Buffer Manager Extra!

Further Reading (Papers I like)

Elizabeth J. O'Neil, Patrick E. O'Neil, Gerhard Weikum: The LRU-K Page Replacement Algorithm For Database Disk Buffering. SIGMOD Conference 1993: 297-306

Goetz Graefe: The five-minute rule 20 years later (and how flash memory changes the rules). Commun. ACM 52(7): 48-59 (2009)

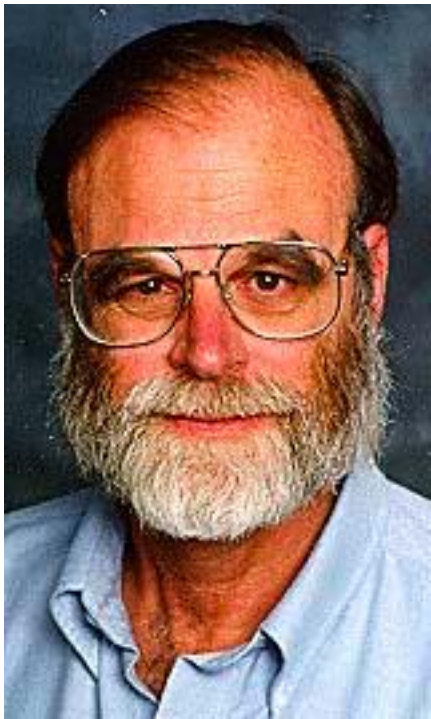
Jim Gray, Gianfranco R. Putzolu: The 5 Minute Rule for Trading Memory for Disk Accesses and The 10 Byte Rule for Trading Memory for CPU Time. SIGMOD Conference 1987: 395-398

Famous Extra One: 5 Minute Rule

The Five Minute Rule

Cache randomly accessed disk pages that are re-used every 5 minutes.
– Wikipedia formulation

Jim Gray



Rule has been updated
(and roughly held) a couple of times

Calculation: Cost of some RAM to hold the page versus fractional cost of the disk.

Calculation for Five Minute Rule (old)

- Disc and controller 15 access/s: 15k
  - Price per access/s is 1K$-2K$
- MB of main memory costs 5k$, so kB costs 5$

If making a 1Kb record resident saves 1 access/s, then we pay 5$ but save about $2000.

Break even point? 2000/5 ~ 400 seconds ≈ 5 minutes

5 Minute Rule abstractly

- RI: Expected interval in seconds between page refs
- M$ cost of main memory ($/byte)
- A$ cost of access per second ($/access/s)
- B size of the record/data to be stored

$$\frac{A\$}{RI} = M\$ * B$$

The Five Minute Rule Now

If you're curious about it:

Goetz Graefe: The five-minute rule 20 years later (and how flash memory changes the rules). Commun. ACM 52(7): 48-59 (2009)

LRU-k: Page Replacement with History

10k ft view of LRU-k

Main Idea: LRU drops pages w/o enough info. More Info (page accesses) better job

Ideal: If for each page p, we could estimate the next time it was going to be accessed, then we could build a better buffer manager.

LRU-k roughly keep around the last k times we accessed each page, and use this to *estimate interarrival times.*

Why might we want LRU-k

- No pool tuning required
- No separate pools required
- We can actually prove that it is optimal (in some sense) in many situations
- No Hints to the optimizer needed. Self-managing!

Notation for LRU-k

- Given a set of pages **P**=p1…pN
- A reference string **r**=r1…rT is a sequence where for each i=1…N ri = pj (j in 1…N).
  - r4=p2 means the fourth page we accessed is page 2
  - rT=p1 means the last page we accessed is page 1

Let k >= 1. Then "backward distance" B(**r**,p,k)
B(**r**,p,k) = x means that page p was accessed at time T-x and k-1 other times in r[T-x,T]
If page p not accessed k times in **r** then B(**r**,p,k) = ∞

LRU-k

- Ok, so we need a victim page, v.
  - If exist page has B(**r**,p,k) = ∞, pick p=v (ties using LRU)
  - O.w., choose v = argmax_{p in **P**} B(**r**,p,k)

LRU-1 is standard LRU

Challenge Early Page Eviction

- K >= 2. page p comes in and evicts a page. Next IO, which page is evicted?

Keep a page for a short amount of time after 1st reference.

Challenge: Correlated References

Correlated references: A single transaction access the same page ten times, each of these accesses is correlated with one another.

If we apply LRU-k directly, what problem may we have?

Compute interarrival guesses w/o correlated references. Pages that happen within a short window are not counted.

The need for History

- LRU-k must keep a history of accesses even for pages that are **not** in the buffer
- Why? Suppose we access a page, then are forced to toss it away and then immediately access it again.

- Upshot: May require a lot of memory.

Follow on work

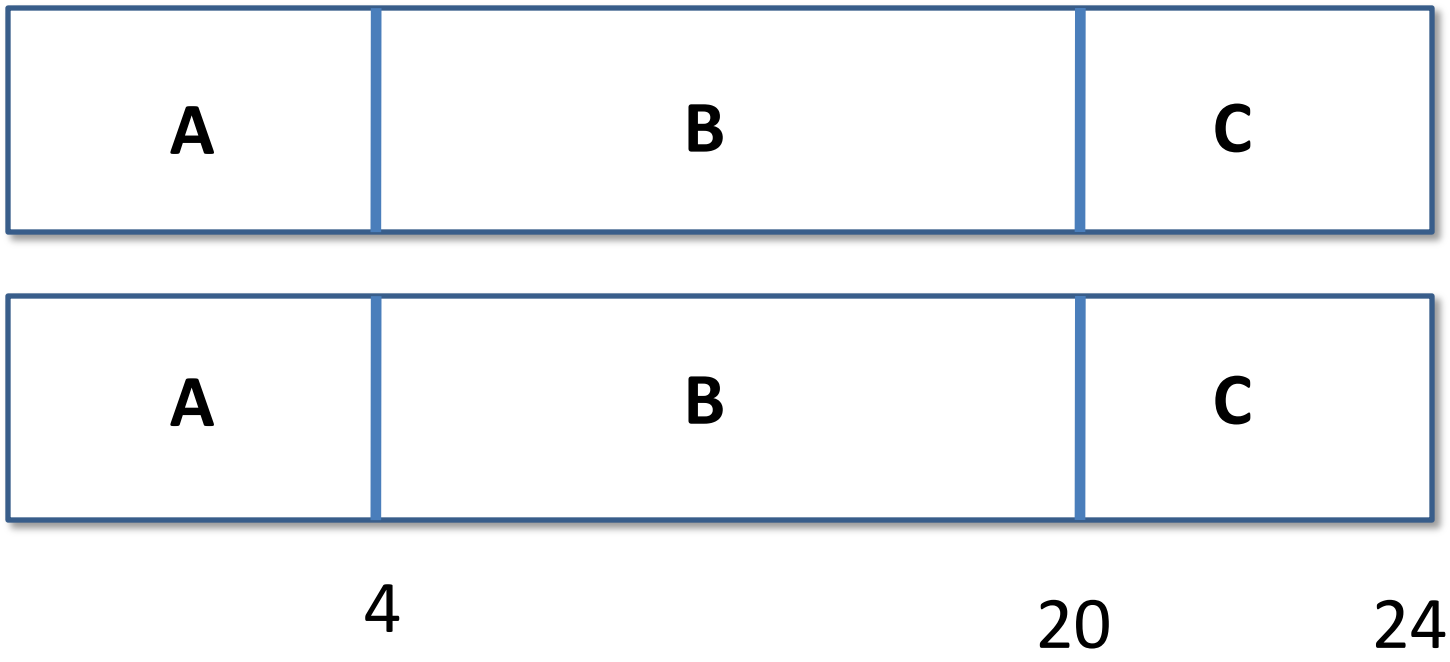
- There was a 1999 paper that extended the conditions for optimality (LRU-3).
  - Key idea is still that we can very accurately estimate interarrival time.
  - Assume that the accesses in the string are independent.
  - Optimal given the knowledge
    - even LRU-1 is optimal given its knowledge!

- We skipped the experiments, but they show that LRU-2 does well (and so does LFU)
  - See paper for details.

Page Formats

Record Formats: Fixed Length

Record has 3 attributes: A, B, C always fixed size (e.g., integers)



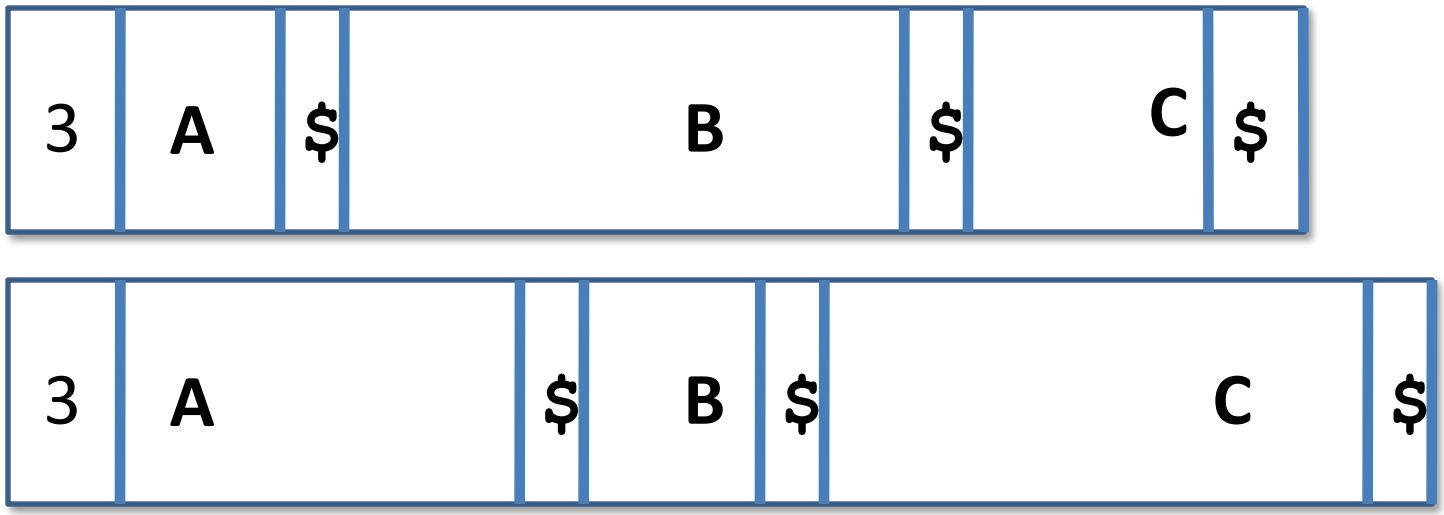
Given start address **START**

1. B at **START**+ 4
2. C at **START** + 20

These offsets stored once in the *Catalog*

Record Formats: Variable Length (I)

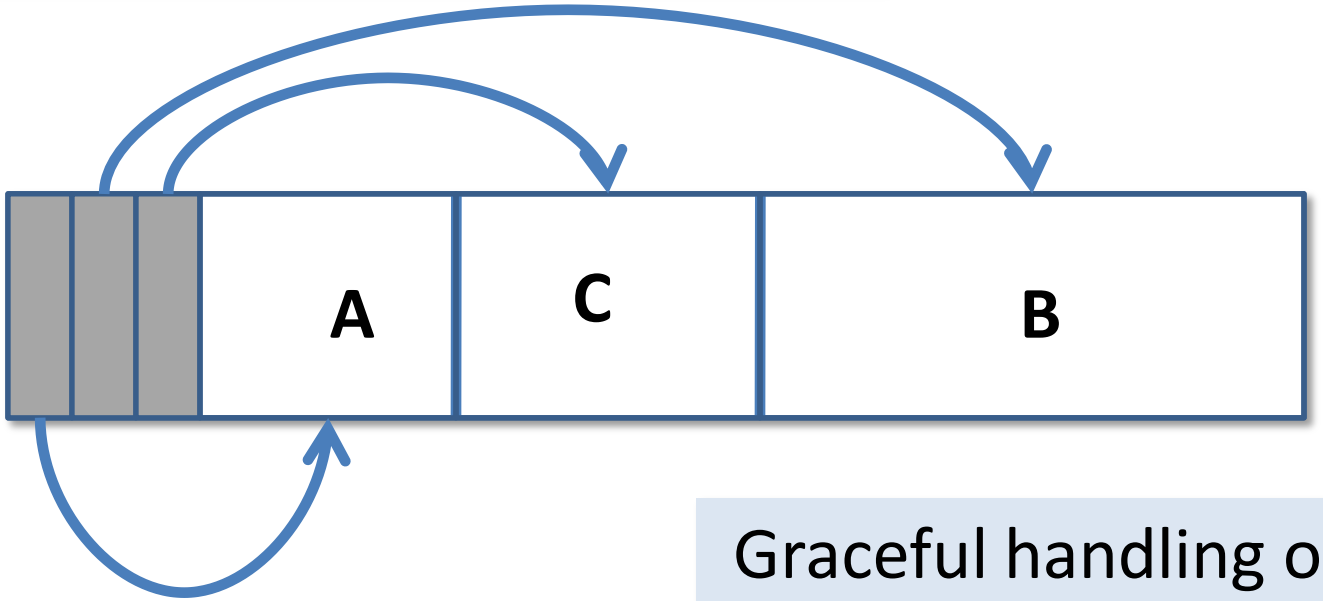
Record has three attributes: A, B, C



Fields delimited by a
special symbol (here $)

Record Formats: Variable Length (I)

Record has three attributes: A, B, C



Graceful handling of NULL. How?

Small array allows us to point to record locations

What if we grow a field? It could bump into another record!

Record Format Summary

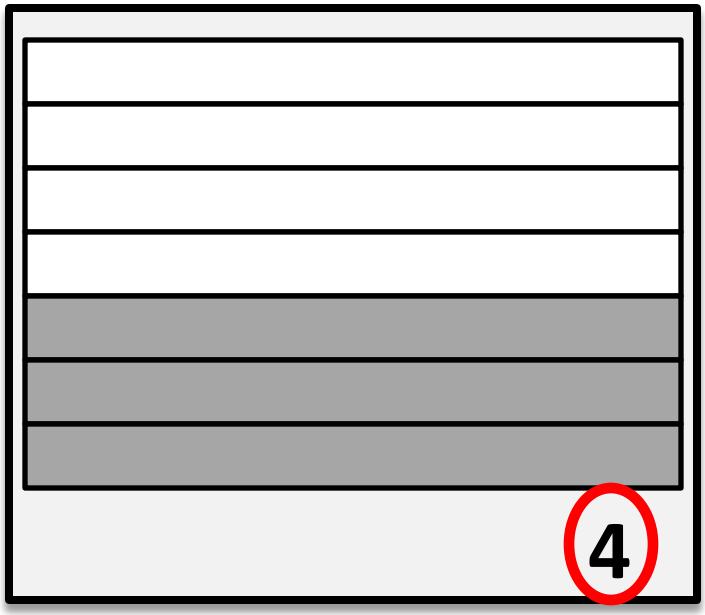
- Fixed length records excellent when
  - Each field, in each record is fixed size
  - Offsets stored in the catalog
- Variable-length records
  - Variable length fields
  - nullable fields some time too
  - Second approach typically dominates.

Page Formats

- A page is a “collection of slots” for records
- A *record id* is identified using the pair:
  - <Page ID, Slot Number>
  - Just need some Unique ID… but this is most common
- Fixed v. Variable Record influences page layout

Fixed, Packed Page

Slot 1
Slot 2



Free Space

Used Pages, then Free Pages

Page Header
contains # of records

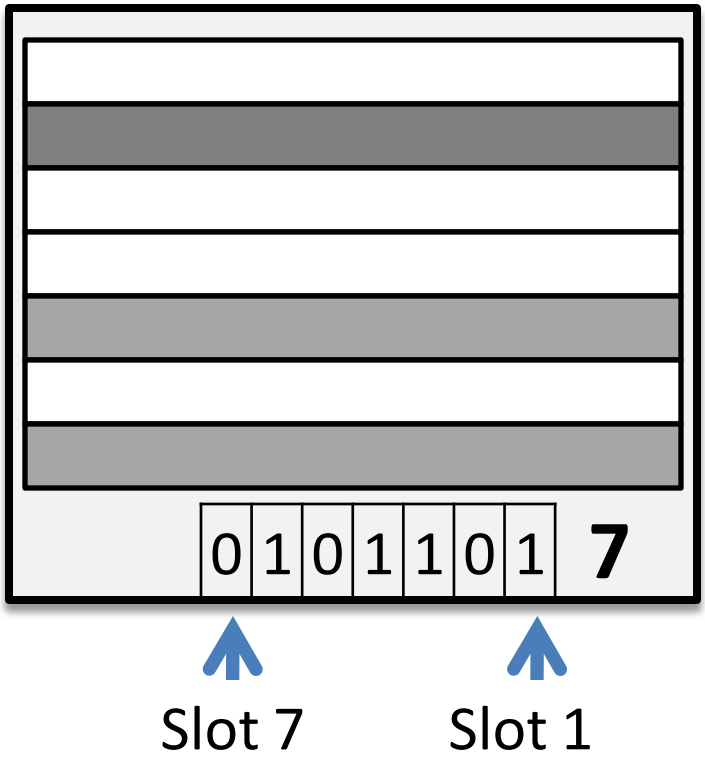
Rid is (PageID, Slot#). What happens if we delete a record?

This approach does not allow RIDs to be external!

Fixed, Unpacked, Bitmap

Slot 1
Slot 2

Free Space



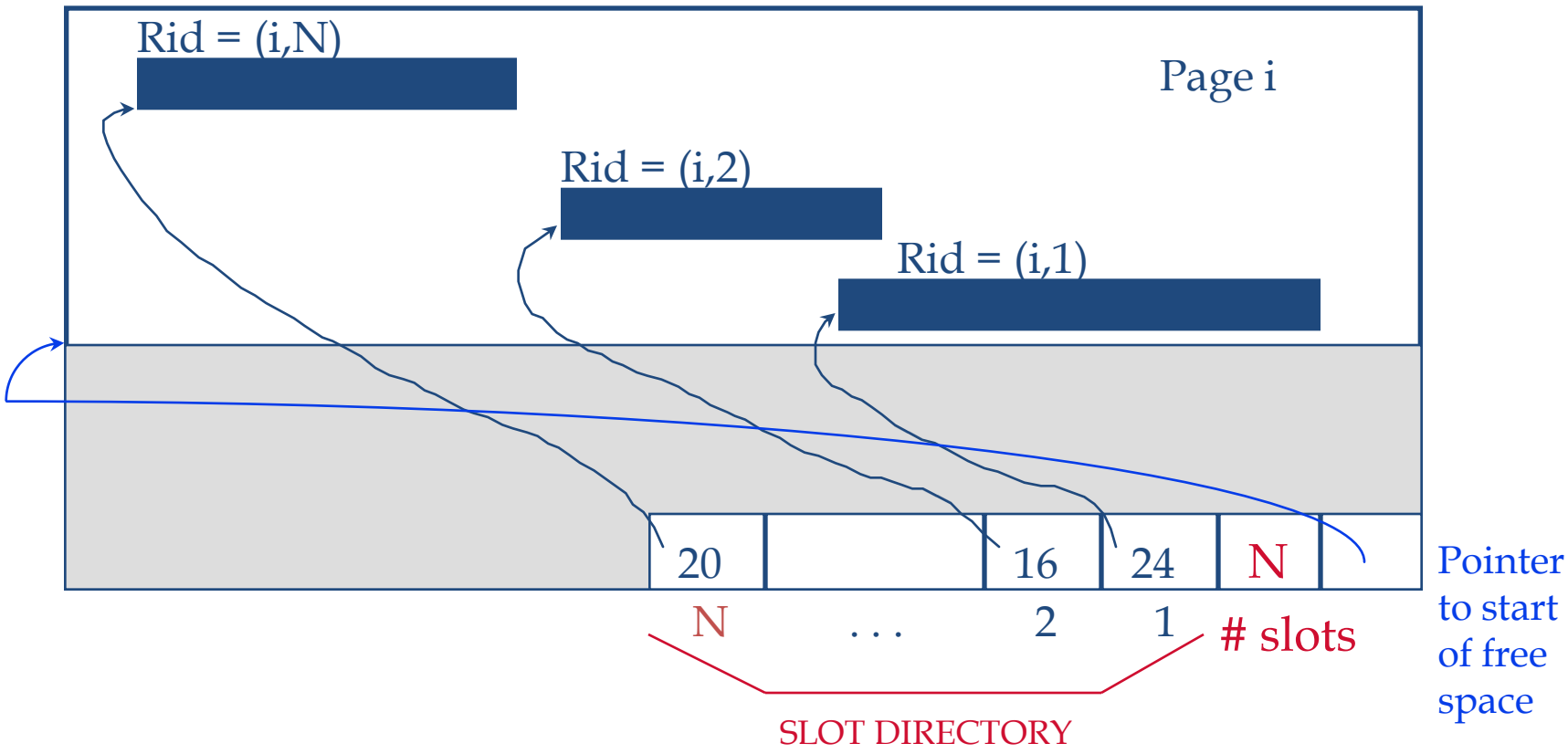
Free and Used comingle

Page Header
contains # of records
& bitmap for each slot

Rid is (PageID, Slot#). What happens if we delete a record?

Set the corresponding bit to 0!

Page Formats: Variable Length Records



* *Can move records on page without changing rid; so, attractive for fixed-length records too.*

Comparison of Formats

- Fixed-length is nice for static or append-only data
  - Log files, purchase logs, etc.
  - But requires fixed length
- Variable-length
  - Of course, variable-length records
  - Also, easier to use if we want to keep it sorted.
  - Why?

Row versus Column

Row-Column

Consider 100 byte pages.

- 10 fields each and each 10 bytes.
- Row store: 1 record per page.
- Col store: 10 records per page.

Pros? Higher throughput.

Local-compression (why?)

Cons? Update times.

Ok, but what about main memory? (Cachelines and DCU)

File of Records