Avoiding Thread Stalls & Switches in Key-Value Stores: New Latch-Free Techniques and More

By

David Lomet: Retired (dlomet@msn.com)

Rui Wang: Microsoft (bradruiwang@hotmail.com)

What's the Point?

- Data Systems want PERFORMANCE with SCALABILITY
- Data conflicts increase with performance
  - Conflicts lead to stalls and thread switches
- We introduce Notices: **reminiscent of latches– but never *block***
  - due to delta updating and "directing execution"

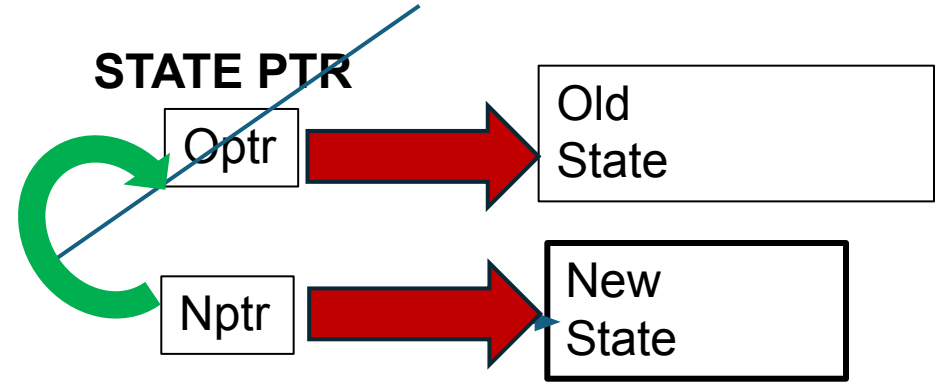
Classic Basics (Compare and Swap)

Data Systems want PERFORMANCE with SCALABILITY

Hard with latches because data conflicts increase with performance

Underlying Latch-free technology

- Compare and Swap (CAS)
- **Traditional Approach**
  - Have pointer to old state: Optr
  - Create new state off-line with a pointer to it: Nptr
  - Use Compare and Swap (**CAS**) to overwrite Optr with Nptr
  - **CAS** switches from Optr to Nptr **atomically**
  - But only if Optr is what is remembered
- Permitting only a single thread's CAS to succeed
- But no thread is BLOCKED!!



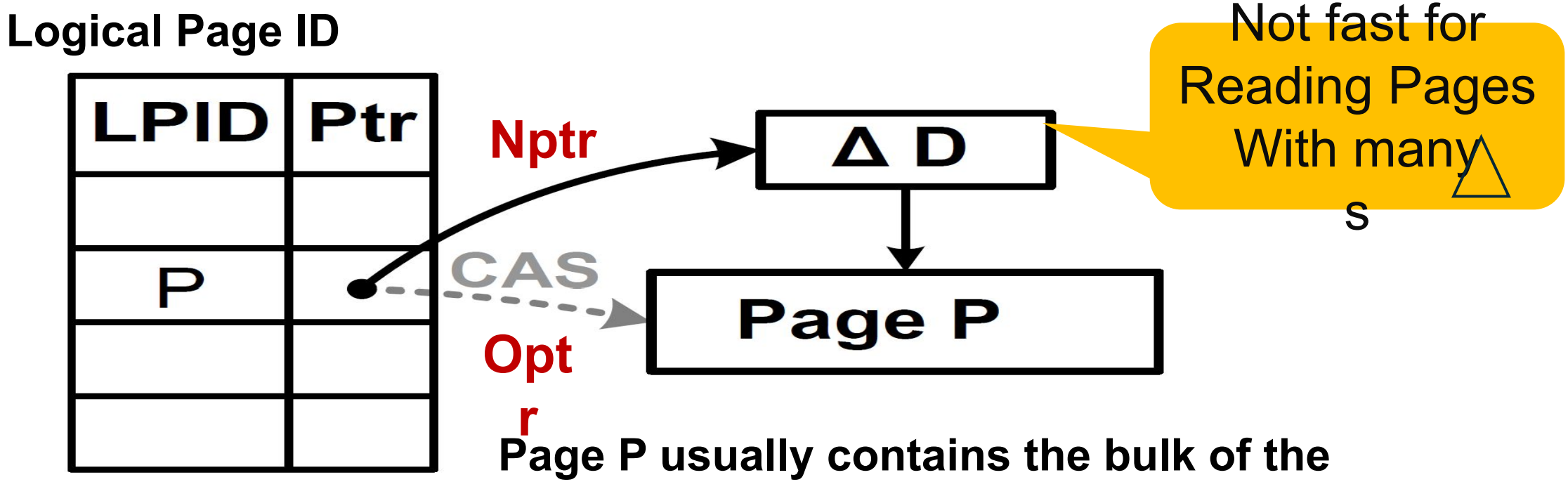
Contention may induce multiple threads to build new state- redundantly

Building new state Can be costly

Delta Updates: very low-cost updating

CAS switches from **Optr** to **Nptr** atomically

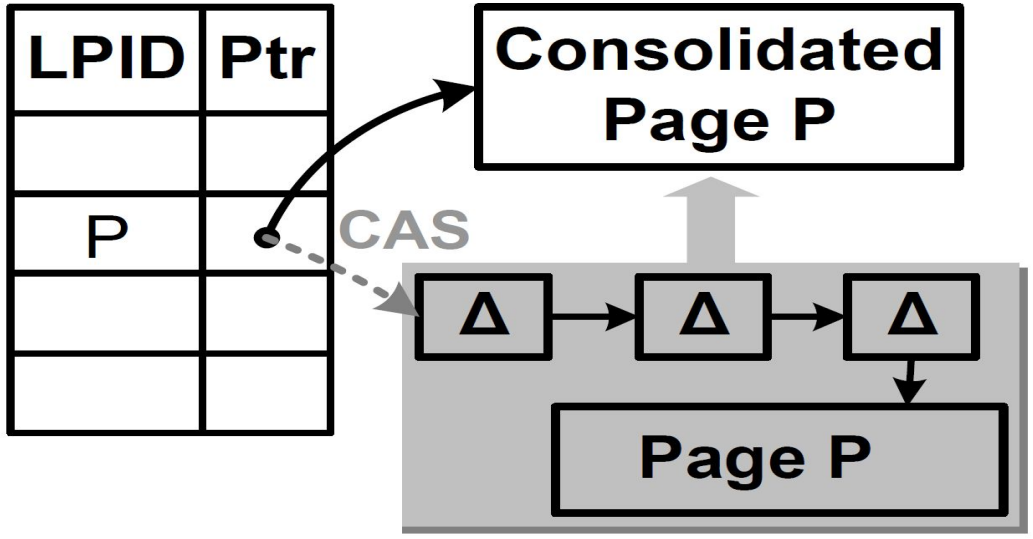
Permitting only one CAS to succeed: VERY FAST for Update



Page P usually contains the bulk of the records – referred to as the "base page"

Consolidating Delta Updates into "Base" Page

- Base page usually contains most of P's records
  - **Base page is not updated in-place.**
  - Rather it is **consolidated** with *delta updates* into a **new base page**
    - Optimized for reading
    - Updating and reading can both continue while consolidation is in progress

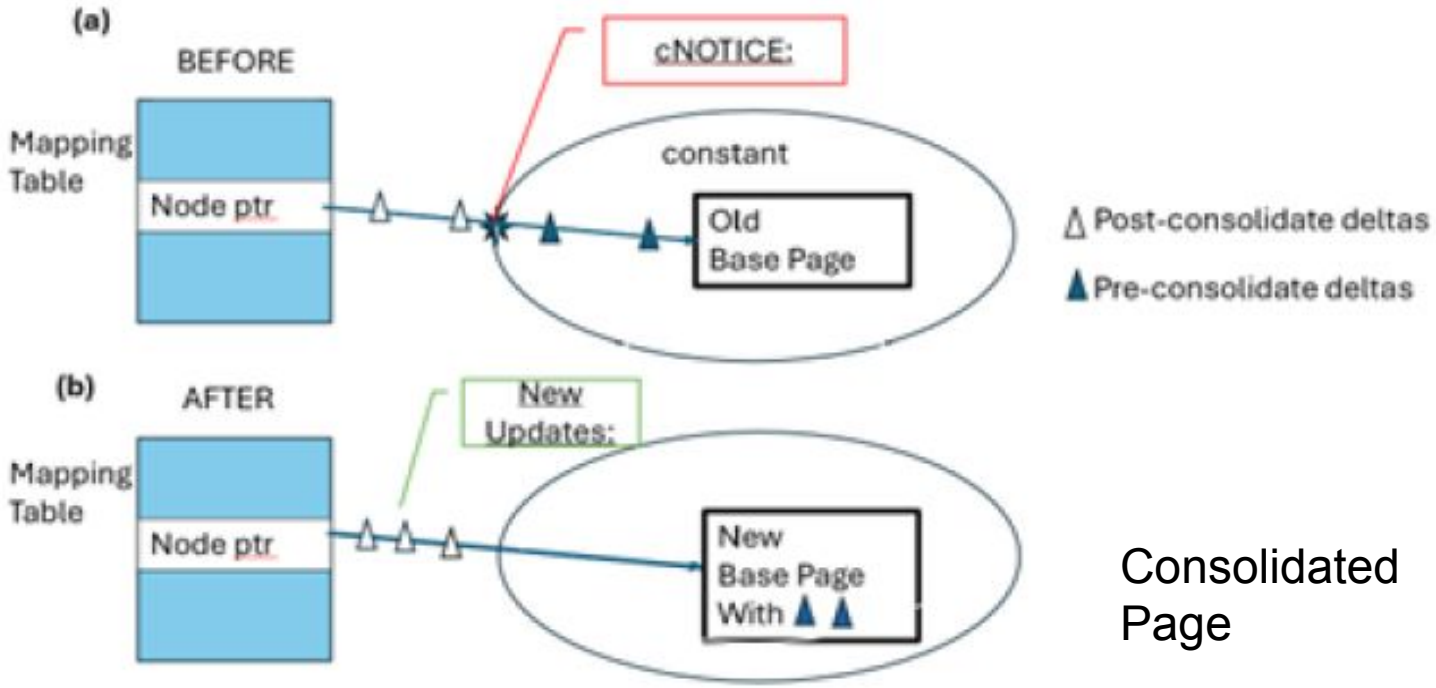


Problem:

1. **P consolidated prior to CAS**
2. Every thread competing to do consolidation builds new page P
3. Only one thread succeeds
4. Other threads have done work in vain

Notices to Avoid Duplicate New States

- **Race to post CONSOLIDATION NOTICE (cNOTICE)**
  - Like competing to set a latch **BUT**
    - **Delta updates permit both reads and updates to continue**
    - **cNOTICE isolates *part of page* being split from further updates**
  - **Only Winner of cNOTICE "race" (CAS) builds consolidated base page**
  - **Lo[illegible] work**

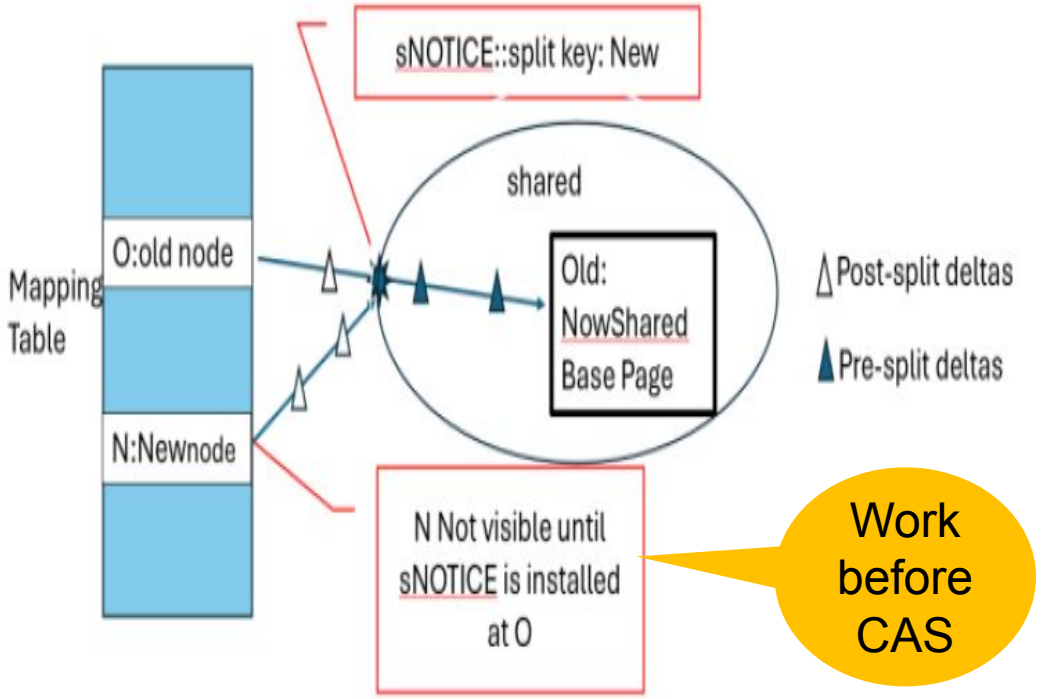


Consolidated Page

Latch-free B-tree Page Splits

Prior to *Conflict Resolution* with sNOTICE Split- after New Node PTR Allocation

- **Posting Notice analogous to setting a latch**
  - **But Delta updates and Reads continue**
  - **Like latches, Notices protect (part of) page stat**
- **No data movement**
  - Allocate N in Mapping Table
  - Install sNOTICE at N
  - sNOTICE provides: <split key, N>
- **sNOTICE at O with CAS provides conflict resolution**
  - **This logically splits node O**
  - Directing updates to N that are more than split key
  - N not visible until sNOTICE installed at O
  - After CAS, data splits between O & N
- **O's shared page is read-only, protected by sNOTICE**
  - Just as a latch might protect it
  - Again, delta updates permit continuous operation executior
- **We can now divide records between O and N**
  - WITHOUT MONDIFYING O
  - Rather, we replace O with new O without N records



Additional Latch-Free Thoughts

- **Storage management done in Epochs**
  - IT IS HARD TO GET A COMPLETELY FREE LUNCH
    - Perhaps only a ***good low-cost lunch***
  - No simple allocation and delete
  - **Epochs tell us when storage we wish to free is no longer visible**
  - At intervals, we close old epoch, begin new
  - Code runs in epochs- allocating/freeing storage in epochs

Contributions from Justin Levandoski

Not all Latch-free opportunities are in access method nodes

- **Log Structured Store:** Buffer batches page writes
  - Thread switch & I/O latency only for writing batch– not for each page
  - Using Fetch & Increment atomic instruction to fill buffer
    Which is faster than CAS and permits multiple winners
  - Use batching as much as possible– our **LSS** (a form of LFS)

Contributions from Sudipta Sengupta

Short list of References

- David B. Lomet, Rui Wang: Avoiding Thread Stalls and Switches in Key-Value Stores: New Latch-Free Techniques and More. CoRR abs/2601.00208 (2026)
- Rui Wang, Xinjun Yang, Feifei Li, David B. Lomet, Xin Liu, Panfeng Zhou, Yongxiang Chen, David Zhang, Jingren Zhou, Jiesheng Wu: Bwe-tree: An Evolution of Bw-tree on Fast Storage. ICDE 2024: 5266-5279
- Justin J. Levandoski, David B. Lomet, Sudipta Sengupta, Adrian Birka, Cristian Diaconu: Indexing on modern hardware: Hekaton and beyond. SIGMOD Conference 2014: 717-720
- Justin J. Levandoski, David B. Lomet, Sudipta Sengupta: LLAMA: A Cache/Storage Subsystem for Modern Hardware. Proc. VLDB Endow. 6(10): 877-888 (2013)
- Justin J. Levandoski, David B. Lomet, Sudipta Sengupta: The Bw-Tree: A B-tree for new hardware platforms. ICDE 2013: 302-313