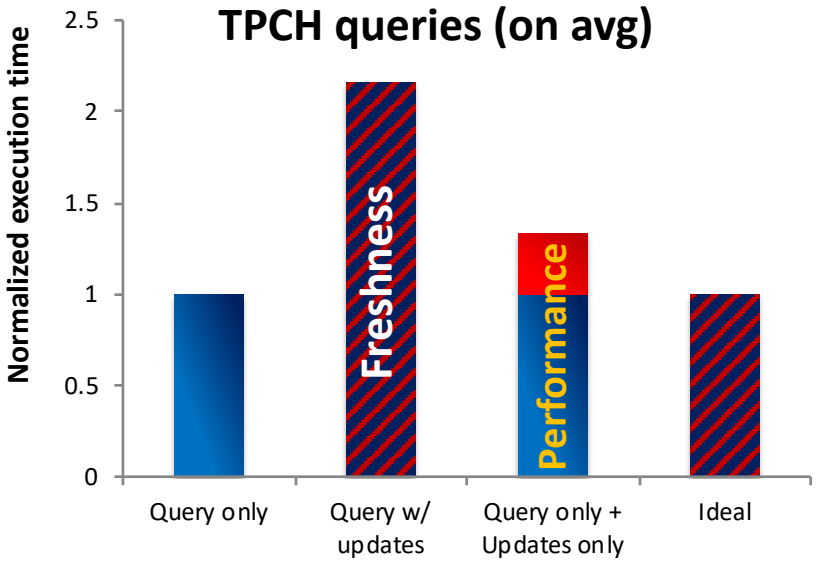
MaSM: Efficient Online Updates in Data Warehouses

Manos Athanassoulis[1] Shimin Chen[2] Anastasia Ailamaki[1]
Phillip Gibbons[2] Radu Stoica[1]
[1]**EPFL** [2]**Intel Labs**

Freshness **vs** Performance

- Data warehouse workload
  - Read-only queries (scans)
  - Scattered updates
  - Difficult to combine *efficiently*

- Traditionally two choices
  - **Freshness:** in-place updates
  - **Performance:** batch updates

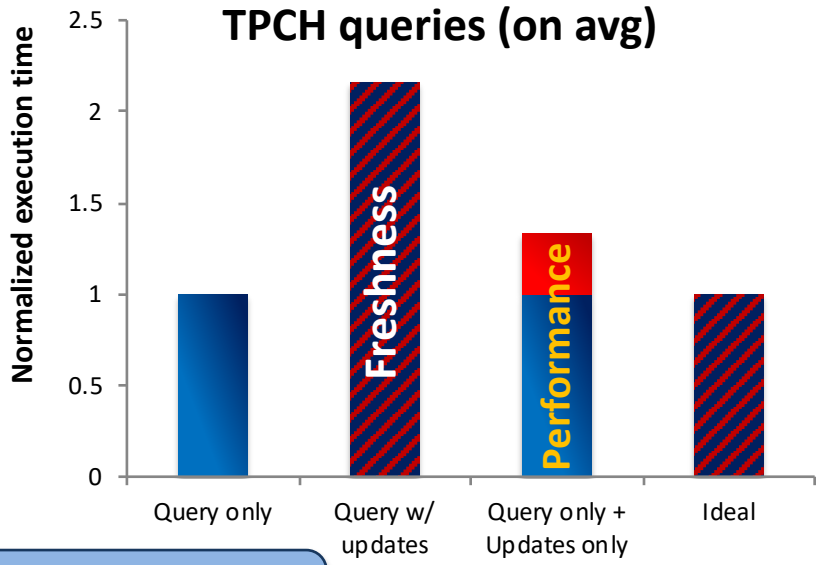


- Ideally, *zero* overhead

Freshness **vs** Performance

- Data warehouse workload
  - Read-only queries (scans)
  - Scattered updates
  - Difficult to combine *efficiently*

- Traditionally two choices
  - **Freshness:** in-place updates
  - **Performance:** batch updates



- Ideally, *zero overhead*

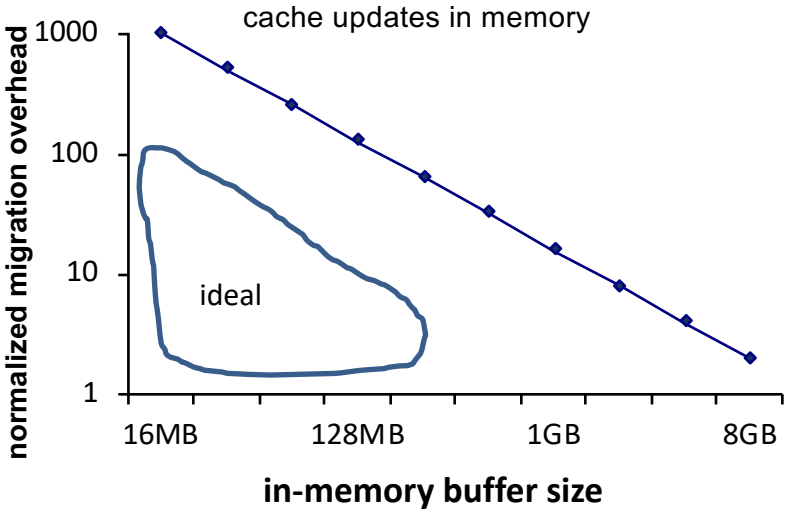
Is zero overhead possible?

Freshness **AND** Performance

In Memory Buffered Updates

[Stonebraker et al .'05] [Heman et al.'10]

- ➢ Apply them online
- ➢ Apply them as differential updates
- x Large memory overhead
- x Trade-off migration overhead for memory footprint



To sum up

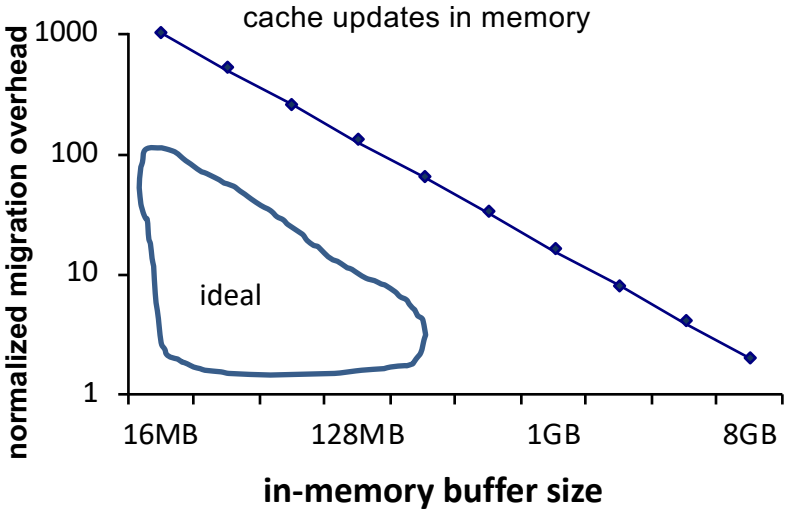
Update Approach	Freshness	Performance	↓ mem overhead
Batched	X	☺	☺
In place	☺	X	☺
In-memory differential	☺	☺	X

Freshness **AND** Performance

In Memory Buffered Updates

[Stonebraker et al .'05] [Heman et al.'10]

- ➢ Apply them online
- ➢ Apply them as differential updates
- x Large memory overhead
- x Trade-off migration overhead for memory footprint

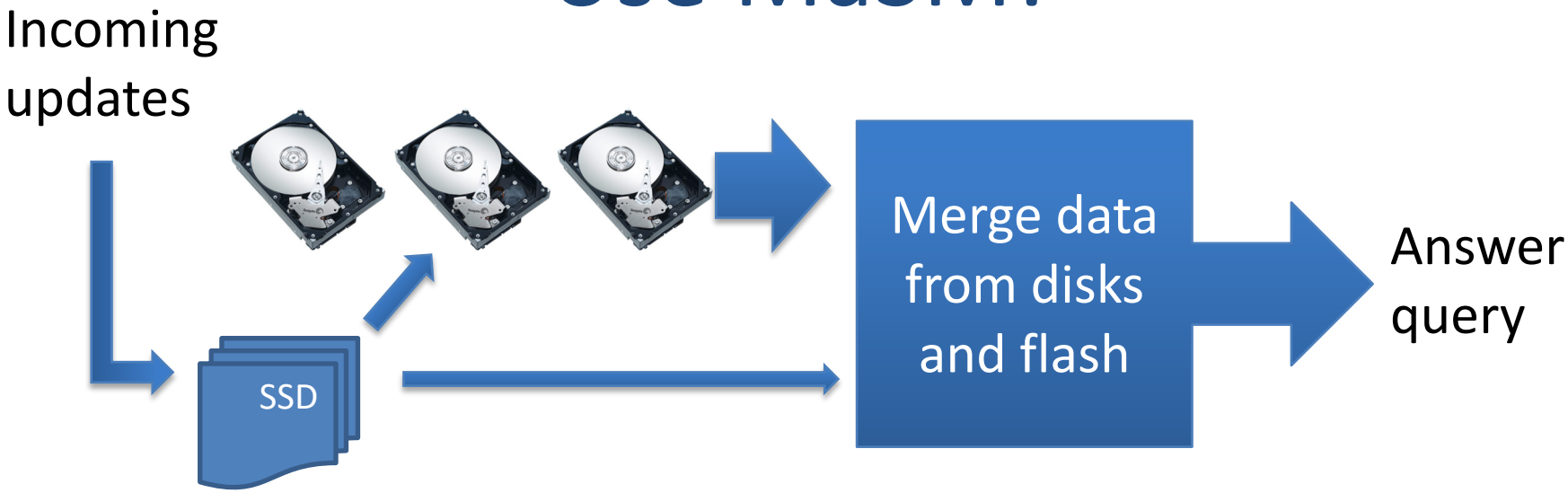


To sum up

Update Approach	Freshness	Performance	↓ mem overhead
Batched	X	☺	☺
In place	☺	X	☺
In-memory differential	☺	☺	X

Can we have the cake and eat it too?

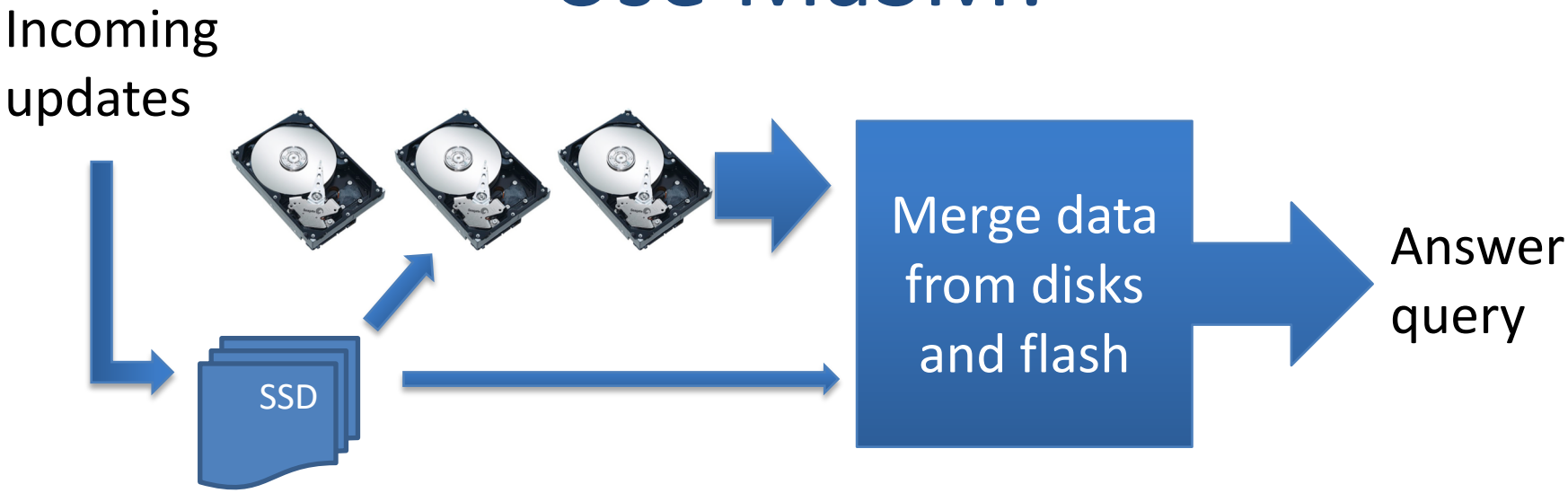
Use MaSM!



- Buffer updates on Flash instead of memory
  - Flash has *larger capacity* and *smaller price*
- **But:** Flash friendly design is important
  - Avoid random writes
  - Limit total writes
    - e.g. Log-Structure Merge Tree incurs a high number writes per update

[O' Neil et al.'96]

Use MaSM!



- Buffer updates on Flash instead of memory
  - Flash has *larger capacity* and *smaller price*
- **But:** Flash friendly design is important
  - Avoid random
  - Limit total writes

Let's think again!

  - e.g. Log-Structure Merge Tree incurs a high number writes per update

[O' Neil et al.'96]

MaSM core idea

Updates (U)

K	Value	Type
1	V1’	Mod
5	V5’’	Mod
9	N/A	Del
19	V19’	Mod
125	V125	Ins
5	V5’’	Mod

Current data

- ✓ Outer join: $D \bowtie U$
- ✓ Keep latest update only

Efficient execution

- ✓ Discard duplicates
- ✓ Re-use information for future queries

Sort-Merge Join

- ✓ Intuitively does both

Data (D)

Key	Value
1	V1
2	V2
3	V3
4	V4
5	V5
6	V6
7	V7
8	V8
9	V9

MaSM core idea

Updates (U)

K	Value	Type
1	V1’	Mod
5	V5’’	Mod
9	N/A	Del
19	V19’	Mod
125	V125	Ins
5	V5’’	Mod

Data (D)

Key	Value
1	V1
2	V2
3	V3
4	V4
5	V5
6	V6
7	V7
8	V8
9	V9

Current data

- ✓ Outer join: $D \bowtie U$
- ✓ Keep latest update only

Efficient execution

- ✓ Discard duplicates
- ✓ Re-use information for future queries

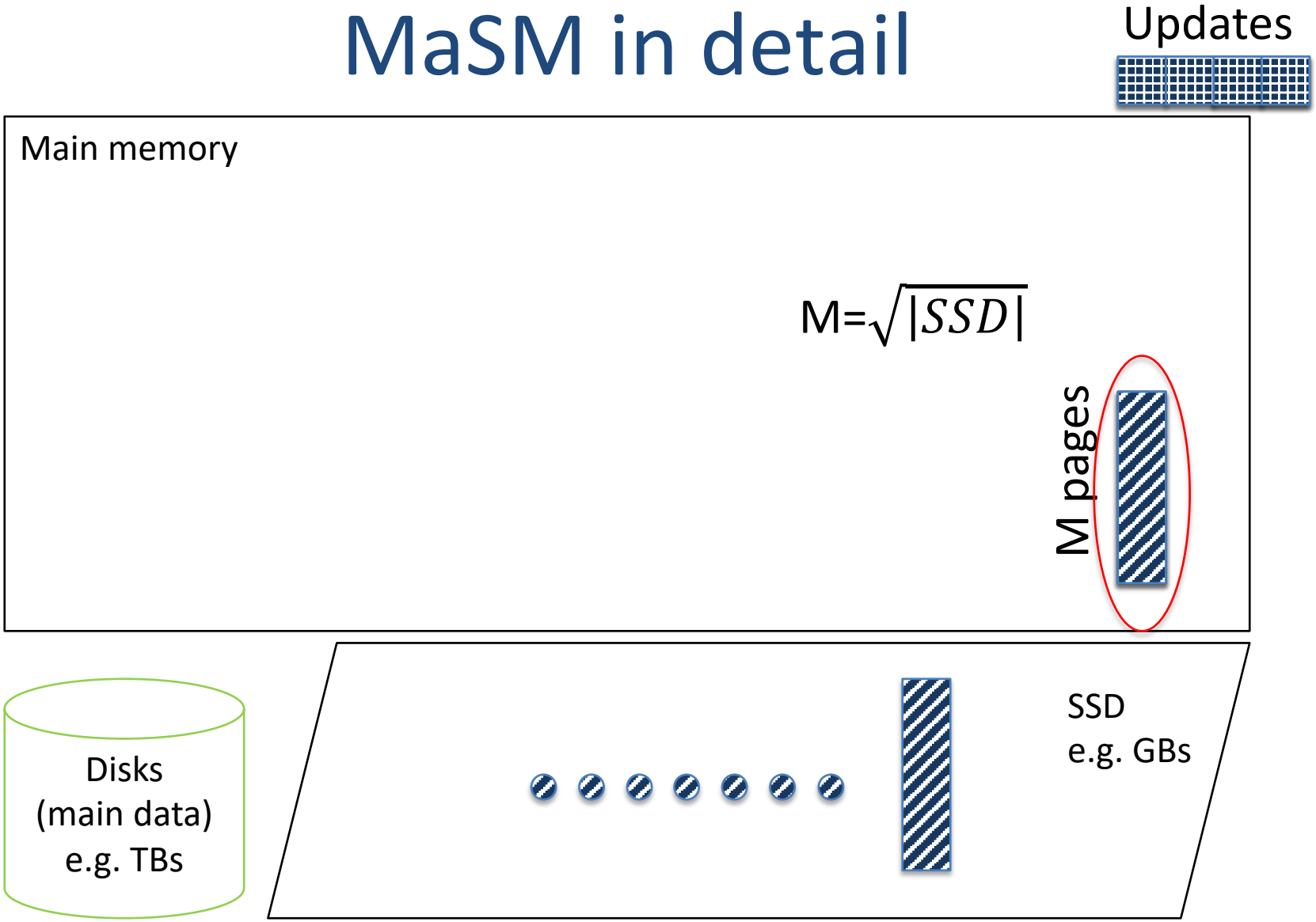
Sort-Merge Join

MaSM merges data with updates using sort-merge join and materializing sorted runs

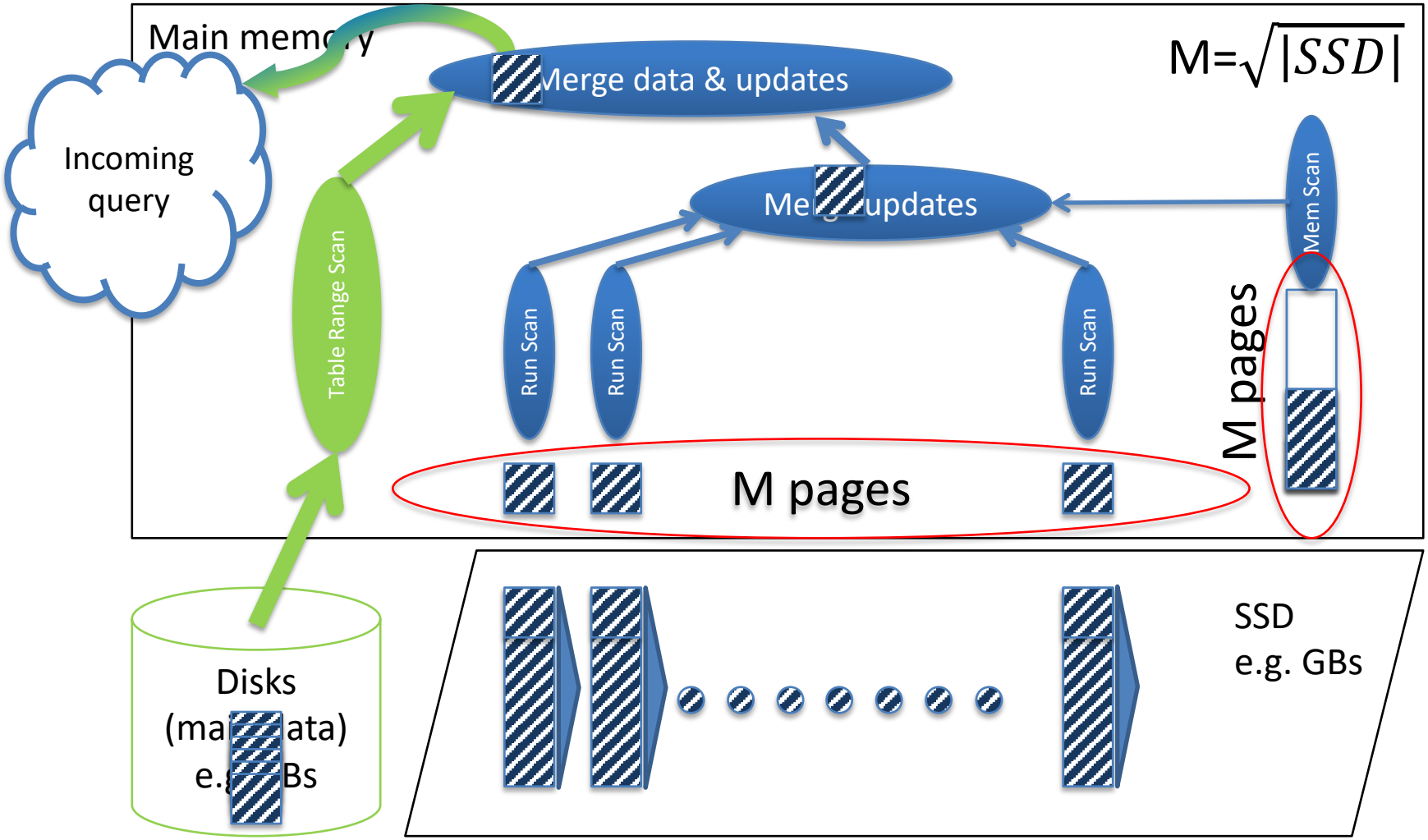
Outline

- Introduction
  - Prior work: Differential Updates
  - MaSM sneak peak
- **MaSM architecture**
- Evaluation
  - Query response time
  - Sustained update rate
- Conclusions

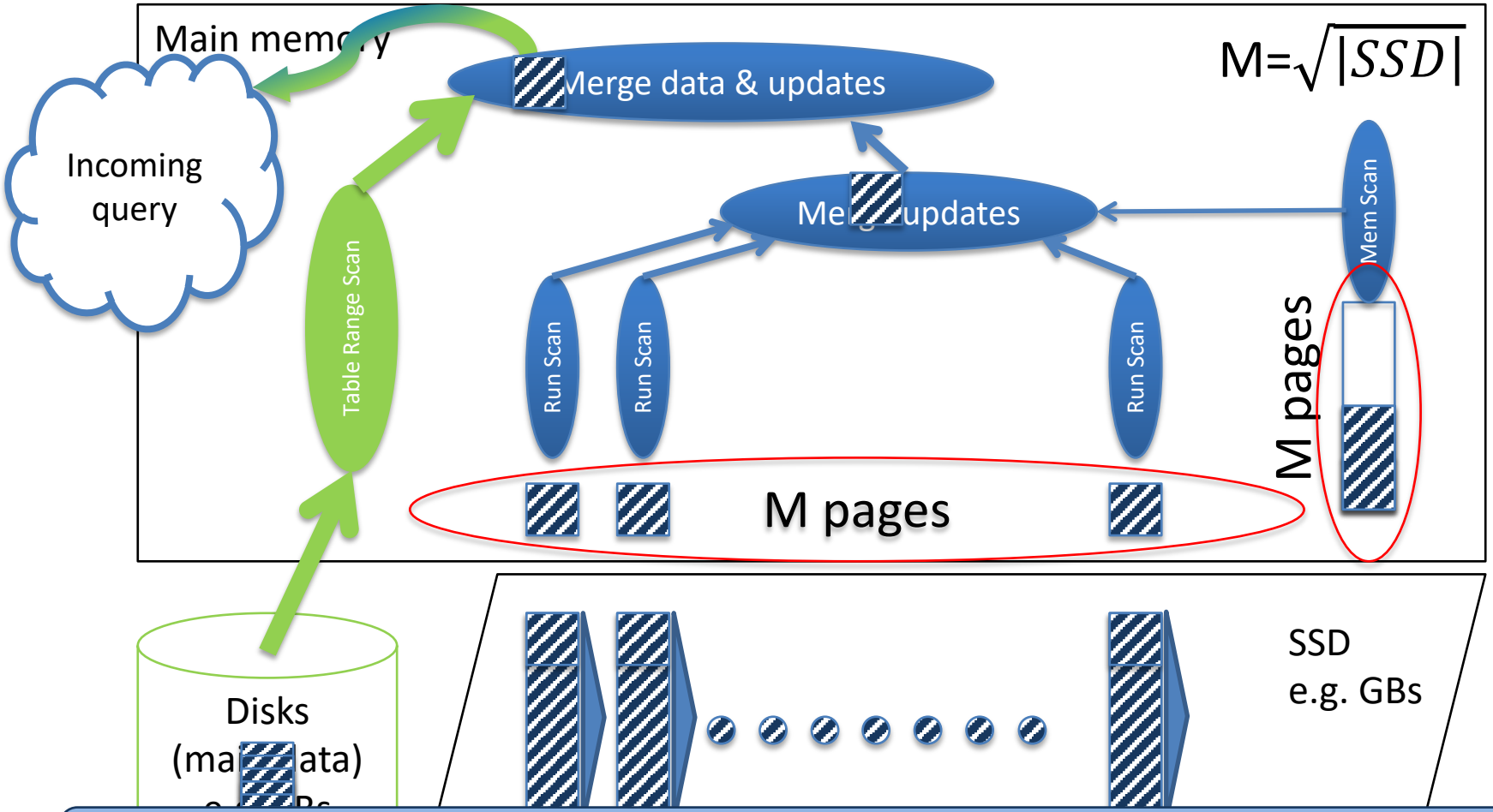
MaSM in detail



MaSM in detail

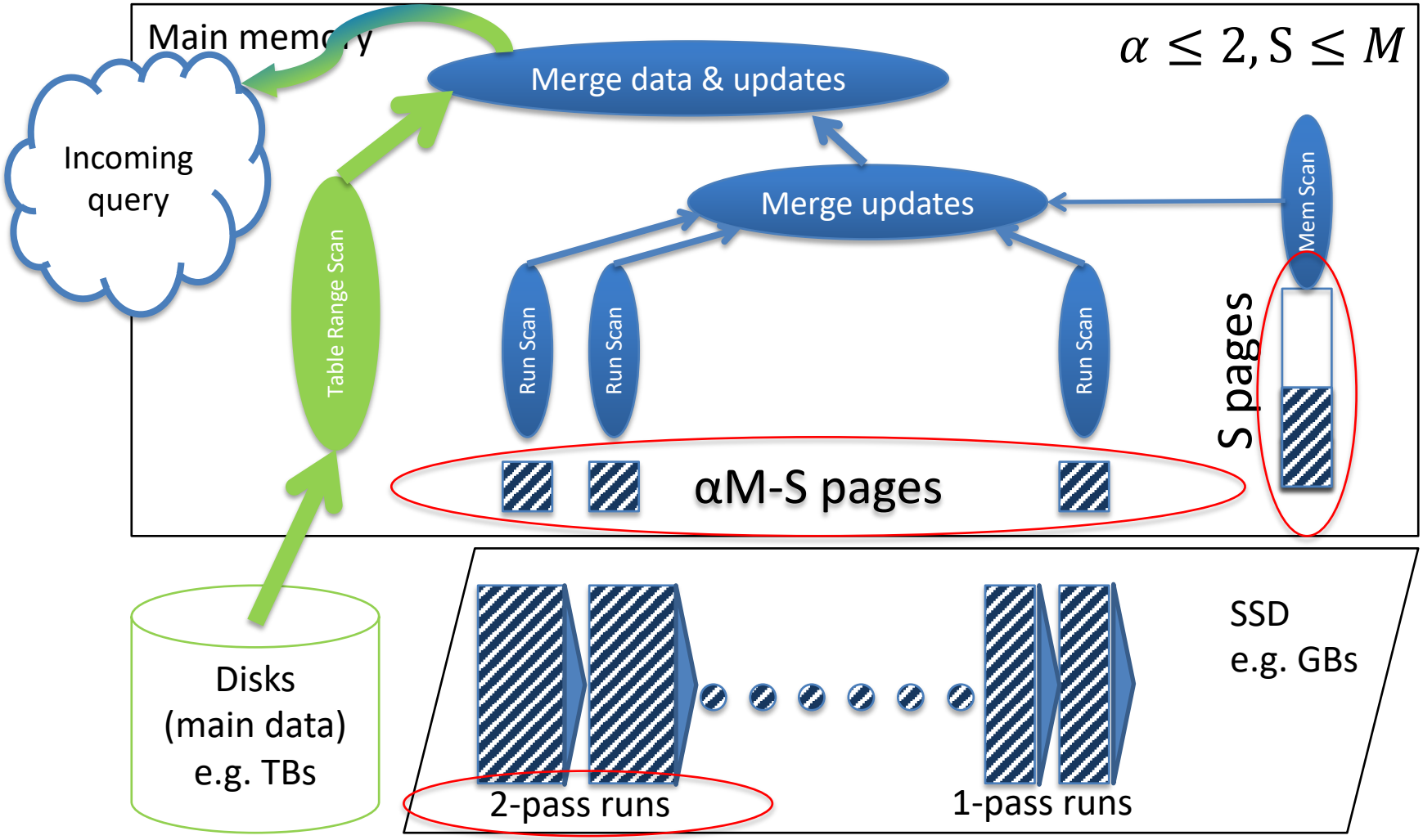


MaSM in detail

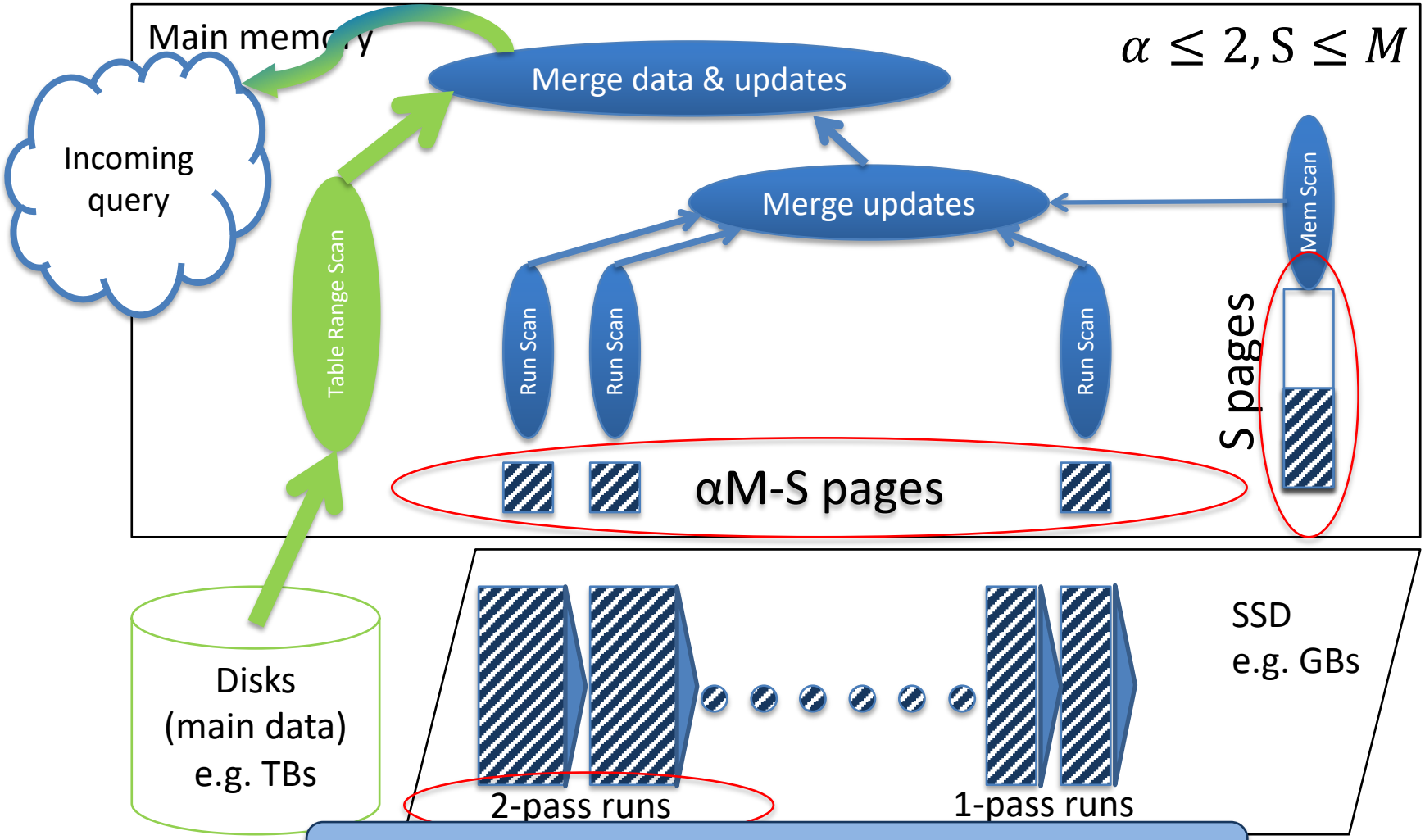


Merge pages from HDD, SSD and RAM with negligible overhead!

Reducing MaSM memory

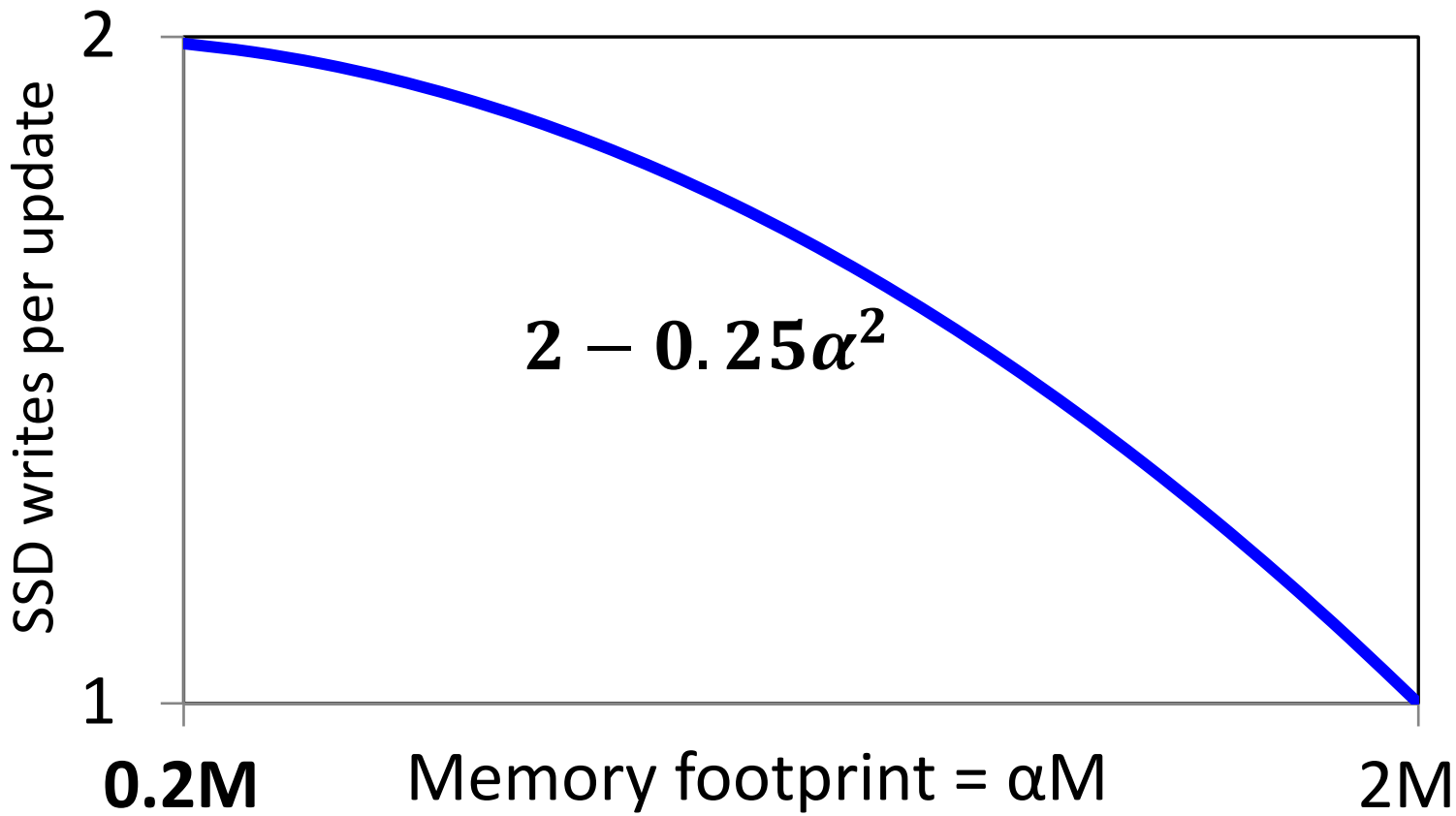


Reducing MaSM memory



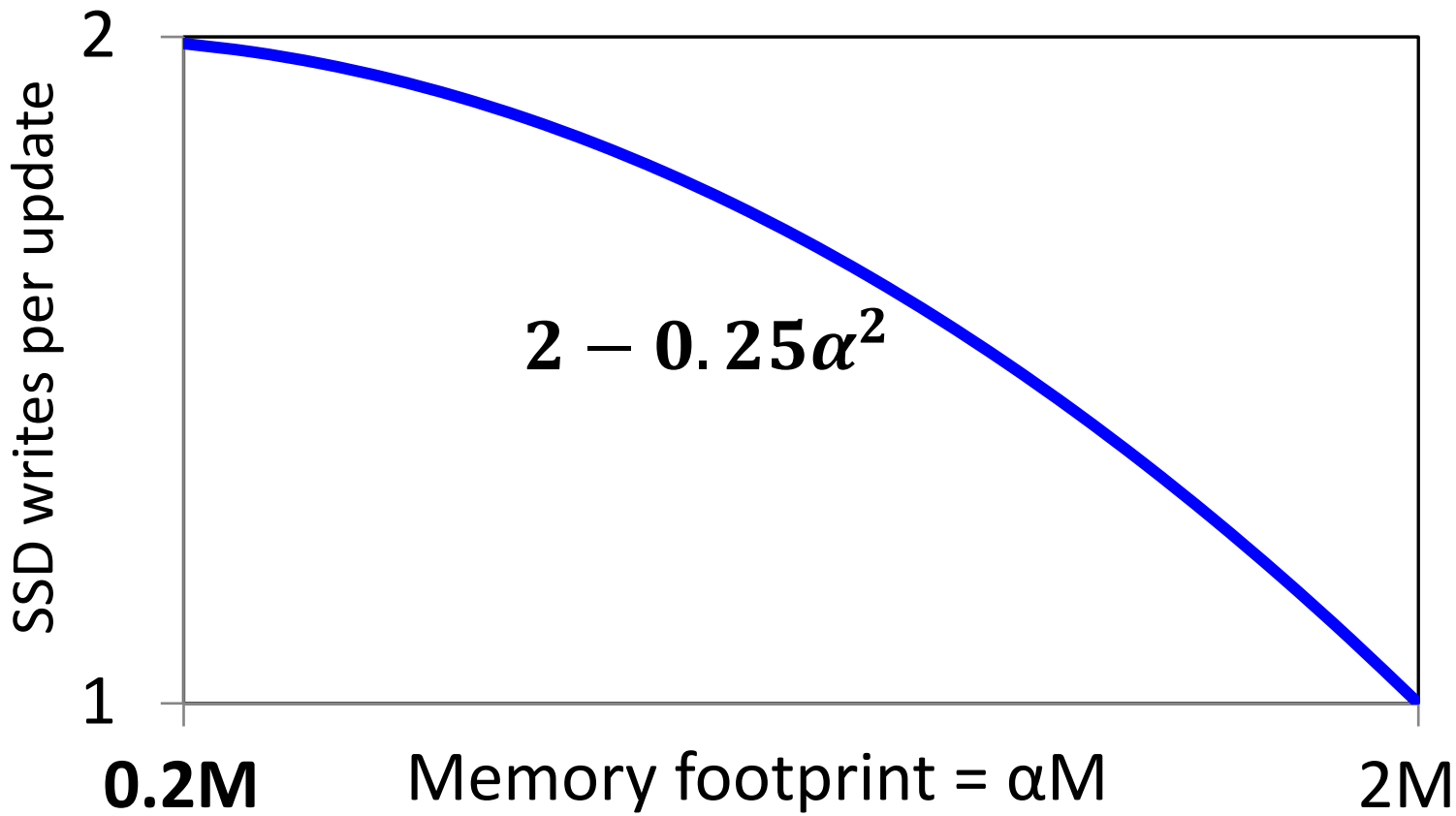
Trade-off extra writes for memory size.

Impact of α on SSD *wear*



$M=\sqrt{|SSD|}$ $f(M) \leq \alpha \leq 2$ e.g., M=1000, $0.2 \leq \alpha \leq 2$

Impact of α on SSD *wear*



$M=\sqrt{|SSD|}$ $f(M) < \alpha < 2$ e.g. M=1000, $0.2 \leq \alpha \leq 2$

10x smaller memory for 2x more writes!

Outline

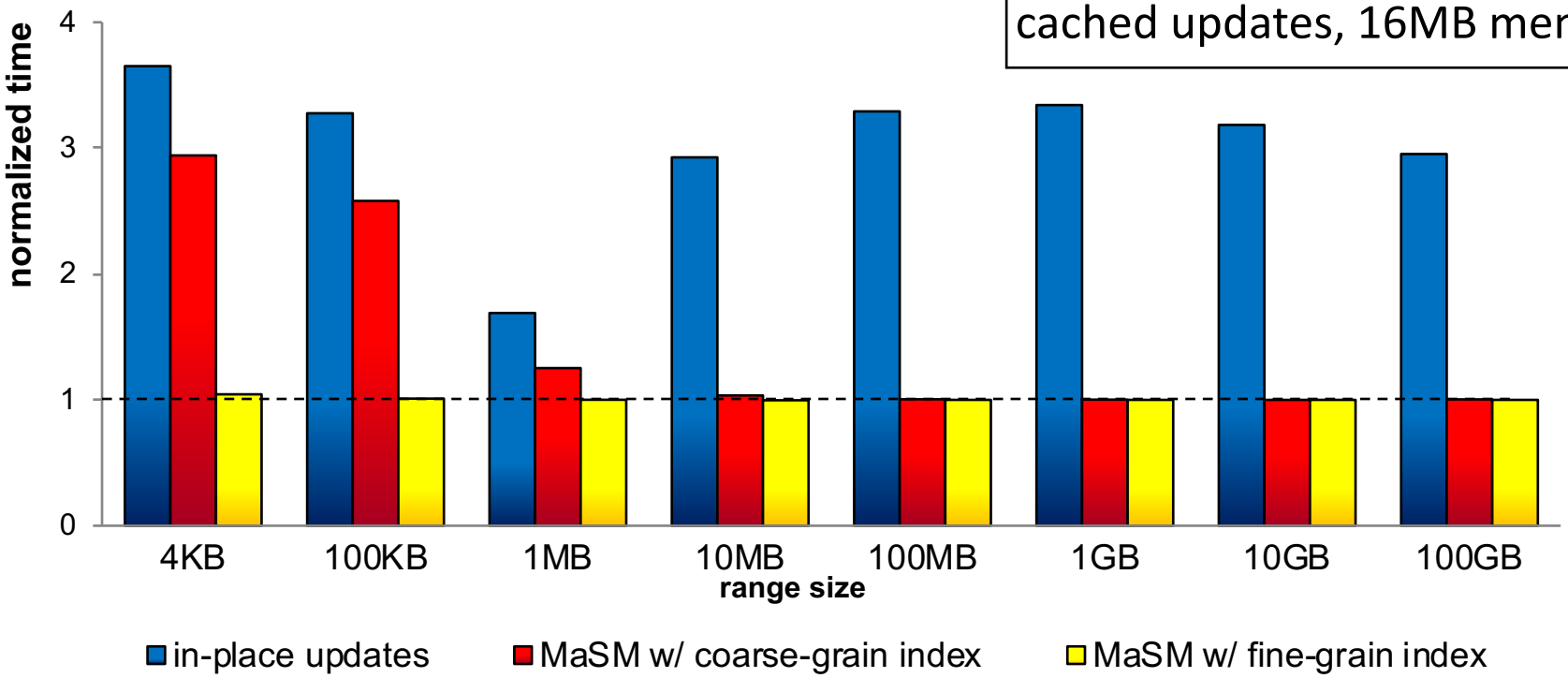
- Introduction
  - Prior work: Differential Updates
  - MaSM sneak peak
- MaSM architecture
- **Evaluation**
  - Query response time
  - Sustained update rate
- Conclusions

Experimental setup

- Dell Precision 690 Workstation
  - Intel Xeon Quad (2.33MHz, 8MB L2), 4GB DRAM, Ubuntu Linux, kernel 2.6.24
- Dedicated SATA disk for main data
  - 7200rpm Seagate Barracuda, 77MB/s sequential bandwidth
- Intel X25-E SSD for caching updates
  - 250 MB/s sequential read, 170MB/s sequential write bandwidth; 35,000 4KB-sized random reads/second
- Prototype row store:
  - Implemented in-place updates, indexed updates, MaSM

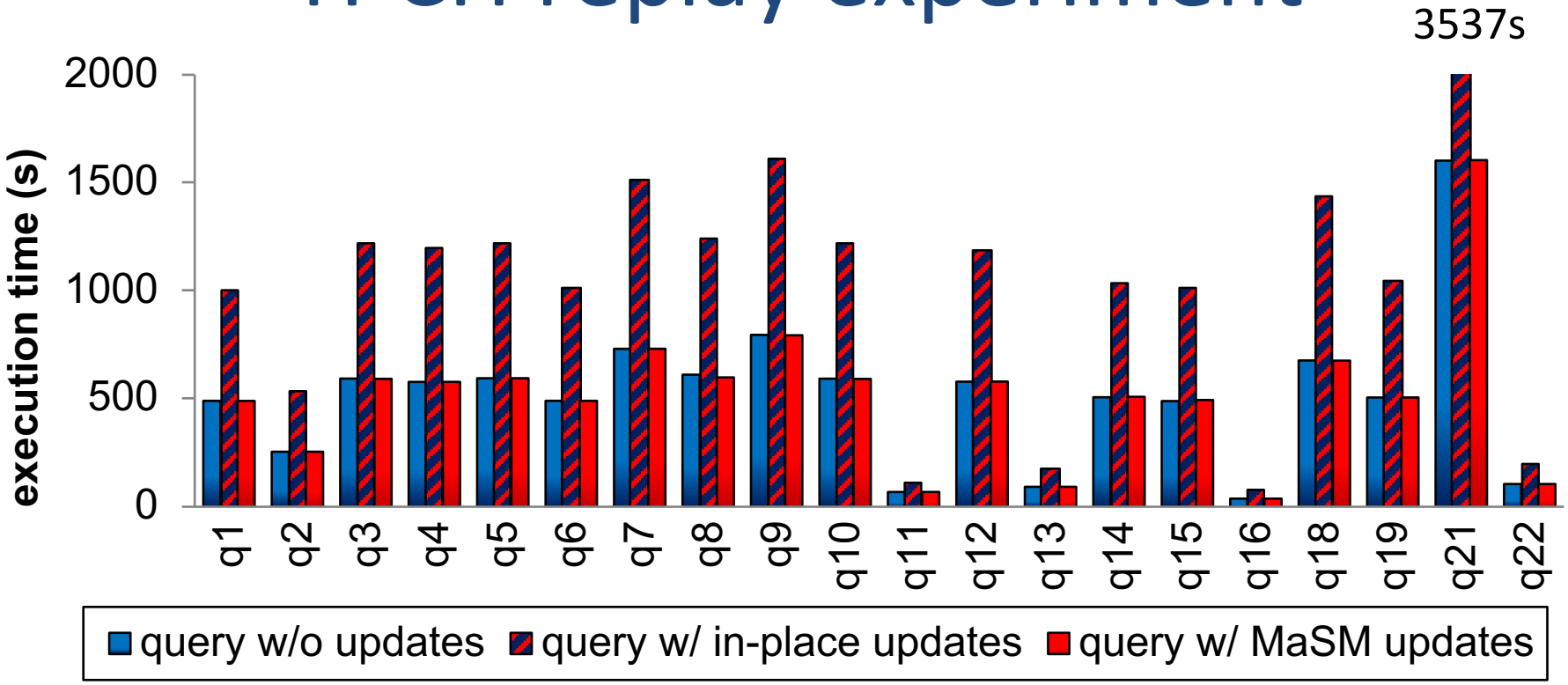
Query performance on synthetic data

100GB main data, 4GB flash for cached updates, 16MB memory



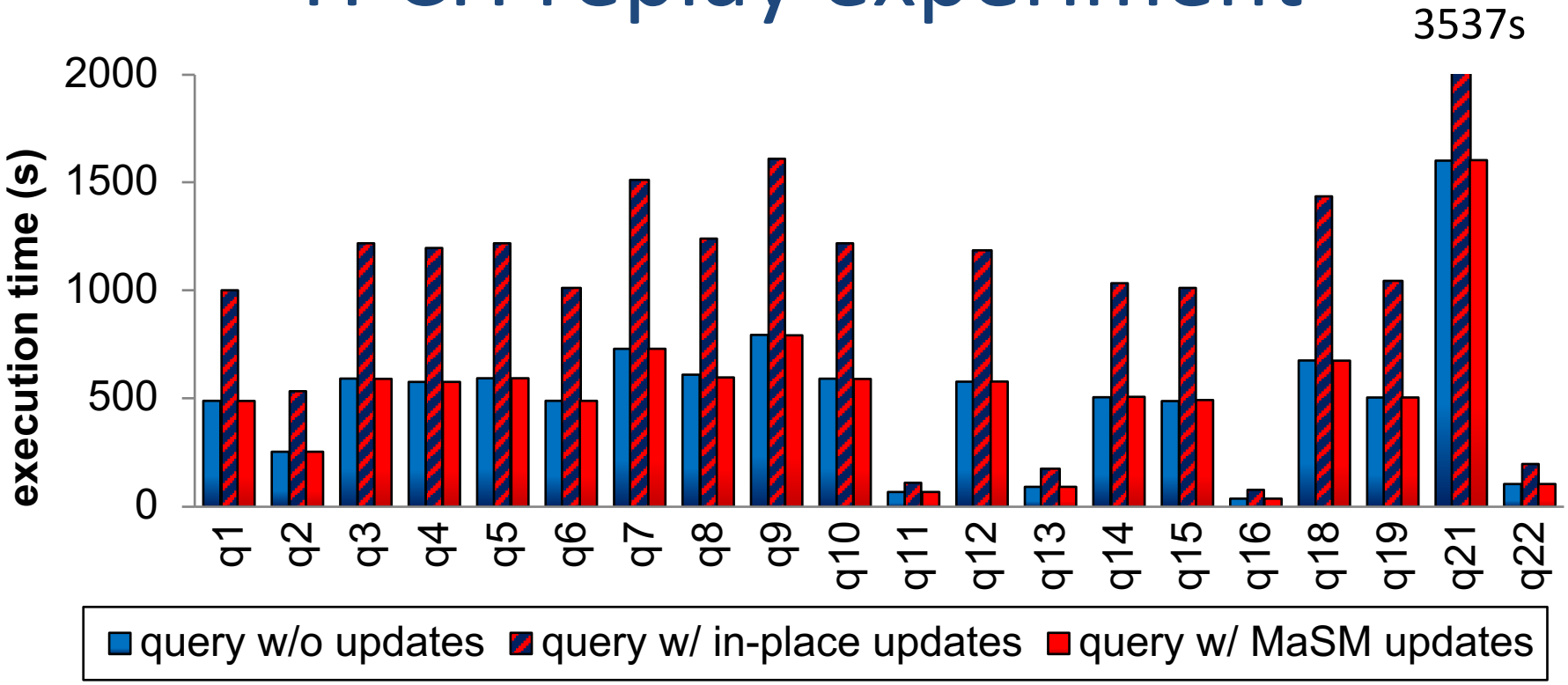
- **MaSM has negligible impact on 10MB or larger scans**
- **MaSM with fine-grain index incurs 4% overhead for 4KB ranges (modeling point queries)**

TPCH replay experiment



- **Replay TPCH disk traces recorded from commercial row store; random online updates**

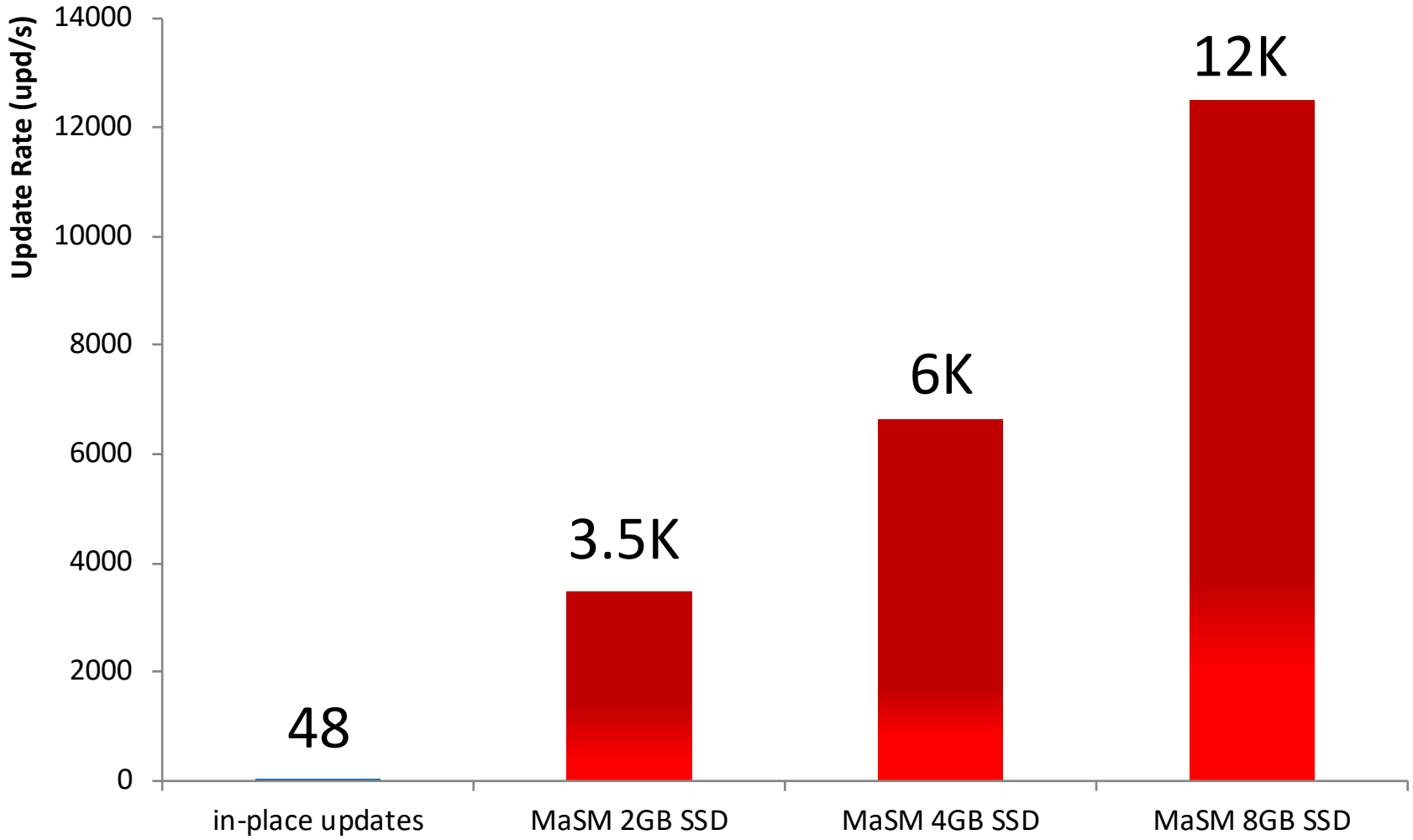
TPCH replay experiment



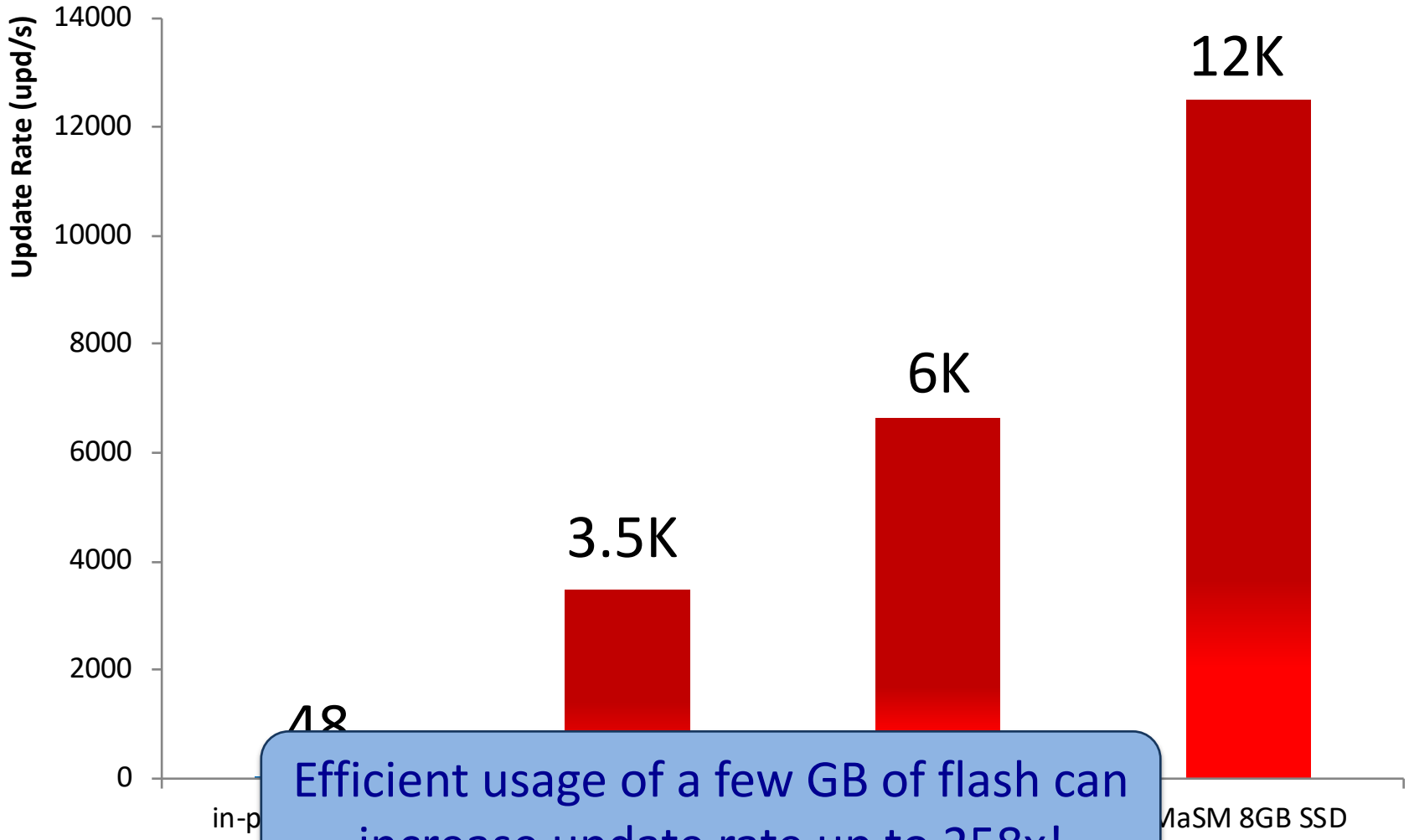
- **Replay TPCH disk traces recorded from commercial row store; random online updates**

Queries with MaSM see less than 1% overhead!

Update performance



Update performance

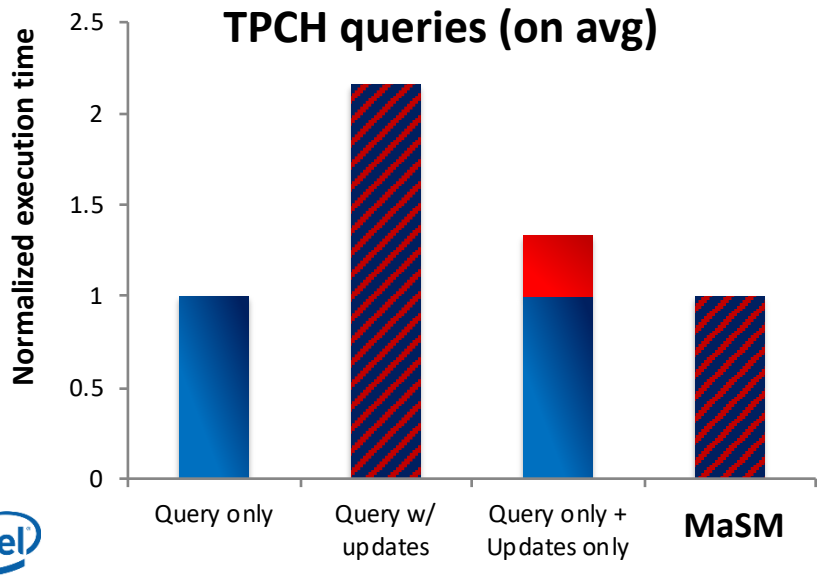


Efficient usage of a few GB of flash can increase update rate up to 258x!

To sum up

MaSM enables **on-line updates** in DW

- Negligible query overhead (less than 1% for TPCH)
- Supports a **high update rate (up to 12k)**
- Tunable *memory footprint* vs *SSD wear*
- Low migration cost (one-time 2.2x)
- SSD-friendly behavior
  - Limited number of writes per updates
  - No random writes on SSD
- Easy DBMS integration
- Ensure ACID properties



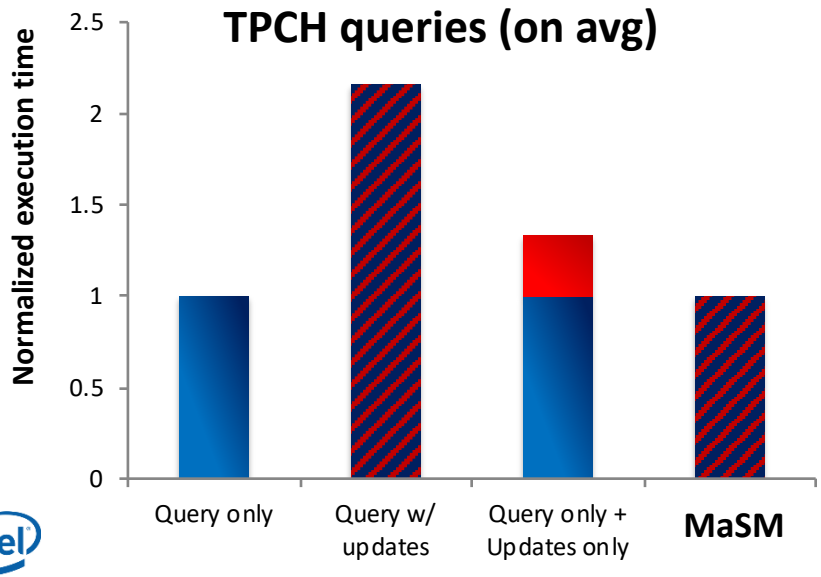
To sum up

MaSM enables **on-line updates** in DW

- Negligible query overhead (less than 1% for TPCH)

Update Approach	Freshness	Performance	↓ mem overhead
Batched	X	☺	☺
In place	☺	X	☺
In-memory differential	☺	☺	X
MaSM and SSD	☺	☺	☺

  - Limited number of writes per updates
  - No random writes on SSD
- Easy DBMS integration
- Ensure ACID properties



Thank you!