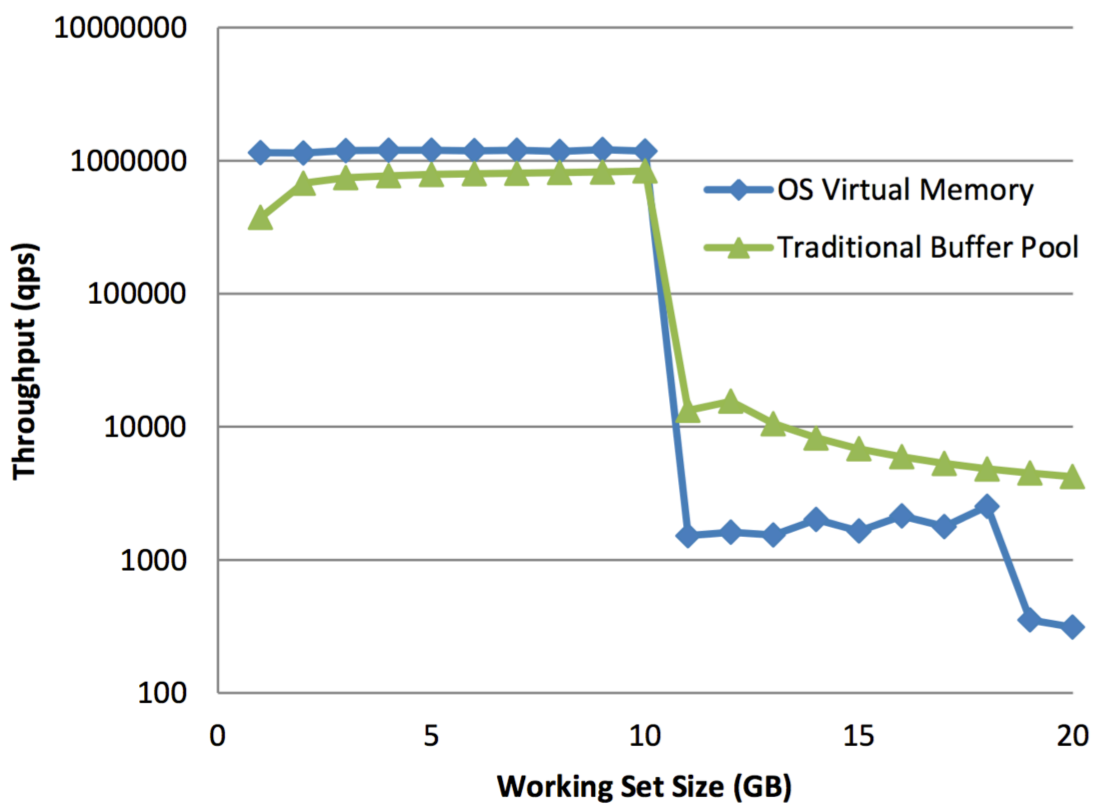
In-Memory Performance for Big Data

Goetz Graefe, Haris Volos, Hideaki Kimura, Harumi Kuno, Joseph Tucek, Mark Lillibridge, Alistair Veitch

VLDB 2014,

presented by Nick R. Katsipoulakis

A Preliminary Experiment



- B-Tree nodes
- 10GB of Memory
- Buffer pool
  - Disk pages
- In-Memory
  - Direct pointers between nodes

Related Work – In-memory databases

- Workload fits
  - e.g. Oracle TimesTen, SQL Server Hekaton, MonetDB, SAP Hana, VoltDB etc.
- Workload does not fit
  - OS VM layer
    - Poor eviction decisions
    - Data integrity issues
  - Compression (frozen data)
  - Identify hot and cold data
    - Stoica and Ailamaki work on VoltDB
    - Decrease statistic cost
    - Anti-Caching

Motivation

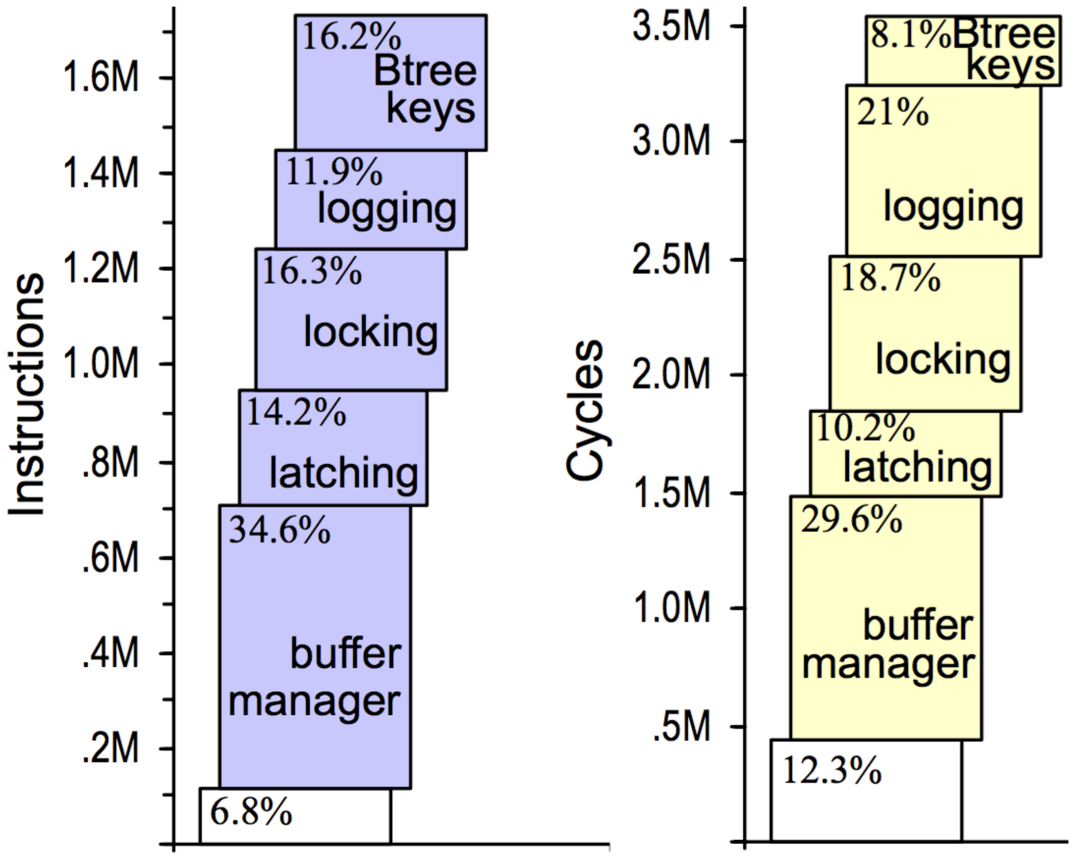
- Combine best of both worlds
  - Near in-memory performance (workload fits)
  - Buffer-pool performance (workload does not fit)
- Buffer Pool
  - Benefits
    - large working sets
    - support for write-ahead logging
    - Insulation from cache-coherence issues
  - Drawbacks:
    - Level of indirection

But first, the System Model

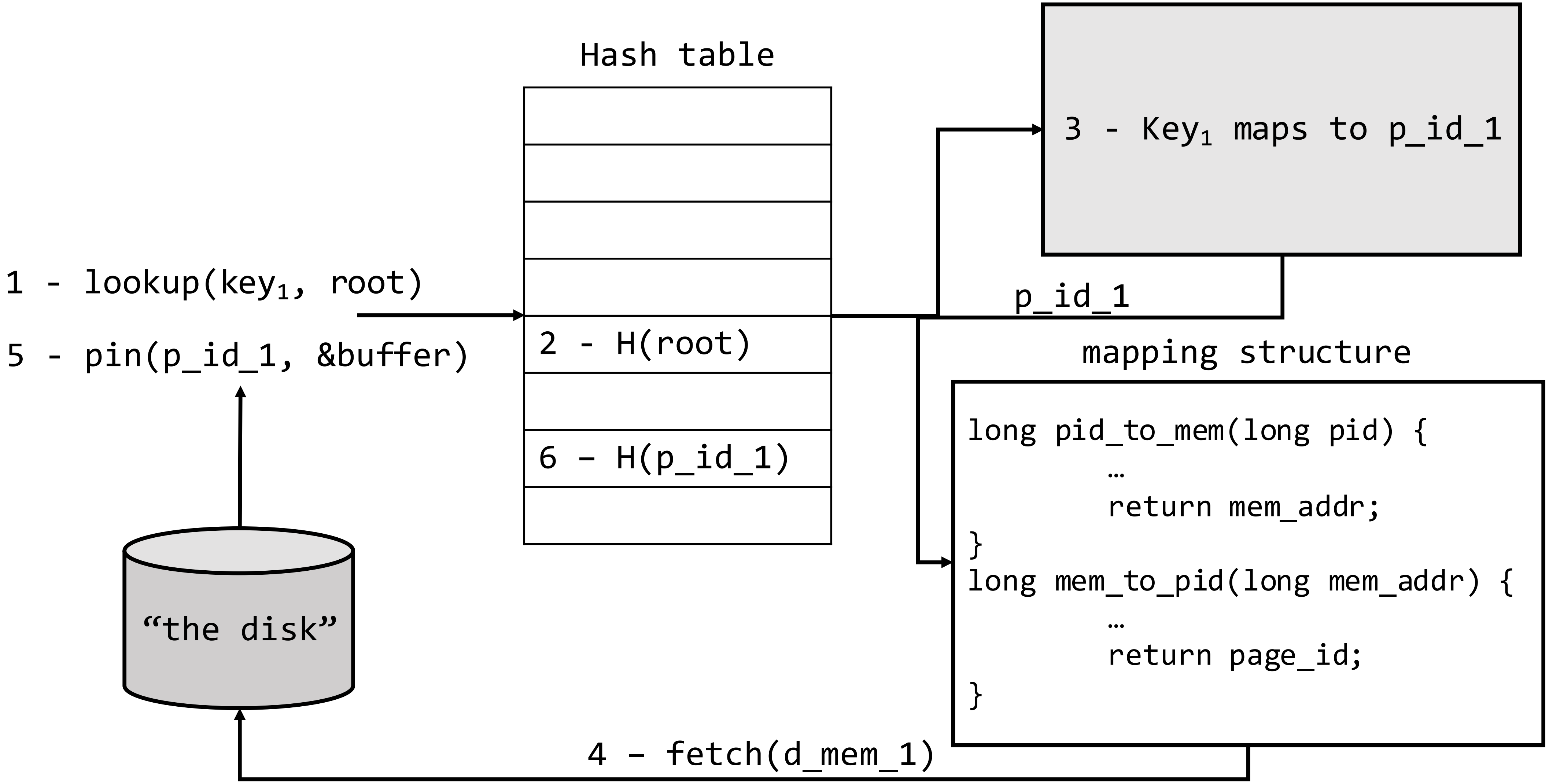
- Transactional Storage Manager
  - ACID guarantees
  - Modern hardware (multi-core architecture)
- Data Storage
  - B-Tree (one node to one disk page)
  - Leaf nodes maintain data
- Buffer pool
  - copies of pages
- Latches and Locks
- Write-ahead Logging

A flashback at Harizopoulos' et al. observation

- Dataset in-memory
- Observations
  - Buffer manager takes up ~30% of both instructions and cycles total
- Idea
  - Faster buffer pool
  - Correctness guarantees



A closer look – the source of all evil



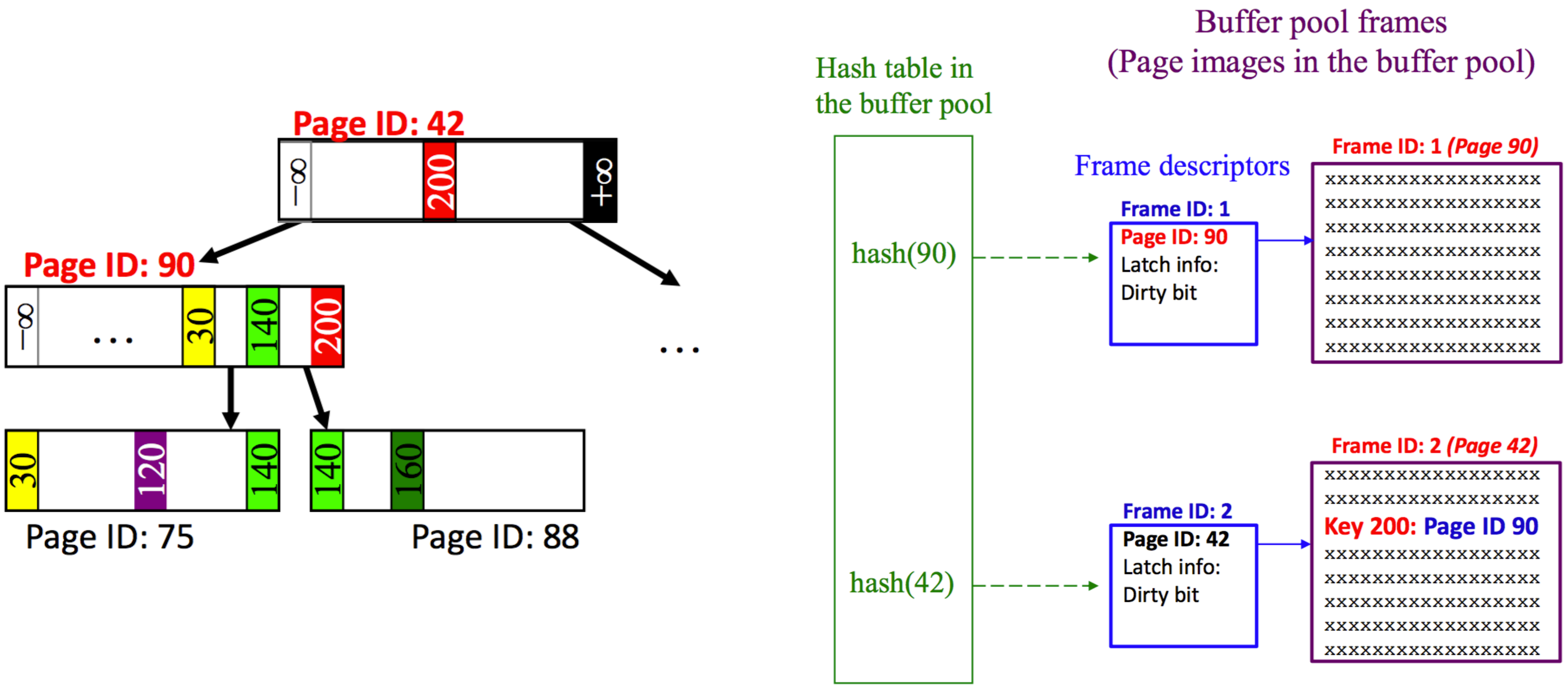
Their proposal for improving the buffer pool

- Decrease buffer pool overhead
  - Remove the accesses to the common mapping structure
- Pointer swizzling
  - lazy
  - not all page-IDs are swizzled
- Contribution
  - Buffer pool re-design. Support pointer (un-)swizzling
  - Eviction policy

But, wait. What about Virtual Memory?

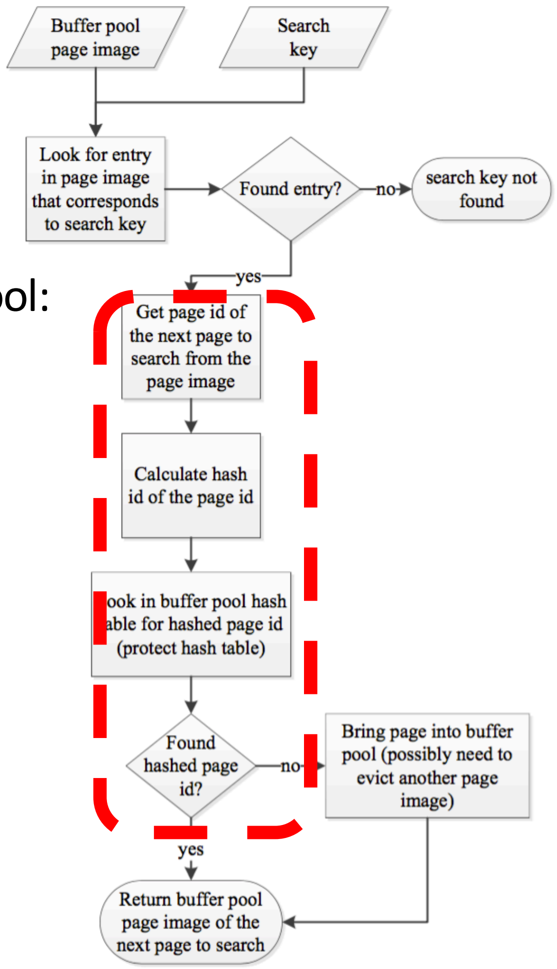
- Correctness requirements might be violated
  - write too early
    - e.g. write a page before the log has concluded
  - write too late
    - e.g. miss a checkpoint because a dirty page have not been written to the backing store
  - recycle non-persistent logs
    - e.g. log page is recycled by the OS VM manager, but, changes have not yet been persisted to actual storage
- `msync()` & `mlock()` do not support:
  - asynchronous read-ahead
  - concurrent multiple writes

A look at traditional B-Tree Nodes and the buffer pool

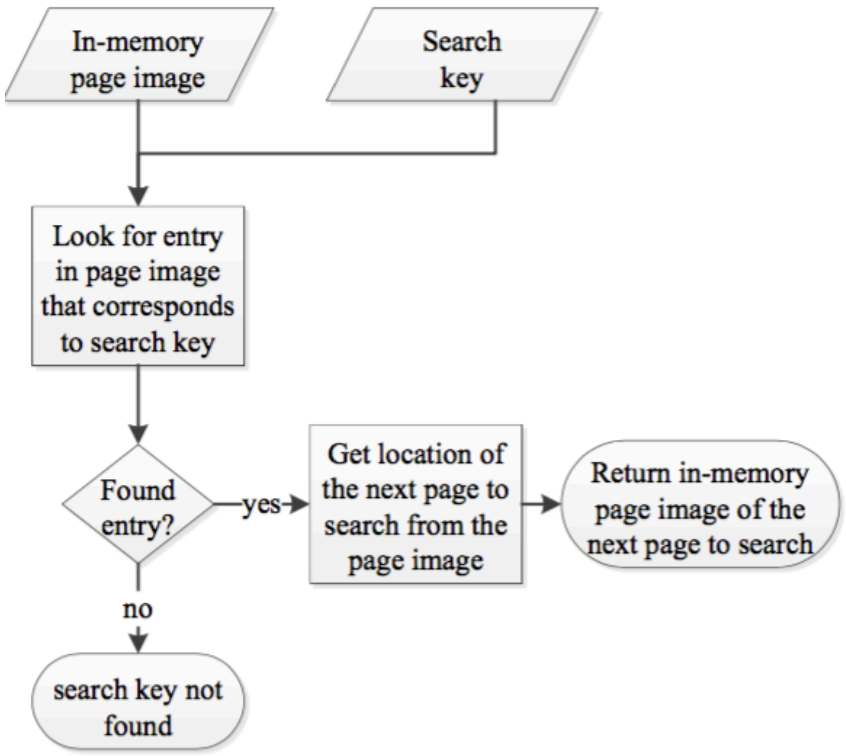


Flow-charts for locating pages

Traditional buffer pool:

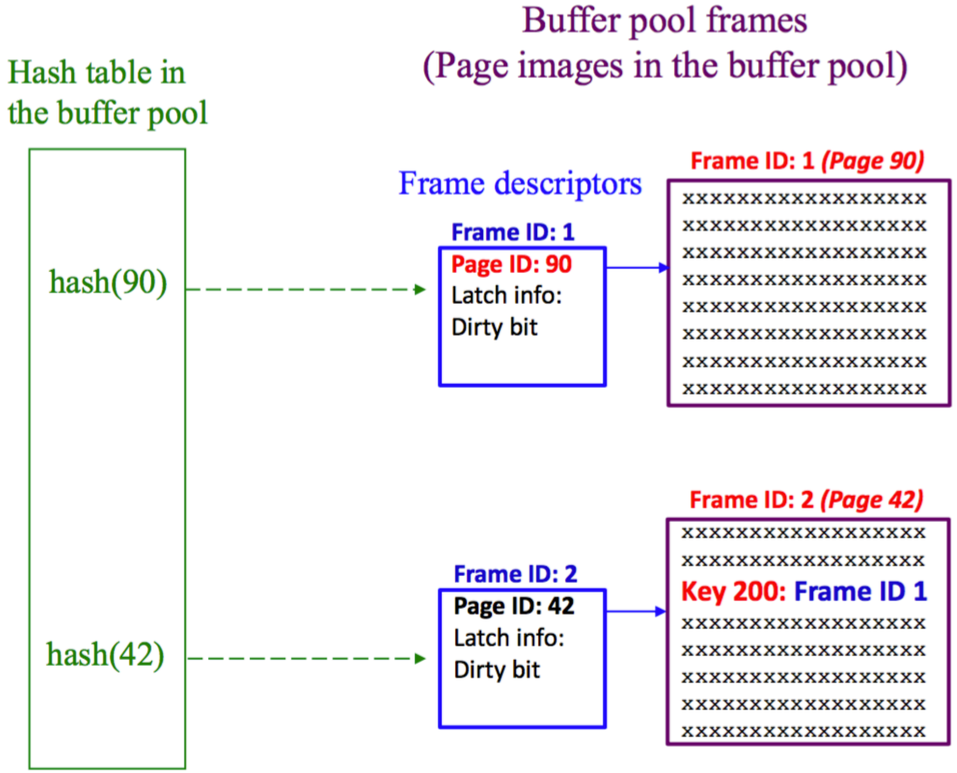


In-memory:

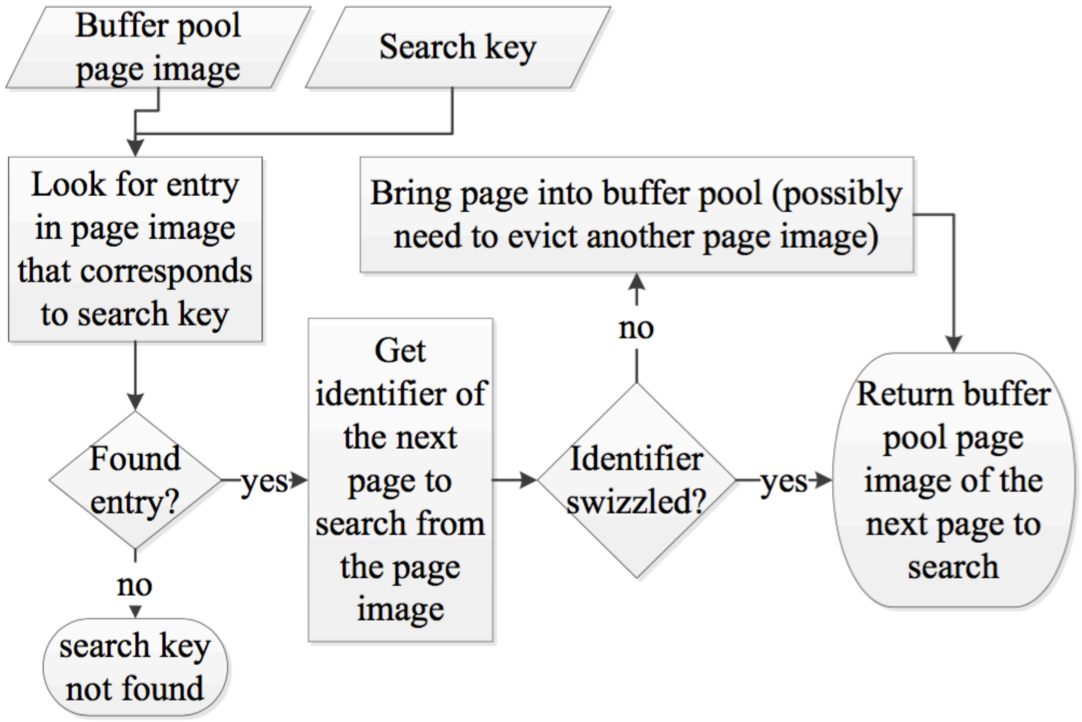


Proposed buffer-pool design with pointer swizzling

Buffer Pool



Flow-chart



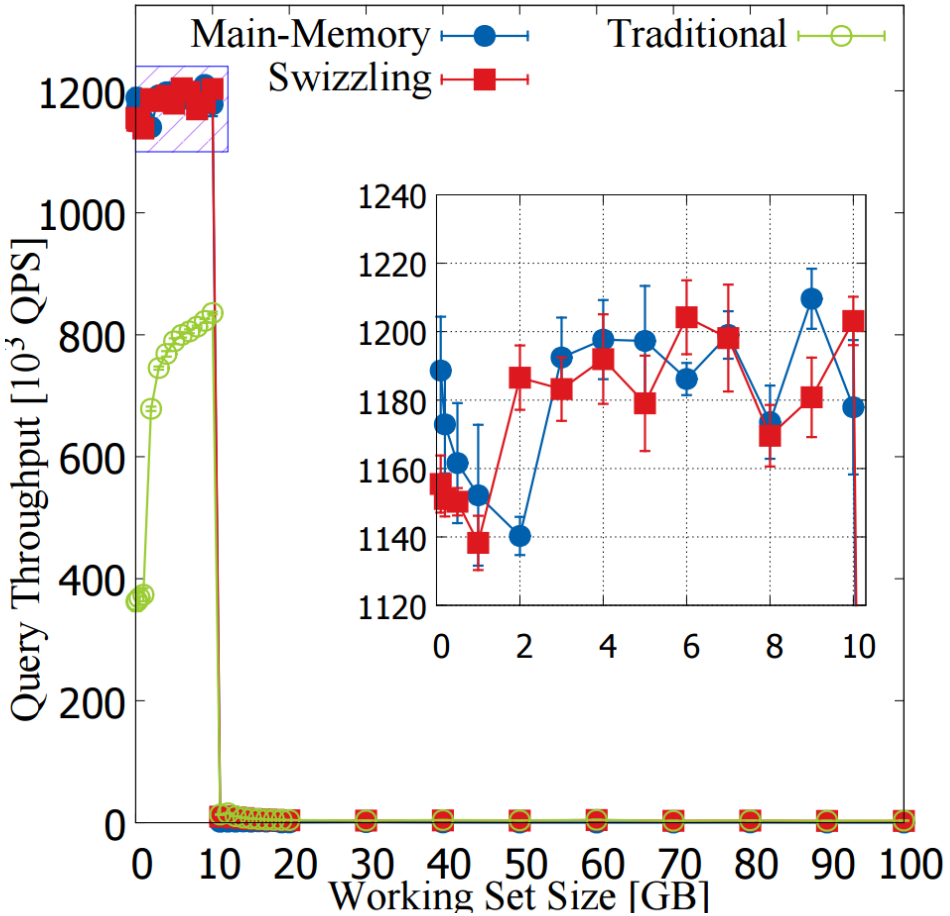
Proposed design with swizzling

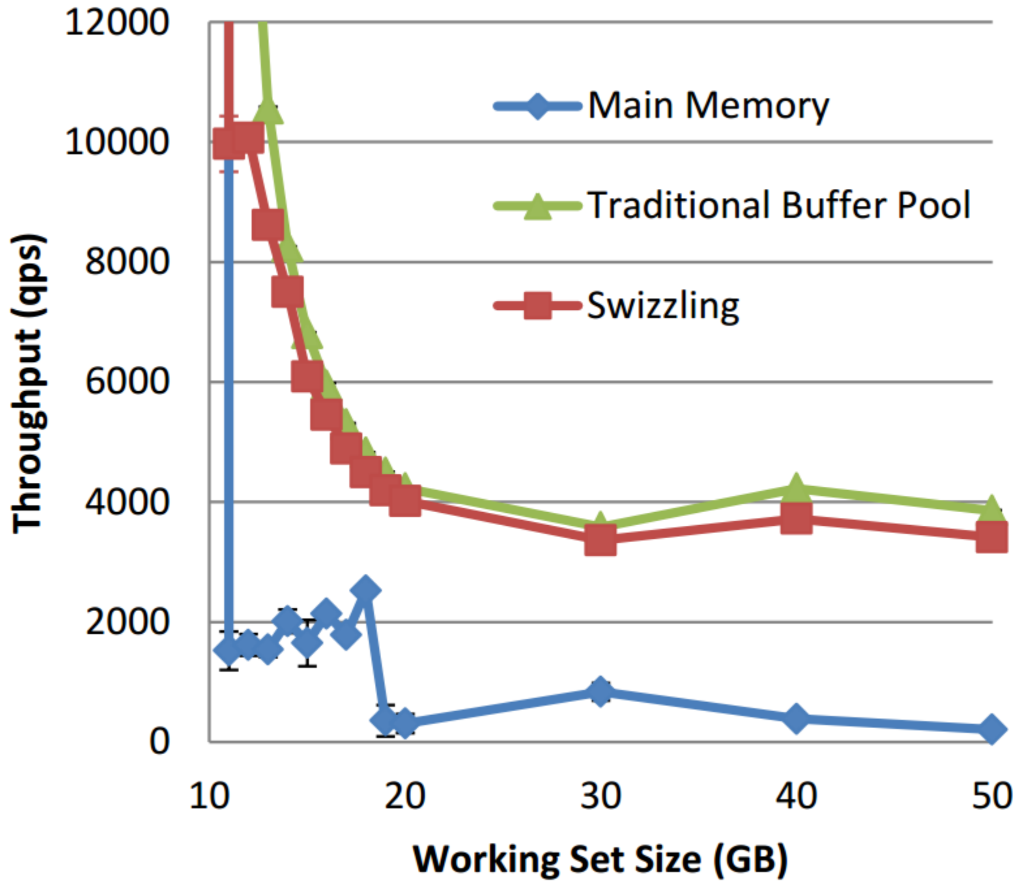
- Pointers are swizzled one at a time
  - Not all pointers are swizzled
- Pool eviction
  - Generalized clock scheme
  - Sweep B-Tree using depth-first search
    - Pages with no recent usage are un-swizzled unless they contain swizzled parent-to-child pointers
- Child-to-parent pointers
  - Expedite un-swizzling
  - Include parent-frame in metadata

Experimental Evaluation

- Shore-MT
  - pointer-swizzling buffer pool
  - traditional buffer pool
  - in-memory
- Testbed: Intel Xeon (4 socket, 24 cores), 256 GB Ram, RAID-10 with 10K rpm drives
- 10GB Buffer pool with O_DIRECT enabled
- 100GB database size
  - Key size 20 bytes
  - Value size 20 bytes

Buffer pool performance – Query performance





Buffer pool performance – Insert performance

- 24 threads
- 50 million records
  - initially 10 million records
- Randomly chosen keys

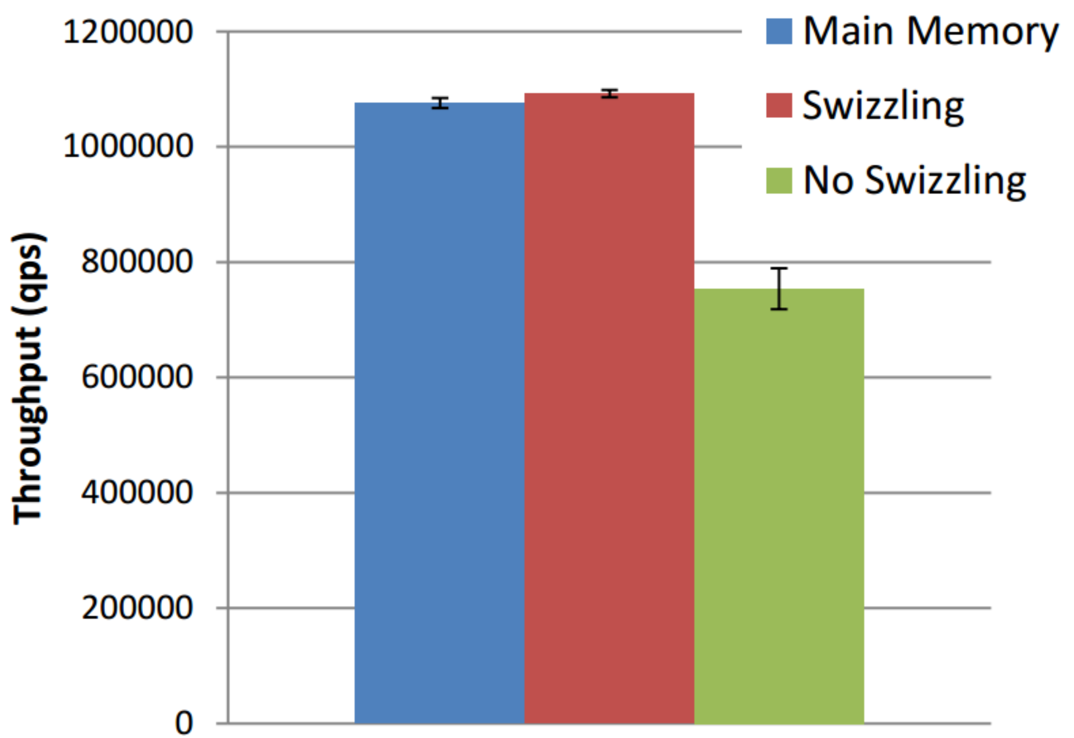
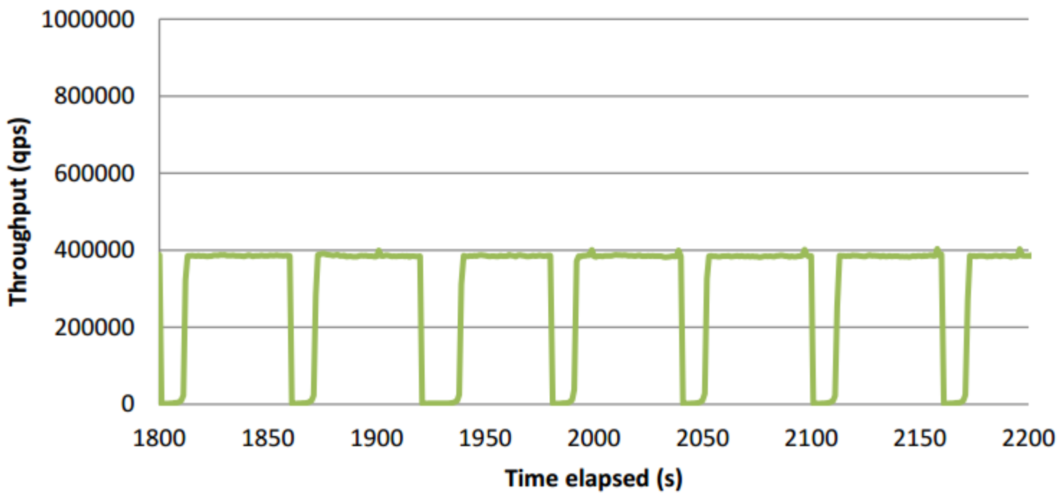
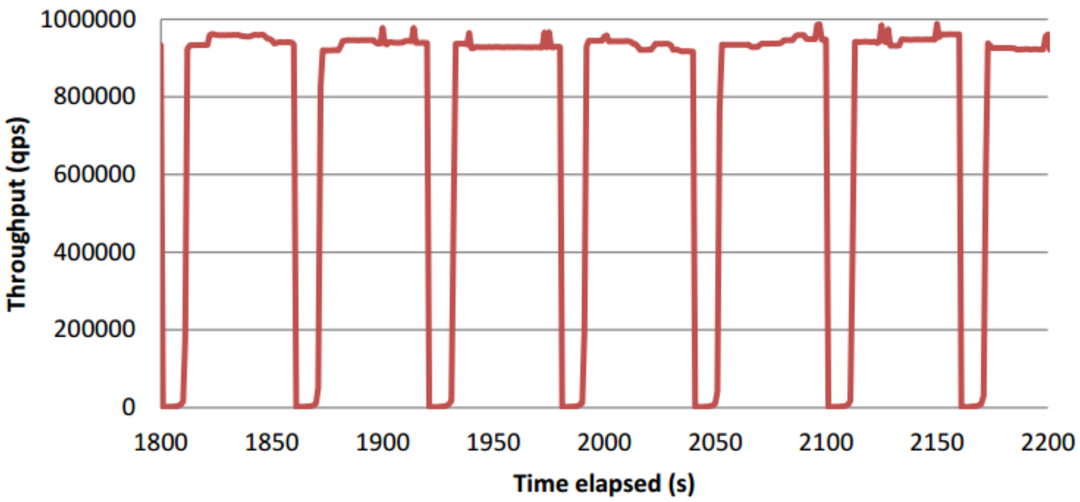


Figure 12: Insertion performance.

Buffer pool performance - Drifting working set

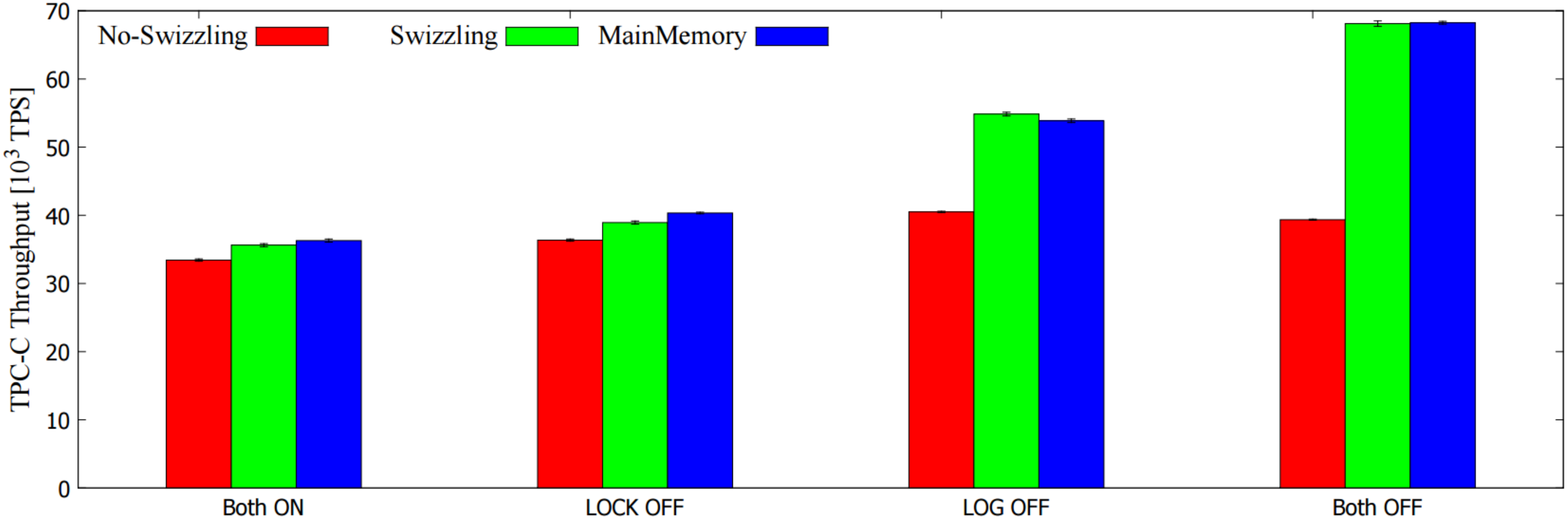


(a) Traditional buffer pool



(b) Buffer pool with swizzling

TPC-C Benchmark



Conclusion - Thoughts

- A way to combine the best-of-both worlds
  - In-memory performance (workload fits)
  - Buffer pool performance (workload does not fit)
- Questions
  - Was it really the “mapping-data-structure” the bottleneck?
  - If a NVM database was used, is pointer-swizzling the answer? Do we still need a buffer manager, or do we need a general “memory manager”?
- Thank you!