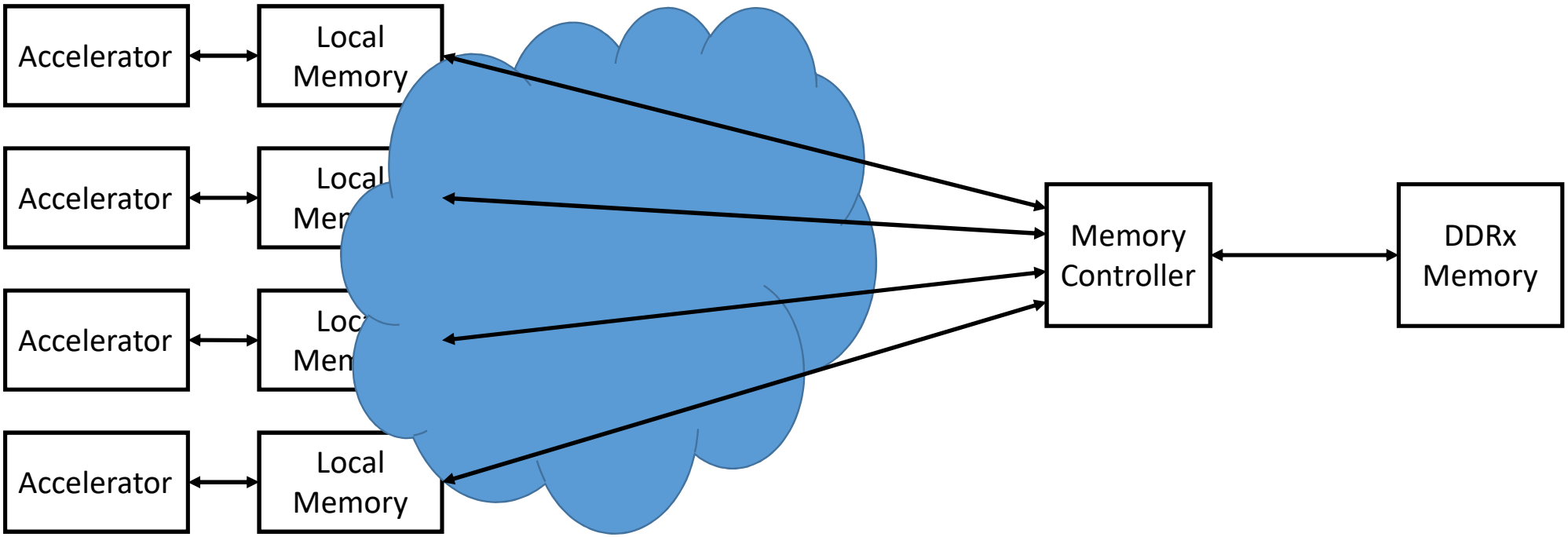
DynaBurst: Dynamically Assemblying DRAM Bursts over a Multitude of Random Accesses

Mikhail Asiatici and Paolo Ienne

Processor Architecture Laboratory (LAP)
School of Computer and Communication Sciences
EPFL

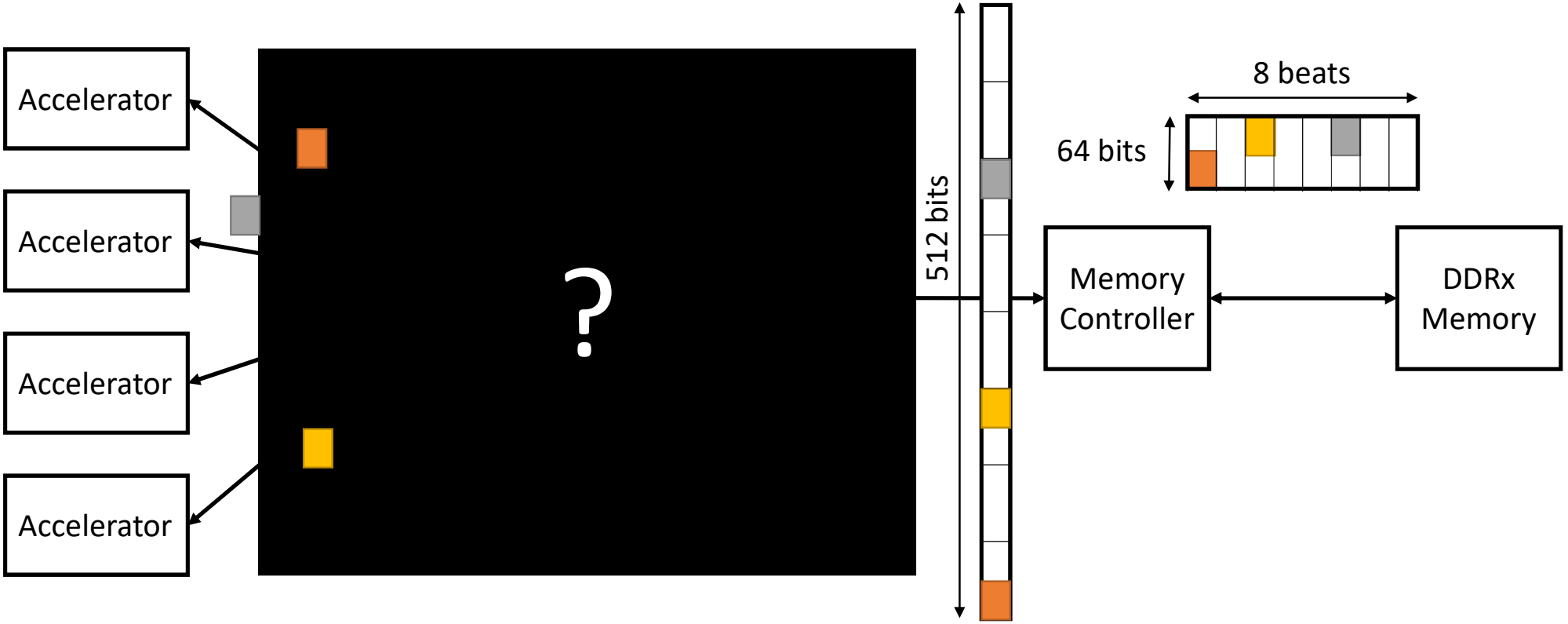
FPL 2019, Barcelona, Spain
11 September 2019

Motivation



Read accesses: **regular**, **predictable local reuse**

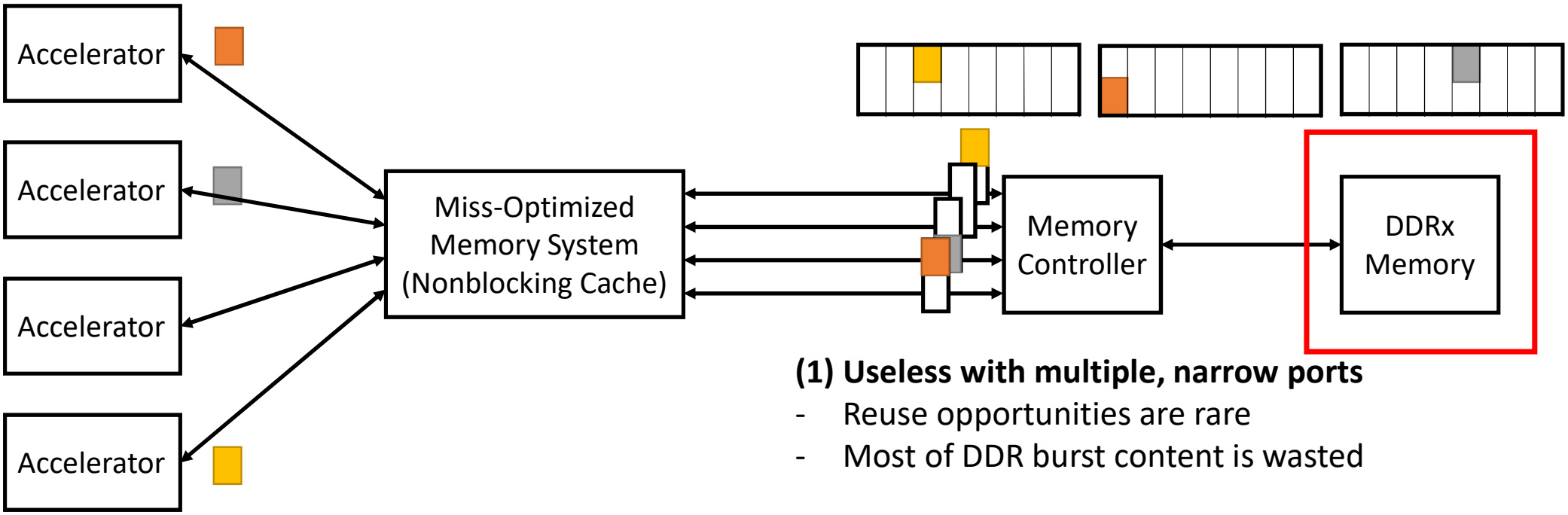
Motivation



Read accesses: **irregular, short, pattern unknown at compile time**

M. Asiatici and P. Ienne, "Stop Crying Over Your Cache Miss Rate: Handling Efficiently Thousands of Outstanding Misses in FPGAs" ISFPGA 2019

Limitations of Prior Work



(1) Useless with multiple, narrow ports

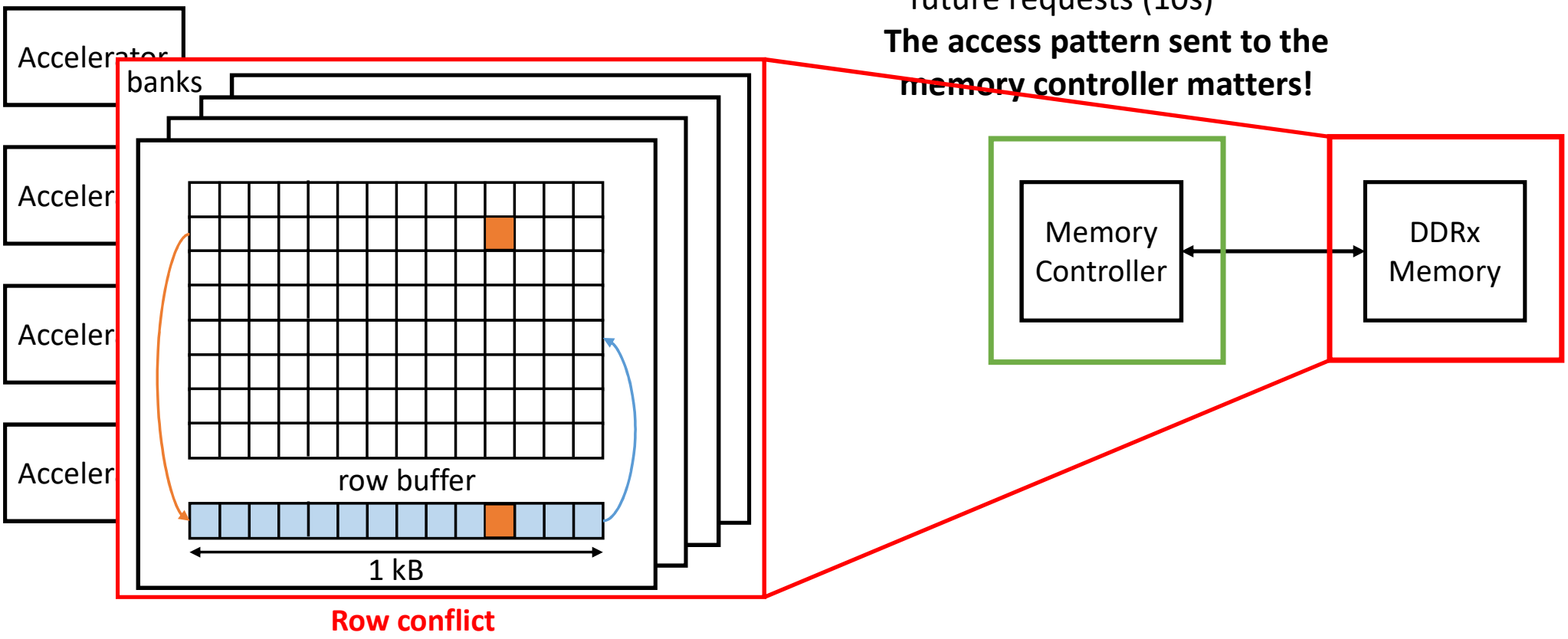
- Reuse opportunities are rare
- Most of DDR burst content is wasted

M. Asiatici and P. Ienne, “Stop Crying Over Your Cache Miss Rate: Handling Efficiently Thousands of Outstanding Misses in FPGAs” ISFPGA 2019

Limitations of Prior Work

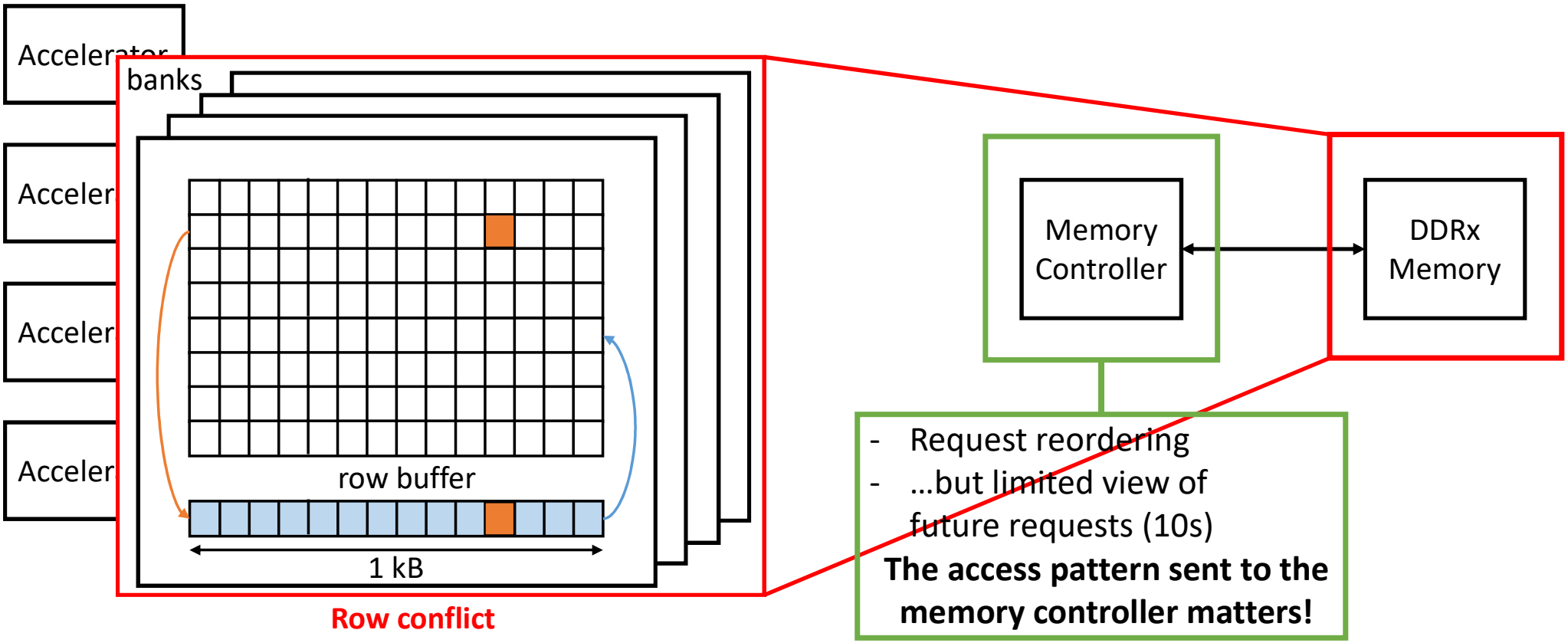
- Request reordering
- …but limited view of future requests (10s)

The access pattern sent to the memory controller matters!



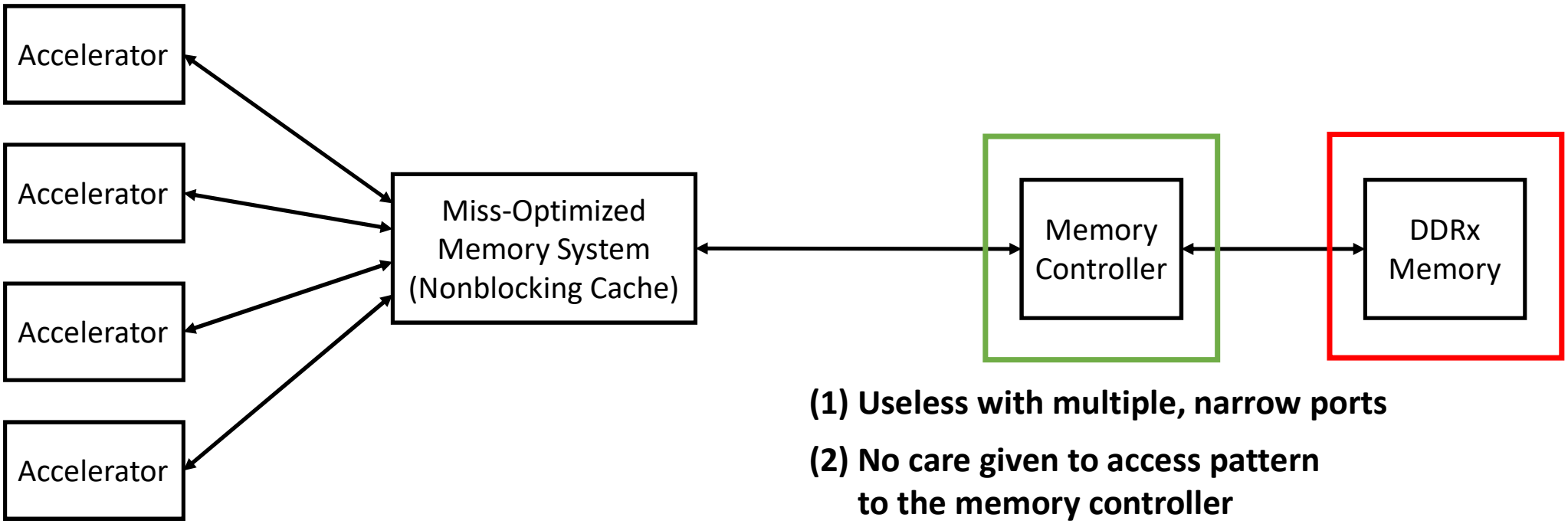
M. Asiatici and P. Ienne, “Stop Crying Over Your Cache Miss Rate: Handling Efficiently Thousands of Outstanding Misses in FPGAs” ISFPGA 2019

Limitations of Prior Work



M. Asiatici and P. Ienne, “Stop Crying Over Your Cache Miss Rate: Handling Efficiently Thousands of Outstanding Misses in FPGAs” ISFPGA 2019

Limitations of Prior Work



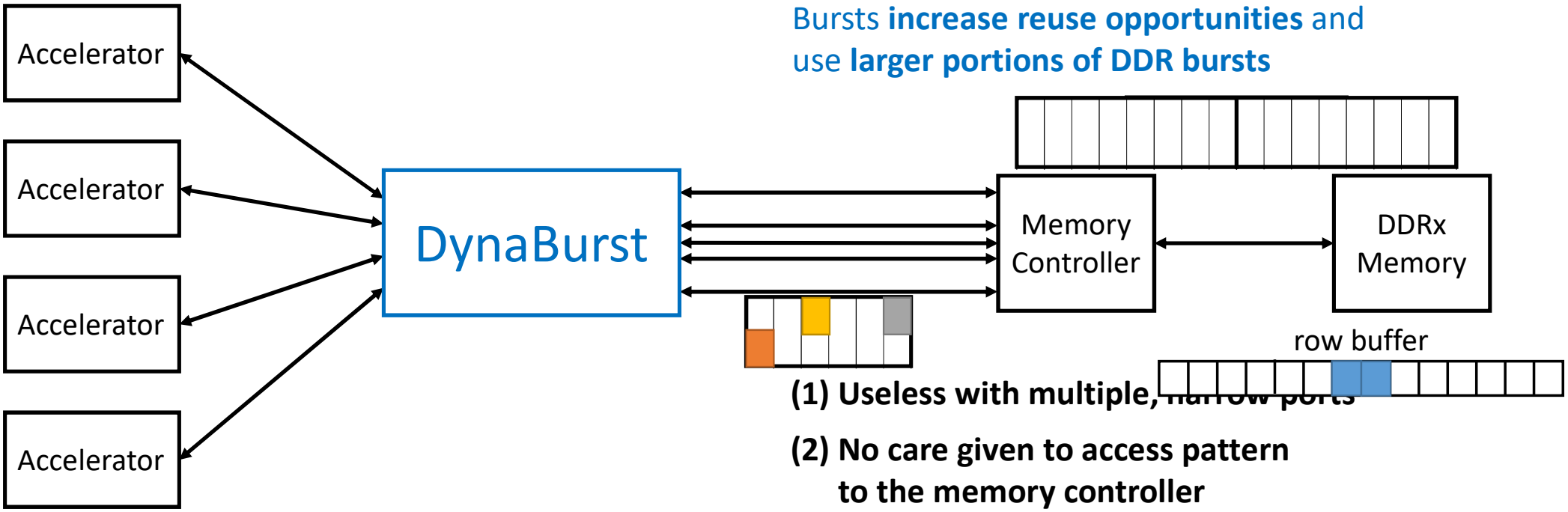
(1) Useless with multiple, narrow ports

(2) No care given to access pattern to the memory controller

→ Up to **60% of bandwidth lost** to row conflicts

M. Asiatici and P. Ienne, "Stop Crying Over Your Cache Miss Rate: Handling Efficiently Thousands of Outstanding Misses in FPGAs" ISFPGA 2019

Key Idea: Bursts of Memory Requests



Bursts **increase reuse opportunities** and use **larger portions of DDR bursts**

(1) Useless with multiple, narrow ports

(2) No care given to access pattern to the memory controller

Access pattern becomes **locally sequential** and we make use of **larger portions of DDR rows, increasing available bandwidth**

Outline

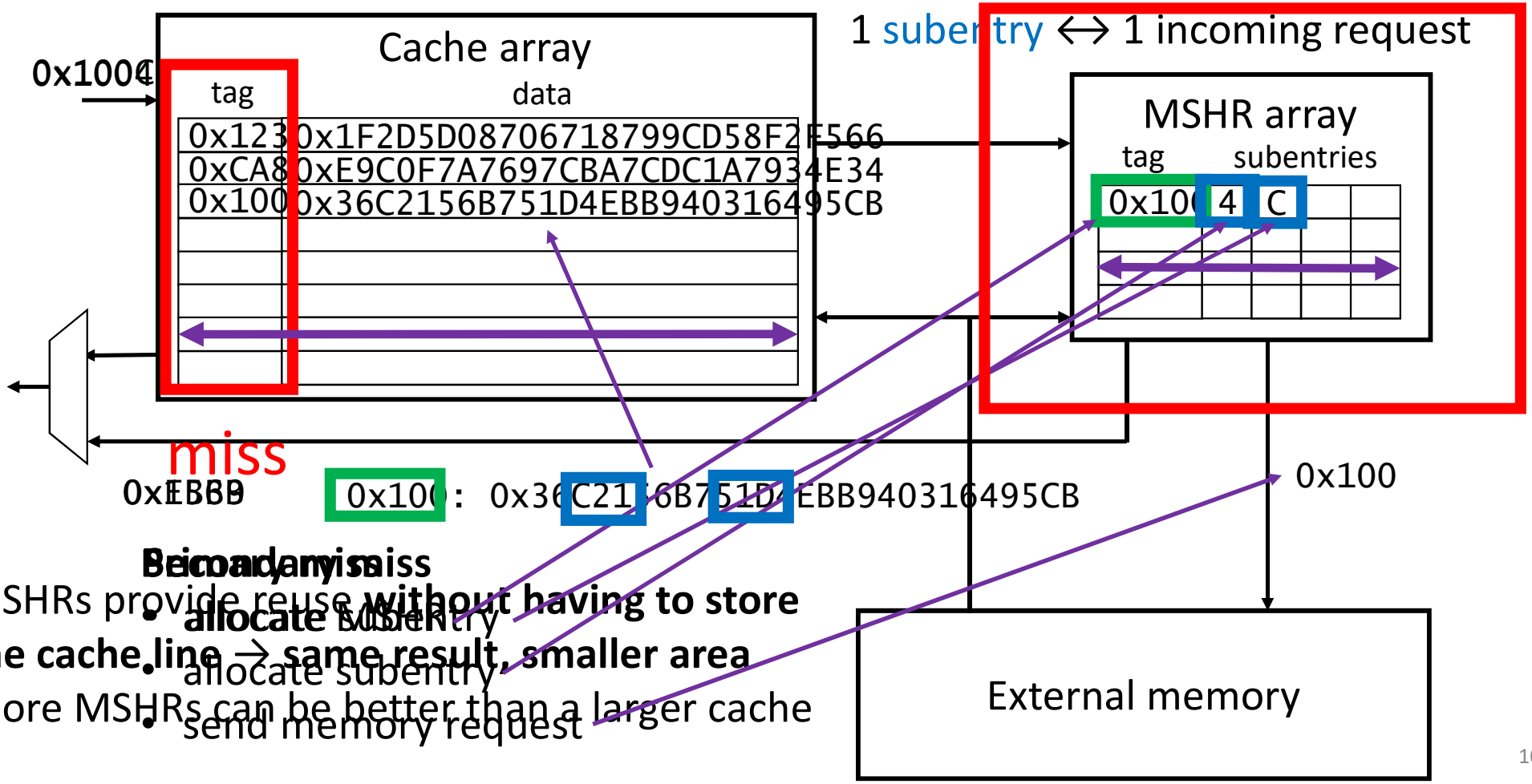
- Nonblocking Caches and Miss-Optimized Memory Systems
- Top-Level Architecture
- Handling Bursts
- Experimental Setup
- Results
- Conclusion

Nonblocking Caches

MSHR = Miss Status Holding Register

1 MSHR ↔ 1 memory request = 1 cache line

1 subentry ↔ 1 incoming request



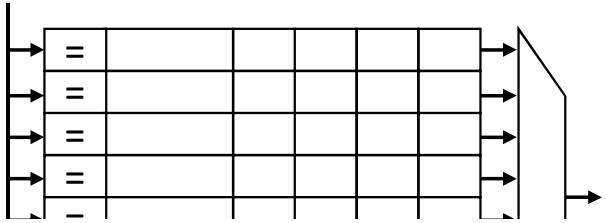
- MSHRs provide reuse **without having to store the cache line** → **same result, smaller area**
- More MSHRs can be better than a larger cache

Primary miss
- allocate MSHR
- allocate subentry
- send memory request

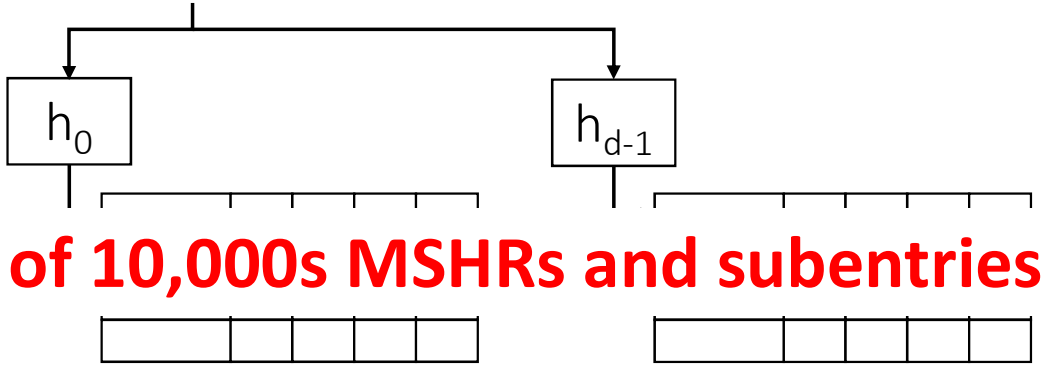
Secondary miss
- allocate subentry

Scaling Up Miss Handling

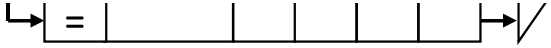
Traditional nonblocking caches



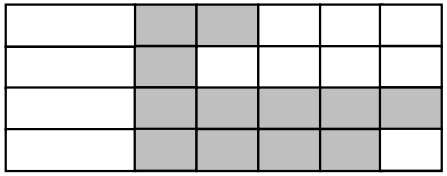
Miss-Optimized Memory System [1]



→ efficient storage and lookup of 10,000s MSHRs and subentries

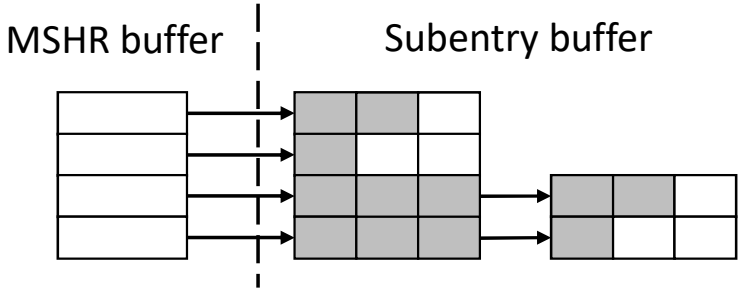


Fully-associative array



Subentry slots statically assigned to MSHRs

Cuckoo hash tables in BRAMs

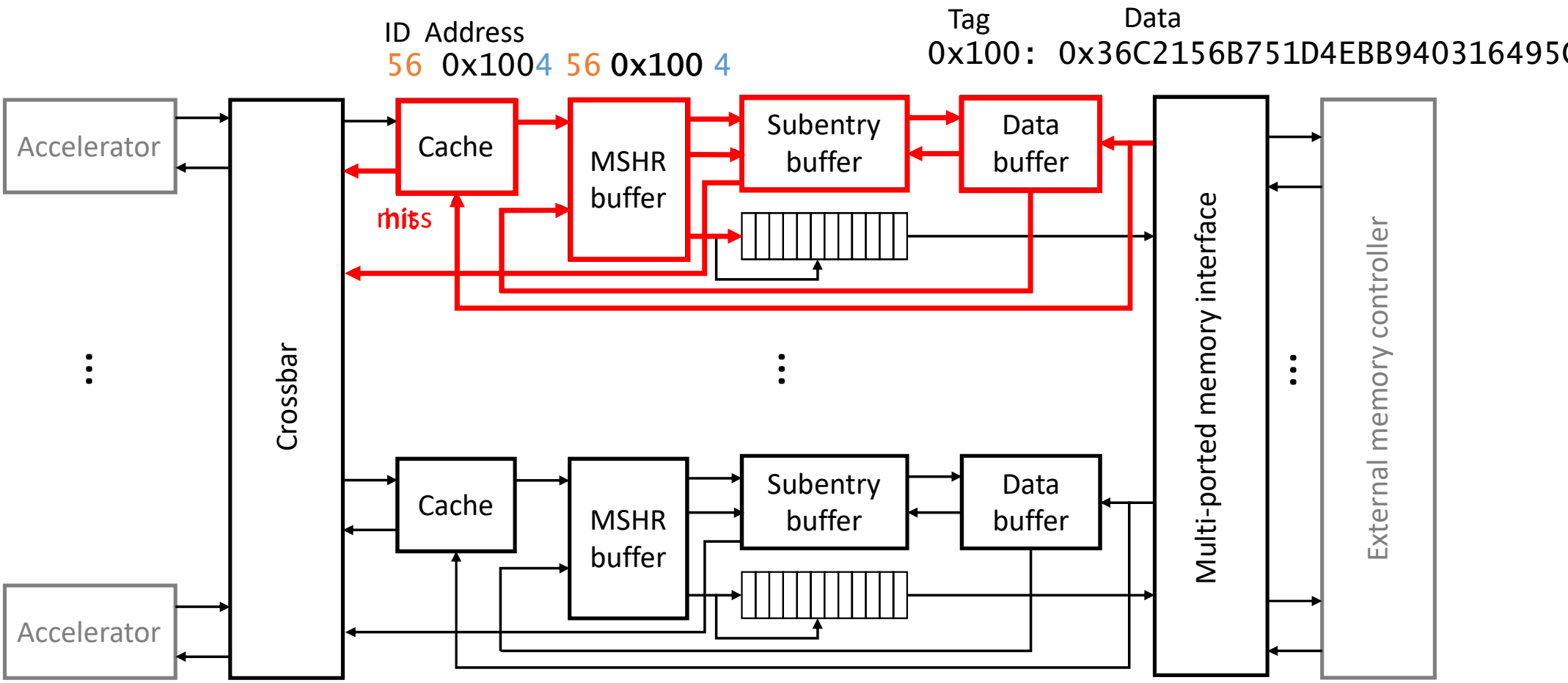


Dynamic subentry allocation

Outline

- Nonblocking Caches and Miss-Optimized Memory System
- Top-Level Architecture
- Handling Bursts
- Experimental Setup
- Results
- Conclusion

Top-Level Architecture

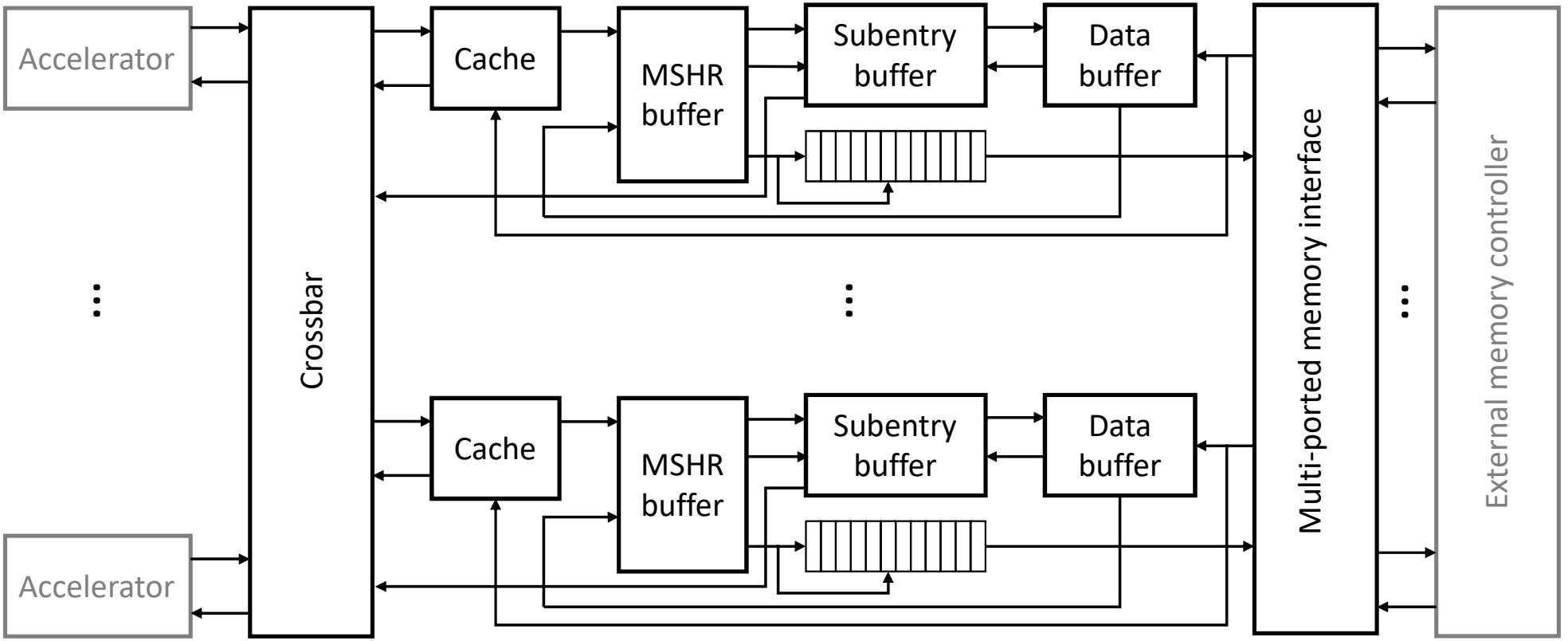


What's New in DynaBurst

Variable-length bursts

Flip-flops → LUTRAM, BRAM

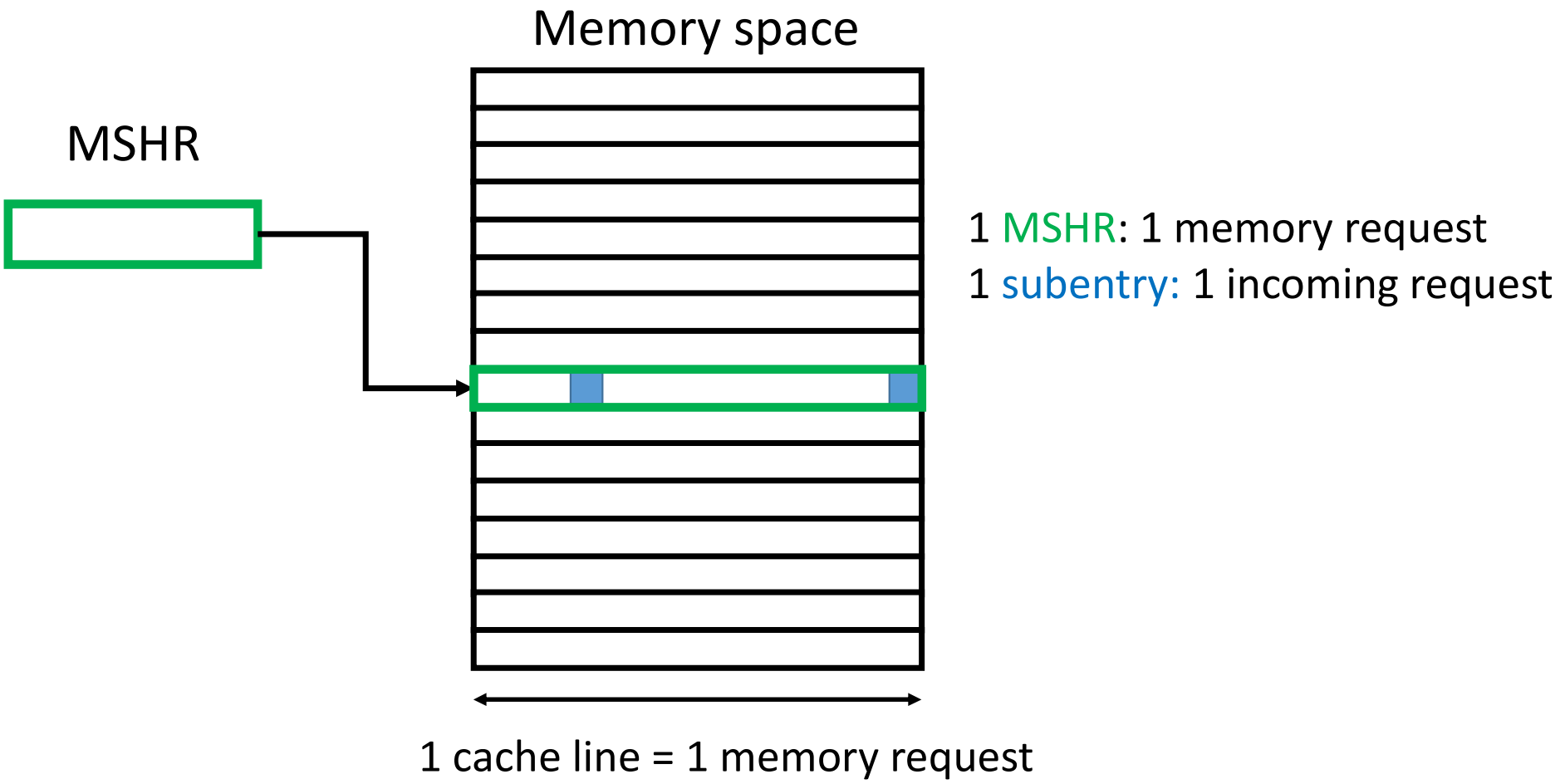
Multi-ported



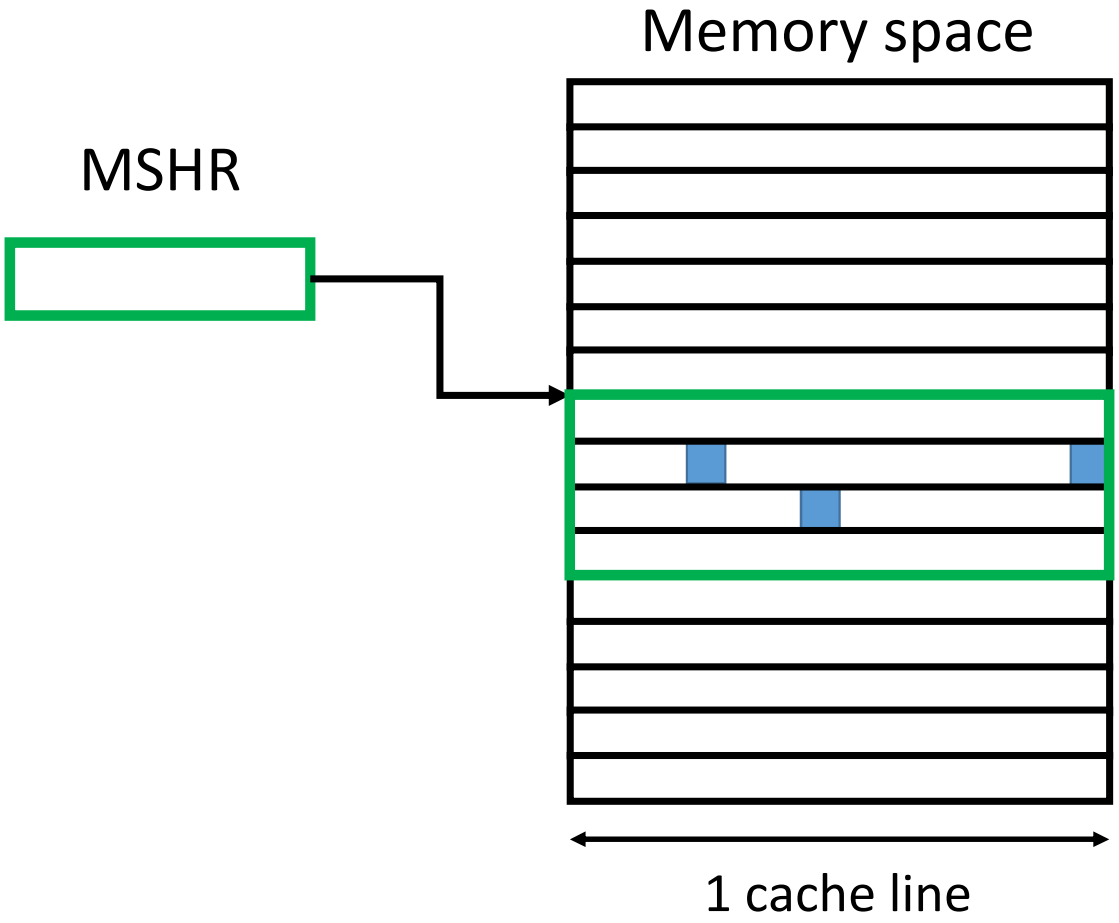
Outline

- Nonblocking Caches and Miss-Optimized Memory System
- Top-Level Architecture
- Handling Bursts
- Experimental Setup
- Results
- Conclusion

From Single Requests to Bursts



From Single Requests to Bursts

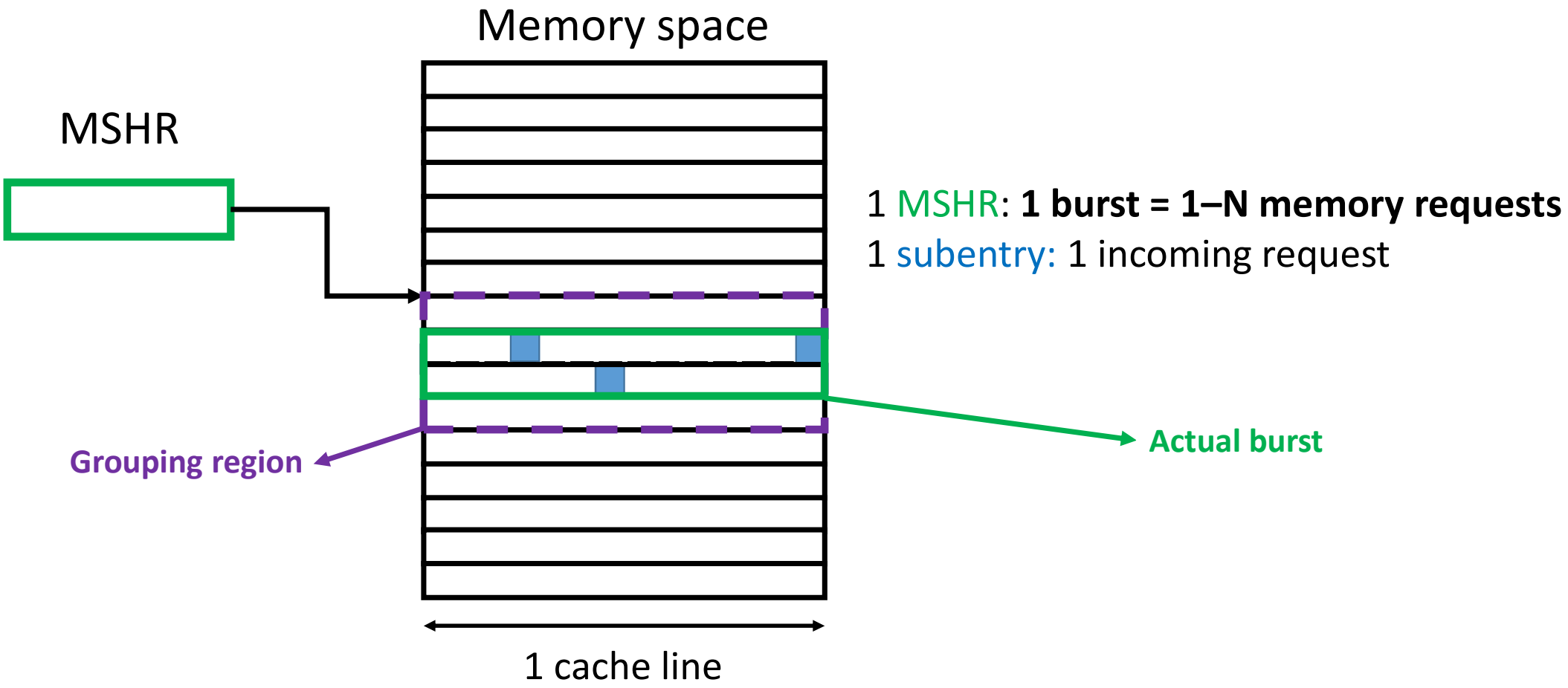


1 MSHR: **1 burst = N memory requests**

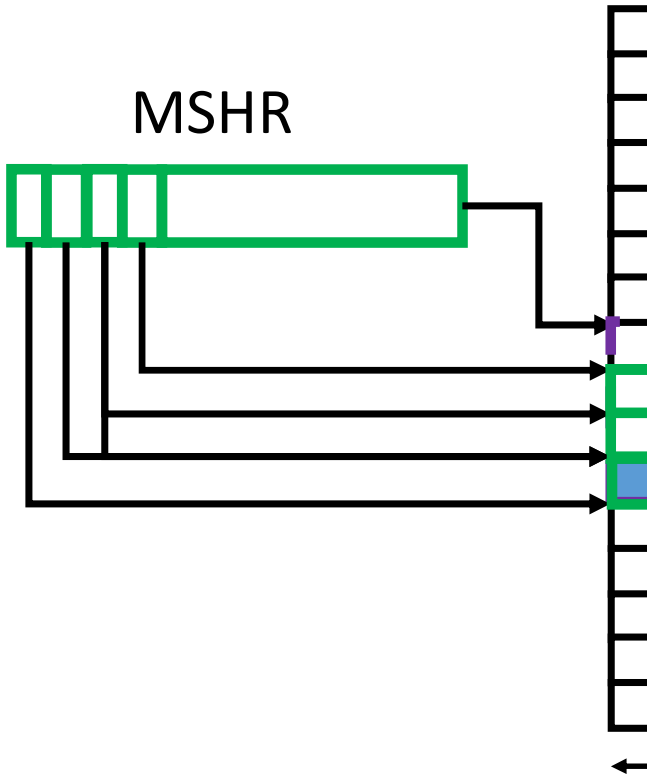
1 subentry: 1 incoming request

Problem: Data wastage

From Single Requests to Bursts



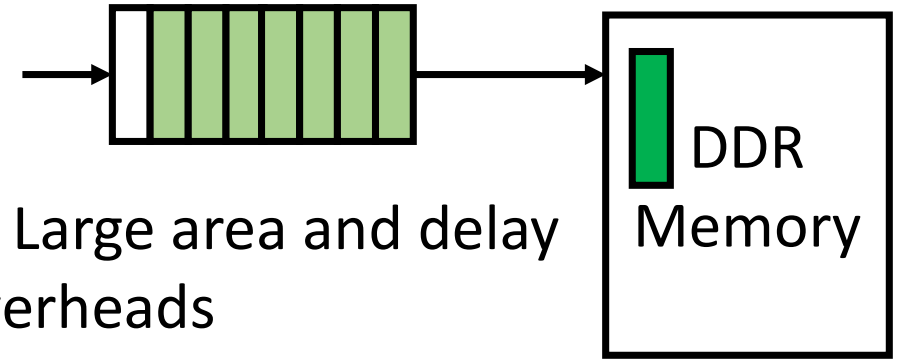
From Single F…rsts



On a new request:
1) MSHR exists?
2a) If yes: request covered by current burst 0 bounds?
3a) If not: burst 0 still in the output queue?
2b) Is burst 1 valid?
3b) If yes: request covered by current burst 1 bounds?
4b) If not: burst 1 still in the output queue?
2c) Is burst 2 valid?
3c) If yes: request covered by current burst 2 bounds?
4c) If not: burst 2 still in the output queue?
…

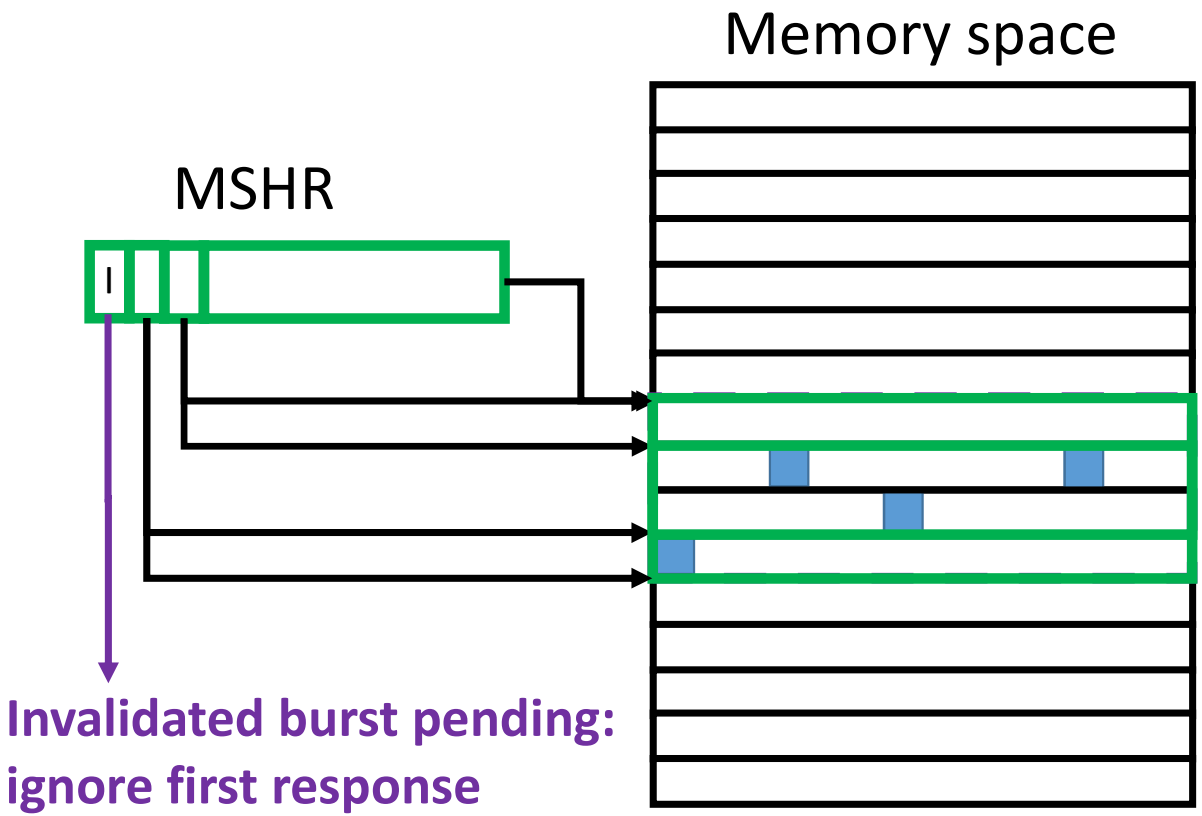
1 MSHR: **1 burst = 1–N memory requests**
1 subentry: 1 incoming request

Output request queue



→ Large area and delay overheads

Burst Invalidations



Burst updatable during $\frac{N_Q}{N_Q + N_{mem}}$ of its lifetime

Capacities:

Queue: ≈ 1,000 – 10,000

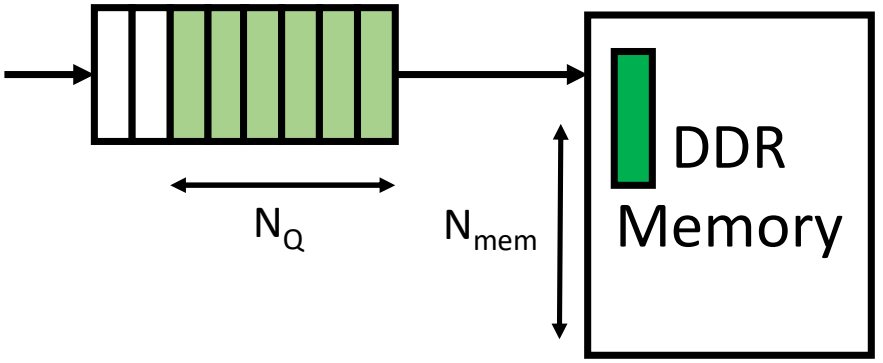
Memory pipeline: ~100

MSHR: 1 – 10 memory requests

1 subentry: 1 incoming request

$\frac{N_Q}{N_Q + N_{mem}} \approx 1$ (see paper)

Output request queue



Outline

- Nonblocking Caches and Miss-Optimized Memory System
- Top-Level Architecture
- Handling Bursts
- Experimental Setup
- Results
- Conclusion

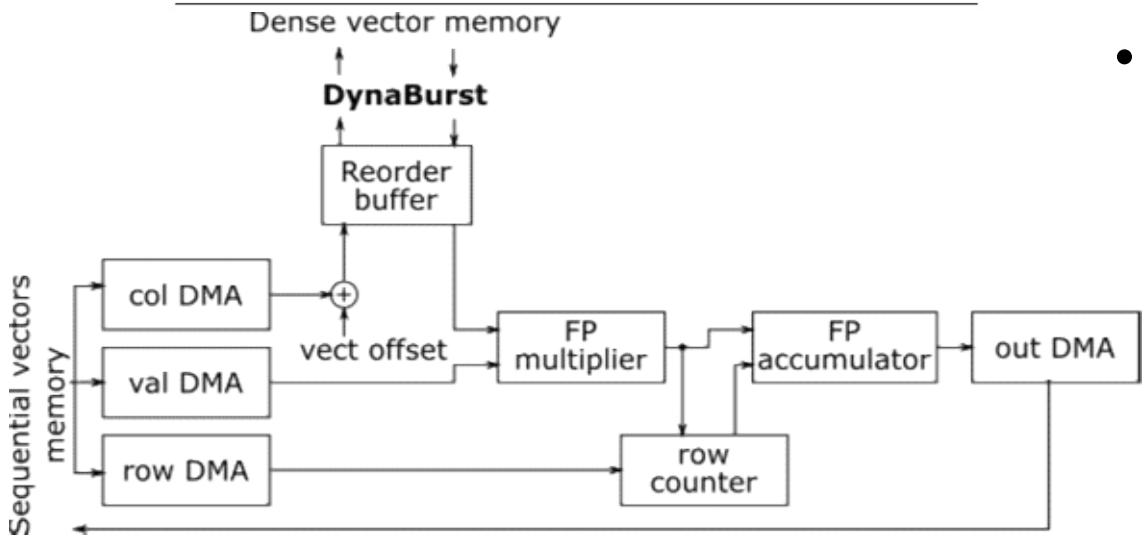
Board: Xilinx ZC706

- XC7Z045 Zynq-7000 FPGA
- 1 GB of DDR3 on processing system (PS) side
  - 3.9 GB/s through **four 64-bit ports** at 150 MHz
- 1 GB of DDR3 on programmable logic (PL) side
  - 12.0 GB/s through **one 512-bit port** at 200 MHz

Accelerators: Compressed Sparse Row SpMV

Algorithm 1 Sparse matrix-vector multiplication (SpMV)

```
1: for r ← 0 to ROWS − 1 do
2:     out[r] ← 0
3:     for i ← idx[r] to idx[r + 1] do
4:         out[r] ← out[r] + val[i] × vect[col[i]]
5:     end for
6: end for
```



- 15 benchmarks from SuiteSparse
  - Include web, social, road network, and linear programming matrices
  - Single-precision floating point values
  - Vector size: 1.7-91 MB (BRAM limit: 2.39 MB)
- Why SpMV?
  - Representative of **latency-tolerant, bandwidth-bound** applications with **various degrees of locality**
  - **Important kernel** in many applications
  - Several sparse **graph algorithms** can be mapped to it

A. Ashari et al. "Fast Sparse Matrix-Vector Multiplication on GPUs for graph applications" SC 2014

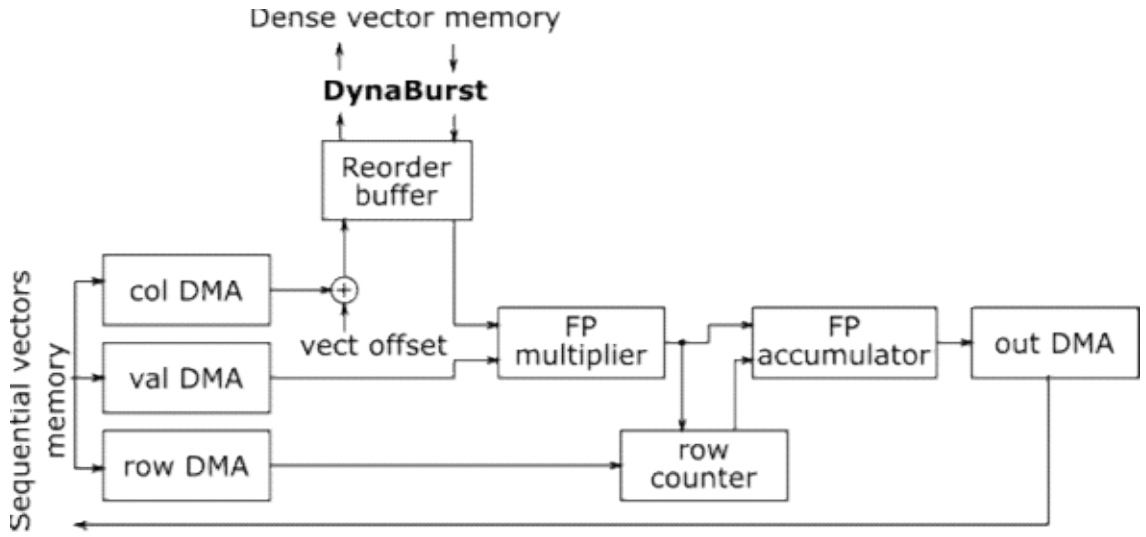
J. Kepner and J. Gilbert "Graph Algorithms in the Language of Linear Algebra" SIAM 2011

https://sparse.tamu.edu/

PL and PS Systems

High bandwidth, single wide port

PL DDR



PS DDR

Low bandwidth, 4 narrow ports

PL system

- Same as in our previous work
- 4 accelerators and banks
- 200 MHz

PS system

- 8 accelerators and banks
- 150 MHz

M. Asiatici and P. Ienne, “Stop Crying Over Your Cache Miss Rate: Handling Efficiently Thousands of Outstanding Misses in FPGAs” ISFPGA 2019

Design Space Exploration

	PL systems (4 banks)	PS systems (8 banks)
Total cache size (KB)	0, 128, 256, 512, 1024	0, 64, 128, 256, 512
Maximum burst length	2, 4, 8, 16	
Miss handling (6 subentries/row)		
- Small	2k MSHR, 12k subentries	4k MSHR, 24k subentries
- Medium	6k MSHR, 48k subentries	8k MSHR, 48k subentries
- Large	16k MSHR, 96k subentries	16k MSHR, 96k subentries

- Baselines: other generic memory systems for irregular access pattern
  - Our prior work (single-request)
    - Each design point compared to **same cache and miss handling configuration**
  - Traditional nonblocking cache with associative MSHRs
    - 16 MSHRs + 8 subentries each, per bank
    - Each design point compared to traditional cache with **closest BRAM utilization**

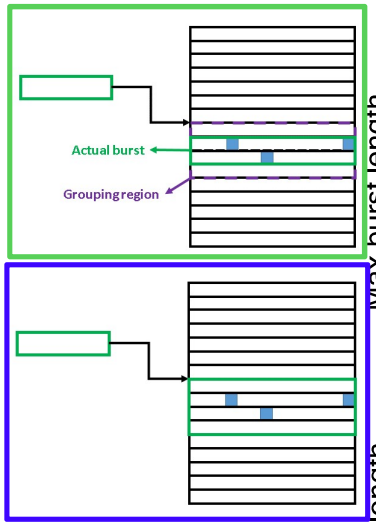
M. Asiatici and P. Ienne, "Stop Crying Over Your Cache Miss Rate: Handling Efficiently Thousands of Outstanding Misses in FPGAs" ISFPGA 2019

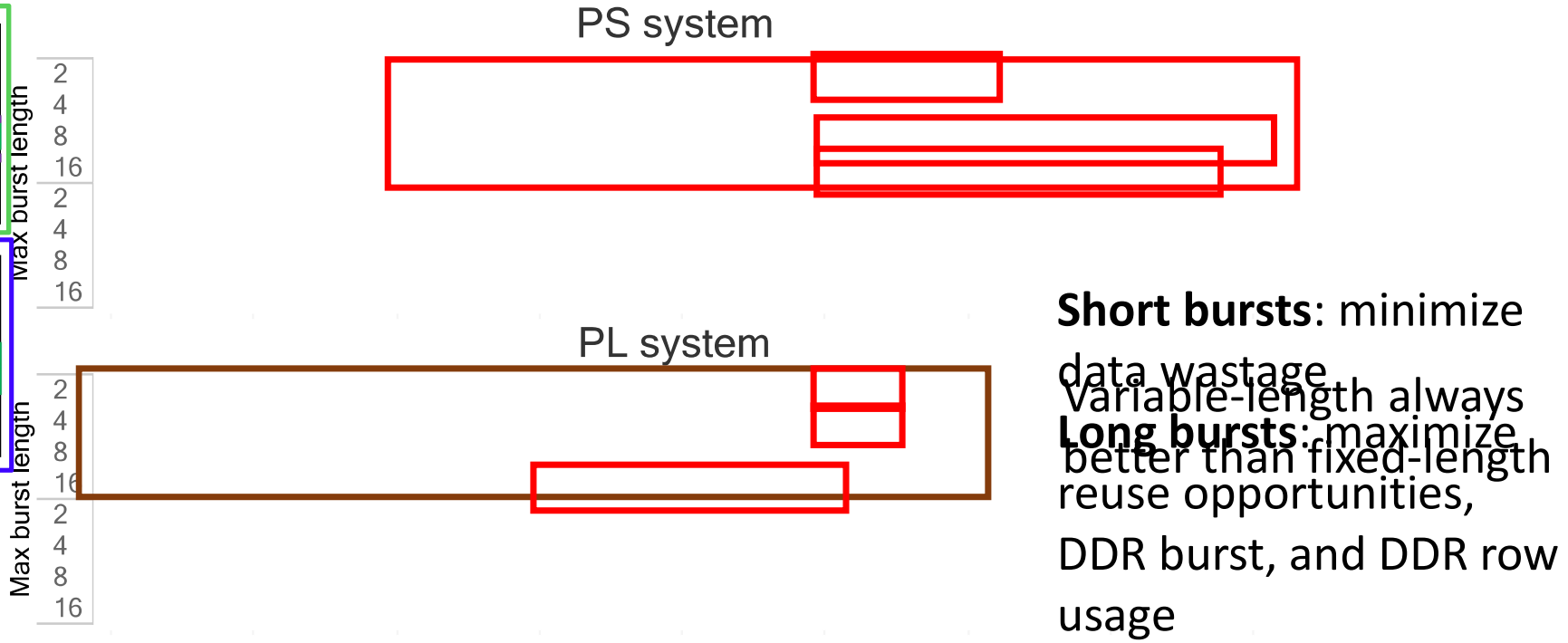
Outline

- Nonblocking Caches and Miss-Optimized Memory System
- Top-Level Architecture
- Handling Bursts
- Experimental Setup
- Results
- Conclusion

Impact of Maximum Burst Length and of Burst Trimming

Bursts much more useful on PS system





Some benefit on specific design points on PL system

Short bursts: minimize data wastage

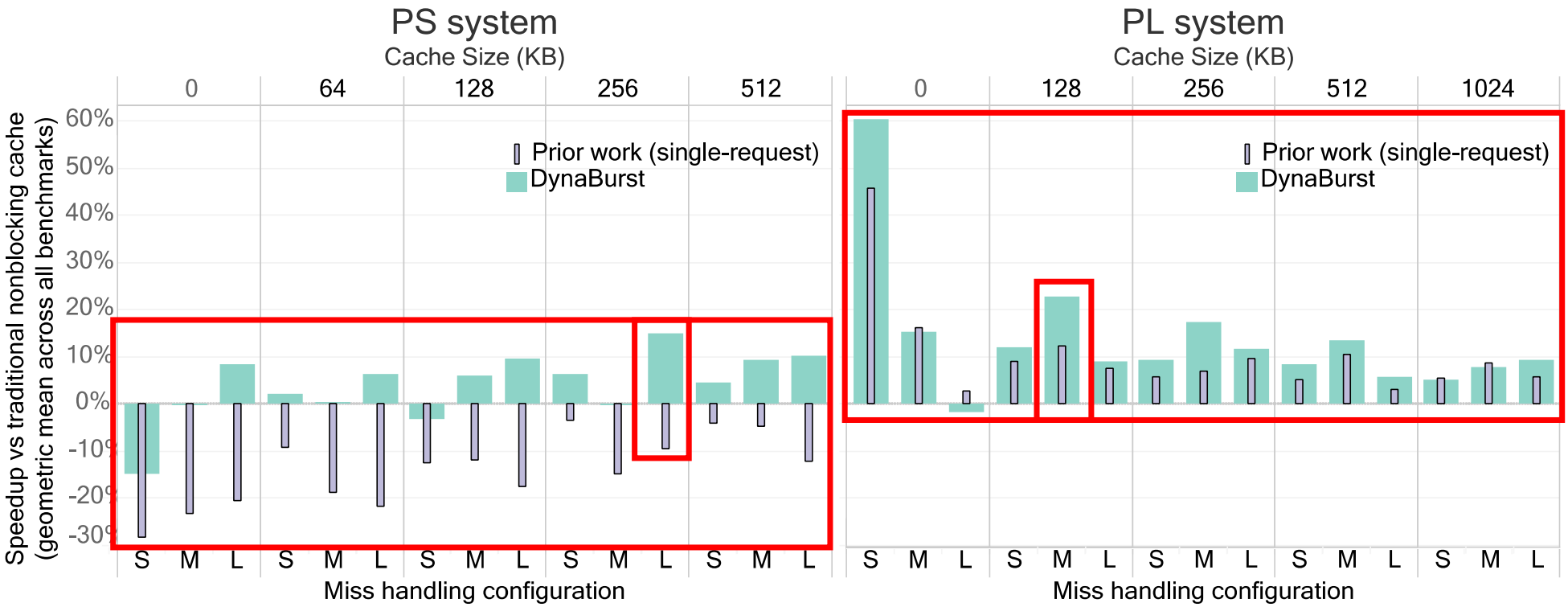
Long bursts: maximize reuse opportunities, DDR burst, and DDR row usage

Variable-length always better than fixed-length

Optimum is somewhere in the middle

We'll now consider only best max burst lengths

Cache and Miss Handling Size Exploration

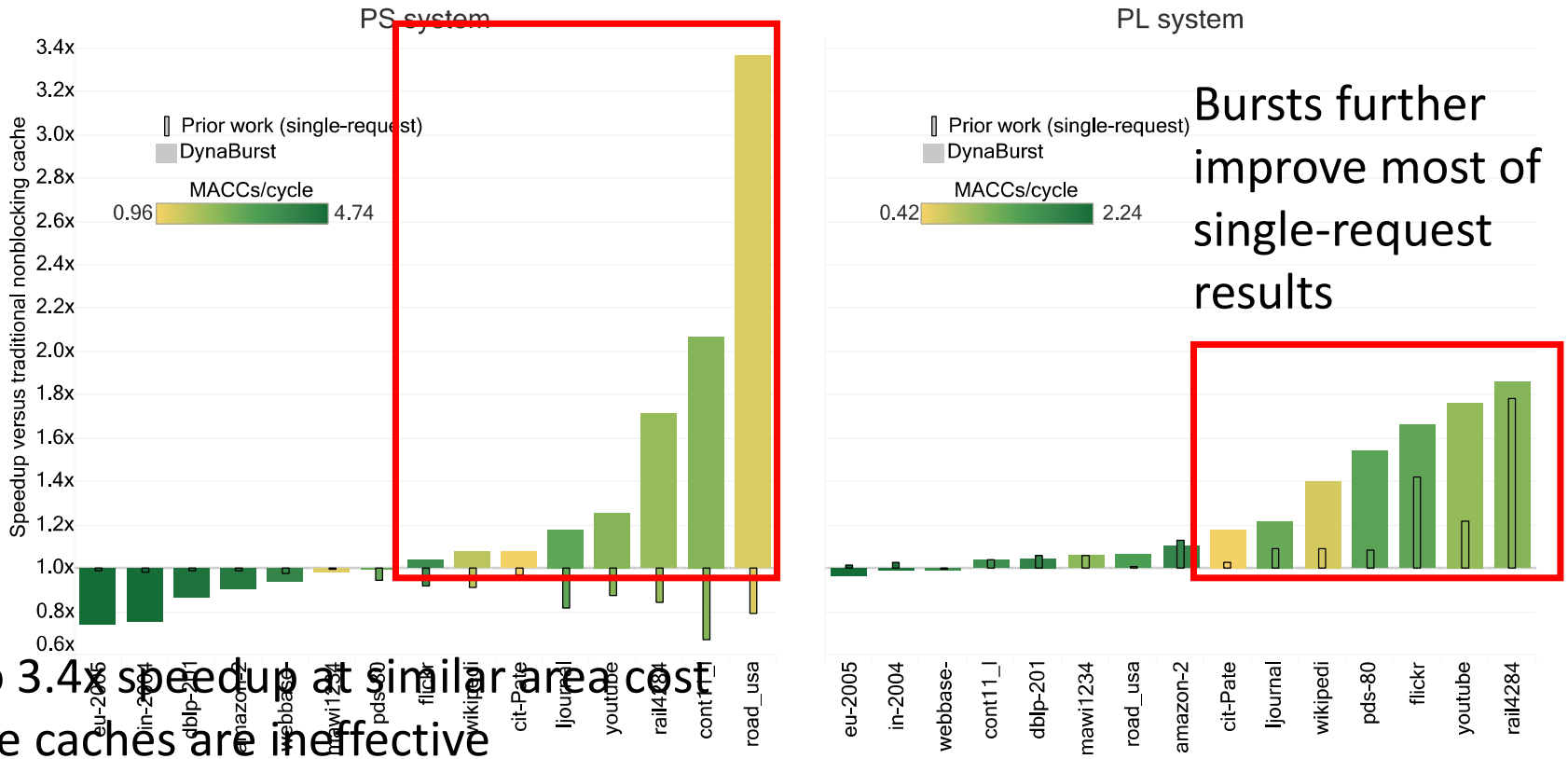


Bursts required to make miss-optimized memory systems cost-effective

Bursts further improve performance on most of the design points

We'll now look into these points

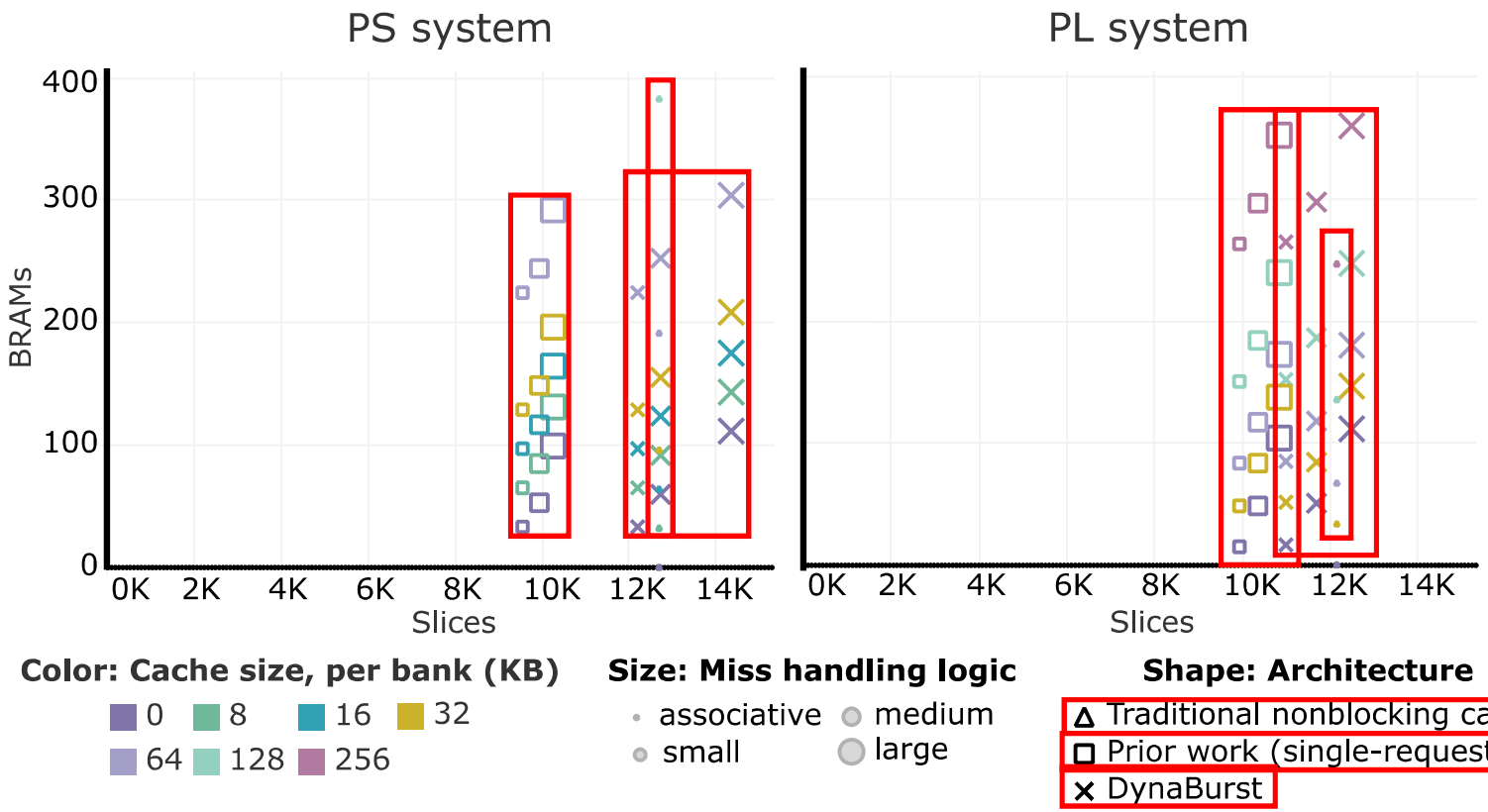
Speedup on Individual Benchmarks



Up to 3.4x speedup at similar area cost, where caches are ineffective

Most effective where traditional cache performance was lower

Resource Utilization



Overhead vs single-request

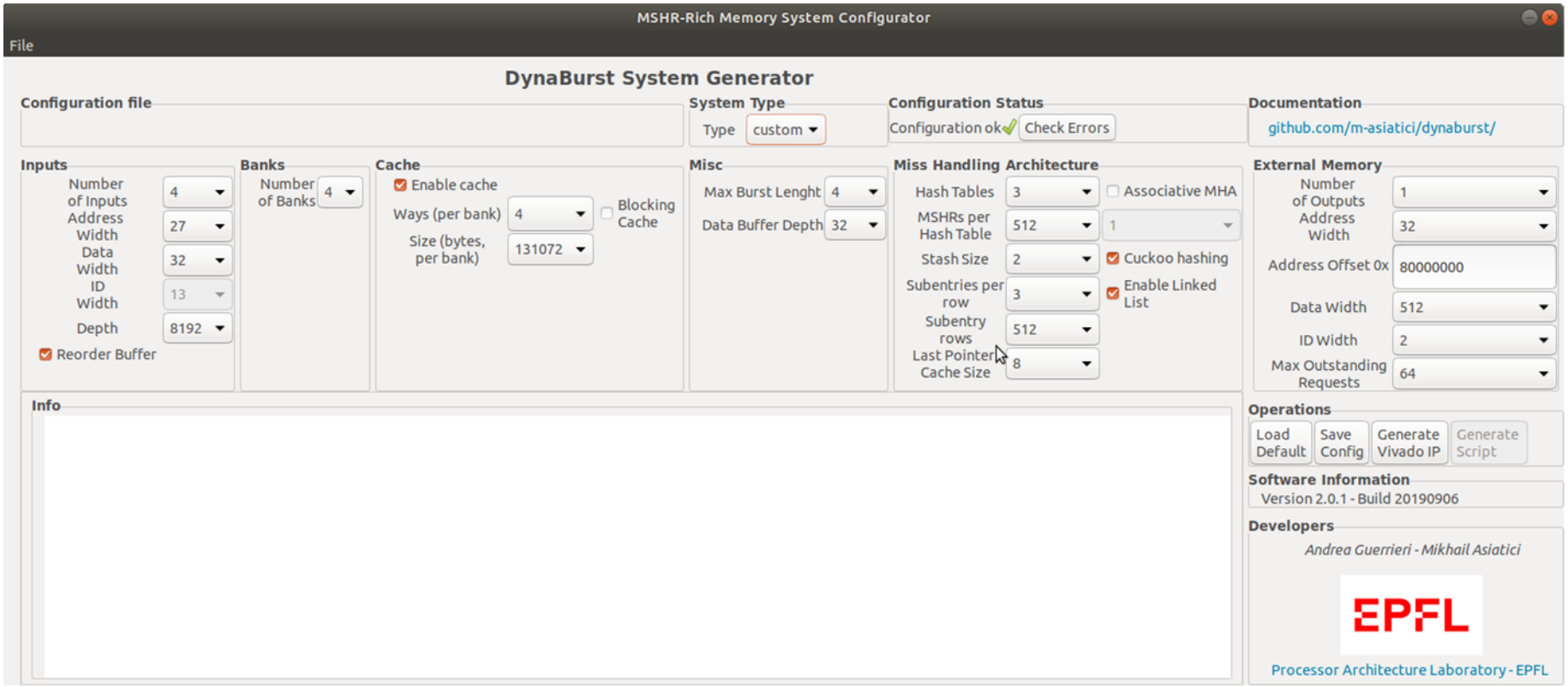
	BRAM	Slices
PS system	0 – 15 %	30 – 40 %
PL system	2 – 15 %	10 – 15 %

Conclusion

- Caches are not effective when read accesses are **irregular** and **short**
- **Single-request miss-optimized memory systems**
  - Reuse same memory request among multiple incoming requests
  - Useful **only when memory controllers have wide ports**
- **DynaBurst**
  - Merges incoming requests into **bursts** of memory requests
  - Controller with narrow ports: up to **3.4x speedup** compared to cache with **similar area**
  - Controller with wide port: up to **2.4x speedup with < 15% area overhead** compared to prior work

Thank you!

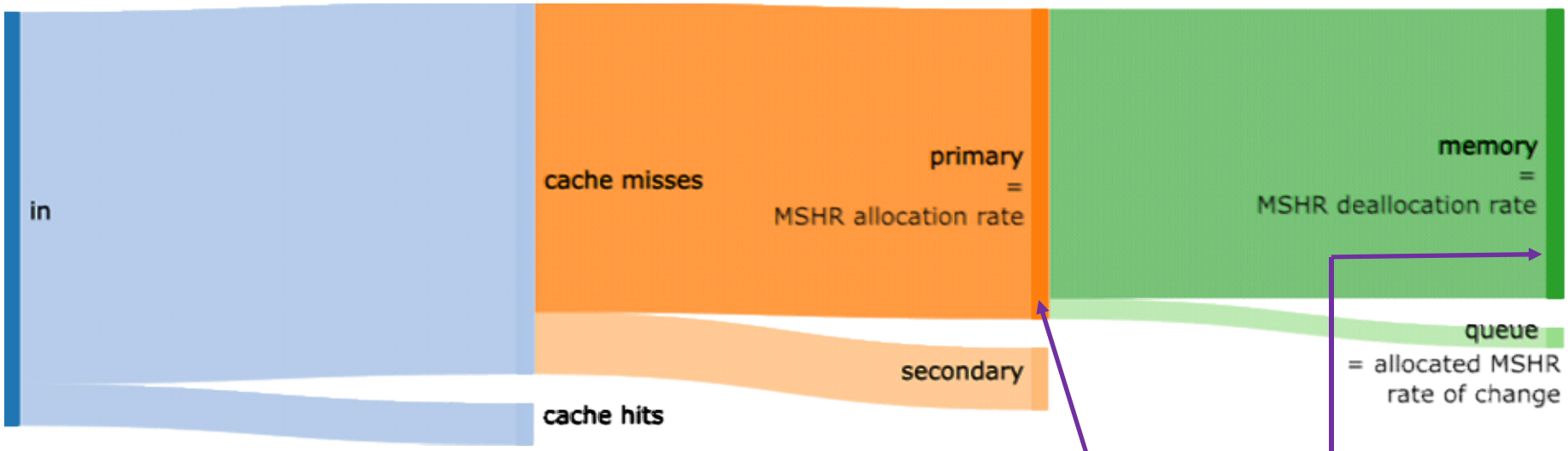
https://github.com/m-asiatici/dynaburst



Backup

Filling the Output Queue

Memory bound → in > memory

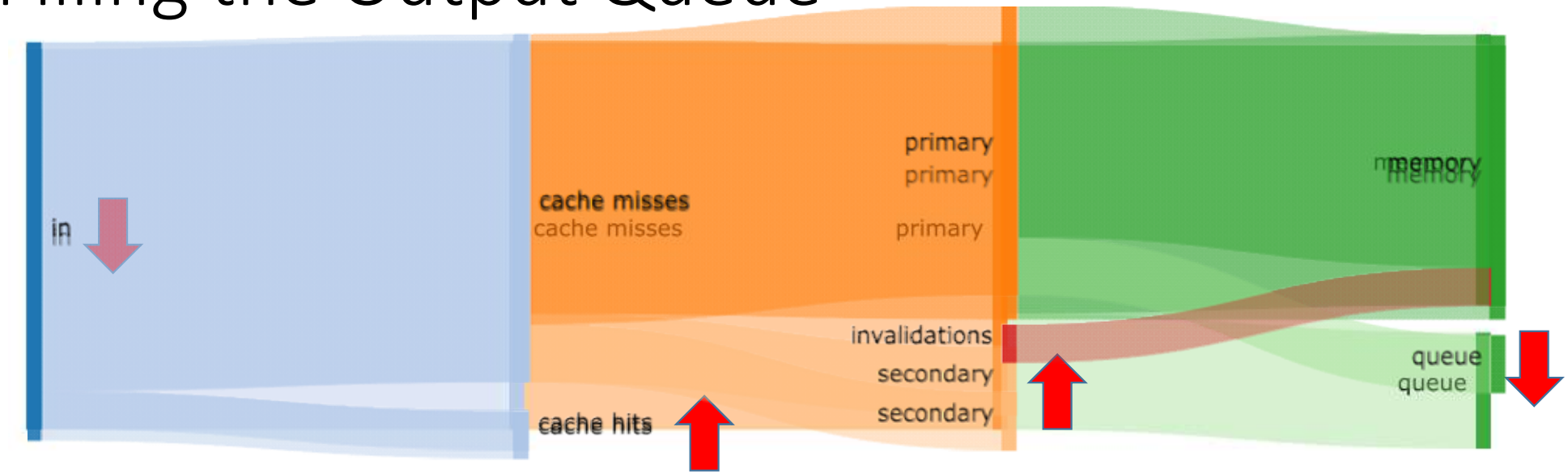


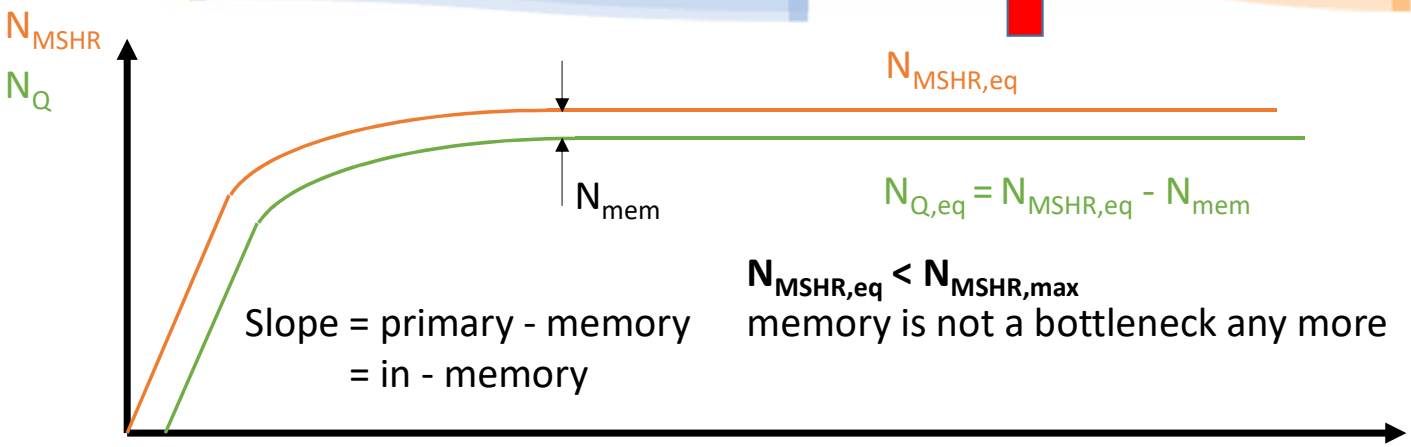
MSHRs:

- Allocated on primary miss
- Deallocated on memory response

Memory response rate = memory request rate

Filling the Output Queue





Slope = primary - memory
= in - memory

$N_{MSHR,eq} < N_{MSHR,max}$
memory is not a bottleneck any more

$N_{MSHR,eq} > N_{MSHR,max}$
- Stalls decrease incoming rate until $N_{MSHR,eq} = N_{MSHR,max}$
- $N_{Q,eq} = N_{MSHR,max} - N_{mem}$

Probability that a burst can be updated:

$$\frac{N_Q}{N_Q + N_{mem}} = \frac{N_{MSHR,max} - N_{mem}}{N_{MSHR,max}} \approx 1$$

$(N_{MSHR,max} \gg N_{mem})$

Invalidations and Burst Usage