CSE 344

APRIL 30TH – SCHEDULING / PARALLEL

DISK SCHEDULING

- **Query optimization**
  - Good DB design
  - Good estimation
  - Hardware independent
- **All Disk I/Os are not created equal**
  - Sectors close to each other are more preferable to read

DISK SCHEDULING

- **Disk I/O behavior**
  - Very rare to have requests come in one at a time
  - Requests come in batches, i.e. read the whole file
- **How does the hardware process a batch?**

DISK SCHEDULING

- **Suppose sectors are ordered from the outside to the inside of the disk**
  - Given a collection of sectors, how do we read them with the smallest amount of head movement?

DISK SCHEDULING

- **What are some strategies for processing the following batch?**
  - 95, 180, 34, 119, 11, 123, 62, 64
  - Assume sectors are numbered from 0-199 and that we start at sector 50

DISK SCHEDULING

- **What are some strategies for processing the following batch?**
  - 95, 180, 34, 119, 11, 123, 62, 64
  - Assume sectors are numbered from 0-199 and that we start at sector 50
  - Ideas?

DISK SCHEDULING

- **95, 180, 34, 119, 11, 123, 62, 64**

DISK SCHEDULING

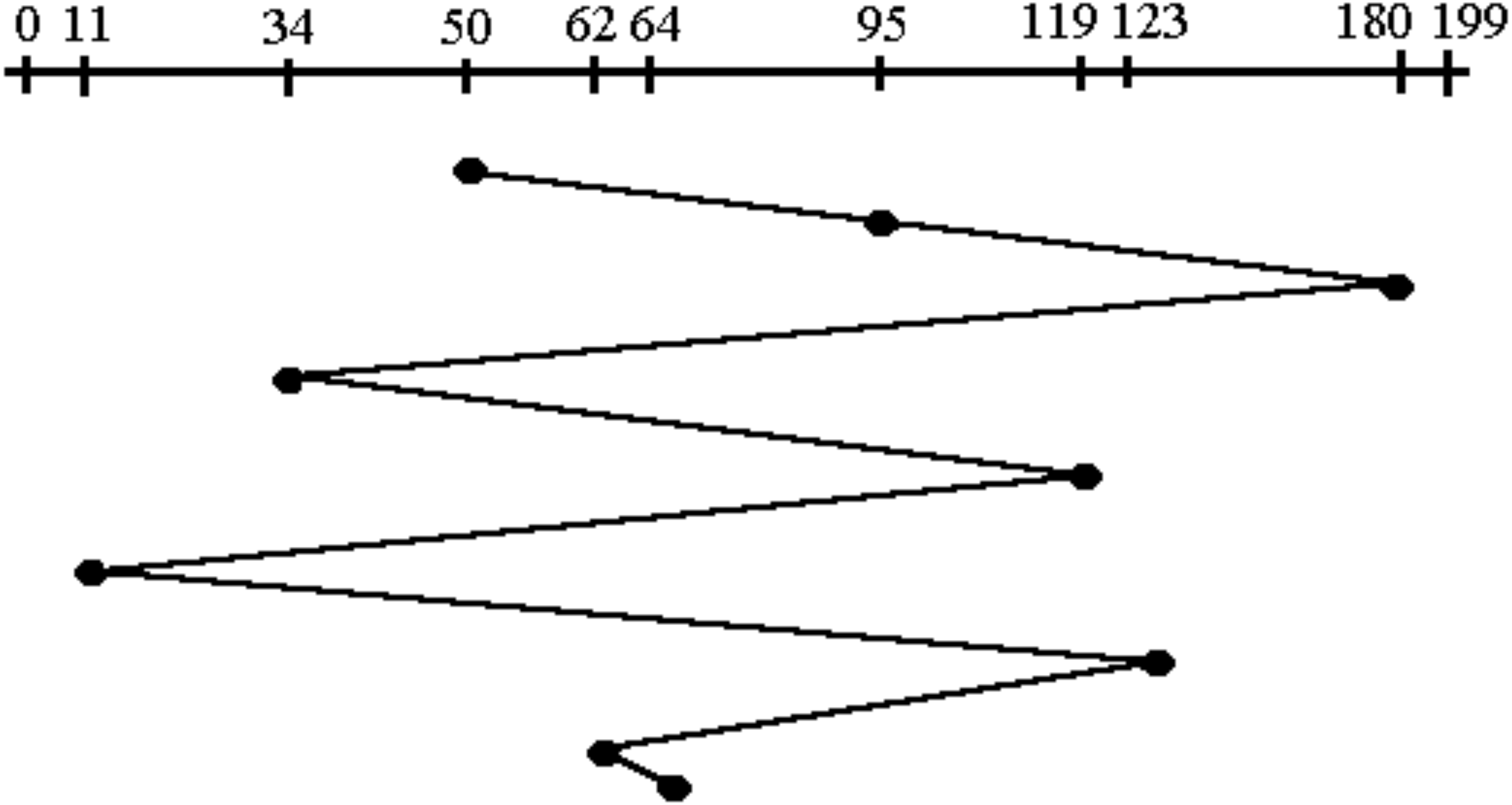
- **95, 180, 34, 119, 11, 123, 62, 64**
  - FIFO. Naive solution
    - Pros/cons?

DISK SCHEDULING

- **95, 180, 34, 119, 11, 123, 62, 64**
  - FIFO. Naive solution
    - Pros/cons?
      - + easy to add new sectors to the queue
      - + almost no computation to maintain
      - - non-optimal, easy to create adversarial batches – doesn‘t really take advantage of batches
      - 640 tracks

DISK SCHEDULING

- **95, 180, 34, 119, 11, 123, 62, 64**



DISK SCHEDULING

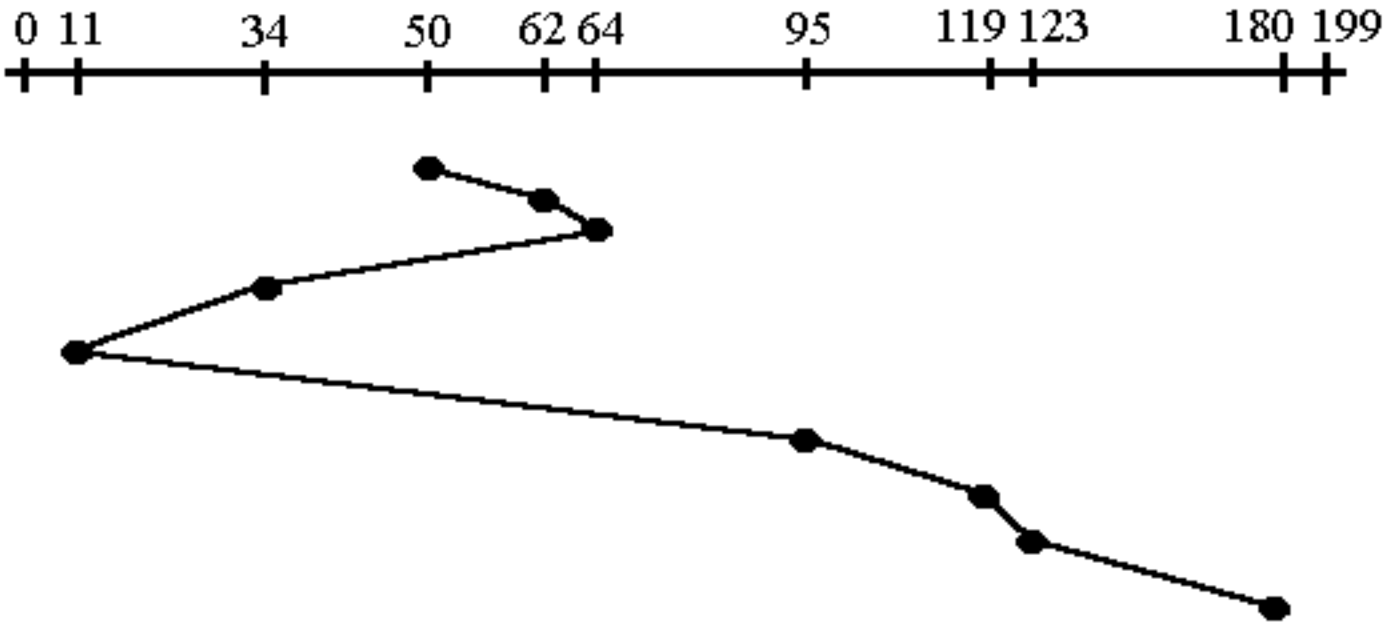
- **95, 180, 34, 119, 11, 123, 62, 64**
  - Get closest (Shortest seek time first)
  - Pros/cons?

DISK SCHEDULING

- **95, 180, 34, 119, 11, 123, 62, 64**
  - Get closest (Shortest seek time first)
  - Pros/cons?
    - + efficient (236)
    - - costly to maintain
    - - starvation

DISK SCHEDULING

- **95, 180, 34, 119, 11, 123, 62, 64**



DISK SCHEDULING

- **95, 180, 34, 119, 11, 123, 62, 64**
  - Sorting:
    - 50,11,34,62,64,95,119,123,180
    - Pros/cons?

DISK SCHEDULING

- **95, 180, 34, 119, 11, 123, 62, 64**
  - Sorting:
    - 50,11,34,62,64,95,119,123,180
    - Pros/cons?
      - + fewer track movements (208)
      - - costly to maintain, add new
      - - doesn’t account for start position
      - + no starvation

DISK SCHEDULING

- **95, 180, 34, 119, 11, 123, 62, 64**
  - How do we modify the ”sorting” algorithm to better take advantage of the start position?

DISK SCHEDULING

- **95, 180, 34, 119, 11, 123, 62, 64**
  - How do we modify the ”sorting” algorithm to better take advantage of the start position?
  - *How does an elevator schedule rides?*

DISK SCHEDULING

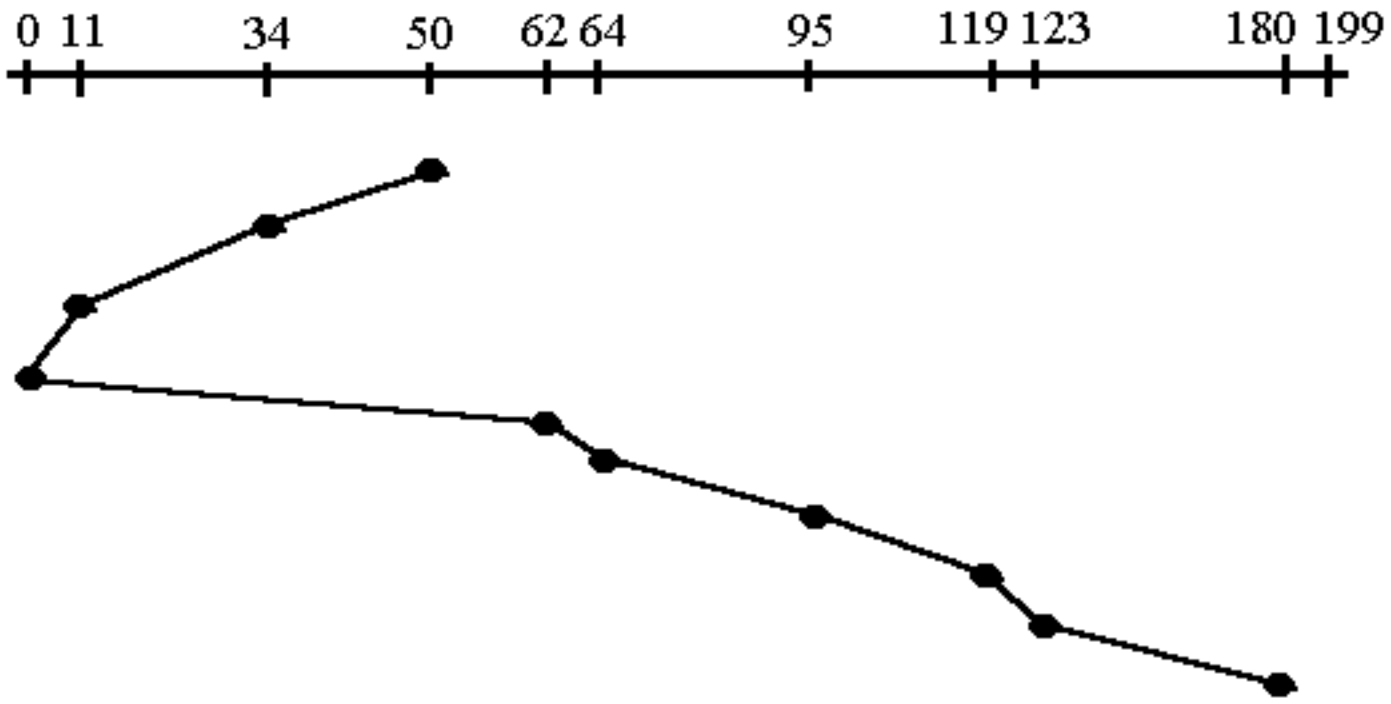
- **95, 180, 34, 119, 11, 123, 62, 64**
  - How do we modify the ”sorting” algorithm to better take advantage of the start position?
  - *How does an elevator schedule rides?*
    - *Start in a position, go in one direction until you reach the end, repeat going the other way*

DISK SCHEDULING

- **95, 180, 34, 119, 11, 123, 62, 64**
  - Elevator algorithm (SCAN)
    - Pros/cons?

DISK SCHEDULING

- **95, 180, 34, 119, 11, 123, 62, 64**
  - Elevator algorithm (SCAN)
    - Pros/cons?



DISK SCHEDULING

- **95, 180, 34, 119, 11, 123, 62, 64**
  - Elevator algorithm (SCAN)
    - Pros/cons?
      - + no starvation
      - - some maintenance
      - + efficient (230)

DISK SCHEDULING

- **Weird fact about disks**
  - Moving the arm *accurately* takes longer than moving it large numbers of tracks
  - Why might this matter?

DISK SCHEDULING

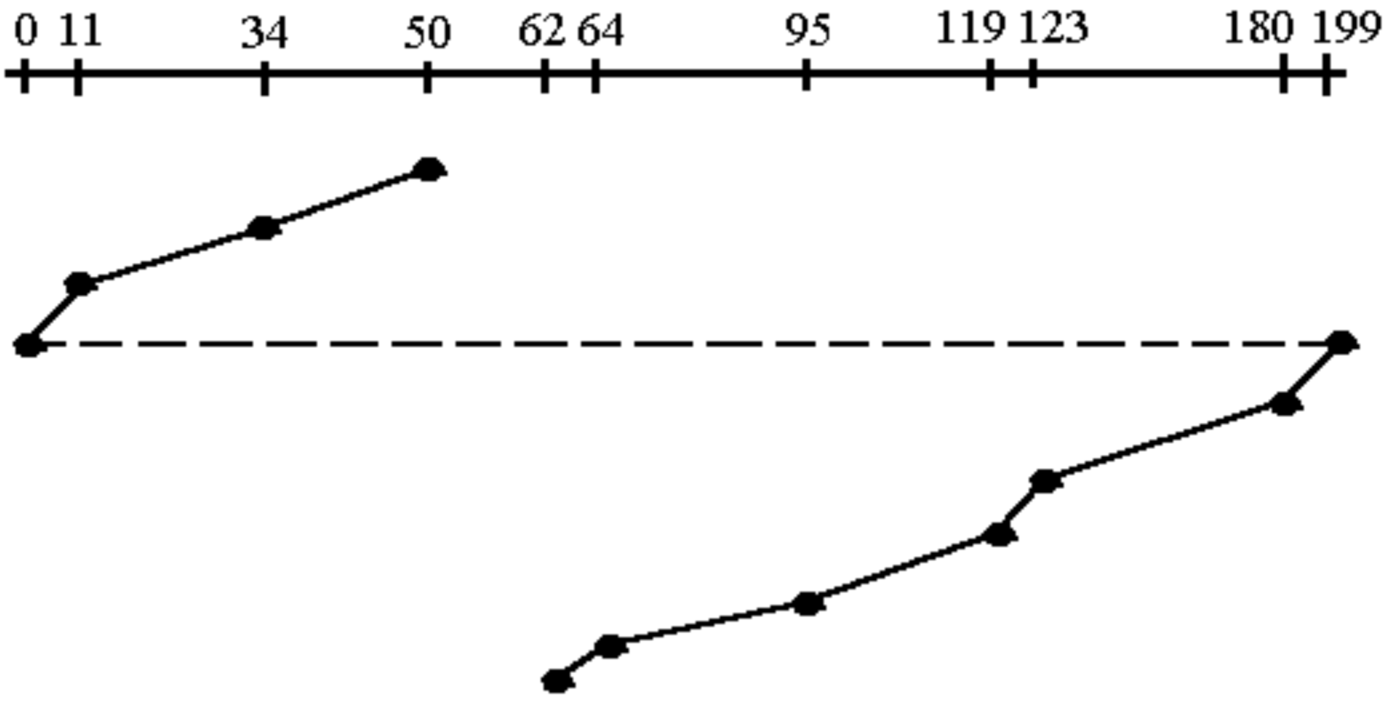
- **Weird fact about disks**
  - Moving the arm *accurately* takes longer than moving it large numbers of tracks
  - Why might this matter?
  - SCAN in only one direction then quickly move the arm back to the beginning (quicker than standard find)
  - C-SCAN

DISK SCHEDULING

- **95, 180, 34, 119, 11, 123, 62, 64**
  - Elevator algorithm (C-SCAN)
    - Pros/cons?
      - + no starvation
      - - some maintenance
      - + efficient (187 + large movement)
      - ~ goes from 0-199

DISK SCHEDULING

- **95, 180, 34, 119, 11, 123, 62, 64**
  - Elevator algorithm (C-SCAN)



DISK SCHEDULING

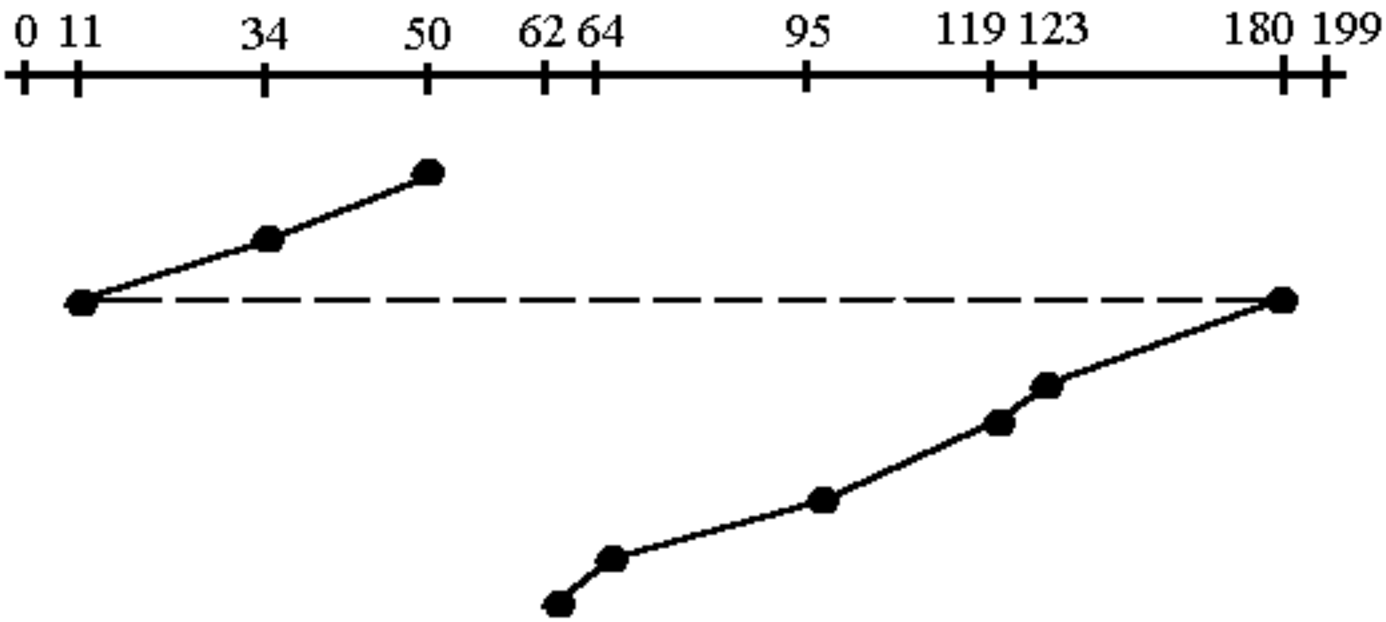
- **95, 180, 34, 119, 11, 123, 62, 64**
  - What if we don’t insist on going all the way to the ends?
    - \- need “accurate” arm movement
    - \+ can save some articulation
    - \- might delay reads from inner/outer sectors

DISK SCHEDULING

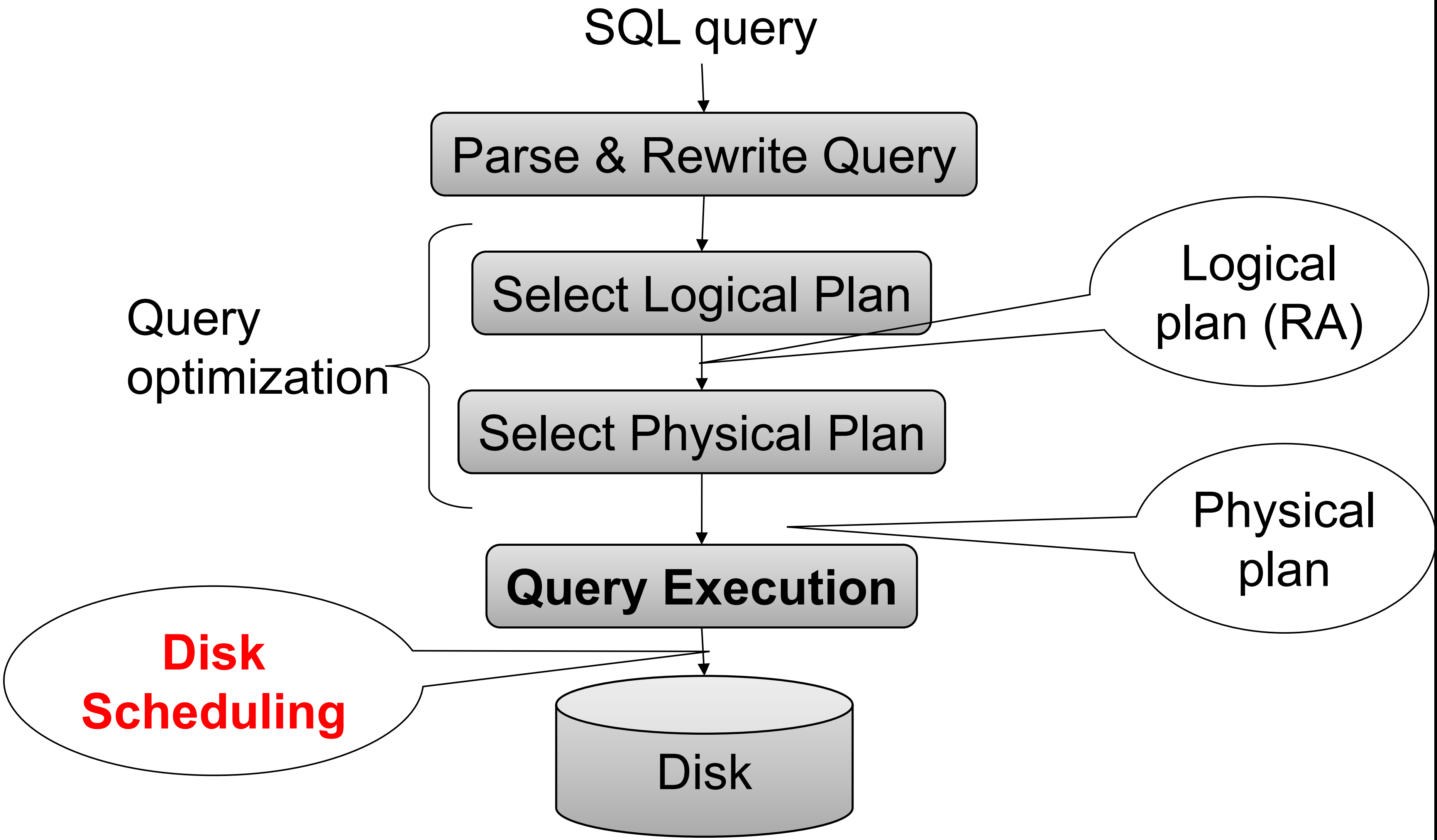
- **95, 180, 34, 119, 11, 123, 62, 64**
  - What if we don't insist on going all the way to the ends? (C-LOOK)
    - \- need "accurate" arm movement
    - \+ can save some articulation (157 + large)
    - \- might delay reads from inner/outer sectors

DISK SCHEDULING

- **95, 180, 34, 119, 11, 123, 62, 64**
  - C-LOOK (circular look)



QUERY EVALUATION STEPS



QUERY EVALUATION

- **Design**
  - Query
  - DBMS
  - Hardware
- **Single machine optimization**
  - Hardware scaleup

WHY COMPUTE IN PARALLEL?

Multi-cores:

- Most processors have multiple cores
- This trend will likely increase in the future

Big data: too large to fit in main memory

- Distributed query processing on 100x-1000x servers
- Widely available now using cloud services
- Recall HW3 and HW6

PERFORMANCE METRICS FOR PARALLEL DBMSS

Nodes = processors, computers

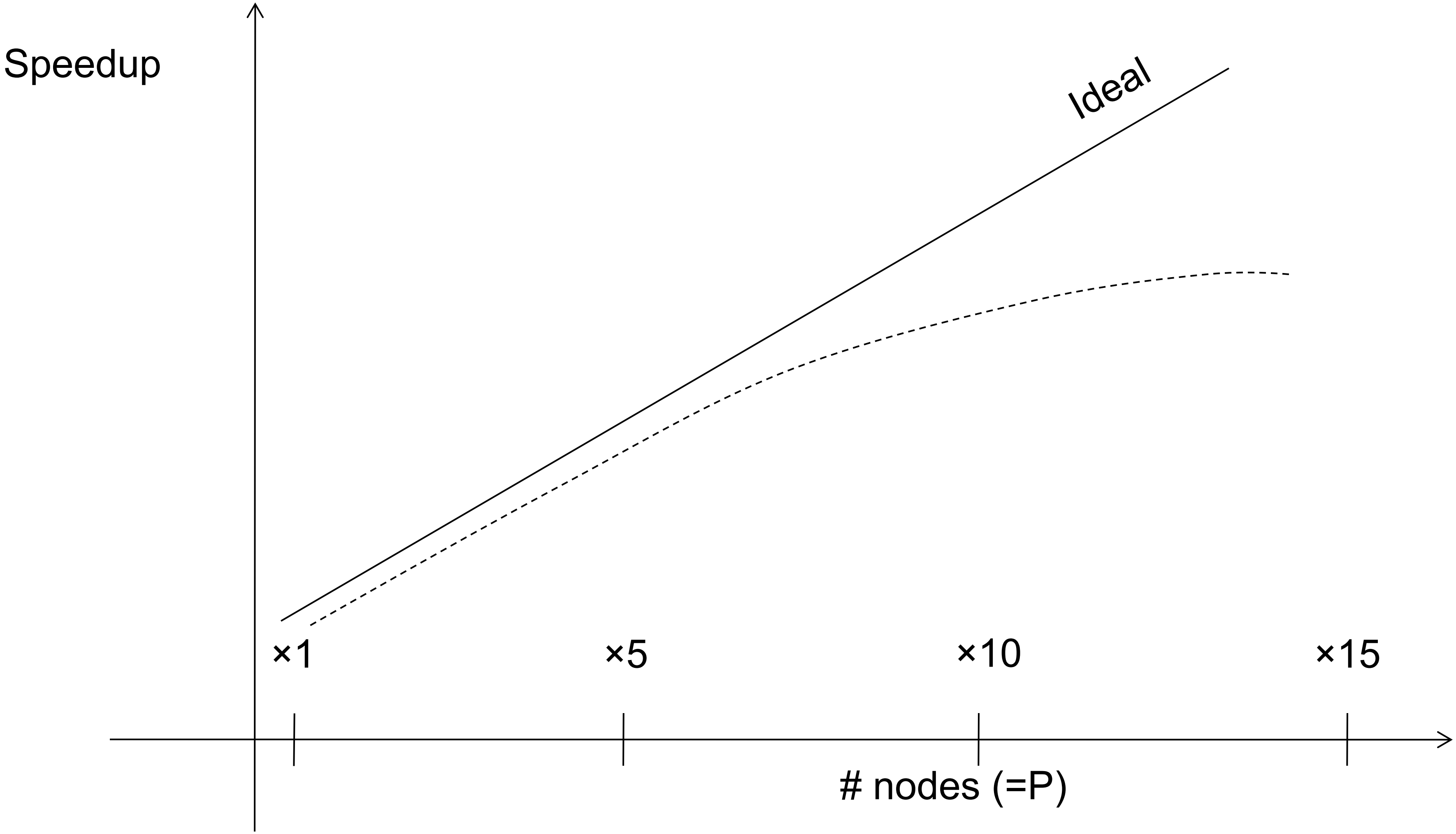
Speedup:

- More nodes, same data ➔ higher speed

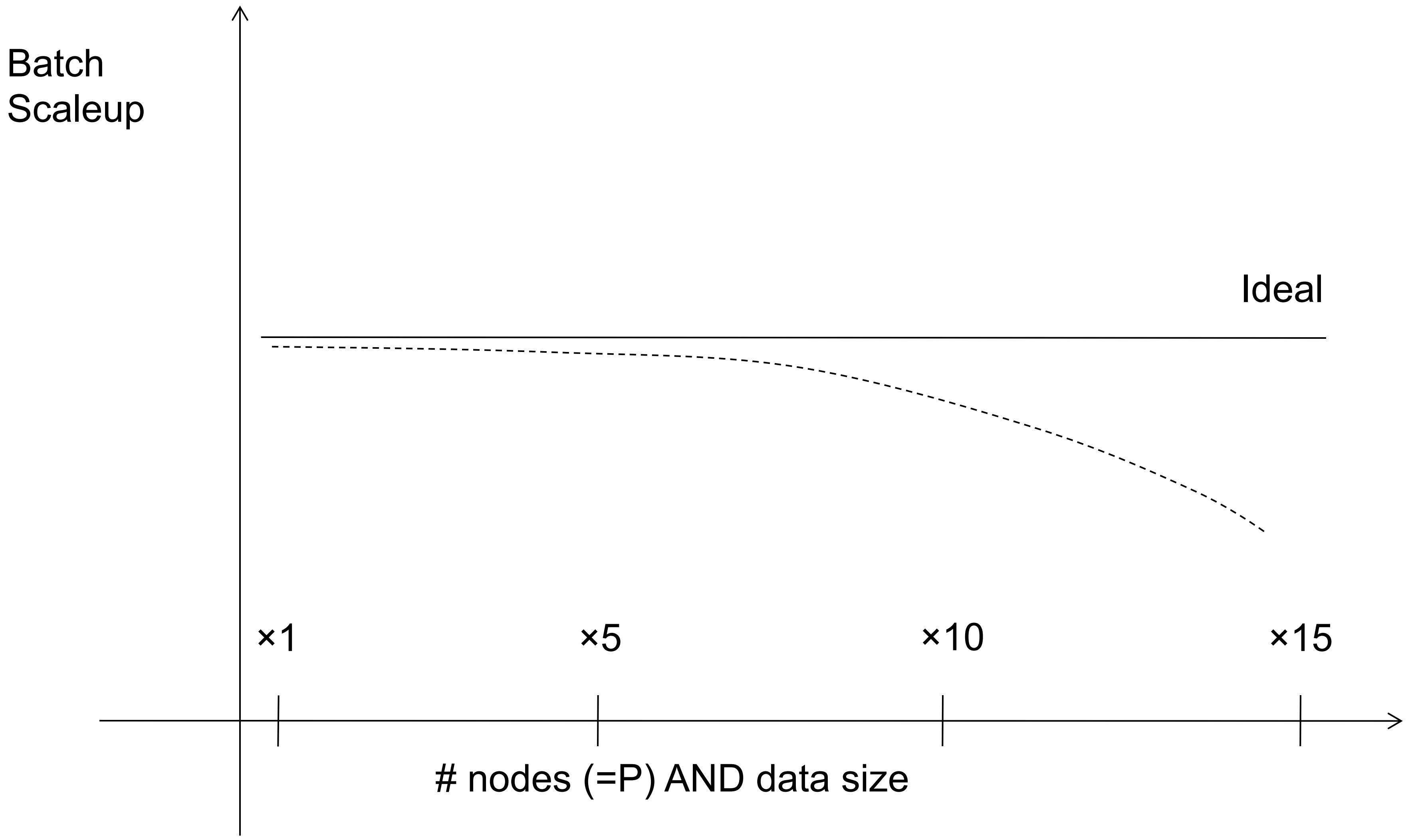
Scaleup:

- More nodes, more data ➔ same speed

LINEAR V.S. NON-LINEAR SPEEDUP



LINEAR V.S. NON-LINEAR SCALEUP



WHY SUB-LINEAR SPEEDUP AND SCALEUP?

Startup cost

- Cost of starting an operation on many nodes

Interference

- Contention for resources between nodes

Skew

- Slowest node becomes the bottleneck

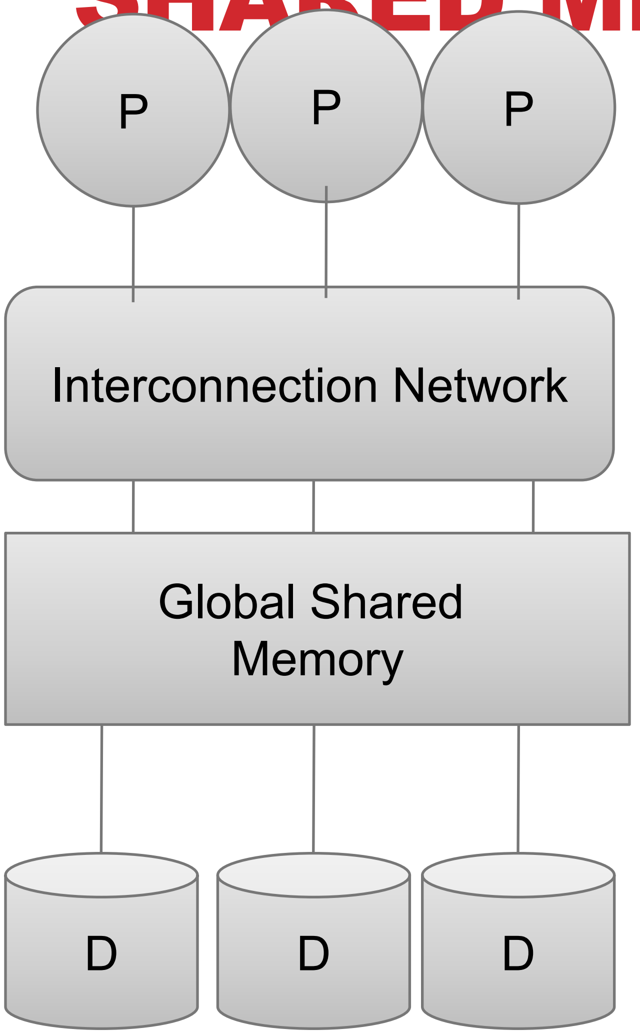
ARCHITECTURES FOR PARALLEL DATABASES

Shared memory

Shared disk

Shared nothing

SHARED MEMORY



Nodes share both RAM and disk

Dozens to hundreds of processors

Example: SQL Server runs on a single machine and can leverage many threads to speed up a query

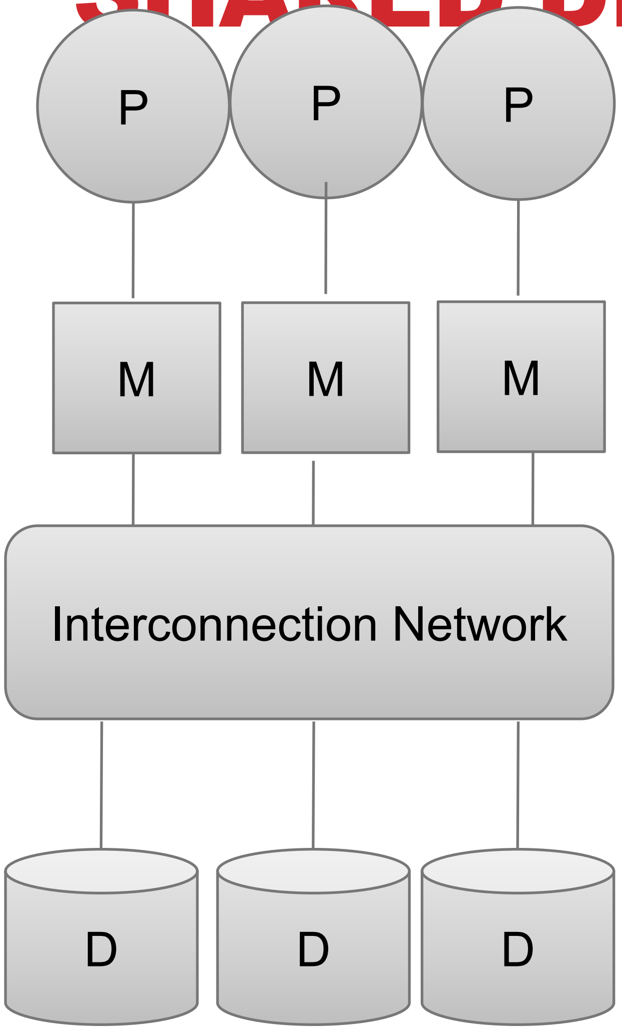
check your HW3 query plans

Easy to use and program

Expensive to scale

- last remaining cash cows in the hardware industry

SHARED DISK



All nodes access the same disks

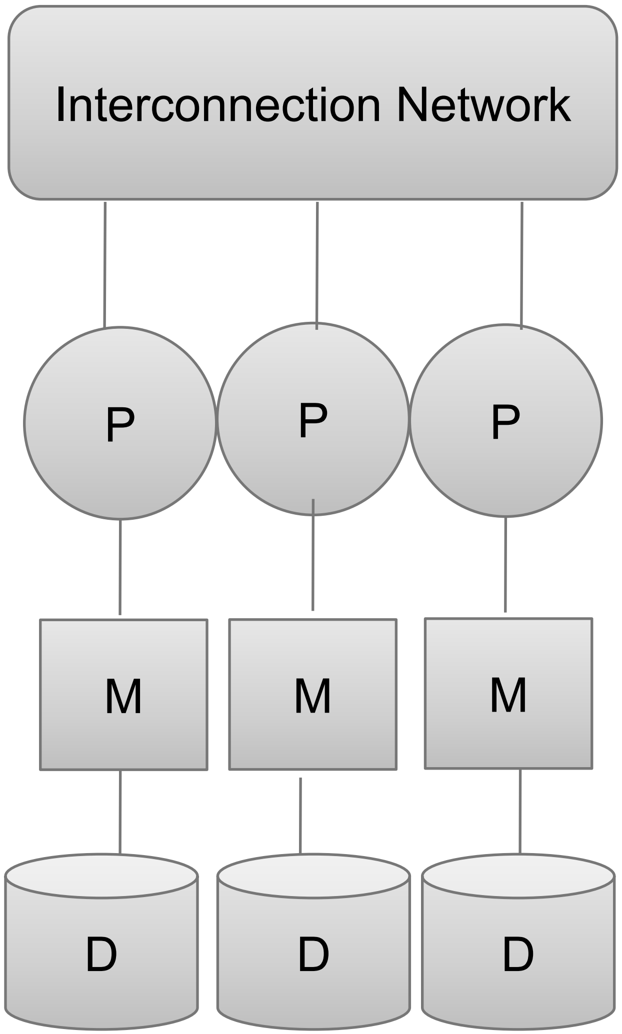
Found in the largest "single-box" (non-cluster) multiprocessors

Example: Oracle

No need to worry about shared memory

Hard to scale: existing deployments typically have fewer than 10 machines

SHARED NOTHING



Cluster of commodity machines on high-speed network

Called "clusters" or "blade servers”

Each machine has its own memory and disk: lowest contention.

Example: Google

Because all machines today have many cores and many disks, shared-nothing systems typically run many "nodes” on a single physical machine.

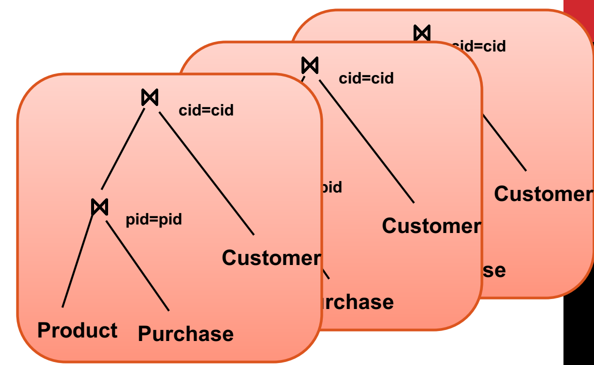
We discuss only Shared Nothing in class

Most difficult to administer and tune.

APPROACHES TO PARALLEL QUERY EVALUATION

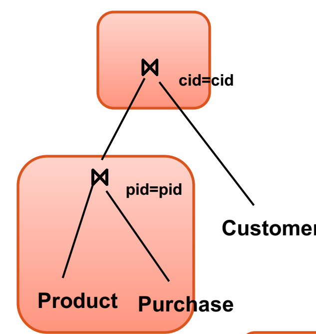
Inter-query parallelism

- Transaction per node
- Good for transactional workloads



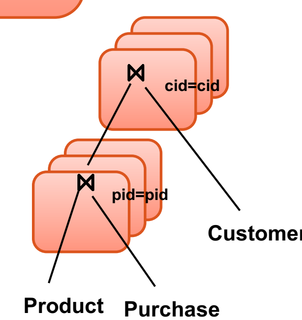
Inter-operator parallelism

- Operator per node
- Good for analytical workloads



Intra-operator parallelism

- Operator on multiple nodes
- Good for both?



We study only intra-operator parallelism: most scalable

DISTRIBUTED QUERY PROCESSING

Data is horizontally partitioned on many servers

Operators may require data reshuffling

First let's discuss how to distribute data across multiple nodes / servers

SINGLE NODE QUERY PROCESSING (REVIEW)

Given relations R(A,B) and S(B, C), no indexes:

Selection: $\sigma_{A=123}(R)$

- Scan file R, select records with A=123

Group-by: $\gamma_{A,sum(B)}(R)$

- Scan file R, insert into a hash table using A as key
- When a new key is equal to an existing one, add B to the value

Join: $R \bowtie S$

- Scan file S, insert into a hash table using B as key
- Scan file R, probe the hash table using B

HORIZONTAL DATA PARTITIONING

Data:

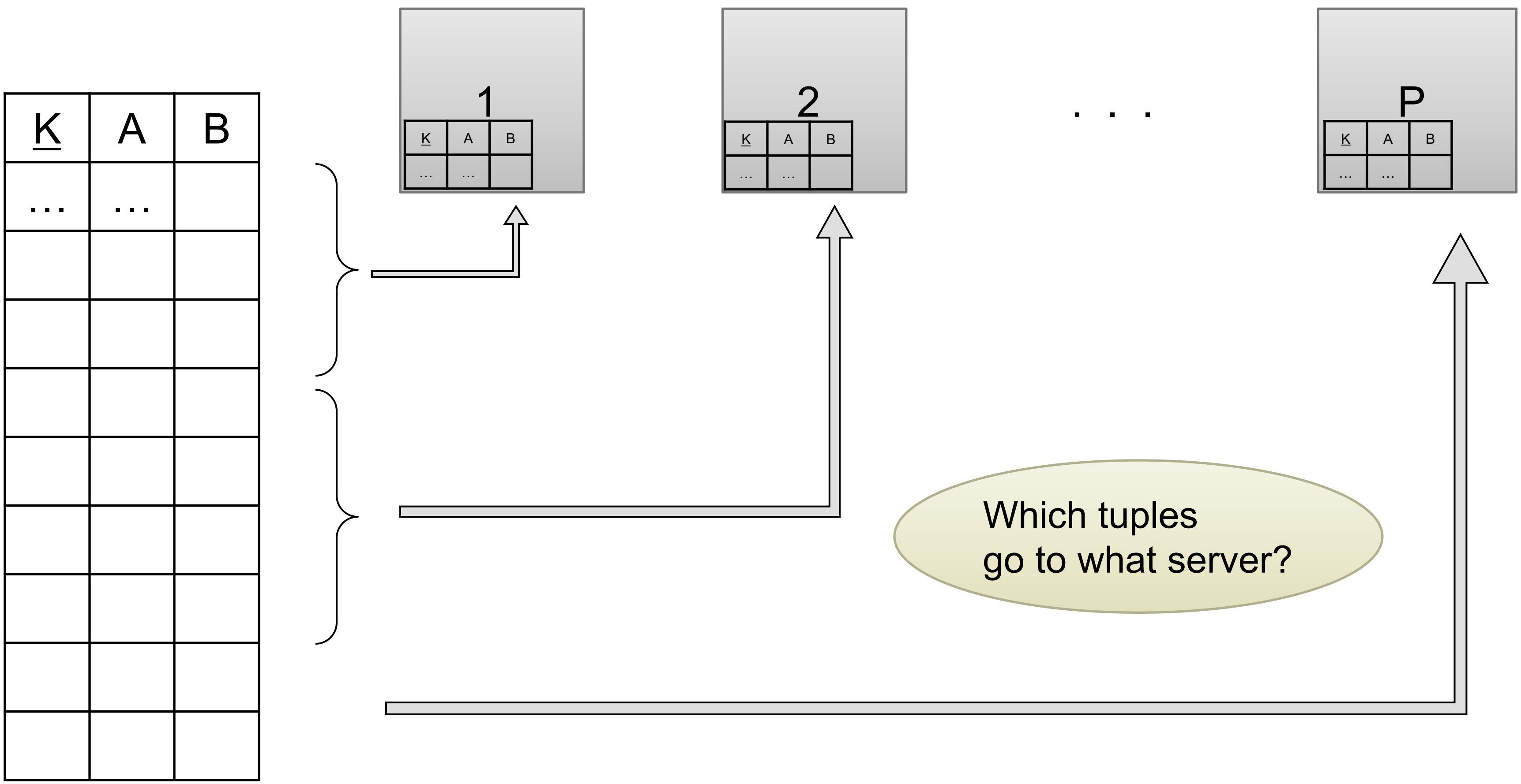
K	A	B
…	…	

Servers:

HORIZONTAL DATA PARTITIONING

Data:

Servers:



HORIZONTAL DATA PARTITIONING

Block Partition:

- Partition tuples arbitrarily s.t. $size(R_1) \approx \ldots \approx size(R_P)$

Hash partitioned on attribute A:

- Tuple t goes to chunk i, where i = h(t.A) mod P + 1
- Recall: calling hash fn's is free in this class

Range partitioned on attribute A:

- Partition the range of A into $-\infty = v_0 < v_1 < \ldots < v_P = \infty$
- Tuple t goes to chunk i, if $v_{i-1} < t.A < v_i$

UNIFORM DATA V.S. SKEWED DATA

Let R(K,A,B,C); which of the following partition methods may result in skewed partitions?

Block partition — Uniform

Hash-partition — Uniform

Assuming good hash function

- On the key K
- On the attribute A

Range partition — May be skewed

E.g. when all records have the same value of the attribute A, then all records end up in the same partition

Keep this in mind in the next few slides