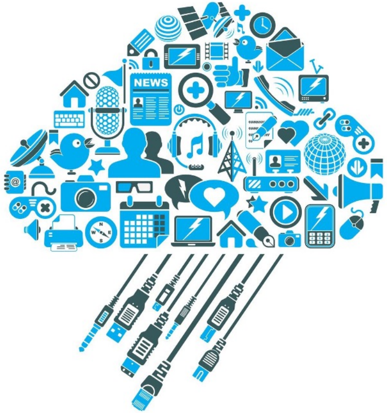
University of CINCINNATI



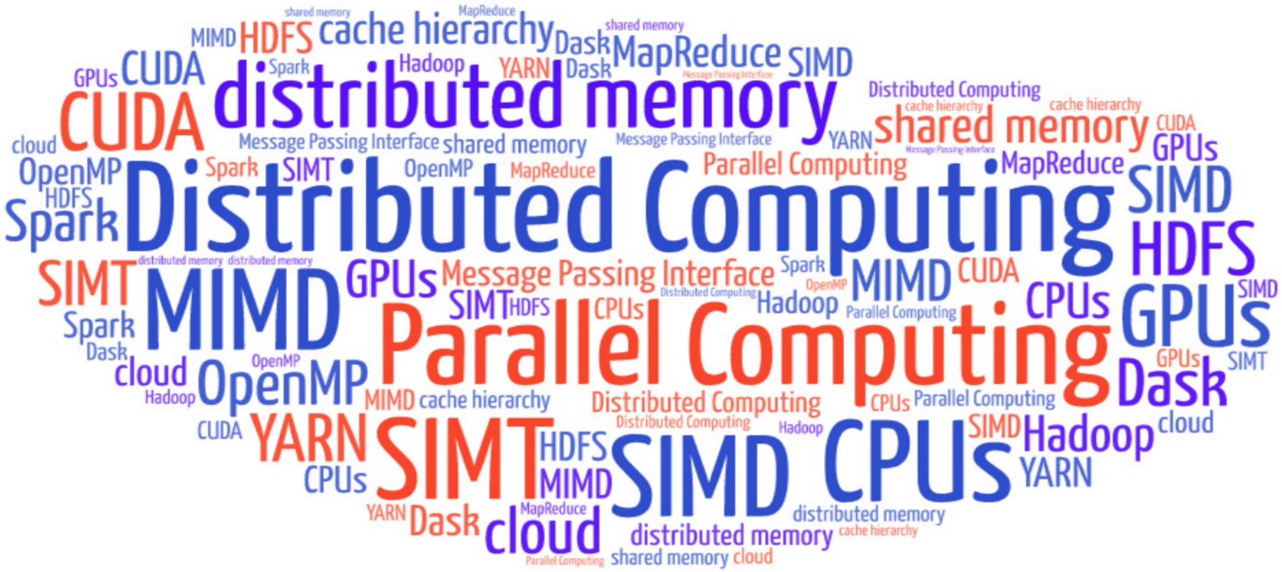
Principles of Distributed Computing

(Mastering Cloud Computing: Chapter#2)

Rashmi Kansakar

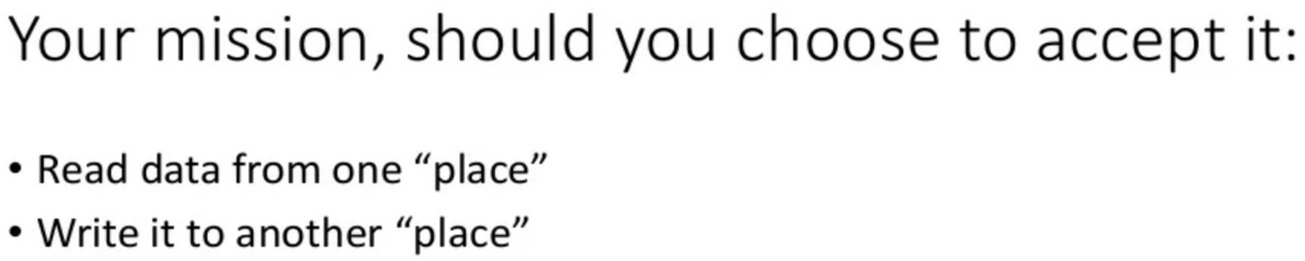
What is Distributed system?

You know you have a distributed system when the crash of a computer you've never heard of stops you form getting any work done. - Leslie Lamport



What is Distributed system?

You know you have a distributed system when the crash of a computer you've never heard of stops you form getting any work done. - Leslie Lamport



Your mission, should you choose to accept it:

- Read data from one "place"
- Write it to another "place"

Navigating 8 fallacies of Distributed Computing

1. The network is reliable.
2. Latency is zero.
3. Bandwidth is infinite.
4. The network is secure.
5. Topology doesn't change.
6. There is one administrator.
7. Transport cost is zero.
8. The network is homogeneous.

First 7 credited to Peter Deutsch, 1994, Sun
8th credited to James Gosling, 1997 , Sun, Java

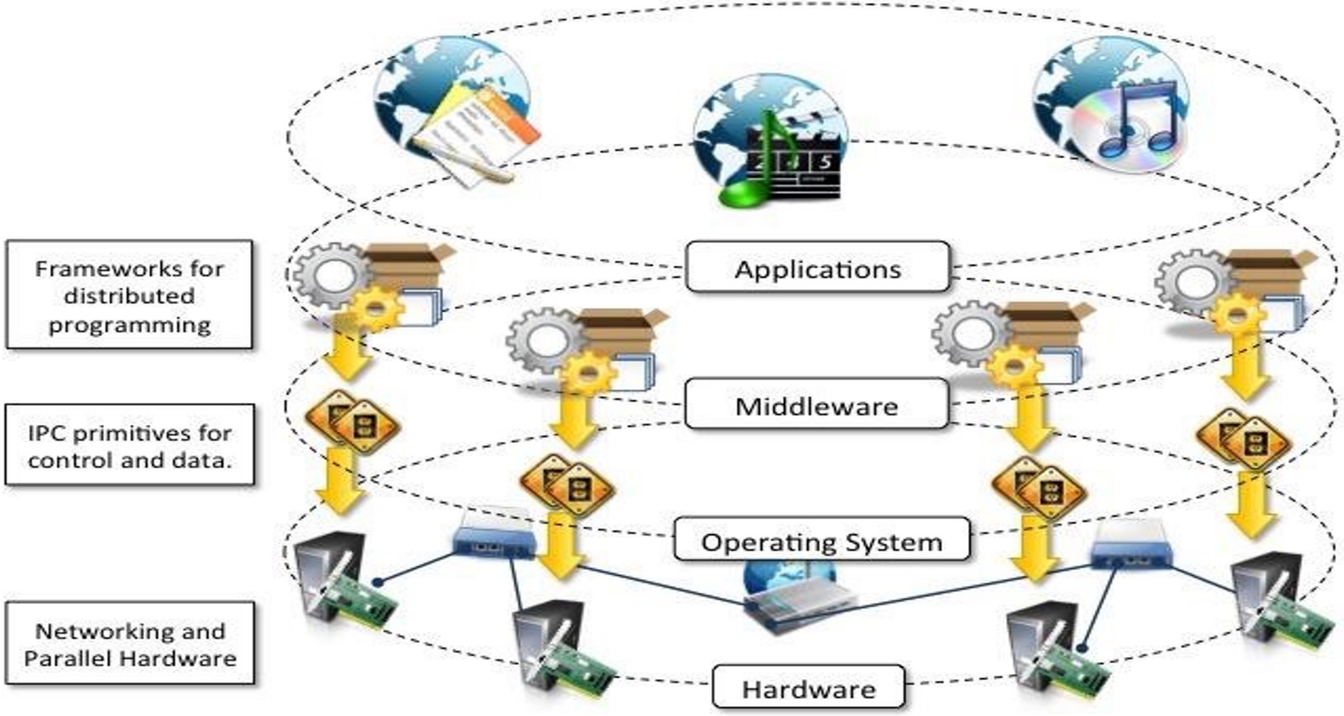
Elements of Distributed Computing

A distributed system is a collection of independent computers that appears to its users as a single coherent system - **Tanenbaum et al**

A distributed system is one in which components located at networked computers communicate and coordinate their actions by passing messages. - **Coulouris et al**

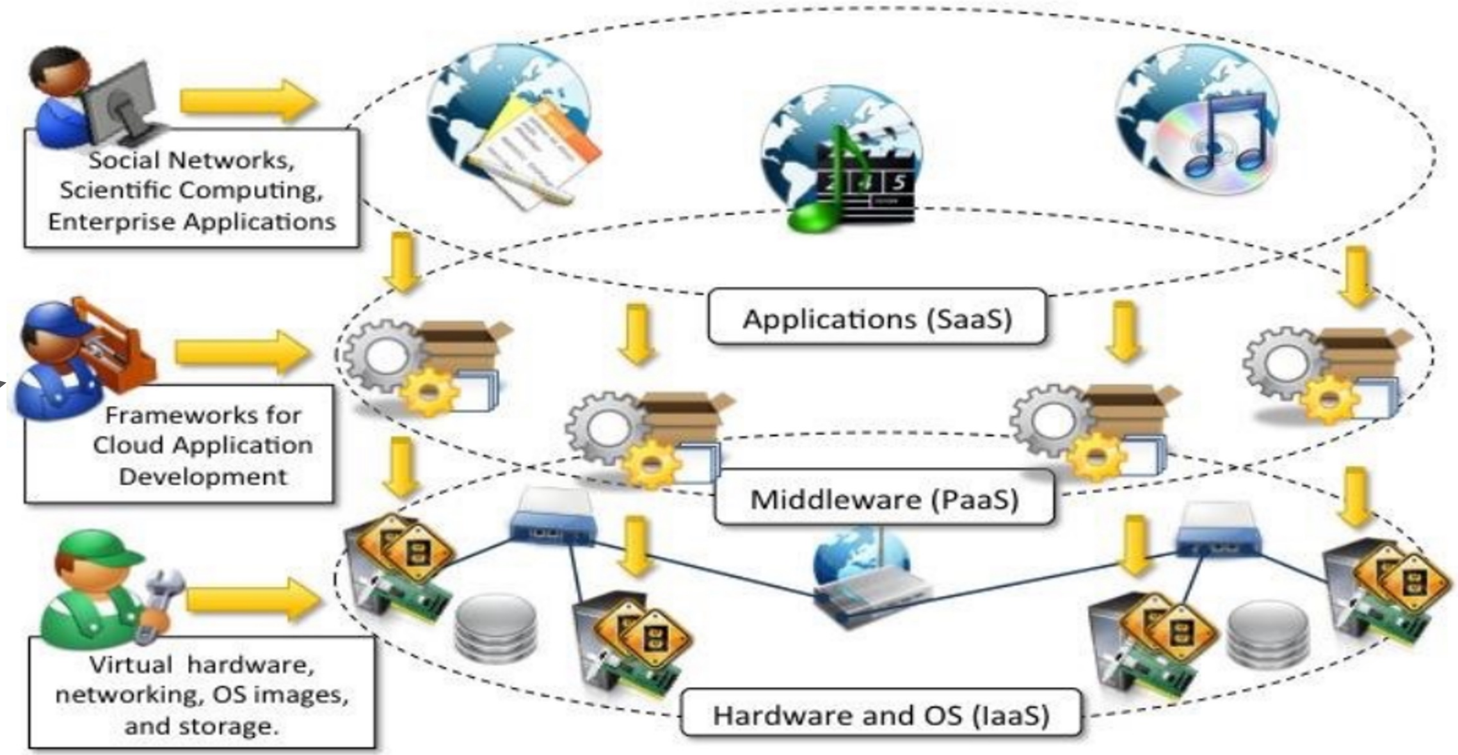
Message passing!!!!

Distributed System Layers


Frameworks for distributed programming
IPC primitives for control and data.
Networking and Parallel Hardware
Applications
Middleware
Operating System
Hardware

Cloud Computing → distributed system

Example


Social Networks, Scientific Computing, Enterprise Applications
Applications (SaaS)
Frameworks for Cloud Application Development
Middleware (PaaS)
Virtual hardware, networking, OS images, and storage.
Hardware and OS (IaaS)

Middleware enables distributed computing

Architectural Styles for Distributed Computing

Architectural styles are mainly used to determine the vocabulary of components and connectors that are used as instances of the style together with a set of constraints on how they can be combined.

Two major classes of architectural style.

- Software architectural styles (software organization)
- System architectural styles (physical organization)

Components and Connectors

- Component : software that encapsulates a function or feature of the system: programs, objects, processes.

- Connector : communication mechanism between components, can be implemented in a distributed manner.

Characteristic of Distributed Systems

- **No Shared Clock** order events - logical clock is achieved by synchronization/coordination
- **No Shared Memory** state is distributed throughout the system
- **Concurrency** tasks are executed concurrently
- **Heterogeneity and Loose Coupling** not required - different OSs and technology

Difference between Parallel & Distributed Computing

Parallel Computing	Distributed Computing
Many operations are performed simultaneously	System components are located at different locations
Single computer is required	Uses multiple computers
Multiple processors perform multiple operations	Multiple computers perform multiple operations
It may have shared or distributed memory	It have only distributed memory
Processors communicate with each other through bus	Computer communicate with each other through message passing.
Improves the system performance	Improves system scalability, fault tolerance and resource sharing capabilities

Software Architectural Styles

Category	Common Architectural Styles
Data-center	Repository
	Blackboard
Data Flow	Pipe and filter
	Batch sequential
Virtual Machine	Rule-based system
	Interpreter
Call and return	Main program and subroutine/top down
	Object-oriented systems
Independent components	Communicating processes
	Event systems

Data Centered Architectures

- Data and access to shared data is core
  - Data integrity is goal
  - Ex: Gmail, Flickr, Google search, Salesforce, Oracle

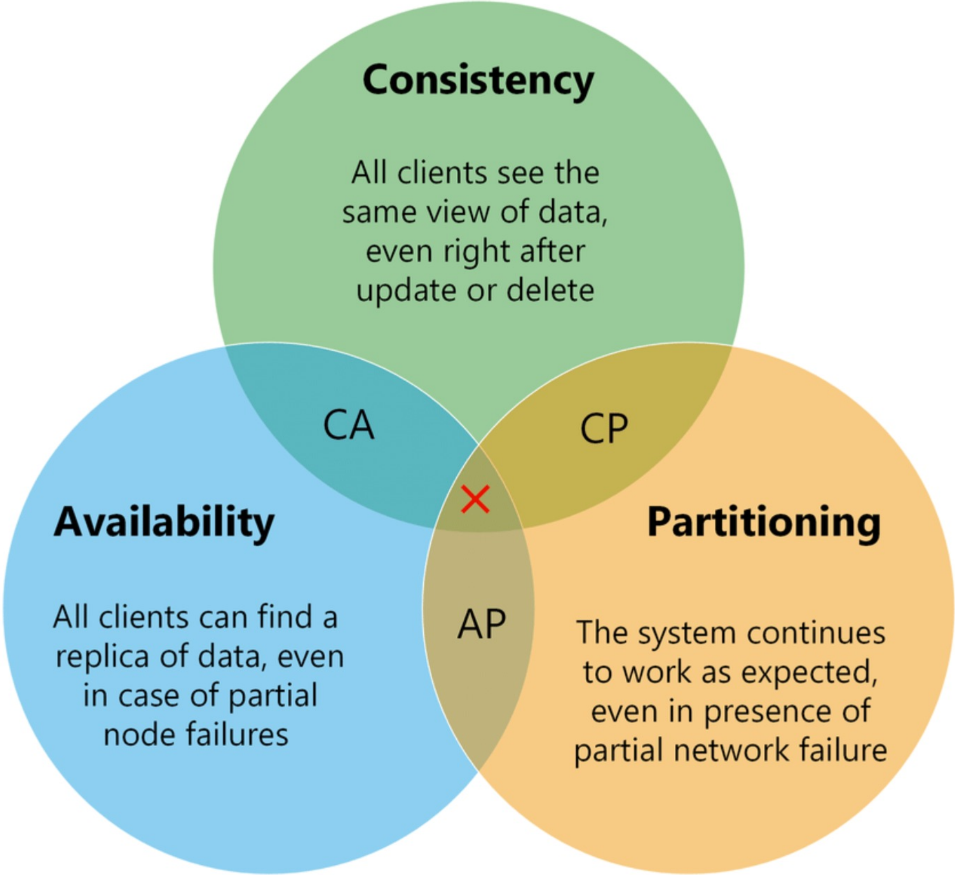
- Repository style
  - Central data structure - current state
  - Independent components - operate on data
  - 2 subtypes:
    - Database systems - components called & act on data
    - Blackboard systems - data-structure is trigger - if/then or expert-system feel - updates itself (example: speech recognition, signal processing)

Explanation

Database Systems

- Centralized access
- You control the access / management

Understanding CAP theorem


Consistency
All clients see the same view of data, even right after update or delete
CA
CP
✕
Availability
All clients can find a replica of data, even in case of partial node failures
AP
Partitioning
The system continues to work as expected, even in presence of partial network failure

What is the CAP theorem?

CAP represents:

1. Consistency
2. Availability
3. Partition tolerance

- You can pick 2 out of 3
- But not all 3 at the same time

Software Architectural Styles

Category	Common Architectural Styles
Data-center	Repository
	Blackboard
Data Flow	Pipe and filter
	Batch sequential
Virtual Machine	Rule-based system
	Interpreter
Call and return	Main program and subroutine/top down
	Object-oriented systems
Independent components	Communicating processes
	Event systems

Data-Flow Architectures

- Availability of data controls, data flows through system
- For when data size exceeds storage capacities, or when long-term storage is not needed
- 2 styles:
    - Batch Sequential - sequence of programs - must wait for previous to finish before next
        - Mainframes
        - Usually output to file, before another program starts
    - Pipe-and-Filter - sequence of programs, but FIFO queues to start processing before previous has finished.
        - Unix shell pipes and tools are good examples:
            - grep, sed, awk

Batch Sequential *Vs.* Pipe-and-Filter

Batch Sequential	Pipe-and-Filter
Coarse grained	Fine grained
High latency	Reduced latency due to incremental processing
External access to input	Localized input
No concurrency	Concurrency possible
Noninteractive	Awkward, but possible

Explanation

Software Architectural Styles

Category	Common Architectural Styles
Data-center	Repository
	Blackboard
Data Flow	Pipe and filter
	Batch sequential
Virtual Machine	Rule-based system
	Interpreter
Call and return	Main program and subroutine/top down
	Object-oriented systems
Independent components	Communicating processes
	Event systems

Virtual Machine Architectures

- Abstract execution environment - rule-based systems, interpreters, command-language processors (**2 types**):
- Rule Based:
  - Inference engine - AI - process control - network intrusion detection
  - Examples includes some of the Predix IoT analytics systems - https://github.com/PredixDev/predix-analytics-sample
- Interpreter:
  - Interprets pseudo-program - abstracts hardware differences away - Java, C#, Perl, PHP

Software Architectural Styles

Category	Common Architectural Styles
Data-center	Repository
	Blackboard
Data Flow	Pipe and filter
	Batch sequential
Virtual Machine	Rule-based system
	Interpreter
Call and return	Main program and subroutine/top down
	Object-oriented systems
Independent components	Communicating processes
	Event systems

Call & Return Architectures

- Components connected via method calls (**3 styles**):
  - **Top-Down:** imperative programming - tree structure - hard to maintain
  - **Object-Oriented:** coupling between data and manipulation operations - easier to maintain - method calling requires object - consistency an issue
  - **Layered Style:** Abstraction layers - modular design - hard to change layers
    - Ex: OS kernels, TCP/IP stack, web applications

Software Architectural Styles

Category	Common Architectural Styles
Data-center	Repository
	Blackboard
Data Flow	Pipe and filter
	Batch sequential
Virtual Machine	Rule-based system
	Interpreter
Call and return	Main program and subroutine/top down
	Object-oriented systems
Independent components	Communicating processes
	Event systems

Independent Components Architectures

- ➢ Life cycles to components (**2 styles**):
  - ○ Communicating Processes: good for distributed systems - concurrent - service based - IPC
  - ○ Event Systems: components have data and manipulation, but add event registering/triggering - callbacks - like layered, but looser connections - hard to reason about correctness of interactions

➔ ObjectiveC/C++ is a good example. Nodejs

System Architectural Styles

- ➢ Describe physical layout

- ➢ 2 styles:
  - ○ Client/Server
  - ○ Peer-to-peer

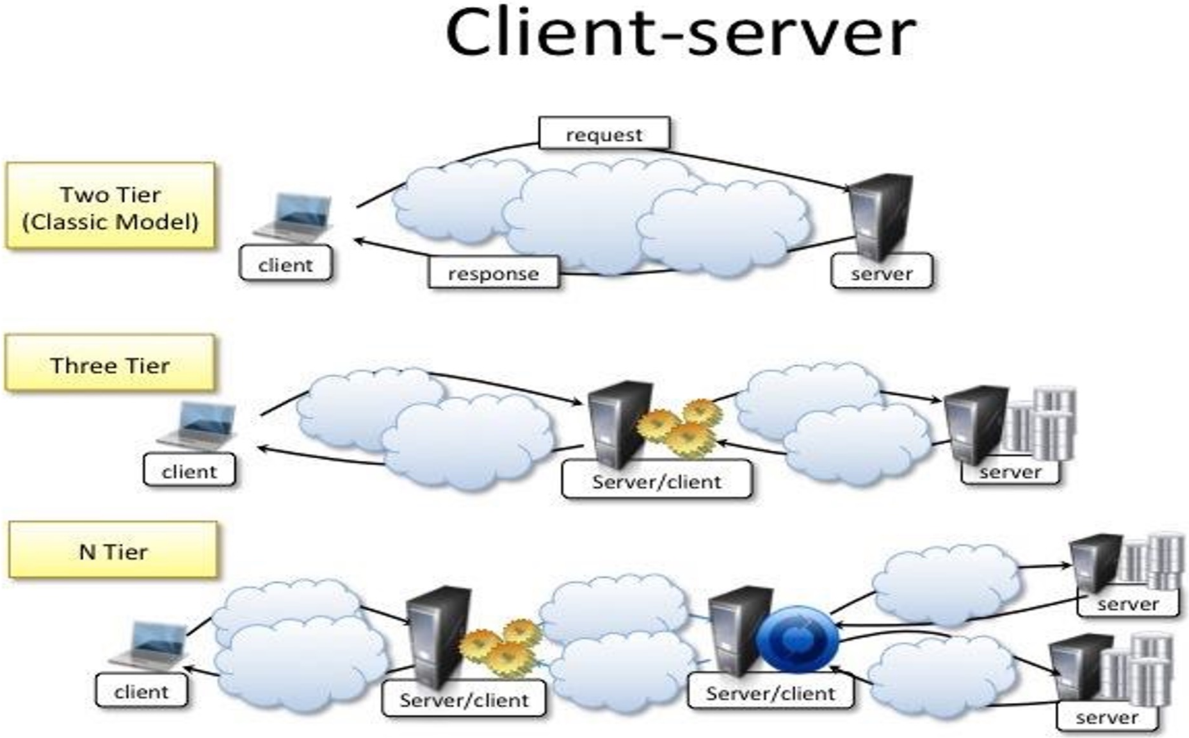
Client / Server

- Very Popular
- Request, accept (client)
- Listen, response (server)
- Suitable for many-to-one situations

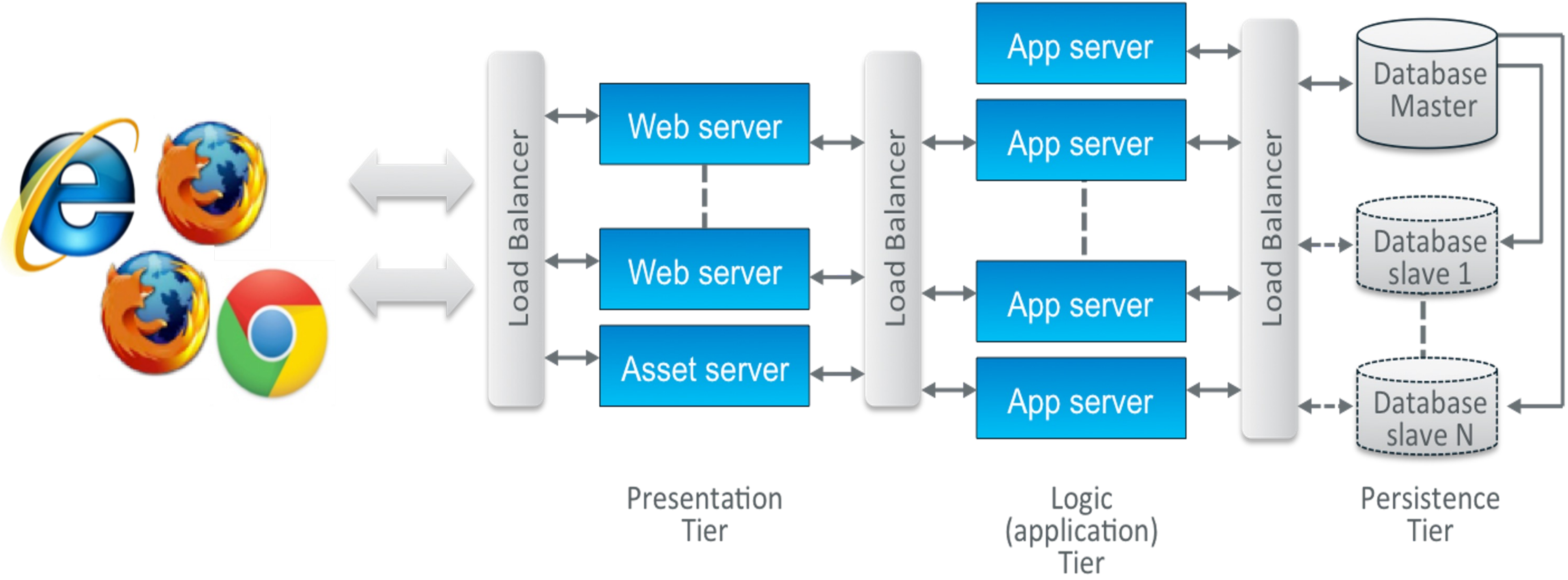
Types of Clients

- Thin-client: (nearly) all data processing done on server (plain HTML)
    - presentation on client
    - app logic and data storage on server
        - java applets in browser. web 2.0
- Fat-client: client processes and transforms data, server just gateway to access data (AJAX-like)
    - Presentation and app logic on client
    - Data storage on server
        - Windows Forms/ D2K

Layered Approach



4-Tier Application

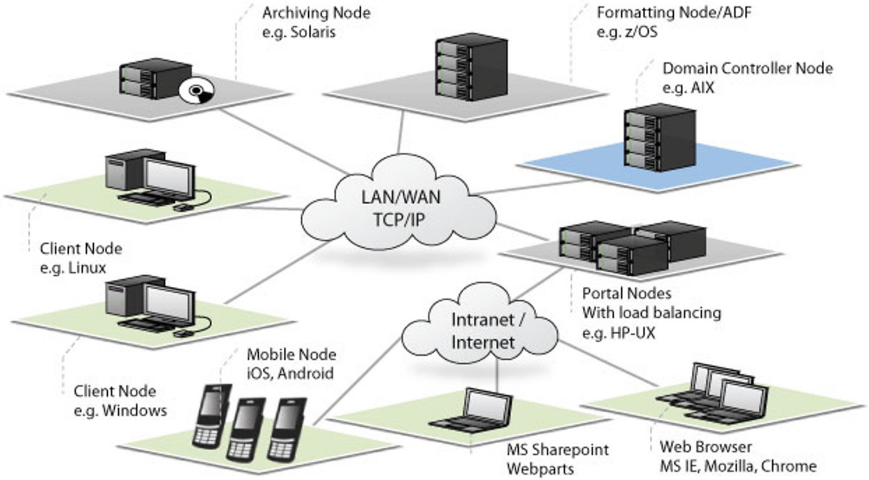


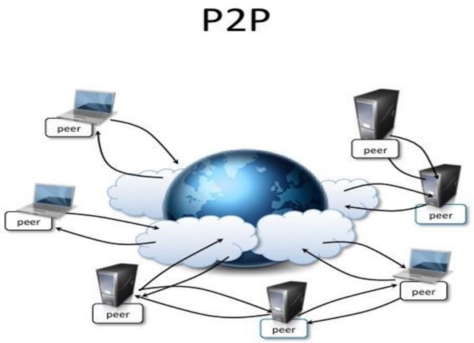
Layer Types

- **Two Tier**
  - Pros: Easier to build
  - Con: Doesn’t scale well
  - Ex: Small dynamic web applications

- **Three Tier/N Tier**
  - Pros: Scales better (add more servers to layer)
  - Con: Harder to maintain
  - Ex: Medium-Large dynamic web applications

Peer-to-Peer

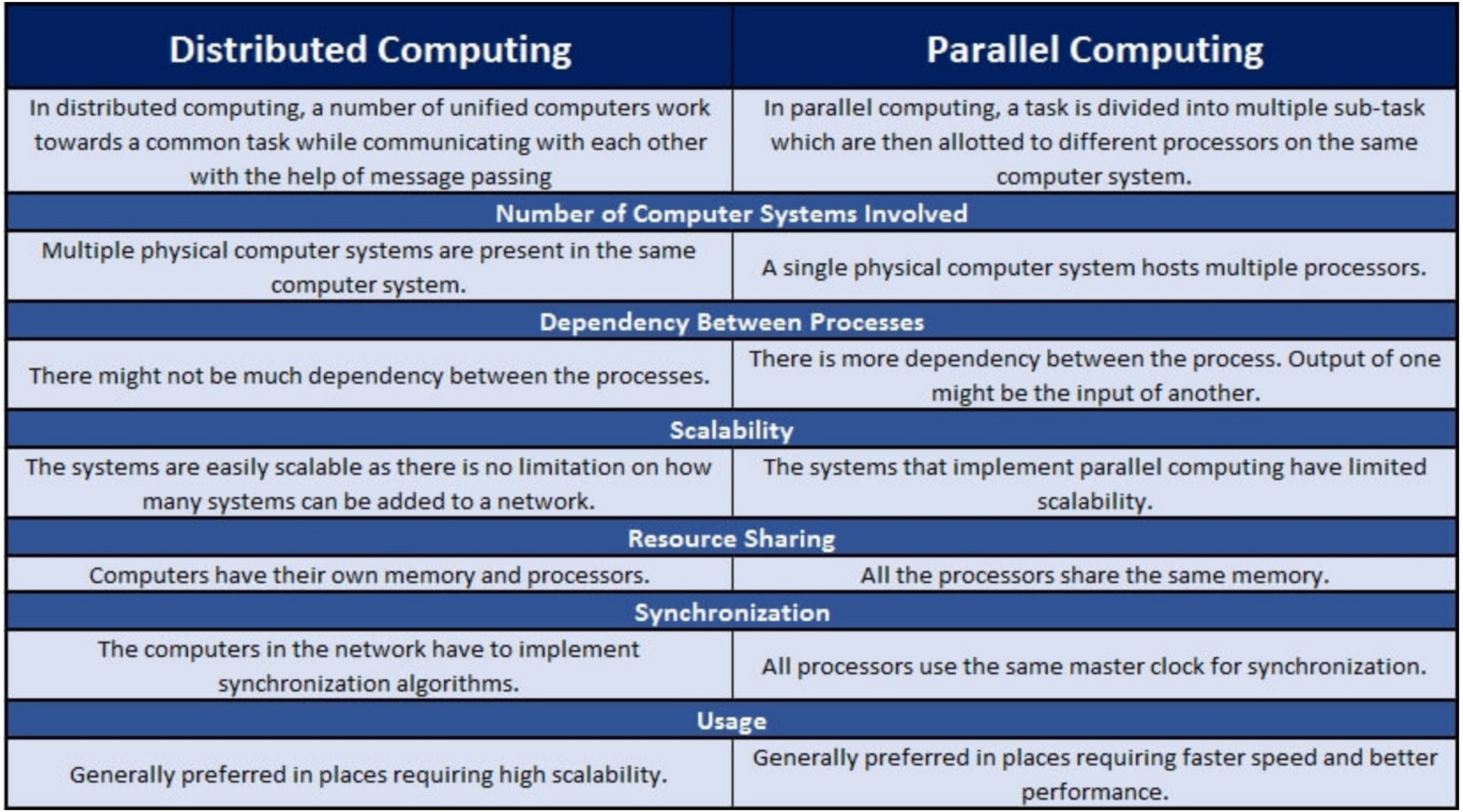




- ➢ Symmetric - everyone client and server
- ➢ Scales very well!
- ➢ Hard to build

- ➢ Used in data centers to distribute data!
  - ○ Ex: Gnutella, BitTorrent, Kazaa, Skype, Tor
- ➢ Multi-level roles possible: Kazaa

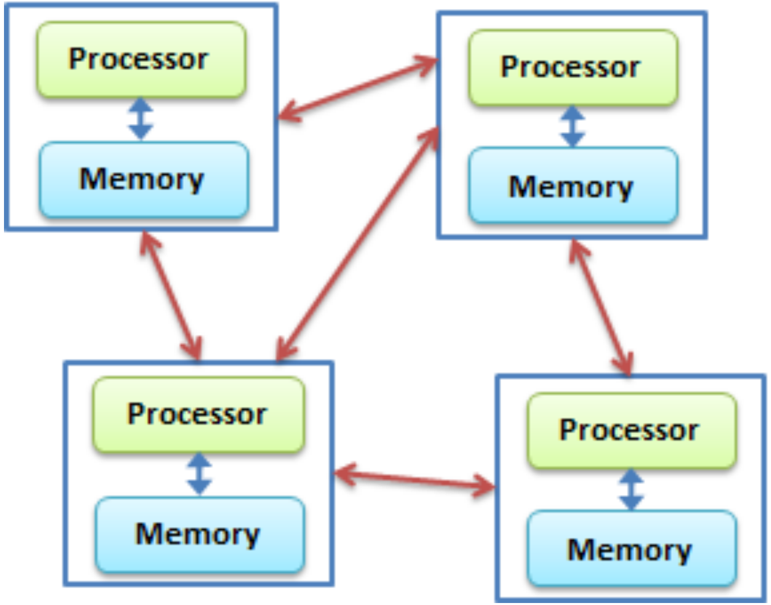
Parallel & Distributed Computing Comparison



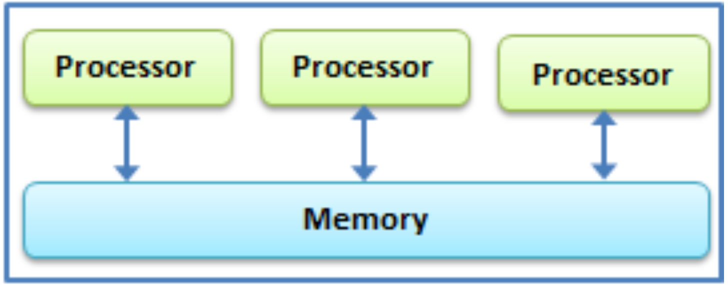
<table>
<tr><th>Distributed Computing</th><th>Parallel Computing</th></tr>
<tr><td>In distributed computing, a number of unified computers work towards a common task while communicating with each other with the help of message passing</td><td>In parallel computing, a task is divided into multiple sub-task which are then allotted to different processors on the same computer system.</td></tr>
<tr><th colspan="2">Number of Computer Systems Involved</th></tr>
<tr><td>Multiple physical computer systems are present in the same computer system.</td><td>A single physical computer system hosts multiple processors.</td></tr>
<tr><th colspan="2">Dependency Between Processes</th></tr>
<tr><td>There might not be much dependency between the processes.</td><td>There is more dependency between the process. Output of one might be the input of another.</td></tr>
<tr><th colspan="2">Scalability</th></tr>
<tr><td>The systems are easily scalable as there is no limitation on how many systems can be added to a network.</td><td>The systems that implement parallel computing have limited scalability.</td></tr>
<tr><th colspan="2">Resource Sharing</th></tr>
<tr><td>Computers have their own memory and processors.</td><td>All the processors share the same memory.</td></tr>
<tr><th colspan="2">Synchronization</th></tr>
<tr><td>The computers in the network have to implement synchronization algorithms.</td><td>All processors use the same master clock for synchronization.</td></tr>
<tr><th colspan="2">Usage</th></tr>
<tr><td>Generally preferred in places requiring high scalability.</td><td>Generally preferred in places requiring faster speed and better performance.</td></tr>
</table>

Parallel & Distributed Computing Comparison

Distributed Computing



Parallel Computing



Distributed Computing Explained

[What is a Distributed System](#)