BITS Pilani
WILP

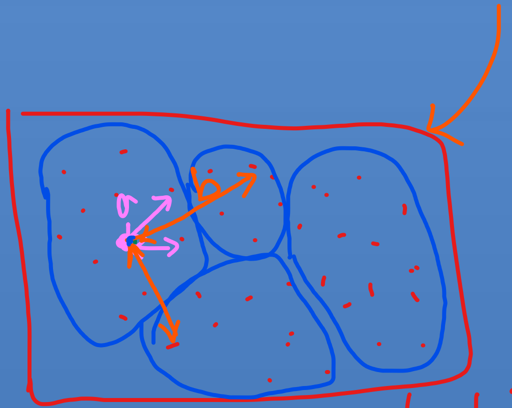
AMLCCLZG516
ML System Optimization

Murali Parameswaran

innovate achieve lead

BITS Pilani

WILP



Silhouette Score (−1 to +1)

Cohesion (a)   Separation (b)

$$S_i = \frac{b_i - a_i}{\max(a_i, b_i)}$$

$$\text{Score} = \frac{\sum S_i}{n}$$

AIML CLZG516
ML System Optimization

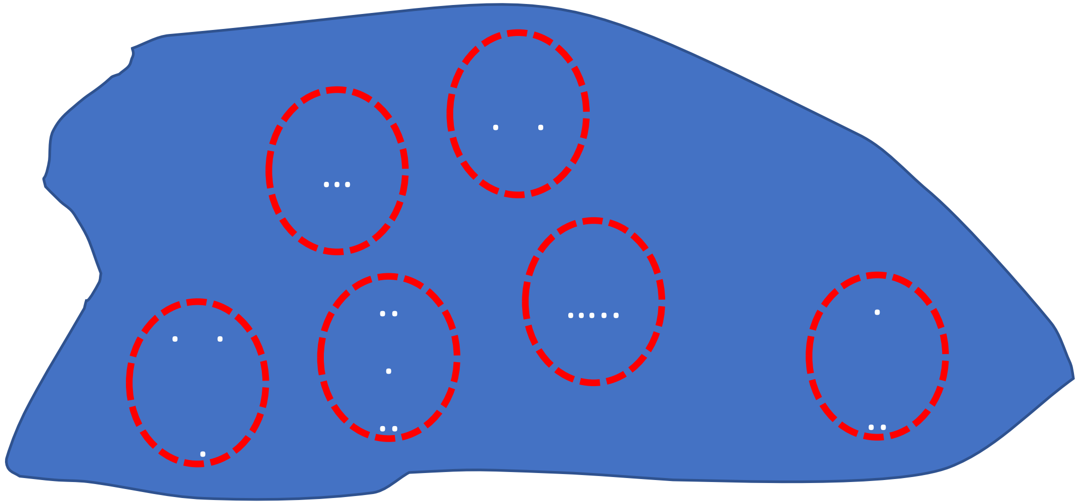
Parallelization of ML Algorithms

- Example: k-Means
- Issues with Large Data Size

Scale-out Clusters – Distributed Memory Programming

Example: Data Clustering using k-Means

- Data Clustering is a classic data analytics problem:
  - Given a set of data points group them into disjoint subsets – clusters – such that:
    - Each cluster is cohesive
    - Different clusters are well-separated



Points are in Euclidean space

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 ;
   whose 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)

1. MapReduce
2. Spark implementation

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 = (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 <u>using map and reduce.</u>
- [Hints:
  - <u>Step 1</u>: "select k representative points" for clusters (randomly)
  - <u>Step 2</u>
    - 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
  - <u>Step 3b</u>: compute the centroid (i.e., mean of) C_j
      - c_j = (reduce + C_j)/$|C_j|$

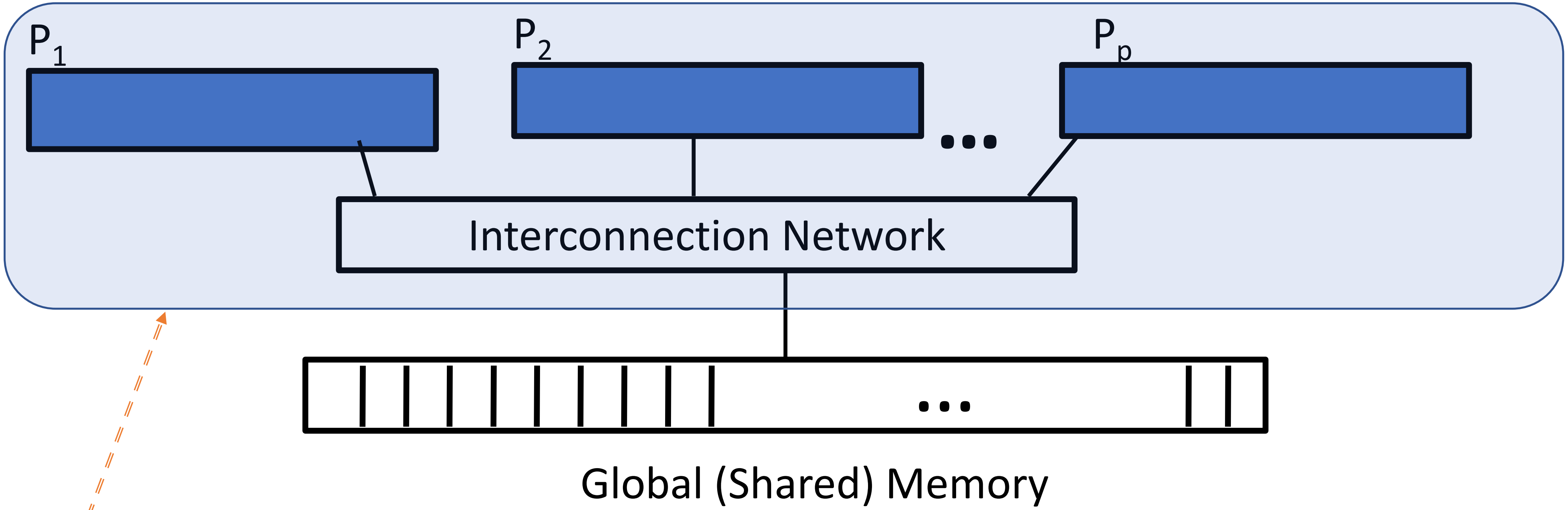
This follows a programming pattern named ***iterative map-reduce*** where map-reduce programming steps applied inside a loop.

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

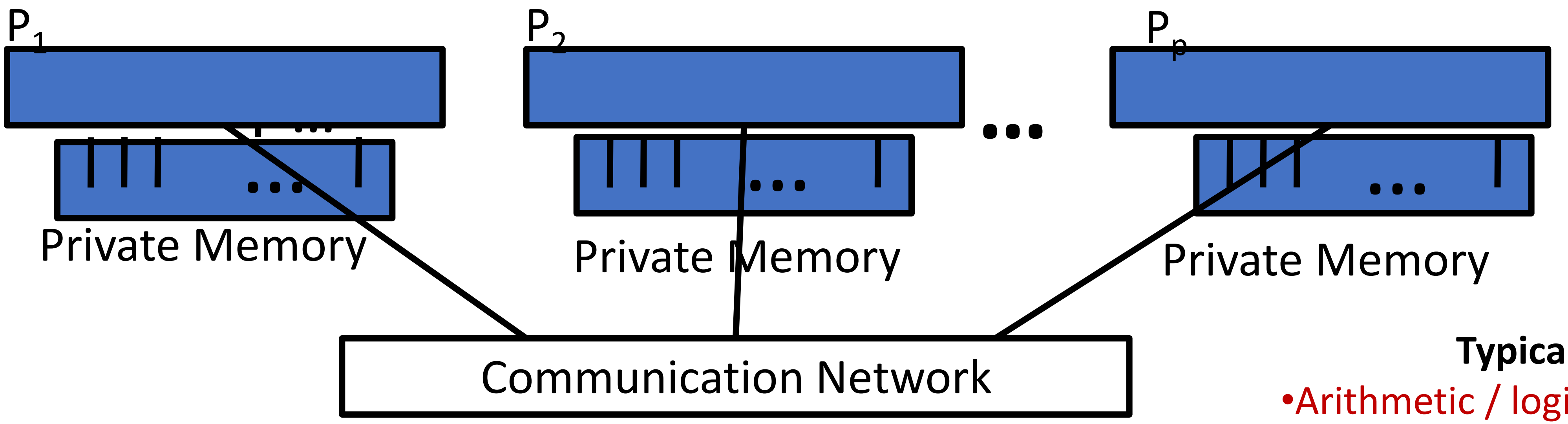
Multi-threaded Programming:
each thread runs on a separate core

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

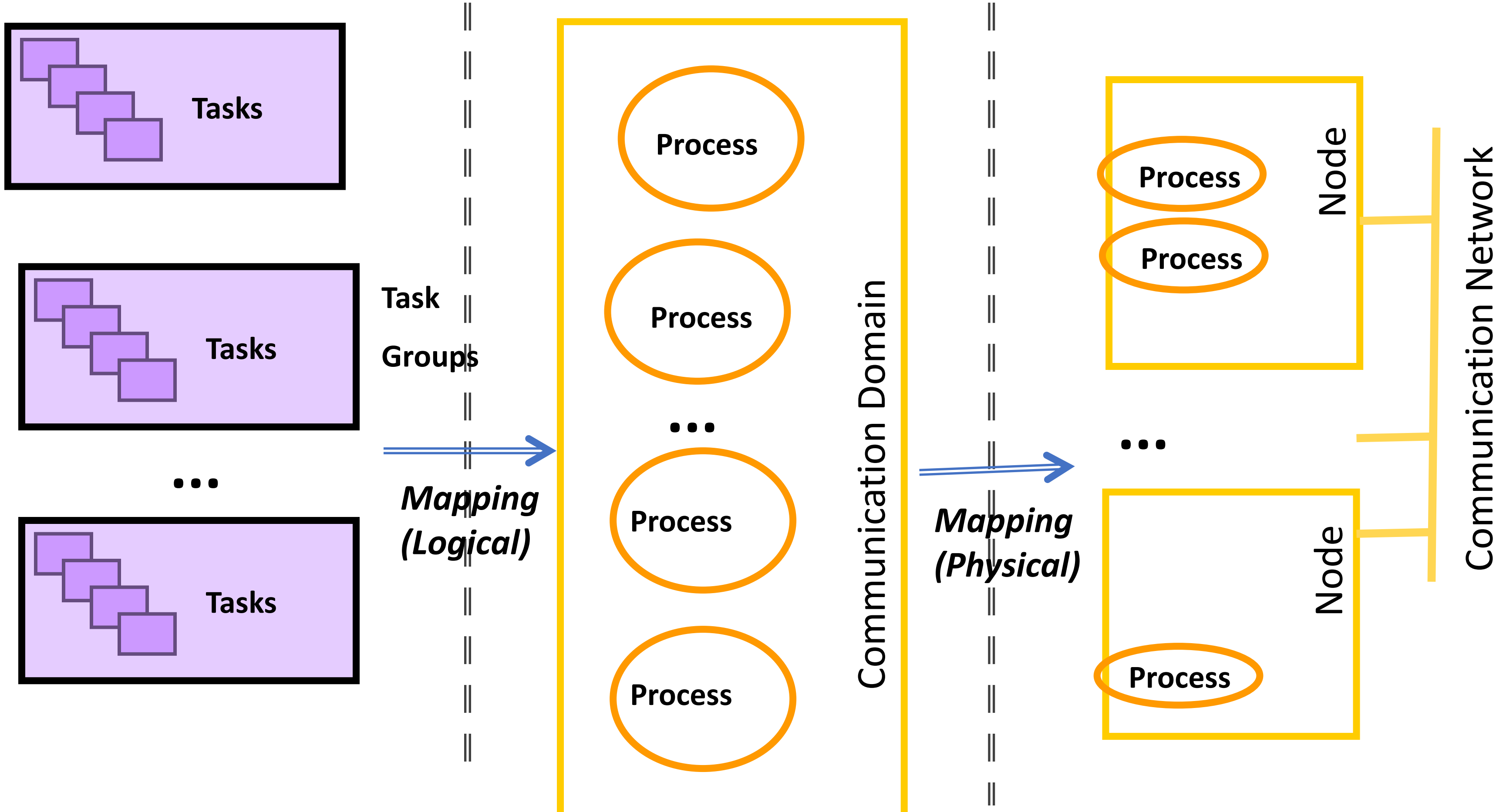
Parallel / Distributed Computing

- A parallel or distributed program is made of multiple tasks that *collaborate* (to achieve a common outcome).

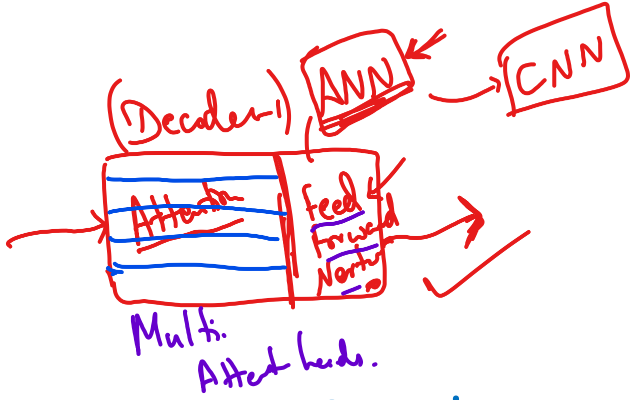
  - Collaboration is achieved by communication:
    - exchange data using shared memory
      - i.e. Task A writes to location L; Task B reads from location L
    - exchange data by passing messages
      - i.e. Task A sends a message to Task B; Task B receives the message from Task A

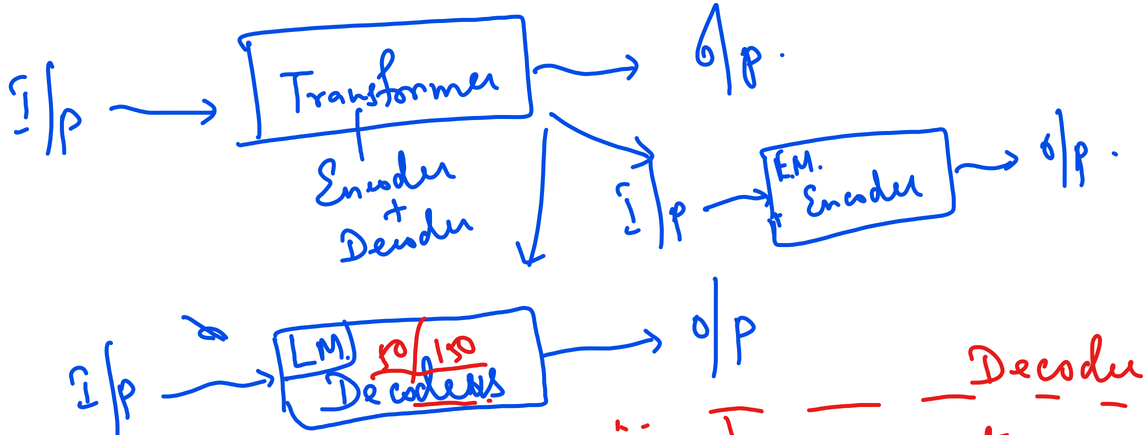
- Multiple processes each with its own address space:

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



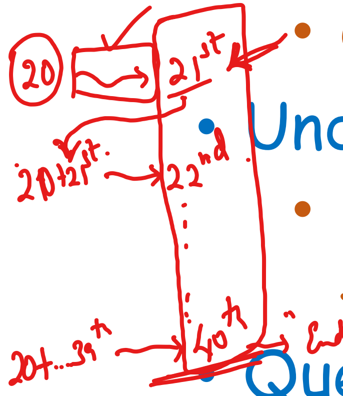
Exercise

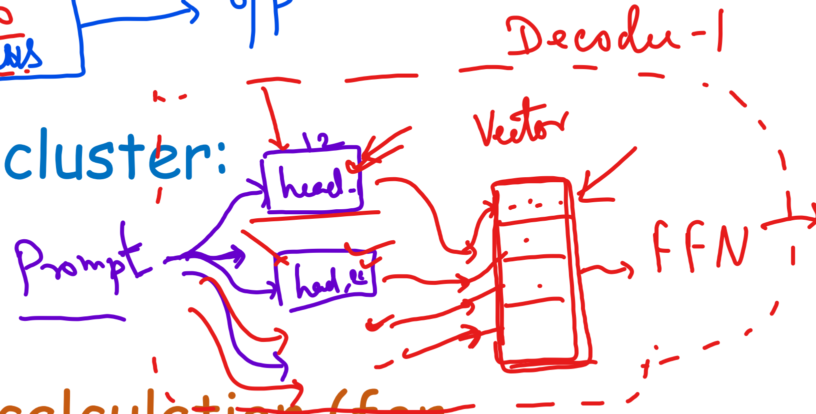




Streaming response

- 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?