Evaluation of Pointer Swizzling Techniques for DBMS Buffer Management

Max Gilbert

University of Kaiserslautern

m_gilbert13@cs.uni-kl.de

June 8, 2017

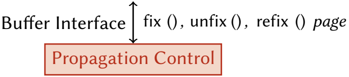
Table of Contents

Section 1

Pointer Swizzling as in "In-Memory Performance for Big Data"

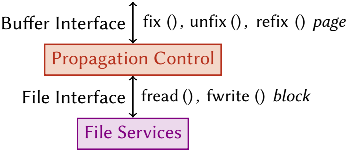
Five Layer Architecture of a DBMS ([HR83] [HR85])

Five Layer Architecture of a DBMS ([HR83] [HR85])



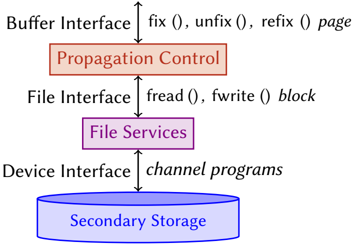
- Mainly consists of the *DBMS buffer bool*
- Pages can be fixed and unfixed by upper layers
- Offers copies of persistent data in main memory

Five Layer Architecture of a DBMS ([HR83] [HR85])



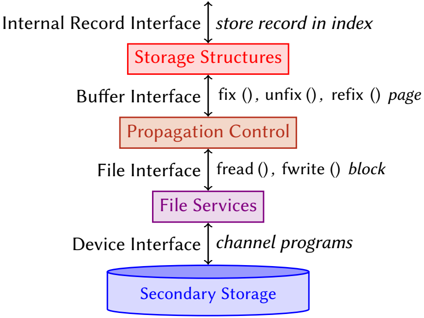
- Also called *file management*
- Offers dynamically growing files and blocks of different length
- Manages file addresses, block borders and unused space
- Abstracts from the different characteristics of *secondary storage devices*

Five Layer Architecture of a DBMS ([HR83] [HR85])



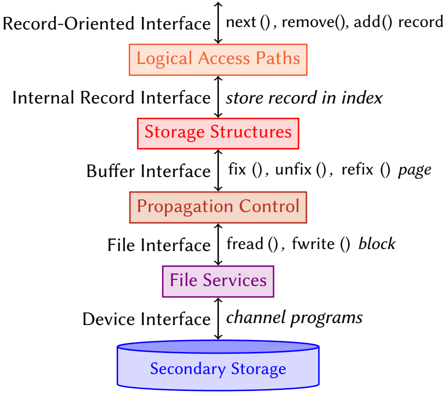
- HDDs, SSDs etc.
- Interface is partially device specific

Five Layer Architecture of a DBMS ([HR83] [HR85])



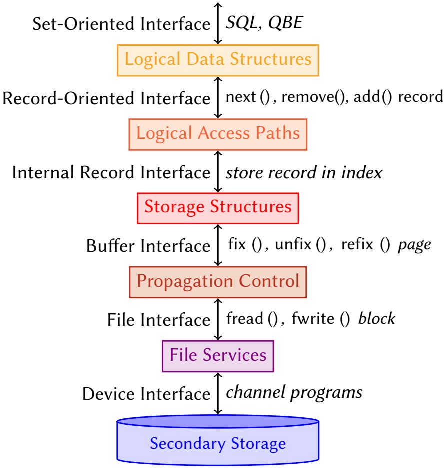
- Also called *record and access path management*
- B* trees, hash indexes etc.
- Allows to navigate through those storage structures
- Maps records to pages
- Commonly offers variable-sized fields, large fields and references between records

Five Layer Architecture of a DBMS ([HR83] [HR85])



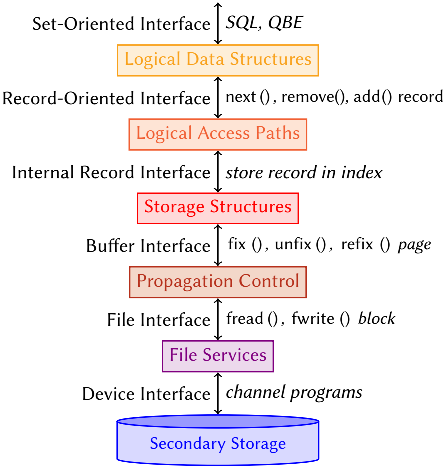
- Also called *navigational access layer*
- Offers an iterator interface over the resulting records
- Allows sorting and selection but no more complex operations
- Introduces data types
- Uppermost interface of *navigational* or *hierarchical databases*

Five Layer Architecture of a DBMS ([HR83] [HR85])



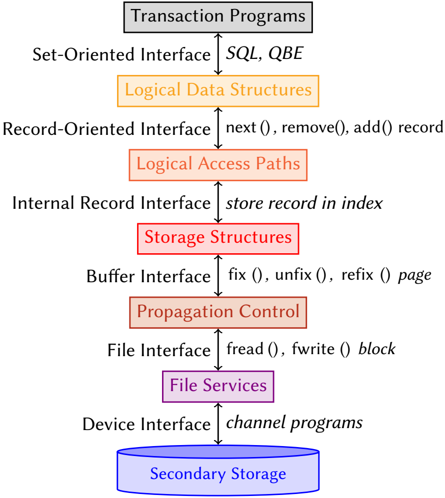
- Also called *non-procedural access layer*
- Offers complex descriptive query languages like *SQL*, *QBE* or *XQuery*
- Maps external identifiers of e.g. relations, views and tuples to internal identifiers using the *metadata management*

Five Layer Architecture of a DBMS ([HR83] [HR85])



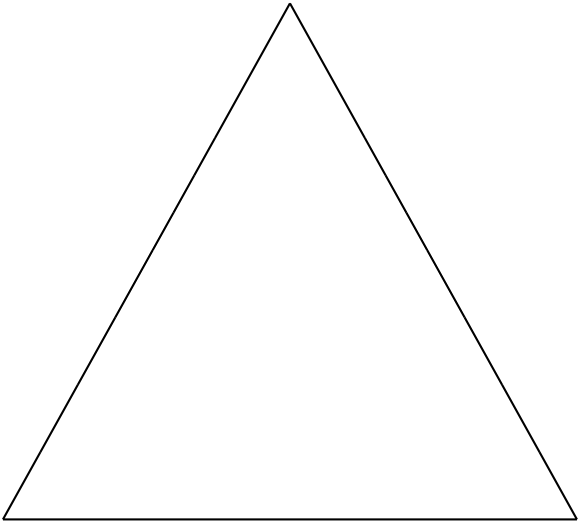
- Offers complex set operations
- Offers *data integrity*, *access control* and *transaction management*
- Most interesting target for the *query optimizer*

Five Layer Architecture of a DBMS ([HR83] [HR85])

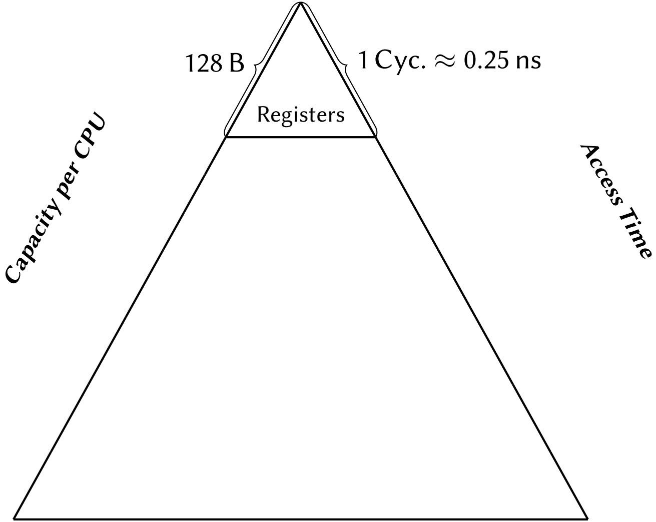


- Application programs uses the service of the DBMS to store persistent data

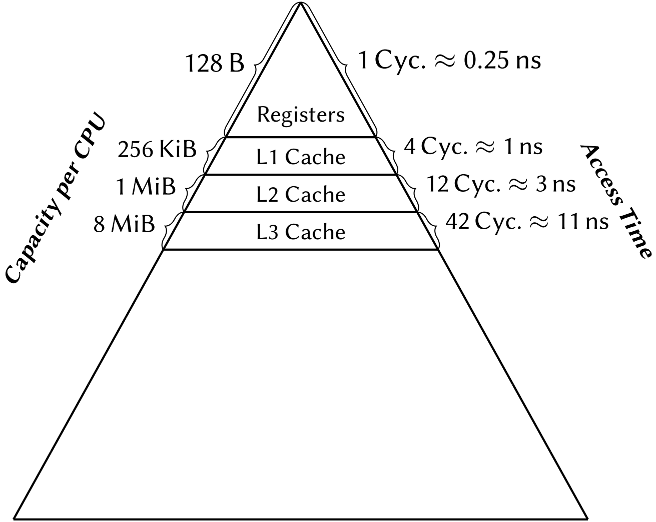
Motivation for the Usage of a Buffer Pool (1)



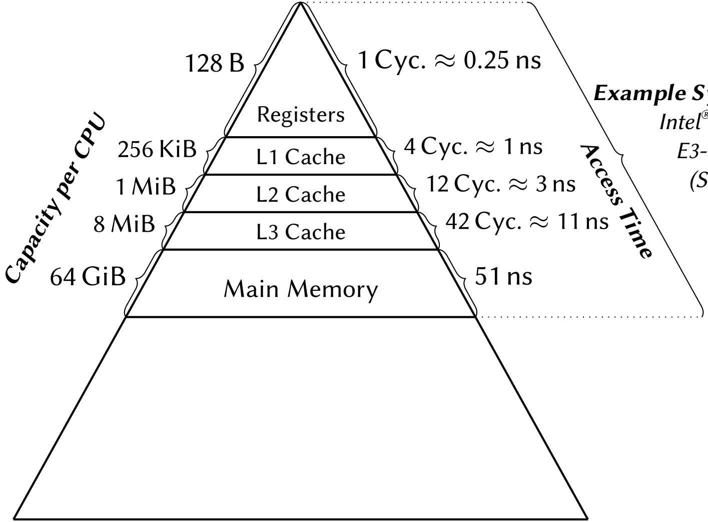
Motivation for the Usage of a Buffer Pool (1)



Motivation for the Usage of a Buffer Pool (1)

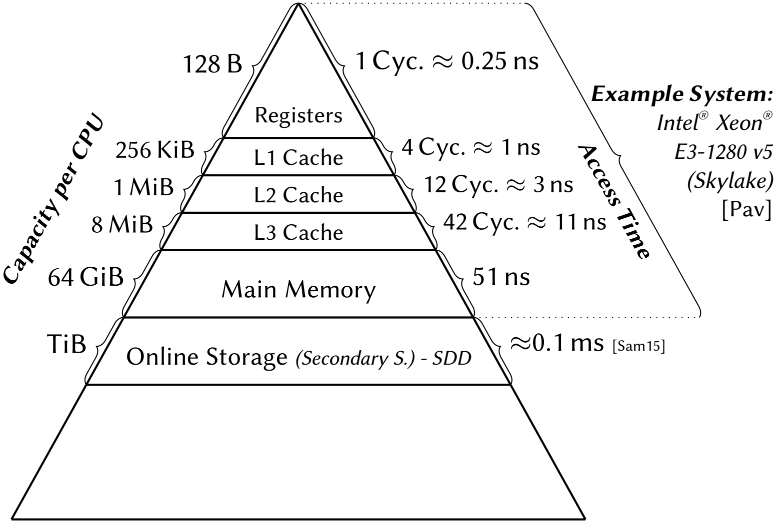


Motivation for the Usage of a Buffer Pool (1)

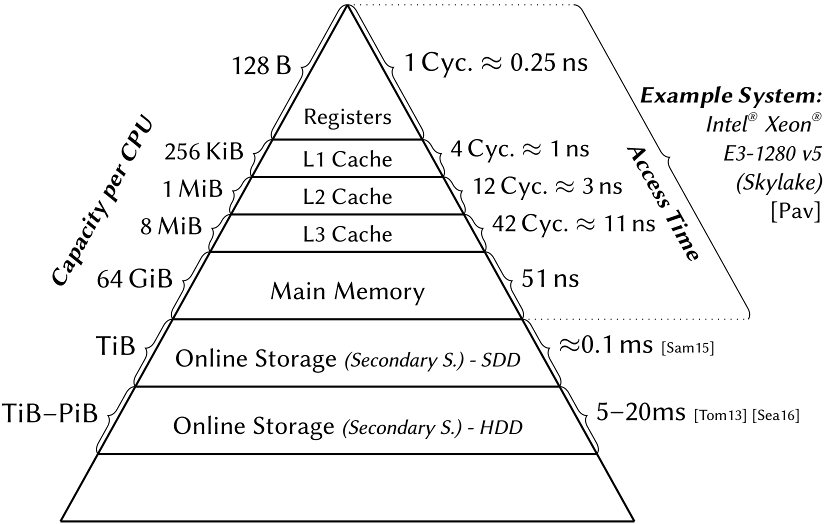


Example System:
Intel® Xeon®
E3-1280 v5
(Skylake)
[Pav]

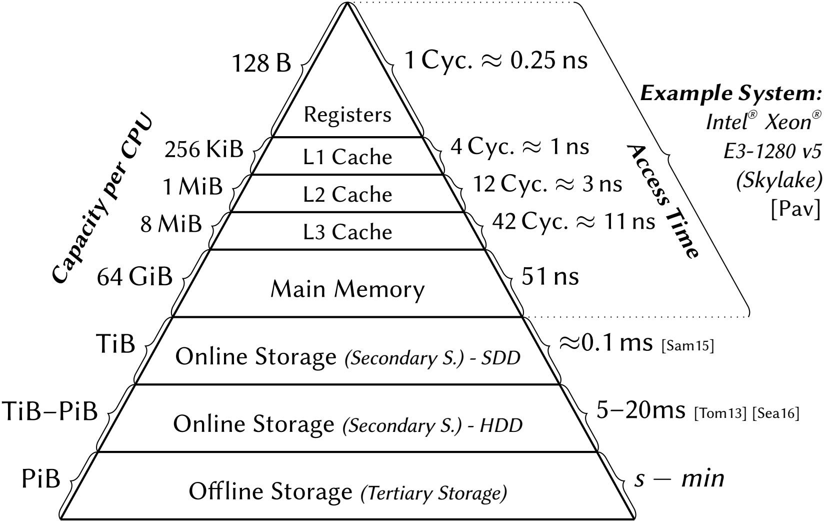
Motivation for the Usage of a Buffer Pool (1)



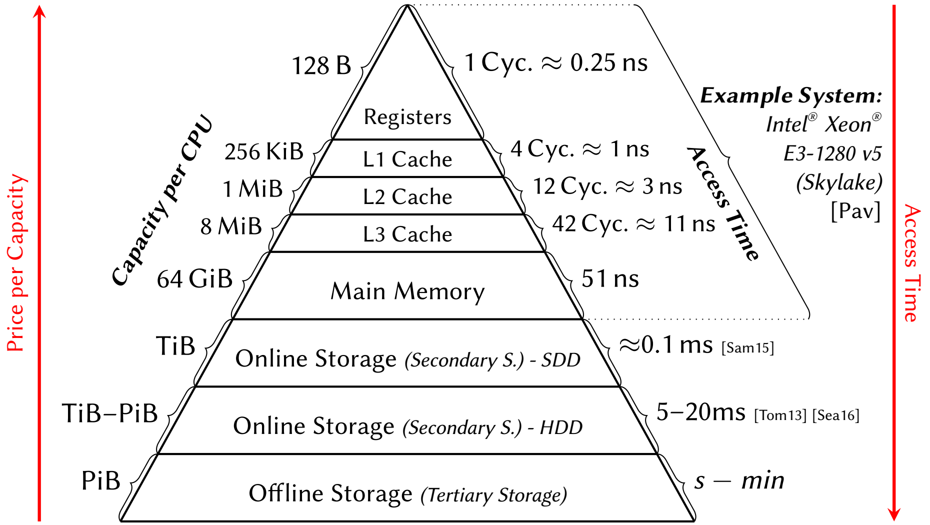
Motivation for the Usage of a Buffer Pool (1)



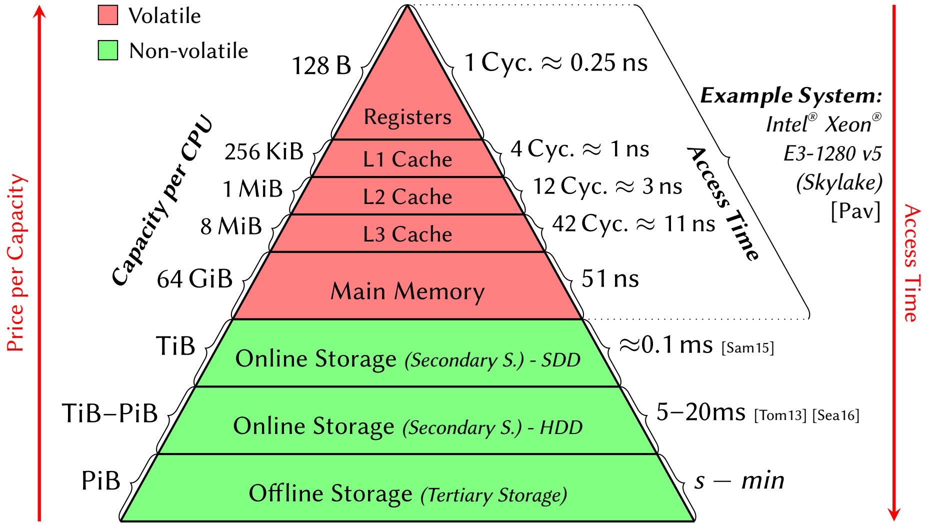
Motivation for the Usage of a Buffer Pool (1)



Motivation for the Usage of a Buffer Pool (1)



Motivation for the Usage of a Buffer Pool (1)



Motivation for the Usage of a Buffer Pool (2)

Motivation for the Usage of a Buffer Pool (2)

Requirements of Memory for a DBMS

Motivation for the Usage of a Buffer Pool (2)

Requirements of Memory for a DBMS

- **Low latency**

Motivation for the Usage of a Buffer Pool (2)

Requirements of Memory for a DBMS

- **Low latency:** Main memory

Motivation for the Usage of a Buffer Pool (2)

Requirements of Memory for a DBMS

- **Low latency:** Main memory
- **High capacity**

Motivation for the Usage of a Buffer Pool (2)

Requirements of Memory for a DBMS

- **Low latency:** Main memory
- **High capacity:** Secondary Storage

Motivation for the Usage of a Buffer Pool (2)

Requirements of Memory for a DBMS

- **Low latency:** Main memory
- **High capacity:** Secondary Storage
- **Non-volatility**

Motivation for the Usage of a Buffer Pool (2)

Requirements of Memory for a DBMS

- **Low latency:** Main memory
- **High capacity:** Secondary Storage
- **Non-volatility:** Secondary Storage

Motivation for the Usage of a Buffer Pool (2)

Requirements of Memory for a DBMS

- **Low latency:** Main memory
- **High capacity:** Secondary Storage
- **Non-volatility:** Secondary Storage
- **Byte-addressable**

Motivation for the Usage of a Buffer Pool (2)

Requirements of Memory for a DBMS

- **Low latency:** Main memory
- **High capacity:** Secondary Storage
- **Non-volatility:** Secondary Storage
- **Byte-addressable:** Main memory

Motivation for the Usage of a Buffer Pool (2)

Requirements of Memory for a DBMS

- **Low latency:** Main memory
- **High capacity:** Secondary Storage
- **Non-volatility:** Secondary Storage
- **Byte-addressable:** Main memory

$\Longrightarrow$ **Caching:** Persistently store the database on *secondary storage* and cache a subset of it in *main memory*.

Subsection 1

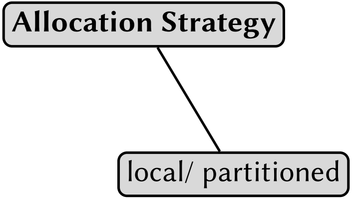
Some Tasks of the DBMS Buffer Management

Memory Allocation in the Buffer Pool ([HR01])

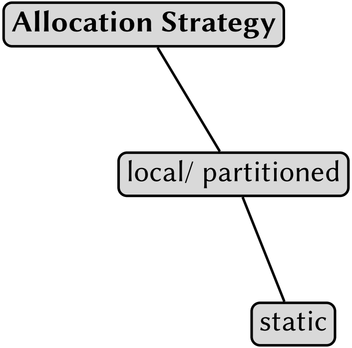
Memory Allocation in the Buffer Pool ([HR01])

Allocation Strategy

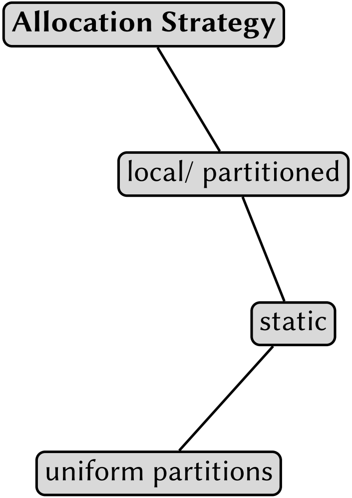
Memory Allocation in the Buffer Pool ([HR01])



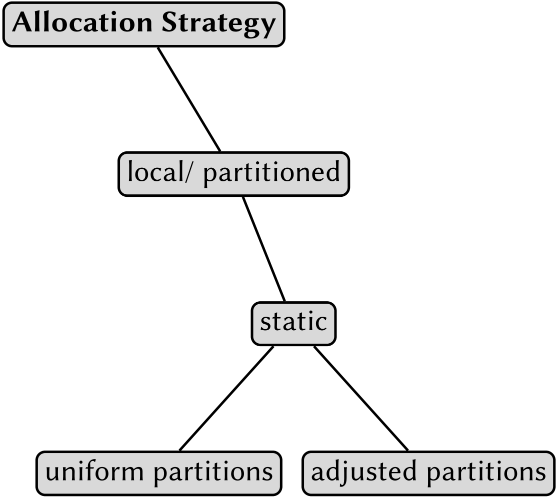
Memory Allocation in the Buffer Pool ([HR01])



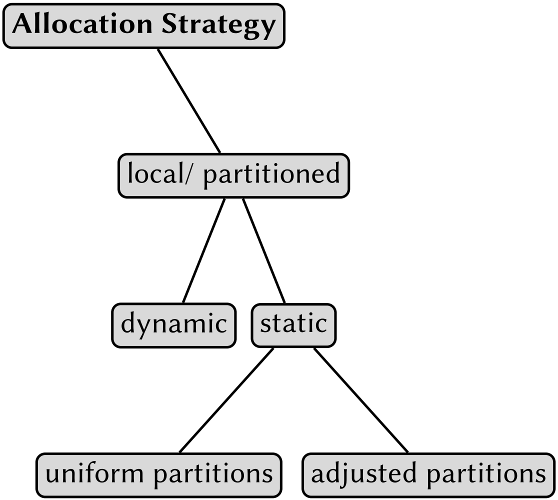
Memory Allocation in the Buffer Pool ([HR01])



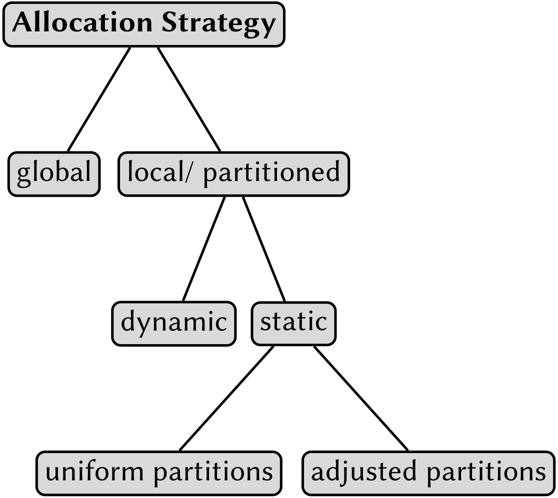
Memory Allocation in the Buffer Pool ([HR01])



Memory Allocation in the Buffer Pool ([HR01])



Memory Allocation in the Buffer Pool ([HR01])



Concurrency Control

Concurrency Control

- **Pages** get accessed concurrently

Concurrency Control

- **Pages** get accessed concurrently
  - Latching for the pages

Concurrency Control

- **Pages** get accessed concurrently
  - Latching for the pages
  - Mutually exclusive write accesses

Concurrency Control

- **Pages** get accessed concurrently
  - Latching for the pages
  - Mutually exclusive write accesses
  - Concurrent read accesses

Concurrency Control

- **Pages** get accessed concurrently
  - Latching for the pages
  - Mutually exclusive write accesses
  - Concurrent read accesses
- **Auxiliary data** gets accessed concurrently

Concurrency Control

- **Pages** get accessed concurrently
  - Latching for the pages
  - Mutually exclusive write accesses
  - Concurrent read accesses
- **Auxiliary data** gets accessed concurrently
  - Latching for the auxiliary data structures

Concurrency Control

- **Pages** get accessed concurrently
  - Latching for the pages
  - Mutually exclusive write accesses
  - Concurrent read accesses
- **Auxiliary data** gets accessed concurrently
  - Latching for the auxiliary data structures
  - … or the usage of special concurrent implementations of the data structures

Subsection 2

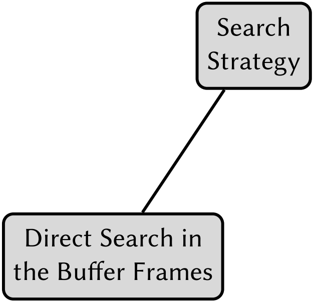
Locate Pages in the Buffer Pool without Pointer Swizzling

Overview of Search Strategies ([HR01])

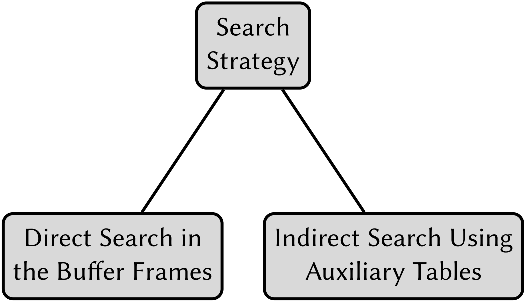
Overview of Search Strategies ([HR01])

Search Strategy

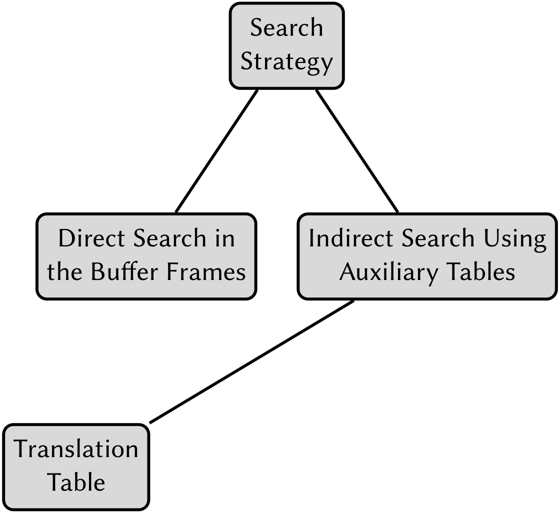
Overview of Search Strategies ([HR01])



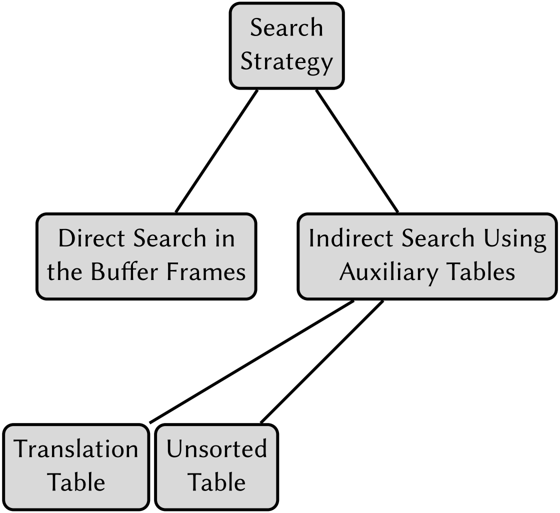
Overview of Search Strategies ([HR01])



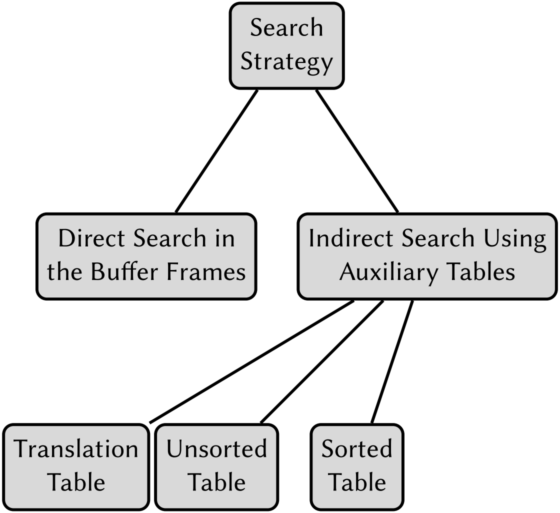
Overview of Search Strategies ([HR01])



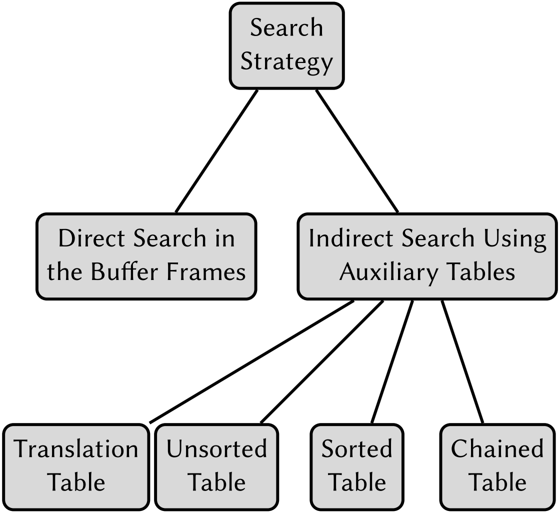
Overview of Search Strategies ([HR01])



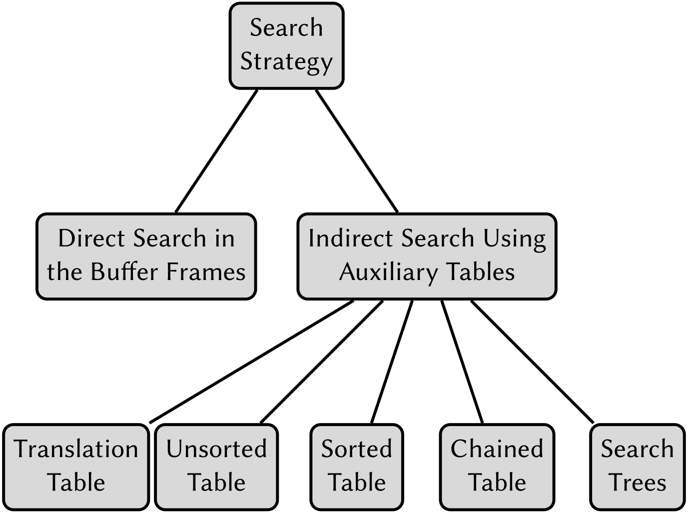
Overview of Search Strategies ([HR01])



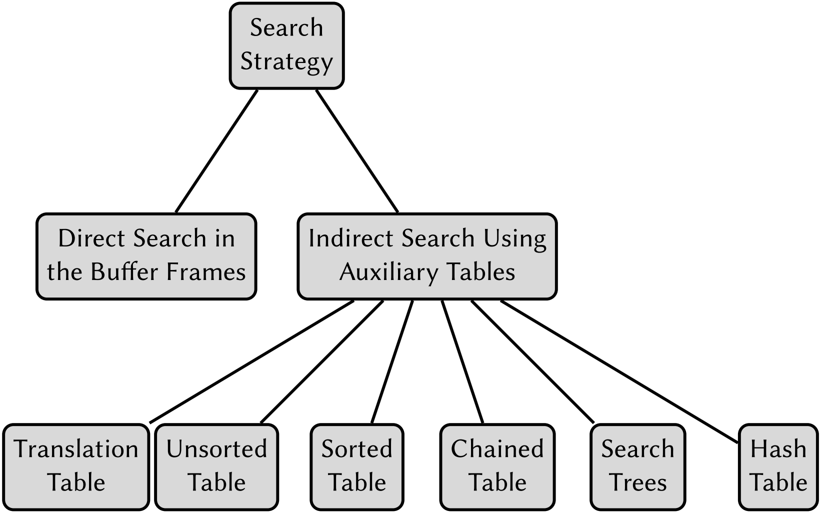
Overview of Search Strategies ([HR01])



Overview of Search Strategies ([HR01])



Overview of Search Strategies ([HR01])



Direct Search in the Buffer Frames & Unsorted Table

Direct Search in the Buffer Frames & Unsorted Table

Direct Search in the Buffer Frames

Direct Search in the Buffer Frames & Unsorted Table

Direct Search in the Buffer Frames

- Checks in each buffer frame the page ID of the contained page

Direct Search in the Buffer Frames & Unsorted Table

Direct Search in the Buffer Frames

- Checks in each buffer frame the page ID of the contained page
- $T_{\text{avg}}^{\text{search}} \in \mathcal{O}\left(\frac{n}{2}\right)$, $T_{\text{worst}}^{\text{search}} \in \mathcal{O}\left(n\right)$

Direct Search in the Buffer Frames & Unsorted Table

Direct Search in the Buffer Frames

- Checks in each buffer frame the page ID of the contained page
- $T_{\text{avg}}^{\text{search}} \in \mathcal{O}\left(\frac{n}{2}\right)$, $T_{\text{worst}}^{\text{search}} \in \mathcal{O}\left(n\right)$
- The usage of virtual memory management can result in extensive swapping due to read access to many pages!

Direct Search in the Buffer Frames & Unsorted Table

Direct Search in the Buffer Frames

- Checks in each buffer frame the page ID of the contained page
- $T_{\text{avg}}^{\text{search}} \in \mathcal{O}\left(\frac{n}{2}\right)$, $T_{\text{worst}}^{\text{search}} \in \mathcal{O}\left(n\right)$
- The usage of virtual memory management can result in extensive swapping due to read access to many pages!

Unsorted Table

Direct Search in the Buffer Frames & Unsorted Table

Direct Search in the Buffer Frames

- Checks in each buffer frame the page ID of the contained page
- $T_{\text{avg}}^{\text{search}} \in \mathcal{O}\left(\frac{n}{2}\right)$, $T_{\text{worst}}^{\text{search}} \in \mathcal{O}\left(n\right)$
- The usage of virtual memory management can result in extensive swapping due to read access to many pages!

Unsorted Table

- Auxiliary data structure of size $S_{\text{pace}} \in \mathcal{O}\left(n\right)$

0	1	2	3	4	5	6	7	8
7785	**6977**	**4347**	**3380**	**5610**	**6376**	**4877**	**3332**	**3354**

Figure: An unsorted table used to map buffer frames to page IDs.

Direct Search in the Buffer Frames & Unsorted Table

Direct Search in the Buffer Frames

- Checks in each buffer frame the page ID of the contained page
- $T_{\text{avg}}^{\text{search}} \in \mathcal{O}\left(\frac{n}{2}\right)$, $T_{\text{worst}}^{\text{search}} \in \mathcal{O}\left(n\right)$
- The usage of virtual memory management can result in extensive swapping due to read access to many pages!

Unsorted Table

- Auxiliary data structure of size $S_{\text{pace}} \in \mathcal{O}\left(n\right)$
- $T_{\text{avg}}^{\text{search}} \in \mathcal{O}\left(\frac{n}{2}\right)$, $T_{\text{worst}}^{\text{search}} \in \mathcal{O}\left(n\right)$

0	1	2	3	4	5	6	7	8
7785	**6977**	**4347**	**3380**	**5610**	**6376**	**4877**	**3332**	**3354**

Figure: An unsorted table used to map buffer frames to page IDs.

Translation Table

Translation Table

- Auxiliary data structure with one entry per page in the database
  $\Longrightarrow \; S_{\text{pace}} \in \mathcal{O}(p)$

0	·
⋮	
3331	·
3332	7
3333	·
⋮	

⋮		⋮		⋮		⋮		⋮		⋮		⋮		⋮	
3352	·	3378	·	4345	·	4875	·	5608	·	6374	·	6975	·	7783	·
3353	·	3379	·	4346	·	4876	·	5609	·	6375	·	6976	·	7784	·
3354	8	3380	3	4347	2	4877	6	5610	4	6376	5	6977	1	7785	0
3355	·	3381	·	4348	·	4878	·	5611	·	6377	·	6978	·	7786	·
3356	·	3382	·	4349	·	4879	·	5612	·	6378	·	6979	·	7787	·
⋮		⋮		⋮		⋮		⋮		⋮		⋮		⋮	

Figure: A translation table used to map page IDs to buffer frames.

Translation Table

- Auxiliary data structure with one entry per page in the database $\implies S_{\text{pace}} \in \mathcal{O}(p)$
- $T^{\text{search}} \in \mathcal{O}(1)$, $T^{\text{insert}} \in \mathcal{O}(1)$

0	·
⋮	
3331	·
3332	7
3333	·
⋮	

⋮		⋮		⋮		⋮		⋮		⋮		⋮		⋮	
3352	·	3378	·	4345	·	4875	·	5608	·	6374	·	6975	·	7783	·
3353	·	3379	·	4346	·	4876	·	5609	·	6375	·	6976	·	7784	·
3354	8	3380	3	4347	2	4877	6	5610	4	6376	5	6977	1	7785	0
3355	·	3381	·	4348	·	4878	·	5611	·	6377	·	6978	·	7786	·
3356	·	3382	·	4349	·	4879	·	5612	·	6378	·	6979	·	7787	·
⋮		⋮		⋮		⋮		⋮		⋮		⋮		⋮	

Figure: A translation table used to map page IDs to buffer frames.

Sorted & Chained Table

Sorted & Chained Table

Sorted Table

Sorted & Chained Table

Sorted Table

- Auxiliary data structure using a table sorted by page ID only containing cached pages

3332	**3354**	**3380**	**4347**	**4877**	**5610**	**6376**	**6977**	**7785**
$\rightarrow$ **7**	$\rightarrow$ **8**	$\rightarrow$ **3**	$\rightarrow$ **2**	$\rightarrow$ **6**	$\rightarrow$ **4**	$\rightarrow$ **5**	$\rightarrow$ **1**	$\rightarrow$ **0**

Figure: A sorted table used to map page IDs to buffer frames.

Sorted & Chained Table

Sorted Table

- Auxiliary data structure using a table sorted by page ID only containing cached pages
- $T_{\text{avg}}^{\text{search}} \in \mathcal{O}\left(\log_2 n\right)$, $T_{\text{avg}}^{\text{insert}} \in \mathcal{O}\left(n \log_2 n\right)$

3332	**3354**	**3380**	**4347**	**4877**	**5610**	**6376**	**6977**	**7785**
$\rightarrow$ **7**	$\rightarrow$ **8**	$\rightarrow$ **3**	$\rightarrow$ **2**	$\rightarrow$ **6**	$\rightarrow$ **4**	$\rightarrow$ **5**	$\rightarrow$ **1**	$\rightarrow$ **0**

Figure: A sorted table used to map page IDs to buffer frames.

Sorted & Chained Table

Sorted Table

- Auxiliary data structure using a table sorted by page ID only containing cached pages
- $T_{\text{avg}}^{\text{search}} \in \mathcal{O}\left(\log_2 n\right)$, $T_{\text{avg}}^{\text{insert}} \in \mathcal{O}\left(n \log_2 n\right)$

3332	**3354**	**3380**	**4347**	**4877**	**5610**	**6376**	**6977**	**7785**
$\rightarrow$ **7**	$\rightarrow$ **8**	$\rightarrow$ **3**	$\rightarrow$ **2**	$\rightarrow$ **6**	$\rightarrow$ **4**	$\rightarrow$ **5**	$\rightarrow$ **1**	$\rightarrow$ **0**

Figure: A sorted table used to map page IDs to buffer frames.

Chained Table

Sorted & Chained Table

Sorted Table

- Auxiliary data structure using a table sorted by page ID only containing cached pages
- $T_{\text{avg}}^{\text{search}} \in \mathcal{O}(\log_2 n)$, $T_{\text{avg}}^{\text{insert}} \in \mathcal{O}(n \log_2 n)$

3332 $\rightarrow$ **7**	**3354** $\rightarrow$ **8**	**3380** $\rightarrow$ **3**	**4347** $\rightarrow$ **2**	**4877** $\rightarrow$ **6**	**5610** $\rightarrow$ **4**	**6376** $\rightarrow$ **5**	**6977** $\rightarrow$ **1**	**7785** $\rightarrow$ **0**

Figure: A sorted table used to map page IDs to buffer frames.

Chained Table

- Auxiliary data structure using a linked list sorted by page ID only containing cached pages

3332 $\rightarrow$ **7**	**3354** $\rightarrow$ **8**	**3380** $\rightarrow$ **3**	**4347** $\rightarrow$ **2**	**4877** $\rightarrow$ **6**	**5610** $\rightarrow$ **4**	**6376** $\rightarrow$ **5**	**6977** $\rightarrow$ **1**	**7785** $\rightarrow$ **0**

Figure: A chained table used to map page IDs to buffer frames.

Sorted & Chained Table

Sorted Table

- Auxiliary data structure using a table sorted by page ID only containing cached pages
- $T_{\text{avg}}^{\text{search}} \in \mathcal{O}\left(\log_2 n\right)$, $T_{\text{avg}}^{\text{insert}} \in \mathcal{O}\left(n \log_2 n\right)$

3332 $\rightarrow$ **7**	**3354** $\rightarrow$ **8**	**3380** $\rightarrow$ **3**	**4347** $\rightarrow$ **2**	**4877** $\rightarrow$ **6**	**5610** $\rightarrow$ **4**	**6376** $\rightarrow$ **5**	**6977** $\rightarrow$ **1**	**7785** $\rightarrow$ **0**

Figure: A sorted table used to map page IDs to buffer frames.

Chained Table

- Auxiliary data structure using a linked list sorted by page ID only containing cached pages
- $T_{\text{avg}}^{\text{search}} \in \mathcal{O}\left(\log_2 n\right)$, $T_{\text{avg}}^{\text{insert}} \in \mathcal{O}\left(\log_2 n\right)$

3332 $\rightarrow$ **7**	**3354** $\rightarrow$ **8**	**3380** $\rightarrow$ **3**	**4347** $\rightarrow$ **2**	**4877** $\rightarrow$ **6**	**5610** $\rightarrow$ **4**	**6376** $\rightarrow$ **5**	**6977** $\rightarrow$ **1**	**7785** $\rightarrow$ **0**

Figure: A chained table used to map page IDs to buffer frames.

Sorted & Chained Table

Sorted Table

- Auxiliary data structure using a table sorted by page ID only containing cached pages
- $T_{\text{avg}}^{\text{search}} \in \mathcal{O}\left(\log_2 n\right)$, $T_{\text{avg}}^{\text{insert}} \in \mathcal{O}\left(n \log_2 n\right)$

3332	**3354**	**3380**	**4347**	**4877**	**5610**	**6376**	**6977**	**7785**
$\rightarrow$ **7**	$\rightarrow$ **8**	$\rightarrow$ **3**	$\rightarrow$ **2**	$\rightarrow$ **6**	$\rightarrow$ **4**	$\rightarrow$ **5**	$\rightarrow$ **1**	$\rightarrow$ **0**

Figure: A sorted table used to map page IDs to buffer frames.

Chained Table

- Auxiliary data structure using a linked list sorted by page ID only containing cached pages
- $T_{\text{avg}}^{\text{search}} \in \mathcal{O}\left(\log_2 n\right)$, $T_{\text{avg}}^{\text{insert}} \in \mathcal{O}\left(\log_2 n\right)$
- Binary search requires more links!

3332	**3354**	**3380**	**4347**	**4877**	**5610**	**6376**	**6977**	**7785**
$\rightarrow$ **7**	$\rightarrow$ **8**	$\rightarrow$ **3**	$\rightarrow$ **2**	$\rightarrow$ **6**	$\rightarrow$ **4**	$\rightarrow$ **5**	$\rightarrow$ **1**	$\rightarrow$ **0**

Figure: A chained table used to map page IDs to buffer frames.

Search Trees

Search Trees

- Auxiliary data structure is similar to the one of the chained table

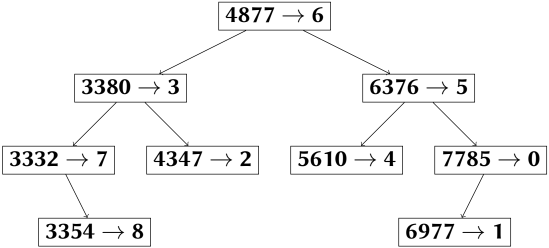


Figure: A balanced search tree used to map page IDs to buffer frames.

Search Trees

- Auxiliary data structure is similar to the one of the chained table
- Many different data structures like AVL-trees, red–black trees or splay trees can be used

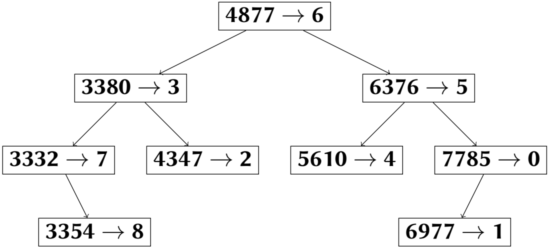


Figure: A balanced search tree used to map page IDs to buffer frames.

Search Trees

- Auxiliary data structure is similar to the one of the chained table
- Many different data structures like AVL-trees, red–black trees or splay trees can be used
- $T_{\text{avg}}^{\text{search}} \in \mathcal{O}(\log n)$, $T_{\text{avg}}^{\text{insert}} \in \mathcal{O}(\log n)$

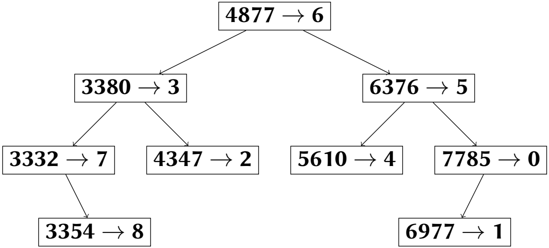


Figure: A balanced search tree used to map page IDs to buffer frames.

Search Trees

- Auxiliary data structure is similar to the one of the chained table
- Many different data structures like AVL-trees, red–black trees or splay trees can be used
- $T_{\text{avg}}^{\text{search}} \in \mathcal{O}(\log n)$, $T_{\text{avg}}^{\text{insert}} \in \mathcal{O}(\log n)$
- The worst case costs and the worst cases vary between the different search tree data structures

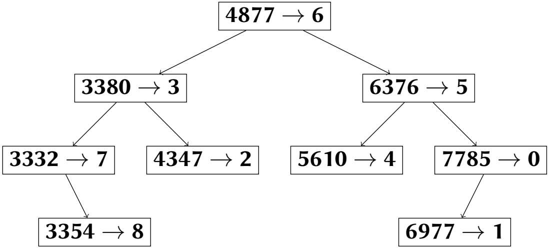
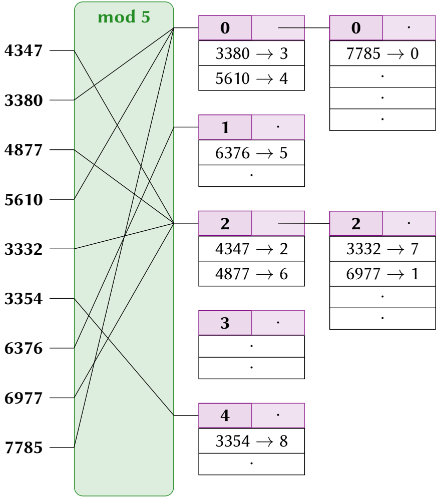


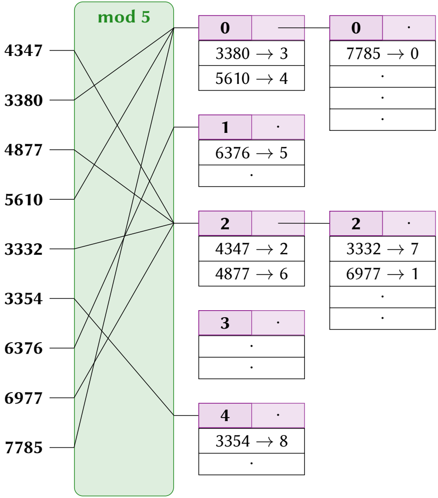
Figure: A balanced search tree used to map page IDs to buffer frames.

Hash Table

Hash Table

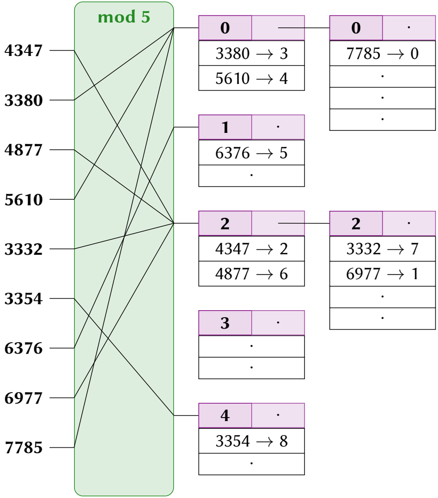

mod 5
4347
3380
4877
5610
3332
3354
6376
6977
7785
0
3380 → 3
5610 → 4
0
7785 → 0
1
6376 → 5
2
4347 → 2
4877 → 6
2
3332 → 7
6977 → 1
3
4
3354 → 8

Hash Table



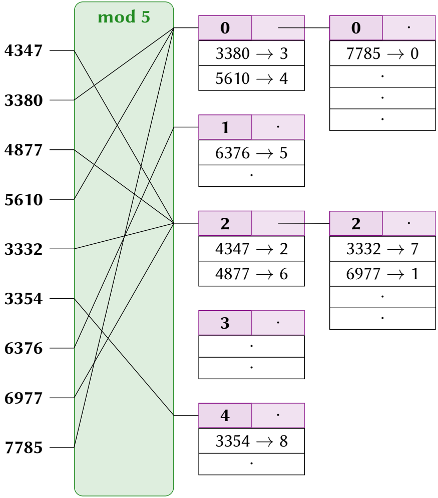
- Each page ID is mapped to a hash bucket using a hash function

Hash Table



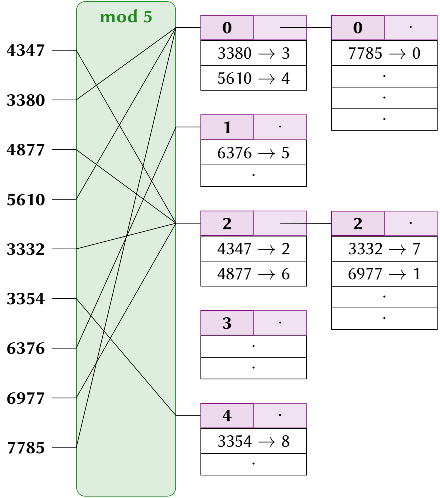
- Each page ID is mapped to a hash bucket using a hash function
- Only the page IDs of buffered pages are in the hash table

Hash Table



- Each page ID is mapped to a hash bucket using a hash function
- Only the page IDs of buffered pages are in the hash table
- If a hash bucket is full, a chained bucket gets added

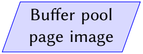
Hash Table



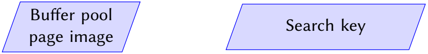
- Each page ID is mapped to a hash bucket using a hash function
- Only the page IDs of buffered pages are in the hash table
- If a hash bucket is full, a chained bucket gets added
- $T_{\text{avg}}^{\text{search}} \in \mathcal{O}(1)$, $T_{\text{avg}}^{\text{insert}} \in \mathcal{O}(1)$, $T_{\text{worst}}^{\text{search}} \in \mathcal{O}(n)$

Locate Pages in Buffer Pool with Hash Table ([Gra+14])

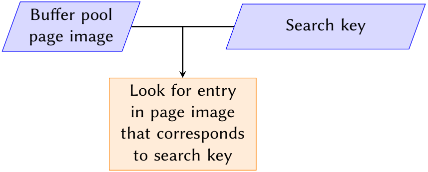
Locate Pages in Buffer Pool with Hash Table ([Gra+14])



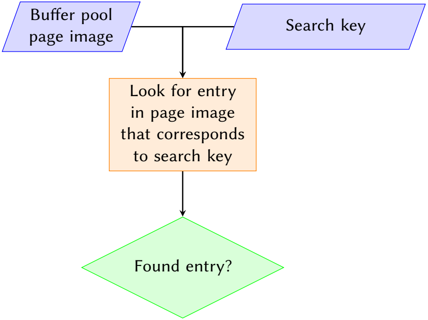
Locate Pages in Buffer Pool with Hash Table ([Gra+14])



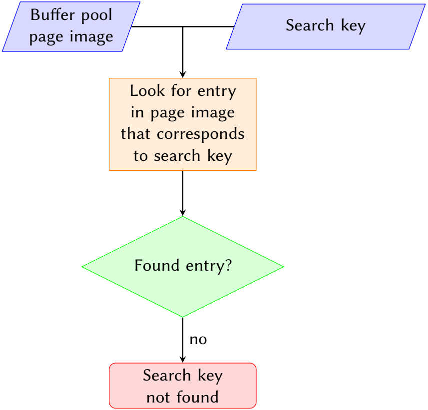
Locate Pages in Buffer Pool with Hash Table ([Gra+14])



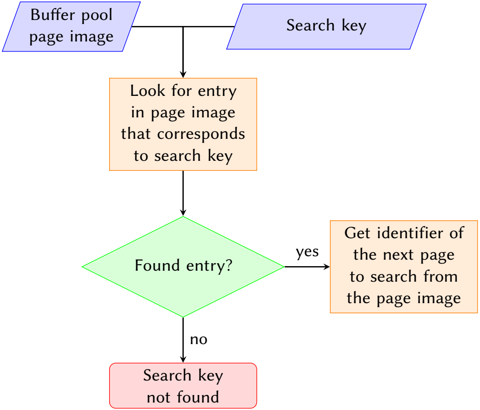
Locate Pages in Buffer Pool with Hash Table ([Gra+14])



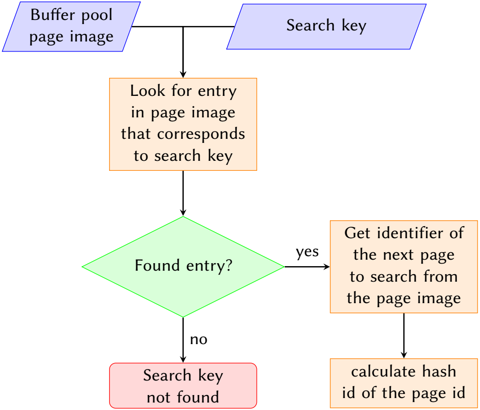
Locate Pages in Buffer Pool with Hash Table ([Gra+14])



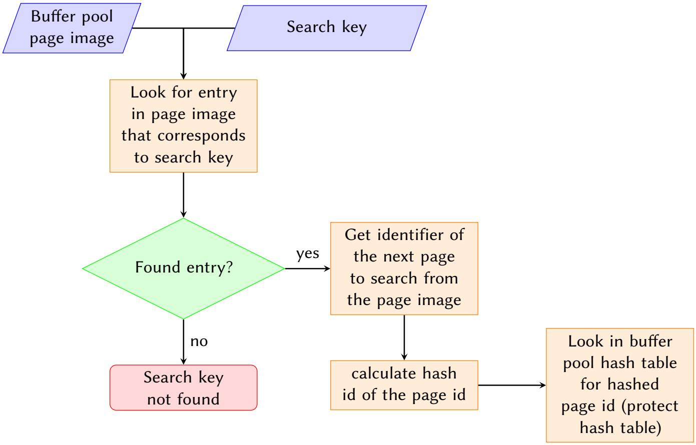
Locate Pages in Buffer Pool with Hash Table ([Gra+14])



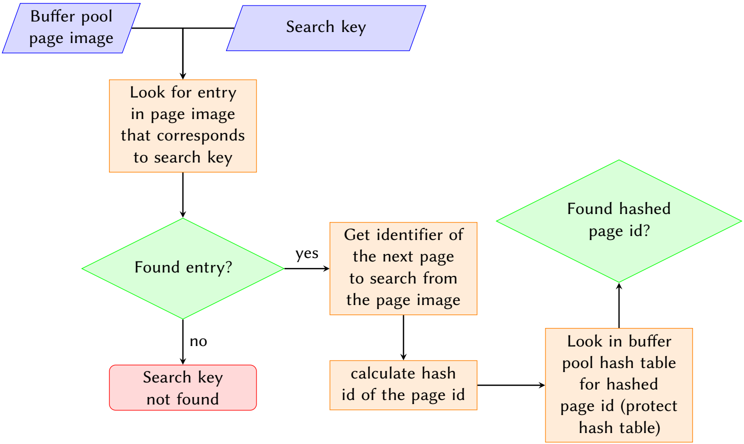
Locate Pages in Buffer Pool with Hash Table ([Gra+14])



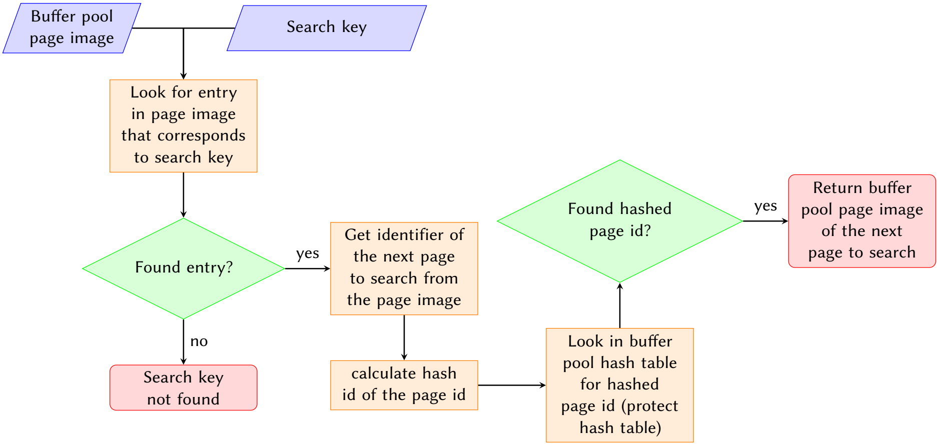
Locate Pages in Buffer Pool with Hash Table ([Gra+14])



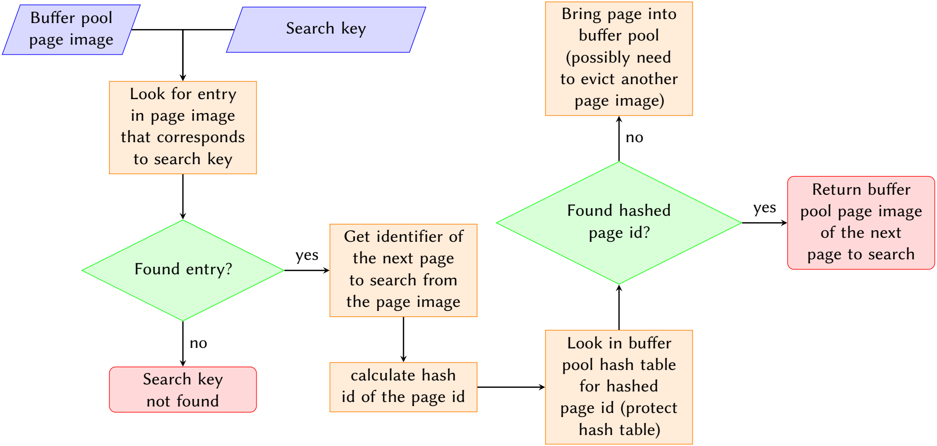
Locate Pages in Buffer Pool with Hash Table ([Gra+14])



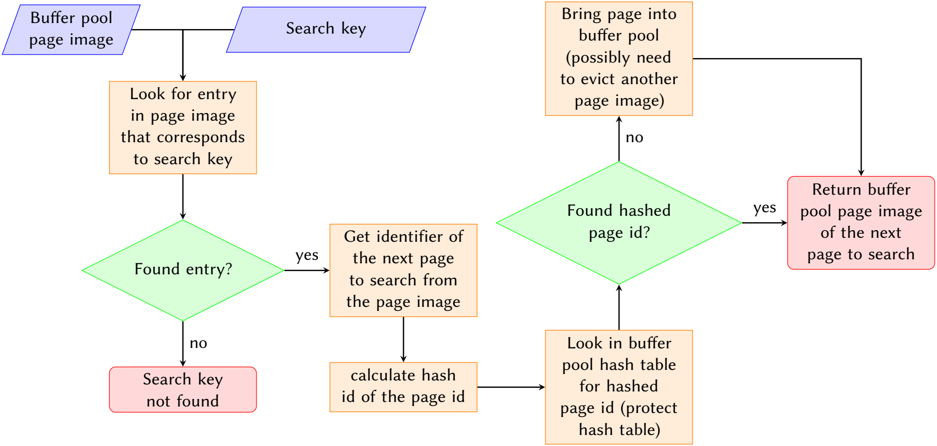
Locate Pages in Buffer Pool with Hash Table ([Gra+14])



Locate Pages in Buffer Pool with Hash Table ([Gra+14])



Locate Pages in Buffer Pool with Hash Table ([Gra+14])



Subsection 3

Locate Pages in the Buffer Pool with Pointer Swizzling

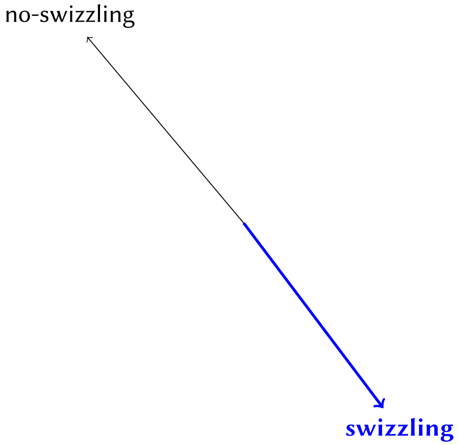
Pointer Swizzling

Definition

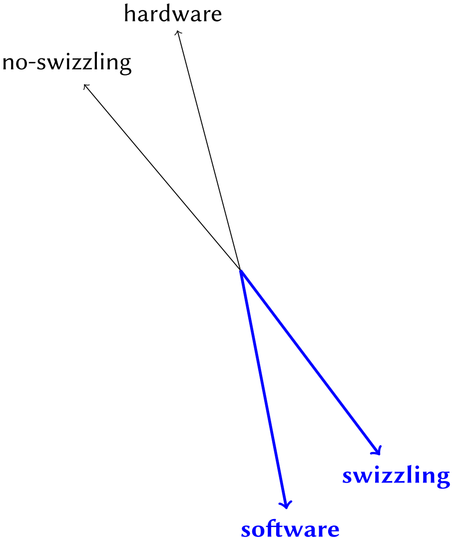
To swizzle a pointer means to transform the address of the persistent object referenced there to a more direct address of the transient object in a way that this transformation could be used during multiple indirections of this pointer ([Mos92]).

Classification of the Pointer Swizzling Approach following [WD95]

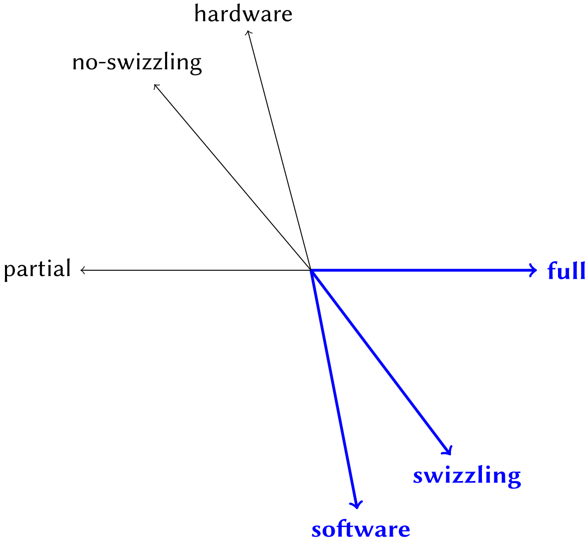
Classification of the Pointer Swizzling Approach following [WD95]



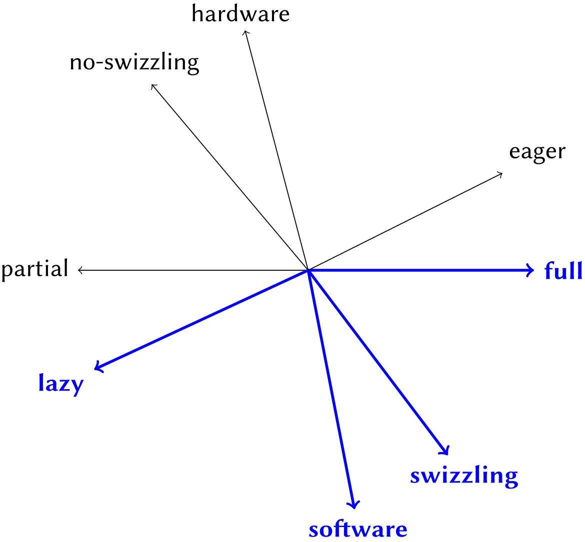
Classification of the Pointer Swizzling Approach following [WD95]



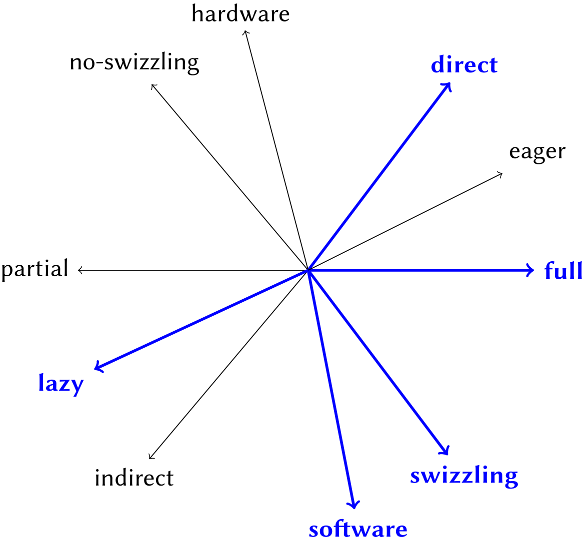
Classification of the Pointer Swizzling Approach following [WD95]



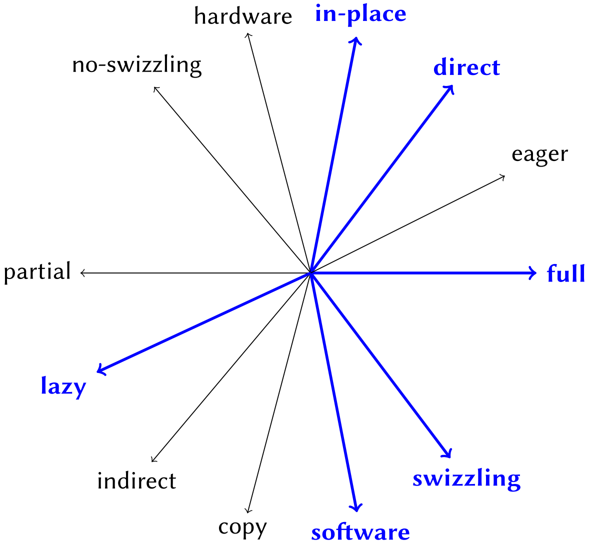
Classification of the Pointer Swizzling Approach following [WD95]



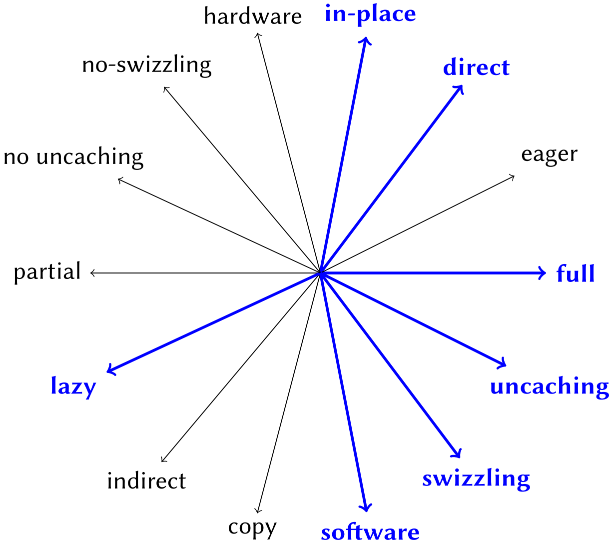
Classification of the Pointer Swizzling Approach following [WD95]



Classification of the Pointer Swizzling Approach following [WD95]

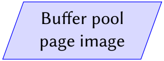


Classification of the Pointer Swizzling Approach following [WD95]



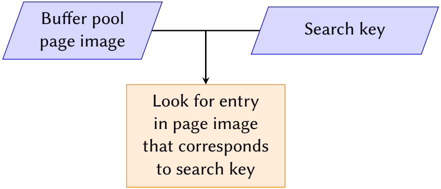
Locate Pages in Buffer Pool w/ Pointer Swizzling ([Gra+14])

Locate Pages in Buffer Pool w/ Pointer Swizzling ([Gra+14])

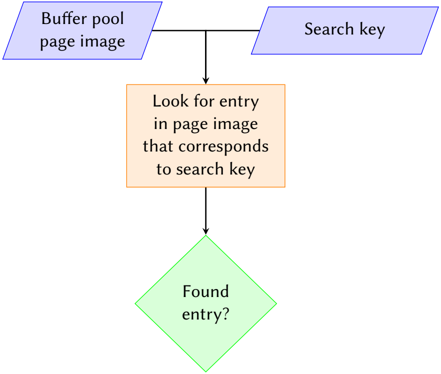


Locate Pages in Buffer Pool w/ Pointer Swizzling ([Gra+14])

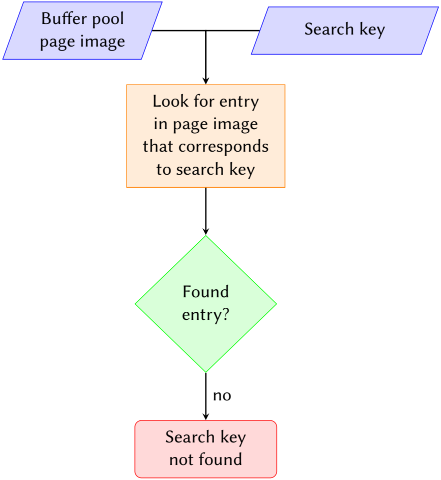
Locate Pages in Buffer Pool w/ Pointer Swizzling ([Gra+14])



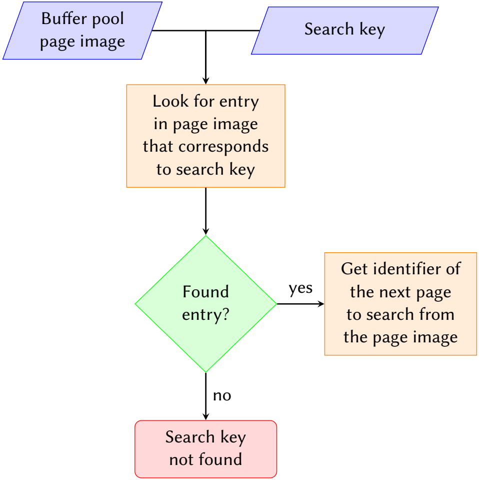
Locate Pages in Buffer Pool w/ Pointer Swizzling ([Gra+14])



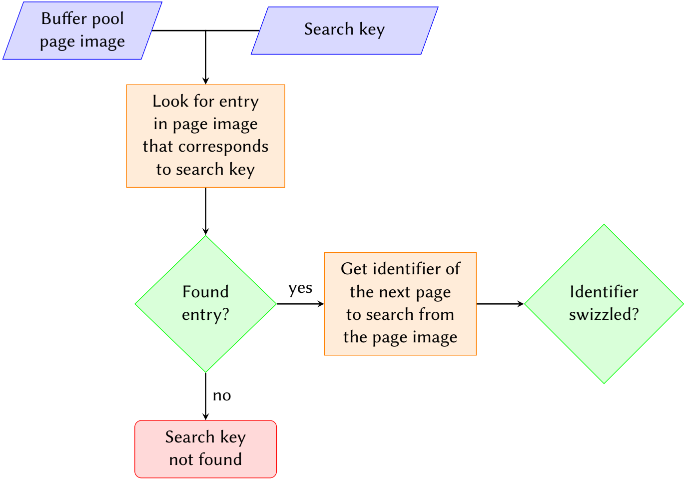
Locate Pages in Buffer Pool w/ Pointer Swizzling ([Gra+14])



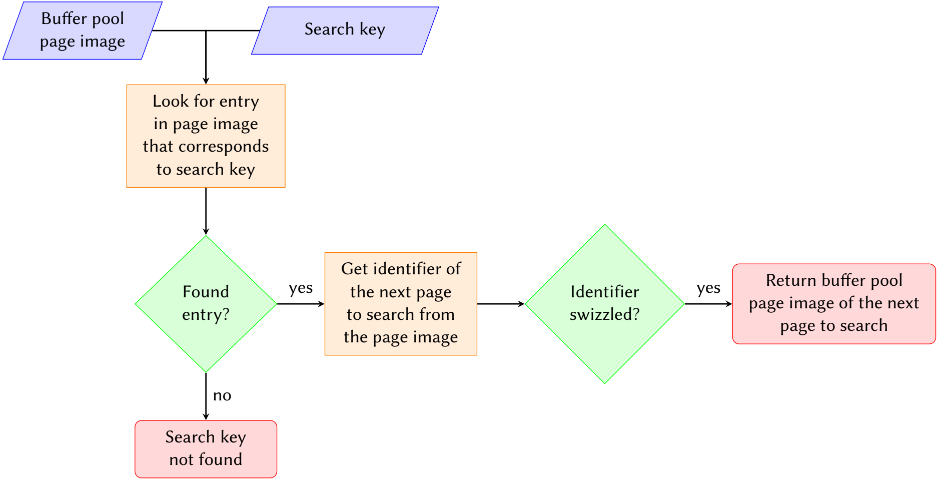
Locate Pages in Buffer Pool w/ Pointer Swizzling ([Gra+14])



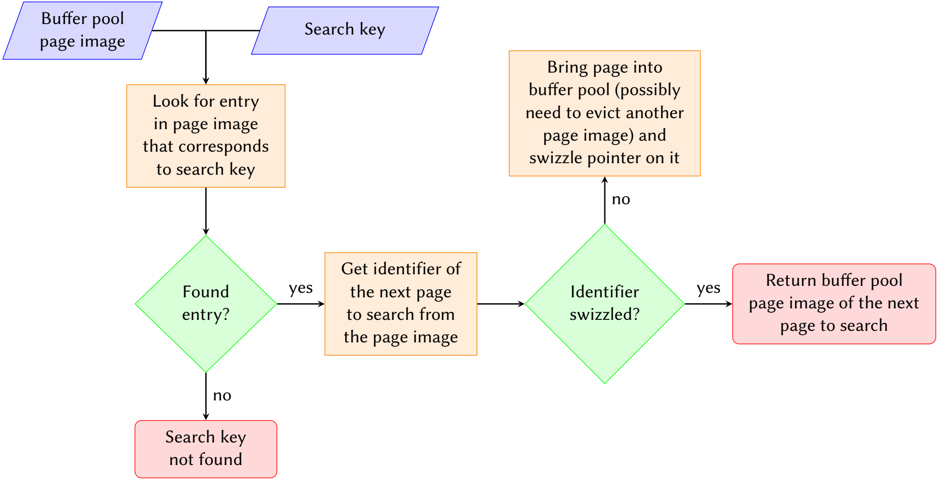
Locate Pages in Buffer Pool w/ Pointer Swizzling ([Gra+14])



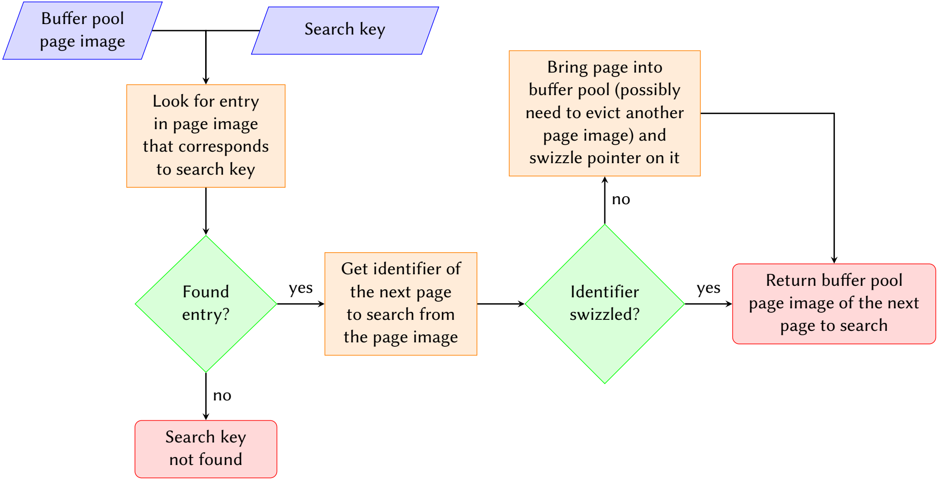
Locate Pages in Buffer Pool w/ Pointer Swizzling ([Gra+14])



Locate Pages in Buffer Pool w/ Pointer Swizzling ([Gra+14])



Locate Pages in Buffer Pool w/ Pointer Swizzling ([Gra+14])



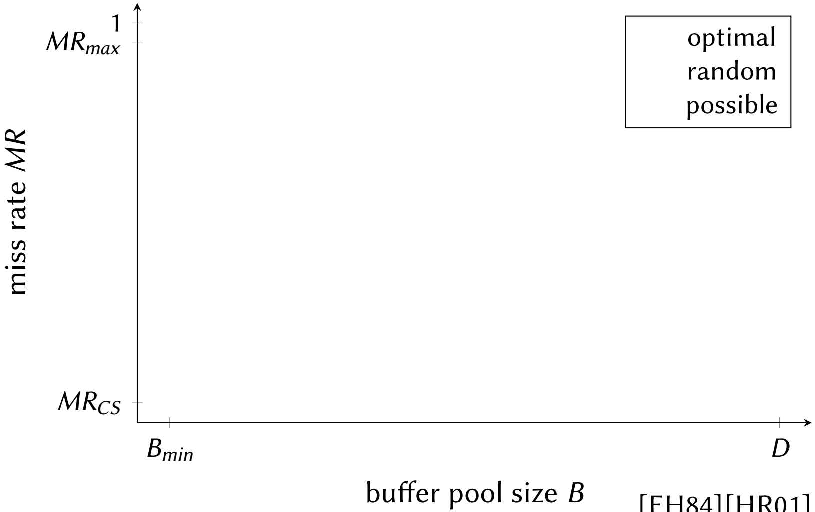
Section 2

Performance Evaluation of the Buffer Management Utilizing Pointer Swizzling

Subsection 1

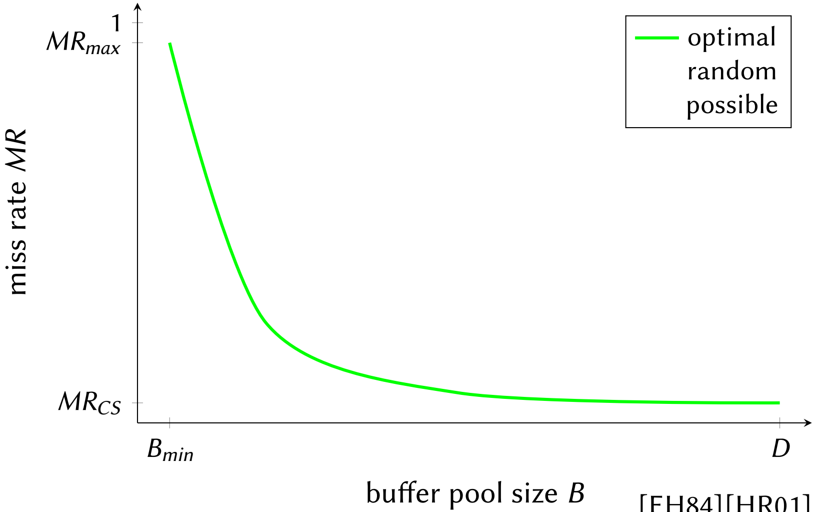
Expected Performance

Performance of Different Buffer Pool Sizes



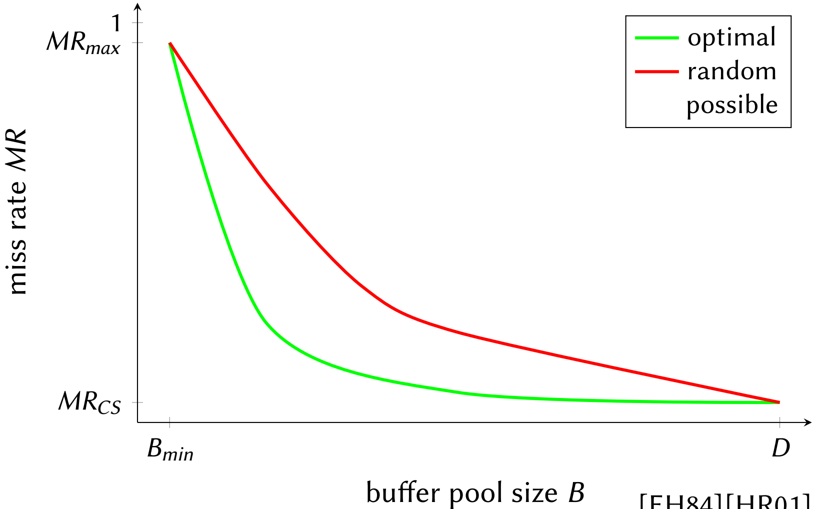
[EH84][HR01]

Performance of Different Buffer Pool Sizes



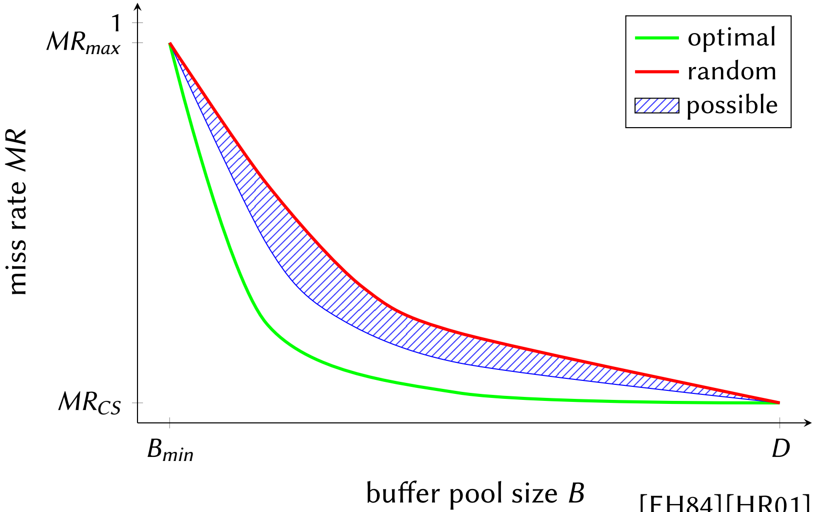
[EH84][HR01]

Performance of Different Buffer Pool Sizes



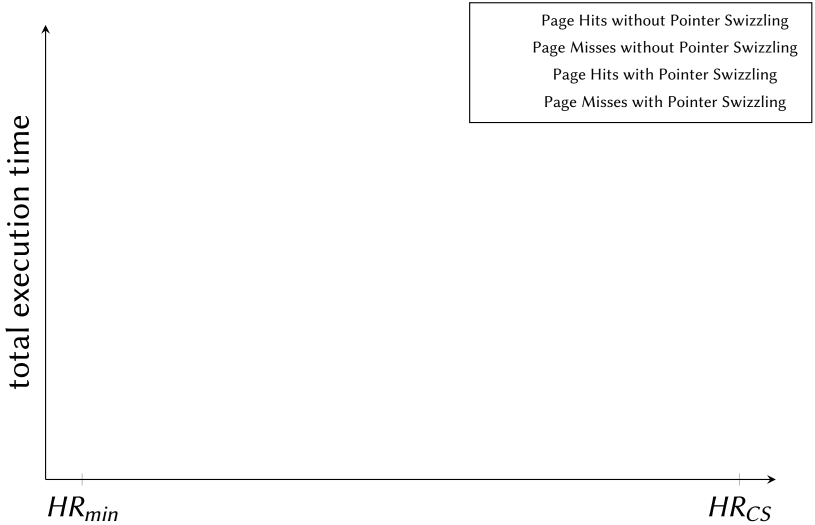
[EH84][HR01]

Performance of Different Buffer Pool Sizes

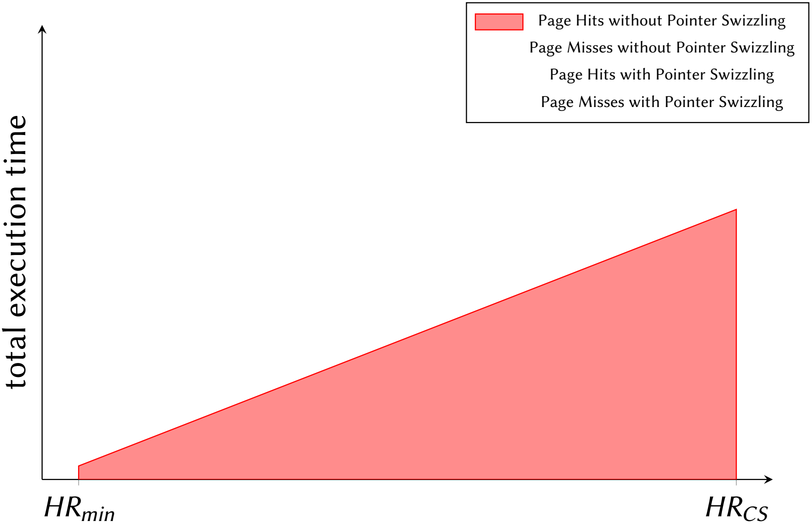


[EH84][HR01]

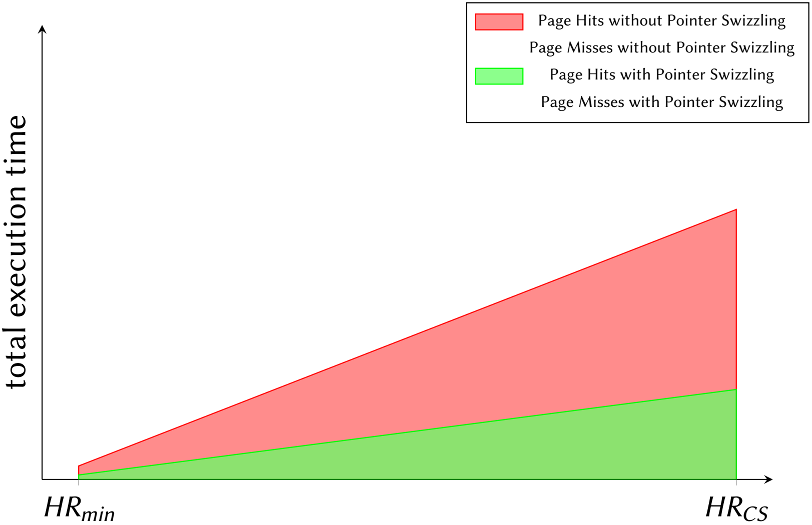
Buffer Management with and without Pointer Swizzling



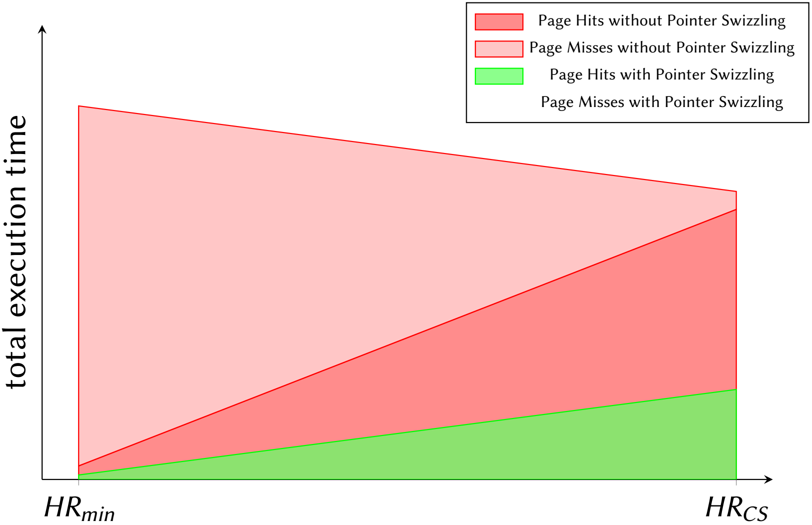
Buffer Management with and without Pointer Swizzling



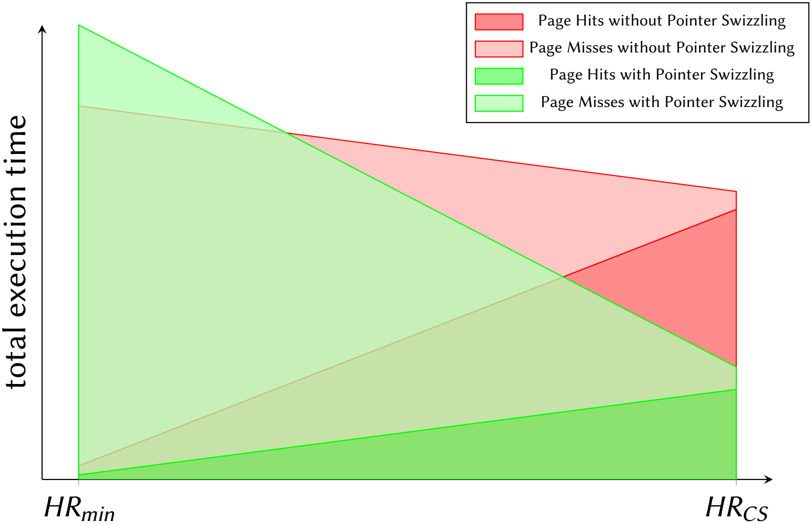
Buffer Management with and without Pointer Swizzling



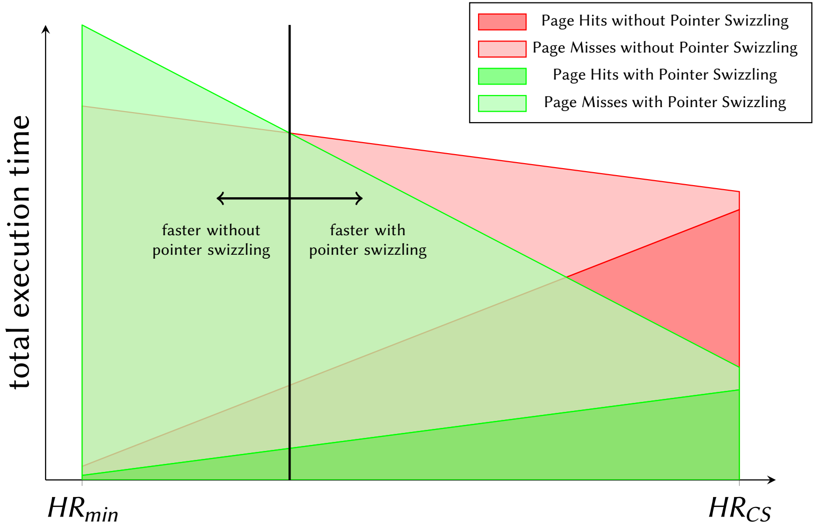
Buffer Management with and without Pointer Swizzling



Buffer Management with and without Pointer Swizzling



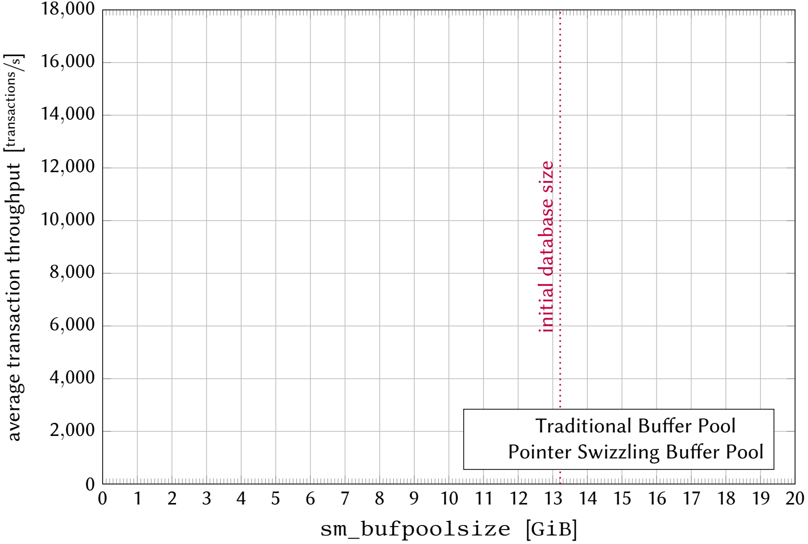
Buffer Management with and without Pointer Swizzling



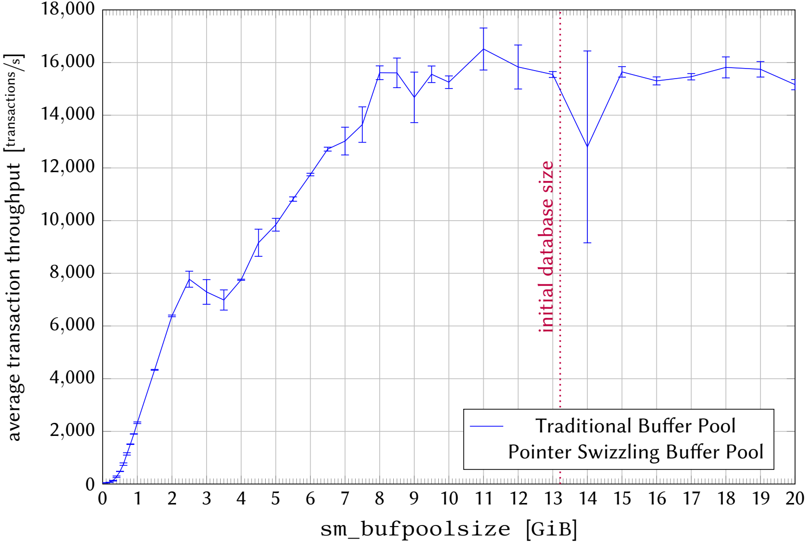
Subsection 2

Measured Performance

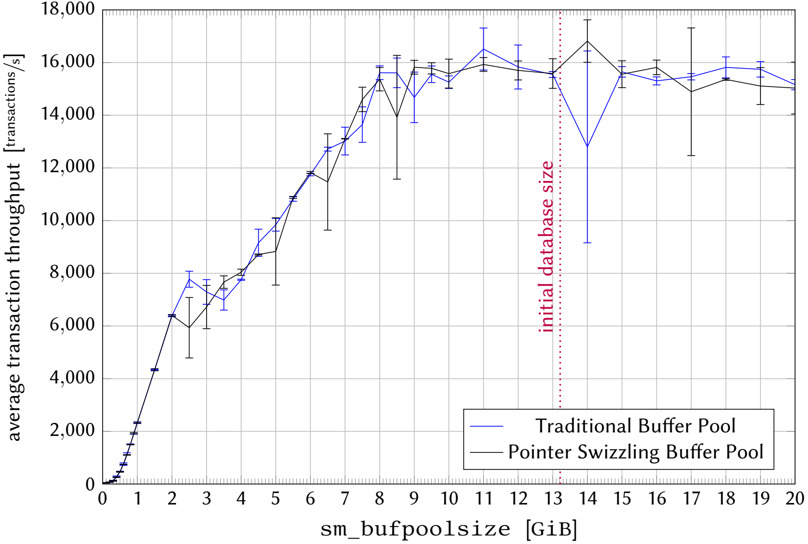
Transaction Throughput (TPC-C)



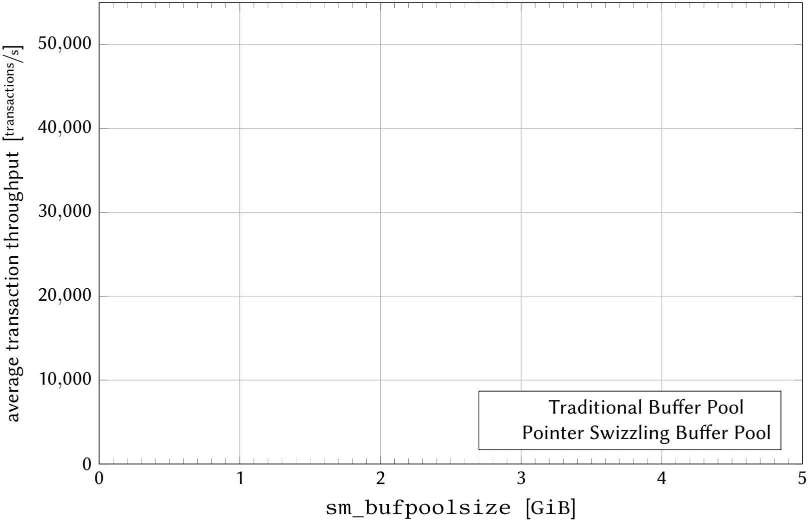
Transaction Throughput (TPC-C)



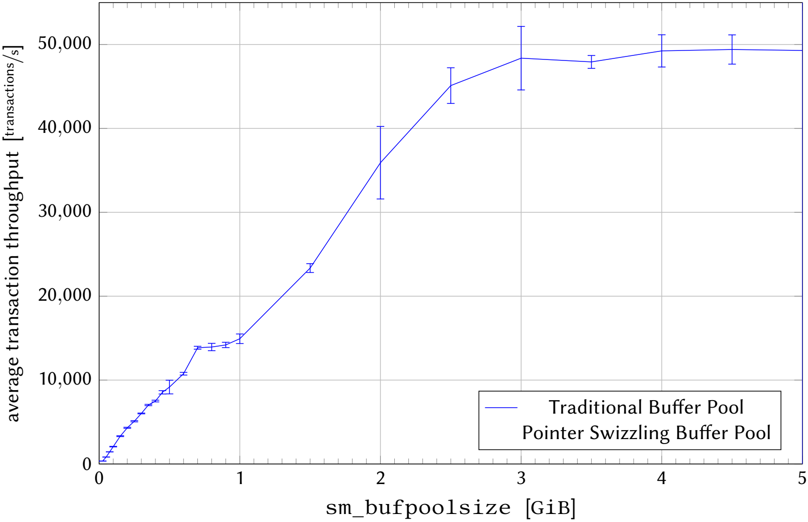
Transaction Throughput (TPC-C)



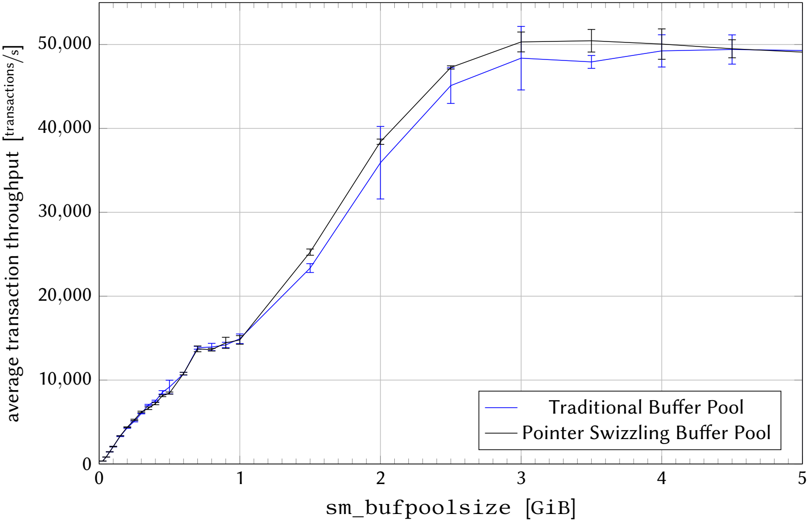
Transaction Throughput (TPC-B)



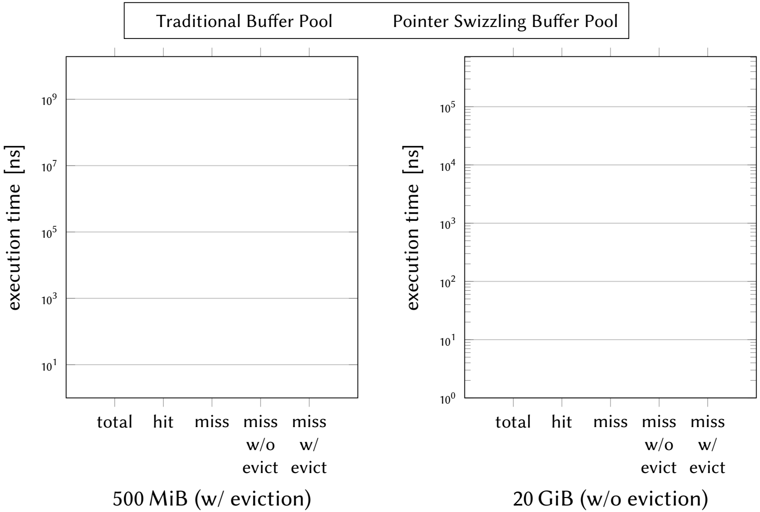
Transaction Throughput (TPC-B)



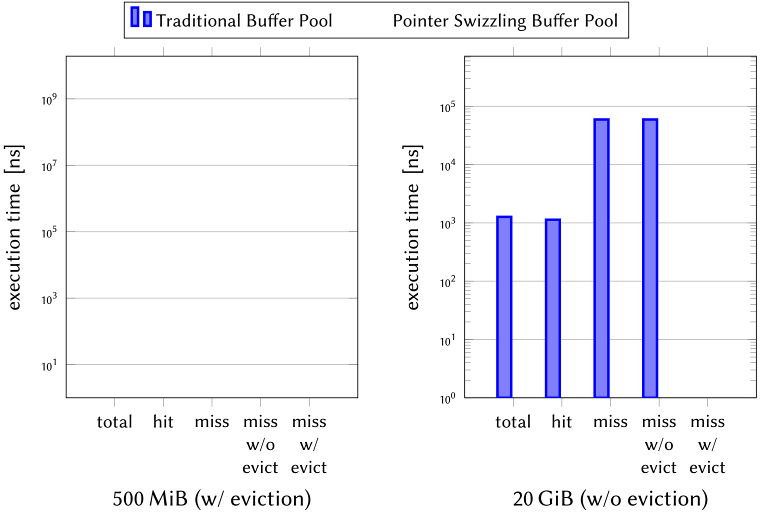
Transaction Throughput (TPC-B)



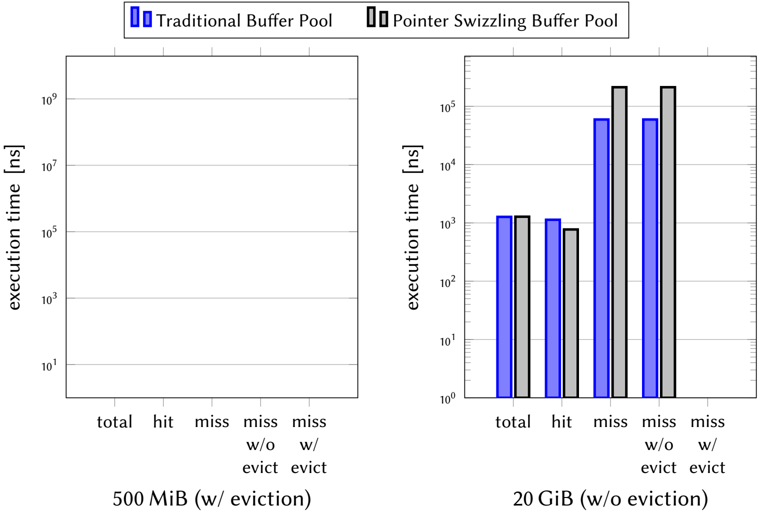
Buffer Pool Performance Acquiring Shared Latches



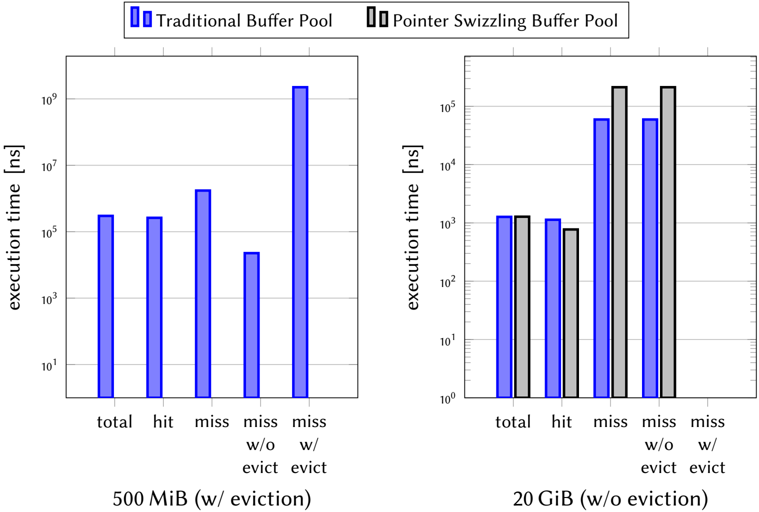
Buffer Pool Performance Acquiring Shared Latches



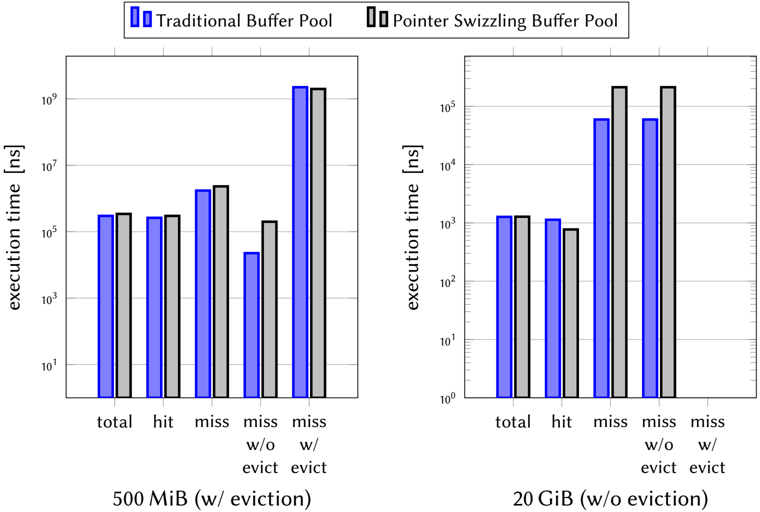
Buffer Pool Performance Acquiring Shared Latches



Buffer Pool Performance Acquiring Shared Latches



Buffer Pool Performance Acquiring Shared Latches



Subsection 3

Conclusion

Conclusion

Conclusion

Overall Performance

Conclusion

Overall Performance

- Pointer swizzling couldn't improve the performance on TPC-C benchmark runs with a duration of 10 min.

Conclusion

Overall Performance

- Pointer swizzling couldn't improve the performance on TPC-C benchmark runs with a duration of 10 min.
- The page hits after the cold start couldn't compensate the overhead of pointer swizzling during the cold start.

Conclusion

Overall Performance

- Pointer swizzling couldn't improve the performance on TPC-C benchmark runs with a duration of 10 min.
- The page hits after the cold start couldn't compensate the overhead of pointer swizzling during the cold start.
- A continuously running DB with large buffer pool could profit from pointer swizzling.

Conclusion

Overall Performance

- Pointer swizzling couldn't improve the performance on TPC-C benchmark runs with a duration of 10 min.
- The page hits after the cold start couldn't compensate the overhead of pointer swizzling during the cold start.
- A continuously running DB with large buffer pool could profit from pointer swizzling.

Buffer Pool Performance

Conclusion

Overall Performance

- Pointer swizzling couldn't improve the performance on TPC-C benchmark runs with a duration of 10 min.
- The page hits after the cold start couldn't compensate the overhead of pointer swizzling during the cold start.
- A continuously running DB with large buffer pool could profit from pointer swizzling.

Buffer Pool Performance

- A page hit is faster when pointer swizzling is activated.

Conclusion

Overall Performance

- Pointer swizzling couldn't improve the performance on TPC-C benchmark runs with a duration of 10 min.
- The page hits after the cold start couldn't compensate the overhead of pointer swizzling during the cold start.
- A continuously running DB with large buffer pool could profit from pointer swizzling.

Buffer Pool Performance

- A page hit is faster when pointer swizzling is activated.
- A page miss is slower when pointer swizzling is activated.

Conclusion

Overall Performance

- Pointer swizzling couldn't improve the performance on TPC-C benchmark runs with a duration of 10 min.
- The page hits after the cold start couldn't compensate the overhead of pointer swizzling during the cold start.
- A continuously running DB with large buffer pool could profit from pointer swizzling.

Buffer Pool Performance

- A page hit is faster when pointer swizzling is activated.
- A page miss is slower when pointer swizzling is activated.
- After the cold start phase, activated pointer swizzling will improve the buffer pool performance for large buffer pools.

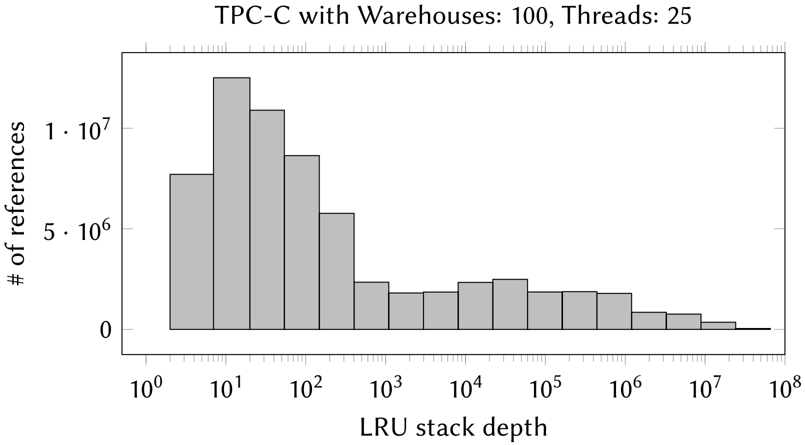
Section 3

Page Eviction Strategies in the Context of Pointer Swizzling

Motivation not to Analyze Different Page Eviction Strategies

Motivation not to Analyze Different Page Eviction Strategies

- Even LRU results in decent hit rates



But ...

But ...

- Page reference pattern containing a loop slightly to long to fit in the buffer pool:

But ...

- Page reference pattern containing a loop slightly to long to fit in the buffer pool:
    - **OPT:** Hit rate close to 1

But ...

- Page reference pattern containing a loop slightly to long to fit in the buffer pool:
  - **OPT:** Hit rate close to 1
  - **LRU:** Hit rate of 0

But ...

- Page reference pattern containing a loop slightly to long to fit in the buffer pool:
  - **OPT:** Hit rate close to 1
  - **LRU:** Hit rate of 0
- Some pages gets referenced very frequent for a limited time:

But ...

- Page reference pattern containing a loop slightly to long to fit in the buffer pool:
  - **OPT:** Hit rate close to 1
  - **LRU:** Hit rate of 0
- Some pages gets referenced very frequent for a limited time:
  - **OPT:** Pages would be evicted after their last reference

But ...

- Page reference pattern containing a loop slightly to long to fit in the buffer pool:
  - **OPT:** Hit rate close to 1
  - **LRU:** Hit rate of 0
- Some pages gets referenced very frequent for a limited time:
  - **OPT:** Pages would be evicted after their last reference
  - **LFU:** Pages waste buffer frames probably during the whole running time of the DB

But ...

- Page reference pattern containing a loop slightly to long to fit in the buffer pool:
  - **OPT:** Hit rate close to 1
  - **LRU:** Hit rate of 0
- Some pages gets referenced very frequent for a limited time:
  - **OPT:** Pages would be evicted after their last reference
  - **LFU:** Pages waste buffer frames probably during the whole running time of the DB
- Huge access time gap $\implies$ Every saved page miss significantly improves the performance

But ...

- Page reference pattern containing a loop slightly to long to fit in the buffer pool:
  - **OPT:** Hit rate close to 1
  - **LRU:** Hit rate of 0
- Some pages gets referenced very frequent for a limited time:
  - **OPT:** Pages would be evicted after their last reference
  - **LFU:** Pages waste buffer frames probably during the whole running time of the DB
- Huge access time gap $\implies$ Every saved page miss significantly improves the performance
- Pointer swizzling even amplifies that effect

Subsection 1

Probable pitfalls when Implementing a Page Eviction Strategy for a DBMS Buffer Manager

General Problems Concerning DBMS Buffer Managers

General Problems Concerning DBMS Buffer Managers

- Fixed pages cannot be evicted but a long timespan between a fix and an unfix of a page could make it a candidate for eviction.

General Problems Concerning DBMS Buffer Managers

- Fixed pages cannot be evicted but a long timespan between a fix and an unfix of a page could make it a candidate for eviction.
- A page pinned for refix cannot be evicted but a long timespan in which a page is pinned could make it a candidate for eviction.

General Problems Concerning DBMS Buffer Managers

- Fixed pages cannot be evicted but a long timespan between a fix and an unfix of a page could make it a candidate for eviction.
- A page pinned for refix cannot be evicted but a long timespan in which a page is pinned could make it a candidate for eviction.
- Dirty pages cannot be evicted but a page being dirty for a long timespan due to the update propagation using write-back policy could make it a candidate for eviction.

Additional Problem When Using Pointer Swizzling

Additional Problem When Using Pointer Swizzling

- A page containing swizzled pointer cannot be evicted but a page unfixed before the last unfix of one of its child pages could make it a candidate for eviction before its child pages got evicted.

Solutions

Solutions

- Check each of the restrictions before the eviction of a page.

Solutions

- Check each of the restrictions before the eviction of a page.
- Update the statistics of the eviction strategy during an unfix, too.

Solutions

- Check each of the restrictions before the eviction of a page.
- Update the statistics of the eviction strategy during an unfix, too.
- Update the statistics of the eviction strategy during an pin and unpin, too.

Solutions

- Check each of the restrictions before the eviction of a page.
- Update the statistics of the eviction strategy during an unfix, too.
- Update the statistics of the eviction strategy during an pin and unpin, too.
- Use write-thru for update propagation or a page cleaner decoupled from the buffer pool as proposed in [SHG16].

Solutions

- Check each of the restrictions before the eviction of a page.
- Update the statistics of the eviction strategy during an unfix, too.
- Update the statistics of the eviction strategy during an pin and unpin, too.
- Use write-thru for update propagation or a page cleaner decoupled from the buffer pool as proposed in [SHG16].
- Use a page eviction strategy that takes into account the content of pages (like the structure of an B tree).

Subsection 2

Different Page Replacement Strategies

Some Page Replacement Algorithms

Consideration during selection decision				

[EH84][HR01]

Some Page Replacement Algorithms

Consideration during selection decision		Age		

[EH84][HR01]

Some Page Replacement Algorithms

Consideration during selection decision		Age		
		No consideration		

[EH84][HR01]

Some Page Replacement Algorithms

Consideration during selection decision		**Age**		
		No consideration		
References				

[EH84][HR01]

Some Page Replacement Algorithms

Consideration during selection decision		Age		
		No consideration		
References	No consideration			

[EH84][HR01]

Some Page Replacement Algorithms

Consideration during selection decision		Age		
		No consideration	Since most recent reference	Since first reference
References	No consideration			

[EH84][HR01]

Some Page Replacement Algorithms

Consideration during selection decision		Age		
		No consideration	Since most recent reference	Since first reference
References	No consideration			
	Most recent reference			
	All references			

[EH84][HR01]

Some Page Replacement Algorithms

Consideration during selection decision		Age: No consideration	Age: Since most recent reference	Age: Since first reference
References	No consideration			FIFO
References	Most recent reference		LRU CLOCK	
References	All references	LFU	GCLOCK-V1 DGCLOCK LRU-K LRD-V2	LRD-V1

[EH84][HR01]

Some Page Replacement Algorithms

<table>
<tr><th colspan="2" rowspan="2">Consideration during selection decision</th><th colspan="3">Age</th></tr>
<tr><th>No consideration</th><th>Since most recent reference</th><th>Since first reference</th></tr>
<tr><th rowspan="4">References</th><th>No consideration</th><td>RANDOM</td><td></td><td>FIFO</td></tr>
<tr><th>Most recent reference</th><td></td><td>LRU
CLOCK</td><td></td></tr>
<tr><th rowspan="2">All references</th><td rowspan="2">LFU</td><td>GCLOCK-V1
DGCLOCK</td><td>LRD-V1</td></tr>
<tr><td colspan="2">LRU-K
LRD-V2</td></tr>
</table>

[EH84][HR01]

Some Page Replacement Algorithms

<table>
<tr><th colspan="2" rowspan="2">Consideration during selection decision</th><th colspan="3">Age</th></tr>
<tr><th>No consideration</th><th>Since most recent reference</th><th>Since first reference</th></tr>
<tr><th rowspan="4">References</th><th>No consideration</th><td>RANDOM</td><td></td><td>FIFO</td></tr>
<tr><th>Most recent reference</th><td></td><td>LRU
CLOCK
GCLOCK-V2</td><td></td></tr>
<tr><th rowspan="2">All references</th><td rowspan="2">LFU</td><td>GCLOCK-V1
DGCLOCK</td><td>LRD-V1</td></tr>
<tr><td colspan="2">LRU-K
LRD-V2</td></tr>
</table>

[EH84][HR01]

Some More Page Replacement Algorithms

PRIORITY-LRU

MRU

Clock-Pro

LIRS

WSclock

SEQ

CART

LRFU

MQ

HSS

ARC

EELRU

SLRU

PLRU

LFV

VAR-PAGE-LRU

Pannier

NFU

DEAR

[HR01][Wan01][HSS11][Wik17]

Some More Page Replacement Algorithms

PRIORITY-LRU

MRU

Clock-Pro

LIRS

CAR

WSclock

SEQ

CART

LRFU

MQ

HSS

ARC

EELRU

PLRU

LFV

SLRU

VAR-PAGE-LRU

Pannier

NFU

DEAR

[HR01][Wan01][HSS11][Wik17]

RANDOM

Overview

RANDOM

Overview

- Simplest page eviction strategy

RANDOM

Overview

- Simplest page eviction strategy
- Evicts a random page that can be evicted

RANDOM

Overview

- Simplest page eviction strategy
- Evicts a random page that can be evicted
- Won’t evict frequently used pages as they’re latched all the time

GCLOCK

Overview

GCLOCK

Overview

- Slight enhancement of the CLOCK algorithm: *generalized CLOCK*

GCLOCK

Overview

- Slight enhancement of the CLOCK algorithm: *generalized CLOCK*
- Uses finer-grained statistics about the recency of page references

GCLOCK

Overview

- Slight enhancement of the CLOCK algorithm: *generalized CLOCK*
- Uses finer-grained statistics about the recency of page references
- Parameter *k* defines granulation of statistics

GCLOCK

Overview

- Slight enhancement of the CLOCK algorithm: *generalized CLOCK*
- Uses finer-grained statistics about the recency of page references
- Parameter k defines granulation of statistics
  - $\boldsymbol{k = 1}$: CLOCK

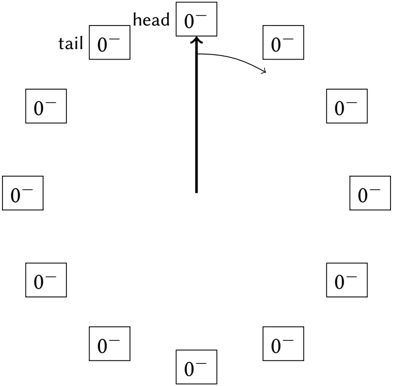
GCLOCK

Overview

- Slight enhancement of the CLOCK algorithm: *generalized CLOCK*
- Uses finer-grained statistics about the recency of page references
- Parameter k defines granulation of statistics
  - $\boldsymbol{k = 1}$**:** CLOCK
  - $\boldsymbol{k = \#}$**frames:** Similar to LRU

GCLOCK

Example



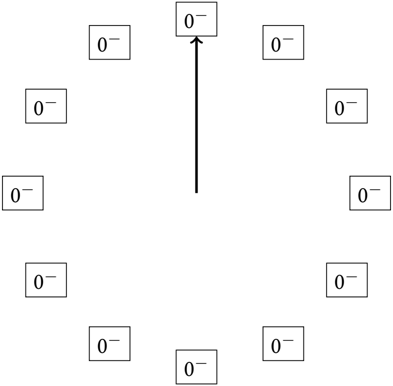
GCLOCK

Example

Cold Starting the Buffer Pool!

GCLOCK

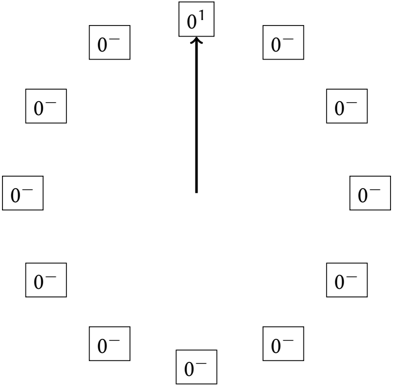
Example



Page References: 1, 2, 3, 4, 5, 6, 7, 8, 3, 4, 6, 1, 8, 16, 12, 34, 65, 3, 4, 7, 2, 65, 23, 2, 3, 5

GCLOCK

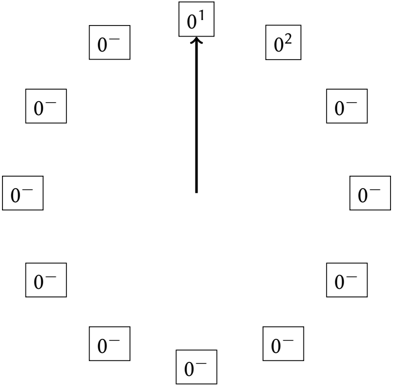
Example



Page References: 1, 2, 3, 4, 5, 6, 7, 8, 3, 4, 6, 1, 8, 16, 12, 34, 65, 3, 4, 7, 2, 65, 23, 2, 3, 5

GCLOCK

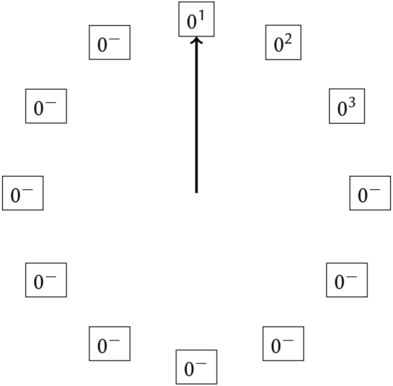
Example



Page References: 1, 2, 3, 4, 5, 6, 7, 8, 3, 4, 6, 1, 8, 16, 12, 34, 65, 3, 4, 7, 2, 65, 23, 2, 3, 5

GCLOCK

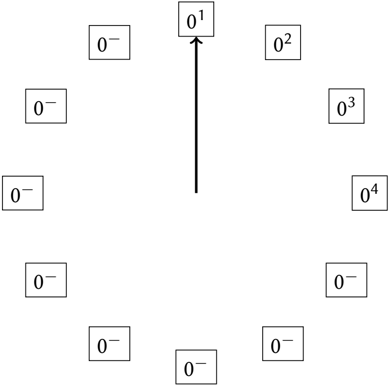
Example



Page References: 1, 2, 3, 4, 5, 6, 7, 8, 3, 4, 6, 1, 8, 16, 12, 34, 65, 3, 4, 7, 2, 65, 23, 2, 3, 5

GCLOCK

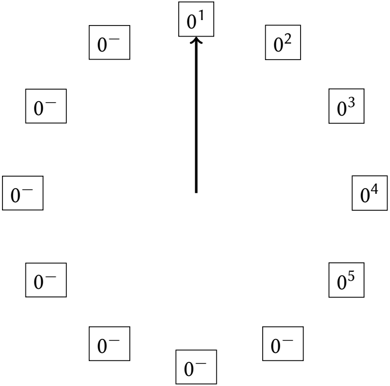
Example



Page References: 1, 2, 3, 4, 5, 6, 7, 8, 3, 4, 6, 1, 8, 16, 12, 34, 65, 3, 4, 7, 2, 65, 23, 2, 3, 5

GCLOCK

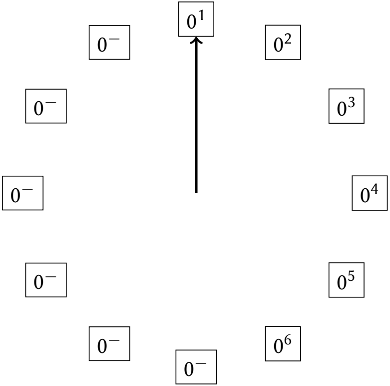
Example



Page References: 1, 2, 3, 4, 5, 6, 7, 8, 3, 4, 6, 1, 8, 16, 12, 34, 65, 3, 4, 7, 2, 65, 23, 2, 3, 5

GCLOCK

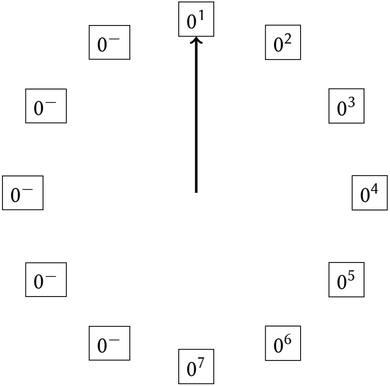
Example



Page References: 1, 2, 3, 4, 5, 6, 7, 8, 3, 4, 6, 1, 8, 16, 12, 34, 65, 3, 4, 7, 2, 65, 23, 2, 3, 5

GCLOCK

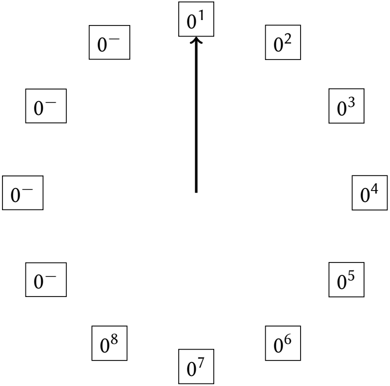
Example



Page References: 1, 2, 3, 4, 5, 6, 7, 8, 3, 4, 6, 1, 8, 16, 12, 34, 65, 3, 4, 7, 2, 65, 23, 2, 3, 5

GCLOCK

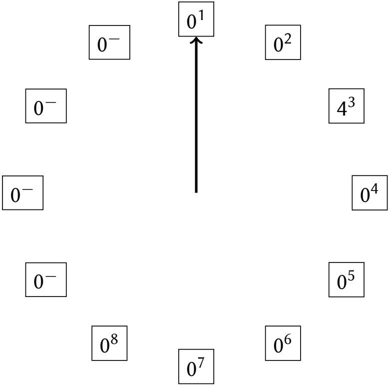
Example



Page References: 1, 2, 3, 4, 5, 6, 7, 8, 3, 4, 6, 1, 8, 16, 12, 34, 65, 3, 4, 7, 2, 65, 23, 2, 3, 5

GCLOCK

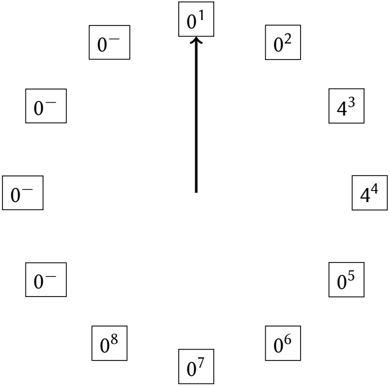
Example



Page References: 1, 2, 3, 4, 5, 6, 7, 8, 3, 4, 6, 1, 8, 16, 12, 34, 65, 3, 4, 7, 2, 65, 23, 2, 3, 5

GCLOCK

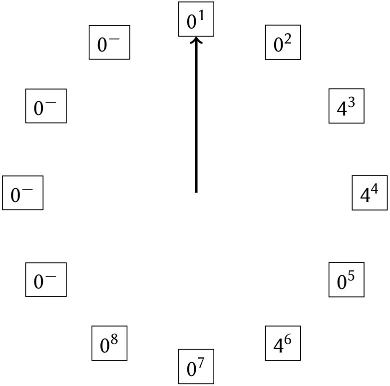
Example



Page References: 1, 2, 3, 4, 5, 6, 7, 8, 3, 4, 6, 1, 8, 16, 12, 34, 65, 3, 4, 7, 2, 65, 23, 2, 3, 5

GCLOCK

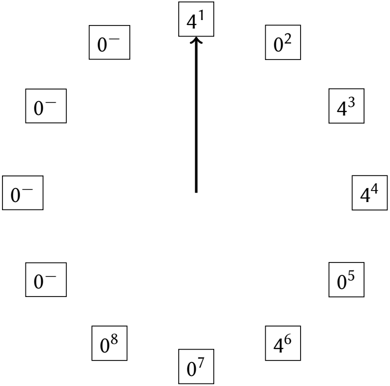
Example



Page References: 1, 2, 3, 4, 5, 6, 7, 8, 3, 4, 6, 1, 8, 16, 12, 34, 65, 3, 4, 7, 2, 65, 23, 2, 3, 5

GCLOCK

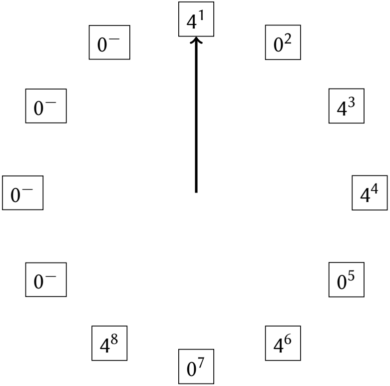
Example



Page References: 1, 2, 3, 4, 5, 6, 7, 8, 3, 4, 6, 1, 8, 16, 12, 34, 65, 3, 4, 7, 2, 65, 23, 2, 3, 5

GCLOCK

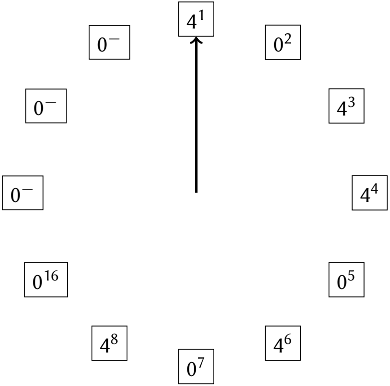
Example



Page References: 1, 2, 3, 4, 5, 6, 7, 8, 3, 4, 6, 1, 8, 16, 12, 34, 65, 3, 4, 7, 2, 65, 23, 2, 3, 5

GCLOCK

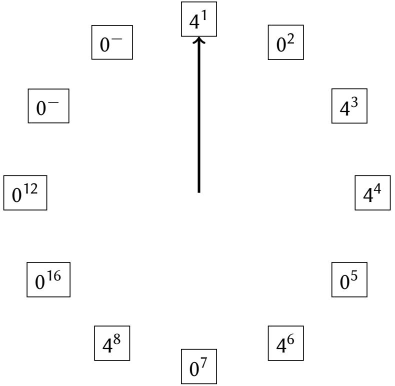
Example



Page References: 1, 2, 3, 4, 5, 6, 7, 8, 3, 4, 6, 1, 8, 16, 12, 34, 65, 3, 4, 7, 2, 65, 23, 2, 3, 5

GCLOCK

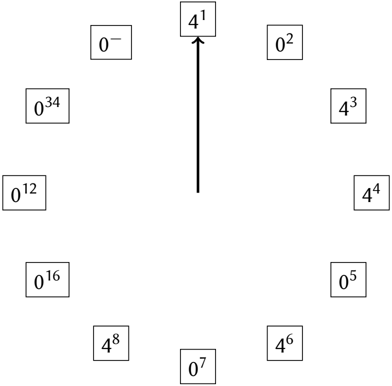
Example



Page References: 1, 2, 3, 4, 5, 6, 7, 8, 3, 4, 6, 1, 8, 16, 12, 34, 65, 3, 4, 7, 2, 65, 23, 2, 3, 5

GCLOCK

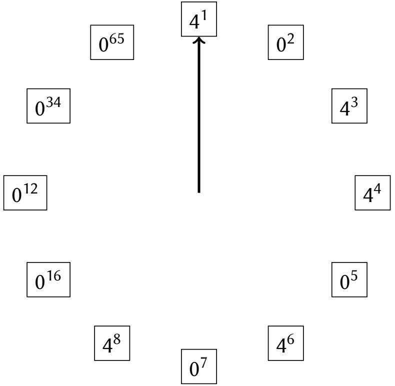
Example



Page References: 1, 2, 3, 4, 5, 6, 7, 8, 3, 4, 6, 1, 8, 16, 12, 34, 65, 3, 4, 7, 2, 65, 23, 2, 3, 5

GCLOCK

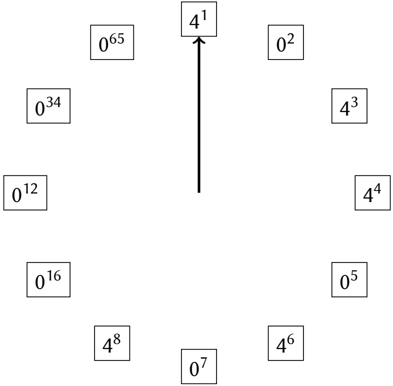
Example



Page References: 1, 2, 3, 4, 5, 6, 7, 8, 3, 4, 6, 1, 8, 16, 12, 34, 65, 3, 4, 7, 2, 65, 23, 2, 3, 5

GCLOCK

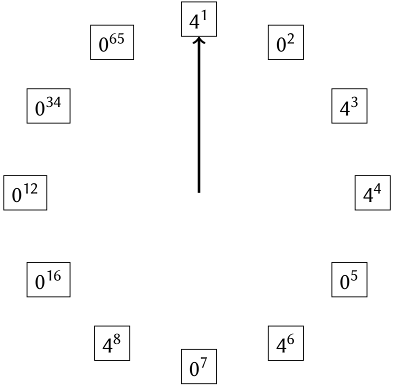
Example



Page References: 1, 2, 3, 4, 5, 6, 7, 8, 3, 4, 6, 1, 8, 16, 12, 34, 65, 3, 4, 7, 2, 65, 23, 2, 3, 5

GCLOCK

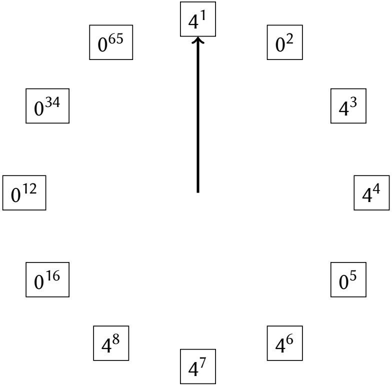
Example



Page References: 1, 2, 3, 4, 5, 6, 7, 8, 3, 4, 6, 1, 8, 16, 12, 34, 65, 3, 4, 7, 2, 65, 23, 2, 3, 5

GCLOCK

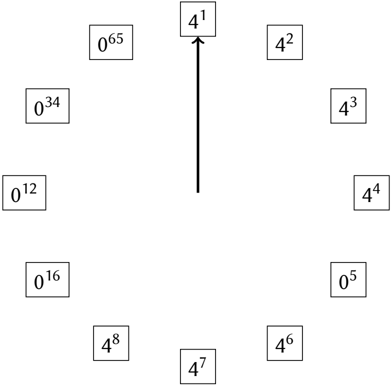
Example



Page References: 1, 2, 3, 4, 5, 6, 7, 8, 3, 4, 6, 1, 8, 16, 12, 34, 65, 3, 4, 7, 2, 65, 23, 2, 3, 5

GCLOCK

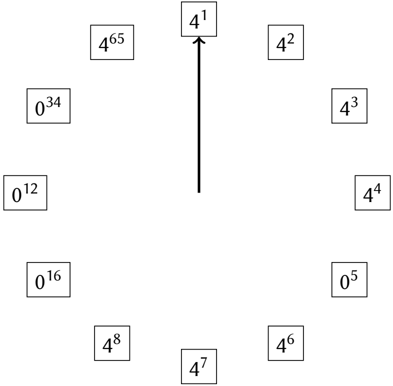
Example



Page References: 1, 2, 3, 4, 5, 6, 7, 8, 3, 4, 6, 1, 8, 16, 12, 34, 65, 3, 4, 7, 2, 65, 23, 2, 3, 5

GCLOCK

Example



Page References: 1, 2, 3, 4, 5, 6, 7, 8, 3, 4, 6, 1, 8, 16, 12, 34, 65, 3, 4, 7, 2, 65, 23, 2, 3, 5

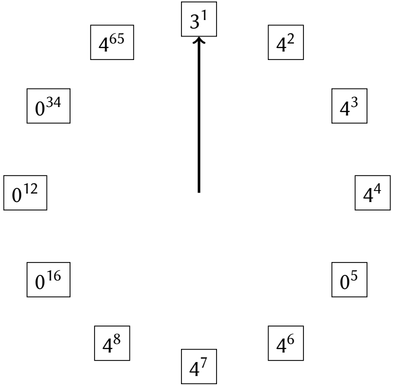
GCLOCK

Example

Evicting Pages!

GCLOCK

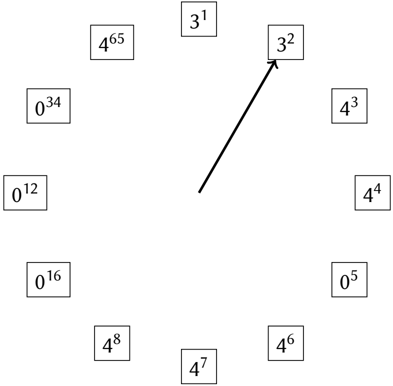
Example



Page References: 1, 2, 3, 4, 5, 6, 7, 8, 3, 4, 6, 1, 8, 16, 12, 34, 65, 3, 4, 7, 2, 65, 23, 2, 3, 5

GCLOCK

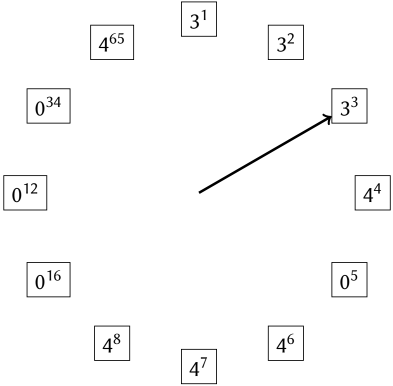
Example



Page References: 1, 2, 3, 4, 5, 6, 7, 8, 3, 4, 6, 1, 8, 16, 12, 34, 65, 3, 4, 7, 2, 65, 23, 2, 3, 5

GCLOCK

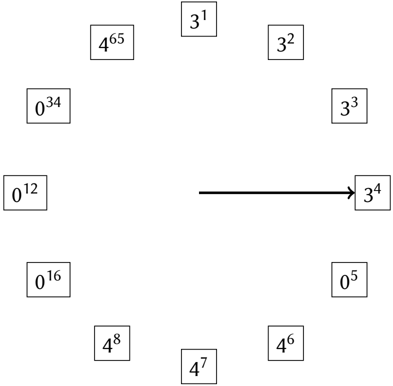
Example



Page References: 1, 2, 3, 4, 5, 6, 7, 8, 3, 4, 6, 1, 8, 16, 12, 34, 65, 3, 4, 7, 2, 65, 23, 2, 3, 5

GCLOCK

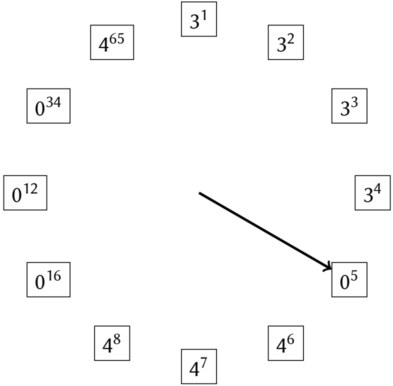
Example



Page References: 1, 2, 3, 4, 5, 6, 7, 8, 3, 4, 6, 1, 8, 16, 12, 34, 65, 3, 4, 7, 2, 65, 23, 2, 3, 5

GCLOCK

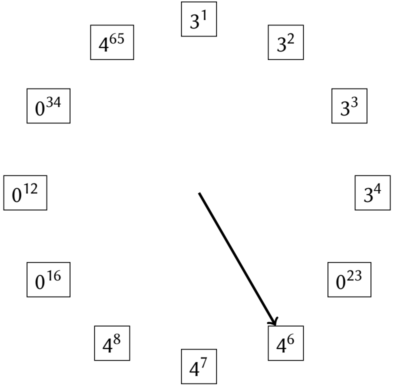
Example



Page References: 1, 2, 3, 4, 5, 6, 7, 8, 3, 4, 6, 1, 8, 16, 12, 34, 65, 3, 4, 7, 2, 65, 23, 2, 3, 5

GCLOCK

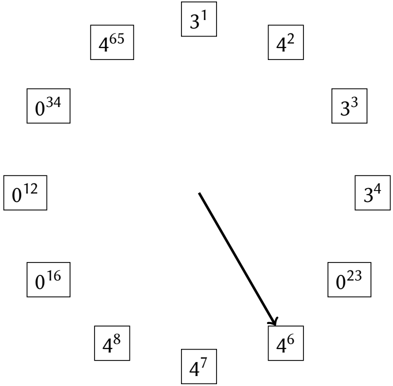
Example



Page References: 1, 2, 3, 4, 5, 6, 7, 8, 3, 4, 6, 1, 8, 16, 12, 34, 65, 3, 4, 7, 2, 65, 23, 2, 3, 5

GCLOCK

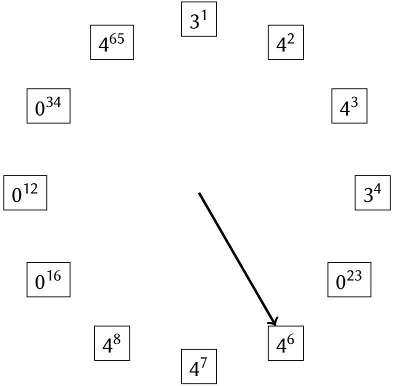
Example



Page References: 1, 2, 3, 4, 5, 6, 7, 8, 3, 4, 6, 1, 8, 16, 12, 34, 65, 3, 4, 7, 2, 65, 23, 2, 3, 5

GCLOCK

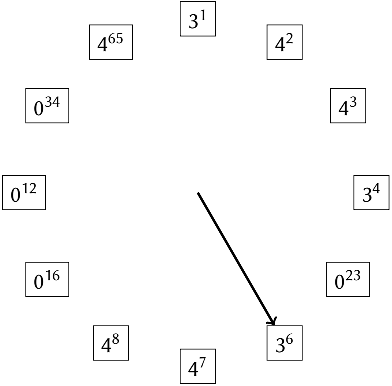
Example



Page References: 1, 2, 3, 4, 5, 6, 7, 8, 3, 4, 6, 1, 8, 16, 12, 34, 65, 3, 4, 7, 2, 65, 23, 2, 3, 5

GCLOCK

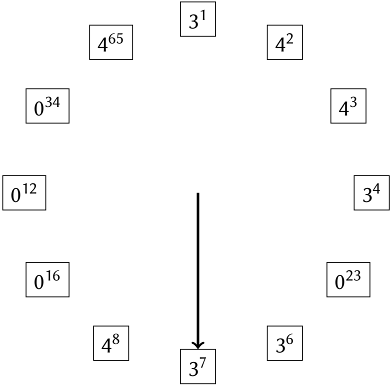
Example



Page References: 1, 2, 3, 4, 5, 6, 7, 8, 3, 4, 6, 1, 8, 16, 12, 34, 65, 3, 4, 7, 2, 65, 23, 2, 3, 5

GCLOCK

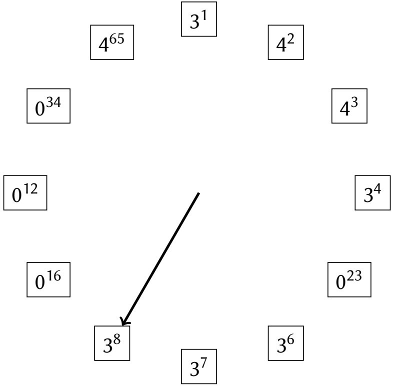
Example



Page References: 1, 2, 3, 4, 5, 6, 7, 8, 3, 4, 6, 1, 8, 16, 12, 34, 65, 3, 4, 7, 2, 65, 23, 2, 3, 5

GCLOCK

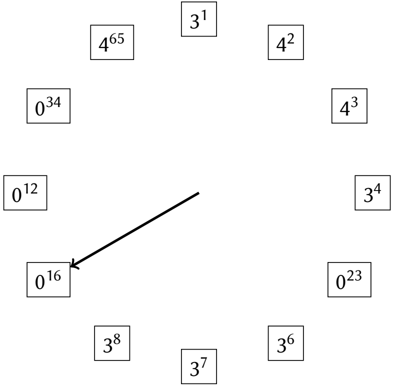
Example



Page References: 1, 2, 3, 4, 5, 6, 7, 8, 3, 4, 6, 1, 8, 16, 12, 34, 65, 3, 4, 7, 2, 65, 23, 2, 3, 5

GCLOCK

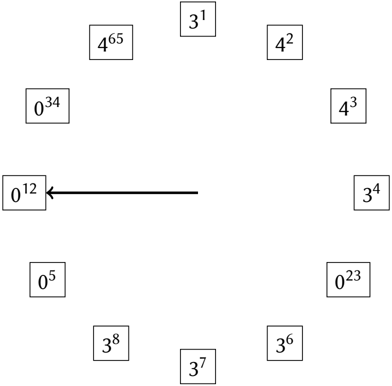
Example



Page References: 1, 2, 3, 4, 5, 6, 7, 8, 3, 4, 6, 1, 8, 16, 12, 34, 65, 3, 4, 7, 2, 65, 23, 2, 3, 5

GCLOCK

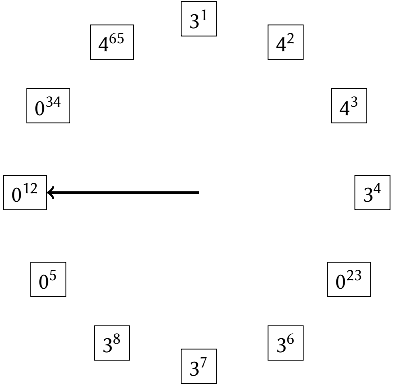
Example



Page References: 1, 2, 3, 4, 5, 6, 7, 8, 3, 4, 6, 1, 8, 16, 12, 34, 65, 3, 4, 7, 2, 65, 23, 2, 3, 5

GCLOCK

Example



Page References: 1, 2, 3, 4, 5, 6, 7, 8, 3, 4, 6, 1, 8, 16, 12, 34, 65, 3, 4, 7, 2, 65, 23, 2, 3, 1

GCLOCK

Algorithm (1)

GCLOCK

Algorithm (1)

```
procedure GET_PAGE(x)

end procedure
```

GCLOCK

Algorithm (1)

procedure GET_PAGE(x)
    if $x \in$ buffer pool **then**

    end if
end procedure

GCLOCK

Algorithm (1)

procedure GET_PAGE(x)
  if $x \in$ buffer pool **then**
    $referenced[x] \leftarrow k$

  end if
end procedure

GCLOCK

Algorithm (1)

procedure GET_PAGE(x)
  if $x \in$ buffer pool **then**
    $referenced\,[x] \leftarrow k$
  else if buffer pool is full **then**

  end if
end procedure

GCLOCK

Algorithm (1)

procedure GET_PAGE(x)
    if $x \in$ buffer pool **then**
        $referenced[x] \leftarrow k$
    else if buffer pool is full **then**
        EVICT
        INSERT(x)
        $referenced[x] \leftarrow 0$
    end if
end procedure

GCLOCK

Algorithm (1)

```
procedure GET_PAGE(x)
    if x ∈ buffer pool then
        referenced[x] ← k
    else if buffer pool is full then
        EVICT
        INSERT(x)
        referenced[x] ← 0
    else


    end if
end procedure
```

GCLOCK

Algorithm (1)

```
procedure GET_PAGE(x)
    if x ∈ buffer pool then
        referenced[x] ← k
    else if buffer pool is full then
        EVICT
        INSERT(x)
        referenced[x] ← 0
    else
        INSERT(x)
        referenced[x] ← 0
    end if
end procedure
```

GCLOCK

Algorithm (2)

GCLOCK

Algorithm (2)

```
procedure EVICT

end procedure
```

GCLOCK

Algorithm (2)

```
procedure EVICT
    found ← false
    while found ≠ true do



    end while
end procedure
```

GCLOCK

Algorithm (2)

```
procedure EVICT
    found ← false
    while found ≠ true do
        x ←GET_NEXT



    end while
end procedure
```

GCLOCK

Algorithm (2)

procedure EVICT
    found $\leftarrow$ false
    while *found* $\neq$ true **do**
        $x \leftarrow$GET_NEXT
        if *referenced* $[x] = 0$ **then**

        end if
    end while
end procedure

GCLOCK

Algorithm (2)

procedure EVICT
    found $\leftarrow$ false
    while *found* $\neq$ true **do**
        $x \leftarrow$ GET_NEXT
        if *referenced* $[x] = 0$ **then**
            found $\leftarrow$ true
            REMOVE_NEXT

        end if
    end while
end procedure

GCLOCK

Algorithm (2)

```
procedure EVICT
    found ← false
    while found ≠ true do
        x ← GET_NEXT
        if referenced[x] = 0 then
            found ← true
            REMOVE_NEXT
        else


        end if
    end while
end procedure
```

GCLOCK

Algorithm (2)

```
procedure EVICT
    found ← false
    while found ≠ true do
        x ← GET_NEXT
        if referenced[x] = 0 then
            found ← true
            REMOVE_NEXT
        else
            referenced[x] ← referenced[x] − 1
            MOVE_HAND
        end if
    end while
end procedure
```

GCLOCK

Advantage of Higher *k*-Values

GCLOCK

Advantage of Higher *k*-Values

Advantages of Lower *k*-Values

GCLOCK

Advantage of Higher k-Values

- More detailed statistics about page references
  $\Longrightarrow$ Higher hit rate
  $\Longrightarrow$ Higher performance

Advantages of Lower k-Values

GCLOCK

Advantage of Higher *k*-Values

- More detailed statistics about page references
  $\Longrightarrow$ Higher hit rate
  $\Longrightarrow$ Higher performance

Advantages of Lower *k*-Values

- Lower processing time required to find an eviction victim
  $\Longrightarrow$ Higher performance

GCLOCK

Advantage of Higher *k*-Values

- More detailed statistics about page references
  $\Longrightarrow$ Higher hit rate
  $\Longrightarrow$ Higher performance

Advantages of Lower *k*-Values

- Lower processing time required to find an eviction victim
  $\Longrightarrow$ Higher performance
- Lower memory overhead due to shorter referenced-numbers

GCLOCK

Advantage of Higher *k*-Values

- More detailed statistics about page references
  $\Longrightarrow$ Higher hit rate
  $\Longrightarrow$ Higher performance

Advantages of Lower *k*-Values

- Lower processing time required to find an eviction victim
  $\Longrightarrow$ Higher performance
- Lower memory overhead due to shorter referenced-numbers

$\Longrightarrow$ Trade-off between CPU- and I/O-optimization

CAR

Overview

CAR

Overview

- Extensive enhancement of the CLOCK algorithm: *Clock with Adaptive Replacement* [BM04]

CAR

Overview

- Extensive enhancement of the CLOCK algorithm: *Clock with Adaptive Replacement* [BM04]
- Approximation of the ARC page eviction strategy

CAR

Overview

- Extensive enhancement of the CLOCK algorithm: *Clock with Adaptive Replacement* [BM04]
- Approximation of the ARC page eviction strategy
- Uses two clocks and two LRU-lists

CAR

Overview

- Extensive enhancement of the CLOCK algorithm: *Clock with Adaptive Replacement* [BM04]
- Approximation of the ARC page eviction strategy
- Uses two clocks and two LRU-lists
- Advantages of CAR compared to CLOCK:

CAR

Overview

- Extensive enhancement of the CLOCK algorithm: *Clock with Adaptive Replacement* [BM04]
- Approximation of the ARC page eviction strategy
- Uses two clocks and two LRU-lists
- Advantages of CAR compared to CLOCK:
  - Weighted consideration of reference recency and frequency

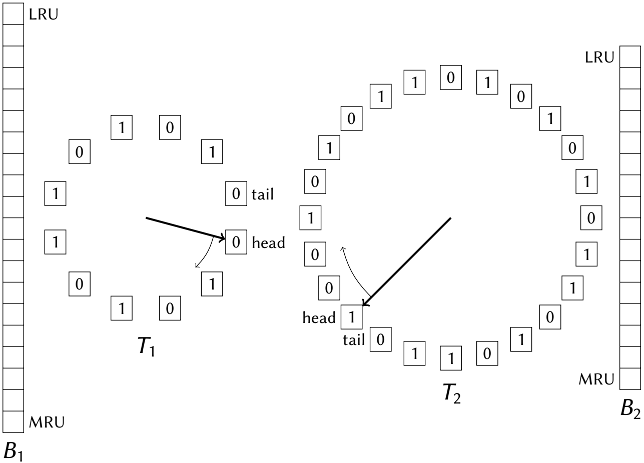
CAR

Overview

- Extensive enhancement of the CLOCK algorithm: *Clock with Adaptive Replacement* [BM04]
- Approximation of the ARC page eviction strategy
- Uses two clocks and two LRU-lists
- Advantages of CAR compared to CLOCK:
  - Weighted consideration of reference recency and frequency
  - Scan-resistence

CAR

Example



CAR

Algorithm (1)

CAR

Algorithm (1)

1: **procedure** GET_PAGE(x)

CAR

Algorithm (1)

1: **procedure** GET_PAGE(x)
2: **if** $x \in T_1 \cup T_2$ **then**

CAR

Algorithm (1)

1: **procedure** GET_PAGE(x)
2: **if** $x \in T_1 \cup T_2$ **then**
3: $referenced[x] \leftarrow \text{true}$

CAR

Algorithm (1)

procedure GET_PAGE(x)
  if $x \in T_1 \cup T_2$ **then**
    referenced $[x] \leftarrow$ true
  else

CAR

Algorithm (1)

procedure GET_PAGE(x)
    if $x \in T_1 \cup T_2$ **then**
        $\mathit{referenced}[x] \leftarrow \text{true}$
    else
        if $|T_1| + |T_2| = c$ **then**

        end if

CAR

Algorithm (1)

procedure GET_PAGE(x)
  if $x \in T_1 \cup T_2$ **then**
    $referenced[x] \leftarrow \text{true}$
  else
    if $|T_1| + |T_2| = c$ **then**
      EVICT

    end if

CAR

Algorithm (1)

procedure GET_PAGE(x)
    if $x \in T_1 \cup T_2$ **then**
        $\mathit{referenced}[x] \leftarrow \text{true}$
    else
        if $|T_1| + |T_2| = c$ **then**
            EVICT
            if $(x \notin B_1 \cup B_2) \wedge (|T_1| + |B_1| = c)$ **then**

            end if
        end if

CAR

Algorithm (1)

procedure GET_PAGE(x)
    if $x \in T_1 \cup T_2$ **then**
        referenced $[x] \leftarrow$ true
    else
        if $|T_1| + |T_2| = c$ **then**
            EVICT
            if $(x \notin B_1 \cup B_2) \wedge (|T_1| + |B_1| = c)$ **then**
                REMOVE_NEXT(B_1)

            end if
        end if

CAR

Algorithm (1)

procedure GET_PAGE(x)
    if $x \in T_1 \cup T_2$ **then**
        $referenced[x] \leftarrow \text{true}$
    else
        if $|T_1| + |T_2| = c$ **then**
            EVICT
            if $(x \notin B_1 \cup B_2) \wedge (|T_1| + |B_1| = c)$ **then**
                REMOVE_NEXT(B_1)
            else if $(x \notin B_1 \cup B_2) \wedge (|T_1| + |T_2| + |B_1| + |B_2| = 2c)$ **then**

            end if
        end if

CAR

Algorithm (1)

procedure GET_PAGE(x)
    if $x \in T_1 \cup T_2$ **then**
        referenced $[x] \leftarrow$ true
    else
        if $|T_1| + |T_2| = c$ **then**
            EVICT
            if $(x \notin B_1 \cup B_2) \wedge (|T_1| + |B_1| = c)$ **then**
                REMOVE_NEXT(B_1)
            else if $(x \notin B_1 \cup B_2) \wedge (|T_1| + |T_2| + |B_1| + |B_2| = 2c)$ **then**
                REMOVE_NEXT(B_2)
            end if
        end if

CAR

Algorithm (2)

```
    end if
end procedure
```

CAR

Algorithm (2)

if $x \notin B_1 \cup B_2$ **then**

end if
end if
end procedure

CAR

Algorithm (2)

if $x \notin B_1 \cup B_2$ **then**
INSERT_INTO(T_1, x)
referenced $[x] \leftarrow$ false

end if
end if
end procedure

CAR

Algorithm (2)

if $x \notin B_1 \cup B_2$ **then**
INSERT_INTO(T_1, x)
referenced $[x] \leftarrow$ false
else if $x \in B_1$ **then**

end if
end if
end procedure

CAR

Algorithm (2)

```
    if x ∉ B_1 ∪ B_2 then
        INSERT_INTO(T_1, x)
        referenced[x] ← false
    else if x ∈ B_1 then
        p ← min{p + max{1, |B_2|/|B_1|}, c}

    end if
  end if
end procedure
```

CAR

Algorithm (2)

if $x \notin B_1 \cup B_2$ **then**
  $\textsc{insert\_into}(T_1, x)$
  $\textit{referenced}\left[x\right] \leftarrow \text{false}$
else if $x \in B_1$ **then**
  $p \leftarrow \min\left\{p + \max\left\{1, \frac{|B_2|}{|B_1|}\right\}, c\right\}$
  $\textsc{insert\_into}(T_2, x)$
  $\textit{referenced}\left[x\right] \leftarrow \text{false}$

end if
end if
end procedure

CAR

Algorithm (2)

if $x \notin B_1 \cup B_2$ **then**
    $\textsc{insert\_into}(T_1, x)$
    $\textit{referenced}\left[x\right] \leftarrow \text{false}$
else if $x \in B_1$ **then**
    $p \leftarrow \min\left\{p + \max\left\{1, \frac{|B_2|}{|B_1|}\right\}, c\right\}$
    $\textsc{insert\_into}(T_2, x)$
    $\textit{referenced}\left[x\right] \leftarrow \text{false}$
else

end if
end if
end procedure

CAR

Algorithm (2)

if $x \notin B_1 \cup B_2$ **then**
  INSERT_INTO(T_1, x)
  $\textit{referenced}\,[x] \leftarrow \text{false}$
else if $x \in B_1$ **then**
  $p \leftarrow \min\left\{p + \max\left\{1, \frac{|B_2|}{|B_1|}\right\}, c\right\}$
  INSERT_INTO(T_2, x)
  $\textit{referenced}\,[x] \leftarrow \text{false}$
else
  $p \leftarrow \max\left\{p - \max\left\{1, \frac{|B_1|}{|B_2|}\right\}, 0\right\}$

end if
end if
end procedure

CAR

Algorithm (2)

if $x \notin B_1 \cup B_2$ **then**
INSERT_INTO(T_1, x)
referenced $[x] \leftarrow$ false
else if $x \in B_1$ **then**
$p \leftarrow \min\left\{p + \max\left\{1, \frac{|B_2|}{|B_1|}\right\}, c\right\}$
INSERT_INTO(T_2, x)
referenced $[x] \leftarrow$ false
else
$p \leftarrow \max\left\{p - \max\left\{1, \frac{|B_1|}{|B_2|}\right\}, 0\right\}$
INSERT_INTO(T_2, x)
referenced $[x] \leftarrow$ false
end if
end if
end procedure

CAR

Algorithm (3)

CAR

Algorithm (3)

1: **procedure** EVICT

CAR

Algorithm (3)

procedure EVICT
    found $\leftarrow$ false
    while *found* $\neq$ true **do**

CAR

Algorithm (3)

procedure EVICT
    found $\leftarrow$ false
    while *found* $\neq$ true **do**
        if $|T_1| \geq \max\{1, p\}$ **then**

CAR

Algorithm (3)

procedure EVICT
  found $\leftarrow$ false
  while *found* $\neq$ true **do**
    if $|T_1| \geq \max\{1, p\}$ **then**
      $x \leftarrow$GET_NEXT_FROM(T_1)

CAR

Algorithm (3)

procedure EVICT
    found $\leftarrow$ false
    while *found* $\neq$ true **do**
        if $|T_1| \geq \max\{1, p\}$ **then**
            $x \leftarrow$ GET_NEXT_FROM(T_1)
            if $\textit{referenced}[x] = \text{false}$ **then**

            end if

CAR

Algorithm (3)

procedure EVICT
    found $\leftarrow$ false
    while *found* $\neq$ true **do**
        if $|T_1| \geq \max\{1, p\}$ **then**
            $x \leftarrow$ GET_NEXT_FROM(T_1)
            if *referenced* $[x]$ = false **then**
                found $\leftarrow$ true
                REMOVE_NEXT(T_1)
                INSERT_INTO(B_1, x)

            end if

CAR

Algorithm (3)

```
procedure EVICT
    found ← false
    while found ≠ true do
        if |T_1| ≥ max{1, p} then
            x ← GET_NEXT_FROM(T_1)
            if referenced[x] = false then
                found ← true
                REMOVE_NEXT(T_1)
                INSERT_INTO(B_1, x)
            else


            end if
```

CAR

Algorithm (3)

procedure EVICT
    found $\leftarrow$ false
    while *found* $\neq$ true **do**
        if $|T_1| \geq \max\{1, p\}$ **then**
            $x \leftarrow$ GET_NEXT_FROM(T_1)
            if *referenced* $[x]$ = false **then**
                found $\leftarrow$ true
                REMOVE_NEXT(T_1)
                INSERT_INTO(B_1, x)
            else
                referenced $[x] \leftarrow$ false

            end if

CAR

Algorithm (3)

```
procedure EVICT
    found ← false
    while found ≠ true do
        if |T_1| ≥ max{1, p} then
            x ← GET_NEXT_FROM(T_1)
            if referenced[x] = false then
                found ← true
                REMOVE_NEXT(T_1)
                INSERT_INTO(B_1, x)
            else
                referenced[x] ← false
                MOVE_HAND(T_1)
            end if
```

CAR

Algorithm (4)

```
        end if
    end while
end procedure
```

CAR

Algorithm (4)

```
        else
        end if
    end while
end procedure
```

CAR

Algorithm (4)

    else
        $x \leftarrow \textsc{get\_next\_from}(T_2)$

    end if
  end while
end procedure

CAR

Algorithm (4)

else
  $x \leftarrow$ GET_NEXT_FROM(T_2)
  if $\textit{referenced}[x] = \text{false}$ **then**

  end if
end if
end while
end procedure

CAR

Algorithm (4)

```
        else
            x ←GET_NEXT_FROM(T_2)
            if referenced[x] = false then
                found ← true
                REMOVE_NEXT(T_2)
                INSERT_INTO(B_2, x)



            end if
        end if
    end while
end procedure
```

CAR

Algorithm (4)

```
        else
            x ← GET_NEXT_FROM(T_2)
            if referenced[x] = false then
                found ← true
                REMOVE_NEXT(T_2)
                INSERT_INTO(B_2, x)
            else


            end if
        end if
    end while
end procedure
```

CAR

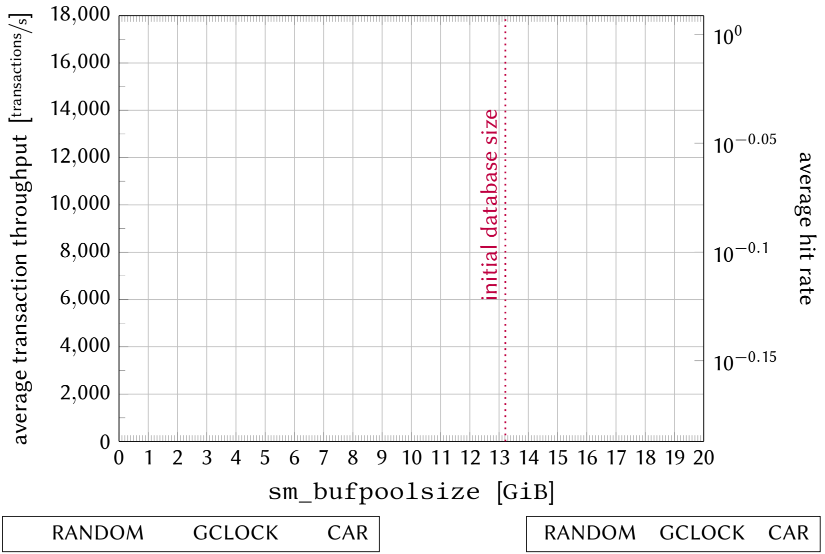
Algorithm (4)

```
        else
            x ← GET_NEXT_FROM(T_2)
            if referenced[x] = false then
                found ← true
                REMOVE_NEXT(T_2)
                INSERT_INTO(B_2, x)
            else
                referenced[x] ← false
                MOVE_HAND(T_2)
            end if
        end if
    end while
end procedure
```

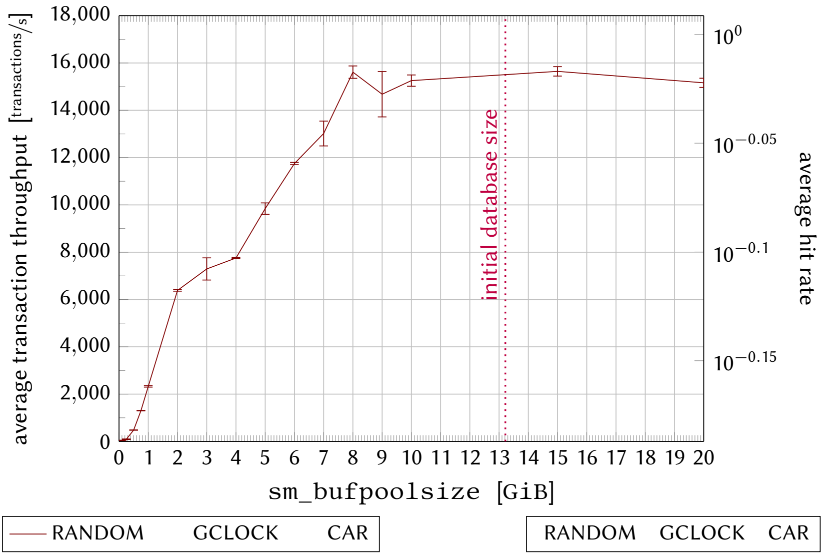
Subsection 3

Performance Evaluation

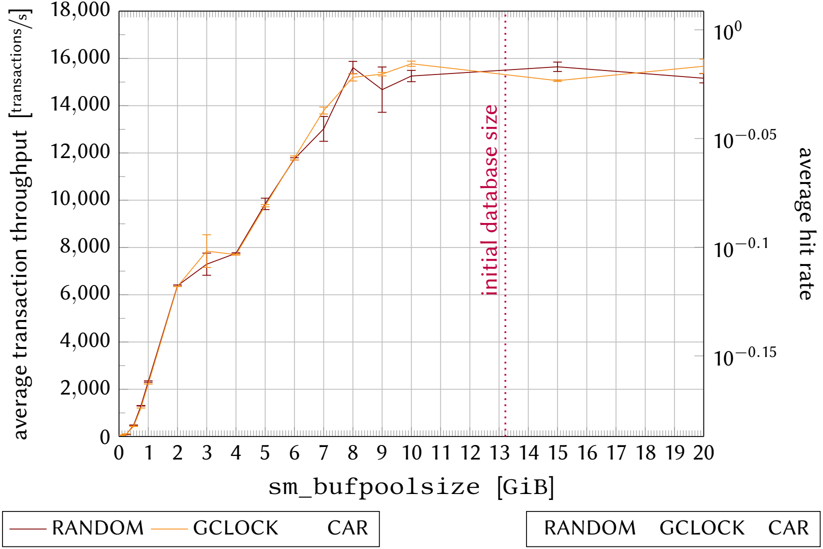
Buffer Pool Without Pointer Swizzling (TPC-C)



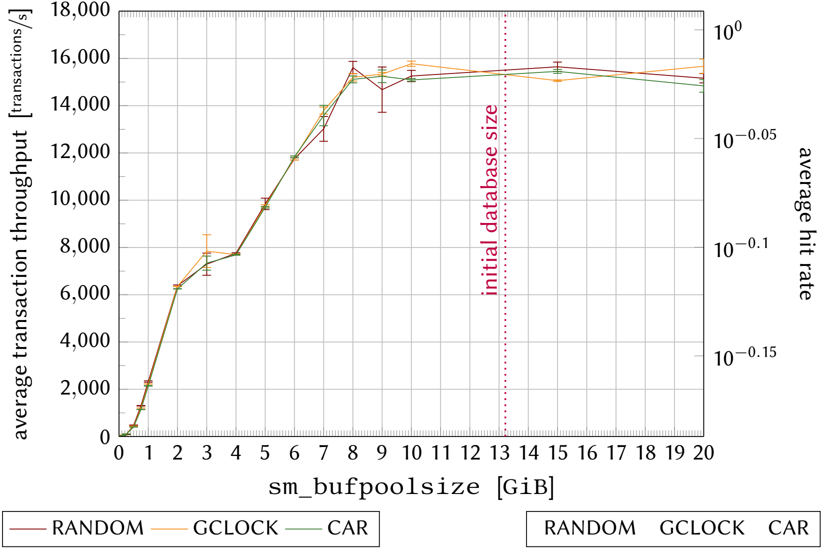
Buffer Pool Without Pointer Swizzling (TPC-C)



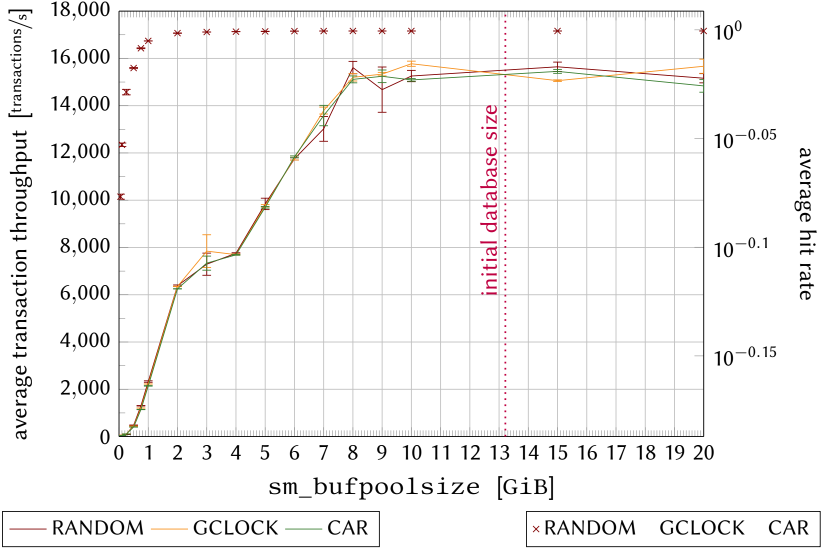
Buffer Pool Without Pointer Swizzling (TPC-C)



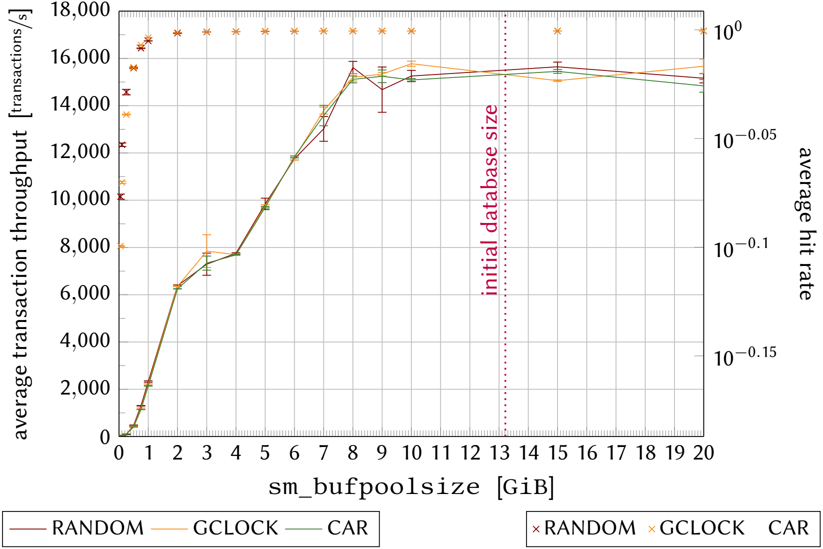
Buffer Pool Without Pointer Swizzling (TPC-C)



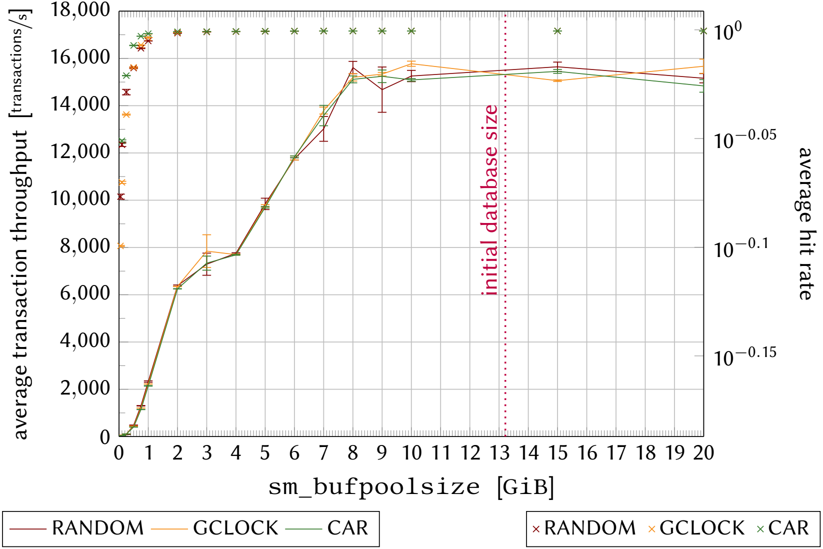
Buffer Pool Without Pointer Swizzling (TPC-C)



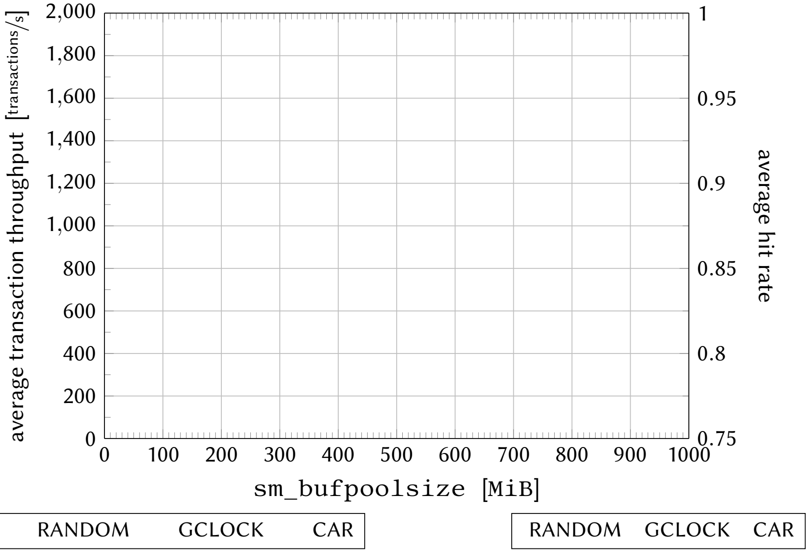
Buffer Pool Without Pointer Swizzling (TPC-C)



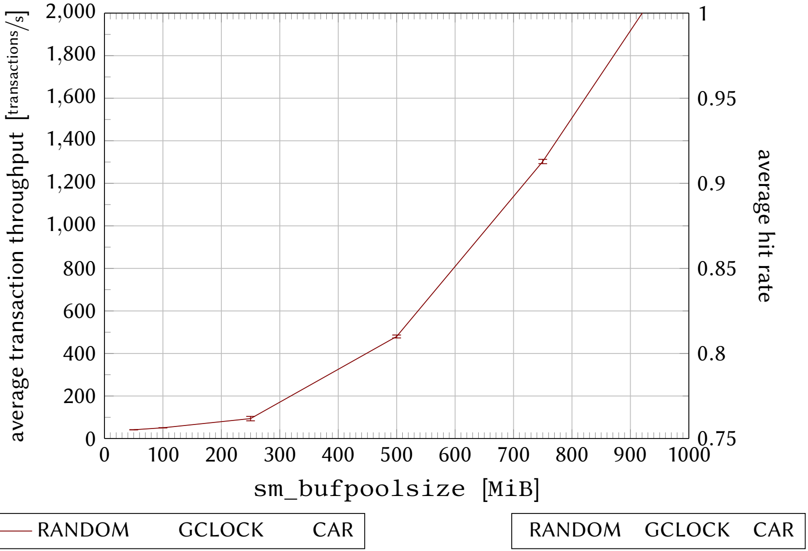
Buffer Pool Without Pointer Swizzling (TPC-C)



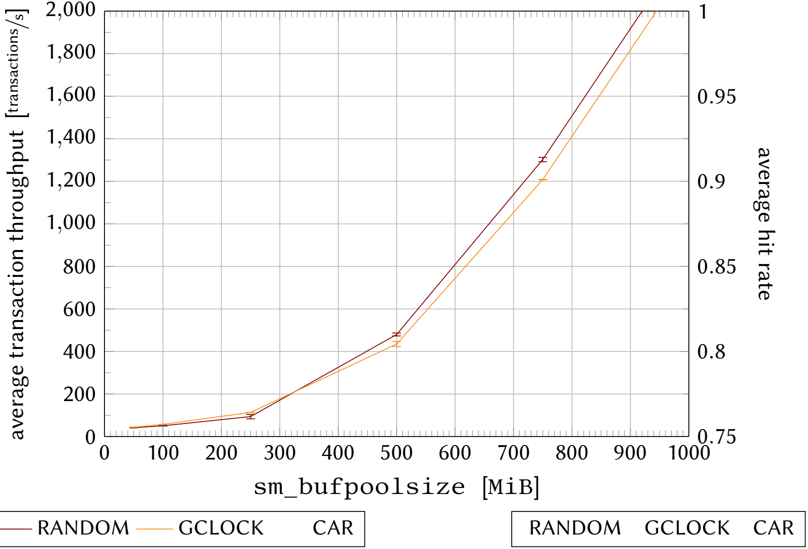
Buffer Pool Without Pointer Swizzling (≤1 GiB, TPC-C)



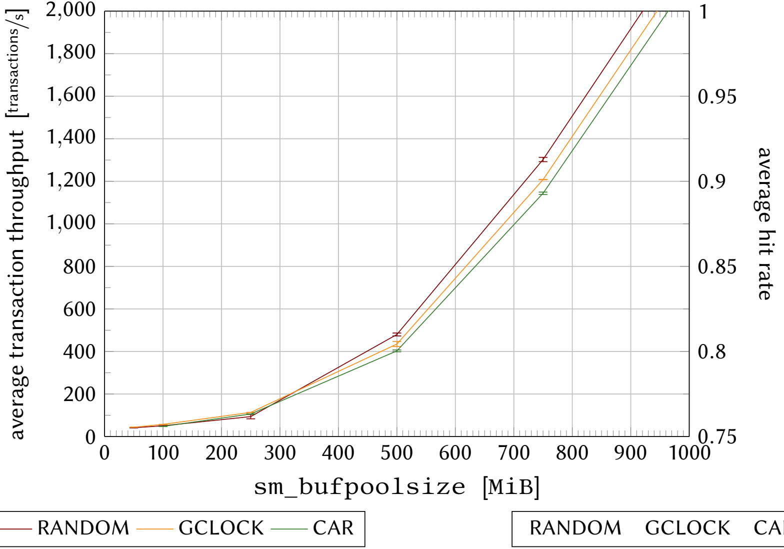
Buffer Pool Without Pointer Swizzling (≤1 GiB, TPC-C)



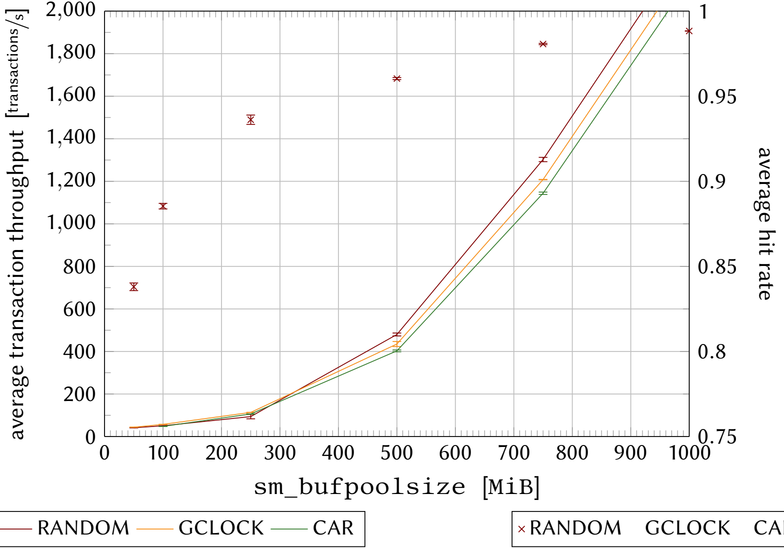
Buffer Pool Without Pointer Swizzling (≤1 GiB, TPC-C)



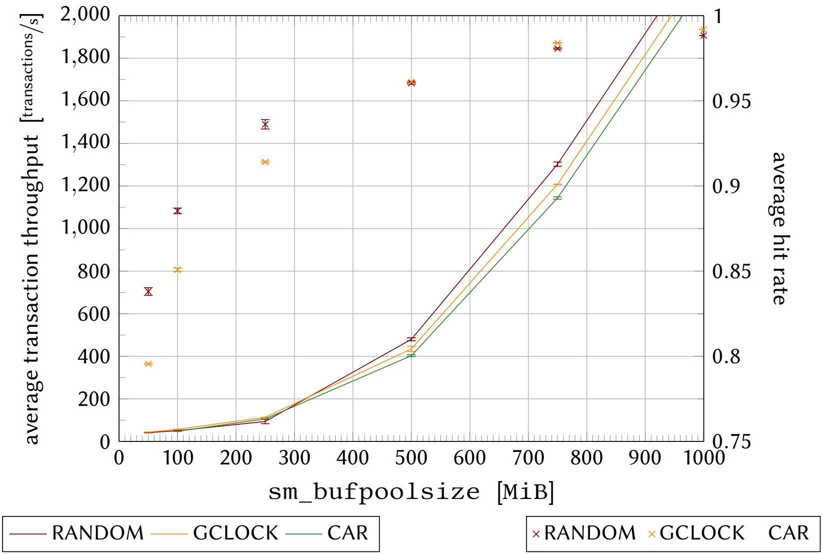
Buffer Pool Without Pointer Swizzling (≤1 GiB, TPC-C)



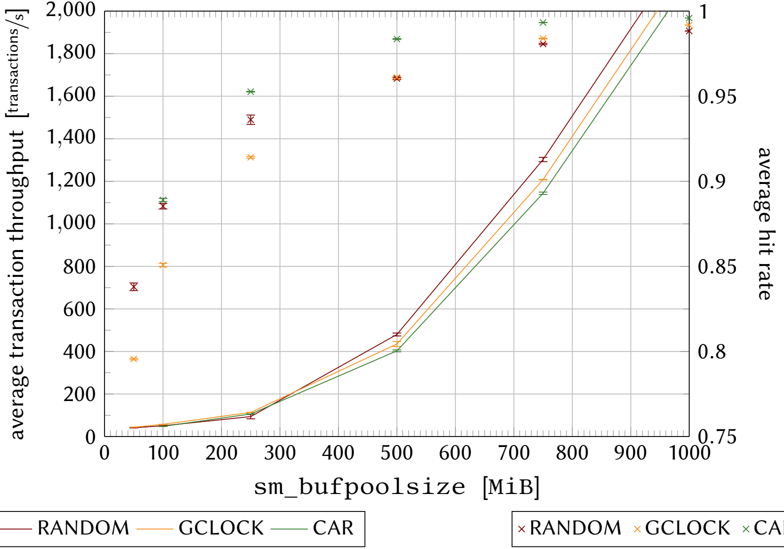
Buffer Pool Without Pointer Swizzling (≤1 GiB, TPC-C)



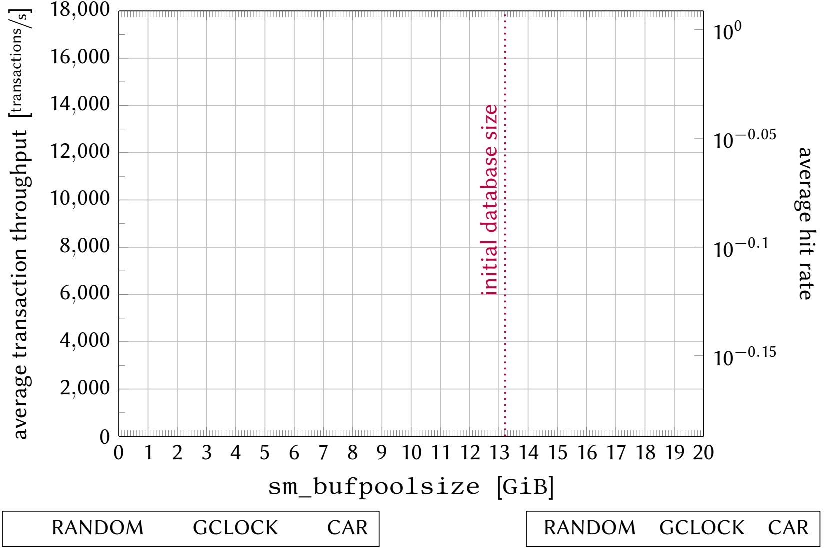
Buffer Pool Without Pointer Swizzling (≤1 GiB, TPC-C)



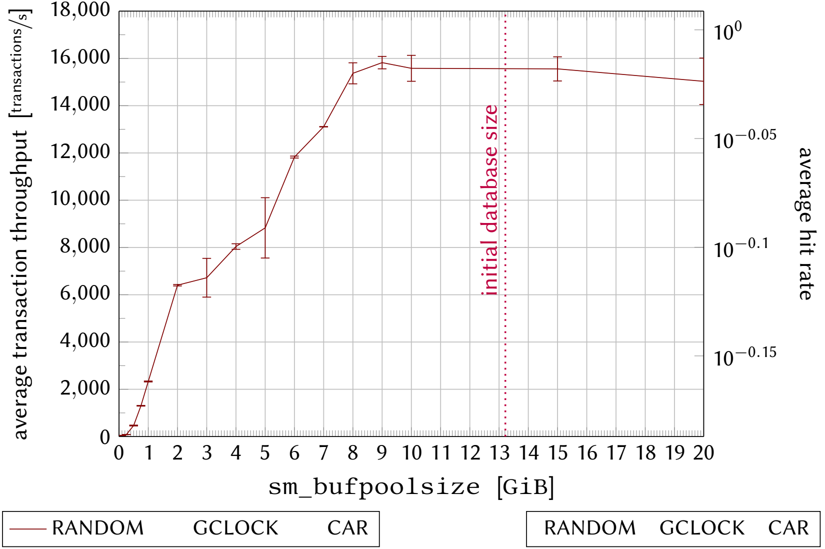
Buffer Pool Without Pointer Swizzling (≤1 GiB, TPC-C)



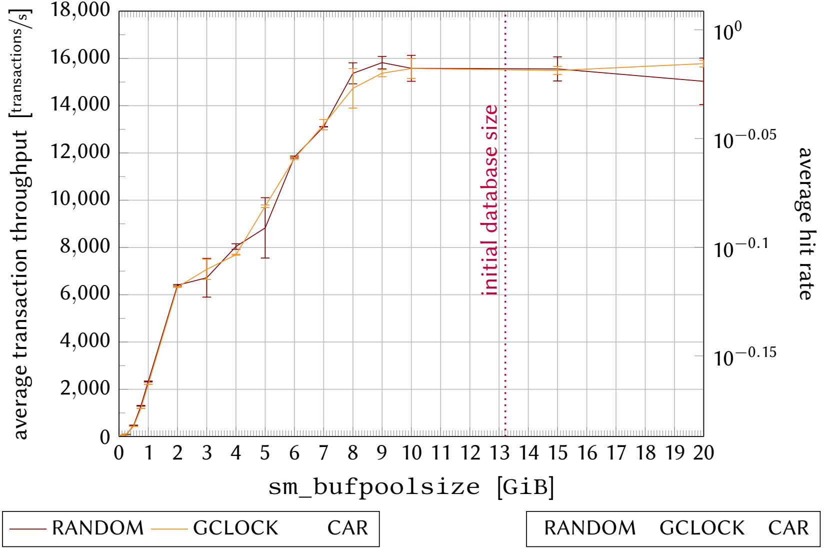
Buffer Pool With Pointer Swizzling (TPC-C)



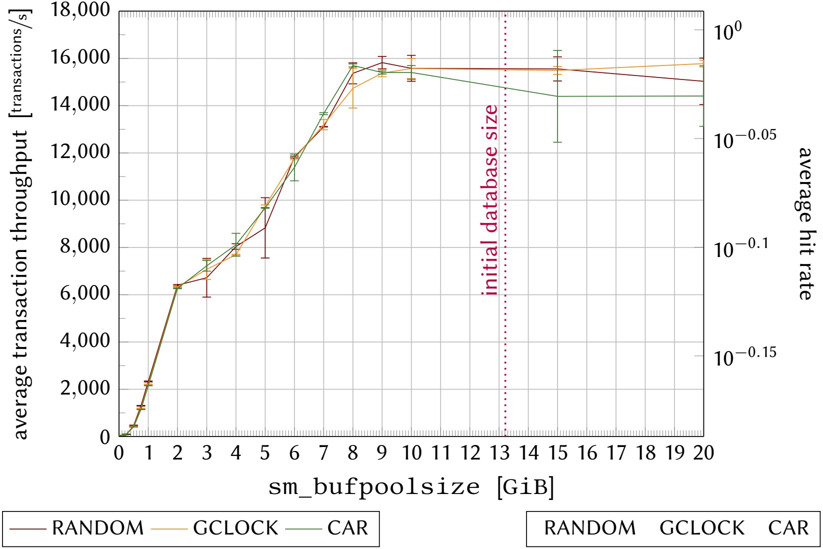
Buffer Pool With Pointer Swizzling (TPC-C)



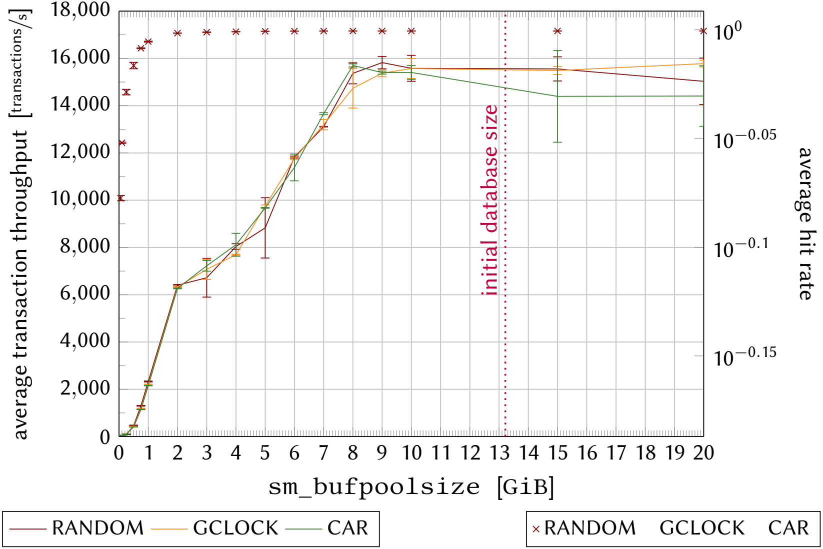
Buffer Pool With Pointer Swizzling (TPC-C)



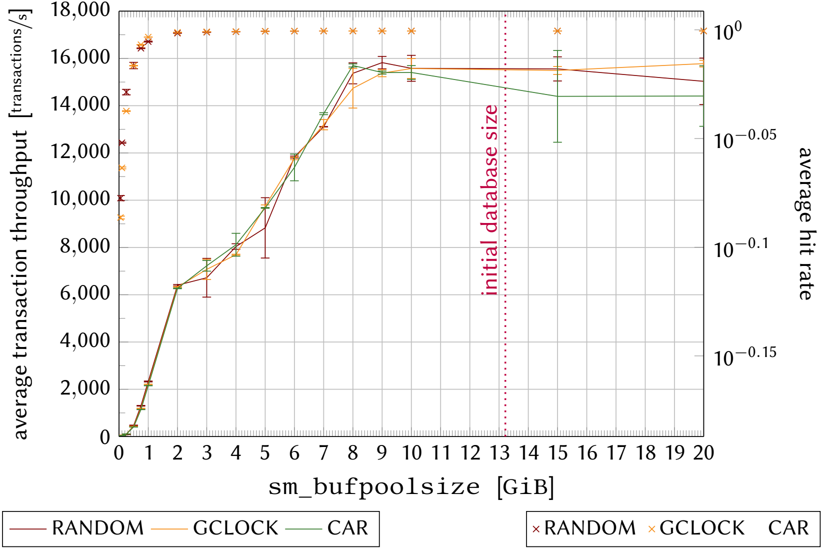
Buffer Pool With Pointer Swizzling (TPC-C)



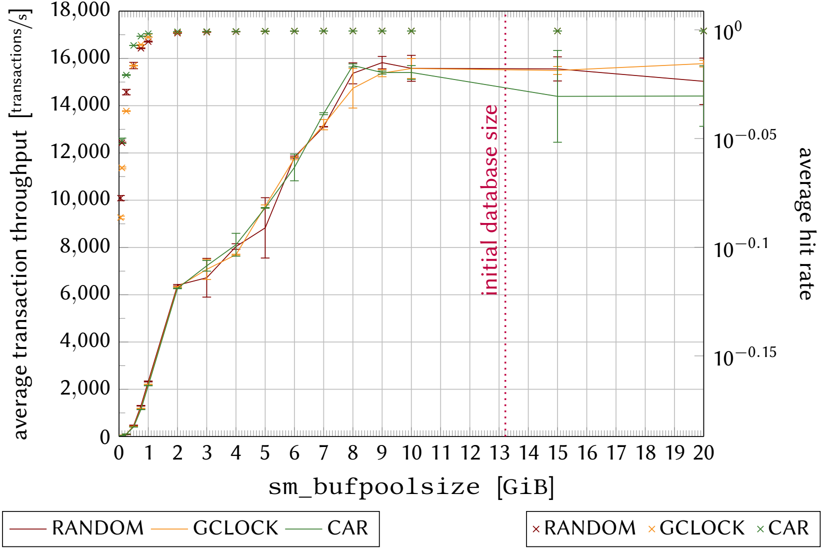
Buffer Pool With Pointer Swizzling (TPC-C)



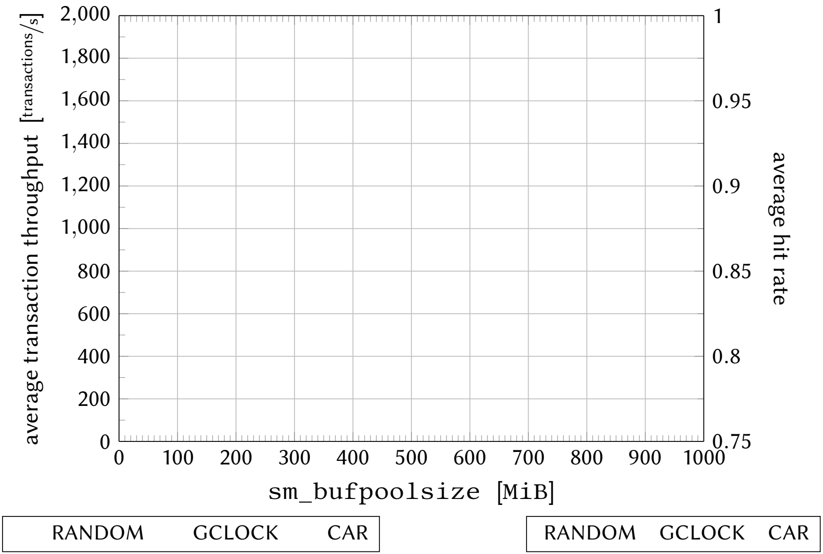
Buffer Pool With Pointer Swizzling (TPC-C)



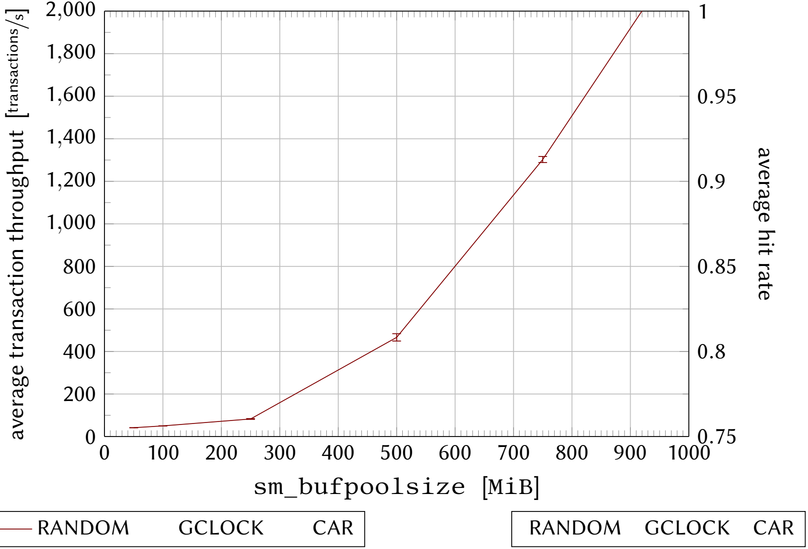
Buffer Pool With Pointer Swizzling (TPC-C)



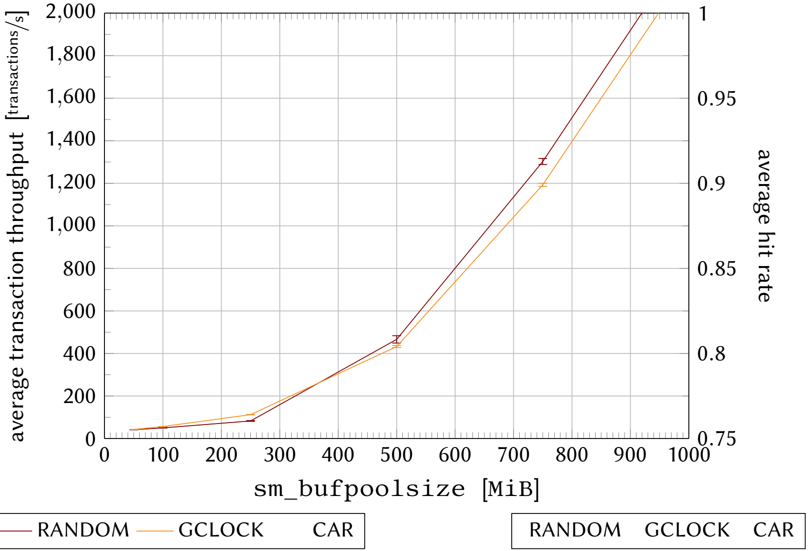
Buffer Pool With Pointer Swizzling (≤1 GiB, TPC-C)



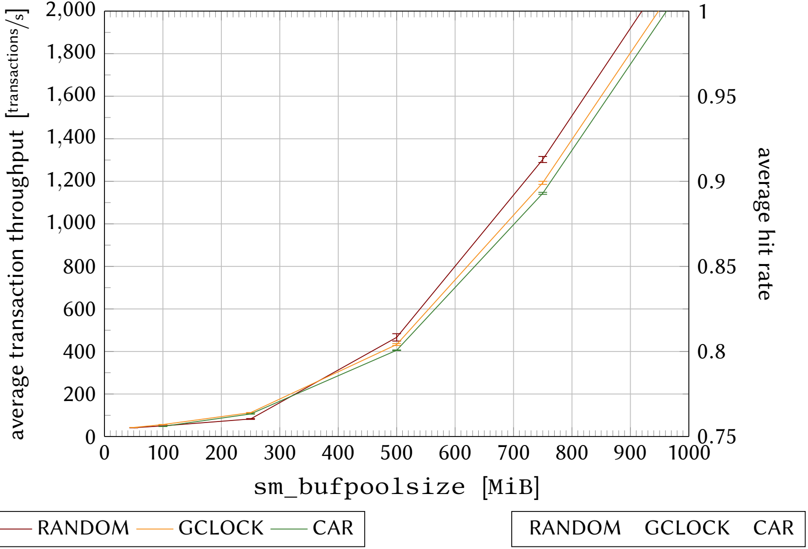
Buffer Pool With Pointer Swizzling (≤1 GiB, TPC-C)



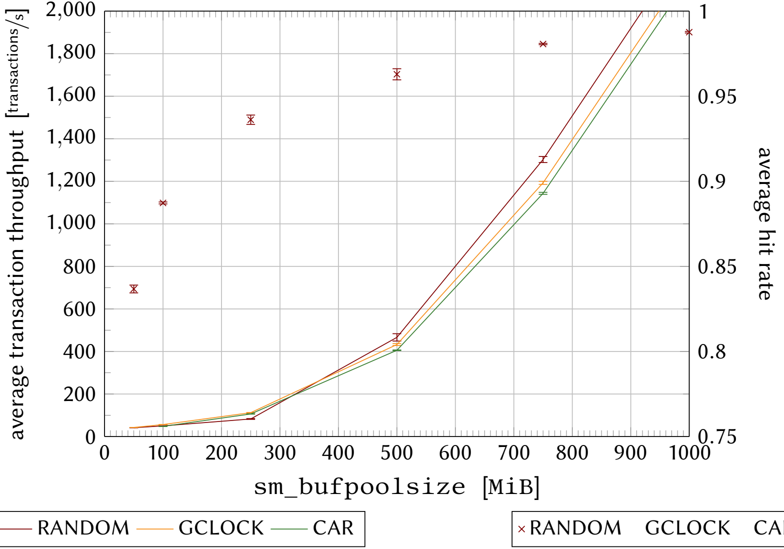
Buffer Pool With Pointer Swizzling (≤1 GiB, TPC-C)



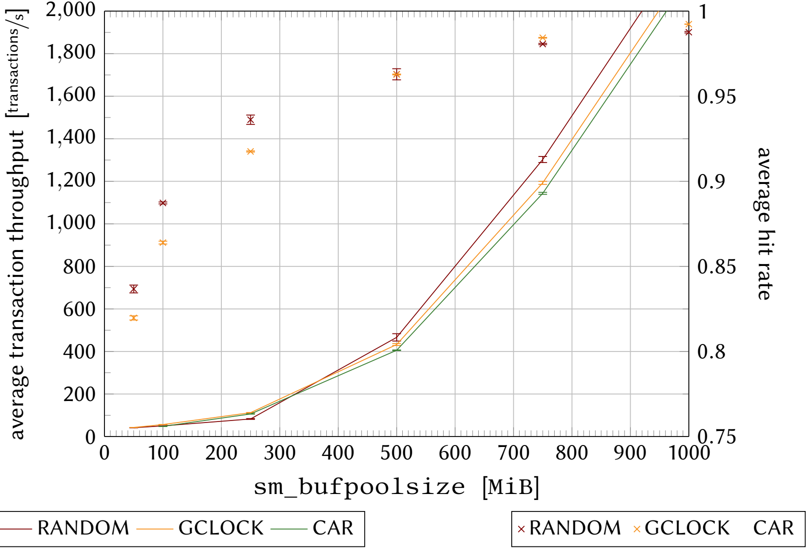
Buffer Pool With Pointer Swizzling (≤1 GiB, TPC-C)



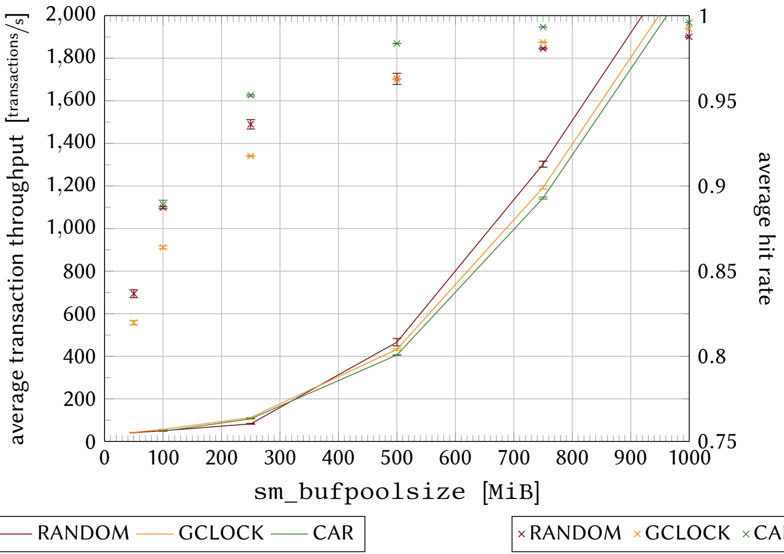
Buffer Pool With Pointer Swizzling (≤1 GiB, TPC-C)



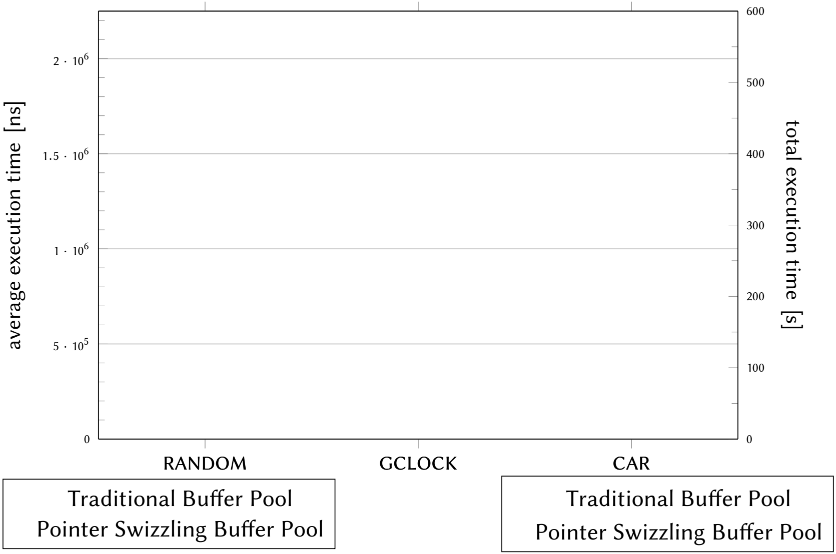
Buffer Pool With Pointer Swizzling (≤1 GiB, TPC-C)



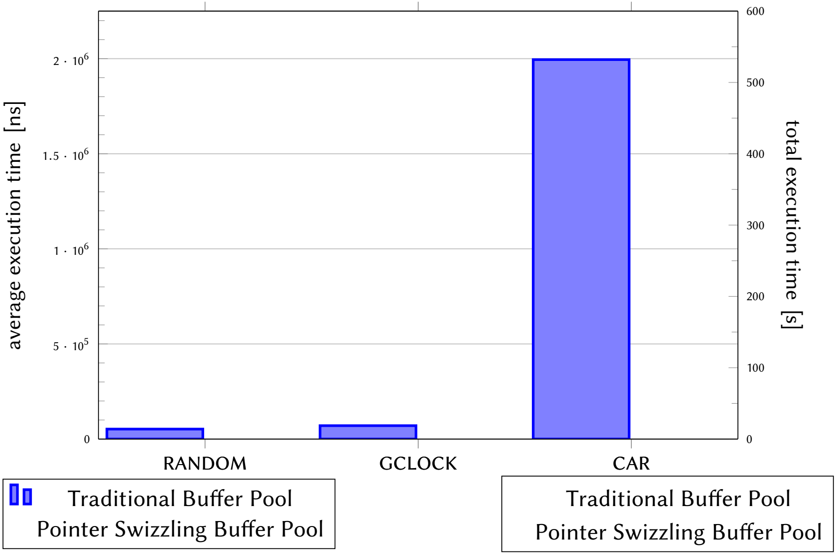
Buffer Pool With Pointer Swizzling (≤1 GiB, TPC-C)



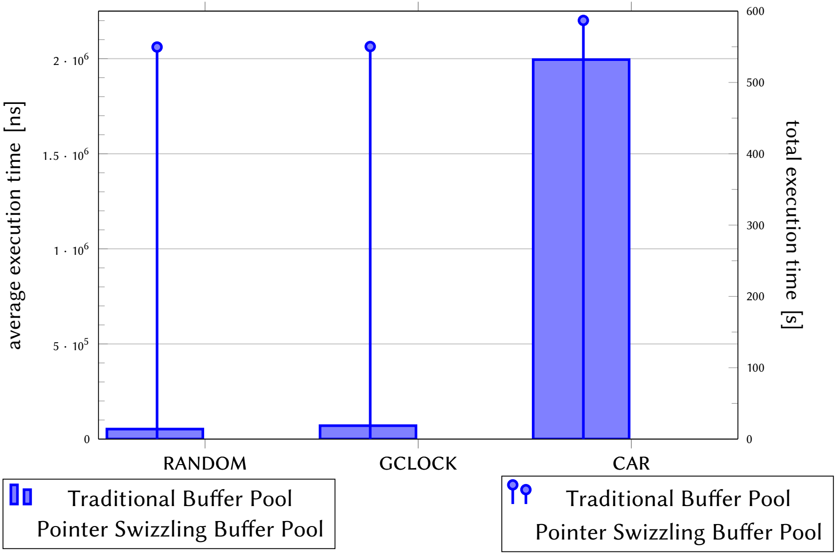
Operation Performance



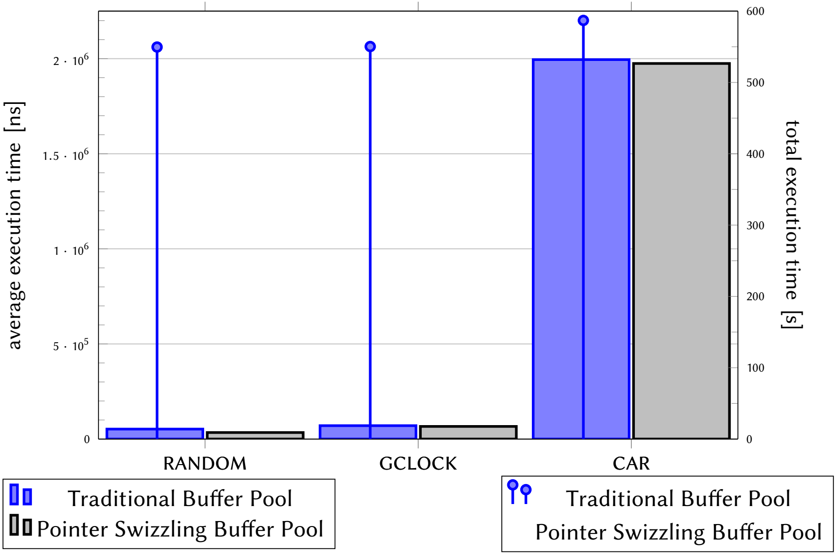
Operation Performance



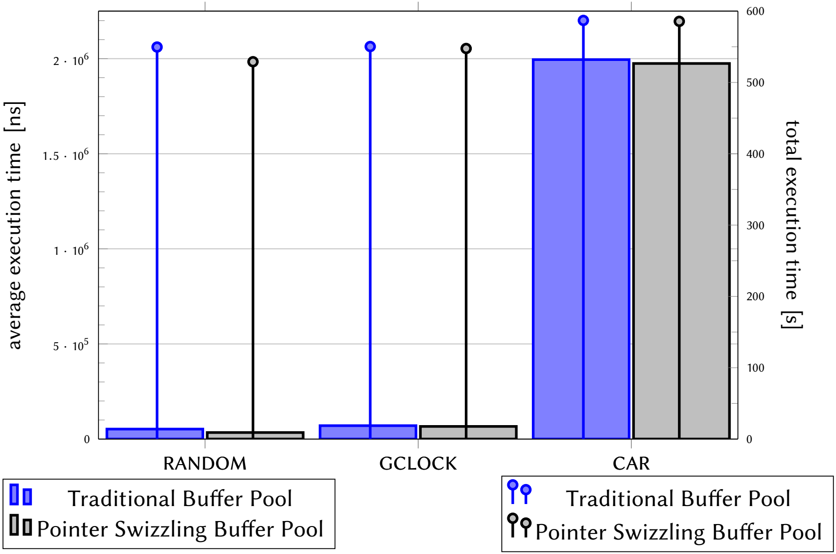
Operation Performance



Operation Performance



Operation Performance



Subsection 4

Conclusion

Conclusion

Performance

Conclusion

Performance

- CAR has a significantly higher hit rate than RANDOM or GCLOCK

Conclusion

Performance

- CAR has a significantly higher hit rate than RANDOM or GCLOCK
- The hit rate of GCLOCK isn't significantly higher than the one of RANDOM

Conclusion

Performance

- CAR has a significantly higher hit rate than RANDOM or GCLOCK
- The hit rate of GCLOCK isn't significantly higher than the one of RANDOM
- Major differences in hit rate are only for buffer pool sizes of $\leq \frac{1}{10}$ of the database size

Conclusion

Performance

- CAR has a significantly higher hit rate than RANDOM or GCLOCK
- The hit rate of GCLOCK isn't significantly higher than the one of RANDOM
- Major differences in hit rate are only for buffer pool sizes of $\leq \frac{1}{10}$ of the database size
- The computational effort spent to do CAR eviction is 27–58 times higher

Conclusion

Performance

- CAR has a significantly higher hit rate than RANDOM or GCLOCK
- The hit rate of GCLOCK isn't significantly higher than the one of RANDOM
- Major differences in hit rate are only for buffer pool sizes of $\leq \frac{1}{10}$ of the database size
- The computational effort spent to do CAR eviction is 27–58 times higher
- The overall performance of CAR isn't better than the one of RANDOM or GCLOCK

References I

Sorav Bansal and Dharmendra S. Modha. "CAR: Clock with Adaptive Replacement". Mar. 31, 2004.

Wolfgang Effelsberg and Theo Härder. "Principles of Database Buffer Management". Dec. 1984.

Enterprise-Festplatten: 36 High-Performance-Festplatten im Vergleichstest. Oct. 2, 2013. URL: http://www.tomshardware.de/enterprise-hdd-sshd,testberichte-241390-6.html (visited on Feb. 8, 2017).

Goetz Graefe et al. "In-Memory Performance for Big Data". Sept. 2014.

References II

- Theo Härder and Erhard Rahm. *Datenbanksysteme - Konzepte und Techniken der Implementierung.* 2001. ISBN: 978-3-642-62659-3.
- Theo Härder and Andreas Reuter. “Concepts for Implementing a Centralized Database Management System”. 1983.
- Theo Härder and Andreas Reuter. “Architektur von Datenbanksystemen für Non-Standard-Anwendungen”. Jan. 1985.
- Andreas Heuer, Gunter Saake, and Kai-Uwe Sattler. *Datenbanken - Implementierungstechniken.* 2011. ISBN: 978-3-8266-9156-0.
- J. Eliot B. Moss. “Working with Persistent Objects: To Swizzle or Not to Swizzle”. Aug. 1992.

References III

Igor Pavlov. *Intel Skylake.* URL: http://www.7-cpu.com/cpu/Skylake.html (visited on Jan. 19, 2017).

Lucas Sauer Caetano Lersch, Theo Härder, and Goetz Graefe. "Update propagation strategies for high-performance OLTP". Aug. 14, 2016.

Seagates Speicherriese ist schnell und sehr sparsam. Aug. 16, 2016. URL: https://www.computerbase.de/2016-08/seagate-enterprise-capacity-3.5-hdd-10tb-test/3/#diagramm-zugriffszeiten-lesen-h2benchw-316 (visited on Feb. 8, 2017).

References IV

- Wenguang Wang. "Storage Management in RDBMS". Aug. 17, 2001. URL: http://www.gohappycup.com/personal/comprehensive.pdf (visited on Feb. 2, 2017).
- Seth J. White and David J. DeWitt. "QuickStore: A High Performance Mapped Object Store". Oct. 1995.
- "Why SSDs Are Awesome - An SSD Primer". Aug. 2015.
- Wikipedia. *Page replacement algorithm.* 2017. URL: https://en.wikipedia.org/wiki/Page_replacement_algorithm (visited on Mar. 21, 2017).

Your Turn to Ask ...