Analysis and Optimization of the Memory Hierarchy for Graph Processing Workloads

Abanti Basak, Shuangchen Li, Xing Hu, Sang Min Oh, Xinfeng Xie, Li Zhao*, Xiaowei Jiang*, and Yuan Xie

University of California, Santa Barbara
**Server Architecture Group, Alibaba Inc.*

CRISP
Center for Research on Intelligent Storage and Processing in Memory

UC Santa Barbara
Scalable Energy-efficient Architecture Lab

Executive Summary

Memory-bound behavior in single-machine in-memory graph processing

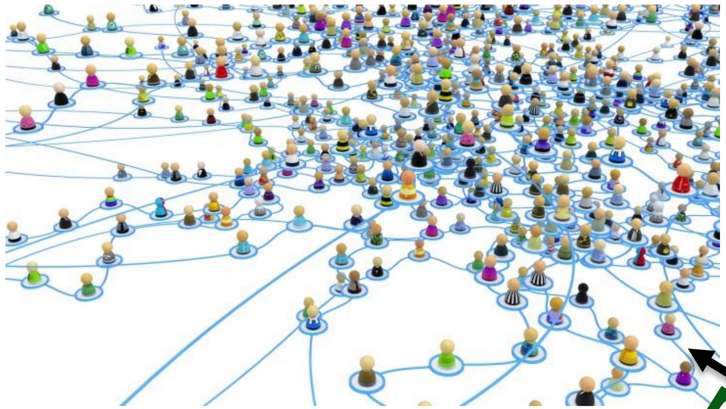
Data-aware characterization of the core and the cache hierarchy to understand the memory-bound behavior

Architecture design and evaluation of DROPLET, a **data-aware and decoupled prefetcher** for graphs to solve the memory access bottleneck

Section I

Memory-bound behavior in single-machine in-memory graph processing

- Application domains
- Single-machine in-memory graph processing
- Memory access bottleneck



Graph Processing

Social networks

Transportation

Recommender systems

Fraud detection

Financial money flows

Web search

Interest in Graph Processing

Google

Microsoft

IBM

Pinterest

Alibaba Group
阿里巴巴集团

facebook

amazon web services

twitter

ORACLE

Single-Machine In-Memory Graph Processing

Many common-case industry and academic graphs fit in RAM of a high-end server

Ringo [SIGMOD ‘15]

Pixie [WWW ‘18]

Wtf [WWW ’13]

SNAP [TIST ‘16]

Ligra [PPoPP ‘13]

GraphJet [VLDB ’16]

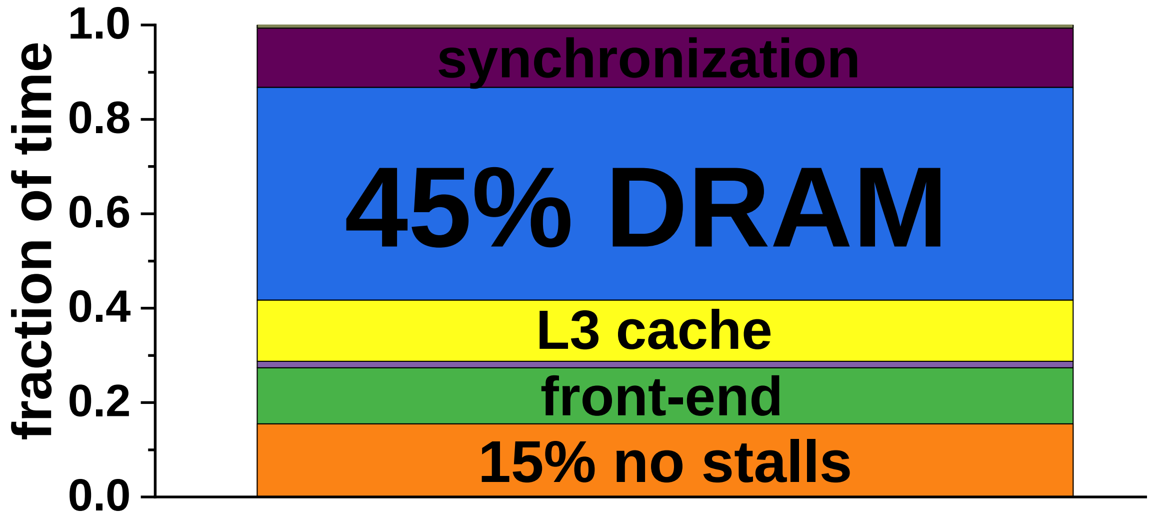
Big-memory machines

Ex: 1) Intel Xeon with 1.5TB RAM
2) HPE’s MACHINE

Bottleneck…

45% of cycles are DRAM-bound stall cycles

Only 15% of cycles are fully utilized by core without stalling



Cycle stack of PageRank on Orkut dataset

Data collected using Sniper on a quad-core architecture

Section II

Data-aware characterization of the core and the cache hierarchy to understand the memory-bound behavior

- Novelty
- Background
- Characterization setup
- Profiling observations
- Summary

Characterization of Core and Cache Hierarchy

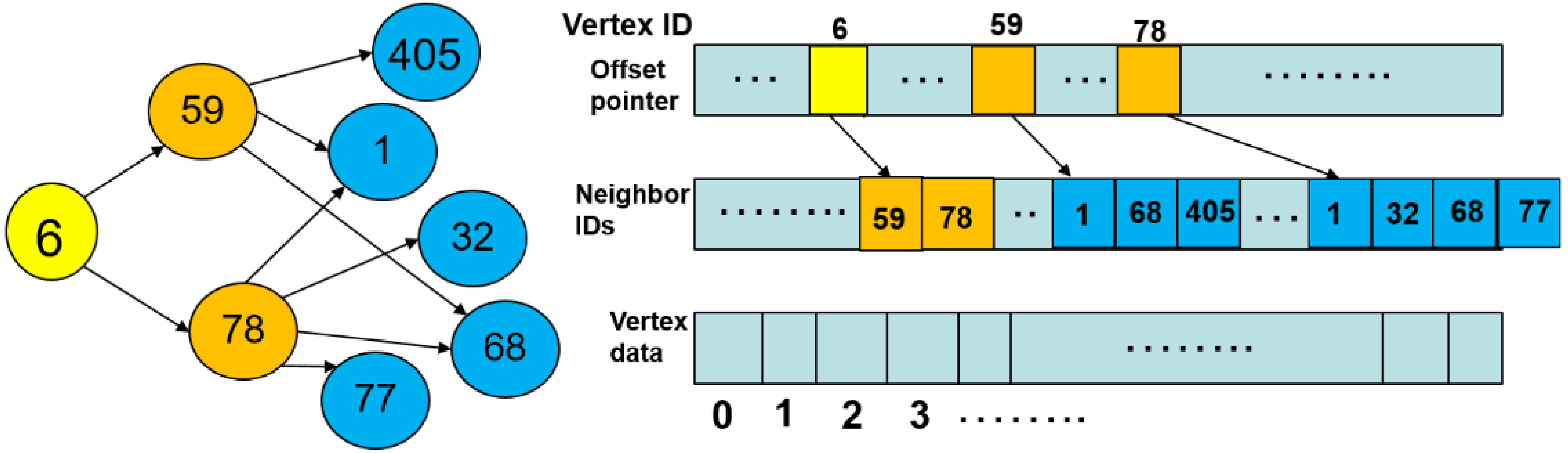
Novelty compared to prior characterization [IISWC ‘15, MASCOTS ‘16, SC ’15]:

data-aware profiling: guidelines to managing different data types

simulated environment: explicit exploration of performance sensitivity of hardware design parameters

Background: Data Type Terminology

Compressed Sparse Row (CSR) data layout



- **Structure data** -> neighbor ID array
- **Property data** -> vertex data array
- **Intermediate data** -> any other data

Experimental Setup

Hardware Characteristics on SniperSIM

- 4-core, 128-entry ROB, 2.66GHz
- Private L1D/I caches, 32KB, 8-way SA, 4 cycles
- Private L2 cache, 256KB, 8-way SA, 8 cycles
- Shared L3, 8MB, 16-way SA, 30 cycles
- DDR3 DRAM, access latency = 45 ns

Algorithms (GAP Benchmark)

- Connected Components (**CC**)
- PageRank (**PR**)
- Betweenness Centrality (**BC**)
- Breadth First Search (**BFS**)
- Single Source Shortest Path (**SSSP**)

Datasets (|V|= # vertices, |E| = # edges)

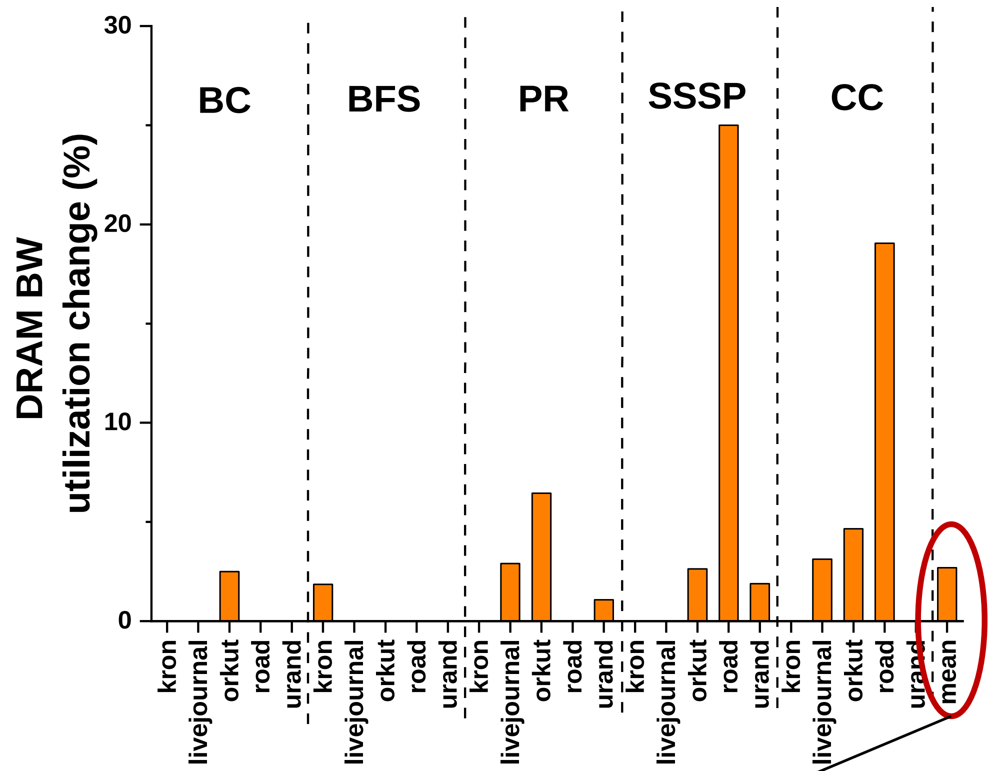
Kron	16.8M |V|	260M |E|
Urand	8.4M |V|	134M |E|
Orkut	3M |V|	117M |E|
LiveJournal	4.8M |V|	68.5M |E|
Road	23.9M |V|	57.7M |E|

Profiling Overview

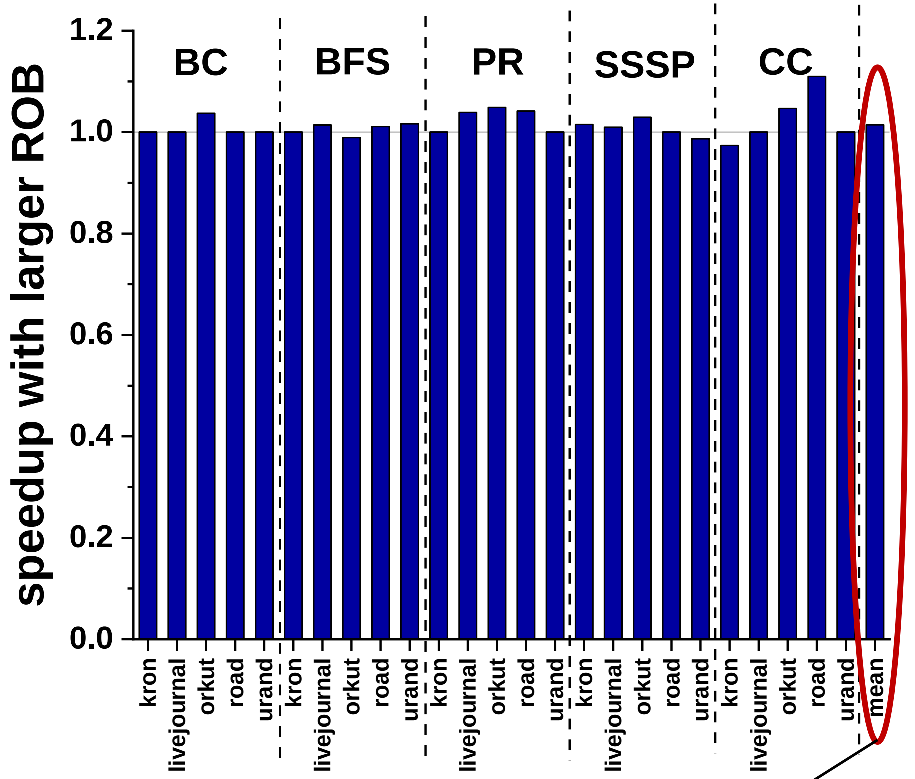
- Can we achieve higher Memory-Level Parallelism (MLP)?
  - If not, what factor is restricting MLP?
- What is the relative performance sensitivity of different cache levels?
- How do different data types use the memory hierarchy?

Instruction Window size does not hinder MLP

We increase IW size to 4X......



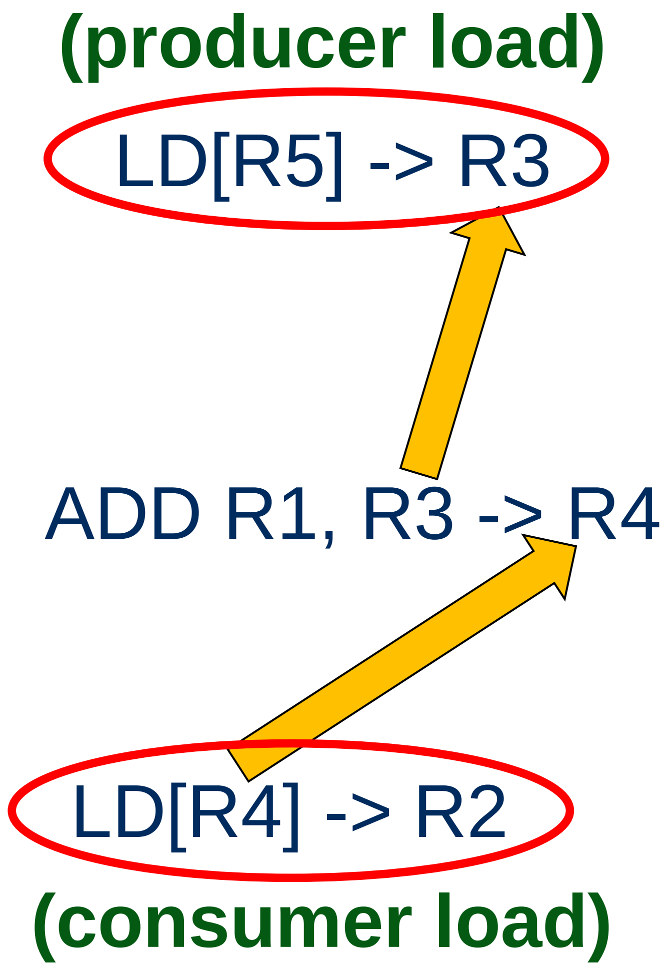
Average memory BW utilization increases by only 2.7%

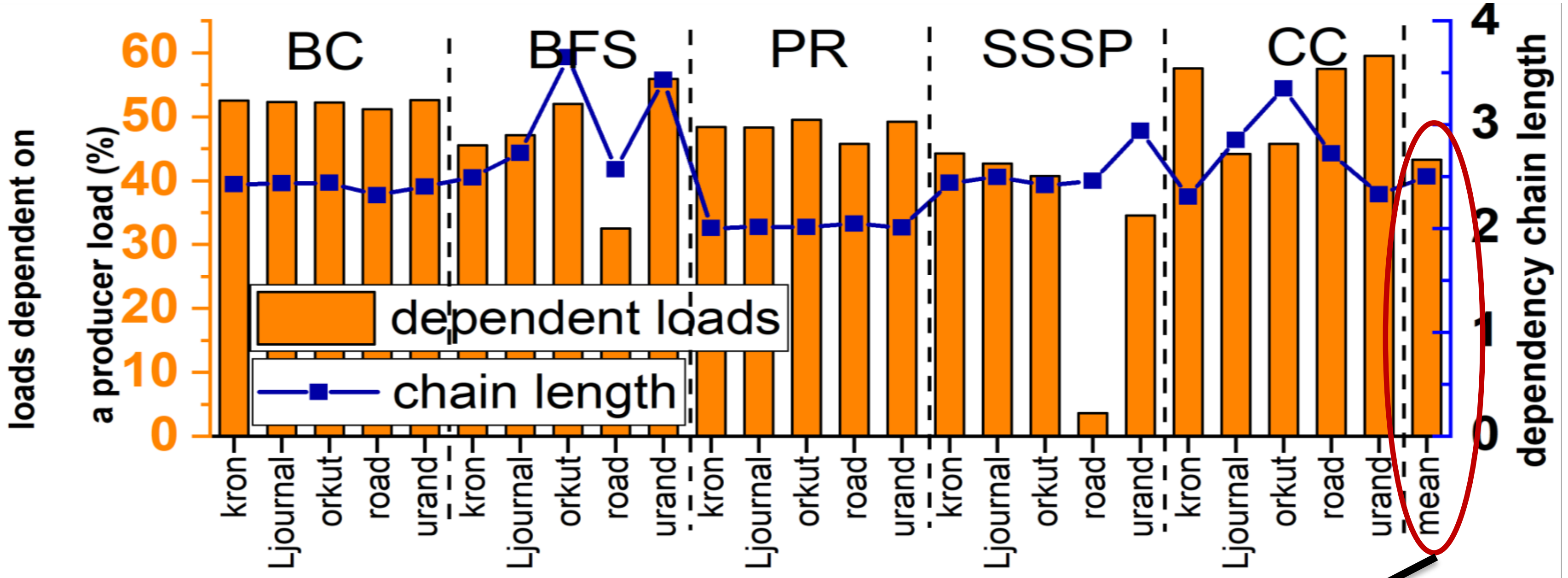


Average speedup is only 1.44%

Load-load dependency hinders MLP

For every load in ROB, we track its dependency backward until we find an older load….

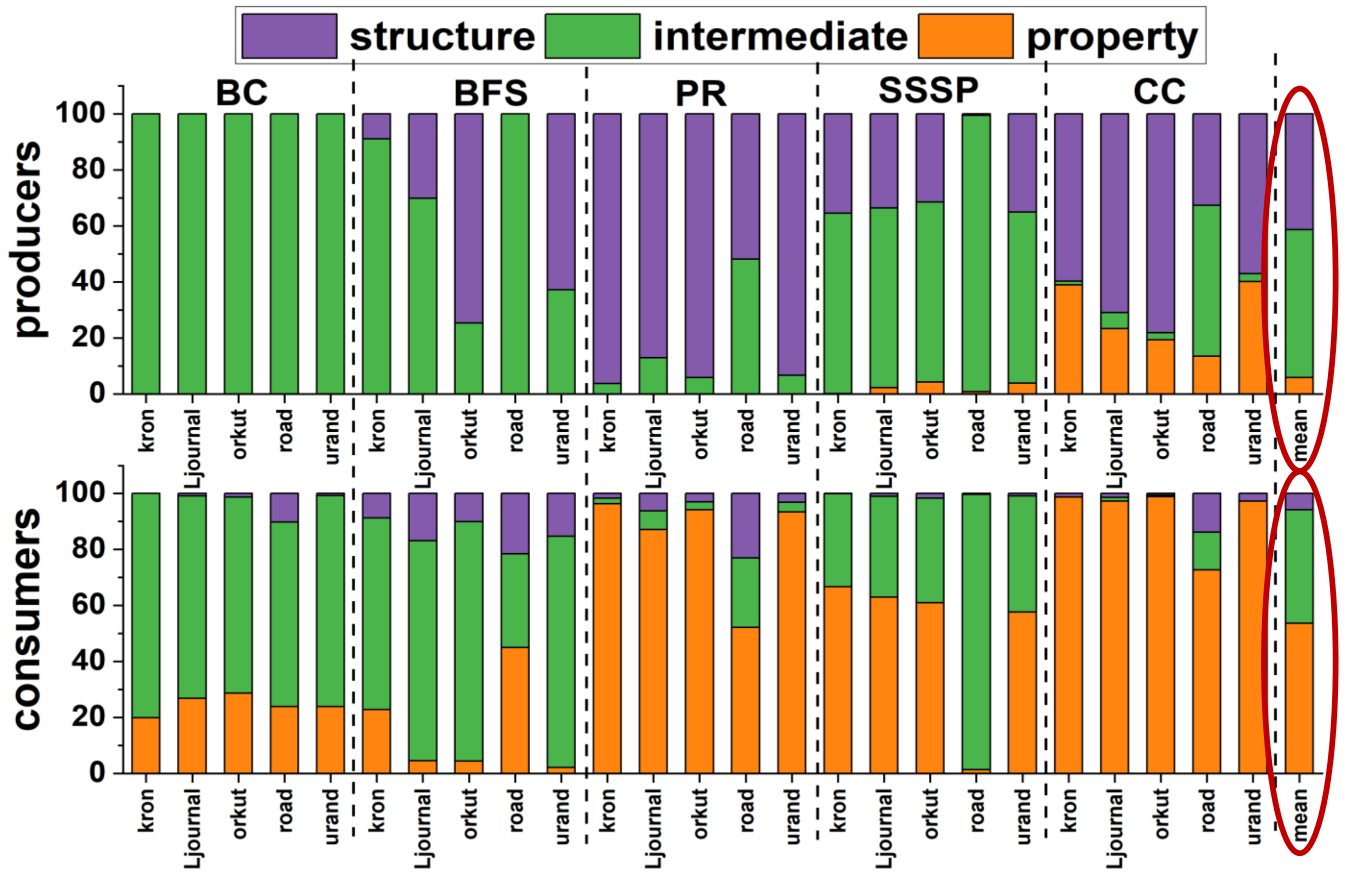




43.2% of loads are part of a dependency chain with chain length of 2.5

Property data is consumer in load-load dependency

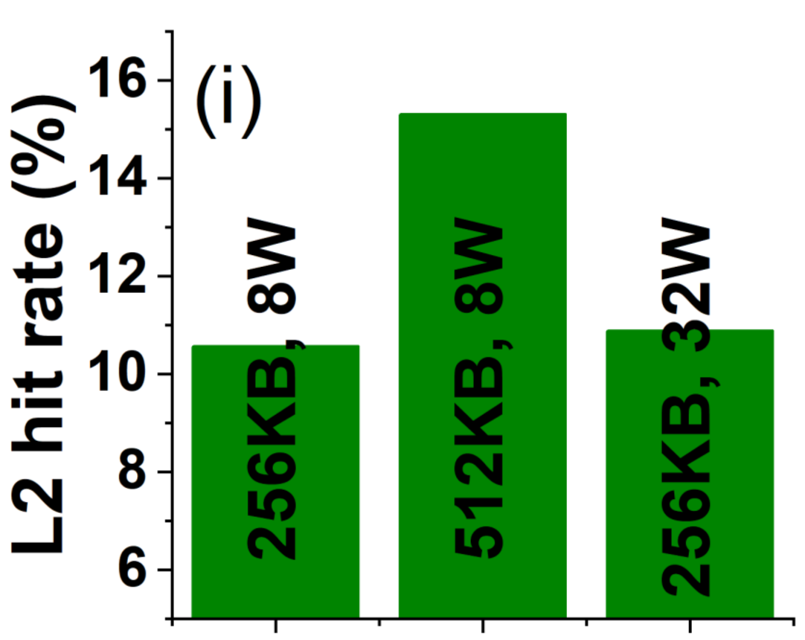
We break down producer and consumer loads by application data type…

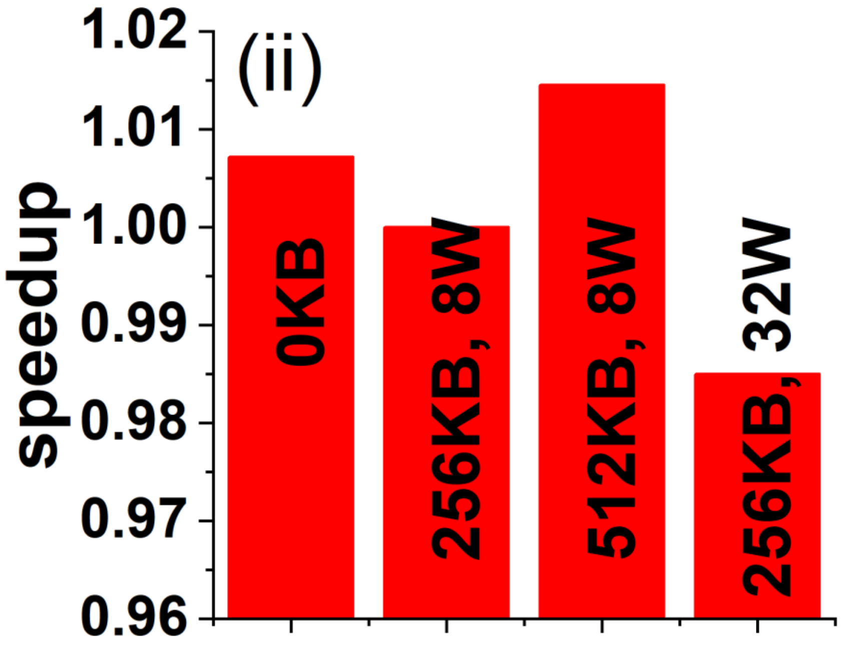


- Property data is mostly a consumer (54%) rather than a producer (6%)
- Structure data is mostly a producer (41%) rather than a consumer (6%)

Private L2 cache shows negligible performance sensitivity

We vary L2 cache configurations…

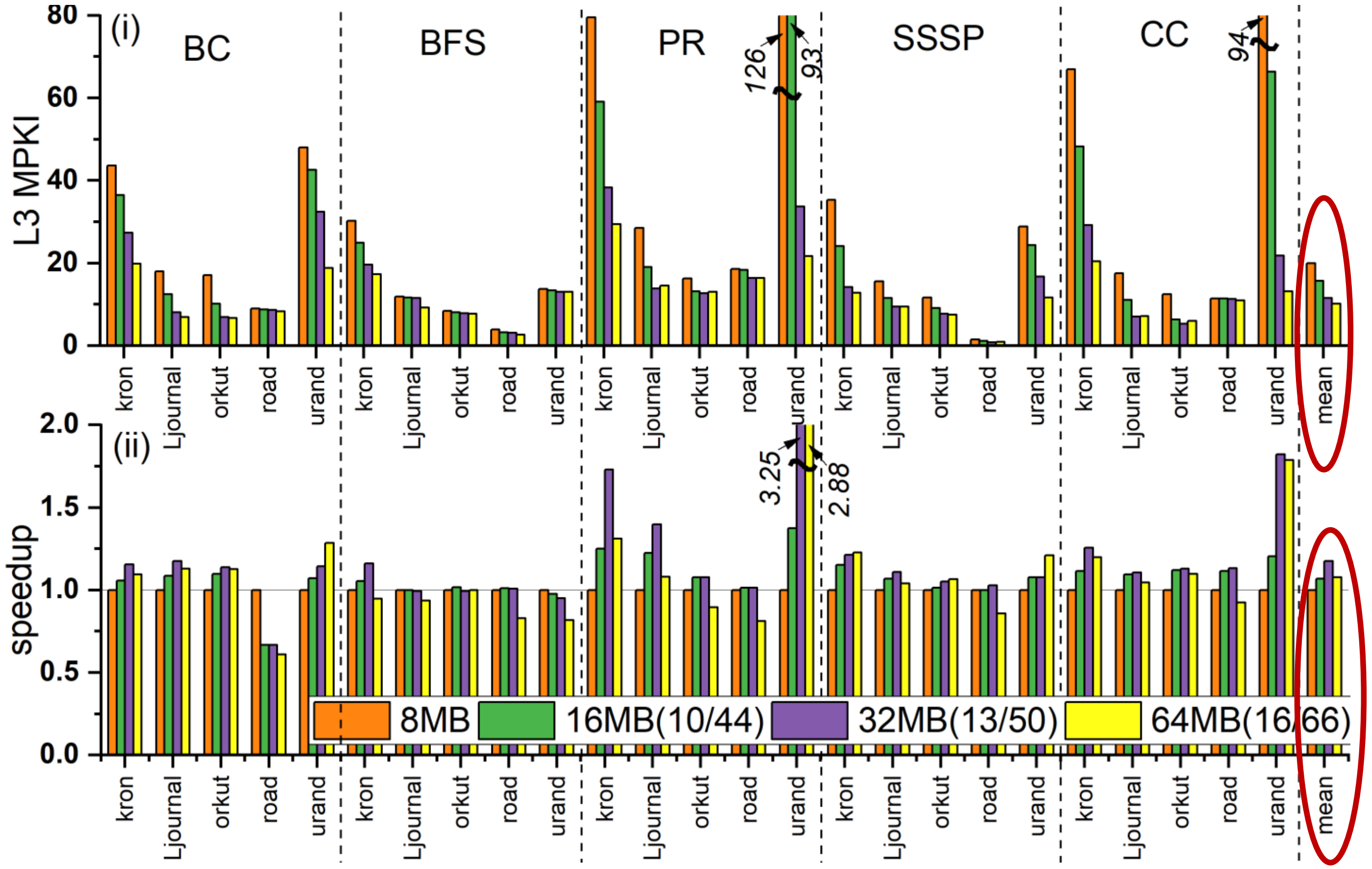




An architecture without private L2 caches is just as fine for graph processing

Shared LLC shows higher performance sensitivity

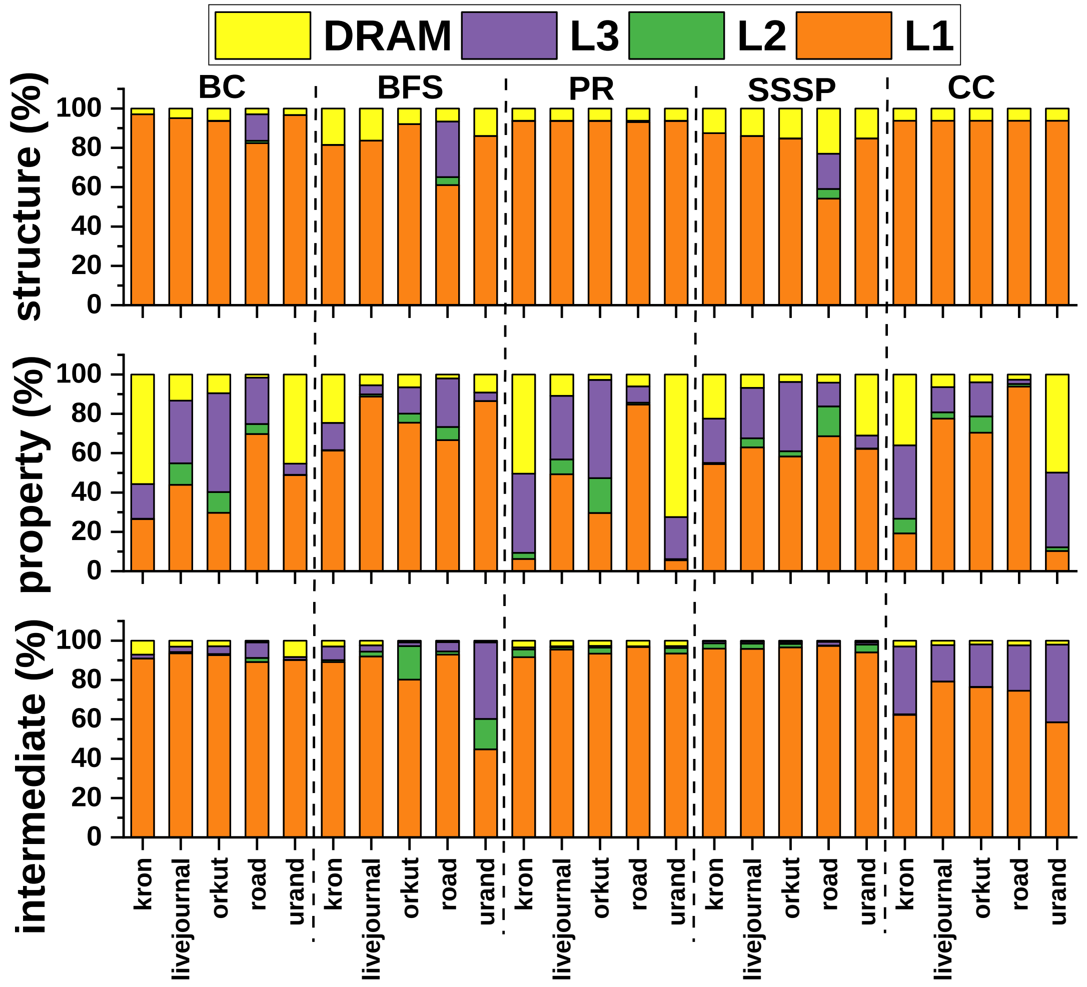
We vary shared LLC capacities…



17.4% performance improvement for 4X increase in LLC capacity

Heterogeneous Reuse Distances

We break down memory hierarchy usage by application data type….

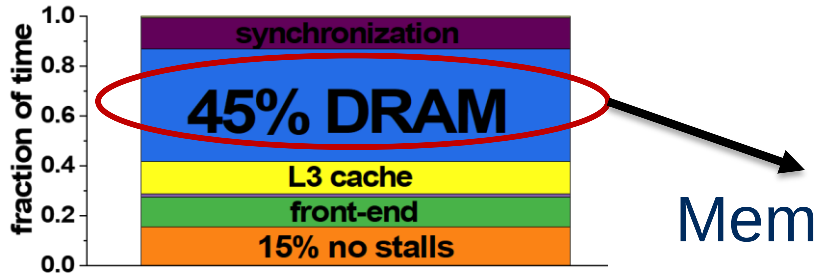


Structure data has the largest reuse distance: serviced by L1 and DRAM

Property data has a larger reuse distance than that serviced by L2 cache

Intermediate data accesses are mostly on-chip cache hits

To Summarize….



Memory-bound behavior caused by:

- Heterogeneous reuse distances of different data types leading to intensive DRAM accesses for structure and property data

- Low MLP due to load-load dependency chains, limiting the possibility of overlapping DRAM accesses

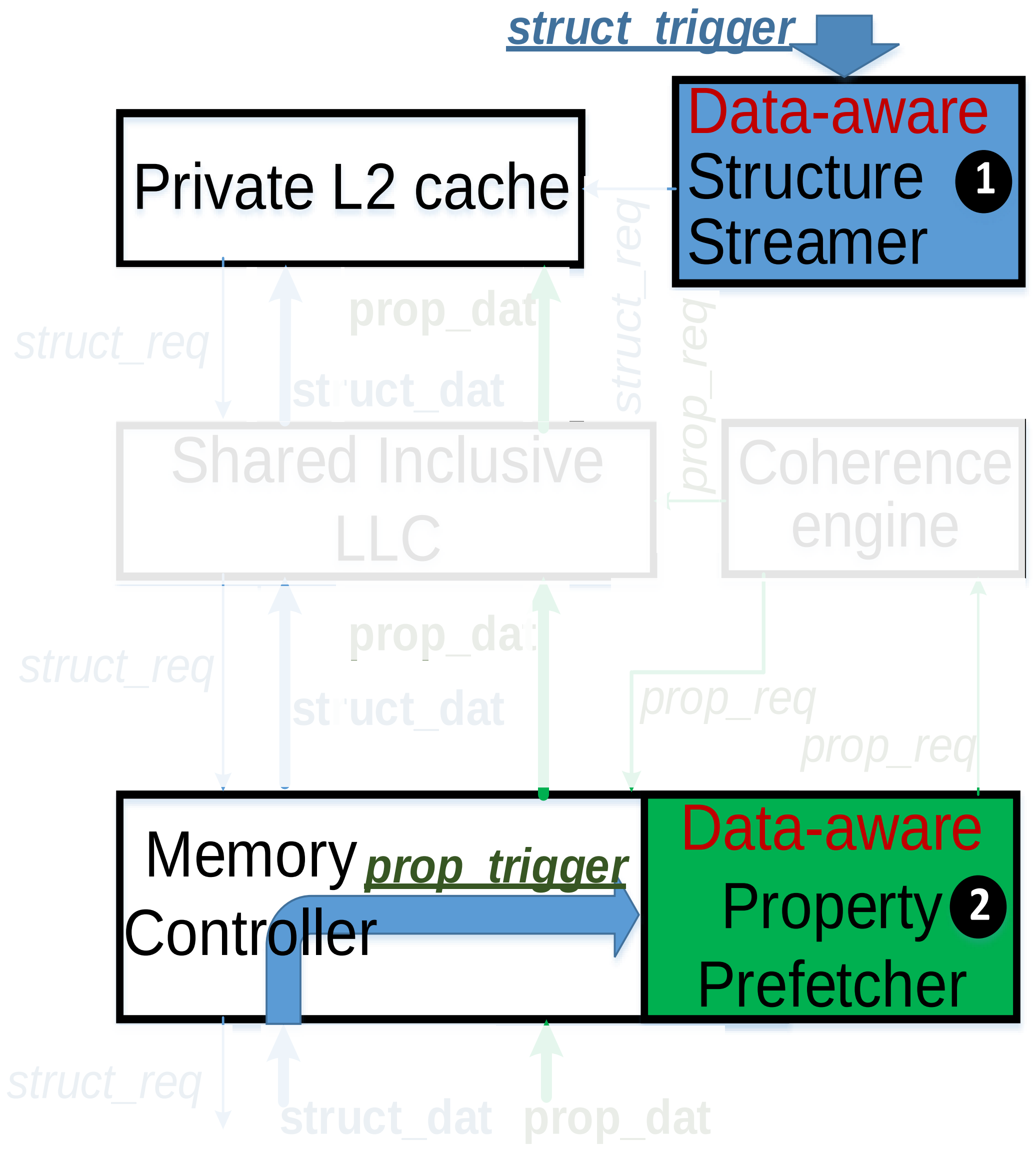
Section III

Architecture design and evaluation of DROPLET, a **data-aware and decoupled prefetcher** for graphs to solve the memory access bottleneck

- DROPLET introduction
- DROPLET overview
- L2 structure streamer
- Property prefetcher
- Evaluation

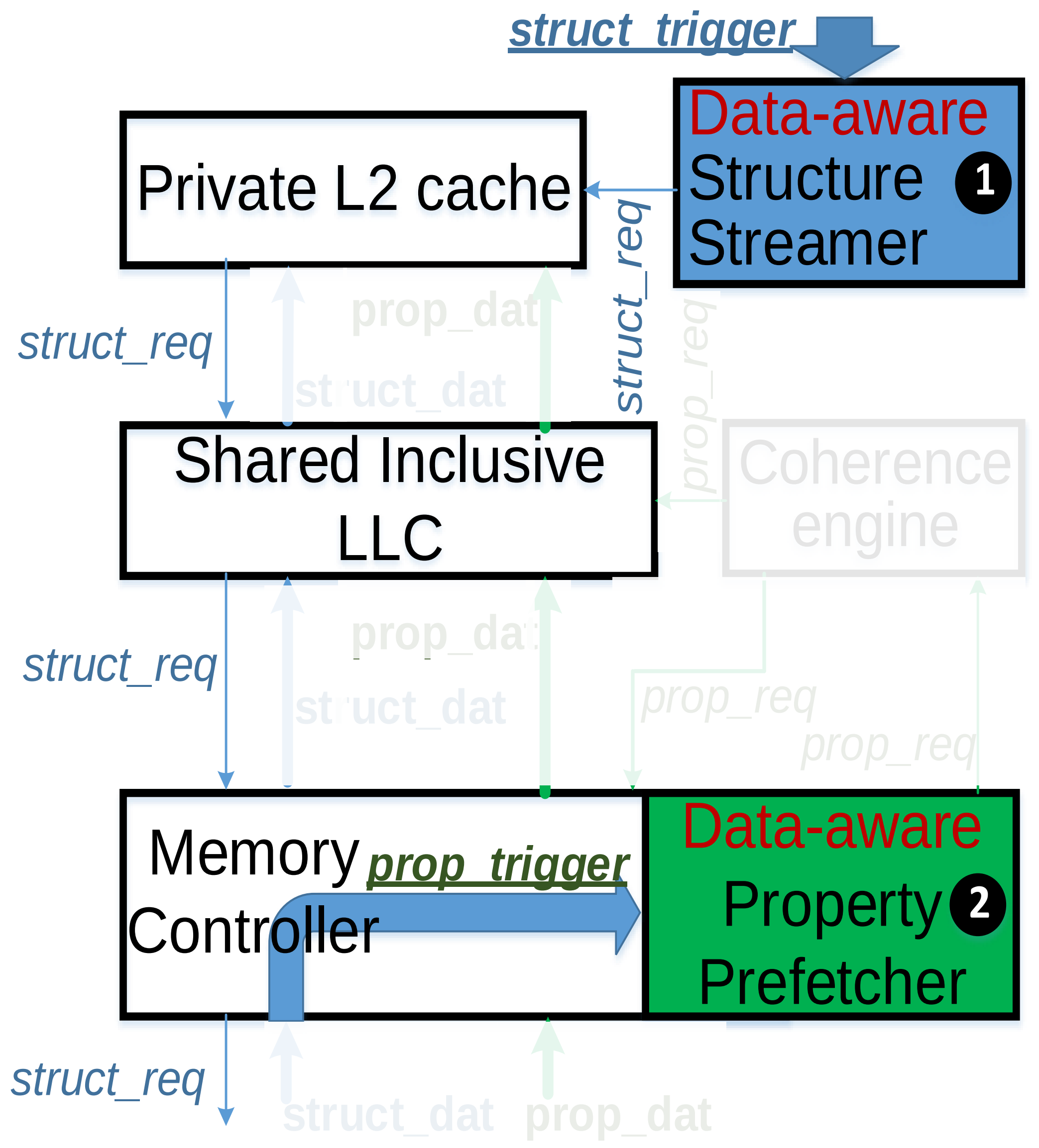
DROPLET: Data-AwaRe DecOuPLed PrEfeTcher

Data-aware: prefetches data types according to reuse distances



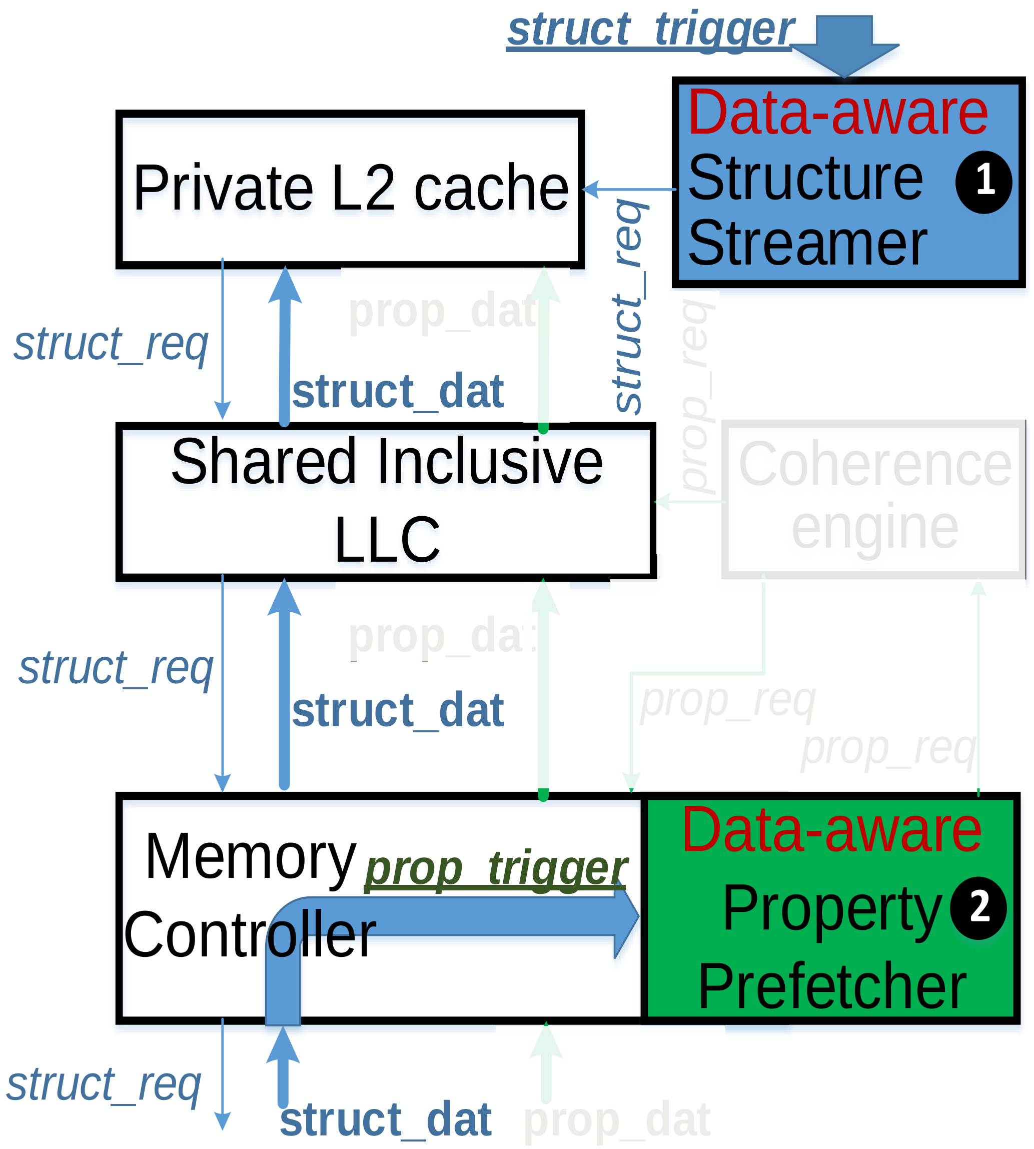
Decoupled: overcomes serialization from load-load dependency

DROPLET Overview



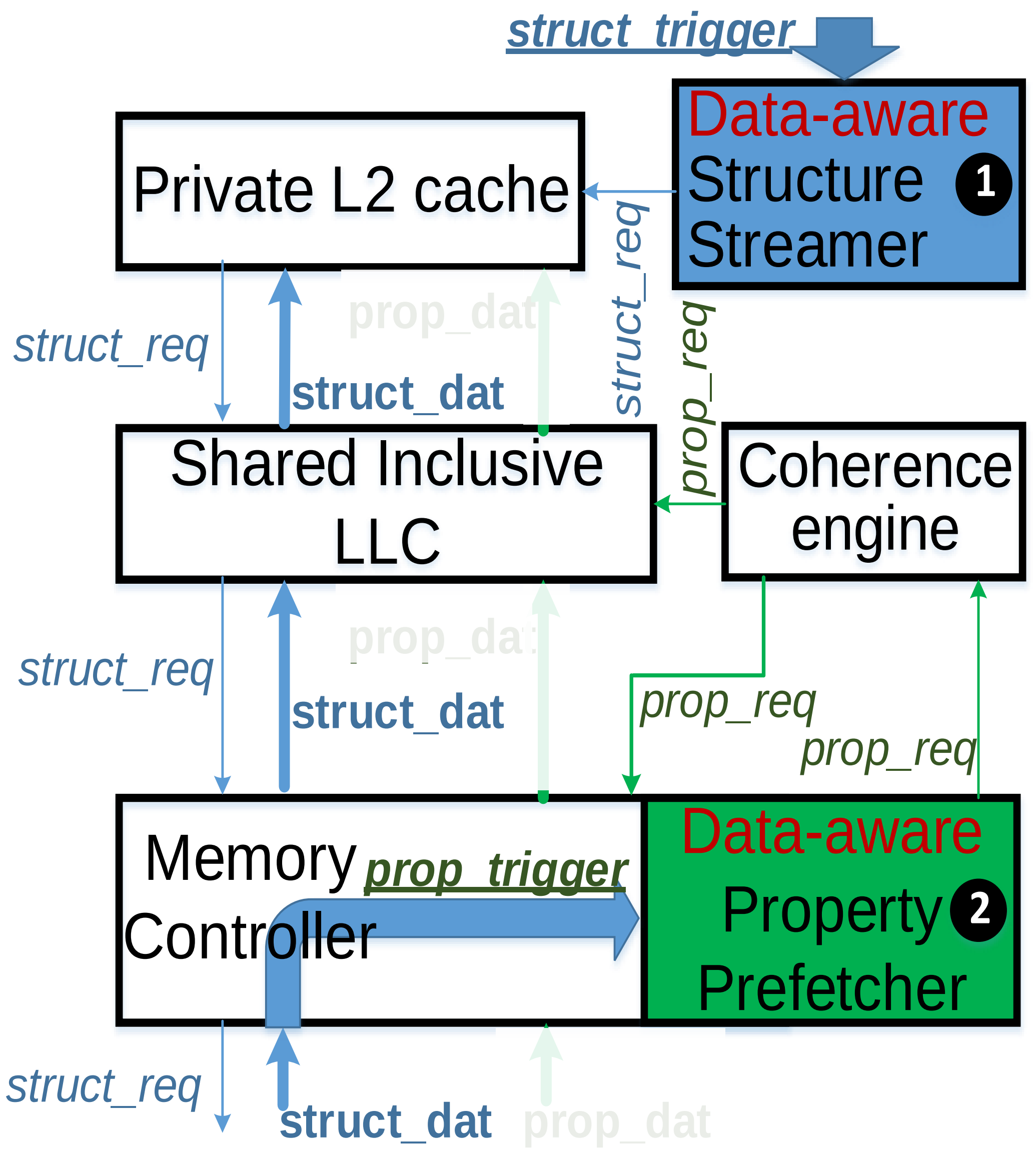
Data-aware structure streamer sends a prefetch request for structure data

DROPLET Overview



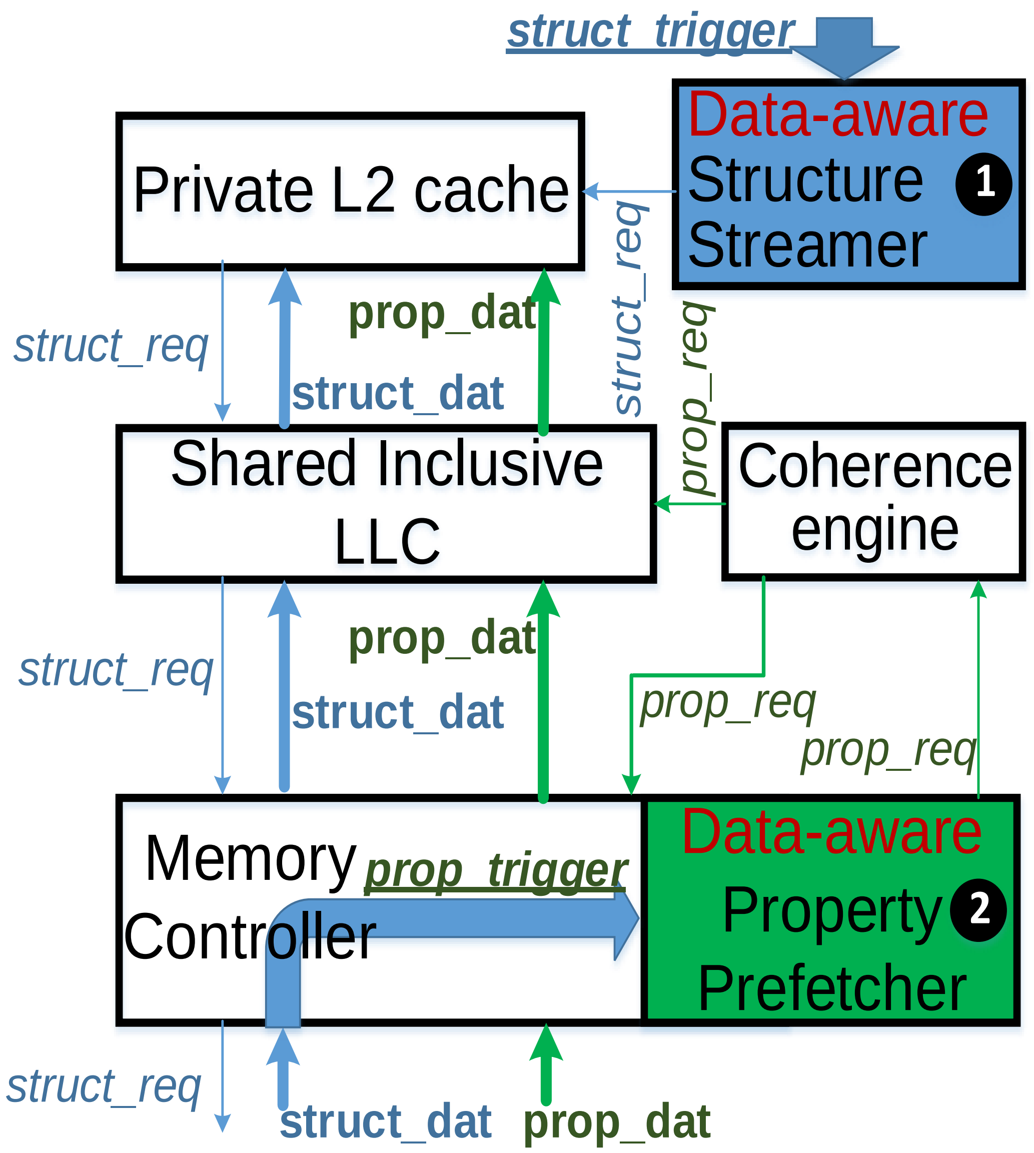
- Copy of prefetched structure cacheline triggers property prefetcher in MC.
- Property prefetcher uses information in structure cacheline to calculate property prefetch addresses.

DROPLET Overview



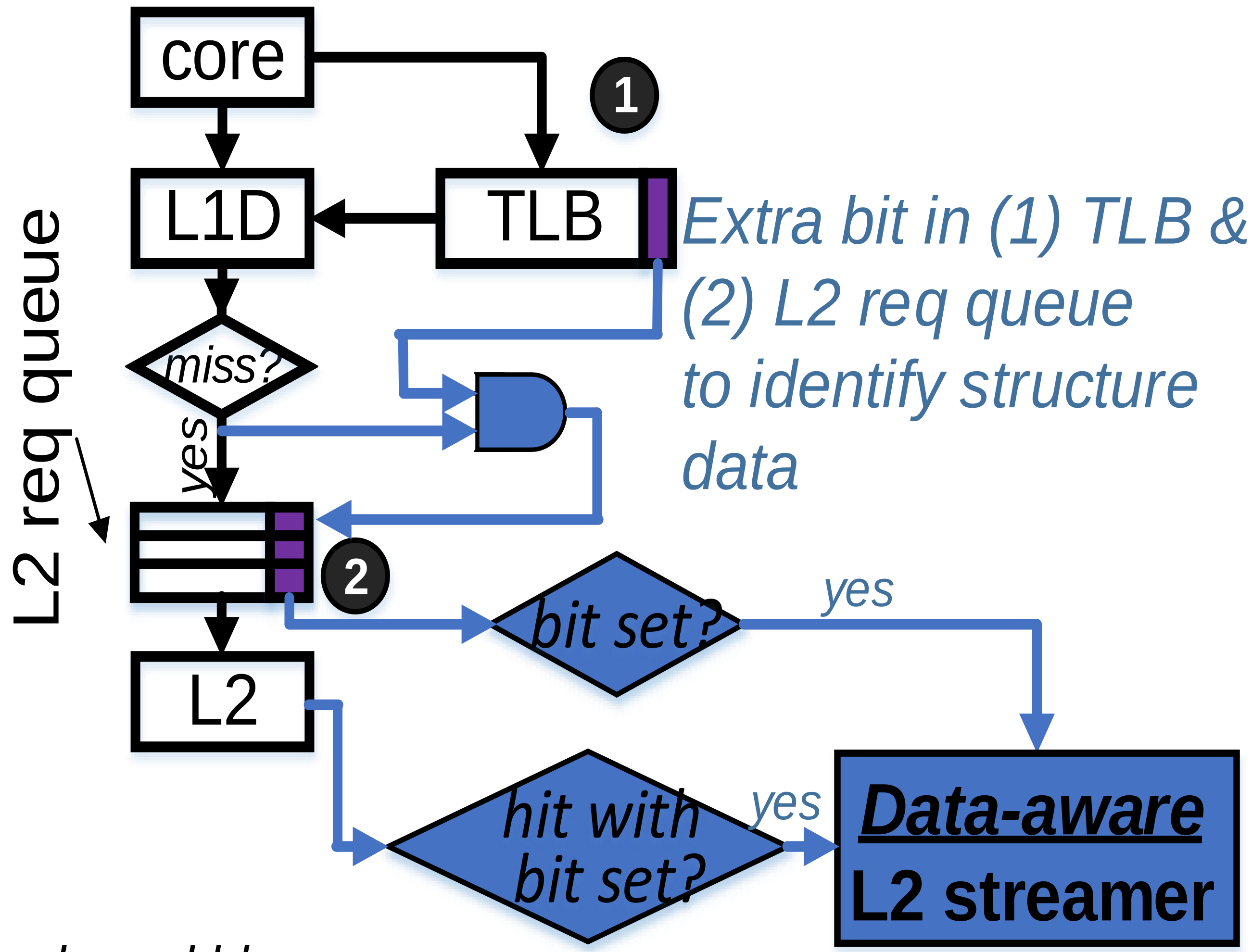
- Property prefetch address is used to check the coherence engine for on-chip presence of data
- If not on-chip, line up request in MC
- If on-chip, query LLC for property data

DROPLET Overview



Place prefetched property data in L2

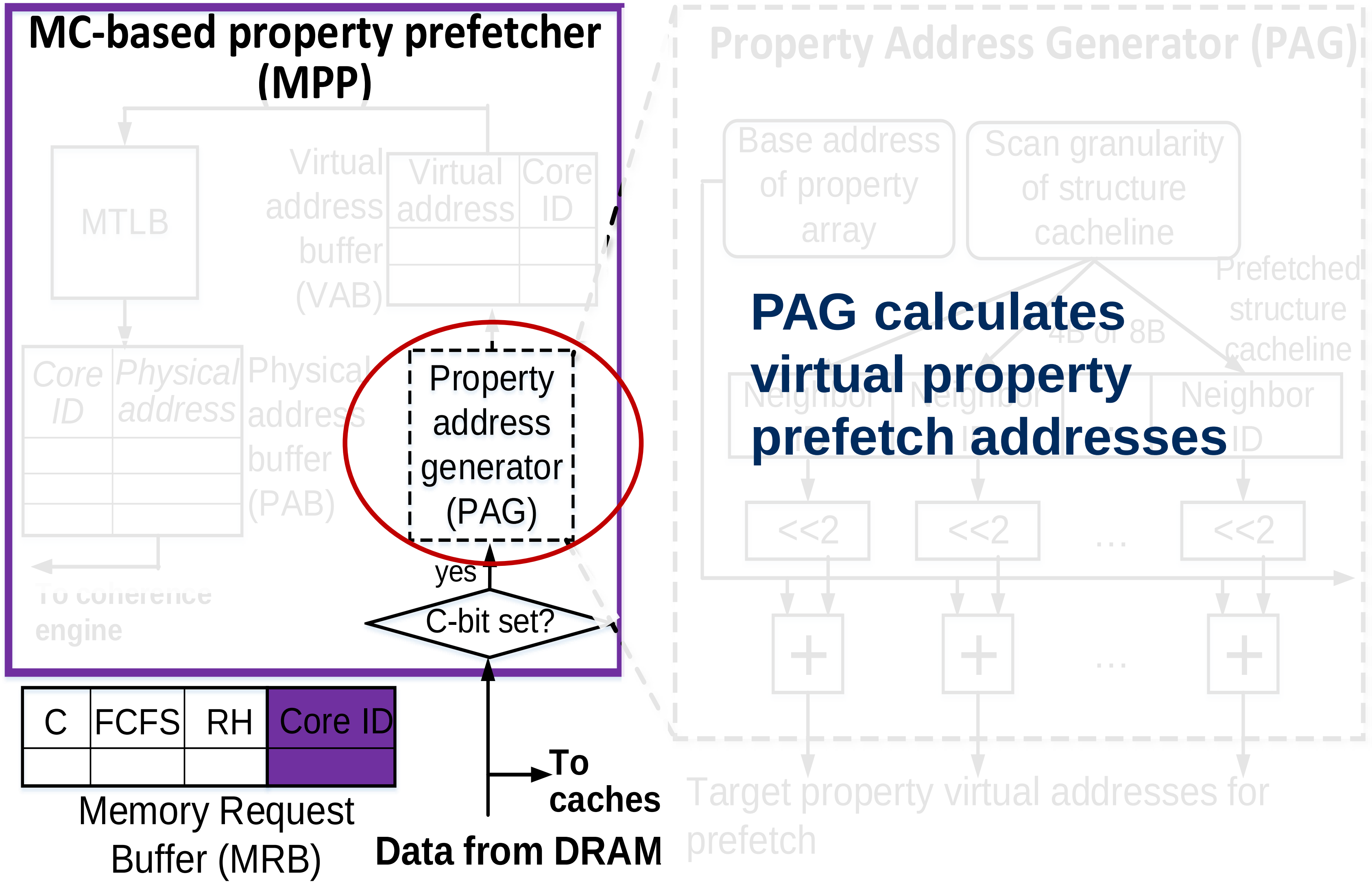
L2 Structure Streamer



Extra bit in (1) TLB & (2) L2 req queue to identify structure data

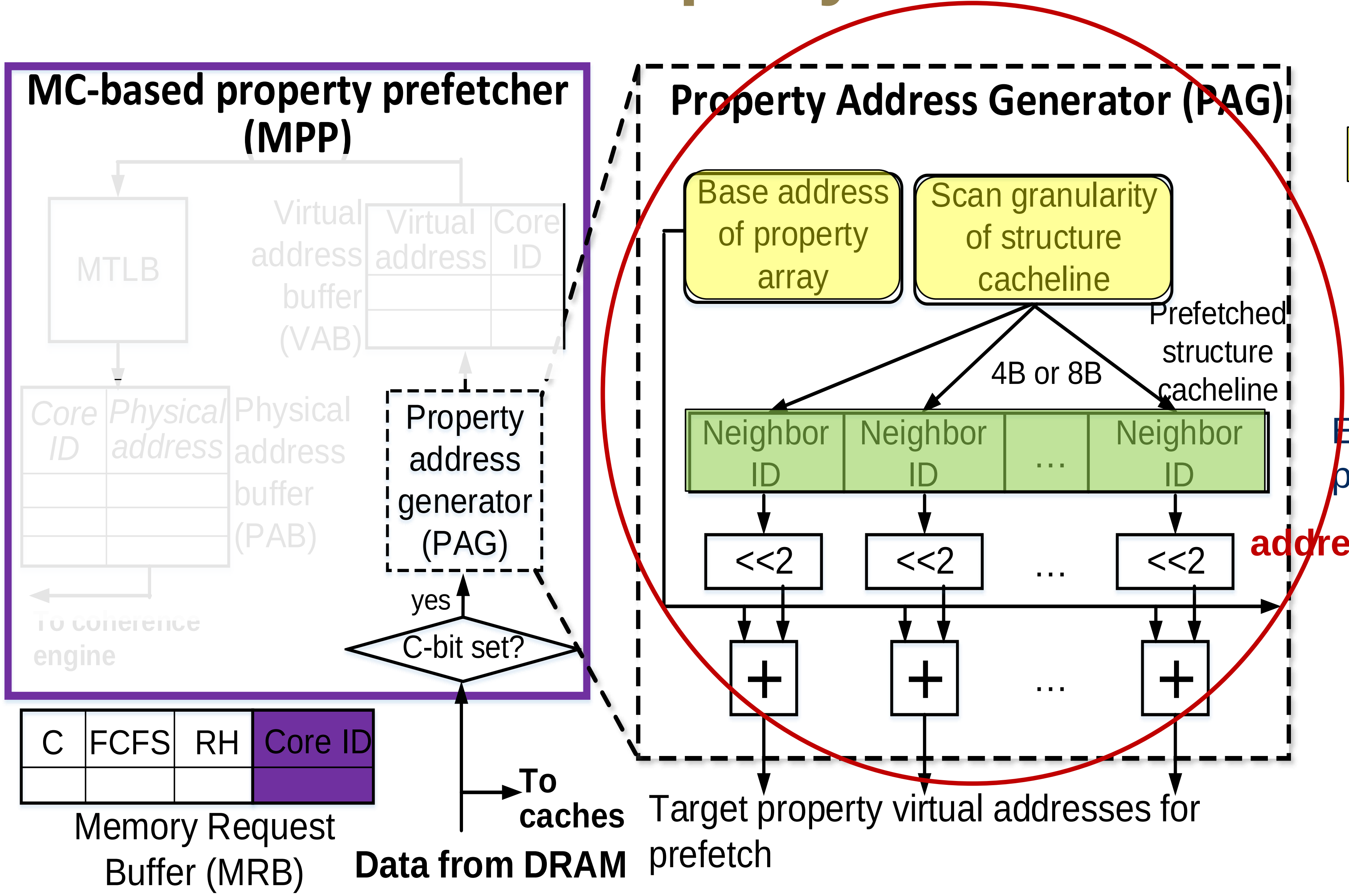
Changes shaded in purple and blue

Property Prefetcher


MC-based property prefetcher (MPP)
MTLB
Virtual address buffer (VAB)
Virtual address
Core ID
Core ID
Physical address
Physical address buffer (PAB)
Property address generator (PAG)
yes
C-bit set?
To coherence engine
C
FCFS
RH
Core ID
Memory Request Buffer (MRB)
To caches
Data from DRAM
Property Address Generator (PAG)
Base address of property array
Scan granularity of structure cacheline
4B or 8B
Prefetched structure cacheline
Neighbor
Neighbor
Neighbor ID
<<2
<<2
<<2
+
+
+
Target property virtual addresses for prefetch

PAG calculates virtual property prefetch addresses

Property Prefetcher


MC-based property prefetcher (MPP)
MTLB
Virtual address buffer (VAB)
Virtual address
Core ID
Core ID
Physical address
Physical address buffer (PAB)
Property address generator (PAG)
To coherence engine
yes
C-bit set?
C
FCFS
RH
Core ID
Memory Request Buffer (MRB)
To caches
Data from DRAM
Property Address Generator (PAG)
Base address of property array
Scan granularity of structure cacheline
4B or 8B
Prefetched structure cacheline
Neighbor ID
Neighbor ID
…
Neighbor ID
<<2
<<2
…
<<2
+
+
…
+
Target property virtual addresses for prefetch

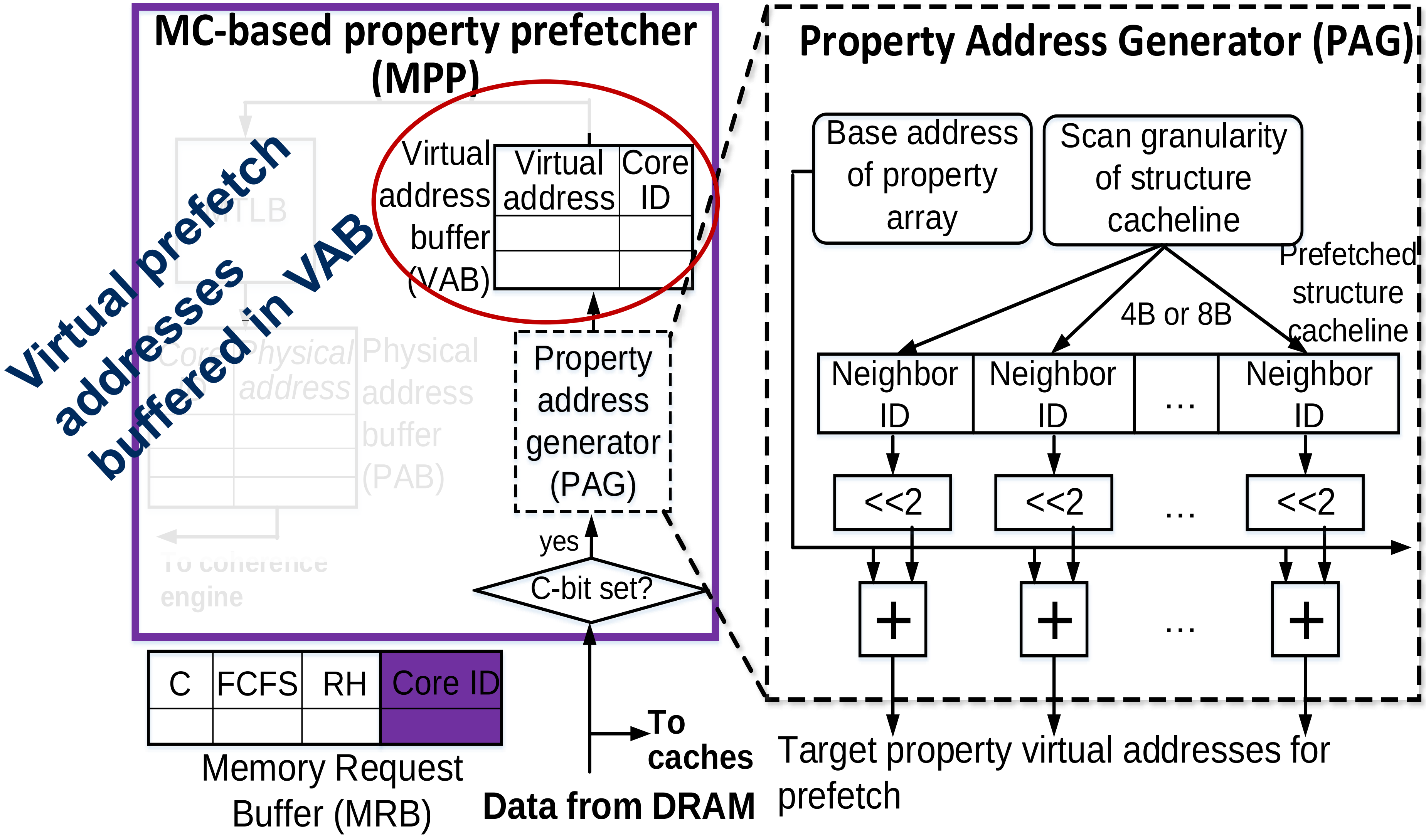
🟨 Information obtained from application layer

🟩 Prefetched structure cacheline

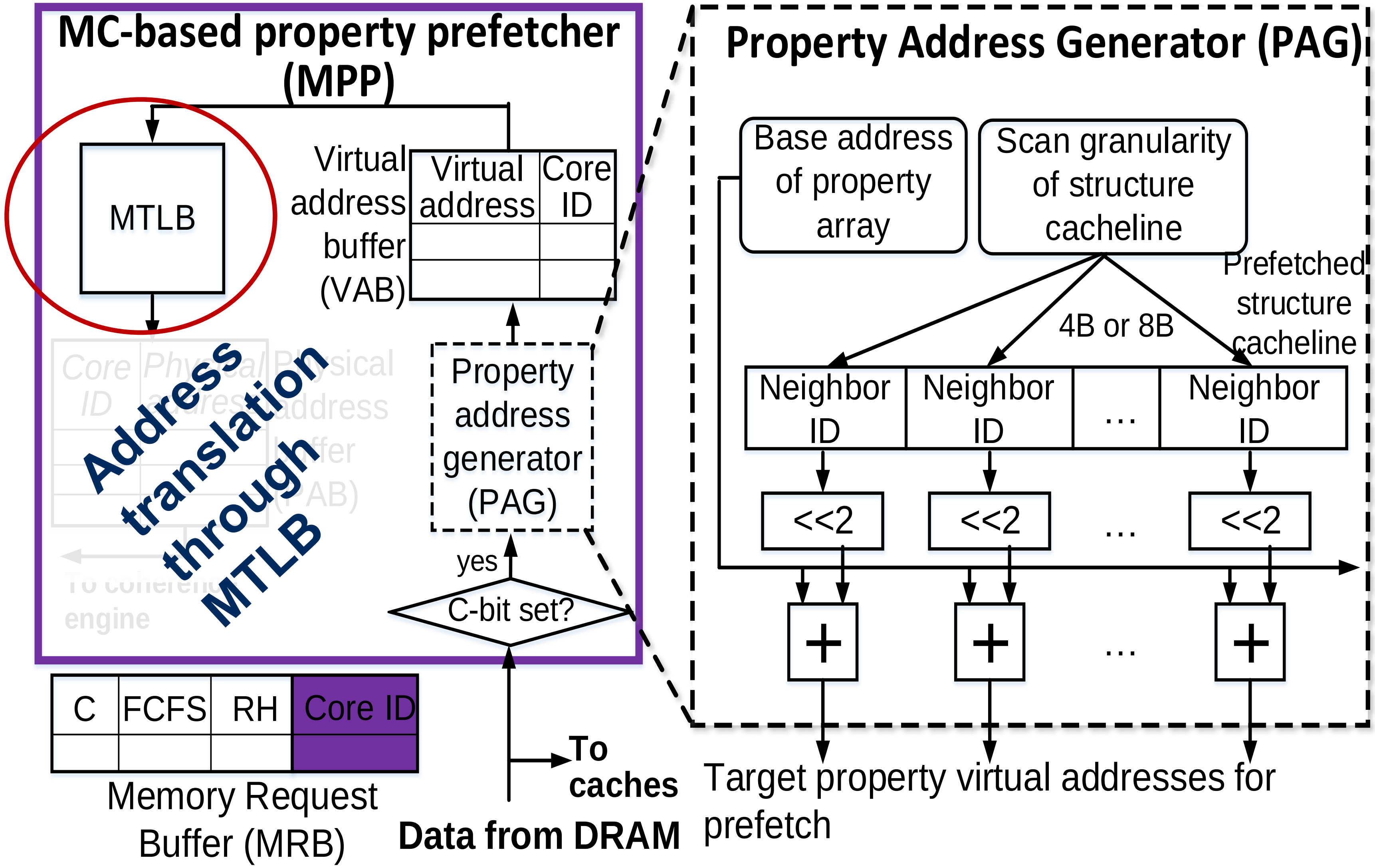
Equation to calculate property addresses:

address = base + 4 x neighbor ID

Property Prefetcher

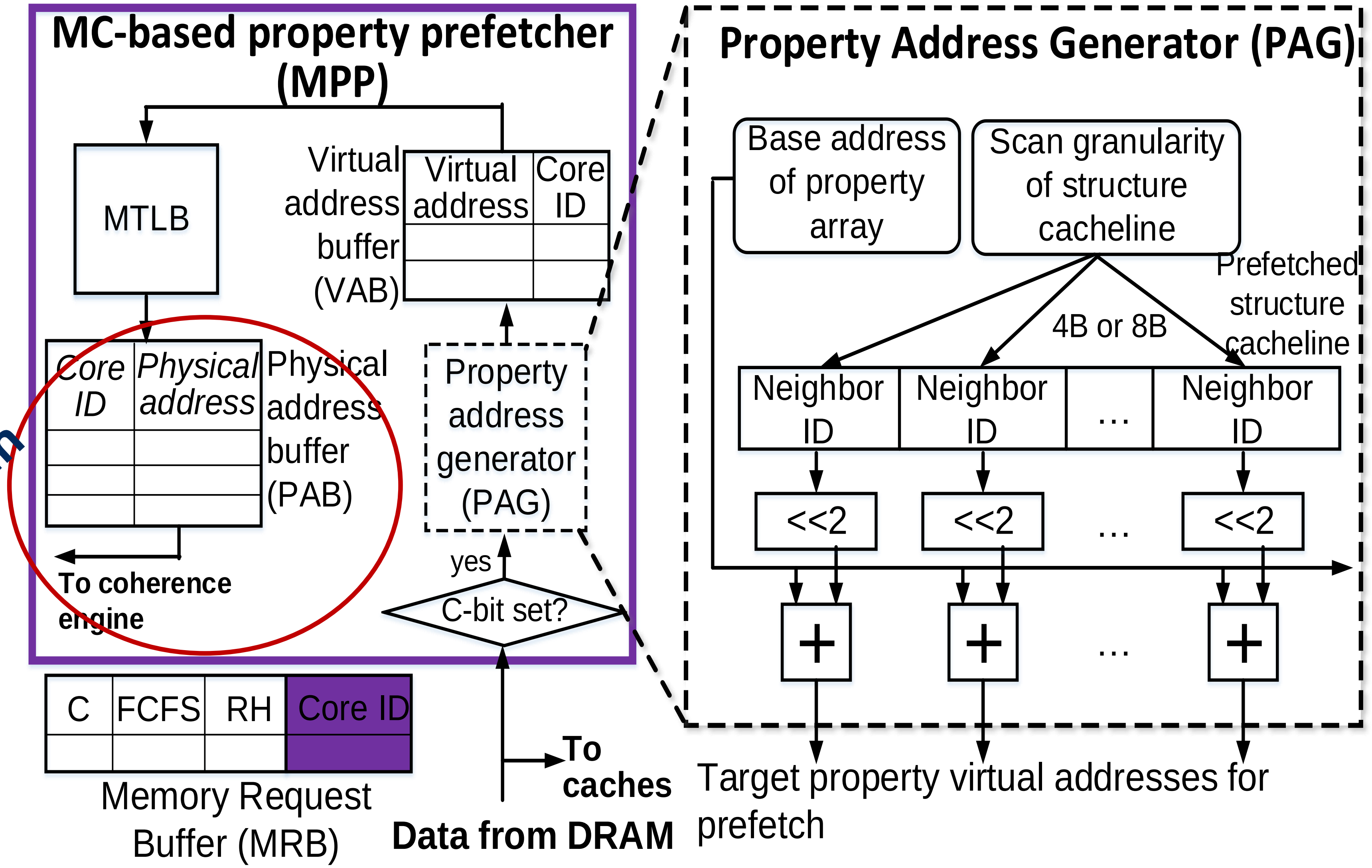

MC-based property prefetcher (MPP)
TLB
Virtual address buffer (VAB)
Virtual address
Core ID
Core
Physical address
Physical address buffer (PAB)
Property address generator (PAG)
yes
C-bit set?
To coherence engine
Virtual prefetch addresses buffered in VAB
C
FCFS
RH
Core ID
Memory Request Buffer (MRB)
To caches
Data from DRAM
Property Address Generator (PAG)
Base address of property array
Scan granularity of structure cacheline
4B or 8B
Prefetched structure cacheline
Neighbor ID
Neighbor ID
…
Neighbor ID
<<2
<<2
…
<<2
+
+
…
+
Target property virtual addresses for prefetch

Property Prefetcher

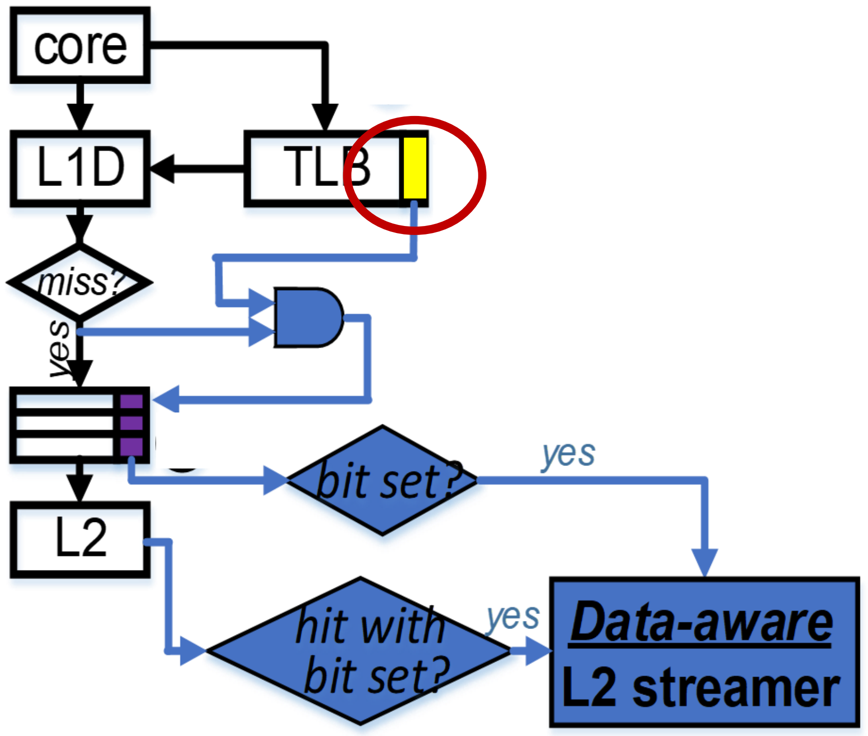

MC-based property prefetcher (MPP)
MTLB
Virtual address buffer (VAB)
Virtual address
Core ID
Core ID
Physical address
Physical address buffer (PAB)
To coherence engine
Address translation through MTLB
Property address generator (PAG)
yes
C-bit set?
C
FCFS
RH
Core ID
Memory Request Buffer (MRB)
To caches
Data from DRAM
Property Address Generator (PAG)
Base address of property array
Scan granularity of structure cacheline
4B or 8B
Prefetched structure cacheline
Neighbor ID
Neighbor ID
…
Neighbor ID
<<2
<<2
…
<<2
+
+
…
+
Target property virtual addresses for prefetch

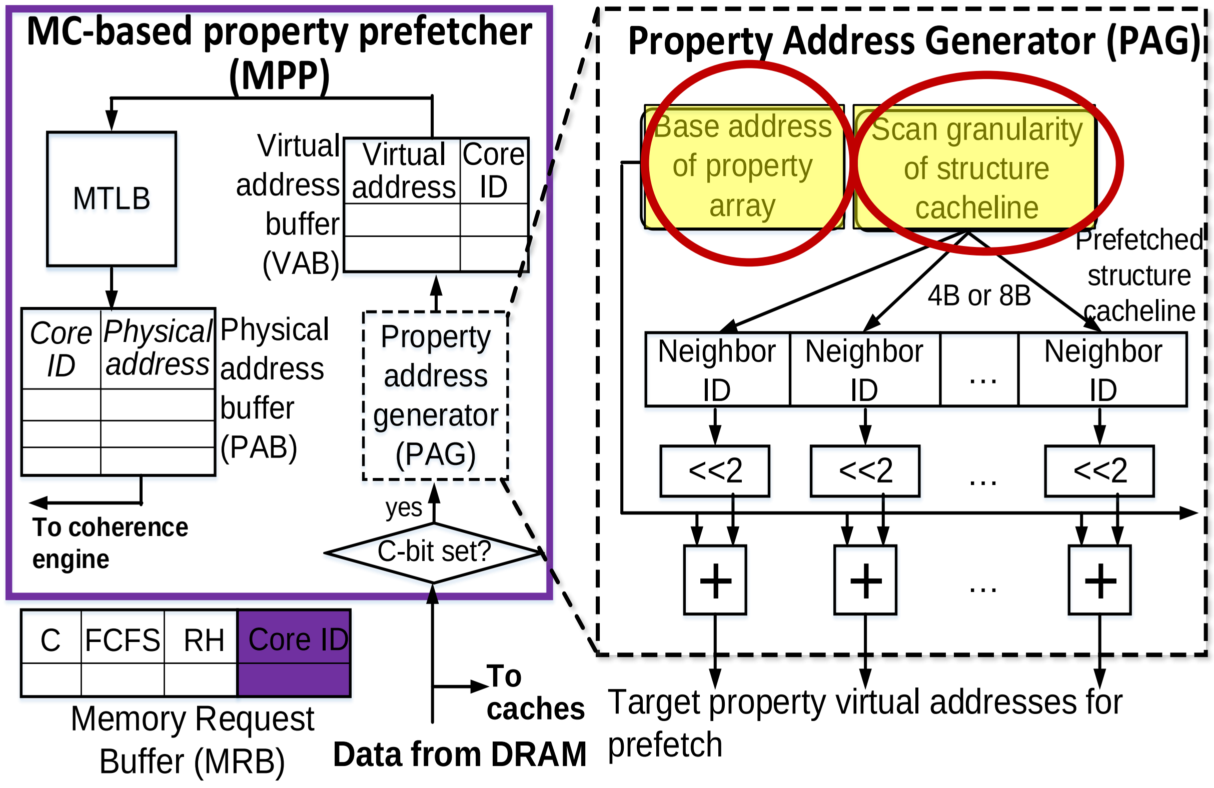
Property Prefetcher

Physical addresses buffered in PAB


MC-based property prefetcher (MPP)
MTLB
Virtual address buffer (VAB)
Virtual address
Core ID
Core ID
Physical address
Physical address buffer (PAB)
To coherence engine
Property address generator (PAG)
yes
C-bit set?
C
FCFS
RH
Core ID
Memory Request Buffer (MRB)
To caches
Data from DRAM
Property Address Generator (PAG)
Base address of property array
Scan granularity of structure cacheline
Prefetched structure cacheline
4B or 8B
Neighbor ID
Neighbor ID
…
Neighbor ID
<<2
<<2
<<2
+
+
+
Target property virtual addresses for prefetch

Help From Application Layer




MC-based property prefetcher (MPP)
MTLB
Virtual address buffer (VAB)
Virtual address
Core ID
Core ID
Physical address
Physical address buffer (PAB)
Property address generator (PAG)
To coherence engine
yes
C-bit set?
C
FCFS
RH
Core ID
Memory Request Buffer (MRB)
To caches
Data from DRAM
Property Address Generator (PAG)
Base address of property array
Scan granularity of structure cacheline
Prefetched structure cacheline
4B or 8B
Neighbor ID
Neighbor ID
...
Neighbor ID
<<2
<<2
...
<<2
+
+
...
+
Target property virtual addresses for prefetch

Specialized ***malloc*** passes these information from application to hardware

More in paper!

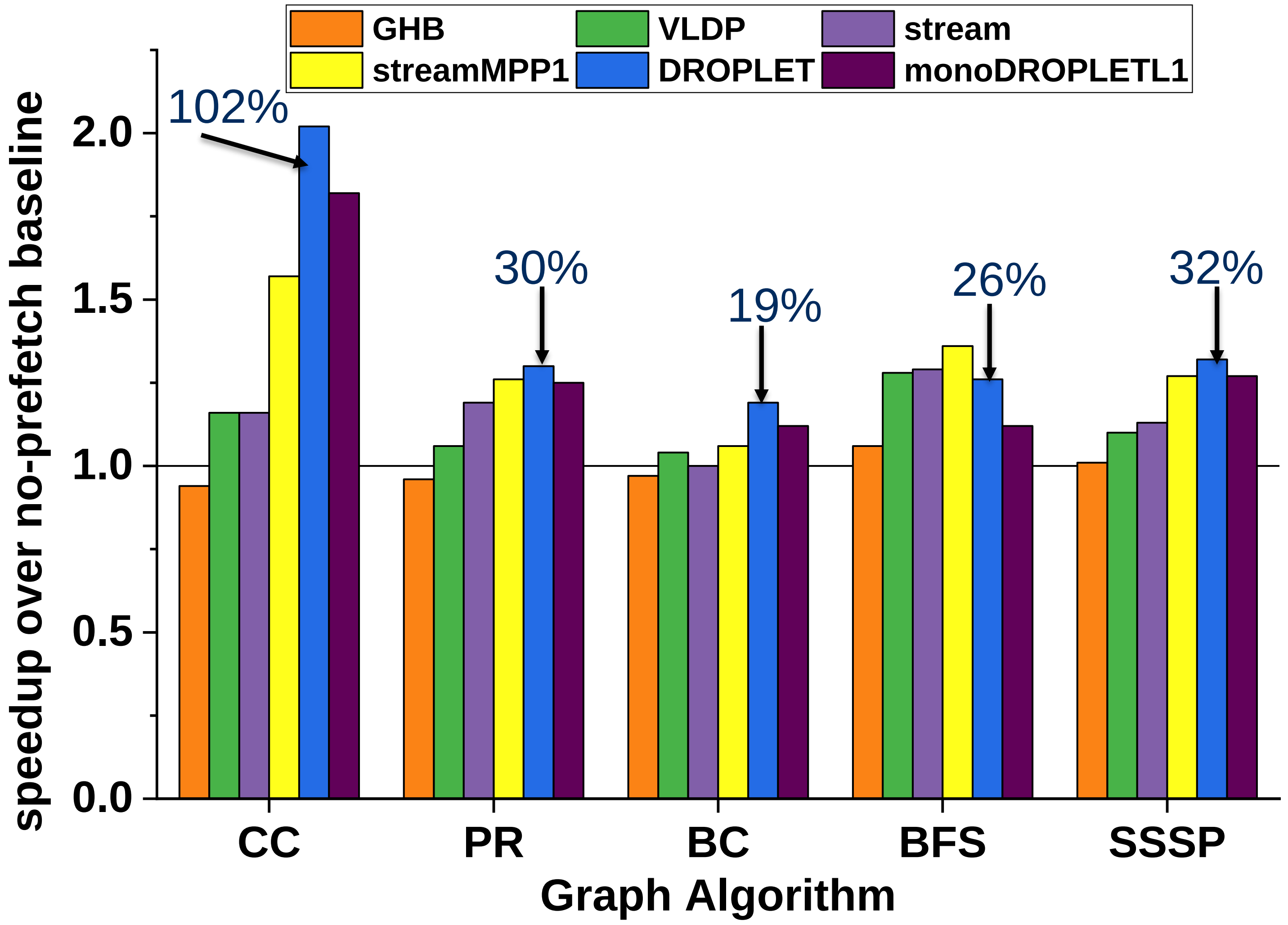
Experiments

DROPLET is compared to:

- **No-prefetch** baseline
- Conventional L2 **stream** prefetcher
- Variable Length Delta Prefetcher (**VLDP**) at L2
- Global History Buffer (**GHB**) at L2
- **streamMPP1**: conventional L2 streamer + property prefetcher in MC
- **monoDROPLETL1**: monolithic data-aware streamer and property prefetcher at L1. Similar to state-of-the-art graph prefetcher (ICS ’16*).

* S. Ainsworth and T. M. Jones, “*Graph prefetching Using Data Structure Knowledge,*” ICS 2016

Data-Aware + Decoupled = High Performance



DROPLET achieves performance improvements of:

- **19%-102%** over a no-prefetch baseline
- **9%-74%** over a stream prefetcher
- **14%-74%** over VLDP
- **19%-115%** over GHB
- **4%-12%** over state-of-the art graph prefetcher

More experiments in paper!

Conclusions

- **Memory access** is the primary bottleneck in single-machine in-memory graph processing.

- Memory access bottleneck arises from:

    1) **Heterogeneous reuse distances** of different data types, leading to DRAM accesses for graph structure and property data.

    2) **Load-load dependency** restricts MLP.

- DROPLET, a **data-aware and decoupled prefetcher**, effectively solves memory access bottleneck.

Thanks! Questions

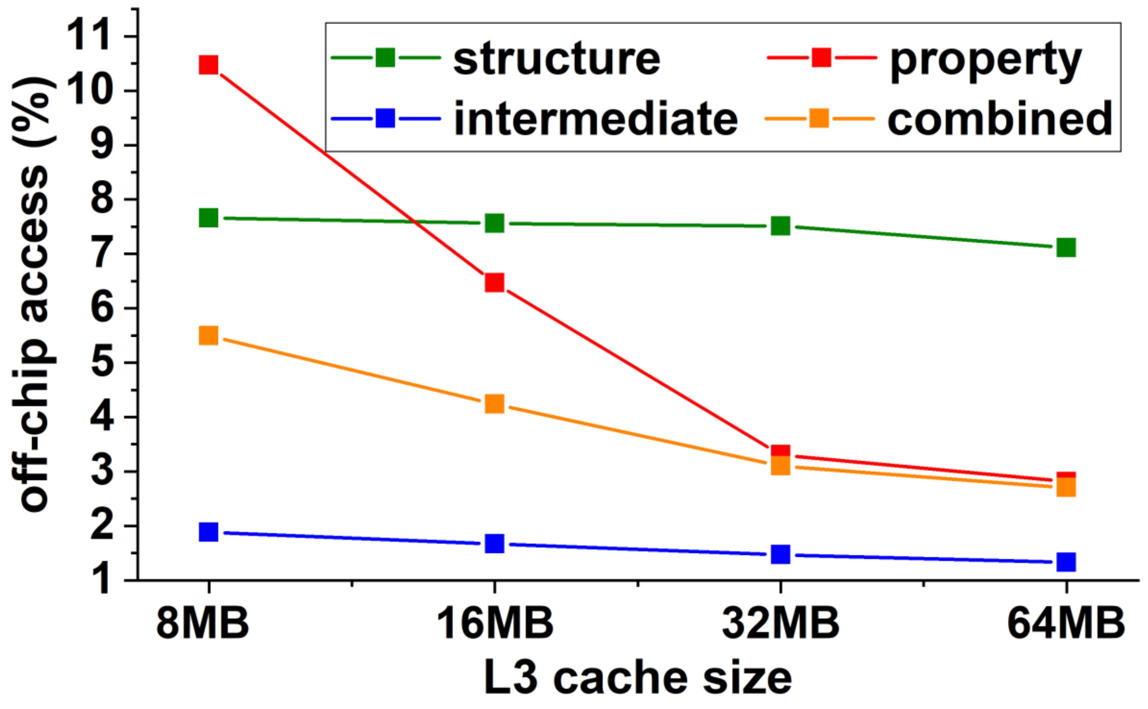
Analysis and Optimization of the Memory Hierarchy for Graph Processing Workloads

Abanti Basak, Shuangchen Li, Xing Hu, Sang Min Oh, Xinfeng Xie, Li Zhao*, Xiaowei Jiang*, and Yuan Xie

University of California, Santa Barbara
**Server Architecture Group, Alibaba Inc.*

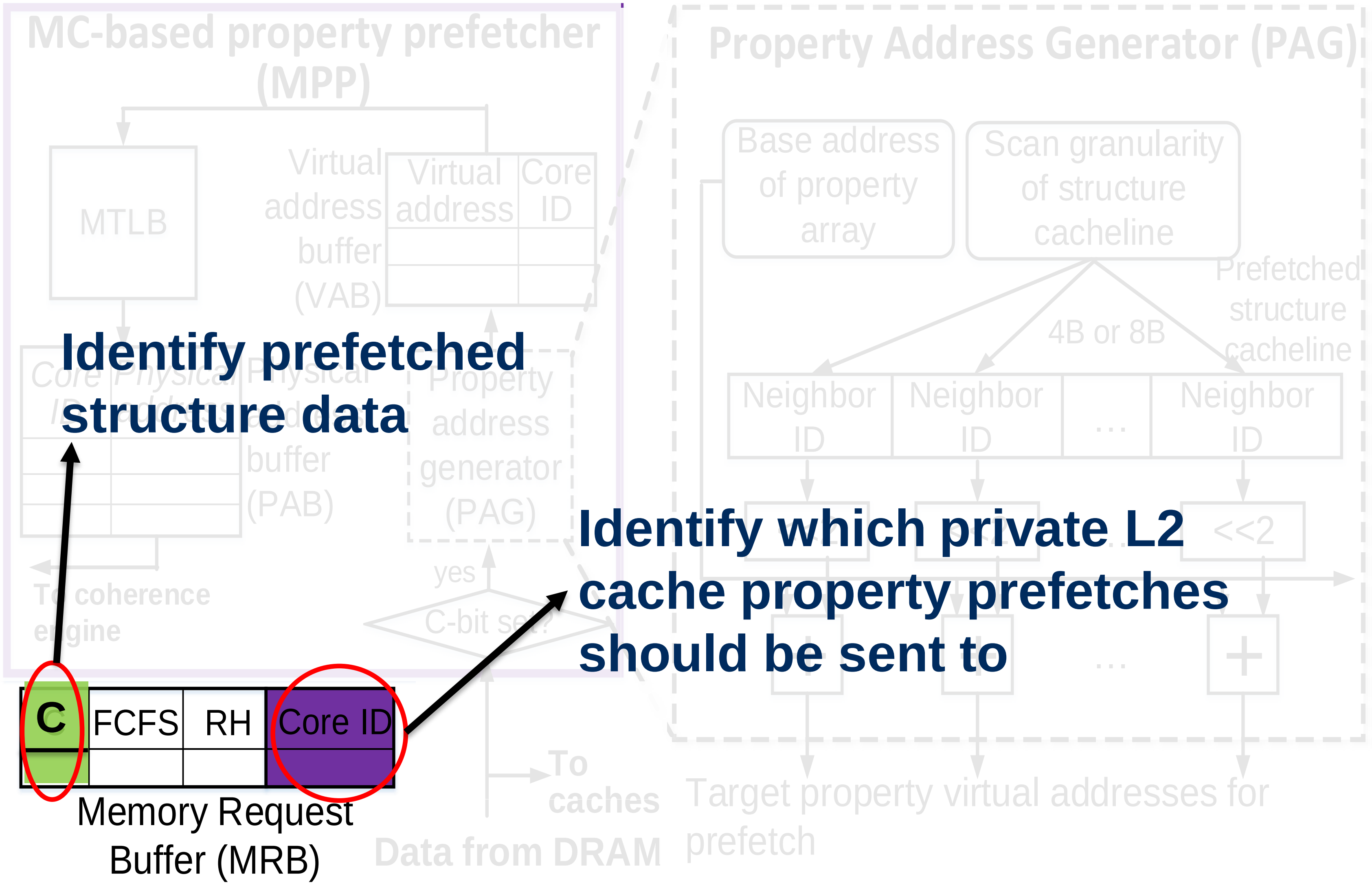
Backup Slides

Property data benefits from LLC capacity



With increasing LLC capacity, most reduction in DRAM accesses comes from property data

Property Prefetcher


MC-based property prefetcher (MPP)
MTLB
Virtual address buffer (VAB)
Virtual address
Core ID
Core ID
Physical address
Physical address buffer (PAB)
Property address generator (PAG)
To coherence engine
yes
C-bit set?
C
FCFS
RH
Core ID
Memory Request Buffer (MRB)
To caches
Data from DRAM
Property Address Generator (PAG)
Base address of property array
Scan granularity of structure cacheline
4B or 8B
Prefetched structure cacheline
Neighbor ID
Neighbor ID
…
Neighbor ID
<<2
+
…
+
Target property virtual addresses for prefetch

Identify prefetched structure data

Identify which private L2 cache property prefetches should be sent to

Hardware Overhead

- **Extra bits in TLB**: 1.56% storage overhead in paging structure
- **Extra bits in L2 request queue**: 1.54% storage overhead
- **Property prefetcher in MC**: 0.0348% area overhead compared to entire chip