



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

DSECL ZG 522: Big Data Systems

Session 2: Parallel and Distributed Systems

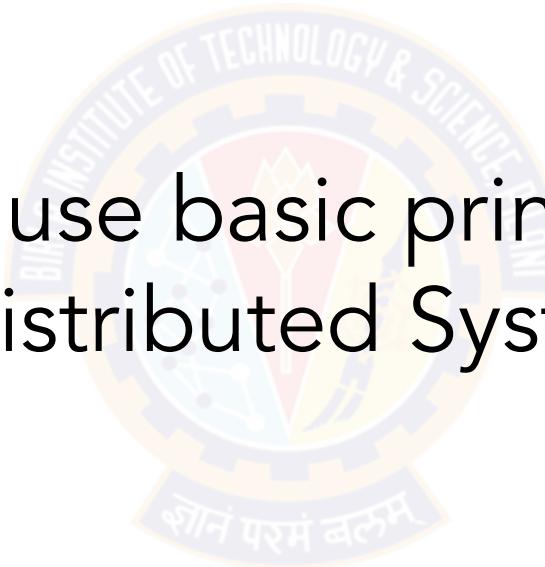
Dr. Anindya Neogi

Associate Professor (Off Campus)

anindya.neogi@pilani.bits-pilani.ac.in

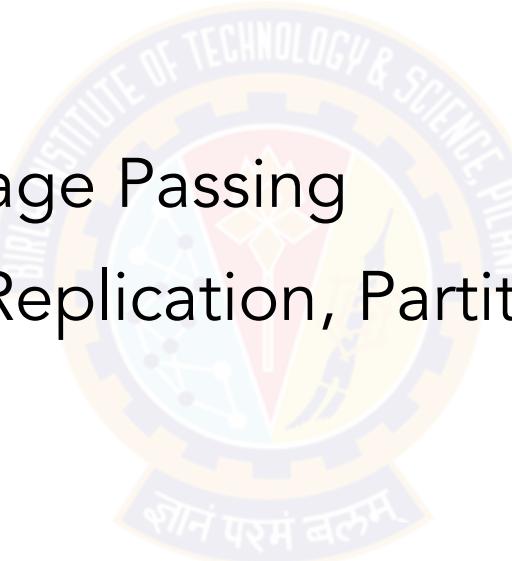
Context

Big Data Systems use basic principles of Parallel
and Distributed Systems



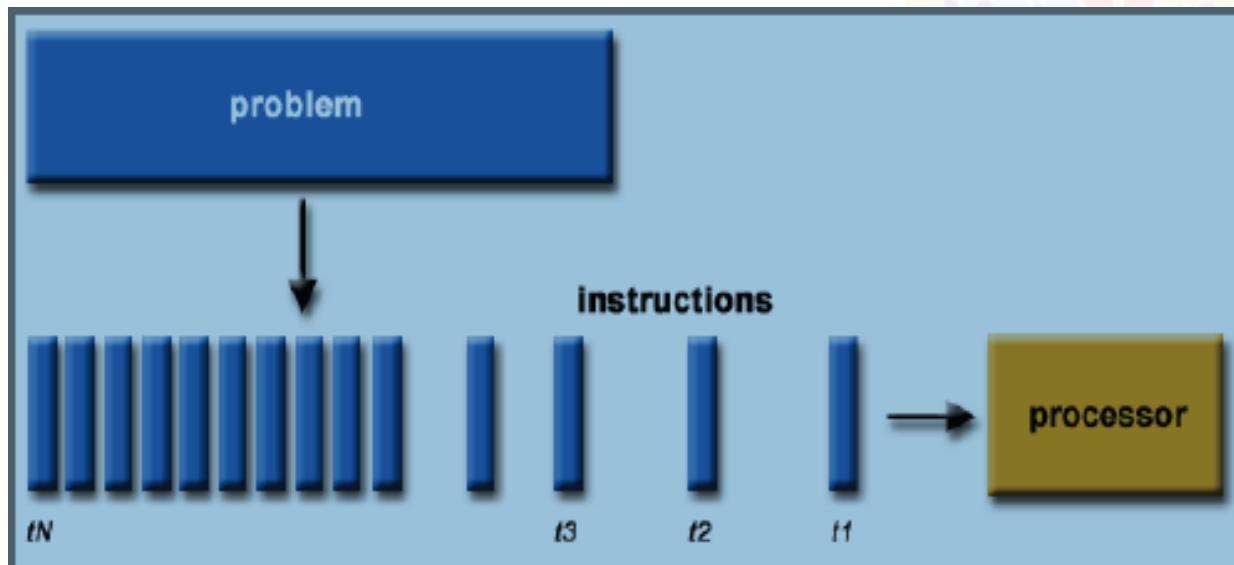
Topics for today

- What are parallel / distributed systems
- Motivation for parallel / distributed systems
- Limits of parallelism
- Shared Memory vs Message Passing
- Data access strategies - Replication, Partitioning, Messaging
- Cluster computing



Serial Computing

- Software written for serial computation:
 - ✓ A problem is broken into a discrete series of instructions
 - ✓ Instructions are executed sequentially one after another
 - ✓ Executed on a single processor
 - ✓ Only one instruction may execute at any moment in time
 - ✓ Single data stores - memory and disk



Extra info:

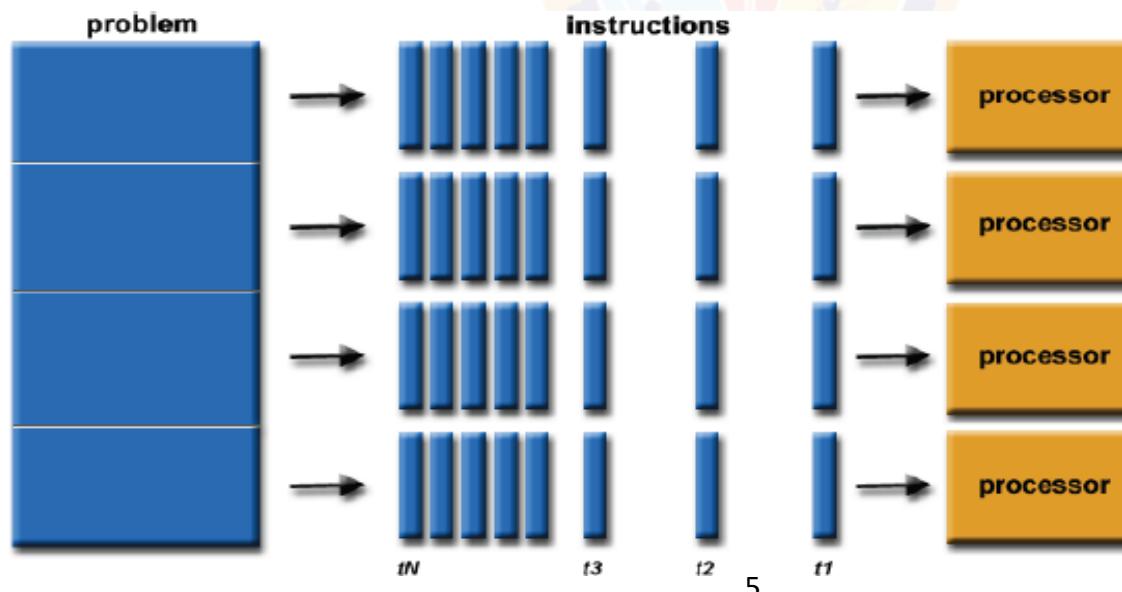
Von Neumann architecture : common memory store and pathways between instructions and data - causes Von Neumann bottleneck

Harvard architecture separates them to reduce bottleneck.

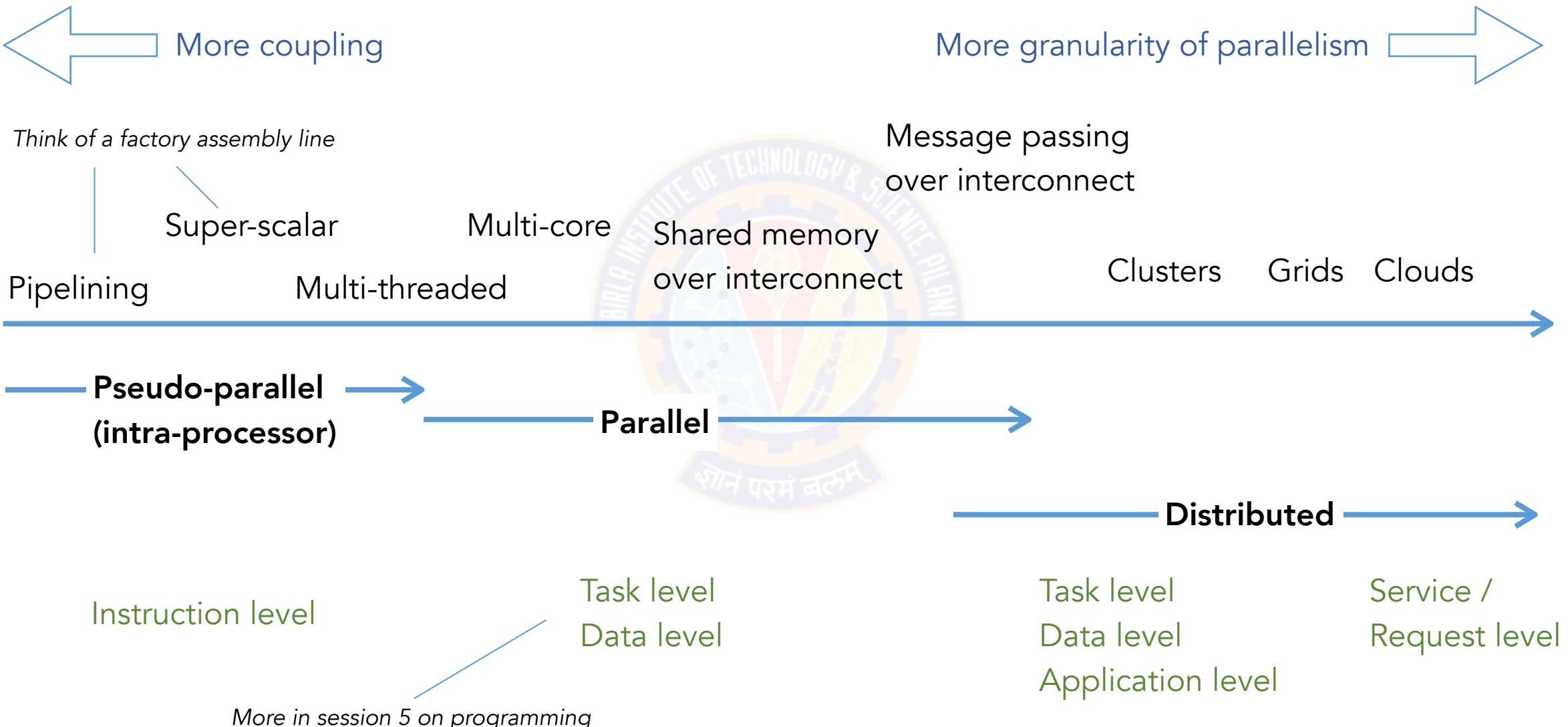
Modern architectures use separate caches for instruction and data.

Parallel Computing

- Simultaneous use of multiple compute resources to solve a computational problem
 - ✓ A problem is broken into discrete parts that can be solved concurrently
 - ✓ Each part is further broken down to a series of instructions
 - ✓ Instructions from each part execute simultaneously on different processors
 - ✓ Different processors can work with independent memory and storage
 - ✓ An overall control/coordination mechanism is employed

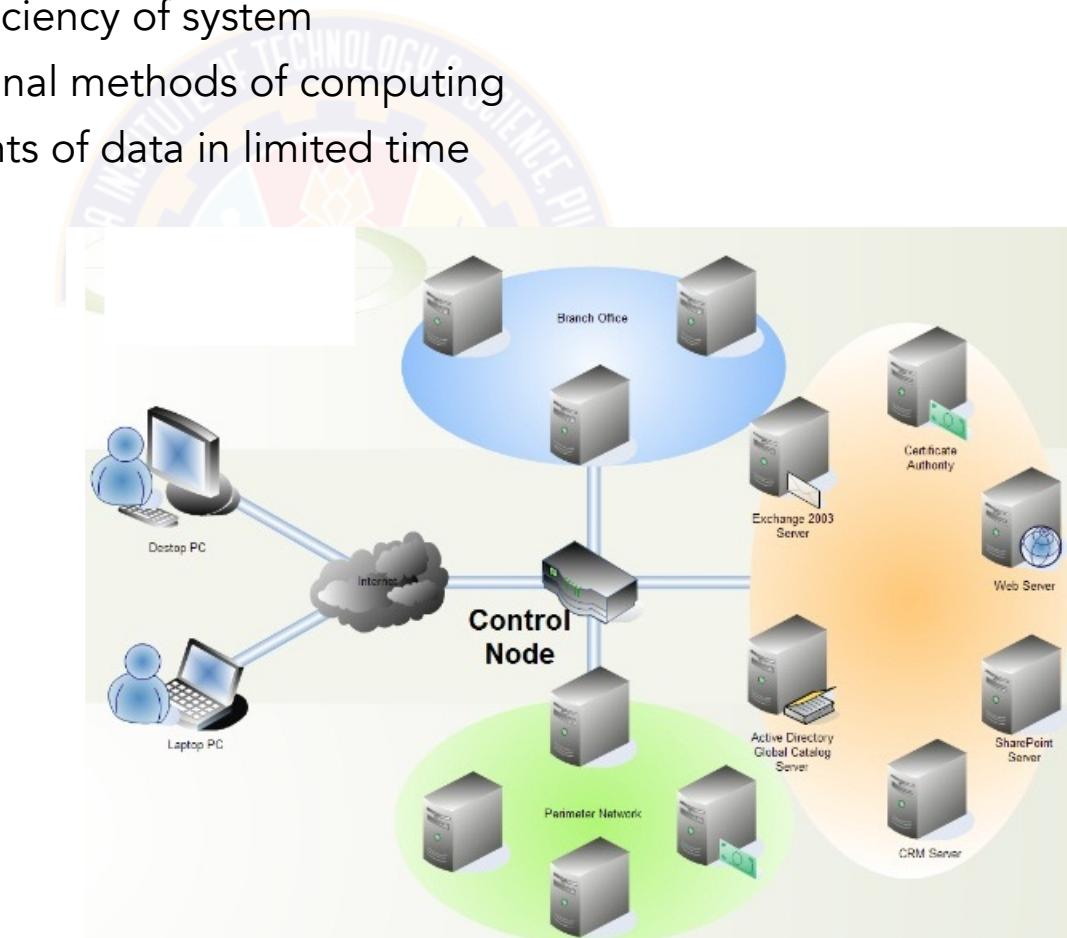


Spectrum of Parallelism



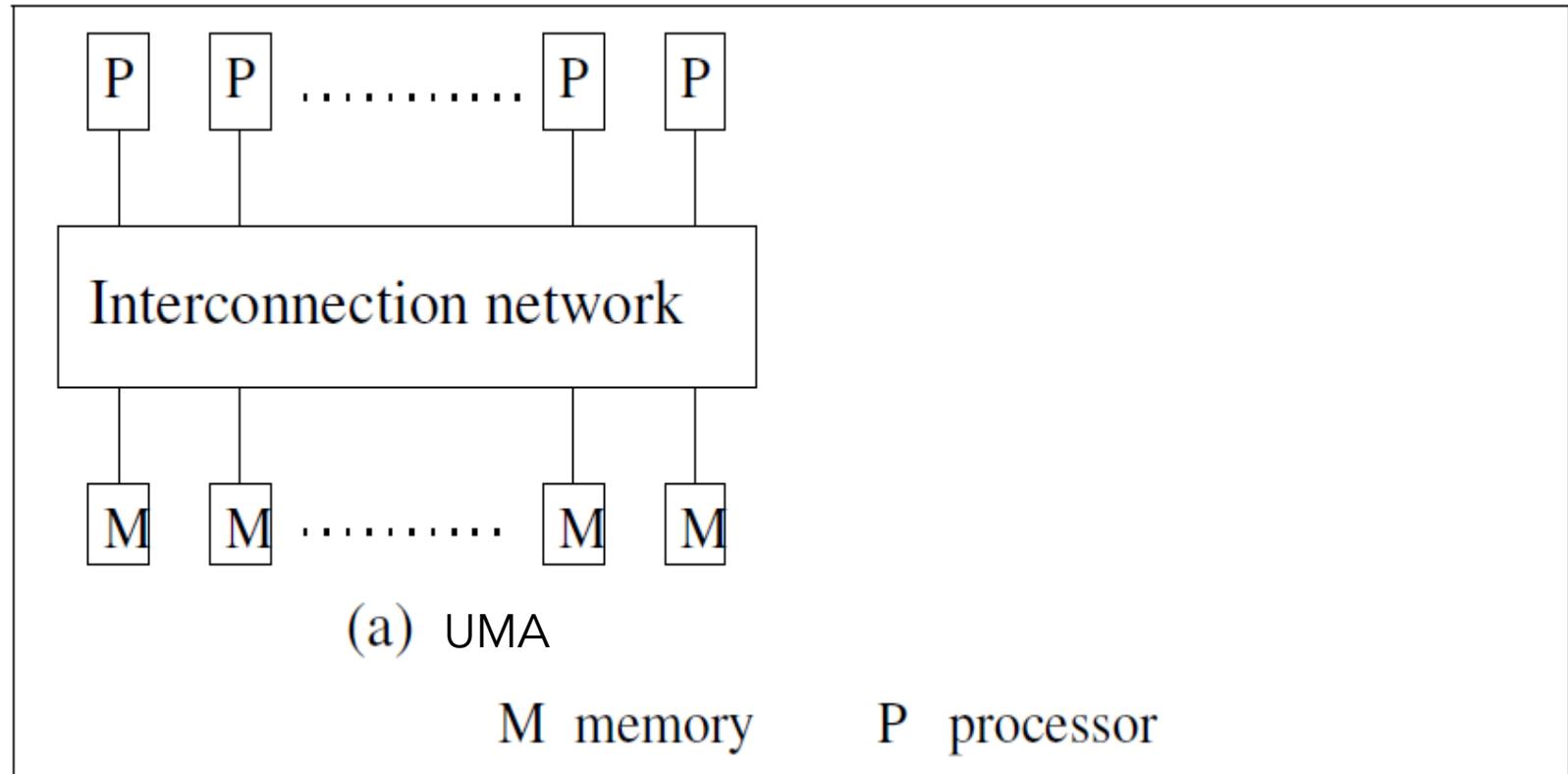
Distributed Computing

- In distributed computing,
 - ✓ Multiple computing resources are connected in network and computing tasks are distributed across these resources
 - ✓ Results into increase in speed and efficiency of system
 - ✓ Faster and more efficient than traditional methods of computing
 - ✓ More suitable to process huge amounts of data in limited time

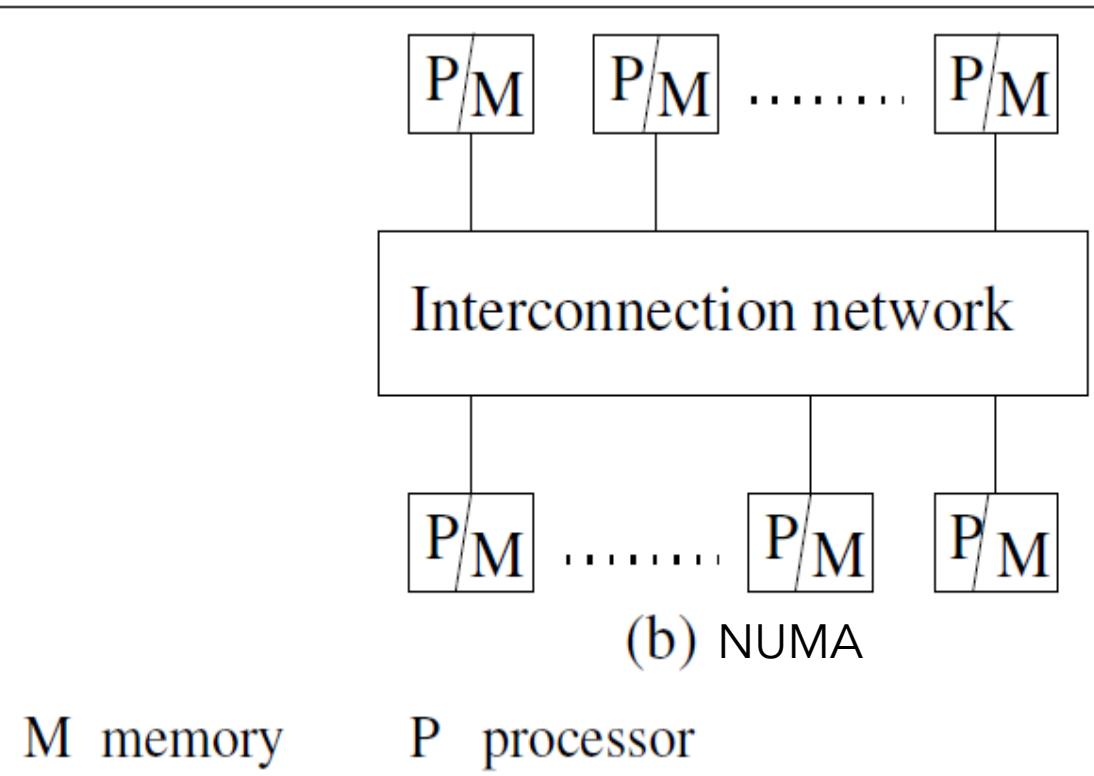


Multi-processor Vs Multi-computer systems

- » Uniform Memory Access Multiprocessor
- » Shared memory address space
- » No common clock
- » Fast interconnect

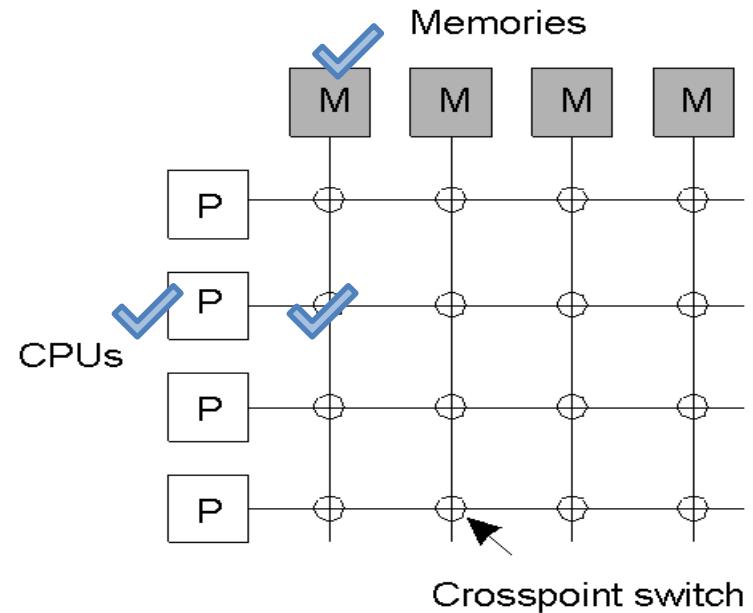


Multi-processor Vs Multi-computer systems

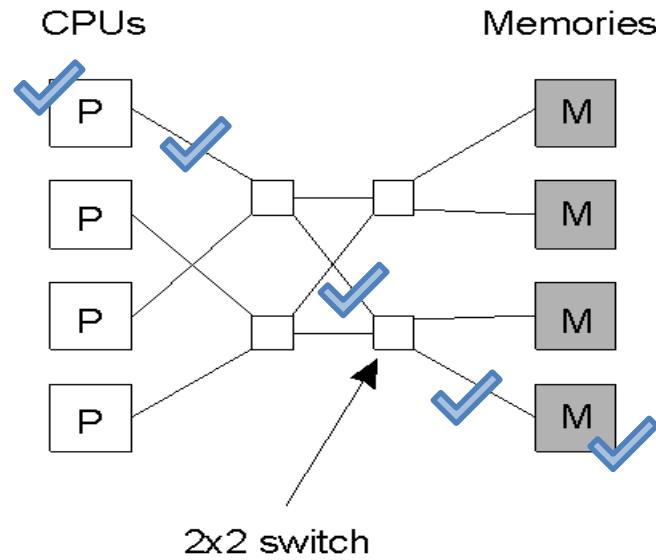


- » Non Uniform Memory Access Multicomputer
- » May have shared address spaces
- » Typically message passing
- » No common clock

Interconnection Networks



(a)



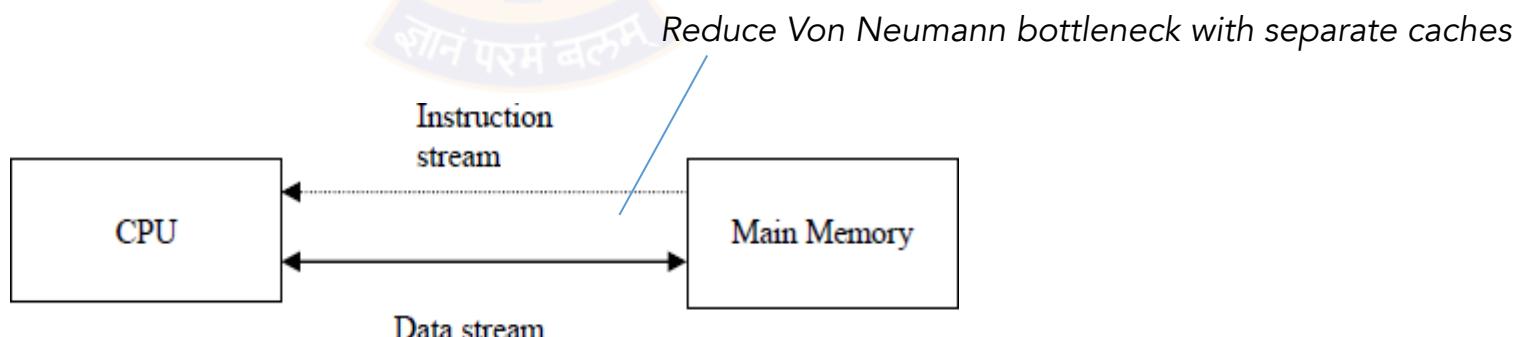
(b)

- a) A crossbar switch - faster
- b) An omega switching network - cheaper

Classify based on Instruction and Data parallelism

Instruction Stream and Data Stream

- The term 'stream' refers to a sequence or flow of either instructions or data operated on by the computer.
- In the complete cycle of instruction execution, a flow of instructions from main memory to the CPU is established. This flow of instructions is called **instruction stream**.
- Similarly, there is a flow of operands between processor and memory bi-directionally. This flow of operands is called **data stream**.



Flynn's Taxonomy

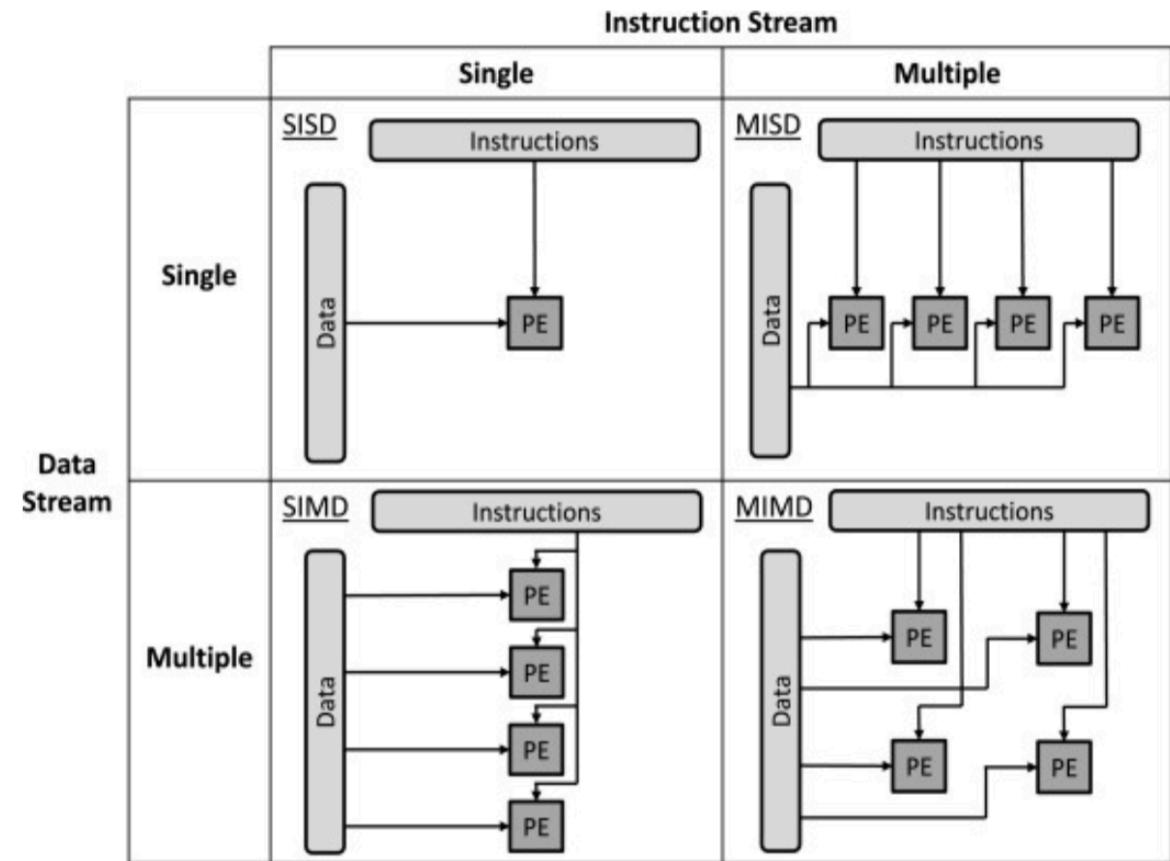
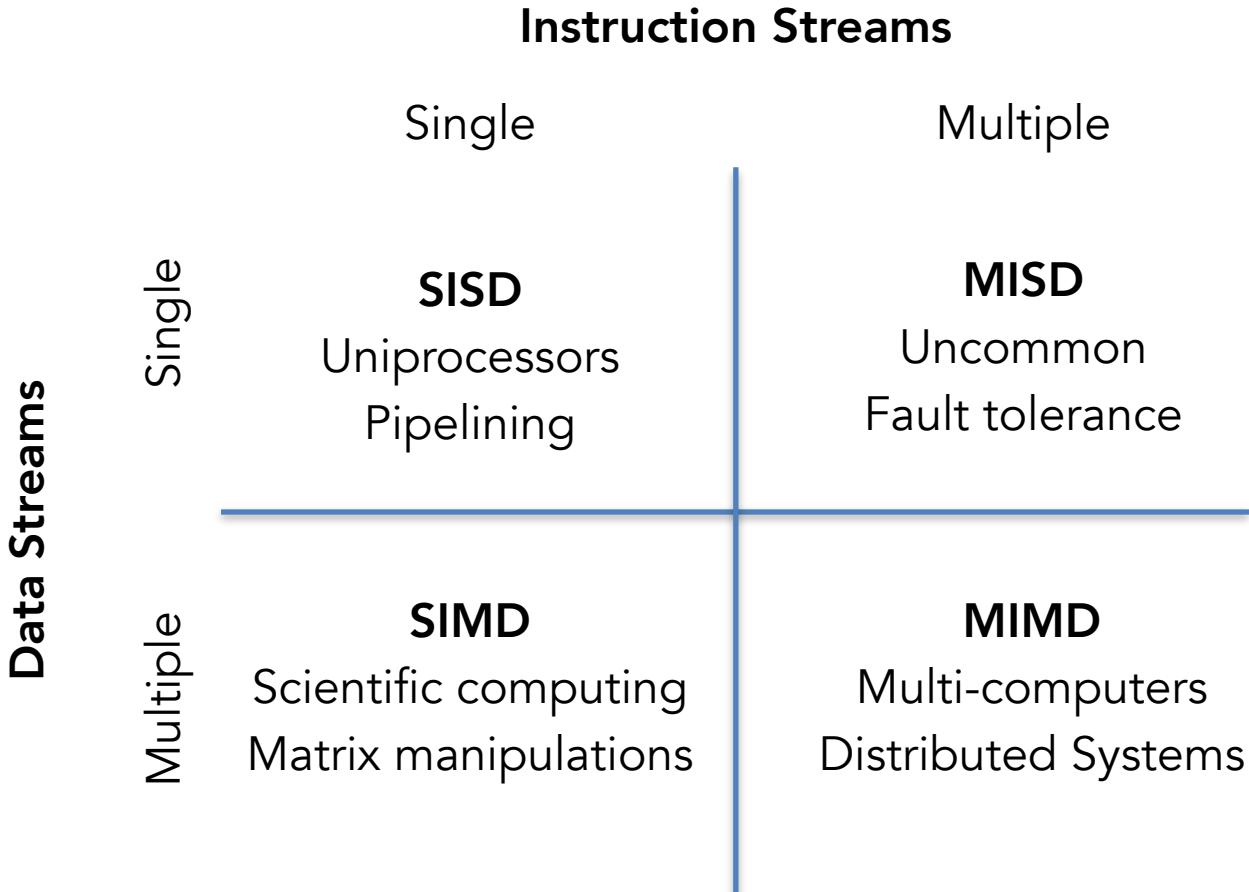


Image from sciedirect.com

Some basic concepts

(esp. for programming in Big Data Systems)

» Coupling

- » Tight - SIMD, MISD shared memory systems
- » Loose - NOW, distributed systems, no shared memory

» Speedup

- » how much faster can a program run when given N processors as opposed to 1 processor — $T(1) / T(N)$
- » We will study Amdahl's Law, Gustafson's Law

» Parallelism of a program

- » Compare time spent in computations to time spent for communication via shared memory or message passing

» Granularity

- » Average number of compute instructions before communication is needed across processors

» Note:

- » **If coarse granularity, use distributed systems else use tightly coupled multi-processors/computers**
- » **Potentially high parallelism doesn't lead to high speedup if granularity is too small leading to high overheads**

Comparing Parallel and Distributed Systems

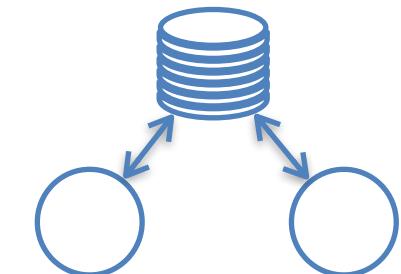
Distributed System	Parallel System
Independent, autonomous systems connected in a network accomplishing specific tasks	Computer system with several processing units attached to it
Coordination is possible between connected computers with own memory and CPU	A common shared memory can be directly accessed by every processing unit in a network
Loose coupling of computers connected in network, providing access to data and remotely located resources	Tight coupling of processing resources that are used for solving single, complex problem
Programs have coarse grain parallelism	Programs may demand fine grain parallelism

Motivation for parallel / distributed systems (1)

- Inherently distributed applications
 - e.g. financial tx involving 2 or more parties
- Better scale in creating multiple smaller parallel tasks instead of a complex task
 - e.g. evaluate an aggregate over 6 months data
- Processors getting cheaper and networks faster
 - e.g. Processor speed 2x / 1.5 years, network traffic 2x/year, processors limited by energy consumption
- Better scale using replication or partitioning of storage
 - e.g. replicated media servers for faster access or shards in search engines
- Access to shared remote resources
 - e.g. remote central DB
- Increased performance/cost ratio compared to special parallel systems
 - e.g. search engine runs on a Network-of-Workstations



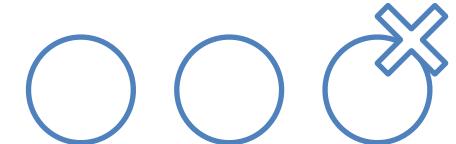
replicated / partitioned storage



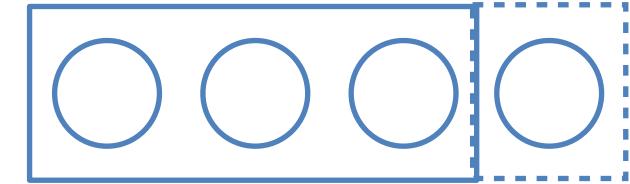
remote shared resource

Motivation for parallel / distributed systems (2)

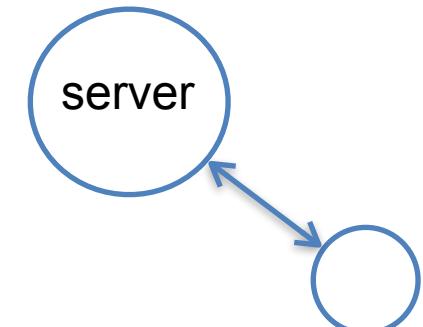
- Better reliability because less chance of multiple failures
 - Be careful about Integrity : Consistent state of a resource across concurrent access
- Incremental scalability
 - Add more nodes in a cluster to scale up
 - e.g. Clusters in Cloud services, autoscaling in AWS
- Offload computing closer to user for scalability and better resource usage
 - Edge computing



cluster nodes



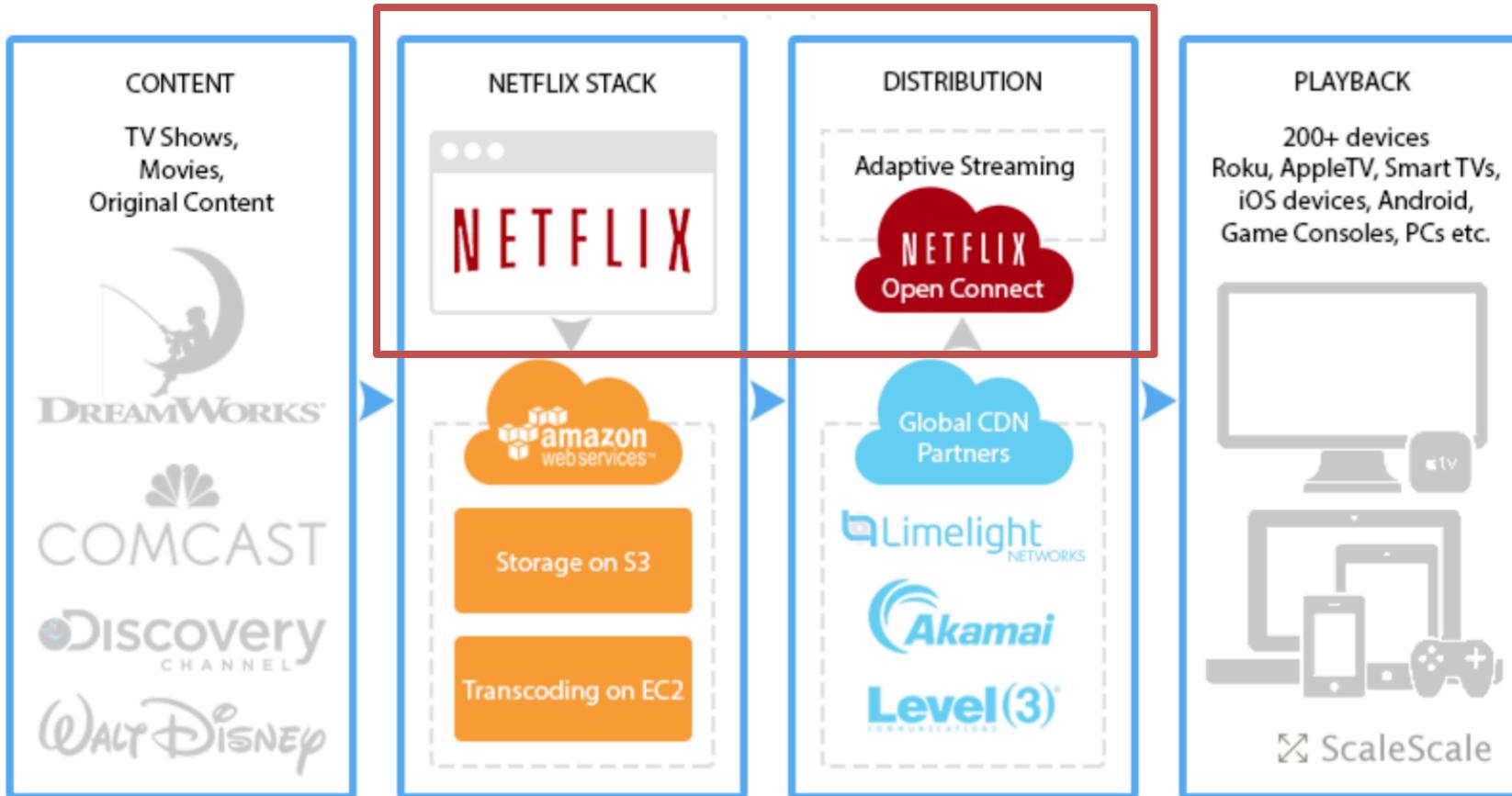
resize cluster



offload some error handling to edge

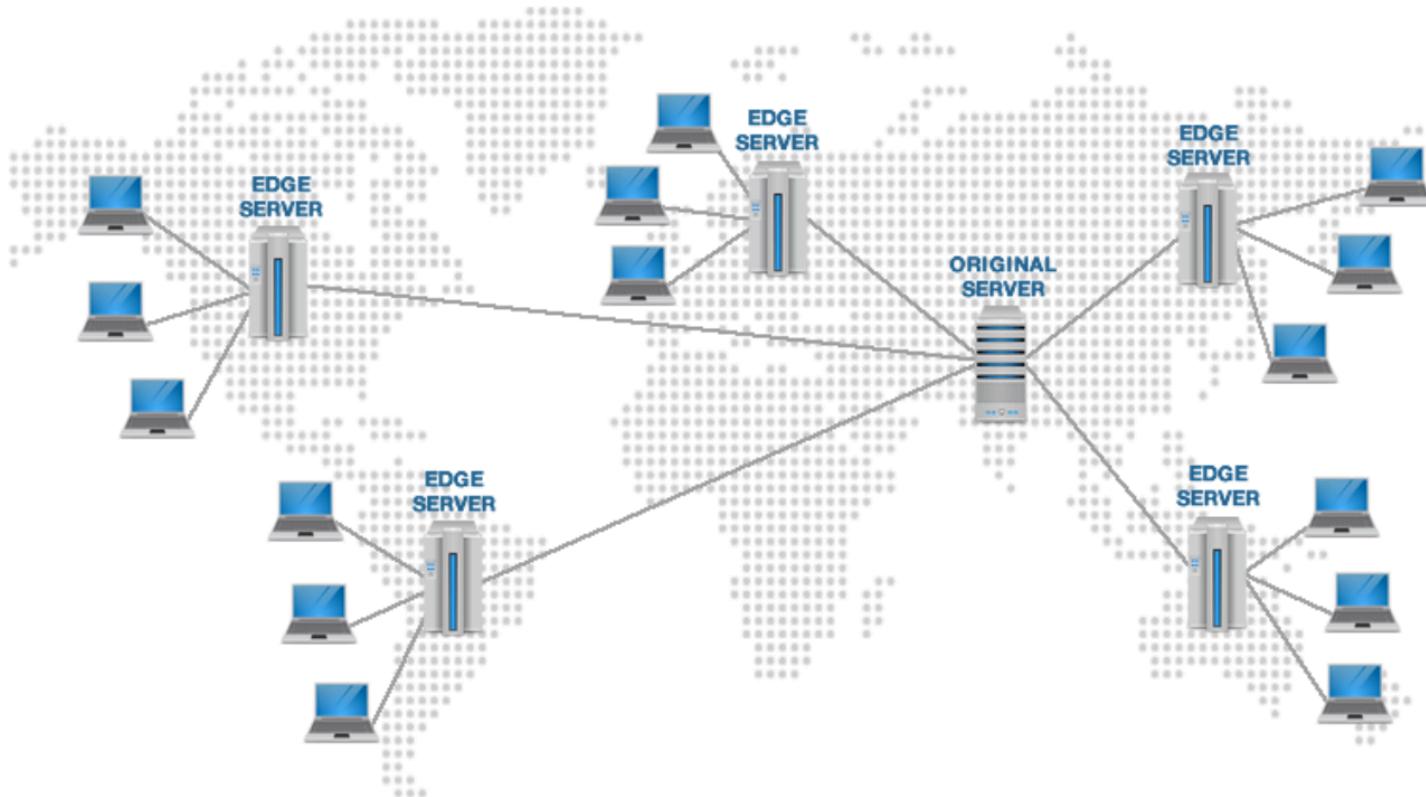
Example: Netflix

~700+ distributed micro-services and hardware, integrated with other vendors



reference: <https://medium.com/refraction-tech-everything/how-netflix-works-the-hugely-simplified-complex-stuff-that-happens-every-time-you-hit-play-3a40c9be254b>

Distributed network of content caching servers



This would be a P2P network if you were using bit torrent for free

Examples

In each of the following cases, what is the motivation for parallel / distributed computing ?

1. Airline scheduling
2. Summary Statistics of Historic Sales Data of a Retailer
3. Web Crawler
4. Weather forecasting

Techniques for High Data Processing

Method	Description	Usage
Cluster computing	A collection of computers, homogenous or heterogenous, using commodity components running open source or proprietary software, communicating via message passing	Commonly used in Big Data Systems , such as Hadoop
Massively Parallel Processing (MPP)	Typically proprietary Distributed Shared Memory machines with integrated storage	May be used in traditional Data Warehouses , Data processing appliances, e.g. EMC Greenplum (postgreSQL on an MPP)
High-Performance Computing (HPC)	Known to offer high performance and scalability by using in-memory computing	Used to develop specialty and custom scientific applications for research where results is more valuable than cost

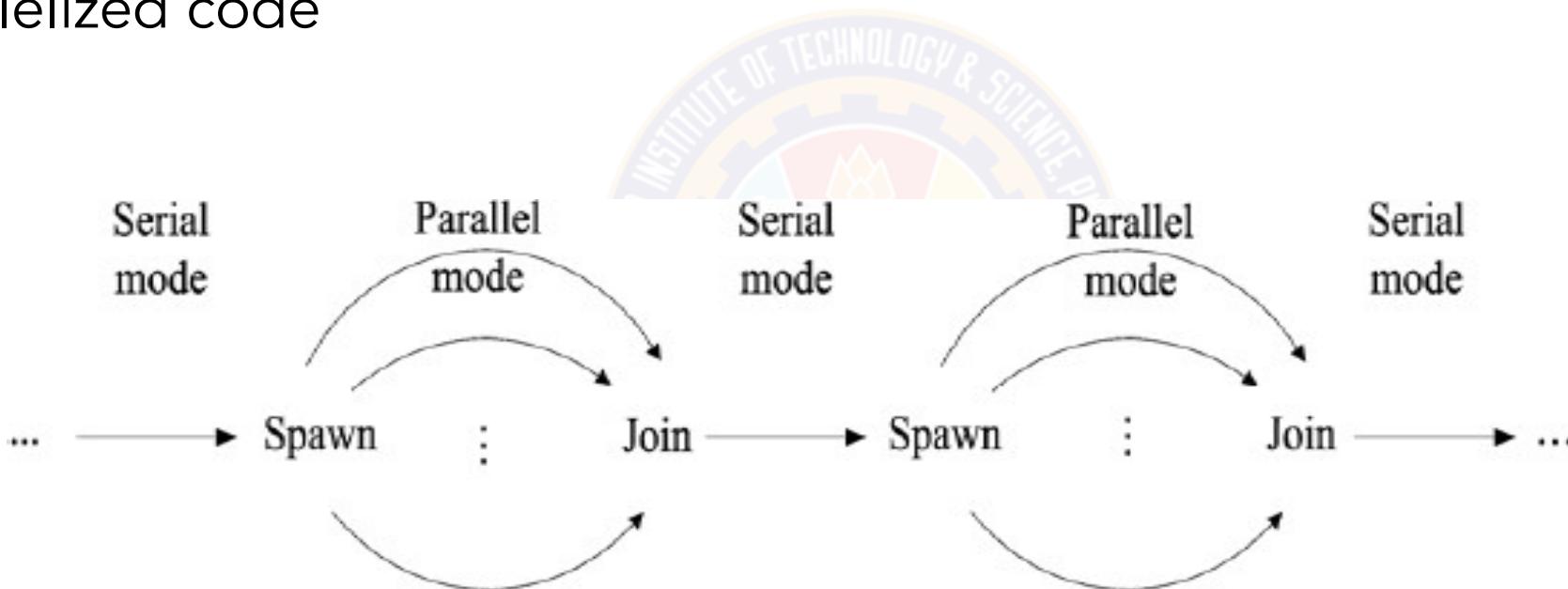
Topics for today

- What are parallel / distributed systems
- Motivation for parallel / distributed systems
- Limits of parallelism
- Shared Memory vs Message Passing
- Data access strategies - Replication, Partitioning, Messaging
- Cluster computing



Limits of Parallelism

- A parallel program has some sequential / serial code and significant parallelized code



Amdahl's Law

- $T(1)$: Time taken for a job with 1 processor
- $T(N)$: Time taken for same job with N processors
- Speedup $S(N) = T(1) / T(N)$
- $S(N)$ is ideally N when it is a perfectly parallelizable program, i.e. data parallel with no sequential component
- Assume fraction of a program that cannot be parallelised (serial) is f and $1-f$ is parallel

$$\checkmark T(N) \geq f * T(1) + (1-f) * T(1) / N$$

Only parallel portion is faster by N

- $S(N) = T(1) / (f * T(1) + (1-f) * T(1) / N)$
- $S(N) = 1 / (f + (1-f) / N)$
- Implication :
 - ✓ If $N \Rightarrow \infty$, $S(N) \Rightarrow 1/f$
 - ✓ The effective speedup is limited by the sequential fraction of the code

Amdahl's Law - Example

10% of a program is sequential and there are 100 processors.

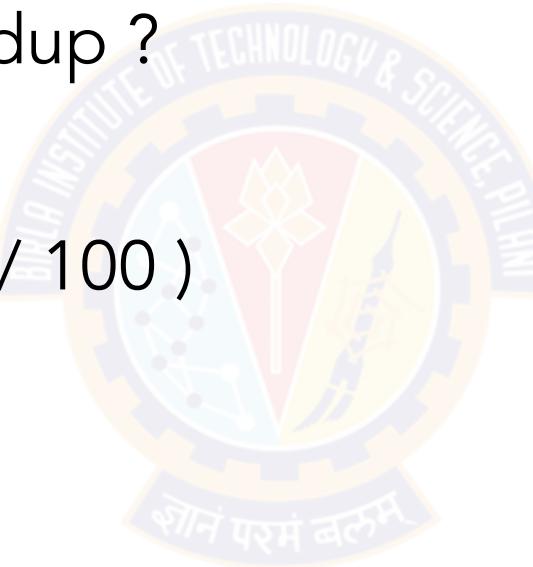
What is the effective speedup ?

$$S(N) = 1 / (f + (1-f) / N)$$

$$S(100) = 1 / (0.1 + (1-0.1) / 100)$$

$$= 1 / 0.109$$

$$= 9.17 \text{ (approx)}$$



Limitations in speedup

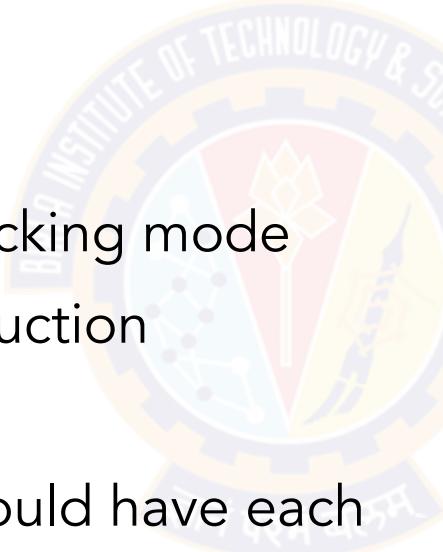
Besides the sequential component of the program, communication delays also result in reduction of speedup

A and B exchange messages in blocking mode

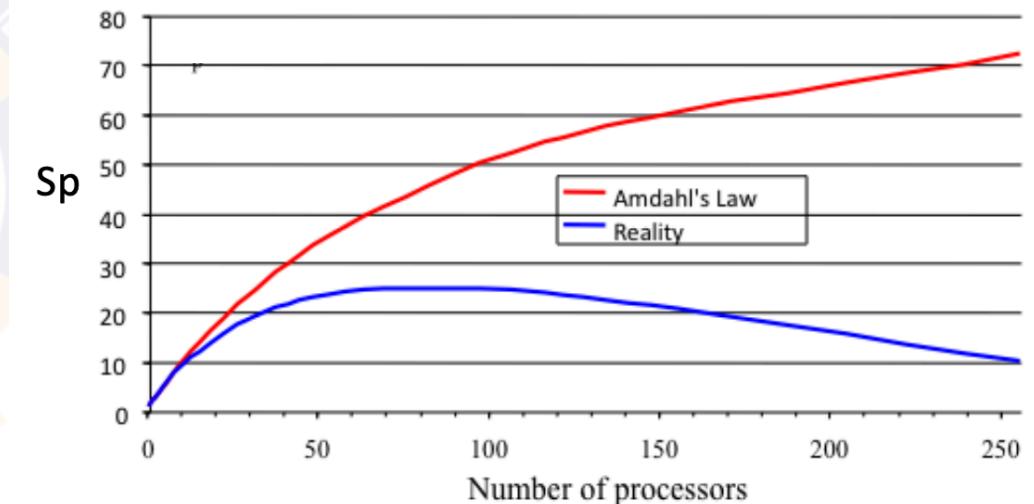
Say processor speed is 0.5ns / instruction

Say network delay one way is 10 us

For one message delay, A and B would have each executed $10\text{us}/0.5\text{ns} = 20000$ instructions



+ context switching, scheduling, load balancing, I/O ...



Super-linear speedup

Memory hierarchy / caches may increase the speedup

Single processor will have cache size D

But N processors will have total cache size $N * D$

So in a parallel environment the effect of cache may go up to N fold

Partitioning a data parallel program to run across multiple processors may lead to a better cache hit.*

Possible in some workloads with not much of other overheads, esp in embarrassingly parallel applications.

* There could be also be cache coherency overheads.

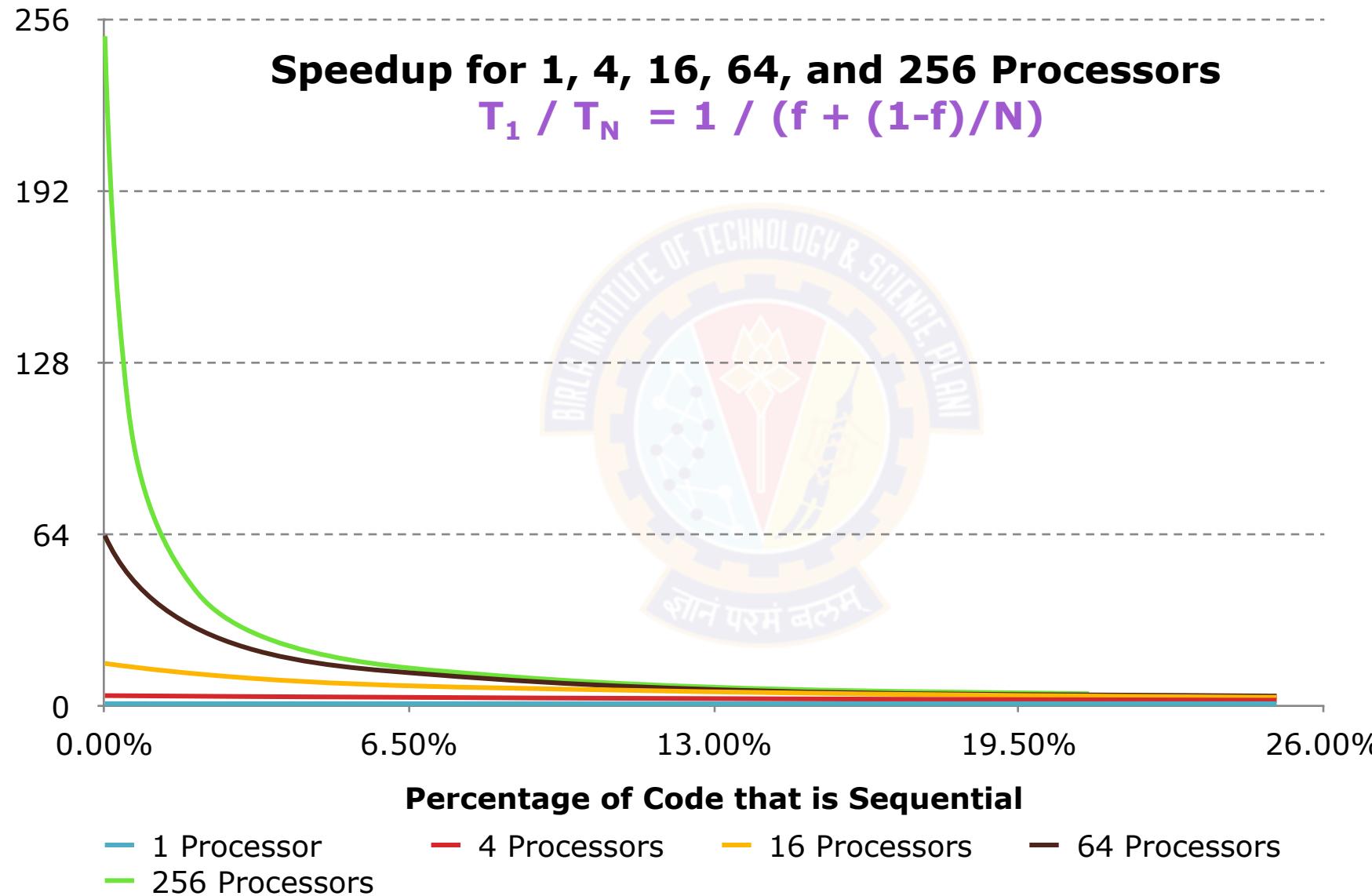
Why Amdahl's Law is such bad news

$$S(N) \sim 1/f, \text{ for large } N$$

Suppose 33% of a program is sequential

- Then even a billion processors won't give a speedup over 3
- For the 256 cores to gain $\geq 100x$ speedup, we need
$$100 \leq 1 / (f + (1-f)/256)$$
Which means $f \leq .0061$ or 99.4% of the algorithm must be perfectly parallelizable !!

Speedup plot



But wait - may be we are missing something

- » The key assumption in Amdahl's Law is **total workload is fixed** as #processors is increased
- » This doesn't happen in practice — sequential part doesn't increase with resources
- » Additional processors can be used for more complex workload and new age larger parallel problems
- » So Amdahl's law under-estimates Speedup
- » What if we assume **fixed workload per processor**

Gustafson-Barsis Law

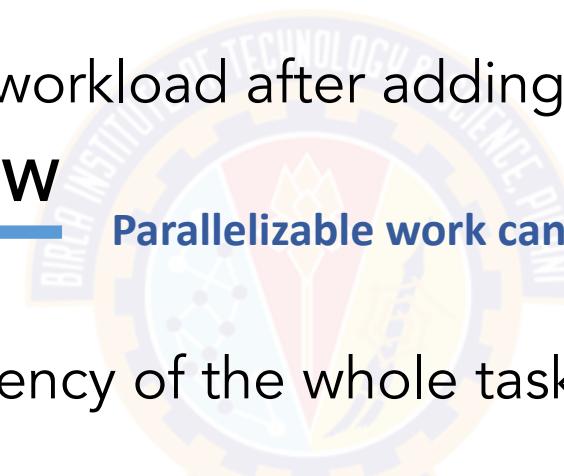
Let W be the execution workload of the program before adding resources

f is the sequential part of the workload

$$\text{So } W = f * W + (1-f) * W$$

Let $W(N)$ be larger execution workload after adding N processors

$$\text{So } W(N) = f * W + N * (1-f) * W$$

Parallelizable work can increase N times

The theoretical speedup in latency of the whole task at a fixed interval time T

$$S(N) = T * W(N) / T * W$$

$$= W(N) / W = (f * W + N * (1-f) * W) / W$$

$$S(N) = f + (1-f) * N$$

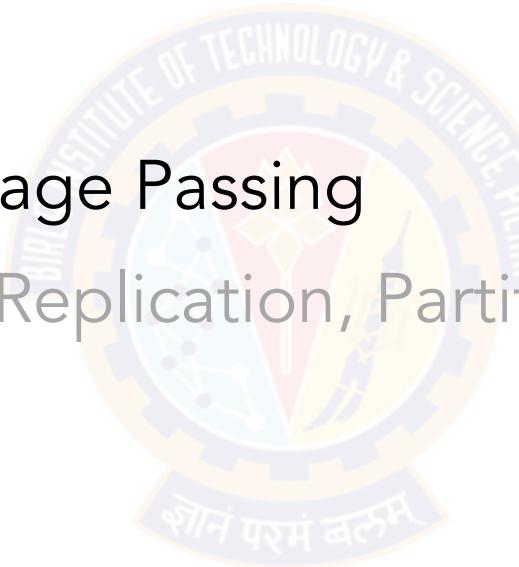
$S(N)$ is not limited by f as N scales

Remember this when we discuss programming in Session 5

So solve larger problems when you have more processors

Topics for today

- What are parallel / distributed systems
- Motivation for parallel / distributed systems
- Limits of parallelism
- Shared Memory vs Message Passing
- Data access strategies - Replication, Partitioning, Messaging
- Cluster computing



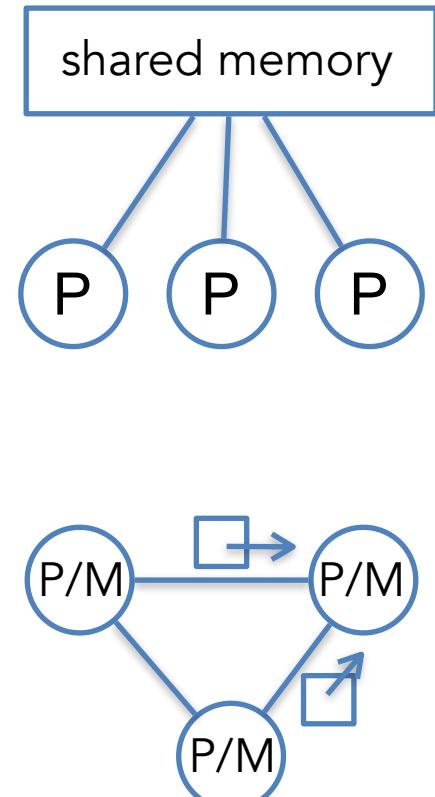
Memory access models

» Shared memory

- » Multiple tasks on different processors access a common address space in UMA or NUMA architectures
- » Conceptually easier for programmers
- » Think of writing a voting algorithm - it is trivial because everyone is in the same room, i.e. writing same variable

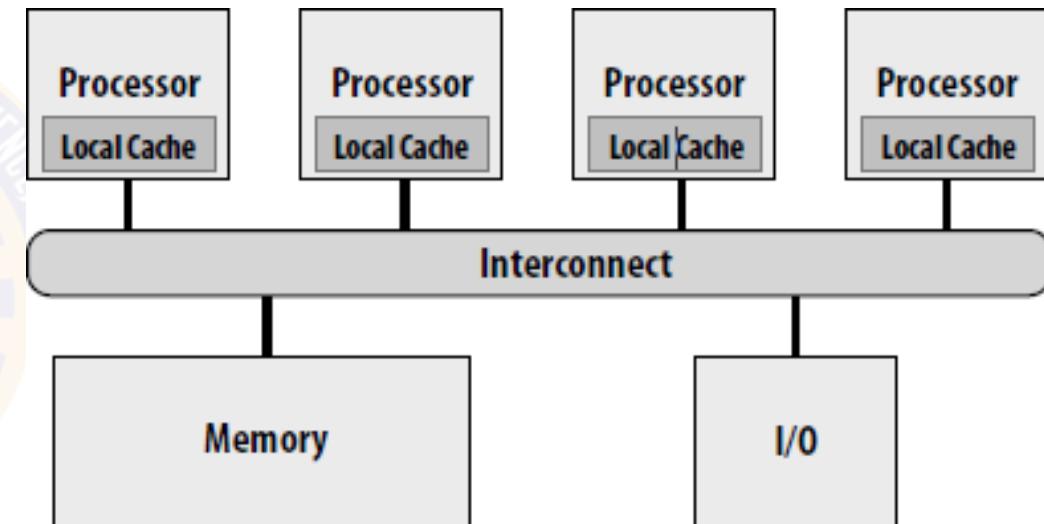
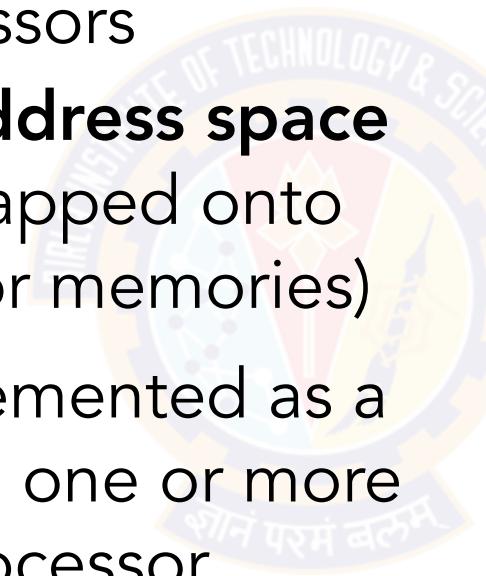
» Distributed memory

- » Multiple tasks – executing a single program – access data from separate (and isolated) address spaces (i.e. separate virtual memories)
- » How will this remote access happen ?



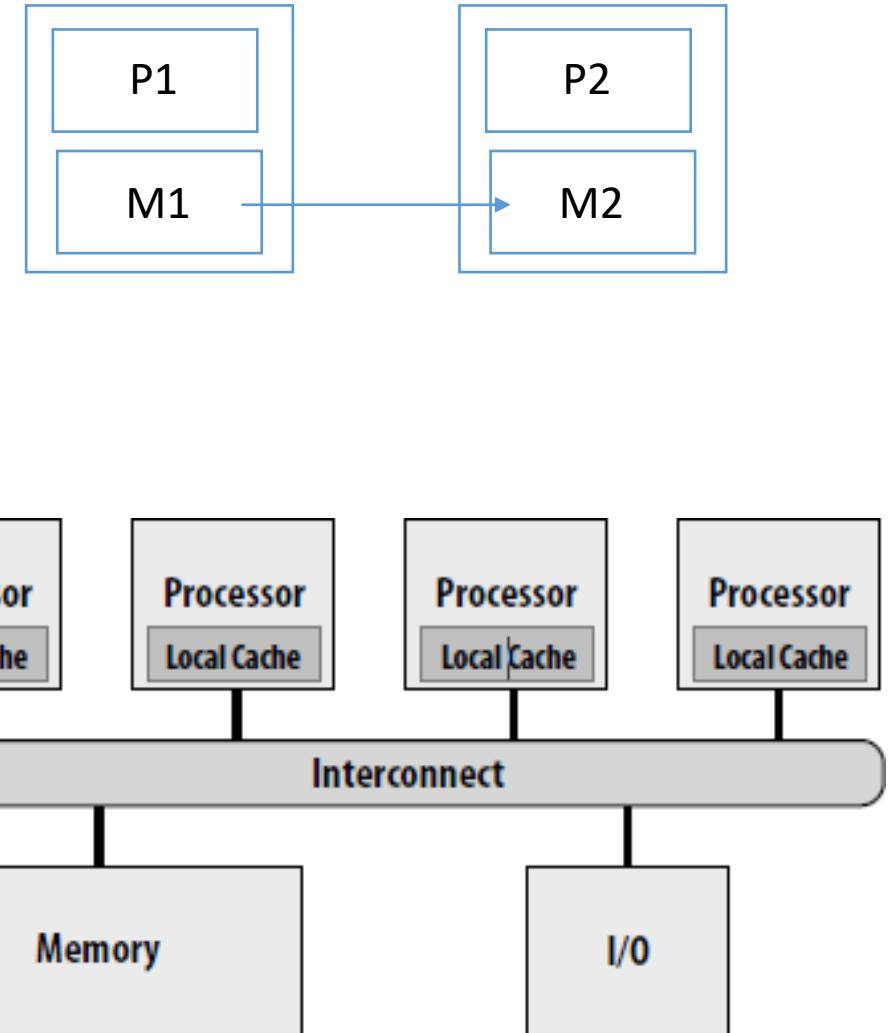
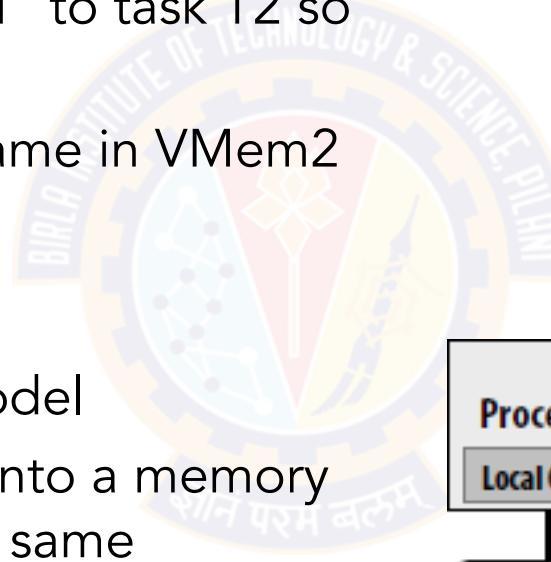
Shared Memory Model: Implications for Architecture

- A shared memory system has
 - ✓ Physical memory (or memories) are accessible by all processors
 - ✓ The **single (logical) address space** of each processor is mapped onto the physical memory (or memories)
- A **single program** is implemented as a collection of **threads**, with one or more threads scheduled in a processor
- Conceptually easier to program



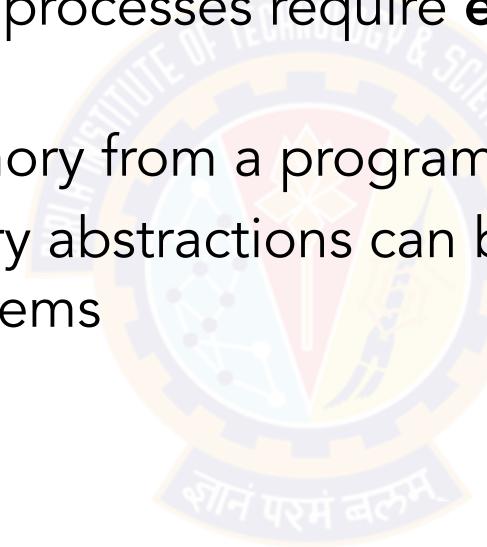
Distributed memory and message passing

- In a Distributed Memory model, data has to be moved across Virtual Memories:
 - ✓ i.e. a data item in VMem1 produced by task T1 has to be “communicated” to task T2 so that
 - ✓ T2 can make a copy of the same in VMem2 and use it.
- Whereas in a Shared Memory model
 - ✓ task T1 writes the data item into a memory location and T2 can read the same
 - ✓ how ? the memory location is part of the logical address space that is shared between the tasks



Computing model for message passing

- **Each data item must be located in one of the address spaces**
 - ✓ Data must be **partitioned explicitly** and placed (i.e. distributed)
 - ✓ All interactions between processes require **explicit communication** i.e. passing of messages
 - ✓ Harder than shared memory from a programming abstraction standpoint
 - ✓ Note: Shared memory abstractions can be theoretically created on message passing systems
- In the simplest form:
 - ✓ a sender (who has the data) and
 - ✓ a receiver (who has to access the data)
 - must co-operate for exchange of data



Communication model for message passing

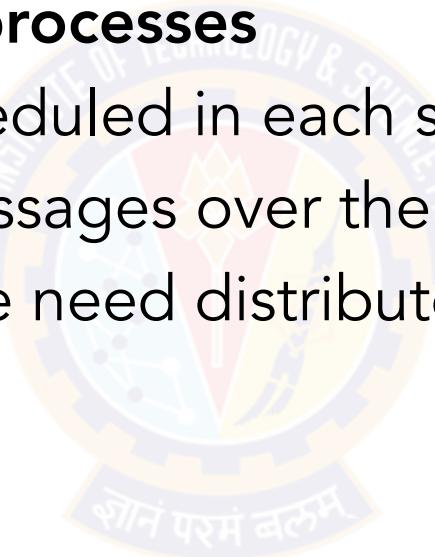
- Processes operate within their own private address spaces
- Processes communicate by sending/receiving messages
 - ✓ **send**: specifies recipient, buffer to be transmitted, and message identifier ("tag")
 - ✓ **receive**: specifies buffer to store data, and optional message identifier
 - ✓ Sending messages is the only way to exchange data between processes 1 and 2



Distributed Memory Model: Implications for Architecture

A distributed memory model is implemented in a distributed system where

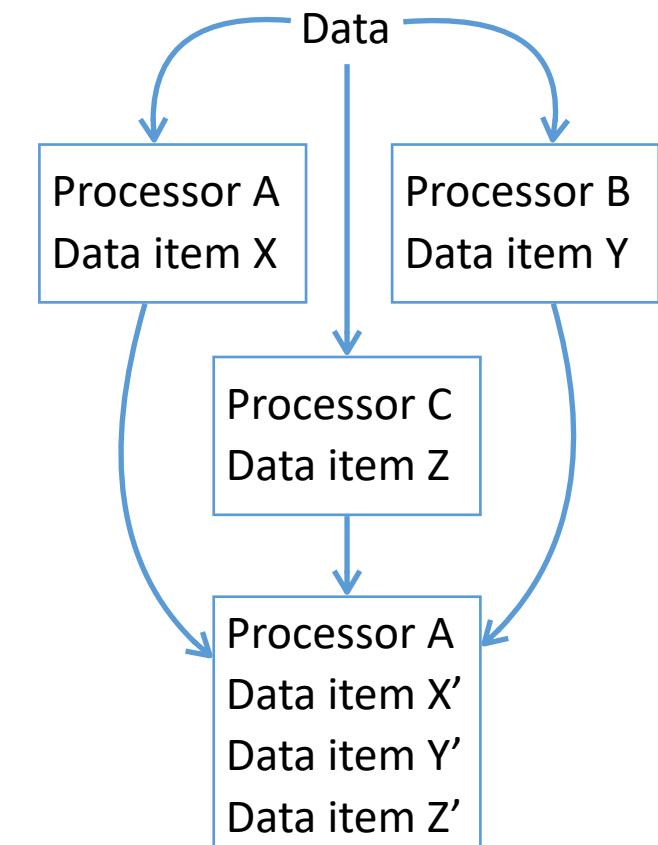
- ✓ A collection of stand-alone systems are connected in a network
- ✓ **A task runs as a collection of processes**
- ✓ One or more processes are scheduled in each system / node
- ✓ Data exchanges happen via messages over the network
- ✓ Harder for programmers - hence need distributed OS / middleware layer to hide some complexity *



* One can create a shared memory view using message passing and vice versa

Message Passing Model – Separate Address Spaces

- Use of separate address spaces complicates programming
- But this **complication is usually restricted to one or two phases:**
 - ✓ **Partitioning** the input data
 - ✓ Improves locality - computation closer to data
 - ✓ Each process is enabled to access data from within its address space, which in turn is likely to be mapped to the memory hierarchy of the processor in which the process is running
 - ✓ **Merging** / Collecting the output data
 - ✓ This is required if each task is producing outputs that have to be combined

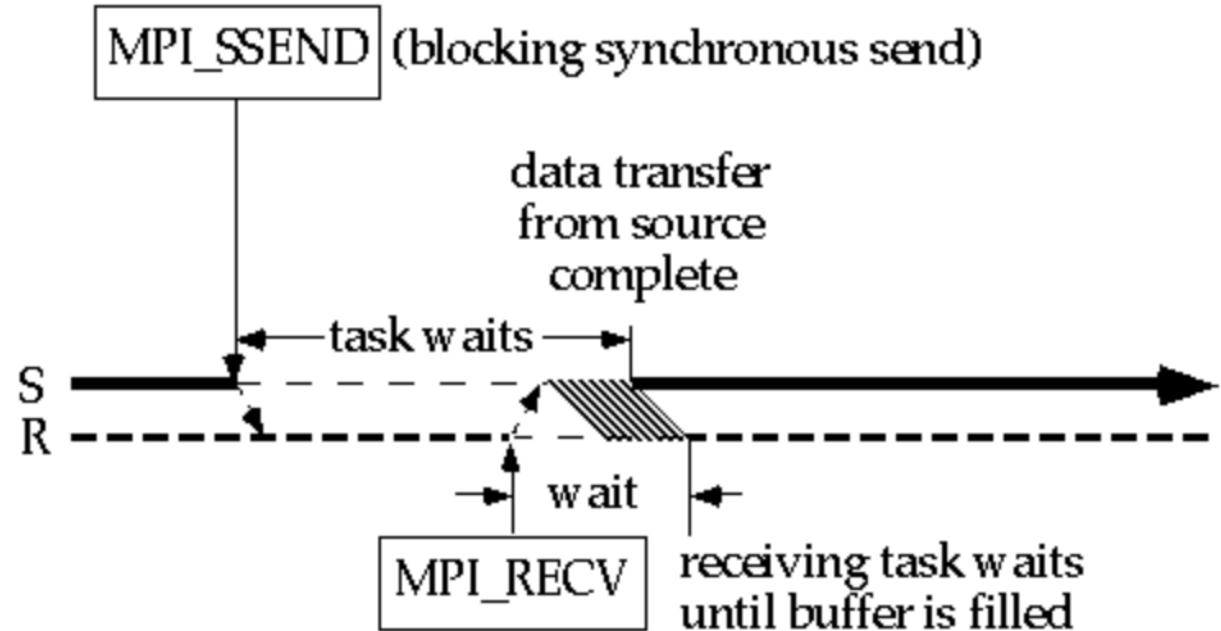


We will see example of Hadoop map-reduce where data is partitioned, outputs communicated over messages and merged to get final answer.

Remember **granularity** ?

Message Passing Primitives

- » Operations: Send and Receive
- » Options:
 - » Blocking vs Non-Blocking
 - » Sync vs Async



- » **Important :** A message can be received in the OS buffer but may not have been delivered to application buffer. This is where a distributed message ordering logic can come in.

Image ref: <https://cvw.cac.cornell.edu/mpip2p/SyncSend>

Send-Receive Options

1	Send	Sync	Blocking	Returns only after data is sent from kernel buffer. Easiest to program but longest wait.
2	Send	Async	Blocking	Returns after data is copied to kernel buffer but not sent. Handle returned to check send status.
3	Send	Sync	Non-blocking	Same as (2) but with no handle.
4	Send	Async	Non-blocking	Returns immediately with handle. Complex to program but minimum wait.
5	Receive	Sync	Blocking	Returns after application gets data.
6	Receive	Sync	Non-blocking	Returns immediately with data or handle to check status. More efficient.

Topics for today

- What are parallel / distributed systems
- Motivation for parallel / distributed systems
- Limits of parallelism
- Shared Memory vs Message Passing
- Data access strategies - Replication, Partitioning, Messaging
- Cluster computing



Data Access Strategies: Partition

- Strategy:
 - ✓ Partition data – typically, equally – to the nodes of the (distributed) system
- Cost:
 - ✓ Network access and merge cost when query needs to go across partitions
- Advantage(s):
 - ✓ **Works well if task/algorithim is (mostly) data parallel**
 - ✓ Works well when there is Locality of Reference within a partition
- Concerns
 - ✓ Merge across data fetched from multiple partitions
 - ✓ Partition balancing
 - ✓ Row vs Columnar layouts - what improves locality of reference ?
 - ✓ Will study shards and partition in Hadoop and MongoDB

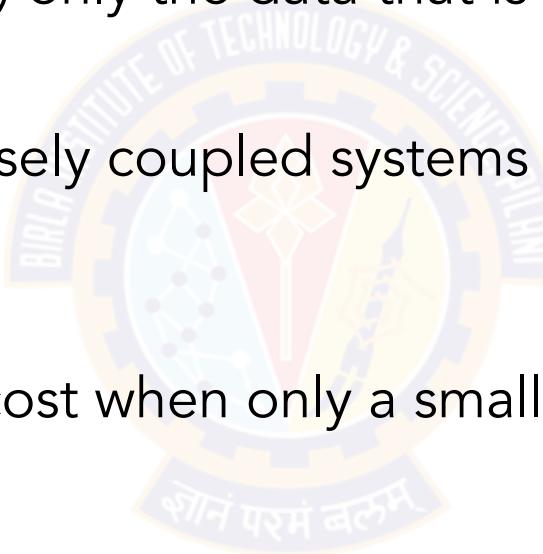
Data Access Strategies: Replication

- Strategy:
 - ✓ Replicate all data across nodes of the (distributed) system
- Cost:
 - ✓ Higher storage cost
- Advantage(s):
 - ✓ All data accessed from local disk: no (runtime) communication on the network
 - ✓ High performance with parallel access
 - ✓ Fail over across replicas
- Concerns
 - ✓ Keep replicas in sync — various consistency models between readers and writers
 - ✓ Will study in depth for MongoDB



Data Access Strategies: (Dynamic) Communication

- Strategy:
 - ✓ Communicate (at runtime) only the data that is required
- Cost:
 - ✓ High network cost for loosely coupled systems and data set to be exchanged is large
- Advantage(s):
 - ✓ Minimal communication cost when only a small portion of the data is actually required by each node
- Concerns
 - ✓ Highly available and performant network
 - ✓ Fairly independent parallel data processing



Data Access Strategies – Networked Storage

- Common Storage on the Network:
 - ✓ Storage Area Network (for raw access – i.e. disk block access)
 - ✓ Network Attached Storage (for file access)
- Common Storage on the Cloud:
 - ✓ Use Storage as a Service
 - ✓ e.g. Amazon S3

More in-depth coverage when studying Amazon storage case study

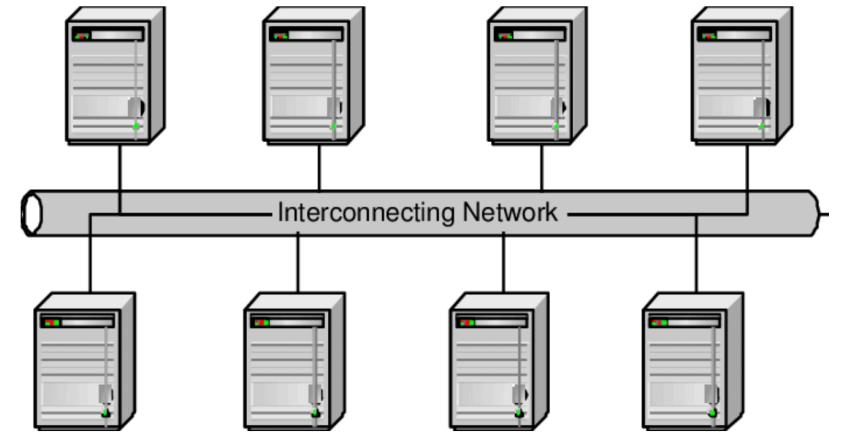
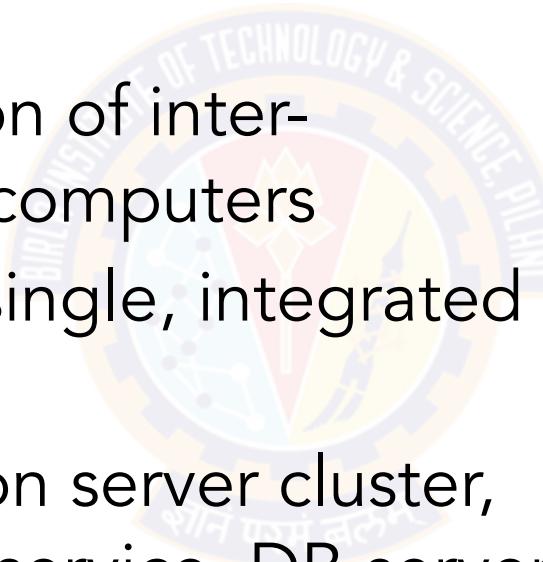
Topics for today

- What are parallel / distributed systems
- Motivation for parallel / distributed systems
- Limits of parallelism
- Shared Memory vs Message Passing
- Data access strategies - Replication, Partitioning, Messaging
- Cluster computing



Computer Cluster - Definition

- A cluster is a type of distributed processing system
 - ✓ consisting of a collection of interconnected stand-alone computers
 - ✓ working together as a single, integrated computing resource
 - ✓ e.g. web and application server cluster, cluster running a Cloud service, DB server cluster



Cluster - Objectives

- A computer cluster is typically built for one of the following two reasons:
 - ✓ High Performance - referred to as compute-clusters
 - ✓ High Availability - achieved via redundancy

An off-the-shelf or custom load balancer, reverse proxy can be configured to serve the use case

- **Question: How is this relevant for Big Data?**

Hadoop nodes are a cluster for performance (independent Map/Reduce jobs are started on multiple nodes) and availability (data is replicated on multiple nodes for fault tolerance)

Most Big Data systems run on a cluster configuration for performance and availability

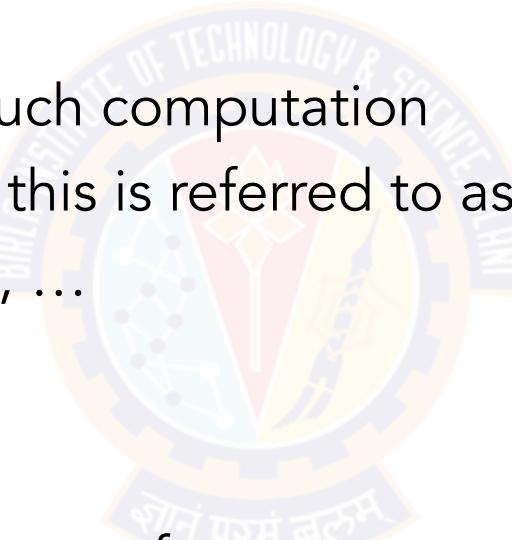
Clusters – Peer to Peer computation

- Distributed Computing models can be classified as:
 - ✓ Client Server models
 - ✓ Peer-to-Peer models
- based on the structure and interactions of the nodes in a distributed system
- Clusters within the nodes use a Peer-to-Peer model of computation.
- There may be special control nodes that allocate and manage work thus having a master-slave relationship.



Client-Server vs. Peer-to-Peer

- Client-Server Computation
 - ✓ A server node performs the core computation – business logic in case of applications
 - ✓ Client nodes request for such computation
 - ✓ At the programming level this is referred to as the request-response model
 - ✓ Email, network file servers, ...
- Peer-to-Peer Computation:
 - ✓ All nodes are peers i.e. they perform core computations and may act as client or server for each other.
 - ✓ bit torrent, some multi-player games, clusters



Cloud and Clusters

- A cloud uses a datacenter as the infrastructure on top of which services are provided
 - e.g. AWS would have a datacenter in many regions - Mumbai, US east, ... (you can pick where you want your services deployed)
- A cluster is the basic building block for a datacenter:
 - ✓ i.e. a datacenter is structured as a collection of clusters
- A cluster can host
 - ✓ a multi-tenant service across clients - cost effective
 - ✓ individual clients and their service(s) - dedicated instances

Can you draw a conceptual diagram to illustrate these cases ?

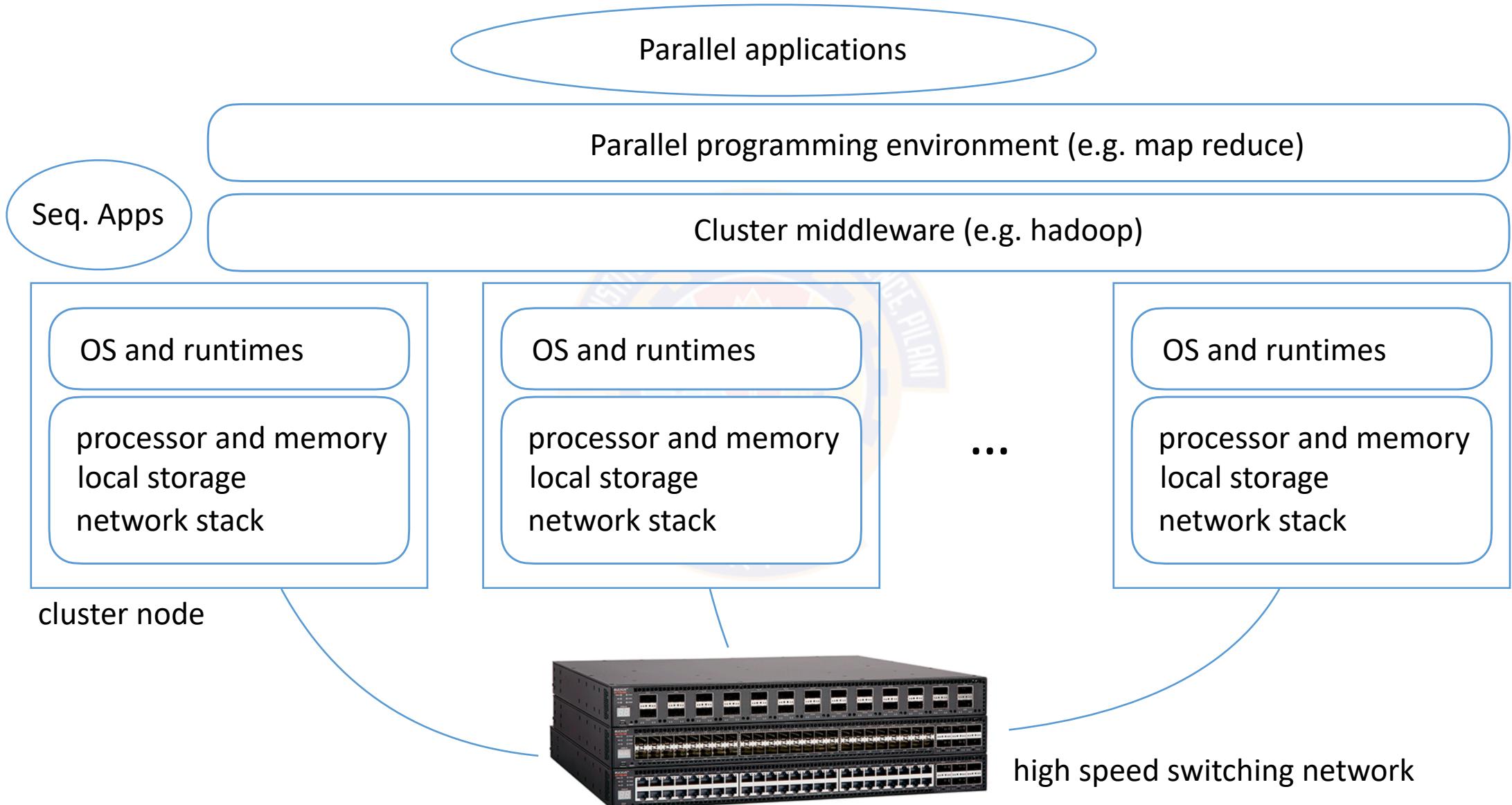
Motivation for using Clusters (1)

- Rate of obsolescence of computers is high
 - ✓ Even for mainframes and supercomputers
 - ✓ Servers (used for high performance computing) have to be replaced every 3 to 5 years.
- Solution: Build a cluster of commodity workstations
 - ✓ Incrementally add nodes to the cluster to meet increasing workload
 - ✓ Add nodes instead of replacing (i.e. let older nodes operate at a lower speed)
 - ✓ This model is referred to as a scale-out cluster

Motivation for using Clusters (2)

- Scale-out clusters with commodity workstations as nodes are suitable for software environments that are resilient:
 - ✓ i.e. individual nodes may fail, but
 - ✓ middleware and software will enable computations to keep running (and keep services available) for end users
 - ✓ for instance, back-ends of Google and Facebook use this model.
- On the other hand, (public) cloud infrastructure is typically built as clusters of servers
 - ✓ due to higher reliability of individual servers – used as nodes – (compared to that of workstations as nodes).

Typical cluster components



Cluster Middleware - Some Functions

Single System Image (SSI) infrastructure

- ✓ Glues together OSs on all nodes to offer unified access to system resources
 - ✓ Single process space
 - ✓ Cluster IP or single entry point
 - ✓ Single auth
 - ✓ Single memory and IO space
 - ✓ Process checkpointing and migration
 - ✓ Single IPC space
 - ✓ Single fs root
 - ✓ Single virtual networking
 - ✓ Single management GUI



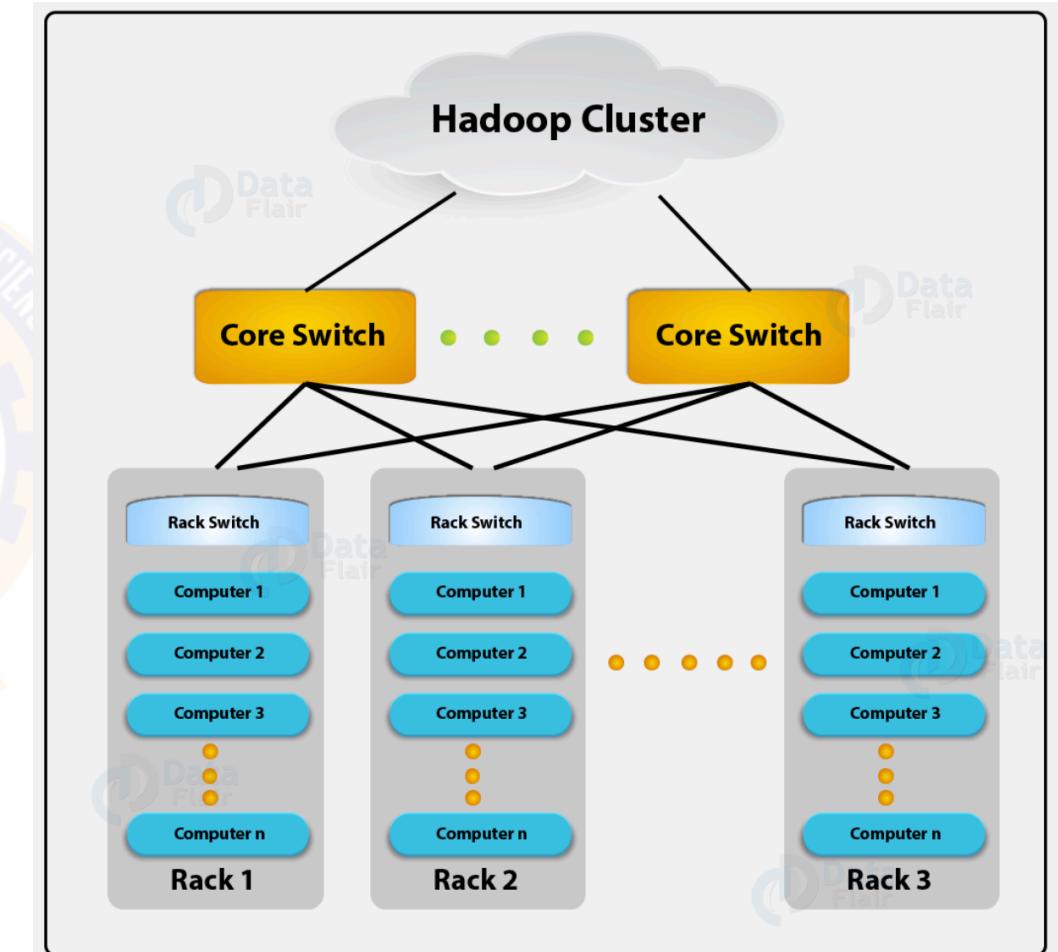
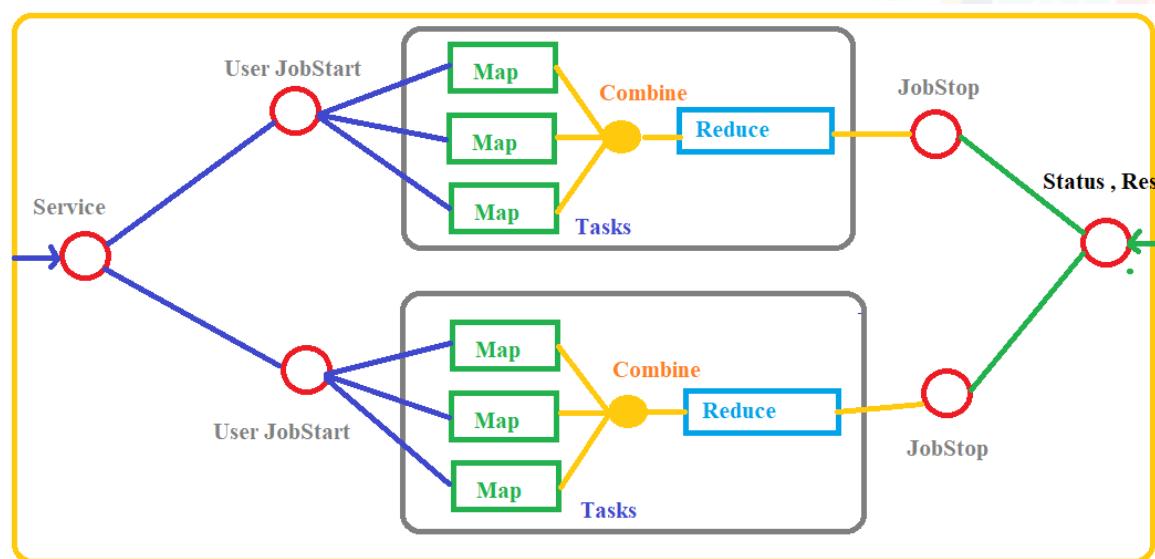
High Availability (HA) Infrastructure

- ✓ Cluster services for
 - ✓ Availability
 - ✓ Redundancy
 - ✓ Fault-tolerance
 - ✓ Recovery from failures

<http://www.cloudbus.org/papers/SSI-CCWhitePaper.pdf>

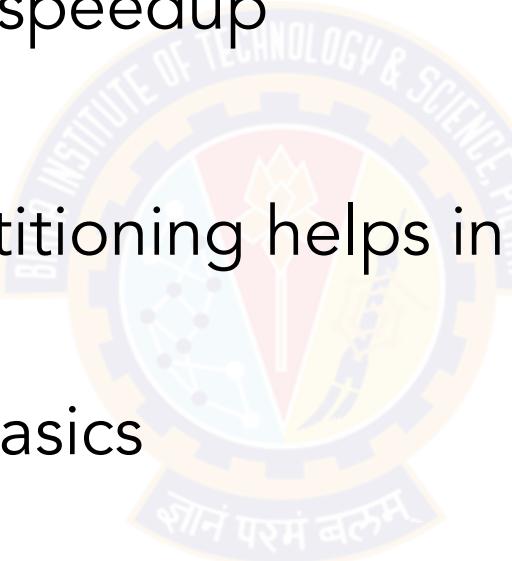
Example cluster: Hadoop

- A job divided into tasks
- Considers every task either as a Map or a Reduce
- Tasks assigned to a set of nodes (cluster)
- Special control nodes manage the nodes for resource management, setup, monitoring, data transfer, failover etc.
- Hadoop clients work with these control nodes to get the job done



Summary

- Motivation and classification of parallel systems
- Computing limits of speedup
- Message passing
- How replication, partitioning helps in Big Data storage and access
- Cluster computing basics





Next Session:
Big Data Analytics and Systems