TECHNISCHE UNIVERSITÄT DRESDEN
Fakultät Informatik Institut für Systemarchitektur, Lehrstuhl Betriebssysteme

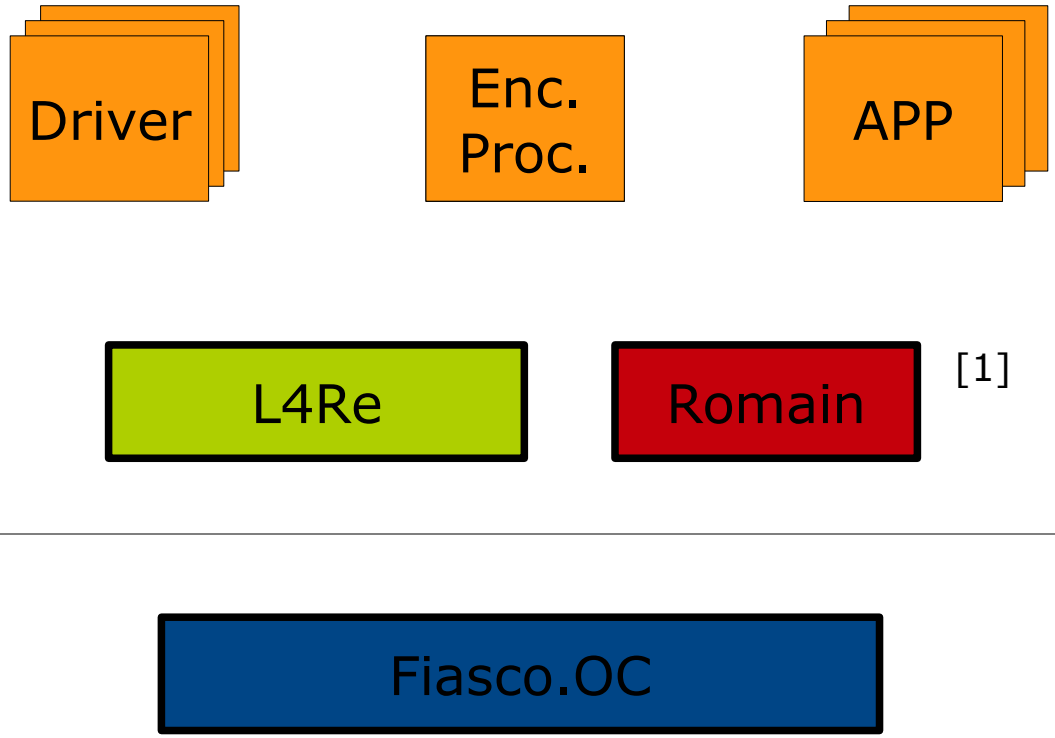
Where Have all the Cycles Gone?

Investigating the Runtime Overheads of OS-Assisted Replication

Björn Döbel, Hermann Härtig
TU Dresden Operating Systems Group

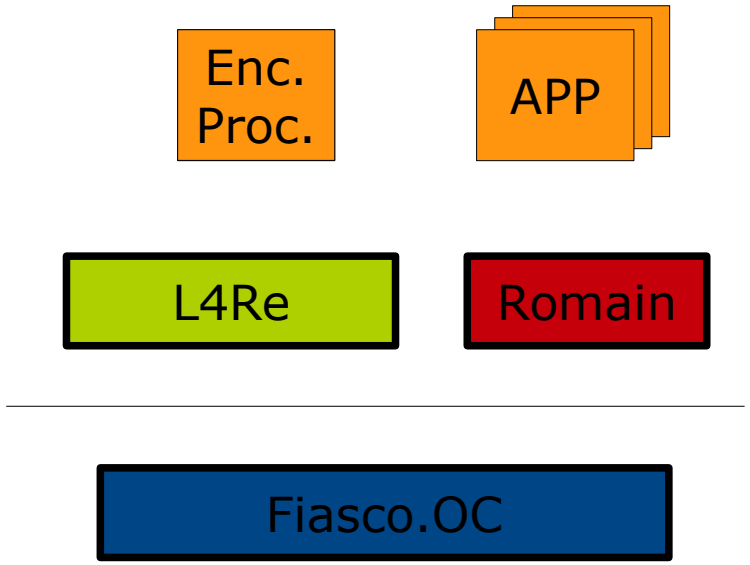
Koblenz, 16.09.2013

ASTEROID – OS-Assisted Replication



[1] Döbel, Härtig, Engel: *Operating System Support for Redundant Multithreading*, EMSOFT 2012

ASTEROID – OS-Assisted Replication

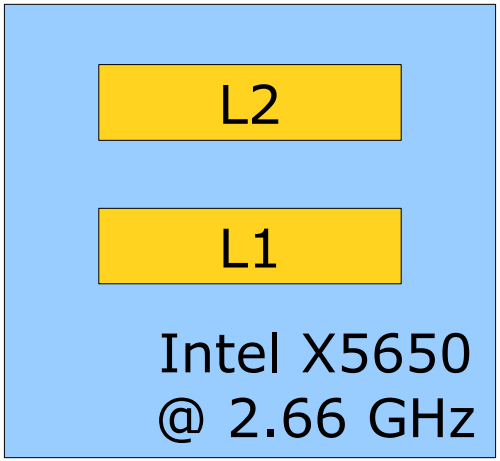


- Interpose on system calls & CPU exceptions
- Replicate memory (no need for ECC)
- Unmodified binary applications

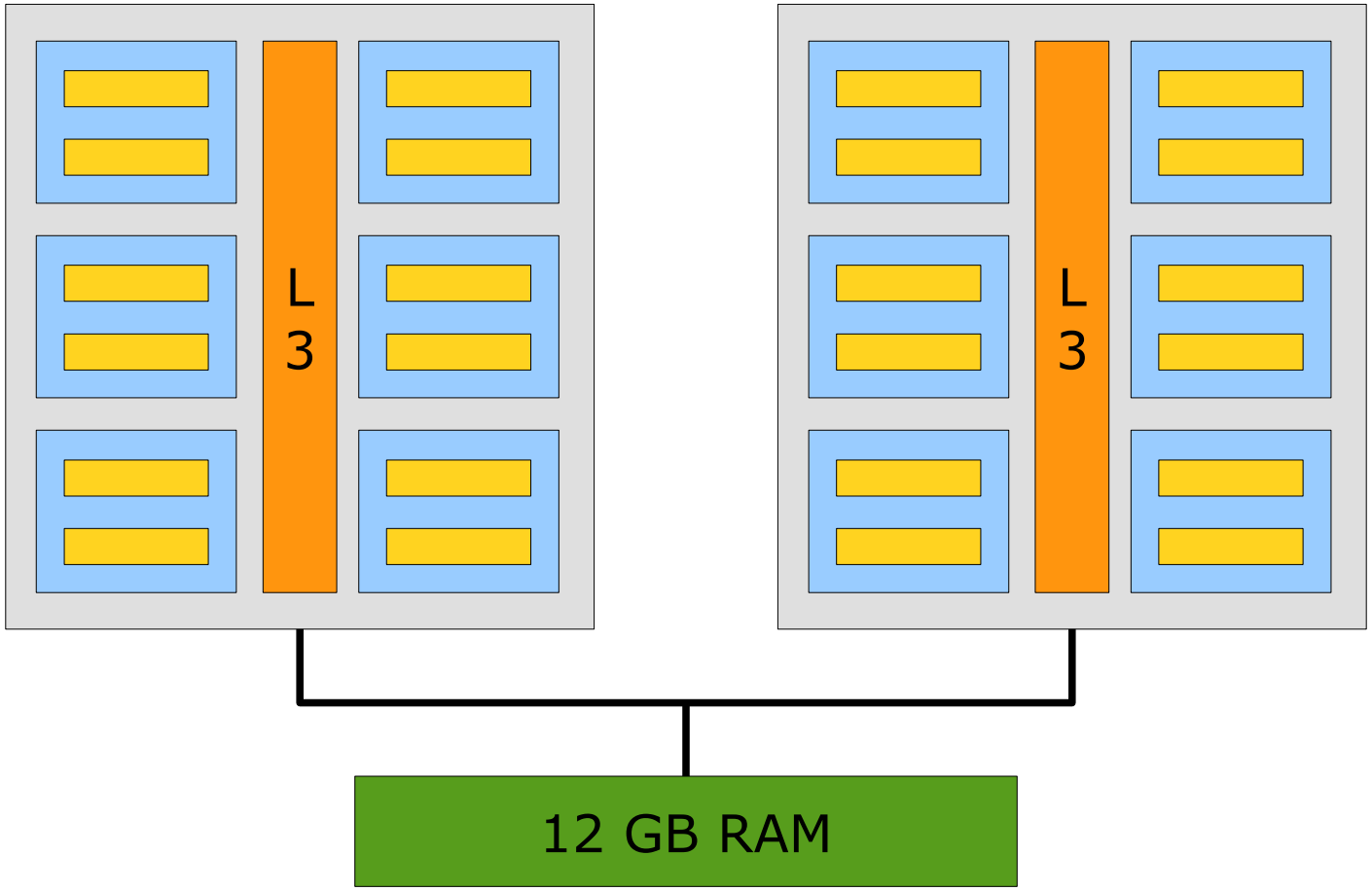
How Much is Replicated Execution?

- **Resource Overhead**
  - Roughly: N x replication → N x resouces
  - Optimizations vs. error coverage
- **Fault Coverage**
  - No complete measurements yet
  - Estimation: matches compiler-assistance
- **Runtime overhead**
  - Should be optimized for
  - This paper: SPEC INT 2006

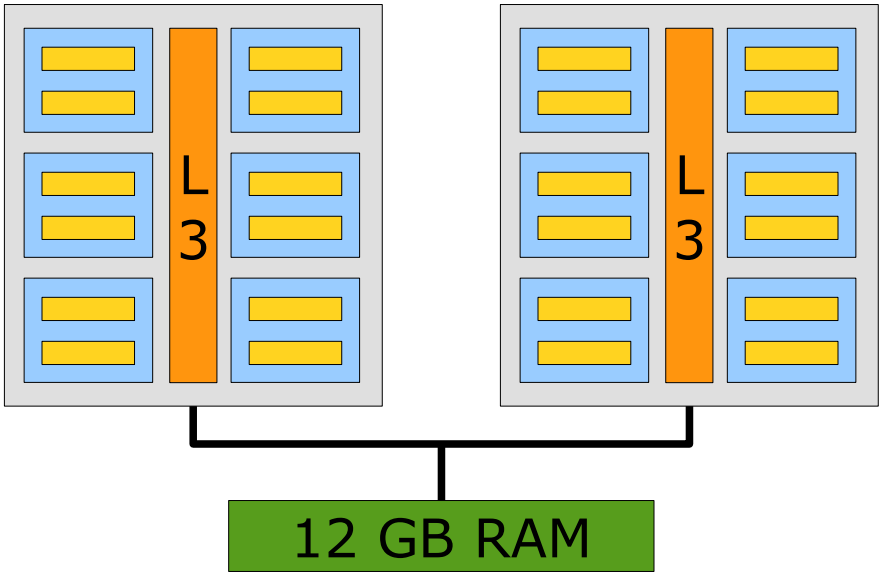
Experiment Setup



Experiment Setup



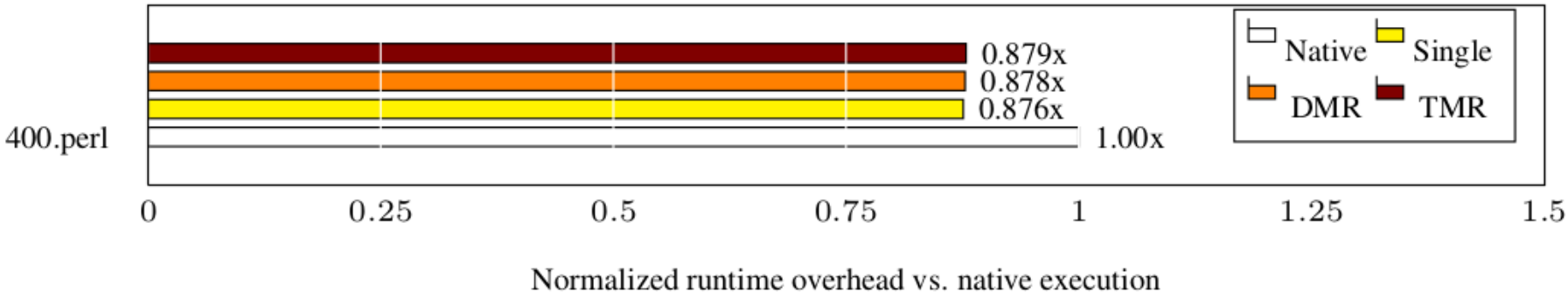
ASTEROID – OS-Assisted Replication



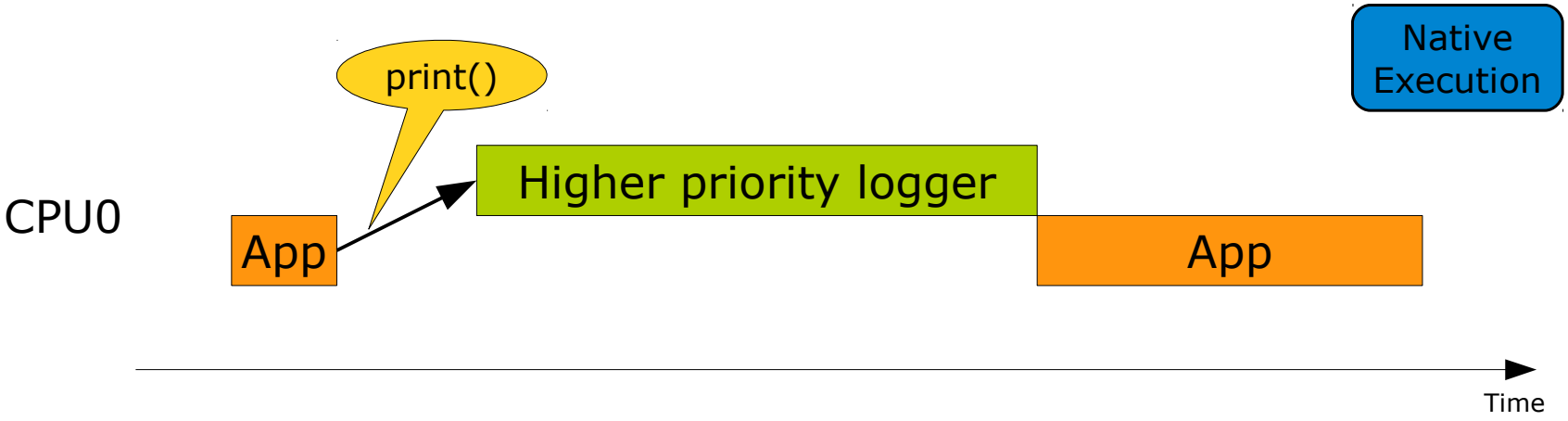
- L4/Fiasco.OC,
  32 bit + Romain
- SPEC INT 2006

```
400.perl
401.bzip2
403.gcc
429.mcf
445.gobmk
456.hmmer
458.sjeng
462.libquantum
464.h264
471.omnet++
473.astar
~~478.xalancbmk~~
```

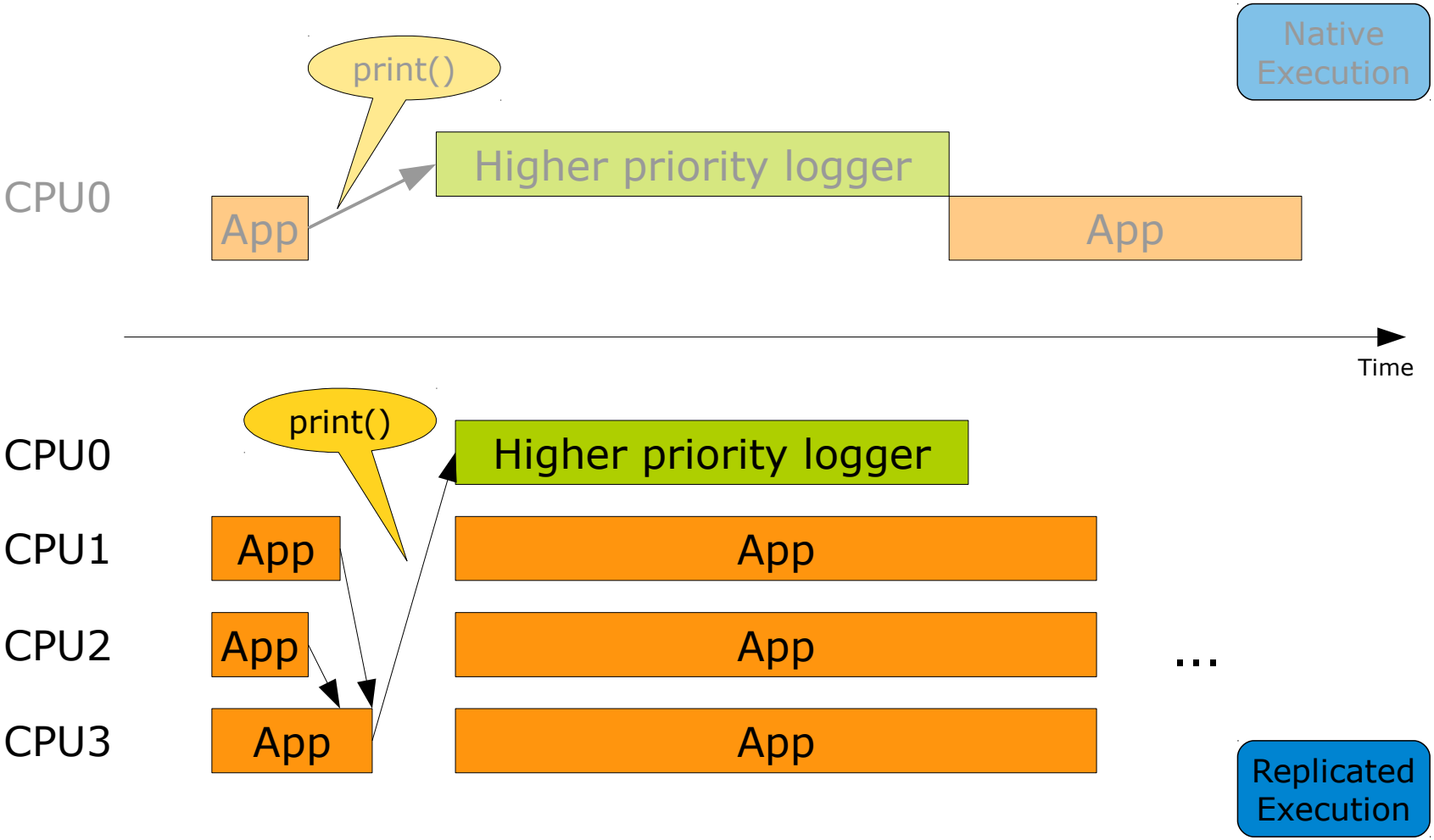

Engage!


400.perl
0.879x
0.878x
0.876x
1.00x
Native
Single
DMR
TMR
0
0.25
0.5
0.75
1
1.25
1.5
Normalized runtime overhead vs. native execution

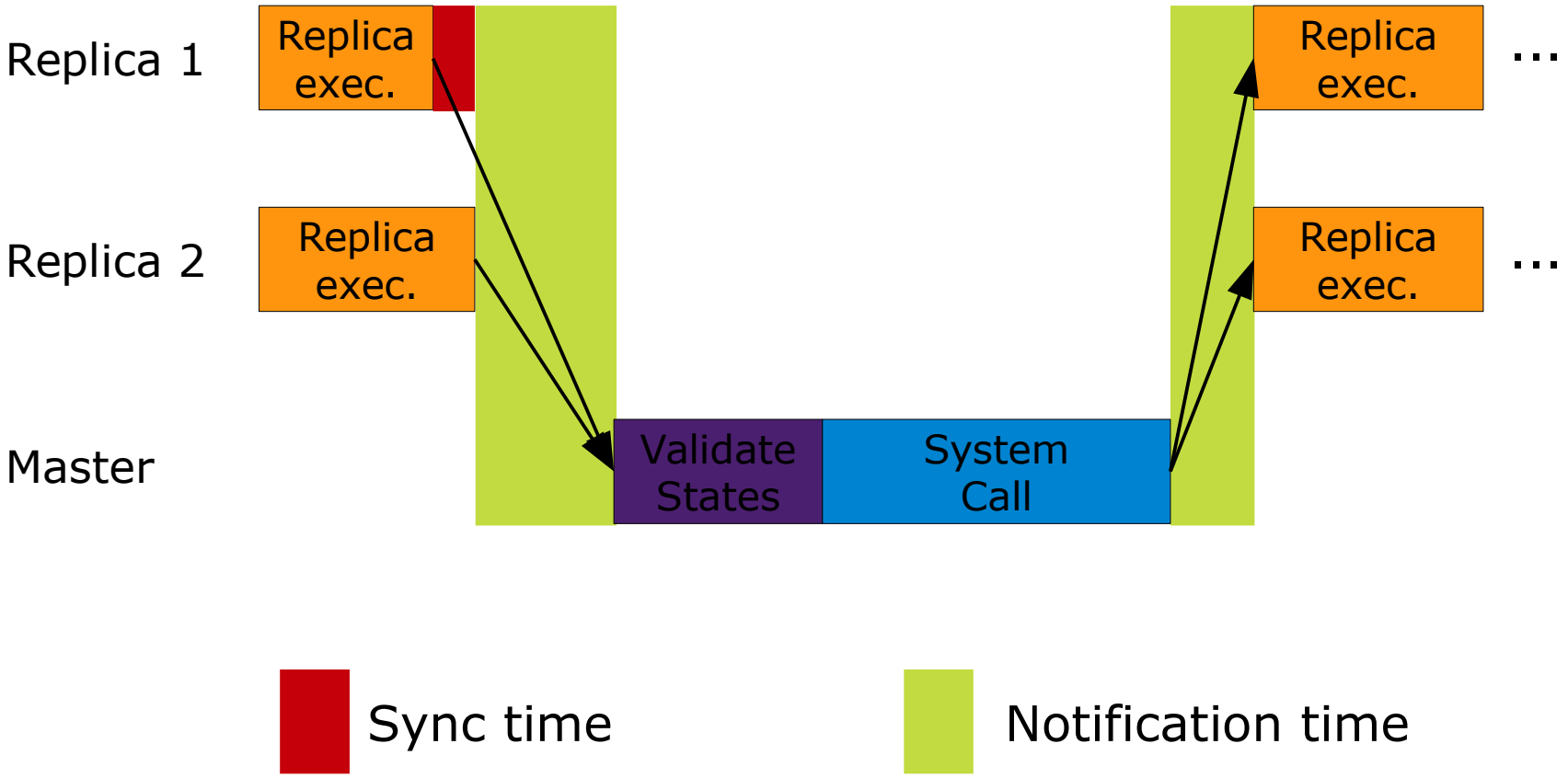
The Problem: CPU Assignment



The Problem: CPU Assignment



Where does overhead come from?

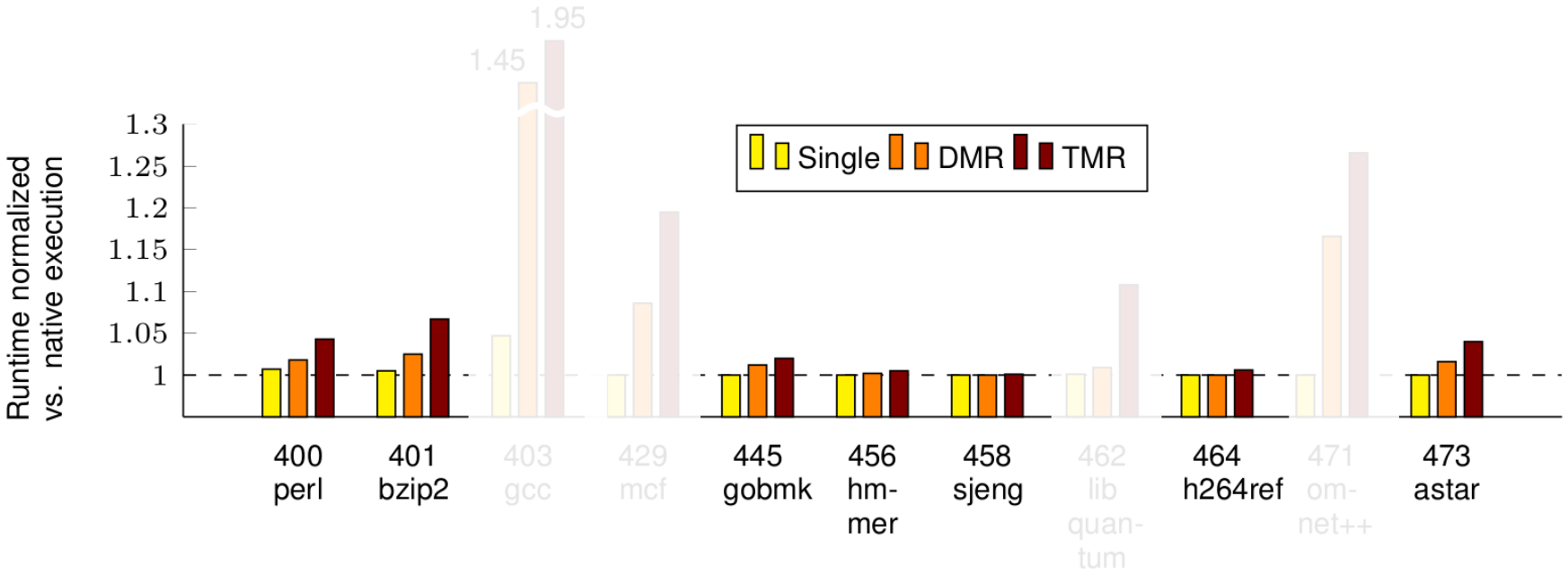


Where does overhead come from?

Source		Overhead vs. native
Per-replica execution	unmodified	+/- 0
Sync time	No background load	~ 0
State comparison		~ 100 cycles
System call	Mostly unmodified	~ 0
Notifications	Local core	~ 2,000 cycles
	On socket	~ 6,000 cycles
	Cross-socket	~ 14,300 cycles

Rule: Prefer placing replicas on the same CPU socket

Replicating SPEC INT 2006

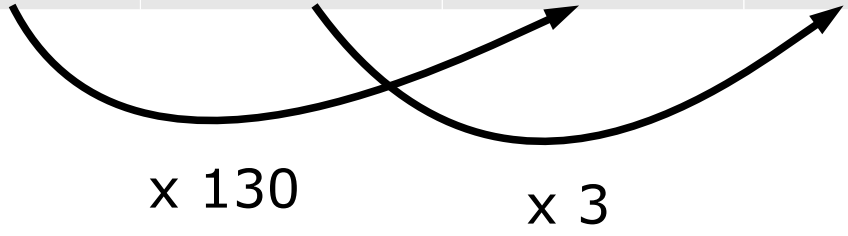


Idea: Reduce Memory Management Overhead

- Assumption: memory management is expensive
- Idea: reduce overhead by using x86 huge pages (4 MB)
- Works for microbenchmark
- SPEC CPU: (nearly) no difference

Secondary Effects: Cache Miss Rates

Benchmark	DMR misses		TMR misses	
	L2	L3	L2	L3
429.mcf	2,600	1,300,000	11,000,000	5,200,000
462.libquantum	2,500	570	440,000	387,000
471.omnet++	270,000	6,900,000	35,000,000	21,200,000

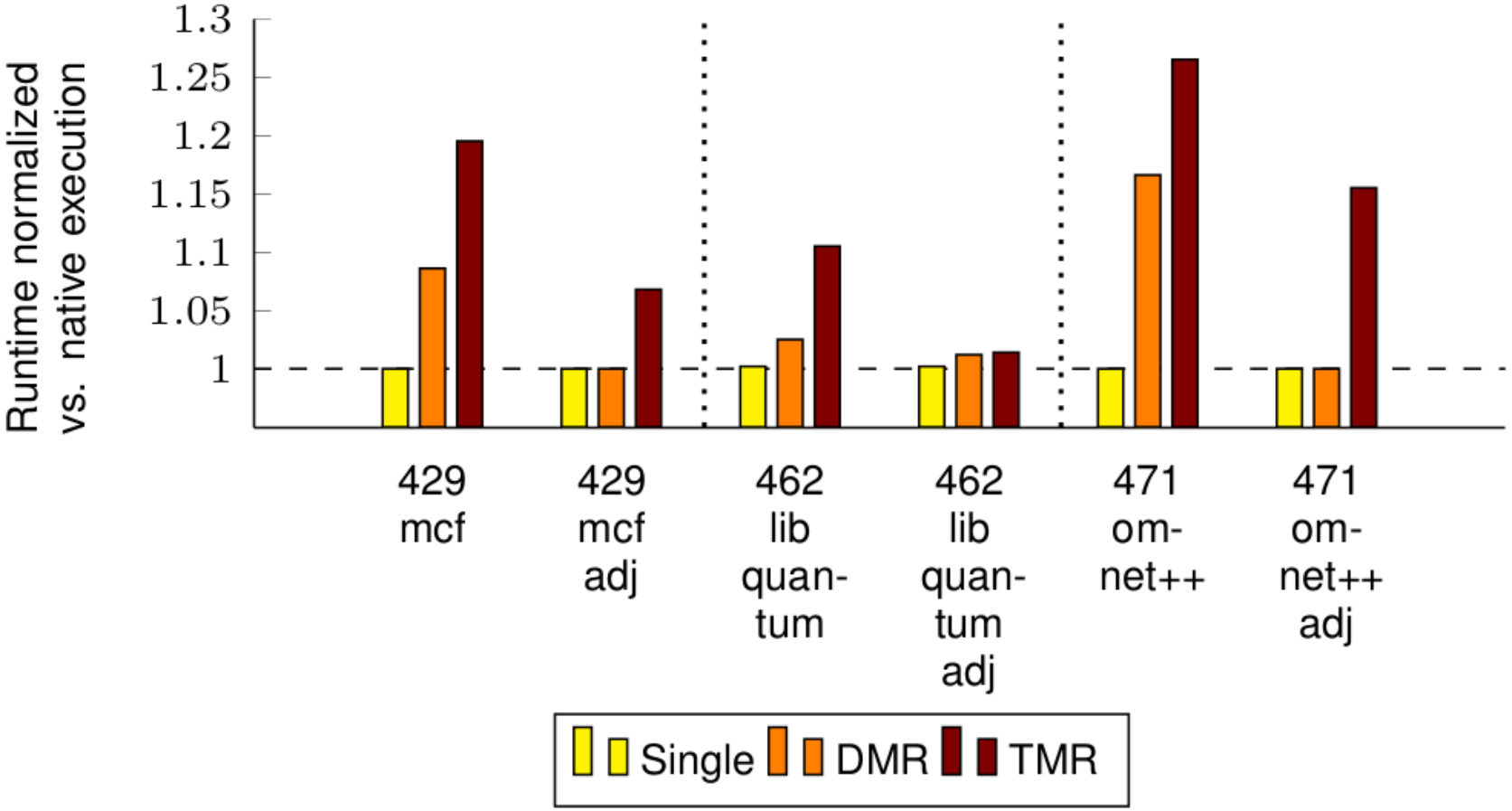


Rule: Prefer placing replicas **on a different** CPU socket

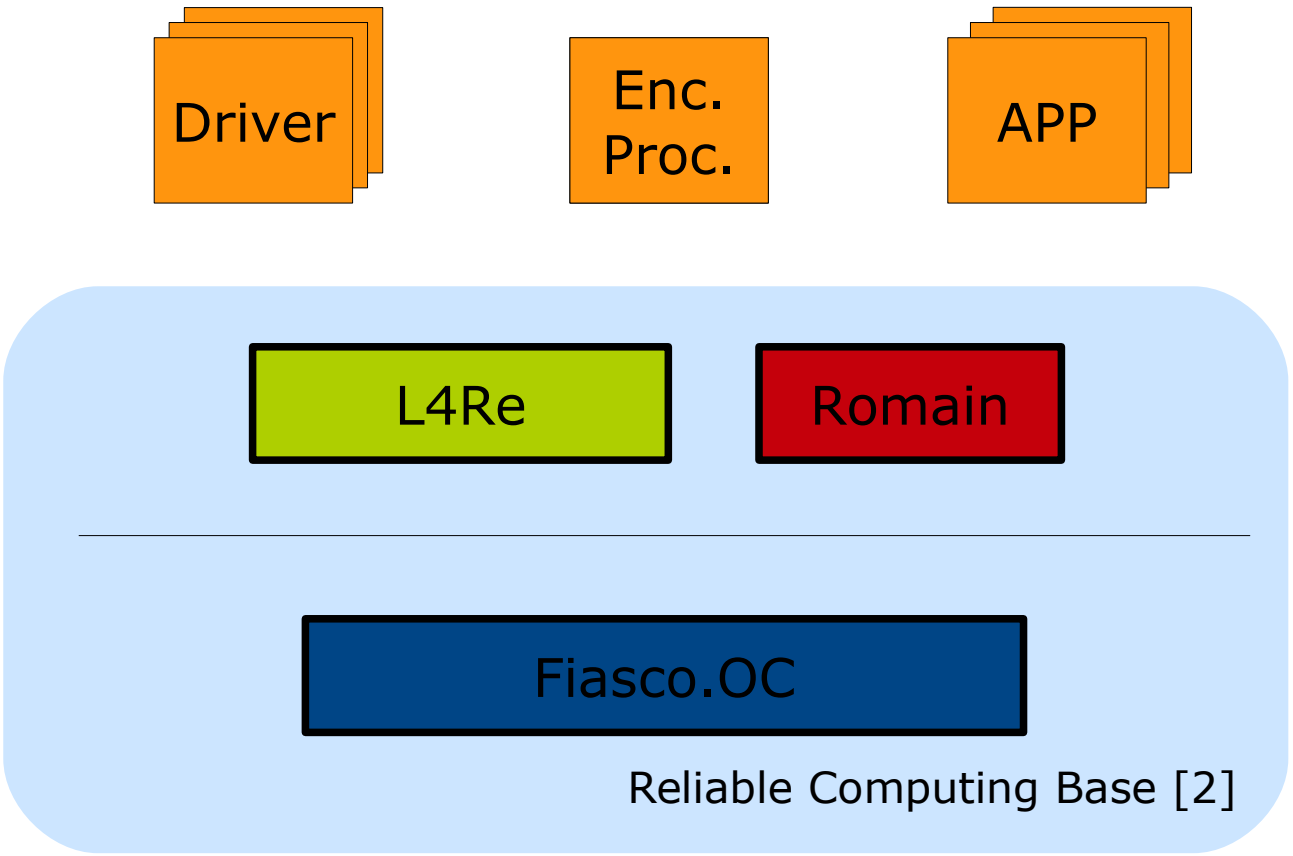
Secondary Effects: Cache Miss Rates

Benchmark	DMR misses		TMR misses	
	L2	L3	L2	L3
429.mcf	2,600 → 2,600	1,300,000 → 930,000	11,000,000 → 11,000,000	5,200,000 → 3,600,000
462.libquantum	2,500 → 2,500	570 → 323	440,000 → 385,000	387,000 → 8,700
471.omnet++	270,000 → 290,000	6,900,000 → 5,500,000	35,000,000 → 34,900,000	21,200,000 → 16,400,000

SPEC INT: Improved L3 Miss Rates



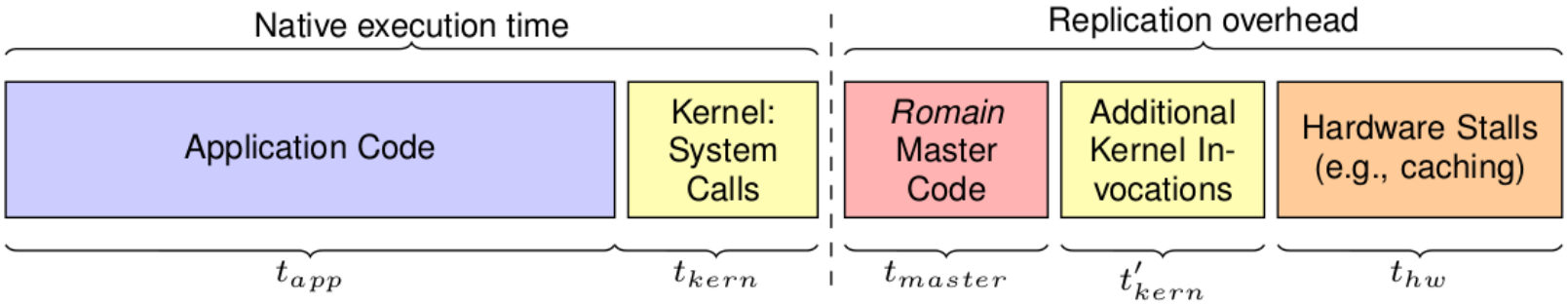
And now for something slightly different...



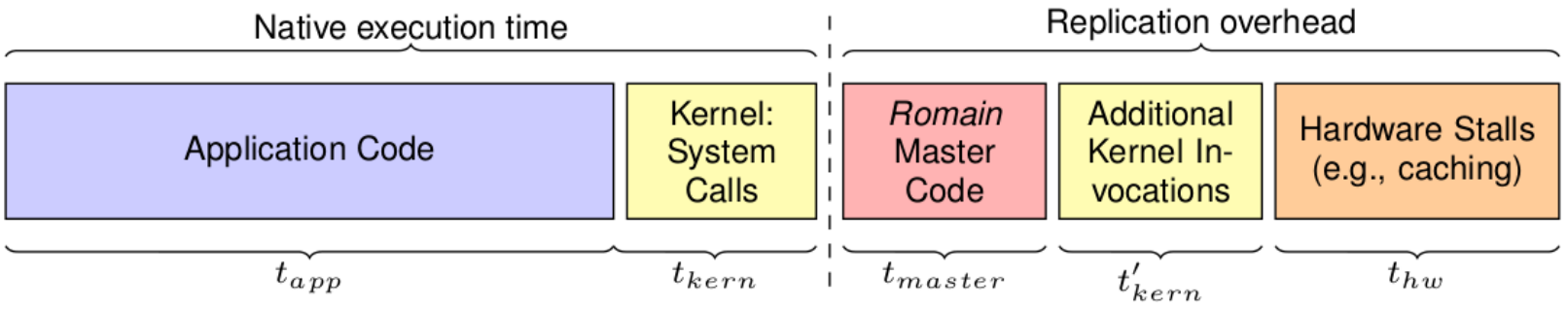
[2] Engel, Döbel: *The Reliable Computing Base – A new Paradigm ...*, SOBRES 2012

Protecting the RCB

- We have full RCB source, so we can apply compiler-level techniques.
- Encoded compiler anyone?
  → Approximate Overhead



Protecting the RCB: 2nd try



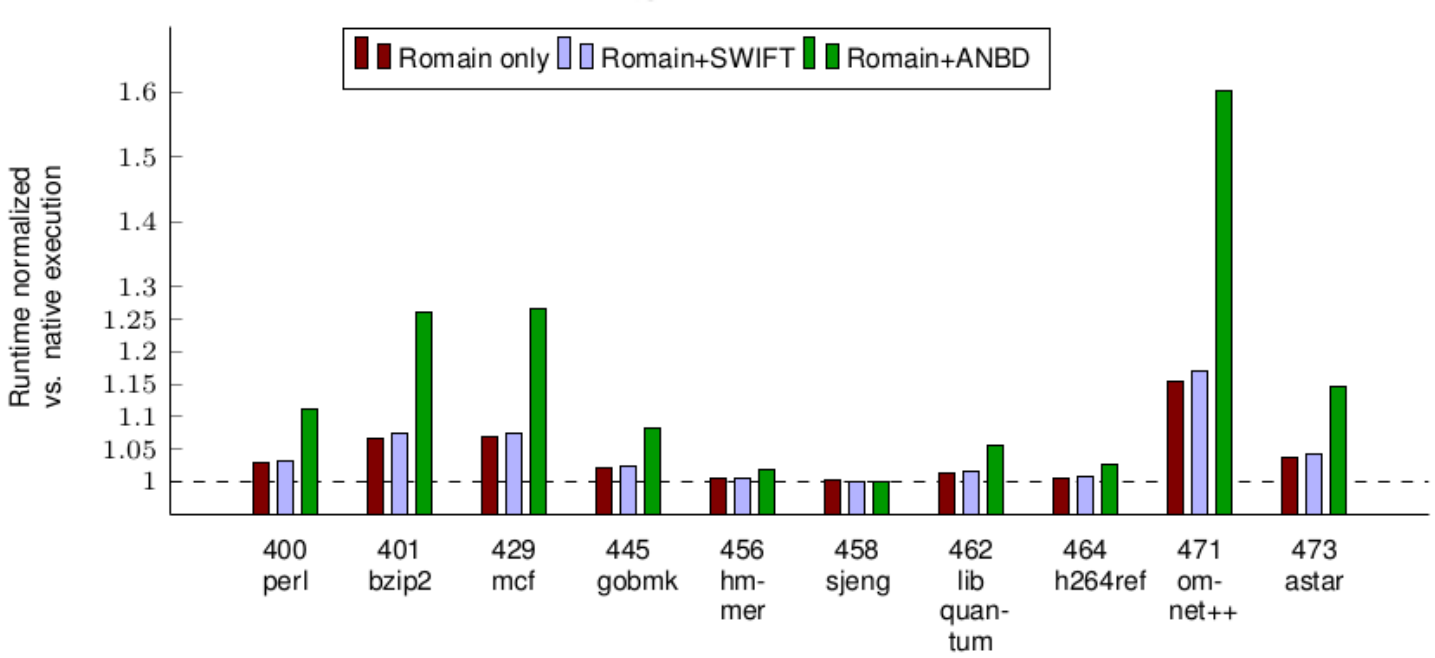
$$t_{prot} := t_{app} + C * (t_{kern} + t_{master} + t_{kern'}) + t_{hw}$$

$$t_{hw} = t_{kern} = t_{kern'} = 0$$

$$\mathbf{t_{prot} := t_{native} + C * t_{Replicated}}$$

Protecting the RCB: 2nd try

- Selected values for C [3]:
  - $C_{SWIFT} = 1.09$
  - $C_{ANBD} = 3.89$



[3] Schiffel, Schmitt, Süßkraut, Fetzer: *Software-Implemented Hardware Error Detection: Costs and Gains*, DEPEND 2010

Summary

- Romain: <5% overhead for 3x replicating most of the SPEC INT 2006 benchmarks
- Replica-core placement matters
- RCB protection may add additional overheads
  - **Combination with compiler-level methods seems feasible**