why I love memory latency

high throughput through latency masking hardware multithreading

WALID NAJJAR
UC RIVERSIDE

the team




the cheerleaders:

Walid A. Najjar, Vassilis J. Tsotras, Vagelis Papalexakis, Daniel Wong





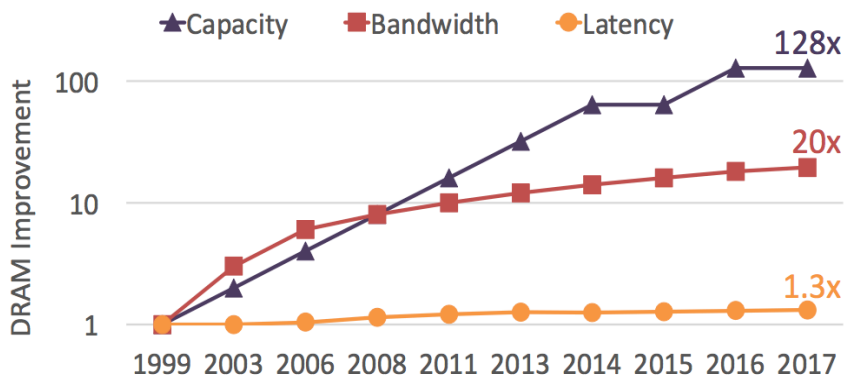

the players:

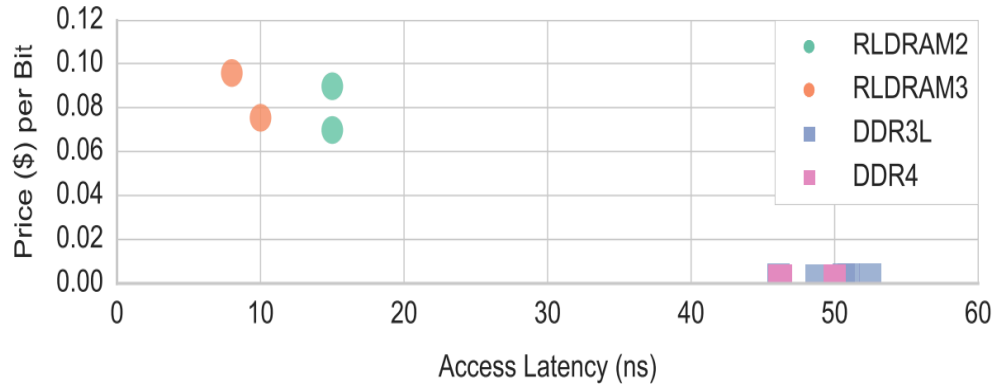
- Robert J. Halstead (Xilinx)
- Ildar Absalyamov (Google)
- Prerna Budhkar (Intel)
- Skyler Windh (Micron)
- Vassileos Zois (UCR)
- Bashar Roumanous (UCR)
- Mohammadreza Rezvani (UCR)

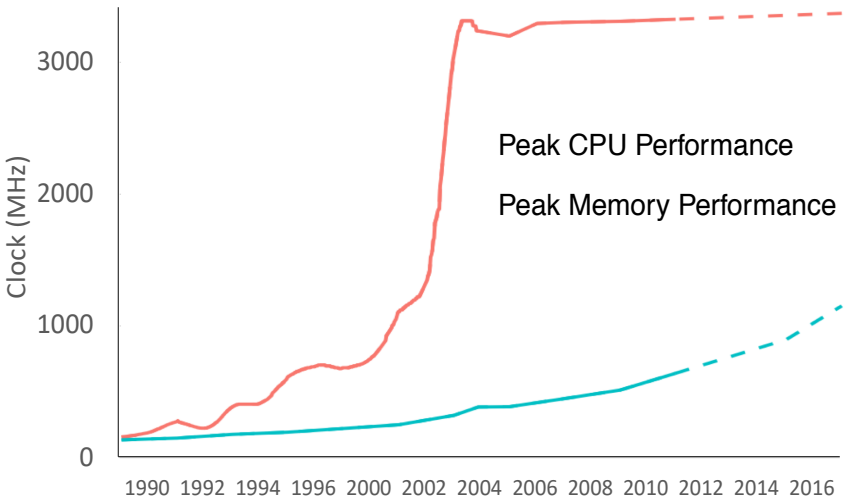
outline

- **background**
- filament execution model
- ~~application - SpMV~~
- application - Selection
- application - Group by aggregation
- conclusion

current DRAM trends





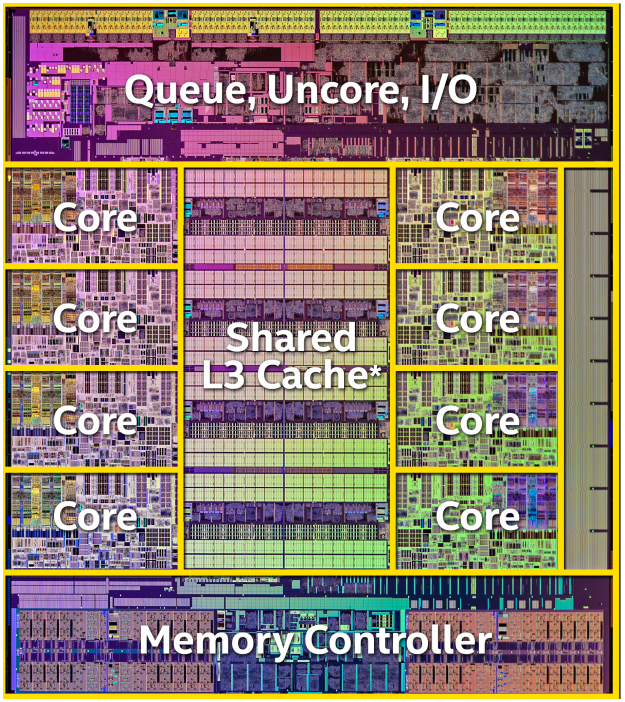


- *Growing memory capacity and low cost have created a niche for in-memory solutions.*
- *Low Latency memory hardware is more expensive (39x) and has lower capacity than commonly used DDRx DRAM.*
- *Though memory performance is improving, it is not likely to match CPU performance anytime soon*
- *Need alternative techniques to mitigate memory latency!*

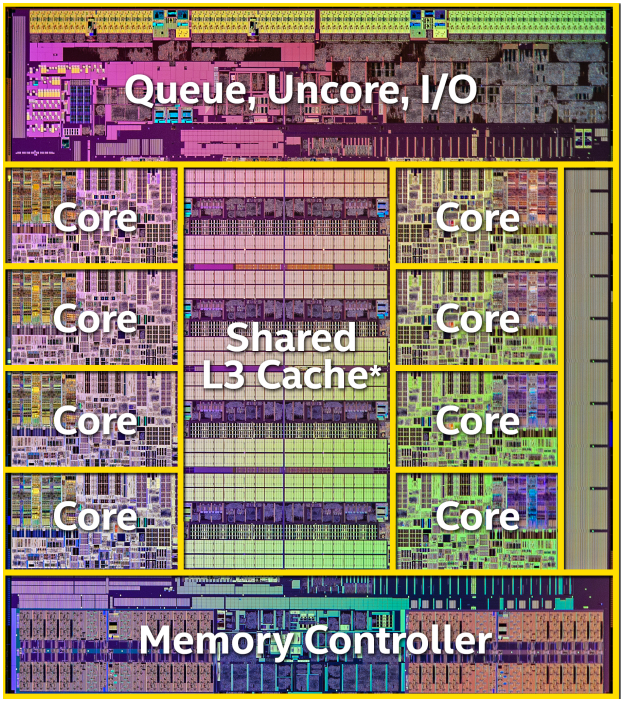
K. K. Chang. Understanding and Improving Latency of DRAM-Based Memory Systems. PhD thesis, Carnegie Mellon University, 2017.

latency mitigation - aka caching

latency mitigation - aka caching



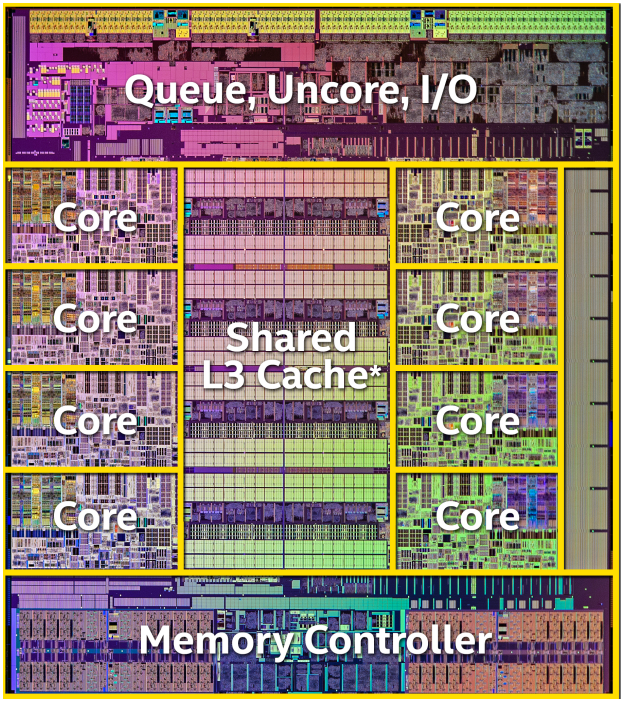
latency mitigation - aka caching



Intel Core-i7-5960x	
Launch Date	*Q3'14*
Clock Freq.	*3.0 GHz*
Cores / Threads	*8 / 16*
L3 Cache	*20 MB*
Memory Channels	*4*
Memory Bandwidth	*68 GB/s*

latency mitigation - aka caching

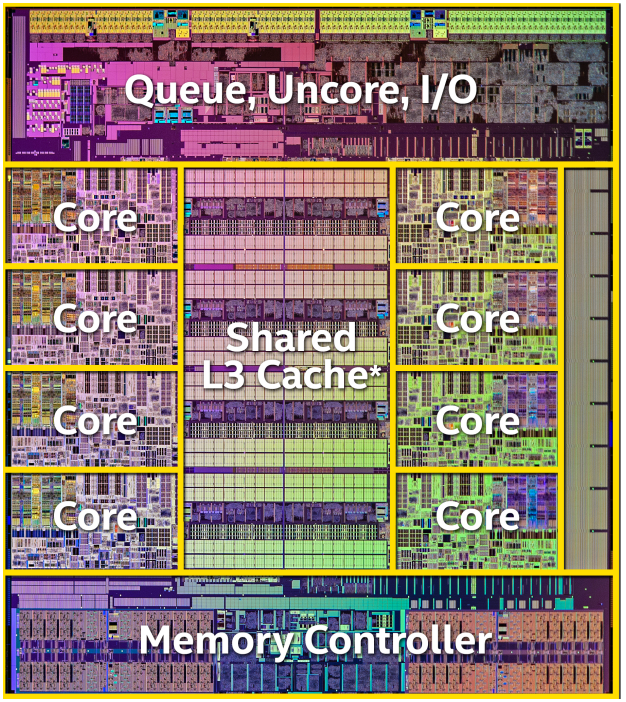
- **caches > 80% of CPU area**,
- big energy sink!
- application must have temporal and/ or spatial locality
- up to 7 levels storage hierarchy (register file, L1, L2, L3, memory, disk cache, disk)



Intel Core-i7-5960x	
Launch Date	*Q3'14*
Clock Freq.	*3.0 GHz*
Cores / Threads	*8 / 16*
L3 Cache	*20 MB*
Memory Channels	*4*
Memory Bandwidth	*68 GB/s*

latency mitigation - aka caching

- **caches > 80% of CPU area**,
- big energy sink!
- application must have temporal and/ or spatial locality
- up to 7 levels storage hierarchy (register file, L1, L2, L3, memory, disk cache, disk)



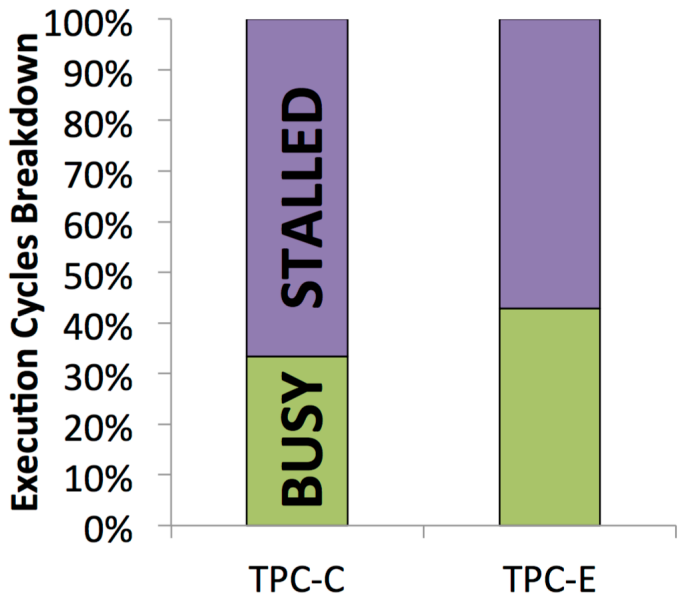
Intel Core-i7-5960x	
Launch Date	*Q3'14*
Clock Freq.	*3.0 GHz*
Cores / Threads	*8 / 16*
L3 Cache	*20 MB*
Memory Channels	*4*
Memory Bandwidth	*68 GB/s*

caches everywhere! reduces **average** memory latency

assumes locality

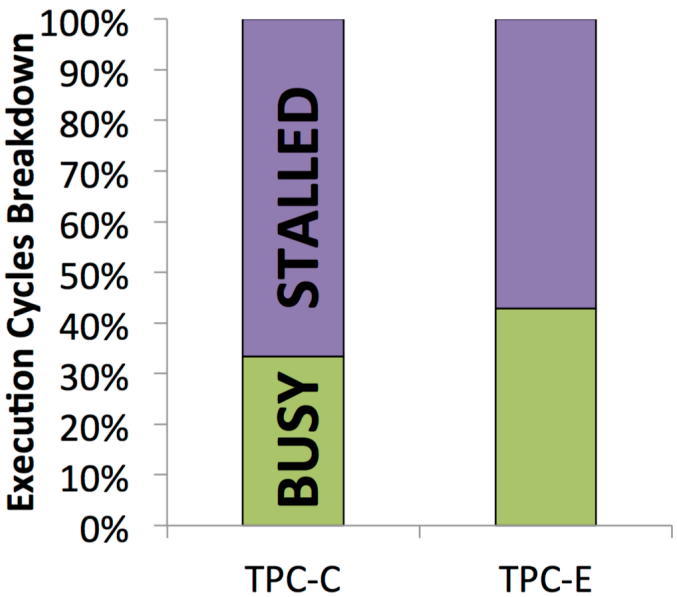
where cycles go - cache misses

where cycles go - cache misses

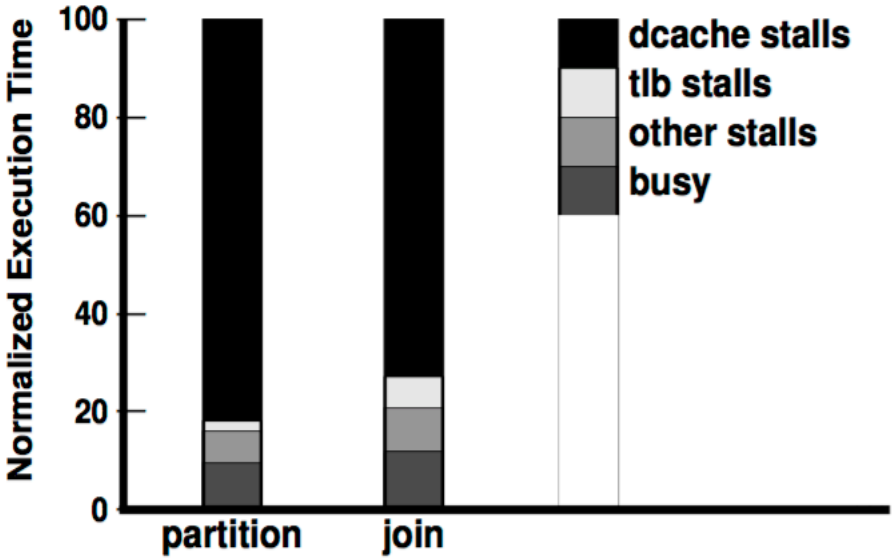


[DaMoN'13] Tözün, P. et al. OLTP in wonderland: where do cache misses come from in major OLTP components?

where cycles go - cache misses



[DaMoN'13] Tözün, P. et al. OLTP in wonderland: where do cache misses come from in major OLTP components?



[TODS'07] Chen, S. et al. Improving Hash Join Performance through Prefetching

big data analytics

big data analytics

BIG data
(x petaB)

big data analytics

BIG data
(x petaB)

+

big data analytics

BIG data
(x petaB)

+

hyper
dimensional

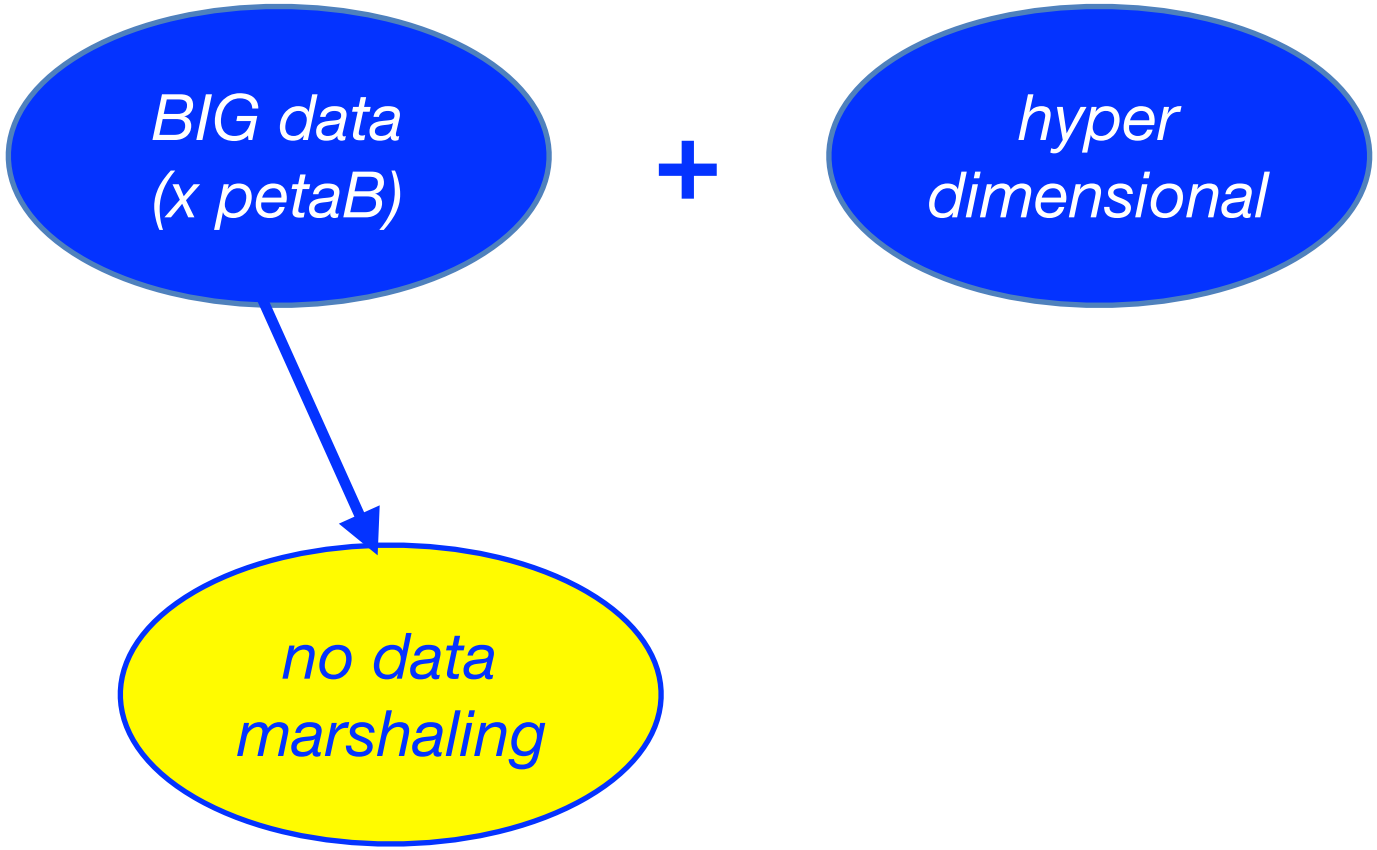
big data analytics

BIG data
(x petaB)

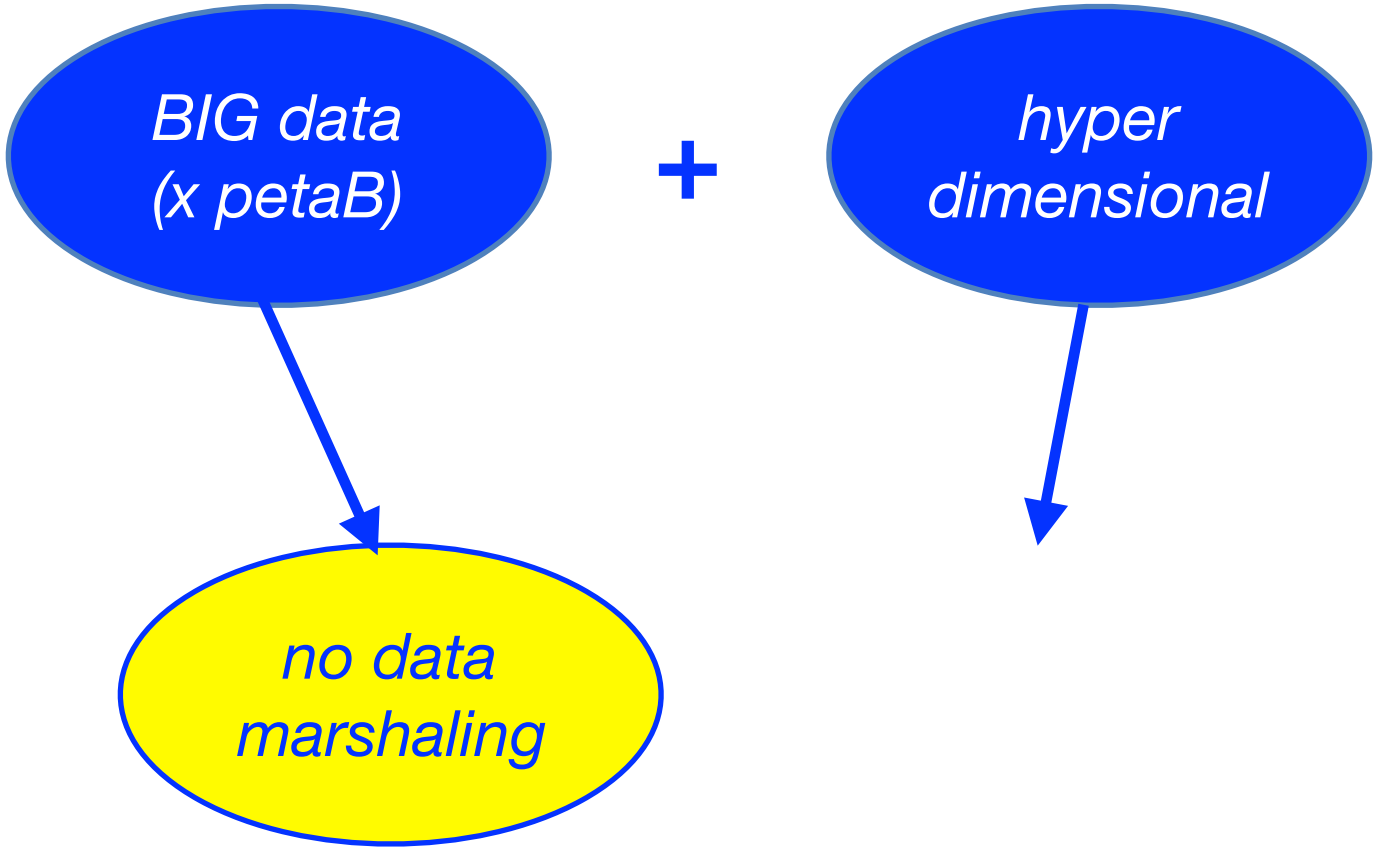
+

hyper
dimensional

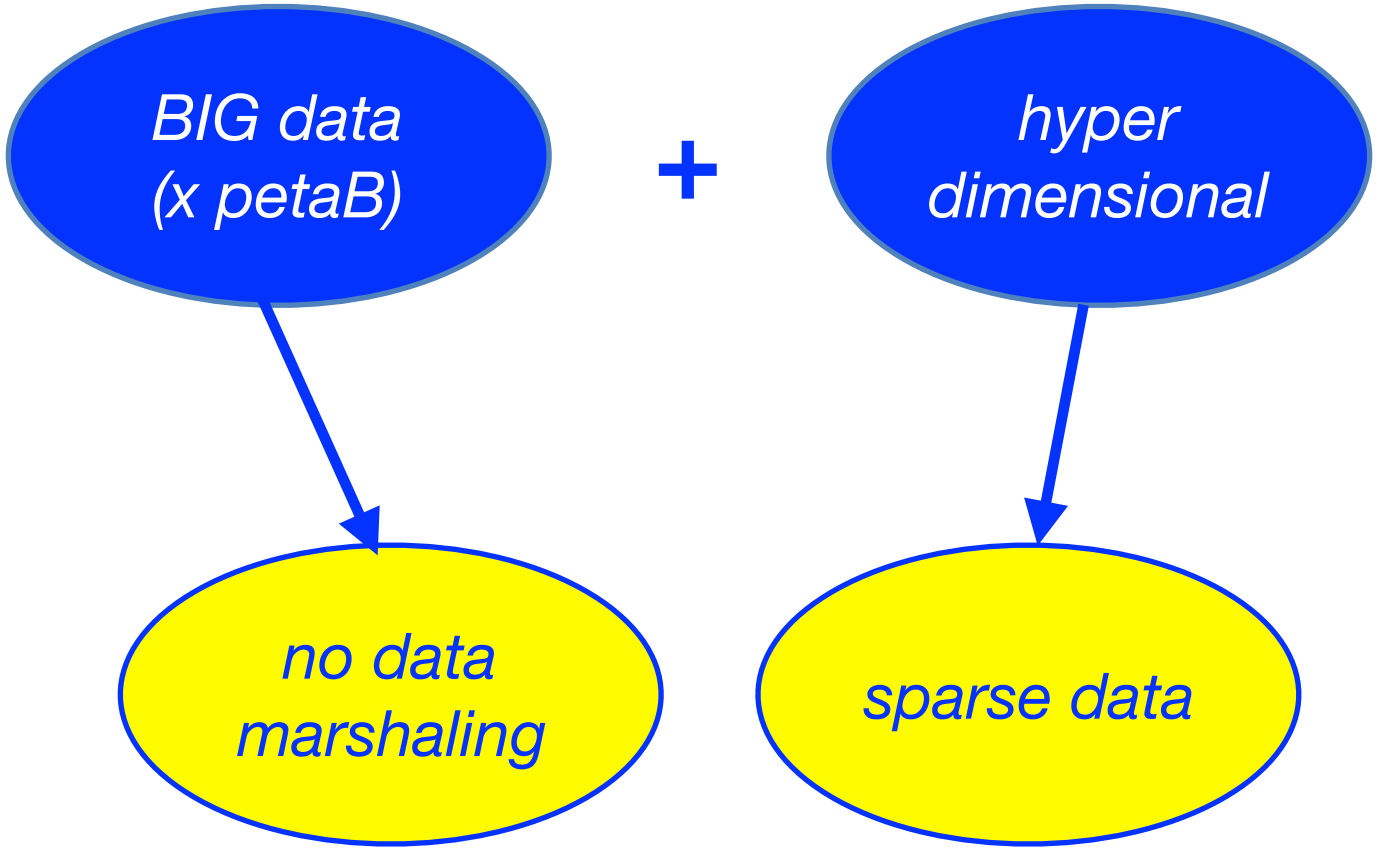
big data analytics



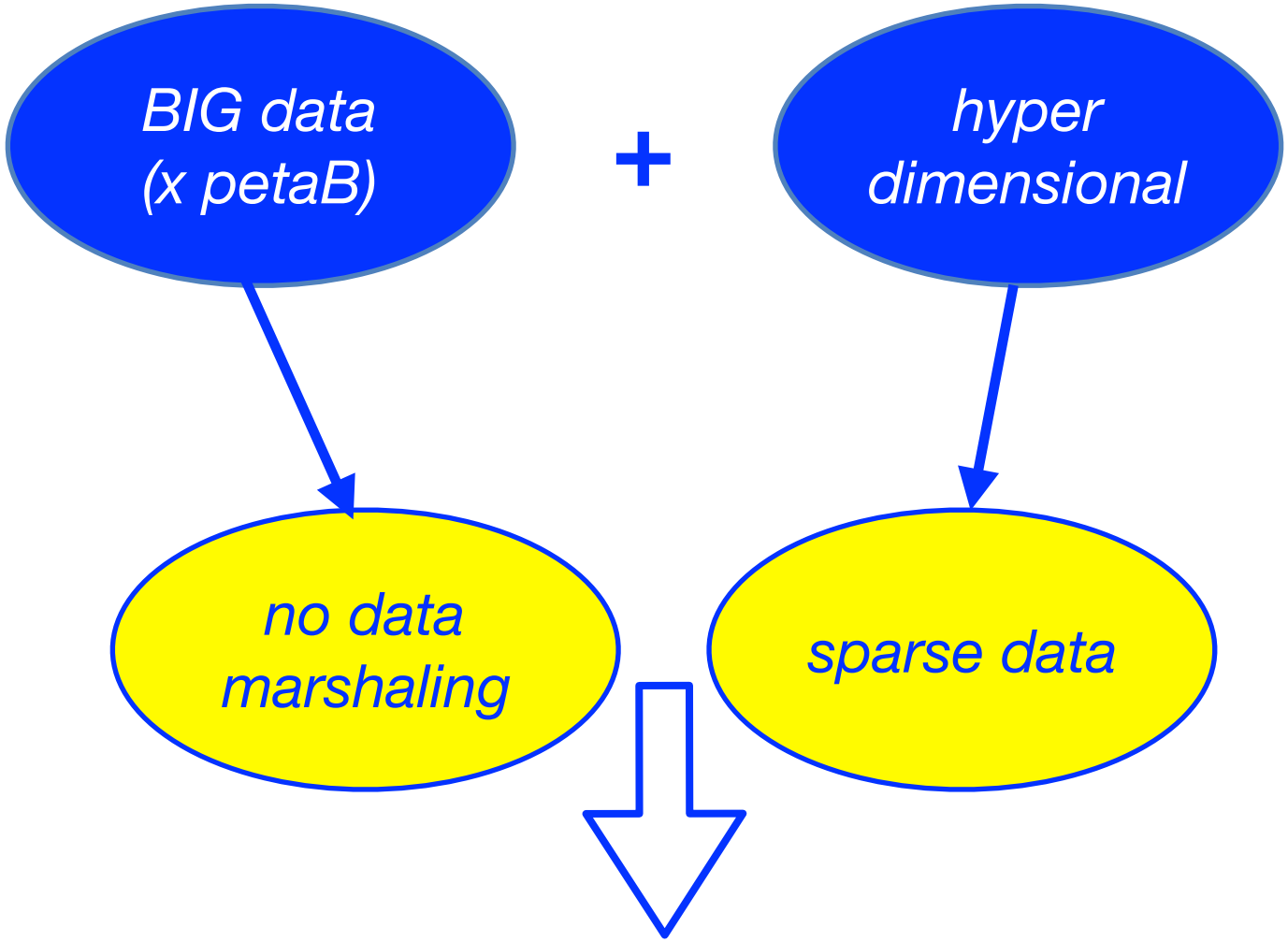
big data analytics



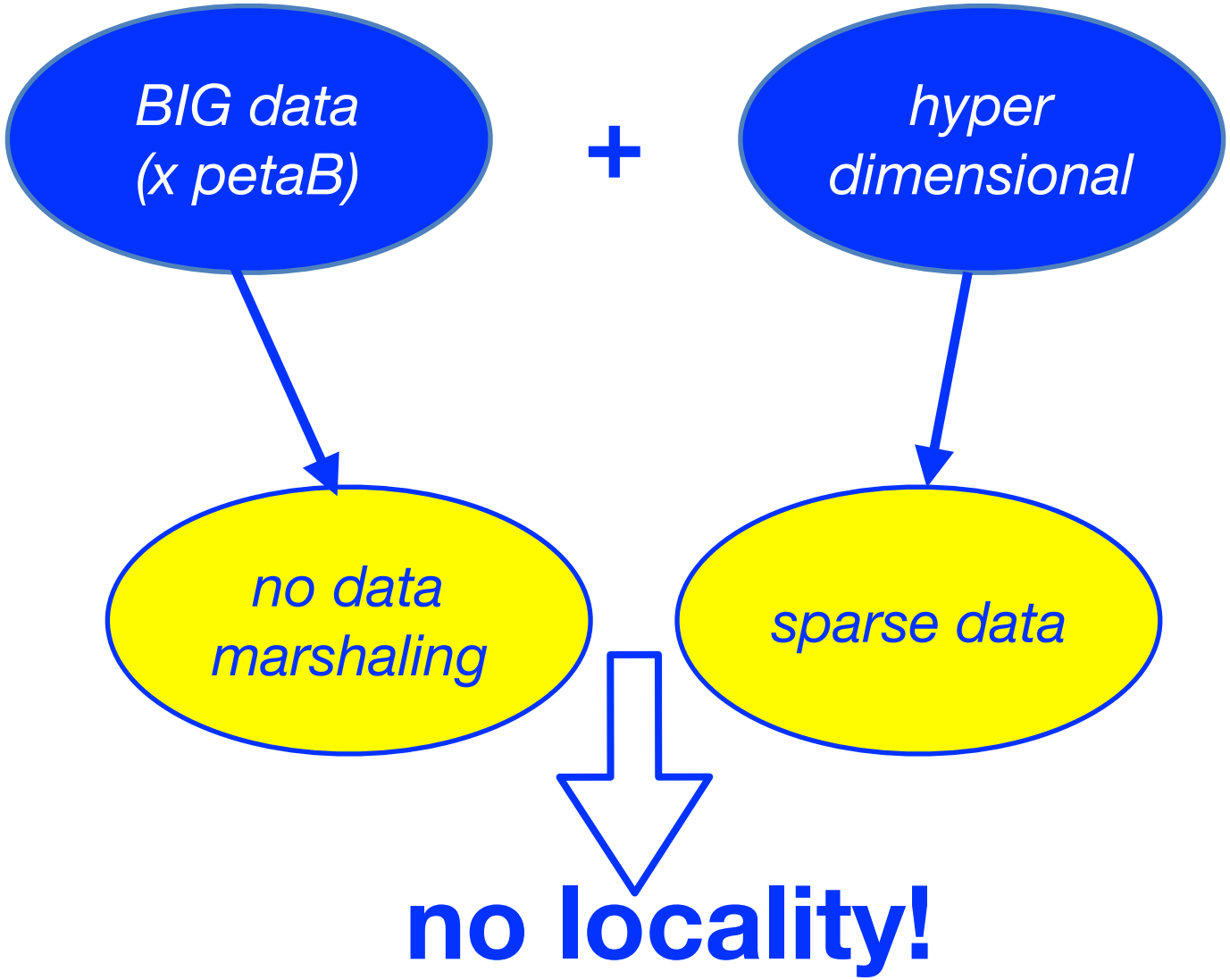
big data analytics



big data analytics



big data analytics



You have to know the past to understand the present

- Carl Sagan

the latency problem & solutions

- latency mitigations, aka "caching"
  - effective only with spatial/temporal localities
- streaming: latency incorporated into pipeline
  - requires spatial locality
- latency masking, aka hardware multithreading
  - fast context switch after memory access
  - small thread context
  - concurrency == outstanding memory requests

Burton Smith (1941 - 2018)

Burton Smith (1941 - 2018)

Burton Smith (1941 - 2018)

- main architect of
  - Denelcor HEP (1982)
  - Horizon (1985-88) @ IDA SRC
  - Tera MTA (1999) later Cray XMT

Burton Smith (1941 - 2018)

- main architect of
  - Denelcor HEP (1982)
  - Horizon (1985-88) @ IDA SRC
  - Tera MTA (1999) later Cray XMT

Burton Smith (1941 - 2018)

- main architect of
  - Denelcor HEP (1982)
  - Horizon (1985-88) @ IDA SRC
  - Tera MTA (1999) later Cray XMT

- latency masking
  - = do useful work while waiting for memory = <u>multithreading</u>
  - switch to a ready process on long-latency I/O operations
  - => higher throughput
  - 50 years ago it was I/O latency => multi-programming

Burton Smith (1941 - 2018)

- main architect of
  - Denelcor HEP (1982)
  - Horizon (1985-88) @ IDA SRC
  - Tera MTA (1999) later Cray XMT

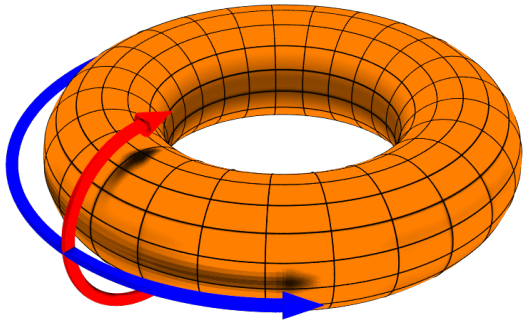
- latency masking
  - = do useful work while waiting for memory = <u>multithreading</u>
  - switch to a ready process on long-latency I/O operations
  - => higher throughput
  - 50 years ago it was I/O latency => multi-programming

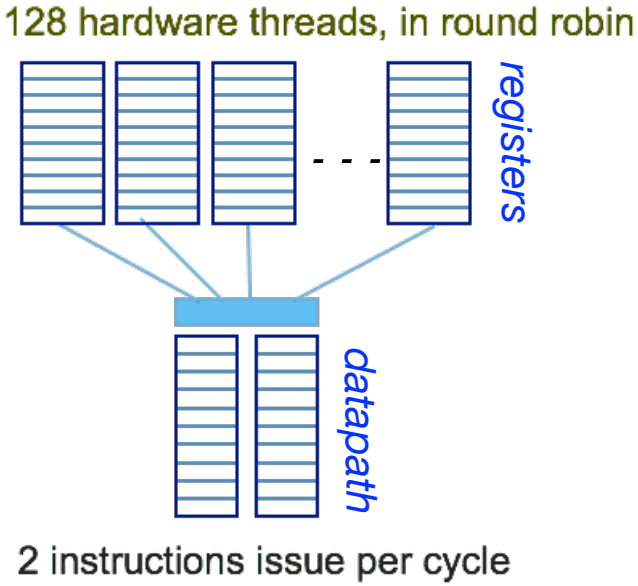
Burton Smith (1941 - 2018)

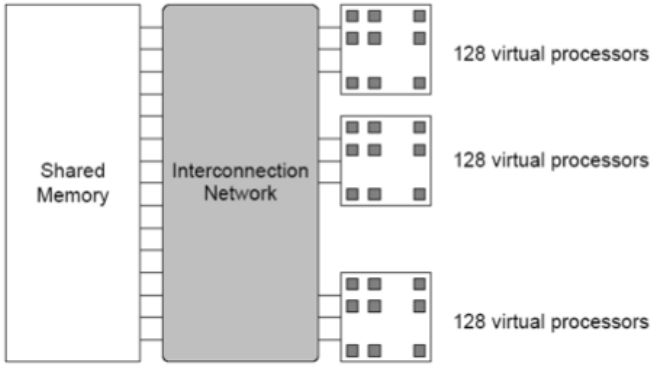
- main architect of
  - Denelcor HEP (1982)
  - Horizon (1985-88) @ IDA SRC
  - Tera MTA (1999) later Cray XMT
- latency masking
  - = do useful work while waiting for memory = <u>multithreading</u>
  - switch to a ready process on long-latency I/O operations
  - => higher throughput
  - 50 years ago it was I/O latency => multi-programming
- https://www.microsoft.com/en-us/research/people/burtons/

Tera MTA/Cray XMT

- supercomputer, barrel processor
- 128 independent threads/processor
- 2 inst/cycle issued: 1 ALU, 1 memory
- 1 full/empty bit per word insures correct ordering of memory accesses
- longest memory latency: 128 cycles
- randomized memory: reduces bank conflicts
- processor & memory nodes on a 3D torus interconnection network
- 4096 nodes: sparsely populated

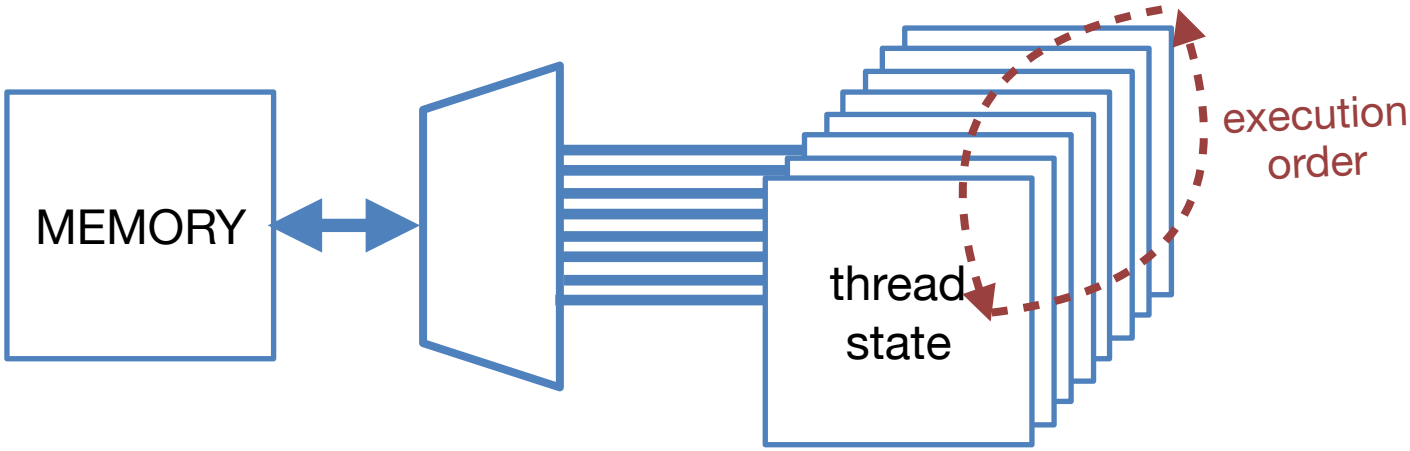






barrel processor

- CPU that switches between threads of execution on <u>every cycle</u>.
  - also known as "interleaved" or "fine-grained" temporal multithreading.
  - each thread is assigned its own program counter and other hardware registers (architectural state).
  - guarantee each thread will execute one instruction every n cycles.
  - n-way barrel processor acts like n separate processors,
    - each running at ~ 1/n the original speed.



outline

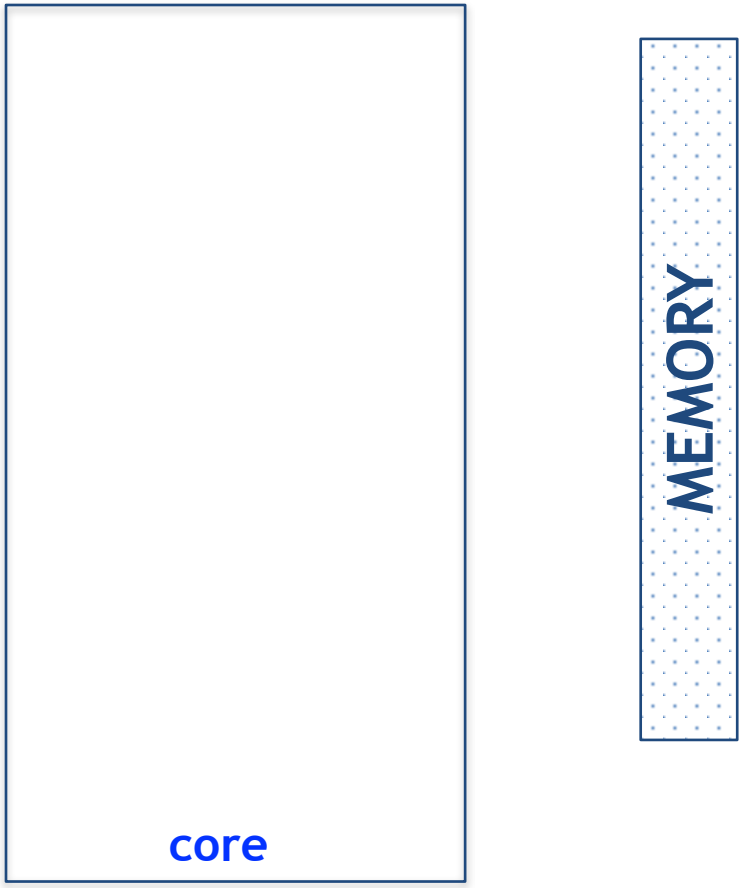
- background
- **filament execution model**
- ~~application - SpMV~~
- application - Selection
- application - Group by aggregation
- conclusion

what if we take it a step further?

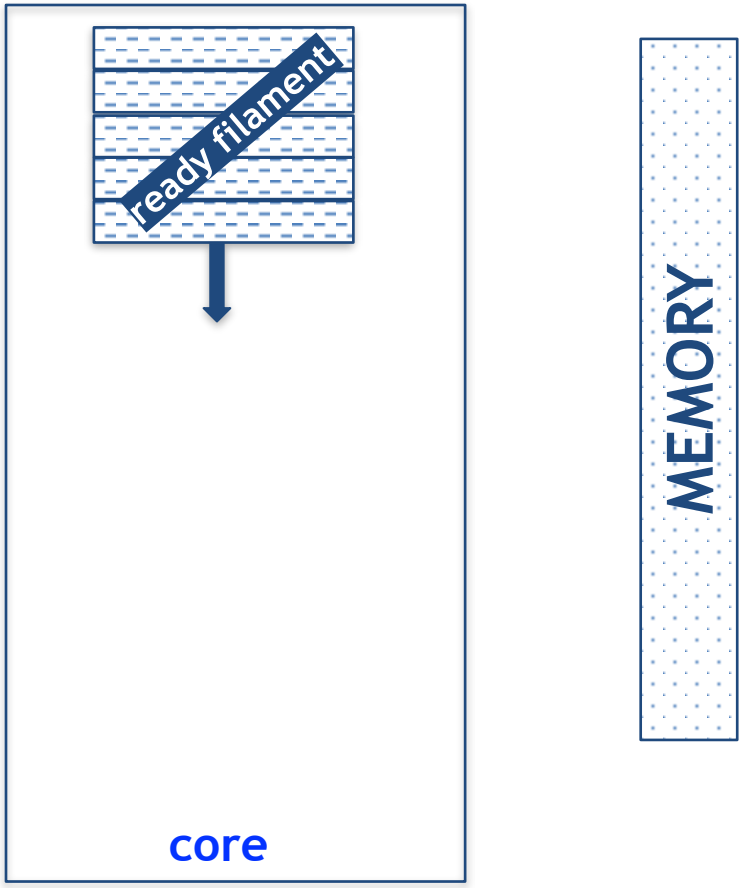
- trade bandwidth for latency
- custom or semi custom processor/accelerator
- no need for large register files and thread state
  - smaller data path
  - more processors
  - more threads per processor (engine or core)
  - much higher throughput!
- suited for large scale irregular applications:
  - data analytics, databases, sparse linear algebra: SpMV, SpMM, graph algorithms, bioinformatics

hardware multithreading

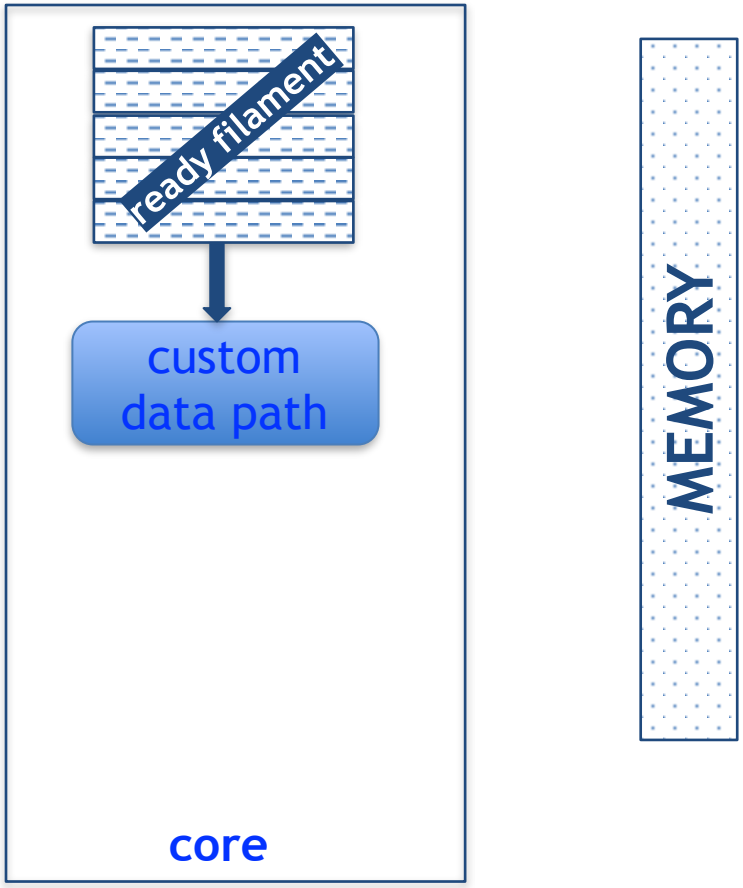
hardware multithreading



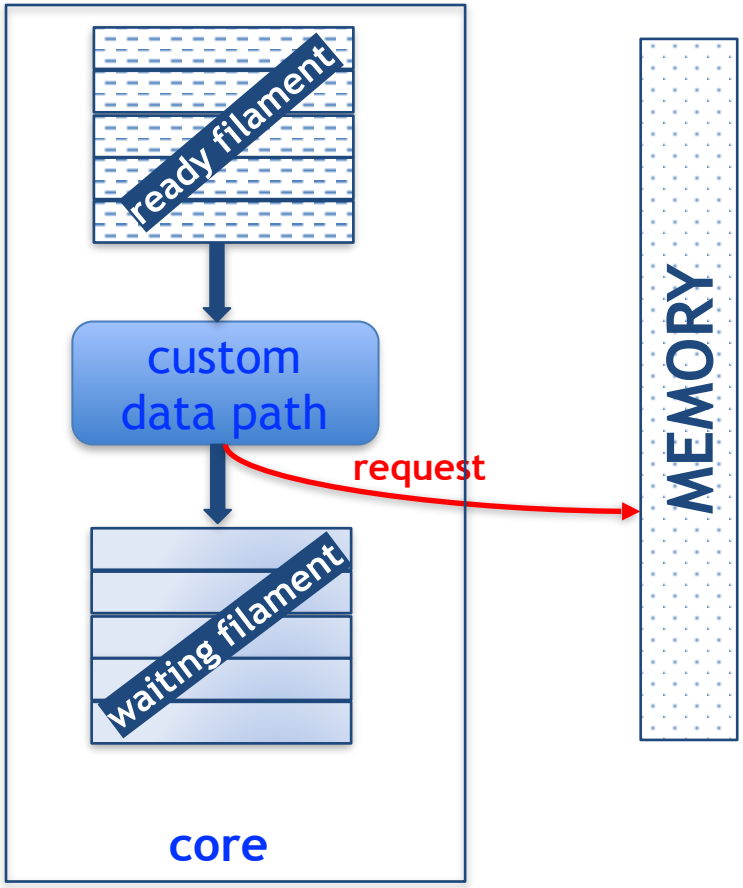
hardware multithreading



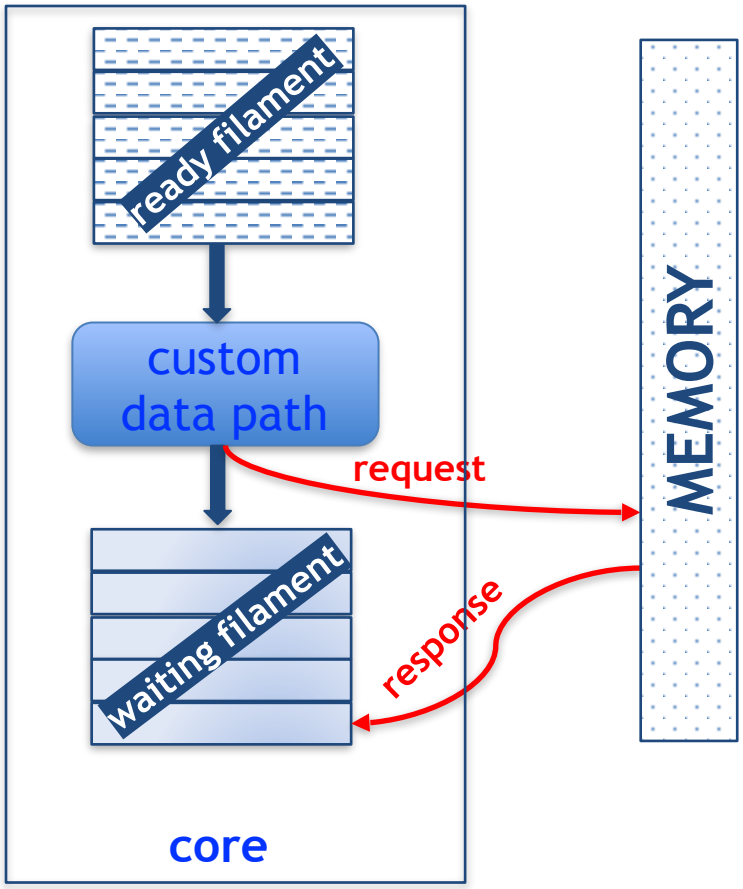
hardware multithreading



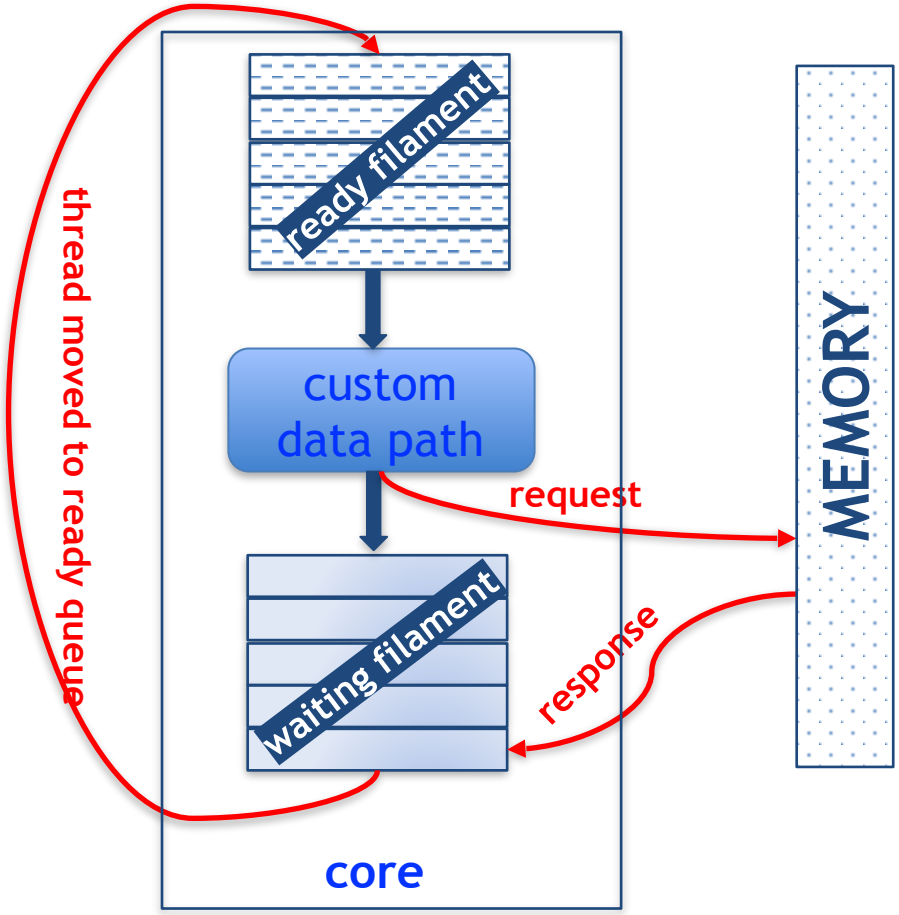
hardware multithreading



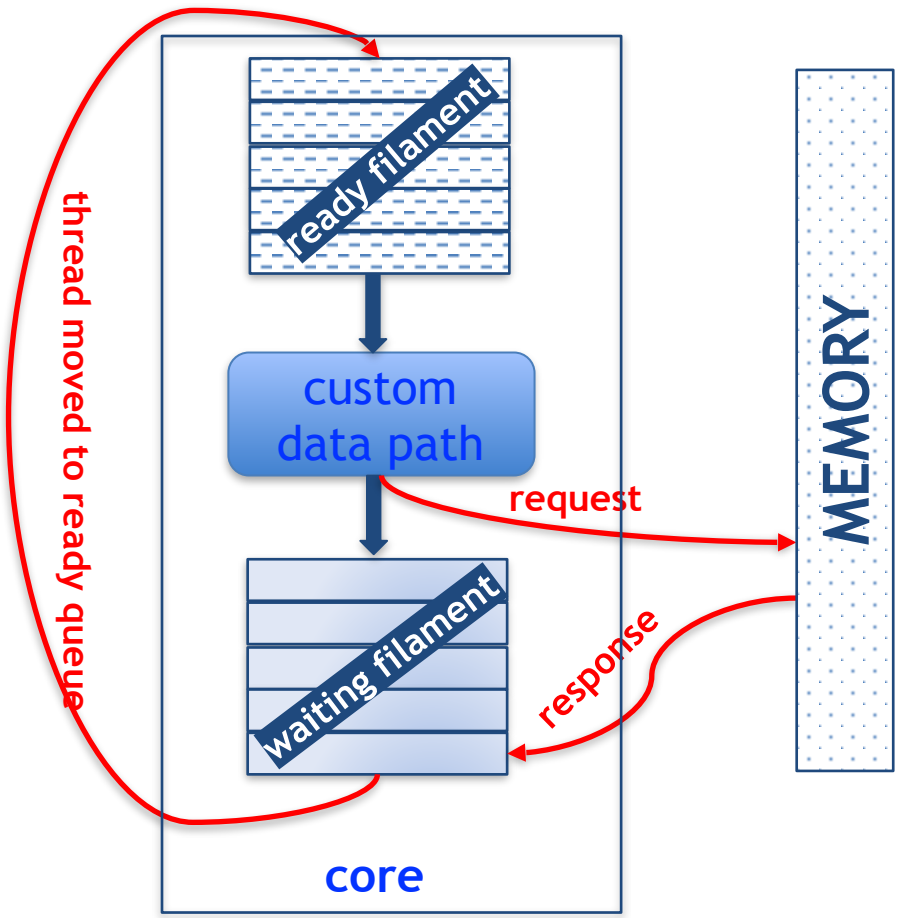
hardware multithreading



hardware multithreading

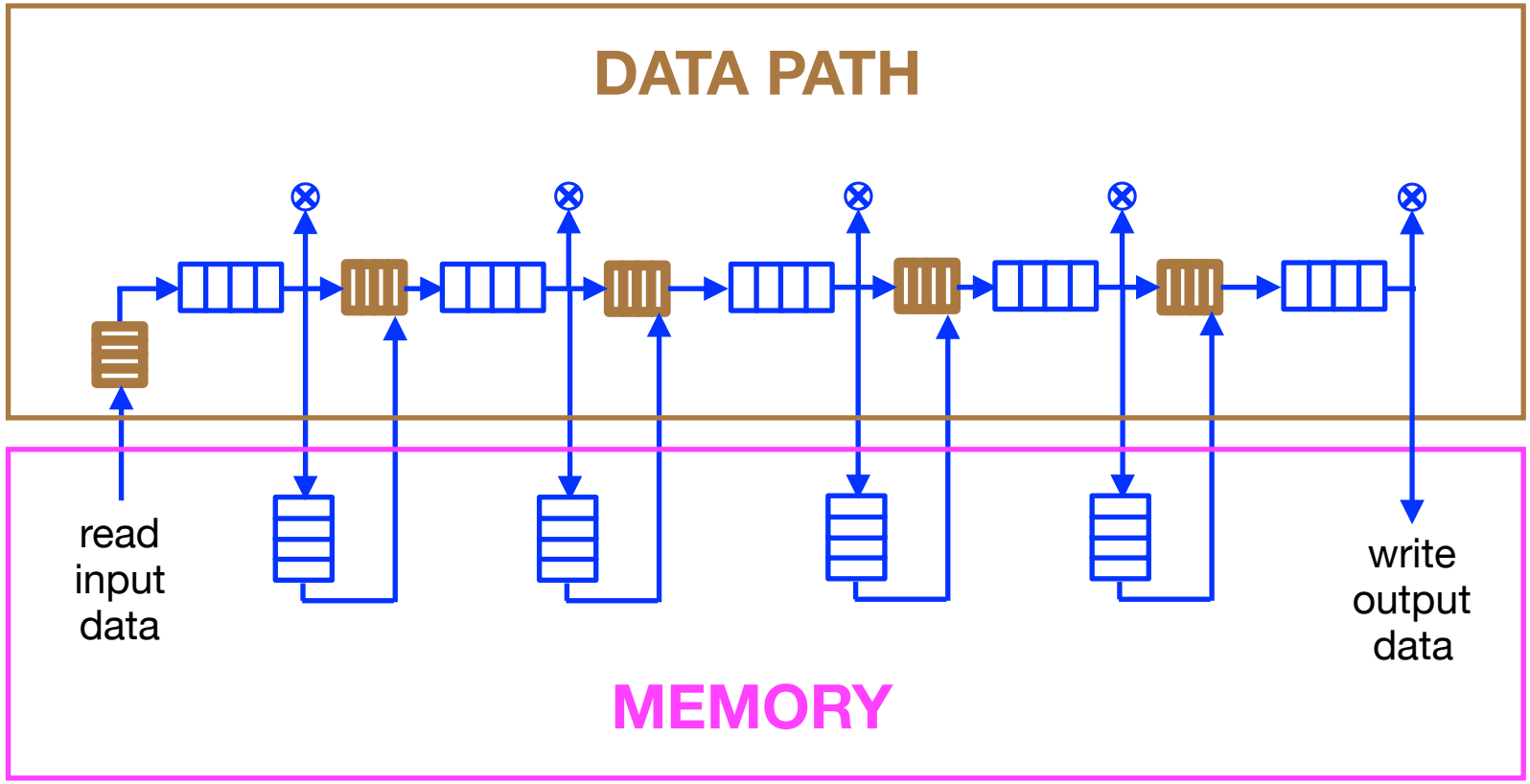


hardware multithreading



- Call them filament, to distinguish from "threads"
- Preempting executing filaments once it access memory
- Ready & Waiting Filament Queues fit 1000s of thread states
- Massive parallelism: ≥ 4,000 waiting filaments

filament data path



wait queue

filament termination

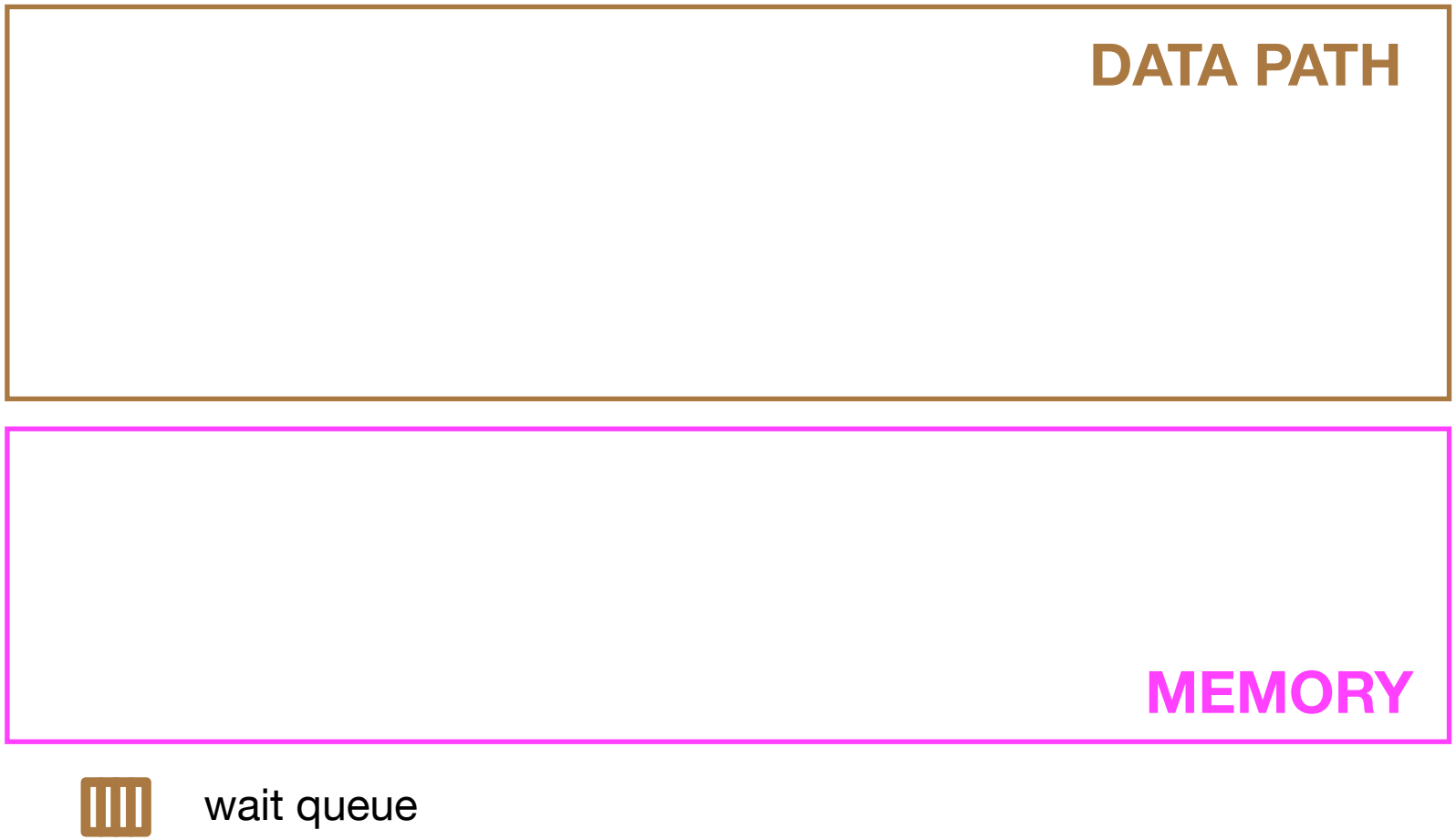
filament data path example

filament data path example

*A[i] += B[i] * C[D[i]]; % typical of sparse linear algebra*

filament data path example

*A[i] += B[i] * C[D[i]]; % typical of sparse linear algebra*



filament data path example

*A[i] += B[i] * C[D[i]]; % typical of sparse linear algebra*

generate i

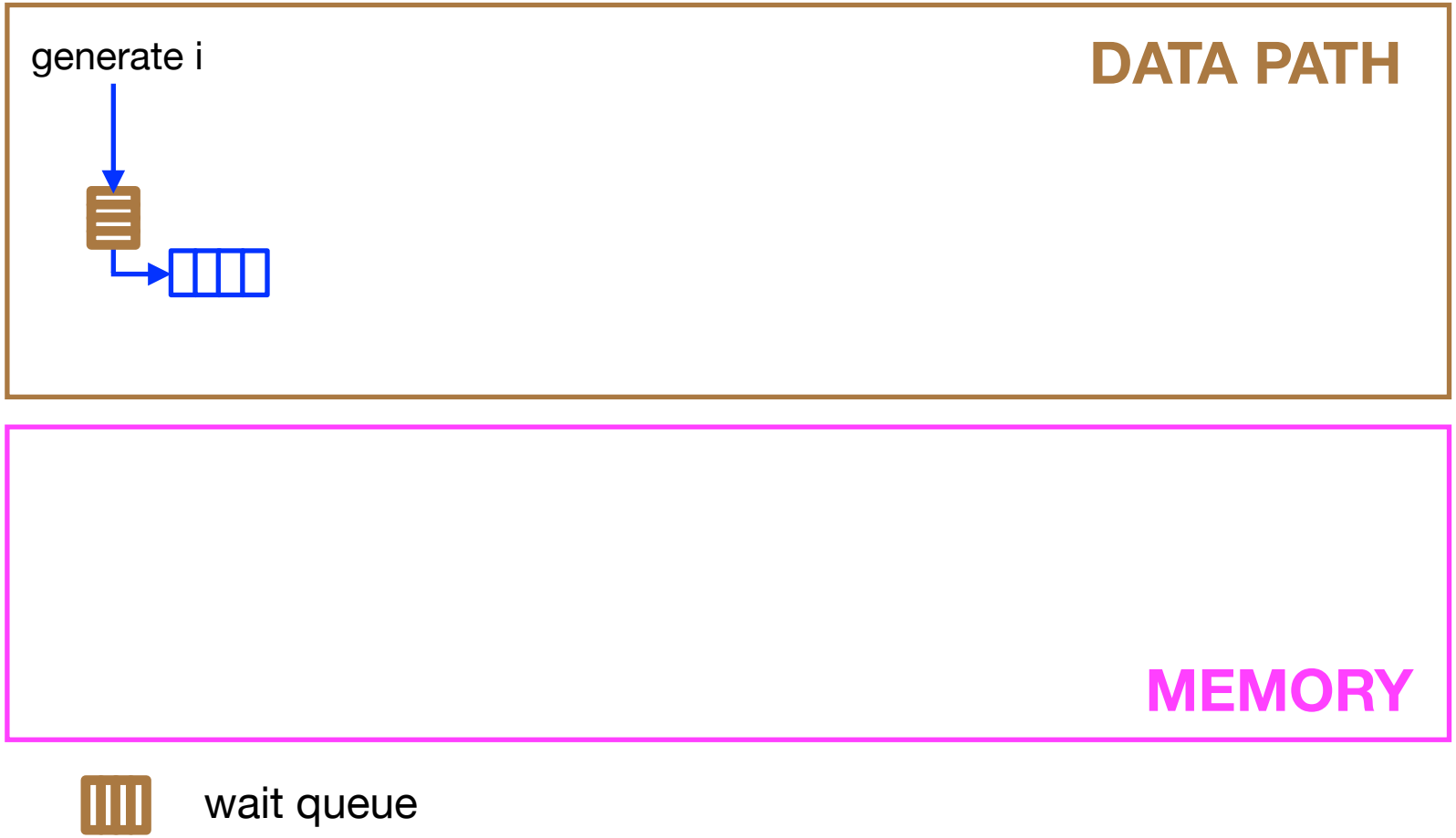
DATA PATH

MEMORY

wait queue

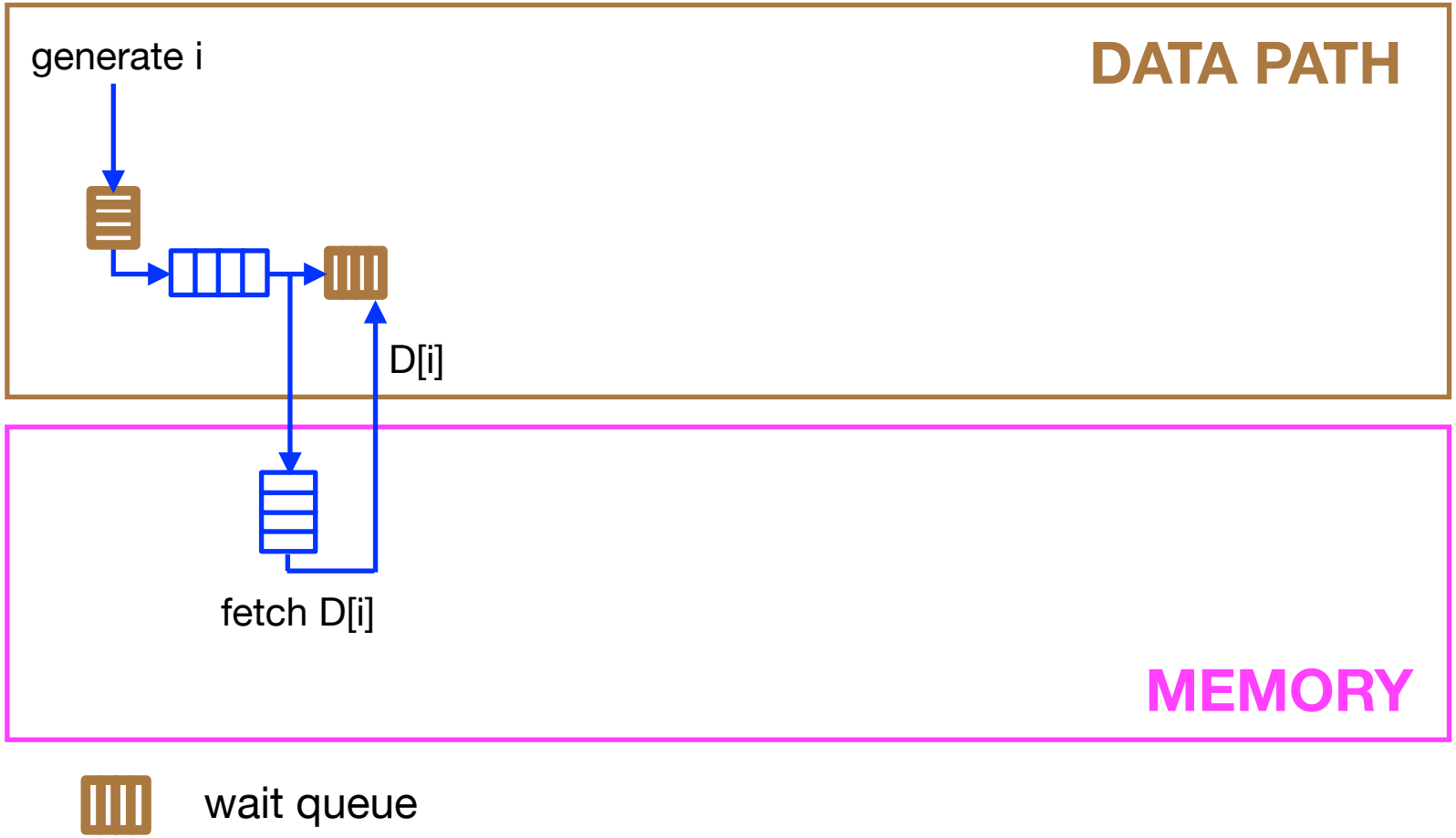
filament data path example

*A[i] += B[i] * C[D[i]]; % typical of sparse linear algebra*



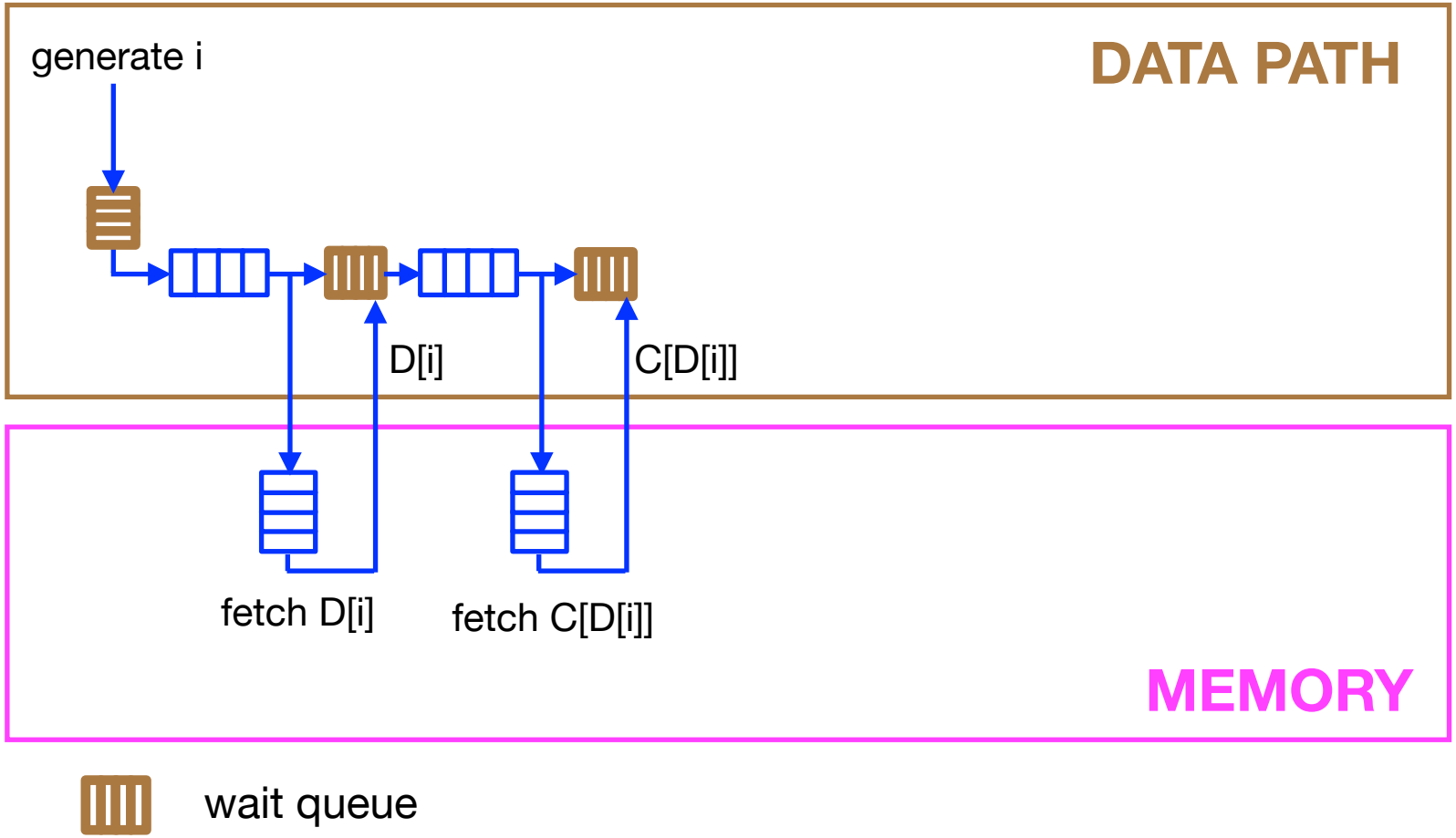
filament data path example

*A[i] += B[i] * C[D[i]]; % typical of sparse linear algebra*



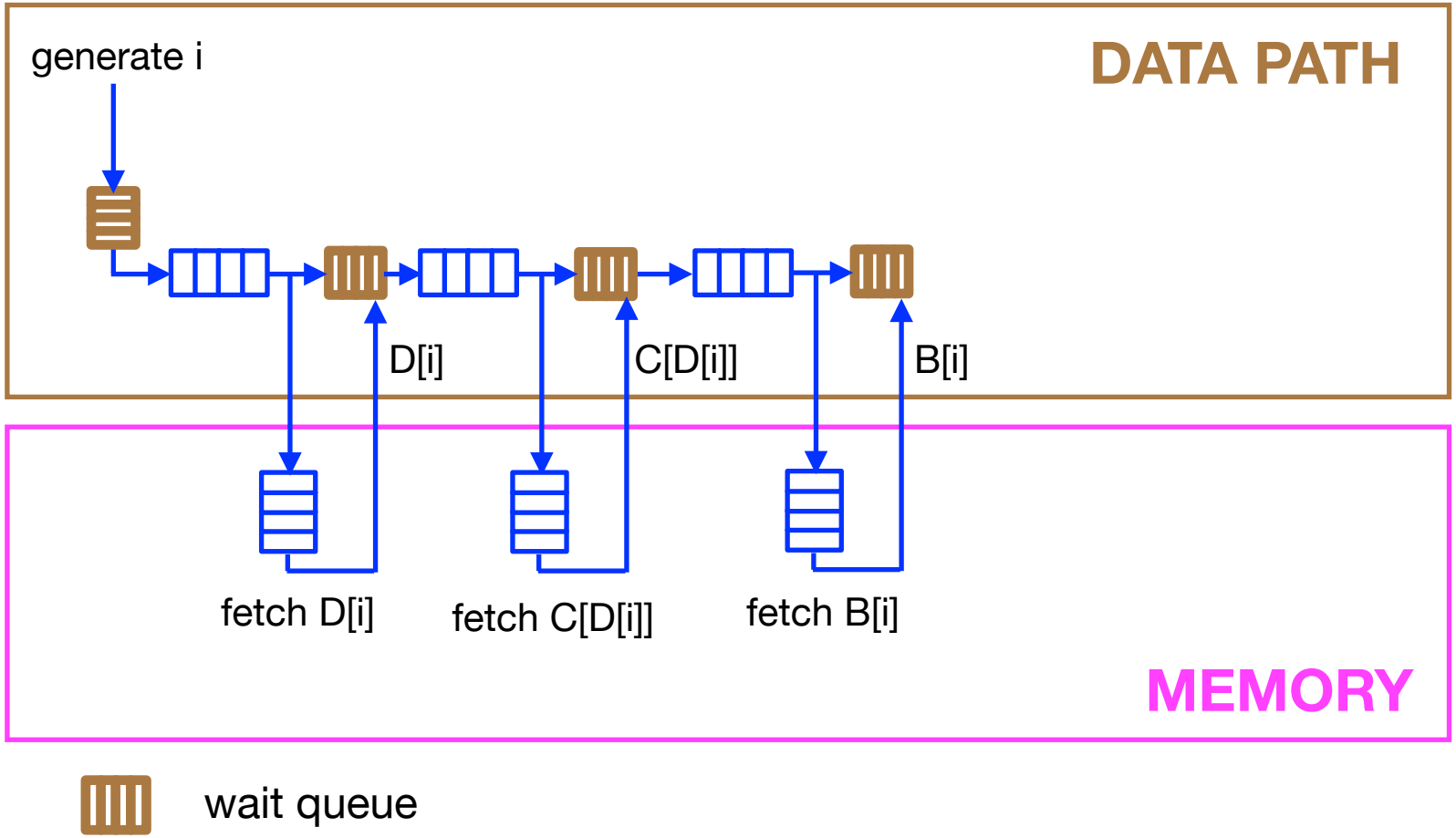
filament data path example

*A[i] += B[i] * C[D[i]]; % typical of sparse linear algebra*



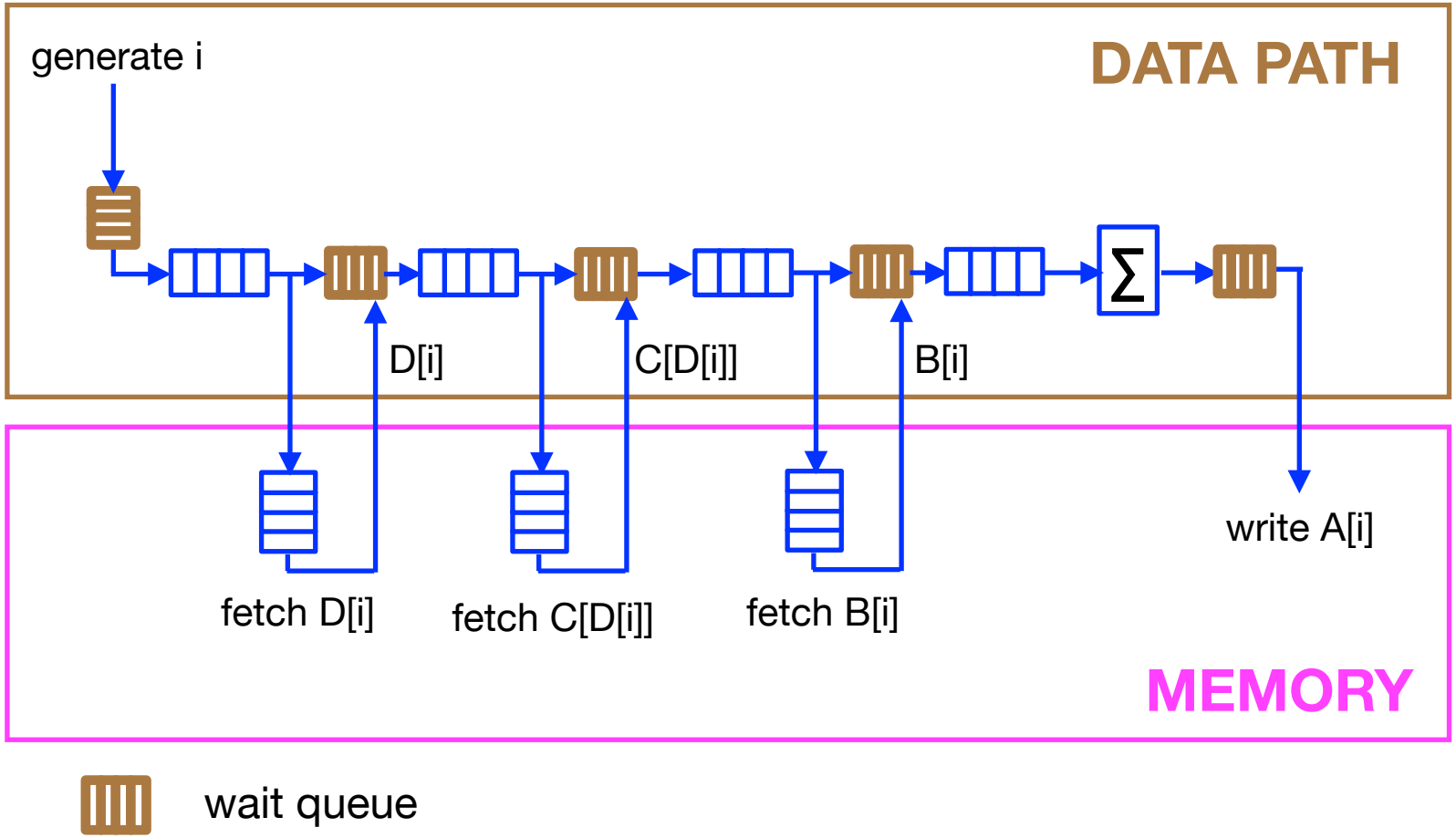
filament data path example

*A[i] += B[i] * C[D[i]]; % typical of sparse linear algebra*



filament data path example

*A[i] += B[i] * C[D[i]]; % typical of sparse linear algebra*

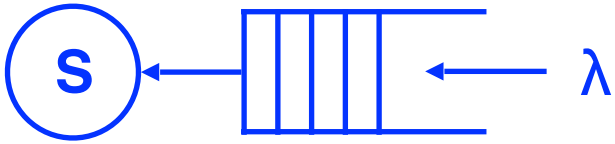


a very important

a very important *and often forgotten Little law*

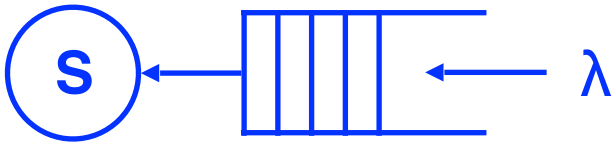
a very important *and often forgotten Little law*

a queueing system



a very important *and often forgotten Little law*

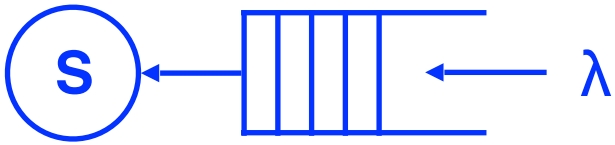
a queueing system



- **N** = customers in system, **w** = wait time
- ∀ distribution of **λ** and **S** (service time)

a very important *and often forgotten Little law*

a queueing system

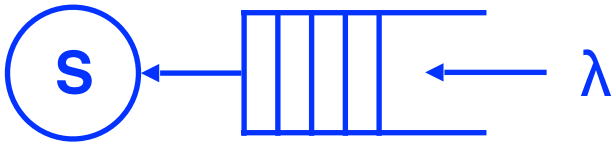


- **N** = customers in system, **w** = wait time
- ∀ distribution of **λ** and **S** (service time)

$$N = \lambda . w$$

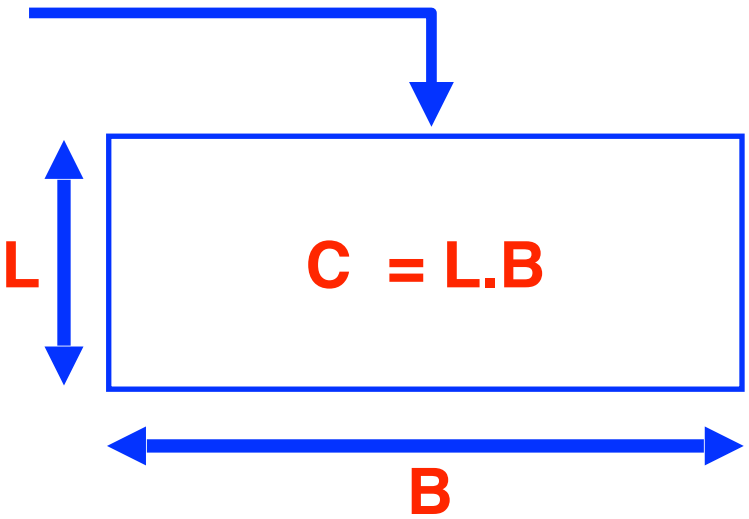
a very important *and often forgotten Little law*

a queueing system



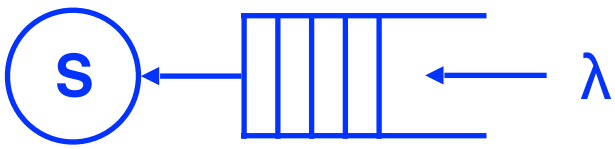
- **N** = customers in system, **w** = wait time
- ∀ distribution of **λ** and **S** (service time)

N = λ.w



a very important *and often forgotten Little law*

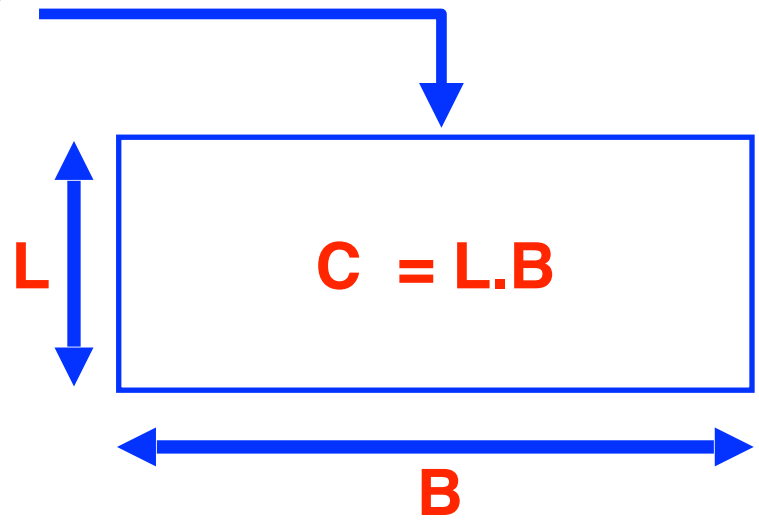
a queueing system



- **N** = customers in system, **w** = wait time
- ∀ distribution of **λ** and **S** (service time)

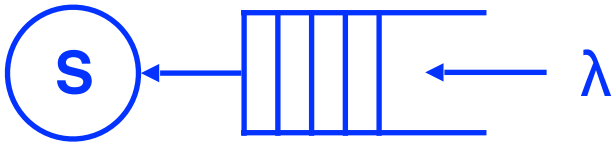
N = λ.w

- L = pipeline latency
- B = memory bandwidth
- C = parallelism



a very important *and often forgotten Little law*

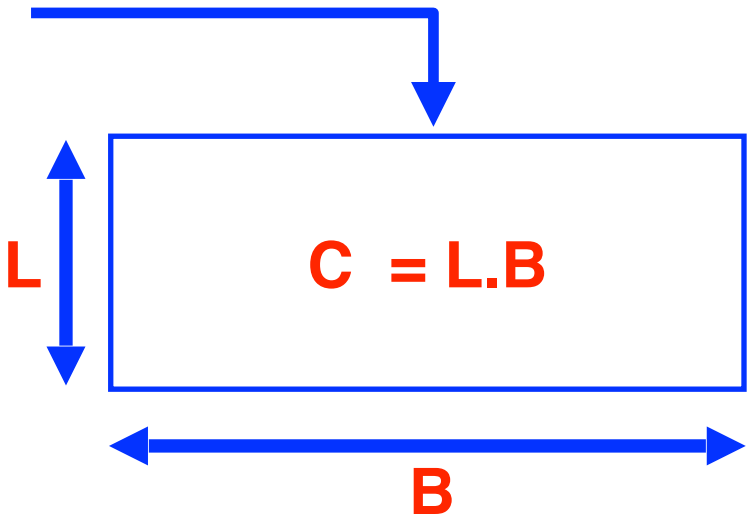
a queueing system



- **N** = customers in system, **w** = wait time
- ∀ distribution of **λ** and **S** (service time)

N = λ.w

- L = pipeline latency
- B = memory bandwidth
- C = parallelism



high bandwidth + hyper-pipelining —> massive parallelism

regular vs. irregular applications

regular vs. irregular applications

- Regular Applications
  - Have good temporal and spatial locality
  - Caches are well suited for these types of applications

regular vs. irregular applications

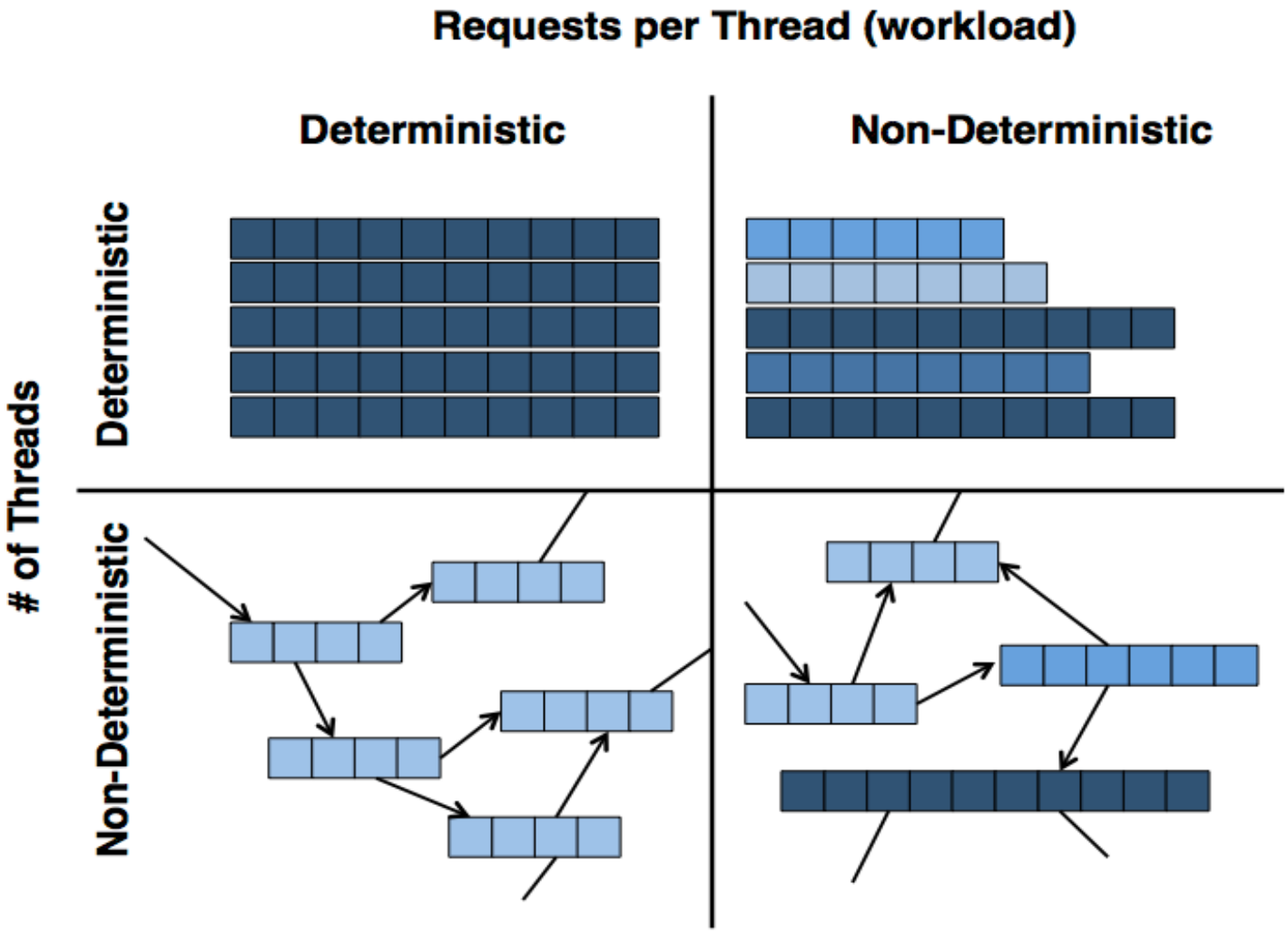
- Regular Applications
  - Have good temporal and spatial locality
  - Caches are well suited for these types of applications
- Irregular Applications
  - Have poor temporal and spatial locality
  - Caches are NOT well suited for these types of applications
  - Multithreading is an alternative
    - Can mask long memory latencies
    - Needs high bandwidth to support a large number of concurrent threads

regular vs. irregular applications

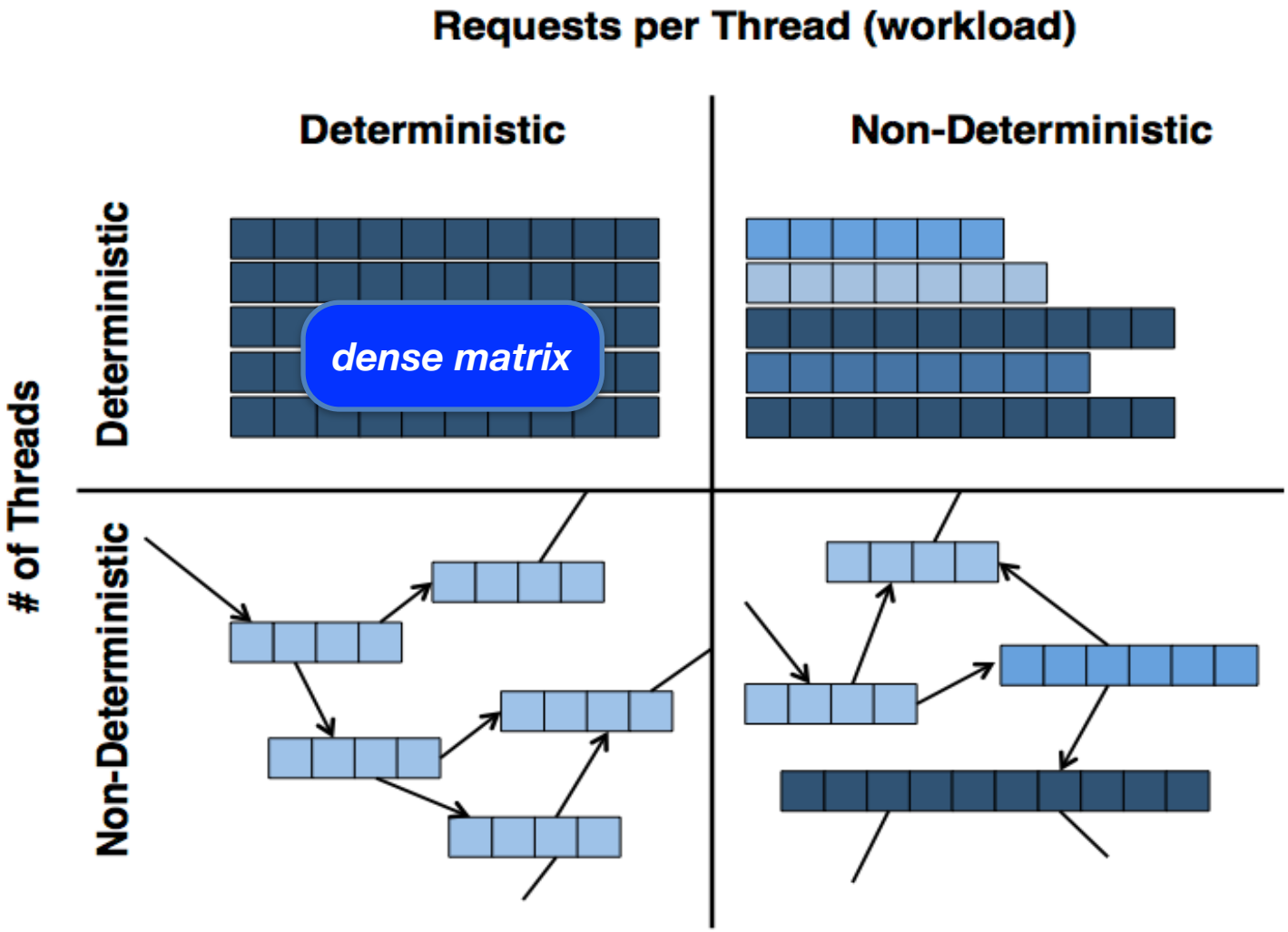
- Regular Applications
  - Have good temporal and spatial locality
  - Caches are well suited for these types of applications
- Irregular Applications
  - Have poor temporal and spatial locality
  - Caches are NOT well suited for these types of applications
  - Multithreading is an alternative
    - Can mask long memory latencies
    - Needs high bandwidth to support a large number of concurrent threads
- Won't increasing the cache size mitigate latency and improve performance?

taxonomy of irregular applications

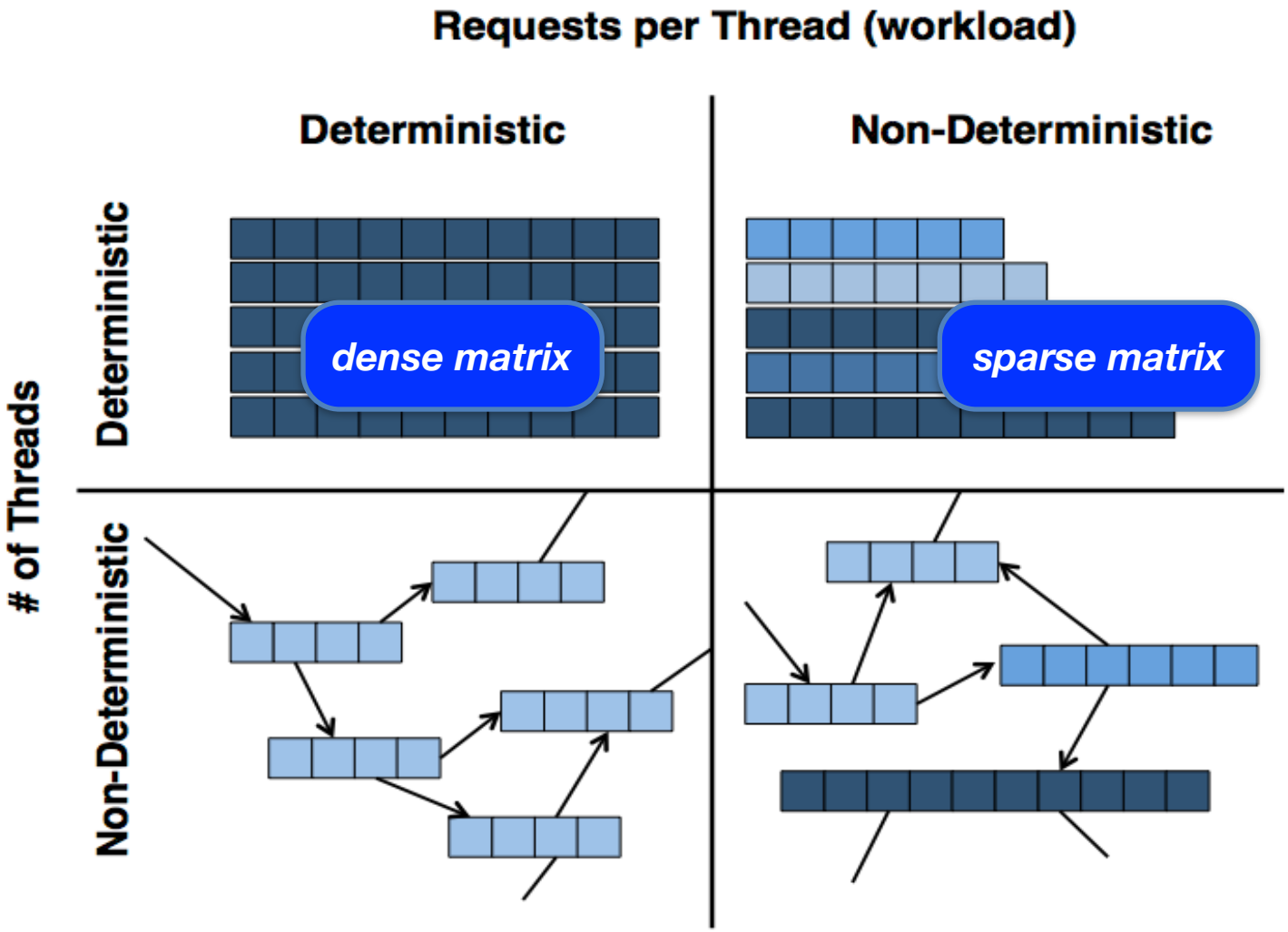
taxonomy of irregular applications



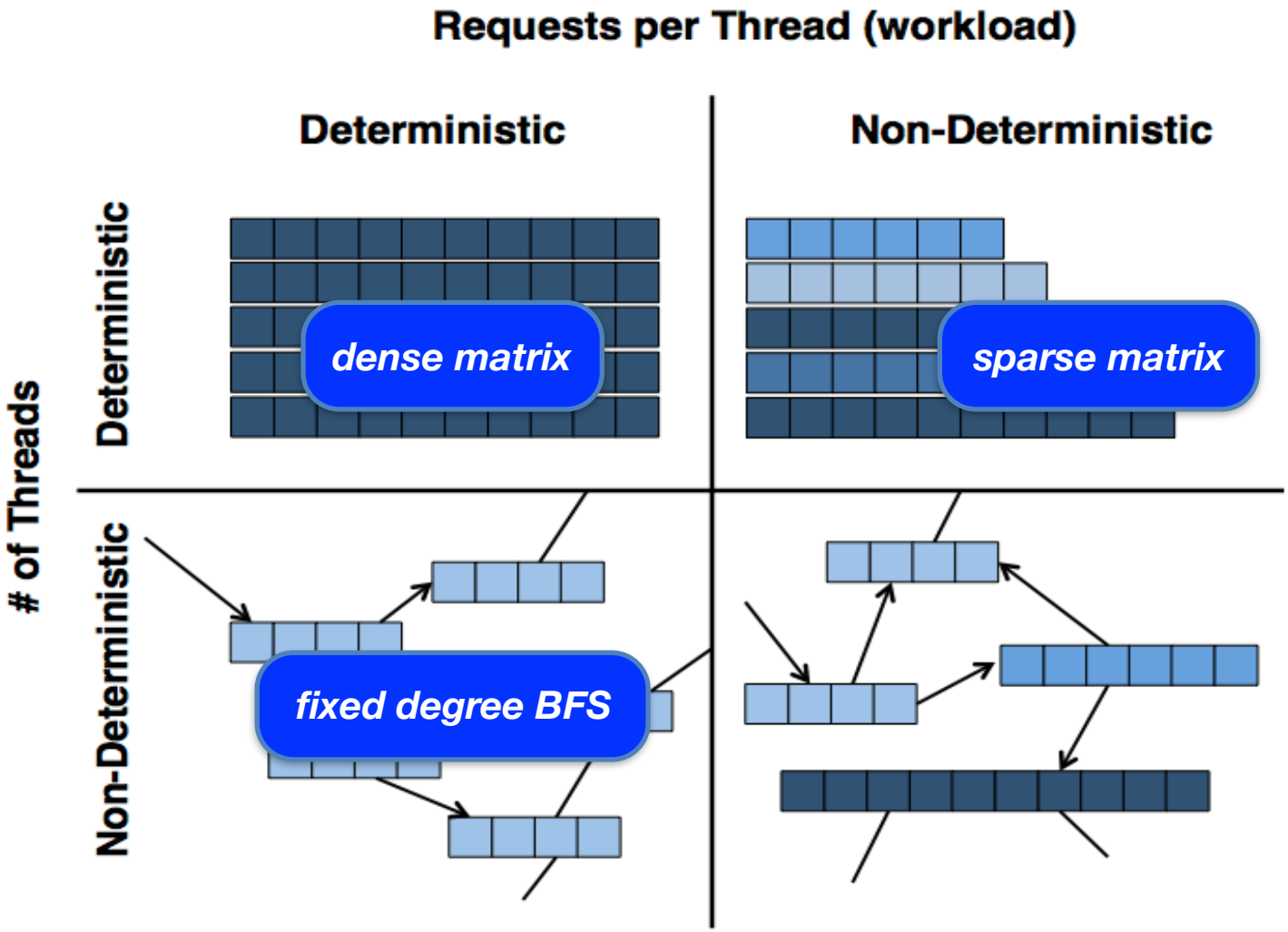
taxonomy of irregular applications



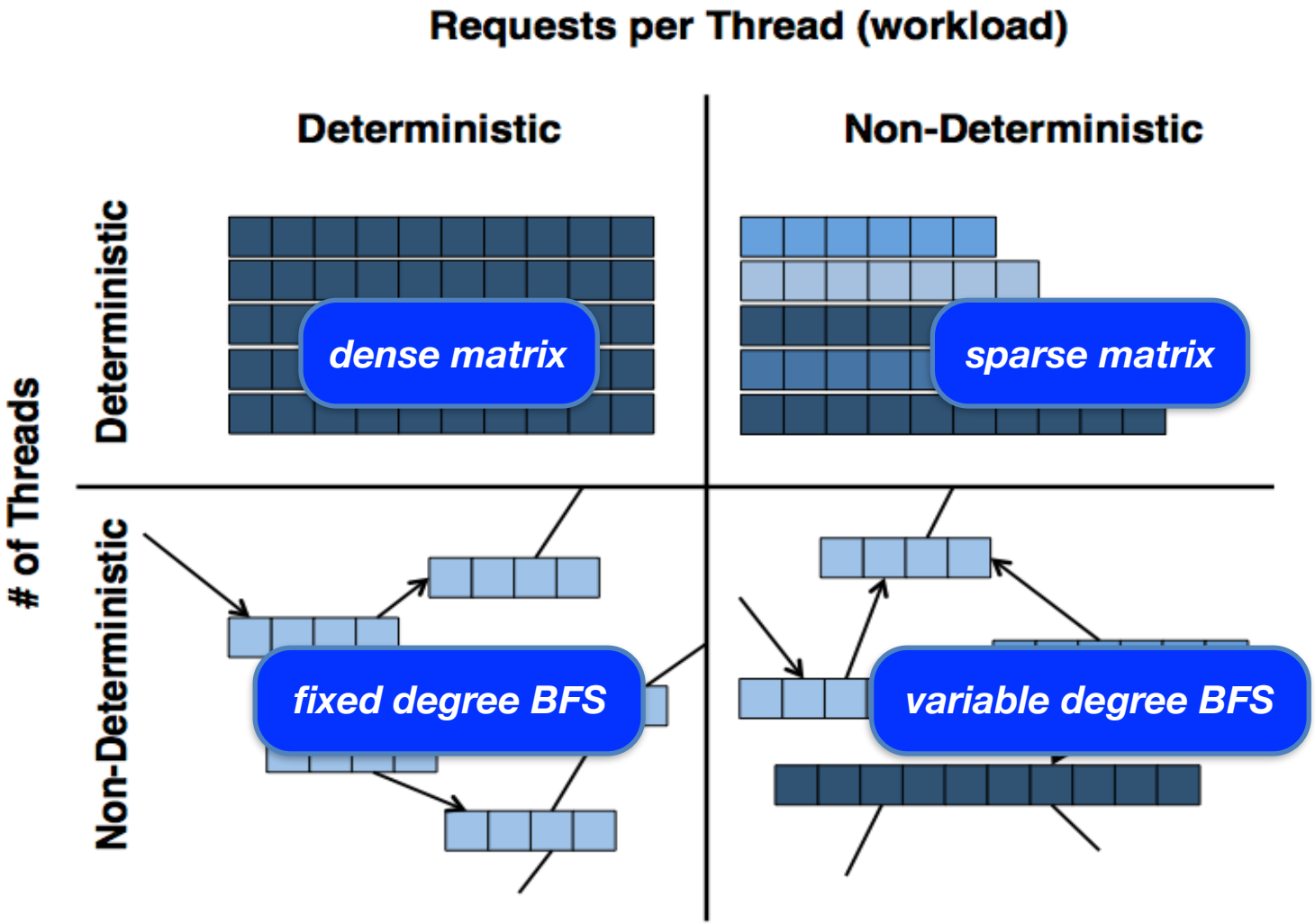
taxonomy of irregular applications



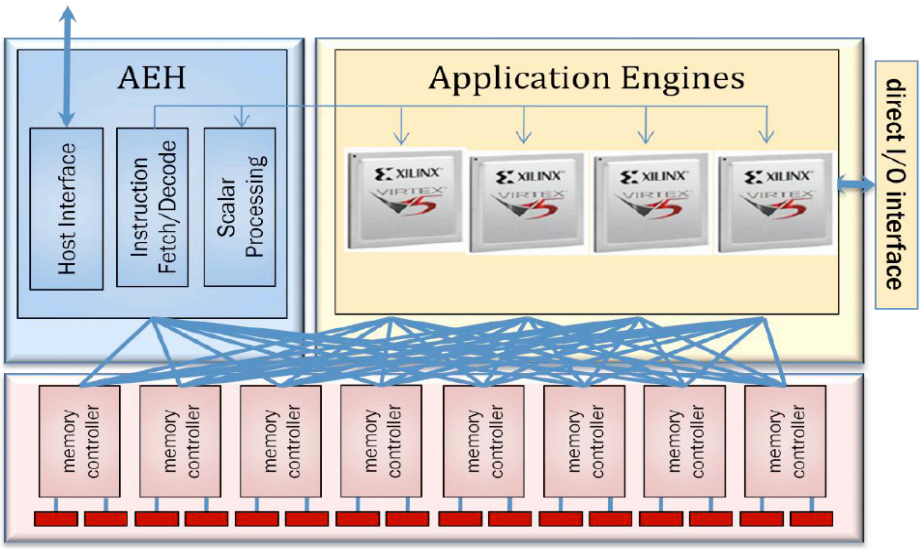
taxonomy of irregular applications



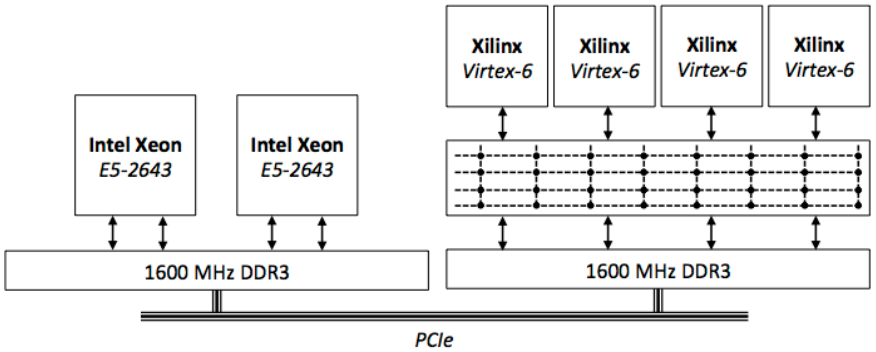
taxonomy of irregular applications

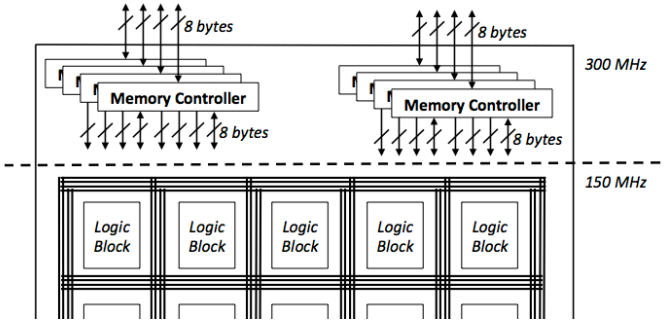


Convey HC-2ex architecture



Convey HC-2ex	
Clock Freq.	150 MHz
Memory Controllers	8
Memory Channels	16 / FPGA
Memory Bandwidth	76.8 GB/s
Outstanding Requests	**~ 500 Requests / Channel**
Data bus width	8 Byte





- multi bank memory architecture
- randomized memory allocation

outline

- background
- filament execution model
- ~~application - SpMV~~
- **application - Selection**
- application - Group by aggregation
- conclusion

selection query

- selection is a database operator
  - selects all rows (tuples) in a relation (table)
  - that satisfy a set of conditions
  - expressed as a logical expression on the attributes
  - attributes (columns) are elements of a tuple
  - e.g. $((a1 > 10) \wedge (a2>0)) \vee ((a10<100) \wedge (a13<0))$
- a very common database operator
  - poor locality
  - relation can be store row or column wise

selection example

selection example

relation example

row/ attribute	a1	a2	a3	a4	a5	a6
0	5	-70	CA	1992	0	12
1	10	70	CO	1990	1	4
2	15	-10	AZ	2001	1	24
3	20	3	NV	1989	0	18

selection example

relation example

row/ attribute	a1	a2	a3	a4	a5	a6
0	5	-70	CA	1992	0	12
1	10	70	CO	1990	1	4
2	15	-10	AZ	2001	1	24
3	20	3	NV	1989	0	18

selection example

relation example

row/ attribute	a1	a2	a3	a4	a5	a6
0	5	-70	CA	1992	0	12
1	10	70	CO	1990	1	4
2	15	-10	AZ	2001	1	24
3	20	3	NV	1989	0	18

query example:

query A: $(a5=1) \vee (a3 = NV) \vee [(a1<20) \wedge (a2>0)]$

query B: $(a2<0) \wedge (a3=CA) \wedge (a5=0)$

selection example

relation example

row/ attribute	a1	a2	a3	a4	a5	a6
0	5	-70	CA	1992	0	12
1	10	70	CO	1990	1	4
2	15	-10	AZ	2001	1	24
3	20	3	NV	1989	0	18

query example:

query A: $(a5=1) \vee (a3 = NV) \vee [(a1<20) \wedge (a2>0)]$

query B: $(a2<0) \wedge (a3=CA) \wedge (a5=0)$

selection example

relation example

row/ attribute	a1	a2	a3	a4	a5	a6
0	5	-70	CA	1992	0	12
1	10	70	CO	1990	1	4
2	15	-10	AZ	2001	1	24
3	20	3	NV	1989	0	18

query example:

query A: $(a5=1) \vee (a3 = NV) \vee [(a1<20) \wedge (a2>0)]$

query B: $(a2<0) \wedge (a3=CA) \wedge (a5=0)$

qualifying rows

row/ query	qA	qB
0	N	Y
1	Y	N
2	Y	N
3	N	N

selection example (2)

query example:

query A: $(a5=1) \vee (a3 = NV) \vee [(a1<20) \wedge (a2>0)]$

query B: $(a2<0) \wedge (a3=CA) \wedge (a5=0)$

predicate control block (PCB) encodes the selection logic:

- early termination (T or F)
- only the required attributes are fetched (no caching)

qA predicate control block

	condition		action	
attribute	op	const	if T	if F
a5	=	1	Terminate **T**	a3
a3	=	NV	Terminate **T**	a5
a1	<	20	a2	Terminate **F**
a2	>	0	Terminate **T**	Terminate **F**

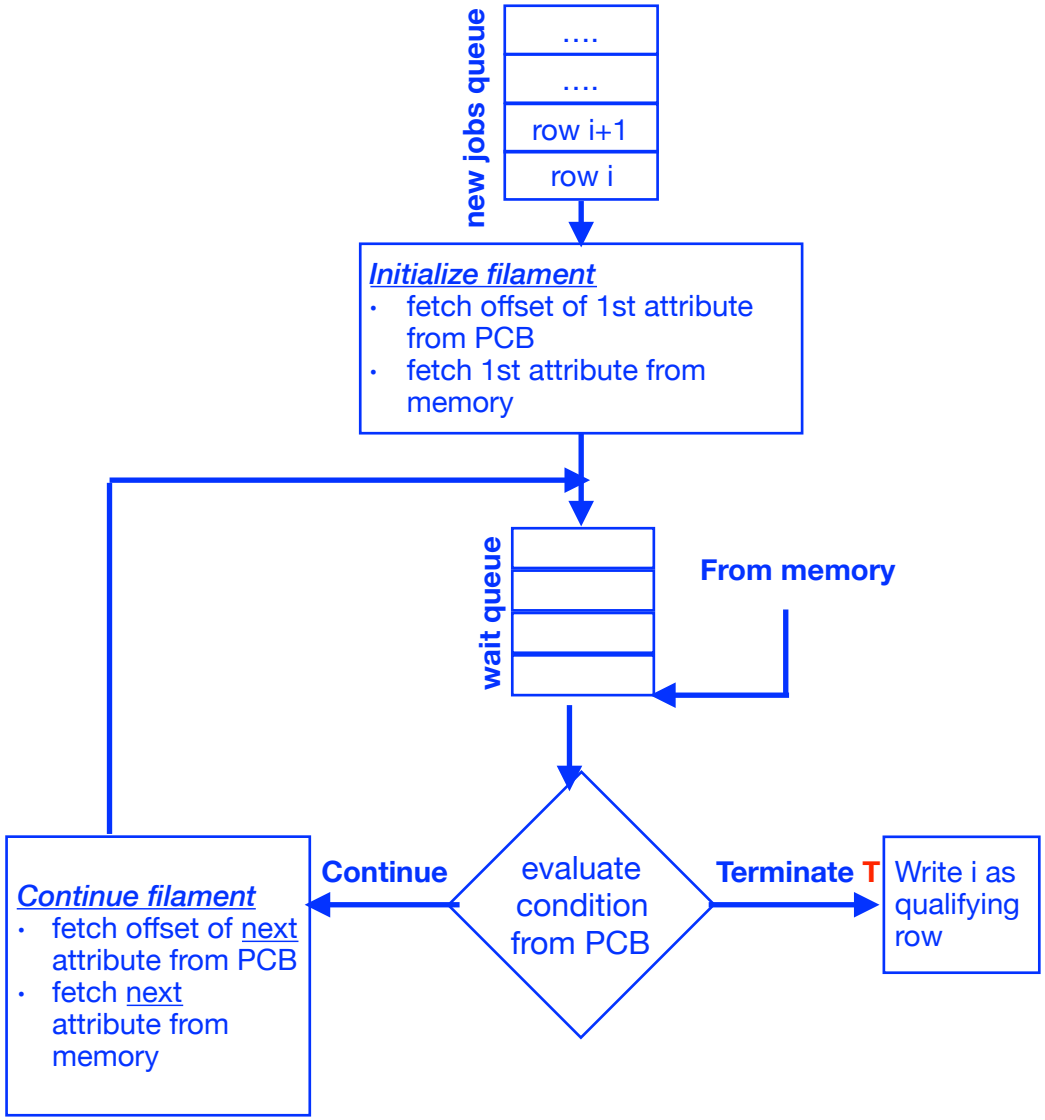
qB predicate control block

	condition		action	
attribute	op	const	if T	if F
a2	<	0	a3	Terminate **F**
a3	=	CA	a5	Terminate **F**
a5	=	0	Terminate **T**	Terminate **F**

selection CDFG

Explanations:

- PCB: process control block (stored locally on the FPGA)
- The state of each filament is the current index into the PCB
- Multiple queries can be executed simultaneously on the same relation, the state becomes the (PCB #, index)
- data is returned from memory in the order fetched, on one channel

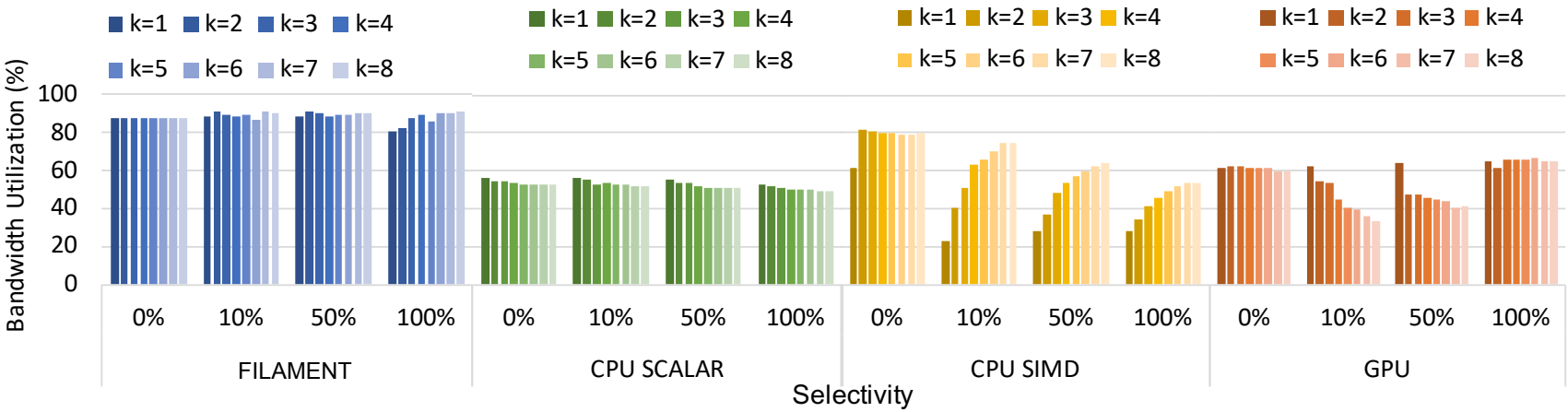


experimental evaluation

- Each row consists of 8 fixed size 64-bit columns.
- Query selectivity was varied from 0% to 100% (0%: no row qualifies, 100% all rows qualify)
- Dataset varied from 8M – 128M.
- Results were obtained on the following platforms:

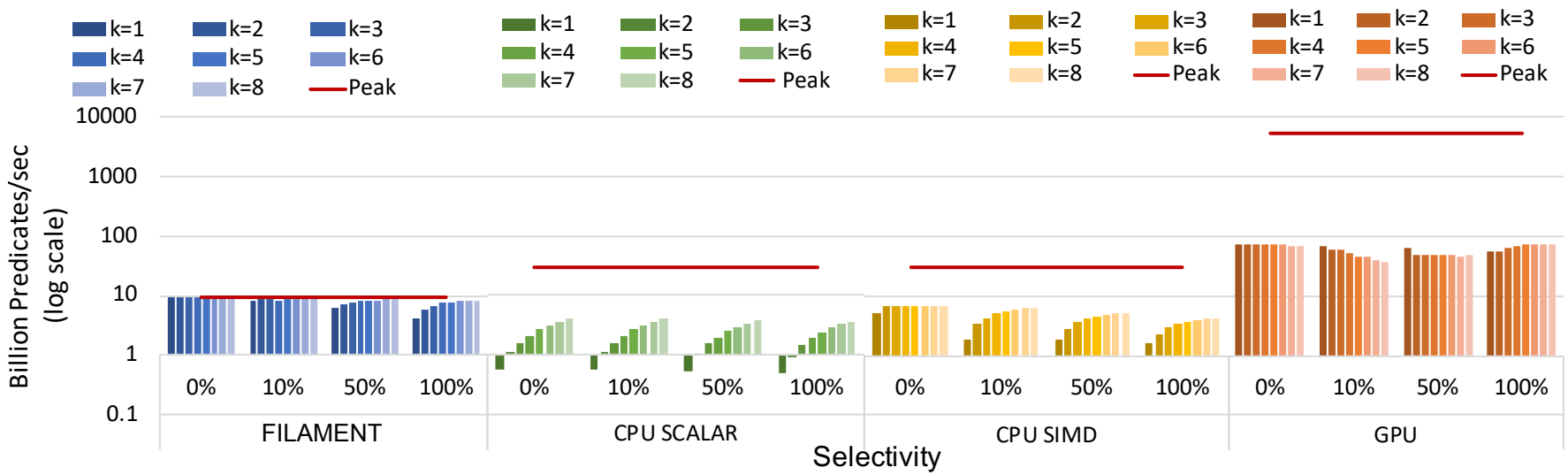
Device	Make & Model	Clock (MHz)	Cores	Memory Size (GB)	Memory Bandwidth, GB/s
CPU	Intel Xeon E5-2643	3,300	8	128	51.2
GPU	Nvidia Titan X	1,500	3,584	12	480.0
FPGA	Virtex6 – 760	150	64	64	76.8

bandwidth utilization - selection



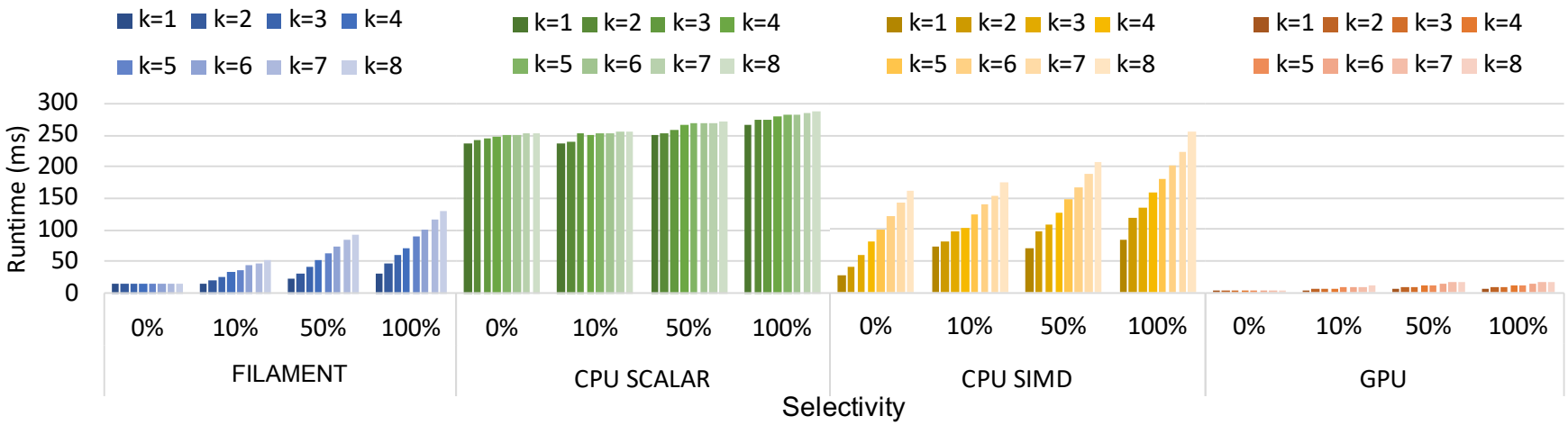
filament has the highest bandwidth utilization

predicate evaluation/sec



peak = theoretical max throughput

runtime (msec)



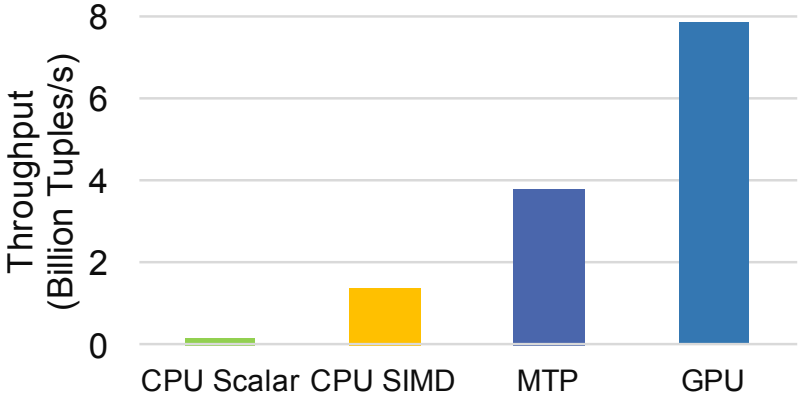
at only 150 MHz

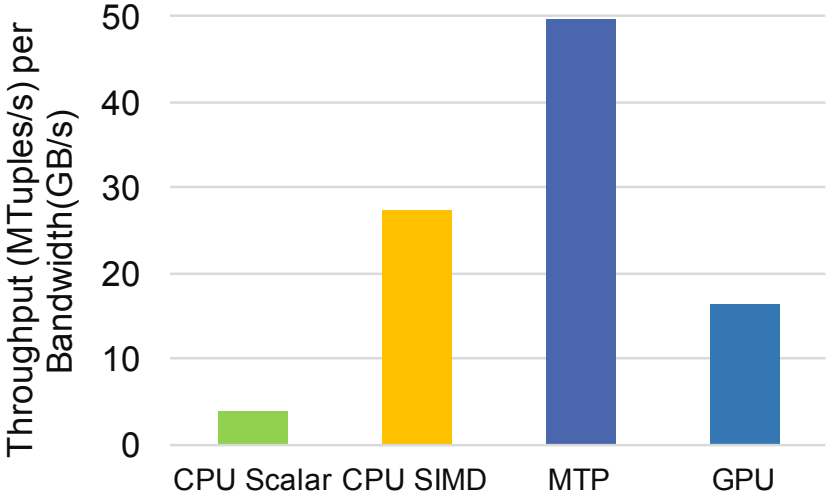
TPC-H benchmark

- Query Q6 for our evaluation because all select conditions apply on the columns of one table and has 5 predicates
- Query Q6:

```
SELECT * FROM lineitem
WHERE l_shipdate>= date '1995-01-01' AND
l_shipdate<date '1996-01-01' AND
l_discount between 0.04 AND 0.06 AND
l_quantity<24;
```

- Lineitem table has 16 columns
- Measured selectivity is 1.91%
- CPU and GPU access common attributes only once.
- MTP treats them as two independent attributes.





TPC-H Evaluation

- **Memory Fetches**: number of columns fetched for evaluation
- **Avg Evaluations per Row**: predicate comparisons per row
- **Effective Bandwidth**: total size of the "Lineitem" table processed per unit time / theoretical peak bandwidth, prescribed in [1].
- **Peak Bandwidth Utilization**: actual amount of data fetched per unit time / theoretical peak bandwidth

	CPU Scalar	CPU SIMD	GPU	Filament
Memory Fetches	100%	18.75%	18.75%	**11.4%**
Avg Evaluations (per Row)	3	3	3	**1.83**
Effective Bandwidth Speedup	0.47	3.44	2.01	**6.13**
Peak Bandwidth Utilization	47.6%	61.2%	36.6%	**70.3%**

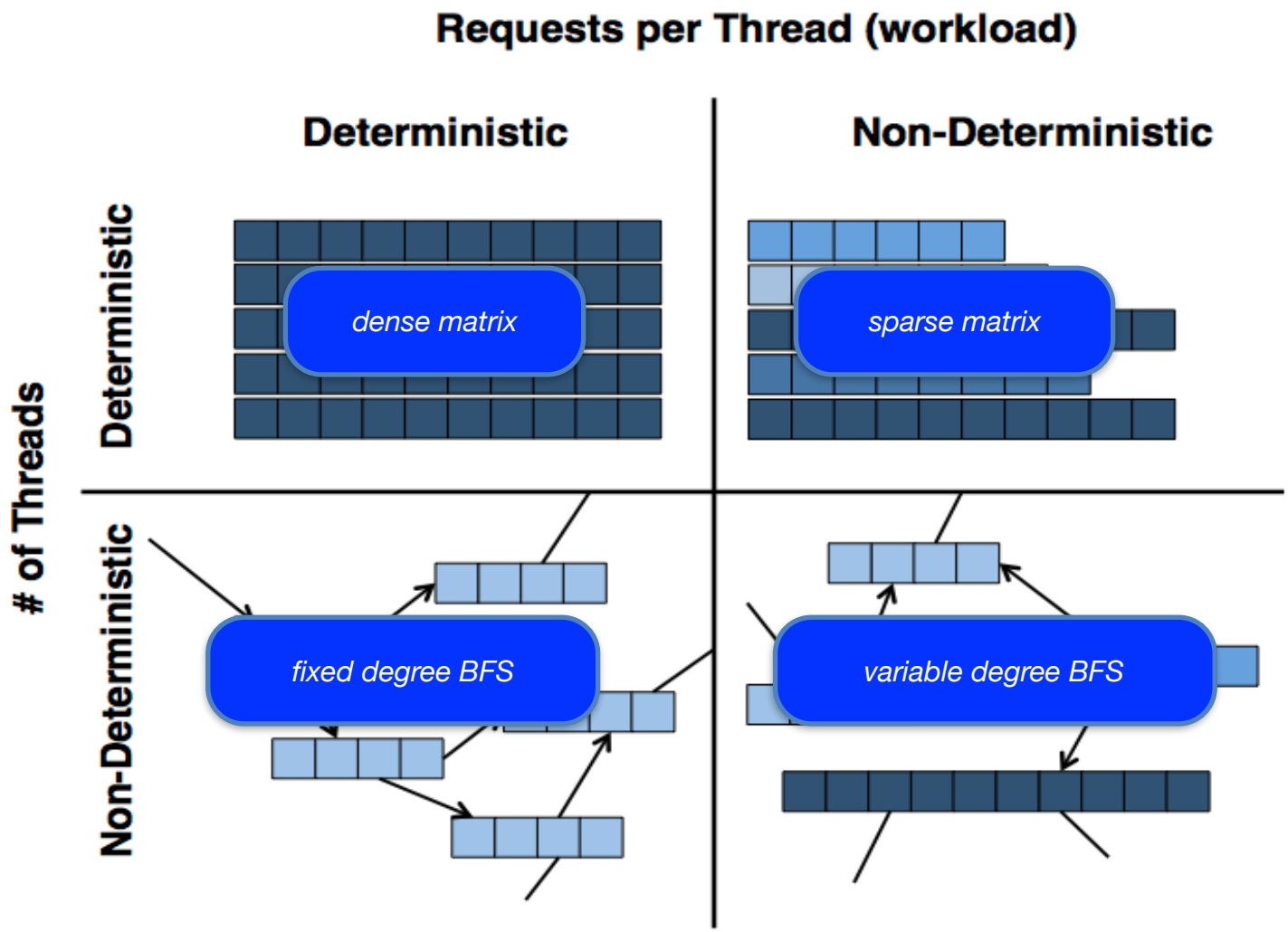
[1] B. Sukhwani, et al. Database Analytics Acceleration Using FPGAs. Parallel Architectures and Compilation Techniques, 2012.

outline

- background
- filament execution model
- ~~application - SpMV~~
- application - Selection
- **application - Group by aggregation**
- conclusion

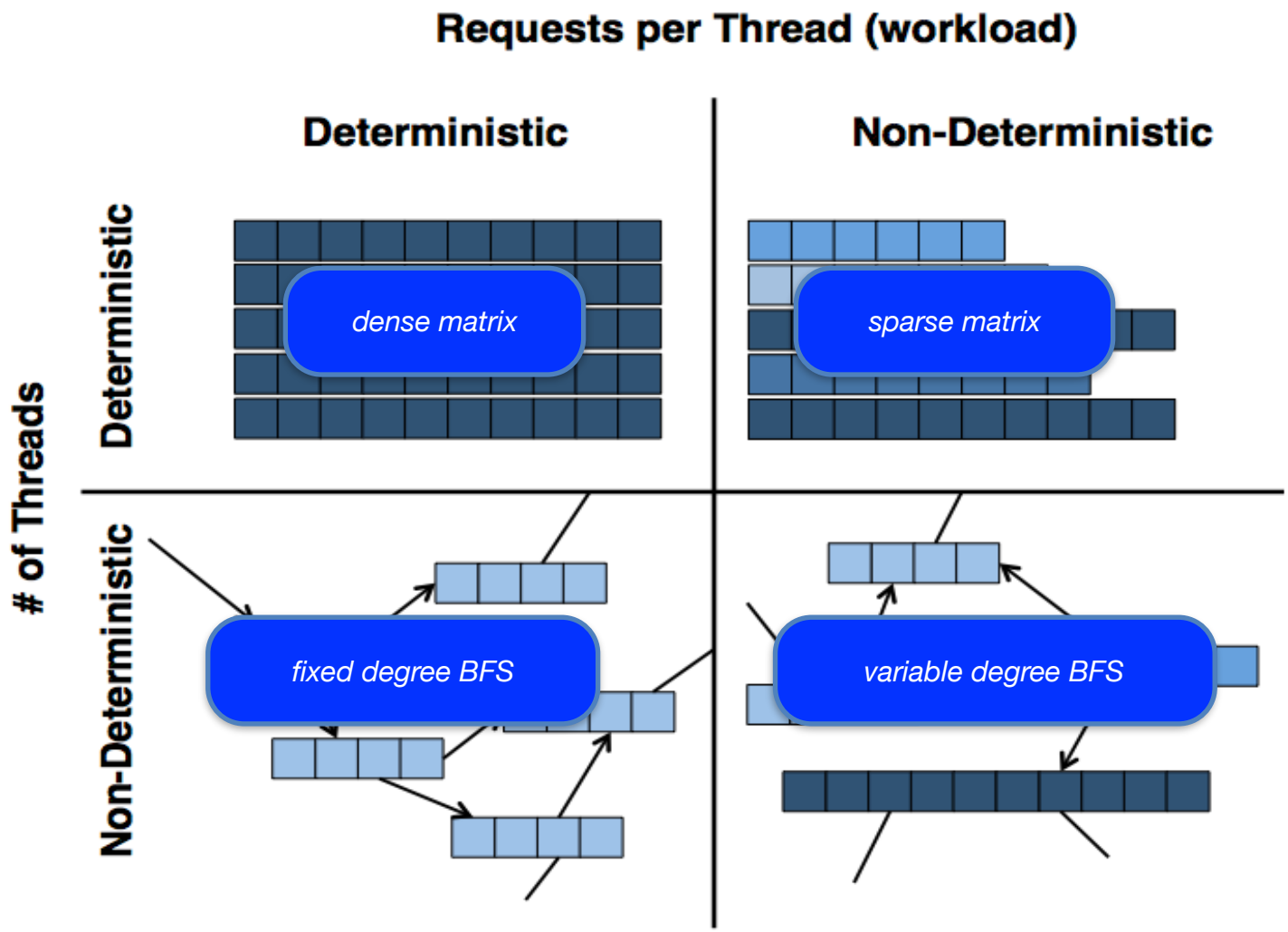
synchronization in irregular applications

synchronization in irregular applications



synchronization in irregular applications

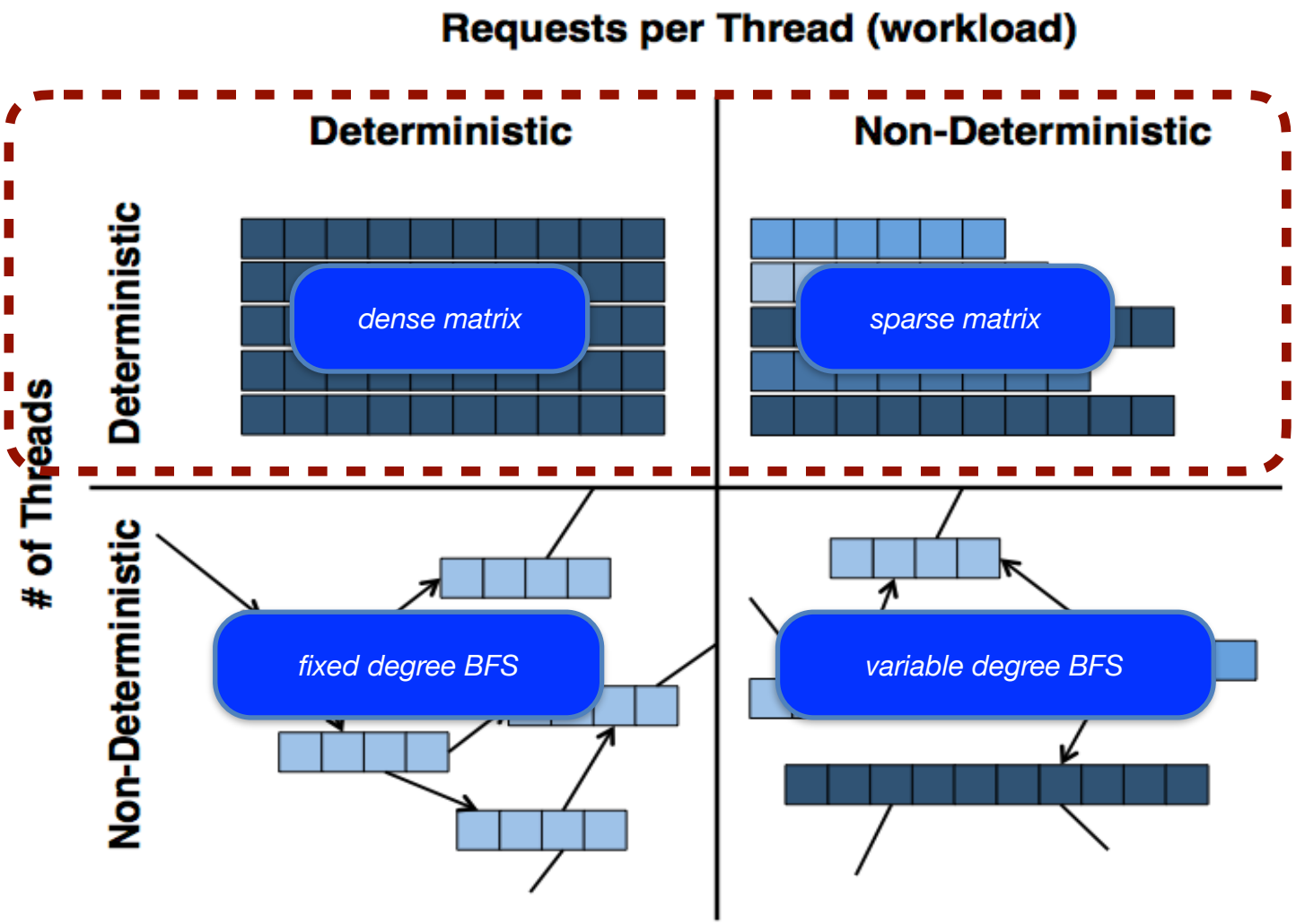
inter-thread synchronization



synchronization in irregular applications

inter-thread synchronization

not needed

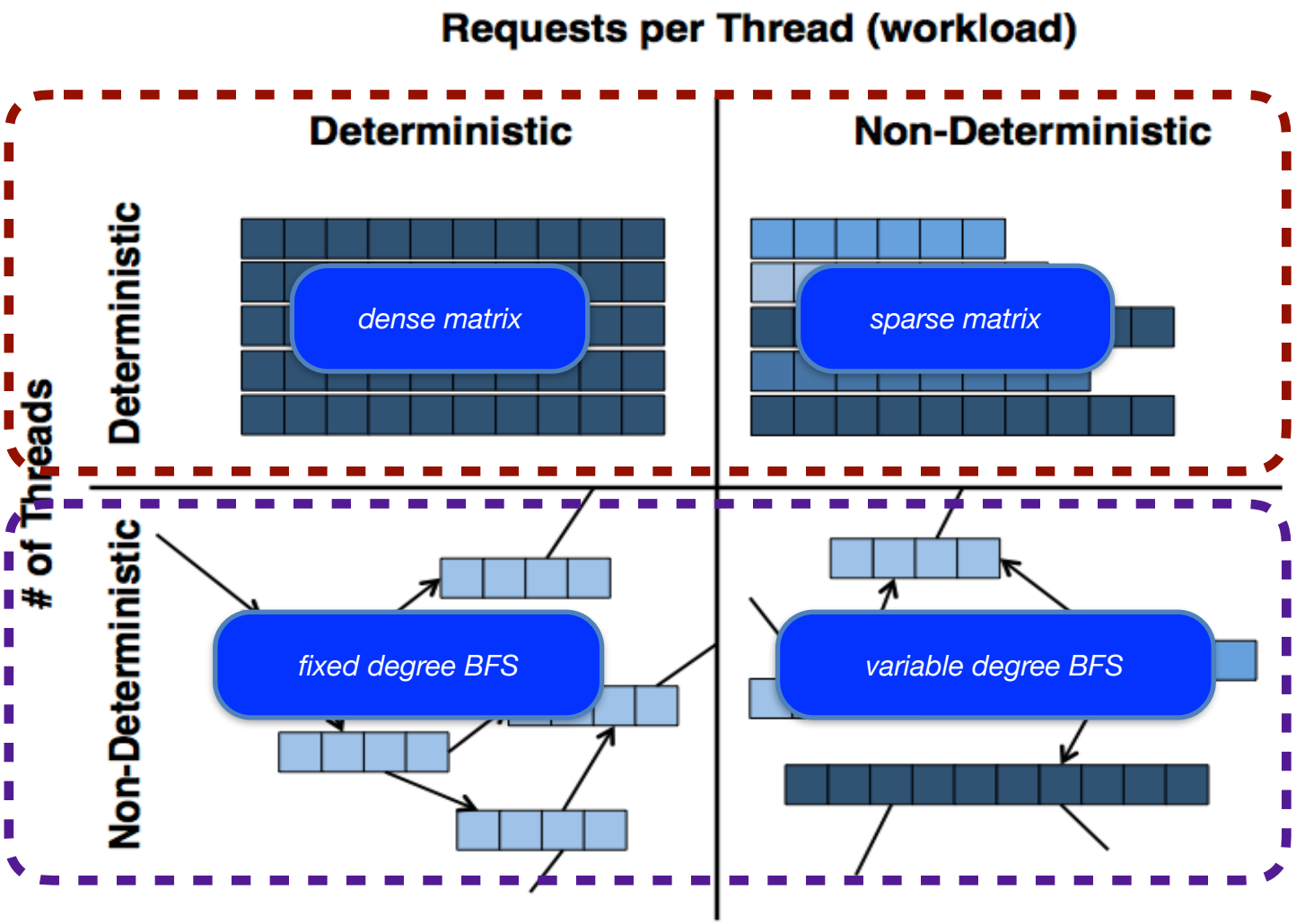


synchronization in irregular applications

inter-thread synchronization

not needed

needed



synchronizing between threads

synchronizing between threads

- Non-deterministic # threads
  - requires inter-thread synchronization
    - only FPGA platforms that supports in-memory synchronization: Convey MX
    - Heavy overhead

synchronizing between threads

- Non-deterministic # threads
  - requires inter-thread synchronization
    - only FPGA platforms that supports in-memory synchronization: Convey MX
    - Heavy overhead
- Move the synchronization on-chip
  - use a CAM to cache recently seen nodes: reduces number of memory accesses
  - map threads (nodes) to engines deterministically
  - deal with load balancing

CAMs as synchronizing caches

CAMs as synchronizing caches

- CAM caches address of all pending memory accesses
  - all accesses are checked in CAM first
    - on miss, address is inserted into CAM
    - on hit, access is deferred
    - read hits get the returned value
    - write hits are deferred => <u>strict serialization</u>

CAMs as synchronizing caches

- CAM caches address of all pending memory accesses
  - all accesses are checked in CAM first
    - on miss, address is inserted into CAM
    - on hit, access is deferred
    - read hits get the returned value
    - write hits are deferred => <u>strict serialization</u>
- some writes need not be done in memory
  - e.g. aggregation: increment count locally ==> reduced traffic to memory
  - very effective on some workloads

CAMs as synchronizing caches

- CAM caches address of all pending memory accesses
  - all accesses are checked in CAM first
    - on miss, address is inserted into CAM
    - on hit, access is deferred
    - read hits get the returned value
    - write hits are deferred => <u>strict serialization</u>
- some writes need not be done in memory
  - e.g. aggregation: increment count locally ==> reduced traffic to memory
  - very effective on some workloads
- each CAM guards a slice of the address space
  - partitioning of the address space
  - deal with load balancing

in-memory aggregation

- A crucial operation for any OLAP workload
- Aggregation is challenging to implement in hardware logic
  - Not all tuples will create a new node (e.g. update existing node)
  - Every tuple reads and writes to the table
- Content Addressable Memories (CAMs)
  - Used to enforce memory locks and merge jobs together
  - Splits nodes among multiple hash tables, need to be merged

*Relation **R***

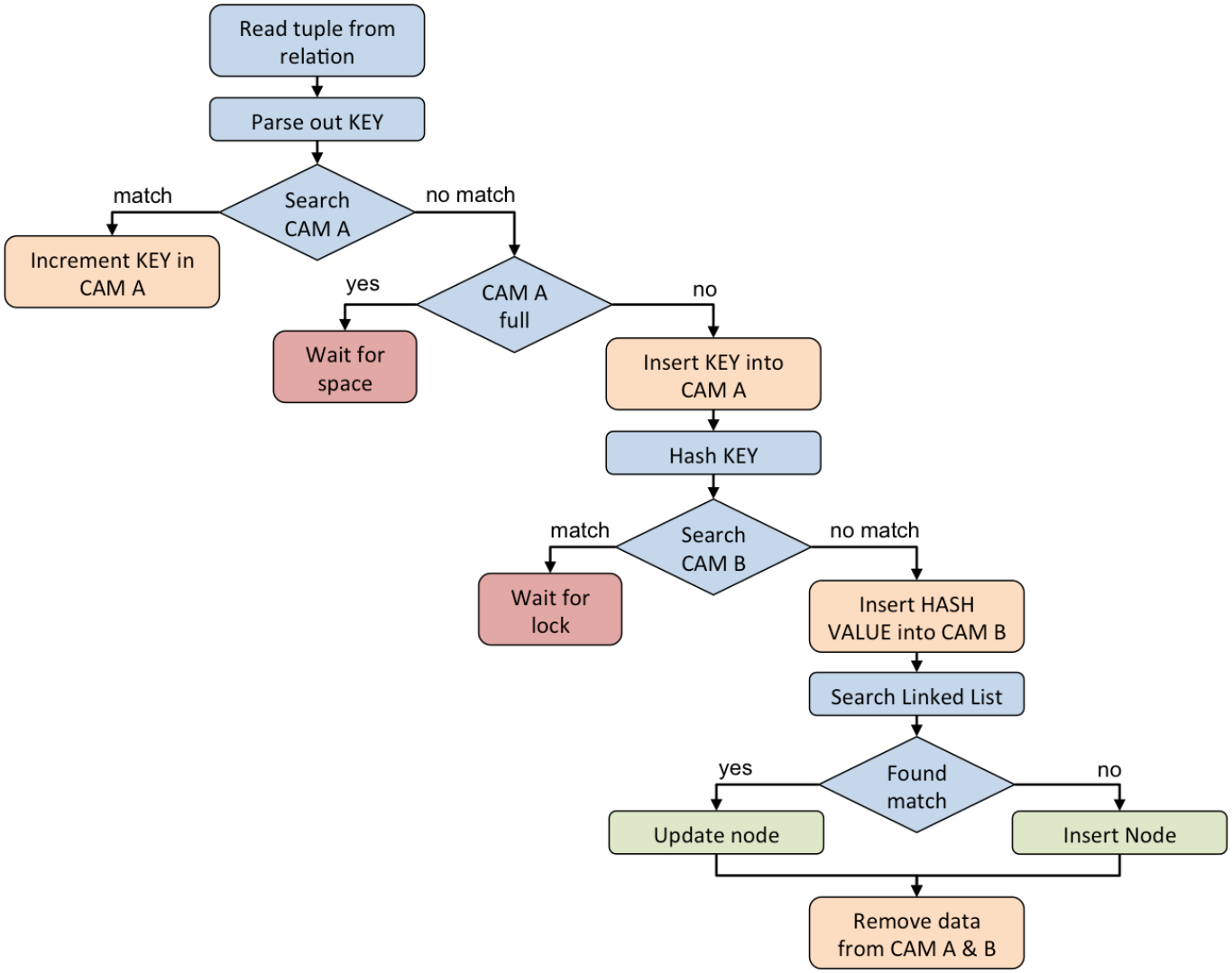
Key	Value
1	123
1	314
2	42
3	7
2	96

Agg Relation

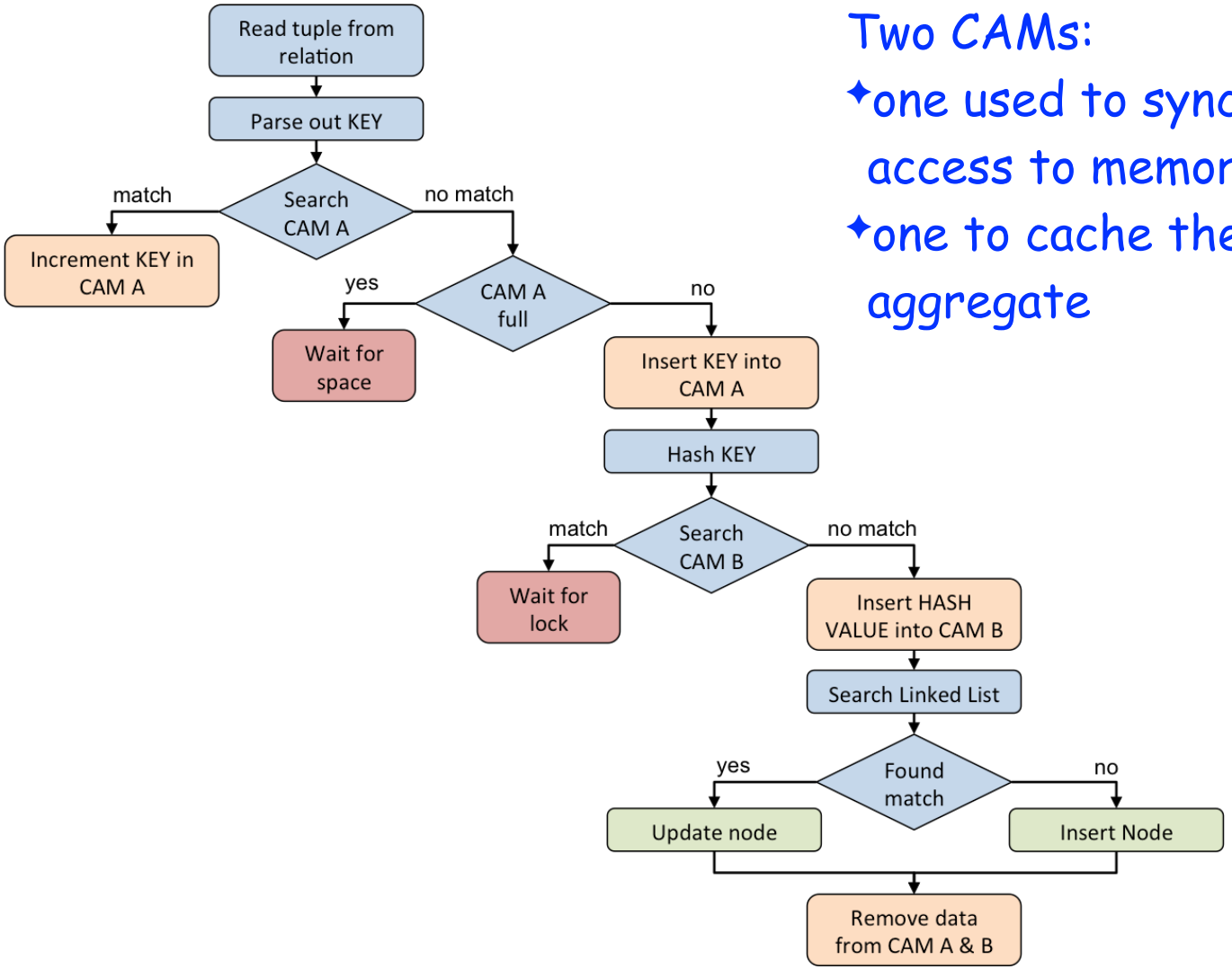
Key	Count
1	2
2	2
3	1

Halstead et al. *FPGA-based Multithreading for In-Memory Hash Joins*, in 7th Biennial Conf. on Innovative Data Systems Research (CIDR'15), Jan. 4-7, 2015, Asilomar, CA.

thread control flow graph



thread control flow graph

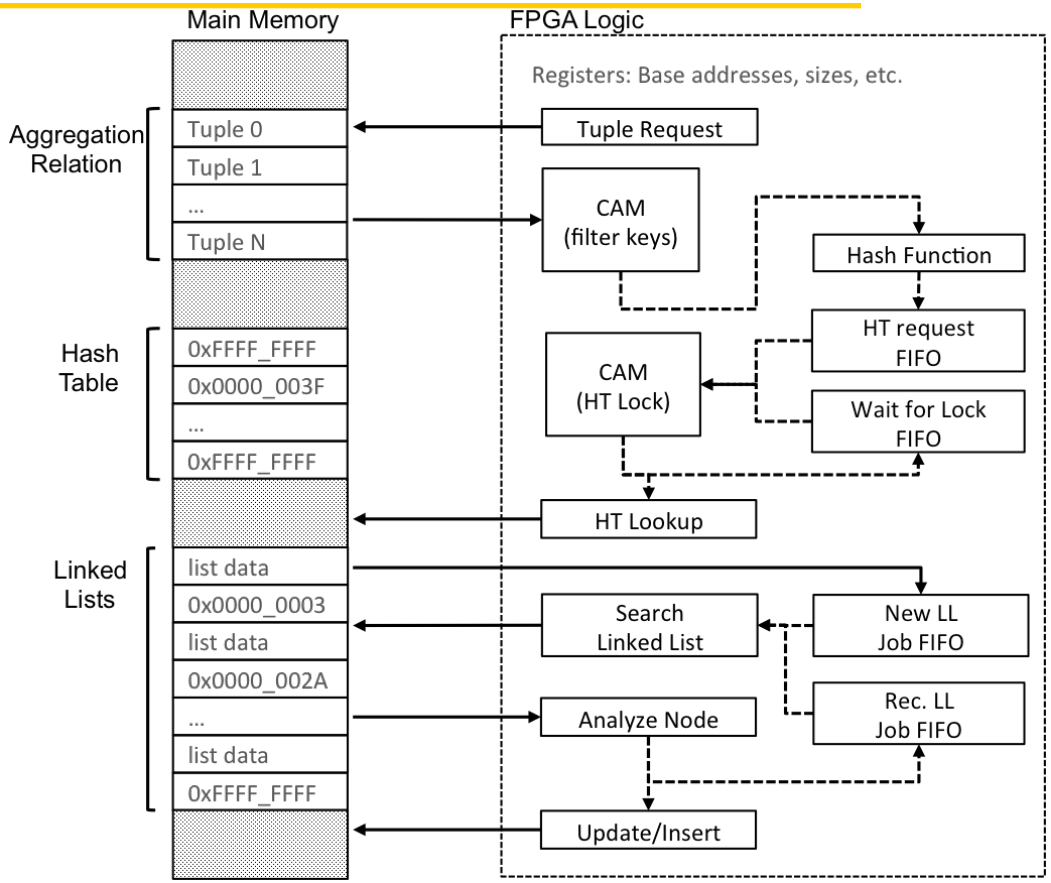


Two CAMs:

- one used to synchronize the access to memory
- one to cache the value of the aggregate

aggregation engine

- **CAM**: Filter Keys
  - Jobs with the same key are merged
- **CAM**: HT Lock
  - Locks individual hash table locations
- **Linked List**
  - Read through to find match
  - Update/Insert node
- Design is limited by the 128 CAM location

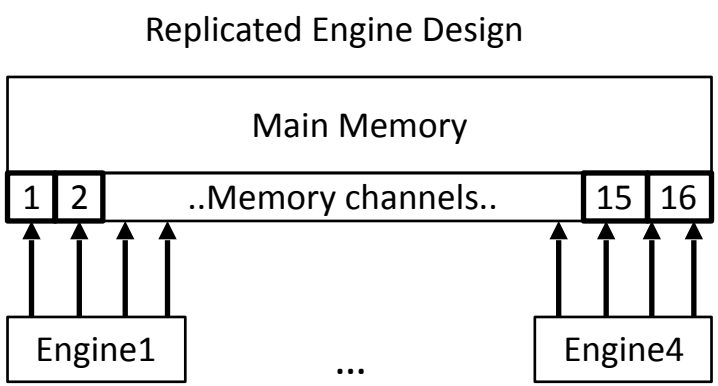


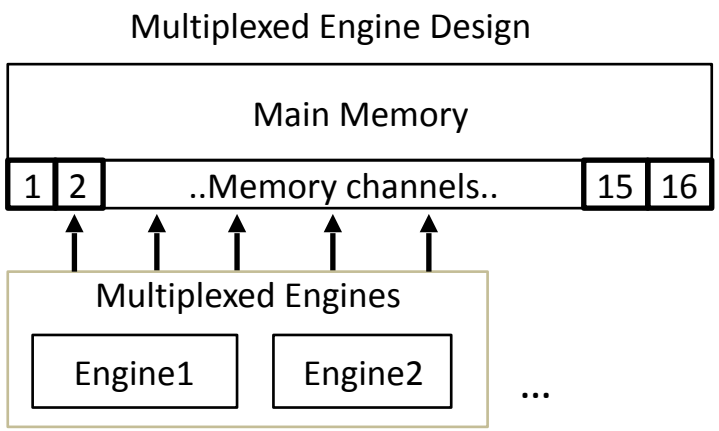
merging operation

- Mapping of nodes to engines done statically
  - bandwidth between FPGAs is not sufficient
  - may becomes the design bottleneck
- Relation keys can be in multiple nodes in separate hash tables
  - keys are not duplicated within a hash table
- After the job is completed
  - multiple hash table must be merged: a streaming operation
  - can be done on FPGA:
    - stream list chains for each hash table bucket
    - merge matching keys together

concurrent FPGA engines

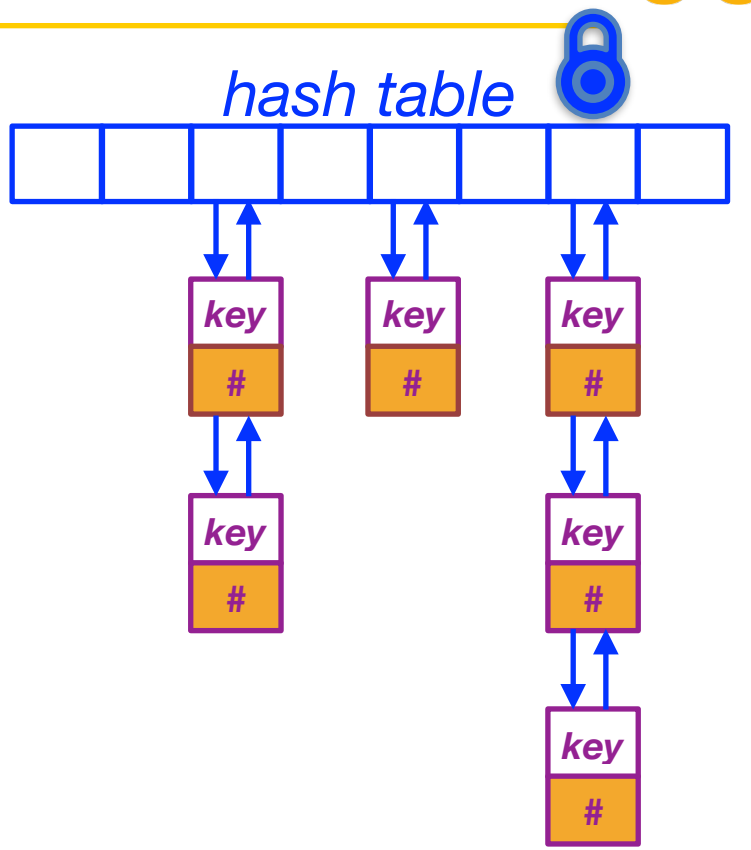
- target platform: Convey HC-2ex
  - 4 Xilinx Virtex 6 FPGAs x 16 Memory channels/FPGA
- two designs
  - replicated cores (4 channels/core, 4 cores/FPGA)
  - multiplexed cores (2.5 channels/core, 6 cores/FPGA) - write channel shared by 2 cores





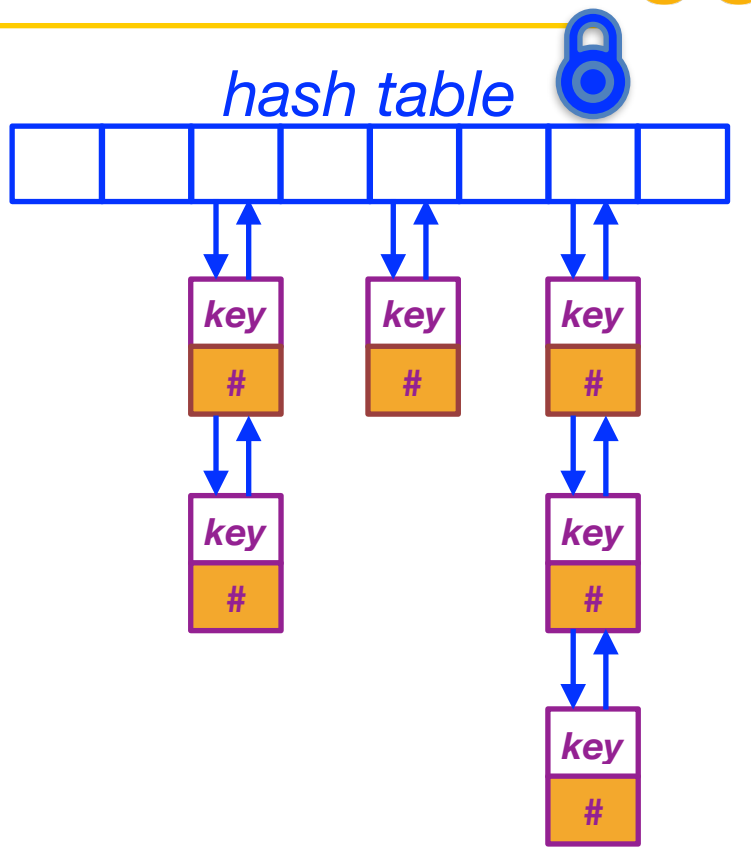
fine v/s coarse-grain locking

fine v/s coarse-grain locking



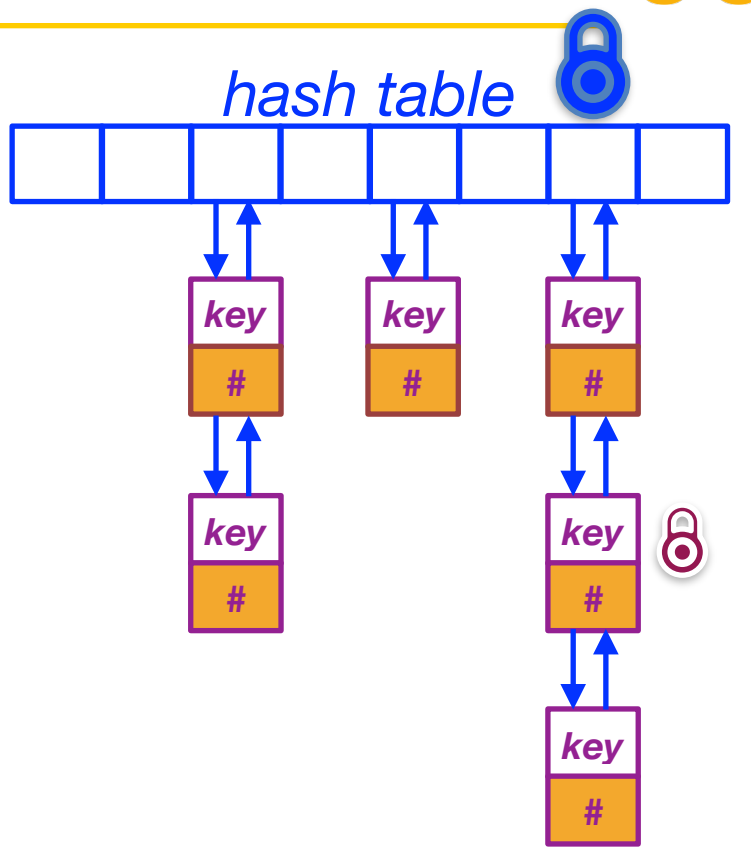
fine v/s coarse-grain locking

- coarse-grain
  - lock on hash table entry
  - fewer locks ==> less parallelism
  - lower pressure on synchronizing CAM



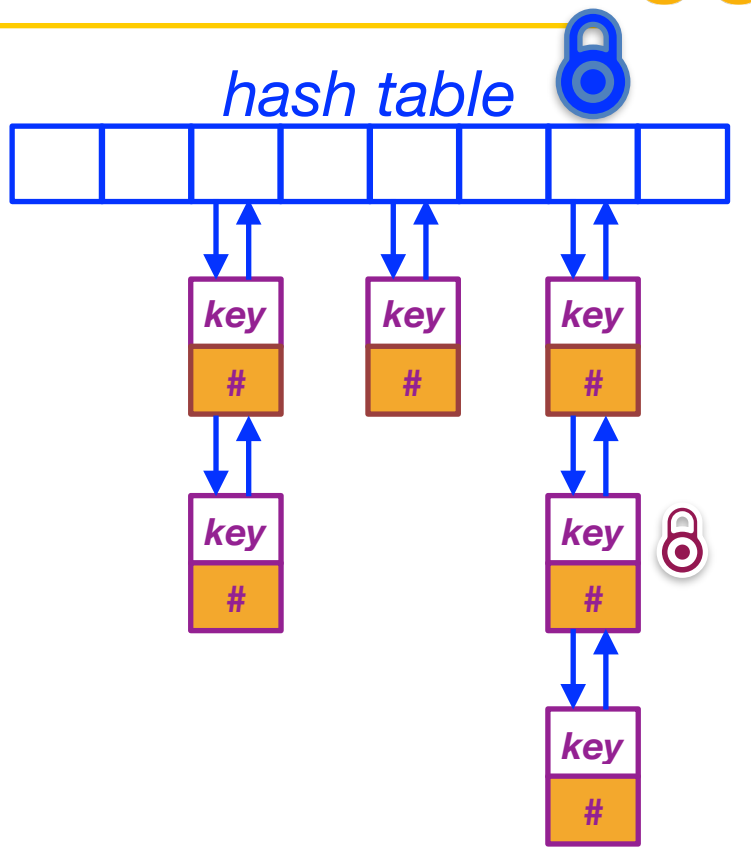
fine v/s coarse-grain locking

- coarse-grain
  - lock on hash table entry
  - fewer locks ==> less parallelism
  - lower pressure on synchronizing CAM



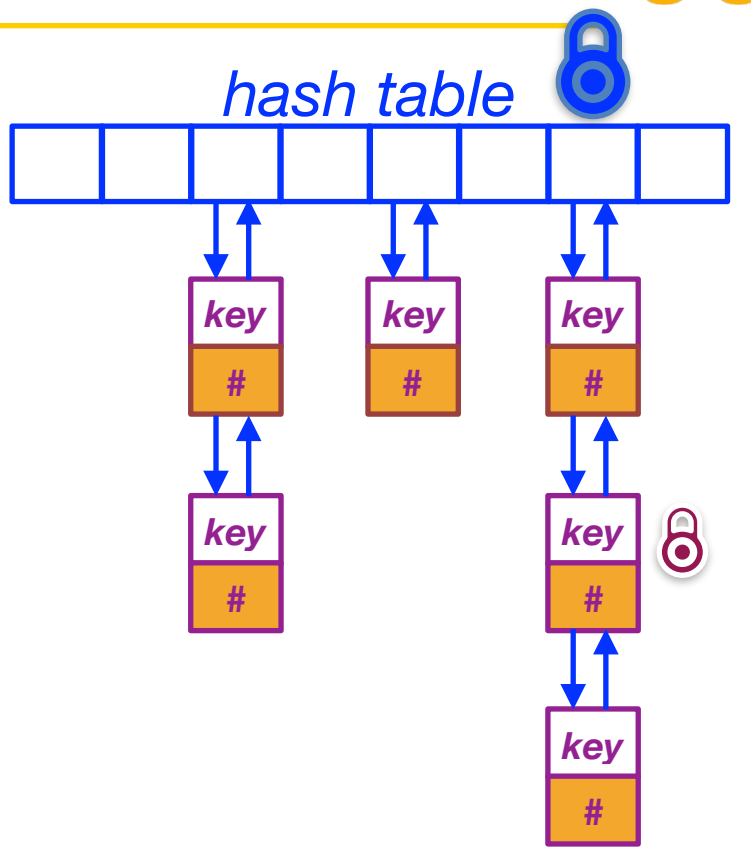
fine v/s coarse-grain locking

- coarse-grain
  - lock on hash table entry
  - fewer locks ==> less parallelism
  - lower pressure on synchronizing CAM
- fine-grain lock
  - lock on linked list entry
  - more parallel filaments
  - higher pressure on CAM



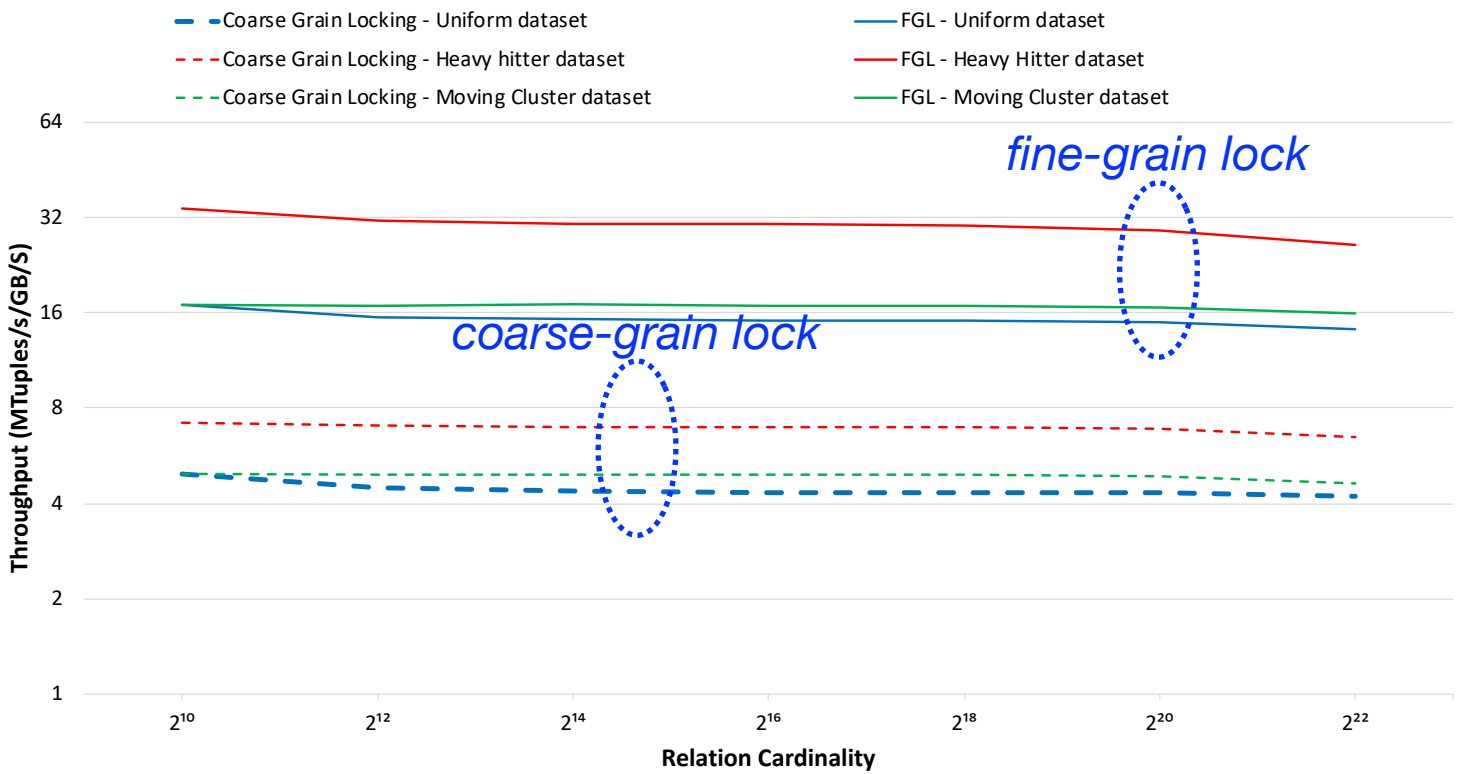
fine v/s coarse-grain locking

- coarse-grain
  - lock on hash table entry
  - fewer locks ==> less parallelism
  - lower pressure on synchronizing CAM
- fine-grain lock
  - lock on linked list entry
  - more parallel filaments
  - higher pressure on CAM



who wins?

experimental evaluation FGL v/s CGL



- fine-grain ~4x higher throughput
- note: constant throughput v/s relation cardinality

comparison to software

- 5 aggregation algorithms
  - Independent Tables
  - Shared Table
  - Partition & Aggregate
  - Hybrid Table
  - Partition with Local Aggregation Table

Independent Tables, Shared Table: Hardware-oblivious [VLDB'07]

Partition & Aggregate: Hardware-conscious [VLDB'07]

Hybrid Table, Partition with Local Aggregation Table: Hybrid [DAMON'11]

[ICDE'13] Balkesen, C. et al. Main-memory Hash Joins on Multi-core CPUs: Tuning to the underlying hardware.
[VLDB'07] J. Cieslewicz et al. Adaptive Aggregation on Chip Multiprocessors.
[DAMON'11] Y. Ye et al. Scalable aggregation on multicore processors

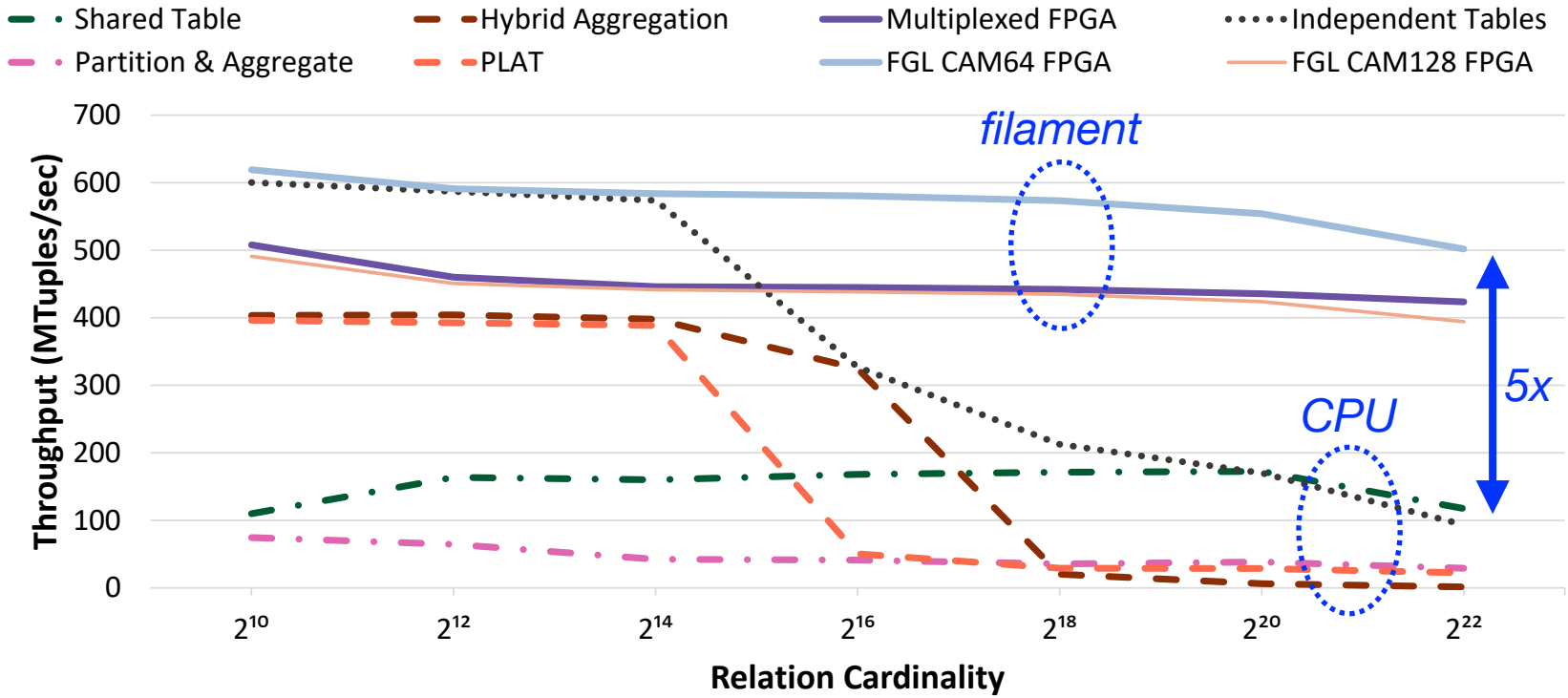
experimental evaluation

- Software Approaches
  - Individual Table
  - Shared Table (atomic)
  - Hybrid Table
  - Partitioning
  - PLAT
- Datasets
  - Uniform, Zipf 0.5, Heavy Hitter, Moving Cluster, Self Similar
- 8M to 64M tuples/ benchmarks
- 1K to 4M cardinality

Hardware Region	
FPGA board	*Virtex-6 760*
# FPGAs	*2*
Clock Freq.	*150 MHz*
Engines per FPGA	*4 / 3*
Memory Channels	*32*
Memory Bandwidth (total)	*38.4 GB/s*

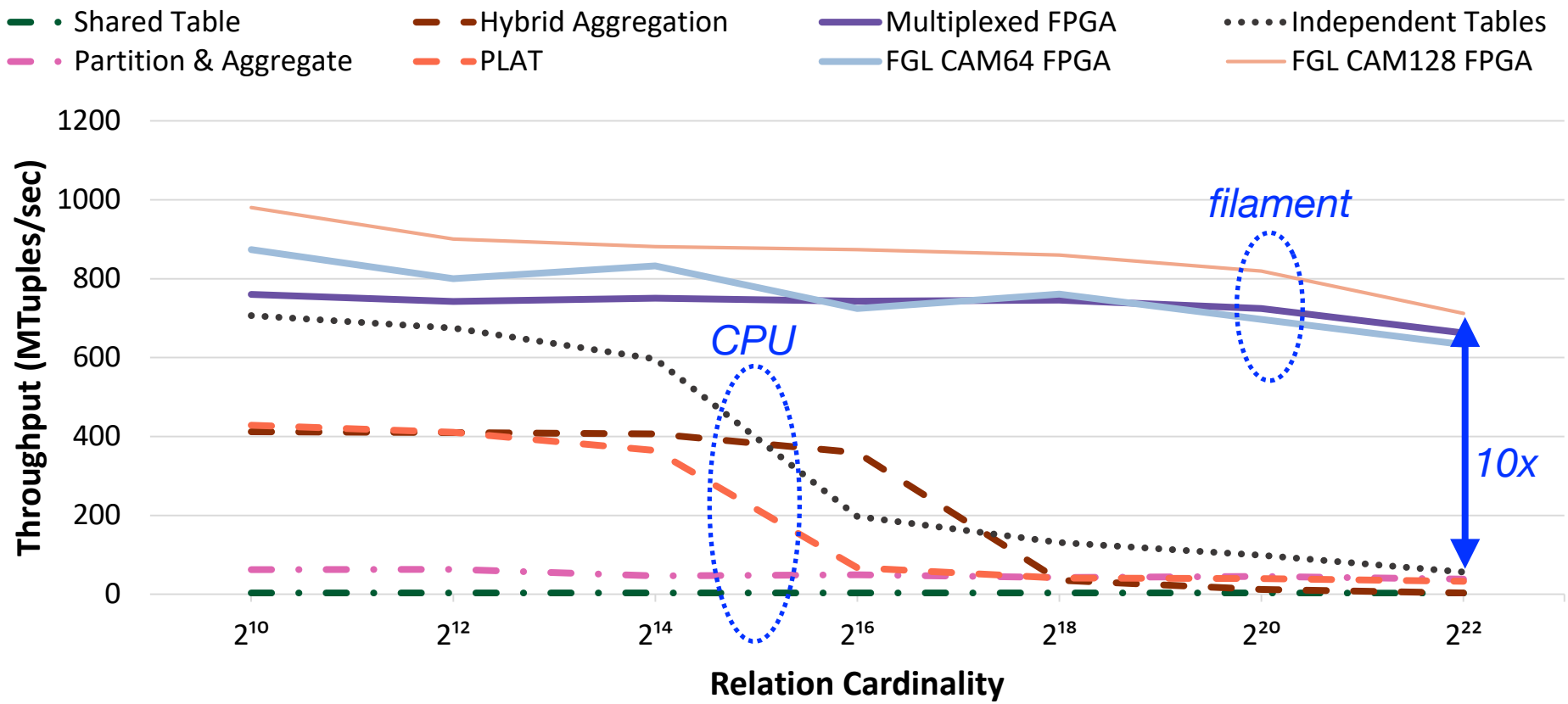
Software Region	
CPU	*Intel Xeon E5-2643*
# CPUs	*1*
Cores / Threads	*4 / 8*
Clock Freq.	*3.3 GHz*
L3 Cache	*10 MB*
Memory Bandwidth (total)	*51.2 GB/s*

uniform workload



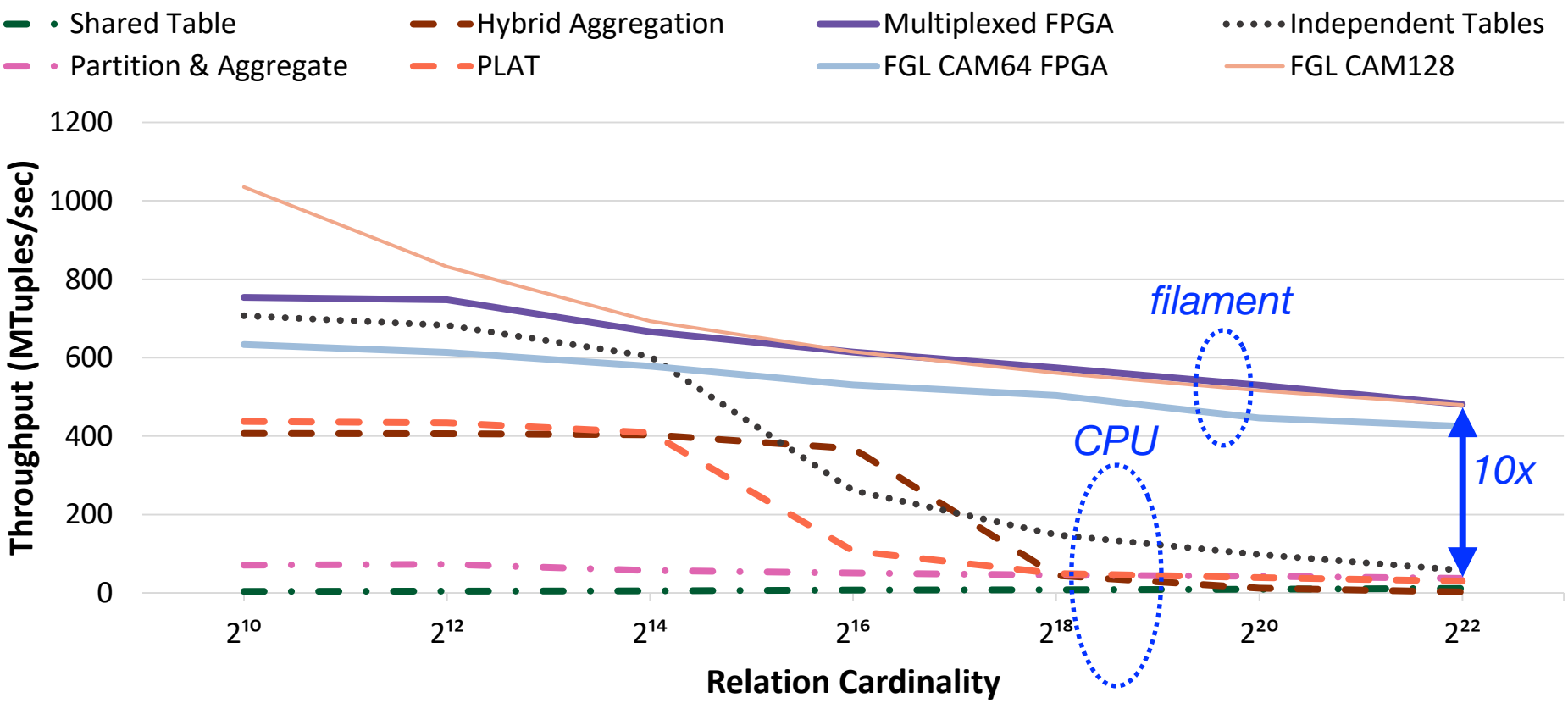
Uniform - 256 M tuples

heavy-hitter workload



Heavy Hitter - 256 M tuples

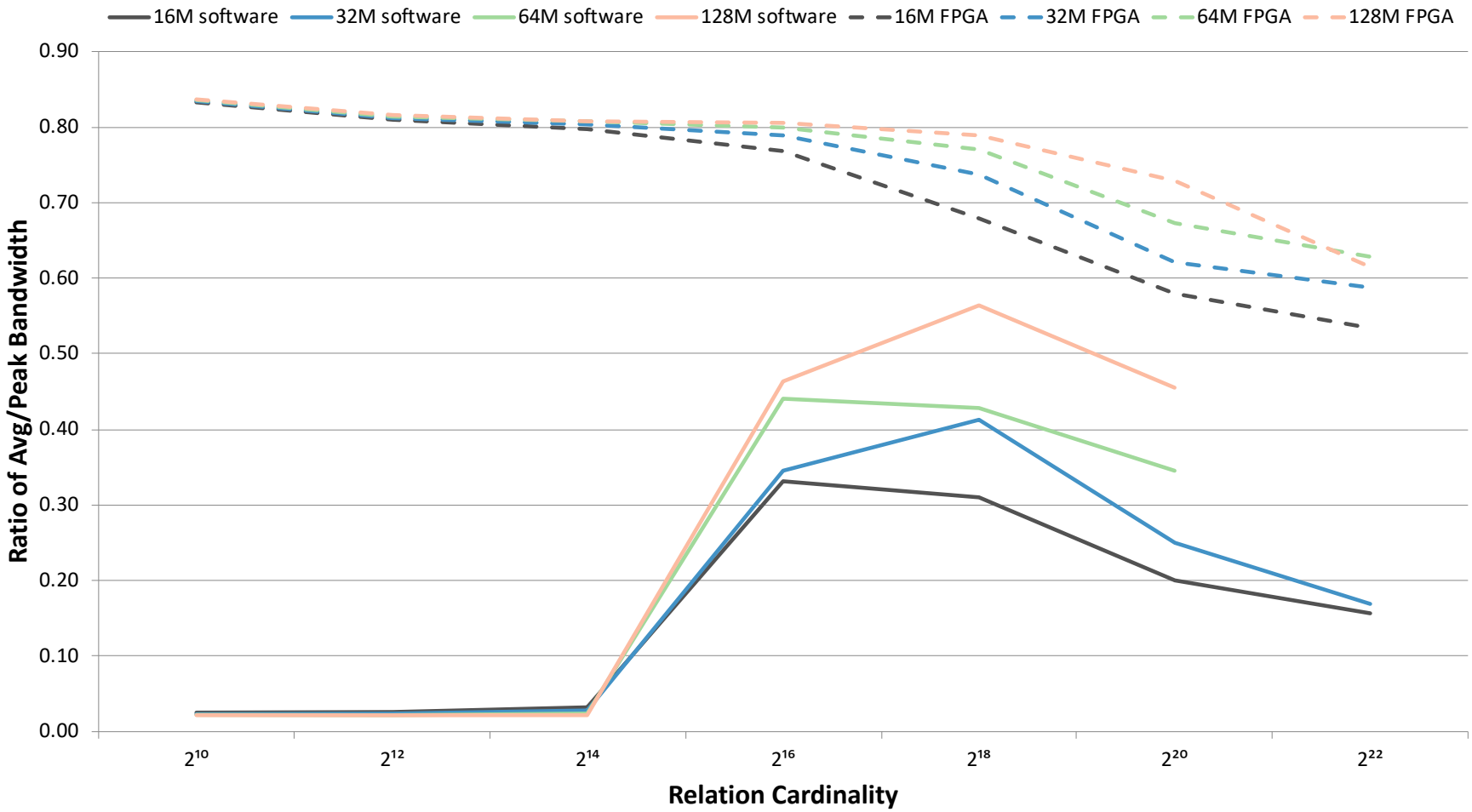
self-similar workload



SelfSimilar — 256 M tuples

bandwidth utilization

- FPGA implementation achieves much higher memory bandwidth utilization



outline

- background
- filament execution model
- ~~application - SpMV~~
- application - Selection
- application - Group by aggregation
- **conclusion**

other results, past and on-going

- Big data is: **BIG** and multi-dimensional
  - access patterns to data are extremely irregular
  - no ideal storage scheme - constant poor locality
- Other applications
  - string matching, hash join, group by aggregation, selection, outer joins, sorting
- Challenges
  - inter-filament synchronization
  - dynamic load re-balancing (at run time)

publications

http://www.cs.ucr.edu/~najjar/publications.html

- P. Budhkar, I. Absalyamov, V. Zois, S. Windh, W. A. Najjar and V. J. Tsotras. *Accelerating In-Memory Database Selections Using Latency Masking Hardware Threads*, in ACM TACO, 2019.
- I. Absalyamov, R. J. Halstead, P. Budhkar, W. A. Najjar, S. Windh, V. J. Tsotras. *FPGA-Accelerated Group-by Aggregation Using Synchronizing Caches*. 12th Int. Workshop on Data Management on New Hardware (DaMoN 2016), Co-located with ACM SIGMOD/PODS, San Francisco, USA, June 27, 2016.
- S. Windh, P. Budhkar and W. A. Najjar. *CAMs as Synchronizing Caches for Multithreaded Irregular Applications on FPGAs*. ICCAD 2015: 331-336
- R. J. Halstead, I. Absalyamov, W. A. Najjar and V. J. Tsotras. *FPGA-based Multithreading for In-Memory Hash Joins*, in 7th Biennial Conference on Innovative Data Systems Research (CIDR 15), January 4-7, 2015, Asilomar, California.
- E. B. Fernandez, J. Villarreal, S. Lonardi, and W. A. Najjar. *FHAST: FPGA-based acceleration of Bowtie in hardware*, in IEEE/ACM Trans. on Computational Biology and Bioinformatics, # 99, Feb. 2015.
- R. J. Halstead, W. A. Najjar and O. Huseini. *SpVM Acceleration with Latency Masking Threads on FPGAs*. Technical Report UCR-CSE-2014-04001

conclusion

- advantages of filament execution
  - very small state allows
    - fast context switching between filaments
    - small storage for waiting queues
  - masking the latency of memory and non volatile storage
    - no reliance on locality, no caches
    - lower energy consumption
- limitations: custom or semi-custom accelerators

thank you!

questions?