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”

Subsection 1

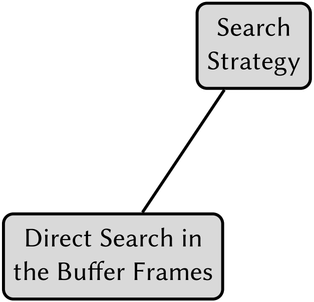
Locate Pages in the Buffer Pool without Pointer Swizzling

Overview of Search Strategies ([HR01])

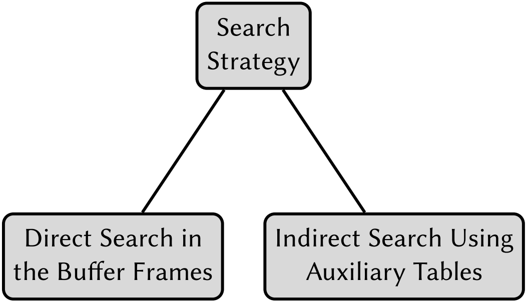
Overview of Search Strategies ([HR01])

Search Strategy

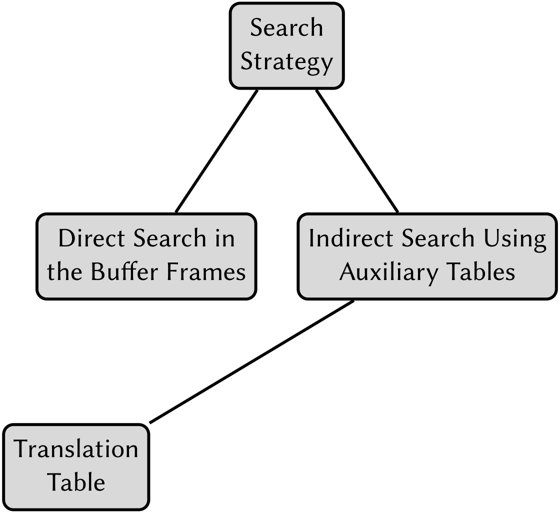
Overview of Search Strategies ([HR01])



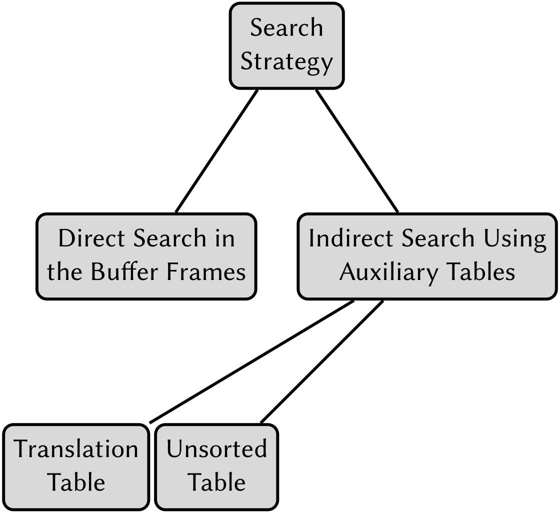
Overview of Search Strategies ([HR01])


Search Strategy
Direct Search in the Buffer Frames
Indirect Search Using Auxiliary Tables

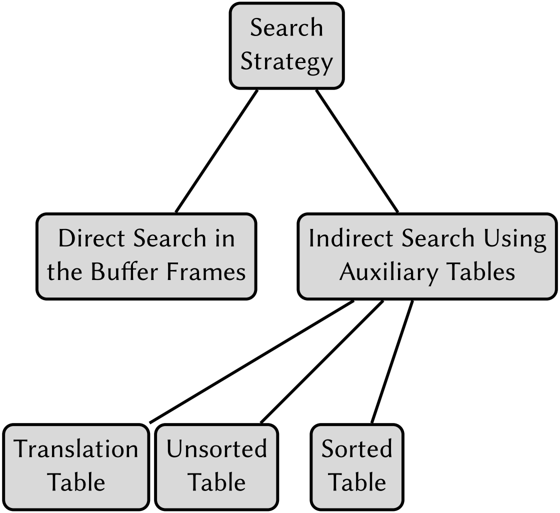
Overview of Search Strategies ([HR01])


Search Strategy
Direct Search in the Buffer Frames
Indirect Search Using Auxiliary Tables
Translation Table

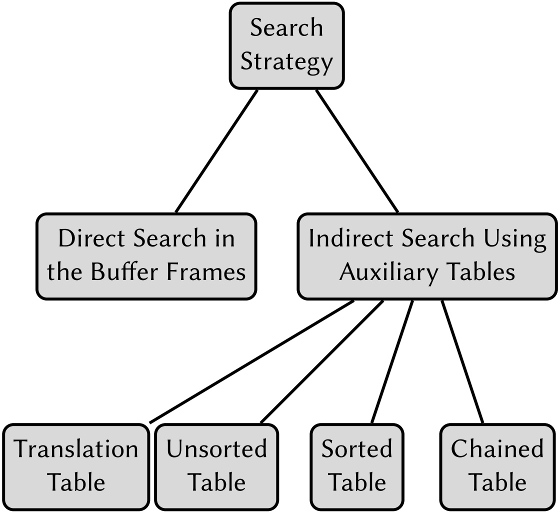
Overview of Search Strategies ([HR01])


Search Strategy
Direct Search in the Buffer Frames
Indirect Search Using Auxiliary Tables
Translation Table
Unsorted Table

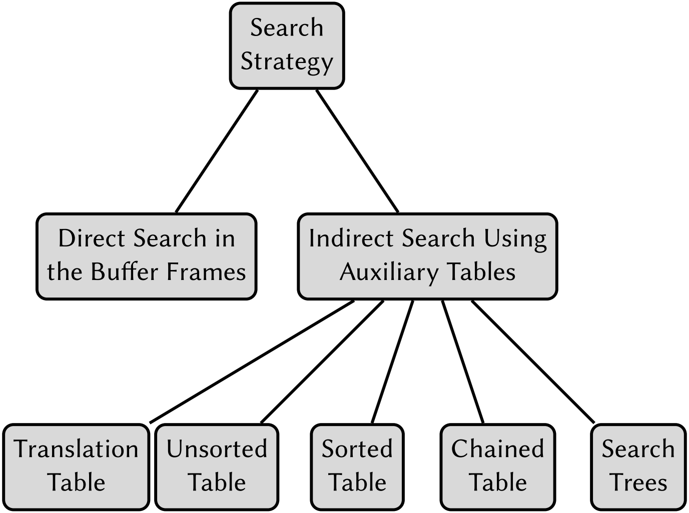
Overview of Search Strategies ([HR01])


Search Strategy
Direct Search in the Buffer Frames
Indirect Search Using Auxiliary Tables
Translation Table
Unsorted Table
Sorted Table

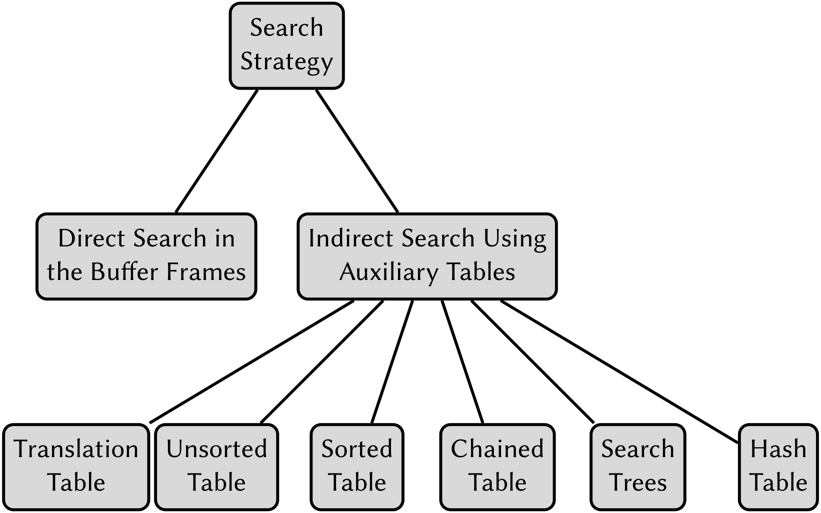
Overview of Search Strategies ([HR01])


Search Strategy
Direct Search in the Buffer Frames
Indirect Search Using Auxiliary Tables
Translation Table
Unsorted Table
Sorted Table
Chained Table

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
- $T^{\text{search}}_{\text{avg}} \in \mathcal{O}\left(\frac{n}{2}\right)$, $T^{\text{search}}_{\text{worst}} \in \mathcal{O}\left(n\right)$
- 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
- $T^{\text{search}}_{\text{avg}} \in \mathcal{O}\left(\frac{n}{2}\right)$, $T^{\text{search}}_{\text{worst}} \in \mathcal{O}\left(n\right)$
- 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 $S_{\text{pace}} \in \mathcal{O}\left(n\right)$
- $T^{\text{search}}_{\text{avg}} \in \mathcal{O}\left(\frac{n}{2}\right)$, $T^{\text{search}}_{\text{worst}} \in \mathcal{O}\left(n\right)$

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
  $\Longrightarrow \; S_{\text{pace}} \in \mathcal{O}\left(p\right)$

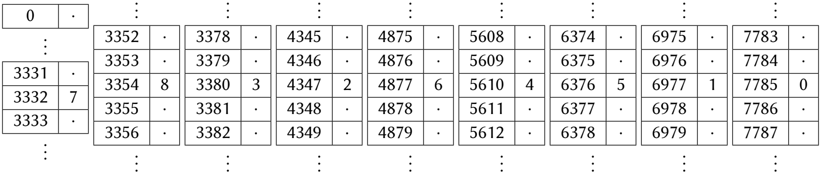
0	·
⋮	
3331	·
3332	7
3333	·
⋮	

⋮		⋮		⋮		⋮		⋮		⋮		⋮		⋮	
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 $\Longrightarrow\ S_{\text{pace}} \in \mathcal{O}(p)$
- $T^{\text{search}} \in \mathcal{O}(1)$, $T^{\text{insert}} \in \mathcal{O}(1)$



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**
→ **7**	→ **8**	→ **3**	→ **2**	→ **6**	→ **4**	→ **5**	→ **1**	→ **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**
→ **7**	→ **8**	→ **3**	→ **2**	→ **6**	→ **4**	→ **5**	→ **1**	→ **0**

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

Chained Table

Sorted & Chained Table

Sorted Table

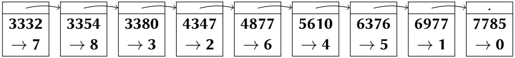
- Auxiliary data structure using a table sorted by page ID only containing cached pages
- $T_{\text{avg}}^{\text{search}} \in \mathcal{O}\left(\log_2 n\right)$, $T_{\text{avg}}^{\text{insert}} \in \mathcal{O}\left(n \log_2 n\right)$

3332	**3354**	**3380**	**4347**	**4877**	**5610**	**6376**	**6977**	**7785**
→ **7**	→ **8**	→ **3**	→ **2**	→ **6**	→ **4**	→ **5**	→ **1**	→ **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	**3354**	**3380**	**4347**	**4877**	**5610**	**6376**	**6977**	**7785**
→ **7**	→ **8**	→ **3**	→ **2**	→ **6**	→ **4**	→ **5**	→ **1**	→ **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 **→ 7**	**3354** **→ 8**	**3380** **→ 3**	**4347** **→ 2**	**4877** **→ 6**	**5610** **→ 4**	**6376** **→ 5**	**6977** **→ 1**	**7785** **→ 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
- $T_{\text{avg}}^{\text{search}} \in \mathcal{O}\left(\log_2 n\right)$, $T_{\text{avg}}^{\text{insert}} \in \mathcal{O}\left(\log_2 n\right)$

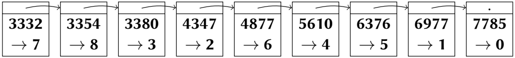


Figure: A chained table used to map page IDs to buffer frames.

Sorted & Chained Table

Sorted Table

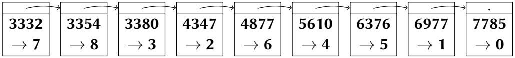
- Auxiliary data structure using a table sorted by page ID only containing cached pages
- $T_{\text{avg}}^{\text{search}} \in \mathcal{O}\left(\log_2 n\right)$, $T_{\text{avg}}^{\text{insert}} \in \mathcal{O}\left(n \log_2 n\right)$

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

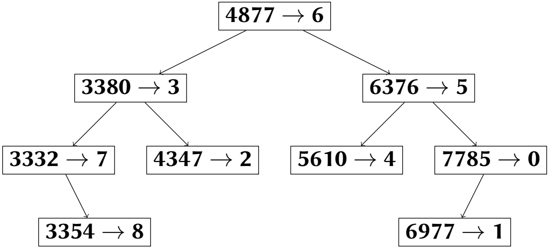


Figure: A balanced search tree used to map page IDs to buffer frames.

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

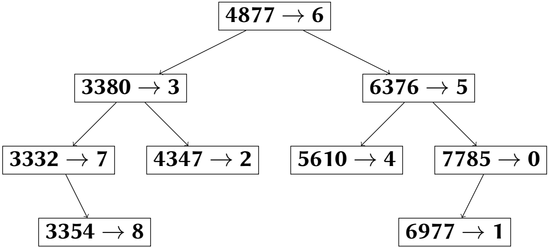


Figure: A balanced search tree used to map page IDs to buffer frames.

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
- $T_{\text{avg}}^{\text{search}} \in \mathcal{O}(\log n)$, $T_{\text{avg}}^{\text{insert}} \in \mathcal{O}(\log n)$

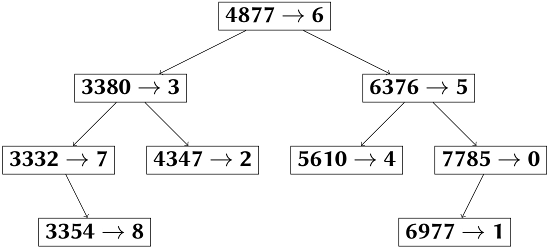


Figure: A balanced search tree used to map page IDs to buffer frames.

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

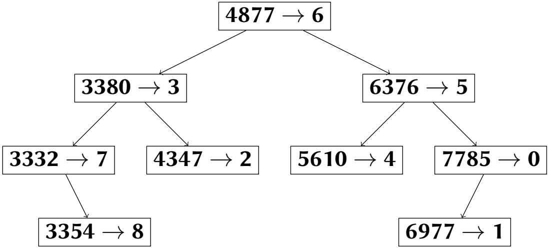
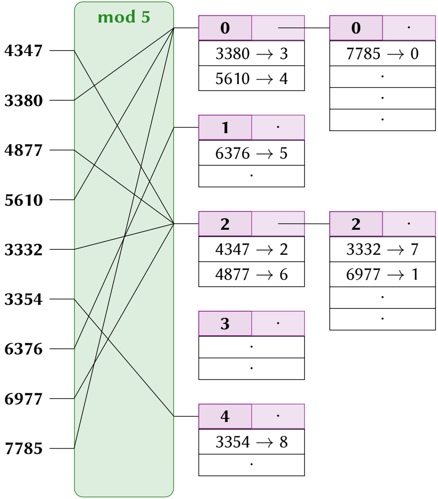


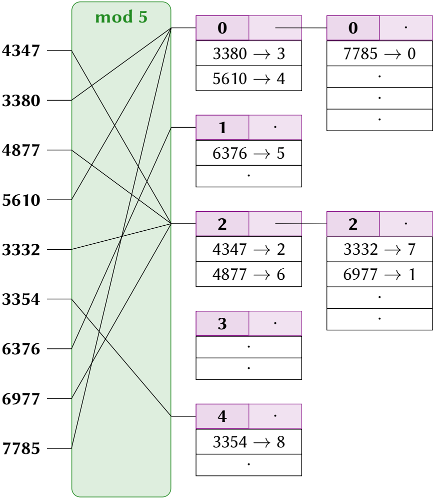
Figure: A balanced search tree used to map page IDs to buffer frames.

Hash Table

Hash Table

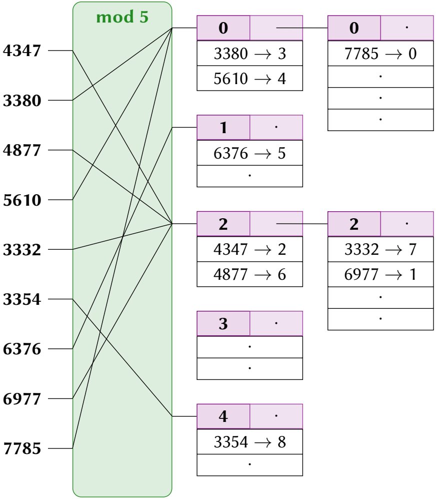

mod 5
4347
3380
4877
5610
3332
3354
6376
6977
7785
0
3380 → 3
5610 → 4
0
7785 → 0
1
6376 → 5
2
4347 → 2
4877 → 6
2
3332 → 7
6977 → 1
3
4
3354 → 8

Hash Table



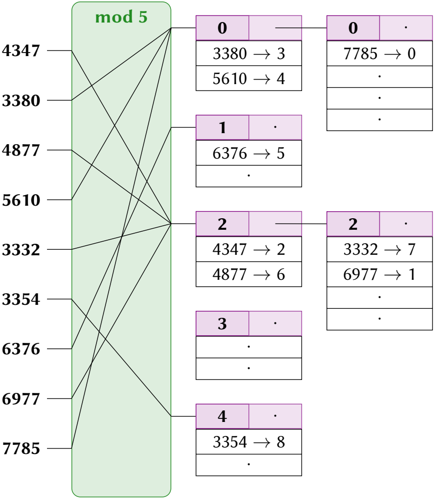
- Each page ID is mapped to a hash bucket using a hash function

Hash Table



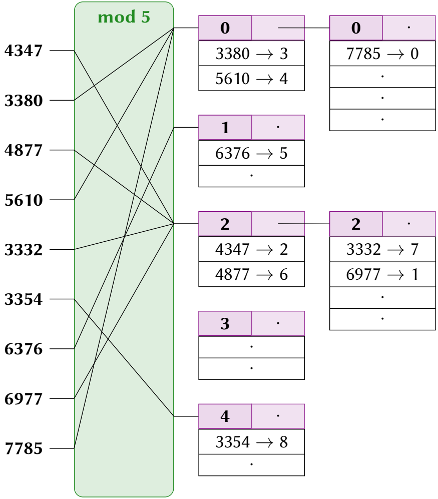
- Each page ID is mapped to a hash bucket using a hash function
- Only the page IDs of buffered pages are in the hash table

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

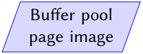
Hash Table



- 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
- $T_{\text{avg}}^{\text{search}} \in \mathcal{O}(1)$, $T_{\text{avg}}^{\text{insert}} \in \mathcal{O}(1)$, $T_{\text{worst}}^{\text{search}} \in \mathcal{O}(n)$

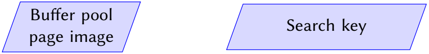
Locate Pages in Buffer Pool with Hash Table ([Gra+14])

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



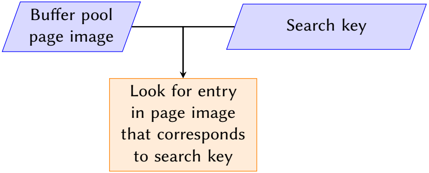
Buffer pool page image

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

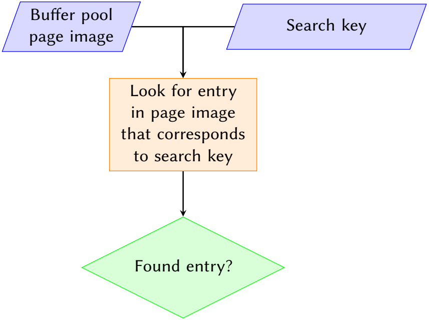

Buffer pool page image

Search key

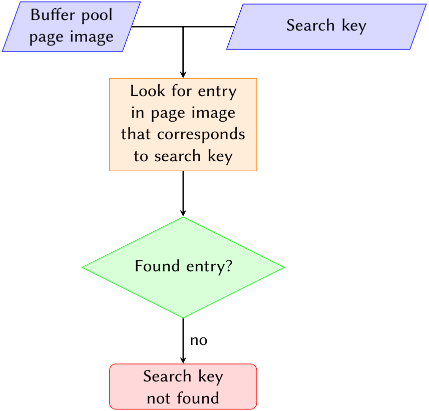
Locate Pages in Buffer Pool with Hash Table ([Gra+14])



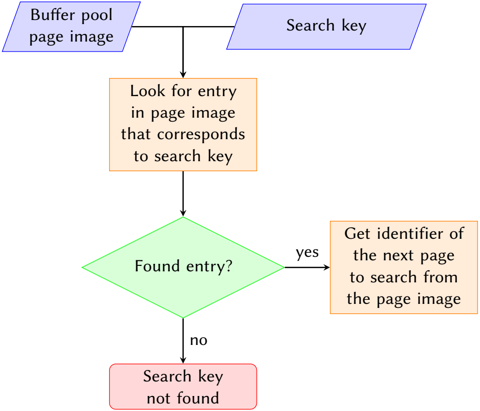
Locate Pages in Buffer Pool with Hash Table ([Gra+14])



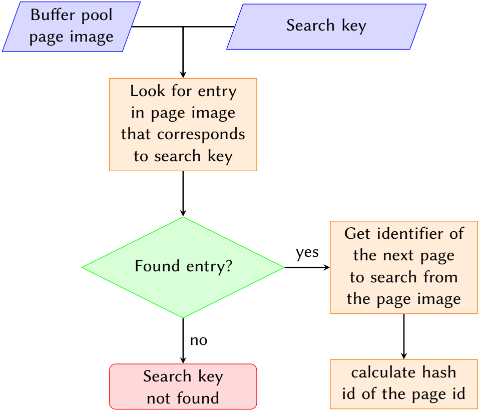
Locate Pages in Buffer Pool with Hash Table ([Gra+14])



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


Buffer pool page image
Search key
Look for entry in page image that corresponds to search key
Found entry?
yes
Get identifier of the next page to search from the page image
no
Search key not found

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


Buffer pool page image

Search key

Look for entry in page image that corresponds to search key

Found entry?

yes

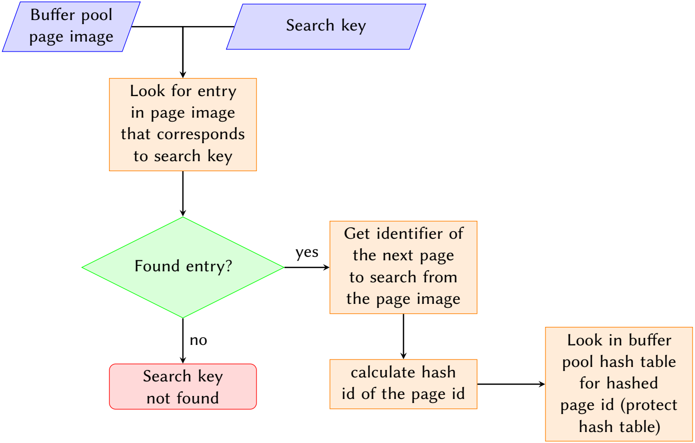
Get identifier of the next page to search from the page image

calculate hash id of the page id

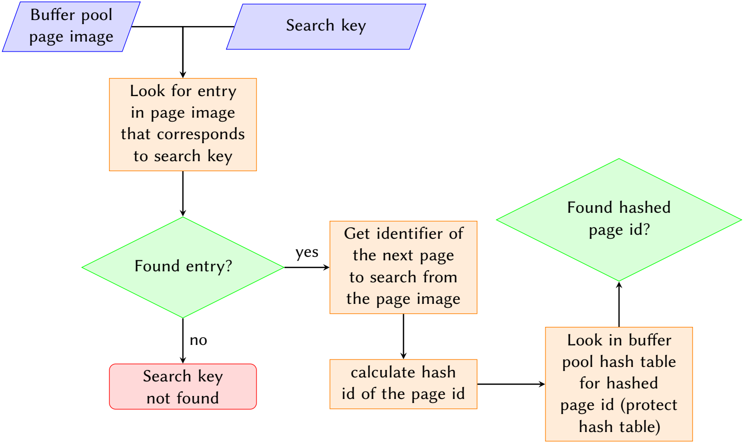
no

Search key not found

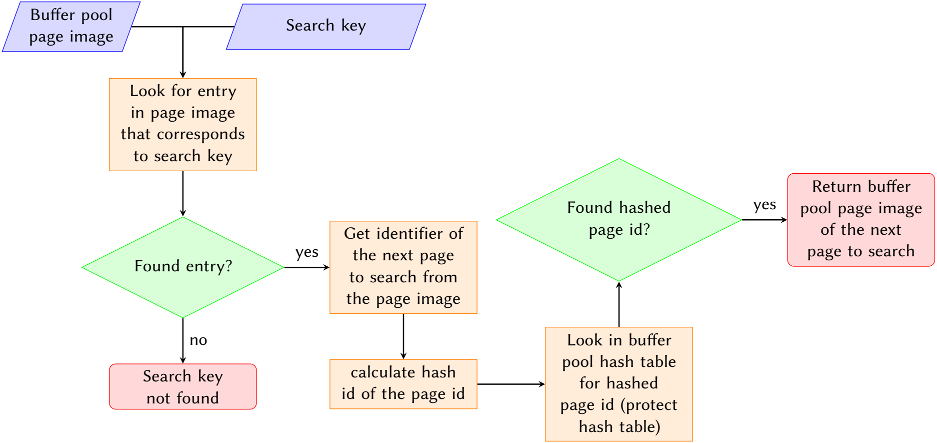
Locate Pages in Buffer Pool with Hash Table ([Gra+14])


Buffer pool page image
Search key
Look for entry in page image that corresponds to search key
Found entry?
yes
Get identifier of the next page to search from the page image
no
Search key not found
calculate hash id of the page id
Look in buffer pool hash table for hashed page id (protect hash table)

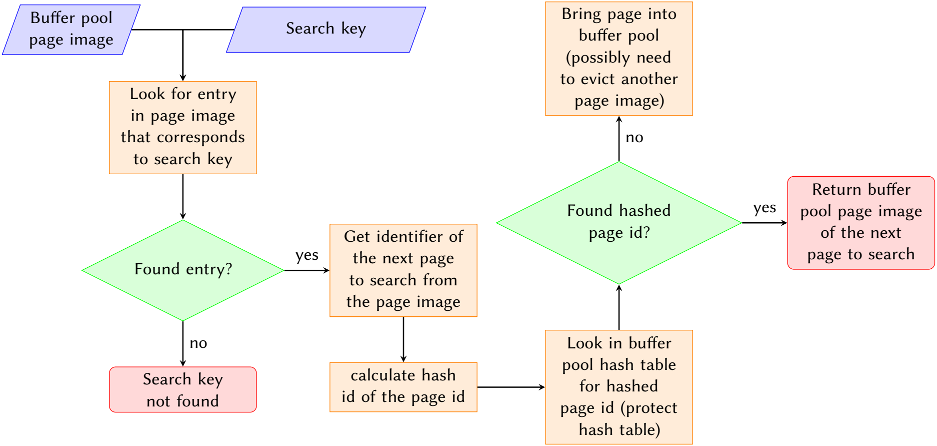
Locate Pages in Buffer Pool with Hash Table ([Gra+14])


Buffer pool page image
Search key
Look for entry in page image that corresponds to search key
Found entry?
yes
no
Search key not found
Get identifier of the next page to search from the page image
calculate hash id of the page id
Look in buffer pool hash table for hashed page id (protect hash table)
Found hashed page id?

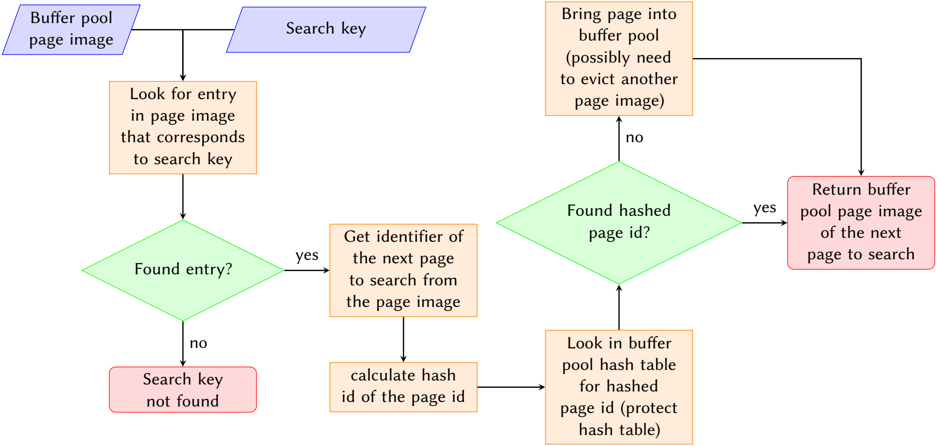
Locate Pages in Buffer Pool with Hash Table ([Gra+14])


Buffer pool page image
Search key
Look for entry in page image that corresponds to search key
Found entry?
no
Search key not found
yes
Get identifier of the next page to search from the page image
calculate hash id of the page id
Look in buffer pool hash table for hashed page id (protect hash table)
Found hashed page id?
yes
Return buffer pool page image of the next page to search

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


Buffer pool page image
Search key
Look for entry in page image that corresponds to search key
Found entry?
yes
no
Search key not found
Get identifier of the next page to search from the page image
calculate hash id of the page id
Look in buffer pool hash table for hashed page id (protect hash table)
Found hashed page id?
no
Bring page into buffer pool (possibly need to evict another page image)
yes
Return buffer pool page image of the next page to search

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


Buffer pool page image
Search key
Look for entry in page image that corresponds to search key
Found entry?
no
Search key not found
yes
Get identifier of the next page to search from the page image
calculate hash id of the page id
Look in buffer pool hash table for hashed page id (protect hash table)
Found hashed page id?
no
Bring page into buffer pool (possibly need to evict another page image)
yes
Return buffer pool page image of the next page to search

Subsection 2

Locate Pages in the Buffer Pool with Pointer Swizzling

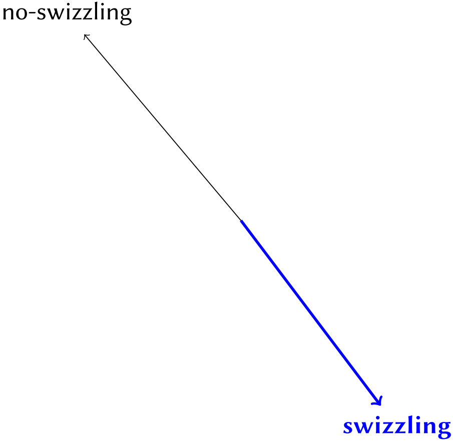
Pointer Swizzling

Definition

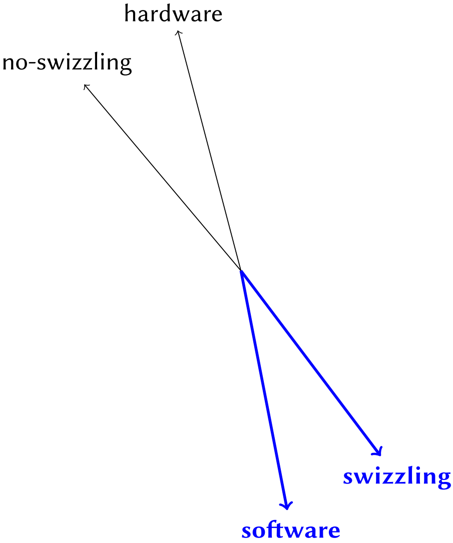
To swizzle a pointer means to transform the address of the persistent object referenced there to a more direct address of the transient object in a way that this transformation could be used during multiple indirections of this pointer ([Mos92]).

Classification of the Pointer Swizzling Approach following [WD95]

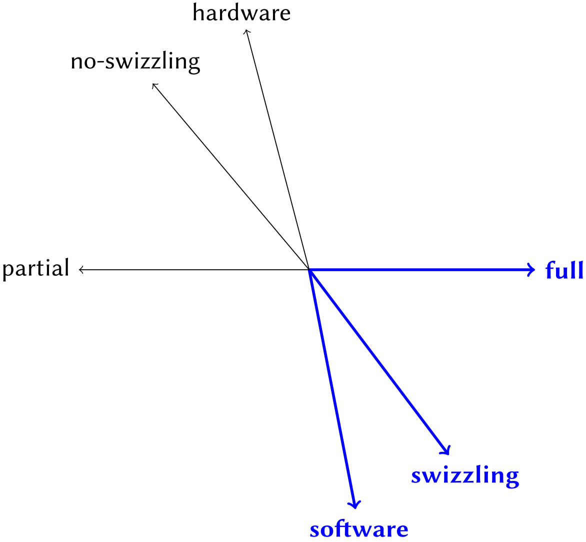
Classification of the Pointer Swizzling Approach following [WD95]



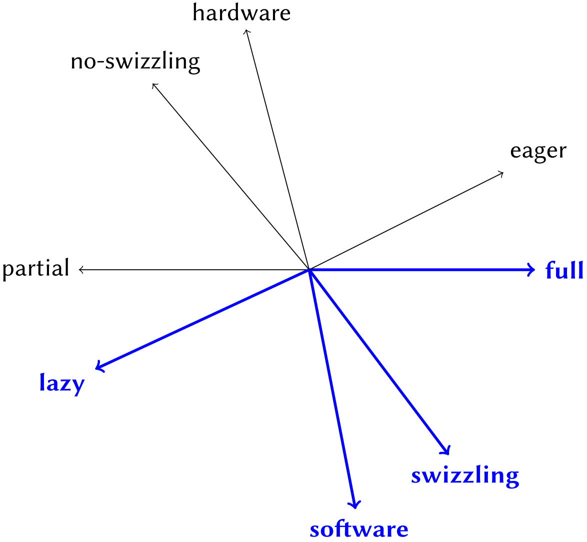
Classification of the Pointer Swizzling Approach following [WD95]



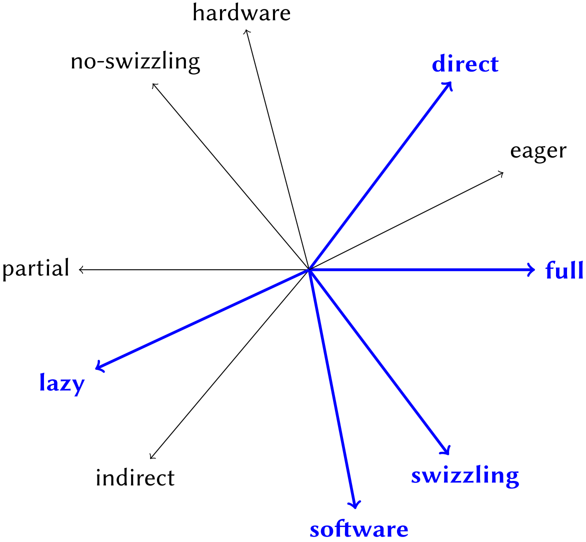
Classification of the Pointer Swizzling Approach following [WD95]



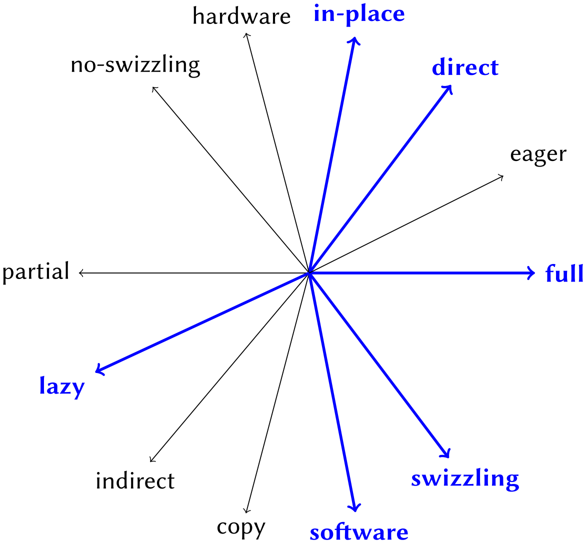
Classification of the Pointer Swizzling Approach following [WD95]



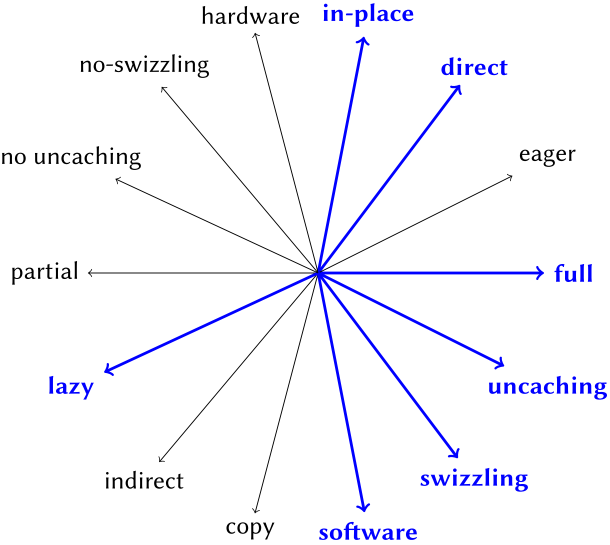
Classification of the Pointer Swizzling Approach following [WD95]



Classification of the Pointer Swizzling Approach following [WD95]

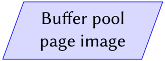


Classification of the Pointer Swizzling Approach following [WD95]



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

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



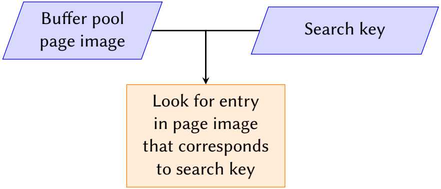
Buffer pool page image

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

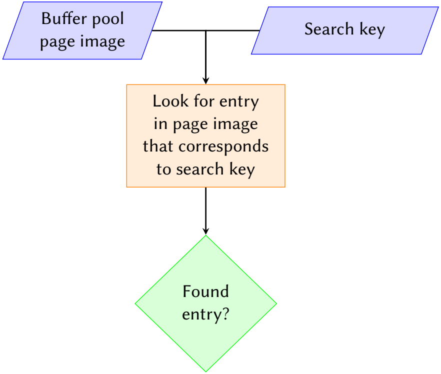
Buffer pool page image

Search key

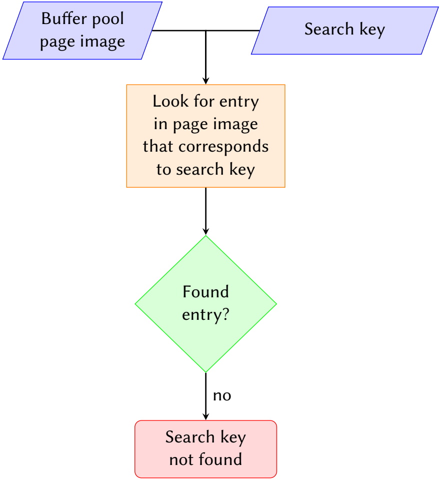
Locate Pages in Buffer Pool w/ Pointer Swizzling ([Gra+14])



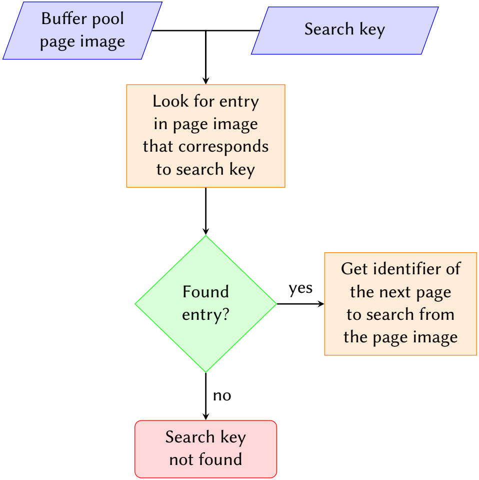
Locate Pages in Buffer Pool w/ Pointer Swizzling ([Gra+14])



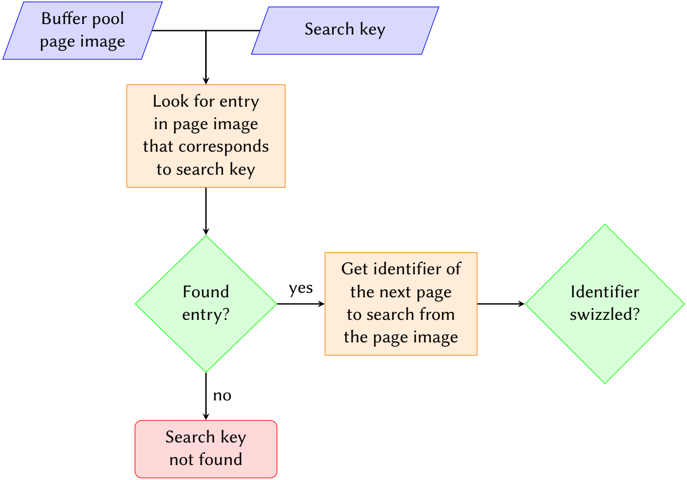
Locate Pages in Buffer Pool w/ Pointer Swizzling ([Gra+14])



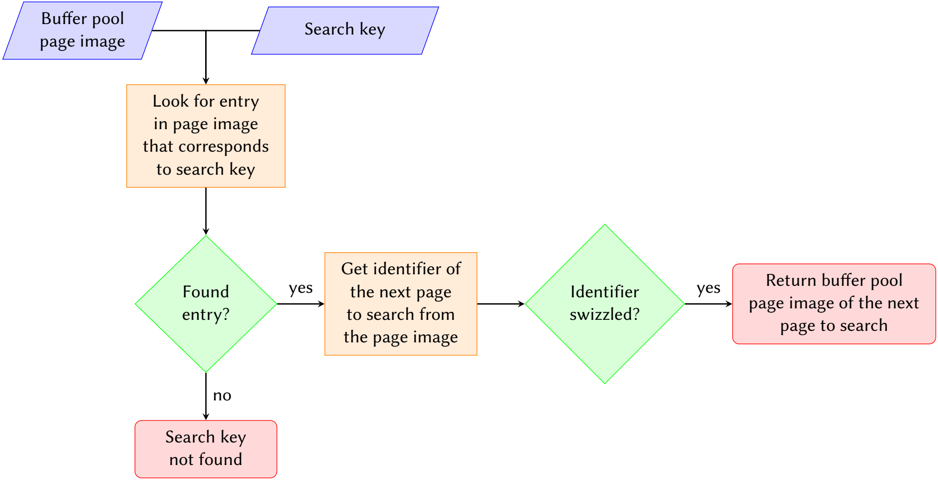
Locate Pages in Buffer Pool w/ Pointer Swizzling ([Gra+14])


Buffer pool page image
Search key
Look for entry in page image that corresponds to search key
Found entry?
yes
Get identifier of the next page to search from the page image
no
Search key not found

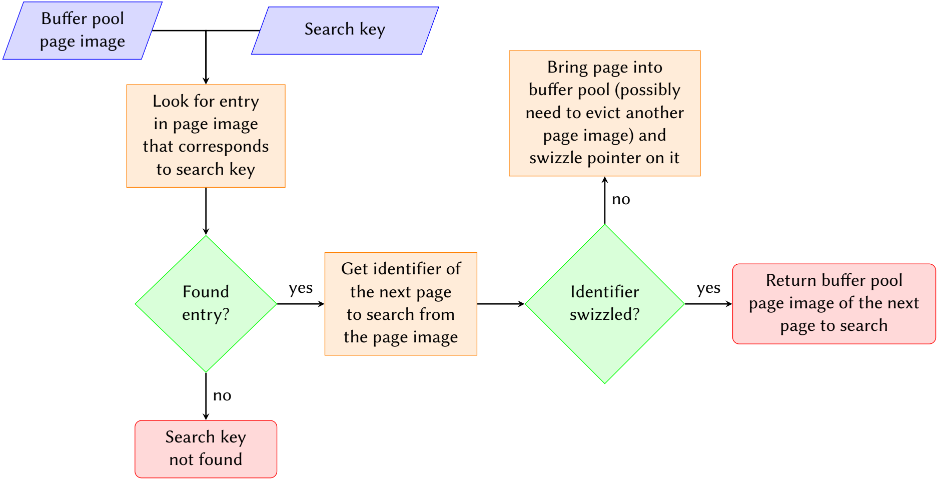
Locate Pages in Buffer Pool w/ Pointer Swizzling ([Gra+14])


Buffer pool page image
Search key
Look for entry in page image that corresponds to search key
Found entry?
yes
Get identifier of the next page to search from the page image
Identifier swizzled?
no
Search key not found

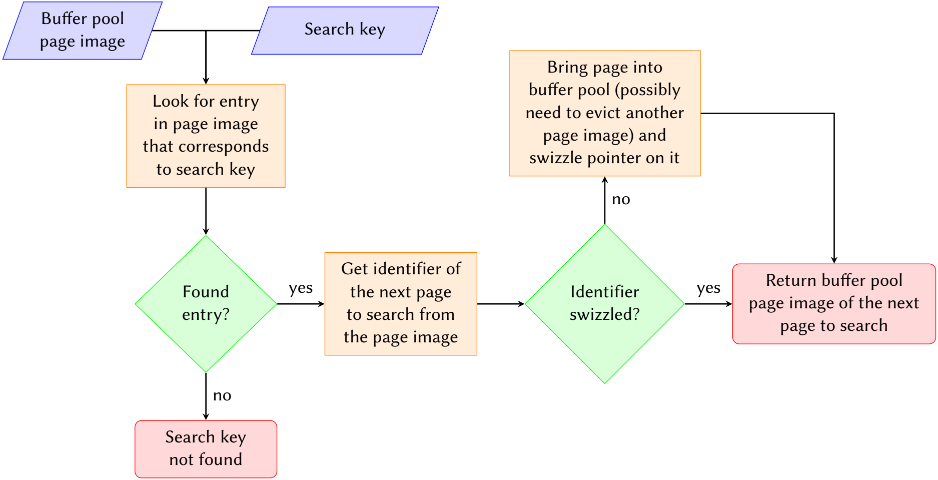
Locate Pages in Buffer Pool w/ Pointer Swizzling ([Gra+14])


Buffer pool page image
Search key
Look for entry in page image that corresponds to search key
Found entry?
yes
no
Search key not found
Get identifier of the next page to search from the page image
Identifier swizzled?
yes
Return buffer pool page image of the next page to search

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


Buffer pool page image
Search key
Look for entry in page image that corresponds to search key
Found entry?
yes
no
Search key not found
Get identifier of the next page to search from the page image
Identifier swizzled?
no
Bring page into buffer pool (possibly need to evict another page image) and swizzle pointer on it
yes
Return buffer pool page image of the next page to search

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


Buffer pool page image
Search key
Look for entry in page image that corresponds to search key
Found entry?
yes
no
Search key not found
Get identifier of the next page to search from the page image
Identifier swizzled?
no
Bring page into buffer pool (possibly need to evict another page image) and swizzle pointer on it
yes
Return buffer pool page image of the next page to search

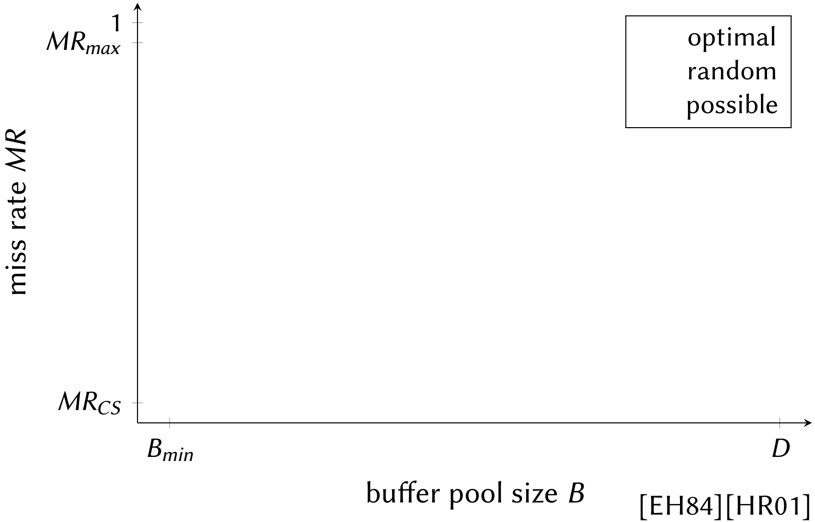
Section 2

Performance Evaluation of the Buffer Management Utilizing Pointer Swizzling

Subsection 1

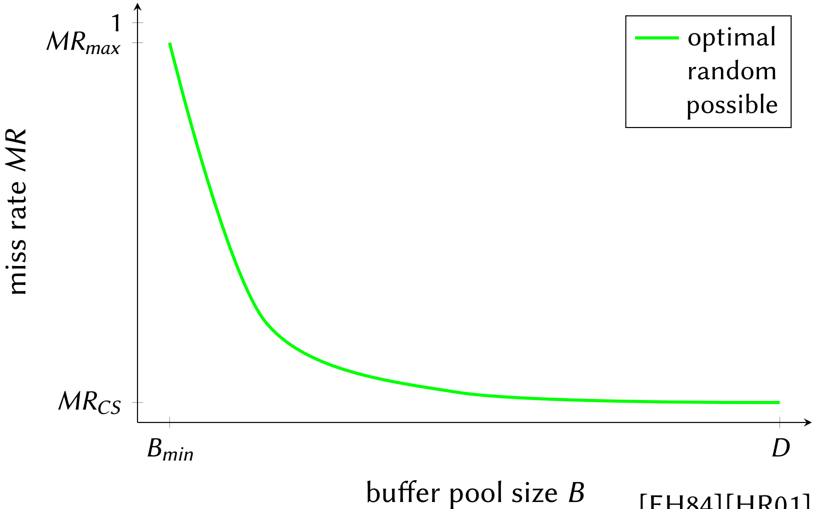
Expected Performance

Performance of Different Buffer Pool Sizes



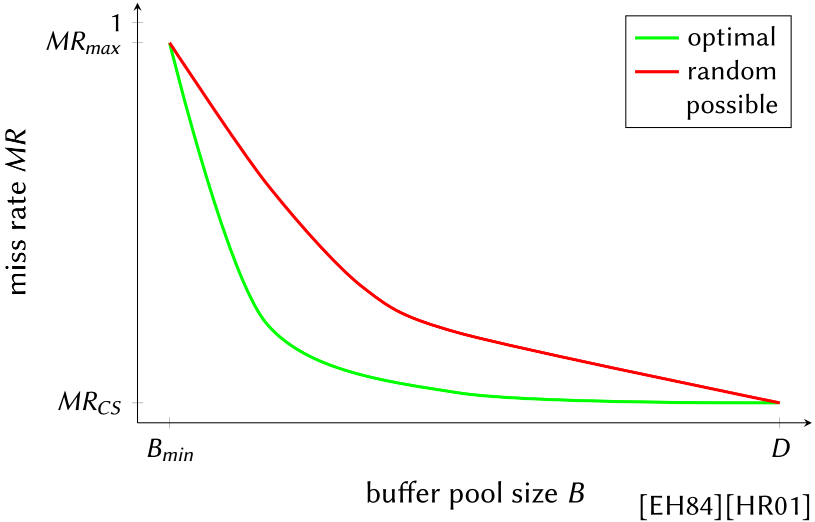
[EH84][HR01]

Performance of Different Buffer Pool Sizes



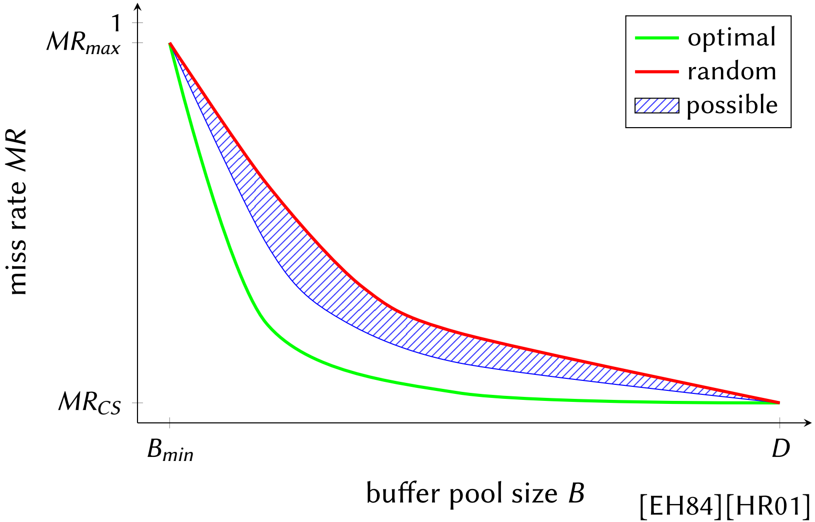
[EH84][HR01]

Performance of Different Buffer Pool Sizes



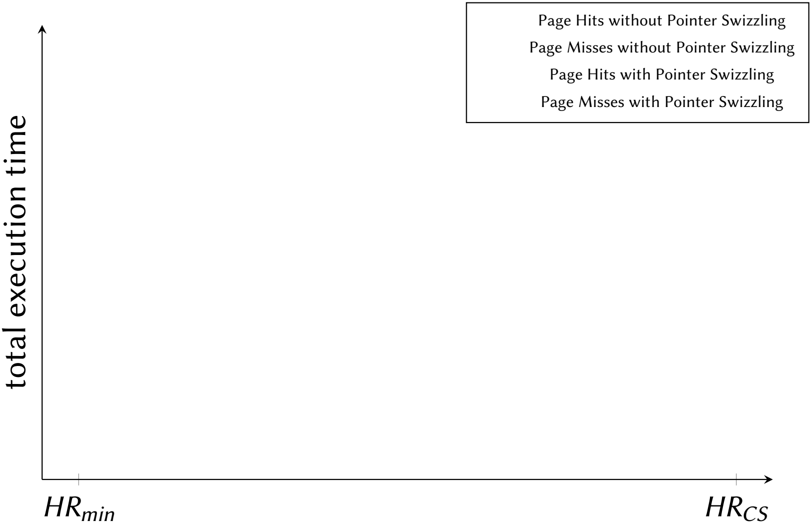
[EH84][HR01]

Performance of Different Buffer Pool Sizes

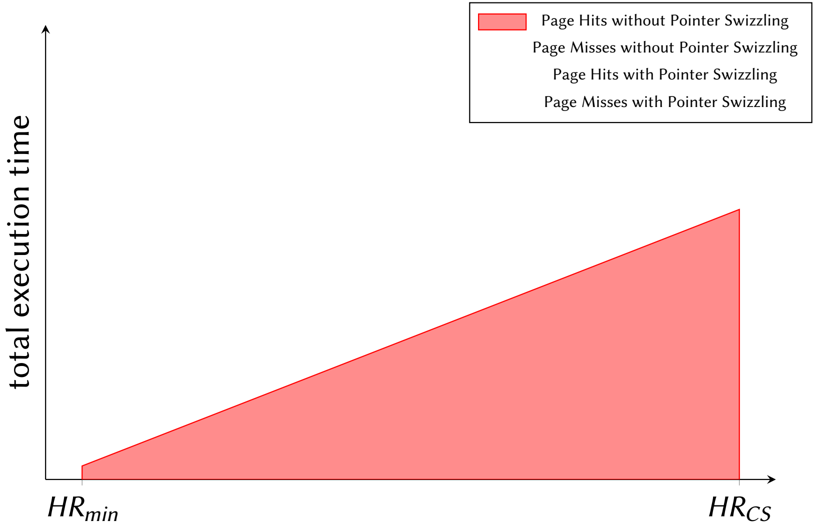


[EH84][HR01]

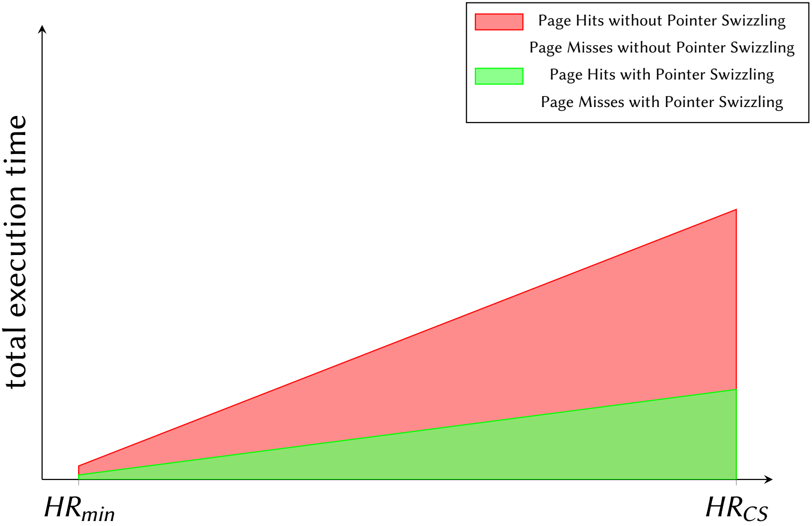
Buffer Management with and without Pointer Swizzling



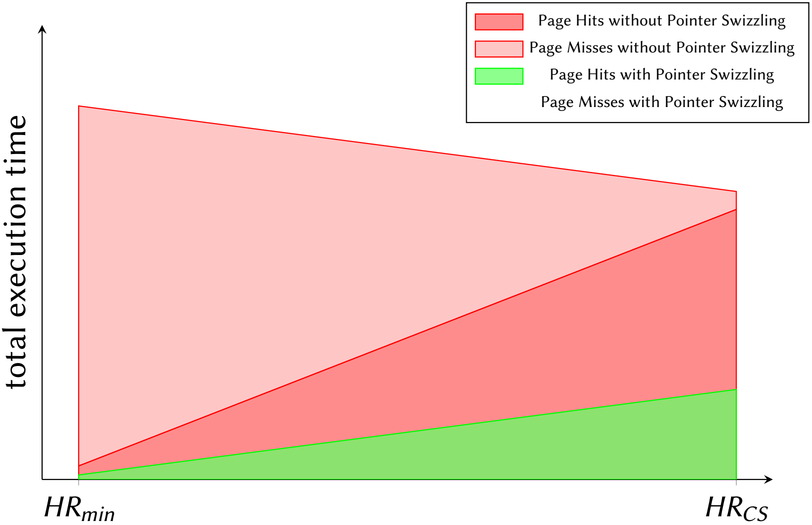
Buffer Management with and without Pointer Swizzling



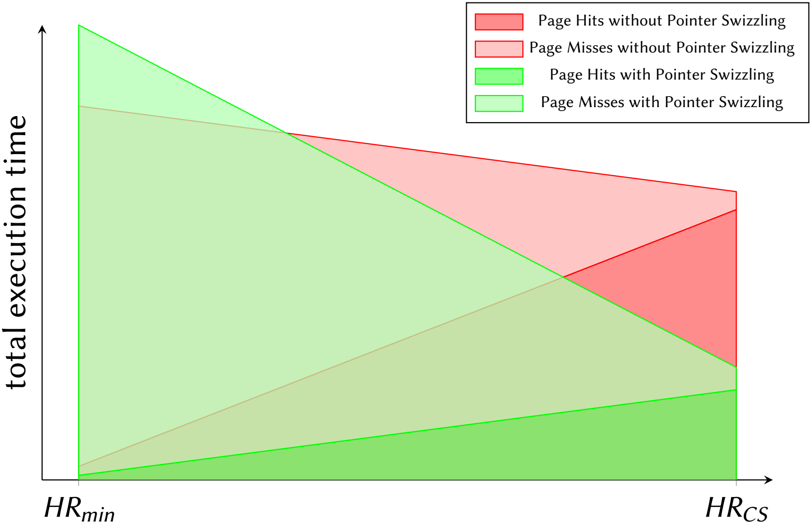
Buffer Management with and without Pointer Swizzling



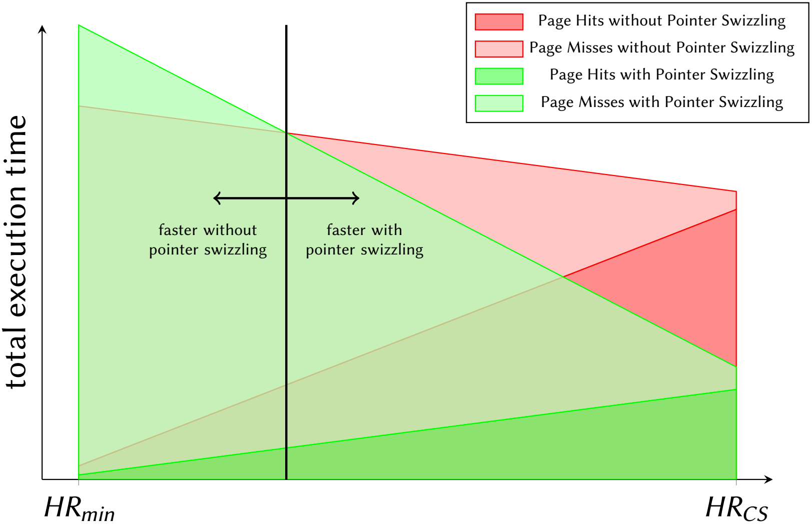
Buffer Management with and without Pointer Swizzling



Buffer Management with and without Pointer Swizzling



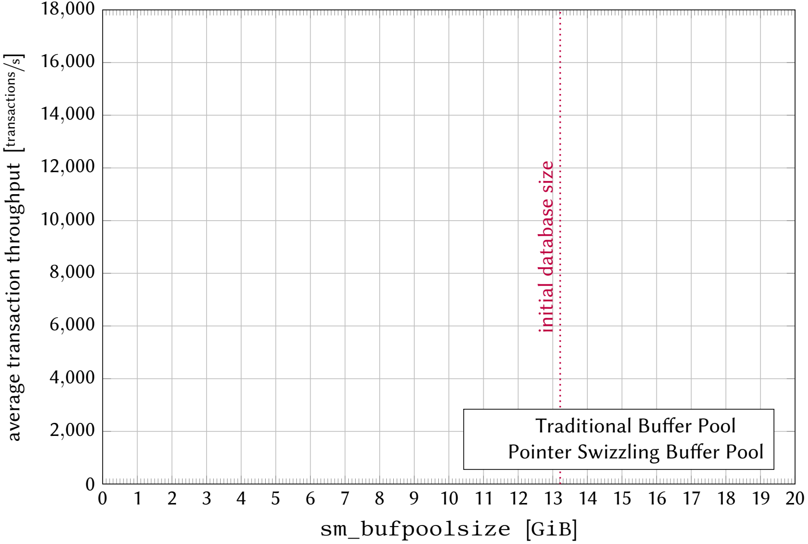
Buffer Management with and without Pointer Swizzling



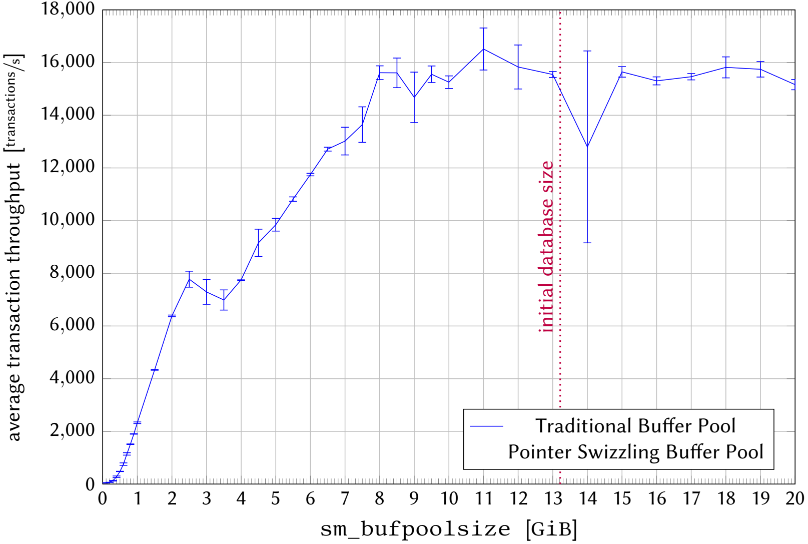
Subsection 2

Measured Performance

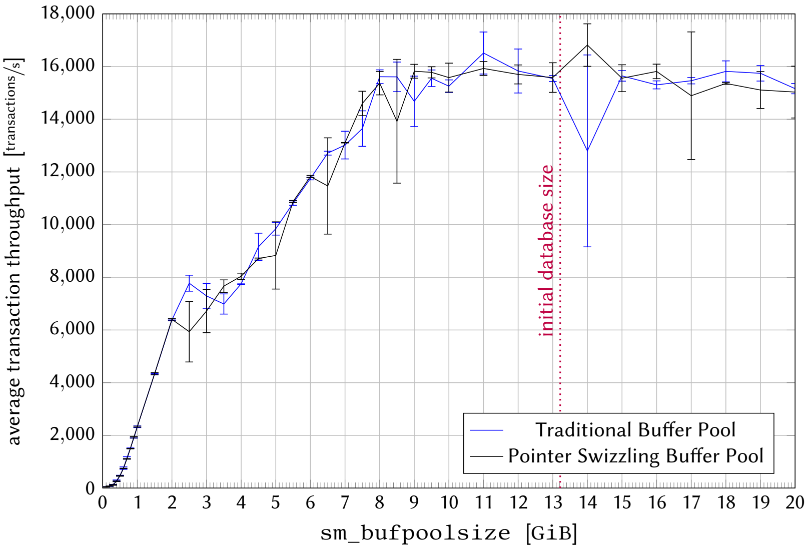
Transaction Throughput (TPC-C)



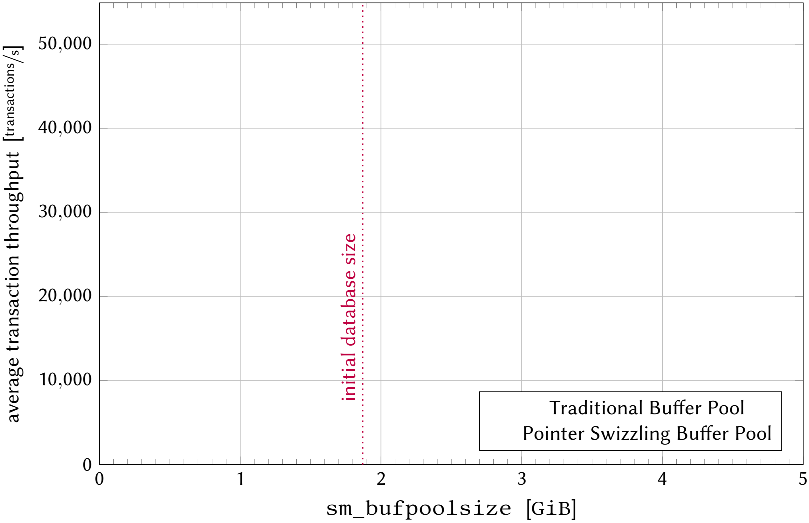
Transaction Throughput (TPC-C)



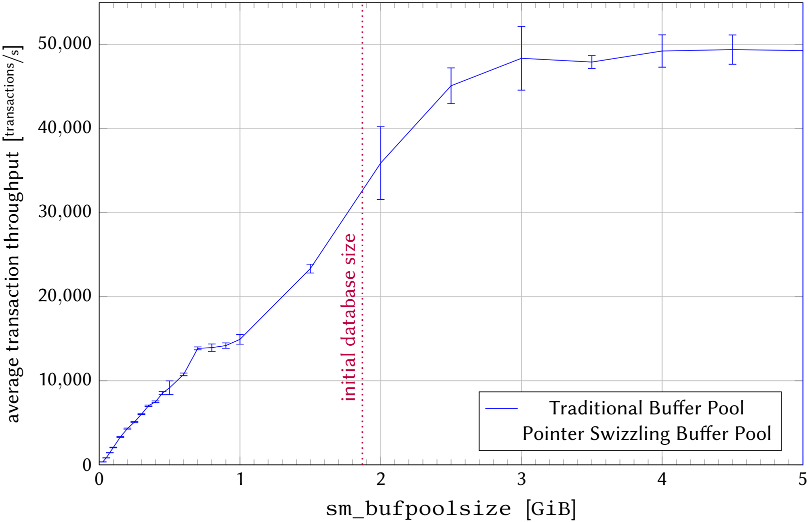
Transaction Throughput (TPC-C)



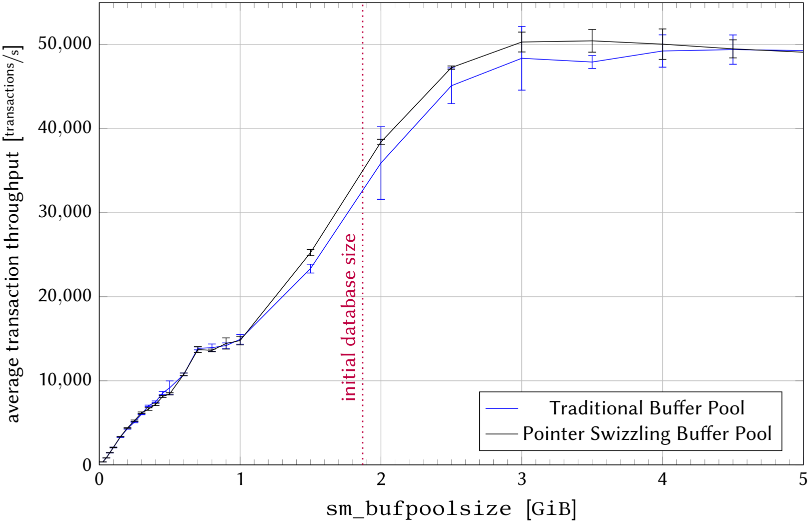
Transaction Throughput (TPC-B)



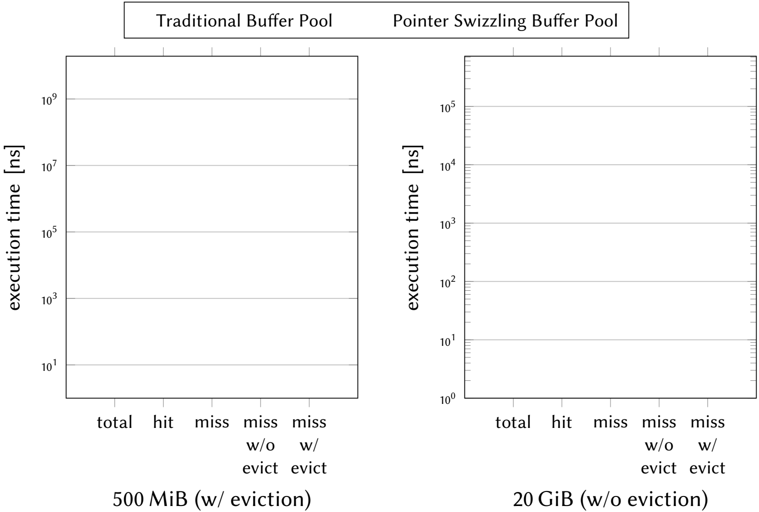
Transaction Throughput (TPC-B)



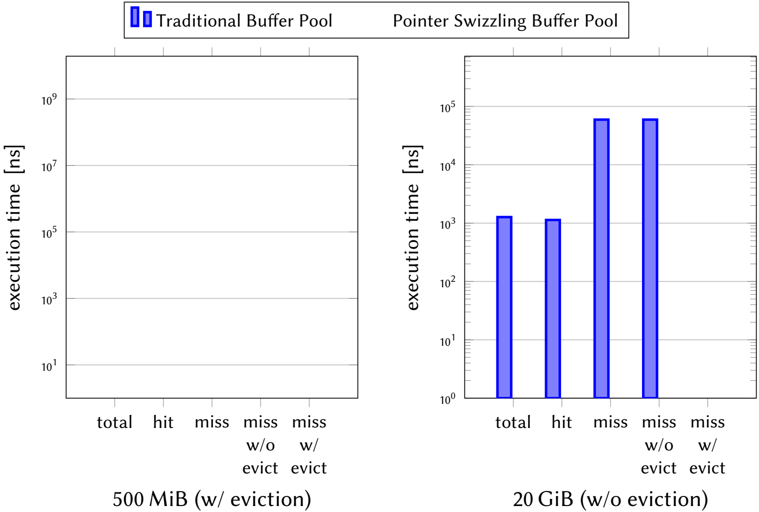
Transaction Throughput (TPC-B)



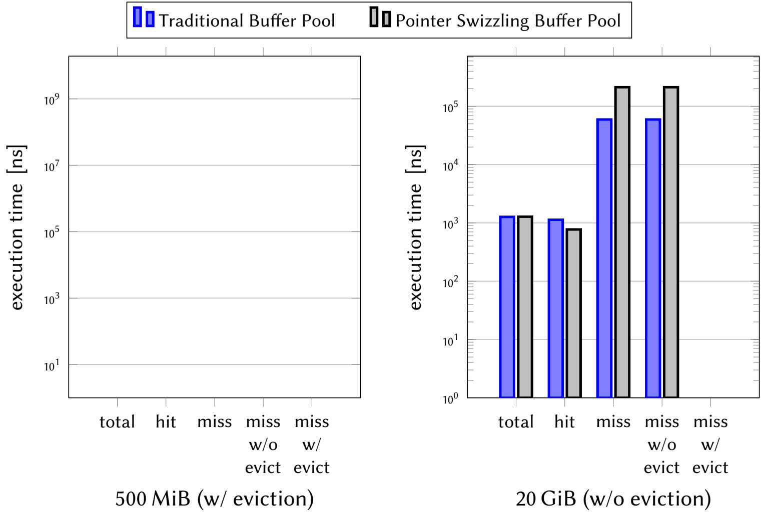
Buffer Pool Performance Acquiring Shared Latches



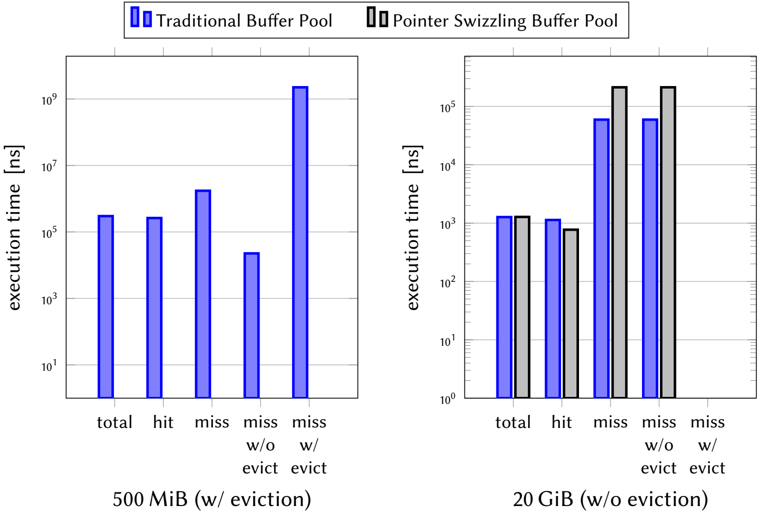
Buffer Pool Performance Acquiring Shared Latches



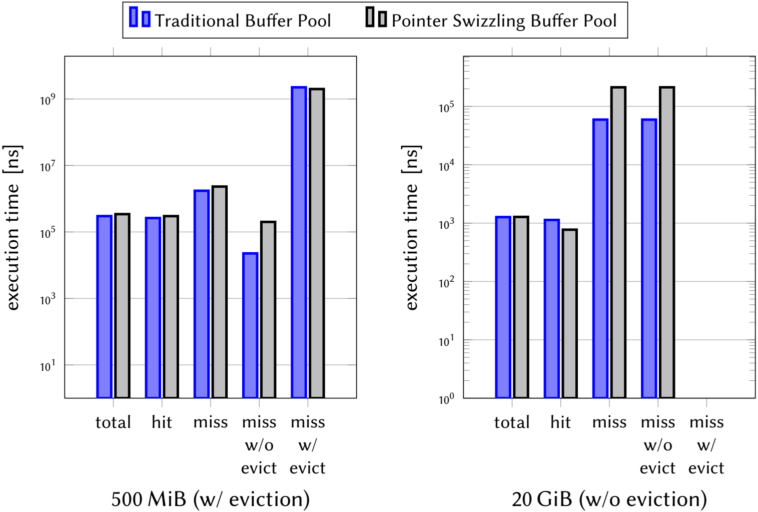
Buffer Pool Performance Acquiring Shared Latches



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

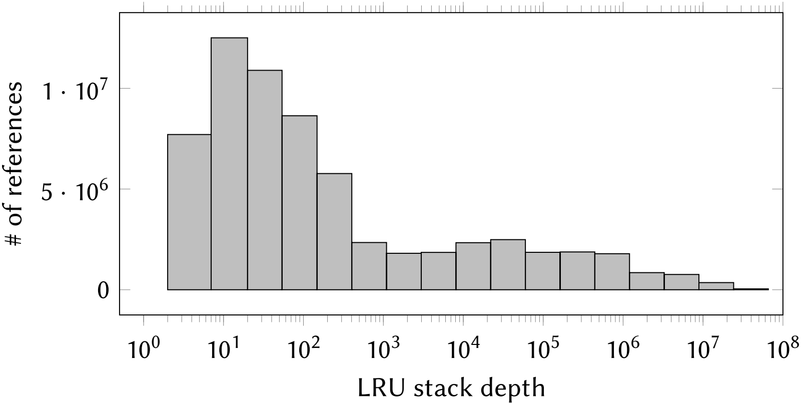
Page Eviction Strategies in the Context of Pointer Swizzling

Motivation not to Analyze Different Page Eviction Strategies

Motivation not to Analyze Different Page Eviction Strategies

- Even LRU results in decent hit rates

TPC-C with Warehouses: 100, Threads: 25



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

Evaluated Page Replacement Strategies

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

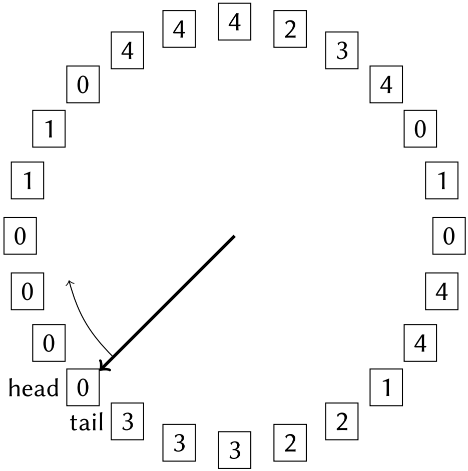
GCLOCK

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

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
  $\implies$ Higher hit rate
  $\implies$ 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

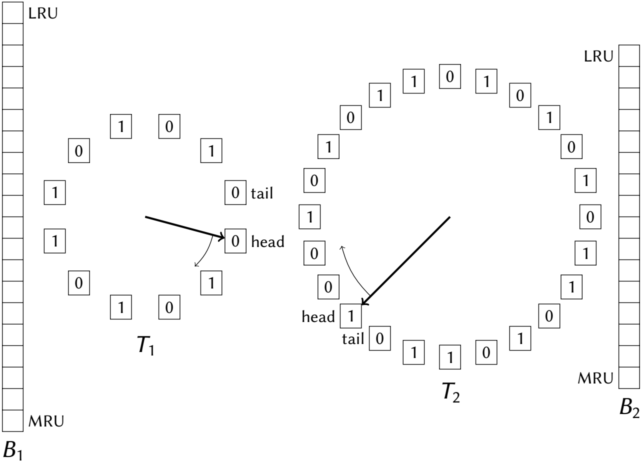
CAR

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

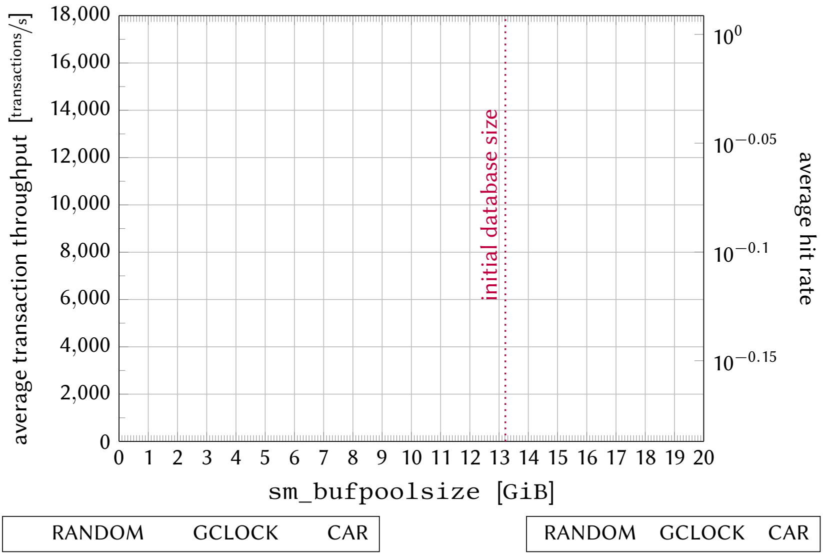
Example


LRU
MRU
B_1
1
0
1
0
0
tail
0
head
1
0
1
0
1
1
0
T_1
0
1
0
1
0
1
0
1
1
0
1
0
1
1
0
tail
1
head
0
0
1
0
1
0
1
1
T_2
LRU
MRU
B_2

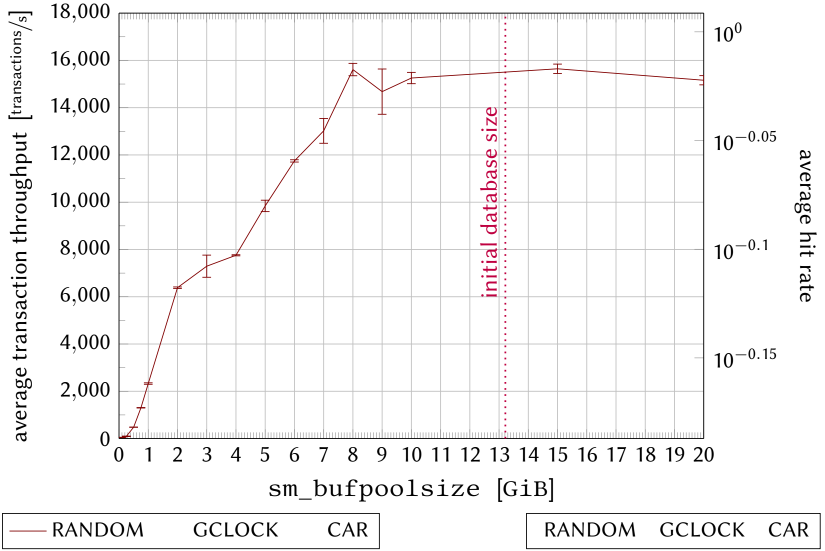
Subsection 3

Performance Evaluation

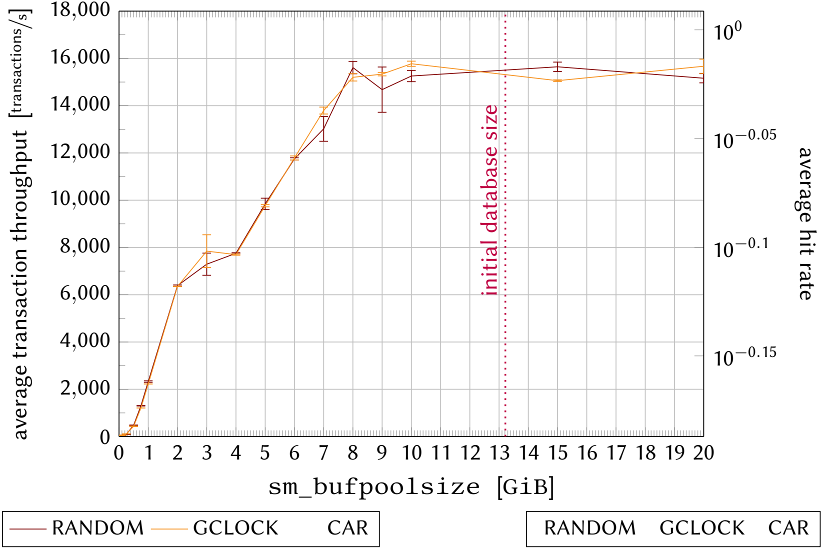
Buffer Pool Without Pointer Swizzling (TPC-C)



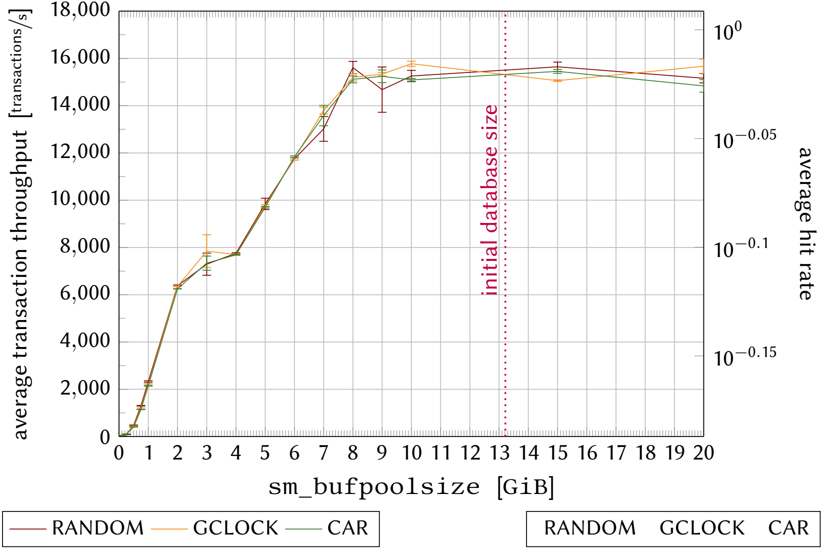
Buffer Pool Without Pointer Swizzling (TPC-C)



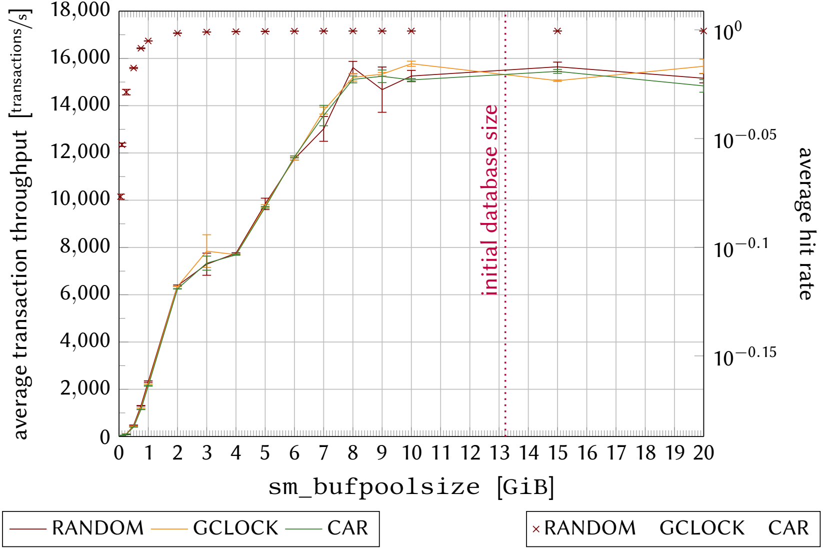
Buffer Pool Without Pointer Swizzling (TPC-C)



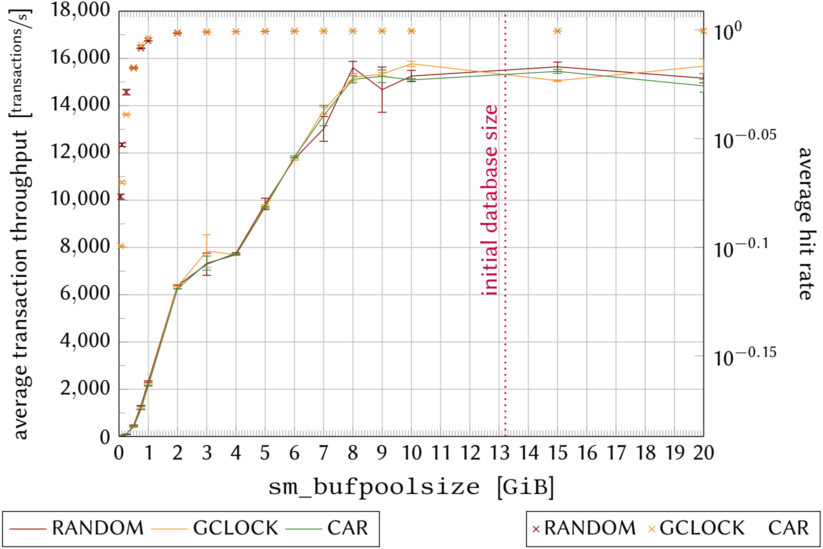
Buffer Pool Without Pointer Swizzling (TPC-C)



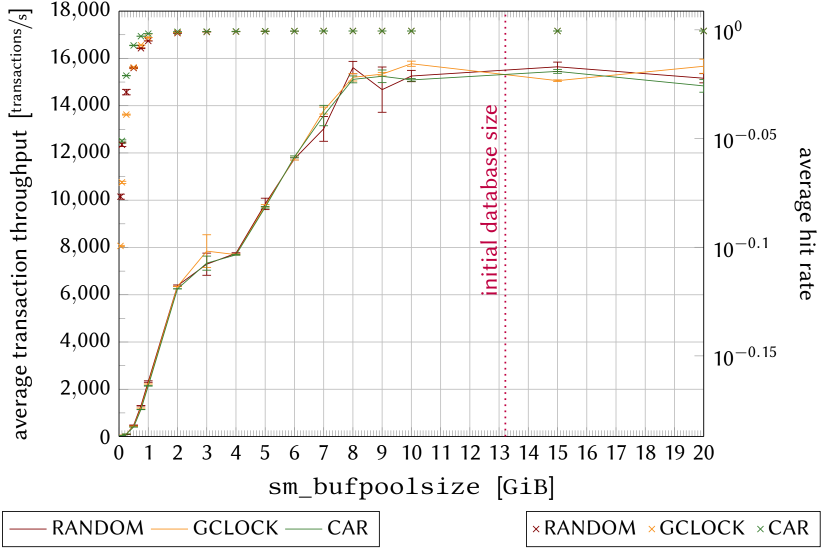
Buffer Pool Without Pointer Swizzling (TPC-C)



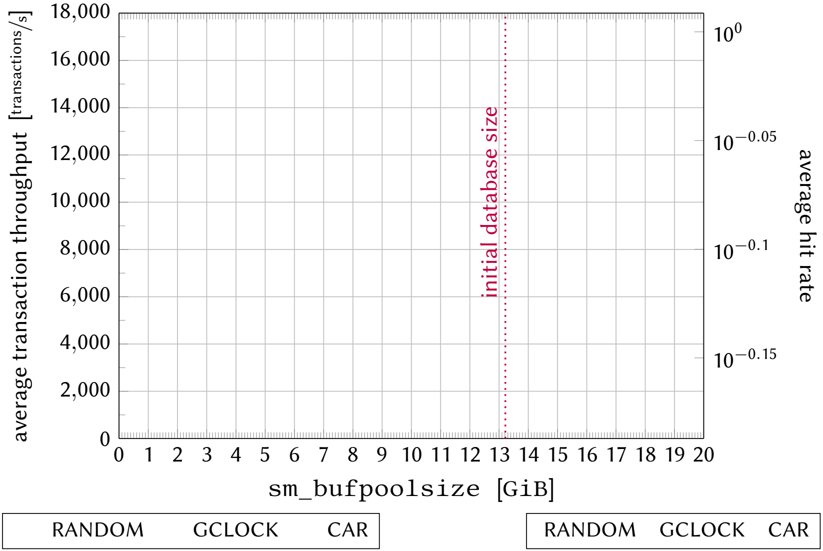
Buffer Pool Without Pointer Swizzling (TPC-C)



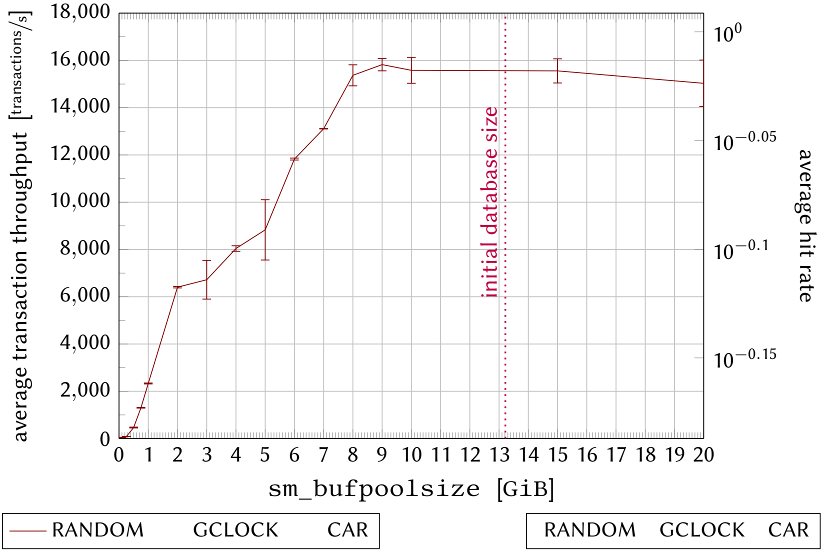
Buffer Pool Without Pointer Swizzling (TPC-C)



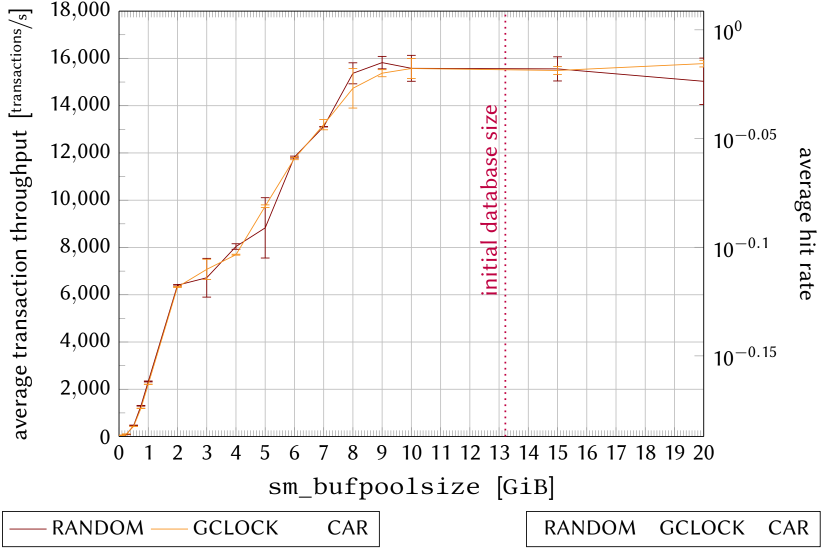
Buffer Pool With Pointer Swizzling (TPC-C)



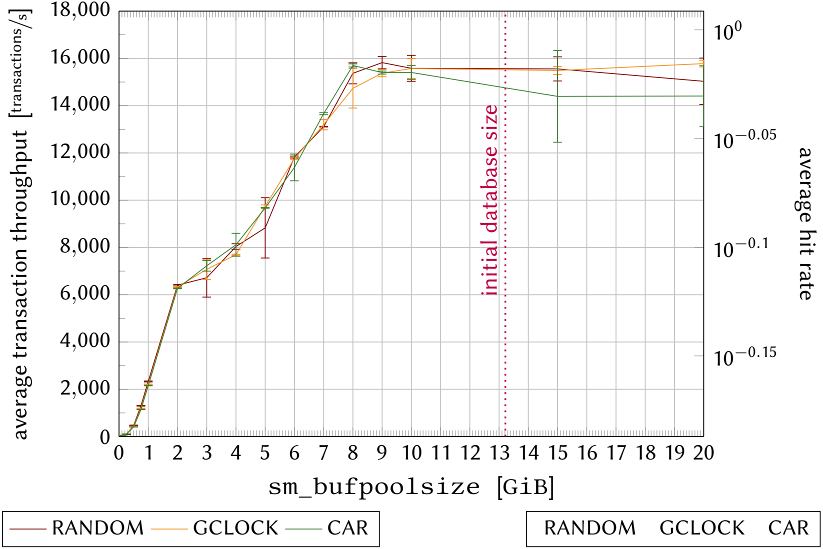
Buffer Pool With Pointer Swizzling (TPC-C)



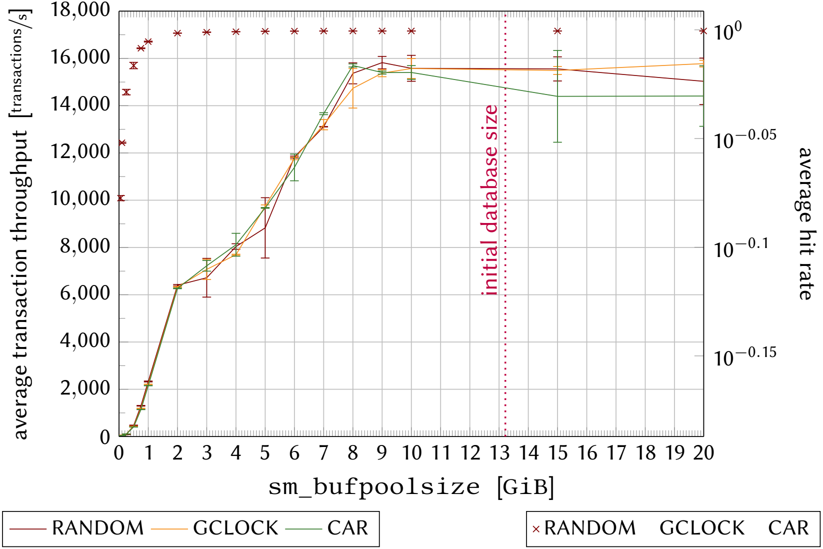
Buffer Pool With Pointer Swizzling (TPC-C)



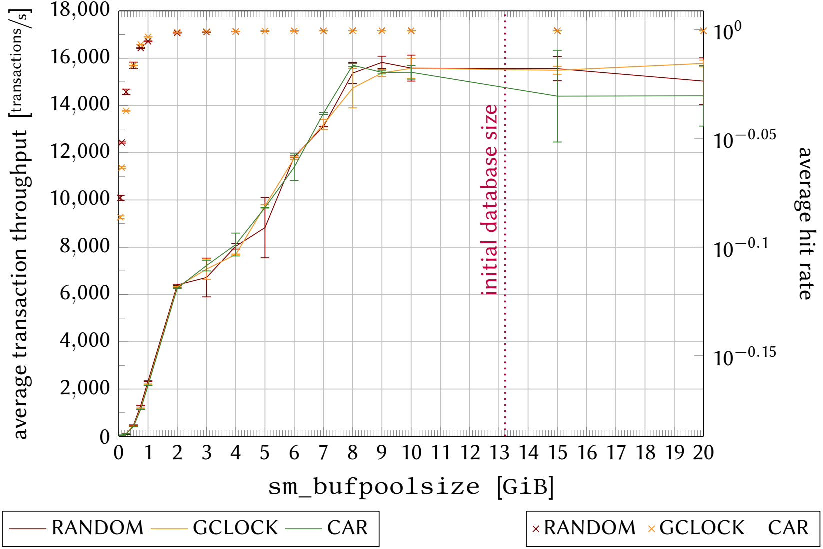
Buffer Pool With Pointer Swizzling (TPC-C)



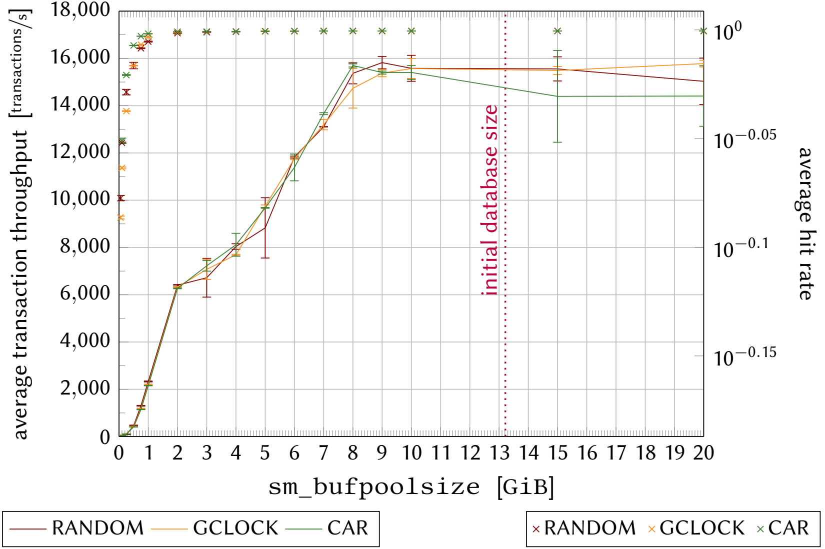
Buffer Pool With Pointer Swizzling (TPC-C)



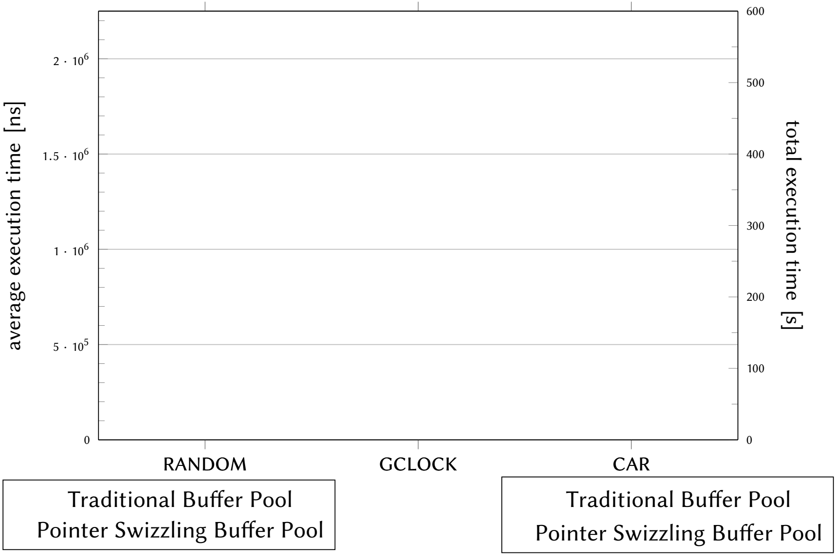
Buffer Pool With Pointer Swizzling (TPC-C)



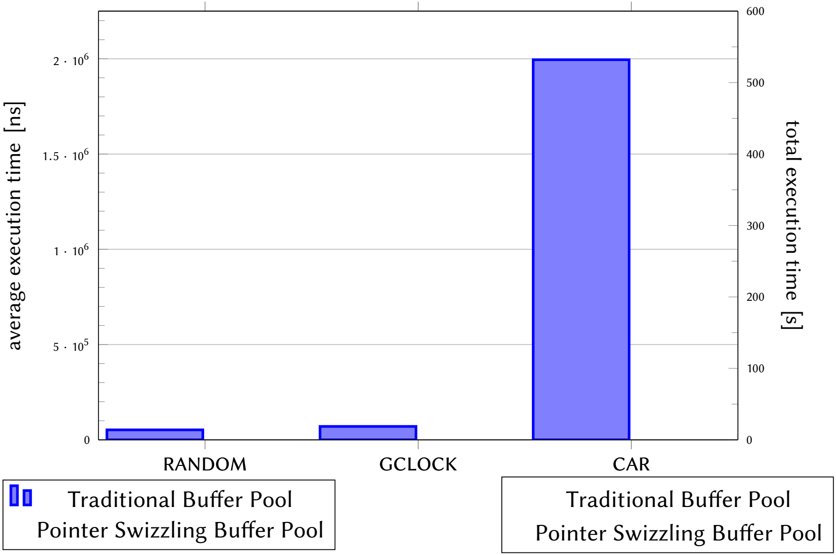
Buffer Pool With Pointer Swizzling (TPC-C)



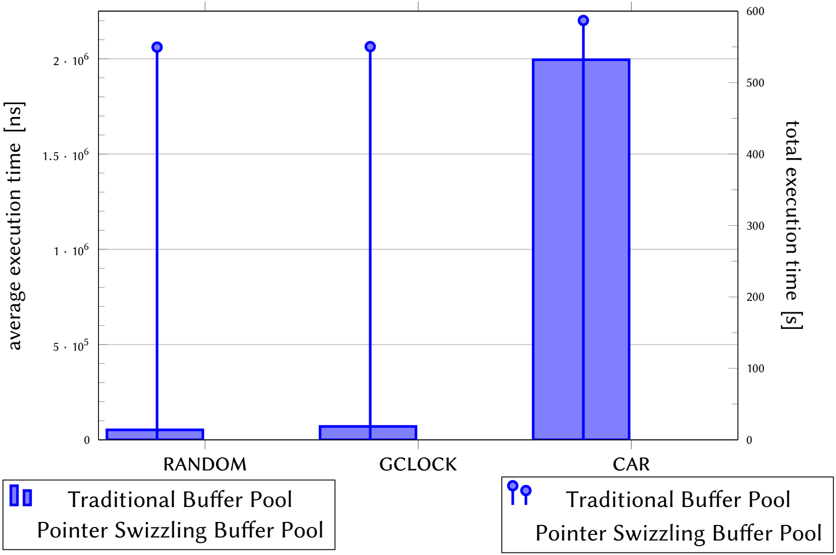
Operation Performance



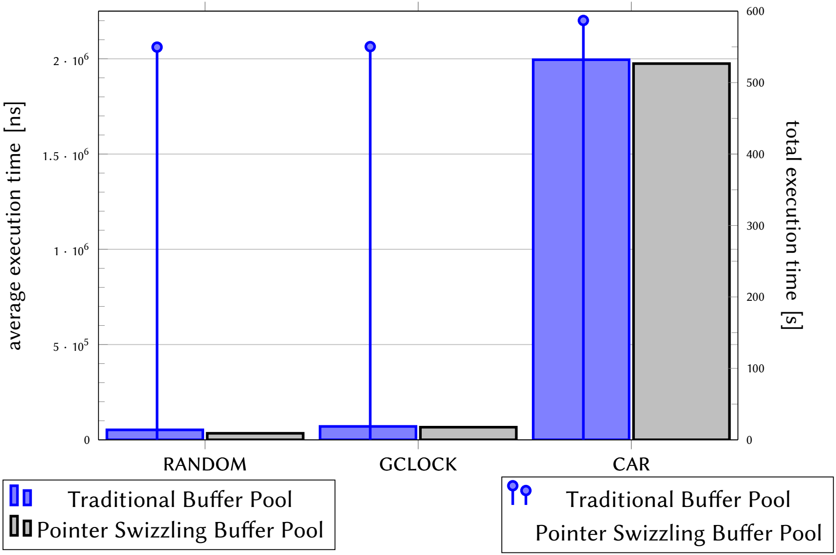
Operation Performance



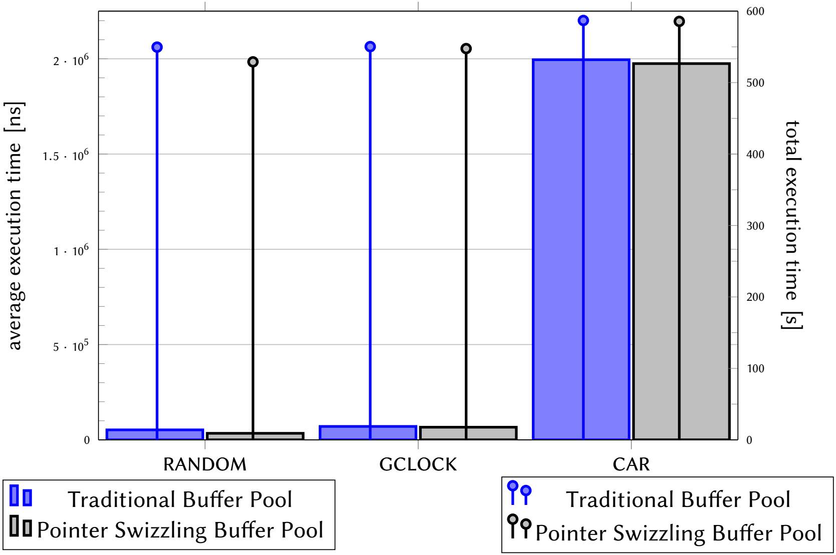
Operation Performance



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.

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

Theo Härder and Erhard Rahm. *Datenbanksysteme - Konzepte und Techniken der Implementierung.* 2001. ISBN: 978-3-642-62659-3.

J. Eliot B. Moss. “Working with Persistent Objects: To Swizzle or Not to Swizzle”. Aug. 1992.

References II

- Lucas Sauer Caetano Lersch, Theo Härder, and Goetz Graefe. “Update propagation strategies for high-performance OLTP”. Aug. 14, 2016.
- Seth J. White and David J. DeWitt. “QuickStore: A High Performance Mapped Object Store”. Oct. 1995.

Your Turn to Ask ...