Concurrent Data Structures for Near-Memory Computing

Zhiyu Liu (Brown)

Irina Calciu (VMware Research)

Maurice Herlihy (Brown)

Onur Mutlu (ETH)

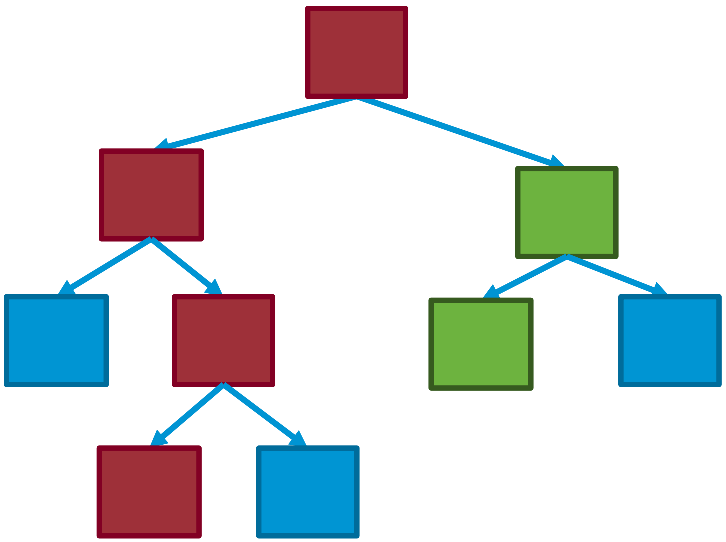
© 2017 VMware Inc. All rights reserved.

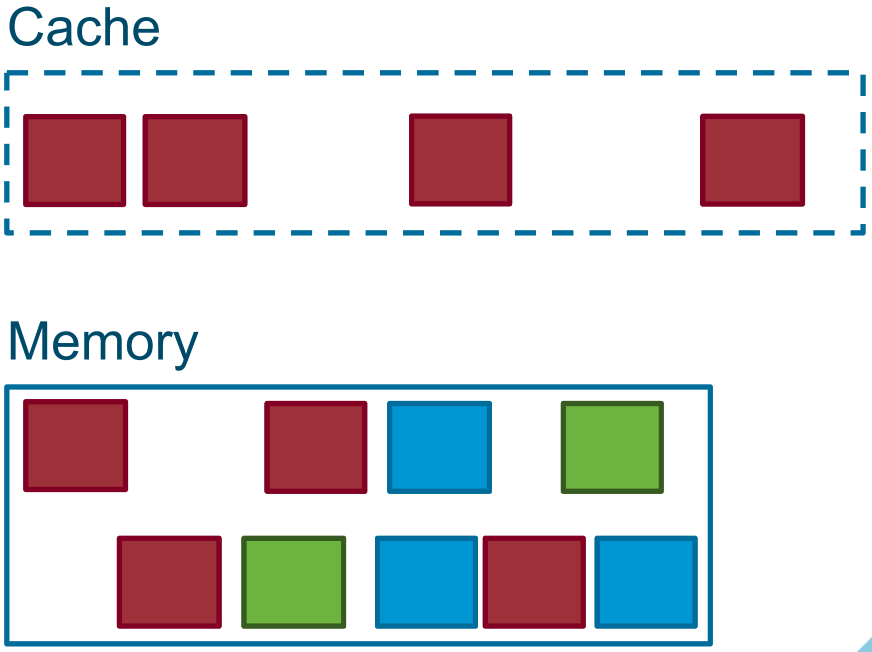
Concurrent Data Structures

Are used everywhere: kernel, libraries, applications

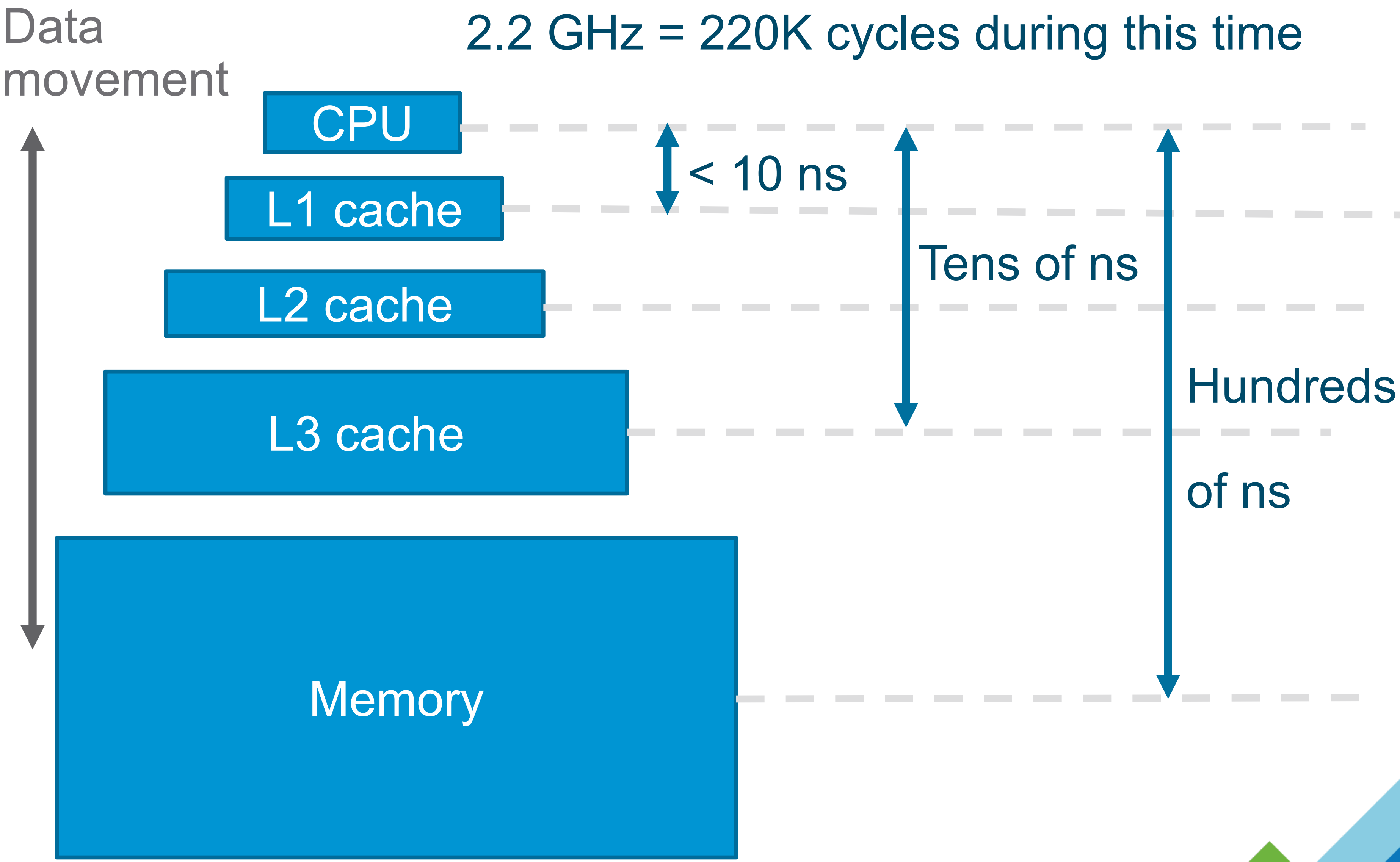
Issues:

- Difficult to design and implement
- Data layout and memory/cache hierarchy play crucial role in performance





The Memory Wall

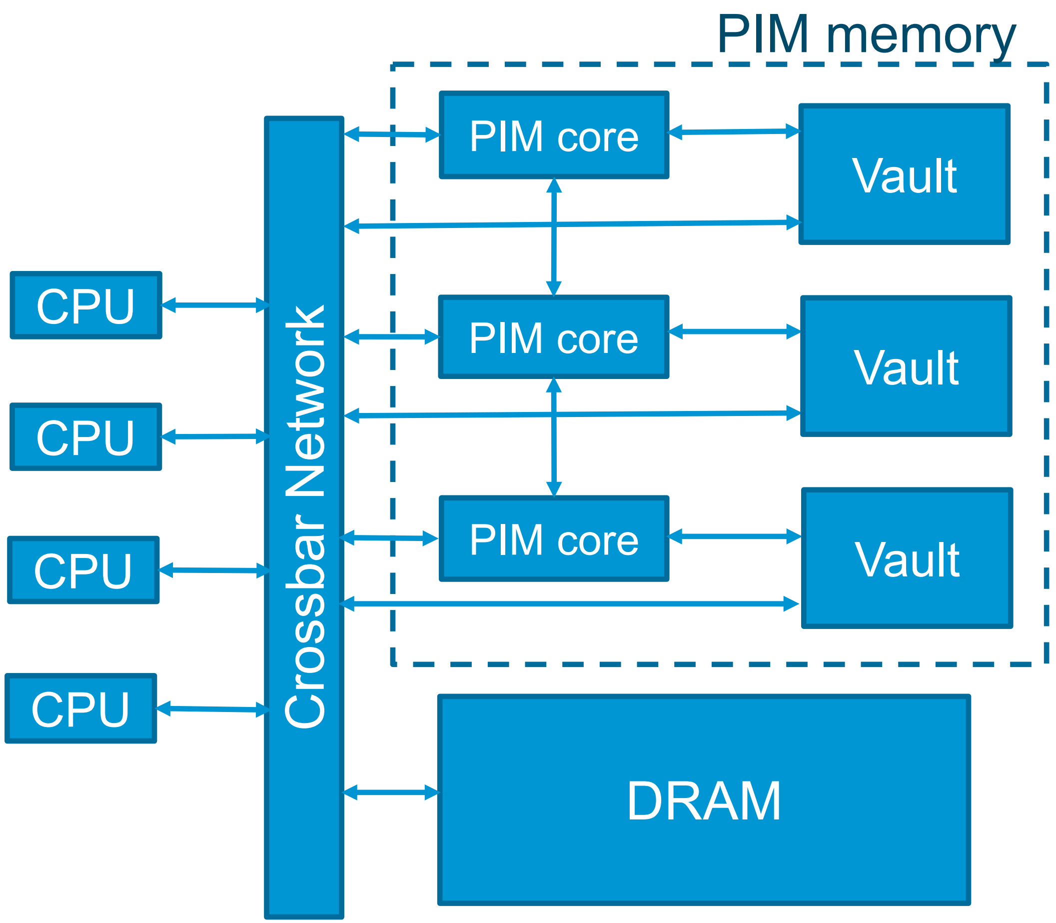


Near Memory Computing

- Also called Processing In Memory (PIM)
- Avoid data movement by doing computation in memory
- Old idea
- New advances in 3D integration and die-stacked memory
- Viable in the near future

Near Memory Computing: Architecture

- Vaults: memory partitions
- PIM cores: lightweight
  - Fast access to its own vault
- Communication
  - Between a CPU and a PIM
  - Between PIMs
  - Via messages sent to buffers



Data Structures + Hardware

- Tight integration between algorithmic design and hardware characteristics
- Memory becomes an active component in managing data
- Managing data structures in PIM
  - Old work: pointer chasing for sequential data structures
  - Our work: concurrent data structures

Goals: PIM Concurrent Data Structures

1. How do PIM data structures compare to state-of-the-art concurrent data structures?

2. How to design efficient PIM data structures?

LOW **CONTENTION** HIGH

Pointer-chasing ⟷ Cacheable

Goals: PIM Concurrent Data Structures

1. How do PIM data structures compare to state-of-the-art concurrent data structures?

2. How to design efficient PIM data structures?

LOW **CONTENTION** HIGH

←——————————————————→

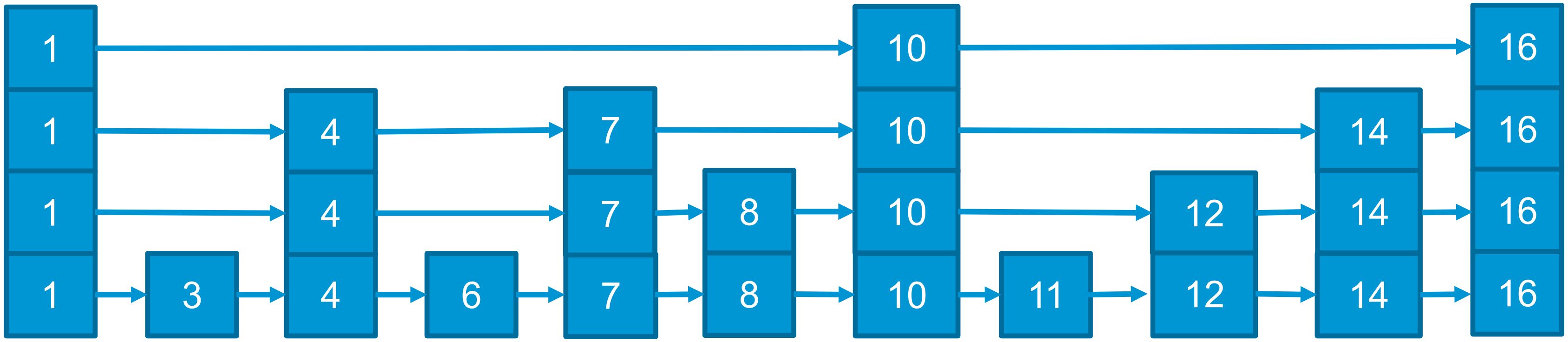
Pointer-chasing

e.g., uniform distribution

skiplist & linkedlist

Cacheable

Pointer Chasing Data Structures



Goals: PIM Concurrent Data Structures

1. How do PIM data structures compare to state-of-the-art concurrent data structures?

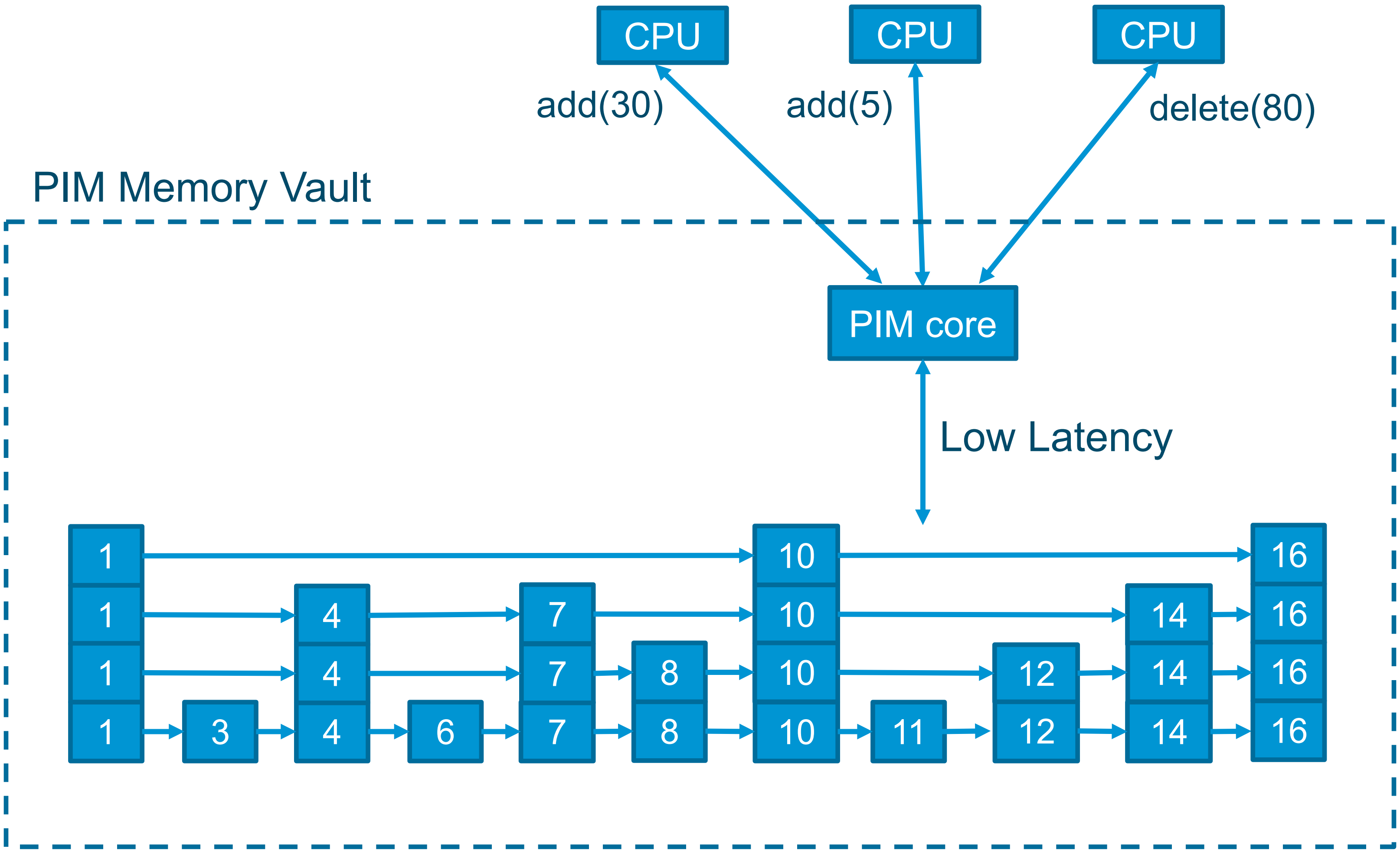
2. How to design efficient PIM data structures?

LOW **CONTENTION** HIGH

Pointer-chasing
e.g., uniform distribution
skiplist & linkedlist

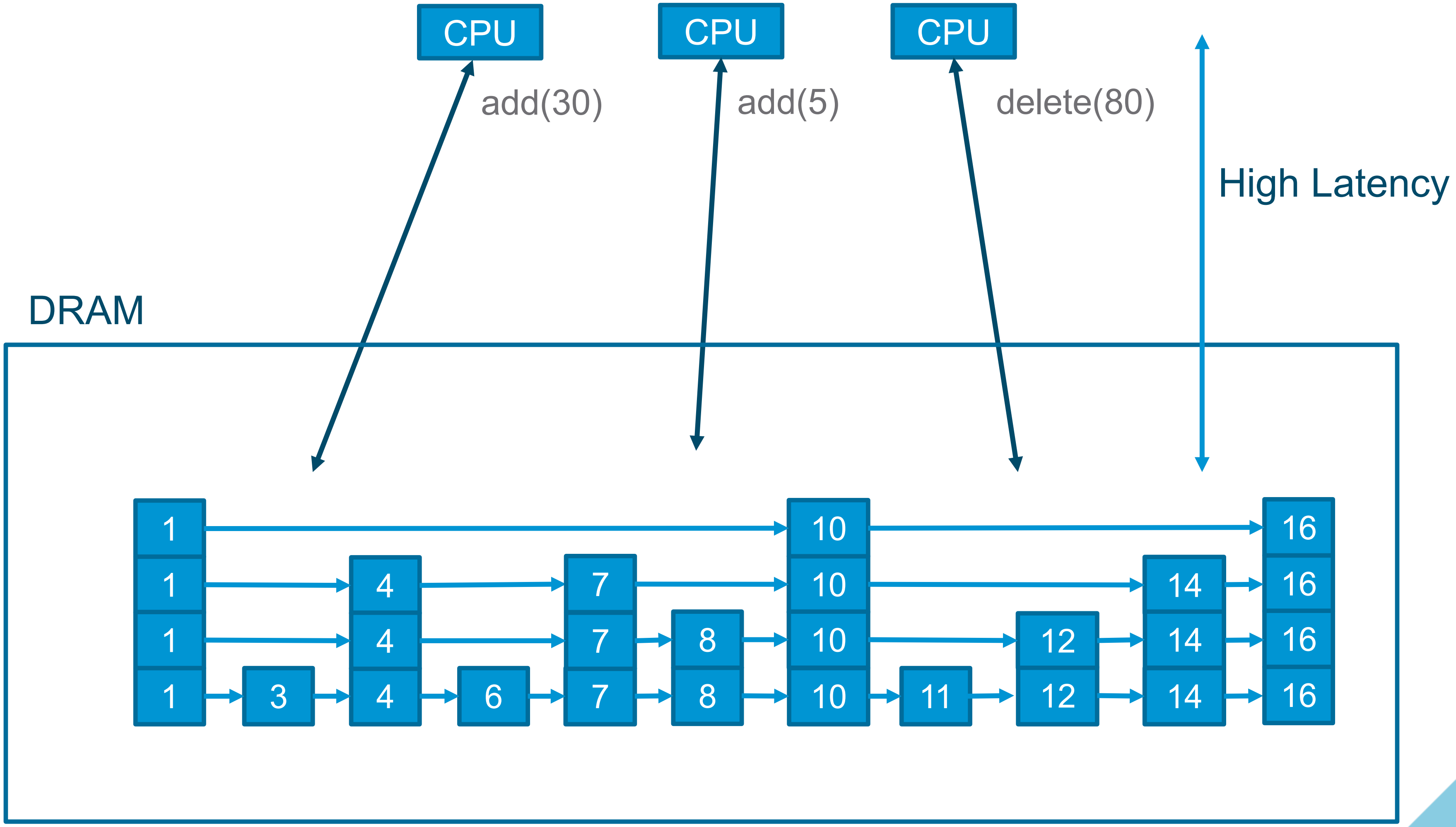
Cacheable

Naïve PIM Skiplist

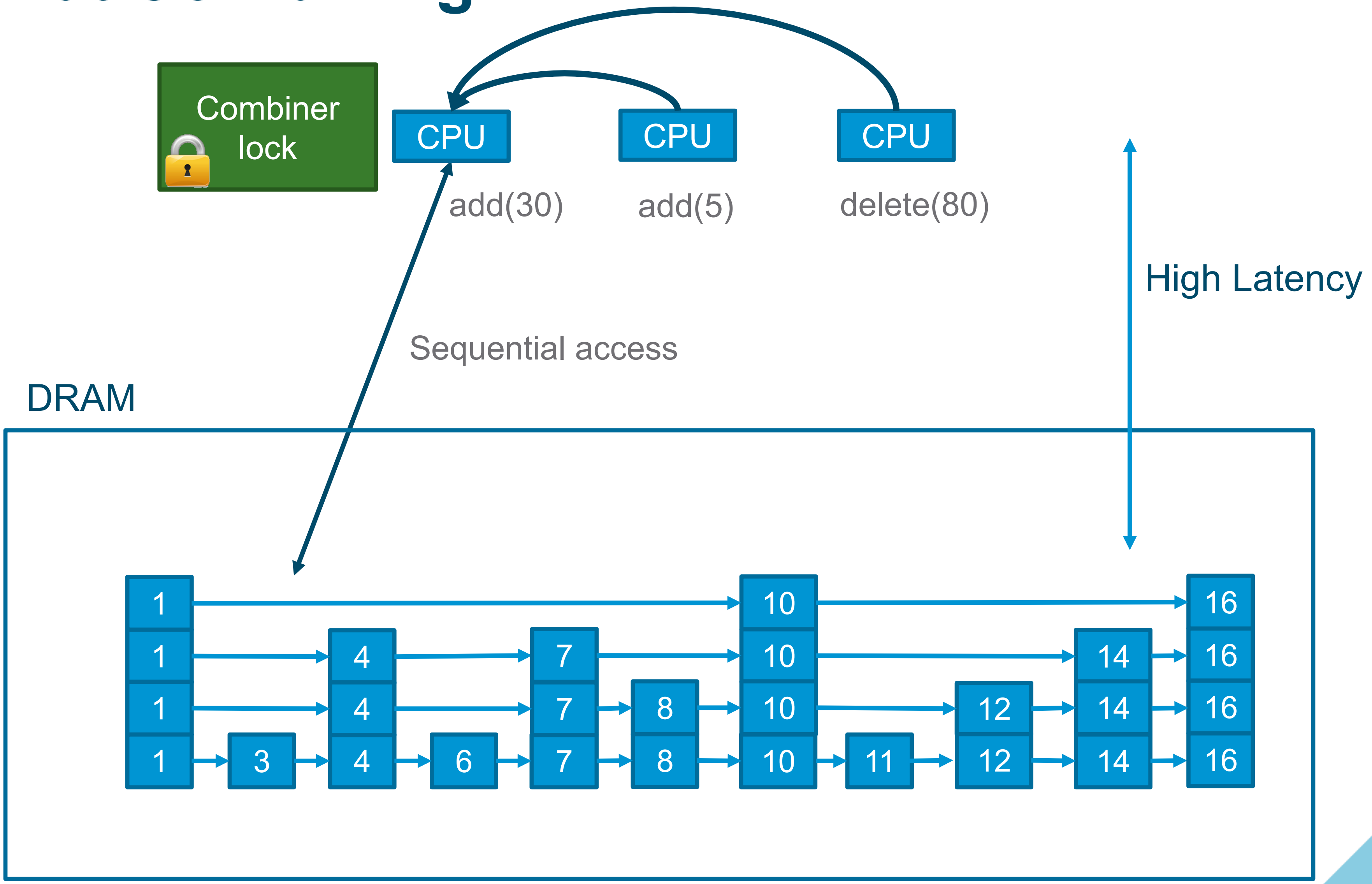


vmware

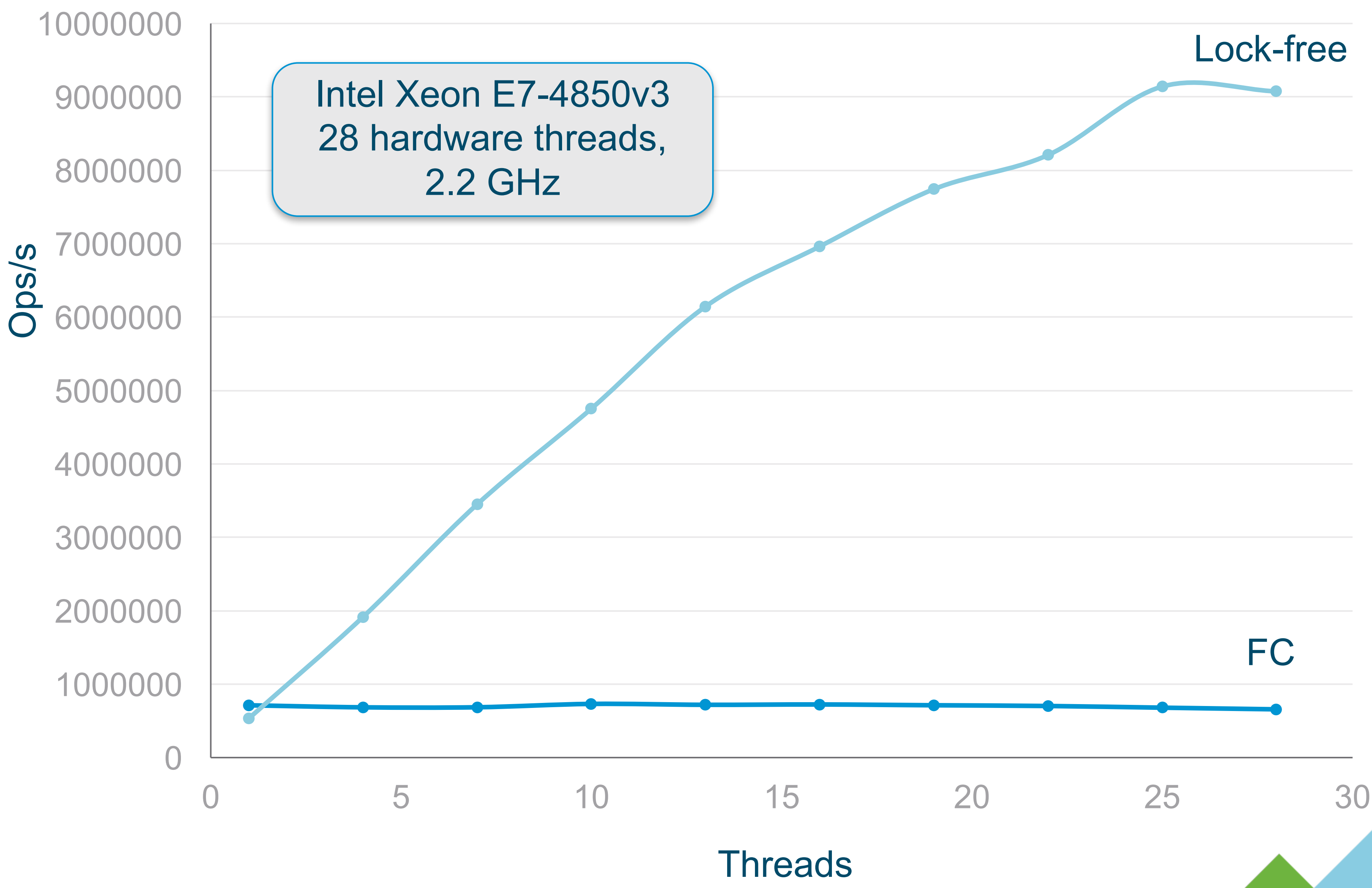
Concurrent Data Structures



Flat Combining



Skiplist Throughput



PIM Performance

N	Size of the skiplist
p	Number of processes
$\mathcal{L}_{CPU}$	Latency of a memory access from the CPU
$\mathcal{L}_{LLC}$	Latency of a LLC access
$\mathcal{L}_{ATOMIC}$	Latency of an atomic instruction (by the CPU)
$\mathcal{L}_{PIM}$	Latency of a memory access from the PIM core
$\mathcal{L}_{MSG}$	Latency of a message from the CPU to the PIM core

PIM Performance

$$\mathcal{L}_{CPU} = r1\,\mathcal{L}_{PIM} = r2\,\mathcal{L}_{LLC} \qquad r1 = r2 = 3$$

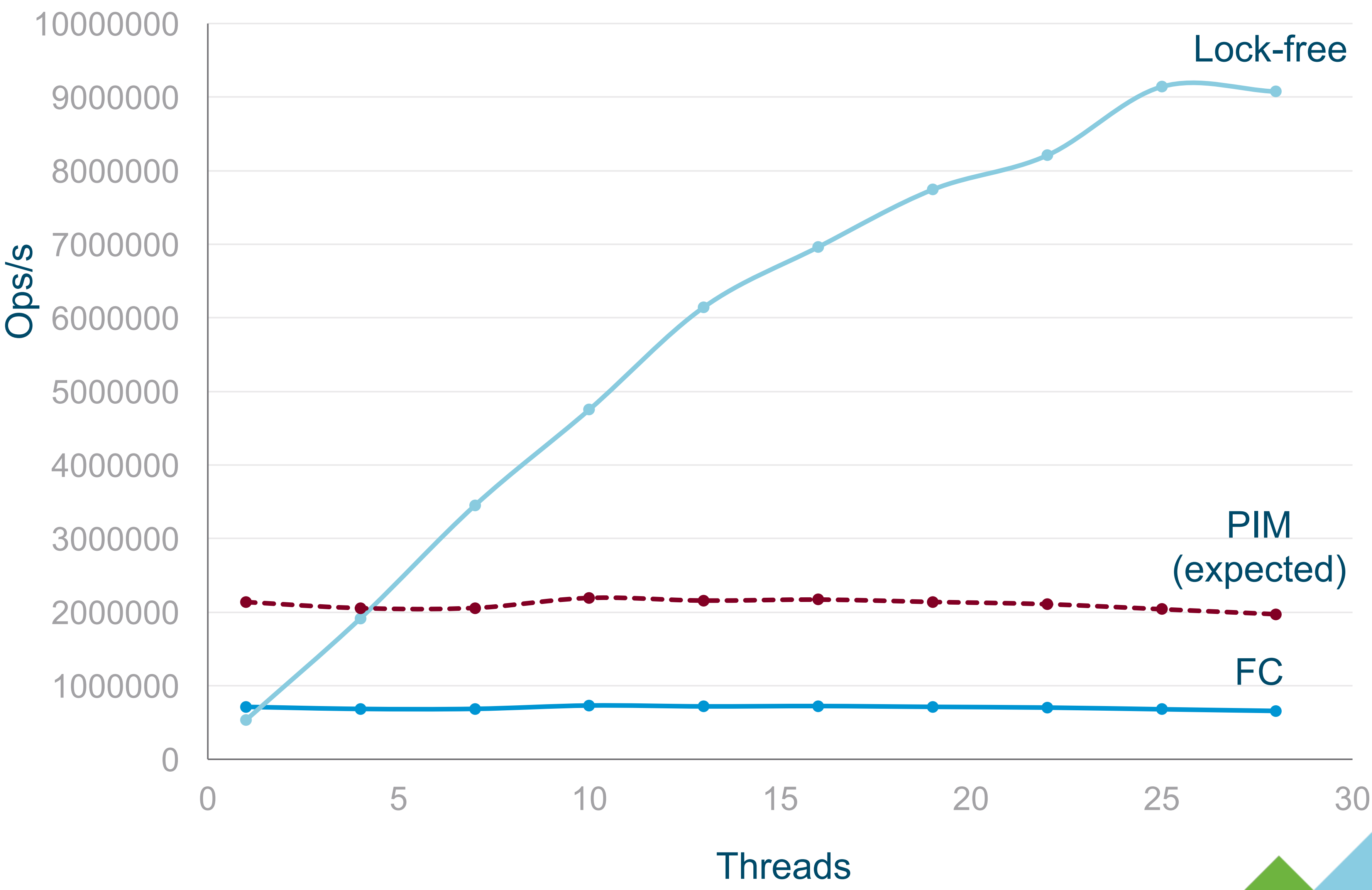
$$\mathcal{L}_{MSG} = \mathcal{L}_{CPU}$$

PIM Performance

Algorithm	Throughput
Lock-free	$p / (\mathcal{B} * \mathcal{L}_{CPU})$
Flat Combining (FC)	$1 / (\mathcal{B} * \mathcal{L}_{CPU})$
PIM	$1 / (\mathcal{B} * \mathcal{L}_{PIM} + \mathcal{L}_{MSG})$

$\mathcal{B}$ = average number of nodes accessed during a skiplist operation

Skiplist Throughput



Goals: PIM Concurrent Data Structures

1. How do PIM data structures compare to state-of-the-art concurrent data structures?

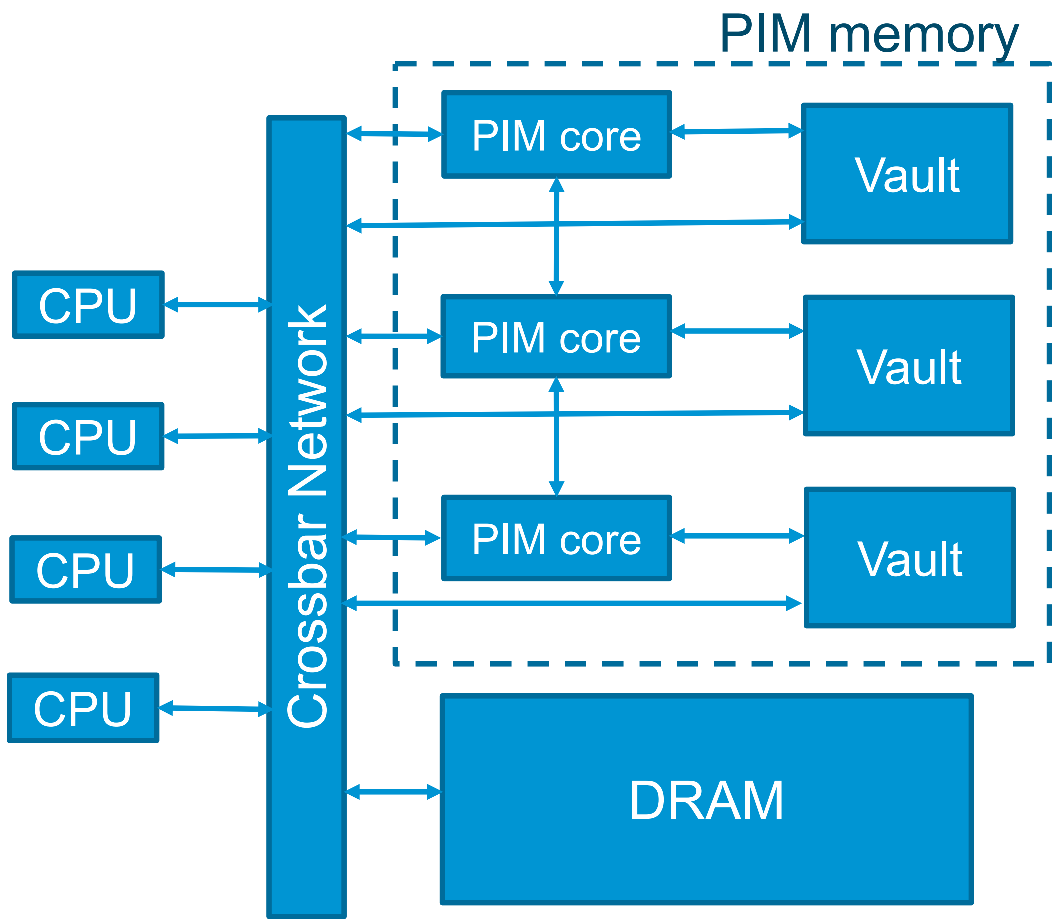
2. How to design efficient PIM data structures?

LOW **CONTENTION** HIGH

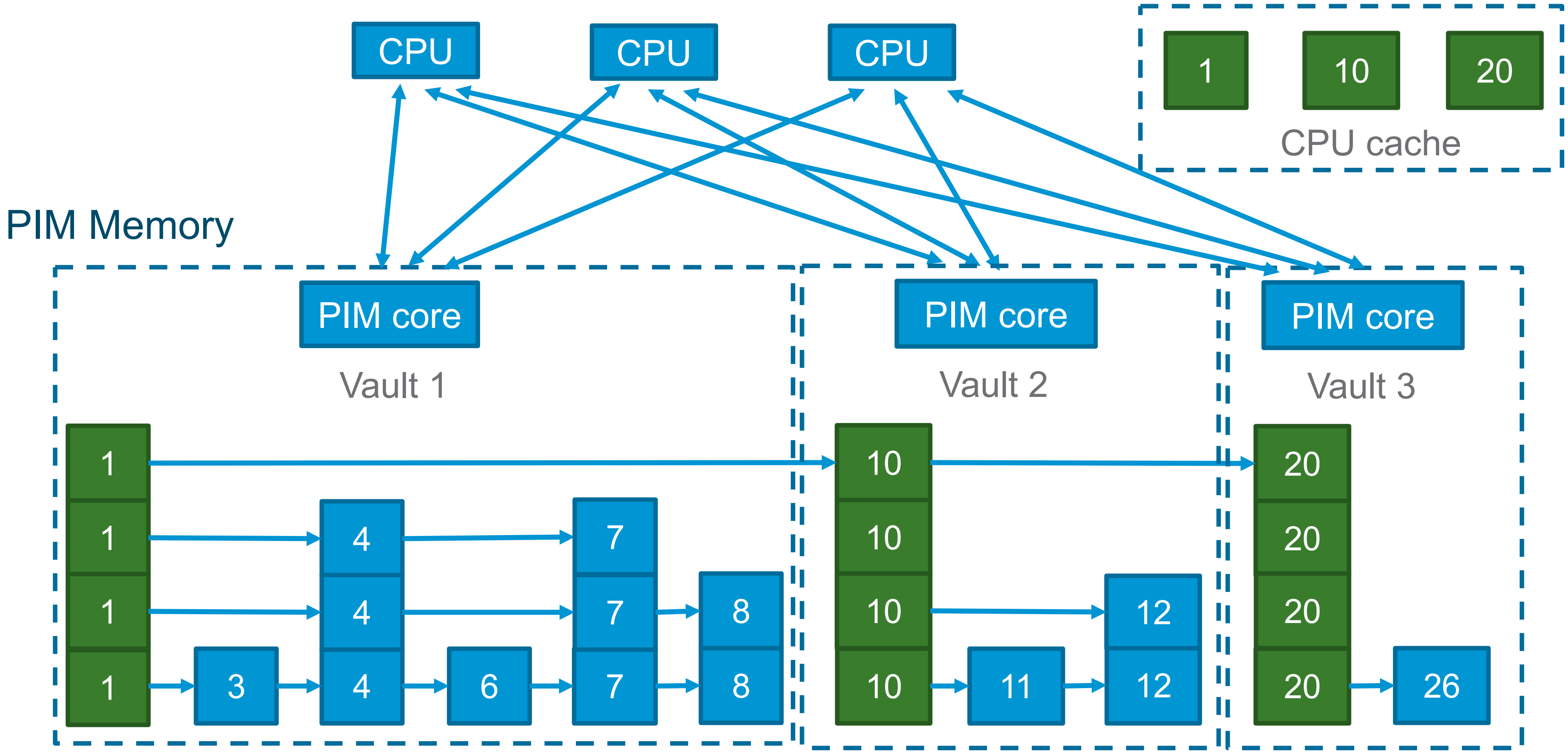
Pointer-chasing
e.g., uniform distribution
skiplist & linkedlist

Cacheable

New PIM algorithm: Exploit Partitioning



PIM Skiplist w/ Partitioning



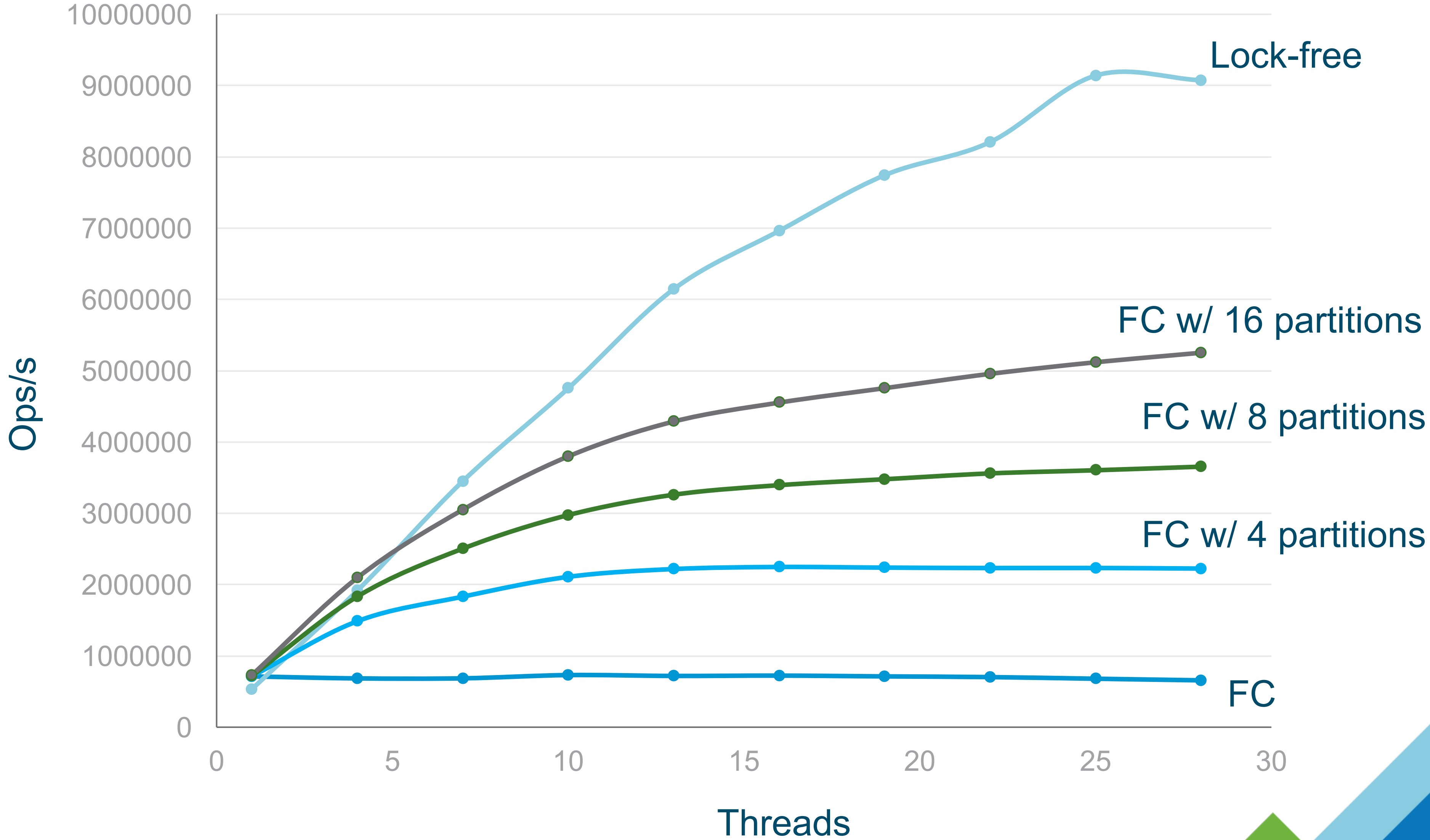
PIM Performance

Algorithm	Throughput
Lock-free	$p / (\mathcal{B} * \mathcal{L}_{CPU})$
Flat Combining (FC)	$1 / (\mathcal{B} * \mathcal{L}_{CPU})$
PIM	$1 / (\mathcal{B} * \mathcal{L}_{PIM} + \mathcal{L}_{MSG})$
FC + k partitions	$k / (\mathcal{B} * \mathcal{L}_{CPU})$
PIM + k partitions	$k / (\mathcal{B} * \mathcal{L}_{PIM} + \mathcal{L}_{MSG})$

$\mathcal{B}$ = average number of nodes accessed during a skiplist operation

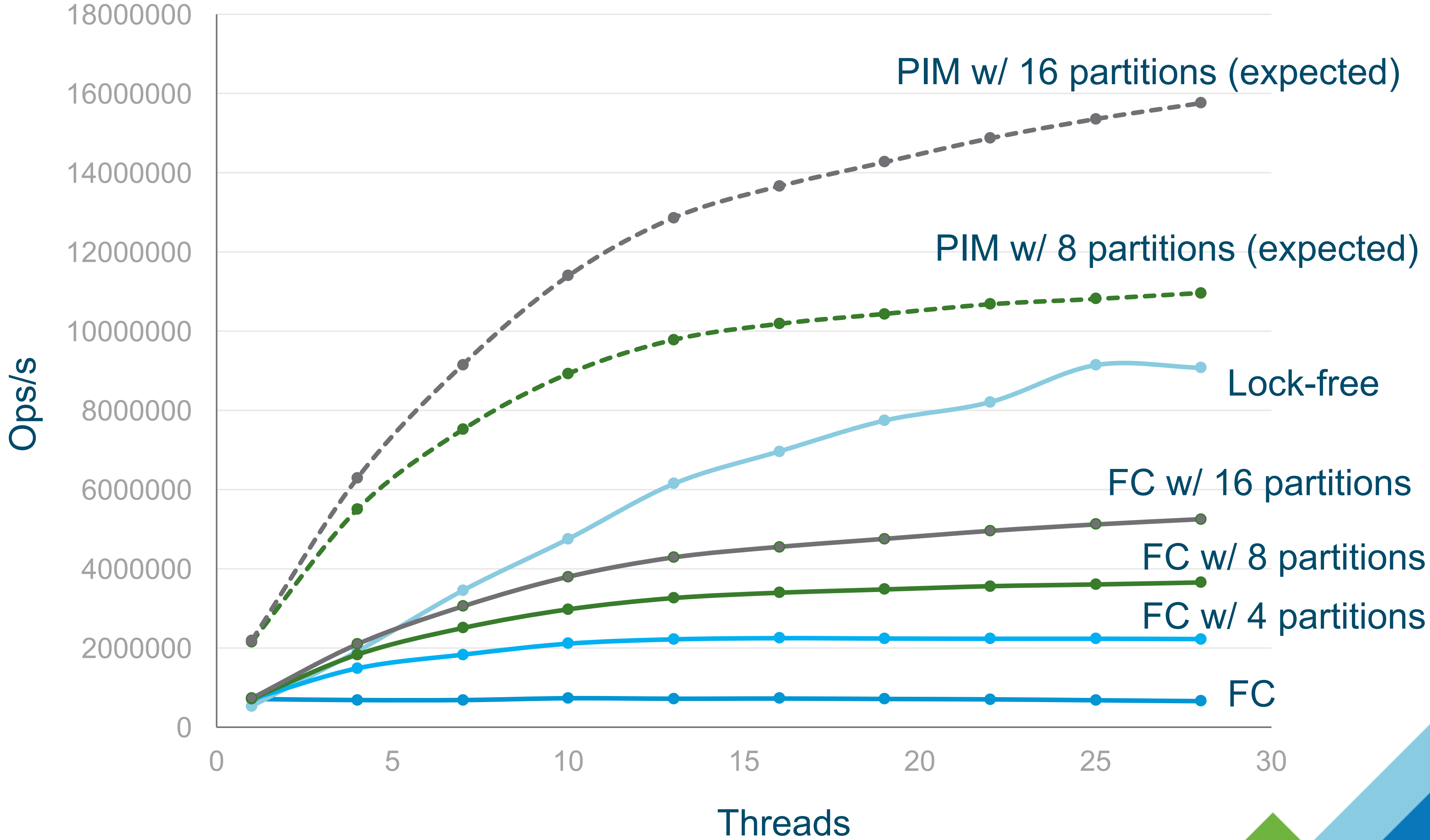
vmware®

Skiplist Throughput



vmware

Skiplist Throughput



Goals: PIM Concurrent Data Structures

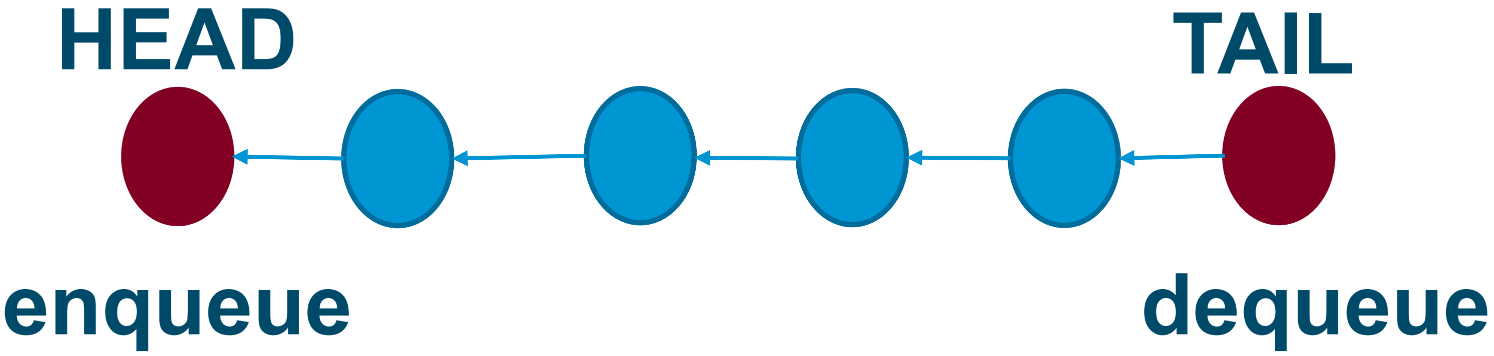
1. How do PIM data structures compare to state-of-the-art concurrent data structures?

2. How to design efficient PIM data structures?

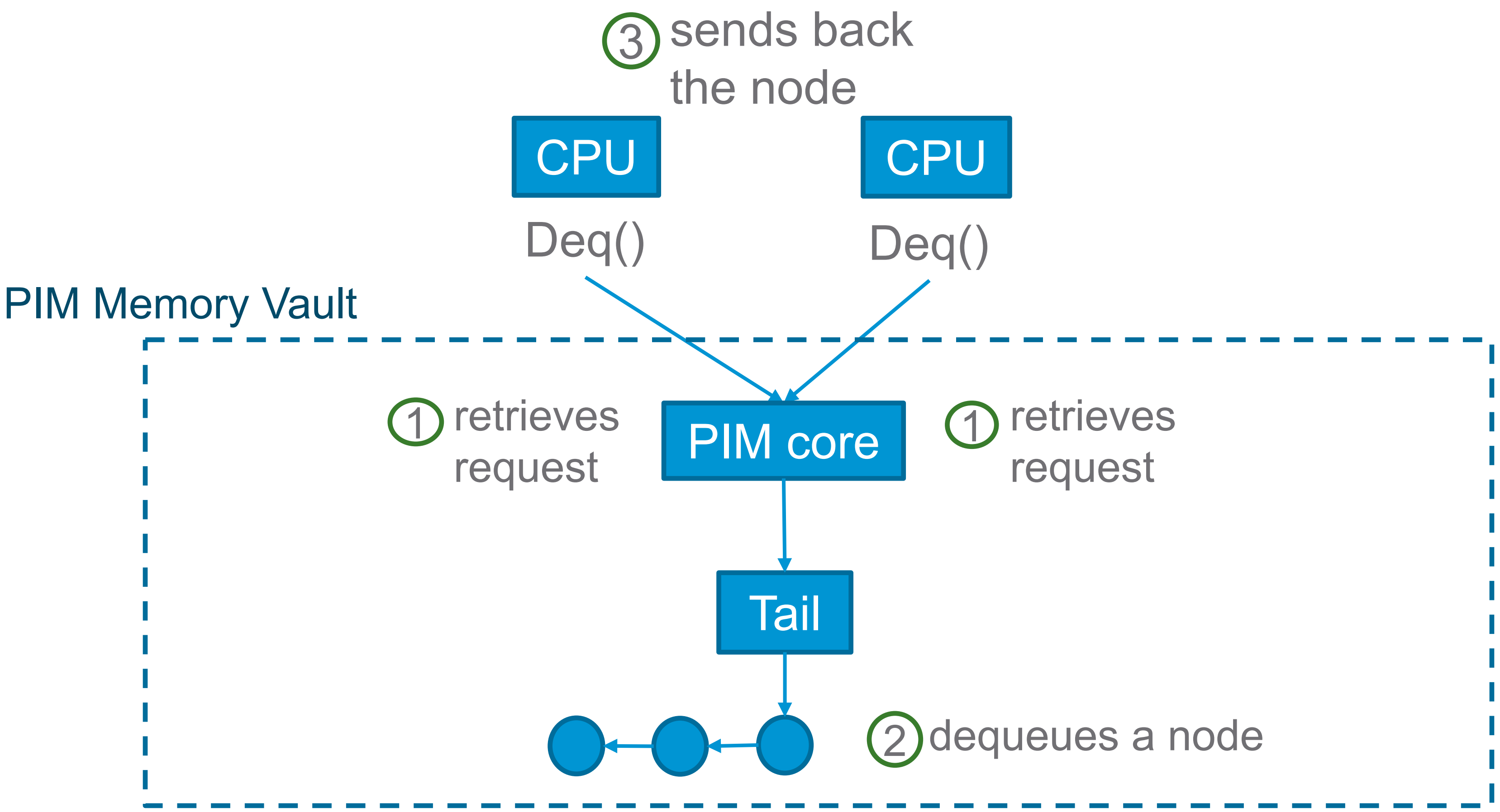
LOW **CONTENTION** HIGH

Pointer-chasing — Cacheable e.g., FIFO queue

FIFO Queue

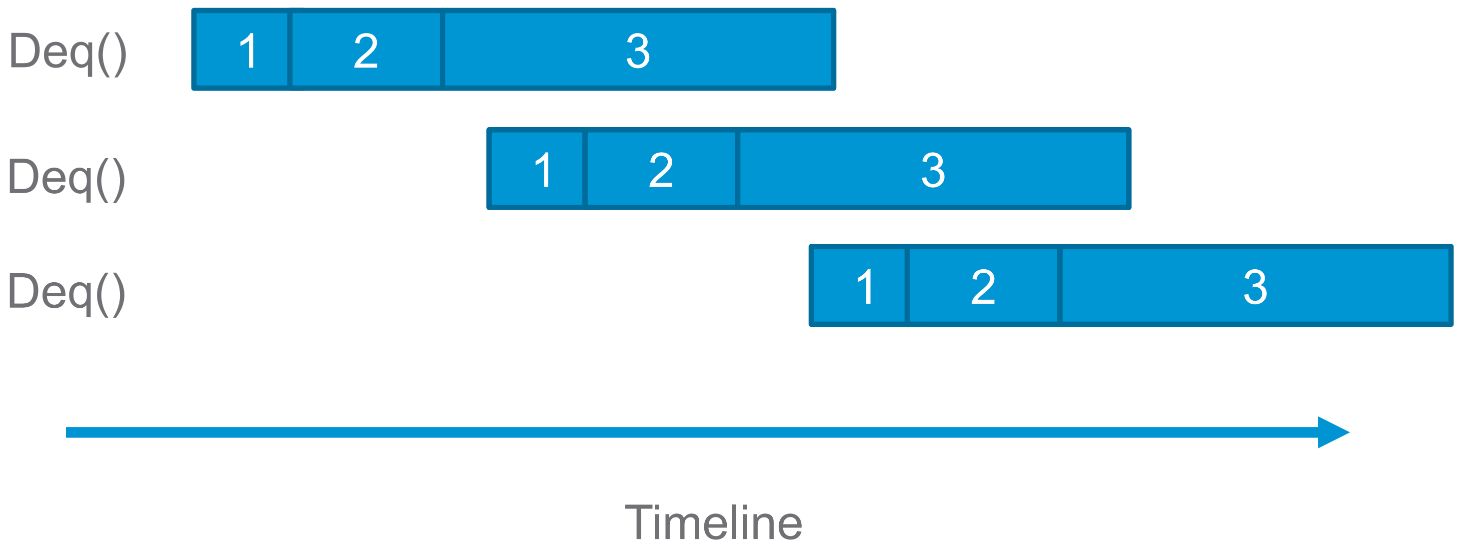


PIM FIFO Queue

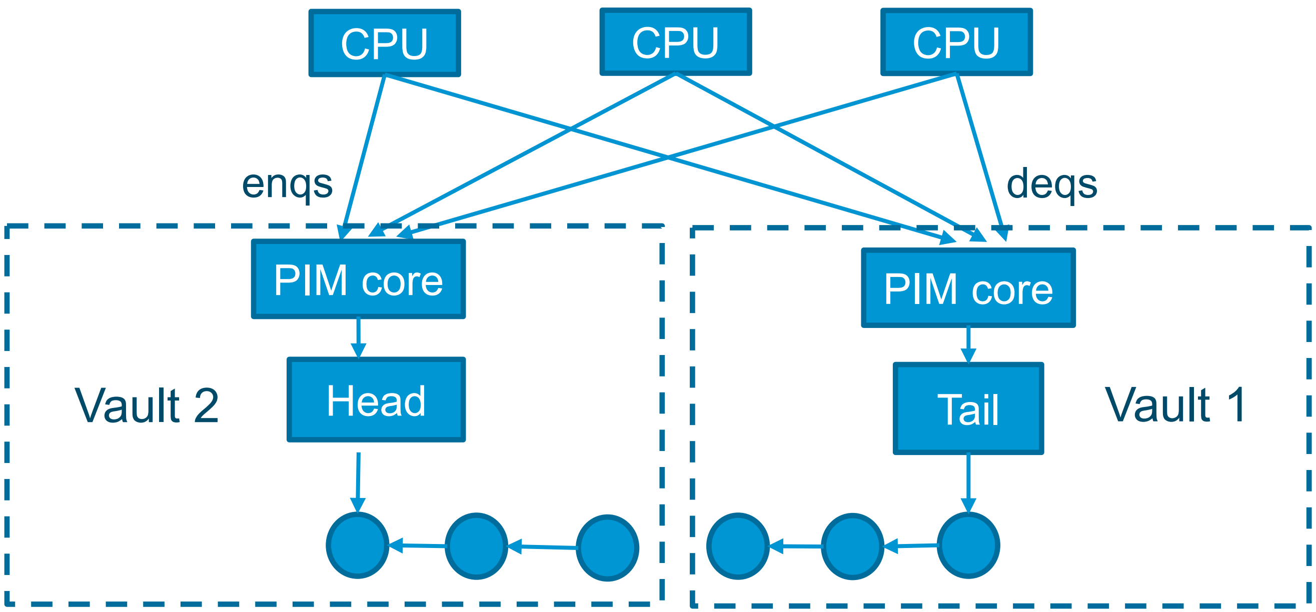


Pipelining

Can overlap the execution of the next request



Parallelize Enqs and Deqs



Conclusion

PIM is becoming feasible in the near future

We investigate Concurrent Data Structures (CDS) for PIM

Results:

- Naïve PIM data structures are less efficient than CDS
- New PIM algorithms can leverage PIM features
  - They outperform efficient CDS
  - They are simpler to design and implement

Thank you!

icalciu@vmware.com

https://research.vmware.com/

Concurrent Data Structures for Near-Memory Computing

Zhiyu Liu (Brown)

Irina Calciu (VMware Research)

Maurice Herlihy (Brown)

Onur Mutlu (ETH)

© 2017 VMware Inc. All rights reserved.