PARALLEL & DISTRIBUTED DATABASES

CS561-SPRING 2012
WPI, MOHAMED ELTABAKH

INTRODUCTION

- **In centralized database:**
  - Data is located in one place (one server)
  - All DBMS functionalities are done by that server
    - Enforcing ACID properties of transactions
    - Concurrency control, recovery mechanisms
    - Answering queries

- **In Distributed databases:**
  - Data is stored in multiple places (each is running a DBMS)
  - New notion of distributed transactions
  - DBMS functionalities are now distributed over many machines
    - Revisit how these functionalities work in distributed environment

WHY DISTRIBUTED DATABASES

- Data is too large
- Applications are by nature distributed
  - Bank with many branches
  - Chain of retail stores with many locations
  - Library with many branches
- Get benefit of distributed and parallel processing
  - Faster response time for queries

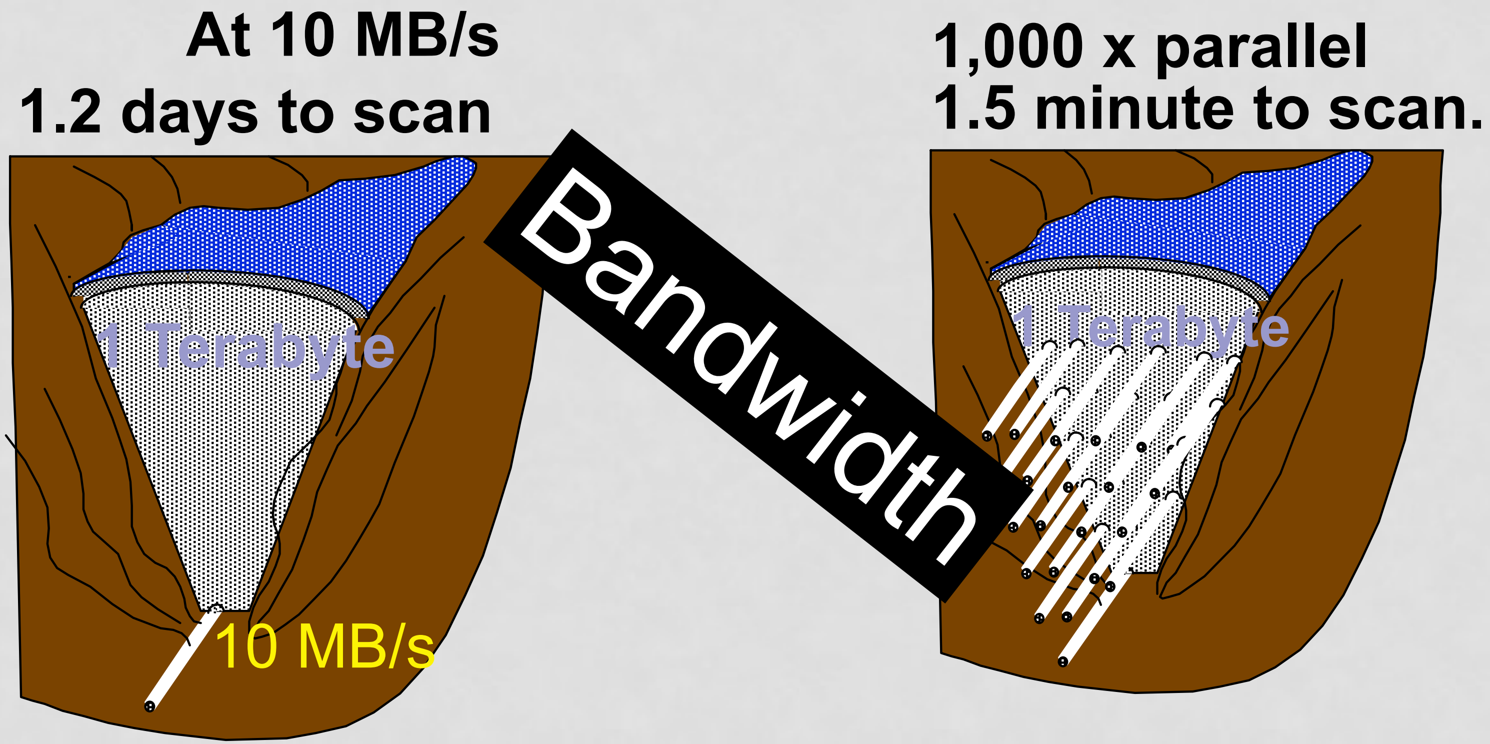
PARALLEL VS. DISTRIBUTED DATABASES

- Distributed processing usually imply parallel processing (not vise versa)
  - Can have parallel processing on a single machine

- **Assumptions about architecture**
  - **Parallel Databases**
    - Machines are physically close to each other, e.g., same server room
    - Machines connects with dedicated high-speed LANs and switches
    - Communication cost is assumed to be small
    - Can shared-memory, shared-disk, or shared-nothing architecture
  - **Distributed Databases**
    - Machines can far from each other, e.g., in different continent
    - Can be connected using public-purpose network, e.g., Internet
    - Communication cost and problems cannot be ignored
    - Usually shared-nothing architecture

PARALLEL DATABASE
&
PARALLEL PROCESSING

WHY PARALLEL PROCESSING

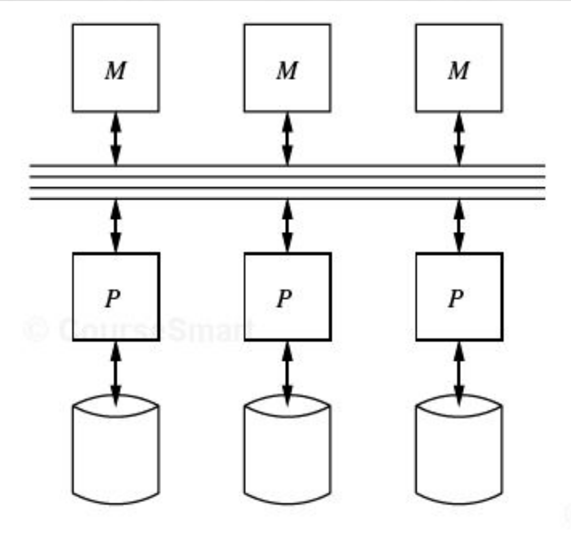


- Divide a big problem into many smaller ones to be solved in parallel
- Increase bandwidth (in our case decrease queries' response time)

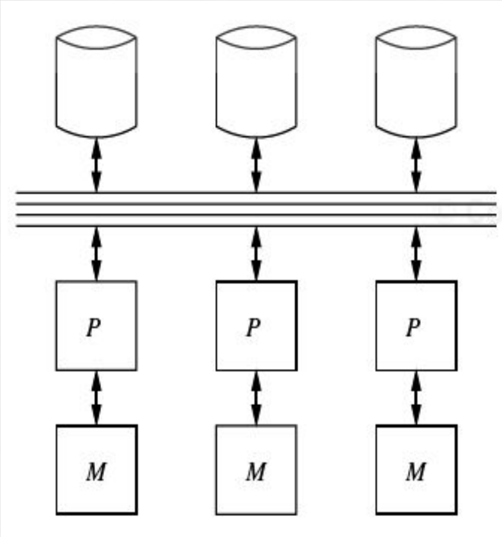
DIFFERENT ARCHITECTURE

- Three possible architectures for passing information

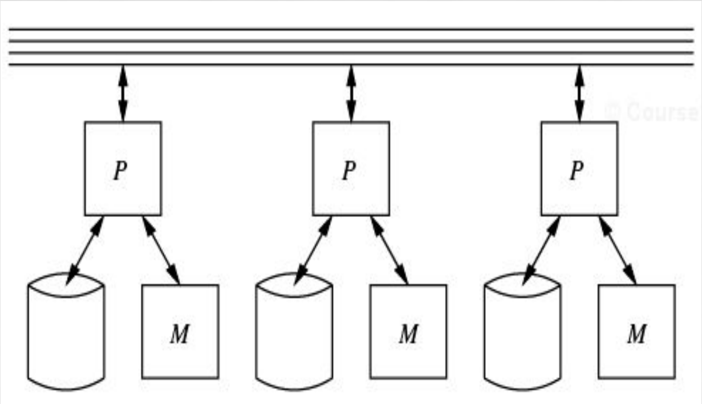
Shared-memory



Shared-disk

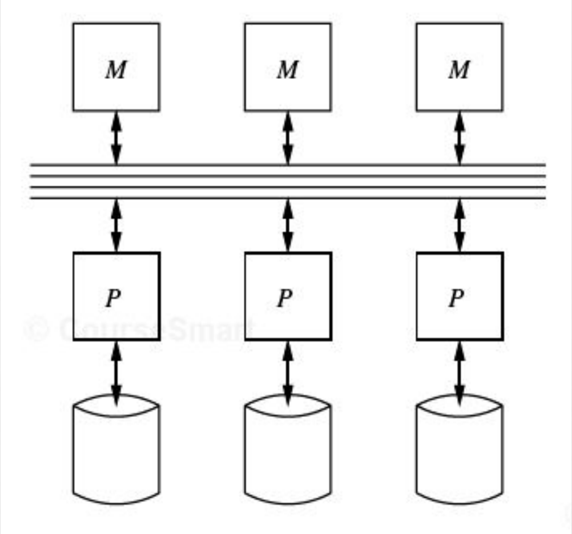


Shared-nothing



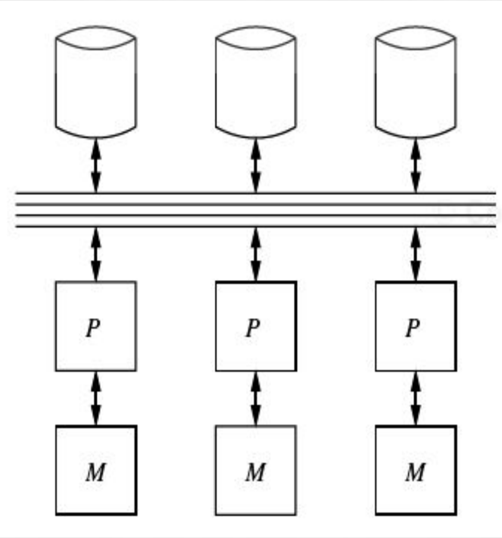
1- SHARED-MEMORY ARCHITECTURE

- Every processor has its own disk
- Single memory address-space for all processors
  - Reading or writing to far memory can be slightly more expensive
- Every processor can have its own local memory and cache as well



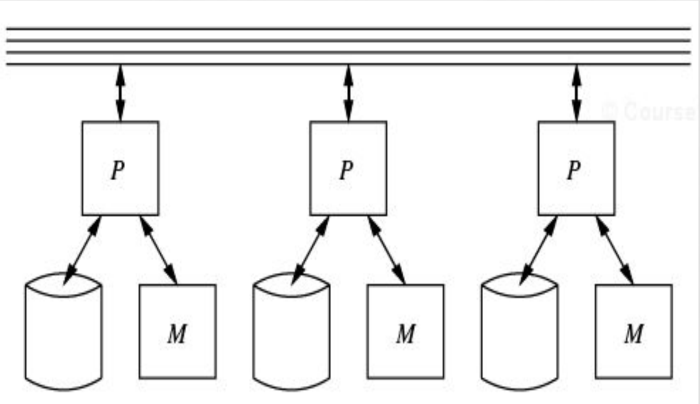
2- SHARED-DISK ARCHITECTURE

- Every processor has its own memory (not accessible by others)
- All machines can access all disks in the system
- Number of disks does not necessarily match the number of processors



3- SHARED-NOTHING ARCHITECTURE

- Most common architecture nowadays
- Every machine has its own memory and disk
  - Many cheap machines (commodity hardware)
- Communication is done through high-speed network and switches
- **Usually machines can have a hierarchy**
  - Machines on same rack
  - Then racks are connected through high-speed switches

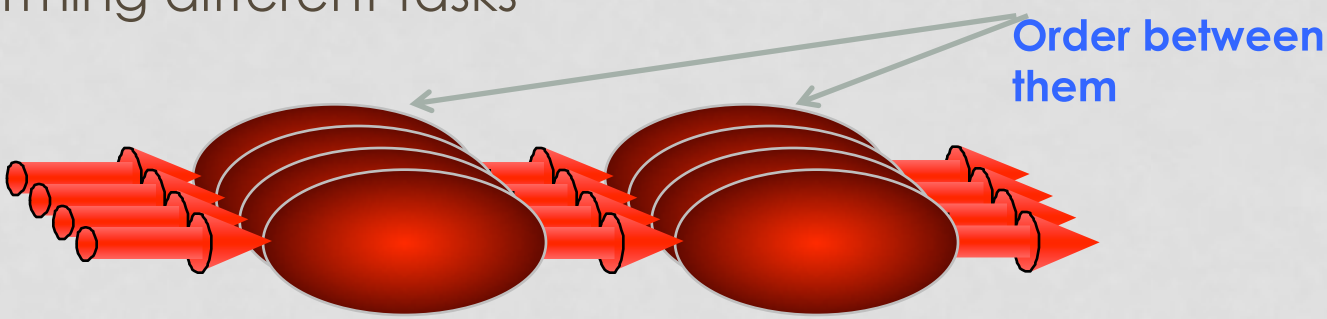


- **Scales better**
- **Easier to build**
- **Cheaper cost**

TYPES OF PARALLELISM

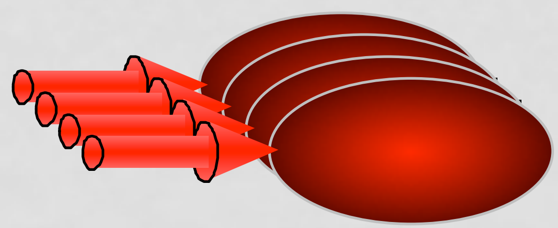
- **Pipeline Parallelism (Inter-operator parallelism)**
  - Ordered (or partially ordered) tasks and different machines are performing different tasks

Pipeline



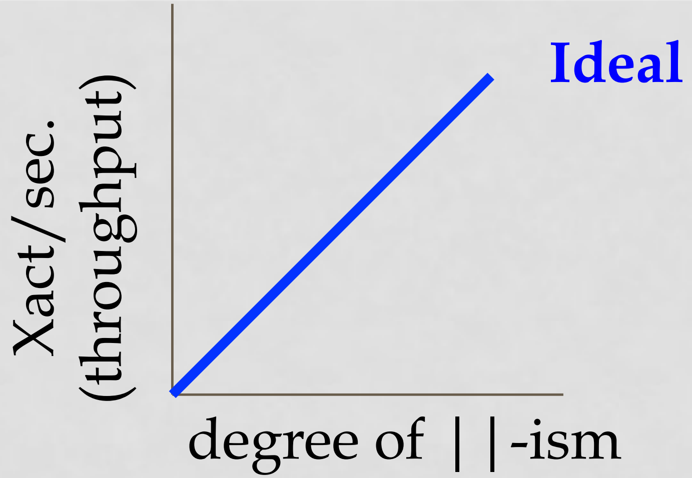
- **Partitioned Parallelism (Intra-operator parallelism)**
  - A task divided over all machines to run in parallel

Partition

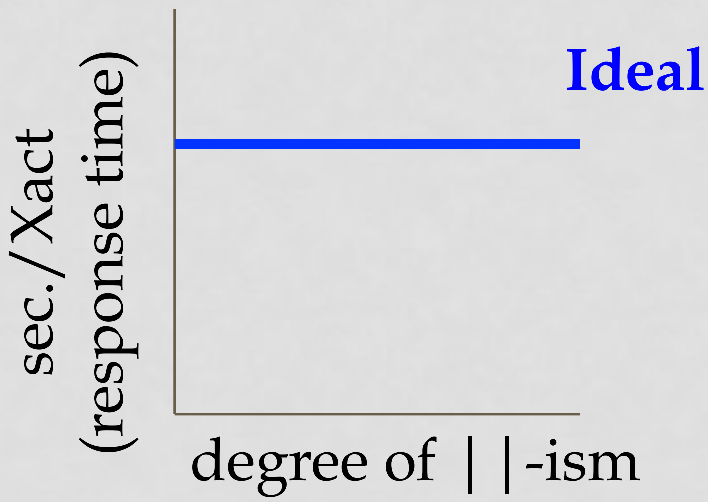


IDEAL SCALABILITY SCENARIO

- Speed-Up
  - More resources means proportionally less time for given amount of data.

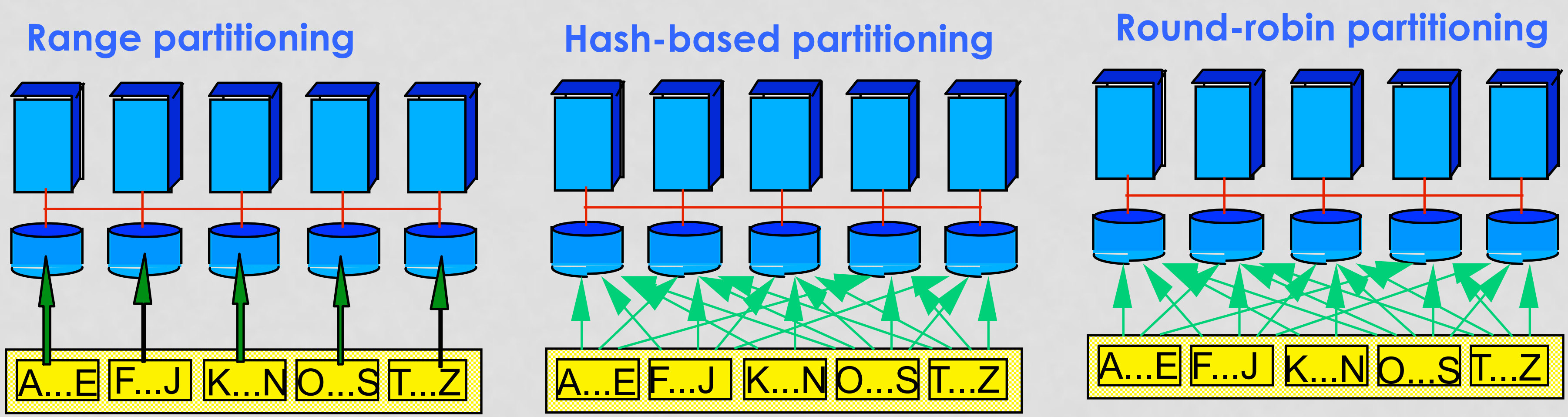


- Scale-Up
  - If resources increased in proportion to increase in data size, time is constant.



PARTITIONING OF DATA

To partition a relation R over m machines



- Shared-nothing architecture is sensitive to partitioning
- Good partitioning depends on what operations are common

PARALLEL ALGORITHMS FOR DBMS OPERATIONS

PARALLEL SCAN $\sigma_c(R)$

- Relation R is partitioned over ***m*** machines
  - Each partition of R is around |R|/m tuples

- Each machine scans its own partition and applies the selection condition c

- **If data are partitioned using round robin or a hash function (over the entire tuple)**
  - The resulted relation is expected to be well distributed over all nodes
  - All partitioned will be scanned

- **If data are range partitioned or hash-based partitioned (on the selection column)**
  - The resulted relation can be clustered on few nodes
  - Few partitions need to be touched

> - Parallel Projection is also straightforward
> - All partitions will be touched
> - Not sensitive to how data is partitioned

PARALLEL DUPLICATE ELIMINATION

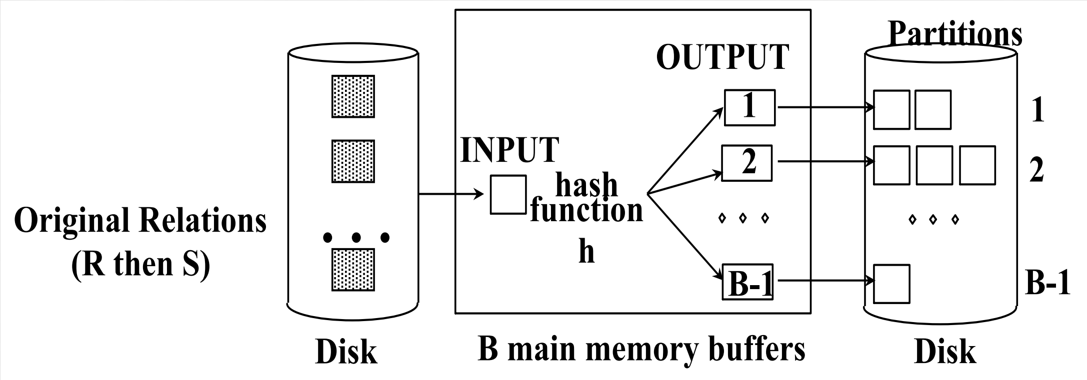
- **If relation is range or hash-based partitioned**
  - Identical tuples are in the same partition
  - So, eliminate duplicates in each partition independently

- **If relation is round-robin partitioned**
  - Re-partition the relation using a hash function
  - So every machine creates m partitions and send the i^{th} partition to machine i
  - machine i can now perform the duplicate elimination

> - Same idea applies to Set Operations (Union, Intersect, Except)
> - But apply the same partitioning to both relations R & S

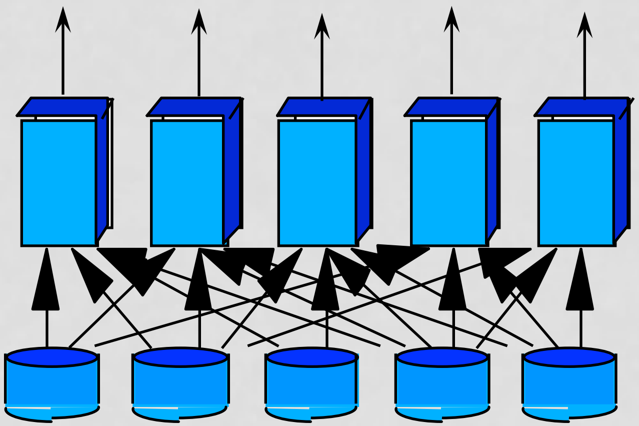
PARALLEL JOIN R(X,Y) ⋈ S(Y,Z)

- **Re-partition R and S on the join attribute Y (natural join) or (equi join)**
  - Hash-based or range-based partitioning
- **Each machine i receives all i[th] partitions from all machines (from R and S)**
  - Each machine can locally join the partitions it has
- Depending on the partitions sizes of R and S, local joins can be hash-based or merge-join



PARALLEL SORTING

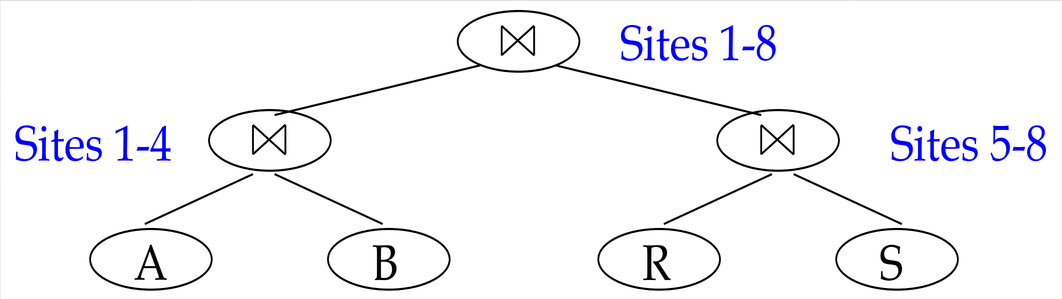
- **Range-based**
  - Re-partition R based on ranges into m partitions
  - Machine i receives all i^{th} partitions from all machines and sort that partition
  - The entire R is now sorted
  - **Skewed data is an issue**
    - Apply sampling phase first
    - Ranges can be of different width



- **Merge-based**
  - Each node sorts its own data
  - All nodes start sending their sorted data (one block at a time) to a single machine
  - This machine applies merge-sort technique as data come

COMPLEX PARALLEL QUERY PLANS

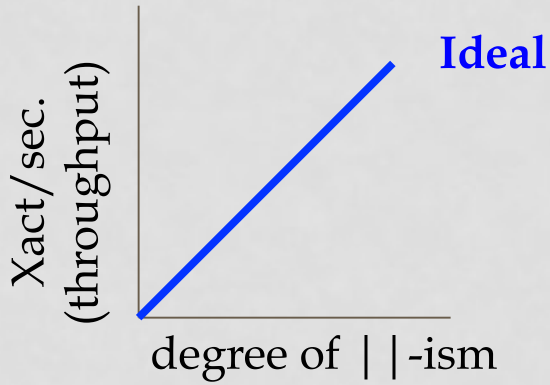
- All previous examples are ***intra-operator parallelism***
- **Complex queries can have *inter-operator parallelism***
  - Different machines perform different tasks

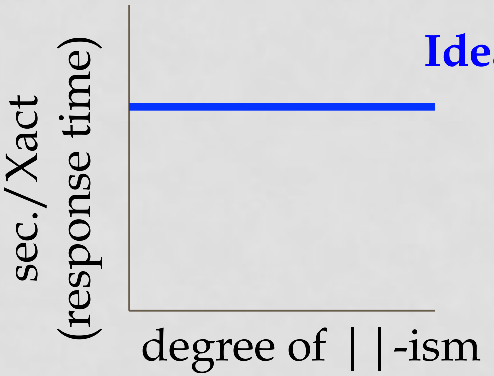


PERFORMANCE OF PARALLEL ALGORITHMS

- **In many cases, parallel algorithms reach their expected lower bound (or close to)**
  - If parallelism degree is m, then the parallel cost is 1/m of the sequential cost
  - Cost mostly refers to query's response time

- **Example**
  - Parallel selection or projection is 1/m of the sequential cost





PERFORMANCE OF PARALLEL ALGORITHMS (CONT'D)

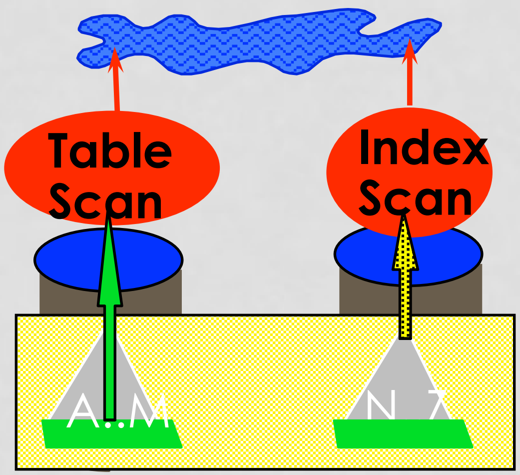
- **Total disk I/Os (sum over all machines) of parallel algorithms can be larger than that of sequential counterpart**
  - But we get the benefit of being done in parallel

- **Example**
  - Merge-sort join **(serial case)** has I/O cost = 3(B(R) + B(S))
  - Merge-sort join **(parallel case)** has total (sum) I/O cost = 5(B(R) + B(S))
    - **Considering the parallelism = 5(B(R) + B(S)) / m**

Number of pages of relations R and S

OPTIMIZING PARALLEL ALGORITHMS

- **Best serial plan != the best parallel one**



- **Trivial counter-example:**
  - Table partitioned with local secondary index at two nodes
  - **Range query:** all data of node 1 and 1% of node 2.
  - Node 1 should do a scan of its partition.
  - Node 2 should use secondary index.

- Different optimization algorithms for parallel plans (more candidate plans)
- Different machines may perform the same operation but using different plans

SUMMARY OF PARALLEL DATABASES

- **Three possible architectures**
  - Shared-memory
  - Shared-disk
  - Shared-nothing (the most common one)

- **Parallel algorithms**
  - Intra-operator
    - Scans, projections, joins, sorting, set operators, etc.
  - Inter-operator
    - Distributing different operators in a complex query to different nodes

- **Partitioning and data layout is important and affect the performance**

- **Optimization of parallel algorithms is a challenge**