Database System Architecture

Outline

System R discussion

Relational DBMS architecture

Alternative architectures & tradeoffs

Outline

System R discussion

Relational DBMS architecture

Alternative architectures & tradeoffs

System R Design

Already had essentially the same architecture as a modern RDBMS!

» SQL

» Many storage & access methods (B-trees, etc)

» Cost-based optimizer

» Compiling queries to assembly

» Lock manager

» Recovery via log + shadow pages

» View-based access control

System R Motivation

Navigational DBMS are hard to use

Can relational DBMS really be practical?

Navigational vs Relational Data

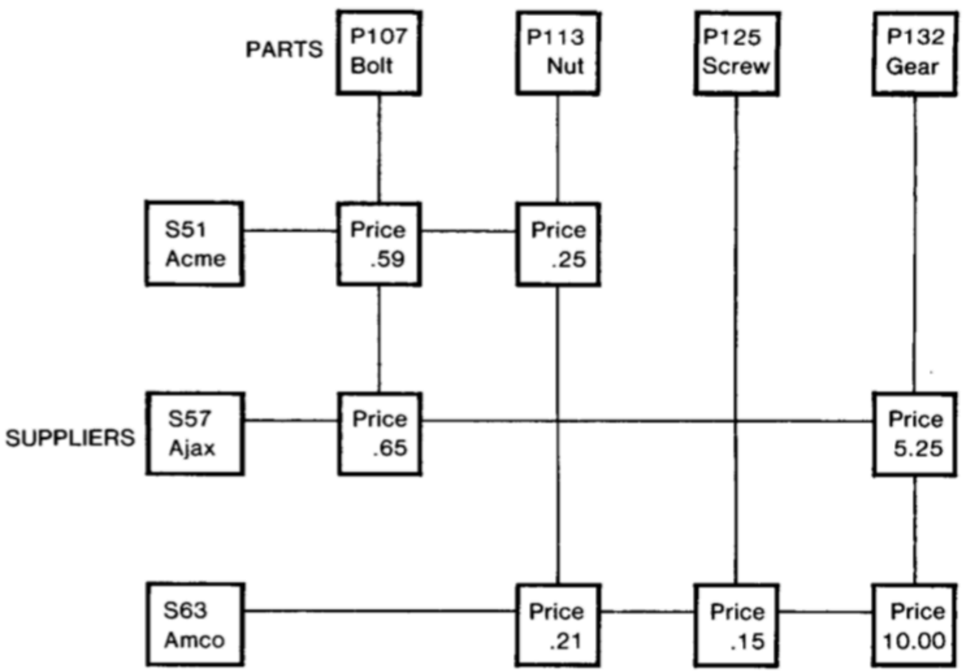


Fig. 1(a). A "Navigational" Database.

PARTS

PARTNO	NAME
P107	Bolt
P113	Nut
P125	Screw
P132	Gear

SUPPLIERS

SUPPNO	NAME
S51	Acme
S57	Ajax
S63	Amco

PRICES

PARTNO	SUPPNO	PRICE
P107	S51	.59
P107	S57	.65
P113	S51	.25
P113	S63	.21
P125	S63	.15
P132	S57	5.25
P132	S63	10.00

Fig. 1(b). A Relational Database.

Why is the relational model more flexible?

Three Phases of Development

Why was System R built in 3 phases?

Storage in System R Phase 0

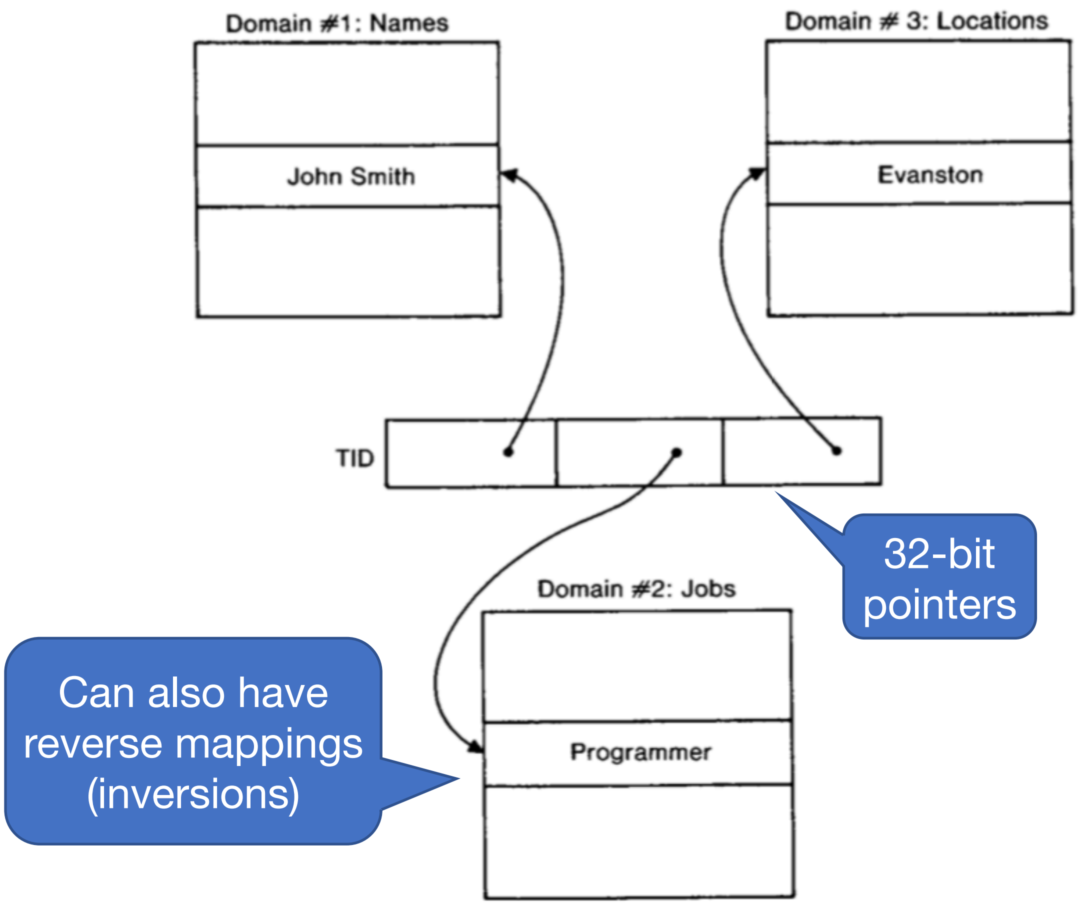


Fig. 2. XRM Storage Structure.

What was the issue with this design?

Too many I/Os:

- For each tuple, look up all its fields
- Use "inversions" to find TIDs with a given value for a field

Storage in System R Phase 1

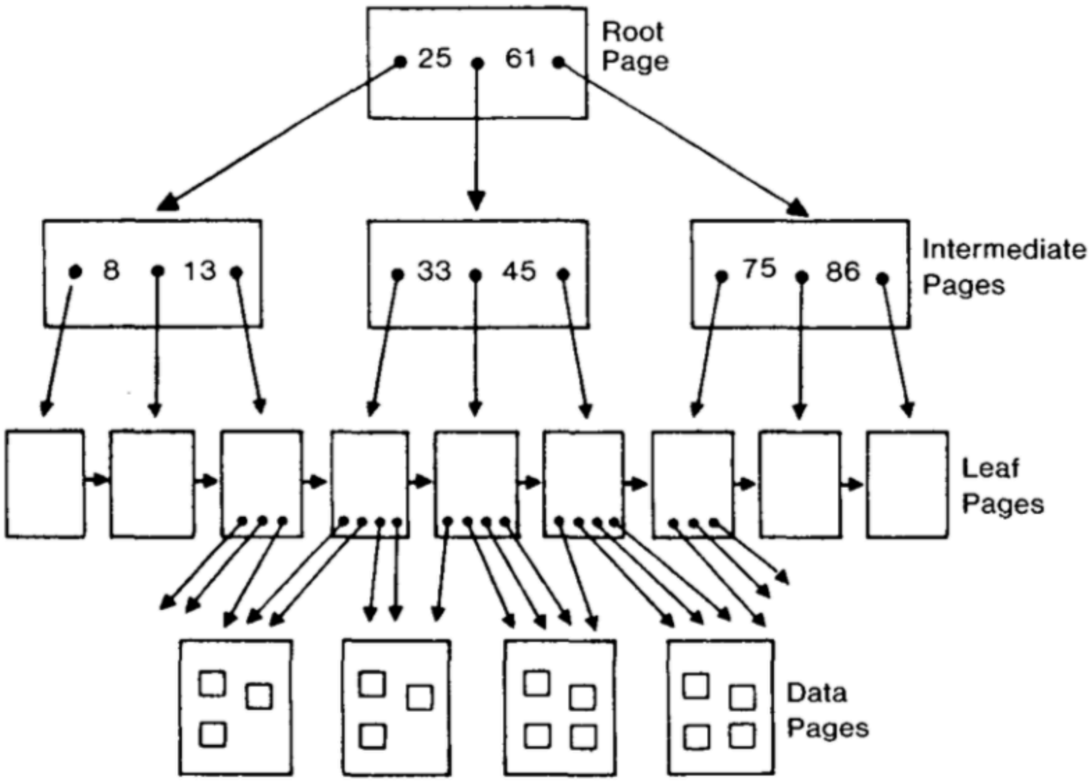


Fig. 6. A B-Tree Index.

B-tree nodes contain values of the column(s) indexed on

Data pages can contain all fields of the record

Give an example query that would be faster with B-Trees!

Query Optimizer

How did the System R optimizer change after Phase 0?

Query Compilation

Why did System R compile queries to assembly code?

How did it compile them?

Do databases still do that today?

Example 1:

```
SELECT  SUPPNO, PRICE
FROM    QUOTES
WHERE   PARTNO = '010002'
AND MINQ<=1000 AND MAXQ>=1000;
```

Operation	CPU time (msec on 168)	Number of I/Os
Parsing	13.3	0
Access Path Selection	40.0	9
Code Generation	10.1	0
Fetch answer set (per record)	1.5	0.7

Recovery

Goal: get the database into a consistent state after a failure

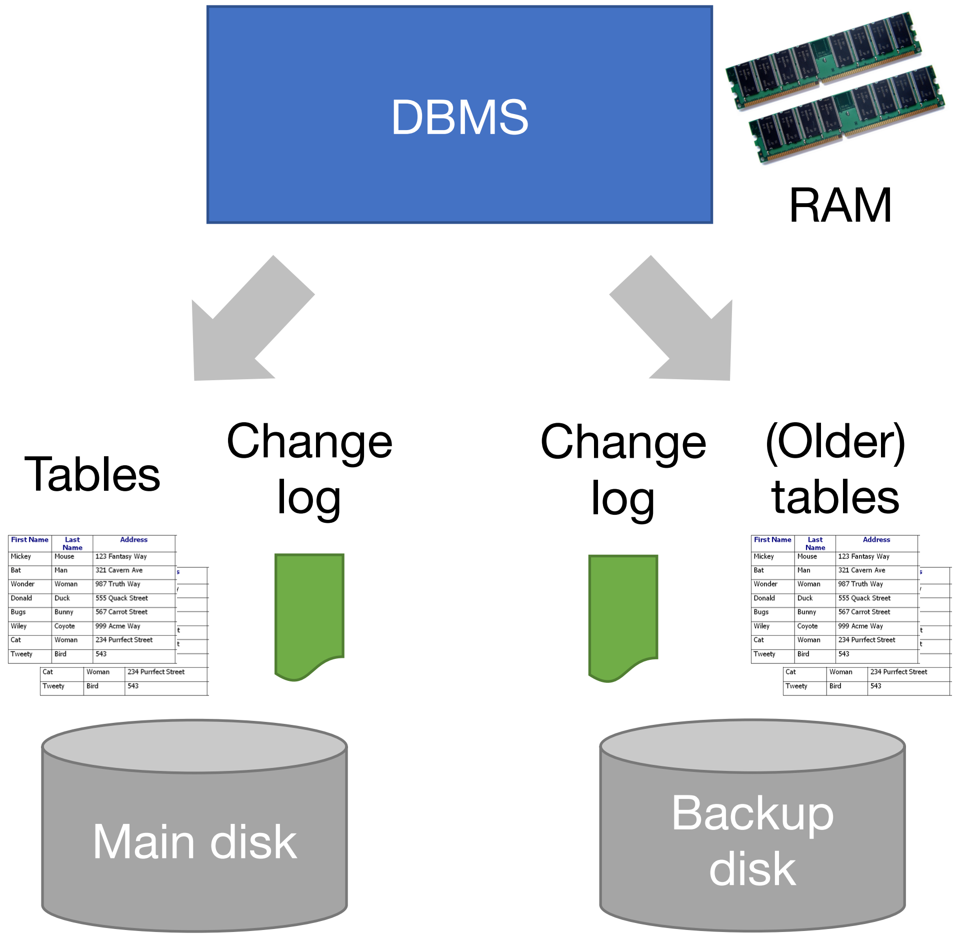
“A consistent state is defined as one in which the database does not reflect any updates made by transactions which did not complete successfully.”

Recovery

Three main types of failures:

- » Disk (storage media) failure
- » System crash
- » Transaction failure

Handling Storage Failure



Handling Transaction Failures

Just undo any changes they made, which we logged in the change log

Nobody else "saw" these changes due to System R's **locking mechanism**

Locking

The problem:

- » Different transactions are concurrently trying to read and update various data records
- » Each transaction wants to see a static view of the database (maybe lock whole DB)
- » For efficiency, we can't let them do that!

Locking and Isolation in System R

Locking:

» Started with “predicate locks” based on expressions: too expensive

» Moved to hierarchical locks: record/page/table, with read/write types and intentions

Isolation levels:

» Level 1: Transaction may read uncommitted data; successive reads to a record may return different values

» Level 2: Transaction may only read committed data, but successive reads can differ

» Level 3: Successive reads return same value

Most apps chose Level 3 since others weren’t much faster

Are There Alternatives to Locking for Concurrency?

Authorization

Goal: give some users access to just parts of the database

- » A manager can only see and update salaries of her employees
- » Analysts can see user IDs but not names
- » US users can't see data in Europe

Authorization

System R used view-based access control

» Define SQL views (queries) for what the user can see and grant access on those

```
CREATE VIEW canadian_customers AS
SELECT customer_name, email_address
FROM customers
WHERE country = “Canada”;
```

Elegant implementation: add the user’s SQL query on top of the view’s SQL query

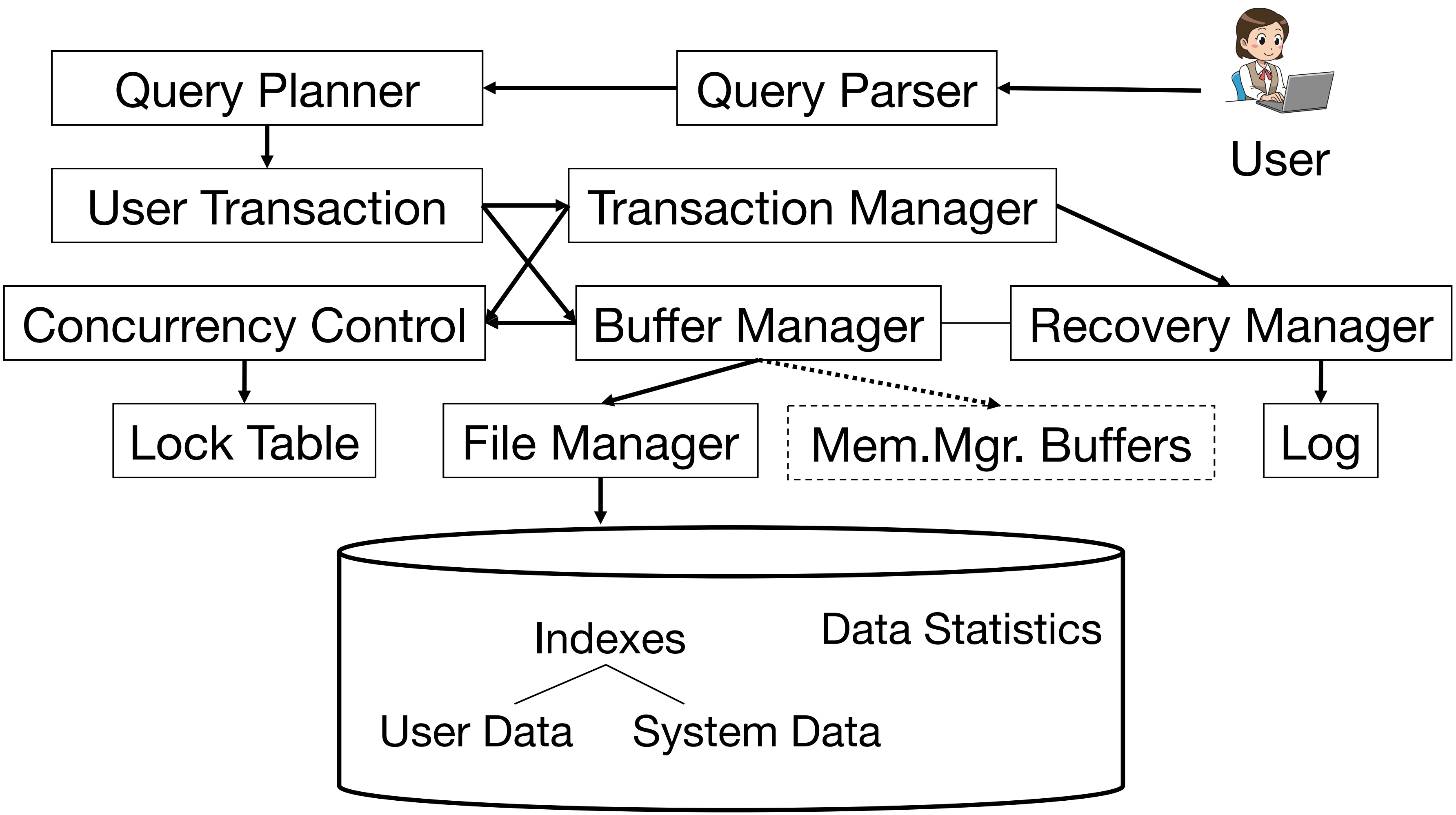
Outline

System R discussion

Relational DBMS architecture

Alternative architectures & tradeoffs

Typical RDBMS Architecture



Boundaries

Some of the components have clear boundaries and interfaces for modularity

» SQL language

» Query plan representation (relational algebra)

» Pages and buffers

Other components can interact closely

» Recovery + buffers + files + indexes

» Transactions + indexes & other data structures

» Data statistics + query optimizer

Differentiating by Workload

2 big classes of commercial RDBMS today

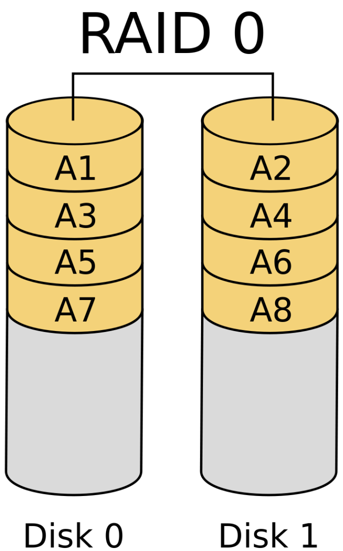
Transactional DBMS: focus on concurrent, small, low-latency transactions (e.g. MySQL, Postgres, Oracle, DB2) → real-time apps

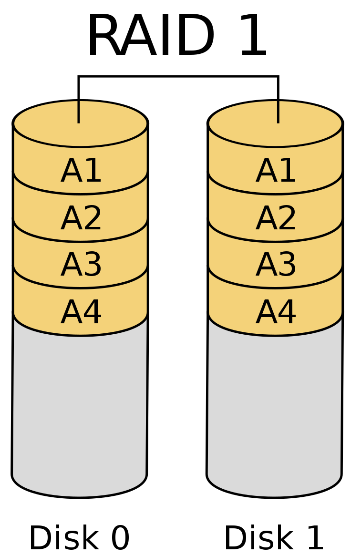
Analytical DBMS: focus on large, parallel but mostly read-only analytics (e.g. Teradata, Redshift, Vertica) → "data warehouses"

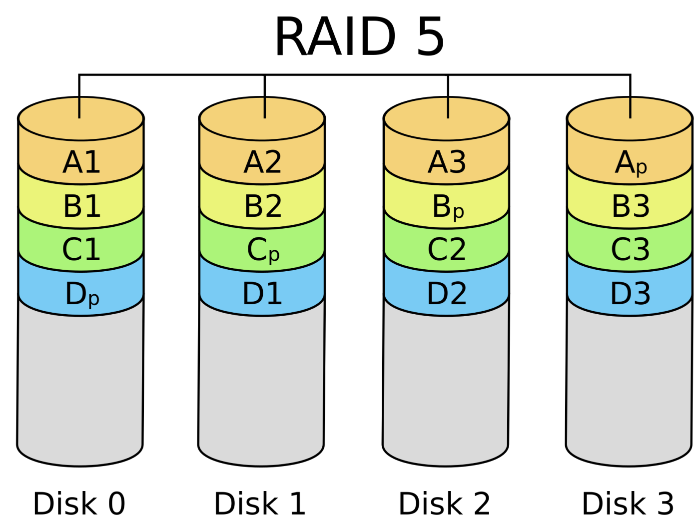
How To Design Components for Transactional vs Analytical DBMS?

Component	Transactional DBMS	Analytical DBMS
Data storage	B-trees, row oriented storage	Column-oriented storage
Locking	Fine-grained, very optimized	Coarse-grained (few writes)
Recovery	Log data writes, minimize latency	Log queries

Common RAID Levels







Striping across 2 disks: adds performance but not reliability

Mirroring across 2 disks: adds reliability but not performance (except for reads)

Striping + 1 parity disk: adds performance and reliability at lower storage cost

Image source: Wikipedia

Summary

Storage devices offer various tradeoffs in terms of latency, throughput and cost

In **all** cases, data layout and access pattern matter because random $\ll$ sequential access

Most systems will combine multiple devices

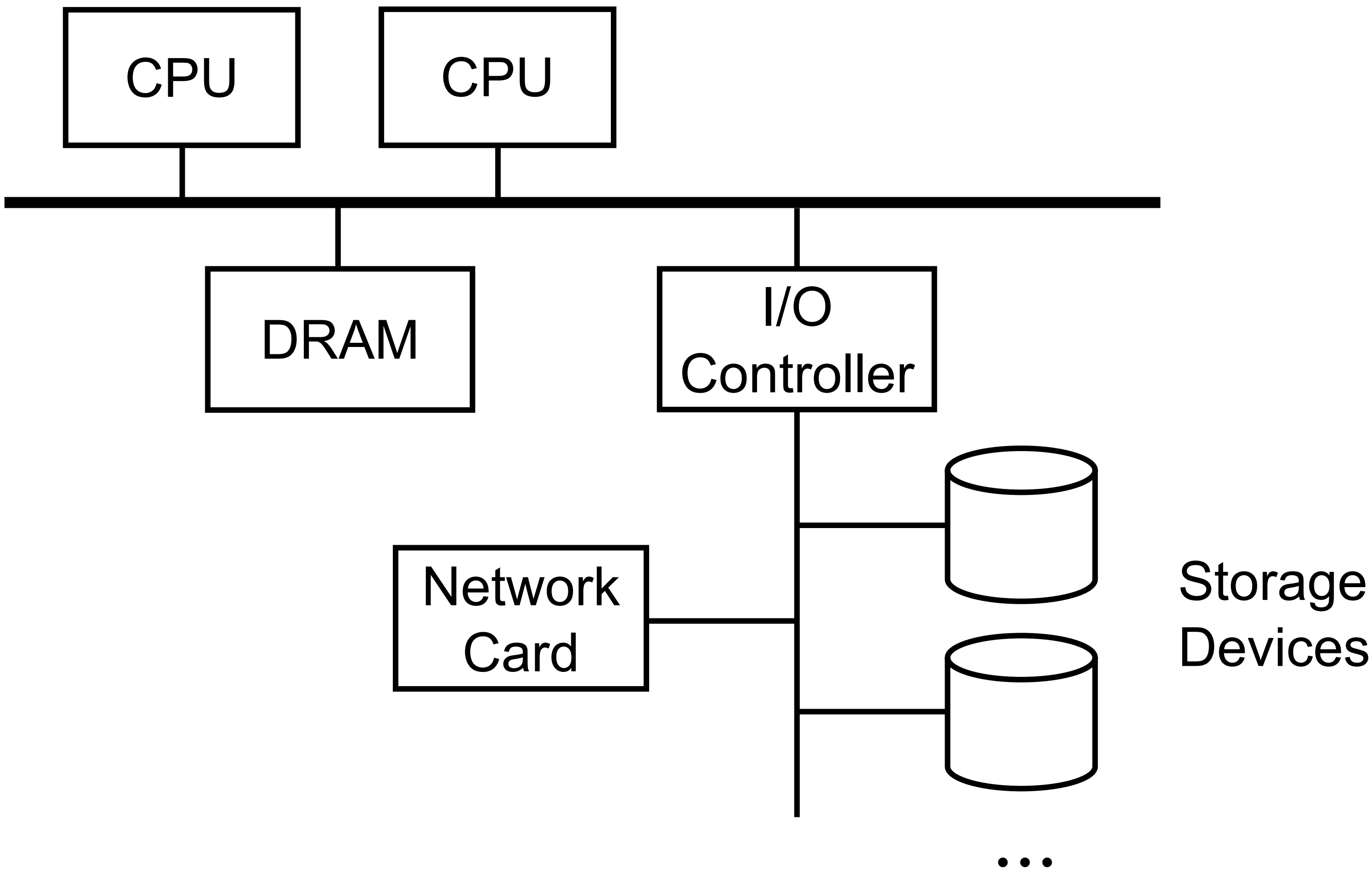
Outline

Relational DBMS architecture

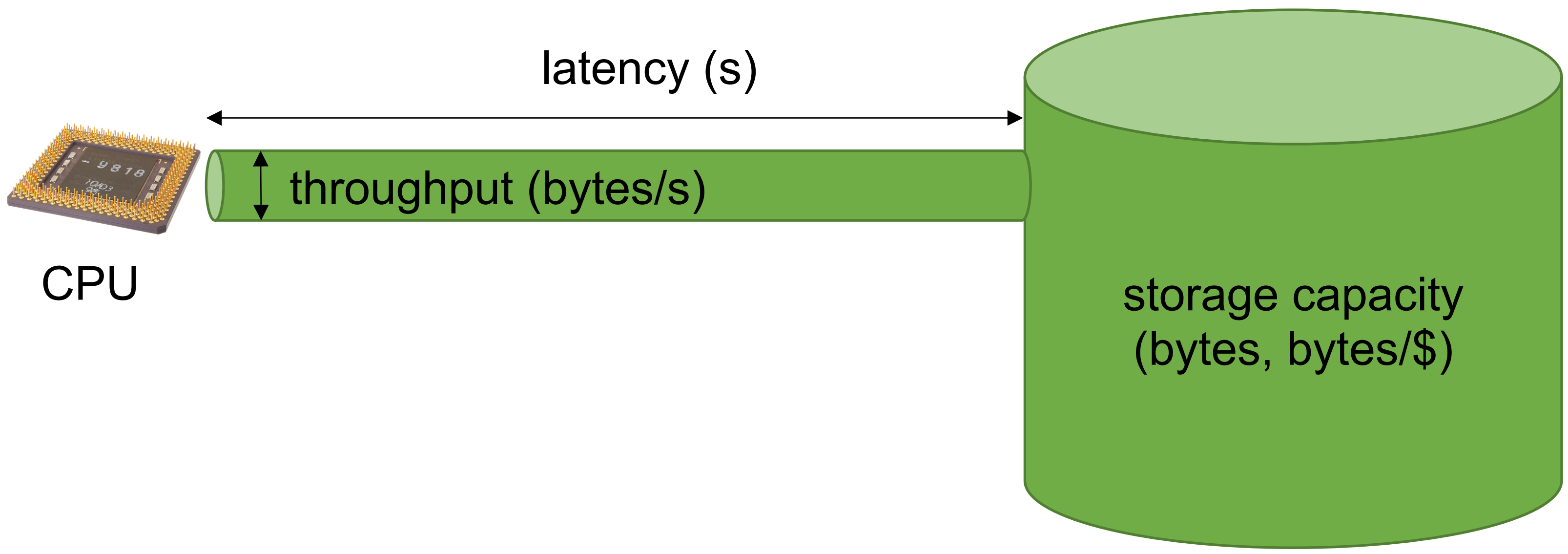
Alternative architectures & tradeoffs

Storage hardware

Typical Server



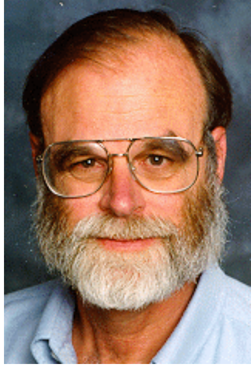
Storage Performance Metrics



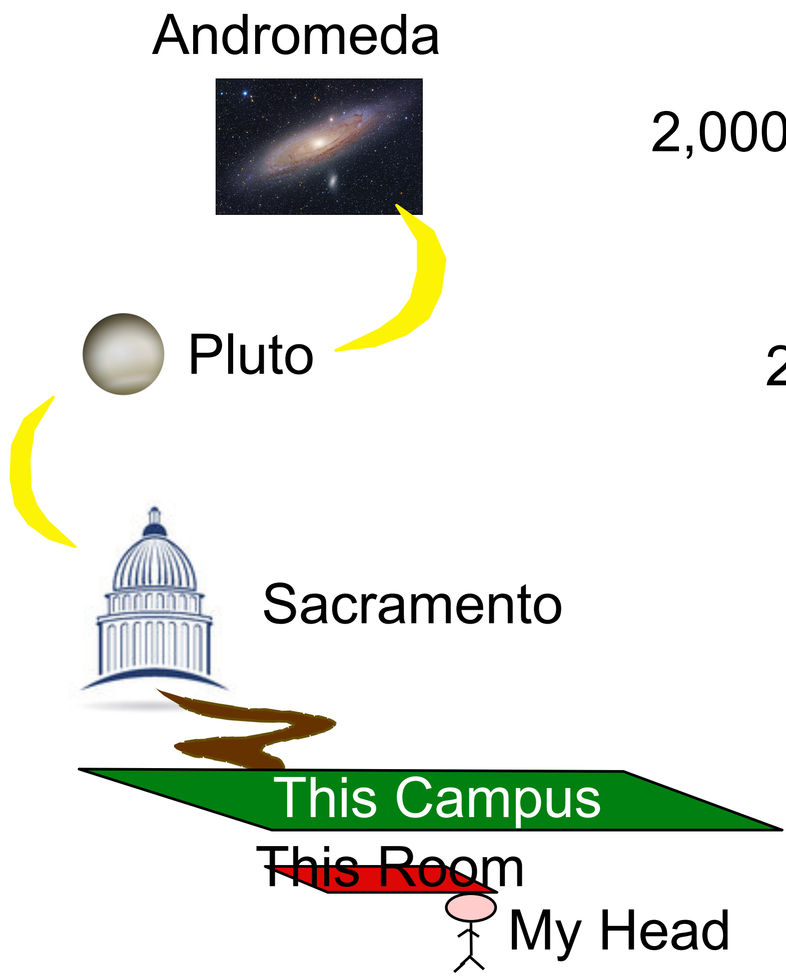
"Numbers Everyone Should Know" from Jeff Dean

L1 cache reference	0.5 ns	
Branch mispredict	5 ns	
L2 cache reference	7 ns	
Mutex lock/unlock	100 ns	
Main memory reference	100 ns	
Compress 1K bytes with Zippy	10,000 ns	0.01 ms
Send 1K bytes over 1 Gbps network	10,000 ns	0.01 ms
Read 1 MB sequentially from memory	250,000 ns	0.25 ms
Round trip within same datacenter	500,000 ns	0.5 ms
Disk seek	10,000,000 ns	10 ms
Read 1 MB sequentially from network	10,000,000 ns	10 ms
Read 1 MB sequentially from disk	30,000,000 ns	30 ms
Send packet CA->Netherlands->CA	150,000,000 ns	150 ms

Storage Latency



10^9	Tape /Optical Robot	2,000 Years
10^6	Disk	2 Years
150	Memory	2 hr
10	L2 Cache	10 min
2	L1 Cache	
1	Registers	1 min



Max Attainable Throughput

Varies significantly by device

- » 100 GB/s for RAM
- » 2 GB/s for NVMe SSD
- » 130 MB/s for hard disk

Assumes large reads (≫1 block)!

Storage Cost

$1000 at NewEgg today buys:

» 0.25 TB of RAM

» 9 TB of NVMe SSD

» 50 TB of magnetic disk

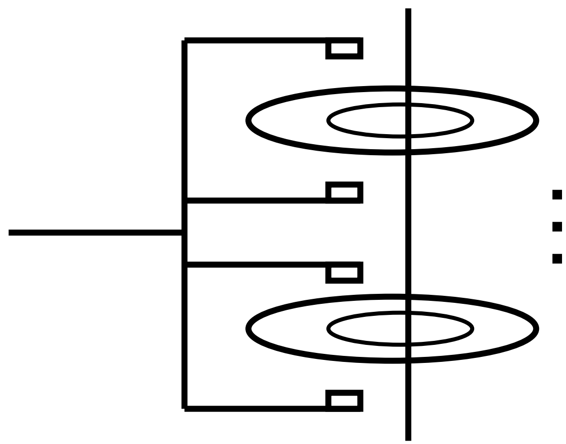
Hardware Trends over Time

Capacity/$ grows exponentially at a fast rate (e.g. double every 2 years)

Throughput grows at a slower rate (e.g. 5% per year), but new interconnects help

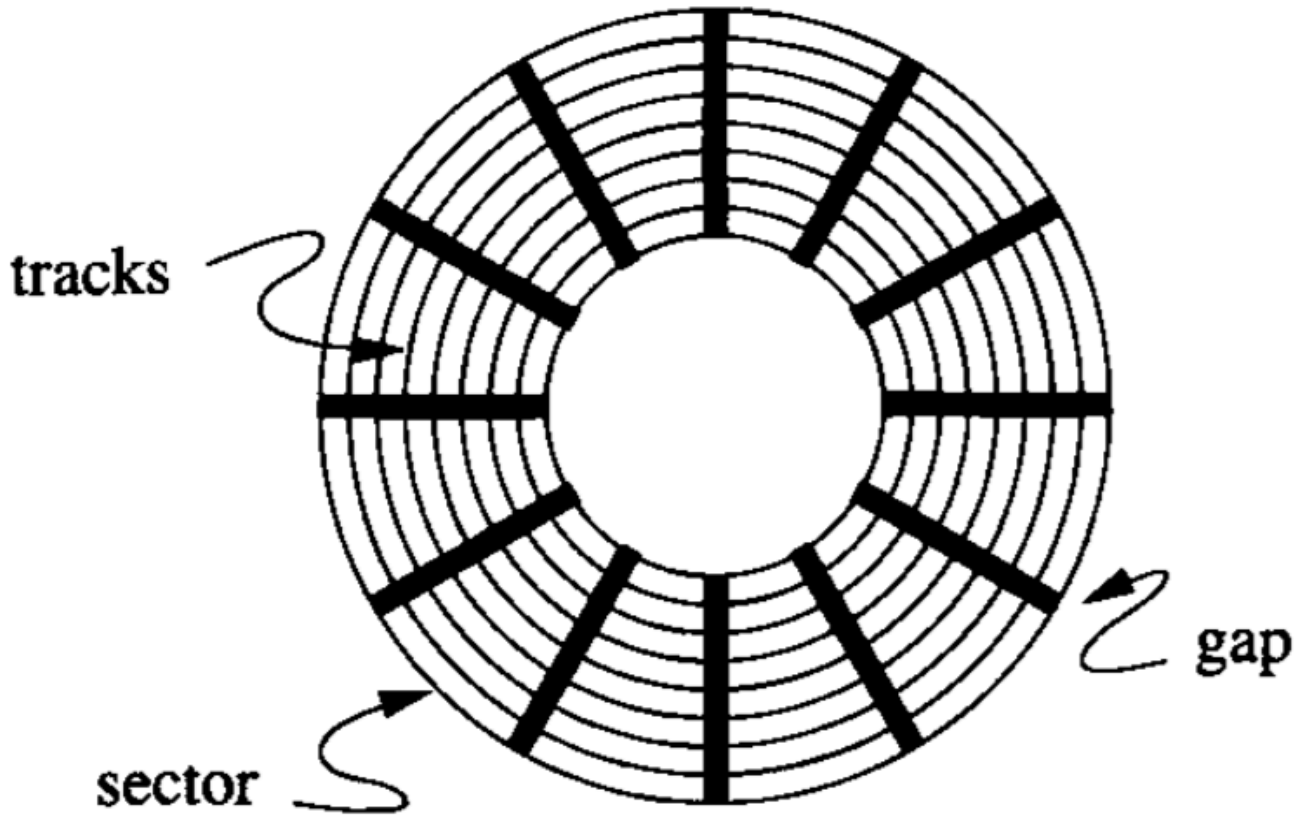
Latency does not improve much over time

Most Common Permanent Storage: Hard Disks



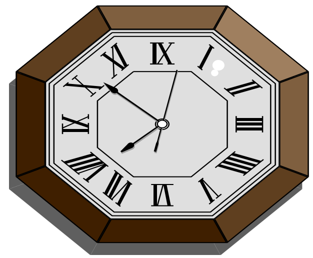
Terms: Platter, Head, Actuator
Cylinder, Track
Sector (physical),
Block (logical), Gap

Top View



Disk Access Time

I want
block X



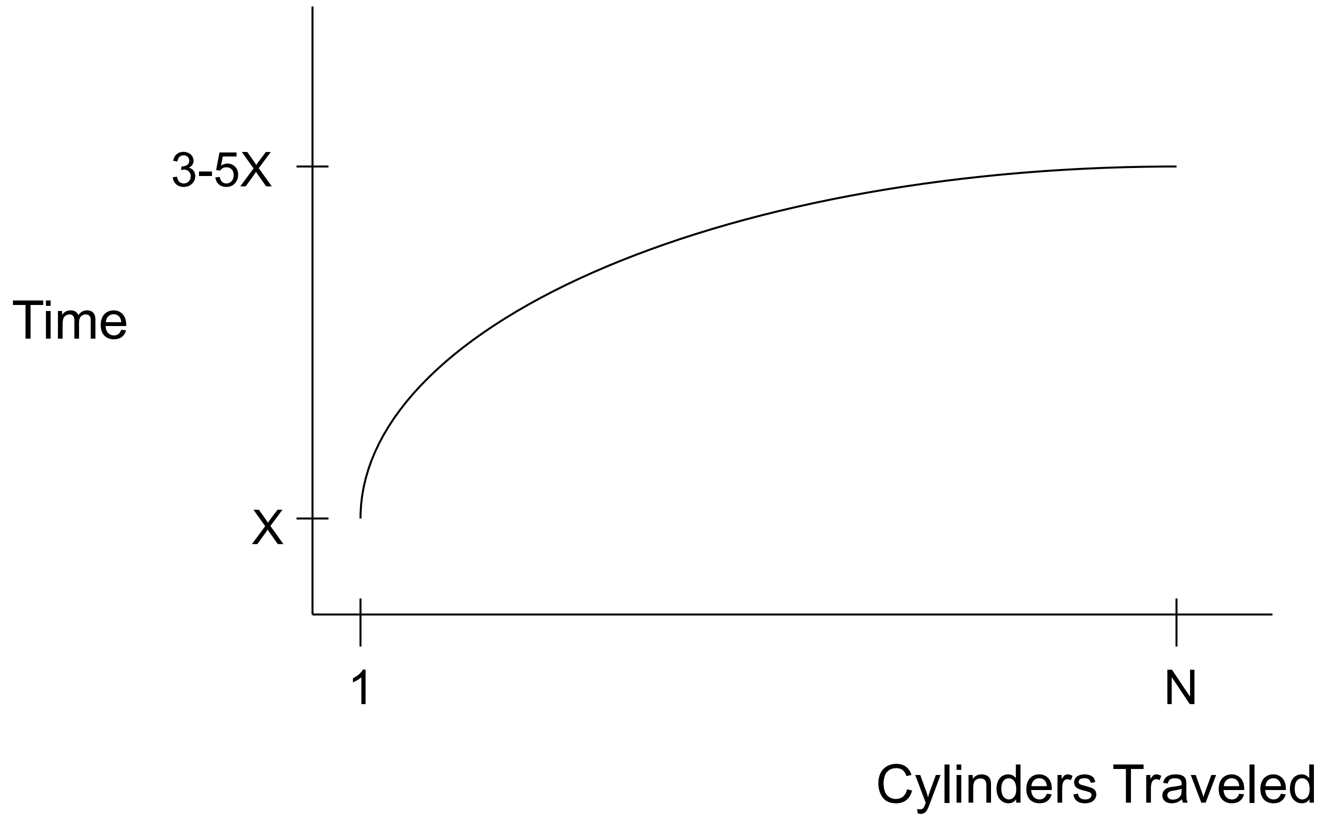
?

block x
in memory

Disk Access Time

Time = Seek Time +
Rotational Delay +
Transfer Time +
Other

Seek Time



Typical Seek Time

Ranges from

» 4 ms for high end drives

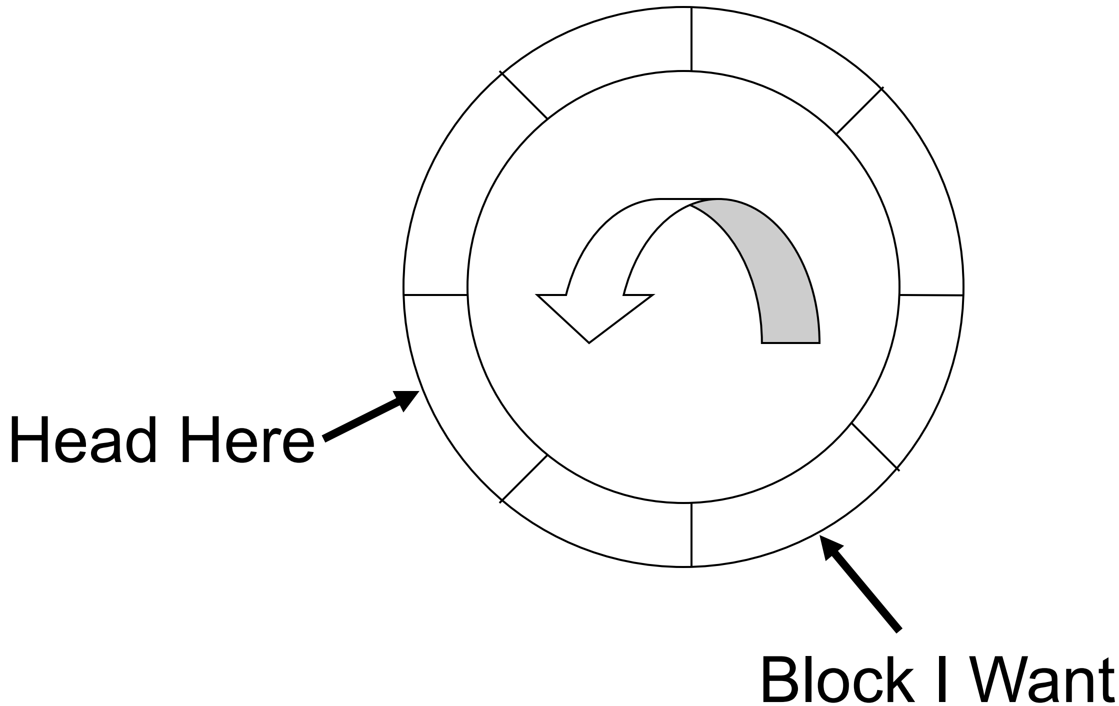
» 15 ms for mobile devices

In contrast, SSD access time ranges from

» 0.02 ms: NVMe

» 0.16 ms: SATA

Rotational Delay



Average Rotational Delay

R = 1/2 revolution R=0 for SSDs

Typical HDD figures

HDD Spindle [rpm]	Average rotational latency [ms]
4,200	7.14
5,400	5.56
7,200	4.17
10,000	3.00
15,000	2.00

Source: Wikipedia, "Hard disk drive performance characteristics"

Transfer Rate

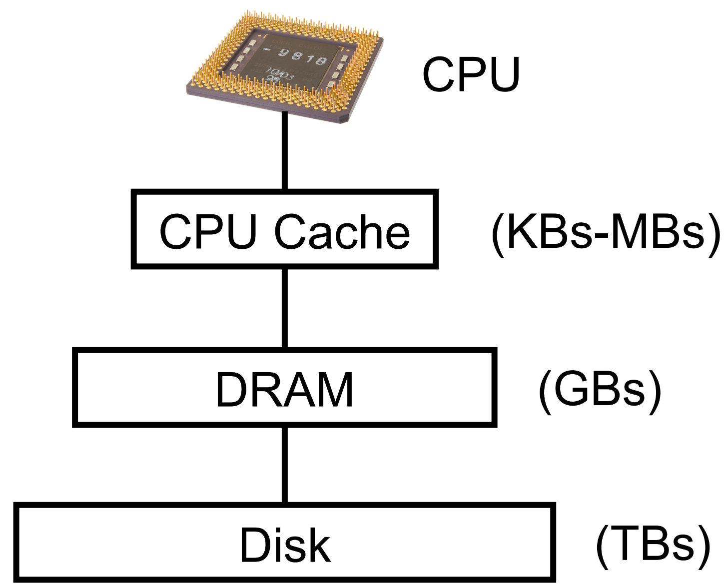
Transfer rate T is around 50-130 MB/s

Transfer time: size / T for contiguous read

Block size: usually 512-4096 bytes

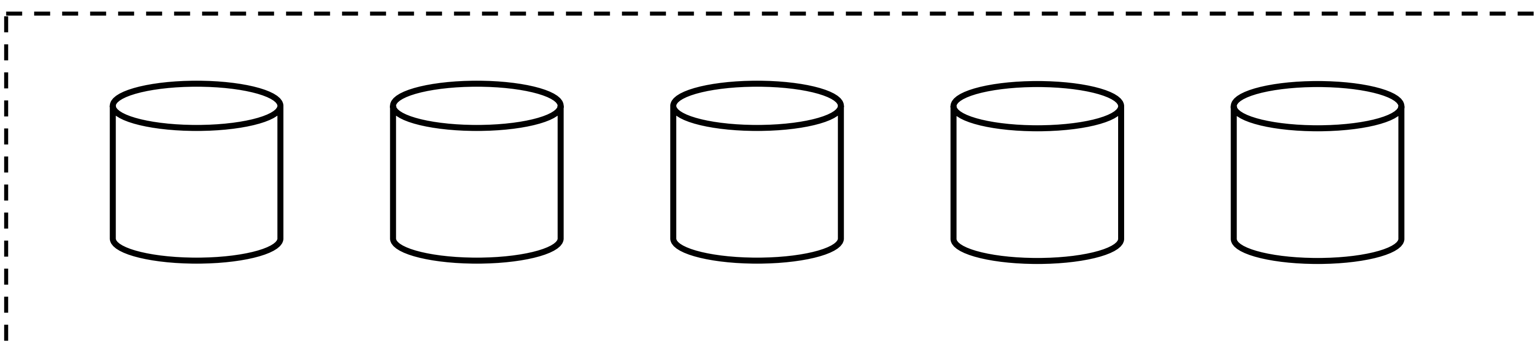
Storage Hierarchy

Typically want to **cache** frequently accessed data at a high level of the storage hierarchy to improve performance



Disk Arrays

Many flavors of "RAID": striping, mirroring, etc to increase **performance** and **reliability**



logically one disk