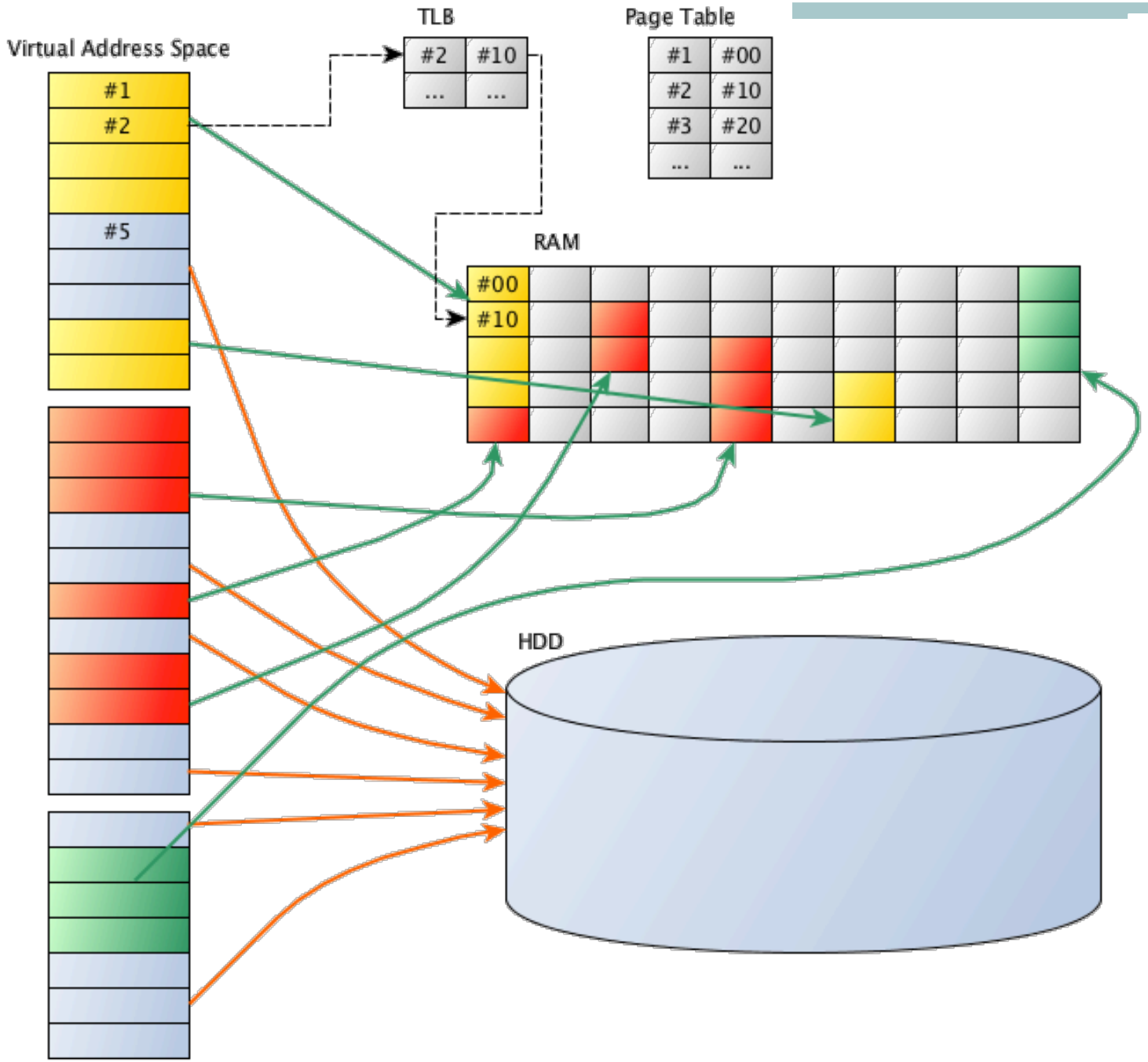
OS Paging and Buffer Management

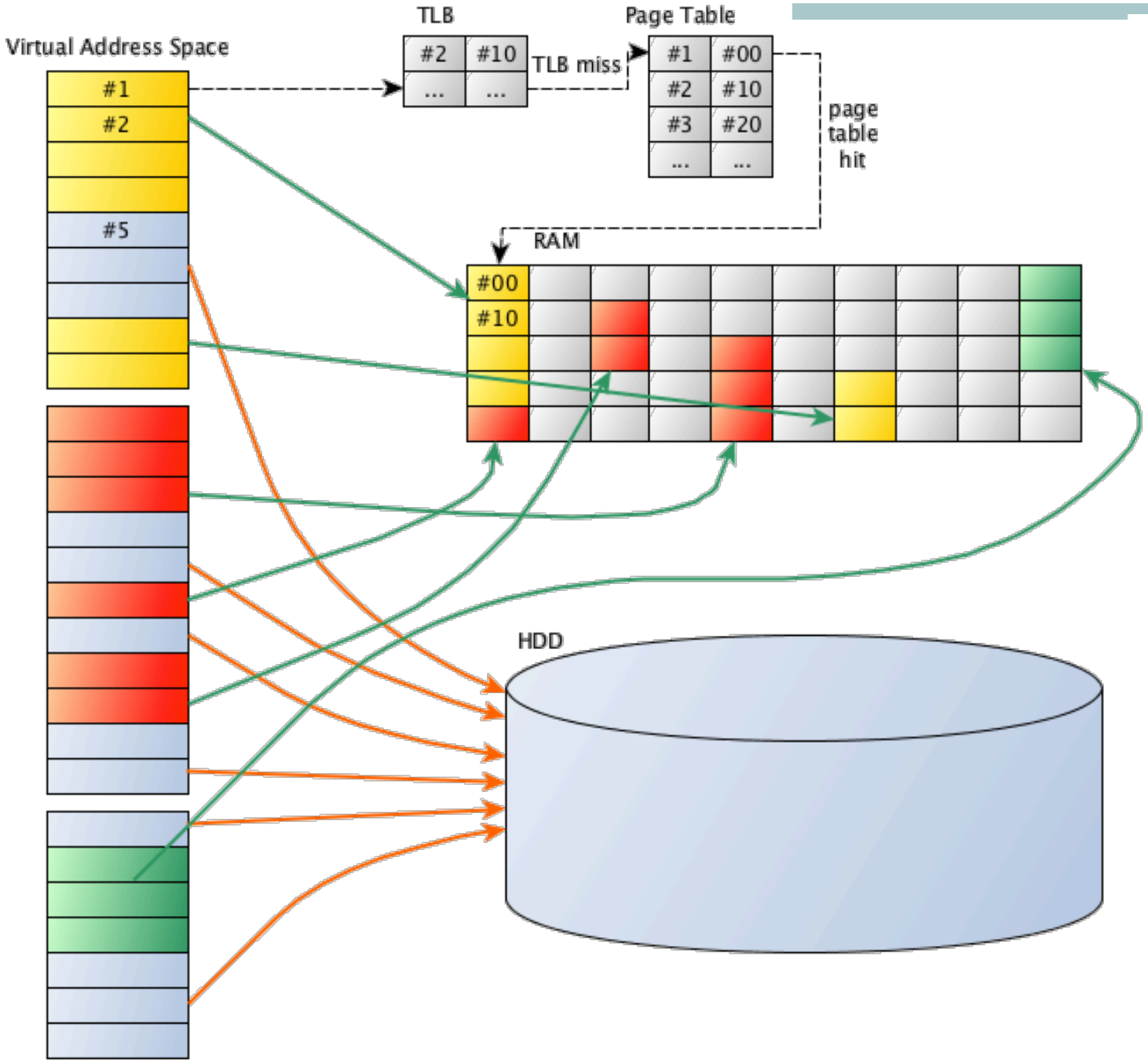
Using the Virtual-Memory Paging Mechanism to Balance Hot/Cold Data in Memory and HDD

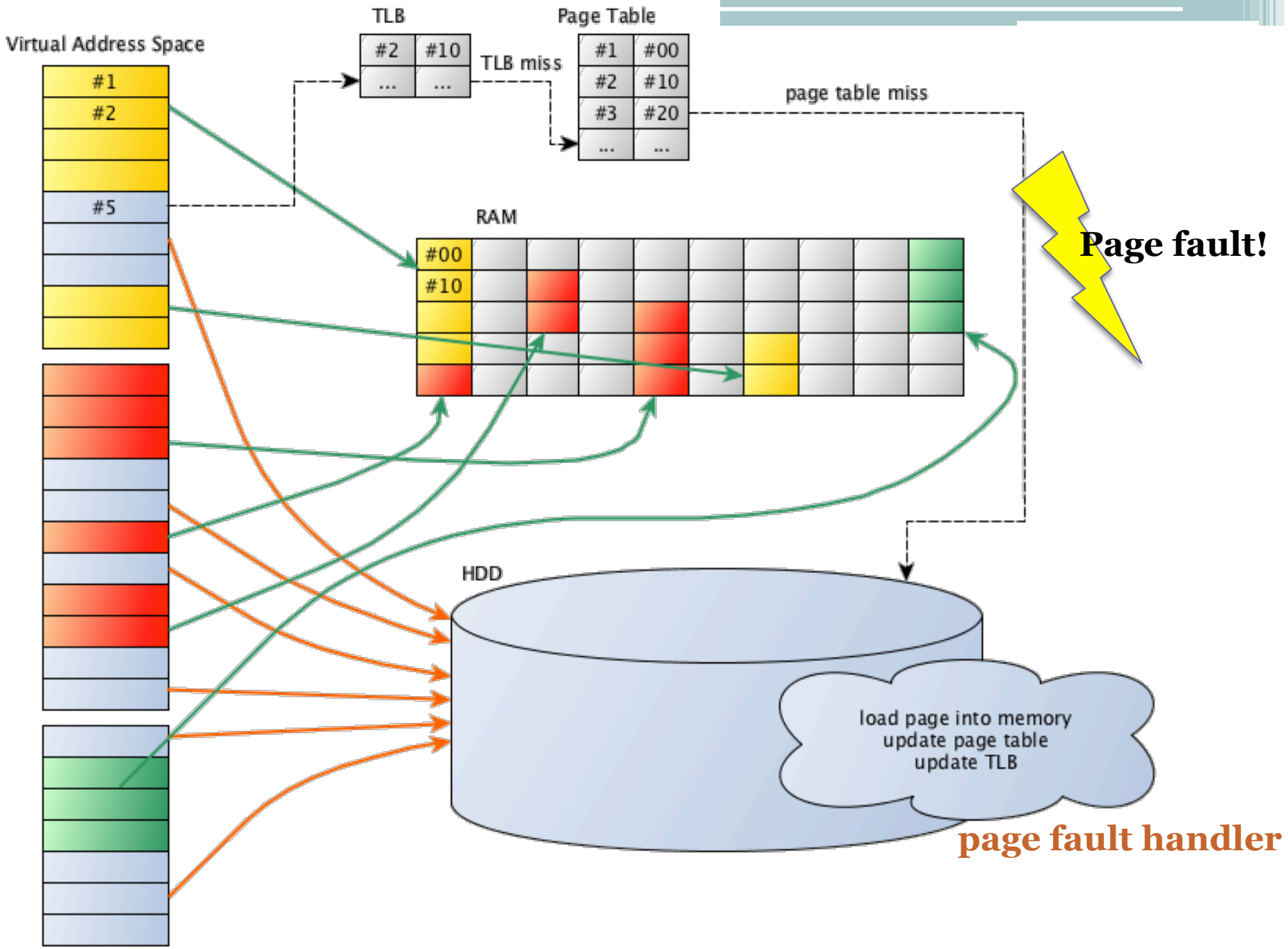
Jana Lampe

Content Overview

- Motivation
  - Virtual paging and its application in the DB context
  - New hardware: SSDs
  - OLTP workloads
- Stoica et al.‘s approach
- “Anti-Caching”

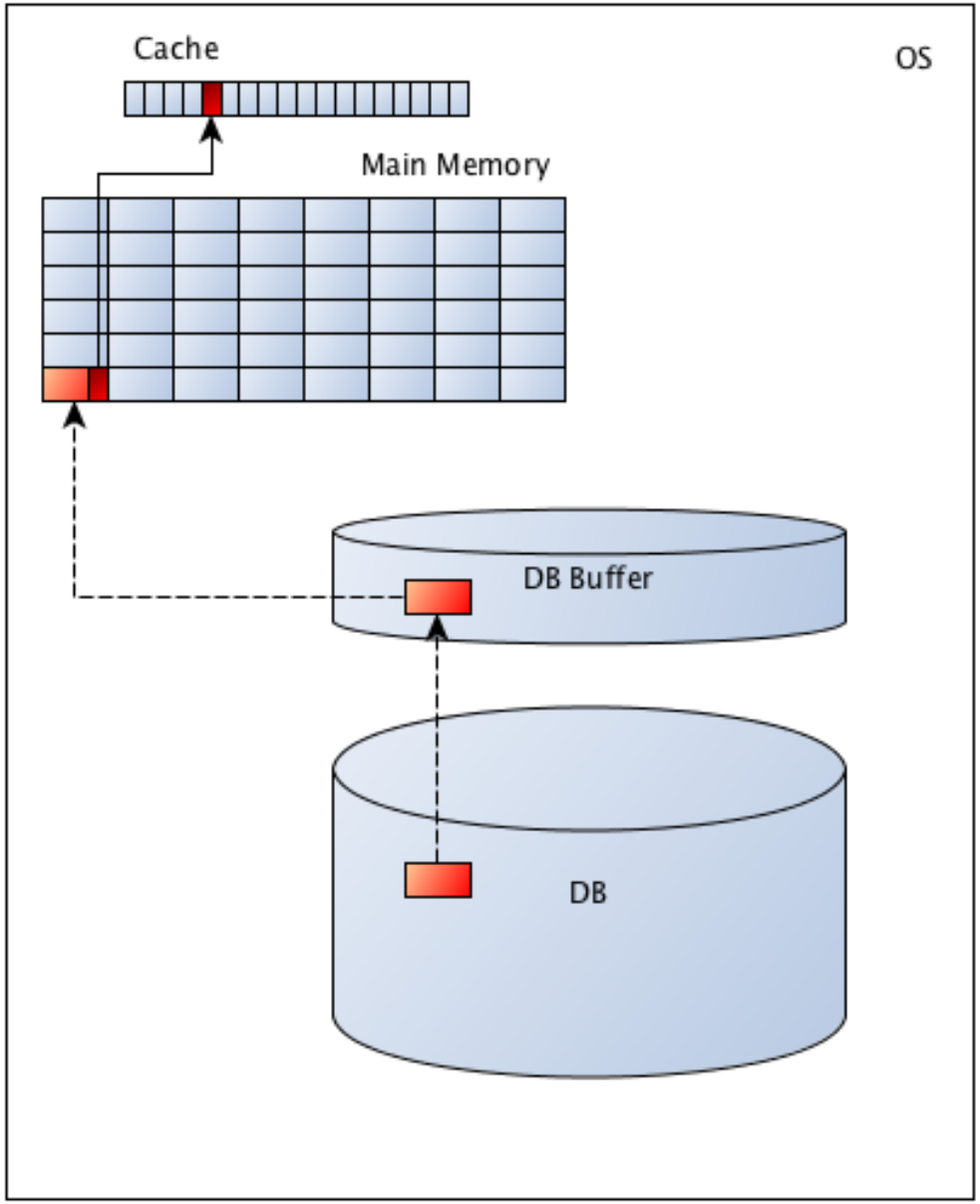

TLB
Page Table
Virtual Address Space
#2
#10
...
...
#1
#00
#2
#10
#3
#20
...
...
#1
#2
#5
RAM
#00
#10
HDD


TLB
Page Table
Virtual Address Space
#2
#10
...
...
TLB miss
#1
#00
#2
#10
#3
#20
...
...
page table hit
#1
#2
#5
RAM
#00
#10
HDD

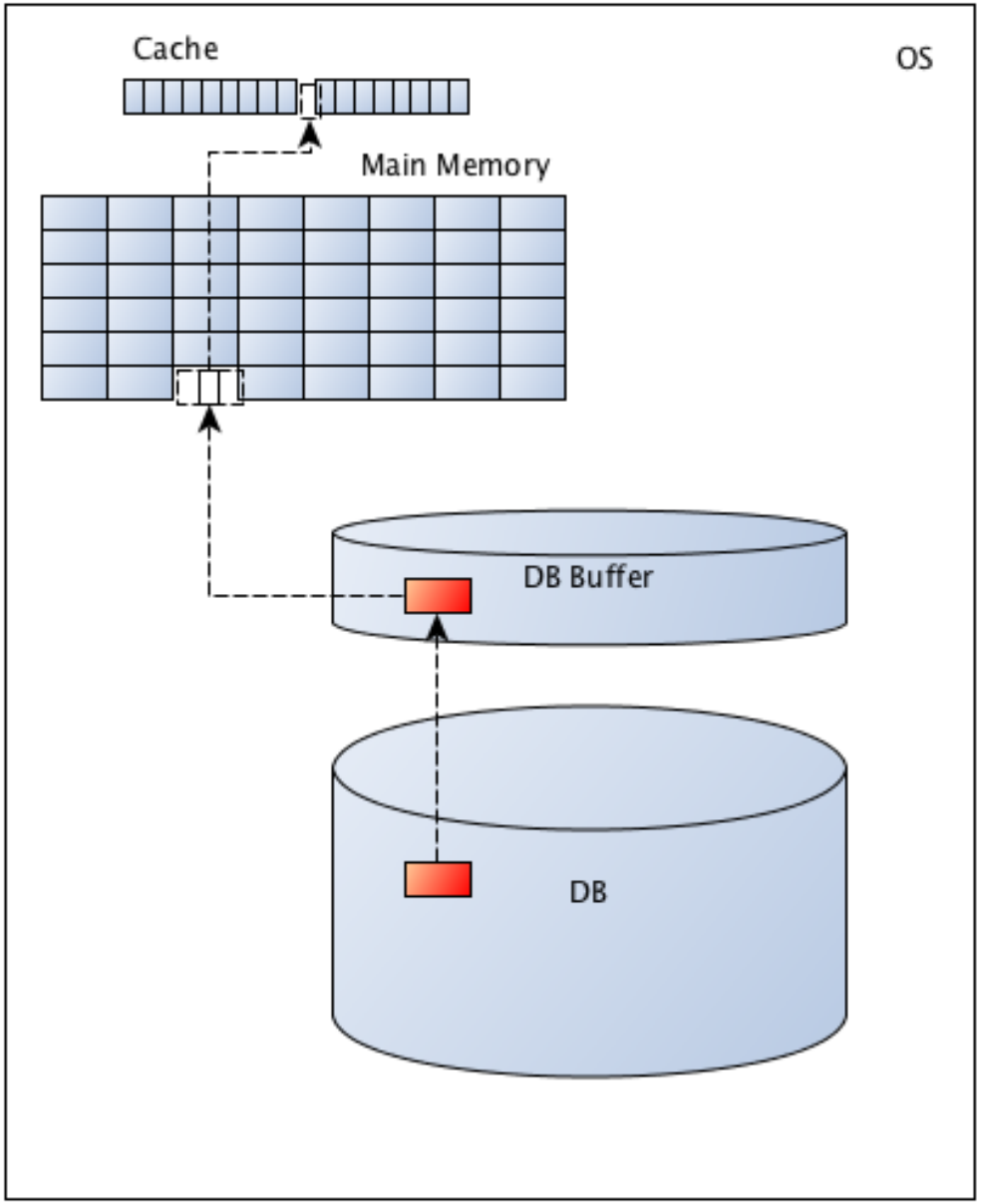

TLB
Page Table
Virtual Address Space
#2
#10
TLB miss
#1
#00
#2
#10
#3
#20
page table miss
#1
#2
#5
RAM
#00
#10
Page fault!
HDD
load page into memory
update page table
update TLB
page fault handler

Application to Traditional DBs

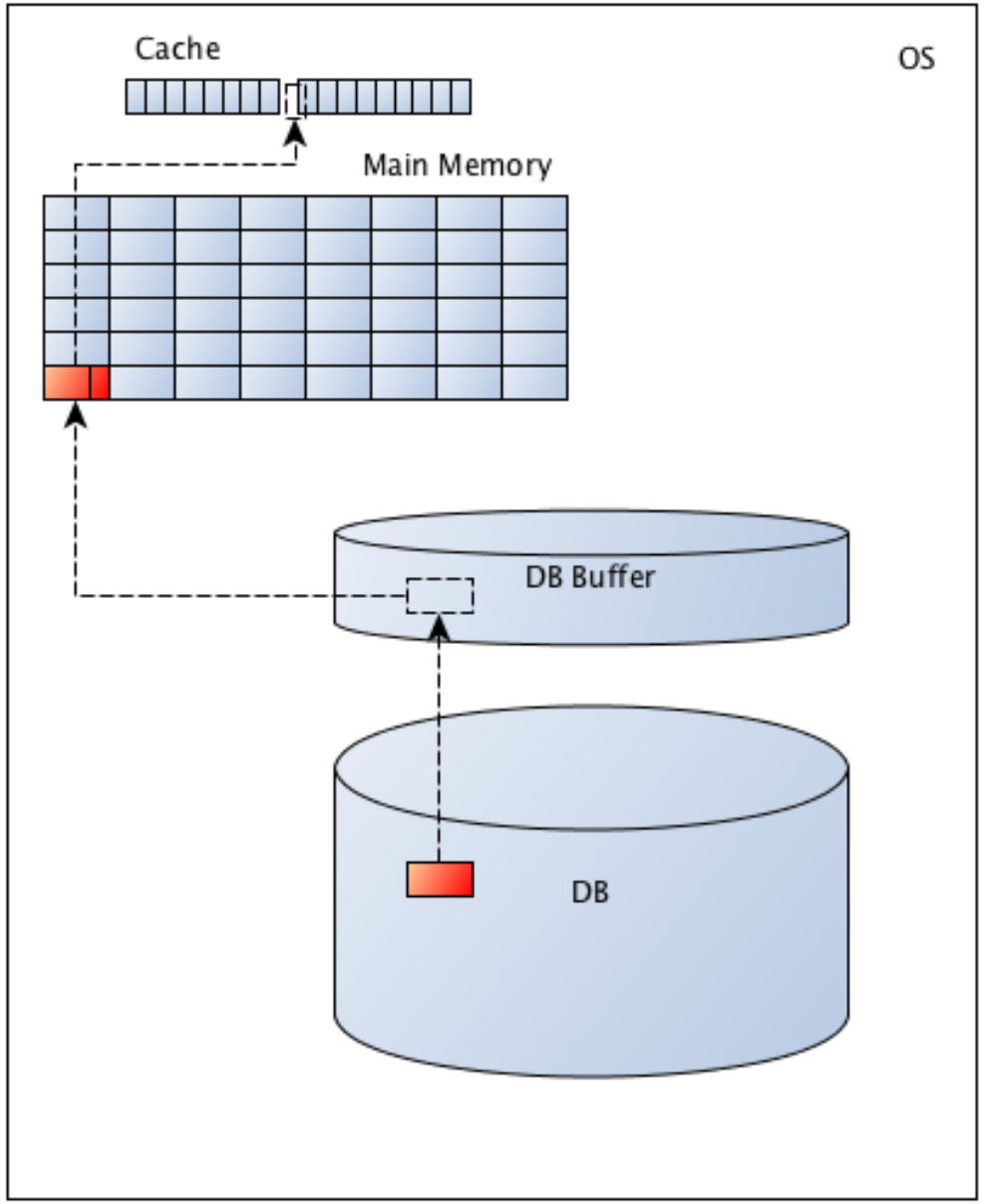
- DBMS is embedded in OS environment like an application program
- Problem: program code and DB buffer are subject to OS paging
  - DB buffer depends on transactional workload
  - Main memory is managed by OS
  - Different page replacement strategies?



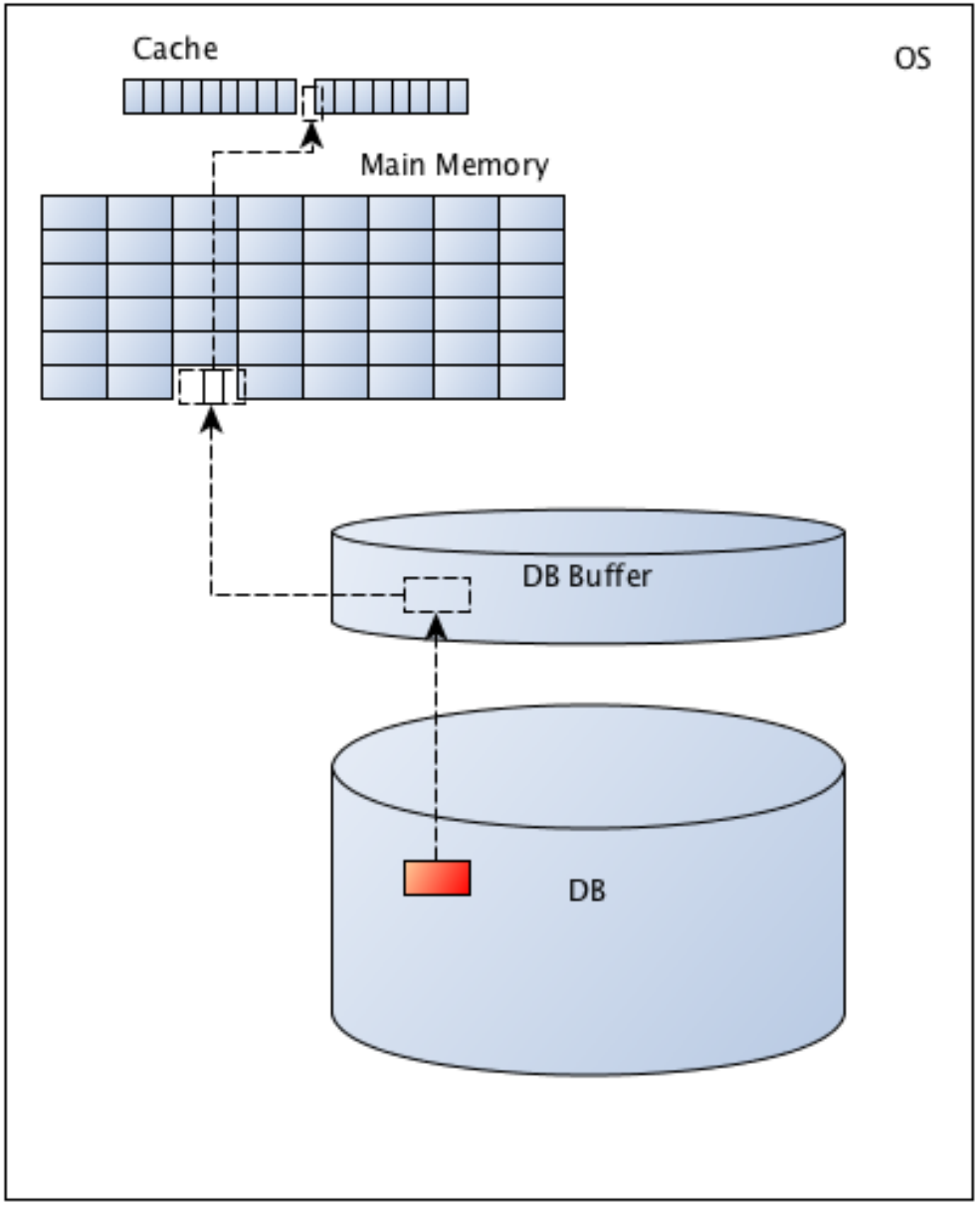
Traditional DB Buffer Management



Page Fault



Database Fault



Double Page Fault

2007: The End of an Architectural Era (Stonebraker et al.)

Hardware Trends: ~2004 ⇒ 2014

Magnetic disks

Capacity	400 GB	**x 15**	6 TB
GB/$	0.05	**x 600**	30
IOPS	200	**x 1**	200

Solid state drives

Capacity	16 GB	**x 30**	480 GB
GB/$	0.0005	**x 3,000**	>1.5
IOPS (4KB read)	1,000 (SCSI)	**x 1,000**	1,000,000+ (PCIe)
			5,000+ (SATA)
IOPS (4KB write)	50 (SCSI)	**x 10,000**	500,000+ (PCIe +RAM)

Phase-change memory

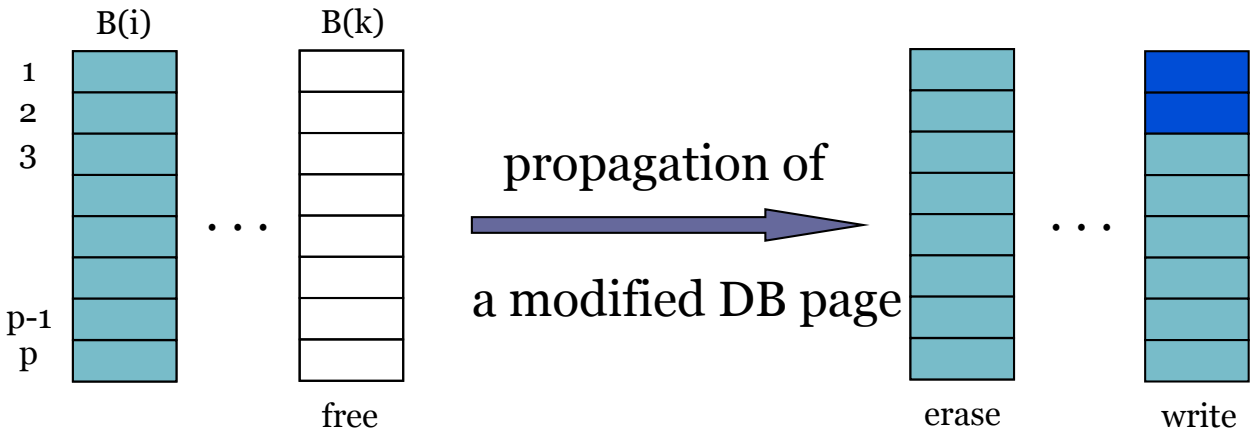
Capacity			1 GB chip (20-nm)
IOPS (64B read)			20,000,000+ (1 chip)
IOPS (64B write)			1,000,000+ (1 chip)

Flash Usage in DB Servers?

- Guarantees persistent data with (almost) zero-energy needs when idle or turned off
- A flash block B, much larger than a disk block,
  - contains p (typically 32 – 128) fixed-size flash pages with 512 B – 2 KB
  - NAND logic does not enable direct update of pages → erasure
- Reads of individual flash pages
- Update of flash pages not possible;

  only overwrites of entire blocks where erasure is needed first

Flash Usage in DB Servers?

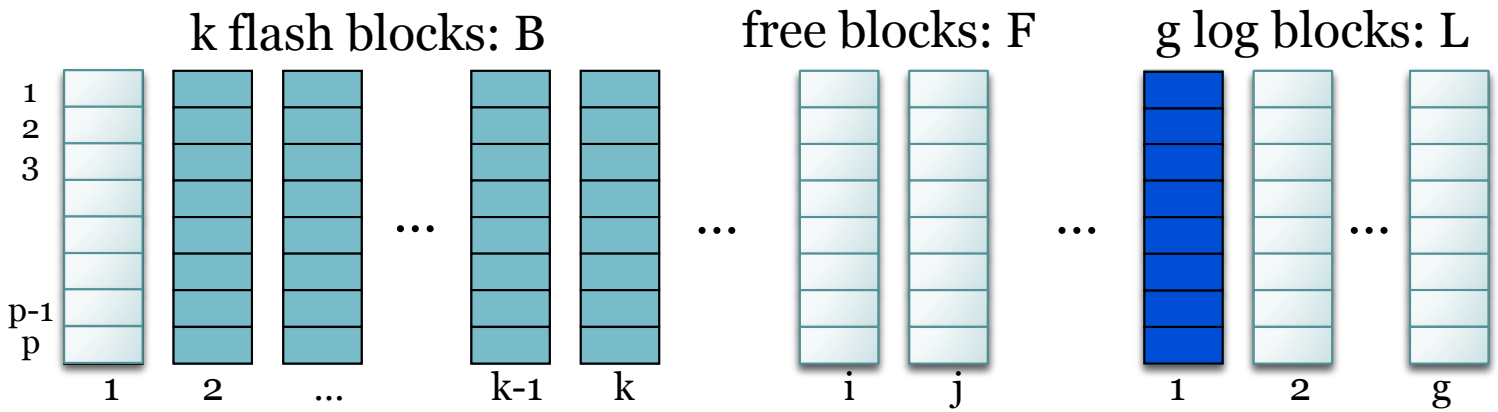


Wear leveling automatically allocates a new flash block and preserves cluster property

Built-in Wear Leveling

- Flash-internal write optimization
- Simplest form of mapping: 1:1 - block level

Metadata (flash directory) must be in RAM



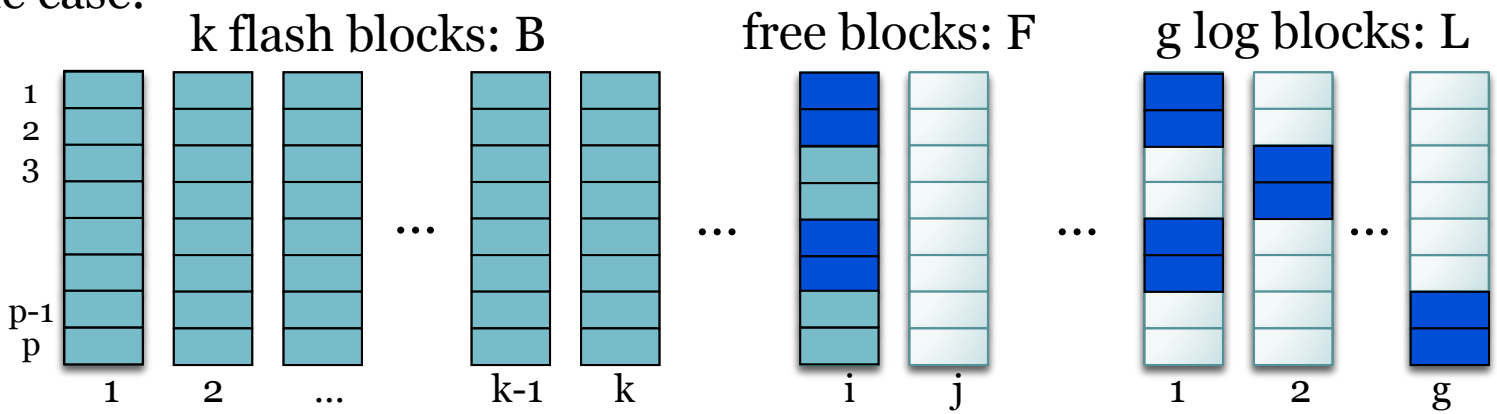
Switch: L1 becomes B1, Erase old B1 → 1 erasure

Built-in Wear Leveling

- Simplest form of mapping: 1:1 - block level

Metadata (flash directory) must be in RAM

Some case:



Merge: L1 and B1 to Fi

Erase B1 and L1 → 2 erasures

Other forms of mapping: n:1 (n:m) - block level (page level)

Merge of n flash blocks and one log block to Fi → n+1 erasures

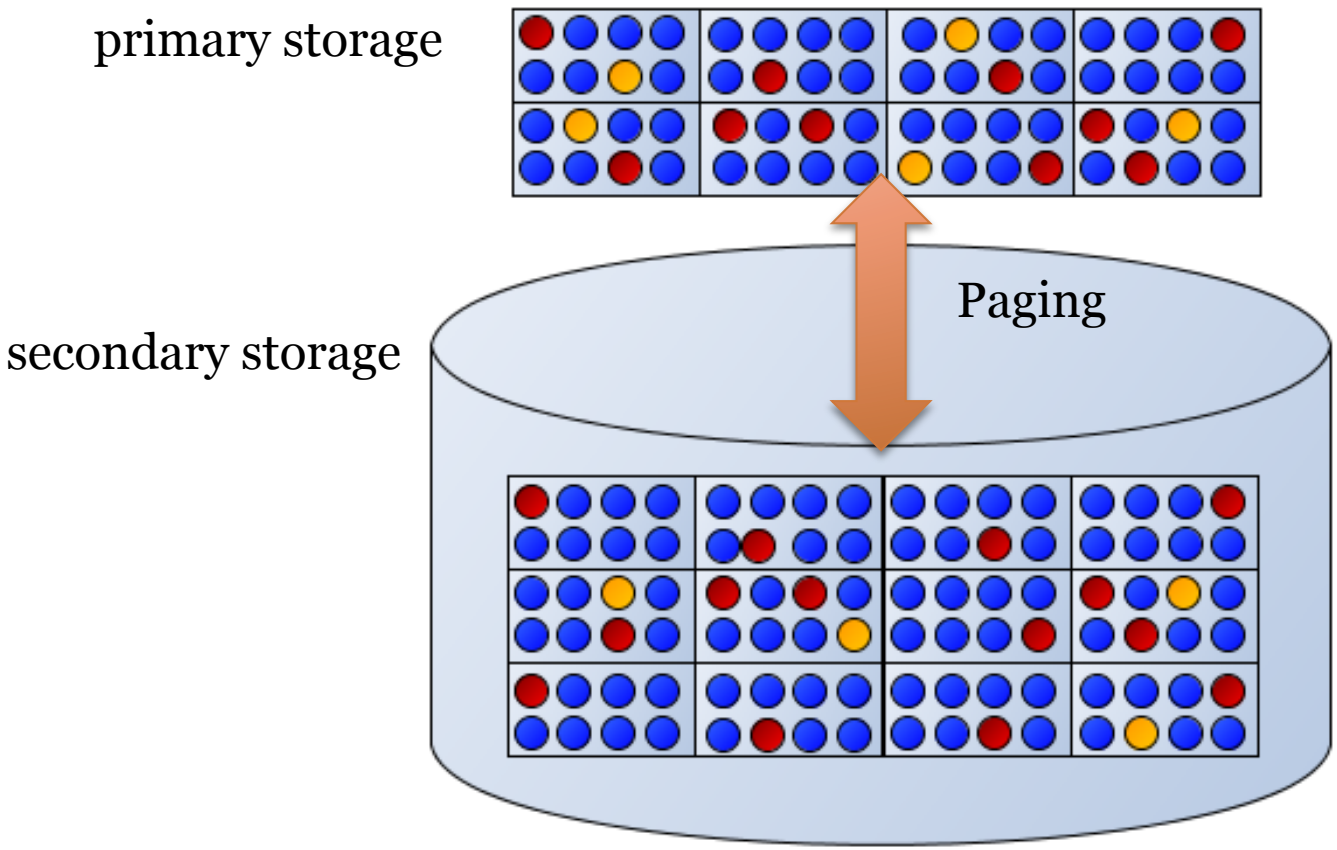
The n-Minute Rule

$$BEInterval = \frac{StoragePrice[\$]}{IO/s} \cdot \frac{1}{ItemSize[byte] \cdot RAMPrice[\$/byte]}$$

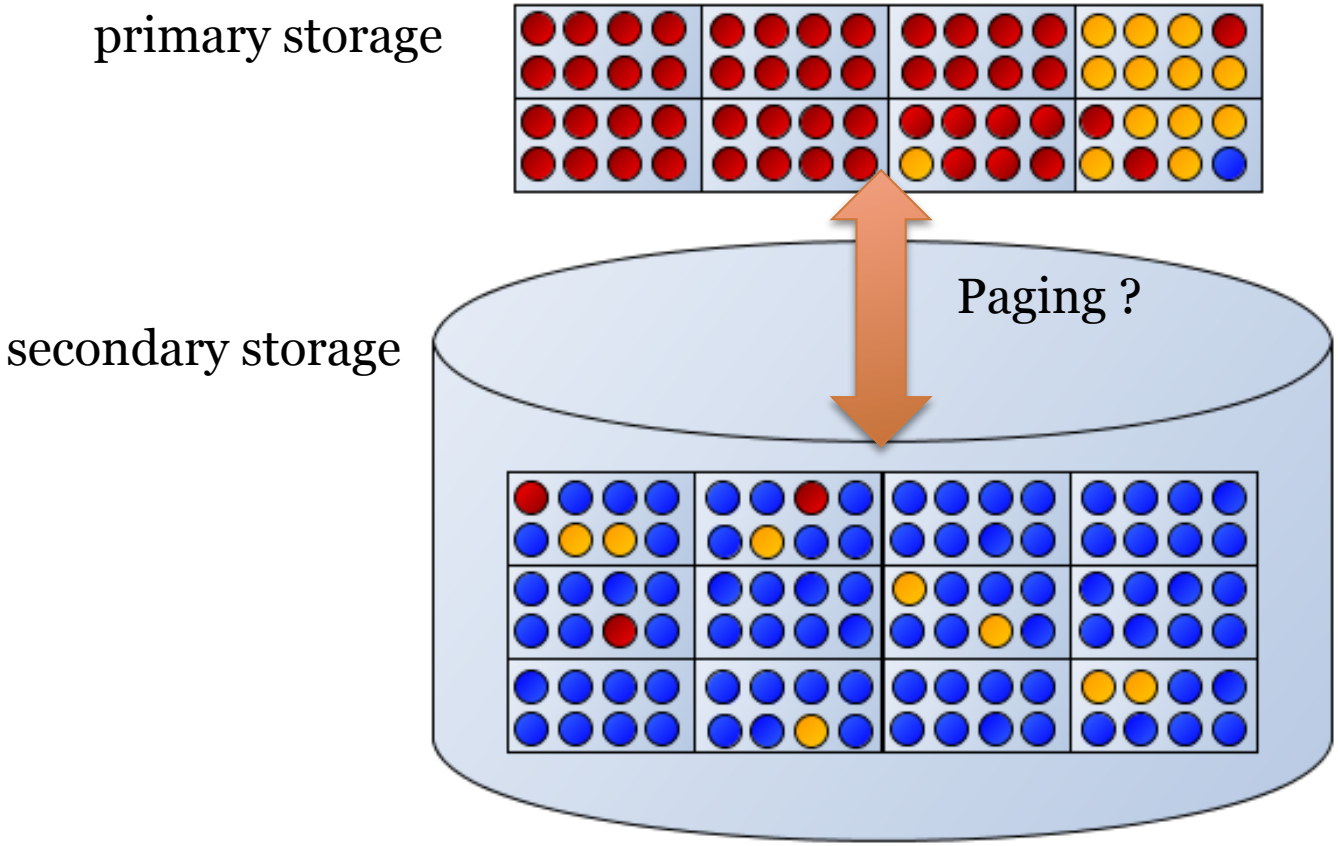
OLTP Workloads

- Relatively short running transactions
- Only limited data is accessed
- Many „cold“ records
  - cold = infrequently accessed
- Few „hot“ records
  - hot = very frequently accessed

Hot and Cold Records in a DB



Hot and Cold Records Organized

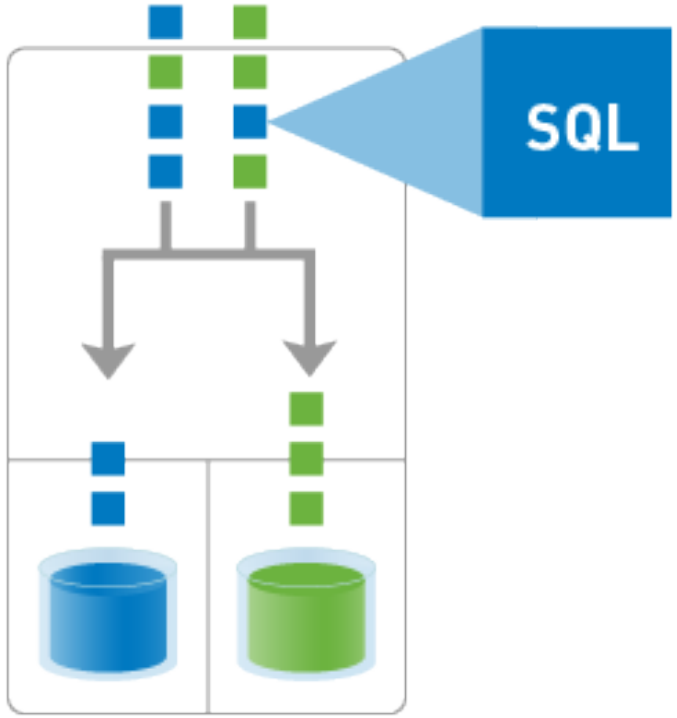


Stoica et al.‘s Approach

- System architecture
  - VoltDB
- Mode of operation
  - Identification of hot and cold data
  - Data reorganization technique
- Evaluation
  - TPC-C benchmark

VoltDB – Key Features

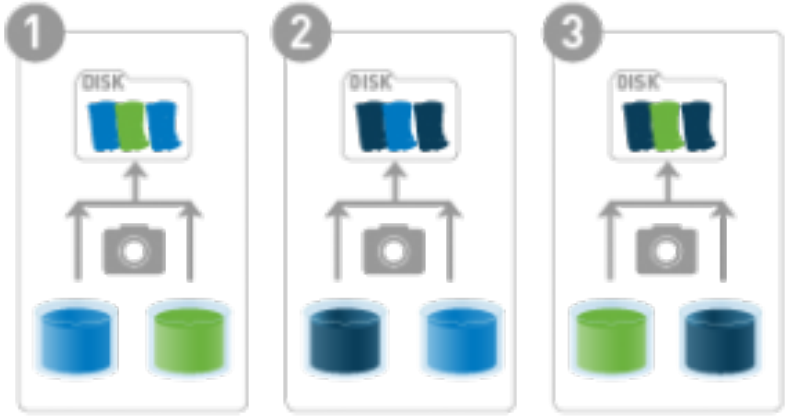
- ACID compliant



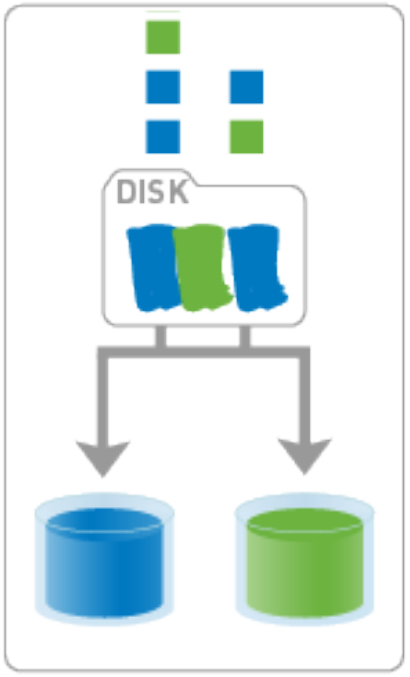
http://voltdb.com/products/key-features/

VoltDB – Key Features

- Durability



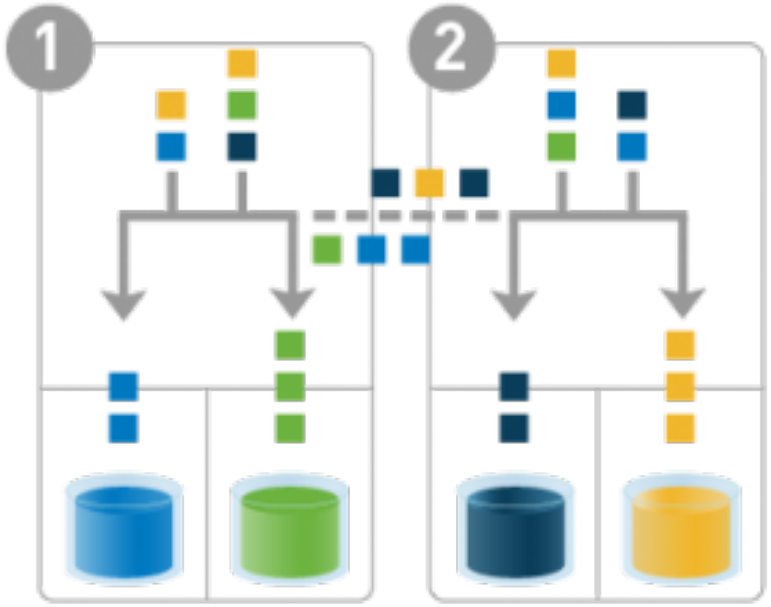
http://voltdb.com/products/key-features/



Snapshots & Command Logs

VoltDB – Key Features

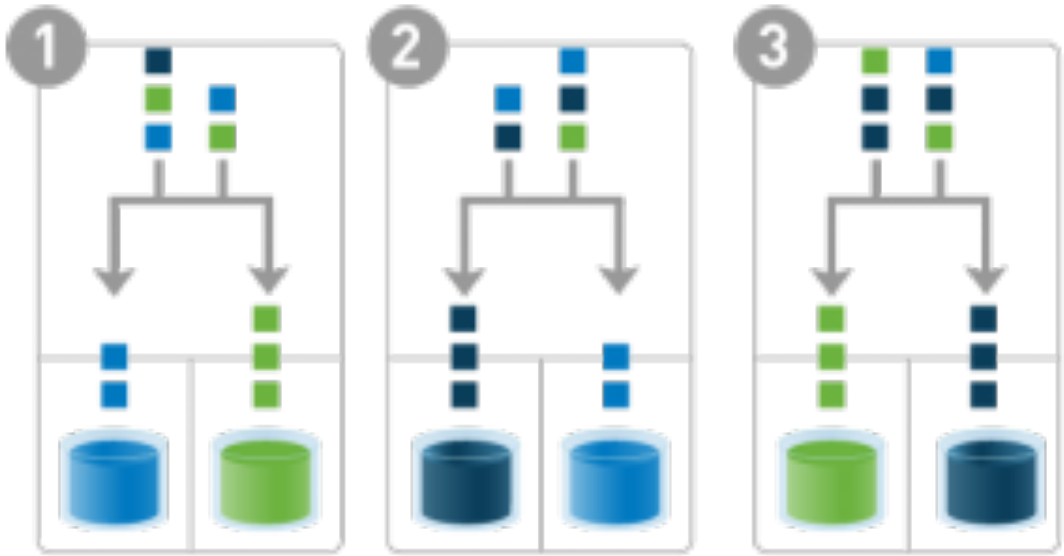
- Supports
  - scaling up
  - scaling out



http://voltdb.com/products/key-features/

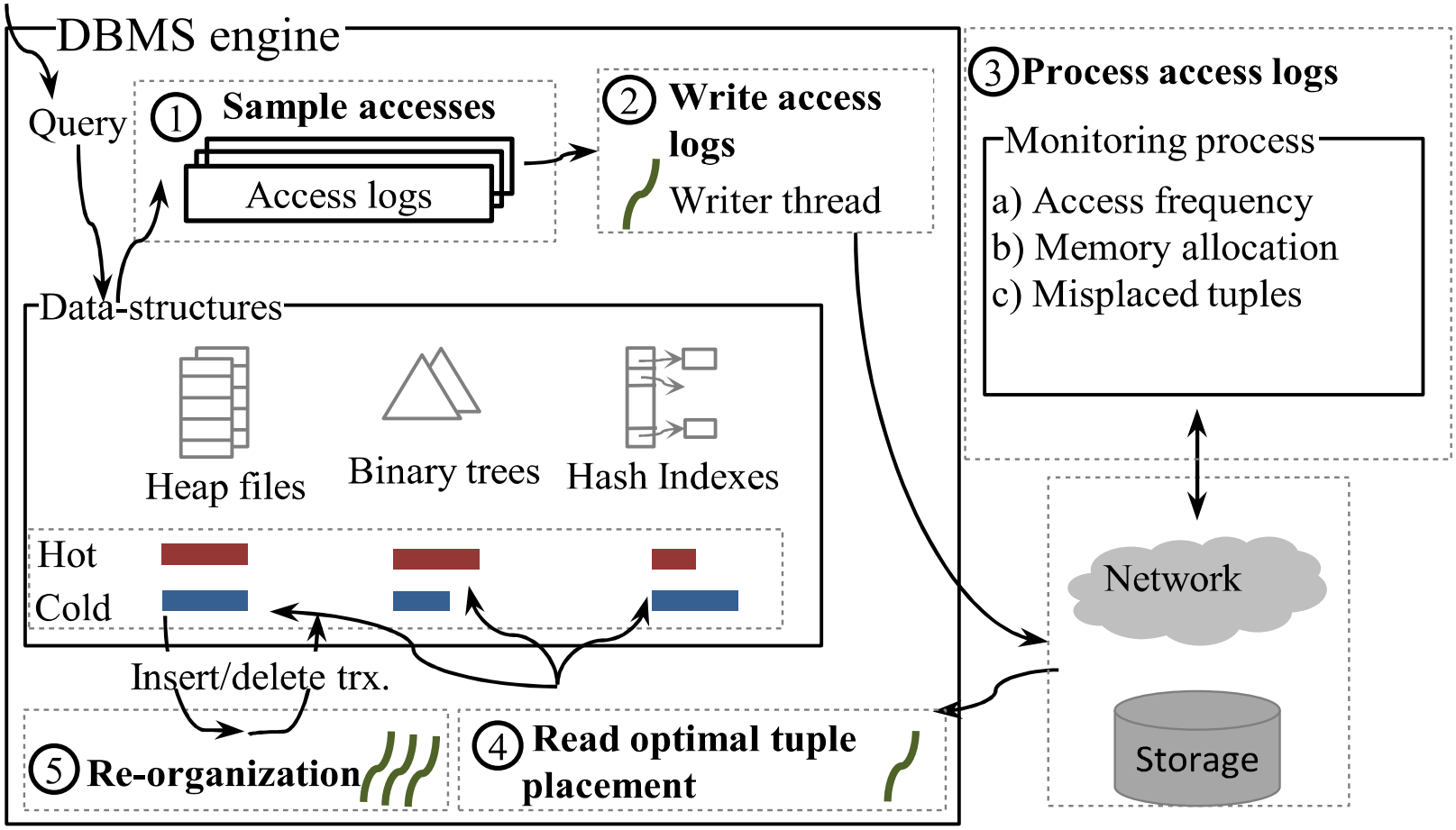
VoltDB – Key Features

- High availability



http://voltdb.com/products/key-features/

Stoica et al.‘s System Architecture



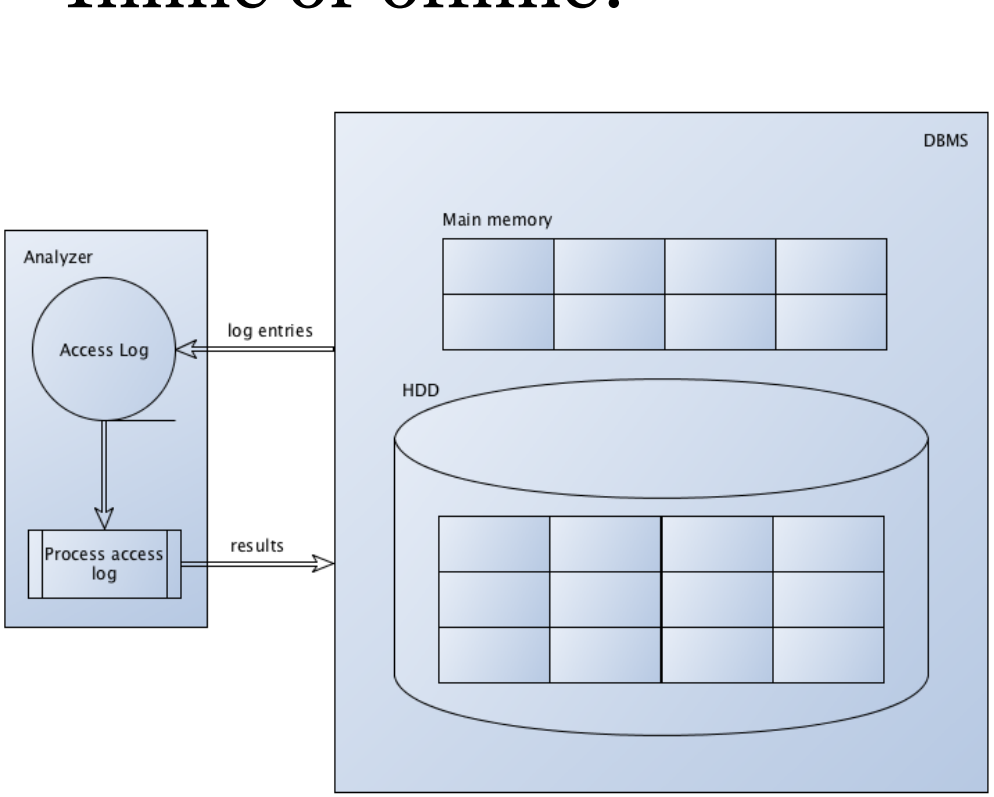
Radu Stoica and Anastasia Ailamaki. Enabling Efficient OS Paging for Main- memory OLTP Databases.

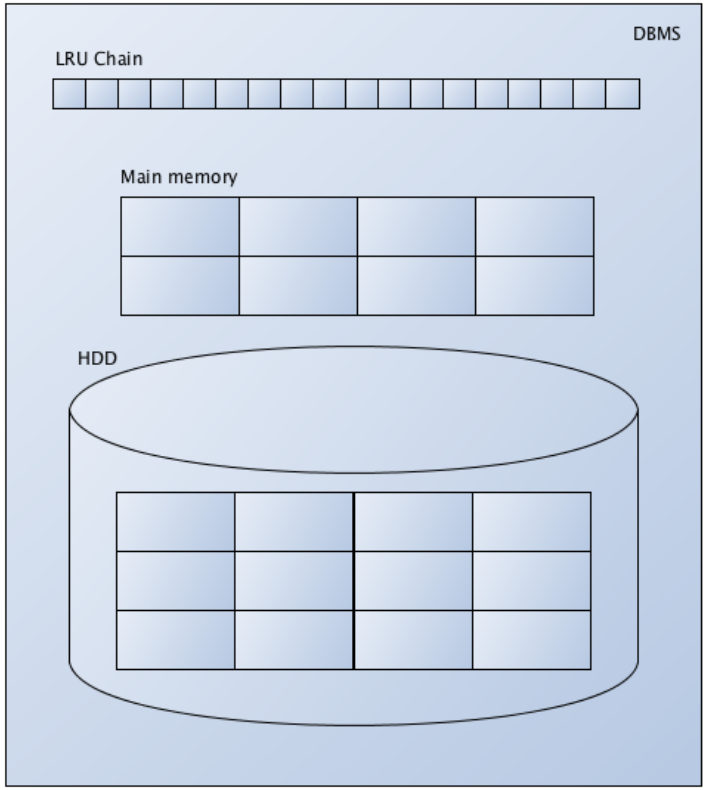
Memory Management

- Always in memory:
  - code mappings, front-end data structures, Java heap
- Only large relational objects can be paged out to SSD
  - memory is allocated sequentially in the virtual address space

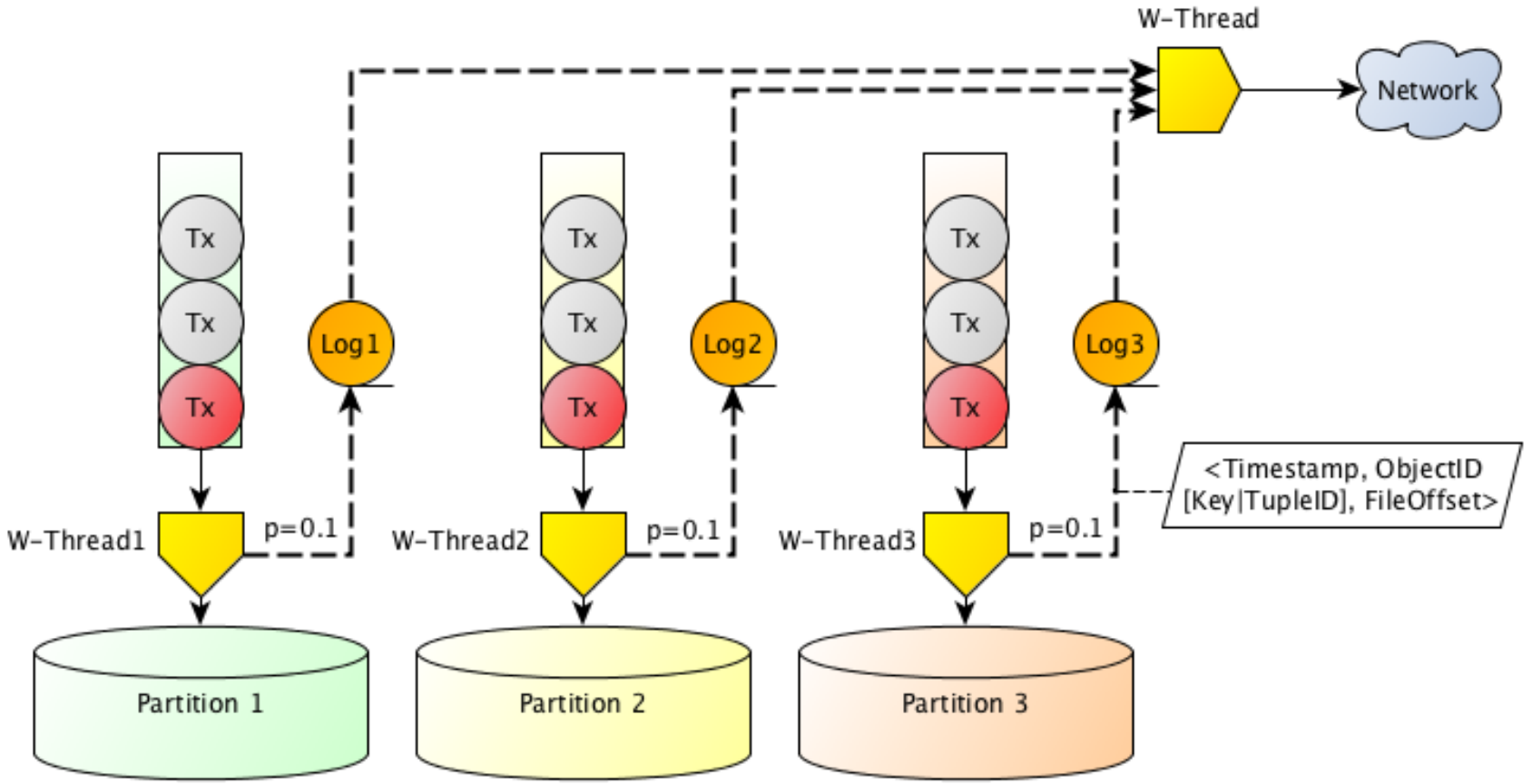
How to Identify Hot and Cold Data in Memory?

- Inline or offline?





Sampling and Access Logs

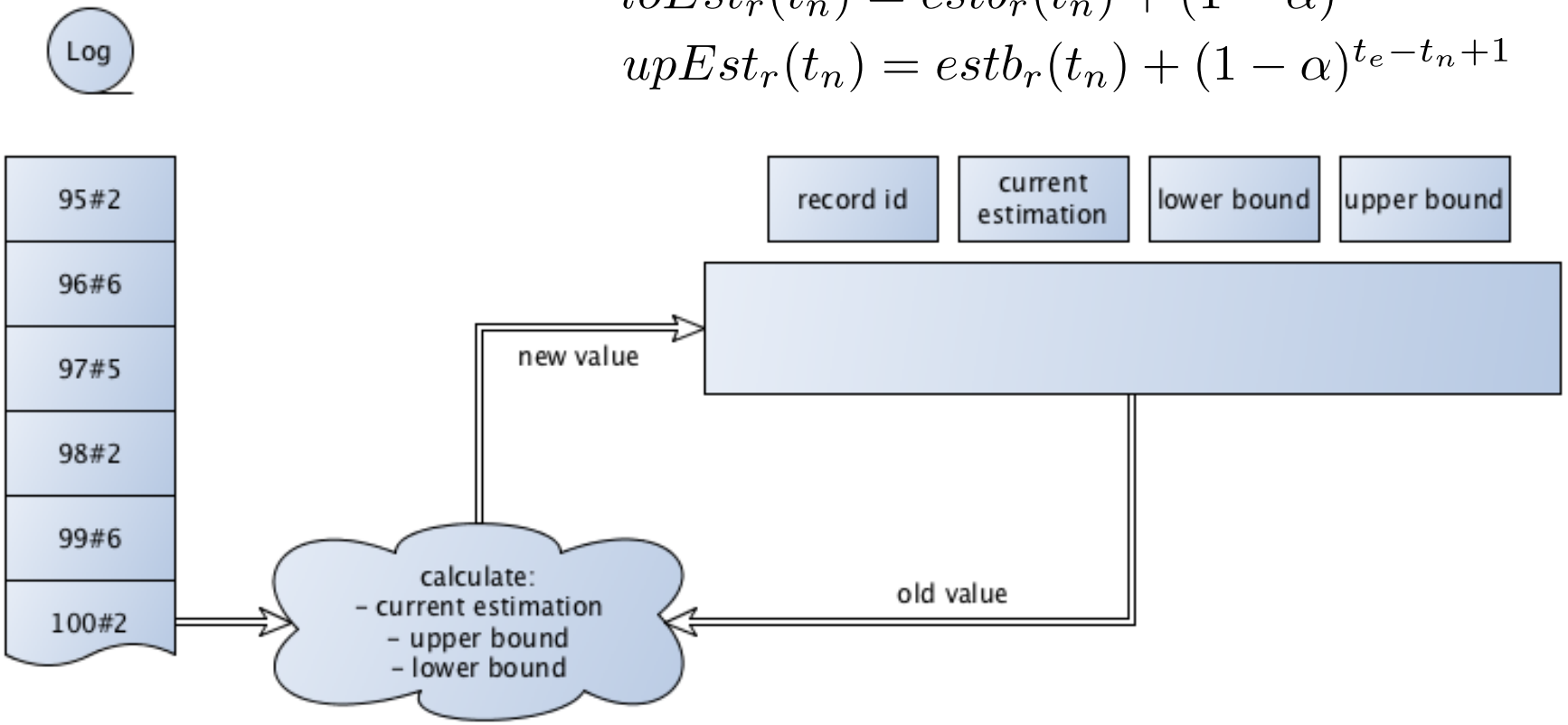


Levandoski et al.'s Backward Algorithm

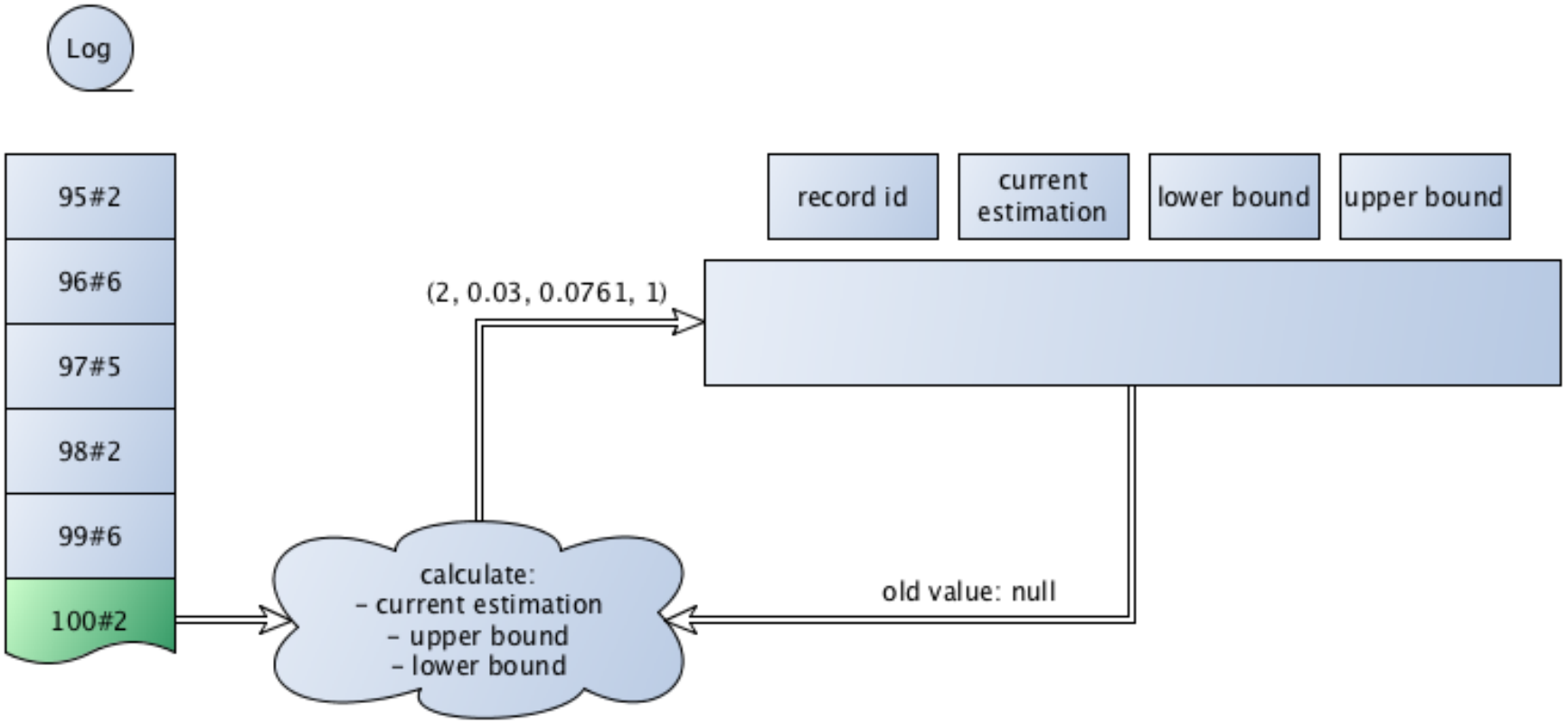
- Exponential smoothing
  $est_r(t_n) = \alpha \cdot \chi_{t_n} + (1 - \alpha) \cdot est_r(t_{n-1})$
- find K hottest tuples
- scan log in reverse order
- log entry = [timeslice,record id]

Levandoski et al.‘s Backward Algorithm

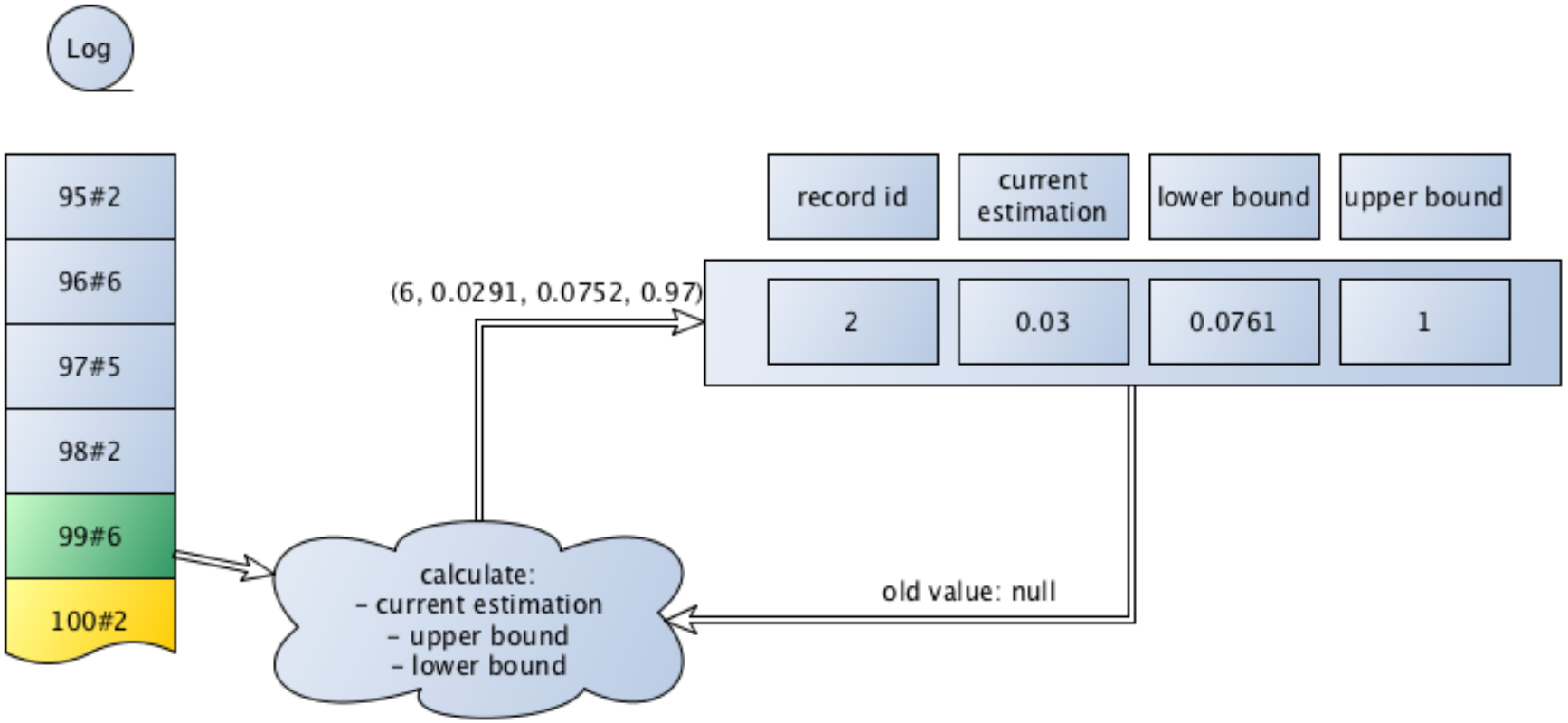
$$estb_r(t_n) = \alpha(1-\alpha)^{t_e - t_n} + estb_r(t_{last})$$
$$loEst_r(t_n) = estb_r(t_n) + (1-\alpha)^{t_e - t_0 + 1}$$
$$upEst_r(t_n) = estb_r(t_n) + (1-\alpha)^{t_e - t_n + 1}$$



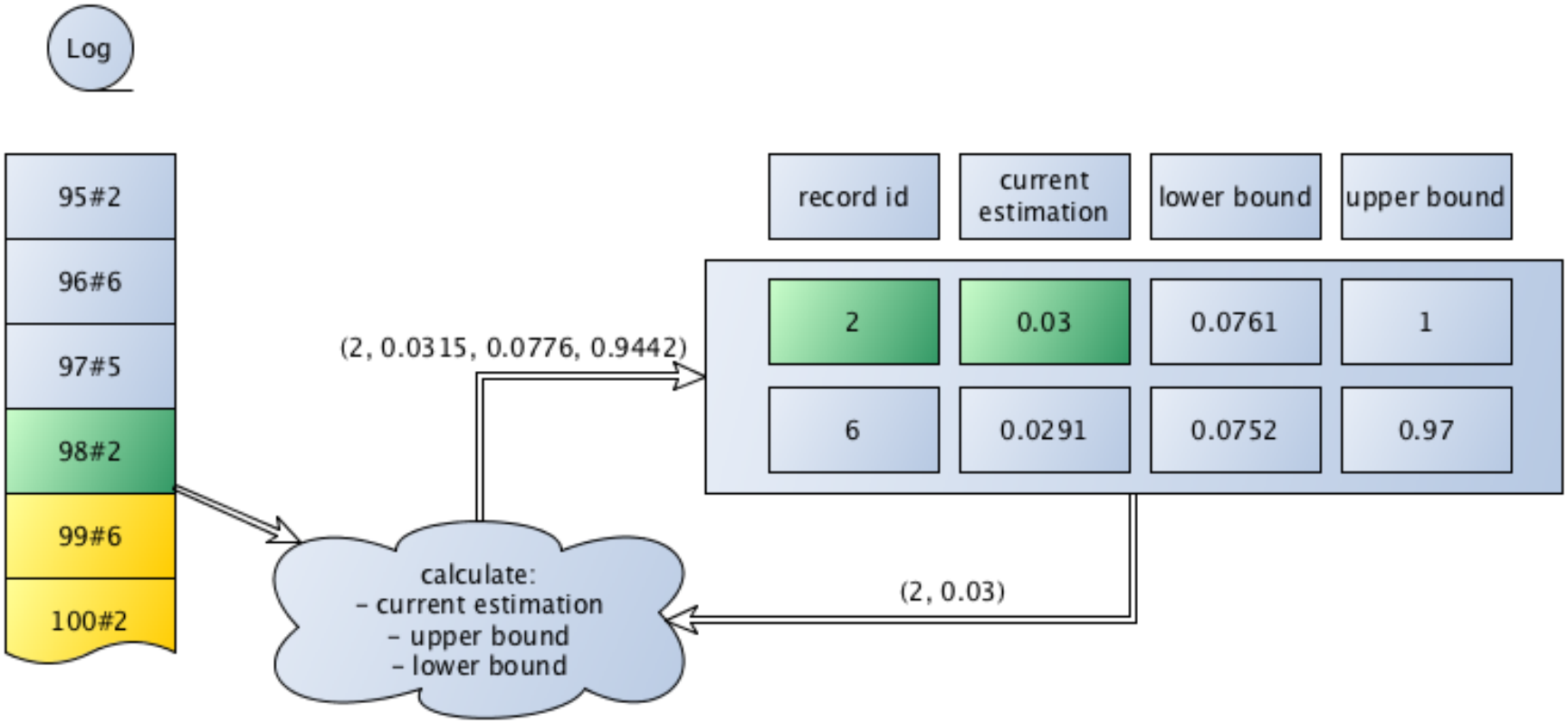
Levandoski et al.‘s Backward Algorithm



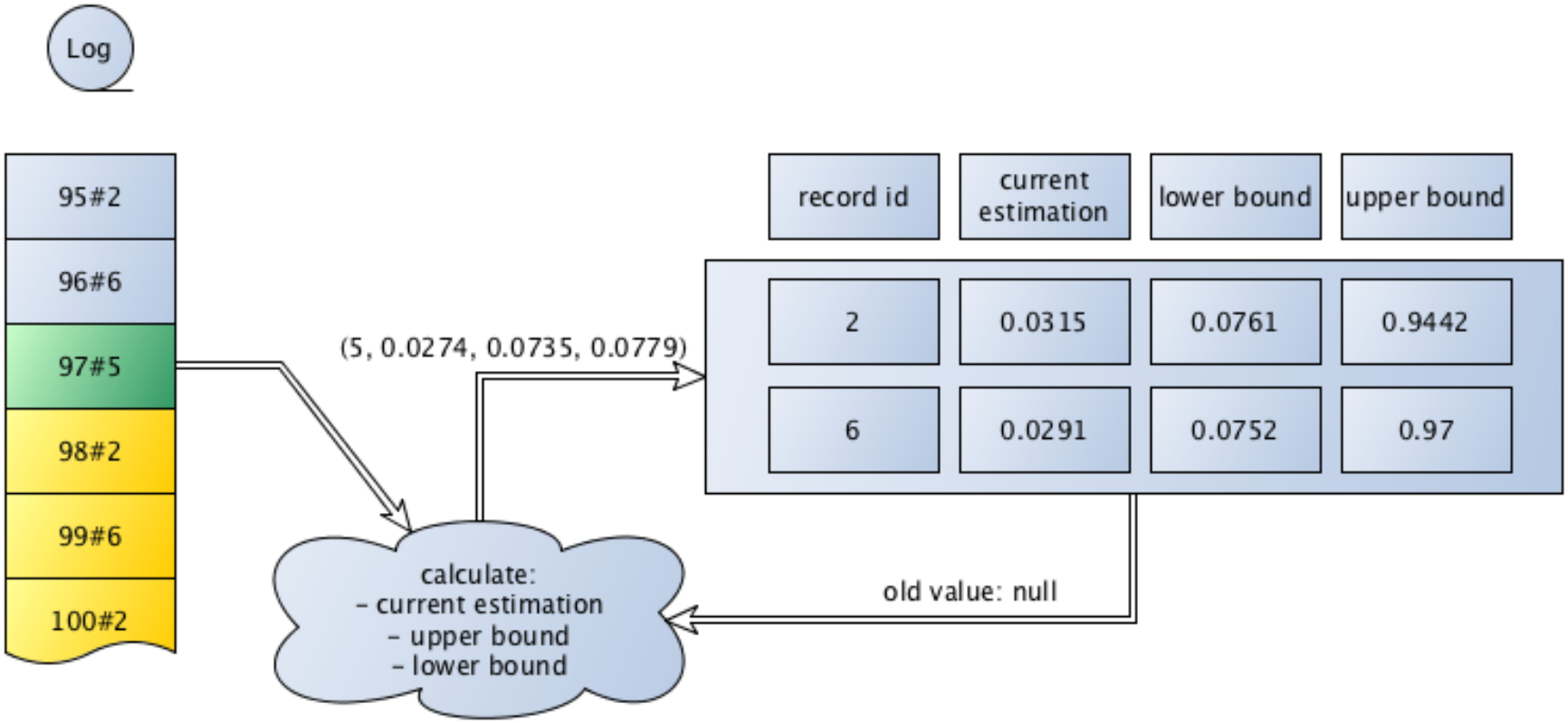
Levandoski et al.‘s Backward Algorithm



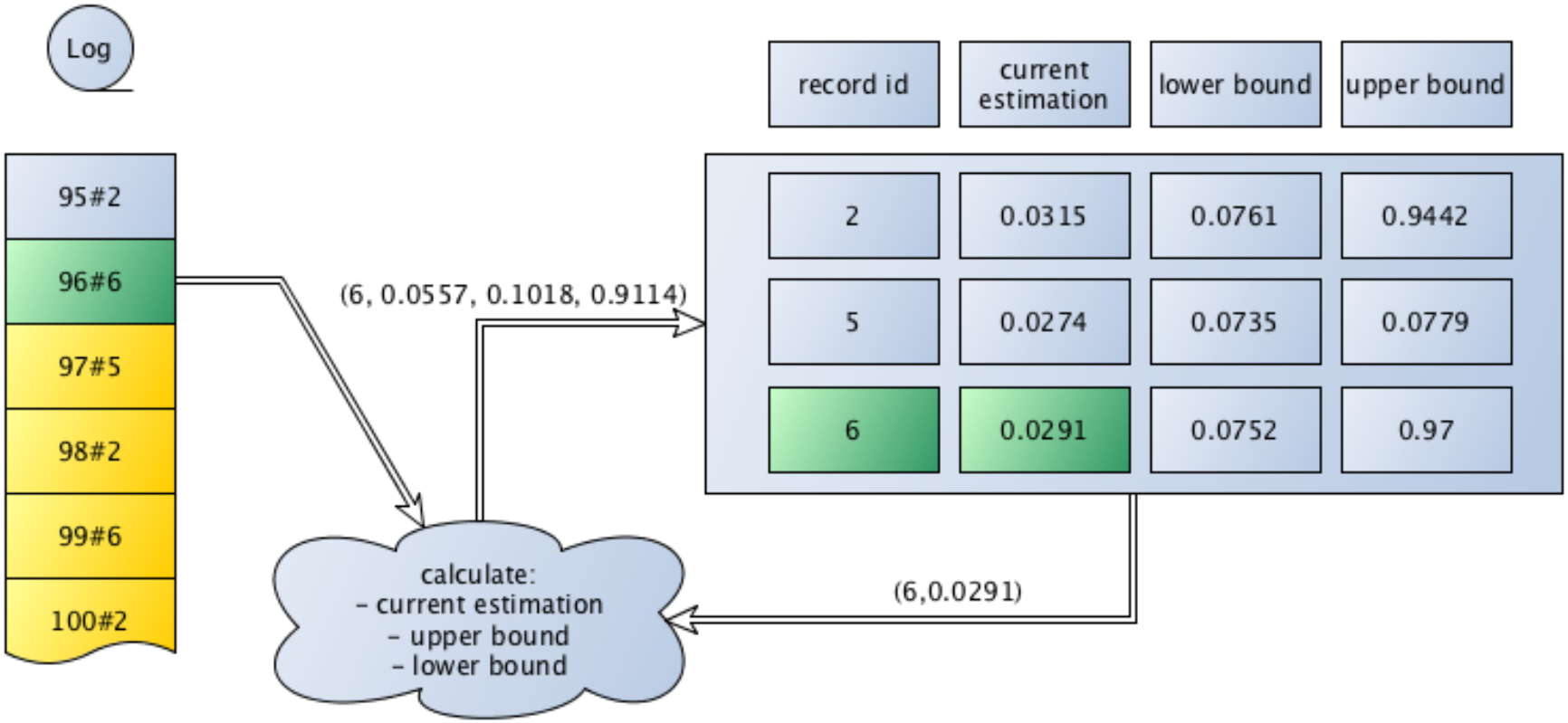
Levandoski et al.‘s Backward Algorithm



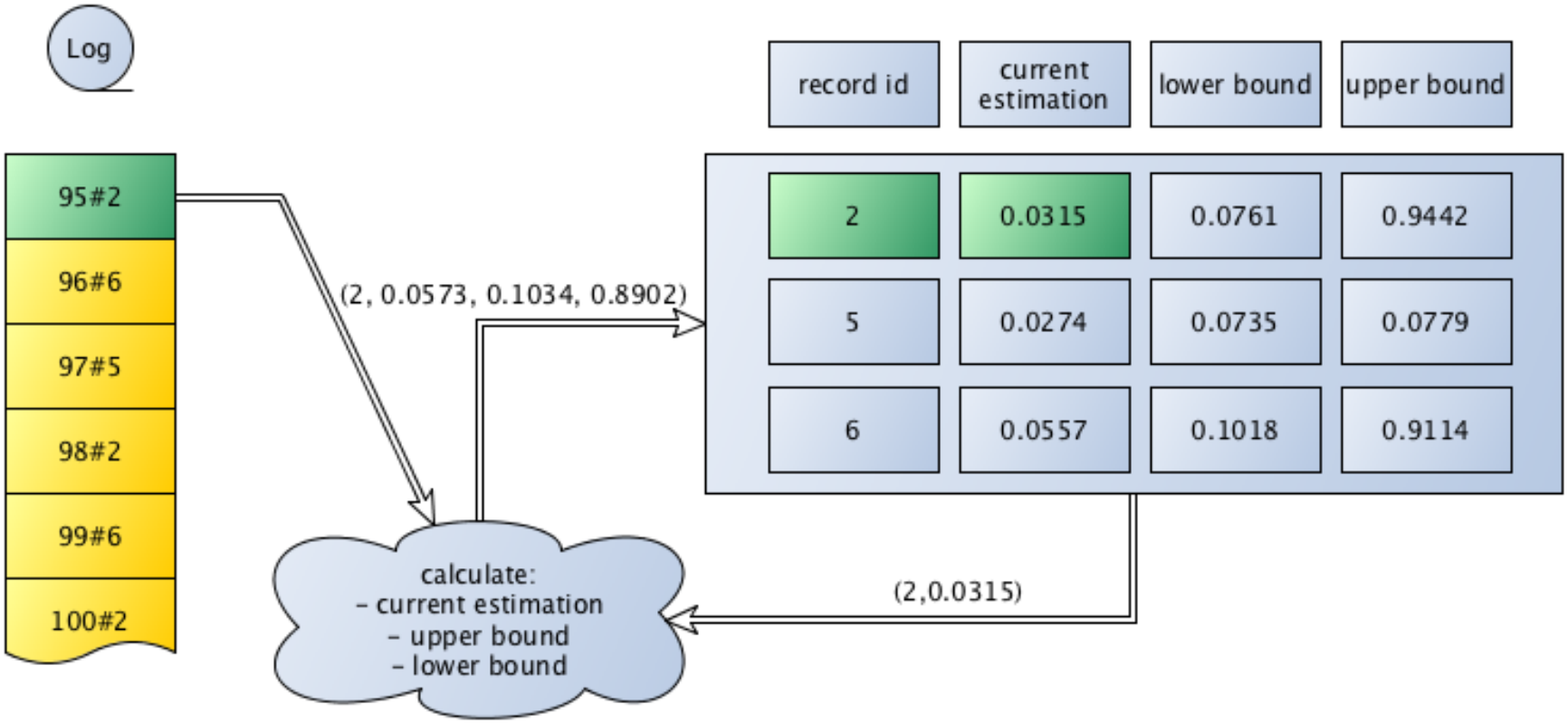
Levandoski et al.‘s Backward Algorithm



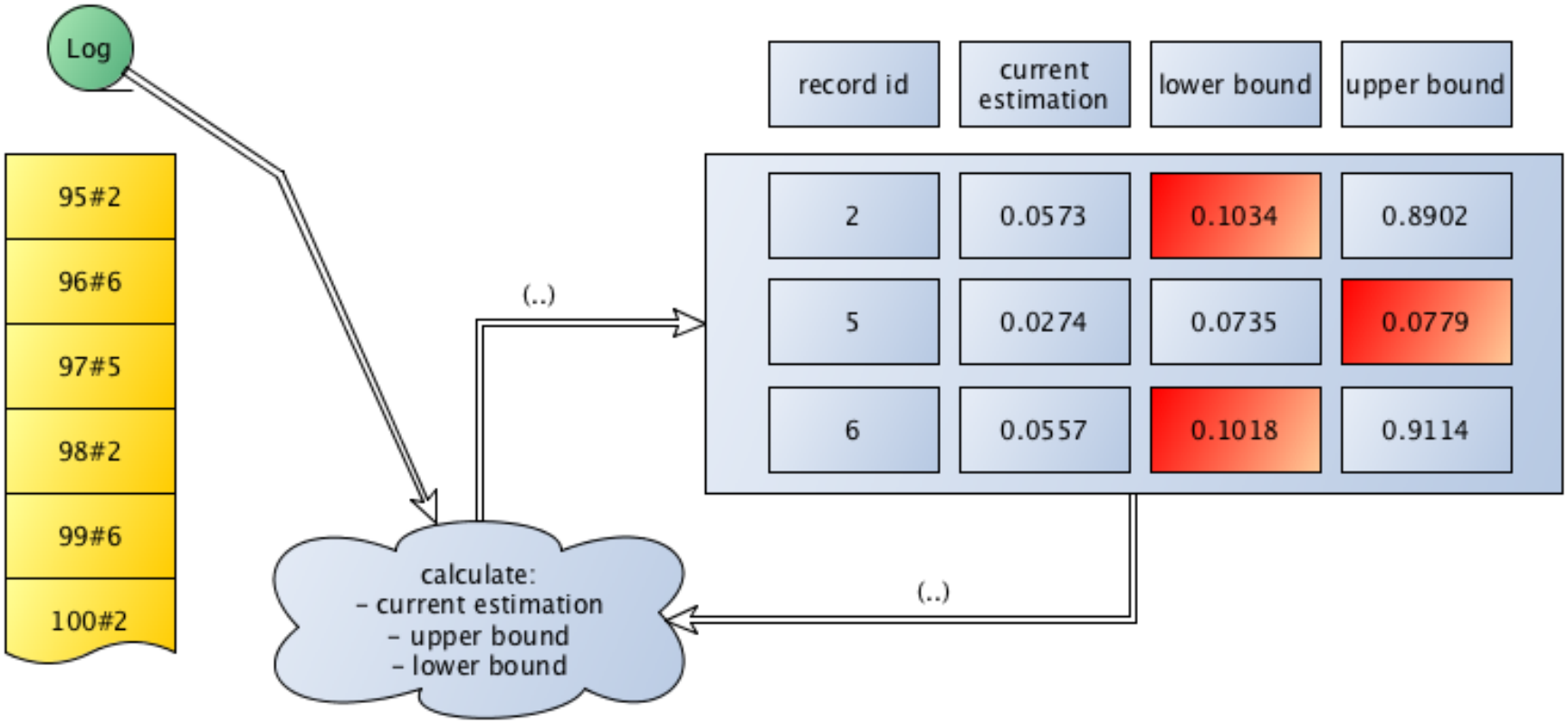
Levandoski et al.‘s Backward Algorithm



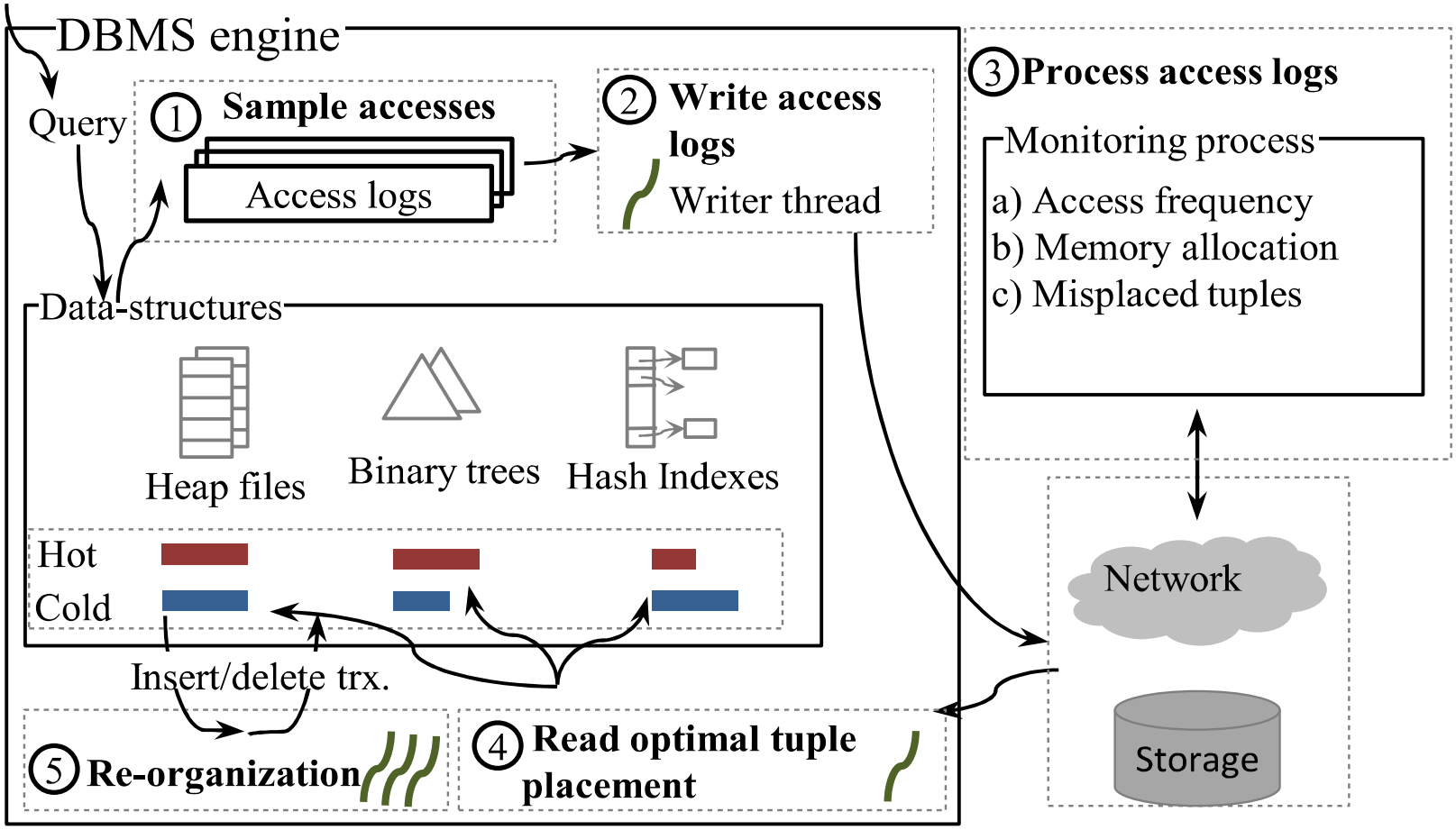
Levandoski et al.‘s Backward Algorithm



Levandoski et al.‘s Backward Algorithm



Stoica et al.‘s System Architecture



Radu Stoica and Anastasia Ailamaki. Enabling Efficient OS Paging for Main- memory OLTP Databases.

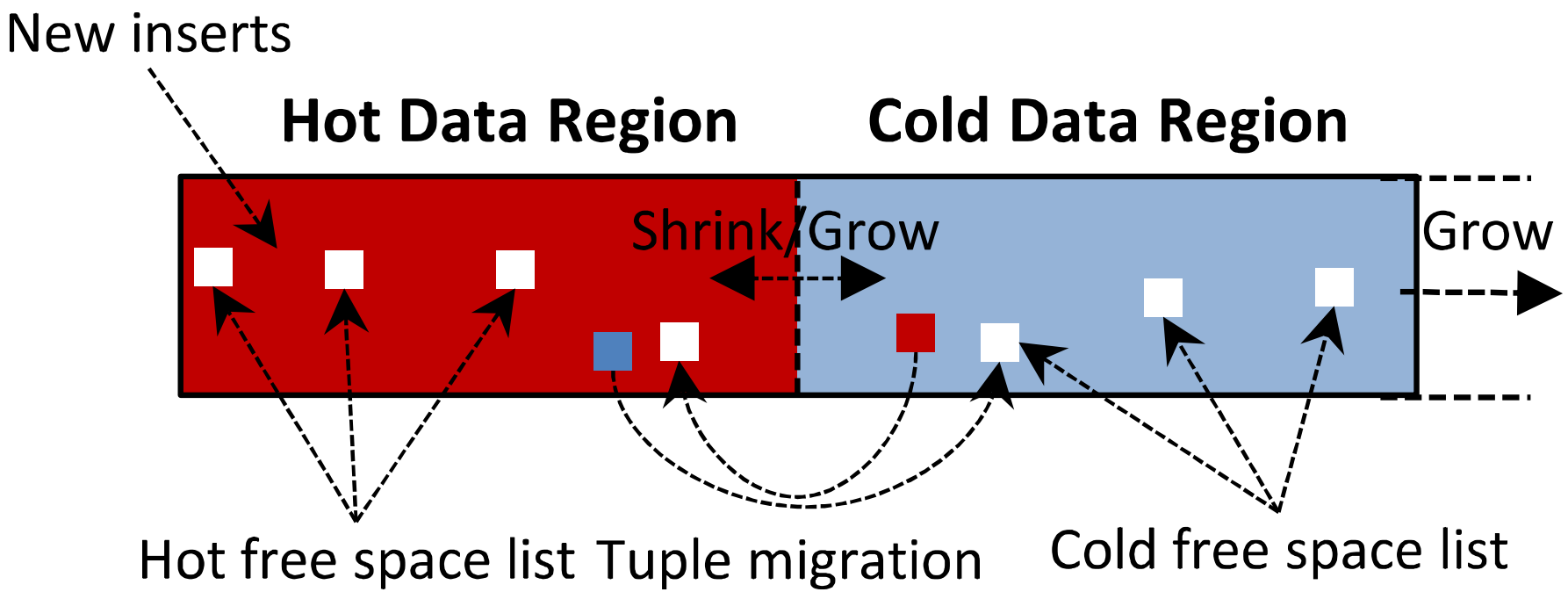
How to Find Misplaced Tuples?

- Compute access density $d(t_i) = \frac{freq(t_i)}{size(t_i)}$
- Compute memory budget of a relational object R_j

$$mem\_budget(R_j) = avg_{t_i \in R_j}(size(t_i)) \cdot |\{t_i \in R_j | t_i \text{ hot}\}|$$

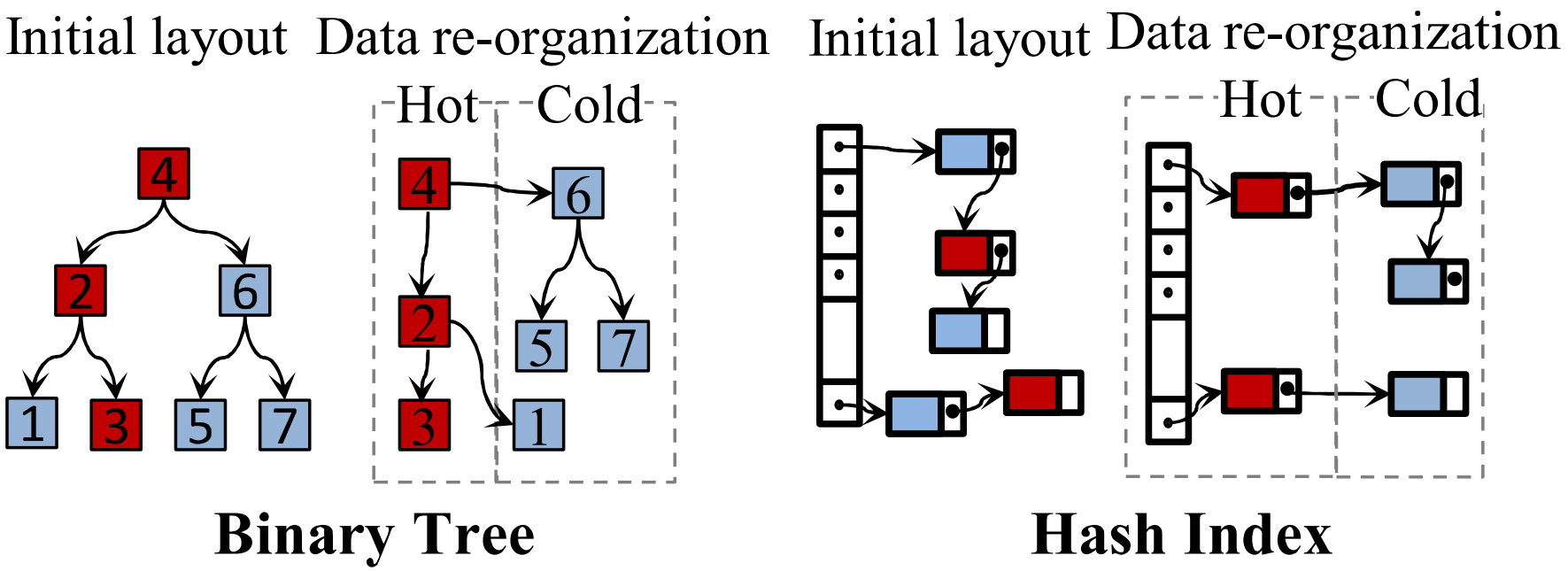
- Output: memory budget + 2 lists of misplaced tuples for each relational object

Data Reorganization



Radu Stoica and Anastasia Ailamaki. Enabling Efficient OS Paging for Main- memory OLTP Databases.

Data Reorganization



Radu Stoica and Anastasia Ailamaki. Enabling Efficient OS Paging for Main- memory OLTP Databases.

Evaluation

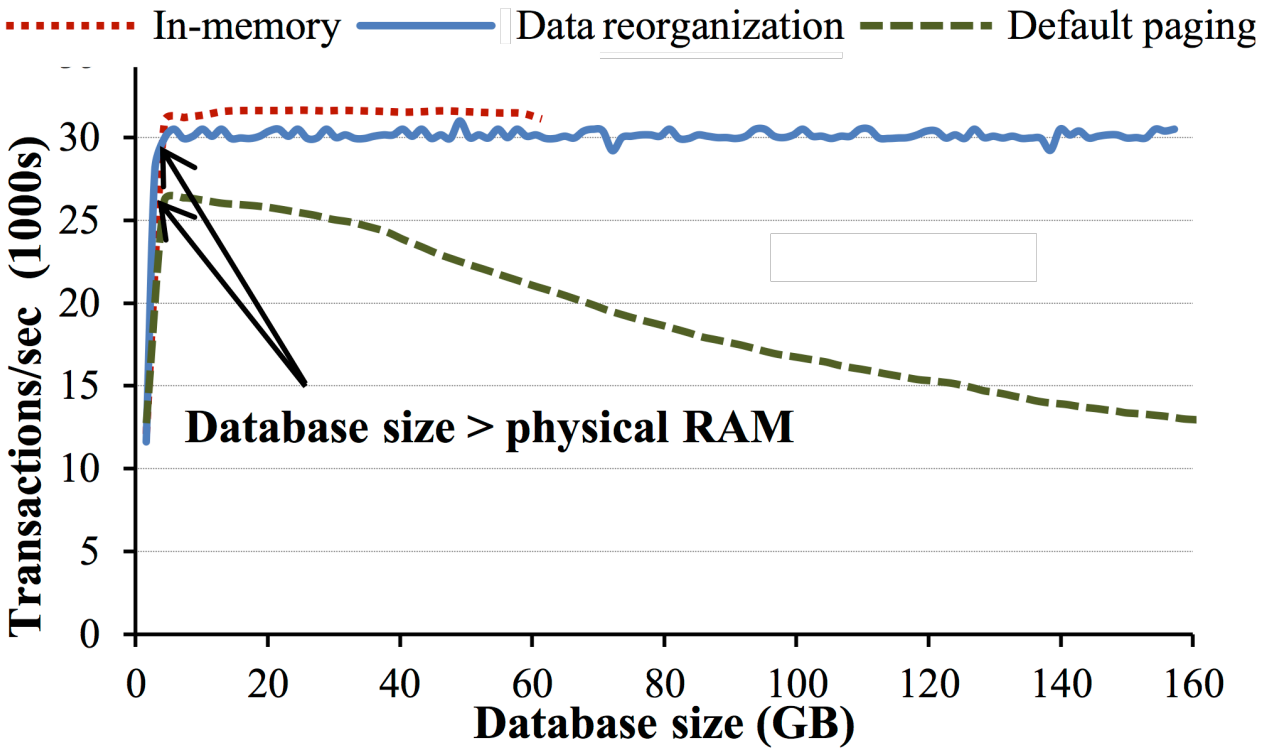
- Hardware
  - 4 socket Quad-Core AMD Opteron (16 cores in total)
  - 64GB physical DRAM
  - 160GB FusionIO PCIe SSD
    - up to 65,000 4kB random read IOPS
    - 20,000-75,000 4kB random write IOPS
- Environment
  - 10Gb local network

Evaluation - 3 Settings

1. In-memory:
   - VoltDB, ~62GB RAM
2. Default paging:
   - VoltDB, ~3GB RAM
3. Data reorganization:
   - modified VoltDB, ~ 3GB RAM

TPC-C Benchmark

- Throughput



Radu Stoica and Anastasia Ailamaki. Enabling Efficient OS Paging for Main- memory OLTP Databases.

TPC-C Benchmark

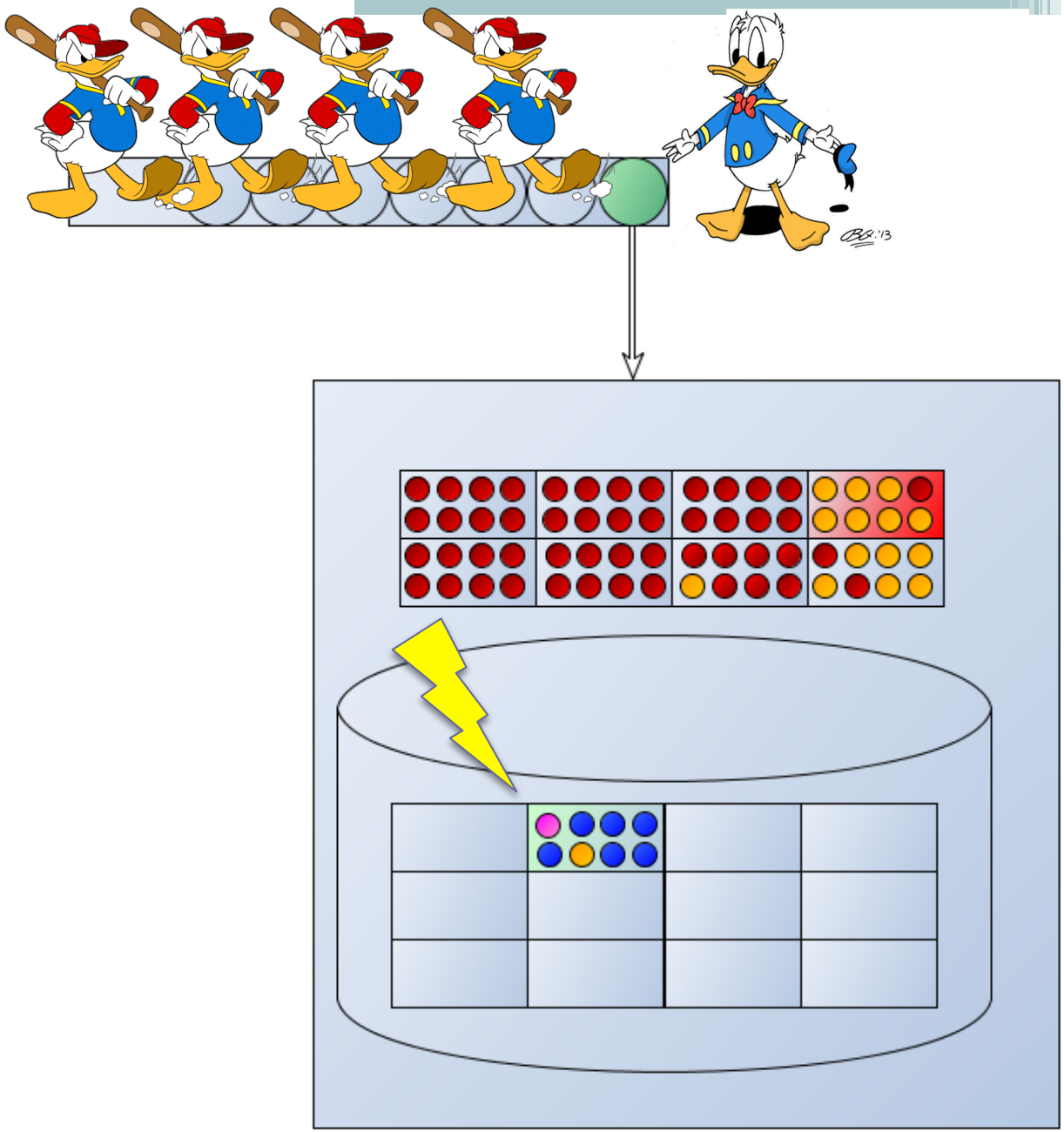
- Transaction latency

Trx.Name	**Server latency(μs)**				**Client latency(μs)**			
	No Paging		Paging		No Paging		Paging	
	avg	σ	avg	σ	avg	σ	avg	σ
NewOrder	283	135	318	182	3270	5430	3301	5509
OrderStatus	82	26	126	58	2950	5486	2990	5499
Payment	149	52	183	107	3027	5458	3094	5492
Delivery	314	152	347	214	3317	5331	3358	5513
StockLevel	797	243	898	348	3531	5372	3611	5570

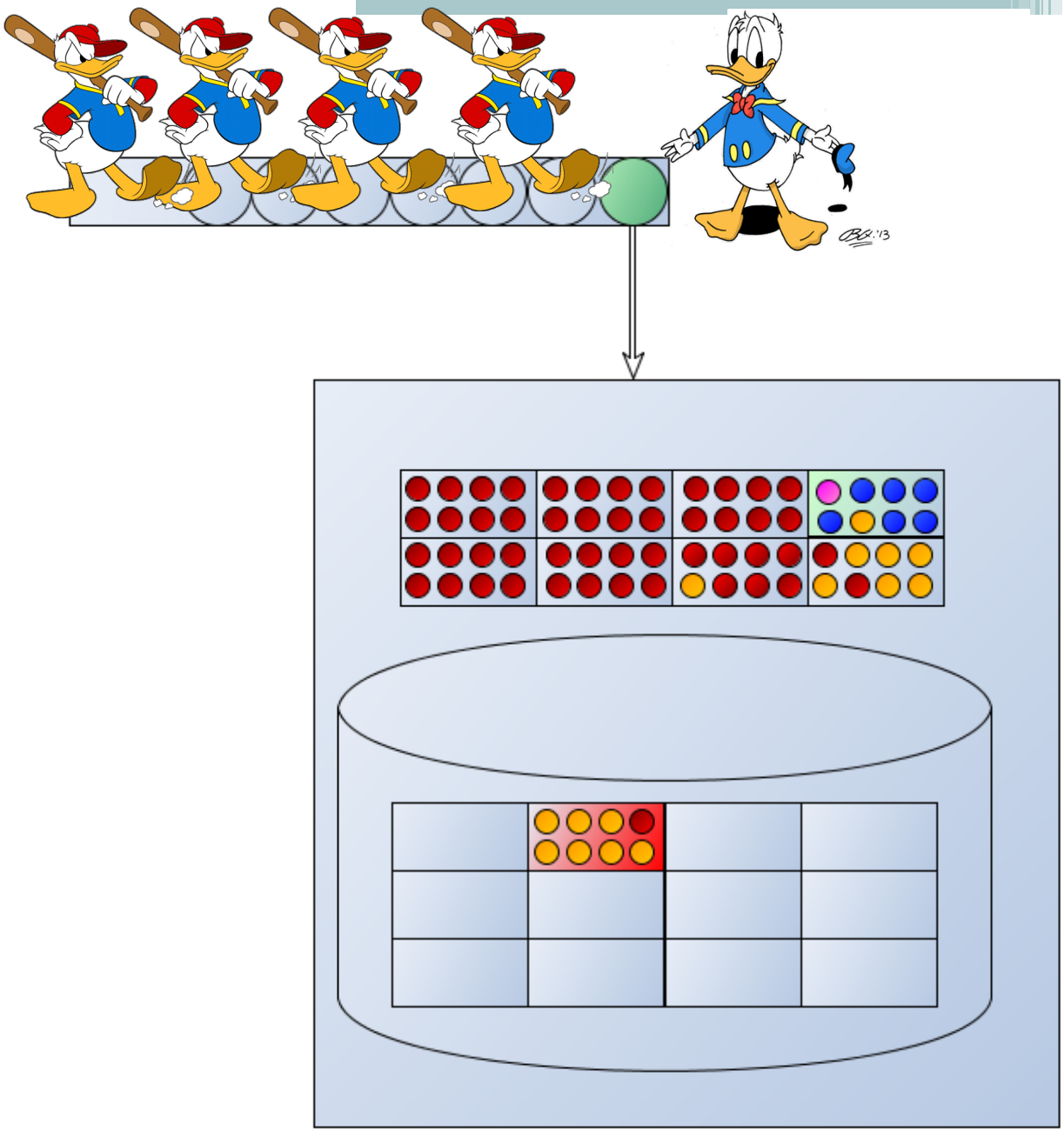
avg = average, σ = standard deviation

Radu Stoica and Anastasia Ailamaki. Enabling Efficient OS Paging for Main- memory OLTP Databases.

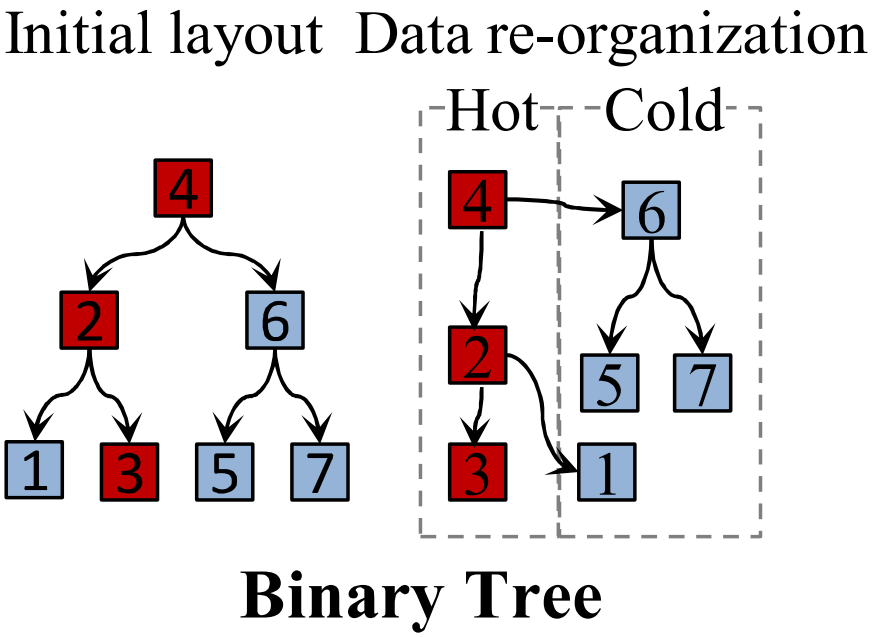
Problem 1: Page faults



Problem 1: Page faults



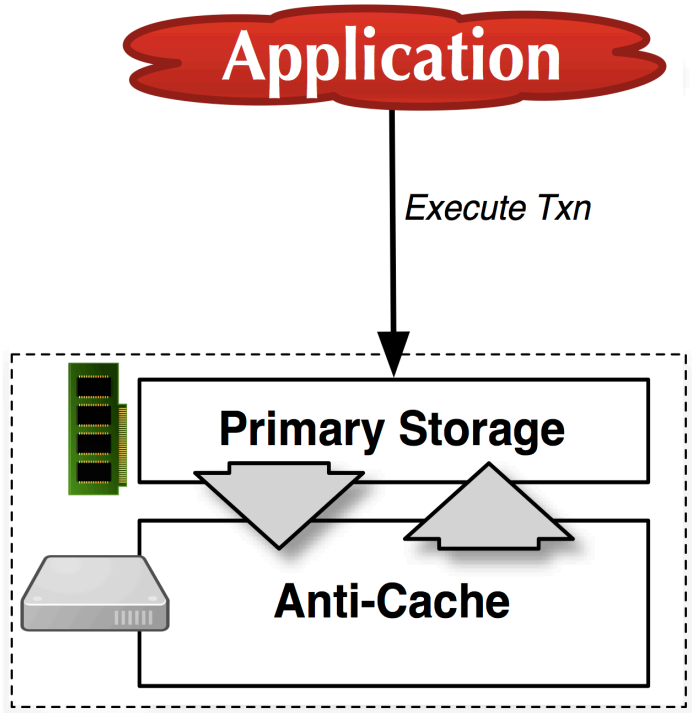
Problem 2: Access Paths



Radu Stoica and Anastasia Ailamaki. Enabling Efficient OS Paging for Main- memory OLTP Databases.

“Anti-Caching”

- Architecture
  - H-Store
- Mode of operation
- Evaluation



Justin DeBrabant, Andrew Pavlo, Stephen Tu, Michael Stonebraker, and Stan Zdonik. Anti-caching: A New Approach to Database Management System Architecture.

Architecture

- H-Store = VoltDB - $$

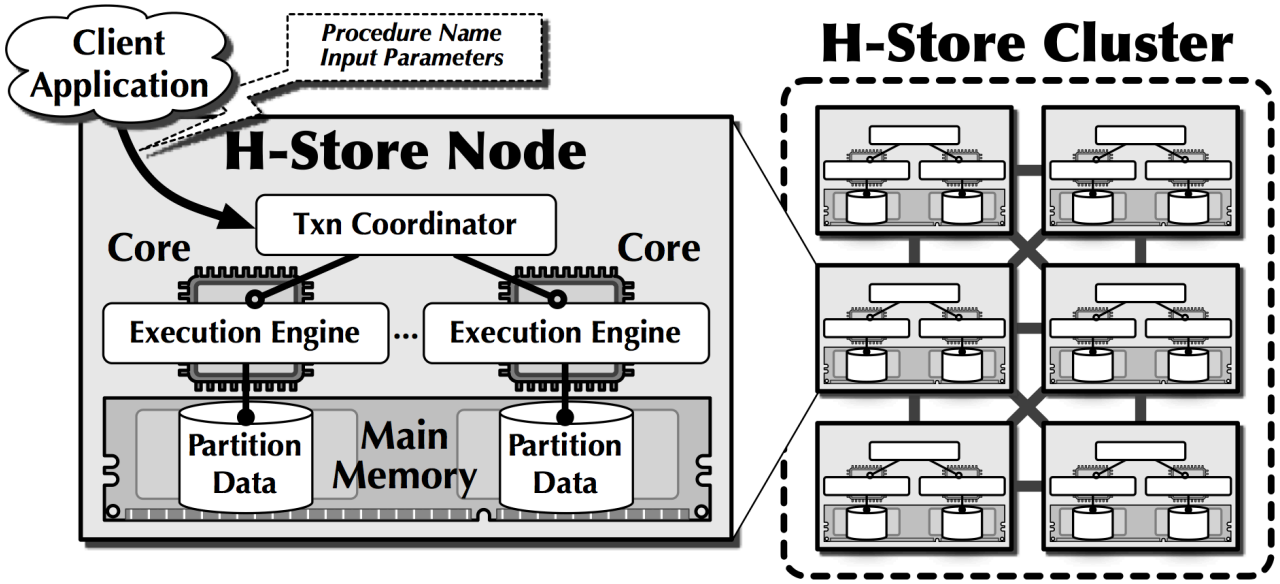
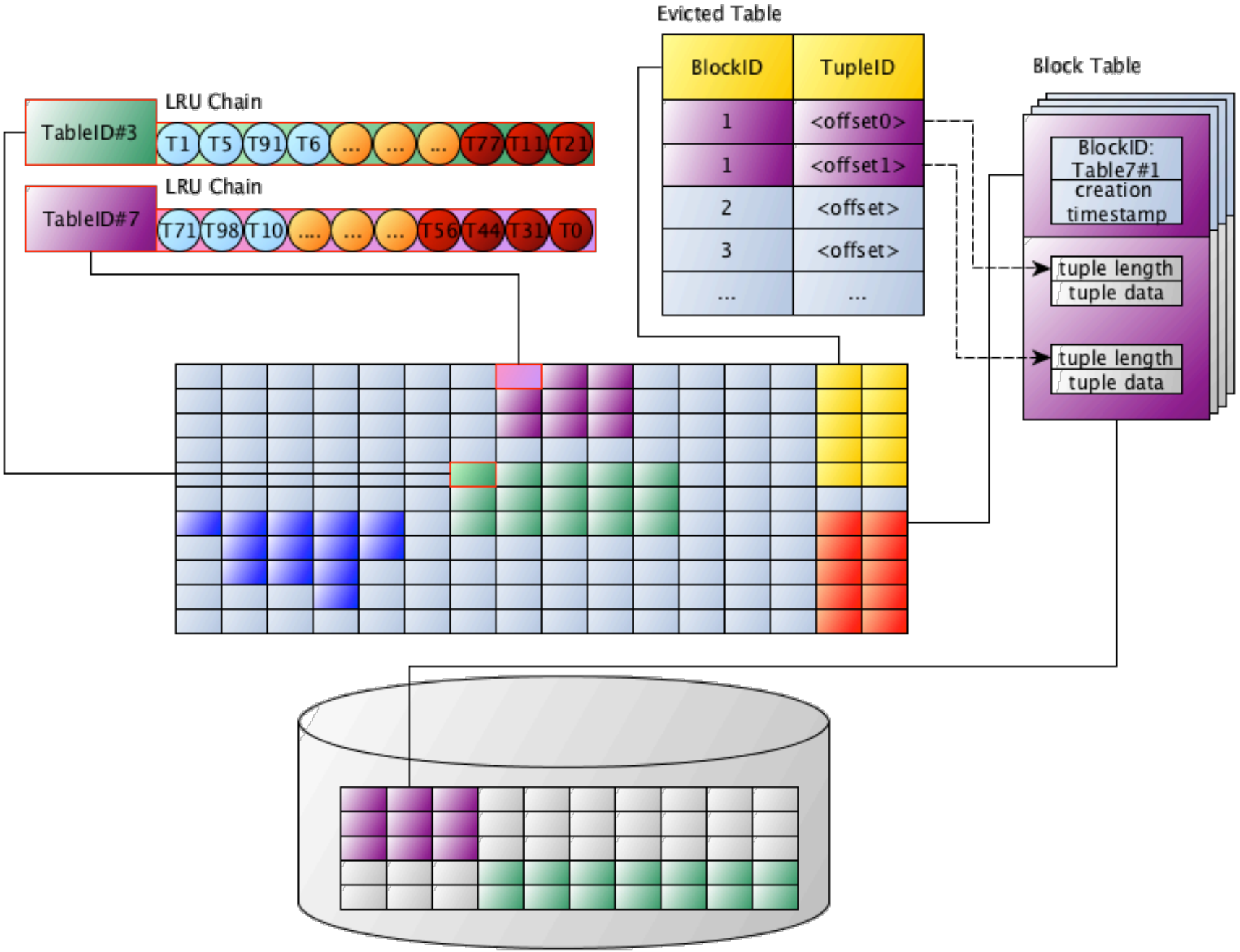
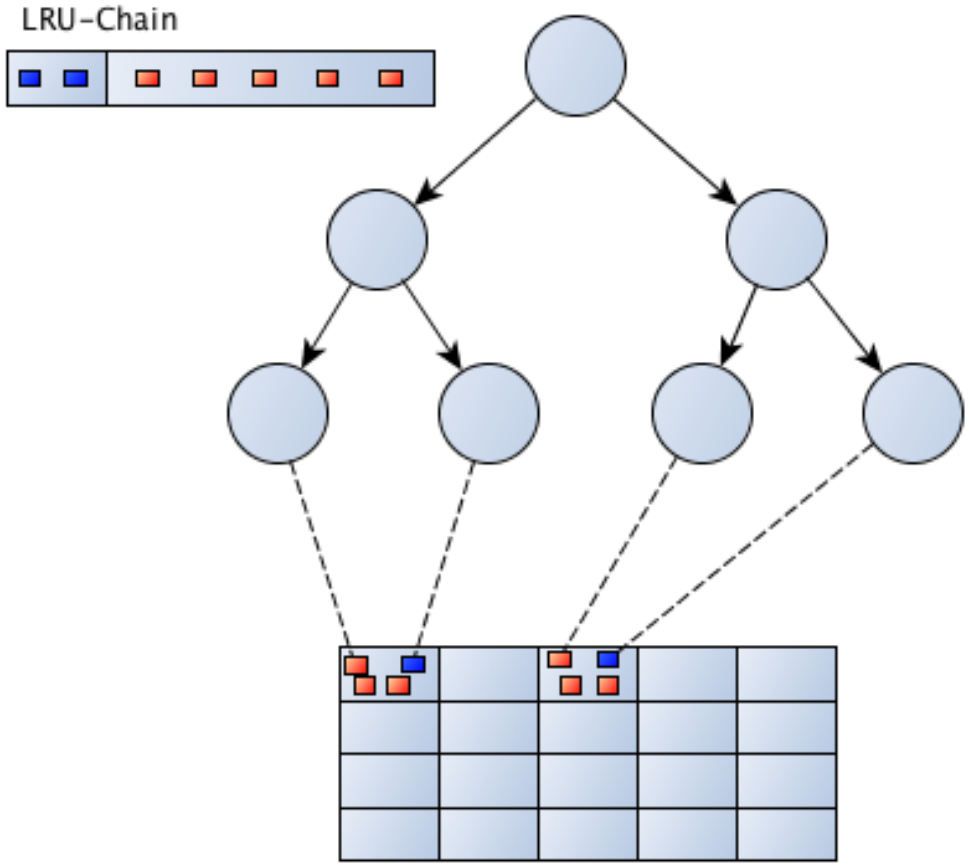


Figure 2: The H-Store Main Memory OLTP system.

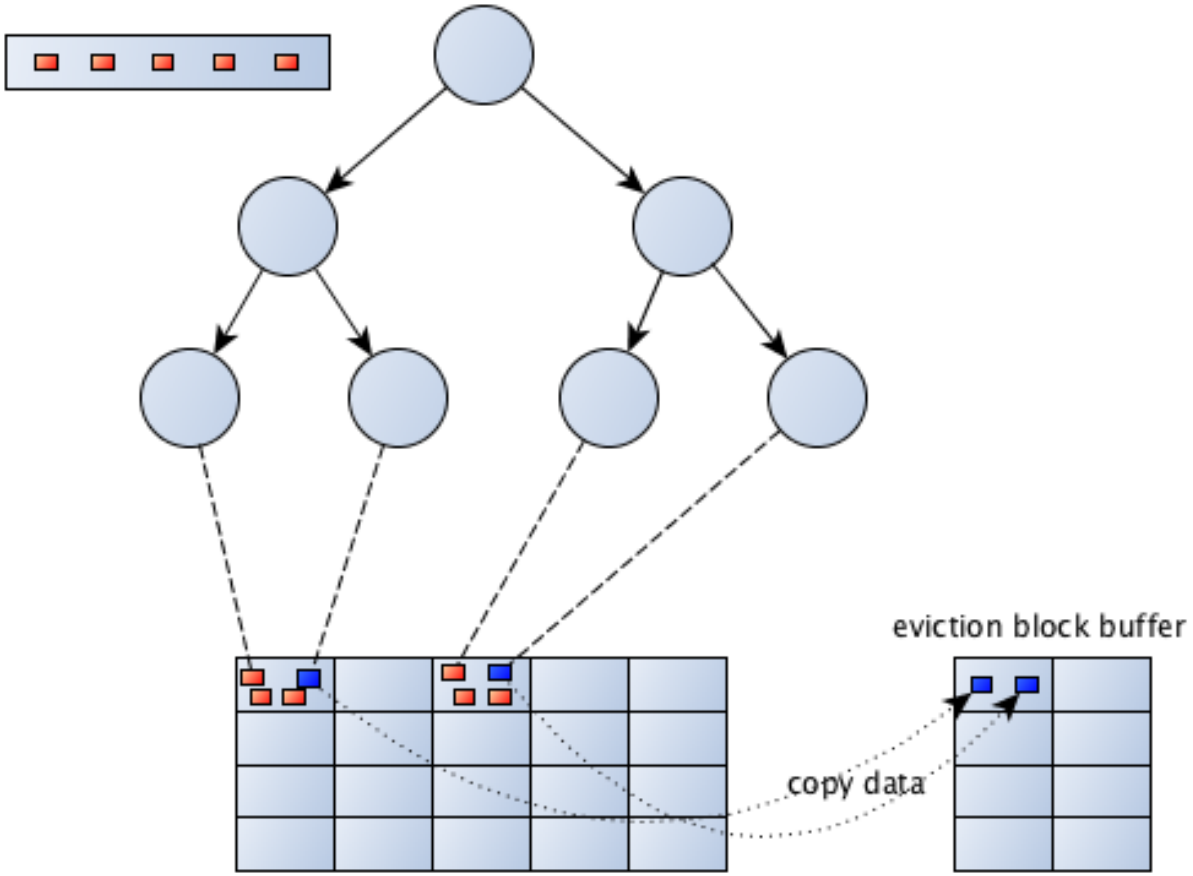
Justin DeBrabant, Andrew Pavlo, Stephen Tu, Michael Stonebraker, and Stan Zdonik. Anti-caching: A New Approach to Database Management System Architecture.


Evicted Table
BlockID
TupleID
1
<offset0>
1
<offset1>
2
<offset>
3
<offset>
...
...
Block Table
BlockID: Table7#1
creation timestamp
tuple length
tuple data
tuple length
tuple data
LRU Chain
TableID#3
T1
T5
T91
T6
...
...
...
T77
T11
T21
LRU Chain
TableID#7
T71
T98
T10
....
...
...
T56
T44
T31
T0

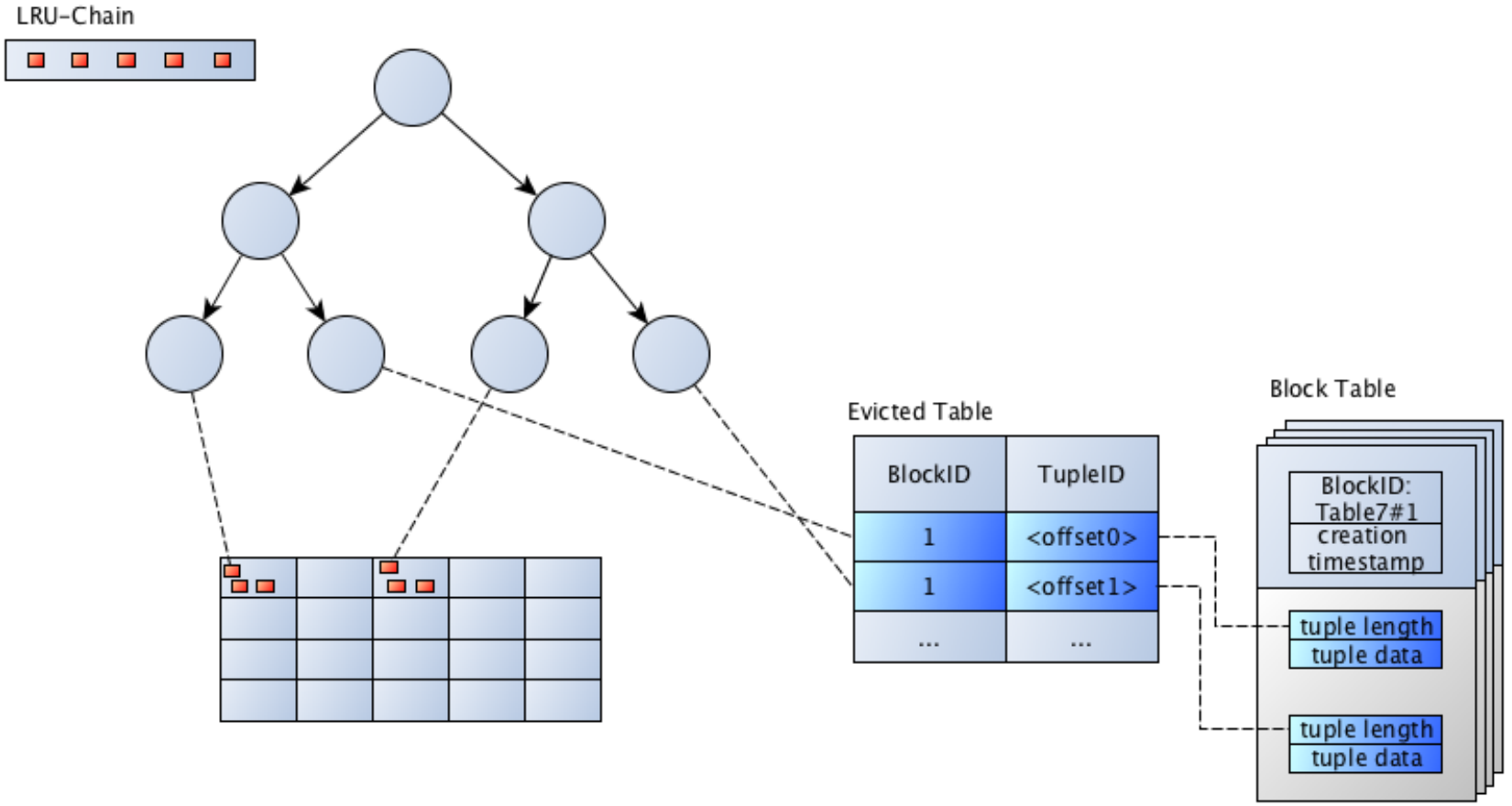
Block Eviction



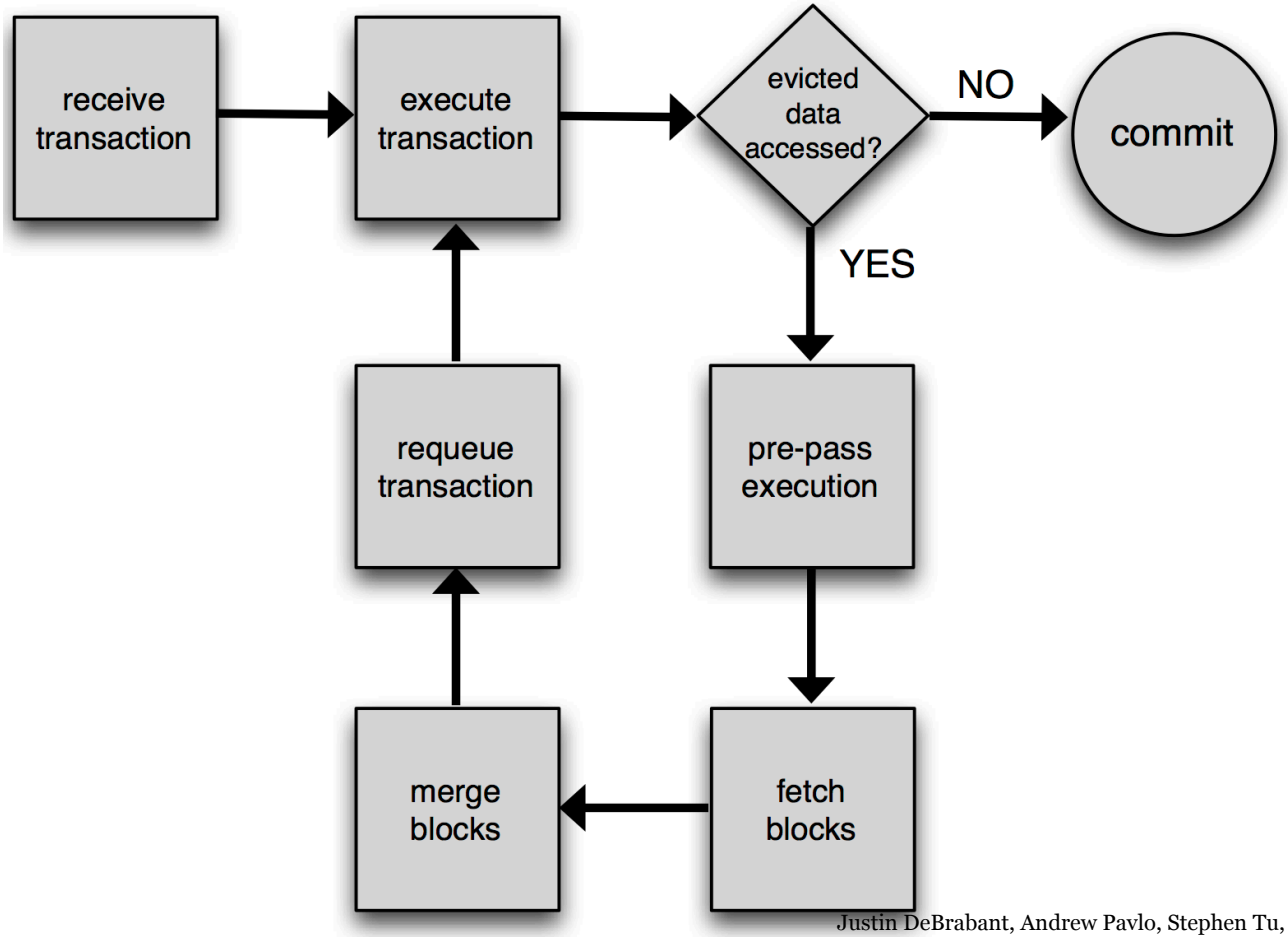
Block Eviction



Block Eviction



Transaction Execution

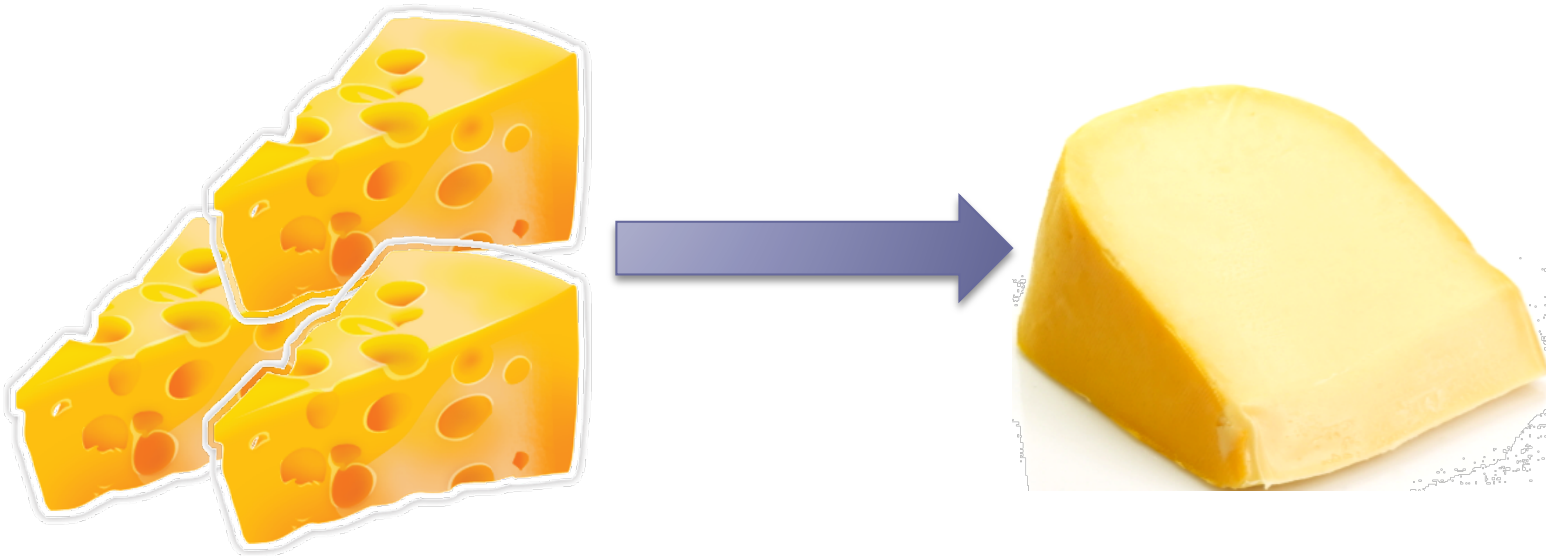


Justin DeBrabant, Andrew Pavlo, Stephen Tu, Michael Stonebraker, and Stan Zdonik. Anti-caching: A New Approach to Database Management System Architecture.

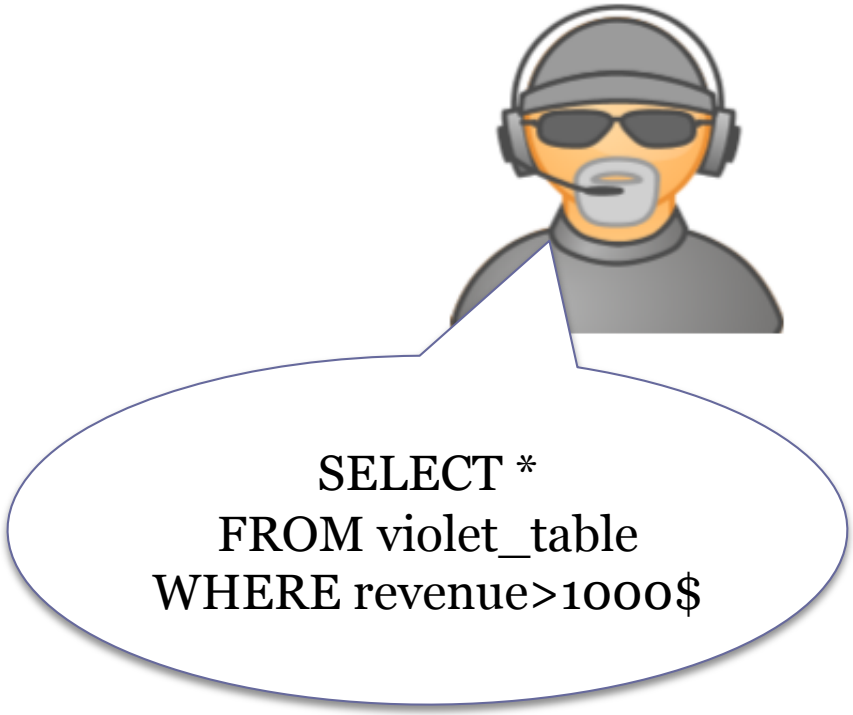
Merging the Blocks

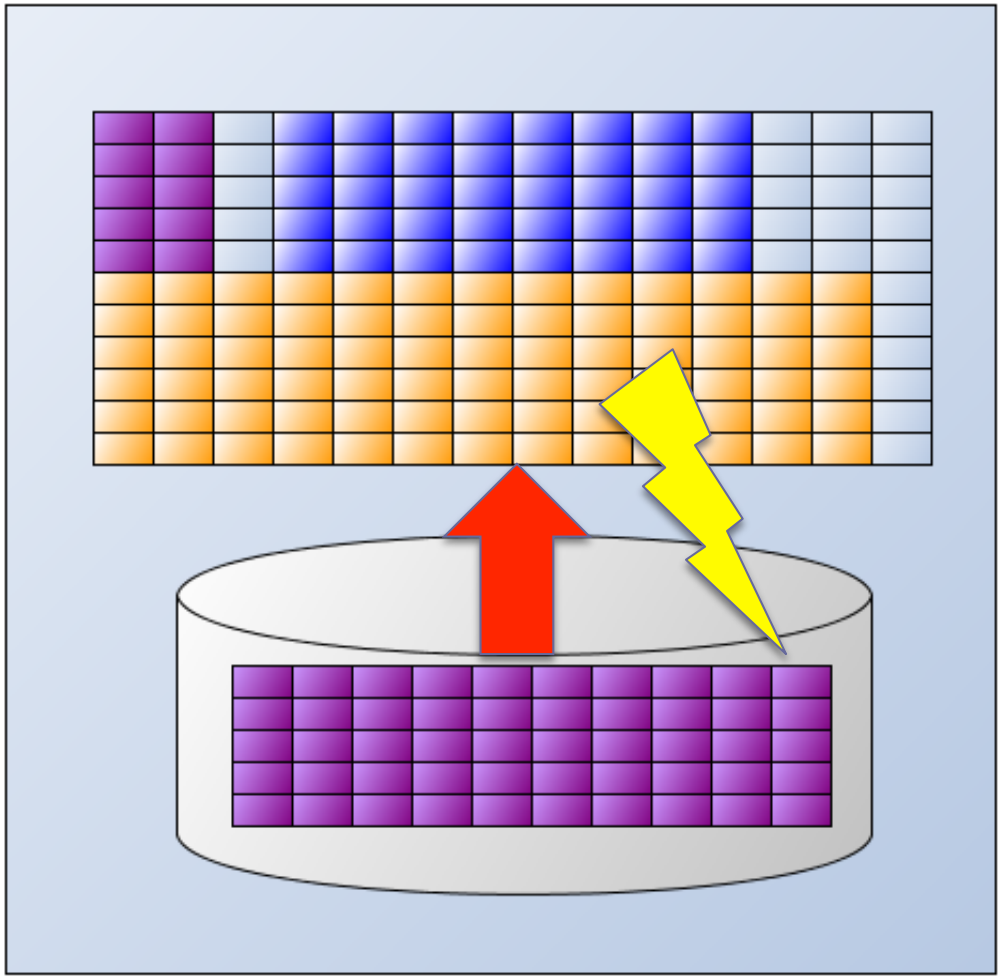
2 possibilities:

- Block Merging
- Tuple Merging



Problem





Comparison

	Stoica et al.	Anti-Caching
hot/cold analysis	offline, after predefined time intervals (~60s), backward algorithm	inline, after predefined time intervals (~5s), LRU
granularity	page level	tuple level
DBMS	VoltDB	H-Store
extensions	access log buffers additional server	book keeping in memory: -evicted table, -block table, -LRU Chain, …
limits	page faults only OLTP workloads	table scan on large tables only OLTP workloads

Conclusion

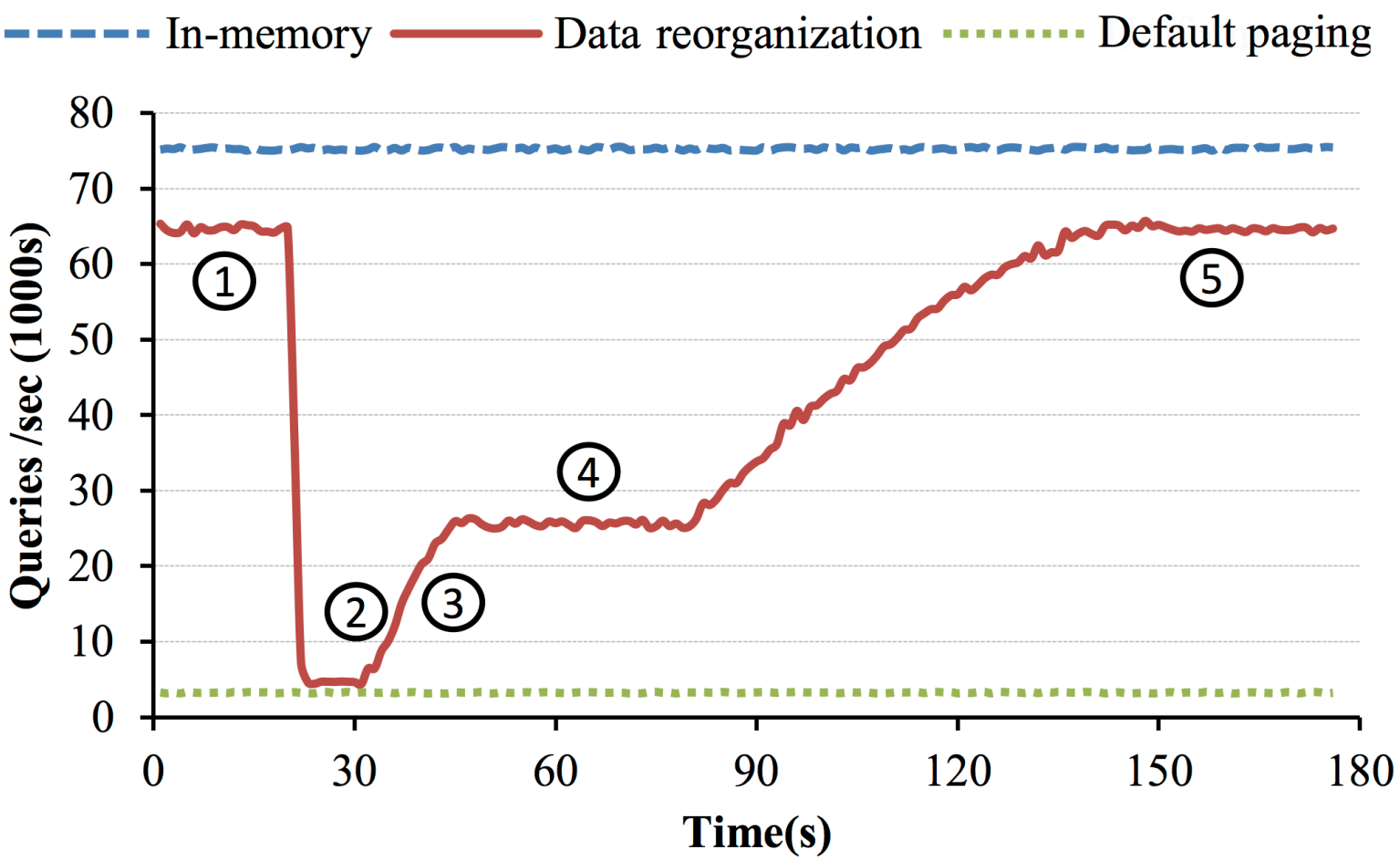
- Well-suited for OLTP workloads
- Size of database increased
- Better energy efficiency
- Stoica et al.
  - lightweight
  - additional server
- Anti-Caching
  - book-keeping overhead

Conclusion

- Highly dependent on workload
- What about OLAP workloads?
  - snapshots and incremental recomputations (?)

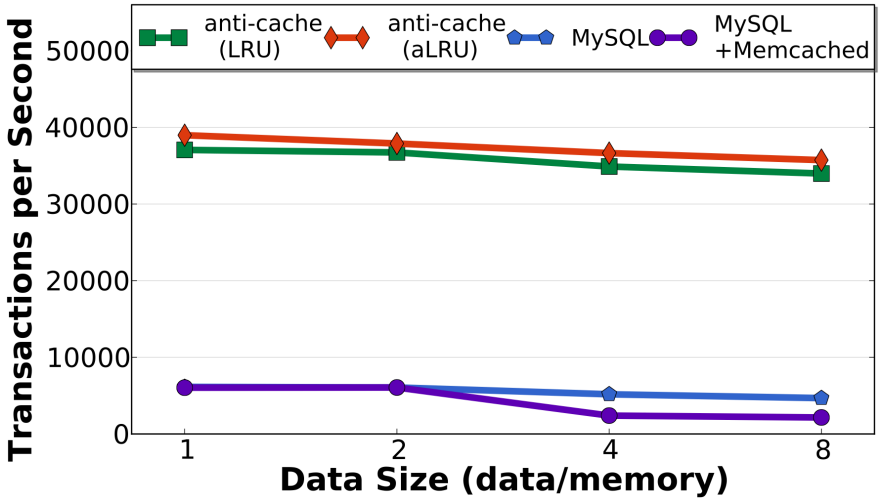
Discussion…

Zipfian Working Set



Radu Stoica and Anastasia Ailamaki. Enabling Efficient OS Paging for Main- memory OLTP Databases.

TPC-C Benchmark "Anti-Caching"



- dual- socket Intel Xeon E5-2620 CPU (12 cores per socket, 15M Cache, 2.00 GHz)
- 7200 RPM disk drive with 7.2 GB/sec for cached reads and about 297 MB/sec for buffered reads

Justin DeBrabant, Andrew Pavlo, Stephen Tu, Michael Stonebraker, and Stan Zdonik. Anti-caching: A New Approach to Database Management System Architecture.