Introduction to Distributed Systems

Slide set 1
Distributed Systems

Graduate Level

K. N. Toosi Institute of Technology
Dr. H. Khanmirza
h.khanmirza@kntu.ac.ir

Course Syllable

- Communication
  - RPC, RMI
  - Message Queuing
    - Communication Patterns
  - MPI
  - Distributed Shared Memory
  - Broadcast & Multicast
    - Epidemic Broadcast
- Coordination
  - Time Synchronization
  - Logical Time, Vector Clock
  - Distributed Lock
  - Distributed Leader Election

Course Syllable

- Naming
  - Flat & Hierarchical Naming
  - Consistent Hashing, DHT, CDN,
  - P2P (Chord, BitTorrent, Pastery, CAN, Topetrsy)
  - Content Centric Networks
- Consistency
  - Protocols
  - Data-centric Consistency
  - Client-centric Consistency

Course Syllable

- Fault Tolerance
  - Fault Types
  - Flooding Consensus
  - Distributed Commit
    - 2PC Algorithm
    - 3PC Algorithm
  - Paxos
  - RAFT
  - Byzantine Failure Problems
    - Oral Message Algorithm
    - Signed Message Algorithm
    - PBFT Algorithm

Course Syllable

- Distributed File Systems
  - Network File System (NFS)
  - Andrew File System (AFS)
  - Coda
  - Google File System (GFS)
- NoSql Databases
  - Concepts and Types
  - Amazon Dynamo
- Blockchain
  - Blockchain Concepts
  - Cryptocurrency Concepts
  - Consensus Algorithms
    - Bitcoin Proof-of-Work
    - Ethereum Proof-of-Stake

Text books & Grading

- Distributed systems: Principles and Paradigms
  - Andrew S. Tanenbaum, Maarten van Steen, 2nd ed – 2007
- Distributed systems: Principles and Paradigms
  - Andrew S. Tanenbaum, Maarten van Steen, 3rd ed – 2017
- Distributed Systems: Concepts and Design
  - George Coulouris, Gordon Blair, 5th ed – 2012
- Papers & Articles
- Grading

Midterm 1	7
final	10
Research	3

What is a Distributed System?

Distributed System

"A distributed system is a collection of autonomous computing elements that appears to its users as a single coherent system.."

- Definition emphasize on
  - Collection of computing nodes behave independently
  - Viewing a single system
- No particular assumption on
  - Node types → sensor, computer, server, a software process, ⋯
  - No special assumption on type of Interconnection

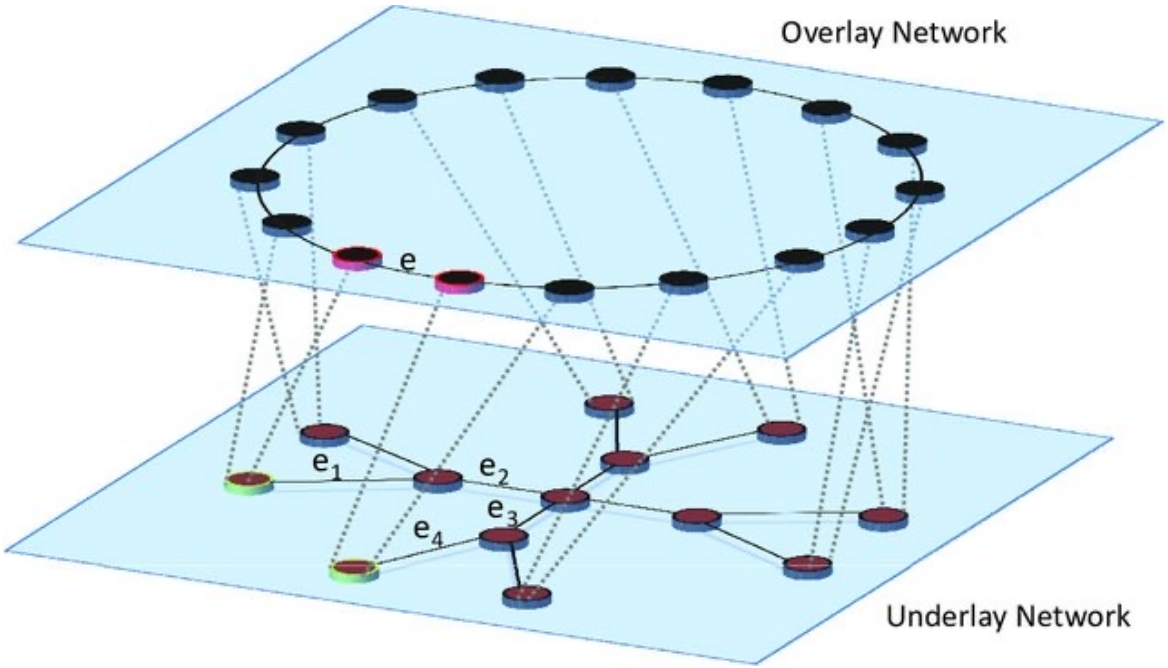
Distributed System

- Collection of computing nodes behave independently
  - Act independently
  - Coordinate & cooperate by exchanging messages
    - Need synchronization no **Global Clock**.
  - Collection of nodes → Groups
    - Closed Groups
    - Open Groups

Distributed System

- Collection of computing nodes behave independently (cont.)
  - Distributed system is often structured as an Overlay Networks
    - A logical network over a physical network
    - Structured vs Unstructured
      - Having well-defined shape and set of neighbors vs random neighbors
    - Dynamic vs Static overlays
      - In P2P clients may leave and join

Distributed System



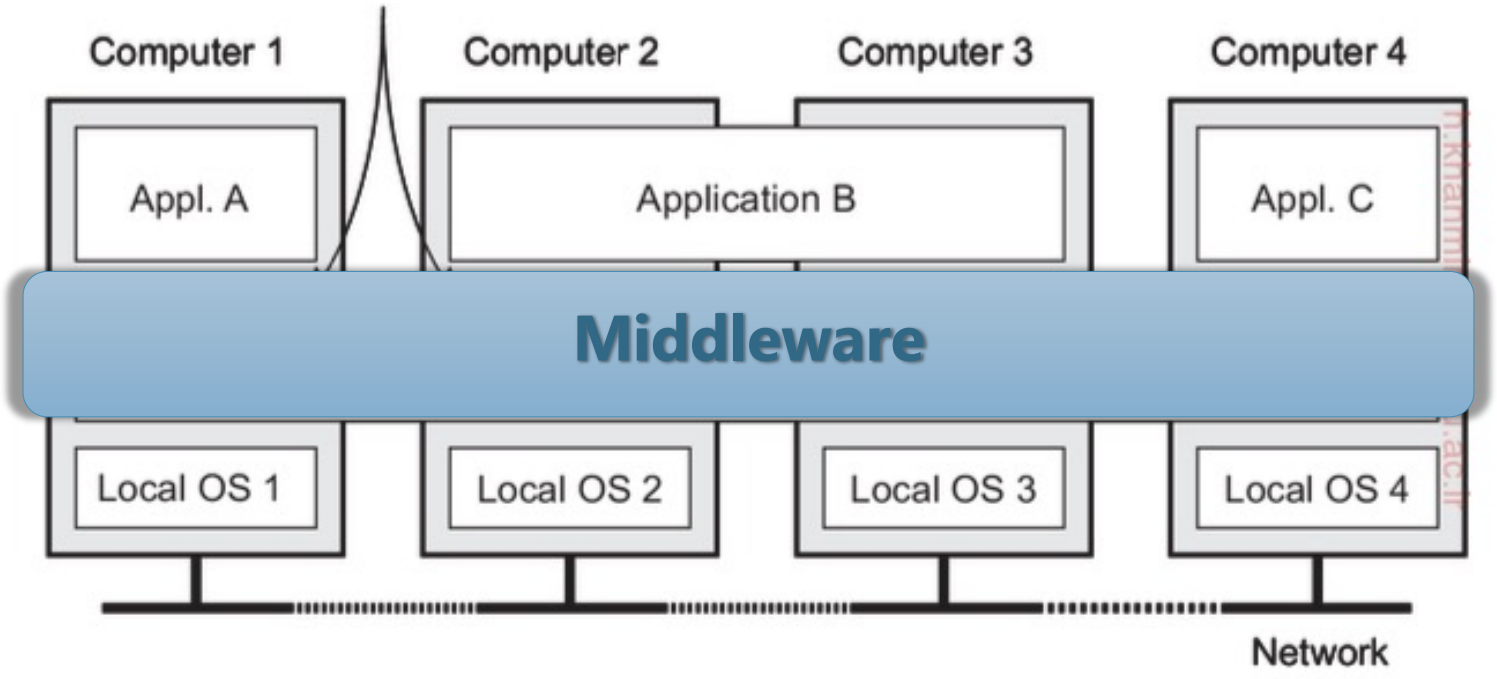
Distributed System

- Viewing a single system
  - A distributed system is coherent if it behaves according to the expectations of its users
    - This is challenging
  - Distribution Transparency
    - In UNIX all resources are viewed like a file
  - Fault Tolerance

Distributed System

- Middleware
  - Middleware → Distributed System like OS → Hardware
  - An abstraction layer to provide common services for a distributed system
- Services
  - Facilities for inter-application communication
  - Security services
  - Accounting services
  - Masking of and recovery from failures
- Examples
  - RPC
  - Transaction → All or nothing service execution
  - Service Composition
  - Reliability

Distributed System



Distributed System- Middleware

- Middleware (cont.) vs. Distributed Operating System
  - Middleware is linked to application over local OS
  - Distributed Operating System is an OS that is installed on all nodes and composed of various services like middleware

When We Build a Distributed System?

Goal 1 – Resource Sharing

- Support user access to remote resources
  - Like printers, data files, web pages, CPU cycles
  - Providing fair share of resources
- Why Sharing is important
  - Economics of sharing expensive resources
  - Connecting users and resources makes it easier to collaborate and exchange information
    - shared memory
    - P2P file sharing
  - Performance enhancement i.e. multiple processors (i.e. Grid)
- Resource sharing introduces security problems

Goal 2 – Distribution Transparency

- To hide the fact that processes and resources are physically distributed across multiple computers possibly separated by large distances, from end user
  - Makes the system more user friendly
- Transparency has several dimensions

Goal 2 - Types of Transparency

Transparency	Description
Access	Hide differences in data representation & resource access (enables interoperability)
Location	Hide location of resource (can use resource without knowing its location)
Migration	is offered by a distributed system when it supports the mobility of processes and resources initiated by users.
Relocation	Hide that resource may be moved while in use
Concurrency	Hide the possibility that the resource may be shared concurrently
Failure	Hide failure and recovery of the resource. How does one differentiate betw. slow and failed?
Replication	Hide the possibility that multiple copies of the resource exist (for reliability and/or availability)

Goal 3 - Openness

- An **open** distributed system is essentially a system that offers components that can easily be used by, or integrated into other systems.

- An open distributed system itself will often consist of components that originate from elsewhere.

- Service interfaces provide this functionality that should be clearly specified and freely available

Goal 3 – Openness - Characteristics

- Openness implies Interoperability, composability, and extensibility
  - Interoperability
    - Two different processes on different platforms should be able to talk together
  - Composability
  - Extensibility
    - Easy to add new components, features or replace an existing one

- Problem: Different platforms with different Oses have different characteristics
  - Endian problem
  - Supporting protocols
  - Data representation

Goal 3 - Openness

- Solution: define service interfaces with an open and platform-independent language

- Interface Definition/Description Language (IDL)
  - Describes software component interfaces (API)
  - Definitions are language- & machine-independent
    - Supports communication between processes running on different OSes written with any programming language

- Examples:
  - OMG IDL: CORBA Interface Description Language
  - WSDL: Web Services Description Language
    - Provides machine-readable descriptions of the services
  - Google Protocol Buffers & Flat Buffers
  - JSON , ...

IDL Languages - WSDL

```
<wsdl:definitions targetNamespace="http://math.example.com" name="MathFunctionsDef">
<wsdl:message name="addIntResponse">
        <wsdl:part name="addIntReturn" type="xsd:int" />
</wsdl:message>
<wsdl:message name="addIntRequest">
        <wsdl:part name="a" type="xsd:int" />
        <wsdl:part name="b" type="xsd:int" />
</wsdl:message>
<wsdl:portType name="AddFunction">
        <wsdl:operation name="addInt" parameterOrder="a b">
                <wsdl:input message="impl:addIntRequest" name="addIntRequest" />
                <wsdl:output message="impl:addIntResponse" name="addIntResponse" />
        </wsdl:operation>
</wsdl:portType>
<service name="MathFunctions"/>
</wsdl:definitions>
```

IDL Languages - JSON

```
{"employees":[
    {"name":"Sam", "email":"someone@gmail.com"},
    {"name":"Bob", "email":"ha32@gmail.com"},
    {"name":"Jai", "email":"ja99@gmail.com"}
 ],
 “organization”: {
     ”name” : ”some-org”,
     “location” : “some-where”
     “established-date” : “1390”
 }
}
```

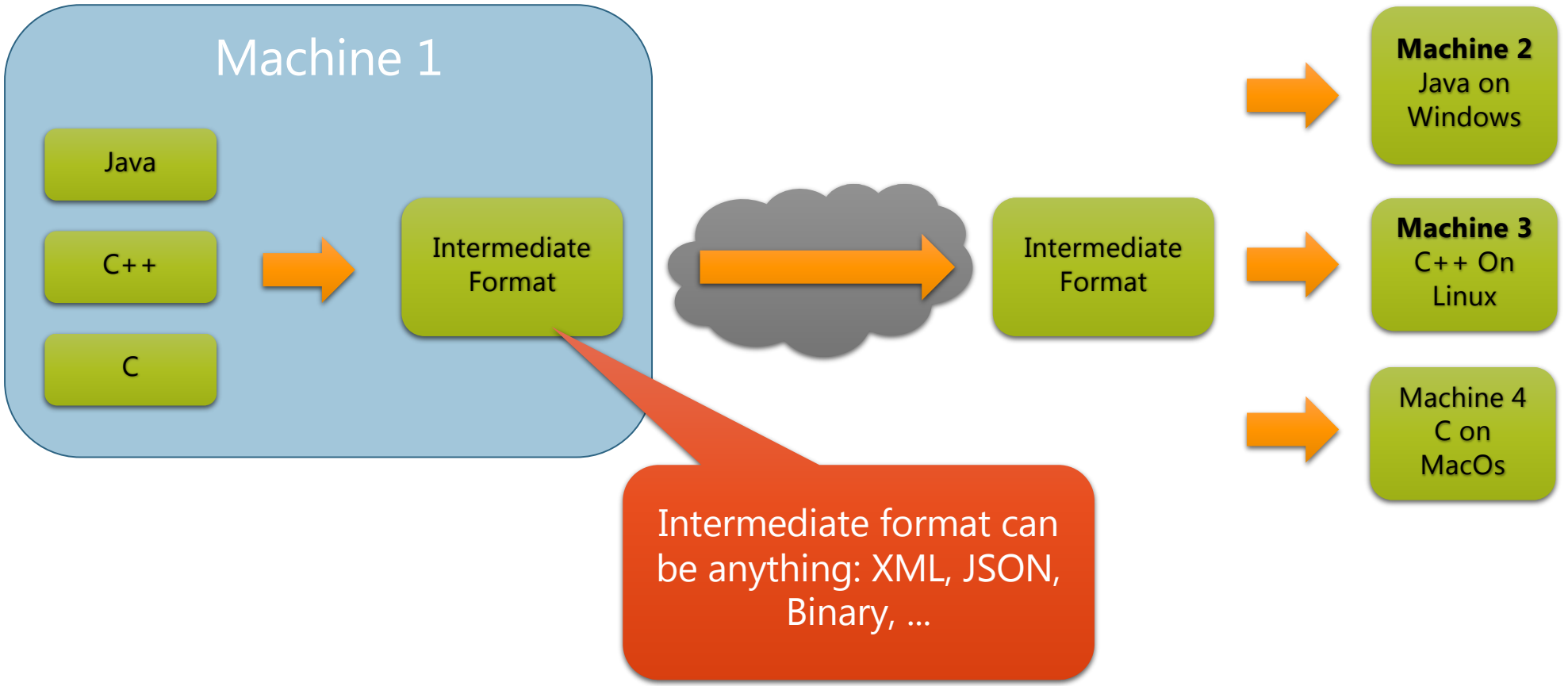
IDL Languages - Protocol Buffers

```
message Person {
        required string name = 1;
        required int32 id = 2;
        optional string email = 3;

        repeated string numbers = 4;
    }

message AddressBook {
        repeated Person people = 1;
    }
```

Using IDLs

Process of developing with IDL

Using IDLs


Machine 1
Java
C++
C
Intermediate Format
Intermediate Format
Machine 2 Java on Windows
Machine 3 C++ On Linux
Machine 4 C on MacOs
Intermediate format can be anything: XML, JSON, Binary, ...

IDL Languages - Protocol Buffers

```
> protoc -I=. --java_out=. addressbook.proto
```

```
AddressBookProtos.Person person =
        AddressBookProtos.Person.newBuilder()
          .setId(12)
          .setName(“ali”)
          .setEmail(“ali@some.com”)
          .addNumbers(“09121234567”)
          .build();

AddressBookProtos.AddressBook ab = AddressBookProtos.AddressBook
      .newBuilder().addPeople(person).build();
ab.writeTo(new FileOutputStream(filePath));

AddressBookProtos.AddressBook ab = AddressBookProtos.AddressBook
           .newBuilder()
           .mergeFrom(new FileInputStream(filePath)).build();
```

Goal 4 - Scalability

- Dimensions that may scale:
  - Size
    - Add more users and resources without sacrificing performance
  - Geographical
    - resources and users may be distributed in long distances. Delays and losses should not affect the system performance
  - Administrative
    - Still be easily managed even if it spans many independent administrative organizations.
- A scalable system still performs as well as before when scales up along any of the three dimensions.

Scaling Techniques

- Hiding Communication Latencies
  - Use asynchronous communication, do other works while waiting for server response
  - Move parts of computation closer to the user
    - client-side computation (i.e. JavaScript for form validation)
- Distribution of Work
  - Partition the work among several computation units
    - DNS

Scaling Techniques

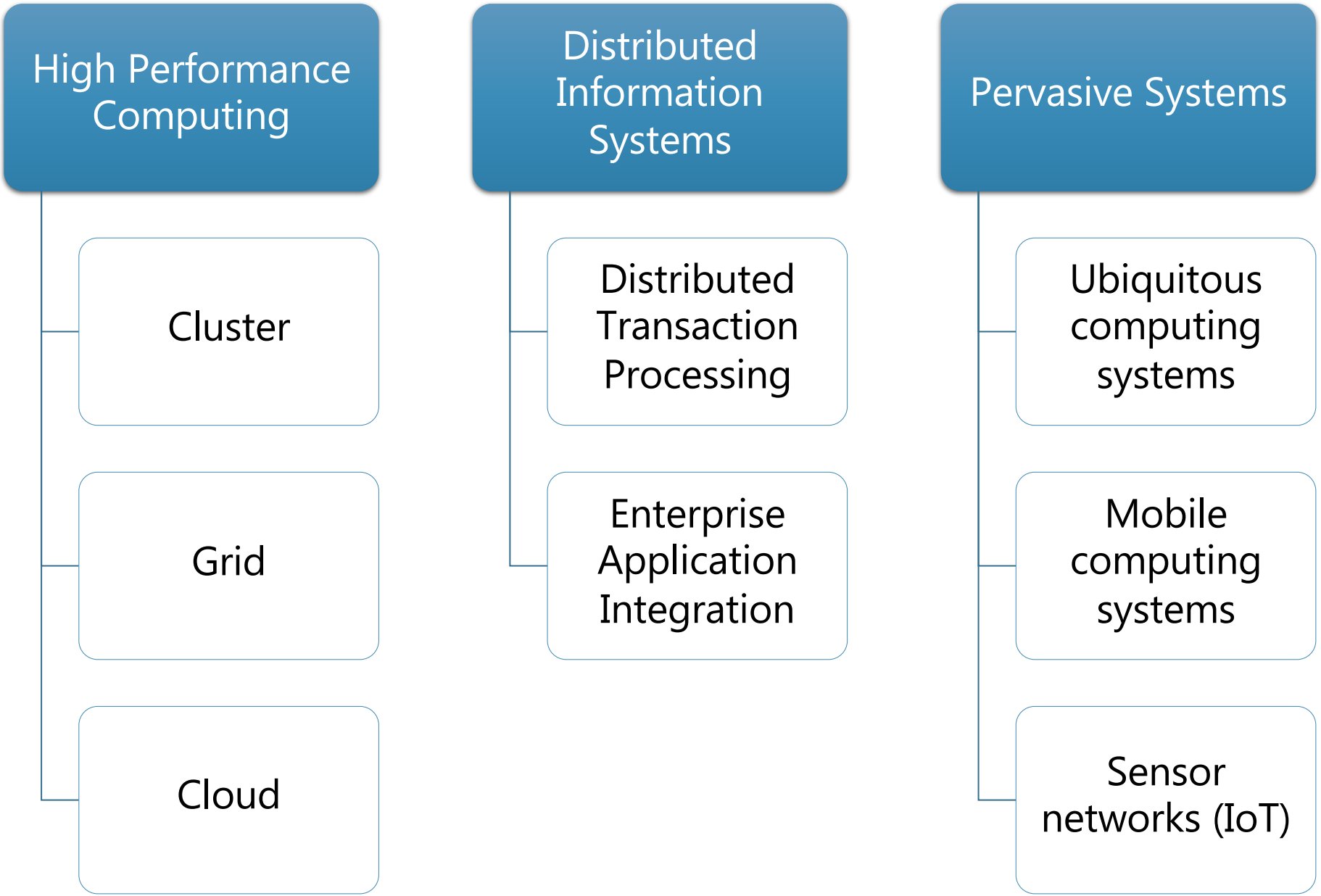
- Replication
  - Make multiple copies from a resource
    - Improves High Availability
    - Improves Communication Latencies (i.e. CDN)
- Caching
  - Special form of replication where decision made by the client of a resource and not by the owner of a resource.

Scaling Techniques

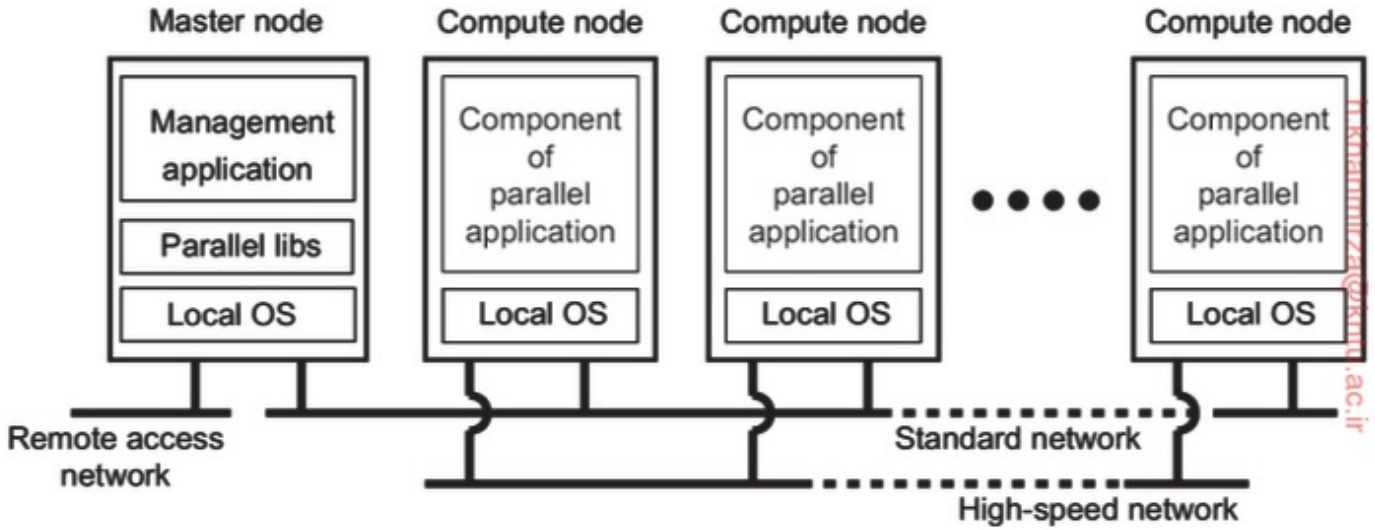
- Replication introduces some new issues → Consistency!
  - Modifying a copy makes that copy different from the others
  - Degree of consistency → Strong vs. Weak
    - First page of google → not so important → weak
    - Electronic stock exchanges → very important → strong
  - Strong consistency needs
    - modifications propagate immediately to all other copies
    - Ordering in concurrent copies
  - Scaling by replication may introduce other, inherently non-scalable solutions

Types of Distributed Systems

Types of Distributed Systems

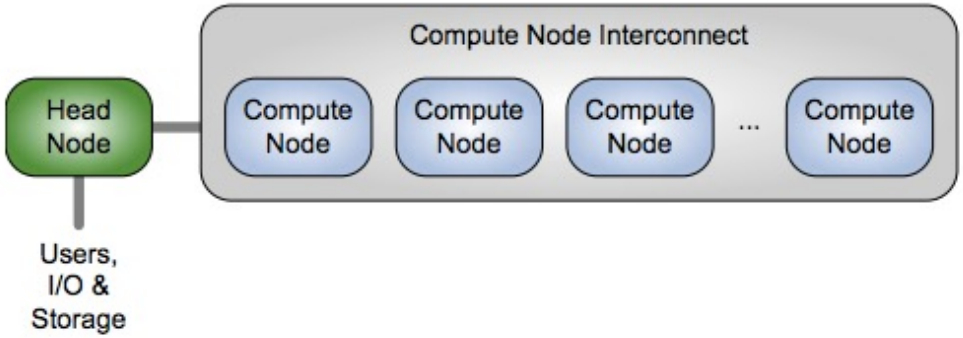


Cluster



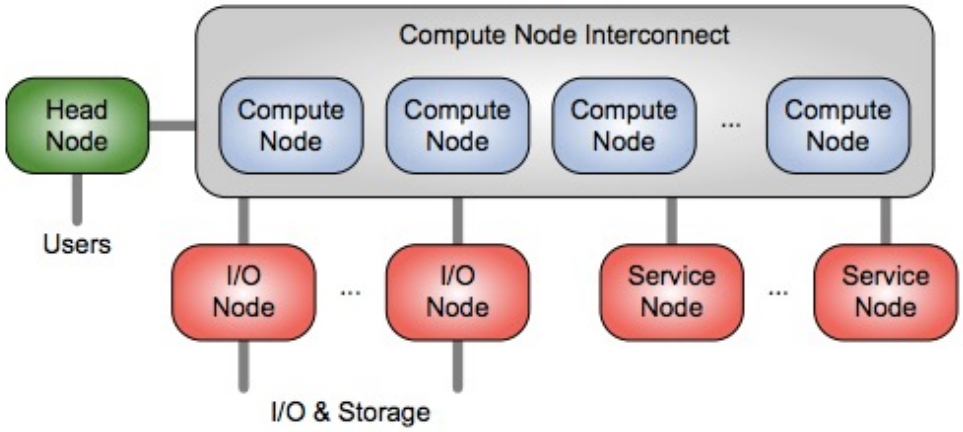
- Collection of systems with similar hardware and OS connected in a LAN (or a geographically-limited area)

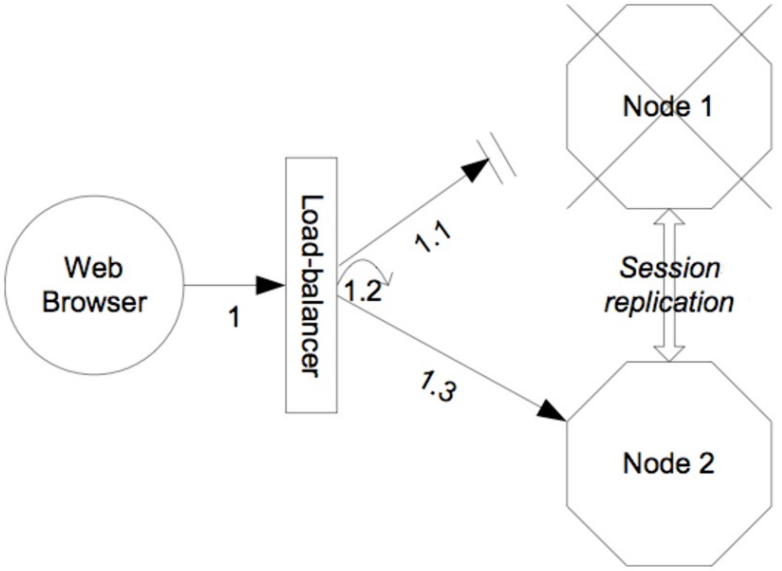
Cluster



- Traditional Beowulf Cluster System Architecture
- Master-slave model
  - Master is the interface between user and all middleware services
  - Master distributes tasks among all systems and gathers results and save
  - Compute nodes run also some middleware functions: fault tolerance

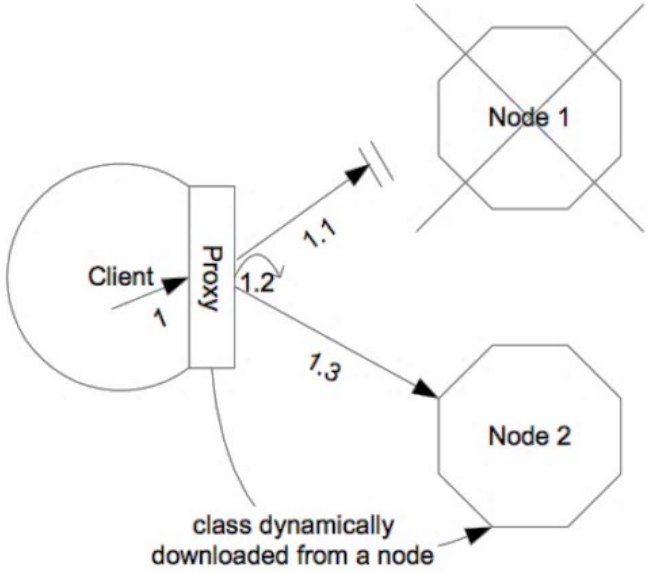
Cluster





- Generic Modern HPC System Architecture
  - Head node can do several functions like Load balancing, filtering, security and access control, ⋯
  - Compute node
    - Runs a lightweight kernel and absolutely minimal necessary libraries.
    - Maybe partitioned into several task groups

Cluster



- Generic Modern HPC System Architecture (cont.)
  - Load balancing and some functions may be offloaded to the client
  - A proxy is downloaded into the client in the first contact
    - Proxy keeps an updated status of the cluster nodes

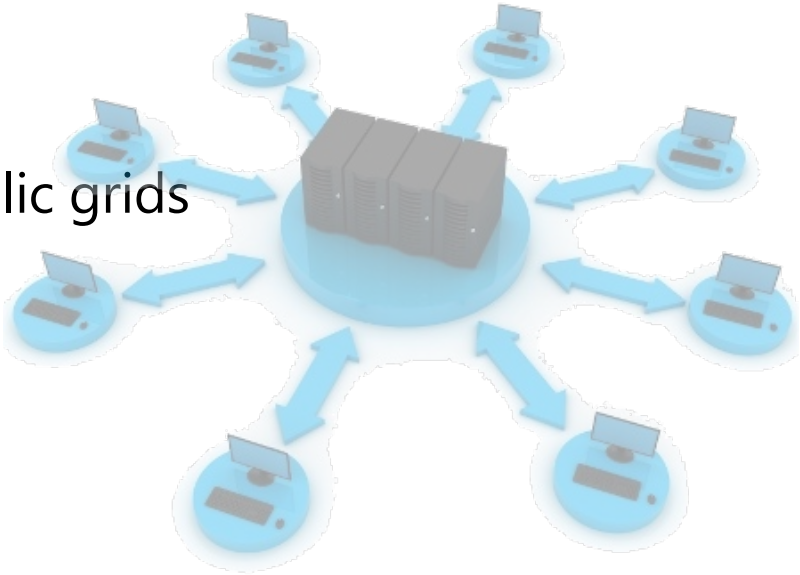
Grid

- Collection of different computing systems with various OSes over a large geographical distant and different administrative domains (i.e. the Globe)

- Grid computing is a network in which machine resources are shared (computing, sensors, ⋯)

Grid

- Open Grid Services Architecture (OGSA)
  - Standardization of grid computing: accessing, security, read/write data, call functions, ⋯
  - Service-oriented architecture
    - Generally follow Web Services standards
- Pros
  - Fail-safe
  - Cheaper resources
  - More efficient using idle-resources
- Cons
  - Licensing problem
  - Hard to get guarantee, can be slow in public grids
  - Apps may need re-coding
  - Need fast interconnections
  - Security and privacy

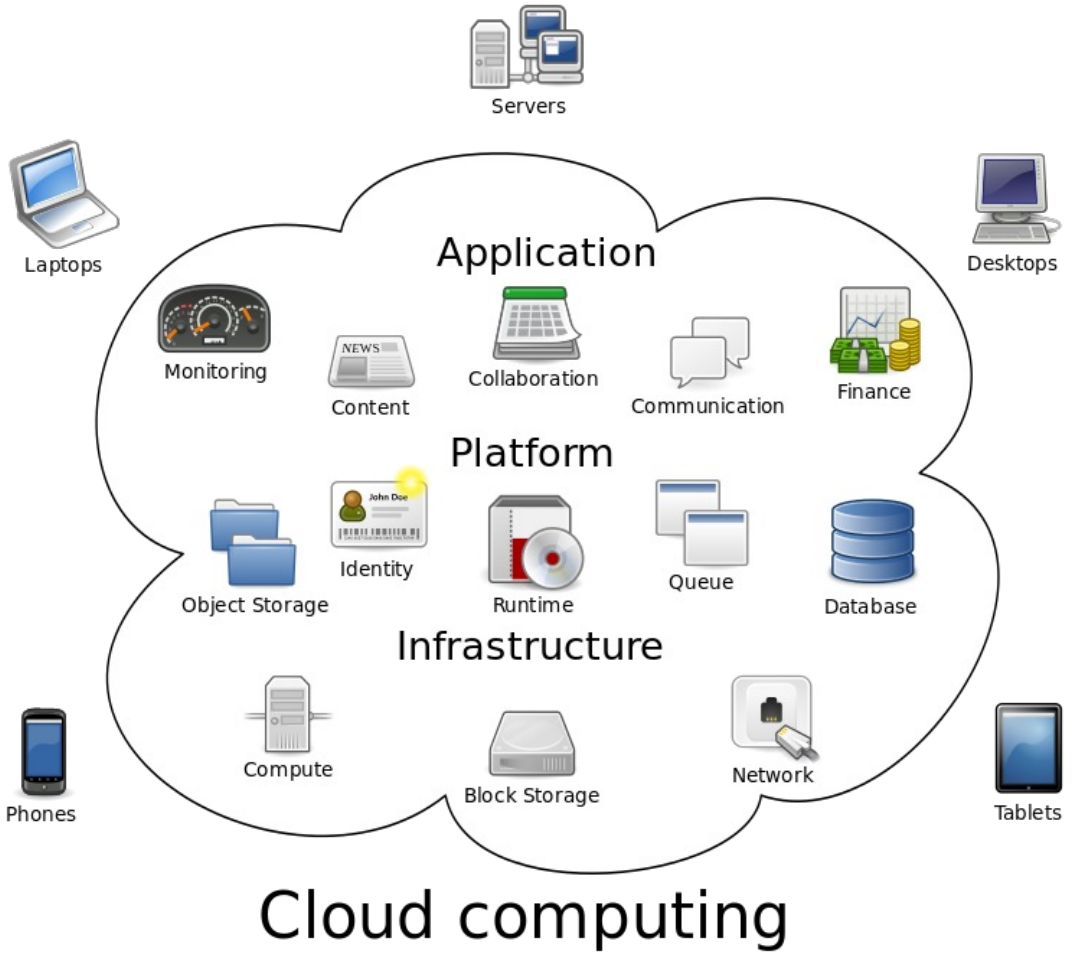


Cloud

- Cloud computing is characterized by an easily usable and accessible pool of virtualized resources.
- Resources can be configured dynamically
  - Provides the basis for scalability, if more work needs to be done, a customer can simply acquire more resources
- Cloud computing is generally based on a pay-per-use model
  - Guarantees are offered by means of customized service level agreements (SLAs).

Cloud

- Clouds are organized in 4 layers
  - Hardware (datacenters)
    - processors, routers, disk also power and cooling systems. (shared hosts, dedicated servers, VPS)
  - Infrastructure
  - Platform
  - Application

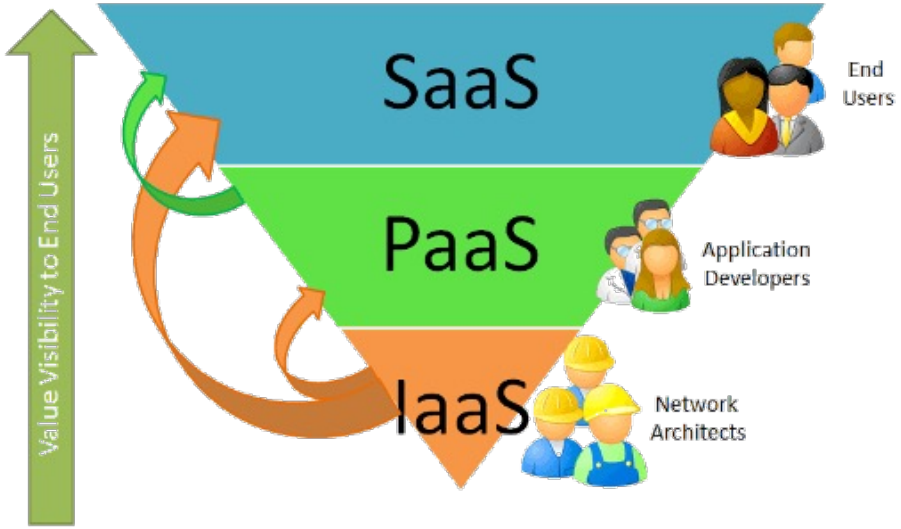


Cloud

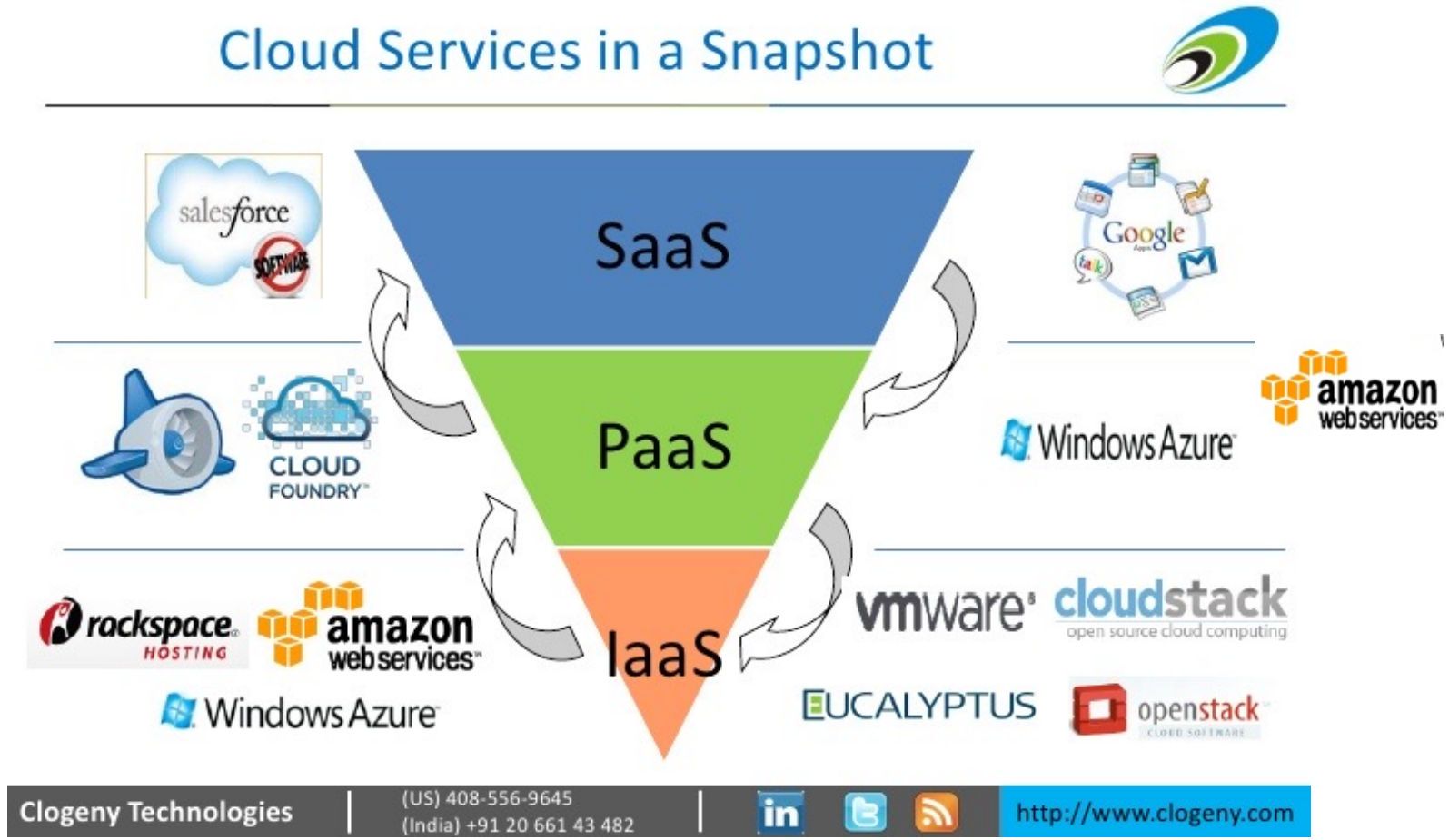
- Infrastructure
  - The backbone of the cloud computing.
  - It deploys virtualization techniques to provide customers an infrastructure consisting of virtual storage and computing resources.
  - Cloud computing evolves around allocating and managing virtual storage devices and virtual servers
- Platform
  - Like an OS for App developers to easily deploy apps
  - Vendor-specific API to access and use DB, Network, web-server…
  - No need to setup, config, everything is ready to use
- Application
  - Applications run on cloud platform
    - spread-sheets, word processing, collaboration, project management, …

Cloud

- Cloud Service Types
  - Software-as-a-Service (SaaS)
    - Cloud applications
  - Platform-as-a-Service (PaaS)
    - Covering the platform layer
  - Infrastructure-as-a-Service(IaaS)
    - Covering the hardware and infrastructure layer



Cloud


Cloud Services in a Snapshot
SaaS
PaaS
IaaS
salesforce
Google
CLOUD FOUNDRY
Windows Azure
amazon web services
rackspace HOSTING
amazon web services
Windows Azure
vmware
cloudstack
open source cloud computing
EUCALYPTUS
openstack
Clogeny Technologies
(US) 408-556-9645
(India) +91 20 661 43 482
http://www.clogeny.com

Cloud

- Pros
  - Can be cheaper
    - You may choose which components to be local and which be on cloud or distribute over several clouds to manage costs!
  - Disaster Recovery
  - Increased flexibility
  - Focus on business
- Cons
  - Vendor Lock-in
  - Internet Connectivity
  - Security & Privacy concerns

Distributed Information Systems

- Applications became more sophisticated and are gradually separated into independent components.

- Sometimes, we need special relations between components (= services)

Distributed Transactions

- All or none semantics

- ACID properties:
  - 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

- When components are distributed, this task become more complicated

Enterprise Application Integration (EAI)

- Basic model of Distributed Systems is Shared DB.
- As applications become more complex, there is a need to connect components directly
  - Remote Procedural Calls (RPC)
- Undirect communication → Messaging
  - Publish/subscribe
    - Can send and receive messages even one of the comm. Sides is not available
  - Distributed Shared Memory

Pervasive Systems

- Previous systems assume
  - Nodes are fixed
  - Nodes have a more or less permanent and high-quality connection to a network
- These assumptions are no longer valid when nodes are embedded systems and/or mobile.
- Pervasive Systems have no special assumption on nodes
  - Battery-powered or plugged in
  - Wireless connection or wired
  - Small-range connection or long-range
  - Where the nodes are resided
- Very close to today's Internet-of-Things concept

Pervasive Systems

- It is considered a distinction between three types of pervasive networks
  - Ubiquitous computing systems
  - Mobile systems
  - Sensor networks

Ubiquitous Computing

- The system is pervasive and continuously present
  - A user continuously interacting with the system, often not even being aware that interaction is taking place!
- Requirements
  - Distribution:
    - Single coherent system of users, sensors, actuators, computation elements
  - Interaction:
    - Interactions mostly should be implicit. User is not aware that his actions give inputs to the system
      - Sitting in the car, seats are adjusted by person
  - Context Awareness:
    - Context: where, who, when, and what
    - Collected data how should be used
      - Face detection results how should be used

Ubiquitous Computing

- Autonomy:
  - Devices operate autonomously without human intervention, and are thus highly self-managed.
    - Address allocation → DHCP
    - Automatic Updates
- Intelligence
  - The system as a whole can handle a wide range of dynamic actions and interactions
    - often methods and techniques from the field of artificial intelligence is used

Mobile Computing

- Characteristics
  - Connected devices vary widely
  - Devices are mobile: location is changed over time
  - Common issues
    - Finding current location of nodes
    - Routing
  - Examples
    - Disruption-tolerant networks (links are not permanent)
    - MANET: Mobile Ad-hoc Networks
    - VANET: Vehicular Ad-hoc Networks
    - Pocket-Switched Networks
      - Networks formed by people!
      - Combination of Social & Data networks
      - Human mobility is mostly predictable: People return to the same place after 24, 48 hours

Sensor Networks

- Sensor nodes often collaborate to efficiently process the sensed data in an application-specific manner

- Sensor networks use wireless communication, and nodes are often battery powered.
  - Their limited resources, restricted communication capabilities, and constrained power consumption demands highest possible efficiency in their design