Improving Locality of Irregular Updates with Hardware Assisted Propagation Blocking

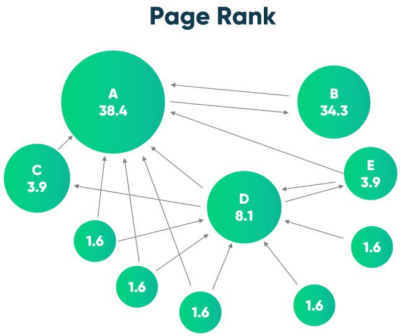
Vignesh Balaji Brandon Lucia

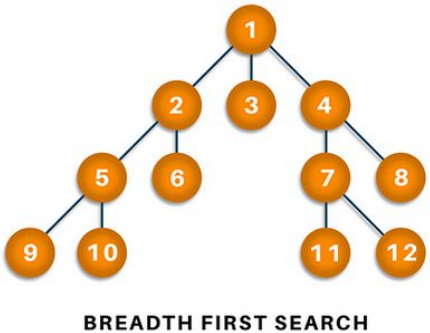
NVIDIA

Carnegie Mellon University

Problem: Irregular Updates Affect Many Applications

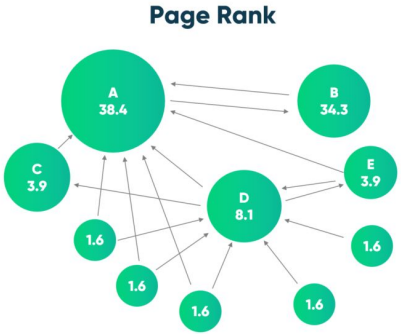
Problem: Irregular Updates Affect Many Applications

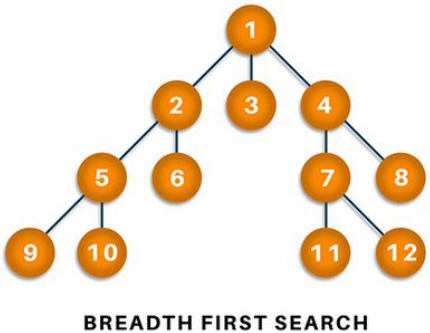




Graph Analytics

Problem: Irregular Updates Affect Many Applications

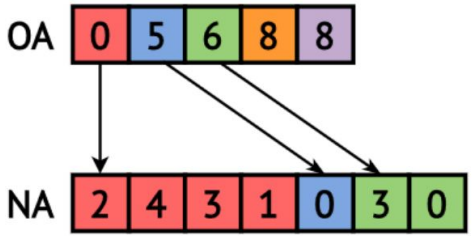




Graph Analytics

0	1
2	0
1	0
0	2
2	3
0	4
0	3

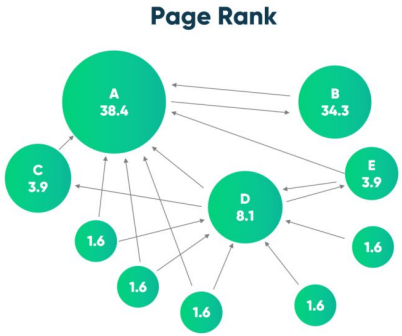
COO
(EdgeList)

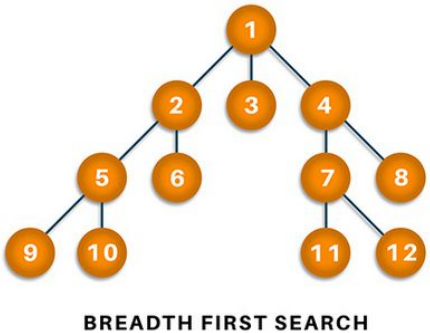


CSR
(Outgoing-neighbors)

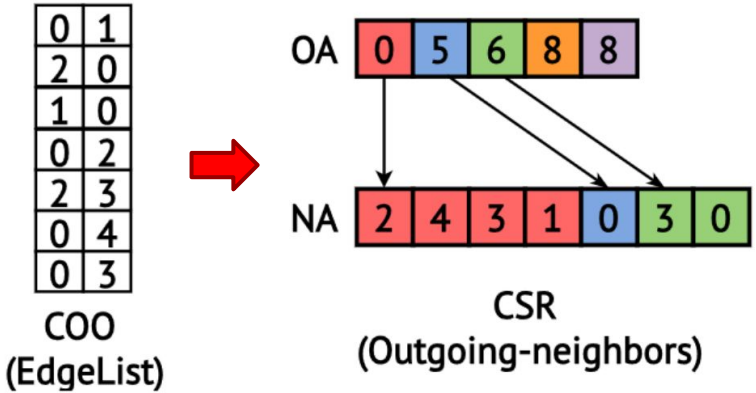
Pre-Processing

Problem: Irregular Updates Affect Many Applications

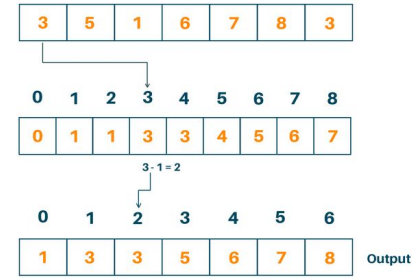




Graph Analytics

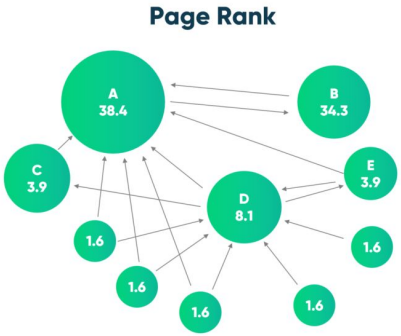


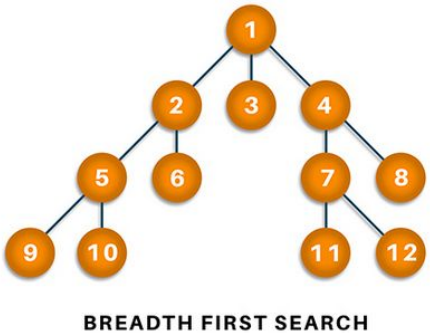
Pre-Processing



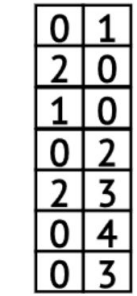
Integer Sort

Problem: Irregular Updates Affect Many Applications

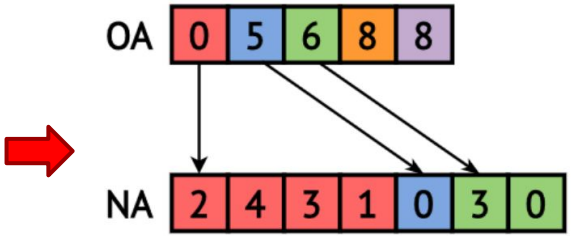




Graph Analytics

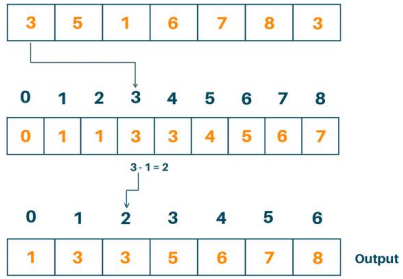


COO
(EdgeList)

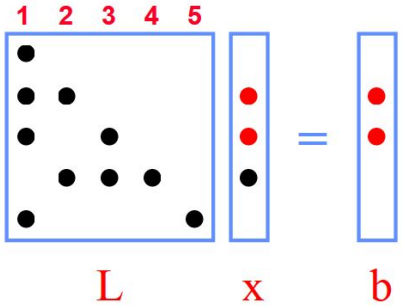


CSR
(Outgoing-neighbors)

Pre-Processing

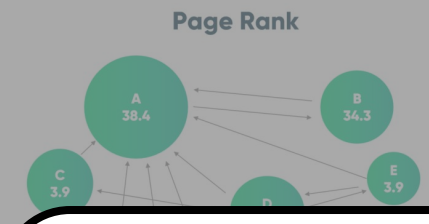


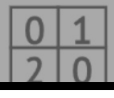
Integer Sort

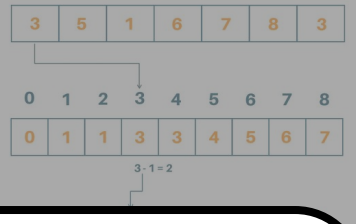


Sparse Linear Algebra

Problem: Irregular Updates Affect Many Applications



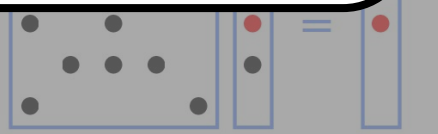




Problems with irregular updates:

BREADTH FIRST SEARCH

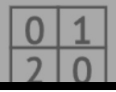
Graph Analytics

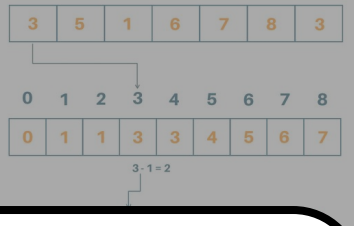


L x b

Sparse Linear Algebra

Problem: Irregular Updates Affect Many Applications

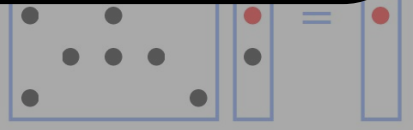




Problems with irregular updates:

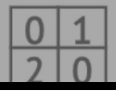
1) **Fine-grained accesses ⇒** ***Poor Spatial Locality***

Graph Analytics



Sparse Linear Algebra

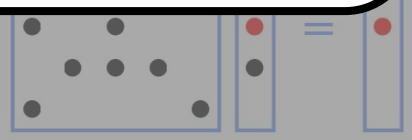
Problem: Irregular Updates Affect Many Applications



Problems with irregular updates:

1) **Fine-grained accesses ⇒** ***Poor Spatial Locality***
2) **Large data footprints ⇒** ***Poor Temporal Locality***

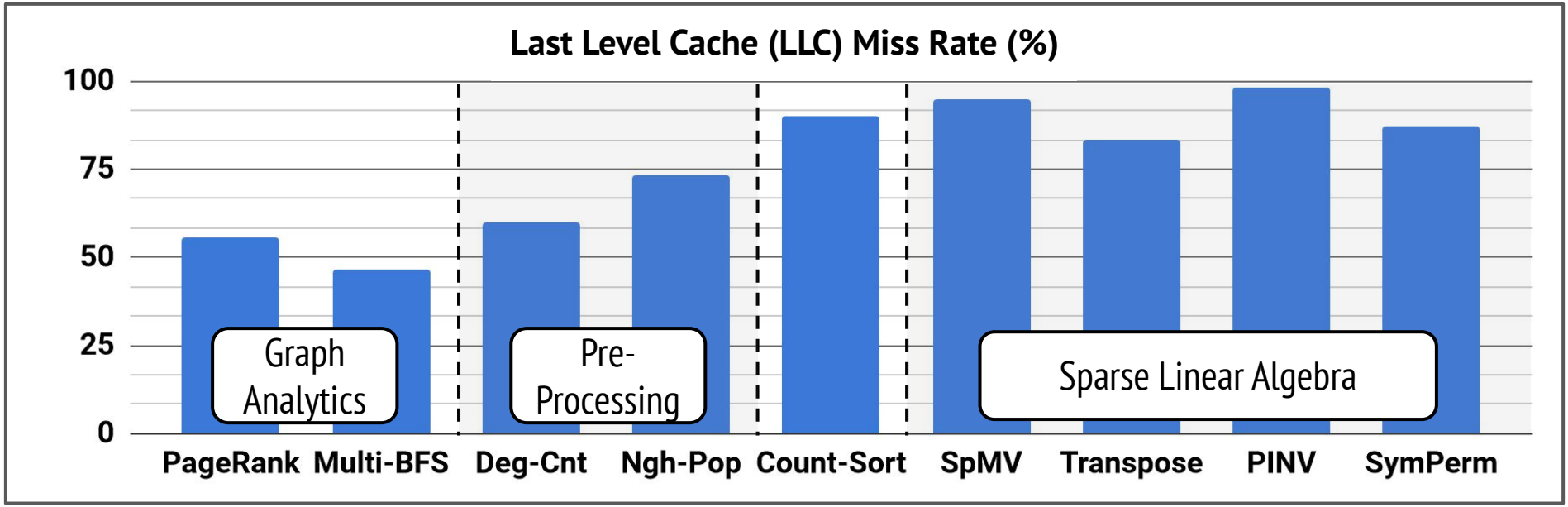
Graph Analytics



Sparse Linear Algebra

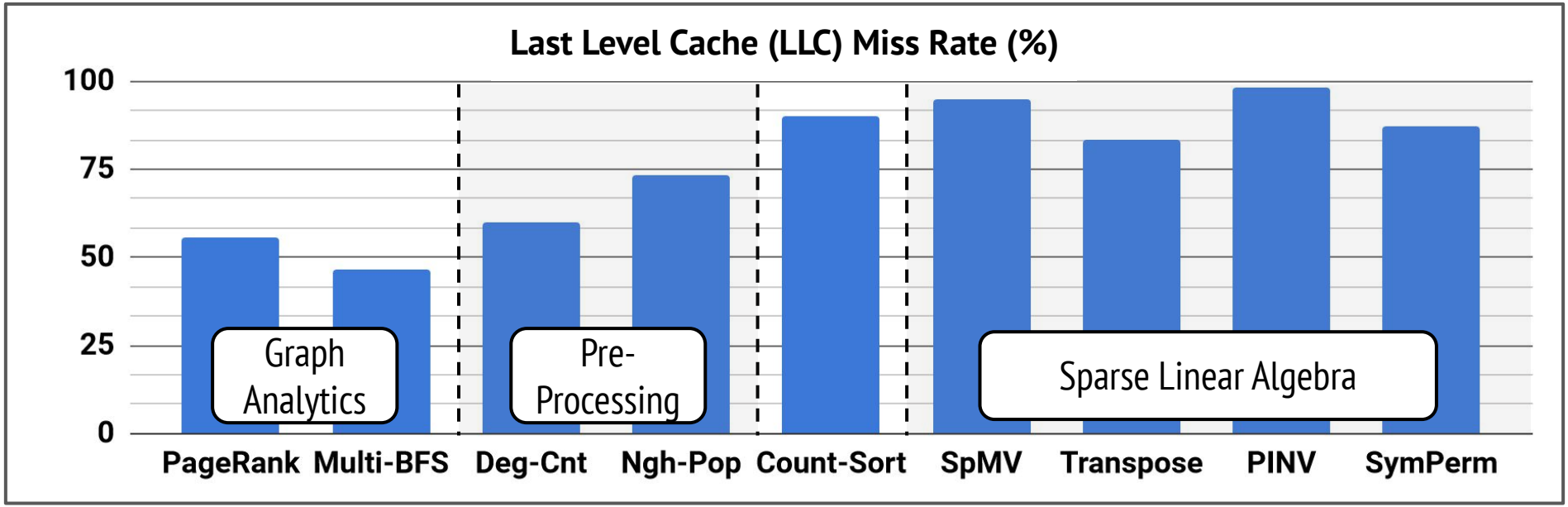
Poor Locality Due To Irregular Updates

Lower is Better



Poor Locality Due To Irregular Updates

Lower is Better



Poor LLC Locality leads to *long-latency* and *energy-intensive* DRAM Accesses

Outline

- **Poor Cache Locality Due To Irregular Updates ✓**
- Propagation Blocking Improves Locality
- Limitations of Propagation Blocking
- COBRA: Architecture Support For Optimizing Propagation Blocking

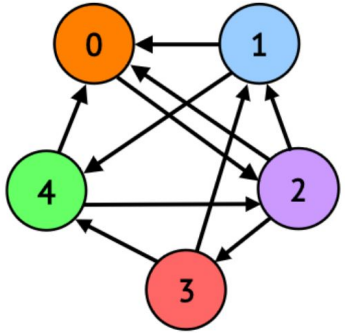
Outline

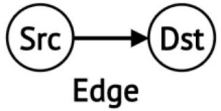
- Poor Cache Locality Due To Irregular Updates ✓
- **Propagation Blocking Improves Locality** ⬅
- Limitations of Propagation Blocking
- COBRA: Architecture Support For Optimizing Propagation Blocking

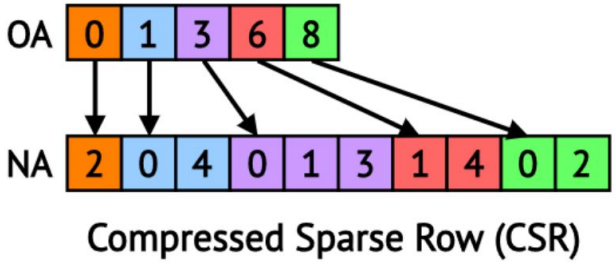
Propagation Blocking Overview

PageRank (*Push traversal*)

```
for src in G:
  for dst in out_neighs(src):
    dstData[dst] += srcData[src]
```





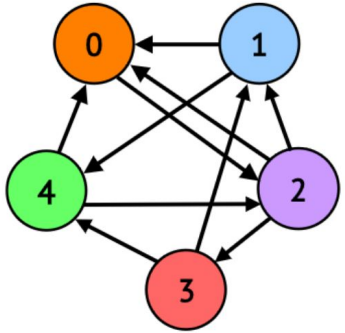


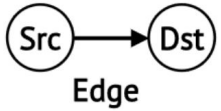
Propagation Blocking Overview

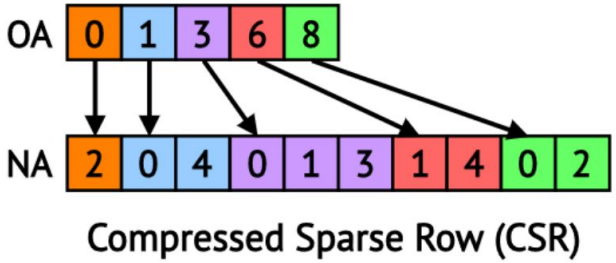
PageRank (*Push traversal*)

```
for src in G:
  for dst in out_neighs(src):
    dstData[dst] += srcData[src]
```

Streaming Reads







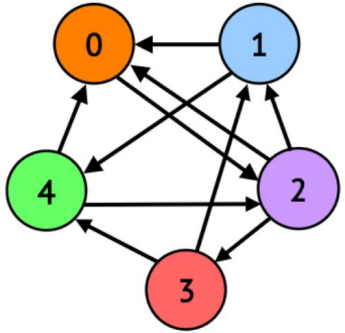
Propagation Blocking Overview

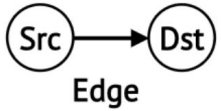
PageRank (*Push traversal*)

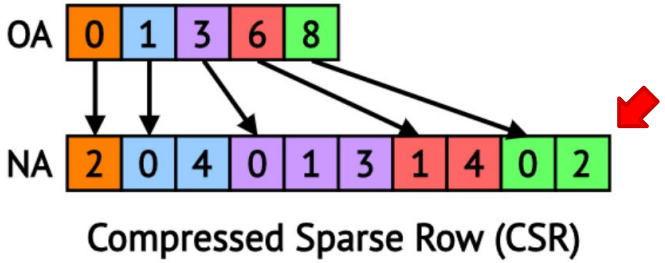
```
for src in G:
  for dst in out_neighs(src):
    dstData[dst] += srcData[src]
```

Irregular Updates

Streaming Reads







Propagation Blocking Overview

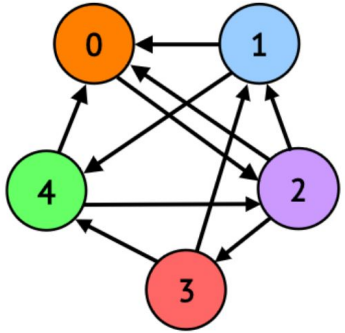
PageRank (*Push traversal*)

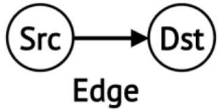
```
for src in G:
  for dst in out_neighs(src):
    dstData[dst] += srcData[src]
```

Irregular Updates

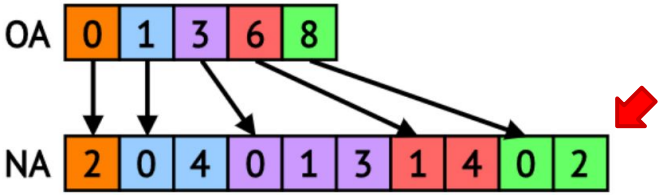
Streaming Reads

Irregular Update Range = **|V|**





Edge



Compressed Sparse Row (CSR)

Propagation Blocking Overview

PageRank (*Push traversal*)

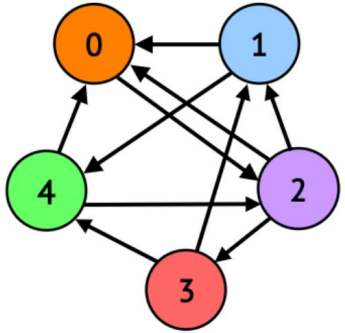
```
for src in G:
  for dst in out_neighs(src):
    dstData[dst] += srcData[src]
```

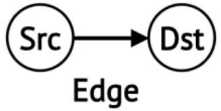
Irregular Updates

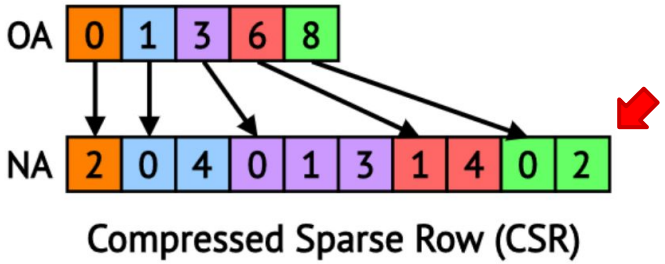
Streaming Reads

Irregular Update Range = **|V|**

Irregular Updates will not fit in an On-Chip Cache







Propagation Blocking Overview

PageRank (*Push traversal*)

```
for src in G:
  for dst in out_neighs(src):
    dstData[dst] += srcData[src]
```

Propagation Blocking Overview

PageRank (*Push traversal*)

```
for src in G:
  for dst in out_neighs(src):
    dstData[dst] += srcData[src]
```

PB (Phase-I) -- Binning

```
for src in G:
  for dst in out_neighs(src):
    updVal = srcData[src]
    binID  = dst / BinRange
    bins[binID].append(dst, updVal)
```

PB (Phase-II) -- Accumulate

```
for bin in bins:
  for tuple in bin:
    (ind, updval) = tuple
    dstData[ind] += updval
```

Propagation Blocking Overview

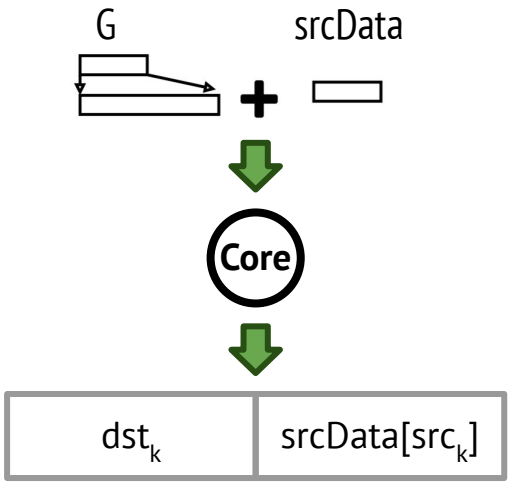
PB (Phase-I) -- Binning

```
for src in G:
  for dst in out_neighs(src):
    updVal = srcData[src]
    binID  = dst / BinRange
    bins[binID].append(dst, updVal)
```

PB (Phase-II) -- Accumulate

```
for bin in bins:
  for tuple in bin:
    (ind, updval) = tuple
    dstData[ind] += updval
```

Propagation Blocking Overview



Update Tuple

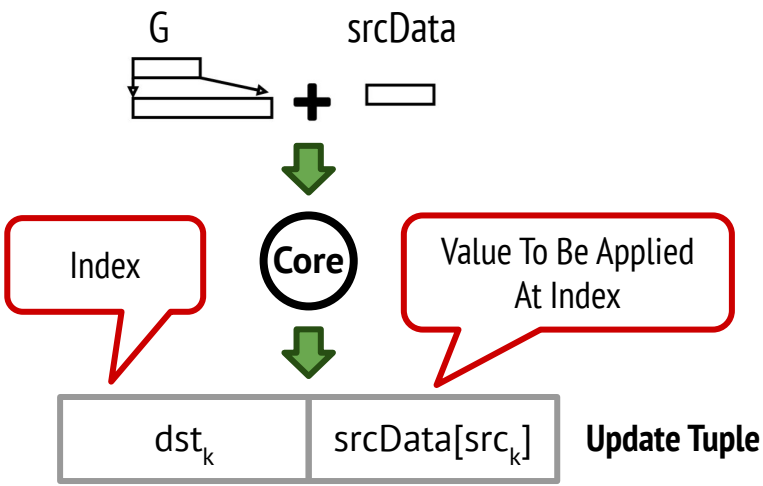
PB (Phase-I) -- Binning

```
for src in G:
  for dst in out_neighs(src):
    updVal = srcData[src]
    binID  = dst / BinRange
    bins[binID].append(dst, updVal)
```

PB (Phase-II) -- Accumulate

```
for bin in bins:
  for tuple in bin:
    (ind, updval) = tuple
    dstData[ind] += updval
```

Propagation Blocking Overview



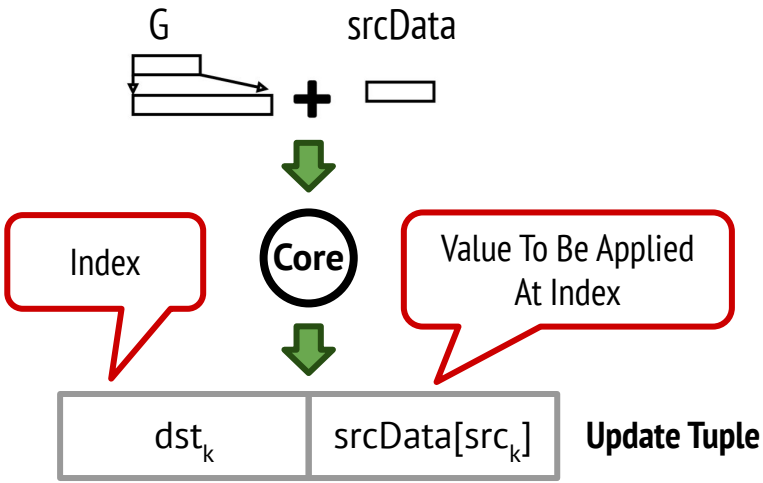
PB (Phase-I) -- Binning

```
for src in G:
  for dst in out_neighs(src):
    updVal = srcData[src]
    binID  = dst / BinRange
    bins[binID].append(dst, updVal)
```

PB (Phase-II) -- Accumulate

```
for bin in bins:
  for tuple in bin:
    (ind, updval) = tuple
    dstData[ind] += updval
```

Propagation Blocking Overview

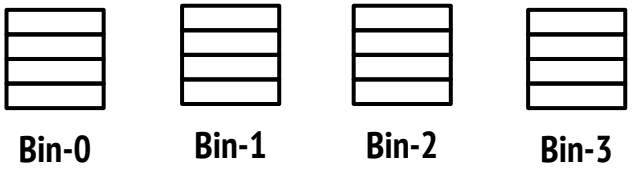


PB (Phase-I) -- Binning

```
for src in G:
  for dst in out_neighs(src):
    updVal = srcData[src]
    binID  = dst / BinRange
    bins[binID].append(dst, updVal)
```

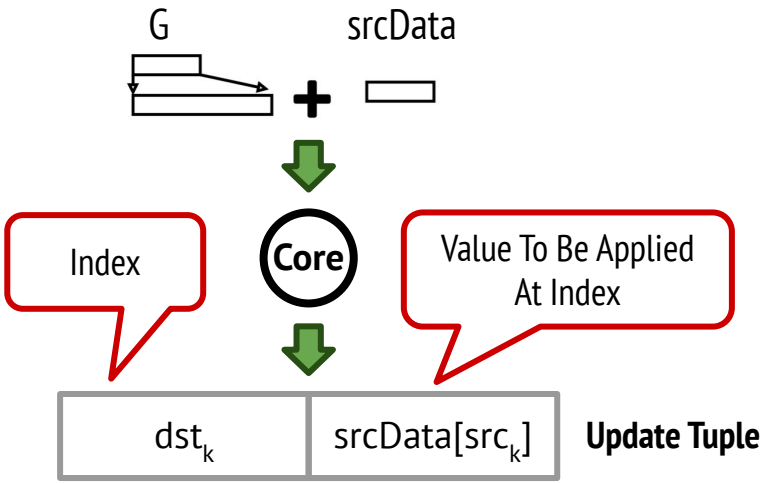
PB (Phase-II) -- Accumulate

```
for bin in bins:
  for tuple in bin:
    (ind, updval) = tuple
    dstData[ind] += updval
```



Bins store update tuples of disjoint range of indices

Propagation Blocking Overview

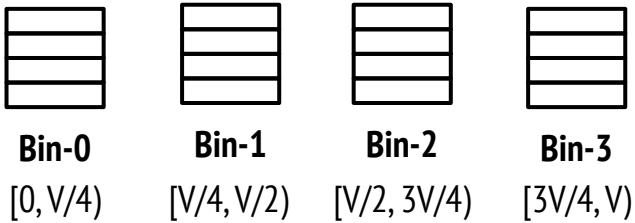


PB (Phase-I) -- Binning

```
for src in G:
  for dst in out_neighs(src):
    updVal = srcData[src]
    binID  = dst / BinRange
    bins[binID].append(dst, updVal)
```

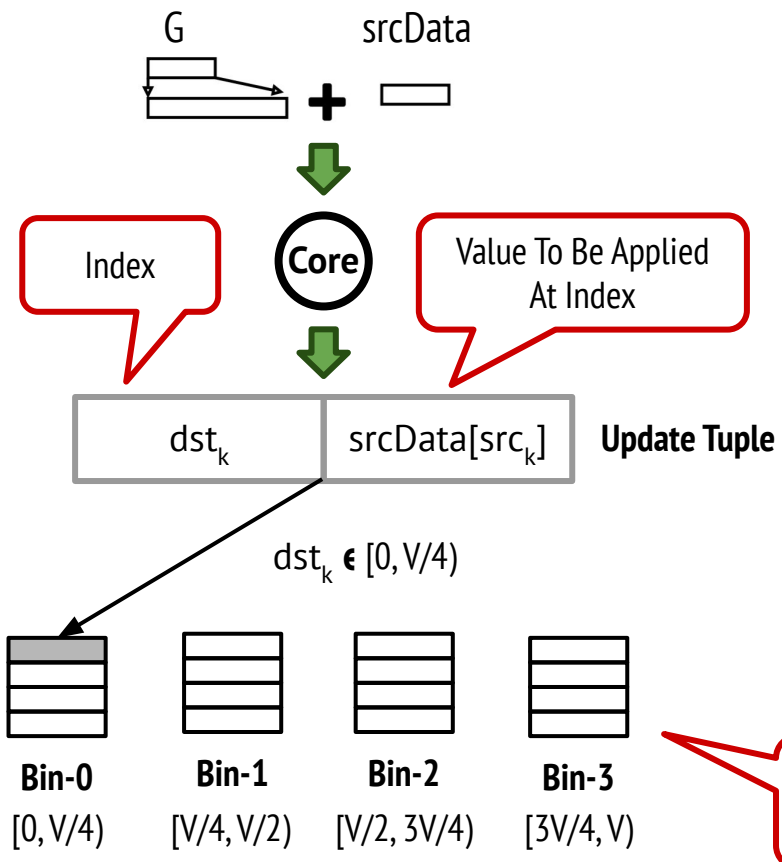
PB (Phase-II) -- Accumulate

```
for bin in bins:
  for tuple in bin:
    (ind, updval) = tuple
    dstData[ind] += updval
```



Bins store update tuples of disjoint range of indices

Propagation Blocking Overview



PB (Phase-I) -- Binning

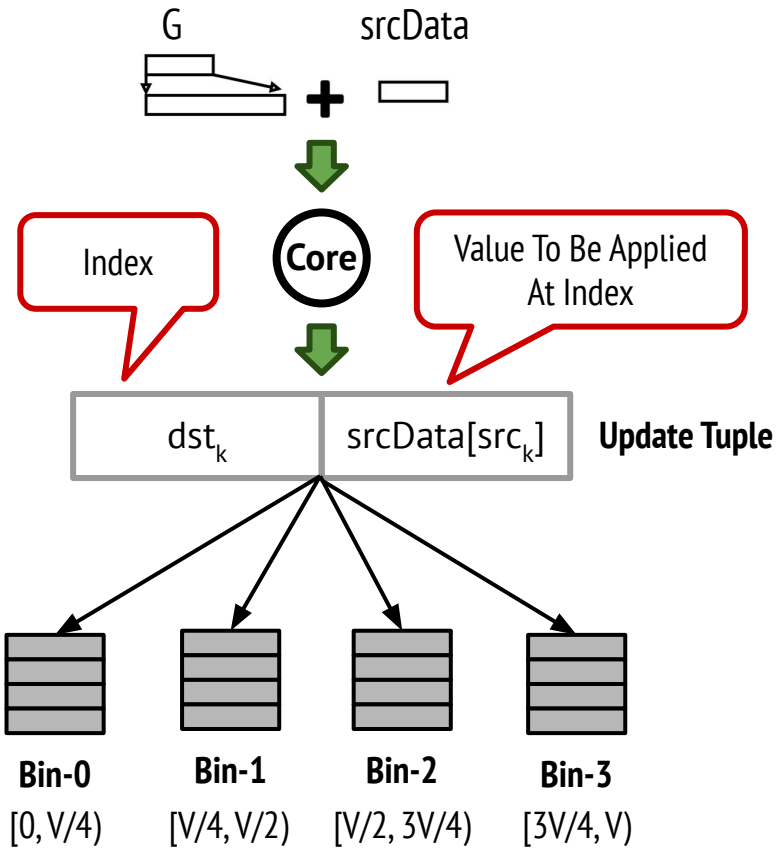
```
for src in G:
  for dst in out_neighs(src):
    updVal = srcData[src]
    binID  = dst / BinRange
    bins[binID].append(dst, updVal)
```

PB (Phase-II) -- Accumulate

```
for bin in bins:
  for tuple in bin:
    (ind, updval) = tuple
    dstData[ind] += updval
```

Bins store update tuples of disjoint range of indices

Propagation Blocking Overview



PB (Phase-I) -- Binning

```
for src in G:
  for dst in out_neighs(src):
    updVal = srcData[src]
    binID  = dst / BinRange
    bins[binID].append(dst, updVal)
```

PB (Phase-II) -- Accumulate

```
for bin in bins:
  for tuple in bin:
    (ind, updval) = tuple
    dstData[ind] += updval
```

Propagation Blocking Overview

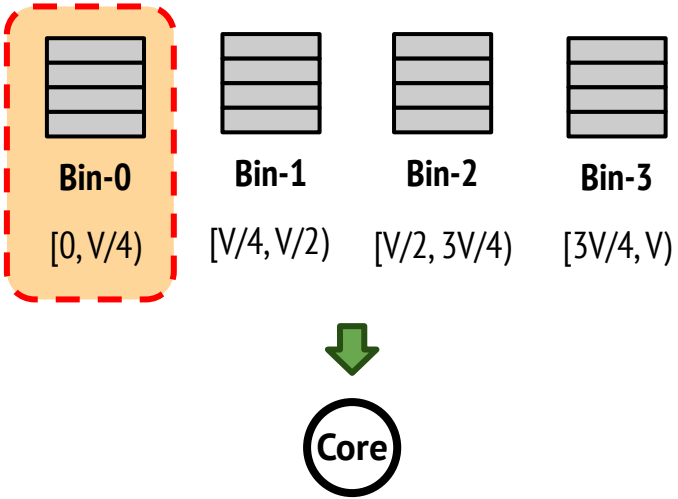
PB (Phase-I) -- Binning

```
for src in G:
  for dst in out_neighs(src):
    updVal = srcData[src]
    binID  = dst / BinRange
    bins[binID].append(dst, updVal)
```

PB (Phase-II) -- Accumulate

```
for bin in bins:
  for tuple in bin:
    (ind, updval) = tuple
    dstData[ind] += updval
```

Propagation Blocking Overview



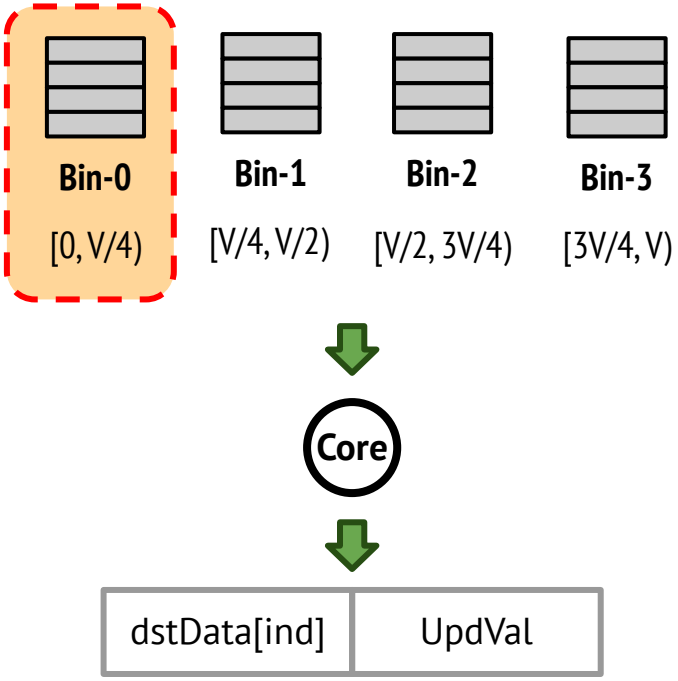
PB (Phase-I) -- Binning

```
for src in G:
  for dst in out_neighs(src):
    updVal = srcData[src]
    binID  = dst / BinRange
    bins[binID].append(dst, updVal)
```

PB (Phase-II) -- Accumulate

```
for bin in bins:
  for tuple in bin:
    (ind, updval) = tuple
    dstData[ind] += updval
```

Propagation Blocking Overview



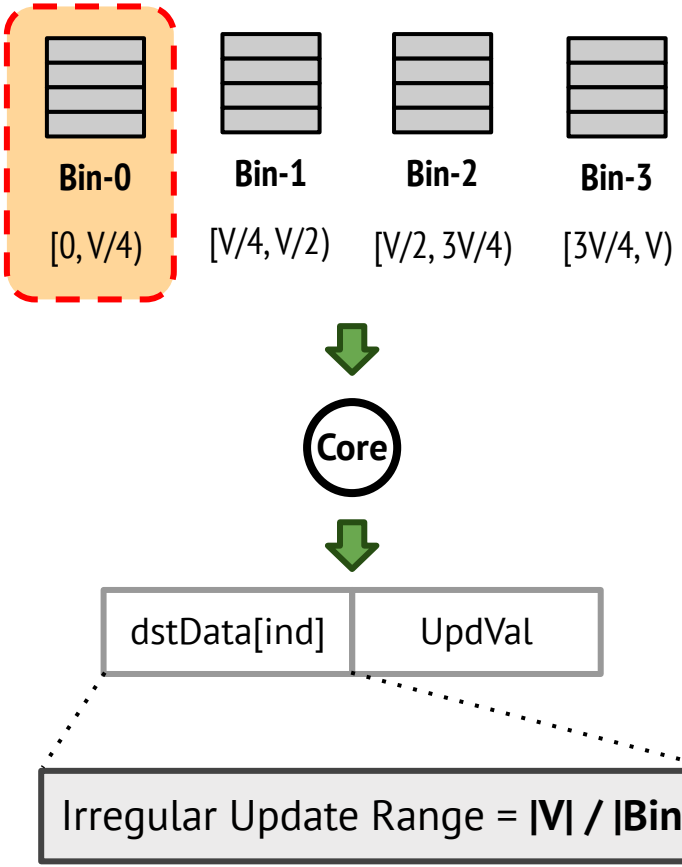
PB (Phase-I) -- Binning

```
for src in G:
  for dst in out_neighs(src):
    updVal = srcData[src]
    binID  = dst / BinRange
    bins[binID].append(dst, updVal)
```

PB (Phase-II) -- Accumulate

```
for bin in bins:
  for tuple in bin:
    (ind, updval) = tuple
    dstData[ind] += updval
```

Propagation Blocking Overview



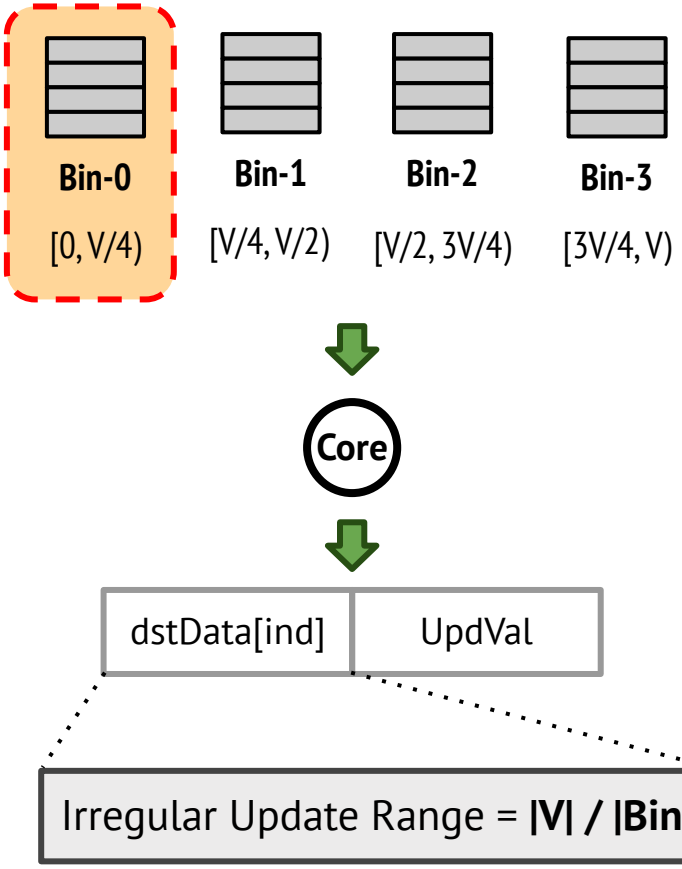
PB (Phase-I) -- Binning

```
for src in G:
  for dst in out_neighs(src):
    updVal = srcData[src]
    binID  = dst / BinRange
    bins[binID].append(dst, updVal)
```

PB (Phase-II) -- Accumulate

```
for bin in bins:
  for tuple in bin:
    (ind, updval) = tuple
    dstData[ind] += updval
```

Propagation Blocking Overview



PB (Phase-I) -- Binning

```
for src in G:
  for dst in out_neighs(src):
    updVal = srcData[src]
    binID  = dst / BinRange
    bins[binID].append(dst, updVal)
```

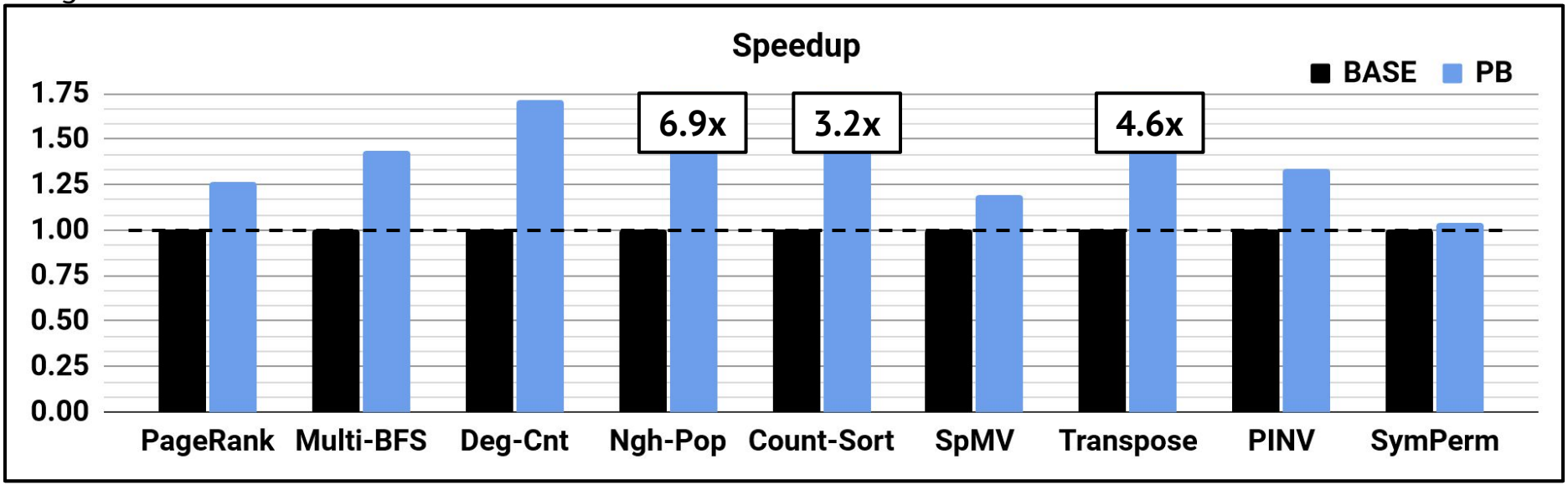
PB (Phase-II) -- Accumulate

```
for bin in bins:
  for tuple in bin:
    (ind, updval) = tuple
    dstData[ind] += updval
```

PB reorganizes irregular updates to improve locality

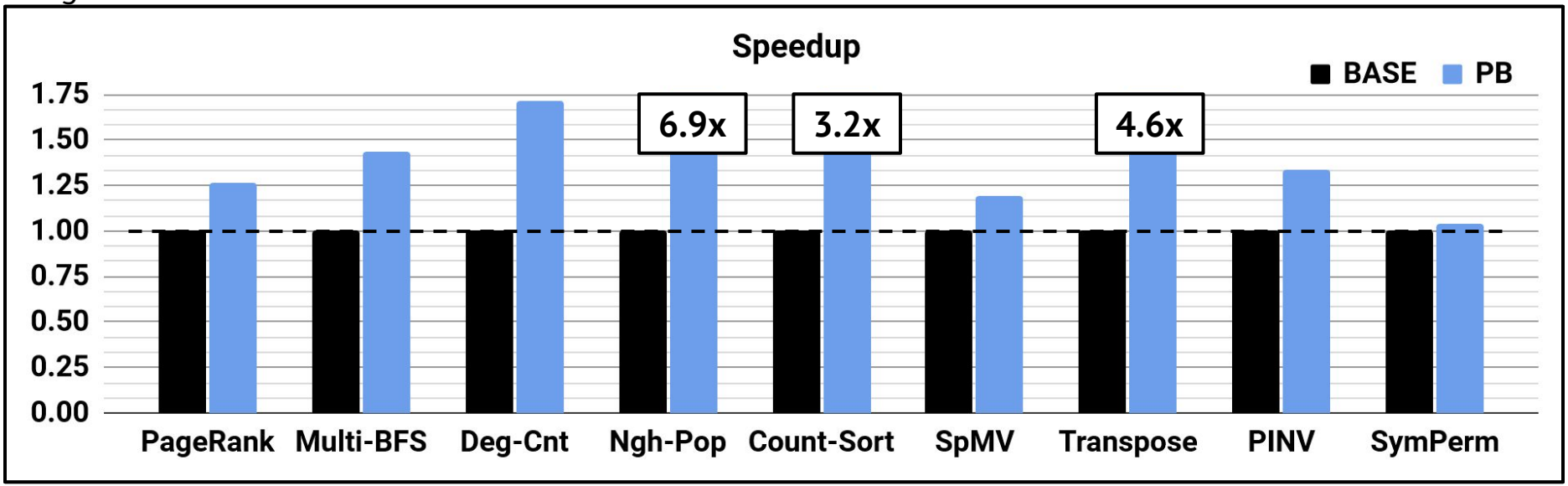
Propagation Blocking Optimizes Apps with Irregular Updates

Higher is Better



Propagation Blocking Optimizes Apps with Irregular Updates

Higher is Better



Propagation Blocking improves locality of irregular updates

Outline

- Poor Cache Locality Due To Irregular Updates ✓
- **Propagation Blocking Improves Locality ✓**
- Limitations of Propagation Blocking
- COBRA: Architecture Support For Optimizing Propagation Blocking

Outline

- Poor Cache Locality Due To Irregular Updates ✓
- Propagation Blocking Improves Locality ✓
- **Limitations of Propagation Blocking** ⬅
- COBRA: Architecture Support For Optimizing Propagation Blocking

Problem: PB Is Unable To Use the Optimal Number of Bins

PB (Phase-I) -- Binning

```
for src in G:
  for dst in out_neighs(src):
    updVal = srcData[src]
    binID  = dst / BinRange
    bins[binID].append(dst, updVal)
```

PB (Phase-II) -- Accumulate

```
for bin in bins:
  for tuple in bin:
    (ind, updval) = tuple
    dstData[ind] += updval
```

Problem: PB Is Unable To Use the Optimal Number of Bins

PB (Phase-I) -- Binning

```
for src in G:
  for dst in out_neighs(src):
    updVal = srcData[src]
    binID  = dst / BinRange
    bins[binID].append(dst, updVal)
```

PB (Phase-II) -- Accumulate

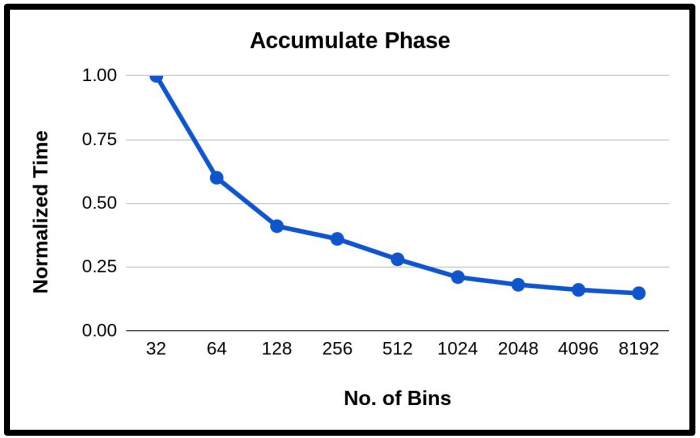
```
for bin in bins:
  for tuple in bin:
    (ind, updval) = tuple
    dstData[ind] += updval
```

Irregular Update Range = **|V| / |Bins|**

Problem: PB Is Unable To Use the Optimal Number of Bins

PB (Phase-I) -- Binning

```
for src in G:
  for dst in out_neighs(src):
    updVal = srcData[src]
    binID  = dst / BinRange
    bins[binID].append(dst, updVal)
```



PB (Phase-II) -- Accumulate

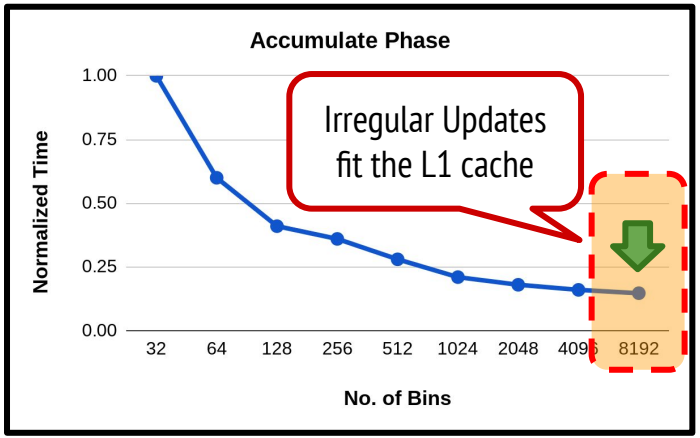
```
for bin in bins:
  for tuple in bin:
    (ind, updval) = tuple
    dstData[ind] += updval
```

Irregular Update Range = **|V| / |Bins|**

Problem: PB Is Unable To Use the Optimal Number of Bins

PB (Phase-I) -- Binning

```
for src in G:
  for dst in out_neighs(src):
    updVal = srcData[src]
    binID  = dst / BinRange
    bins[binID].append(dst, updVal)
```

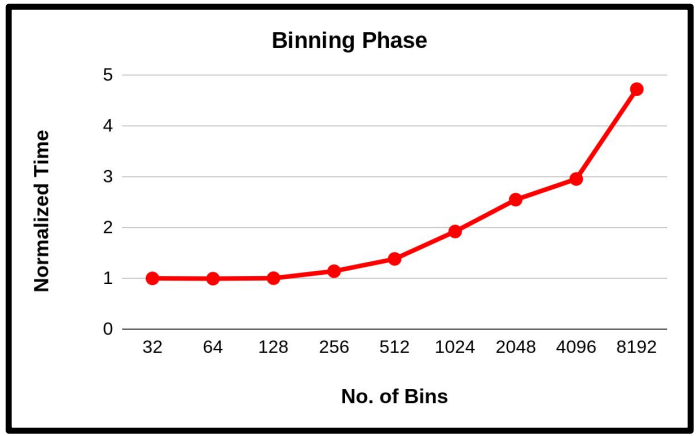


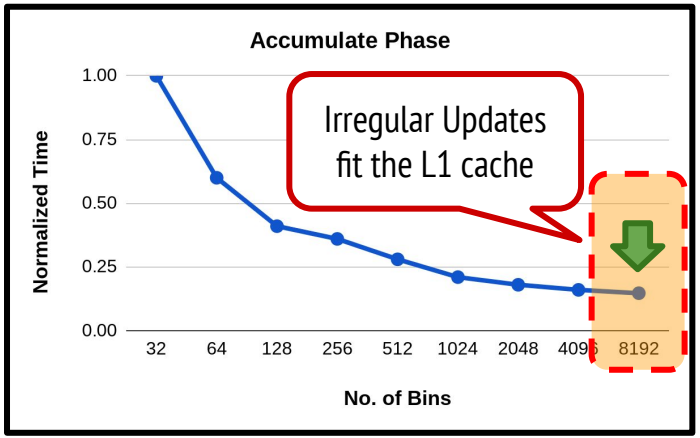
PB (Phase-II) -- Accumulate

```
for bin in bins:
  for tuple in bin:
    (ind, updval) = tuple
    dstData[ind] += updval
```

Irregular Update Range = **|V| / |Bins|**

Problem: PB Is Unable To Use the Optimal Number of Bins





PB (Phase-I) -- Binning

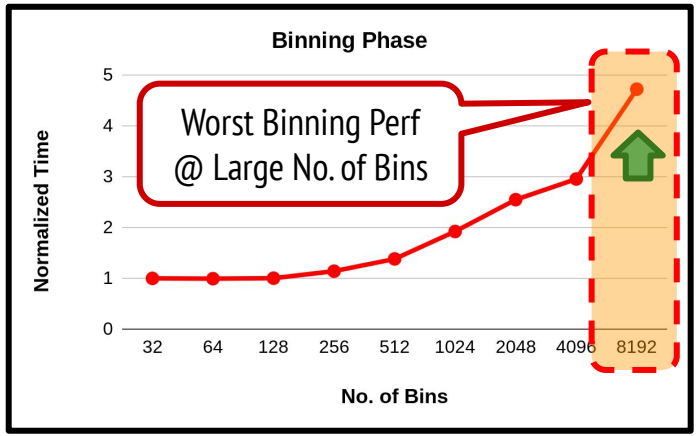
```
for src in G:
  for dst in out_neighs(src):
    updVal = srcData[src]
    binID  = dst / BinRange
    bins[binID].append(dst, updVal)
```

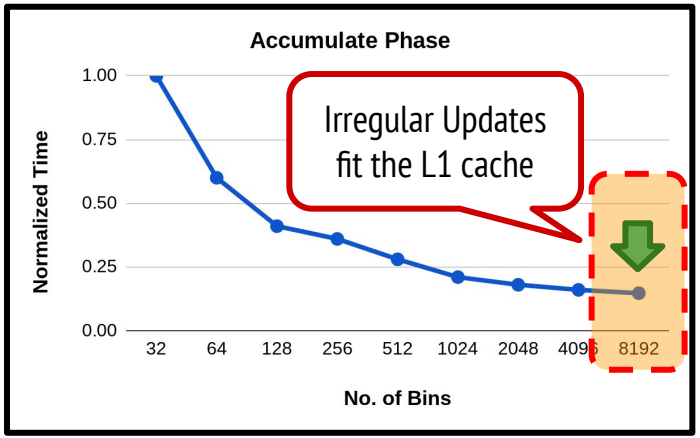
PB (Phase-II) -- Accumulate

```
for bin in bins:
  for tuple in bin:
    (ind, updval) = tuple
    dstData[ind] += updval
```

Irregular Update Range = **|V| / |Bins|**

Problem: PB Is Unable To Use the Optimal Number of Bins





PB (Phase-I) -- Binning

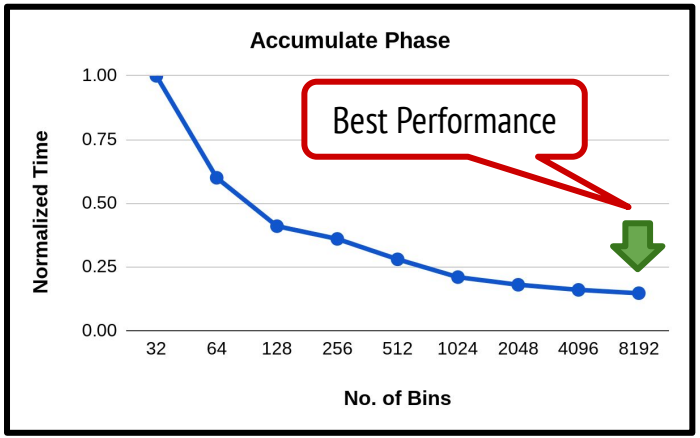
```
for src in G:
  for dst in out_neighs(src):
    updVal = srcData[src]
    binID  = dst / BinRange
    bins[binID].append(dst, updVal)
```

PB (Phase-II) -- Accumulate

```
for bin in bins:
  for tuple in bin:
    (ind, updval) = tuple
    dstData[ind] += updval
```

Irregular Update Range = **|V| / |Bins|**

Problem: PB Is Unable To Use the Optimal Number of Bins



PB (Phase-I) -- Binning

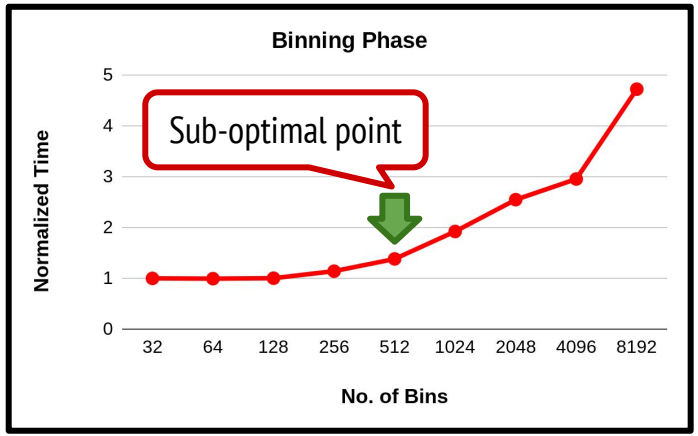
```
for src in G:
  for dst in out_neighs(src):
    updVal = srcData[src]
    binID  = dst / BinRange
    bins[binID].append(dst, updVal)
```

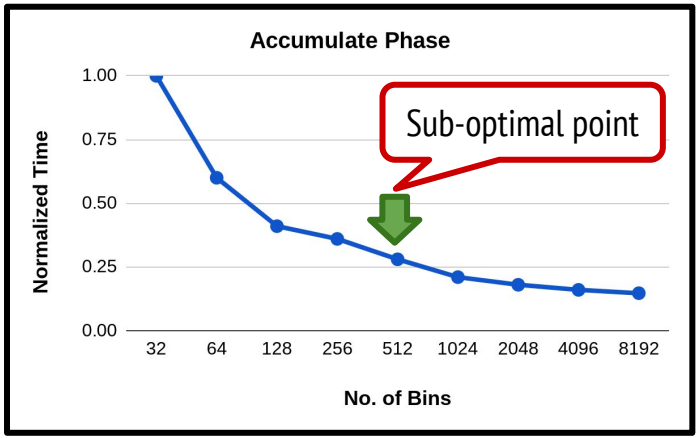
PB (Phase-II) -- Accumulate

```
for bin in bins:
  for tuple in bin:
    (ind, updval) = tuple
    dstData[ind] += updval
```

Irregular Update Range = **|V| / |Bins|**

Problem: PB Is Unable To Use the Optimal Number of Bins





PB (Phase-I) -- Binning

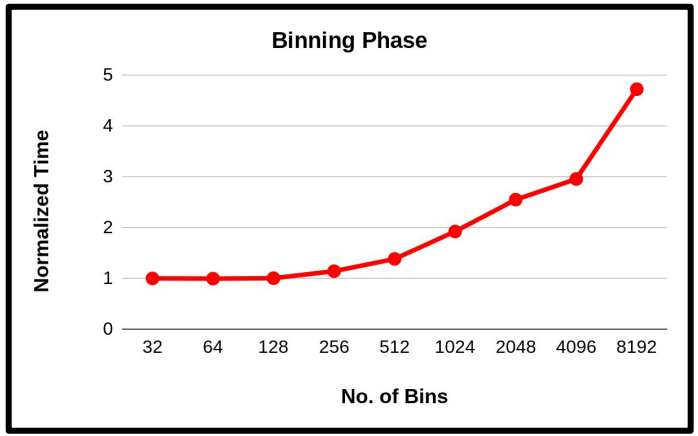
```
for src in G:
  for dst in out_neighs(src):
    updVal = srcData[src]
    binID  = dst / BinRange
    bins[binID].append(dst, updVal)
```

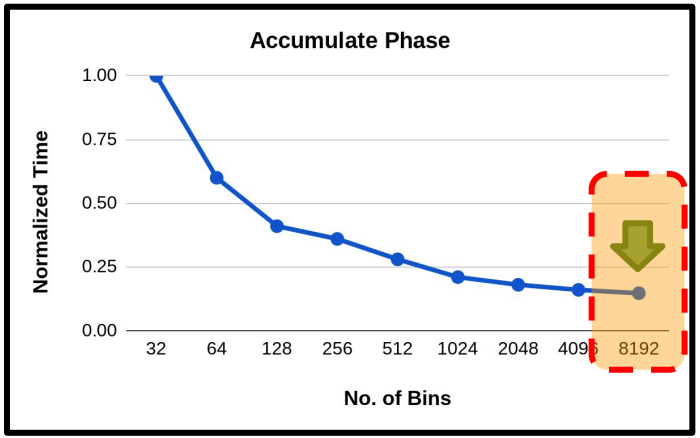
PB (Phase-II) -- Accumulate

```
for bin in bins:
  for tuple in bin:
    (ind, updval) = tuple
    dstData[ind] += updval
```

Irregular Update Range = **|V| / |Bins|**

Problem: PB Is Unable To Use the Optimal Number of Bins





PB (Phase-I) -- Binning

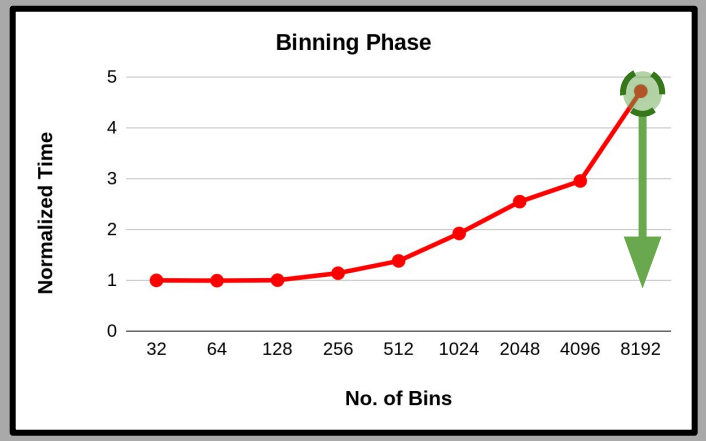
```
for src in G:
  for dst in out_neighs(src):
    updVal = srcData[src]
    binID  = dst / BinRange
    bins[binID].append(dst, updVal)
```

PB (Phase-II) -- Accumulate

```
for bin in bins:
  for tuple in bin:
    (ind, updval) = tuple
    dstData[ind] += updval
```

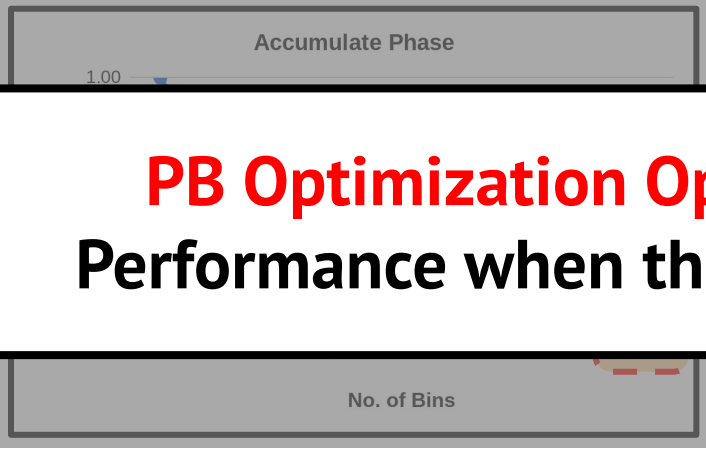
Irregular Update Range = **|V| / |Bins|**

Problem: PB Is Unable To Use the Optimal Number of Bins



PB (Phase-I) -- Binning

```
for src in G:
  for dst in out_neighs(src):
    updVal = srcData[src]
    binID  = dst / BinRange
    bins[binID].append(dst, updVal)
```



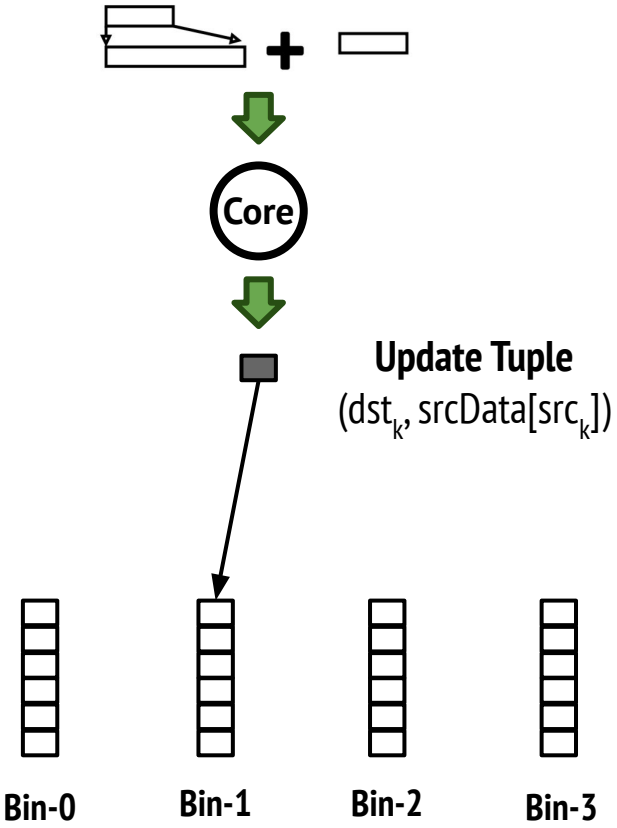
PB (Phase-II) -- Accumulate

```
for bin in bins:
```

Irregular Update Range = **|V| / |Bins|**

PB Optimization Opportunity: Improve Binning Performance when there are a large number of Bins

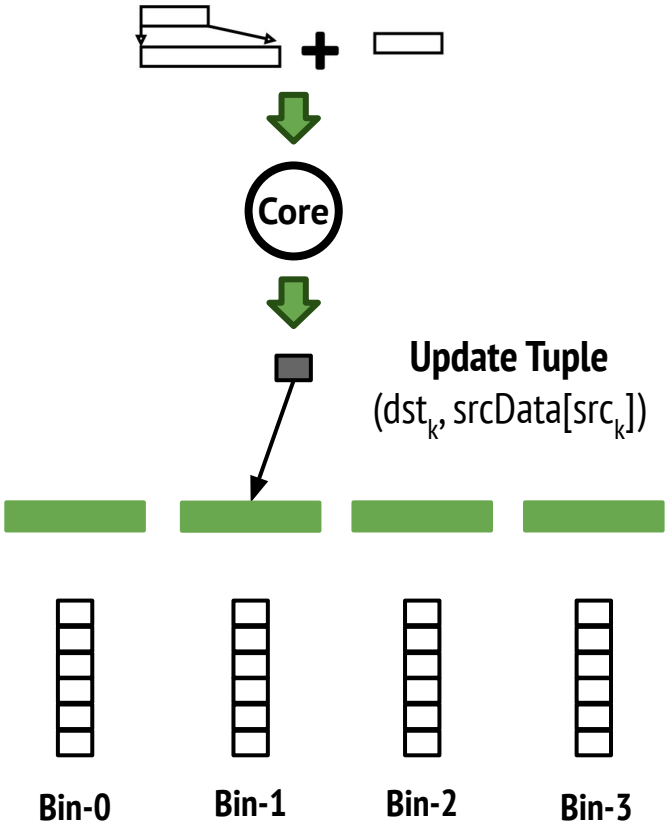
Reason For Poor Binning Performance With Many Bins



PB (Phase-I) -- Binning

```
for src in G:
  for dst in out_neighs(src):
    updVal = srcData[src]
    binID  = dst / BinRange
    bins[binID].append(dst, updVal)
```

Reason For Poor Binning Performance With Many Bins



PB (Phase-I) -- Binning

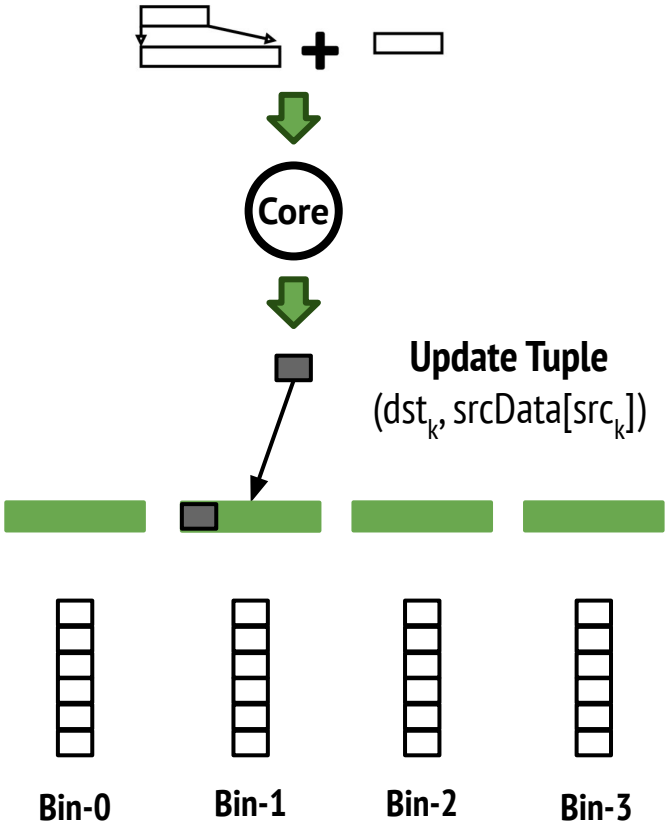
```
for src in G:
  for dst in out_neighs(src):
    updVal = srcData[src]
    binID  = dst / BinRange
    bins[binID].append(dst, updVal)
```

Coalescing Buffers
(C-Buffers)

Cacheline-sized containers

1 C-Buffer / Bin

Reason For Poor Binning Performance With Many Bins



PB (Phase-I) -- Binning

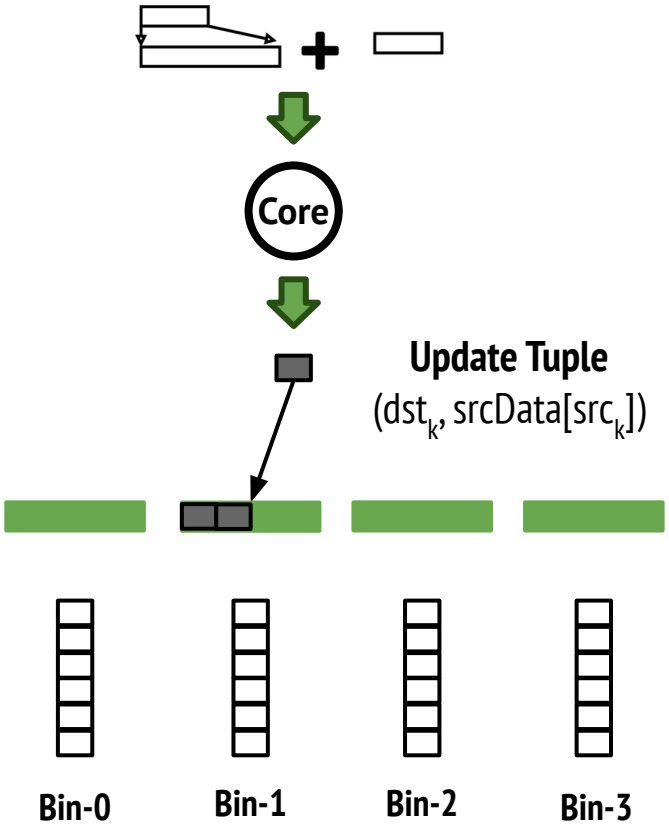
```
for src in G:
  for dst in out_neighs(src):
    updVal = srcData[src]
    binID  = dst / BinRange
    bins[binID].append(dst, updVal)
```

Coalescing Buffers
(C-Buffers)

Cacheline-sized containers

1 C-Buffer / Bin

Reason For Poor Binning Performance With Many Bins



PB (Phase-I) -- Binning

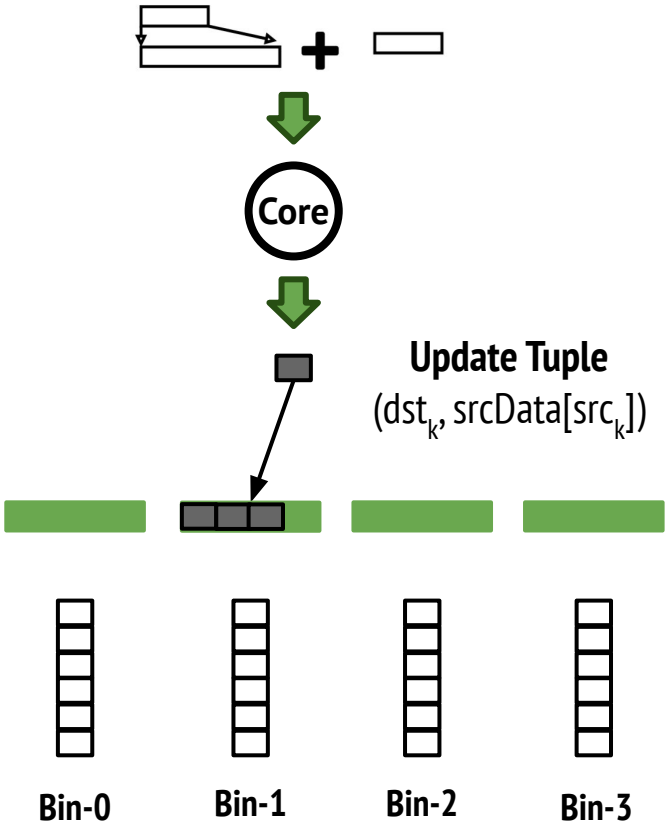
```
for src in G:
  for dst in out_neighs(src):
    updVal = srcData[src]
    binID  = dst / BinRange
    bins[binID].append(dst, updVal)
```

Coalescing Buffers
(C-Buffers)

Cacheline-sized containers

1 C-Buffer / Bin

Reason For Poor Binning Performance With Many Bins



PB (Phase-I) -- Binning

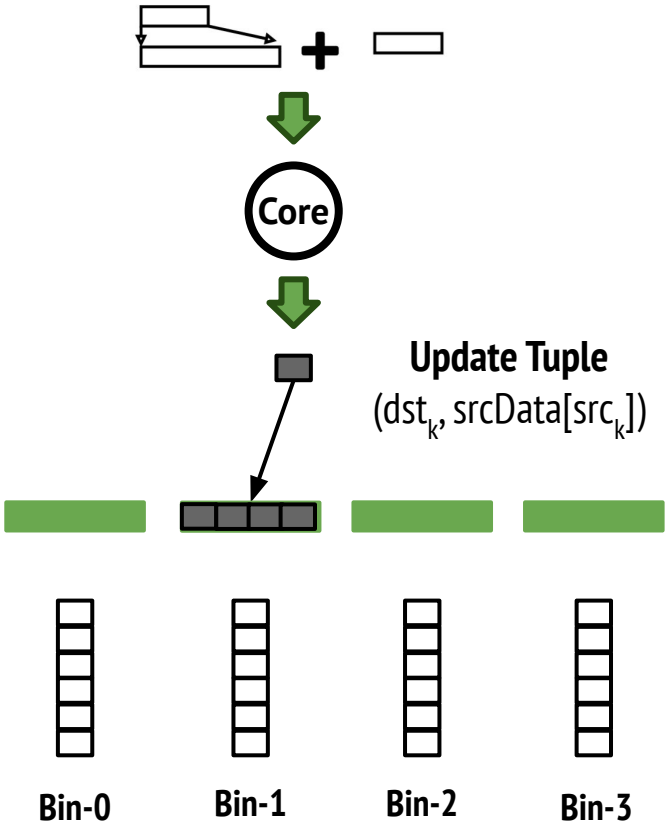
```
for src in G:
  for dst in out_neighs(src):
    updVal = srcData[src]
    binID  = dst / BinRange
    bins[binID].append(dst, updVal)
```

Coalescing Buffers
(C-Buffers)

Cacheline-sized containers

1 C-Buffer / Bin

Reason For Poor Binning Performance With Many Bins



PB (Phase-I) -- Binning

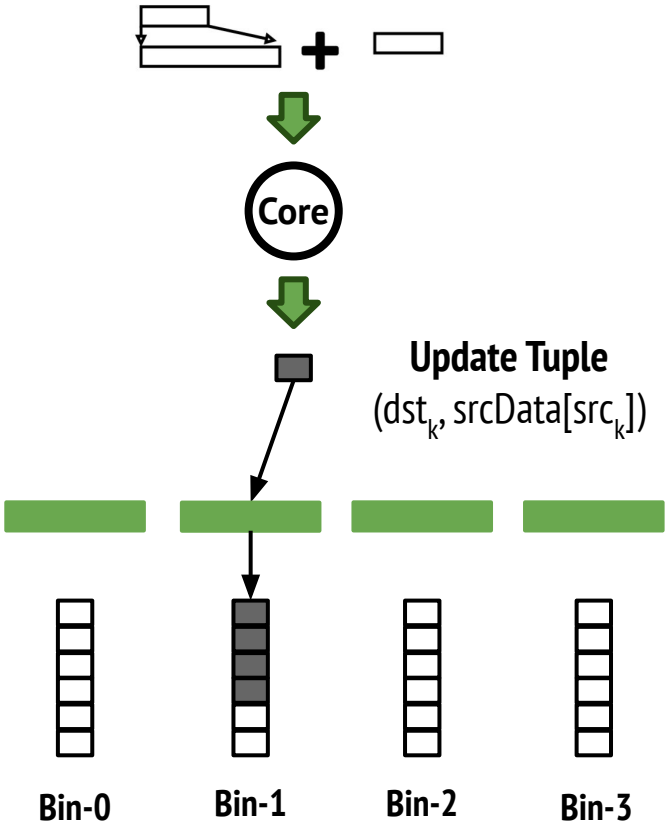
```
for src in G:
  for dst in out_neighs(src):
    updVal = srcData[src]
    binID  = dst / BinRange
    bins[binID].append(dst, updVal)
```

Coalescing Buffers
(C-Buffers)

Cacheline-sized containers

1 C-Buffer / Bin

Reason For Poor Binning Performance With Many Bins



PB (Phase-I) -- Binning

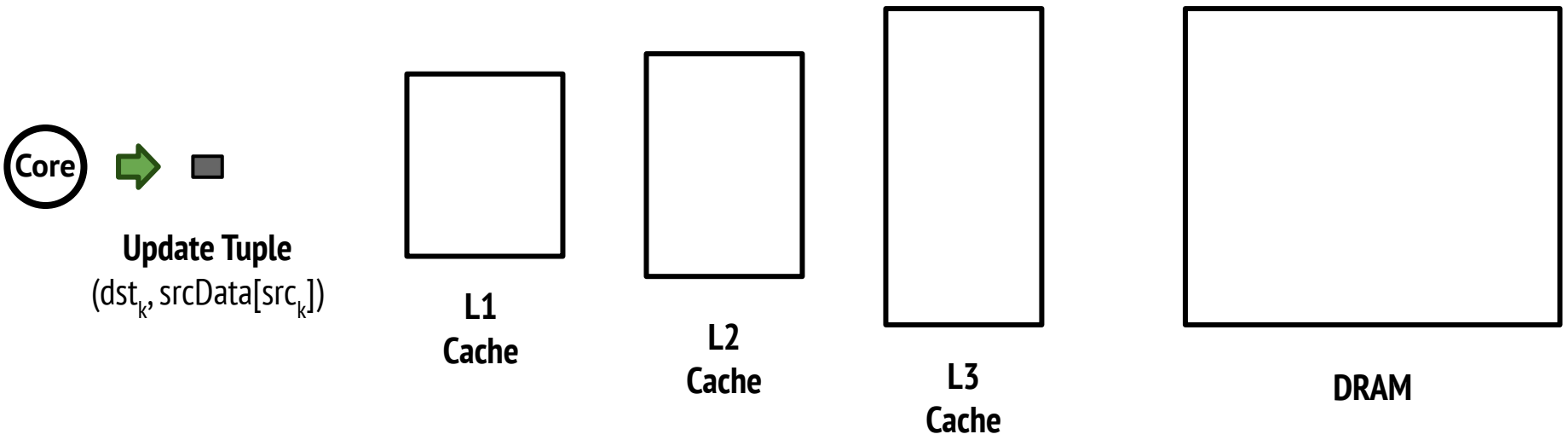
```
for src in G:
  for dst in out_neighs(src):
    updVal = srcData[src]
    binID  = dst / BinRange
    bins[binID].append(dst, updVal)
```

Coalescing Buffers
(C-Buffers)

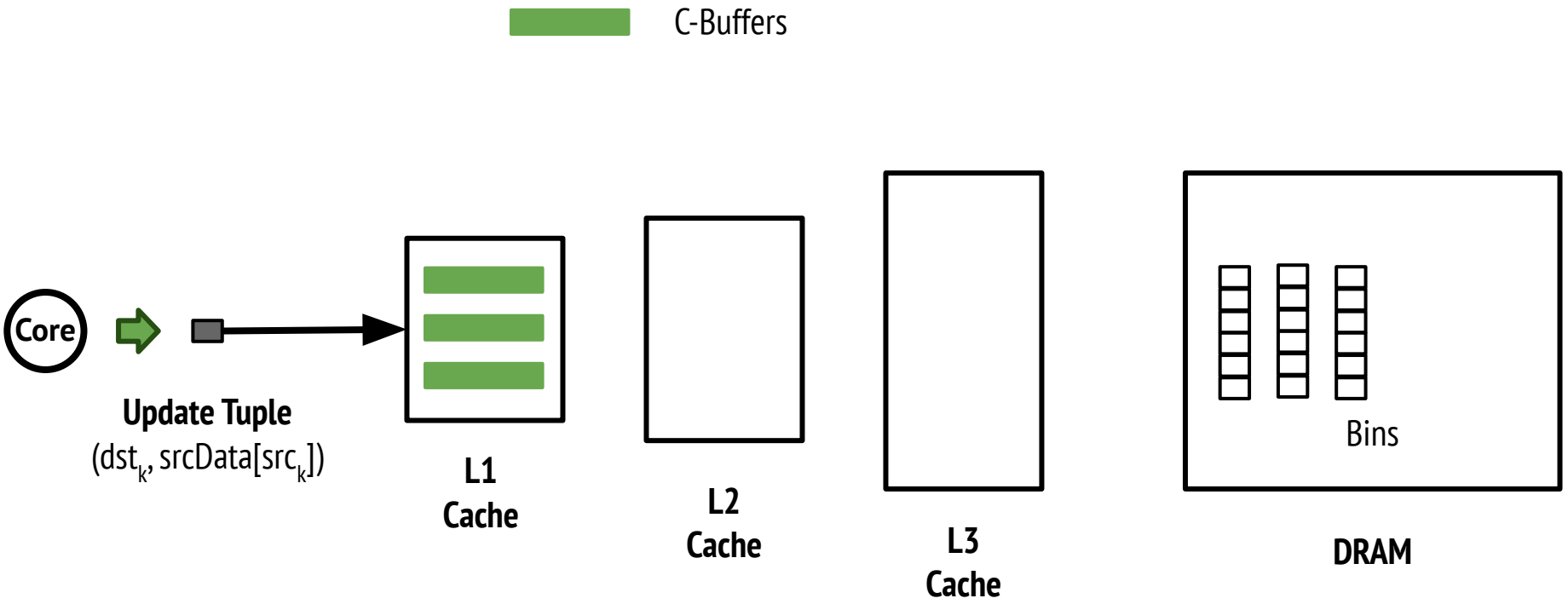
Cacheline-sized containers

1 C-Buffer / Bin

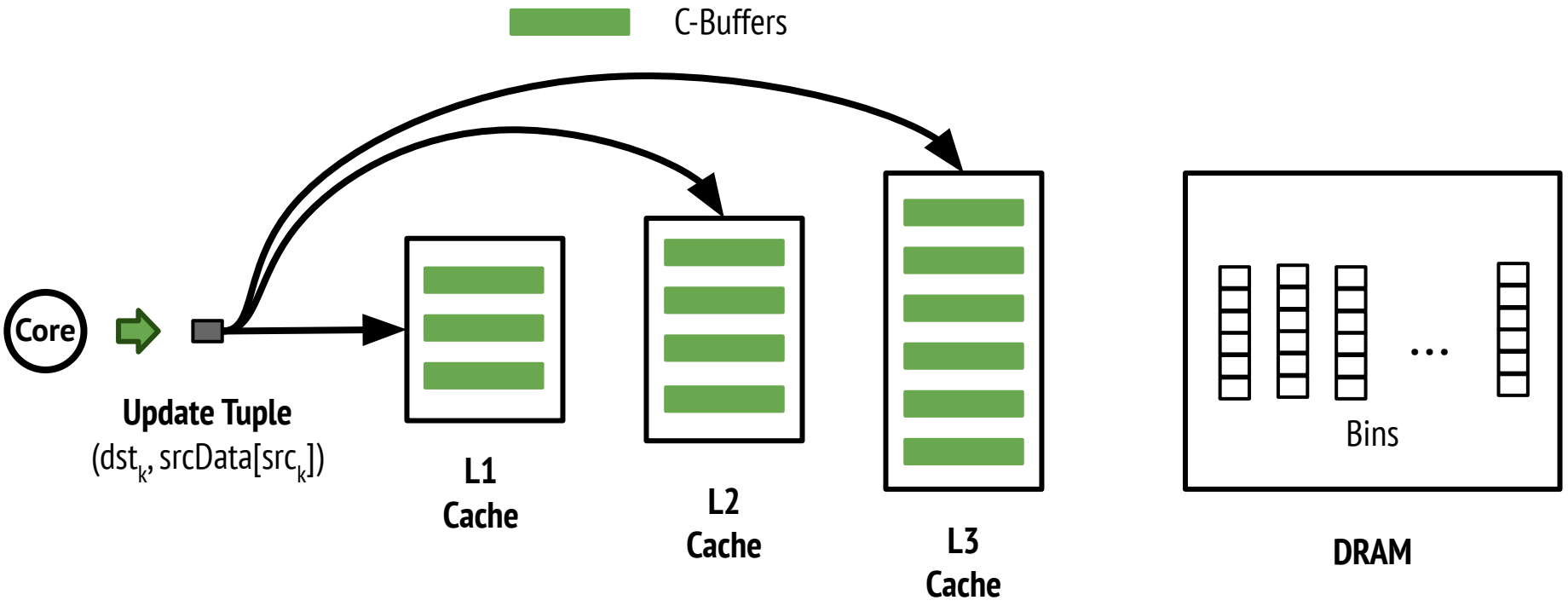
Reason For Poor Binning Performance With Many Bins



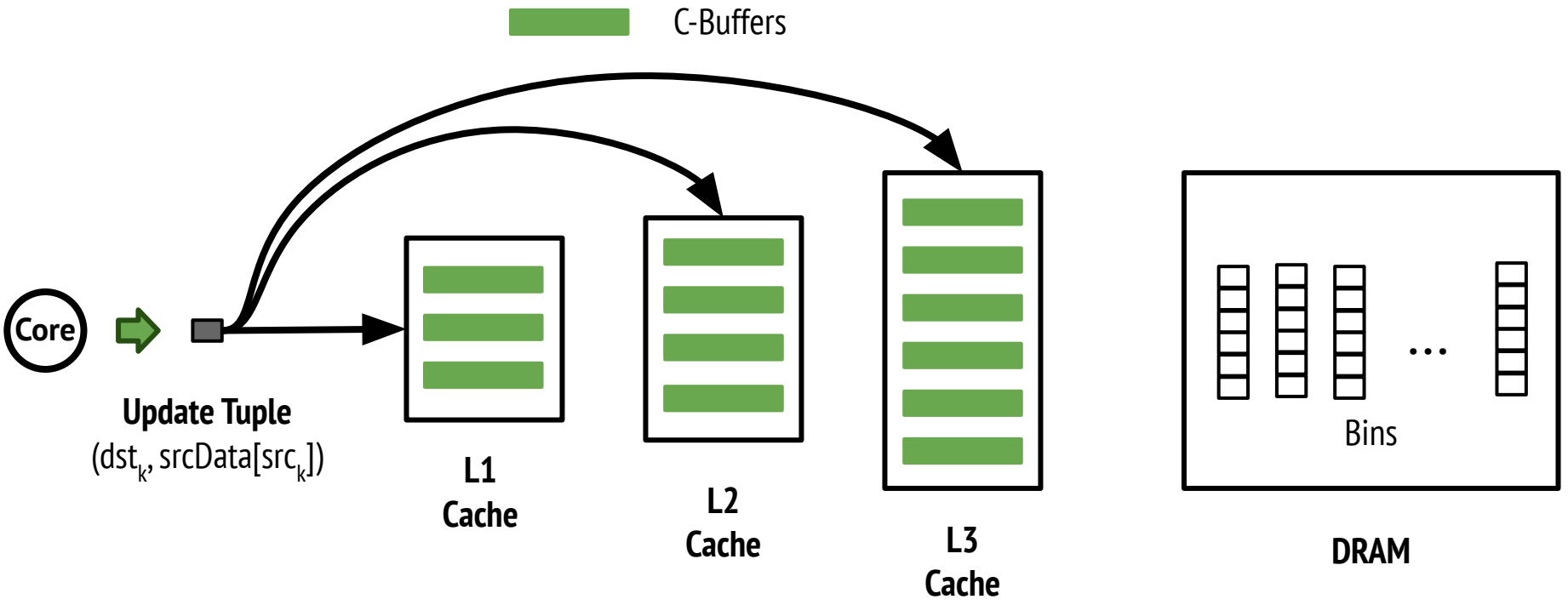
Reason For Poor Binning Performance With Many Bins



Reason For Poor Binning Performance With Many Bins



Reason For Poor Binning Performance With Many Bins



|Bins| ⬆️ ⇒ |C-Buffers| ⬆️ ⇒ Binning Latency ⬆️

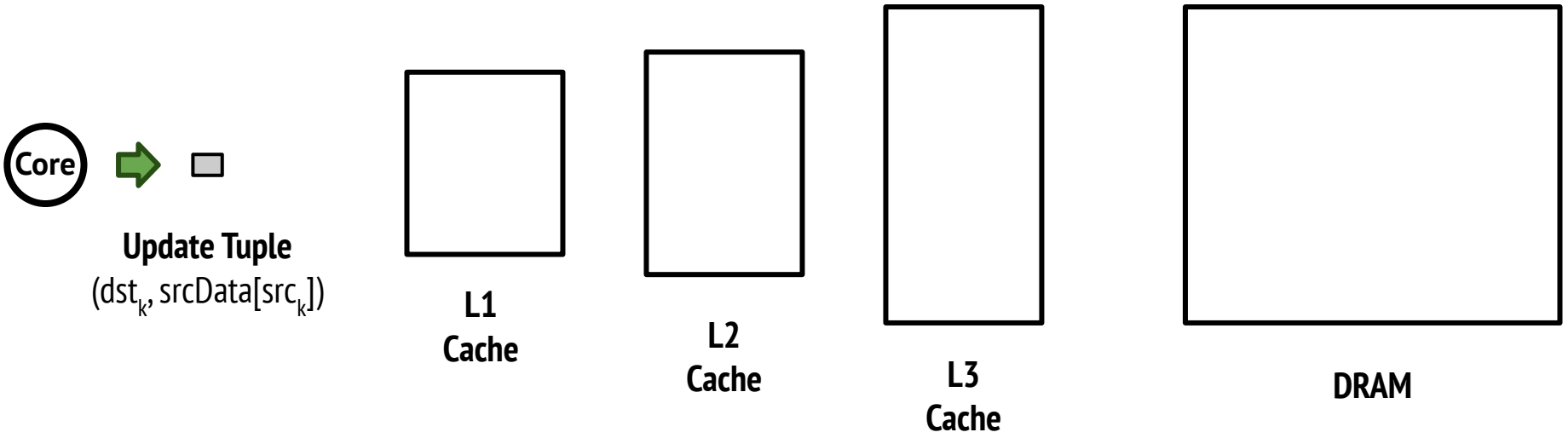
Outline

- Poor Cache Locality Due To Irregular Updates ✓
- Propagation Blocking Improves Locality ✓
- **Limitations of Propagation Blocking ✓**
- COBRA: Architecture Support For Optimizing Propagation Blocking

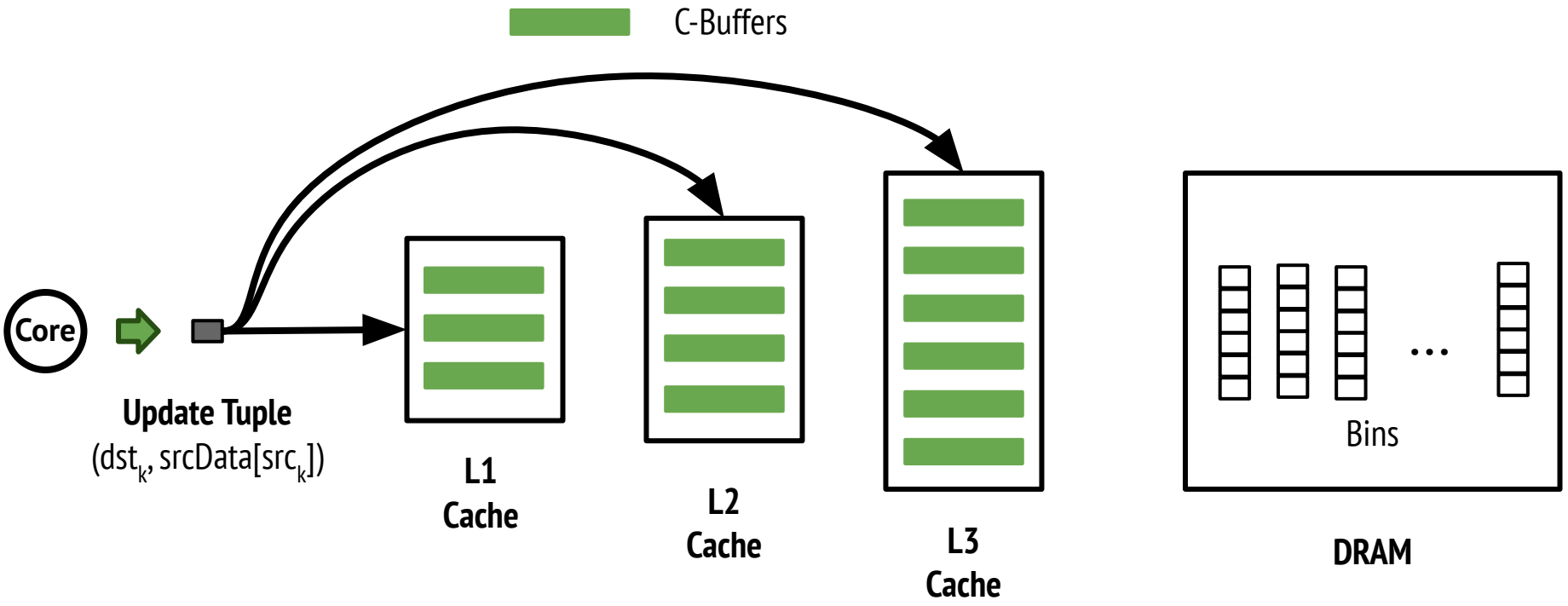
Outline

- Poor Cache Locality Due To Irregular Updates ✓
- Propagation Blocking Improves Locality ✓
- Limitations of Propagation Blocking ✓
- **COBRA: Architecture Support For Optimizing Propagation Blocking** ⬅

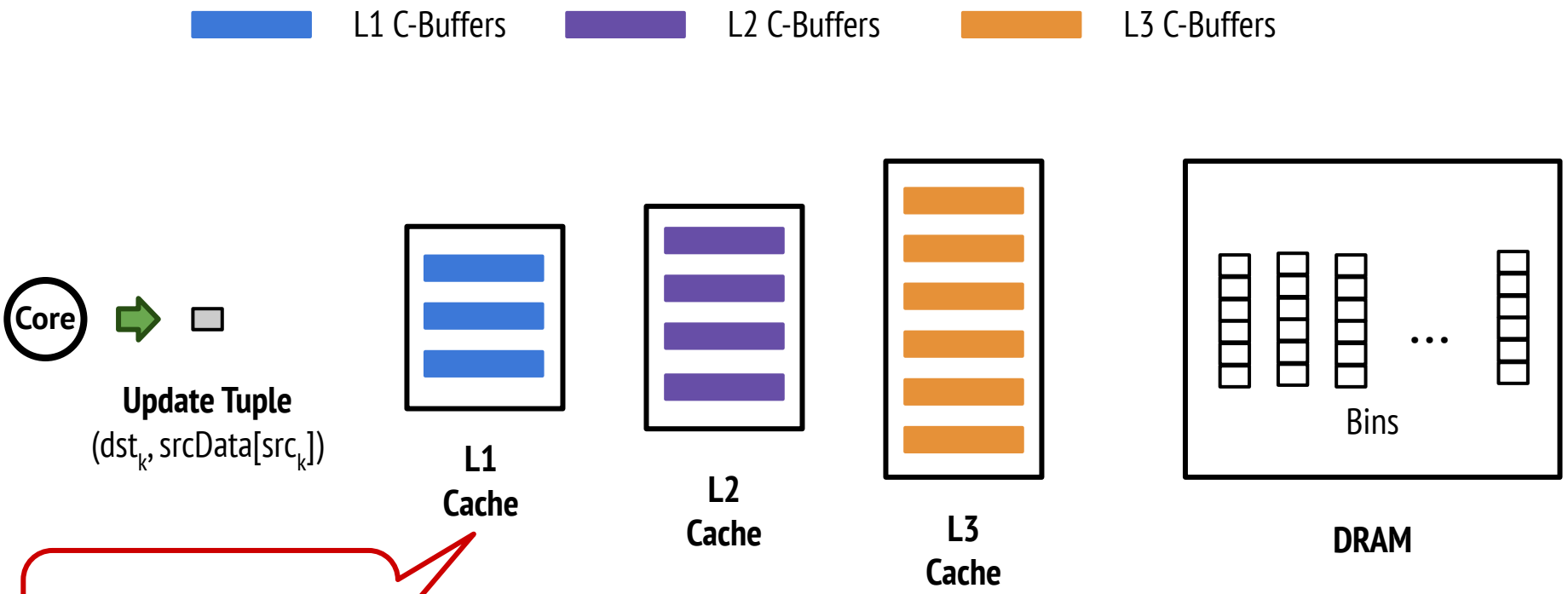
COBRA Insight: Decouple Binning Perf From Bins In Memory



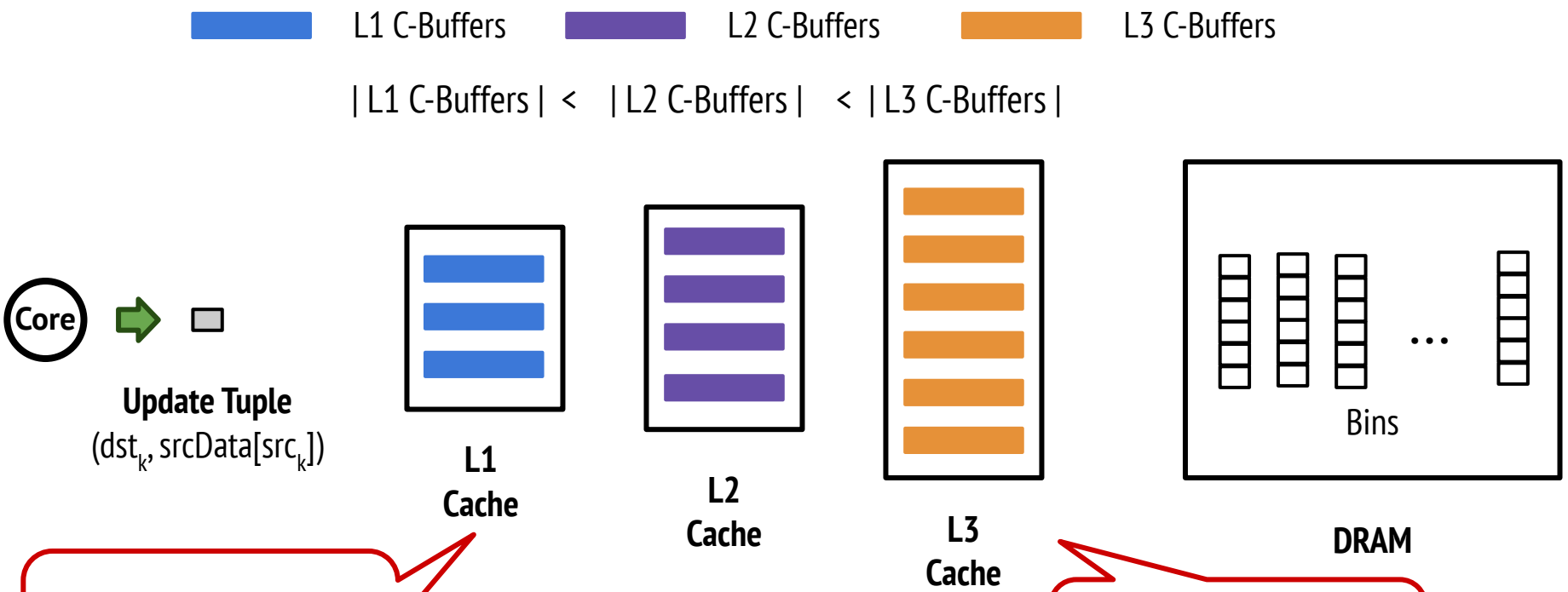
COBRA Insight: Decouple Binning Perf From Bins In Memory



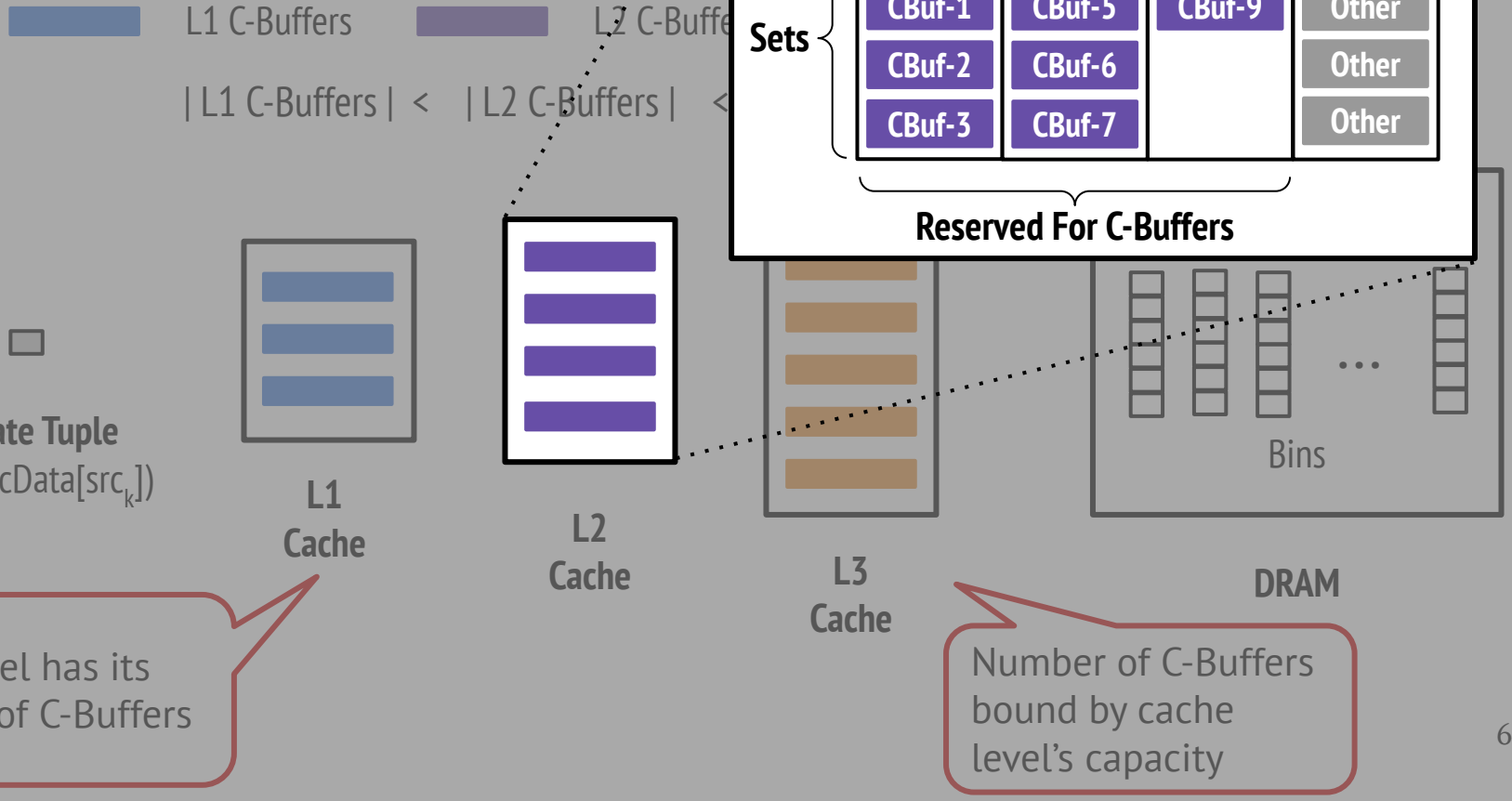
COBRA Insight: Decouple Binning Perf From Bins In Memory



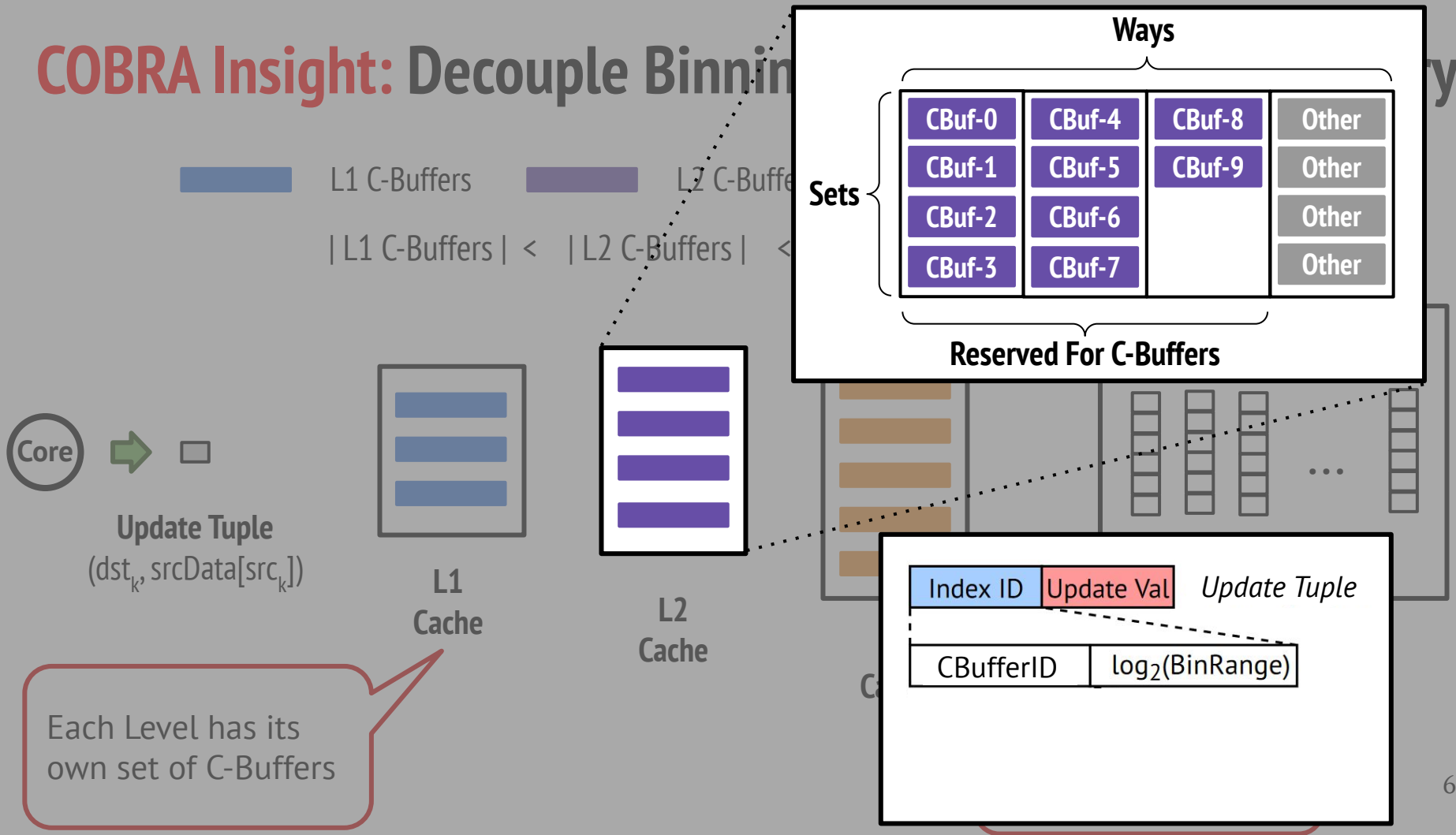
COBRA Insight: Decouple Binning Perf From Bins In Memory



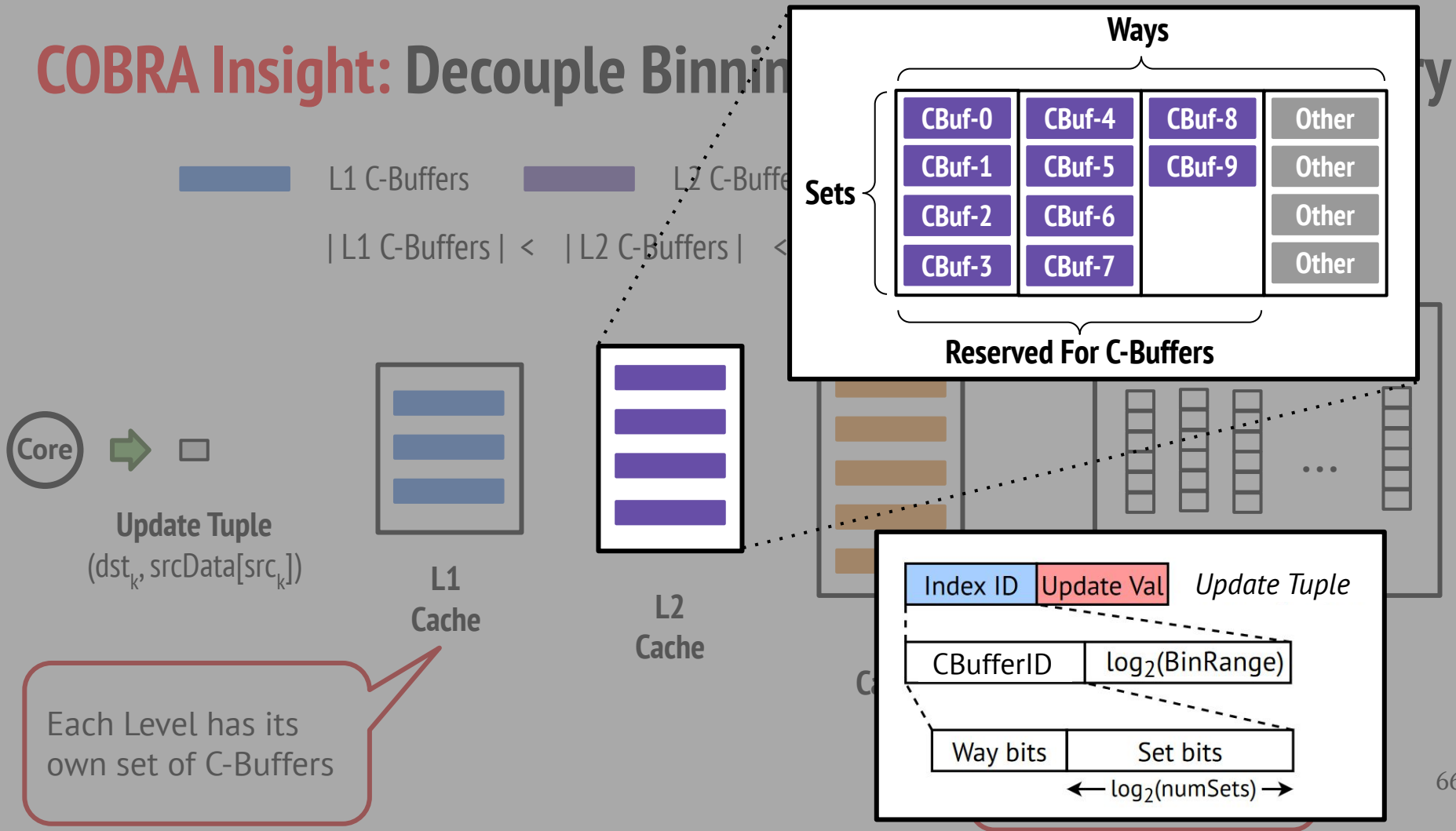
COBRA Insight: Decouple Binnin



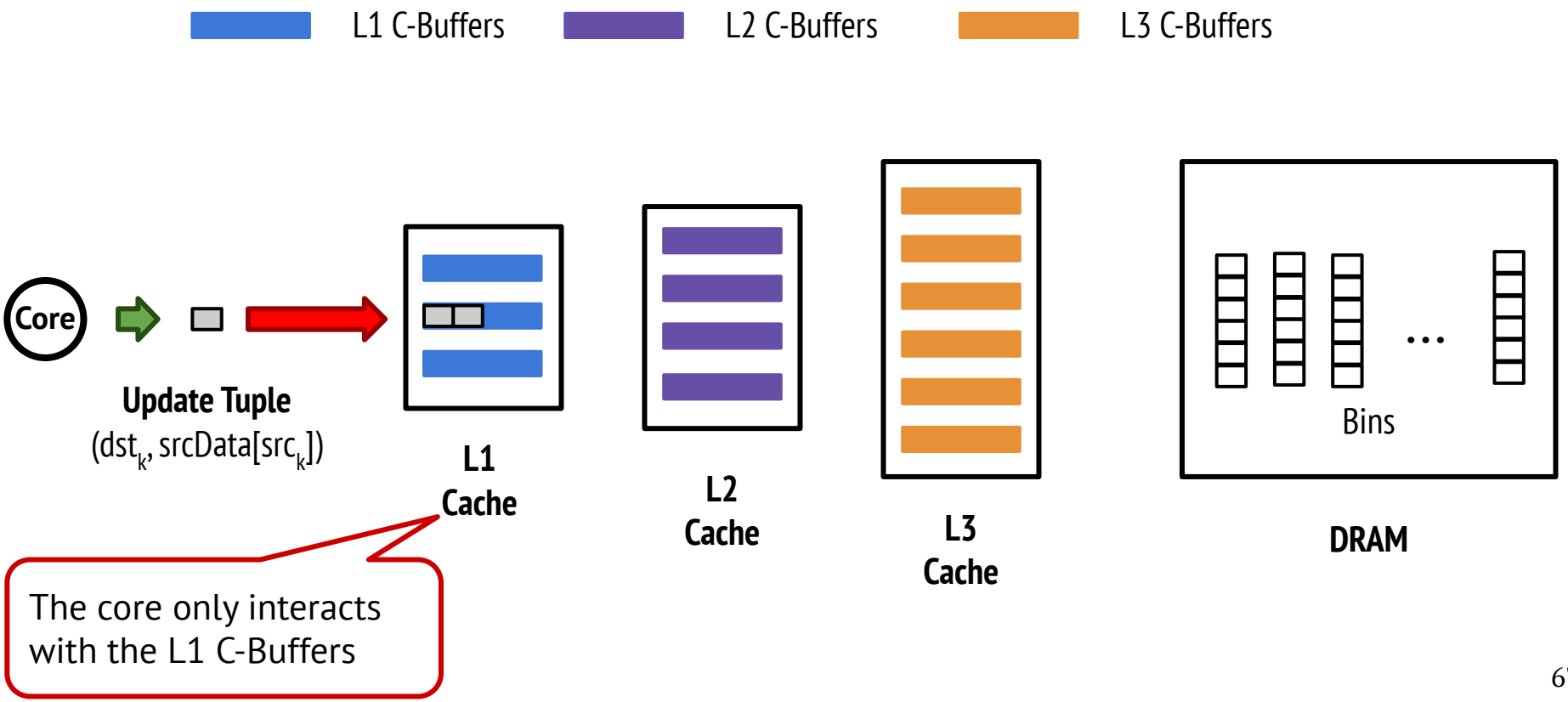
COBRA Insight: Decouple Binnin



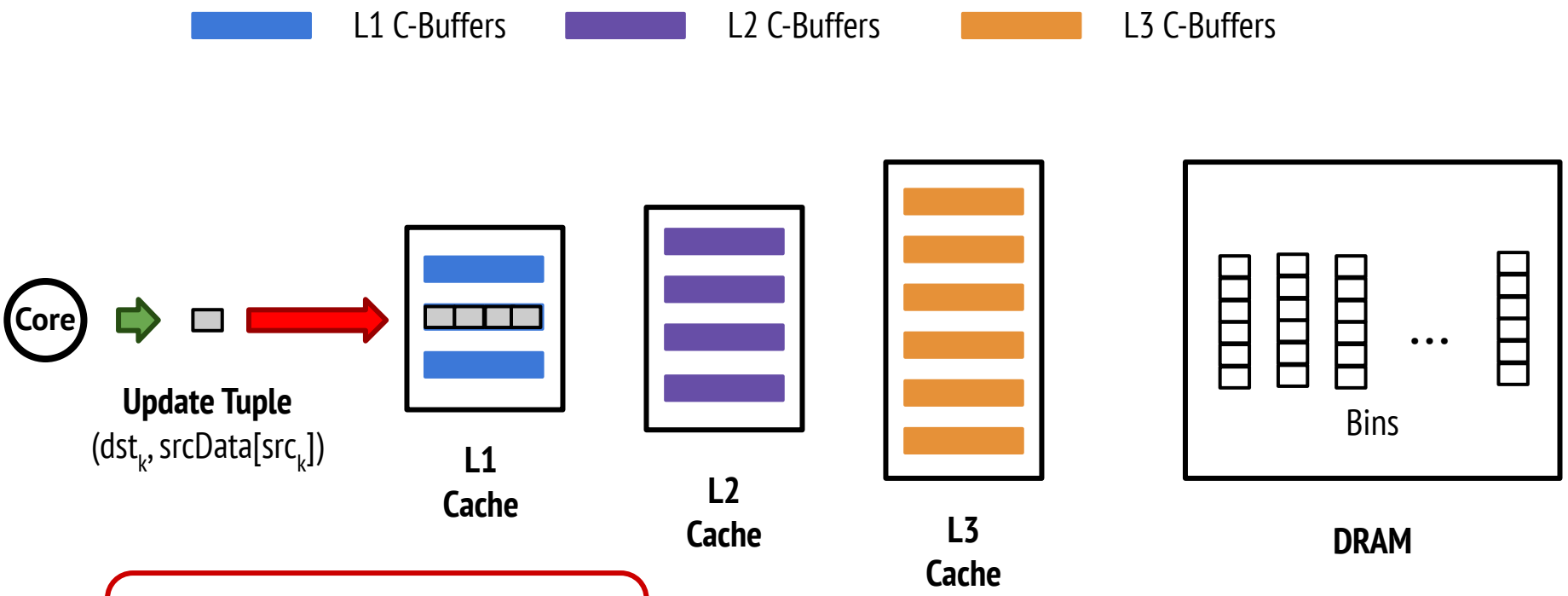
COBRA Insight: Decouple Binnin



COBRA Insight: Decouple Binning Perf From Bins In Memory

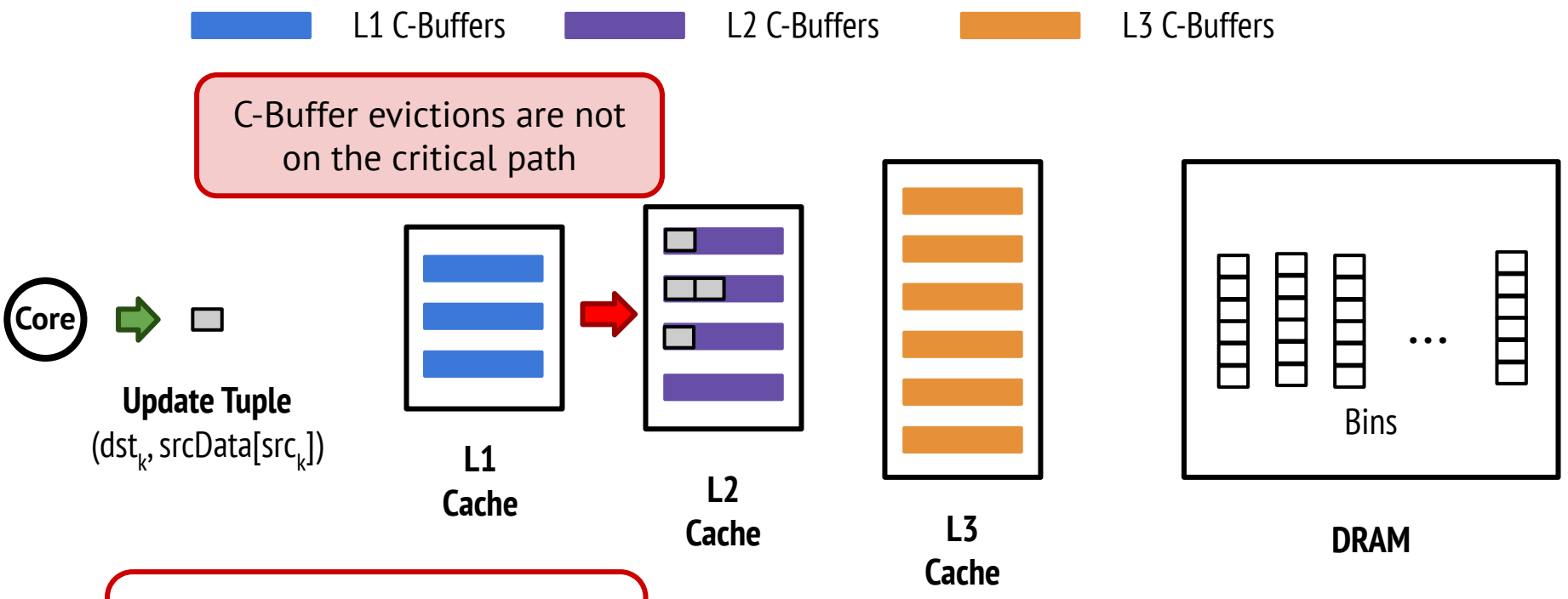


COBRA Insight: Decouple Binning Perf From Bins In Memory

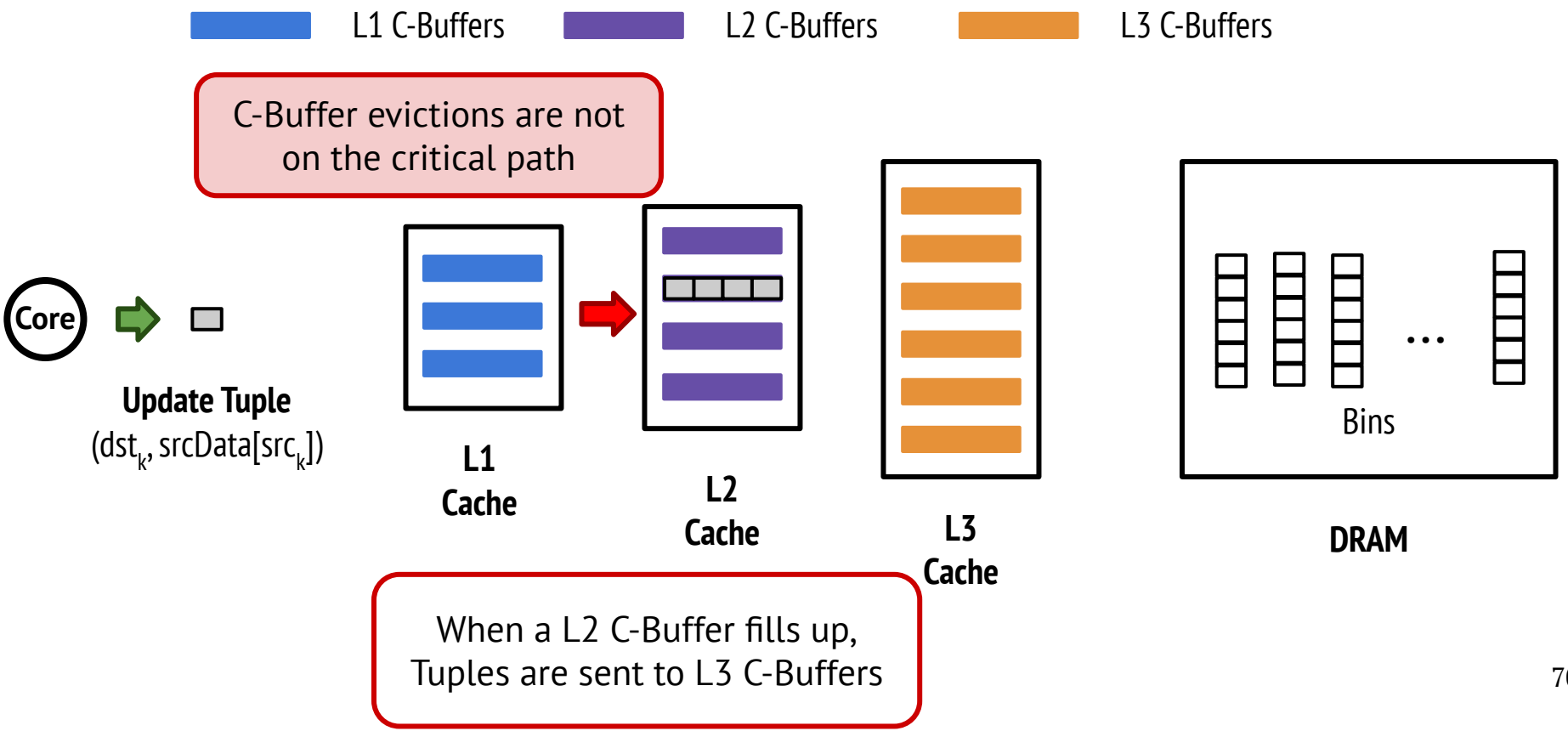


When a L1 C-Buffer fills up,
Tuples are sent to L2 C-Buffers

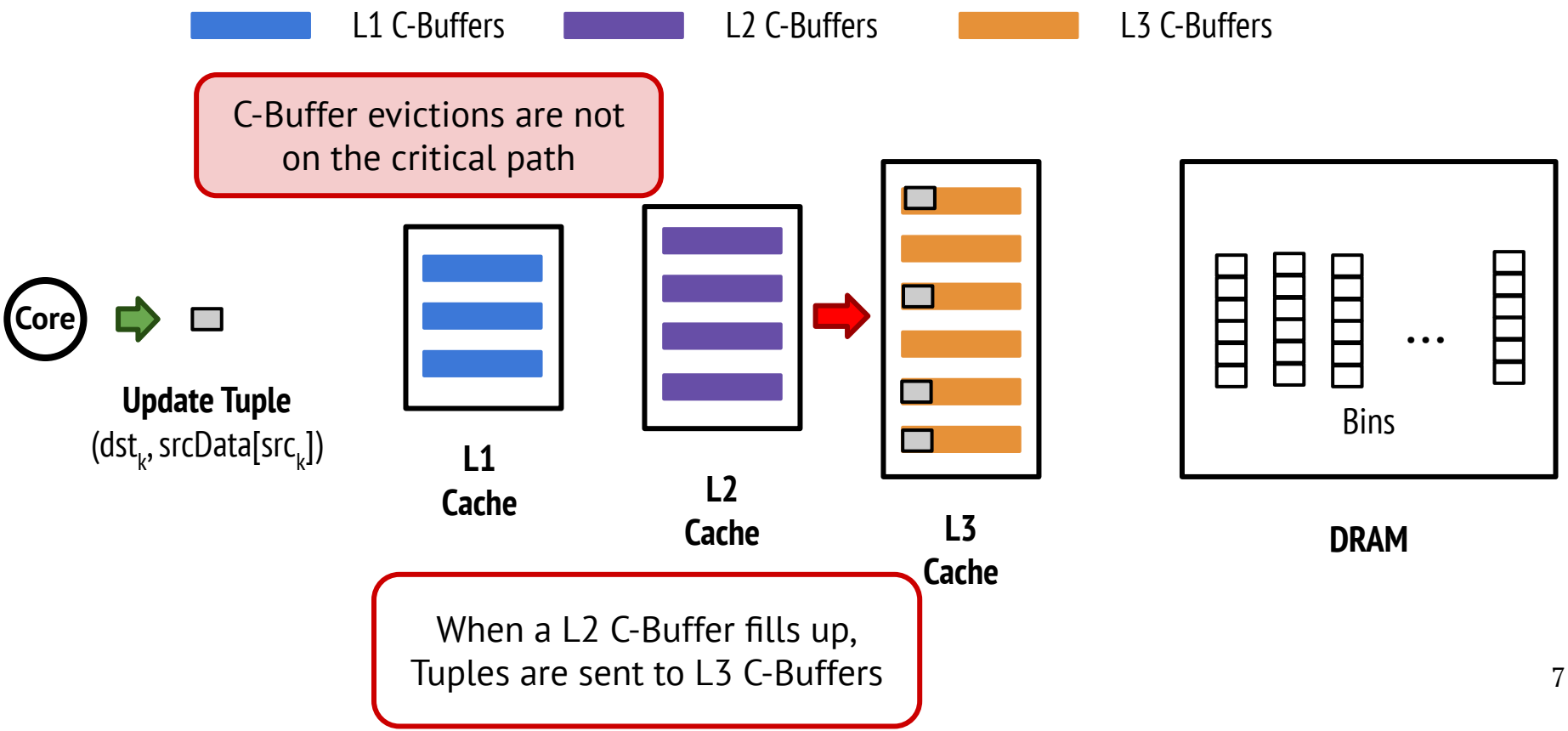
COBRA Insight: Decouple Binning Perf From Bins In Memory



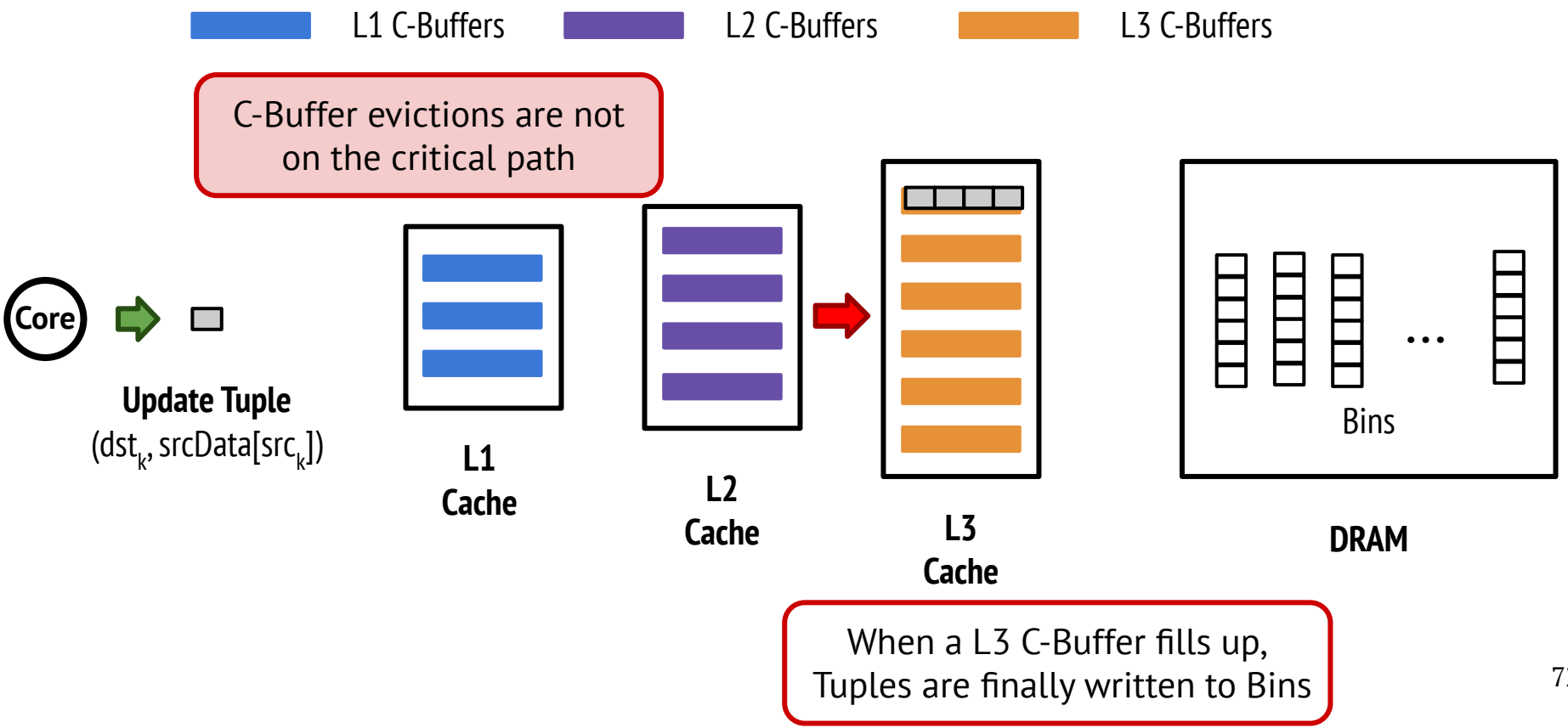
COBRA Insight: Decouple Binning Perf From Bins In Memory



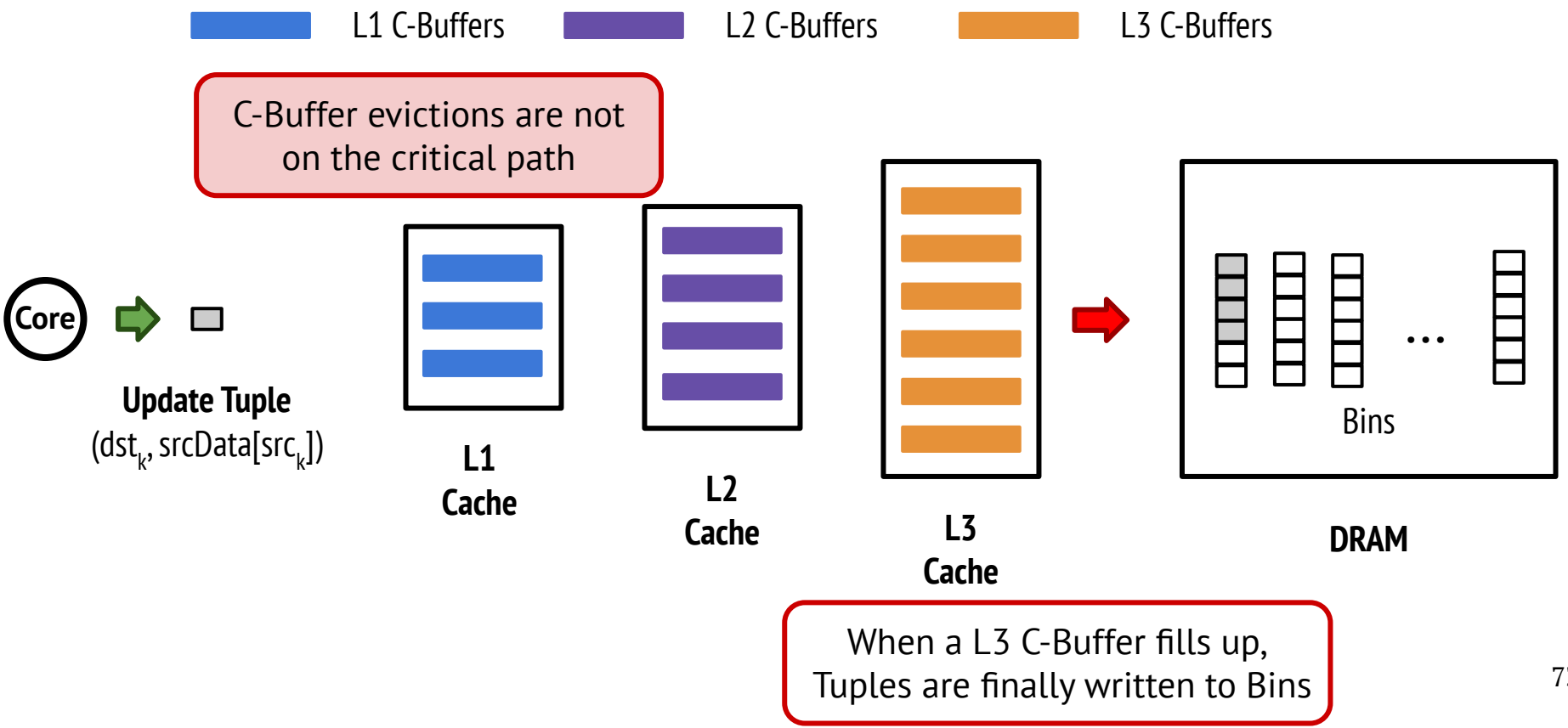
COBRA Insight: Decouple Binning Perf From Bins In Memory



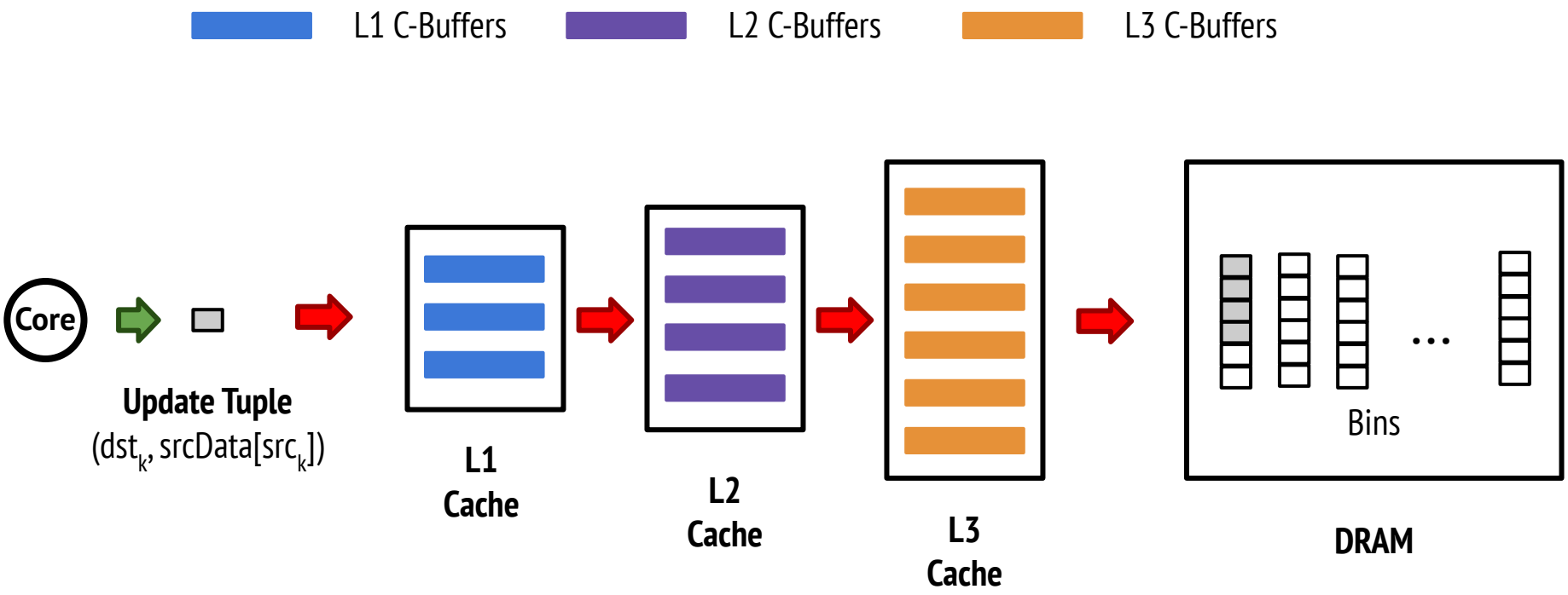
COBRA Insight: Decouple Binning Perf From Bins In Memory



COBRA Insight: Decouple Binning Perf From Bins In Memory

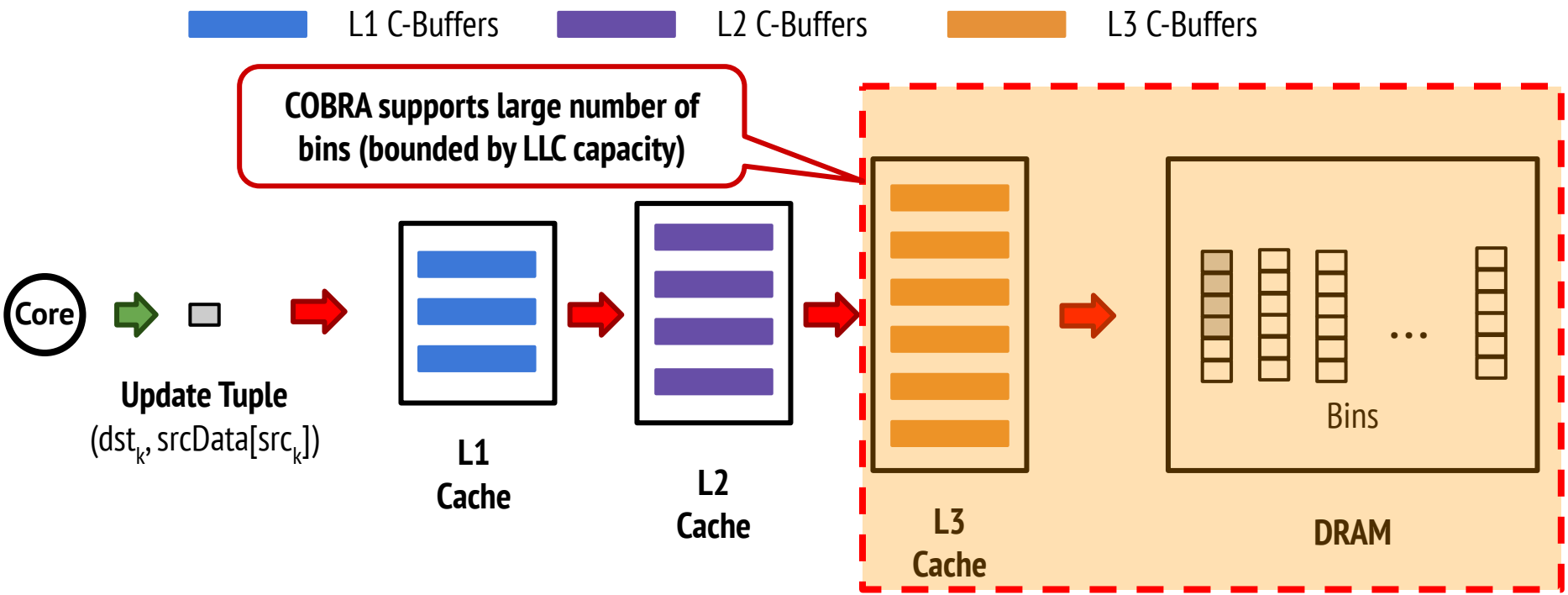


COBRA Insight: Decouple Binning Perf From Bins In Memory



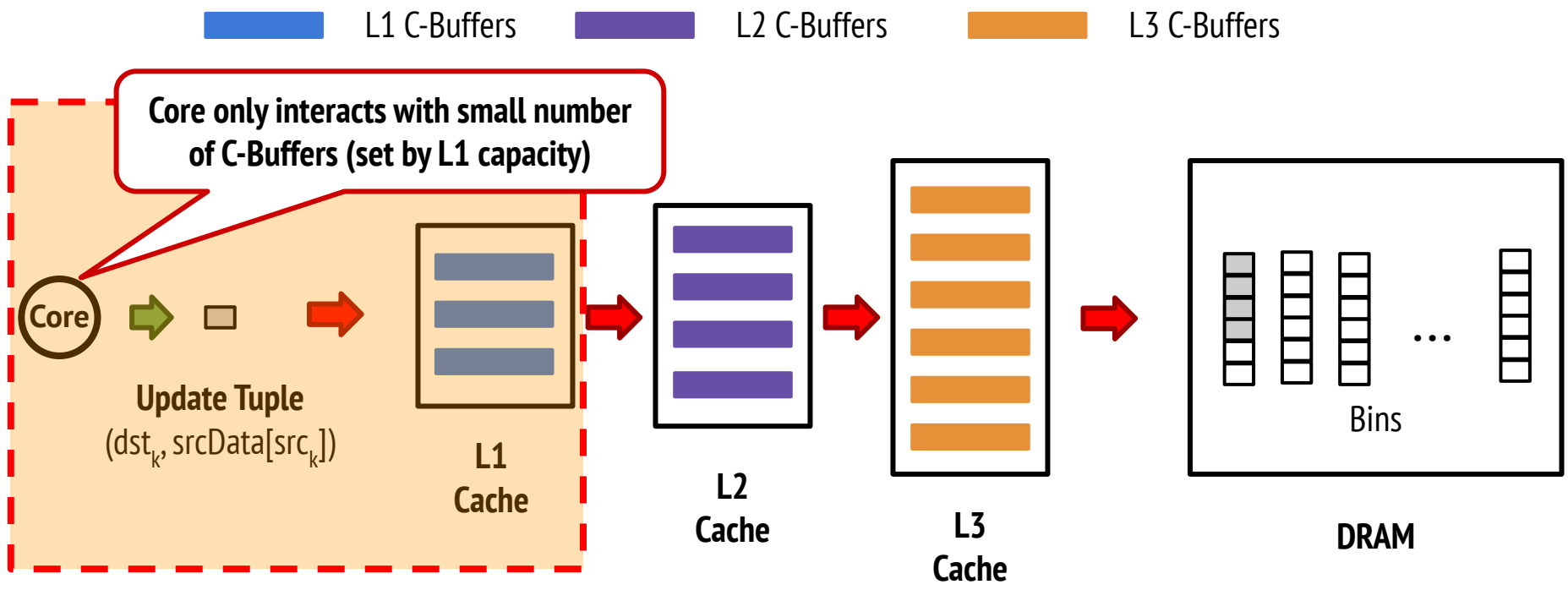
Core ⇒ L1 C-Buffer ⇒ L2 C-Buffer ⇒ L3 C-Buffer ⇒ (In-memory) Bins

COBRA Insight: Decouple Binning Perf From Bins In Memory



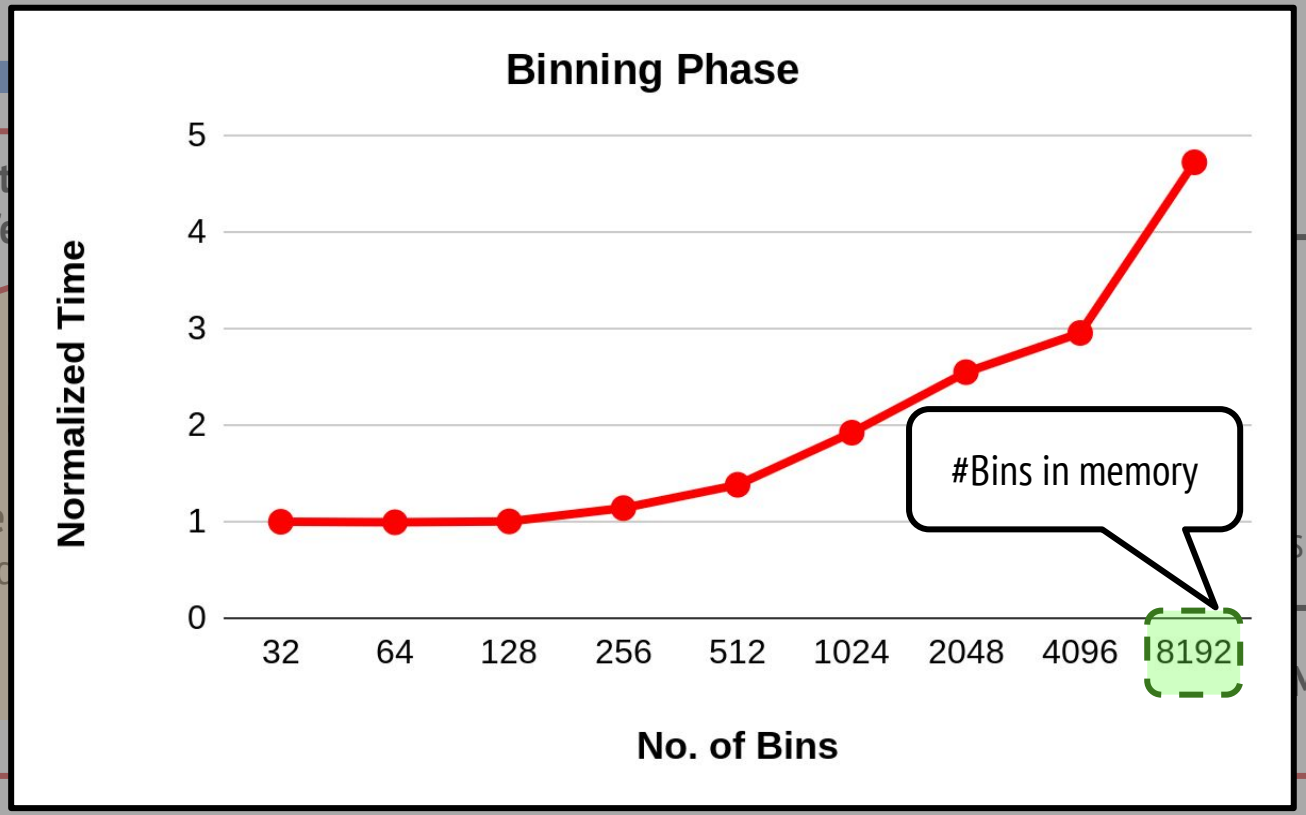
Core ⇒ L1 C-Buffer ⇒ L2 C-Buffer ⇒ **L3 C-Buffer ⇒ (In-memory) Bins**

COBRA Insight: Decouple Binning Perf From Bins In Memory



Core ⇒ L1 C-Buffer ⇒ L2 C-Buffer ⇒ L3 C-Buffer ⇒ (In-memory) Bins

COBRA Insight: Decouple Binning Perf From Bins In Memory



Core only int
of C-Buffe

Core

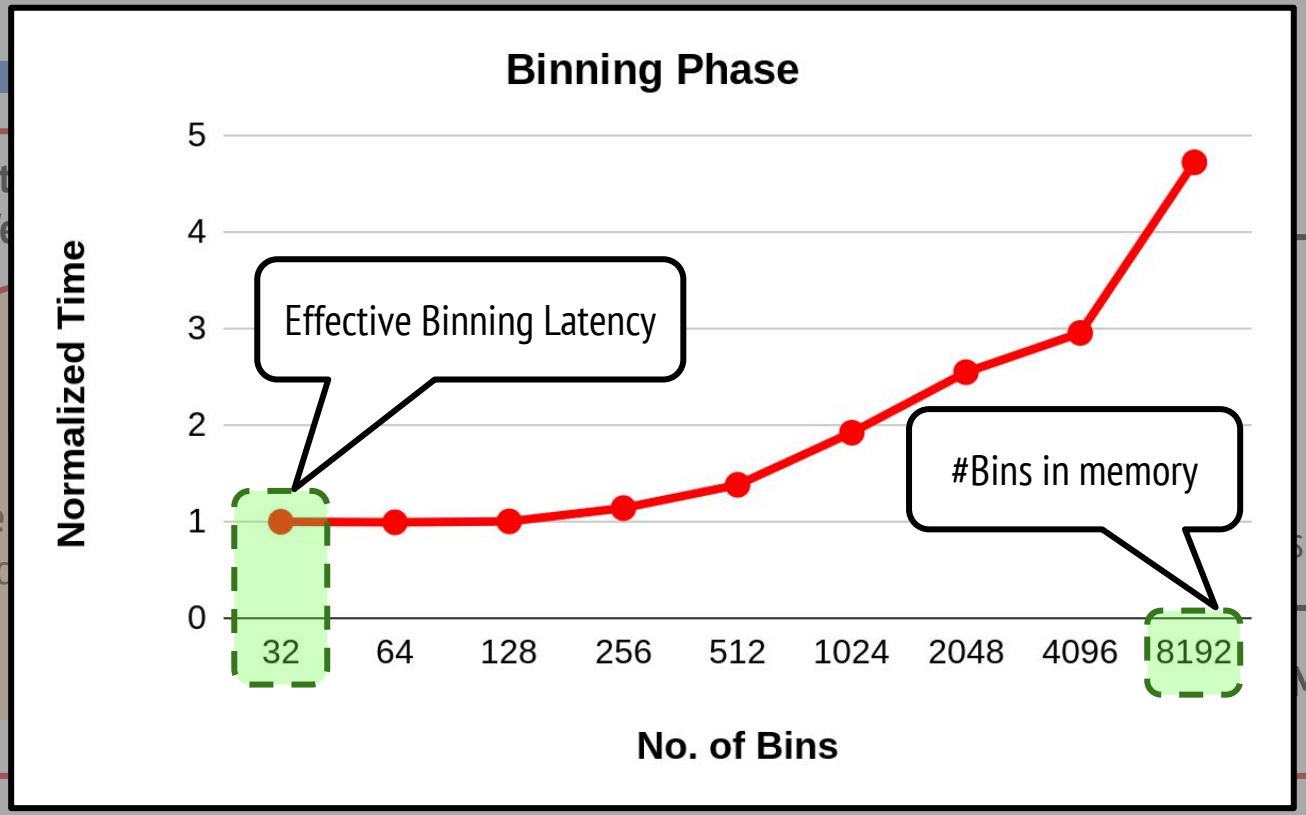
Update Tuple
(dst$_k$, srcData[src

...

M

Core ⇒ L1 C-Buffer ⇒ L2 C-Buffer ⇒ L3 C-Buffer ⇒ (In-memory) Bins

COBRA Insight: Decouple Binning Perf From Bins In Memory



Core only int
of C-Buffe

Core

Update Tuple
(dst$_k$, srcData[src

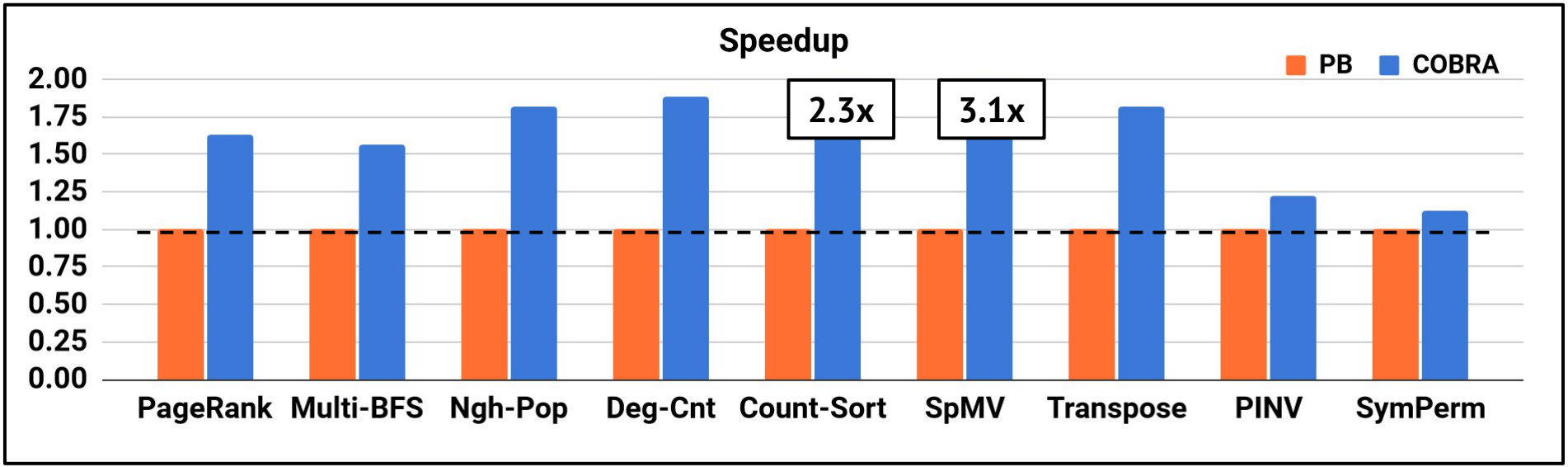
...

M

Core ⇒ L1 C-Buffer ⇒ L2 C-Buffer ⇒ L3 C-Buffer ⇒ (In-memory) Bins

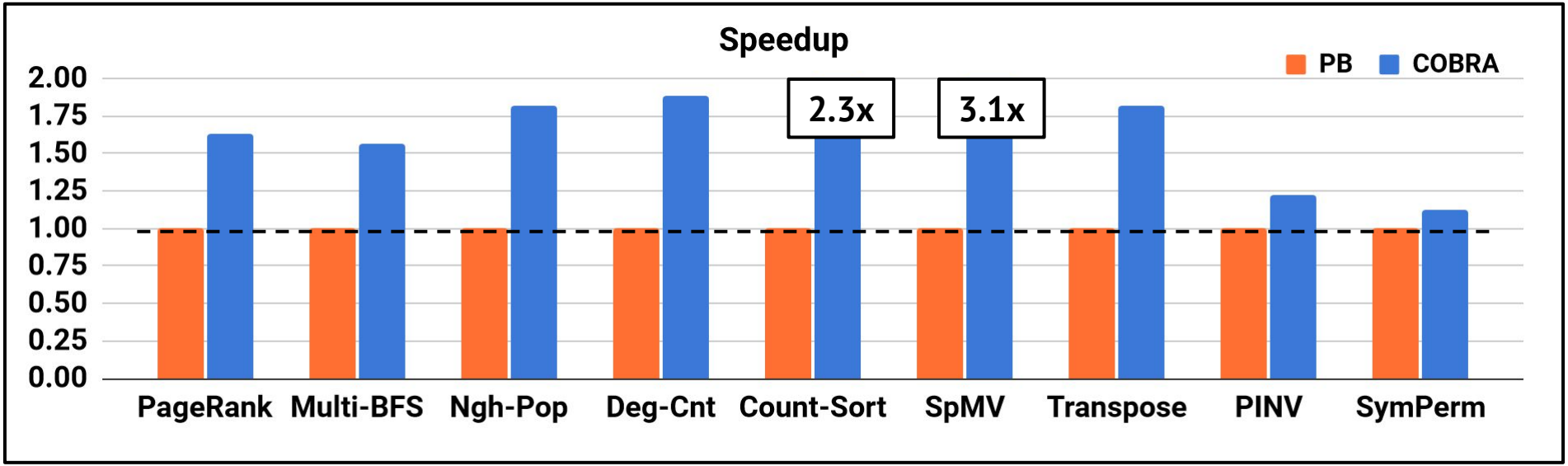
COBRA Improves Propagation Blocking Performance

Higher is Better



COBRA Improves Propagation Blocking Performance

Higher is Better



COBRA eliminates PB's inefficiencies to provide a mean speedup of 1.75x

More Details In The Paper

- COBRA does not require update commutativity
  - Unlike PHI, COBRA applies to workloads with non-commutative updates
  - For commutative updates, COBRA-COMM improves upon PHI

More Details In The Paper

- COBRA does not require update commutativity
  - Unlike PHI, COBRA applies to workloads with non-commutative updates
  - For commutative updates, COBRA-COMM improves upon PHI
- COBRA improves ILP of the Binning phase

More Details In The Paper

- COBRA does not require update commutativity
  - Unlike PHI, COBRA applies to workloads with non-commutative updates
  - For commutative updates, COBRA-COMM improves upon PHI
- COBRA improves ILP of the Binning phase
- COBRA's performance sensitivity studies:
  - No. of ways reserved for C-Buffers
  - Eviction Buffer sizes
  - OS Scheduling quantum
- Comparison of PB vs CAGRA (SOTA 1D Graph Tiling)

Summary

- Propagation Blocking improves performance of a broad range of applications
- Propagation Blocking is unable to select the optimal number of bins
- COBRA improves Propagation Blocking performance by:
  - Improving Binning performance when there are a large number of bins
  - Reducing the instruction overhead of managing C-Buffers in Software

Improving Locality of Irregular Updates with Hardware Assisted Propagation Blocking

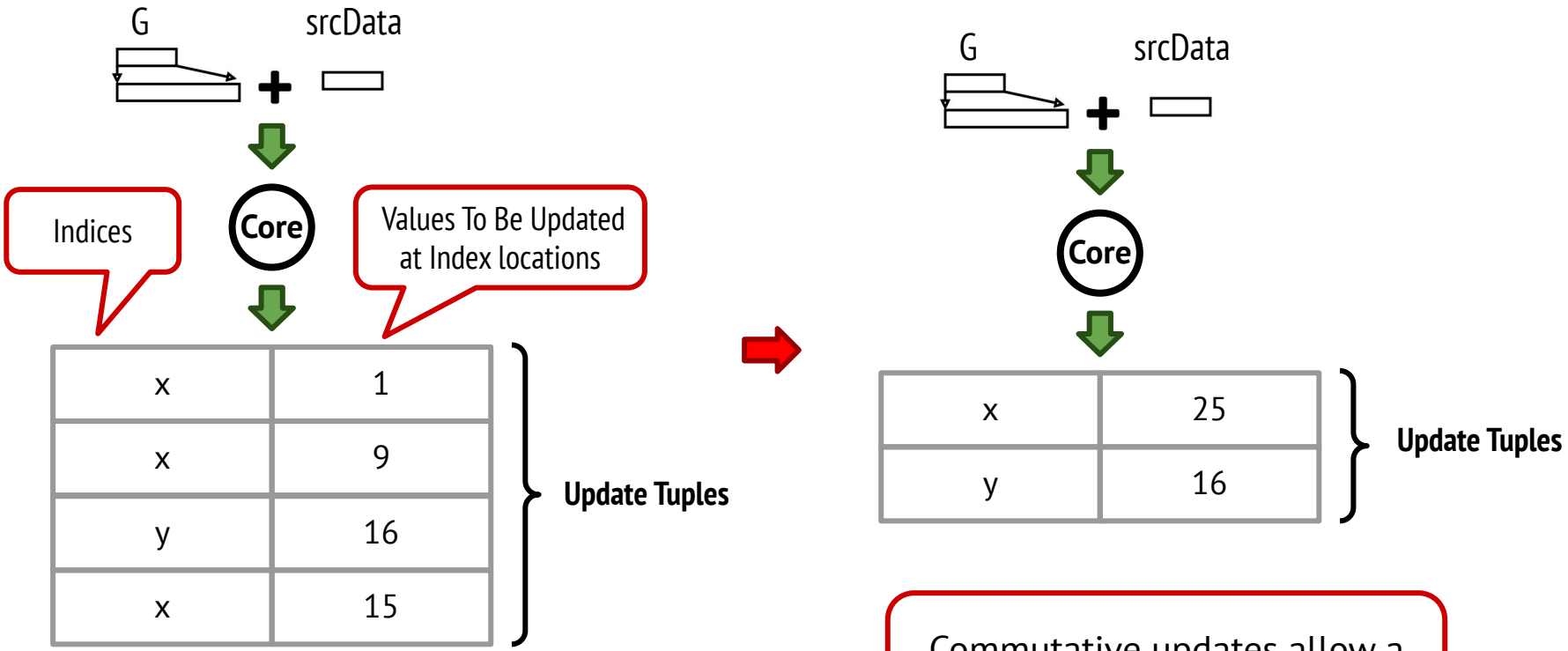
Vignesh Balaji Brandon Lucia

NVIDIA

Carnegie Mellon University

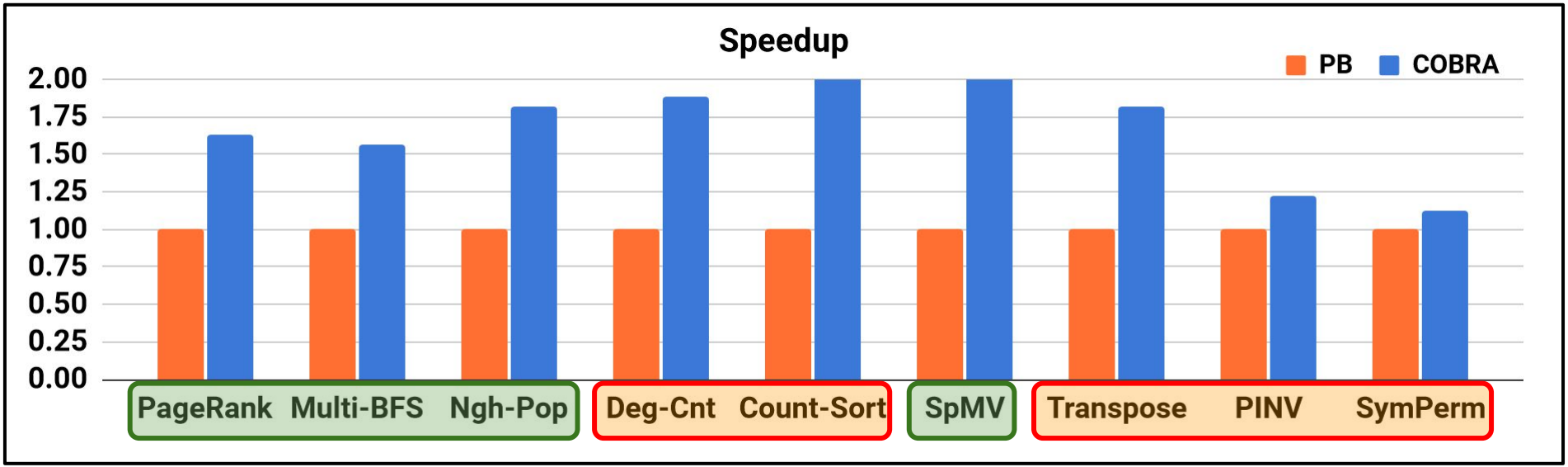
Backup Slides

COBRA Does Not Require Update Commutativity



COBRA Does Not Require Update Commutativity

Higher is Better



Example of a Non-Commutative Workload

Algorithm 1 Kernel to populate neighbors (Edgelist-to-CSR)

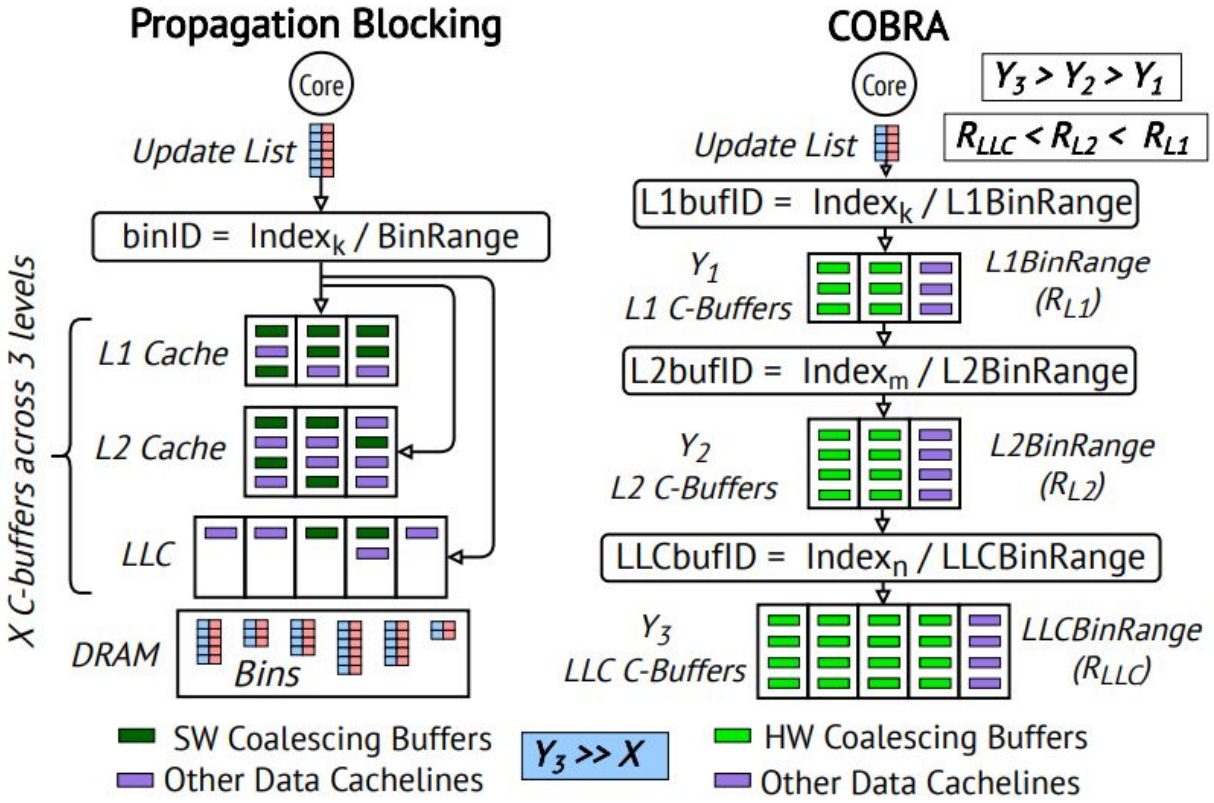
```
1: offsets ← PrefixSum(degrees)        ▷ OA in Figure 1
2: par_for e in EL do
3:     neighs[offsets[e.src]] ← e.dst   ▷ NA
4:     AtomicAdd(offsets[e.src], 1)
```

PB Execution Breakup

APPS →	*PB Phases* ↓	DC	NP	PR	RD	IS	SPMV	PINV	TR	SP
Medium No. of Bins	**Init**	9.91%	5.68%	18.43%	23.75%	5.72%	0.29%	8.6%	14.63%	6.76%
	Binning	73%	54.18%	47%	51.78%	44.47%	77.09%	56.11%	48.16%	14.26%
	Accumulate	17.09%	40.14%	34.57%	24.47%	49.81%	22.61%	35.28%	37.21%	78.98%
Large No. of Bins	**Init**	7.72%	6.01%	13.17%	18.22%	6.95%	0.22%	8.88%	11.96%	6.71%
	Binning	86.15%	78.57%	65.52%	71.60%	77.94%	87.46%	64.42%	67.82%	17.1%
	Accumulate	6.01%	15.42%	21.32%	10.18%	15.12%	12.32%	26.7%	20.22%	76.19%

Table I: **PB execution breakup:** *Binning dominates a PB execution both when using a medium no. of bins (which offers the best overall PB performance) and when using a large no. of bins (which offers the best Accumulate performance).*

COBRA Overview



COBRA C-Buffer Organization

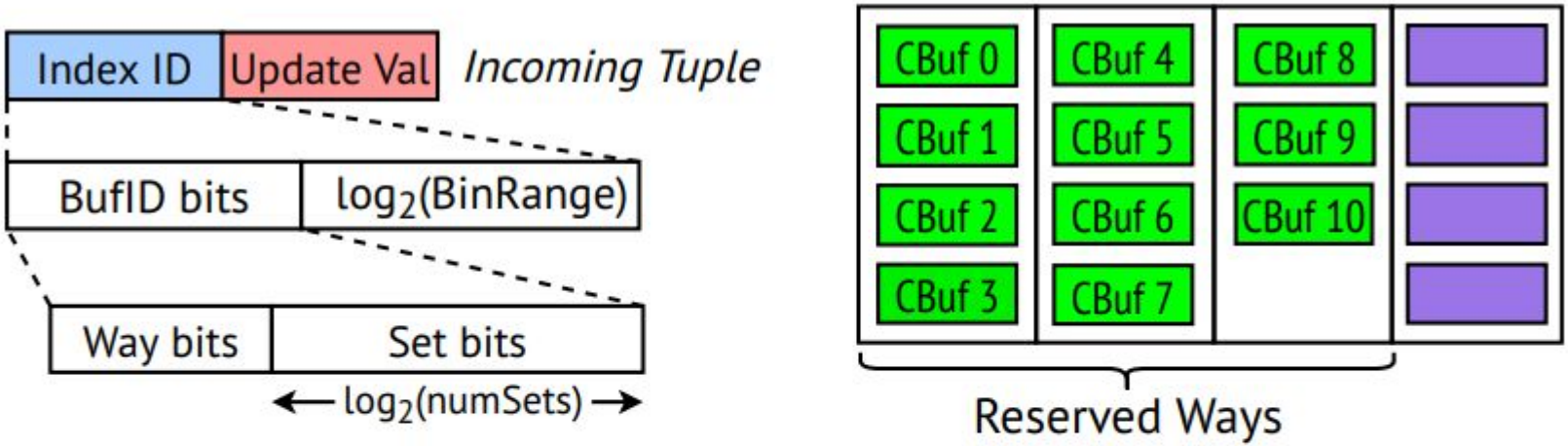


Figure 7: ***C-Buffer*** **organization within each cache level:** *Each cache level has a unique bin range that is used to map an incoming tuple into one of the C-Buffers pinned in cache.*

Handling C-Buffer Evictions

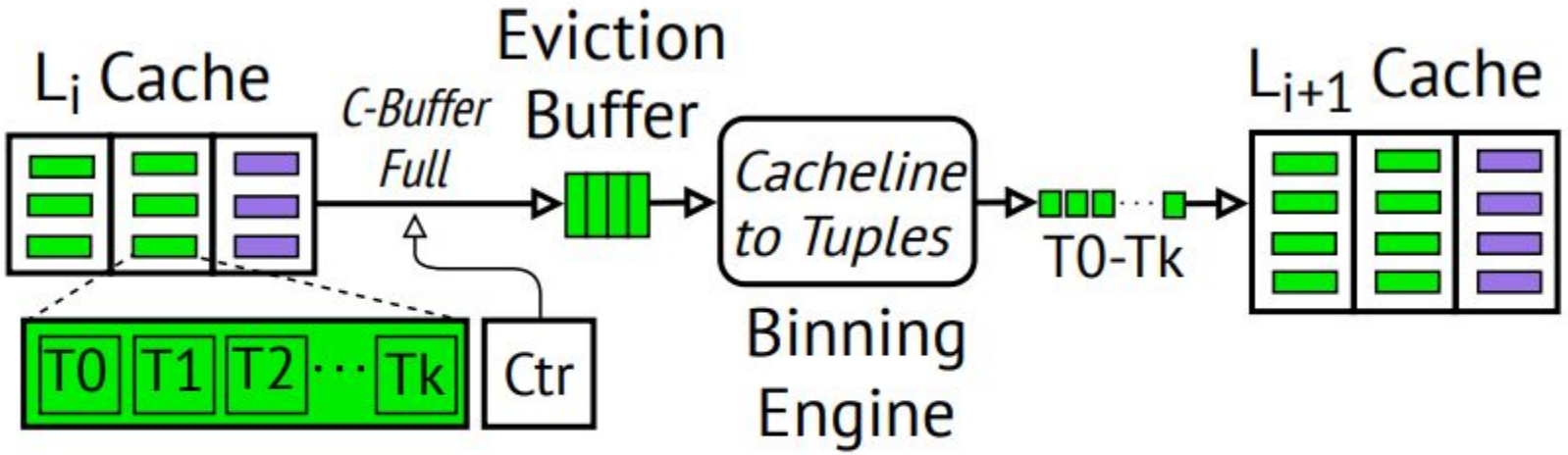


Figure 8: **Handling evictions when a *C-Buffer* fills up:** *Eviction buffers hide the latency of evicting tuples.*

C-Buffers in the LLC

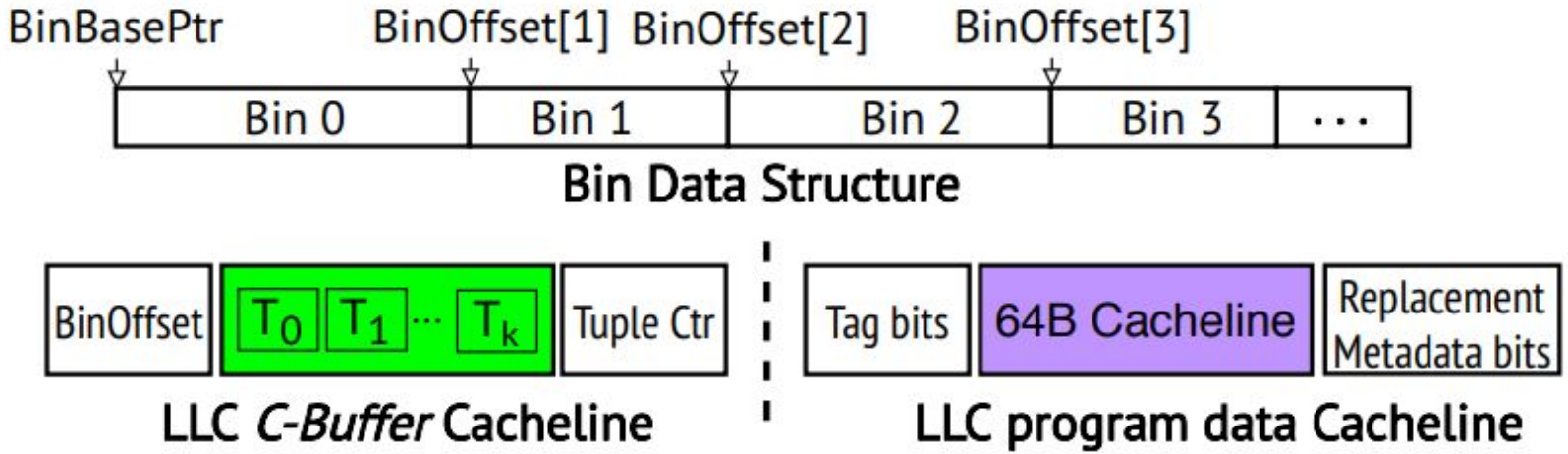


Figure 9: **Organization of per-thread bins in memory:** *BinOffsets are stored in the tag bits of cache lines containing LLC C-Buffers.*

COBRA Optimizes Both Phases of PB

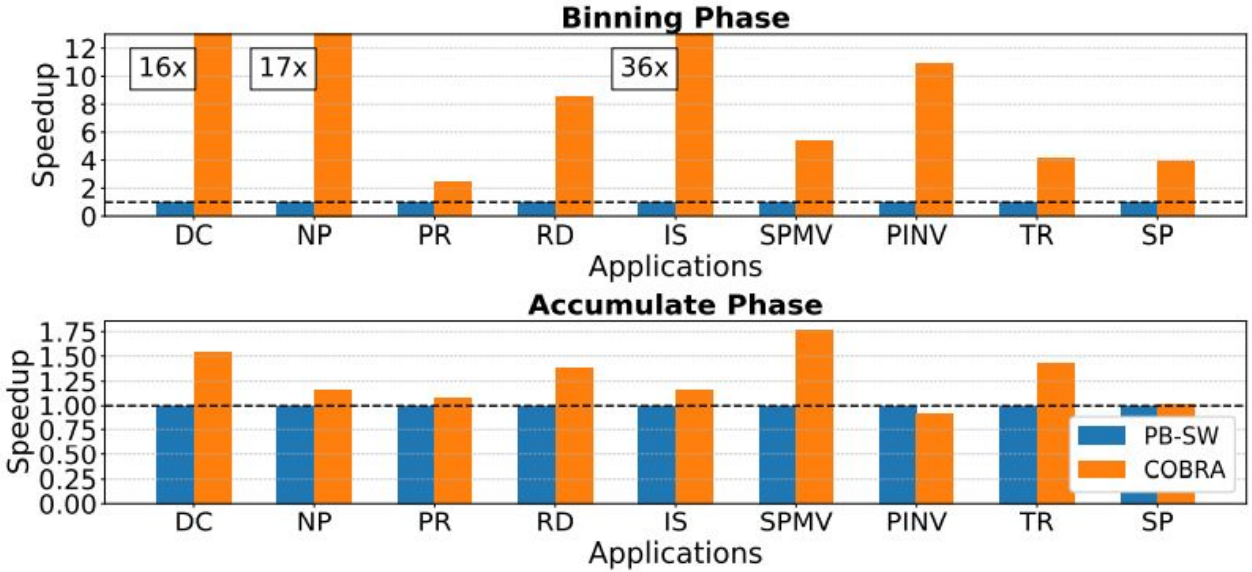


Figure 11: **COBRA speedup across both phases of PB**: *COBRA uses a large number of bins naturally optimizing Accumulate and uses architecture support to optimize Binning.*

COBRA Improves ILP of the Binning Phase

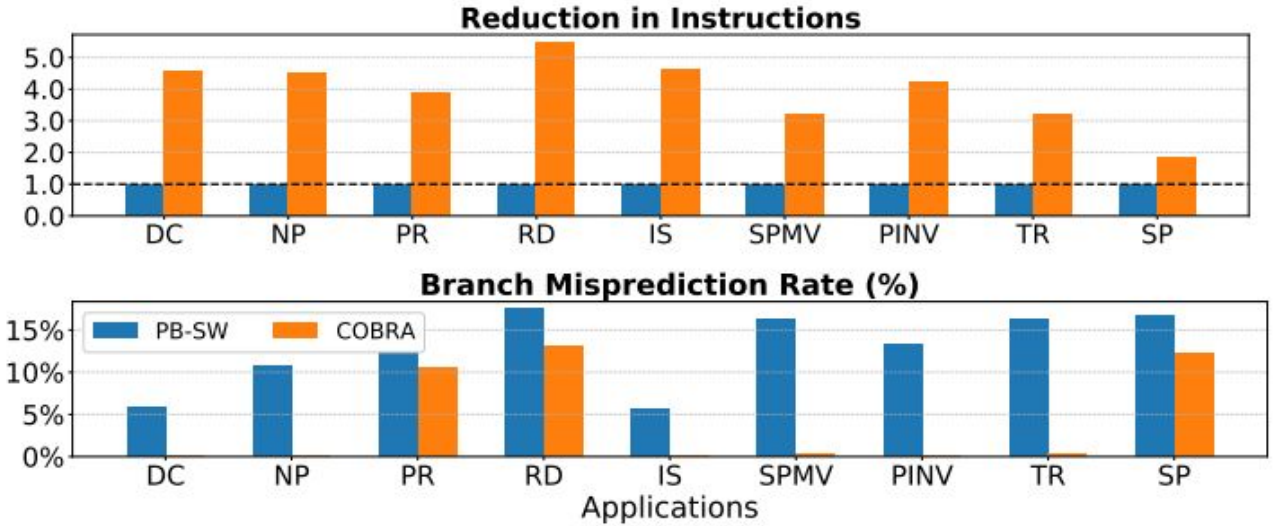
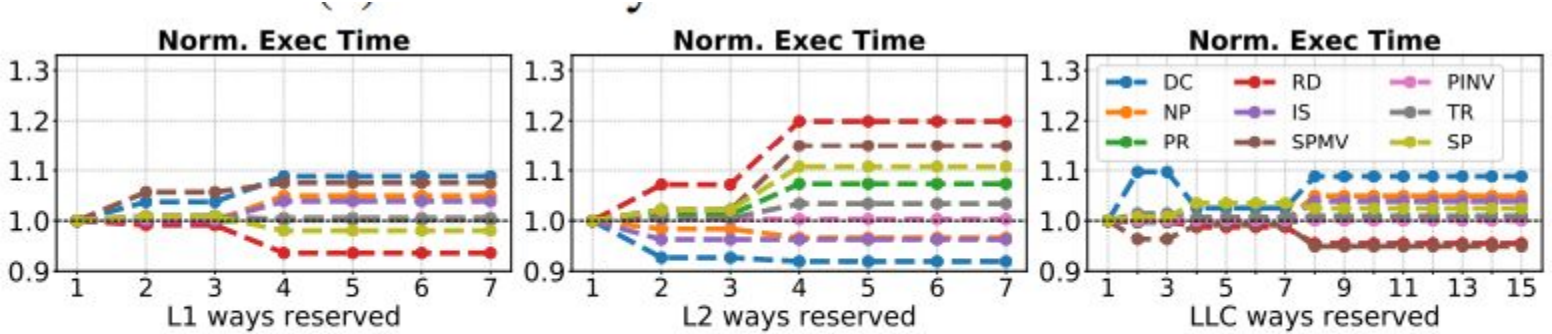


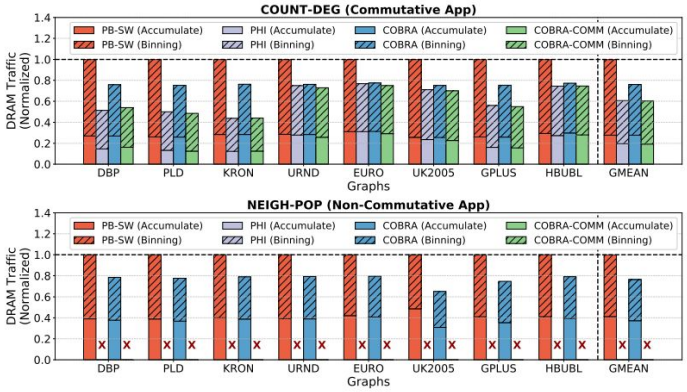
Figure 12: **Efficiency gains from eliminating instruction overhead of binning:** *The* `binupdate` *instruction in COBRA enables an OoO core to exploit more ILP.*

COBRA's Sensitivity to C-Buffer Ways

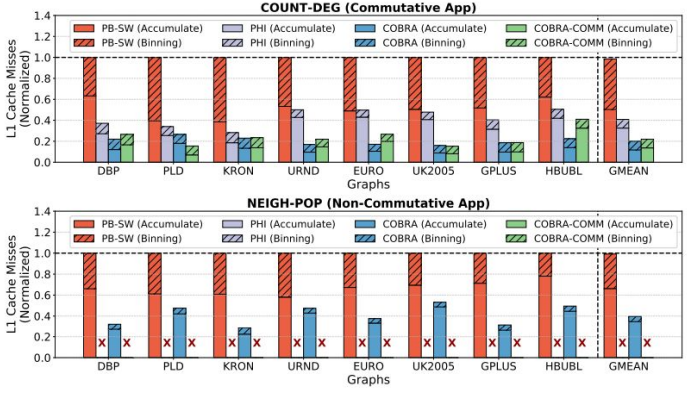


(b) Sensitivity to ways reserved for *C-Buffers*

COBRA vs PHI



(a) DRAM Traffic Reduction



(b) L1 Cache Miss Reduction

Figure 14: **Comparisons against PHI:** *PHI and COBRA-*