Buffer Management

- Most popular replacement schemes:
  - LRU: Least recently used
  - MRU: Most recently used
  - Clock: A heuristic that approximates LRU
    - LRU is expensive.

States of a page

- pinned: pincount >0
- referenced: pincount=0; referenced=1
- available: pincount=0, referenced =0

Clock algorithm…

- pinned: pincount >0
- referenced: pincount=0; referenced=1
- available: pincount=0, referenced =0

1. loop until (found eligible page) or (all pages pinned)

Clock algorithm…

- pinned: pincount >0
- referenced: pincount=0; referenced=1
- available: pincount=0, referenced =0

```
1. loop until (found eligible page) or (all pages pinned)
    1. If page[current] = pinned
        1. current++
    2. If page[current] = referenced
        1. reference = 0; current++
```

Clock algorithm…

- pinned: pincount >0
- peferenced: pincount=0; referenced=1
- available: pincount=0, referenced =0

```
1. loop until (found eligible page) or (all pages pinned)
    1. If page[current] = pinned
        1. current++
    2. If page[current] = referenced
        1. reference = 0; current++
    3. If page [current] not referenced and not pinned
        1. replace page
        2. referenced=1;
        3. current++;
```

Designing BMSs that understand DBMSs

- Observation: root pages of indexes are accessed more often

Designing BMSs that understand DBMSs

- Observation: root pages of indexes are accessed more often
- Solution: Domain separation
  - Allocate separate buffers for different tasks
  - In case no buffer left, borrow from another task

Designing BMSs that understand DBMSs

- Observation: root pages of indexes are accessed more often
- Solution: Domain separation
  - Allocate separate buffers for different tasks
  - In case no buffer left, borrow from another task
- Problems:
  - Fails to capture query dynamicity
    - Some queries are more frequent than others
  - Does not differentiate between different pages
    - Only index-level and index vs. data page
    - (GLRU could solve this!)

Designing BMSs that understand DBMSs

- Observation: priority of a page is not a property of the page!
  - the relation that it belongs to
  - so, each relation needs a working_set

Designing BMSs that understand DBMSs

- Observation: priority is a page is not a property of the page!
  - the relation that it belongs to
  - so, each relation needs a working_set
- New algorithm:
  - Divide buffer pool and allocate per relation
    - Each relation is entitled to one active buffer (no replacement)
  - Each active relation is assigned a resident_set
  - Resident_sets of relations are maintained in a priority list
  - MRU is used for each relation
  - A relation is placed near the top if its pages are unlikely to be reused

Designing BMSs that understand DBMSs

- Observation: priority is a page is not a property of the page!
  - the relation that it belongs to
  - so, each relation needs a working_set
- New algorithm:
  - Divide buffer pool and allocate per relation
    - Each relation is entitled to one active buffer (no replacement)
  - Each active relation is assigned a resident_set
  - Resident_sets of relations are maintained in a priority list
  - MRU is used for each relation
  - A relation is placed near the top if its pages are unlikely to be reused

Tracks the locality of a query through relations…
…but does not work too well

Designing BMSs that understand DBMSs

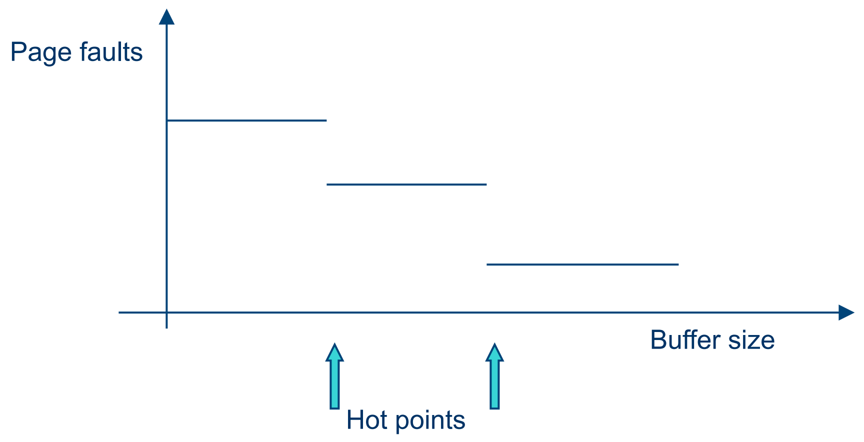
- Observation: Integrate advance knowledge of reference patterns

Designing BMSs that understand DBMSs

- Observation: Integrate advance knowledge of reference patterns
- HotSet: A set of pages with a looping behavior

Designing BMSs that understand DBMSs

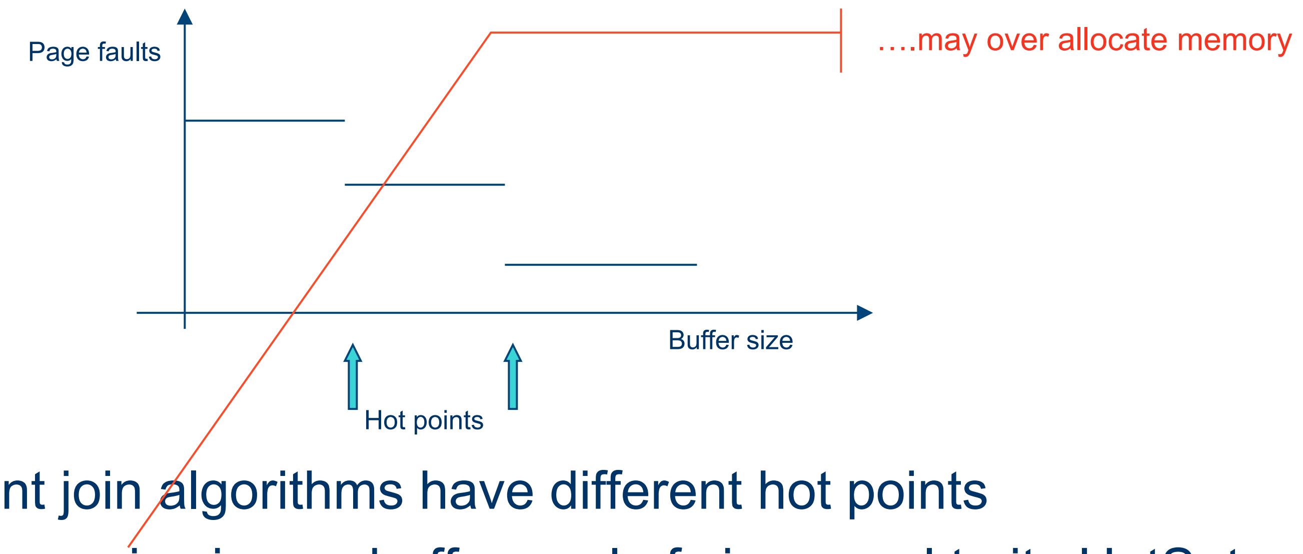
- Observation: Integrate advance knowledge of reference patterns
- HotSet: A set of pages with a looping behavior



- Different join algorithms have different hot points
- Each query is given a buffer pool of size equal to its HotSet

Designing BMSs that understand DBMSs

- Observation: Integrate advance knowledge of reference patterns
- HotSet: A set of pages with a looping behavior



- Different join algorithms have different hot points
- Each query is given a buffer pool of size equal to its HotSet

Designing BMSs that understand DBMSs

- Query Locality Set Model:
  - similar to HotSet
  - separates
    - the modeling of reference behavior

    from
    - any particular buffer management algorithm

Designing BMSs that understand DBMSs

- Query Locality Set Model: similar to HotSet
  - Identify all operations in a query plan
  - Identify locality sets for each operation separately
    - straight sequential (ss)
    - clustered sequential (cs)
    - looping sequential (ls)
    - independent random (ir)
    - clustered random (cr)
    - straight hierarchical (sh)
    - sh+ss
    - sh+cs
    - looping hierarchical (lh)

Access patterns

- Straight sequential(SS) - only need 1 page frame because once you’ve used a page won’t go back to it.
- Clustered Sequential(CS) - size of the cluster needed, have sequential access but using more than 1 page at a time. So need all the pages accessed simultaneously.
- Looping Sequential (LS) - MRU
- Independent Random (IR) - Random access, no pattern, behavior is similar to straight sequential because once accessed it most likely won’t be used again soon. Use LRU
- Clustered Random (CR) - similar to cluster sequential but have to estimate the size of the cluster.
- Straight Hierarchical (SH) - Start at the top of index structure and go down because at every level only use 1 page.

Access patterns

- SH followed by SS - B+ tree is sorted and sequential, so go down tree then follow across the pages so similar to SS
- SH,CS - basically clustered pages.
- Looping Hierarchical - start from the top, go to a data element, go back to top look for next one, go back to top, etc. So upper level pages are accessed more often.
  - Prob(p in i^{th} level) = $1/f^i$
  - 2nd level accessed 2x as much as 1st level, 3rd level 2x as much as 2nd, etc.

Designing BMSs that understand DBMSs

- Query Locality Set Model: similar to HotSet
  - Identify all operations in a query plan
  - Identify locality sets for each operation separately

straight sequential (ss)	one page frame
clustered sequential (cs)	size of cluster
looping sequential (ls)	MRU
independent random (ir)	ss
clustered random (cr)	cs
straight hierarchical (sh)	one page frame
sh+ss	ss
sh+cs	cs
looping hierarchical (lh)	prob(p in ith level) = $1/f^i$

Designing BMSs that understand DBMSs

- Query Locality Set Model: similar to HotSet
  - Identify all operations in a query plan
  - Identify locality sets for each operation separately

straight sequential (ss)	one page frame
clustered sequential (cs)	size of cluster
looping sequential (ls)	MRU
independent random (ir)	ss
clustered random (cr)	cs
straight hierarchical (sh)	one page frame
sh+ss	ss
sh+cs	cs
looping hierarchical (lh)	prob(p in ith level) = $1/f^i$

- DBMin: Assign a locality set per file instance

Designing BMSs that understand DBMSs

- Query Locality Set Model: similar to HotSet
  - Identify all operations in a query plan
  - Identify locality sets for each operation separately

straight sequential (ss)	one page frame	
clustered sequential (cs)	size of cluster	
looping sequential (ls)	MRU	
independent random (ir)	ss	Working set + "New"
clustered random (cr)	cs	
straight hierarchical (sh)	one page frame	
sh+ss	ss	
sh+cs	cs	
looping hierarchical (lh)	prob(p in ith level) = $1/f^i$	

- DBMin: Assign a locality set per file instance *(file,queryid)*

Designing BMSs that understand DBMSs

- DBMin: Assign a locality set per file instance *(file,queryid)*
  - Each locality set is managed separately
  - Each page belongs to one locality set (query)
    - Page belongs to only 1 file instance even if 2 queries are looking at it.
    - If a page does not belong to any locality set, it is set free
  - A global table is used to share data across queries

Designing BMSs that understand DBMSs

- DBMin: Assign a locality set per file instance *(file,queryid)*
  - Each locality set is managed separately
  - Each page belongs to one locality set (query)
    - If a page does not belong to any locality set, it is set free
  - A global table is used to share data across queries
- Algorithm:
  1. Page found in global table and in the requested locality set, l
     1. Use local replacement policy of l

You own it – you manage it

Designing BMSs that understand DBMSs

- DBMin: Assign a locality set per file instance *(file,queryid)*
  - Each locality set is managed separately
  - Each page belongs to one locality set (query)
    - If a page does not belong to any locality set, it is set free
  - A global table is used to share data across queries
- Algorithm:
  1. Page found in global table and in the requested locality set, l
     1. Use local replacement policy of l
  2. Page found in the global table, but not in the locality set
     1. Page has an owner: give the page to the new request
     2. Page has no owner: put the page in locality set, l (may need replacement)

Somebody else owns it – ownership changes

Nobody owns it – take it

Designing BMSs that understand DBMSs

- DBMin: Assign a locality set per file instance *(file,queryid)*
  - Each locality set is managed separately
  - Each page belongs to one locality set (query)
    - If a page does not belong to any locality set, it is set free
  - A global table is used to share data across queries
- Algorithm:
  1. Page found in global table and in the requested locality set, l
     1. Use local replacement policy of l
  2. Page found in the global table, but not in the locality set
     1. Page has an owner: give the page to the new request
     2. Page has no owner: put the page in locality set, l (may need replacement)
  3. Page not found
     1. Bring it to the main memory and proceed as 2.

Not in the buffer – read it from disk

Designing BMSs that understand DBMSs

- DBMin: Assign a locality set per file instance *(file,queryid)*
  - Each locality set is managed separately
  - Each page belongs to one locality set (query)
    - If a page does not belong to any locality set, it is set free
  - A global table is used to share data across queries
- Advantage:
  - When replaced, pages don't necessarily swap to disk. Mark it as free so it's still in memory until kicked out by someone else. So can get it back without having to read the disk. High chance will find it in main memory.
  - Can leverage diverse knowledge about tables, access patterns, and queries. Only a hand full of access patterns are enough.