Mongoing 中文社区

mongoing 中文社区 | IT大咖说 知识分享平台

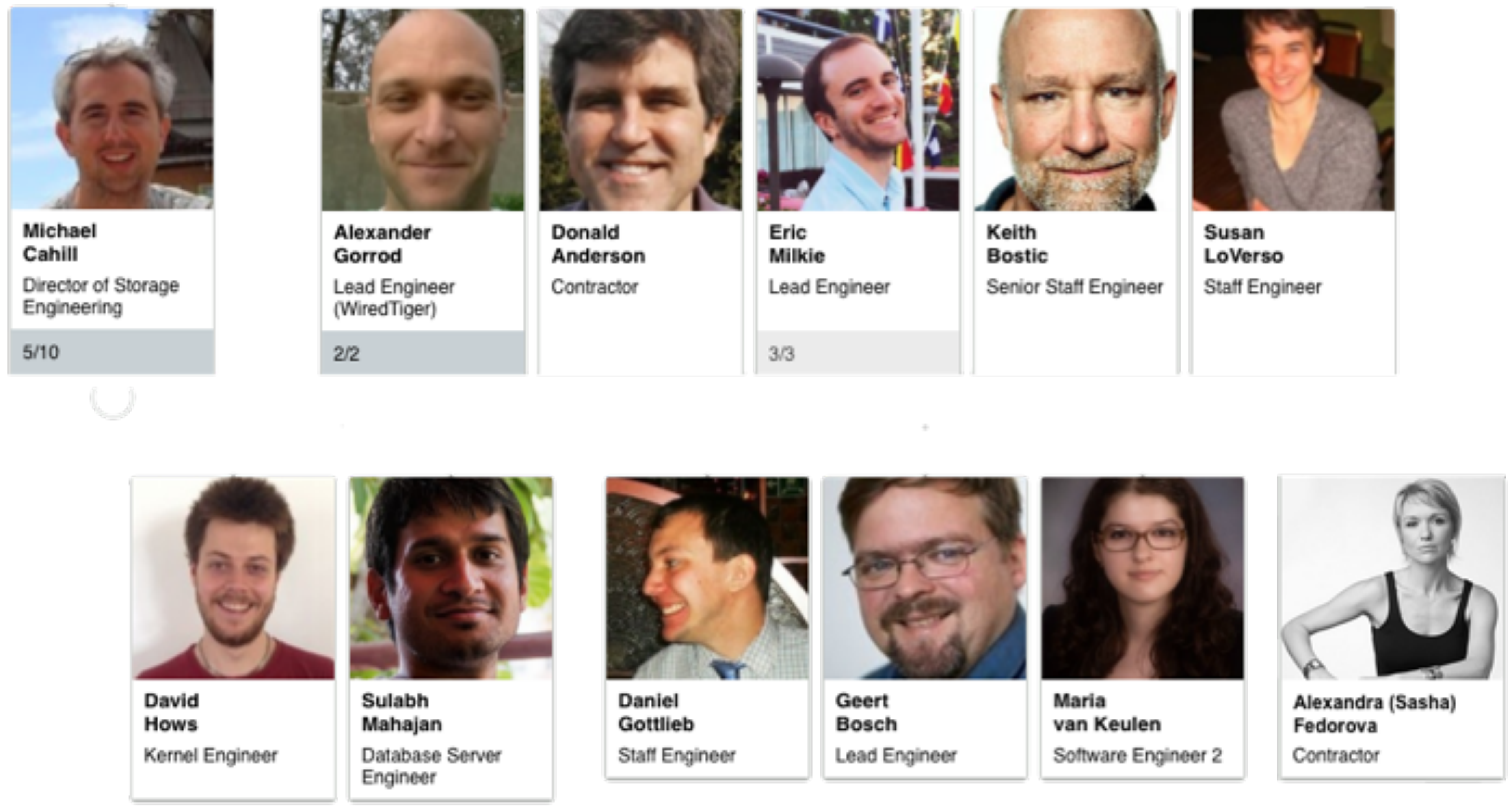
MongoDB 年终盛会 2017

Mongoing中文社区　　海量数据学院

MONGODB STORAGE ROADMAP: 3.6 AND BEYOND

Michael Cahill
Director of Engineering (Storage)

WHO ARE WE?



Michael Cahill
Director of Storage Engineering
5/10

Alexander Gorrod
Lead Engineer (WiredTiger)
2/2

Donald Anderson
Contractor

Eric Milkie
Lead Engineer
3/3

Keith Bostic
Senior Staff Engineer

Susan LoVerso
Staff Engineer

David Hows
Kernel Engineer

Sulabh Mahajan
Database Server Engineer

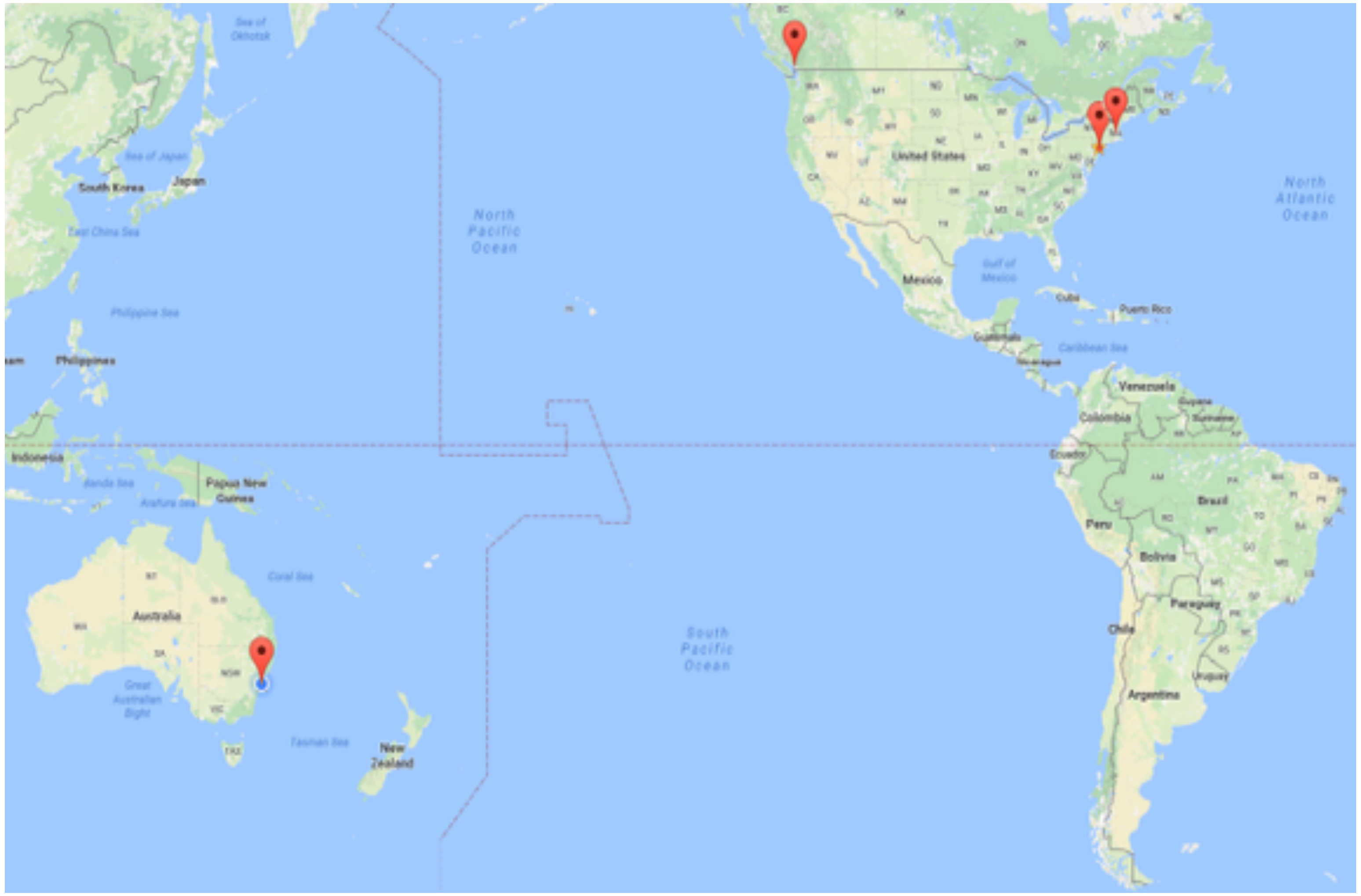
Daniel Gottlieb
Staff Engineer

Geert Bosch
Lead Engineer

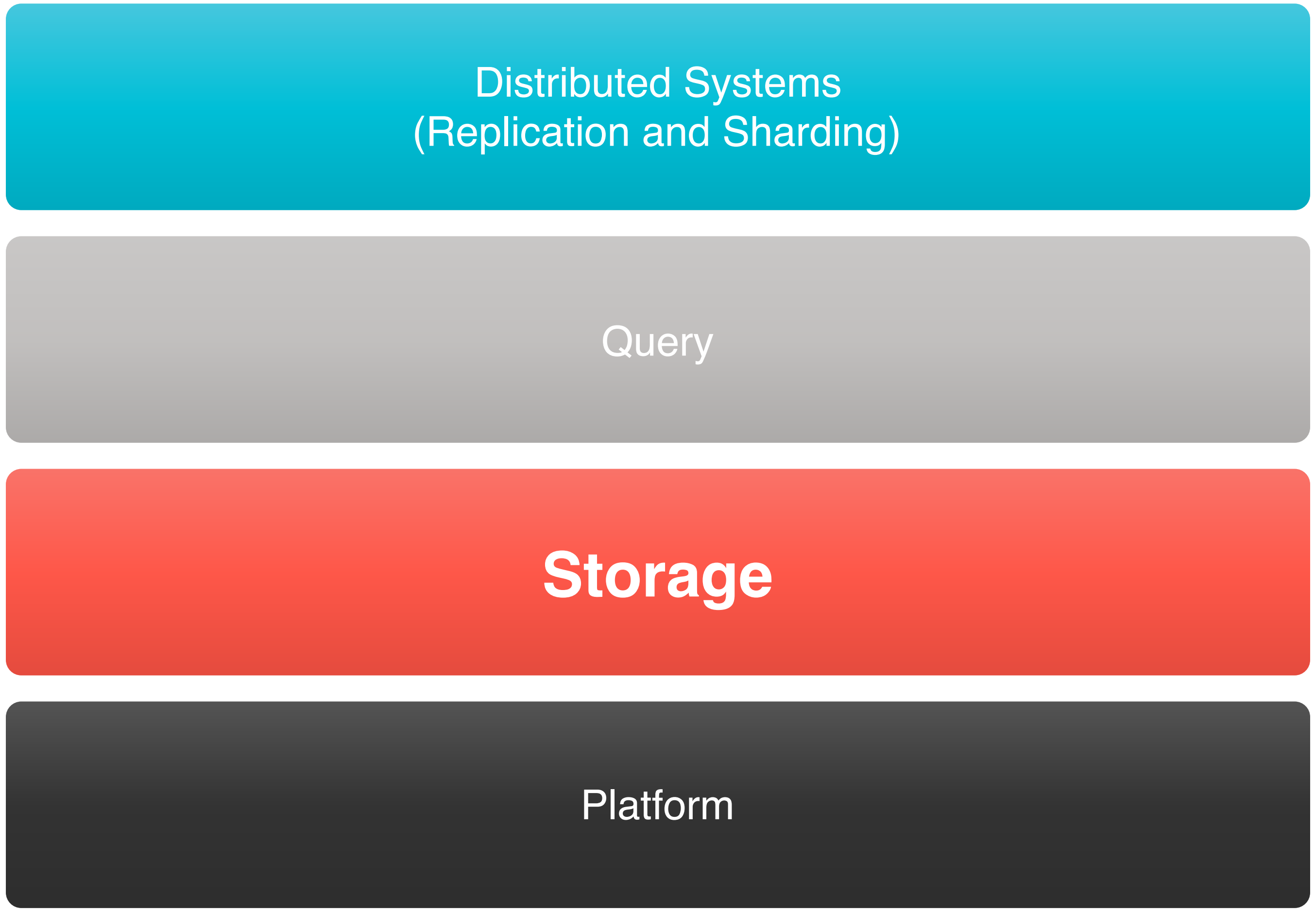
Maria van Keulen
Software Engineer 2

Alexandra (Sasha) Fedorova
Contractor

WHERE ARE WE?


North Pacific Ocean
South Pacific Ocean
North Atlantic Ocean
United States
Mexico
Australia
New Zealand
Japan
South Korea
Philippines
Indonesia
Papua New Guinea
Brazil
Peru
Bolivia
Chile
Argentina
Venezuela
Colombia
Cuba
Puerto Rico

WHERE DO WE FIT?



WHAT DO WE DO?

- ➔ All MongoDB storage engines (MMAPv1, WiredTiger, inMemory, encrypted)
- ➔ Storage Engine API
- ➔ Concurrency control
- ➔ Durability and crash recovery
- ➔ Catalog and metadata (create, drop, rename)
- ➔ Index builds (e.g., foreground vs background

mongoing 中文社区 | IT大咖说 知识分享平台

WHY SHOULD YOU CARE?

- Storage layer keeps your data (crash) safe
- Performance of local operations depends on:
  - Locking / queuing
  - Reading from disk
  - Writing to disk

AGENDA

3.6	3.8+	3.8+	4.0+
Upgrade / downgrade	Deprecate MMAPv1	Transaction support	New storage engines

UPGRADE / DOWNGRADE

Since 3.0, no incompatible changes to files written by WiredTiger

What about?

➔ New compression support

➔ Store deltas when large docs change

MongoDB now has a stable upgrade/downgrade procedure

➔ PM-755 Upgrade/downgrade support in WiredTiger

WHY TRANSACTIONS?

- MongoDB was designed for a NoSQL world
  - One document at a time
  - Transactions across documents less of an application requirement
- MongoDB application domain growing
  - Supporting more traditional applications
  - Often, applications surrounding the existing MongoDB space
- Also, simplifying existing applications

TRANSACTIONS: ACID

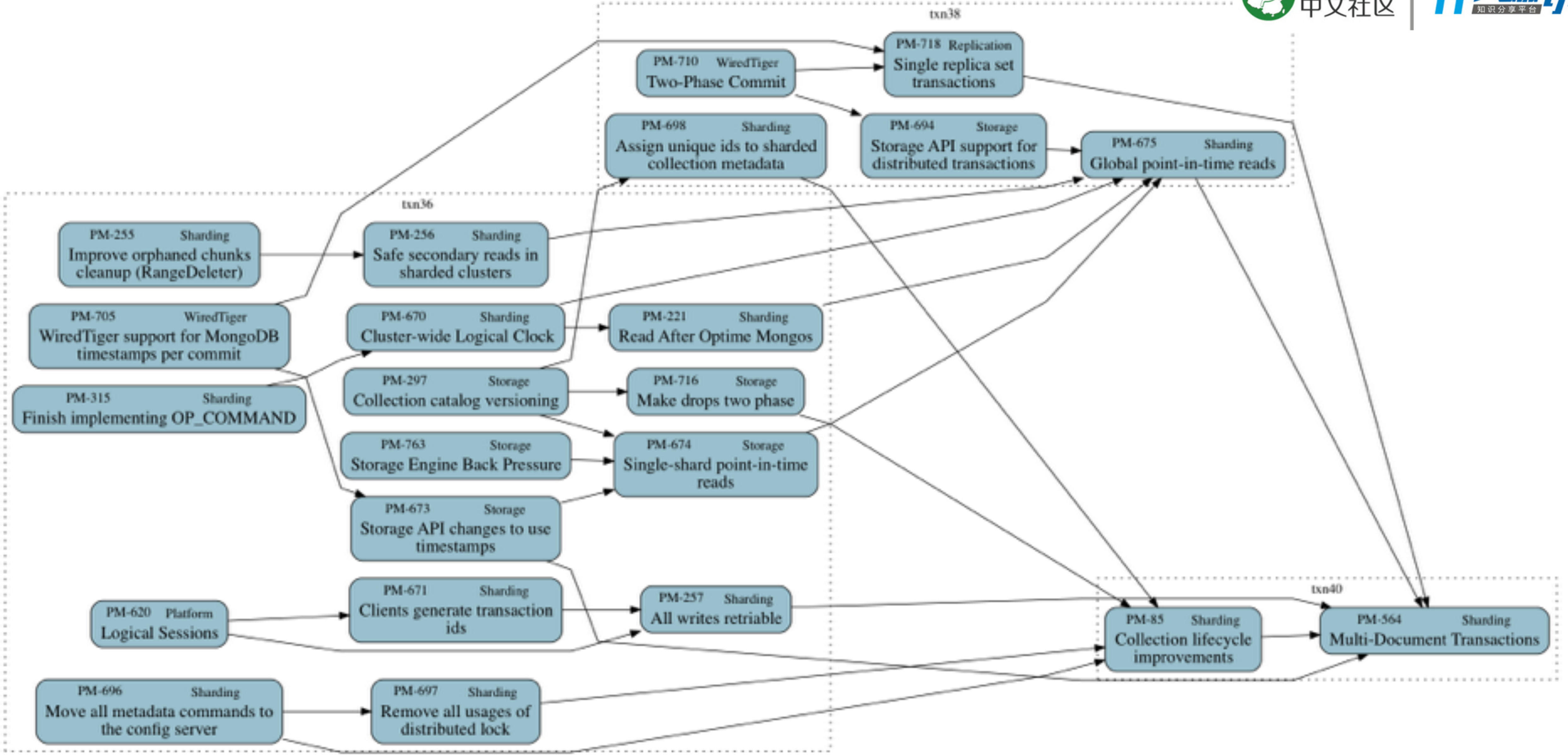
- Atomicity
  - All or nothing
- Consistency
  - Application constraints are not violated
- Isolation
  - Concurrent transactions do not interfere with each other
- Durability
  - Committed updates survive server restarts and network failure

MONGODB'S PRESENT

- ACID for single-document transactions
  - Atomically update multiple fields of a document (and indices)
  - Transaction cannot span multiple documents (or collections)
  - Durability provided by "`w: majority`" updates
- Single server consistency
  - Eventual consistency on the secondaries

TRANSACTION ROADMAP

3.6	3.8	4.0
Safe secondary reads Causal consistency All writes retryable	Single replica set transactions Global point-in-time reads	Multi-doc transactions


txn38
txn36
txn40
PM-710 WiredTiger Two-Phase Commit
PM-718 Replication Single replica set transactions
PM-698 Sharding Assign unique ids to sharded collection metadata
PM-694 Storage Storage API support for distributed transactions
PM-675 Sharding Global point-in-time reads
PM-255 Sharding Improve orphaned chunks cleanup (RangeDeleter)
PM-256 Sharding Safe secondary reads in sharded clusters
PM-705 WiredTiger WiredTiger support for MongoDB timestamps per commit
PM-670 Sharding Cluster-wide Logical Clock
PM-221 Sharding Read After Optime Mongos
PM-315 Sharding Finish implementing OP_COMMAND
PM-297 Storage Collection catalog versioning
PM-716 Storage Make drops two phase
PM-763 Storage Storage Engine Back Pressure
PM-674 Storage Single-shard point-in-time reads
PM-673 Storage Storage API changes to use timestamps
PM-620 Platform Logical Sessions
PM-671 Sharding Clients generate transaction ids
PM-257 Sharding All writes retriable
PM-85 Sharding Collection lifecycle improvements
PM-564 Sharding Multi-Document Transactions
PM-696 Sharding Move all metadata commands to the config server
PM-697 Sharding Remove all usages of distributed lock

STEP 1: DEPRECATE MMAPV1

MMAPv1 is tuned for some use cases that are slower in WiredTiger:

PM-720 Fast in-place updates to large documents

PM-771 Work better with lots of collections

PM-714 Store multiple collections per table

PM-493 / PM-707 Better repair for corrupted databases

TRANSACTION SUPPORT IN 3.6+

WiredTiger already has transactions, how hard can it be?

PM-297 Collection catalog versioning

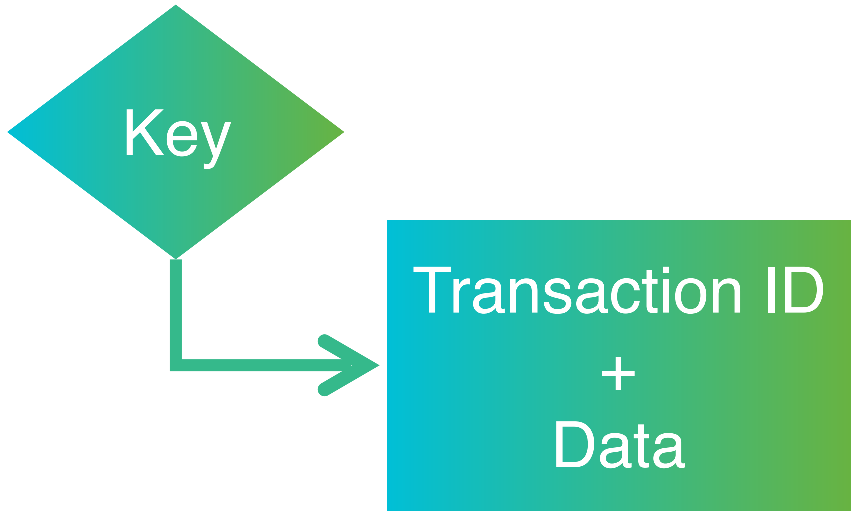
PM-716 Make drops two phase

PM-705 / PM-673 Timestamps in WiredTiger

PM-674 readConcern: majority available by default

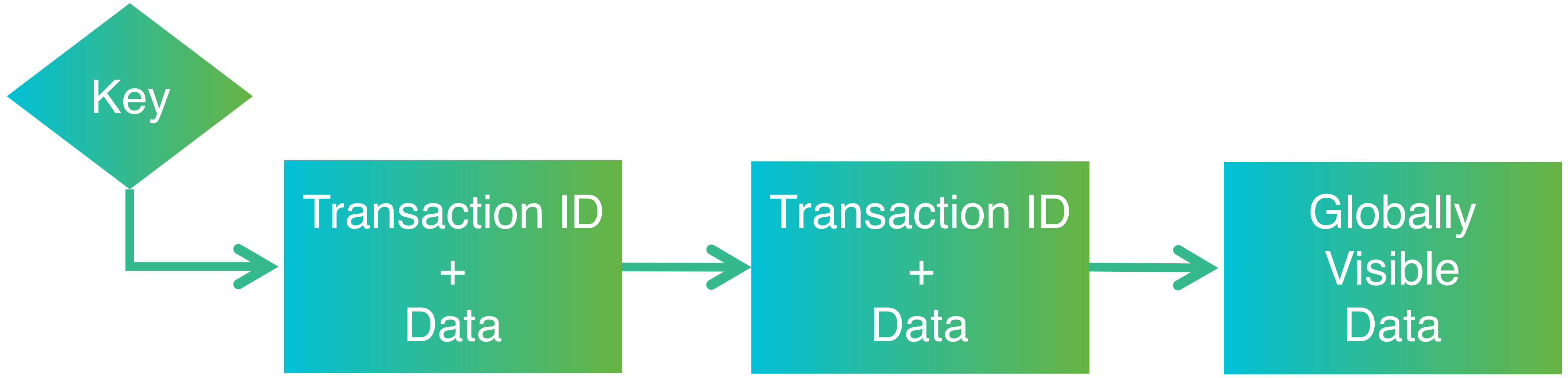
WIREDTIGER UPDATES

- Updates include
  - Transaction ID (is the update committed / visible?)
  - Data package



MULTI-VERSION CONCURRENCY CONTROL

- Each key references
  - Chain of updates in most recently modified order
  - Original value, the update visible to everybody



TIMESTAMP SUPPORT IN WIREDTIGER

- Applications have their own notion of transactions and time
  - Defines an expected commit order
  - Defines durability for a set of systems
- MongoDB now sends transaction timestamps to WiredTiger
  - 8B but expected to grow to encompass system-wide ordering
  - Mix-and-match with native WiredTiger transactions

MONGODB 3.6 READS "AS OF" TIMESTAMP

- Updates now include a commit timestamp
  - Timestamp tracked in WiredTiger's update
  - Smaller is better, as a significant overhead for small updates
- Commit "as of" a timestamp
  - Set during the update or later, at transaction commit
- Read "as of" a timestamp
  - Set at transaction begin
  - Point-in-time reads: largest timestamp less than or equal to value

MONGODB 3.8: STABLE TIMESTAMP

- Limits future replication rollbacks
  - Imagine an election where the primary hasn't seen a committed update
- WiredTiger writes checkpoints at the stable timestamp
  - The storage engine can't write what might be rolled back
- Cannot go backward, must be updated frequently

TRANSACTION SUPPORT LONGER TERM

PM-715 Recover to a timestamp

➔ avoid complex replication rollback logic

Transactional secondary apply of oplog

➔ secondaries apply operations without locking

➔ storage layer returns consistent results

TRANSACTION SUPPORT LONGER TERM

PM-494 Transactional create, drop and rename

PM-663 Hybrid index builds

➔ foreground build speed without locking

PM-710 2-phase commit

➔ detect reads of prepared updates

STORAGE PROJECTS 4.0+?

Write-optimized store (LSM)

Analytics / Column store

➔ Store and query with field granularity

➔ Fast for projections on (lots of) sparse documents

STORAGE PROJECTS 4.0+?

Mobile store

➔ Optimized, low-footprint storage for mobile devices

Cold store

➔ Use S3 (or similar) for cheap, slow, high-availability storage

MONGODB STORAGE ROADMAP

谢谢!

Michael Cahill
  Director of Engineering (Storage)