FOEDUS: OLTP Engine for a Thousand Cores and NVRAM

Hideaki Kimura
HP Labs, Palo Alto, CA

Slides By : Hideaki Kimura

Presented By : Aman Preet Singh

Next-Generation Server Hardware?

HP
The Machine

UC Berkeley
Firebox

Intel
Rack Scale
Architecture

...

Differences

- HP Photonics? Infiniband? QPI?
- HP Memristor? PCM? Flash?
- ...

Commonalities

- 1,000s of CPU Cores
- Fast Interconnect
- Huge Low-Latency NVRAM
- Endurance will be an issue

GAP ?

- Traditional OLTP databases do not scale to large number of cores.
- Recent main-memory databases achieve better scalability but they do not efficiently handle NVRAM.

GAP?

- Software should place data wisely.
  - System on a chip.
  - Systems capable of handling cache in-coherrent architectures.
  - Non-uniform Memory access (NUMA) aware systems.

- Aim should be to try to make the data local.

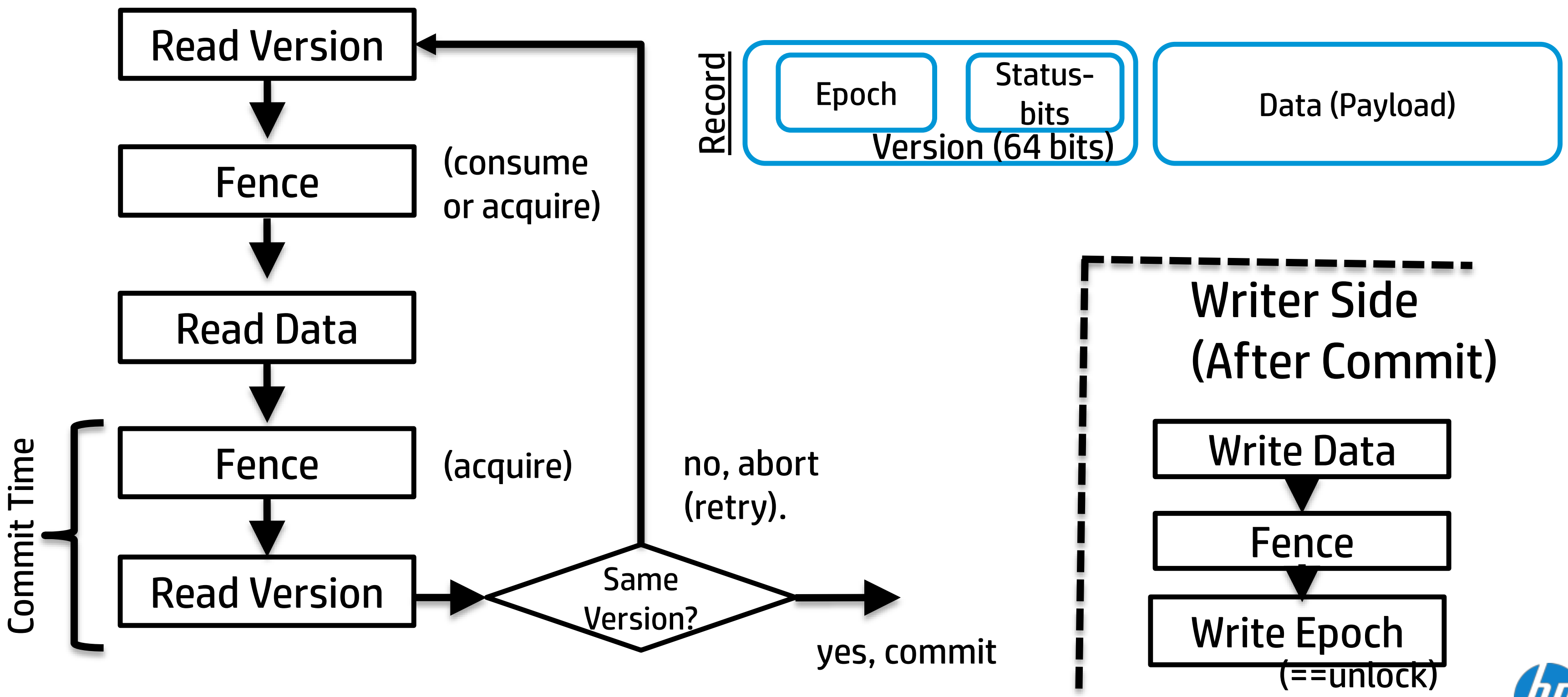
Solution ?

- To provide these capabilities, FOEDUS was built.
  - Ground up database
  - Open source
  - Fully ACID
  - Designed to scale up to thousands of cores
  - Makes the best use of DRAM for OLTP,
  - And NVRAM for OLAP queries.

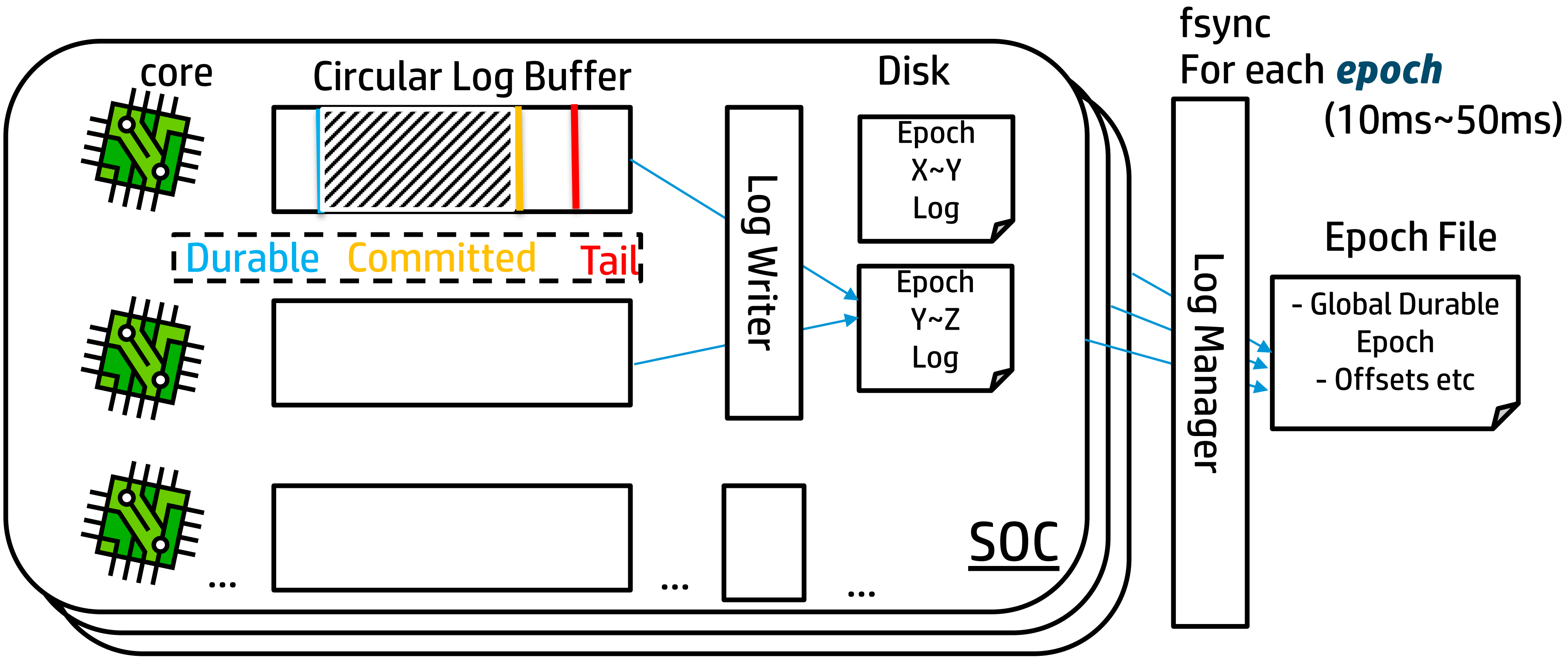
FOEDUS Key Principles

- Bring in-memory speed to NVRAM
  - SILO-like Lightweight Optimistic Concurrency Control
- Dual-Page: physically independent, logically equivalent
  - Mutable Volatile Pages in DRAM
  - Immutable Snapshot Pages in NVRAM
- Master-Tree: Simple and Scalable OCC for NVRAM
  - Masstree + Foster B-Tree + Foster-Twin

SILO: Lightweight/Decentralized OCC [Tu et al]



SILO: Decentralized Logger with Epoch [Tu et al]



SILO's OCC Benefits

✓ **Extremely Lightweight and Scalable**

Block-free (lock-free with retries)

No Write for Reads (cf., read-lock)

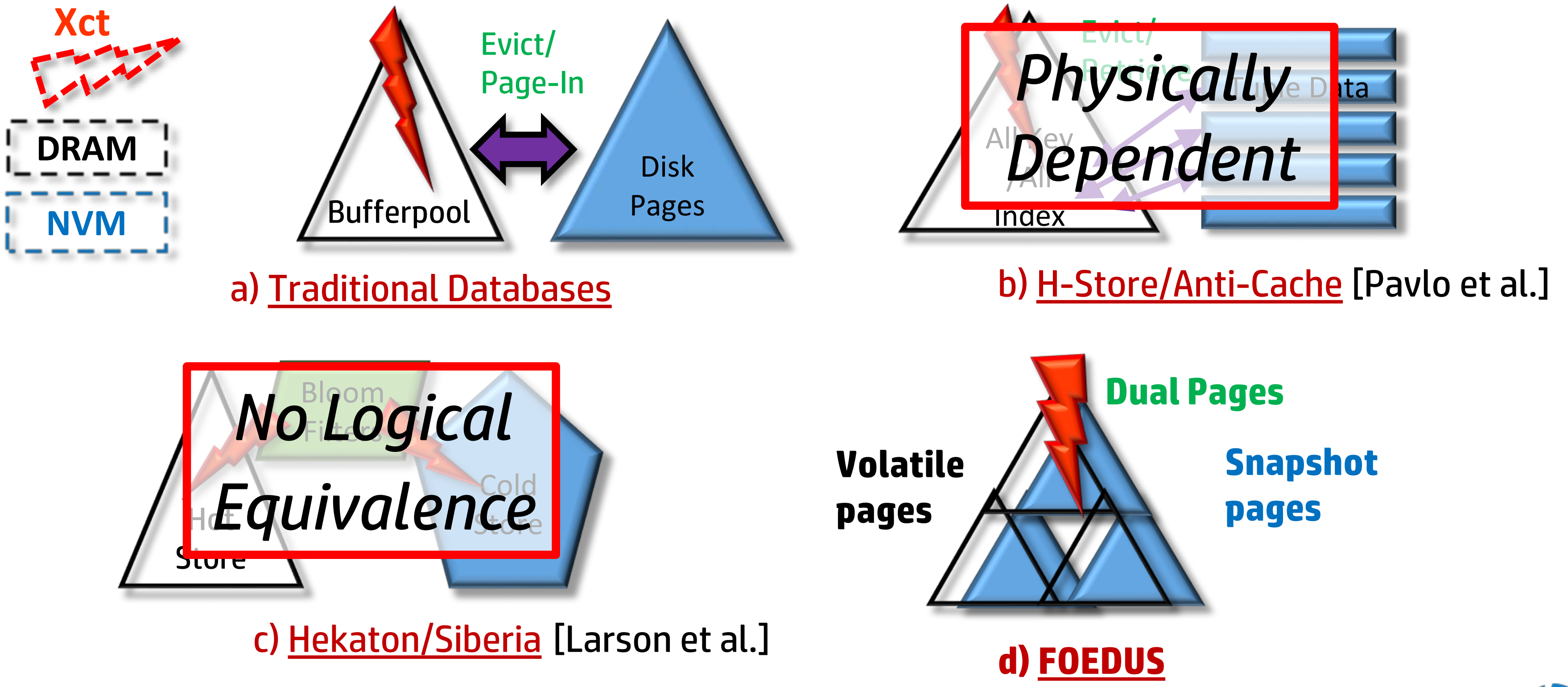
✓ **In-page Lock Mechanism**

No centralized lock manager

Cache Friendly

But, how can we go beyond DRAM?

In-Memory DBMS Beyond DRAM



a) Traditional Databases

b) H-Store/Anti-Cache [Pavlo et al.]

c) Hekaton/Siberia [Larson et al.]

d) **FOEDUS**

Logically Equivalent, Physically Independent Dual

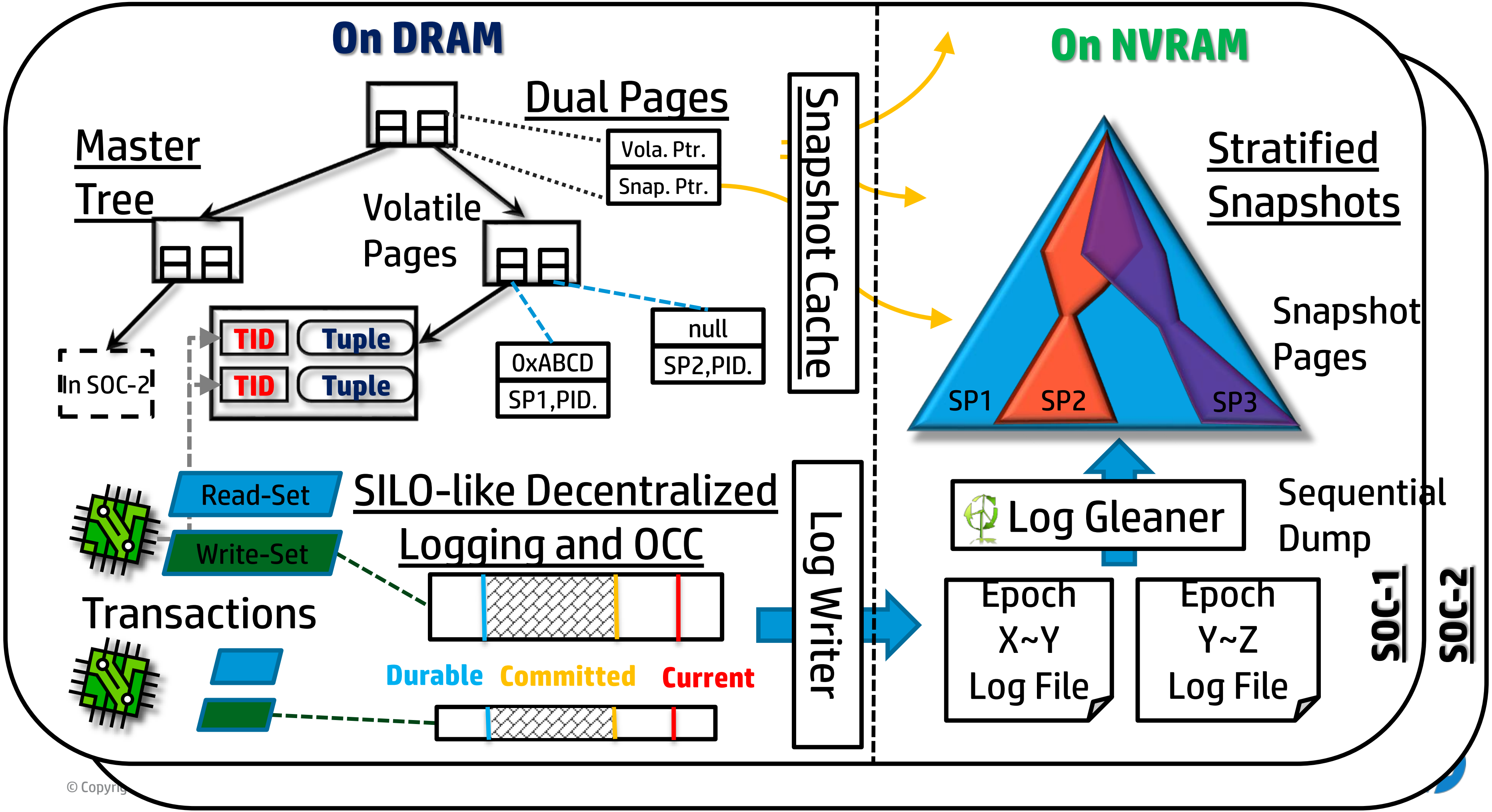
- **Volatile Page: Mutable, on DRAM**
- **Snapshot Page: Immutable, on NVRAM**

Drop Volatile (if X≡Y)

Modify/Add

Install-SP

Volatile Pointer	Snapshot Pointer	
NULL	NULL	: nullptr
X	NULL	: Volatile-Page is just made. No Snapshot yet.
NULL	Y	: Snapshot-Page is latest. No modification.
X	Y	: X is the latest truth, but Y *may* be equivalent.



On DRAM
On NVRAM
Dual Pages
Vola. Ptr.
Snap. Ptr.
Master Tree
Volatile Pages
Snapshot Cache
Stratified Snapshots
In SOC-2
TID
Tuple
TID
Tuple
0xABCD
SP1,PID.
null
SP2,PID.
SP1
SP2
SP3
Snapshot Pages
Read-Set
Write-Set
SILO-like Decentralized Logging and OCC
Transactions
Durable
Committed
Current
Log Writer
Log Gleaner
Sequential Dump
Epoch X~Y Log File
Epoch Y~Z Log File
SOC-1
SOC-2

Constructing Stratified Snapshot from Logical Logs

1. Assign Partitions

↓

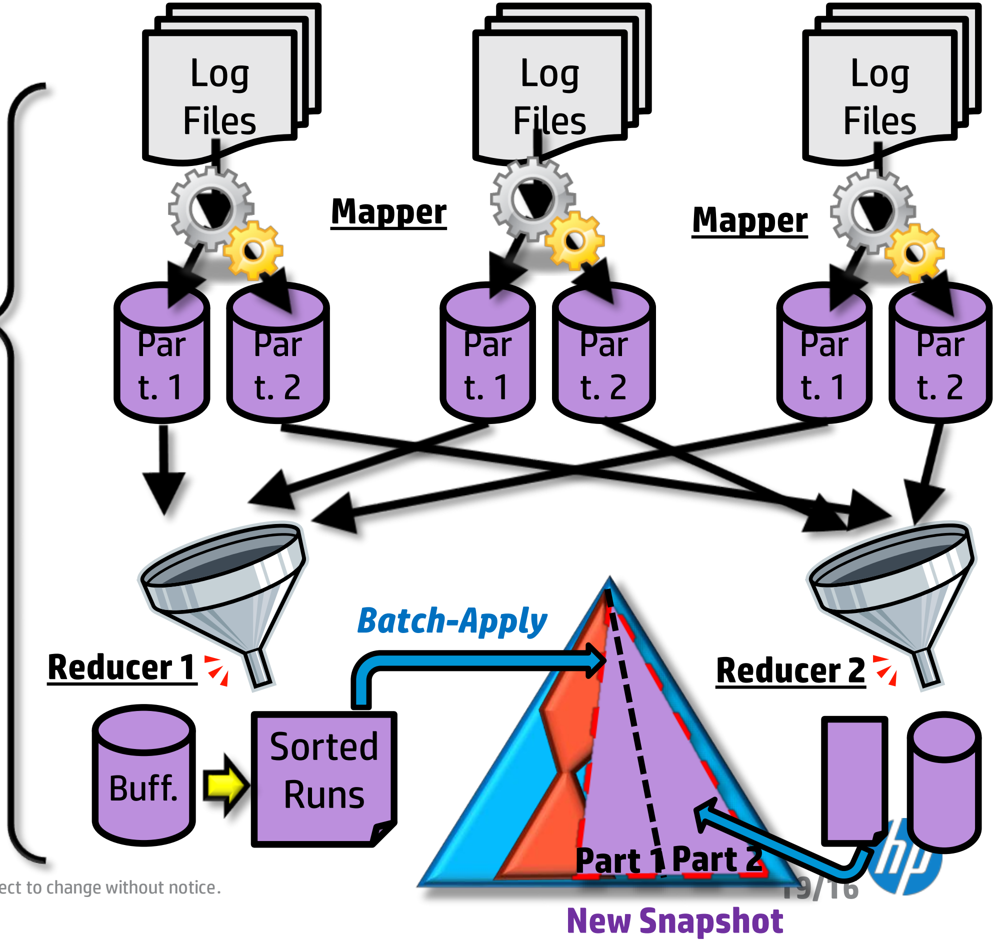
2. Run *Mappers/Reducers*

↓

3. Combine Root/Metadata

↓

4. Install/Drop Pointers



Benefits of Stratified Snapshots

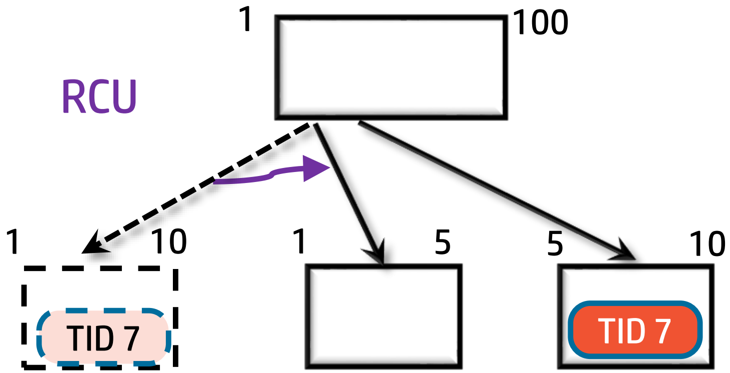
- **Serializable transactions check only a single version of data ↔ LSM-Trees**
- **Drastically more scalable and efficient construction of NVRAM-resident data pages**
  - Separate from transactions/logging
  - Everything Batched, Processed in a tight loop
  - Large Sequential Write only. No frequent flush.

Master-Tree in a Nutshell

- **Mass-tree: *Cache Crafty B-tree* and *OCC* with *in-page* Lock Mechanism.**
- **Foster B-tree: *Single incoming pointer* via *foster-child* to ease page in/out.**
- ***Foster-Twins: Drastically Simplifies OCC and Reduces Aborts/Retries.***

OCC Problem 1: Unnecessary Retries/Aborts

Mass-tree




```
findborder(node root, key k):
retry:    n ← root; v ← stableversion(n)
          if v.isroot is false:
              root ← root.parent; goto retry
descend:  if n is a border node:
              return ⟨n,v⟩
          n′ ← child of n containing k
          v′ ← stableversion(n′)
          if n.version ⊕ v ≤ "locked":   // hand-over-hand validation
              n ← n′; v ← v′; goto descend
          v″ ← stableversion(n)
          if v″.vsplit ≠ v.vsplit:
              goto retry                  // if split, retry from root
          v ← v″; goto descend            // otherwise, retry from n
```

Local Retry!

Global Retry!

Figure 6. Find the border node containing a key.

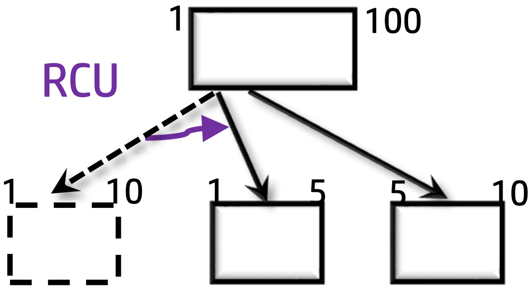
Further, SILO must take Page-Version set and abort if changed by pre-commit.

Lots of Aborts!

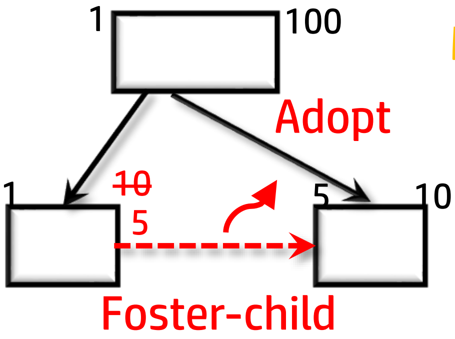
Foster Twins here to Save You!

- Strong Invariants to drastically simplify OCC and eliminate unnecessary aborts/retries
- Still everything in-page
  - No per-tuple GC or compaction/migration
  - No centralized data structure

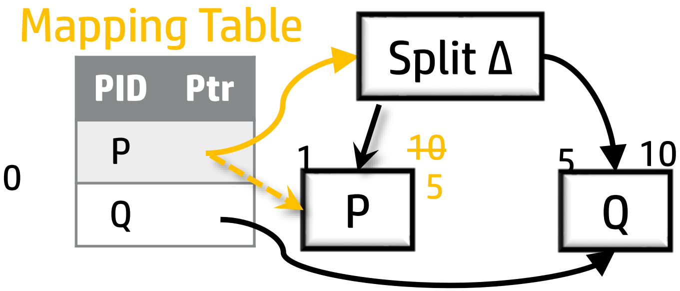
Mass-tree



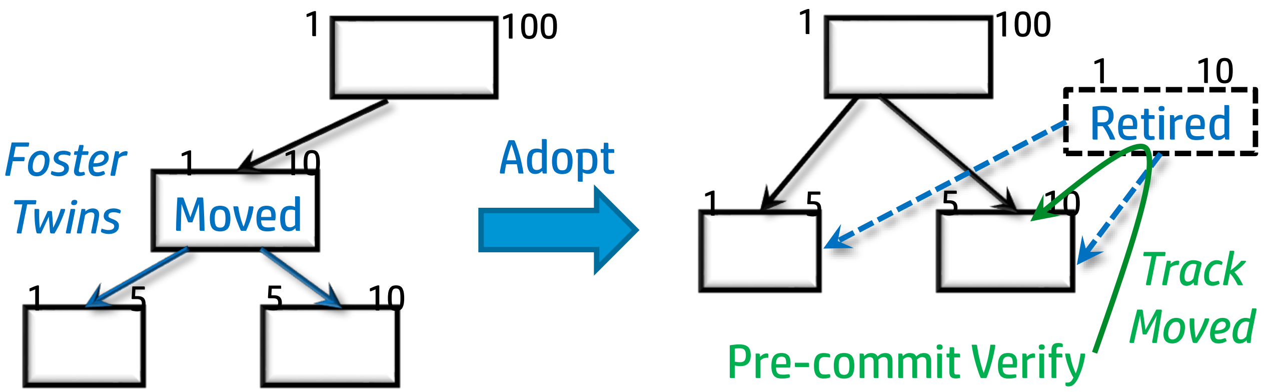
Foster B-Tree



Bw-Tree



Master-Tree



- All data/metadata placed **in-page**
- **Immutable range**
- All information **always track-able**
- All invariants **recursively** hold

Foster-Twins Commit Protocol

Algorithm 1: SILO precommit protocol [13]

Input: R: Read set, W: Write set, N: Node set

`/* Precommit-lock-phase */`
Sort W by unique order;
foreach $w \in W$ **do** Lock w;

Fences, get commit epoch;
`/* Precommit-verify-phase */`
foreach $r, observed \in R$ **do** **if** $r.tid \neq observed$ *and* $r \notin W$ **then** abort;

foreach $n, observed \in N$ **do** **if** $n.version \neq observed$ **then** abort;

Generate TID, apply W, and publish log;

Algorithm 2: FOEDUS precommit protocol

Input: R: Read set, W: Write set, P: Pointer set

`/* Precommit-lock-phase */`
while *until all locks are acquired* **do**
  foreach $w \in W$ **do if** *w.tid.is-moved()* **then** w.tid ← track-moved(w.page, w.record)

  Sort W by unique order;
  foreach $w \in W$ **do** Try lock w. If we fail and find that w.tid.is-moved(), release all locks and retry
end

Fences, get commit epoch;
`/* Precommit-verify-phase */`
foreach $r, observed \in R$ **do**
  if *r.tid.is-moved()* **then** r.tid ← track-moved(r.page, r.record)
  if $r.tid \neq observed$ *and* $r \notin W$ **then** abort;
end
foreach $p \in P$ **do** **if** $p.volatile\text{-}ptr \neq null$ **then** abort;

Generate TID, apply W, and publish log;

Dual Pages: Benefits

✓ **Snapshot Pages are <u>Immutable</u>**

Drastically simplifies Caching/Replication/Traversal/Commit/etc

✓ **Physically <u>Independent</u>**

Transactions never interfere w/ construction/retrieval/eviction of Snapshot Pages

✓ **Logically <u>Equivalent</u>**

- No Bloom Filter or global data needed for Serializability ↔ LSM-Trees
- Volatile Pages <u>guaranteed to</u> contain latest data
- Snapshot Pages <u>guaranteed to</u> be complete *as of snapshot-epoch*

Experiments

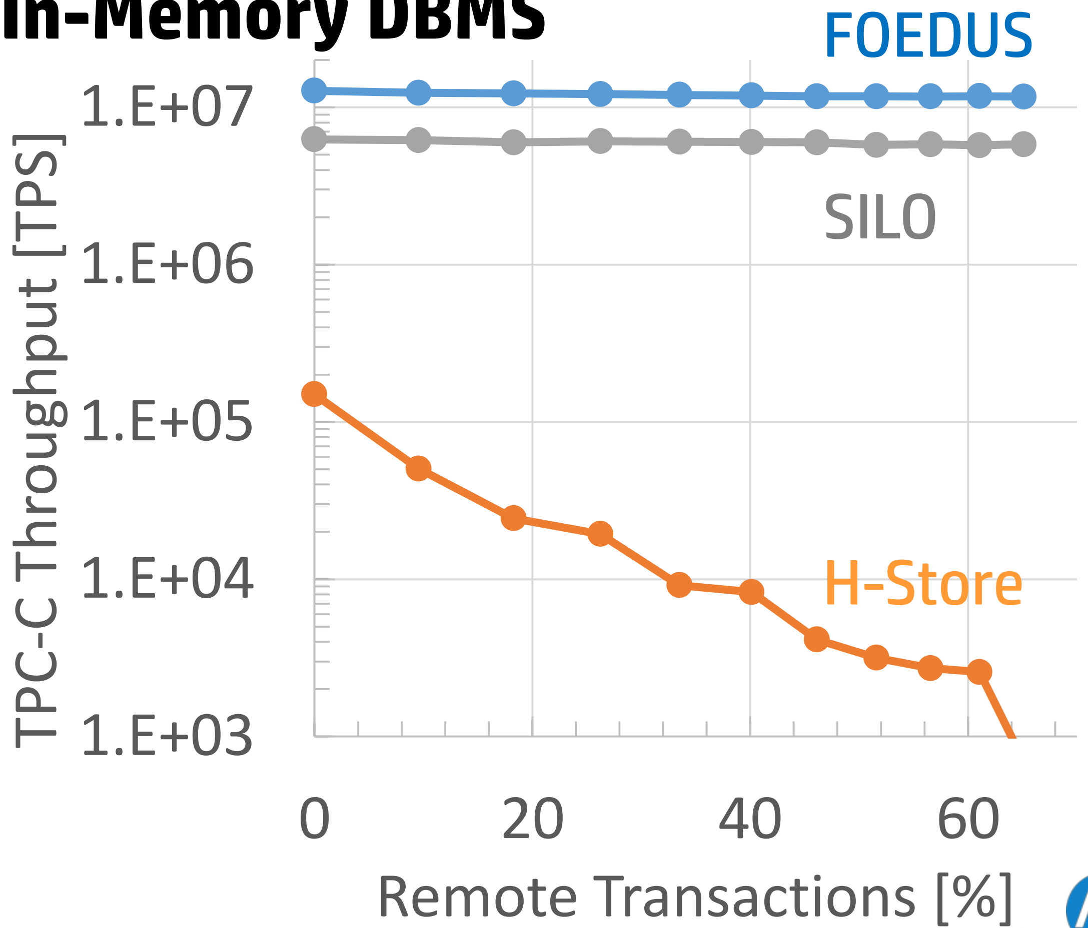
- TPC-C, Serializable
- vs. H-Store, SILO, Shore-MT
- (Traditional DBs (e.g., MySQL) are too slow to compare with.)
- HP Superdome X (DragonHawk):
  16 sockets, 240 cores, 12TB DRAM.
- Emulated NVRAM

Experiment 1: vs In-Memory DBMS

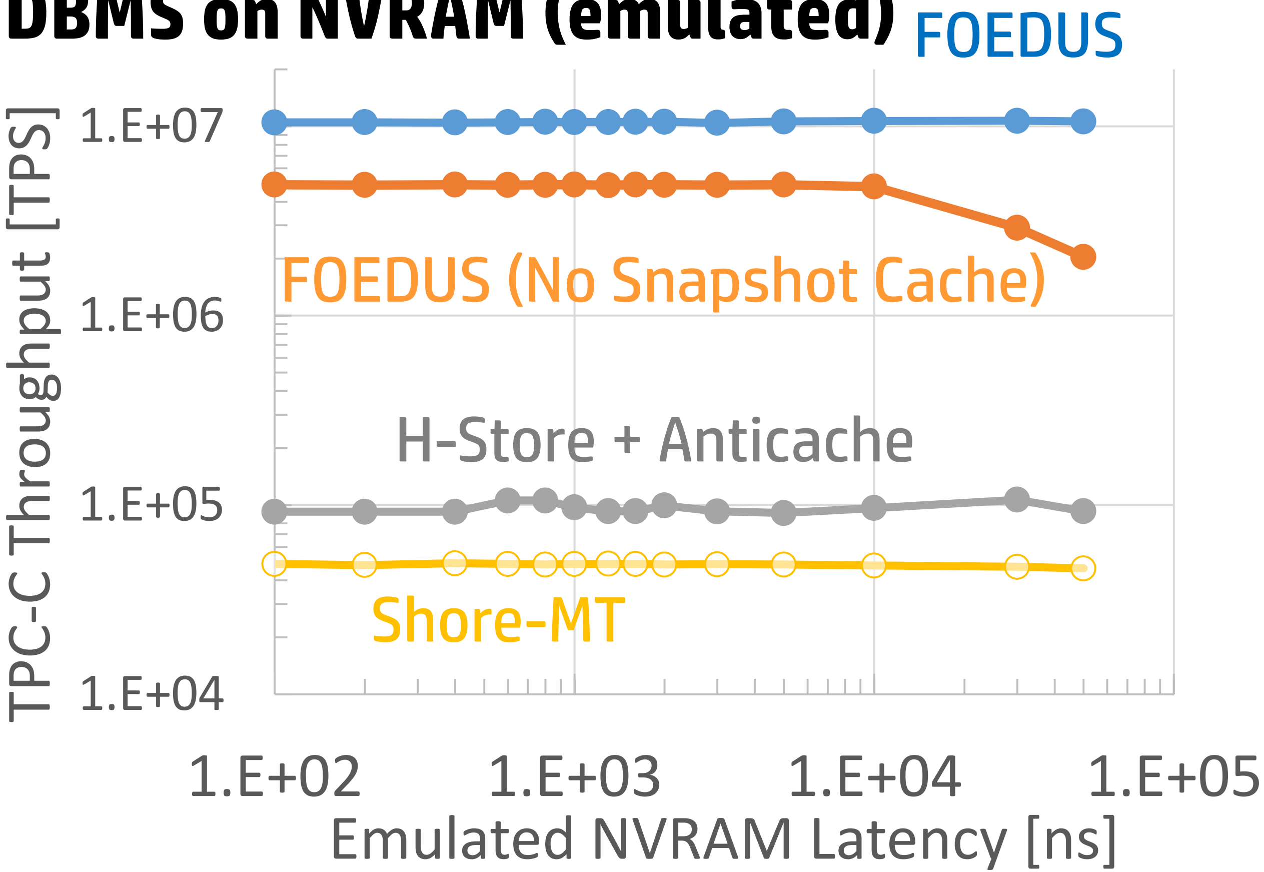
<u>Observations</u>

1. SILO-style OCC much more resilient to contention
2. **100x~** faster than H-Store.
3. More Cores = More Speedup

Cores	Speed-up [/H-Store]
16	33x
60	66x
240	394x



Experiment 2: DBMS on NVRAM (emulated)



<u>Environments</u>

- tmpfs-based NVRAM emulation (varied latency)
- Data/xlog in NVRAM
- H-Store w/ anti-cache

<u>Observations</u>

1. FOEDUS **100x**~ faster
2. Performs best when NVRAM read **<10us**

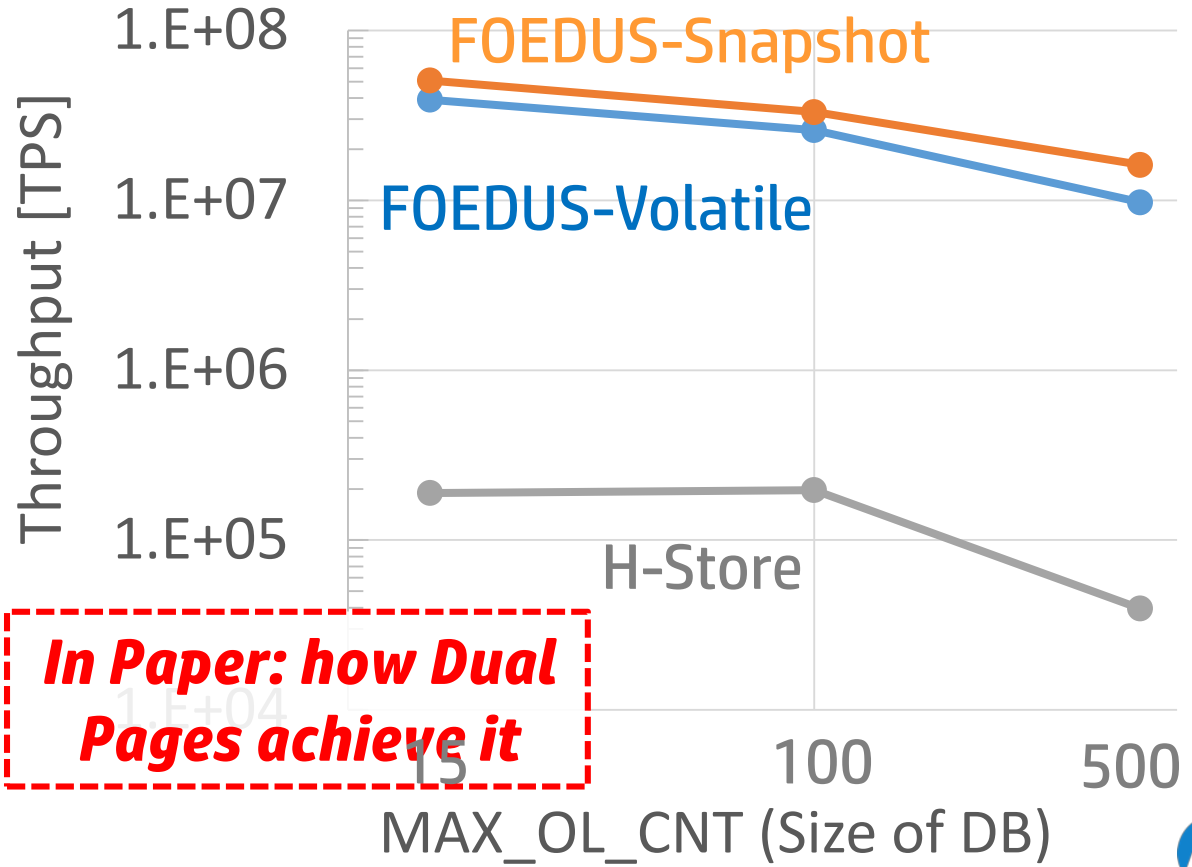
Experiment 3: OLAP Workload

<u>Workload</u>

- 30x Larger Data
- Cursor-Heavy
- Read-Only
- Volatile Pages Only vs Snapshot Pages Only

<u>Observations</u>

1. 100x~ faster than H-Store.
2. FOEDUS runs **faster** when database is cold (!!)



In Paper: how Dual Pages achieve it

Ideas

- The key idea of FEODUS is to use the master-tree model to reduce contention, making OCC lightweight. Can this be extended to distributed memory systems with shared nothing topology, especially where reading remote memory is not very costly, like RDMA?

Reserved Slides

Ongoing Work

- ✓ Try on 1,000 Cores~
- ✓ Further Resilience to Contentions

  Combining Pessimistic Approach When Beneficial
- ✓ SIMD, Xeon-Phi

  Log-Gleaner's Sorting, Page Construction, etc
- ✓ Non-C++ Interface

  SQL/JDBC/ODBC for OLAP. But, for OLTP???

Easy OCC with Foster Twins

❑Simple, Robust, and Efficient Search:

Find a pointer that *__probably__* contains the key;
Follow the page pointer;
If the page's *__key-range__* contain the key: go on;
Else: *__locally__* retry;

No hand-over-hand verification/split counters.
No unnecessary local retry.
Absolutely no global retry.

- No Page-Version-set nor unnecessary aborts: All Records/TIDs always track-able via Foster-Twin!