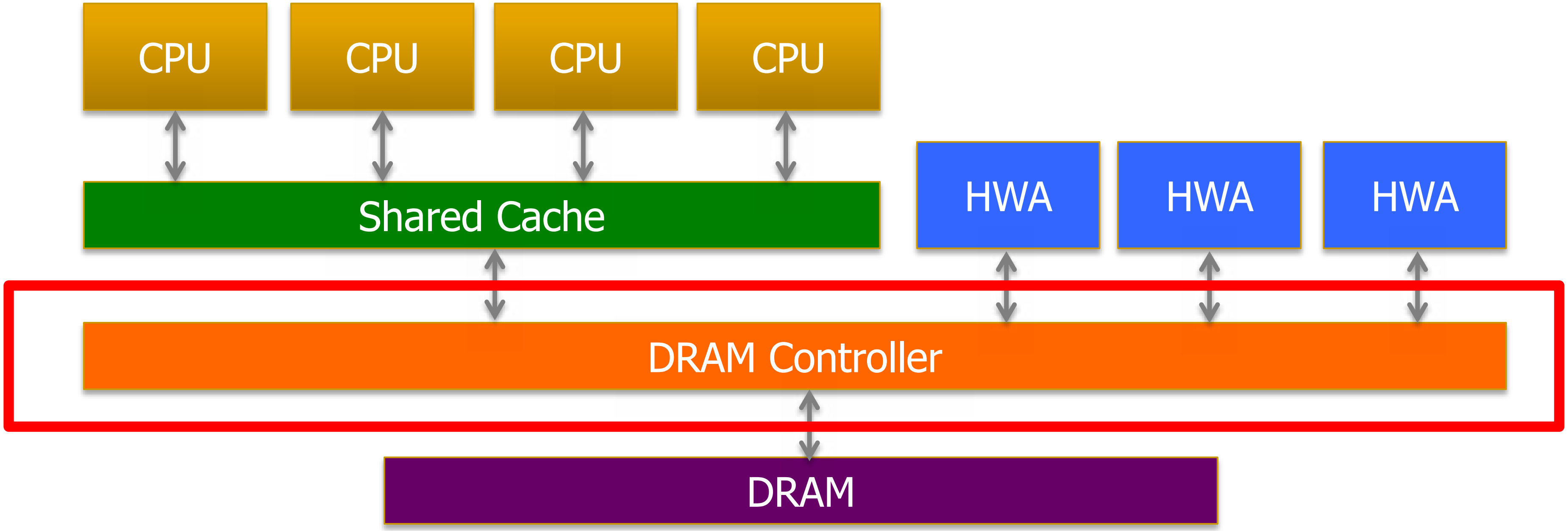
DASH: Deadline-Aware High-Performance Memory Scheduler for Heterogeneous Systems with Hardware Accelerators

Hiroyuki Usui, Lavanya Subramanian
Kevin Chang, Onur Mutlu

DASH source code is available at GitHub
https://github.com/CMU-SAFARI/HWASim

Carnegie Mellon

Current SoC Architectures



- Heterogeneous agents: CPUs and HWAs
  - HWA : Hardware Accelerator
- Main memory is shared by CPUs and HWAs → Interference

How to schedule memory requests from CPUs and HWAs to mitigate interference?

DASH Scheduler: Executive Summary

- Problem: Hardware accelerators (HWAs) and CPUs share the same memory subsystem and interfere with each other in main memory
- Goal: Design a memory scheduler that improves CPU performance while meeting HWAs' deadlines
- Challenge: Different HWAs have different memory access characteristics and different deadlines, which current schedulers do not smoothly handle
  - Memory-intensive and long-deadline HWAs significantly degrade CPU performance *when they become high priority* (due to slow progress)
  - Short-deadline HWAs sometimes miss their deadlines *despite high priority*
- Solution: DASH Memory Scheduler
  - Prioritize HWAs over CPU anytime when the HWA is not making good progress
  - Application-aware scheduling for CPUs and HWAs
- Key Results:

  1) Improves CPU performance for a wide variety of workloads by 9.5%

  2) Meets 100% deadline met ratio for HWAs
- DASH source code freely available on the GitHub

Outline

- Introduction
- Problem with Existing Memory Schedulers for Heterogeneous Systems
- DASH: Key Ideas
- DASH: Scheduling Policy
- Evaluation and Results
- Conclusion

Outline

- Introduction
- Problem with Existing Memory Schedulers for Heterogeneous Systems
- DASH: Key Ideas
- DASH: Scheduling Policy
- Evaluation and Results
- Conclusion

Existing QoS-Aware Scheduling Scheme

- **Dynamic Prioritization for a CPU-GPU System** [Jeong et al., DAC 2012]
  - Dynamically adjust GPU priority based on its progress
  - Lower GPU priority if GPU is making a good progress to achieve its target frame rate

- We apply this scheme for a wide variety of HWAs
  - Compare HWA's current progress against expected progress
    - Current Progress : $\frac{\text{(The number of finished memory requests for a period)}}{\text{(The number of total memory requests for a period)}}$
    - Expected Progress : $\frac{\text{(Elapsed cycles in a period)}}{\text{(Total cycles in a period)}}$
  - Every scheduling unit, dynamically adjust HWA priority
    - If Expected Progress > EmergentThreshold (=0.9) **: HWA > CPU**
    - If (Current Progress) > (Expected Progress) : **HWA < CPU**
    - If (Current Progress) <= (Expected Progress) : **HWA = CPU**

Problems in Dynamic Prioritization

- **Dynamic Prioritization for a CPU-HWA system**
  - Compares HWA's current progress against expected progress
    - Current Progress : $\frac{\text{(The number of finished memory requests for a period)}}{\text{(The number of total memory requests for a period)}}$
    - Expected Progress : $\frac{\text{(Elapsed cycles in a period)}}{\text{(Total cycles in a period)}}$
  - Every scheduling unit, dynamically adjust HWA priority
    - If Expected Progress > EmergentThreshold (=0.9) **: HWA > CPU**
    - If (Current Progress) > (Expected Progress) : **HWA < CPU**
    - If (Current Progress) <= (Expected Progress) : **HWA = CPU**

1. An HWA is prioritized over CPU cores *only when* it is closed to HWA's deadline
   ⇨ The HWA often misses deadlines
2. This scheme does not consider the diverse memory access characteristics of CPUs and HWAs
   - It treats each CPU and each HWA equally

⇨ Missing opportunities to improve system performance

Outline

- Introduction
- Problem with Existing Memory Schedulers for Heterogeneous Systems
- DASH: Key Ideas
- DASH: Scheduling Policy
- Evaluation and Results
- Conclusion

Key Idea 1: Distributed Priority

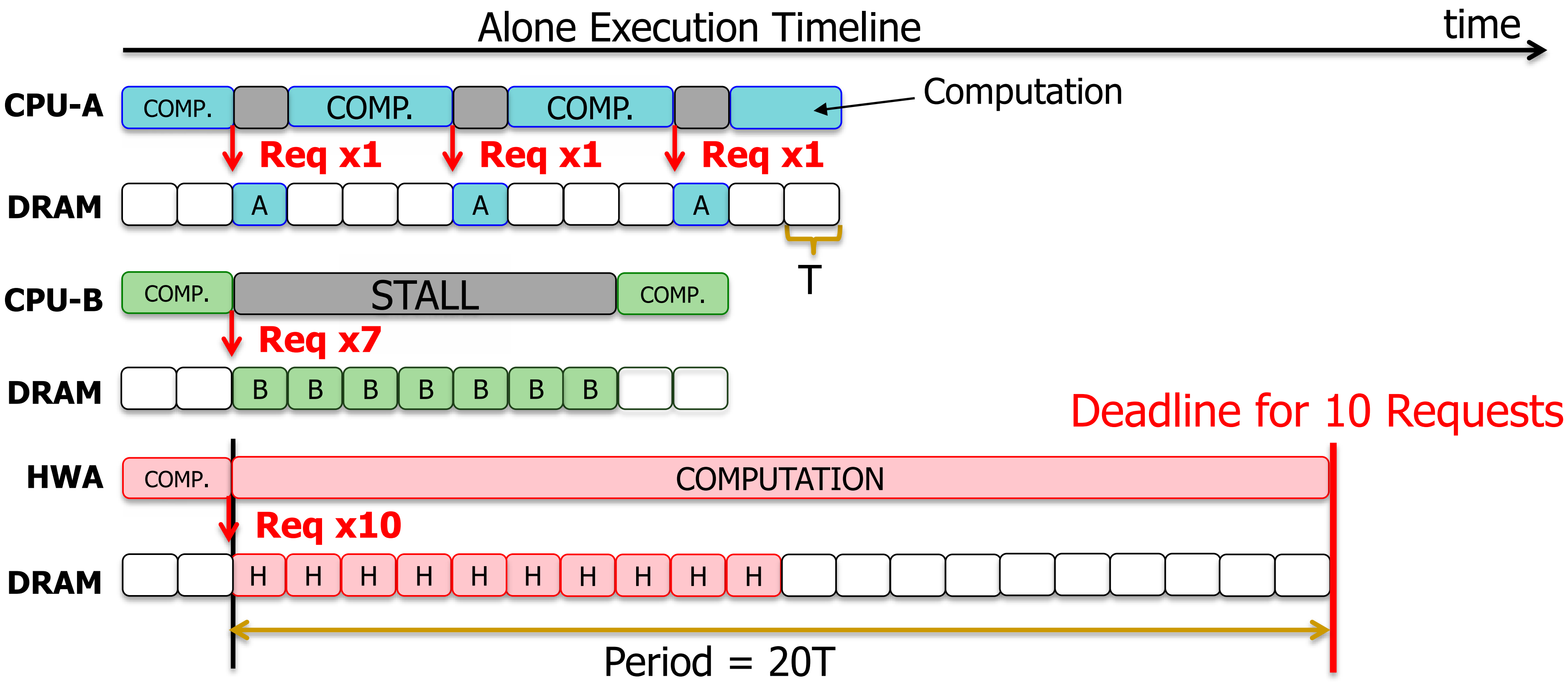
- Problem 1: An HWA is prioritized over CPU cores *only when* it is close to HWA's deadline

- Key Idea 1: **Distributed Prioritization for a CPU-HWA system**
  - Compares HWA's current progress against expected progress
    - Current Progress : $\frac{\text{(The number of finished memory requests for a period)}}{\text{(The number of total memory requests for a period)}}$
    - Expected Progress : $\frac{\text{(Elapsed cycles in a period)}}{\text{(Total cycles in a period)}}$
  - Dynamically adjust HWA priority based on its progress every scheduling unit
    - If Expected Progress > EmergentThreshold (=0.9) **: HWA > CPU**
    - If (Current Progress) > (Expected Progress) : **HWA < CPU**
    - If (Current Progress) <= (Expected Progress) : **HWA > CPU**

Prioritize HWAs over CPU anytime when the HWA is not making good progress

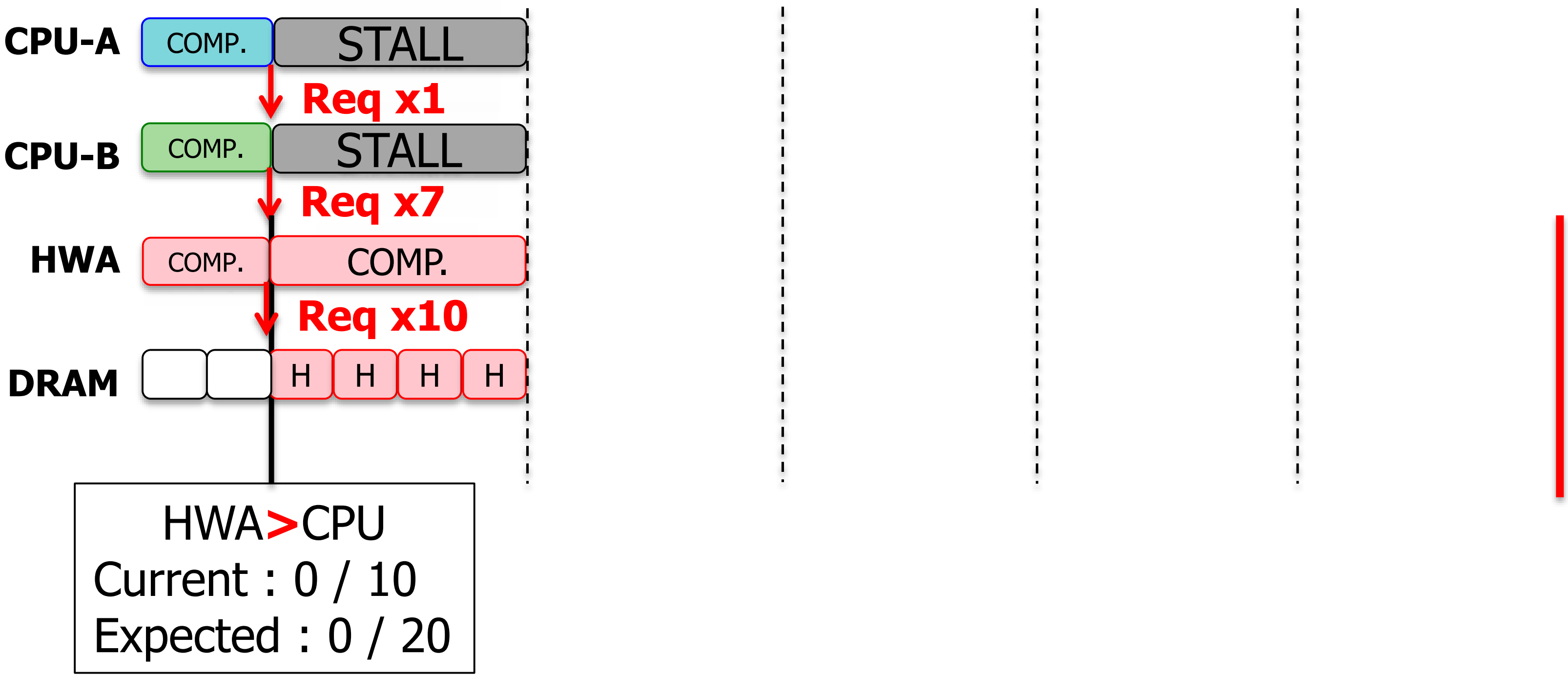
Example: Scheduling HWA and CPU Requests

- Scheduling requests from 2 CPU applications and a HWA
  - CPU-A : memory non-intensive application
  - CPU-B : memory intensive application



DASH: Distributed Priority

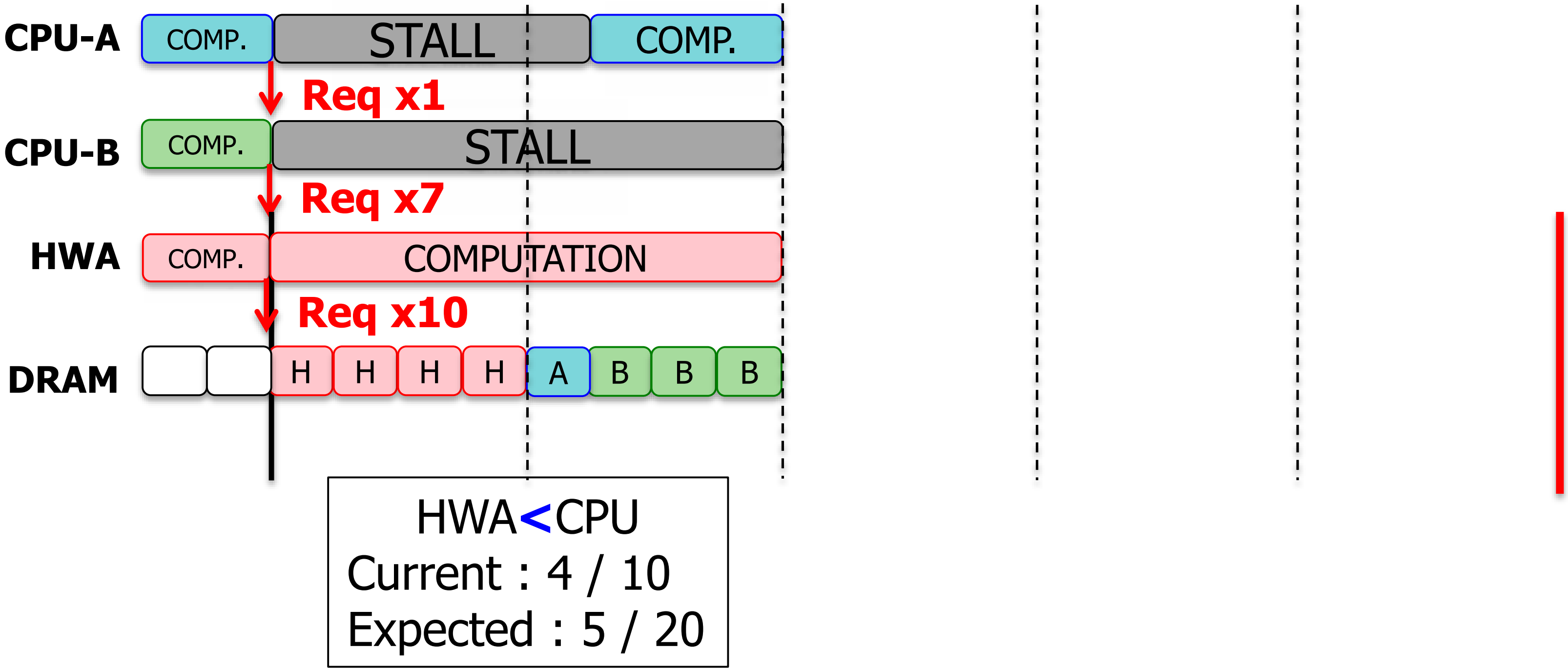
- **Distributed Priority (Scheduling unit = 4T)**



HWA>CPU
Current : 0 / 10
Expected : 0 / 20

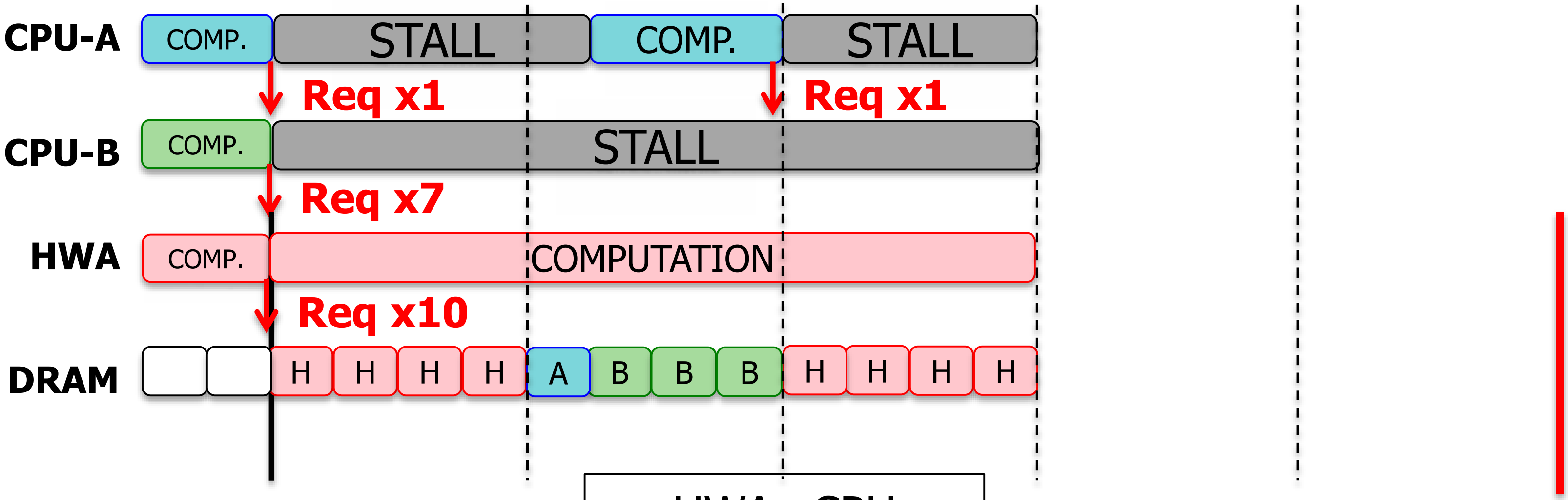
DASH: Distributed Priority

- **Distributed Priority (Scheduling unit = 4T)**



DASH: Distributed Priority

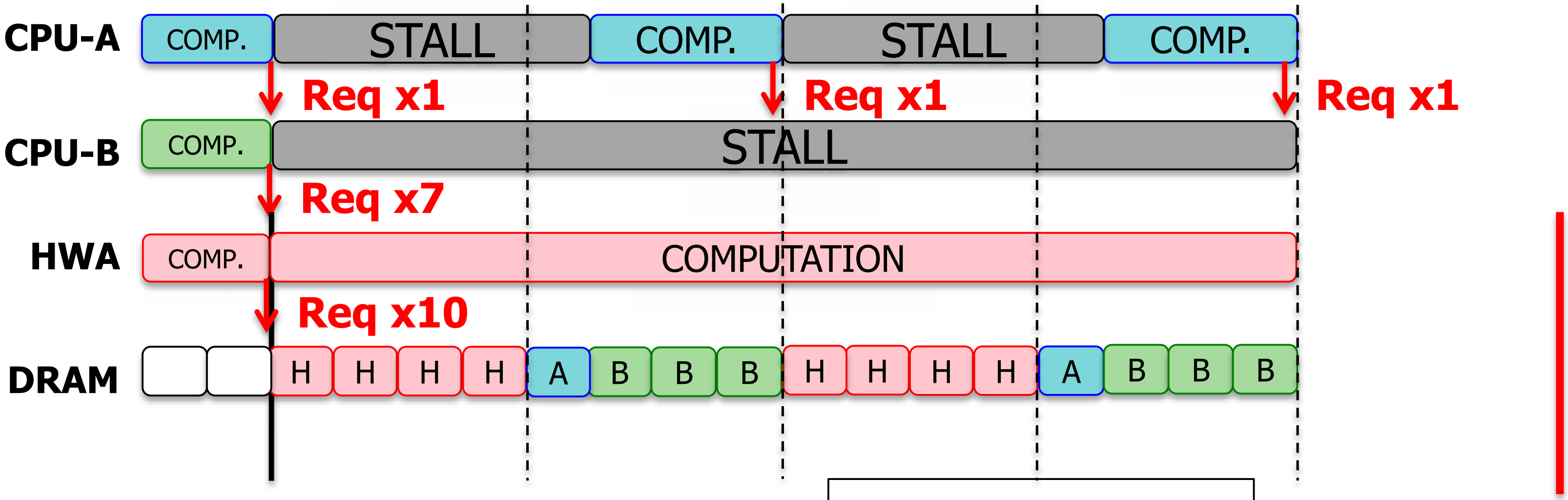
- **Distributed Priority (Scheduling unit = 4T)**



HWA>CPU
Current : 4 / 10
Expected : 8 / 20

DASH: Distributed Priority

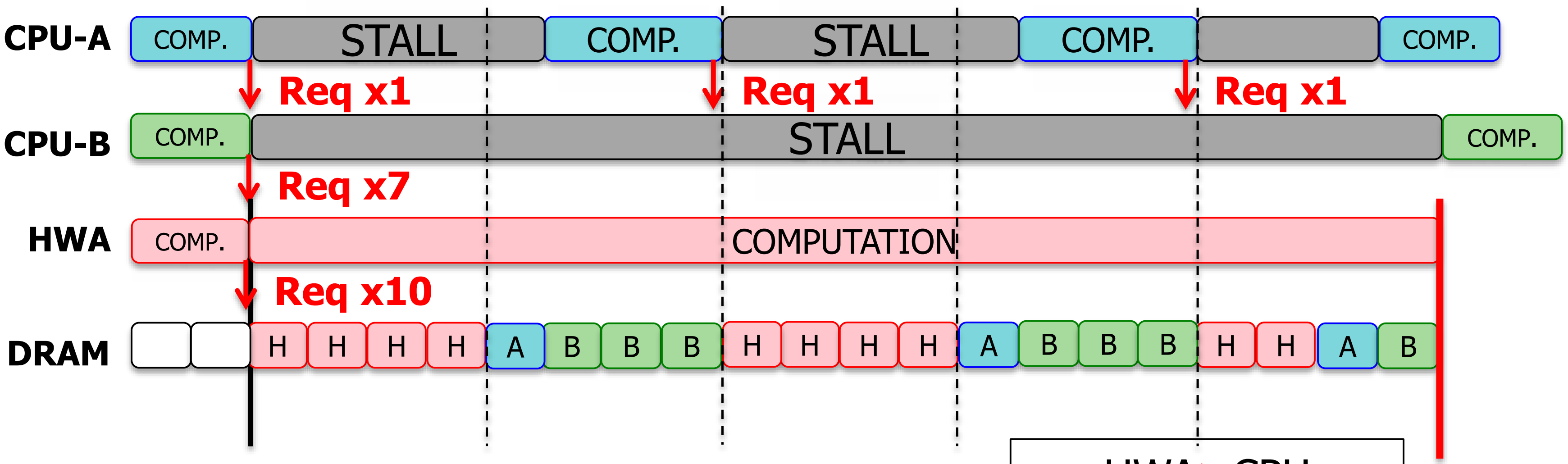
- **Distributed Priority (Scheduling unit = 4T)**



HWA<CPU
Current : 8 / 10
Expected : 12 / 20

DASH: Distributed Priority

- **Distributed Priority (Scheduling unit = 4T)**



HWA>CPU
Current : 8 / 10
Expected : 16 / 20

Problem2: Application-unawareness

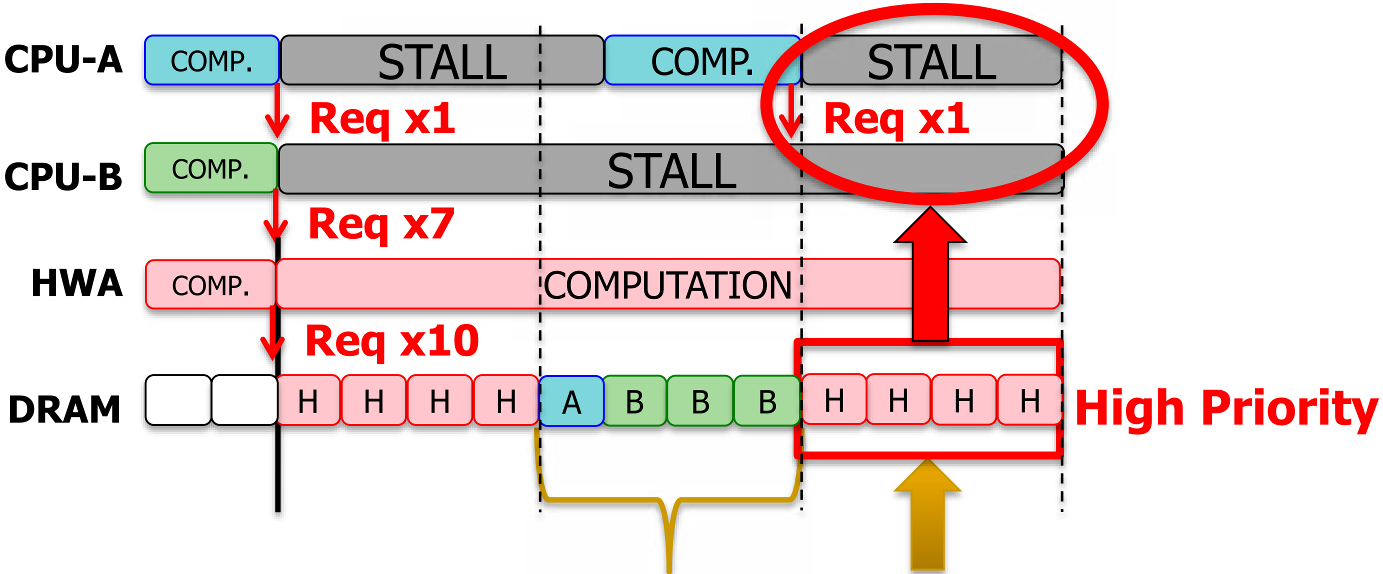
- **Problem 2 (Application-unawareness):** Existing memory schedulers for heterogeneous systems do no consider the diverse memory access characteristics of CPUs and HWAs

- Application-unawareness causes two problems
  - **Problem 2.1:** When a HWA has high priority (i.e., not measuring up to its expected progress), it interferes with *all* CPU cores for a long time
  - **Problem 2.2:** A HWA with a short period misses its deadlines due to fluctuations in available memory bandwidth (due to priority changes of other HWAs)

Problem 2.1 and Its Solution

- Problem 2.1 Restated: When HWA is low priority, it is deprioritized too much → It becomes high priority as a result and destroys CPU progress
- Goal: Avoid making the HWA high priority as much as possible

HWA delays both A and B



When high priority, HWA causes CPU to stall

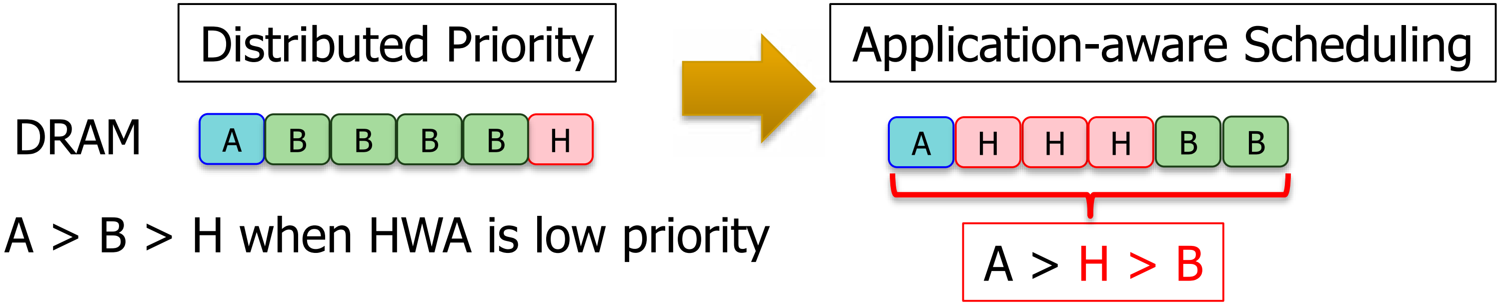
When low priority, the HWA saves requests

Key Idea 2.1: Application-aware Scheduling for CPUs

Key Idea 2.1: HWA priority over CPUs should depend on CPU memory intensity

- **Not all CPUs are equal**
  - Memory-intensive cores are much less vulnerable to memory access latency
  - Memory-non-intensive cores are much more vulnerable to latency

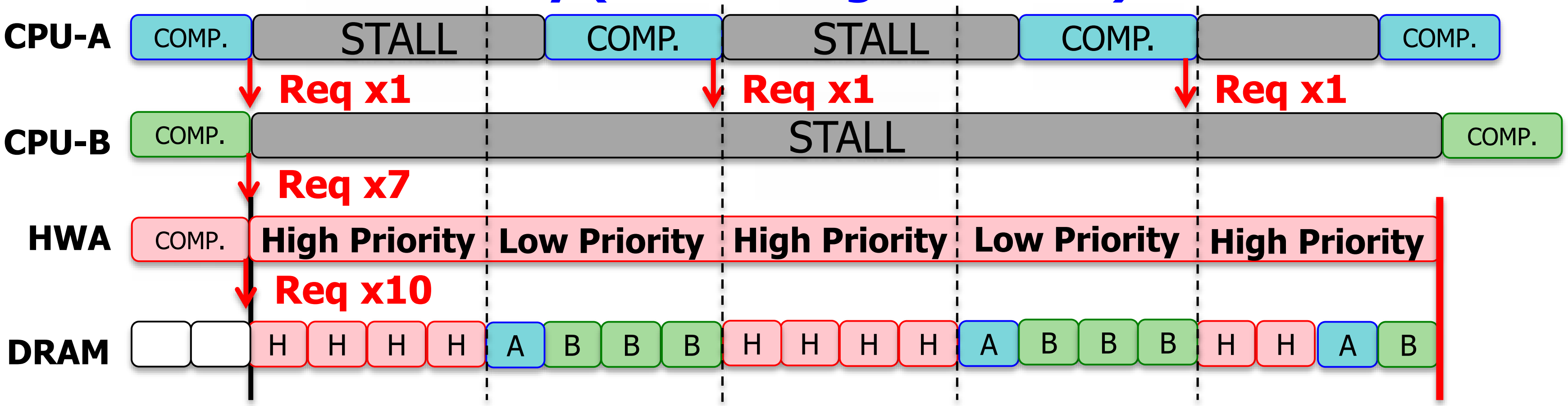
- While HWA has low priority, HWA is prioritized over memory-intensive cores


Distributed Priority
DRAM
A
B
B
B
B
H
A > B > H when HWA is low priority
Application-aware Scheduling
A
H
H
H
B
B
A > H > B

A: Memory-non-intensive, B: Memory-intensive

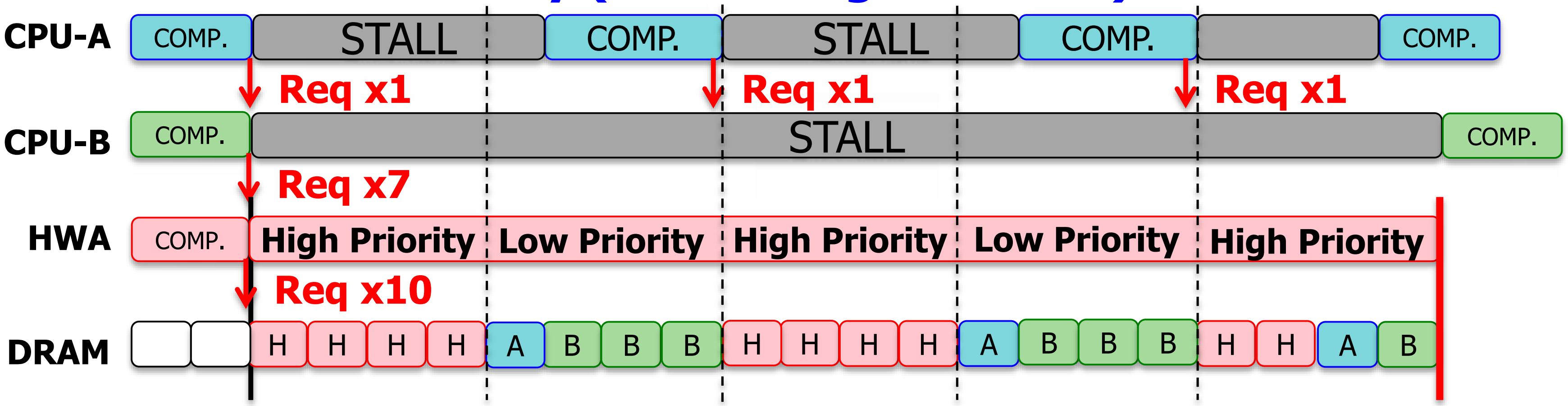
DASH: Application-aware Scheduling

- **Distributed Priority (Scheduling unit = 4T)**

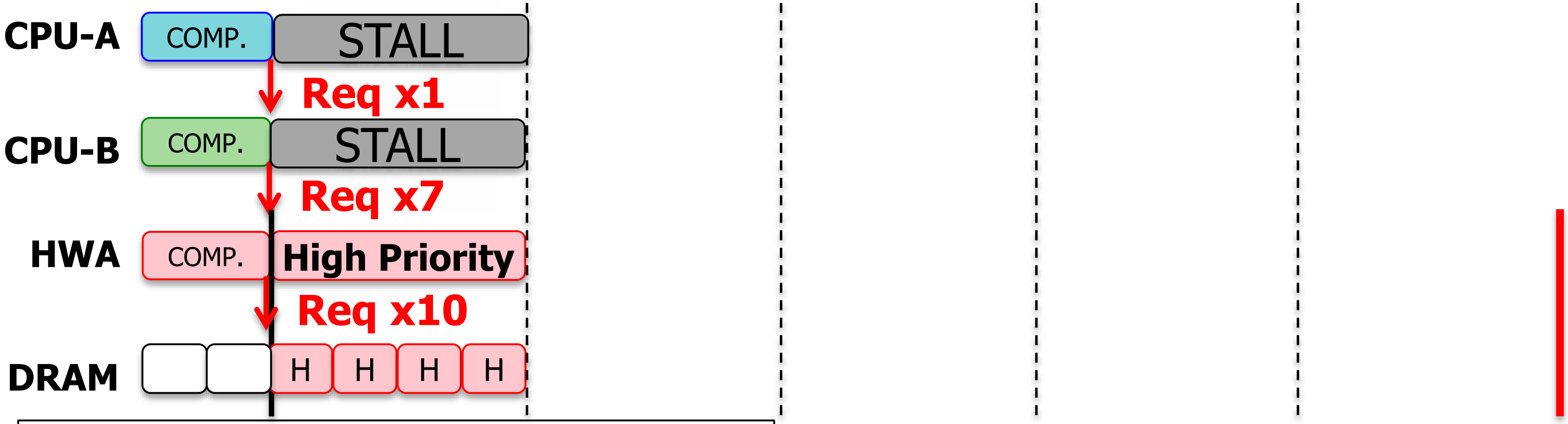


DASH: Application-aware Scheduling

- **Distributed Priority (Scheduling unit = 4T)**



- **Application-aware Scheduling (Scheduling unit = 4T)**

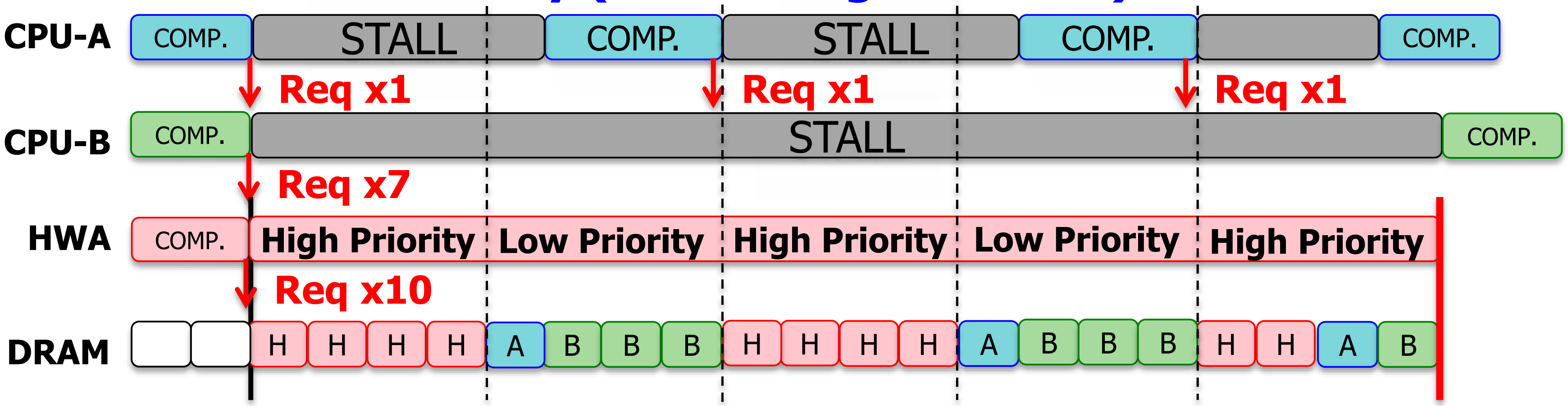


HWA>CPU-A&B

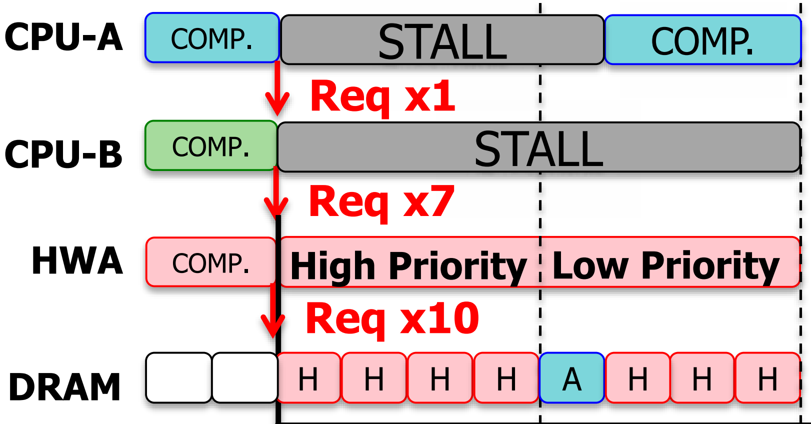
Current : 0 / 10 Expected : 0 / 20

DASH: Application-aware Scheduling

- **Distributed Priority (Scheduling unit = 4T)**



- **Application-aware Scheduling (Scheduling unit = 4T)**

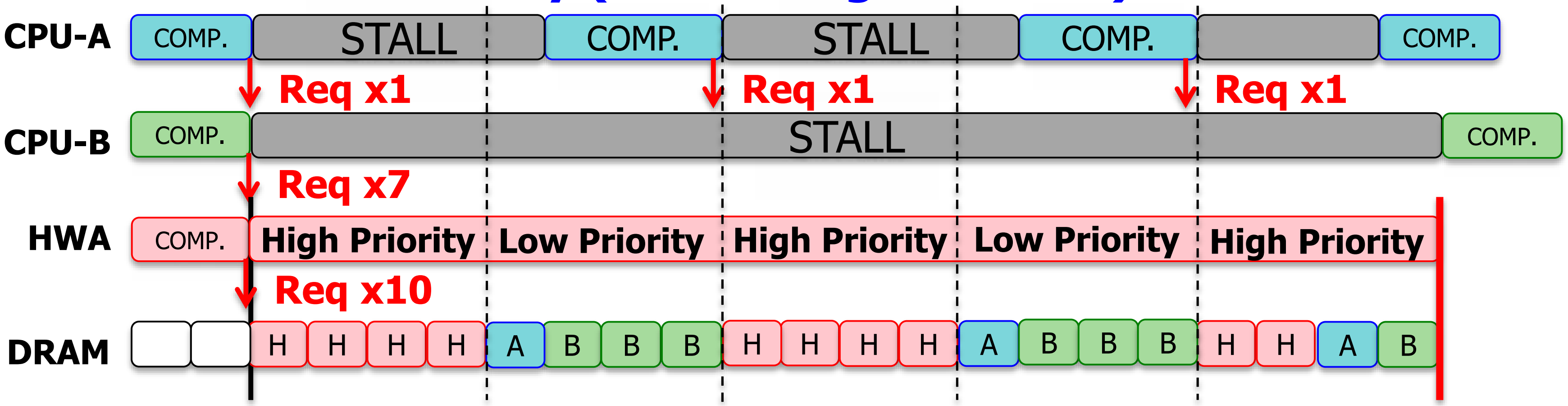


CPU-A > HWA > CPU-B

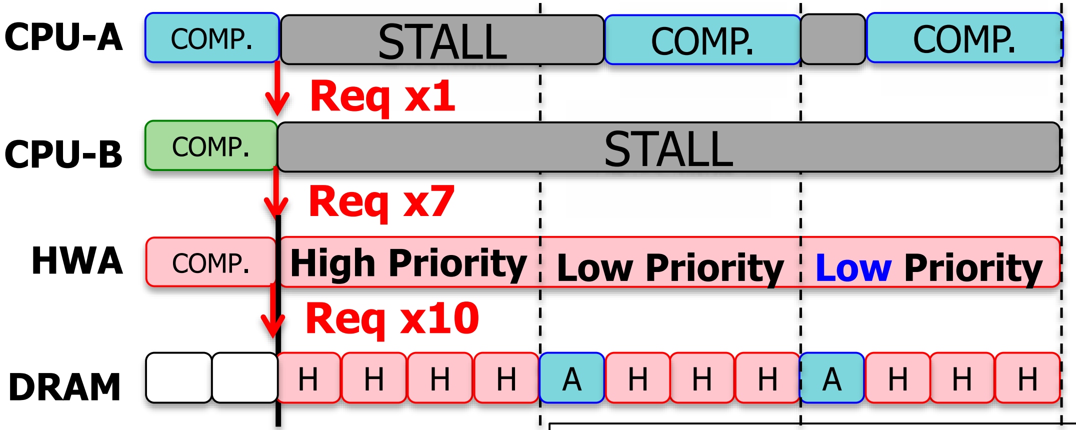
Current : 4 / 10 Expected : 4 / 20

DASH: Application-aware Scheduling

- **Distributed Priority (Scheduling unit = 4T)**



- **Application-aware Scheduling (Scheduling unit = 4T)**

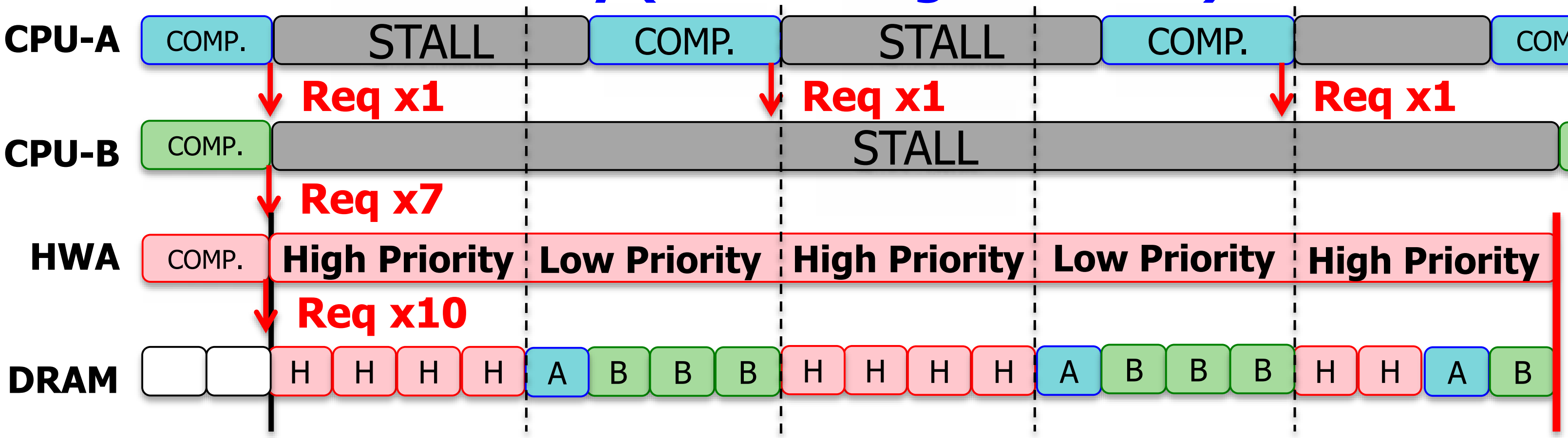


CPU-A > HWA > CPU-B

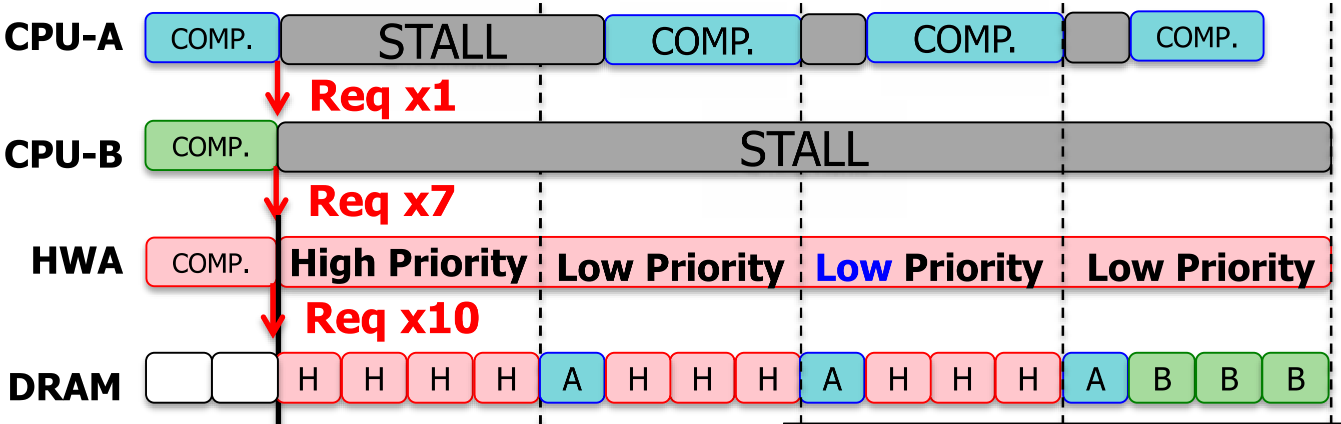
Current : 7 / 10 Expected : 8 / 20

DASH: Application-aware Scheduling

- **Distributed Priority (Scheduling unit = 4T)**



- **Application-aware Scheduling (Scheduling unit = 4T)**

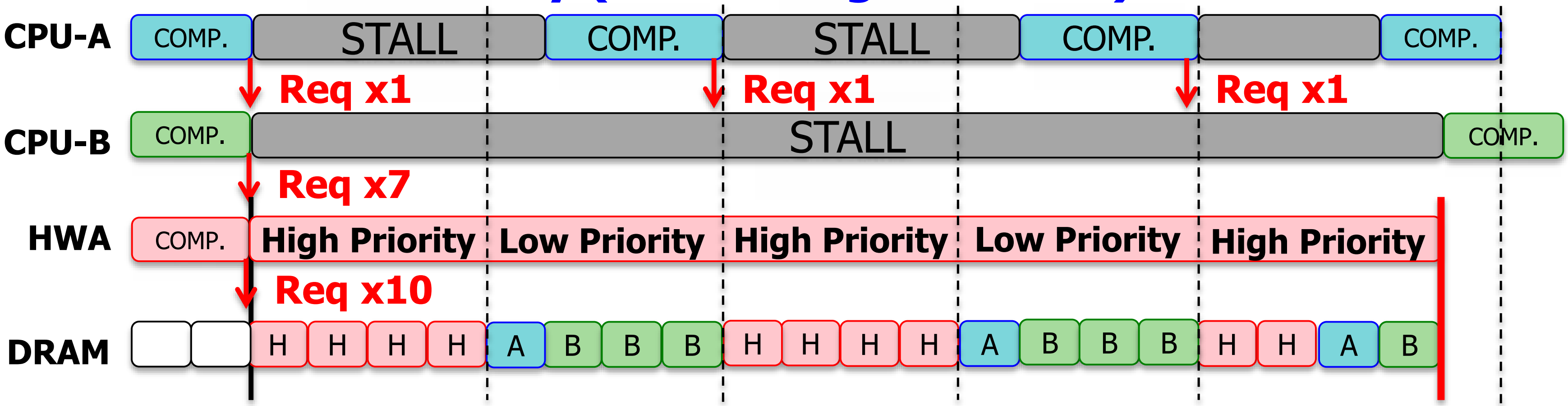


CPU-A > HWA > CPU-B

Current : 10 / 10 Expected : 12 / 20

DASH: Application-aware Scheduling

- **Distributed Priority (Scheduling unit = 4T)**



CPU-A: COMP. | STALL | COMP. | STALL | COMP. | | COMP.

Req x1 | Req x1 | Req x1

CPU-B: COMP. | STALL | COMP.

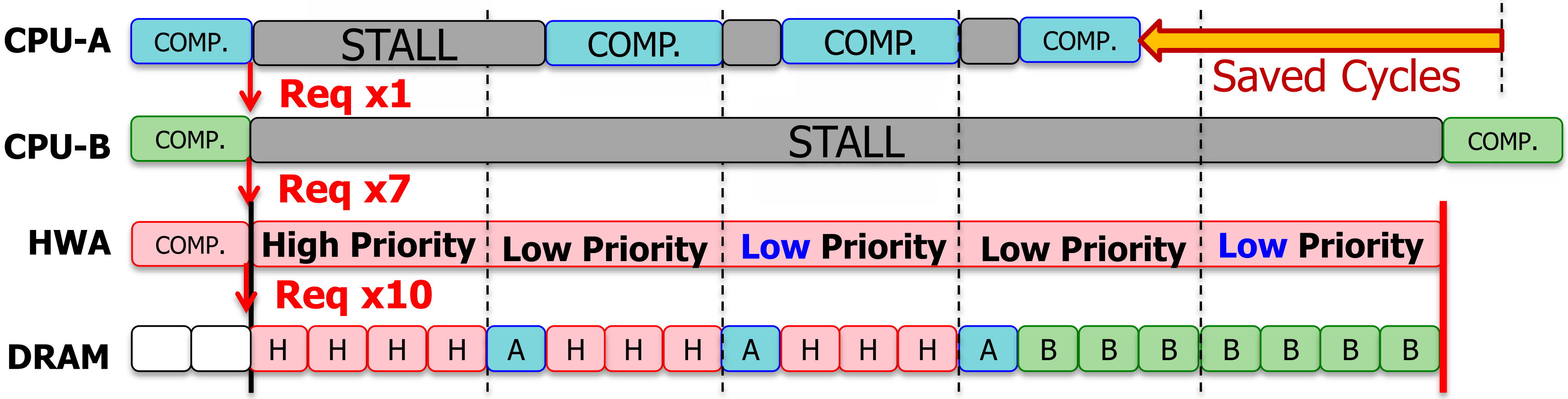
Req x7

HWA: COMP. | High Priority | Low Priority | High Priority | Low Priority | High Priority

Req x10

DRAM: | | H | H | H | H | A | B | B | B | H | H | H | H | A | B | B | B | H | H | A | B

- **Application-aware Scheduling (Scheduling unit = 4T)**



CPU-A: COMP. | STALL | COMP. | | COMP. | | COMP.

Saved Cycles

Req x1

CPU-B: COMP. | STALL | COMP.

Req x7

HWA: COMP. | High Priority | Low Priority | Low Priority | Low Priority | Low Priority

Req x10

DRAM: | | H | H | H | H | A | H | H | H | A | H | H | H | A | B | B | B | B | B | B | B

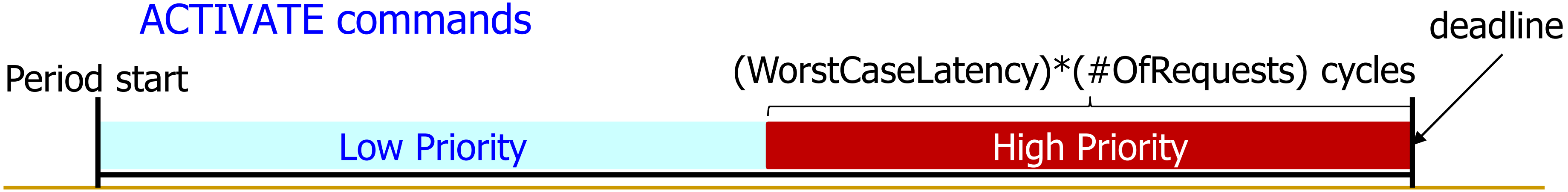
Problem 2.2 and Its Solution

- Problem 2.2: A HWA with a short-deadline-period misses its deadlines due to fluctuations in available memory bandwidth (due to priority changes of other HWAs)

	HWA-A	HWA-B
Period	63,041 Cycles	5,447 Cycles
Bandwidth	8.32 GB/s	475 MB/s

HWA-A: meets all its deadlines
HWA-B: misses a deadline every 2000 periods

- Key Idea 2.2: Estimate the worst-case memory access latency and give a short-deadline-period HWA the highest priority for (WorstCaseLatency) * (NumberOfRequests) cycles close to its deadline
  - WorstCaseLatency = tRC : the minimum time between two DRAM row ACTIVATE commands



DASH: Summary of Key Ideas

1. Distributed priority

2. Application-aware scheduling

3. Worst-case memory access latency based prioritization

Outline

- Introduction
- Problem with Existing Memory Schedulers for Heterogeneous Systems
- DASH: Key Ideas
- DASH: Scheduling Policy
- Evaluation and Results
- Conclusion

DASH: Scheduling Policy

- DASH scheduling policy
  1. Short-deadline-period HWAs with high priority
  2. Long-deadline-period HWAs with high priority
  3. Memory non-intensive CPU applications
  4. Long-deadline-period HWAs with low priority
  5. Memory-intensive CPU applications
  6. Short-deadline-period HWAs with low priority

DASH: Scheduling Policy

- DASH scheduling policy

  1. Short-deadline-period HWAs with high priority
  2. Long-deadline-period HWAs with high priority
  3. Memory non-intensive CPU applications
  4. Long-deadline-period HWAs with low priority
  5. Memory-intensive CPU applications
  6. Short-deadline-period HWAs with low priority

Switch probabilistically

Outline

- Introduction
- Problem with Existing Memory Schedulers for Heterogeneous Systems
- DASH: Key Ideas
- DASH: Scheduling Policy
- Evaluation and Results
- Conclusion

Experimental Methodology (1/2)

- New Heterogeneous System Simulator
  - We have released this at GitHub (https://github.com/CMU-SAFARI/HWASim)
- Configurations
  - 8 CPUs (2.66GHz), 32KB/L1, 4MB Shared/L2
  - 4 HWAs
  - DDR3 1333 DRAM x 2 channels
- Workloads
  - CPUs: 80 multi-programmed workloads
    - SPEC CPU2006, TPC, NAS parallel benchmark
  - HWAs:
    - Image processing
    - Image recognition [Lee+ ICCD 2009] [Viola and Jones CVPR 2001]
- Metrics
  - CPUs : Weighted Speedup
  - HWAs : Deadline met ratio (%)

Experimental Methodology (2/2)

- Parameters of the HWAs

	Period	**Bandwidth**	**Deadline Group**
IMG : Image Processing	33 ms	360MB/s	Long
HES : Hessian	2 us	478MB/s	Short
MAT : Matching (1) 20fps	35.4 us	8.32 GB/s	Long
MAT : Matching (2) 30fps	23.6 us	5.55 GB/s	Long
RSZ : Resize	46.5 – 5183 us	2.07 – 3.33 GB/s	Long
DET : Detect	0.8 – 9.6 us	1.60 – 1.86 GB/s	Short

- Configurations of 4 HWAs

	Configuration
Config-A	IMG x 2, HES, MAT(2)
Config-B	HES, MAT(1), RSZ, DET

Evaluated Memory Schedulers

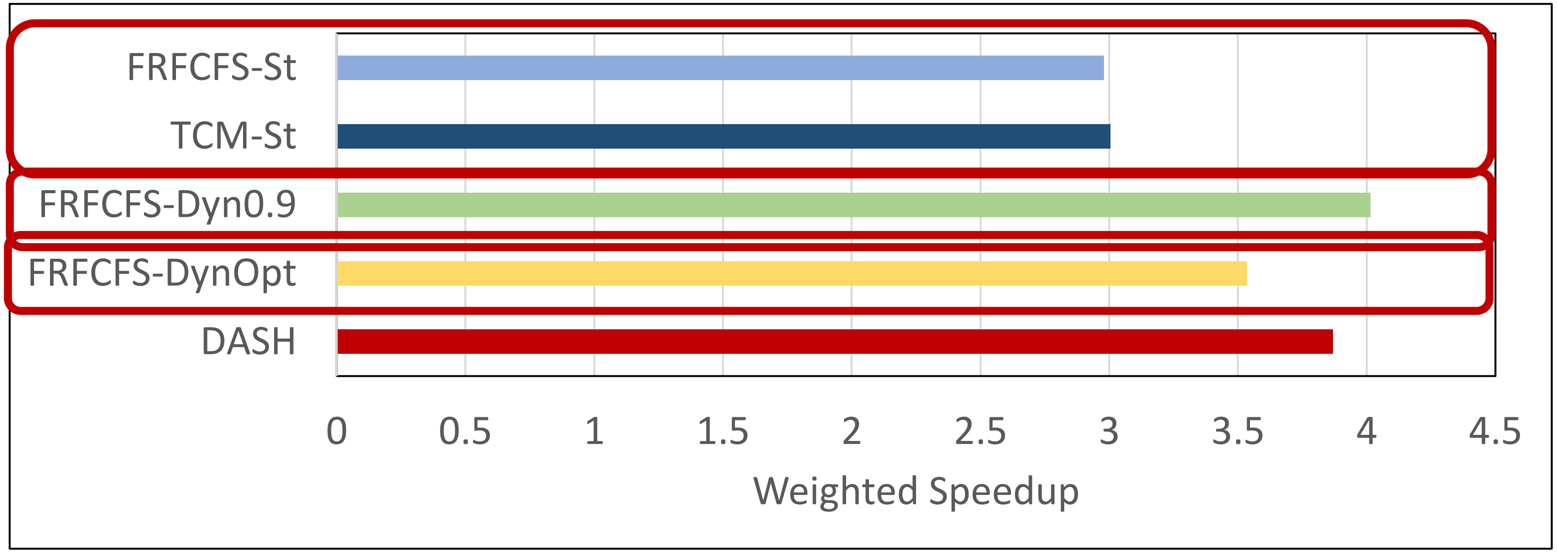
- FRFCFS-St, TCM-St: FRFCFS or TCM with *static priority* for HWAs
  - HWAs *always* have higher priority than CPUs
  - FRFCFS-St: FRFCFS [Zuravleff and Robinson US Patent 1997, Rixner et al. ISCA 2000] for CPUs
    - Prioritizes row-buffer hits and older requests
  - TCM-St: TCM [Kim+ MICRO 2010] for CPUs
    - Always prioritizes memory-non-intensive applications
    - Shuffles thread ranks of memory-intensive applications
- FRFCFS-Dyn: FRFCFS with *dynamic priority* for HWAs [Jeong et al., DAC 2012]
  - HWA's priority is dynamically adjusted based on its progress
    - FRFCFS-Dyn0.9: EmergentThreshold = **0.9** for all HWAs (Only after 90% of the HWA's period elapsed, the HWA has higher priority than CPUs)
    - FRFCFS-DynOpt: Each HWA has different EmergentThreshold to meet its deadline

Config-A			**Config-B**			
IMG	HES	MAT	HES	MAT	RSZ	DET
0.9	0.2	0.2	0.5	0.4	0.7	0.5

- DASH: *Distributed Priority* + Application-aware scheduling for CPUs + HWAs
  - TCM is used for CPUs to classify memory intensity of CPUs
  - EmergentThreshold = 0.8 for all HWAs

Performance and Deadline Met Ratio

- **Weighted Speedup for CPUs**

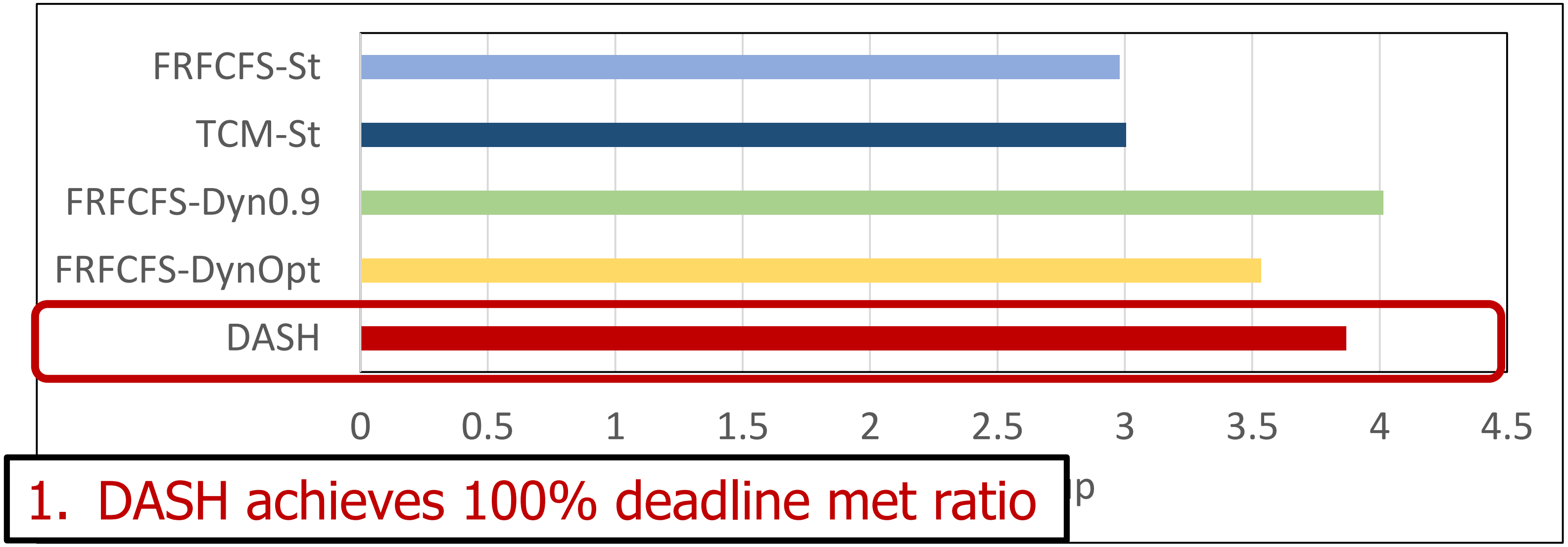


- **Deadline Met Ratio (%) for HWAs**

	IMG	HES	MAT	RSZ	DET
FRFCFS-St	100	100	100	100	100
TCM-St	100	100	100	100	100
FRFCFS-Dyn0.9	100	99.4	46.01	97.98	97.14
FRFCFS-DynOpt	100	100	99.997	100	99.99
DASH	100	100	100	100	100

Performance and Deadline Met Ratio

- **Weighted Speedup for CPUs**



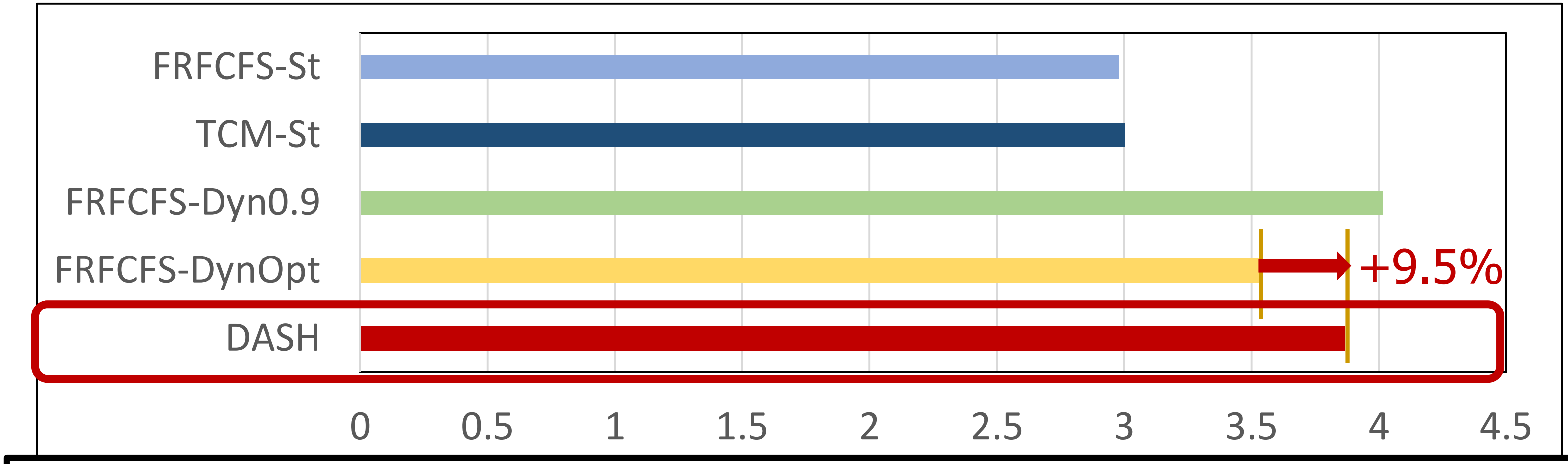
1. DASH achieves 100% deadline met ratio

- **Deadline Met Ratio (%) for HWAs**

	IMG	HES	MAT	RSZ	DET
FRFCFS-St	100	100	100	100	100
TCM-St	100	100	100	100	100
FRFCFS-Dyn0.9	100	99.4	46.01	97.98	97.14
FRFCFS-DynOpt	100	100	99.997	100	99.99
DASH	100	100	100	100	100

Performance and Deadline Met Ratio

- **Weighted Speedup for CPUs**

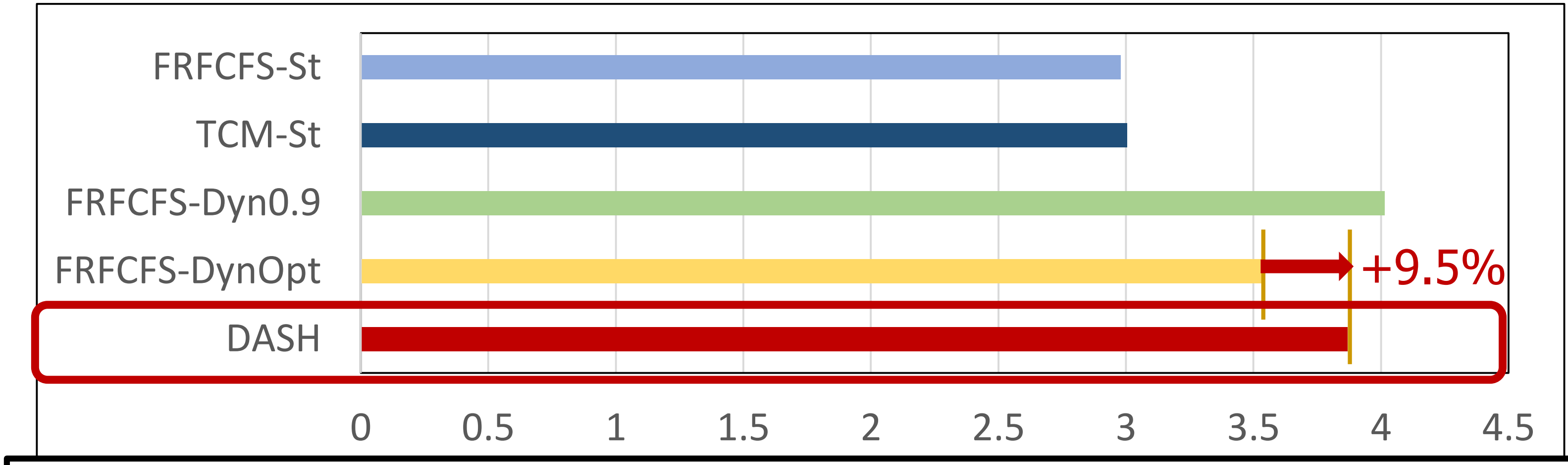


1. DASH achieves 100% deadline met ratio
2. DASH achieves better performance (+9.5%) than FRFCFS-DynOpt that meets the most of HWAs' deadlines (Optimized for HWAs)

FRFCFS-St	100	100	100	100	100
TCM-St	100	100	100	100	100
FRFCFS-Dyn0.9	100	99.4	46.01	97.98	97.14
FRFCFS-DynOpt	100	100	99.997	100	99.99
DASH	100	100	100	100	100

Performance and Deadline Met Ratio

- **Weighted Speedup for CPUs**



1. DASH achieves 100% deadline met ratio
2. DASH achieves better performance (+9.5%) than FRFCFS-DynOpt that meets the most of HWAs' deadlines (Optimized for HWAs)
3. DASH achieves comparable performance to FRFCFS-Dyn0.9 that frequently misses HWAs' deadlines (Optimized for CPUs)

TCM-St	100	100	100	100	100
FRFCFS-Dyn0.9	100	99.4	46.01	97.98	97.14
FRFCFS-DynOpt	100	100	99.997	100	99.99
DASH	100	100	100	100	100

DASH Scheduler: Summary

- Problem: Hardware accelerators (HWAs) and CPUs share the same memory subsystem and interfere with each other in main memory
- Goal: Design a memory scheduler that improves CPU performance while meeting HWAs' deadlines
- Challenge: Different HWAs have different memory access characteristics and different deadlines, which current schedulers do not smoothly handle
  - Memory-intensive and long-deadline HWAs significantly degrade CPU performance *when they become high priority* (due to slow progress)
  - Short-deadline HWAs sometimes miss their deadlines *despite high priority*
- Solution: DASH Memory Scheduler
  - Prioritize HWAs over CPU anytime when the HWA is not making good progress
  - Application-aware scheduling for CPUs and HWAs
- Key Results:

  1) Improves CPU performance for a wide variety of workloads by 9.5%

  2) Meets 100% deadline met ratio for HWAs
- DASH source code freely available on the GitHub

DASH: Deadline-Aware High-Performance Memory Scheduler for Heterogeneous Systems with Hardware Accelerators

Hiroyuki Usui, Lavanya Subramanian
Kevin Chang, Onur Mutlu

DASH source code is available at GitHub
https://github.com/CMU-SAFARI/HWASim

Carnegie Mellon

Backup Slides

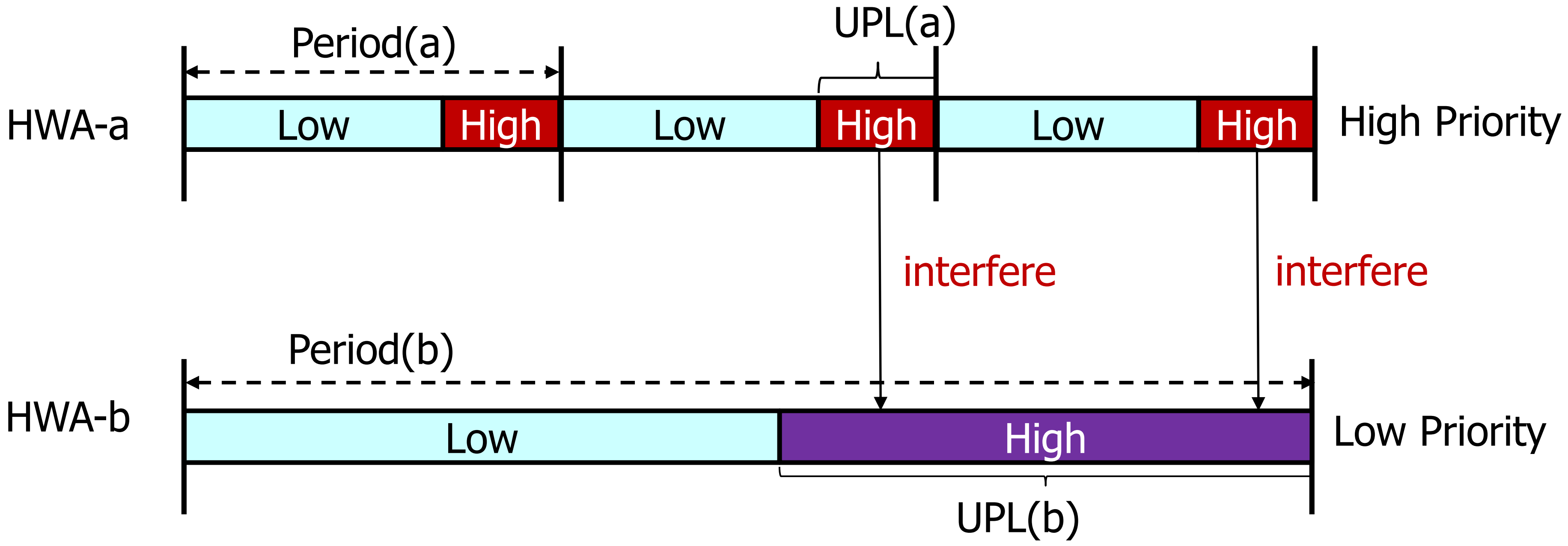
Probabilistic Switching of Priorities

- Each Long-deadline-period HWA x has probability $P_b(x)$

- Scheduling using $P_b(x)$
  - With a probability $P_b(x)$
    - Memory-intensive applications > Long-deadline-period HWA x
  - With a probability $1 - P_b(x)$
    - Memory-intensive applications < Long-deadline-period HWA x

- Controlling $P_b(x)$
  - Initial : $P_b(x)$ = 0
  - Every $SwitchingUnit$:
    - If CurrentProgress > ExpectedProgress : $P_b(x) \mathrel{+}= 1\%$
    - If CurrentProgress < ExpectedProgress : $P_b(x) \mathrel{-}= 5\%$

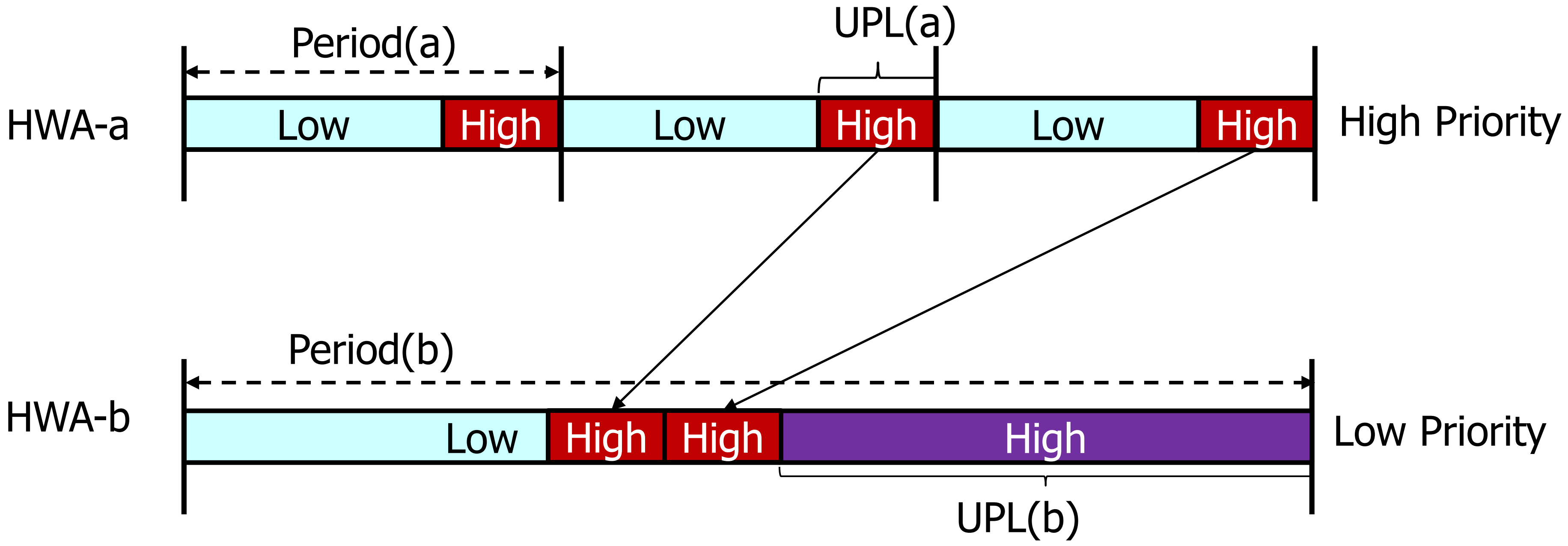
Priorities for Multiple Short-deadline-period HWAs



- A HWA with shorter deadline period is given higher priority (HWA-a > HWA-b)
- UPL = Urgent Period Length : *tRC x NumberOfRequests + a*
- During UPL(b), HWA-a will interfere HWA-b for (UPL(a) x 2) cycles at maximum
  - $\lceil UPL(b)/Period(a) \rceil = 2$
  - HWA(b) might fail the deadline due to the interference from HWA-a

SAFARI

Priorities for Multiple Short-deadline-period HWAs



- A HWA with shorter deadline period is given higher priority (HWA-a > HWA-b)
- UPL = Urgent Period Length : *tRC x NumberOfRequests + a*
- During UPL(b), HWA-a will interfere HWA-b for (UPL(a) x 2) cycles at maximum
  - $\lceil UPL(b)/Period(a) \rceil = 2$
  - HWA(b) might fail the deadline due to the interference from HWA-a
- HWA-b is prioritized when the time remaining in the period is (UPL(b) + UPL(a) x 2) cycles

Storage required for DASH

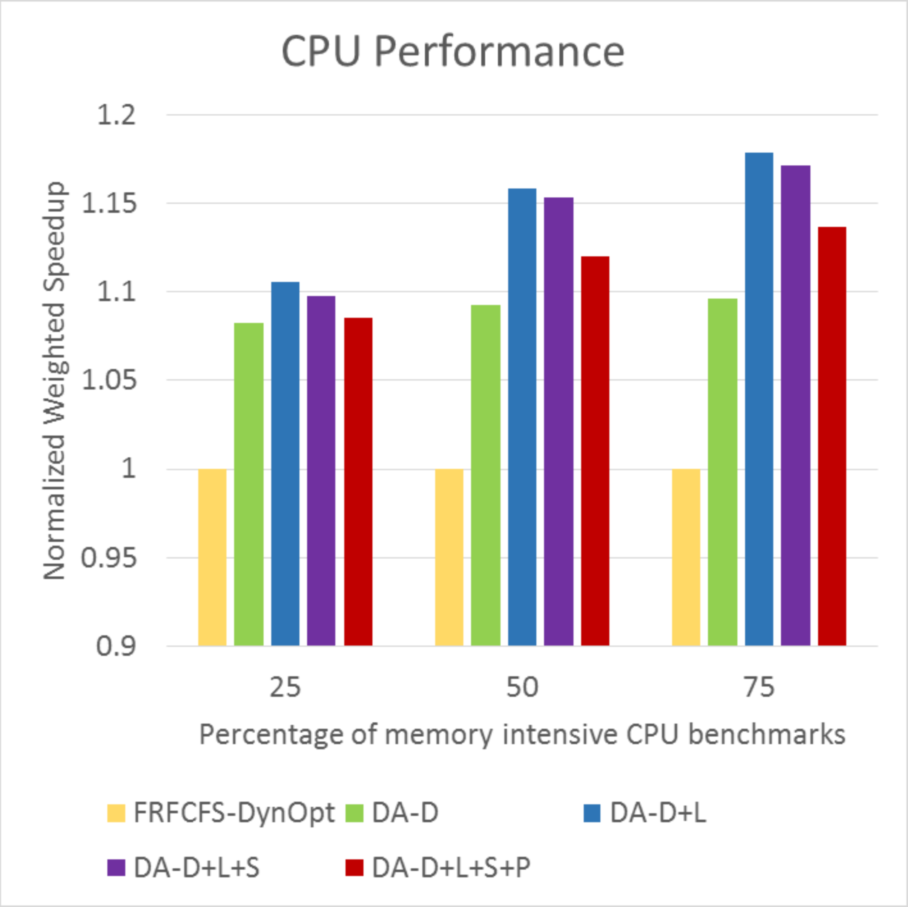
- **20 bytes** for each long-deadline-period HWA
- **12 bytes** for each short-deadline-period HWA

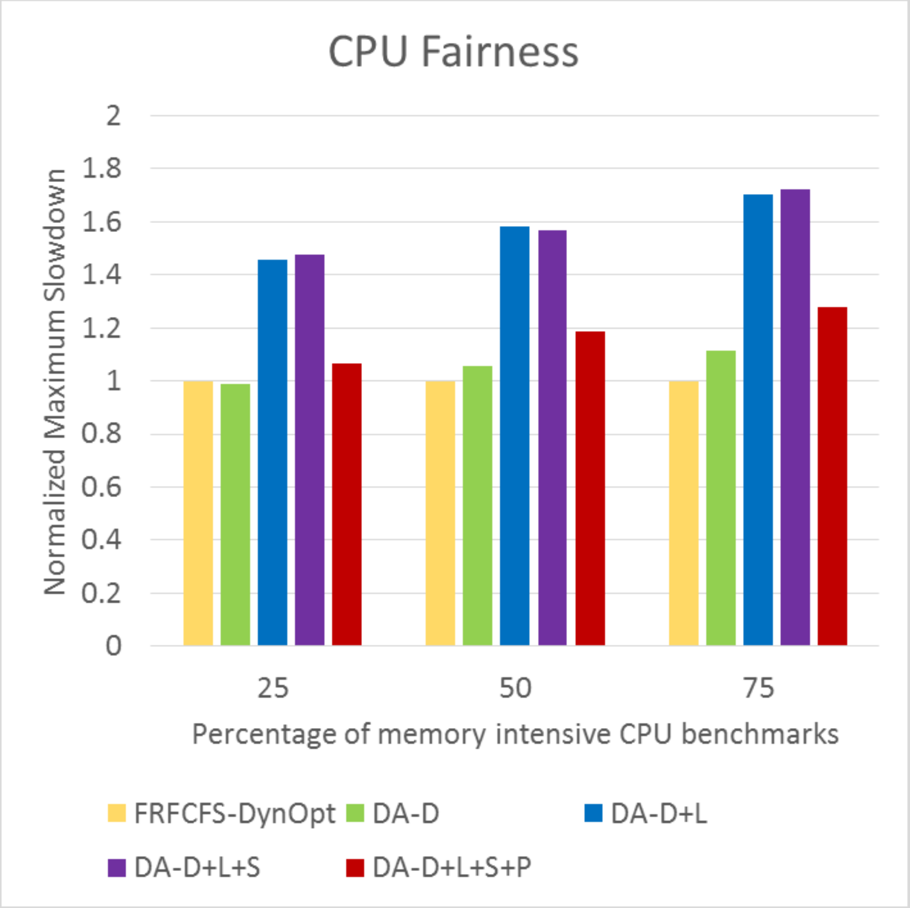
For long-deadline-period HWA	
Name	Function
Curr-Req	Number of requests completed in a deadline period
Total-Req	Total Number of requests to be completed in a deadline period
Curr-Cyc	Number of cycles elapsed in a deadline period
Total-Cyc	Total number of cycles in a deadline period
Pb	Probability for the priority switching between memory-intensive applications and HWA
For a short-deadline-period HWA	
Name	Function
Priority-Cyc	Indicates when the priority is transitioned to high
Curr-Cyc	Number of cycles elapsed in a deadline period
Total-Cyc	Total number of cycles elapsed in a deadline period

Simulation Parameter Details

- SchedulingUnit : 1000 CPU cycles
- SwitchingUnit : 500 CPU cycles
- ClusterFactor : 0.15
  - Fraction of total memory bandwidth allocated to memory-non-intensive CPU applications

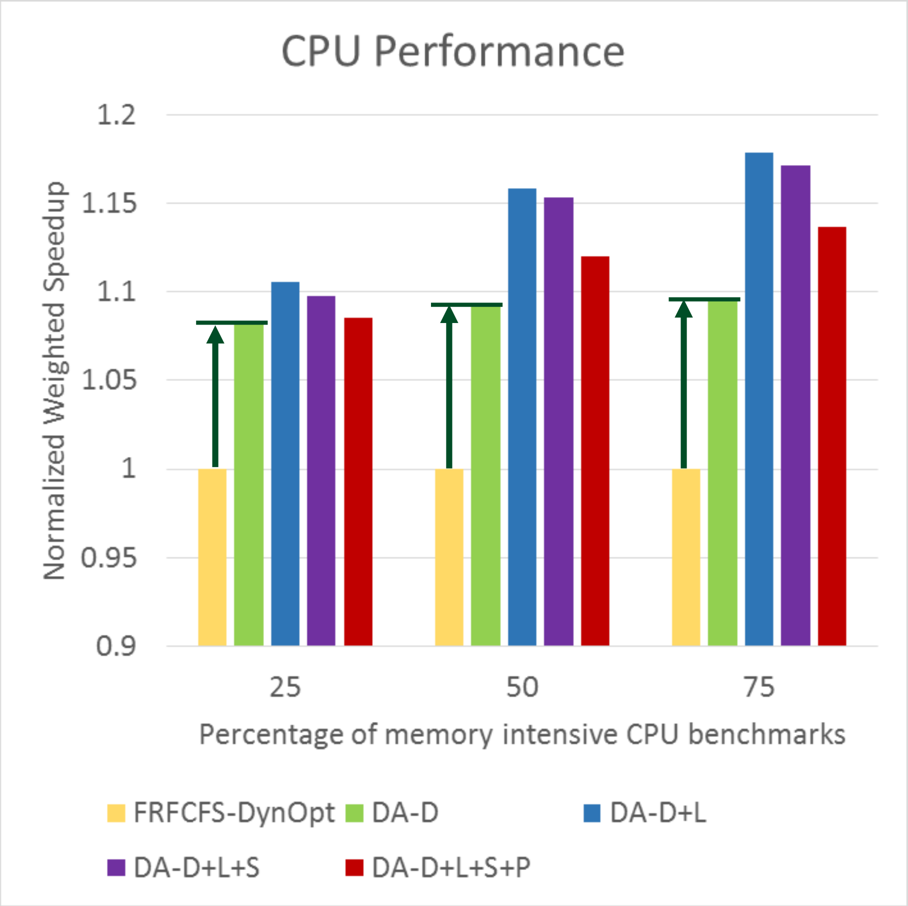
Performance breakdown of DASH

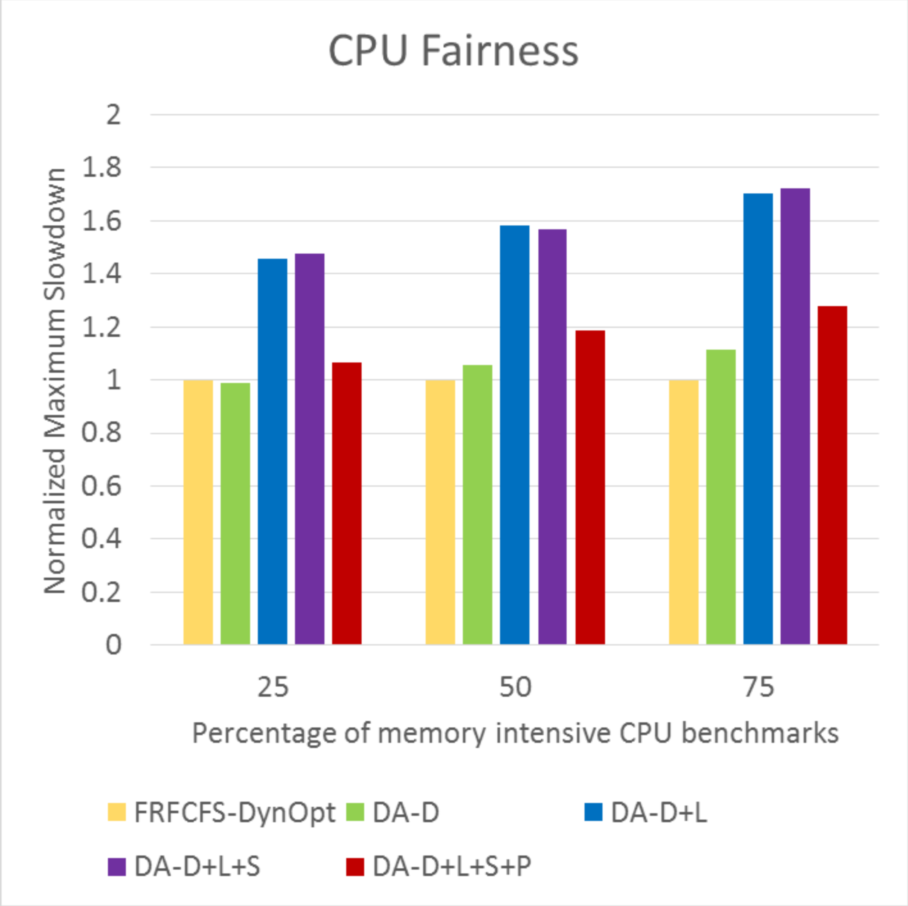




- DA-D : Distributed Priority
- DA-D+L : DA-D + application-aware priority for CPUs
- DA-D+L+S : DA-D+L + worst-case latency based priority for short-deadline HWAs
- DA-D+L+S+P (DASH) : DA-D+L+S + probabilistic prioritization

Performance breakdown of DASH

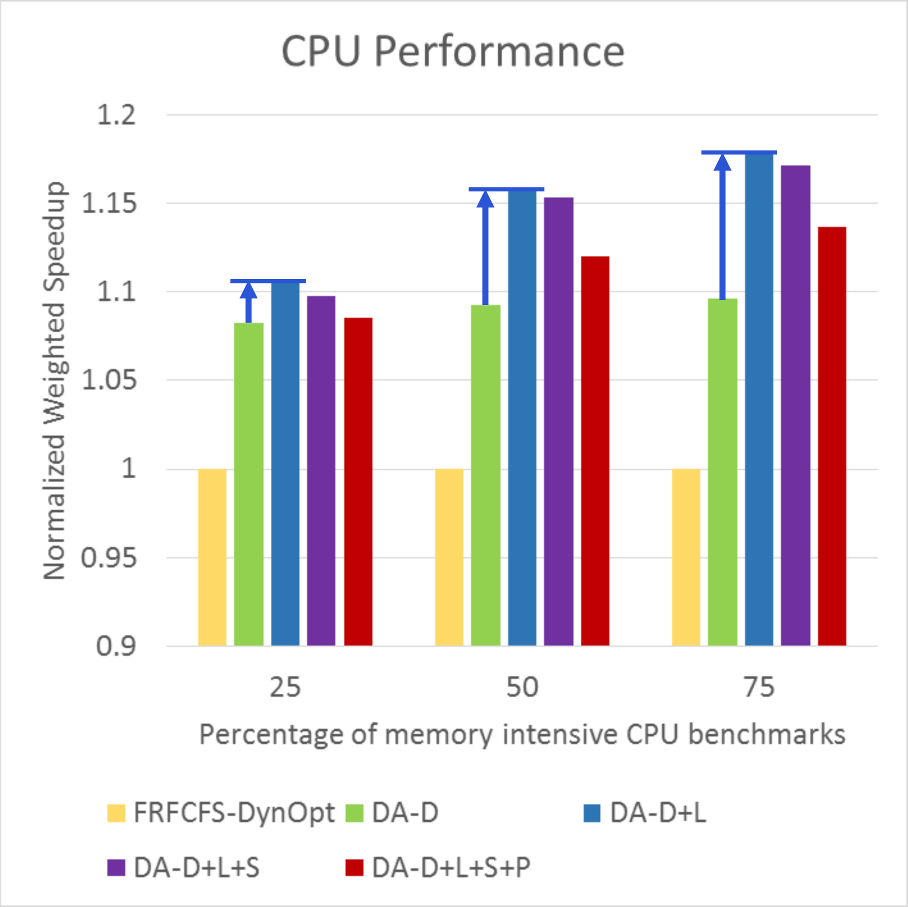


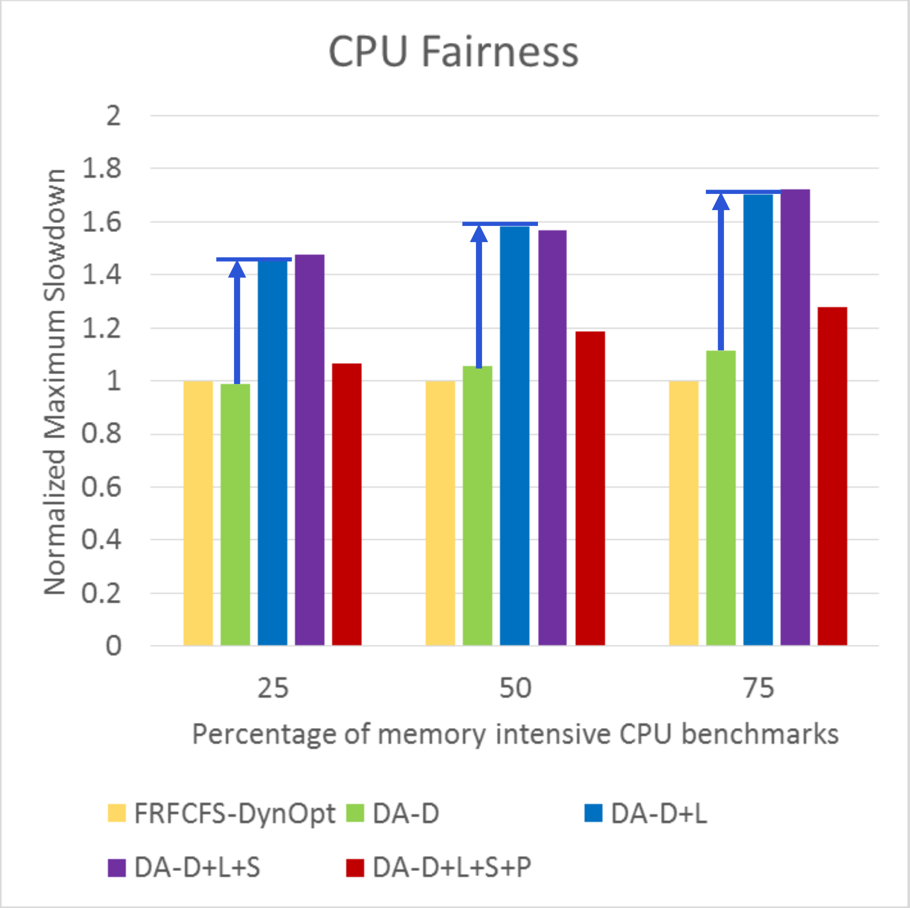


- DA-D : Distributed Priority
- e HWAs
- DA-D+L+S+P (DASH) : DA-D+L+S + probabilistic prioritization

Distributed priority improves performance (Max +9.5%)

Performance breakdown of DASH

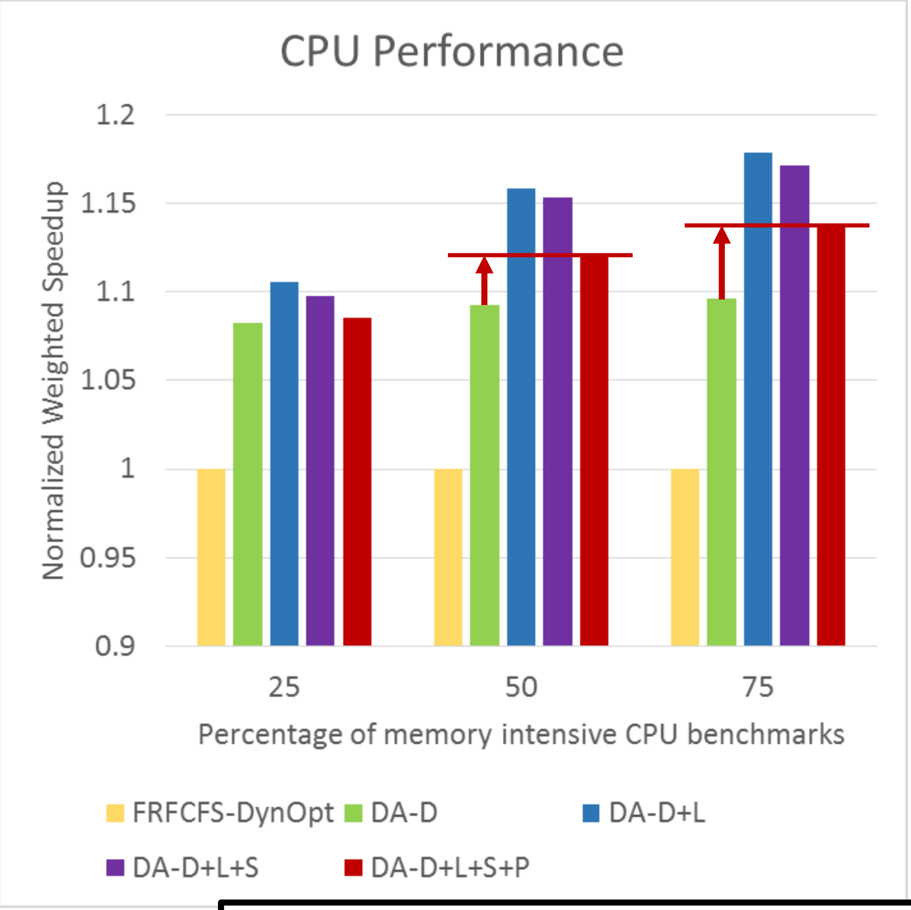


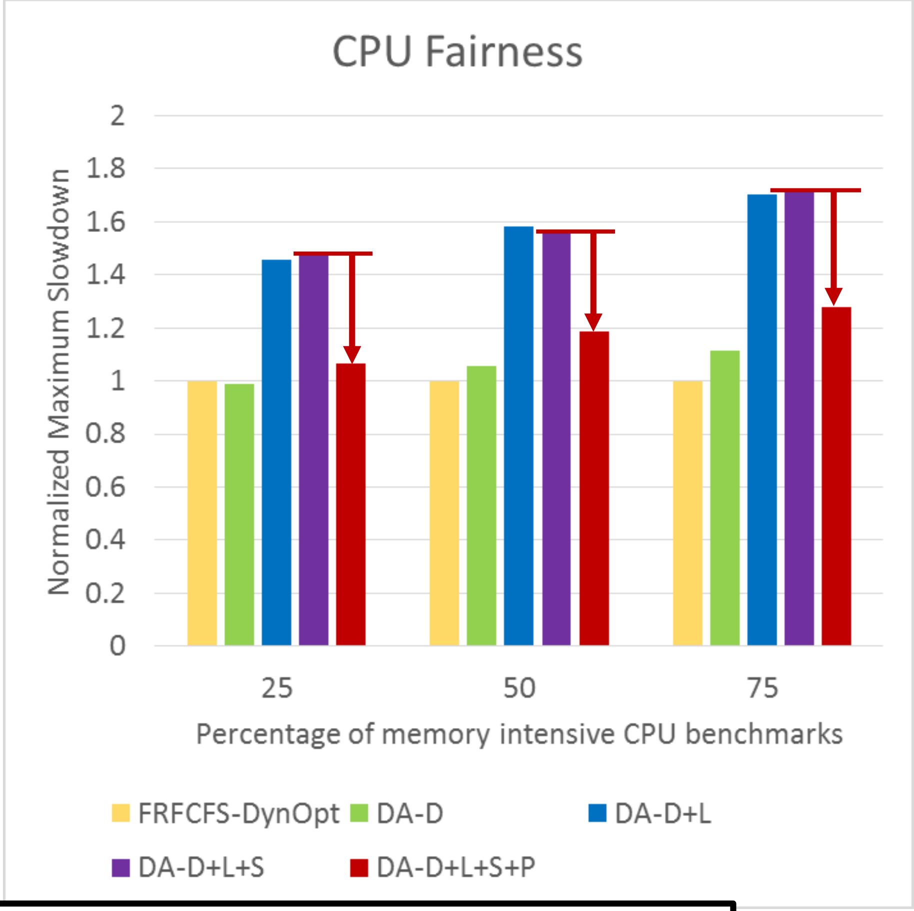


- DA-D : Distributed Priority
- DA-D+L : DA-D + application-aware priority for CPUs

Application-aware priority for CPUs improves performance especially as the memory intensity increases (Max +7.6%)

Performance breakdown of DASH





Probabilistic prioritization achieves good balance between performance and fairness

- D
- D
- D WAs
- DA-D+L+S+P (DASH) : DA-D+L+S + probabilistic prioritization

SAFARI

Performance breakdown of DASH

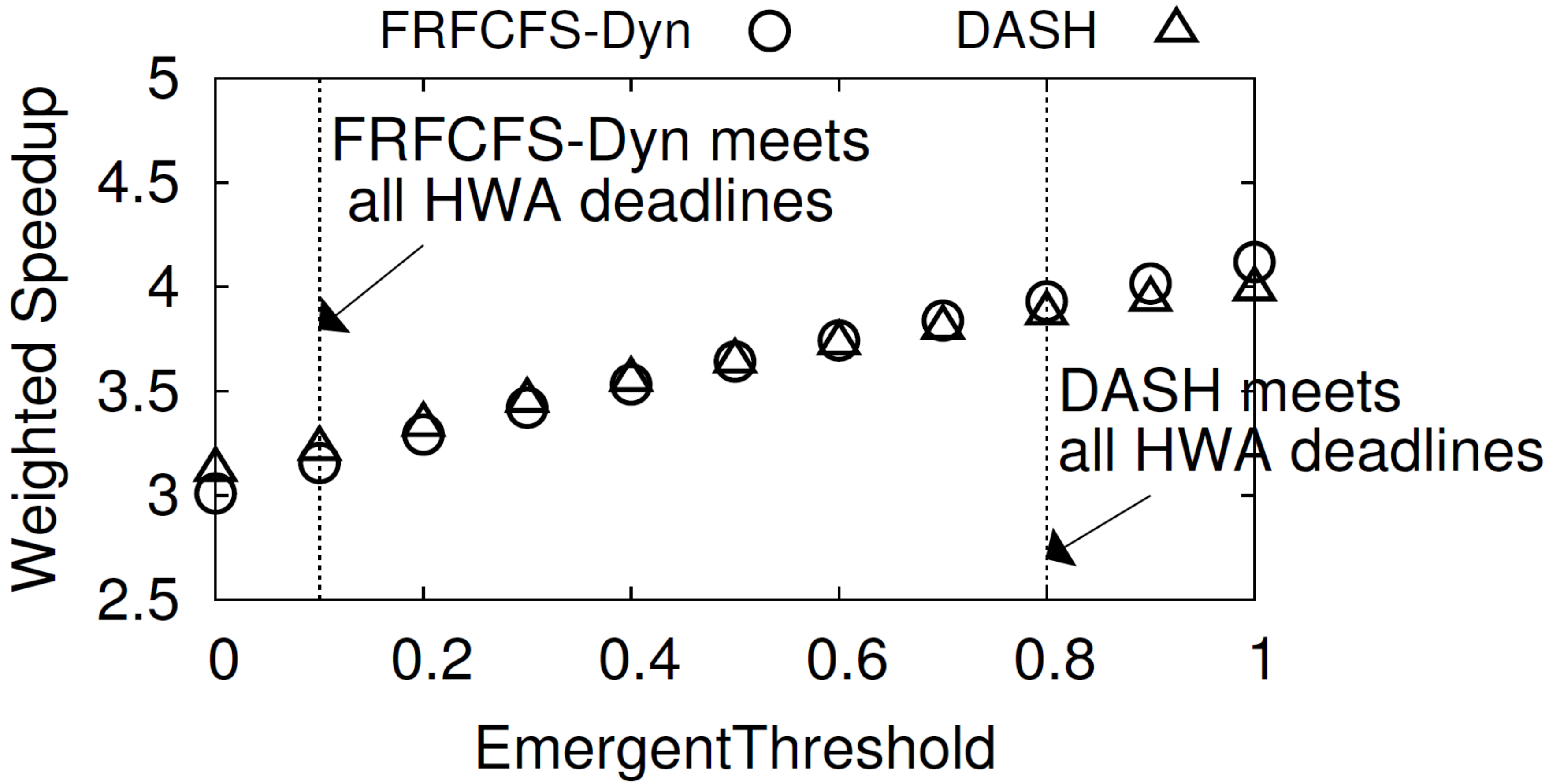
- Deadline Met Ratio

Name	IMG	HES	MAT	RSZ	DET
Deadline group	Long	Short	Long	Long	Short
FRFCFS-DynOpt	100	100	99.997	100	99.99
DA-D	100	**99.999**	100	100	**99.88**
DA-D+L	100	**99.999**	100	100	**99.87**
DA-D+L+S	100	**100**	100	100	**100**
DA-D+L+S+P	100	100	100	100	100

1. Short-deadline HWAs (HES and DET) misses deadlines on distributed priority (DA-D) and application-aware priority for CPU (DA-D+L)
2. Worst-case latency based priority (DA-D+L+S) enables short-deadline HWAs to meet their deadline

Impact of EmergentThreshold

CPU performance sensitivity to EmergentThreshold



DASH can meet all deadlines with a high EmergentThreshold value (=0.8)

Impact of EmergentThreshold

Deadline-met ratio(%) of FRFCFS-Dyn

Emergent Threshold	Config-A		Config-B			
	HES	MAT	HES	MAT	RSZ	DET
0–0.1	100	100	100	100	100	100
0.2	100	99.987	100	100	100	100
0.3	99.992	93.74	100	100	100	100
0.4	99.971	73.179	100	100	100	100
0.5	99.945	55.76	99.9996	99.751	100	99.997
0.6	99.905	44.691	99.989	94.697	100	99.96
0.7	99.875	38.097	99.957	86.366	100	99.733
0.8	99.831	34.098	99.906	74.69	99.886	99.004
0.9	99.487	31.385	99.319	60.641	97.977	97.149
1	96.653	27.32	95.798	33.449	55.773	88.425

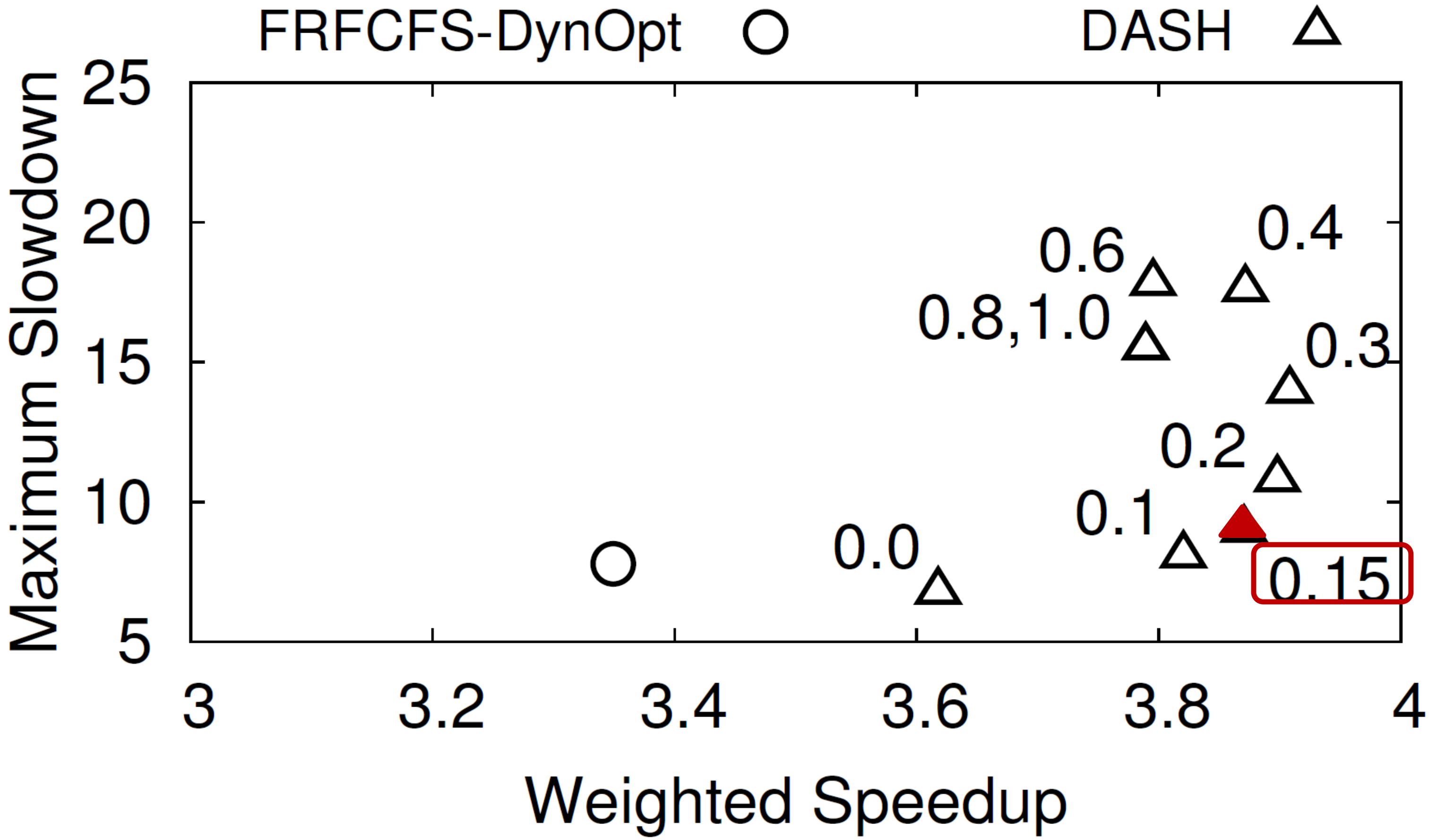
Impact of EmergentThreshold

Deadline-met ratio(%) of DASH

Emergent Threshold	Config-A		Config-B			
	HES	MAT	HES	MAT	RSZ	DET
0–0.8	100	100	100	100	100	100
0.9	100	99.997	100	99.993	100	100
1	100	68.44	100	75.83	95.93	100

Impact of ClusterFactor

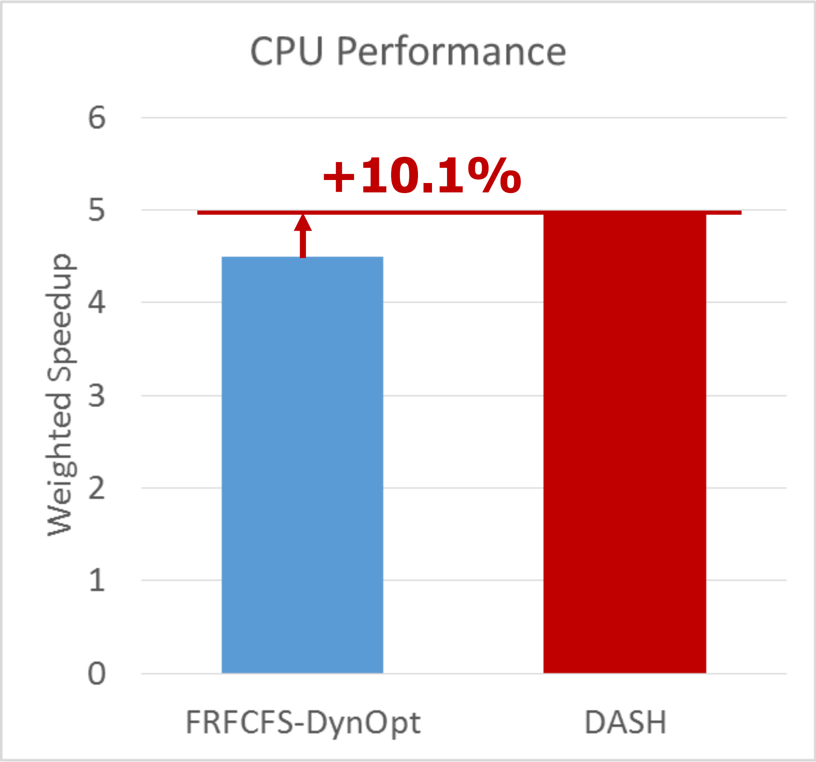
CPU performance sensitivity to Cluster Factor

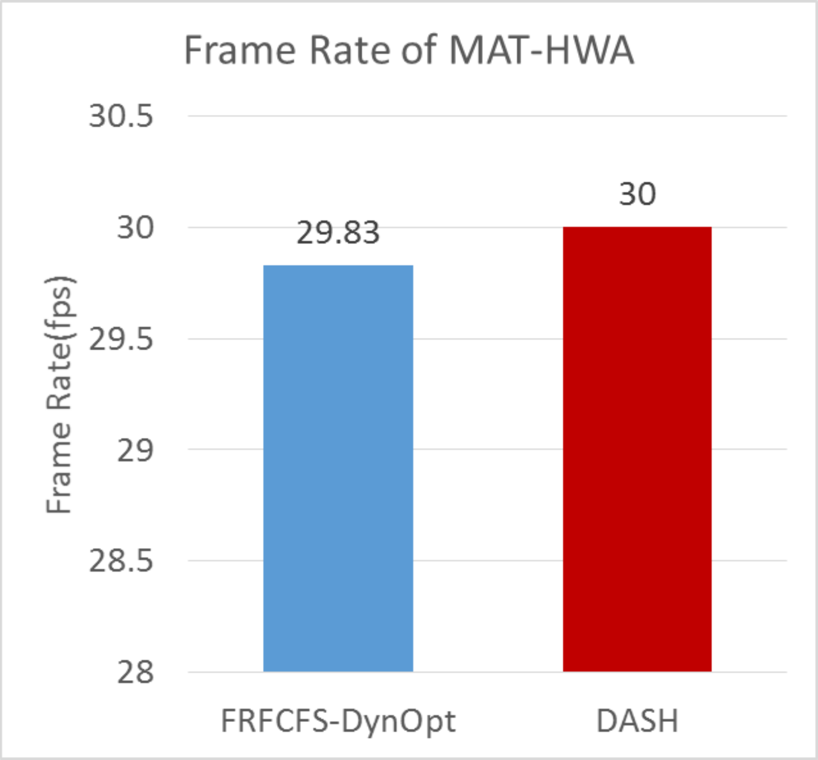


ClusterFactor is an effective knob for trading off CPU performance and fairness

Evaluations with GPUs

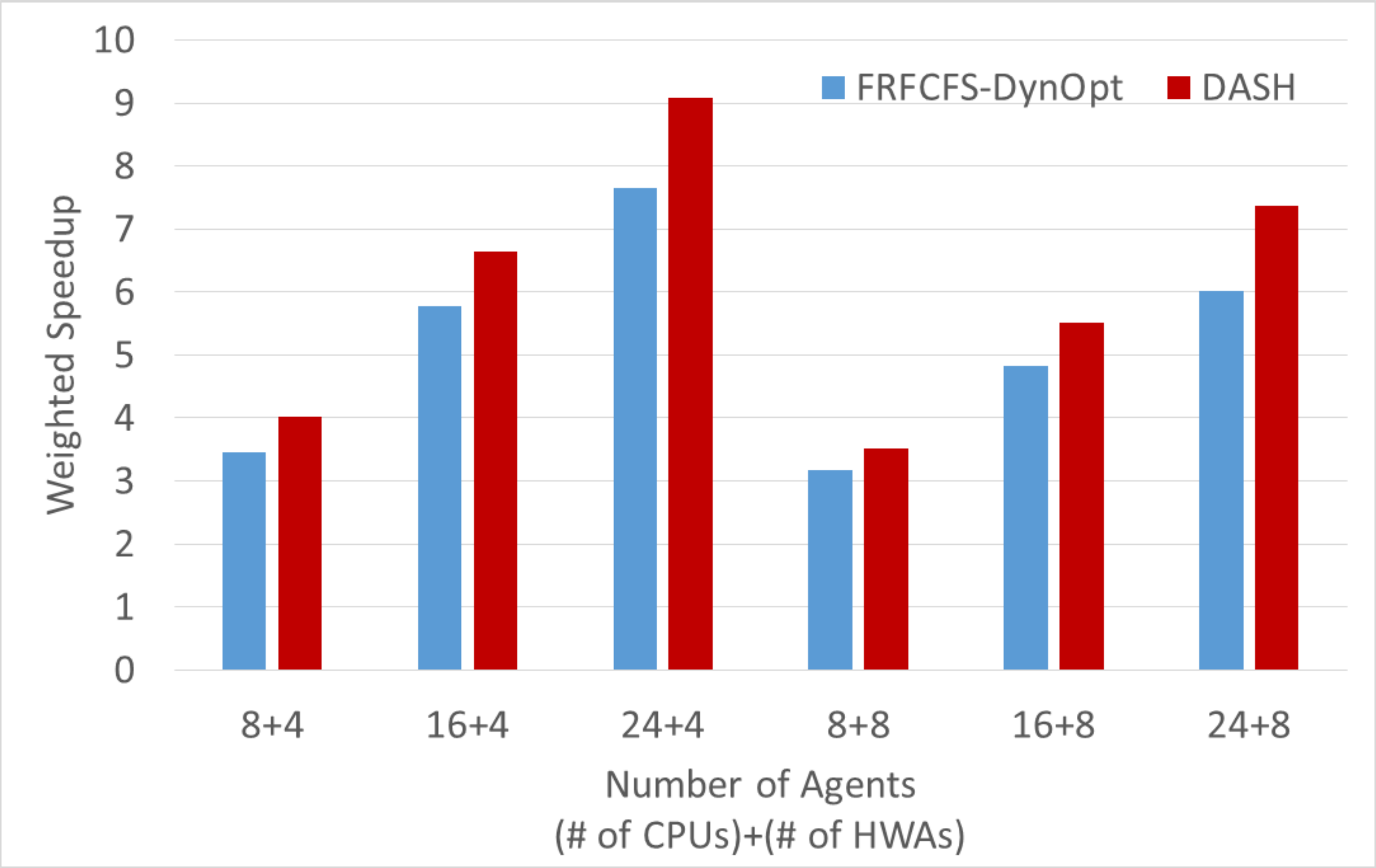
- 8 CPUs + 4 HWA (Config-A) + GPU
  - 6 GPU traces : 3D mark and game





SAFARI

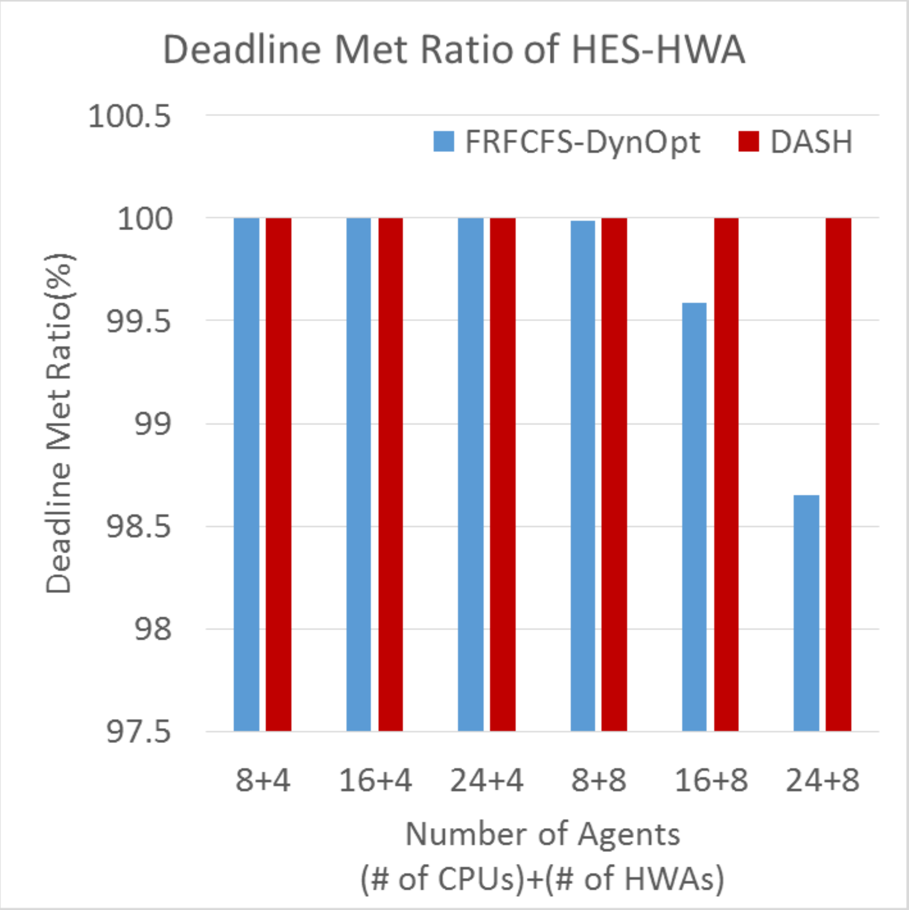
Sensitivity to Number of Agents

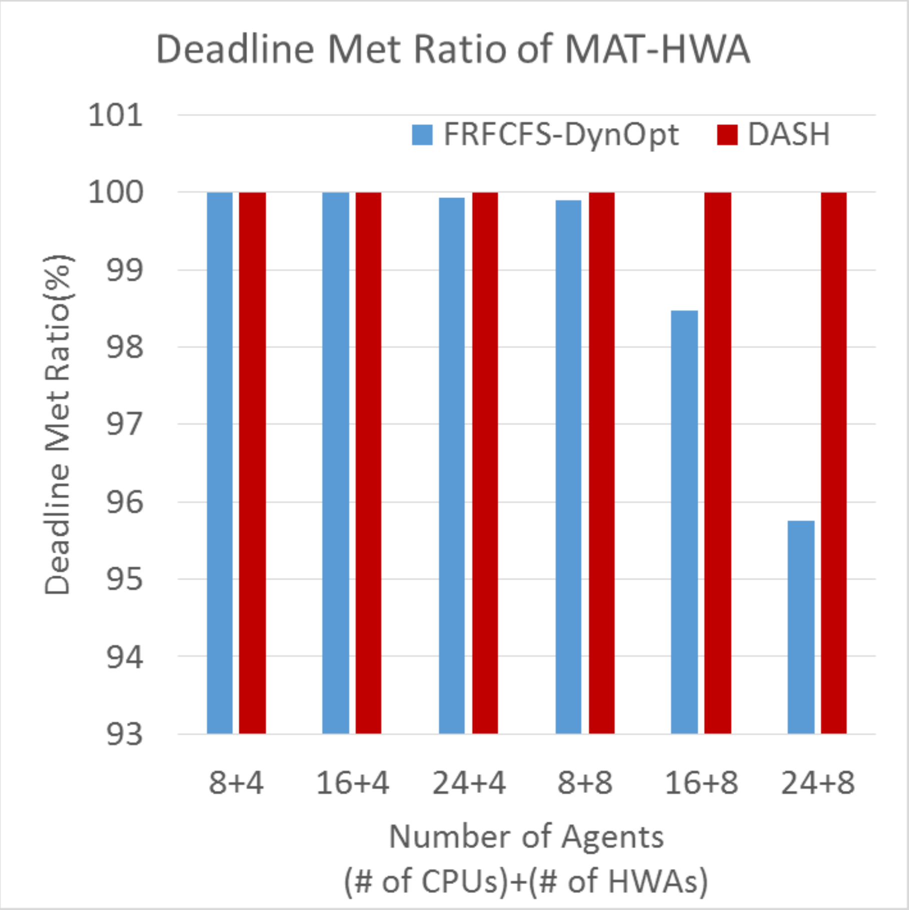


As the number of agents increases, DASH achieves greater performance improvement

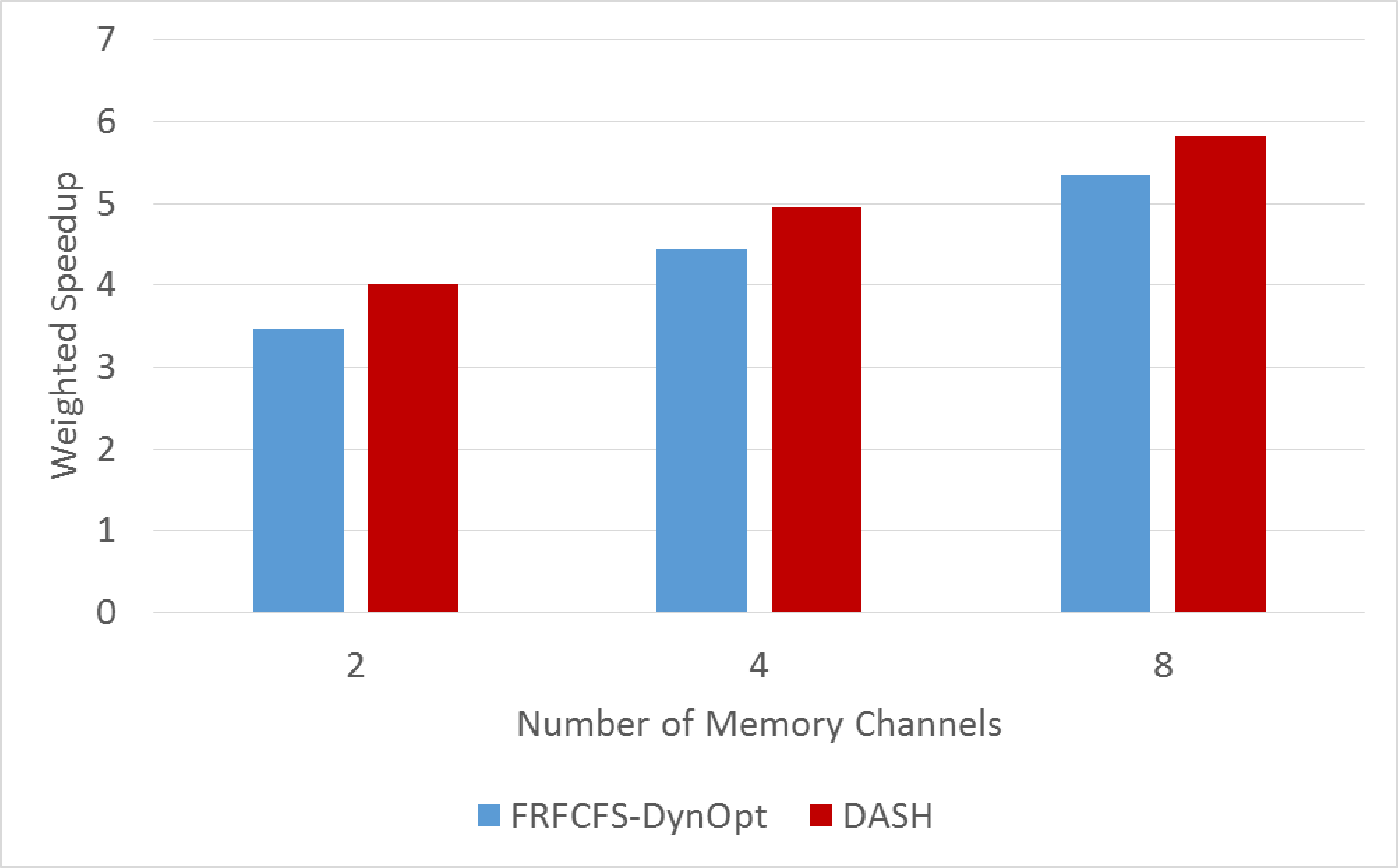
8HWAs : IMG x 2, MAT x 2, HES x 2, RSZ x 1, DET x 1

Sensitivity to Number of Agents





Sensitivity to Number of Channels



DASH: Application-aware scheduling for HWAs

- Categorize HWAs as long-deadline-period vs. short-deadline-period statically

- Adjust the priorities of each dynamically

  - Short-deadline-period HWA: becomes high priority if *time remaining in period = tRC x NumberOfRequests + a*

  - Long-deadline-period HWA: becomes high priority if *Current progress ≤ Expected progress*