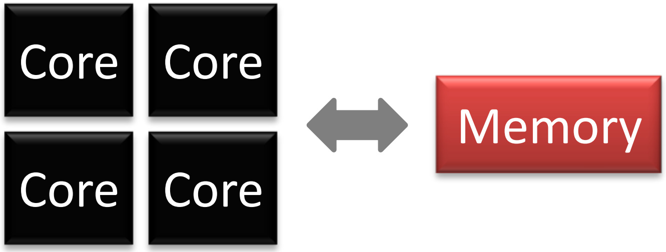
Thread Cluster Memory Scheduling:
Exploiting Differences in Memory Access Behavior

Yoongu Kim
Michael Papamichael
Onur Mutlu
Mor Harchol-Balter

Carnegie Mellon

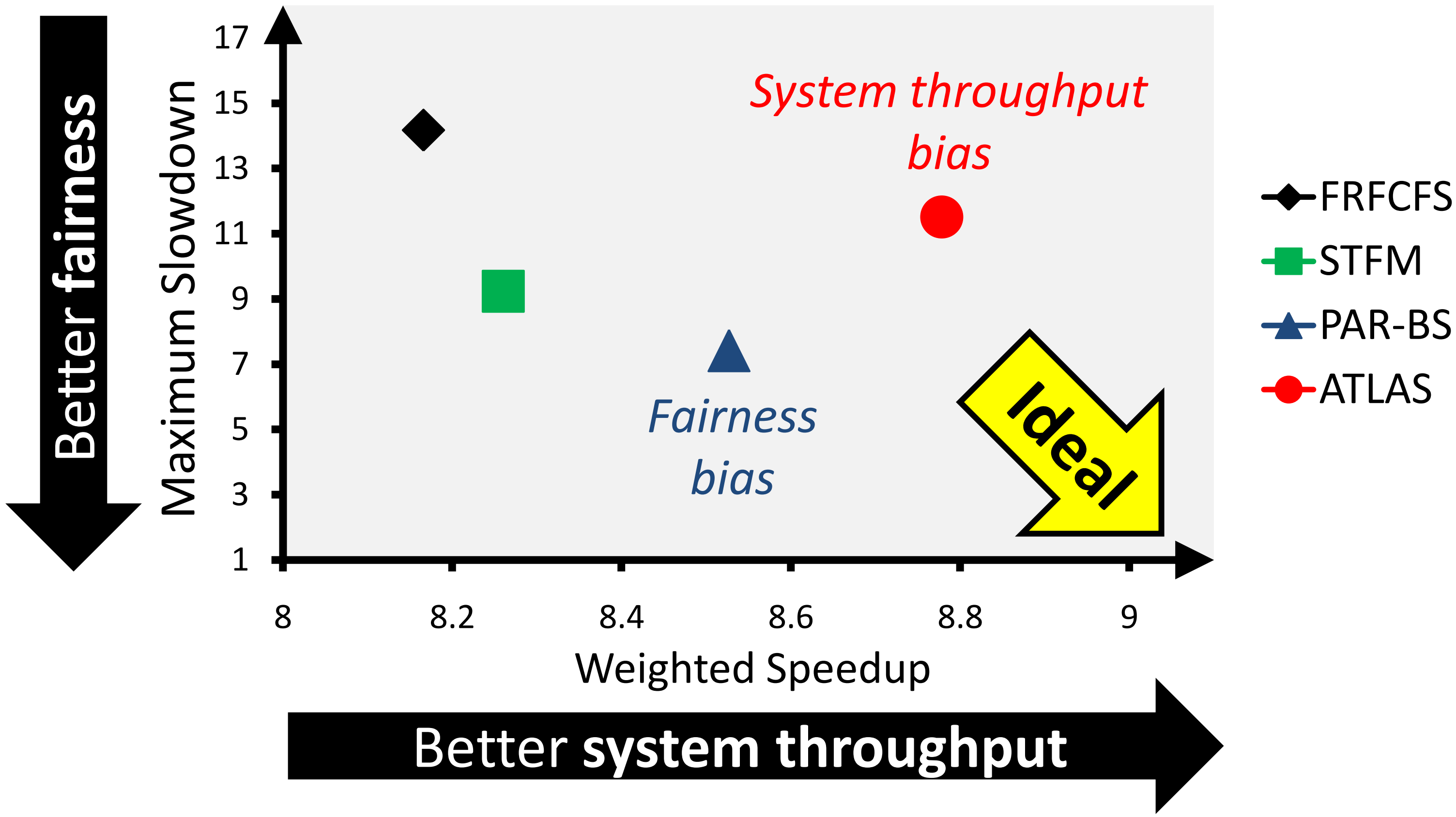
Motivation

- Memory is a shared resource



- Threads' requests contend for memory
  - Degradation in single thread performance
  - Can even lead to starvation
- How to schedule memory requests to increase both system throughput and fairness?

Previous Scheduling Algorithms are Biased

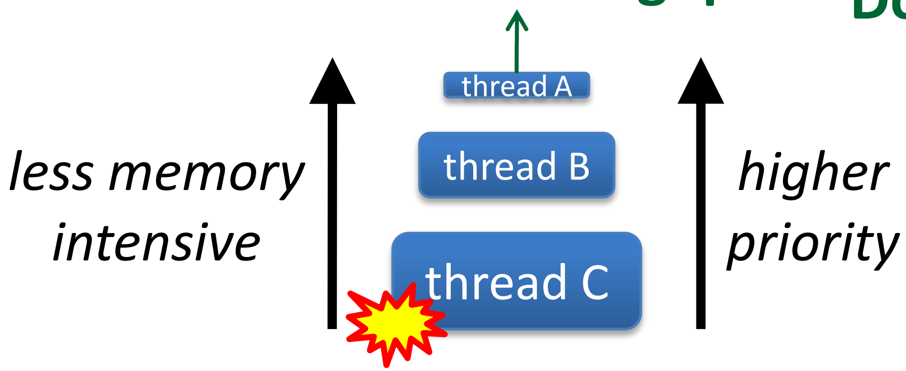


No previous memory scheduling algorithm provides both the best fairness and system throughput

Why do Previous Algorithms Fail?

***Throughput biased** approach*	***Fairness biased** approach*
Prioritize less memory-intensive threads	Take turns accessing memory

Good for throughput



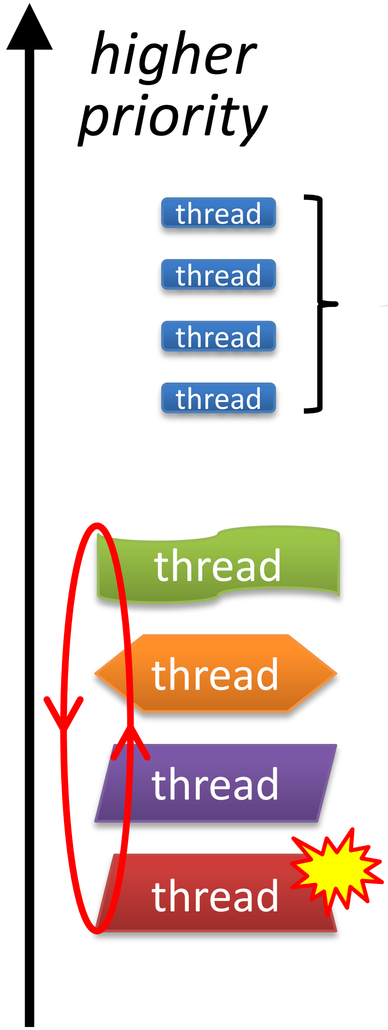
starvation → unfairness

Does not starve

not prioritized → reduced throughput

Single policy for all threads is insufficient

Insight: Achieving Best of Both Worlds



For Throughput

Prioritize memory-non-intensive threads

For Fairness

Unfairness caused by memory-intensive being prioritized over each other

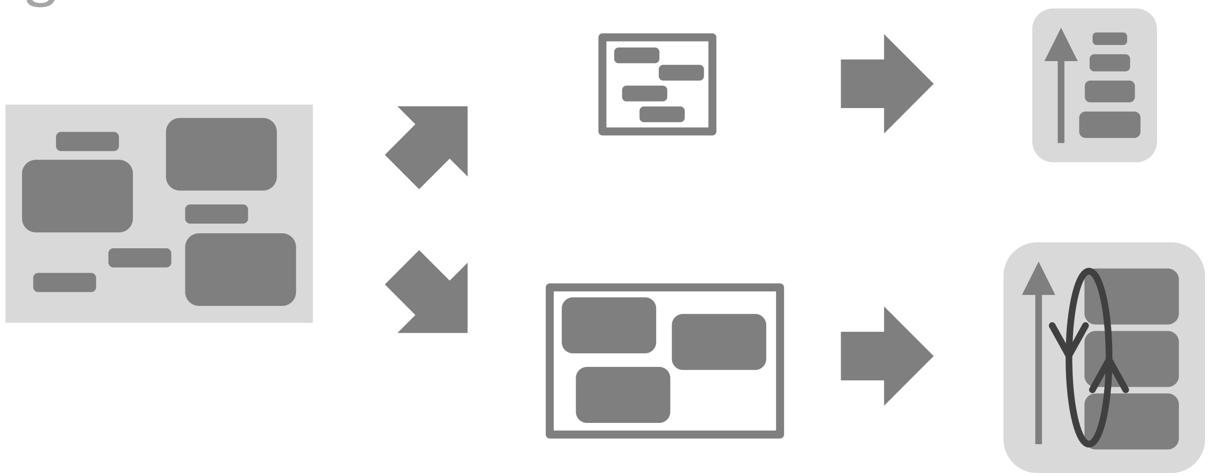
- Shuffle threads

Memory-intensive threads have different vulnerability to interference

- Shuffle <u>asymmetrically</u>

Outline

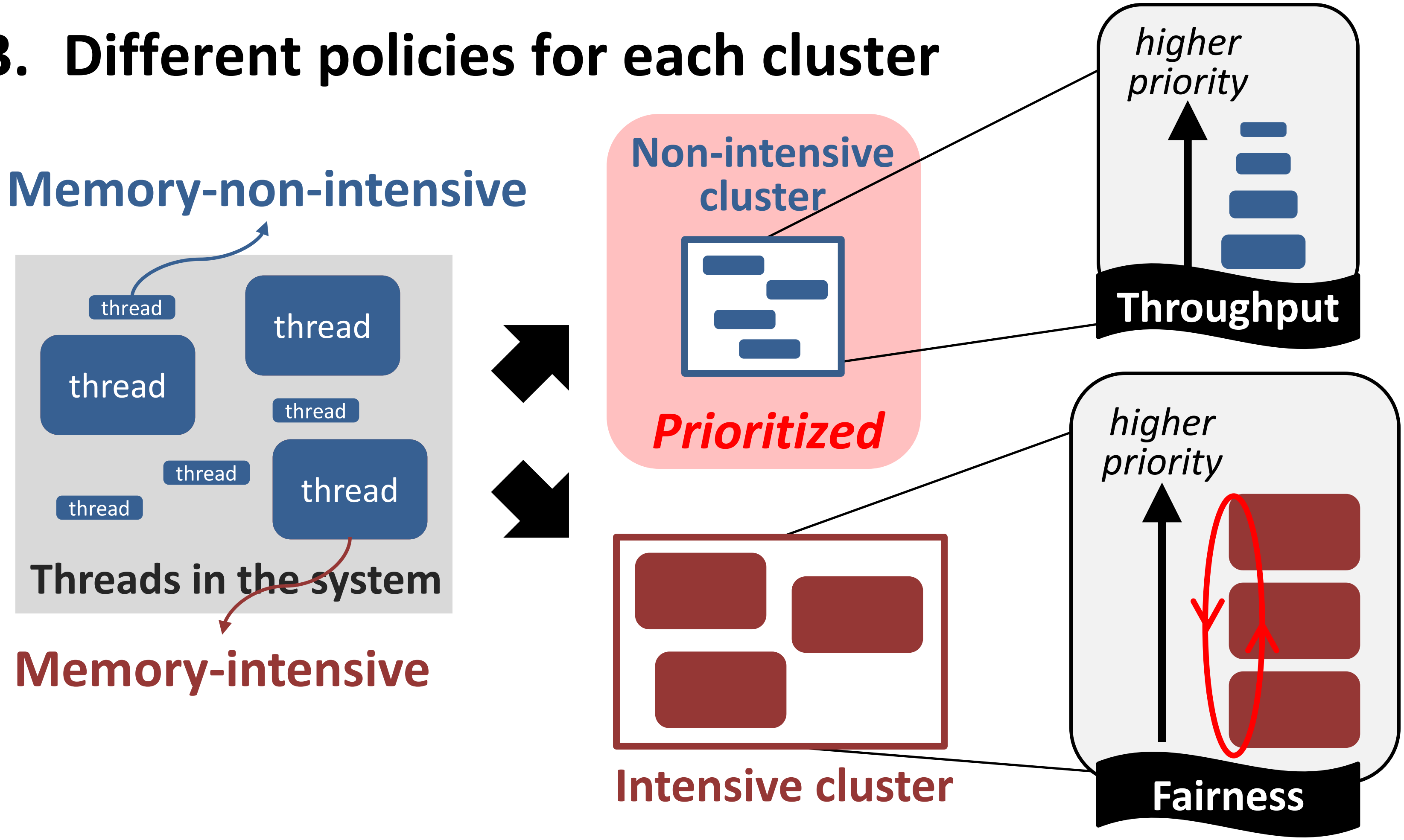
- ❑ Motivation & Insights
- ❑ **Overview**
- ❑ Algorithm



- ❑ Bringing it All Together
- ❑ Evaluation
- ❑ Conclusion

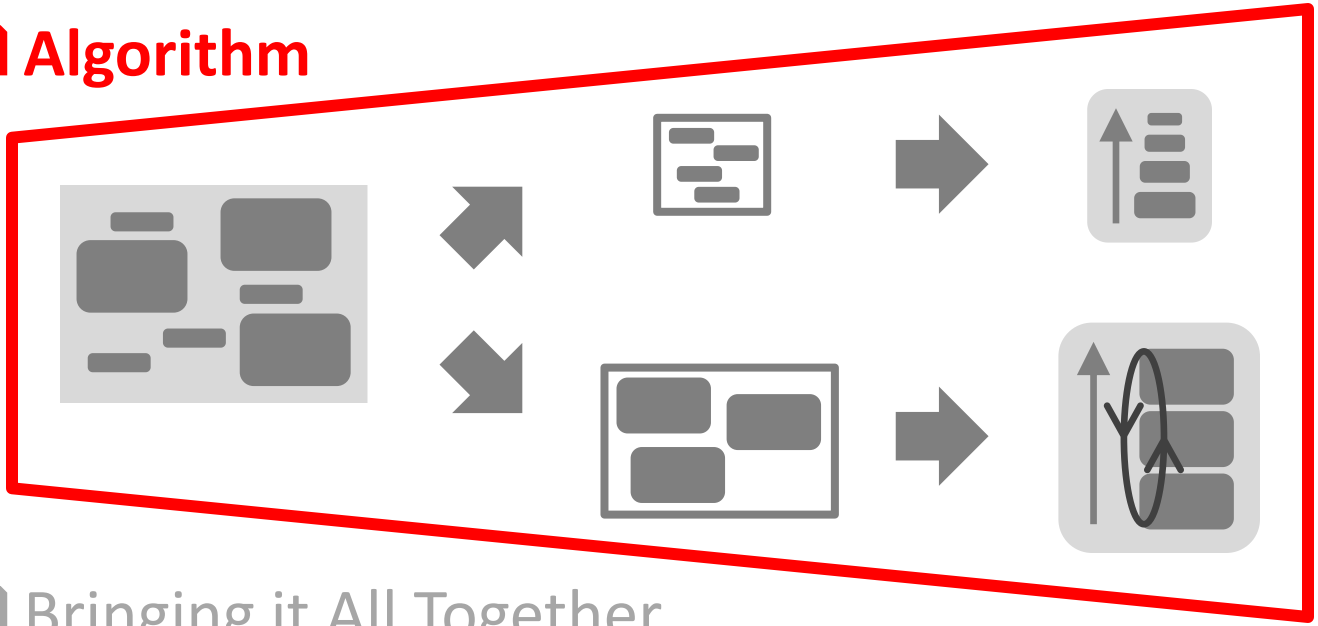
Overview: Thread Cluster Memory Scheduling

1. Group threads into two ***clusters***
2. Prioritize **non-intensive cluster**
3. Different policies for each cluster



Outline

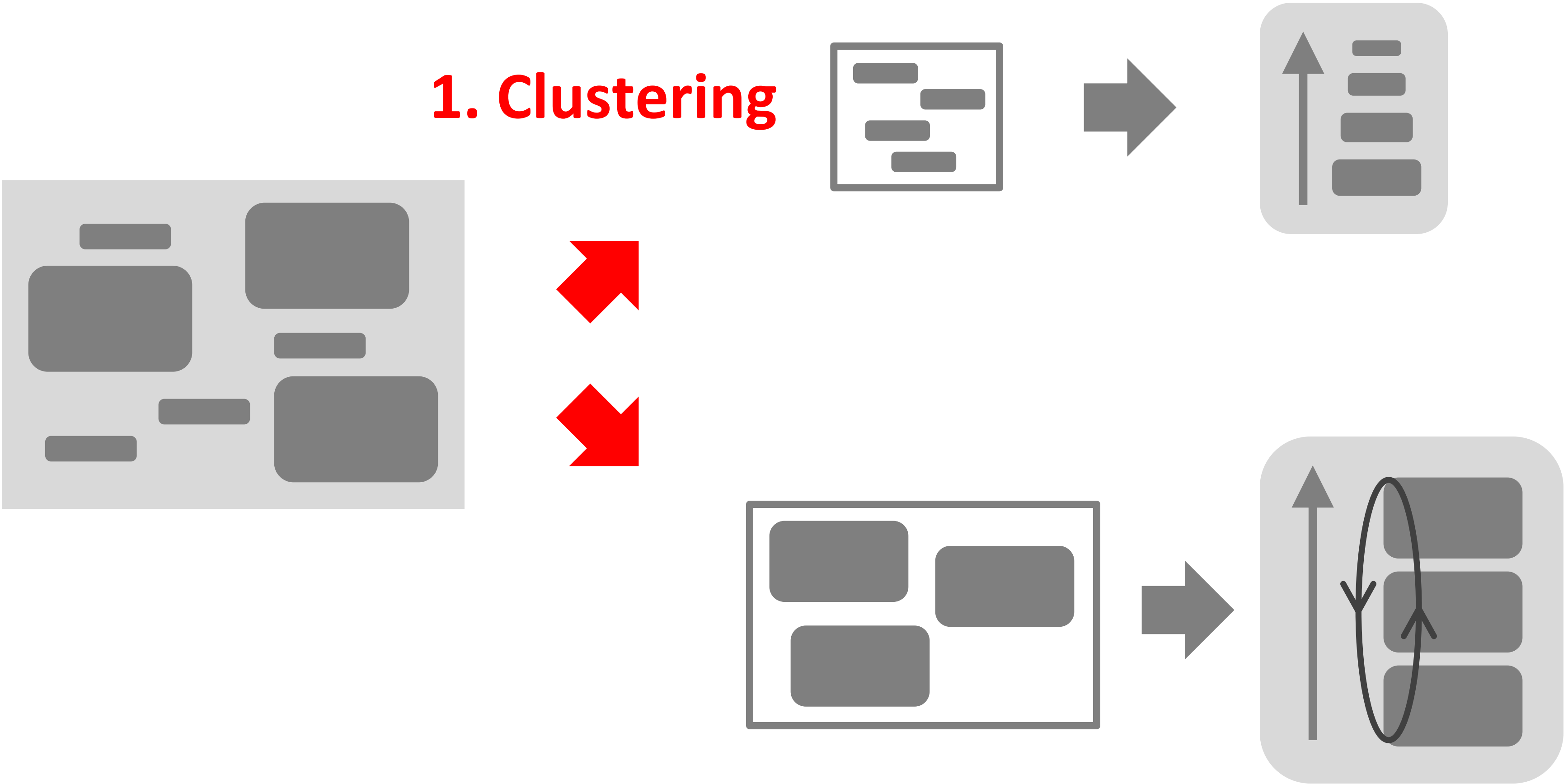
- ❑ Motivation & Insights
- ❑ Overview
- ❑ **Algorithm**



- ❑ Bringing it All Together
- ❑ Evaluation
- ❑ Conclusion

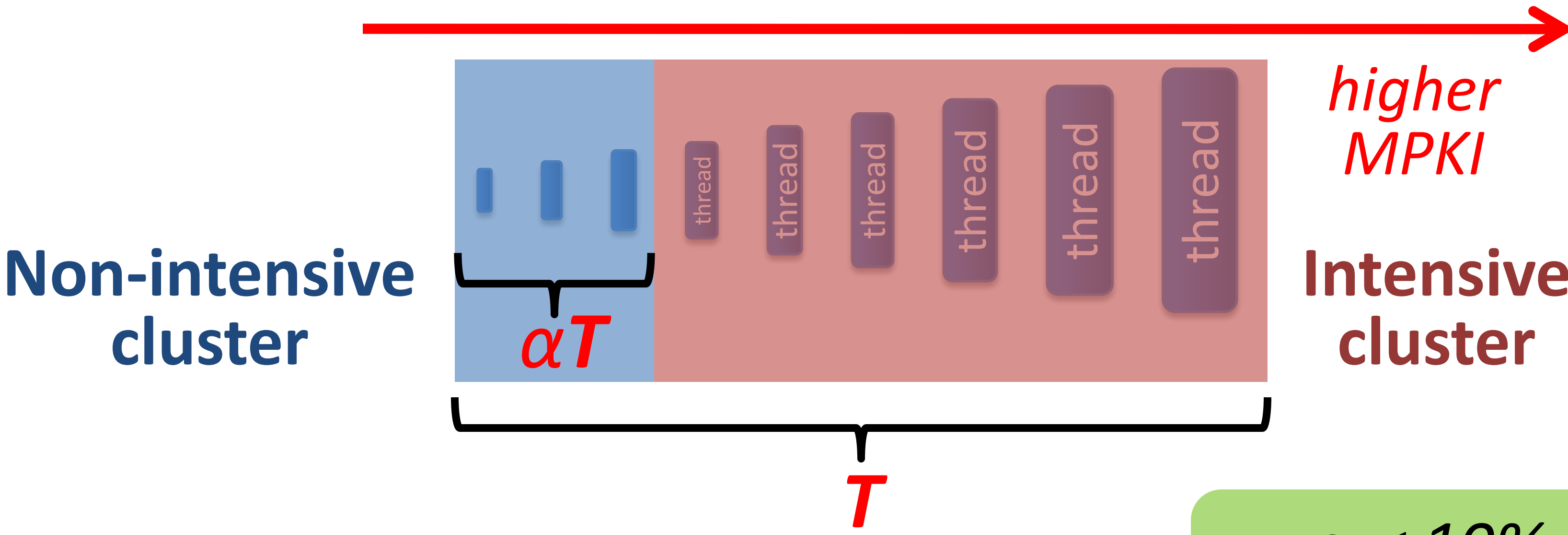
TCM Outline

1. Clustering



Clustering Threads

Step1 Sort threads by **MPKI** (misses per kiloinstruction)



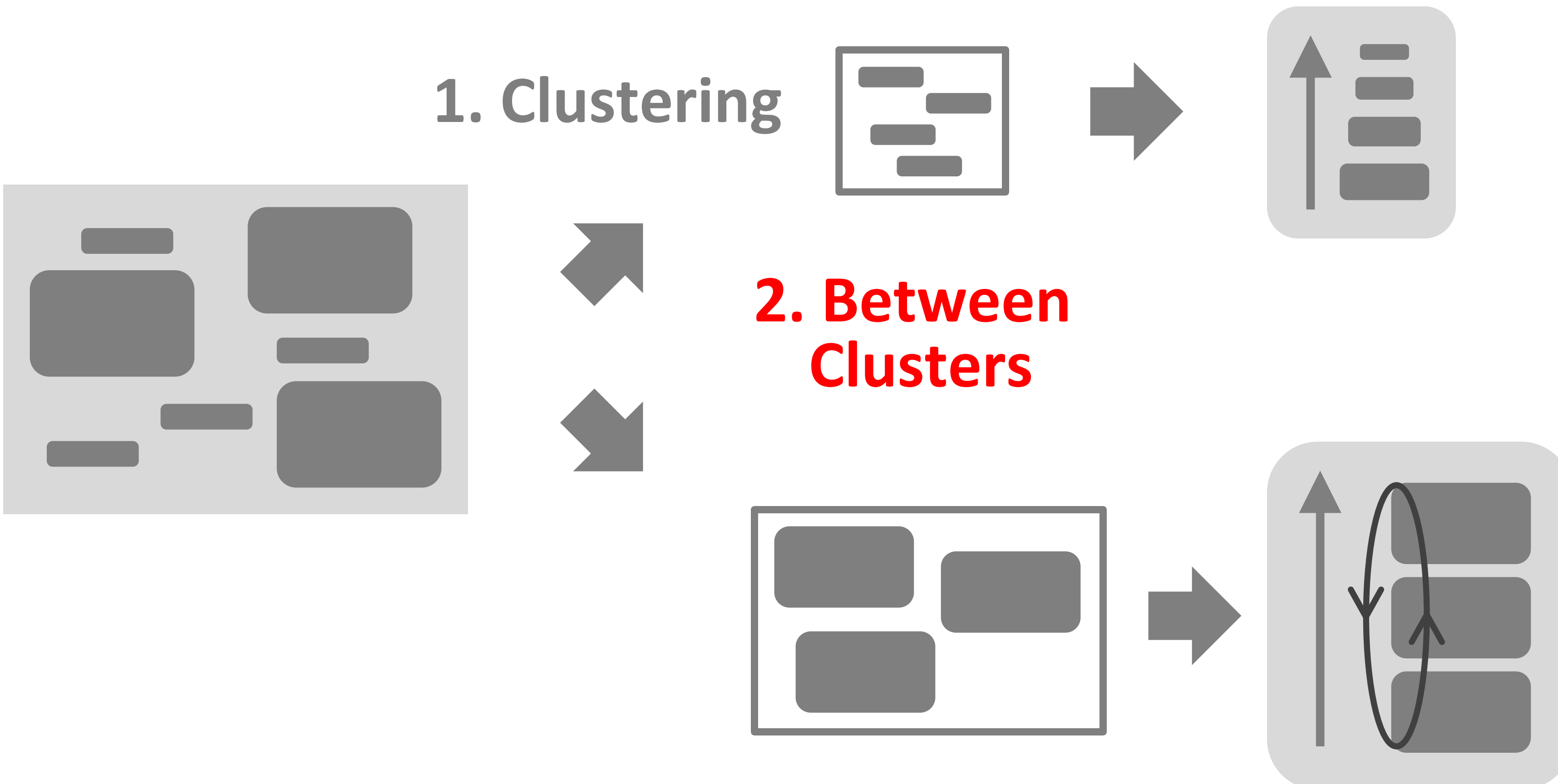
T **=** Total ***memory bandwidth usage***

***α** < 10%*

ClusterThreshold

Step2 Memory bandwidth usage ***αT*** divides clusters

TCM Outline



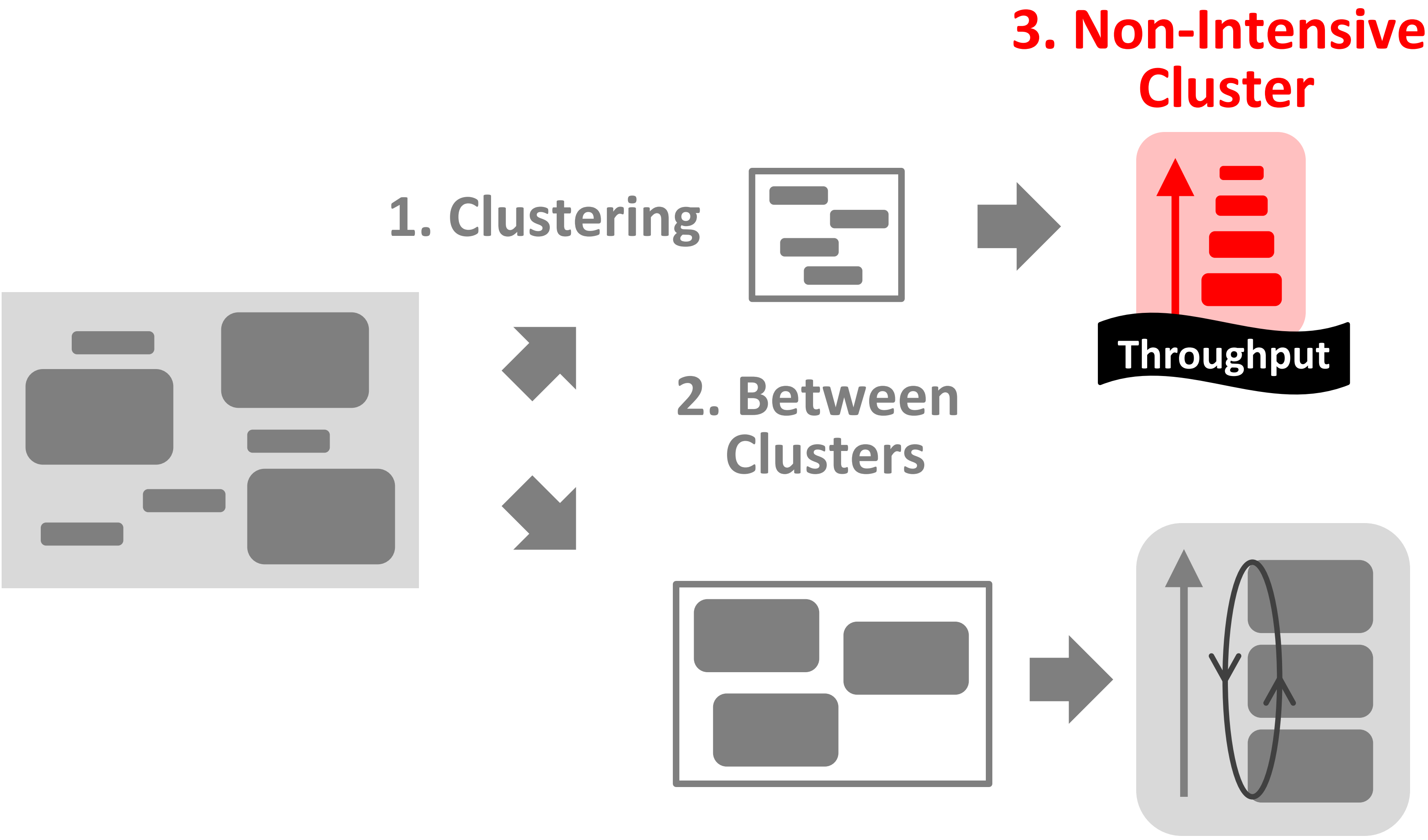
Prioritization Between Clusters

Prioritize non-intensive cluster

- **Increases system throughput**
  – Non-intensive threads have greater potential for making progress

- **Does not degrade fairness**
  – Non-intensive threads are "light"
  – Rarely interfere with intensive threads

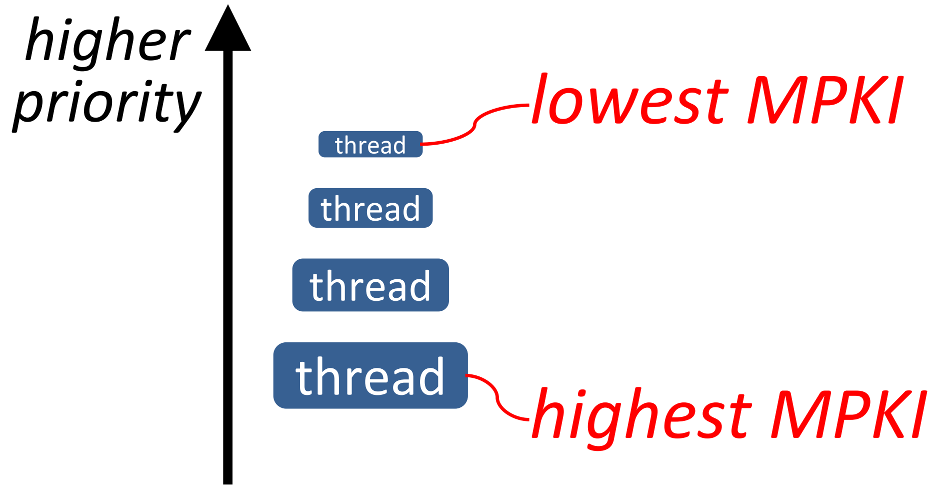
TCM Outline



3. Non-Intensive Cluster
1. Clustering
Throughput
2. Between Clusters

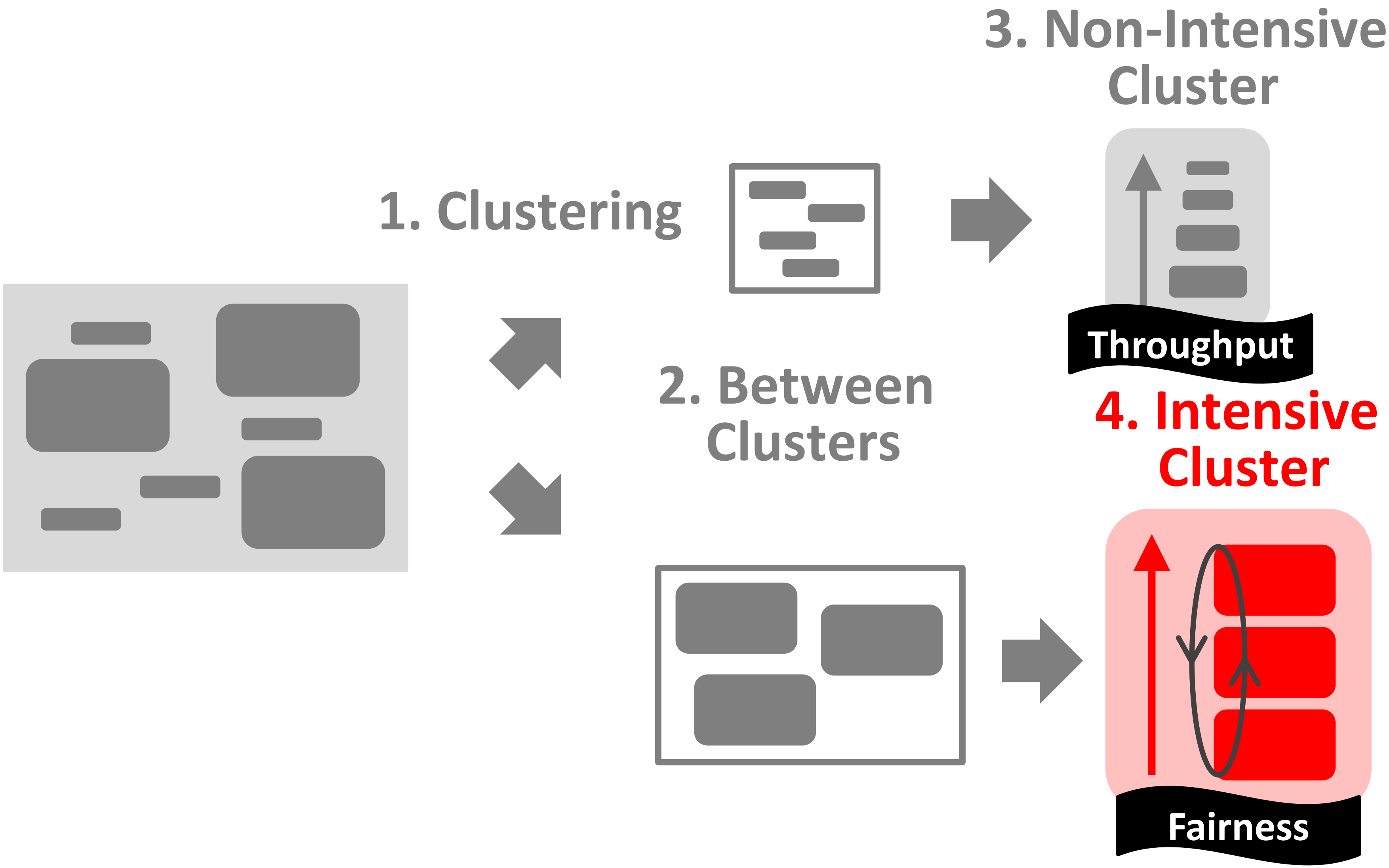
Non-Intensive Cluster

Prioritize threads according to MPKI



- **Increases system throughput**
  - Least intensive thread has the greatest potential for making progress in the processor

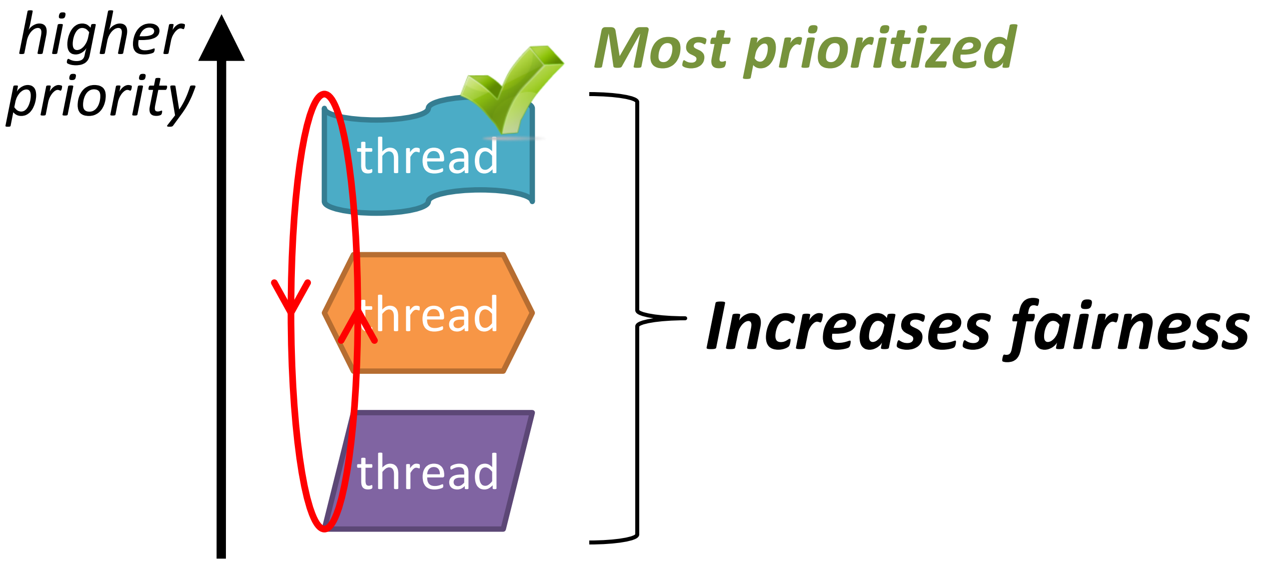
TCM Outline



1. Clustering
2. Between Clusters
3. Non-Intensive Cluster
Throughput
4. Intensive Cluster
Fairness

Intensive Cluster

Periodically shuffle the priority of threads



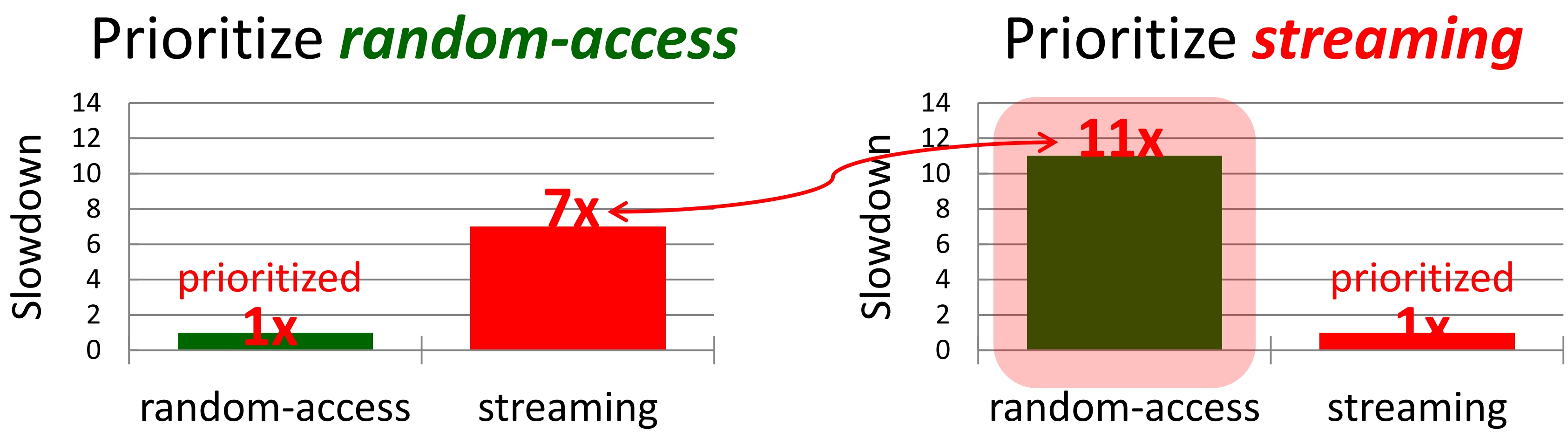
- Is treating all threads equally good enough?
- ***BUT:*** *Equal turns ≠ Same slowdown*

Case Study: A Tale of Two Threads

Case Study: Two intensive threads contending

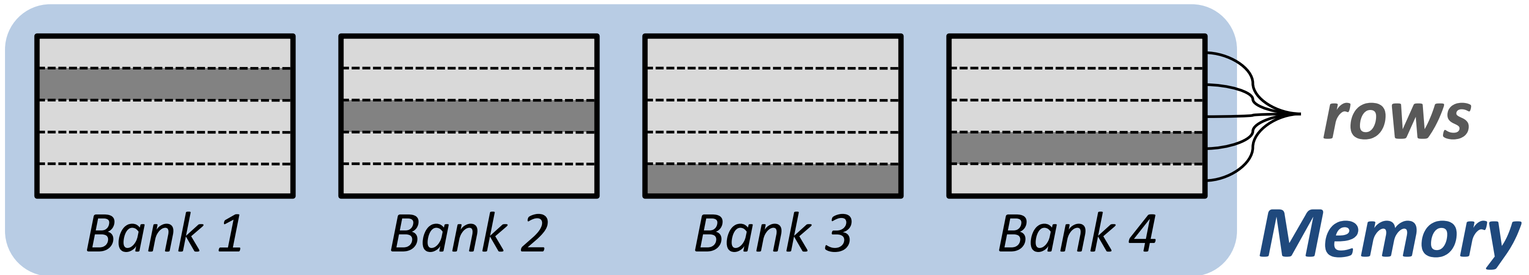
1. ***random-access***
2. ***streaming***

Which is slowed down more easily?



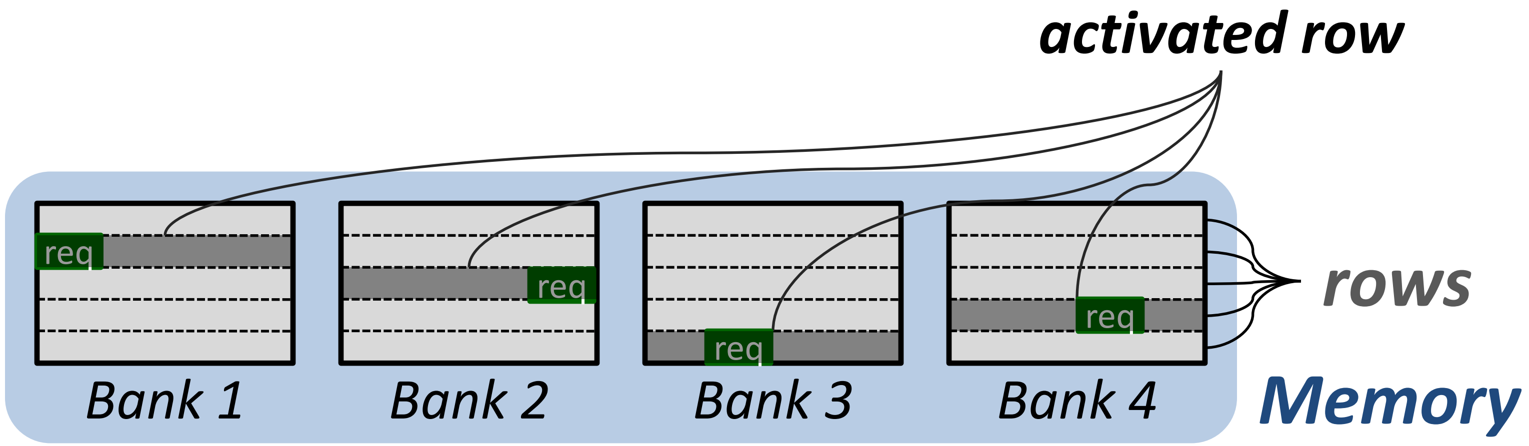
random-access thread is more easily slowed down

Why are Threads Different?



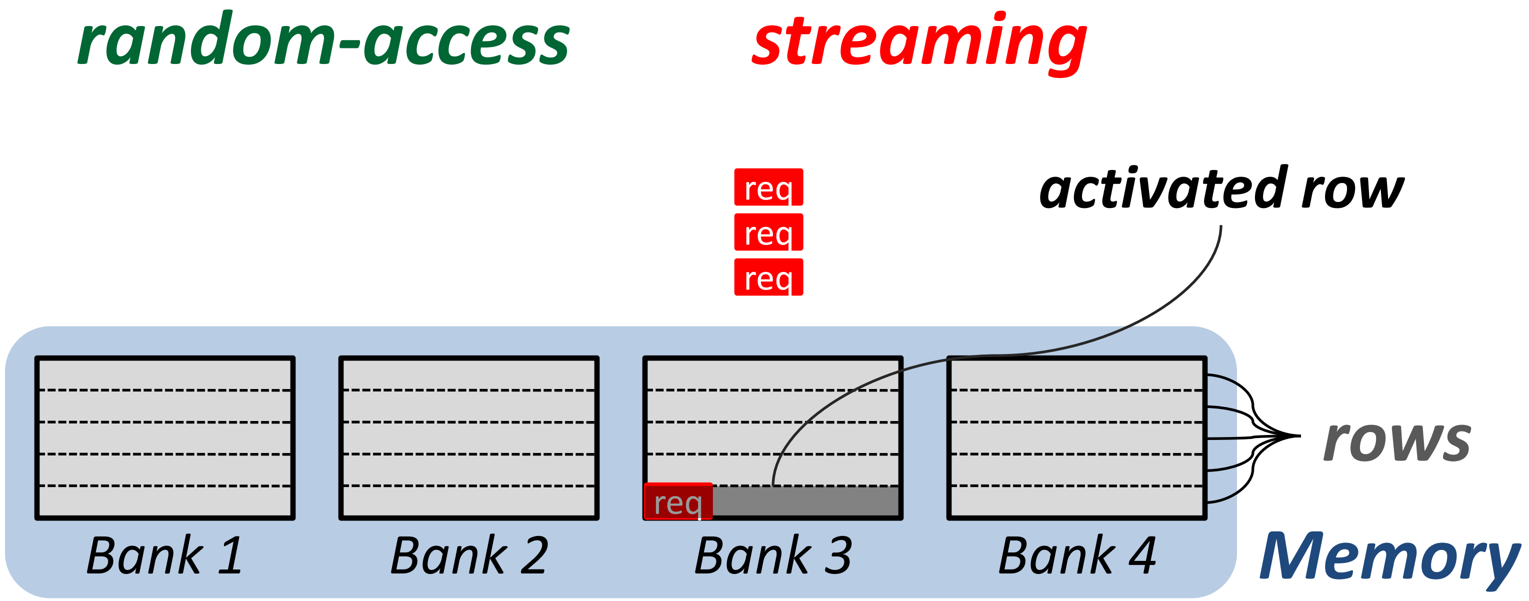
Why are Threads Different?

random-access



- All requests parallel
- High **bank-level parallelism**

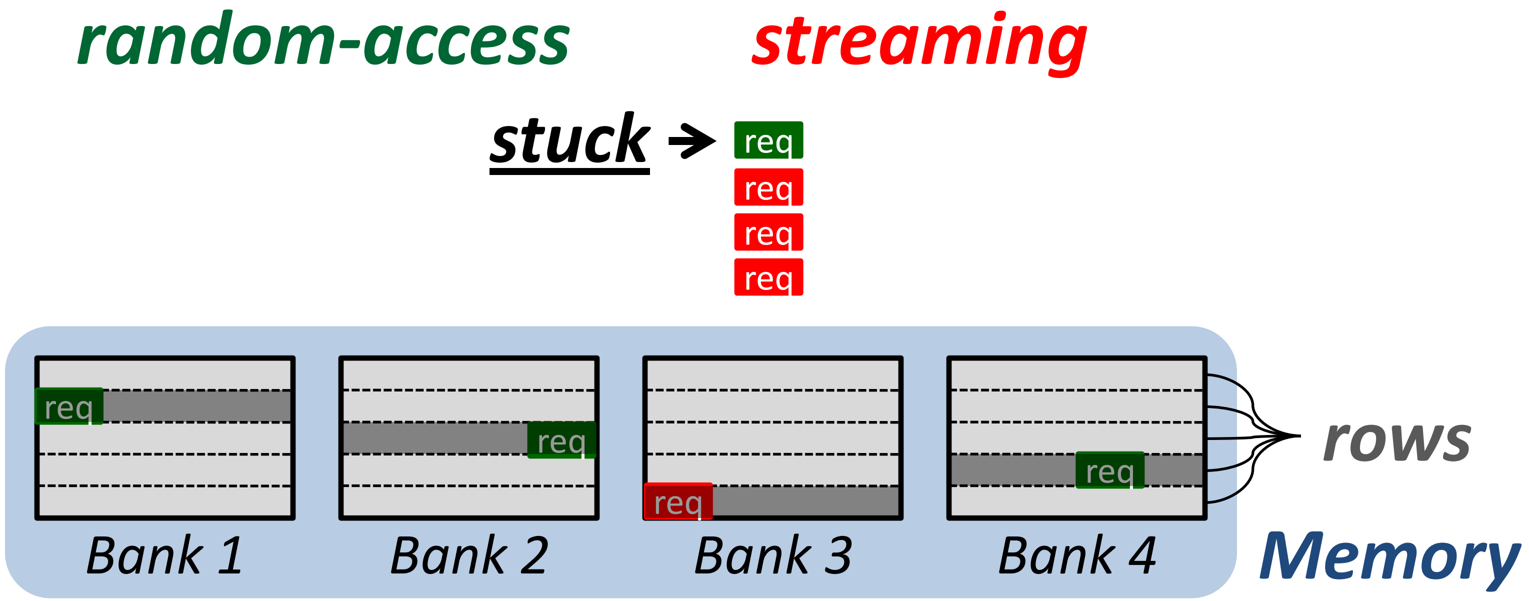
Why are Threads Different?



- All requests parallel
- High **bank-level parallelism**

- All requests ➔ Same row
- High **row-buffer locality**

Why are Threads Different?

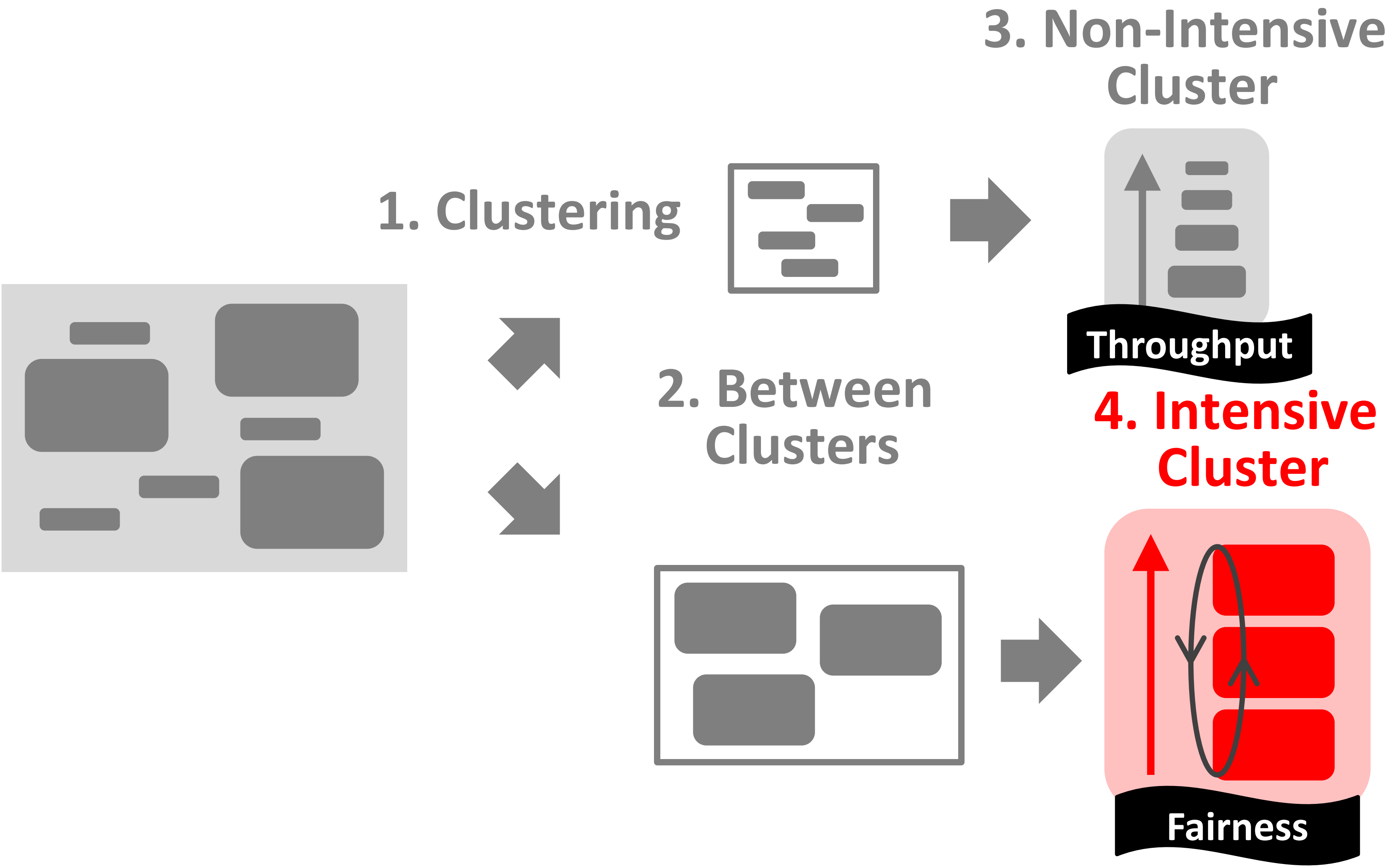


- All requests parallel
- High **bank-level parallelism**

- All requests → Same row
- High **row-buffer locality**

Vulnerable to interference

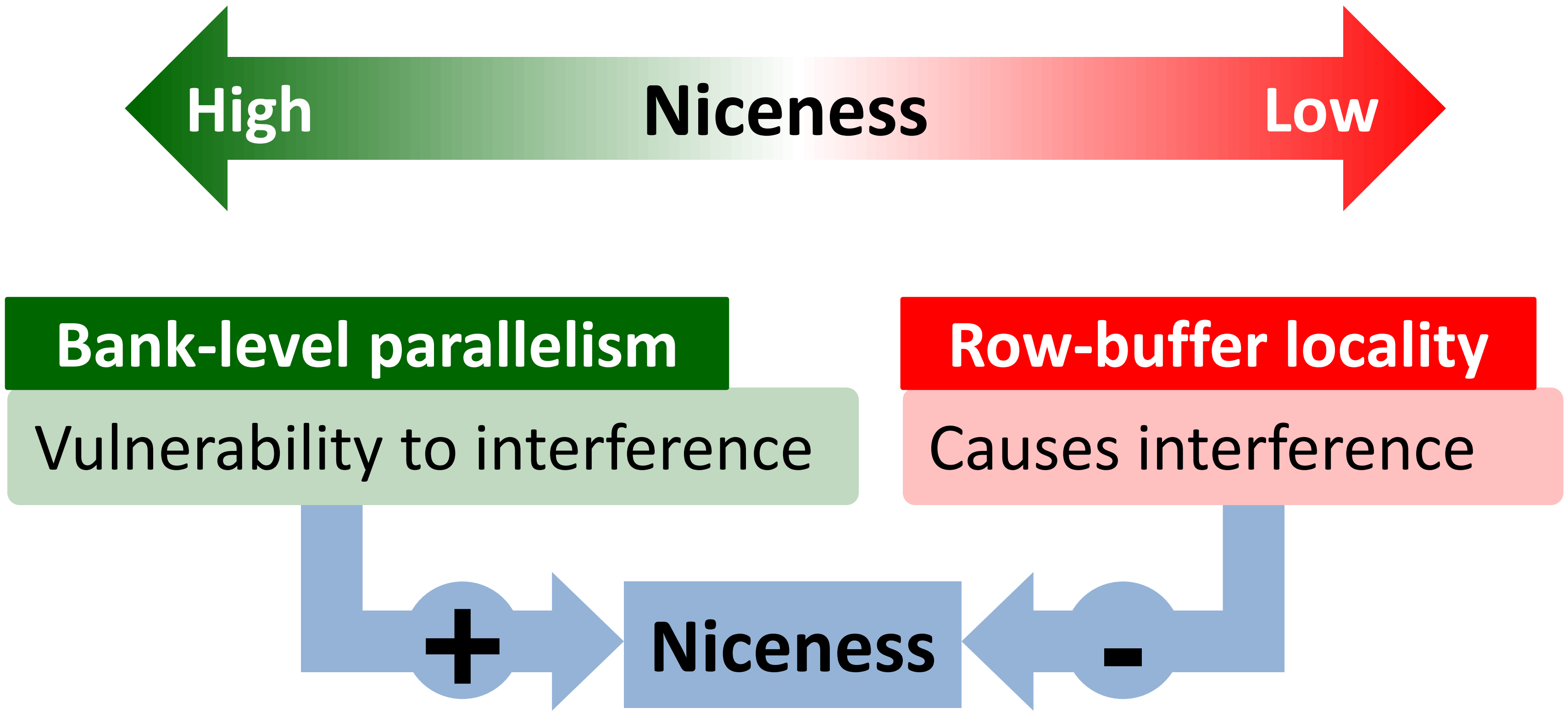
TCM Outline



3. Non-Intensive Cluster
1. Clustering
Throughput
2. Between Clusters
4. Intensive Cluster
Fairness

Niceness

How to quantify difference between threads?

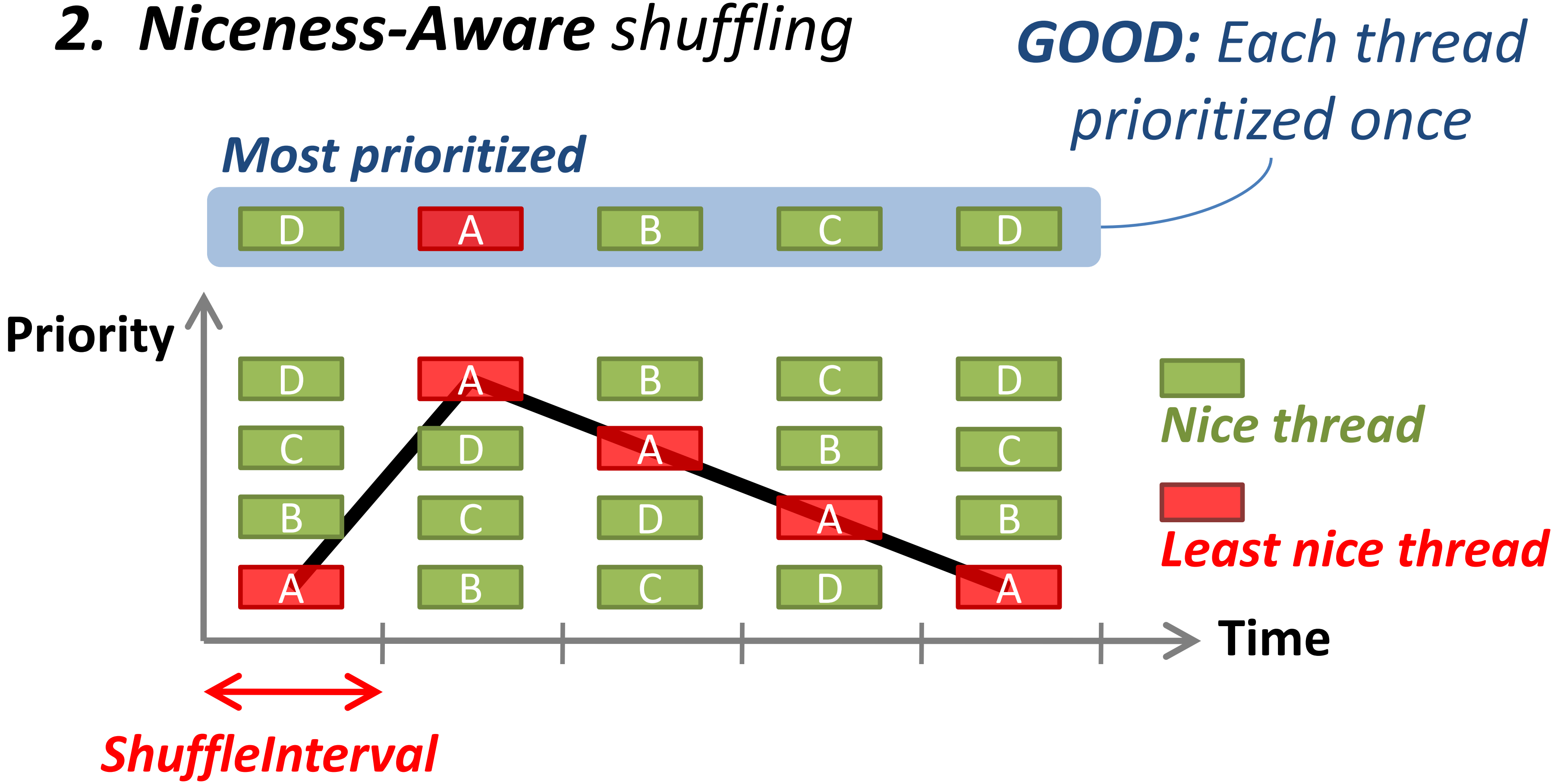


Shuffling: Round-Robin vs. Niceness-Aware

1. ***Round-Robin*** *shuffling* *What can go wrong?*
2. ***Niceness-Aware*** *shuffling*

Shuffling: Round-Robin vs. Niceness-Aware

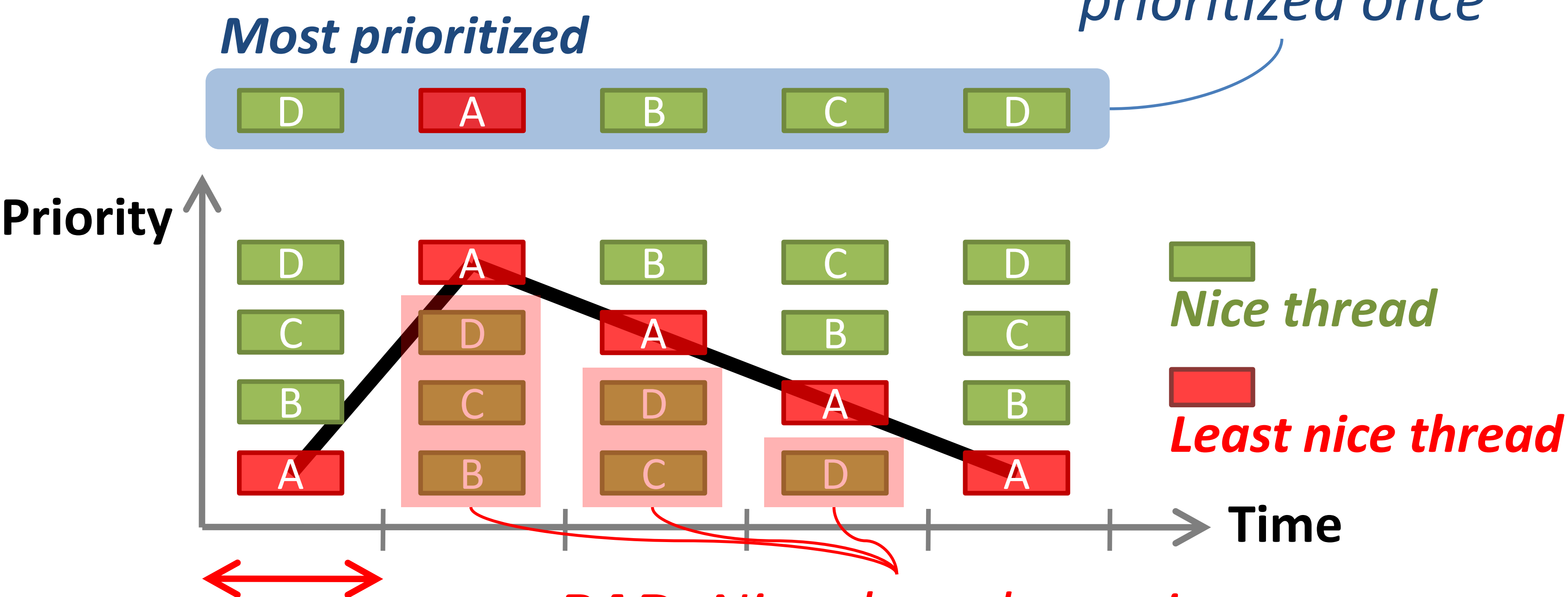
1. ***Round-Robin** shuffling* ← *What can go wrong?*
2. ***Niceness-Aware** shuffling*



Shuffling: Round-Robin vs. Niceness-Aware

1. ***Round-Robin*** *shuffling* ← *What can go wrong?*
2. ***Niceness-Aware*** *shuffling*

GOOD: *Each thread prioritized once*



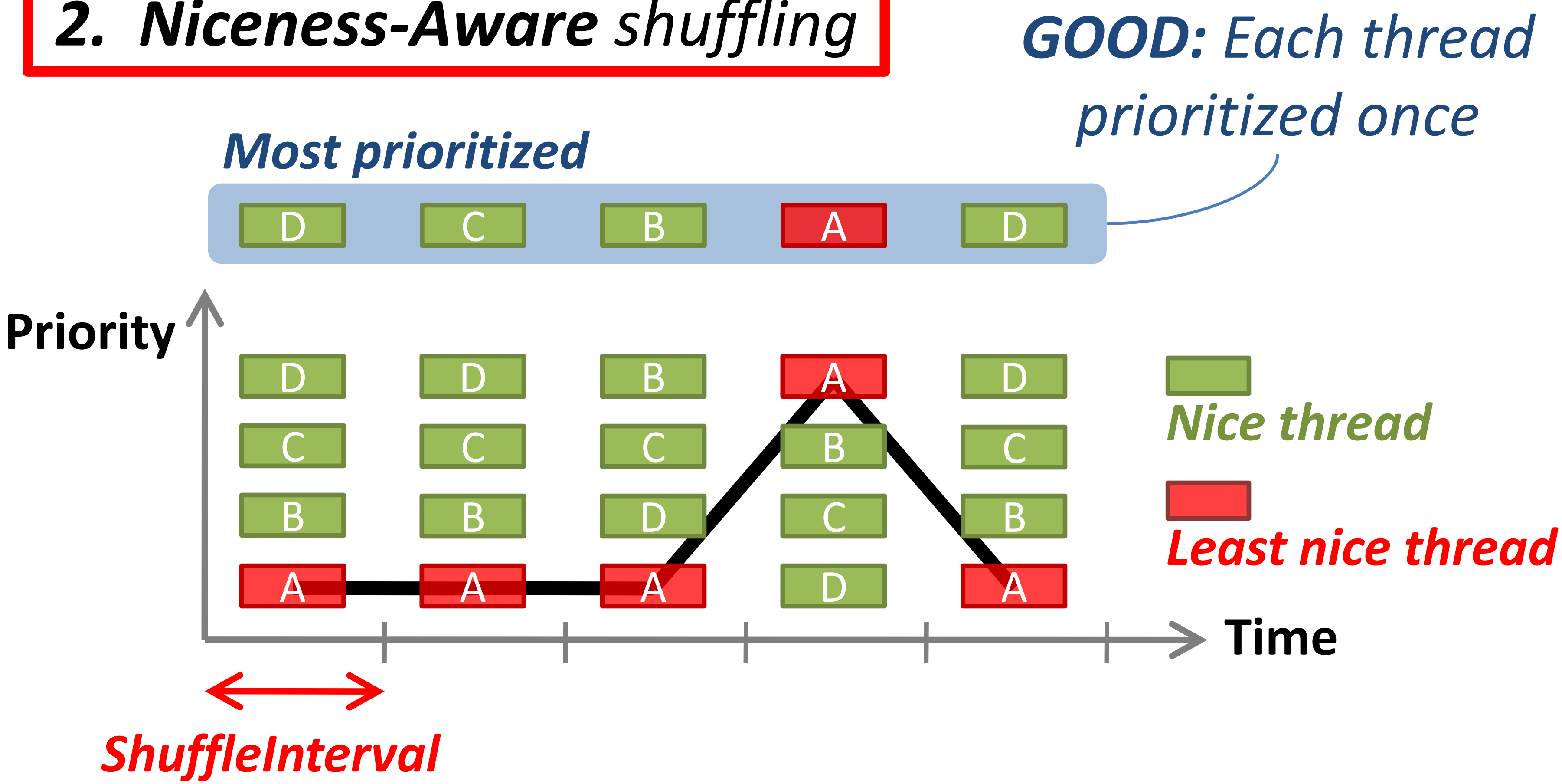
BAD: *Nice threads receive lots of interference*

Shuffling: Round-Robin vs. Niceness-Aware

1. ***Round-Robin*** *shuffling*
2. ***Niceness-Aware*** *shuffling*

Shuffling: Round-Robin vs. Niceness-Aware

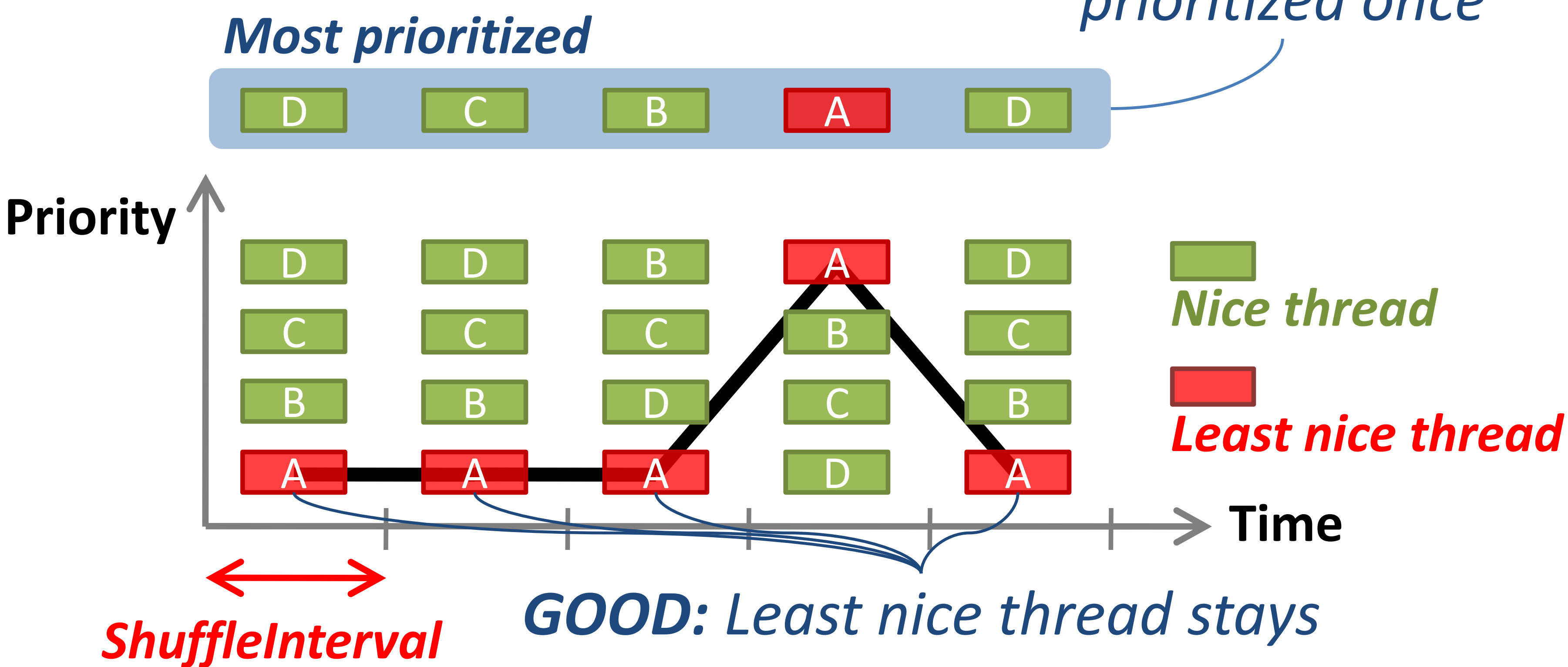
1. ***Round-Robin** shuffling*
2. ***Niceness-Aware** shuffling*



Shuffling: Round-Robin vs. Niceness-Aware

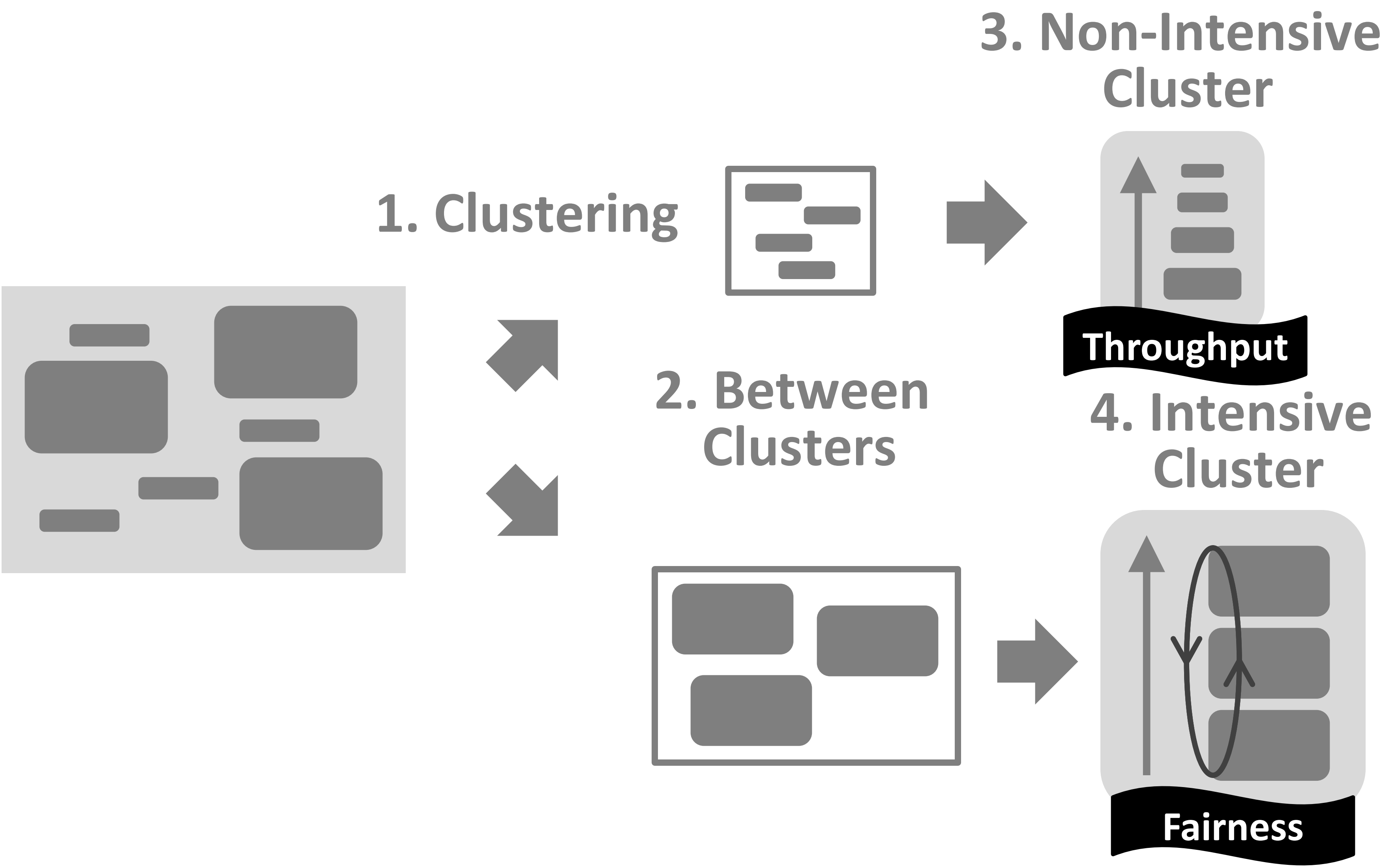
1. ***Round-Robin** shuffling*
2. ***Niceness-Aware** shuffling*

***GOOD:** Each thread prioritized once*



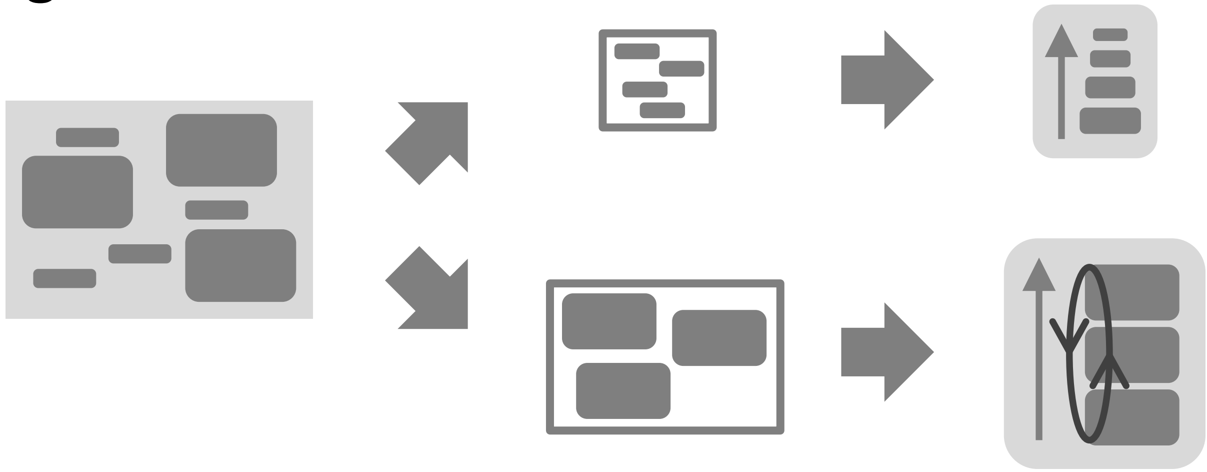
***GOOD:** Least nice thread stays mostly deprioritized*

TCM Outline


1. Clustering
2. Between Clusters
3. Non-Intensive Cluster
Throughput
4. Intensive Cluster
Fairness

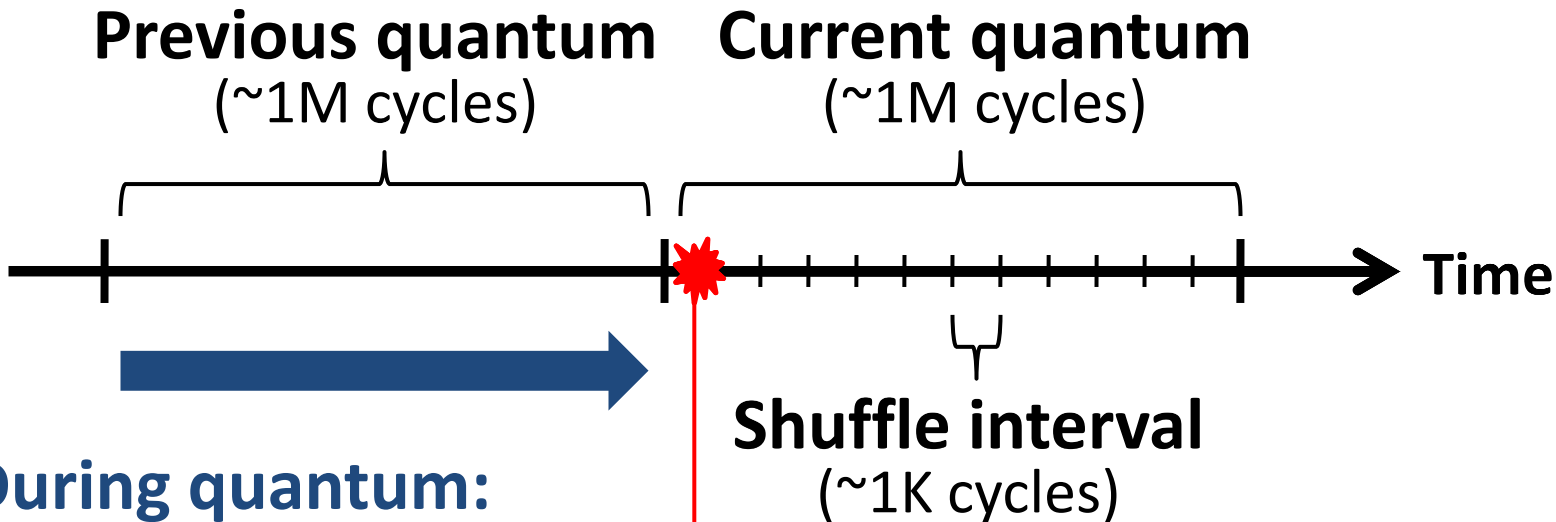
Outline

- ❑ Motivation & Insights
- ❑ Overview
- ❑ Algorithm



- ❑ **Bringing it All Together**
- ❑ Evaluation
- ❑ Conclusion

Quantum-Based Operation



During quantum:

- Monitor thread behavior
  1. Memory intensity
  2. Bank-level parallelism
  3. Row-buffer locality

Beginning of quantum:

- Perform clustering
- Compute niceness of intensive threads

TCM Scheduling Algorithm

1. ***Highest-rank***: Requests from higher ranked threads prioritized
   - **Non-Intensive** cluster **>** **Intensive** cluster
   - **Non-Intensive** cluster: lower intensity ➔ higher rank
   - **Intensive** cluster: rank shuffling

2. ***Row-hit***: Row-buffer hit requests are prioritized

3. ***Oldest***: Older requests are prioritized

Implementation Costs

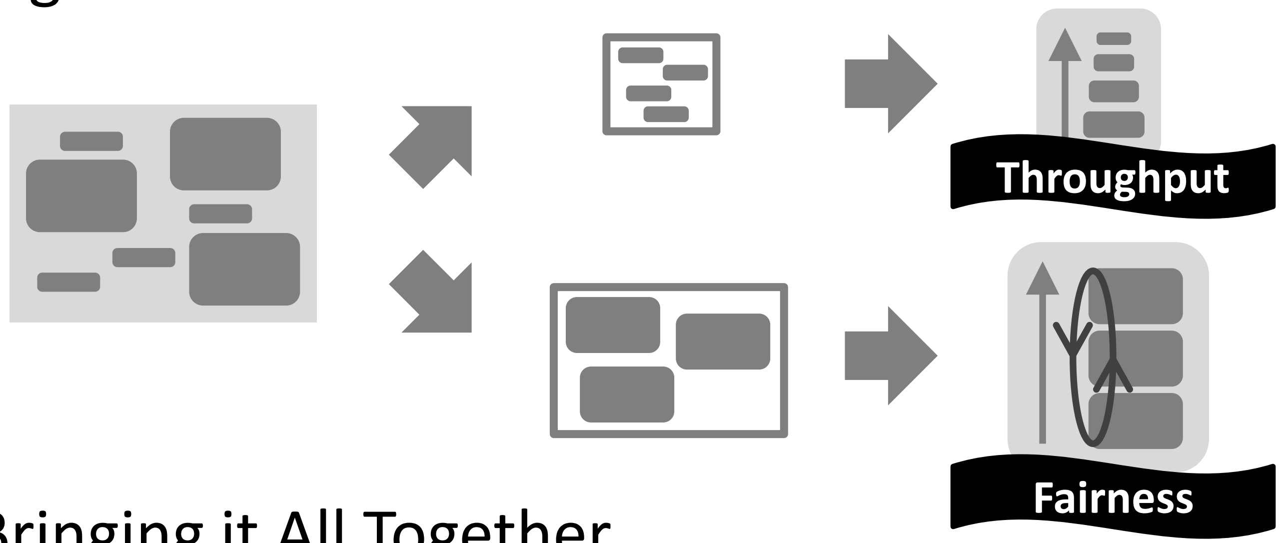
Required storage at memory controller *(24 cores)*

Thread memory behavior	Storage
MPKI	~0.2kb
Bank-level parallelism	~0.6kb
Row-buffer locality	~2.9kb
Total	**< 4kbits**

- No computation is on the critical path

Outline

- ❑ Motivation & Insights
- ❑ Overview
- ❑ Algorithm



- ❑ Bringing it All Together
- ❑ **Evaluation**
- ❑ Conclusion

Metrics & Methodology

- **Metrics**

System throughput

$$Weighted\ Speedup = \sum_i \frac{IPC_i^{shared}}{IPC_i^{alone}}$$

Unfairness

$$Maximum\ Slowdown = \max_i \frac{IPC_i^{alone}}{IPC_i^{shared}}$$

- **Methodology**
  - Core model
    - 4 GHz processor, 128-entry instruction window
    - 512 KB/core L2 cache
  - Memory model: DDR2
  - 96 multiprogrammed SPEC CPU2006 workloads

Previous Work

FRFCFS [Rixner et al., ISCA00]: Prioritizes row-buffer hits

– Thread-oblivious → Low throughput & Low fairness

STFM [Mutlu et al., MICRO07]: Equalizes thread slowdowns

– Non-intensive threads not prioritized → Low throughput

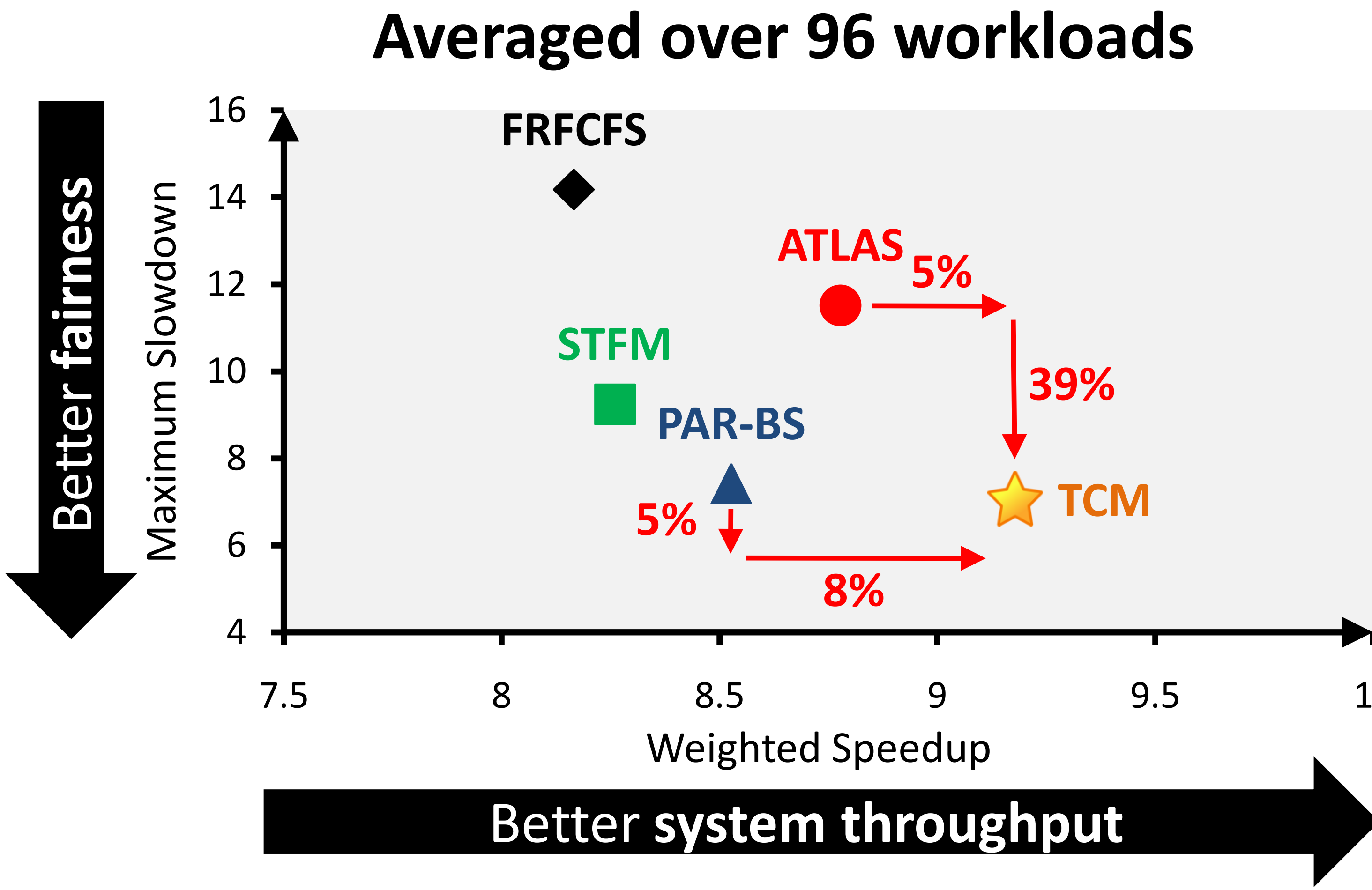
PAR-BS [Mutlu et al., ISCA08]: Prioritizes oldest batch of requests while preserving bank-level parallelism

– Non-intensive threads not always prioritized → Low throughput

ATLAS [Kim et al., HPCA10]: Prioritizes threads with less memory service

– Most intensive thread starves → Low fairness

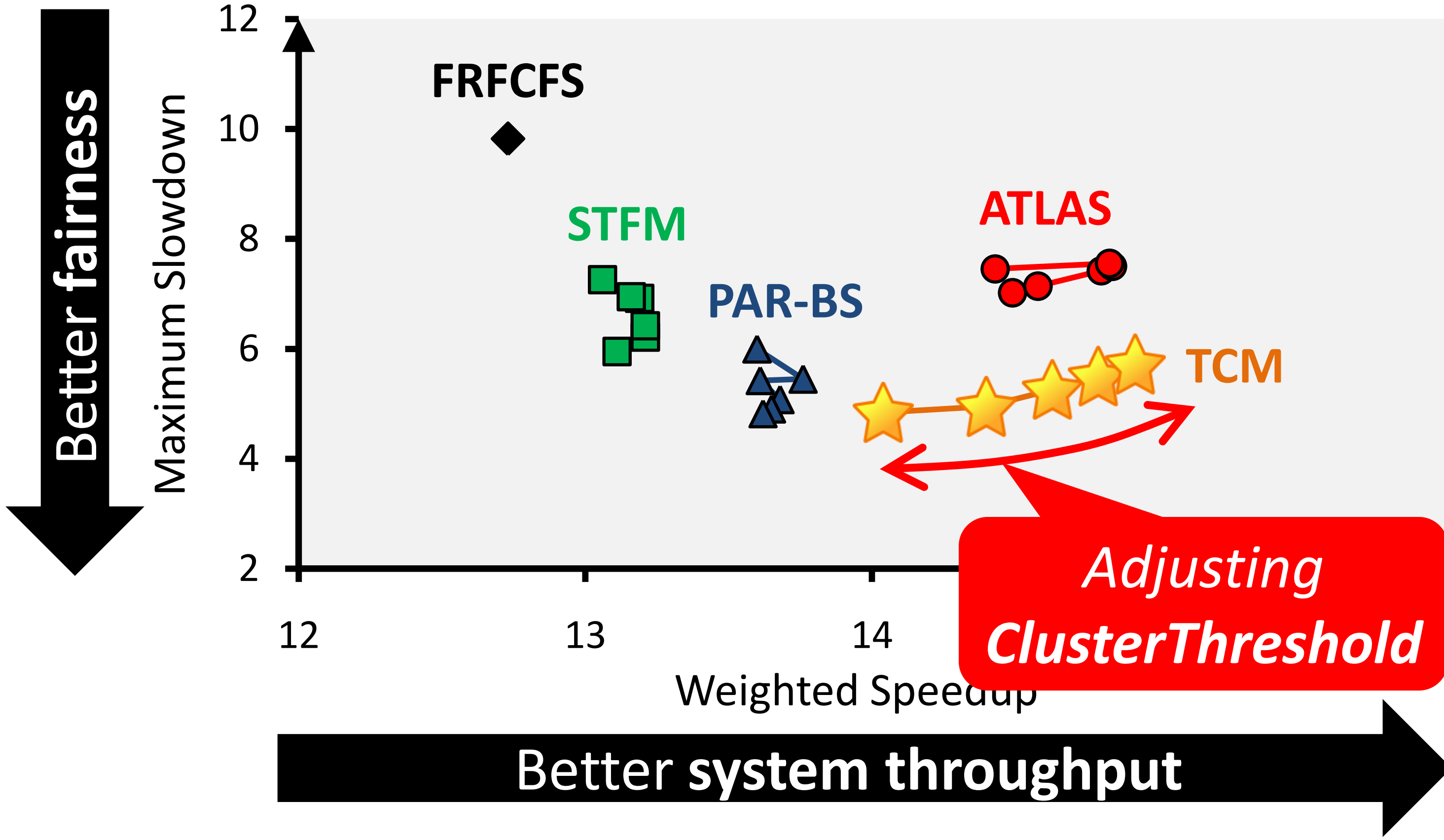
Results: Fairness vs. Throughput



TCM provides best fairness and system throughput

Results: Fairness-Throughput Tradeoff

When configuration parameter is varied…



TCM allows robust fairness-throughput tradeoff

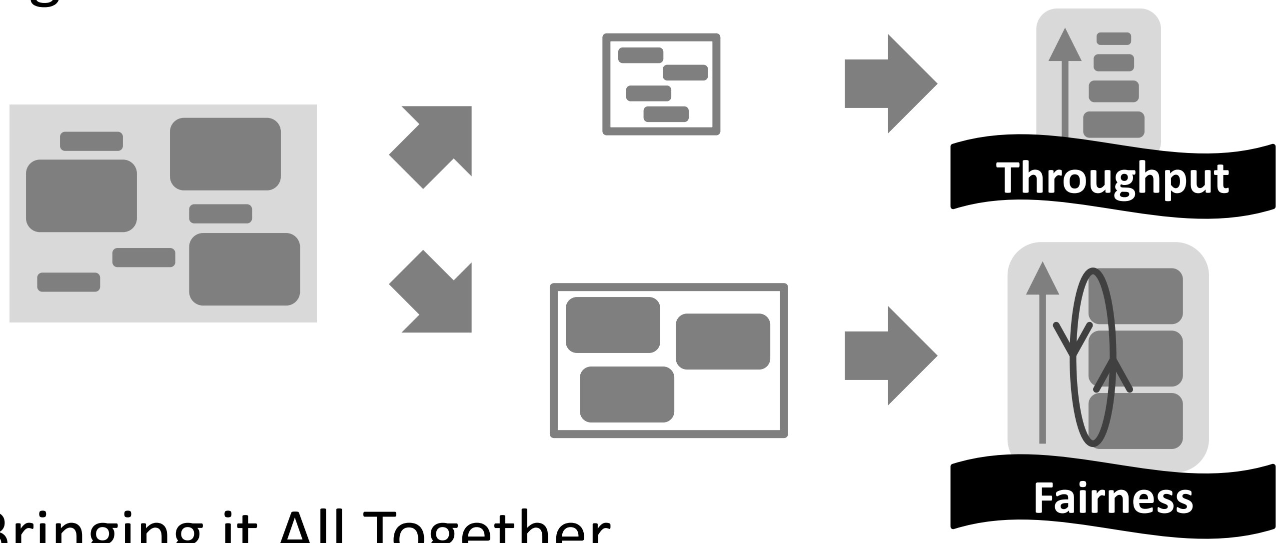
Operating System Support

- ***ClusterThreshold*** is a tunable knob
  - OS can trade off between fairness and throughput

- Enforcing thread weights
  - OS assigns weights to threads
  - TCM enforces thread weights within each cluster

Outline

- ❑ Motivation & Insights
- ❑ Overview
- ❑ Algorithm



- ❑ Bringing it All Together
- ❑ Evaluation
- ❑ **Conclusion**

Conclusion

- No previous memory scheduling algorithm provides both high ***system throughput*** and ***fairness***
  - **Problem:** They use a single policy for all threads

- TCM groups threads into two ***clusters***
  1. Prioritize ***non-intensive*** cluster → throughput
  2. Shuffle priorities in ***intensive*** cluster → fairness
  3. Shuffling should favor ***nice*** threads → fairness

- *TCM provides the best system throughput and fairness*

THANK YOU

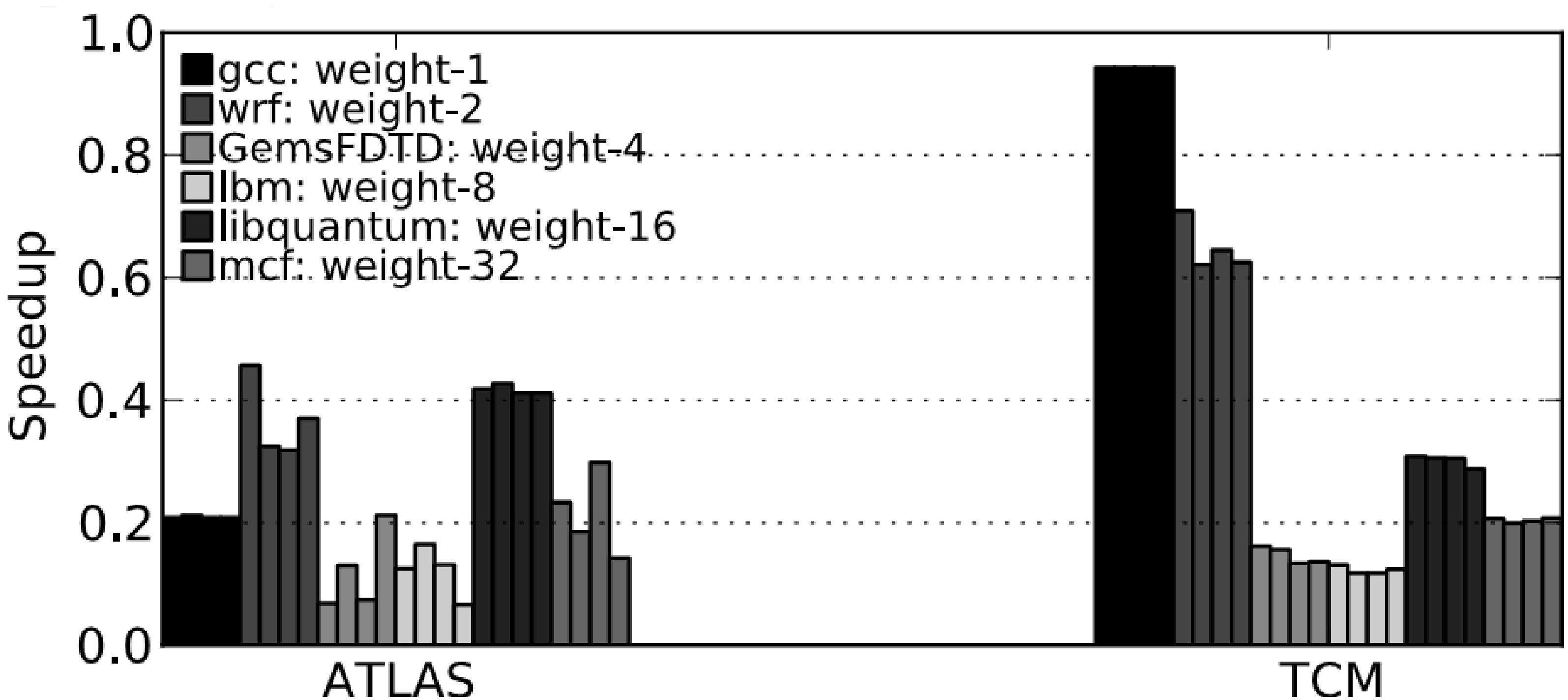
Thread Cluster Memory Scheduling: Exploiting Differences in Memory Access Behavior

Yoongu Kim
Michael Papamichael
Onur Mutlu
Mor Harchol-Balter

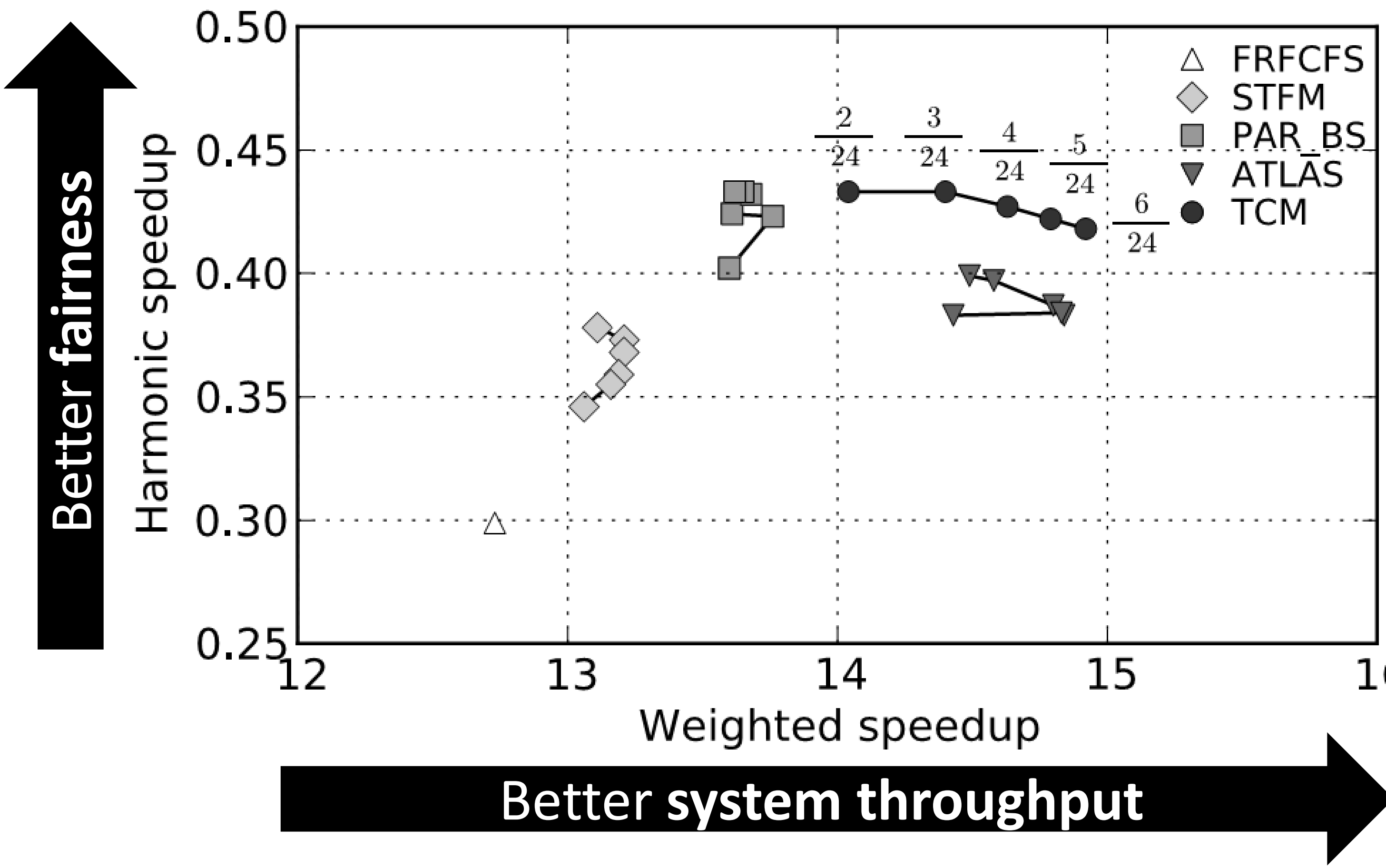
Carnegie Mellon

Thread Weight Support

- Even if heaviest weighted thread happens to be the most intensive thread…
  - Not prioritized over the least intensive thread



Harmonic Speedup



Shuffling Algorithm Comparison

- Niceness-Aware shuffling
  - Average of maximum slowdown is lower
  - Variance of maximum slowdown is lower

	Shuffling Algorithm	
	Round-Robin	**Niceness-Aware**
E(Maximum Slowdown)	5.58	4.84
VAR(Maximum Slowdown)	1.61	0.85

Sensitivity Results

	ShuffleInterval (cycles)			
	500	600	700	800
System Throughput	14.2	14.3	14.2	14.7
Maximum Slowdown	6.0	5.4	5.9	5.5

	Number of Cores				
	4	8	16	24	32
System Throughput *(compared to ATLAS)*	0%	3%	2%	1%	1%
Maximum Slowdown *(compared to ATLAS)*	-4%	-30%	-29%	-30%	-41%