WISCONSIN
UNIVERSITY OF WISCONSIN–MADISON

CS 839: Design the Next-Generation Database

Lecture 12: HBM

Xiangyao Yu
2/27/2020

Announcements

Upcoming deadlines:

- Form groups: **Today**
- Proposal due: **Mar. 10**

Fill this Google sheet for course project information

- https://docs.google.com/spreadsheets/d/1W7ObfjLqjDChm49GqrLg49x6r4B28-f-PBpQPHX01Mk/edit?usp=sharing

Project Proposal

Use VLDB 2020 format

- https://vldb2020.org/formatting-guidelines.html

The proposal is **1-page** containing the following

- Project name
- Author list
- Abstract (1-2 paragraphs about your idea)
- Introduction (Why is the problem interesting; what's your contribution)
- Methodology (how do you plan to approach the problem)
- Task-list (Who works on what tasks of the project)
- Timeline (List of milestones and when you plan to achieve them)

Submit proposal by March 10 to https://wisc-cs839-ngdb20.hotcrp.com

Discussion Highlights

High availability and remote recovery

- HA requires synchronous round trip
- Network may become a bottleneck with HA
- WBL cannot be simply extended to provide HA or remote recovery. Possible solutions (1) add REDO logging (2) use RDMA to directly update remote NVM

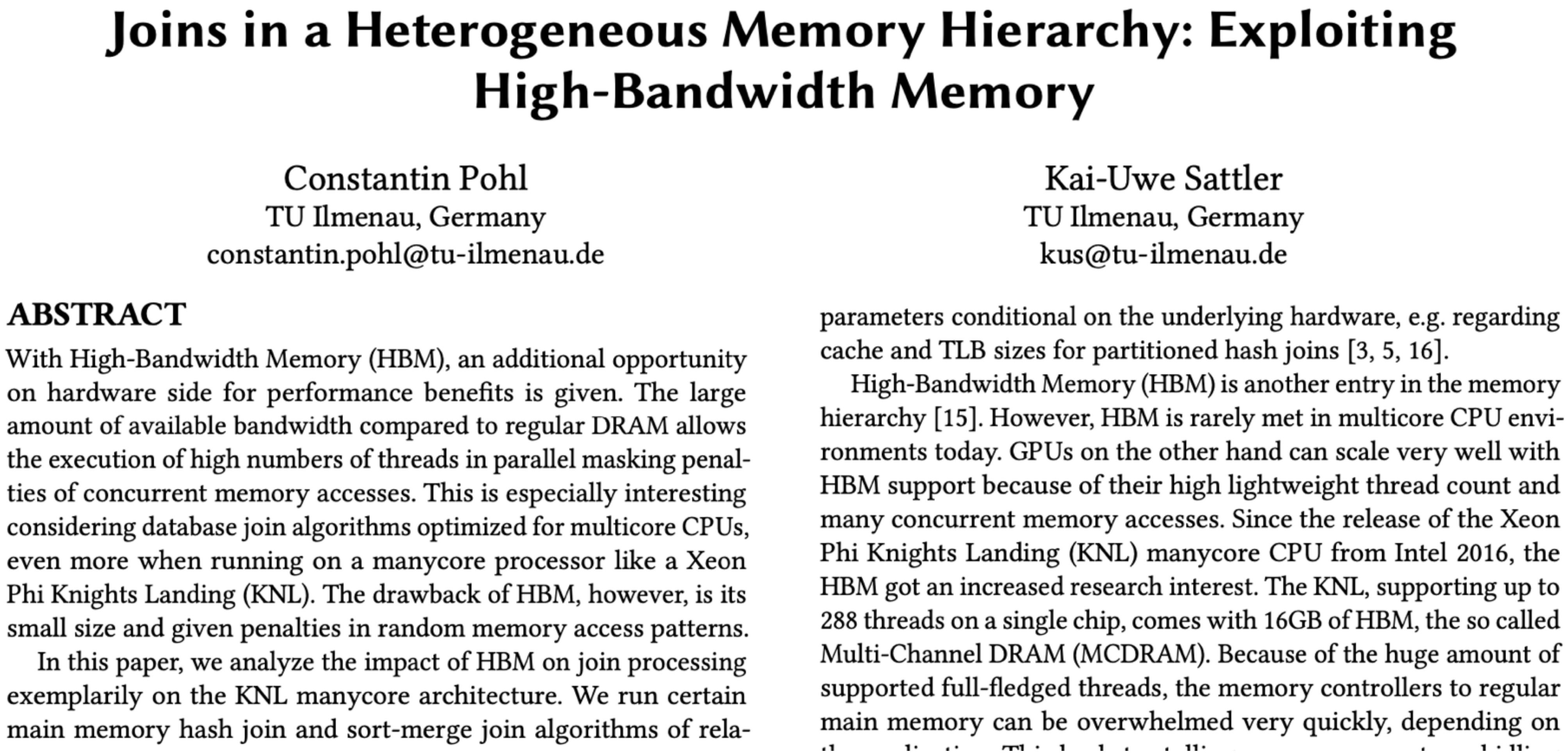
REDO vs. UNDO vs. WBL

- WBL pros: instant recovery, small log size
- WBL cons: requires multi-versioning, works only for NVM
- UNDO: bounded log size (typically small)

WBL with three-tier architecture

- Challenges: Pulling cold data from SSD, page vs. byte granularity
- Log and hot data in NVM, cold data in SSD

Today’s Paper



Joins in a Heterogeneous Memory Hierarchy: Exploiting High-Bandwidth Memory

Constantin Pohl
TU Ilmenau, Germany
constantin.pohl@tu-ilmenau.de

Kai-Uwe Sattler
TU Ilmenau, Germany
kus@tu-ilmenau.de

ABSTRACT

With High-Bandwidth Memory (HBM), an additional opportunity on hardware side for performance benefits is given. The large amount of available bandwidth compared to regular DRAM allows the execution of high numbers of threads in parallel masking penalties of concurrent memory accesses. This is especially interesting considering database join algorithms optimized for multicore CPUs, even more when running on a manycore processor like a Xeon Phi Knights Landing (KNL). The drawback of HBM, however, is its small size and given penalties in random memory access patterns.

In this paper, we analyze the impact of HBM on join processing exemplarily on the KNL manycore architecture. We run certain main memory hash join and sort-merge join algorithms of rela-

parameters conditional on the underlying hardware, e.g. regarding cache and TLB sizes for partitioned hash joins [3, 5, 16].

High-Bandwidth Memory (HBM) is another entry in the memory hierarchy [15]. However, HBM is rarely met in multicore CPU environments today. GPUs on the other hand can scale very well with HBM support because of their high lightweight thread count and many concurrent memory accesses. Since the release of the Xeon Phi Knights Landing (KNL) manycore CPU from Intel 2016, the HBM got an increased research interest. The KNL, supporting up to 288 threads on a single chip, comes with 16GB of HBM, the so called Multi-Channel DRAM (MCDRAM). Because of the huge amount of supported full-fledged threads, the memory controllers to regular main memory can be overwhelmed very quickly, depending on

DaMoN@SIGMOD 2018

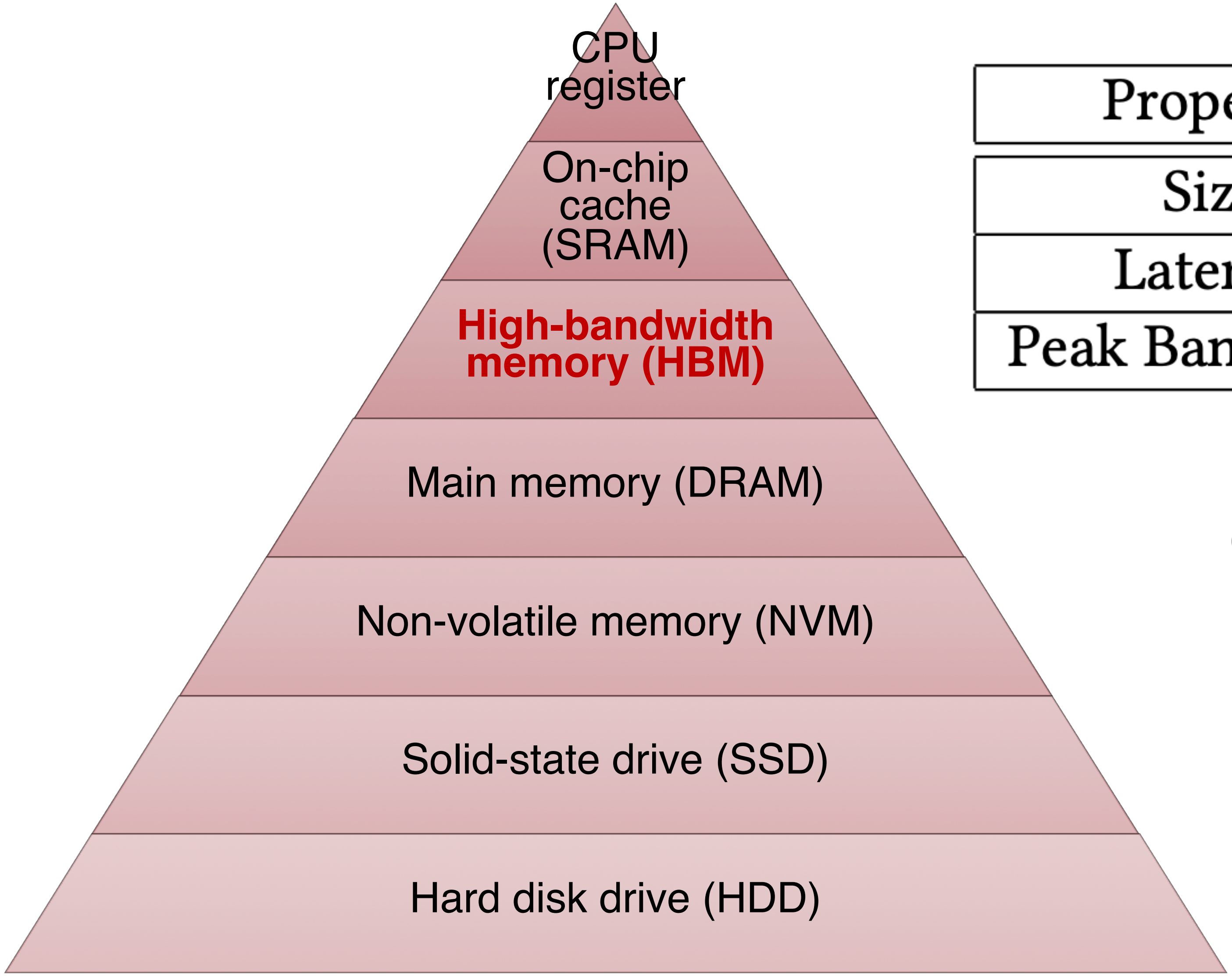
Today’s Agenda

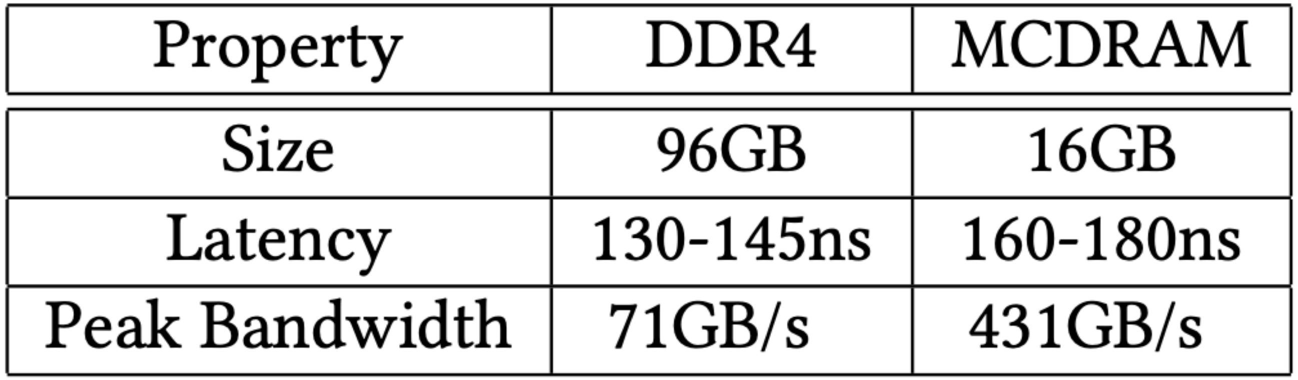
High Bandwidth Memory (HBM)

Intel Xeon Phi processor

Joins on HBM

High Bandwidth Memory (HBM)



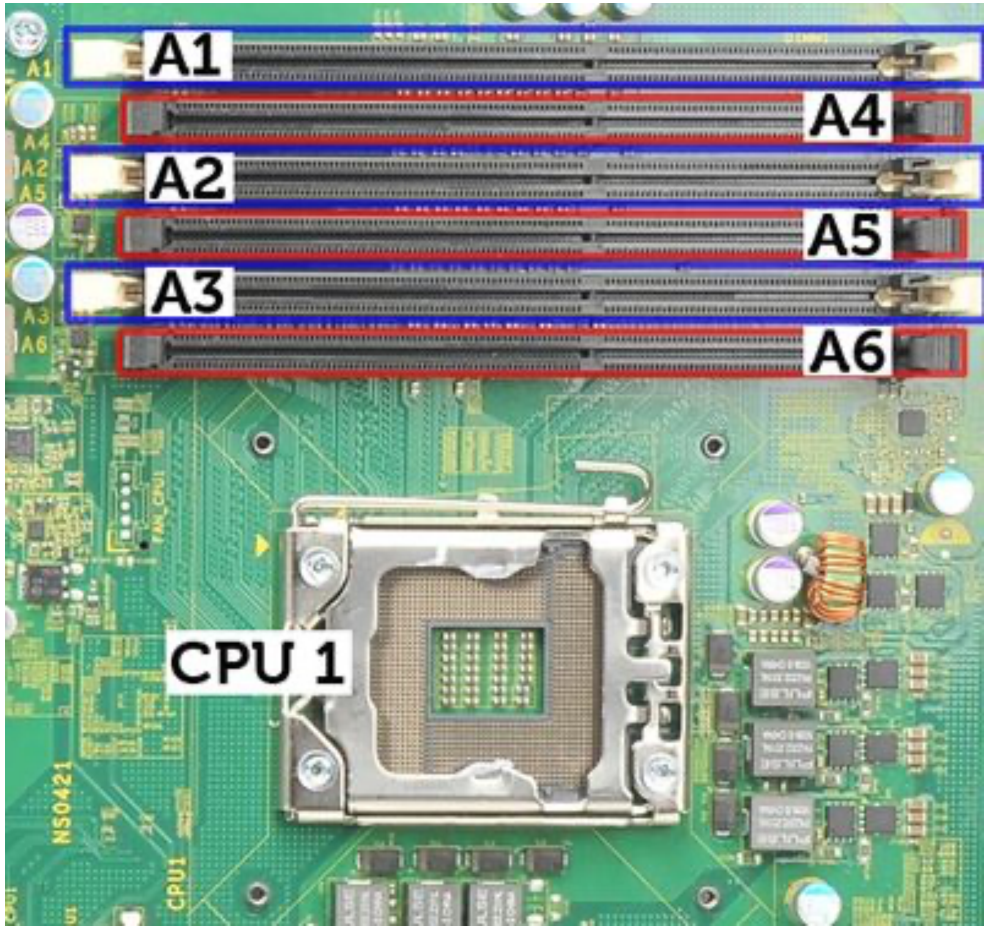


Property	DDR4	MCDRAM
Size	96GB	16GB
Latency	130-145ns	160-180ns
Peak Bandwidth	71GB/s	431GB/s

Compared to DRAM, HBM has

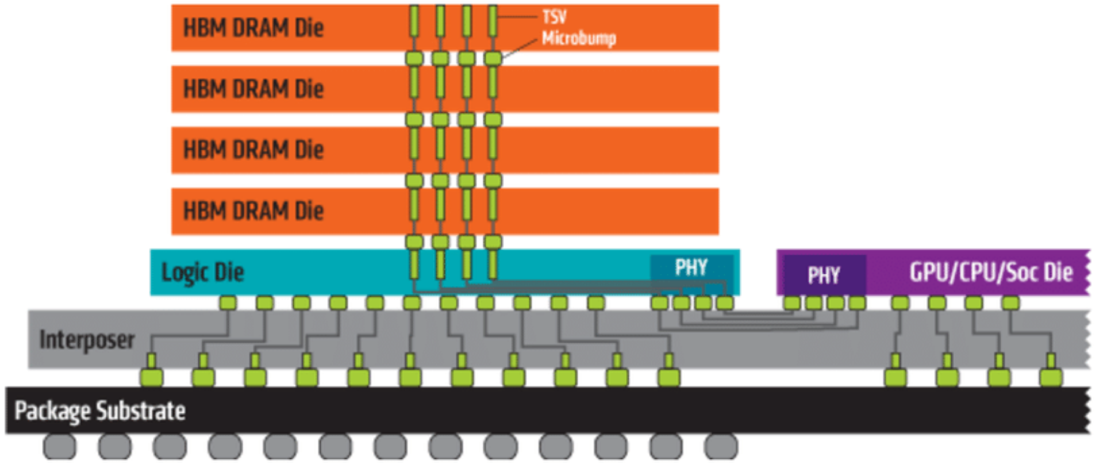
- Slightly higher latency
- Much higher bandwidth
- Limited capacity

High Bandwidth Memory (HBM)



DRAM DIMMs

High-bandwidth memory



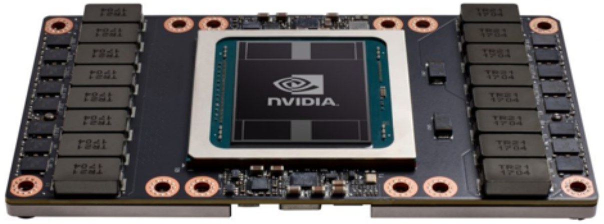
HBM Specs

	HBM (2013)	HBM2 (2016)	HBM2E (2018)	HBM3 (2020?)
Max Capacity (per stack)	4 GB	8 GB	24 GB	64 GB
Max bandwidth (per stack)	128 GB/s	256 GB/s	307 GB/s	512 GB/s

- High bandwidth memory has growing capacity and bandwidth

Where is HBM Used?

- Nvidia Pascal GPU, Nvidia Volta GPU



- AMD Radeon GPU

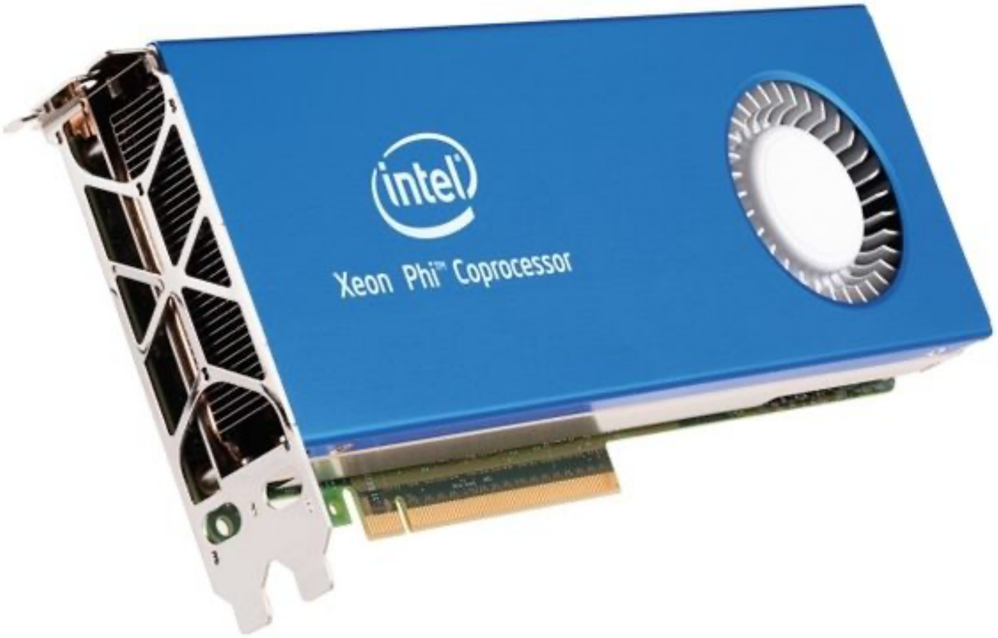
- Intel Xeon Phi CPU

HBM is mostly used in GPUs today

Intel Xeon Phi Processors

Xeon Phi

Knights Corner



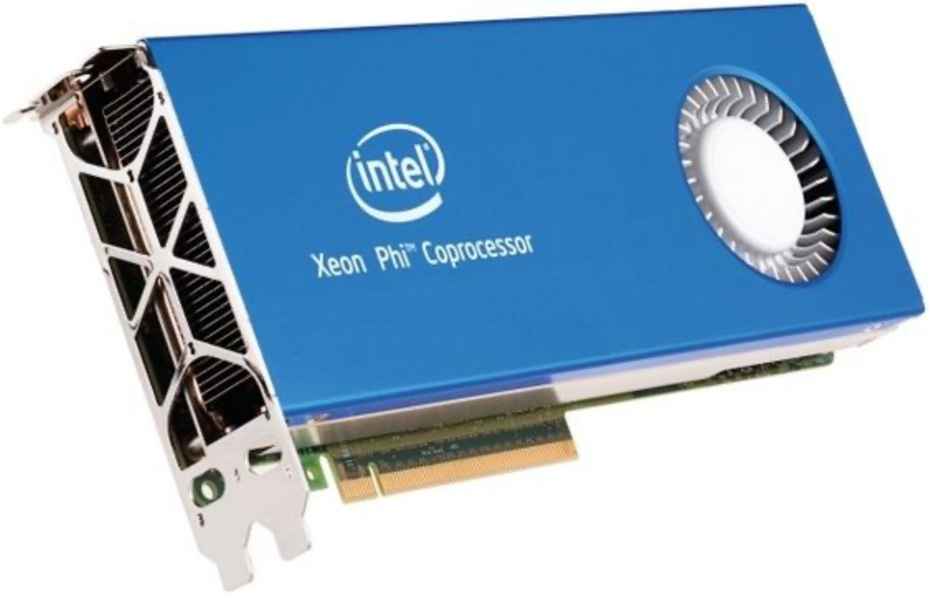
PCIe-based coprocessor

Knights Landing

Standalone CPU

Knights Corner

- 61 physical cores (244 threads)
- 16 GB GDDR5 memory
- Communicate with CPU through PCIe

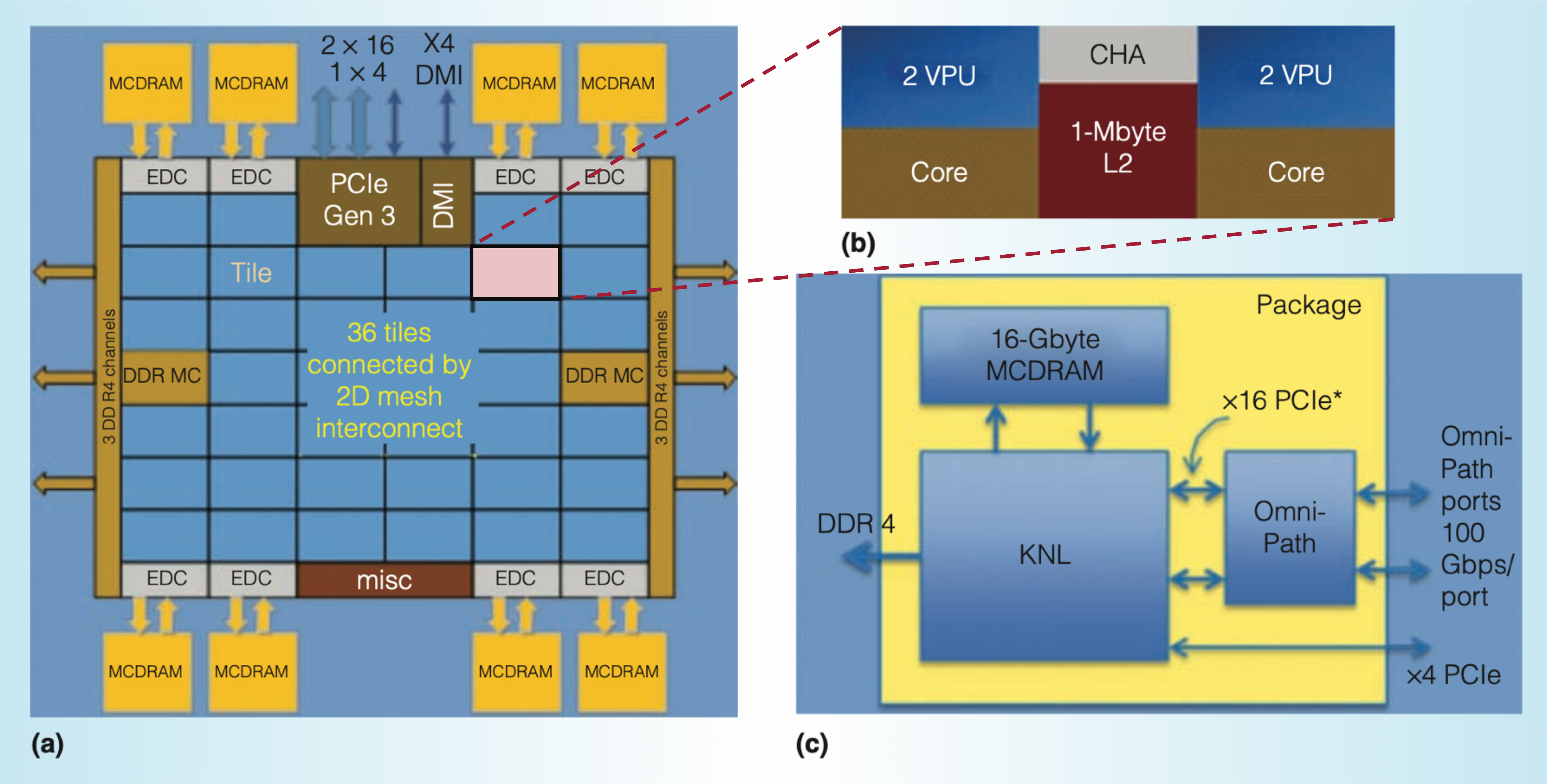


Knights Landing (KNL)

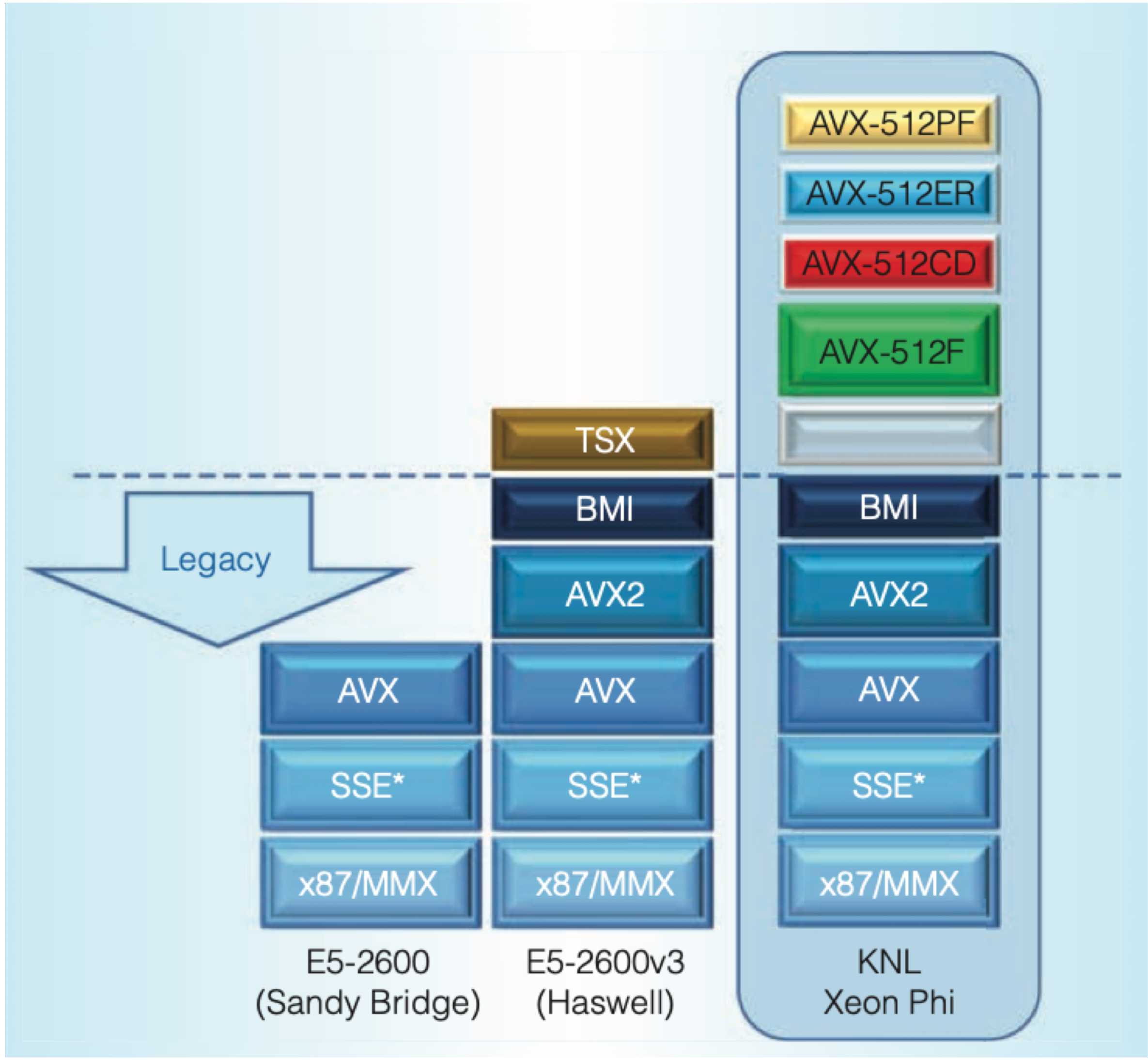
- 72 physical cores (288 threads)
- 16 GB MCDRAM (HMC)
- 400+ GB/s
- Standalone processor

	Throughput	Power	Throughput/Power
Intel Skylake	128 GFLOPS	100+ Watts	~1 GFLOPS/Watt
Knights Corner	1209 GFLOPS	300 Watts	~4 GFLOPS/Watt
Knights Landing	3456 GFLOPS	200+ Watts	~13 GFLOPTS/Watt
NVIDIA V100	15 TFLOPS	200+ Watts	~75 GFLOPS/Watt

KNL – Architecture



KNL – Instruction Set

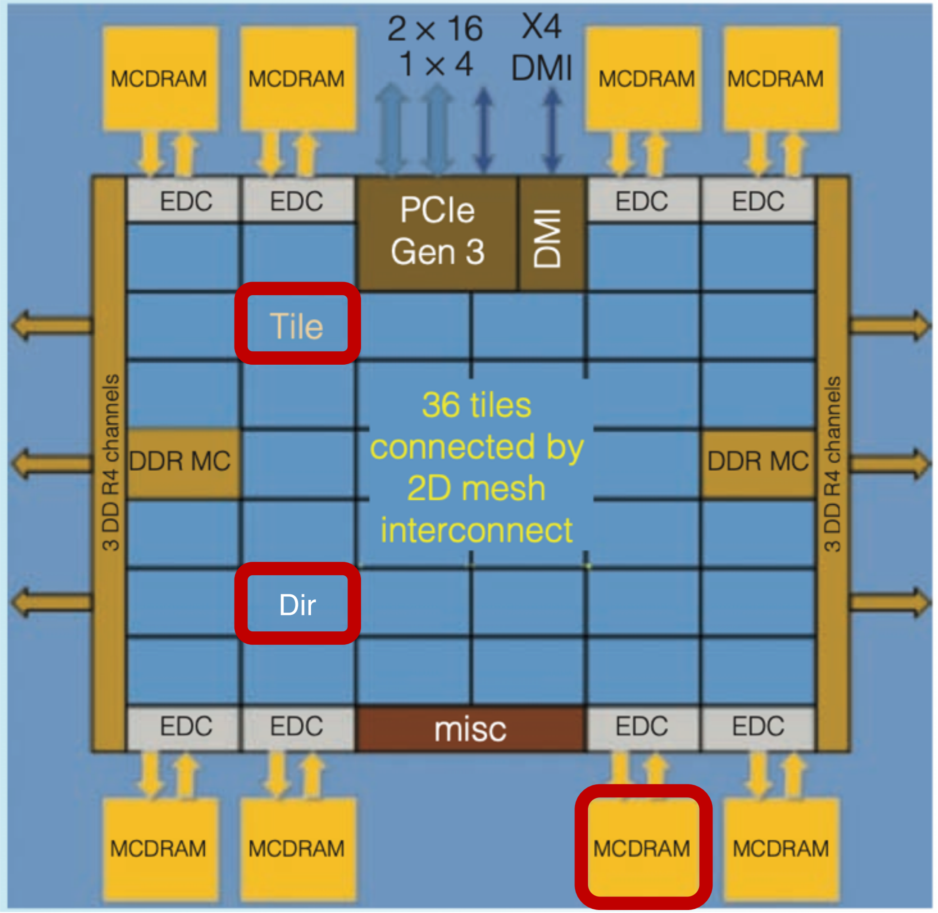


SIMD instructions

- Streaming SIMD Extensions (SSE), 128 bits
- Advanced Vector Extensions (AVX), 256 bits
- **AVX-512, 512 bits**

KNL – Cluster Modes

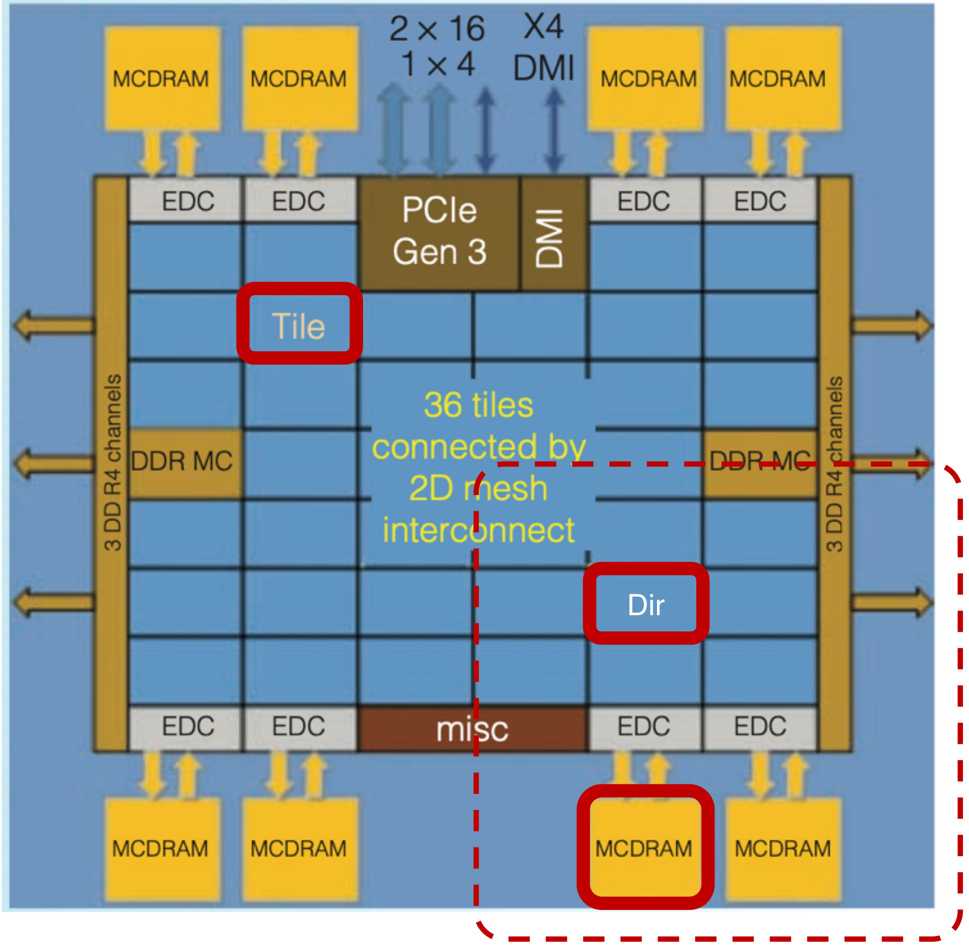
All-to-all mode: No affinity between the tile, directory, and memory



KNL – Cluster Modes

All-to-all mode: No affinity between the tile, directory, and memory

Quadrant mode: Divides the KNL chip into four quadrants; affinity between directory and memory, no affinity between tile and directory

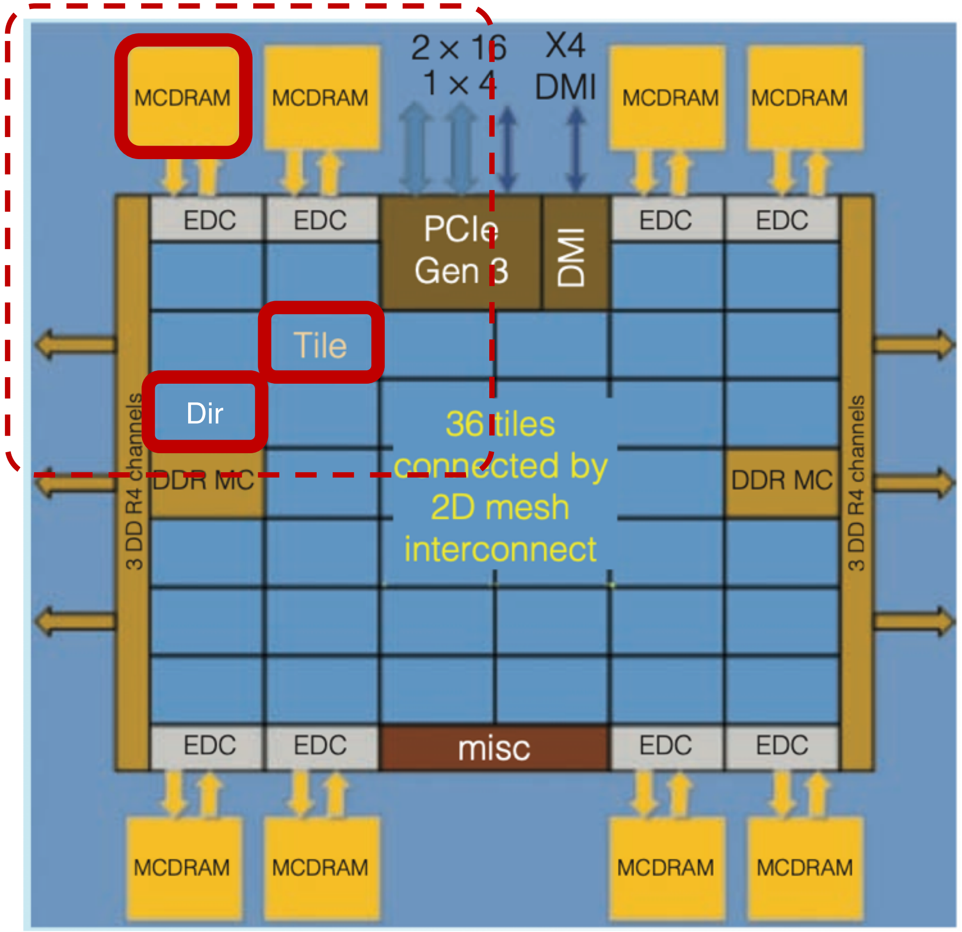


KNL – Cluster Modes

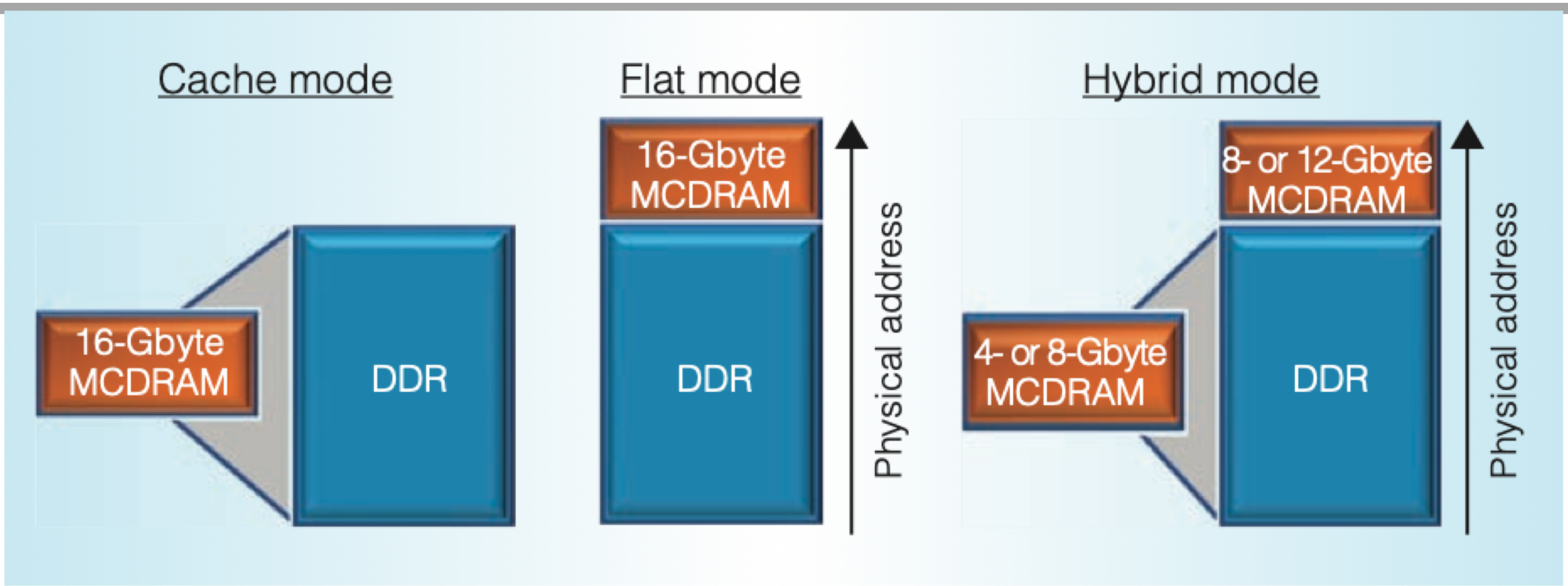
All-to-all mode: No affinity between the tile, directory, and memory

Quadrant mode: Divides the KNL chip into four quadrants; affinity between directory and memory, no affinity between tile and directory

Sub-NUMA clustering (SNC): Divides the KNL chip into two or four nonuniform memory access (NUMA) domains. Affinity between tile, directory, and memory



KNL – Memory Modes

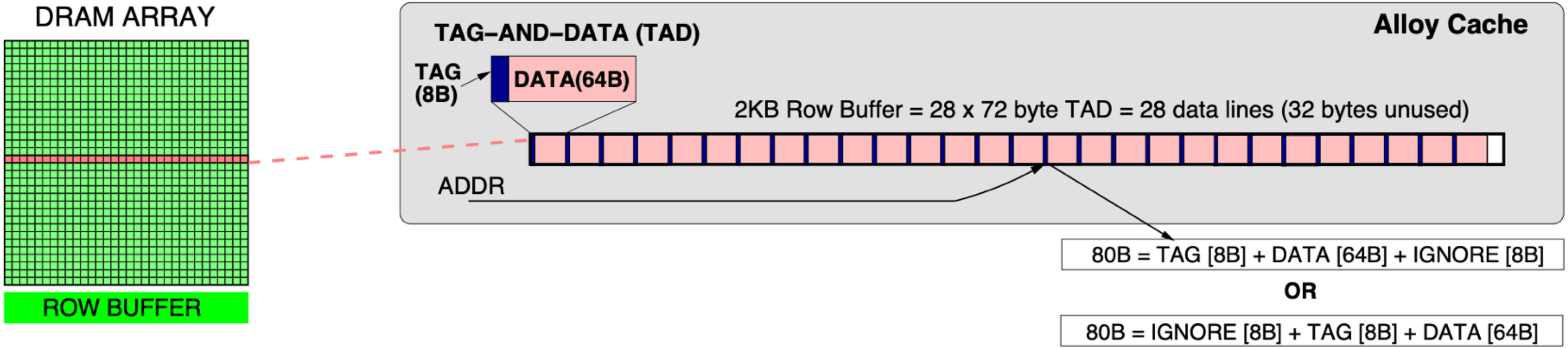


Cache Mode: Memory side cache for DDR memory; transparent to software

Flat Mode: Software sees both address spaces (malloc() vs. hbw_malloc())

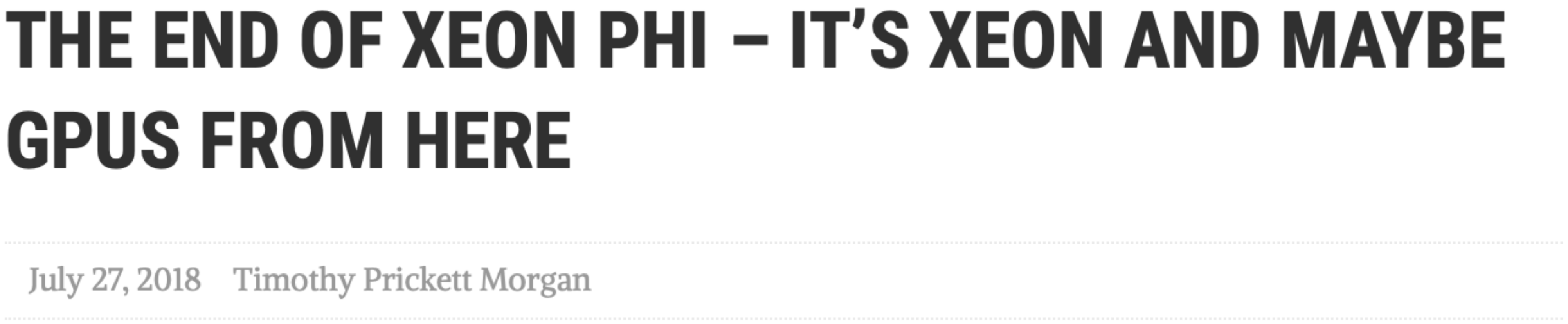
Hybrid Mode: Software sees both address spaces

KNL – Memory Modes – Cache Mode


DRAM ARRAY
ROW BUFFER
TAG–AND–DATA (TAD)
Alloy Cache
TAG
(8B)
DATA(64B)
2KB Row Buffer = 28 x 72 byte TAD = 28 data lines (32 bytes unused)
ADDR
80B = TAG [8B] + DATA [64B] + IGNORE [8B]
OR
80B = IGNORE [8B] + TAG [8B] + DATA [64B]

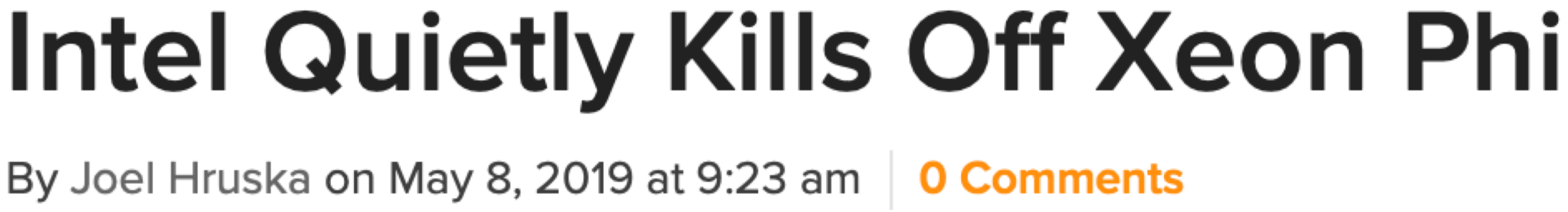
- Direct-mapped cache (A cacheline maps to only one location in MCDRAM)
- Cache lookup: loads both tag and data
  - For a hit, return data
  - For a miss, access DDR DRAM

Xeon Phi Today?



THE END OF XEON PHI – IT'S XEON AND MAYBE GPUS FROM HERE

July 27, 2018 Timothy Prickett Morgan



Intel Quietly Kills Off Xeon Phi

By Joel Hruska on May 8, 2019 at 9:23 am | 0 Comments

Joins on HBM

Join Algorithms

Symmetric Hash Join

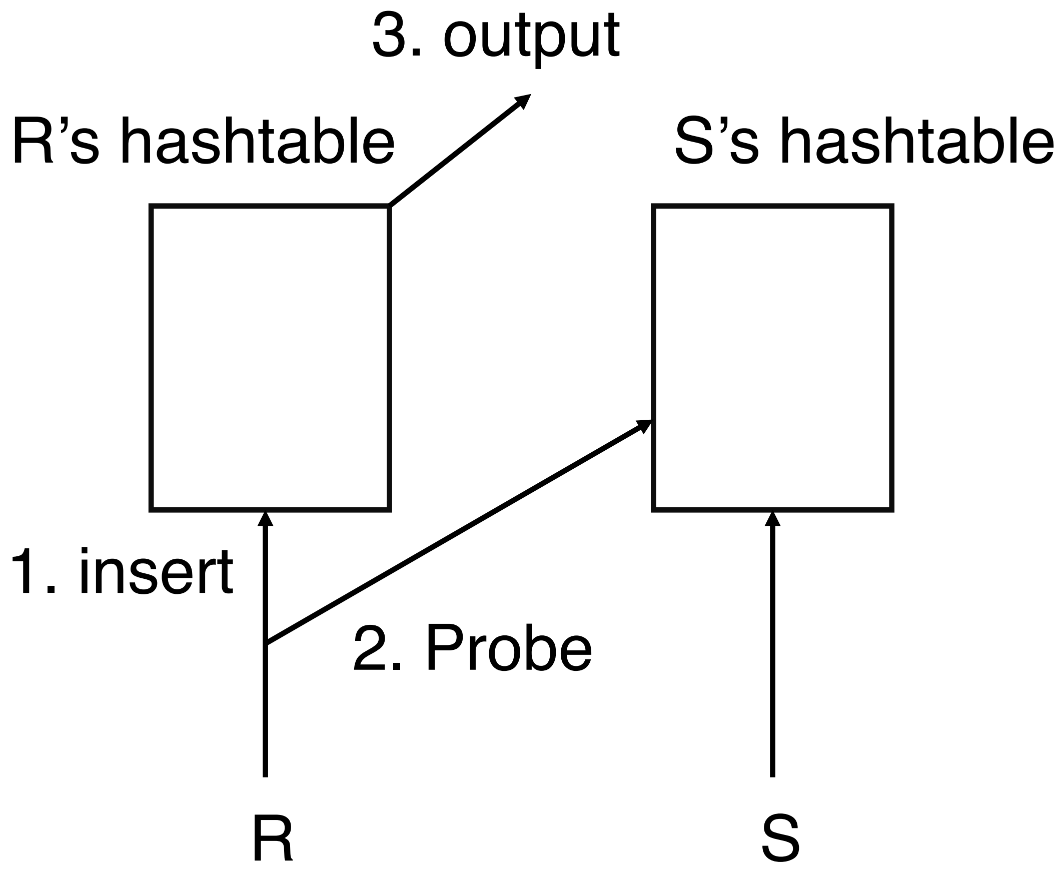
No Partitioning Hash Join

Partitioned Hash Join

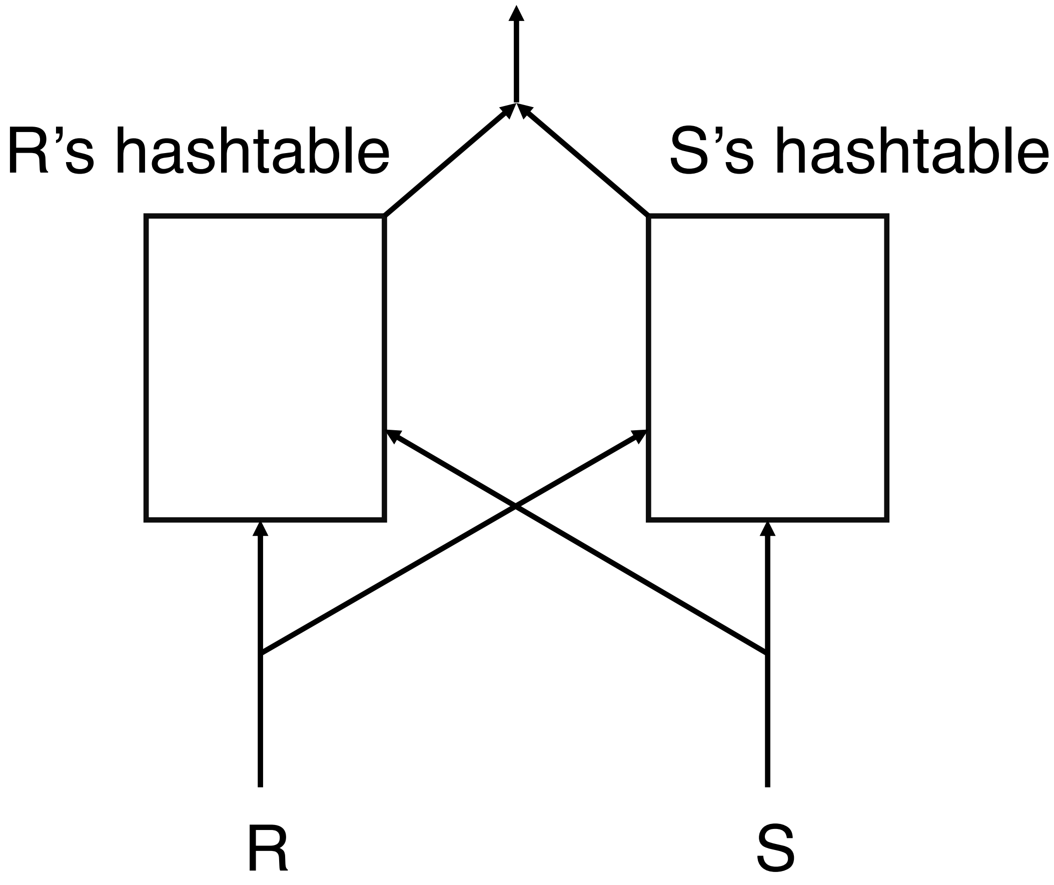
Parallel Radix Join

M-Way Sort-Merge Join

Symmetric Hash Join (SHJ)



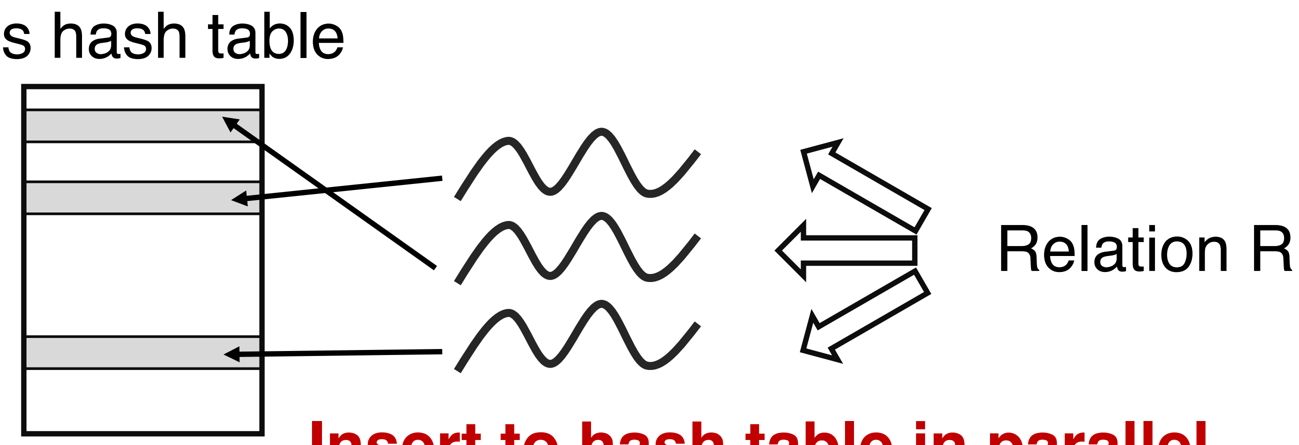
Symmetric Hash Join (SHJ)



- SHJ produces output tuples as early as possible
- If one relation arrives entirely before the other relation, SHJ degenerates to a simple hash join

No Partitioning Hash Join

- Build Phase

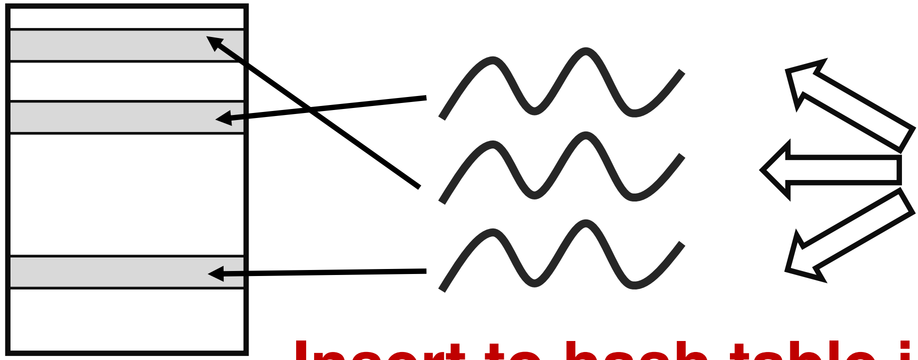


Need latches to protect concurrent inserts

Insert to hash table in parallel

No Partitioning Hash Join

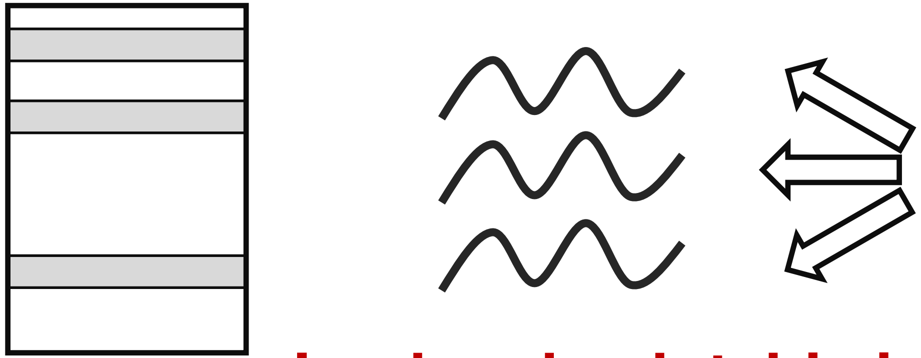
- Build Phase



Need latches to protect concurrent inserts

Insert to hash table in parallel

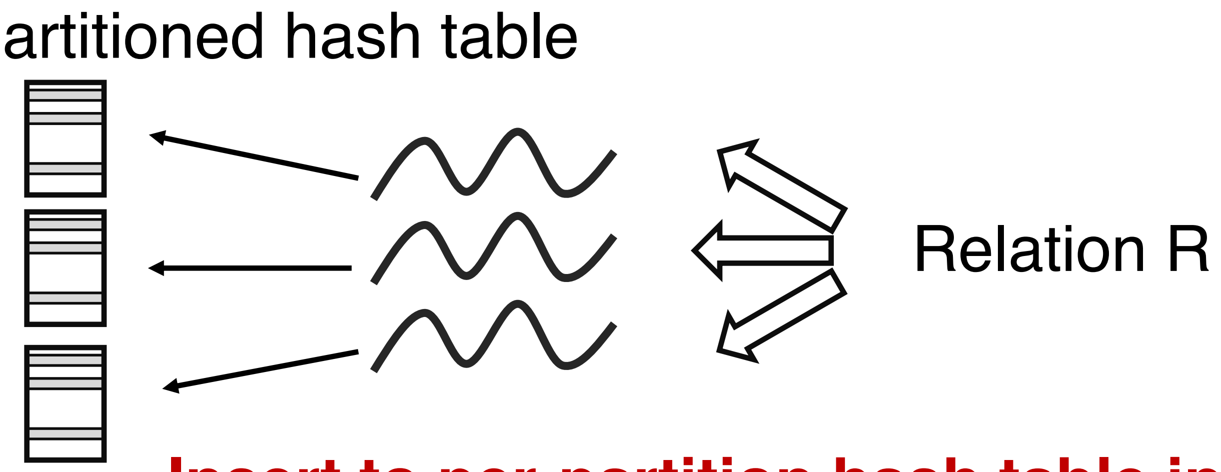
- Probe Phase



Lookup hash table in parallel

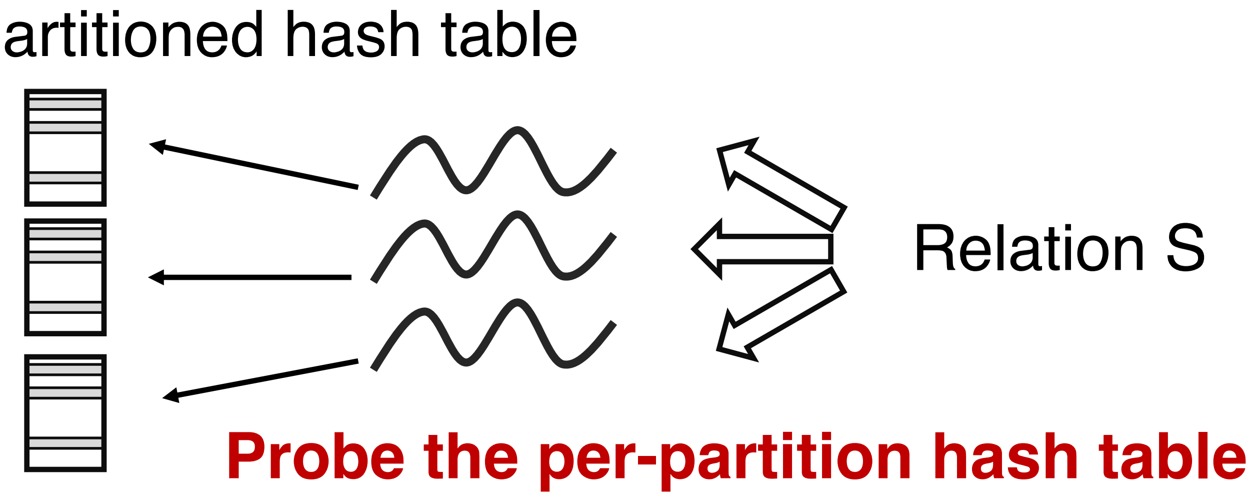
Partitioned Hash Join

- Build Phase



Insert to per-partition hash table in parallel

- Probe Phase



Probe the per-partition hash table

Radix Hash Join

Want a partition to fit in cache

$\Rightarrow$ Large number of partitions

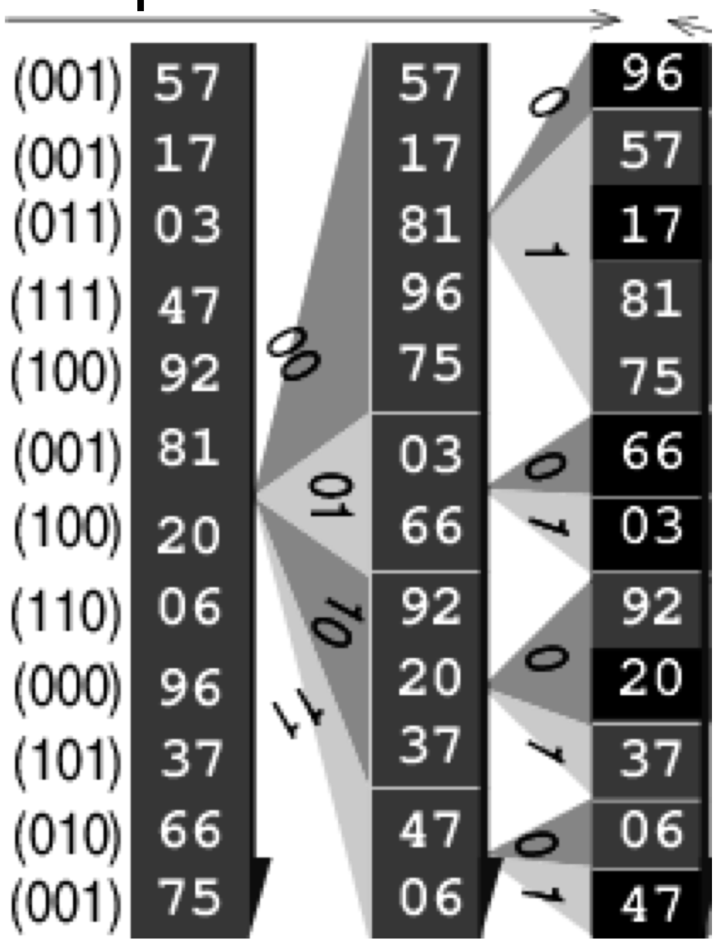
$\Rightarrow$ TLB misses

Radix Hash Join

Want a partition to fit in cache

⇒ Large number of partitions

⇒ TLB misses

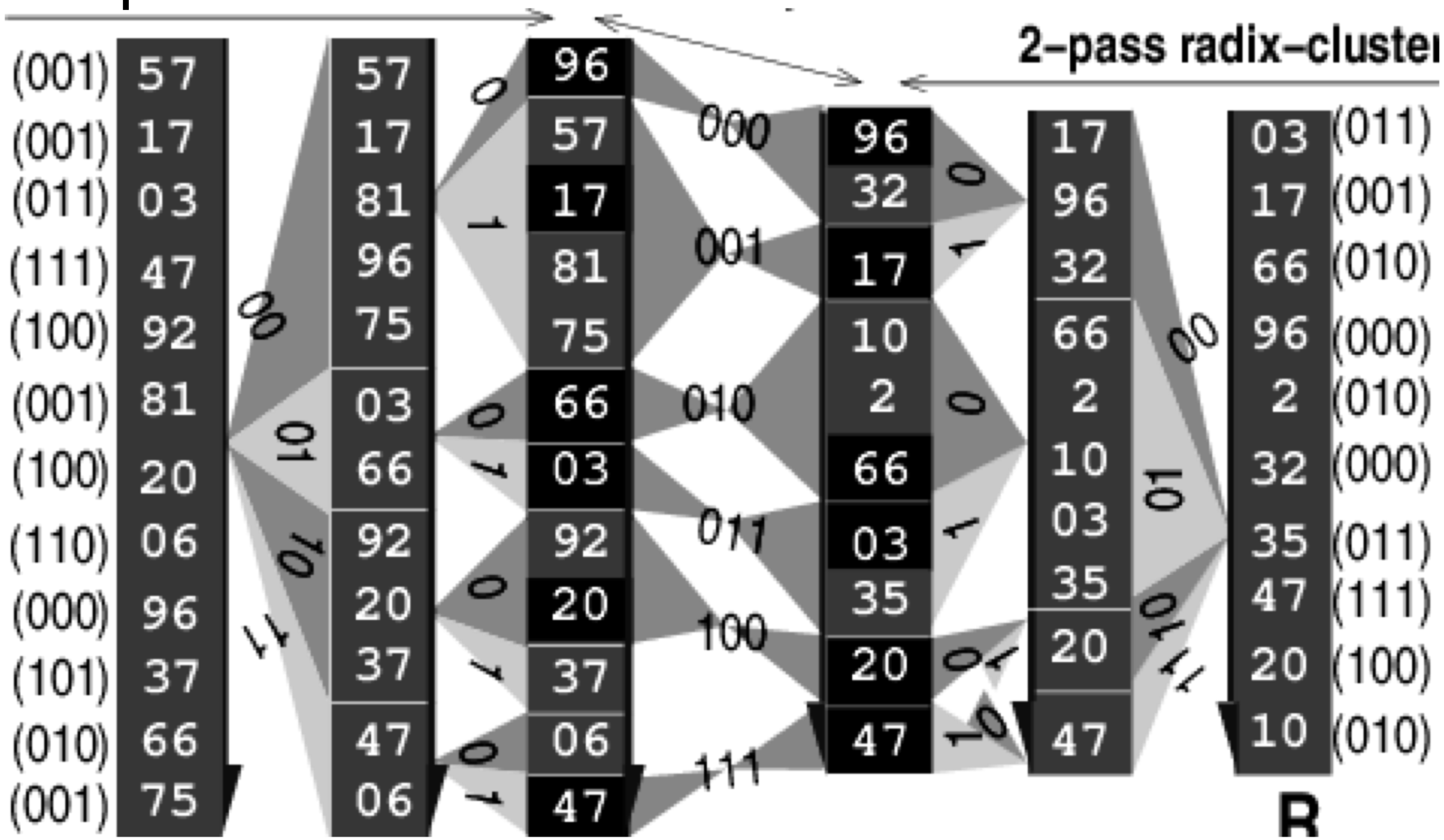


Radix Hash Join

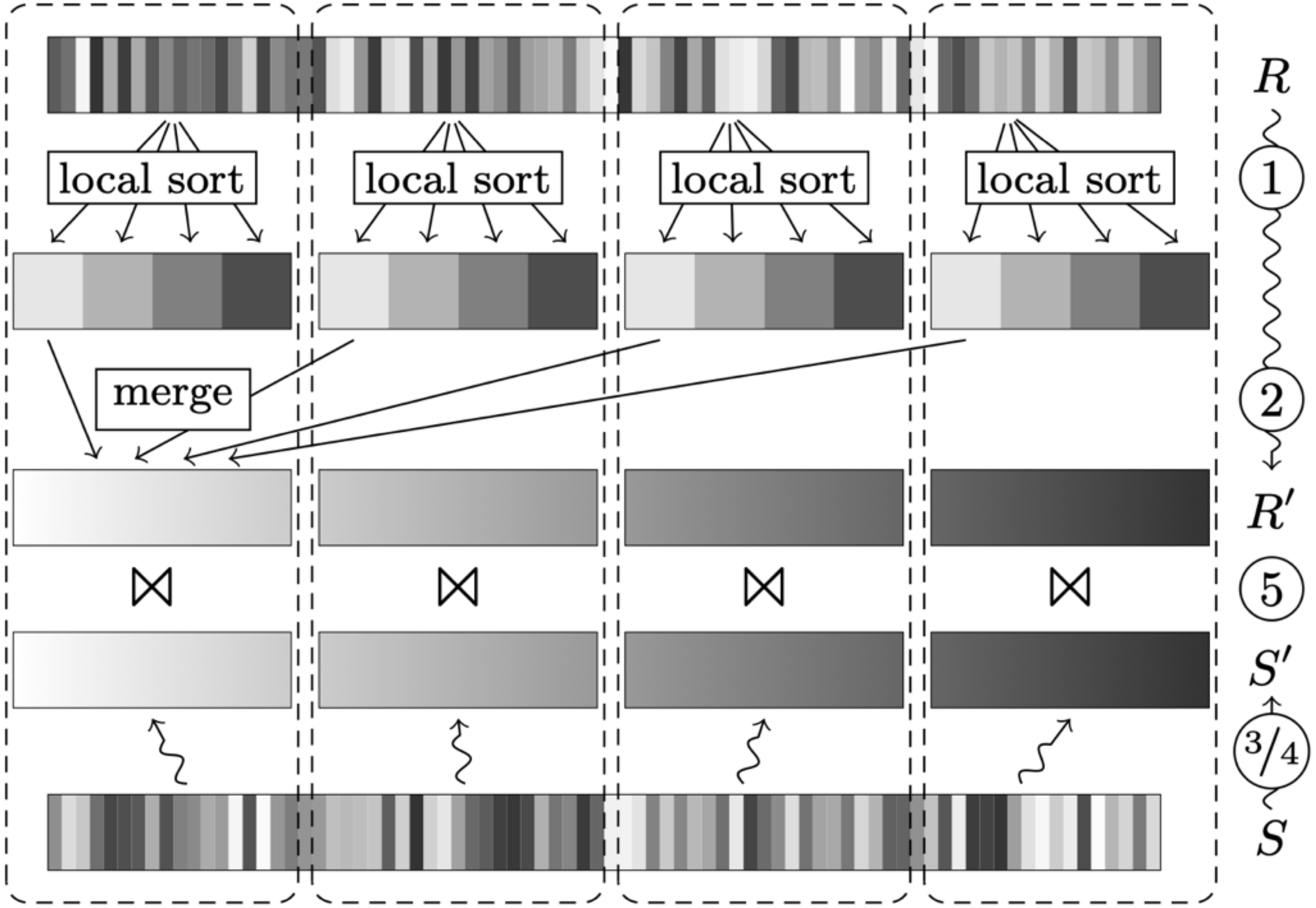
Want a partition to fit in cache

⇒ Large number of partitions

⇒ TLB misses



M-Way Sort-Merge Join

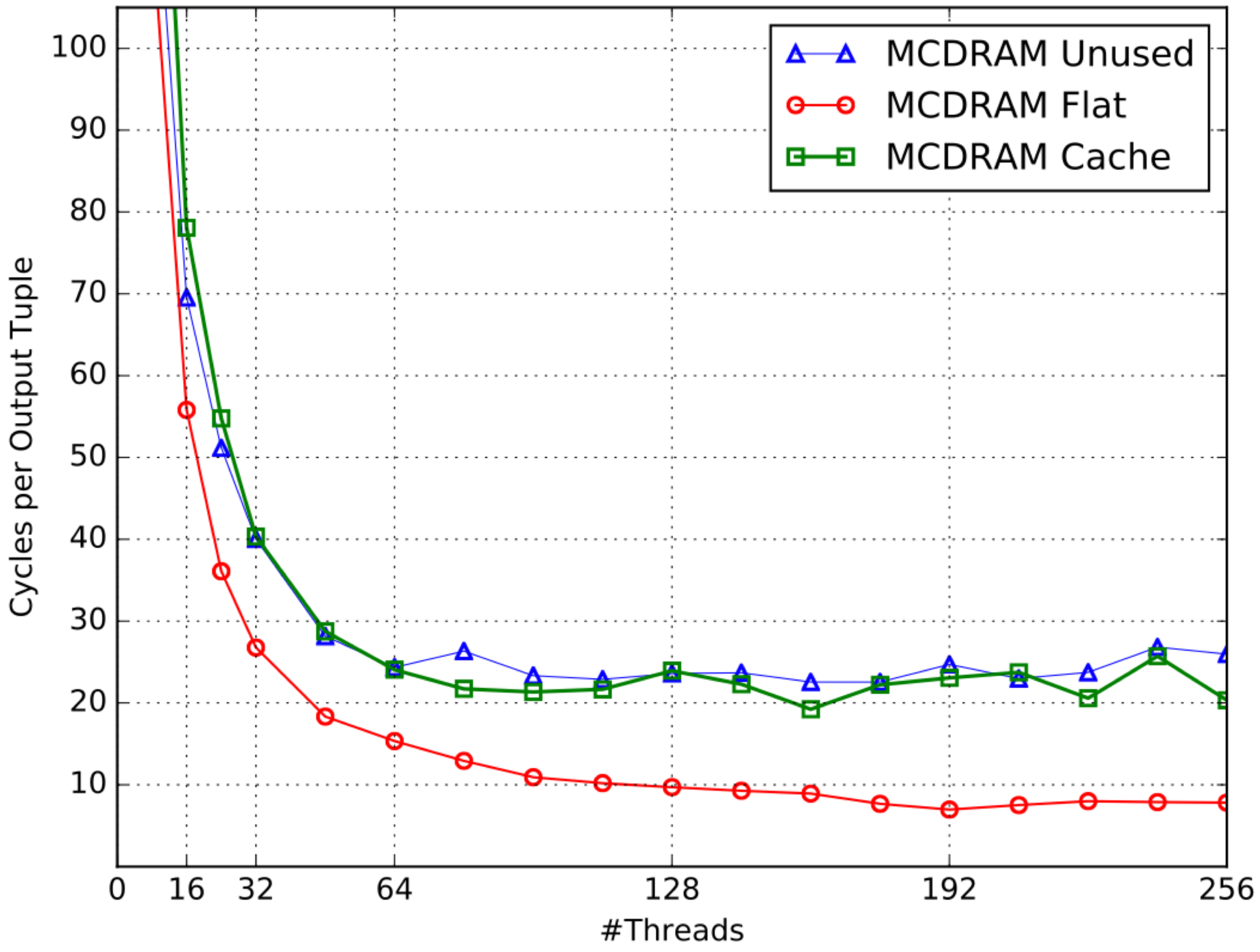


For both R and S

- Range-partition the local chunk of the input relation
- Sort local partitions one-by-one
- Merge partitions across threads

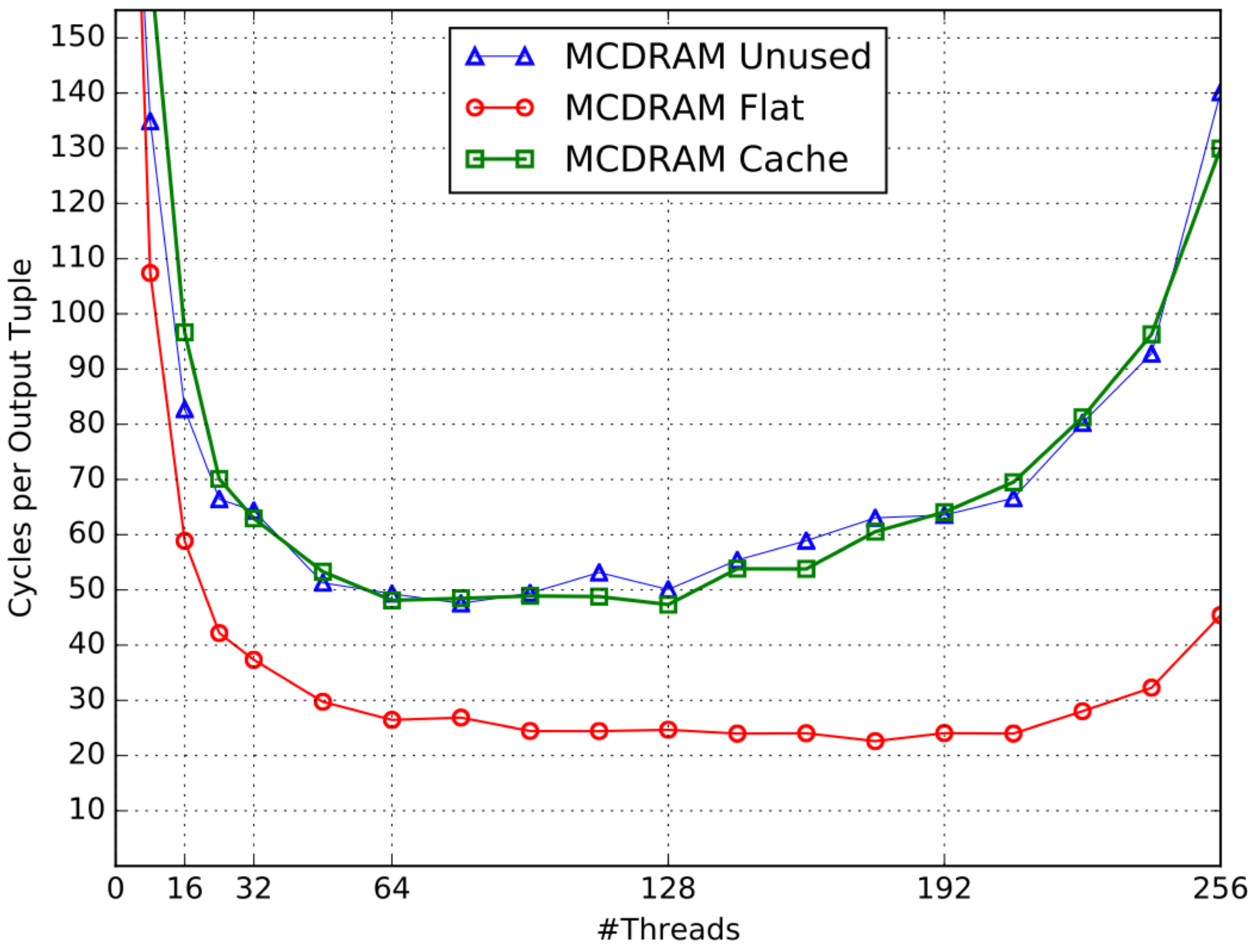
Merge sorted R and S through a linear scan

Evaluation – No Partitioning Join



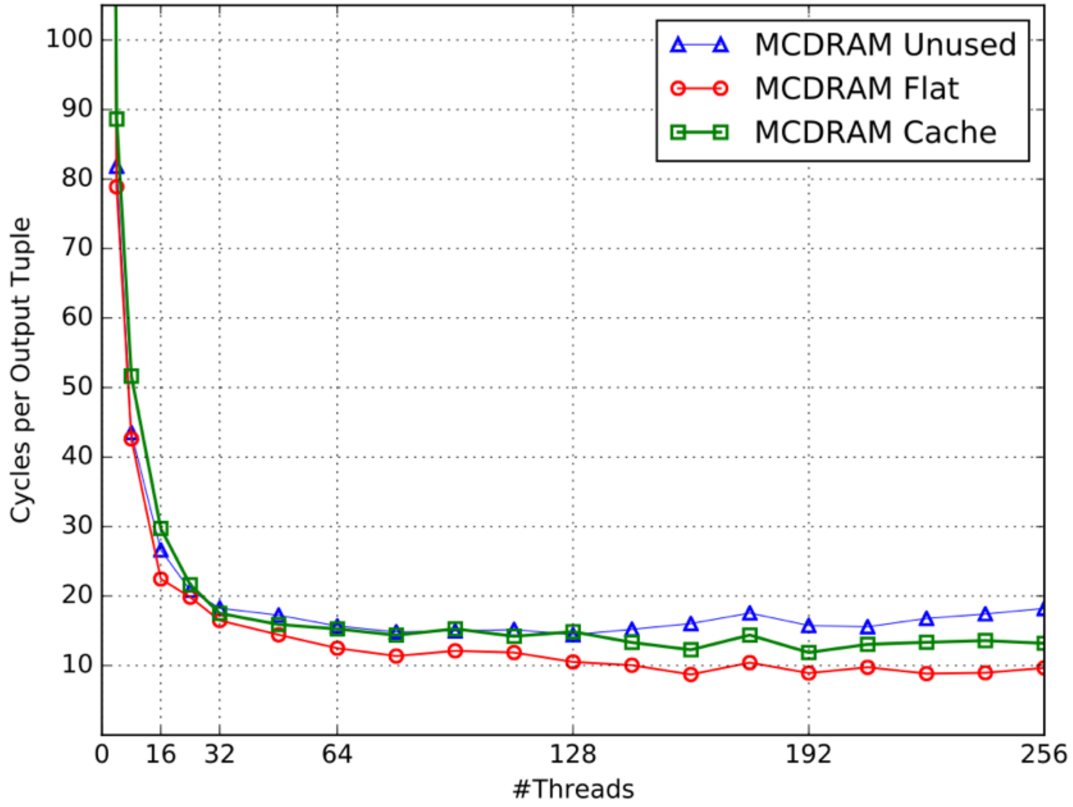
- Flat mode improves performance by 3.5x
- Cache mode does not improve performance

Evaluation – Parallel Hash Join

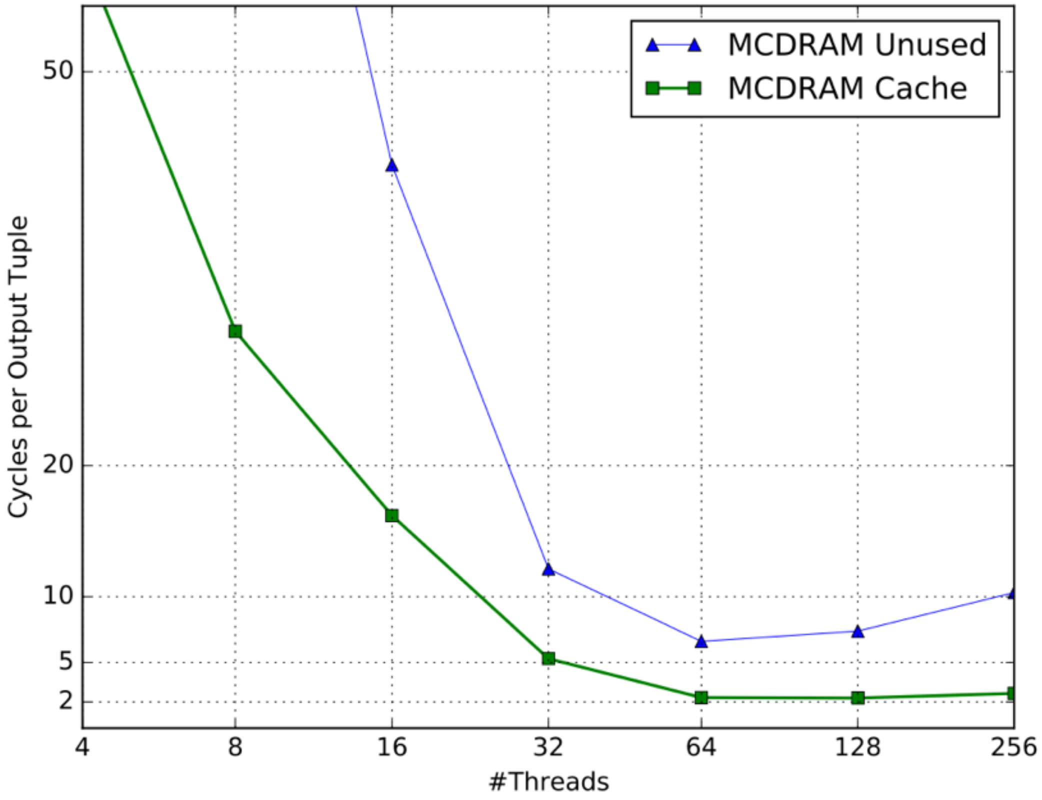


- Flat mode improves performance by 3.5x
- Cache mode does not improve performance
- Parallel hash join performs worse than no partitioning join

Evaluation – Parallel Radix Join



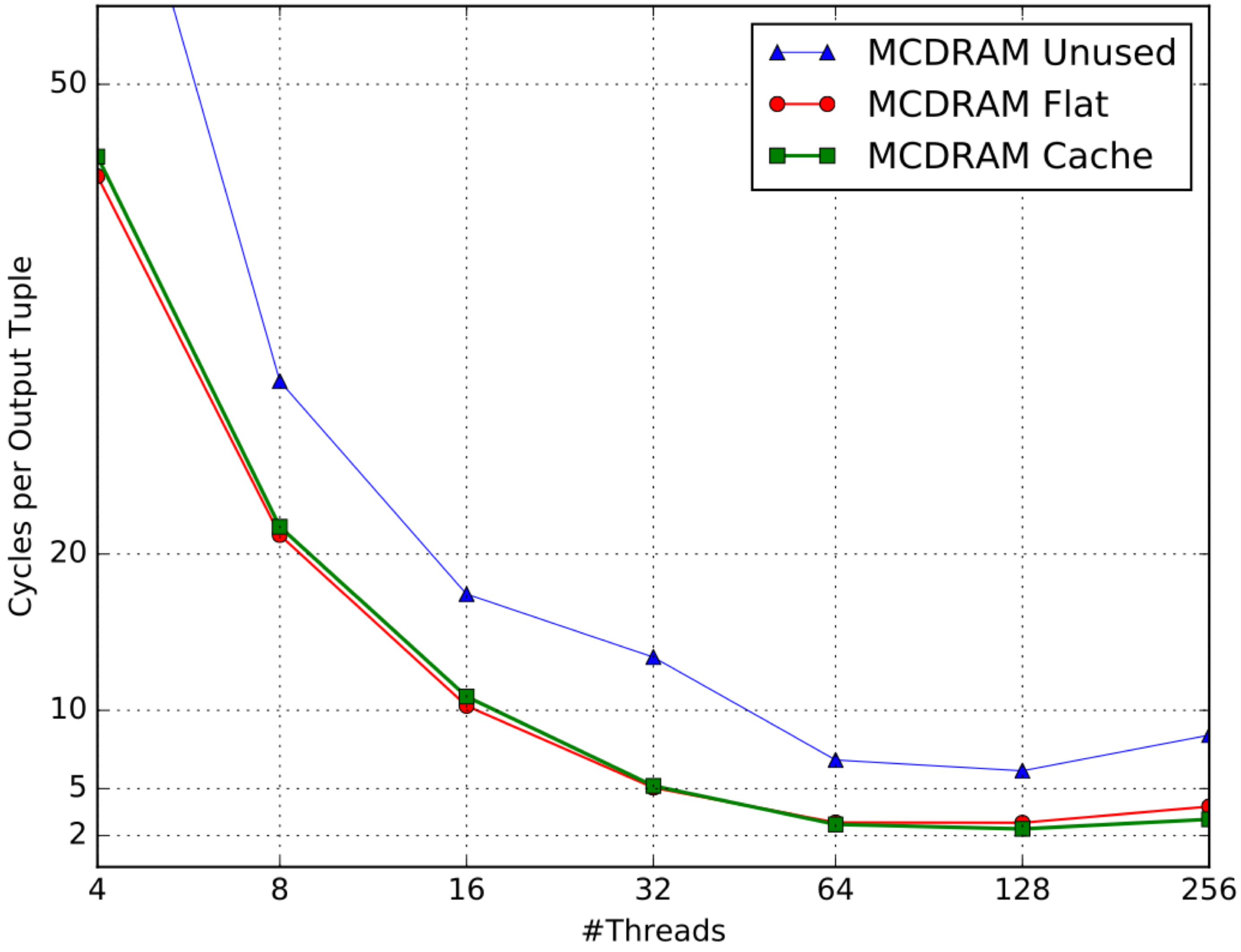
Implementation of [5]



Implementation of [3]

- Cache mode improves some performance
- Optimized Radix join is much faster than no partitioning join

Evaluation – Sort-Merge Join



Lessons Learned

1. Flat mode better than cache mode
2. High amounts of threads can easily lead to memory bottlenecks
3. Do not place everything in HBM
4. Random memory access patterns do not saturate bandwidth in general, being therefore not ideal for HBM
5. High bandwidth improves highly parallel hash joins and sort-merge joins more or less equally
6. Uneven load balancing by skew as well as latches on partitions are not noticeably influenced by HBM

Summary

Xeon Phi: High bandwidth memory integrated with multicore CPU

HBM is more popular with GPU than CPU

Cache mode does not always lead to performance improvement

NVM – Q/A

What is AVX?

What is non-uniform memory access (NUMA)

“Cycles per output tuple” and bandwidth are inversely correlated

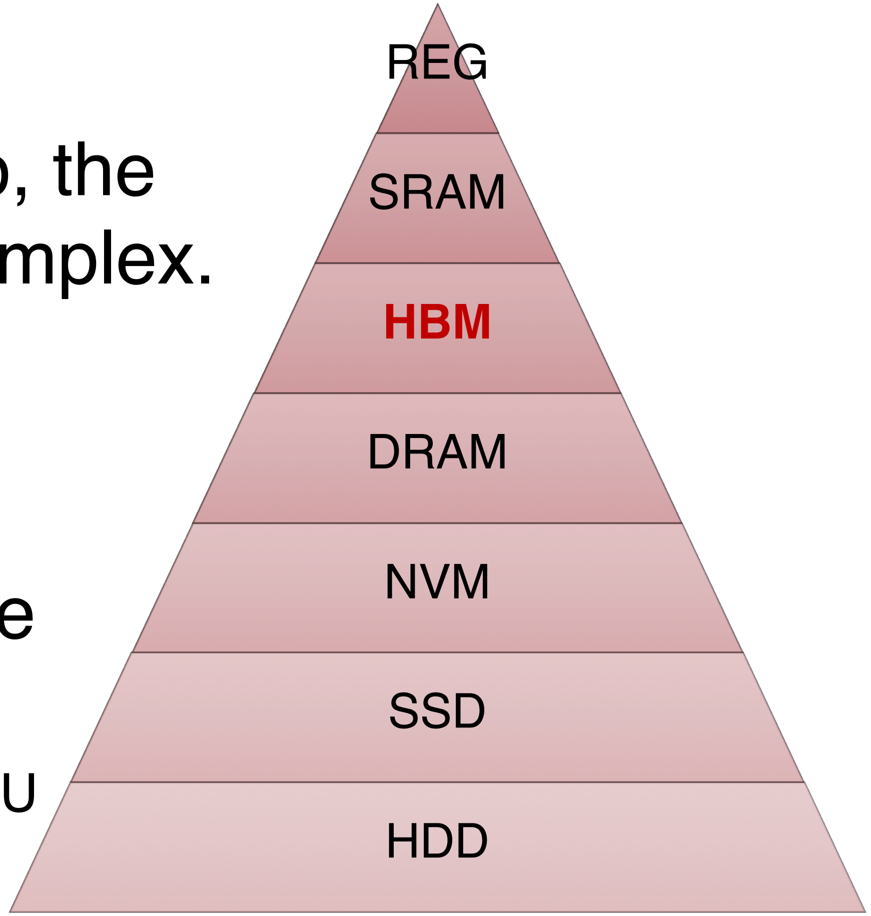
HBM + NVM?

Different DB variants for each type of data (e.g., relational data, graphs, streams etc.)

Group Discussion

What in your opinion makes HBM more successful with GPU than CPU?

With new memory/storage devices showing up, the storage hierarchy is getting more and more complex. How do you think the trend will continue?



Do you think APU can be a promising database accelerator in the future?

- Accelerated processing unit (APU) integrates CPU and GPU (and potentially HBM) on a single die

Before Next Lecture

Submit discussion summary to https://wisc-cs839-ngdb20.hotcrp.com

- **Deadline: Friday 11:59pm**

Submit review for

- Query Processing on Smart SSDs: Opportunities and Challenges
- [optional] Enabling Cost-effective Data Processing with Smart SSD