COSC 460 Lecture 2: Data Storage

Professor Michael Hay
Fall 2018

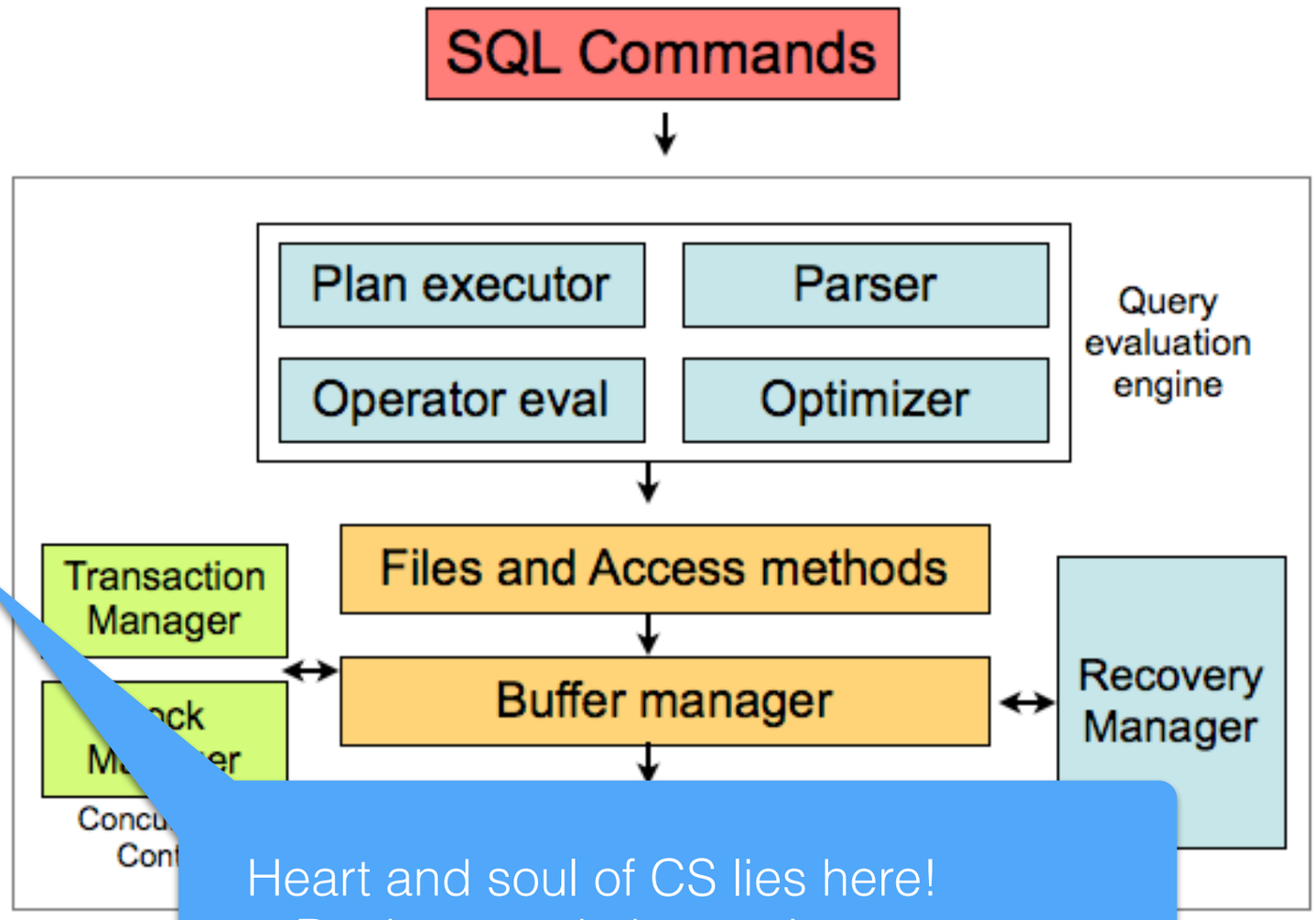
Credits: Slides adapted from Gehrke, Franklin, Widom, Miklau, Kot, and possibly others

Recap

- Relational model: data “stored” in tables
- Tables are a ***logical*** **description** only;
- Relational model does not say *anything* about actual ***physical*** storage
- Virtue of relational model is **physical data independence**: application programs work at logical level with tables and DBMS worries about details of mapping logical to physical storage
- Example of **abstraction**, a key idea in computer science.

Architecture of DBMS

Logical level



Messy stuff

Heart and soul of CS lies here!

- Design good abstractions
- Engineer efficient implementations so users of higher level abstraction layer live happy lives!

Physical level

Today

- Dive down into messy details of physical storage
- Today: disk, files, buffer manager
- Monday: file formats (exciting!)

Memory hierarchy

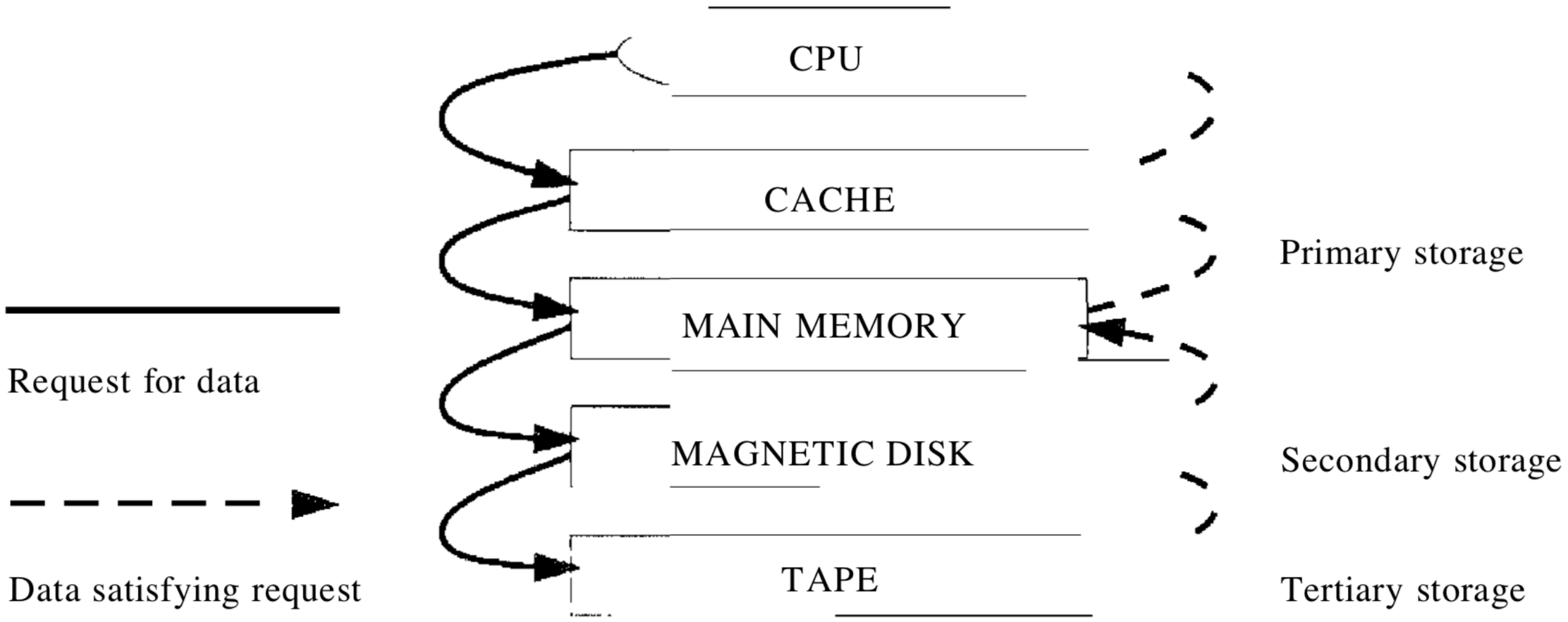
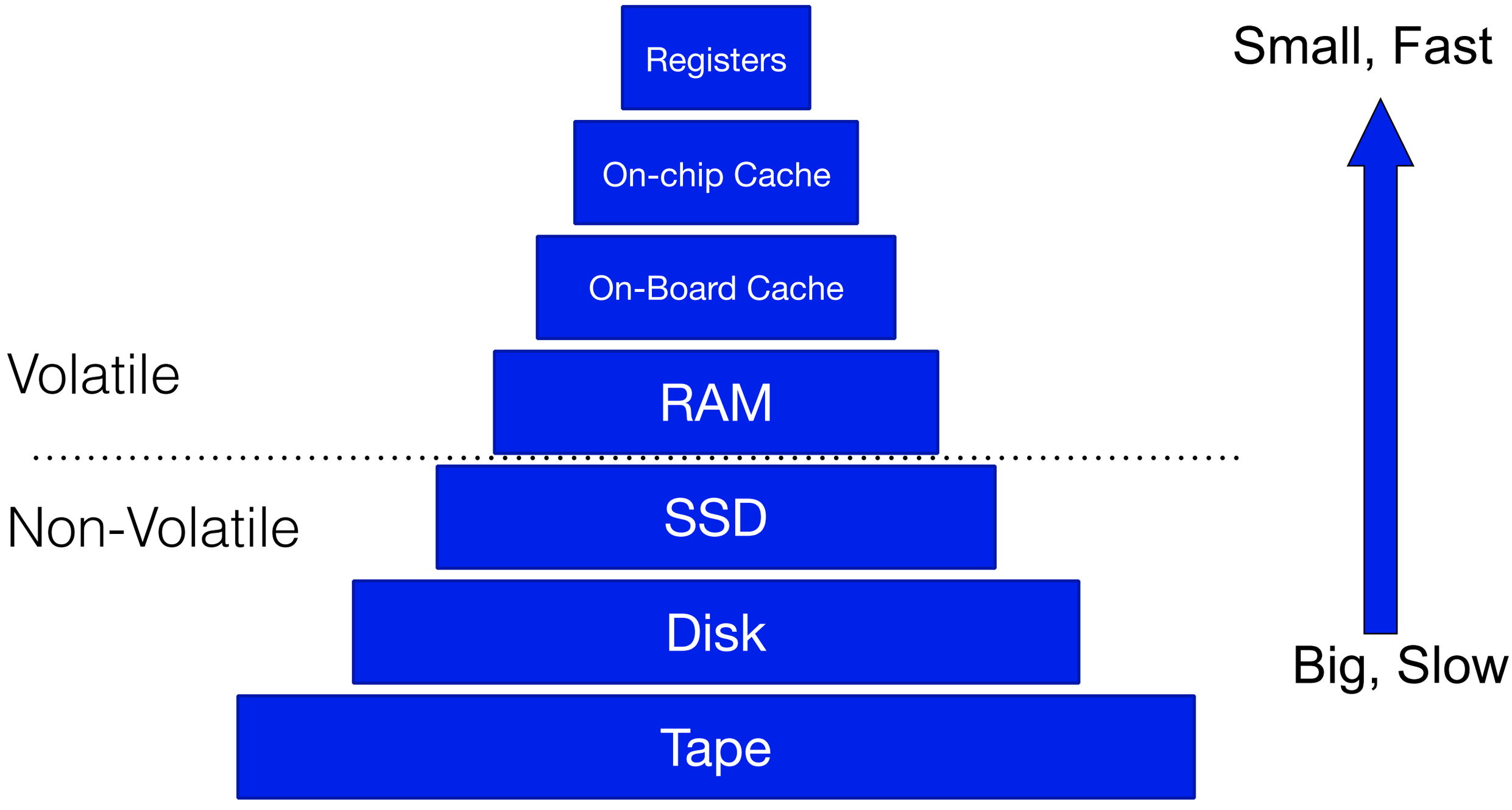


Fig. 9.1 from Cow book

Storage hierarchy



Storing Data

- Requirements of DBMS: ability to…
  - store ***large*** amounts of data,
  - in storage medium that is ***reliable***,
  - obtain ***fast access*** to data.
- Another key factor: storage media ***cost***

Access speed

- Random access*
  - Disk: 316 values/sec
  - SSD: 1924 values/sec
  - Memory: 36,700,000 values/sec

* Random is worst-case scenario for disks (more later)

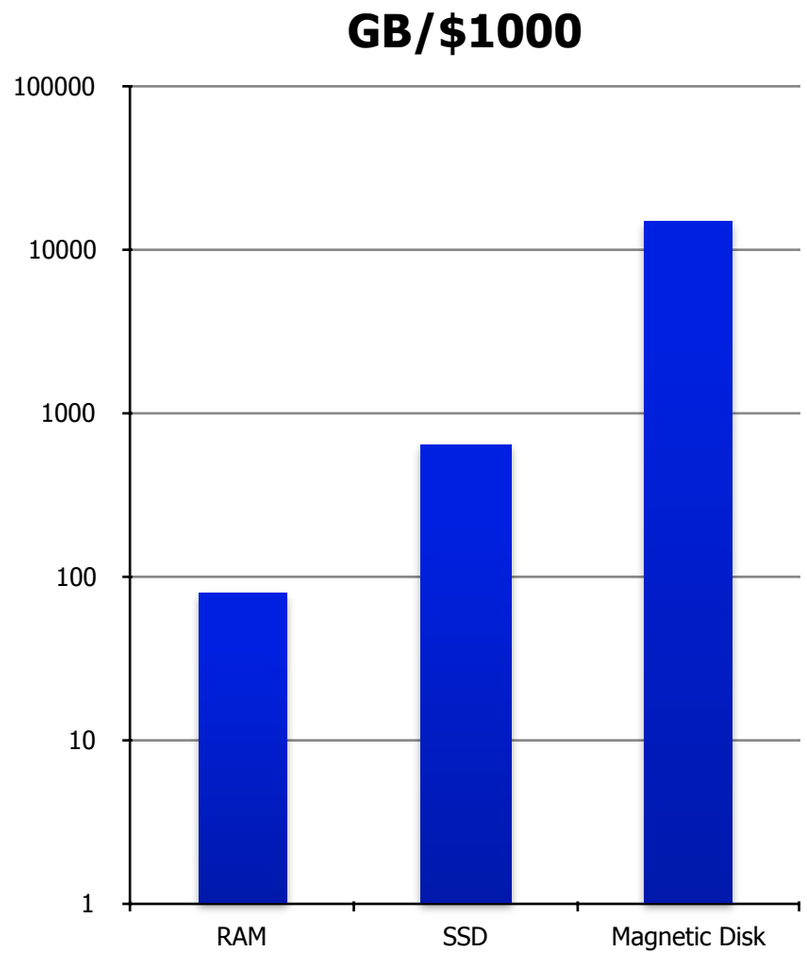
Reliability

- Disk: very reliable
- SDD: pretty reliable but some issues with wear over time (in write-intensive environments)
- RAM: volatile! when power goes out, so does data!

Cost

For $1000, PCConnection offers:

- ~0.08TB of RAM
- ~1TB of Solid State Disk
- ~19TB of Magnetic Disk



Current state

- ~~Most~~ Many DBMS running today store data on magnetic disks
- Rapid changes underway

Durable Write Cache in Flash Memory SSD for Relational and NoSQL Databases

Woon-Hak Kang
Sungkyunkwan University
Suwon, 440-746, Korea
woonagi319@skku.edu

Sang-Won Lee*
Sungkyunkwan University
Suwon, 440-746, Korea
swlee@skku.edu

Bongki Moon
Seoul National University
Seoul, 151-744, Korea
bkmoon@snu.ac.kr

Yang-Suk Kee
Samsung Semiconductor Inc.
Milpitas, USA, 95035
yangseok.ki@ssi.samsung.com

Moonwook Oh
Samsung Electronics
Hwasung, 445-701, Korea
mw.oh@samsung.com

ABSTRACT

In order to meet the stringent requirements of low latency as well as high throughput, web service providers with large data centers have been replacing magnetic disk drives with flash memory solid-state drives (SSDs). They commonly use relational and NoSQL database engines to manage OLTP workloads in the warehouse-scale computing environments. These modern database engines rely heavily on redundant writes and frequent cache flushes to guarantee the atomicity and durability of transactional updates. This has become a serious bottleneck of performance in both relational and NoSQL database engines.

This paper presents a new SSD prototype called *DuraSSD* equipped with *tantalum capacitors*. The tantalum capacitors make the device cache inside *DuraSSD* durable, and additional firmware features of *DuraSSD* take advantage of the durable cache to support the atomicity and durability of page writes. It is the first time that a flash memory SSD with durable cache has been used to achieve an order of magnitude improvement in transaction throughput without compromising the atomicity and durability. Considering that the simple capacitors increase the total cost of a
than one percent, *DuraSSD* clearly provides
means for transactional support. *DuraSSD*
to alleviate the problem of high tail latency
write stalls.

Categories and Subject Descriptors

H.2 [DATABASE MANAGEMENT]: Systems

General Terms

Design; Reliability; Performance

Keywords

Atomicity; Durability; SSD; Durable Cache

1. INTRODUCTION

In the era of warehouse-scale computing, a large-scale application runs on hundreds or thousands of servers equipped with their own storage and networking subsystems. When an application is made up of many tasks running in parallel, completion of the application is often delayed by a few tasks experiencing a disproportionate amount of latency, thus affecting negatively the overall utilization of computing resources as well as the quality of services. This latency problem will be aggravated further with an increasing number of parallel tasks, because the variance of latencies in parallel tasks is always amplified by the system scale.

This latency concern, known as *high tail latency*, poses serious challenges for online service providers operating computers and data centers [9]. Studies on
eased server side delays show that users re-
the speed of web services and a slower user
s long term behavior. For example, Ama-
100ms of latency cost them one percent in
e found an extra half second in search re-
sult generation dropped traffic 20 percent. Shopzilla found a five-second speed-up resulted in a 25 percent increase in page views, a 7 to 12 percent increase in revenue, a 50 per-

(2014)

Anti-Caching: A New Approach to Database Management System Architecture

Justin DeBrabant
Brown University
debrabant@cs.brown.edu

Andrew Pavlo
Brown University
pavlo@cs.brown.edu

Stephen Tu
MIT CSAIL
stephentu@csail.mit.edu

Michael Stonebraker
MIT CSAIL
stonebraker@csail.mit.edu

Stan Zdonik
Brown University
sbz@cs.brown.edu

ABSTRACT

The traditional wisdom for building disk-based relational database management systems (DBMS) is to organize data in heavily-encoded blocks stored on disk, with a main memory block cache. In order to improve performance given high disk latency, these systems use a multi-threaded architecture with dynamic record-level locking that allows multiple transactions to access the database at the same time. Previous research has shown that this results in substantial overhead for on-line transaction processing (OLTP) applications [15].

The next generation DBMSs seek to overcome these limitations with architecture based on main memory resident data. To overcome the restriction that all data fit in main memory, we propose a new technique, called *anti-caching*, where cold data is moved to disk in a transactionally-safe manner as the database grows in size. Because data initially resides in memory, an anti-caching architecture reverses the traditional storage hierarchy of disk-based systems. Main memory is now the primary storage device.

We implemented a prototype of our anti-caching proposal in a high-performance, main memory OLTP DBMS and performed a series of experiments across a range of database sizes, workload skews, and read/write mixes. We compared its performance with an open-source, disk-based DBMS optionally fronted by a distributed main memory cache. Our results show that for higher skewed workloads the anti-caching architecture has a pe
tage over either of the other architectures tested
data size 8× larger than memory.

1. INTRODUCTION

Historically, the internal architecture of DBM
icated on the storage and management of data in heavily-encoded disk blocks. In most systems, there is a header at the beginning of each disk block to facilitate certain operations in the system. For

DBMSs invariably maintain a buffer pool of blocks in main memory for faster access. When an executing query attempts to read a disk block, the DBMS first checks to see whether the block already exists in this buffer pool. If not, a block is evicted to make room for the needed one. There is substantial overhead to managing the buffer pool, since blocks have to be pinned in main memory and the system must maintain an eviction order policy (e.g., least recently used). As noted in [15], when all data fits in main memory, the cost of maintaining a buffer pool is nearly one-third of all the CPU cycles used by the DBMS.

The expense of managing disk-resident data has fostered a class of new DBMSs that put the entire database in main memory and thus have no buffer pool [11]. TimesTen was an early proponent of this approach [31], and more recent examples include H-Store [2, 18], MemSQL [3], and RAMCloud [25]. H-Store (and its commercial version VoltDB [4]) performs significantly better than disk-based DBMSs on standard OLTP benchmarks [29] because of this main memory orientation, as well as from avoiding the overhead of concurrency control and heavy-weight data logging [22].

The fundamental problem with main memory DBMSs, however, is that this improved performance is only achievable when the database is smaller than the amount of physical memory available in the system. If the database does not fit in memory, then the operating
o page virtual memory, and main memory ac-
page faults. Because page faults are transparent
case the main memory DBMS, the execution of
led while the page is fetched from disk. This is a
n in a DBMS, like H-Store, that executes transac-
out the use of heavyweight locking and latching.
ll main memory DBMSs warn users not to exceed the amount of real memory [5]. If memory is exceeded (or if it might be at some point in the future), then a user must either (1) provision new hardware and migrate their database to a larger

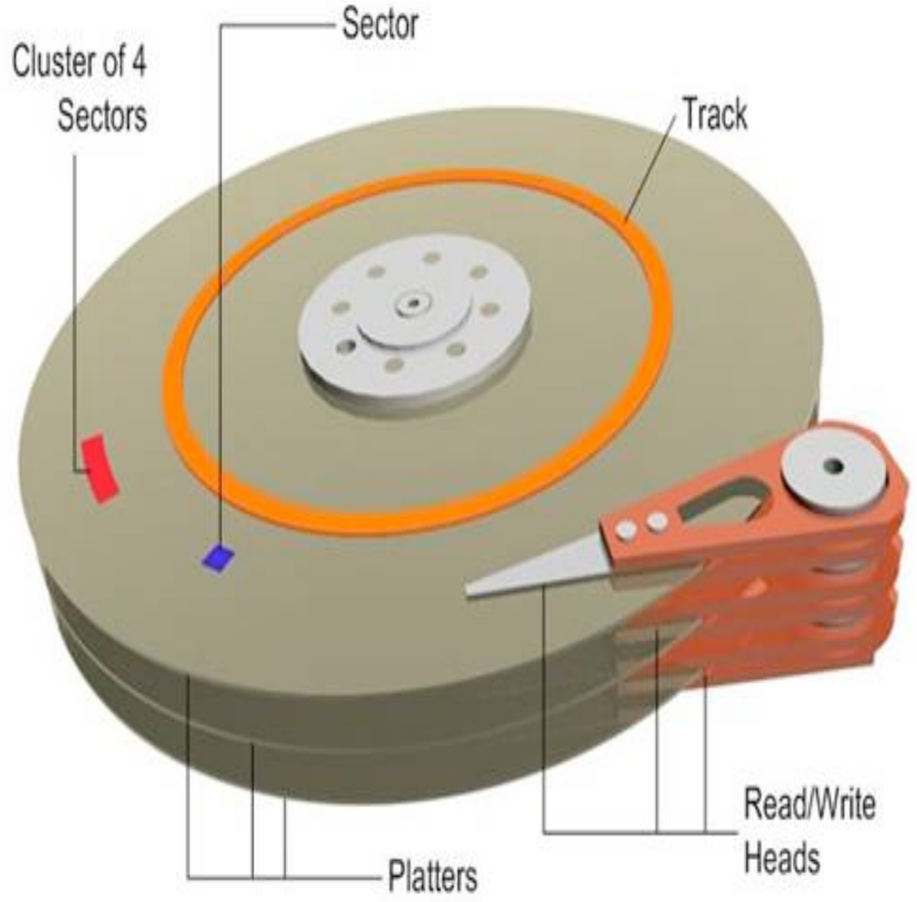
(2013)

This course

- Focus on simplified model: memory and disk
- Dominant cost is **I/O**
- Why what you learn will endure…
  - New technologies borrow lessons learned from previous technologies
  - There will likely always be some form of memory hierarchy: fast but volatile, slow but stable (this is true even for today's main memory DBs!)
  - Study *design process* of DBMS:
    - Make modeling assumptions
    - Design algorithms under those assumptions

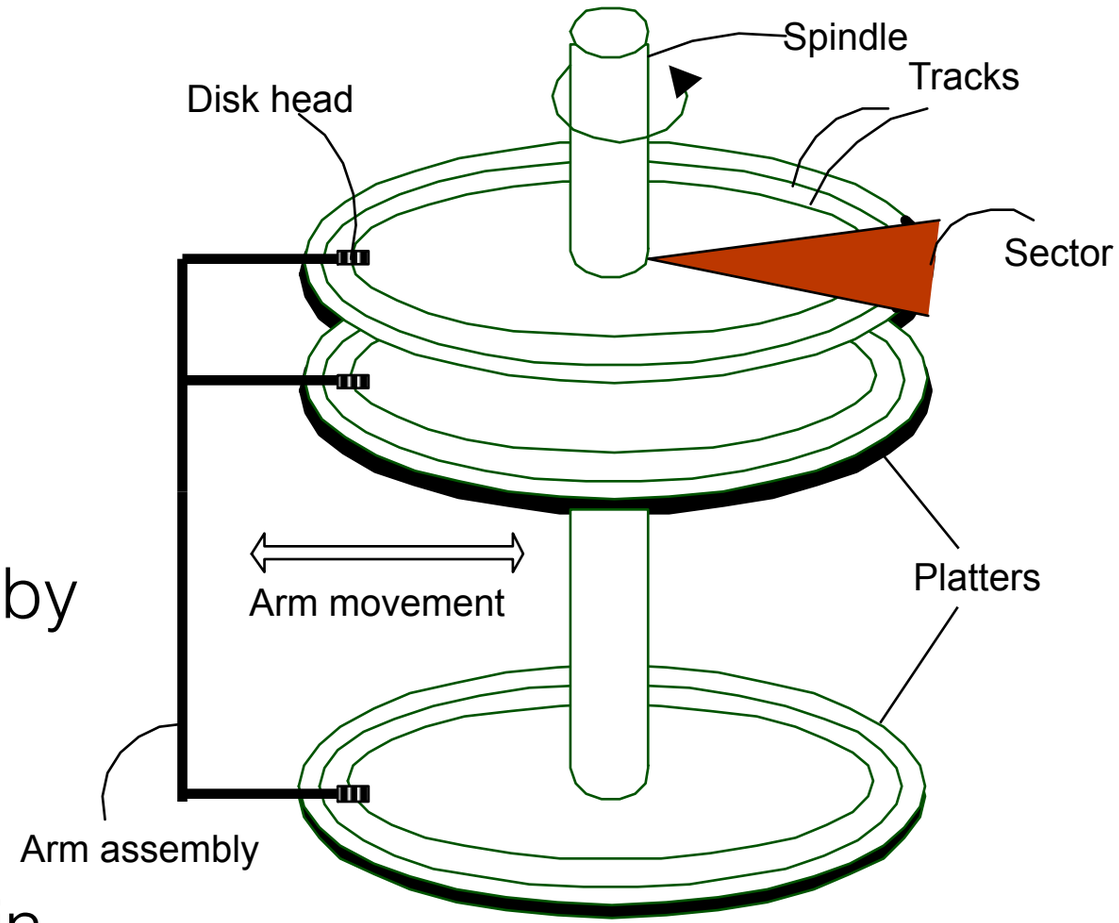
Anatomy of a disk

- Platters spin
- Arm assembly moved in/out to position a head on desired track.
- Tracks under heads make a **cylinder** (imaginary!)
- Only one head reads/writes at a time
- Block size is multiple of sector size (which is fixed)



Block layout

- Standard block size: 4K
- Where is “next” block?
  - blocks on same track, followed by
  - blocks on same cylinder, followed by
  - blocks on adjacent cylinder
- **Sequential access**: reading blocks in order according to notion of “next”



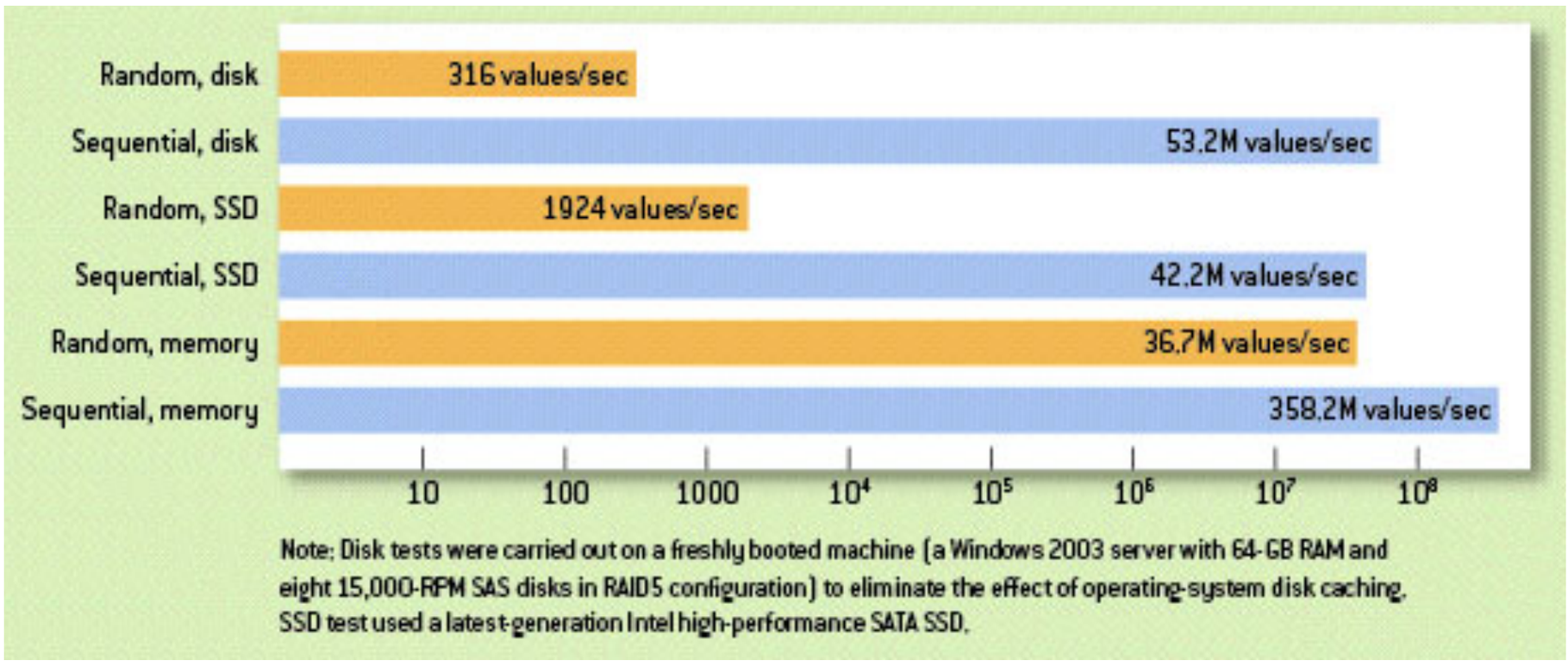
Accessing a Disk Page

- Time to access (read/write) a disk block:
  - **seek time** (moving arms to position disk head on track)
  - **rotational delay** (waiting for block to rotate under head)
  - **transfer time** (actually moving data to/from disk surface)

- Seek time and rotational delay dominate.
  - Seek time varies from about 1 to 20msec
  - Rotational delay varies from 0 to 10msec
  - Transfer rate is about 1msec per 4KB page
- Key to lower I/O cost: ***reduce seek/rotation delays!***
- (Aside: if disk is shared, wait time can be a big factor too.)

Retrieval rates

- Disk: sequential access is 5 orders of magnitude faster than random!



- Sequential access reasonably high throughput (compared to SSD and RAM)

From A. Jacobs, “The Pathologies of Big Data”, ACM Queue Magazine, July 2009

Recap

- Memory: fast but volatile (and expensive!)
- Disk: slow but stable (and cheap!)
- Disk: sequential access *much* faster than random access (why?)
- DBMS tries to minimize I/O cost

pollev.com/cosc460

Poll

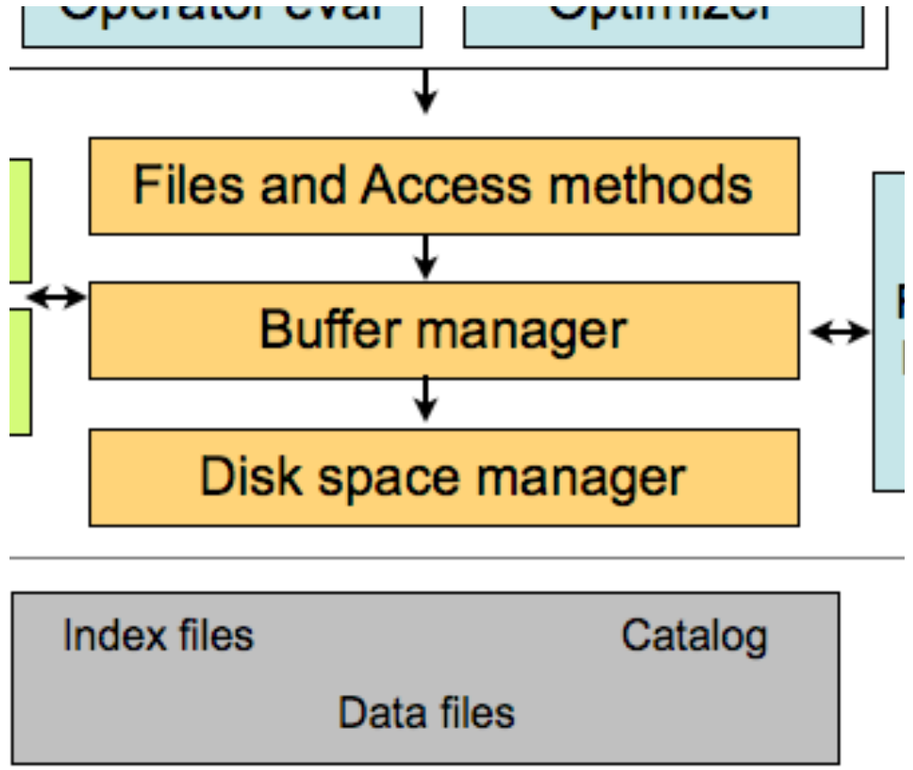
Requesting data from disks can be slow. What is a technique that can be used to improve access speed?

1) Caching
2) Pre-fetching
3) Organize "related" data sequentially on disk
4) All of the above

Instructions: *I will give you 1-2 minutes to think on your own.*
Vote 1.
Then you will discuss w/ neighbor (1 min).
Vote 2.
Then we'll discuss as class.

Architecture

- File of Records
- Buffer Manager
- Disk space manager
- OS Filesystem
- Disk



(details shown on board)

pollev.com/cosc460

Poll

Which layer in DBMS architecture provides physical data independence? If none do, choose the layer that comes closest.

1) OS Filesystem
2) Disk Manager
3) Buffer Manager
4) File of Records

Instructions: *I will give you 1-2 minutes to think on your own.*
Vote 1.
Then you will discuss w/ neighbor (1 min).
Vote 2.
Then we'll discuss as class.

Buffer Manager

(details shown on board)

Some terminology

- **Disk Page** – the unit of transfer between the disk and memory
  - Typically set as a config parameter for the DBMS.
  - Typical value between 4 KBytes to 32 KBytes.
- **Frame** – a unit of memory
  - Typically the same size as the Disk Page Size
- **Buffer Pool** – a collection of frames used by the DBMS to temporarily keep data for use by the query processor.
  - Note: sometime use the term “buffer” and “frame” synonymously.

Question

Suppose we did not maintain dirty bit and always assumed the page was dirty. This would require modifying the algorithm. The result would be

1) slower
2) more prone to failure
3) both A and B
4) none of the above

Instructions: *I will give you 1-2 minutes to think on your own.*
Vote 1.
Then you will discuss w/ neighbor (1 min).
Vote 2.
Then we'll discuss as class.

Replacement Policies

(details shown on board)