In-Memory Accelerator for MongoDB™

Yakov Zhdanov, Director R&D

GridGain Systems
www.gridgain.com

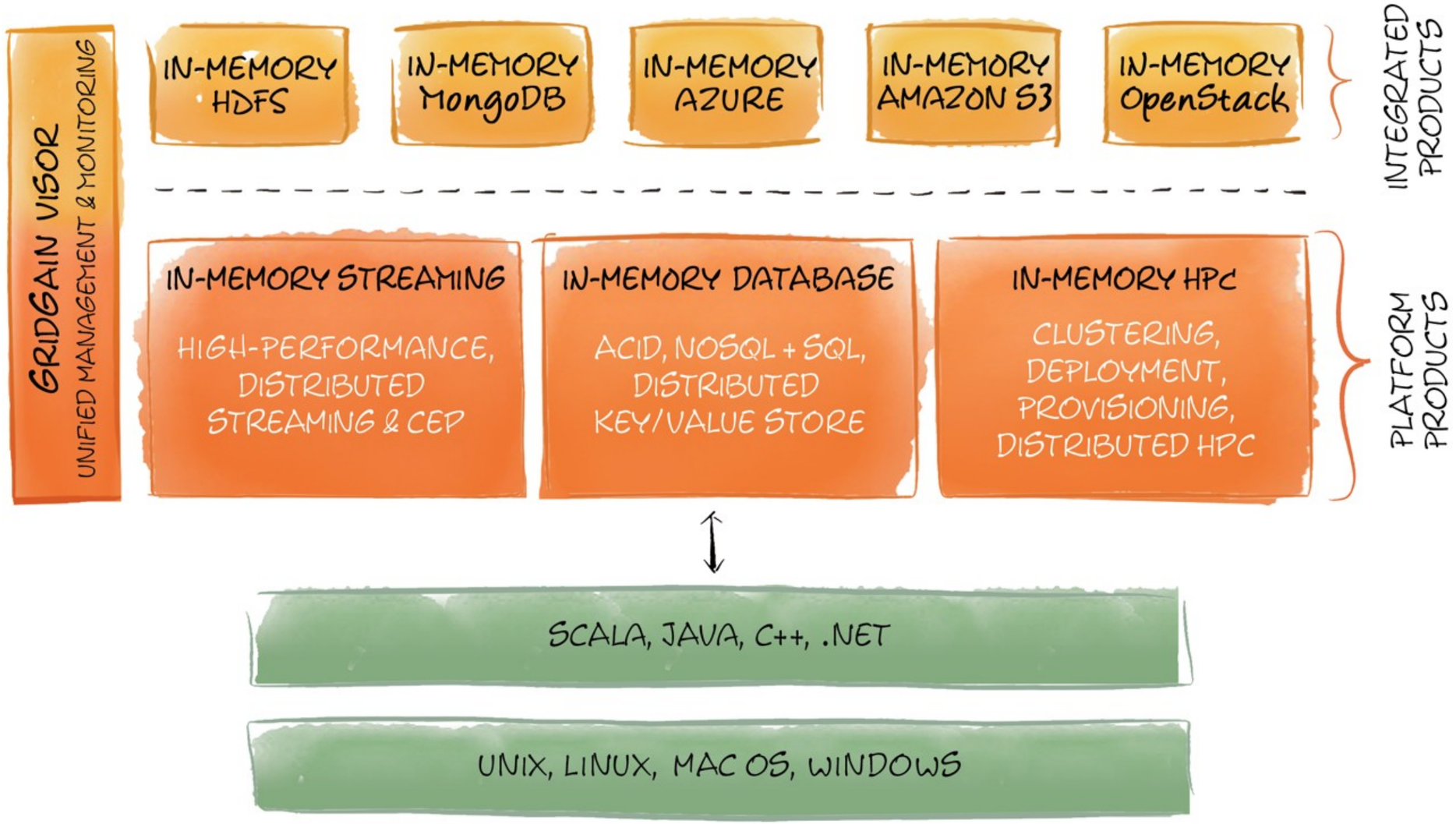
GridGain: In-Memory Computing Leader

- 5 years in production
- 100s of customers & users
- Starts every 10 secs worldwide
- Over 15,000,000 starts globally

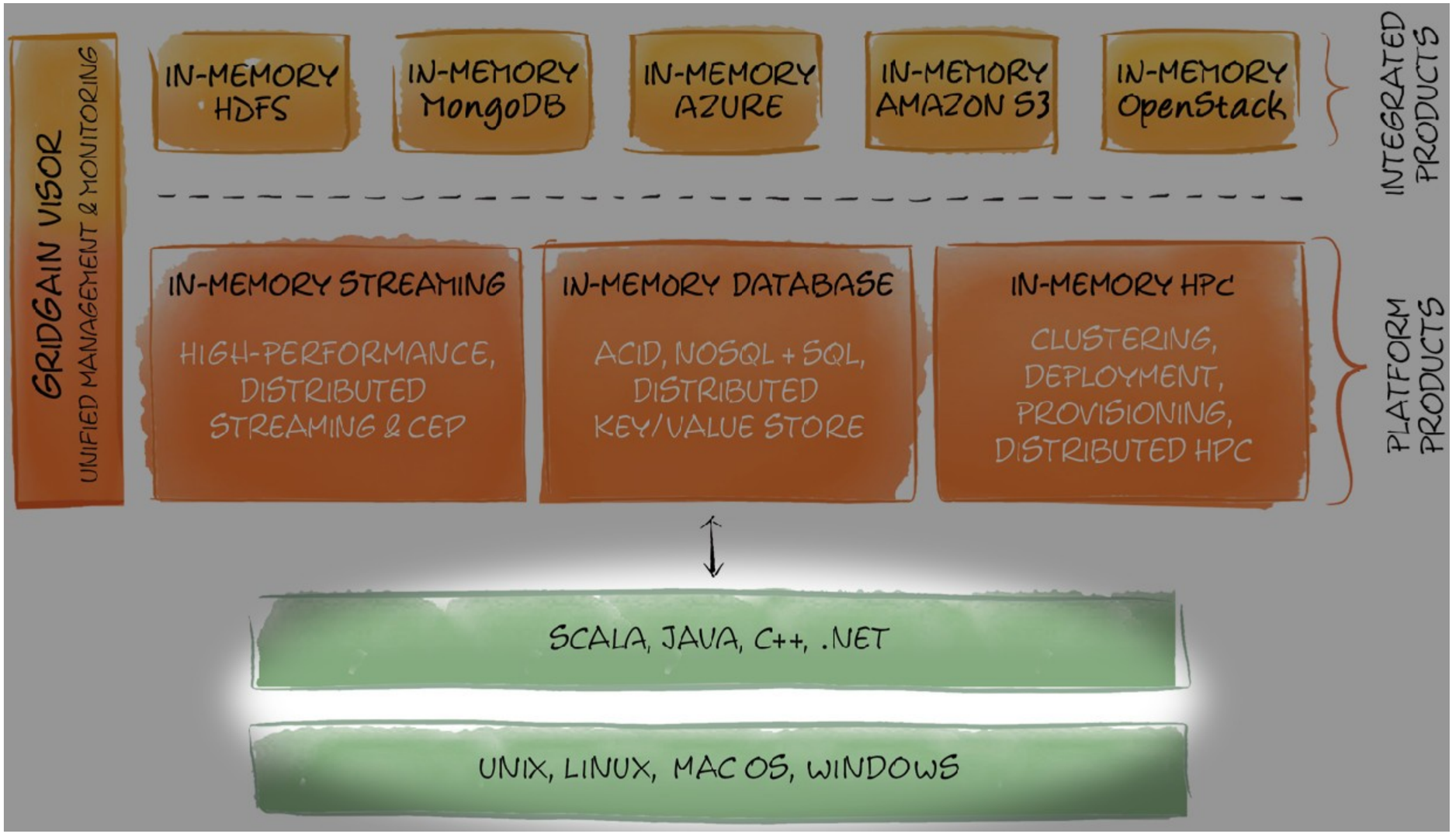
Agenda

- GridGain Technology Overview
- Why In-Memory Accelerator For MongoDB?
- How to Accelerate
- Feature And Performance Comparison
- QA

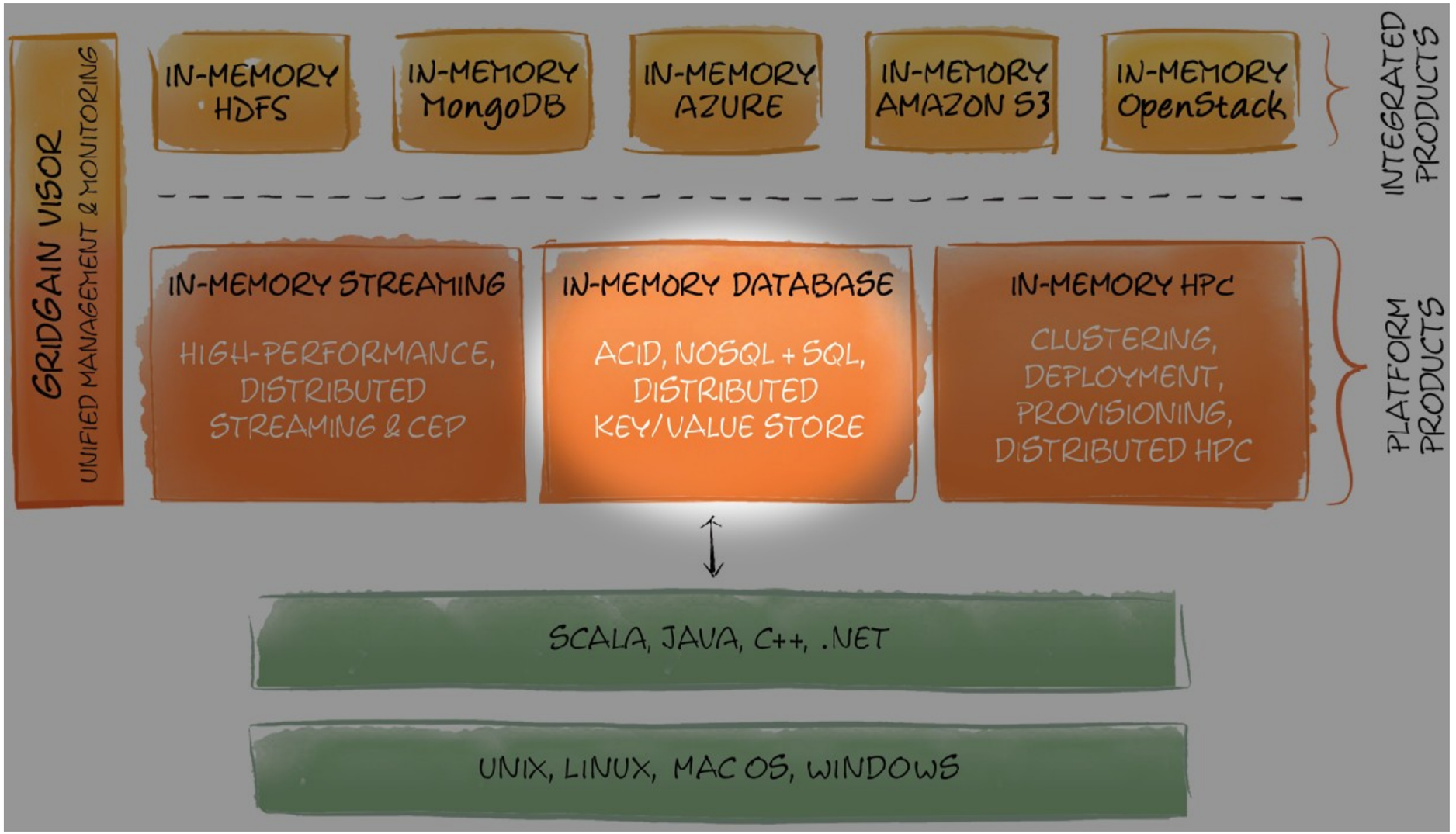
GridGain: Complete In-Memory Stack



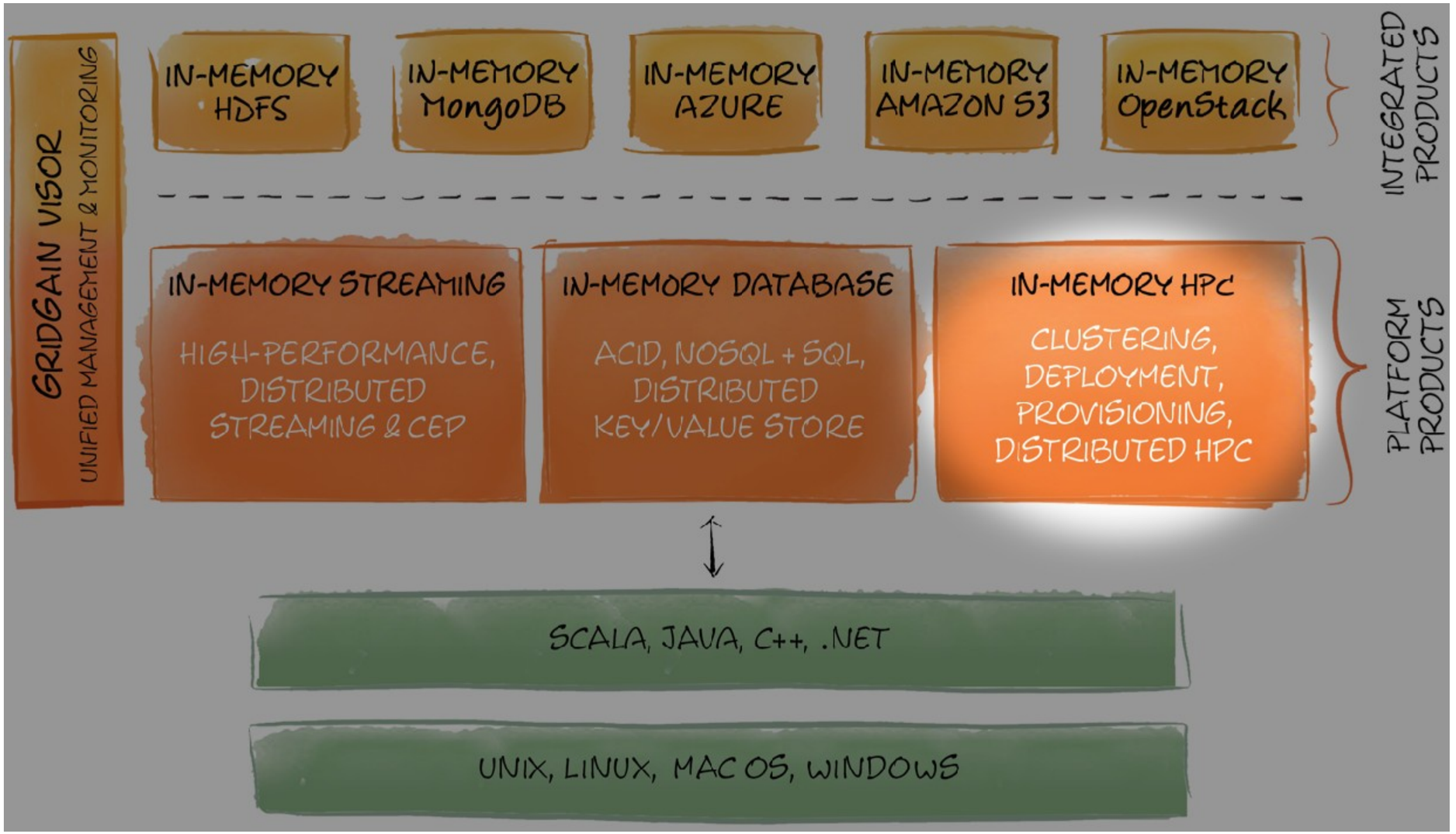
GridGain: Complete In-Memory Stack



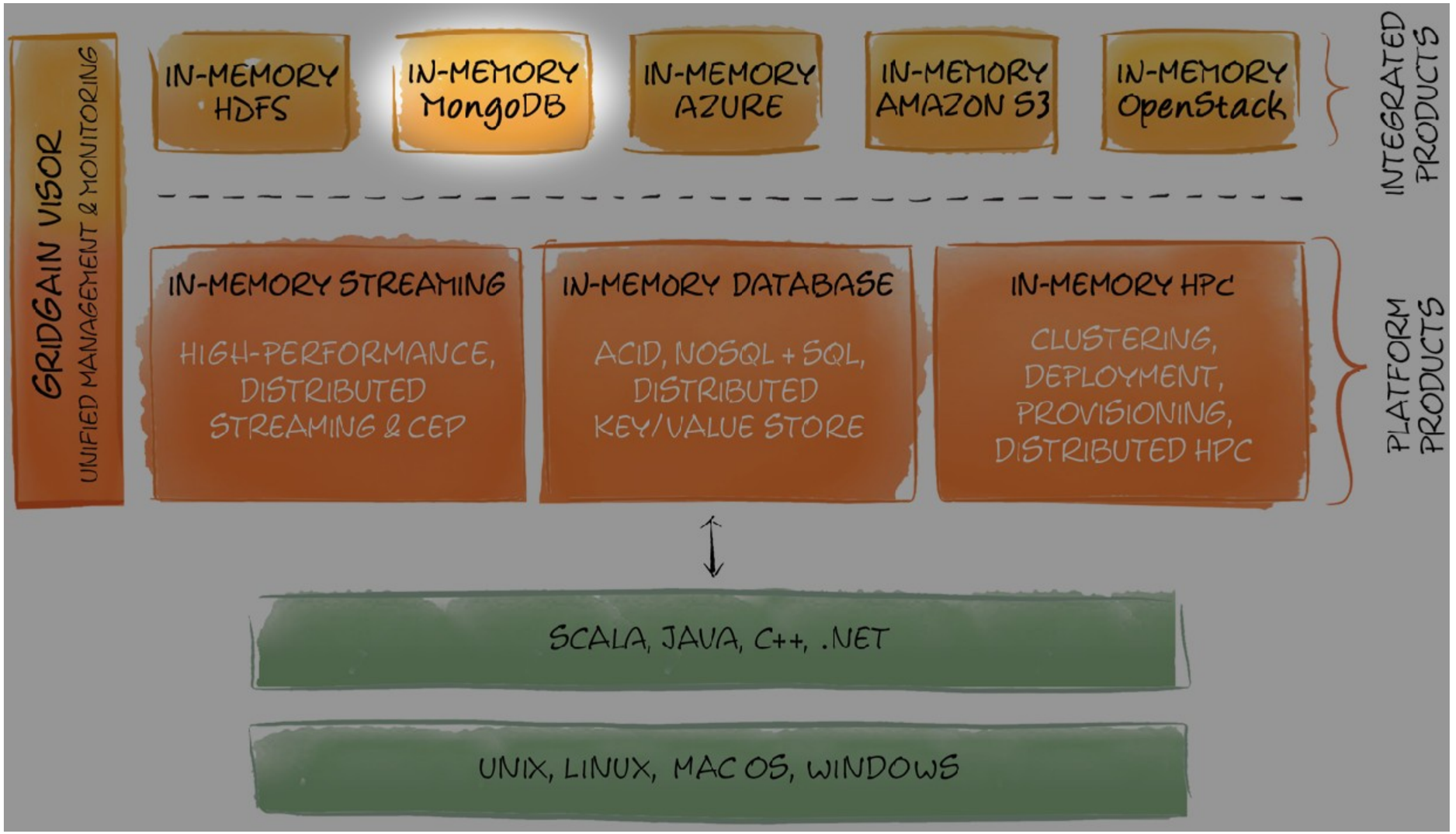
GridGain: Complete In-Memory Stack



GridGain: Complete In-Memory Stack

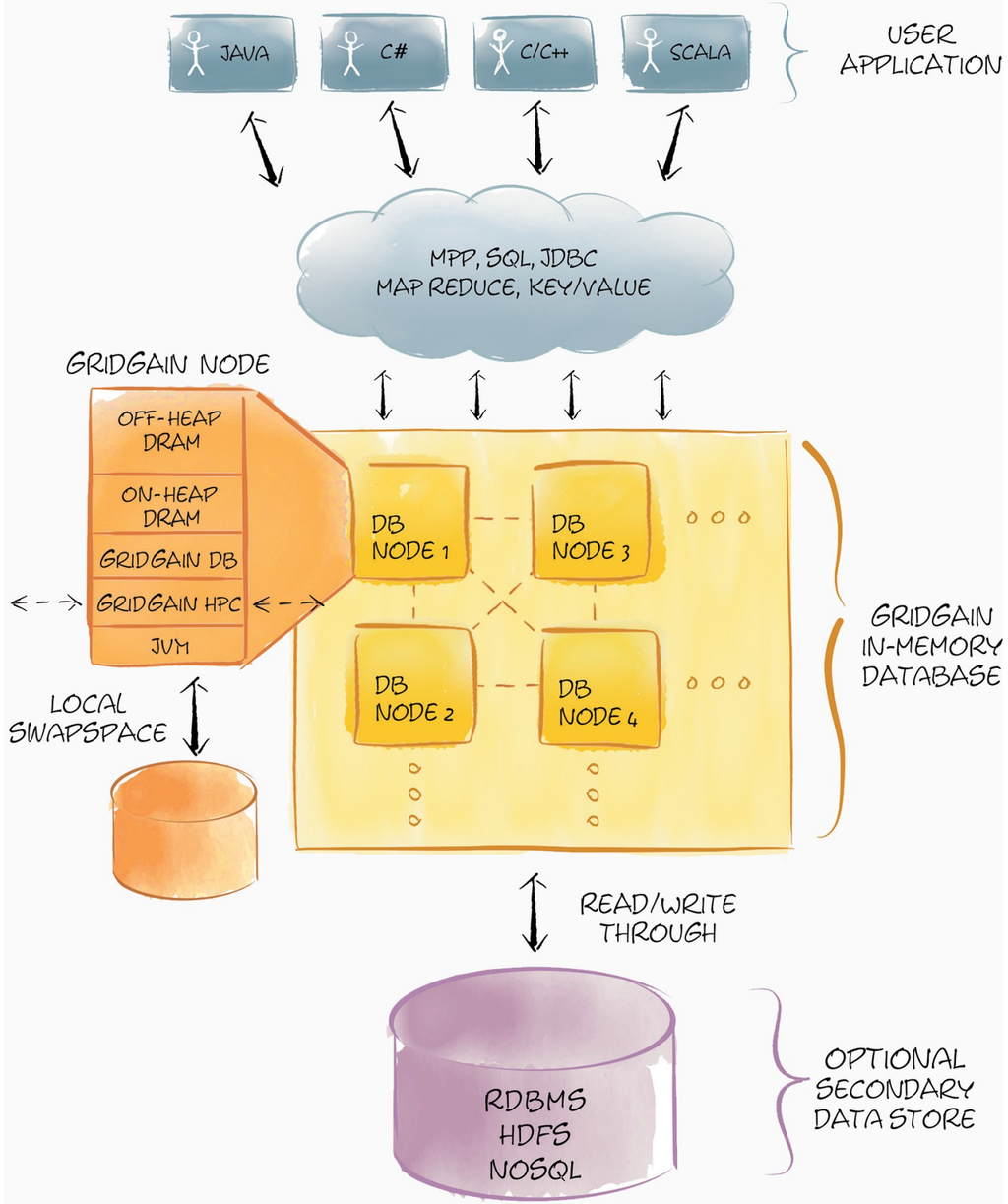


GridGain: Complete In-Memory Stack



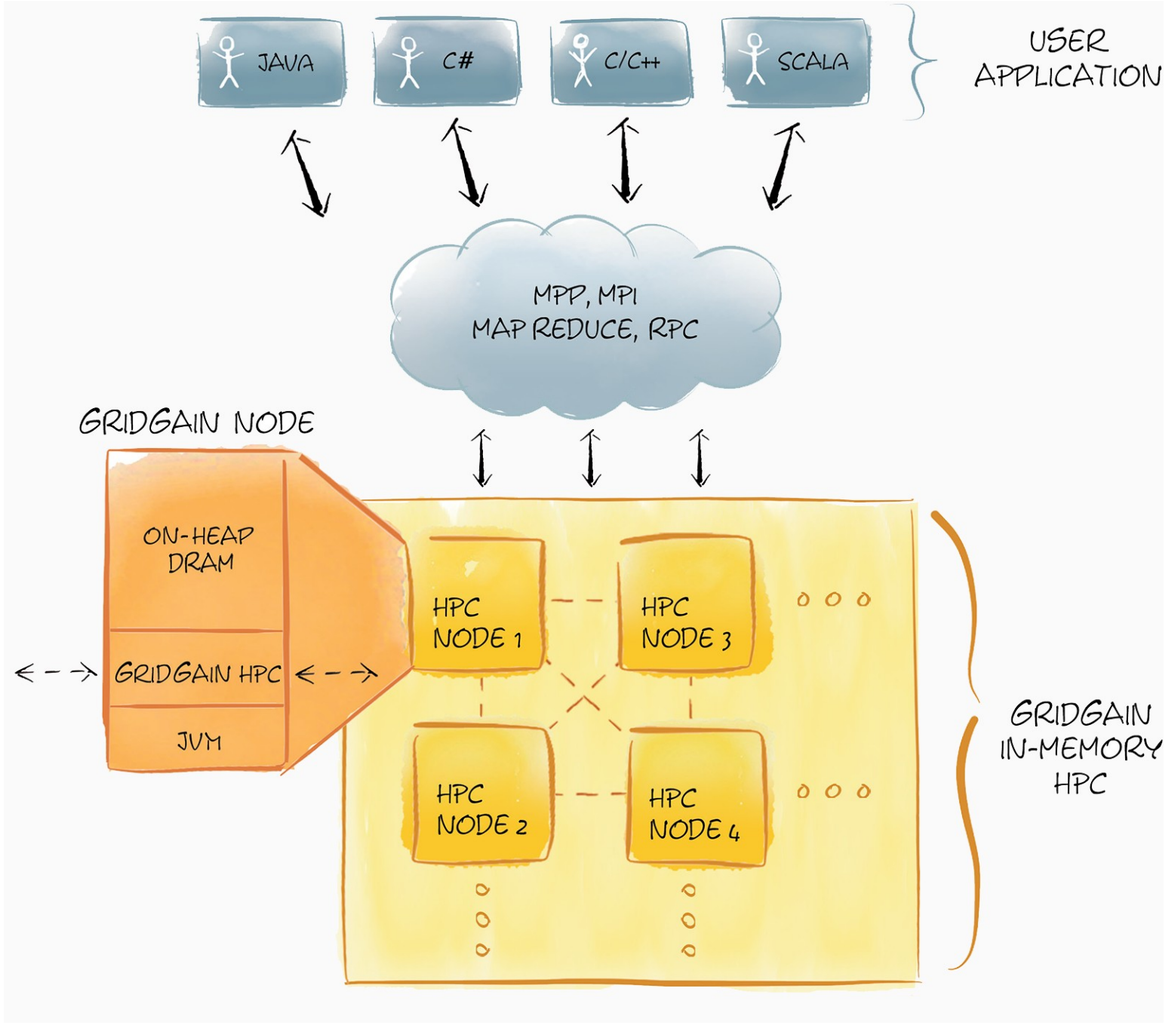
GridGain: In-Memory DataBase

- Local, Replicated, Partitioned
- MVCC-Based Concurrency
- HyperLocking Support
- Off-Heap Memory Support
- Write-Behind Cache
- ACID Transactions
- Pluggable Data Indexing
- SQL and Lucene Querying, custom SQL Functions



GridGain: In-Memory High Performance Computing

- Affinity Collocation
- Zero Deployment
- Fault Tolerance
- Load Balancing
- Collision Resolution
- Job Checkpointing
- Distributed Continuations



In-Memory Accelerator For MongoDB: Disclaimer

- We Do Not Compete with MongoDB
- MongoDB is Very Popular Product And Constantly Growing
- We Like the API and Features

In-Memory Accelerator For MongoDB: Why?

But In Some Cases Mongo Does Not Work

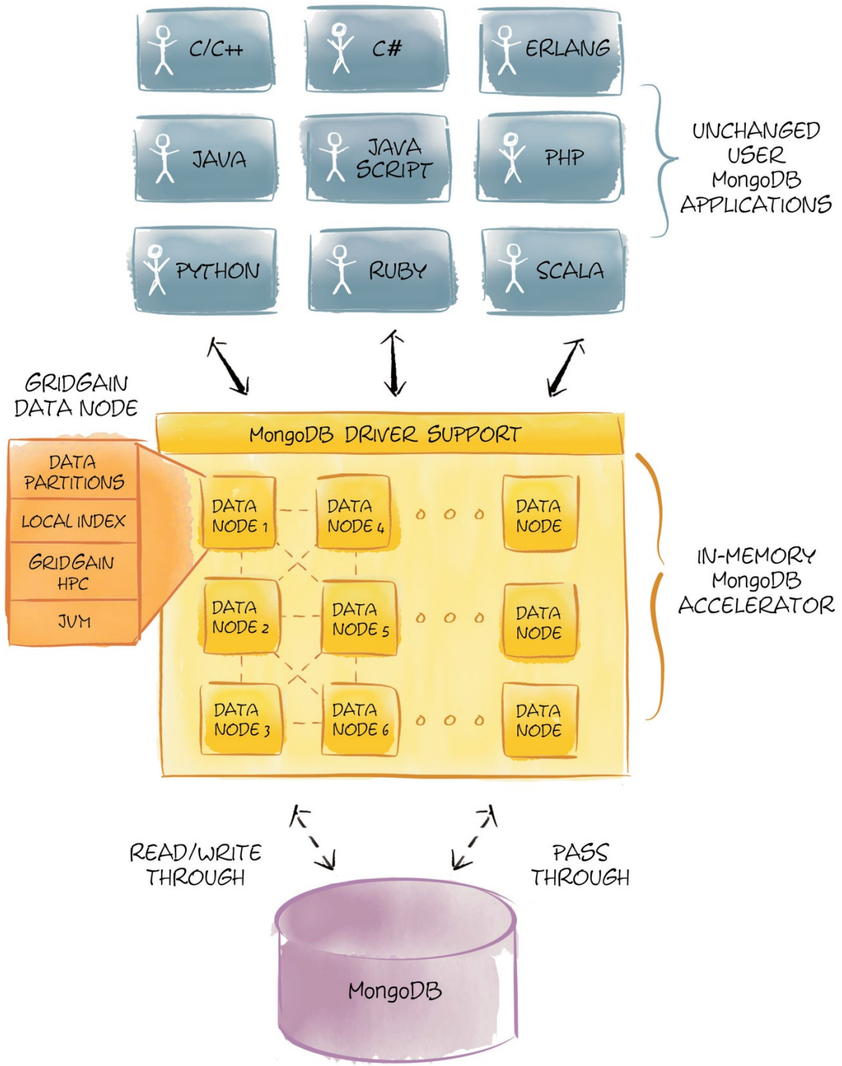
- Database-Level Read/Write Locks
- Disk IO
- Complex Sharded Deployment
- Hardly Scalable

In-Memory Accelerator For MongoDB: Why?

What Can Be Done to Address These Problems?

In-Memory Accelerator For MongoDB: Idea

- Implement Server Supporting The MongoDB Wire Protocol
- Store Documents In IMDB In ***PARTITIONED*** Mode
- Store Metadata In IMDB In ***REPLICATED*** Mode
- Server(s) Routes Queries To Nodes, Aggregates And Sends Results Back



In-Memory Accelerator For MongoDB: Goals

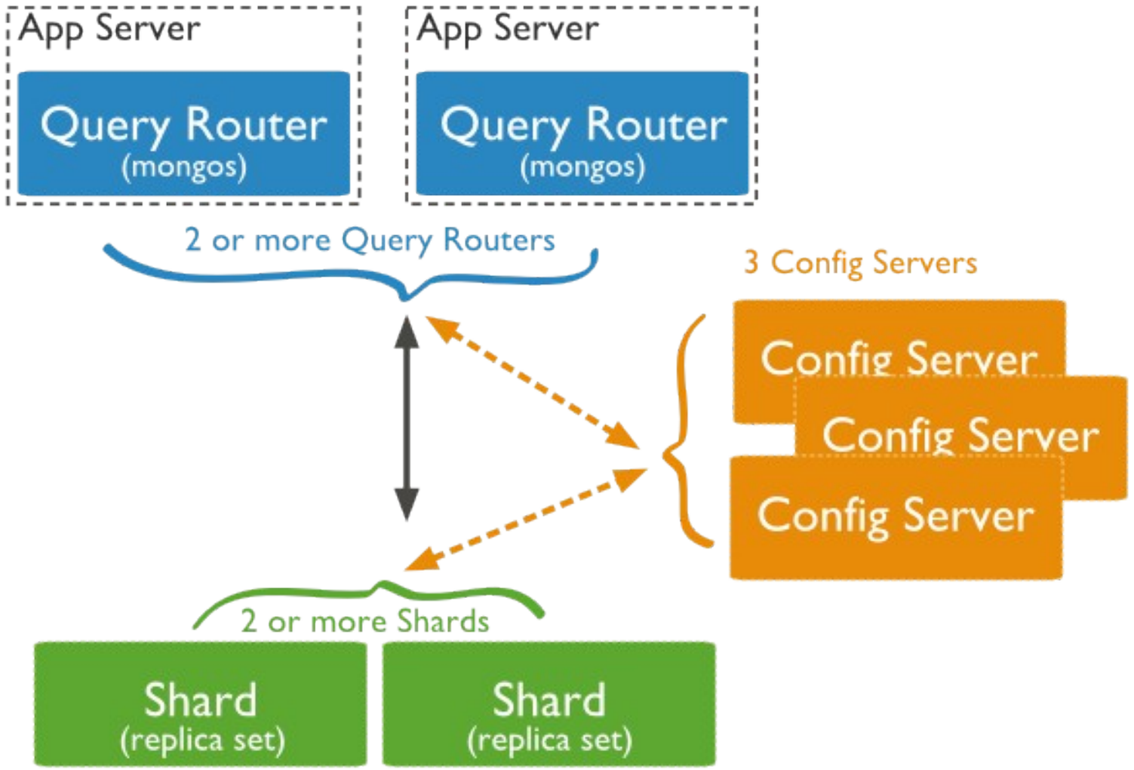
- Up to 10x Better Performance
- Allow Easier Horizontal Scalability to 1,000+ Nodes
- Allow Vertical Scalability to Terabytes Of RAM
- Fully Concurrent In-Memory Processing
- Efficient & Balanced Memory Management
- Transparent Background Repartitioning
- Embedded Mongo API for In-Memory Computing

MongoDB: Sample Use Case

- **Sample Deployment**
  - 5 Shards, 1 TB Each
  - 3 Replicas in Each Shard
  - 15 Servers Total, 3 Config Servers
- **Problem Statement**
  - Small Portion of 1TB is in Memory on Each Server
  - Slower Performance Due to Constant Paging
  - Slower Performance Due to Lack of Concurrency
  - Queries Span Multiple Shards
  - Some Queries Take Minutes

MongoDB: Sharding

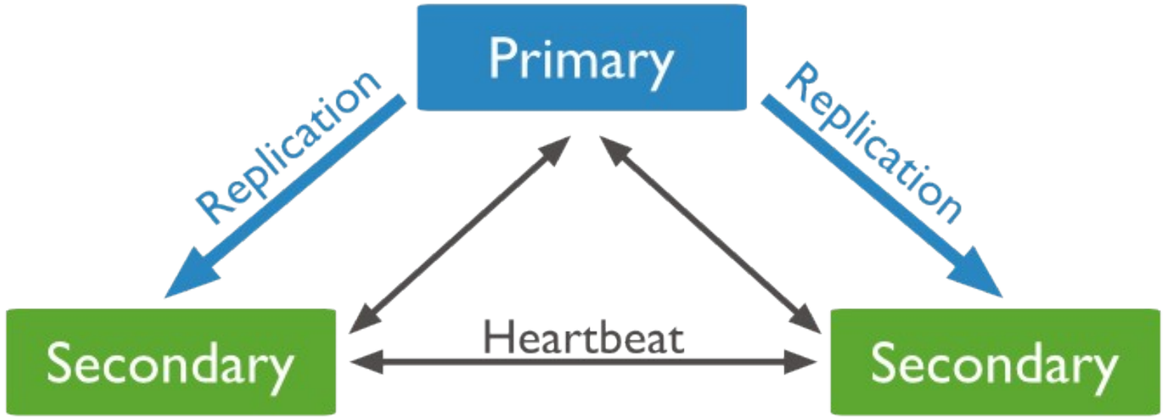
- **Pros**
- Some Parallelism on Writes
- Bigger Throughput



- **Cons**
- Writes are Still Not Fully Parallel
- Query Router Does More Work
- Complex Deployment
- Load May Not Be Even Across Shards

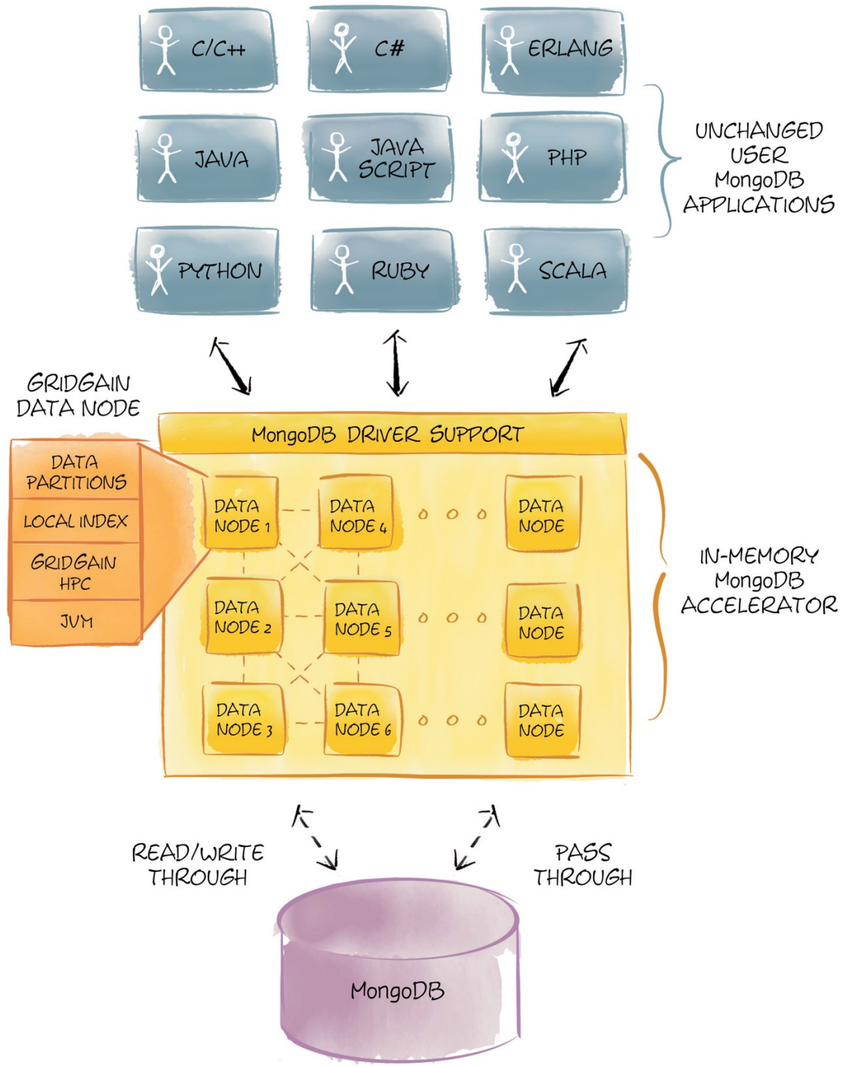
MongoDB: Replica Sets

- **Pros**
- Fault Tolerant
- If Primary Fails, Secondary Becomes Primary
- Queries Can Run in Parallel
- **Cons**
- Excessive Memory Paging
- Difficult to Fit the Whole Dataset In Memory
- Writes are Not Concurrent
- Data is Not Hot on Servers



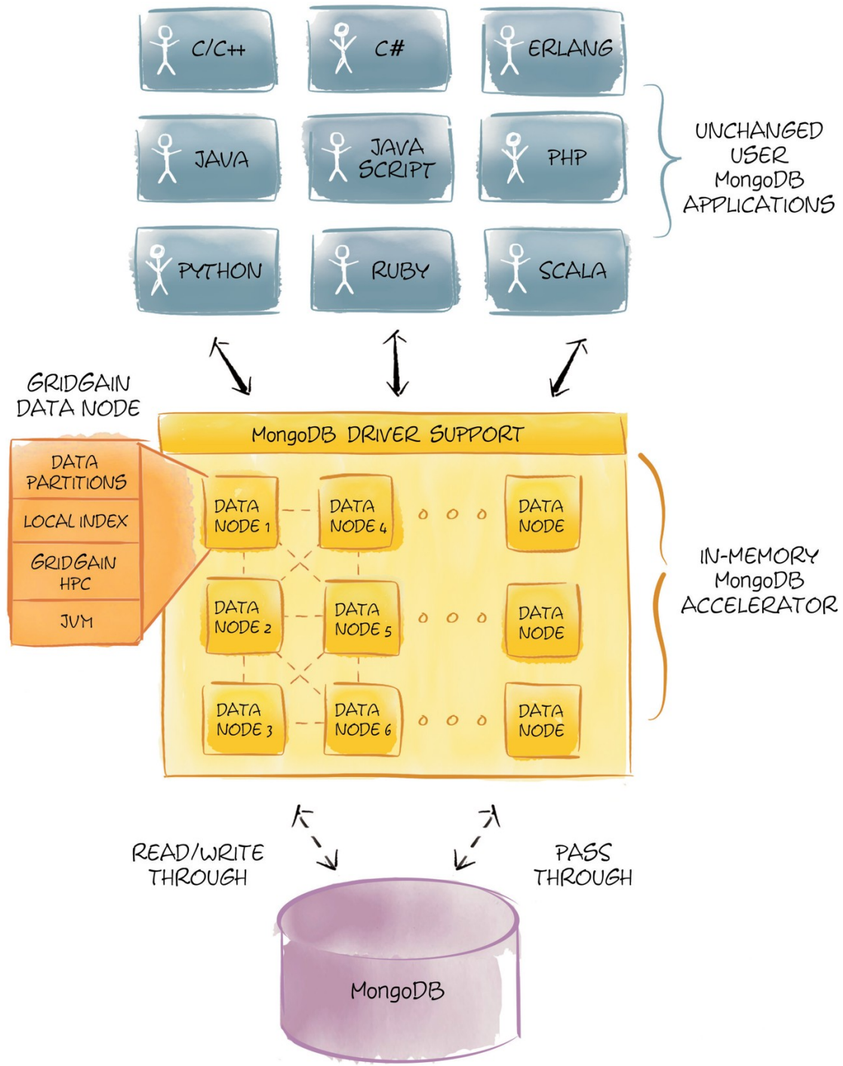
In-Memory Accelerator For MongoDB: Plug-N-Play Integration

- **Plug-N-Play Integration**
  - Zero Code Change
  - Any Mongo Client
- **Full MongoDB Operation Support**
  - Aggregation Framework Supported
  - Distributed Sorting and Grouping Algorithms



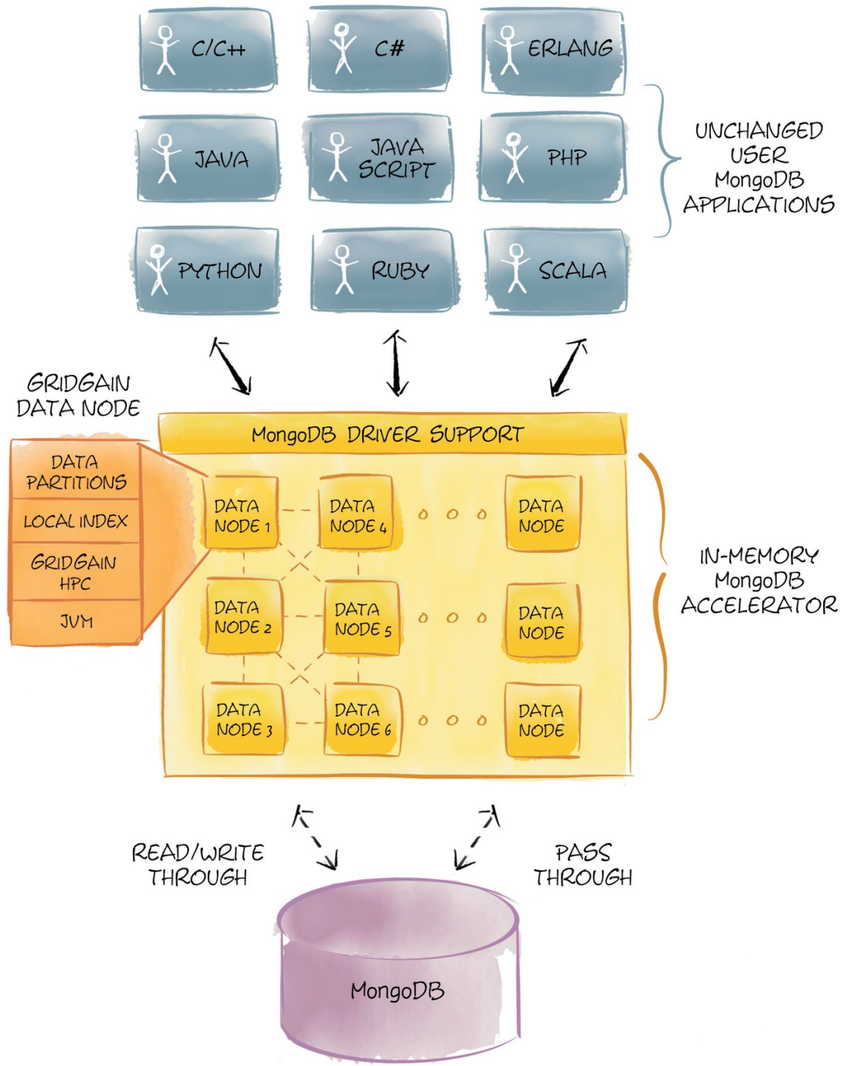
In-Memory Mongo DB Accelerator: Memory First

- **Optimized For Memory-First vs Disk-First**
- Remove Disk I/O
- Remove Memory Paging
- On-Heap and Off-Heap Document Caching
- Direct Memory Access vs Block Data Access



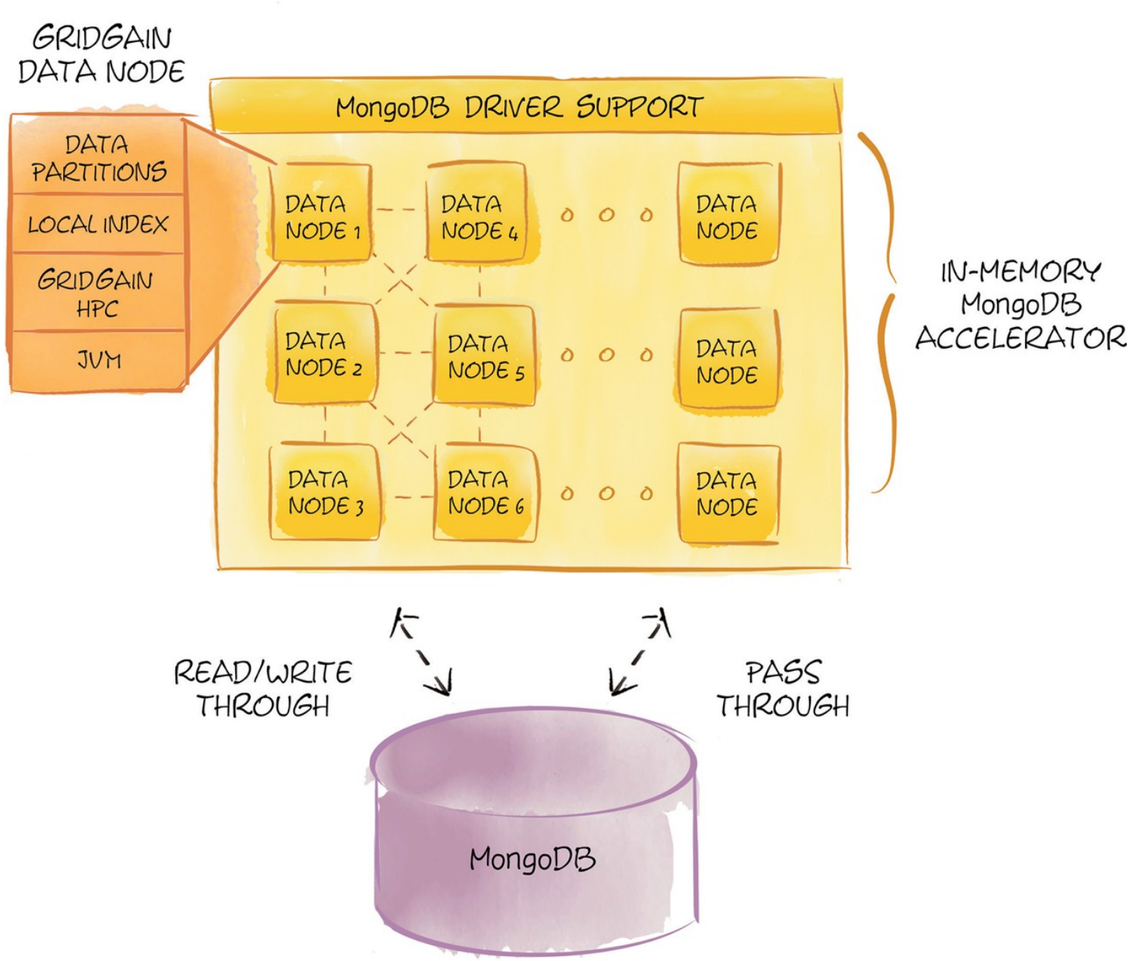
In-Memory Accelerator For MongoDB: Concurrency And Indexing

- **Fully Concurrent**
- No Global Write Locks
- All Writes and Reads Happen Concurrently
- Contention-Free MVCC-based Approach
- **Comprehensive Document Indexing**
- Fully Concurrent Indexes Based on SnapTrees
- On-Heap and Off-Heap Indexing



In-Memory Accelerator For MongoDB: Data Partitioning

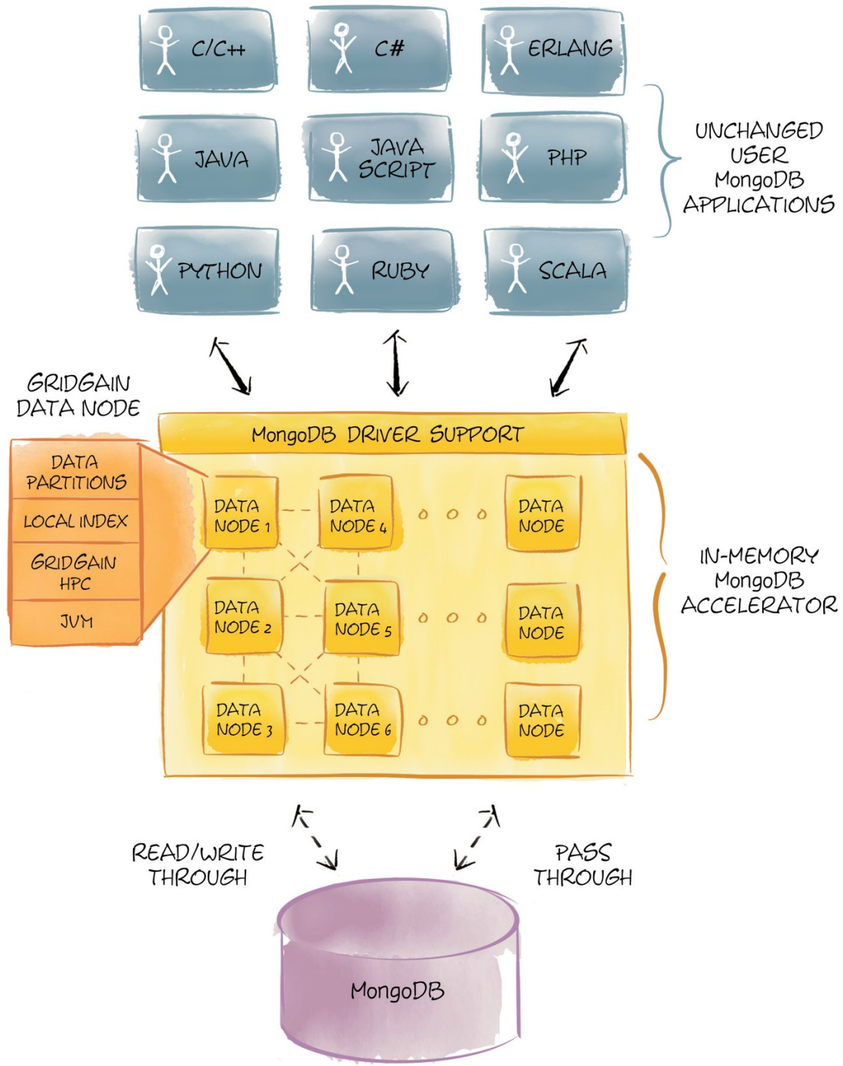
- No Sharding - Data is Split Into Ranges, Not Shards
- No Need for Query Router - *mongos*
- Data Is Evenly Distributed
- Load Is Evenly Distributed
- Backups For Redundancy And Querying
- Writes and Reads are Fully Concurrent



In-Memory Accelerator For MongoDB: Operation Modes & Memory Management

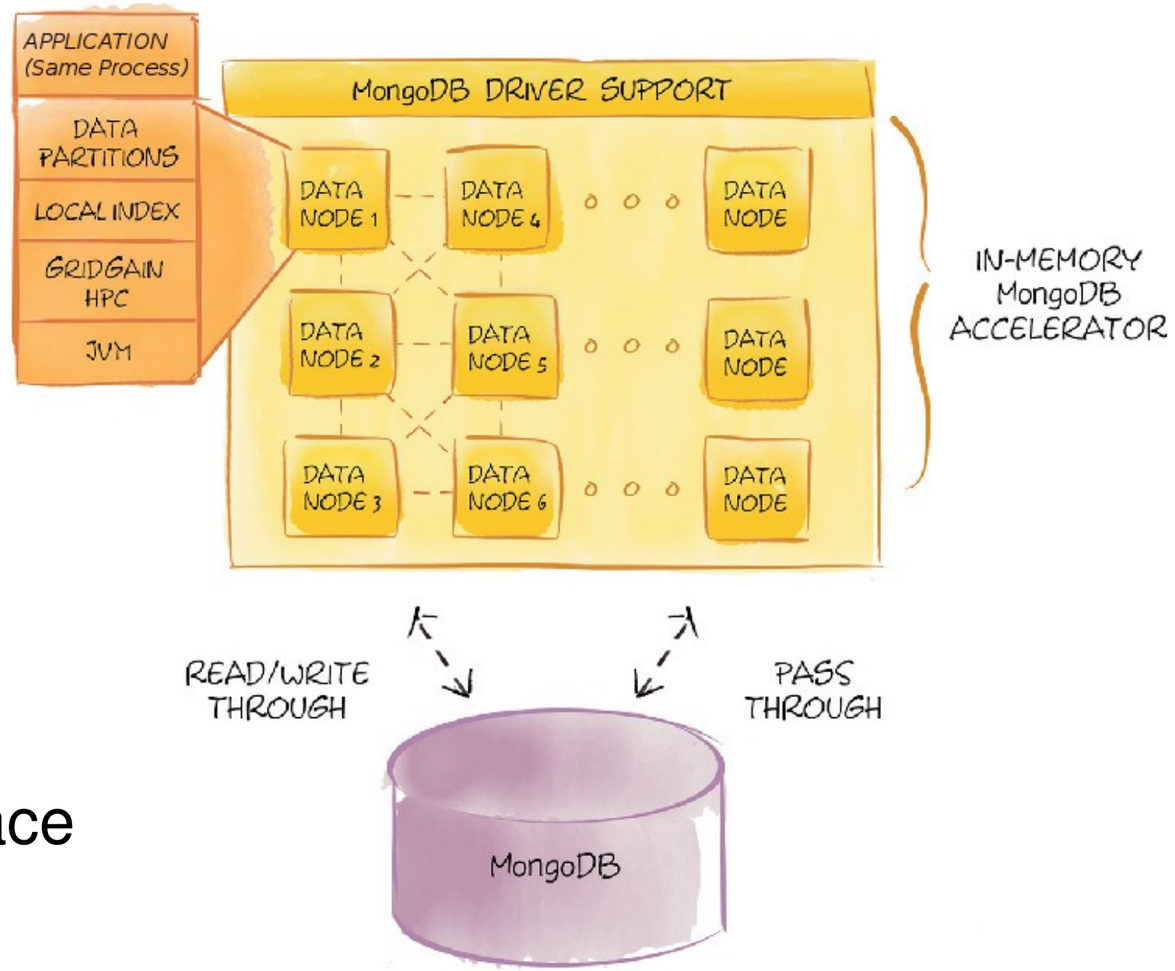
- **Dual Operation Mode**
- PRIMARY - Only In Memory
- PROXY - Only In Native MongoDB
- DUAL - Sync Or Async Writes to MongoDB

- **Efficient Memory Management**
- Pick DBs Or Collections to Store in Memory
- Custom Commands to Load and Unload Data
- Field Name Compaction



In-Memory Accelerator For MongoDB: Embedded Mode

- **Embedded Mode for JVM**
- Remove Unnecessary Network Hops
- Gain Another Performance Increase
- Same API as for MongoDB Java Driver
- Java-Based In-Memory MapReduce
- Extra APIs for Better Affinity Collocation
- Logic and Data in the Same Memory Space



In-Memory Accelerator For MongoDB: Sample Use Case

- **Need to Accelerate?**
  - Handle Large Amount Of Queries In Real Time
  - Handle 3 - 5 Terabytes Of Data
  - Updates Are As Frequent As Reads
  - Under 1 second SLAs

- **How to Accelerate?**
  - Utilize Existing Servers - GridGain Node On Every Server
  - Only Put Highly Utilized Collections In Memory
  - Data Is Fully In Memory
  - No Memory Paging
  - Even Load And Data Distribution
  - Real Time Repartitioning
  - Better Performance For Queries

Comparison: Scalability

- **GridGain's In-Memory Accelerator**
- Memory First
- Scales to 1000s Of Nodes
- Simple Unified Deployment
- Data and Load are Evenly Distributed
- Per-Document Redundancy
- Elastically Add and Remove Nodes

- **MongoDB**
- Disk First
- Best on Under 40 Nodes
- Complex Sharded Deployment
- Data and Load Distribution are Uneven
- Per-Server Redundancy
- Adding and Removing Nodes is Inefficient

Comparison: Concurrency

- **GridGain's In-Memory Accelerator**
- No Global Write Locks
- Fully Concurrent Writes
- Reads Are Concurrent With Writes

- **MongoDB**
- Database-Wide Write Locks
- Writes Are Sequential
- Reads Must Wait For Writes

Comparison: Memory Management

- **GridGain's In-Memory Accelerator**
- Field Name Compaction
- Data is Always In-Memory
- No Memory Paging
- Data is Always Hot
- Faster Querying
- Partial Indexes

- **MongoDB**
- No Field Name Compaction
- Only Small Subset of Data is in Memory
- Constant Paging In and Out Of Memory
- Secondaries Do Not Keep Hot Data in RAM
- Queries Are Slow When Hitting Disk
- No Partial Indexes

Questions and Answers

Private beta: http://www.gridgain.com

Follow GridGain: @gridgain