mongoDB

Thanks for joining:

A Technical Introduction to WiredTiger

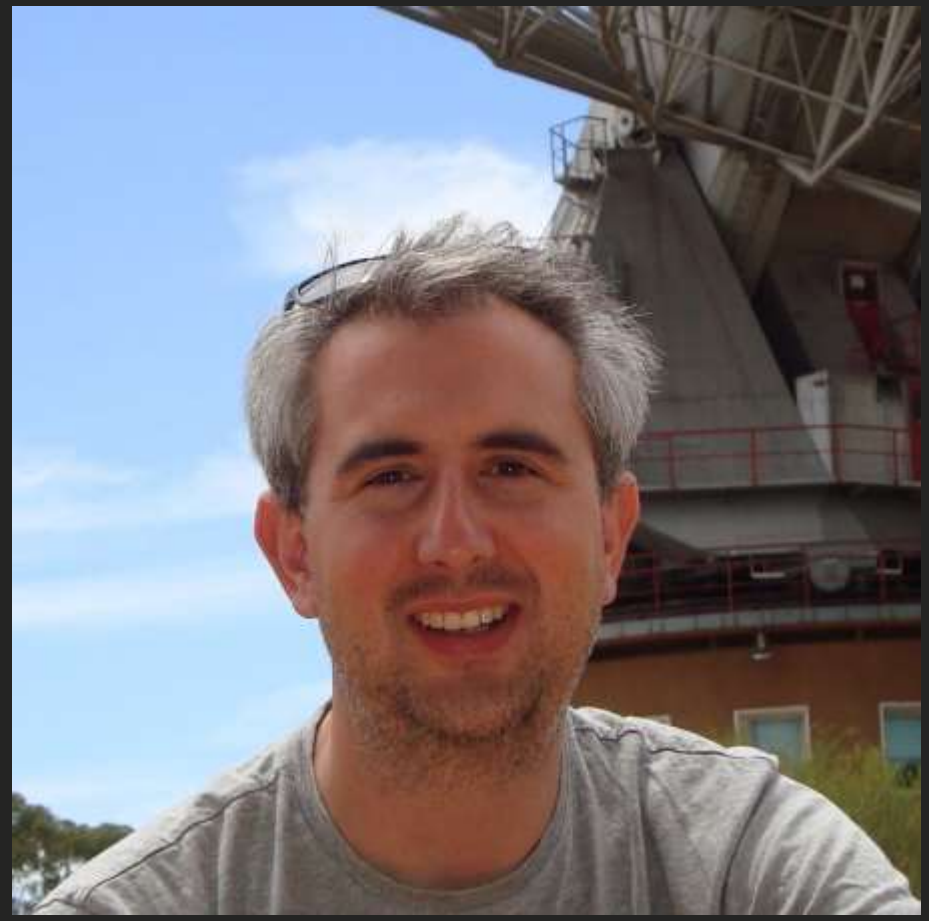
Michael Cahill
Director of Engineering (Storage), MongoDB

We'll begin shortly.

About The Presenter

- Michael Cahill
- WiredTiger team lead at MongoDB

- Sydney, Australia
- michael.cahill@mongodb.com
- @m_j_cahill



What You Will Learn

- WiredTiger Architecture
- In-memory performance
- Document-level concurrency
- Compression and checksums
- Durability and the journal
- What’s next?

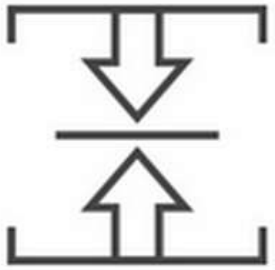
This webinar is not…

- How to write stand-alone WiredTiger apps
  – contact info@wiredtiger.com if you are interested
- How to configure MongoDB with WiredTiger for your workload

You may have seen this:

MongoDB 3.0 Now Available

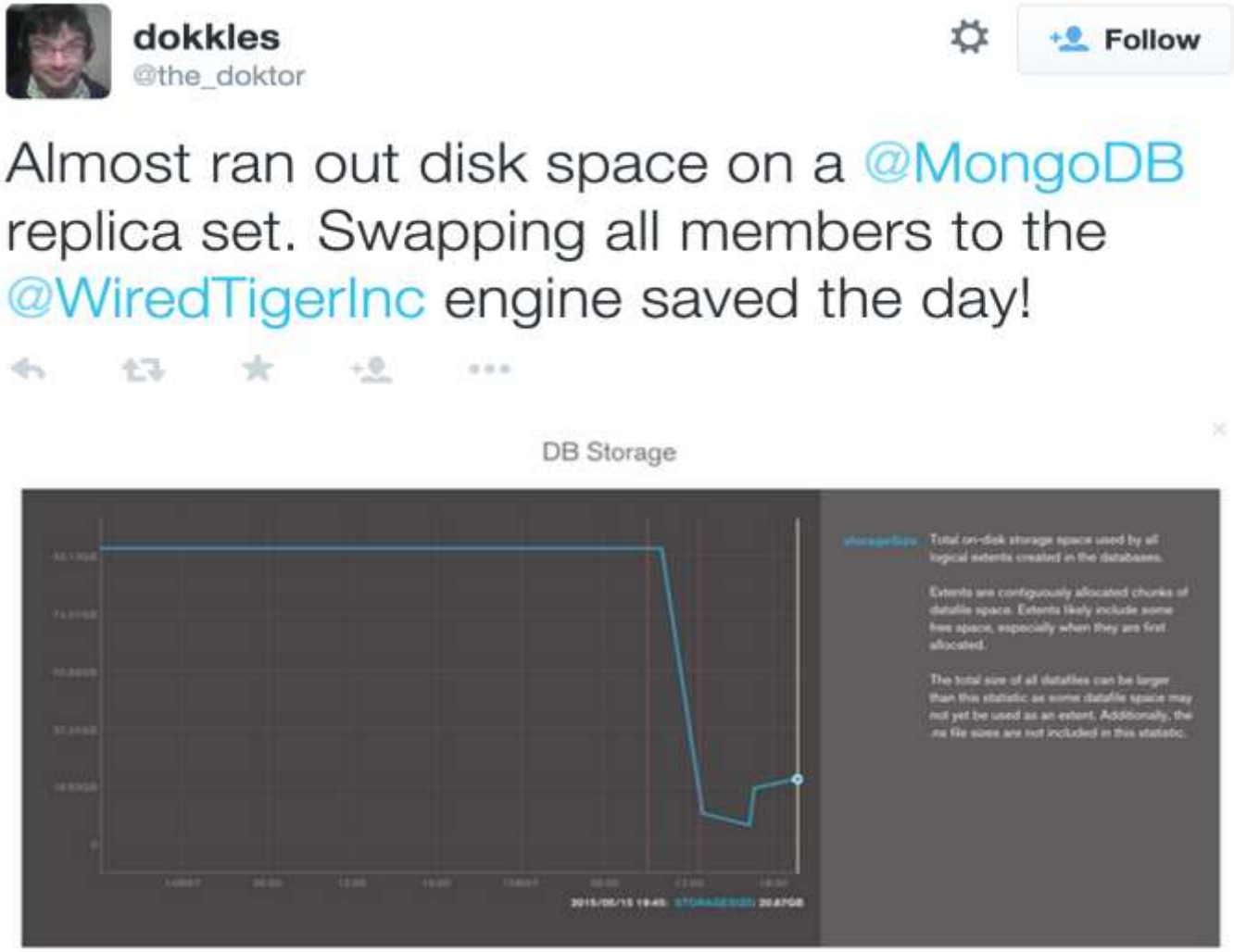
7x-10x Better Performance



80% Less Storage

95% Reduction in Ops

or this…


dokkles
@the_doktor
Follow
Almost ran out disk space on a @MongoDB replica set. Swapping all members to the @WiredTigerInc engine saved the day!
DB Storage

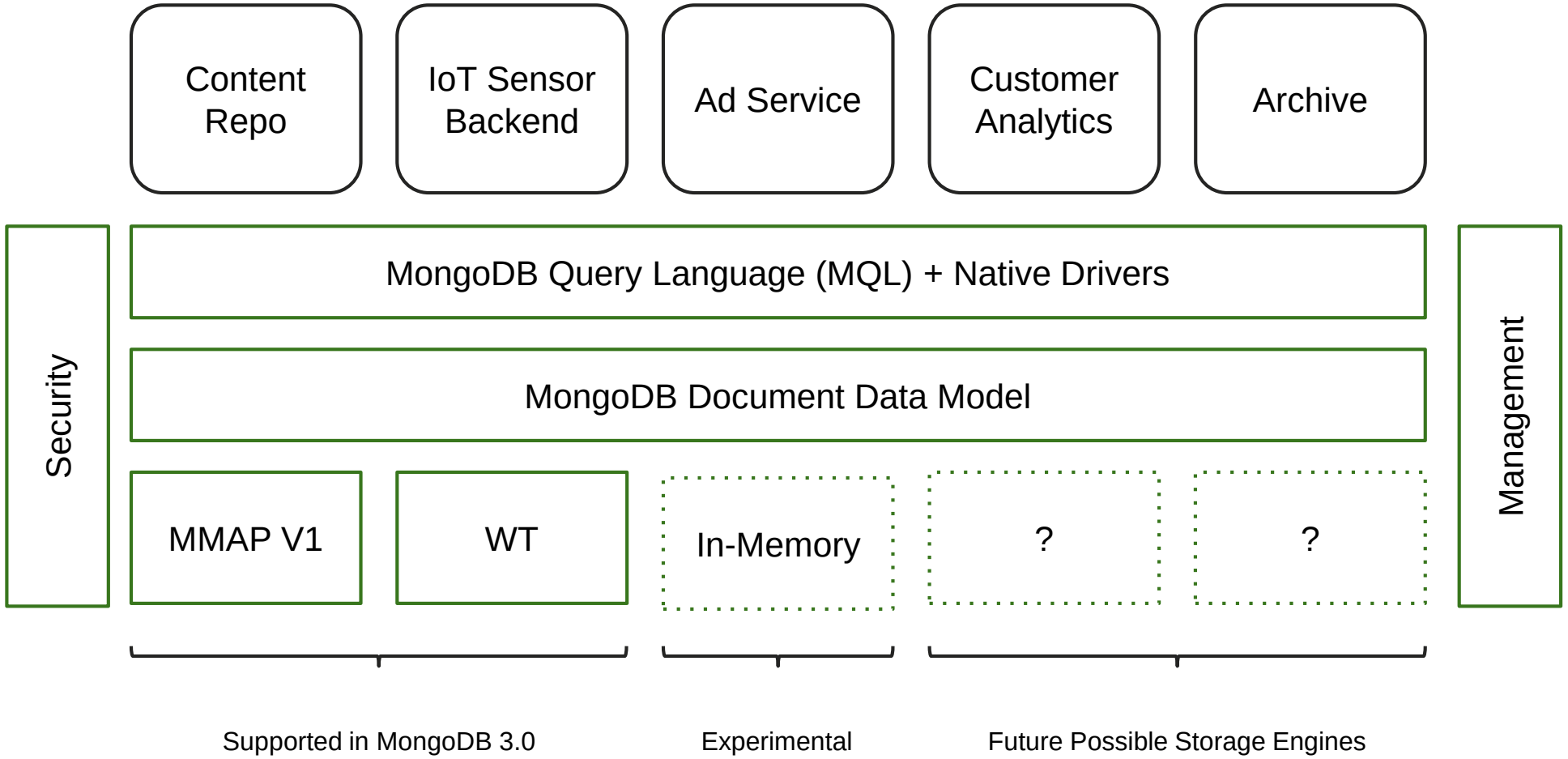
What You Will Learn

- ➢ WiredTiger Architecture
- In-memory performance
- Document-level concurrency
- Compression and checksums
- Durability and the journal
- What’s next?

MongoDB's Storage Engine API

- Allows different storage engines to "plug-in"
  - different workloads have different performance characteristics
  - mmap is not ideal for all workloads
  - more flexibility
    - mix storage engines on same replica set/sharded cluster
    - MongoDB can innovate faster
- Opportunity to integrate further (HDFS, native encrypted, hardware optimized …)
- Great way for us to demonstrate WiredTiger's performance

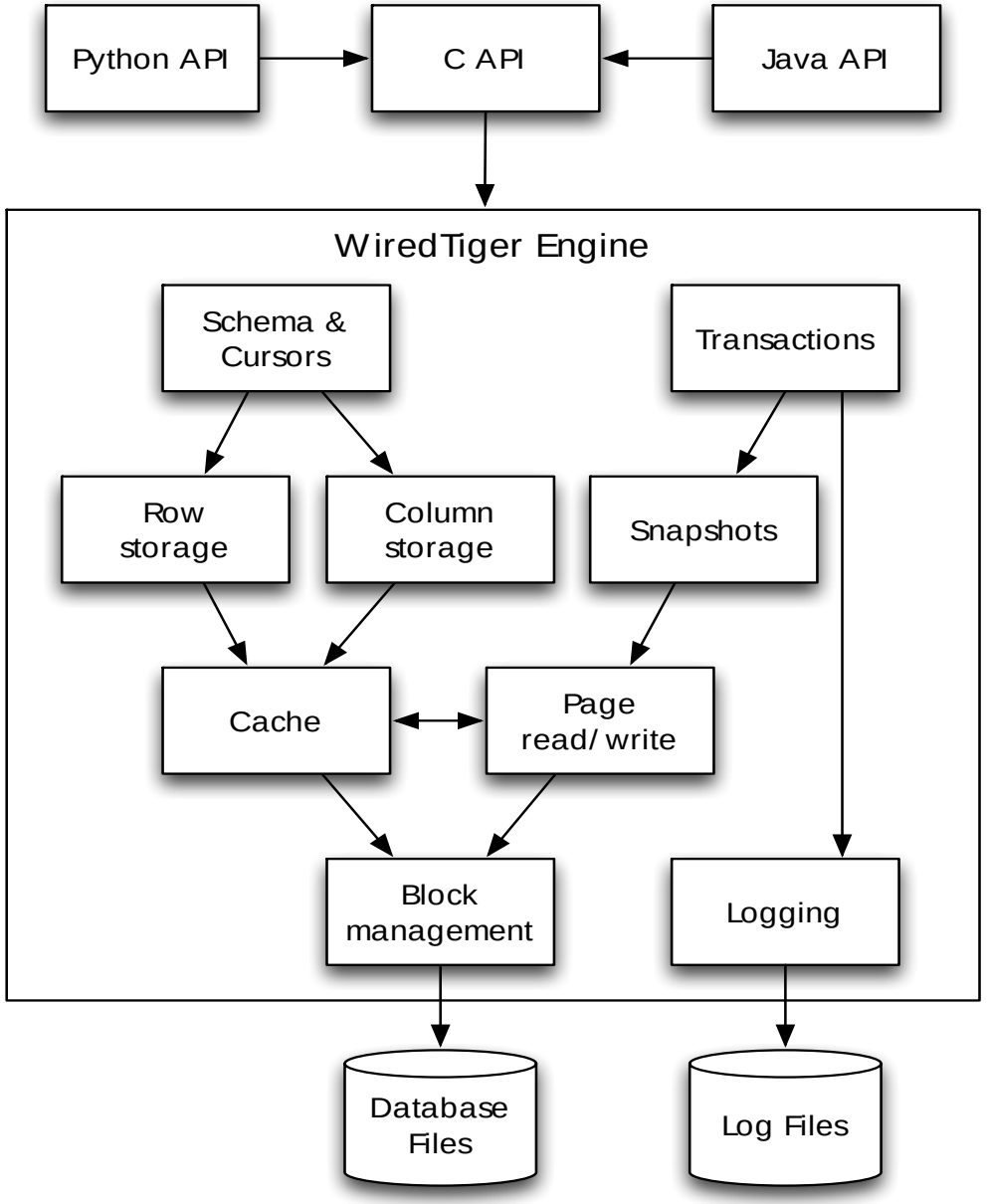
Storage Engine Layer



Motivation for WiredTiger

- Take advantage of modern hardware:
  - many CPU cores
  - lots of RAM
- Minimize contention between threads
  - lock-free algorithms, e.g., hazard pointers
  - eliminate blocking due to concurrency control
- Hotter cache and more work per I/O
  - compact file formats
  - compression

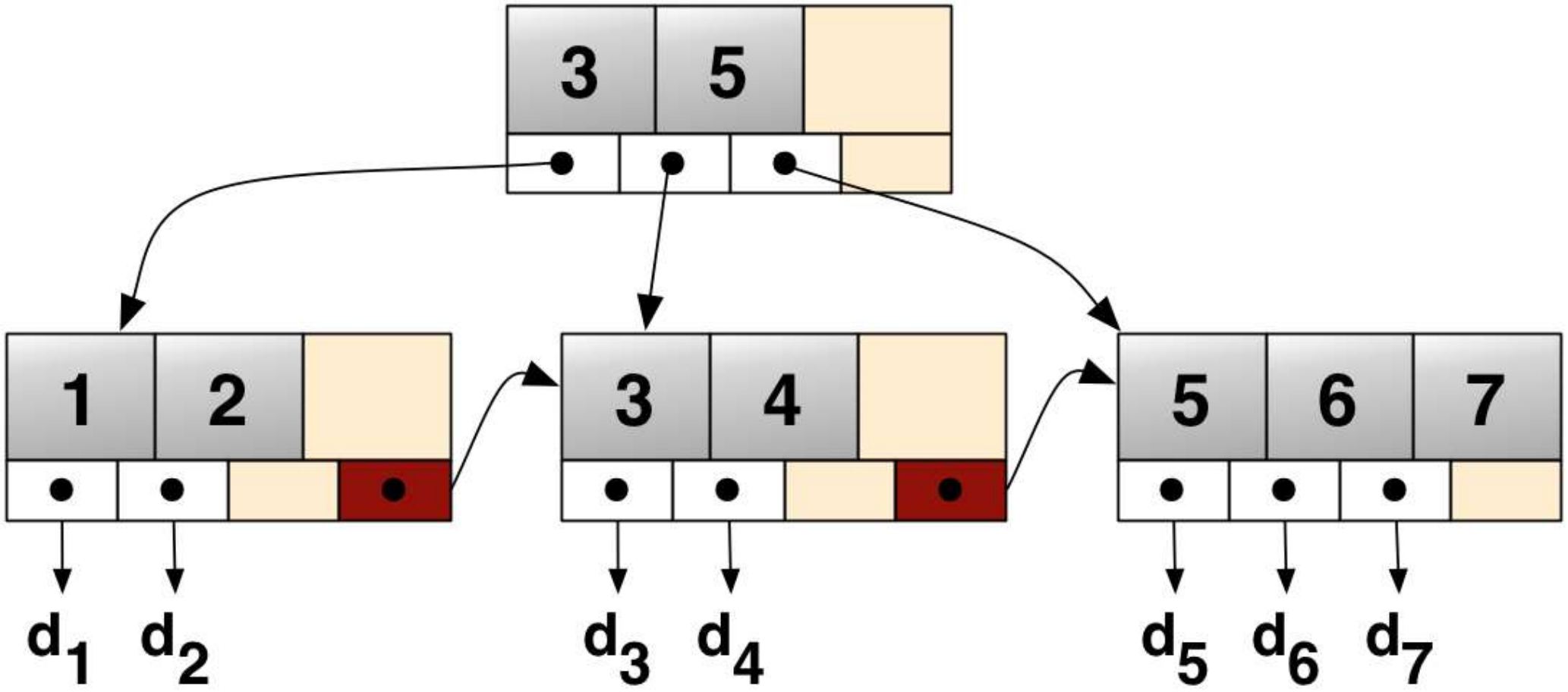
WiredTiger Architecture



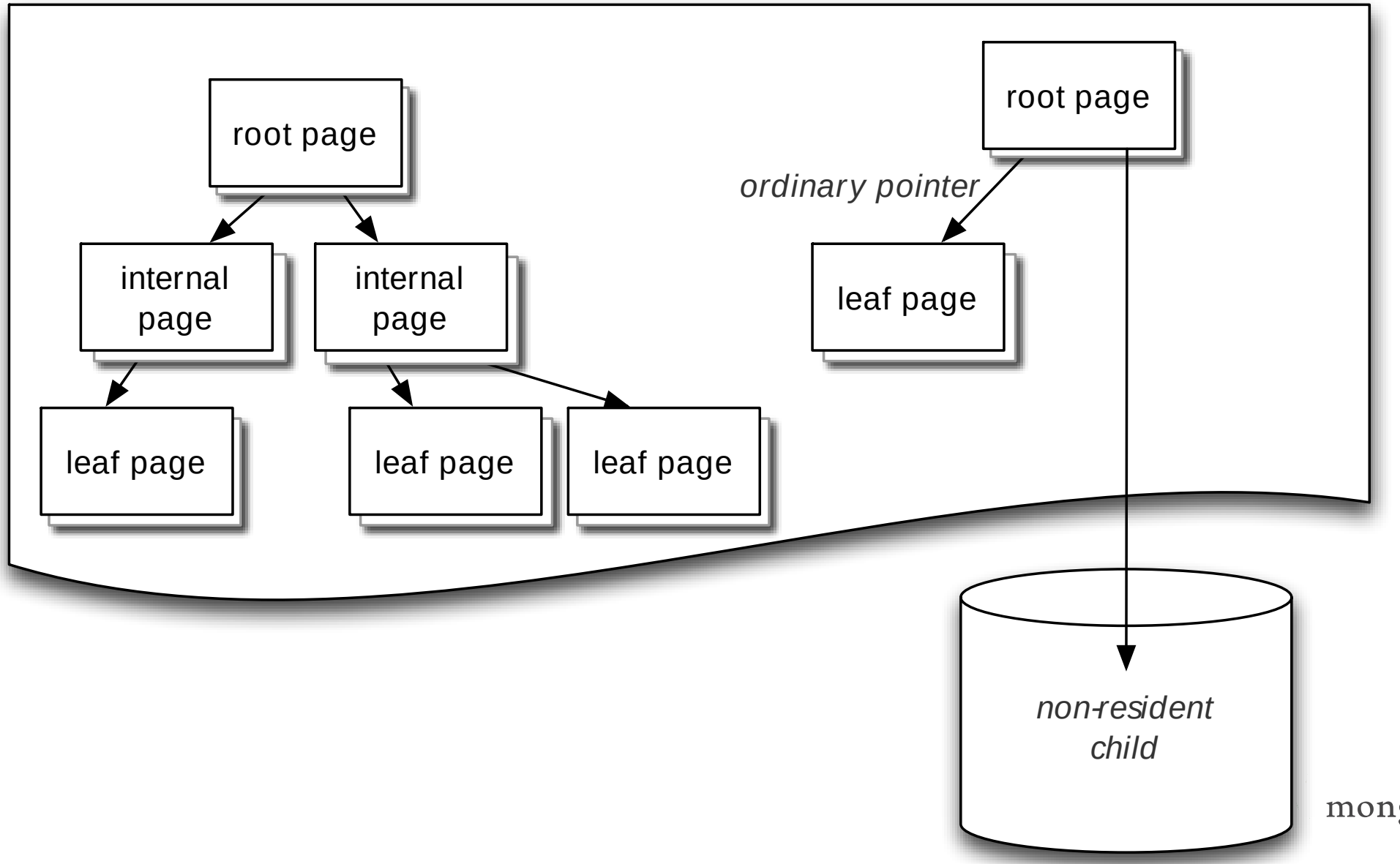
What You Will Learn

- ✓ WiredTiger Architecture
- ➢ In-memory performance
- Document-level concurrency
- Compression and checksums
- Durability and the journal
- What’s next?

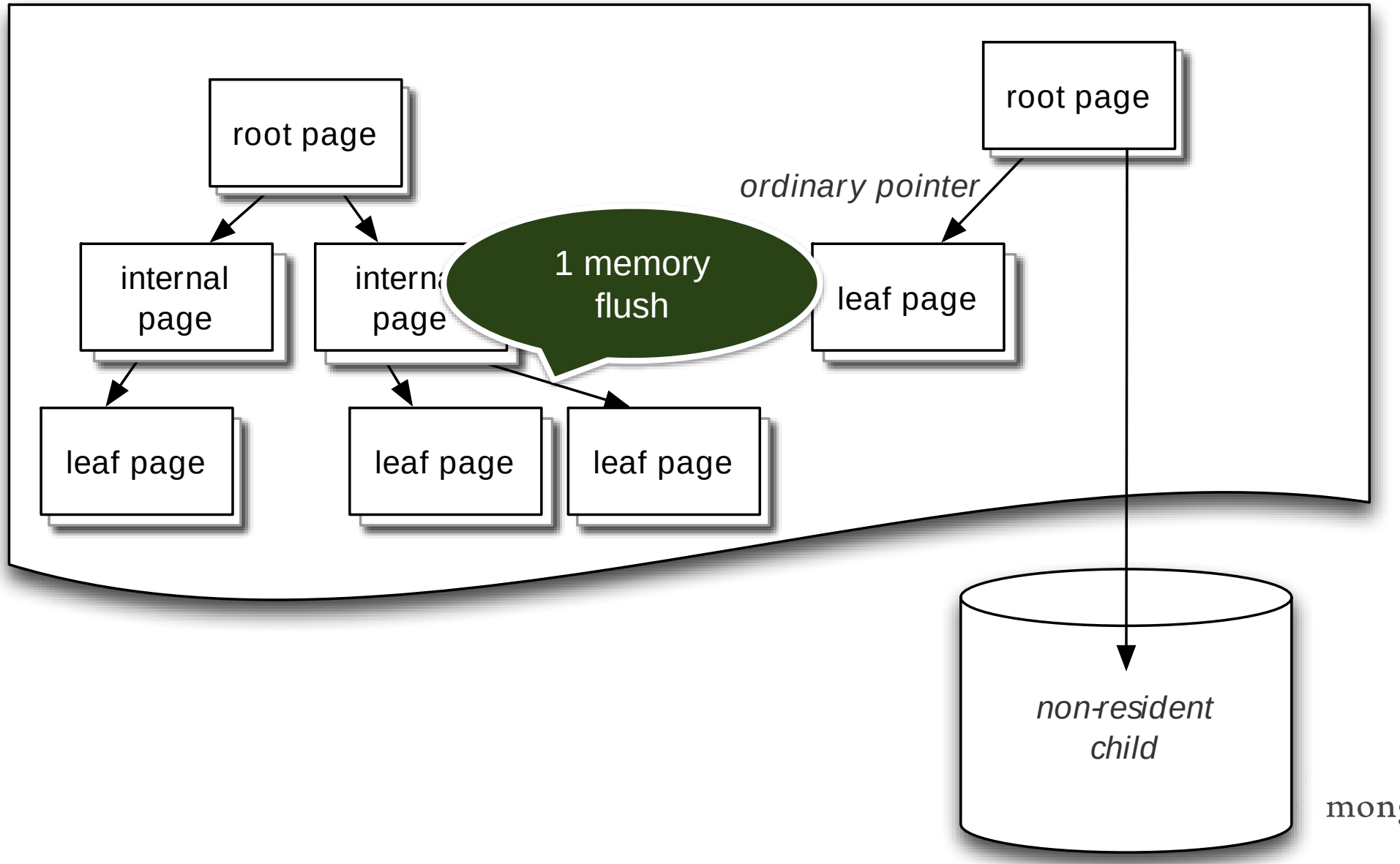
Traditional B+tree (ht wikipedia)



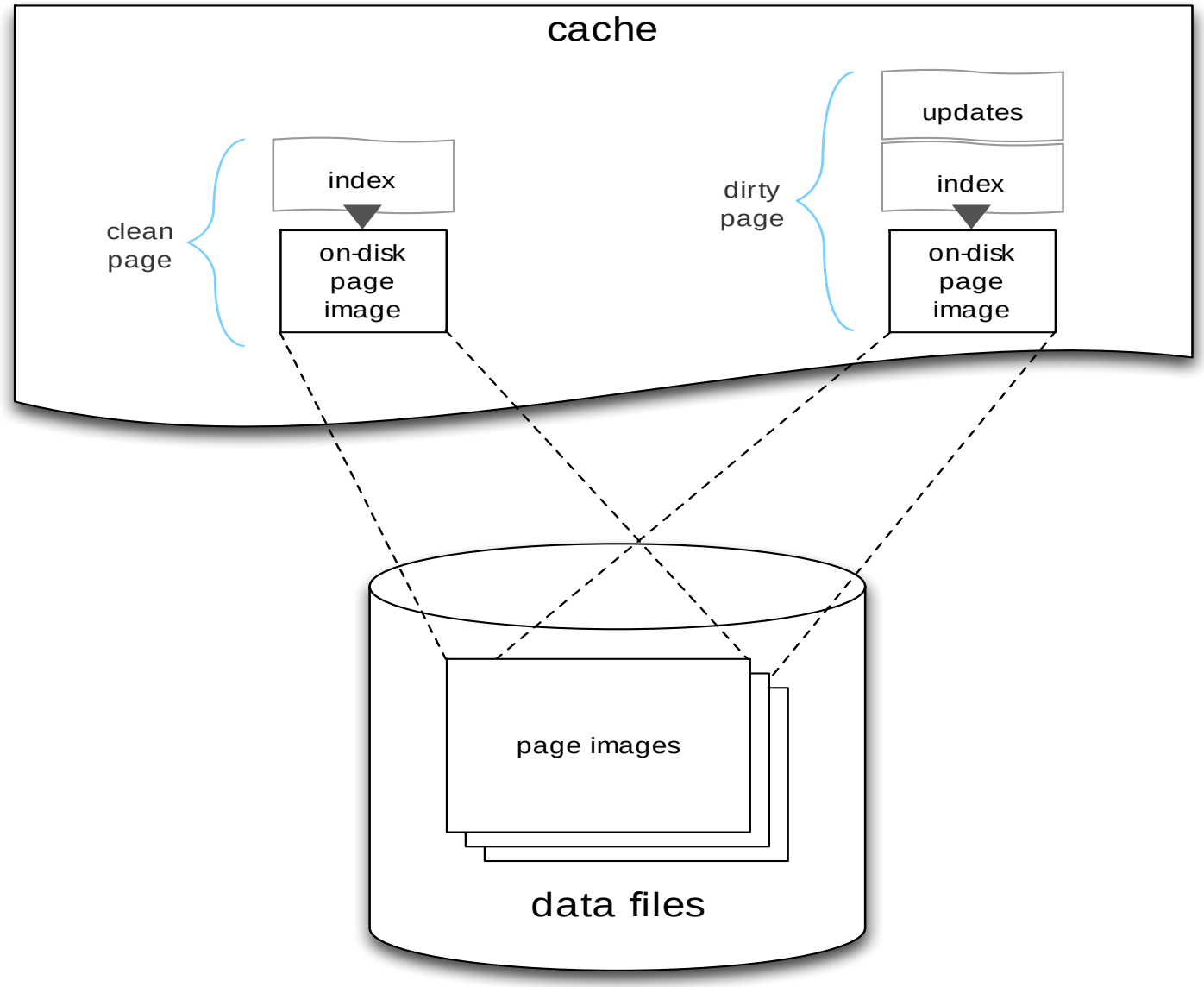
Trees in cache



Hazard Pointers



Pages in cache



In-memory performance

- Trees in cache are optimized for in-memory access
- Follow pointers to traverse a tree
  - no locking to access pages in cache
- Keep updates separate from clean data
- Do structural changes (eviction, splits) in background threads

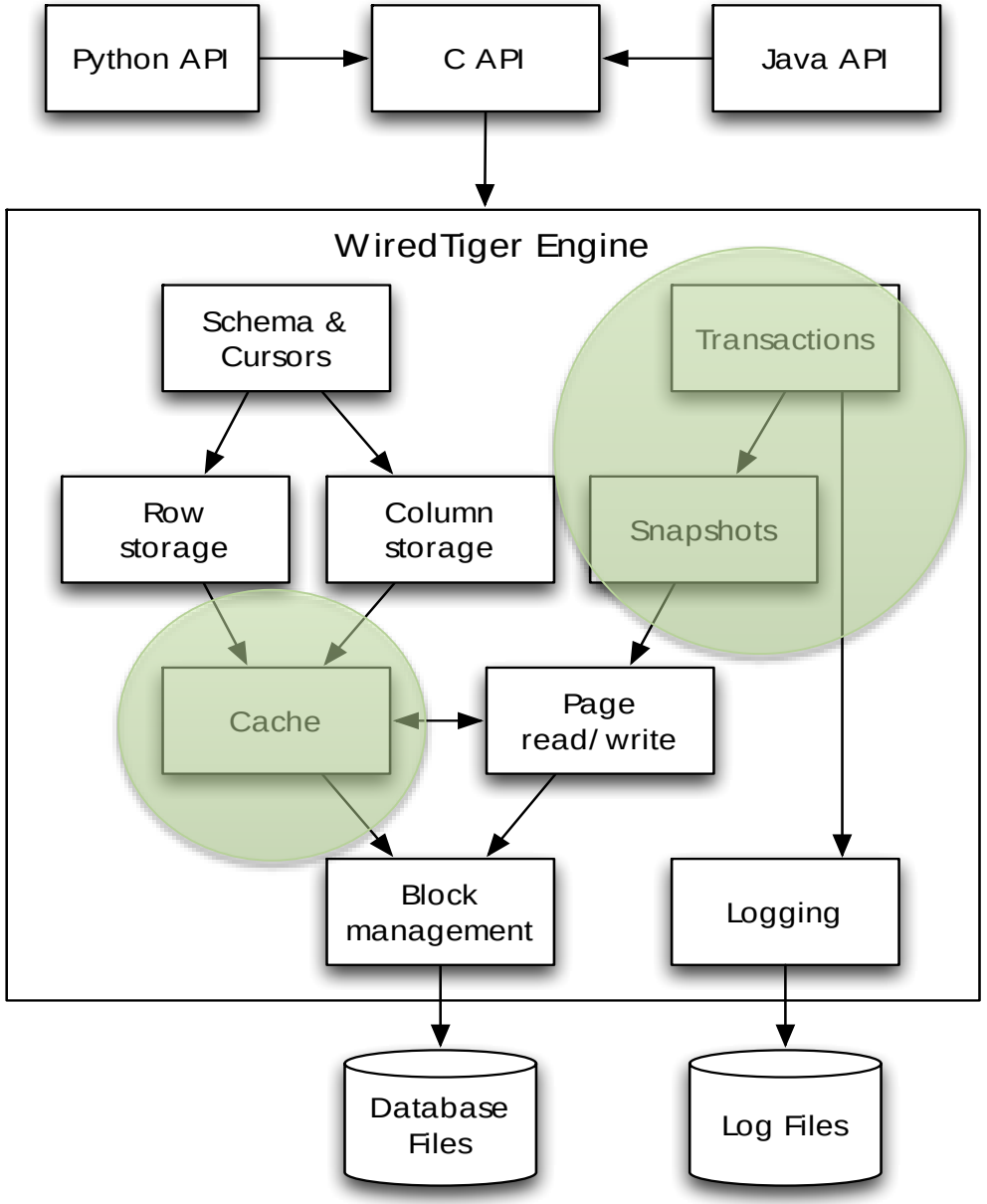
What You Will Learn

- ✓ WiredTiger Architecture
- ✓ In-memory performance
- ➢ Document-level concurrency
- • Compression and checksums
- • Durability and the journal
- • What’s next?

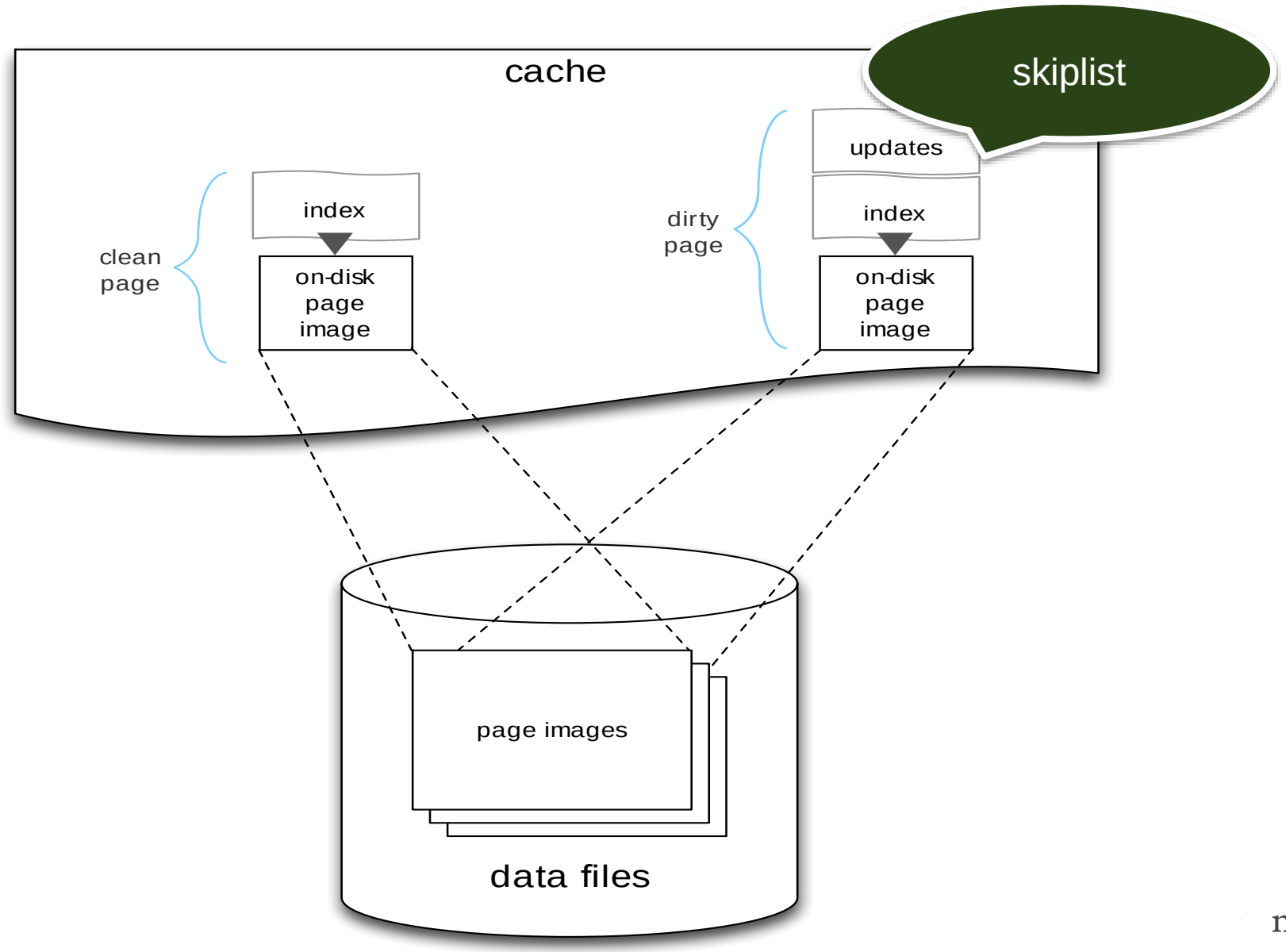
What is Concurrency Control?

- Computers have
  - multiple CPU cores
  - multiple I/O paths
- To make the most of the hardware, software has to execute multiple operations in parallel
- Concurrency control has to keep data consistent
- Common approaches:
  - locking
  - keeping multiple versions of data (MVCC)

WiredTiger Concurrency Control



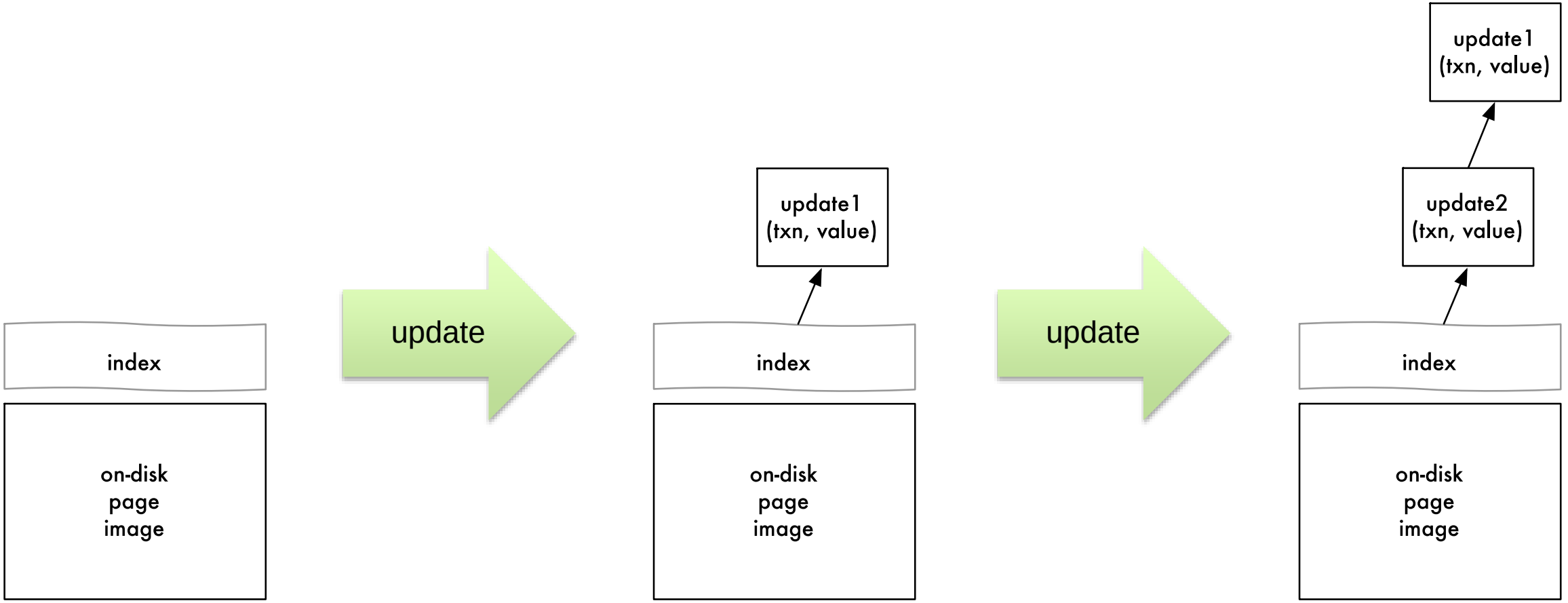
Pages in cache



Multiversion Concurrency Control (MVCC)

- Multiple versions of records kept in cache
- Readers see the committed version before the transaction started
  - MongoDB “yields” turn large operations into small transactions
- Writers can create new versions concurrent with readers
- Concurrent updates to a single record cause write conflicts
  - MongoDB retries with back-off

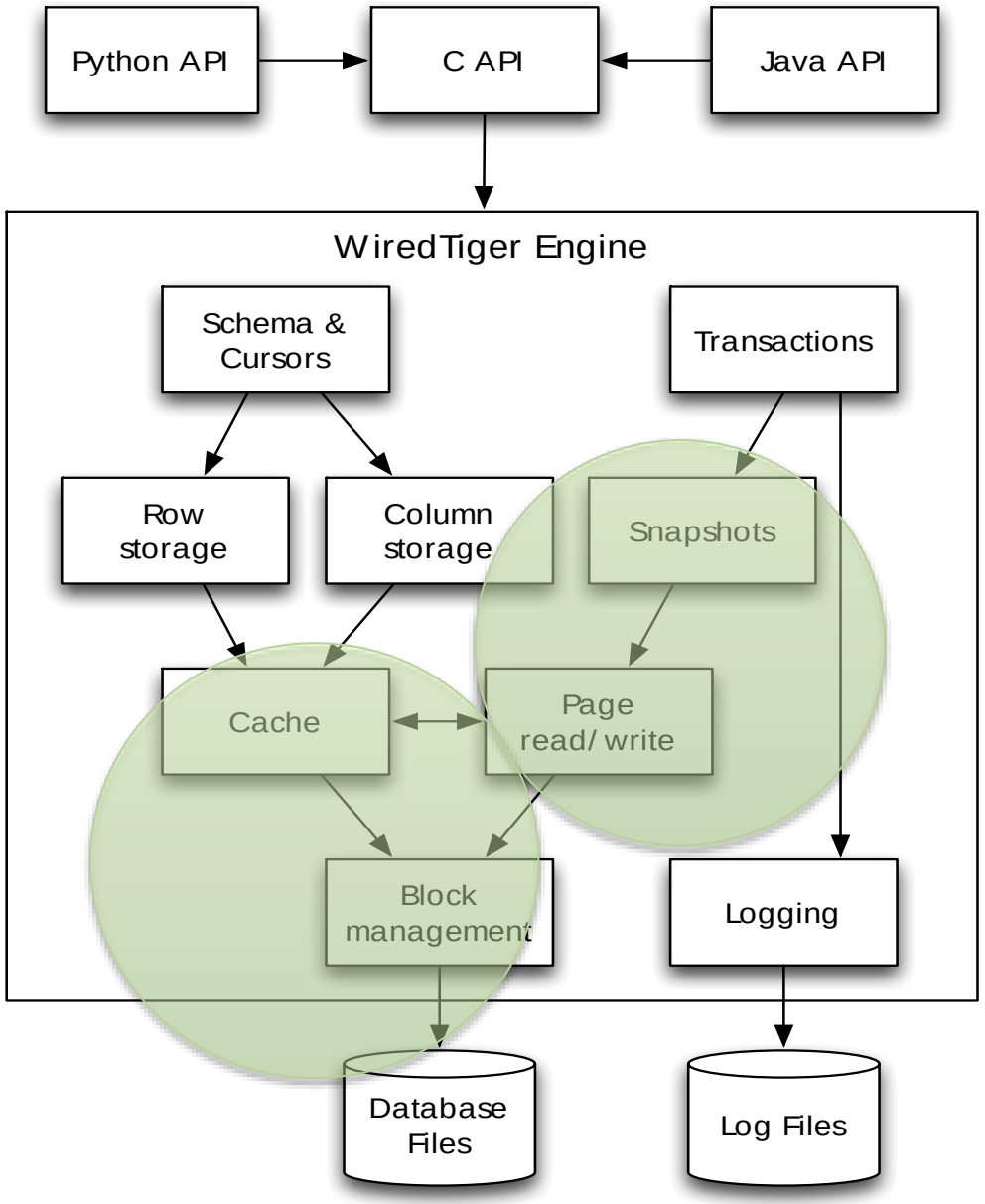
MVCC In Action



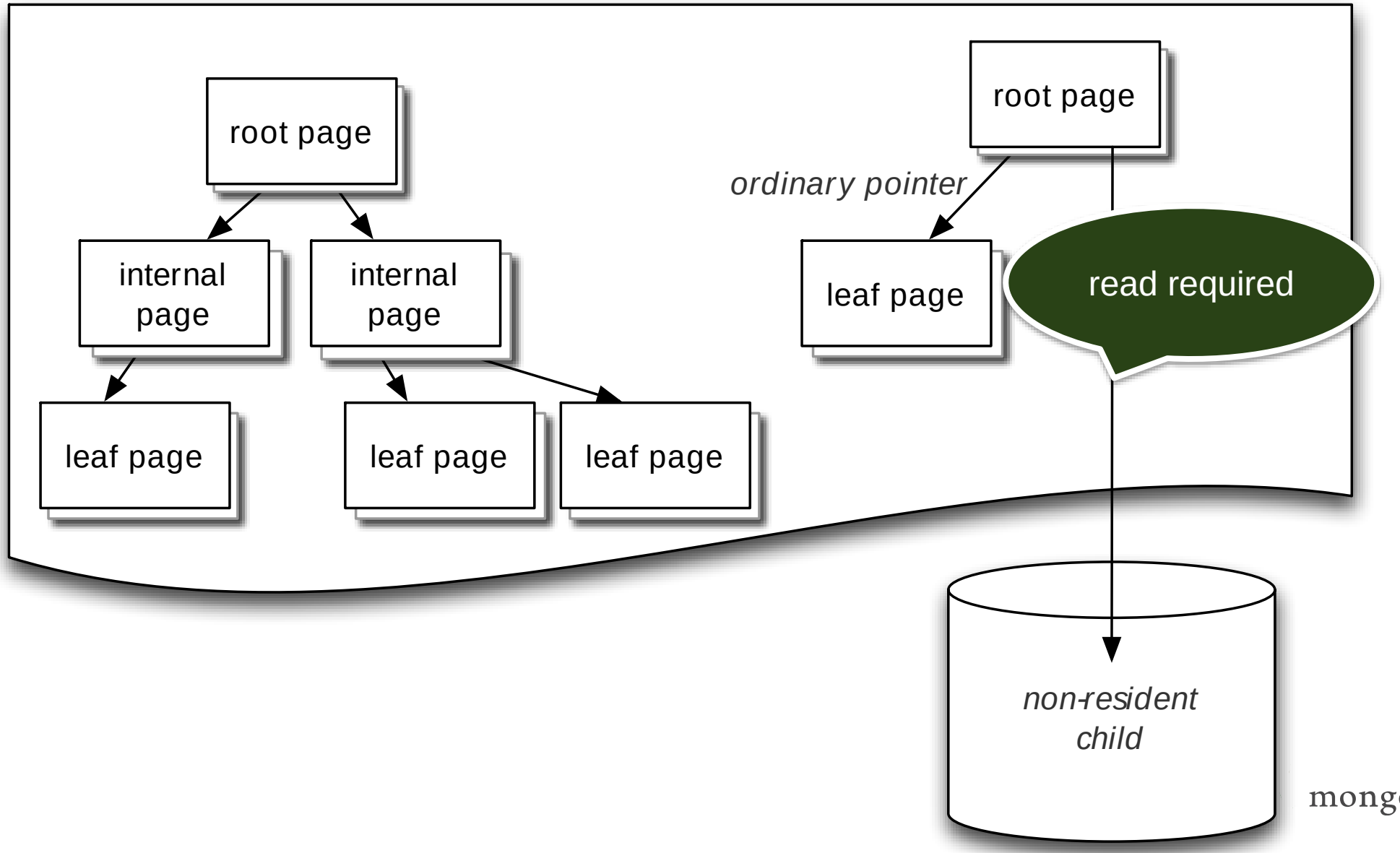
What You Will Learn

- ✓ WiredTiger Architecture
- ✓ In-memory performance
- ✓ Document-level concurrency
- ➢ Compression and checksums
- • Durability and the journal
- • What’s next?

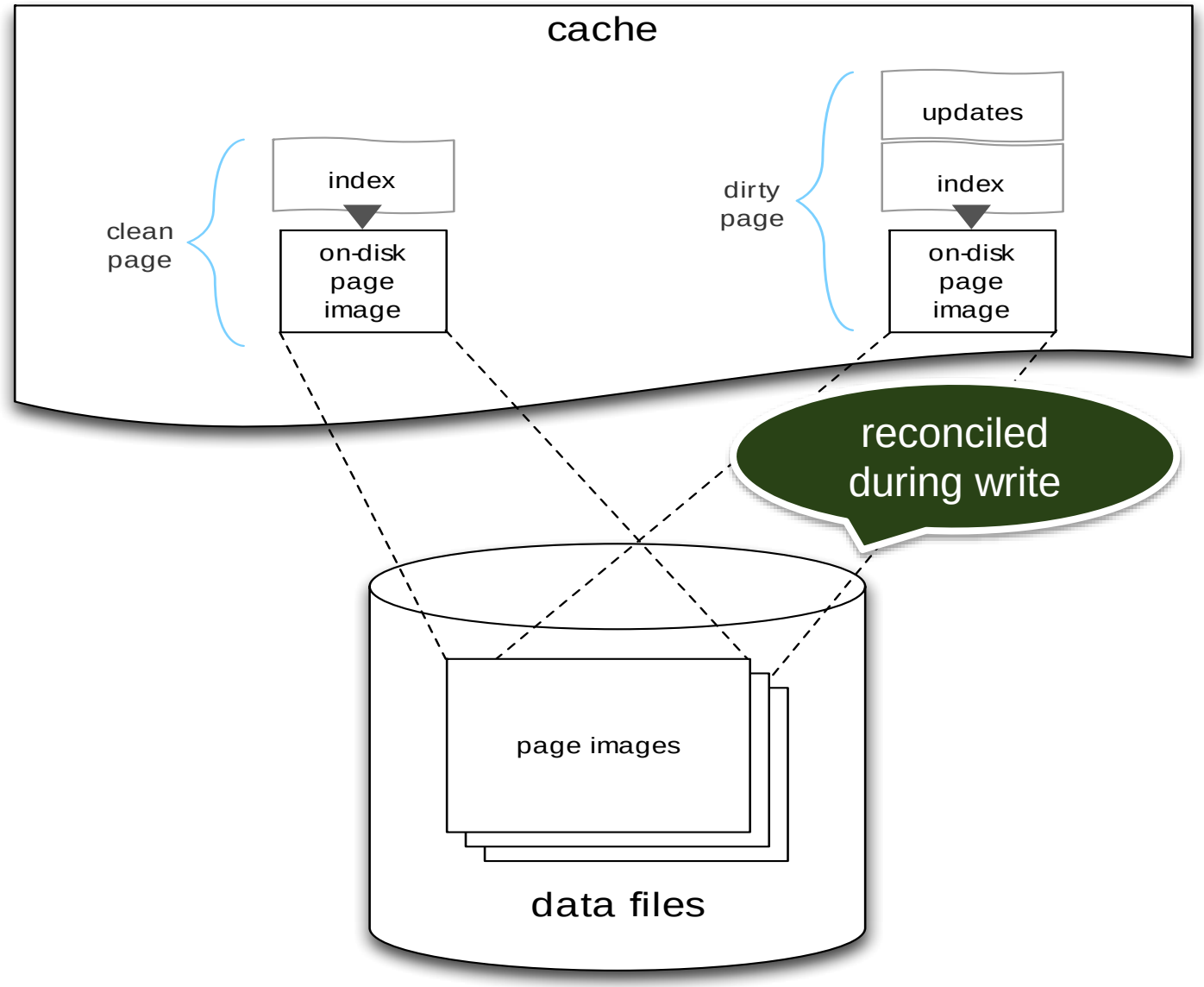
WiredTiger Page I/O



Trees in cache

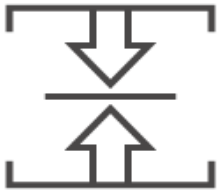


Pages in cache



Checksums

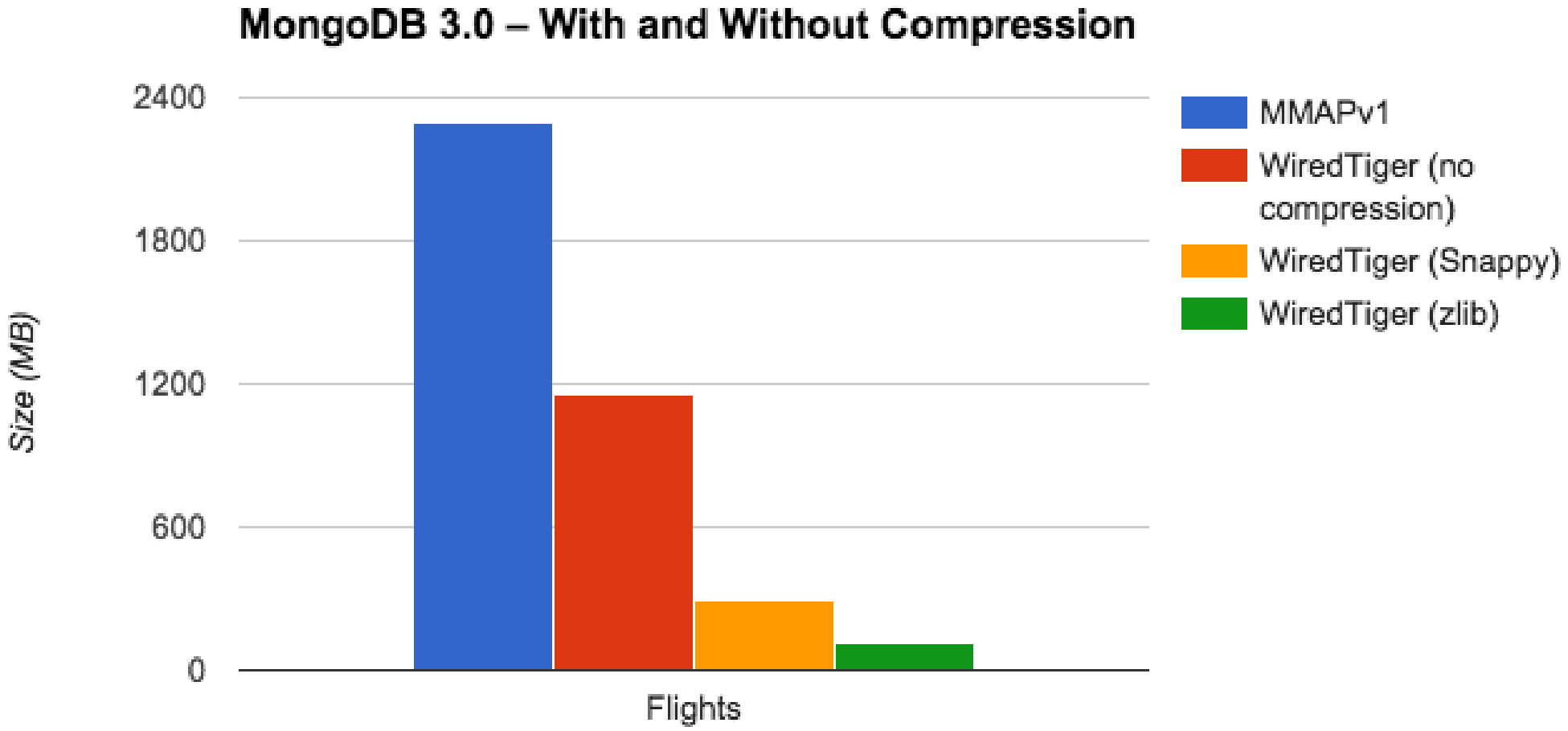
- A checksum is stored with every page
- Checksums are validated during page read
  - detects filesystem corruption, random bitflips
- WiredTiger stores the checksum with the page address (typically in a parent page)
  - extra safety against reading a stale page image



Compression

- WiredTiger uses snappy compression by default in MongoDB
- Supported compression algorithms:
  - snappy [default]: good compression, low overhead
  - zlib: better compression, more CPU
  - none
- Indexes also use prefix compression
  - stays compressed in memory

Compression in Action



(Flights database, ht Asya)

What You Will Learn

- ✓ WiredTiger Architecture
- ✓ In-memory performance
- ✓ Document-level concurrency
- ✓ Compression and checksums
- ➢ Durability and the journal
- What's next?

Journal and Recovery

- Write-ahead logging (aka journal) enabled by default
- Only written at transaction commit
- Log records are compressed with snappy by default
- Group commit for concurrency
- Automatic log archive / removal
- On startup, we rely on finding a consistent checkpoint in the metadata
- Use the metadata to figure out how much to roll forward

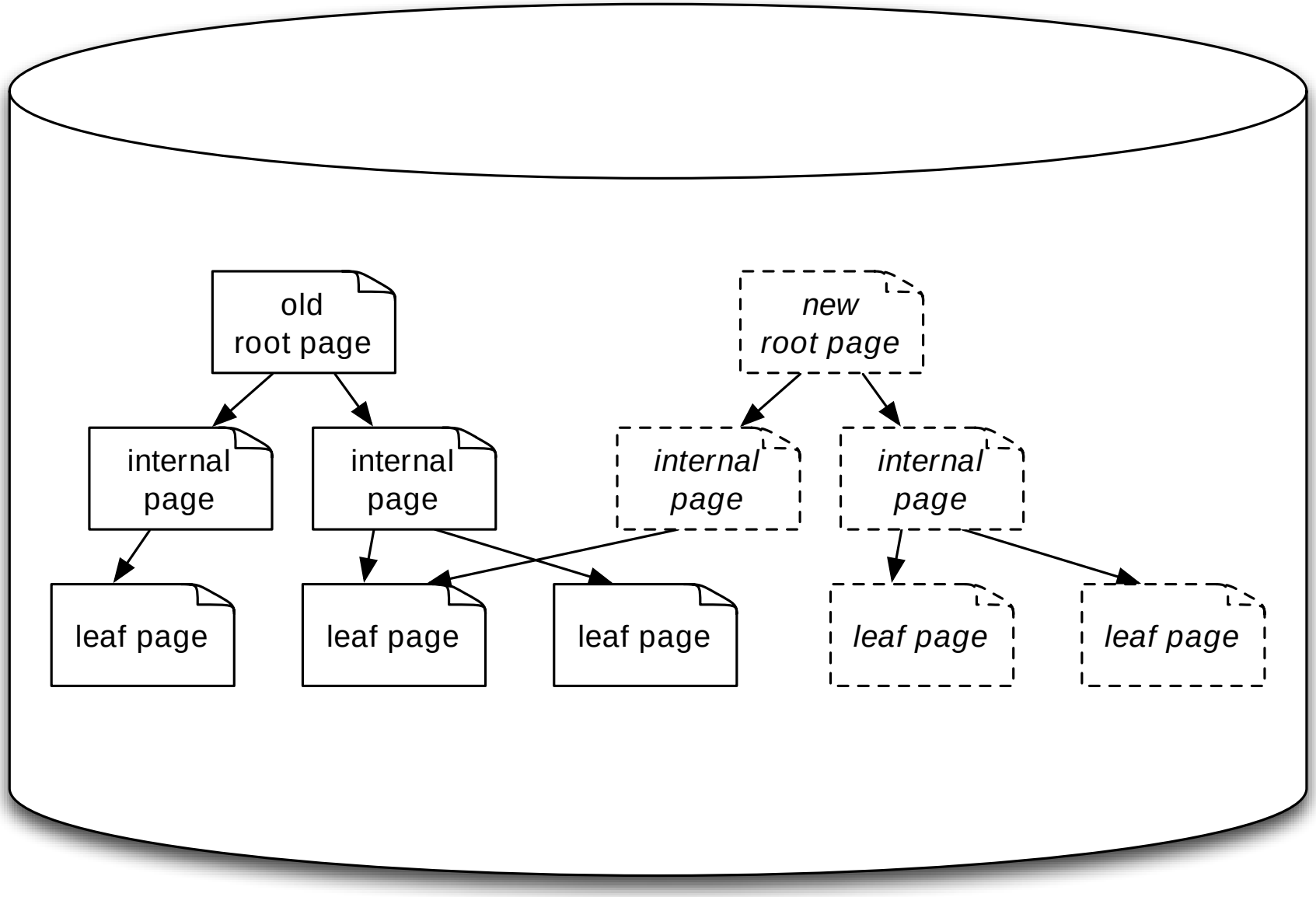
Durability without Journaling

- MMAPv1 require the journal for consistency
  - running with “nojournal” is unsafe
- WiredTiger doesn't have this need: no in-place updates
  - checkpoints every 60 seconds by default
  - with “nojournal”, updates since the last checkpoint may be lost
  - data will still be consistent
- Replication can guarantee durability

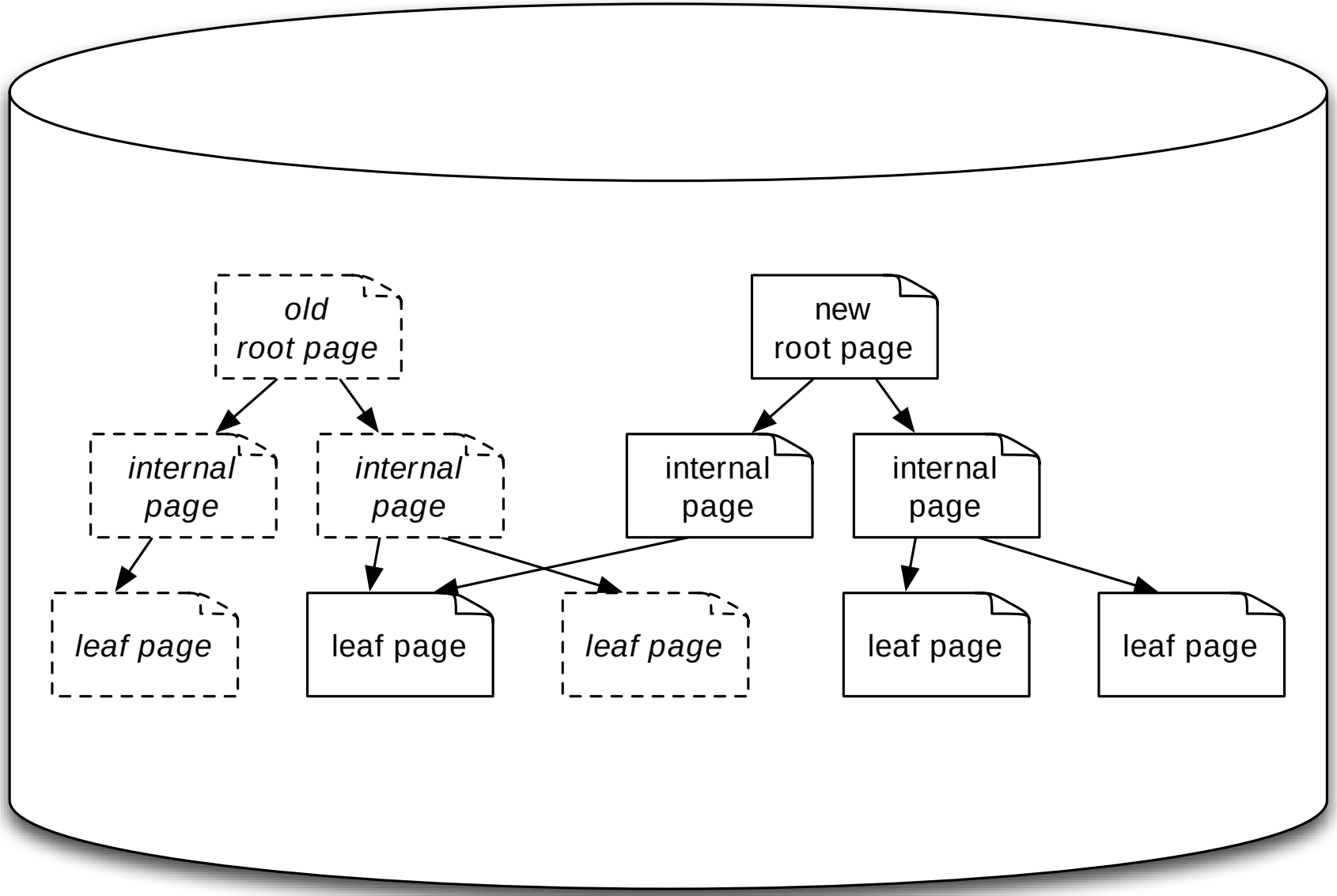
Writing a checkpoint

1. Write dirty leaf pages
   – don’t overwrite, write new versions into free space
2. Write the internal pages, including the root
   – the old checkpoint is still valid
3. Sync the file
4. Write the new root’s address to the metadata
   – free pages from old checkpoints once the metadata is durable

Checkpoints in Action



Checkpoints in Action (cont.)



What You Will Learn

- ✓ WiredTiger Architecture
- ✓ In-memory performance
- ✓ Document-level concurrency
- ✓ Compression and checksums
- ✓ Durability and the journal
- ➢ What’s next?

Adam Midvidy
@amidvidy

Follow

WiredTiger just became the default storage engine in the @MongoDB source tree. Congrats @WiredTigerInc @m_j_cahill github.com/mongodb/mongo/…

GitHub

SERVER-17861 Change the default storage engine to wiredTiger. ·...

WiredTiger is used as the default storage engine if the dbpath does not contain any data files. Otherwise, the storage engine specified in the storage.bson metadata file is used when the --storageEngi

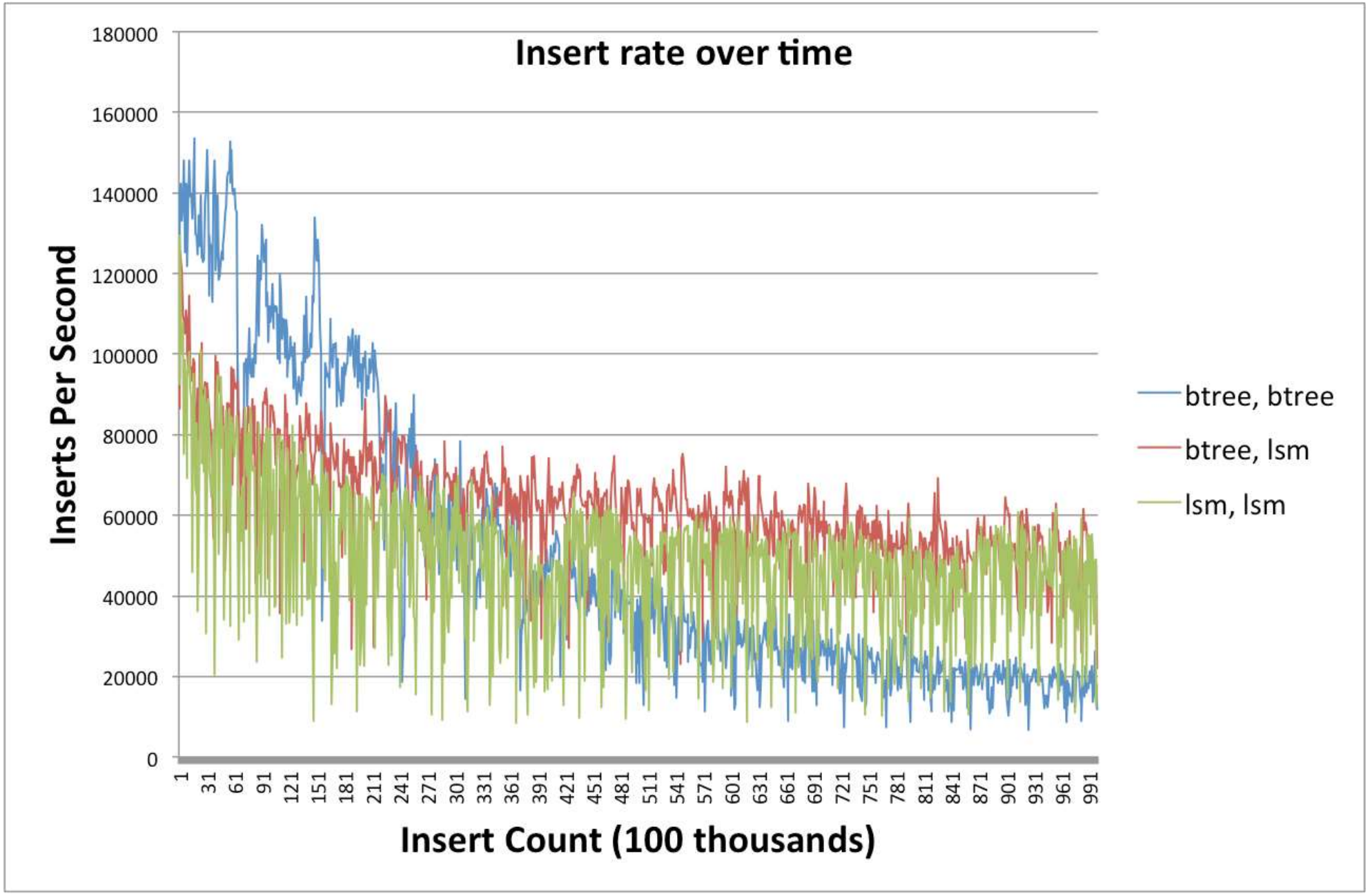
View on web

RETWEETS	FAVORITES
16	10

7:04 PM - 21 May 2015

WiredTiger LSM support

- Out-of-order insert workloads
- Data set much larger than cache
- Query performance is not (as) important
- Resource overhead of background maintenance acceptable


Insert rate over time
Inserts Per Second
180000
160000
140000
120000
100000
80000
60000
40000
20000
0
1
31
61
91
121
151
181
211
241
271
301
331
361
391
421
451
481
511
541
571
601
631
661
691
721
751
781
811
841
871
901
931
961
991
Insert Count (100 thousands)
btree, btree
btree, lsm
lsm, lsm

What's next for WiredTiger?

- Tune for (many) more workloads
  - avoid stalls during checkpoints with 100GB+ caches
  - make capped collections (including oplog) more efficient
  - Mark Callaghan seeing ~2x speedup in 3.1 snapshots: http://smalldatum.blogspot.com/
- Make Log Structured Merge (LSM) trees work well with MongoDB
  - out-of-cache, write-heavy workloads
- Adding encryption
- More advanced transactional semantics in the storage engine API

mongoDB

Thanks!

Questions?

Michael Cahill
michael.cahill@mongodb.com