Buffer Management Strategies

CS 346

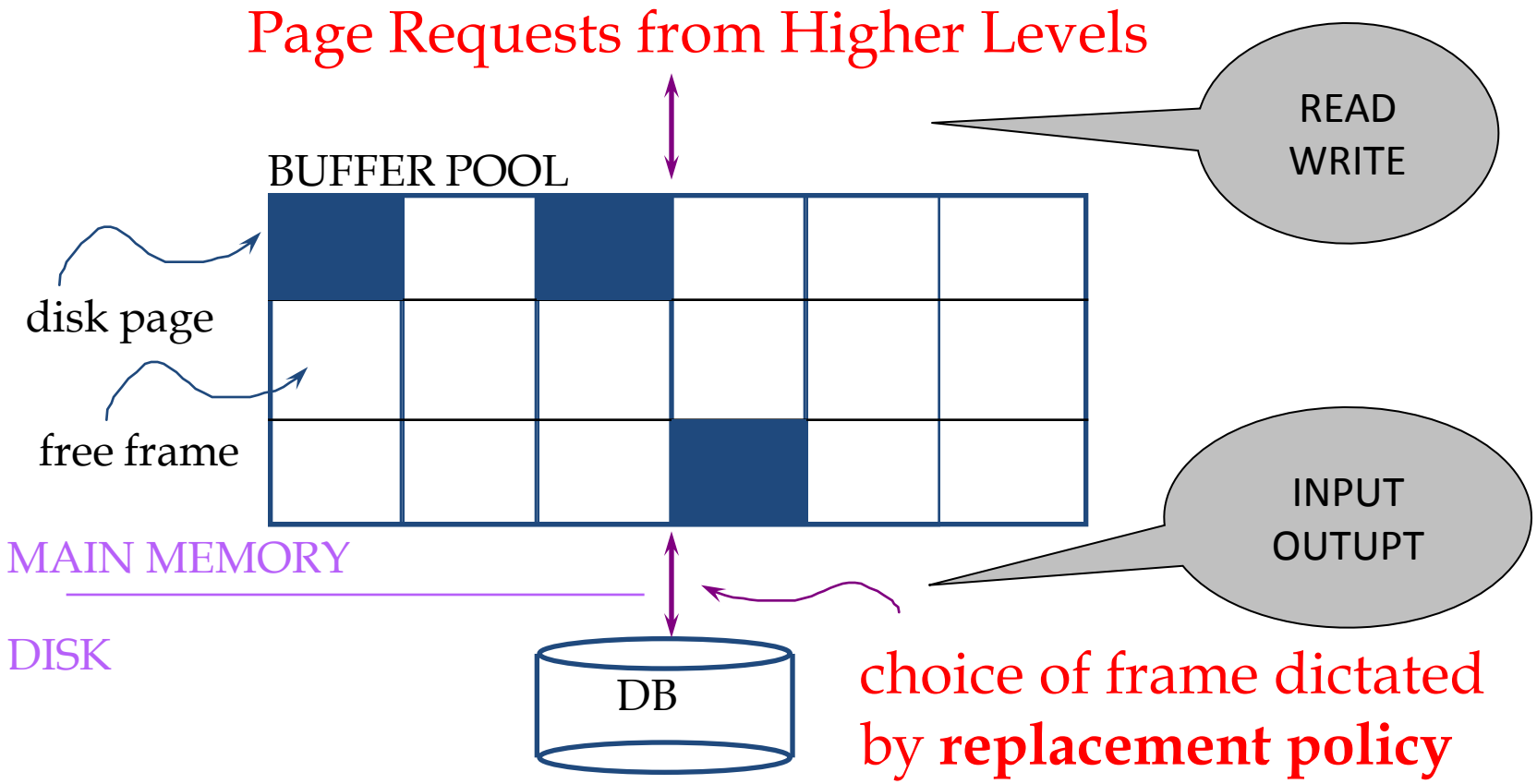
Outline

- CS346-level Buffer Manager Background
- Three Important Algorithms
- QLSM Model
- DBMin Algorithm
- Experiments

Buffer Managers

Buffer manager intelligently shuffles data from main memory to disk:
It is transparent to higher levels of DBMS operation

Buffer Management in a DBMS



- Data must be in RAM for DBMS to operate on it!
- Table of <frame#, pageid> pairs is maintained

When a page is requested…

- A *page* is the unit of memory we request
- If Page in the pool
  - Great no need to go to disk!

- If not? Choose a frame to replace.
  - If there is a free frame, use it!
    - **Terminology**: We pin a page (means it's in use)
  - If not? We need to choose a page to remove!
  - How DBMS makes choice is a **replacement policy**

Once we choose a page to remove

- A page is *dirty*, if its contents have been changed after writing
  - Buffer Manager keeps a dirty bit

- Say we choose to evict P
  - If P is *dirty,* we write it to disk
  - If P is not dirty, then what?

How do we pick a frame?

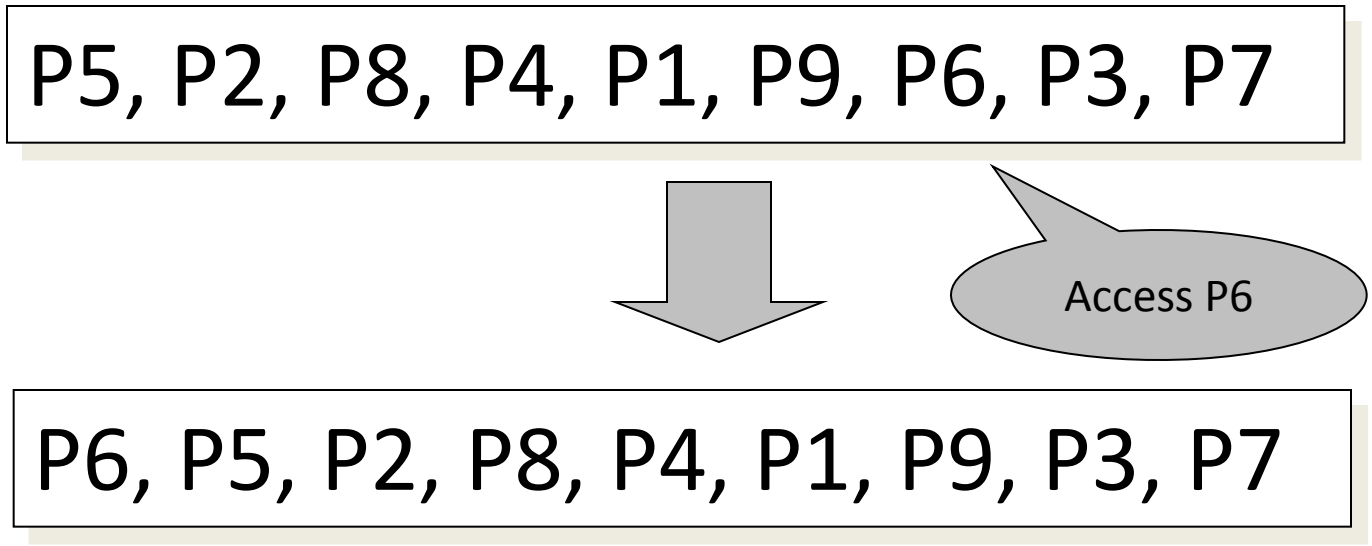
Needs to decide on page replacement policy

- Examples: LRU, Clock algorithm, MRU

Some work well in OS, but not always in DB… more later

Least Recently Used (LRU)

- Order pages by the time of last accessed
- Always replace the least recently accessed



LRU is expensive (why ?)

The Clock Approximation

- Instead we maintain a “last used clock”
  - Think of pages ordered 1…N around a clock
  - “The hand” sweeps around
  - Pages keep a “ref bit”
- Whenever a page is referenced, set the bit
- If current is has ref bit == false choose it
- If current is referenced, then unset ref bit and move on

“Approximates LRU” since referenced pages less likely

MRU

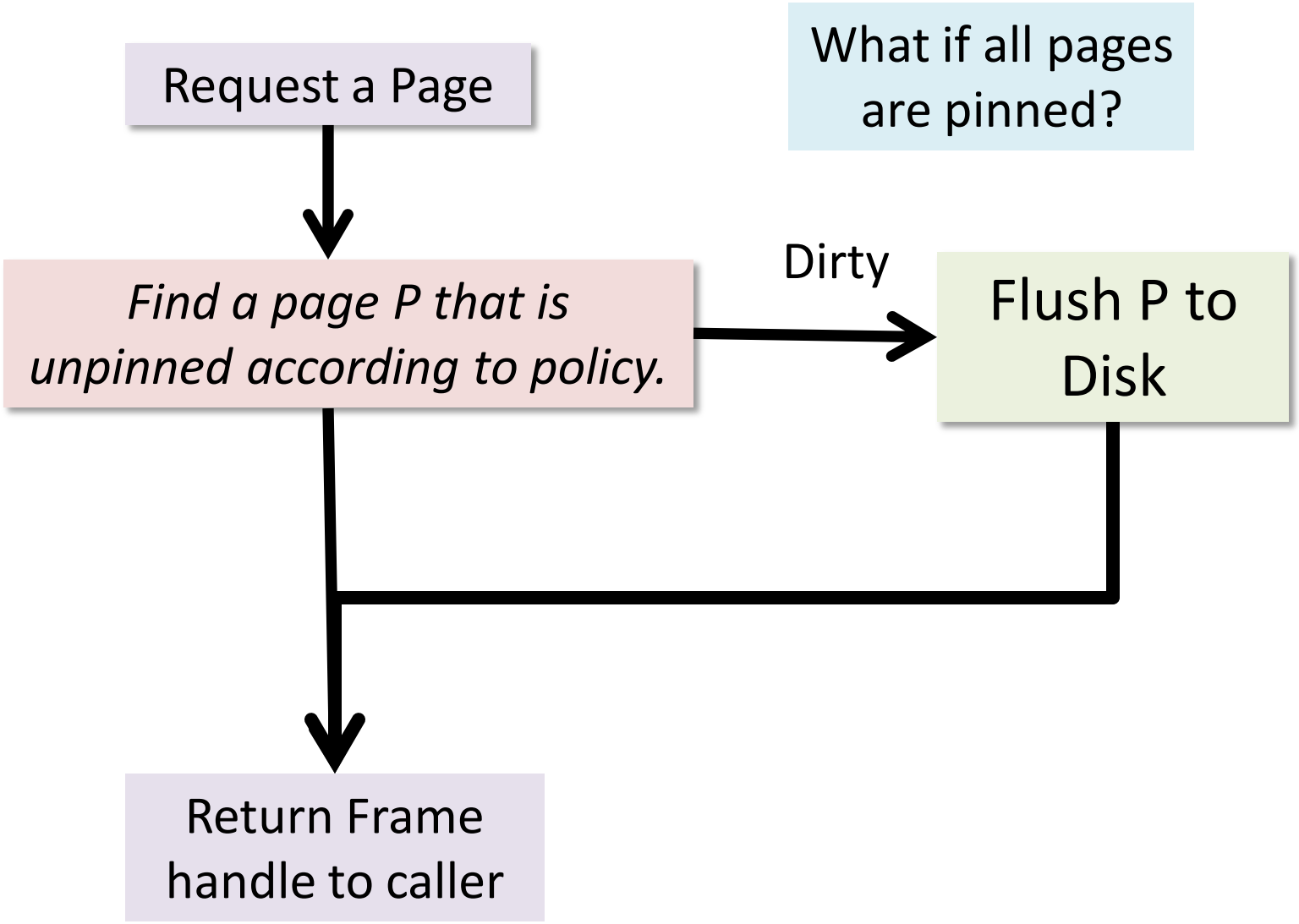
- **M**ost **R**ecently **U**sed.

- Why would you ever want to use this?

Hint: Consider scanning a relation that has 1 Mn pages, but we only have 1000 buffer pages…

This nasty situation is called *Sequential Flooding.* Each page request causes an I/O.

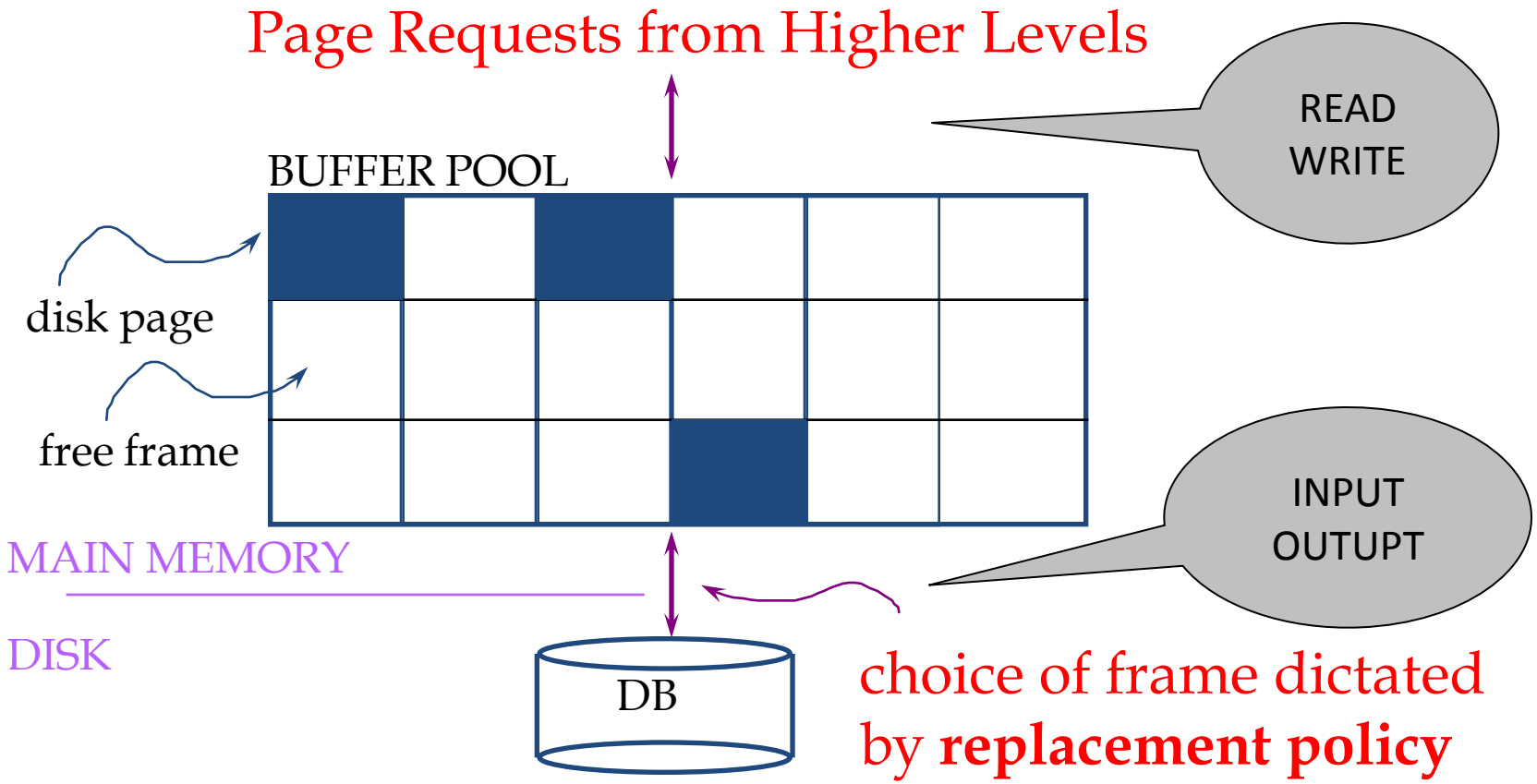
Simplified Buffer Manager Flowchart



Doesn't the OS manage Pages too?

- *Portability*: Different OS, Different support
  - Journaling, nothing, something crazy
- Limitations in OS: files cannot span disks.
- DBMS requires ability to *force pages* to disk
  - Recovery (much later)
- DBMS is better able to predict *page reference patterns*
  - Prefetching is harder

Buffer Management Summary



- Data must be in RAM for DBMS to operate on it!
- Table of <frame#, pageid> pairs is maintained

3 Important Algorithms (and ideas)

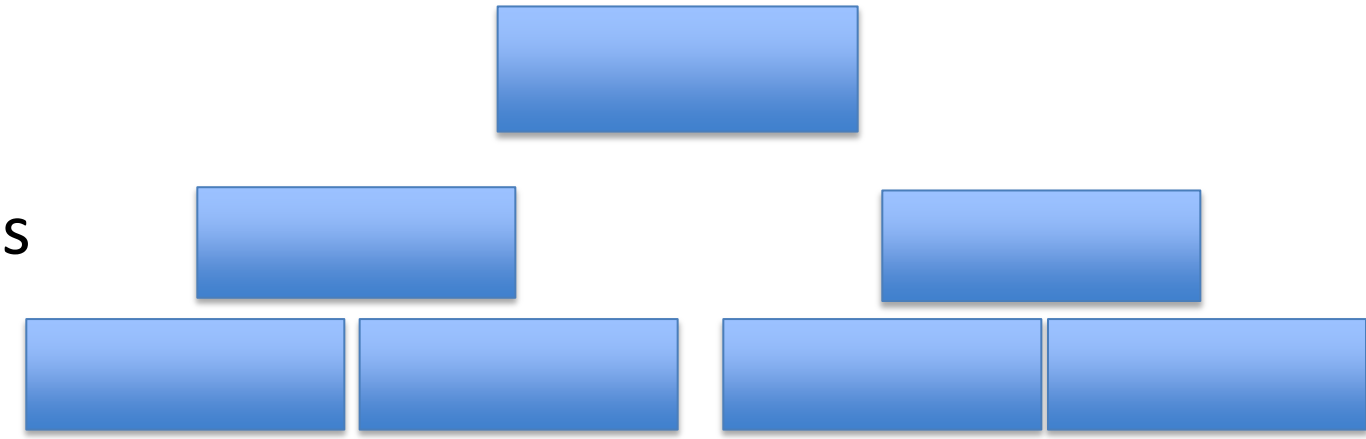
(I) Domain Separation (Reiter ’76)

- Separate pages into statically assigned domains
  - If page of type X is needed, then allocate in pool X
  - LRU in each Domain
  - If none are available in current domain, then borrow

1: Root

2: Internal Nodes

3: Leaves



Pros and Cons

- Pro: Big observation. Not all pages are the same!
- Con 1: Concept of domain is static
  - Replacement should depend on how page is used
  - E.g., a page of a relation in a scan v. a join
- Con 2: No priorities among domains
  - Index pages more frequently used than data pages
- Con 3: Multiuser issues.
  - No load control
  - Multiple users may compete for pages and *interfere*

(II) “New” Algorithm

Two Key Observations

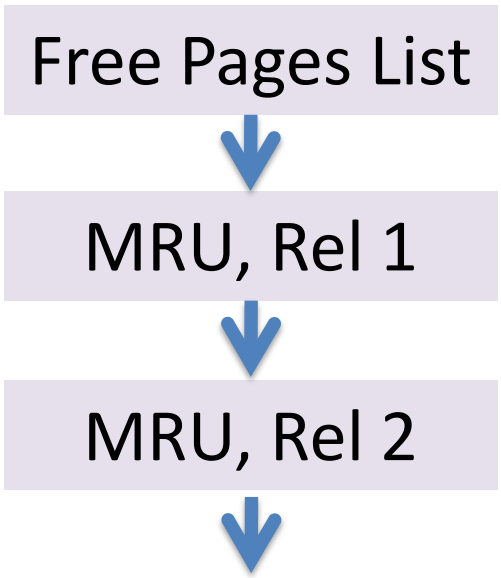
1. *“The priority of a page is not a property of the page; in contrast, it is a property of the relation to which that page belongs.”*
2. Each relation should have a **Working Set**

Separate Buffer pool by relation

Each relation is assigned:

1. Resident Set of Pages (MRU)
2. A small set exempt from replacement consideration

Resident Sets



Search through, top-down

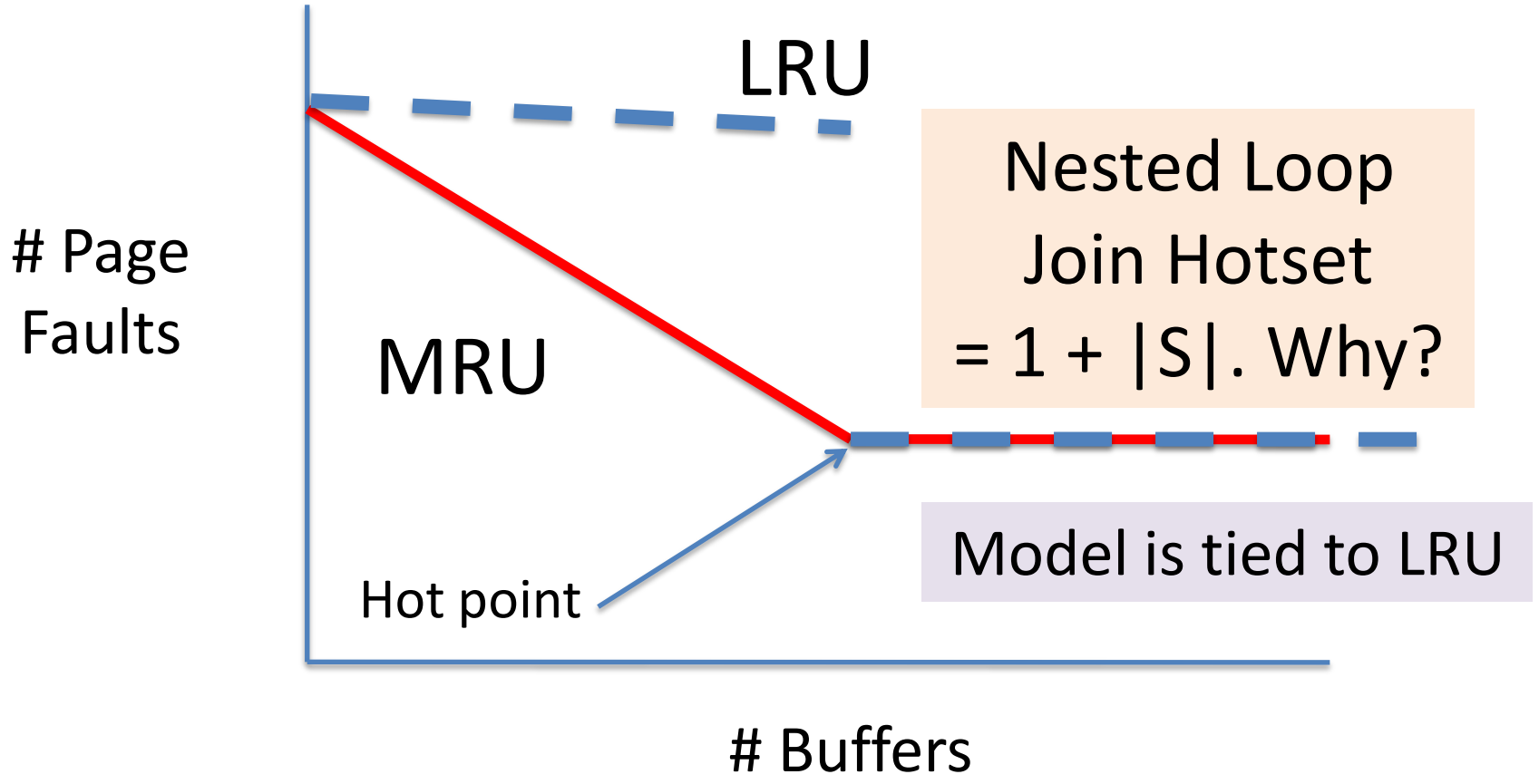
Intuition:
- If near the top, then unlikely to be reused.
- If near the bottom then pages are protected.

Pros and Cons of Resident Sets

- MRU only in limited cases. When?
- How do you order the resident sets?
  - Heuristic based (one could imagine some stats)
- Searching through list may be slow
- Multiuser? How do we extend this idea to work for multiple users?

(III) Hot Set Observation

A **hotset** is a set of pages that an operation will loop over many times.



Hotset Drawbacks

- The model is tied to LRU – but LRU is awful in some cases.
  - Where?
  - What can be cheaper than LRU and (sometimes) as effective?

Quiz

1. How does the buffer pool in a database differ from what you'd find in an OS?

2. When is MRU better than LRU?

3. Suppose you have a single buffer pool with N pages. Suppose that |R| = M pages and |S| = N+1 pages. How many IOs do you incur? (hint: what buffer policy do you use?)

Motivation for QLSM

1. Want to understand that pages should be treated differently. (from Reiter)
2. Want to understand that where a relation comes from matters
3. Want to understand that looping behavior (hot sets) makes a big difference
    1. And is predictable!

QLSM. Main Insights

- **Q**uery **L**ocality **S**et **M**odel

- Database access methods **are** predictable
  - handful of macro
  - So, define a handful of reference access methods

- We don’t need to tie it to LRU.

Example

- Consider an index nested loop join with an index on the joining attribute.
- Two locality sets:
  1. The index and inner relation
  2. The outer relation.

Handful of References. Sequential.

Sequential References: Scanning a relation.

1. Straight Sequential (SS). Only one access without repetition.
   - How many pages should we allocate?
2. Clustered Sequential (CS). Think Merge Join
   - What should we keep together?
3. Looping Sequential (CR). Nested loop Join.
   - What replacement policy on inner?

Random

Random References.

1. Independent Random. Example?

2. Clustered Random.

- Clustered inner index, outer is clustered
- Indexes are non-unique. (Similar to CS)

Hierarchical

- Straight Hierarchical: B+Tree Probe (X = 10)
- Hierarchical + Straight Sequential (X >= 10)
- Hierarchical + Cluster Sequential (X>= 10)
- Looping Hierarchical: Index-Nested Loop Join
  - Inner relation has the index.

Discussion

Do you believe this taxonomy is complete? To what extent is it complete?

Could you build a similar taxonomy of operations for Java programs?

DBMin

DBMin

- Buffers are managed on a **per file instance**
  - Active instances of the same file are given different buffer pools – and *may use different replacement policies!*
  - Files **share pages** via global table
- Set of buffered pages associated with a file instance is called its **locality set**
- Replacement Policy (RP) of a buffer simply moves to Global Free list (not actual eviction!)

How are concurrent queries supported?

Search Algorithm Cases

- Case I: page in both global table and locality set of requesting process
  - Simply update stats (of RP) and return
- Case II: Page in memory, but not in locality set
  - If page has an owner, then simply return it
  - O.w., page is allocated to requestor's LS
    - Could cause an "eviction" to free page list
- Case III: Page not in memory.
  - Bring it in, proceed as in case II.

Q: How could a page be in memory and not have an owner?

DBMin's Load Controller

- Activated when a file is opened or closed
  - Checks whether predicted locality set would fit in the buffer.
  - If not, suspend query.

- How does the LC know how big the locality set is?

Estimating Locality Set Size/Policy

Question: How big is the locality set? What is the policy?

- Straight Sequential (one-off scan)
- Clustered Sequential
- Independent Random
  - If sparse, go for either 1 or b. Policy?
  - If not, could upper bound using Yao's formula
- Looping Hierarchical Root traversed children more frequently
  - If cannot hold a entire level, access may look random
  - So, 3-4 pages may suffice (more now)

Algorithm Highlights/Summary

- Different pages *used* in different ways
  - A page meritocracy!
  - Classification and Taxonomy
- Allows us to use better replacement policy
  - Big wins with right policy
- Sharing of pages through global table
  - Local replacement policy just puts on global free
- Load control is built in

Questions to Reinforce the Material

- Working Set does not perform well on Joins.
  - Why?
- With a load-controller, every simple algorithm outperforms WS.
  - What does the load controller prevent?
- Is it reasonable to build such a load-controller?

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