innovate achieve lead

BITS Pilani

WILP

AMLCCLZG516
ML System Optimization

Murali Parameswaran

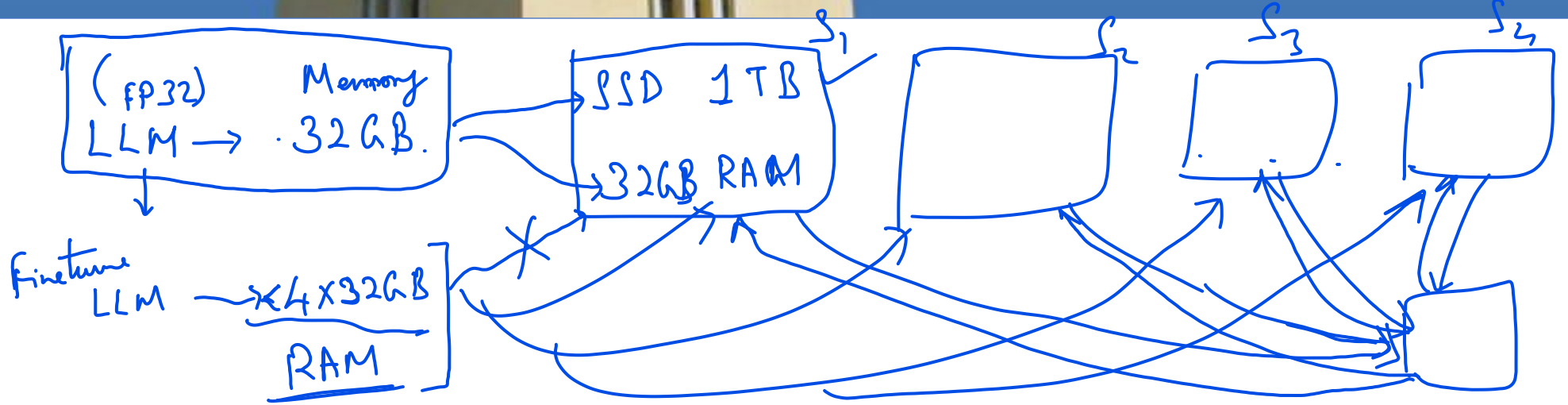
innovate achieve lead

BITS Pilani

WILP

AIML CLZG516
ML System Optimization

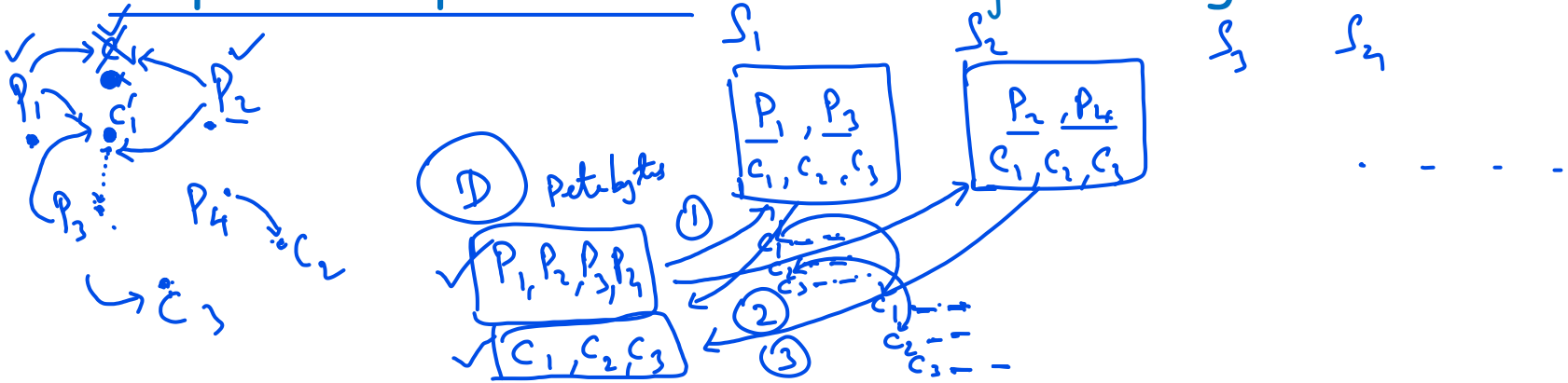
Review



K - means Clustering using map-reduce

- Step 1: "select representative points" for clusters $C_j = \{ c_j \}$ for j=1 to k
- Step 2:
  - map "compute distance" on D x Cs where Cs is the set of clusters
  - map "assign point to the closest cluster" on D
    - This requires: reduce min on point-cluster distances
- Step 3: for each cluster C_j compute its centroid (i.e., mean)
  - map on Cs:
    - $c_j = (\text{reduce} + C_j)/|C_j|$
- Repeat Steps 2 and 3 until all c_j converge

D x Cs = { (x, c_j) ... }
map comp_dist D x Cs



K – means Clustering using map-reduce

- Step 1: "select representative points" for clusters
- Step 2:
  - map "compute distance" on D x Cs where Cs is the set of clusters
  - map "assign point to the closest cluster" on D
    - This requires: reduce min on point-cluster distances
- Step 3: for each cluster C_j compute its centroid (i.e., mean)
  - map on Cs:
    - $c_j = (\text{reduce} + \ C_j)/|C_j|$
- Repeat Steps 2 and 3 until all c_j converge

{ (xi, d1j) }
= map comp_dist D
xi is a point in D
dij = distances of xi to clusters

reduce min dij

This reduce is required to return the cluster (with the min distance) and not the min distance:

Refer to reduce-key vs. reduce-val in Spark!

K – means Clustering

- Exercise: Implement k-means clustering using map and reduce.
- [Hints:
  - Step 1: "select k representative points" for clusters (randomly)
  - Step 2
    - map "compute distance" on D x Cs where Cs is the set of clusters
    - map "assign point to the closest cluster" on D
      - This requires: reduce min on point-cluster distances
  - Step 3b: compute the centroid (i.e., mean of) C_j
    - c_j = (reduce + C_j)/$|C_j|$

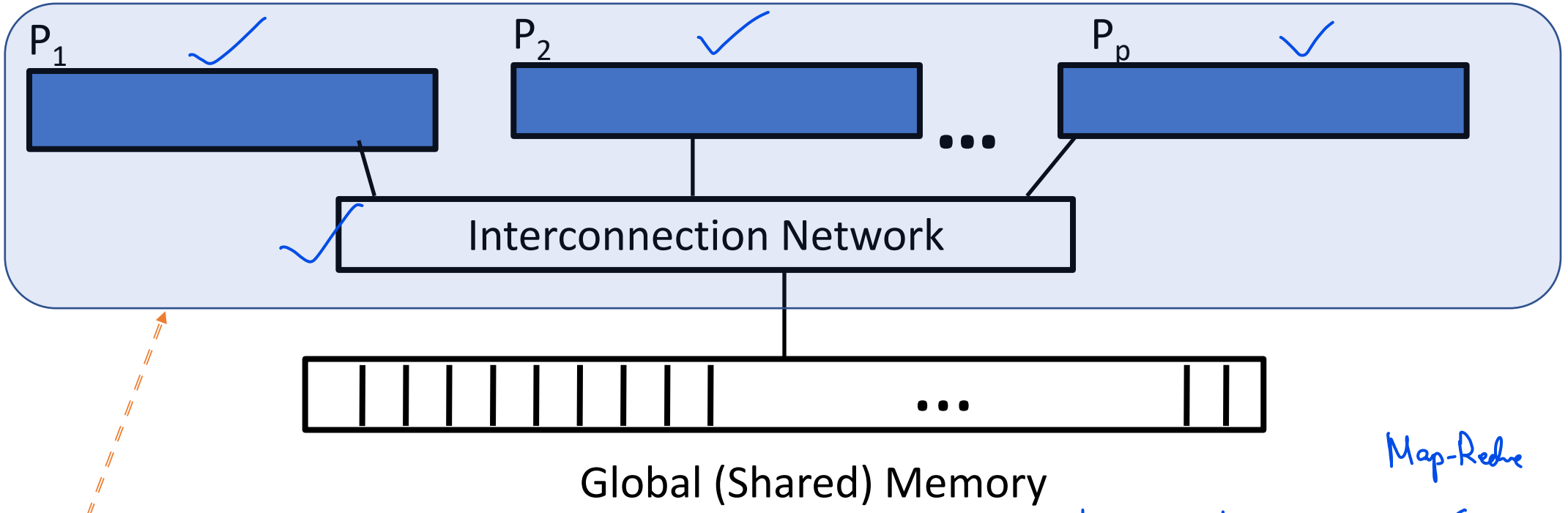
This follows a programming pattern named ***iterative map-reduce*** where map-reduce programming steps applied inside a loop.

Exercise: Speedup of k-means using map-reduce

- For each of the steps:
  - Calculate the speedup (and the number of processors)
- T_{seq} = I*(|D|*(k+k)+(k*|C|)) [Step 2: k distances req. k steps;
- min. computation req k-1 steps;
  - I is number of iterations
- T_{par} (p) = I*(|D|/p*(k+k)+(|C|) - assuming step 3 is done with only k processors; |D|/p points per processor in step 2
- p processors; k<p
- Speedup (p) = T_{seq} / T_{par} = (|D|*2*k + k*|C|)/((|D|/p)*(k+k)+|C|)
- ~p (close to ideal)

Parallel Programming: *Shared Memory* Model

So far we have looked at a target environment that uses a shared memory model:



e.g. a multi-core chip

Map-Reduce

harddisk-Memory - ✓ ✗

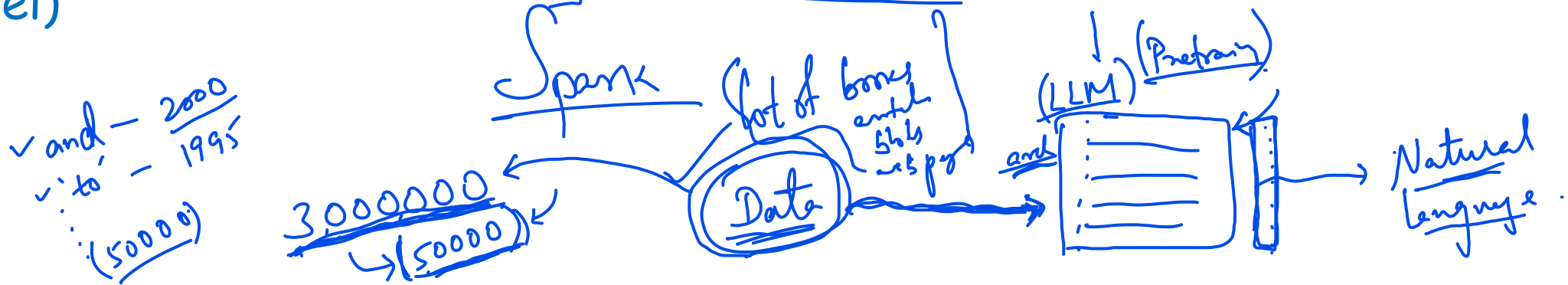
Compute Mem - ✗ ✗

Multi-threaded Programming:

each thread runs on a separate core

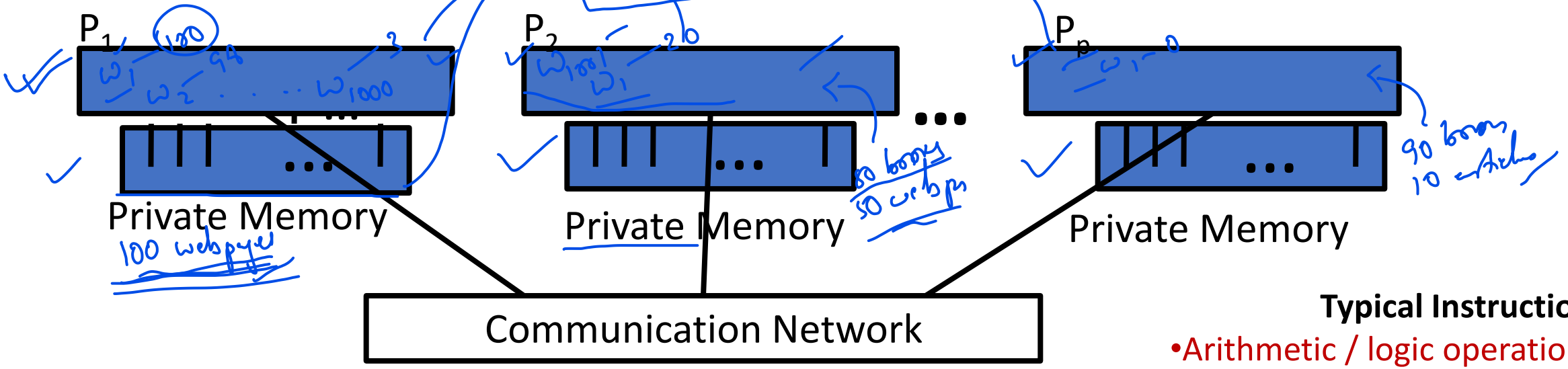
Large volumes of Data

- When the volume of data that we have to process is in 100s of GB if not in TB,
  - Then all the data cannot be kept in one computer
  - And brought into memory for processing
- We a model where data can be stored on multiple computers (i.e., their hard disks)
  - All of which participate in computing.
- This leads us to a distributed computing model (aka message passing model)



Algorithm Design – Parallel: *Distributed Memory* Model

Target environment:



Typical Instructions

- Arithmetic / logic operations,
- Load / Store, and
- Jump / Branch
- **Send / receive**

Distributed Programming:

- a program is made of multiple processes
- *each process runs on a separate computer*
- *processes exchange messages (i.e., data for collaboration)*

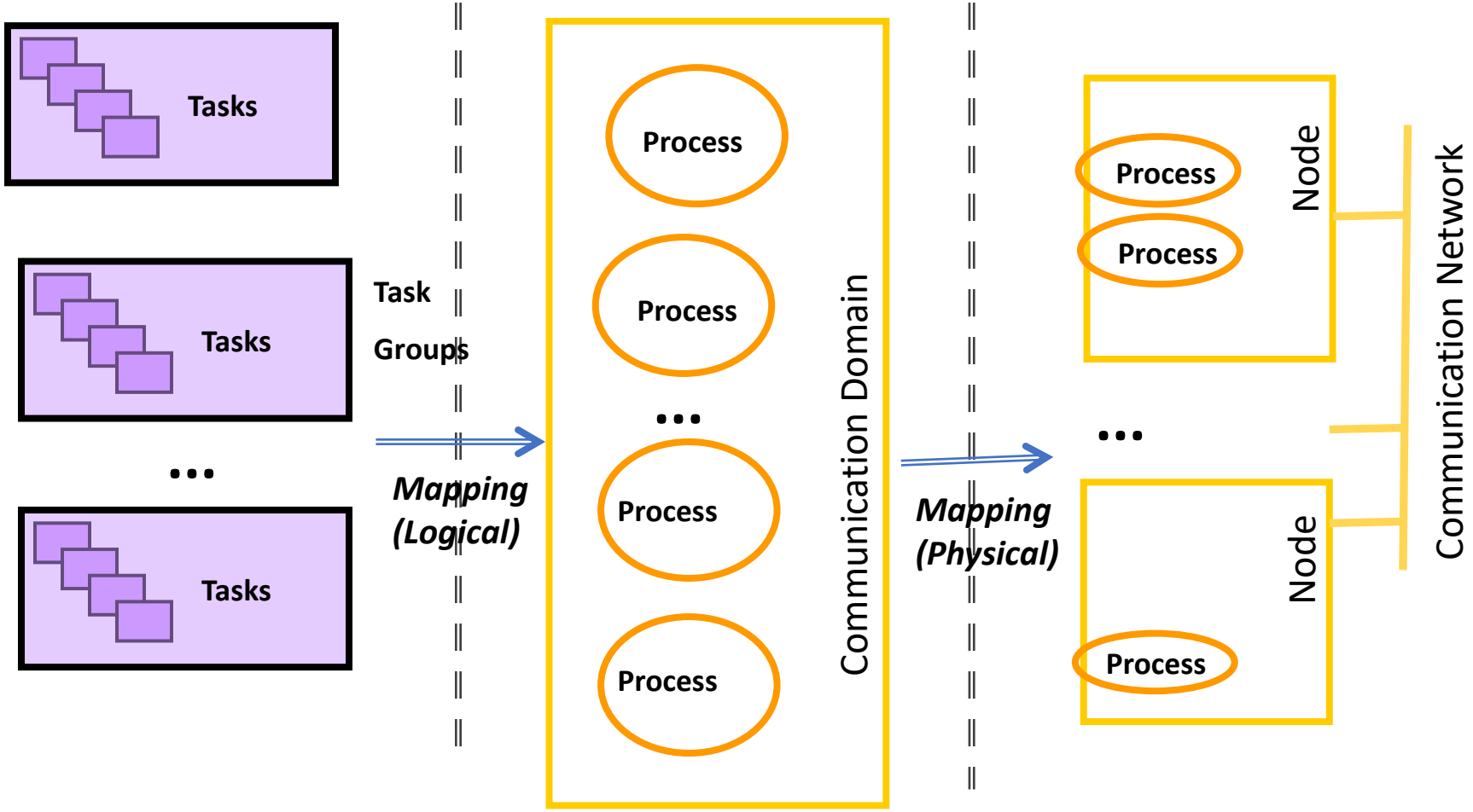
e.g. a cluster

Parallel / Distributed Computing

- A parallel or distributed program is made of multiple tasks that *collaborate* (to achieve a common outcome).
  - Collaboration is achieved by communication:
    - exchange data using shared memory
      - i.e. Task A writes to location L; Task B reads from location L
    - exchange data by passing messages
      - i.e. Task A sends a message to Task B; Task B receives the message from Task A

- Multiple processes each with its own address space:

E.g. processes run on nodes connected in a network : (i) each node runs its own OS and (ii) each process is allocated its own (logical) address space that is mapped onto the (physical) resources of that node



Exercise

- Implement k-means on Spark
- Calculate – on paper - speedup of k-means using a cluster:
  - Calculate communication cost
- Understand:
  - the difference between this and the previous calculation (for shared memory programming)
- Questions:
  - How do you distribute the data initially?
  - Cost?
  - Pattern?

innovate achieve lead

BITS Pilani

WILP

AIML CLZG516
ML System Optimization

Scale-out Clusters – Distributed Memory Programming

Distributed Algorithms – Communication Overhead

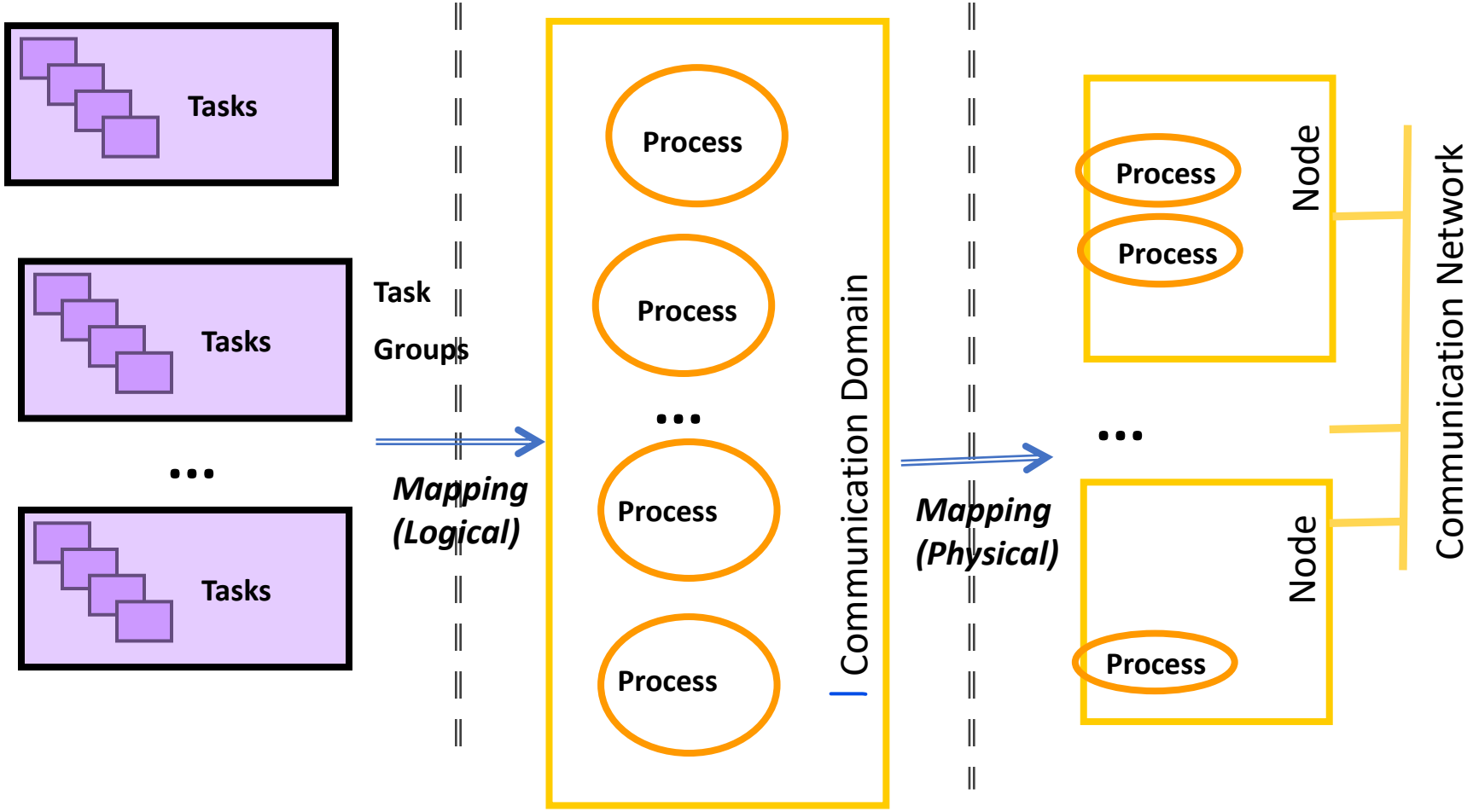
Distributed ML Algorithms

- Example: k-Means

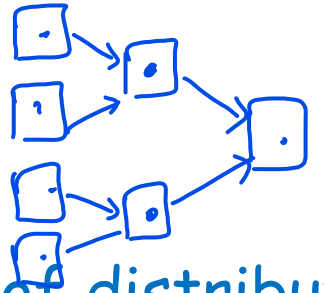
- Model Parallelism

- Multiple processes each with its own address space:

E.g. processes run on nodes connected in a network : (i) each node runs its own OS and (ii) each process is allocated its own (logical) address space that is mapped onto the (physical) resources of that node



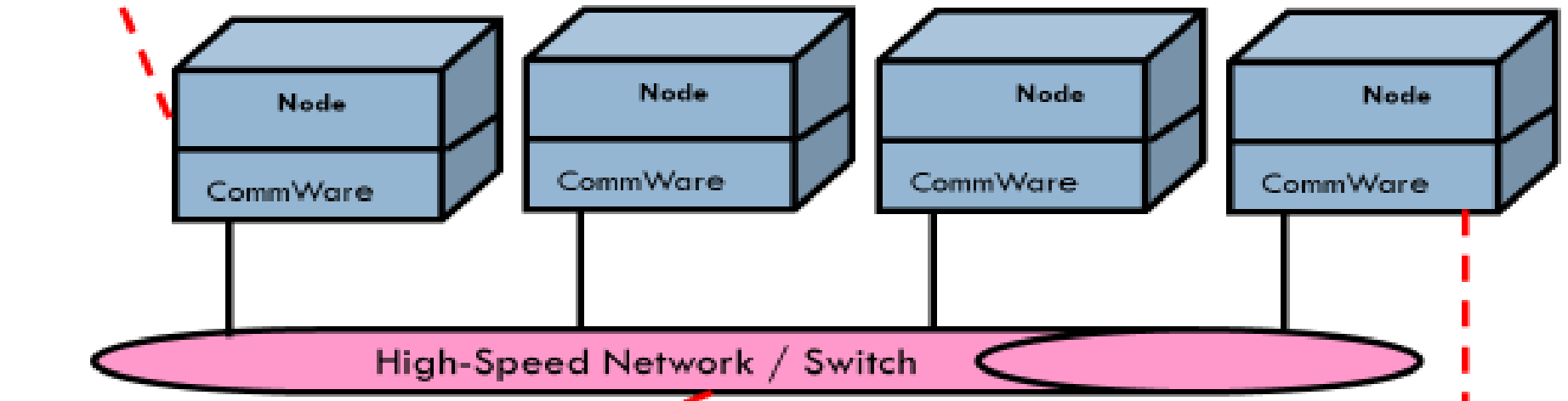
(Compute) Clusters



- Clusters are a modern example of distributed computing often used for high performance or big data computing.
  - Search engines - in the mid-90s - were using supercomputers at the back-end.
    - High obsolescence rate:
      - The supercomputers were getting obsolete (or unable to meet the computing needs) in five years or less.
    - Replacing a super-computer every five years was not cost-effective.
      - Clusters were available since 1980s and
      - Google realized that they are cost effective if older compute-nodes are replaced incrementally.
- Today, they are known as *scale out clusters* or *commodity clusters* and are used
  - with tens of thousands of nodes by Google, Facebook, Netflix, and others
  - for large scale processing of data.

Typical Cluster: Components - Base

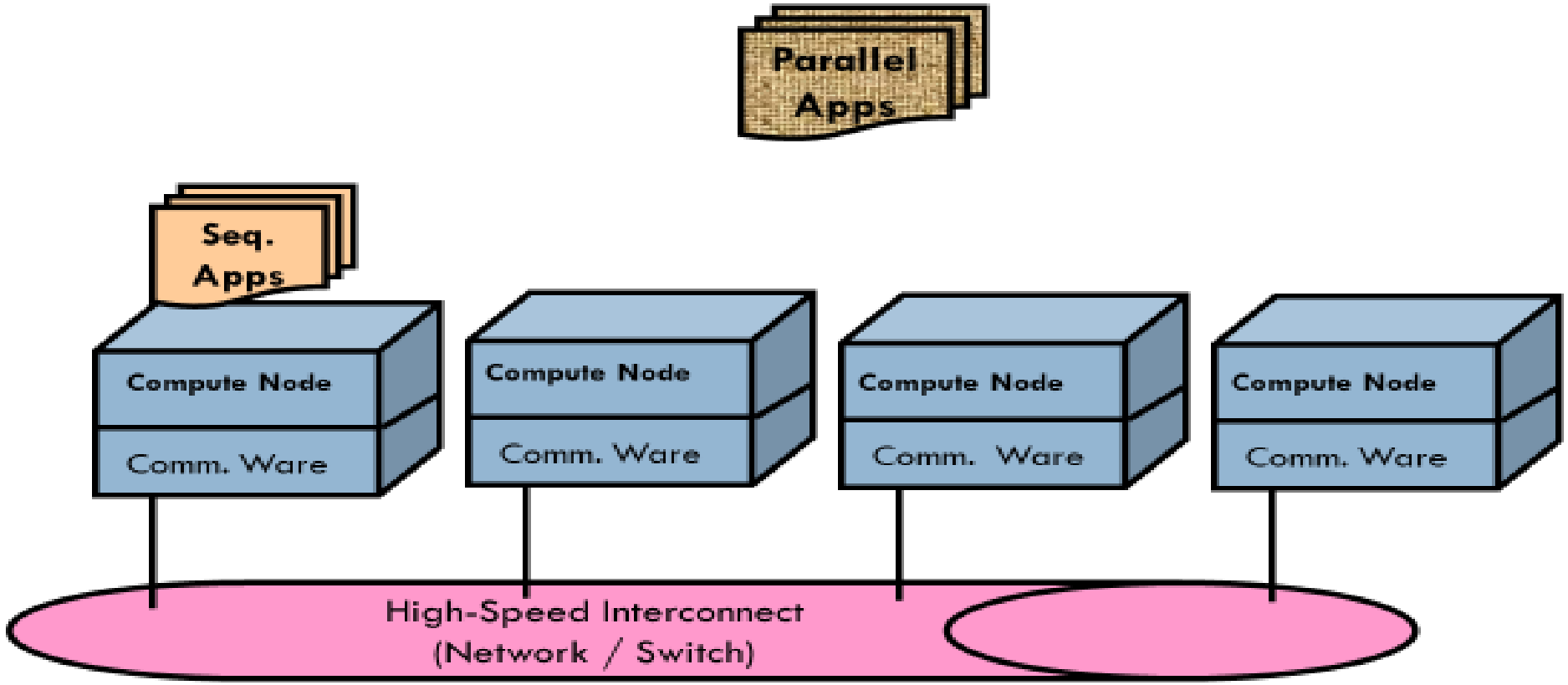
- **Processor+Memory+Storage**
- **OS+Run-time environment**



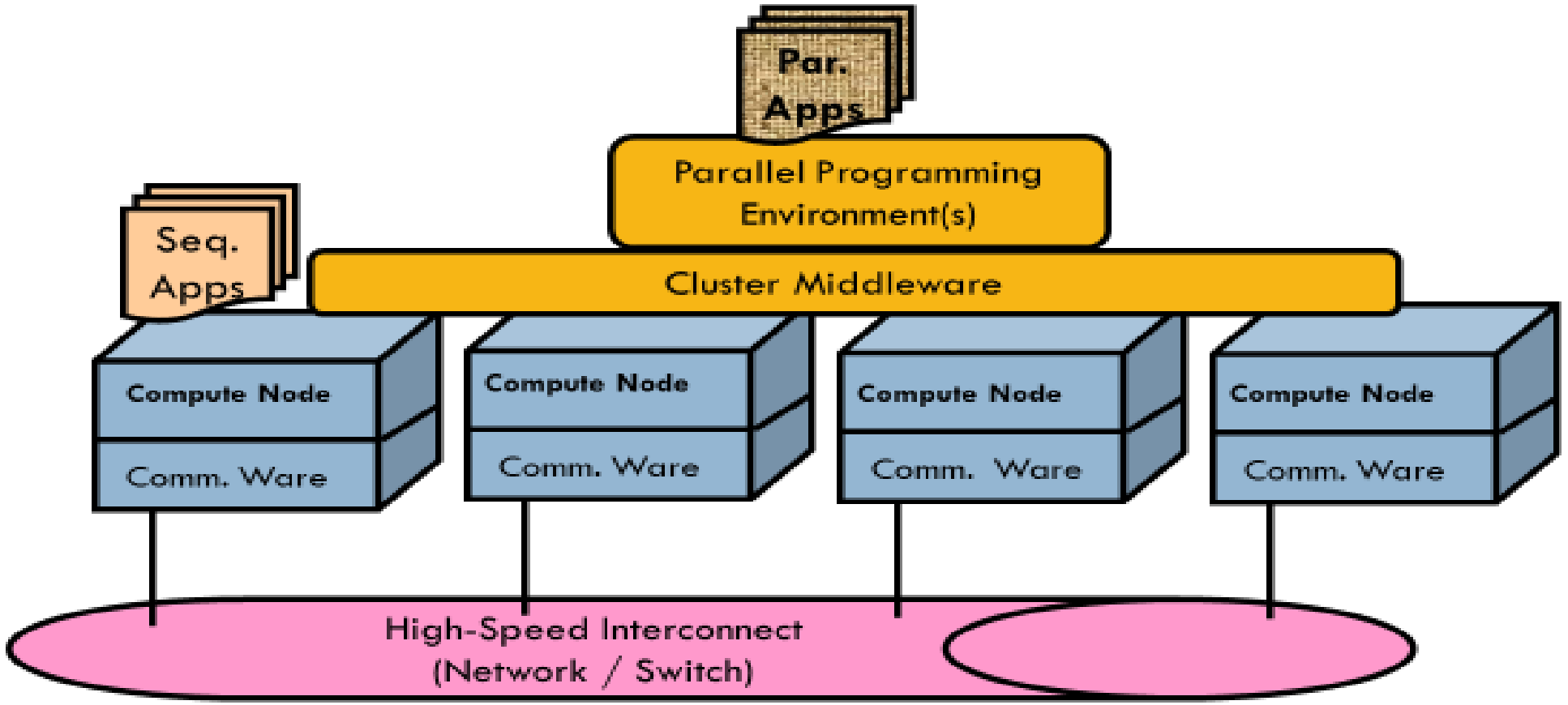
Gigabit Ethernet or Myrinet/Infiniband

- **Network Interface Cards**
- **Fast Communication Protocols and Services**

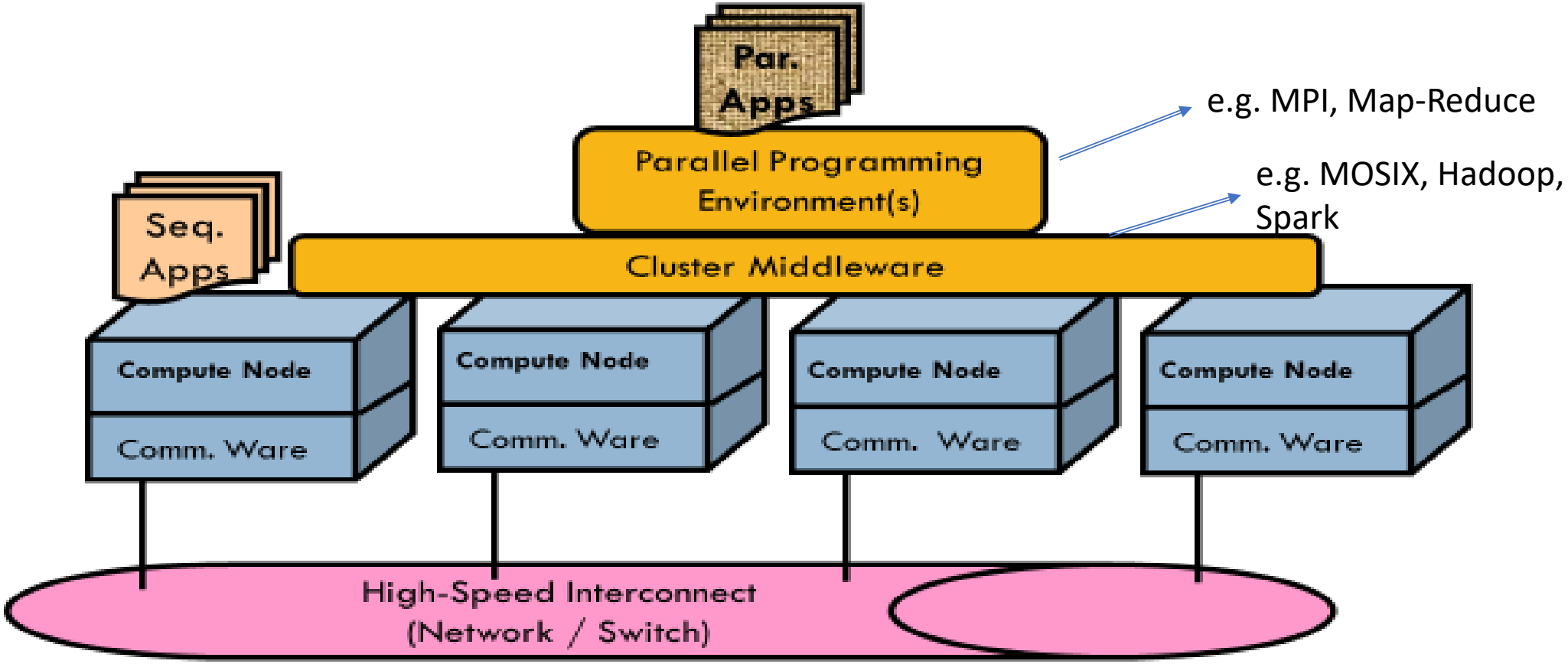
Typical Cluster - Requirements



Typical Cluster Architecture



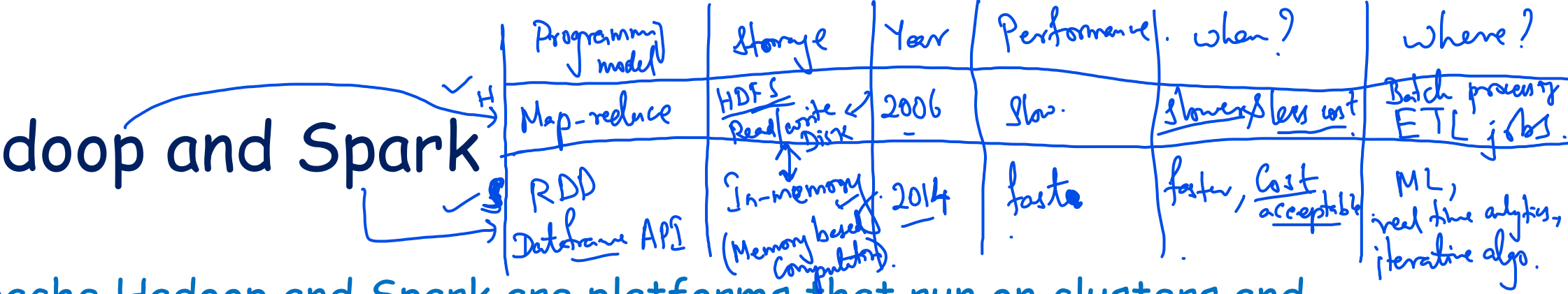
Typical Cluster Architecture



Scale-out Clusters

- Case in-point:
  - Search engines - in the mid-90s - were using supercomputers at the back-end.
  - High obsolescence rate:
    - The supercomputers were getting obsolete (or unable to meet the computing needs) in five years or less.
  - Replacing a super-computer every five years was not cost-effective.
    - Clusters were available since 1980s and
    - Google realized that they are cost effective if older compute-nodes are replaced incrementally.
- Today, they are known as *scale out clusters* or *commodity clusters* and are used
  - with tens of thousands of nodes by Google, Facebook, Netflix, and others
  - for large scale processing of data.

Hadoop and Spark



- Apache Hadoop and Spark are platforms that run on clusters and support map-reduce programming
- Hadoop supports programming with data stored on files
- Spark supports in-memory distributed programming with RDDs (Resilient Distributed Data)
  - Programmable Data structures
    - Distributed in the memories of multiple nodes in a distributed system (e.g. a cluster)

Exercise

- Implement k-means on Spark
- Calculate – on paper - speedup of k-means using a cluster:
  - Calculate communication cost
- Understand:
  - the difference between this and the previous calculation (for shared memory programming)
- Questions:
  - How do you distribute the data initially?
  - Cost?
  - Pattern?

K – means Clustering

Inputs: Dataset D, A positive integer k

Output: A partition Cs of D with size k

(i.e., k disjoint clusters covering all points in D)

Approach:

1. Chose k data points (as representatives) from D, say c_1, c_2, … c_k
2. Assign each point x in D to the cluster C_j ;
   that has the closest center c_j
3. Choose k new representatives based on
   minimizing local average distance within each cluster [Notion of ***cohesion***]
4. Iterate steps 2 and 3 until (the cluster centers converge)

K – means Clustering using map-reduce

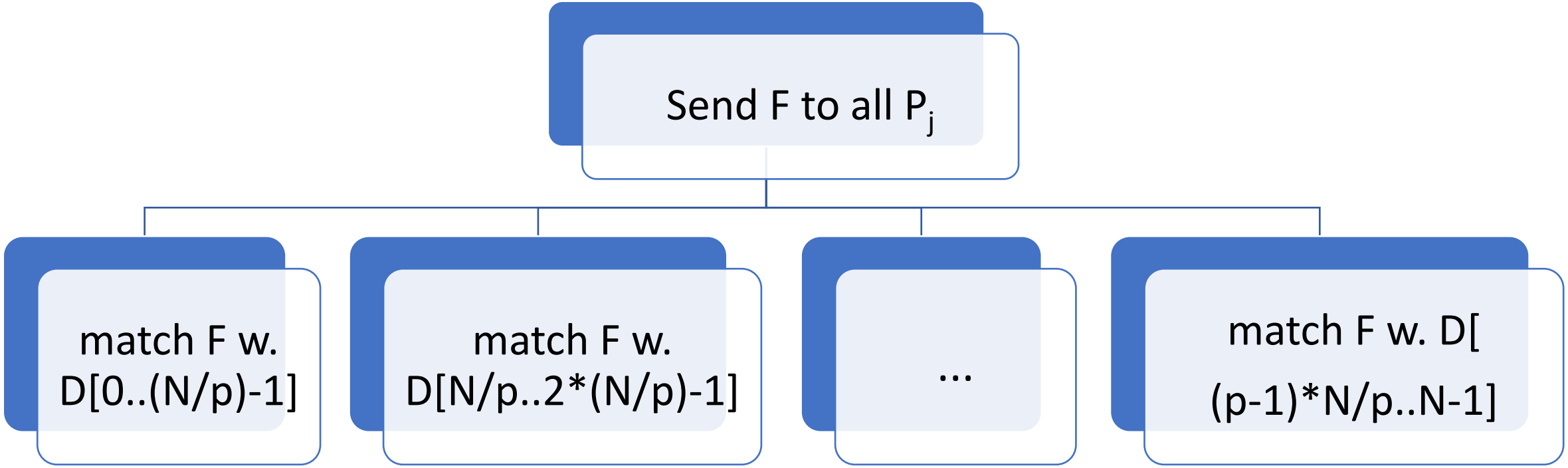
- Step 1: "select representative points" for clusters $C_j = \{ c_j \}$ for j=1 to k
- Step 2:
  - map "compute distance" on D x Cs where Cs is the set of clusters
  - map "assign point to the closest cluster" on D
    - This requires: reduce min on point-cluster distances
- Step 3: for each cluster C_j compute its centroid (i.e., mean)
  - map on Cs:
    - c_j = (reduce + C_j)/$|C_j|$
- Repeat Steps 2 and 3 until all c_j converge

D x Cs = { (x, c_j) … }
map comp_dist D x Cs

Data Parallel Execution - Examples

Fingerprint Matching:

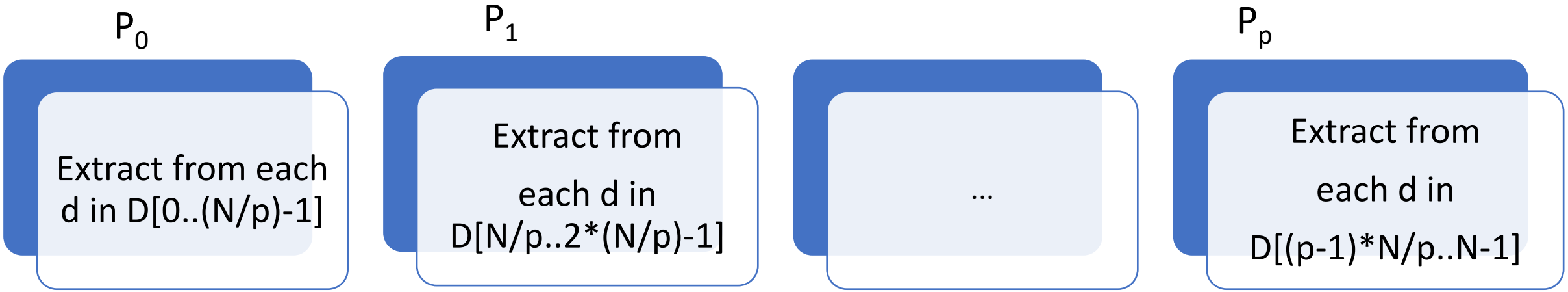
- Match a given print F with a database D of prints available;
- Assume D is distributed.



Communication cost?

Data Parallel Execution - Examples

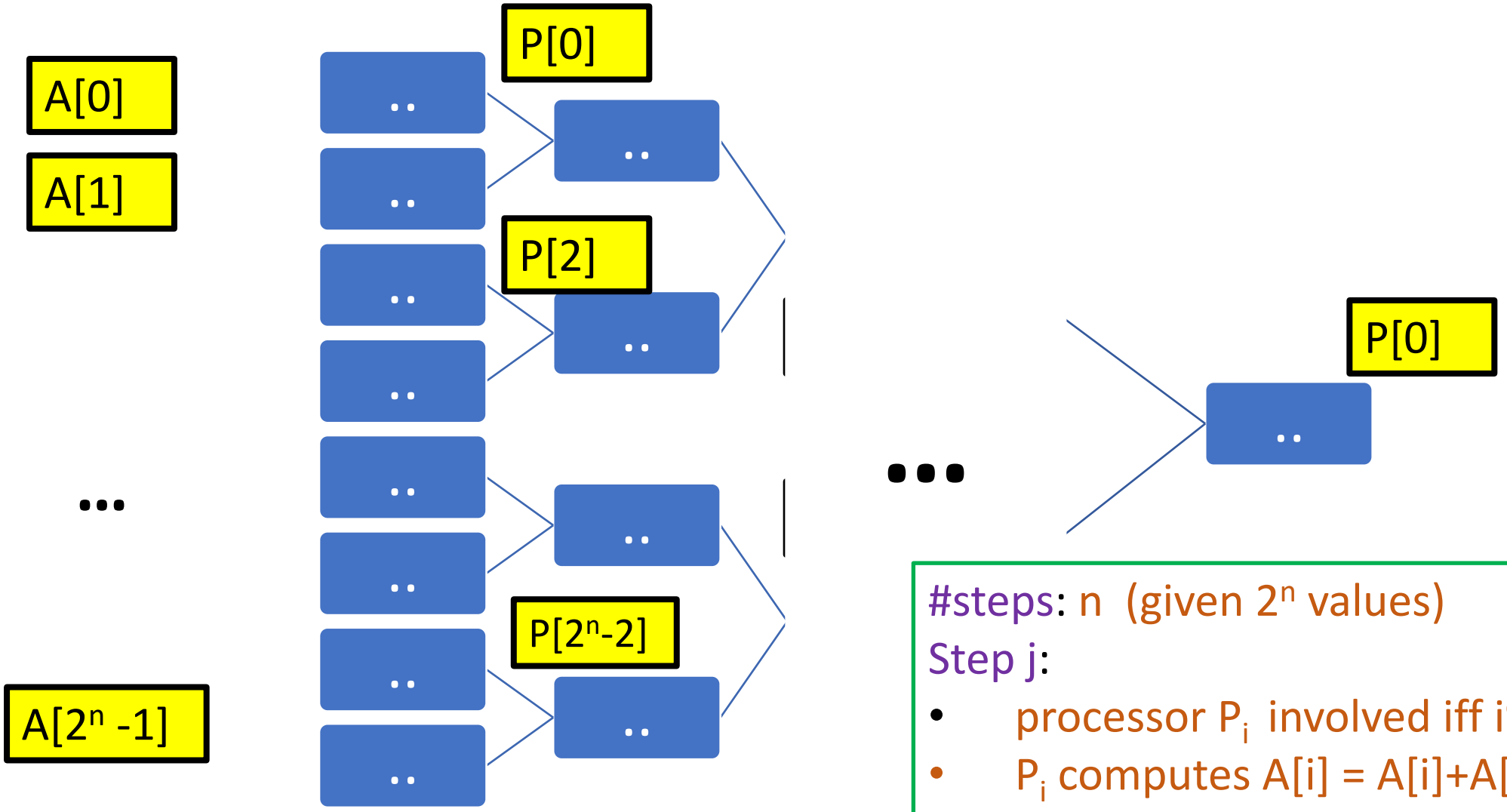
Extracting keywords from each document in a distributed collection D:
[Assume|D| = N]



P_0 : Collect keywords from all processors

Communication cost?

Example: Parallel Summation (Shared memory)



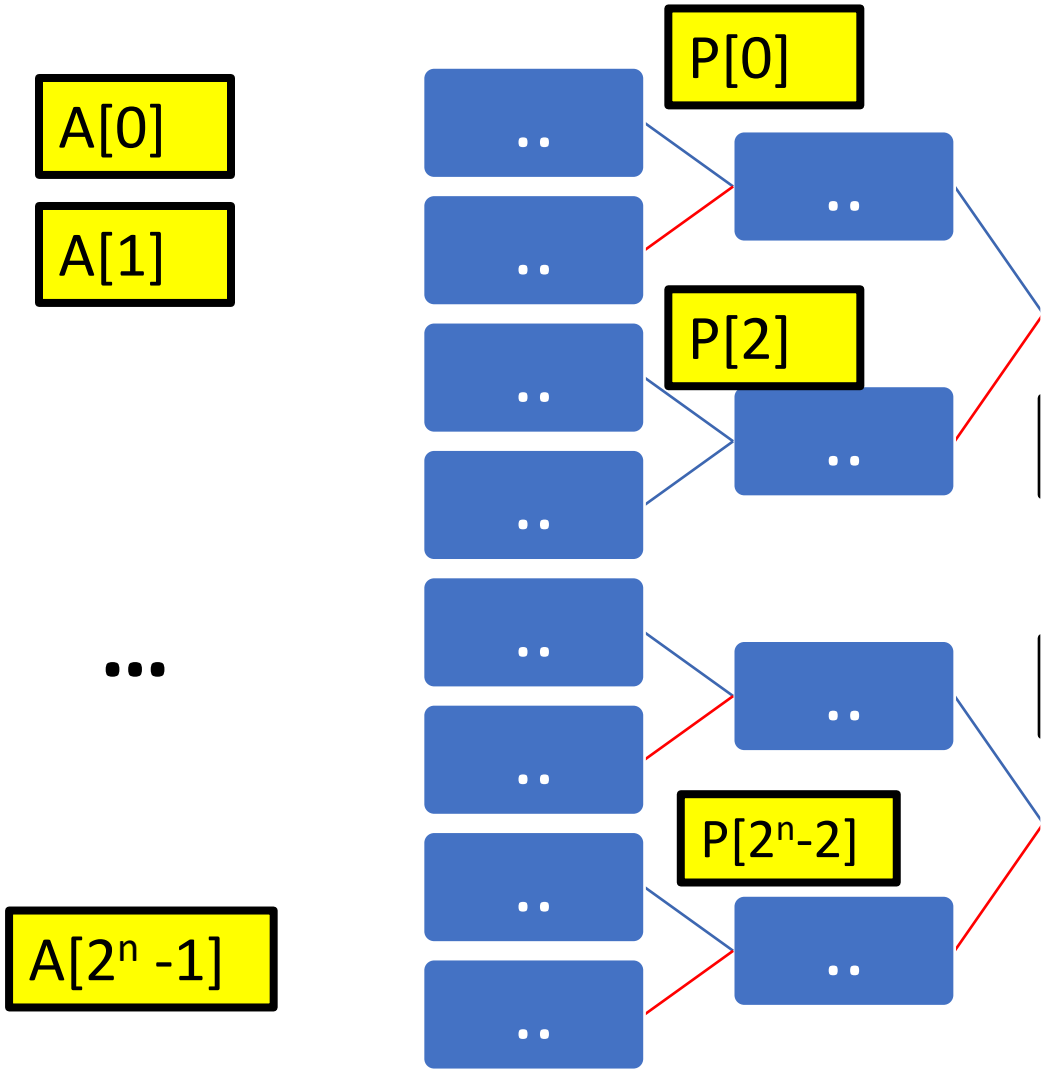
#steps: n (given 2^n values)

Step j:

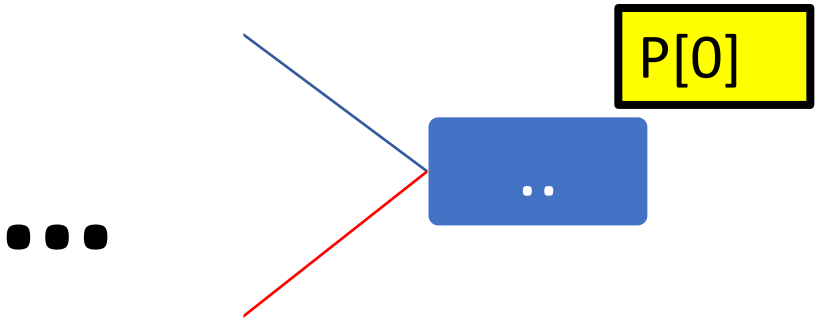
- processor P_i involved iff $i\%2^j == 0$
- P_i computes $A[i] = A[i]+A[i+2^{j-1}]$

A is in (global) shared memory

Example: Parallel Summation (Distributed memory)



indicates communication



#steps: n (given 2^n values)

Step j: (P_i has A[i])

- processor P_i involved iff $i\%2^j == 0$
- $A[i+2^{j-1}]$ is sent to P[i]
- P_i computes $A[i] = A[i]+A[i+2^{j-1}]$

Algorithm Design – Speedup: Caveat

- Summation by reduction executed on a distributed system:
  - before processor P_i performs $A[i] = A[i] + A[i+2^{j-1}]$
    - there is communication involved: $A[i+2^{j-1}]$ sent from $P_{i+2^{j-1}}$ to P_i
- The time complexity of a reduction algorithm will increase
  - i.e. *speedup will decrease*.

- Speedup (of summation by reduction of **N** values with **N/2** processors):

$$S(N,p) = T_{seq}(N) / T_{par}(N,N/2)$$
$$= ((N-1)*T_{add}) / (\log N*(T_{add} + T_{msg}))$$

where T_{add} is the time taken for adding two values
and T_{msg} is the time taken for sending (and receiving) a message

Algorithm Design – Speedup: Caveat

- Speedup (of summation by reduction of **N** values with **N/2** processors):

  $S(N,N/2) = T_{seq}(N) / T_{par}(N,N/2)$

  $\approx (N/\log N) * (1 / (1 + T_{msg}/T_{add}))$

- T_{msg} includes
  - set-up cost (for *send* and *recv*), which is typically fixed and
  - transmission cost (which depends on the length of the message)
- Typically, $T_{msg} >> T_{add}$

	1990s	Today (2020s)
T_{msg}	*ms*	*us*
T_{add}	*us* (or 10s of *ns*)	< 1*ns*

ms milli-seconds
us micro-seconds
ns nano-seconds

- Implication: Speedup << N/logN

Reduce as in Google's Map-Reduce

- Pragmatics of ***reduce*** (*similar to that of* **map**):
  - (Assumption) Input Data partitioned and stored
  - Management of Messaging / Communication
  - Spawning of processes, Scheduling, and Load Balancing
  - Recovery from process / node failures

Communication is managed transparently:

i.e., programming is easier,

but communication cost is still the same!

K – means Clustering: Algorithm outline

Inputs: Dataset D, A positive integer k

Output: A partition Cs of D with size k

(i.e., k disjoint clusters covering all points in D)

Approach:

1. Chose k data points (as representatives) from D, say c_1, c_2, … c_k
2. Assign each point x in D to the cluster C_j :
   that has the closest center c_j
3. Choose k new representatives based on
   minimizing local average distance within each cluster [Notion of ***cohesion***]
4. Iterate steps 2 and 3 until (the cluster centers converge)

K – means Clustering on a distributed system

- Assume D is distributed in **p** processors (or nodes)
  - Each processor p_i has D_i of size (|D|/p points)
- Step 1: "select representative points" for clusters $C_j = \{ c_j \}$ for j=1 to k
- Step 2:
  - Communicate c_j (for j = 1 to k) to all p nodes (i.e. processors)
  - On each processor p_i
    - for each point x in D_i
    - {
    - for j=1 to k { d_j = distance(x,c_j); }
    - assign x to the cluster with minimum d_j
    - } // each p_i has k clusters

…

K – means Clustering on a distributed system

- Assume D is distributed in **p** processors
  - Each processor p_i has D_i of size (|D|/p points)
- Step 1: "select representative points" for clusters $C_j = \{ c_j \}$ for j=1 to k
- Step 2:
  - Communicate c_j (for j = 1 to k) to all p nodes (i.e. processors)
  - On each processor p_i, "compute distances":
    - for each point x in D_i
    - {
    - for j=1 to k { d_j = distance(x,c_j); }
    - assign x to the cluster with minimum d_j
    - } // each p_i has k clusters
- Step 3: collect and distribute clusters to k different processors
  - for each cluster C_j in processor p_j
    - c_j = (reduce + C_j)/|C_j|
- Repeat Steps 2 and 3 until all c_j converge

K – means Clustering on a distributed system

- Assume D is distributed in **p** processors
  - Each processor p_i has D_i of size (|D|/p points)
- Step 1: "select representative points" for clusters $C_j = \{ c_j \}$ for j=1 to k
- Step 2:
  - Communicate c_j (for j = 1 to k) to all p nodes (i.e. processors)
  - On each processor p_i ← map *(distributed)*
    - for each point x in D_i
    - {
    - for j=1 to k { d_j = distance(x,c_j); } ← map *(shared mem.)*
    - assign x to the cluster with minimum d_j ← reduce *(shared mem.)*
    - } // each p_i has k clusters
- Step 3: collect and distribute the clusters to **k** different processors
  - for each cluster C_j in processor p_j (j=1 to k) ← map *(distributed)*
    - c_j = (reduce + C_j)/|C_j| ← reduce *(shared mem.)*
- Repeat Steps 2 and 3 until all c_j converge

K – means on a distributed system: Communication cost

- Assume D is distributed in **p** processors
  - Each processor p_i has D_i of size (|D|/p points)
- Step 1: "select representative points" for clusters $C_j = \{ c_j \}$ for j=1 to k
- Step 2:
  - Communicate c_j (for j = 1 to k) to all p nodes (i.e. processors)
  - On each processor p_j, "compute distances":
    - for each point x in D_i
    - {
    - for j=1 to k { d_j = distance(x,c_j); }
    - assign x to the cluster with minimum d_j
    - } // each p_i has k clusters
- Step 3: collect and distribute clusters to k different processors
  - for each cluster C_j in processor p_j
    - c_j = (reduce + C_j)/|C_j|
- Repeat Steps 2 and 3 until all c_j converge

k values

|D| values

K – means on a distributed system: Implementation

- Use MPI (Message Passing Interface) to program on a distributed system:
  - Defaults to a SPMD model (i.e. same program/code on multiple processors but different data)
    - i.e. map is implicit
  - Constructs available for send, recv, reduce
- Alternative:
  - Use Hadoop map-reduce
  - Data stored on files (of nodes in a cluster)

K – means on a shared memory system: Implementation

- Use OpenMP library or P-Threads library) to program on a shared memory) system:
  - Defaults to a SPMD model (i.e. same program/code on multiple processors but different data)
    - i.e. map is implicit
  - Constructs available for send, recv, reduce
- Alternative:
  - Use multi-threaded programming in Java

K-means on a commodity cluster

- A cluster of nodes (workstations/servers)
  - Each node has a multi-core processor
- Use MPI programming on the cluster (in C or in Java)
  - The code for each node can be multi-threaded
    - Use OpenMP or P-Threads

K – means on Spark

- Distributed memory model (i.e. a cluster) where
  - Every node can be a shared memory processor (i.e. a multi-core processor)
  - Supports map and reduce as constructs
  - Programming in Java or Scala or other languages
  - In-memory processing
    - i.e. data may be stored on files,
    - but map and reduce work on data structures (RDDs) stored in memory
      - Unlike on Hadoop where map and reduce operate on files

Task Parallelism – Example

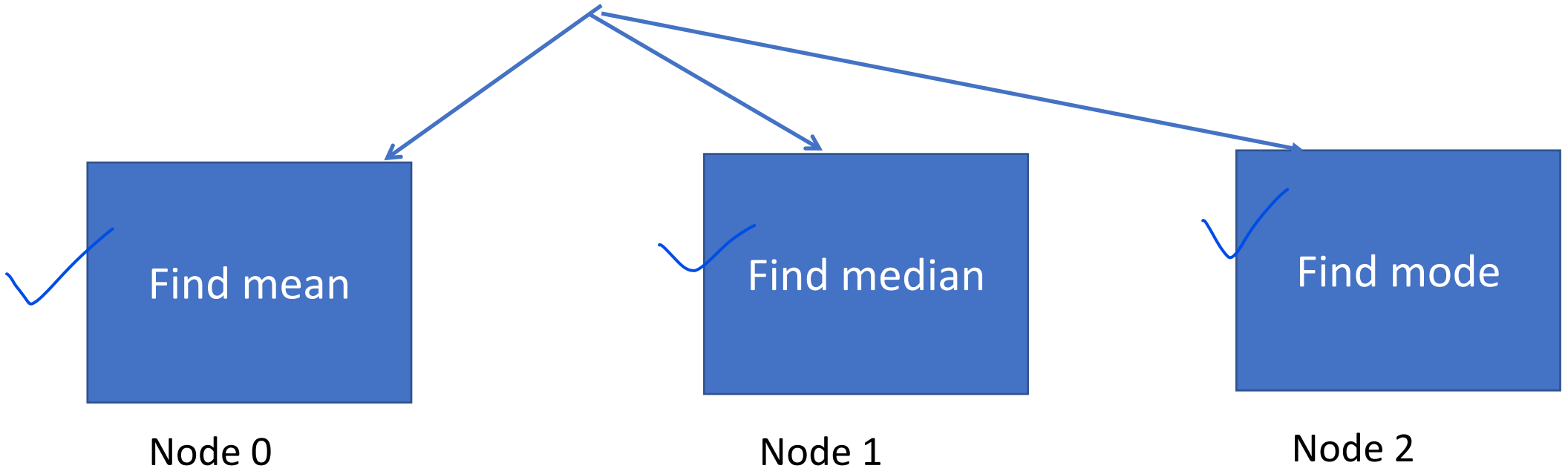
Recall Task Parallelism on Shared memory computers

The same is possible on Distributed memory systems!

Communication?

Assumption:

Input data is available on all nodes.

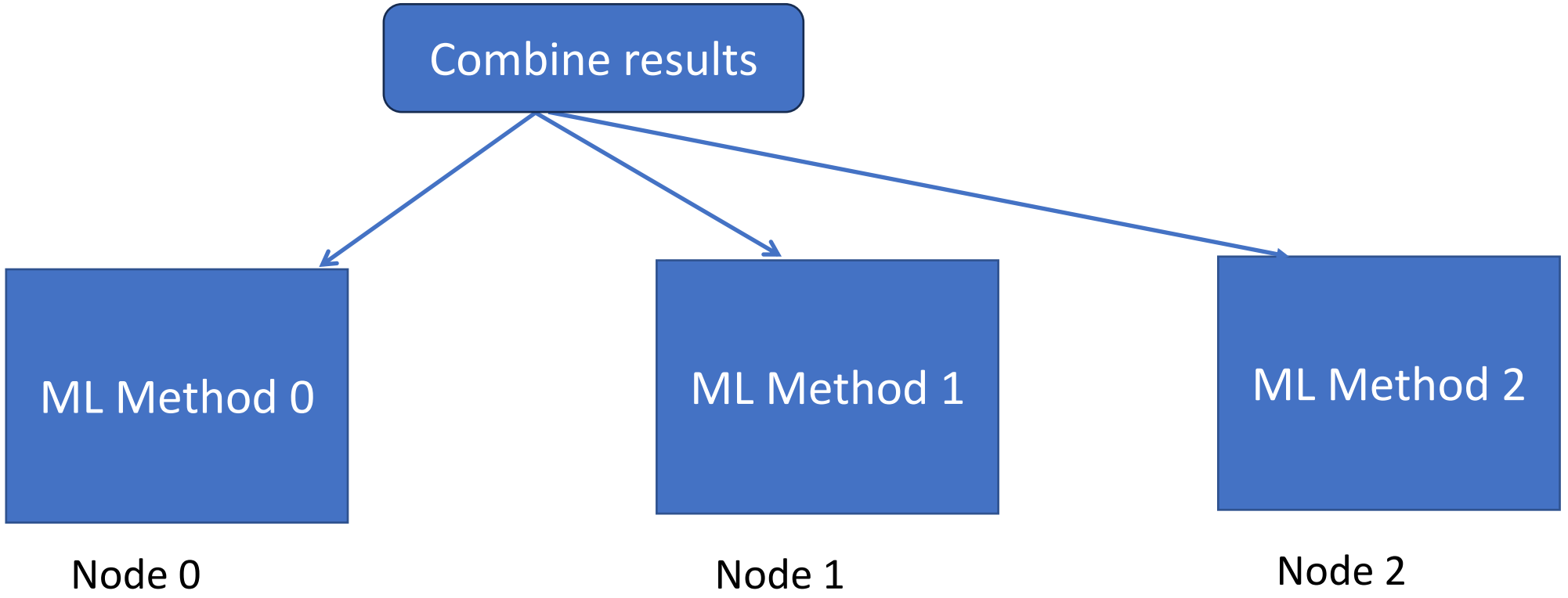


Ensemble Methods

- Task Parallelism – in the context of ML – is also known as Model parallelism
  - Recall the Bagging solution:
    - Multiple different models from the same data (via sampling)
  - Ensemble methods
    - Multiple different models from the same data (via different training methods)

Model Parallelism – Example: Ensembles

Assumption:
Input data is available on all nodes.



Communication Cost?