Database Technology Group

TECHNISCHE UNIVERSITÄT DRESDEN

Thomas Kissinger

DEXTER

Parallel In-Memory Indexing and Direct Query Processing on Prefix Trees

> Application-Level Trends

- *Evolution of DWH Applications*

- *Advanced Analytics*
  - Sophisticated statistical models
  - Machine learning

→ Mixed worklods

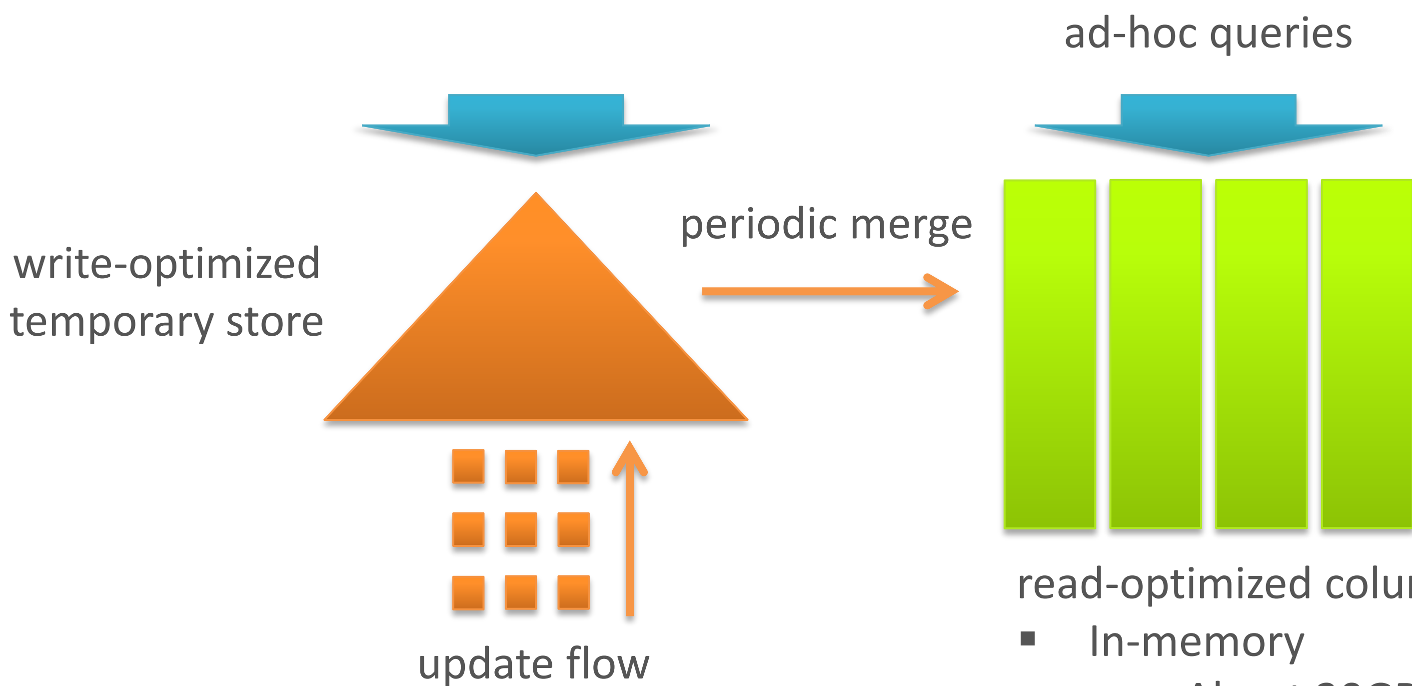
- *Operational BI*

→ High update rates

→ Transactional workloads

> State-of-the-Art System Architecture

- *SAP HANA, C-Store, …*



read-optimized column stores
- In-memory
  - About 30GB/s theoretical peak read performance
- Highly parallel

- *Shortcomings*
  - Poor point query support (e.g. Advanced Analytics)
  - Additional query of temporary store necessary
  - Periodic merge rebuilds column store and stalls queries in the meantime

- *Shifting Memory Hierarchy*
  - In-memory databases/indexing
  - Cache-awareness
  - Other Block/MTU sizes
  - Sequential access still faster than random access

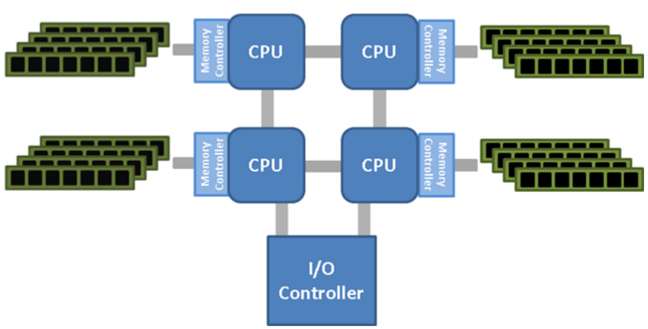
- *More and more Cores*
  - clock rate got stuck
  - Massive parallelism
  - Optimized software required

I7-2600
8 HW Threads

Xeon E7
20 HW Threads

Knight's Ferry
128 HW Threads

- *Modern Hardware Architectures*
  - SIMD instructions
  - Multiple memory channels
  - SMP changes to NUMA
  - Heterogeneous Hardware (GPUs, Co-processors …)
  - Adaptive Hardware (FPGAs, …)



> State-of-the-Art In-Memory Indexes

- **FAST: fast architecture sensitive tree search on modern CPUs and GPUs**
  - **SIGMOD 2010**

    Changkyu Kim, Jatin Chhugani, Nadathur Satish, Eric Sedlar, Anthony D. Nguyen, Tim Kaldewey, Victor W. Lee, Scott A. Brandt and Pradeep Dubey
    - Memory hierarchy optimized binary tree
    - Read-optimized index structure
    - 51M reads/s, but only 10 updates/s for a tree of 64M keys

- **PALM: Parallel Architecture-Friendly Latch-Free Modifications to B+ Trees on Many-Core Processors**
  - **VLDB 2011**

    Jason Sewall, Jatin Chhugani, Changkyu Kim, Nadathur Satish and Pradeep Dubey
    - B+-Tree based index
    - Synchronous batch updates to avoid latches

Both index structures suffer from a poor update performance

> Outline

- Introduction and Trends
- DEXTER Project
    - Overview
    - Project Map
- Core Indexing
- Direct Query Processing
- Conclusions

DEXTER Project

*(**Dre**sden Inde**x** for **T**ransactional Access on **E**merging Technologies)*

- *Transactional Index, optimized for mixed queries and high update rates*
- *Sponsored by SFB 912: HAEC (Highly Adaptive Energy-Efficient Computing)*

Matthias Böhm

Thomas Kissinger

Peter B. Volk

Benjamin Schlegel

Ulrike Fischer

Dirk Habich

> DEXTER Project Map

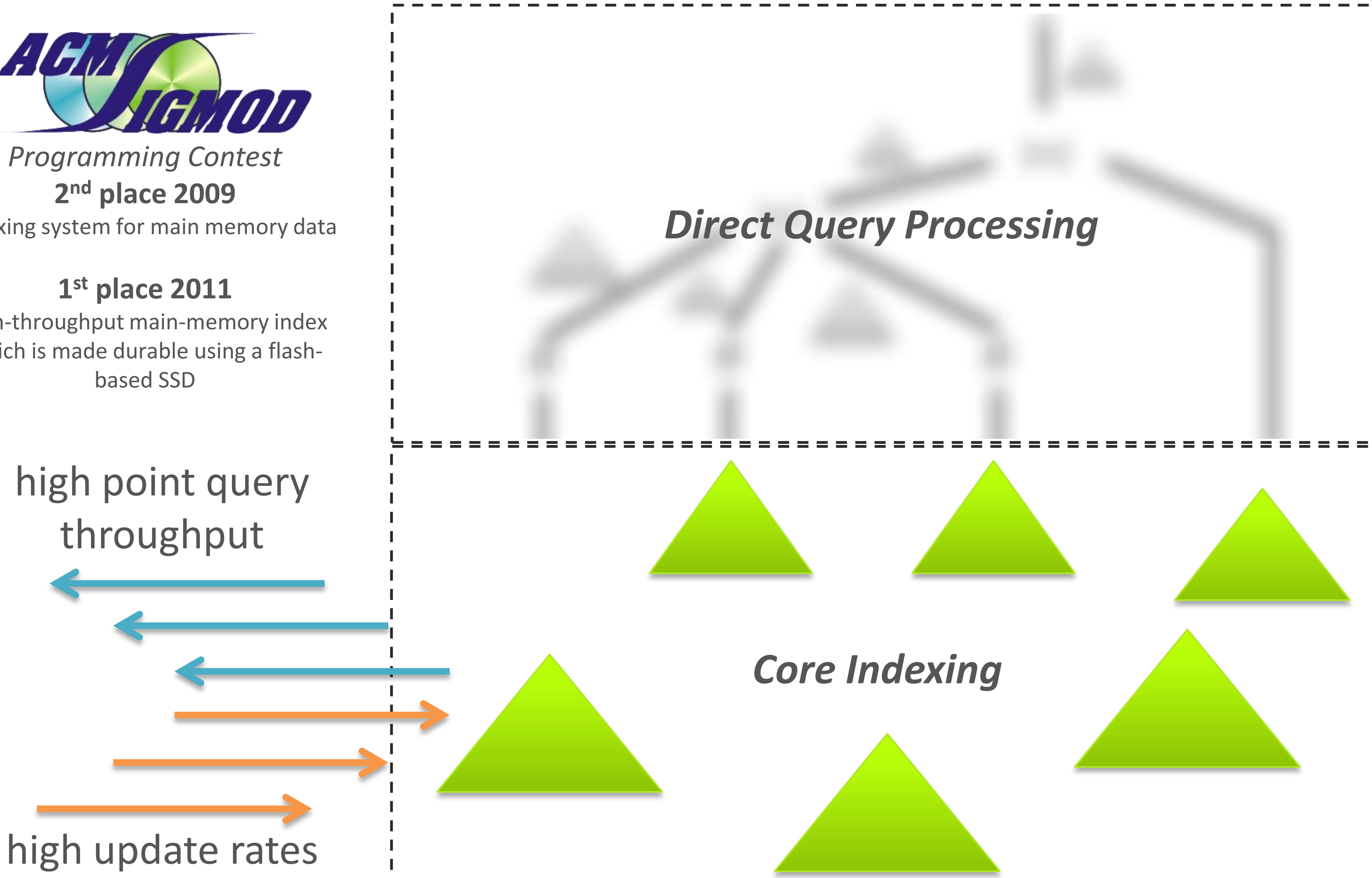
Programming Contest

2nd place 2009

indexing system for main memory data

1st place 2011

high-throughput main-memory index which is made durable using a flash-based SSD



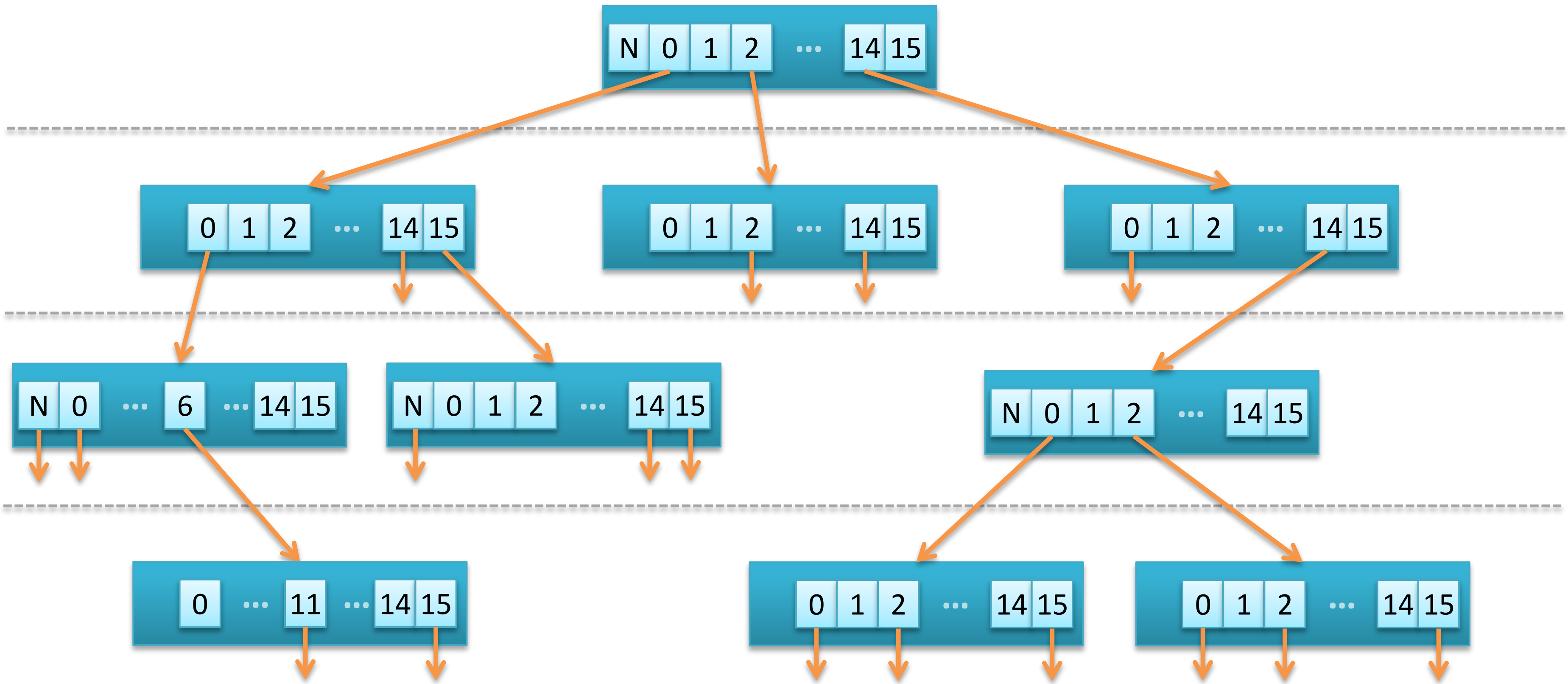
> Outline

- Introduction and Trends
- DEXTER Project
- Core Indexing
  - Generalized Prefix Trees
  - Design Space for parallel In-Memory Indexing
- Direct Query Processing
- Conclusions

> Core Indexing Structure: Prefix Tree

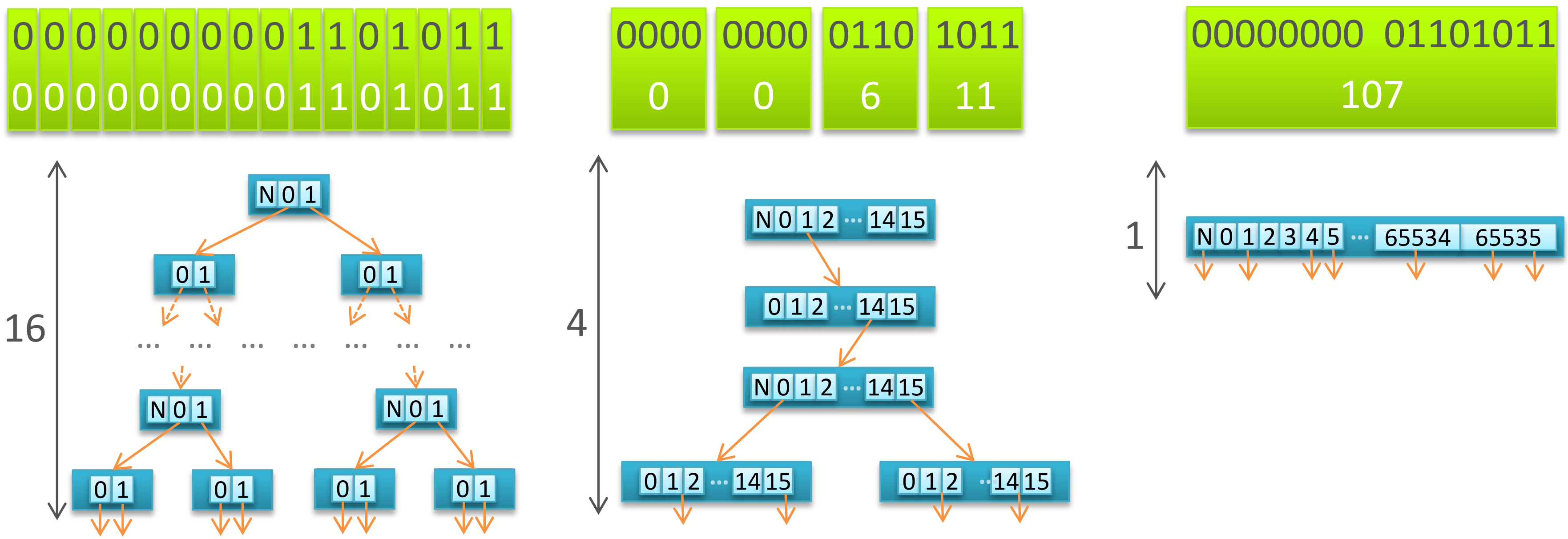
Static Prefix Length: **k' = 4**

- Only one 64bit access per node
- Deterministic path
- No Balancing



> Prefix Tree: Configurations

Static Prefix Length: **k’ = 1** | Static Prefix Length: **k’ = 4** | Static Prefix Length: **k’ = 16**



flatter tree → Less memory transfers

deeper tree → More memory transfers / Better memory utilization

Variable prefix length on tree level or node level possible

- *Two important dimensions for parallel in-memory indexing*

Partitioning \ Batching	No	Yes
No	FAST DEXTER	DEXTER Batched Reads
Yes		PALM DEXTER BUZZARD

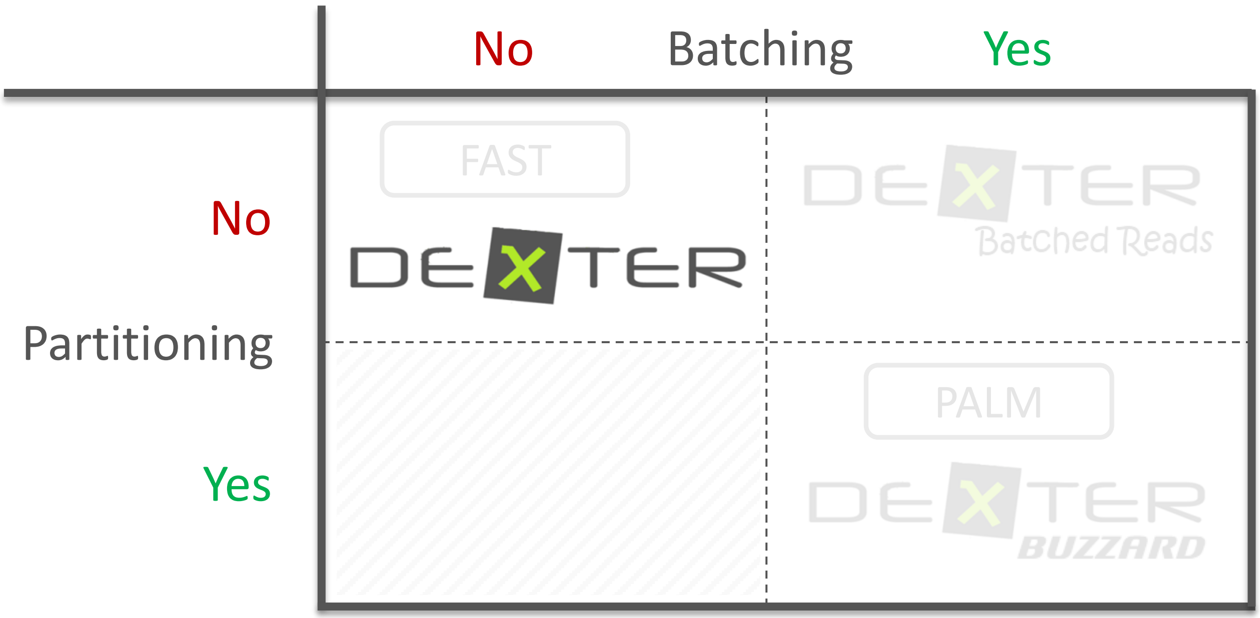
Batching = No:
- \+ Low latency

Batching = Yes:
- \+ High Throughput
- − Increased Latency

Partitioning = No:
- \+ Direct Processing
- − Synchronization

Partitioning = Yes:
- \+ Independent Data
- − Indirect Processing

- *Two important dimensions for parallel in-memory indexing*



\+ Direct Processing

– Synchronization

\+ Independent Data

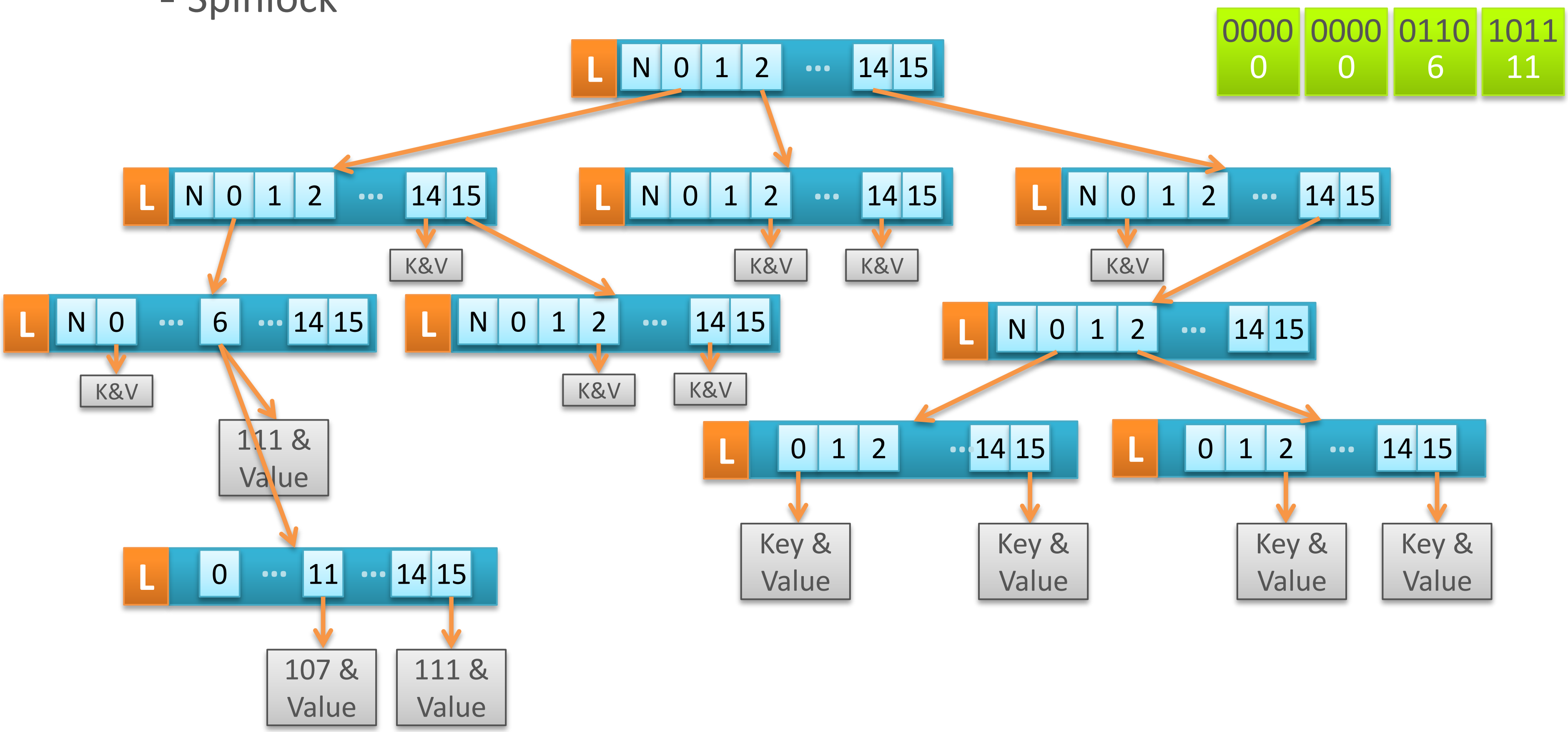
– Indirect Processing

\+ Low latency

\+ High Throughput

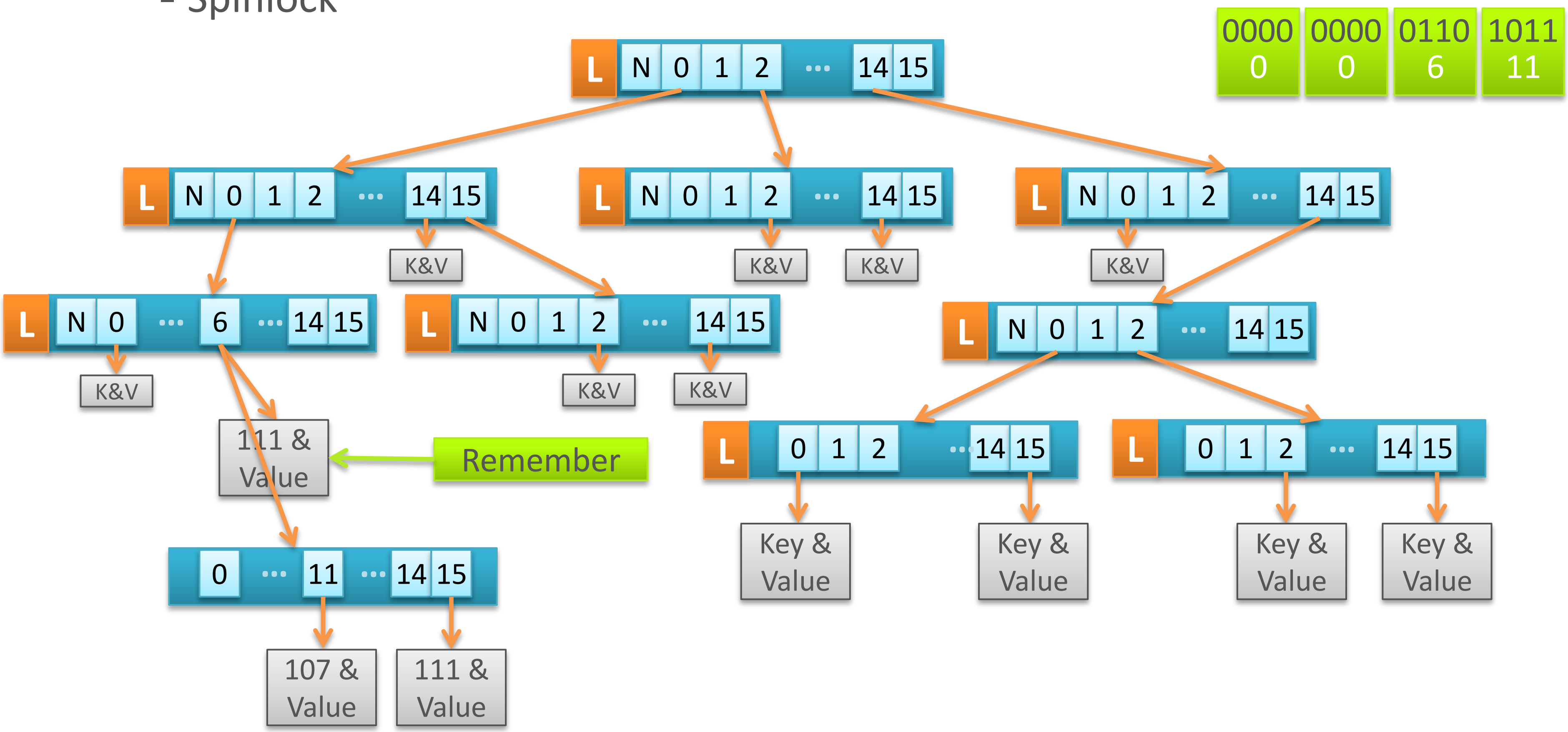
– Increased Latency

- Heavyweight Latches
  - Mutex/Futex
  - Spinlock



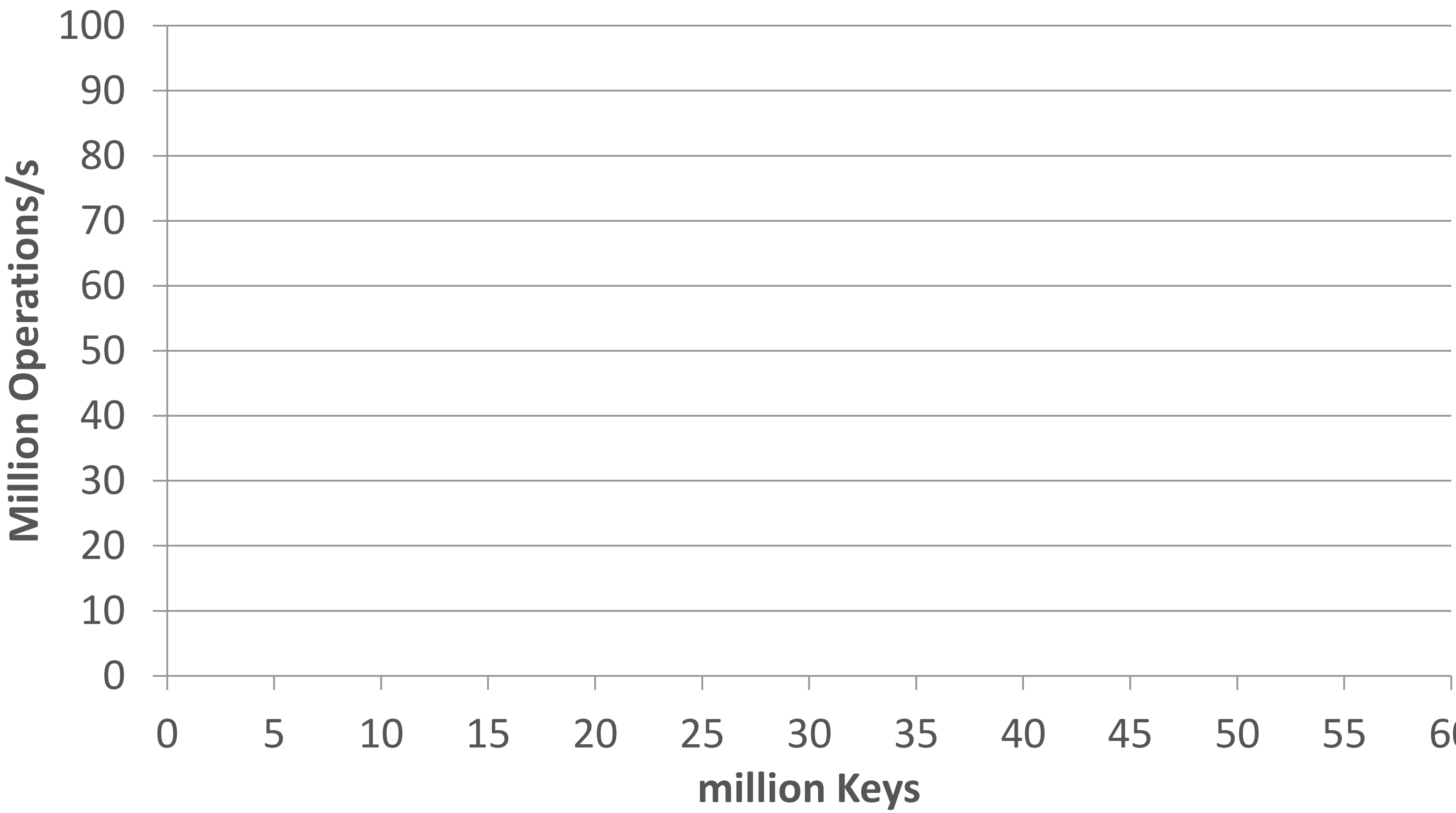
Read-Copy-Updates (RCU) - aware Memory Manager

- Heavyweight Latches
  - Mutex/Futex
  - Spinlock
- Lightweight Atomics
  - Compare-and-Swap (CAS)



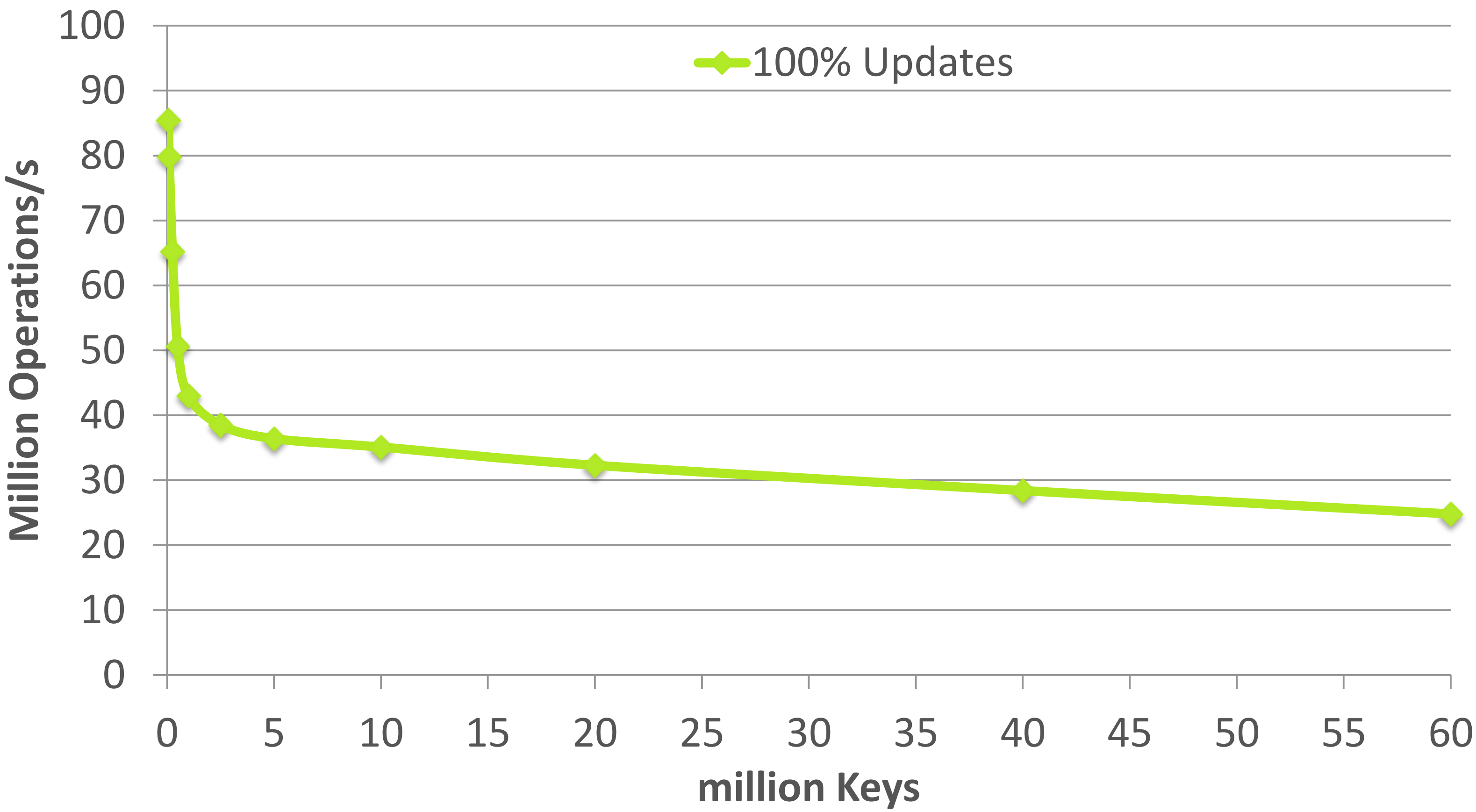
Read-Copy-Updates (RCU) - aware Memory Manager

- *Intel i7-2600 (4 cores @3.4GHz, 2 memory channels, 16GB DDR3 @1333MHz)*



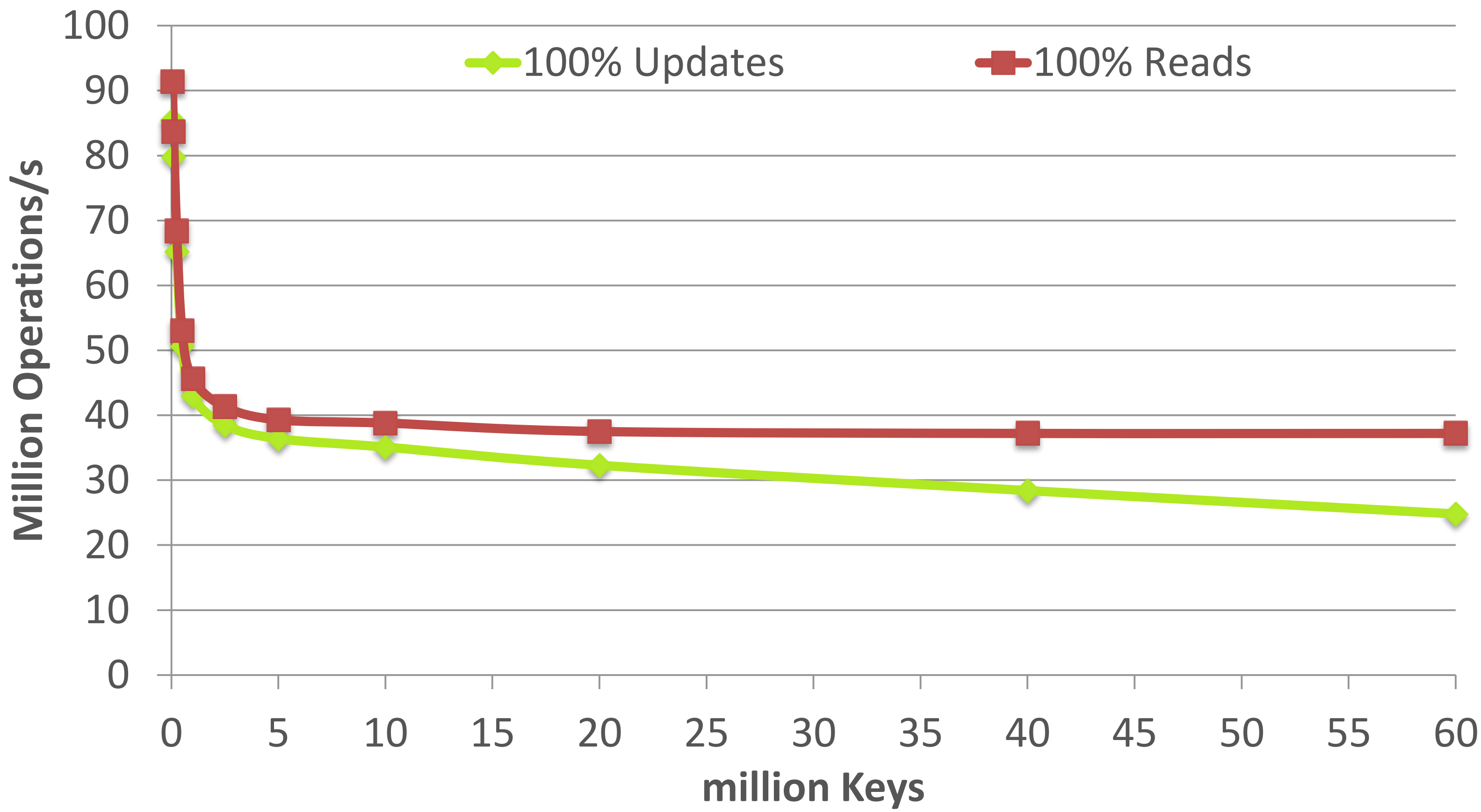
> Parallel DEXTER: Evaluation

- *Intel i7-2600 (4 cores @3.4GHz, 2 memory channels, 16GB DDR3 @1333MHz)*



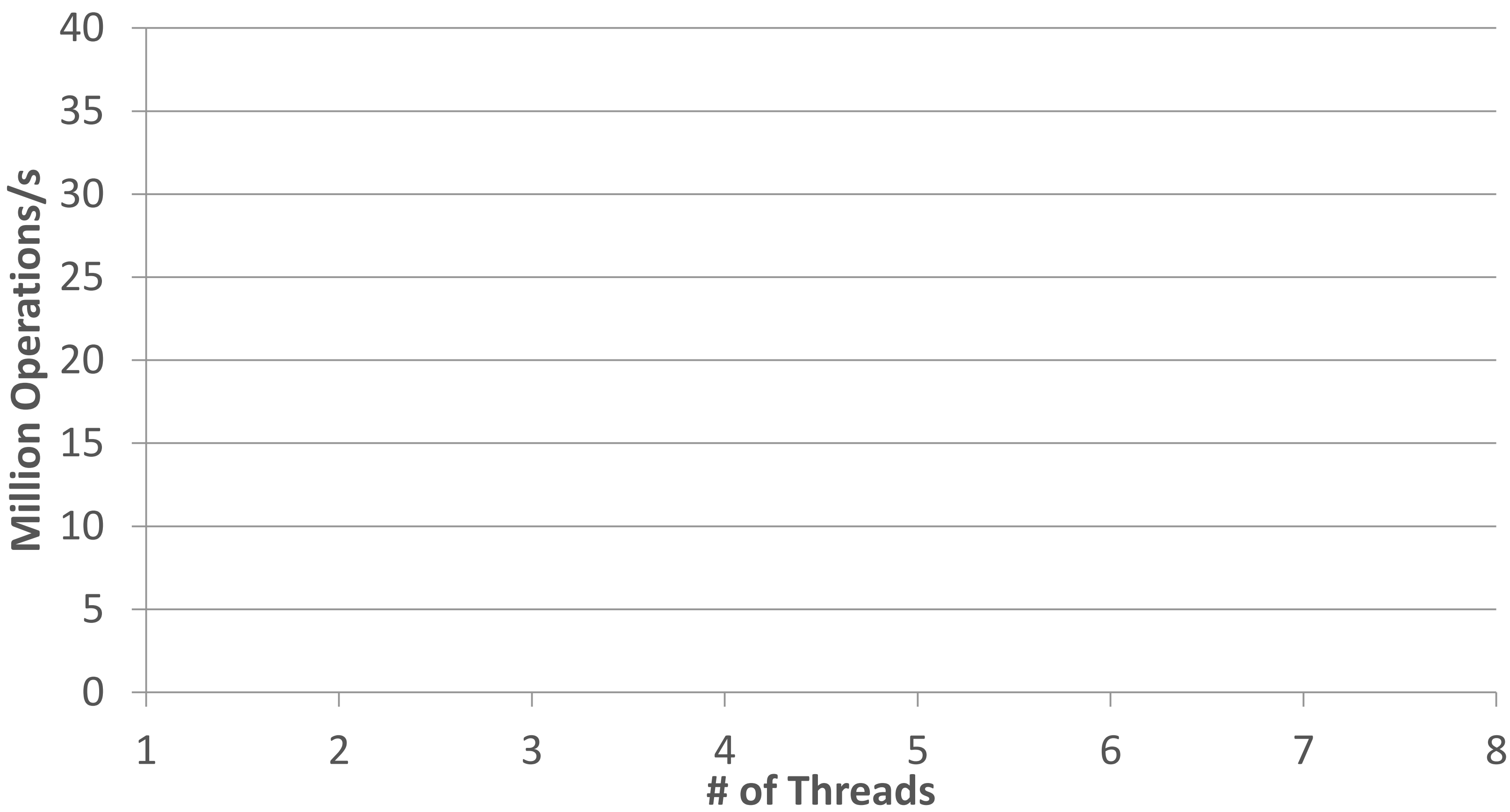
> Parallel DEXTER: Evaluation

- *Intel i7-2600 (4 cores @3.4GHz, 2 memory channels, 16GB DDR3 @1333MHz)*



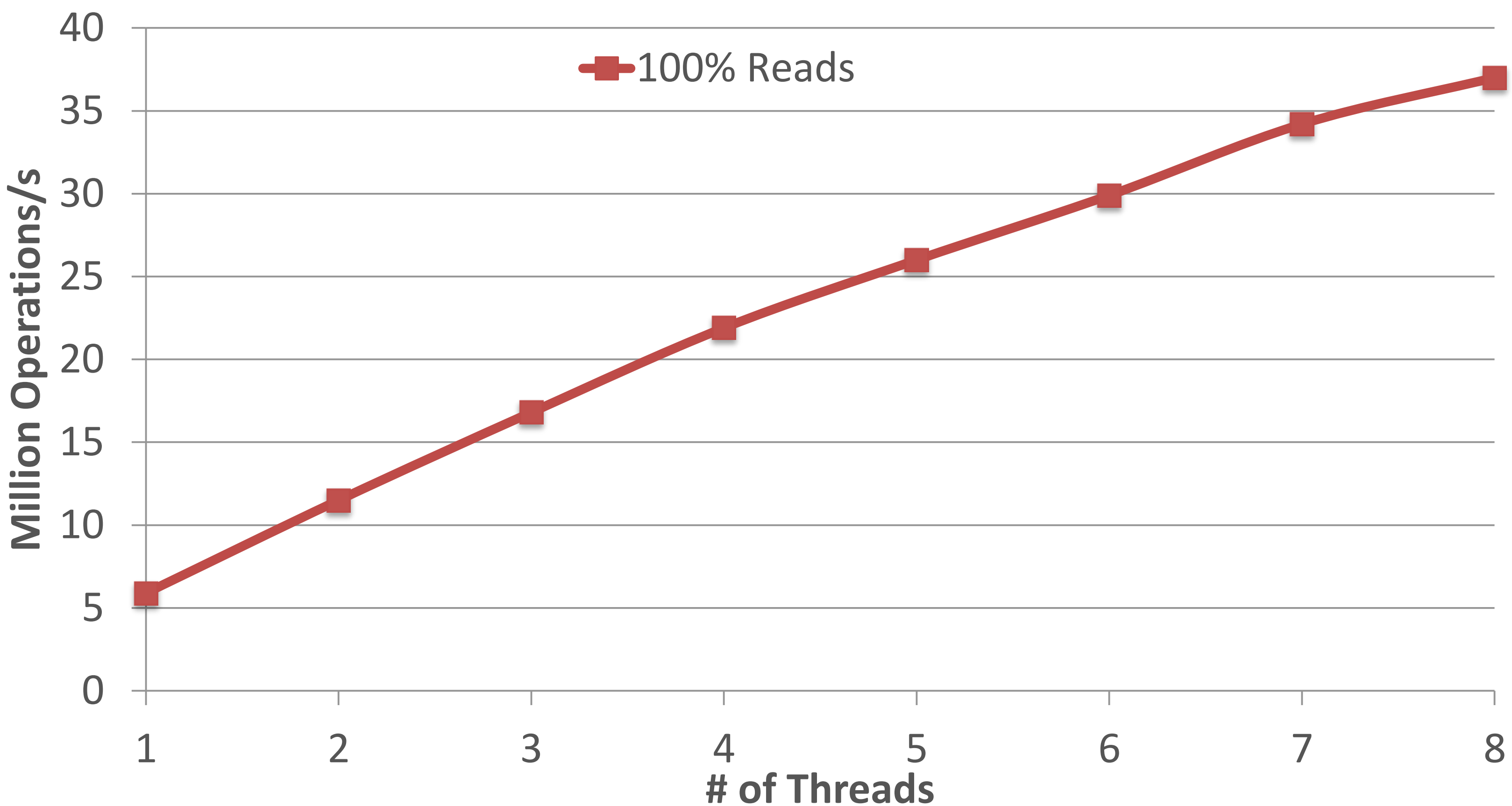
> Parallel DEXTER: Evaluation

- *Intel i7-2600 (4 cores @3.4GHz, 2 memory channels, 16GB DDR3 @1333MHz)*
- *20M Keys (32bit)*



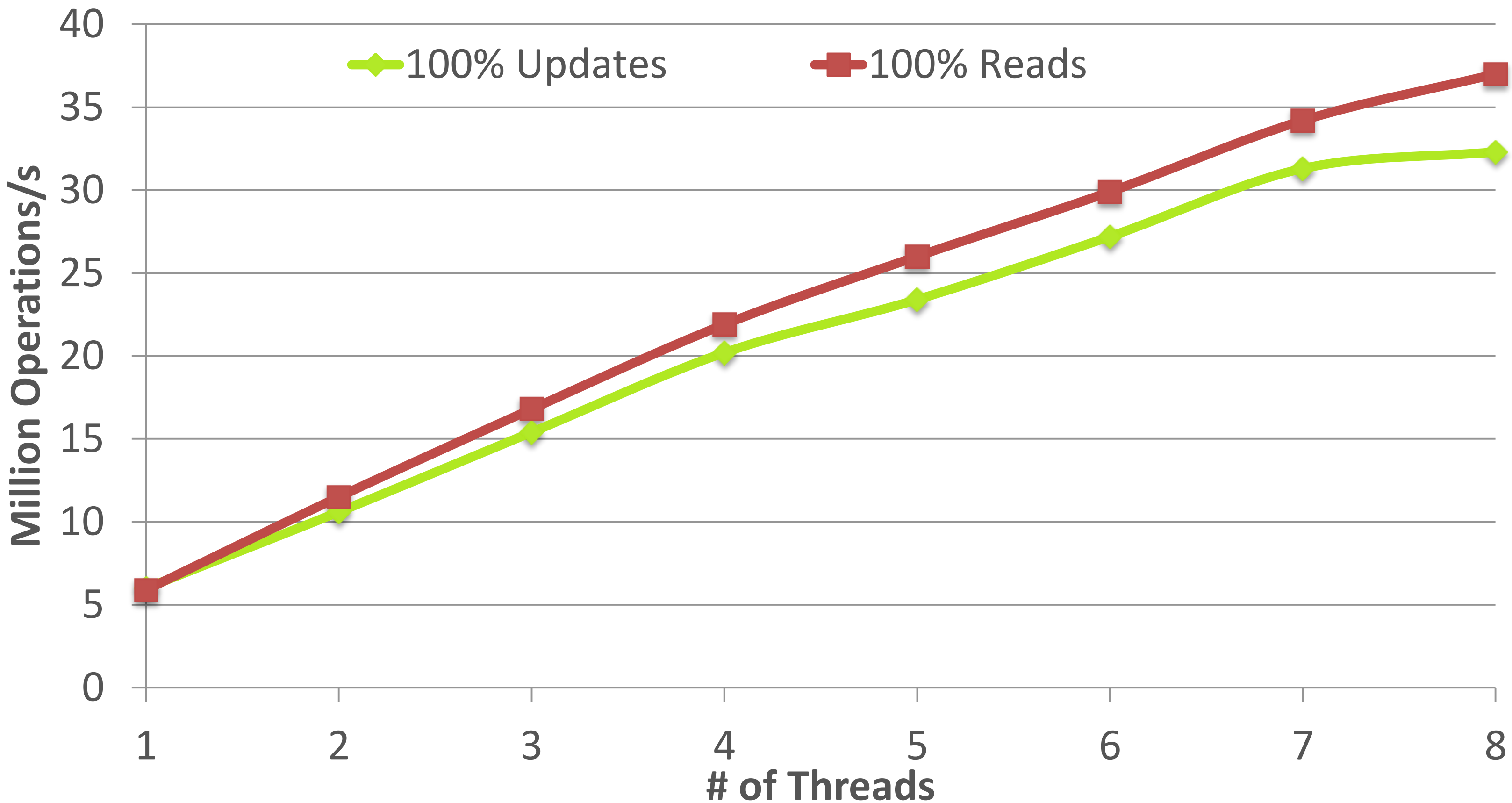
> Parallel DEXTER: Evaluation

- *Intel i7-2600 (4 cores @3.4GHz, 2 memory channels, 16GB DDR3 @1333MHz)*
- *20M Keys (32bit)*

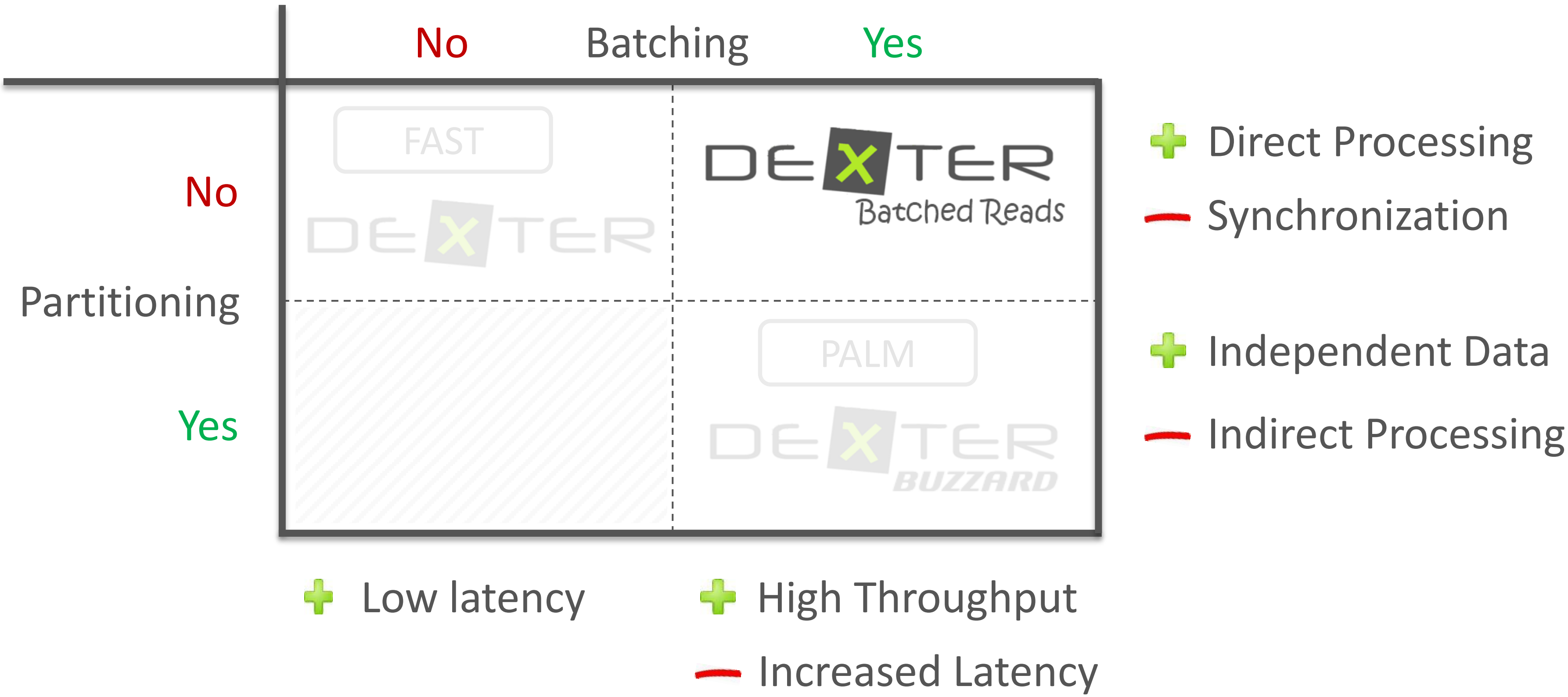


> Parallel DEXTER: Evaluation

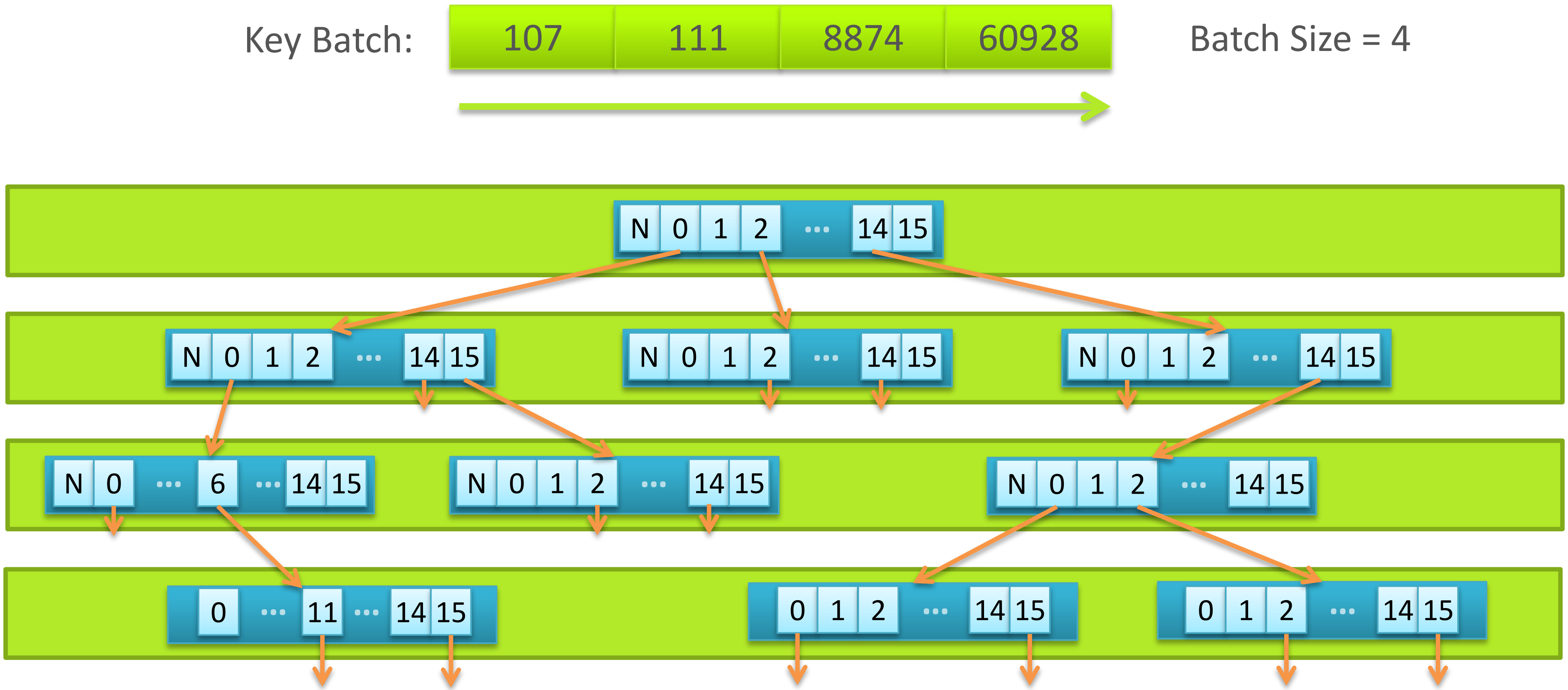
- *Intel i7-2600 (4 cores @3.4GHz, 2 memory channels, 16GB DDR3 @1333MHz)*
- *20M Keys (32bit)*



- *Two important dimensions for parallel in-memory indexing*



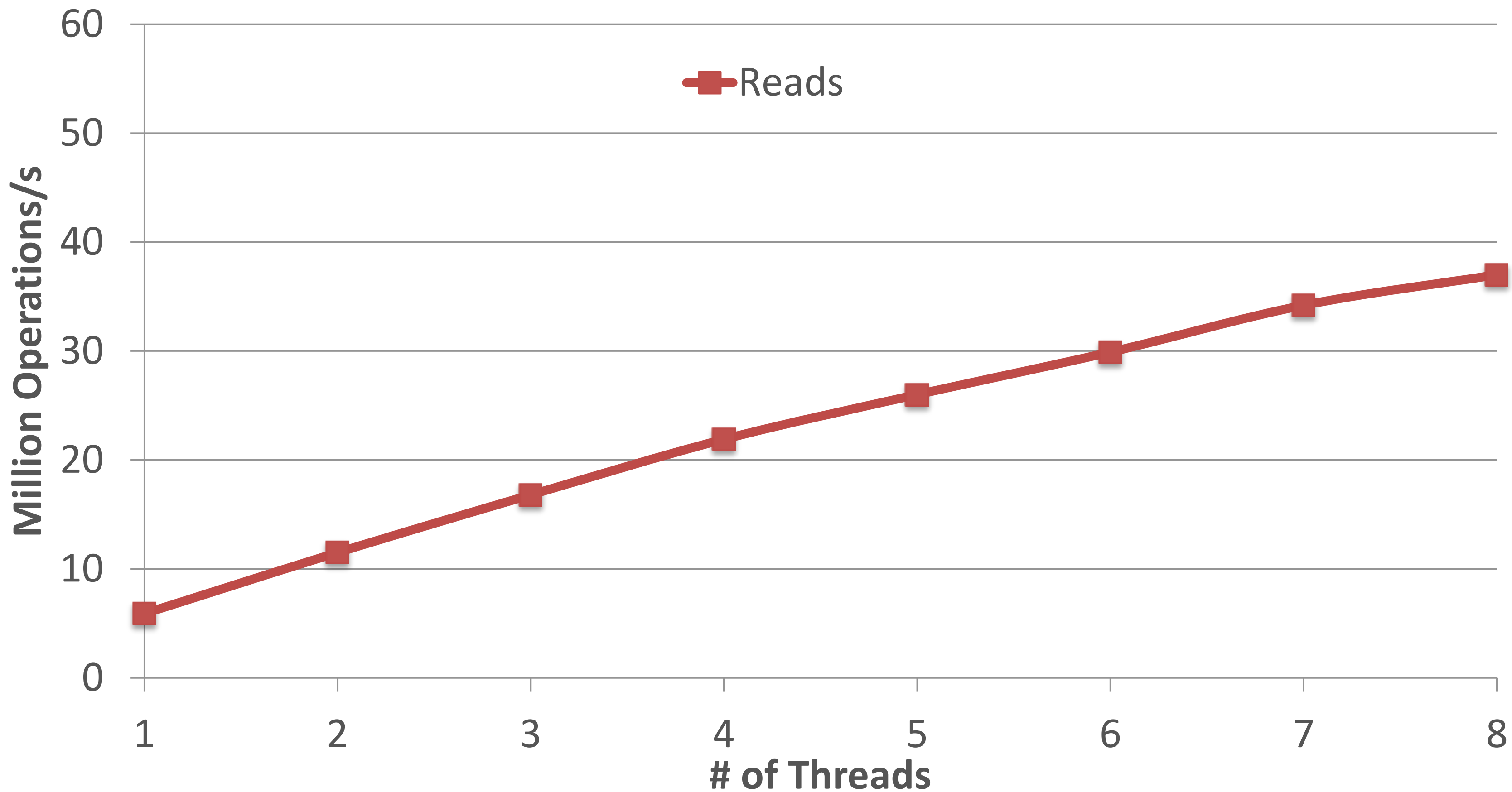
> Parallel DEXTER: Batched Reads



- Reads some nodes (especially top-level nodes) only once
- Prefetching possible

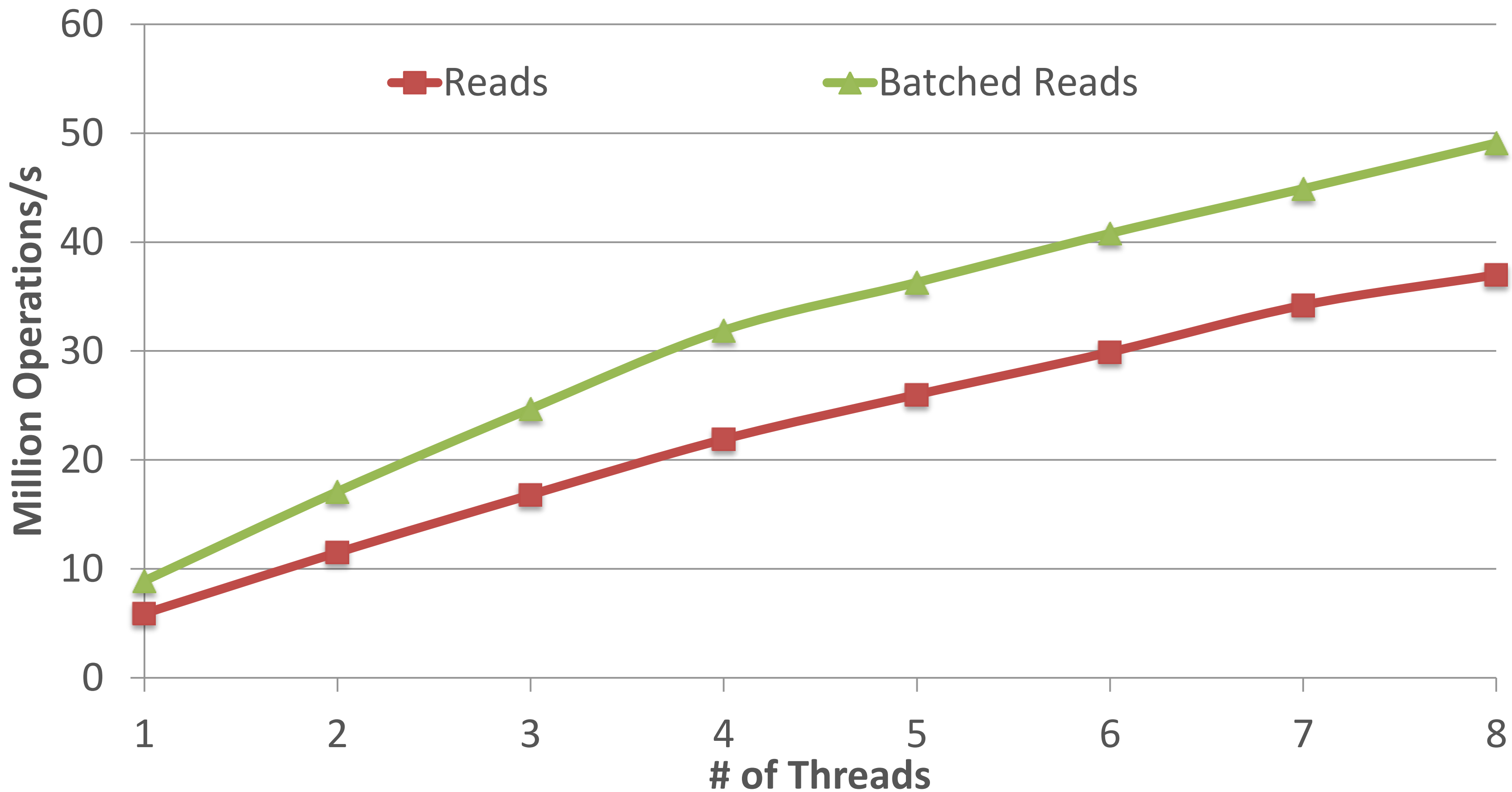
> Parallel DEXTER: Batched Reads: Eval

- *Intel i7-2600 (4 cores @3.4GHz, 2 memory channels, 16GB DDR3 @1333MHz)*
- *20M keys (32bit), 256-batch size*



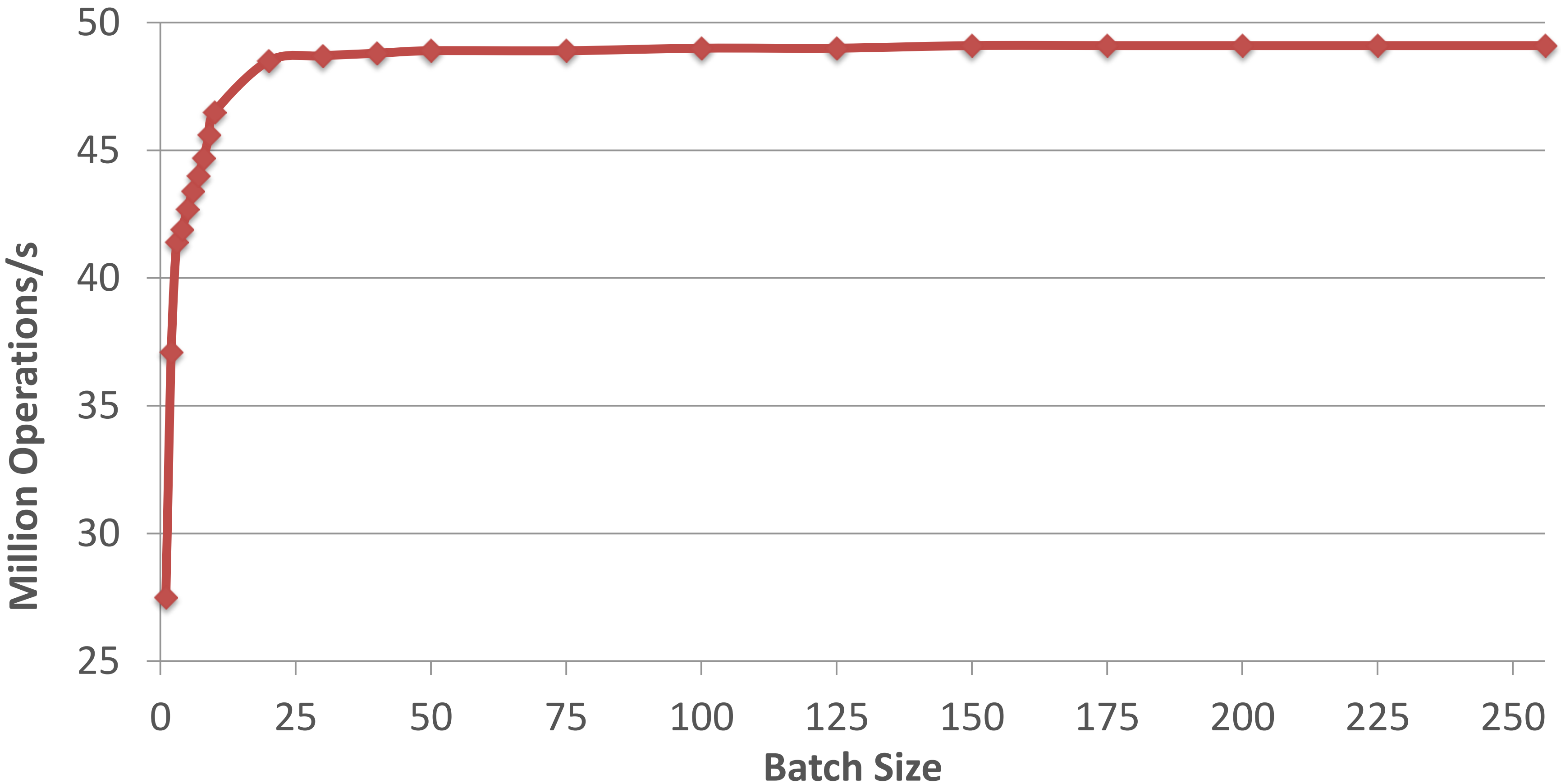
> Parallel DEXTER: Batched Reads: Eval

- *Intel i7-2600 (4 cores @3.4GHz, 2 memory channels, 16GB DDR3 @1333MHz)*
- *20M keys (32bit), 256-batch size*

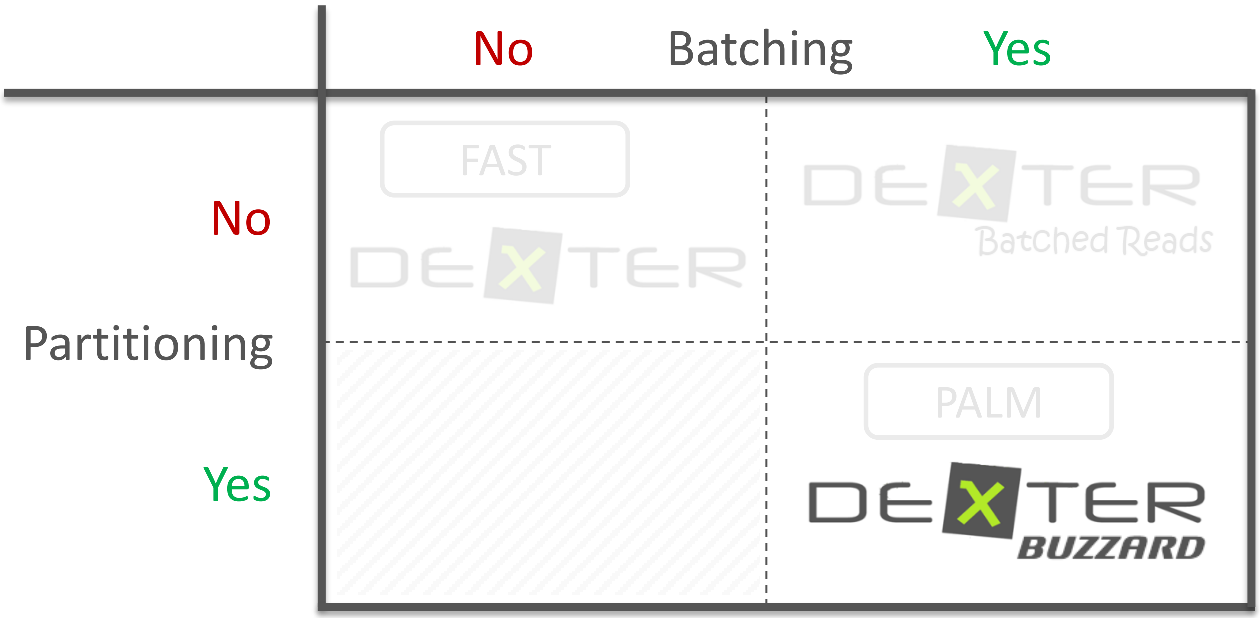


> Parallel DEXTER: Batched Reads: Eval

- *Intel i7-2600 (4 cores @3.4GHz, 2 memory channels, 16GB DDR3 @1333MHz)*
- *20M keys (32bit), dynamic batch size*



- *Two important dimensions for parallel in-memory indexing*

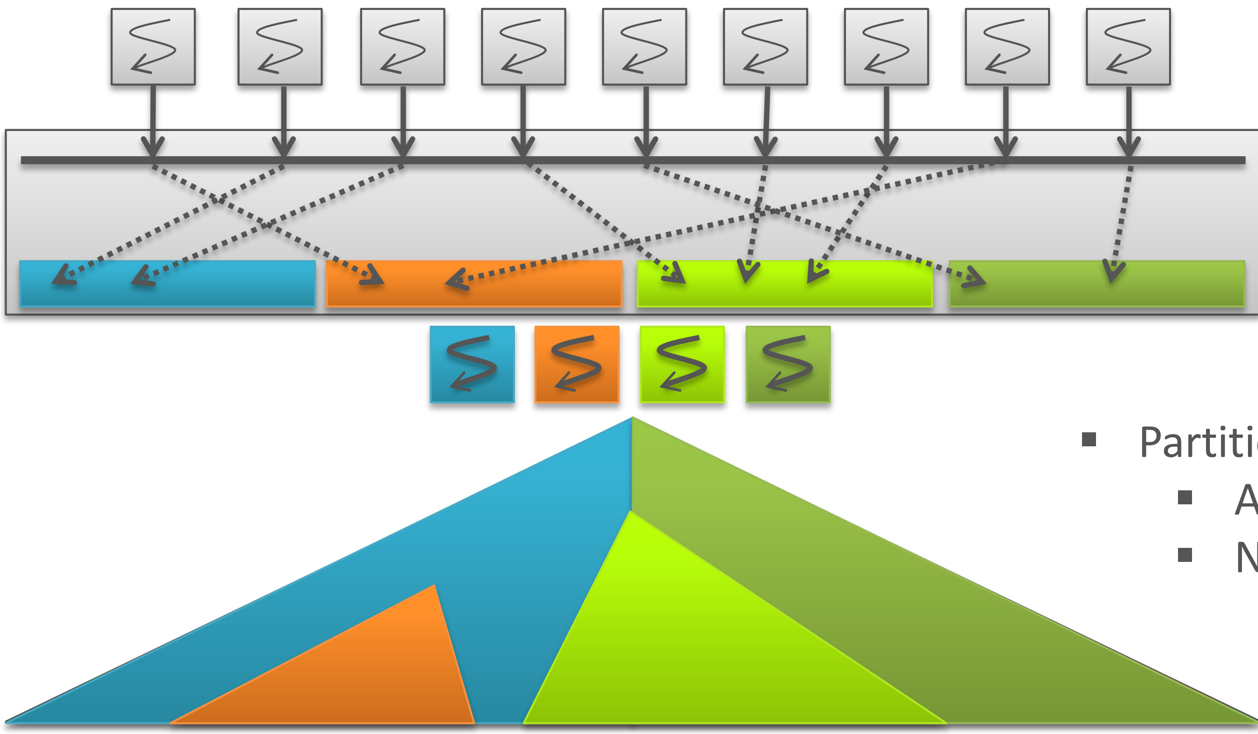


- ✚ Direct Processing
- – Synchronization

- ✚ Independent Data
- – Indirect Processing

- ✚ Low latency

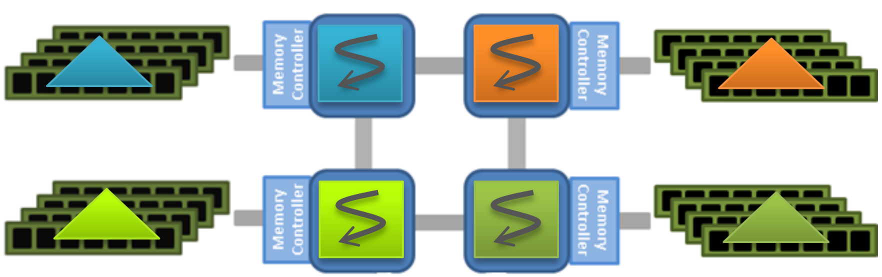
- ✚ High Throughput
- – Increased Latency



Incoming Requests

- Thread Mediation Layer
  - Decoupling

- Partitioned Prefix Tree
  - Adaptive partitioning
  - No synchronization needed

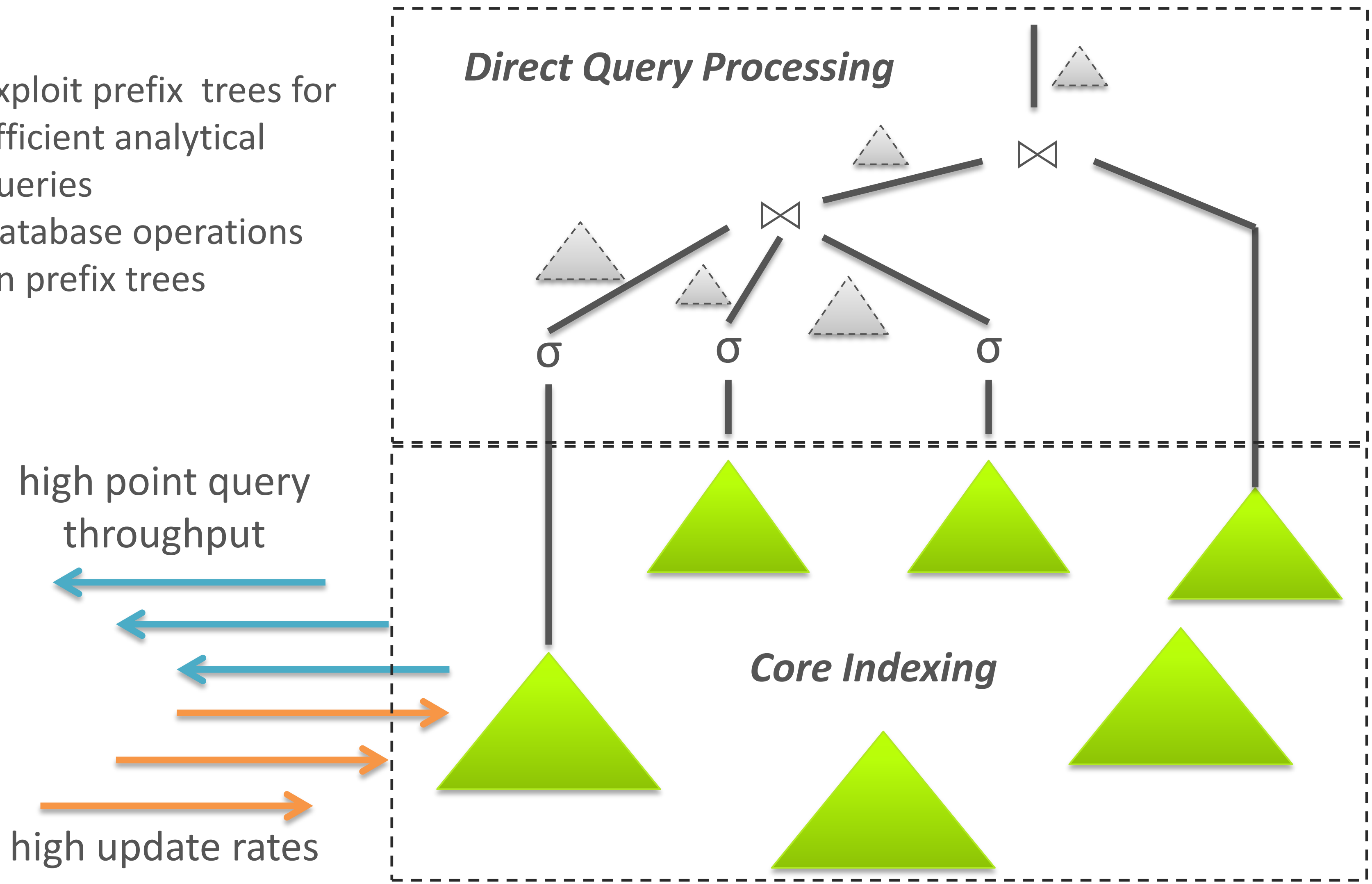


- Memory Mapping
  - Map tree partition to local memory on NUMA systems

- INTRODUCTION AND TRENDS
- DEXTER PROJECT
- CORE INDEXING
- DIRECT QUERY PROCESSING
  - Overview
  - Memory Mirroring
  - Range Select
- CONCLUSIONS

> DEXTER Project Map

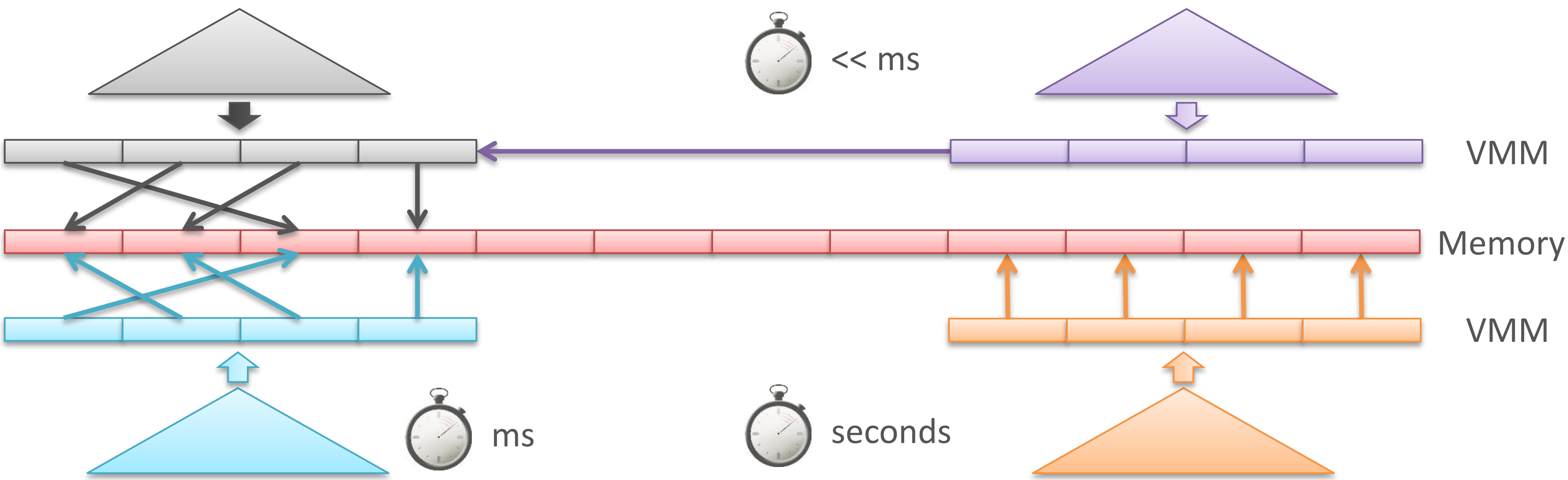
- Exploit prefix trees for efficient analytical queries
- Database operations on prefix trees



> Memory Mirroring

- *Idea: Utilize functionality of the underlying hardware and OS to get fast copies of prefix trees for destructive DB operators*

- *Original Prefix Tree*
- *mmapped shared memory*
  - References old VMM
  - Integrates into address space

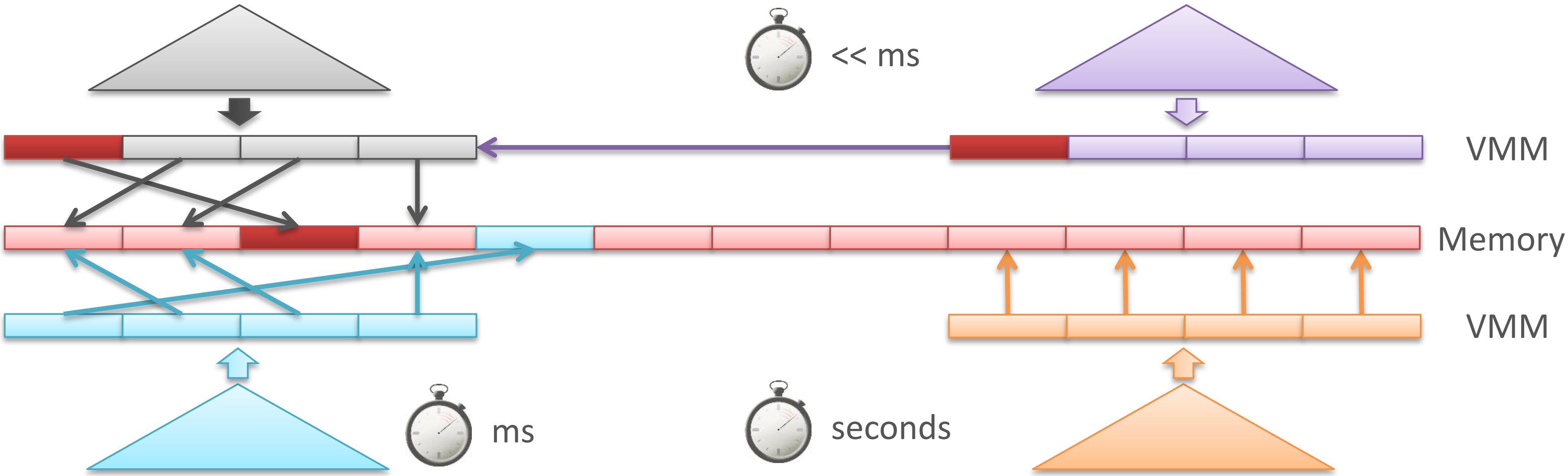


- *fork*
  - Copies VMM
  - Complex to control
- *memcpy*
  - Copies physical memory
  - Fully independent data

> Memory Mirroring

- *Idea: Utilize functionality of the underlying hardware and OS to get fast copies of prefix trees for destructive DB operators*

- *Original Prefix Tree*
- *mmapped shared memory*
  - References old VMM
  - Integrates into address space

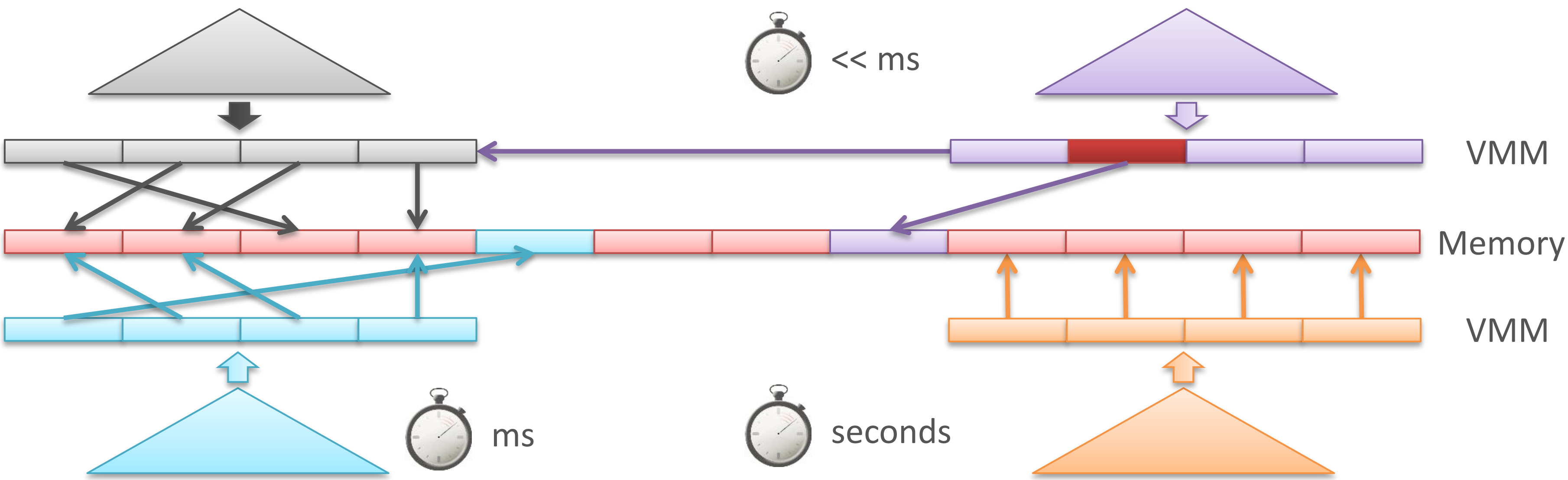


- *fork*
  - Copies VMM
  - Complex to control
- *memcpy*
  - Copies physical memory
  - Fully independent data

> Memory Mirroring

- *Idea: Utilize functionality of the underlying hardware and OS to get fast copies of prefix trees for destructive DB operators*

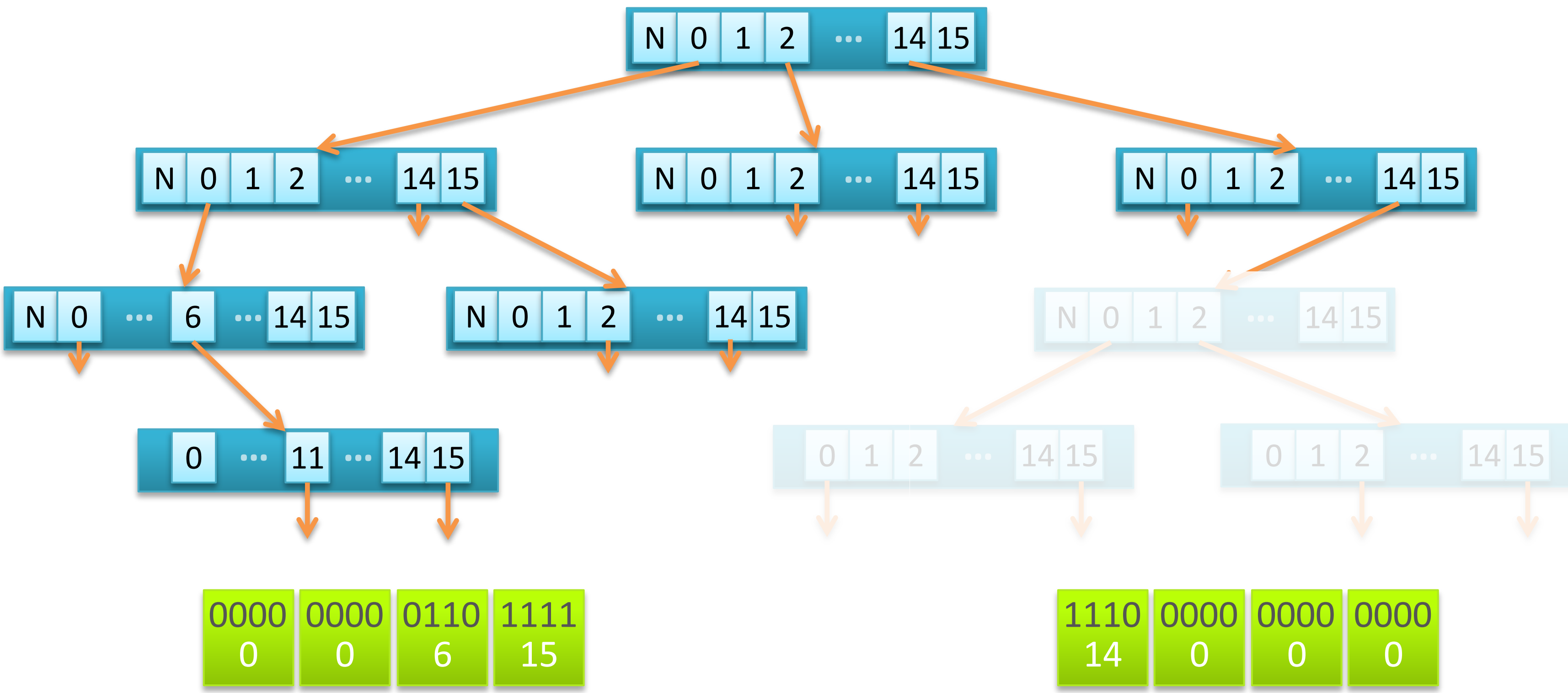
- *Original Prefix Tree*
- *mmapped shared memory*
  - References old VMM
  - Integrates into address space



- *fork*
  - Copies VMM
  - Complex to control
- *memcpy*
  - Copies physical memory
  - Fully independent data

> Range Selection

- Example for a destructive unary DB operator
- Range selection: $\sigma_{111 \le A \le 57344}$



- Prune beyond left key path
- Prune beyond right key path

> Outline

- Introduction and Trends
- DEXTER Project
- Core Indexing
- Direct Query Processing
- Conclusions

> Conclusions

Requirements from Trends

- *Mixed workloads, high update rates, and high responsiveness*
- *Increased parallelism*

Summary

- *Generalized prefix tree with balanced read/write performance*
- *Direct Query Processing on prefix trees*
- *Main concepts: exploit prefix tree structure for pruning*
- *Promising results and plenty of open work*

Conclusions

- *Column stores do not fit the needs of Operational BI and Advanced Analytics*
- *Read/write balanced index structures have a high potential for filling this gap*

Thomas Kissinger

DEXTER

Parallel In-Memory Indexing and Direct Query Processing on Prefix Trees

http://wwwdb.inf.tu-dresden.de/dexter