Architectures; Parallel Databases

Amol Deshpande
CMSC424

Spring 2020 – Online Instruction Plan

Modified to swap the last two projects

- Week 1: File Organization and Indexes
- Week 2: Query Processing
- Week 3: Query Optimization; Architectures/Parallel 1
- Week 4: Parallel Databases + MapReduce; Transactions 1
- Week 5: Transactions 2

Spring 2020 – Online Instruction Plan

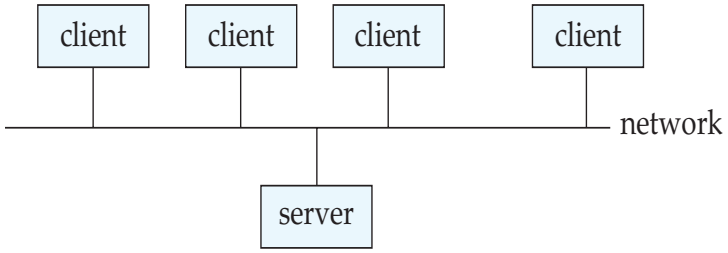
- Week 1: File Organization and Indexes
- Week 2: Query Processing
- Week 3 (Homework Due April 17, Noon)
  - ★ Query Optimization 1: Overview, Statistics
  - ★ Query Optimization 2: Equivalences, Search Algorithms
  - ★ Architectures/Parallel Databases Introduction
- Week 4: Parallel Databases; Mapreduce; Transactions 1
  - ★ Map-reduce and Apache Spark (will post early for Project 5)
- Week 5: Transactions 2

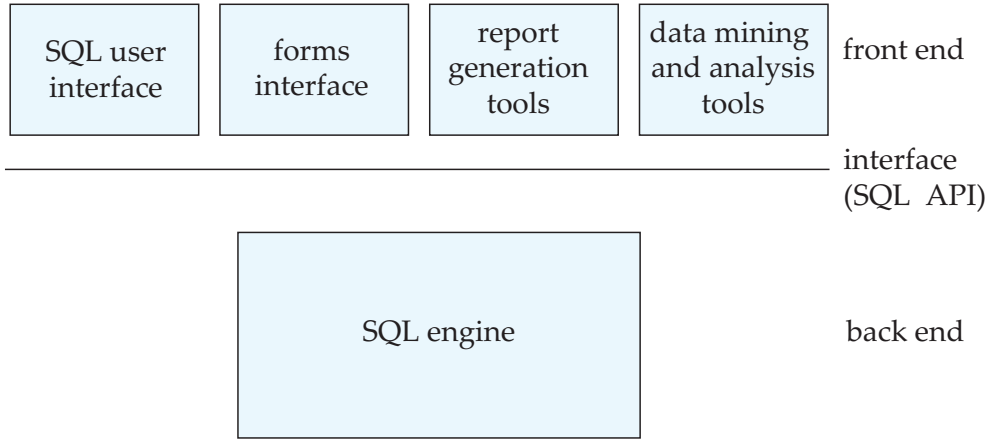
Architectures; Parallel Databases

- Book Chapters
  - 17.1, 17.3, 18.1, 18.2
- Key topics:
  - Brief overview of different types of architectures
  - Why parallel databases are critical today for performance
  - Speedup vs Scaleup
  - How to distribute data across a collection of parallel disks

Client-Server Systems

- Database functionality can be divided into:
  - **Back-end**: manages access structures, query evaluation and optimization, concurrency control and recovery.
  - **Front-end**: consists of tools such as *forms*, *report-writers*, and graphical user interface facilities.
- The interface between the front-end and the back-end is through SQL or through an application program interface.





Parallel Databases

- Why ?
  - More transactions per second, or less time per query
  - Throughput vs. Response Time
  - Speedup vs. Scaleup
- Database operations are *embarrassingly parallel*
  - E.g. Consider a join between R and S on R.b = S.b
- But, perfect speedup doesn't happen
  - Start-up costs
  - Interference
  - Skew

Parallel Databases

- Parallel machines increasingly very common and affordable
- Databases growing increasingly large ("BIG" data)
- Large-scale parallel database systems increasingly used for:
  - storing large volumes of data
  - processing time-consuming decision-support queries
  - providing high throughput for transaction processing

- Key Questions for Database People:
  - How to partition data across a collection of storage devices (disks)
  - How to execute an "operation" across a group of computers
    - In different configurations (shared-memory vs shared-disk vs shared-nothing vs NUMA)
    - Trade-offs and bottlenecks can be vastly different
  - How to deal with "failures"

Parallel Systems

- Parallel database systems consist of multiple processors and multiple disks connected by a fast interconnection network.
- A **coarse-grain parallel** machine consists of a small number of powerful processors
- A **massively parallel** or **fine grain parallel** machine utilizes thousands of smaller processors.
- Two main performance measures:
  - **throughput** --- the number of tasks that can be completed in a given time interval
  - **response time** --- the amount of time it takes to complete a single task from the time it is submitted

Speed-Up and Scale-Up

- **Speedup**: a fixed-sized problem executing on a small system is given to a system which is *N*-times larger.
  - Measured by:

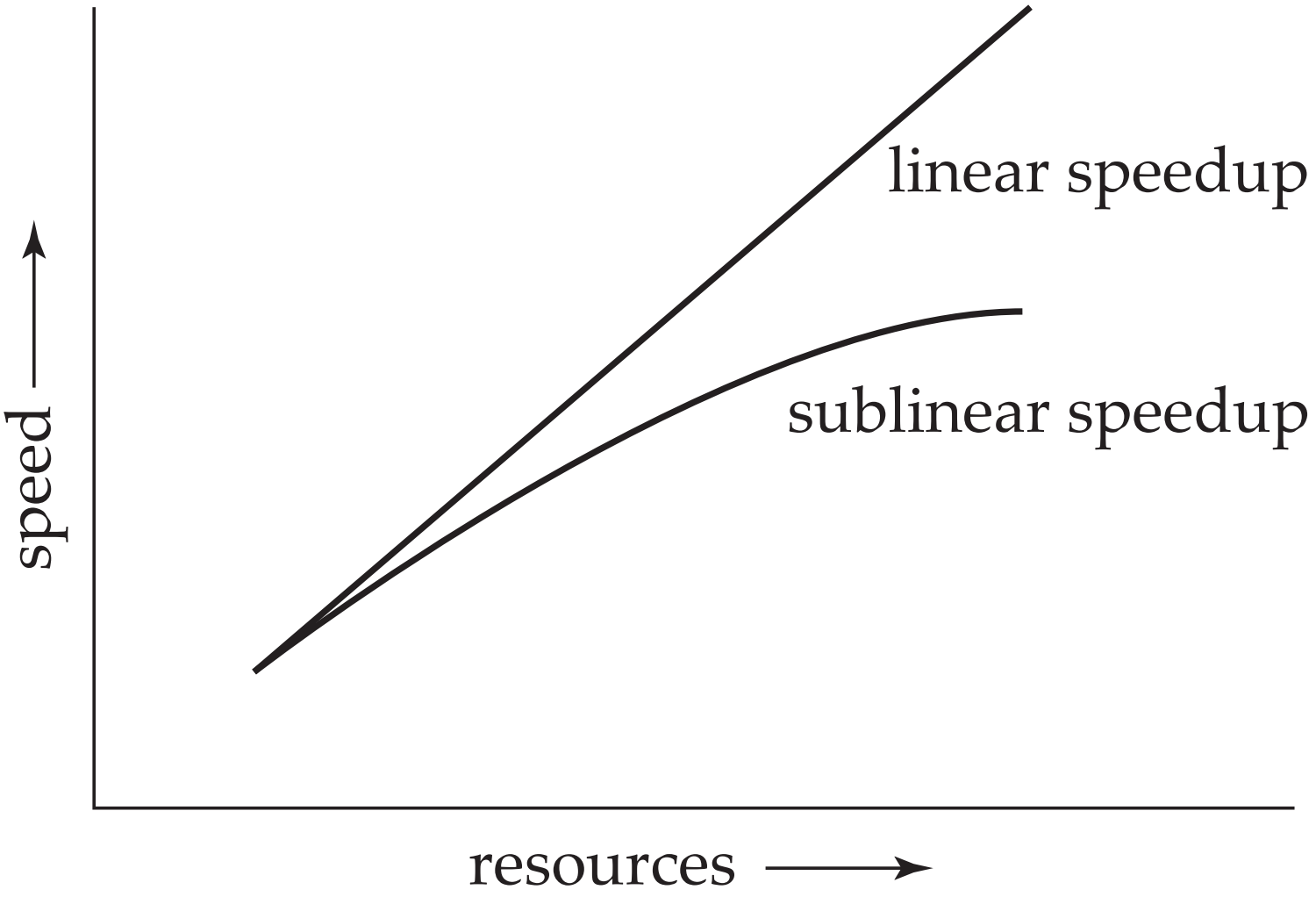
$$speedup = \frac{small\ system\ elapsed\ time}{large\ system\ elapsed\ time}$$

  - Speedup is **linear** if equation equals N.
- **Scaleup**: increase the size of both the problem and the system
  - *N*-times larger system used to perform *N*-times larger job
  - Measured by:

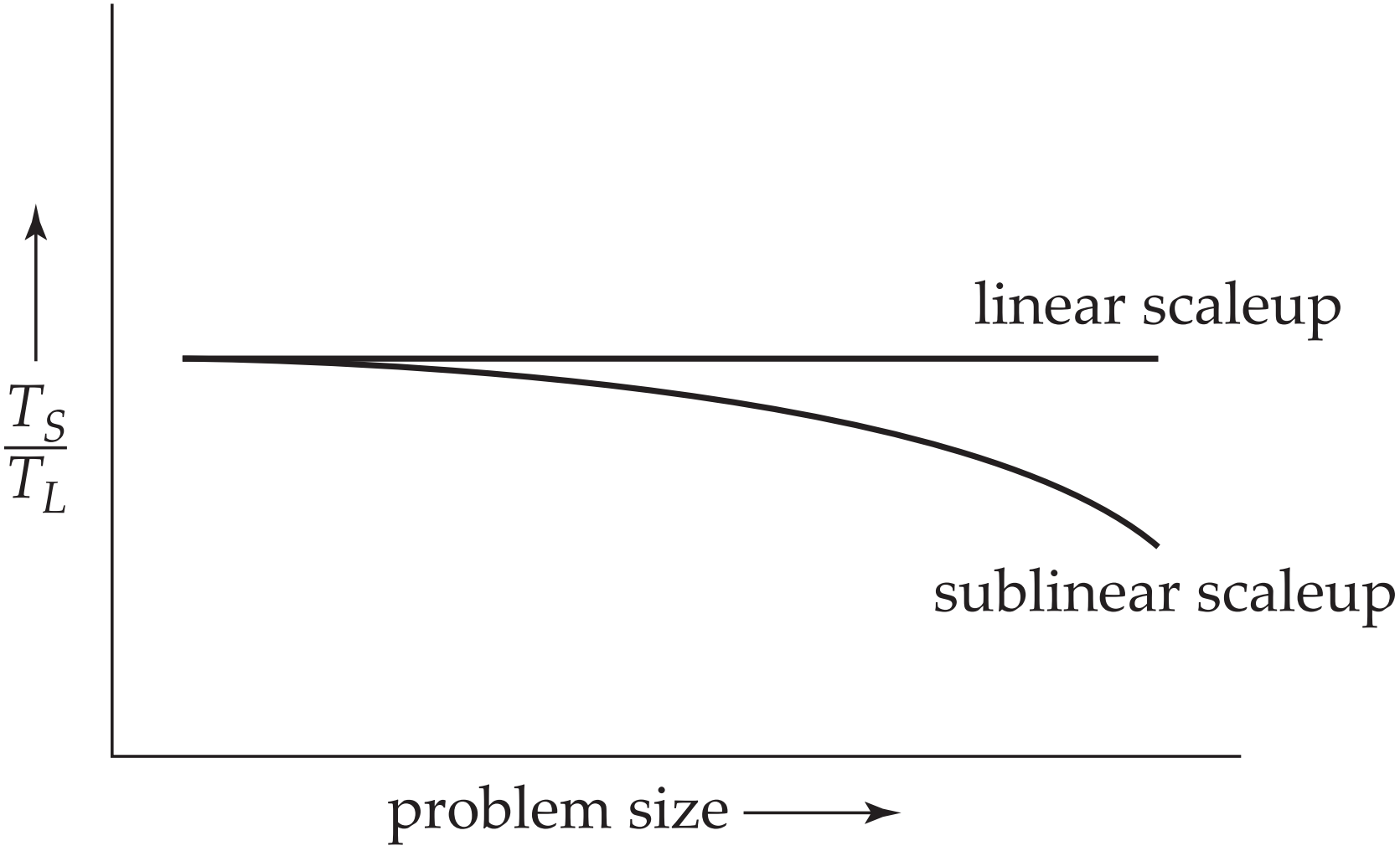
$$scaleup = \frac{small\ system\ small\ problem\ elapsed\ time}{big\ system\ big\ problem\ elapsed\ time}$$

  - Scale up is **linear** if equation equals 1.

Speedup



Scaleup



Factors Limiting Speedup and Scaleup

Speedup and scaleup are often sublinear due to:

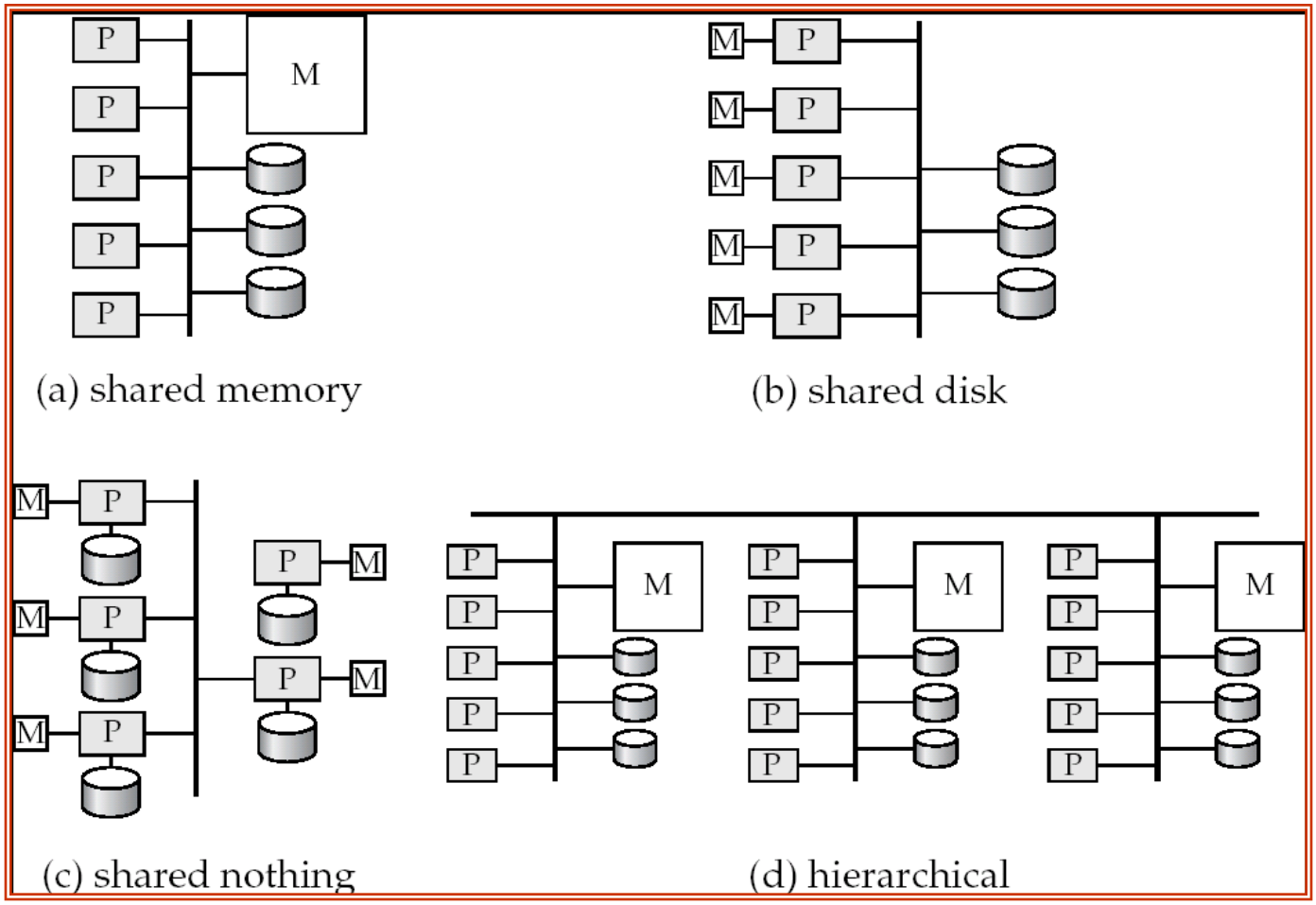
- **Startup costs**: Cost of starting up multiple processes may dominate computation time, if the degree of parallelism is high.

- **Interference**: Processes accessing shared resources (e.g., system bus, disks, or locks) compete with each other, thus spending time waiting on other processes, rather than performing useful work.

- **Skew**: Increasing the degree of parallelism increases the variance in service times of parallely executing tasks. Overall execution time determined by **slowest** of parallely executing tasks.

Parallel Databases

- Shared-nothing vs. shared-memory vs. shared-disk



Parallel Databases

	Shared Memory	Shared Disk	Shared Nothing
Communication between processors	Extremely fast	Disk interconnect is very fast	Over a LAN, so slowest
Scalability ?	Not beyond 32 or 64 or so (memory bus is the bottleneck)	Not very scalable (disk interconnect is the bottleneck)	Very very scalable
Notes	Cache-coherency an issue	Transactions complicated; natural fault-tolerance.	Distributed transactions are complicated (deadlock detection etc);
Main use	Low degrees of parallelism	Not used very often	Everywhere

Aside: Distributed Databases

- Over a wide area network
- Typically not done for *performance reasons*
  - For that, use a parallel system
- Done because of necessity
  - Imagine a large corporation with offices all over the world
  - Also, for redundancy and for disaster recovery reasons
- Lot of headaches
  - Especially if trying to execute transactions that involve data from multiple sites
    - Keeping the databases in sync
      - *2-phase commit* for transactions uniformly hated
    - Autonomy issues
      - Even within an organization, people tend to be protective of their unit/department
    - Locks/Deadlock management
  - Works better for query processing
    - Since we are only reading the data

Parallel Databases

- Parallel machines increasingly very common and affordable
- Databases growing increasingly large (“BIG” data)
- Large-scale parallel database systems increasingly used for:
  - storing large volumes of data
  - processing time-consuming decision-support queries
  - providing high throughput for transaction processing

- Key Questions for Database People:
  - How to partition data across a collection of storage devices (disks)
  - How to execute an “operation” across a group of computers
    - In different configurations (shared-memory vs shared-disk vs shared-nothing vs NUMA)
    - Trade-offs and bottlenecks can be vastly different
  - How to deal with “failures”

I/O (Storage) Parallelism

- Horizontal partitioning – tuples of a relation are divided among many disks such that each tuple resides on one disk.
- Also called “sharding” in distributed setting
- Partitioning techniques (number of disks = *n*):

  Round-robin:

  Send the *I* th tuple inserted in the relation to disk ***i*** **mod** ***n***.

  Hash partitioning:

  - Choose one or more attributes as the partitioning attributes.
  - Choose hash function *h* with range 0…*n* - 1
  - Let *i* denote result of hash function *h* applied tothe partitioning attribute value of a tuple. Send tuple to disk *i*.

  Range partitioning:

  - Simiarly to “hashing”, but do it based on ranges (e.g., tuples with value of ”A” from 0-100 go to disk1, 101-200 go to disk2, etc).

Comparison of Partitioning Techniques

- How well partitioning techniques support different types of data access ?
  1. Scanning the entire relation.
  2. Locating a tuple associatively – **point queries**. (E.g., $r.A = 25$.)
  3. Locating all tuples such that the value of a given attribute lies within a specified range – **range queries (E.g.,** $10 \leq r.A < 25$.)

	Scanning	Point Queries	Range Queries
Round-robin	Very good – balanced work	Very bad – need to scan all	Very bad – need to scan all
Hashing	Very good – balanced	Very good for queries in partitioning attribute	Bad – some queries can be handled
Range partitioning	Good – harder to guarantee balanced work	Good for queries on partitioning attribute	Very good for queries on partitioning attribute

Handling of Skew

- The distribution of tuples to disks may be **skewed** — that is, some disks have many tuples, while others may have fewer tuples.
- **Types of skew:**
  - **Attribute-value skew**.
    - Some values appear in the partitioning attributes of many tuples; all the tuples with the same value for the partitioning attribute end up in the same partition.
    - Can occur with range-partitioning and hash-partitioning.
  - **Partition skew**.
    - With range-partitioning, badly chosen partition vector may assign too many tuples to some partitions and too few to others.
    - Less likely with hash-partitioning if a good hash-function is chosen.

Dealing with Skew

- Analyze the relation (or a random sample) to create better partitions
  - E.g., instead of dividing in equal ranges, try to identify ranges that are more balanced
  - A random sample usually sufficient for this purpose
  - Can also construct a “histogram” for this purpose
- Another option:
  - Create a large number of partitions (e.g., use 1000 partitions for 10 machines)
  - Map the partitions to machines more carefully
  - Can move partitions around to address skew later in the process
- Attribute-value skew harder to deal with
  - If you want to partition on “zipcode”, and there is a zipcode with half the tuples, not much you can do
    - Zipcode is just not a good partitioning attributes

Summary

- Parallel databases increasingly common because of hardware trends and faster networks
- Databases are “infinitely parallelizable”
  - Data can be partitioned across any number of disks
  - Relational operators are easily parallelizable (as we will see later)
    - But we do need to watch out of skew and interference
- Next Videos:
  - Query execution across a parallel database system