A Scalable Processing-in-Memory Accelerator for Parallel Graph Processing

Junwhan Ahn, Sungpack Hong[*], Sungjoo Yoo, Onur Mutlu[+], Kiyoung Choi

Seoul National University [*]Oracle Labs [+]Carnegie Mellon University

Graphs

- Abstract representation of object relationships
  - Vertex: object (e.g., person, article, …)
  - Edge: relationship (e.g., friendships, hyperlinks, …)

- Recent trend: explosive increase in graph size

36 Million Wikipedia Pages	1.4 Billion Facebook Users	300 Million Twitter Users	30 Billion Instagram Photos

Large-Scale Graph Processing

- Example: Google's PageRank

$$R[i] = \alpha + \boxed{\sum_{j \in \mathrm{Succ}(i)}} \boxed{w_{ji} R[j]}$$

```
for (v: graph.vertices) {
  for (w: v.successors) {
    w.next_rank += weight * v.rank;
  }
}
for (v: graph.vertices) {
  v.rank = v.next_rank; v.next_rank = alpha;
}
```

Large-Scale Graph Processing

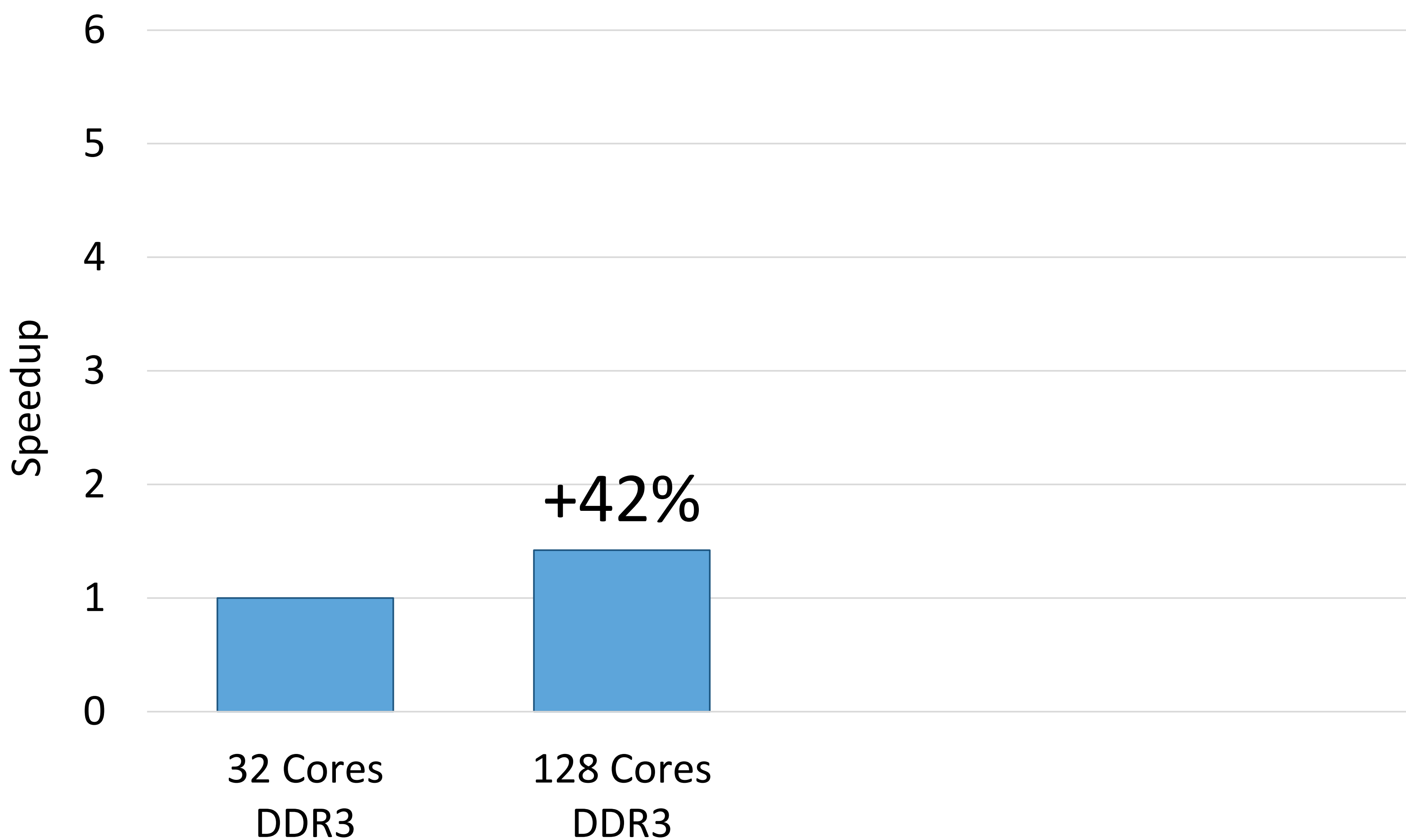
- Example: Google's PageRank

Independent to Each Vertex

Vertex-Parallel Abstraction

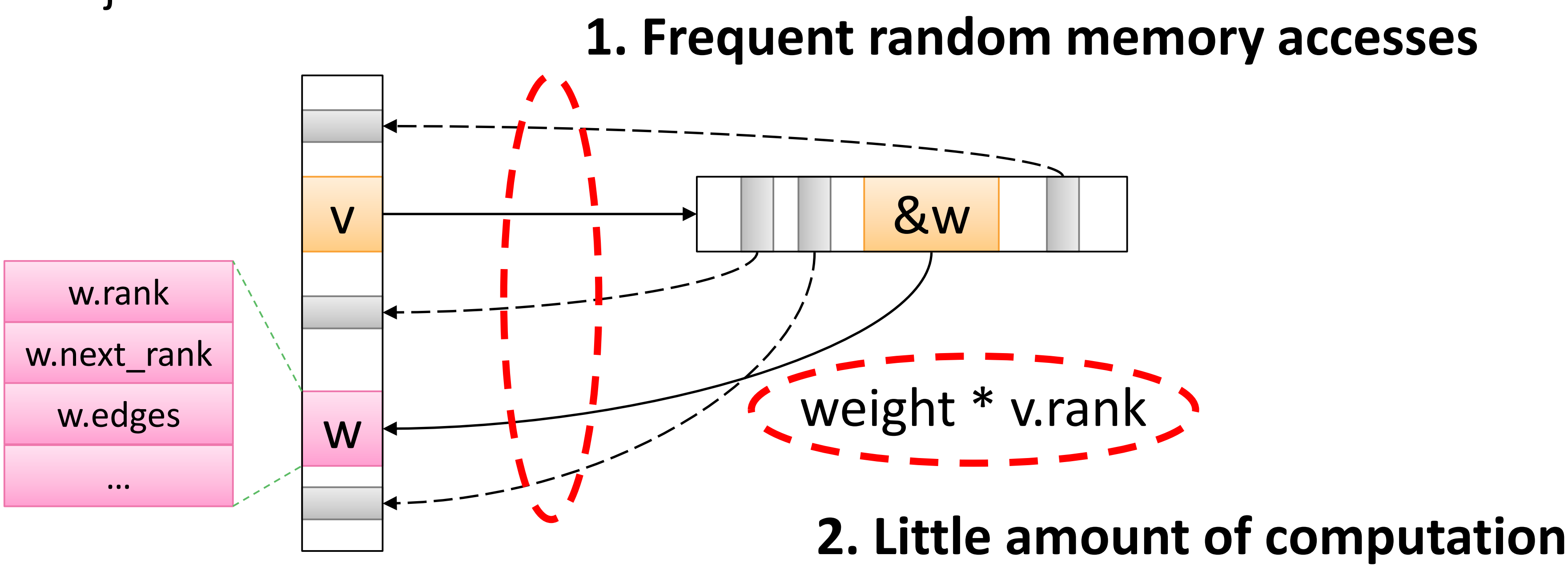
```
for (v: graph.vertices) {
    for (w: v.successors) {
        w.next_rank += weight * v.rank;
    }
}
for (v: graph.vertices) {
    v.rank = v.next_rank; v.next_rank = alpha;
}
```

PageRank Performance



Bottleneck of Graph Processing

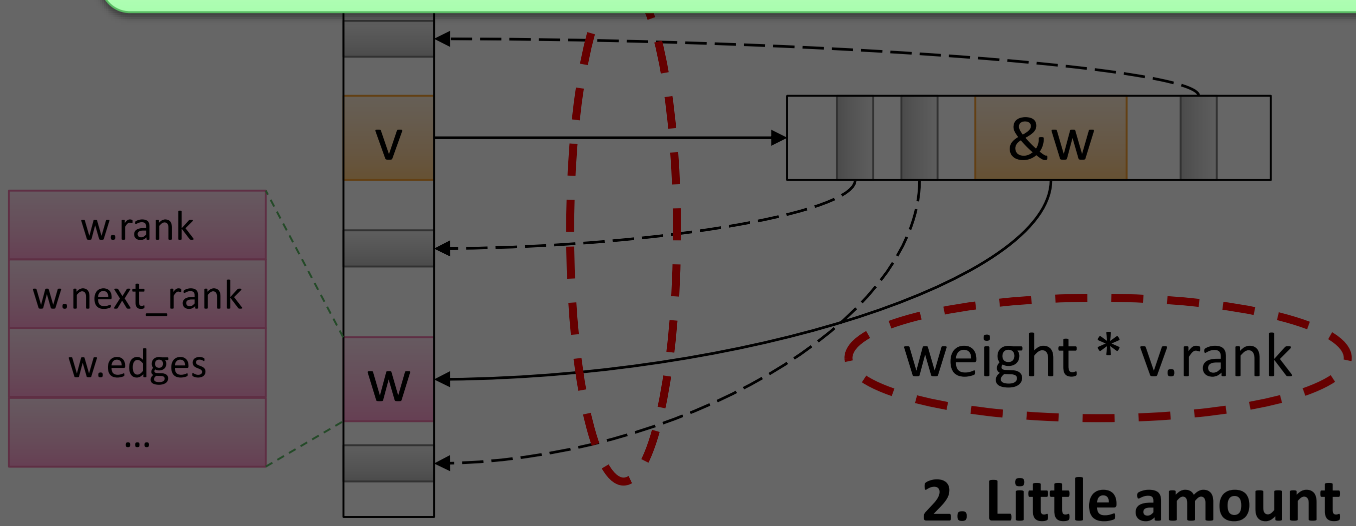
```
for (v: graph.vertices) {
  for (w: v.successors) {
    w.next_rank += weight * v.rank;
  }
}
```



Bottleneck of Graph Processing

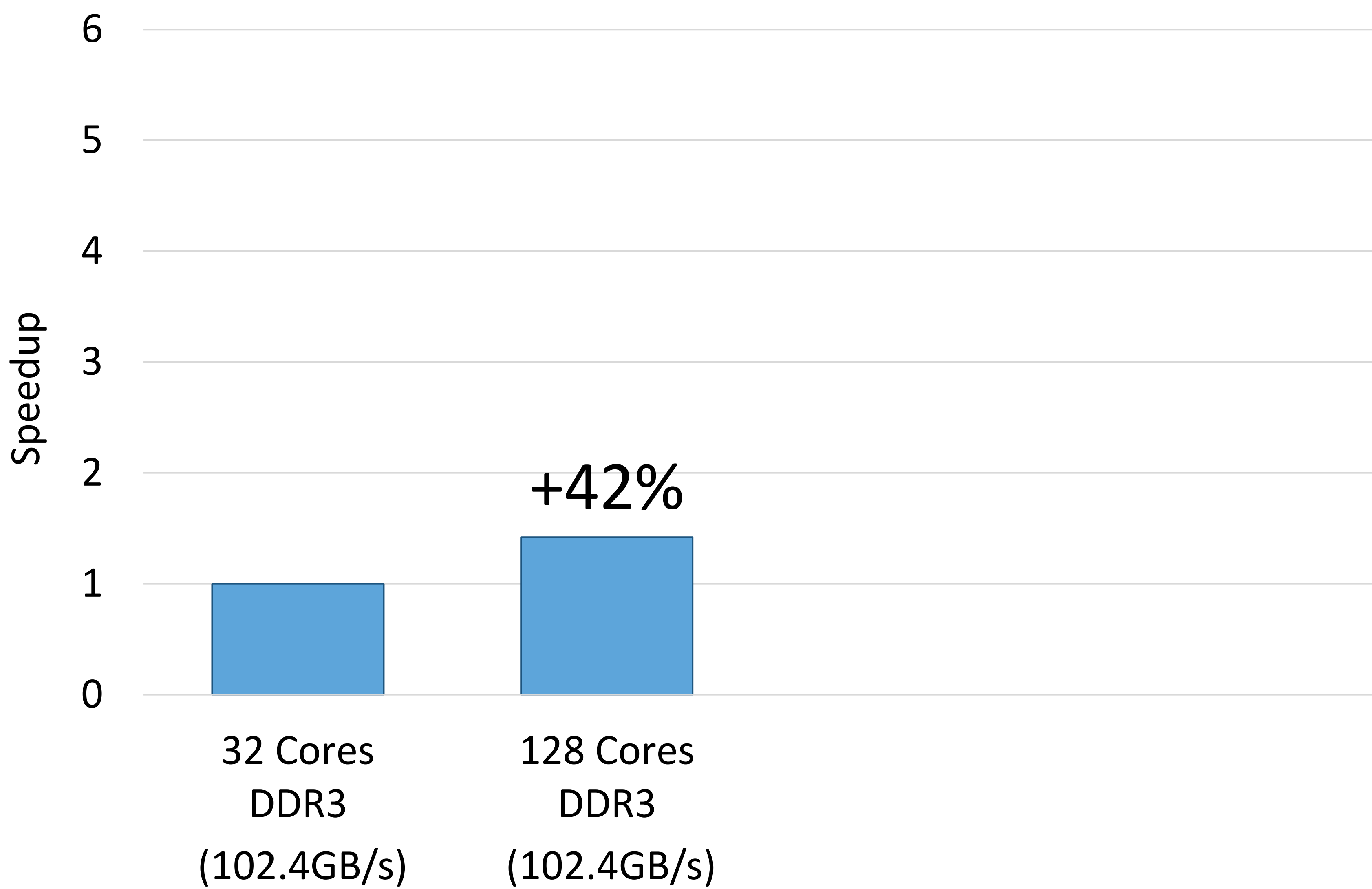
```
for (v: graph.vertices) {
  for (w: v.successors) {
    w.next_rank += weight * v.rank;
  }
```

High Memory Bandwidth Demand

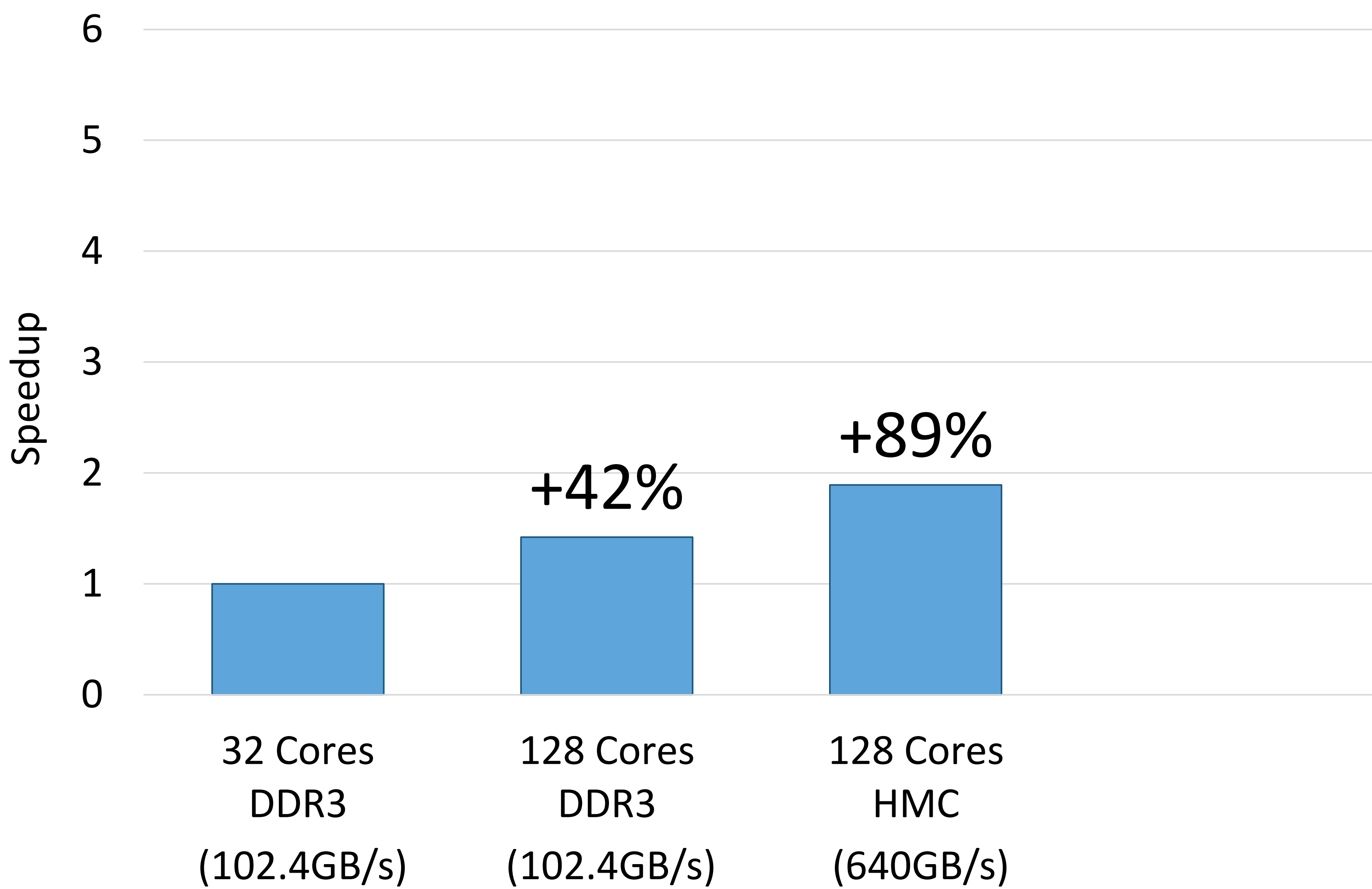


2. Little amount of computation

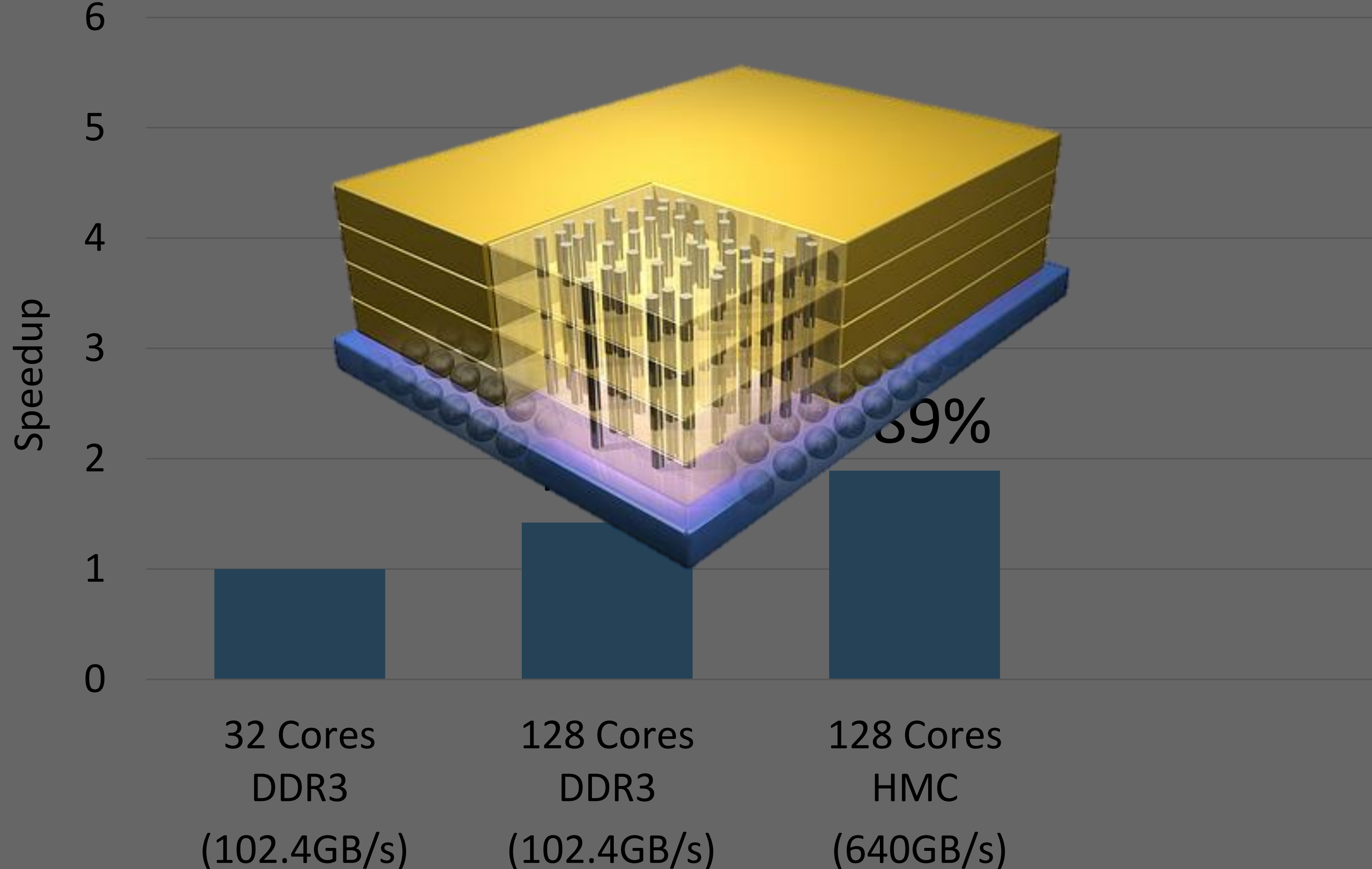
PageRank Performance



PageRank Performance



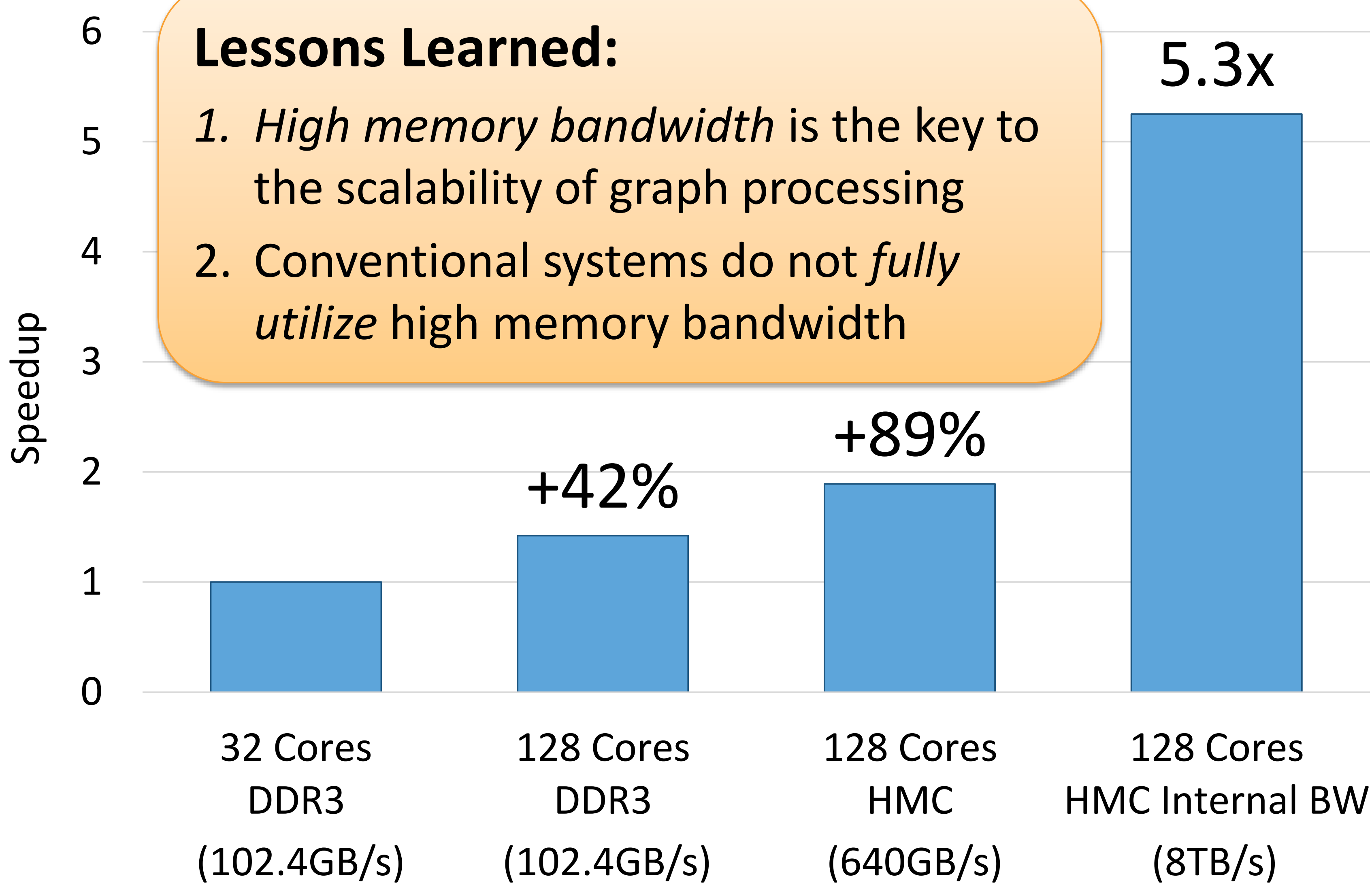
PageRank Performance



PageRank Performance

Lessons Learned:

1. *High memory bandwidth* is the key to the scalability of graph processing
2. Conventional systems do not *fully utilize* high memory bandwidth

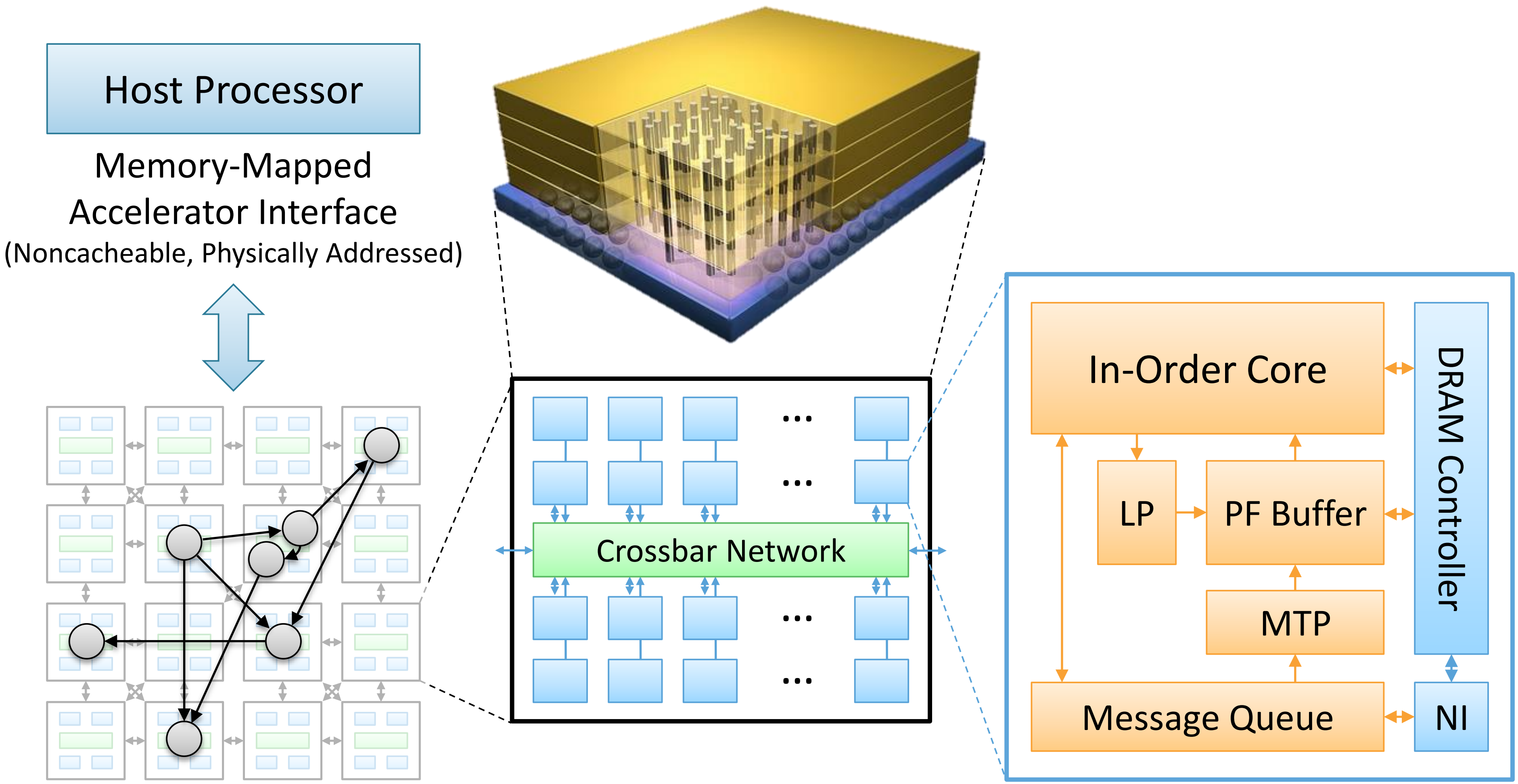


Challenges in Scalable Graph Processing

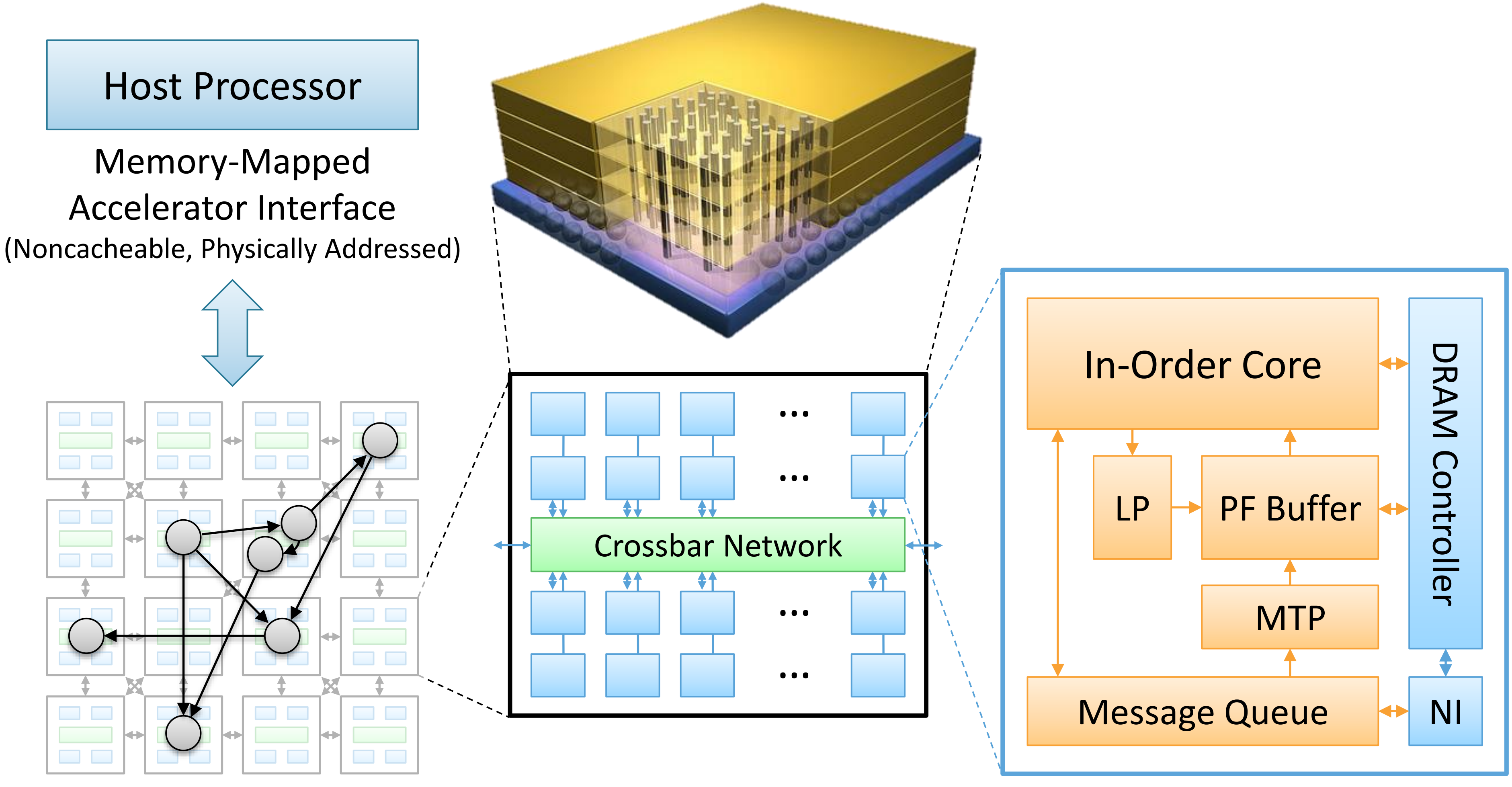
- **Challenge 1**: How to provide *high memory bandwidth* to computation units in a practical way?
  - Processing-in-memory based on 3D-stacked DRAM

- **Challenge 2**: How to design computation units that *efficiently exploit large memory bandwidth*?
  - Specialized in-order cores called *Tesseract* cores
    - Latency-tolerant programming model
    - Graph-processing-specific prefetching schemes

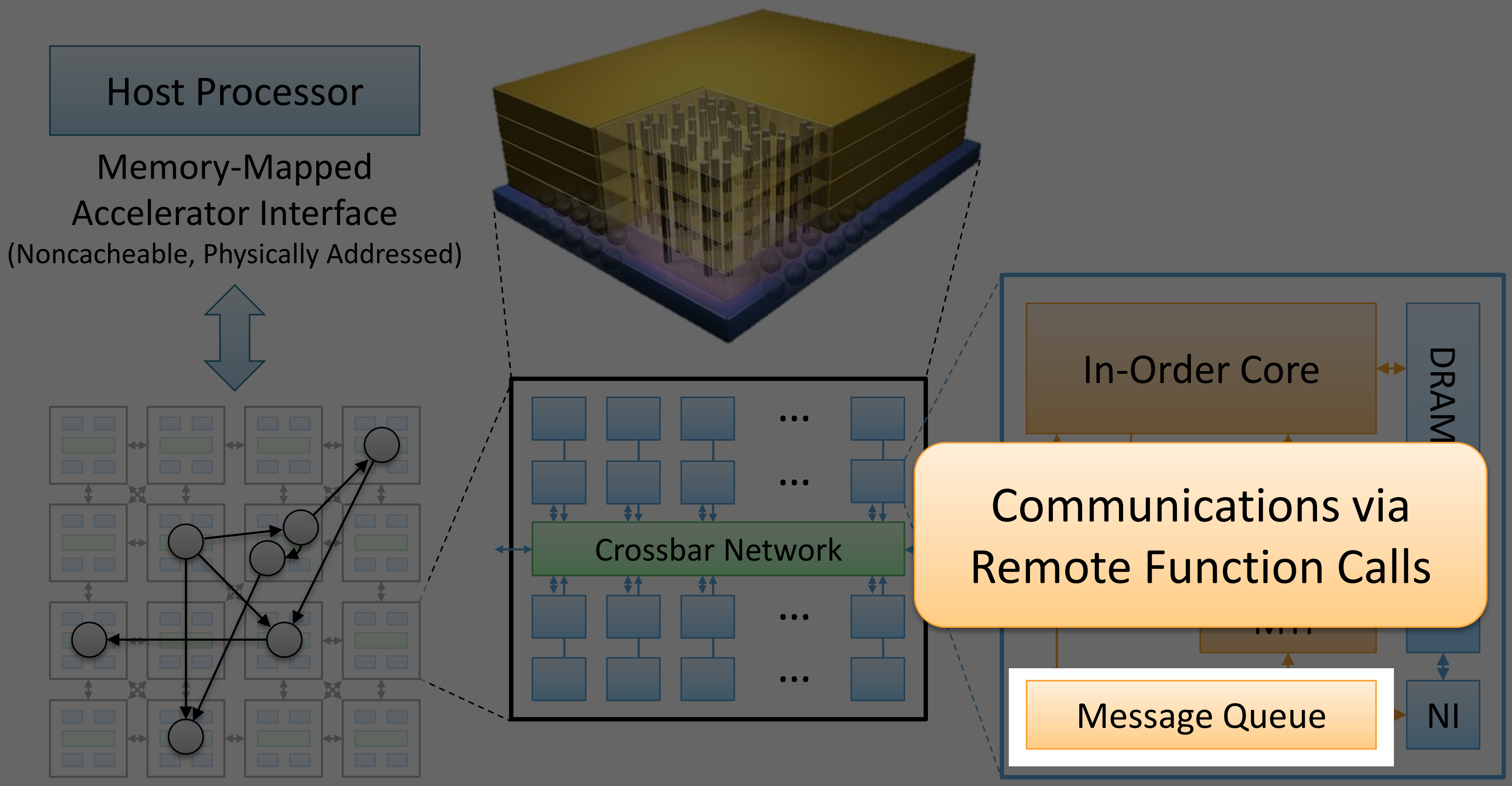
Tesseract System



Tesseract System

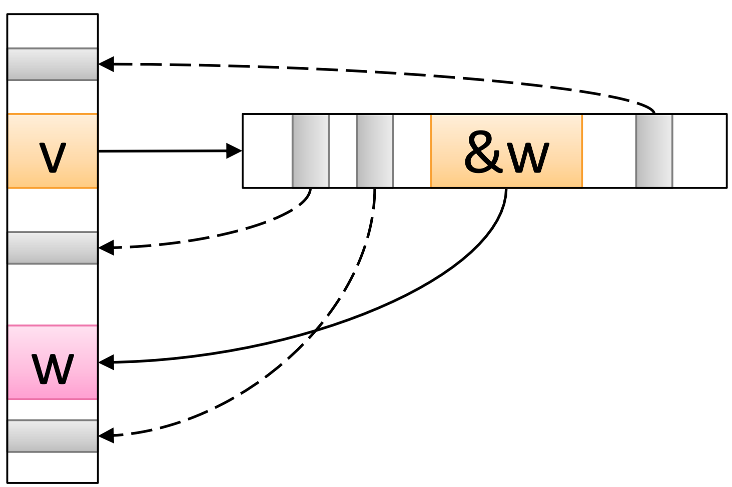


Tesseract System



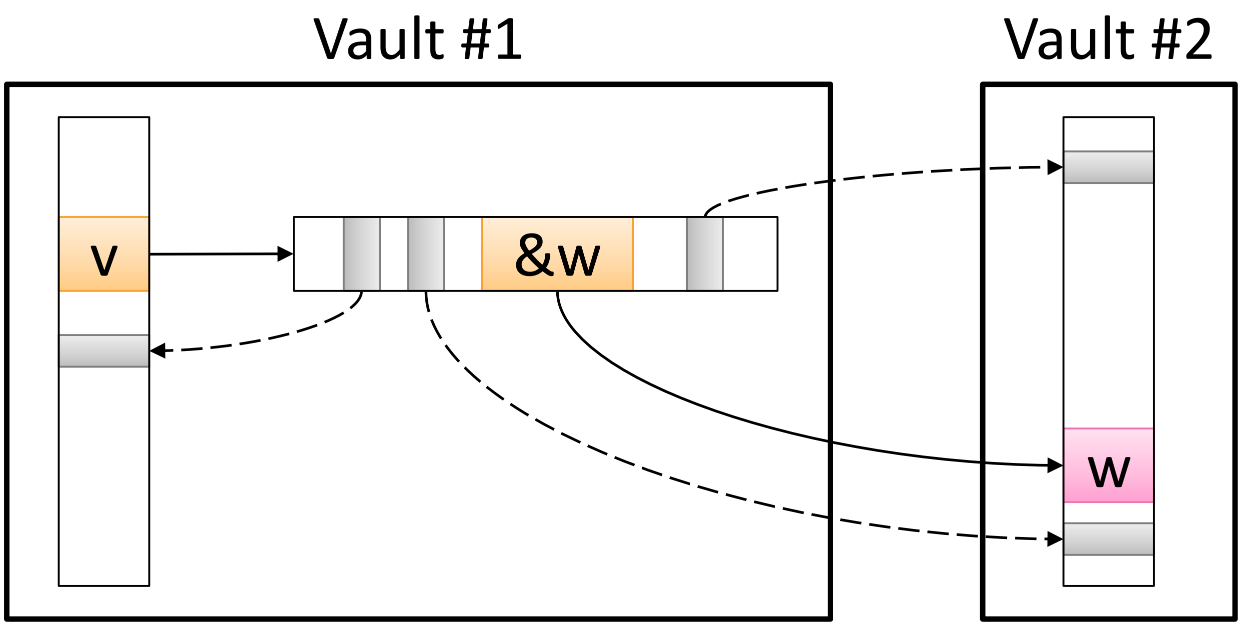
Communications in Tesseract

```
for (v: graph.vertices) {
  for (w: v.successors) {
    w.next_rank += weight * v.rank;
  }
}
```



Communications in Tesseract

```
for (v: graph.vertices) {
  for (w: v.successors) {
    w.next_rank += weight * v.rank;
  }
}
```

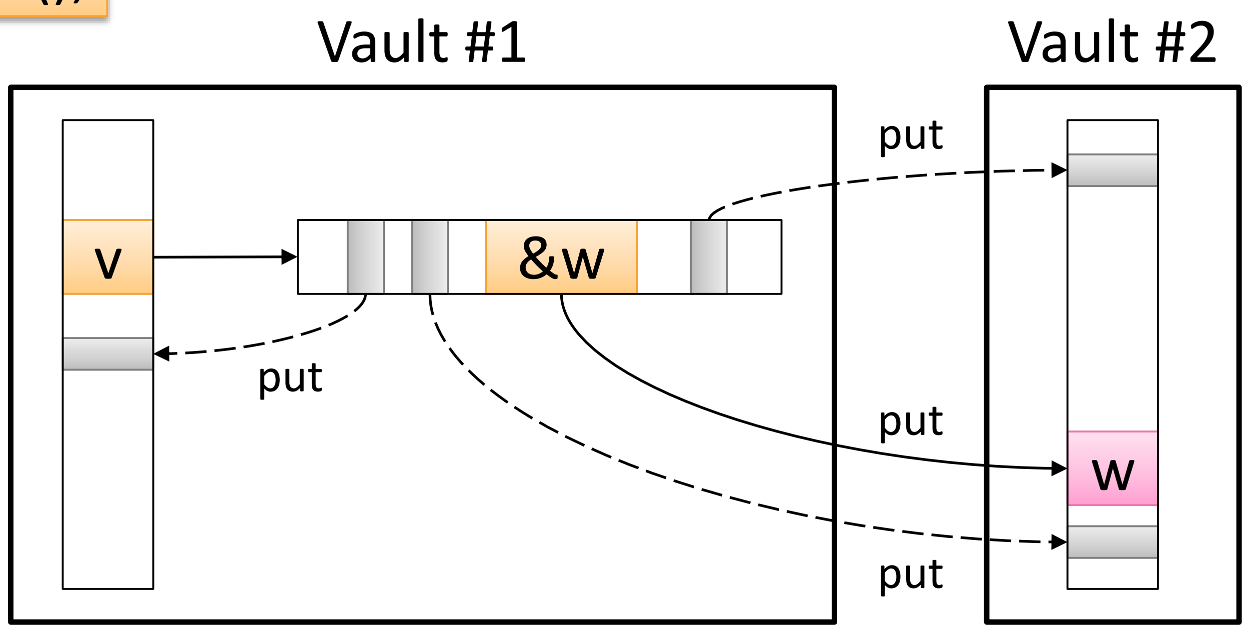


Communications in Tesseract

```
for (v: graph.vertices) {
    for (w: v.successors) {
        put(w.id, function() { w.next_rank += weight * v.rank; });
    }
}
barrier();
```

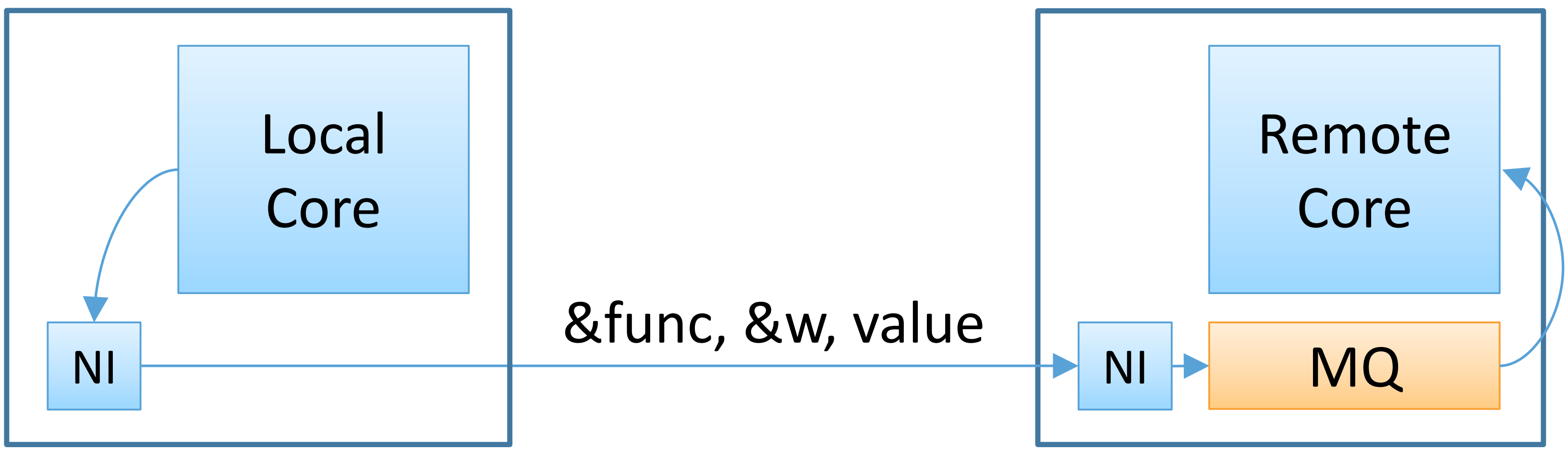
Non-blocking Remote Function Call

Can be **delayed** until the nearest barrier



Non-blocking Remote Function Call

1. Send function address & args to the remote core
2. Store the incoming message to the message queue
3. Flush the message queue when it is full or a synchronization barrier is reached

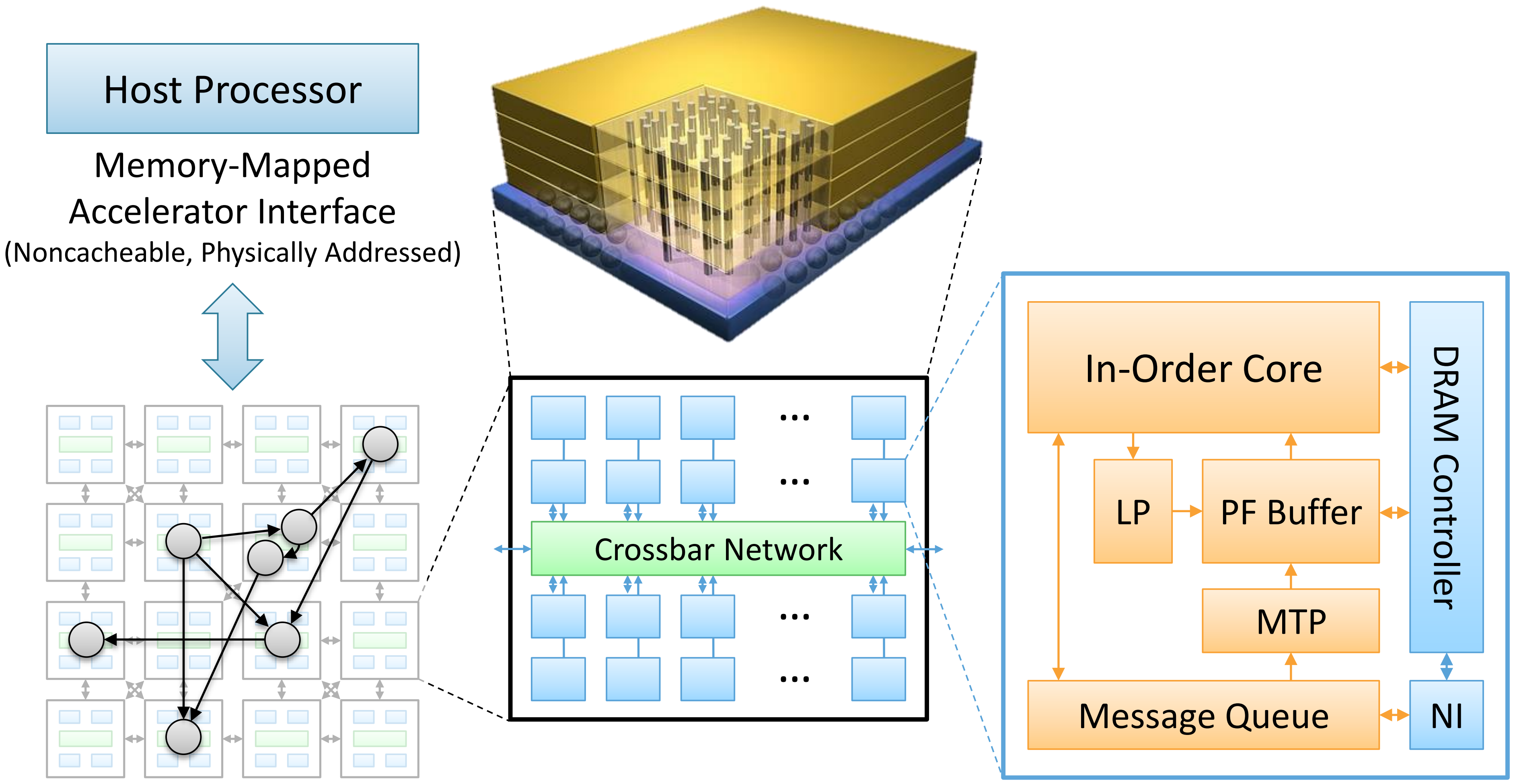


put(w.id, function() { w.next_rank += value; })

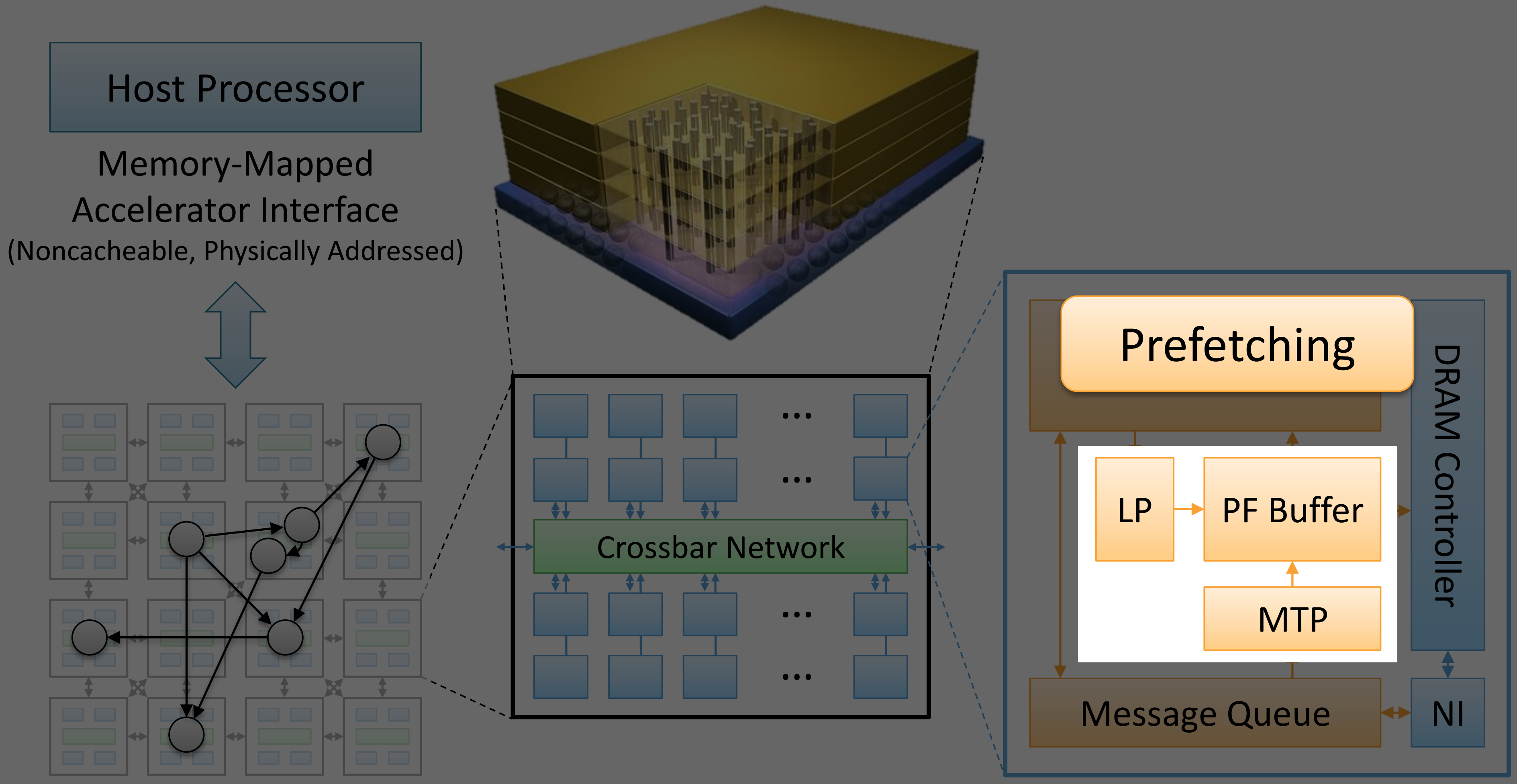
Benefits of Non-blocking Remote Function Call

- Latency hiding through fire-and-forget
  - Local cores are not blocked by remote function calls
- Localized memory traffic
  - No off-chip traffic during remote function call execution
- No need for mutexes
  - Non-blocking remote function calls are atomic
- Prefetching
  - Will be covered shortly

Tesseract System

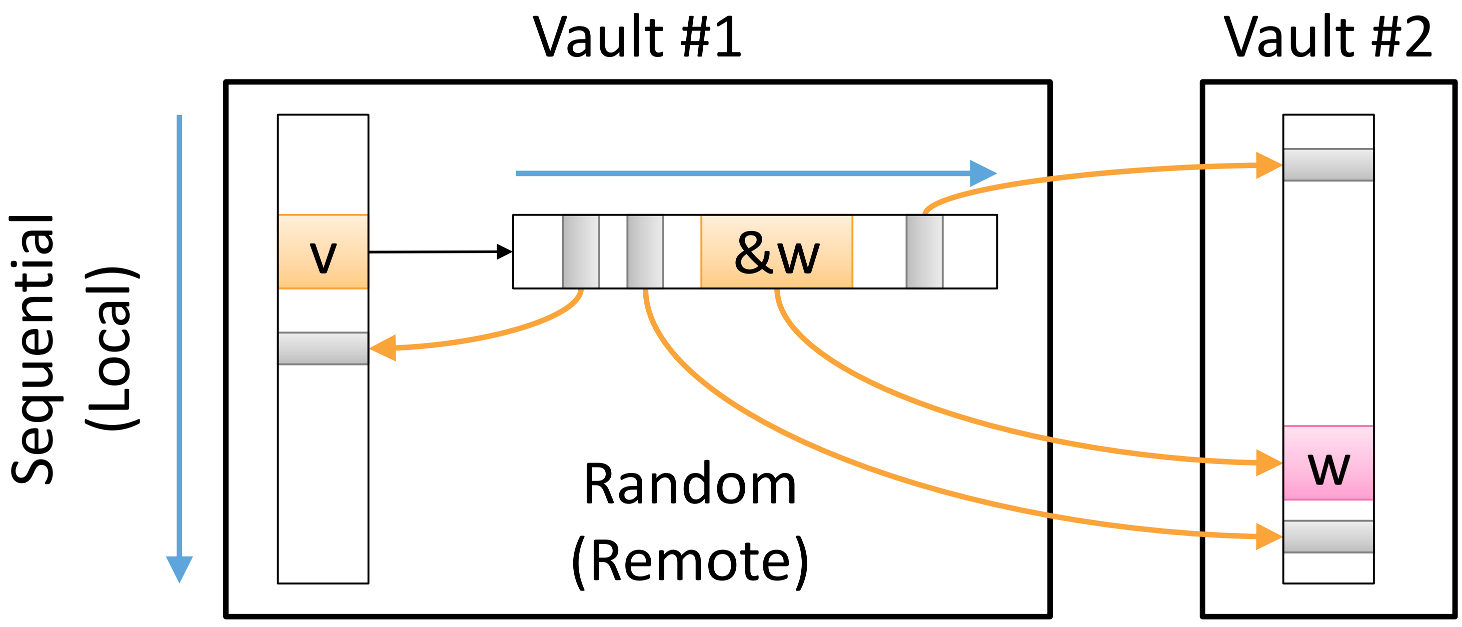


Tesseract System



Memory Access Patterns in Graph Processing

```
for (v: graph.vertices) {
  for (w: v.successors) {
    put(w.id, function() { w.next_rank += weight * v.rank; });
  }
}
```



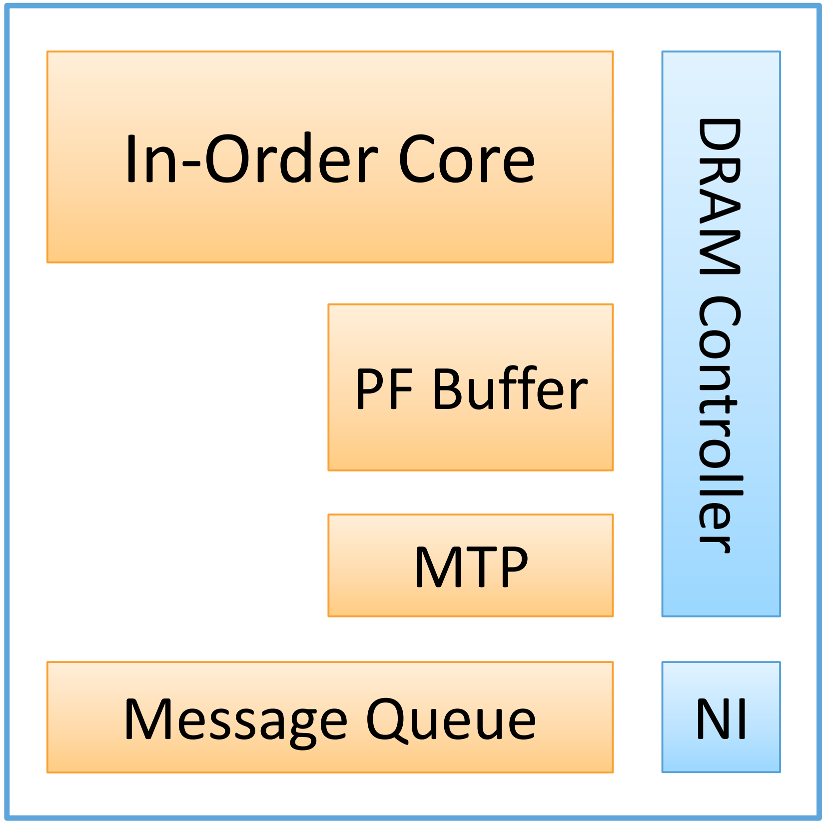
Message-Triggered Prefetching

- Prefetching random memory accesses is difficult
- Opportunities in Tesseract
  - Domain-specific knowledge of target data address
  - Time slack of non-blocking remote function calls

```
for (v: graph.vertices) {
  for (w: v.successors) {
    put(w.id, function() { w.next_rank += weight * v.rank; });
  }
}
barrier();
```

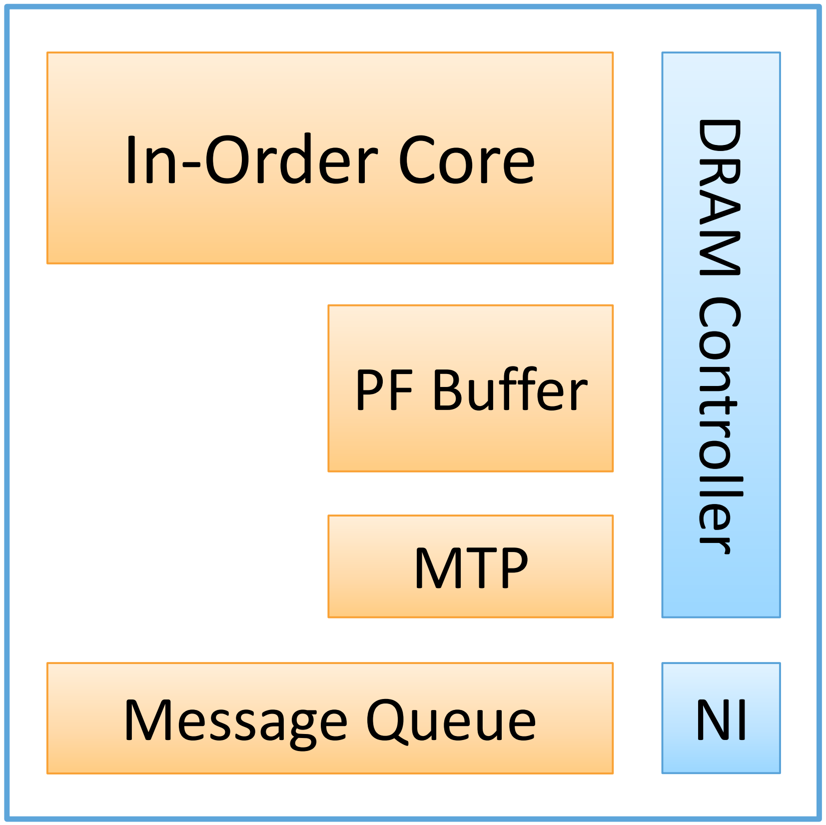
Can be **delayed**
until the nearest barrier

Message-Triggered Prefetching



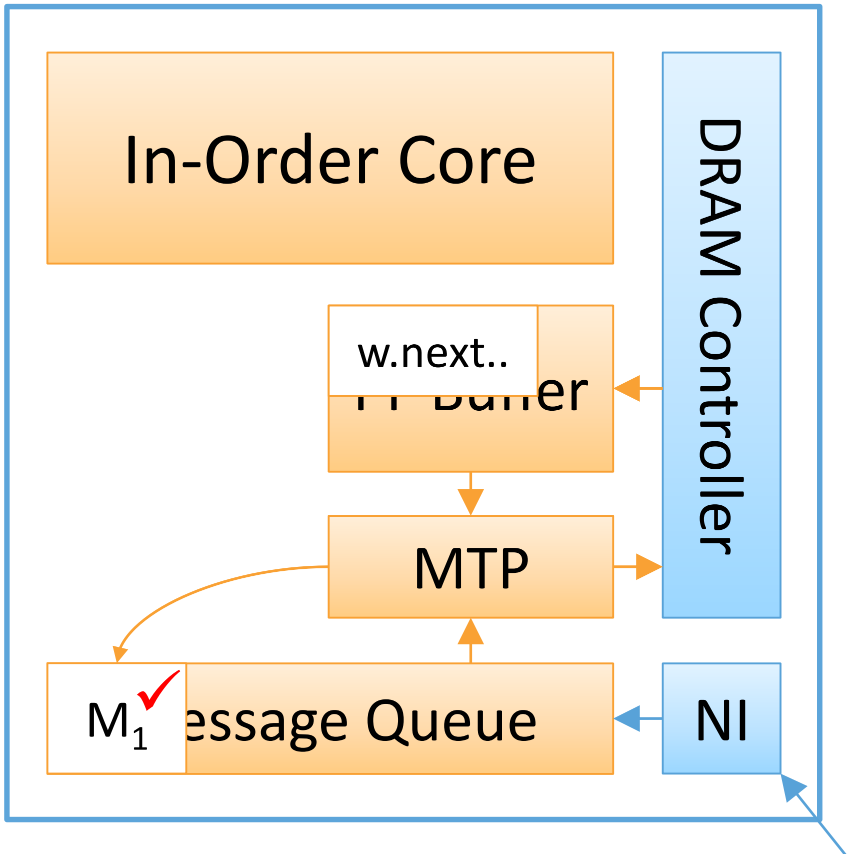
put(w.id, function() { w.next_rank += value; })

Message-Triggered Prefetching



put(w.id, function() { w.next_rank += value; }, &w.next_rank)

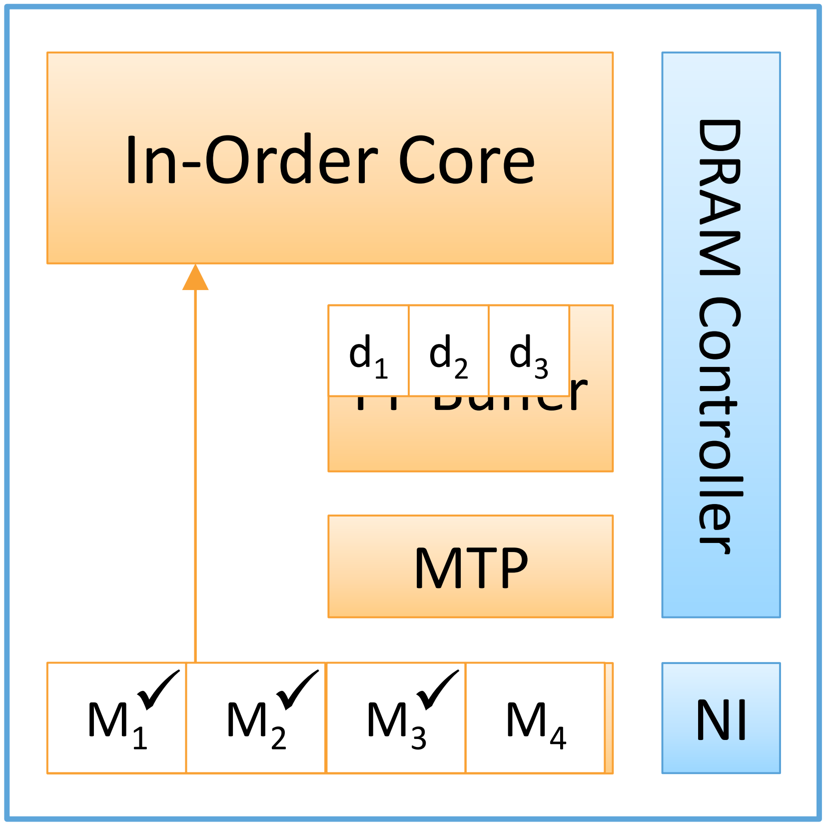
Message-Triggered Prefetching



1. Message M_1 received
2. Request prefetch
3. Mark M_1 as ready when the prefetch is serviced

put(w.id, function() { w.next_rank += value; }, &w.next_rank)

Message-Triggered Prefetching



1. Message M_1 received
2. Request prefetch
3. Mark M_1 as ready when the prefetch is serviced
4. Process multiple **ready** messages at once

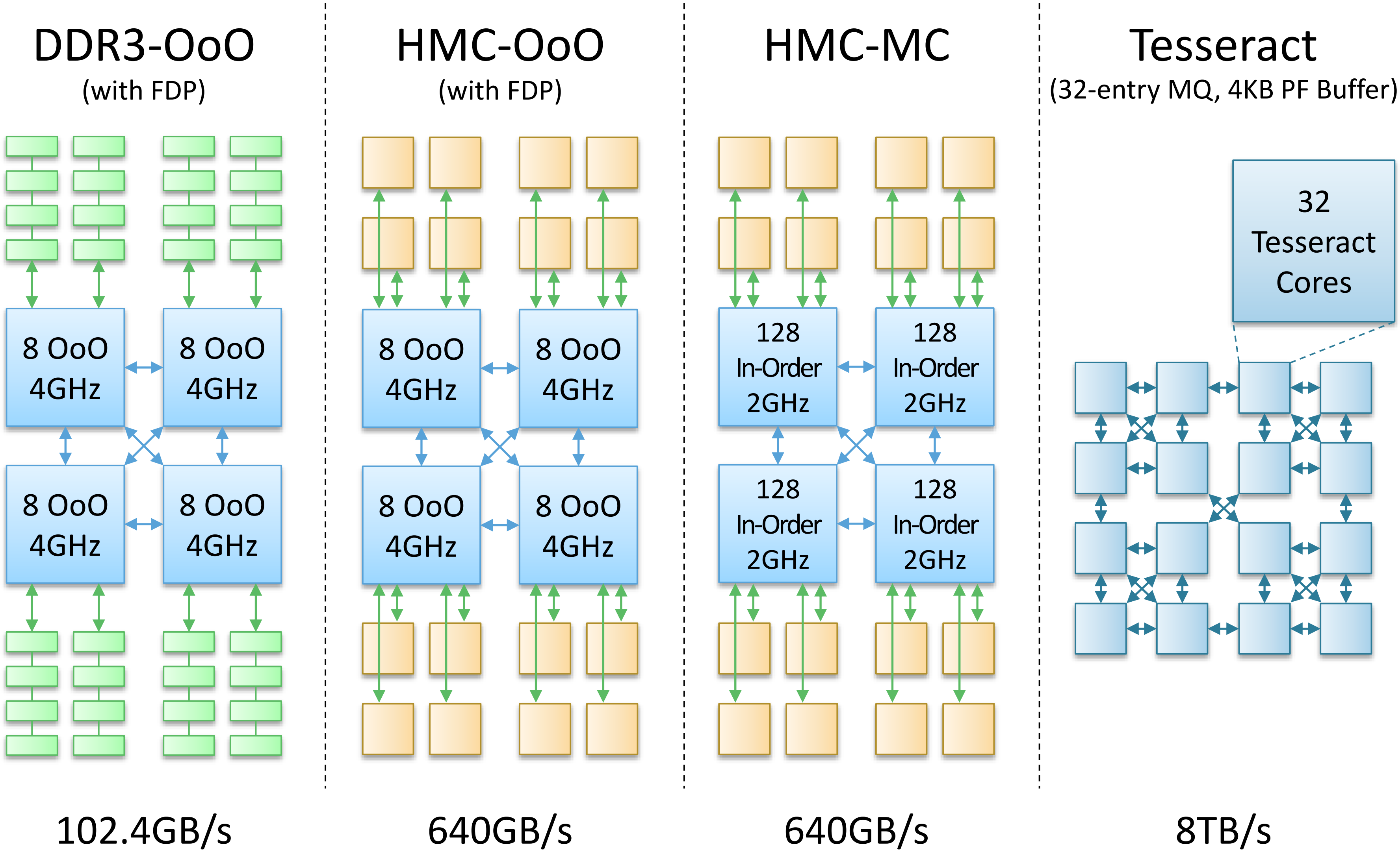
```
put(w.id, function() { w.next_rank += value; }, &w.next_rank)
```

Other Features of Tesseract

- Blocking remote function calls
- List prefetching
- Prefetch buffer
- Programming APIs
- Application mapping

Please see the paper for details

Evaluated Systems

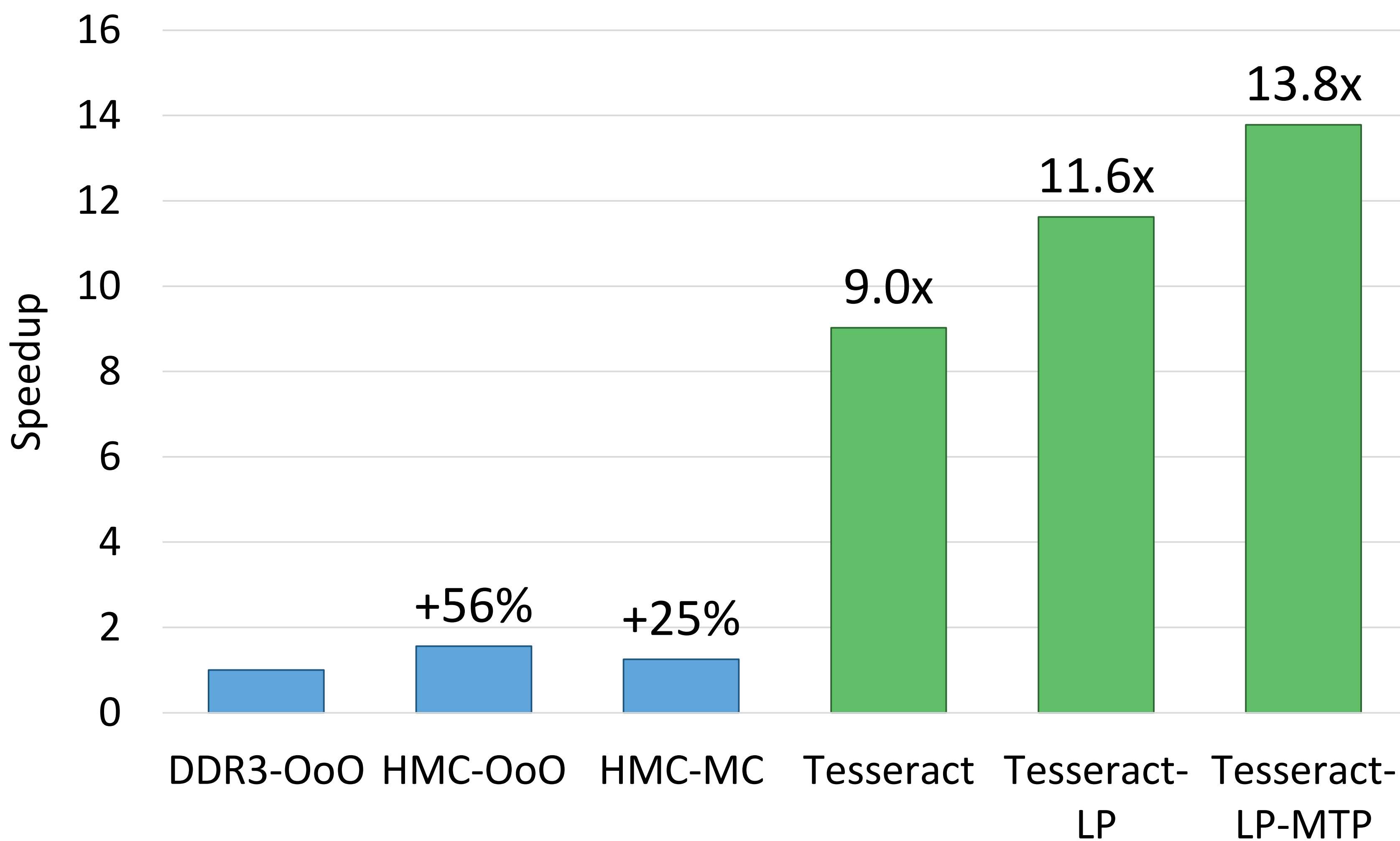


Workloads

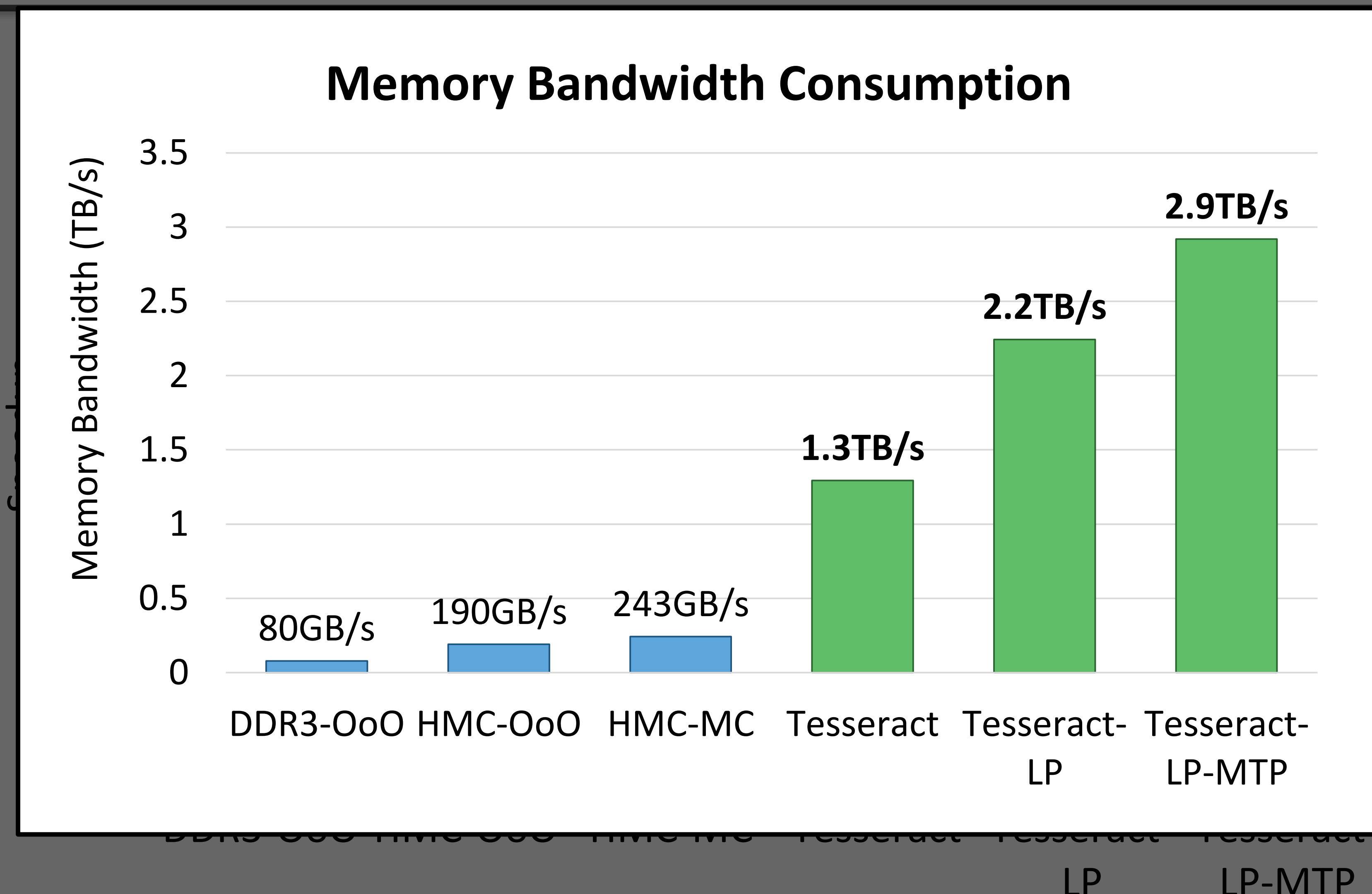
- Five graph processing algorithms
  - Average teenage follower
  - Conductance
  - PageRank
  - Single-source shortest path
  - Vertex cover

- Three real-world large graphs
  - ljournal-2008 (social network)
  - enwiki-2003 (Wikipedia)
  - indochina-0024 (web graph)
  - 4~7M vertices, 79~194M edges

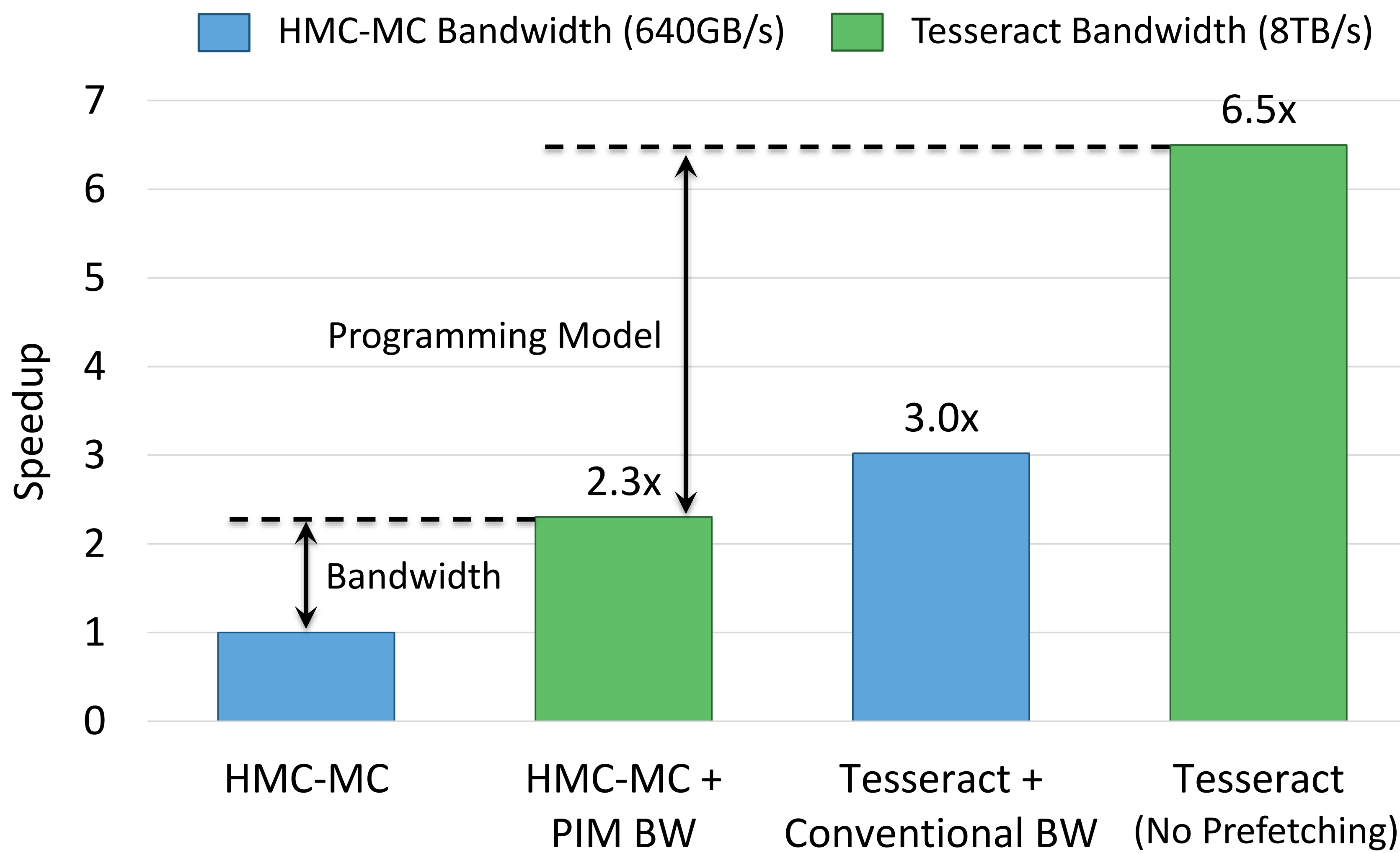
Performance



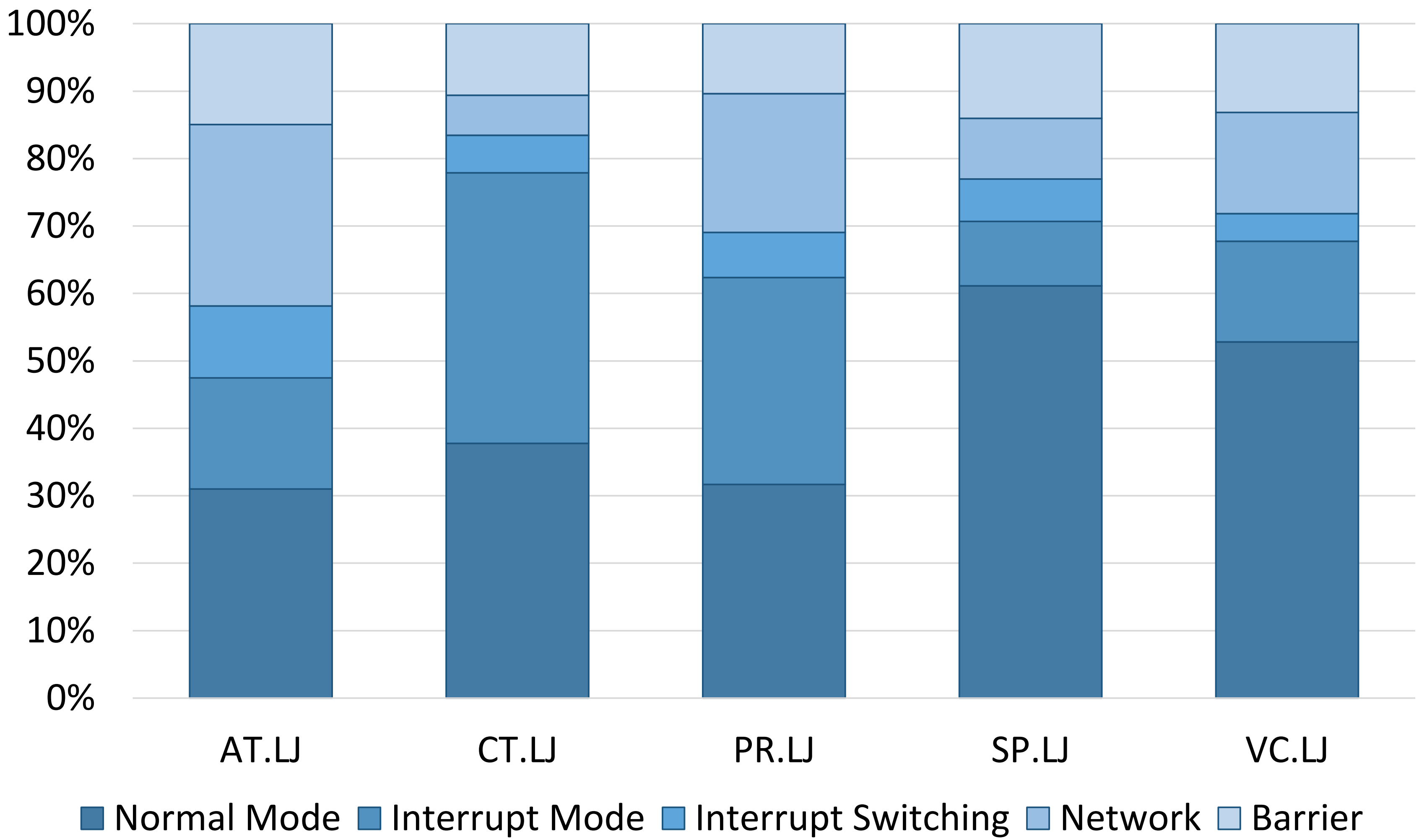
Performance



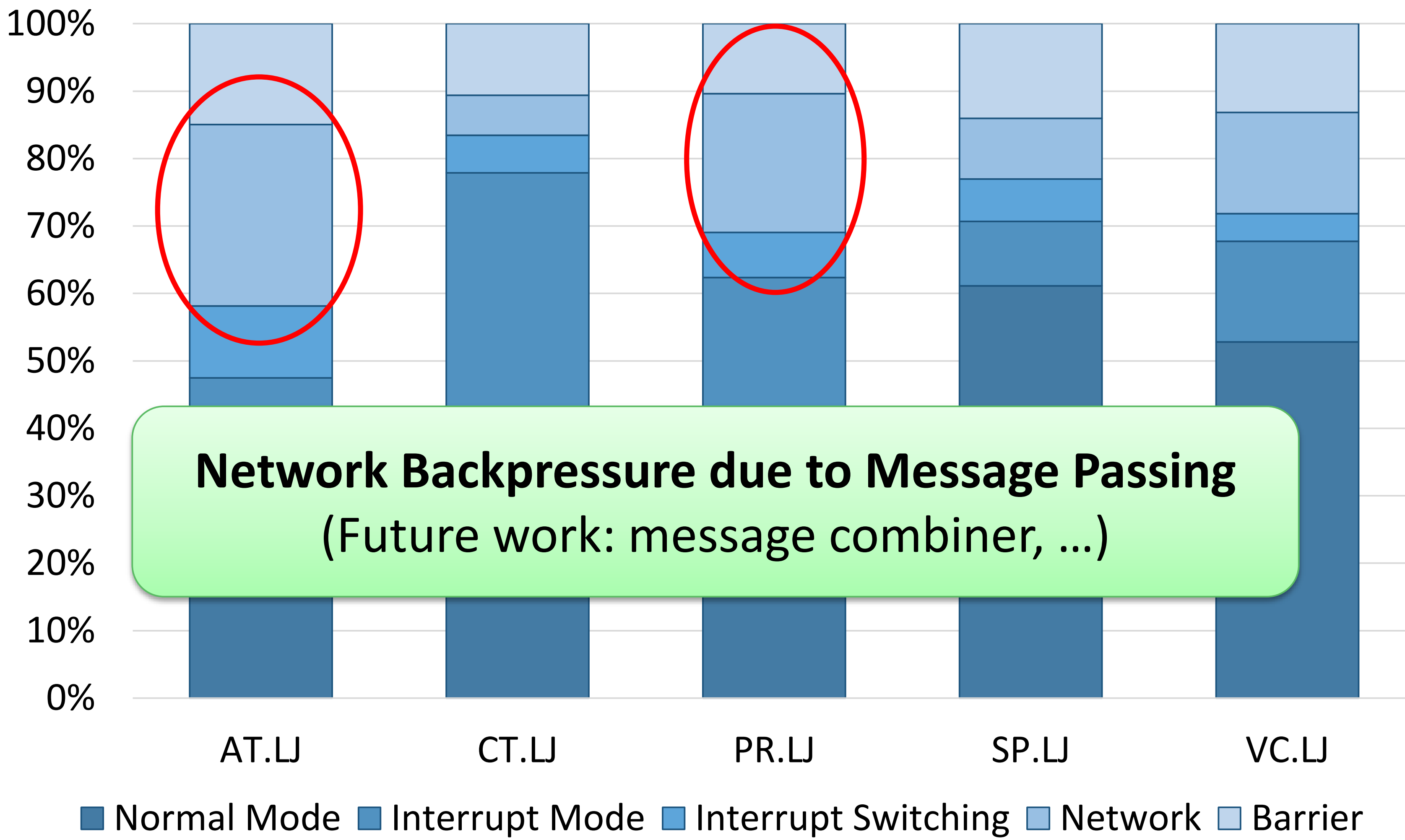
Iso-Bandwidth Comparison



Execution Time Breakdown



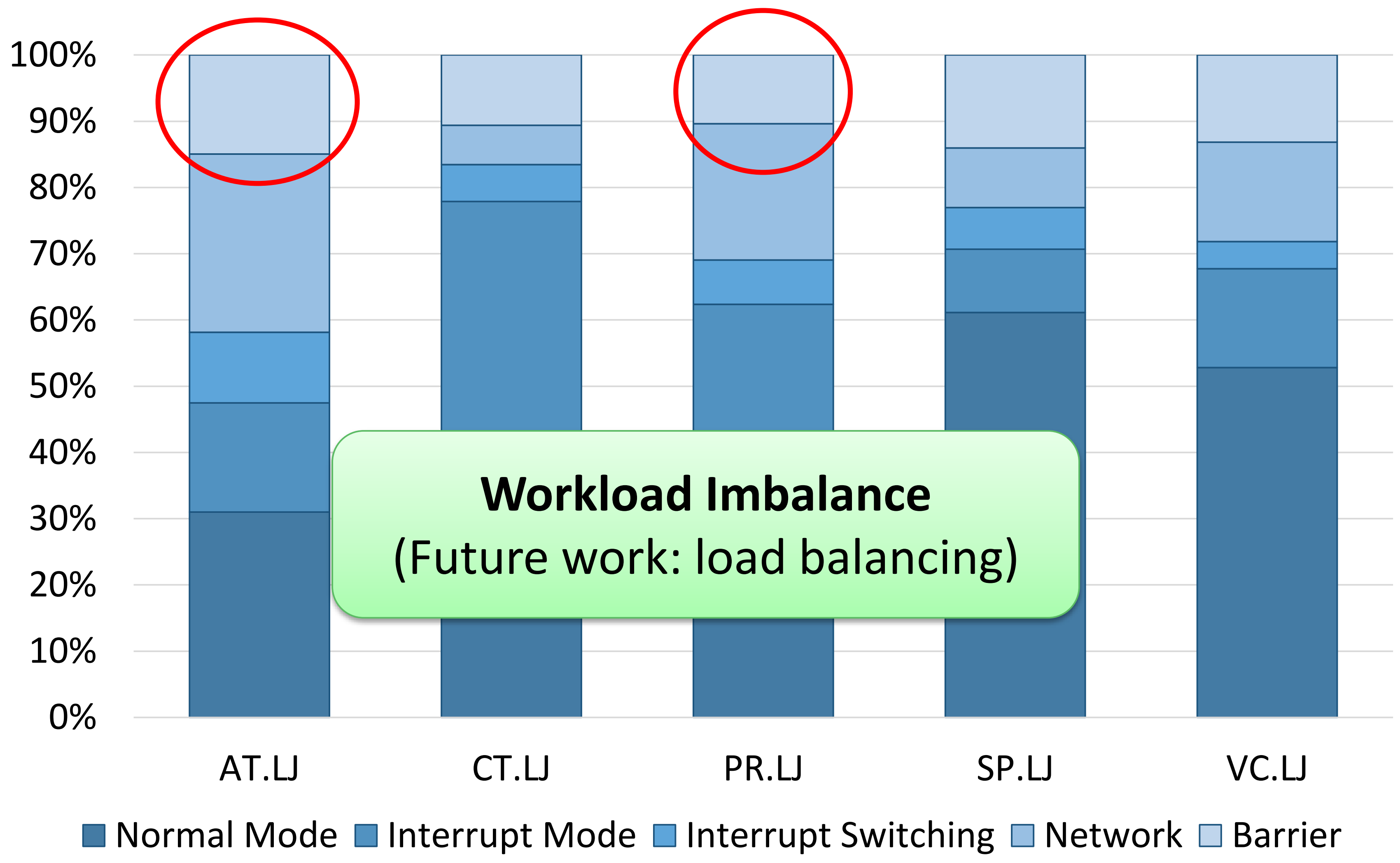
Execution Time Breakdown



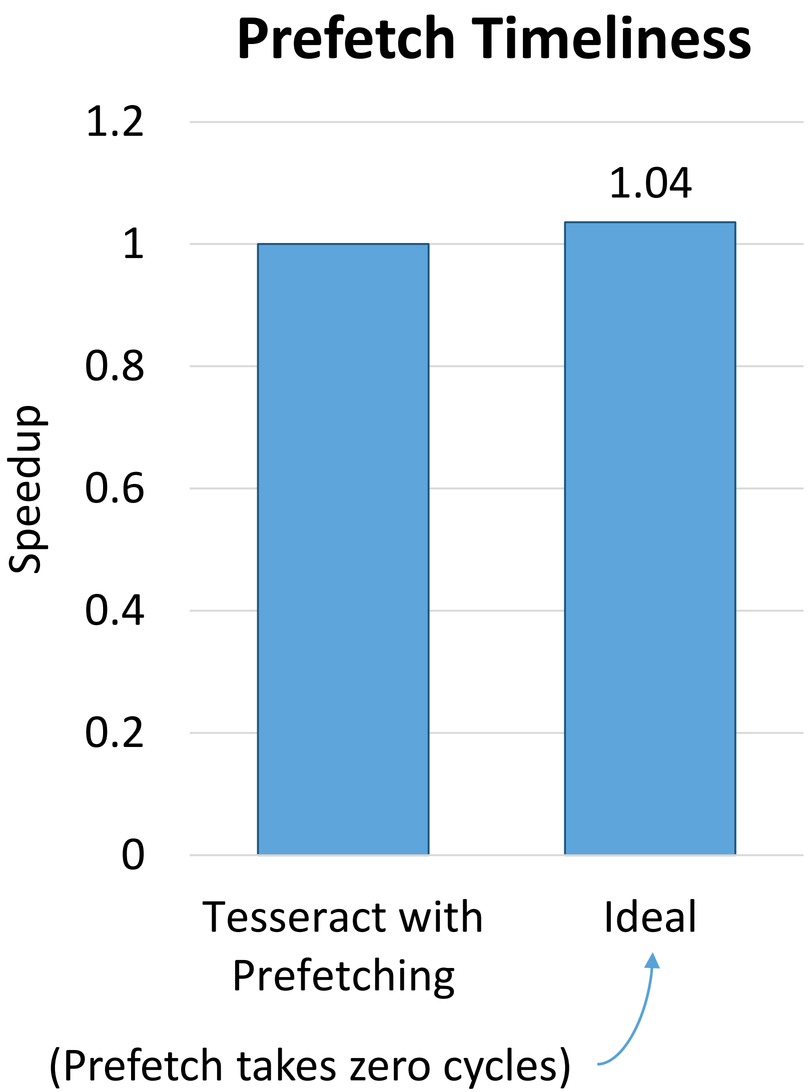
Network Backpressure due to Message Passing

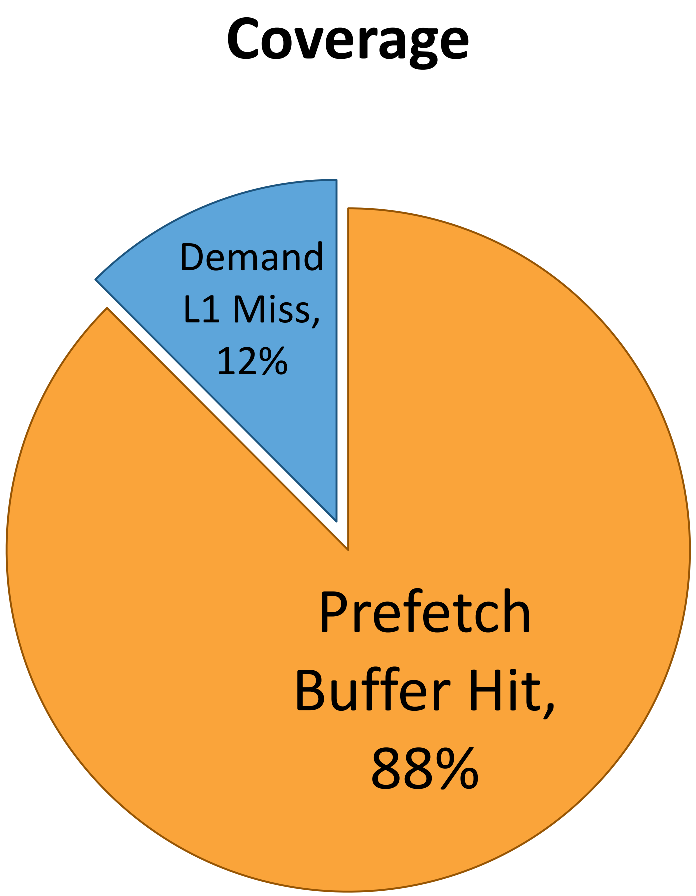
(Future work: message combiner, ...)

Execution Time Breakdown

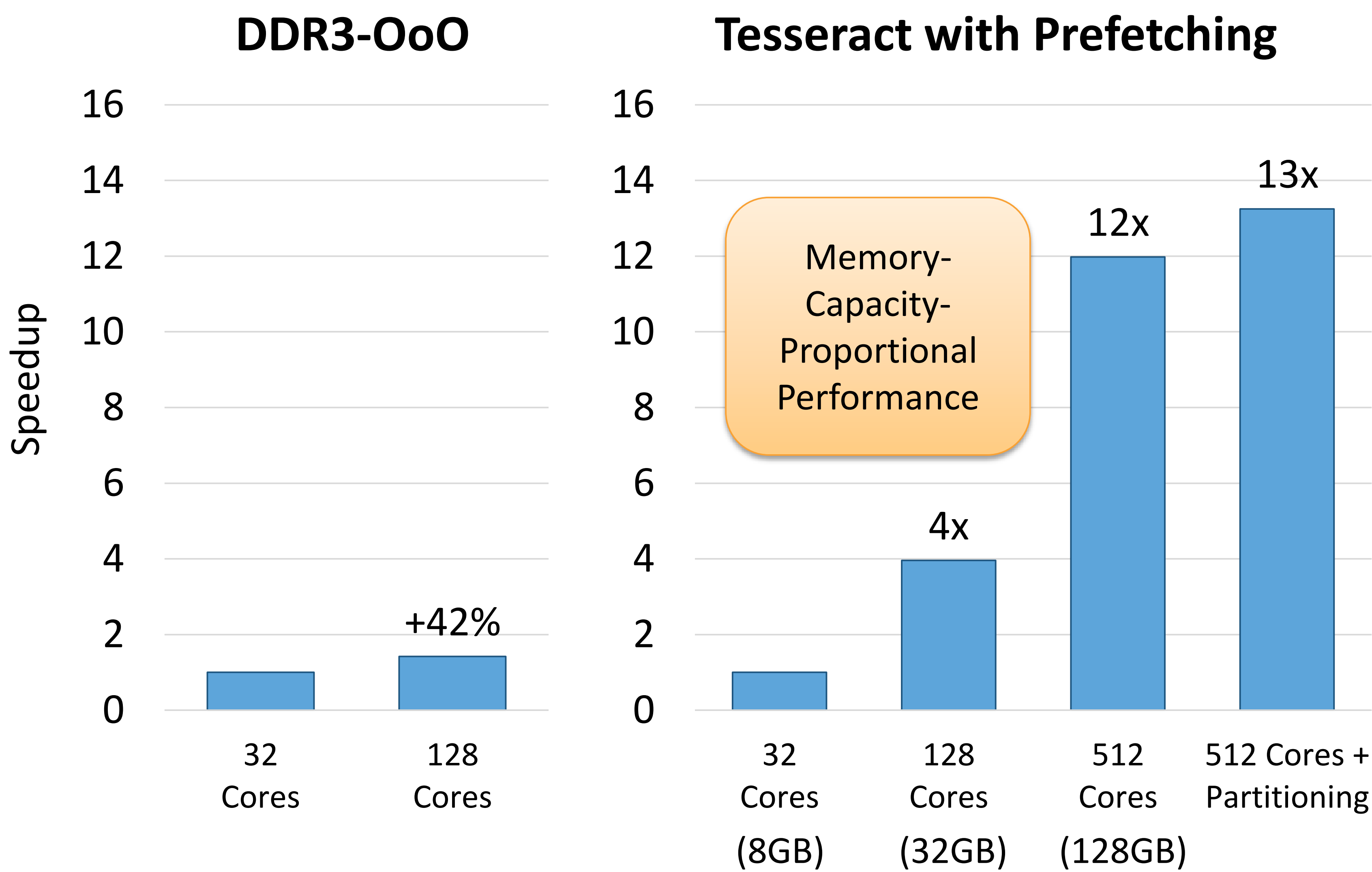


Prefetch Efficiency

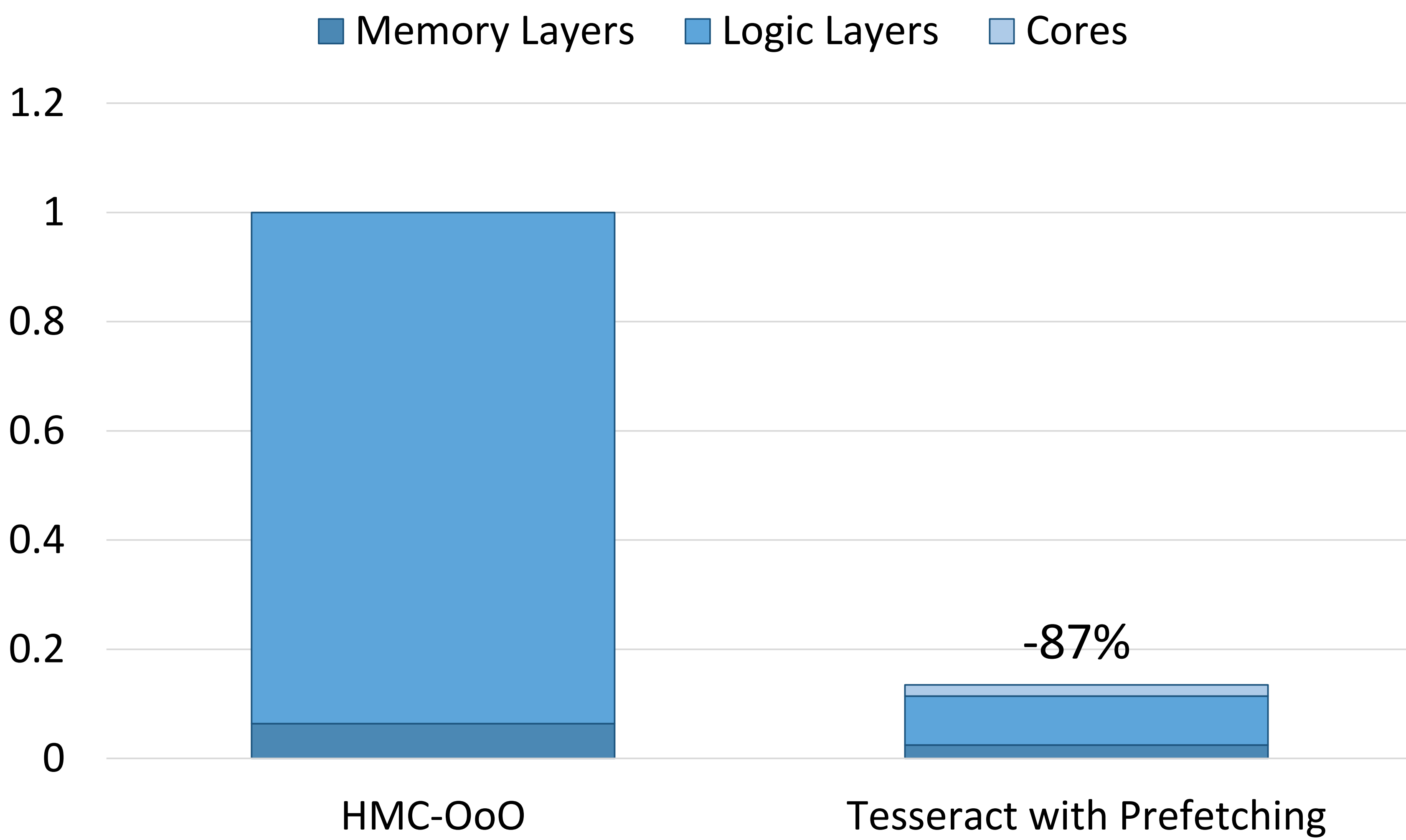




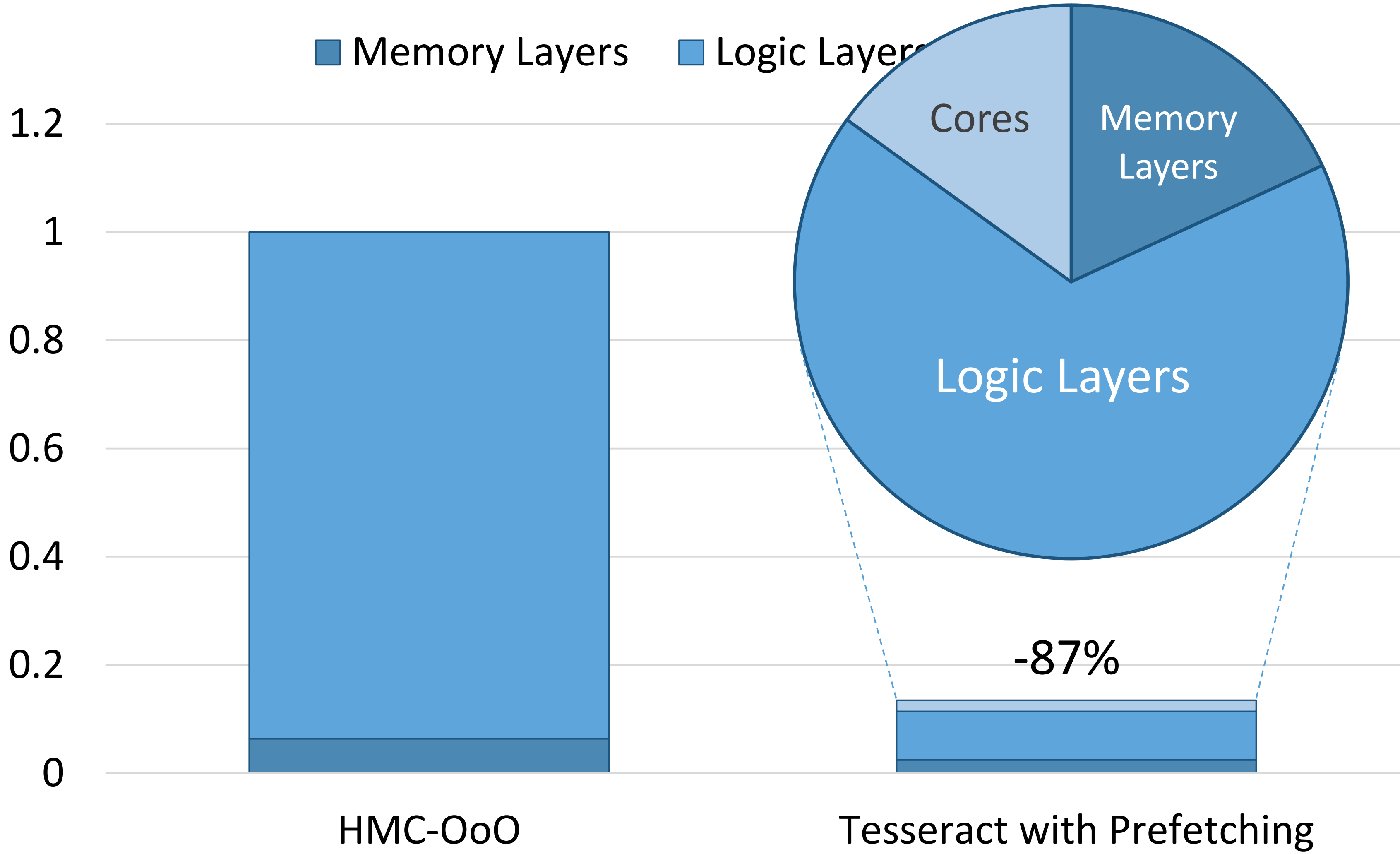
Scalability



Memory Energy Consumption



Memory Energy Consumption



Conclusion

- Revisiting the PIM concept in a new context
  - Cost-effective 3D integration of logic and memory
  - Graph processing workloads demanding high memory bandwidth
- Tesseract: scalable PIM for graph processing
  - Many in-order cores in a memory chip
  - New message passing mechanism for latency hiding
  - New hardware prefetchers for graph processing
  - Programming interface that exploits our hardware design
- Evaluations demonstrate the benefits of Tesseract
  - 14x performance improvement & 87% energy reduction
  - Scalable: *memory-capacity-proportional* performance

A Scalable Processing-in-Memory Accelerator for Parallel Graph Processing

Junwhan Ahn, Sungpack Hong*, Sungjoo Yoo,
Onur Mutlu+, Kiyoung Choi

Seoul National University *Oracle Labs +Carnegie Mellon University