A Predictable Execution Model for COTS-based Embedded Systems

Research by:

Rodolfo Pellizzoni, Emiliano Betti, Stanley Bak, Gang Yao, John Criswell, Marco Caccamo and Russell Kegley

Presented by:

Neda Paryab

Outline

- Problem Statement
- PREM system
- Evaluation
- Critiques

Intro.

Increasing usage of Commercial-Off-The-Shelf (COTS) components

Why (vs. Customized systems) ?

- Cheap → massive production
- General Purpose → flexible for different applications and not binded to a single SW/HW
- Less design defects → design for reuse (silver bullet? wrong assumptions - integration issues)
- Backward compatible with legacy products
- High performance

Problems?

- Not suitable for all applications → what about environmental constraints? such as temperature, radiation exposure and etc.

- Maybe not suitable for safety critical systems, such as flight control or medical equipment. Not Reliable?

COTS issues for real-time

The main drawback of using COTS components within a real-time system is the presence of unpredictable timing anomalies.

- Contentions due to initiating access shared resources (such as cache) by multiple active components (such as CPU cores and I/O peripherals)
  - Leads to timing degradation
  - Low-level arbiters of these shared resources are not typically designed to provide real-time guarantees
  - Also, each of active devices initiate their access requests independent (unaware) of each other

- Solution: compute precise bounds on worst-case timing delays caused by shared resource access contention.
  - How to do it in a realistic way?

Predictable Execution Model (PREM)

- Enforces a high-level co-schedule among CPU tasks and peripherals which can greatly reduce or outright eliminate low-level contention for shared resources access.
- Proposes to control the operating point of each shared resources (cache, memory, interconnection buses, and etc.) to avoid timing delays due to contention

Advantages

➢ COTS high performance

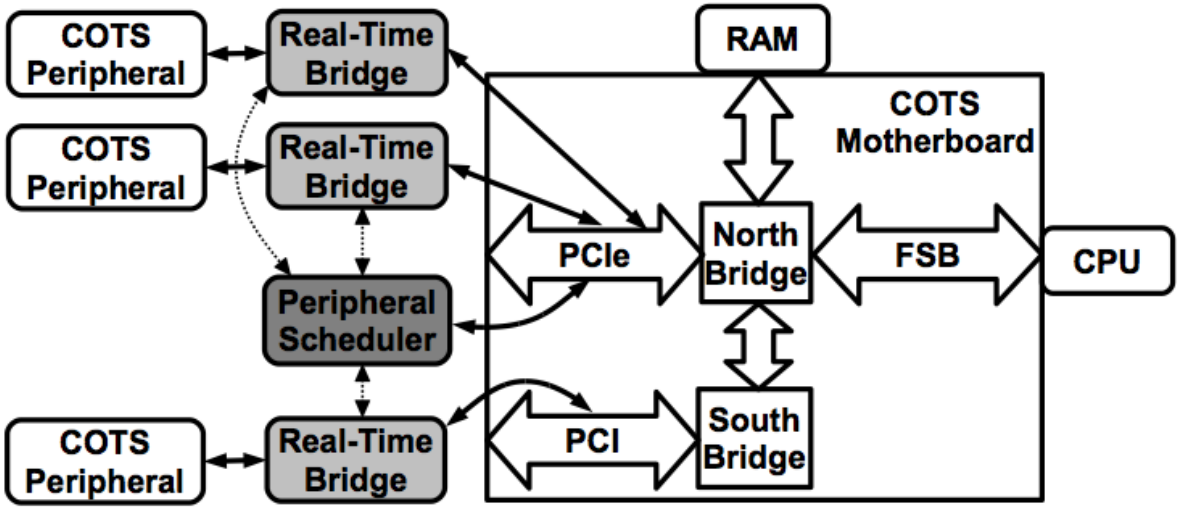
➢ Real-time predictability

PREM

- Co-schedules (at a high level) all active COTS components in the system
  - predictable, system-wide execution based on a rule set
  - less pessimistic than safe upper bounds (for non-real-time COTS) than some other approaches

PREM

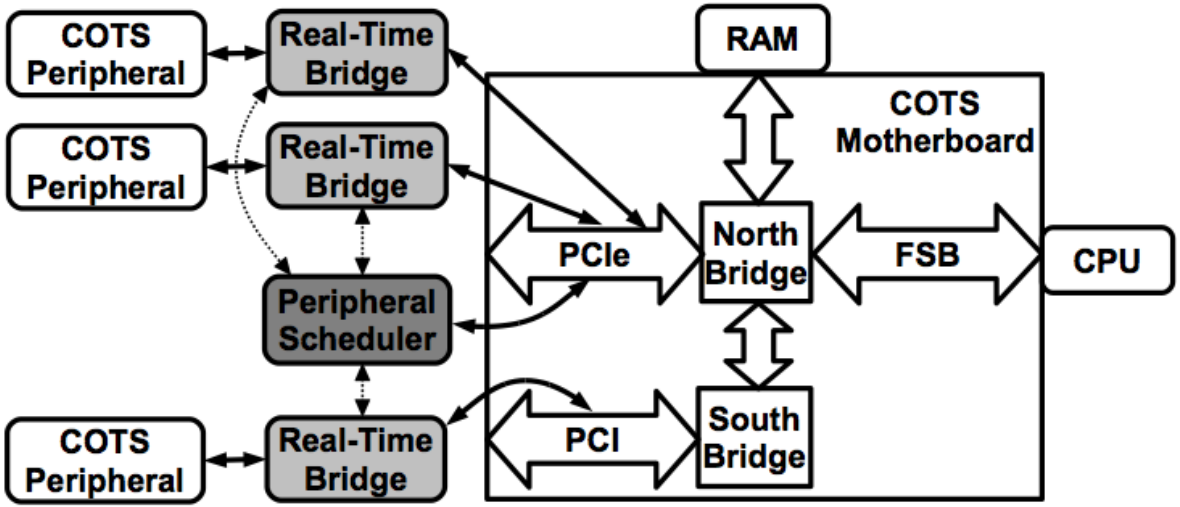
- Co-schedules (at a high level) all active COTS components in the system
  - predictable, system-wide execution based on a rule set
  - less pessimistic than safe upper bounds (for non-real-time COTS) than some other approaches



A diagram of the proposed architecture

PREM HW components

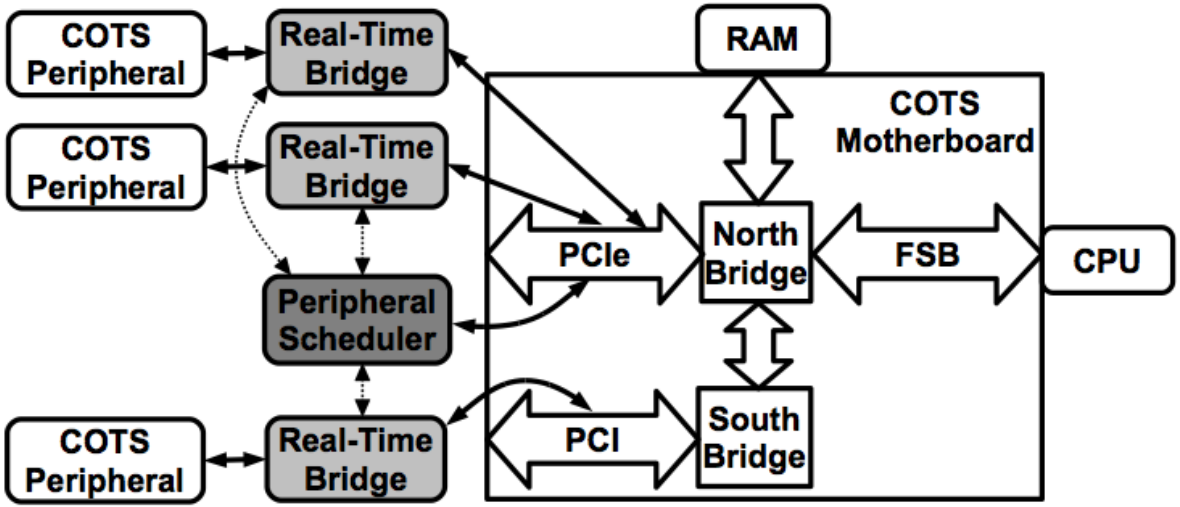
- ➔ Real-Time Bridge
- ➔ Peripheral Scheduler



PREM HW components

Predictability on memory access

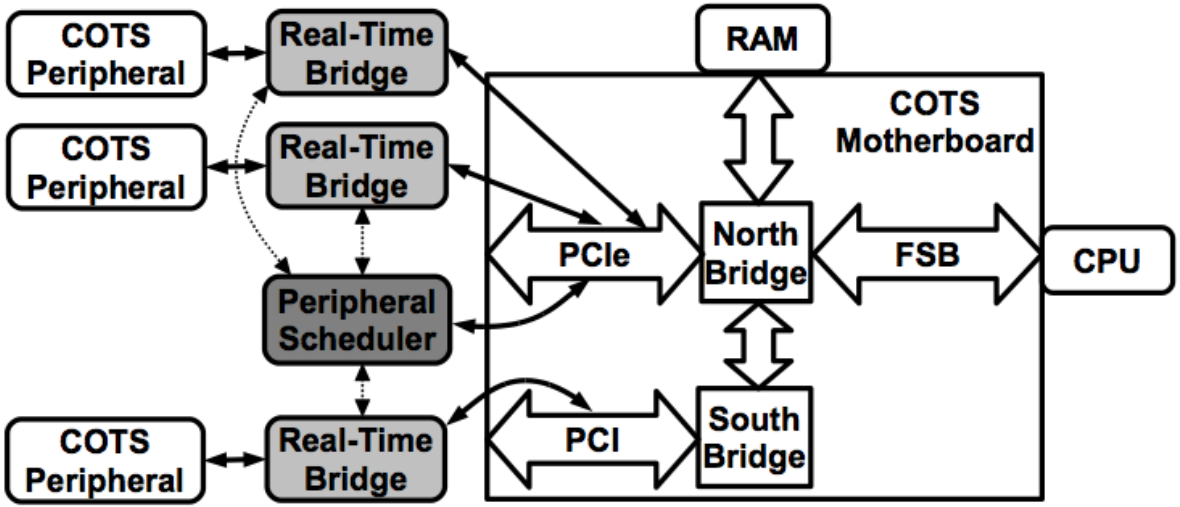
- ➔ Real-Time Bridge;
  - ◆ interposes between COTS peripheral and the rest of the system
  - ◆ provides traffic virtualization and isolation
- ➔ Peripheral Scheduler



PREM HW components

- ➔ Real-Time Bridge;
  - ◆ interposes between COTS peripheral and the rest of the system
  - ◆ provides traffic virtualization and isolation
- ➔ Peripheral Scheduler

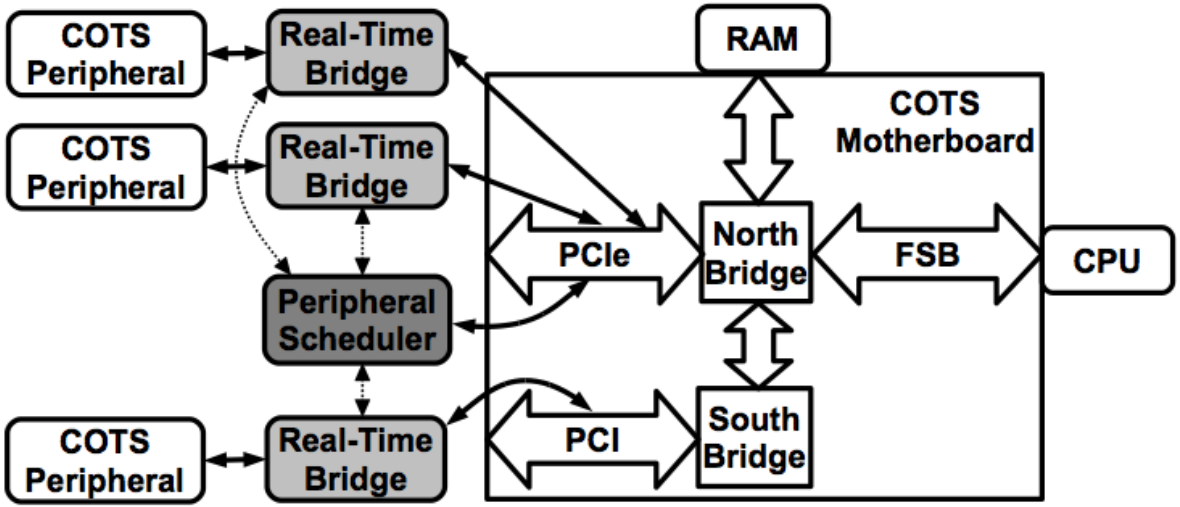
Flexibility on I/O flows



PREM HW components

- ➔ Real-Time Bridge
- ➔ Peripheral Scheduler
  - ◆ enables system-wide coscheduling after receiving scheduling messages from CPU
  - ◆ schedules the I/O flows of the bridges

synchronizes with CPU

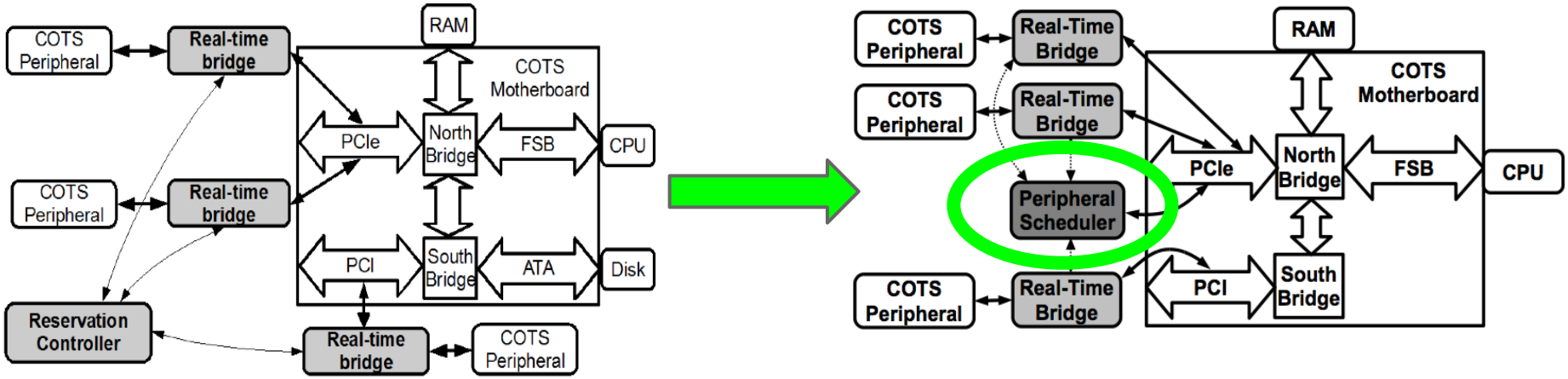


PREM challenges (1/3)

1. unpredictable manner of I/O peripherals with DMA master capabilities to access shared resources

PREM challenges (1/3)

1. unpredictable manner of I/O peripherals with DMA master capabilities to access shared resources



PREM challenges (2/3)

2. unpredictable pattern of tasks to do bus and memory access (in particular, lack of predictable cache fetches in main memory)

PREM challenges (2/3)

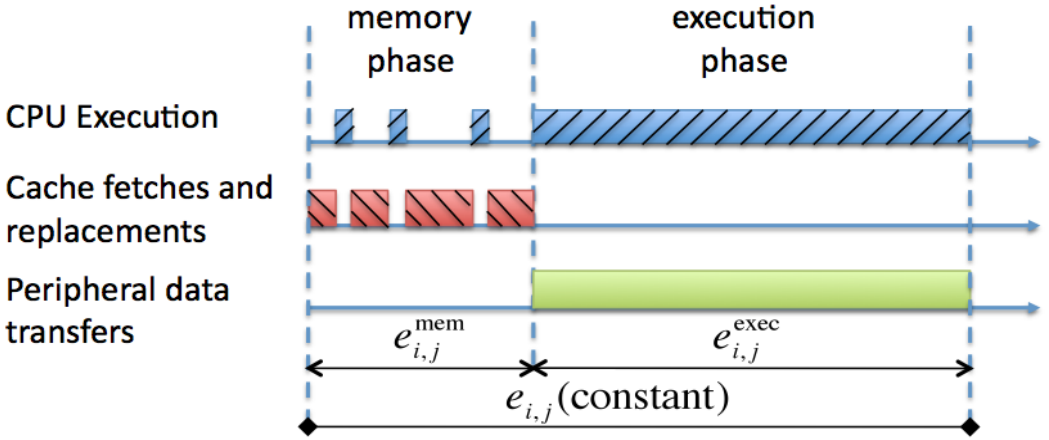
2. unpredictable pattern of tasks to do bus and memory access
(in particular, lack of predictable cache fetches in main memory)

PREM introduces a feature:

- Jobs are divided into a sequence of non-preemptive scheduling intervals
  - some of them, named *predictable intervals* are executed predictable and without cache misses by prefetching all required data at the beginning of each of their own intervals
  - their execution times are kept constant

Predictable intervals

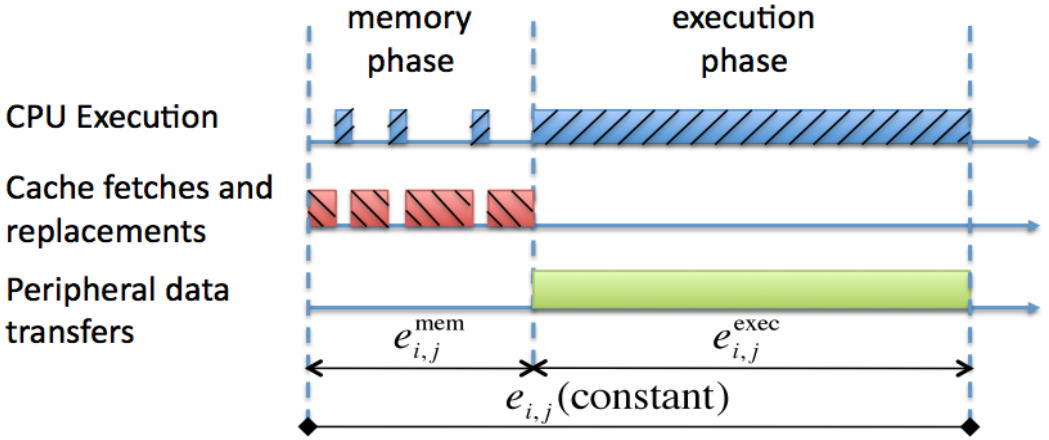
- specially compiled to execute according to the illustrated model
- divided into two different phases: *memory and execution phases*
  - during the *initial memory phase*, the CPU accesses main memory to do cache line fetches and replacement
  - now, all the required cache for the predictable interval is available in the last level cache
  - during the execution phase, useful computation without last level cache misses will be done



- the length of execution phase is forced to be equal to $e_{i,j}$constant
- $e_{i,j} = e_{i,j}^{\text{mem}} + e_{i,j}^{\text{exec}}$
- busy-wait until the constant time units have elapsed since the beginning of the interval
- Predictable intervals do not contain any system call and cannot be preempted by interrupt handlers, (guarantees no memory contention within execution phase)

Predictable intervals

- specially compiled to execute according to the illustrated model
- divided into two different phases: *memory and execution phases*
  - during the *initial memory phase*, the CPU accesses main memory to do cache line fetches and replacement
  - now, all the required cache for the predictable interval is available in the last level cache
  - during the execution phase, useful computation without last level cache misses will be done



- the length of execution phase is forced to be equal to $e_{i,j}$constant
- $e_{i,j} = e_{i,j}^{\text{mem}} + e_{i,j}^{\text{exec}}$
- busy-wait until the constant time units have elapsed since the beginning of the interval
- Predictable intervals do not contain any system call and cannot be preempted by interrupt handlers, (guarantees no memory contention within execution phase)

Question: what about OS system calls?

The scheduling intervals are classified into:

- Predictable intervals (as discussed until now)
- Compatible intervals

The scheduling intervals are classified into:

- Predictable intervals (as discussed until now)
- Compatible intervals

Properties:

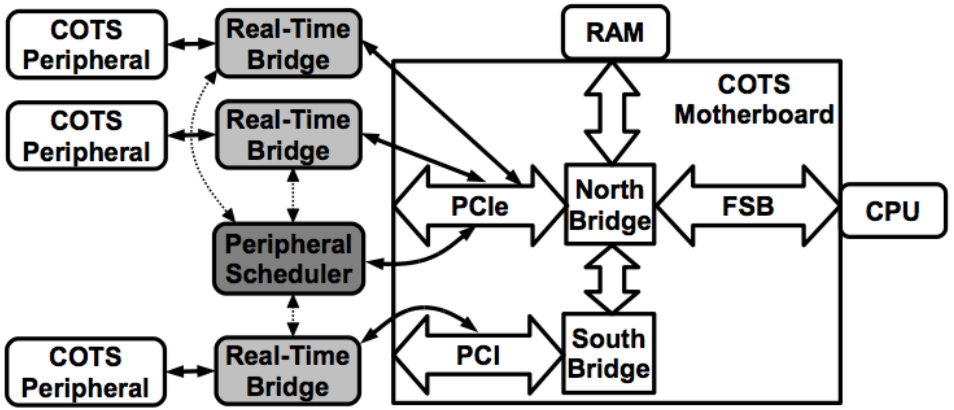
➔ are compiled and executed without any special provision as the other type of intervals
➔ so, cache misses can happen at any time
➔ OS system calls are allowed to be performed in these intervals

PREM challenges (3/3)

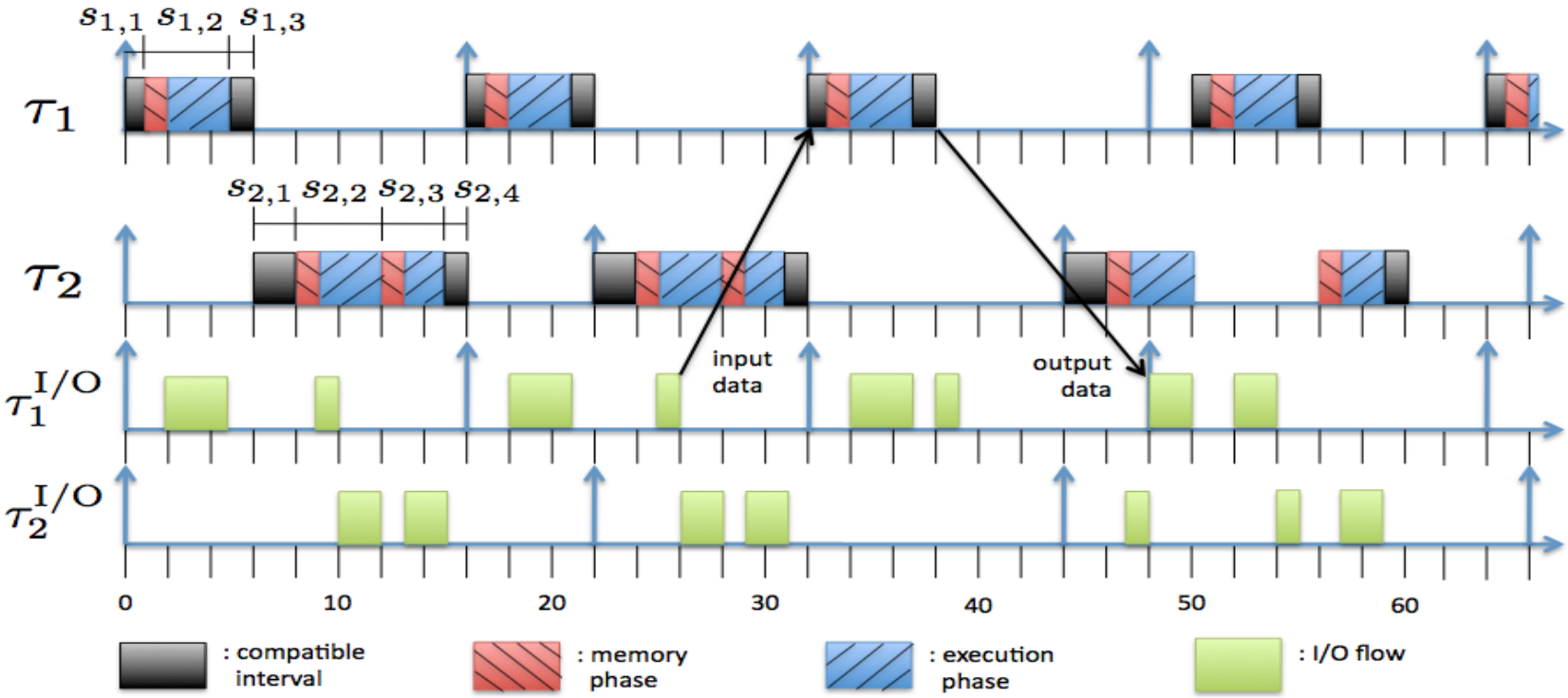
3. low-level COTS arbiters are usually designed to achieve fairness instead of real-time performance

Solution:

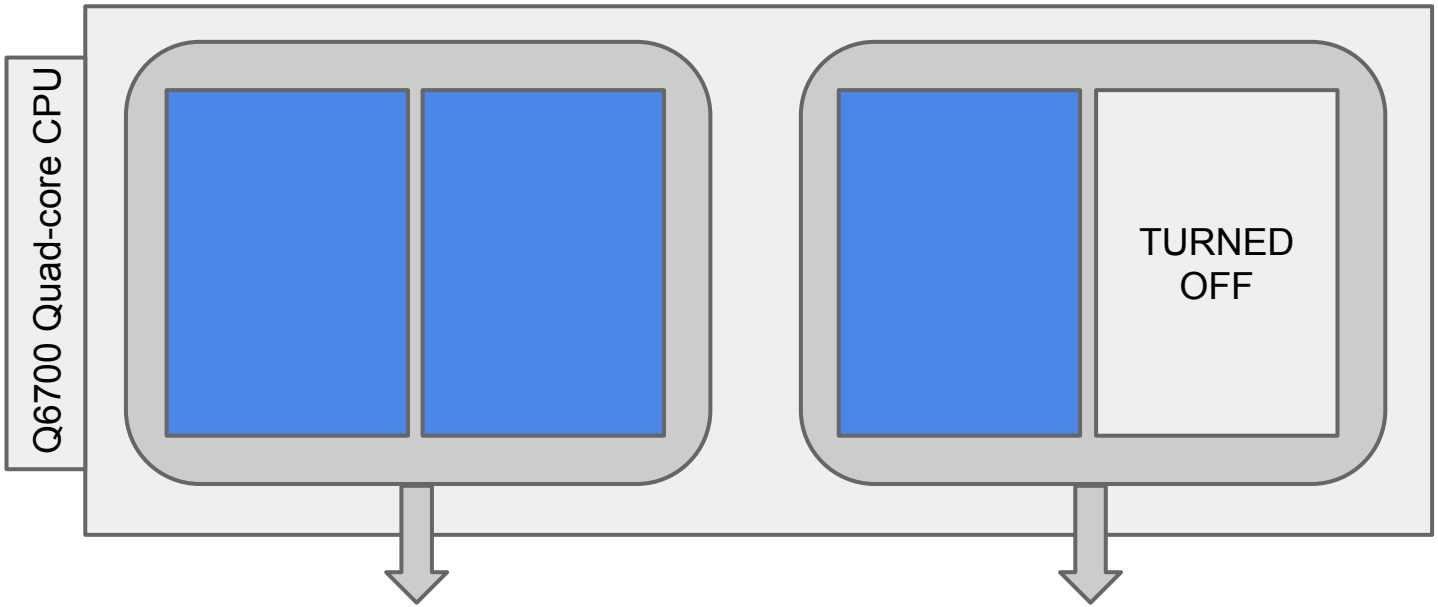
- Peripheral Scheduler!
- Then, within a task's predictable interval, the scheduled peripheral can access bus and memory (without cache-miss delay)



System-Level Predictable Schedule



Timing noises issue (Linux)



Evaluation (PREM vs. Non-PREM)

- ➔ Cache-miss & Cache-prefetch
  - DES Cypher Benchmark
  - JPEG Image Encoding Benchmark
  - Automation Program Group (MIBENCH)

- ➔ WCET (synthetic applications)
  - random_access
  - linear_access

Results (Cache-miss & Cache-prefetch)

Input bytes	4K	8K	32K	128K	512K	1M
Non-PREM miss	151	277	1046	4144	16371	32698
PREM prefetch	255	353	1119	4185	16451	32834
PREM exec-miss	1	1	1	1	1	104

TABLE I
DES BENCHMARK CACHE MISSES.

	PREM			Non-PREM	
	prefetch	exec-miss	time(μs)	miss	time(μs)
JPEG(1 Mpix)	810	13	778	588	797
JPEG(8 Mpix)	1736	19	3039	1612	3110
qsort	3136	3	2712	3135	2768
susan_smooth	313	2	7159	298	7170
susan_edge	680	4	3089	666	3086
susan_corner	3286	3	341	598	232

TABLE II
MIBENCH RESULTS WITHOUT PERIPHERAL TRAFFIC.

Critiques

- what I liked, other than the PREM system:
  - both SW and HW issues were monitored together and made the whole execution model more practical
  - demonstration was done in both ways of running benchmarks and synthetic applications as well as mathematical analysis

- my questions:
  - probably, no clear decision about the complex code segments if they should be placed in the *compatible intervals,* at the same time those intervals should be hold as short as possible…
  - as mentioned, real-time bridges and peripheral scheduler require software drivers, what about predictability of those tasks?

Thanks!