Storage (continued)

R&G Chapter 9

(slides adapted from content by J.Gehrke, J.Shanmugasundaram, and/or C.Koch)

Oracle accounts available!

Email okennedy@buffalo.edu with your UBIT if interested

Class recordings posted on Piazza

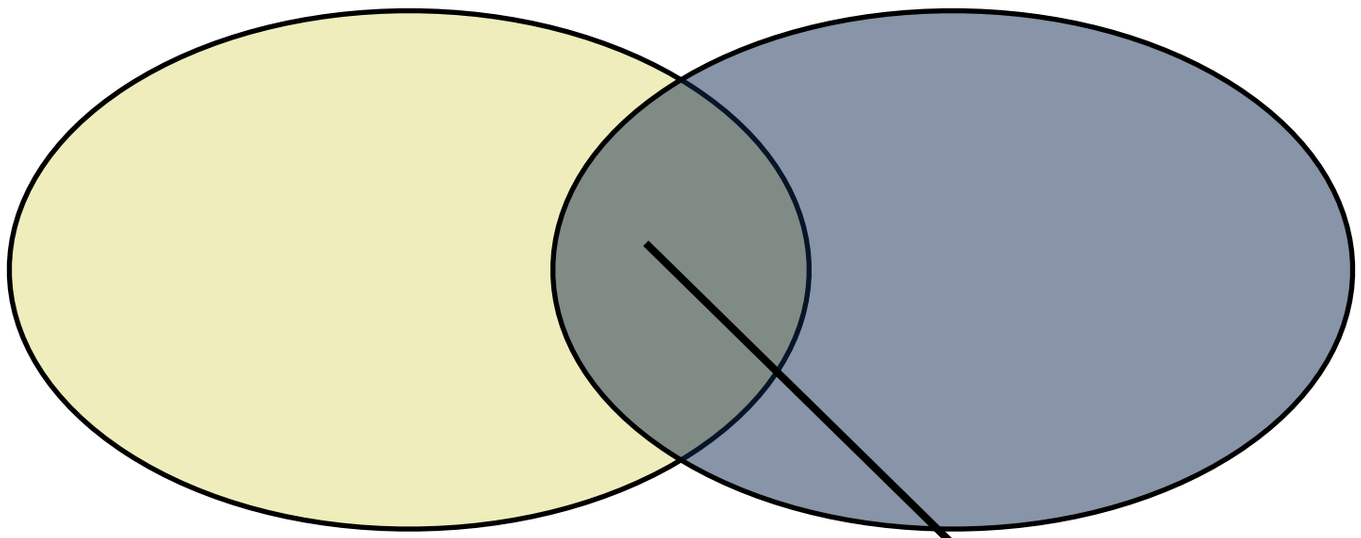
Homework 1 due tonight at 11:59 PM

Homework 2 posted later tonight

Review+

Officers O
(Name, OfficerID)

Visited V
(OfficerID,Planet)

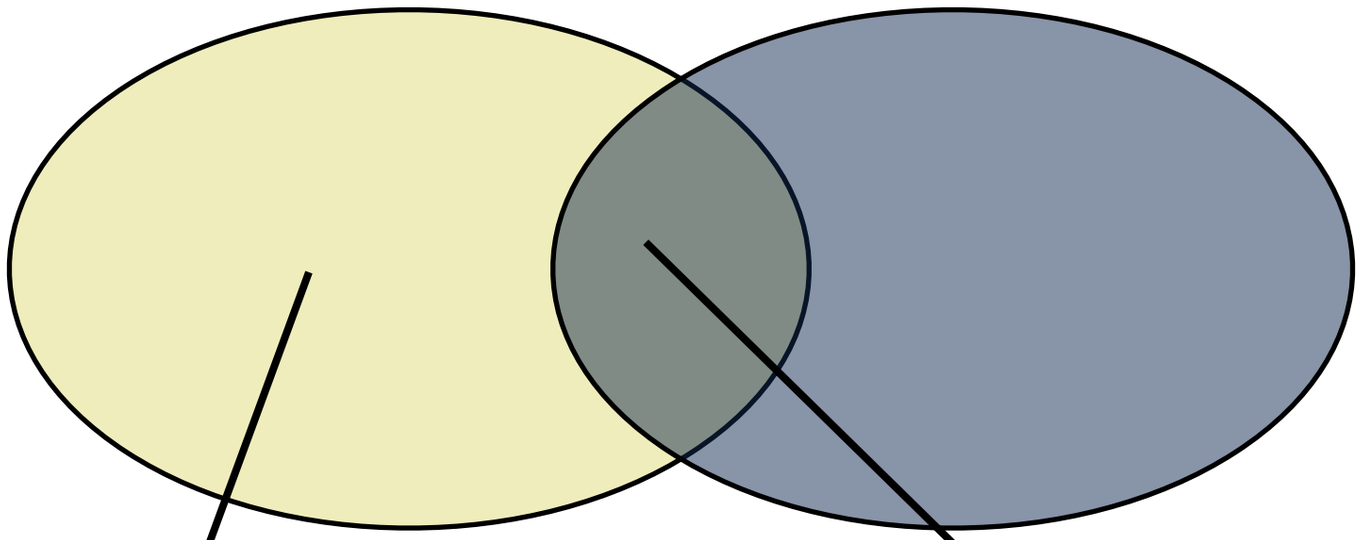


All pairs <O,V>

Review+

Officers O
(Name, OfficerID)

Visited V
(OfficerID,Planet)



All tuples <O, NULL>
(where O is unmatched)

All pairs <O,V>

What if we want the result to include officers who have never visited a planet?

Review+

Officers O
(Name, OfficerID)

```
Kirk,     1
Spock,    2
Redshirt,3
```

Visited V
(OfficerID,Planet)

```
1, Vulcan
1, Earth
2, Vulcan
```

Result

```
Kirk,      Vulcan
Kirk,      Earth
Spock,     Vulcan
Redshirt,  NULL
```

Find the Relational Algebra expression to compute this

Review+

Normal Join Results

$$[\pi_{\texttt{Name},\texttt{Planet}}(O \bowtie V)]$$

$$\cup$$

$$[\pi_{\texttt{Name},\texttt{NULL}\rightarrow\texttt{Planet}}(((\pi_{\texttt{OfficerID}}O) - (\pi_{\texttt{OfficerID}}V)) \bowtie O)]$$

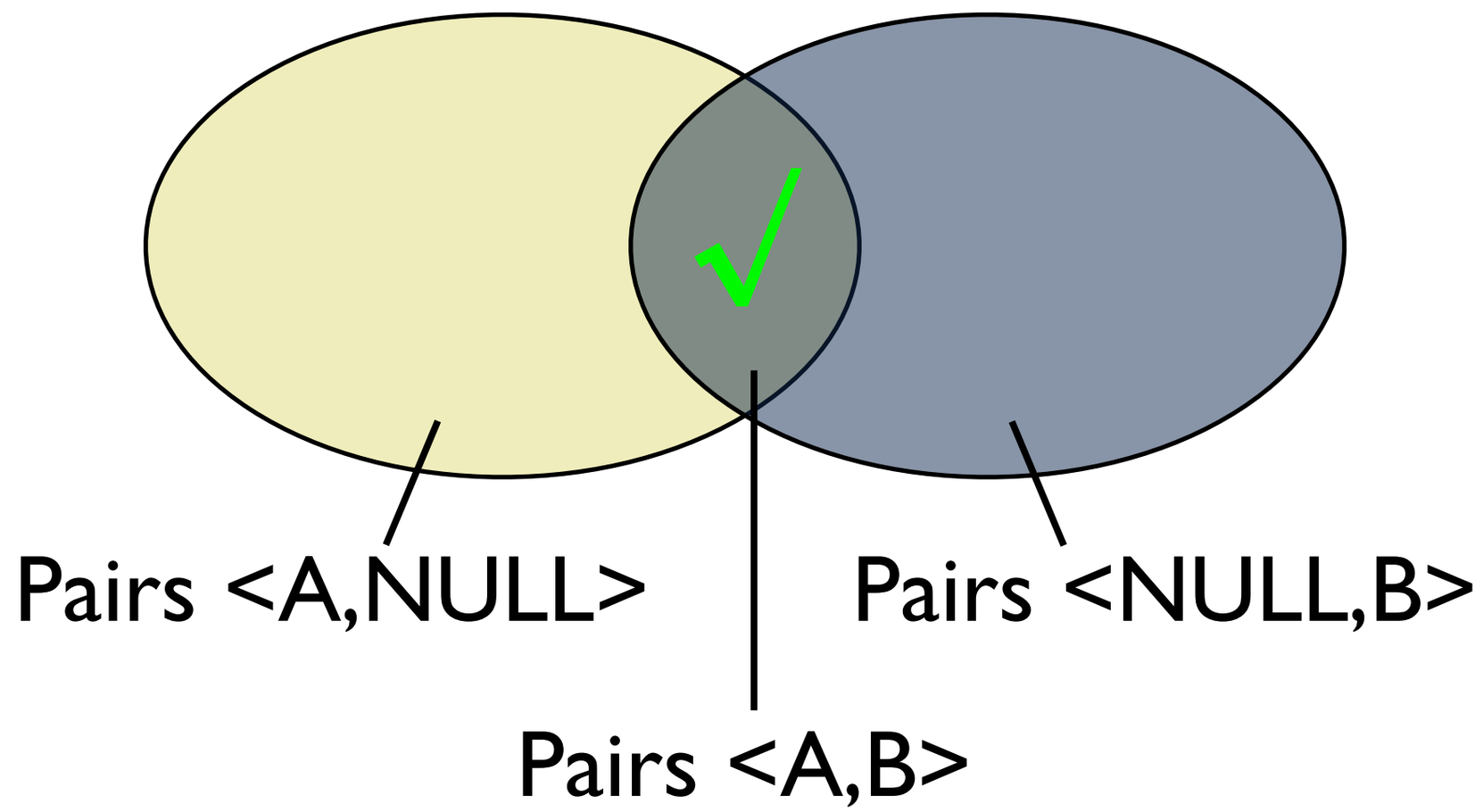
Extend with NULL Unjoined OfficerIDs

Review+

This occurs often enough that we have a name for it:
{LEFT | RIGHT | FULL} OUTER JOIN

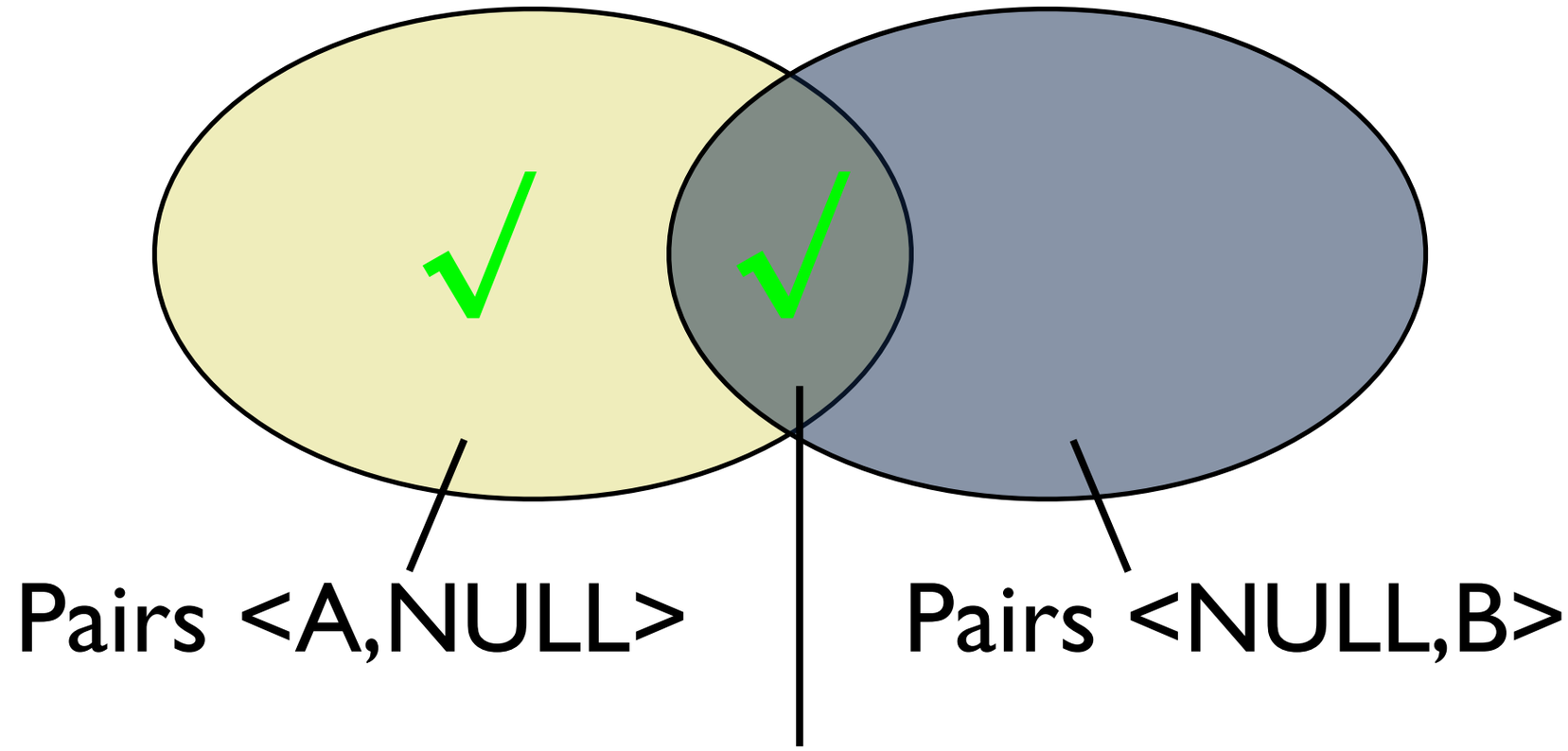
```
SELECT O.Name, V.Planet
FROM Officers O LEFT OUTER JOIN
     Visited V
ON O.OfficerID = V.OfficerID
```

Review+



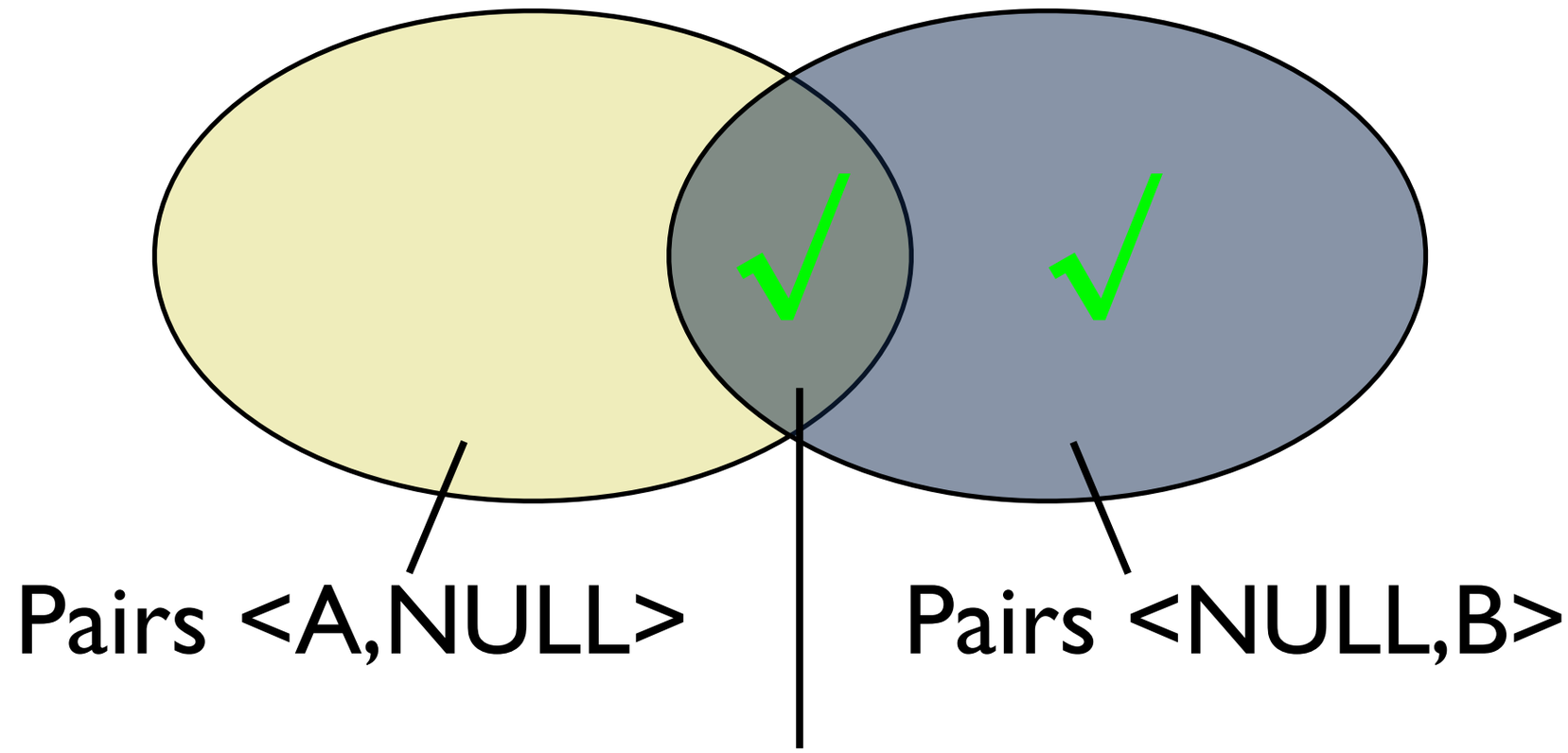
Review+

LEFT OUTER JOIN



Review+

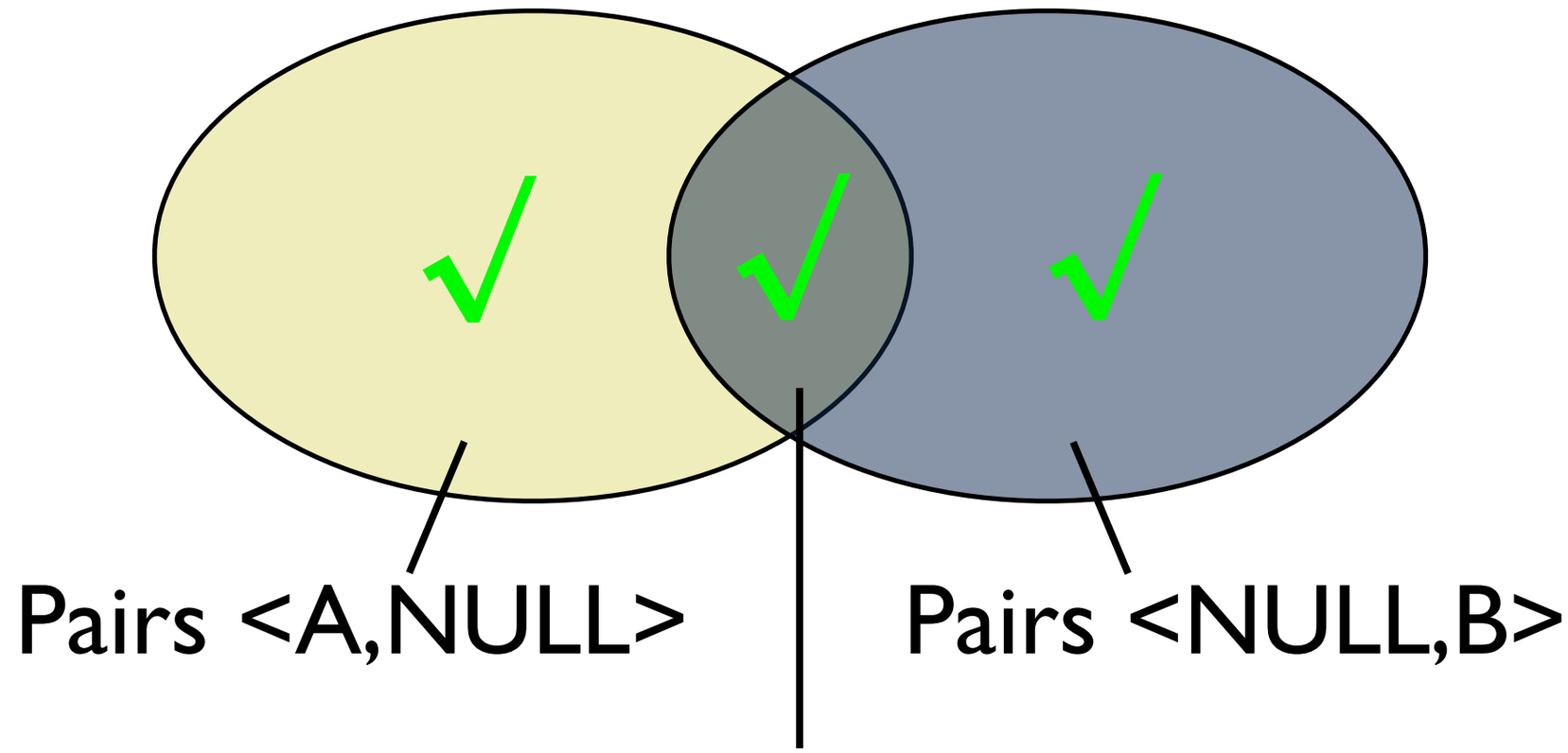
RIGHT OUTER JOIN



Pairs <A,B>

Review+

FULL OUTER JOIN



Review

- Computations are done on data in RAM
- Data might be stored on a HDD/SDD
  - Why?
- Impedance mismatch between access granularities
  - HDD/SDDs operate on **pages**.
  - Queries operate on **records**.

Review

- Computations are done on data in RAM
- Data might be stored on a HDD/SDD
  - Why? Data that needs persistence/doesn't fit in RAM
- Impedance mismatch between access granularities
  - HDD/SDDs operate on **pages**.
  - Queries operate on **records**.

Files and Data

- A **File** is a collection of pages
  - A **Page** is a collection of records
    - A **Record** is a data value (e.g., a tuple)
- We need an infrastructure to ensure that records we need are in memory.
- We need some way to organize and store files, pages, and records.

The Buffer Manager

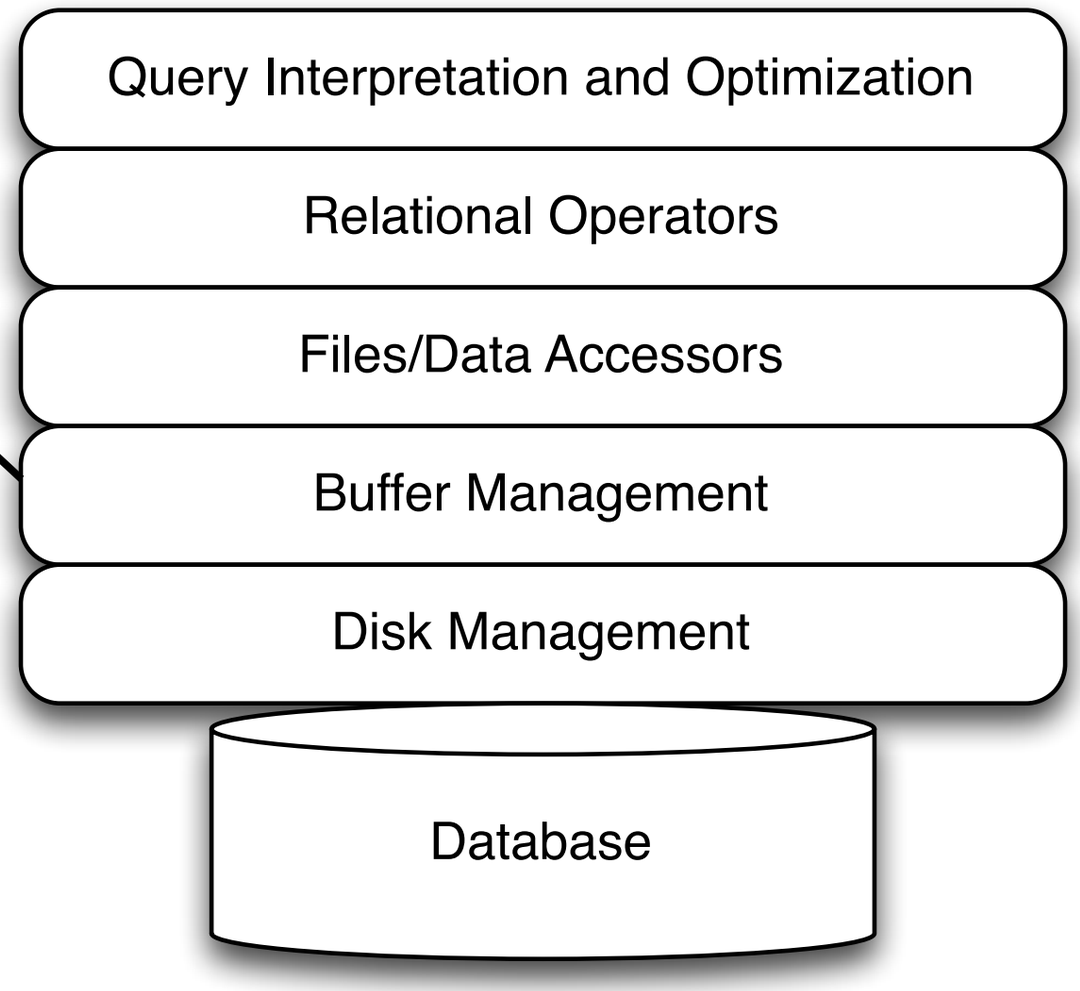
API

Allocate a page

Deallocate a page

Read from a page

Write to a page



The Buffer Manager

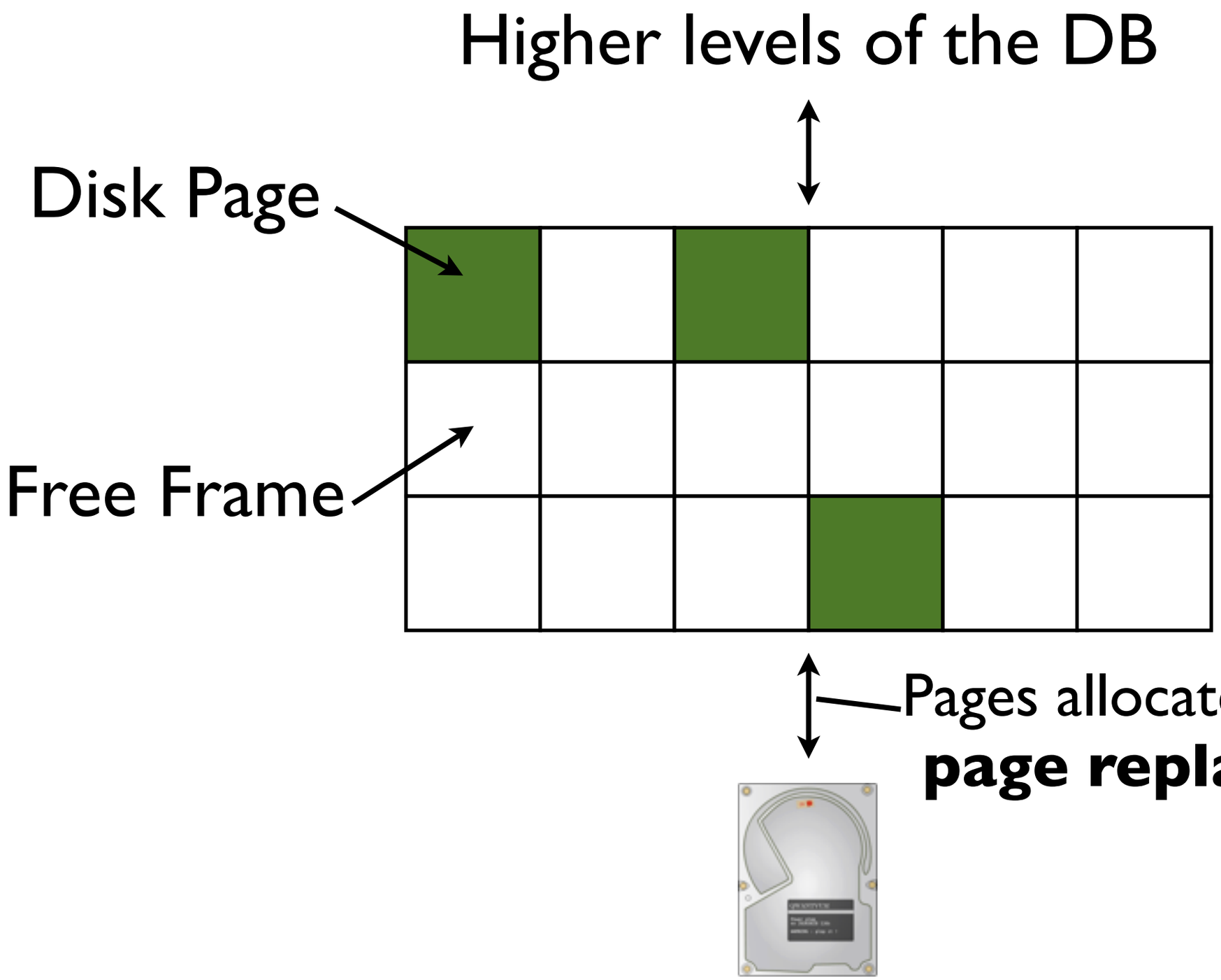


image credit: openclipart.org

When a page is requested

- Is the page in the buffer pool?
  - Yes? **Pin** the page and return the address.
- Otherwise pick a frame for replacement
  - If the frame is dirty, write it to disk
  - Read requested page into chosen frame
  - **Pin** the page and return its address.

When a page is requested

- Is the page in the buffer pool?
  - Yes? **Pin** the page and return the address.
- Otherwise pick a frame for replacement
  - If the frame is dirty, write it to disk
  - Read requested page into chosen frame
  - **Pin** the page and return its address.

Pages can be prefetched by requesting several pages at a time.

Pinned Pages

- Pinning a page indicates that it is being used.
- The requestor must unpin the page when done.
  - The requestor must also indicate whether the page has been modified (with a 'dirty' bit)
  - Dirty pages must be written to disk
- Pages may be requested multiple times
  - Use a pin count (reference count) to keep track.
- Concurrency Control/Recovery may require other operations when replacing a frame.

Buffer Replacement

- Frames are chosen for replacement by a **buffer replacement policy**.
  - (e.g., LRU, MRU, Clock)
- Policy can have a big impact!
  - Depends on the access pattern.
- What is a worst-case scenario for LRU?

Why not let the OS handle it? (e.g., Virtual Memory)

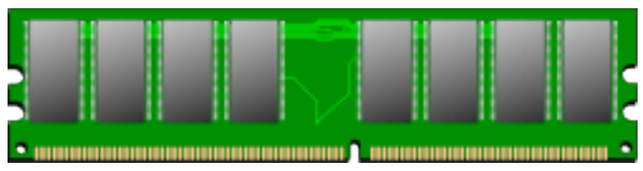
- Portability issues (differences in OS support)
- Minor Limitations (e.g., files can't span disks)
- DB accesses have specific access patterns.
  - Fine tuned replacement policies.
  - Query-aware pre-fetching.
- "Virtual memory" may buffer write requests.

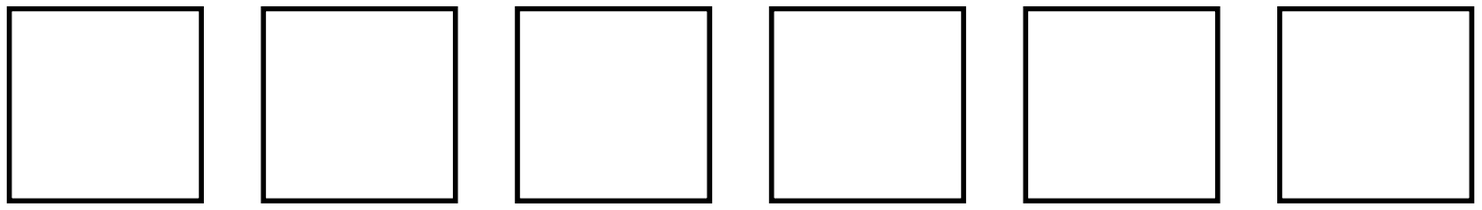
Example

```
SELECT *
FROM Officers O
WHERE O.Ship = ‘1701A’
```

… where ‘Officers’ is sorted by ‘Ship’

Example-OS Paging





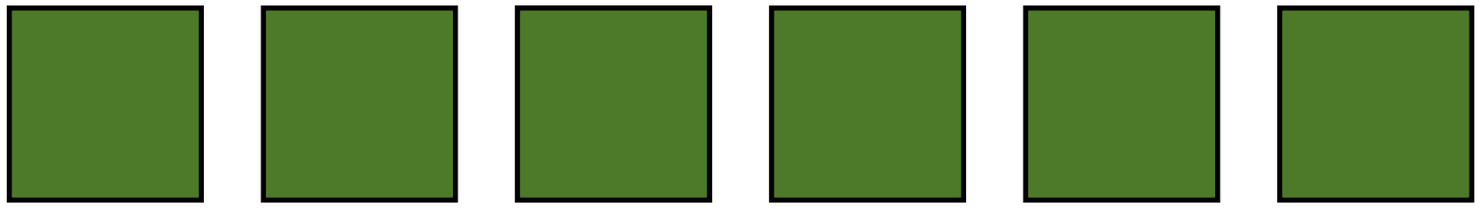
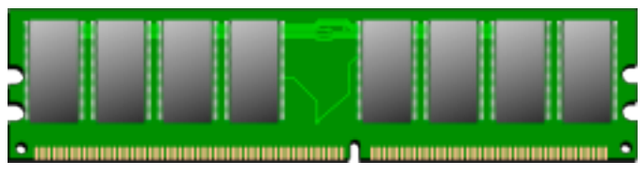
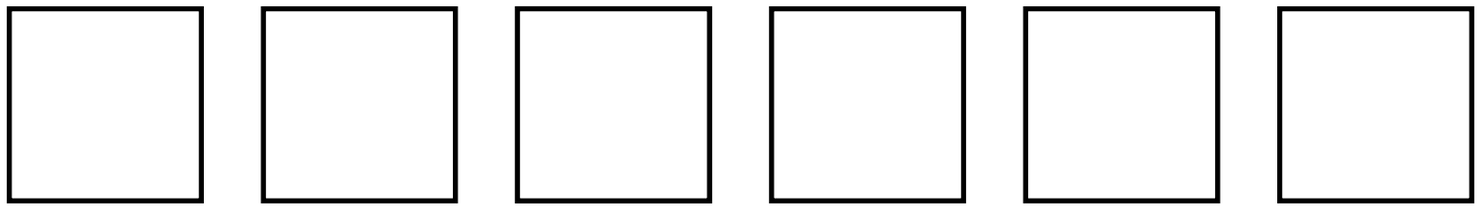
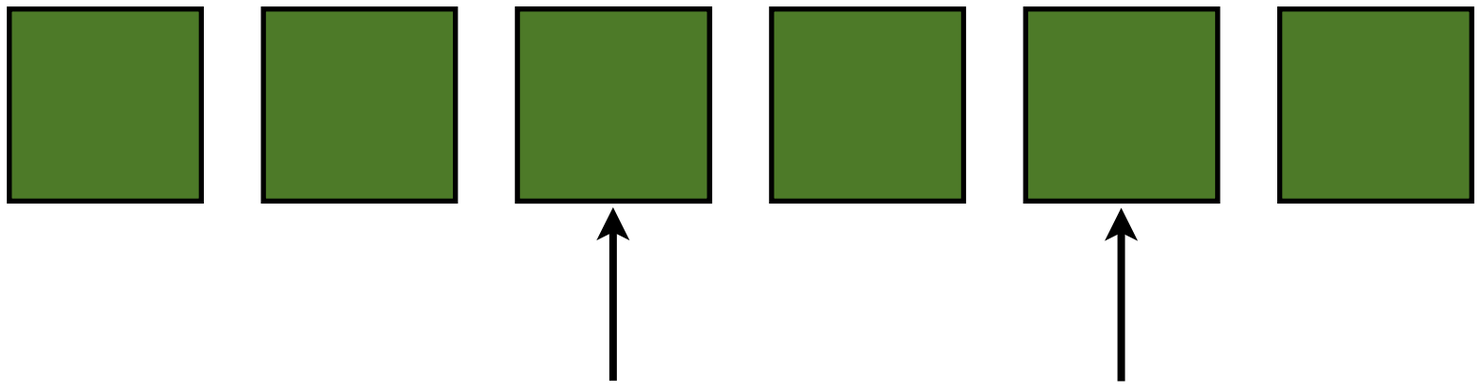


image credit: openclipart.org

Example-OS Paging



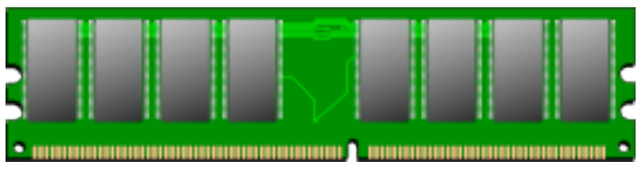


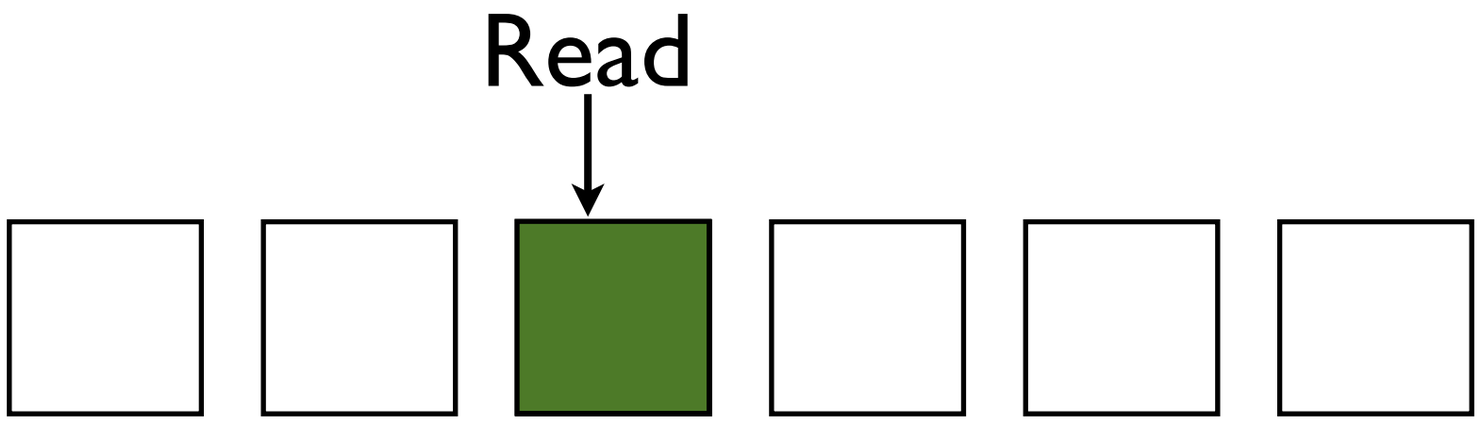


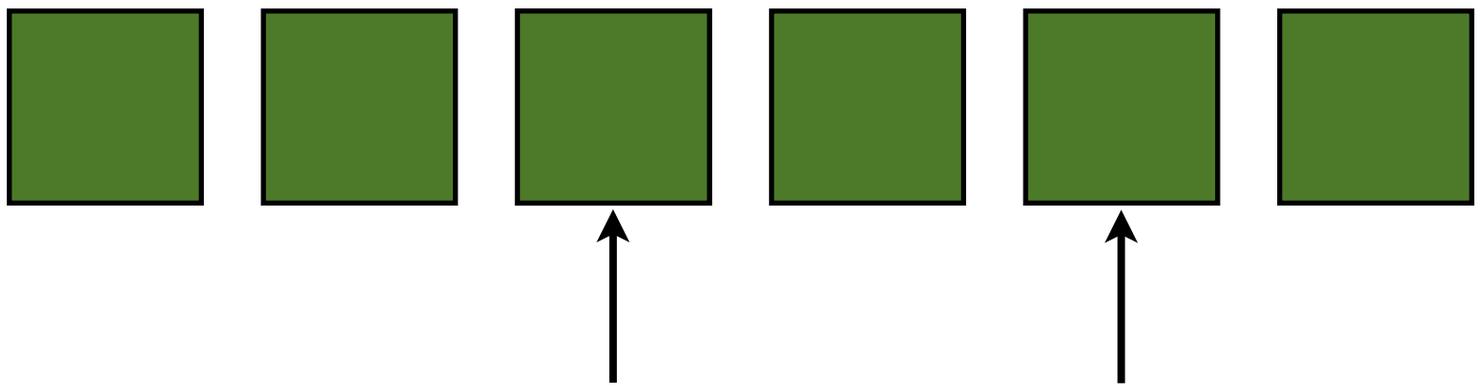
Index: ‘Enterprise officers in this range’

image credit: openclipart.org

Example-OS Paging



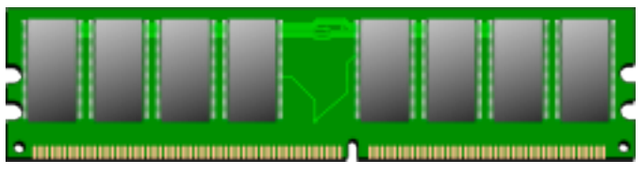


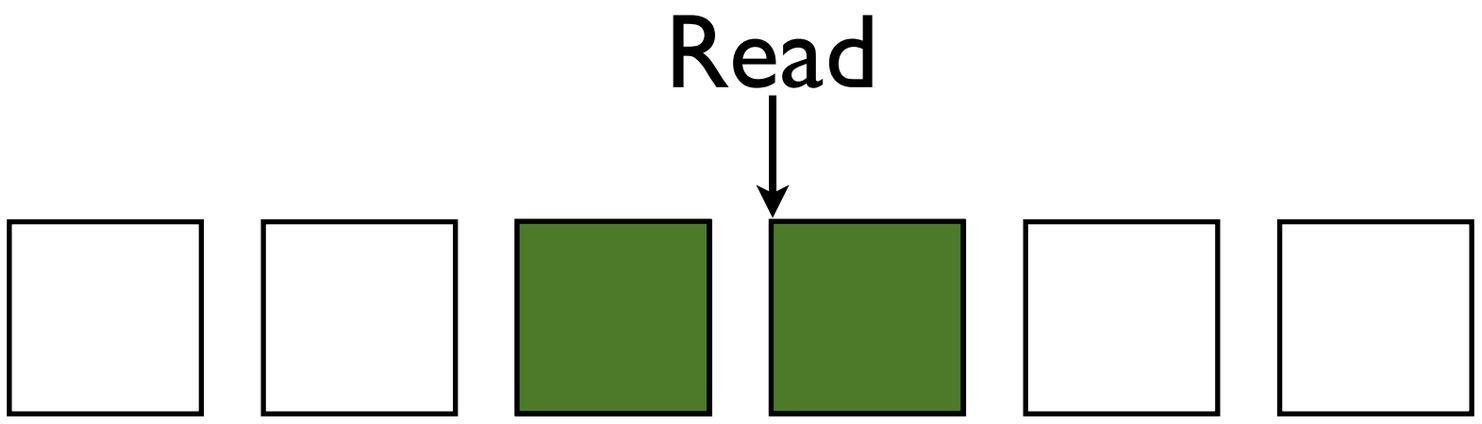


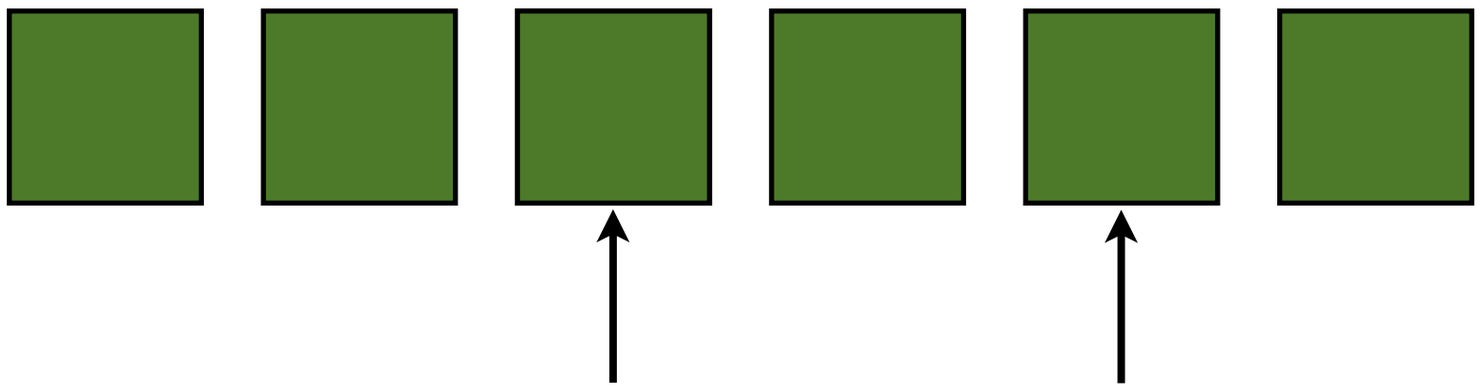
Index: ‘Enterprise officers in this range’

image credit: openclipart.org

Example-OS Paging



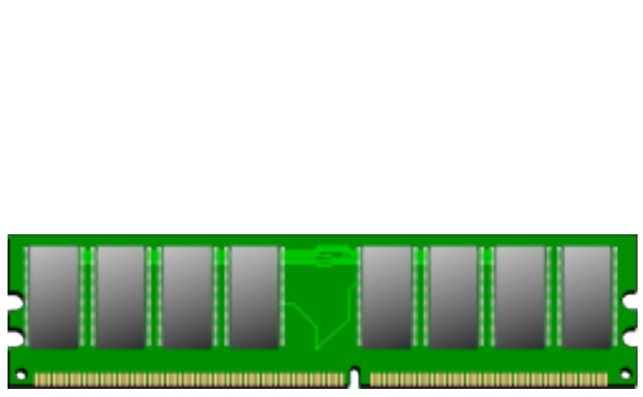


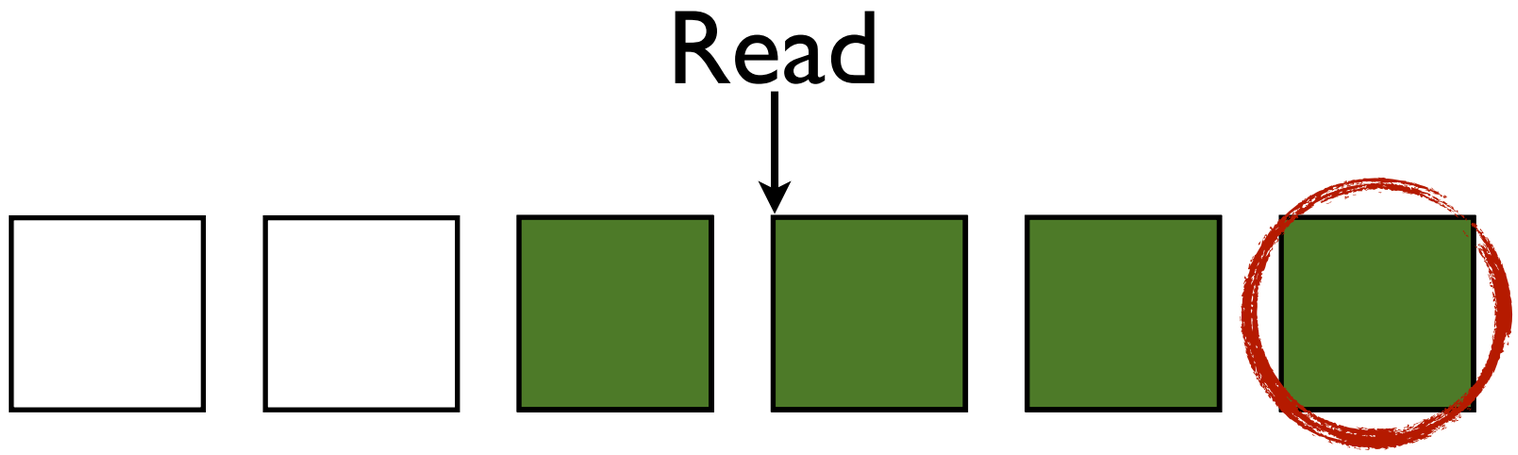


Index: ‘Enterprise officers in this range’

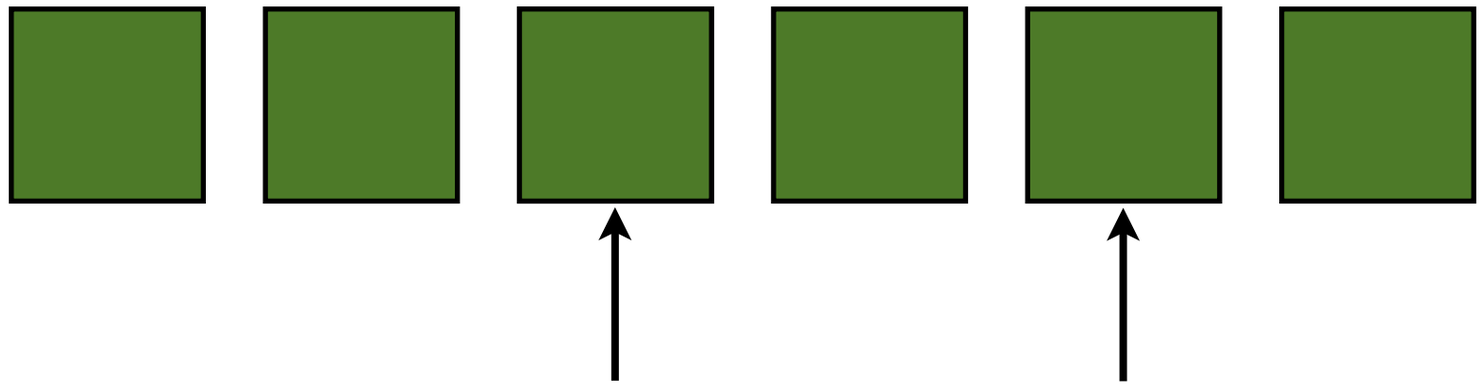
image credit: openclipart.org

Example-OS Paging





Aha, a Sequential Read!

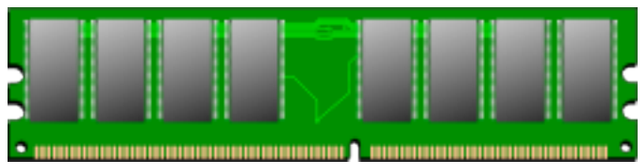


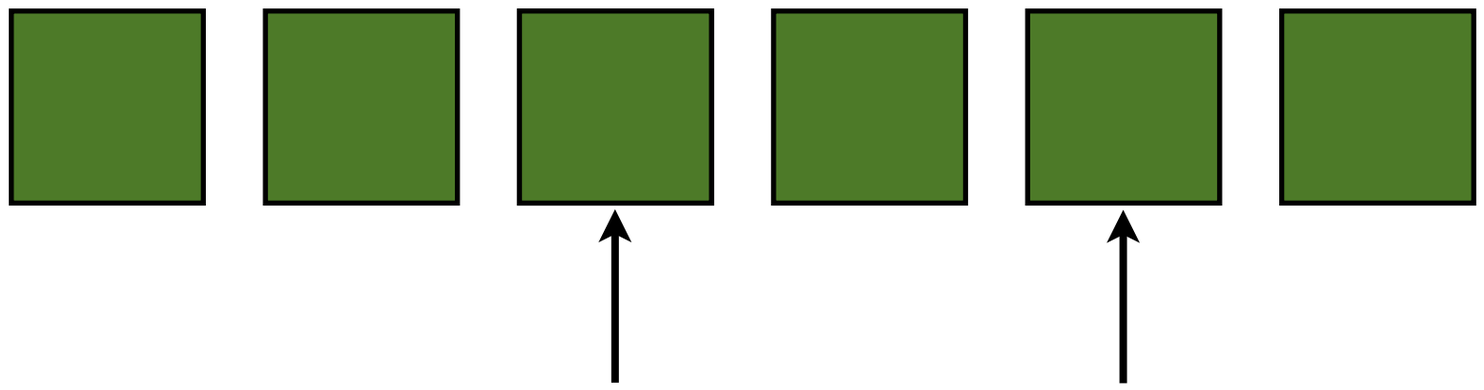
Index: ‘Enterprise officers in this range’

image credit: openclipart.org

Example-DB Paging

Read

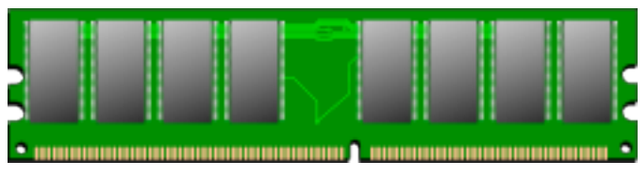


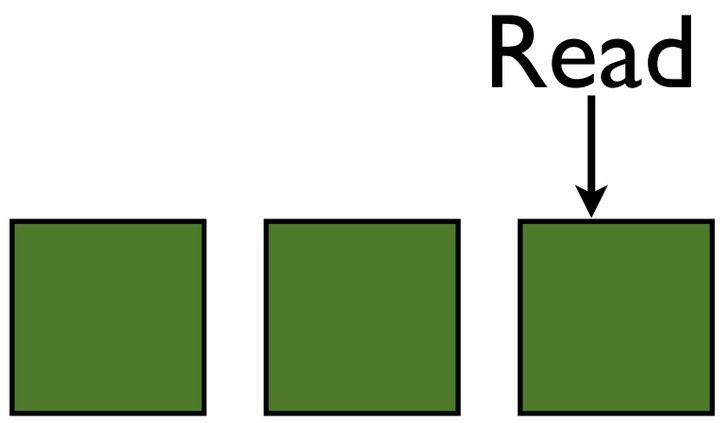


Index: ‘Enterprise officers in this range’

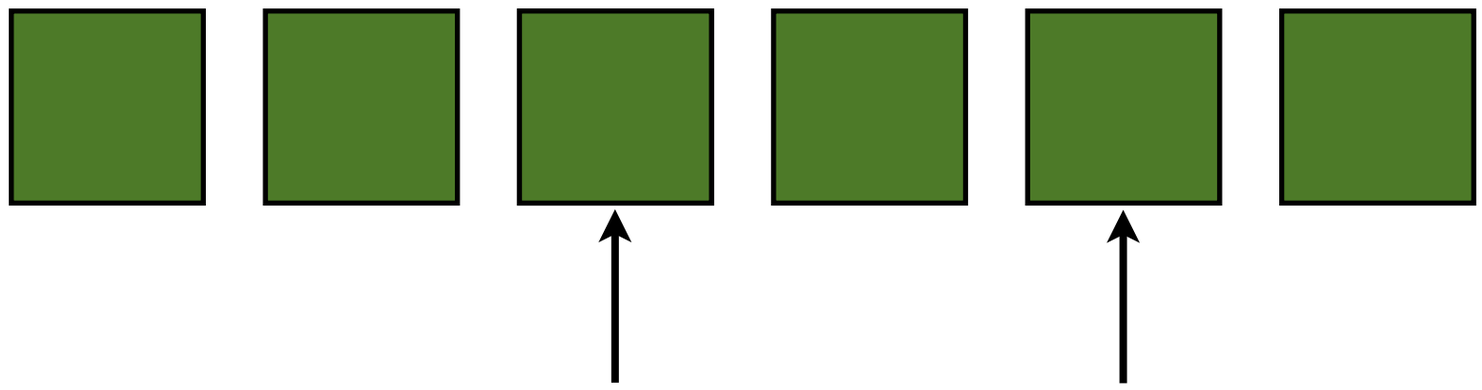
image credit: openclipart.org

Example-DB Paging





The DB knows exactly what to read!



Index: 'Enterprise officers in this range'

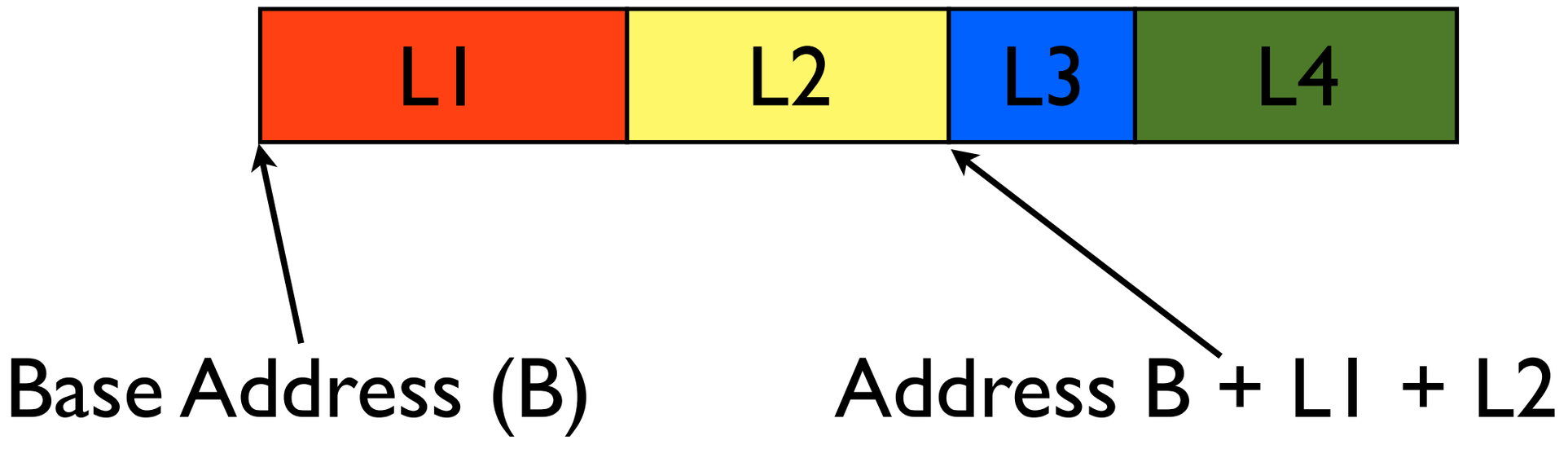
image credit: openclipart.org

Data Organization

- How do we store data?
  - How are records represented on-disk? (Serialization)
  - How are records stored within a page?
  - How are pages organized in a file?
  - What other metadata do we need?
- Our solutions must also be persisted to disk.

Record (Tuple) Formats

- Fixed Length Records



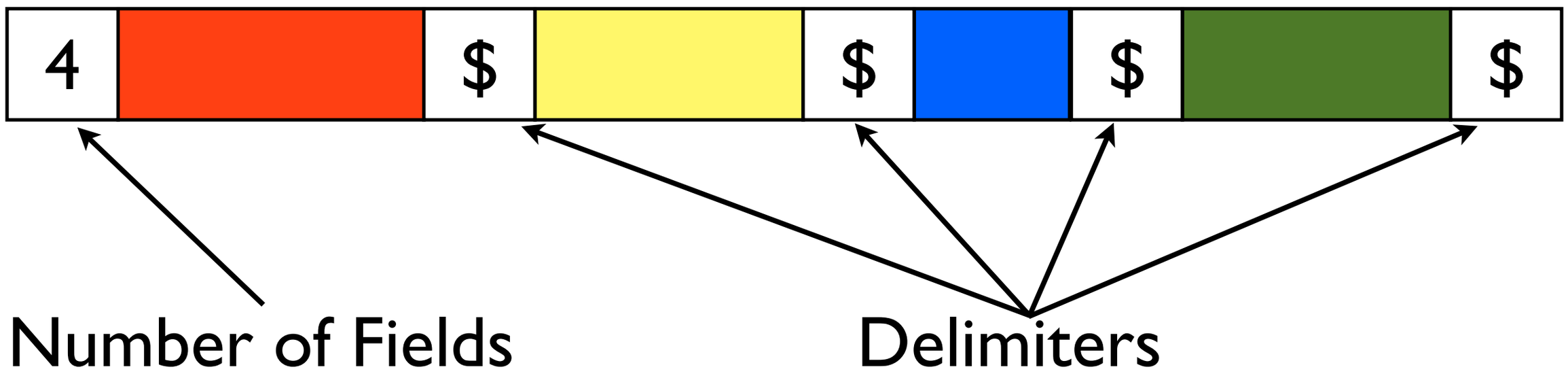
Record information stored in **System Catalog**

What are some advantages/disadvantages of storing records this way?

- Information about field types same for all records in a file; stored in *system catalogs*.
- Finding *i'th* field does not require scan of record (constant (fast) time).
- Constant size records can mean wasted space

Record (Tuple) Formats

- Delimited Records



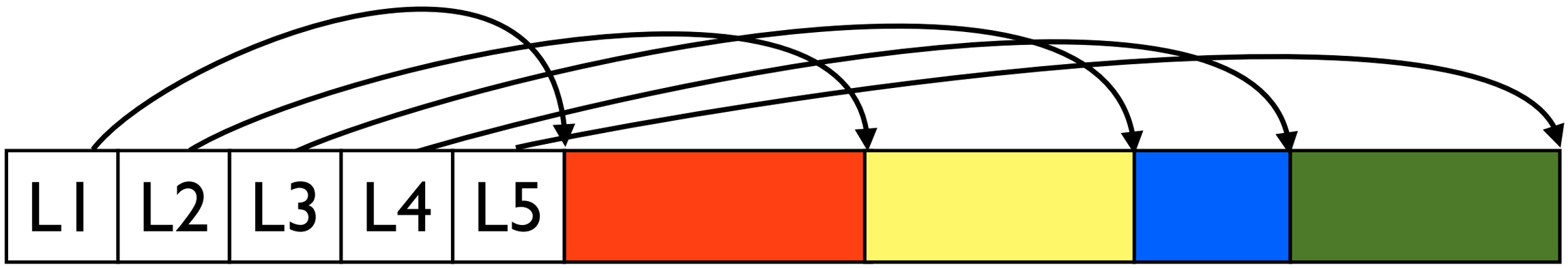
What are some advantages/disadvantages of storing records this way?

Friday, January 25, 13

Minimal storage space used for records
Easy for humans to read (e.g., CSV)
Sanity checking! 4 fields = 4 delimiters -- can detect buffer overflows.
Record data must guarantee that delimiter character never occurs
Finding field n requires scanning all prior characters

Record (Tuple) Formats

- Self-Describing Records

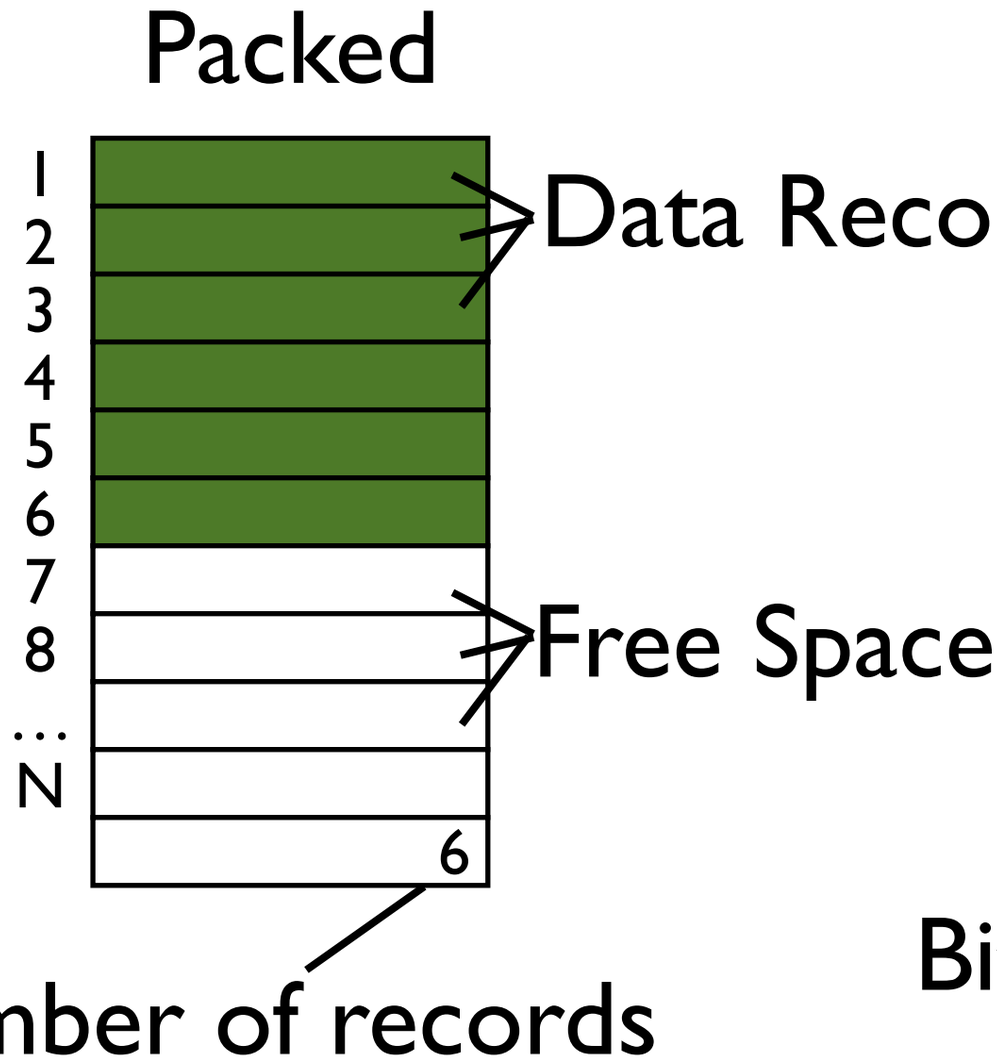


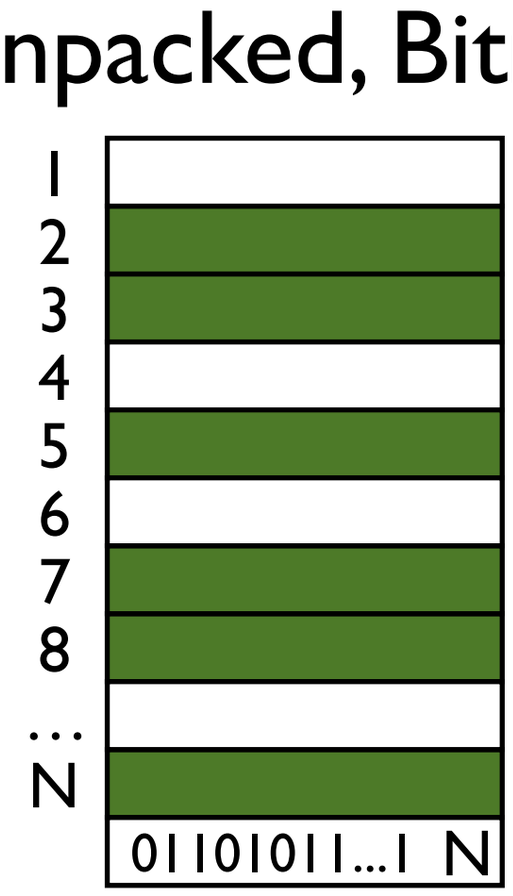
Array of Field Offsets

What are some advantages/disadvantages of storing records this way?

* Offers direct access to i'th field, efficient storage of *nulls* (special *don't know* value); small directory overhead.
* Susceptible to buffer overflows
* Harder for humans to read

Page Formats



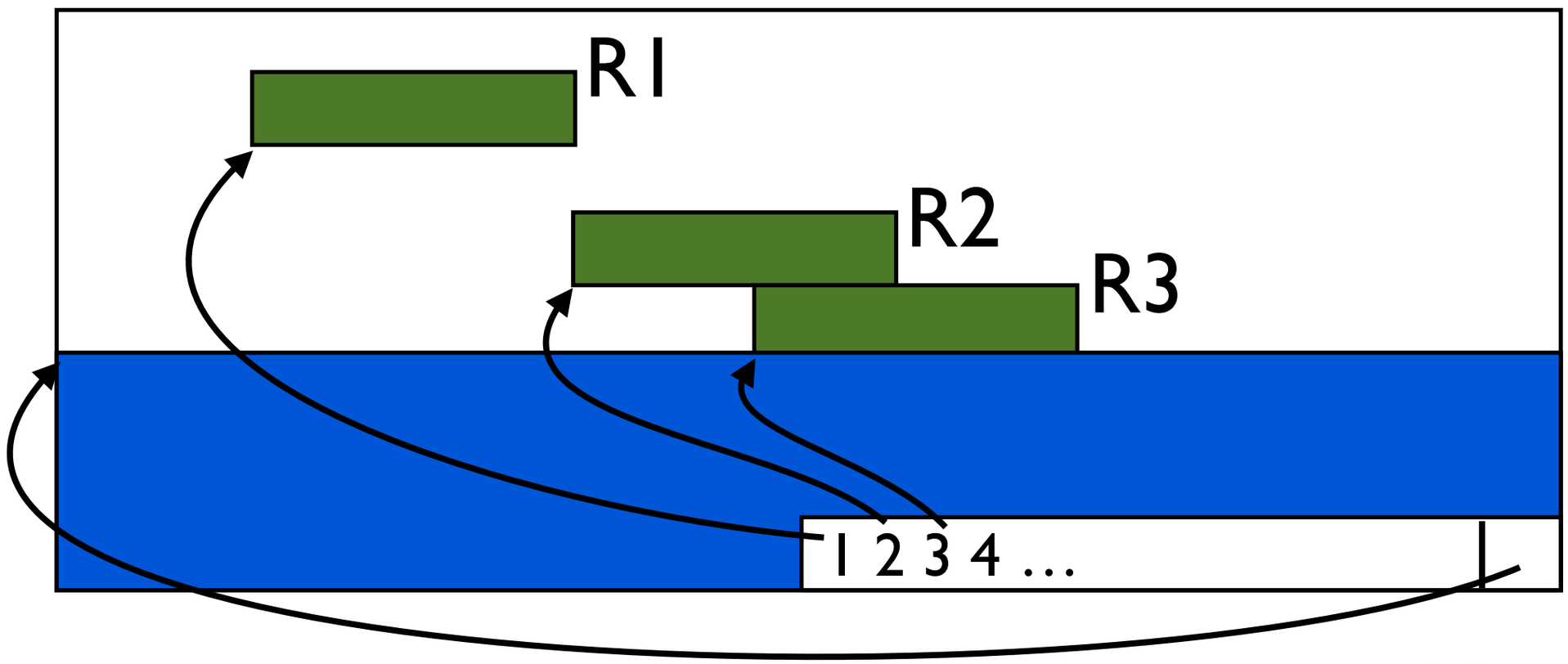


What are advantages/disadvantages of these formats?

- *Both require records of fixed size (or at least maximal length)*
- *Packed*
  - *Easier to find free slots*
  - *Less memory to access when doing full page scans*
  - *Can't support data layouts that reference a record's location in a page (need a mapping from record id to page location)*
  - *Deletions are expensive (especially if data is stored in sorted order)*
- *Unpacked*

Page Formats

Variable Size Records



Pointer to start of free space

What are advantages/disadvantages of this format?

Friday, January 25, 13

Flexible. Don't need fixed/maximal record sizes.
Deletions are cheap
Natural incorporation of mappings from Record ID to location in page
Lots of wasted space -- deletions don't get filled in
(need a garbage collection step to compact pages)
Hard to get optimal use of cache lines

Files of Records

IO is done at the Page/Block level

… but queries are done at the Record level

File: A collection of pages of records that must support:

Insert/Delete/Update a record
Read a record (using record ID)
Scan all records (possibly with some condition)

Unordered (Heap) Files

Store records in no particular order

Disk pages are allocated/freed as file grows and shrinks

Support for record level operations by:

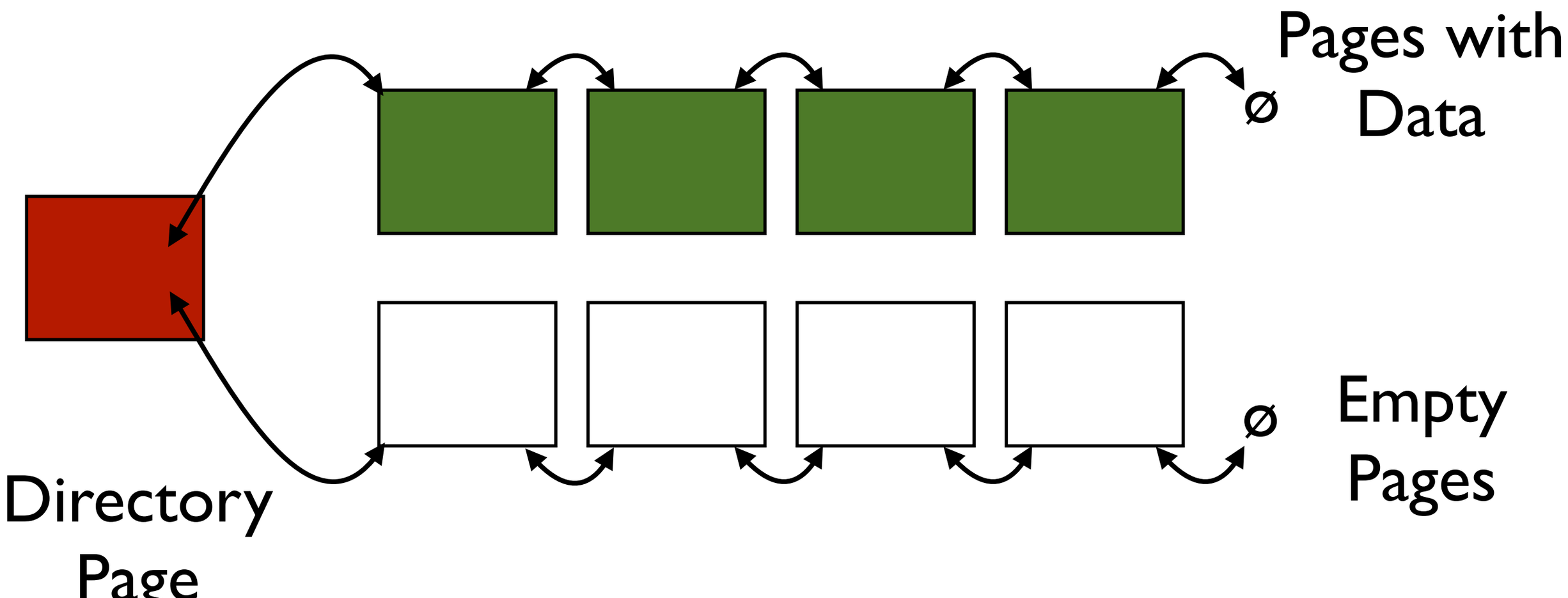
Keeping track of pages in the file

Keeping track of free space in each page

Keeping track of records on each page

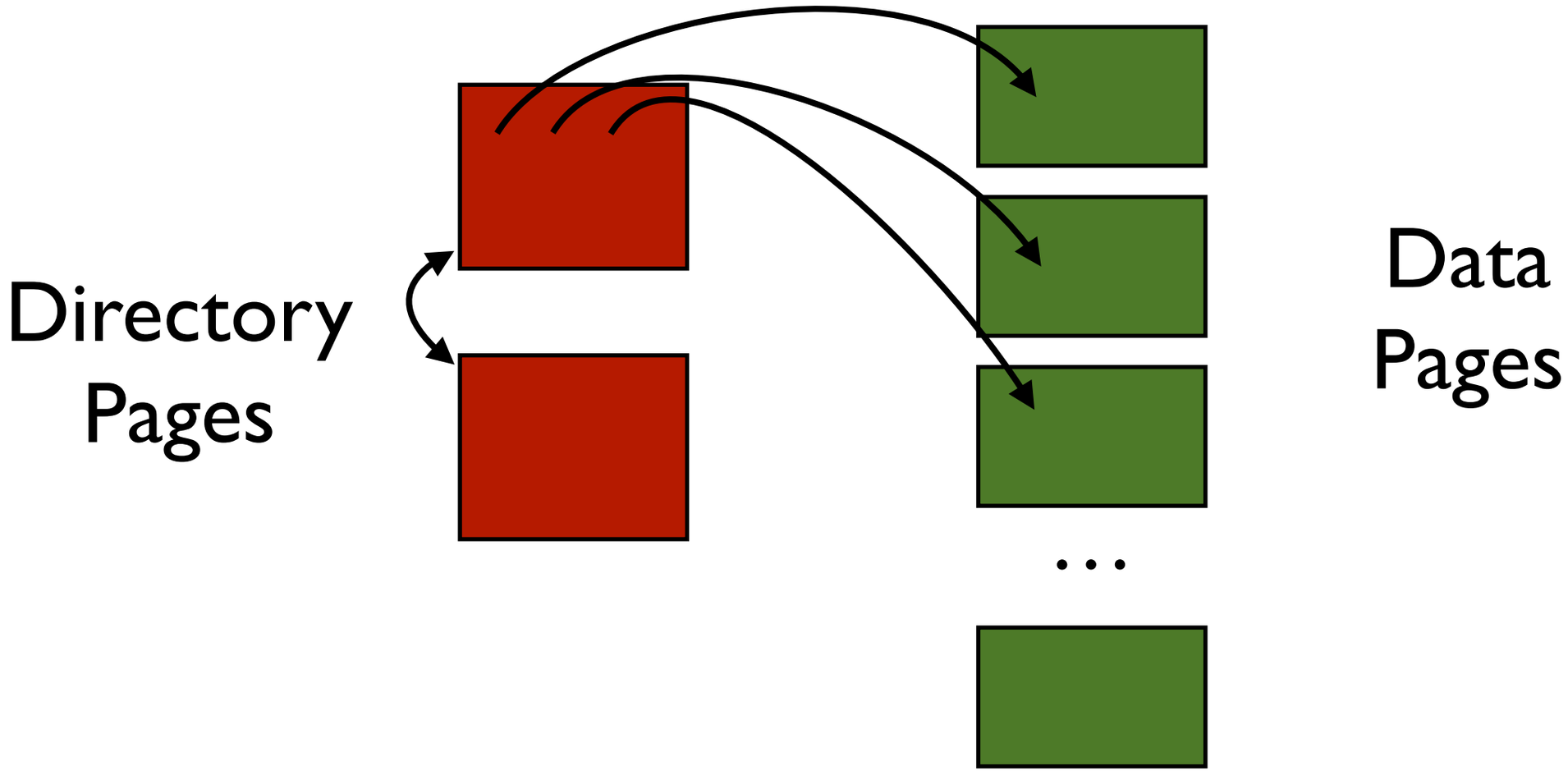
This data must be stored somewhere!

Unordered (Heap) Files

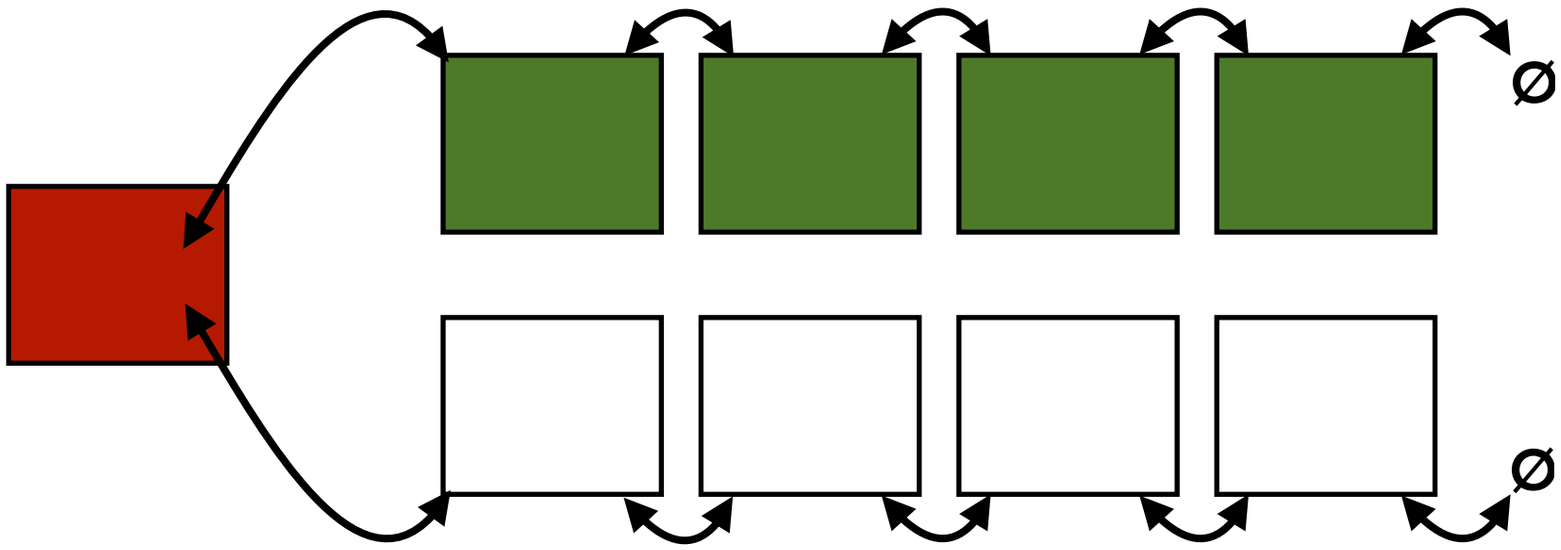


Each page contains 2 pointers plus data

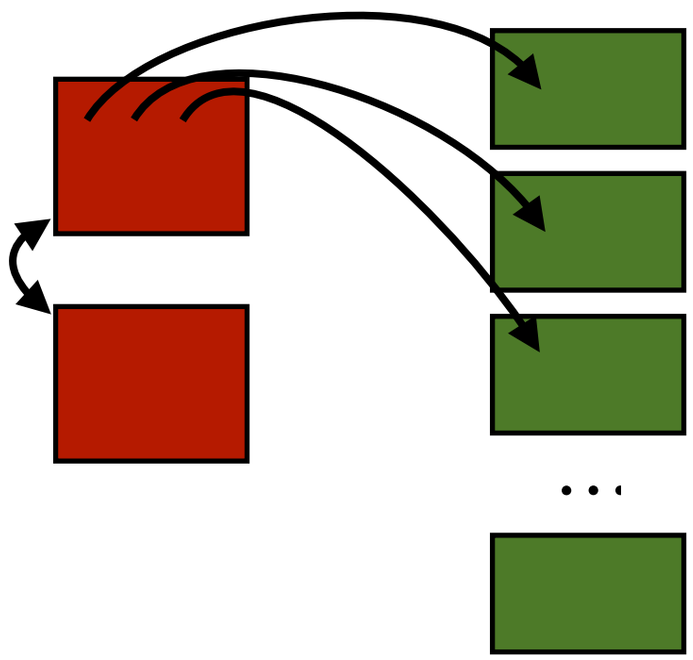
Unordered (Heap) Files



Directories are a collection of pages (e.g., a linked list)
Directories point to all data pages
(entries can include # of free pages)



What are the advantages and disadvantages of each?



Linked list structure only supports full table scan access.
Use of directory pages makes the file bigger, but can be extended to support clustered/ sorted files.

System Catalog

- For each index:
  - Type of index, Attribute(s) indexed
- For each relation
  - Name, filename, file structure
  - Attribute/field names/types
  - Data modeling information
- And other information
  - View Definitions
  - Stored Procedures

The catalog is itself a relation!

Attr_Cat

Attr_Name,	Rel_Name,	Type,	Position
Attr_Name,	Attr_Cat,	string,	1
Rel_Name,	Attr_Cat,	string,	2
Type,	Attr_Cat,	string,	3
Position,	Attr_Cat,	integer,	4
FirstName,	Officers,	string,	1
LastName,	Officers,	string,	2
Ship,	Officers,	integer,	3
Rank,	Officers,	decimal,	4
Name,	Ships,	string,	1
ID,	Ships,	integer,	2

…

Summary

- Buffer manager brings pages into RAM
  - Page stays in RAM until freed by requestor.
  - Written to disk when frame is replaced.
  - Choice of frame based on replacement policy.
- Different storage formats useful in a variety of cases.
  - Both record and fields can be stored in different ways.

Summary

- File layer tracks pages in each file, and tracks pages with free space.
- Indexes support efficient retrieval of records based on values of certain fields
  - More on this later
- Catalog relations store information about relations, indexes, etc…