mongoDB

mongoDB

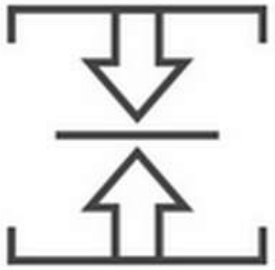
A Technical Introduction to WiredTiger

Michael Cahill
Director of Engineering (Storage), MongoDB

You may have seen this:

MongoDB 3.0 Now Available

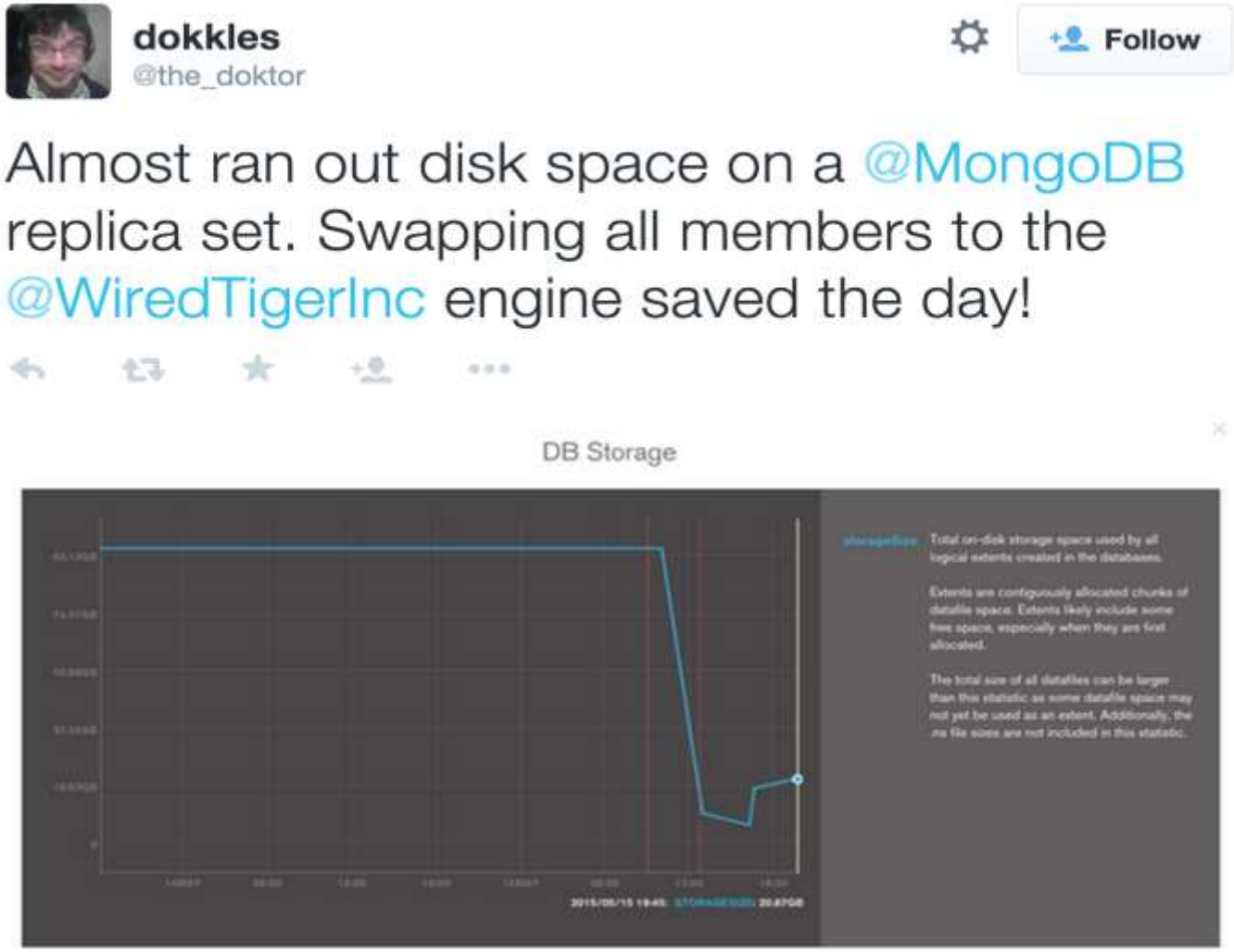
7x-10x Better Performance



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

How does WiredTiger do it?

What's different about WiredTiger?

- Document-level concurrency
- In-memory performance
- Multi-core scalability
- Checksums
- Compression
- Durability with and without journaling

This presentation is not…

- How to write stand-alone WiredTiger apps
- How to configure MongoDB with WiredTiger for your workload

WiredTiger Background

Why create a new storage engine?

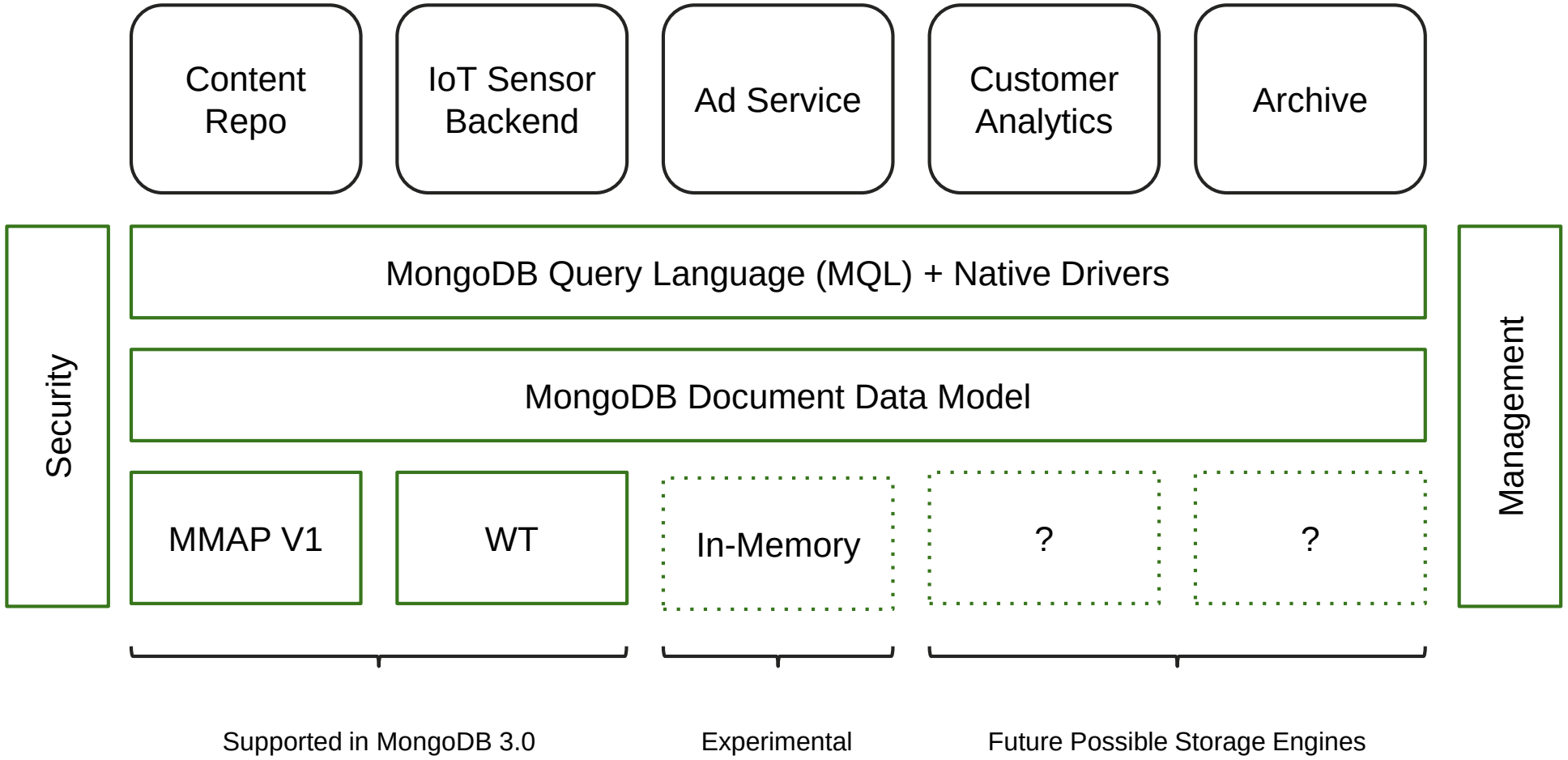
- Minimize contention between threads
  - lock-free algorithms, hazard pointers
  - eliminate blocking due to concurrency control

- Hotter cache and more work per I/O
  - compact file formats
  - compression
  - big-block I/O

MongoDB's Storage Engine API

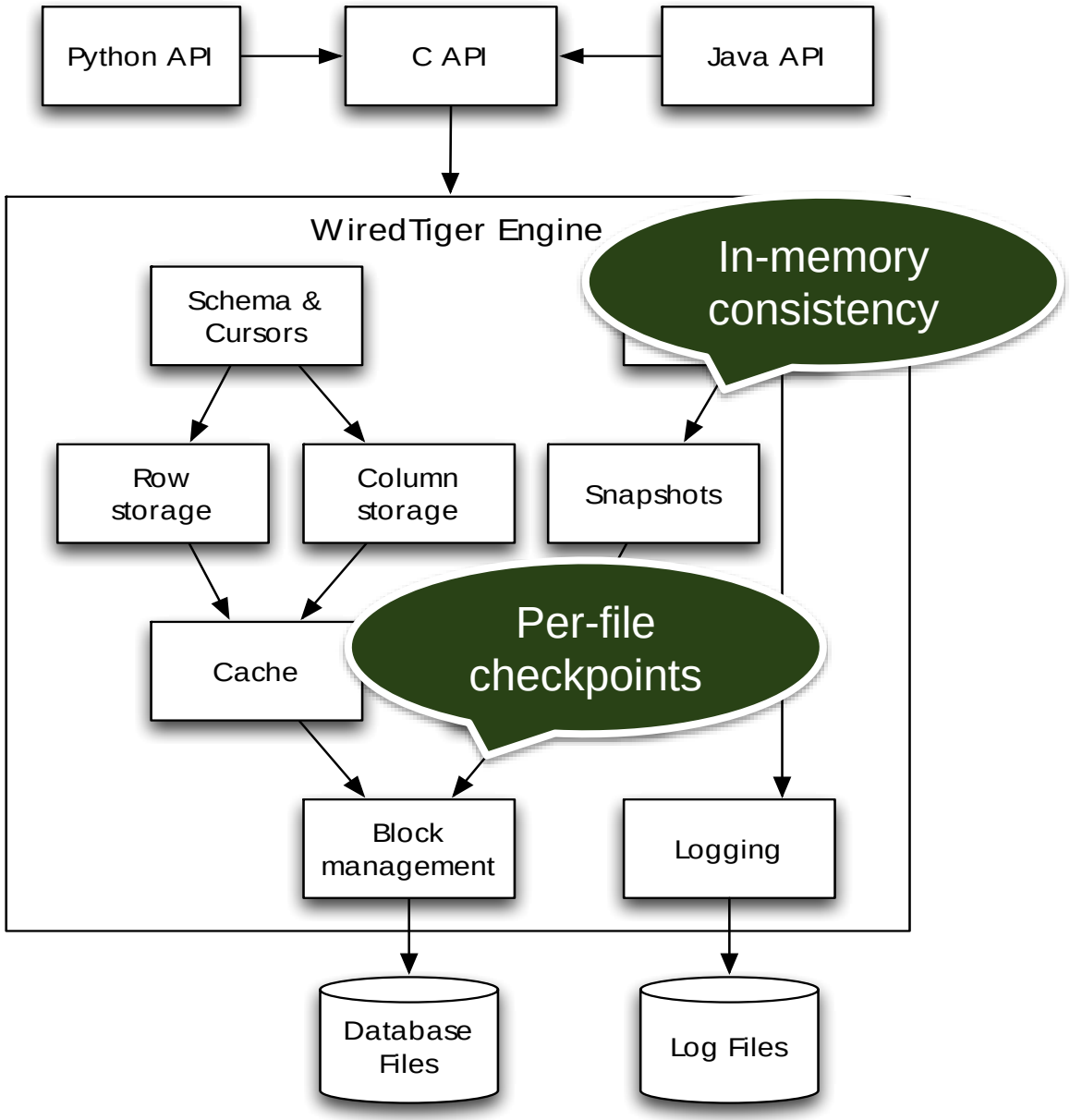
- Allows different storage engines to "plug-in"
  - Different workloads have different performance characteristics
  - mmap is not ideal for all workloads
  - More flexibility
    - mix storage engines on same replica set/sharded cluster
- Opportunity to integrate further (HDFS, native encrypted, hardware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

Storage Engine Layer

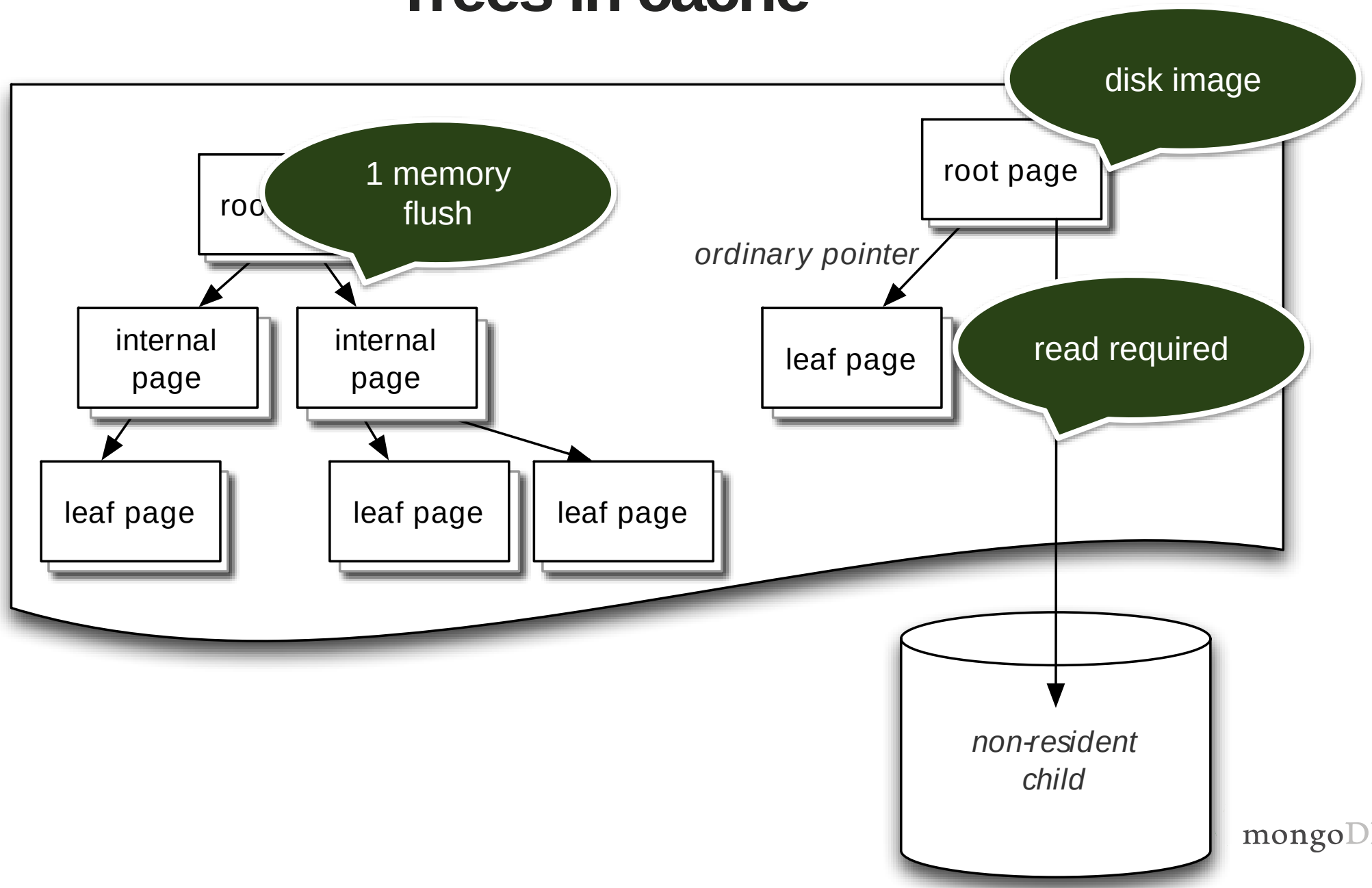


WiredTiger Architecture

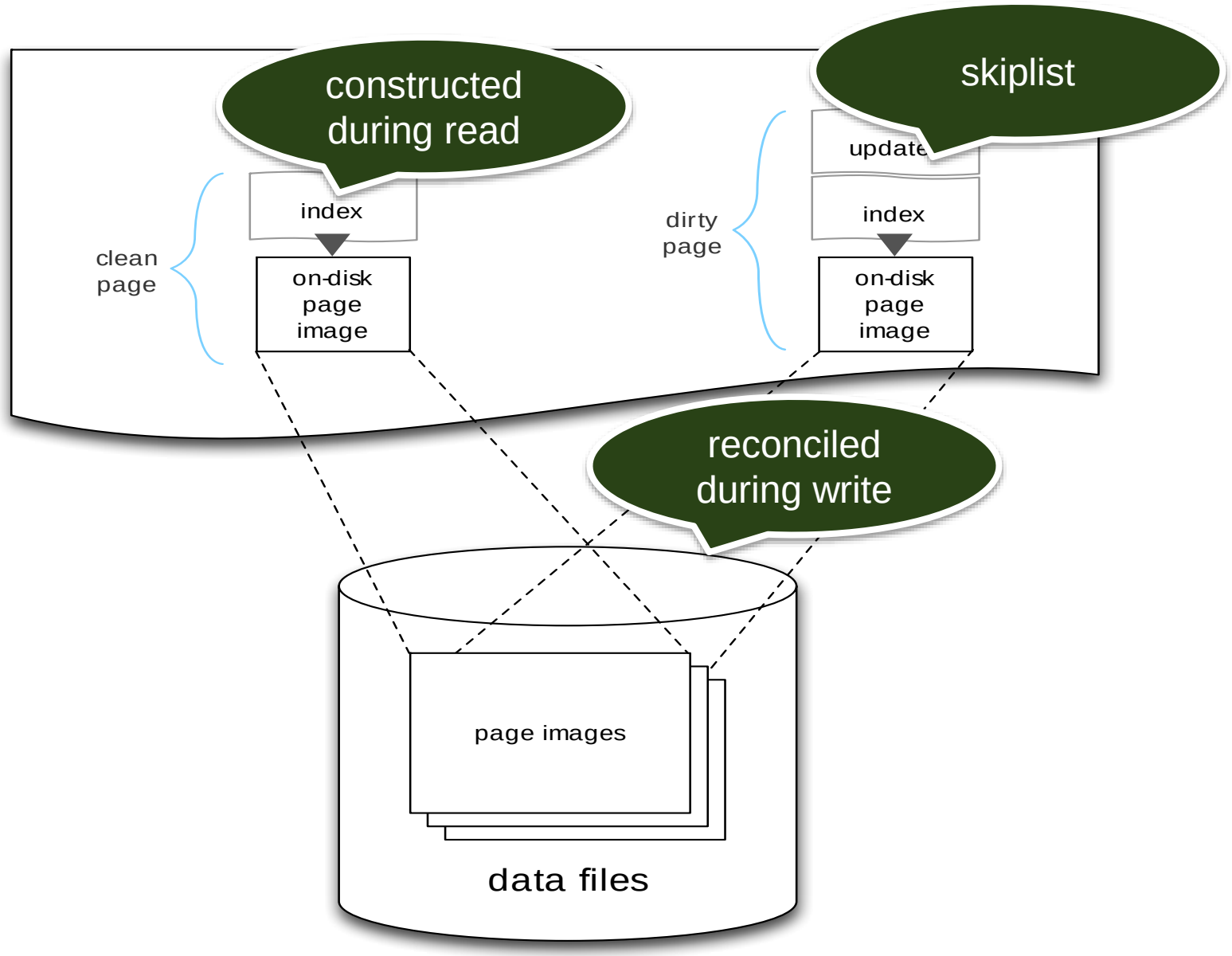
WiredTiger Architecture



Trees in cache



Pages in cache



Concurrency
and
multi-core scaling

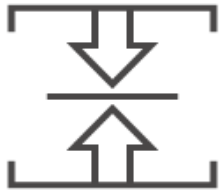
Multiversion Concurrency Control (MVCC)

- Multiple versions of records kept in cache
- Readers see the version that was committed before the operation 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

Transforming data during I/O

Checksums

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



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 are compressed using prefix compression
  - allows compression in memory

Durability with and without Journal

Writing a checkpoint

1. Write the leaves
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

Durability without Journaling

- MMAPv1 uses a write-ahead log (journal) to guarantee consistency
  - Running with “nojournal” is unsafe
- WiredTiger doesn't have this need: no in-place updates
  - Write-ahead log can be truncated at checkpoints
    - Every 2GB or 60sec by default – configurable
  - Updates are written to optional journal as they commit
    - Not flushed on every commit by default
    - Recovery rolls forward from last checkpoint
- Replication can guarantee durability

Journal and recovery

- Optional write-ahead logging
- Only written at transaction commit
- snappy compression by default
- Group commit
- Automatic log archive / removal
- On startup, we rely on finding a consistent checkpoint in the metadata
- Check LSNs in the metadata to figure out where to roll forward from

So, what's different about WiredTiger?

- Multiversion Concurrent Control
  - No lock manager
- Non-locking data structures
  - Multi-core scalability
- Different representation of in-memory data vs on-disk
  - Enabled checksums, compression
- Copy-on-write storage
  - Durability without a journal

What’s next?

What's next for WiredTiger?

- WiredTiger btree access method is optional in 3.0
- plan to make it the default in 3.2
- tuning for (many) more workloads
- make Log Structured Merge (LSM) 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