CSIE52400/CSIEM0140 Distributed Systems

Lecture 01 Introduction

Shiow-yang Wu (吳秀陽)
Department of Computer Science and Information Engineering
National Dong Hwa University

About History

- "Study the past if you would define the future."
  — **Confucius**

- "If you don't know history, then you don't know anything. You are a leaf that doesn't know it is part of a tree. "
  — **Michael Crichton**

- "History will be kind to me for I intend to write it."
  — **Winston S. Churchill**

Brief History of the Net

- 1948: First stored-program computer (**Manchester Small-Scale Experimental Machine**, **SSEM**)
- 1969~70: **ARPANet**
- 1971: First **email** sent over ARPANet
- 1972: **TCP/IP**
- 1975: **Microsoft** (Paul Allen and Bill Gates)
- 1981: **IBM P**ersonal **C**omputer (Beginning of desktop computing)
- 1982: **L**ocal **A**rea **N**etworks (First widespread distributed systems)
- 1984: **D**omain **N**ame **S**ystem (**DNS**) created
- 1988: First **Internet worm** (Robert Tappan Morris) disrupting 10% computers on the Internet)
- 1989: ARPAnet ends. **WWW** (Tim Berners-Lee) begins

Brief History (cont.)

- 1990: First **IoT** device (a toaster that could be turned on/off over the Internet)
- 1991: 1st Web server came online
- 1992: The **Internet Society** founded
- 1993: **Mosaic** (1st graphical Web browser)
- 1994: **Yahoo** (1st Web directory service by David Filo and Jerry Yang)
- 1994: **Netscape** (the world gets a popular navigator)
- 1995: **Java** (James Gosling) and **JavaScript** (Brendan Eich)
- 1995: Commercial Internet (**Amazon**, **Echobay**(**eBay**), **MSN**)
- 1995: **Match.com** (1st dating site)
- 1996: **HoTMaiL** (1st free web-based email provider)
- 1997: **WiFi** released (the world becomes wireless)
- 1997: **Weblog** (Jorn Barger), **Blog** (1999, by Peter Merholz)
- 1998: **Google** (Larry Page and Sergey Brin)
- 1998: **IPv6** (128-bit addresses) protocol published
- 1999: The **Internet of Things** term coined by Kevin Ashton
- 1999: **Napster** (1st wide spread P2P file sharing service)

Brief History (cont.)

- 2000: **Gnutella**, **Freenet** (P2P networks)
- 2000: **DDoS** attach (by a high school student Mike "MafiaBoy Calce)
- 2001: **BitTorrent** (most popular P2P service)
- 2001: **Wikipedia** (a landmark Web2.0 service)
- 2002: **Blog** becomes popular
- 2003: **Skype** (VoIP service by Niklas Zennström and Janus Friis)
- 2003: **MySpace** (becomes the most popular social network)
- 2003: **iTunes** launched by Apple (record store dies a slow death)
- 2004: **Facebook** founded (Mark Zuckerberg)
- 2004: **Gmail** announced on April Fools Day
- 2005: **YouTube** (Chad Hurley, Steve Chen, and Jawed Karim)
- 2005: **Google Maps** & **Google Earth** (a virtual globe)
- 2005: Roger Mougalas coined the term **Big Data**
- 2005: **Hadoop** created by Yahoo! on top of Google MapReduce

Brief History (cont.)

- 2006: **Twitter** founded
- 2006: Amazon launched **Amazon Web Service** (**AWS**) for utility computing (the cloud computing era)
- 2007: **Google Street View**
- 2007: **Google surpasses Microsoft** (most valuable and visited)
- 2007: **Netflix** begins streaming
- 2007: **iPhone** introduced by Apple
- 2008: **Mobile devices surpass PCs** to become the major Internet access devices (mobile computing)
- 2008: **Google App Engine** (cloud computing platform)
- 2008: **Dropbox** (cloud storage)
- 2008: **Bitcoin** (cryptocurrency)
- 2009: **Bing** search engine (Microsoft strikes back)
- 2009: **Google Docs** (cloud office suit)
- 2009: **Facebook** (most used social media platform)

Brief History (cont.)

- 2008~2009: **Internet of Things** was born (more "things or objects" were connected to the Internet than people)
- 2010: **HBO** starts streaming
- 2010: **OnLive** (cloud gaming)
- 2010: More people visited Facebook than Google
- 2010: **iPad** released by Apple (multi-touch, gesture interface, tablet era)
- 2010: **China** announce major investments in IoT
- 2010: Eric Schmidt (chairman of Google) speech: as much data is created every two days as was created from the beginning of human civilization to 2003
- 2010: **Instagram** launched (photo/video sharing)

Brief History (cont.)

- 2011: **Google+** social networking
- 2012: **IPv6** public launch
- 2012: **Big Data** paradigm takes off
- 2013: 1 billion views a day on **YouTube mobile**
- 2014: **Mobile Internet** use overtakes desktop
- 2014: The year of the **IoT** (3.7 billion connected things)
- 2014: Google **Android Wear** (wearable computing)
- 2014: **Moto 360** (smartwatch) by Motorola
- 2015: **Apple Watch** shipped
- 2015: Google developing **IoT OS**
- 2015: Huawei launches IoT platform

Brief History (cont.)

- 2016: **AlphaGo** beats Lee Sedol (a Go world champ)
- 2016: **VR/AR** take off
- 2016: **Pokémon Go** launched
- 2016: **TikTok** dances to the top
- 2017: **AI & Deep learning** everywhere
- 2018: **Facebook–Cambridge Analytica data scandal**
- 2019: **@world_record_egg** became the most-liked post
- 2019: **5G** rolls out
- 2019: Google claim Quantum Supremacy with quantum computer Sycamore solving a problem in 200 seconds which would take 10,000 years on the fastest traditional computer

COVID-19 Pandemic

- **Dec 2019**: a novel virus was identified in Wuhan, China and quickly spread around the world (later named COVID-19).
- **30 Jan 2020**: WHO declared a **P**ublic **H**ealth **E**mergency of **I**nternational **C**oncern(**PHEIC**, 國際關注的公共衛生緊急事件)
- **11 Mar 2021**: WHO declared a pandemic(大流行)
- As of **11 Feb 2024**: > 774 million confirmed cases, > 7 million deaths (WHO COVID-19 dashboard)
- One of the **deadliest** pandemics in human history
- COVID-19 changes everything.
- **Working from home** becomes the new normal.
- We rely on the Internet and distributed services even more!

Brief History (cont.)

- 2020: ANDROID used by 75% of world's mobile devices
- 2021: **Non-Fungible Tokens** (**NFTs**) became popular
- 2021: **Meta** (the new Facebook) and **Metaverse**
- 2022: **ChatGPT** launches by OpenAI
- 2019~2024: **Starlink** (SpaceX started launching Starlink satellites in 2019. 5,289 satellites as of Jan 2024)
- 2017~2024: **Deepfake** makes the whole world confuse and in risk
- 2022~2024: **Generative AI** and **AI-generated content** sweep the world
- 2024: **Neuralink brain chip**'s 1st patient able to control mouse through thinking

The Trend

- Centralized systems
- Desktop computing
- Distributed computing
- Internet/Web computing
- Cloud computing
- Mobile computing
- Social computing
- Computing Everywhere (P2P, Web, Grid, Cloud, Jungle, Fog, Edge computing)

Definition of Distributed Systems

- A distributed system is a collection of autonomous computing elements (nodes) and related software that appears to its users as a single coherent system.
- How to connect independent computers?

⇒ Networks and protocols

- How to make them appear as a single system?

⇒ Distributed software

- Characteristics:
  - Prone to failure (failure rate)
  - Bandwidth and latency
  - Collaboration

Distributed System Model

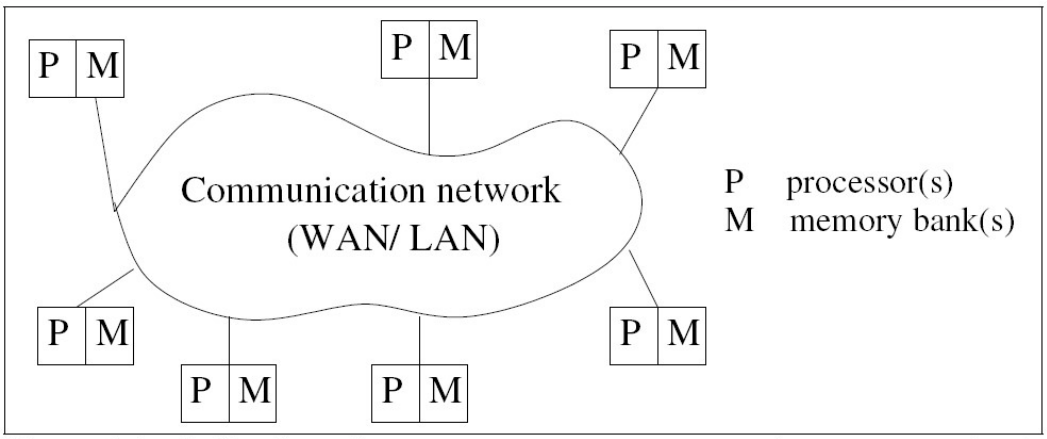


Figure 1.1: A distributed system connects processors by a communication network.

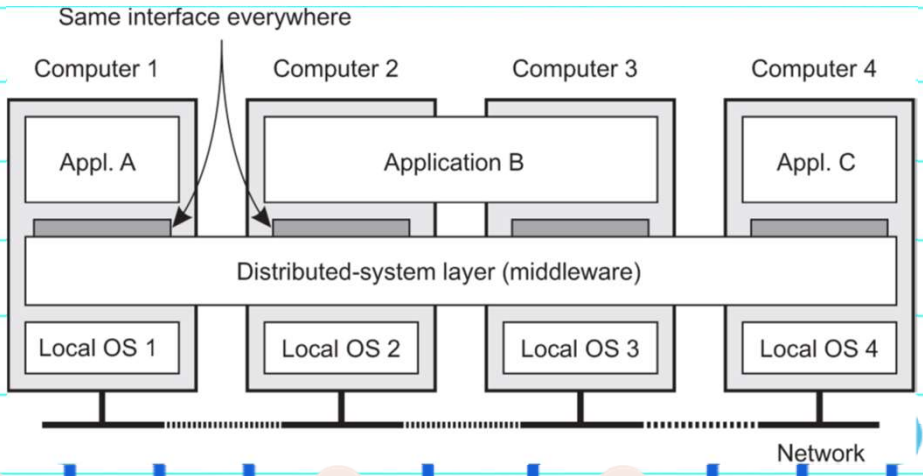
Distributed System Definition

- A distributed system is one in which hardware and software components located at networked computers communicate and coordinate their actions only by passing messages.

Distributed Applications			
Middleware			
Machine	Machine	……	Machine
Passing messages			
Networks			

Middleware

- The OS of distributed systems
- Contain commonly used components and functions



Relation between Software Components

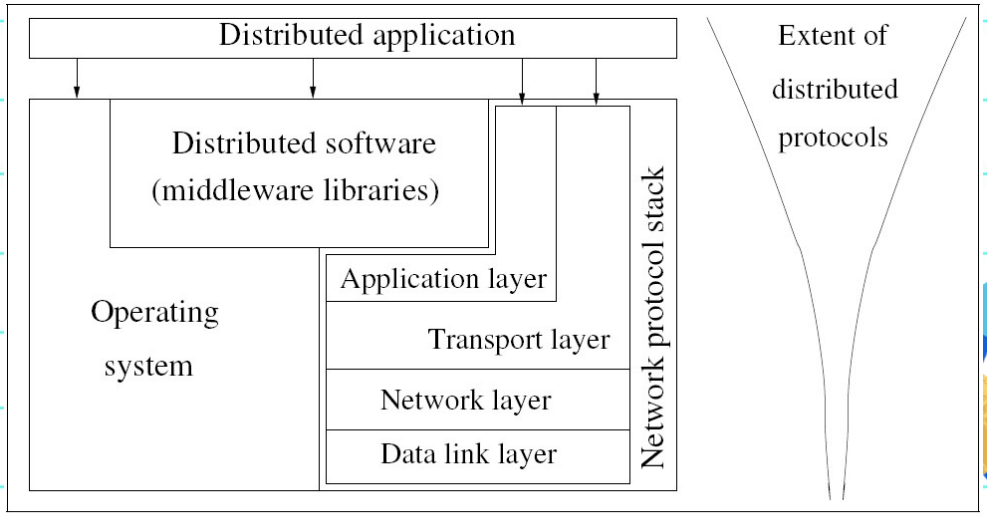


Figure 1.2: Interaction of the software components at each process.

Yet Another Definition of Distributed System

- "You know you have one when the crash of a computer you've never heard of stops you from getting any work done."

  — Leslie Lamport

Distributed Systems are Complex

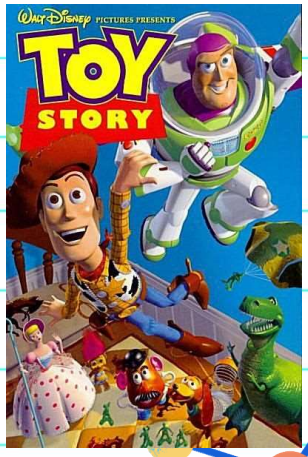
- Architecture: common organizations
- Process: what kind of processes, and their relationships
- Communication: facilities for exchanging data
- Coordination: application-independent algorithms
- Naming: how do you identify resources?
- Consistency and replication: performance requires of data, which need to be the same
- Fault tolerance: keep running in the presence of partial failures
- Security: ensure authorized access to resources
- Why do we need them?

Motivations

- Inherently distributed computation/systems
- Sharing of resources, information and services
- Access to remote resources
- Increased performance/cost ratio
- Improving availability, reliability, fault tolerance, and scalability
- Modularity and incremental expandability
- Mobility
- Pervasiveness

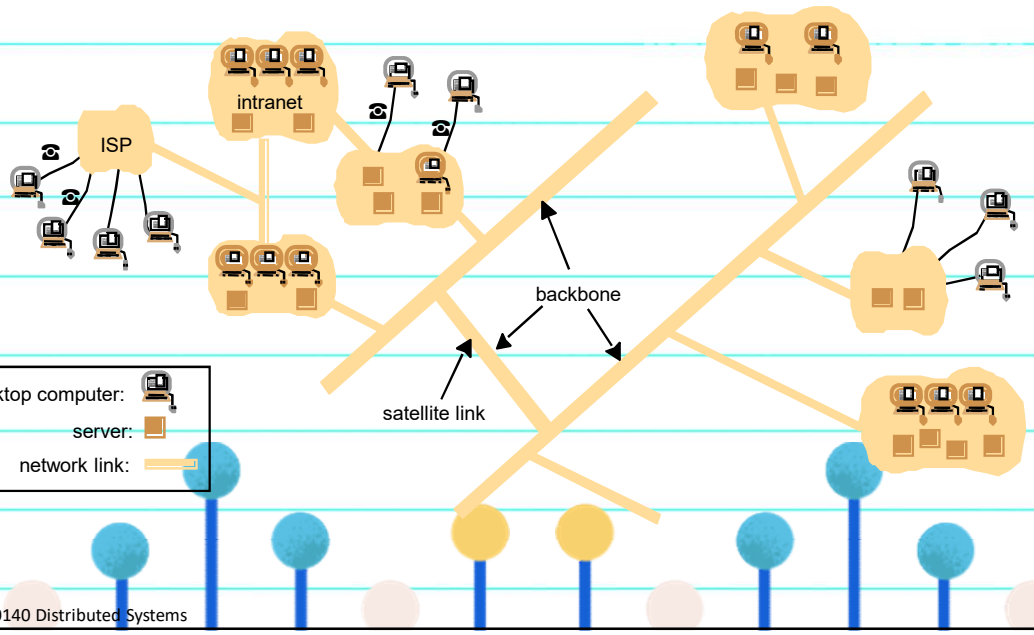
Real-life Example

- The film Toy Story (1995)
  - 1st feature-length computer animated movie
  - A 77-minute long film
  - Used 117 Sun SparcStations
  - To draw (or "render") all of the 114,000 frames

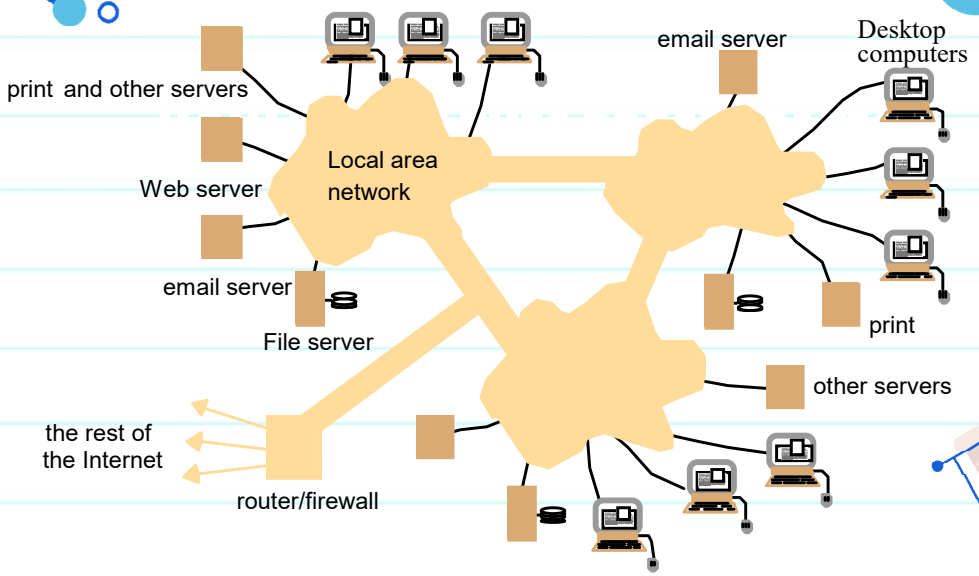


- One SparcStation would have taken **43** years of non-stop computing to do this (at that time) !!

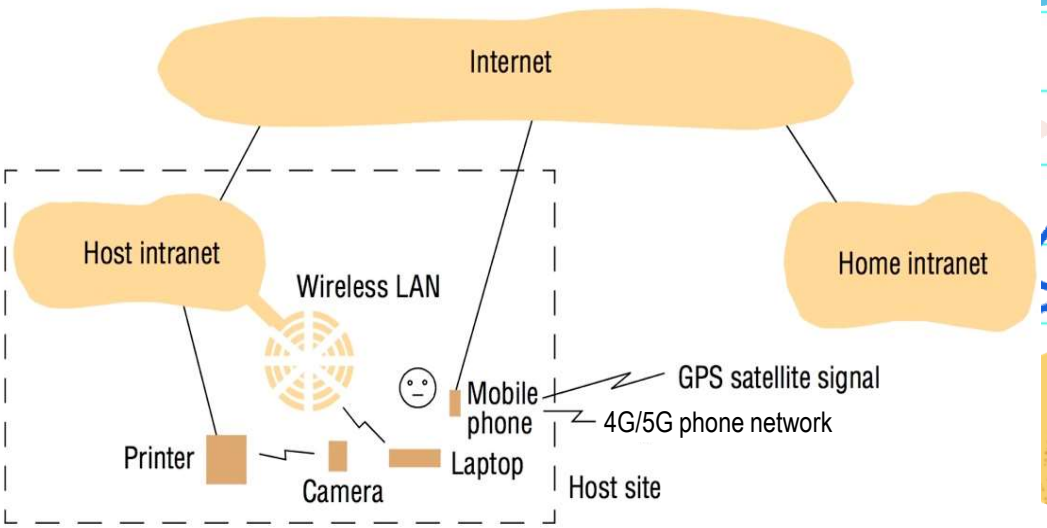
Example: Internet



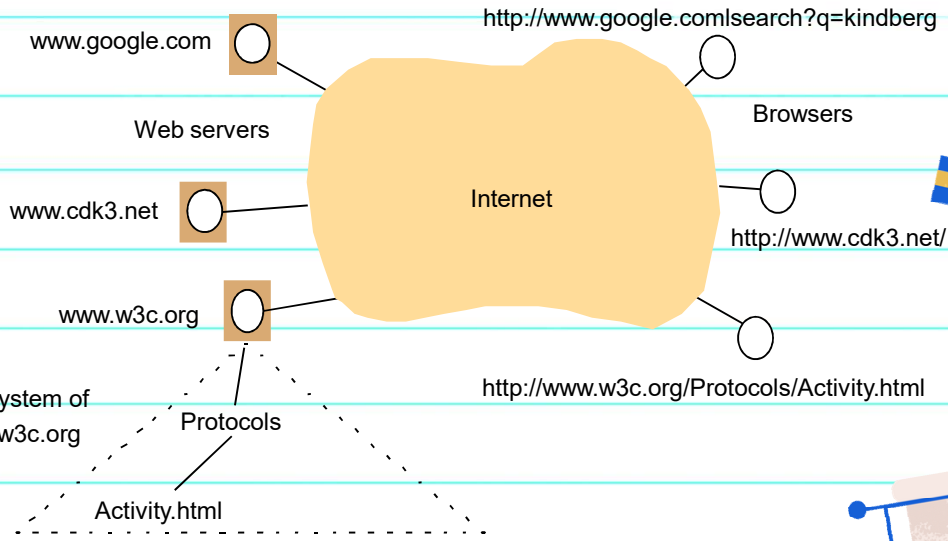
Example: A Typical Intranet



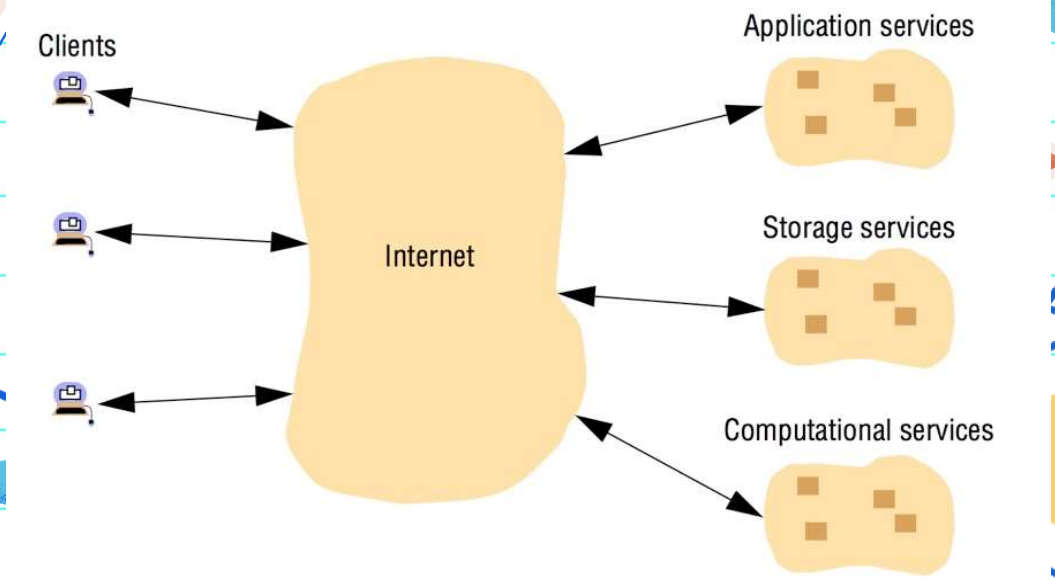
Portable/Handheld Devices in a Distributed System



Web Servers and Web Browsers



Cloud Services



Jungle Computing

- Use diverse, distributed and highly non-uniform high performance systems to achieve peak performance.

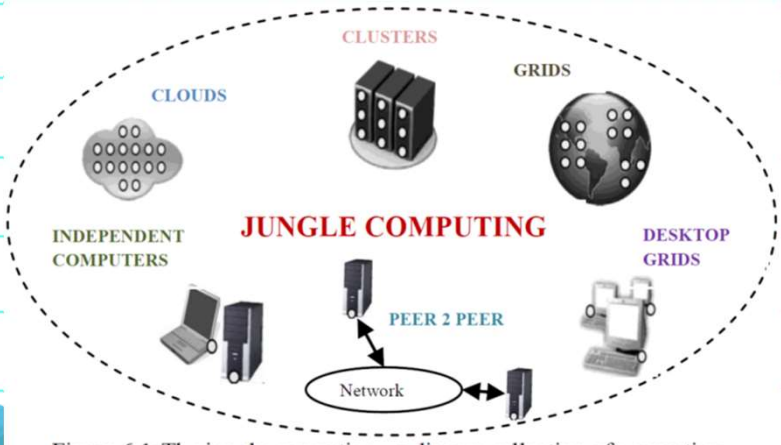
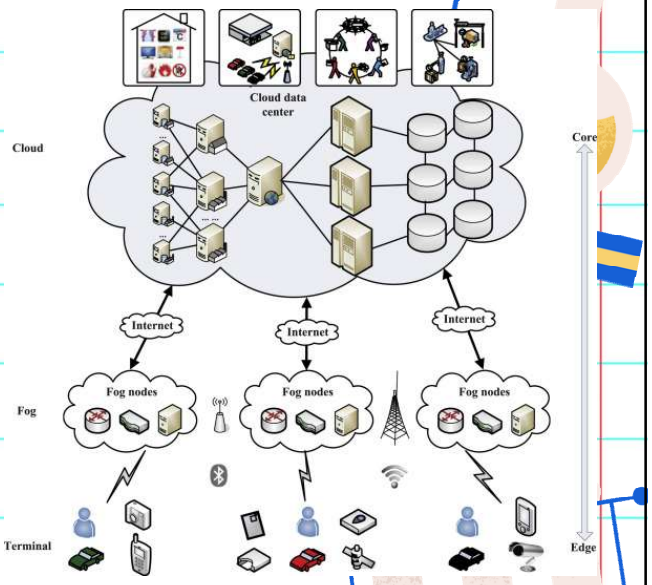
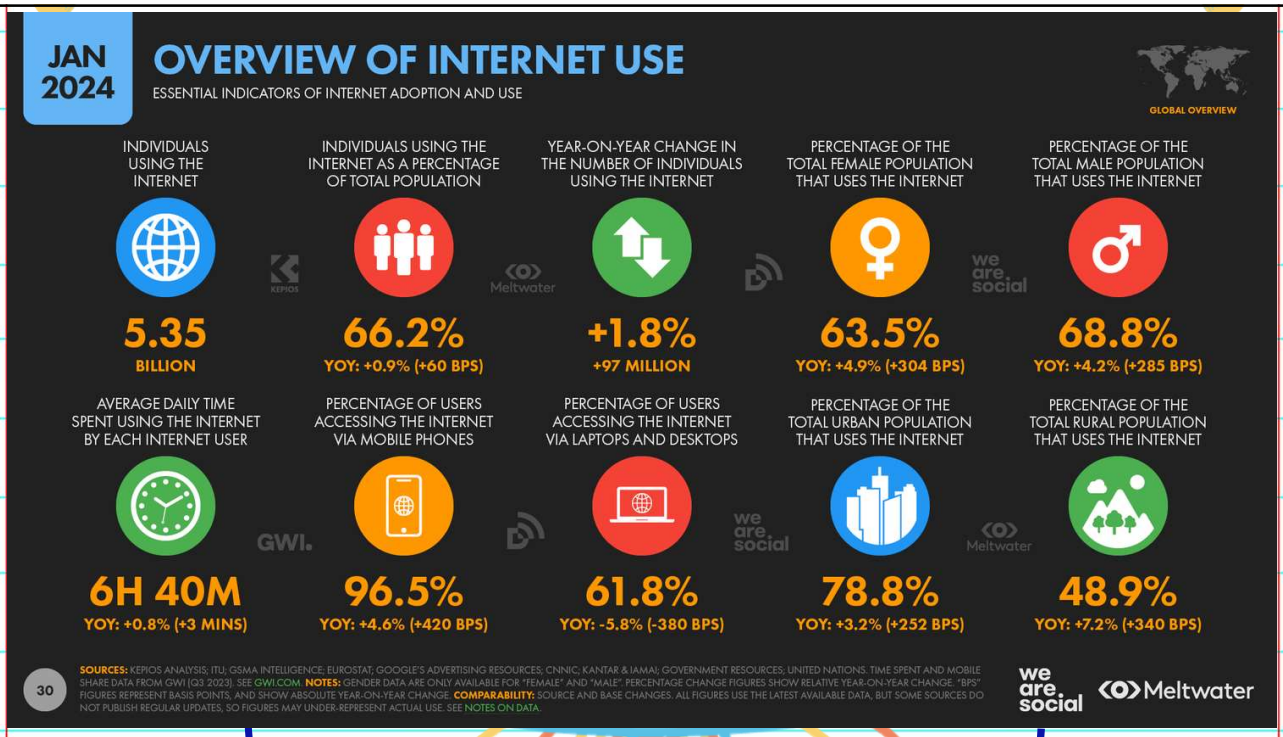


Figure 6.1: The jungle computing - a diverse collection of computing

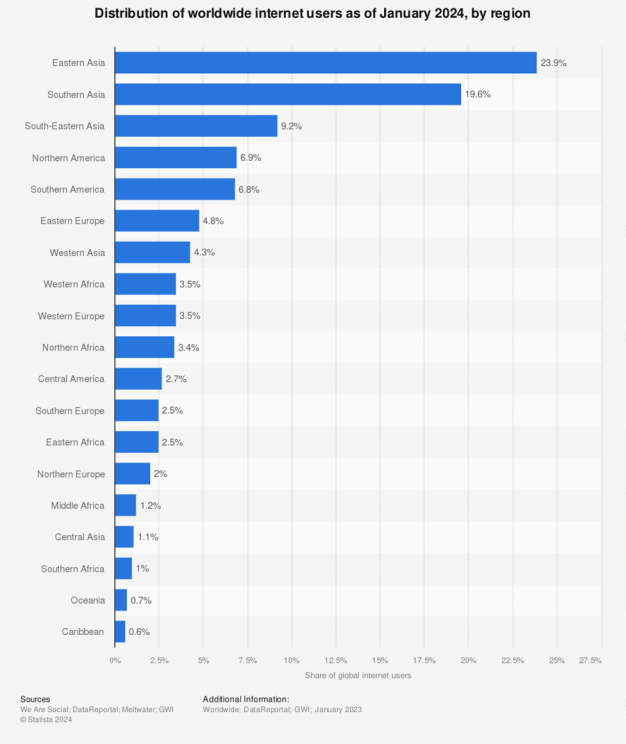
Fog/Edge Computing

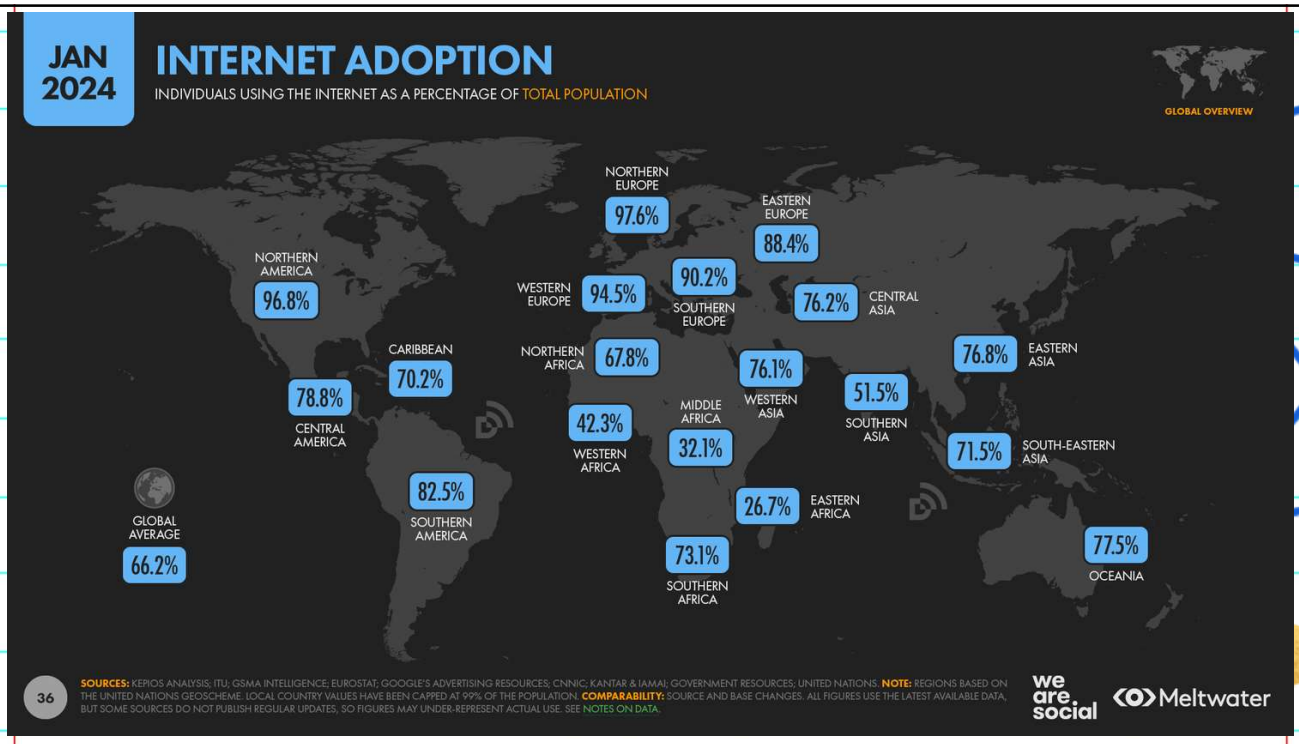
- Extend cloud computing to the edge so that data, compute, storage and applications are distributed in the most logical, efficient place between the data source and the cloud.

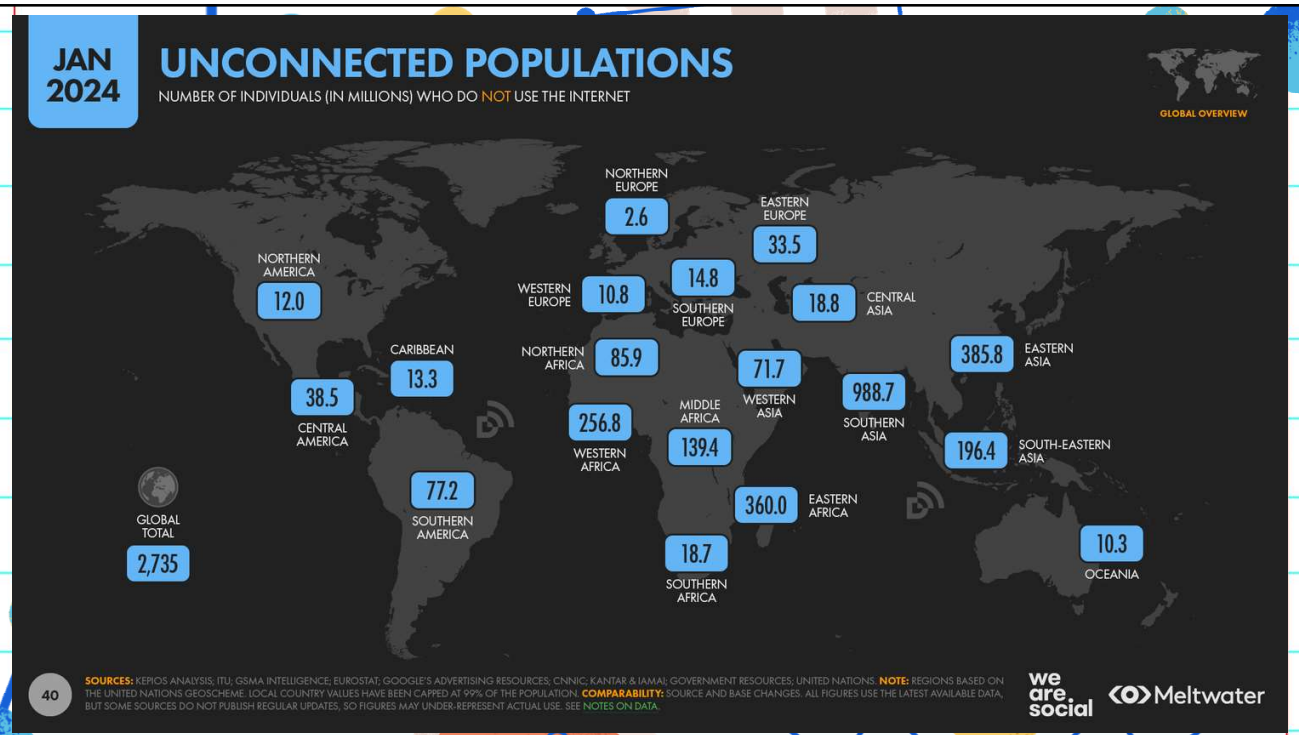




Distributeion of Internet Users




JAN 2024
INTERNET ADOPTION
INDIVIDUALS USING THE INTERNET AS A PERCENTAGE OF TOTAL POPULATION
GLOBAL OVERVIEW
NORTHERN EUROPE 97.6%
EASTERN EUROPE 88.4%
NORTHERN AMERICA 96.8%
WESTERN EUROPE 94.5%
SOUTHERN EUROPE 90.2%
CENTRAL ASIA 76.2%
CARIBBEAN 70.2%
NORTHERN AFRICA 67.8%
EASTERN ASIA 76.8%
WESTERN ASIA 76.1%
CENTRAL AMERICA 78.8%
MIDDLE AFRICA 42.3%
SOUTHERN ASIA 51.5%
WESTERN AFRICA 32.1%
SOUTH-EASTERN ASIA 71.5%
GLOBAL AVERAGE 66.2%
SOUTHERN AMERICA 82.5%
EASTERN AFRICA 26.7%
SOUTHERN AFRICA 73.1%
OCEANIA 77.5%
36
SOURCES: KEPIOS ANALYSIS; ITU; GSMA INTELLIGENCE; EUROSTAT; GOOGLE'S ADVERTISING RESOURCES; CNNIC; KANTAR & IAMAI; GOVERNMENT RESOURCES; UNITED NATIONS. NOTE: REGIONS BASED ON THE UNITED NATIONS GEOSCHEME. LOCAL COUNTRY VALUES HAVE BEEN CAPPED AT 99% OF THE POPULATION. COMPARABILITY: SOURCE AND BASE CHANGES. ALL FIGURES USE THE LATEST AVAILABLE DATA, BUT SOME SOURCES DO NOT PUBLISH REGULAR UPDATES, SO FIGURES MAY UNDER-REPRESENT ACTUAL USE. SEE NOTES ON DATA.
we are social
Meltwater


JAN 2024
UNCONNECTED POPULATIONS
NUMBER OF INDIVIDUALS (IN MILLIONS) WHO DO NOT USE THE INTERNET
GLOBAL OVERVIEW
NORTHERN EUROPE 2.6
EASTERN EUROPE 33.5
NORTHERN AMERICA 12.0
WESTERN EUROPE 10.8
SOUTHERN EUROPE 14.8
CENTRAL ASIA 18.8
CARIBBEAN 13.3
NORTHERN AFRICA 85.9
EASTERN ASIA 385.8
WESTERN ASIA 71.7
CENTRAL AMERICA 38.5
MIDDLE AFRICA 256.8
SOUTHERN ASIA 988.7
WESTERN AFRICA 139.4
SOUTH-EASTERN ASIA 196.4
GLOBAL TOTAL 2,735
SOUTHERN AMERICA 77.2
EASTERN AFRICA 360.0
SOUTHERN AFRICA 18.7
OCEANIA 10.3
40
SOURCES: KEPIOS ANALYSIS; ITU; GSMA INTELLIGENCE; EUROSTAT; GOOGLE'S ADVERTISING RESOURCES; CNNIC; KANTAR & IAMAI; GOVERNMENT RESOURCES; UNITED NATIONS. NOTE: REGIONS BASED ON THE UNITED NATIONS GEOSCHEME. LOCAL COUNTRY VALUES HAVE BEEN CAPPED AT 99% OF THE POPULATION. COMPARABILITY: SOURCE AND BASE CHANGES. ALL FIGURES USE THE LATEST AVAILABLE DATA, BUT SOME SOURCES DO NOT PUBLISH REGULAR UPDATES, SO FIGURES MAY UNDER-REPRESENT ACTUAL USE. SEE NOTES ON DATA.
we are social
Meltwater

Rates of Growth

- Moore's Law
  - "Number of transistors in chips doubles every 18 months".
  - ⇒ Every 10 years, processors are 100 times more powerful.
- Gilder's Law
  - "Bandwidth grows at least three times faster than computer power".
  - ⇒ Assuming bandwidth doubles every 12 months; every 10 years, it is 1000 times better.
- Consequences ?

Key Design Challenges

- Connecting users and sharing resources
- Heterogeneity
- Distribution transparency
- Openness
- Dependability
- Availability and reliability
- Fault handling
- Scalability
- Security and privacy
- Concurrency
- Mobility and location dependency
- Quality of service (QoS)
- Quality of experience (QoE)

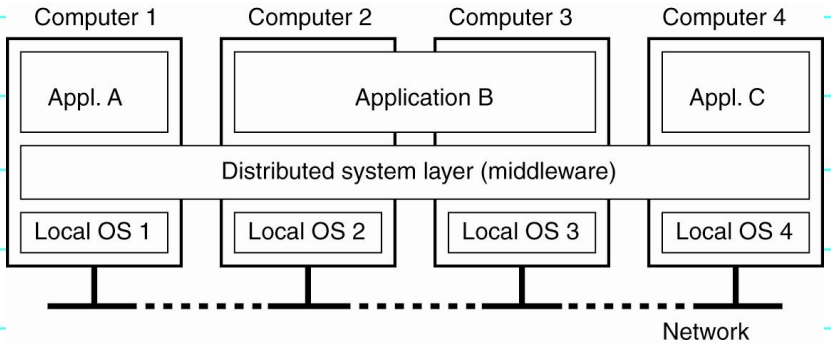
Sharing Resources

- Canonical examples:
  - Cloud-based shared storage and files
  - Peer-to-peer assisted multimedia streaming
  - Shared mail services (think of outsourced mail systems)
  - Shared Web hosting (think of content distribution networks)
- Observation
  - "*The network is the computer*" (John Gage, then at Sun Microsystems)

Heterogeneity(異質性)

- Networks
  - Ethernet, token ring, wired/wireless, satellite, …
- Computer hardware
  - Different components, brands, form factor, …
- Operating systems
  - different API of Unix, Windows, iOS, …
- Programming languages
  - different representations for data structures
- Implementations from different developers
  - no application standards

Middleware to Hide Heterogeneity



To hide the heterogeneity, a distributed system is usually organized as **middleware**. Note that the middleware layer extends over multiple machines, and offers each application the **same interface**.

Transparency(通透性)

Transparency	Description
Access	**Hide** differences in data representation and how an object is accessed (local and remote resources are accessed using identical operations)
Location	Hide where an object is located (access objects without the need to know their physical or network location such as building or IP address)
Migration	Hide that an object may move to another location
Relocation	Hide that an object may be moved to another location while in use
Replication	Hide that an object is replicated (multiple instances of objects can be used to increase reliability and performance without knowledge of the replicas by users or application programmers)

Transparency (contd.)

Transparency	Description
Concurrency	Hide that an object may be shared by several competitive users (several processes can operate concurrently using shared objects without interference between them)
Failure	Hide the failure and recovery of an object (conceal faults, allowing users and application programs to complete their tasks despite the failure of hardware or software components)
Performance	Allow the system to be reconfigured to improve performance as loads vary
Scaling	Allow the system and applications to expand in scale without change to the system structure or the application algorithms
Persistence	Hide whether an object is in memory or on disk
Programming	Hide the physical distribution from the application developers

Degree of Transparency

- It is not necessarily a good idea to hide all distribution aspects
  - Eg. Location-based services
- We often need to consider the degree of transparency
- May need to trade-off between transparency and performance
- **Question**: How hard should we try to make things transparent? Can we make things truly transparent?

Degree of Transparency

- **Observation**: Aiming at **full** distribution transparency may be too much:
  - Users may be located in different continents; distribution is apparent
  - There are communication latencies that cannot be hidden.
  - Completely hiding failures of networks/nodes is (theoretically and practically) impossible.
    - Cannot distinguish a slow computer from a failing one
    - Can never be sure that a server actually performed an operation before a crash
  - Full transparency may be too costly or cost performance
    - Keeping Web caches exactly up-to-date with the master copy
    - Immediately flushing write operations to disk for fault tolerance

Degree of Transparency

- Exposing distribution may be good
  - Making use of location-based services (finding your nearby friends)
  - When dealing with users in different time zones
  - When it makes it easier for a user to understand what's going on (when e.g., a server does not respond for a long time, report it as failing).
- Conclusion
  - Distribution transparency is a nice goal, but achieving it is a different story, and it should often not even be aimed at.

Openness

- Openness of a computer system
  - is the characteristic that determines whether the system can be extended and re-implemented in various way.
  - Eg. UNIX, open source software
- Openness of distributed systems
  - determined by the degree to witch new resource sharing services can be added and be made available for use by a variety of other programs.
  - Eg. Web
- How to deal with openness?
  - key interfaces are published.
  - Eg. RFC

Openness (cont.)

- **Open distributed system**: A system that offers components/services that can easily be used by, or integrated into other systems. Often offers services according to standard rules that describe the syntax and semantics of those services.
- Rules are formalized into protocols.
- Services are specified through well-defined interfaces (in an Interface Definition Languages, IDL).
- Proper specifications should be
  - Complete: everything necessary has been specified
  - Neutral: do not prescribe what an implementation should look like
- **Goals**: interoperability, portability, extensibility
- Separate policy from mechanism

Policies versus Mechanisms

- **Implementing openness**: policies
  - What level of consistency do we require for client-cached data?
  - Which operations do we allow downloaded code to perform?
  - Which QoS requirements do we adjust in the face of varying bandwidth?
  - What level of secrecy do we require for communication?

Policies versus Mechanisms

- **Implementing openness**: mechanisms
  - Allow (dynamic) setting of caching policies
  - Support different levels of trust for mobile code
  - Provide adjustable QoS parameters per data stream
  - Offer different encryption algorithms
  - …

On Strict Separation

- **Observation**
  - The stricter the separation between policy and mechanism, the more we need to ensure proper mechanisms, potentially leading to many configuration parameters and complex management
- Finding a balance
  - Hard-coding policies often simplifies management, and reduces complexity at the price of less flexibility. There is no obvious solution.

Dependability

- Basic
  - A component provides services to clients. To provide services, the component may require the services from other components ⇒ a component may depend on some other component.
- Specifically
  - A component C depends on C^* if the correctness of C's behavior depends on the correctness of C^*'s behavior. (Components are processes or channels.)

Dependability

- Requirements related to dependability

Requirement	Description
Availability	Readiness for usage
Reliability	Continuity of service delivery
Safety	Very low probability of catastrophes
Maintainability	How easy can a failed system be repaired

Reliability vs Availability

- Reliability $R(t)$ of component C
  - Conditional probability that C has been functioning correctly during $[0, t)$ given C was functioning correctly at the time $T = 0$.
- Traditional metrics:
  - Mean Time To Failure (*MTTF*): The average time until a component fails
  - Mean Time To Repair (*MTTR*): The average time needed to repair a component
  - Mean Time Between Failures (*MTBF*): Simply *MTTF* + *MTTR*.

Fault Handling

- Terminology: Failure, error, fault

Term	Description	Example
Failure	A component is not living up to its specifications	Crashed program
Error	Part of a component that can lead to a failure	Programming bug
Fault	Cause of an error	Sloppy programmer

Fault Handling

Term	Description	Example
Fault prevention	Prevent the occurrence of a fault	Don't hire sloppy programmers
Fault tolerance	Build a component and make it mask the occurrence of a fault	Build each component by two independent programmers
Fault removal	Reduce the presence, number, or seriousness of a fault	Get rid of sloppy programmers
Fault forecasting	Estimate current presence, future incidence, and consequences of faults	Estimate how a recruiter is doing when it comes to hiring sloppy programmers

Scalability 1

- A system is described as scalable
  - if it will remain effective when there is a significant increase in the number of resources and the number of users
- A scalable example system: the Internet
- Design challenges
  - Controlling the cost of physical resources
    - e.g., servers support users at most O(n)
  - Controlling the performance loss
    - e.g., DNS no worse than O(logn)
  - Preventing software resources from running out
    - e.g., IP address
  - Avoiding performance bottlenecks
    - e.g., partitioning name table of DNS, cache and replication

Scalability 2

- Three dimensions:
  - **Size scalability**: no. of users and/or processes
  - **Geographically scalability**: max distance between nodes
  - **Administratively scalability**: no. of admin domains
- Scalability limitations:

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

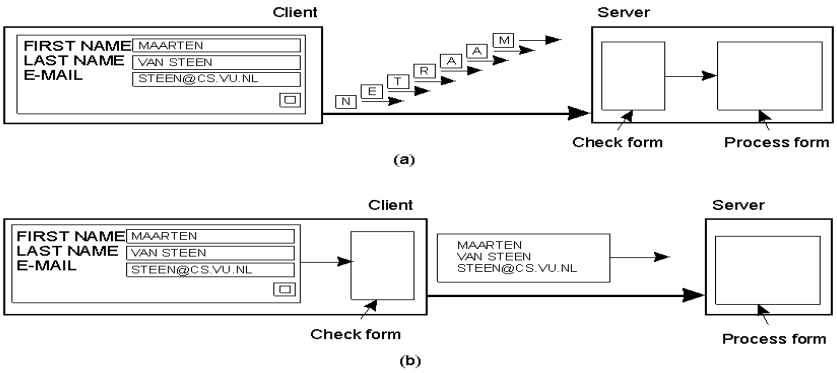
Scaling Techniques

- **Basic techniques** for scaling:
  - Use only decentralized algorithms
  - Hiding communication latencies
  - Distribution
  - Replication
  - (more details in coming slides)

Decentralized Algorithms

- Why are centralized algorithms bad?
  - Single point of failure
  - Lots of communications to one point (bottleneck)
- Use only decentralized algorithms
  - No complete information about system state.
  - Make decisions based only on local information.
  - Failure of one does not ruin the algorithm.
  - No global clock.
- Centralized or decentralized: DNS, BitTorrent
- There are hybrids.

Hiding Comm. Latencies

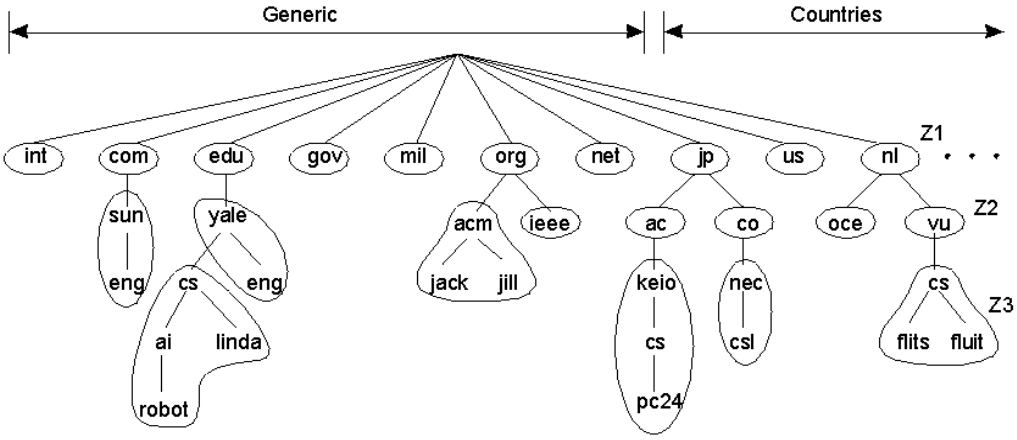


The difference between letting:

a)the server or

b)clients check forms as they are being filled

Distribution



An example of dividing the DNS name space into zones.

Replication

- Server based replication
  - Scalable web servers
  - Web proxy caching
  - Replicated databases
  - Pushing techniques
  - Auto replication in cloud storage
- Client based replication
  - Caching
  - Prefetching
- Static vs. dynamic replication
- Consistency and synchronization overhead

Security

- Confidentiality
  - protection against disclosure to unauthorized individuals
    - e.g. ACL in Unix File System
- Integrity
  - protection against alteration or corruption
    - e.g. checksum
- Availability
  - protection against interference with the means to access the resources
    - e.g. Denial of Service(DOS)

Security Tasks

- A distributed system may be not secure, not dependable.
- To ensure confidentiality and integrity, we need to take some security measures.
- Authentication: verifying the correctness of a claimed identity
- Authorization: does an identified entity has proper access rights?
- Trust: one entity can be assured that another will perform particular actions according to a specific expectation

Privacy

- Limited access to personal information
- Secrecy (option to conceal information)
- Hide information usage
- Control over others' use of personal information
- States of privacy
- Information privacy – Healthcare, financial, biological, criminal, residence, locational, political, …
- Protecting privacy
  - Privacy laws
  - Policy, policy communication, policy enforcement
  - Privacy preserving design

Failure Handling

- Detecting failures
  - e.g. checksum for corrupted data
  - Sometimes impossible, e.g. a remote crashed server in the Internet
- Masking failures
  - e.g. Retransmit message, standby server
- Tolerating failures
  - e.g. a web browser cannot contact a web server
- Recovery from failures
  - e.g. Roll back
- Redundancy
  - e.g. IP route, replicated name table of DNS

Concurrency

- Correctness
  - ensure the correctness of operations on shared resource in a concurrent environment
    - e.g. records bids for an auction
- Performance
  - Ensure the high performance of concurrent operations

Mobility and Location Dependency

- Mobility changes everything, offers new opportunities
- Location management
- Location Based Services (LBS)
  - "Show me all the restaurants within 5 miles of my current location."
- Location-dependent data management
- Tracking moving objects
- Group tracking
- Mobile crowd sensing & computing (MCSC)

Quality of Service (QoS)

- Functionality is not enough.
- We want quality as well.
- Main nonfunctional properties of systems:
  - Levels of services
  - Levels of performance
  - Availability
  - Reliability
  - Security
- Adaptability is the key to meet QoS requirements.

Quality of Experience (QoE)

- A measure of a customer's experience on services.
- Measure satisfaction both objectively and subjectively.
- Major factors that affect QoE
  - Cost
  - Reliability
  - Efficiency
  - Privacy
  - Security
  - Interface user-friendliness
  - User confidence.

Types of Distributed Systems

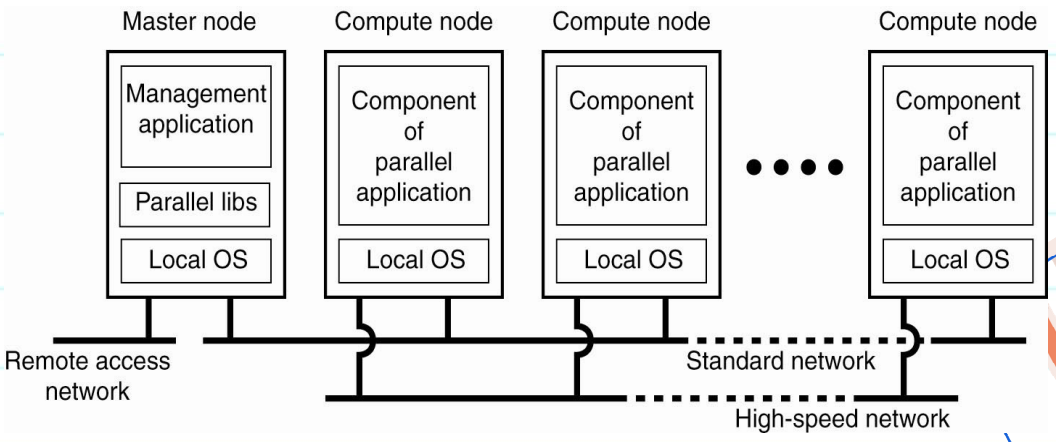
- Distributed computing systems — for high performance computing tasks
  - Cluster computing — collection of workstations or PCs connected by high-speed LAN
  - Grid computing — federation of computers over WAN
- Distributed information systems — for organizational information processing and enterprise application integration
- Distributed pervasive systems — with many mobile and embedded devices

Examples of Distributed Systems

- Cluster computing systems
- Grid computing systems
- Cloud computing systems
- Transaction processing systems
- Enterprise application integration
- Distributed pervasive systems
- IoT, Sensor networks
- …
- (more details to follow)

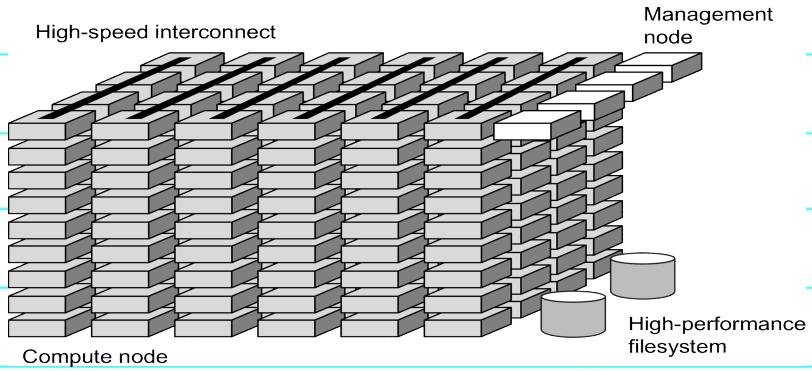
Cluster Computing Systems

- An example of a cluster computing system consisting of homogeneous computing nodes.



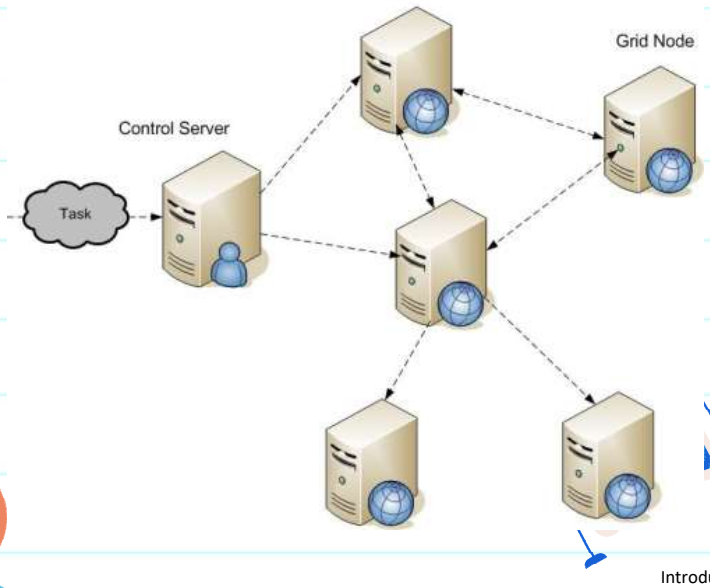
Modern Cluster

- Modern clusters in data centers are organized by stacks of compute nodes which are essentially a group of high-end systems connected through a LAN with
  - Homogeneous: same OS, near-identical hardware
  - Single, or tightly coupled managing node(s)



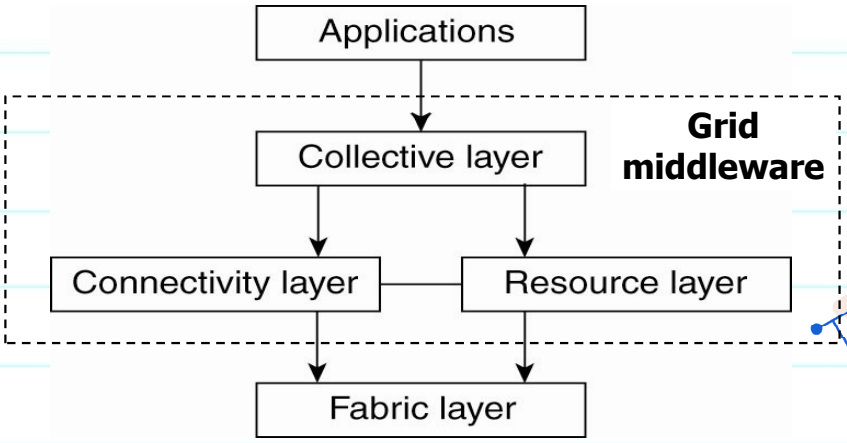
Grid Computing Systems(1)

- A pool of servers, storage systems and networks that can be used as a single big system.
- Grid computing is to manage all these resources in the execution of tasks.



Grid Computing Systems(2)

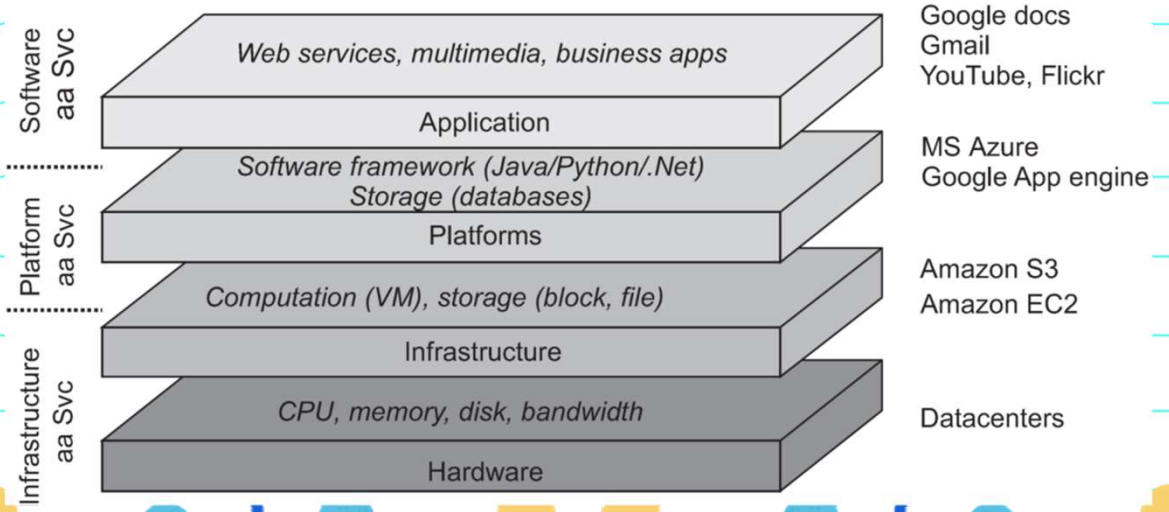
- A layered architecture for grid computing systems consisting of heterogeneous computers. (next slide for details)



Grid Computing Systems(3)

- **Fabric layer** — Provide interfaces to local resources
  - Querying resource state and capabilities
  - Local resource managenent
- **Connectivity layer** — Support communication protocols for grid transactions that span the usage of resources.
  - Security, authentication
- **Resource layer** — Manage a single resource
  - Access control
- **Collective layer** — Handle access to multiple resources
  - Resource discovery, allocation
  - Task scheduling

Cloud Computing

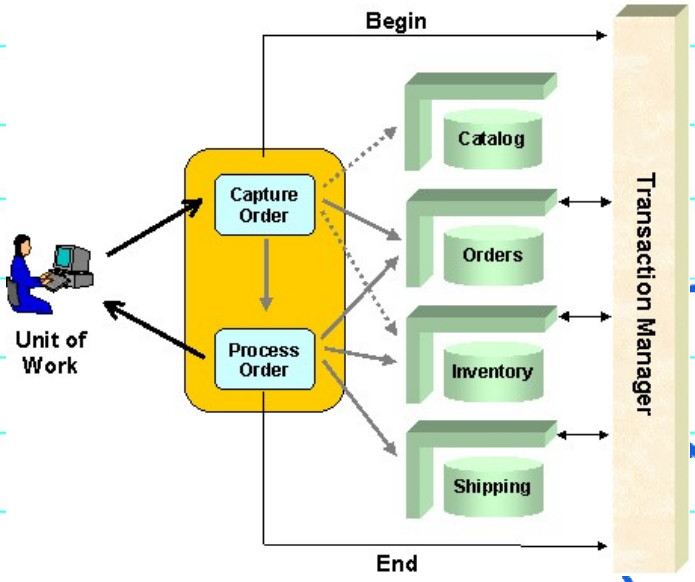


Transaction Processing Systems

- Systems for processing online transactions on databases.
- Example primitives for transactions.

Primitive	Description
BEGIN_TRANSACTION	Mark the start of a transaction
END_TRANSACTION	Terminate the transaction and try to commit
ABORT_TRANSACTION	Kill the transaction and restore the old values
READ	Read data from a file, a table, or otherwise
WRITE	Write data to a file, a table, or otherwise

Transaction Processing



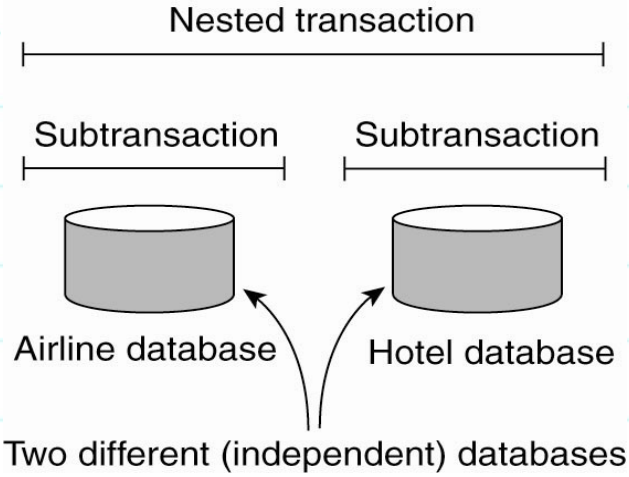
Transactions

Characteristic properties of transactions (**ACID**):

- **Atomic**: To the outside world, the transaction happens indivisibly.
- **Consistent**: The transaction does not violate system invariants.
- **Isolated**: Concurrent transactions do not interfere with each other.
- **Durable**: Once a transaction commits, the changes are permanent.

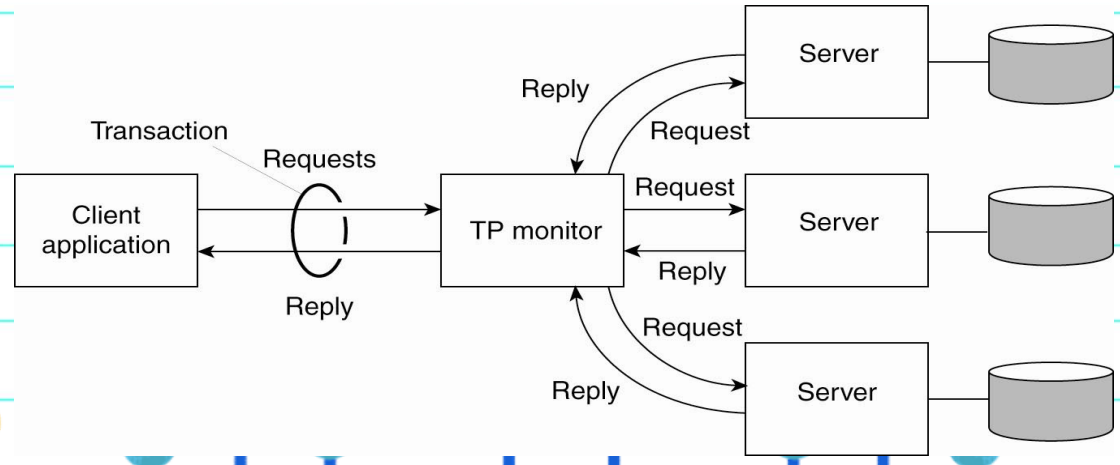
Nested Transaction

- Transactions may have subtransactions.



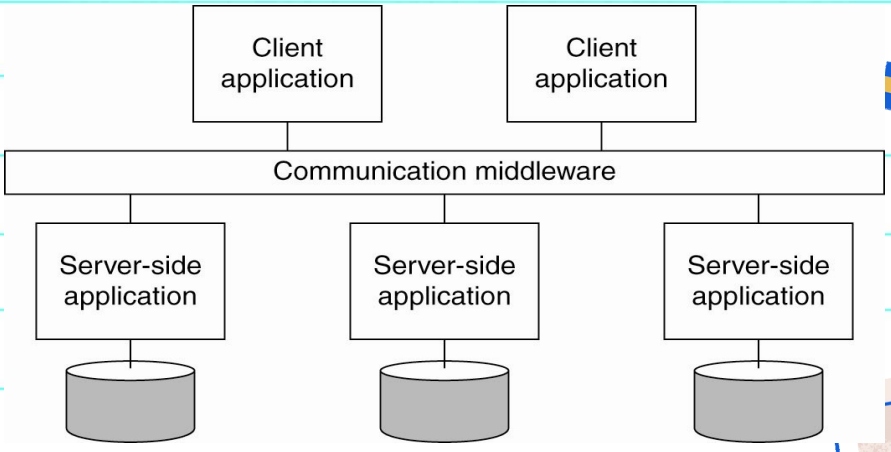
Transaction Processing Monitor

- The role of a TP monitor in distributed systems (e.g. reserving three different flights).



Enterprise Application Integration

- Middleware as a communication facilitator in enterprise application integration.



How to integrate?

- File transfer: Technically simple, but not flexible
  - Figure out file format and layout
  - Figure out file management
  - Update propagation, and update notifications
- Shared database: Much more flexible, but still requires common data scheme next to resk of bottleneck
- Remote procedure call: Effective when execution of a series of actions is needed.
- Messaging: RPCs require caller and callee to be up and running at the same time. Messaging allows decoupling in time and space.

Distributed Pervasive Systems

- Proliferation of mobile, embedded sensing and computing devices raises the new era of pervasive computing characterized by the fact that the system naturally blends into the user's environment.
- Requirements for pervasive systems
  - Embrace contextual changes.
  - Encourage ad hoc composition.
  - Recognize sharing as the default.

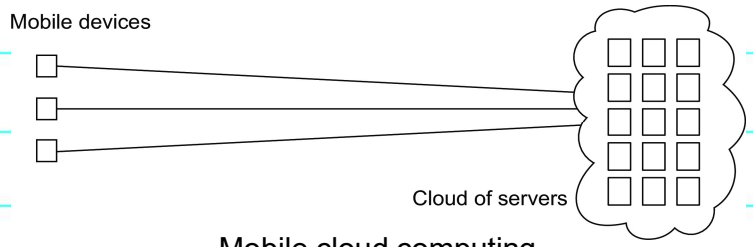
Ubiquitous Systems

- Core elements
  1. (Distribution) Devices are networked, distributed, and accessible transparently
  2. (Interaction) Interaction between users and devices is highly unobtrusive
  3. (Context awareness) The system is aware of a user's context to optimize interaction
  4. (Autonomy) Devices operate autonomously without human intervention, and are thus highly self-managed
  5. (Intelligence) The system as a whole can handle a wide range of dynamic actions and interactions

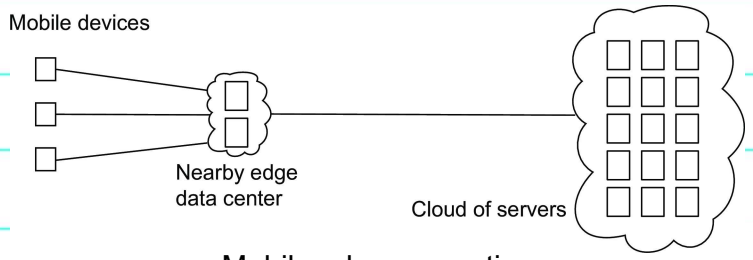
Mobile Computing

- **Distinctive features**
  - A myriad of different mobile devices (smartphones, tablets, GPS devices, remote controls, active badges).
  - Mobile implies that a device's location is expected to change over time ⇒ change of local services, reachability, etc. Keyword: discovery.
  - Maintaining stable communication can introduce serious problems.
  - For a long time, research has focused on directly sharing resources between mobile devices. It never became popular and is by now considered to be a fruitless path for research.
- Bottomline: Mobile devices set up connections to stationary servers, essentially bringing mobile computing in the position of clients of cloud-based services.

Mobile Computing



Mobile cloud computing



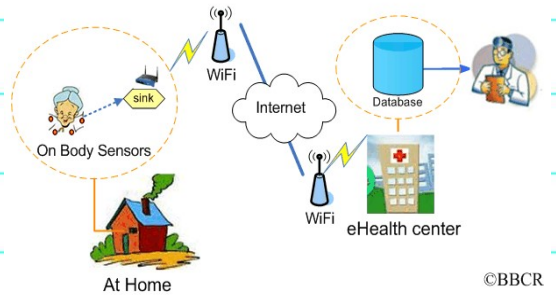
Mobile edge computing

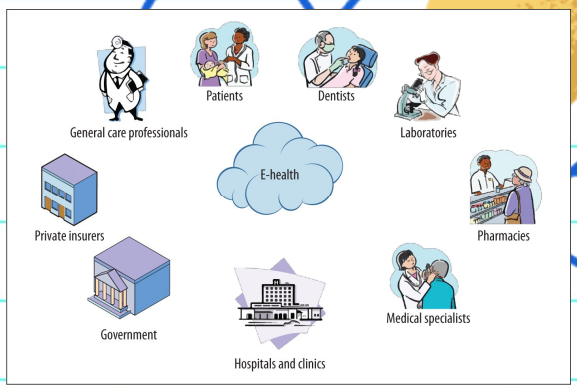
E-Health Systems (1)

- Questions to be addressed for health care systems:
  - Where and how should monitored data be stored?
  - How can we prevent loss of crucial data?
  - What infrastructure is needed to generate and propagate alerts?
  - How can physicians provide online feedback?
  - How can extreme robustness of the monitoring system be realized?
  - What are the security issues and how can the proper policies be enforced?

E-Health Systems (2)

- From personal monitoring to hospital healthcare provisioning.



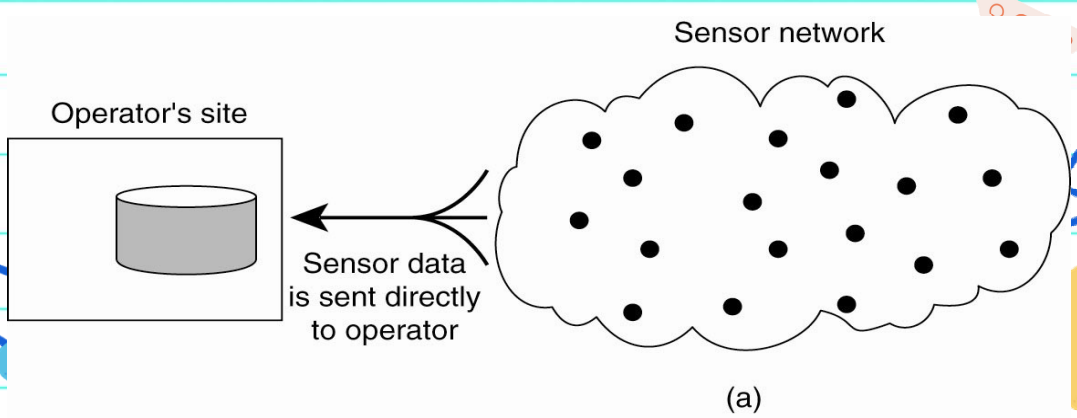


Sensor Networks (1)

- Network of small sensors with computing and wireless communication capabilities
- Questions concerning sensor networks:
  - How do we (dynamically) set up an efficient infrastructure (e.g. tree) in a sensor network?
  - How does aggregation of results take place? Can it be controlled?
  - What happens when network links fail?

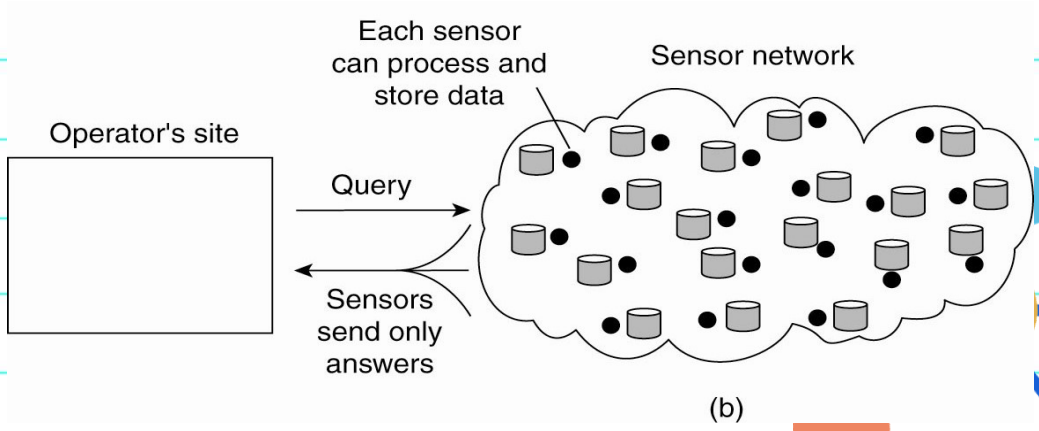
Sensor Networks (2)

- Organizing a sensor network database, while storing and processing data (a) only at the operator's site or …



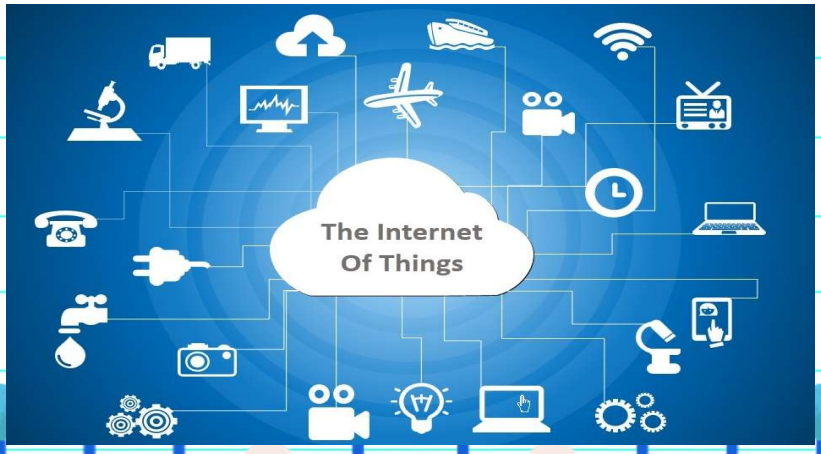
Sensor Networks (3)

- Organizing a sensor network database, while storing and processing data … or (b) only at the sensors.

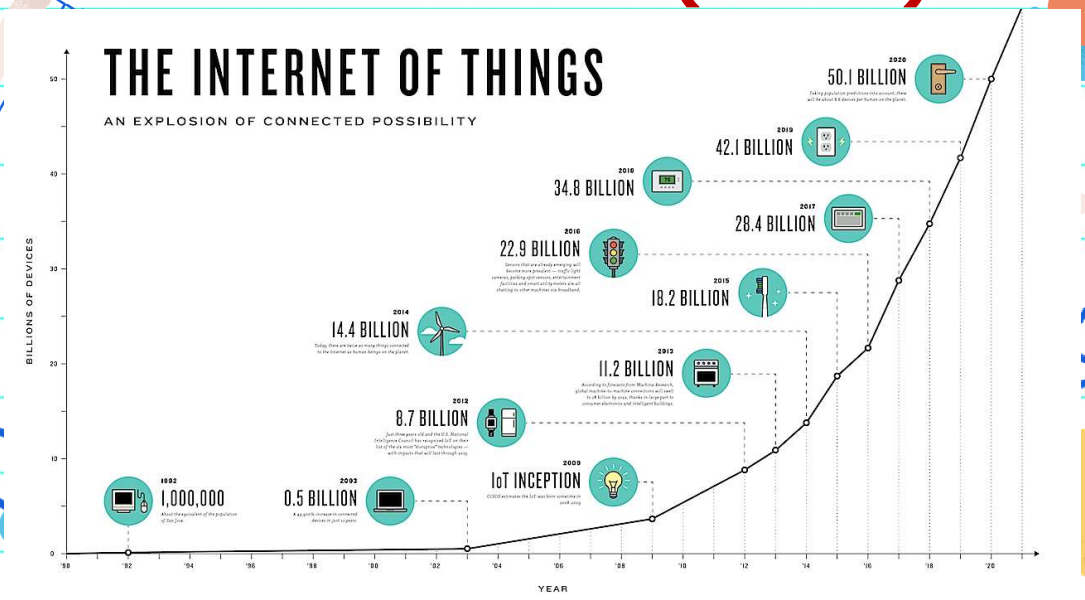


Internet of Things (IoT)

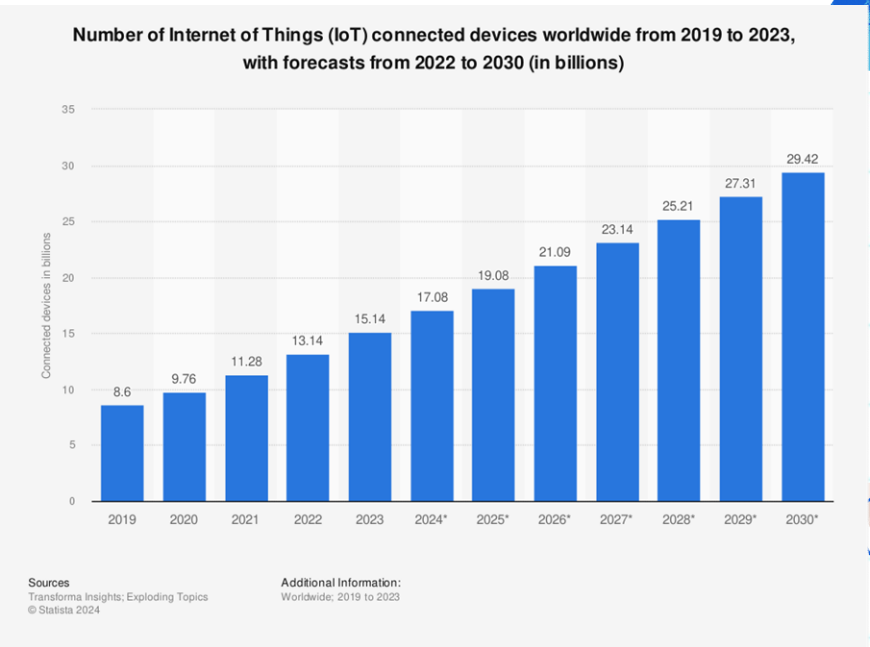
- Internetworking of physical devices, vehicles, buildings, … and other objects (with electronics, software, sensors, actuators, and networks) to collect and exchange data.



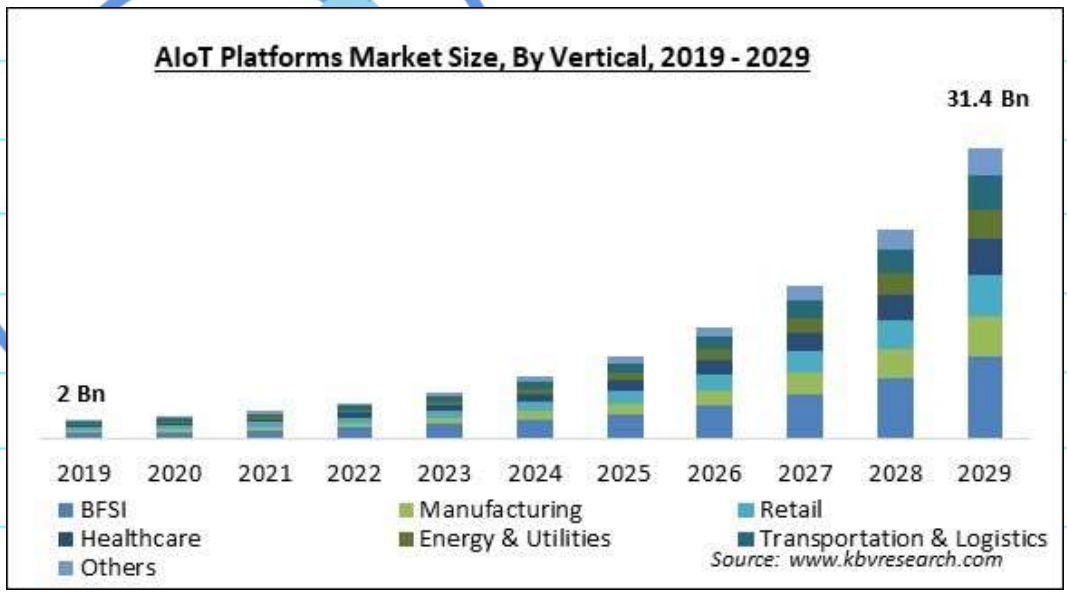
IoT Forecast (2016)



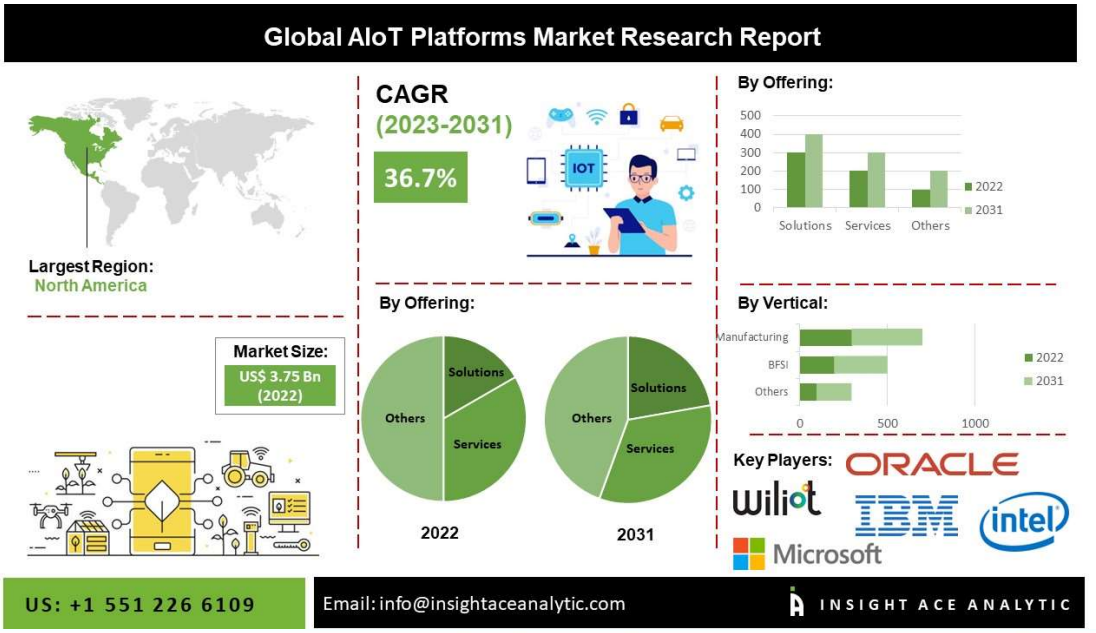
IoT State & Forecast



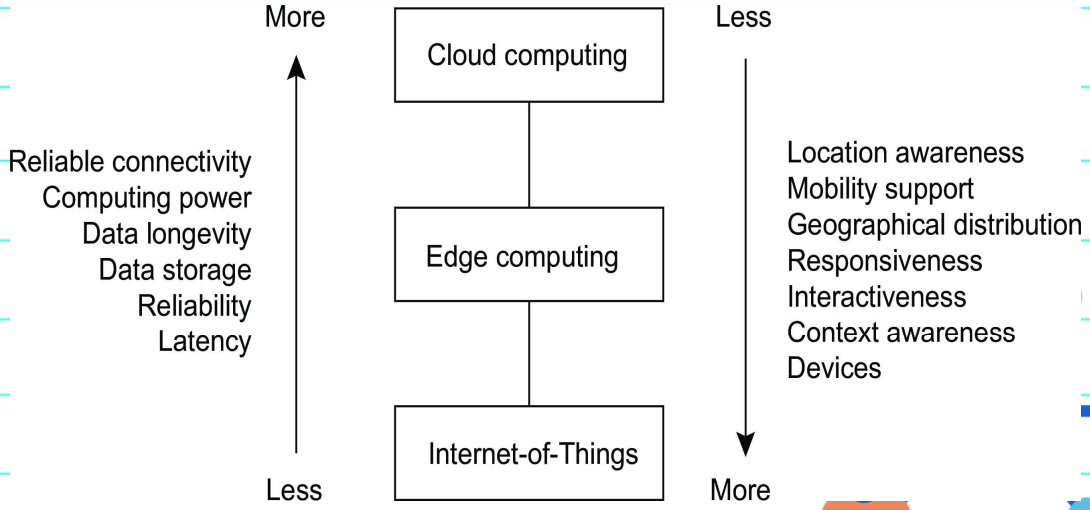
AIoT Platforms Market Size



Global AIoT Platforms Market



The Cloud-Edge Continuum



Fallacies of Distributed Computing

- False assumptions made by developers:
  - The network is reliable
  - The network is secure
  - The network is homogeneous
  - Topology doesn't change
  - Latency is zero
  - Bandwidth is infinite
  - Transport cost is zero
  - There is one administrator
  - System clocks are identical

Summary

- Distributed systems are everywhere today.
- Many different types of distributed systems for different environments and applications.
- Each type of system has its own characteristics and design challenges.
- However, there are some fundamental issues common to all distributed systems.
- We will discuss theories and techniques to handle these issues as well as the extensions and adaptation needed for different types of systems.