Fairness via Source Throttling:

A configurable and high-performance fairness substrate for multi-core memory systems

Eiman Ebrahimi*

Chang Joo Lee*

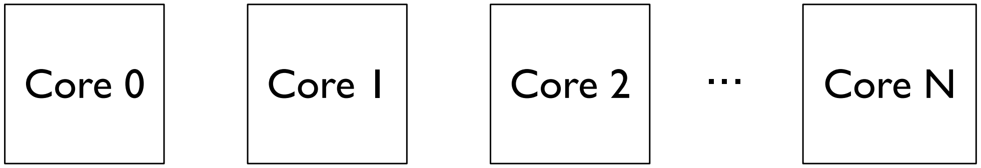
Onur Mutlu‡

Yale N. Patt*

* HPS Research Group
The University of Texas at Austin

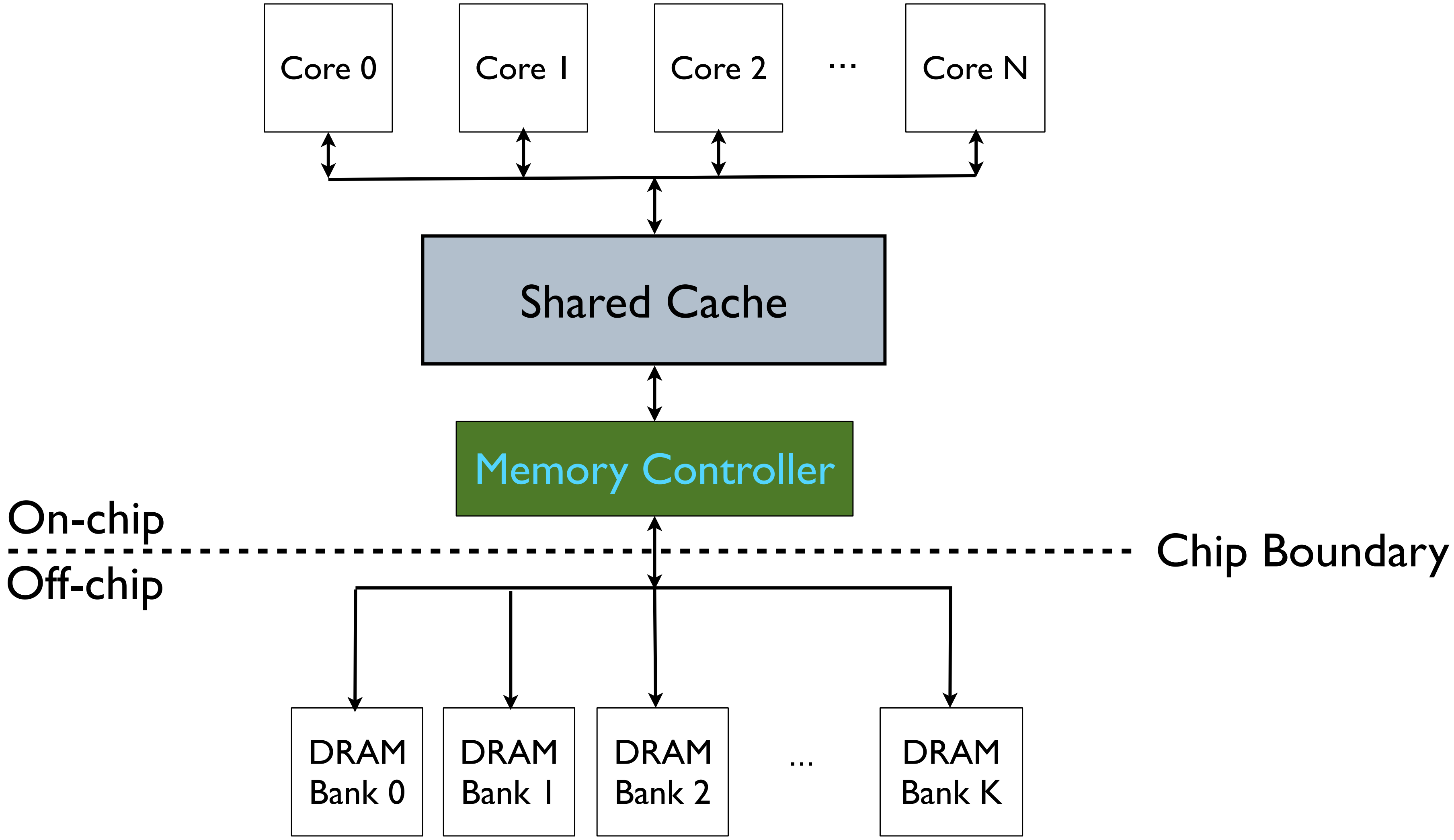
‡ Computer Architecture Laboratory
Carnegie Mellon University

Background and Problem

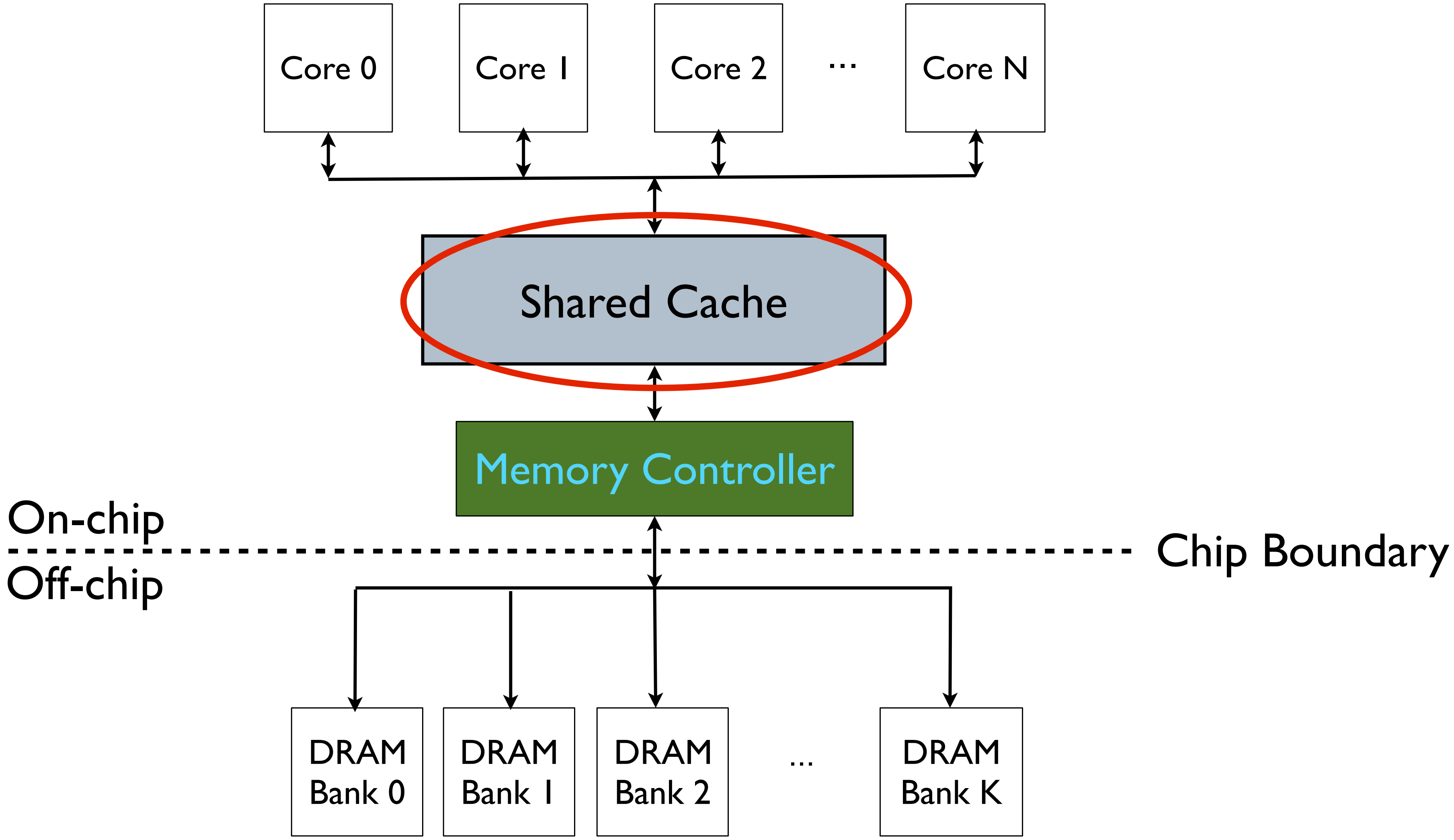


Core 0 Core 1 Core 2 … Core N

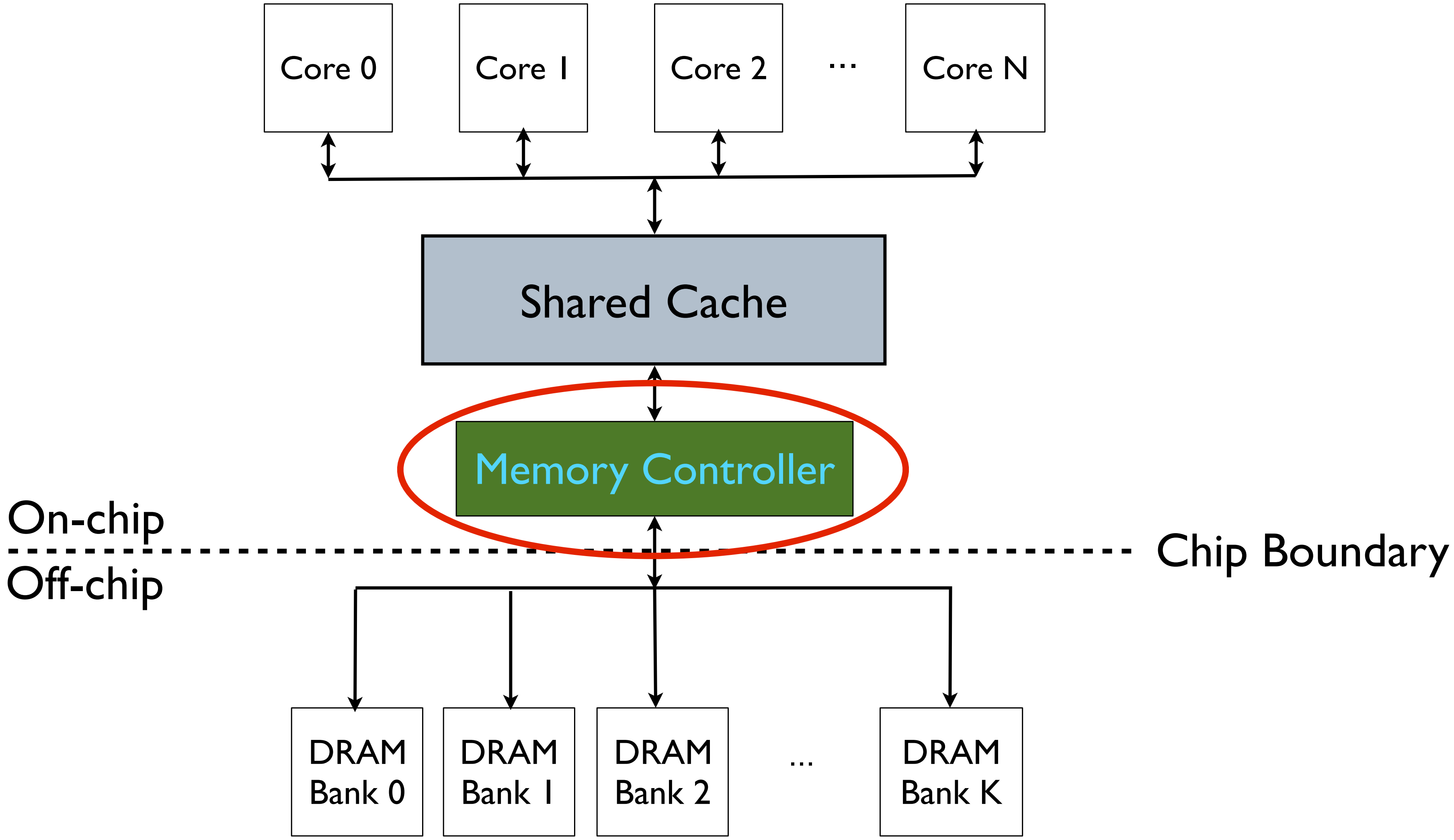
Background and Problem



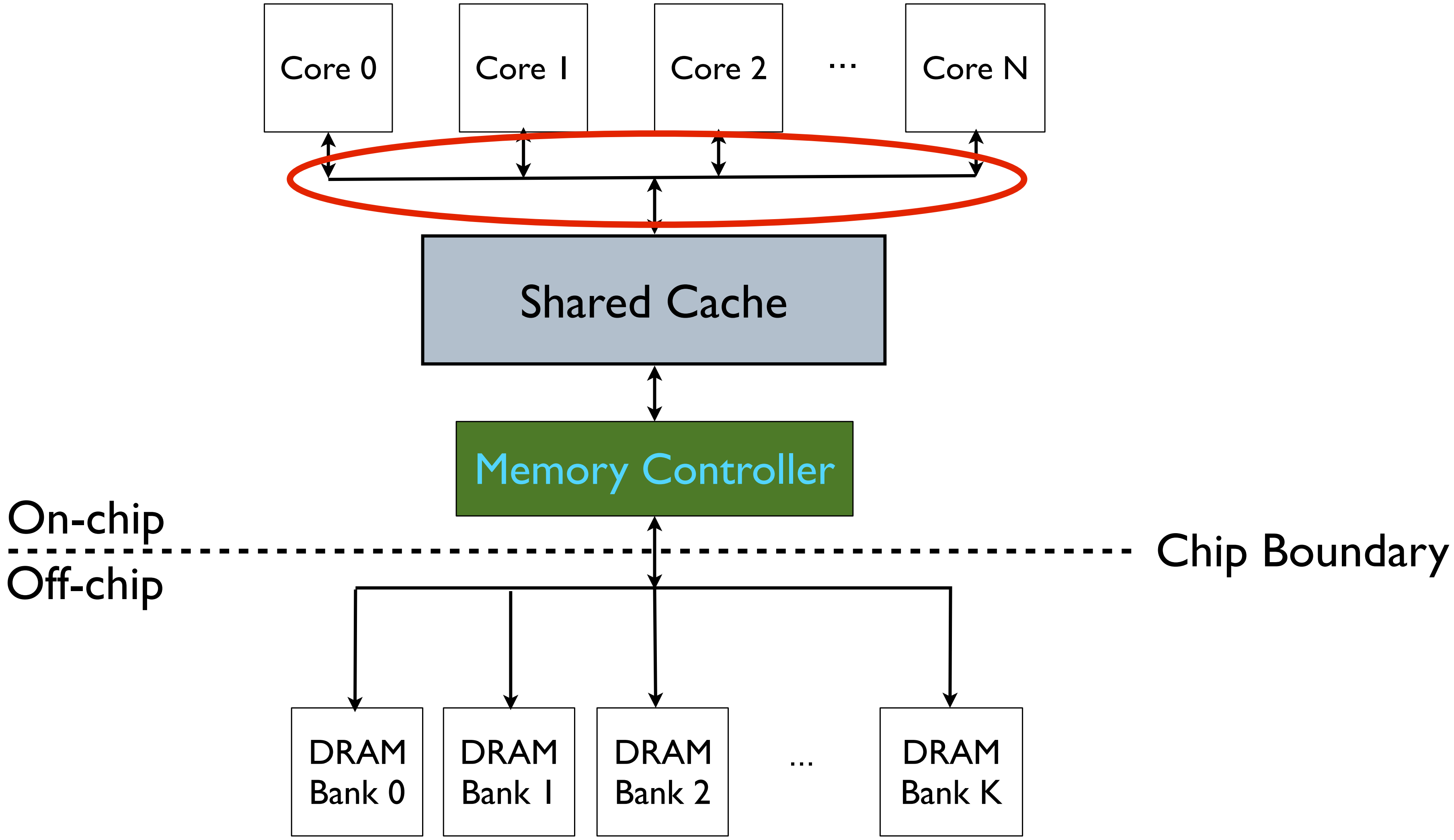
Background and Problem



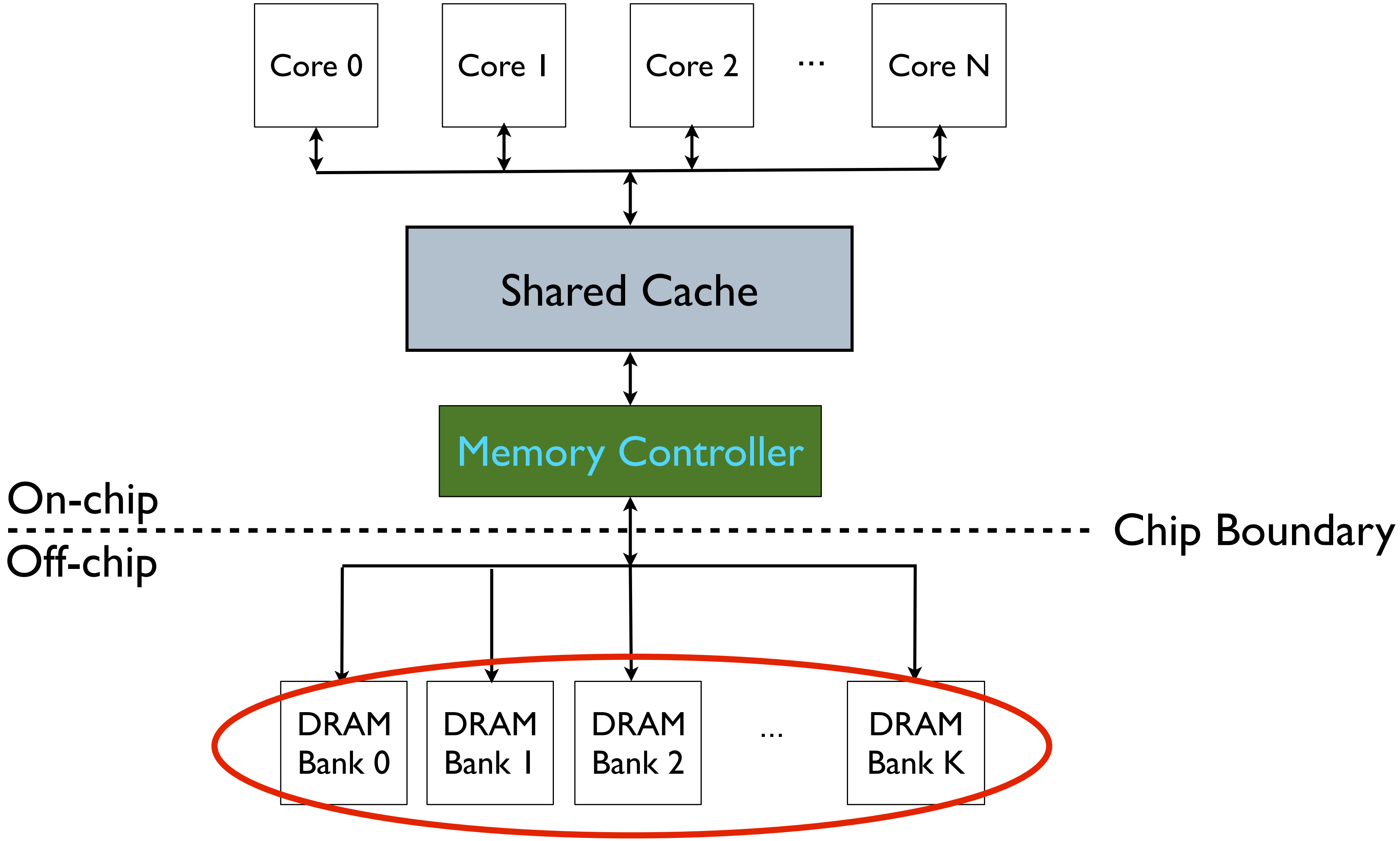
Background and Problem



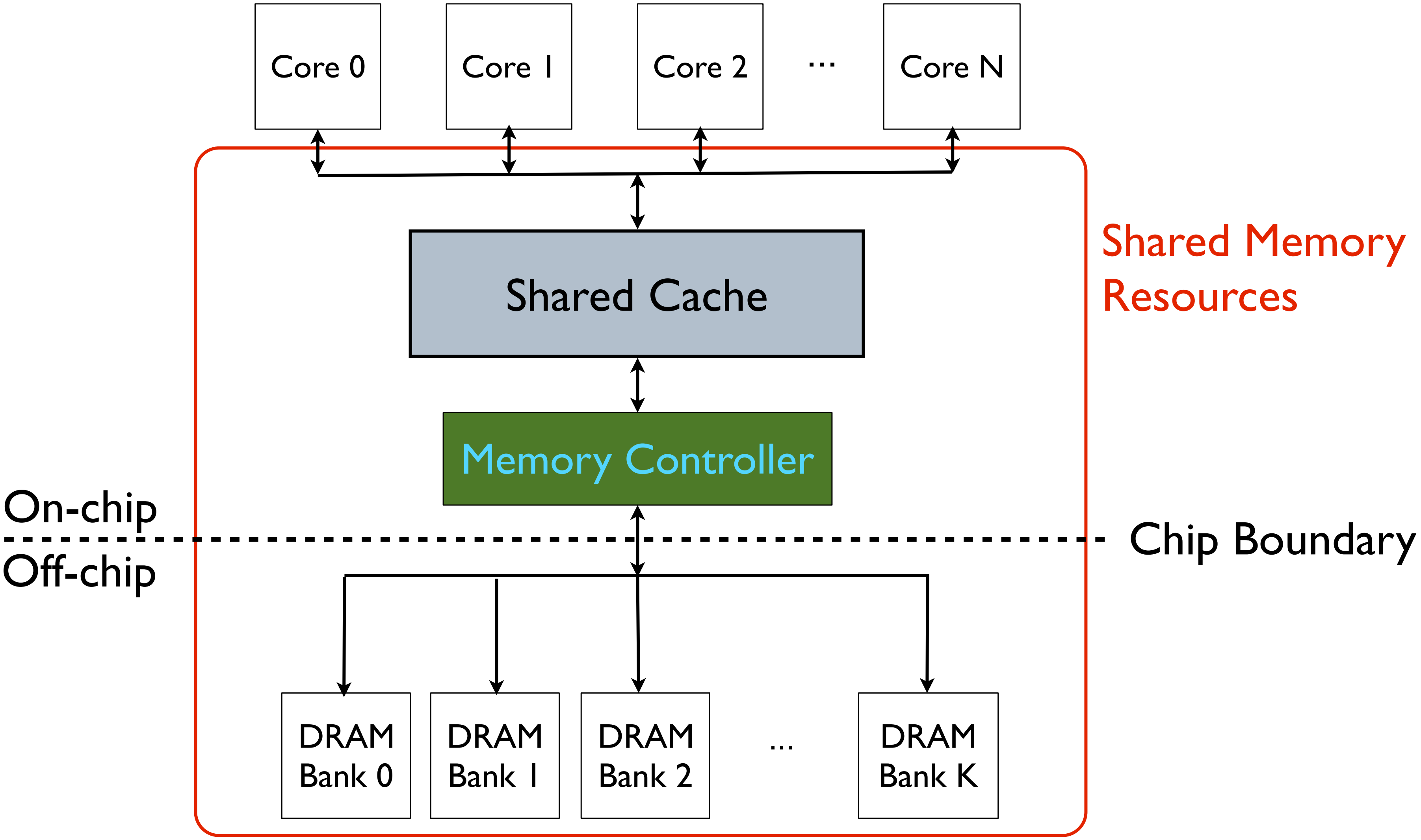
Background and Problem



Background and Problem



Background and Problem



Background and Problem

- Applications slow down due to interference from memory requests of other applications

Background and Problem

- Applications slow down due to interference from memory requests of other applications
- A memory system is **fair** if slowdowns of same-priority applications are equal (MICRO '06, MICRO '07, ISCA '08)

Background and Problem

- Applications slow down due to interference from memory requests of other applications
- A memory system is **fair** if slowdowns of same-priority applications are equal (MICRO '06, MICRO '07, ISCA '08)
- Slowdown of application i = $\dfrac{T_i^{Shared}}{T_i^{Alone}}$

Background and Problem

- Applications slow down due to interference from memory requests of other applications
- A memory system is **fair** if slowdowns of same-priority applications are equal (MICRO '06, MICRO '07, ISCA '08)
- Slowdown of application i = $\dfrac{T_i^{\text{Shared}}}{T_i^{\text{Alone}}}$
- Unfairness = $\dfrac{\text{Max}\{\text{Slowdown } i\} \text{ over all applications } i}{\text{Min}\{\text{Slowdown } i\} \text{ over all applications } i}$

  (MICRO '07)

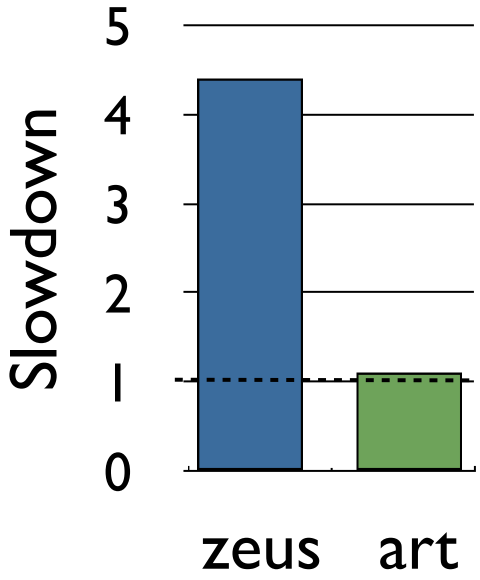
Background and Problem

Background and Problem

- Magnitude of each application's slowdown depends on concurrently running applications' memory behavior

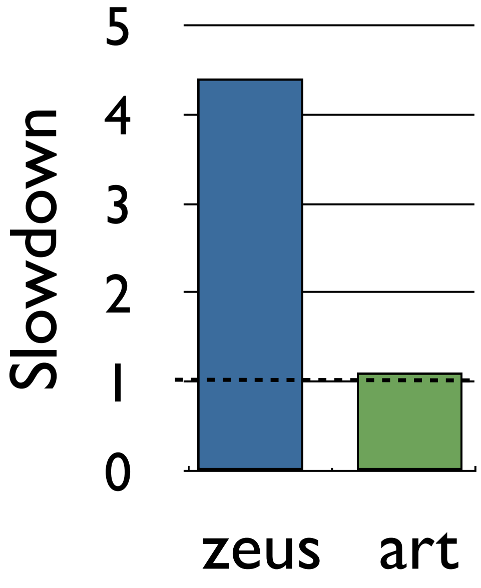
Background and Problem

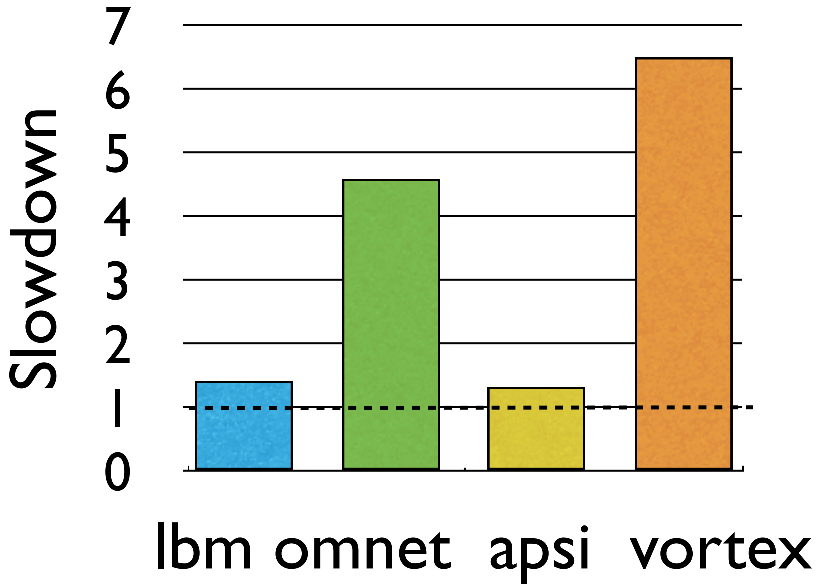
- Magnitude of each application’s slowdown depends on concurrently running applications’ memory behavior



Background and Problem

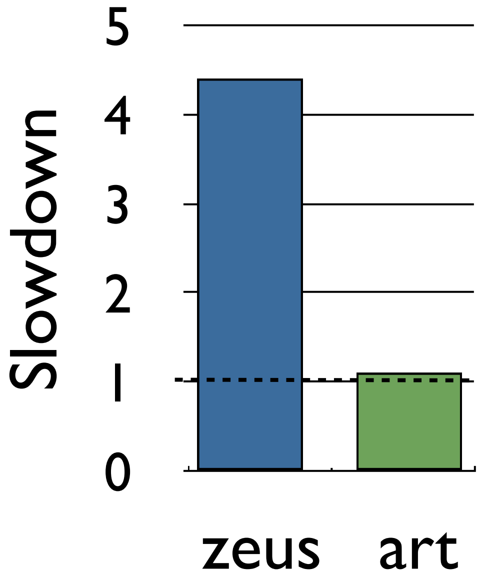
- Magnitude of each application’s slowdown depends on concurrently running applications’ memory behavior

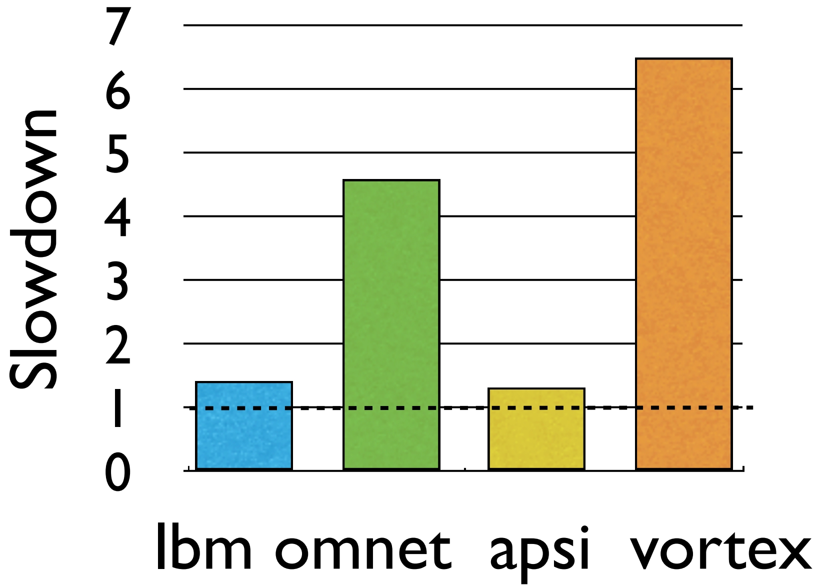




Background and Problem

- Magnitude of each application's slowdown depends on concurrently running applications' memory behavior

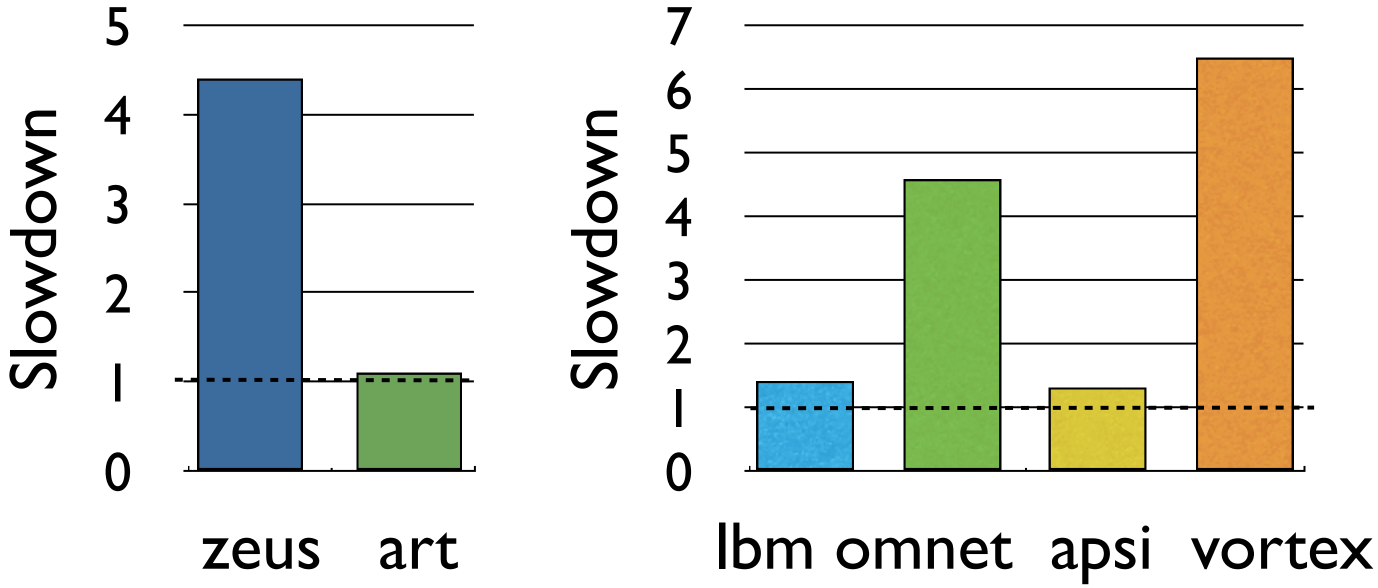




- Large disparities in slowdowns are unacceptable

Background and Problem

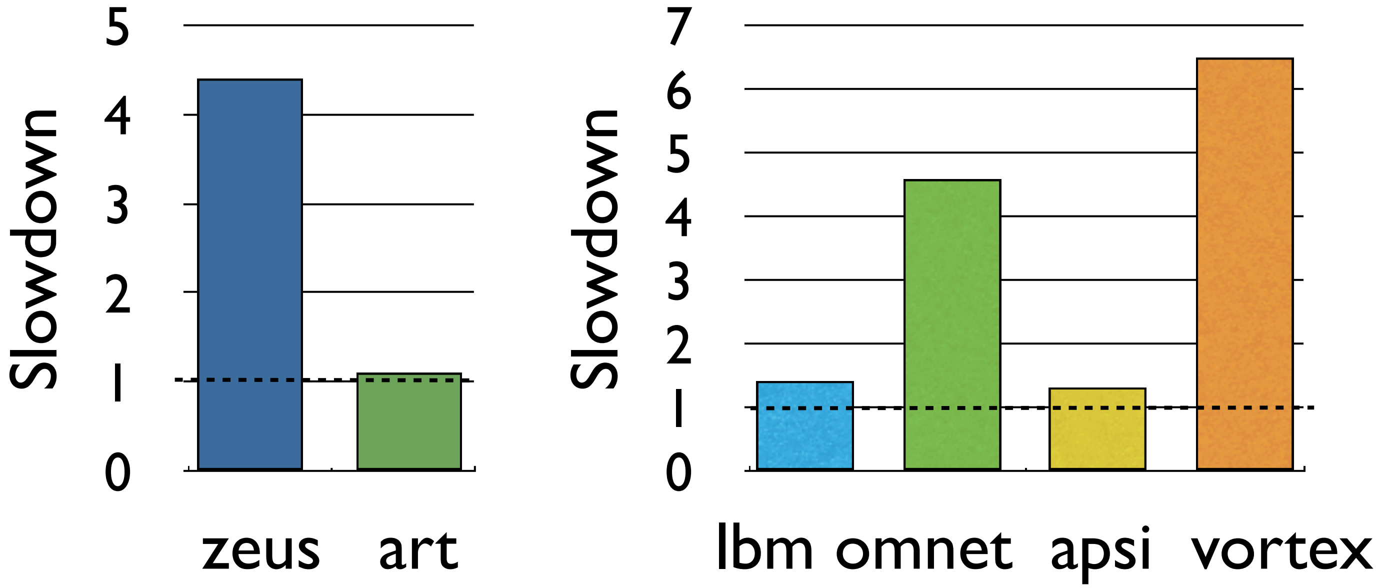
- Magnitude of each application's slowdown depends on concurrently running applications' memory behavior



- Large disparities in slowdowns are unacceptable
  - Low system performance

Background and Problem

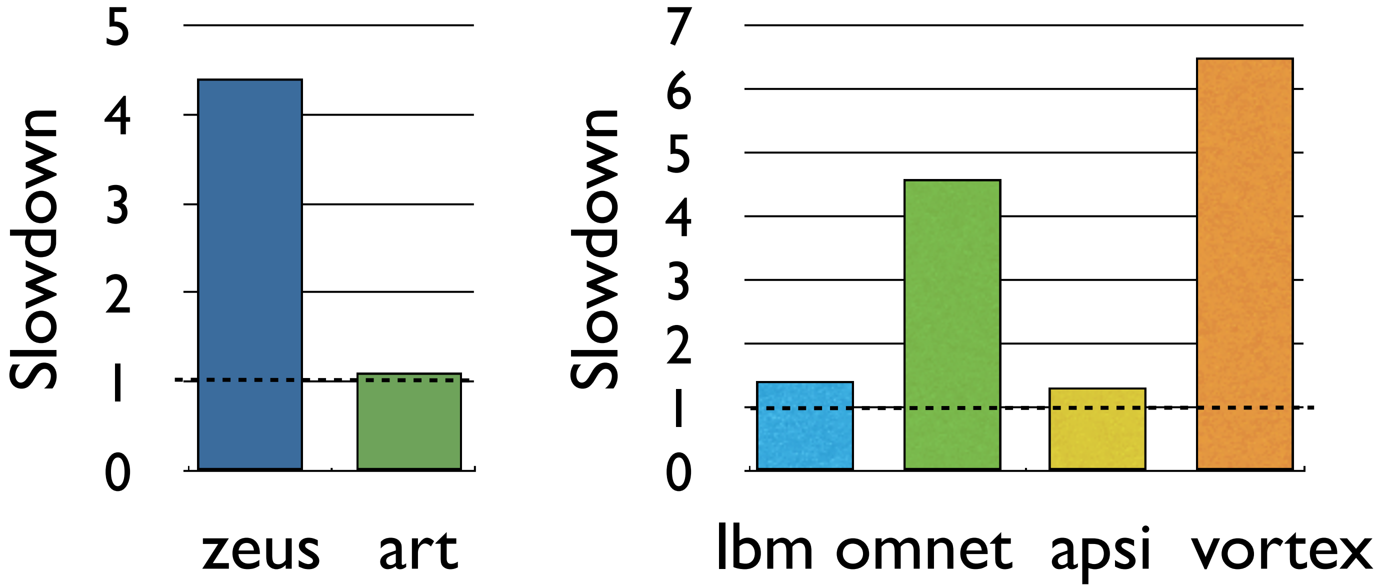
- Magnitude of each application’s slowdown depends on concurrently running applications’ memory behavior



- Large disparities in slowdowns are unacceptable
  - Low system performance
  - Vulnerability to denial of service attacks

Background and Problem

- Magnitude of each application's slowdown depends on concurrently running applications' memory behavior



- Large disparities in slowdowns are unacceptable
  - Low system performance
  - Vulnerability to denial of service attacks
  - Difficult for system software to enforce priorities

Outline

- Background and Problem
- Motivation for Source Throttling
- Fairness via Source Throttling (FST)
- Evaluation
- Conclusion

Prior Approaches

Prior Approaches

- Primarily manage inter-application interference in only one particular resource
  - Shared Cache, Memory Controller, Interconnect, etc.

Prior Approaches

- Primarily manage inter-application interference in only one particular resource
  - Shared Cache, Memory Controller, Interconnect, etc.
  - Combining techniques for the different resources can result in negative interaction

Prior Approaches

- Primarily manage inter-application interference in only one particular resource
  - Shared Cache, Memory Controller, Interconnect, etc.
  - Combining techniques for the different resources can result in negative interaction
- Approaches that coordinate interaction among techniques for different resources require complex implementations

Prior Approaches

- Primarily manage inter-application interference in only one particular resource
  - Shared Cache, Memory Controller, Interconnect, etc.
  - Combining techniques for the different resources can result in negative interaction
- Approaches that coordinate interaction among techniques for different resources require complex implementations

Our Goal: Enable fair sharing of the entire memory system by dynamically detecting and controlling interference in a coordinated manner

Our Approach

Our Approach

- Manage inter-application interference at the cores, not at the shared resources

Our Approach

- Manage inter-application interference at the cores, not at the shared resources
- Dynamically estimate unfairness in the memory system

Our Approach

- Manage inter-application interference at the cores, not at the shared resources
- Dynamically estimate unfairness in the memory system
- If unfairness > system-software-specified target then
  throttle down core causing unfairness &
  throttle up core that was unfairly treated

Unmanaged Interference

A:

B:

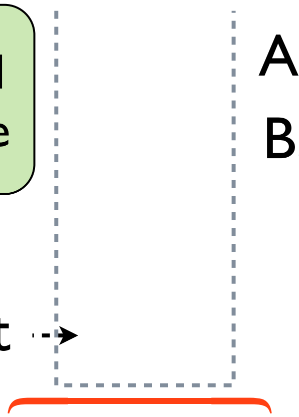
Fair Source Throttling

A:

B:

queue of requests to shared resources

Unmanaged Interference



Oldest

A:

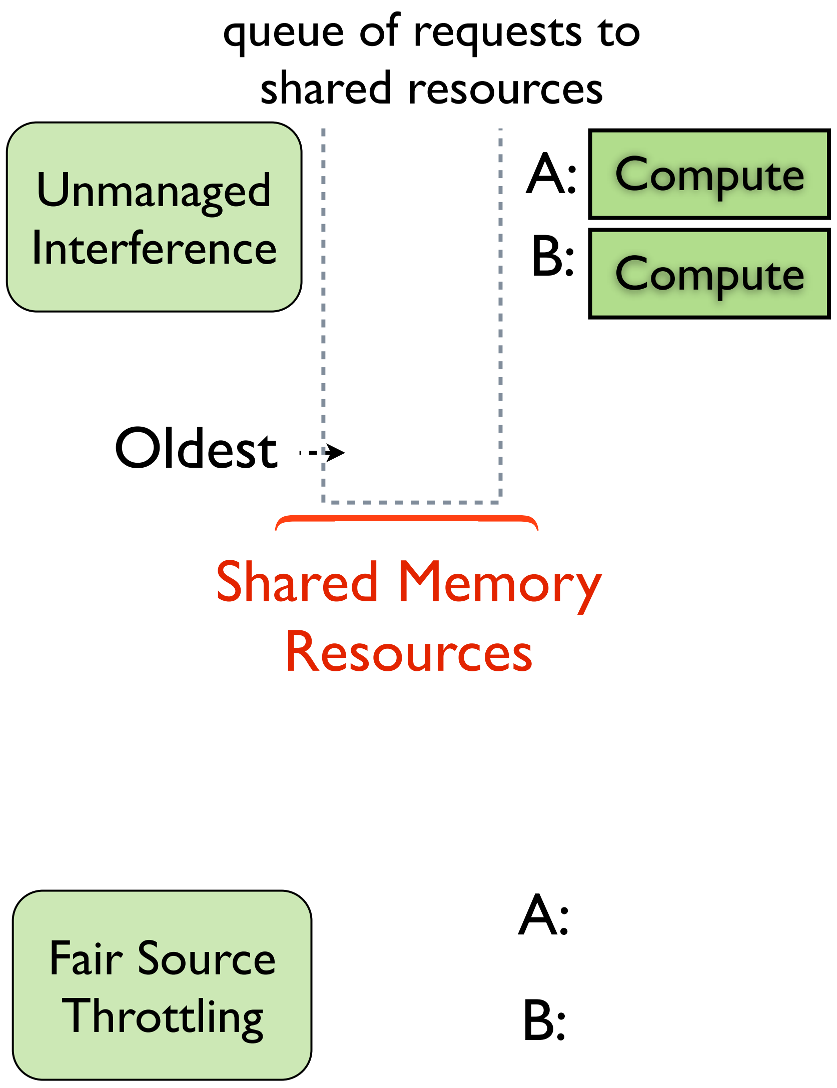
B:

Shared Memory Resources

Fair Source Throttling

A:

B:



queue of requests to shared resources

Unmanaged Interference

A: Compute

B: Compute

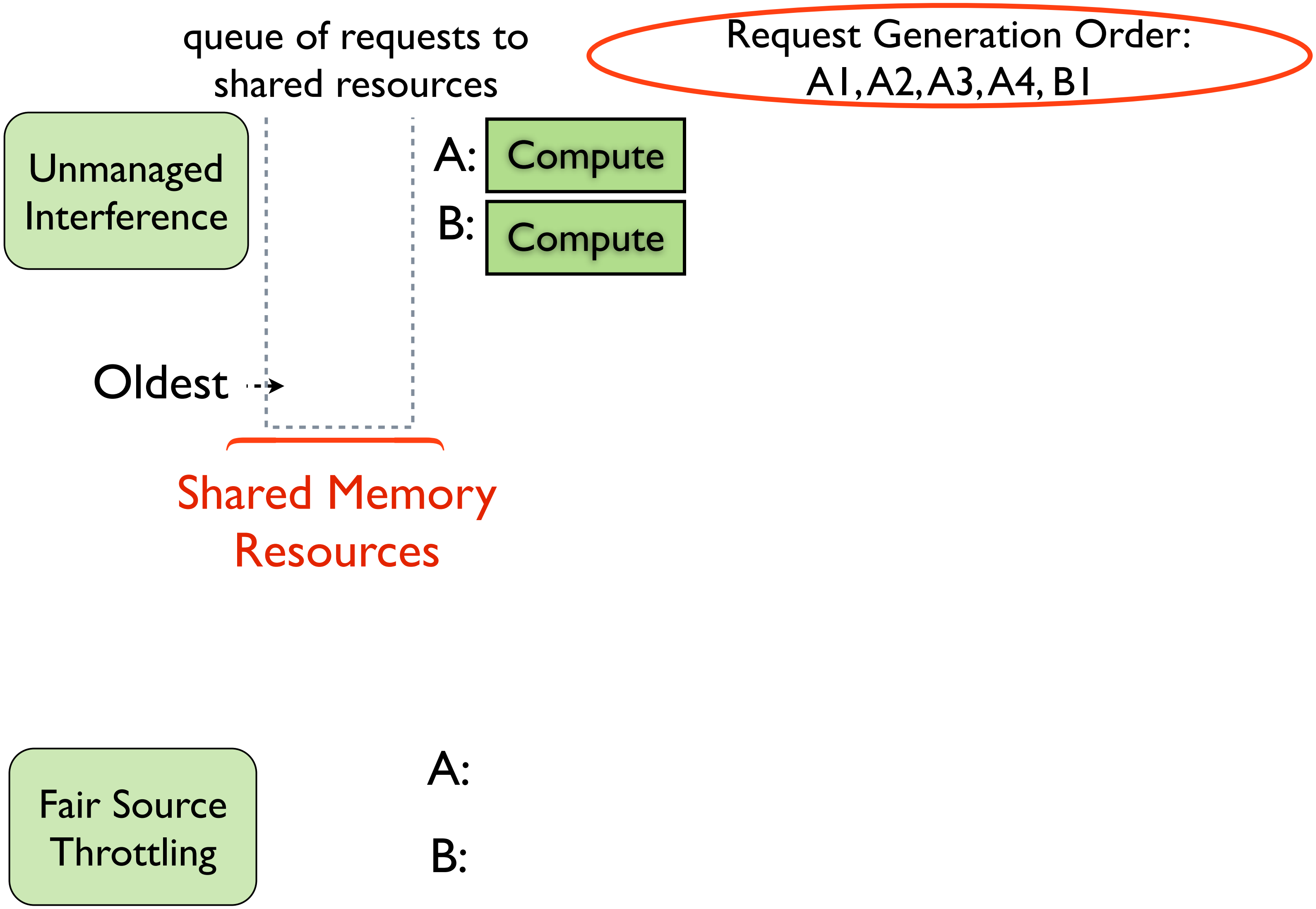
Oldest

Shared Memory Resources

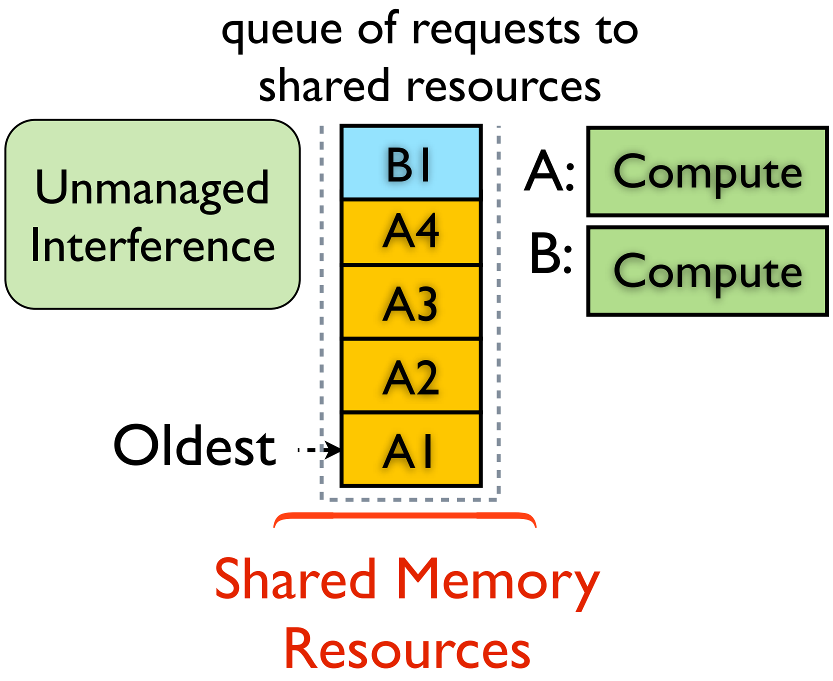
Fair Source Throttling

A:

B:


queue of requests to shared resources
Request Generation Order: A1,A2,A3,A4, B1
Unmanaged Interference
A:
Compute
B:
Compute
Oldest
Shared Memory Resources
Fair Source Throttling
A:
B:

Request Generation Order:
A1, A2, A3, A4, B1



queue of requests to shared resources

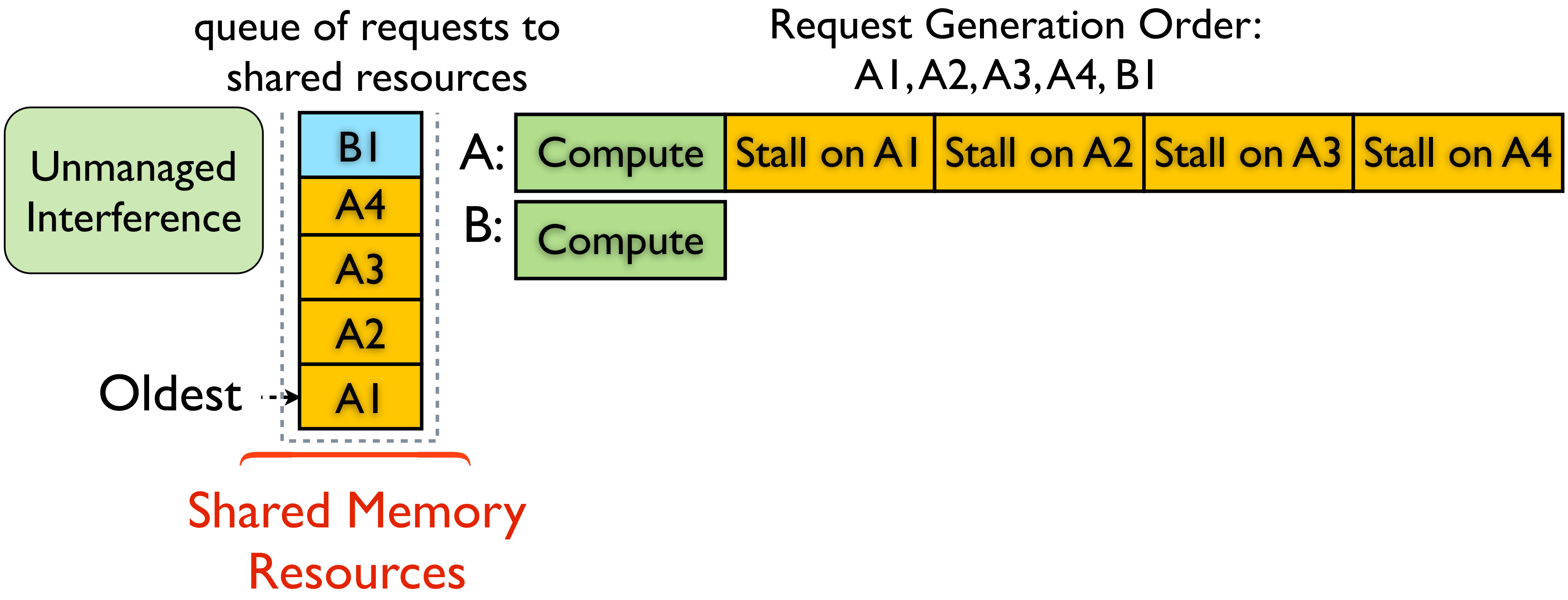
Unmanaged Interference

Shared Memory Resources

Fair Source Throttling

A:

B:



queue of requests to shared resources

Request Generation Order: A1, A2, A3, A4, B1

Unmanaged Interference

B1

A4

A3

A2

Oldest A1

A: Compute | Stall on A1 | Stall on A2 | Stall on A3 | Stall on A4

B: Compute

Shared Memory Resources

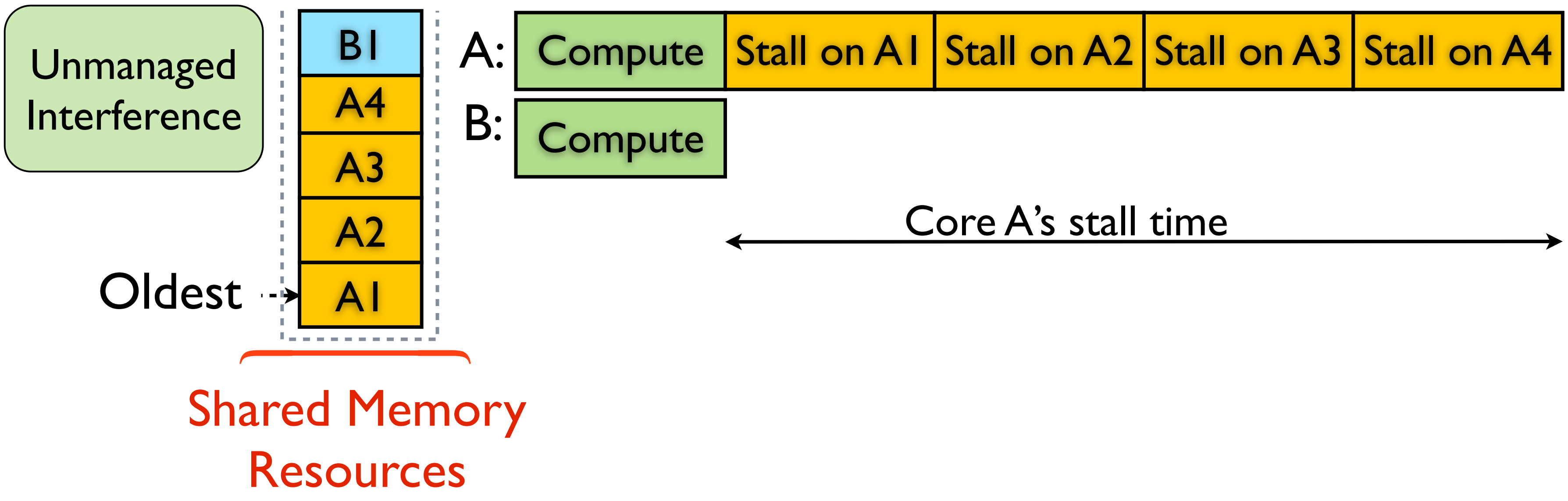
Fair Source Throttling

A:

B:

queue of requests to shared resources

Request Generation Order: A1, A2, A3, A4, B1



Unmanaged Interference

B1
A4
A3
A2
A1

Oldest

A: Compute | Stall on A1 | Stall on A2 | Stall on A3 | Stall on A4

B: Compute

Core A's stall time

Shared Memory Resources

Fair Source Throttling

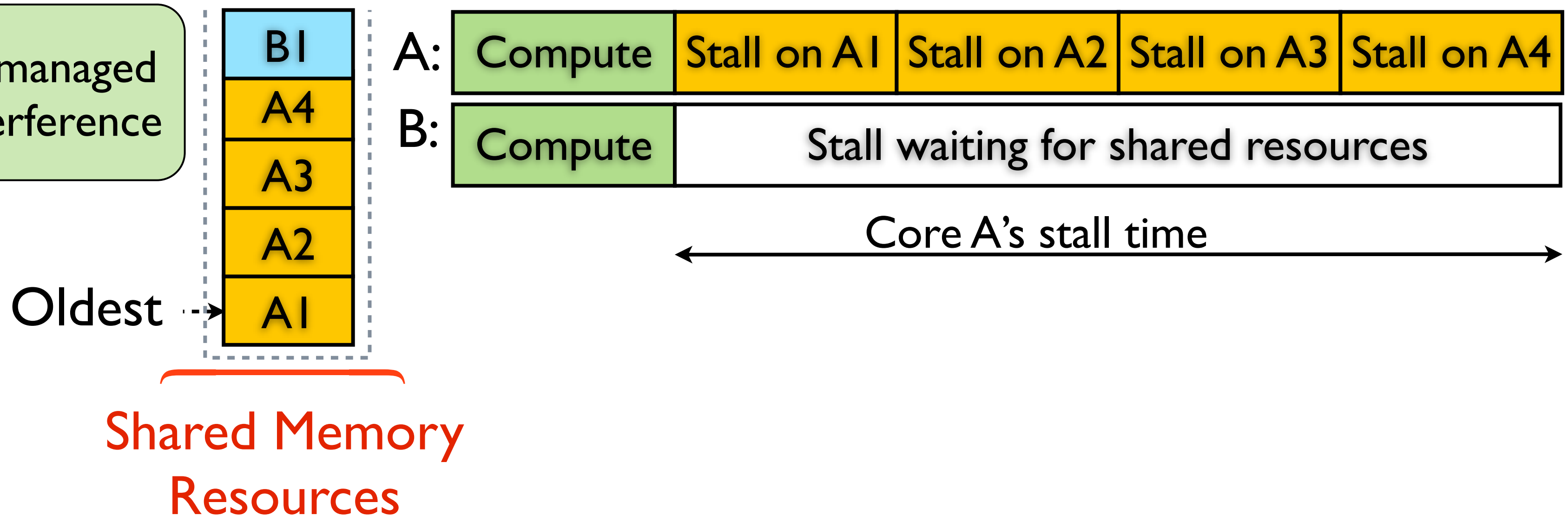
A:

B:

Unmanaged Interference

queue of requests to shared resources

Request Generation Order: A1, A2, A3, A4, B1



B1
A4
A3
A2
A1

Oldest

Shared Memory Resources

A: Compute | Stall on A1 | Stall on A2 | Stall on A3 | Stall on A4

B: Compute | Stall waiting for shared resources

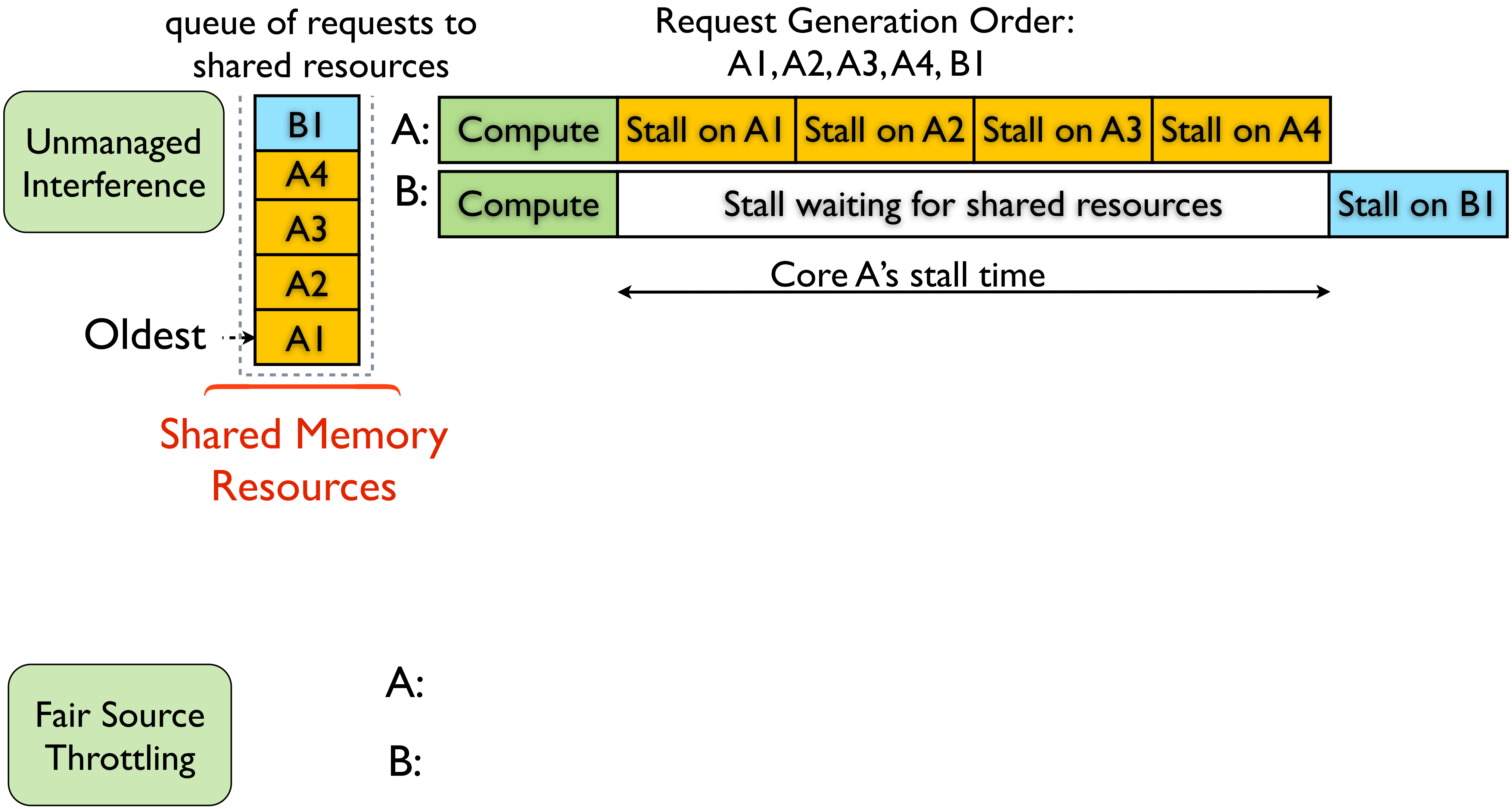
Core A's stall time

Fair Source Throttling

A:

B:

Request Generation Order:
A1, A2, A3, A4, B1



queue of requests to shared resources

Unmanaged Interference

B1

A4

A3

A2

A1

Oldest

Shared Memory Resources

A: Compute | Stall on A1 | Stall on A2 | Stall on A3 | Stall on A4

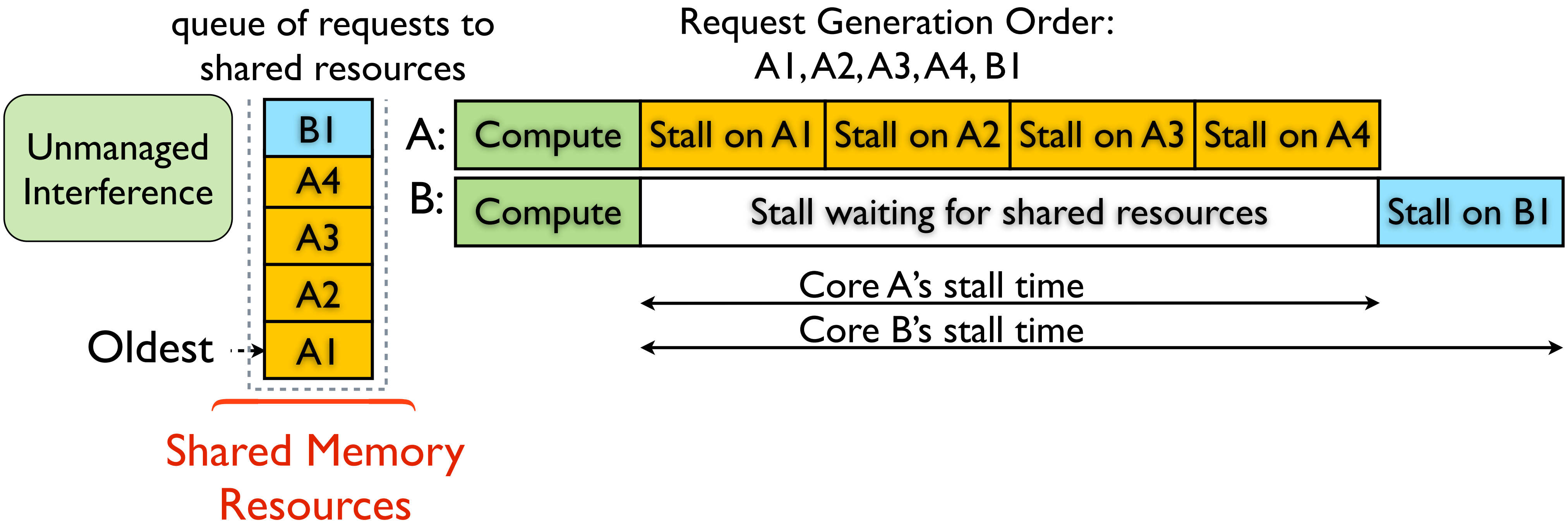
B: Compute | Stall waiting for shared resources | Stall on B1

Core A's stall time

Fair Source Throttling

A:

B:



Unmanaged Interference

queue of requests to shared resources

B1
A4
A3
A2
A1

Oldest

Shared Memory Resources

Request Generation Order: A1, A2, A3, A4, B1

A: Compute | Stall on A1 | Stall on A2 | Stall on A3 | Stall on A4

B: Compute | Stall waiting for shared resources | Stall on B1

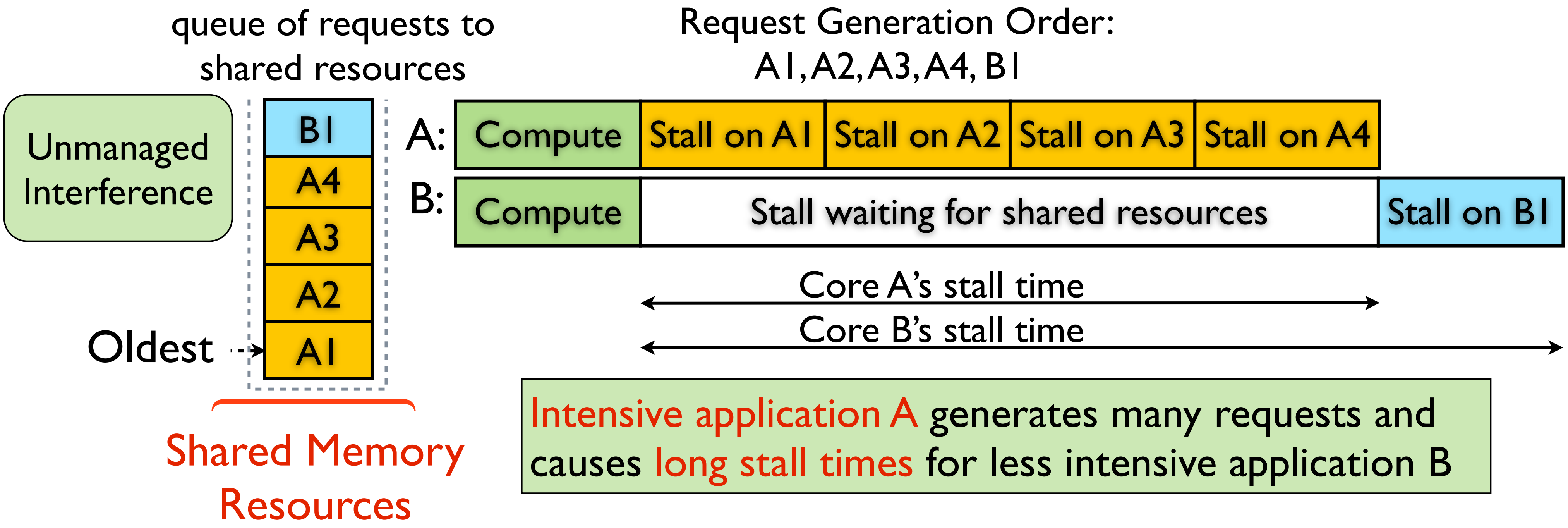
Core A's stall time

Core B's stall time

Fair Source Throttling

A:

B:



Intensive application A generates many requests and causes long stall times for less intensive application B

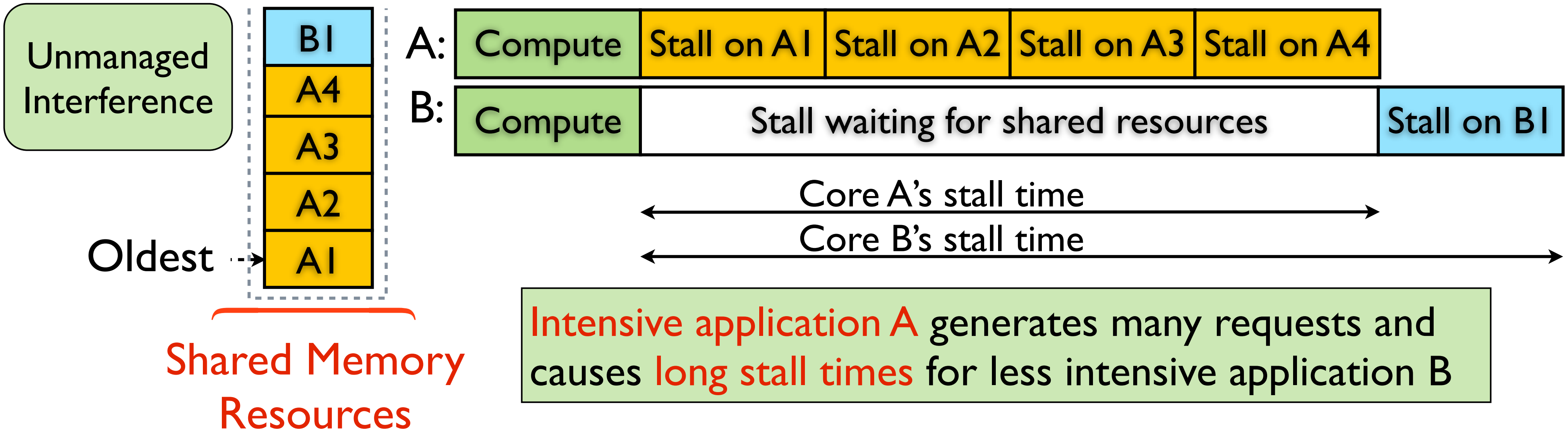
Fair Source Throttling

A:

B:

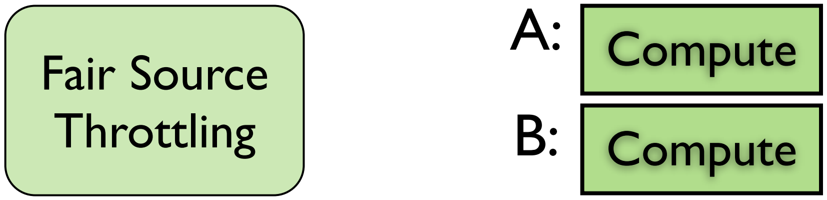
queue of requests to shared resources

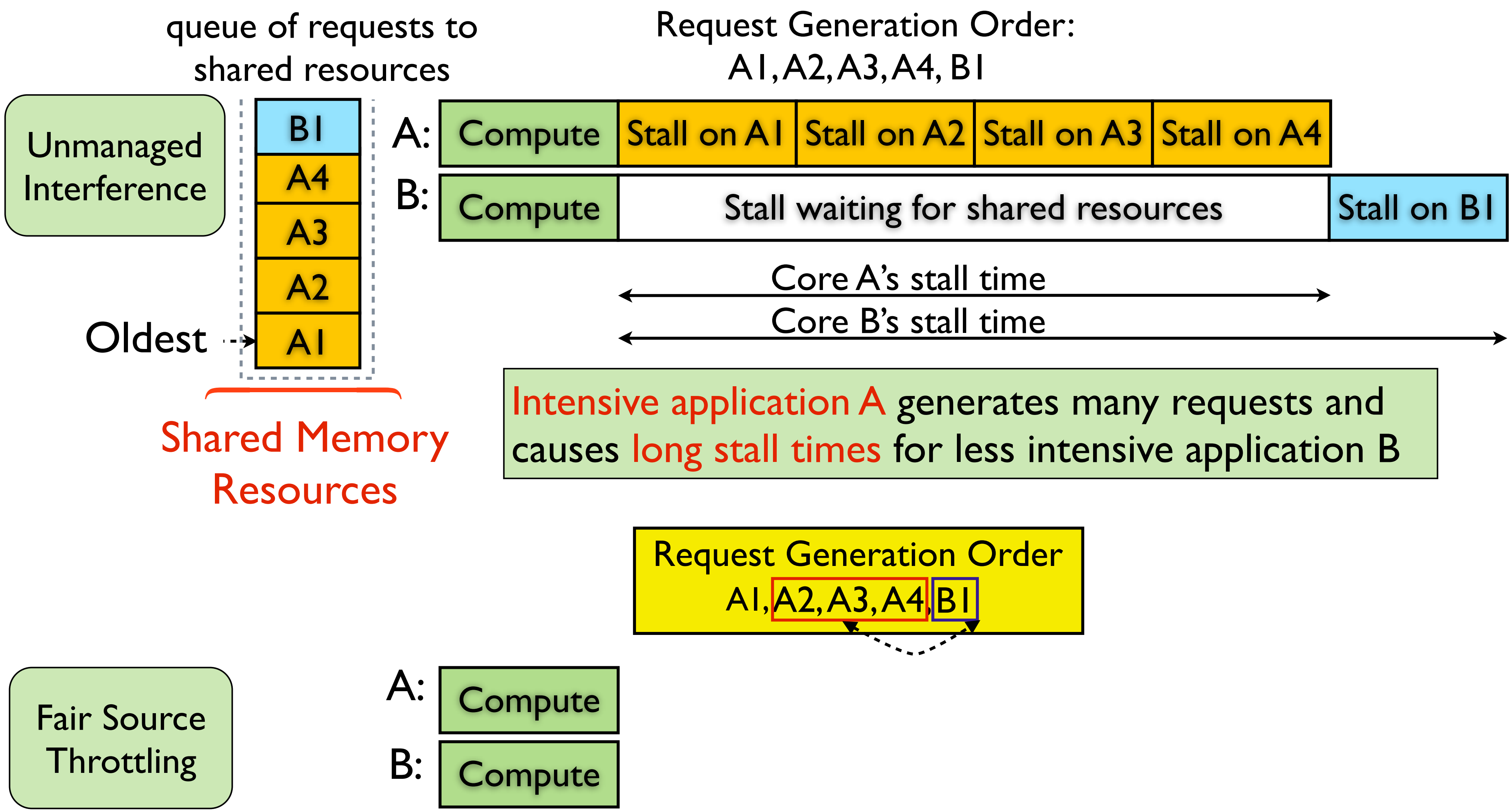
Request Generation Order:
A1, A2, A3, A4, B1



Intensive application A generates many requests and causes long stall times for less intensive application B

Request Generation Order
A1, A2, A3, A4, B1





queue of requests to shared resources

Request Generation Order: A1, A2, A3, A4, B1

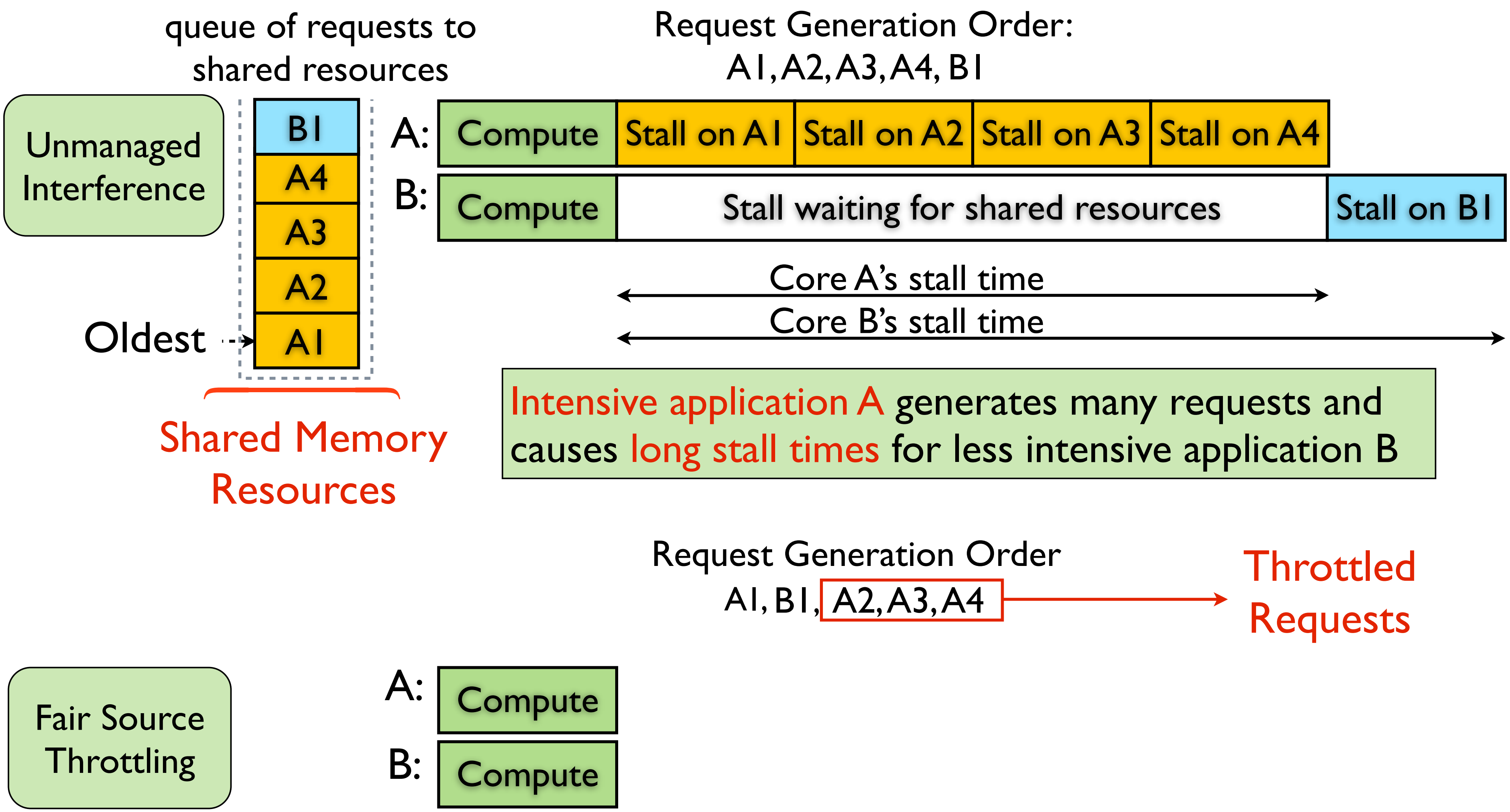
Unmanaged Interference

Shared Memory Resources

Intensive application A generates many requests and causes long stall times for less intensive application B

Request Generation Order
A1, A2, A3, A4, B1

Fair Source Throttling



Unmanaged Interference

queue of requests to shared resources

B1

A4

A3

A2

A1

Oldest

Shared Memory Resources

Request Generation Order:
A1, A2, A3, A4, B1

A: Compute | Stall on A1 | Stall on A2 | Stall on A3 | Stall on A4

B: Compute | Stall waiting for shared resources | Stall on B1

Core A's stall time

Core B's stall time

Intensive application A generates many requests and causes long stall times for less intensive application B

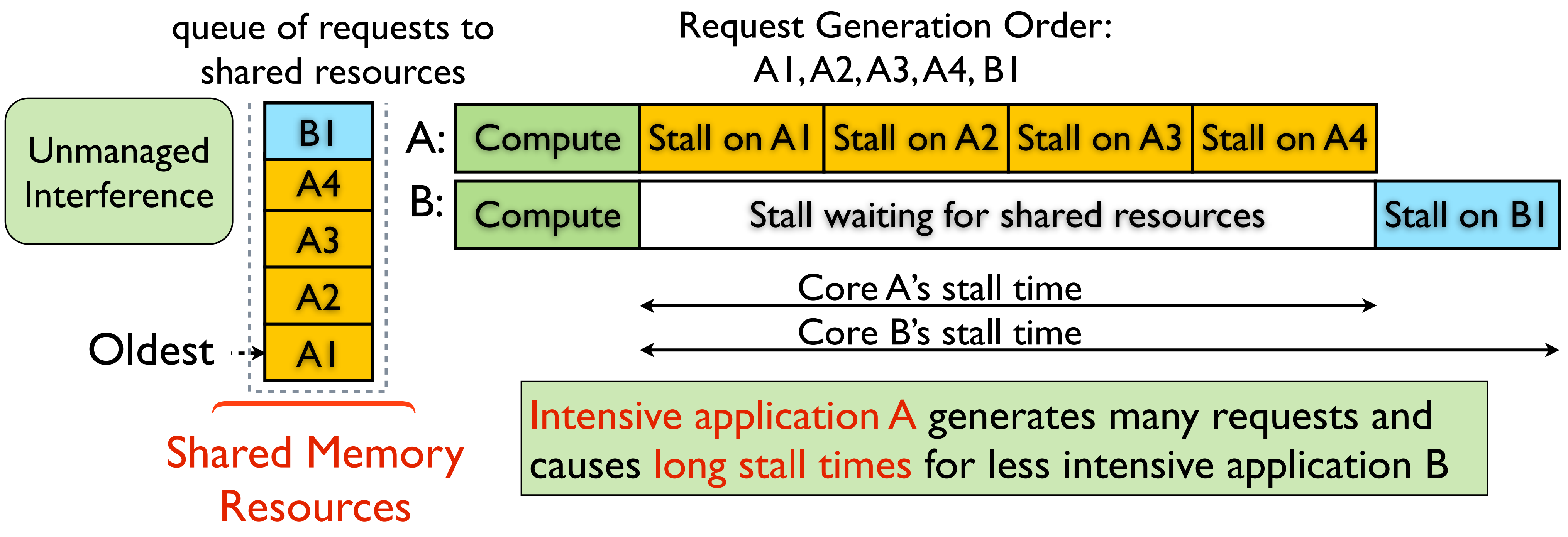
Request Generation Order
A1, B1, A2, A3, A4

Throttled Requests

Fair Source Throttling

A: Compute

B: Compute



queue of requests to shared resources

Unmanaged Interference

B1

A4

A3

A2

Oldest A1

Shared Memory Resources

Request Generation Order:
A1, A2, A3, A4, B1

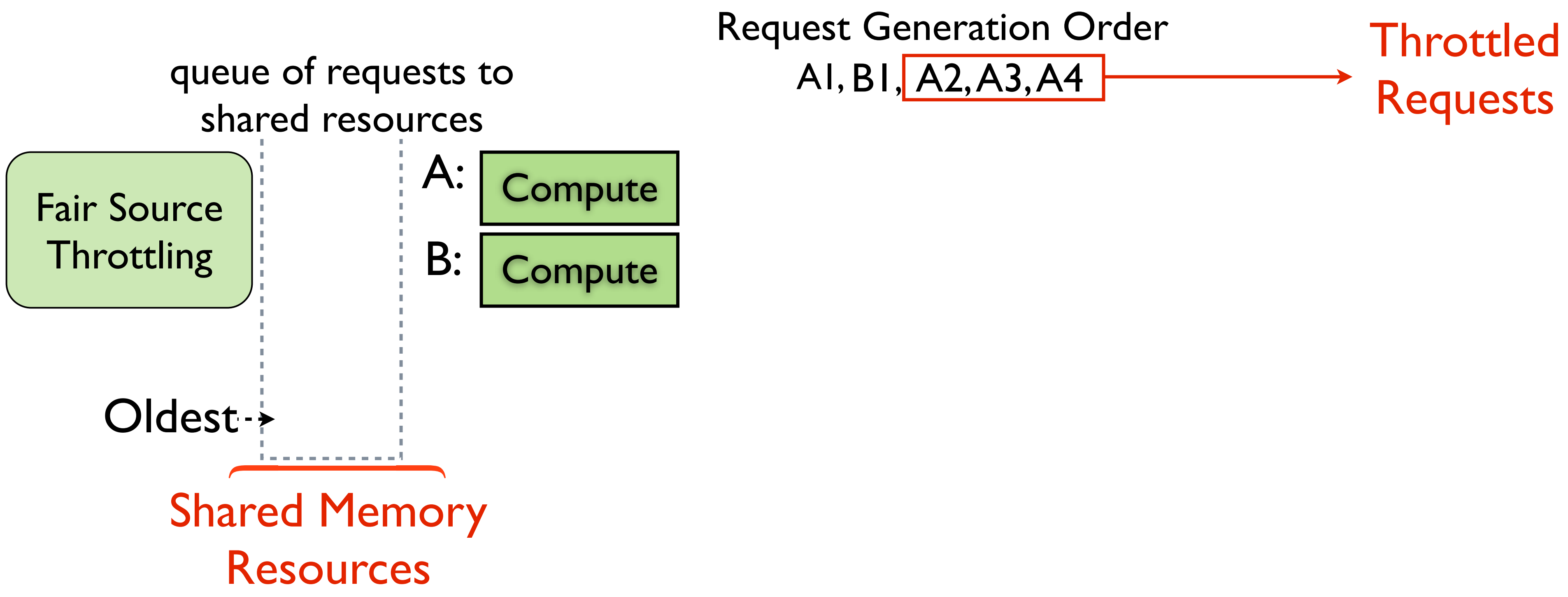
A: Compute | Stall on A1 | Stall on A2 | Stall on A3 | Stall on A4

B: Compute | Stall waiting for shared resources | Stall on B1

Core A's stall time

Core B's stall time

Intensive application A generates many requests and causes long stall times for less intensive application B



queue of requests to shared resources

Fair Source Throttling

Oldest

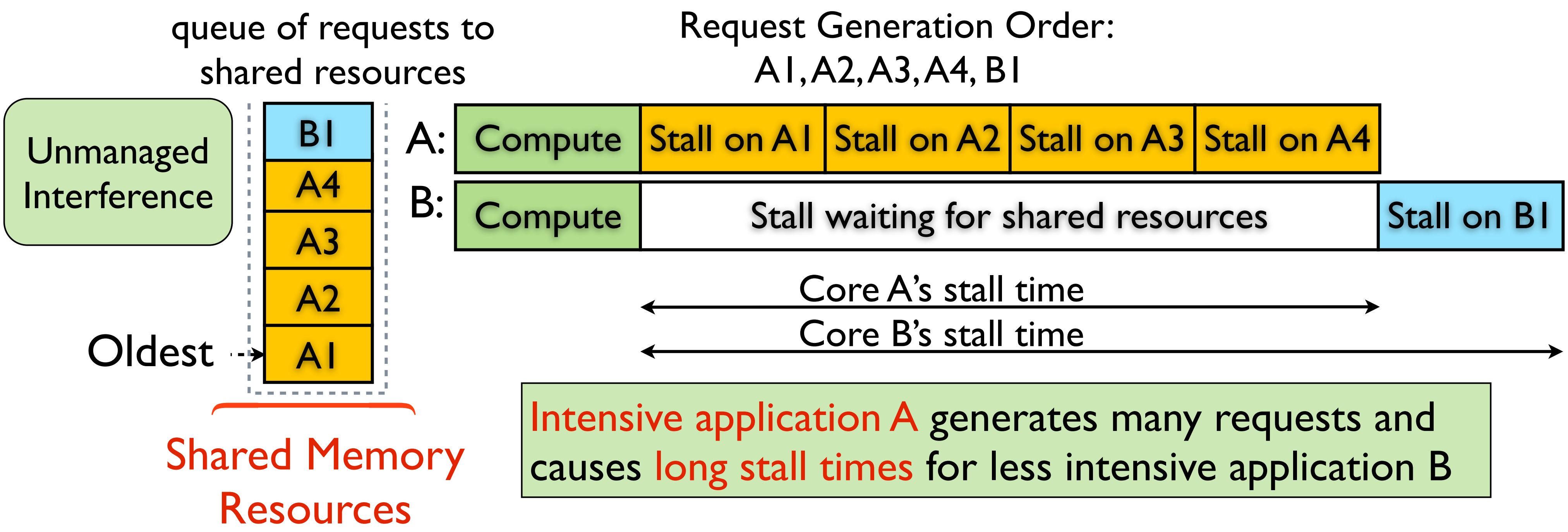
Shared Memory Resources

Request Generation Order
A1, B1, A2, A3, A4

Throttled Requests

A: Compute

B: Compute



queue of requests to shared resources

Request Generation Order: A1, A2, A3, A4, B1

Unmanaged Interference

B1
A4
A3
A2
Oldest A1

A: Compute | Stall on A1 | Stall on A2 | Stall on A3 | Stall on A4

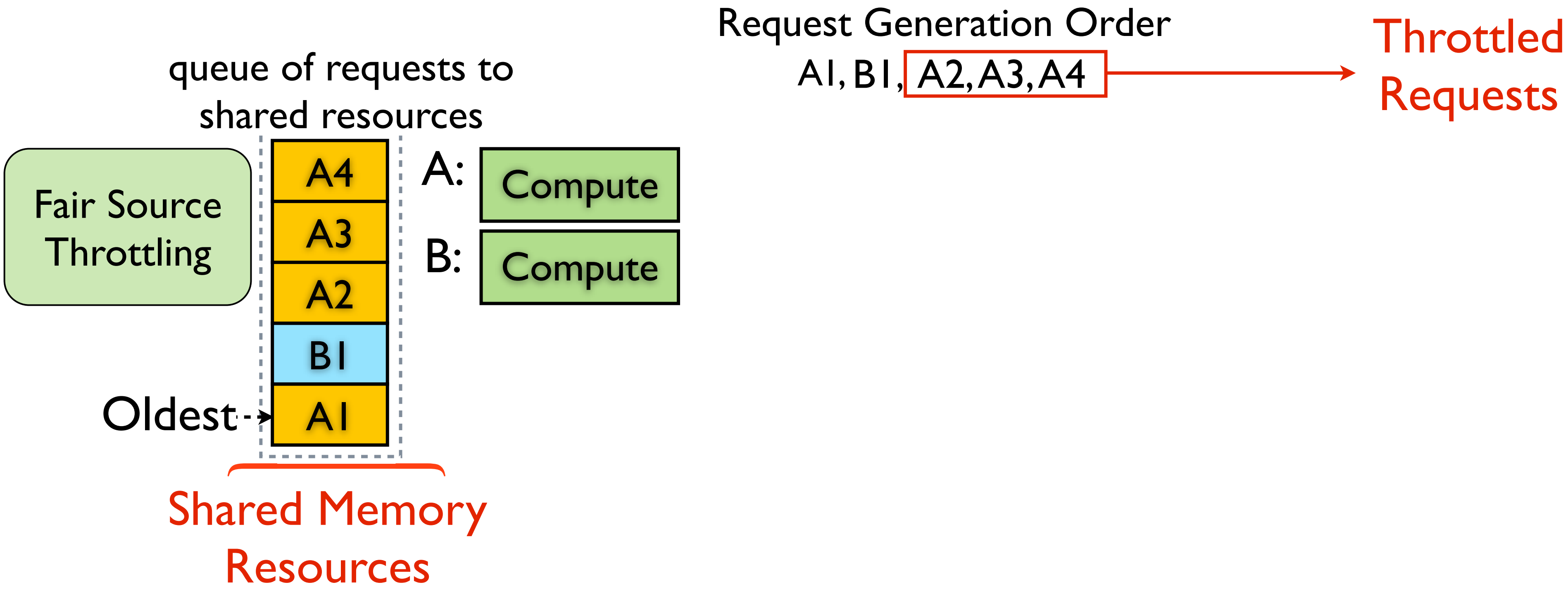
B: Compute | Stall waiting for shared resources | Stall on B1

Core A's stall time

Core B's stall time

Shared Memory Resources

Intensive application A generates many requests and causes long stall times for less intensive application B



Request Generation Order

A1, B1, A2, A3, A4

Throttled Requests

queue of requests to shared resources

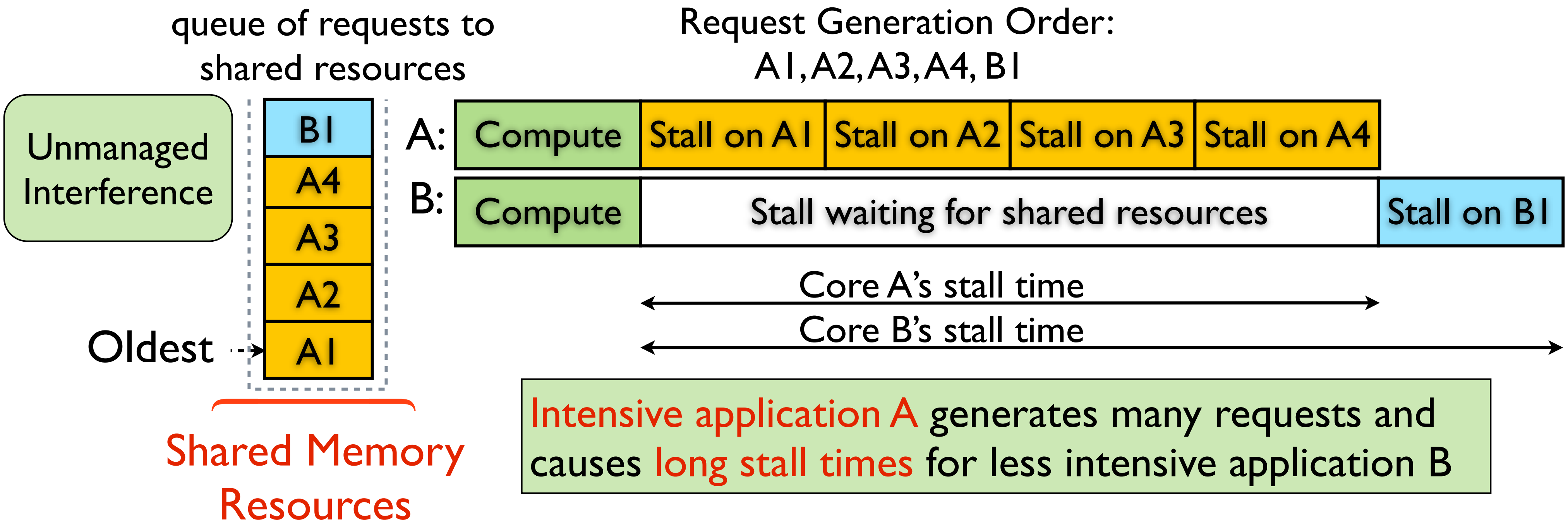
Fair Source Throttling

A4
A3
A2
B1
Oldest A1

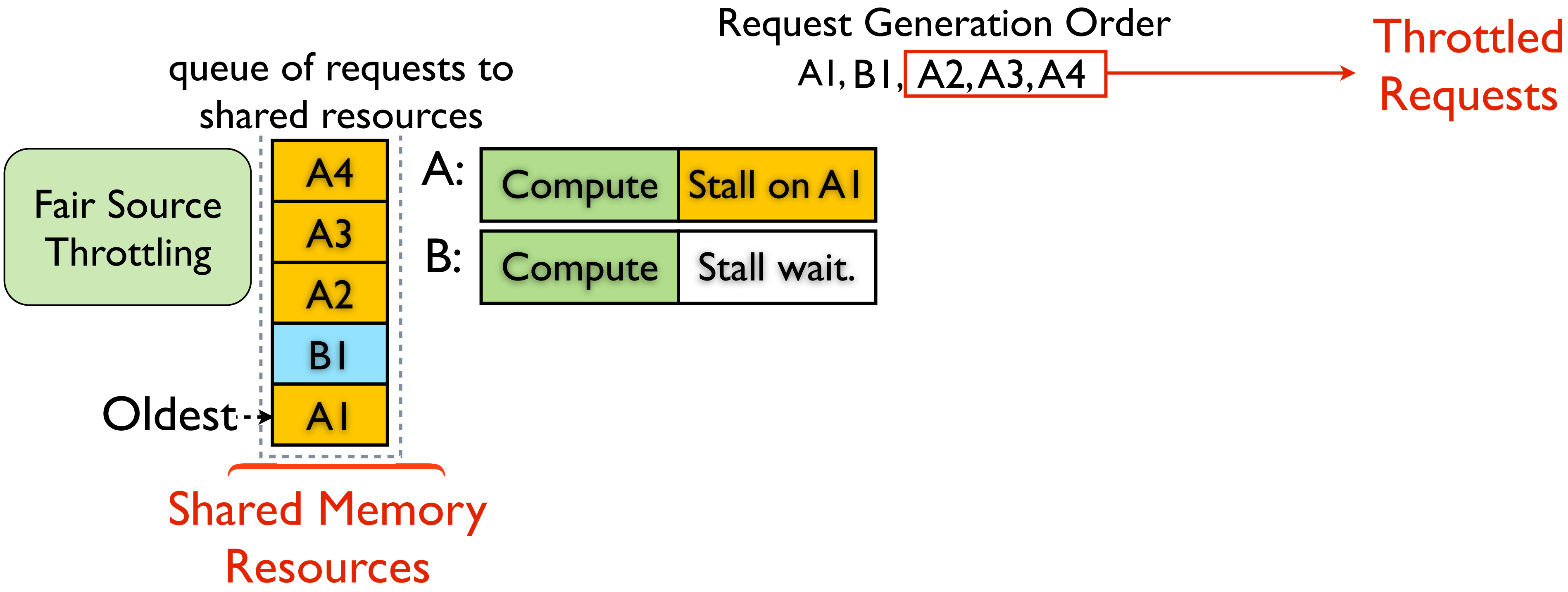
A: Compute

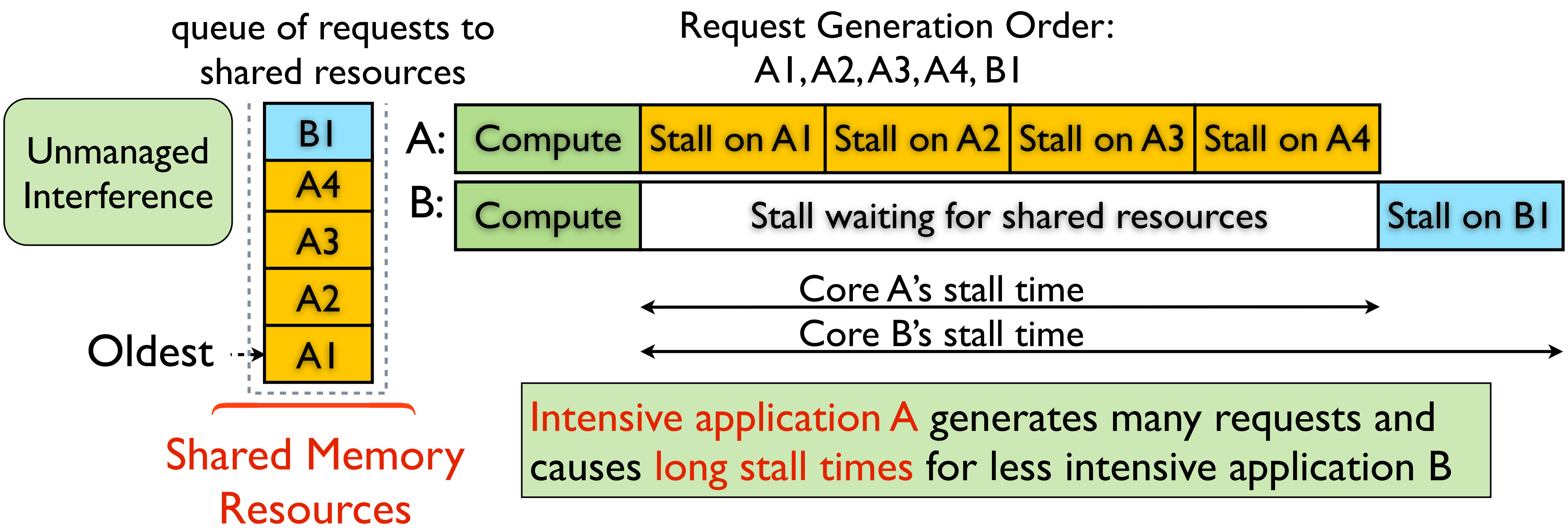
B: Compute

Shared Memory Resources



Intensive application A generates many requests and causes long stall times for less intensive application B





Unmanaged Interference

queue of requests to shared resources

B1

A4

A3

A2

Oldest → A1

Shared Memory Resources

Request Generation Order:
A1, A2, A3, A4, B1

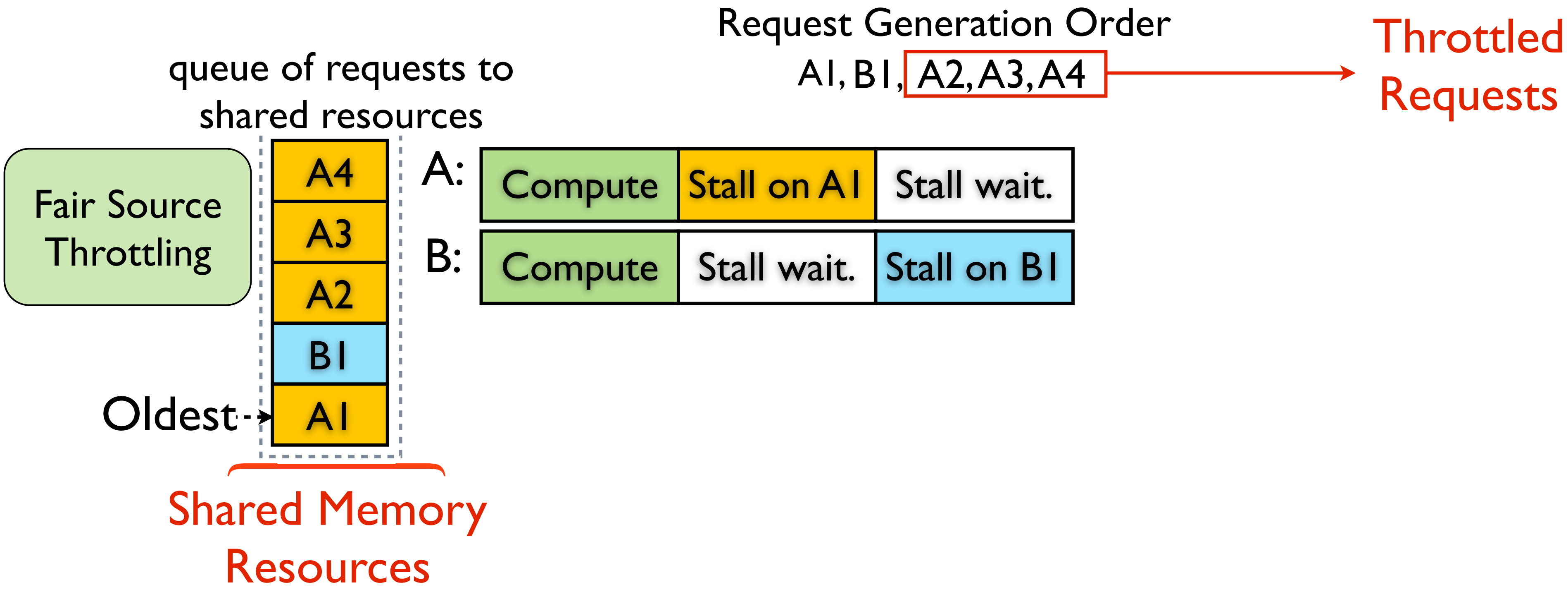
A: Compute | Stall on A1 | Stall on A2 | Stall on A3 | Stall on A4

B: Compute | Stall waiting for shared resources | Stall on B1

Core A's stall time

Core B's stall time

Intensive application A generates many requests and causes long stall times for less intensive application B



Fair Source Throttling

queue of requests to shared resources

A4

A3

A2

B1

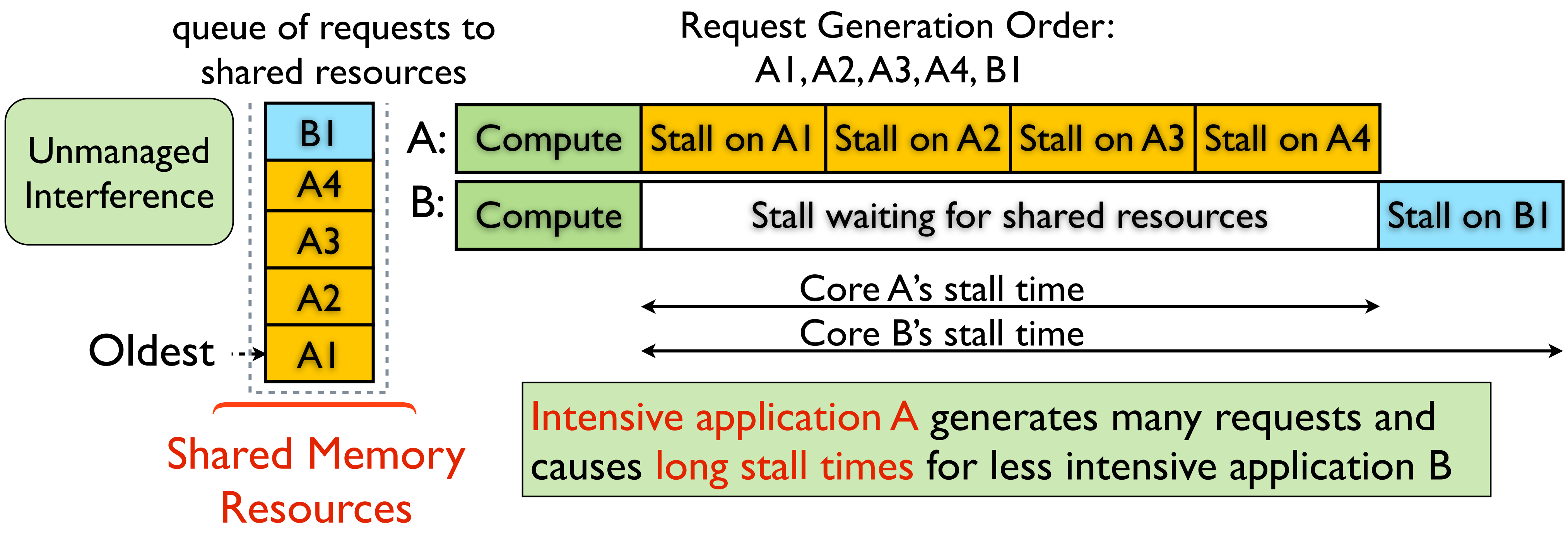
Oldest → A1

Shared Memory Resources

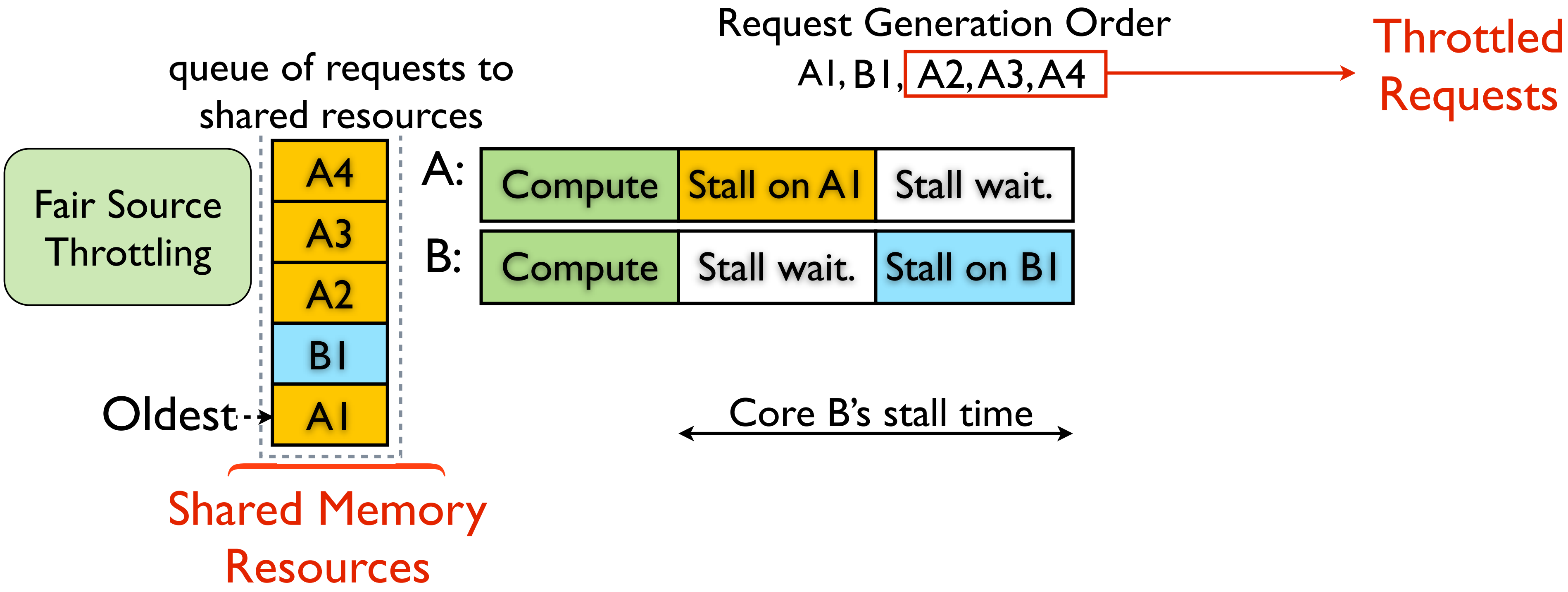
Request Generation Order
A1, B1, A2, A3, A4 → Throttled Requests

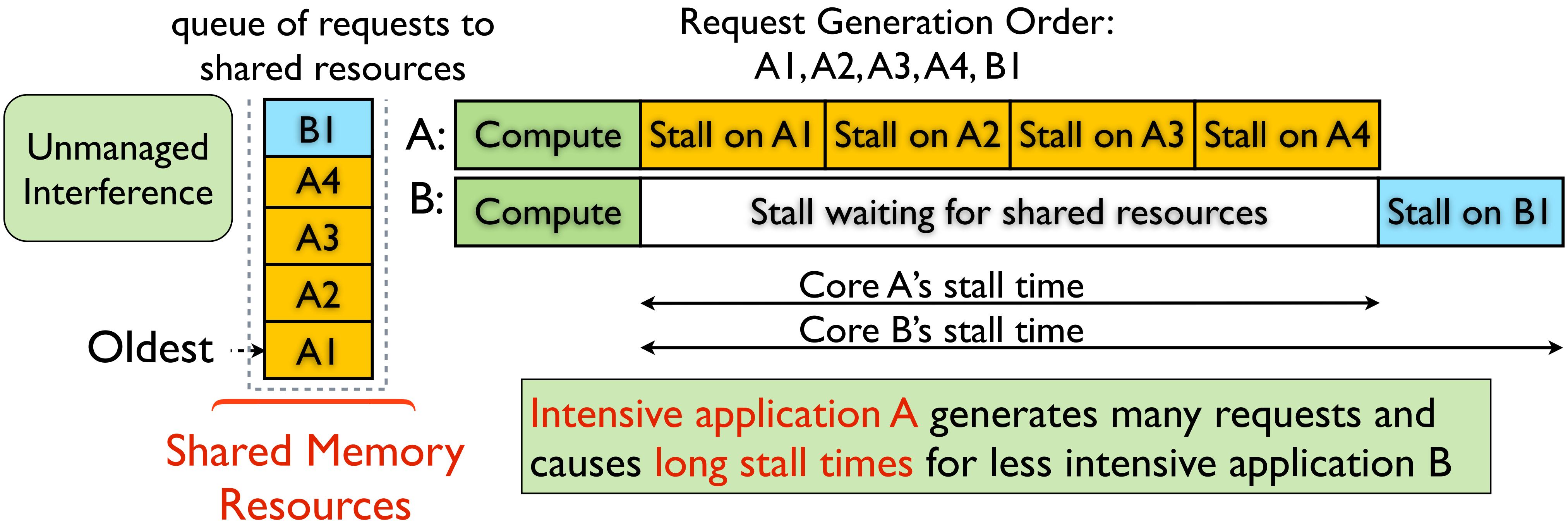
A: Compute | Stall on A1 | Stall wait.

B: Compute | Stall wait. | Stall on B1



Intensive application A generates many requests and causes long stall times for less intensive application B





queue of requests to shared resources

Request Generation Order:
A1, A2, A3, A4, B1

Unmanaged Interference

B1
A4
A3
A2
Oldest A1

A: Compute | Stall on A1 | Stall on A2 | Stall on A3 | Stall on A4

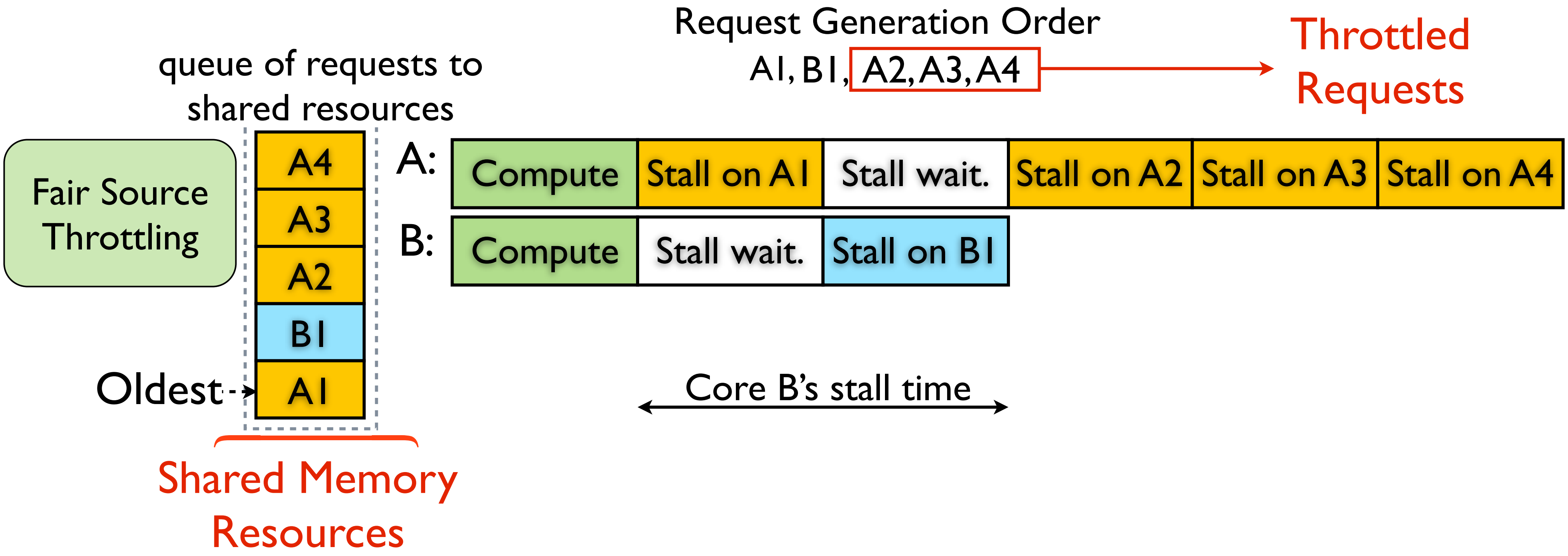
B: Compute | Stall waiting for shared resources | Stall on B1

Core A's stall time

Core B's stall time

Shared Memory Resources

Intensive application A generates many requests and causes long stall times for less intensive application B



Request Generation Order
A1, B1, A2, A3, A4

Throttled Requests

queue of requests to shared resources

Fair Source Throttling

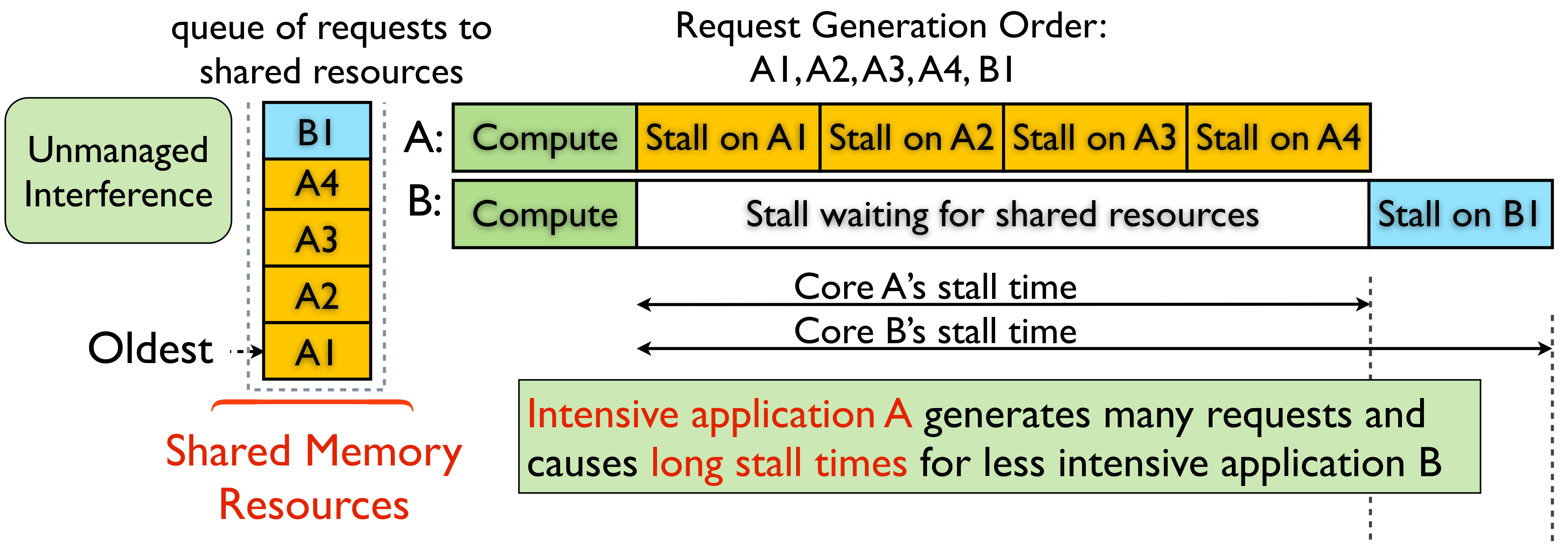
A4
A3
A2
B1
Oldest A1

A: Compute | Stall on A1 | Stall wait. | Stall on A2 | Stall on A3 | Stall on A4

B: Compute | Stall wait. | Stall on B1

Core B's stall time

Shared Memory Resources



Unmanaged Interference

queue of requests to shared resources

B1

A4

A3

A2

Oldest → A1

Shared Memory Resources

Request Generation Order:
A1, A2, A3, A4, B1

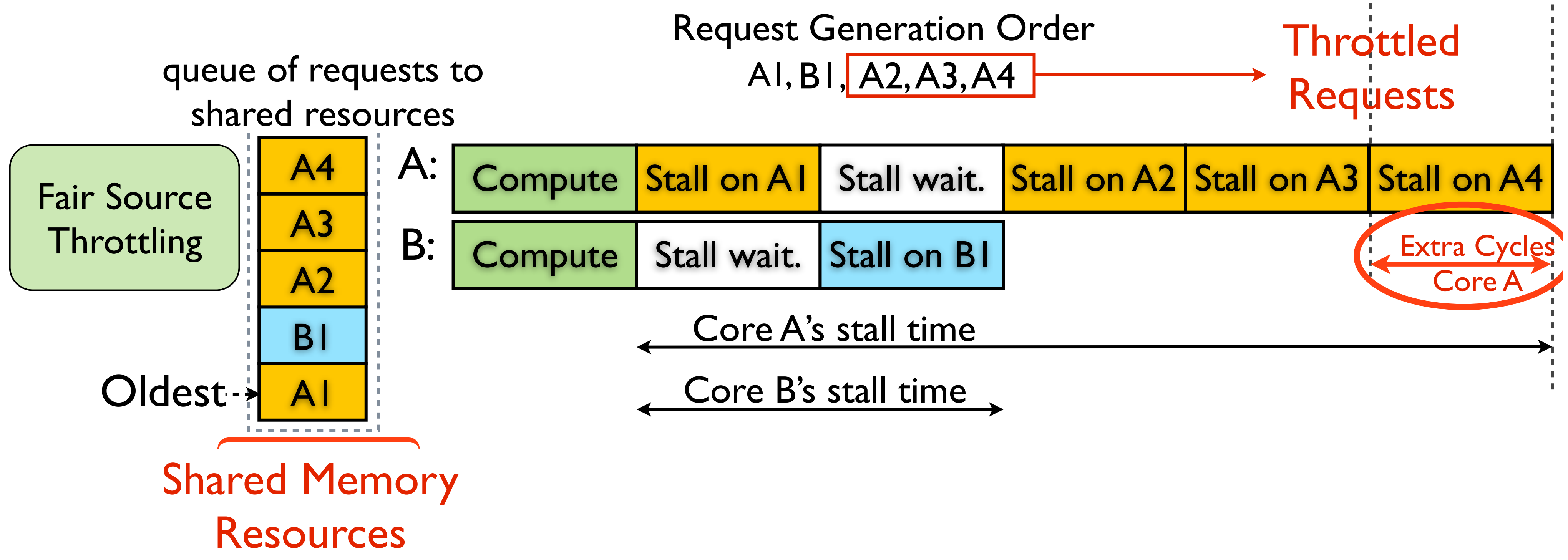
A: Compute | Stall on A1 | Stall on A2 | Stall on A3 | Stall on A4

B: Compute | Stall waiting for shared resources | Stall on B1

Core A's stall time

Core B's stall time

Intensive application A generates many requests and causes long stall times for less intensive application B



Fair Source Throttling

queue of requests to shared resources

A4

A3

A2

B1

Oldest → A1

Shared Memory Resources

Request Generation Order
A1, B1, A2, A3, A4 → Throttled Requests

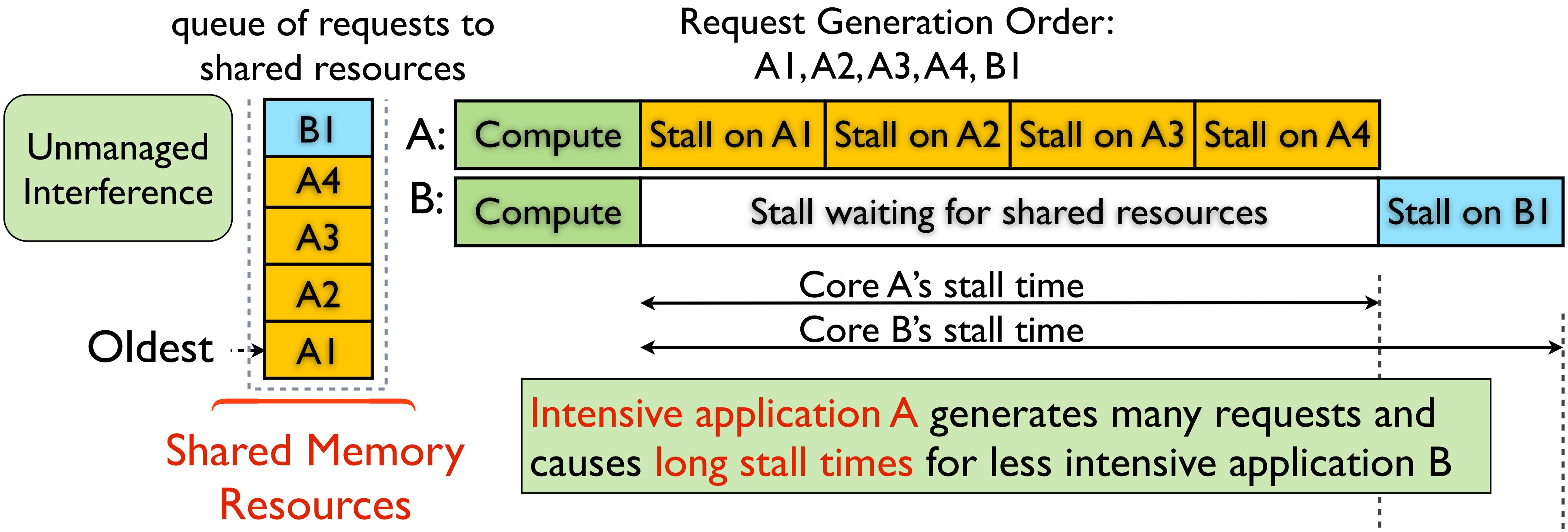
A: Compute | Stall on A1 | Stall wait. | Stall on A2 | Stall on A3 | Stall on A4

B: Compute | Stall wait. | Stall on B1

Extra Cycles Core A

Core A's stall time

Core B's stall time



queue of requests to shared resources

Request Generation Order: A1, A2, A3, A4, B1

Unmanaged Interference

B1
A4
A3
A2
Oldest A1

A: Compute | Stall on A1 | Stall on A2 | Stall on A3 | Stall on A4

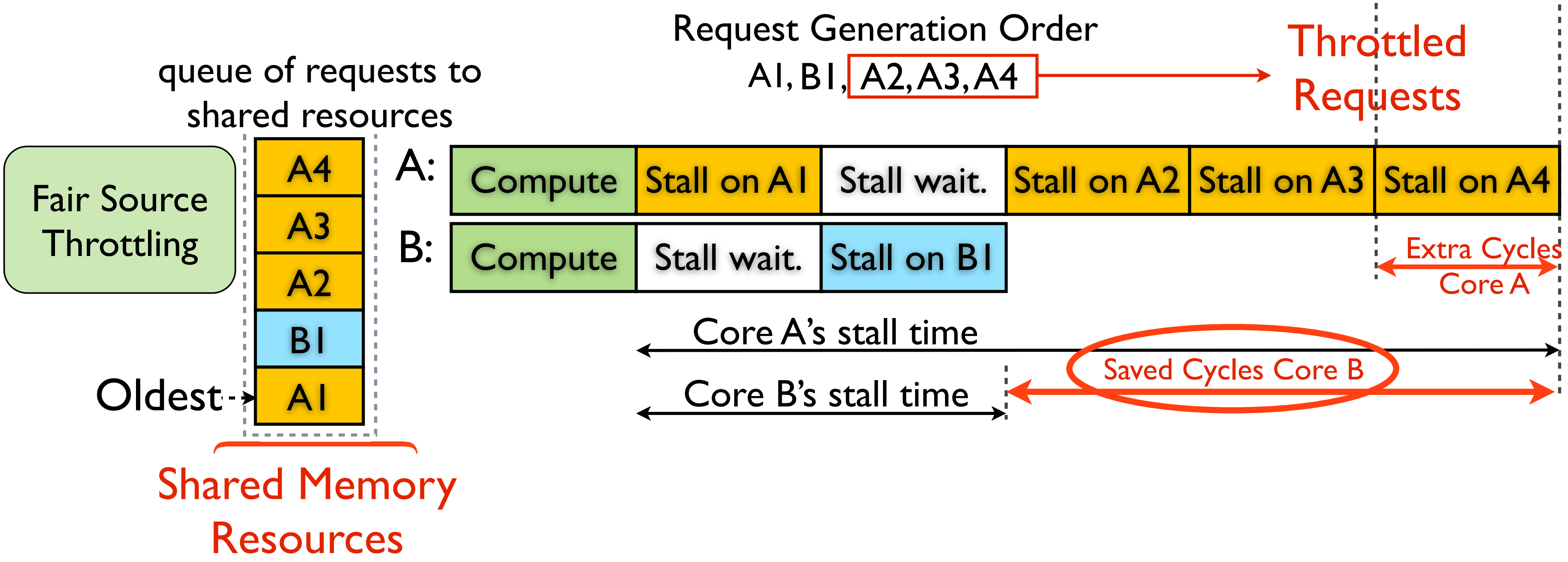
B: Compute | Stall waiting for shared resources | Stall on B1

Core A's stall time

Core B's stall time

Shared Memory Resources

Intensive application A generates many requests and causes long stall times for less intensive application B



Request Generation Order A1, B1, A2, A3, A4

Throttled Requests

queue of requests to shared resources

Fair Source Throttling

A4
A3
A2
B1
Oldest A1

A: Compute | Stall on A1 | Stall wait. | Stall on A2 | Stall on A3 | Stall on A4

B: Compute | Stall wait. | Stall on B1

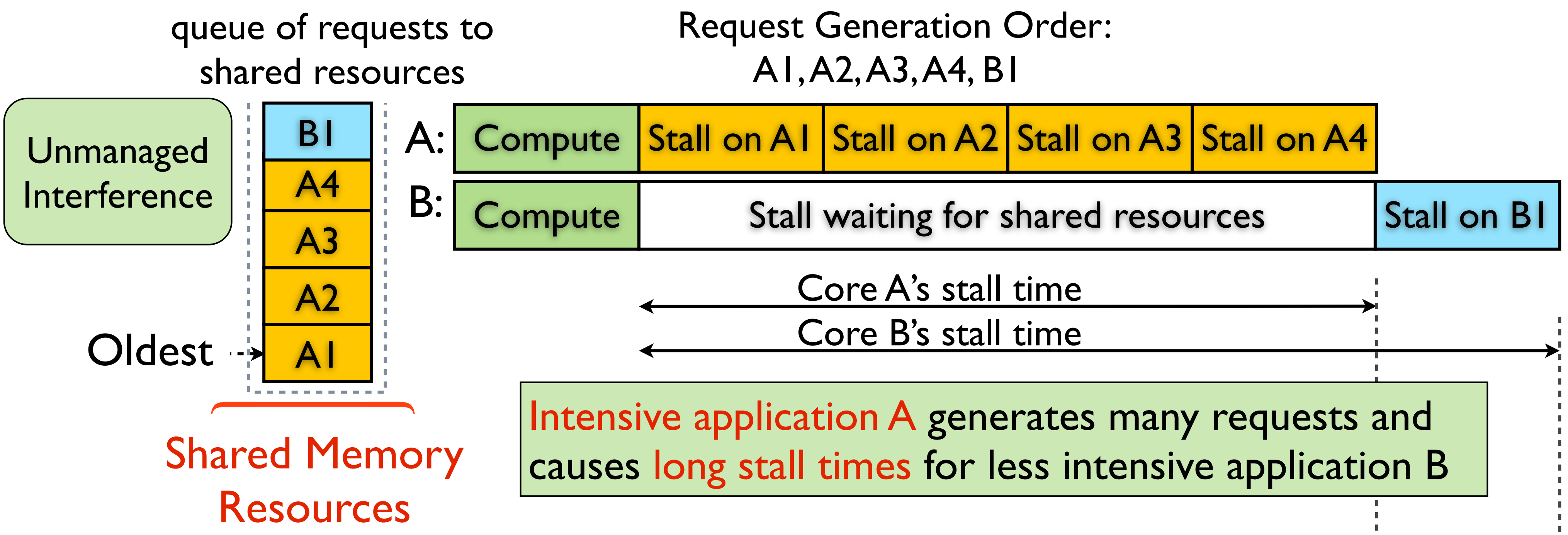
Extra Cycles Core A

Core A's stall time

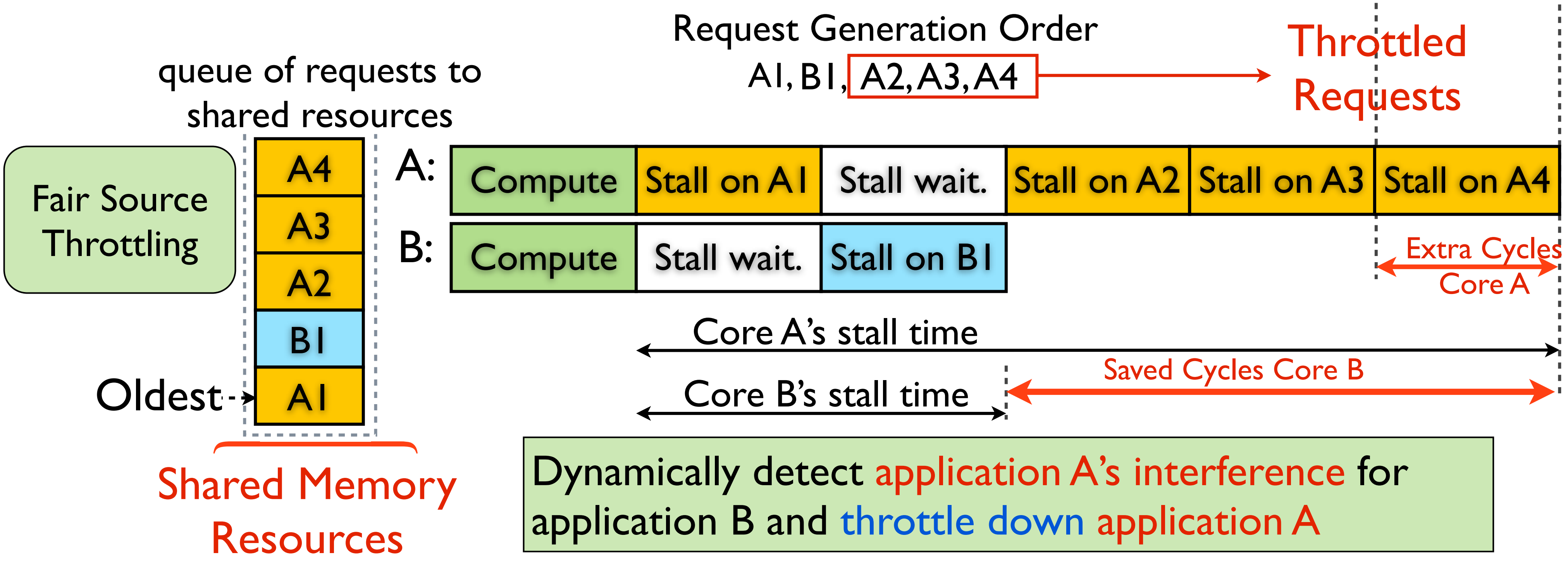
Saved Cycles Core B

Core B's stall time

Shared Memory Resources



Intensive application A generates many requests and causes long stall times for less intensive application B



Dynamically detect application A's interference for application B and throttle down application A

Outline

- Background and Problem
- Motivation for Source Throttling
- Fairness via Source Throttling (FST)
- Evaluation
- Conclusion

Fairness via Source Throttling (FST)

Fairness via Source Throttling (FST)

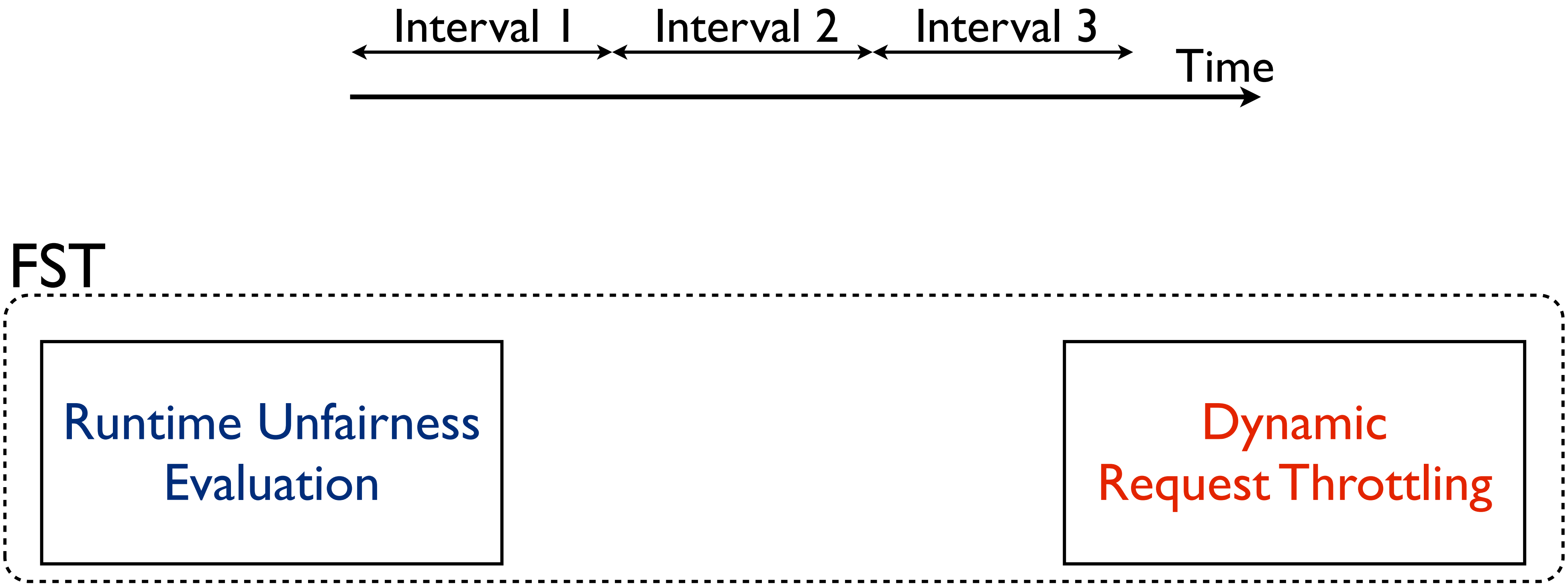
- Runtime Unfairness Evaluation
  - Dynamically estimates the unfairness in the memory system

Fairness via Source Throttling (FST)

- Runtime Unfairness Evaluation
  - Dynamically estimates the unfairness in the memory system

- Dynamic Request Throttling
  - Adjusts how aggressively each core makes requests to the shared resources

Fairness via Source Throttling (FST)



Interval 1

Interval 2

Interval 3

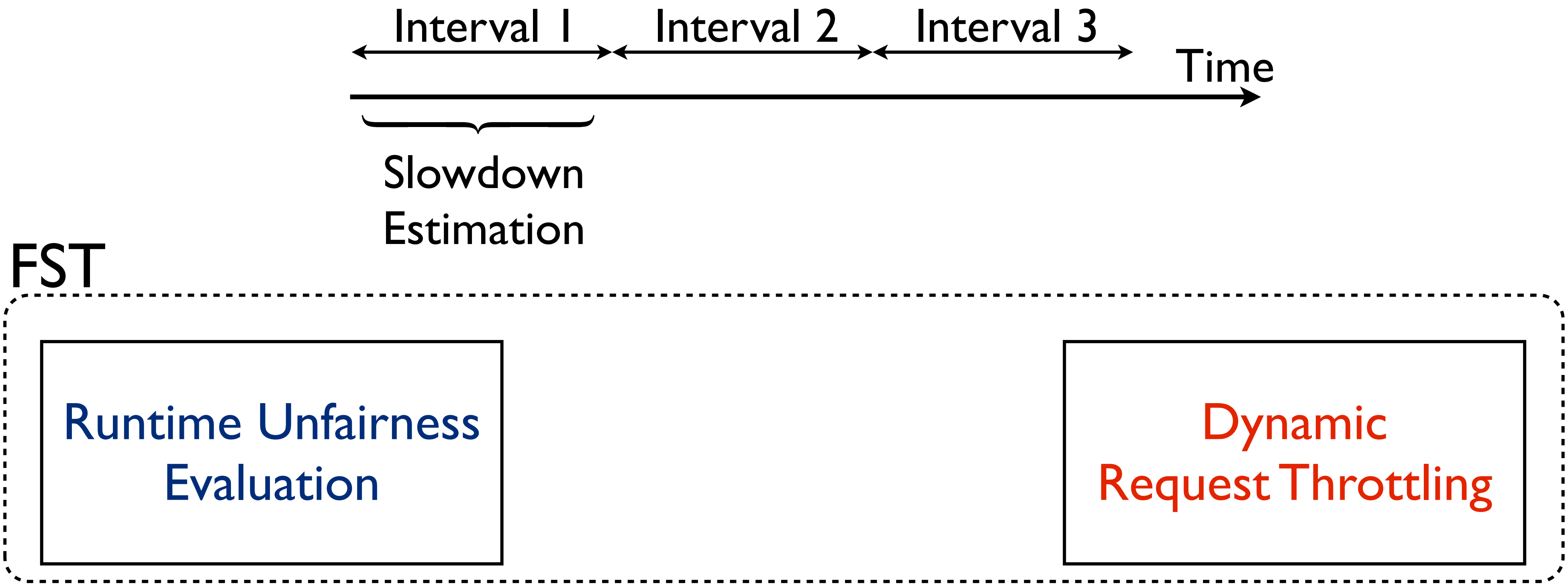
Time

FST

Runtime Unfairness Evaluation

Dynamic Request Throttling

Fairness via Source Throttling (FST)



Interval 1

Interval 2

Interval 3

Time

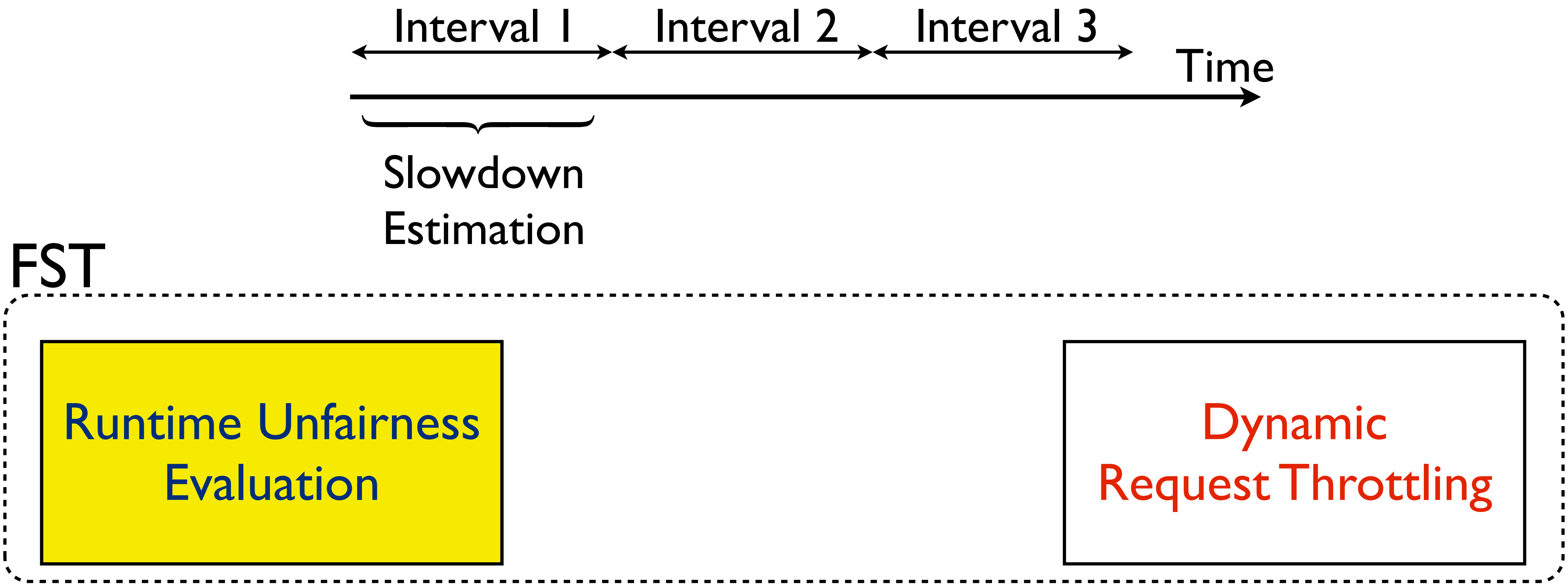
Slowdown Estimation

FST

Runtime Unfairness Evaluation

Dynamic Request Throttling

Fairness via Source Throttling (FST)



Interval 1

Interval 2

Interval 3

Time

Slowdown Estimation

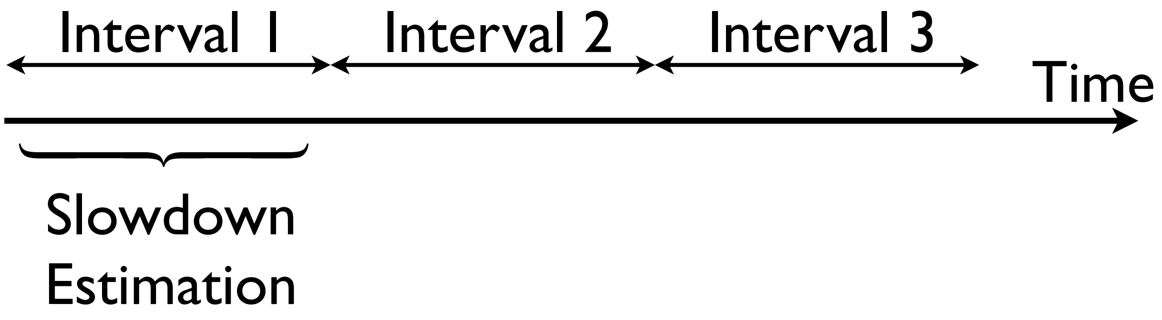
FST

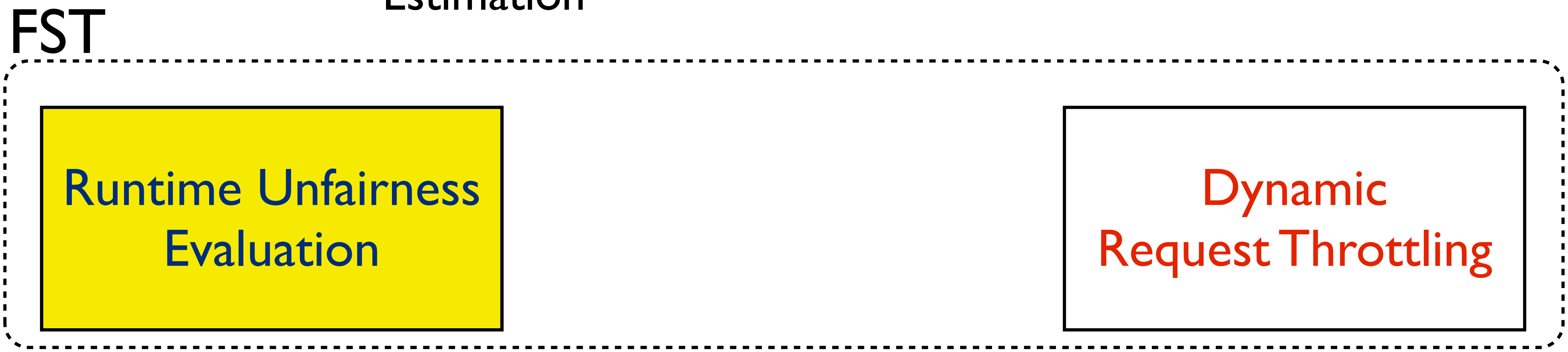
Runtime Unfairness Evaluation

Dynamic Request Throttling

1- Estimating system unfairness

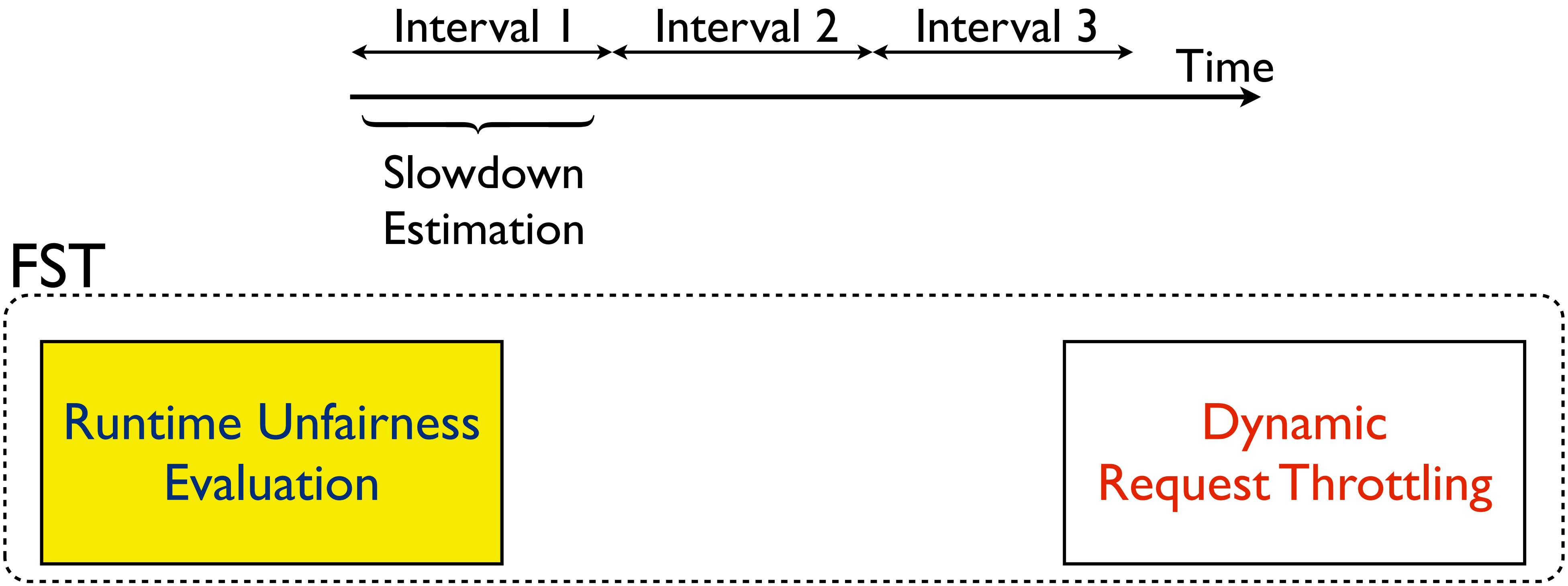
Fairness via Source Throttling (FST)





1- Estimating system unfairness
2- Find app. with the highest slowdown (App-slowest)

Fairness via Source Throttling (FST)



Interval 1

Interval 2

Interval 3

Time

Slowdown Estimation

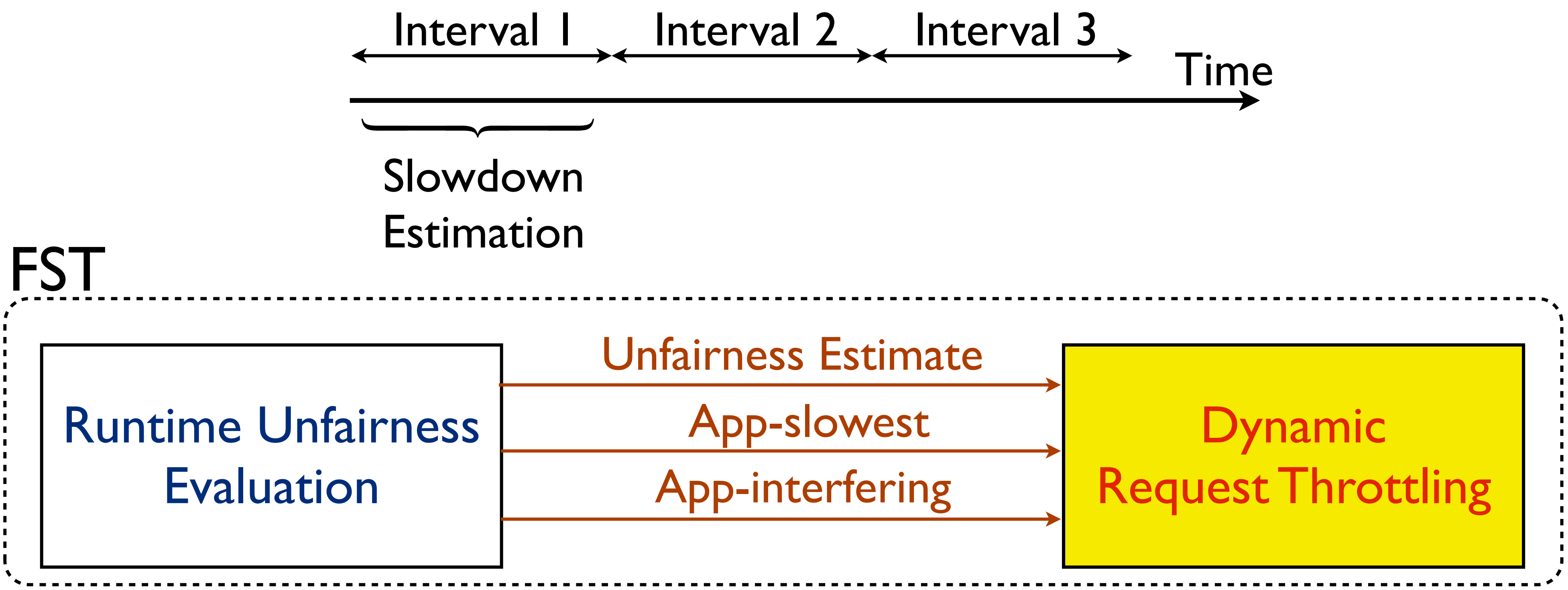
FST

Runtime Unfairness Evaluation

Dynamic Request Throttling

1- Estimating system unfairness
2- Find app. with the highest slowdown (App-slowest)
3- Find app. causing most interference for App-slowest (App-interfering)

Fairness via Source Throttling (FST)



Interval 1

Interval 2

Interval 3

Time

Slowdown Estimation

FST

Runtime Unfairness Evaluation

Unfairness Estimate

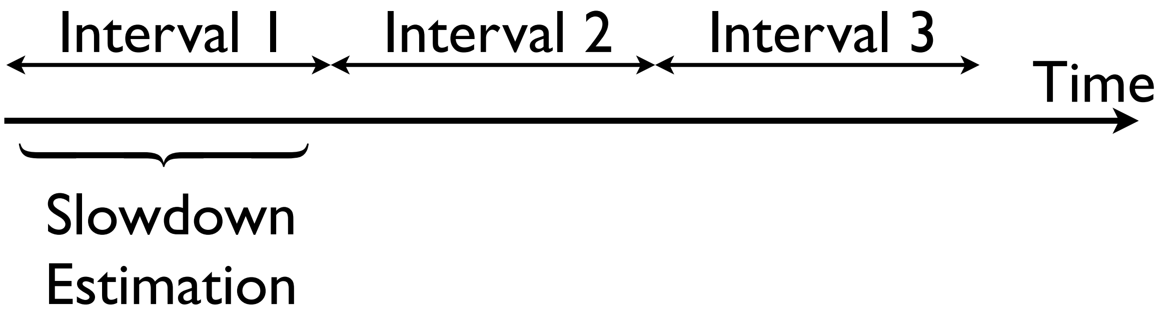
App-slowest

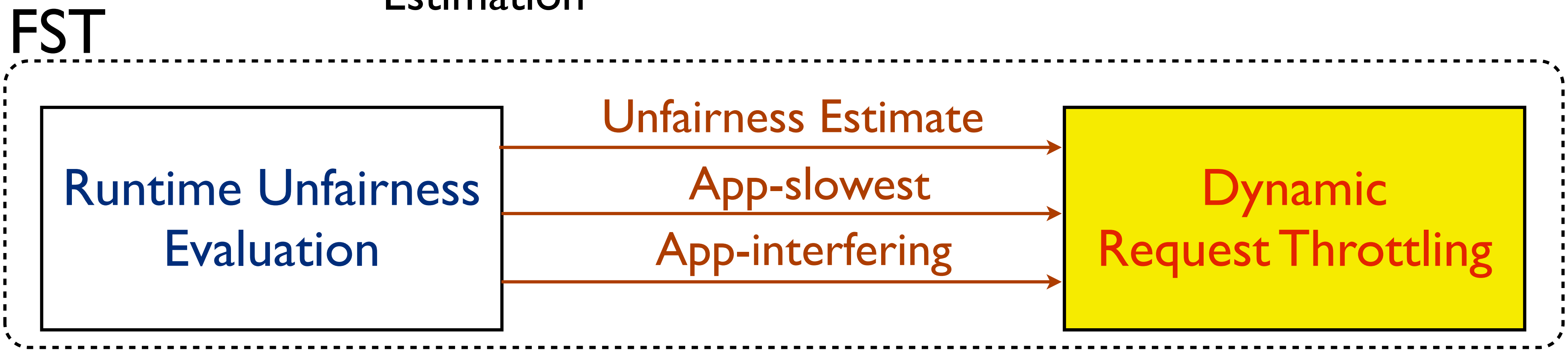
App-interfering

Dynamic Request Throttling

1- Estimating system unfairness
2- Find app. with the highest slowdown (App-slowest)
3- Find app. causing most interference for App-slowest (App-interfering)

Fairness via Source Throttling (FST)

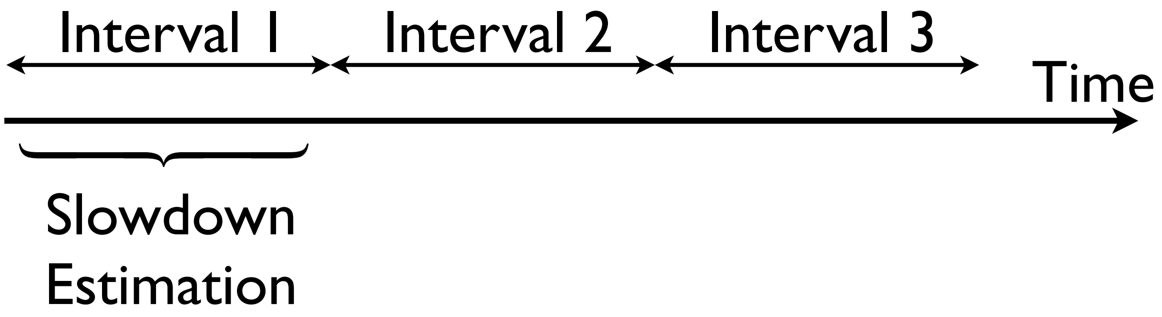




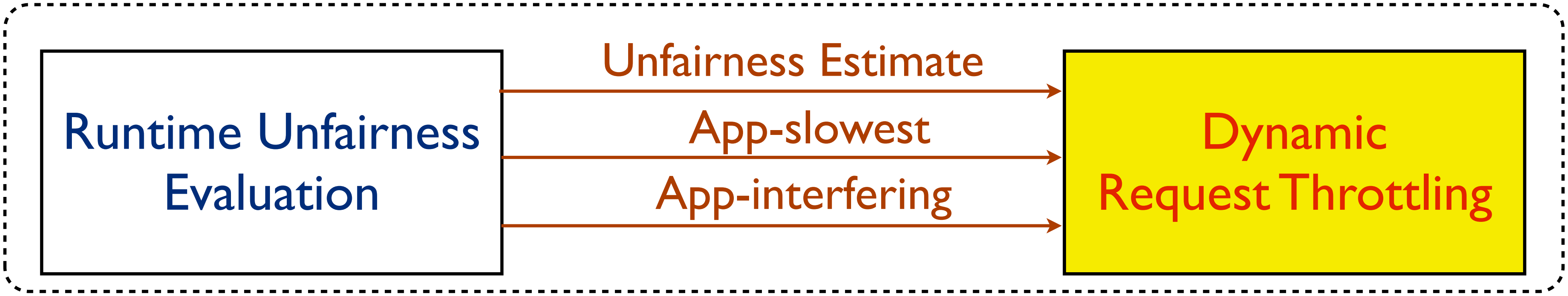
1- Estimating system unfairness
2- Find app. with the highest slowdown (App-slowest)
3- Find app. causing most interference for App-slowest (App-interfering)

```
if (Unfairness Estimate >Target)
{
```

Fairness via Source Throttling (FST)



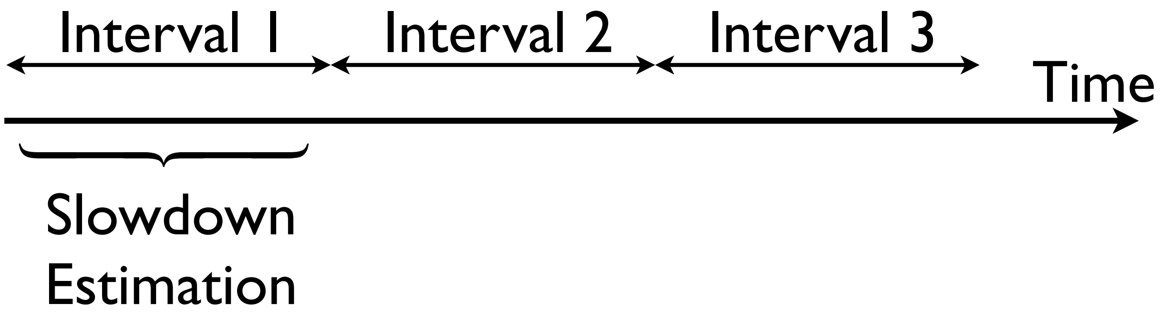
FST



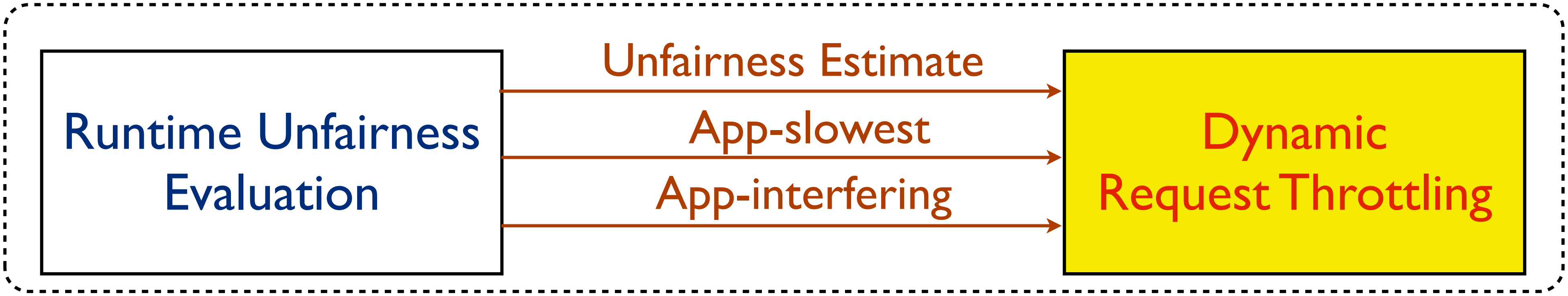
1- Estimating system unfairness
2- Find app. with the highest slowdown (App-slowest)
3- Find app. causing most interference for App-slowest (App-interfering)

```
if (Unfairness Estimate >Target)
{
 1-Throttle down App-interfering
```

Fairness via Source Throttling (FST)



FST

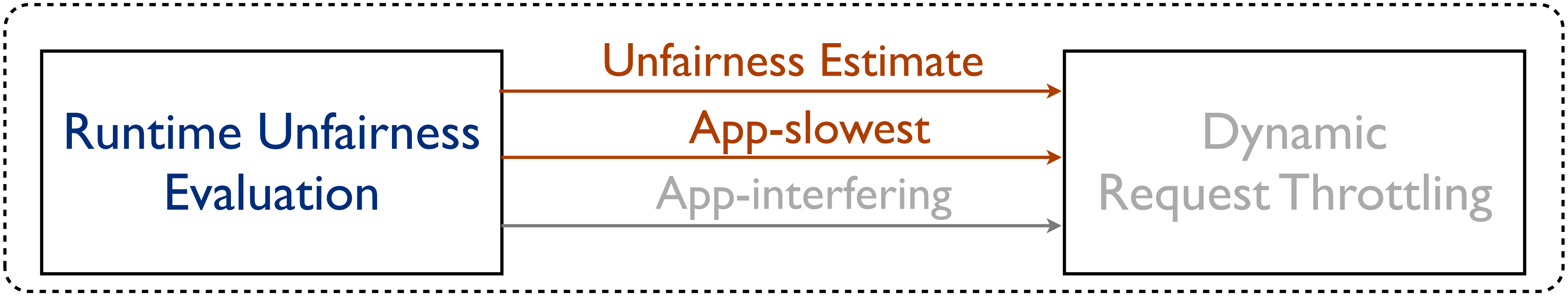


1- Estimating system unfairness
2- Find app. with the highest slowdown (App-slowest)
3- Find app. causing most interference for App-slowest (App-interfering)

```
if (Unfairness Estimate >Target)
{
 1-Throttle down App-interfering
 2-Throttle up App-slowest
}
```

Fairness via Source Throttling (FST)

FST



1- Estimating system unfairness
2- Find app. with the highest slowdown (App-slowest)
3- Find app. causing most interference for App-slowest (App-interfering)

```
if (Unfairness Estimate >Target)
{
 1-Throttle down App-interfering
 2-Throttle up App-slowest
}
```

Estimating System Unfairness

- Unfairness = $\dfrac{\text{Max}\{\text{Slowdown } i\} \text{ over all applications } i}{\text{Min}\{\text{Slowdown } i\} \text{ over all applications } i}$

- Slowdown of application i = $\dfrac{T_i^{\text{Shared}}}{T_i^{\text{Alone}}}$

Estimating System Unfairness

- Unfairness = $\dfrac{\text{Max}\{\text{Slowdown } i\} \text{ over all applications } i}{\text{Min}\{\text{Slowdown } i\} \text{ over all applications } i}$

- Slowdown of application i = $\dfrac{T_i^{\text{Shared}}}{T_i^{\text{Alone}}}$

- How can T_i^{Alone} be estimated in shared mode?

Estimating System Unfairness

- Unfairness = $\dfrac{\text{Max}\{\text{Slowdown } i\} \text{ over all applications } i}{\text{Min}\{\text{Slowdown } i\} \text{ over all applications } i}$

- Slowdown of application i = $\dfrac{T_i^{Shared}}{T_i^{Alone}}$

- How can T_i^{Alone} be estimated in shared mode?

- T_i^{Excess} is the number of extra cycles it takes application i to execute due to interference

Estimating System Unfairness

- Unfairness = $\frac{\text{Max}\{\text{Slowdown } i\} \text{ over all applications } i}{\text{Min}\{\text{Slowdown } i\} \text{ over all applications } i}$

- Slowdown of application i = $\frac{T_i^{\text{Shared}}}{T_i^{\text{Alone}}}$

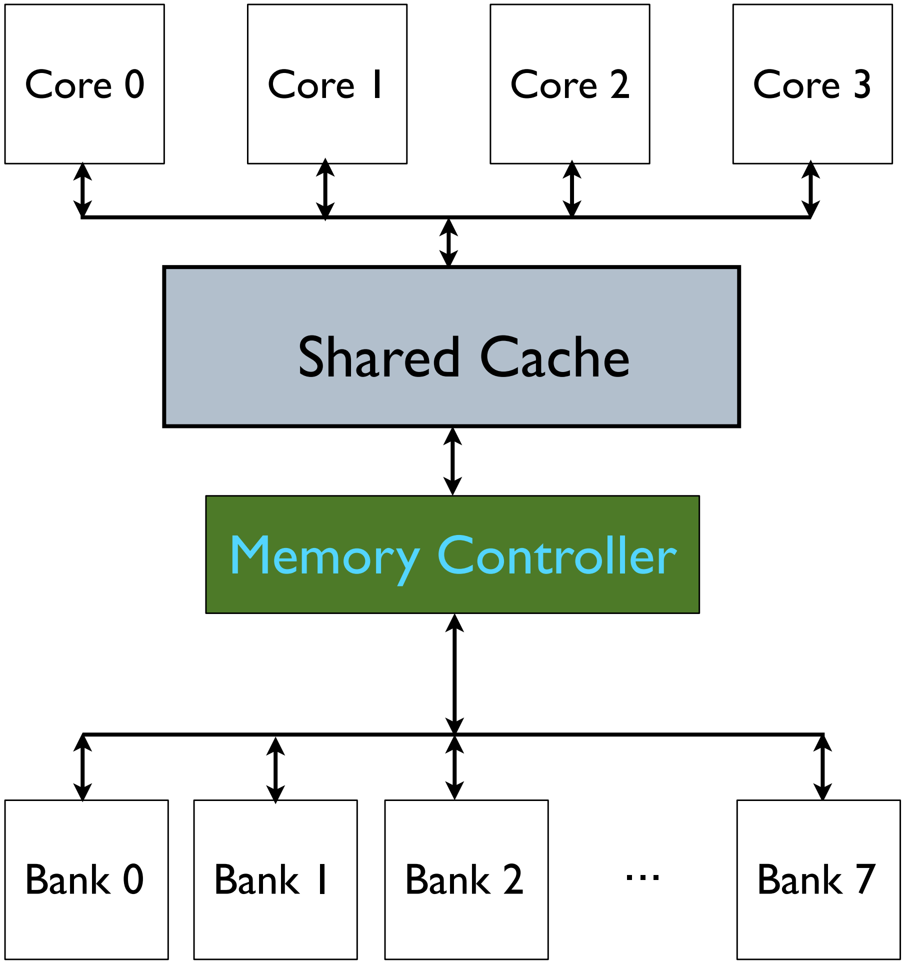
- How can T_i^{Alone} be estimated in shared mode?

- T_i^{Excess} is the number of extra cycles it takes application *i* to execute due to interference

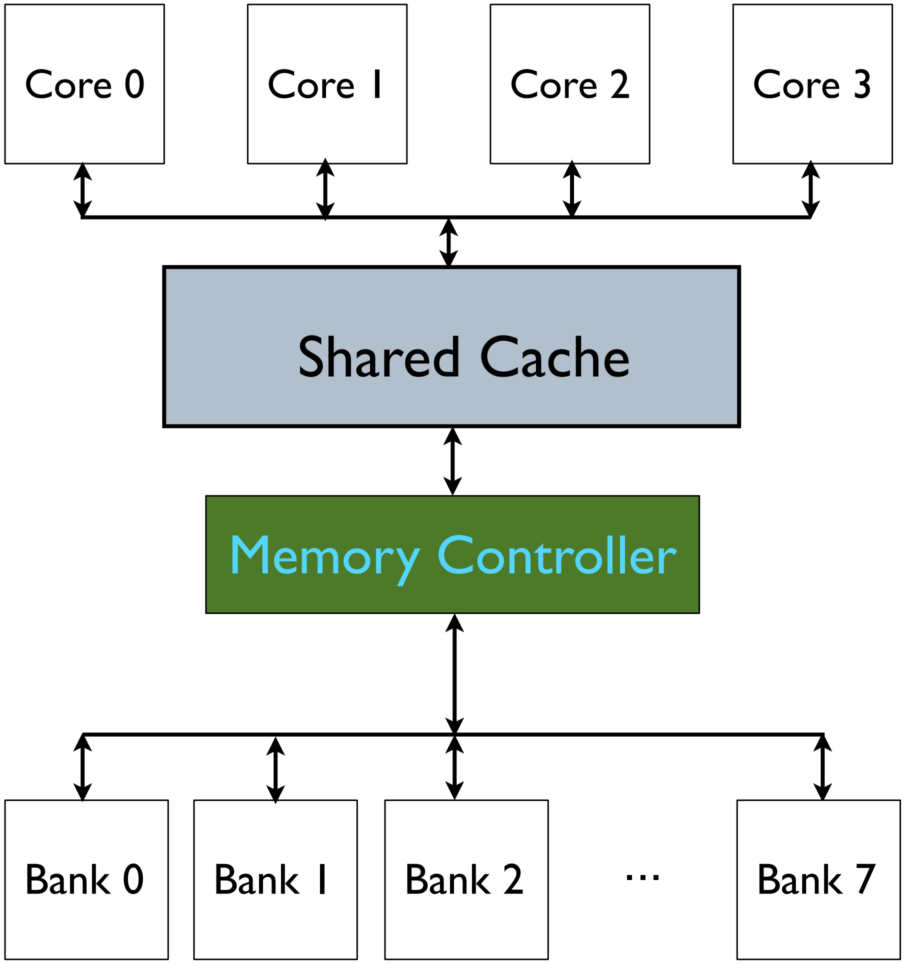
- $T_i^{\text{Alone}} = T_i^{\text{Shared}} - T_i^{\text{Excess}}$

Tracking Inter-Core Interference

Tracking Inter-Core Interference

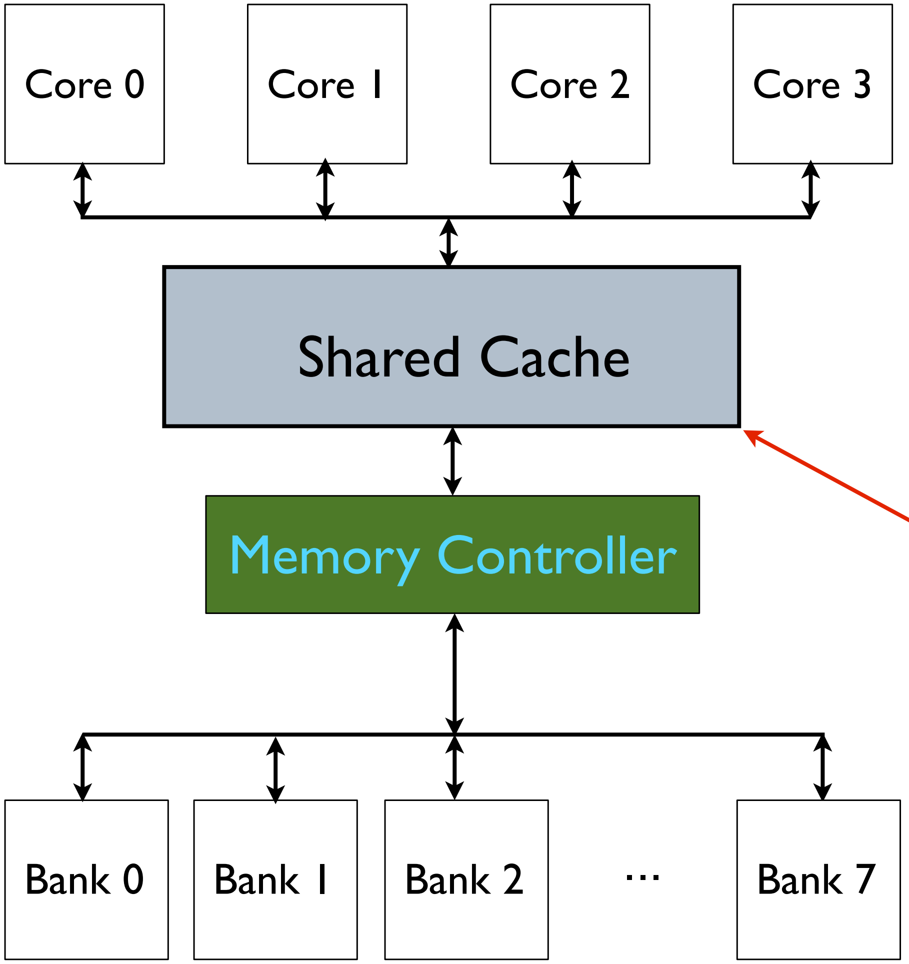


Tracking Inter-Core Interference



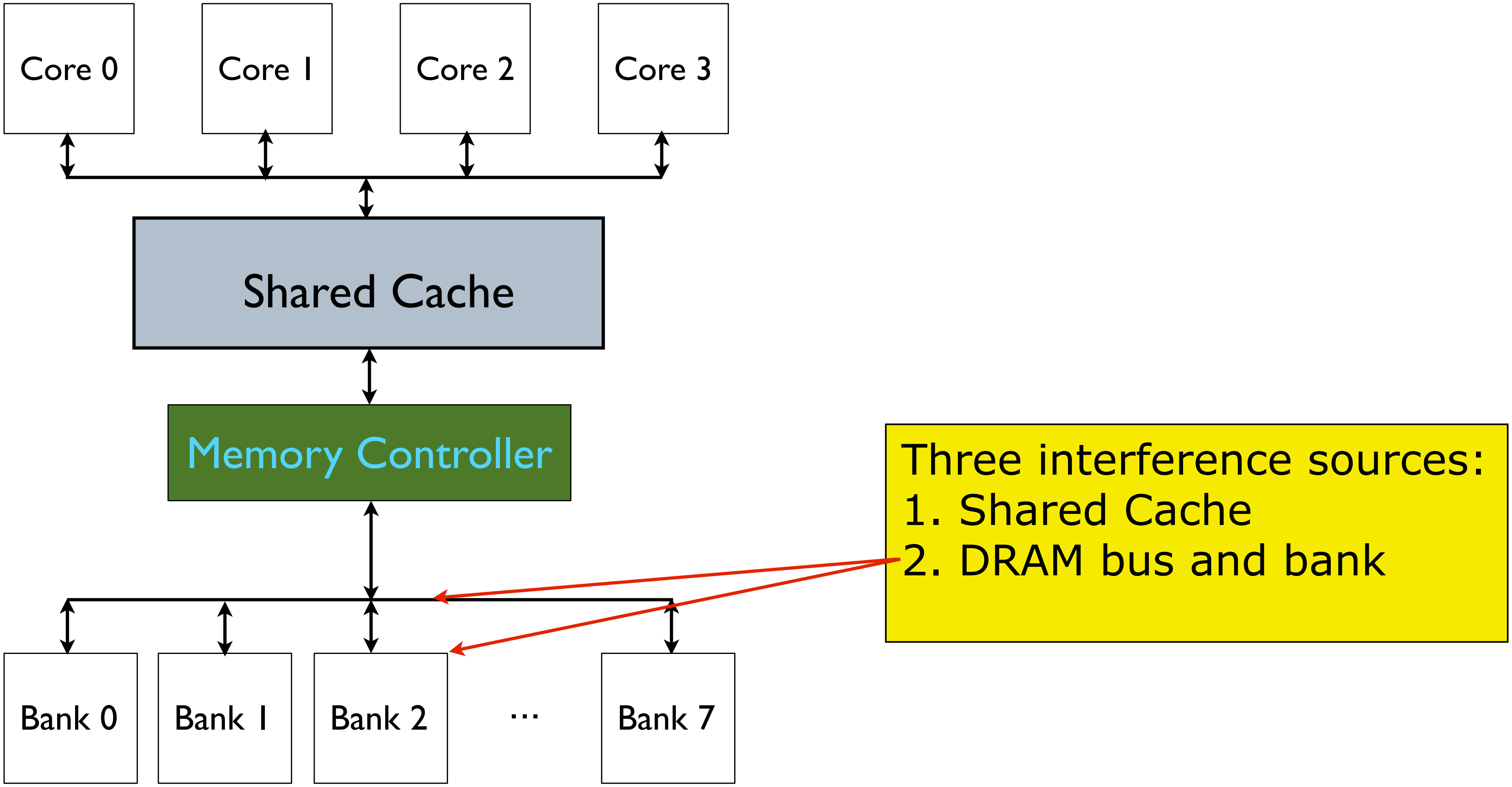
Three interference sources:

Tracking Inter-Core Interference



Three interference sources:
1. Shared Cache

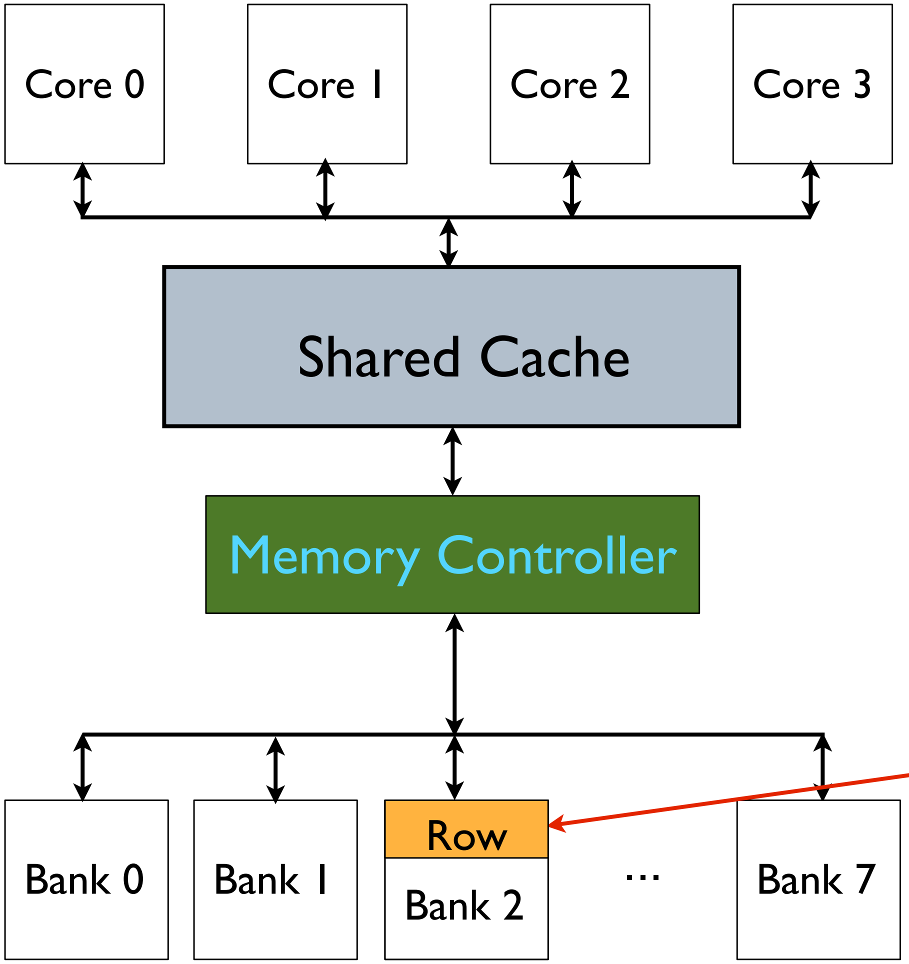
Tracking Inter-Core Interference



Three interference sources:

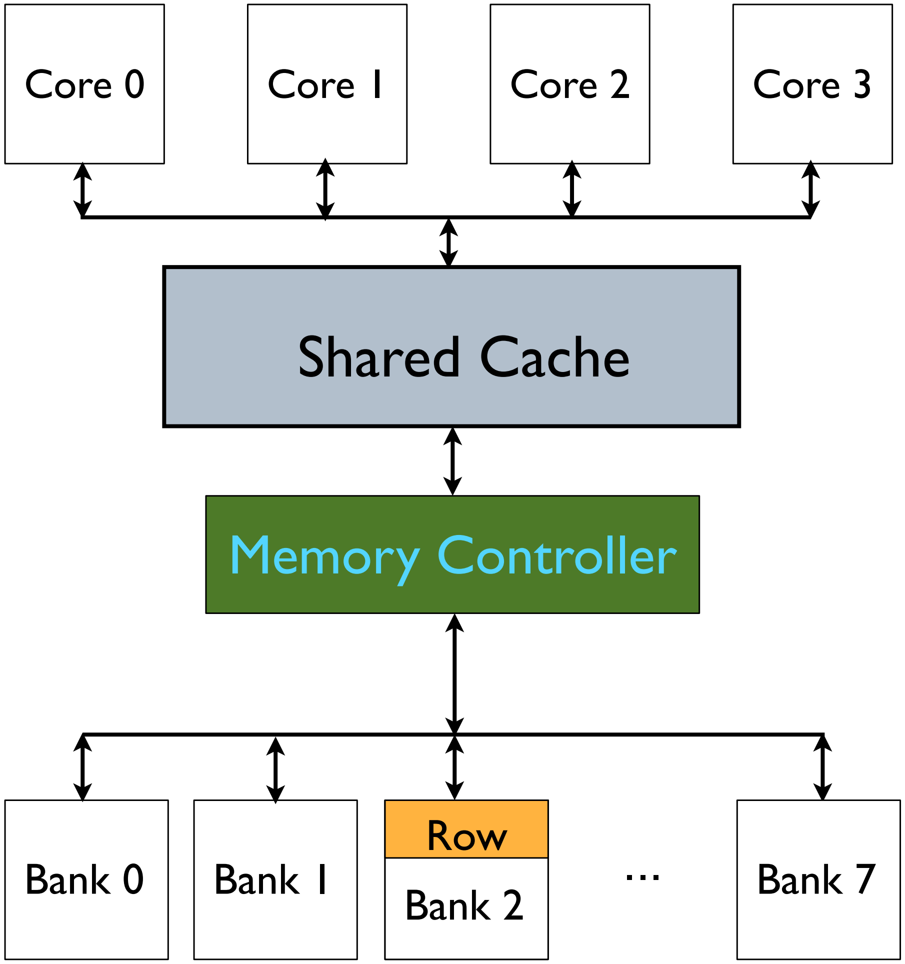
1. Shared Cache
2. DRAM bus and bank

Tracking Inter-Core Interference



Three interference sources:
1. Shared Cache
2. DRAM bus and bank
3. DRAM row-buffers

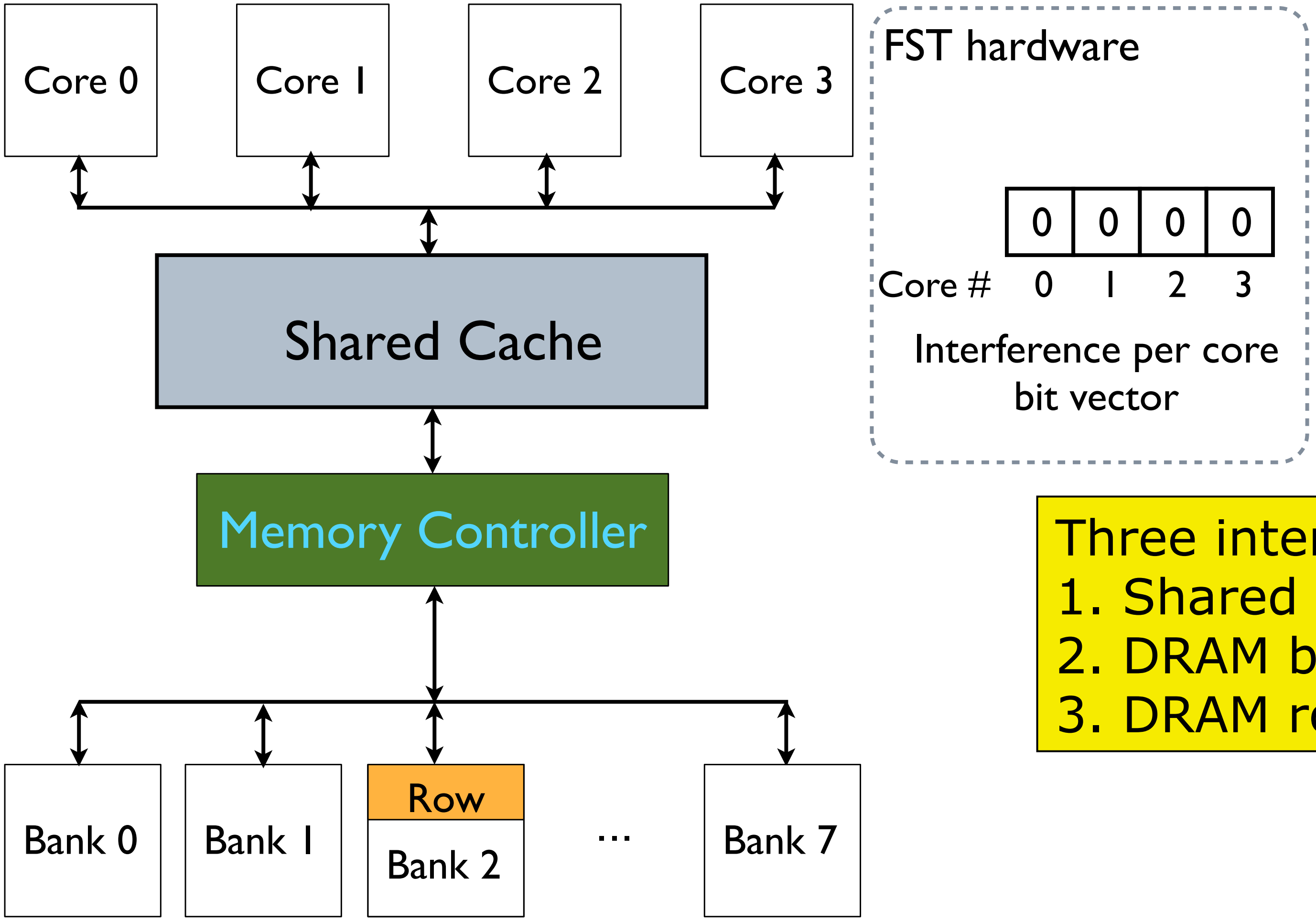
Tracking Inter-Core Interference



Three interference sources:

1. Shared Cache
2. DRAM bus and bank
3. DRAM row-buffers

Tracking Inter-Core Interference

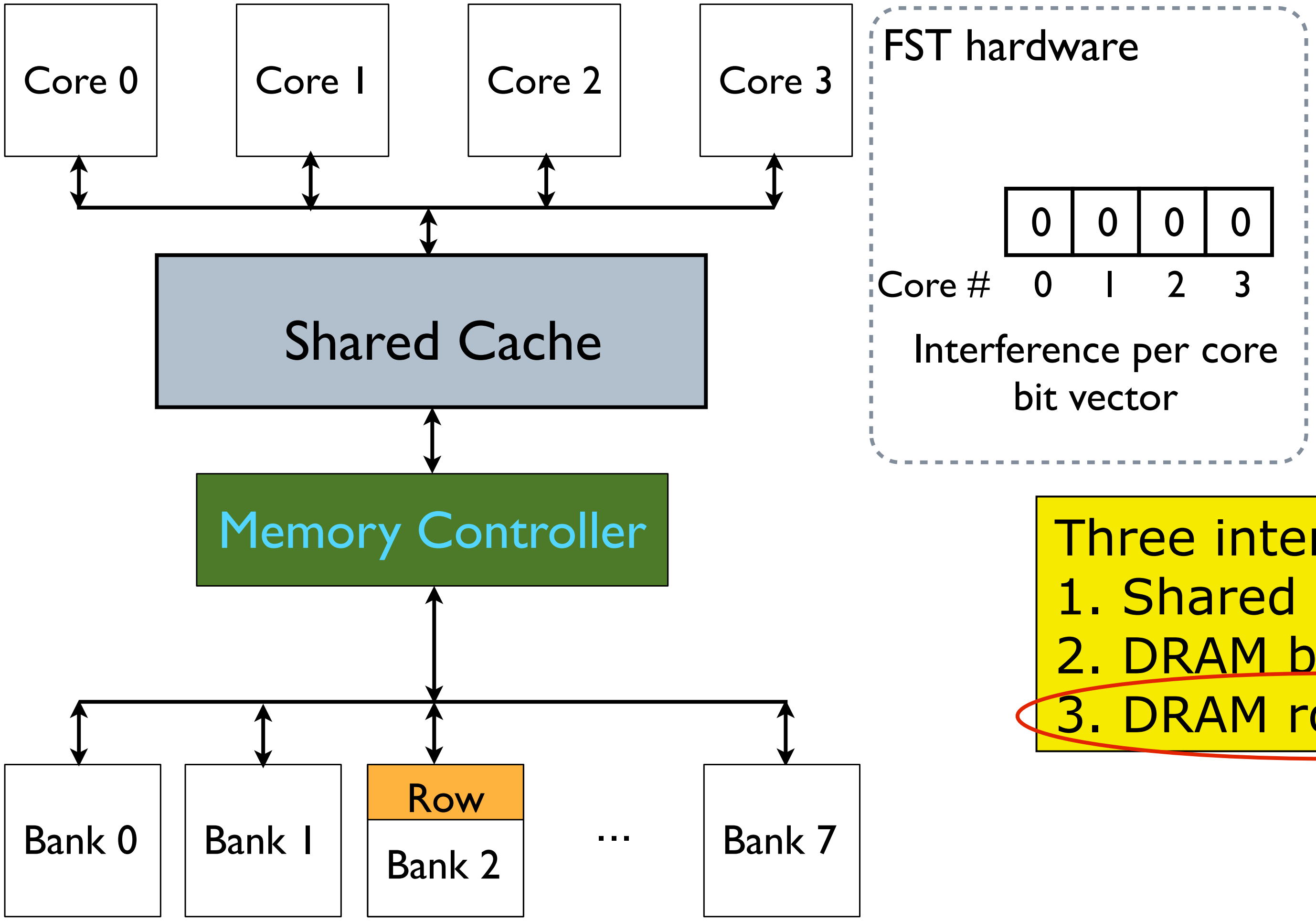


	0	0	0	0
Core #	0	1	2	3

Three interference sources:

1. Shared Cache
2. DRAM bus and bank
3. DRAM row-buffers

Tracking Inter-Core Interference



Three interference sources:

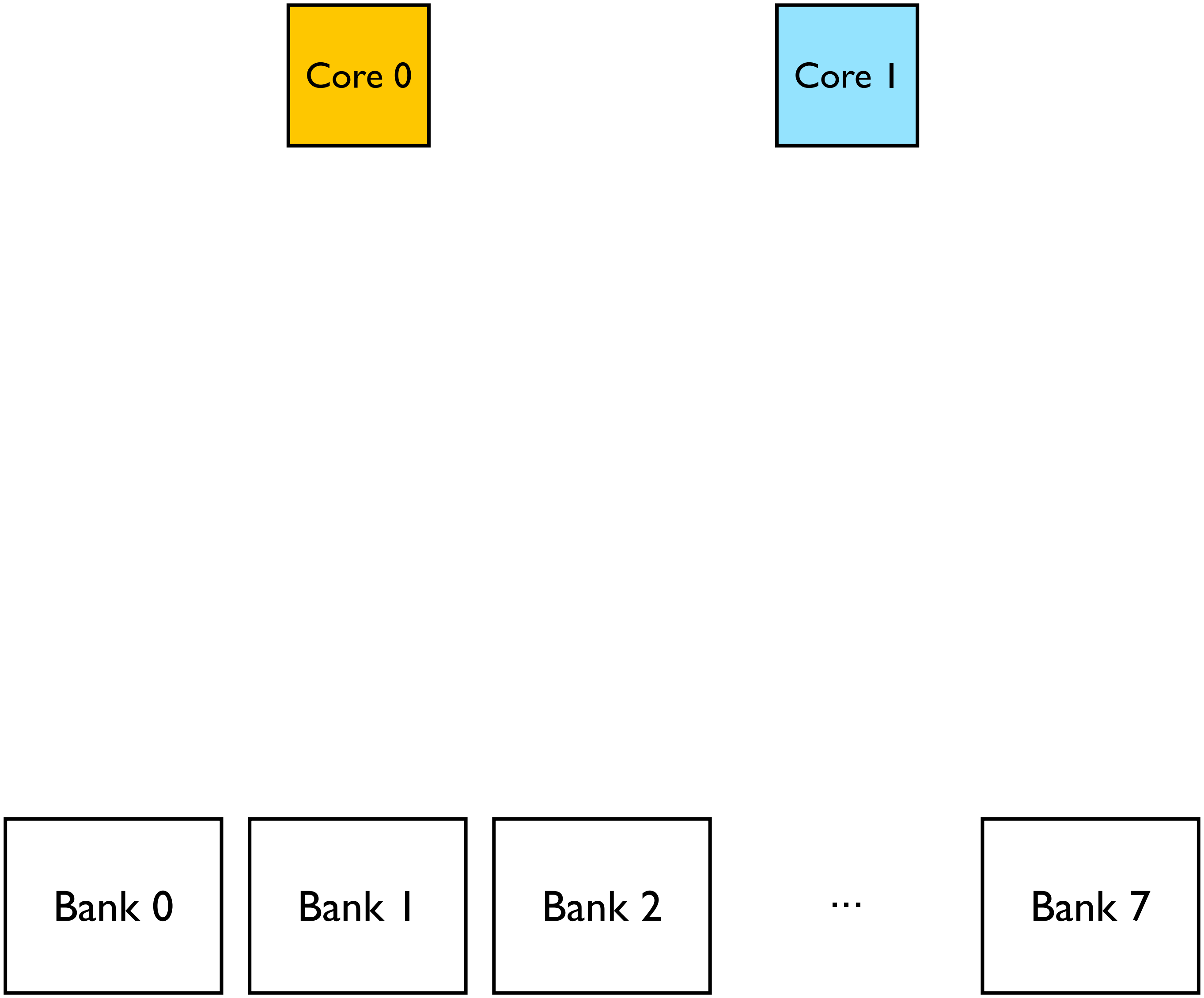
1. Shared Cache
2. DRAM bus and bank
3. DRAM row-buffers

Tracking DRAM Row-Buffer Interference

Core 0

Core 1

Tracking DRAM Row-Buffer Interference



Core 0

Core 1

Bank 0

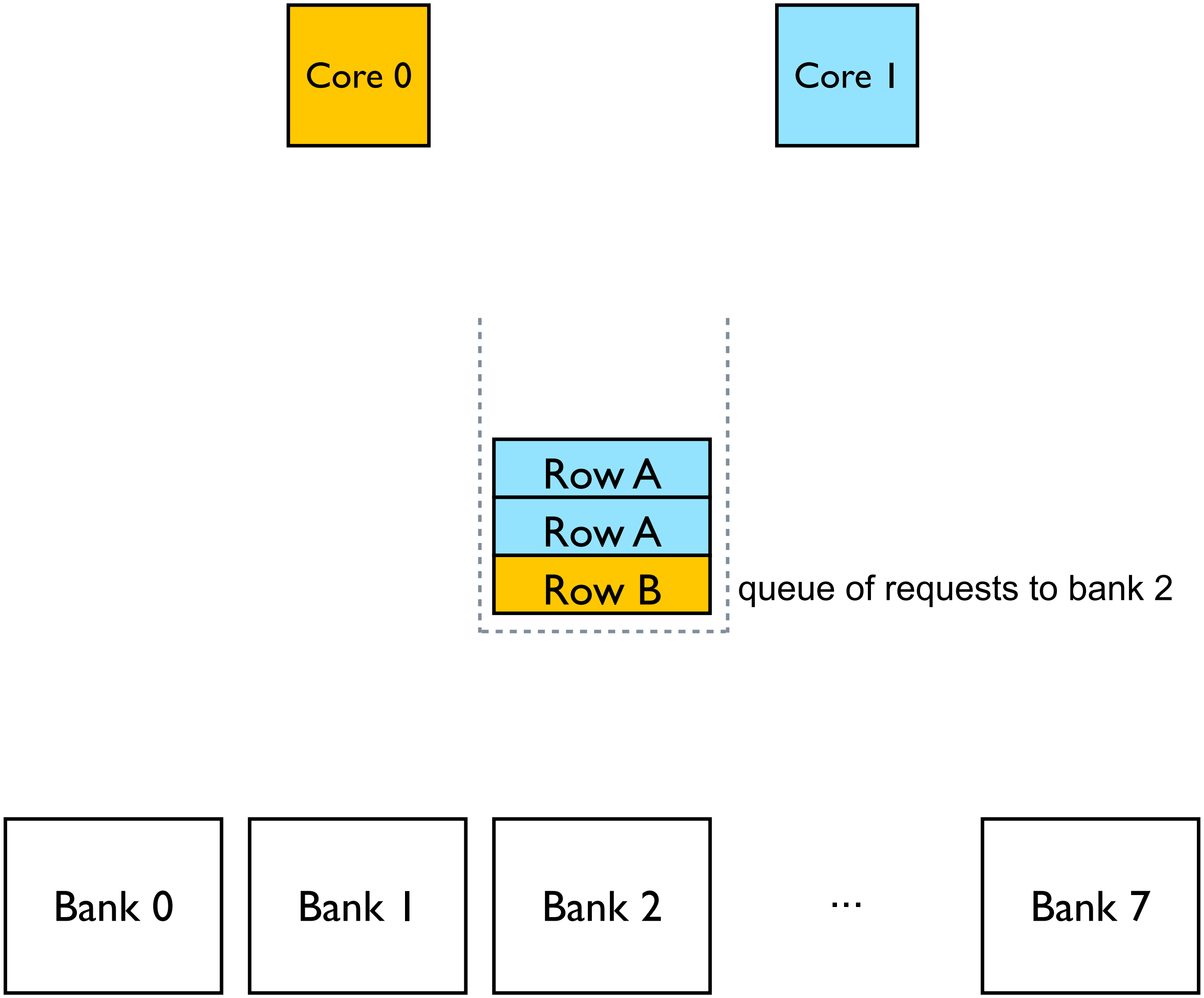
Bank 1

Bank 2

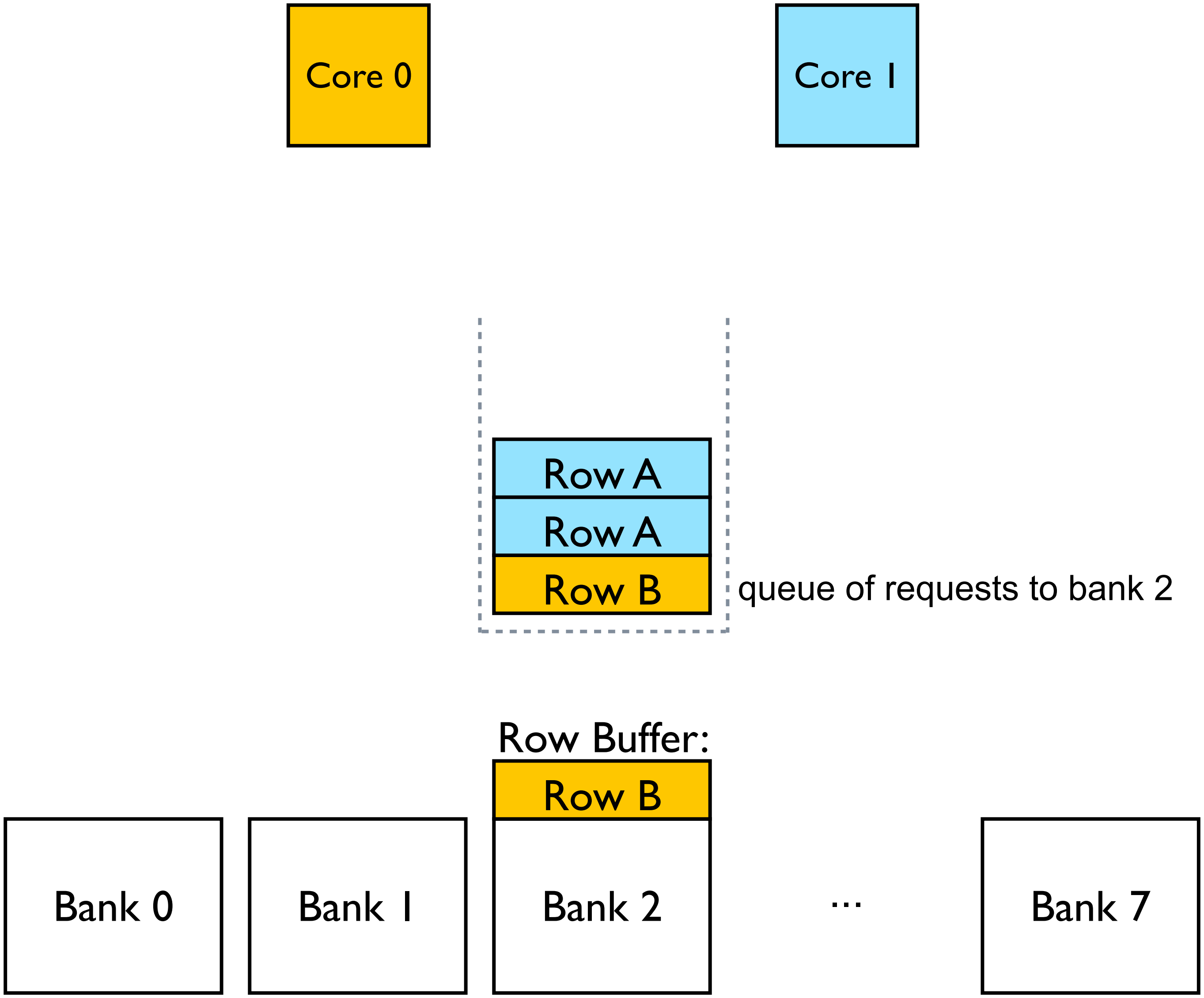
...

Bank 7

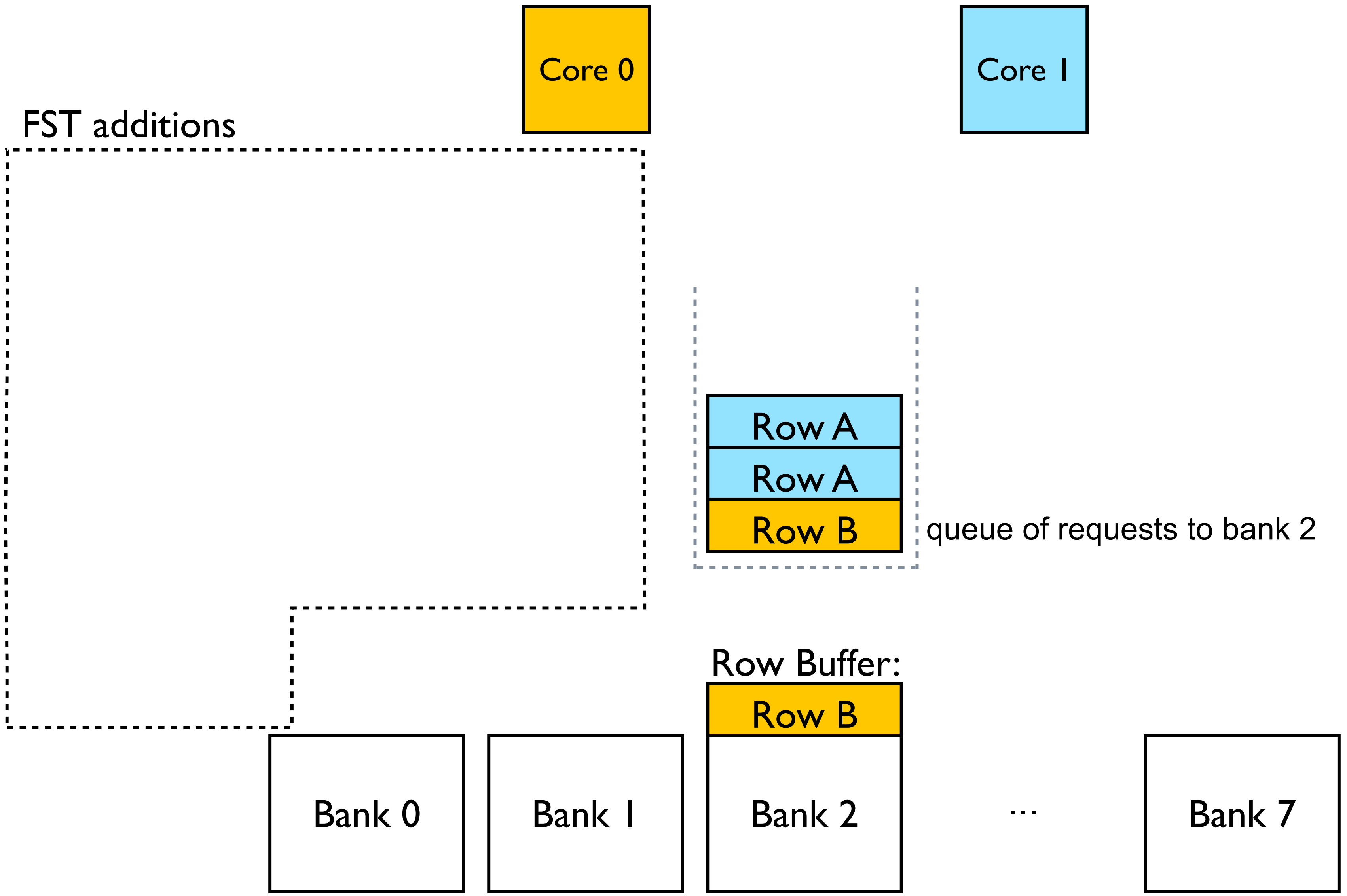
Tracking DRAM Row-Buffer Interference



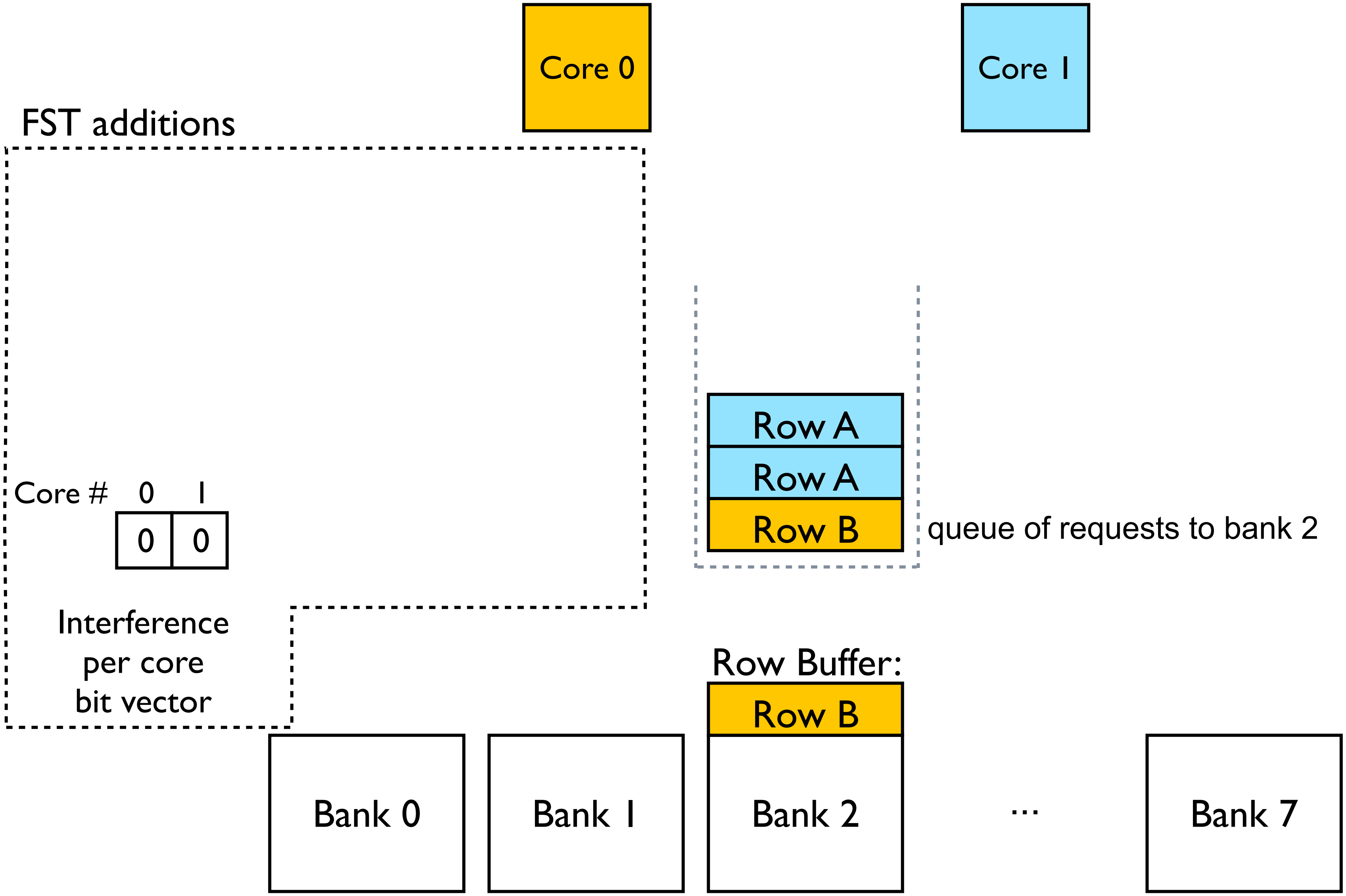
Tracking DRAM Row-Buffer Interference



Tracking DRAM Row-Buffer Interference

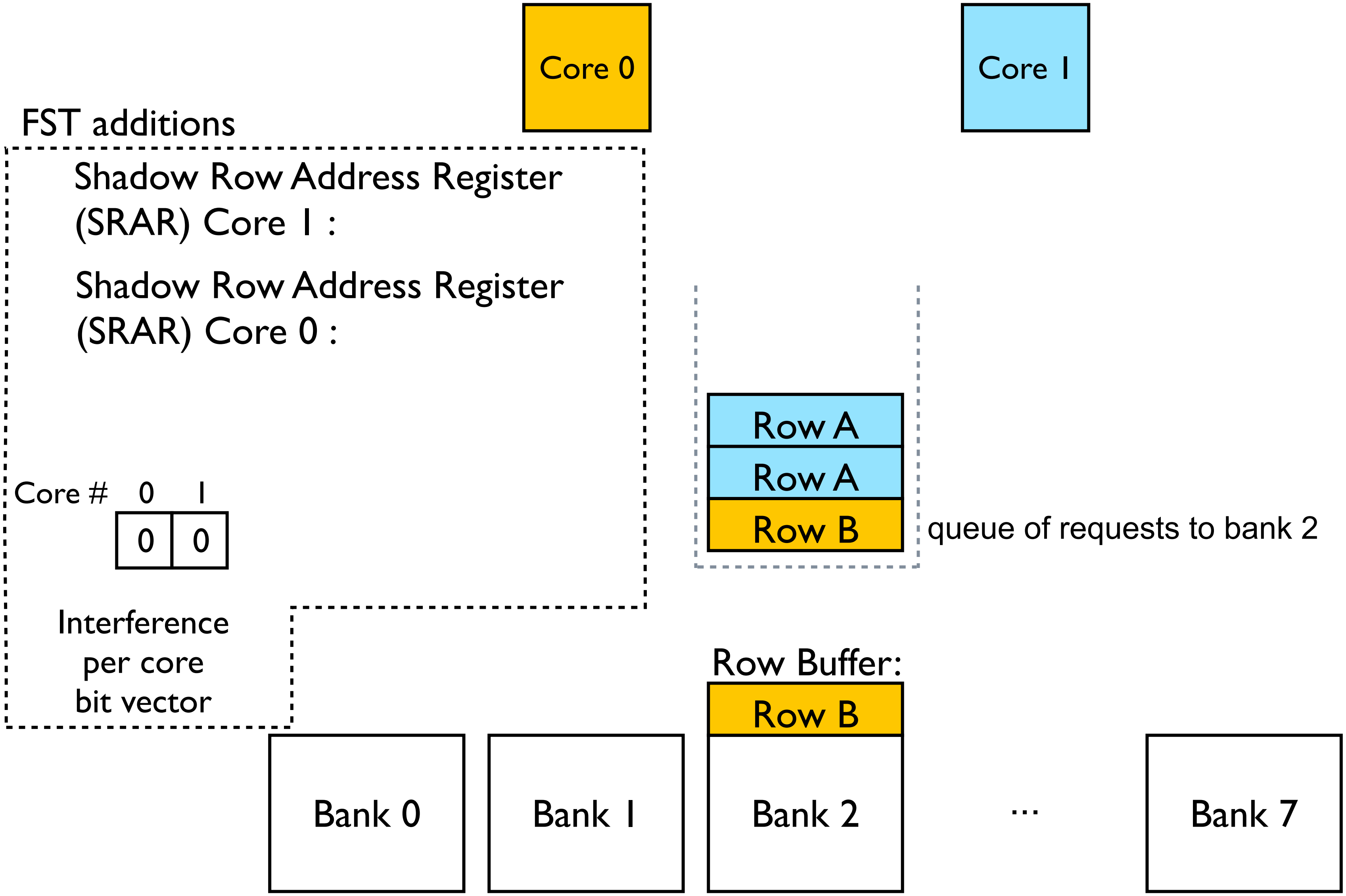


Tracking DRAM Row-Buffer Interference



Core #	0	1
	0	0

Tracking DRAM Row-Buffer Interference



Core 0

Core 1

FST additions

Shadow Row Address Register (SRAR) Core 1 :

Shadow Row Address Register (SRAR) Core 0 :

Core #	0	1
	0	0

Interference per core bit vector

Row A

Row A

Row B

queue of requests to bank 2

Row Buffer:

Row B

Bank 0

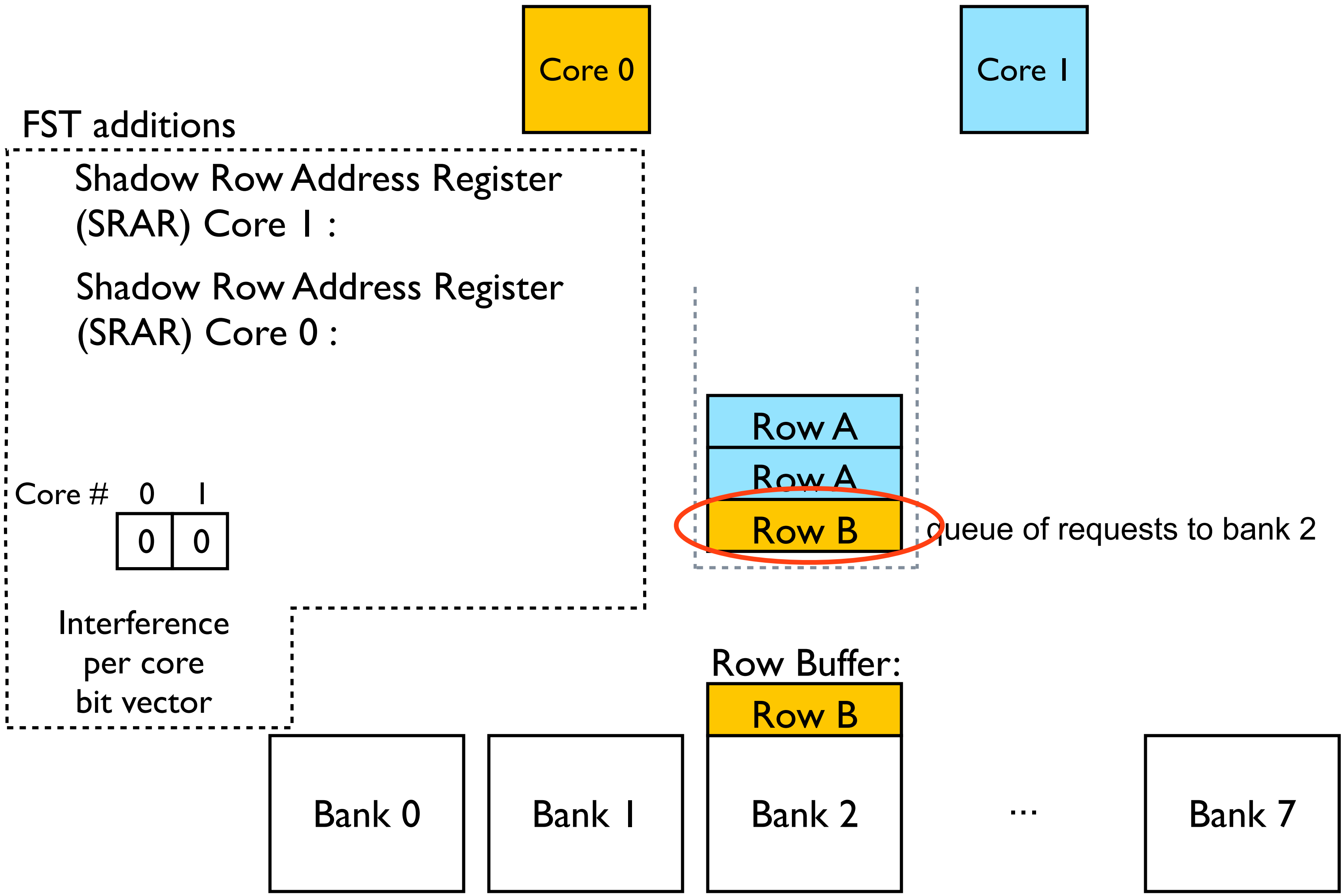
Bank 1

Bank 2

...

Bank 7

Tracking DRAM Row-Buffer Interference



Core 0

Core 1

FST additions

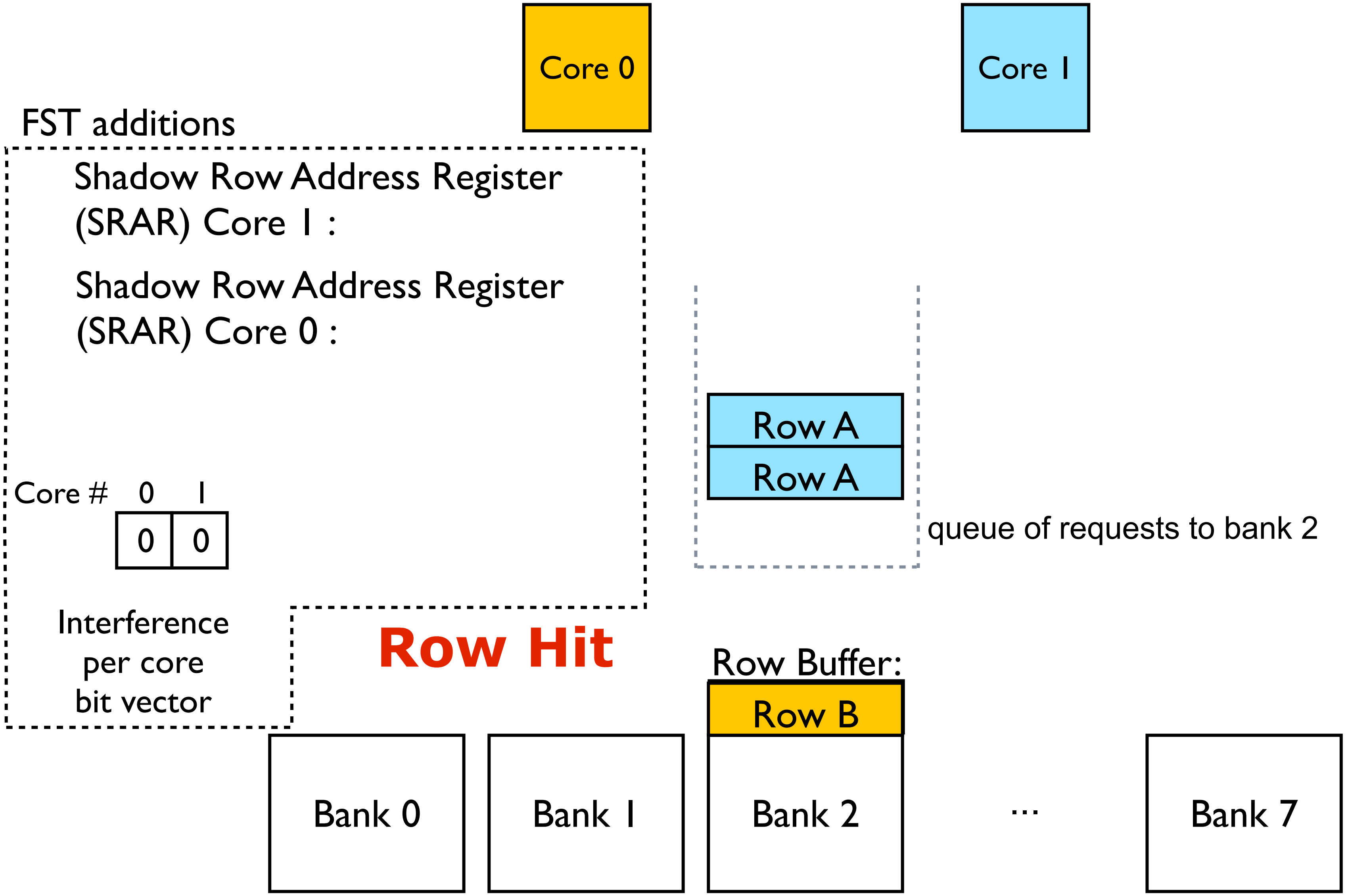
Shadow Row Address Register (SRAR) Core 1 :

Shadow Row Address Register (SRAR) Core 0 :

Core #	0	1
	0	0

Interference per core bit vector

Tracking DRAM Row-Buffer Interference



Core 0

Core 1

FST additions

Shadow Row Address Register (SRAR) Core 1 :

Shadow Row Address Register (SRAR) Core 0 :

Core #	0	1
	0	0

Interference per core bit vector

Row Hit

Row A

Row A

queue of requests to bank 2

Row Buffer:

Row B

Bank 0

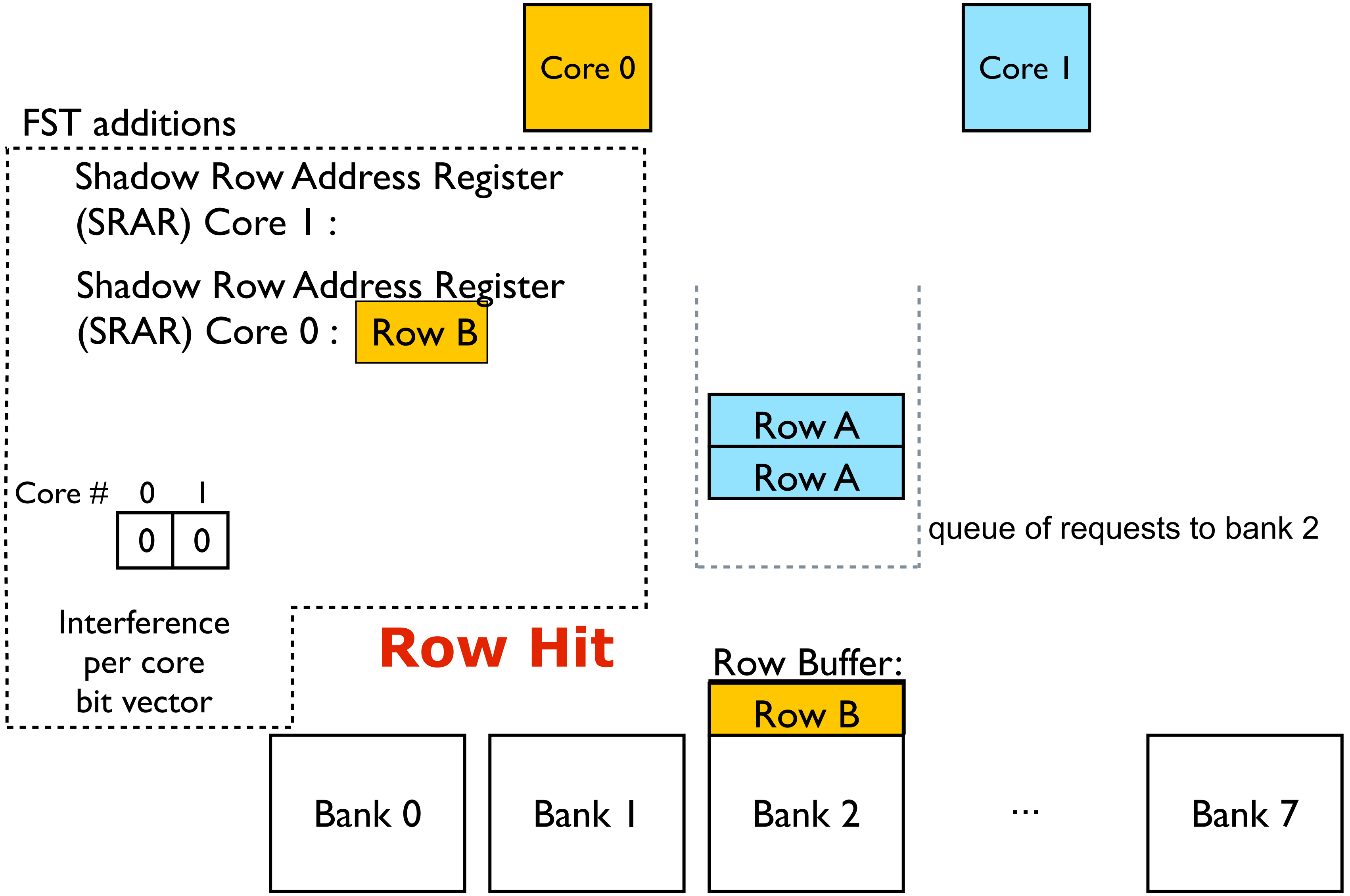
Bank 1

Bank 2

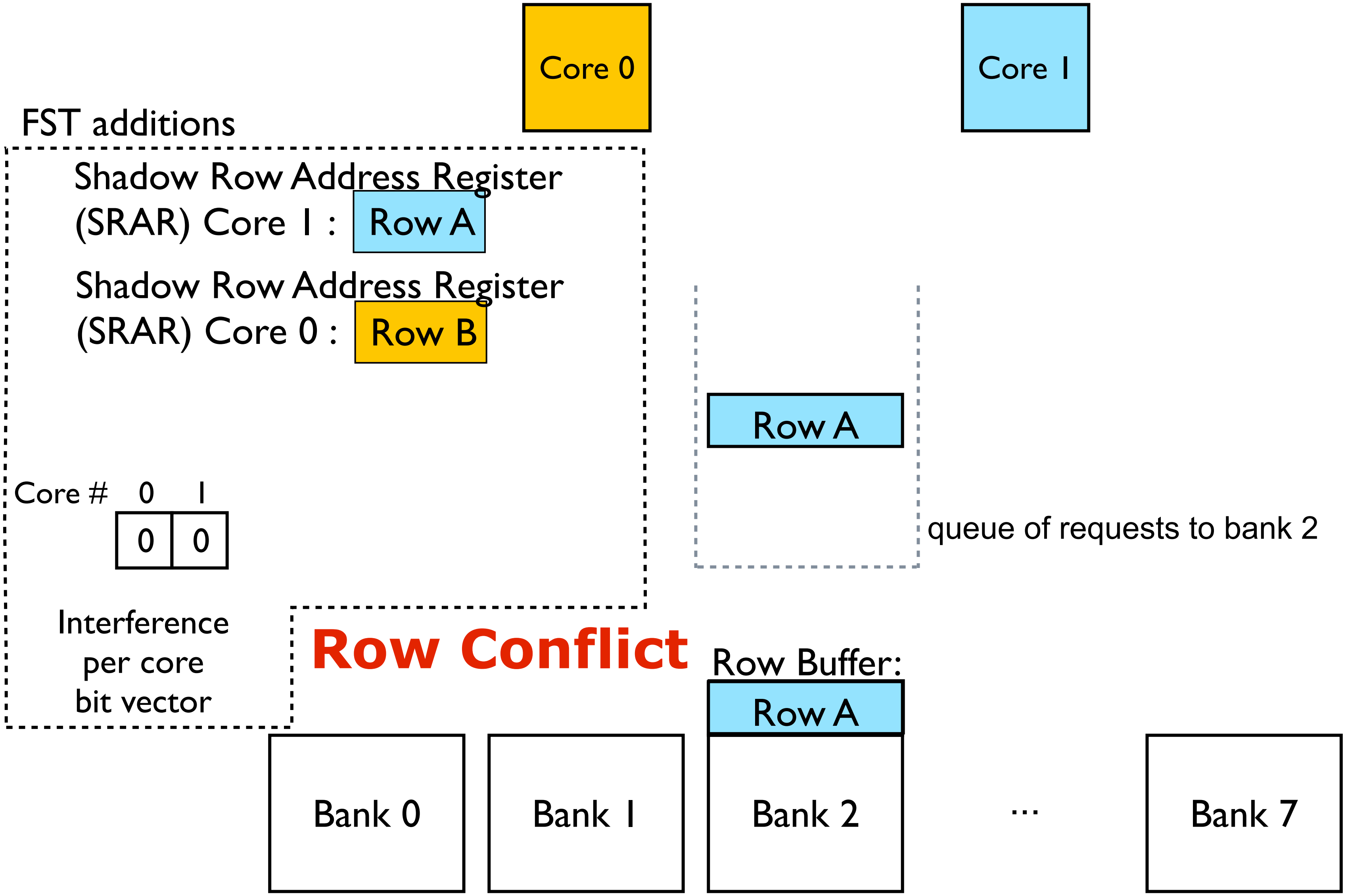
...

Bank 7

Tracking DRAM Row-Buffer Interference


Core 0
Core 1
FST additions
Shadow Row Address Register (SRAR) Core 1 :
Shadow Row Address Register (SRAR) Core 0 : Row B
Row A
Row A
queue of requests to bank 2
Core # 0 1
0 0
Interference per core bit vector
Row Hit
Row Buffer:
Row B
Bank 0
Bank 1
Bank 2
...
Bank 7

Tracking DRAM Row-Buffer Interference



Core 0

Core 1

FST additions

Shadow Row Address Register (SRAR) Core 1 : Row A

Shadow Row Address Register (SRAR) Core 0 : Row B

Core # 0 1

0	1
0	0

Interference
per core
bit vector

Row A

queue of requests to bank 2

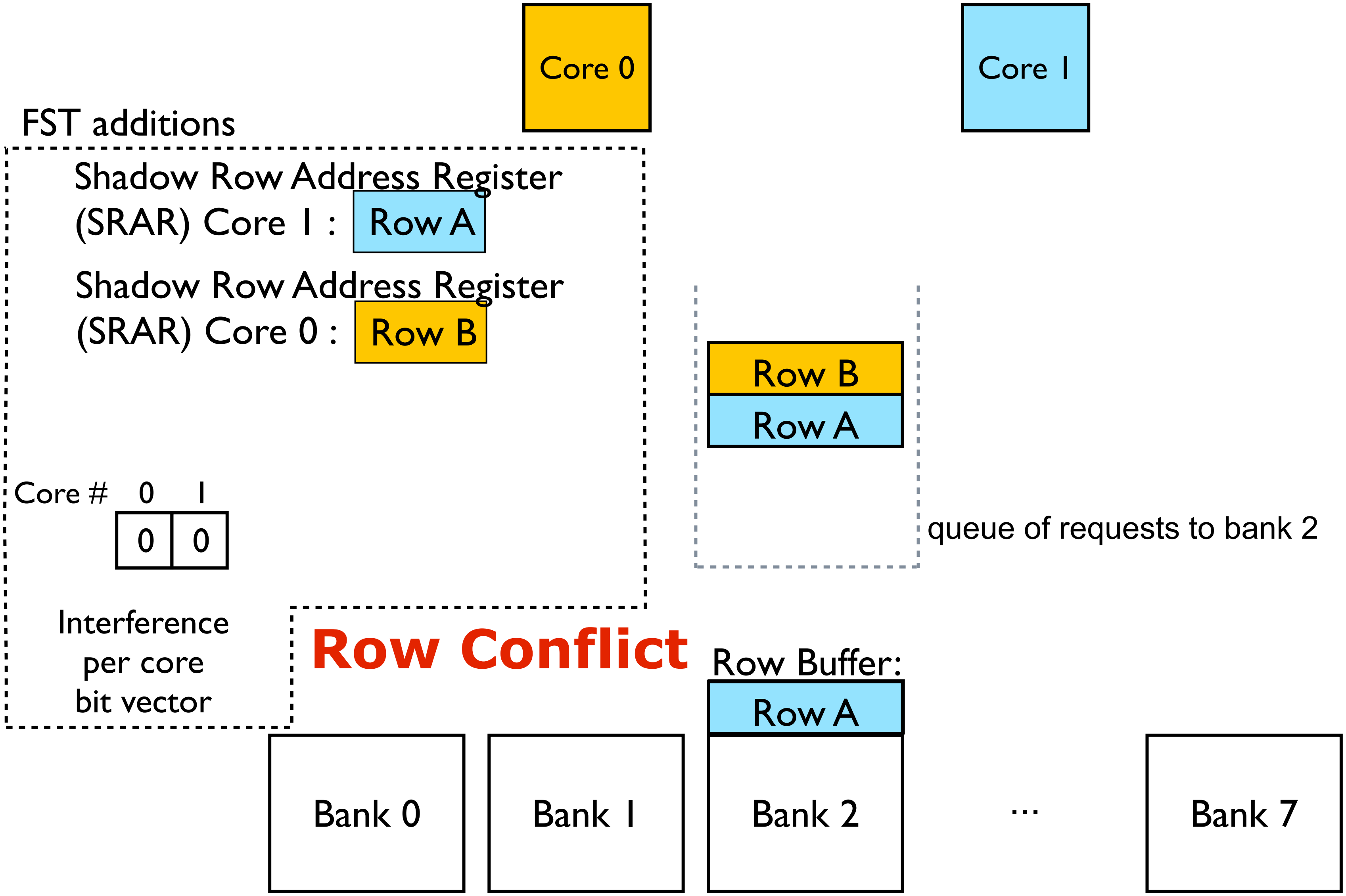
Row Conflict

Row Buffer:

Row A

Bank 0 Bank 1 Bank 2 … Bank 7

Tracking DRAM Row-Buffer Interference



Core 0

Core 1

FST additions

Shadow Row Address Register (SRAR) Core 1 : Row A

Shadow Row Address Register (SRAR) Core 0 : Row B

Core #	0	1
	0	0

Interference per core bit vector

Row B

Row A

queue of requests to bank 2

Row Conflict

Row Buffer: Row A

Bank 0

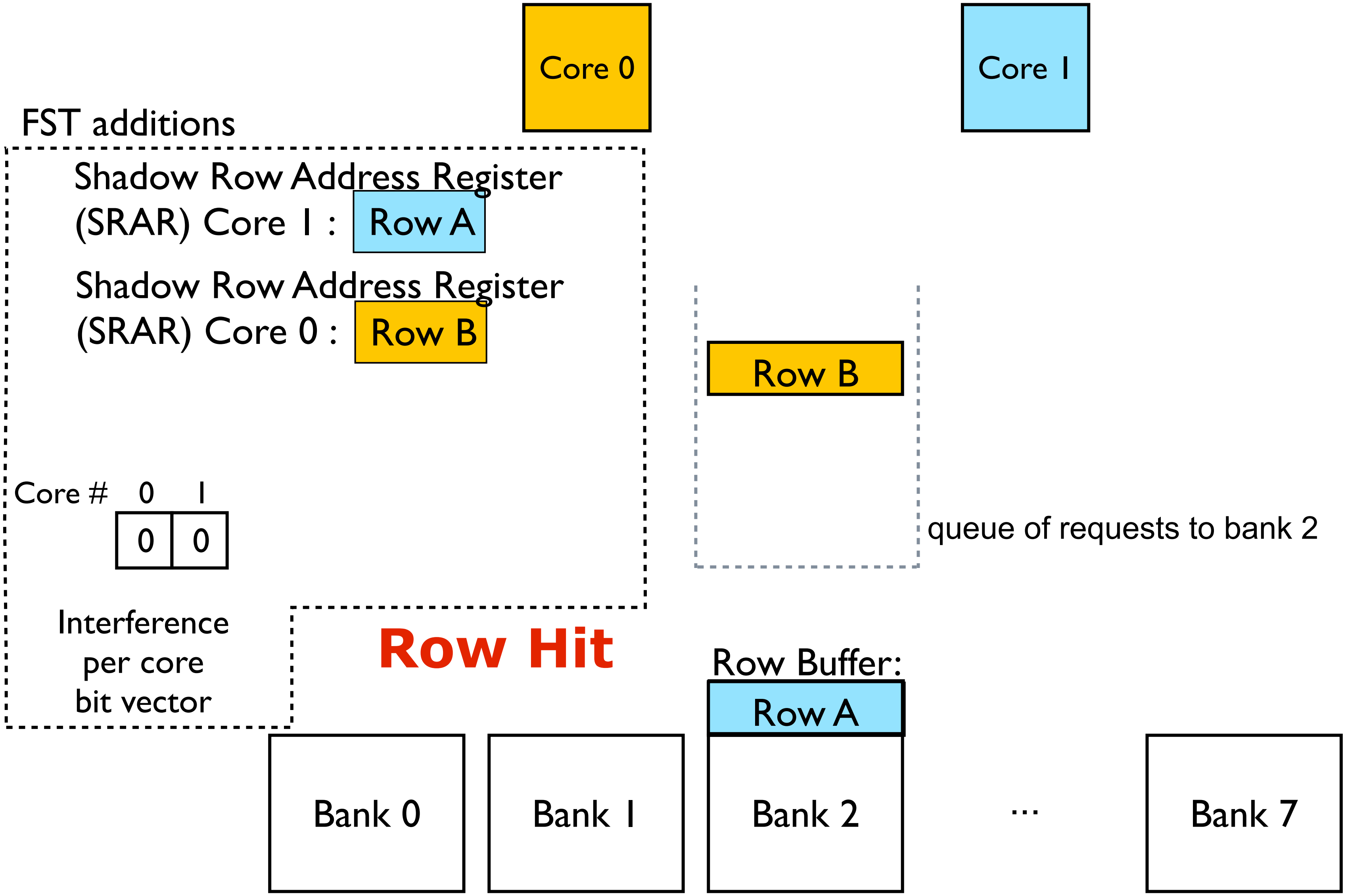
Bank 1

Bank 2

…

Bank 7

Tracking DRAM Row-Buffer Interference



Core 0

Core 1

FST additions

Shadow Row Address Register (SRAR) Core 1 : Row A

Shadow Row Address Register (SRAR) Core 0 : Row B

Core #	0	1
	0	0

Interference per core bit vector

Row B

queue of requests to bank 2

Row Hit

Row Buffer: Row A

Bank 0

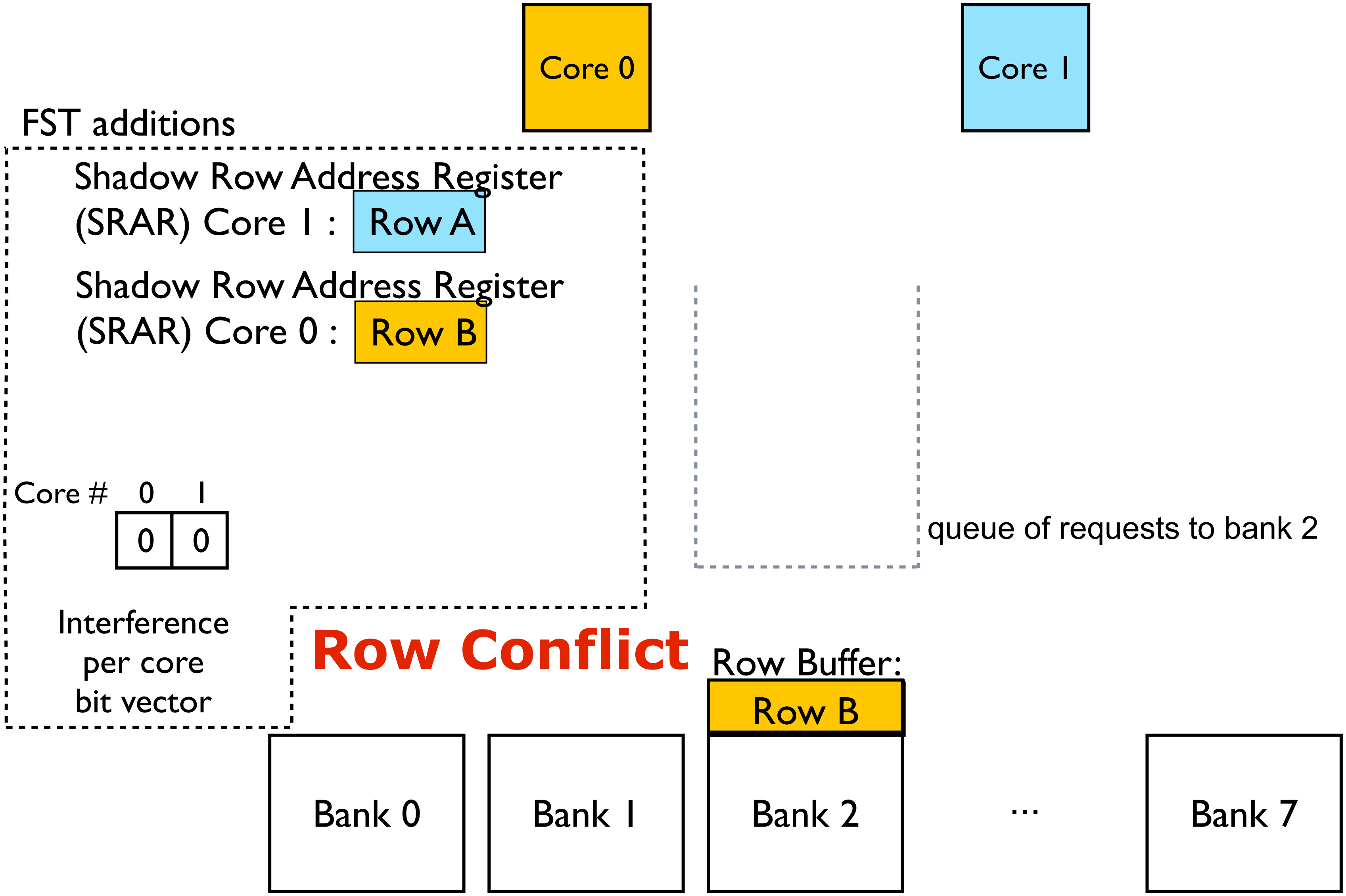
Bank 1

Bank 2

…

Bank 7

Tracking DRAM Row-Buffer Interference



Core 0

Core 1

FST additions

Shadow Row Address Register (SRAR) Core 1 : Row A

Shadow Row Address Register (SRAR) Core 0 : Row B

Core #	0	1
	0	0

Interference per core bit vector

queue of requests to bank 2

Row Conflict

Row Buffer: Row B

Bank 0

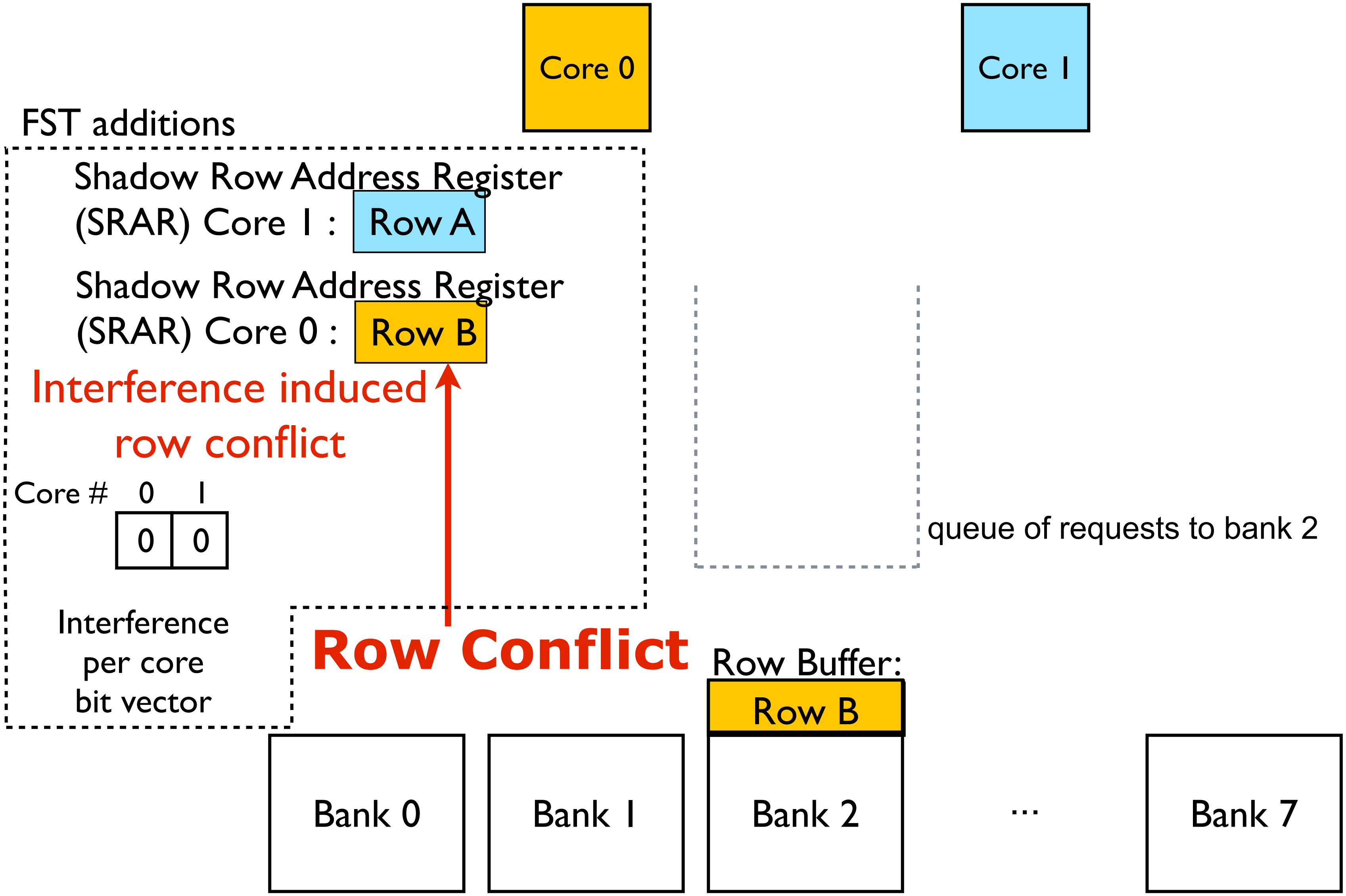
Bank 1

Bank 2

…

Bank 7

Tracking DRAM Row-Buffer Interference



Core 0

Core 1

FST additions

Shadow Row Address Register (SRAR) Core 1 : Row A

Shadow Row Address Register (SRAR) Core 0 : Row B

Interference induced row conflict

Core #	0	1
	0	0

Interference per core bit vector

Row Conflict

queue of requests to bank 2

Row Buffer: Row B

Bank 0

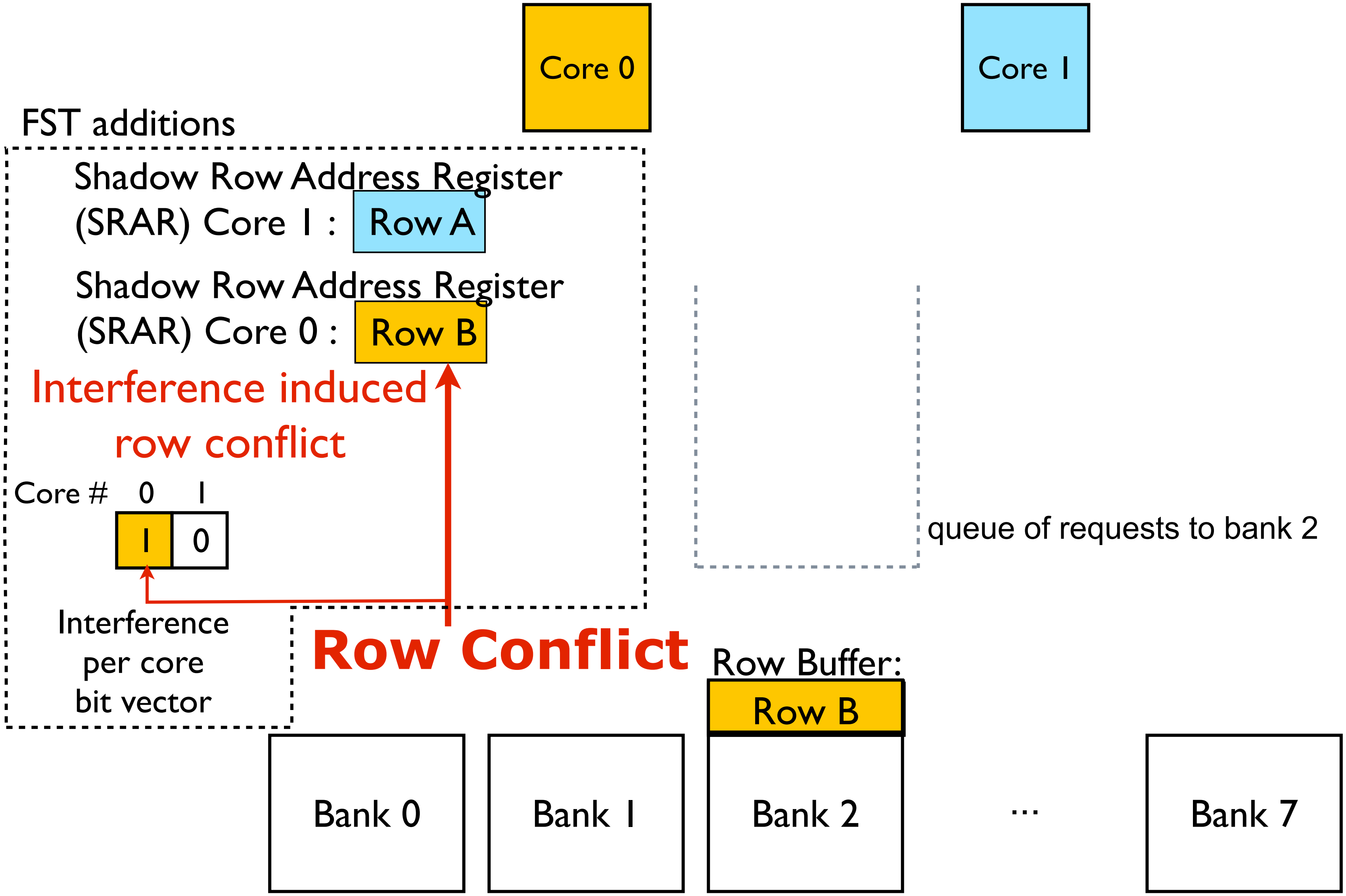
Bank 1

Bank 2

…

Bank 7

Tracking DRAM Row-Buffer Interference



Core 0

Core 1

FST additions

Shadow Row Address Register (SRAR) Core 1 : Row A

Shadow Row Address Register (SRAR) Core 0 : Row B

Interference induced row conflict

Core #	0	1
	1	0

Interference per core bit vector

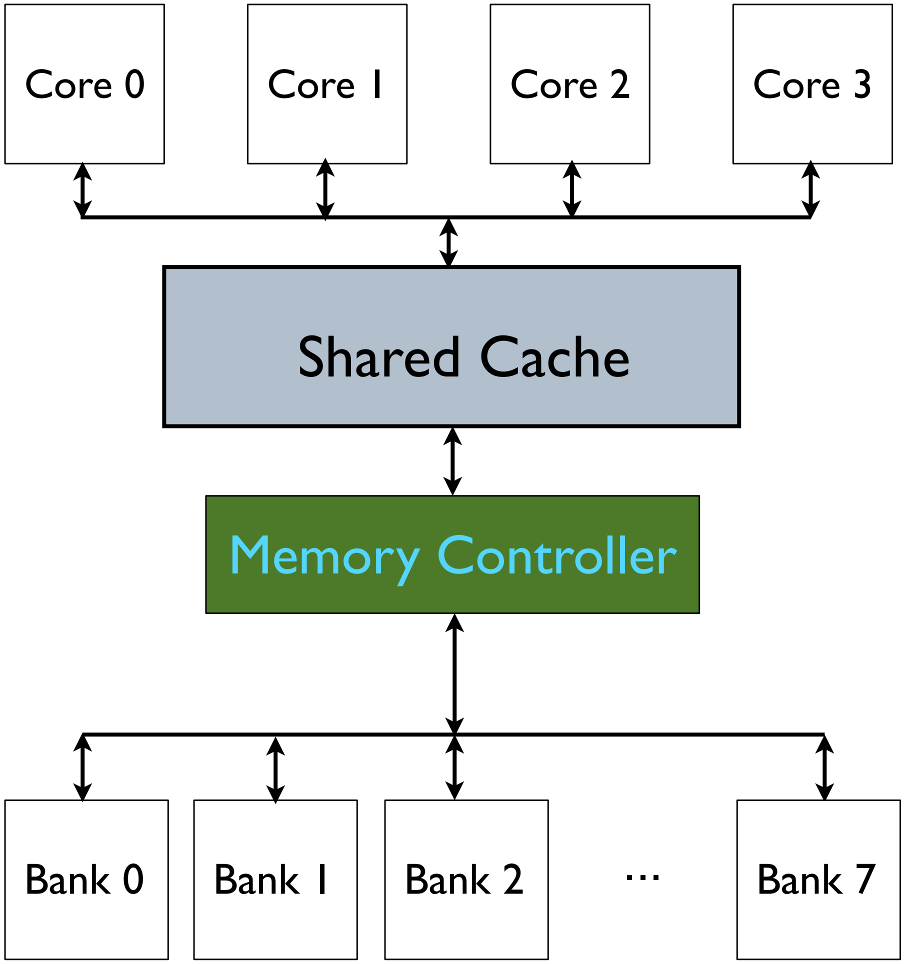
queue of requests to bank 2

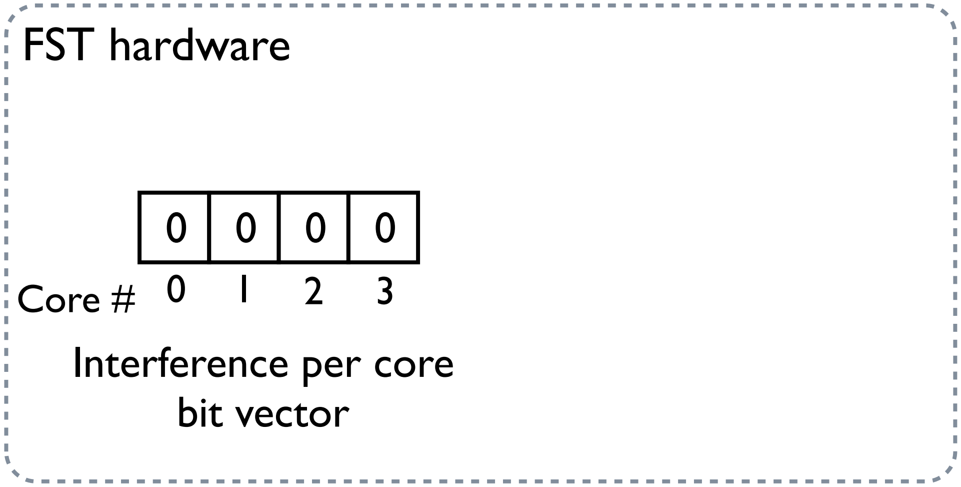
Row Conflict

Row Buffer: Row B

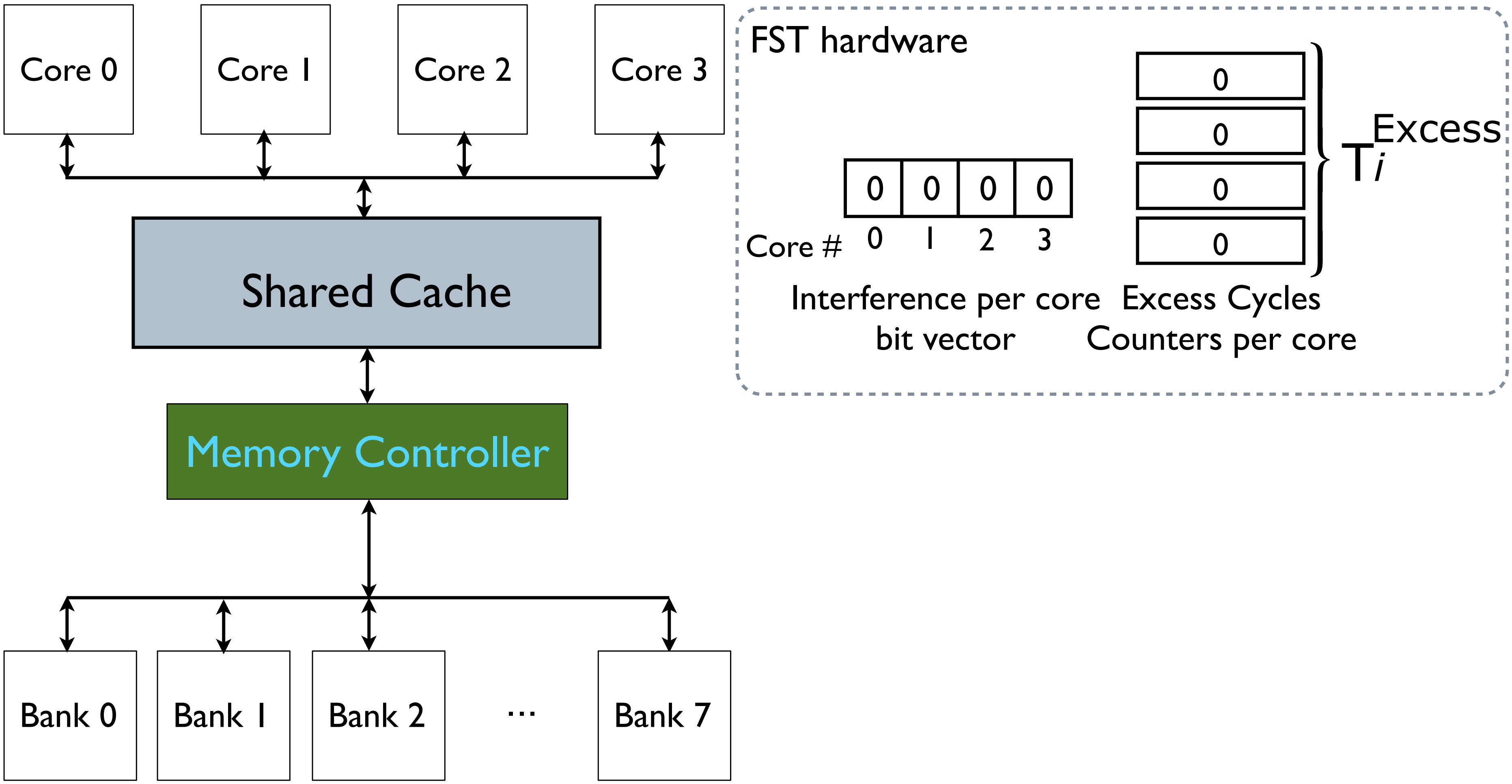
Bank 0 Bank 1 Bank 2 … Bank 7

Tracking Inter-Core Interference





Tracking Inter-Core Interference



Core 0

Core 1

Core 2

Core 3

Shared Cache

Memory Controller

Bank 0

Bank 1

Bank 2

…

Bank 7

FST hardware

Core #	0	1	2	3
	0	0	0	0

Interference per core bit vector

0

0

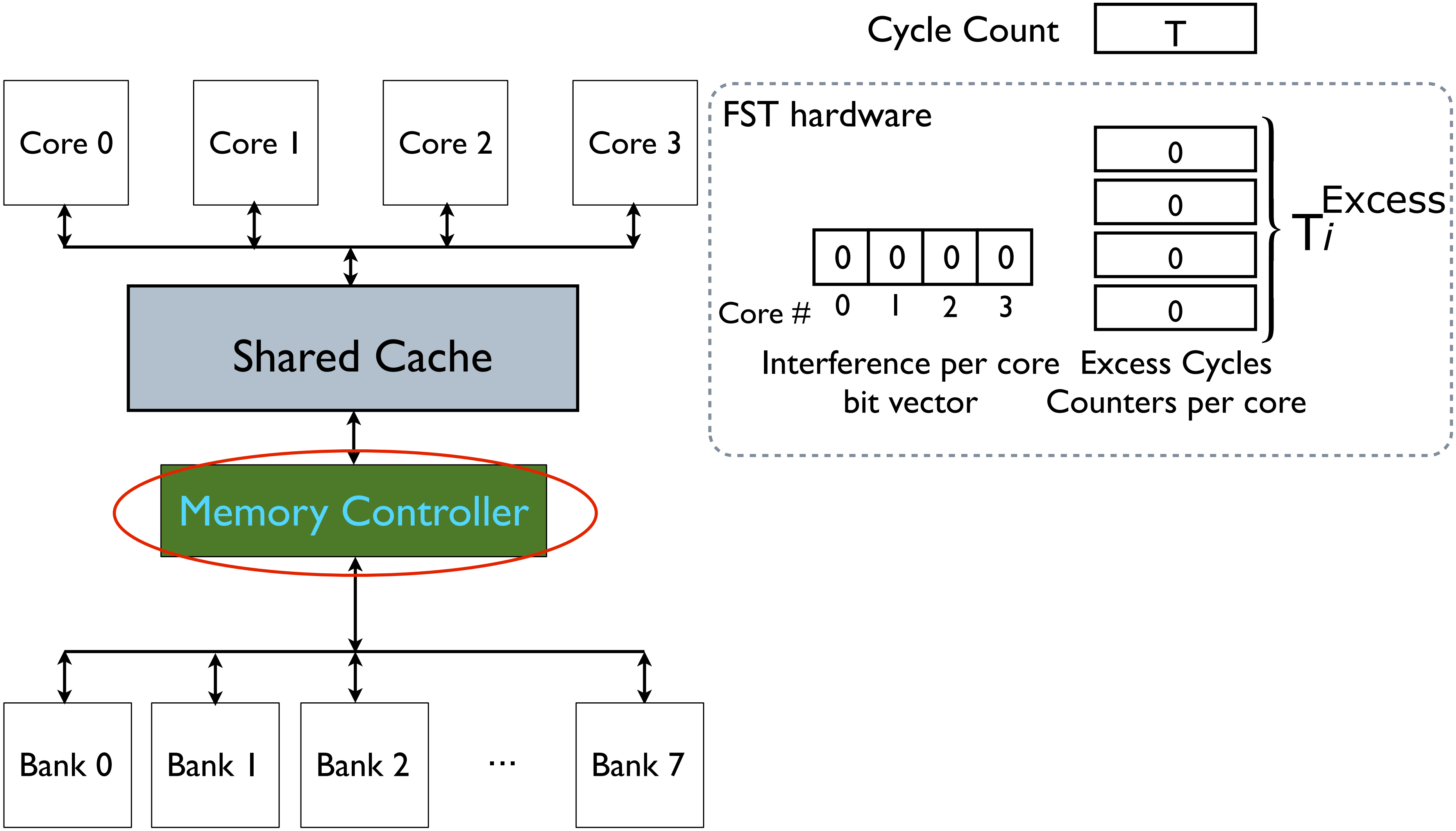
0

0

T_i^{Excess}

Excess Cycles Counters per core

Tracking Inter-Core Interference



Cycle Count T

Core 0

Core 1

Core 2

Core 3

Shared Cache

Memory Controller

Bank 0

Bank 1

Bank 2

…

Bank 7

FST hardware

Core #	0	1	2	3
	0	0	0	0

Interference per core bit vector

0

0

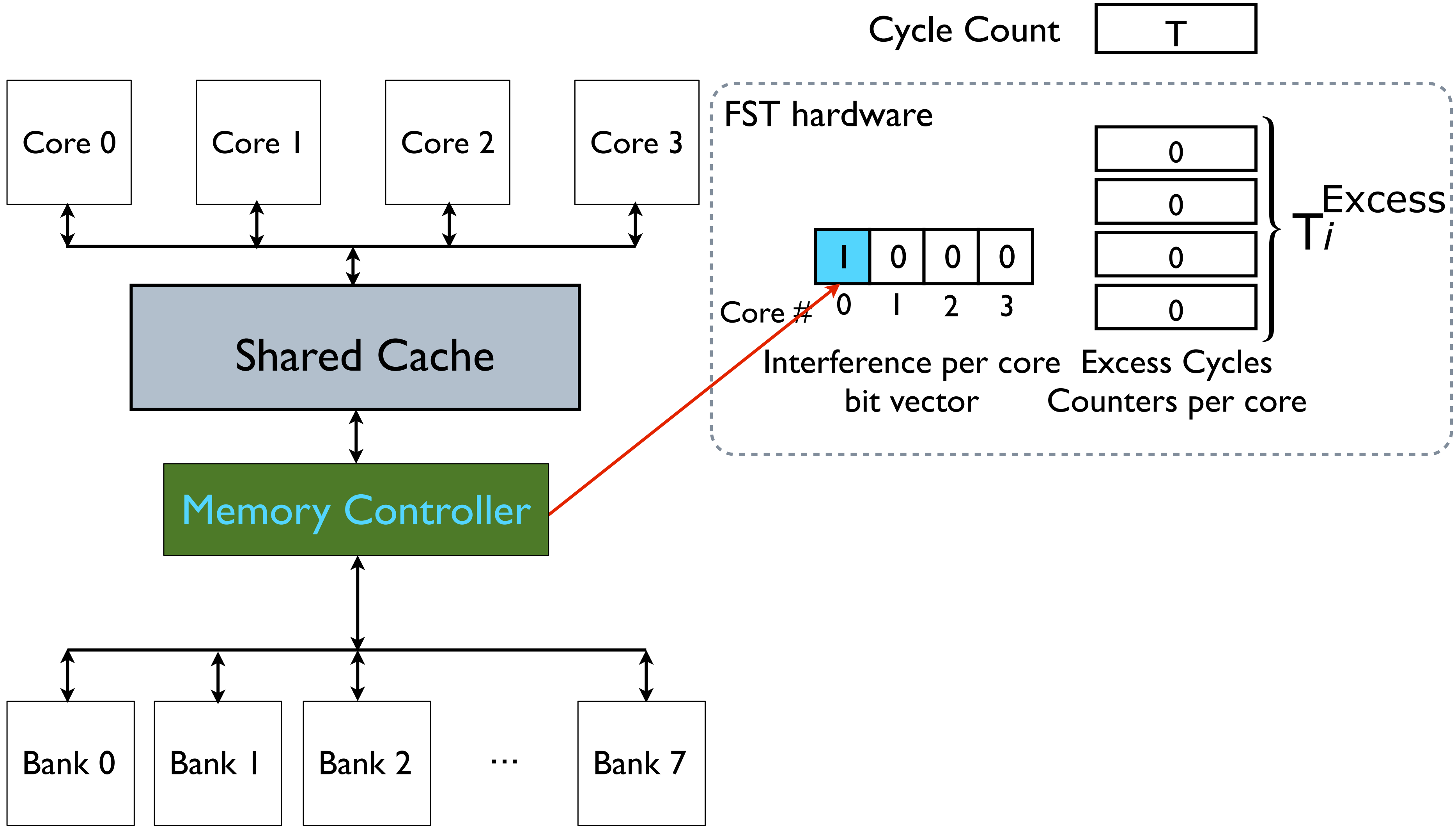
0

0

T_i^{Excess}

Excess Cycles Counters per core

Tracking Inter-Core Interference



Cycle Count T

Core 0

Core 1

Core 2

Core 3

Shared Cache

Memory Controller

Bank 0

Bank 1

Bank 2

…

Bank 7

FST hardware

1 0 0 0

Core # 0 1 2 3

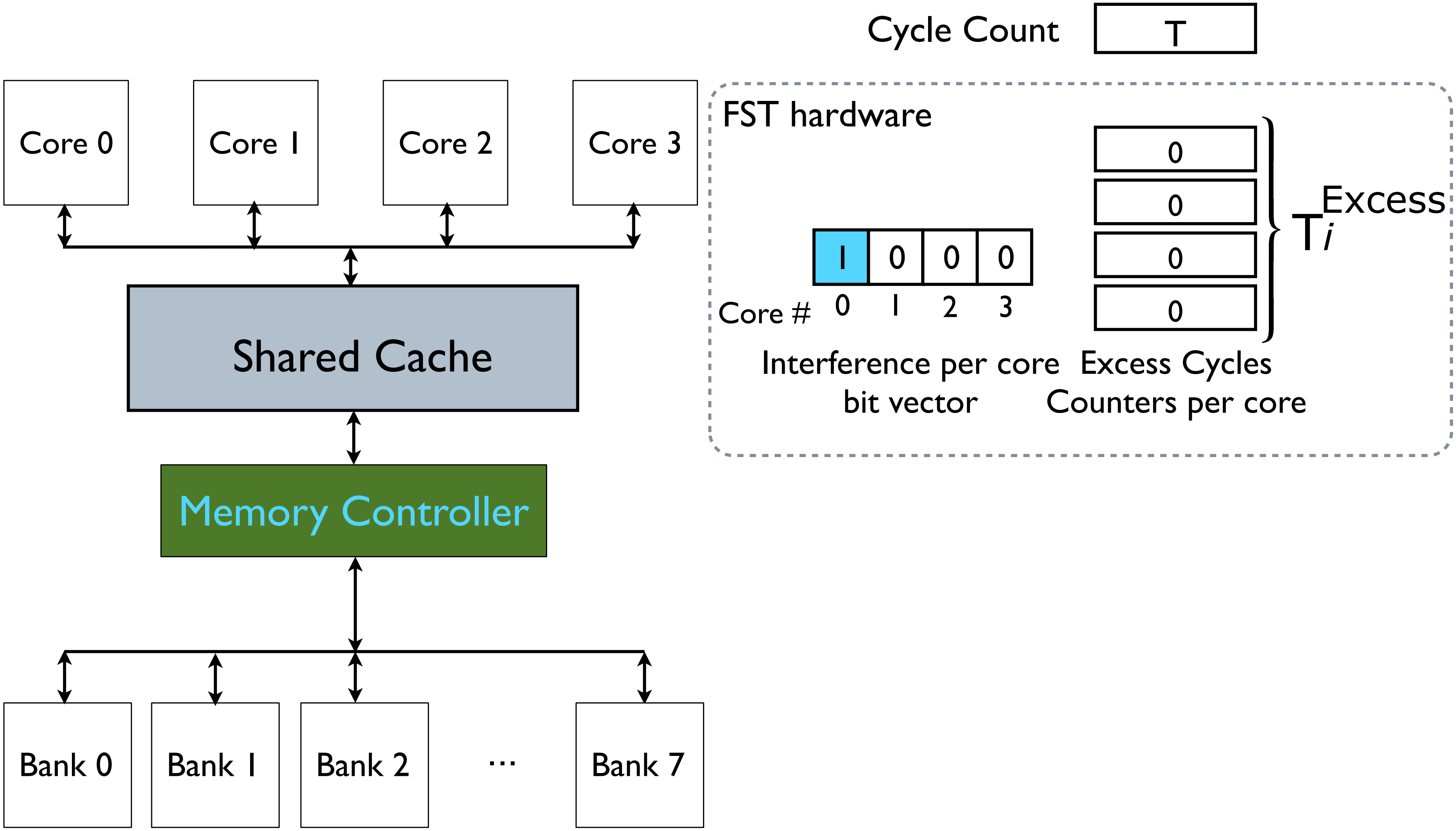
Interference per core bit vector

0
0
0
0

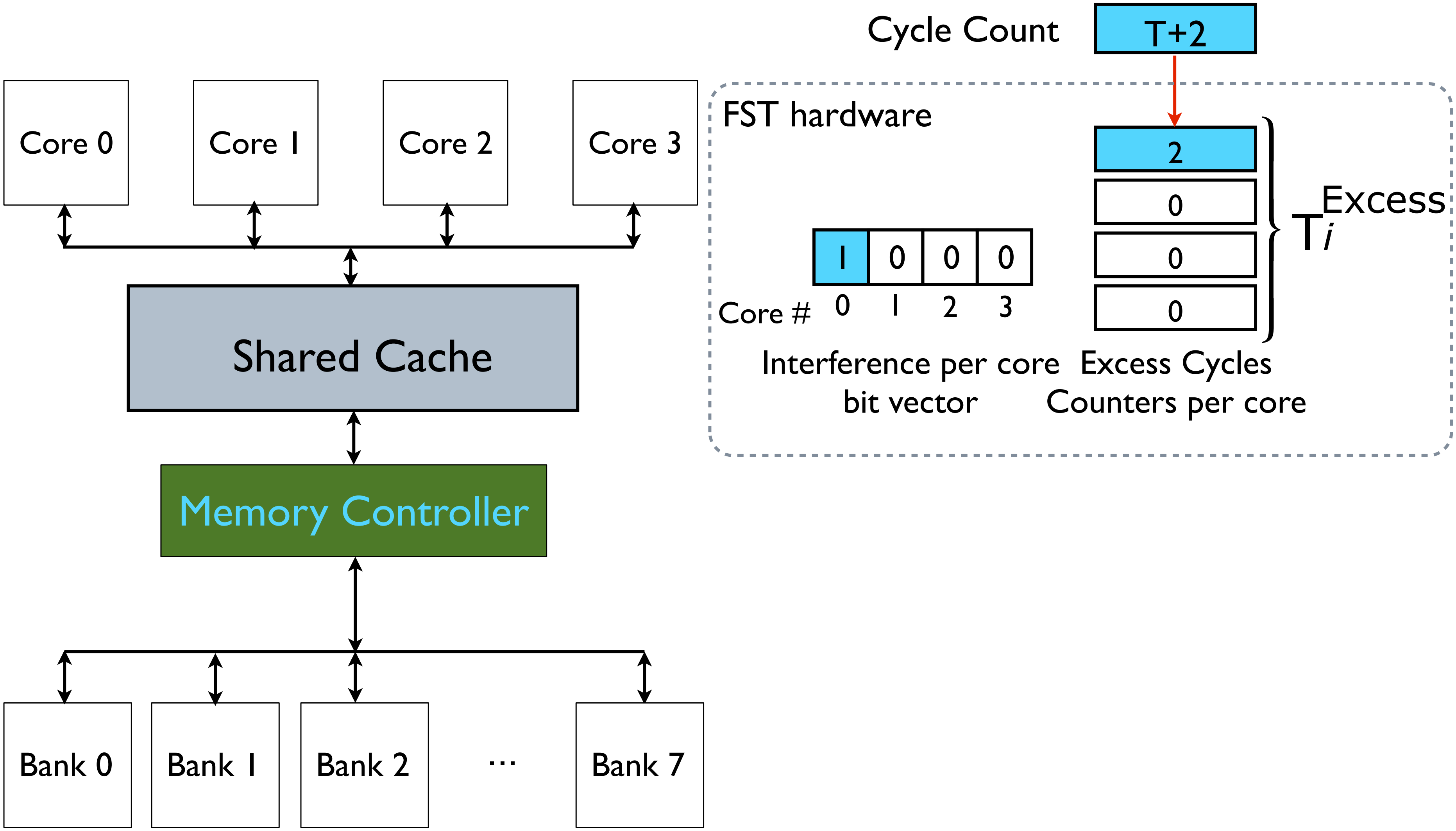
T_i^{Excess}

Excess Cycles Counters per core

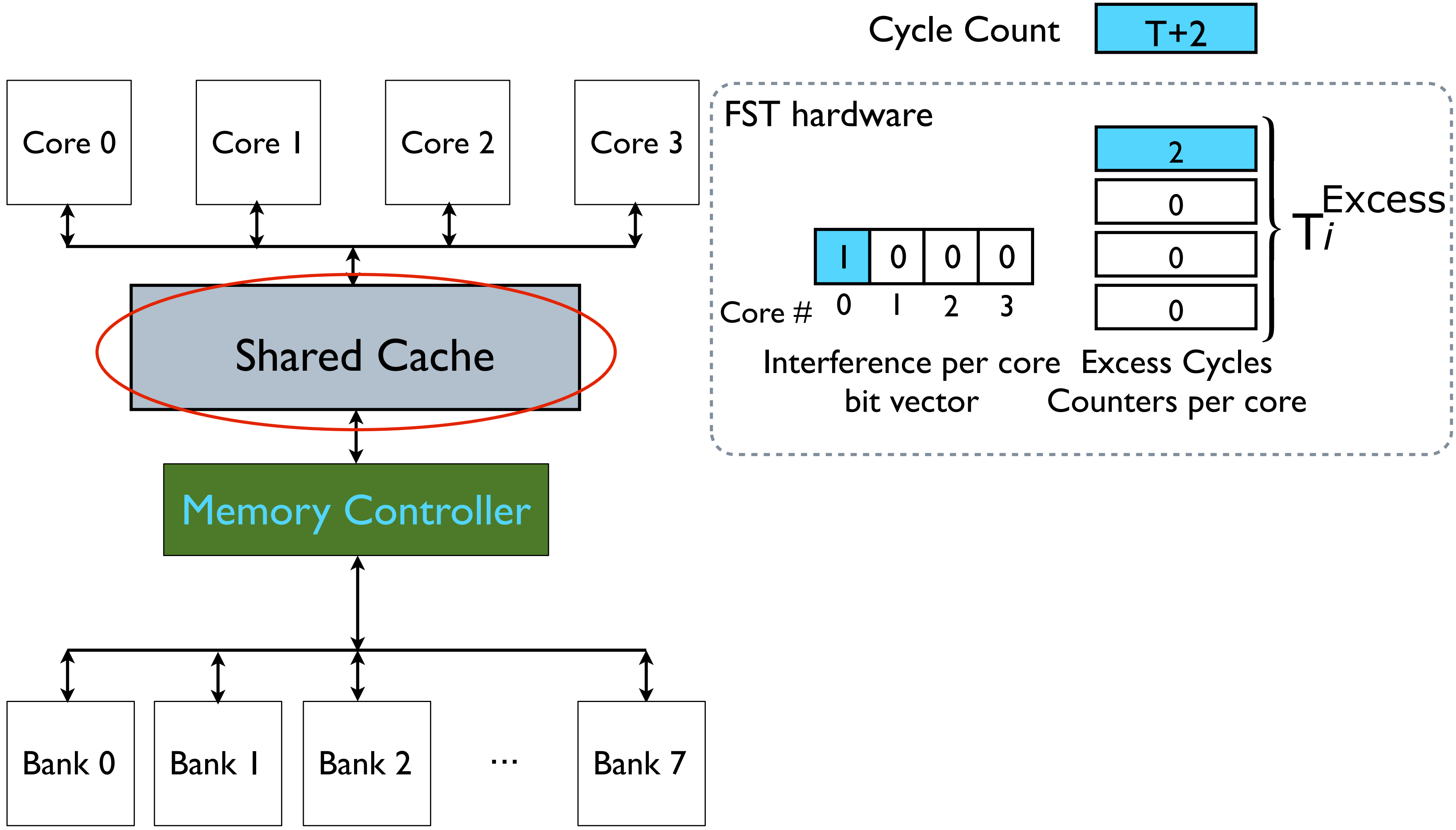
Tracking Inter-Core Interference



Tracking Inter-Core Interference



Tracking Inter-Core Interference



Cycle Count T+2

Core 0

Core 1

Core 2

Core 3

Shared Cache

Memory Controller

Bank 0

Bank 1

Bank 2

…

Bank 7

FST hardware

Core #	0	1	2	3
	1	0	0	0

Interference per core bit vector

2

0

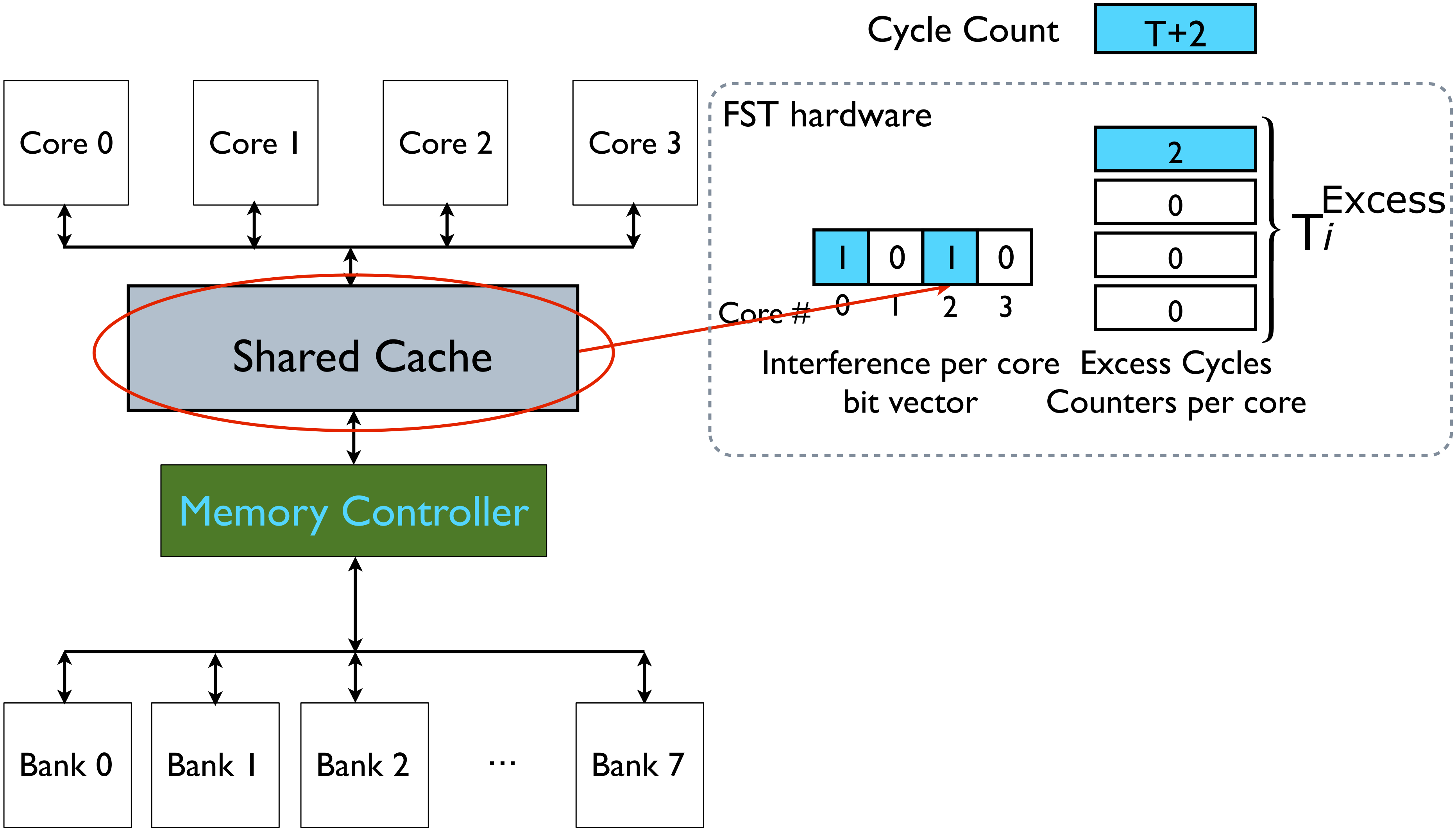
0

0

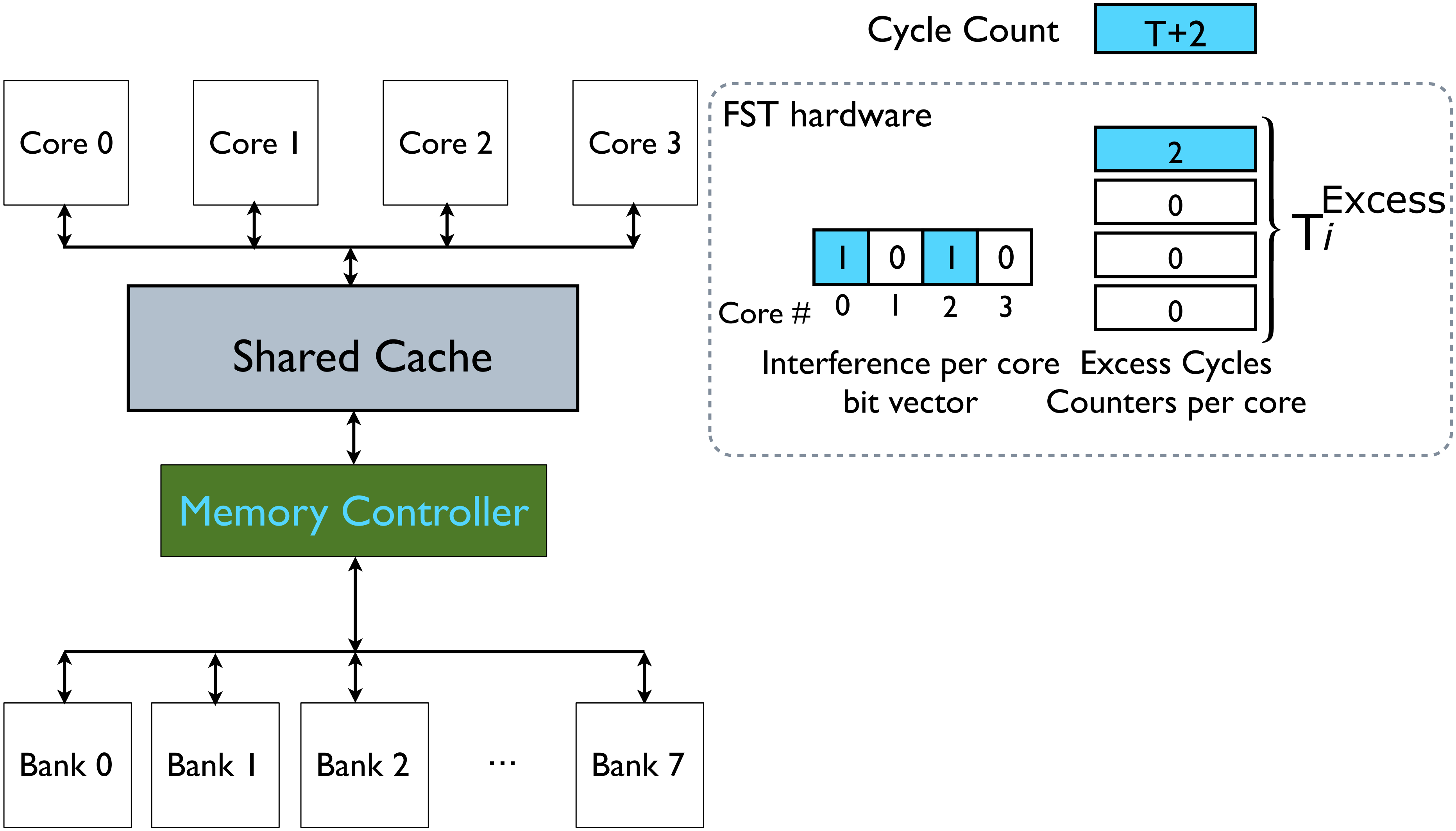
Excess Cycles Counters per core

T_i^{Excess}

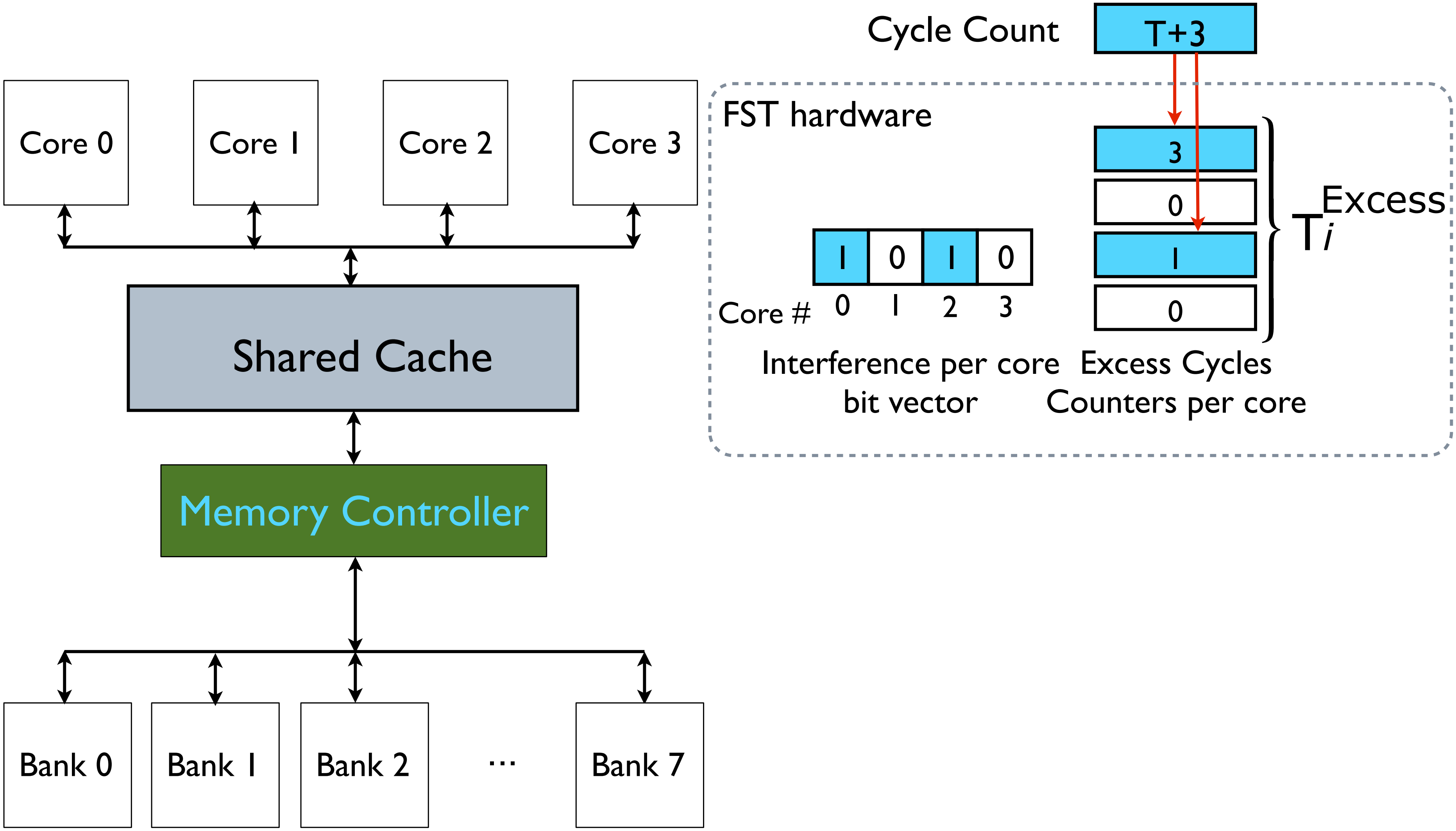
Tracking Inter-Core Interference



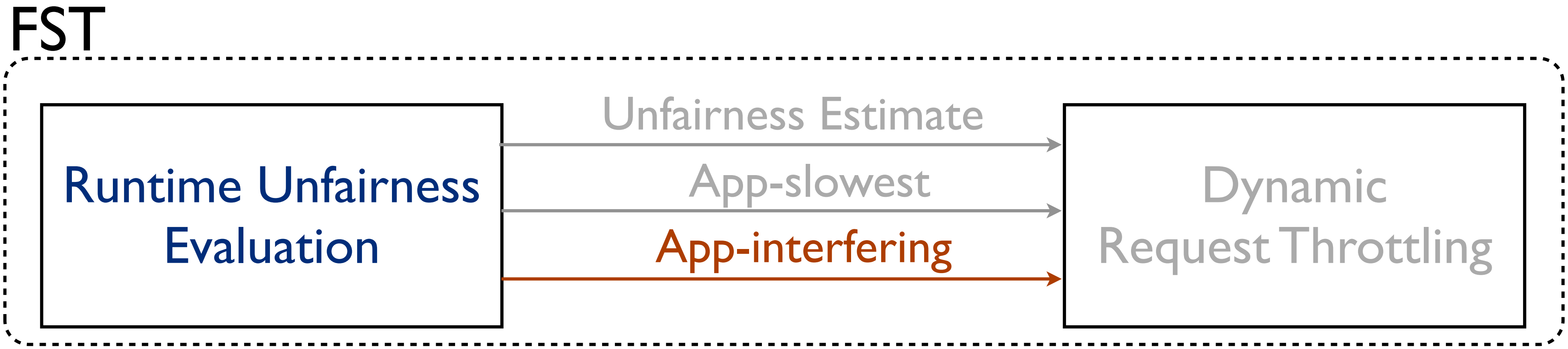
Tracking Inter-Core Interference



Tracking Inter-Core Interference



Fairness via Source Throttling (FST)



1- Estimating system unfairness
2- Find app. with the highest slowdown (App-slowest)
3- Find app. causing most interference for App-slowest (App-interfering)

```
if (Unfairness Estimate >Target)
{
 1-Throttle down App-interfering
 2-Throttle up App-slowest
}
```

Tracking Inter-Core Interference

Tracking Inter-Core Interference

- To identify App-interfering, for each core *i*

Tracking Inter-Core Interference

- To identify App-interfering, for each core *i*
  - FST separately tracks interference caused by each core *j* (*j* *!* *i*)

Tracking Inter-Core Interference

- To identify App-interfering, for each core *i*
  - FST separately tracks interference caused by each core *j* (*j* ! *i*)

Interference per core
bit vector

Core #	0	1	2	3
	0	0	0	0

Tracking Inter-Core Interference

- To identify App-interfering, for each core *i*
  - FST separately tracks interference caused by each core *j* (*j ! i*)

Interference per core
bit vector

Core #	0	1	2	3
0	-	0	0	0
1	0	-	0	0
2	0	0	-	0
3	0	0	0	-

Tracking Inter-Core Interference

- To identify App-interfering, for each core *i*
  - FST separately tracks interference caused by each core *j* (*j* ! *i*)

Interference per core
bit vector

Interfered with core

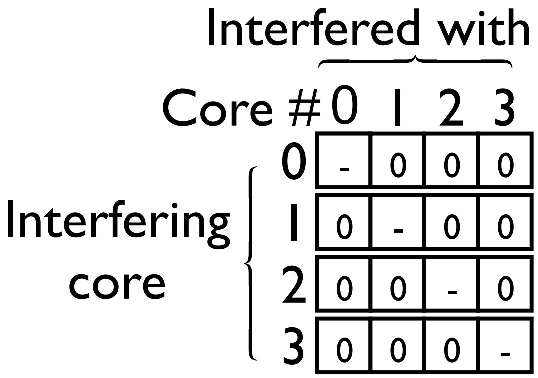
Interfering core

Core #	0	1	2	3
0	-	0	0	0
1	0	-	0	0
2	0	0	-	0
3	0	0	0	-

Tracking Inter-Core Interference

- To identify App-interfering, for each core *i*
  - FST separately tracks interference caused by each core *j* (*j* ! *i*)

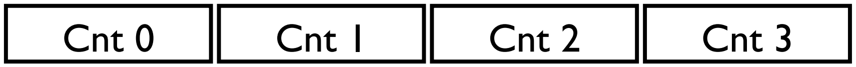
Interference per core
bit vector



Interfered with core

Interfering core	Core #	0	1	2	3
	0	-	0	0	0
	1	0	-	0	0
	2	0	0	-	0
	3	0	0	0	-

Excess Cycles
Counters per core



Cnt 0	Cnt 1	Cnt 2	Cnt 3

Tracking Inter-Core Interference

- To identify App-interfering, for each core *i*
  - FST separately tracks interference caused by each core *j* (*j* ! *i*)

Interference per core
bit vector

Interfered with core

Interfering core

Core #	0	1	2	3
0	-	0	0	0
1	0	-	0	0
2	0	0	-	0
3	0	0	0	-

Excess Cycles
Counters per core

-	Cnt 0,1	Cnt 0,2	Cnt 0,3
Cnt 1,0	-	Cnt 1,2	Cnt 1,3
Cnt 2,0	Cnt 2,1	-	Cnt 2,3
Cnt 3,0	Cnt 3,1	Cnt 3,2	-

Tracking Inter-Core Interference

- To identify App-interfering, for each core *i*
  - FST separately tracks interference caused by each core *j* (*j* ! *i*)

Interference per core
bit vector

Interfered with core

Interfering core

Core #	0	1	2	3
0	-	0	0	0
1	0	-	0	0
2	0	1	-	0
3	0	0	0	-

Excess Cycles
Counters per core

-	Cnt 0,1	Cnt 0,2	Cnt 0,3
Cnt 1,0	-	Cnt 1,2	Cnt 1,3
Cnt 2,0	Cnt 2,1	-	Cnt 2,3
Cnt 3,0	Cnt 3,1	Cnt 3,2	-

Tracking Inter-Core Interference

- To identify App-interfering, for each core *i*
  - FST separately tracks interference caused by each core *j* (*j* ! *i*)

Interference per core
bit vector

Interfered with core

Interfering core

Core #	0	1	2	3
0	-	0	0	0
1	0	-	0	0
2	0	1	-	0
3	0	0	0	-

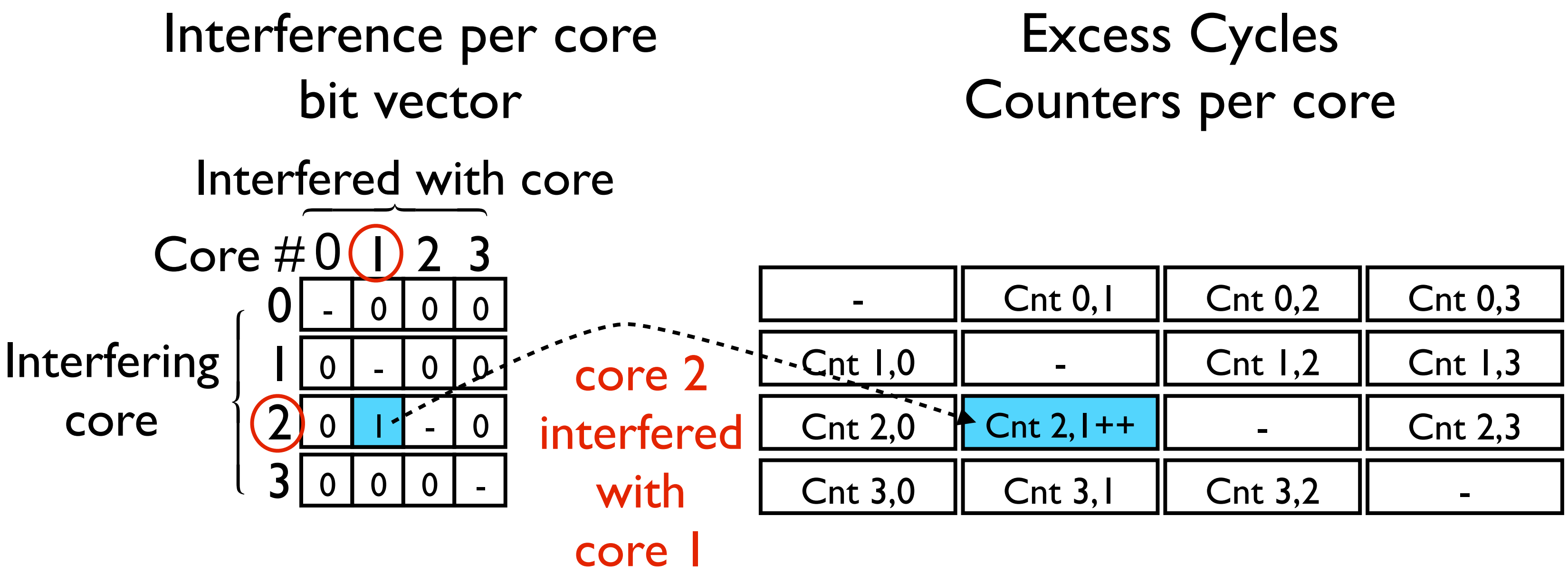
core 2 interfered with core 1

Excess Cycles
Counters per core

-	Cnt 0,1	Cnt 0,2	Cnt 0,3
Cnt 1,0	-	Cnt 1,2	Cnt 1,3
Cnt 2,0	Cnt 2,1	-	Cnt 2,3
Cnt 3,0	Cnt 3,1	Cnt 3,2	-

Tracking Inter-Core Interference

- To identify App-interfering, for each core *i*
  - FST separately tracks interference caused by each core *j* (*j* ! *i*)



Interference per core bit vector

Interfered with core

Interfering core

Core #	0	1	2	3
0	-	0	0	0
1	0	-	0	0
2	0	1	-	0
3	0	0	0	-

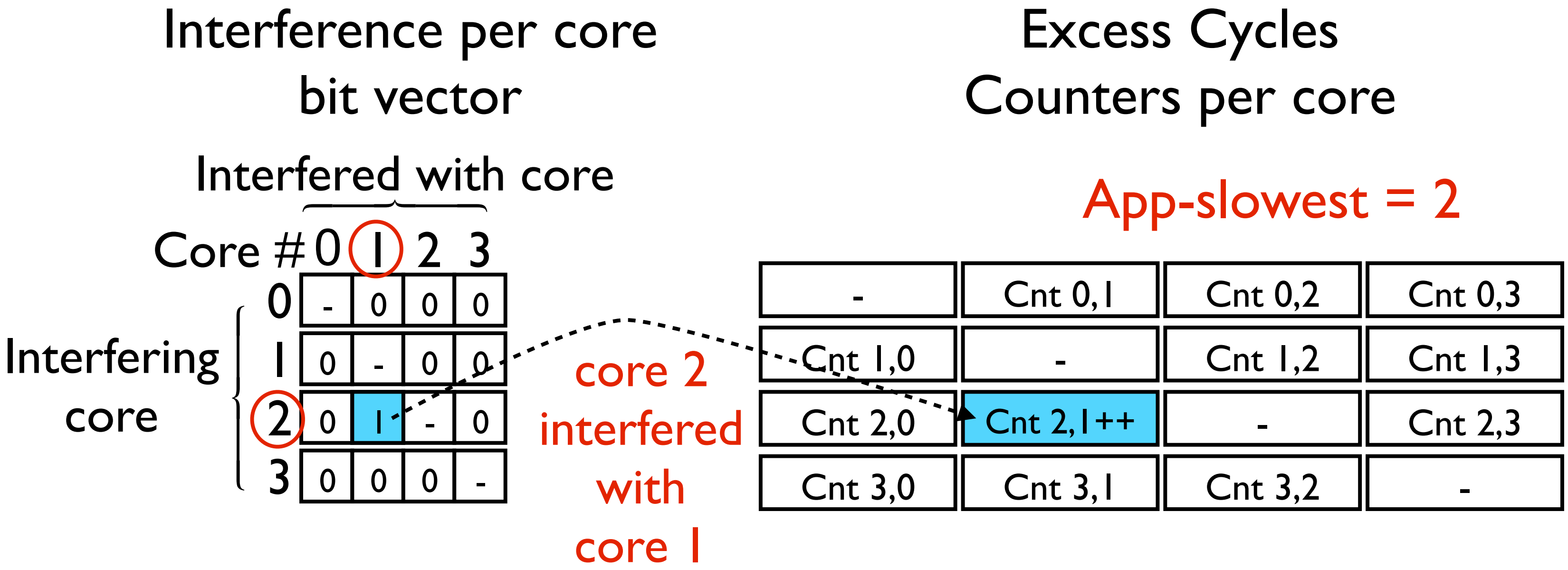
core 2 interfered with core 1

Excess Cycles Counters per core

-	Cnt 0,1	Cnt 0,2	Cnt 0,3
Cnt 1,0	-	Cnt 1,2	Cnt 1,3
Cnt 2,0	Cnt 2,1++	-	Cnt 2,3
Cnt 3,0	Cnt 3,1	Cnt 3,2	-

Tracking Inter-Core Interference

- To identify App-interfering, for each core *i*
  - FST separately tracks interference caused by each core *j* (*j ! i*)



Interference per core bit vector

Interfered with core

Core #	0	1	2	3
0	-	0	0	0
1	0	-	0	0
2	0	1	-	0
3	0	0	0	-

Interfering core

core 2 interfered with core 1

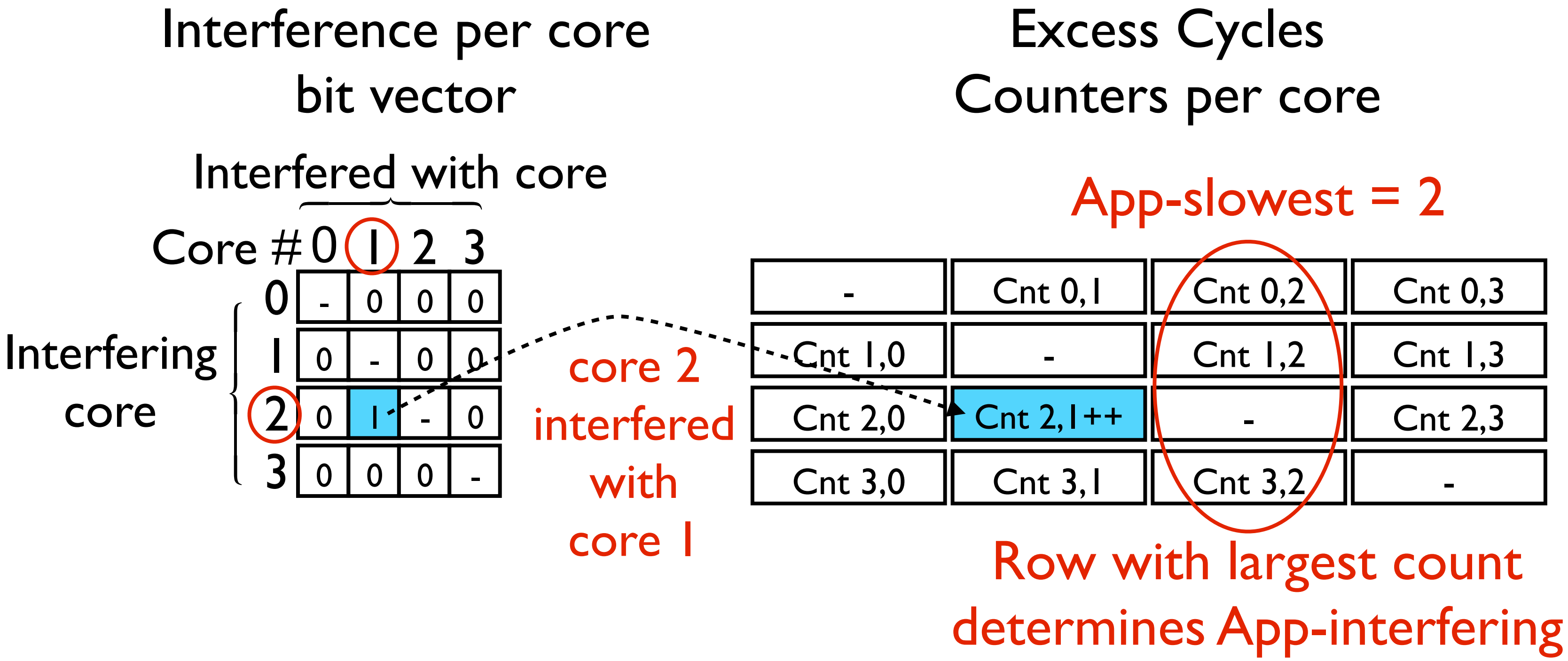
Excess Cycles Counters per core

App-slowest = 2

-	Cnt 0,1	Cnt 0,2	Cnt 0,3
Cnt 1,0	-	Cnt 1,2	Cnt 1,3
Cnt 2,0	Cnt 2,1++	-	Cnt 2,3
Cnt 3,0	Cnt 3,1	Cnt 3,2	-

Tracking Inter-Core Interference

- To identify App-interfering, for each core *i*
  - FST separately tracks interference caused by each core *j* (*j* ! *i*)



Interference per core bit vector

Interfered with core

Interfering core

Core #	0	1	2	3
0	-	0	0	0
1	0	-	0	0
2	0	1	-	0
3	0	0	0	-

core 2 interfered with core 1

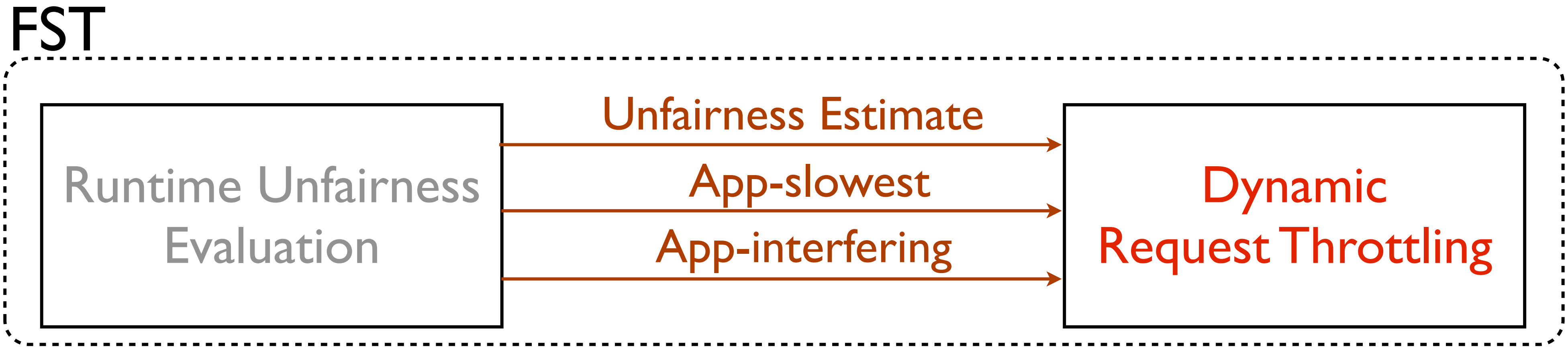
Excess Cycles Counters per core

App-slowest = 2

-	Cnt 0,1	Cnt 0,2	Cnt 0,3
Cnt 1,0	-	Cnt 1,2	Cnt 1,3
Cnt 2,0	Cnt 2,1++	-	Cnt 2,3
Cnt 3,0	Cnt 3,1	Cnt 3,2	-

Row with largest count determines App-interfering

Fairness via Source Throttling (FST)



1- Estimating system unfairness
2- Find app. with the highest slowdown (App-slowest)
3- Find app. causing most interference for App-slowest (App-interfering)

```
if (Unfairness Estimate >Target)
{
 1-Throttle down App-interfering
 2-Throttle up App-slowest
}
```

Dynamic Request Throttling

Dynamic Request Throttling

- Goal: Adjust how aggressively each core makes requests to the shared resources

Dynamic Request Throttling

- Goal: Adjust how aggressively each core makes requests to the shared resources
- Mechanisms:
  - Miss Status Holding Register (MSHR) quota

Dynamic Request Throttling

- Goal: Adjust how aggressively each core makes requests to the shared resources
- Mechanisms:
  - Miss Status Holding Register (MSHR) quota
    - Controls the number of concurrent requests accessing shared resources from each application

Dynamic Request Throttling

- Goal: Adjust how aggressively each core makes requests to the shared resources
- Mechanisms:
  - Miss Status Holding Register (MSHR) quota
    - Controls the number of concurrent requests accessing shared resources from each application
  - Request injection frequency

Dynamic Request Throttling

- Goal: Adjust how aggressively each core makes requests to the shared resources
- Mechanisms:
  - Miss Status Holding Register (MSHR) quota
    - Controls the number of concurrent requests accessing shared resources from each application
  - Request injection frequency
    - Controls how often memory requests are issued to the last level cache from the MSHRs

Dynamic Request Throttling

- Throttling level assigned to each core determines both MSHR quota and request injection rate

Dynamic Request Throttling

- Throttling level assigned to each core determines both MSHR quota and request injection rate

Throttling level	MSHR quota	Request Injection Rate
100%	128	Every cycle
50%	64	Every other cycle
25%	32	Once every 4 cycles
10%	12	Once every 10 cycles
5%	6	Once every 20 cycles
4%	5	Once every 25 cycles
3%	3	Once every 30 cycles
2%	2	Once every 50 cycles

Total # of MSHRs: 128

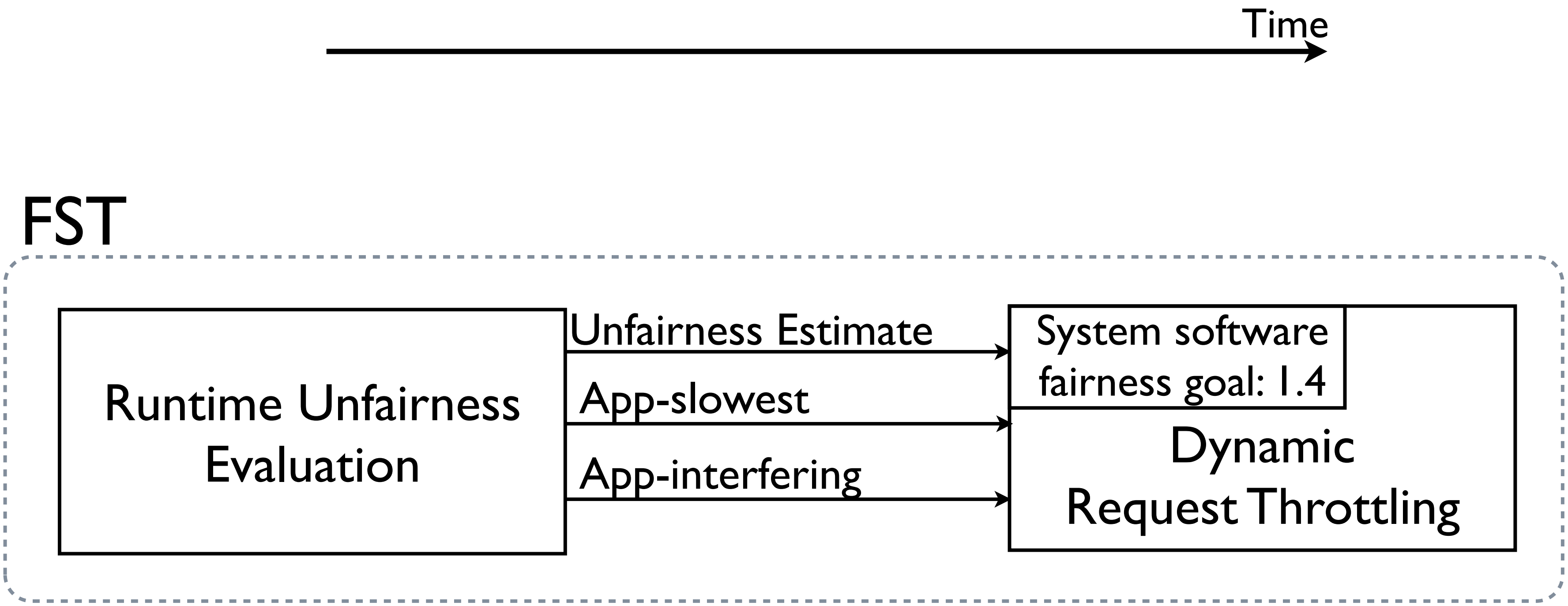
Dynamic Request Throttling

- Throttling level assigned to each core determines both MSHR quota and request injection rate

Throttling level	MSHR quota	Request Injection Rate
100%	128	Every cycle
50%	64	Every other cycle
25%	32	Once every 4 cycles
10%	12	Once every 10 cycles
5%	6	Once every 20 cycles
4%	5	Once every 25 cycles
3%	3	Once every 30 cycles
2%	2	Once every 50 cycles

Total # of MSHRs: 128

FST at Work



Time

FST

Runtime Unfairness Evaluation

Unfairness Estimate

App-slowest

App-interfering

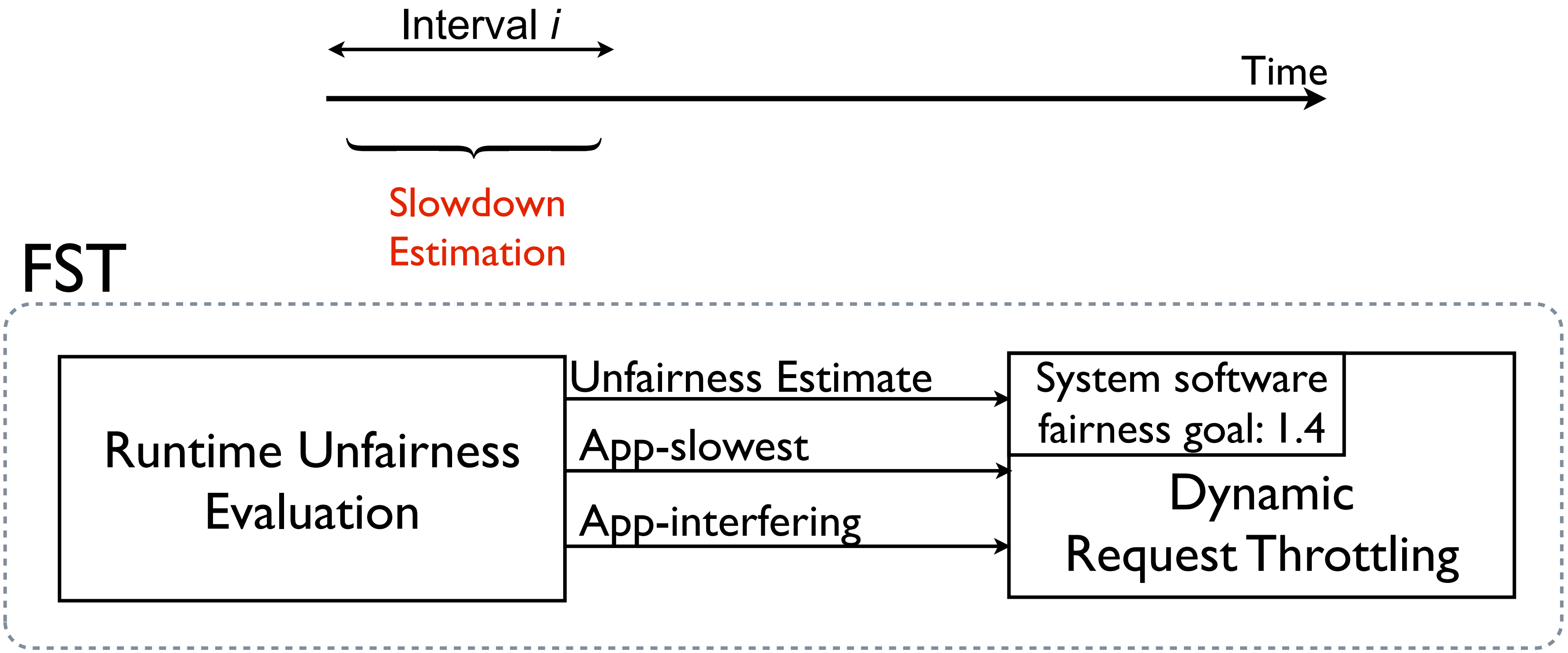
System software fairness goal: 1.4

Dynamic Request Throttling

	Core 0	Core 1	Core 2	Core 3
Interval *i*				
Interval *i* + 1				
Interval *i* + 2				

Throttling Levels

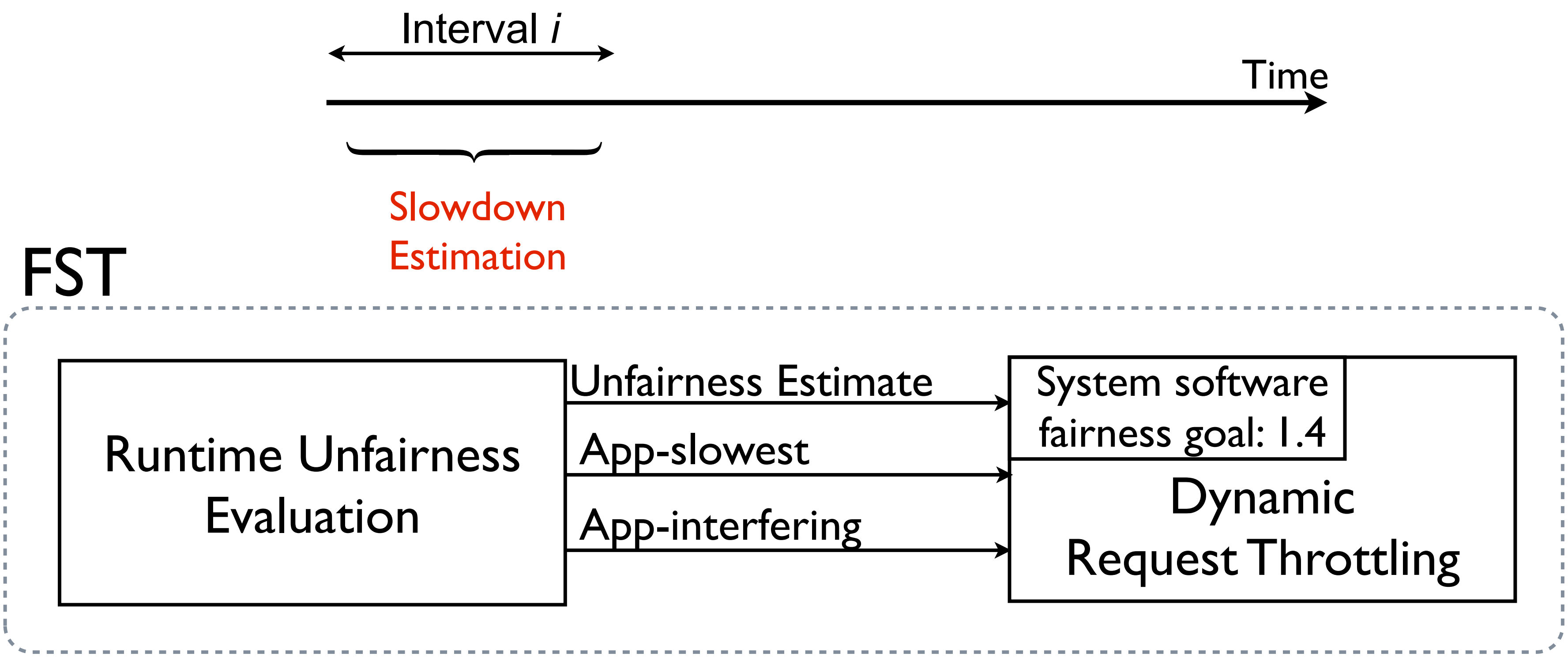
FST at Work


Interval *i*
Time
Slowdown Estimation
FST
Runtime Unfairness Evaluation
Unfairness Estimate
App-slowest
App-interfering
System software fairness goal: 1.4
Dynamic Request Throttling

	Core 0	Core 1	Core 2	Core 3
Interval *i*	50%	100%	10%	100%
Interval *i* + 1				
Interval *i* + 2				

Throttling Levels

FST at Work



Interval *i*

Time

Slowdown Estimation

FST

Runtime Unfairness Evaluation

Unfairness Estimate

App-slowest

App-interfering

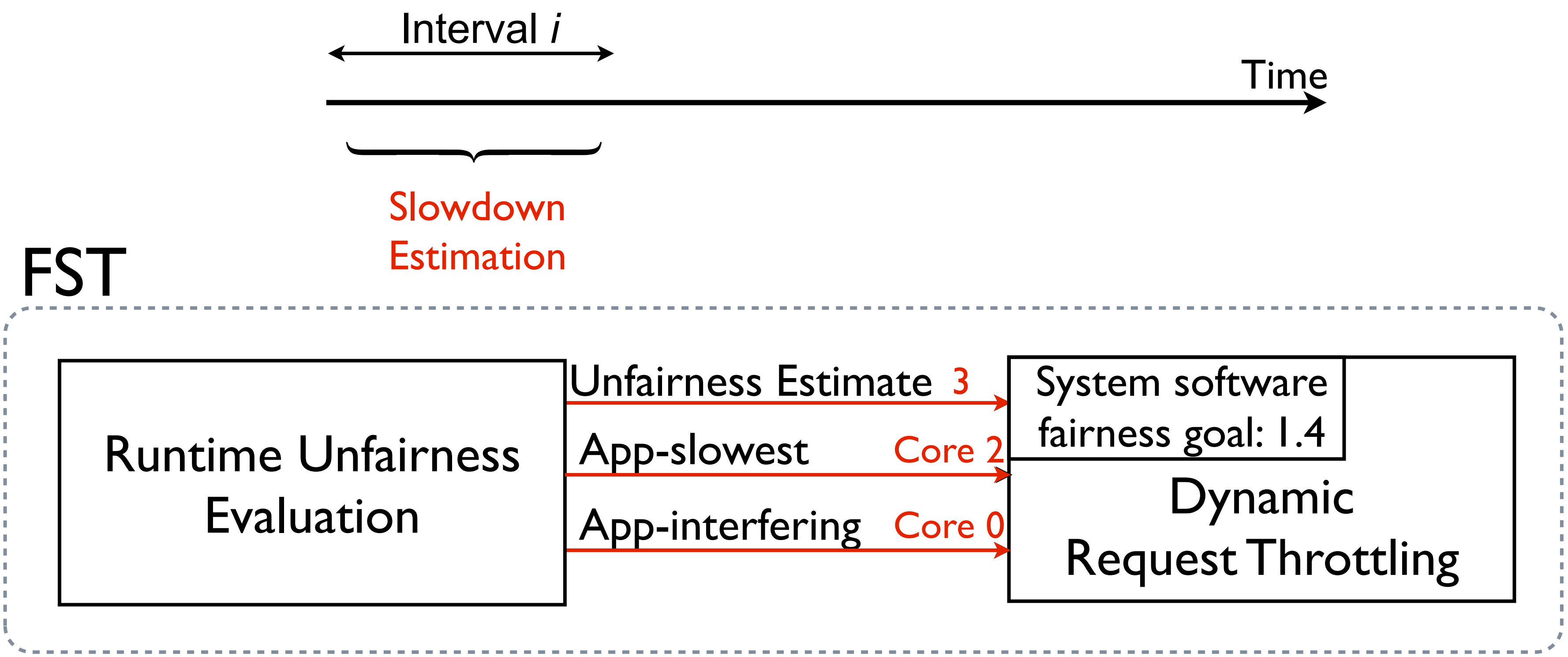
System software fairness goal: 1.4

Dynamic Request Throttling

	Core 0	Core 1	Core 2	Core 3
Interval *i*	50%	100%	10%	100%
Interval *i* + 1				
Interval *i* + 2				

Throttling Levels

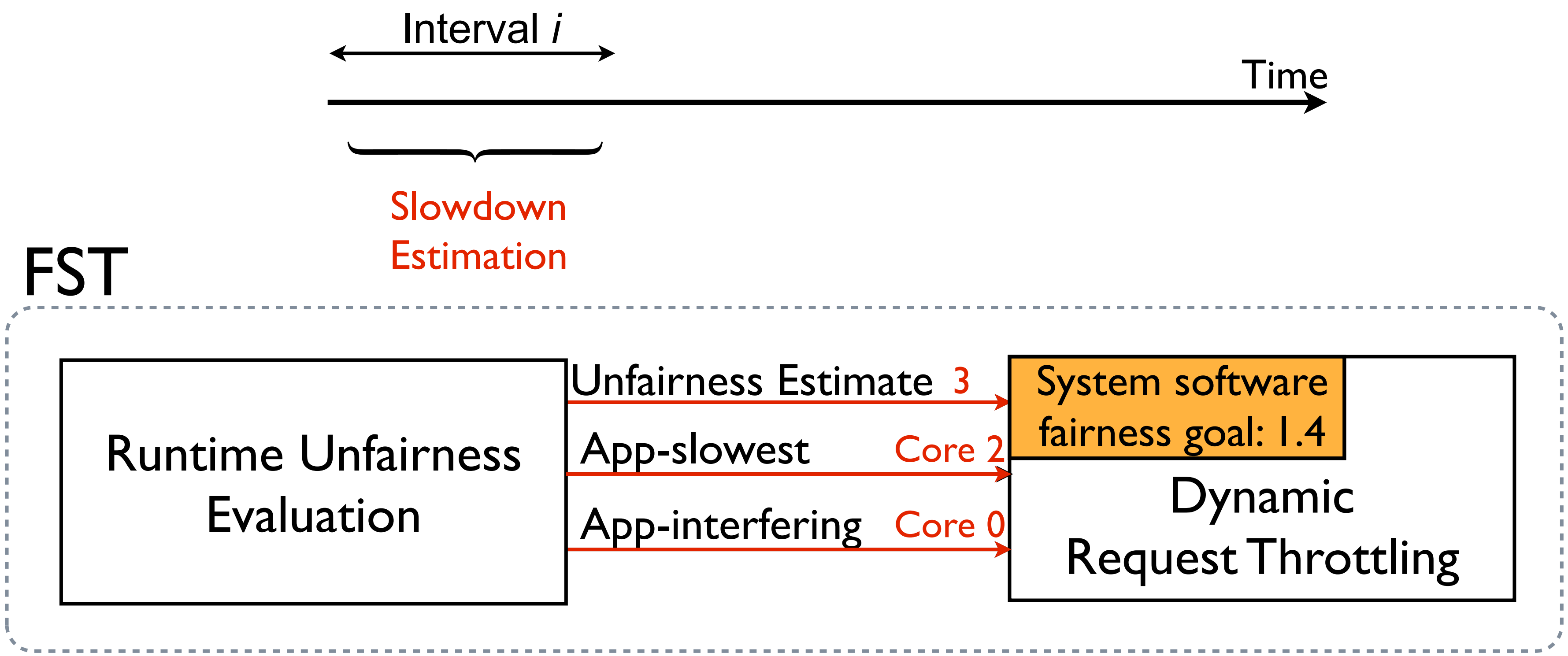
FST at Work



	Core 0	Core 1	Core 2	Core 3
Interval *i*	50%	100%	10%	100%
Interval *i* + 1				
Interval *i* + 2				

Throttling Levels

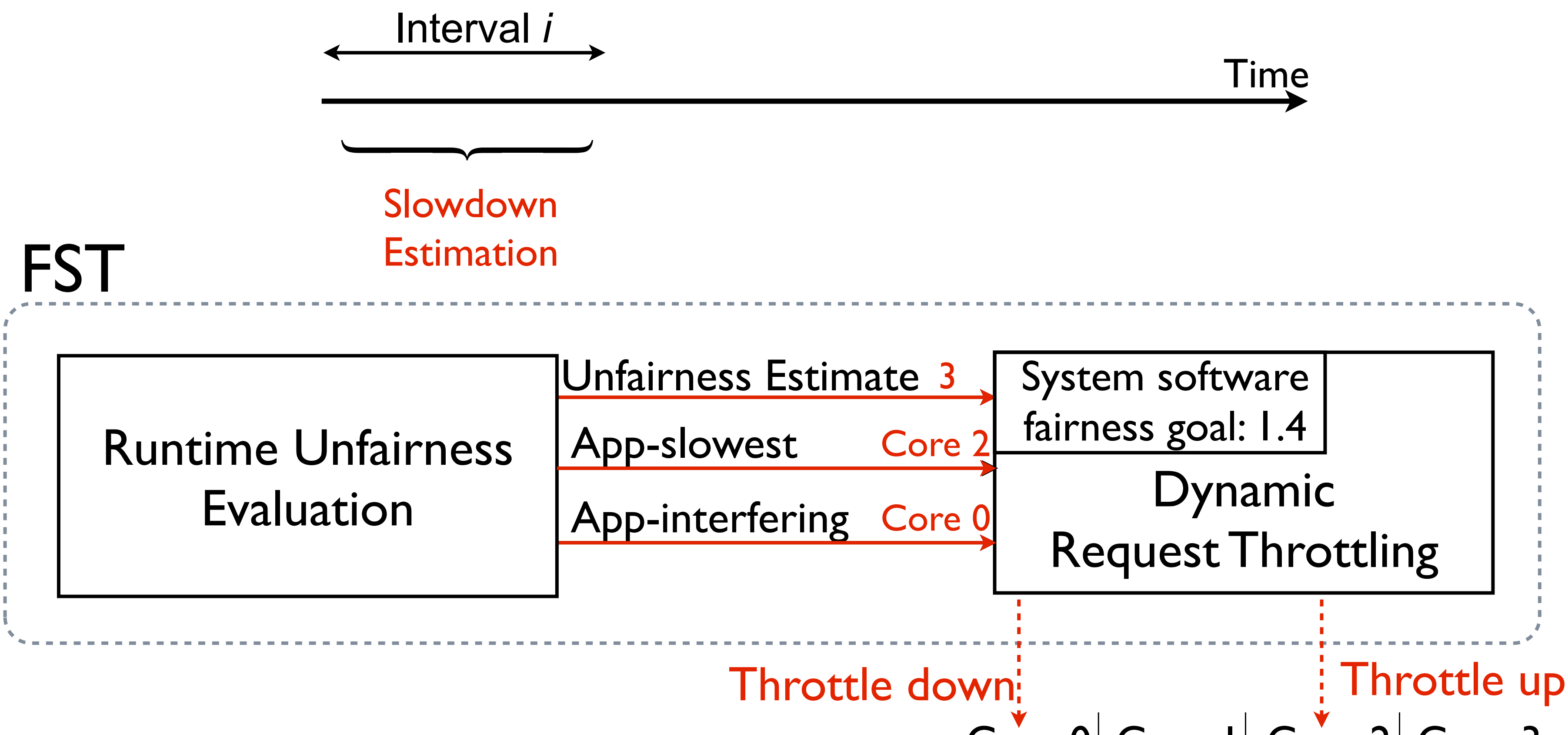
FST at Work



	Core 0	Core 1	Core 2	Core 3
Interval *i*	50%	100%	10%	100%
Interval *i* + 1				
Interval *i* + 2				

Throttling Levels

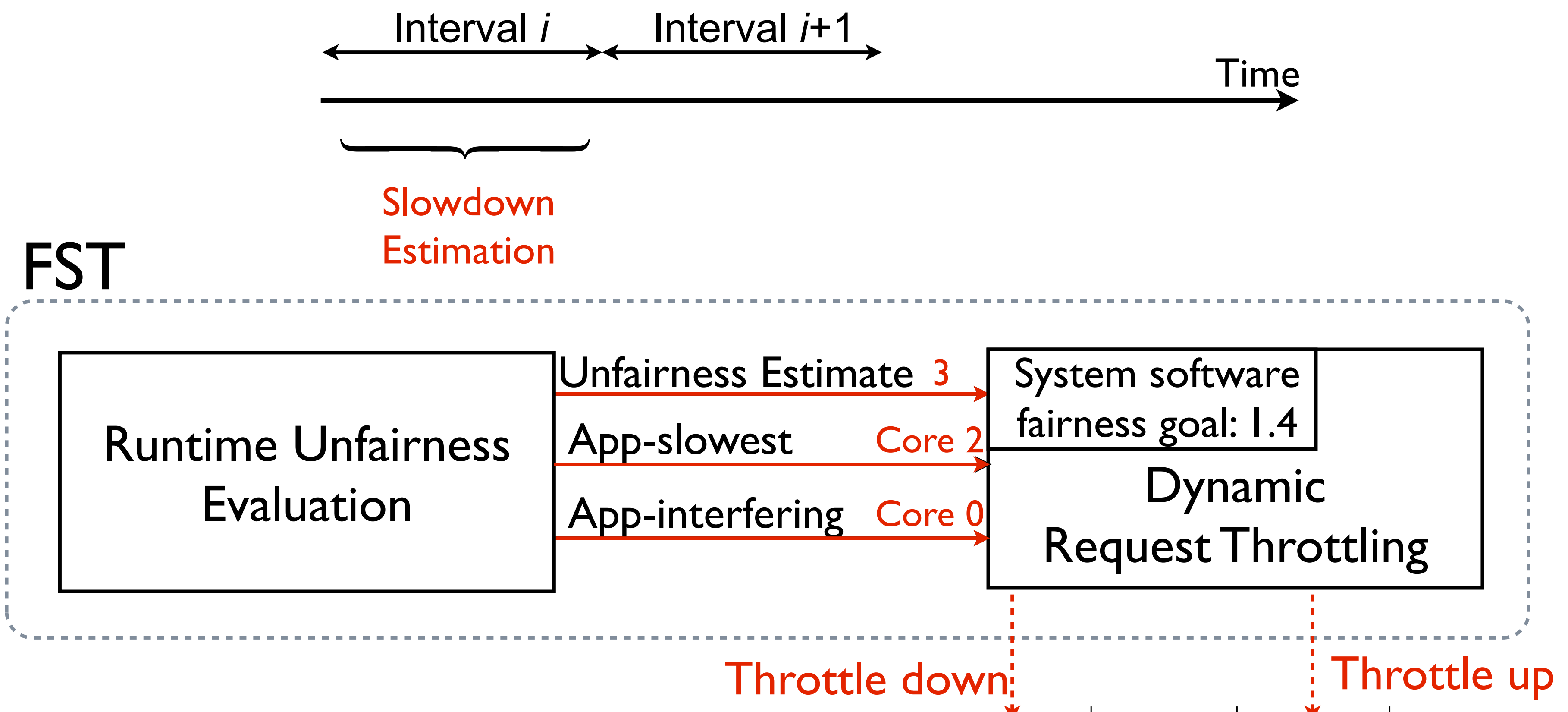
FST at Work



	Core 0	Core 1	Core 2	Core 3
Interval *i*	50%	100%	10%	100%
Interval *i* + 1				
Interval *i* + 2				

Throttling Levels

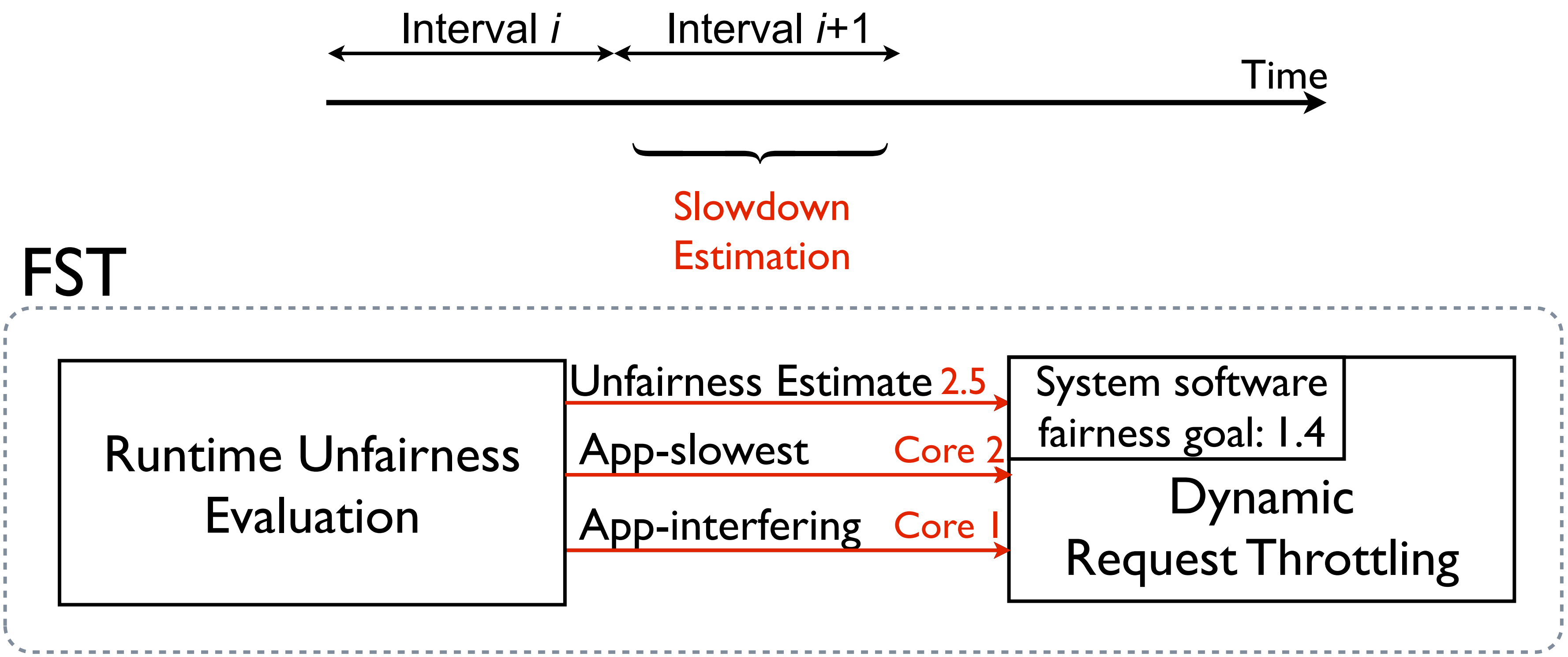
FST at Work



	Core 0	Core 1	Core 2	Core 3
Interval *i*	50%	100%	10%	100%
Interval *i* + 1	25%	100%	25%	100%
Interval *i* + 2				

Throttling Levels

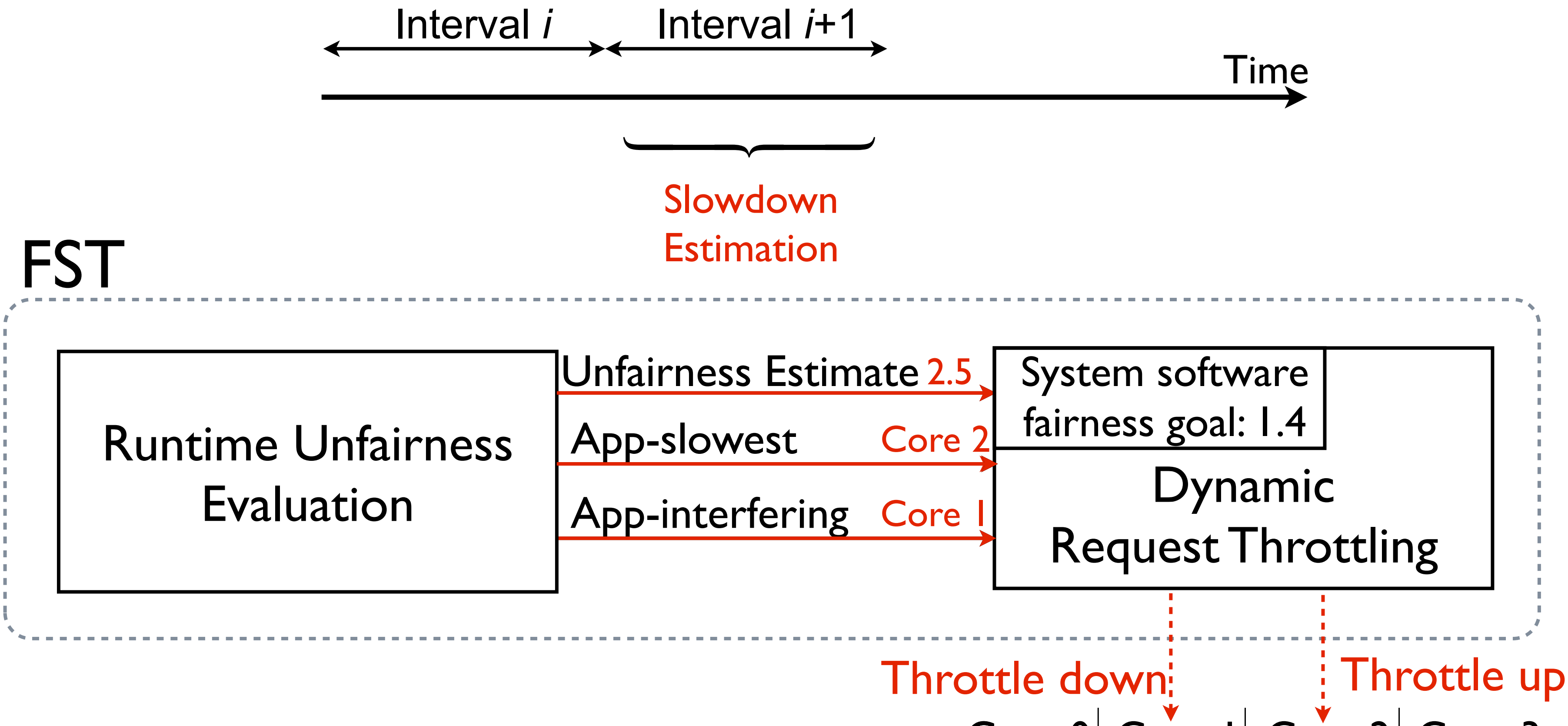
FST at Work



	Core 0	Core 1	Core 2	Core 3
Interval *i*	50%	100%	10%	100%
Interval *i* + 1	25%	100%	25%	100%
Interval *i* + 2				

Throttling Levels

FST at Work



Interval *i*

Interval *i*+1

Time

Slowdown Estimation

FST

Runtime Unfairness Evaluation

Unfairness Estimate 2.5

App-slowest Core 2

App-interfering Core 1

System software fairness goal: 1.4

Dynamic Request Throttling

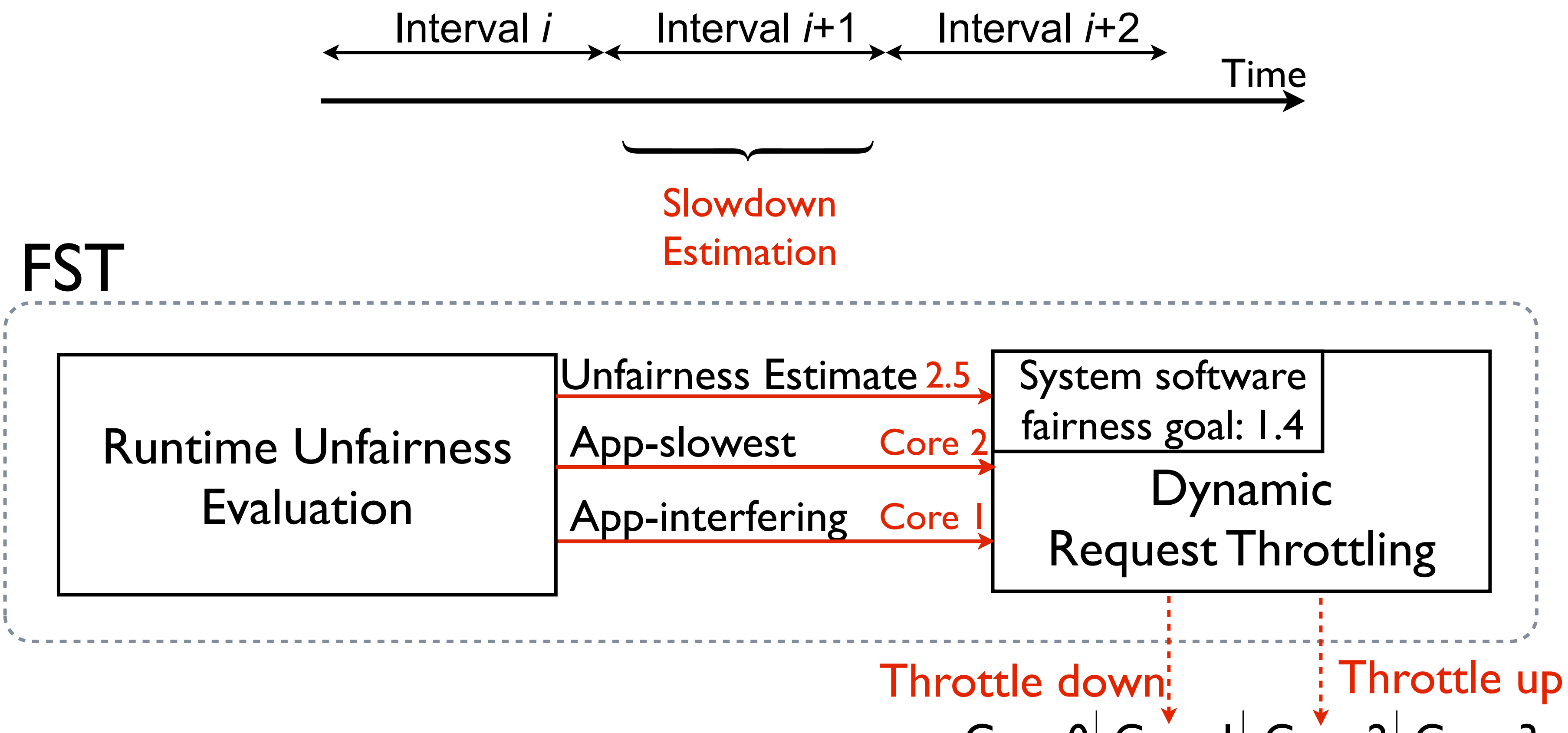
Throttle down

Throttle up

	Core 0	Core 1	Core 2	Core 3
Interval *i*	50%	100%	10%	100%
Interval *i* + 1	25%	100%	25%	100%
Interval *i* + 2				

Throttling Levels

FST at Work



	Core 0	Core 1	Core 2	Core 3
Interval *i*	50%	100%	10%	100%
Interval *i* + 1	25%	100%	25%	100%
Interval *i* + 2	25%	50%	50%	100%

Throttling Levels

System Software Support

System Software Support

- Different fairness objectives can be configured by system software

System Software Support

- Different fairness objectives can be configured by system software
  - *Estimated Unfairness > Target Unfairness*

System Software Support

- Different fairness objectives can be configured by system software
  - *Estimated Unfairness > Target Unfairness*
  - *Estimated Max Slowdown > Target Max Slowdown*

System Software Support

- Different fairness objectives can be configured by system software
  - *Estimated Unfairness > Target Unfairness*
  - *Estimated Max Slowdown > Target Max Slowdown*
  - *Estimated Slowdown(i) > Target Slowdown(i)*

System Software Support

- Different fairness objectives can be configured by system software
  - *Estimated Unfairness > Target Unfairness*
  - *Estimated Max Slowdown > Target Max Slowdown*
  - *Estimated Slowdown(i) > Target Slowdown(i)*

- Support for thread priorities

System Software Support

- Different fairness objectives can be configured by system software
  - *Estimated Unfairness > Target Unfairness*
  - *Estimated Max Slowdown > Target Max Slowdown*
  - *Estimated Slowdown(i) > Target Slowdown(i)*

- Support for thread priorities
  - *Weighted Slowdown(i) =*
    Estimated Slowdown(i) x Weight(i)

Hardware Cost

- Total storage cost required for 4 cores is ~ 12KB
- FST does not require any structures or logic that are on the processor’s critical path

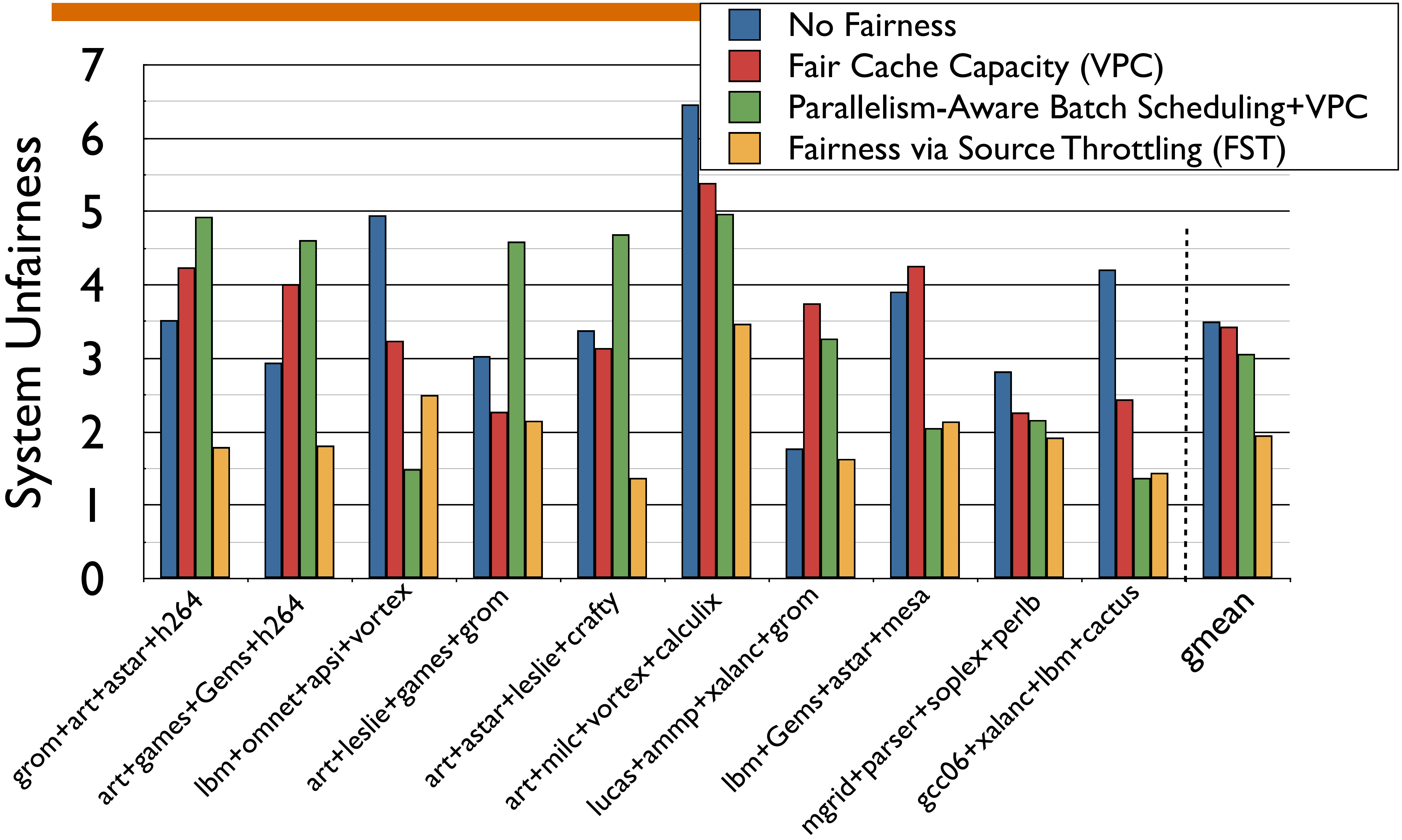
Outline

- Background and Problem
- Motivation for Source Throttling
- Fairness via Source Throttling (FST)
- Evaluation
- Conclusion

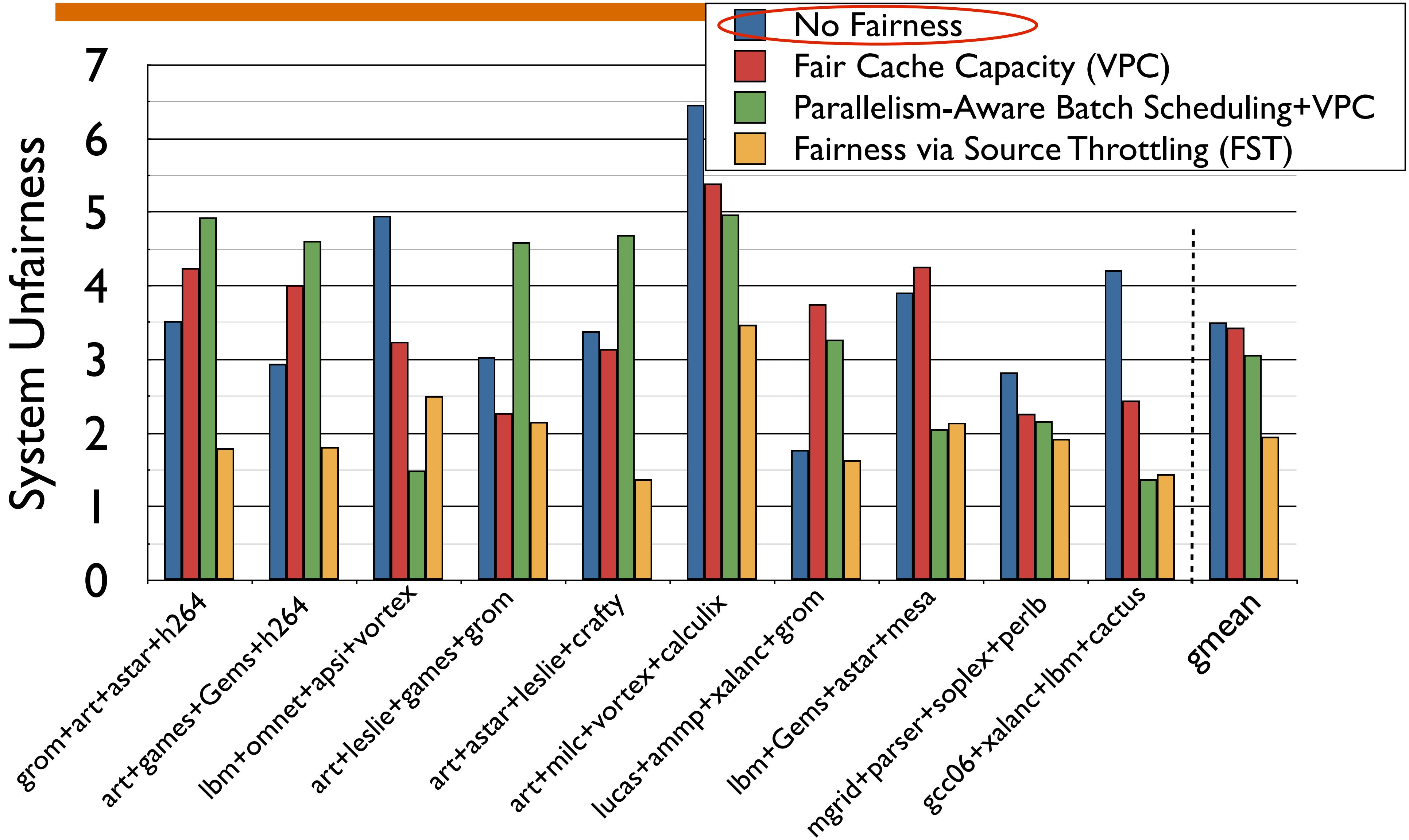
Evaluation Methodology

- x86 cycle accurate simulator
- Baseline processor configuration
  - Per-core
    - 4-wide issue, out-of-order, 256 entry ROB
  - Shared (4-core system)
    - 128 MSHRs
    - 2 MB, 16-way L2 cache
  - Main Memory
    - DDR3 1333 MHz
    - Latency of 15ns per command (*tRP, tRCD, CL*)
    - 8B wide core to memory bus

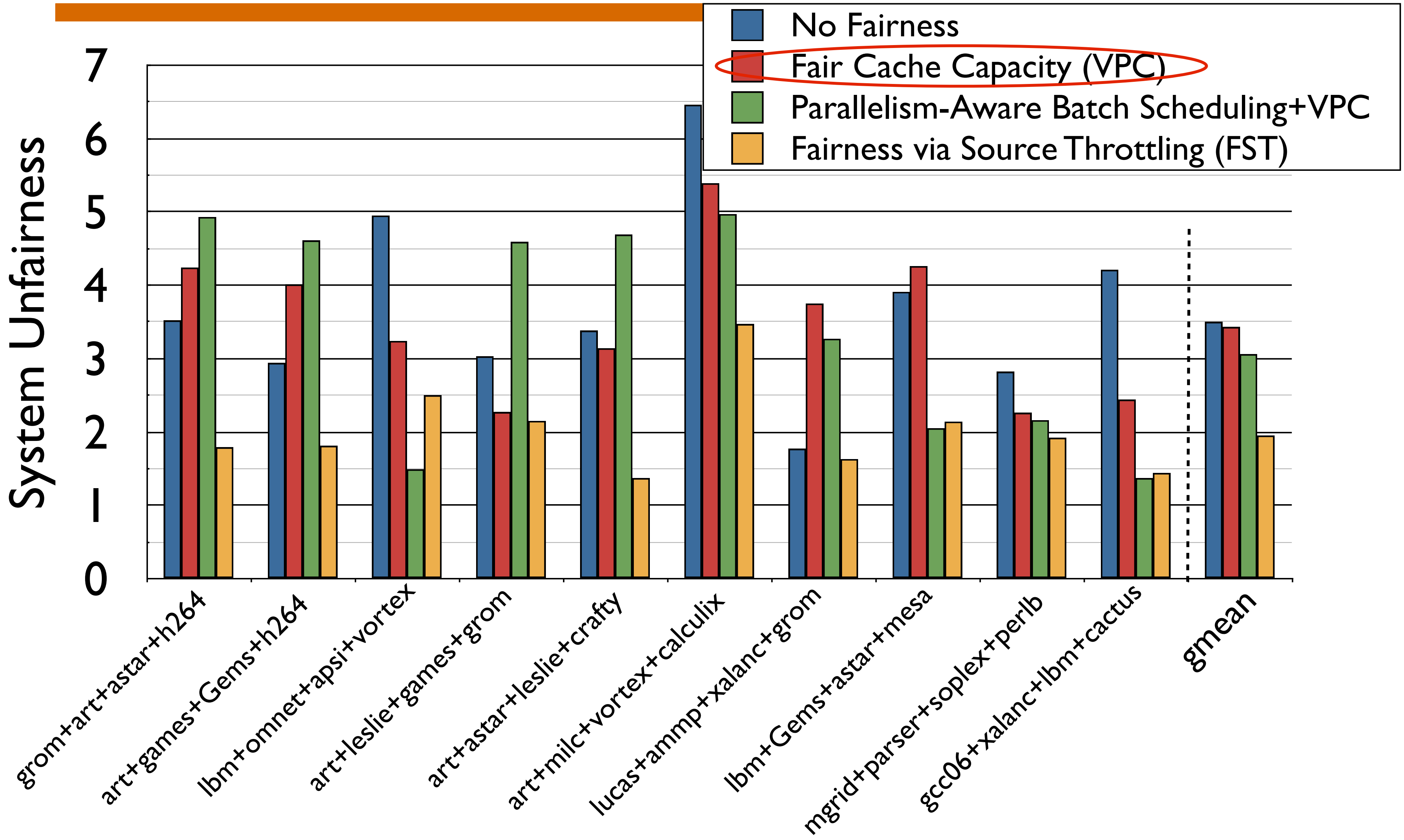
System Unfairness Results



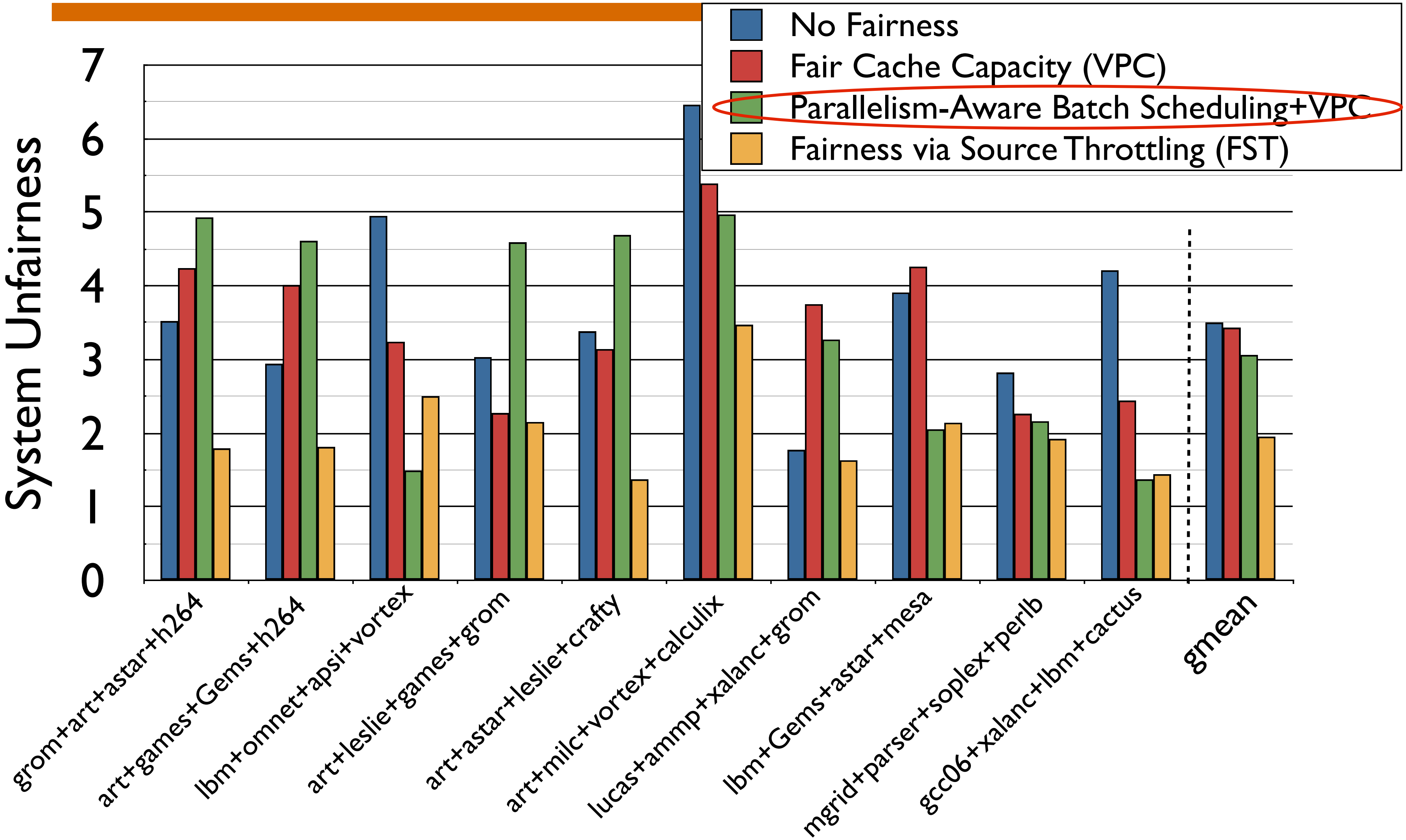
System Unfairness Results



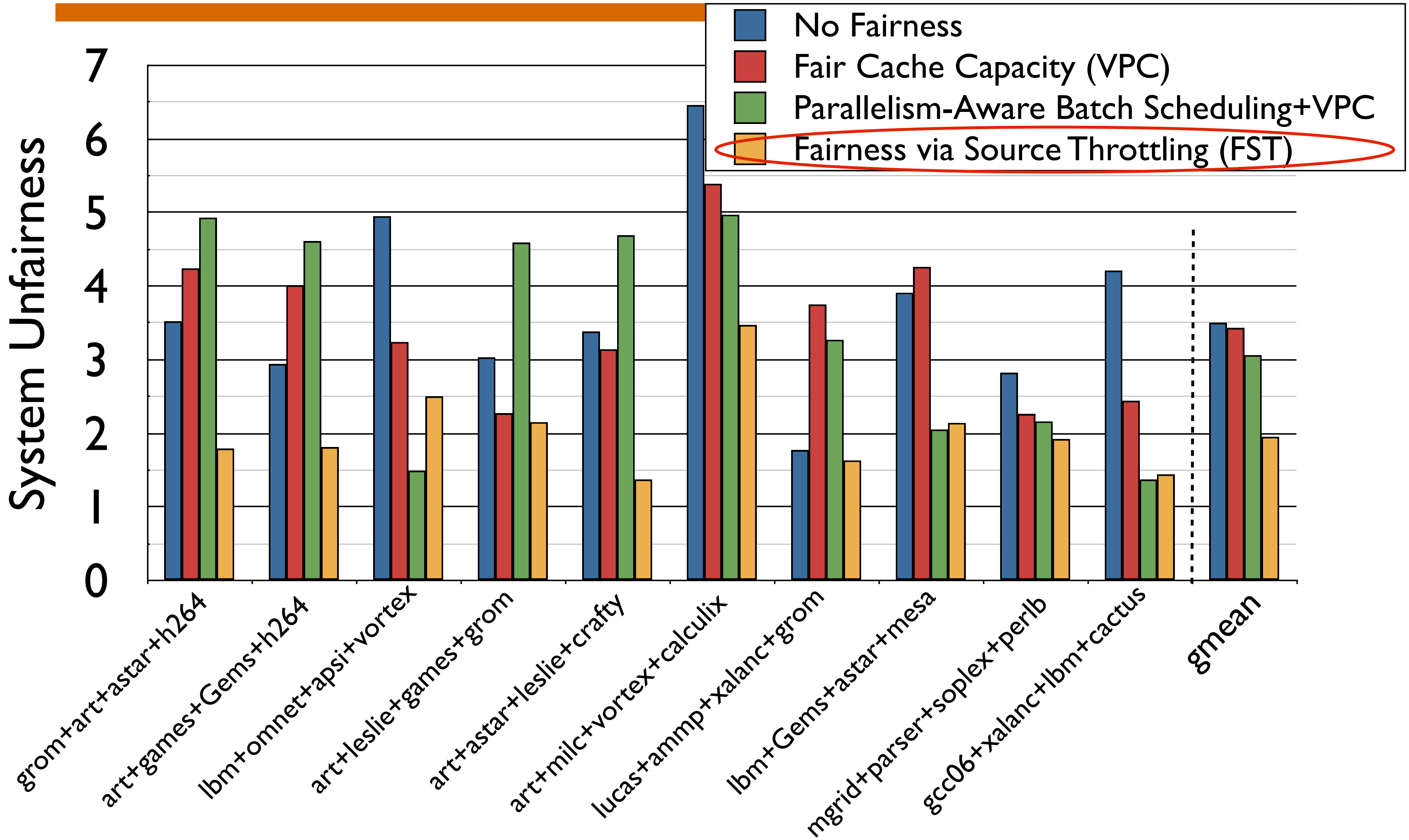
System Unfairness Results



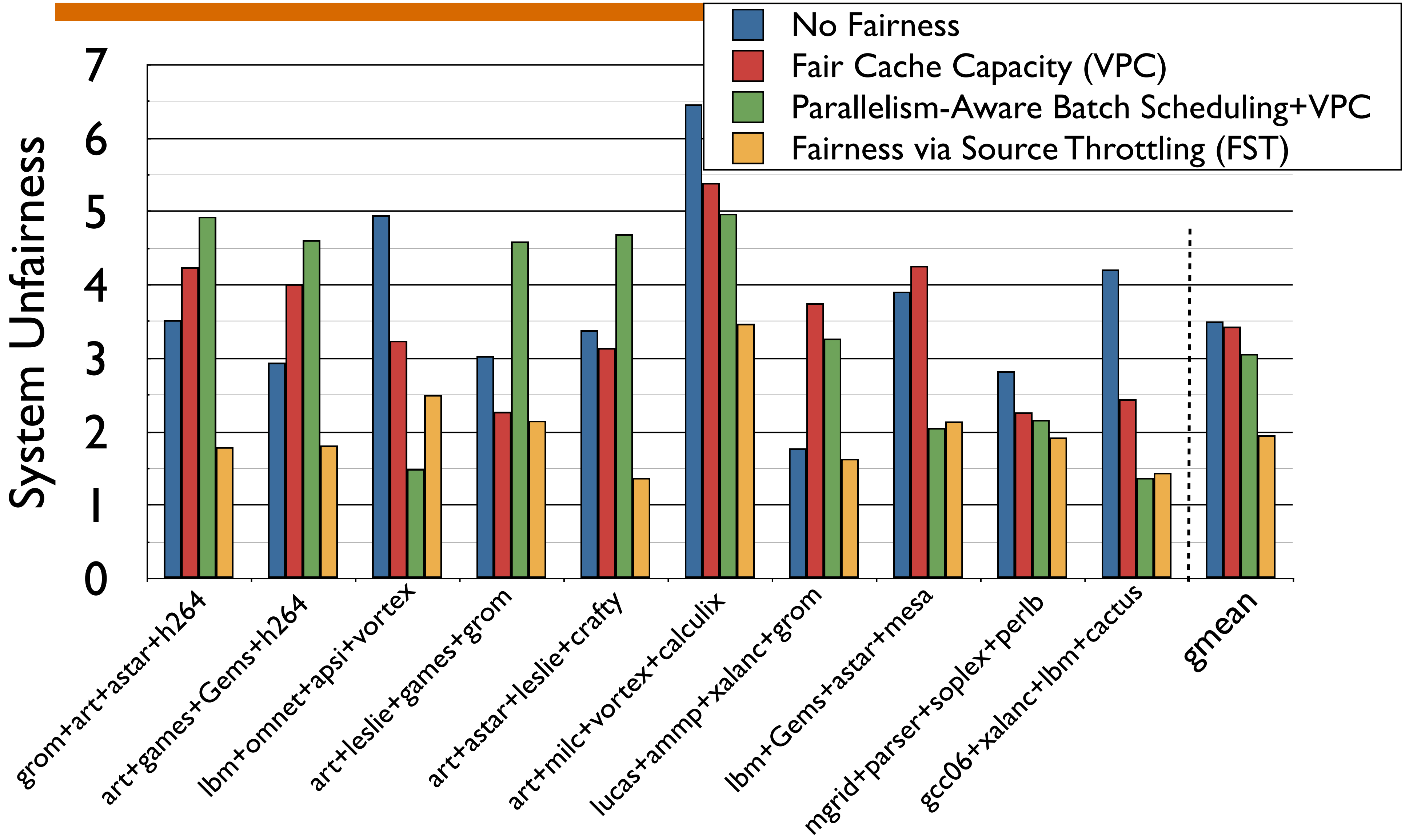
System Unfairness Results



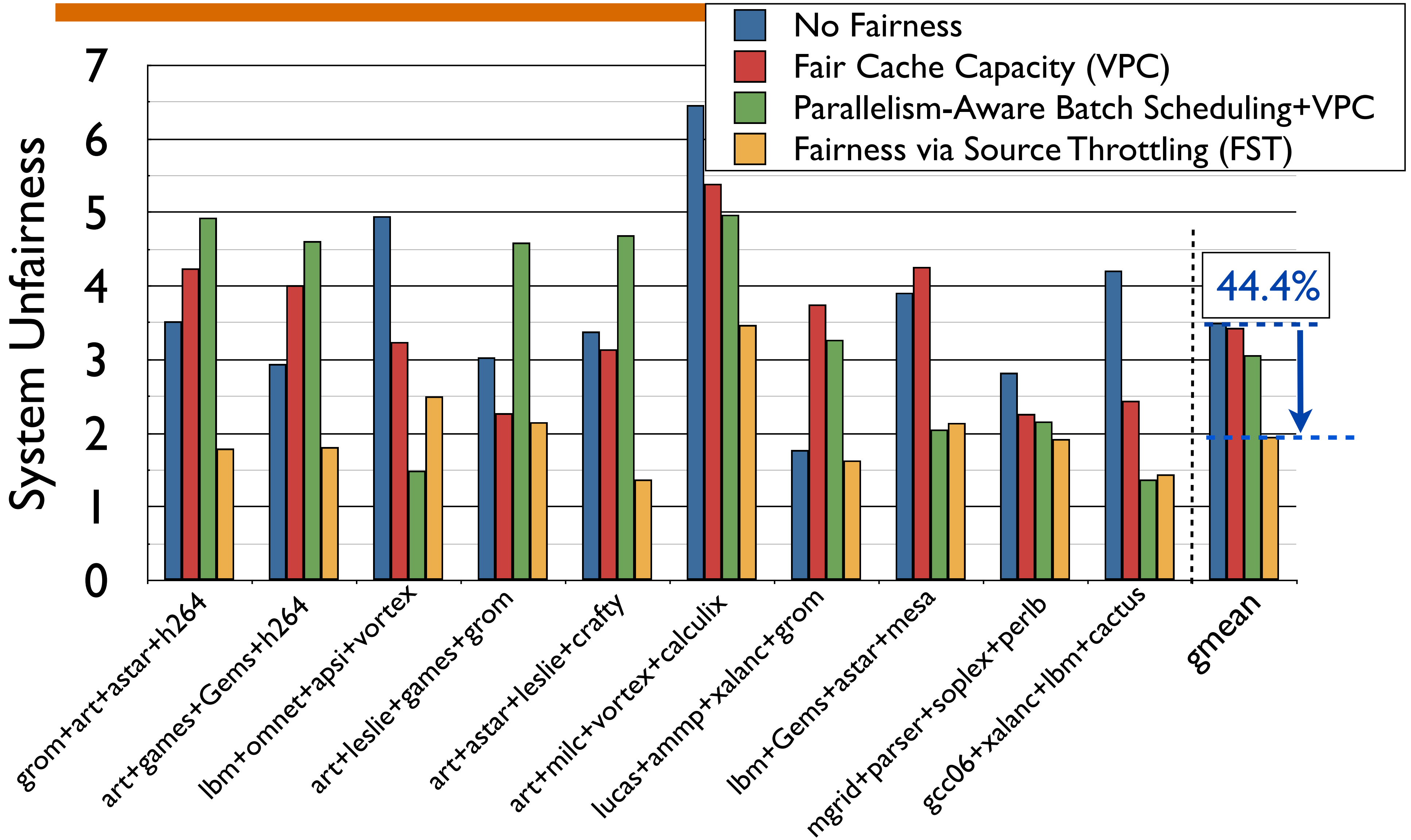
System Unfairness Results



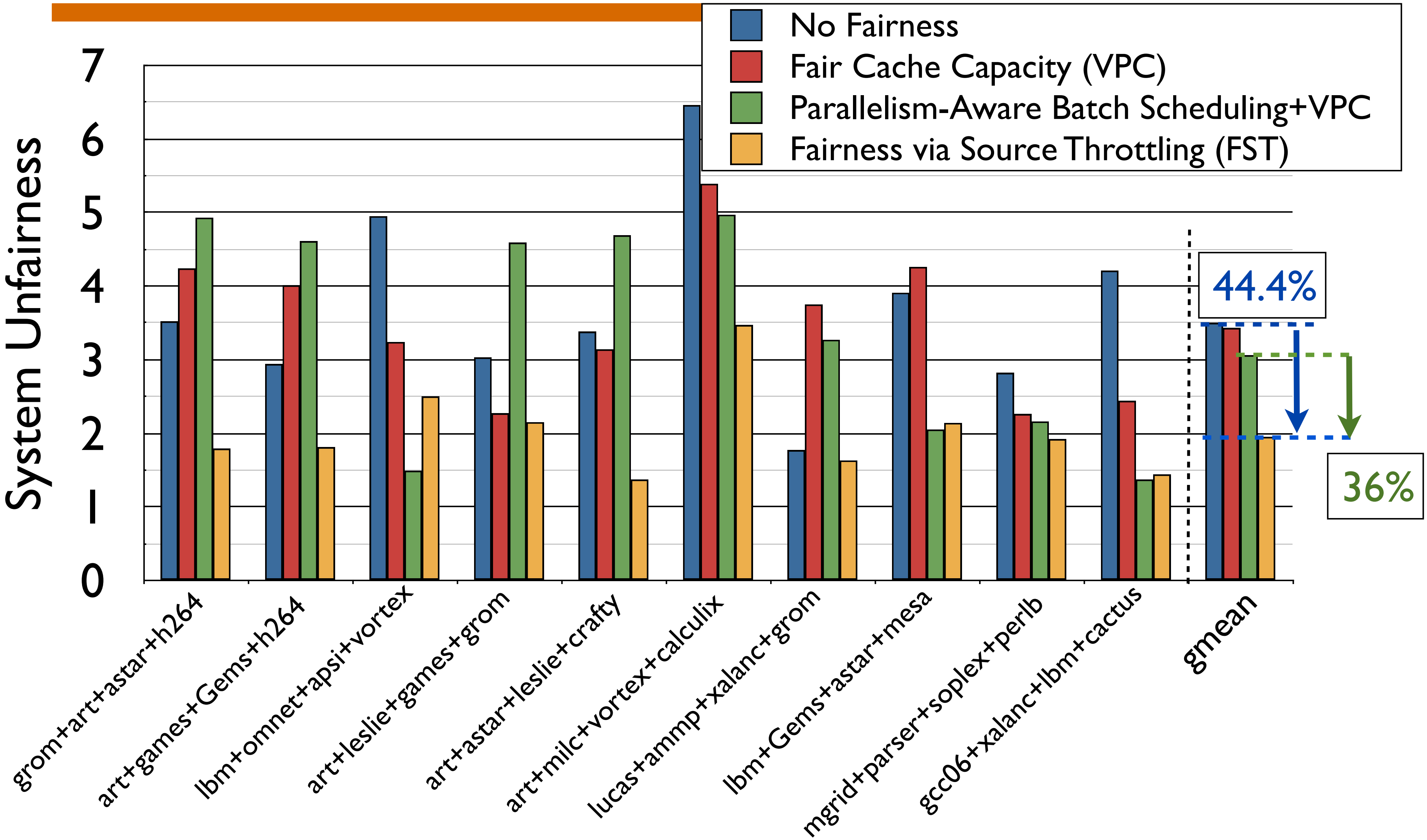
System Unfairness Results



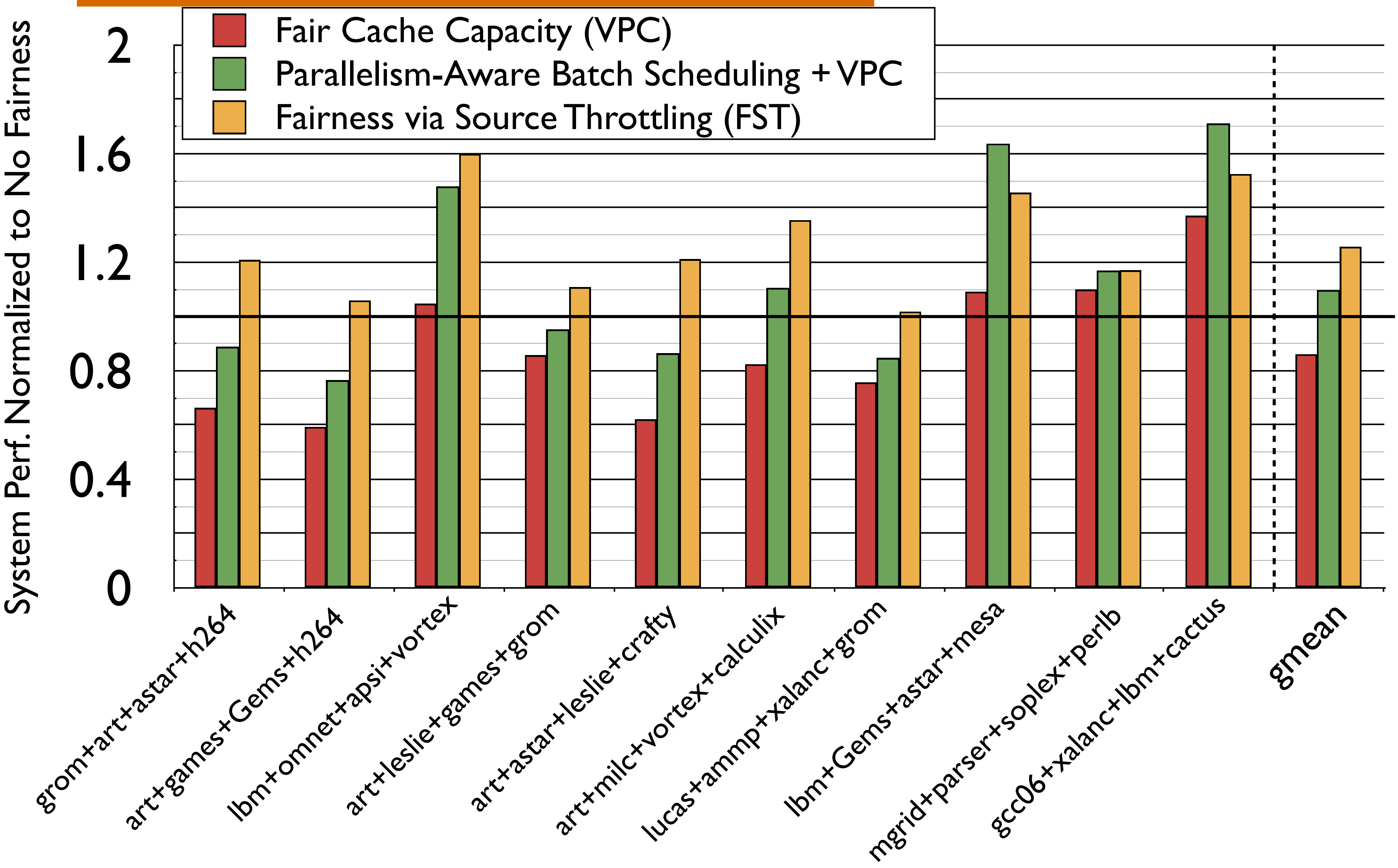
System Unfairness Results



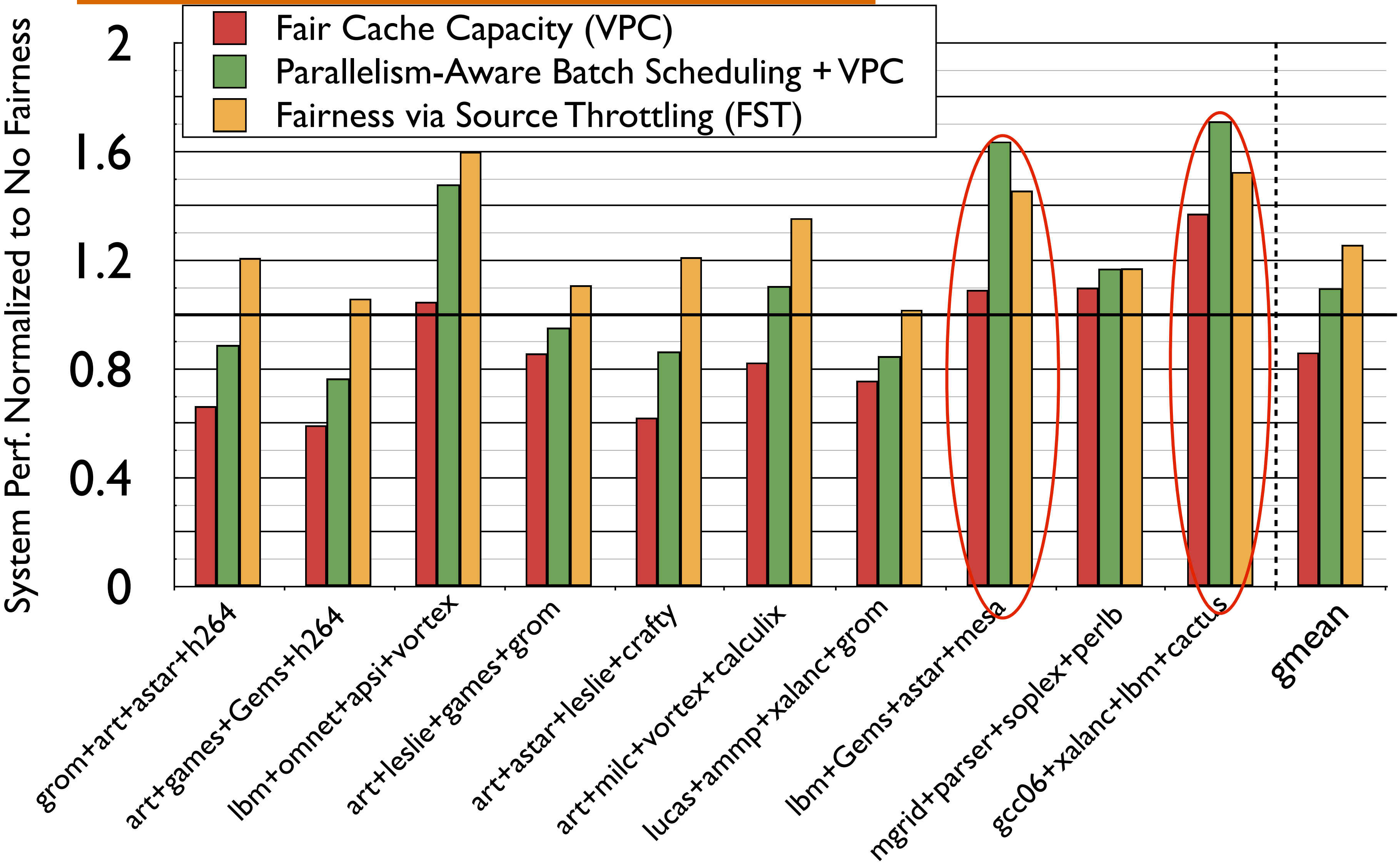
System Unfairness Results



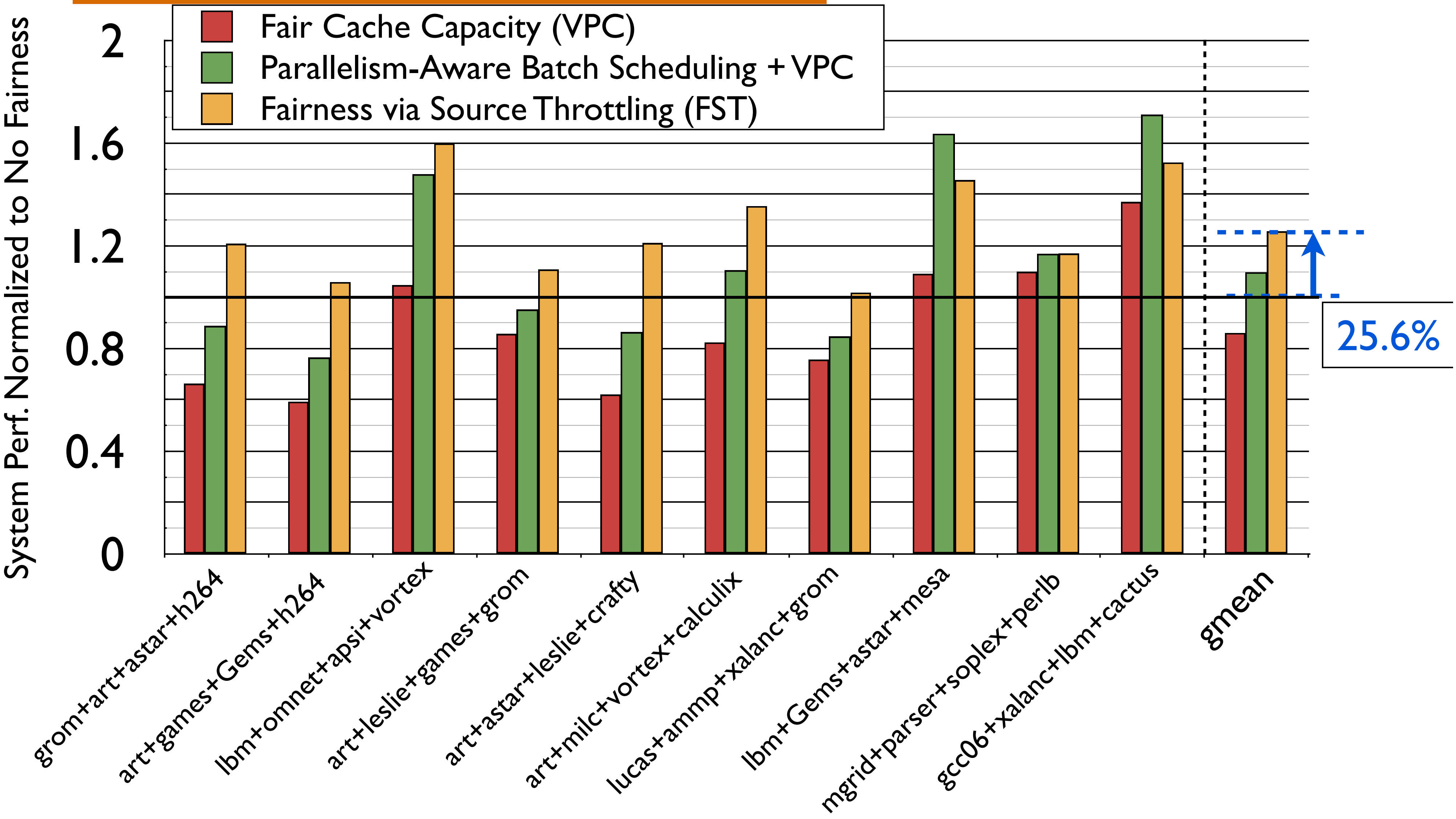
System Performance Results



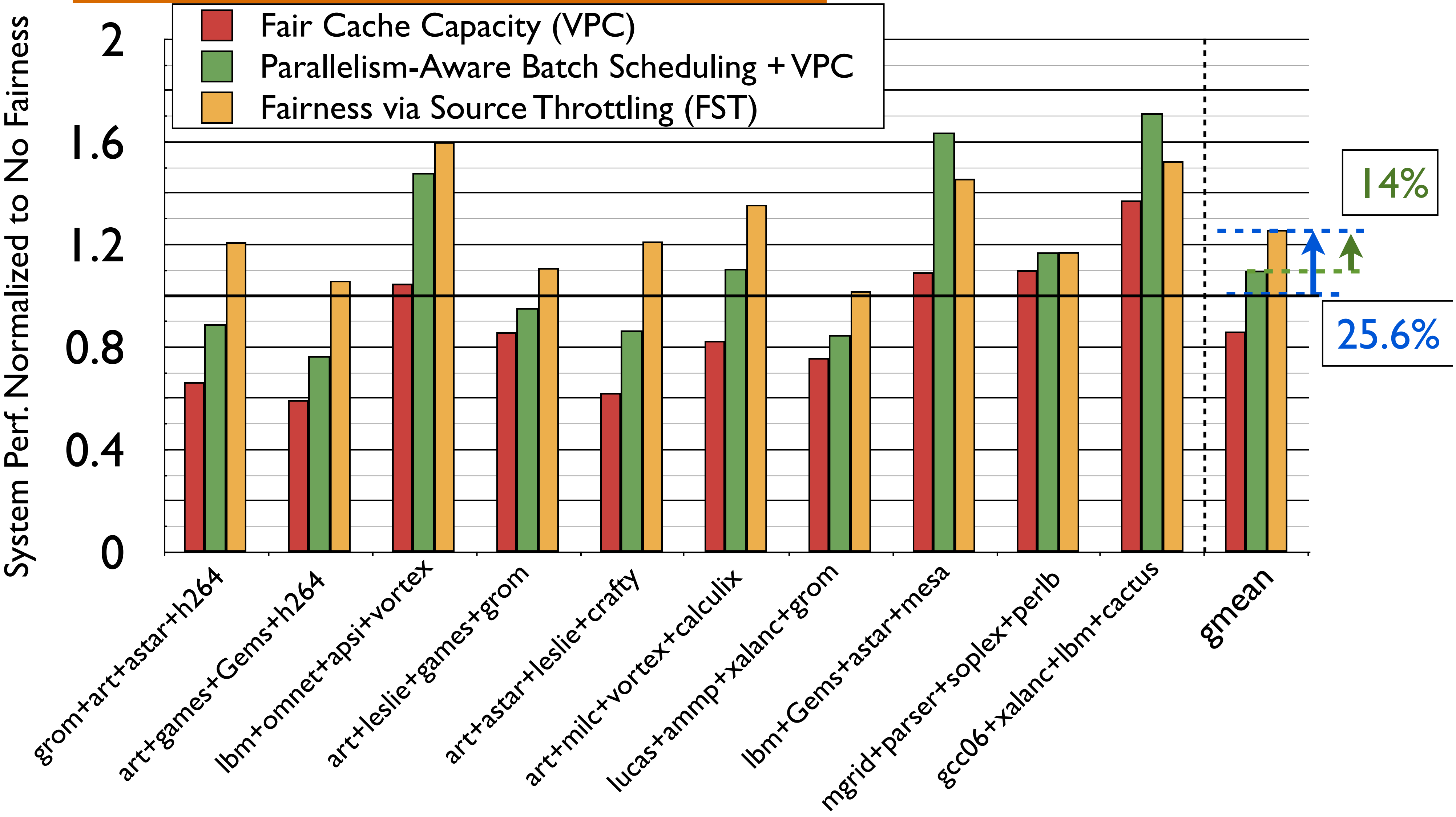
System Performance Results



System Performance Results



System Performance Results



Conclusion

- Fairness via Source Throttling (FST) is a new fair and high-performance shared resource management approach for CMPs
- Dynamically monitors unfairness and throttles down sources of interfering memory requests
- Eliminates the need for and complexity of multiple per-resource fairness techniques
- Improves both system fairness and performance
- Incorporates thread weights and enables different fairness objectives

Fairness via Source Throttling:

A configurable and high-performance fairness substrate for multi-core memory systems

Eiman Ebrahimi[*]

Chang Joo Lee[*]

Onur Mutlu[‡]

Yale N. Patt[*]

[*] HPS Research Group
The University of Texas at Austin

[‡] Computer Architecture Laboratory
Carnegie Mellon University