CMSC424: Database Design
Database Architectures; Storage

March 9, 2020

Instructor: Amol Deshpande
amol@cs.umd.edu

Announcements Etc.

- Aim to have the grades out by end of this week

- Project 3:
  - We will briefly discuss the “query plans” part today – other topics are either already covered (Outerjoins, Triggers) or will not be (e.g., Java-JDBC)

- Will start recording lectures and upload them

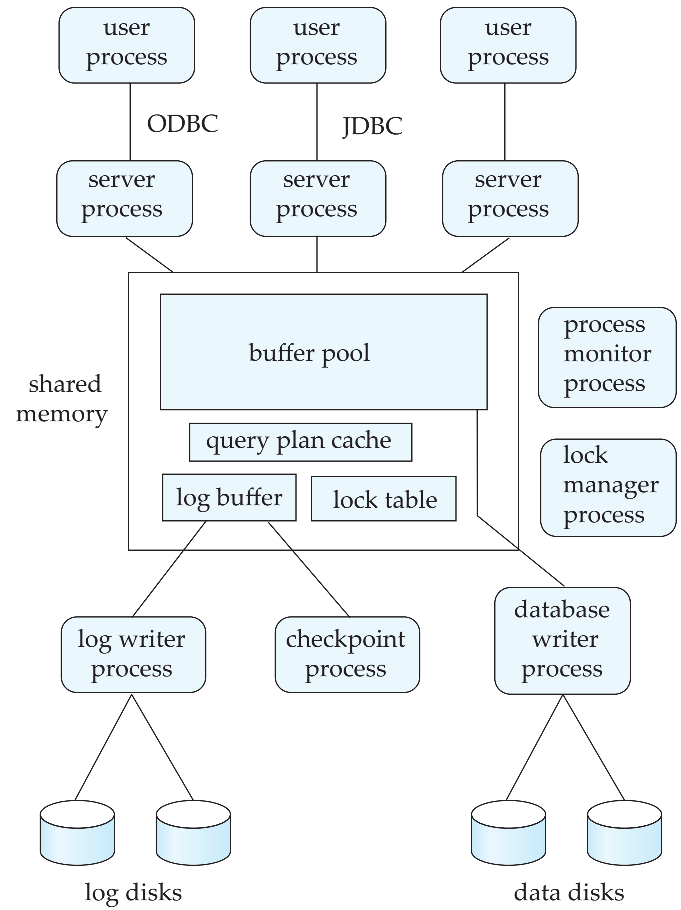
- If you are uncomfortable about coming to office hours in person, message us to set up virtual office hours
  - Will come up with some policy

Plan for Today

- General background and alternatives
- Storage Hierarchy
- Specific Storage Media
  - Disks
  - Solid State Drives

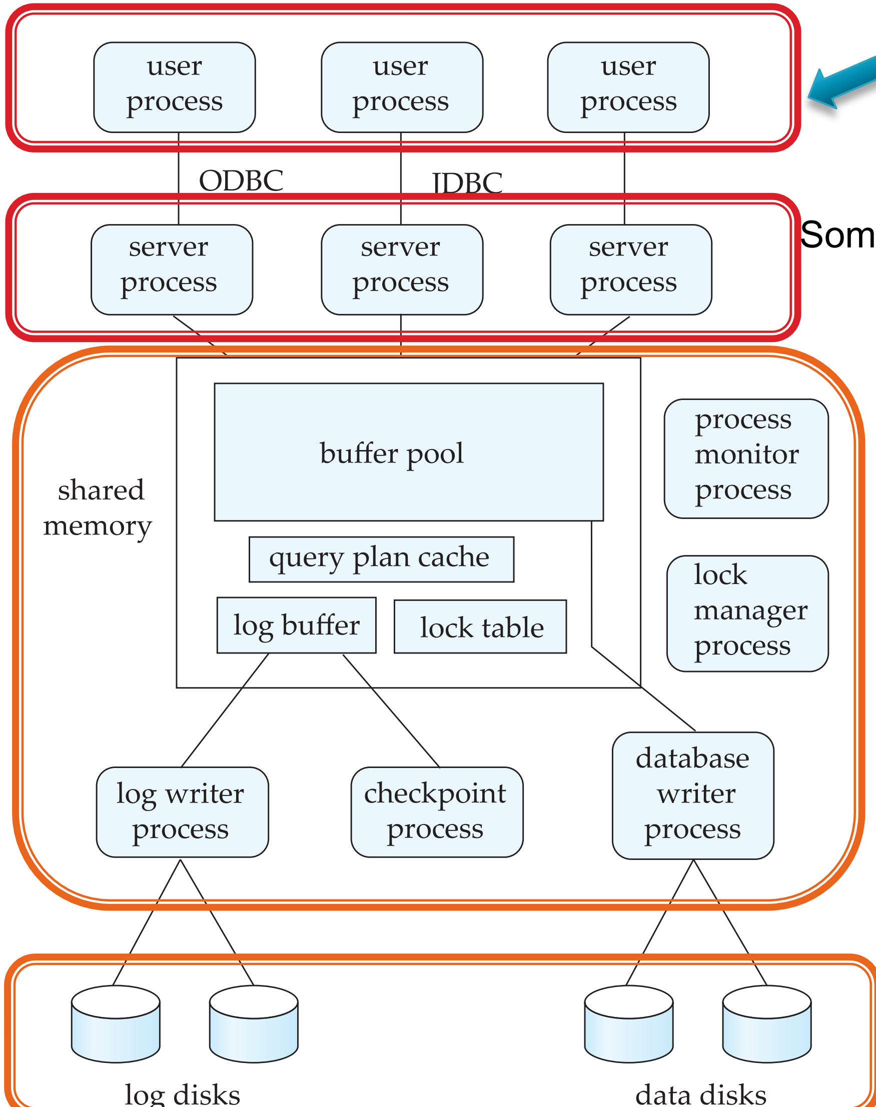
Database Architecture: Pre-2000's

- All data was typically in hard disks or arrays of hard disks
- RAM (Memory) was never enough
  - So always had to worry about what was in memory vs not
- Almost no real "distributed" execution
  - Different from "parallel", i.e., on co-located clusters of computers
- Relatively well-understood use cases
  - Report generation
  - Interactive data analysis and exploration
  - Supporting transactions



From Chapter 20

Traditional RDBMS Architecture



Clients may be anywhere – e.g., ATMs, desktops, laptops, web apps etc.

Talk to the database using standard protocols like JDBC/ODBC, SOAP, or REST (today), or proprietary protocols

Some sort of load balancer or intake mechanism

Typical components in a database system: some for queries, some for transactions

Maybe on a single physical computer or a cluster connected by a fast network

Data Storage Systems:
(1) Punch cards (long time ago)
(2) Hard disks (still prevalent)
(3) SSDs

Need "redundancy" and "fault-tolerance"
Data once stored should always be there

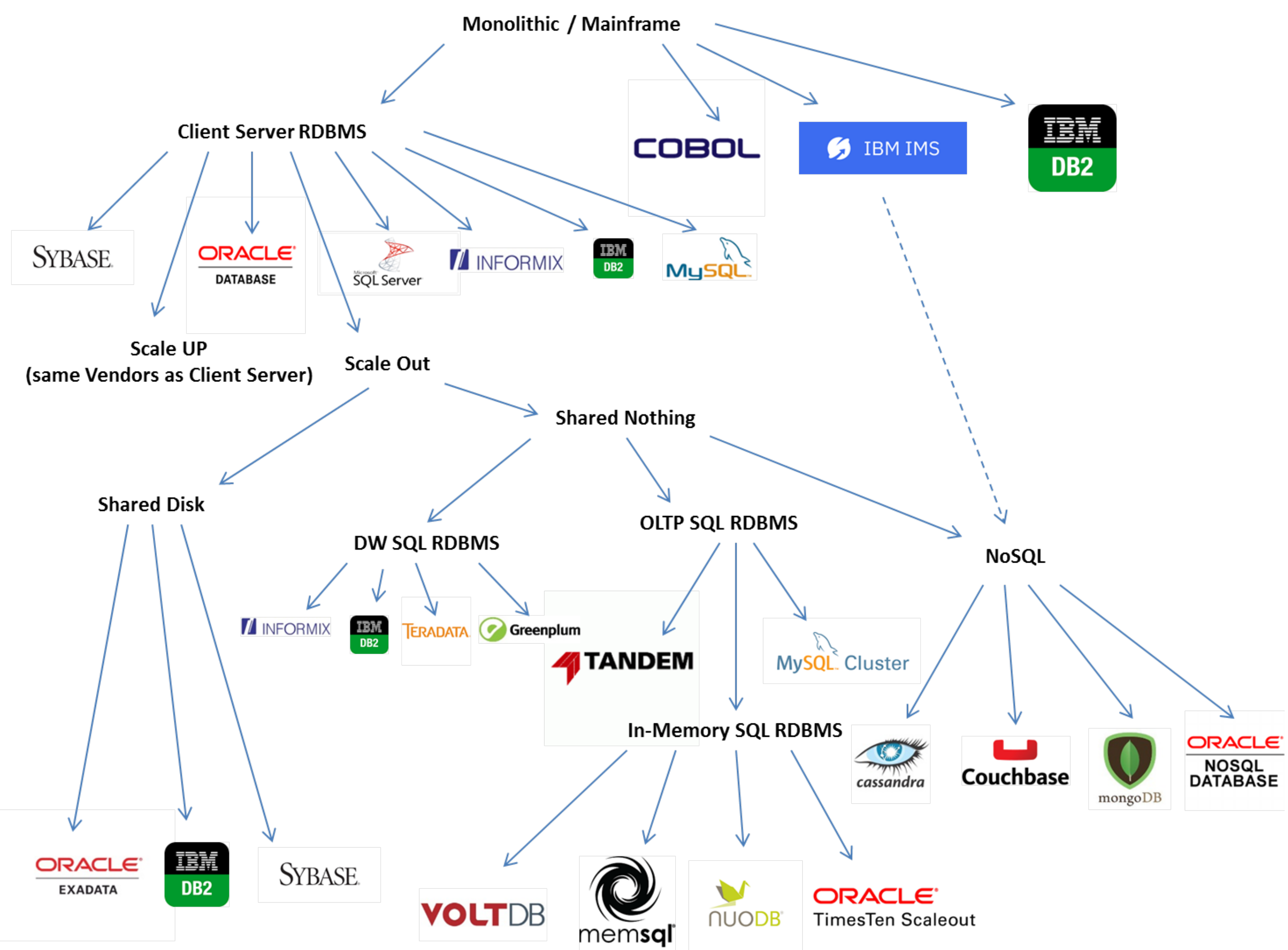
RAID = Redundant Array of Independent Disks

Database Architecture : Today

- Much more diversity in the architectures that we see
  - More modern hardware architectures
    - Massively parallel computers
    - SSDs
    - Massive amounts of RAM – often don't need to worry about data fitting in memory
    - Much faster networks, even over a wide area
    - Virtualization and Containerization
    - Cloud Computing
  - As a result: Data and execution typically distributed all over the place

- Much more diversity in data processing applications
  - Much more non-relational data (images, text, video)
  - Data Analytics/Machine learning more common use-cases

- Much more diversity in "data models"
  - Document data models (JSON, XML), Key-value data model, Graph data model, RDF



From: https://blogs.oracle.com/timesten/the-evolution-of-db-architectures
(Oracle-focused)

Data Warehouses

For: Large-scale data processing (TBs to PBs)

Parallel architectures (lots of co-located computers)

SQL and Reporting

No transactions

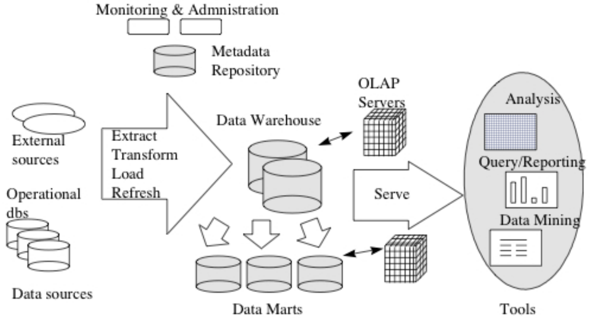


Figure 1. Data Warehousing Architecture

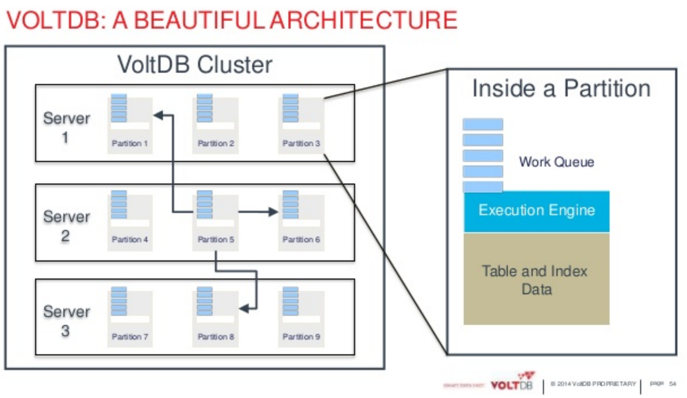
In-memory OLTP (on-line transaction processing)

For: Extremely fast transactions

Many-core or parallel architectures

Very limited SQL – mostly focused on “writes”

Typically assume data fits in memory across servers



Highly available, distributed OLTP

For: Distributed scenarios where clients are all over the world

Focus on “consistency” – how to make sure all users see the same data

Limited SQL – mostly focused on “writes”

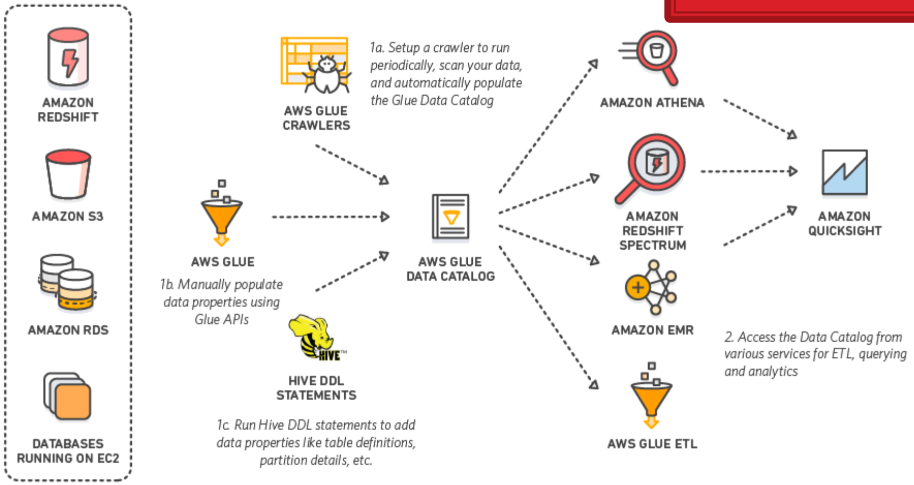
Considerations of memory vs disk less important

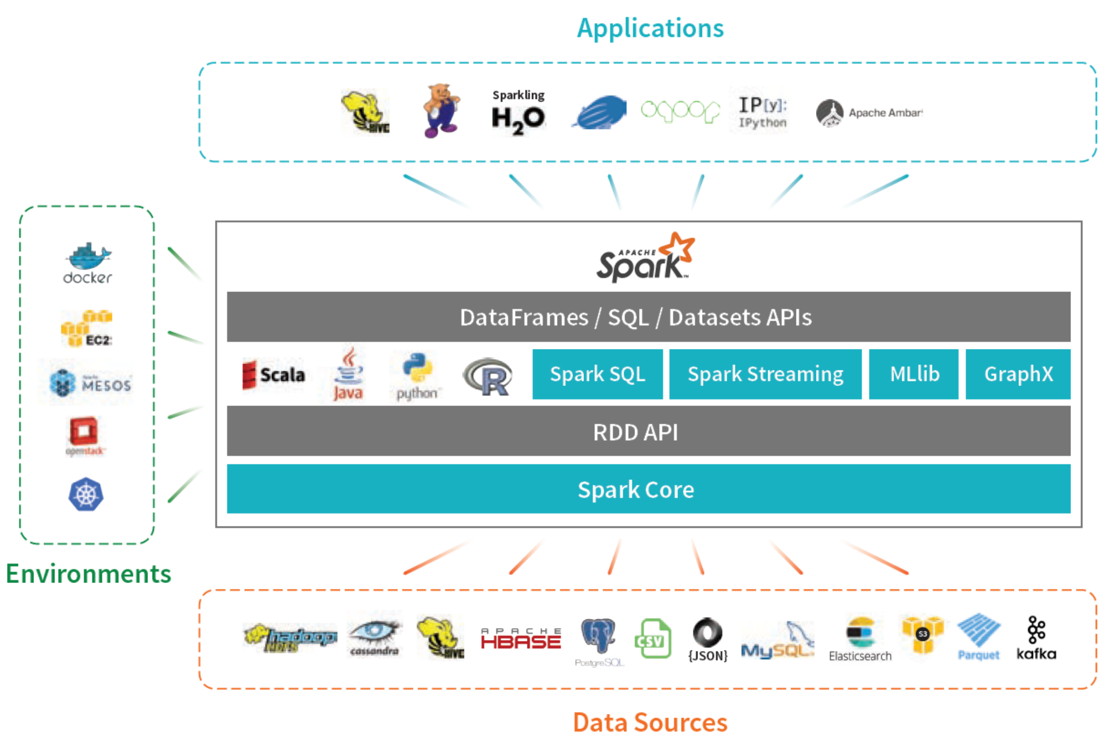
Extract-Transform-Load Systems, or Map-Reduce, or Big Data Frameworks

AWS Glue

For: Large-scale, “ad hoc” data analysis

Mix of parallel and distributed architectures
Data usually coming from many different sources
Mix of SQL, Machine Learning, and ad hoc tasks (e.g., do image analysis, followed by SQL)



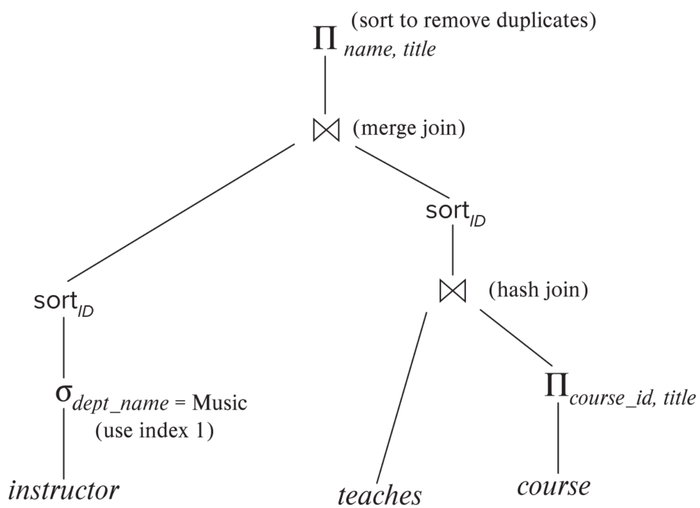


Apache Spark

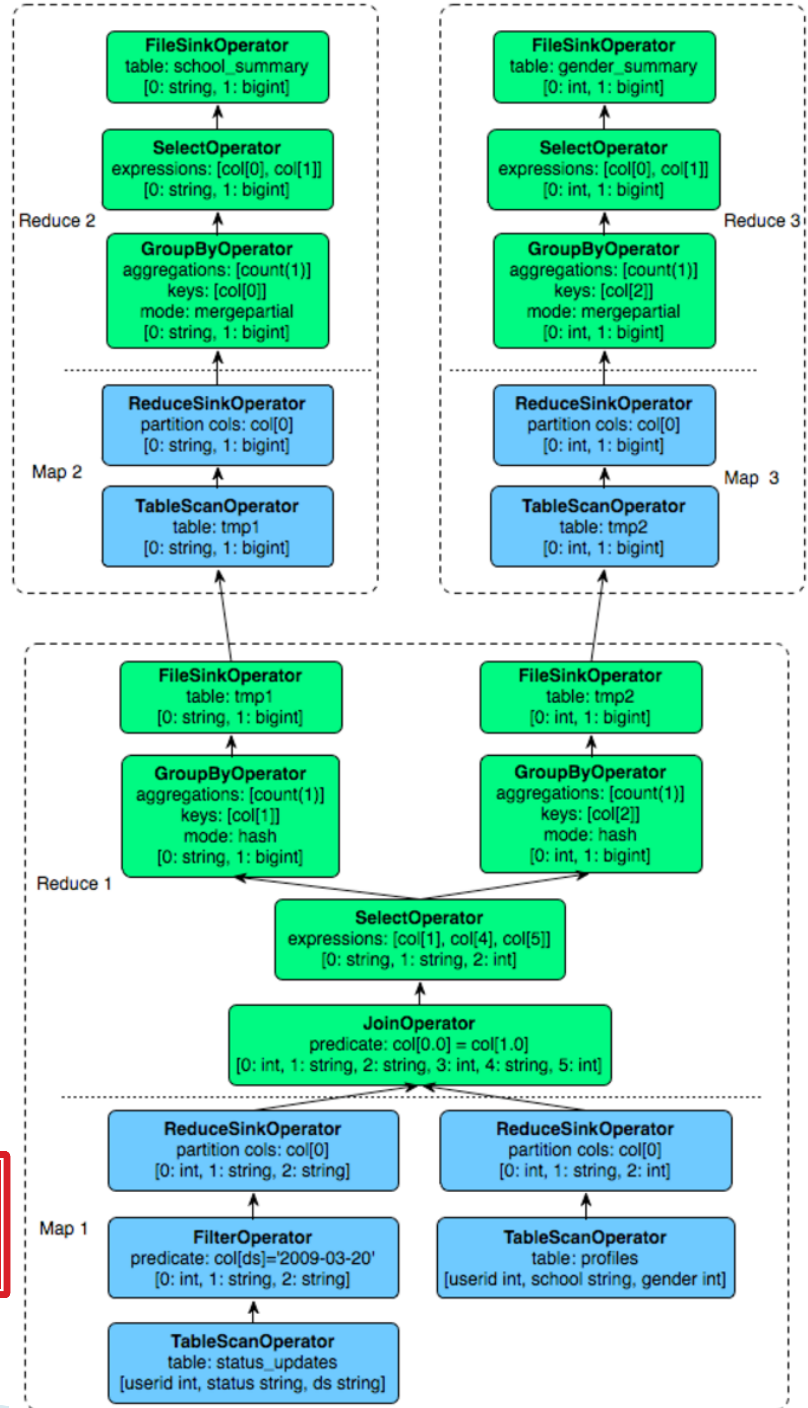
Okay…

- Key takeaway: Modern data architectures are a mess
  - We haven’t talked about NoSQL (MongoDB, etc.), Machine Learning, “Streaming”…
- Fundamentals haven’t changed that much though
  - We are still either:
    - Going from some “input datasets” to an “output dataset” (queries/analytics)
    - Modifying data (transactions)
  - SQL is still very common, albeit often disguised
    - Spark RDD operations map nicely to SQL joins and aggregates (unified now)
    - MongoDB lookups, filters, and aggregates map to joins, selects, and aggregates in SQL
- But “performance trade-offs” are all over the place now
  - 30 years ago, we worried a lot about hard disks and things fitting in memory
  - Today, focus more on networks and distributed operations
- Focus has shifted to other aspects of data processing pipelines
  - Analytics/Machine learning, data cleaning, statistics

Query Plans vs…

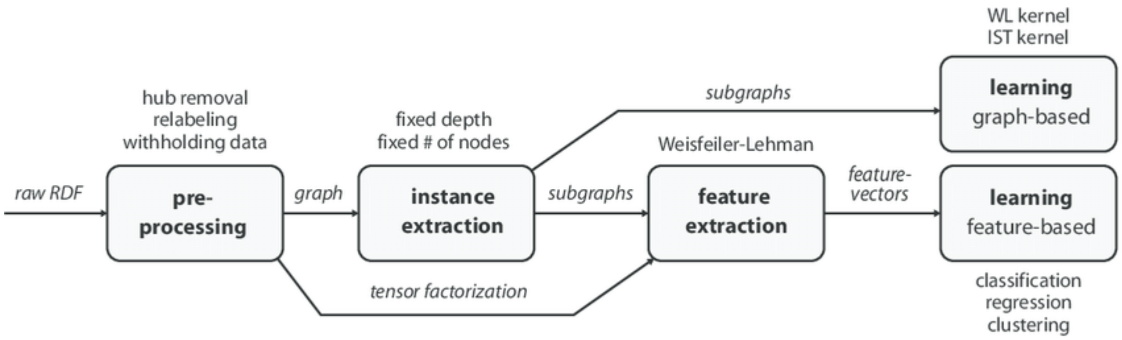


SQL "Query Plan"



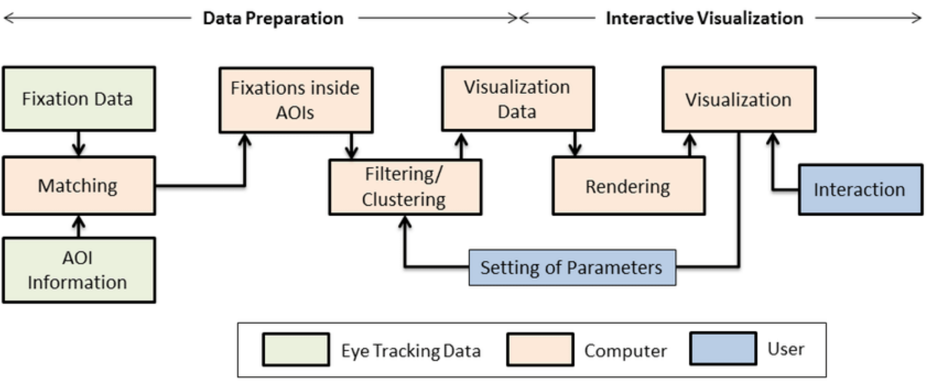
Apache Hive "Query Plan"
(Hive is an SQL layer on top of Hadoop)

vs … Data Transformation Pipelines



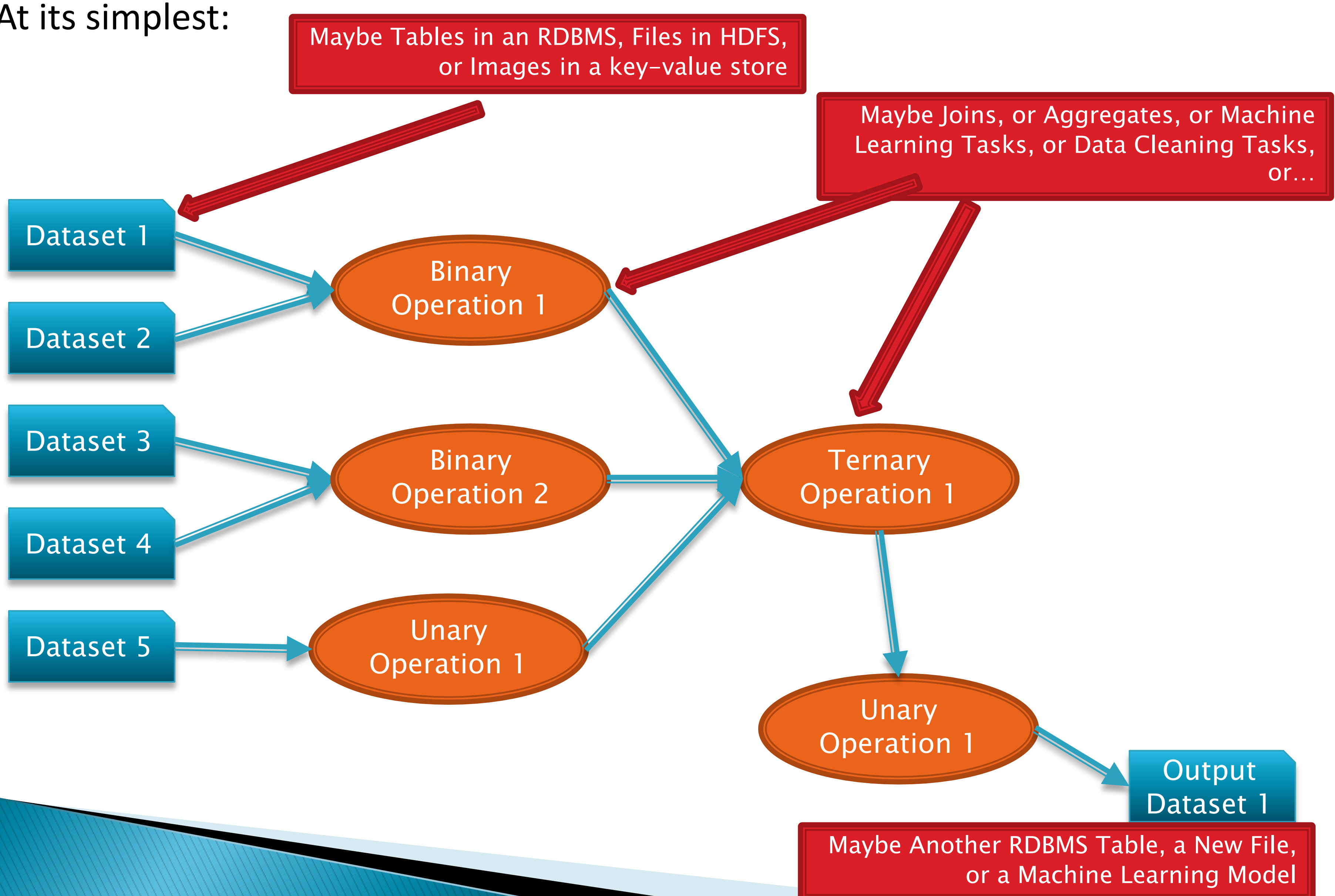
Machine Learning Pipeline

Data Preparation and Visualization Pipeline



Okay…

- Many similarities across different ways to process and analyze data
- At its simplest:



Okay…

- Many similarities across different ways to process and analyze data

- Some considerations that we see repeated:
  - Are there multiple ways to accomplish the goals?
    - i.e., are there multiple pipelines or SQL Query Plans that will accomplish the same task
  - How to “enumerate” all of them?
    - i.e., how to automatically come up with all the different options?
  - How to decide which is the ”best”?
    - Ideally based on some consideration of total cost (e.g., total CPU time)
  - How to ”find” the best plan?
    - Called “query optimization” in databases

- RDBMSs have been doing this for 4-5 decades now
  - The classic paper on SQL query optimization is from 1979
    - Outlined the approach still in use today
- Same ideas re-discovered repeatedly in other contexts (e.g., Hadoop)

In This Class...

- We have to limit the scope drastically

- Focus on:
  - Single-server Relational Databases
  - Assume hard disks are still important and memory is limited
  - Go deep into different ways to execute queries, and find the best queries

- Will briefly discuss:
  - Parallel architectures and query processing there
  - Map-reduce architectures and considerations there-in

- Most of the key concepts valid in modern databases (including NoSQL) and Big Data Frameworks

Plan for Today

- General background and alternatives

- Storage Hierarchy

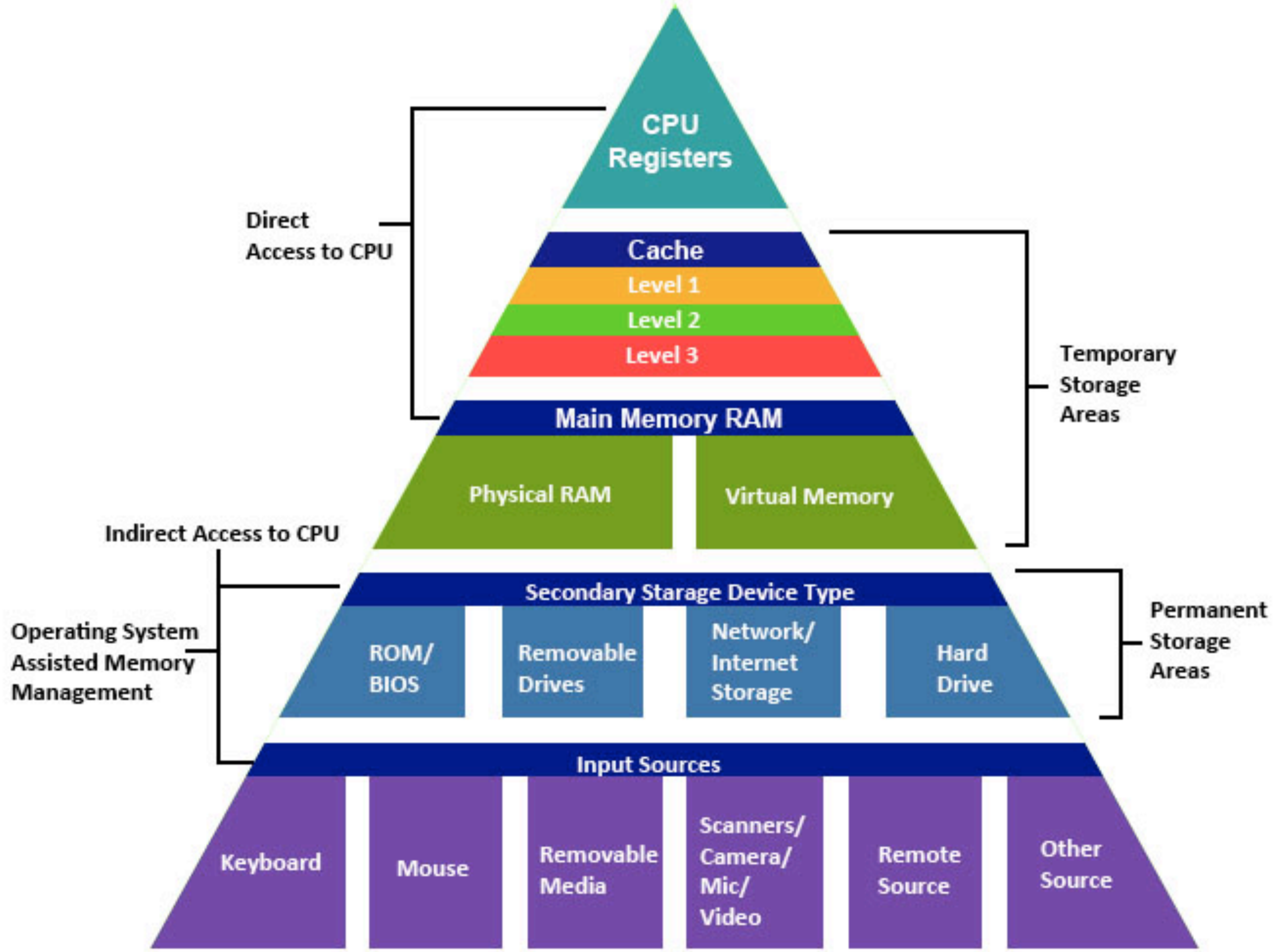
- Specific Storage Media
  - Disks
  - Solid State Drives

Storage Hierarchy

- Tradeoffs between speed and cost of access
- Volatile vs nonvolatile
  - Volatile: Loses contents when power switched off
- Sequential vs random access
  - Sequential: read the data contiguously
    - select * from employee
  - Random: read the data from anywhere at any time
    - select * from employee where name like ‘__a__b’
- Why care ?
  - Need to know how data is stored in order to optimize, to understand what’s going on

How important is this today?

- Trade-offs shifted drastically over last 10-15 years
  - Especially with fast network, SSDs, and high memories
  - However, the volume of data is also growing quite rapidly
- Some observations:
  - Cheaper to access another computer's memory than accessing your own disk
  - Cache is playing more and more important role
  - Enough memory around that data often fits in memory of a single machine, or a cluster of machines
  - "Disk" considerations less important
    - Still: Disks are where most of the data lives today
  - Similar reasoning/algorithms required though

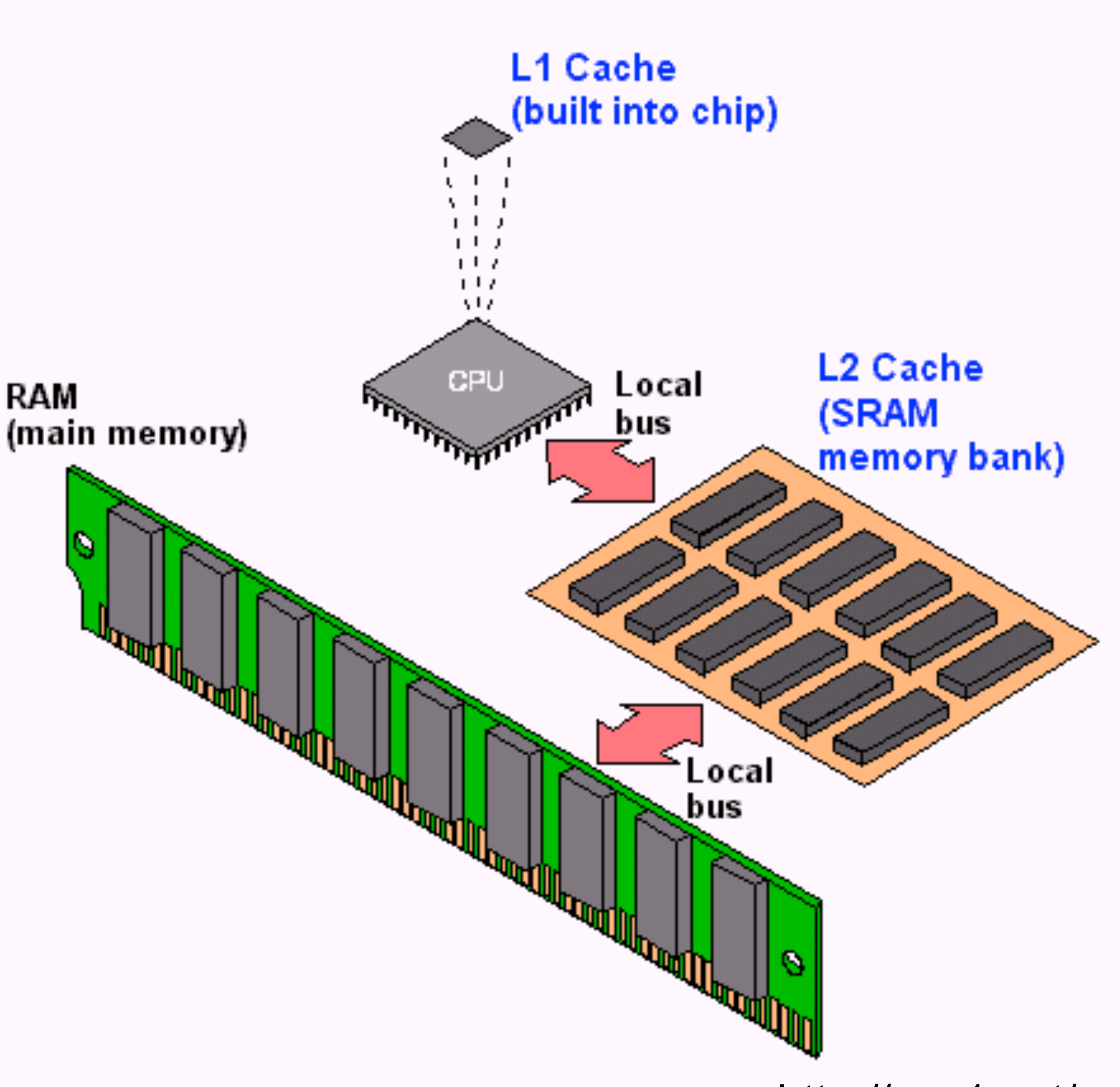


source: http://cse1.net/recaps/4-memory.html

Storage Hierarchy: Cache

- Cache
  - Super fast; volatile; Typically on chip
  - L1 vs L2 vs L3 caches ???
    - L1 about 64KB or so; L2 about 1MB; L3 8MB (on chip) to 256MB (off chip)
    - Huge L3 caches available now-a-days
  - Becoming more and more important to care about this
    - Cache misses are expensive
  - Similar tradeoffs as were seen between main memory and disks
  - Cache-coherency ??

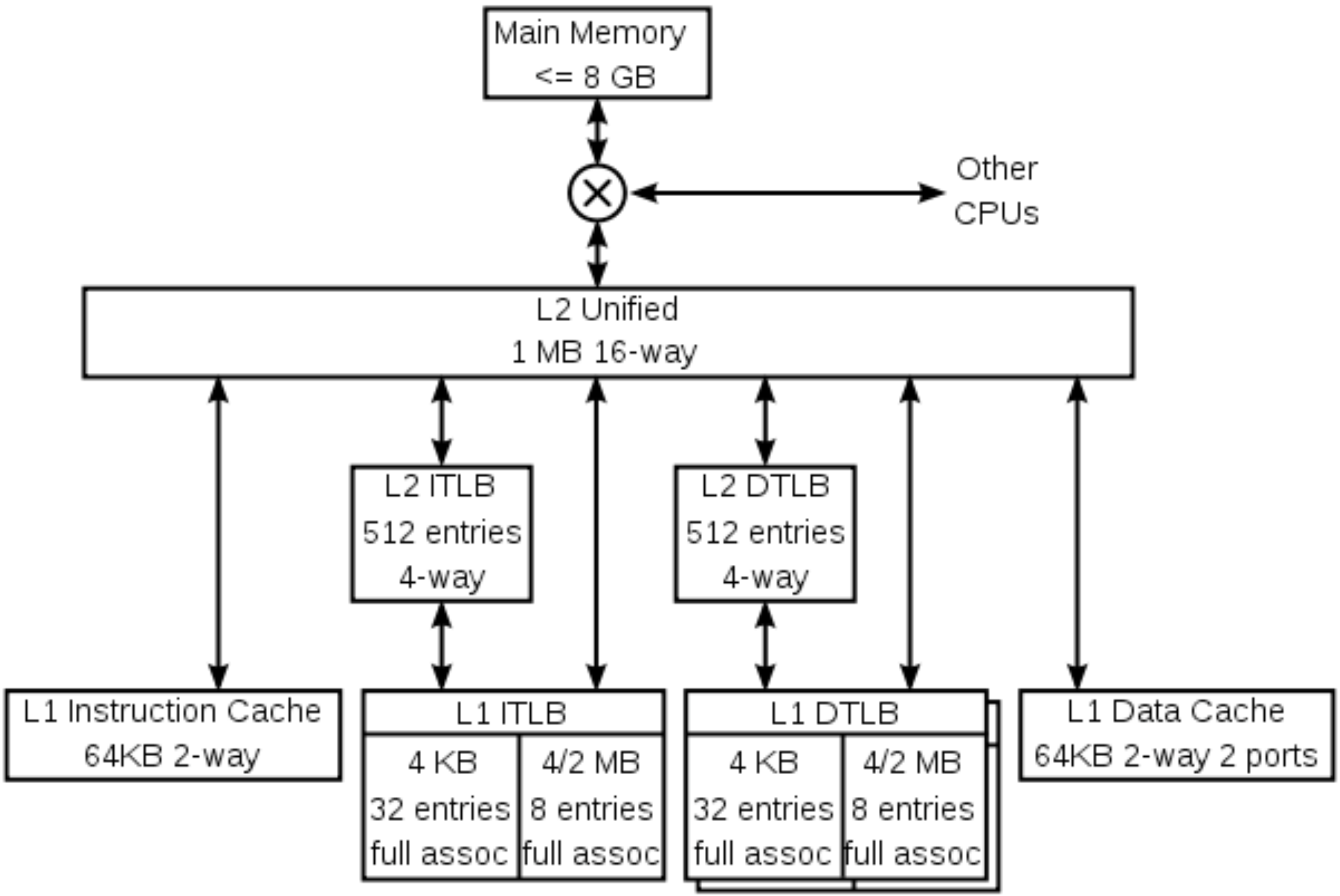
Storage Hierarchy: Cache



source: http://cse1.net/recaps/4-memory.html

Storage Hierarchy: Cache

K8 core in the AMD Athlon 64 CPU



Storage Hierarchy

- Main memory
  - 10s or 100s of ns; volatile
  - Pretty cheap and dropping: 1GByte < 100$
  - Main memory databases feasible now-a-days
- Flash memory
  - Limited number of write/erase cycles
  - Non-volatile, slower than main memory (especially writes)
  - Examples ?
- *Question*
  - *How does what we discuss next change if we use flash memory only ?*
  - *Key issue: <u>Random access as cheap as sequential access</u>*

Storage Hierarchy

- Magnetic Disk (Hard Drive)
  - Non-volatile
  - Sequential access much much faster than random access
  - Discuss in more detail later
- Optical Storage - CDs/DVDs; Jukeboxes
  - Used more as backups… Why ?
  - Very slow to write (if possible at all)
- Tape storage
  - Backups; super-cheap; painful to access
  - IBM just released a secure tape drive storage solution

Storage…

- Primary
  - e.g. Main memory, cache; typically volatile, fast
- Secondary
  - e.g. Disks; Solid State Drives (SSD); non-volatile
- Tertiary
  - e.g. Tapes; Non-volatile, super cheap, slow

Storage Hierarchy

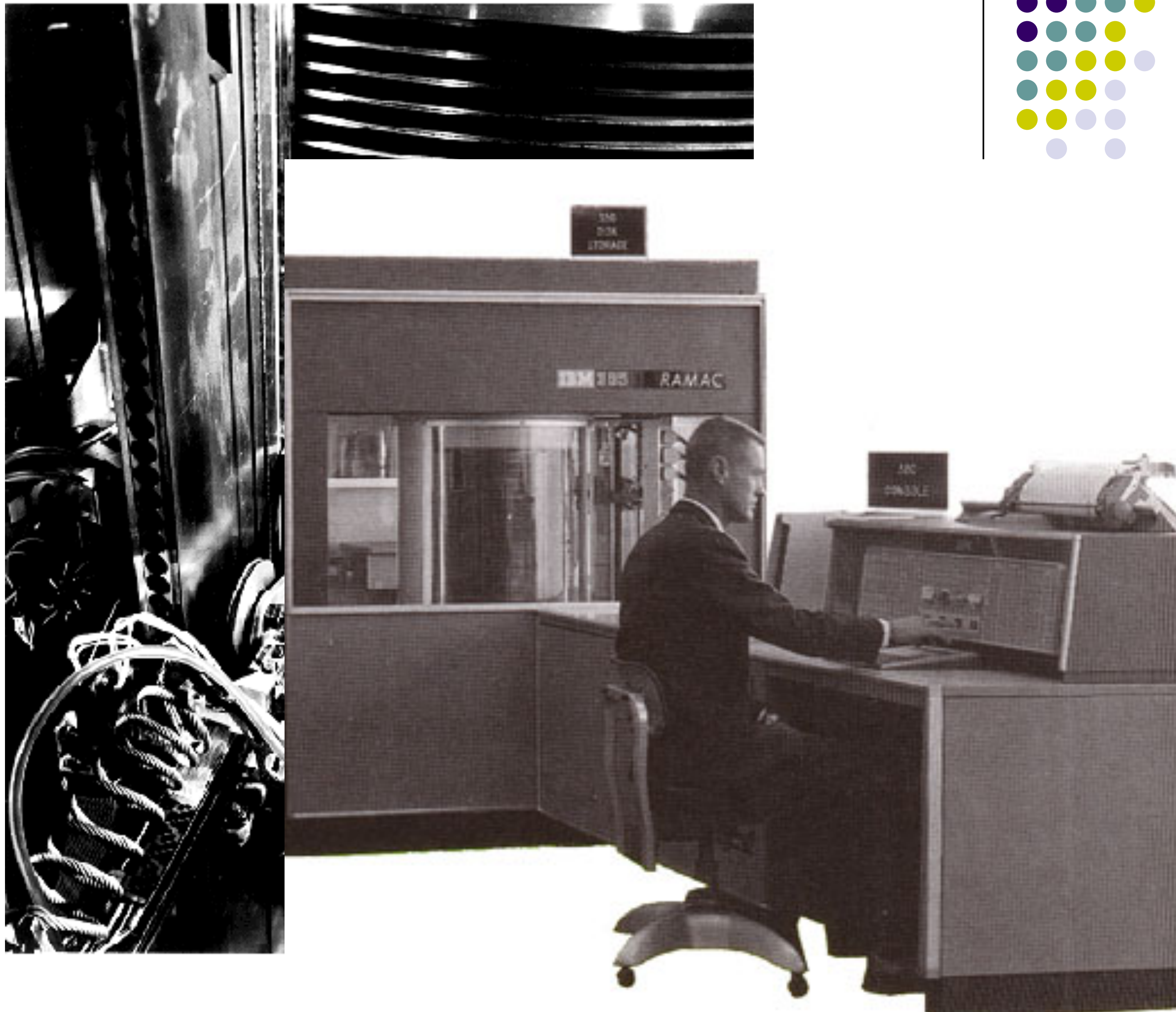
Storage type	Access time	Relative access time
L1 cache	0.5 ns	Blink of an eye
L2 cache	7 ns	4 seconds
1MB from RAM	0.25 ms	5 days
1MB from SSD	1 ms	23 days
HDD seek	10 ms	231 days
1MB from HDD	20 ms	1.25 years

source: http://cse1.net/recaps/4-memory.html

Plan for Today

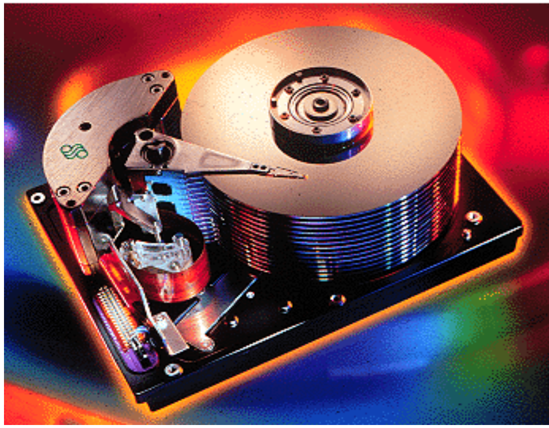
- General background and alternatives
- Storage Hierarchy
- Specific Storage Media
  - Disks
  - Solid State Drives

1956
IBM RAMAC
24” platters
100,000 characters each
5 million characters



1979
SEAGATE
5MB

1998
SEAGATE
47GB



2006
Western Digital
500GB
Weight (max. g): 600g

Latest:

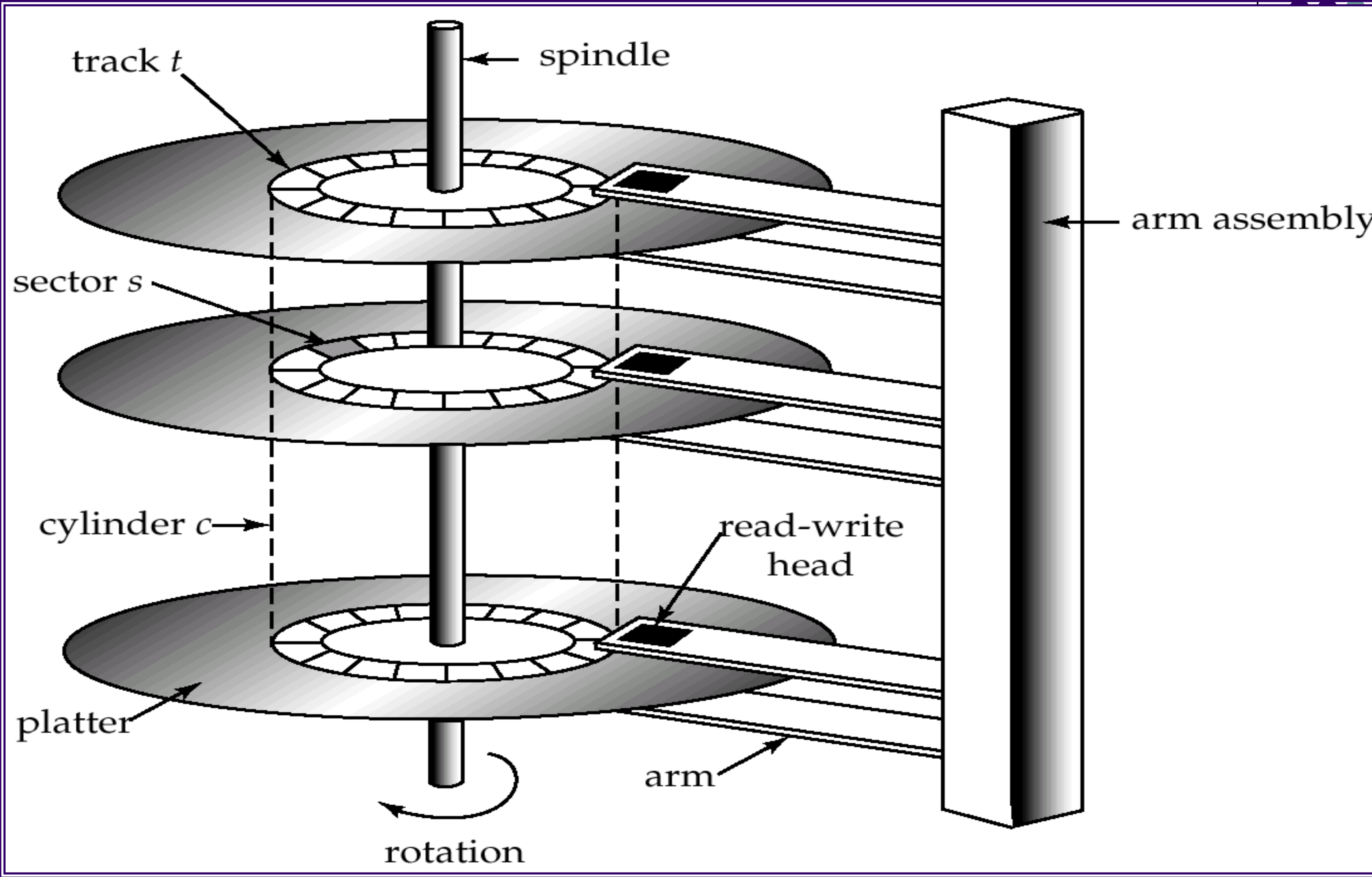
Single hard drive:

Seagate Barracuda 7200.10 SATA
750 GB
7200 rpm
weight: 720g
Uses “perpendicular recording”

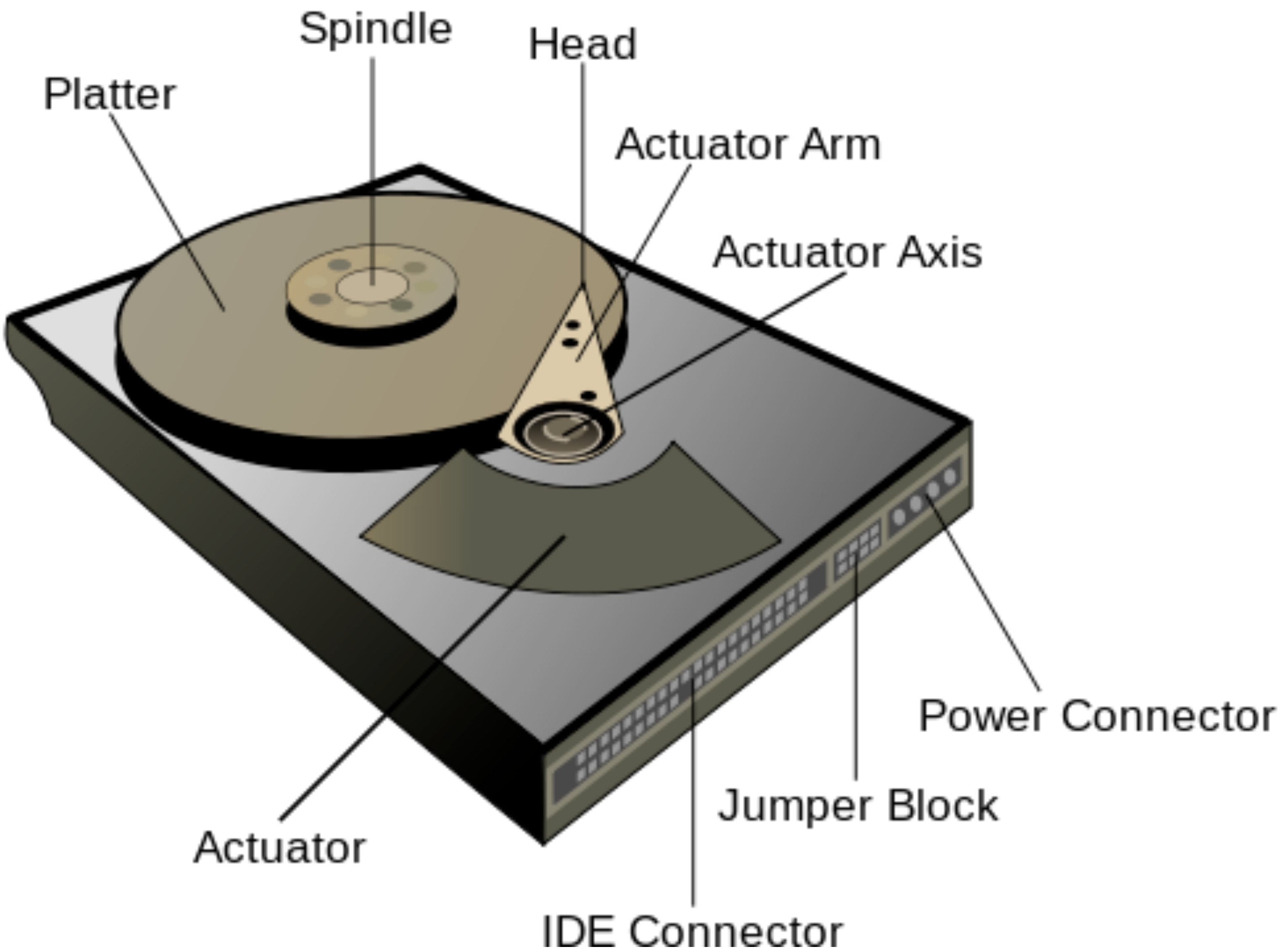
Microdrives

IBM 1 GB

Toshiba 80GB



track t
spindle
arm assembly
sector s
cylinder c
read-write
head
platter
arm
rotation


Spindle
Head
Platter
Actuator Arm
Actuator Axis
Power Connector
Jumper Block
Actuator
IDE Connector

“Typical” Values

Diameter:	1 inch → 15 inches
Cylinders:	100 → 2000
Surfaces:	1 or 2
(Tracks/cyl)	2 (floppies) → 30
Sector Size:	512B → 50K
Capacity →	360 KB to 2TB (as of Feb 2010)
Rotations per minute (rpm) →	5400 to 15000

Accessing Data

- Accessing a sector
  - Time to *seek* to the track (seek time)
    - average 4 to 10ms
  - + Waiting for the sector to get under the head (rotational latency)
    - average 4 to 11ms
  - + Time to transfer the data (transfer time)
    - very low
  - About 10ms per access
    - So if randomly accessed blocks, can only do 100 block transfers
    - 100 x 512bytes = 50 KB/s
- Data transfer rates
  - Rate at which data can be transferred (w/o any seeks)
  - 30-50MB/s to up to 200MB/s (Compare to above)
    - Seeks are bad !

Seagate Barracuda: 1TB

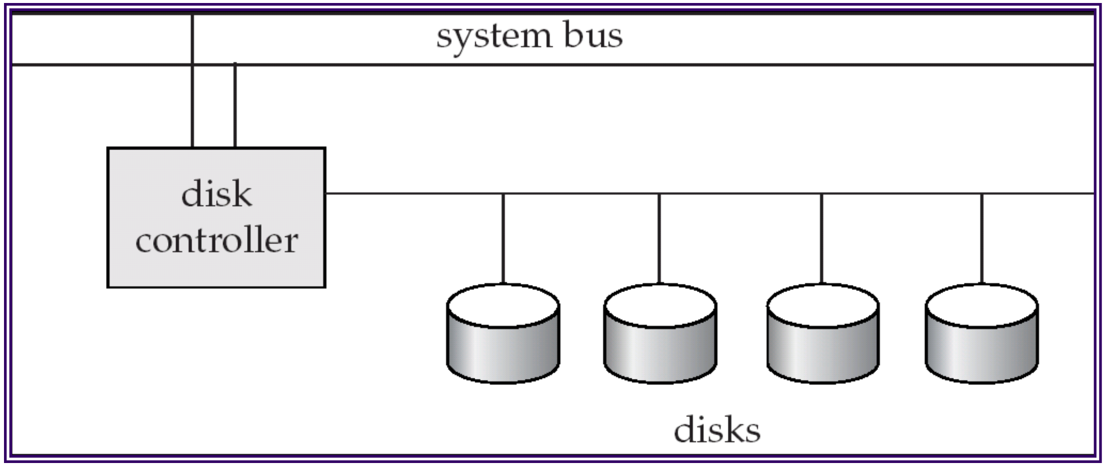
- Heads 8, Disks 4
- Bytes per sector: 512 bytes
- Default cylinders: 16,383
- Defaults sectors per track: 63
- Defaults read/write heads: 16
- Spindle speed: 7200 rpm
- Internal data transfer rate: 1287 Mbits/sec max
- Average latency: 4.16msec
- Track-to-track seek time: 1msec-1.2msec
- Average seek: 8.5-9.5msec
- We also care a lot about power now-a-days
    - Why ?

Reliability

- Mean time to/between failure (MTTF/MTBF):
  - 57 to 136 years
- Consider:
  - 1000 new disks
  - 1,200,000 hours of MTTF each
  - On average, one will fail 1200 hours = 50 days !

Disk Controller

- Interface between the disk and the CPU
- Accepts the commands
- *checksums* to verify correctness
- Remaps bad sectors



Optimizing block accesses

- Typically sectors too small
- Block: A contiguous sequence of sectors
  - 512 bytes to several Kbytes
  - All data transfers done in units of blocks
- Scheduling of block access requests ?
  - Considerations: *performance* and *fairness*
  - <u>*Elevator algorithm*</u>