Microsoft Research

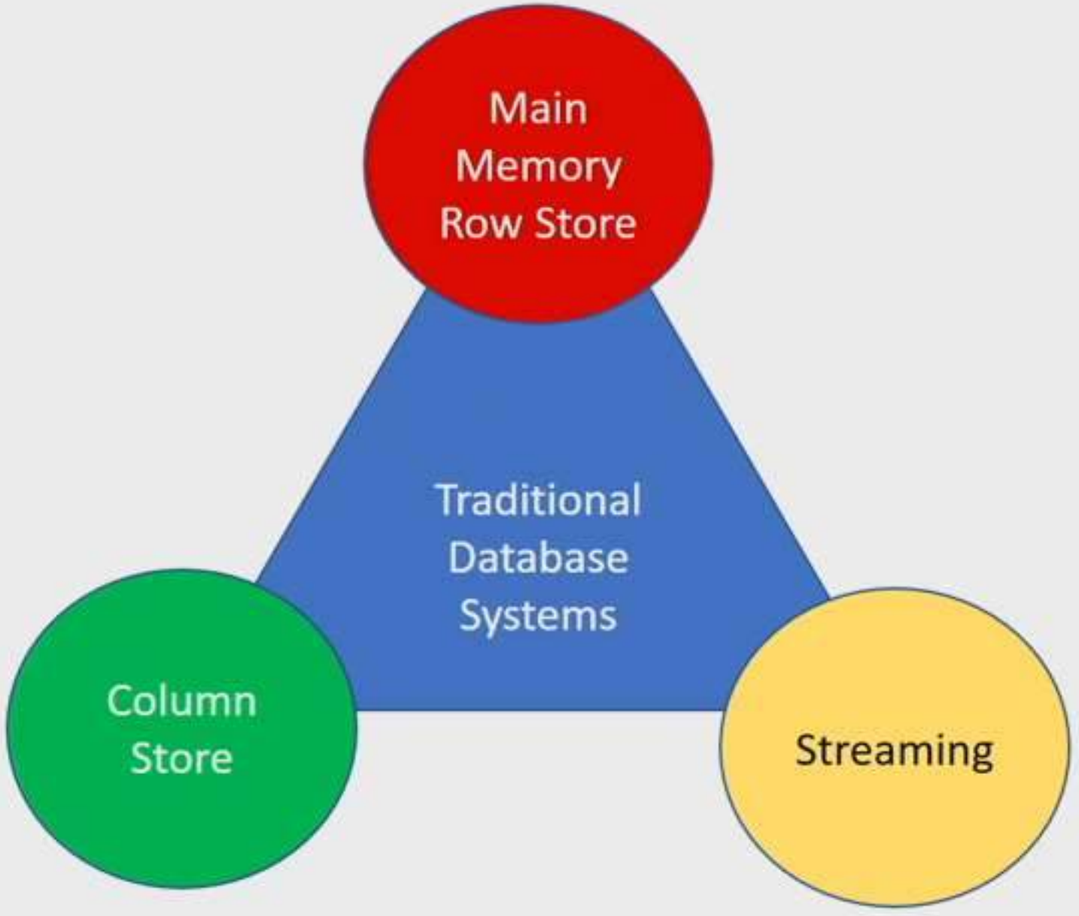
Each year Microsoft Research hosts hundreds of influential speakers from around the world including leading scientists, renowned experts in technology, book authors, and leading academics, and makes videos of these lectures freely available.

Cost/Performance in Modern Data Stores

How Data Caching Systems Succeed

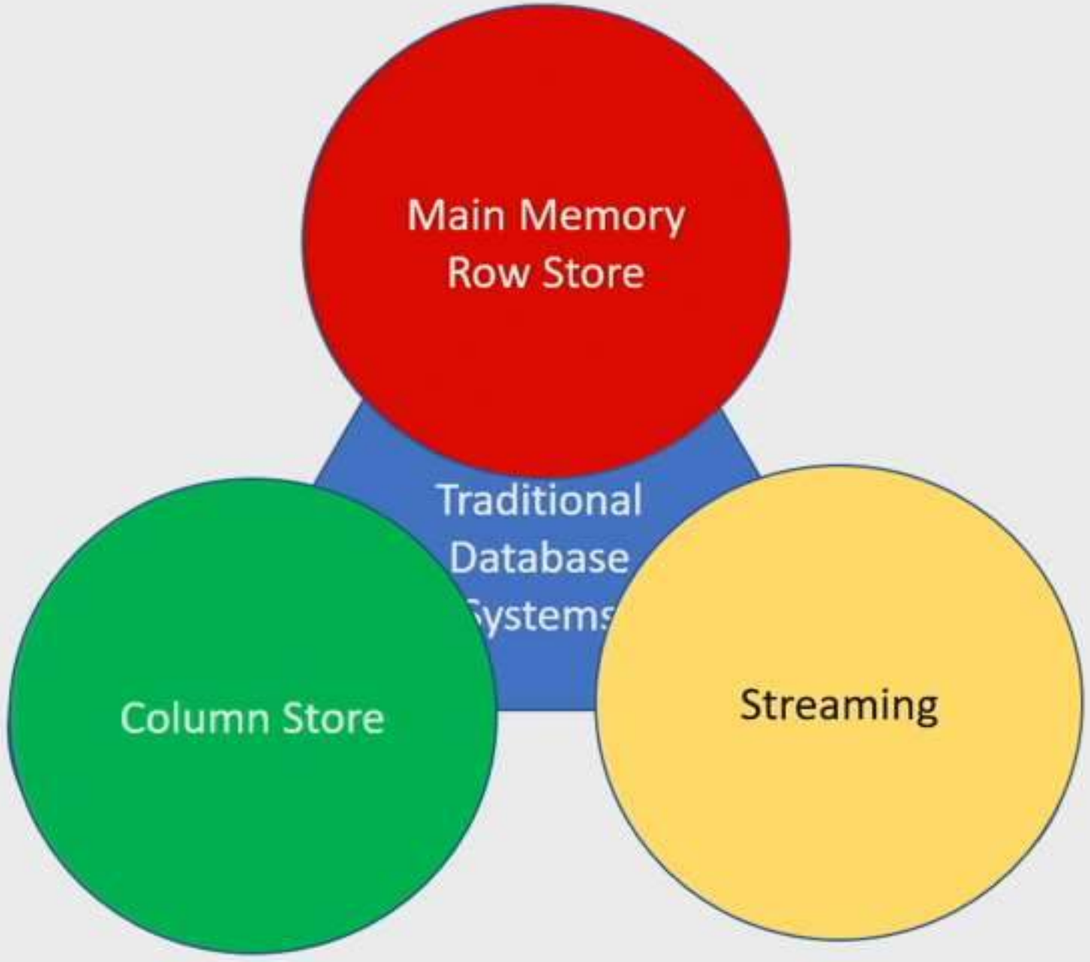
David Lomet
Microsoft Research
Redmond, WA 98052

Traditional Databases are Challenged

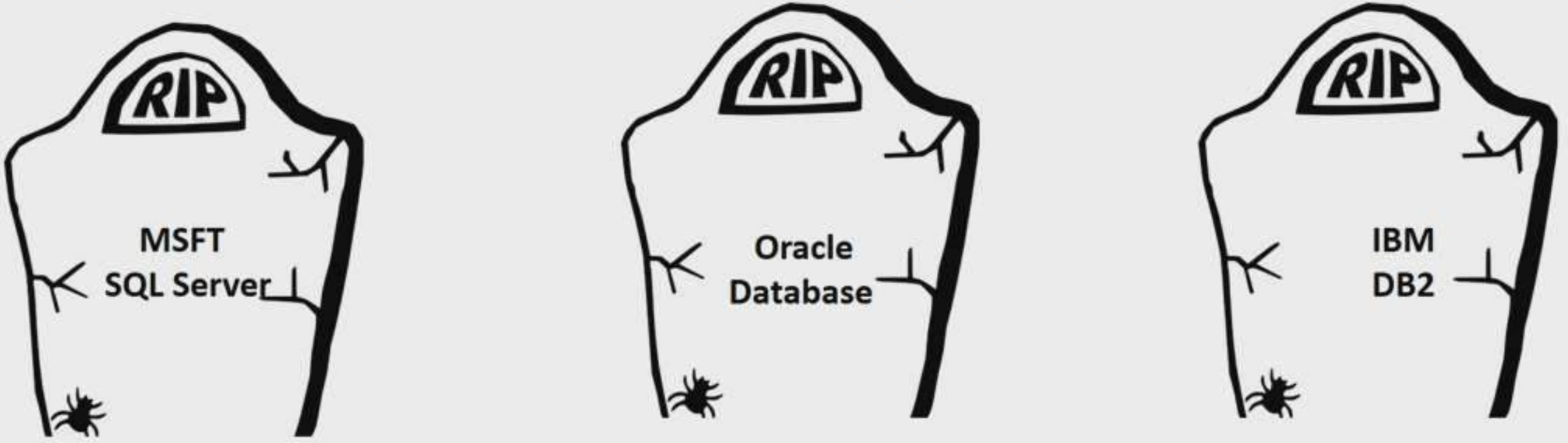


Traditional Databases are Challenged

And the threat is
Perceived to be growing



Traditional Data Caching Systems are Dead



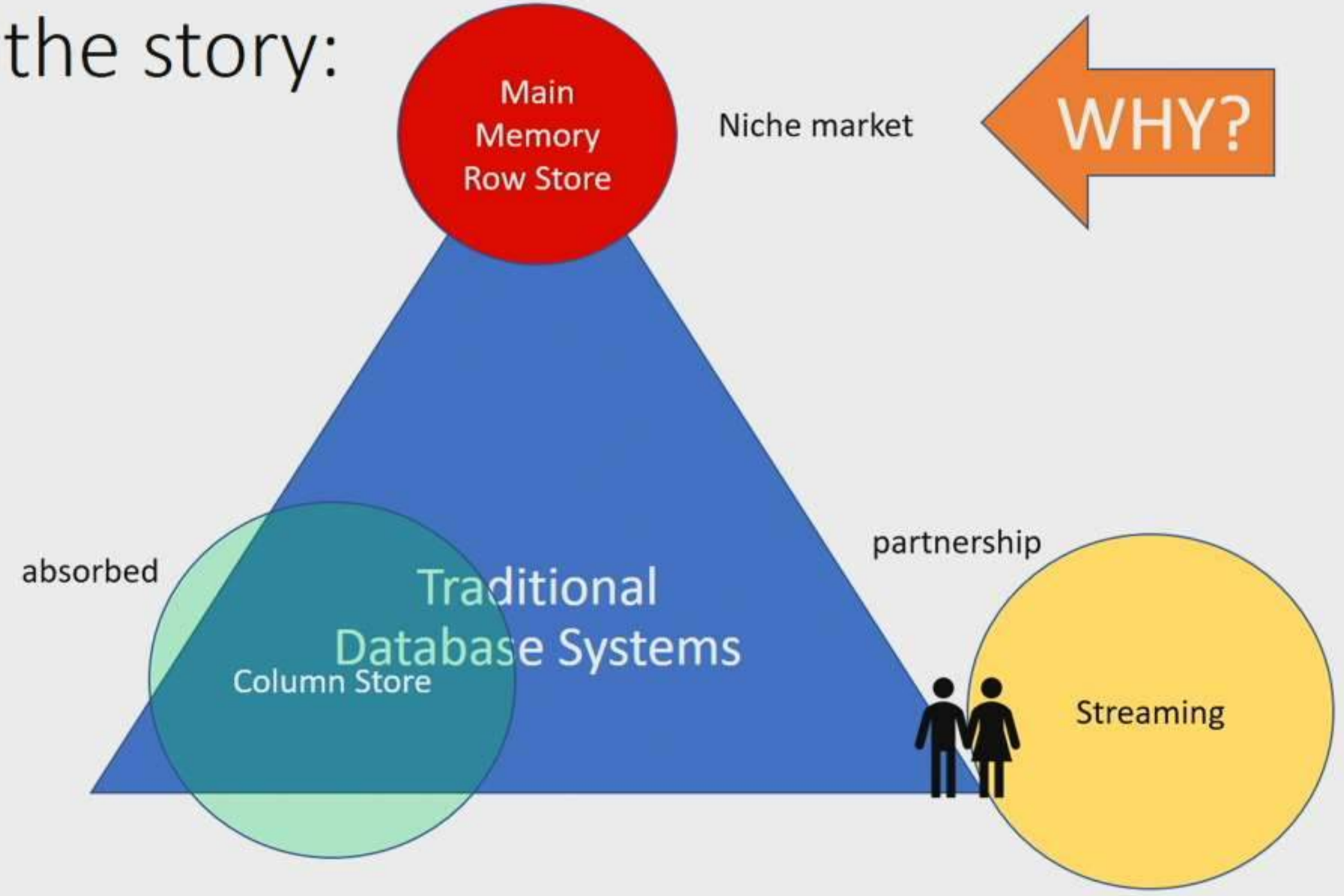
Amazon Aurora

Traditional Data Caching Systems are Dead

The Elephants Still Dance

WHY?

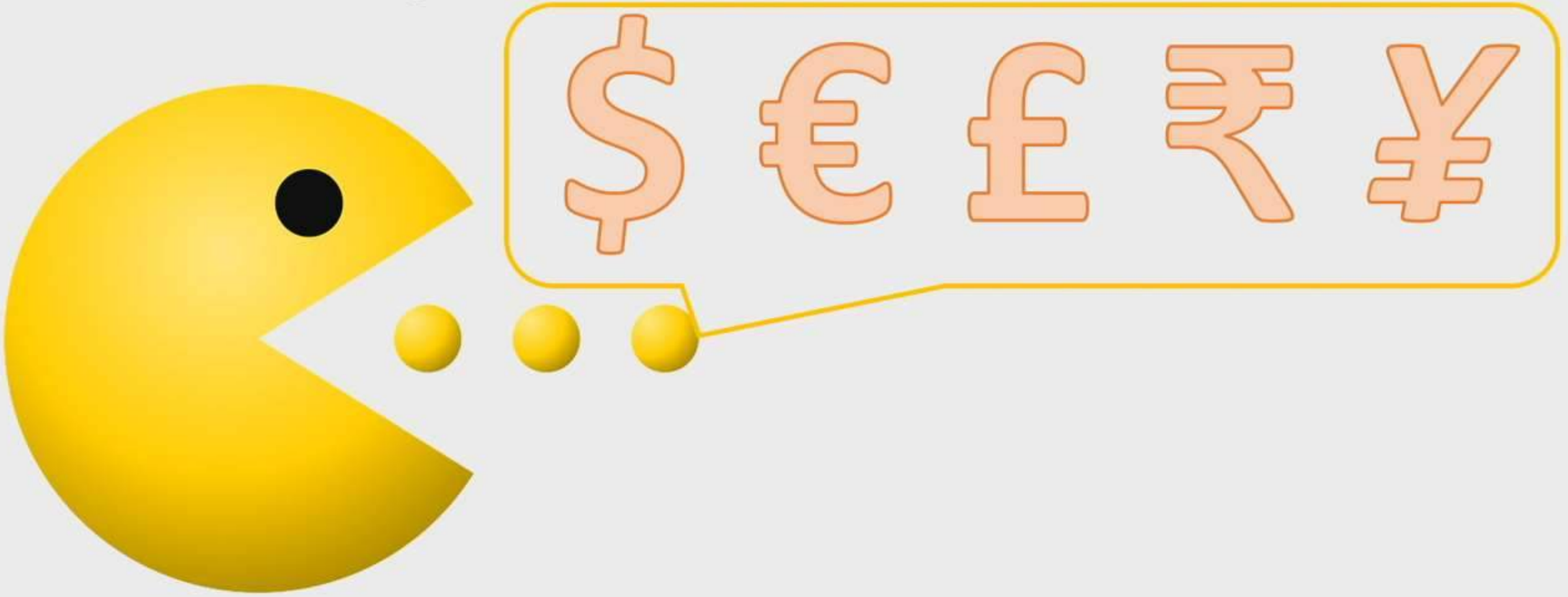
Here's the story:



Economic Factors

- Hardware infrastructure needs to be
  - Cheap => Commodity hardware
- Software infrastructure needs to be
  - Efficient => high performance ***on this hardware***
- Elastic Provisioning
  - Free up expensive resources when otherwise under-utilized
  - Give resources to those with higher value (willing to pay more)
  - Under-provision for SLAs and “pocket” the difference

Money Talks!



Traditional Database Systems Are

Caching Data Stores

Traditional DBMSs have good Cost/Performance

- We need to move away from being fixated only on performance
- Costs vary enormously between systems
- We describe why traditional systems
  - Have lower performance than recent research systems
  - Yet succeed by having better cost/performance
- And suggest a change in focus for our field

Caching Data Store …. Uses a Cache

- Moves data from secondary storage into main memory when it is to be operated on
- Removes data from main memory when it is no longer being operated on
  - Writes it back to secondary storage when it has been changed
  - Drops it from main memory when unchanged
- Data lives permanently on secondary storage

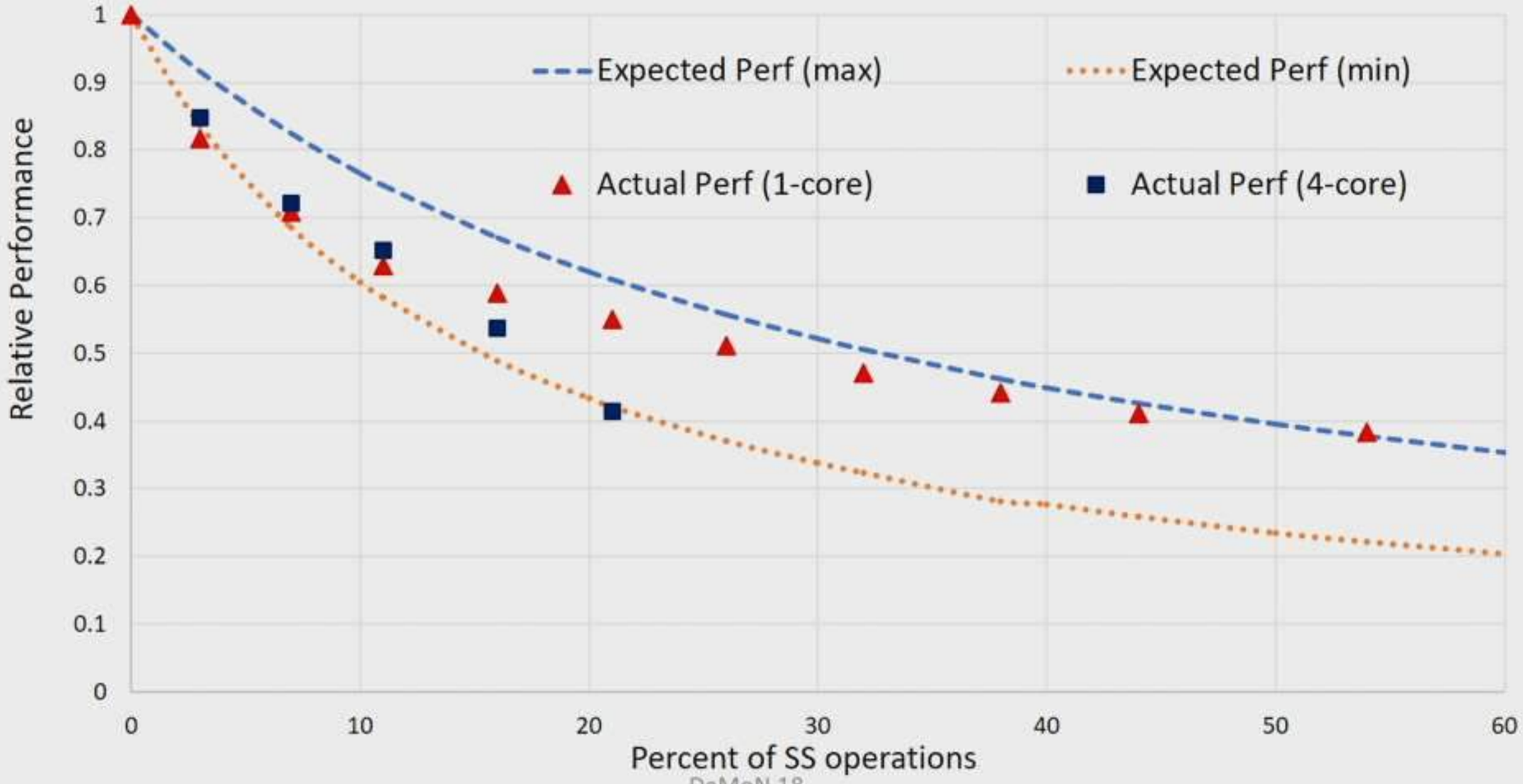
Result: Two Forms of Operation

- **MM**: main memory
  - VERY fast– N*10**6 ops/sec
- **SS**: secondary storage
  - About **5.8X** execution time of MM operations (our system)
  - Must do MM operation + I/O to bring data into main memory

- **Caching data stores look like a BAD idea**

Mixed Operation Performance

weighted average of MM and SS operations: R = SS/MM ~ 5.8 +- 30%

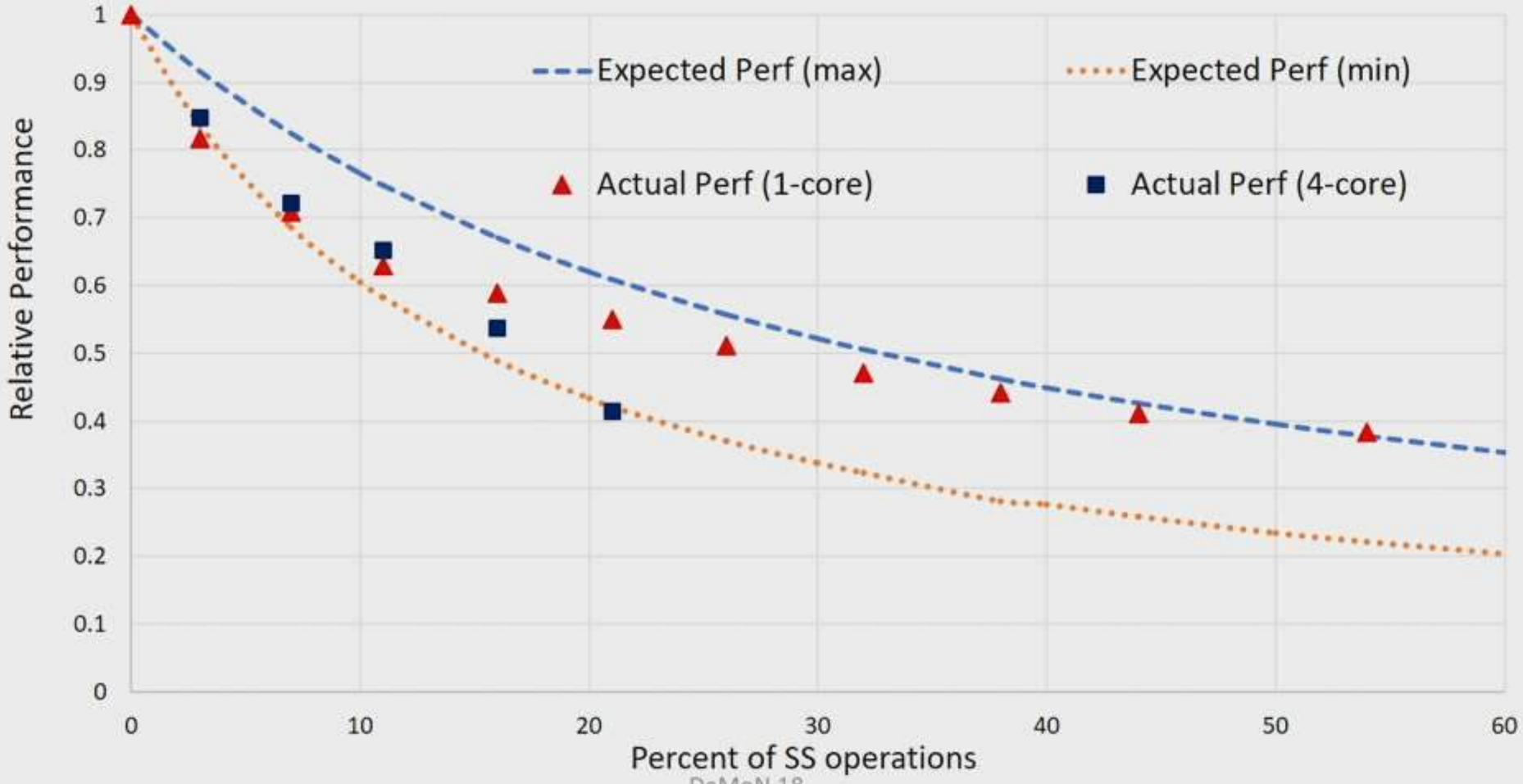


A Performance Disaster?

- Each added SS operation pushes performance lower
  - Toward SLOW SS performance
  - Away from FAST MM performance
  - Why Bother???

Mixed Operation Performance

weighted average of MM and SS operations: R = SS/MM ~ 5.8 +- 30%



A Performance Disaster?

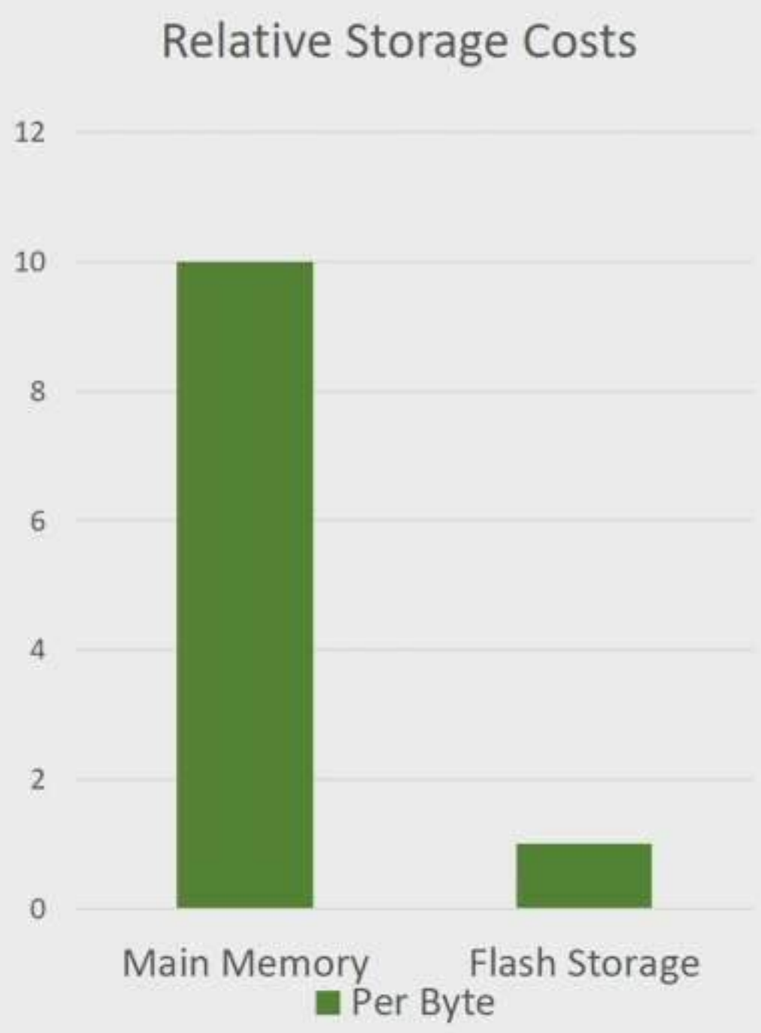
- Each added SS operation pushes performance lower
  - Toward SLOW SS performance
  - Away from FAST MM performance
  - Why Bother???
- Costs matter in addition to performance

So let’s look at Cost/Performance

Two Costs for Operating on Data

- Storage Cost
  - Always paid
  - Most of cost for cold data
    - Or when data becomes cold
- Execution Cost
  - Paid only when data is operated on
  - Most of cost for hot data
    - Or when data becomes hot

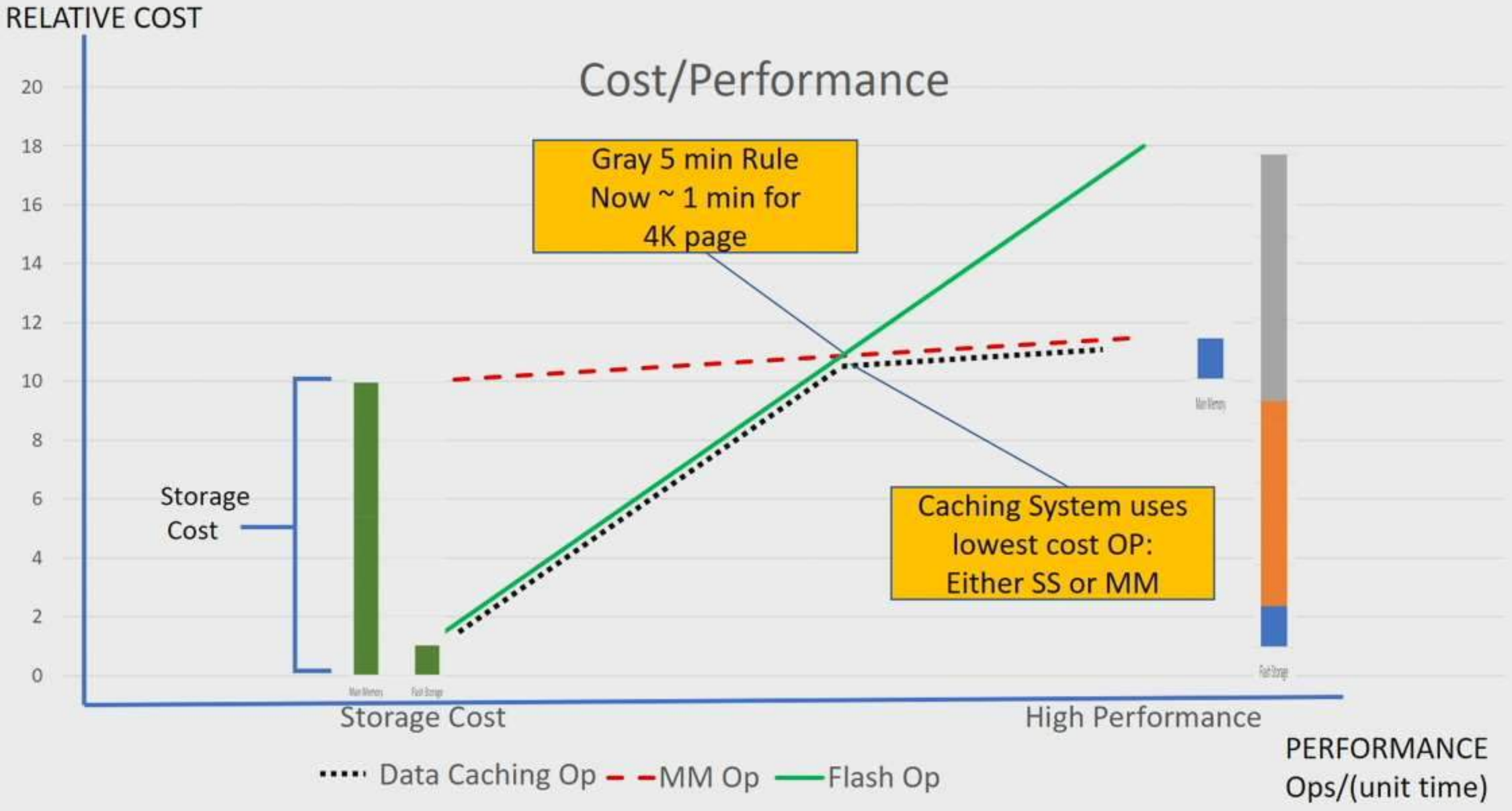
Its All About Relative Costs



Approximate Bw-tree Relative Numbers

Cost Analysis

- We compute cost/sec and plot it against operations/sec executed
- Cost is a linear function of storage and execution costs:
  - C = A + BX, where we will plot C vs X
- Cost/sec = (storage-rental/sec) + (cost/operation)*(operations/sec)
- **C = A + B * X**
- **A: Storage cost** = (cost/byte)*(size of data) and is constant for an operation on a piece of data
  - DRAM cost/byte ~ 10X Flash cost/byte
  - At ops/sec = 0, all cost is storage cost: **[storage cost is y-intercept]**
- **Execution cost** = (cost/operation)*(operations/sec)
  - **B: Cost/operation:** SS op ~ 12X MM op
  - **Cost/operation determines the slope of the cost "curve"**


RELATIVE COST
Cost/Performance
20
18
16
14
12
10
8
6
4
2
0
Gray 5 min Rule
Now ~ 1 min for
4K page
Caching System uses
lowest cost OP:
Either SS or MM
Storage
Cost
Storage Cost
High Performance
PERFORMANCE
Ops/(unit time)
Data Caching Op
MM Op
Flash Op

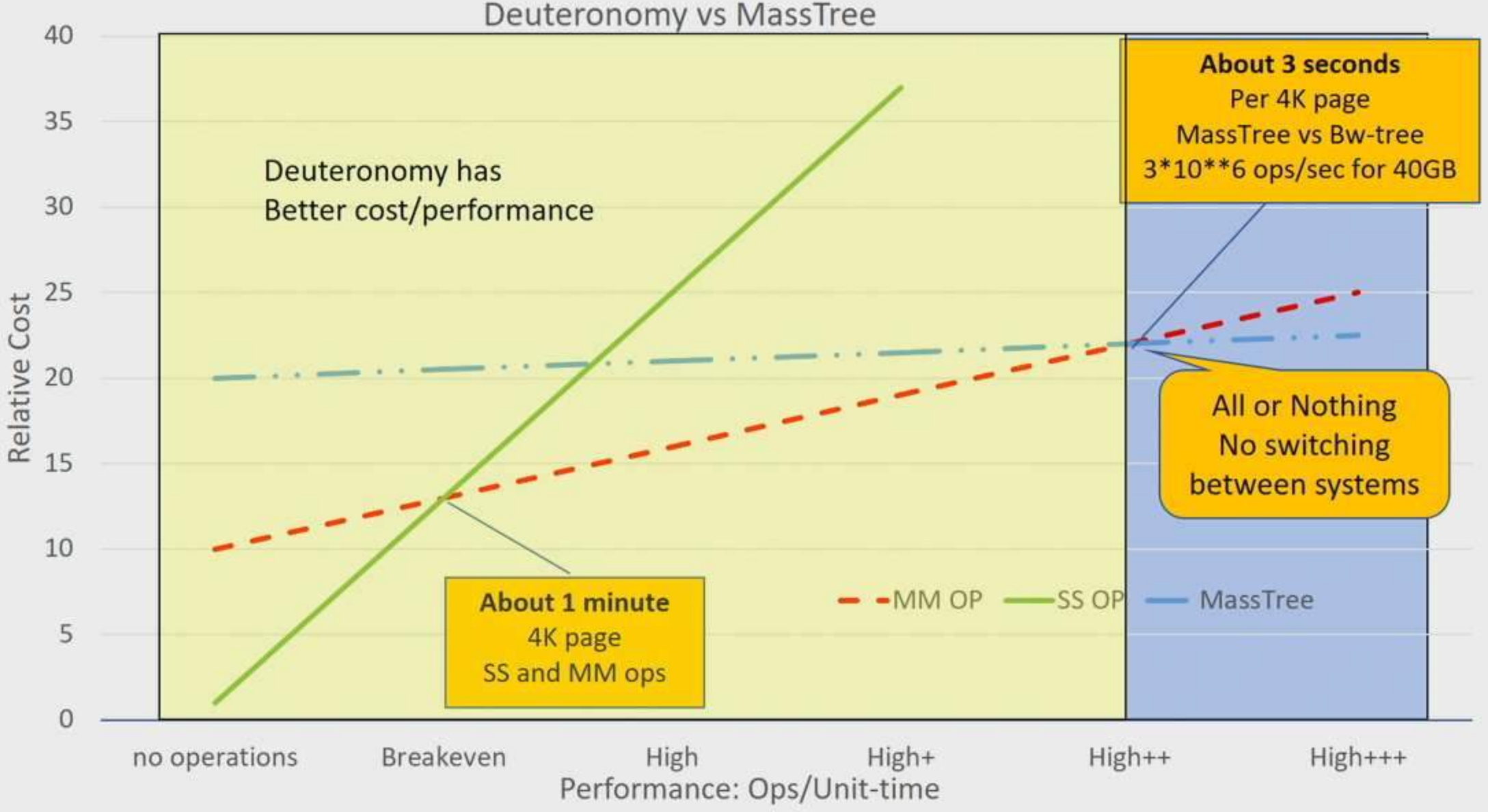
How About Other Kinds of Systems

Main Memory Systems Perform Better

indeed, data caching systems perform better when fully cached
– but with worse cost/performance

- MassTree: high performance main memory data store
- Deuteronomy: high performance caching data store
- MassTree has higher performance than Deuteronomy's Bw-tree
- Relative costs
  - Execution cost: Deuteronomy is 2.6 X MassTree
    - because MassTree is 2.6X faster
  - Storage cost: MassTree is 2.3 X Deuteronomy
    - Because MassTree uses 2.3X the storage of Deuteronomy

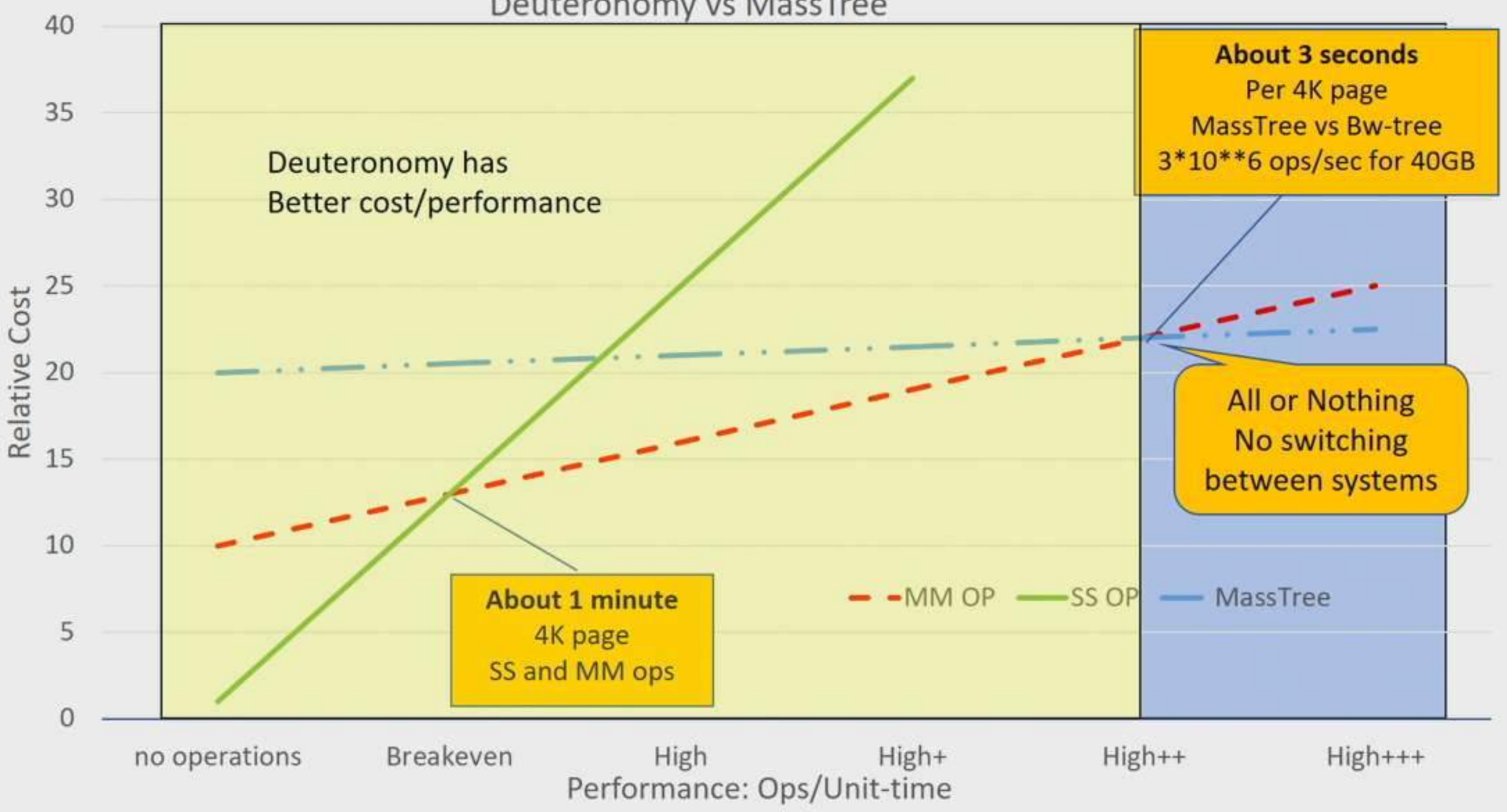
Cost/Performance: [Not to Scale] Data Caching Frequently Better



Lowering Storage Cost

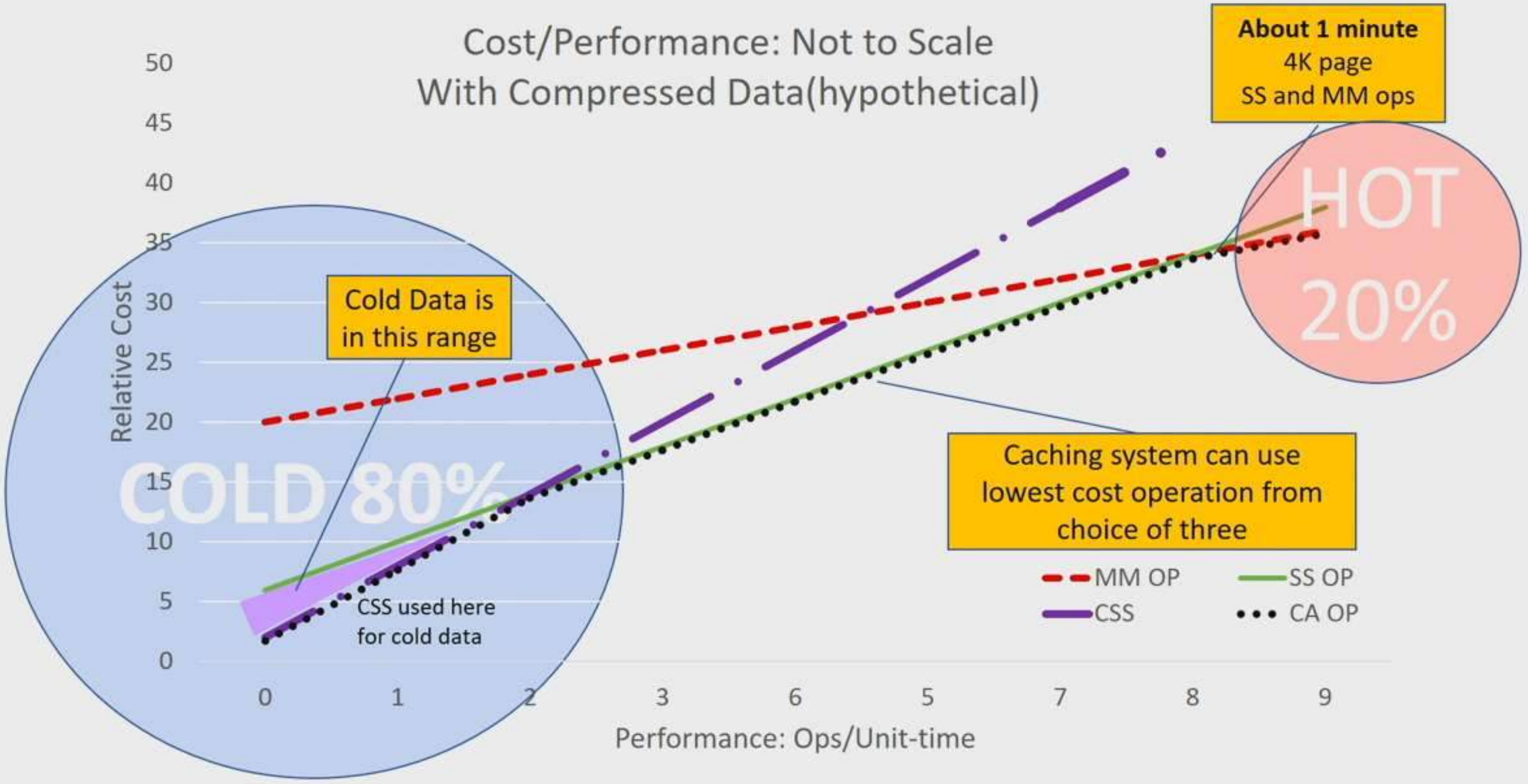
- Facebook problem:
  - ***huge data volume***
  - almost all cold
  - But requiring high performance and decent latency

Cost/Performance: [Not to Scale] Data Caching Frequently Better



Lowering Storage Cost

- Facebook problem:
  - ***huge data volume***
  - almost all cold
  - But requiring high performance and decent latency
- Buying processors to be able to attach more SSDs
  - To accommodate volume of data
- Solution: data caching system + data compression
  - Use SSD storage for decent latency
  - Scale out processors for good performance
  - Data compression to lower storage cost
    - Recall that storage cost is (cost/byte)*(size of data)
    - ***At the expense of increased execution cost***


Cost/Performance: Not to Scale
With Compressed Data(hypothetical)
About 1 minute
4K page
SS and MM ops
HOT
20%
COLD 80%
Cold Data is
in this range
Caching system can use
lowest cost operation from
choice of three
CSS used here
for cold data
MM OP
SS OP
CSS
CA OP
Relative Cost
Performance: Ops/Unit-time
0
5
10
15
20
25
30
35
40
45
50
0
1
2
3
6
5
7
8
9

Making a High Performance and Low Cost Data Caching Systems

How do we provide good cost/perf?

Data caching system is part of answer

But how do we optimize cost/performance in a data caching system

- ❑Increase in-memory multi-core performance
  - Increased performance reduces cost – must be relevant to data caching
  - Improve single core performance, and multi-core scale-out
- ❑Reduce number of I/Os
  - Blind writes, log structuring, record cache
- ❑Reduce data movement cost: SSD to/from main memory
  - Reduce execution cost of I/O: e.g., user-level I/O
  - Reduce SSD I/O cost: e.g., SSD with more IOPS
  - **Modify SSD: summer project with Jae Young Do and Ivan Picoli**
- ❑Reduce cost of secondary storage
  - Compress Data for fewer bytes
  - Use flash, not NVRAM (NVRAM ~ 3X cost of Flash)

CAVEATS

Two important ones

Our Numbers are ApPR☺XIMATE

- Performance based on
  - Set of point experiments
  - Executed on our machines
  - Using our Deuteronomy system
- Costs from web prices
  - Variable at any one time
  - Changing over time

- General thrust of the results is solid
  - But your mileage may vary!

Best cost/performance is NOT always best

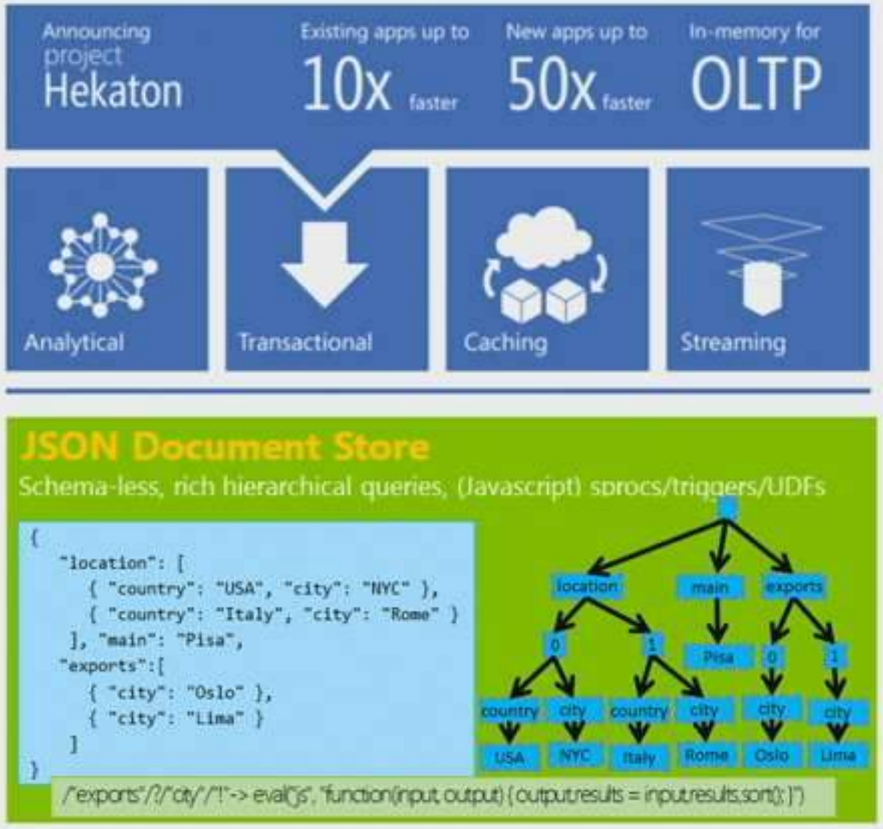
- Low costs are a plus, but not the only thing
- We want *max[value – cost]*
  - If value is high enough, low cost may be secondary
- Value might depend on latency (or peak performance)
  - Some of the time, lowest latency wins the game and captures all the value
    - E.g. some stock trading applications
    - But this is not the common case
- Most of time, interactive system latency is adequate
  - Anything less than a millisecond per op should be fine
  - E.g. online shopping

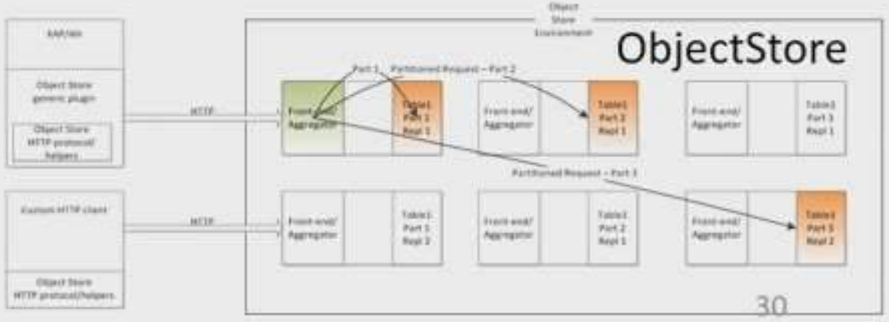
Recap

- Two costs
  - Storage: always present
  - Execution: only when operations execute
- When data is:
  - Cold: storage cost dominates
  - Hot: execution cost dominates
- Data Caching Systems "win" on cost/performance because they
  - Move hot data to (high cost) main memory cache for low execution cost
  - Move cold data to (low cost) secondary storage for low storage cost
- **We should be focusing on data caching improvements!**

Talk brought to you by Bw-Tree in Production

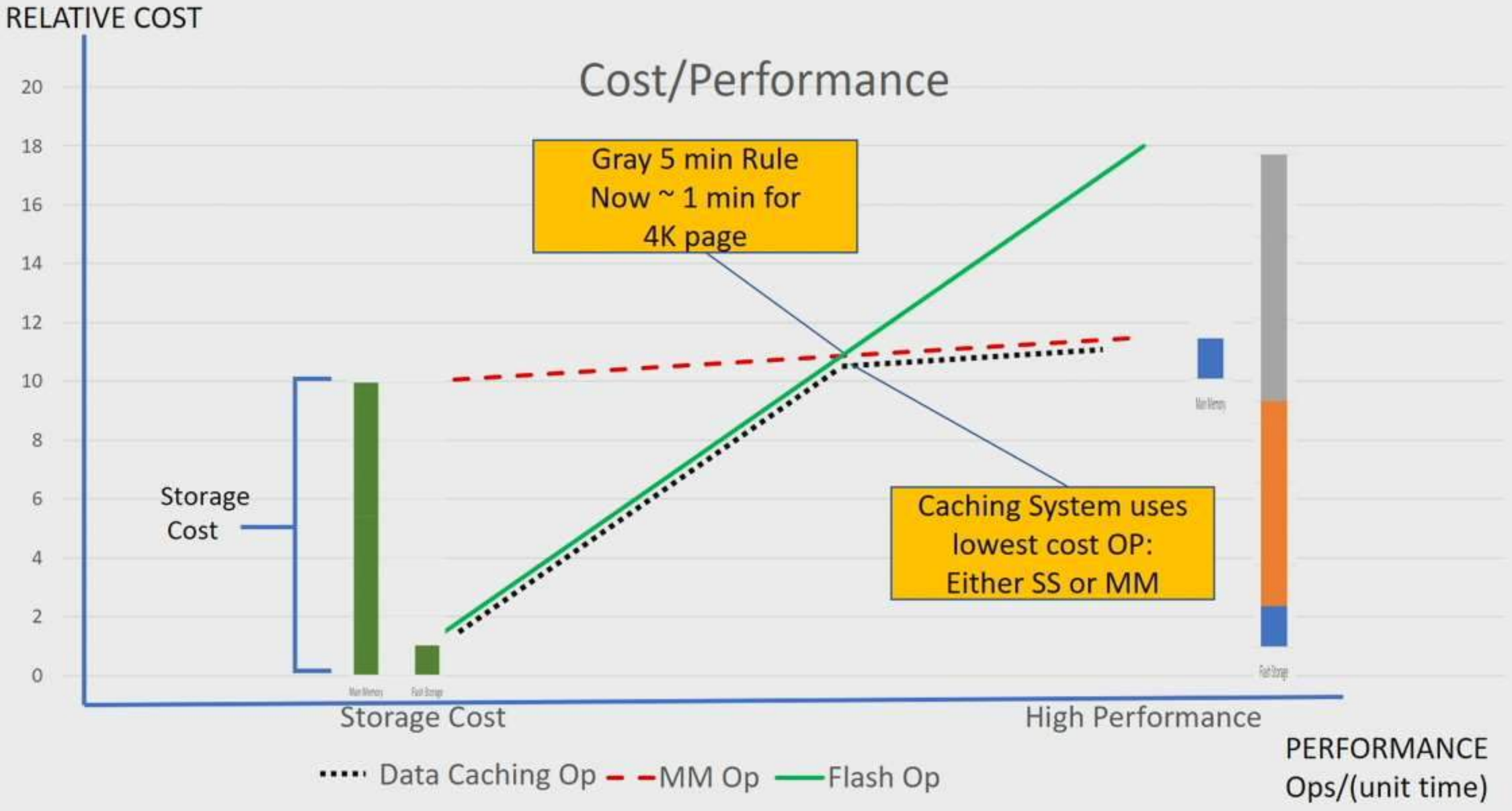
- SQL Server Hekaton: Key-sequential index
  - Lock-free for high concurrency
    - consistent with Hekaton's non-blocking main memory architecture
  - **In-memory Bw-tree**
- Azure DocumentDB: Indexing engine
  - Rich query processing over a schema-free JSON model, with *automatic indexing*
  - Sustained document ingestion at high rates
  - **Bw-tree + LLAMA**
- Bing ObjectStore: Sorted key-value store
  - Supports range queries
  - Optimized for flash SSDs
  - **Bw-tree + LLAMA**





Some References

- David B. Lomet: **Cost/performance in modern data stores: how data caching systems succeed.** DaMoN 2018: 9:1-9:10 https://dl.acm.org/citation.cfm?doid=3211922.3211927
- Justin J. Levandoski, David B. Lomet, Sudipta Sengupta: **The Bw-Tree: A B-tree for new hardware platforms.** ICDE 2013: 302-313
- Justin J. Levandoski, David B. Lomet, Sudipta Sengupta: **LLAMA: A Cache/Storage Subsystem for Modern Hardware.** PVLDB 6(10): 877-888 (2013) http://www.vldb.org/pvldb/vol6/p877-levandoski.pdf
- Justin J. Levandoski, David B. Lomet, Sudipta Sengupta, Ryan Stutsman, Rui Wang: **High Performance Transactions in Deuteronomy.** CIDR 2015 http://cidrdb.org/cidr2015/Papers/CIDR15_Paper15.pdf


RELATIVE COST
Cost/Performance
20
18
16
14
12
10
8
6
4
2
0
Gray 5 min Rule
Now ~ 1 min for
4K page
Caching System uses
lowest cost OP:
Either SS or MM
Storage
Cost
Storage Cost
High Performance
PERFORMANCE
Ops/(unit time)
Data Caching Op
MM Op
Flash Op