CS422
Database systems

Data Storage

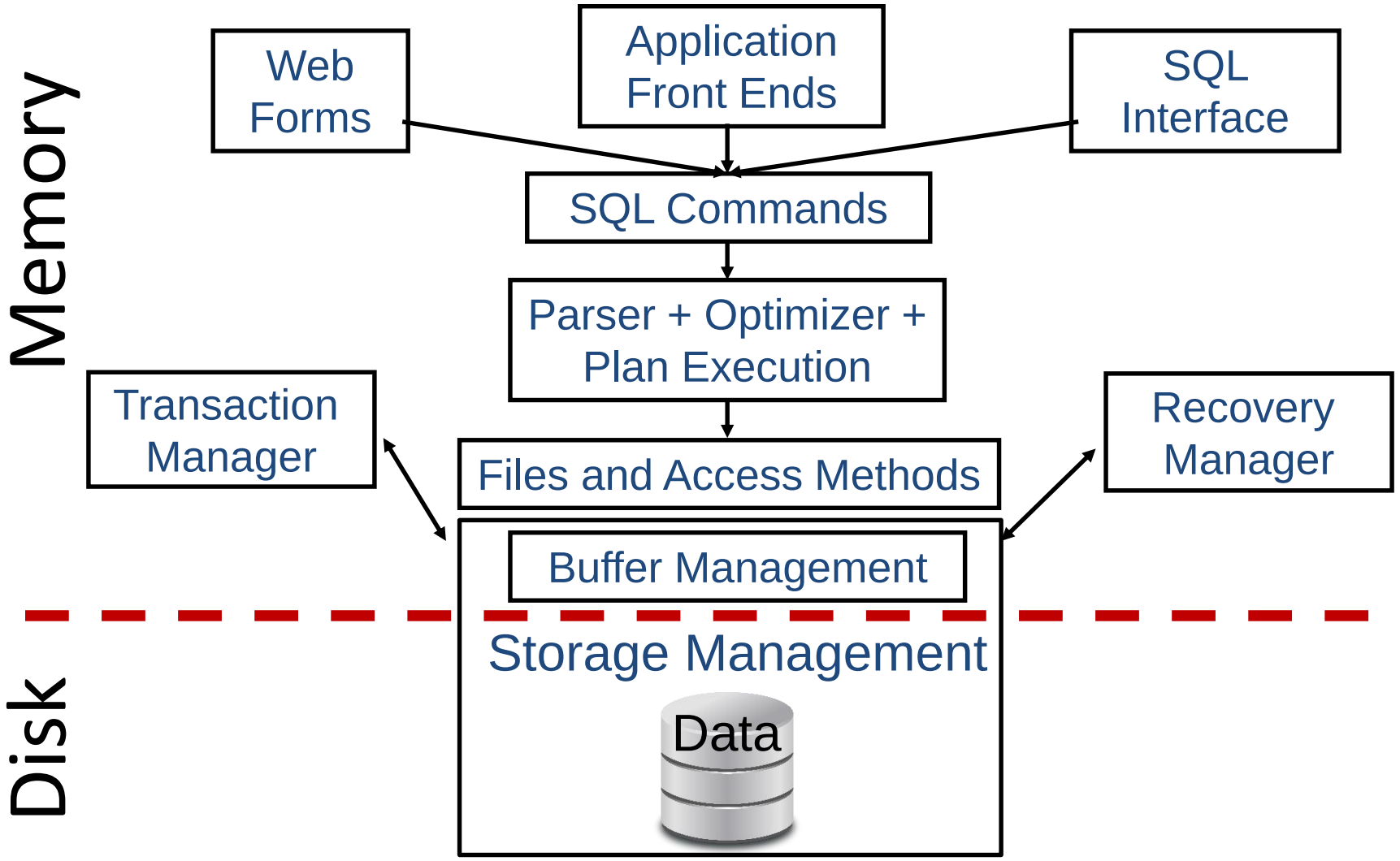
Data-Intensive Applications and Systems (DIAS) Laboratory
École Polytechnique Fédérale de Lausanne

"It is better to have 100 functions operate on one data structure than to have 10 functions operate on 10 data structures"
– Alan Perlis

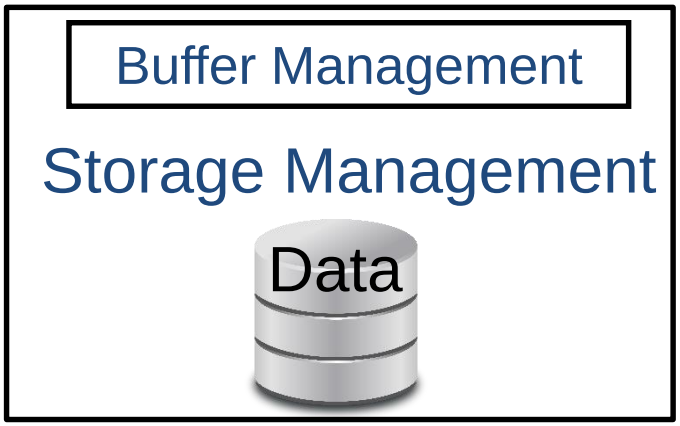
Some slides adapted from:

- Andy Pavlo
- CS-322

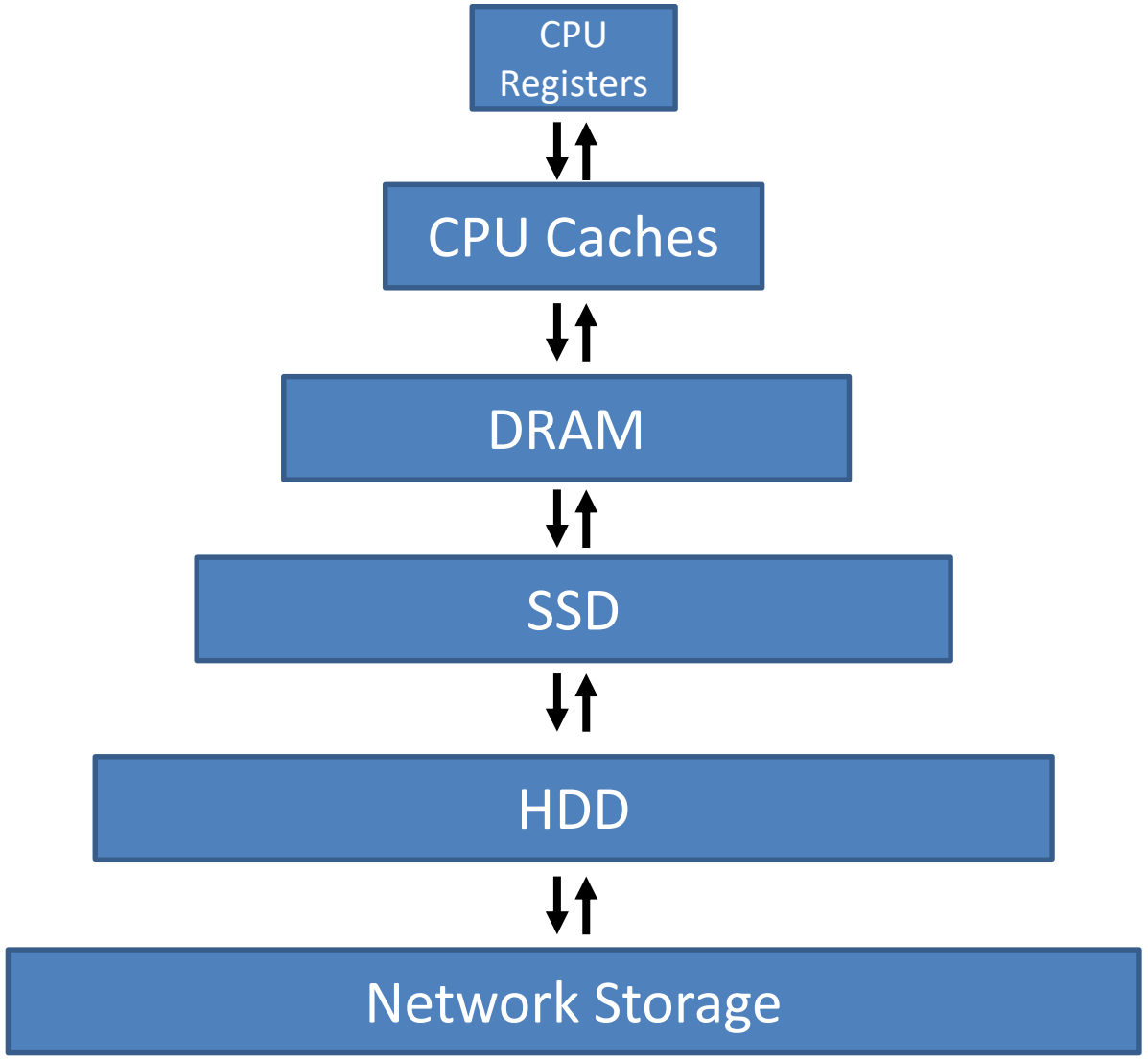
(Simplified) DBMS Architecture



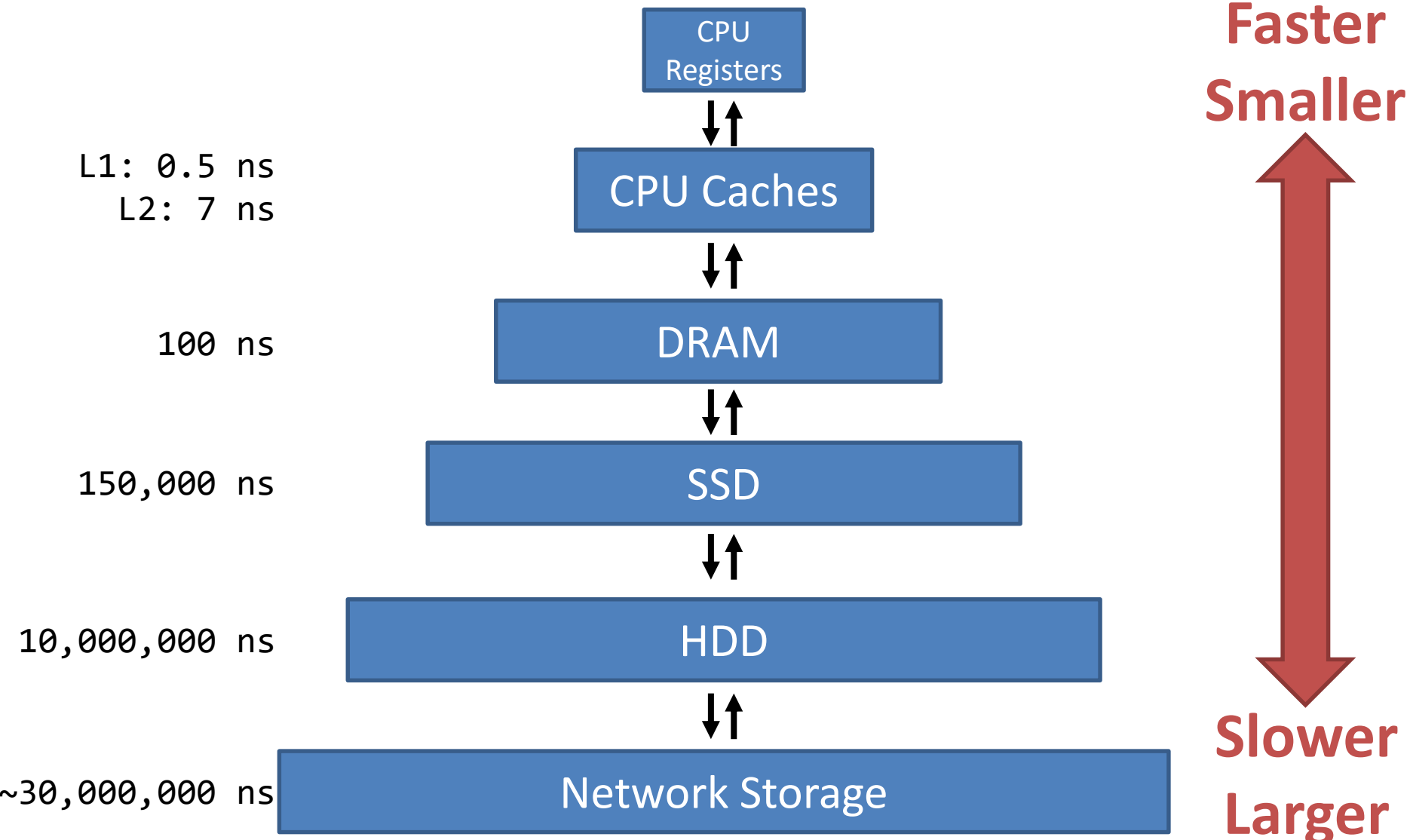
Today's topic



Storage Hierarchy



Access Times



Goals

Allow the DBMS to manage databases that exceed the amount of memory available

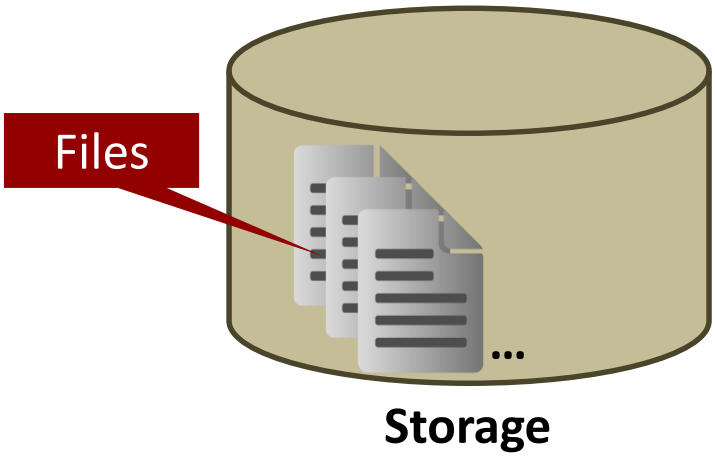
Try to have the working set in main memory

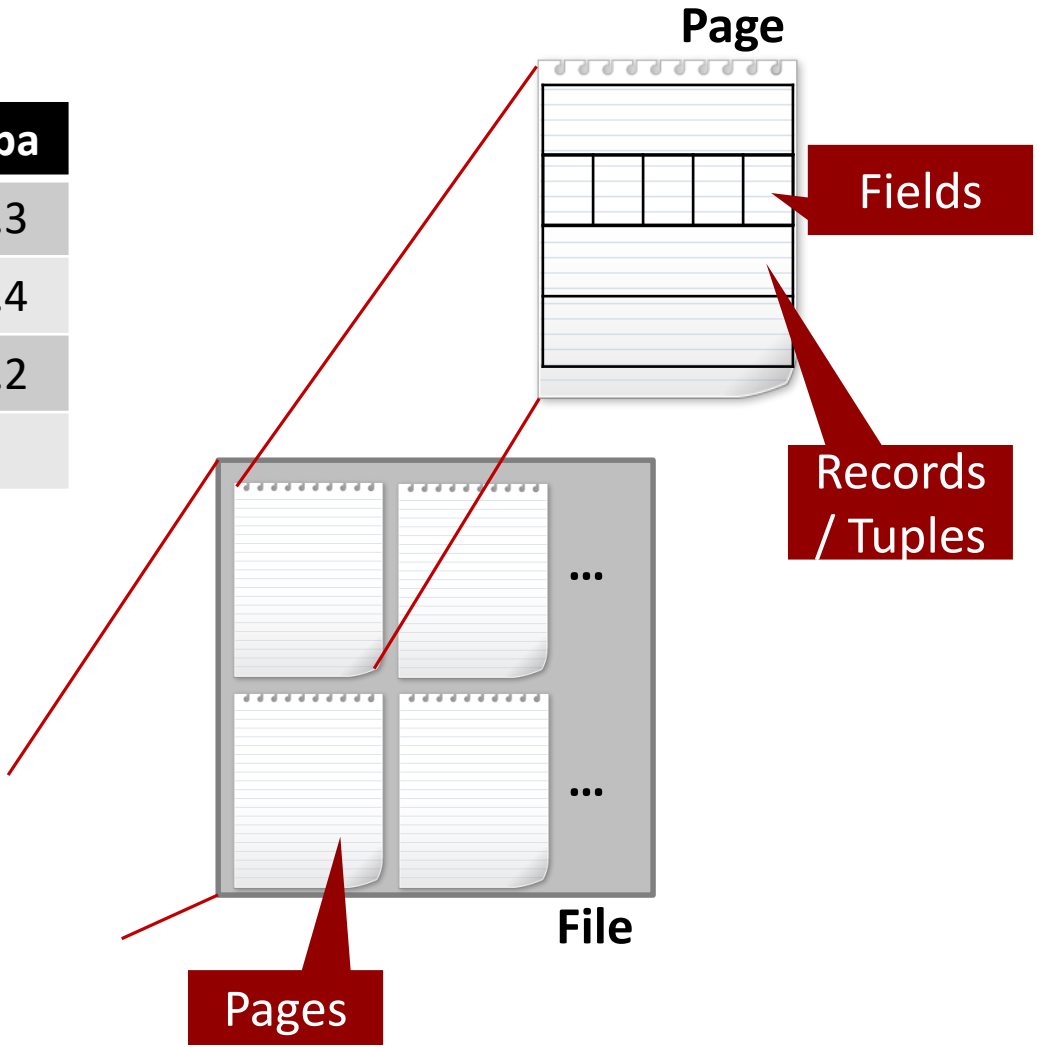
Try to reuse data placed in topmost layers as much as possible

Outline

Students

sid	name	login	age	gpa
50000	Dave	dave@cs	19	3.3
53666	Jones	jones@cs	18	3.4
53688	Smith	smit@ee	18	3.2
…	…	…	…	…





Outline

- File Storage
- Page Layout
- Buffer Management

File Storage

The DBMS stores a database as one or more files on disk.

The **Storage Manager** is responsible for maintaining a database's files, and organizes them as a collection of **pages**.

- Tracks data read/written to pages
- Tracks available space

Alternative File Organizations

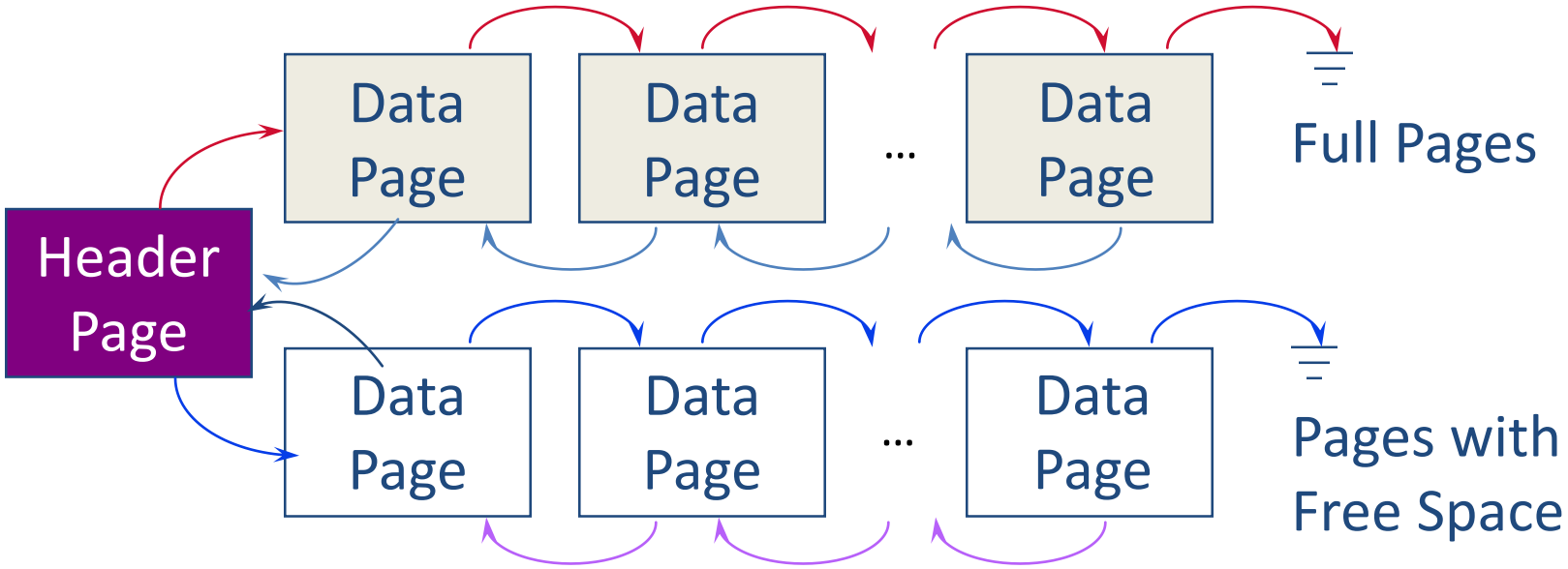
Many alternatives exist, *each good for some situations, and not so good in others.*
A non-exhaustive list is the following*:*

- Heap files: Suitable when typical access is a file scan retrieving all records.
- Sorted Files: Best for retrieval in some order, or for retrieving a ‘range’ of records.

Heap (Unordered) Files

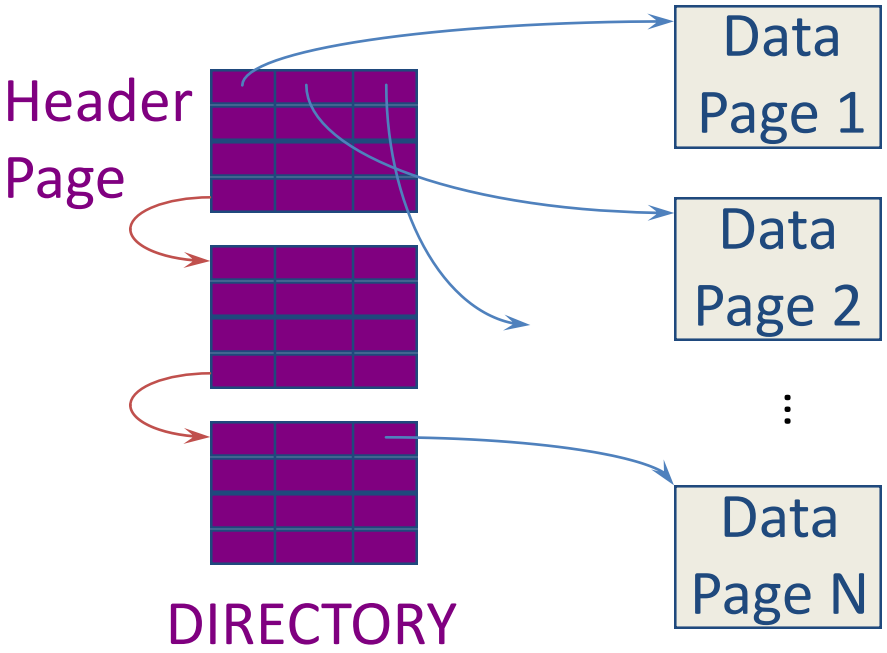
- Simplest file structure
  - contains records in no particular order
  - Need to be able to scan, search based on rid
- As file grows and shrinks, disk pages are allocated and de-allocated.
  - Need to manage free space

Heap File Implemented Using Lists



- <Heap file name, header page id> stored somewhere
- Each page contains 2 ‘pointers’ plus data.
- Manage free pages using free list
  - What if most pages have some space?

Heap File Using a Page Directory

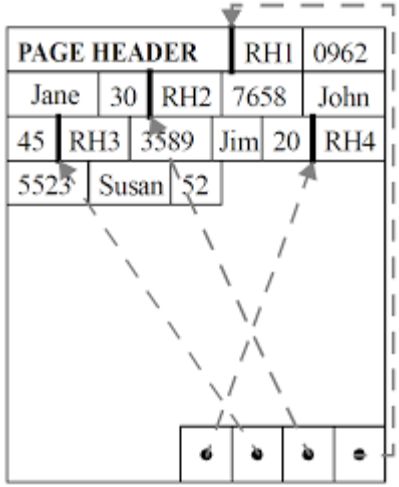


- The directory is a collection of pages
  - linked list implementation is just one alternative.
- The entry for a page can include the number of free bytes on the page.
  - *Much smaller than linked list of all HF pages*!

Outline

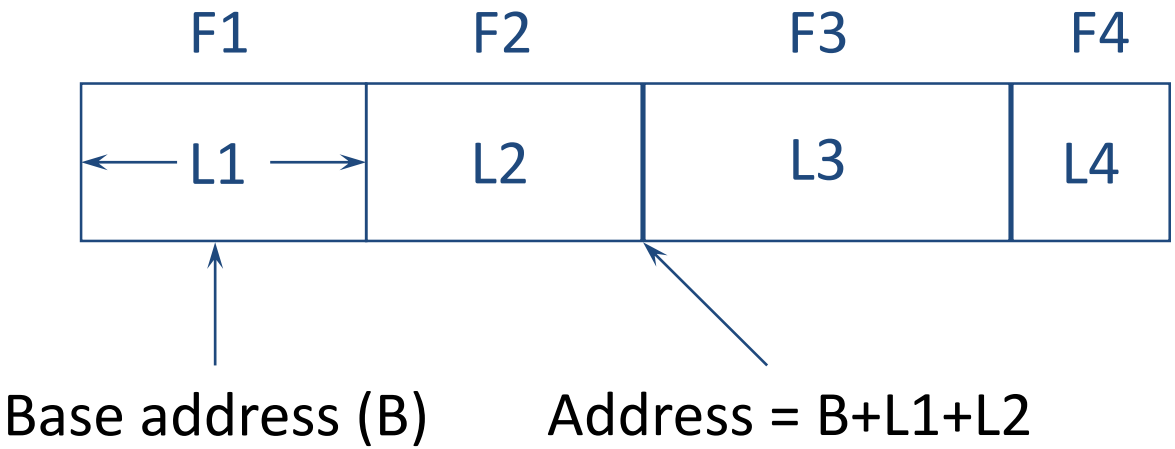
- File Storage
- Page Layout
  - NSM, aka row-oriented
- Buffer Management

The N-ary Storage Model



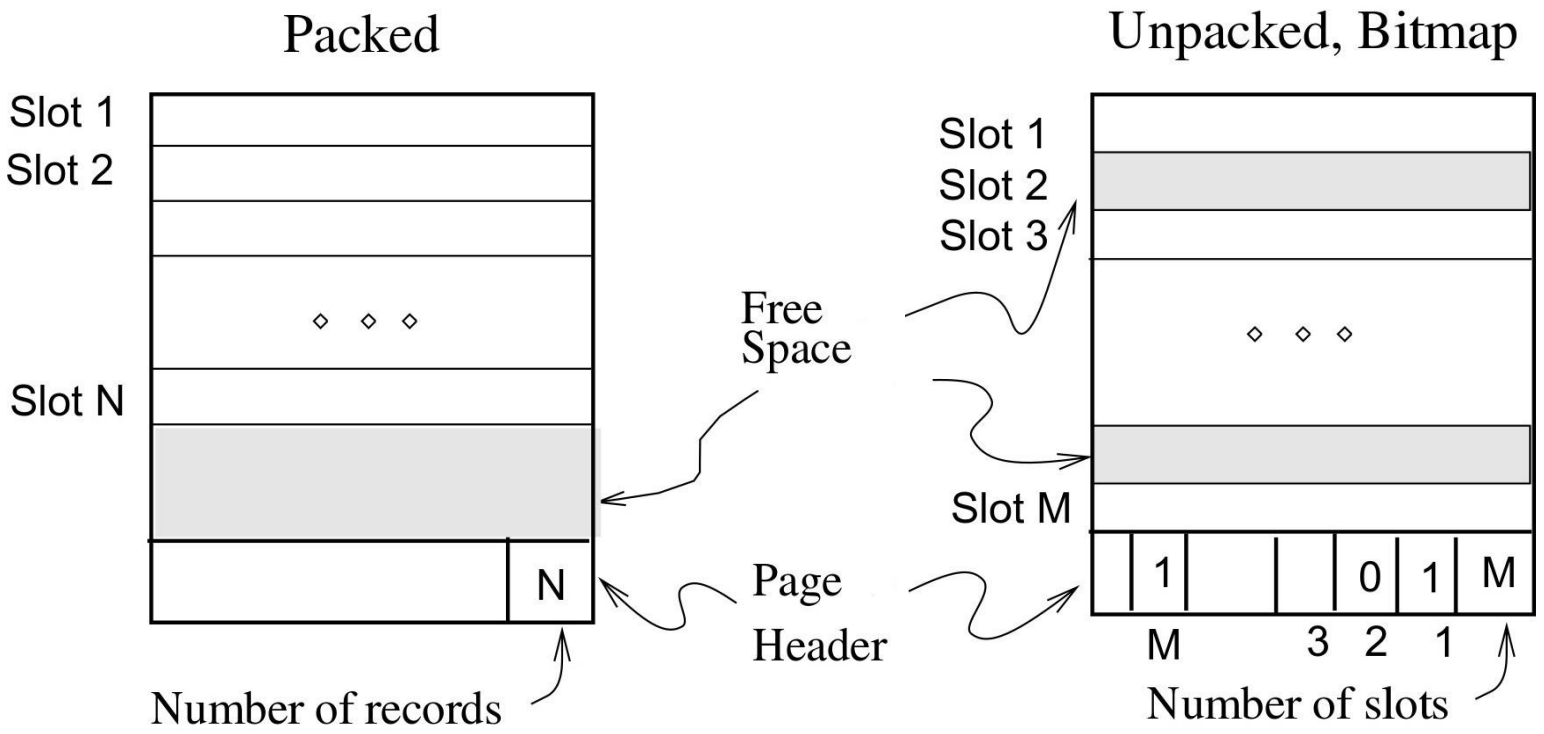
- Page = collection of slots
- Each slot stores one record
  - Record identifier: <page_id, slot_number>
  - Option 2: <uniq> -> <page_id, slot_number>
- Page format should support
  - Fast searching, inserting, deleting
- *Page format* depends on record format
  - Fixed-Length
  - Variable-Length

Record Formats: Fixed-Length



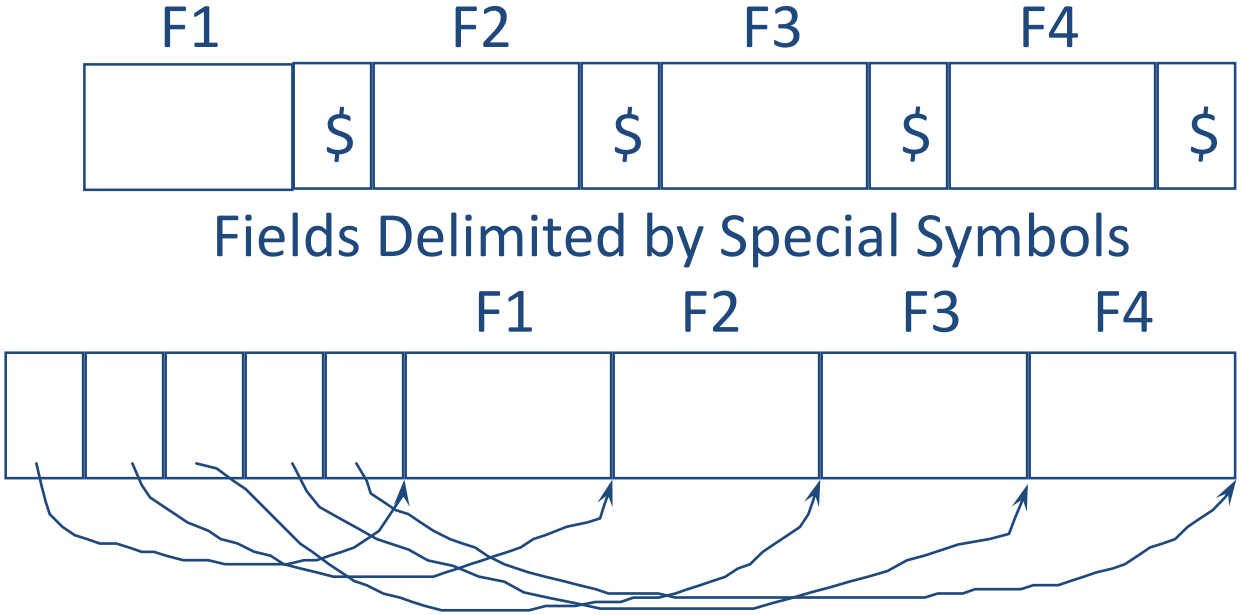
- Schema is stored in ***system catalog***
  - Number of fields is fixed for all records of a table
  - Domain is fixed for all records of a table
- Each field has fixed length
- Finding i^{th} field is done via arithmetic.

Page Format: Fixed-Length Records



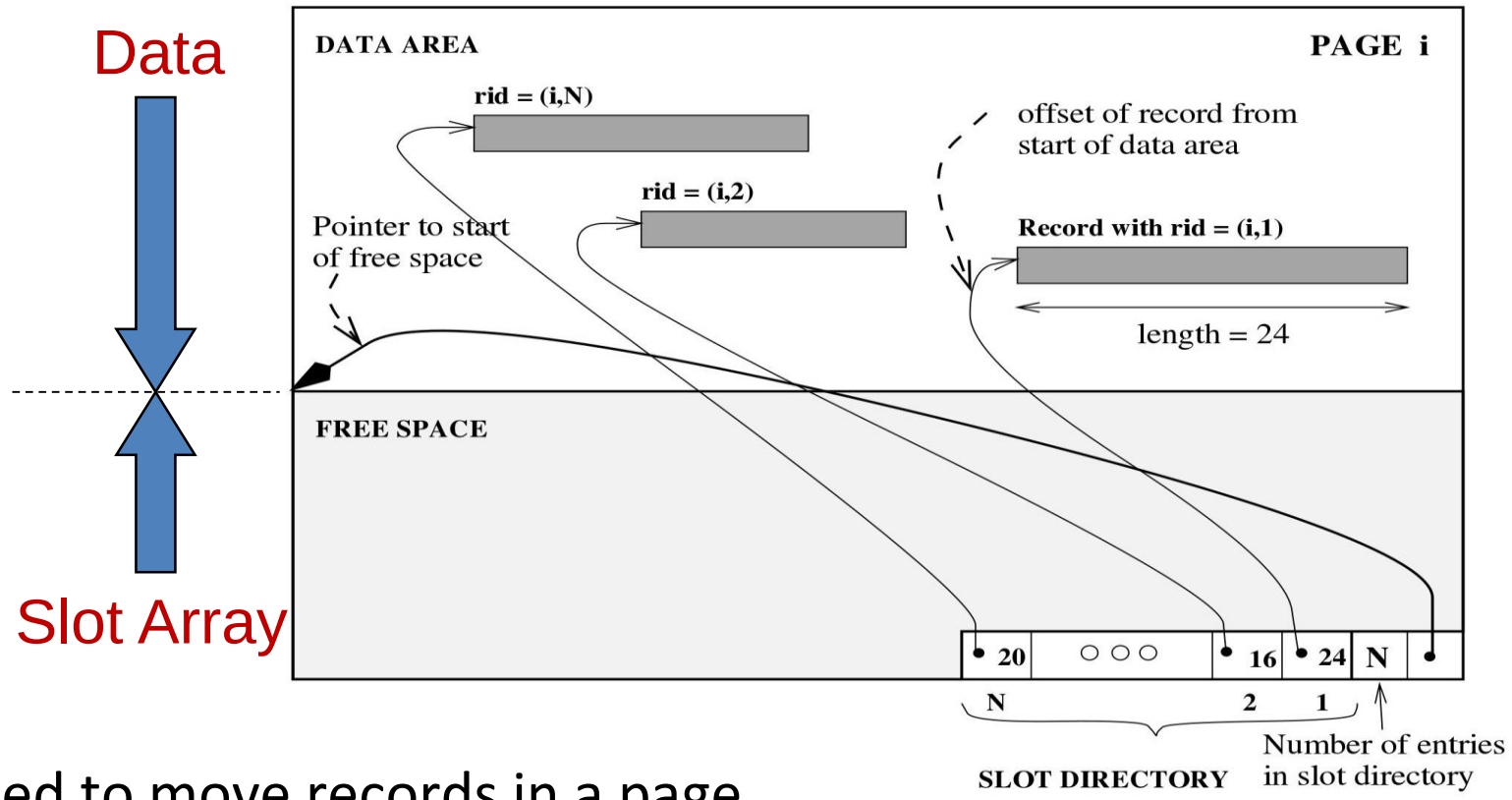
- *Record id = <page id, slot #>*
- In the *packed* case, moving records for free space management changes rid; maybe unacceptable.

Record Formats: Variable-Length



- Array of field offsets is typically superior
  - Direct access to fields
  - Clean way of handling NULL values
  - Oracle 8: length—data pairs, DB2: Array of offsets

Page Format: Variable-Length Records



- Need to move records in a page
  - Allocation/deletion must find/release free space
- Maintain slot directory with <record offset, record length> pairs
  - Records can move on page without changing rid
  - Useful for freely moving fixed-length records (ex: sorting)

Variable-Length Records: Issues

- If a field grows and no longer fits?
  - shift all subsequent fields
- If record no longer fits in page?
  - Move a record to another page after modification
- What if record size > page size?
  - SQL Server record size = 8KB
  - DB2 record size = page size

Outline

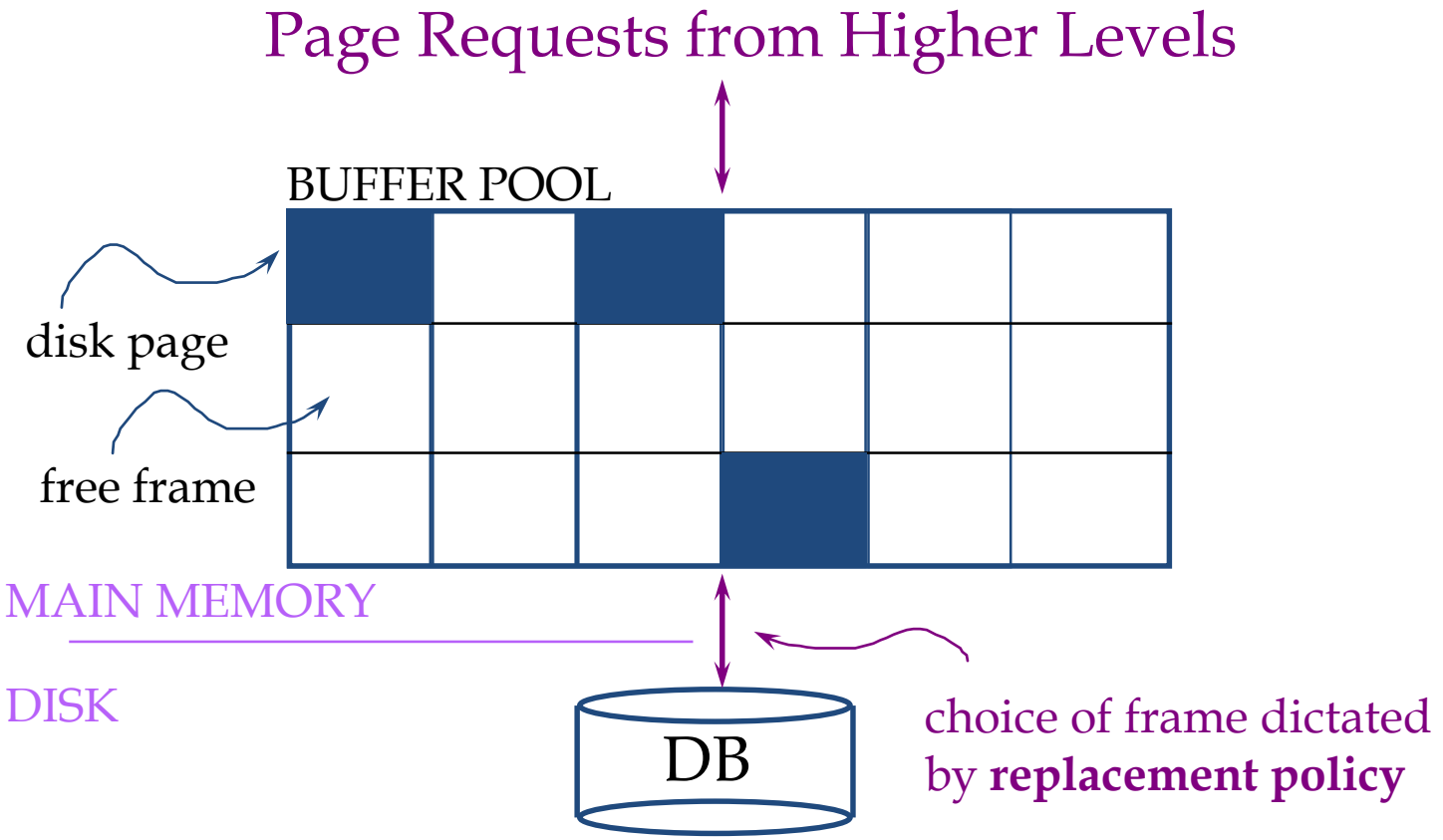
- File Storage
- Page Layout
  - NSM, aka row-oriented
- Buffer Management

Can't we just use the OS buffering?

- Layers of abstraction are good … but:
  - Unfortunately, OS often gets in the way of DBMS

- DBMS needs to do things "its own way"
  - Specialized prefetching
  - Control over buffer replacement policy
    - LRU not always best (sometimes worst!!)
  - Control over thread/process scheduling
    - "Convoy problem"
      - Arises when OS scheduling conflicts with DBMS locking
  - Control over flushing data to disk
    - WAL protocol requires flushing log entries to disk

Buffer Management in a DBMS



- *Data must be in RAM for DBMS to operate on it!*
- *Buffer manager hides the fact that not all data is in RAM (just like hardware cache policies hide the fact that not all data is in the caches)*

When a Page is Requested ...

- Buffer pool information table contains:
  <frame#, pageid, pin_count, dirty>

- If requested page is not in pool:
  - Choose a frame for *replacement*
    (only un-pinned pages are candidates)
  - If frame is "dirty", write it to disk
  - Read requested page into chosen frame
- *Pin* the page and return its address.

* *If requests can be predicted (e.g., sequential scans) pages can be pre-fetched several pages at a time!*

More on Buffer Management

- Requestor of page must unpin it, and indicate whether page has been modified:
  - *dirty* bit is used for this.
- Page in pool may be requested many times,
  - a *pin count* is used. A page is a candidate for replacement iff *pin count* = 0 (*"unpinned"*)
- CC & recovery may entail additional I/O when a frame is chosen for replacement

Buffer Replacement Policy

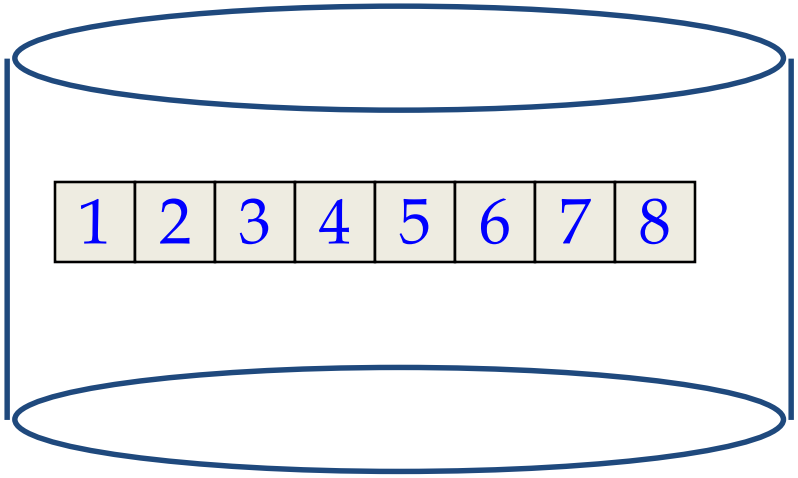
- Frame is chosen for replacement by a *replacement policy:*
  - Least-recently-used (LRU), MRU, Clock, etc.

- Policy can have big impact on # of I/O's; depends on the *access pattern*.

LRU Replacement Policy

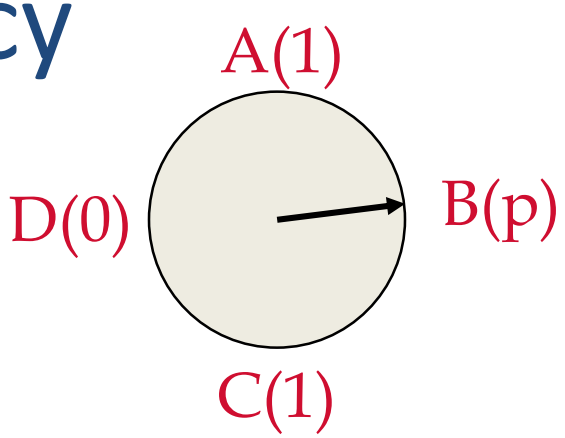
- *Least Recently Used (LRU)*
  - for each page in buffer pool, keep track of time last *unpinned*
  - replace the frame which has the oldest (earliest) time
  - very common policy: intuitive and simple
- Problems?
- *Problem: Sequential flooding*
  - LRU + repeated sequential scans.
  - # buffer frames < # pages in file means each page request causes an I/O. *MRU* much better in this situation (but not in all situations, of course).

Sequential Flooding – Illustration



Repeated scan of file …

“Clock” Replacement Policy



- An approximation of LRU.
- Arrange frames into a cycle, store one “reference bit” per frame
- When pin count goes to 0, reference bit set on.
- When replacement necessary:

```
do {
        if (pincount == 0 && ref bit is off)
                choose current page for replacement;
        else if (pincount == 0 && ref bit is on)
                turn off ref bit;
        advance current frame;
} until a page is chosen for replacement;
```


OTHERSIDE
RED HOT CHILI PEPPERS

Outline

- File Storage
  - Log-structured
- Page Layout
  - NSM, aka row-oriented
  - DSM, aka column-oriented
- Buffer Management

The database as the log

Many alternatives exist, *each good for some situations, and not so good in others.*
A non-exhaustive list is the following*:*

- Heap files: Suitable when typical access is a file scan retrieving all records.
- Sorted Files: Best for retrieval in some order, or for retrieving a ‘range’ of records.
- Log-structured Files: Best for very fast insertions/deletions/updates

Log-structured files

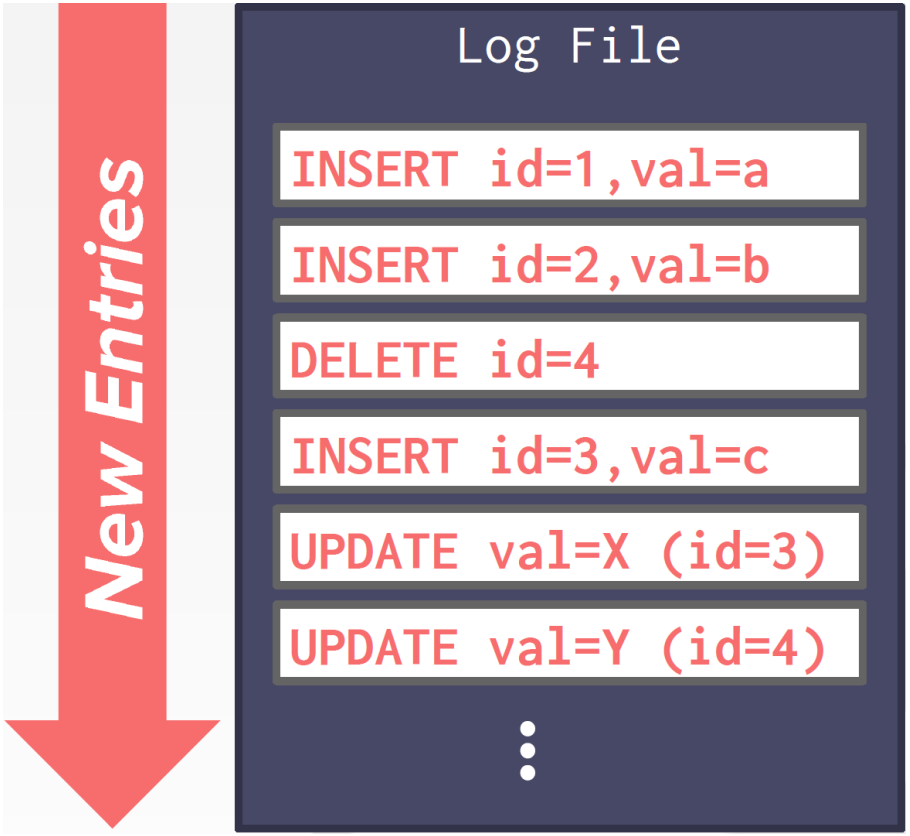
Instead of storing tuples in pages, the DBMS only stores **log records**.

The system appends log records to the files of how the database was modified.

- Inserts: Store the entire tuple
- Deletes: Mark tuple as deleted
- Updates: Store delta of just the attributes that were modified

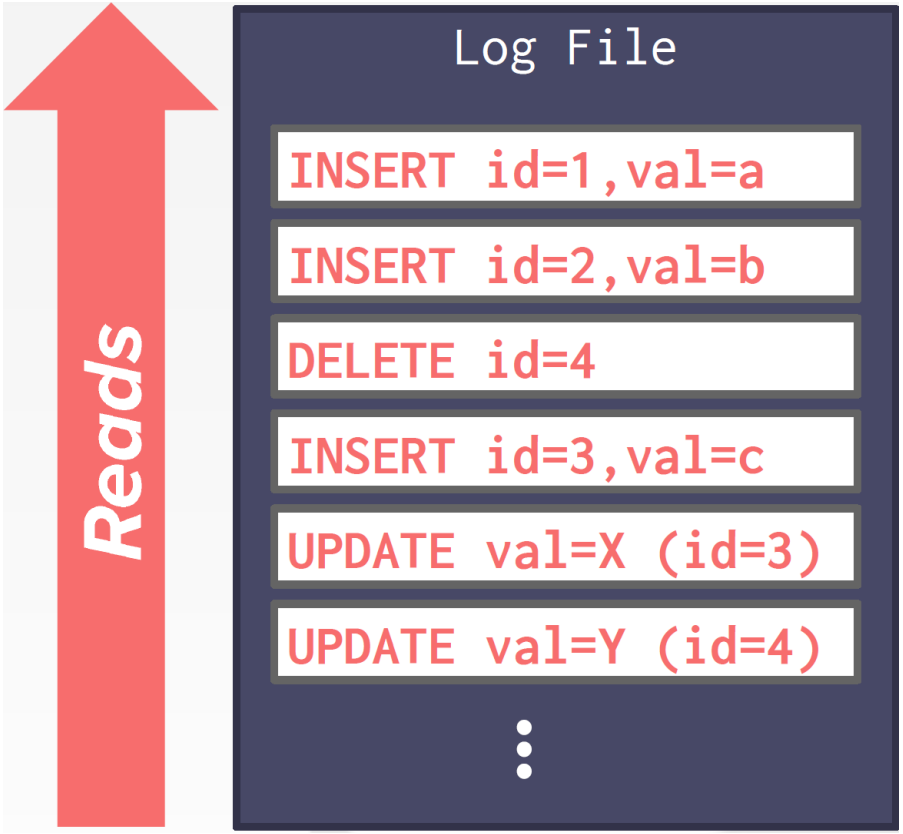
Writing to log-structured files

- Inserts: Store the entire tuple
- Deletes: Mark tuple as deleted
- Updates: Store delta of just the attributes that were modified



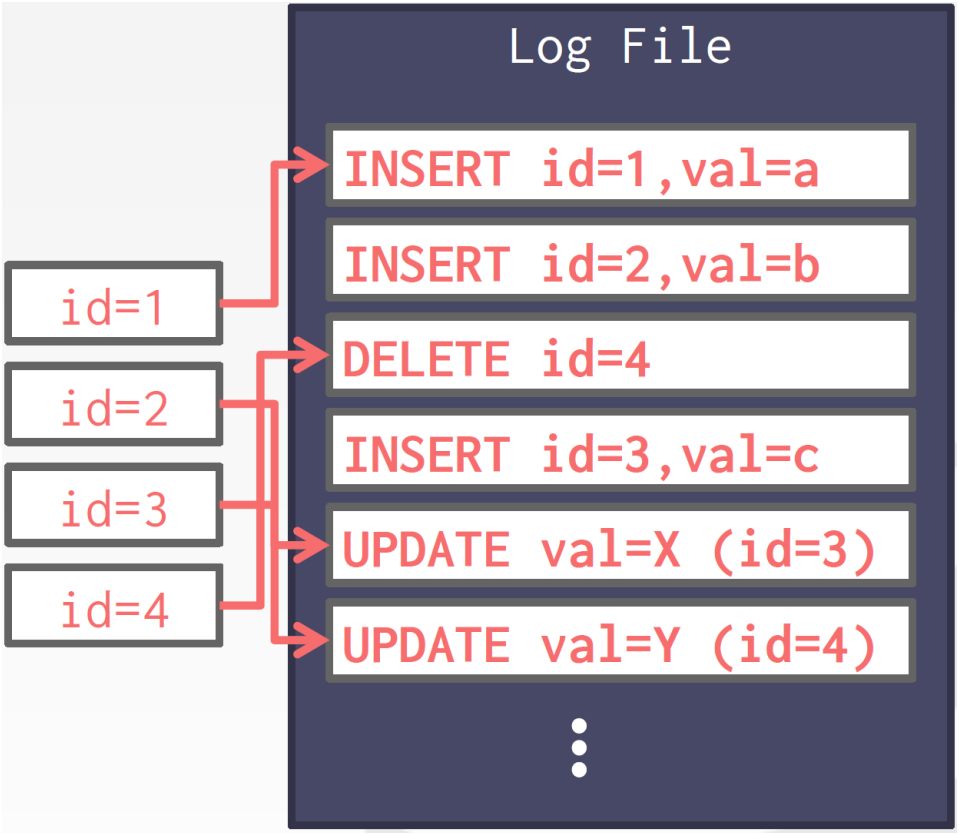
Reading from log-structured files

– DBMS scans log backwards, and “recreates” the tuple



Reading from log-structured files

- DBMS scans log backwards, and “recreates” the tuple
- Build indexes to allow **jumps** in the log
- Periodically compact the log



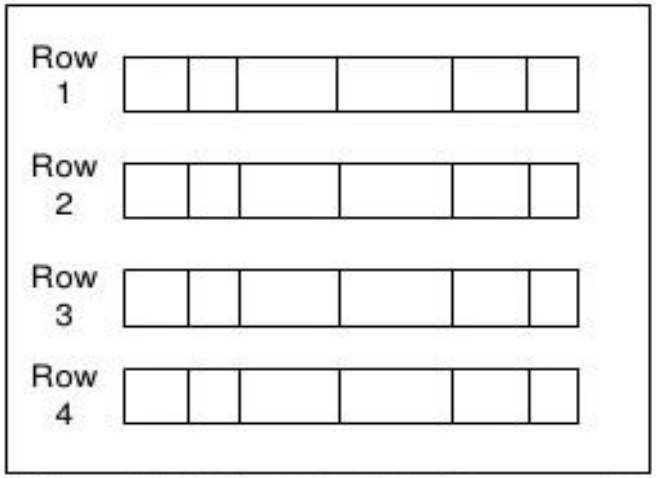
Outline

- File Storage
- Page Layout
  - NSM, aka row-oriented
  - DSM, aka column-oriented
- Buffer Management

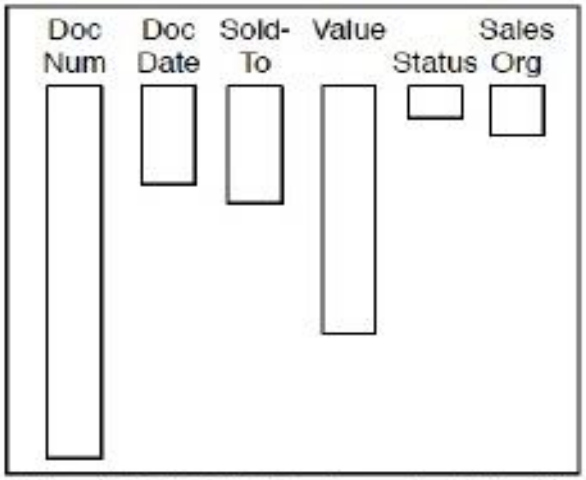
Decomposition Storage Model (DSM)

Document Number	Document Date	Sold-To Party	Order Value	Status	Sales Organization	...
95769214	2009-10-01	584	10.24	CLOSED	Germany Frankfurt	...
95769215	2009-10-01	1215	124.35	CLOSED	Germany Berlin	...
95779216	2009-10-21	584	47.11	OPEN	Germany Berlin	...
95779217	2009-10-21	454	21.20	OPEN	Germany Frankfurt	...

Row Store

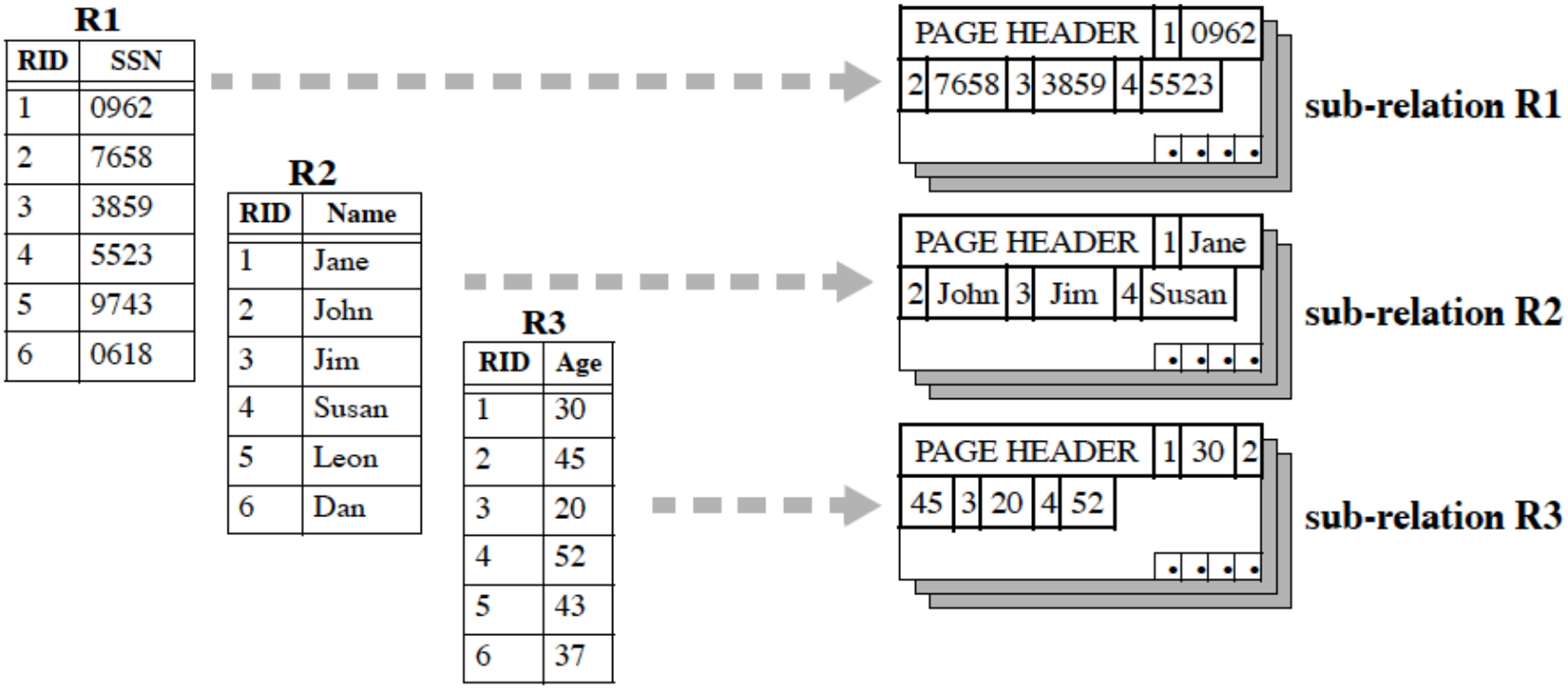


Column Store



In-Memory/HANA Enterprise Data Management | SAP UA Conference | March 23rd 2012 | Dr. Alexander Zeier, MIT

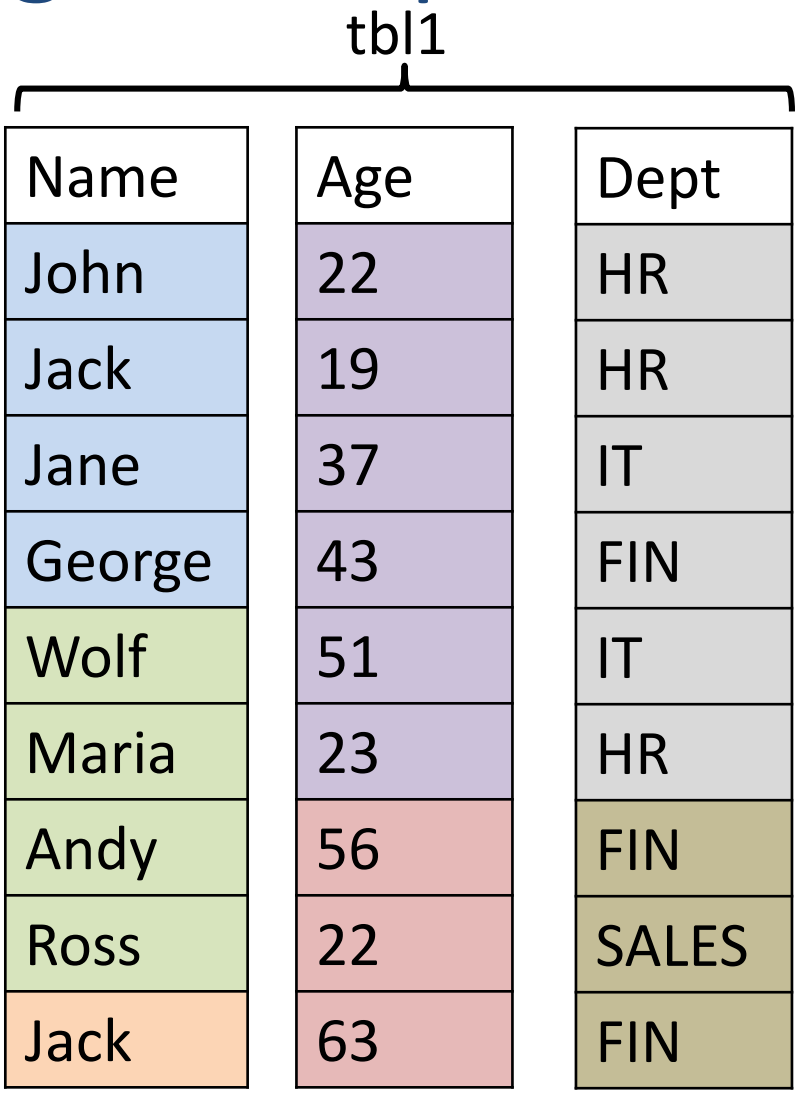
DSM Page Format



Decompose a relational table to sub-tables per attribute

Columnar storage example

- Columns stored in pages
  - Denoted with different colors
- Each column can be accessed individually
  - Pages loaded only for the desired attributes



Three different files: tbl1.name tbl1.age tbl1.dept

DSM Properties

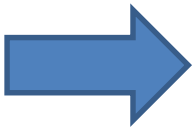
Pros:

- Saves IO by bringing only the relevant attributes
- (Very) memory- and CPU-friendly
- Compressing columns is typically easier

Compression

- Lossless compression
- IO reduction → less CPU wait time
  - Introduces small additional CPU load on otherwise idle CPU
- Run-length encoding (RLE)

Dept
HR
HR
SALES
IT
IT



CDept
(2 x HR)
(1 x SALES)
(2 x IT)

Compression (2)

- Bit-vector encoding
  - Useful when we have categorical data
  - One bit vector for each distinct value
  - Useful when a few distinct values
  - Vector length = # elements

Dept
HR
HR
SALES
IT
IT

HR: 11000
SALES: 00100
IT: 00011

Compression (3)

- Dictionary encoding
  - Replace long values (e.g., strings) with integers
  - Useful when a few distinct values

Dept
HR
IT
HR
SALES
HR
FINANCE
FINANCE
IT

Dictionary	
1	HR
2	IT
3	SALES
4	FINANCE

CDept
1
2
1
3
1
4
4
2

Smaller	- memory requirements
dictionaries	- cache utilization
improve	- effectiveness of run-length encoding

- Frequency partitioning
  - Reorganize each column to **reduce entropy** at each page

Dept
HR
IT
FIN
FIN
HR
HR
FIN
HR
SALES

Column reorganization

Dept	
1	HR
5	HR
6	HR
8	HR
2	IT
3	FIN
4	FIN
7	FIN
9	SALES

Dictionary-based compression with **per-page** dictionaries

CDept	
1	1
5	1
6	1
8	1
2	1
3	2
4	2
7	2
9	1

Operators over compressed data

No need to decompress for most query operators

- Dictionary encoding => integer comparisons faster than string comparisons

```
SELECT name FROM tbl WHERE DEPT="HR"
vs
SELECT name FROM tbl WHERE CDEPT=1
```

  - Per-page dictionaries?
- Run-length encoding → batch processing
- Bit-vector encoding → find the ones directly from the bit vectors

```
SELECT COUNT(*) FROM tbl WHERE CDEPT="HR"
```

DSM Properties

Pros:

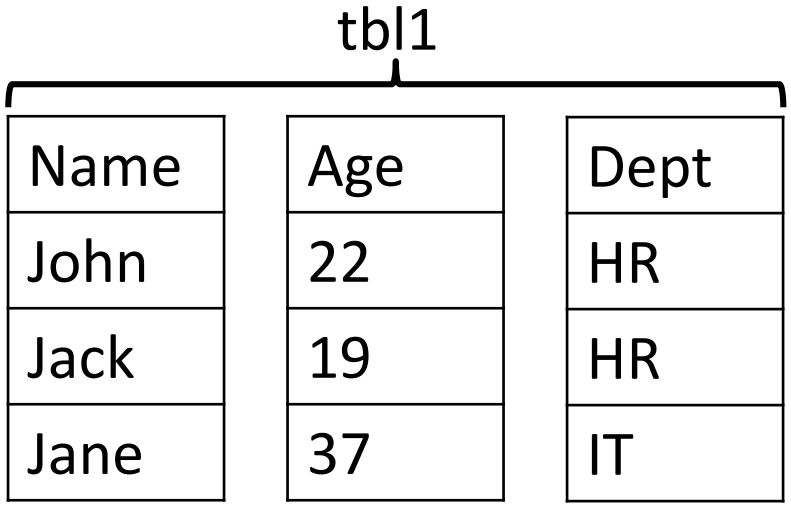
- Saves IO by bringing only the relevant attributes
- (Very) memory- and CPU-friendly
- Compressing columns is typically easier

Cons:

- Writes more expensive
- Have to materialize relations at some point

Column stores: Writes

- Row insertions/deletions
  - Affects all columns
  - Multiple I/Os
  - Complicated transactions
- Deletes/updates: Implicit
  - Mark record as deleted!

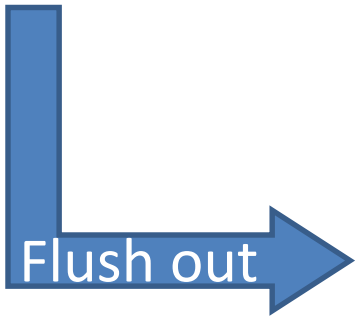


- Massive data loading:
  Write-optimized storage (WOS)

Write-optimized storage

In-memory buffer (fixed-size)

Name	Age	Dept



Filesystem storage: **3 different files**, possibly compressed!

Name	Age	Dept
John	22	HR
Jack	19	HR
Jane	37	IT
Jake	43	FIN
Jill	24	IT
James	56	FIN
Jessica	34	IT

Batch-loading:

- <Jill, 24, IT>
- <James, 56, FIN>
- <Jessica, 34, IT>

Write rows in-memory, flush columns to disk

The materialization problem

(Strictly speaking,) we always need to know the tuple identifier of each column entry => Size Bloat.

tid	Name
1	John
2	Jack
3	Jane

tid	Age
1	22
2	19
3	37

tid	Dept
1	HR
2	HR
3	IT

Alternative: Virtual ids

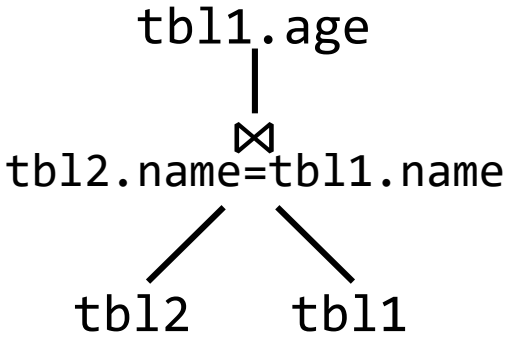
- Identical order across columns
- No need to store ids
- Minimal book-keeping with fixed-width columns

Name	Age	Dept
John	22	HR
Jack	19	HR
Jane	37	IT

Virtual ids not always applicable

The materialization problem

- When compressing columns, they may stop being fixed-width
- When joining tables, columns can get shuffled
  => Cannot use virtual ids
  => Stitching causes random accesses



tid	Name
1	John
3	Jane
2	Jack

tid	Age
1	22
2	19
3	37

The order of tbl1.name entries can change after the join!!!

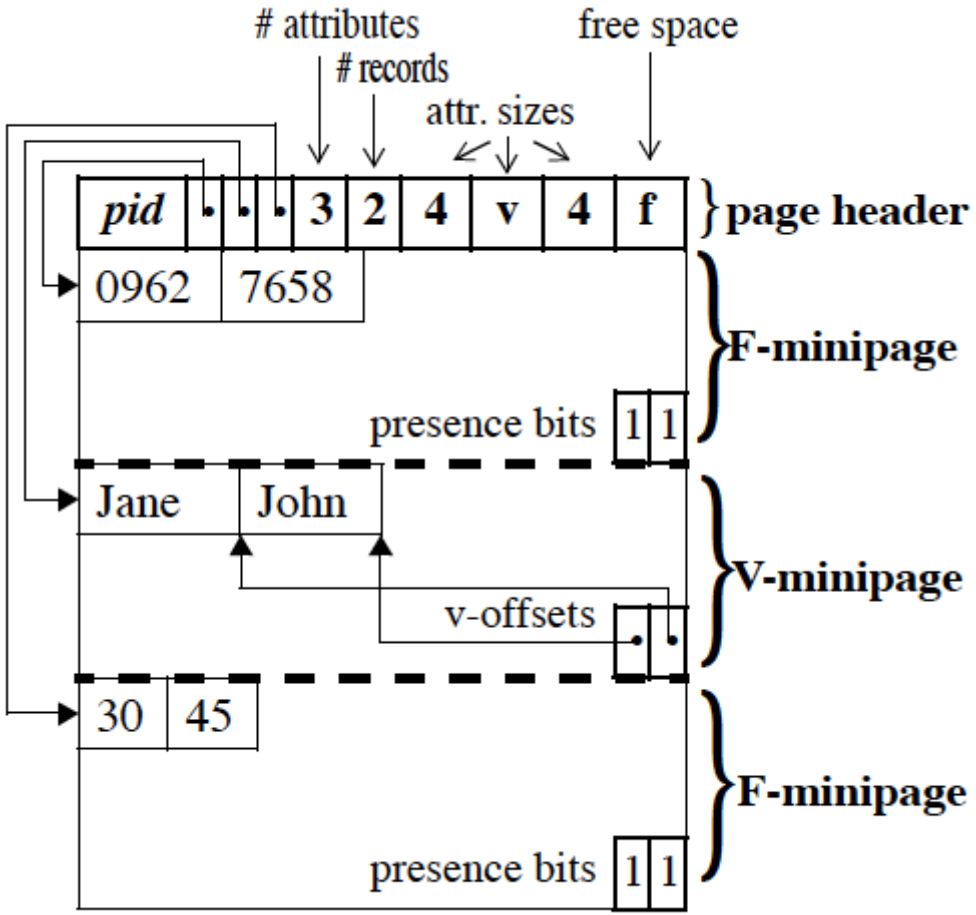
Outline

- File Storage
- Page Layout
  - NSM, aka row-oriented
  - DSM, aka column-oriented
  - PAX, a hybrid solution
- Buffer Management

Partition Attributes Across (PAX)

Decompose a slotted-page internally in mini-pages per attribute

- ✓ Cache-friendly
- ✓ Compatible with slotted-pages
- ✓ Retain NSM I/O pattern
  - ✓ No column “stitching”
  - ✓ No per-column tuple ids
- ✓ Brings only relevant attributes to cache



PAX Americana

- DSM most suitable for analytical queries, but required major rewrites of existing DBMS, and penalized transactions **a lot**.

- PAX can replace NSM in-place
  - Oracle moved to PAX
  - So did most Hadoop-oriented file formats
    - Parquet
    - Arrow
    - …

Conclusion

- File & Page layouts
- Row stores
  - Transactions
  - Frequent inserts/updates/deletes
- Column stores
  - Data analytics, data exploration
  - Mostly read-only data
  - Most queries access very few attributes

One size does not fit all:
Different workloads require different storage layouts and data access methods

Reading material

- Row stores: COW Book chapter 8 (material of CS322)
- D. Abadi et al.: The Design and Implementation of Modern Column-Oriented Database Systems. Foundations and Trends in Databases, vol. 5, no. 3, **pp. 227-263 only**, 2013. Available online at: http://db.csail.mit.edu/pubs/abadi-column-stores.pdf
- I. Alagiannis, S. Idreos, A. Ailamaki: H2O: A hands-free adaptive store. SIGMOD'14. Available online at: http://dl.acm.org/citation.cfm?doid=2588555.2610502

Optional readings

- The remainder of: "The Design and Implementation of Modern Column-Oriented Database Systems"