for web development because of its scalability.

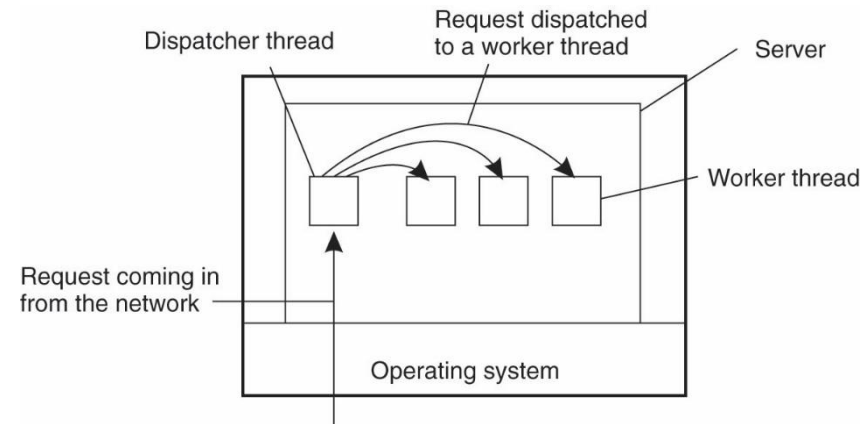
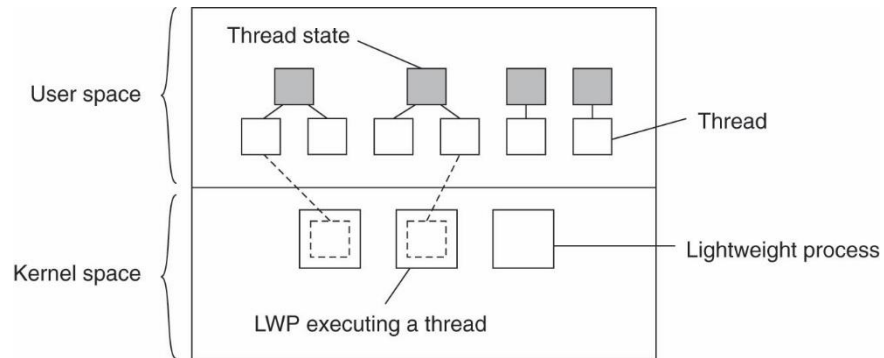
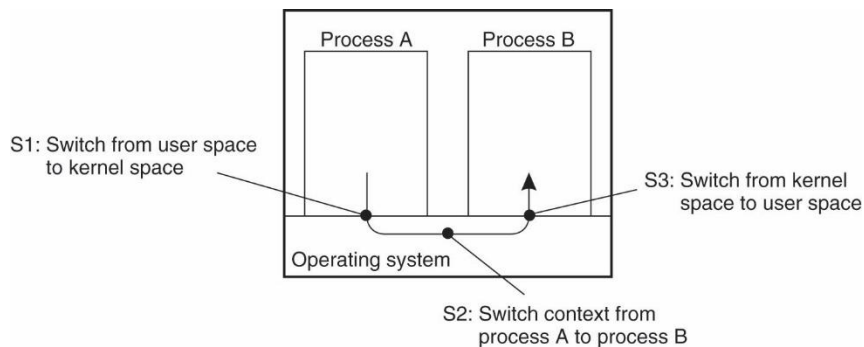
Google has demonstrated: a pure SN system can scale hugely simply by adding nodes in the form of inexpensive computers, since there is no single bottleneck to slow the system down. Google calls this **sharding**.

A SN system typically partitions its data among many nodes on different databases (assigning different computers to deal with different users or queries), or may require every node to maintain its own copy of the application's data, using some kind of coordination protocol. This is often referred to as database sharding.

Shared nothing architectures have become prevalent in the data warehousing space. Shared nothing architectures certainly take longer to respond to queries that involve joins over large data sets from different partitions (machines). However, the potential for scaling is huge.

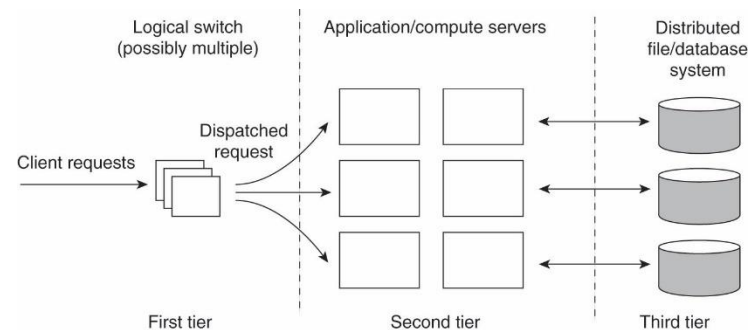
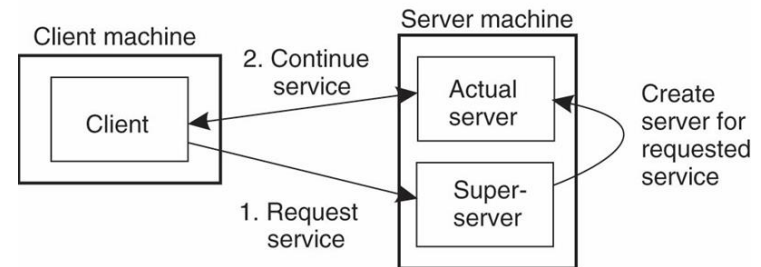
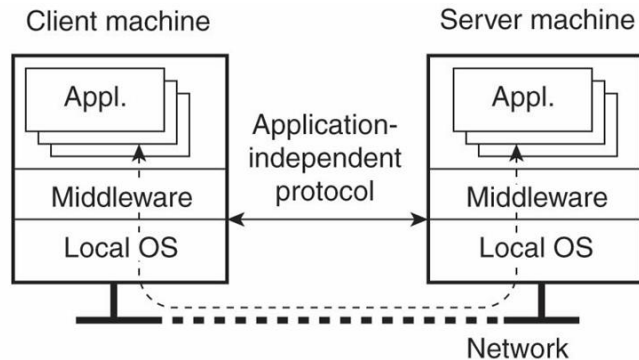
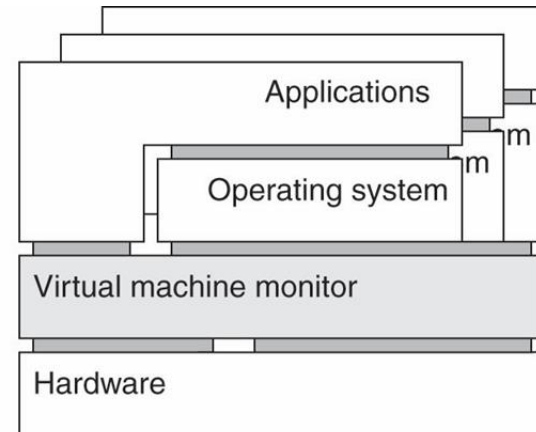
Processes execution architecture

- Shared processes (single machine) (message passing)
- Shared memory processes/thread
- Multithread Servers



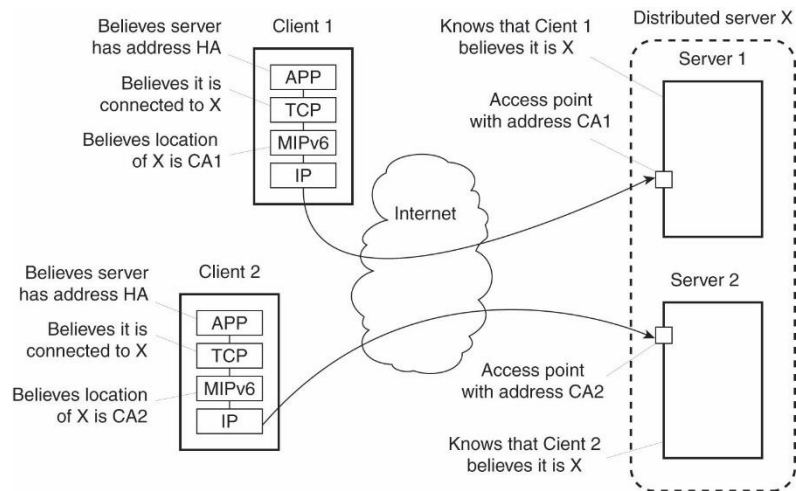
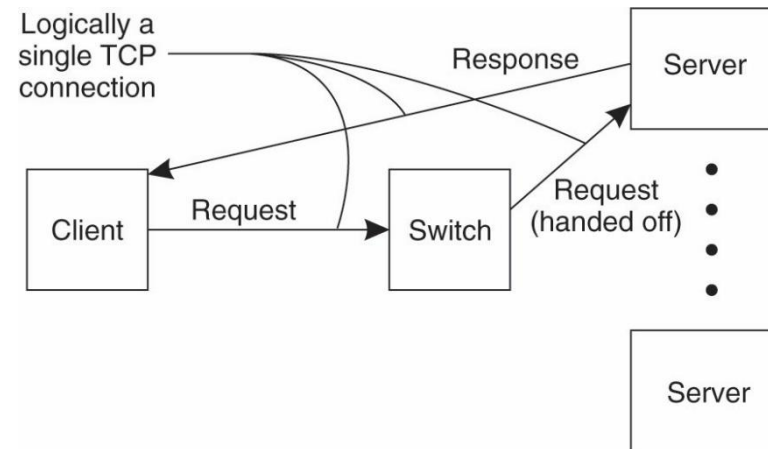
Processes execution architecture

- Virtualized Architectures
- Networked Architectures
- Client-Multiserver
- Client-Multiserver 3Tier



Processes execution architecture

- Server Cluster (handoff)
- Distributed Server



Cloud Computing vs. Distributed Computing

Computer network technologies have witnessed huge improvements and changes in the last 20 years. After the arrival of Internet (the most popular computer network today), the networking of computers has led to several novel advancements in computing technologies like Distributed Computing and Cloud Computing.

The term distributed systems and cloud computing systems slightly refer to different things, however the underlying concept between them is same.

Distributed Computing can be defined as the use of a distributed system to solve a single large problem by breaking it down into several tasks where each task is computed in the individual computers of the distributed system.

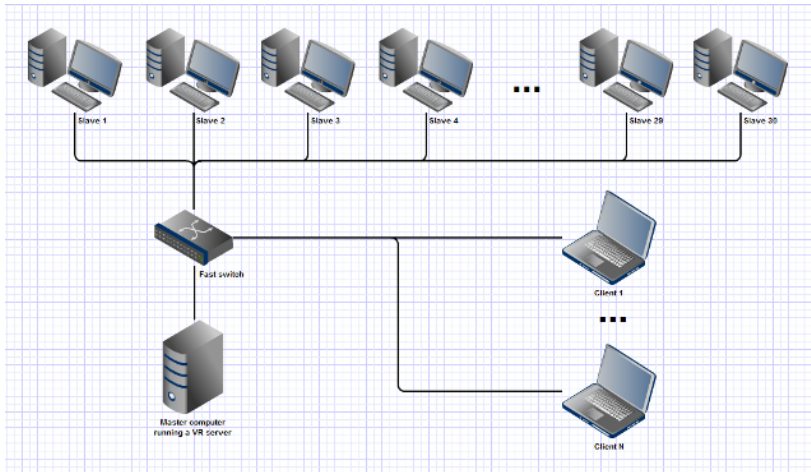
A distributed system consists of more than one self directed computer that communicates through a network.

All the computers connected in a network communicate with each other to attain a common goal by making use of their own local memory.

On the other hand, different users of a computer possibly might have different requirements and the distributed systems will tackle the coordination of the shared resources by helping them communicate with other nodes to achieve their individual tasks.

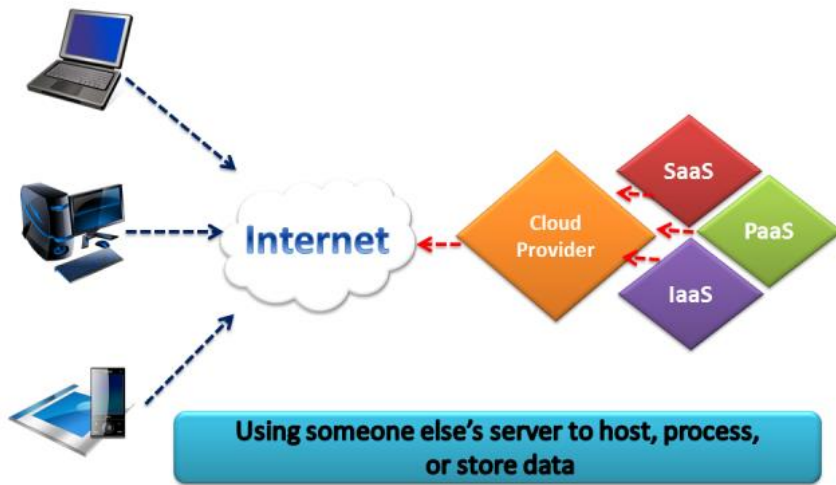
Examples: World Wide Web, Social Media (Facebook), Hadoop's Distributed File System (HDFS), ATM, Cloud Network Systems(Specialized form of Distributed Computing Systems), Google Bots, Google Web Server, Indexing Server, ...

Cloud Computing vs. Distributed Computing



The image illustrates the working of master/slave architecture model of distributed computing architecture where the master node has unidirectional control over one or more slave nodes.

The task is distributed by the master node to the configured slaves and the results are returned to the master node.



Cloud computing is a style of computing where massively scalable and flexible IT-related capabilities are delivered as a service to the users using Internet technologies, services may include:

Infrastructure, platform, applications, and storage space.

The users pay for these services, resources they actually use.

They do not need to build infrastructure of their own.

Cloud Computing vs. Distributed Computing

Cloud computing enables companies to consume computing resources as a utility — just like electricity — rather than having to build and maintain computing infrastructures in-house.”

Cloud computing usually refers to providing a service via the internet.

This service can be pretty much anything, from business software that is accessed via the web to off-site storage or computing resources whereas distributed computing means splitting a large problem to have the group of computers work on it at the same time.



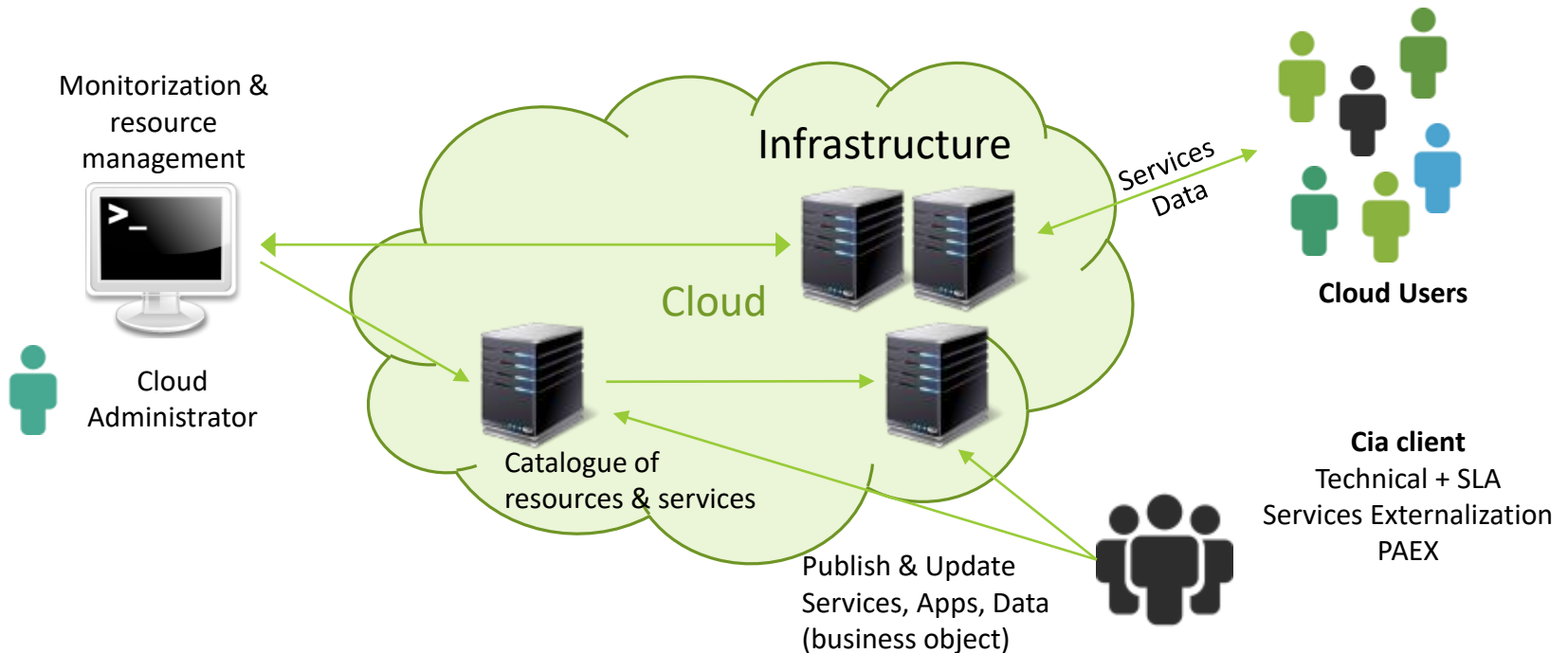
Examples:

YouTube is the best example of cloud storage which hosts millions of user uploaded video files.

Picasa and Flickr host millions of digital photographs allowing their users to create photo albums online by uploading pictures to their service's servers.

Google Docs is another best example of cloud computing that allows users to upload presentations, word documents and spreadsheets to their data servers. Google Docs allows users edit files and publish their documents for other users to read or make edits.

Architecture of Cloud Computing



“Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. This cloud model is composed of five essential characteristics, three service models, and four deployment models”.

National Institute of Standards & Technology (NIST)

Most Representative (Today): Cloud Computing

Essential Characteristics:

- On-demand self-service
- Broad network access
- Resource pooling.
- Rapid elasticity
- Measured service.

Service Models:

Software as a Service (SaaS): The capability provided to the consumer is to use the provider's applications running on a cloud infrastructure

Platform as a Service (PaaS): The capability provided to the consumer is to deploy onto the cloud infrastructure consumer-created or acquired applications created using programming

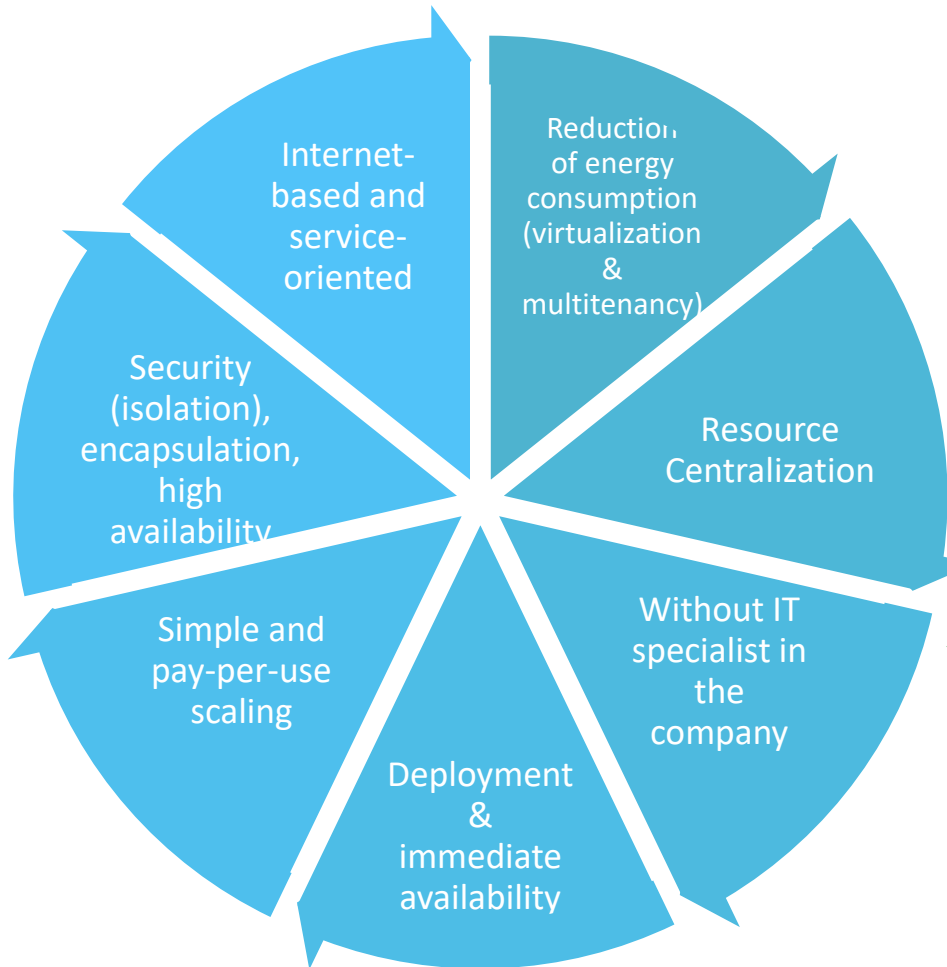
Infrastructure as a Service (IaaS): The capability provided to the consumer is to provision processing, storage, networks, and other fundamental computing resources where the consumer is able to deploy and run arbitrary software, which can include operating systems and applications.

Deployment Models:

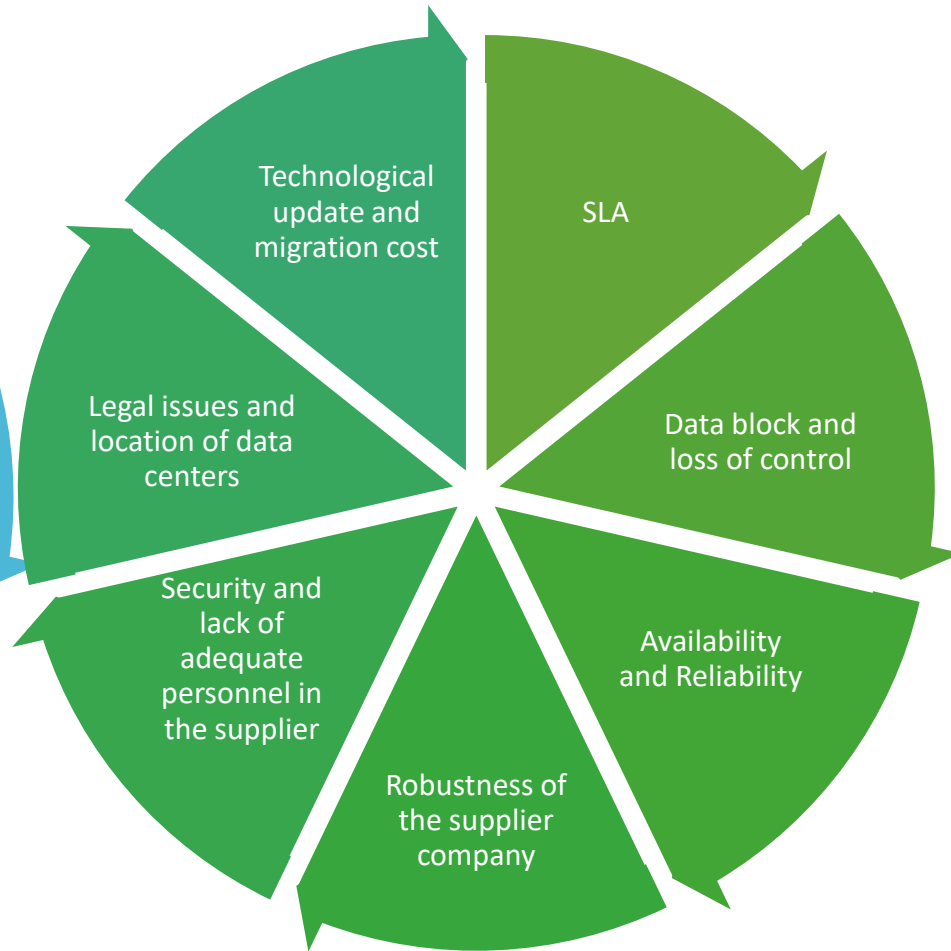
- Private cloud.
- Community cloud.
- Public cloud.
- Hybrid cloud

Most Representative (Today): Cloud Computing

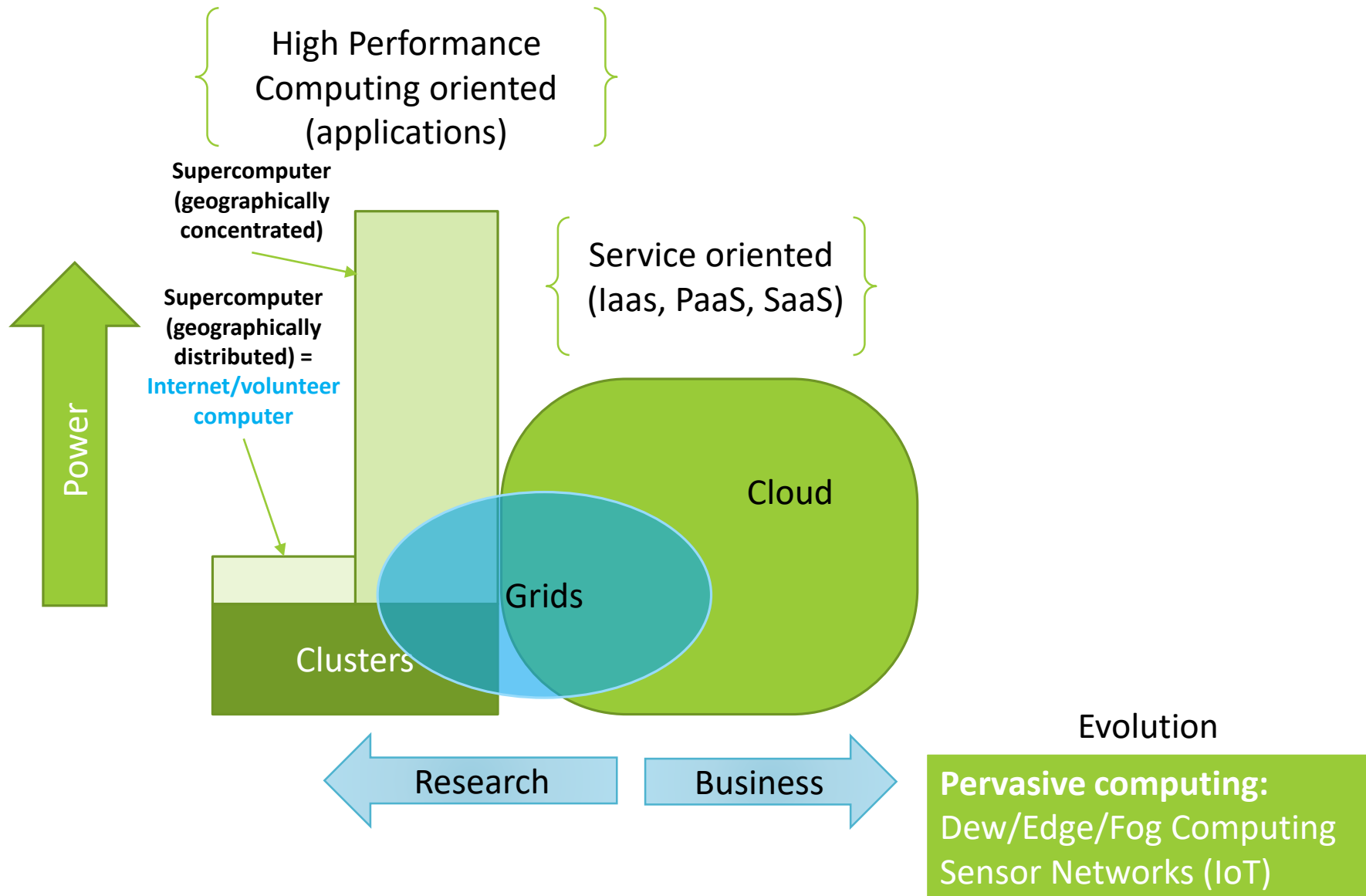
Advantages



Risks (possible)

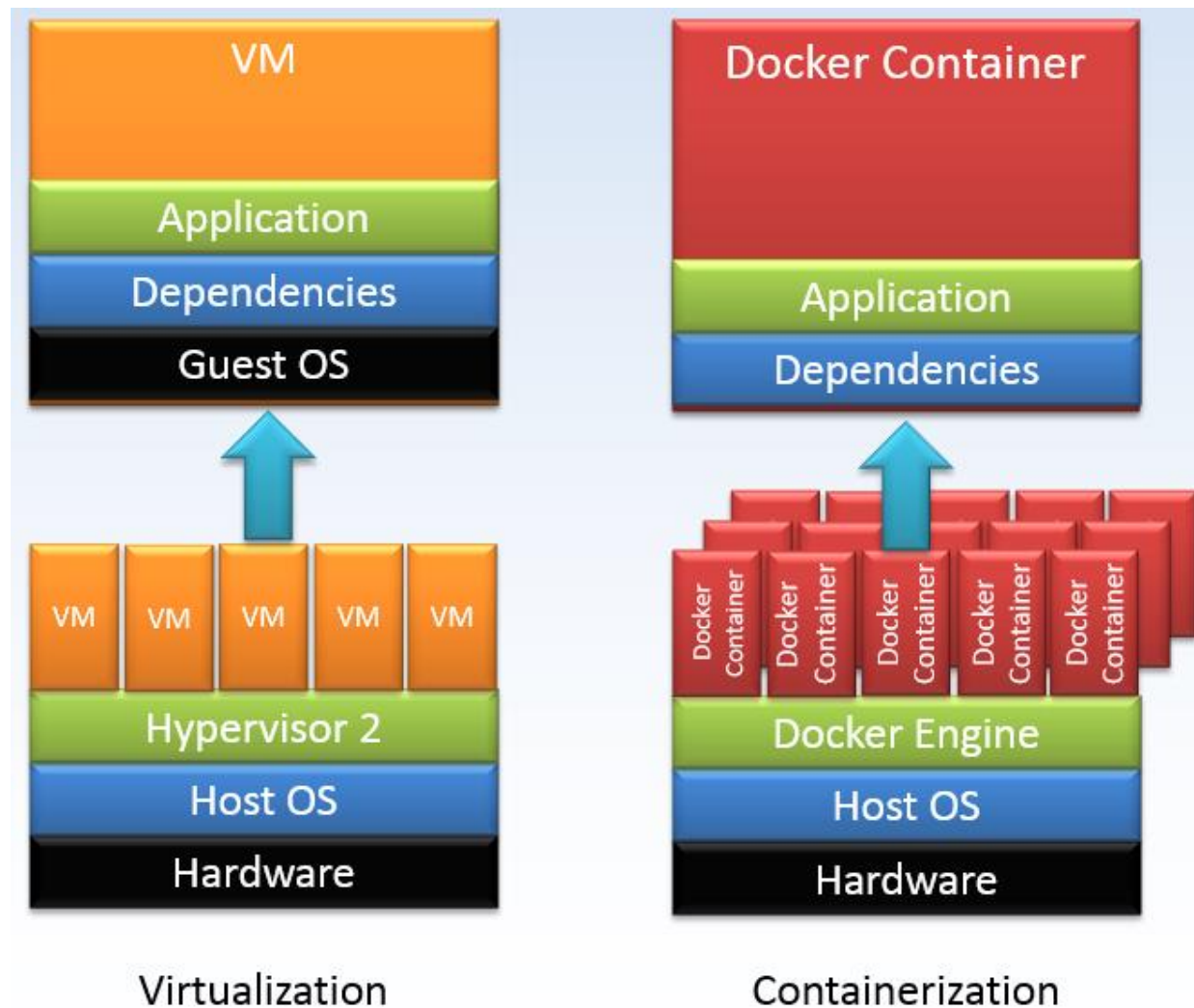


Where the Distributed Systems are used?



Today: what is the key to a massive DS?

A collection of independent processing nodes that appears to its users as a single coherent system.



Distributed Systems Organization (today)

Distributed Computing systems

- Client/Server Systems →
- Cluster Systems →
 - Dedicated Systems
 - Opportunistic Systems
- Grid Systems ↘
- Internet Computing/Volunteer Computing →
- Cloud Computing ↑ ↑
- Dew Computing (micro-services based) ↑
- Edge Computing (Fog) ↑

Distributed Data Systems

- Distributed file systems →
- Peer to peer systems →
- Content Distribution Networks (CDNs) →
- Hadoop, key-value stores/NoSQL ↑ ↑

Distributed Pervasive Systems (Sensor networks - near future)

Examples

Proxy Servers, OwnCloud, Github/Git

Slurm/SGE

HTCondor

Globus, Unicore, OSG, ...

mobile apps/Boinc

Azure, AWS, Google, OpenNebula, OpenStack ...

Ceph, Gluster, Lustre, Azure, S3...

Bitcoin, Gnutella, Skype, GigaTribe,
Retroscore ...

Akamai

Cloudera, Hortonworks, MapR,
MongoDB/Cassandra

Kaa, Ubuntu IOT, RIOT, Node-Red, Mbed, Iotivity,
DSA, PlatformIO, TinyOS, Contiki, Nano-RK,

Distributed Computing Systems

Client/Server Systems

Is a server (hw o hw+sw) that acts as an intermediary for requests from clients seeking resources from other servers.

A client connects to the proxy server, requesting some service, (file, connection, web page, ...) available from a different server, the proxy server evaluates the conditions/rules of this and transfer or not the connection.

Proxies were invented to add structure and encapsulation to distributed systems.

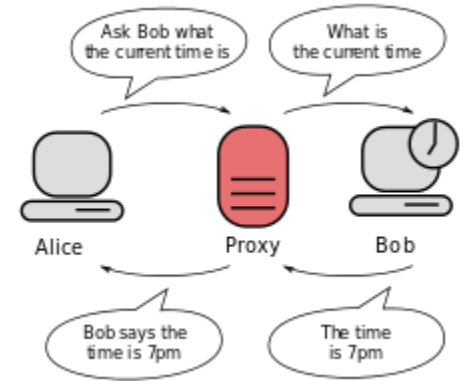
Today, most proxies are web proxies, facilitating access to content on the World Wide Web, providing anonymity and may be used to bypass IP address blocking.

A proxy server may reside on a local computer, or at various points between the client's computer and destination servers on the Internet.

A proxy server that passes requests and responses unmodified is usually called a **gateway** or sometimes a **tunneling proxy**.

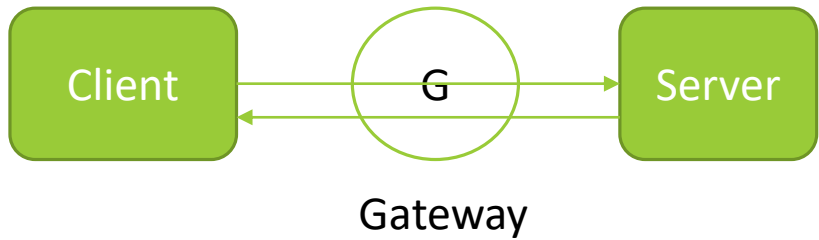
A **forward proxy** is an Internet-facing proxy used to retrieve from a wide range of sources (in most cases anywhere on the Internet).

A **reverse proxy** is usually an internal-facing proxy used as a front-end to control and protect access to a server on a private network. A reverse proxy commonly also performs tasks such as load-balancing, authentication, decryption or caching.

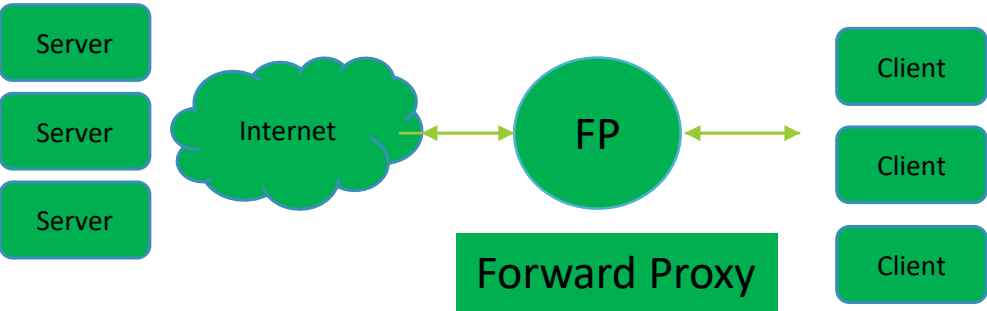
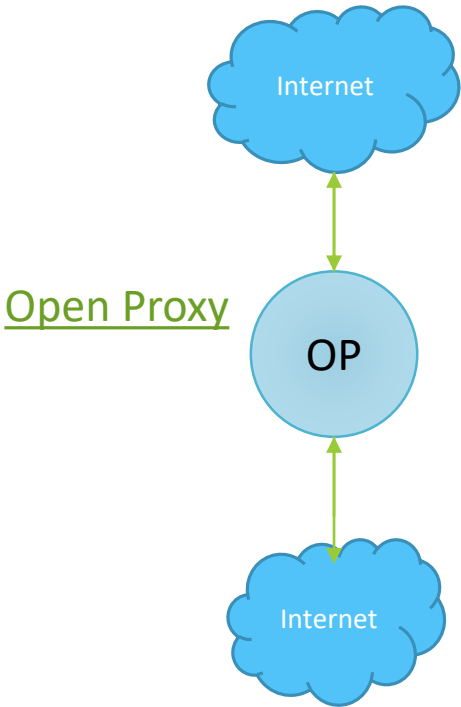
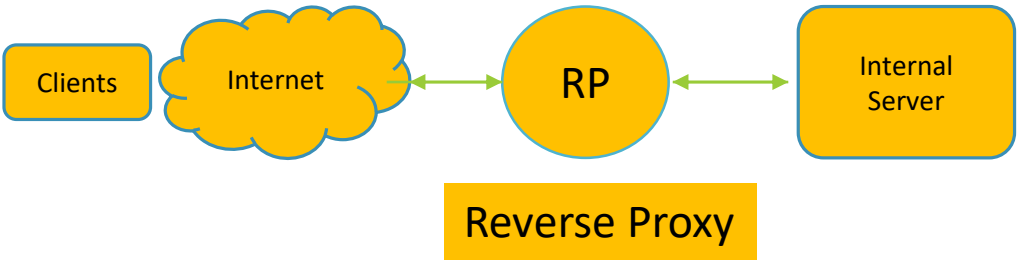


Distributed Computing systems

Client/Server Systems

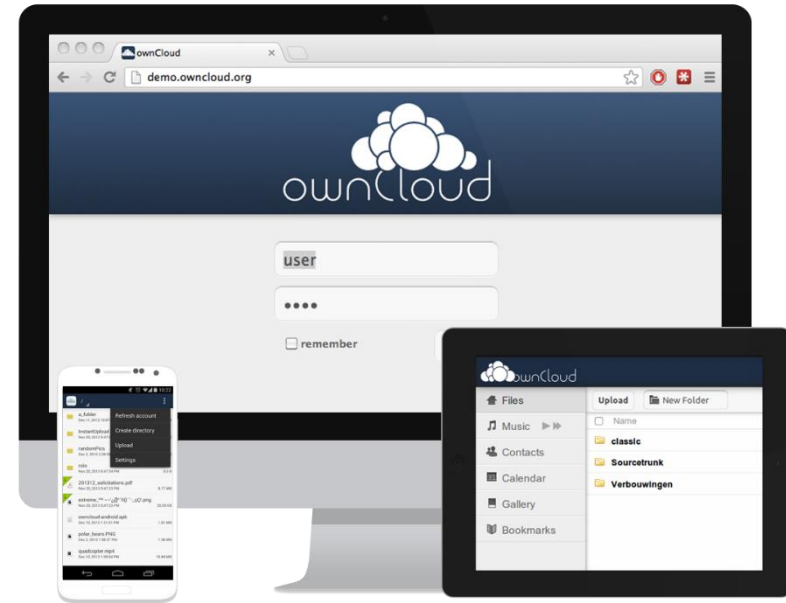


Proxy Servers



ownCloud is a suite of client–server software for creating file hosting services and using them.

ownCloud is very similar to Dropbox, but Server Edition of ownCloud is free and open-source, and thereby allowing anyone to install and operate it without charge on a private server.



Functionality: store files, contacts, calendars and more on a server. Sync Data. Share Data. Encryption (encrypt data in transit with secure https connections and encryption app to encrypt data on storage for improved security and privacy).

Supports extensions: allow it to work like Google Drive, with online document editing, calendar and contact synchronization, and more.

Its openness eschews enforced quotas on storage space or the number of connected clients, instead having hard limits (like on storage space or number of users) defined only by the physical capabilities of the server.

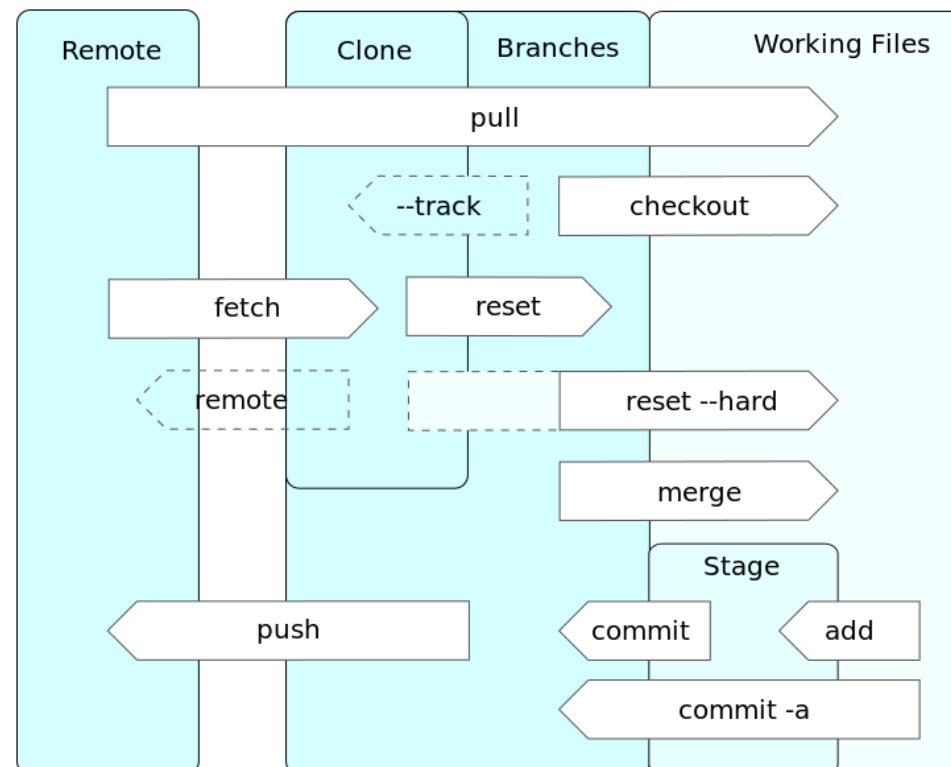
Is a version control system for tracking changes in computer files and coordinating work on those files among multiple people.

It is primarily used for source code management in software development, but it can be used to keep track of changes in any set of files.

Advantages of this distributed revision control system: speed, data integrity, and support for distributed, non-linear workflows.

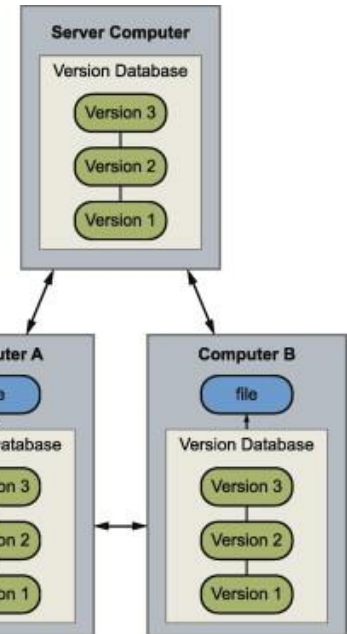
Created by Linus Torvalds (2005) for development of the Linux kernel, with other kernel developers contributing to its initial development.

Every Git directory on every computer is a full-fledged repository with complete history and full version tracking abilities, and can be work/merged with remote servers or independent of network access or a central server.



Distributed Computing Systems

Client/Server Systems



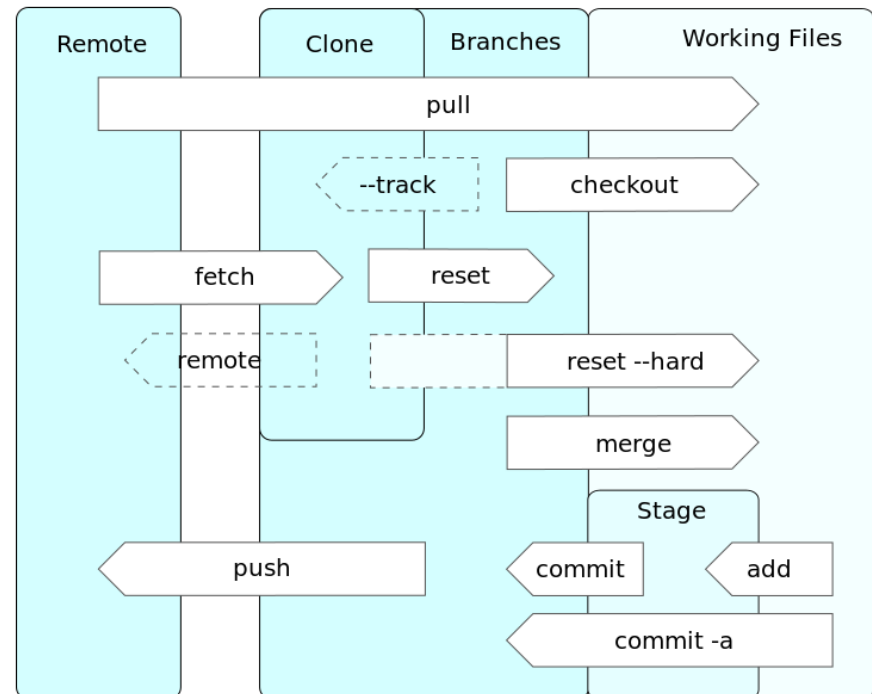
Is a version control system for tracking changes in computer files and coordinating work on those files among multiple people.

It is primarily used for source code management in software development, but it can be used to keep track of changes in any set of files.

Advantages of this distributed revision control system: speed, data integrity, and support for distributed, non-linear workflows.

Created by Linus Torvalds (2005) for development of the Linux kernel, with other kernel developers contributing to its initial development.

Every Git directory on every computer is a full-fledged repository with complete history and full version tracking abilities, and can be work/merged with remote servers or independent of network access or a central server.



Git repositories may be hosted on a local computer or on a central server

Hosting providers for Git

1. **GitHub**: provides free hosting of publicly available Git repositories. For private repositories (only visible for selected people), is necessary to pay a monthly fee.
2. **Bitbucket**: allows unlimited public and private repositories. The number of participants for a free private repository is currently limited to 5 collaborators.

GitHub is a web-based Git or version control repository and Internet hosting service.

It is mostly used for code. It offers all of the distributed version control and source code management functionality of Git as well as adding its own features.

It provides access control and several collaboration features such as bug tracking, feature requests, task management, and wikis for every project.

April 2017, GitHub reports having almost 20 million users and 57 million repositories, making it the largest host of source code in the world.

Slurm Workload Manager (Simple Linux Utility for Resource Management - SLURM), is a free and open-source job scheduler for Linux and Unix-like kernels, used by many of the world's supercomputers and computer clusters.

Provide: **First**, it allocates exclusive and/or non-exclusive access to resources (computer nodes) to users for some duration of time so they can perform work. **Second**, it provides a framework for starting, executing, and monitoring work (typically a parallel job such as MPI) on a set of allocated nodes. **Finally**, it arbitrates contention for resources by managing a queue of pending jobs.

Slurm is used on about 60% of the TOP500 supercomputers and uses a best fit algorithm based on Hilbert curve scheduling or fat tree network topology in order to optimize locality of task assignments on parallel computers.

Oracle Grid Engine, previously known as **Sun Grid Engine** (SGE) or CODINE or GRD was a grid computing computer cluster software system (also batch-queuing system).

The original **Sun Grid Engine** is an open-source project closed in 2010, but versions of the technology are still available under its original Sun License as **Son of Grid Engine**, Open Grid Scheduler and Univa Grid Engine.

Grid Engine is typically used on a computer farm or high-performance computing (HPC) cluster and is responsible for accepting, scheduling, dispatching, and managing the remote and distributed execution of large numbers of standalone, parallel or interactive user jobs.

It also manages and schedules the allocation of distributed resources such as processors, memory, disk space, and software licenses.

HTCondor is a specialized workload management system for compute-intensive jobs.

It provides a job queueing mechanism, scheduling policy, priority scheme, resource monitoring, and resource management.

Users submit their serial or parallel jobs to HTCondor, HTCondor places them into a queue, chooses when and where to run the jobs based upon a policy, carefully monitors their progress, and ultimately informs the user upon completion.

HTCondor can be used to **manage a cluster of dedicated compute nodes** (such as a "Beowulf" cluster).

HTCondor to effectively harness **wasted CPU power from otherwise idle desktop workstations**.

HTCondor can be configured to only use desktop machines where the keyboard and mouse are idle. Should HTCondor detect that a machine is no longer available (such as a key press detected), in many circumstances HTCondor is able to transparently produce a checkpoint and migrate a job to a different machine which would otherwise be idle.

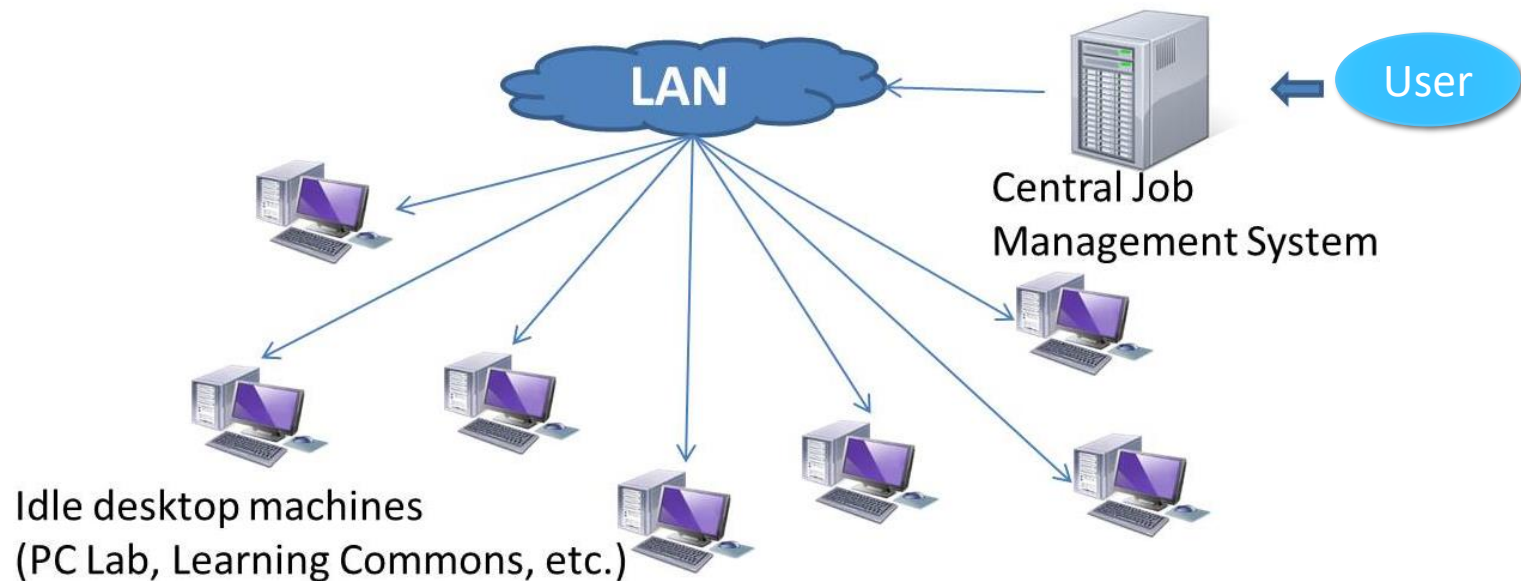
HTCondor does not require a shared file system across machines - if no shared file system is available, HTCondor can transfer the job's data files on behalf of the user, or HTCondor may be able to transparently redirect all the job's I/O requests back to the submit machine.

As a result, HTCondor can be used to seamlessly combine all of an organization's computational power into one resource.

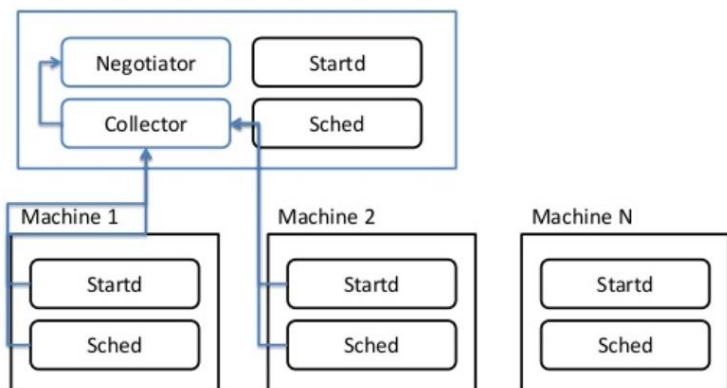
Distributed Computing Systems

Cluster Systems

HTCondor



Central Manager Machine



HTCondor implements ClassAds that works in a fashion similar to the newspaper classified advertising want-ads.

All machines in the HTCondor pool advertise their resource properties, both static and dynamic, such as available RAM memory, CPU type, CPU speed, virtual memory size, physical location, and current load average, in a resource offer ad.

A user specifies a resource request ad when submitting a job. The request defines both the required and a desired set of properties of the resource to run the job.

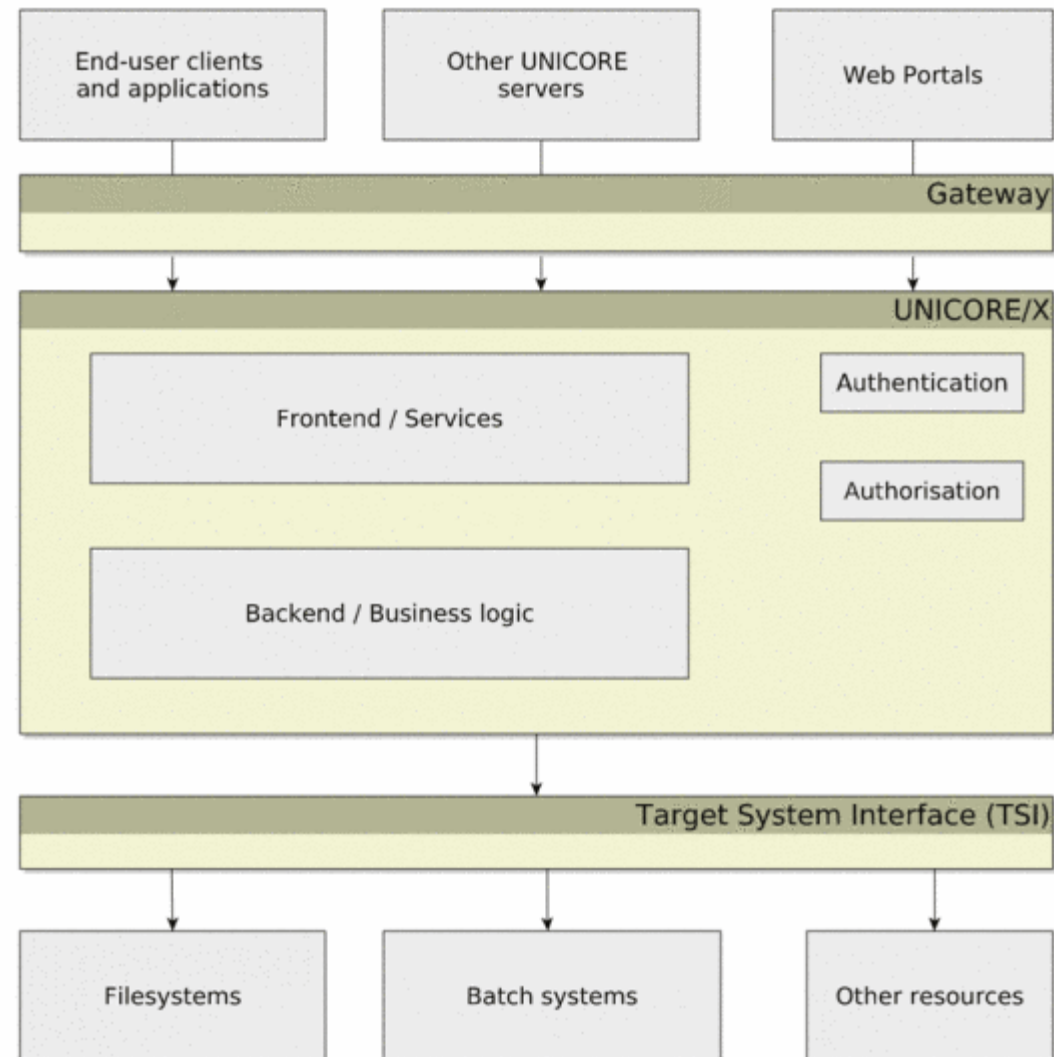
HTCondor acts as a broker by matching and ranking resource offer ads with resource request ads, making certain that all requirements in both ads are satisfied.

UNICORE (Uniform Interface to Computing Resources) offers a ready-to-run system including client and server software. UNICORE makes distributed computing and data resources available in a seamless manner.

UNICORE is used in e-infrastructures of any nature and without limitations on the type of computing resource.

Single PCs, cluster systems interconnected similar to the EGI Grid infrastructure to leadership HPC systems like the ones forming the PRACE Research Infrastructure and the Extreme Science and Engineering Discovery Environment (XSEDE) are supported as well as a variety of storage types.

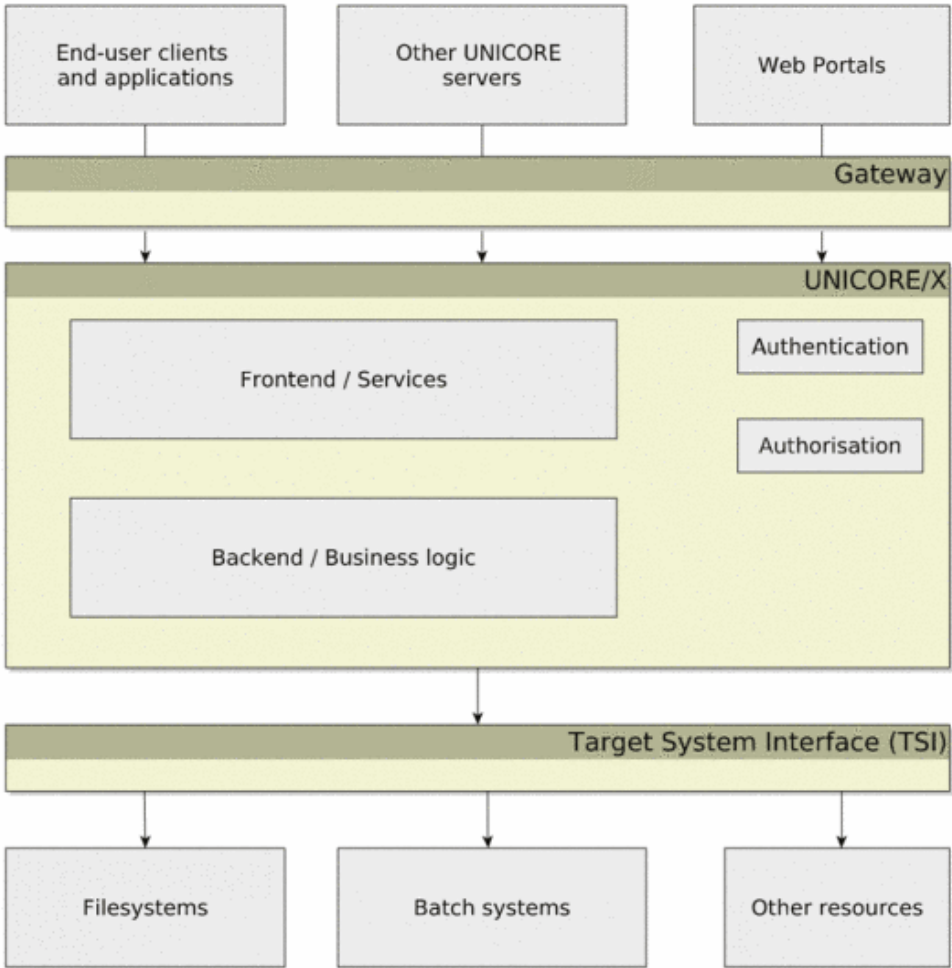
Data management functionality includes high-performance file transfer and metadata handling. UNICORE's flexible security architecture enables many usage scenarios.



Distributed Computing Systems

Grid

Unicore



HTTPS reverse proxy

Accepts the client requests that are transmitted via the gateway, authenticates the request, checks authorization and invokes the appropriate service.

Connects to the local operating system, file system and resource management (batch) system. The TSI is used to submit jobs, check job status, perform file I/O, and more. The TSI is implemented as a daemon running with administrator (root) permissions on the resource.

On a HPC machine or compute cluster, the TSI relies on an existing resource manager such as Slurm or Torque, and can be easily adapted to local settings.

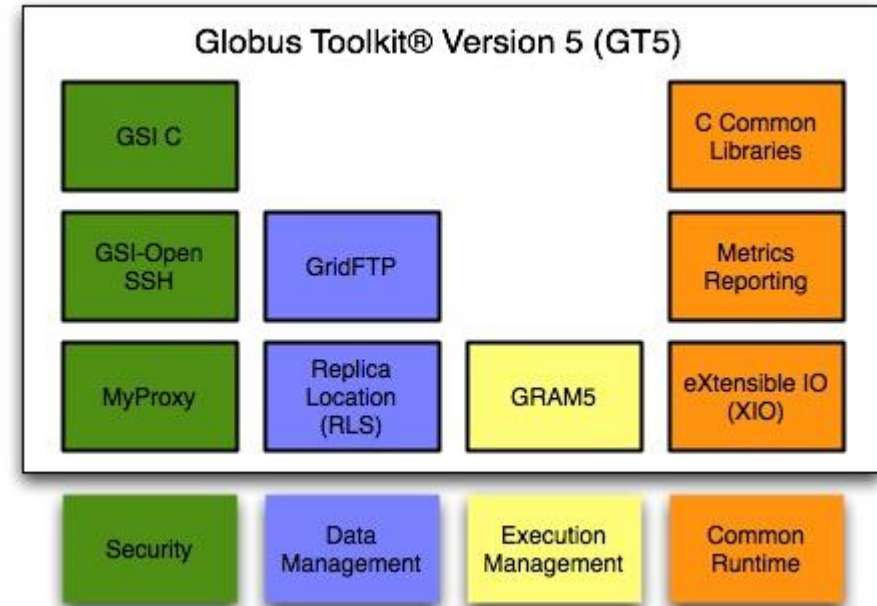
Distributed Computing Systems

Grid

Globus

The Globus Toolkit is an open source toolkit for grid computing developed and provided by the Globus Alliance.

On the 25 May 2017 it was announced that the open source support for the project will be discontinued in January 2018.



The open source Globus® Toolkit is a fundamental enabling technology for the "Grid," letting people share computing power, databases, and other tools securely online across corporate, institutional, and geographic boundaries without sacrificing local autonomy.

The toolkit includes software services and libraries for resource monitoring, discovery, and management, plus security and file management.

Distributed Computing Systems

Grid

OSG

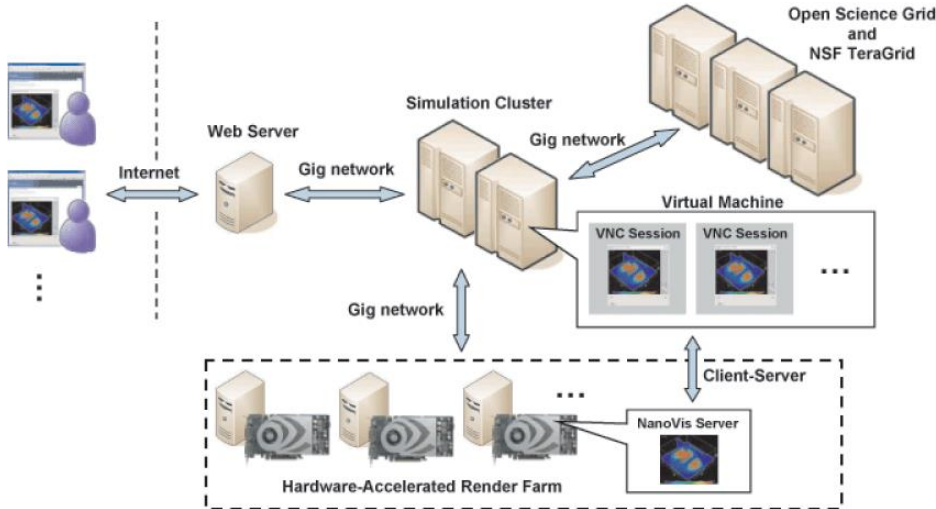
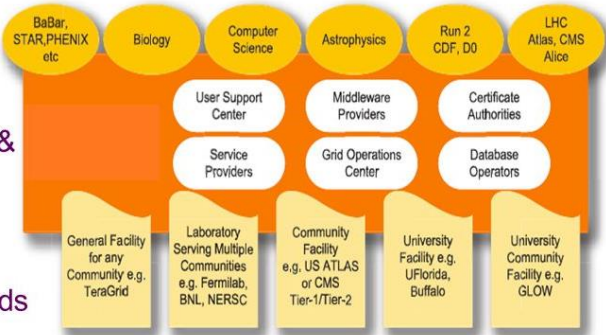
The Open Science Grid (OSG) advances science through open distributed computing. The OSG is a multi-disciplinary partnership to federate local, regional, community and national cyberinfrastructures to meet the needs of research and academic communities at all scales.

The Open Science Grid (OSG) provides provide common service and support for resource providers and scientific institutions using a distributed fabric of high throughput computational services. The OSG does not own resources but provides software and services to users and resource providers alike to enable the opportunistic usage and sharing of resources. The OSG is jointly funded by the Department of Energy and the National Science Foundation.

Virtual Organizations

Common Services & Software

Resources: Sites, Campuses, Clouds

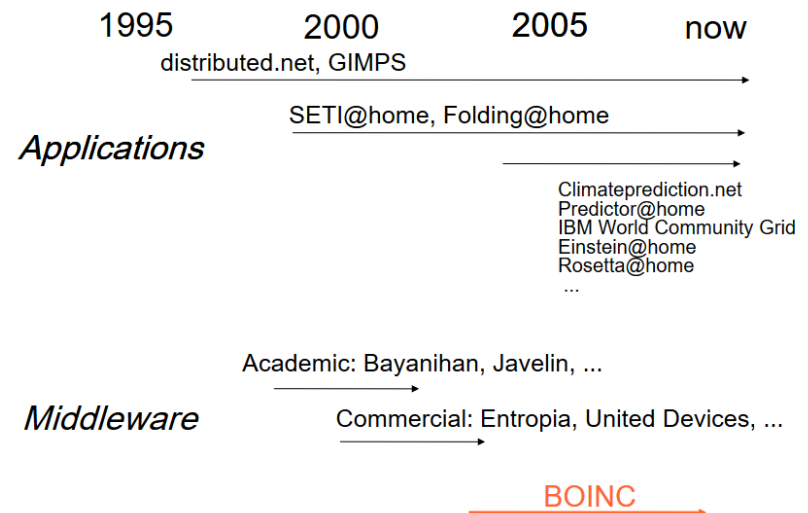
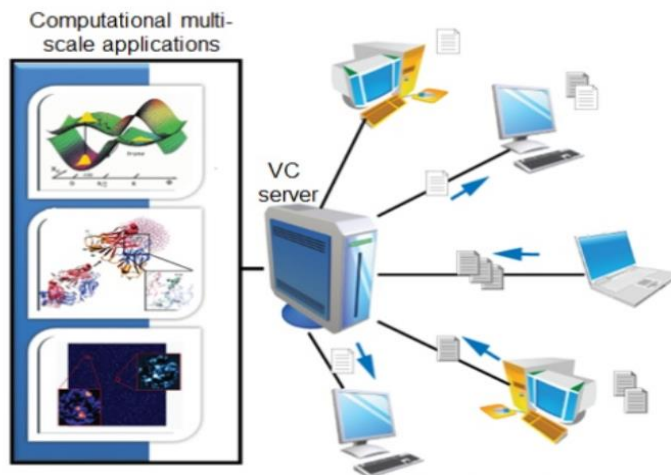


Volunteer computing is a type of distributed computing, *an arrangement in which people, so-called volunteers, provide computing resources to projects, which use the resources to do distributed computing and/or storage.*

Thus, computer owners or users donate their computing resources (such as processing power and storage) to one or more "projects".

Volunteers are frequently members of the general public in the possession of their own personal computers with a Internet connection, but also organizations can act as volunteers and provide their computing resources.

Projects in this context are mostly science-related projects executed by universities or academia in general.



Origin: The client software of the early volunteer computing projects consisted of a single program that combined the scientific computation and the distributed computing infrastructure. This monolithic architecture was inflexible.

Now: VC has moved to middleware systems that provide a distributed computing infrastructure independent from the scientific computation:

- The Berkeley Open Infrastructure for Network Computing ([BOINC](#)) is the most widely used middleware system. (clients for W, Mac OS, Linux, Android, Unix variants).
- [XtremWeb](#) is used primarily as a research tool. It is developed by a group based at the University of Paris-South.
- [JPPF enables](#) applications with large processing power requirements to be run on any number of computers, in order to reduce their processing time. This is done by splitting an application into smaller parts that can be executed simultaneously on different machines.

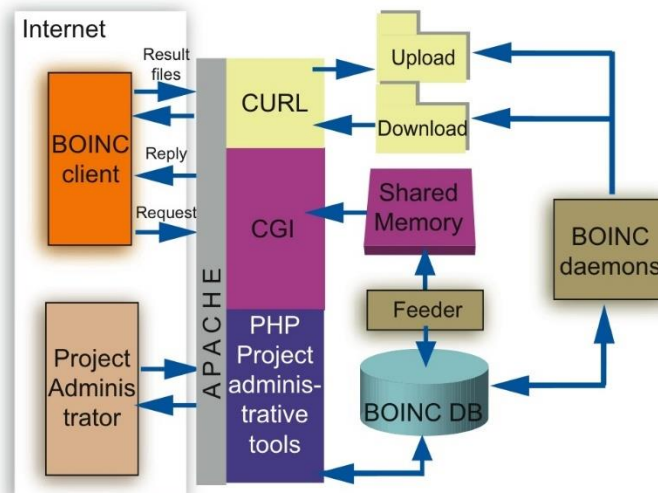
Basic structure: a client program runs on the volunteer's computer. It periodically contacts project-operated servers over the Internet, requesting jobs and reporting the results of completed jobs. This "pull" model is necessary because many volunteer computers are behind firewalls that do not allow incoming connections. The system keeps track of each user's "credit", a numerical measure of how much work that user's computers have done for the project.

The **Berkeley Open Infrastructure for Network Computing** (BOINC), an open-source middleware system, supports volunteer and grid computing.

Originally developed to support the SETI@home project, it became generalized as a platform for other distributed applications in areas as diverse as mathematics, linguistics, medicine, molecular biology, climatology, environmental science, and astrophysics, among others. BOINC aims to enable researchers to tap into the enormous processing resources of multiple personal computers around the world.

BOINC development originated with a team based at the Space Sciences Laboratory (SSL) at the University of California, Berkeley (leader of SETI@home). As a high-performance distributed computing platform, BOINC brings together about 311,742 active participants and 834,343 active computers (hosts) worldwide processing on average 16.912 PetaFLOPS as of 13 January 2017.

BOINC code runs on various operating systems (W, macOS, Android, Linux & FreeBSD).



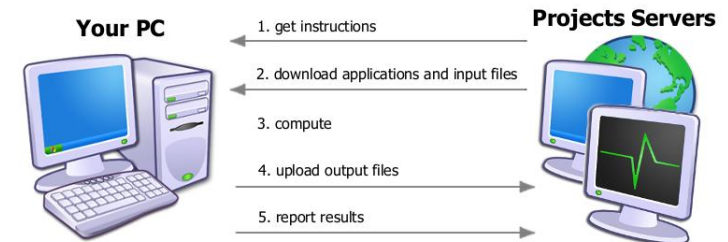
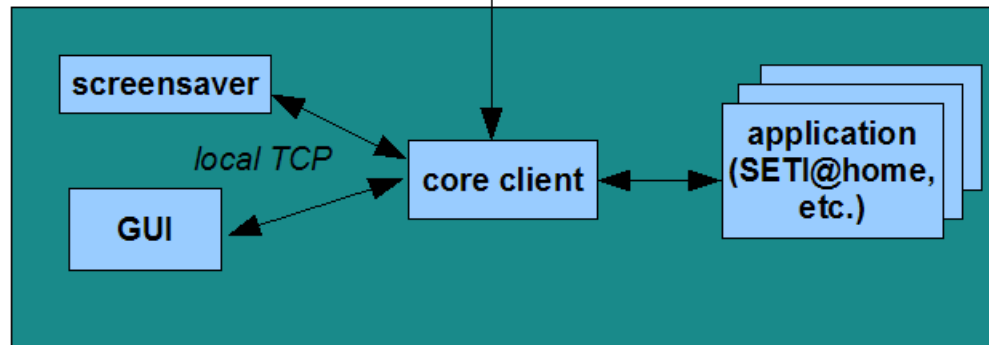
Distributed Computing Systems

Internet Computing/Volunteer Computing

The schedulers and data server programs are installed on computers owned and managed by the projects to which you will donate time on your computer.

schedulers, data servers

HTTP



The **core client** communicates with external servers via the HTTP communications protocol to get and report work. The core client runs and controls applications.

Applications are the programs that do scientific computing. Several of them may run at the same time on a computer with more than one CPU.

The **GUI**, provides a graphical interface that lets you control the core client (to suspend and resume applications). The GUI communicates with the core client by a TCP connection.

The **screensaver** runs when you're away from the computer. It communicates with the core client by local TCP, instructing it to tell one of the applications to generate screensaver graphics. Note that not all projects provide screensaver graphics.

CloudSigma 

 codeanywhere

{cloud} Codenvy


tutorialspoint
SIMPLYEASYLEARNING

OpenNebula.org


openstack™

oVirt

Public IaaS (AWS, Google Cloud, Azure,...)

- **(pseudo) IaaS**

Private IaaS (eucalyptus, OpenQRM, ...)



Public PaaS (AWS, Google Cloud, Azure, OpenShift, ...)



Private PaaS: Cloudify, Dokku, Tsuru, Flynn, ...



Public/Private SaaS: Drupal, OwnCloud, ...

Dew computing is a new computing paradigm appeared after the widely acceptance of cloud computing.

Two key features: first, local computers (desktops, laptops, tablets, and smart phones) provide rich micro-services independent of cloud services; second, these micro services inherently collaborate with cloud services.

Dew computing concerns the distribution of workloads between cloud servers and local computers, and its focus is the software organization of local computers. The goal of dew computing is to fully realize the potentials of local computers and cloud services.

Fog computing is a scenario where a huge number of heterogeneous (wireless and sometimes autonomous) ubiquitous and decentralised devices communicate and potentially cooperate among them and with the network to perform storage and processing tasks without the intervention of third-parties.

These tasks can be for supporting basic network functions or new services and applications that run in a sandboxed environment. Users leasing part of their devices to host these services get incentives for doing so.

Two categories: first category is computer, including desktop, laptop, hand-held, and all other kinds of computers. The second category is automation device, including sensors, controllers, chips, disks, network devices, and so on.

Computers can be directly operated and programmed by human users. Automation devices usually have computing power, sometimes they have pretty strong computing power, but they are not operated directly by human users when they are in normal running mode.

Fog Computing mainly involves automation devices while Dew Computing mainly involves computers.



ioAuthoring: let's you create layouts of microservices called tracks, that run on ioFog. It's as easy as drag & drop and changeable without programming. Use it for development, testing and production, as you can change where and when it is running independently from the machine it is running on. Perfect for scalable projects.

ioFog: By installing ioFog to a server or Raspberry Pi, you create an agent that is a piece of the IOTRACKS which enables you to run microservices on it dynamically.

Microservices: Use pre-existing microservices and create new ones. Use them to connect sensors and devices, integrate them with existing systems, do processing in real-time. Works for lightest filtering to deepest analytics and machine learning. Write in any language or framework - bring together multiple devices for interoperability.



Kaa is a multi-purpose middleware platform for the IoT that allows building complete end-to-end IoT solutions, connected applications, and smart products.

The Kaa platform provides an open, feature-rich toolkit for the IoT product development and thus dramatically reduces associated cost, risks, and time-to-market.

For a quick start, Kaa offers a set of out-of-the-box enterprise-grade IoT features that can be easily plugged in and used to implement a large majority of the IoT use cases.



PlatformIO is an open source ecosystem for IoT development
Cross-platform IDE and unified debugger. Remote unit testing and firmware updates

The friendly Operating System for the Internet of Things



The Arm Mbed IoT Device Platform provides the operating system, cloud services, tools and developer ecosystem to make the creation and deployment of commercial, standards-based IoT solutions possible at scale.



Kaa is a multi-purpose middleware platform for the IoT that allows building complete end-to-end IoT solutions, connected applications, and smart products.

The Kaa platform provides an open, feature-rich toolkit for the IoT product development and thus dramatically reduces associated cost, risks, and time-to-market.

For a quick start, Kaa offers a set of out-of-the-box enterprise-grade IoT features that can be easily plugged in and used to implement a large majority of the IoT use cases.



PlatformIO is an open source ecosystem for IoT development Cross-platform IDE and unified debugger. Remote unit testing and firmware updates



The friendly Operating System for the Internet of Things

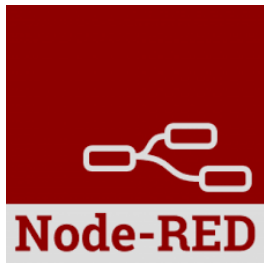


The Arm Mbed IoT Device Platform provides the operating system, cloud services, tools and developer ecosystem to make the creation and deployment of commercial, standards-based IoT solutions possible at scale.



Ubuntu for the Internet of Things

From home control to drones, robots and industrial systems, Ubuntu Core provides robust security, app stores and reliable updates. Ubuntu makes development easy, and snap packages make Ubuntu Core secure and reliable for widely distributed devices.



Node-RED is a programming tool for wiring together hardware devices, APIs and online services in new and interesting ways.

It provides a browser-based editor that makes it easy to wire together flows using the wide range of nodes in the palette that can be deployed to its runtime in a single-click.



IoTivity is an open source software framework enabling seamless device-to-device connectivity to address the emerging needs of the Internet of Things.



Distributed Services Architecture (DSA), is an open source IoT platform that facilitates device inter-communication, logic and applications at every layer of the Internet of Things infrastructure. The objective is to unify the disparate devices, services and applications into a structured and adaptable real-time data model.



TinyOS is an "operating system" designed for low-power wireless embedded systems. Fundamentally, it is a work scheduler and a collection of drivers for microcontrollers and other ICs commonly used in wireless embedded platforms.

Contiki

Contiki is an open source operating system for the Internet of Things. Contiki connects tiny low-cost, low-power microcontrollers to the Internet. Contiki is a powerful toolbox for building complex wireless systems.

Nano-RK

Nano-RK is a fully preemptive reservation-based real-time operating system (RTOS) from Carnegie Mellon University with multi-hop networking support for use in wireless sensor networks. Nano-RK currently runs on the FireFly Sensor Networking Platform as well as the MicaZ motes. It includes a light-weight embedded resource kernel (RK) with rich functionality and timing support using less than 2KB of RAM and 18KB of ROM.

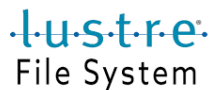
Distributed File Systems



Ceph is a free-software storage platform, implements object storage on a single distributed computer cluster, and provides interfaces for object-, block- and file-level storage. Ceph aims primarily for completely distributed operation without a single point of failure, scalable to the exabyte level, and freely available.



GlusterFS is a scale-out network-attached storage file system. It has found applications including cloud computing, streaming media services, and content delivery networks. GlusterFS was developed originally by Gluster, Inc. (acquired by Red Hat in 2011)



Lustre is a type of parallel distributed file system, generally used for large-scale cluster computing. It provides high performance file systems for computer clusters ranging in size from small workgroup clusters to large-scale, multi-site clusters.



Microsoft Azure Storage is a cloud service that provides storage that is highly available, secure, durable, scalable, and redundant. MAS consists of three data services: Blob storage, File storage, and Queue storage.



Amazon S3 is object storage built to store and retrieve any amount of data from anywhere – web sites and mobile apps, corporate applications, and data from IoT sensors or devices. It is designed to deliver 99.999999999% durability, and stores data for millions of applications used by market leaders in every industry.

Peer to Peer Systems



Bitcoin is a worldwide cryptocurrency and digital payment system called the first decentralized digital currency, as the system works without a central repository or single administrator. It was invented by an unknown programmer, or a group of programmers, under the name Satoshi Nakamoto and released as open-source software in 2009. The system is peer-to-peer, and transactions take place between users directly, without an intermediary. These transactions are verified by network nodes and recorded in a public distributed ledger called a blockchain.



Gnutella is a large peer-to-peer network (2000). It was the first decentralized peer-to-peer network of its kind, leading to other, later networks adopting the model. 3 million nodes by January 2006.



Skype is a telecommunications application software product that specializes in providing video chat and voice calls from computers, tablets, and mobile devices via the Internet to other devices or telephones/smartphones. Skype originally featured a hybrid peer-to-peer and client–server system. 2012: Skype has been powered entirely by Microsoft-operated supernodes. 2016/17, redesigned its Skype clients in a way that transitioned Skype from peer-to-peer service to a centralized Azure service and adjusted the user interfaces of apps to make text-based messaging more prominent than voice calling.



GigaTribe is a peer-to-peer file sharing network (2005) that offers free and paid versions; with the paid version users may restrict access to their encrypted files to a group of trusted friends.



Retroshare creates encrypted connections between the user and their friends to create a network of computers, and provide various distributed services on top of it: forums, channels, asynchronous messaging,... Retroshare is fully decentralized, and designed to provide maximum security and anonymity to its users beyond direct friends. The software is entirely Open-Source and free.

Content Distribution Networks (CDNs)



The Akamai Intelligent Platform is a distributed cloud computing platform that operates worldwide. It is a network of over 216,000 servers deployed in more than 120 countries. These servers reside in more than 1,500 of the world's networks gathering real-time information about traffic, congestion, and trouble spots. Each Akamai server is equipped with proprietary software that uses complex algorithms to process requests from nearby users, and then serve the requested content.

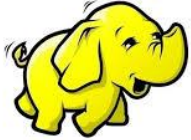
Content delivery process: The content delivery process begins with a user submitting a request to a browser. When a user enters a URL, a DNS request is triggered and an IP address is retrieved. With the IP address, the browser can then contact a web server directly for subsequent requests. In a content delivery network structure, the domain name of the URL is translated by the mapping system into the IP address of an edge server to serve the content to the user.

Akamai delivers web content over its Intelligent Platform by transparently mirroring elements such as HTML, CSS, software downloads, and media objects from customers' servers. The Akamai server is automatically picked depending on the type of content and the user's network location. Receiving content from an Akamai server close to the user allows for faster download times and less vulnerability to network congestion. Akamai claims to provide better scalability by delivering the content over the last-mile from servers close to end-users, avoiding the middle-mile bottleneck of the Internet.

In addition to using Akamai's own servers, Akamai delivers content from other end-users' computers, in a form of peer-to-peer networking.

Big Data Infrastructure

hadoop



Apache Hadoop is an open-source software framework used for distributed storage and processing of dataset of big data using the MapReduce programming model. It consists of computer clusters built from commodity hardware. All the modules in Hadoop are designed with a fundamental assumption that hardware failures are common occurrences and should be automatically handled by the framework.



Apache Spark provides programmers with an application programming interface centred on a data structure called the resilient distributed dataset (RDD), a read-only multiset of data items distributed over a cluster of machines, that is maintained in a fault-tolerant way. It was developed in response to limitations in the MapReduce cluster computing paradigm, which forces a particular linear dataflow structure on distributed programs: MapReduce programs read input data from disk, map a function across the data, reduce the results of the map, and store reduction results on disk. Spark's RDDs function as a working set for distributed programs that offers a (deliberately) restricted form of distributed shared memory.

**Not
Only SQL**

A NoSQL (non SQL or non relational) database provides a mechanism for storage and retrieval of data that is modelled in means other than the tabular relations used in relational databases. Such databases have existed since the late 1960s, but did not obtain the "NoSQL" moniker until a surge of popularity in the early twenty-first century, triggered by the needs of Web 2.0 companies such as Facebook, Google, and Amazon.com. NoSQL databases are increasingly used in big data and real-time web applications.



MongoDB is a free and open-source cross-platform document-oriented database program. Classified as a NoSQL database program, MongoDB uses JSON-like documents with schemas. MongoDB is developed by MongoDB Inc., and is published under a combination of the GNU Affero General Public License and the Apache License.

Big Data Infrastructure



Cloudera's open-source Apache Hadoop distribution, CDH (Cloudera Distribution Including Apache Hadoop), targets enterprise-class deployments of that technology. Cloudera says that more than 50% of its engineering output is donated upstream to the various Apache-licensed open source projects (Apache Hive, Apache Avro, Apache HBase, and so on) that combine to form the Hadoop platform. Cloudera is also a sponsor of the Apache Software Foundation.

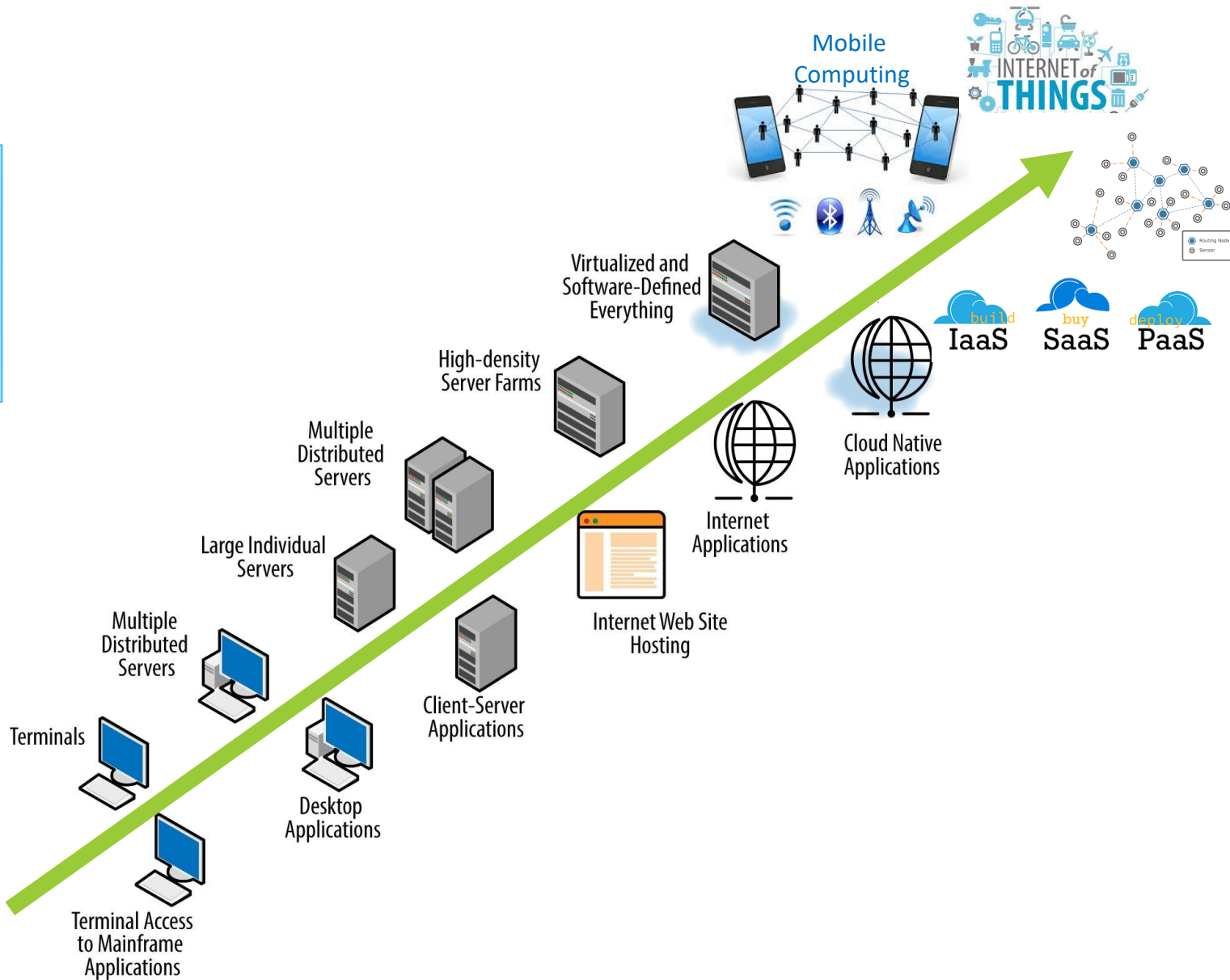


Hortonworks is a big data software company based in Santa Clara, California. The company develops and supports Apache Hadoop, for the distributed processing of large data sets across computer clusters.



MapR Technologies, Inc. is an enterprise software company headquartered in California. MapR provides global access to a wide variety of data sources from a single cluster, including big data workloads such as Apache Hadoop and Apache Spark, a distributed file system, a distributed file system, a multi-model database management system, and event streaming. Combining analytics in real-time with operational applications, its technology runs on both commodity hardware and public cloud computing services.

Evolution



Answers & comments

Conclusions

[Distributed Systems: Principles and Paradigms](#), Andrew S. Tanenbaum and Maarten Van Steen, 2006. (Second Edition)

Information Systems Management in Practice. B. McNurlin, R. Sprague, T. Bui. Pearson; 8 edition. 2008

[Distributed Systems: The Overall Architecture](#). Peter Lo. 2008. Hong Kong Polytechnic University. U51020.

[The NIST Definition of Cloud Computing](#). NIST.

[Operating Systems. Concepts & Theory](#). Tong Lai Yu (2010). California State University.

[Application Architecture Guide](#), 2nd Edition. Microsoft. 2009.

[Software architecture](#). Wikipedia.

[The Evolution of Distributed Systems Towards Microservices Architecture](#). Tasneem Salah, M. Jamal Zemerly, Chan Yeob Yeun, Mahmoud Al-Qutayri, Yousof Al-Hammadi. 20016.

[HTCondor](#). User/Administration Manual. 2017.

[Unicore](#). 2017 Software: <https://sourceforge.net/projects/unicore/> [UNICORE 7 — Middleware services for distributed and federated computing](#). IEEE. 2016.

[Open Science Grid](#). 2016. [Globus Toolkit](#). 2017, [Boinc](#) 2017, [Boinc Virtual Machine](#) 2017, [Boinc Docker](#) 2017.

Tots els materials, enllaços, imatges, formats, protocols i informació utilitzada en aquesta presentació són propietat dels seus respectius autors, i es mostren amb fins didàctics i sense ànim de lucre, excepte aquells que sota llicències d'ús o distribució lliure cedides i/o publicades a aquest efecte. (Articles 32-37 de la Llei 23/2006, Spain).

Sota cap motiu s'han de publicar o reproduir amb altre objectiu pel qual ha estat fet ja que es pot incórrer en activitats delictives i/o punibles.