MSiA-413 Introduction to Databases and Information Retrieval

Lecture 9
Unique Keys, Storage Hierarchy Basics

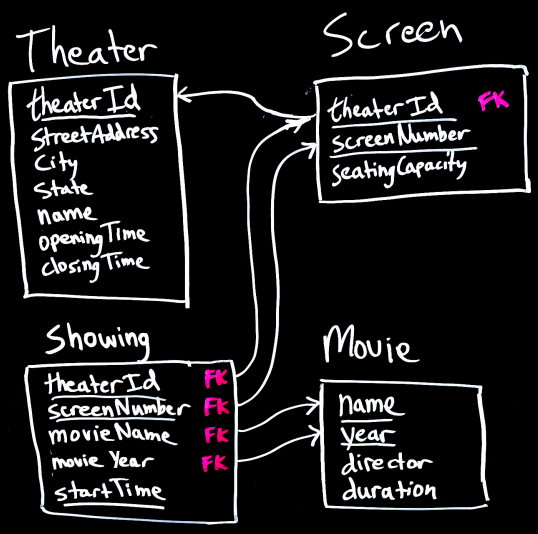
Instructor: Nikos Hardavellas

Slides adapted from Steve Tarzia

Last Lecture

- INNER JOINs
- JOINs with multiple matches
- JOINs on the same table multiple times

Movie Theater

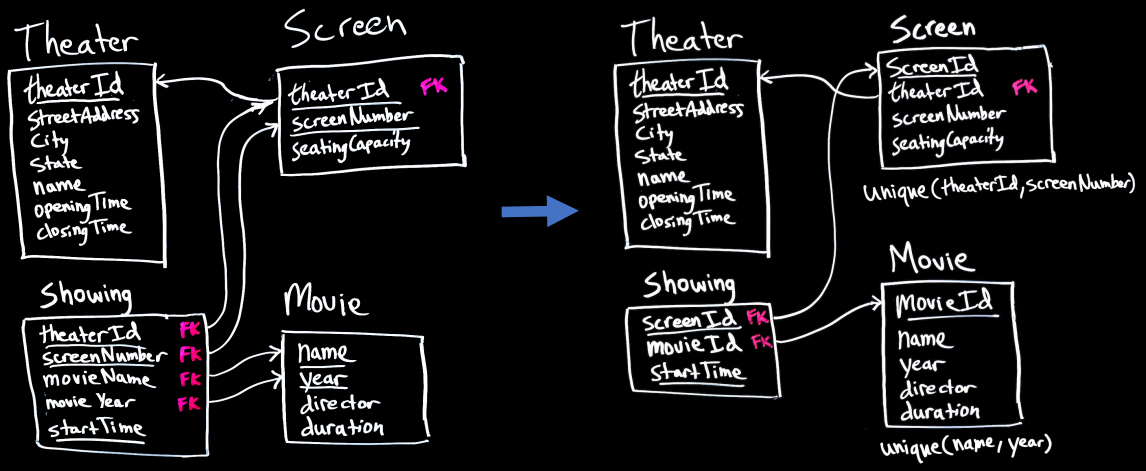


Composite Primary Keys

- Primary Keys uniquely identify rows
  - Used as ***indexes*** to find a row of interest
  - Prevent duplication
- Often we need more than one column to uniquely identify rows
  - e.g., a Screen is uniquely identified by "theaterId" *and* "screenNumber."
  - "theaterId" alone cannot be a primary key because it is OK for multiple screens to exist at the same theater, as long as they have different "screenNumber"
  - "screenNumber" alone cannot be a primary key because different theaters can use the same screen numbers (1, 2, 3 …)
- However, composite primary keys make parent-child relationships messy

Adding "ScreenId" and "MovieID" simplifies the schema

Showing table becomes smaller and JOINs are simpler



Non-primary/Unique Keys

- Common to create a meaningless "ID" column for primary key @ parent, then add a non-primary composite key to enforce the integrity constraint
- Consider the movie theater example:
  - "movieId" is meaningless
    - ...but it is a convenient way for other tables to refer to movies
  - add a ***unique key*** on (name, year) to prevent two instances of the same movie
  - "Showing" table can have just a single column "movieId" as a foreign key
    - ...instead of two columns (name, year)
- Example:

```
CREATE TABLE Movie
(MovieId PRIMARY KEY AUTOINCREMENT,
 name NVARCHAR(100),
 year DATETIME,
 UNIQUE(name, year) ON CONFLICT ABORT);
```

Part 2: Storage Hierarchy Basics

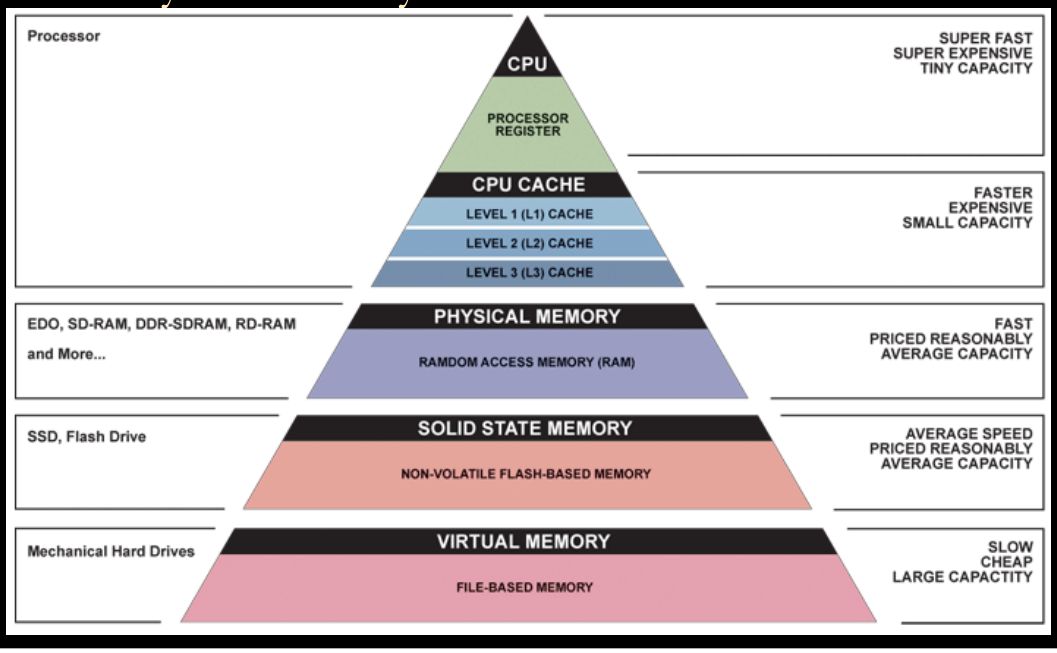
Computers have a hierarchy of storage

Larger, but slower

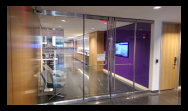
delay		*capacity*
300ps	CPU Registers	1 kB (kilobyte)
5ns	CPU Caches (L2)	16 MB
50ns	Random Access Memory (RAM)	16 GB
100µs	Flash Storage (SSD)	1 TB
5ms	Magnetic Disk	8 TB

- Disk is about *ten billion* times larger than registers, but is about *ten million* times slower (latency)
- Goal is to work as much as possible in the top levels
- Large, rarely-needed data is stored at the bottom level

Memory Hierarchy



Jim Gray's Storage Analogy: How far away is the data?

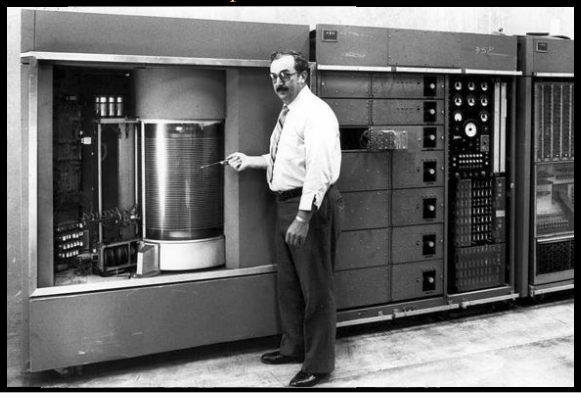
10^{10}	Tape		Andromeda	2,000 Years
10^{7}	Disk		Pluto	2 Years
100	Memory (DRAM)		Milwaukee	1.5 hr
10	On board cache		On campus	10 min
1	On chip cache		This building	2-4 min
1	Registers		My head	1 min

Disk storage

- Workhorse storage devices
  - 100-1,000x GB
  - ms to read (100,000x longer than DRAM, 10,000,000x longer than cache)

IBM 350 Disk Storage Unit
Announced Sep. 4, 1956

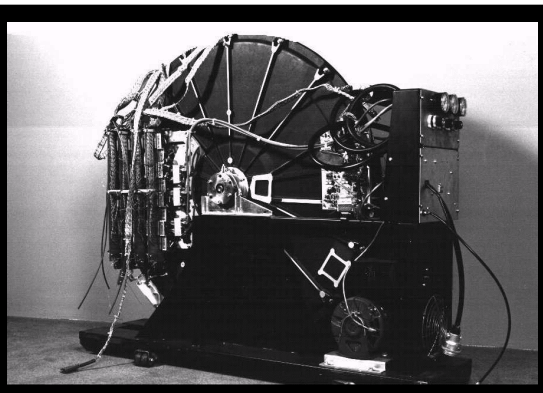
Size?
5MB



WD Red 10TB NAS,
2018

IBM Microdrive, 1999
340 MB

Disks & Files



- DBMS stores information on disks
  - In an electronic world, magnetic disks are a mechanical anachronism!
- This has major implications for DBMS design!
  - READ: transfer data from disk to main memory (DRAM)
  - WRITE: transfer data from RAM to disk
  - Both are high-cost operations, relative to in-memory operations, so must be planned carefully!

Why Not Store It All in Main Memory?

- Too expensive
  - High-end Databases today fall into the Petabyte range
  - ~ 60% of the cost of a production system is in the disks
- Main memory is volatile
  - We want data to be saved between runs (obviously!)
- Main-memory database systems do exist
  - Smaller size, performance optimized
  - Volatility is ok for some applications
- Typical DBMS hierarchy:
  - Main memory (DRAM) for currently used data
  - Disk for the main database (secondary storage)
    - Mainly for analytics / mining the data

What about Flash?

- Flash chips used for >20 years
- Flash evolved
  - USB keys

  - Storage in mobile devices

  - Consumer and enterprise flash disks (SSD)

- Flash in a DBMS
  - Main storage
  - Accelerator/enabler (Specialized cache, logging device)

Disks

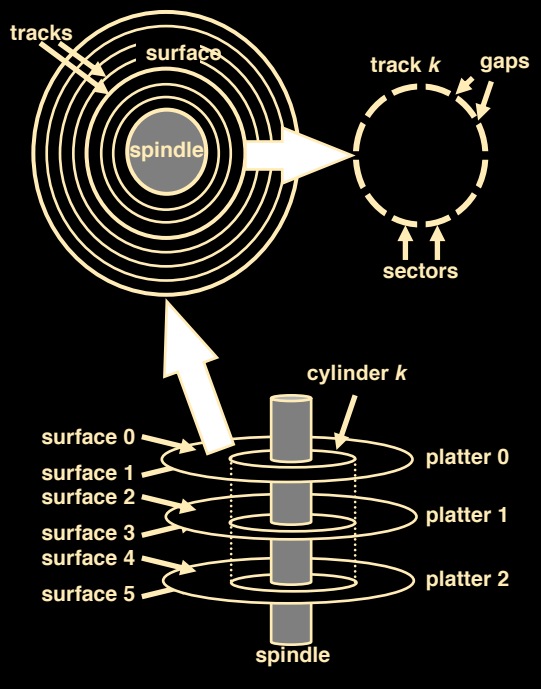
- Secondary storage device of choice
- Main advantage over tapes:

random access vs. *sequential*

- Data is stored and retrieved in units called disk blocks or pages
- Unlike DRAM, time to retrieve a disk page varies
  - Depending on location on disk
  - Relative placement of pages on disk has major impact on DBMS performance!

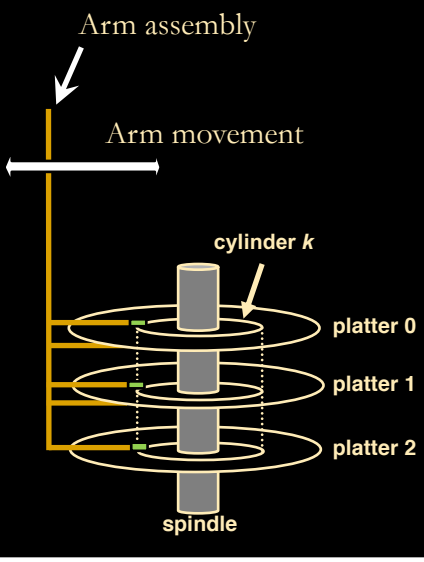
Disk geometry

- Disks consist of platters, each with two surfaces
- Each surface consists of concentric rings called tracks
- Each track consists of sectors separated by gaps
- Sectors contain equal # of data bits (typically 512B)
- Aligned tracks across surfaces/platters form a cylinder



Anatomy of a Disk

- The platters spin (5-15 KRPM)
- The arm assembly is moved in or out to position a head on a desired track.
- Tracks under heads make a cylinder (imaginary!)
- Only one head reads/writes at any one time
- Block size is a multiple of the fixed sector size
- Newer disks have several "zones", with more data on outer tracks

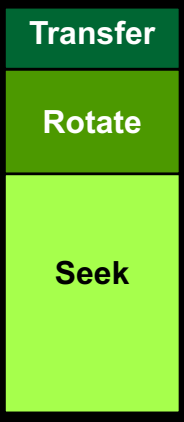


Accessing a Disk Page

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 (moving data to/from disk surface)

- Seek time varies from about 1 to 20 ms
- Rotational delay varies from 0 to 10 ms
- Transfer rate is < 1ms per 4KB page
- Key to lower I/O cost: **reduce seek/rotation delays**
  - Sequential accesses to data



Rules of thumb…

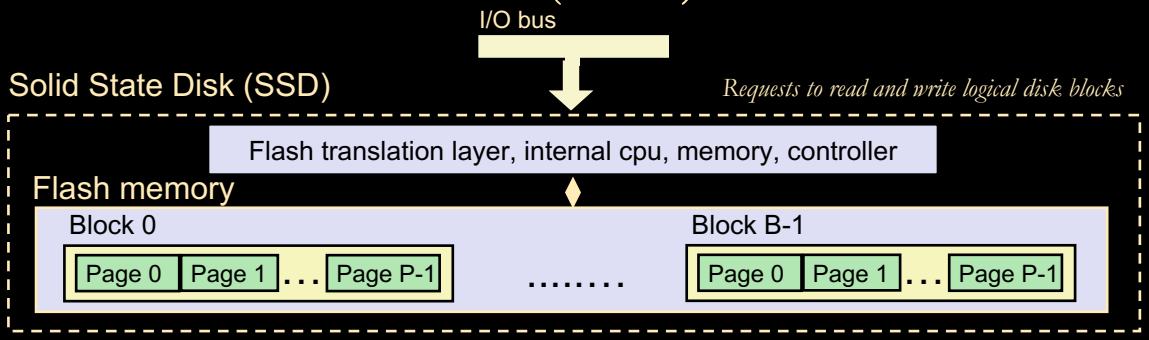
1. Memory access much faster than disk I/O (~ **100,000x**)

2. "Sequential" I/O faster than "random" I/O (~ **10x**)

Flash Solid State Disks (SSDs)

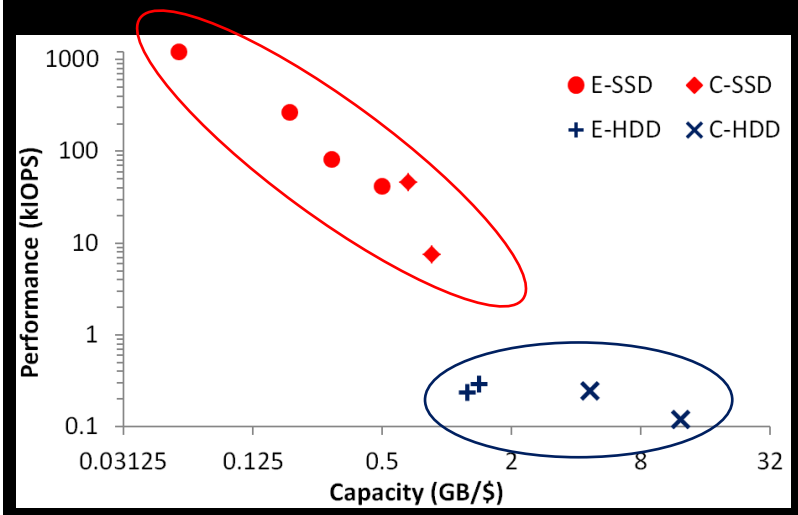
- Secondary storage *or* caching layer
- Main advantage over disks:
  random reads as fast as ***sequential*** reads
- But slow random writes
- Data organized in **pages** (similarly to disks)
- Pages organized in **flash blocks**
- Like DRAM (main memory), time to retrieve a disk page is **independent** of the location on flash disk

Flash Solid State Disks (SSDs)



- Pages: 512KB to 4KB, Blocks: 32 to 128 pages
- Data read/written in units of pages
- Page can be written only after its block has been erased
- A block wears out after ~1M repeated writes (SLC NAND, Micron/Sun, 2008)
  - May sound a lot, but it isn't. 1M seconds ≈ 12 days
  - Thermal annealing may push this to 100M writes (Macronix, 2013)

Flash disks vs. HDD



HDD

- ✓ Large – cheap capacity
- x Inefficient random reads
- ✓ Slow wear-out

SSD (Flash) disks

- x Small – expensive capacity
- ✓ Very efficient random reads
- x Slow writes (throughput)
- x Fast wear-out

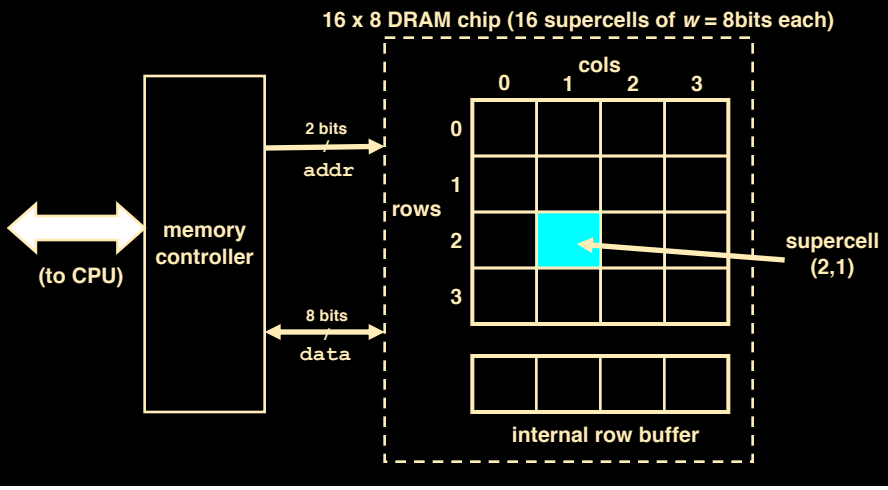
Dynamic Random Access Memory (DRAM)

- Similar to "array of bytes", each bit stored in a "cell"
- Dynamic RAM (DRAM)
  - Each cell stores a bit as a charge in a capacitor
  - Capacitors lose charge; each cell must be refreshed every 10-100 ms
  - More sensitive to disturbances (EMI, radiation, …) than SRAM
  - *Slower* than SRAM, but *cheaper and denser*
- Static RAM (SRAM)
  - Each cell stores a bit in a bi-stable circuit, typically a six-transistor circuit
  - Static – no need for periodic refreshing; keeps data while powered
  - Relatively insensitive to disturbances such as electrical noise
    - Energetic particles (alpha particles, cosmic rays) can flip stored bits

	Transistors/bit	Access time	Refresh?	Sensitive?	Cost	Applications
SRAM	6	1X	No	Yes (but less so)	100X	Cache memory
DRAM	1	10X	Yes	Yes	1X	Main memory, frame buffers

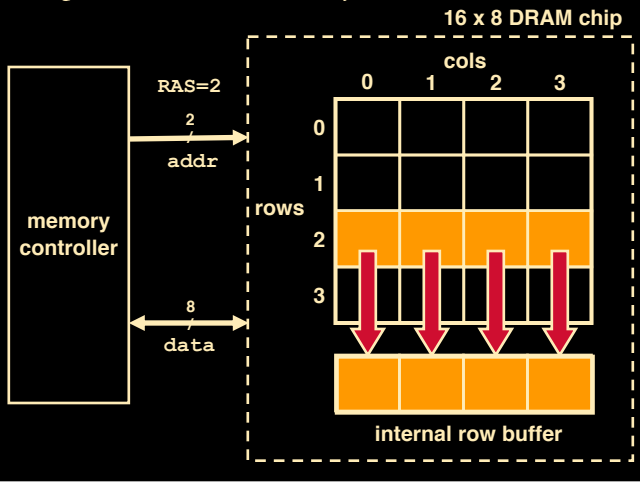
Conventional DRAM organization

- DRAM chip partitioned into supercells
  - Each supercell consists of *w* DRAM cells
  - Supercells organized as arrays of *r* rows and *c* cols (*r*c=d*)
  - A *(d × w)* DRAM stores *(d*w)* bits



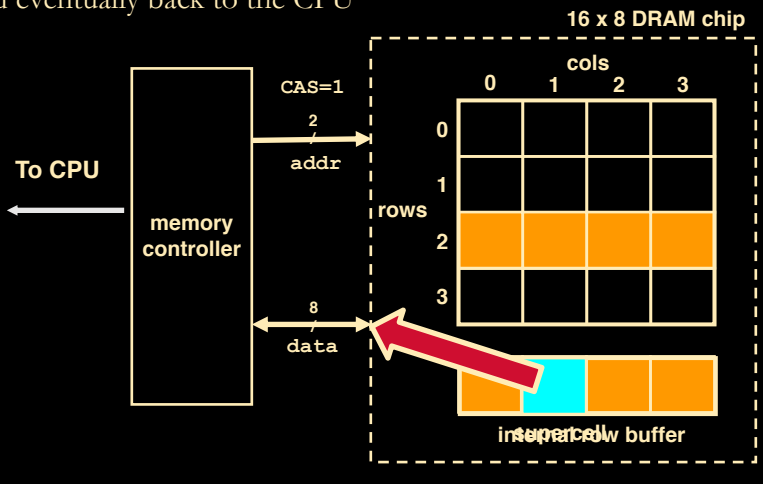
Reading DRAM supercell (2,1)

- Access done in two steps
  - Step 1(a): Row access strobe (**RAS**) selects row 2
  - Step 1(b): Row copied from DRAM array to row buffer



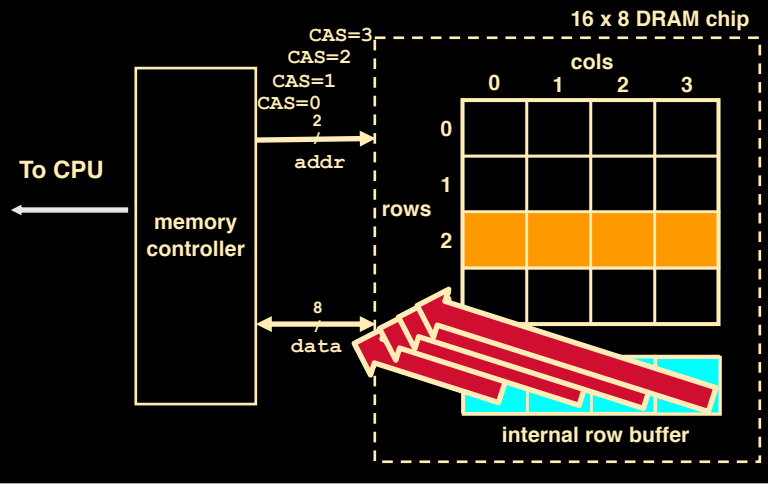
Reading DRAM supercell (2,1)

- …
  - Step 2(a): Column access strobe (**CAS**) selects column 1
  - Step 2(b): Copy supercell (2,1) from row buffer to data lines, and eventually back to the CPU

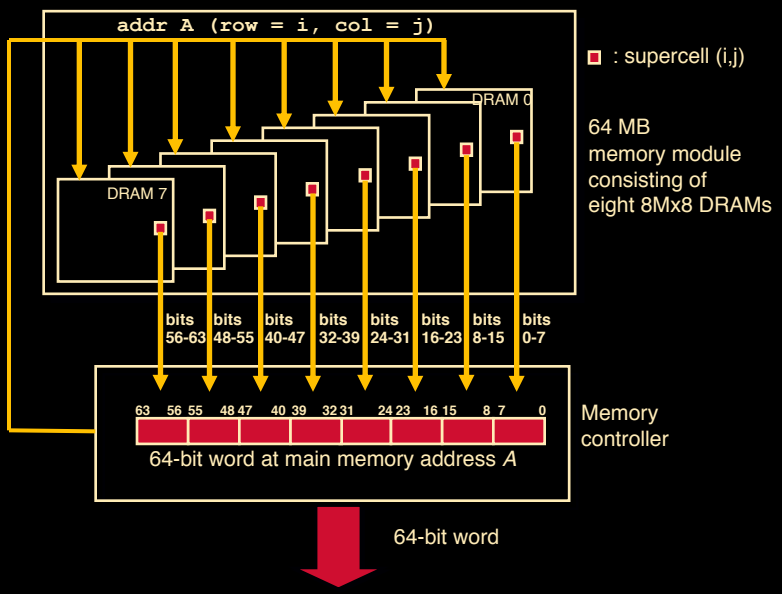


Fast Page Mode: Reading (2,0)(2,1)(2,2)(2,3)

- …
  - Consecutive **CAS** select consecutive columns of the same row
  - Direct row buffer access, no precharge → VERY fast memory access



Memory module basic idea



: supercell (i,j)

64 MB memory module consisting of eight 8Mx8 DRAMs

Caches

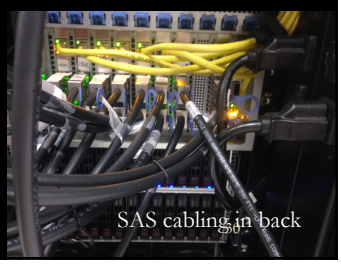
- ***Cache:*** A smaller, faster storage device that keeps a copy of a subset of the data from a larger, slower device

- Fundamental idea of a memory hierarchy:
  - For each *k*, the faster, smaller device at level *k* serves as a cache for the larger, slower device at level *k+1*
  - Each level stores some of the most frequently accessed data
  - The closer the cache is to the processor, the hotter the cached data

- Why do memory hierarchies work?
  - Programs tend to access data at level *k* more often than they access data at level *k+1*
  - Thus, the storage at level *k+1* can be slower, and thus larger and cheaper per bit
  - *Net effect: A* ***large pool of memory*** *that* ***costs as much as the cheap*** *storage near the bottom, but that serves data to programs at a rate close to the* ***rate of the fast*** *storage near the top*

A Database Server @ NU

- 264 fast (10k RPM) magnetic disks (for production)
- 56 slow (7200 RPM) magnetic disks (for backup)
- ~150 TB storage capacity
- Comprised of 6 physical chassis (boxes) in one big cabinet, about the size of a coat closet



Stack Overflow database

- Questions and Answers from a popular programming help website
  - 150 GB of data
  - 29M posts
  - 55M comments
- Reading through all the data takes about 1,000 seconds (17 minutes)
- We do not want to wait 17 minutes for an answer
- We need a way to quickly find the data of interest

How to work with very large datasets fast?

Indexing

- When working with large amounts of data we have to know where to find an item of interest
- In the paper analogy, we do not want to request every box from the warehouse and scan through every page to find what we're looking for
- *Sorting* the data can help tremendously, because it allows *binary search*