CISTER SEMINAR SERIES

Analyzing the Contention on the shared memory bus for COTS-Based Multicores

Presented by

Dakshina Dasari

29 April 2011

Agenda

- Motivation
- Problem of shared low level resources
- Proposed Method
- Future work

COTS-based Multicores

Increasingly used in embedded systems

- Low power, high computing capabilities
- Faster to design and market

Finding WCET in multicores difficult

- COTS: Undocumented parameters
- COTS: Not predictable
- Shared resource contention (low-level)
- Uniprocessor theories developed not applicable

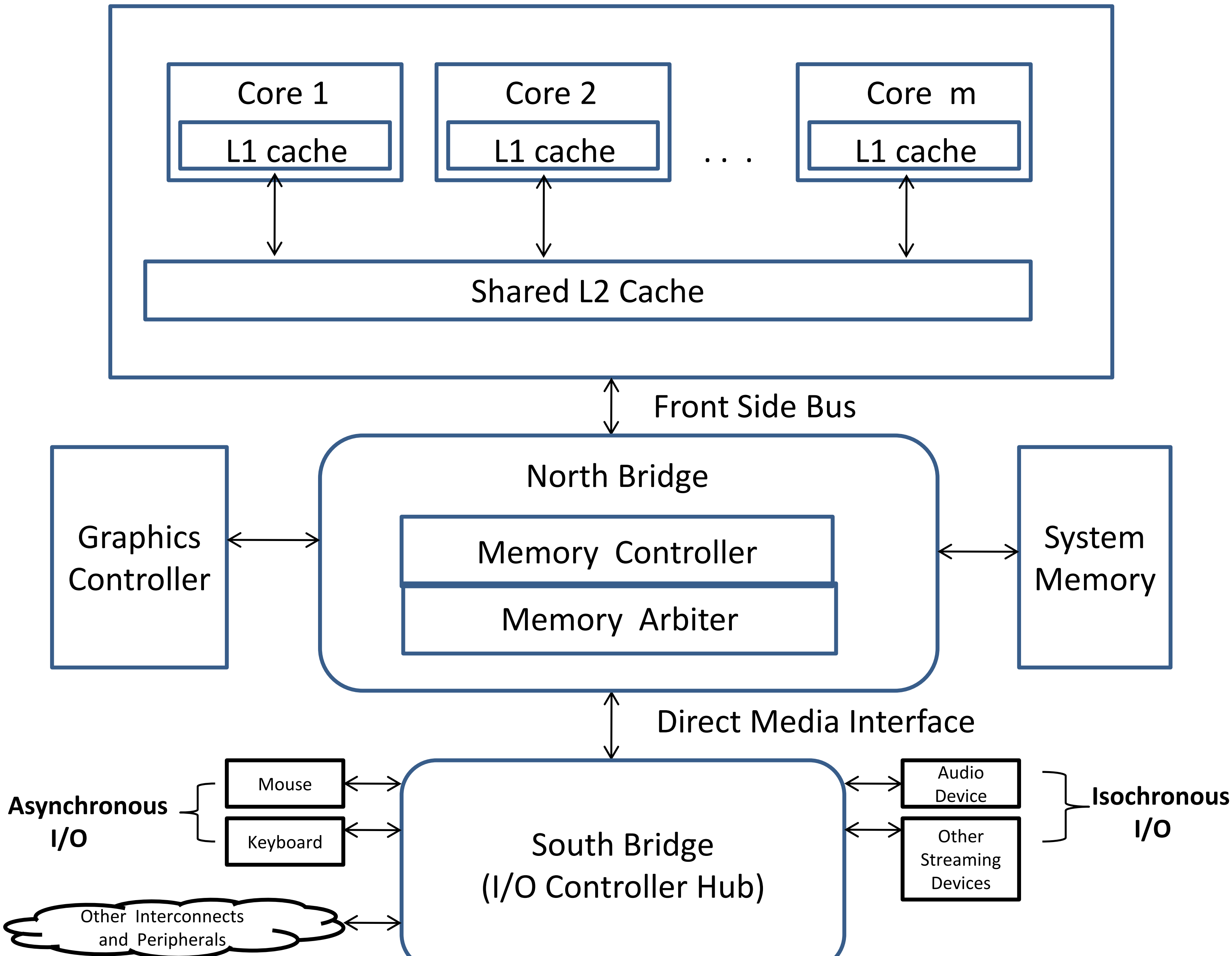
COTS: Commercial-off -the -shelf

Implications

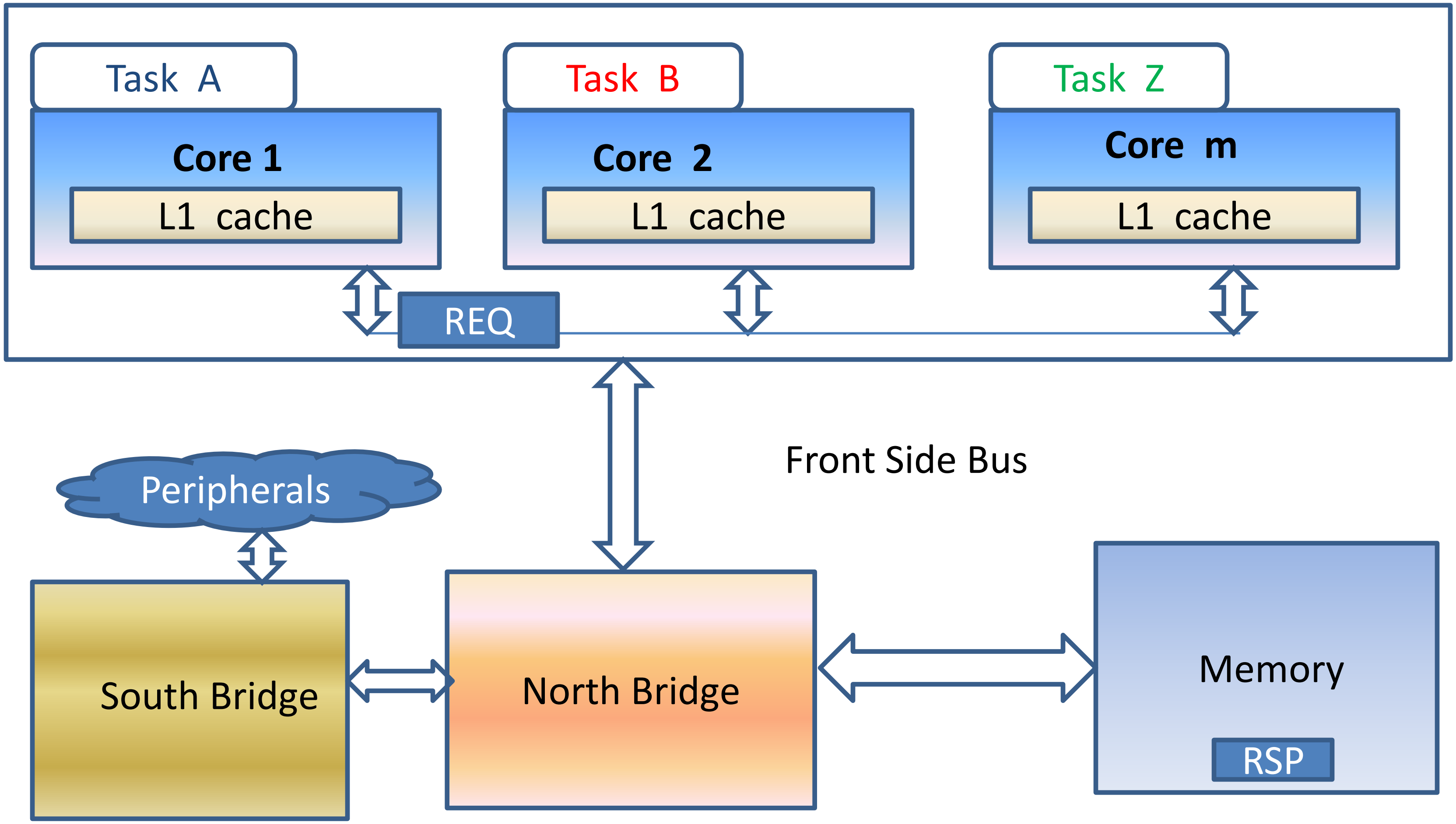
- Usage of very simple models in research
  - Do not reflect underlying hardware
- Generalized assumptions
  - Non tight WCET estimates

The industry trend does not seem to be towards building predictable systems ☹

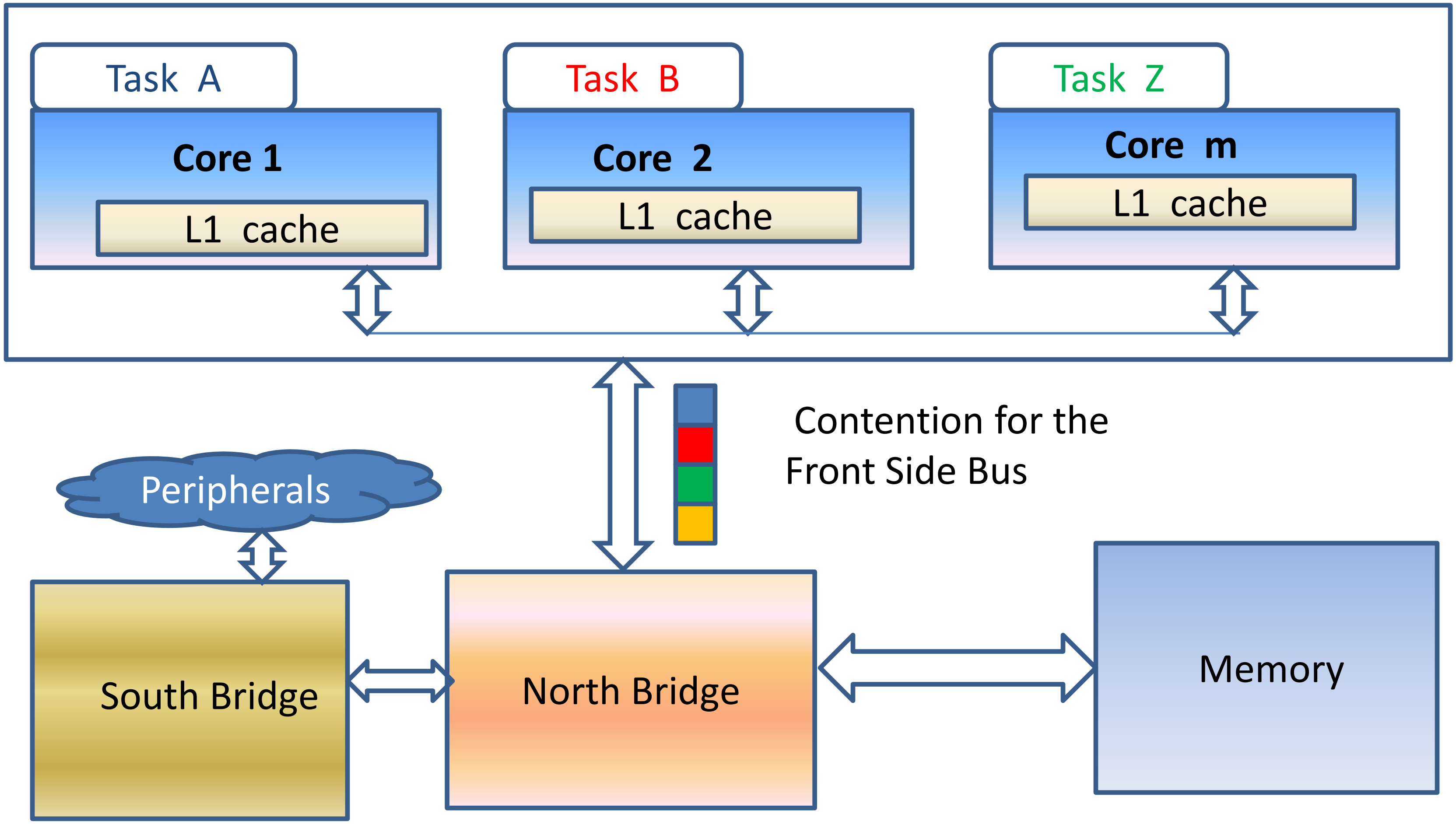
But performance oriented systems


Core 1
L1 cache
Core 2
L1 cache
. . .
Core m
L1 cache
Shared L2 Cache
Front Side Bus
North Bridge
Memory Controller
Memory Arbiter
Graphics Controller
System Memory
Direct Media Interface
Mouse
Keyboard
Asynchronous I/O
Other Interconnects and Peripherals
South Bridge
(I/O Controller Hub)
Audio Device
Other Streaming Devices
Isochronous I/O

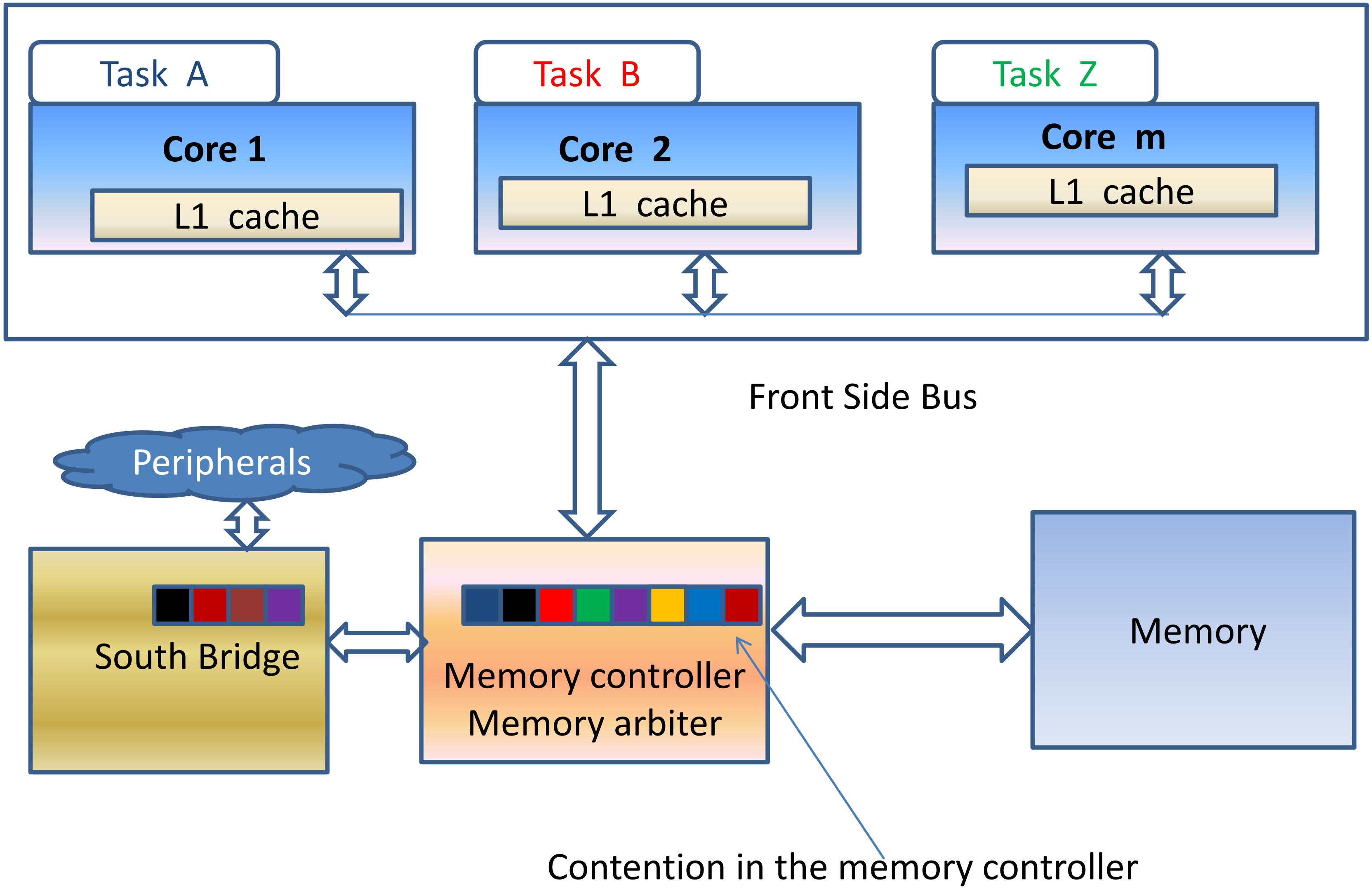
Shared resource contention



Shared resource contention



Shared resource contention



Nondeterminism in computing accurate WCET

- Total Time for a request =

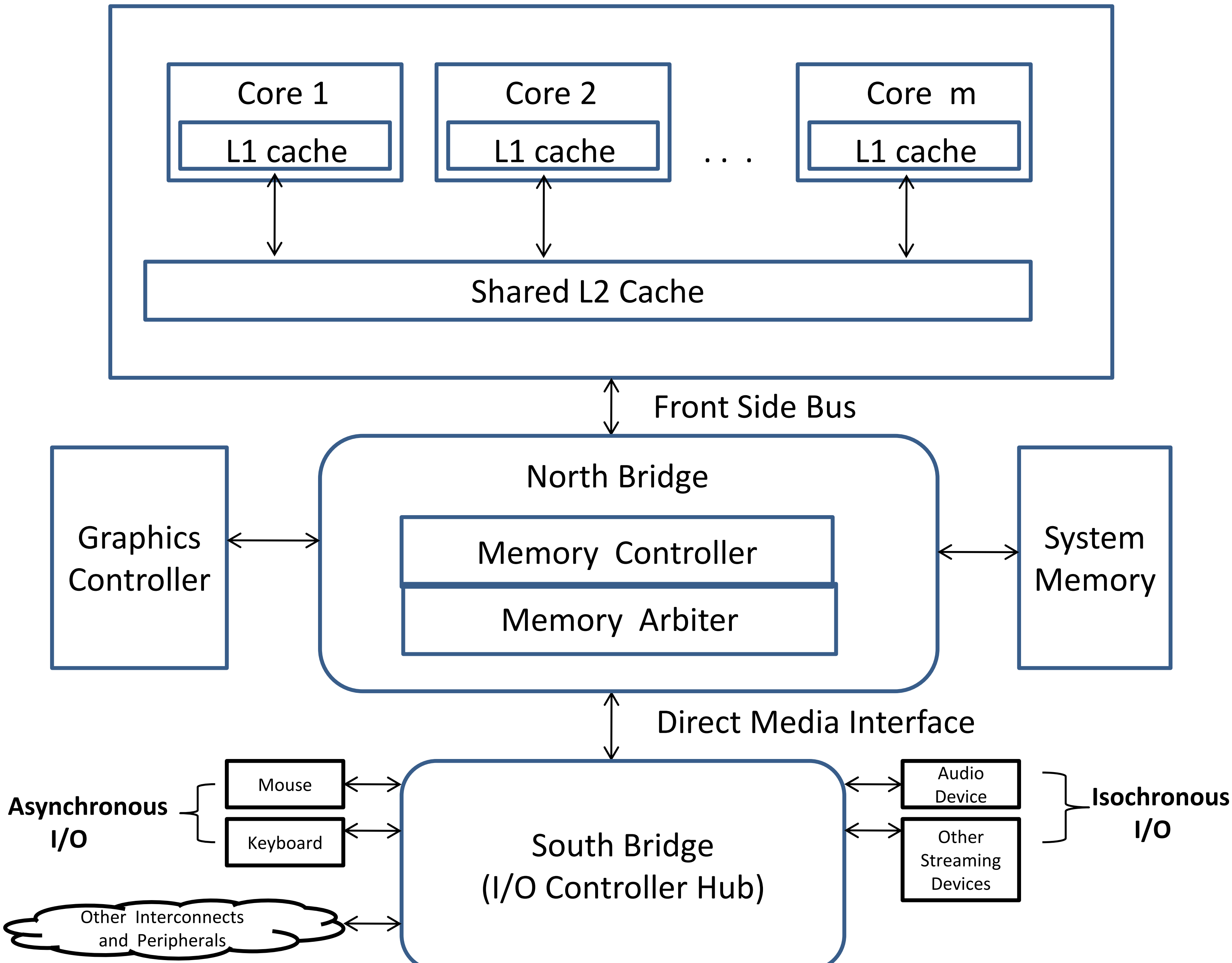
  T_FSB + // FSB contention
  T_FSB_NB + // transmission over FSB
  T_NB // NB contention
  T_NB_MEM + // tx time
  T_MEM + //memory access time
  T_MEM_NB + // tx time
  T_NB_FSB // tx time

Nontrivial to accurately determine : T_FSB, T_NB and T_MEM

Issues

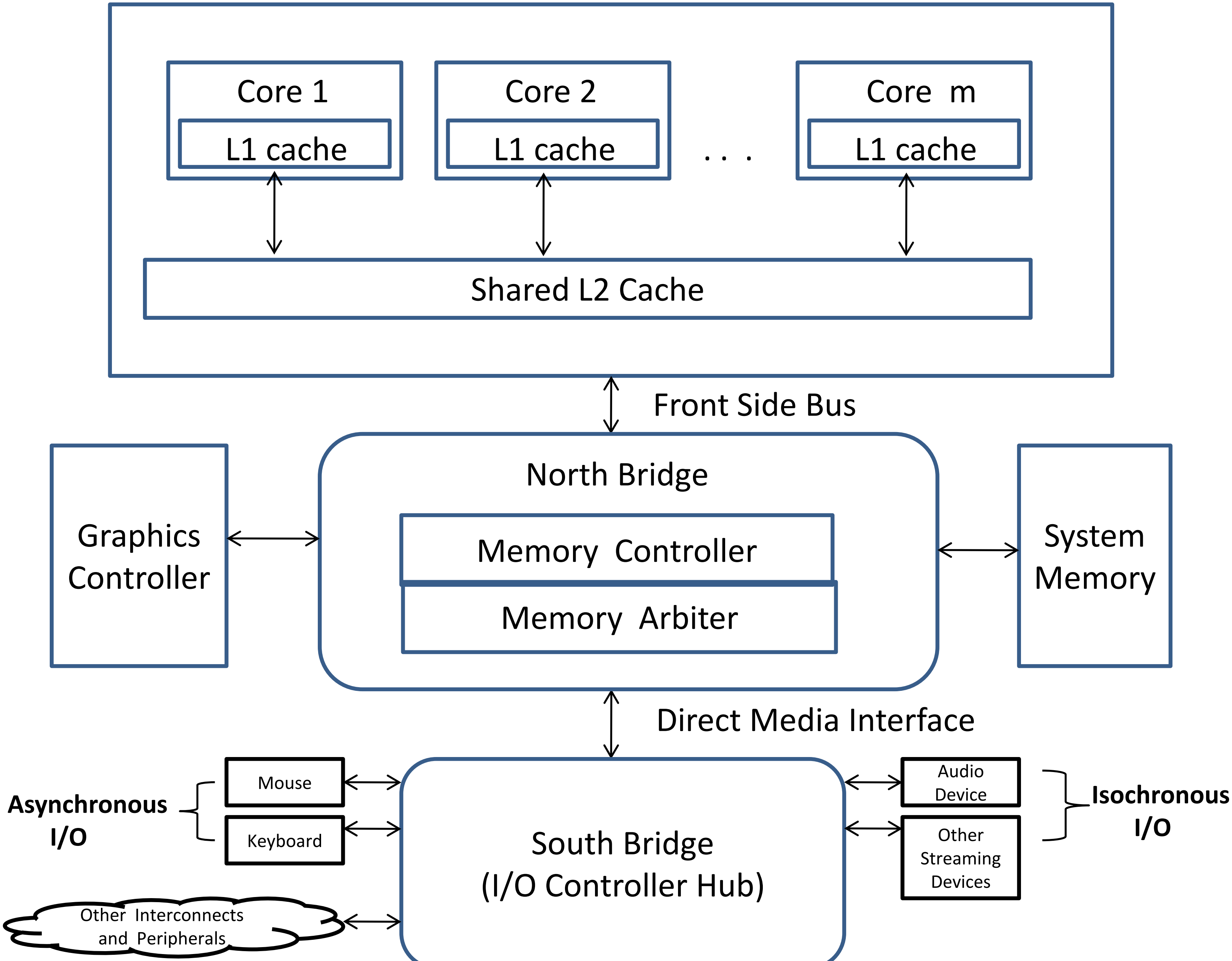
Non-accuracy due to some undocumented parameters :

- Size of buffers in NB not stated
- Arbitration algorithm in the NB is vendor proprietary
- Memory access time is variable for each request and dependent on memory access scheduling techiques


Core 1
L1 cache
Core 2
L1 cache
. . .
Core m
L1 cache
Shared L2 Cache
Front Side Bus
North Bridge
Memory Controller
Memory Arbiter
Graphics Controller
System Memory
Direct Media Interface
Asynchronous I/O
Mouse
Keyboard
Other Interconnects and Peripherals
South Bridge (I/O Controller Hub)
Audio Device
Other Streaming Devices
Isochronous I/O

Contention in the FSB

- Resolved using a Round Round Algorithm
  - (Disclaimer : wrt to intel processors)
  - Fairness : Order of transmission is fixed apriori (1-2-3-4-1)
  - Bus owner parks onto the bus until other owners assert the bus-request line
    - To Reduce switching overhead
  - Non-idling: A bus owner can keep transmitting when other cores do not transmit


Core 1
L1 cache
Core 2
L1 cache
. . .
Core m
L1 cache
Shared L2 Cache
Front Side Bus
North Bridge
Memory Controller
Memory Arbiter
Graphics Controller
System Memory
Direct Media Interface
Mouse
Keyboard
Asynchronous I/O
Other Interconnects and Peripherals
South Bridge
(I/O Controller Hub)
Audio Device
Other Streaming Devices
Isochronous I/O

Contention in the North Bridge

Request Type	Service slots (system cycles)
DRAM Maintenance Requests (Refresh)	X (High Priority = 1)
Display (Isochronous)	Y
Streaming (Isochronous)	Z
CPU (Asynchronous)	W
	Total = N system cycles

- Schedule period repeats after every N cycles
- Flexible, Slot based mechanism
- Tries to meet QoS requirements of Isochronous (Periodic requests) with low-latency requirements of Asynchronous requests

Contention in the North Bridge

Request Type	Service slots (system cycles)
DRAM Maintenance Requests (Refresh)	X (High Priority = 1)
Display (Isochronous)	Y
Streaming (Isochronous)	Z
CPU (Asynchronous)	W
	Total = N system cycles

- Flexible but non-predictable
- Weights assigned to request types not specified
- Difficult to accurately compute an upper-bound

Nondeterminism in computing accurate WCET

- Total Time for a request =

  T_FSB + // FSB contention
  T_FSB_NB + // transmission over FSB
  T_NB // NB contention
  T_NB_MEM + // tx time
  T_MEM + //memory access time
  T_MEM_NB + // tx time
  T_NB_FSB // tx time

Nontrivial to accurately determine : T_FSB, T_NB and T_MEM

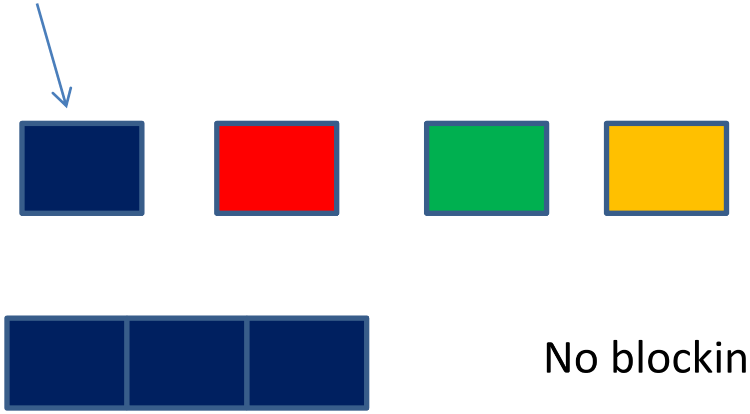
Summary of the discussion

- Method to obtain maximum time to service a request (TR) by adding individual factors difficult
- Workaround :
  - Measure end to end latency for a large number of requests
  - Record the maximum value
  - Use this value for WCET estimation

Problem Definition

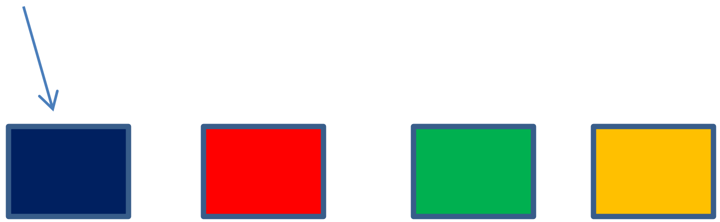
- Compute the WCET of a task, considering contention on the bus on , given the following :
- WCET in isolation
- A multicore system with
  - Private caches : Cores do not share the cache
  - Shared front side bus with **Round Robin Bus Arbitration Algorithm**
- Task model
  - Non pre-emptive (Tasks run uninterrupted)
    - No Cache Related Pre-emption Delay and context switch overhead
  - Constrained deadline ($D_i <= T_i$) Periodic tasks
  - Partitioned scheduling (Tasks do not migrate)

Round Robin algorithm



No blocking from other tasks

Round Robin algorithm



No blocking from other tasks

$$C_i^{mix} = C_i^{iso}$$

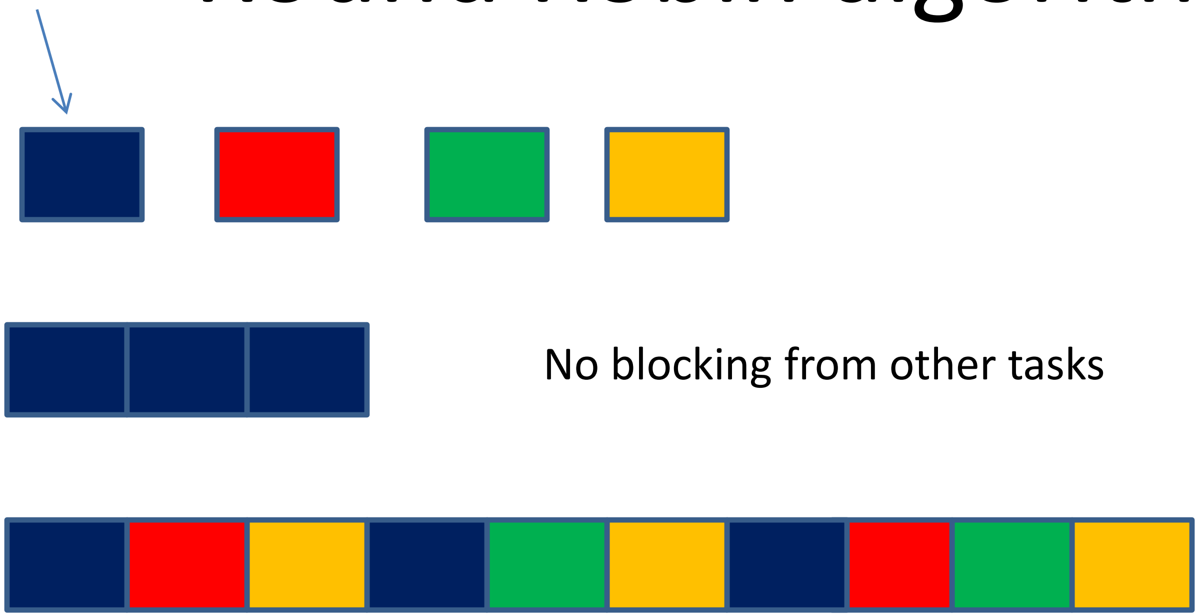
where

TR: Time to serve a request

C_i^{iso} :WCET in isolation

C_i^{mix} :WCET when run with other tasks

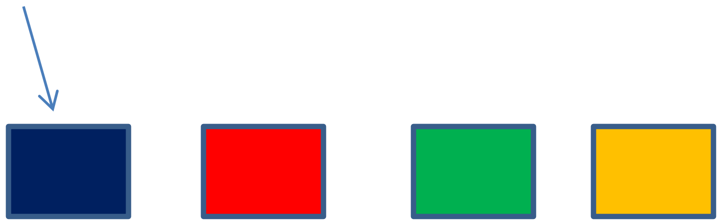
Round Robin algorithm



No blocking from other tasks

Blocked by 7 requests

Round Robin algorithm



Blocked by 7 requests

$$\mathbf{C_i^{mix} = C_i^{iso} + 7 * TR}$$

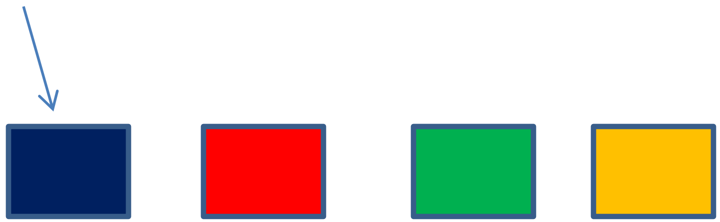
where

TR: Time to serve a request

C_i^{iso} :WCET in isolation

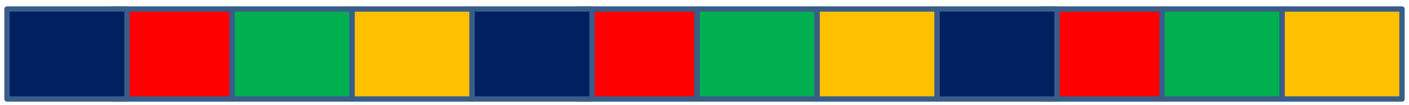
C_i^{mix} :WCET when run with other tasks

Round Robin algorithm



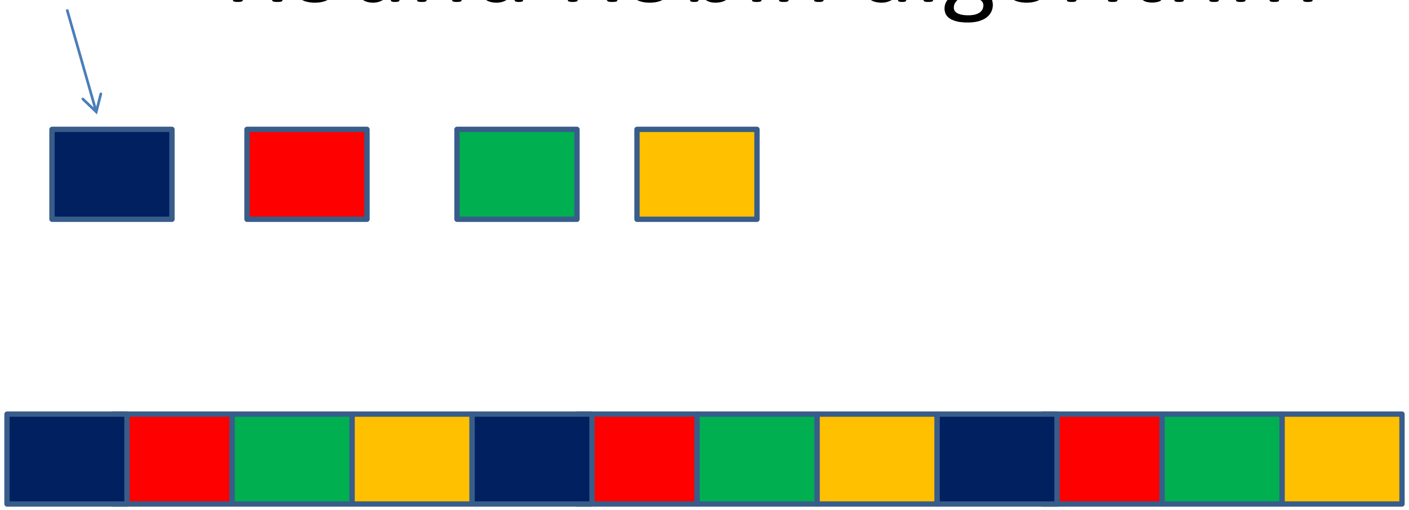
No blocking from other tasks

Blocked by 7 requests



Worst case Blocked by 9 requests

Round Robin algorithm



Worst case Blocked by 9 requests

$$C_i^{mix} = C_i^{iso} + 9 * TR$$

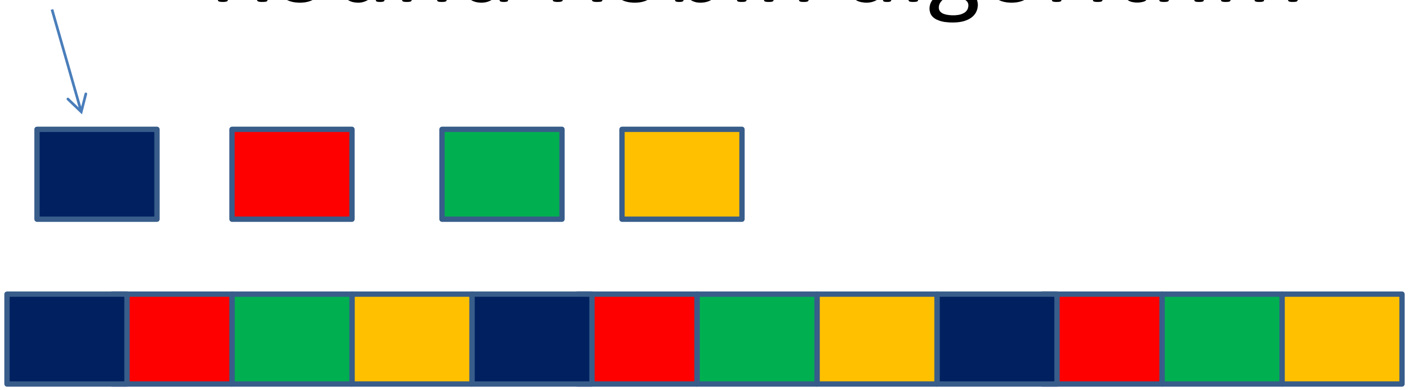
where

TR: Time to serve a request

C_i^{iso} : WCET in isolation

C_i^{mix} : WCET when run with other tasks

Round Robin algorithm



Worst case Blocked by 9 requests

$$C_i^{mix} = C_i^{iso} + 9 * TR$$

where

TR: Time to serve a request

C_i^{iso} : WCET in isolation

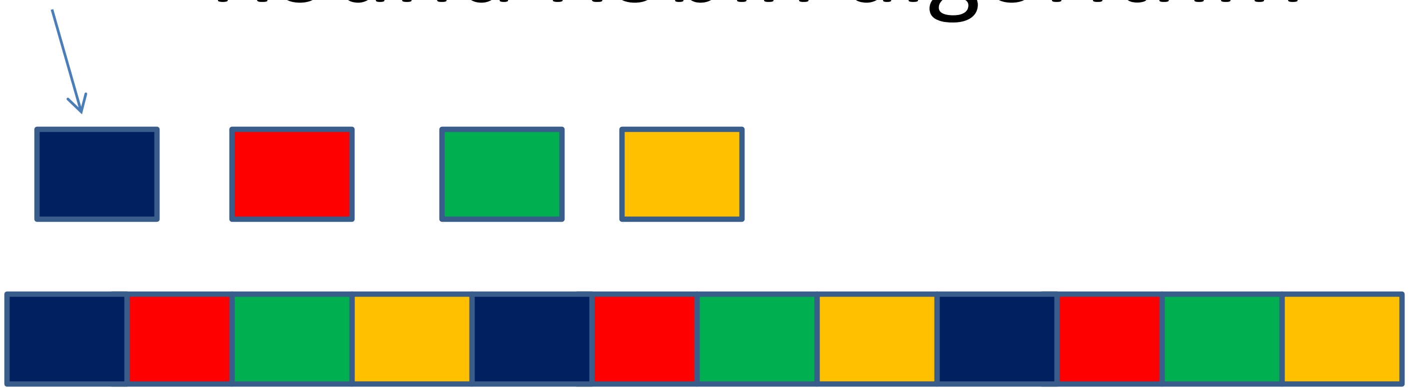
C_i^{mix} : WCET when run with other tasks

m : number of cores

$RQST_i(t)$: Requests generated by task i in time t

$$C_i^{mix} = C_i^{iso} + RQST_i(C_i^{iso}) * (m-1) * TR$$

Round Robin algorithm



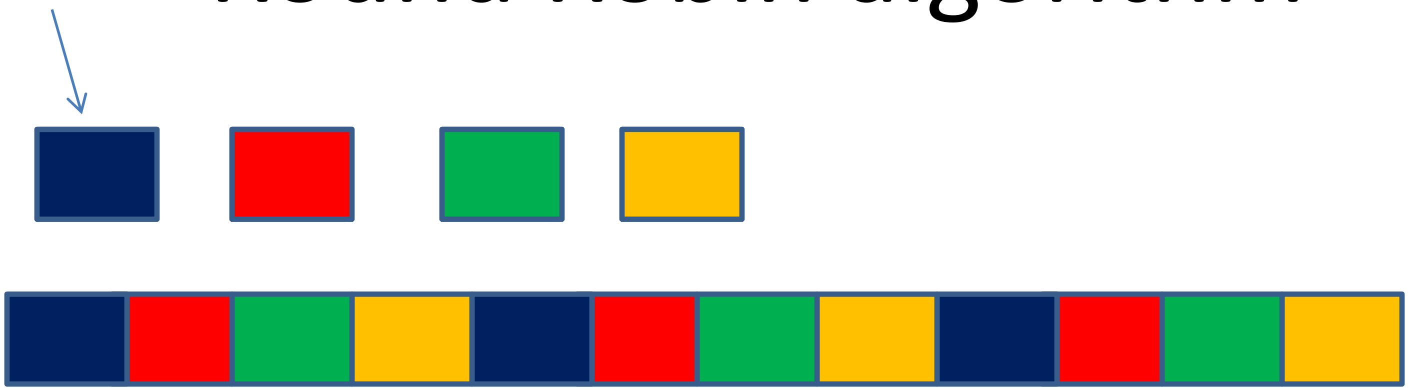
Worst case Blocked by #Max = $(m-1) * RQST_i(C_i^{iso})$ requests

$C_i^{mix} = C_i^{iso} + RQST_i(C_i^{iso}) * (m-1) * TR$

Very pessimistic !!

- Tasks on other cores may not generate #Max requests
- There may be no tasks scheduled on the other cores

Round Robin algorithm



Worst case Blocked by Max = $(m-1) * RQST_i(C_i^{iso})$ requests

$$C_i^{mix} = C_i^{iso} + RQST_i(C_i^{iso}) * (m-1) * TR$$

Very pessimistic !!

- Ex : Task i generates 2000 requests $RQST_i(C_i^{iso}) = 2000$
- Co-scheduled tasks on other cores generate 20 requests
- By the bound $C_i^{mix} = C_i^{iso} + 2000 * (3) * TR$
- **Actual :** $C_i^{mix} = C_i^{iso} + 20 * TR$

Round Robin algorithm

Pessimistic bound:

$$C_i^{mix} = C_i^{iso} + RQST_i(C_i^{iso}) * (m-1) * TR$$

For tighter WCET bounds we need:

$$C_i^{mix} = C_i^{iso} + \text{Requests\_from\_other\_cores} * TR$$

(during execution of task i)

We need a **Per-Core Request Estimator Function**

Round Robin algorithm

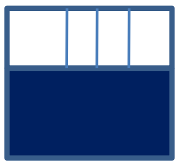
C_i^{mix} = C_i^{iso} + Requests_from_other_cores * TR (during execution of task i)

We need a **Per-Core Request Estimator Function**

$PCRE_j(t)$: Returns maximum number of requests generated by tasks scheduled on core 'j' during time interval 't'

The Method

Task A



ADVERSARY

Core 1

Core 2 Core3 Core4

Interference queue of length = (m-1) * $RQST_A(\mathbf{C_A^{iso}})$ = 3 * 3 = 9 slots

At time 0

$PCRE_2(0) = PCRE_3(0) = PCRE_4(0) = 0$

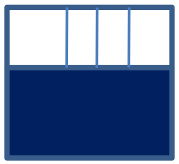
m = 4 cores

$\mathbf{C_A^{iso} = 4}$ TR = 0.05

$\mathbf{C_A^{mix} = 4}$

The Method

Task A



ADVERSARY

Core 1 Core 2 Core3 Core4

Interference queue of length = (m-1) * $RQST_A(\mathbf{C_A^{iso}})$ = 3 * 3 = 9 slots

At time 0

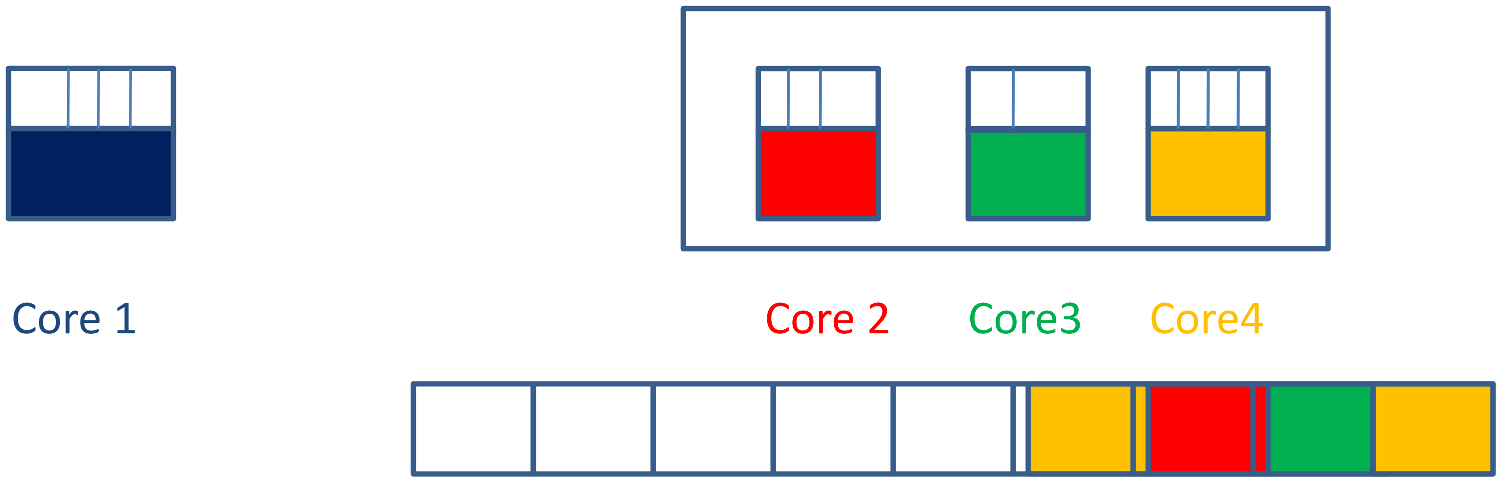
$PCRE_2(0) = PCRE_3(0) = PCRE_4(0) = 0$

m = 4 cores

$\mathbf{C_A^{iso} = 4}$ TR = 0.05

$\mathbf{C_A^{mix} = 4}$

The Method



At time = 4

$PCRE_2(4)$ = 1 $PCRE_3(4)$ = 1 $PCRE_4(4)$ = 2

m = 4 cores

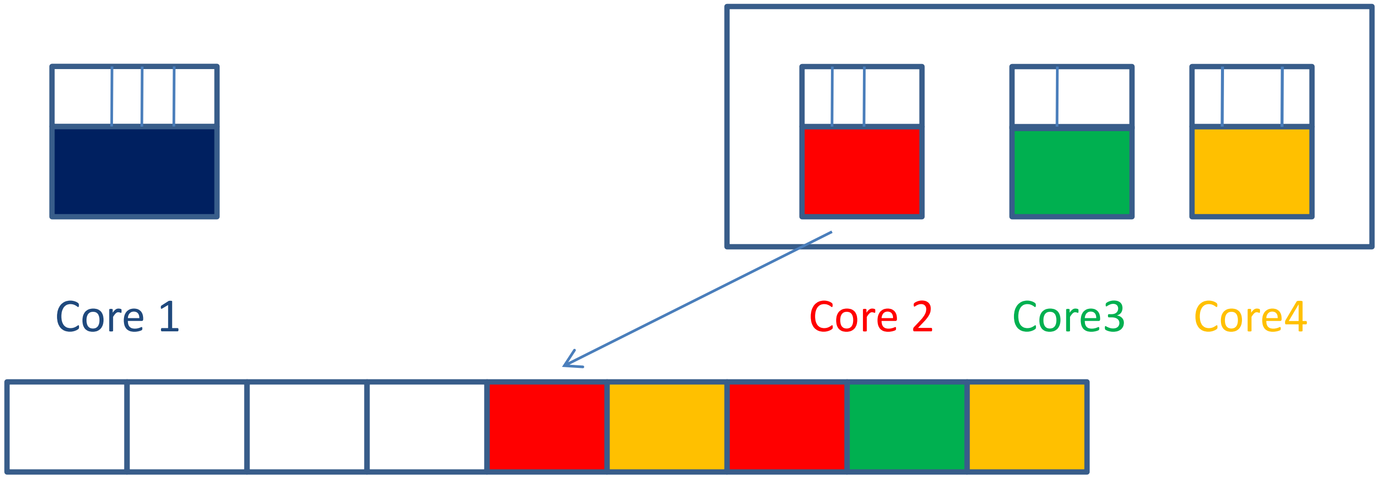
C_A^{iso} = 4 TR = 0.05

C_A^{mix} = 4 + 4 *0.05

= 4.20

Execution time of task A increases

The Method



At time = 4.20

Δ : Increased execution time

$PCRE_2(\Delta) = 0$ $PCRE_3(\Delta) = 1$ $PCRE_4(\Delta) = 0$

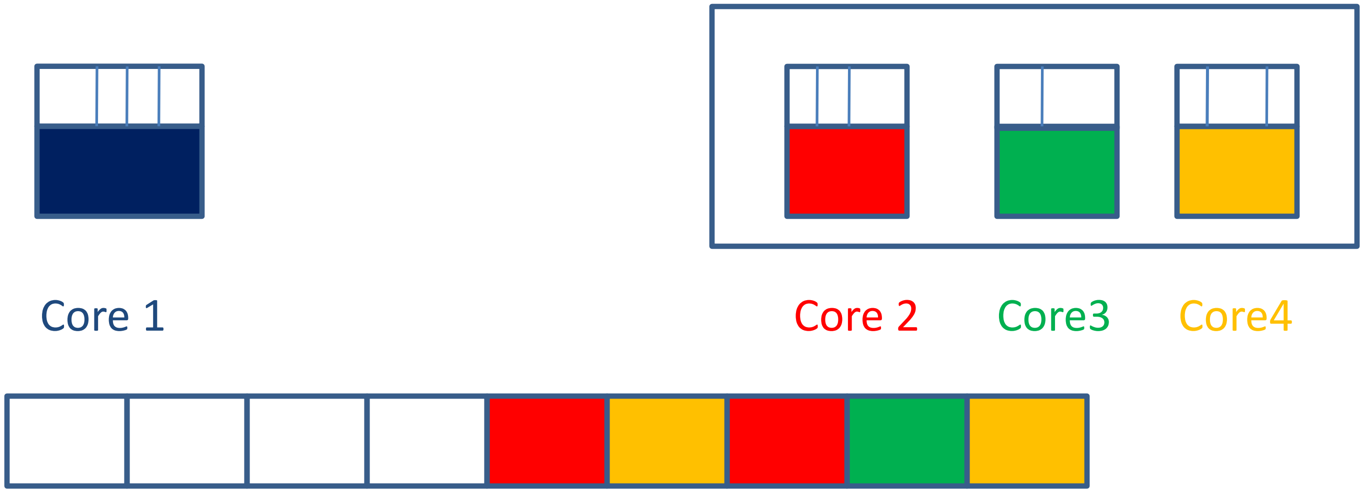
$\mathbf{C_A^{iso} = 4}$ TR = 0.05

$\mathbf{C_A^{mix} = 4.20 + 1 * 0.05}$

$\mathbf{= 4.25}$

Execution time of task A further increases

The Method



At time = 4.25

Δ : Increased execution time

$PCRE_2(\Delta) = 0$ $PCRE_3(\Delta) = 0$ $PCRE_4(\Delta) = 0$

No more requests from other cores !!!

$\mathbf{C_A^{iso} = 4}$ TR = 0.05

$\mathbf{C_A^{mix} = 4.25 + 0}$

$\mathbf{= 4.25}$

Final wcet = 4.25

The algorithm

Initialization Step

$C_i^0 = C_i^{iso}$

$iqlen_i^0 = RQST_i(C_i^0) * (m-1)$

$external\_rqst_i^0 = \sum_{j \neq \pi(i)} PCRE_j(C_i^0)$

$blocking\_rqst_i^0 = \min(iqlen_i^0, external\_rqst_i^0)$

Iteration Step

$C_i^k = C_i^{k-1} * blocking\_rqst_i^{k-1} * TR$

$iqlen_i^k = iqlen_i^{k-1} - blocking\_rqst_i^{k-1}$

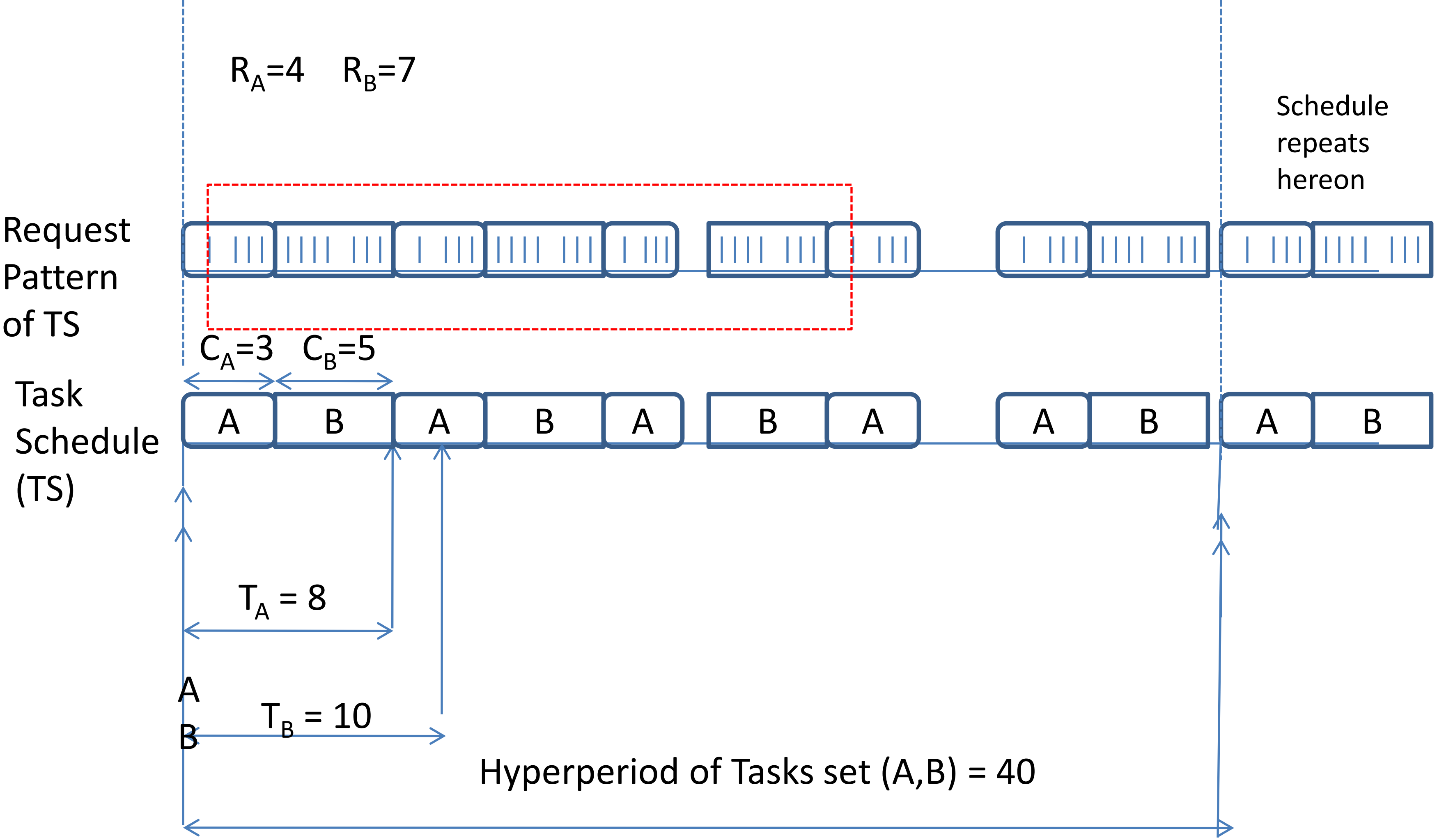
$external\_rqst_i^k = \sum_{j \neq \pi(i)} (PCRE_j(C_i^k) - PCRE_j(C_i^{k-1}))$

$blocking\_rqst_i^k = \min(iqlen_i^{k-1}, external\_rqst_i^{k-1})$

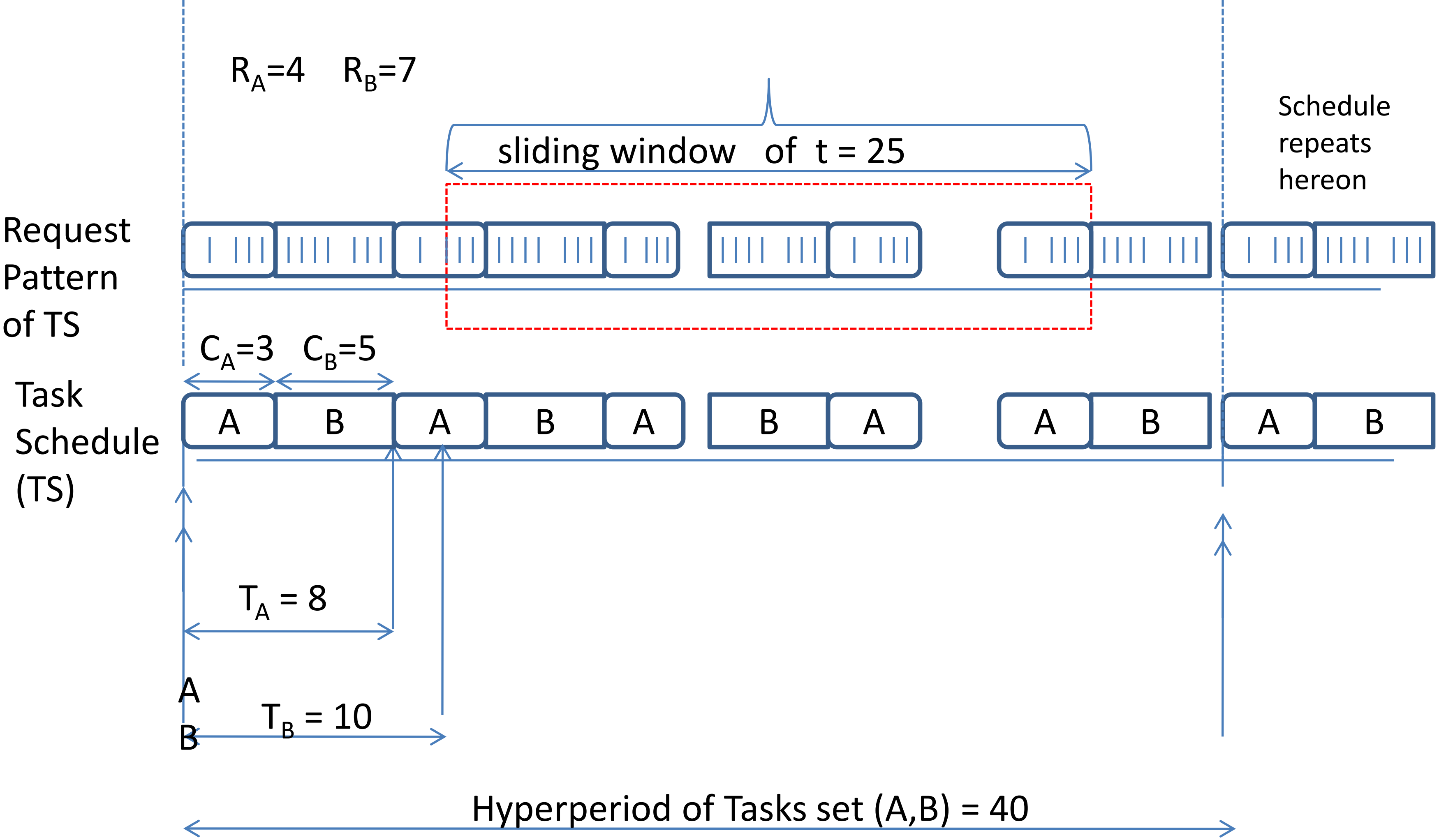
Stopping Conditions :

$iqlen_i^k = 0$ RR Upper bound reached

$blocking\_rqst_i^k = 0$ No more requests from other cores



Per Core Request Estimator



Per Core Request Estimator

Obtaining request patterns

- PCRE(t) depends on the
  exact request pattern of the tasks
  - **Measurements**
    - Performance monitoring counters
    - Special purpose registers in microprocessors

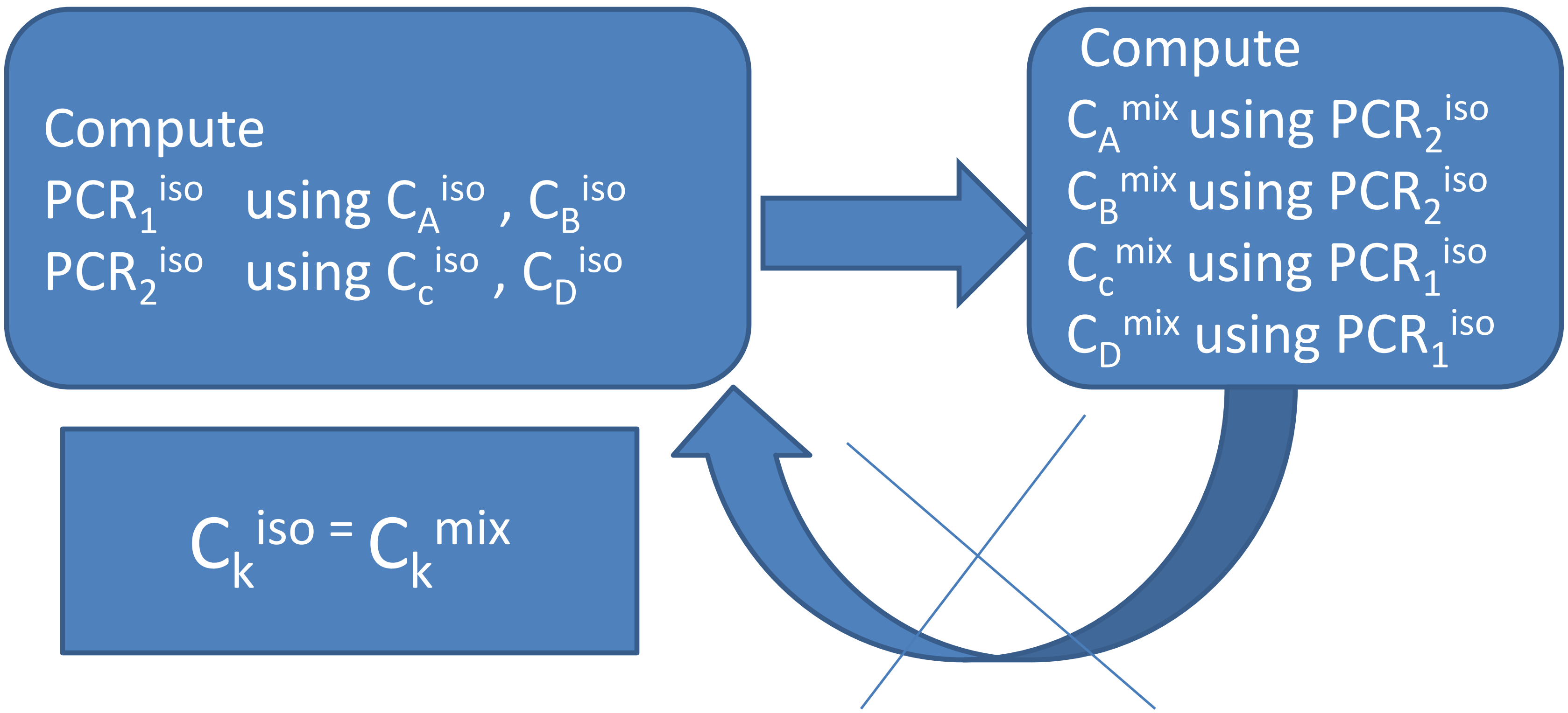
      Reset counter, select event (like L1 cache misses)

      Code block to be monitored

      Stop counter, Read values
  - Static analysis

System wide analysis

Tasks A, B assigned to core 1
Tasks C, D assigned to core 2



System wide analysis

Task A

Why:

Private caches
No extra cache misses

Non premptive tasks
No extra cache misses

Increase only due to bus contention

Increased execution does not increase number of requests generated

Request density decreases .
Therefore $PCR_i^{iso}(t) >= PCR_i^{mix}(t)$

System wide analysis

Tasks A, B assigned to core 1
Tasks C, D assigned to core 2

Compute
PCR_1^{iso} using C_A^{iso}, C_B^{iso}
PCR_2^{iso} using C_c^{iso}, C_D^{iso}

→

Compute
C_A^{iso} using PCR_2^{iso}
C_B^{iso} using PCR_2^{iso}
C_c^{iso} using PCR_1^{iso}
C_D^{iso} using PCR_1^{iso}

Other Related work carried out

- Response time analysis for an arbitration-agnostic bus contention algorithm for COTS-based systems
- Measurement based framework for generating a request profile for a task
  - Uses performance monitoring counters
- Preliminary shared cache analysis

Future Work

- Reducing Bus Contention with resource-aware schedulers
- Addressing contention considering shared caches
- Addressing Pre-emptive task models