>>

Page Internals

- Pages
- Page Formats
- Page Formats
- Storage Utilisation
- Overflows

Λ >>

❖ Pages

Database applications view data as:

- a collection of records (tuples)
- records can be accessed via a **`TupleId`**/**`RecordId`**/**`RID`**
- **`TupleId`** = (**`PageID`** + **`TupIndex`**)

The disk and buffer manager provide the following view:

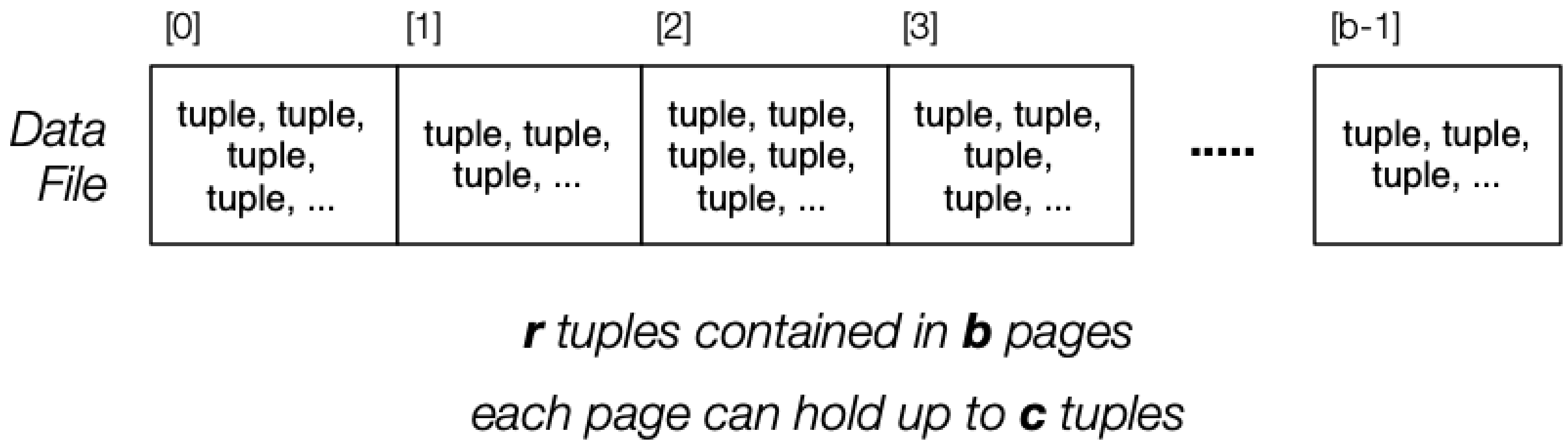
- data is a sequence of fixed-size pages (aka "blocks")
- pages can be (random) accessed via a **`PageID`**
- each page contains zero or more tuple values

Page format = how space/tuples are organised within a page

<< ^ >>

❖ Pages (cont)

Data files consist of pages containing tuples:


[0]
[1]
[2]
[3]
[b-1]
Data File
tuple, tuple, tuple, tuple, ...
tuple, tuple, tuple, ...
tuple, tuple, tuple, tuple, tuple, ...
tuple, tuple, tuple, tuple, ...
.....
tuple, tuple, tuple, ...
r tuples contained in b pages
each page can hold up to c tuples

Each data file (in PostgreSQL) is related to one table.

<< ∧ >>

❖ Page Formats

Ultimately, a **`Page`** is simply an array of bytes (**`byte[]`**).

We want to interpret/manipulate it as a collection of **`Record`**s (tuples).

Tuples are addressed by a record ID (**`rid`** = **`(PageId,TupIndex)`**)

Typical operations on **`Page`**s:

- **`request_page(pid)`** ... get page via its **`PageId`**
- **`get_record(rid)`** ... get record via its **`TupleId`**
- **`rid = insert_record(pid,rec)`** ... add new record
- **`update_record(rid,rec)`** ... update value of record
- **`delete_record(rid)`** ... remove record from page

<< ^ >>

❖ Page Formats (cont)

Page format = tuples + data structures allowing tuples to be found

Characteristics of **`Page`** formats:

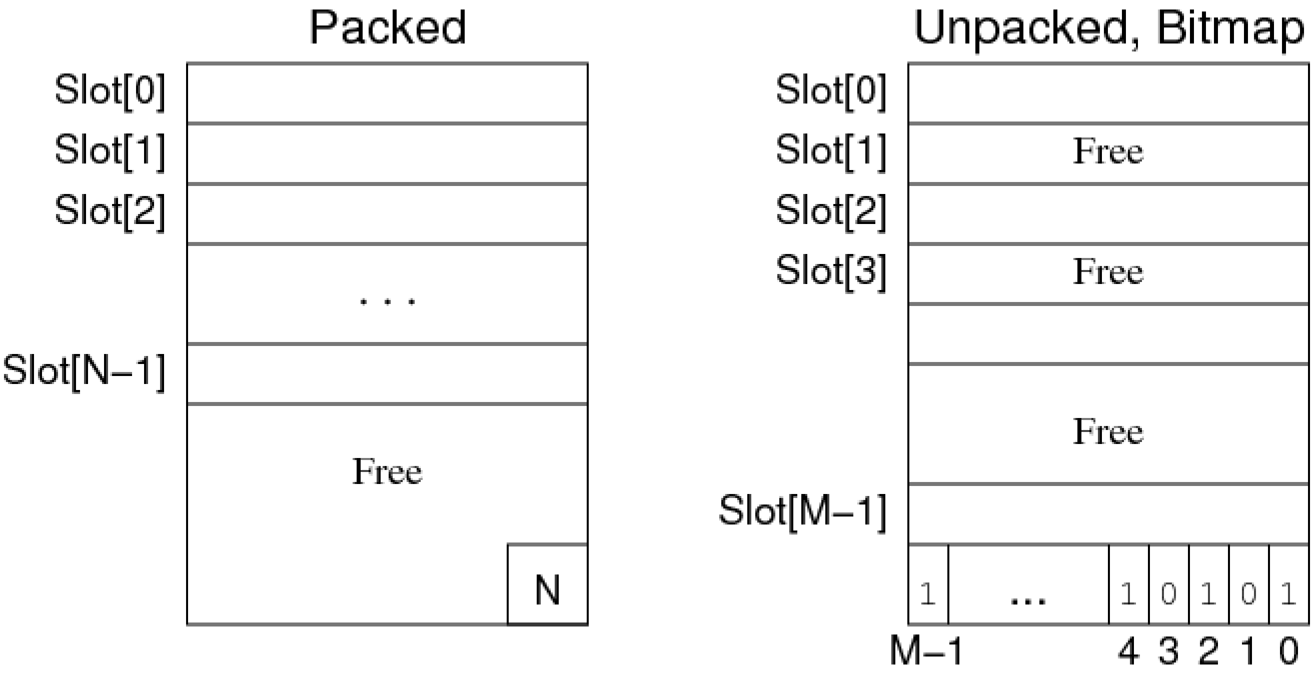
- record size variability (fixed, variable)
- how free space within **`Page`** is managed
- whether some data is stored outside **`Page`**
    - does **`Page`** have an associated overflow chain?
    - are large data values stored elsewhere? (e.g. TOAST)
    - can one tuple span multiple **`Page`**s?

Implementation of **`Page`** operations critically depends on format.

❖ Page Formats (cont)

For fixed-length records, use record slots.

- insert: place new record in first available slot
- delete: two possibilities for handling free record slots:



<< ^ >>

❖ Page Formats

For variable-length records, must use slot directory.

Possibilities for handling free-space within block:

- compacted (one region of free space)
- fragmented (distributed free space)

In practice, a combination is useful:

- normally fragmented (cheap to maintain)
- compacted when needed (e.g. record won't fit)

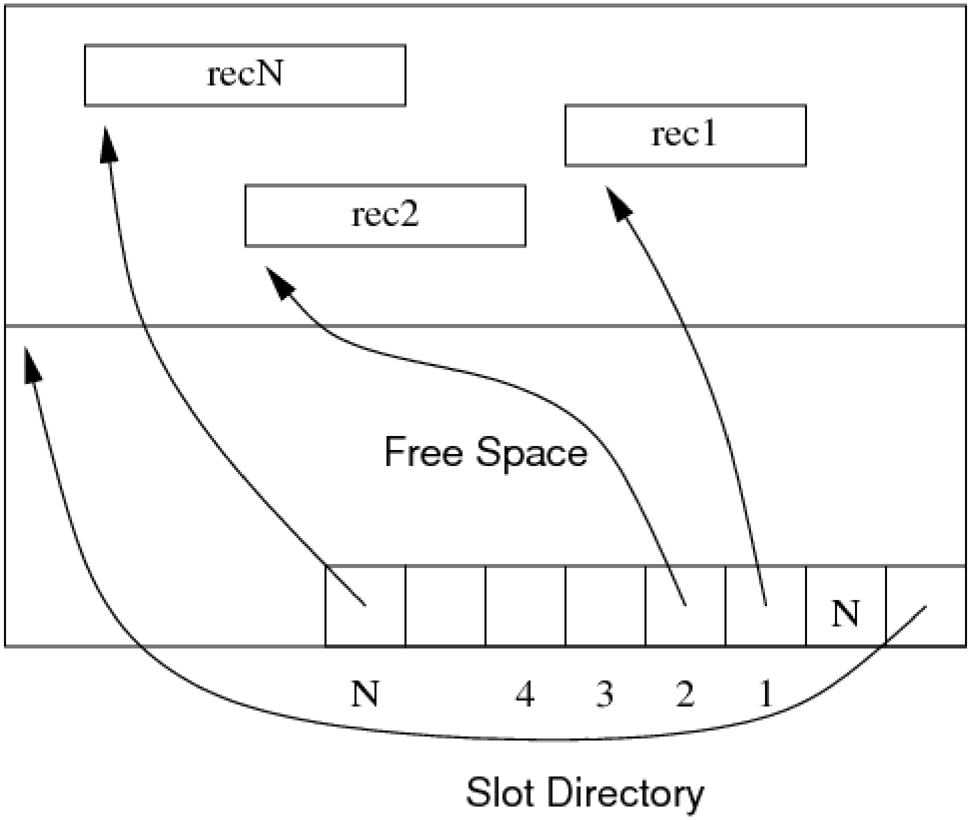
Important aspect of using slot directory

- location of tuple within page can change, tuple index does not change

<< ^ >>

❖ Page Formats (cont)

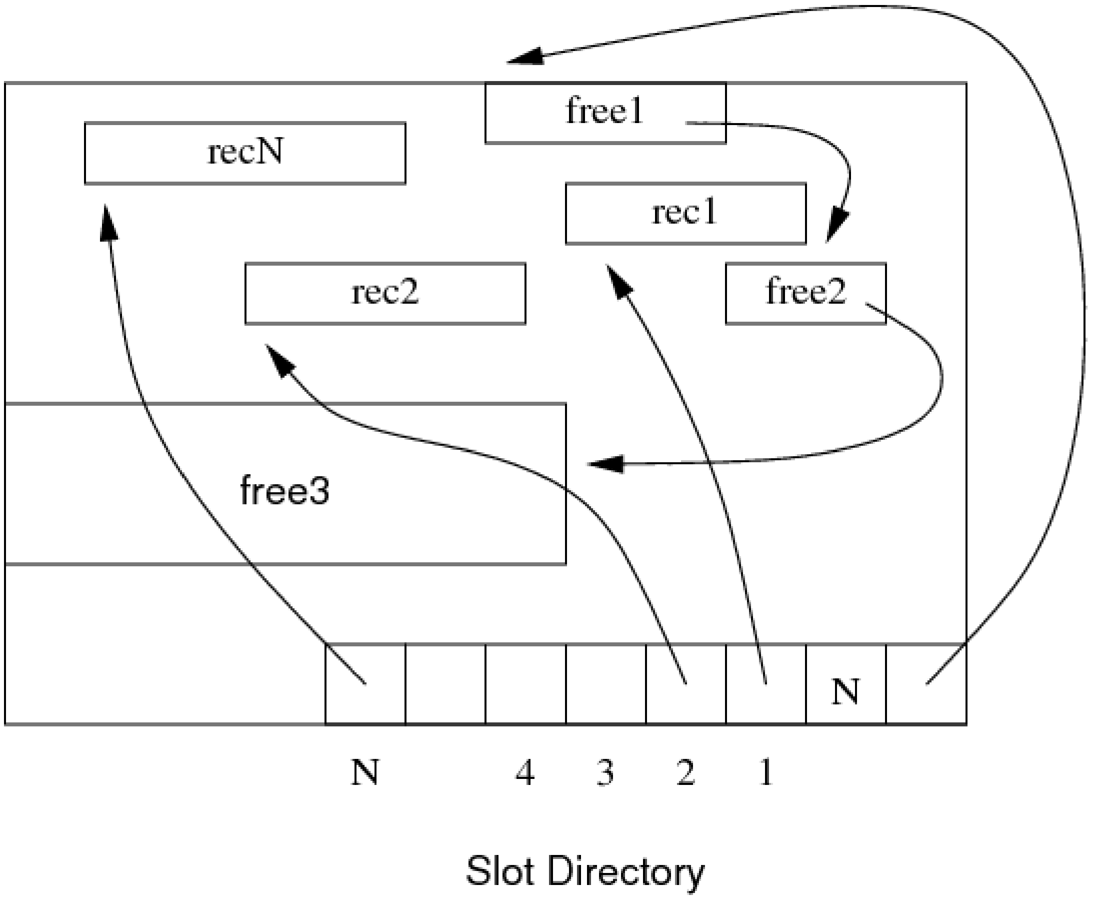
Compacted free space:



Note: "pointers" are implemented as word offsets within block.

❖ Page Formats (cont)

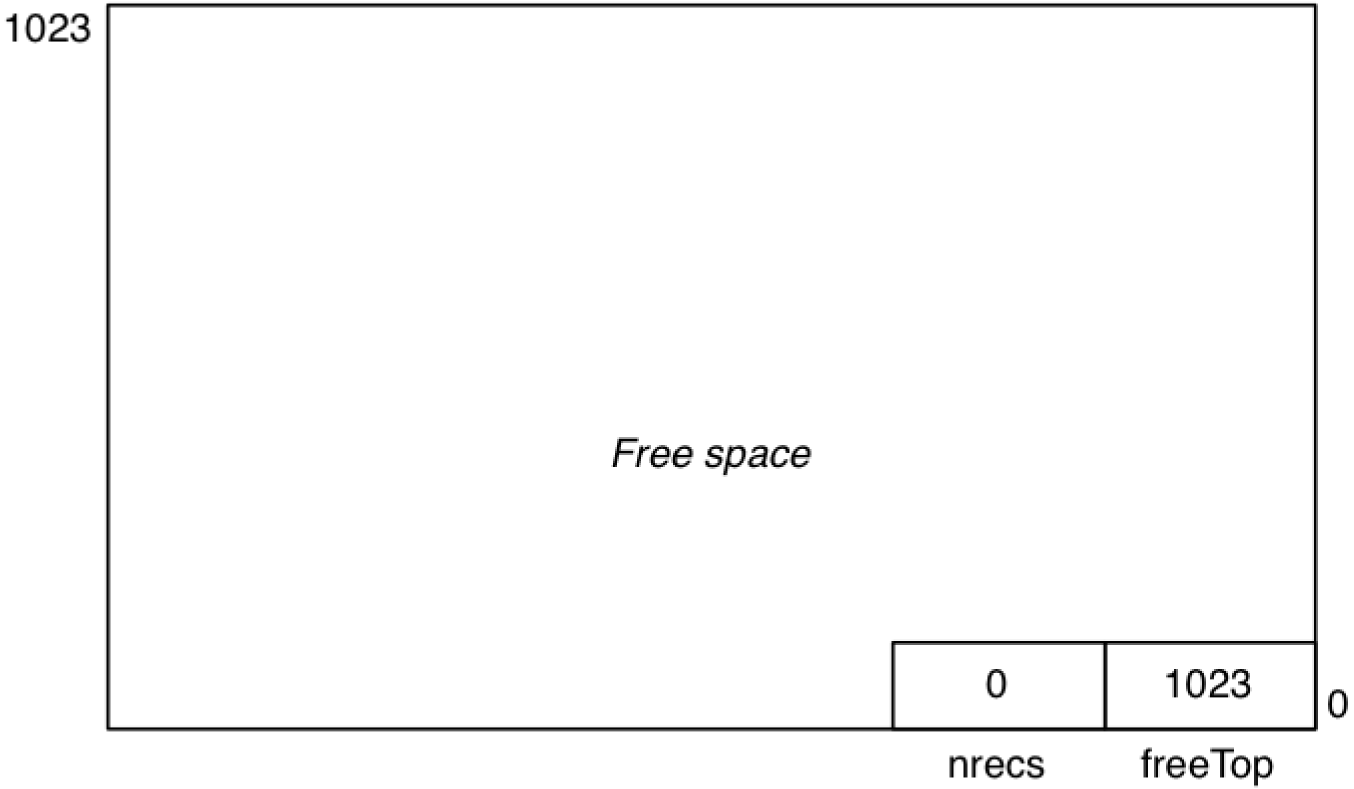
Fragmented free space:



<< ^ >>

❖ Page Formats (cont)

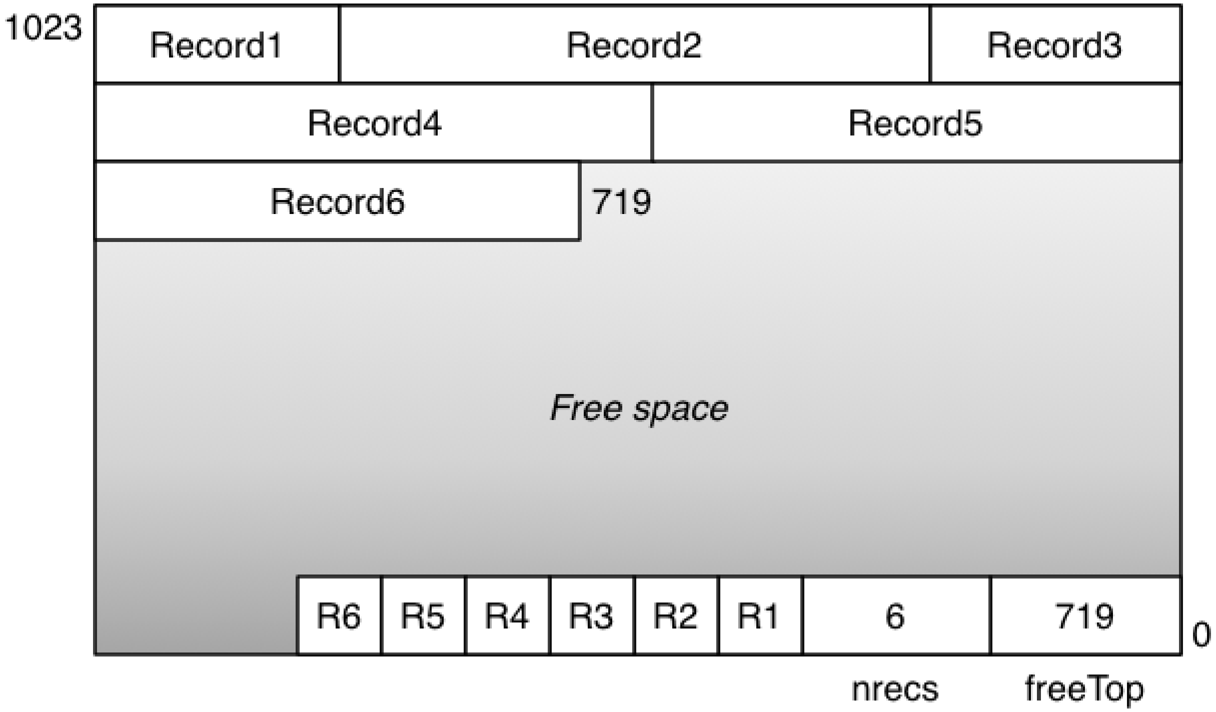
Initial page state (compacted free space) ...



<< ^ >>

❖ Page Formats (cont)

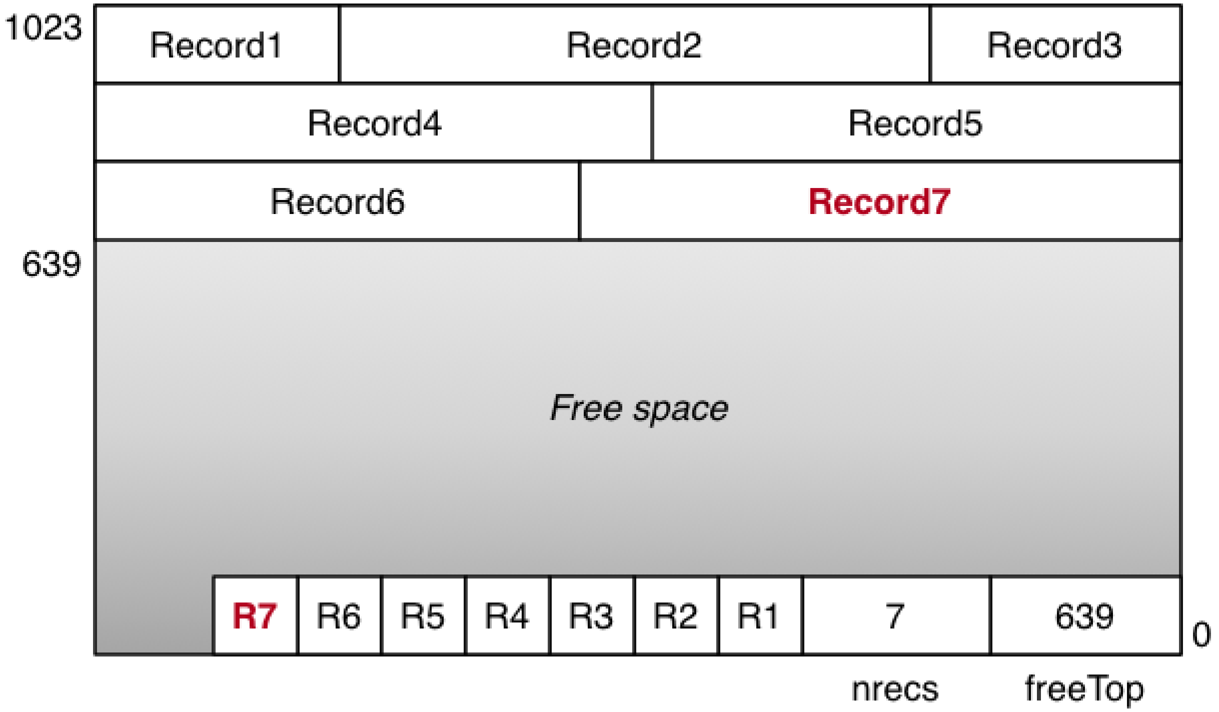
Before inserting record 7 (compacted free space) ...



<< ^ >>

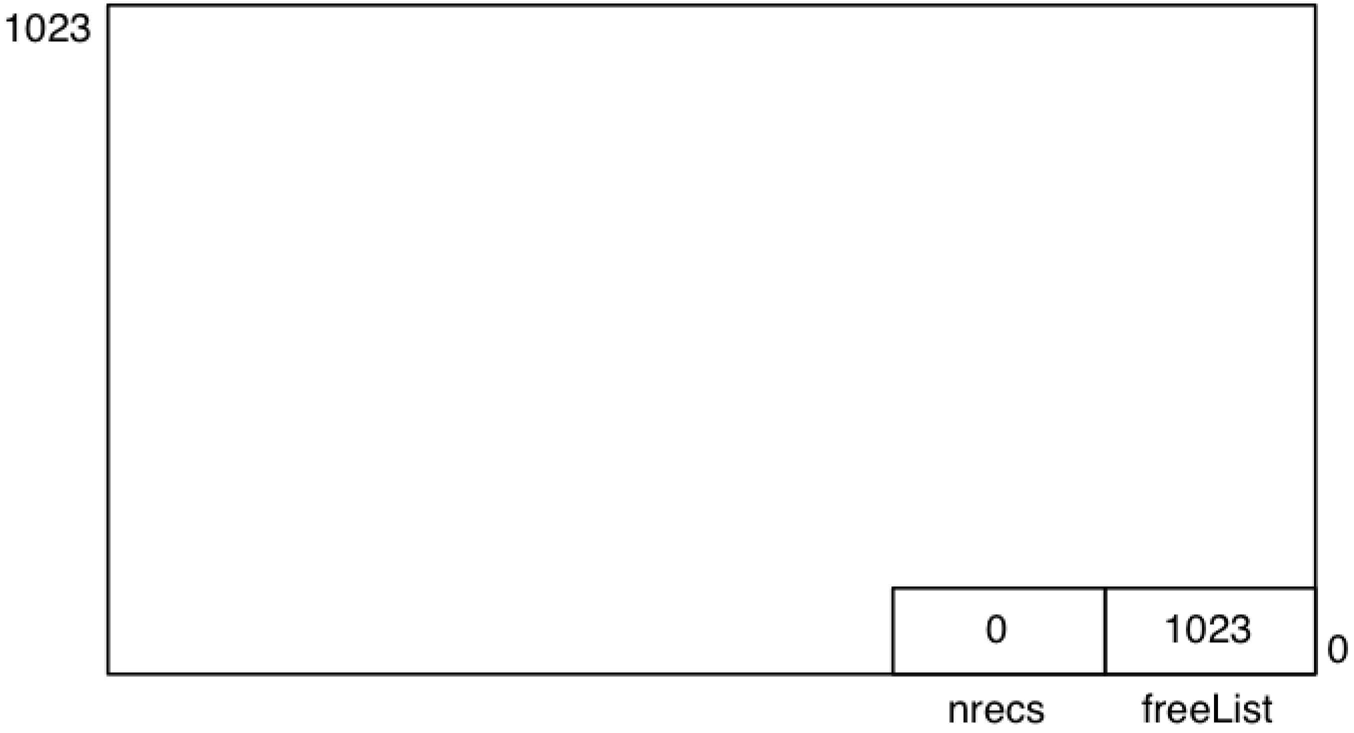
❖ Page Formats (cont)

After inserting record 7 (80 bytes) ...



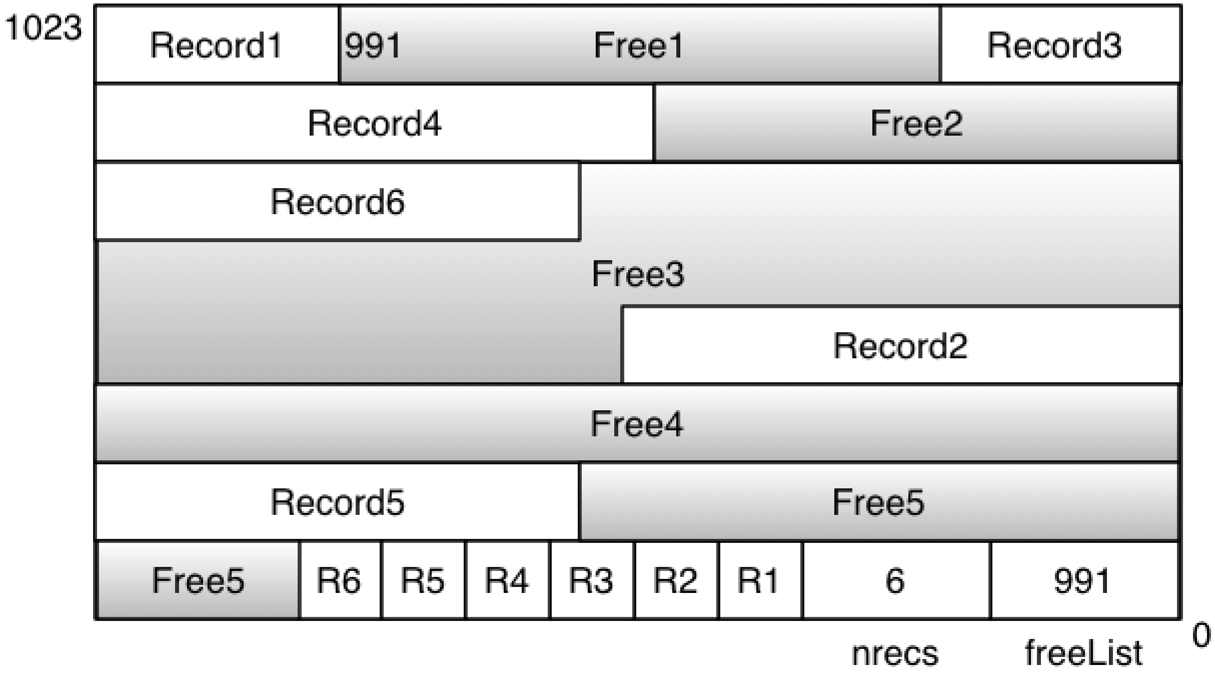
❖ Page Formats (cont)

Initial page state (fragmented free space) ...



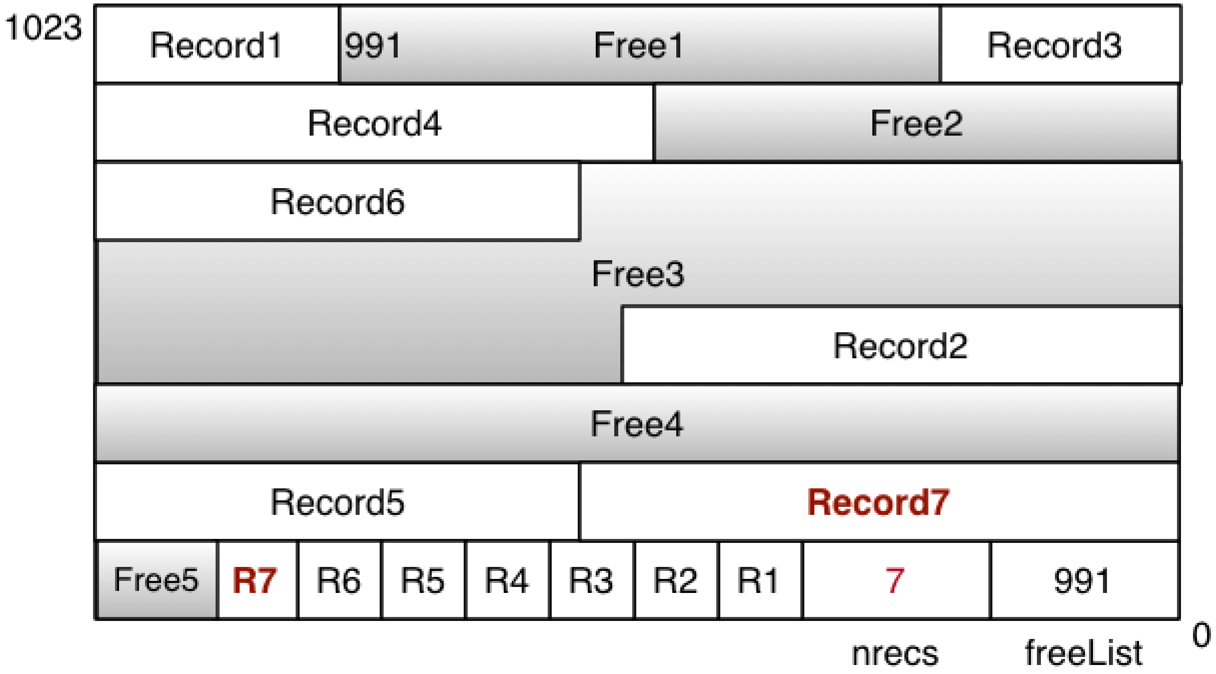
❖ Page Formats (cont)

Before inserting record 7 (fragmented free space) ...



❖ Page Formats (cont)

After inserting record 7 (80 bytes) ...



<< ∧ >>

❖ Storage Utilisation

How many records can fit in a page? (denoted C = capacity)

Depends on:

- page size ... typical values: 1KB, 2KB, 4KB, 8KB
- record size ... typical values: 64B, 200B, app-dependent
- page header data ... typically: 4B - 32B
- slot directory ... depends on how many records

We typically consider *average* record size (R)

Given C, $HeaderSize + C*SlotSize + C*R \leqslant PageSize$

<< ∧ >>

❖ Overflows

Sometimes, it may not be possible to insert a record into a page:

1. no free-space fragment large enough
2. overall free-space is not large enough
3. the record is larger than the page
4. no more free directory slots in page

For case (1), can first try to compact free-space within the page.

If still insufficient space, we need an alternative solution ...

<< ∧ >>

❖ Overflows (cont)

File organisation determines how cases (2)..(4) are handled.

If records may be inserted anywhere that there is free space

- cases (2) and (4) can be handled by making a new page
- case (3) requires either spanned records or "overflow file"

If file organisation determines record placement (e.g. hashed file)

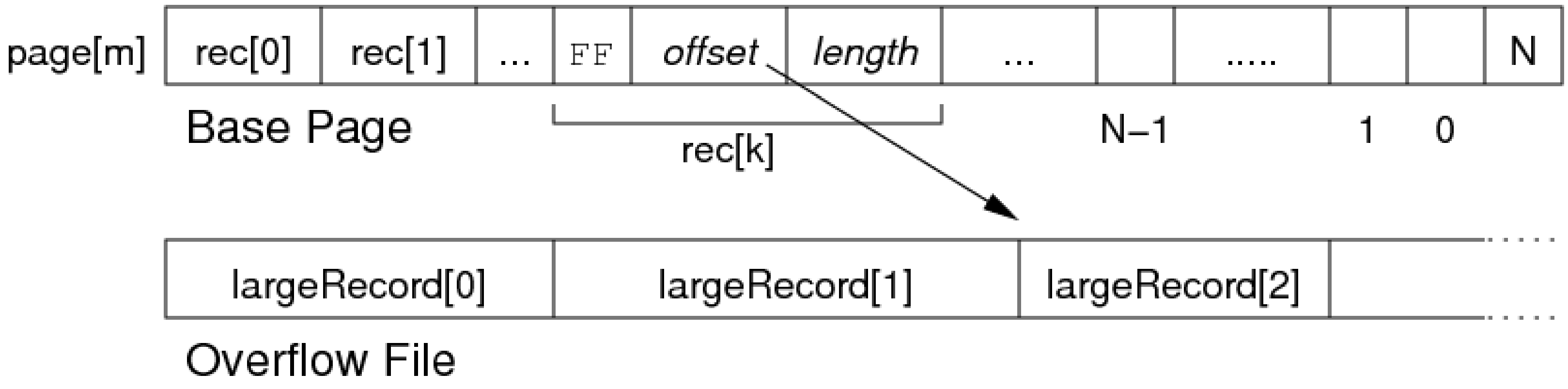
- cases (2) and (4) require an "overflow page"
- case (3) requires an "overflow file"

With overflow pages, *rid* structure may need modifying *(rel,page,ovfl,rec)*

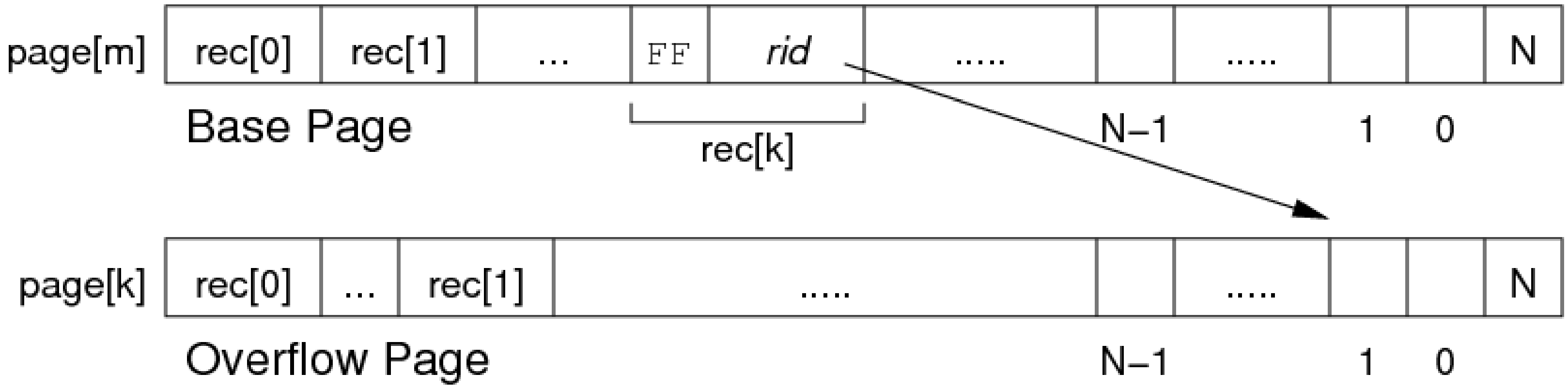
<< ∧

❖ Overflows (cont)

Overflow files for very large records and BLOBs:



Record-based handling of overflows:



We discuss overflow pages in more detail when covering Hash Files.

Produced: 23 Feb 2021