ADAPTIVE RESOURCE SHARING IN MULTICORES

Control mechanisms for the timing correct use of shared main memory

Kai Lampka (UU), Adam Lackorzynski (TUD),
Jonas Flodin (UU) and Wang Yi (UU)

Kai Lampka
kai.lampka@it.uu.se
Embedded Systems Group
Department of Information Technology, Uppsala University

MAIN BRANCHES IN RESEARCH ON EMBEDDED REAL-TIME SYSTEMS

① **Quantitative Evaluation** of Systems
(exhaustive ←→ empirically)

② **Design and implement SW mechanisms** (access protocols, memory mappings, ….) or **HW**.

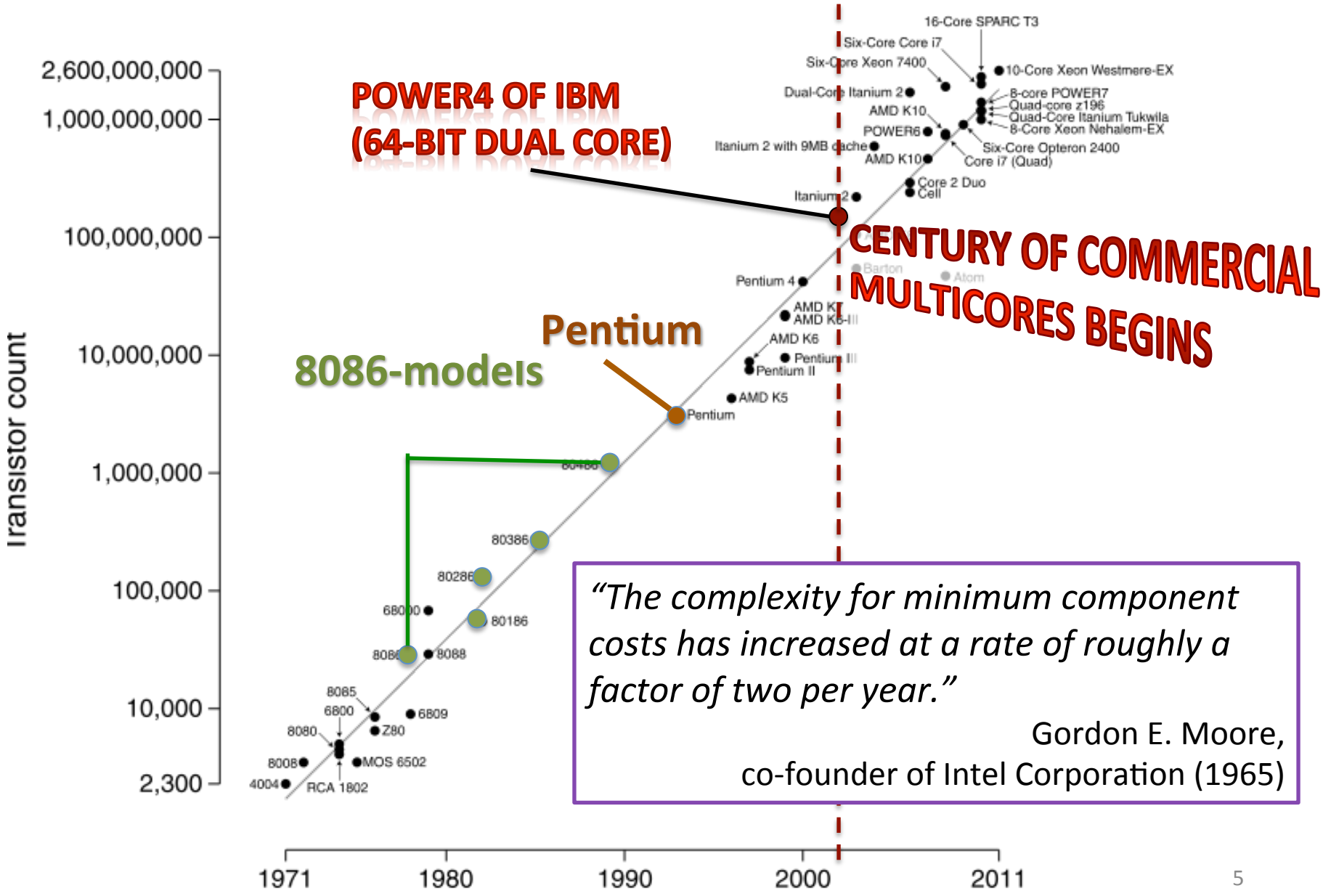
AGENDA

1. **Introduction**: Microcontrollers and technical evolution
2. **Shared memory in multicores:** Cost reduction vs Predictability
3. **Controlling applications at run-time:** Memory access control for hard real-time and best-effort applications running in parallel
4. **Conclusion**

PART 1
INTRODUCTION

Microcontrollers and technical evolution

MOORE



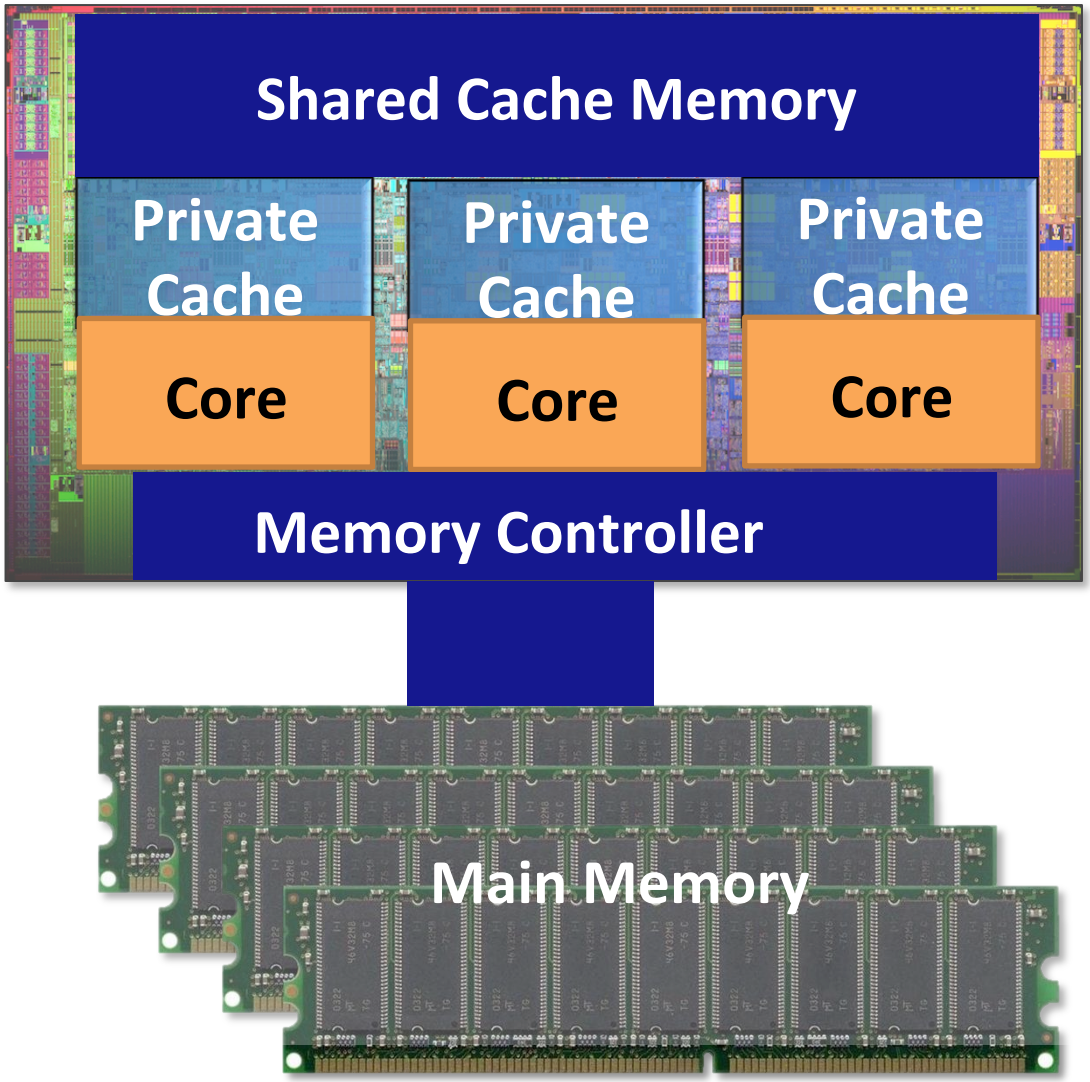
"The complexity for minimum component costs has increased at a rate of roughly a factor of two per year."

Gordon E. Moore,
co-founder of Intel Corporation (1965)

PART 2
SHARED MEMORY IN MULTICORE PROCESSORS

Architecture specific characteristics which challenge the timing correctness of applications

COMMERCIAL-OFF-THE-SHELF (COTS) ARCHITECTURES



Characteristics

- ✧ Core-local (private) Cache
- ✧ Shared Cache
- ✧ Shared main memory

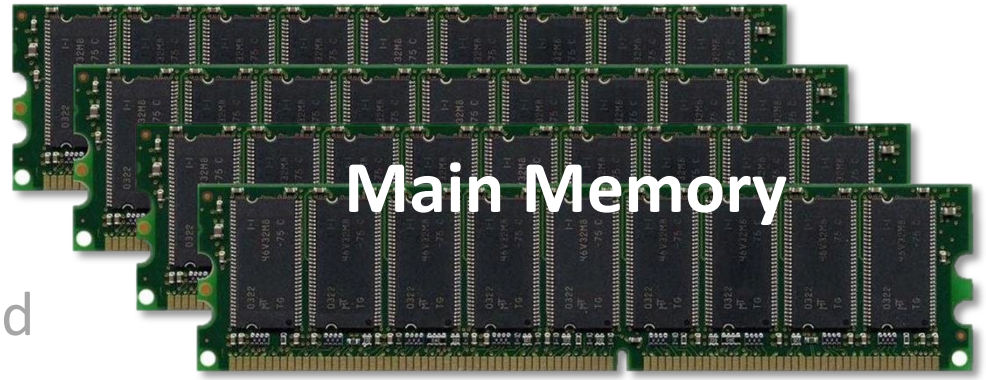
- a miss at level *i* yields a fetch attempt at level *i+1*
- **each access comes with extra timing costs**
- **acceses, e.g., to the main memory, are difficult to bound because of „fancy“ arbitration strategies**

MAIN MEMORY: ACCESS COORDINATION

Layout of memory

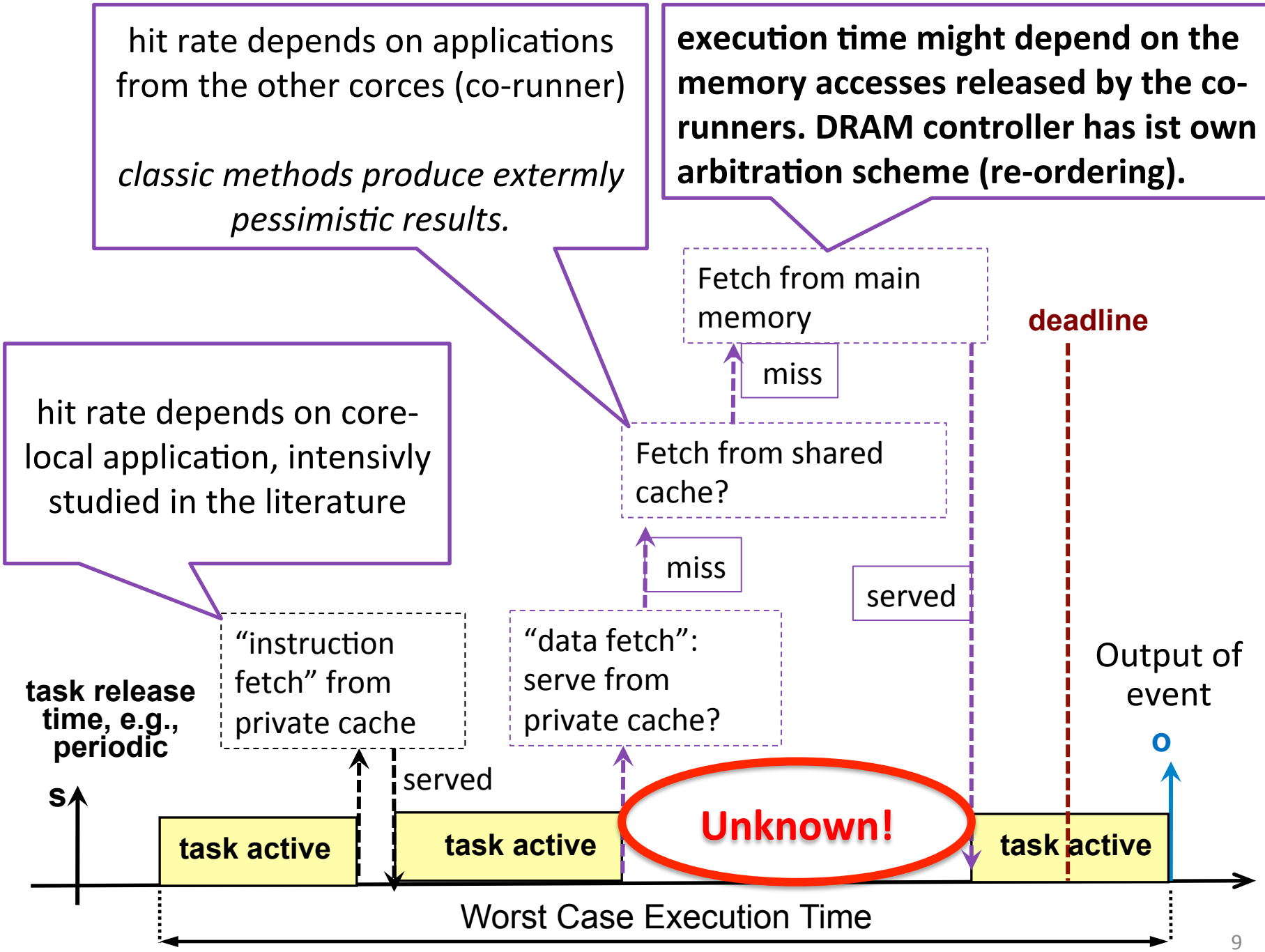
- ✧ organized in banks
- ✧ banks can be accessed quasi-parallel
- ✧ banks are split in rows
- ✧ bank-local cache
- ✧ successive accesses to the same row are faster as long as as the row is kept in the cache

=> *Open-row hit access policy for speeding up memory accesses*



„Open Row-hit policy“

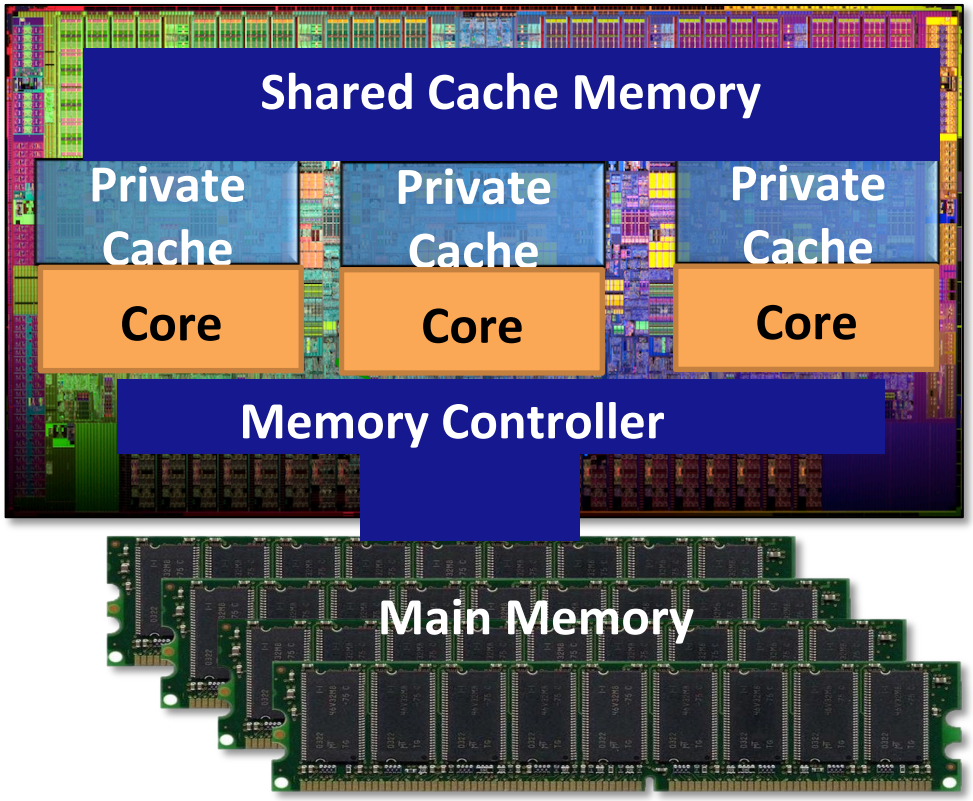
- ✧ keep complete row in the bank-local cache
- ✧ subsequent accesses from the same core commonly refer to the same row (locality)
- ✧ Re-ordering of accesses from all cores (Re-ordering policy of DRAM-controller)
- ✧ ***„Worst-case“ response time?***


hit rate depends on applications from the other corces (co-runner)
classic methods produce extermly pessimistic results.
execution time might depend on the memory accesses released by the co-runners. DRAM controller has ist own arbitration scheme (re-ordering).
hit rate depends on core-local application, intensivly studied in the literature
Fetch from main memory
deadline
miss
Fetch from shared cache?
miss
served
"instruction fetch" from private cache
"data fetch": serve from private cache?
Output of event
task release time, e.g., periodic
o
s
served
task active
task active
Unknown!
task active
Worst Case Execution Time

INTERFERENCES FROM SHARING THE MAIN MEMORY: AN EXAMPLE

Platform

- ✧ Intel Xeon 2.67GHz, 6-core CPU.
- ✧ 1 OS Thread / Core
- ✧ Caches: L1 private, L2 fully shared
- ✧ Shared main memory



Szenario

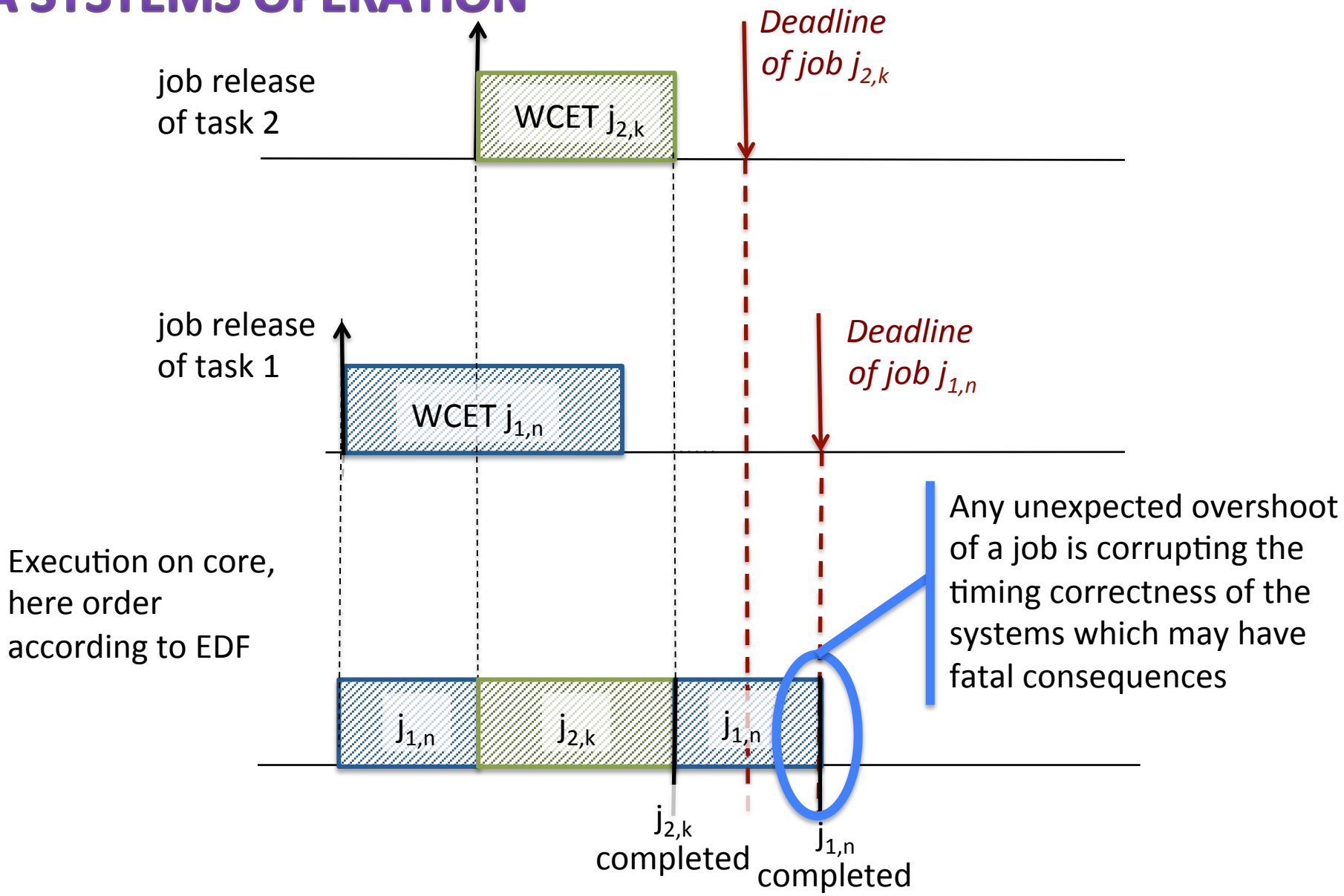
- ✧ *Embedded Microprocessor Benchmark Consortium* (EEMBC) Benchmarksuite 1.1
- ✧ 1 core for administration/ collect data
- ✧ 1 core for hard real-time applications, the ones we measure
- ✧ 4 cores running interfering applications

INTERFERENCES FROM SHARING THE MAIN MEMORY

Hard real-time task	Miss rate	Worst slowdown	Worst co-runner
a2time	1.408	32.3%	aifftr
aifftr	1.767	20.9%	bitmnp
aifirf	1.123	23.1%	canrdr
aiifft	1.405	25.6%	ttsprk
basefp	1.202	30.7%	aifirf
bitmnp	1.454	36.5%	aifirf
cacheb	1.179	17.0%	matrix
canrdr	1	25.5%	rspeed
idctrn	1.422	27.2%	cacheb
iirflt	1.488	22.7%	aiifft
matrix	1.981	30.9%	a2time
pntrch	2.306	47.6%	bitmnp
puwmod	1.62	28.6%	idctrn
rspeed	1.387	25.1%	idctrn
tblook	1.46	26.7%	idctrn
ttsprk	1.384	35.5%	bitmnp

Tasks of the EEMBC-benchmark suite (1 to 4 cores)

INCORRECTLY BOUNDING THE WCRT IS A THREAT TO A SYSTEMS OPERATION



WHY DO WE NEED TO GUARANTEE UPPER BOUNDS ON THE WCRT?

- **Real-time Scheduling**:
  Do all task invocations meet their deadlines?

  -> unexpected timing violations

- **Performance-Analysis**:
  end-to-end latency & buffer space

  -> unexpected timing & memory violations

=> unexpected service requests at a shared resource, e.g., main memory, have the potential to inject additional delays into the WCET/WCRT of a job

False WCET/WCRT introduce (systematic) errors, which are impossible to be repaired at a later stage of the development cycle

PART 3
CONTROLLING APPLICATIONS AT RUN-TIME

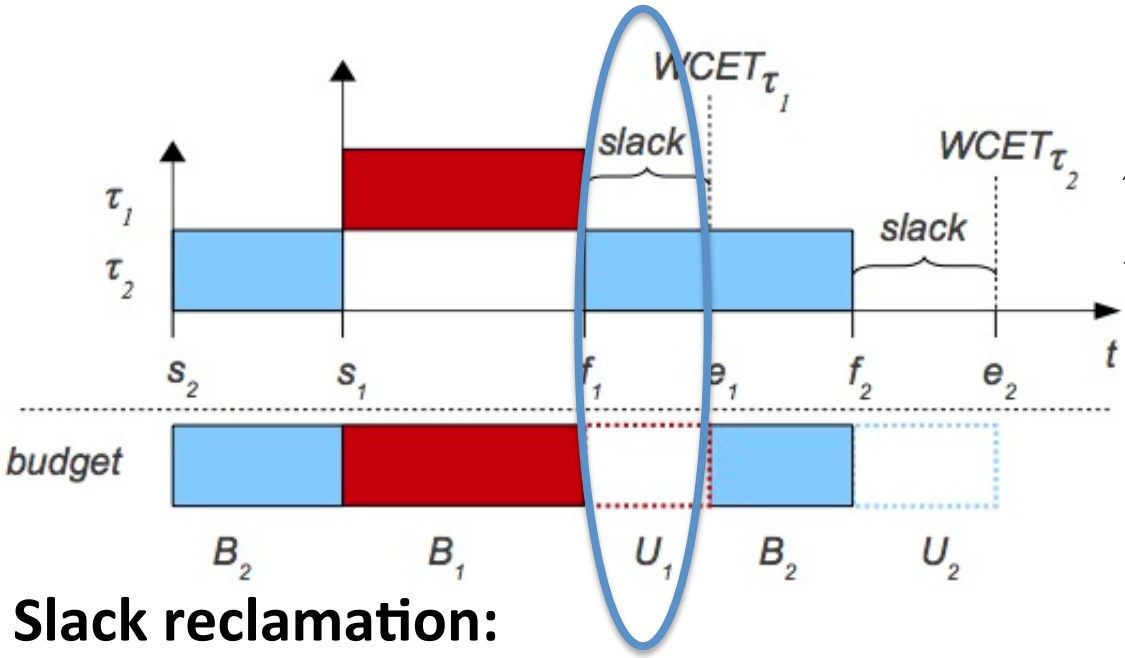
Memory access control with parallel hard real-time applications

Joint work with Jonas Flodin (PhD student) and Wang Yi (Chair for ES)

DYNAMIC BUDGETING (MAIN IDEA)

- Map soft and hard real-time cores exclusively to cores
- memory accesses of soft real-time applications is tracked with architecture-inherent performance monitors (increment upon cache miss).
- Each hart RT task allows the co-runner to access the memory up to a certain budget. **This guarantees the upper bound on the delays injected into the WCET (not tight though)**
- upon termination, the hard RT nullifies ist enforced budget
- if all budgets are nullified, soft RT tasks accesse resource as needed
- Jonas Flodin, Kai Lampka, Wang Yi: Dynamic budgeting for settling DRAM contention of co-running hard and soft real-time tasks. SIES 2014: 151-159

EXAMPLE: 1 CORE WITH 2 HARD RT TASKS



τ_i: task id

B_i: Budget from Task i

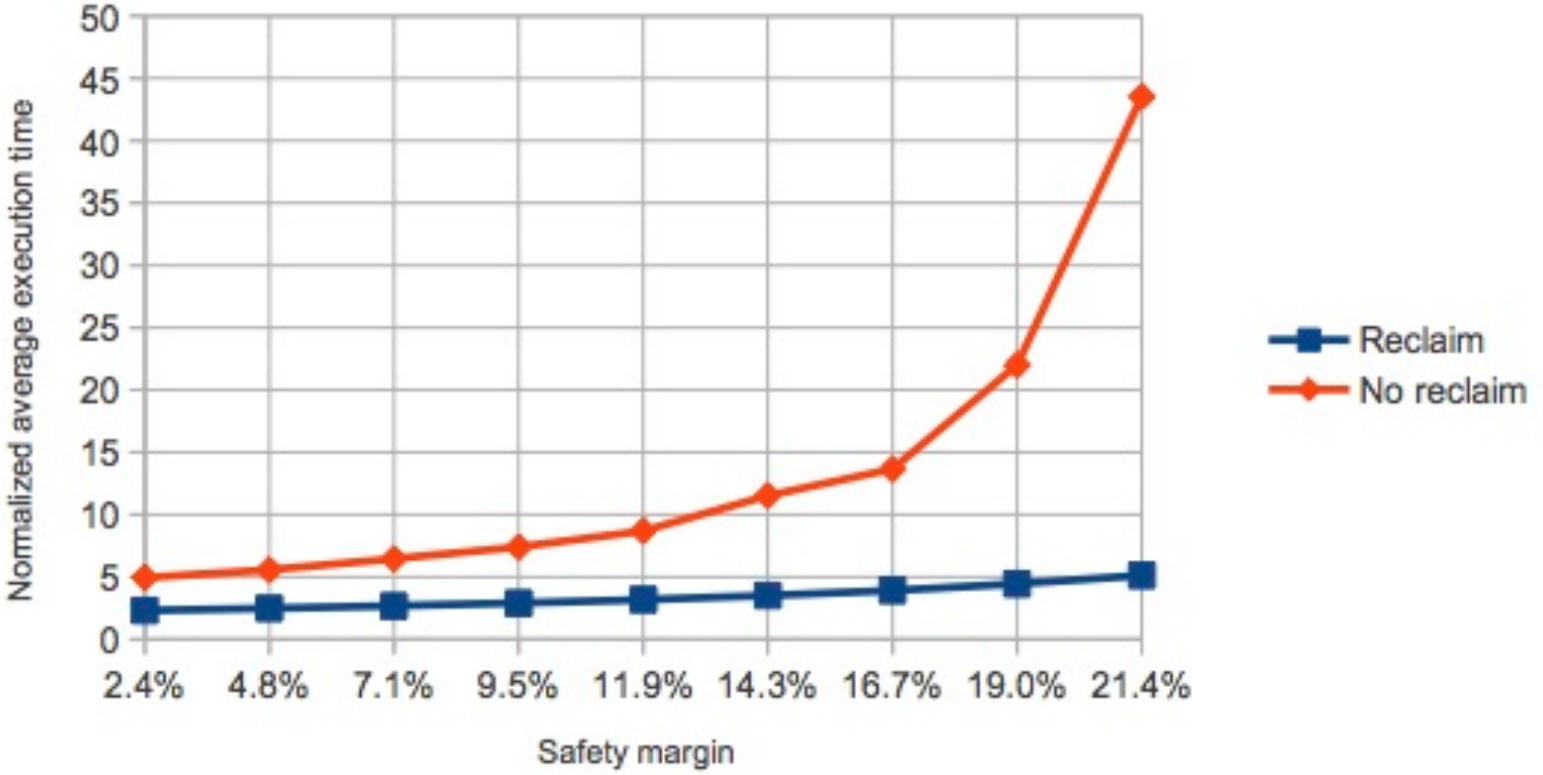
U_i: unused access bandwidth

WCET : Worst-case Execution Time

Slack reclamation:

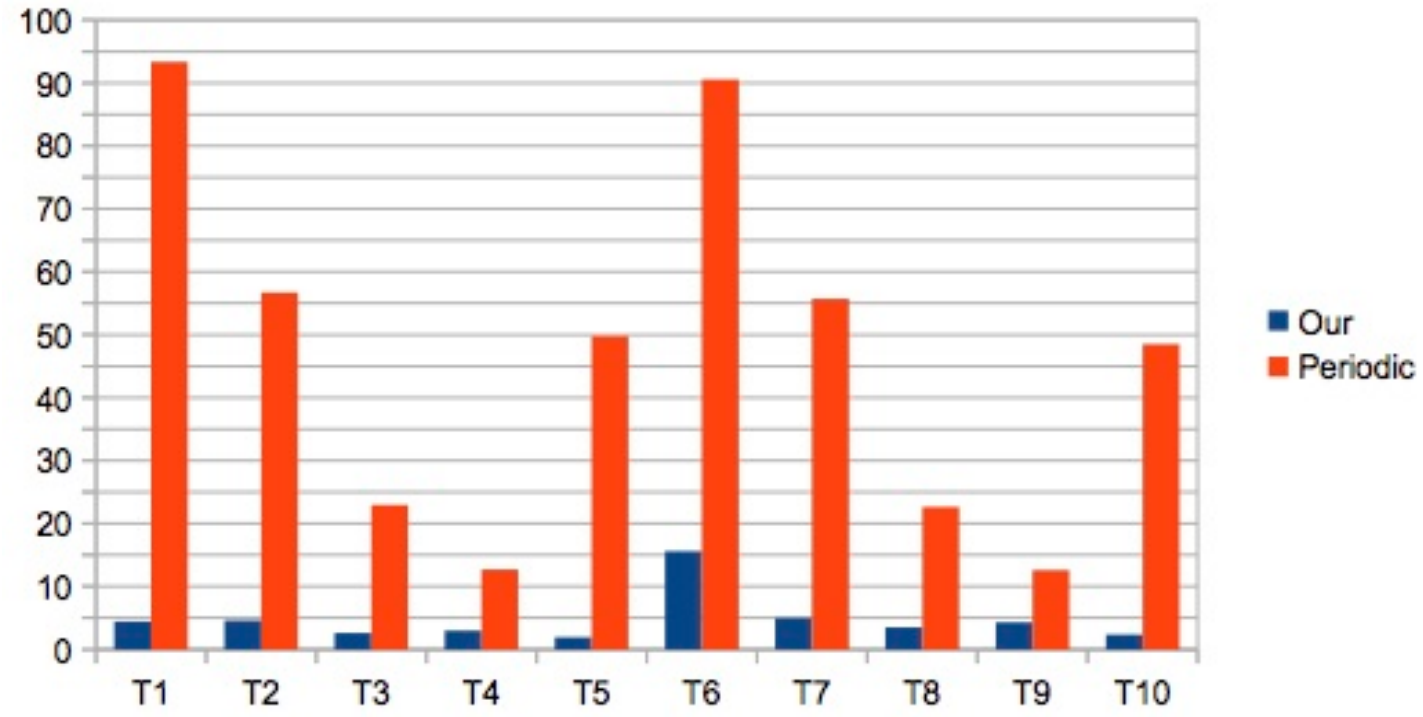
- Early completition of task 1 allows one to nullify budget B_1 during $[f_1,e_1]$
- Budget B_2 needs not to be activated before e_1, no task execution other than Task 1 assumed in the analysis during $[f_1,e_1]$
- Any delay of Task 2 during $[f_1,e_1]$ is without effect on the feasability

EMPIRICAL EVALUATION (1)



Normalised execution time of task “bitmnp”
as “best-effort” application with and without dynamic budgetings (slack reclaim vs. no slack reclaim)

EMPIRICAL EVALUATION

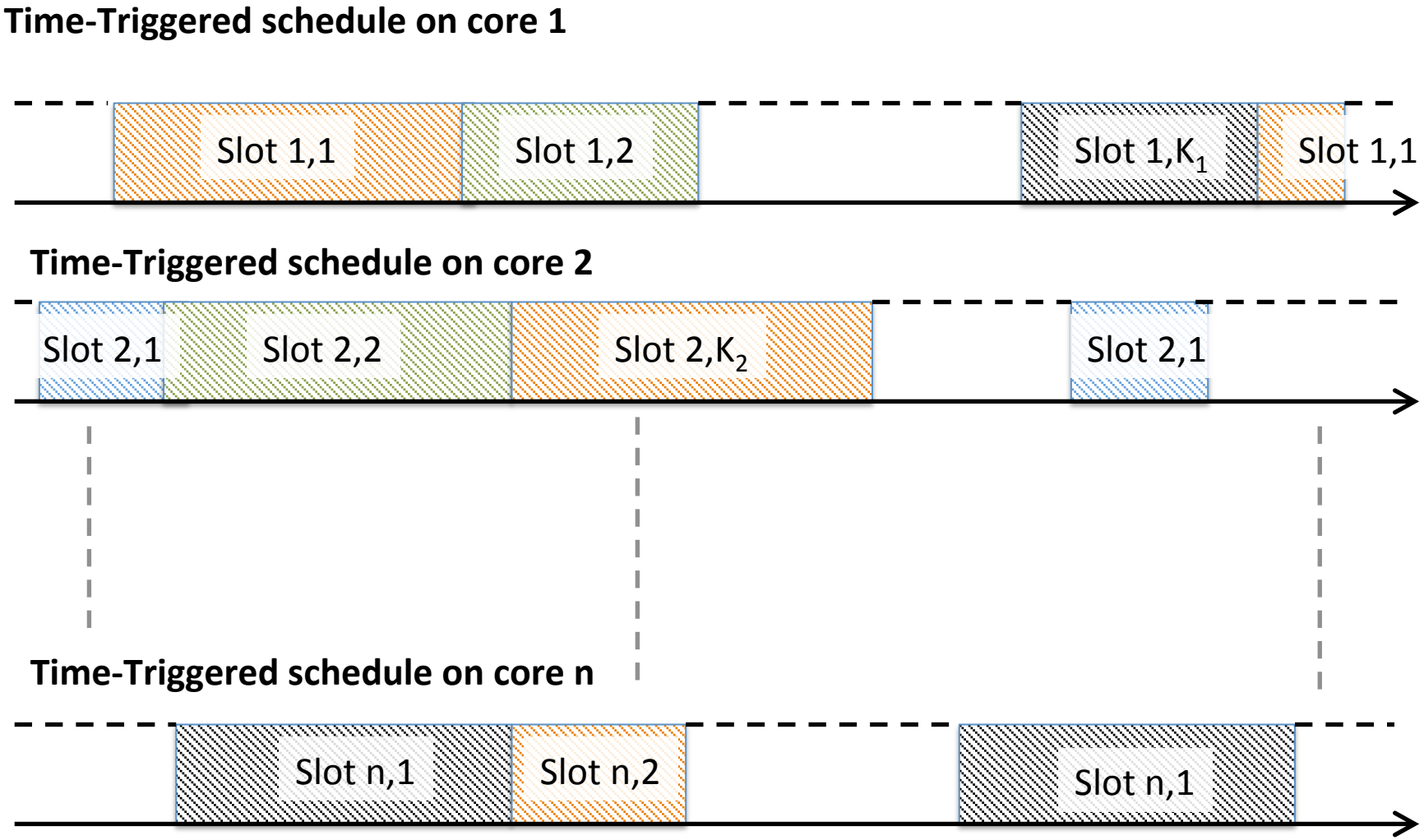


Average execution time of different tasks under

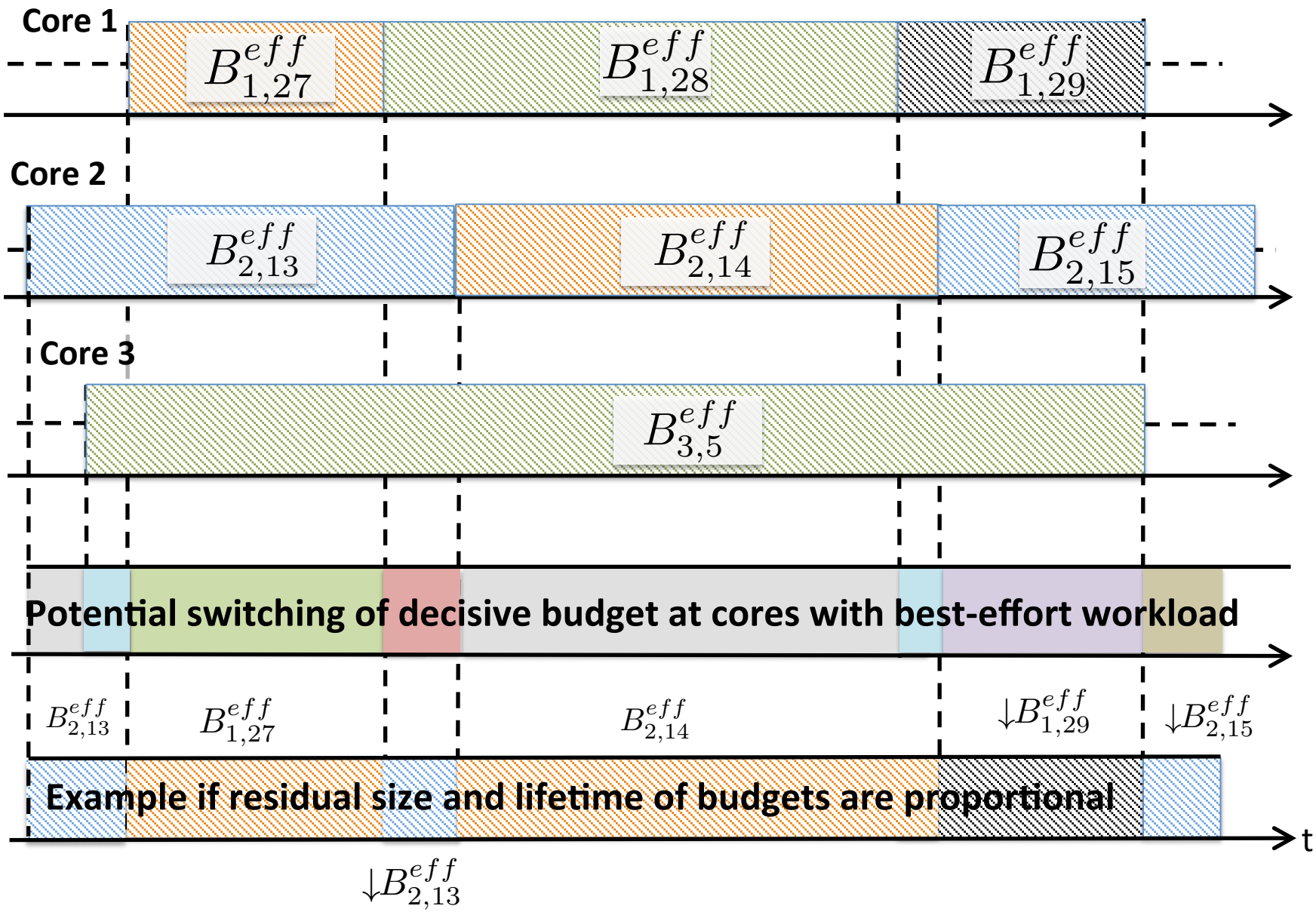
(a) dynamic budgeting (blue) and

(b) strictly periodic budgeting (red).

ONGOING WORK – BUDGETS FOR MEMORY ACCESS UNDER TT-EXECUTION ORDERS



REDUCES NUMBER OF IPCS FOR UPDATING BUDGETS



ONGOING WORK – INITIAL PROBLEMS

- ✧ Enforce budgets via scheduling contexts in L4
- ✧ Exploit Performance Monitor Counters for counting last level cache misses
- ✧ Injection of stalling intervals into the execution of best effort applications showed significance of prefetching.
- ✧ Disabling the prefetcher via (IA32_MISC_ENABLE) did not work.

ONGOING WORK – INITIAL RESULTS (NOT IN THE PAPER)

- TI OMAP5 platform.
- 2 ARM Cortex-A15 cores (800MHz).
- A15's performance counter offers a BUS_ACCESS counter

(A) Greedy memory use

Runtime	CPU0	CPU1
1	8762ms	-
2	15228ms	15551ms

(B) Non-Greedy use

CPU0	CPU1
12835ms	-
16734ms	16763ms

TABLE I. BENCHMARK RESULTS FOR A MEMORY-INTENSIVE BENCHMARKS RUNNING ON ONE AND ONE TWO CORES IN PARALLEL.

RELATED WORK

(A) strictly periodic budgeting for the "Best-effort" applications (BEA)

(B) Slack is not reclaimed by the BEA

- H. Yun, G. Yao, R. Pellizzoni, M. Caccamo, and L. Sha. *Memory access control in multiprocessor for real-time systems with mixed criticality*. ECRTS 2012.
  Only one core with hard RT applications, new budget lifting

- H. Yun, G. Yao, R. Pellizzoni, M. Caccamo, and L. Sha. *Memguard: Memory bandwidth reservation system for efficient performance isolation in multi-core platforms*. RTAS 2013. ***slack reclamation only among BEA***

- M. Behnam, R. Inam, T. Nolte, and M. Sjödin. *Multi-core composability in the face of memory-bus contention*. SIGBED Rev., 10(3):35–42, Oct. 2013.
  No slack reclamation

CONCLUSION

1. **Embedded goes Multicore**
2. **Sharing the main memory between hard and soft real-time applications may impose new challenges for the timing correctness of systems**
3. **Controlling applications at run-time:** *Dynamic memory access control* for hard real-time and best-effort applications running in parallel