Multiprocessor scheduling, part 1 -Challenges-

Pontus Ekberg

2018-10-03

What is a multiprocessor?

- Simplest answer: A machine with >1 processors!

  ⟶ In scheduling theory, we include *multicores* in this definition

- Multiprocessors allow us to run >1 threads at the same time

  ⟶ Great for Size, Weight and Power (SWaP) efficiency

Multiprocessor scheduling

- Multiprocessor scheduling has been studied for many decades, also for real-time systems

  $\longrightarrow$ The *multicore revolution* has made it highly relevant today!

Multiprocessor scheduling

- Multiprocessor scheduling has been studied for many decades, also for real-time systems

  ⟶ The *multicore revolution* has made it highly relevant today!

- Multiprocessor scheduling for real-time systems is *hard*

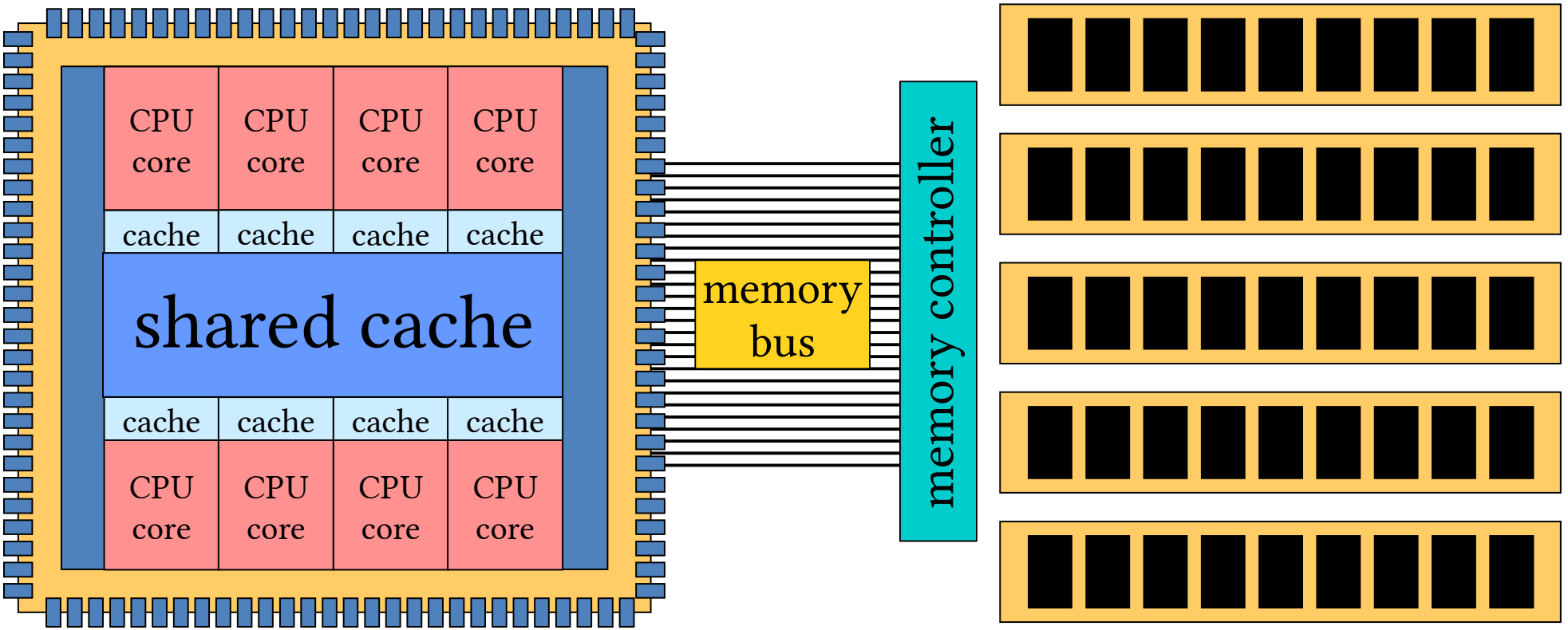
  ⟶ It is difficult to find good scheduling algorithms and schedulability tests

  ⟶ There are many huge practical challenges with using multicore processors in real-time systems

  ⟶ Lots of open problems (*exciting!*)

What is a multicore processor?

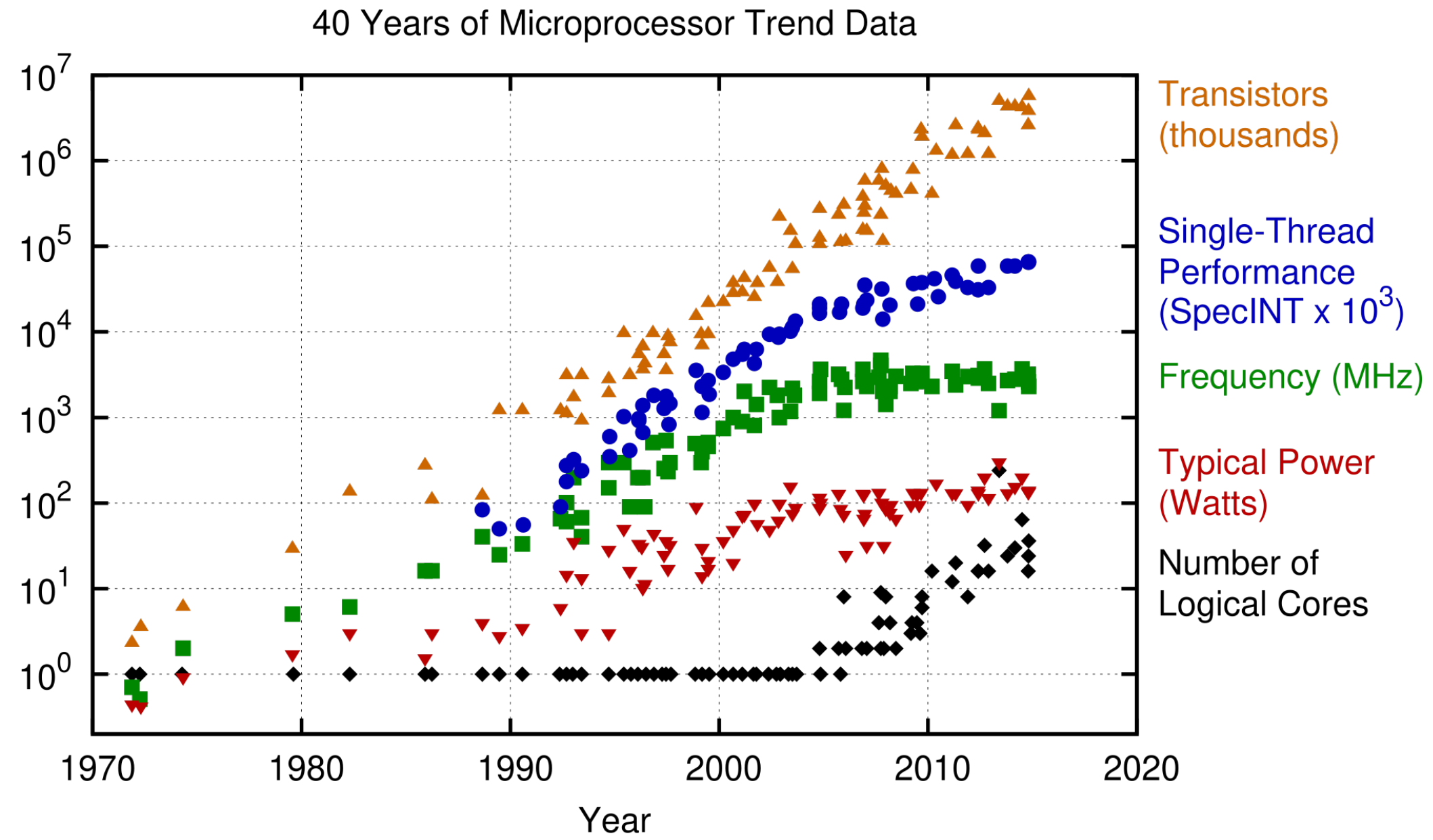
Multicore ≈ Tightly coupled CPU cores that share a memory system



Multicore processor

off-chip memory (DRAM)

Why do we use multicores?



Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten
New plot and data collected for 2010-2015 by K. Rupp

Caches

- DRAM performance has increased much slower than CPU performance

  ⟶ Relatively speaking, memory accesses become slower and slower

- Caches were introduced to *hide* this speed gap

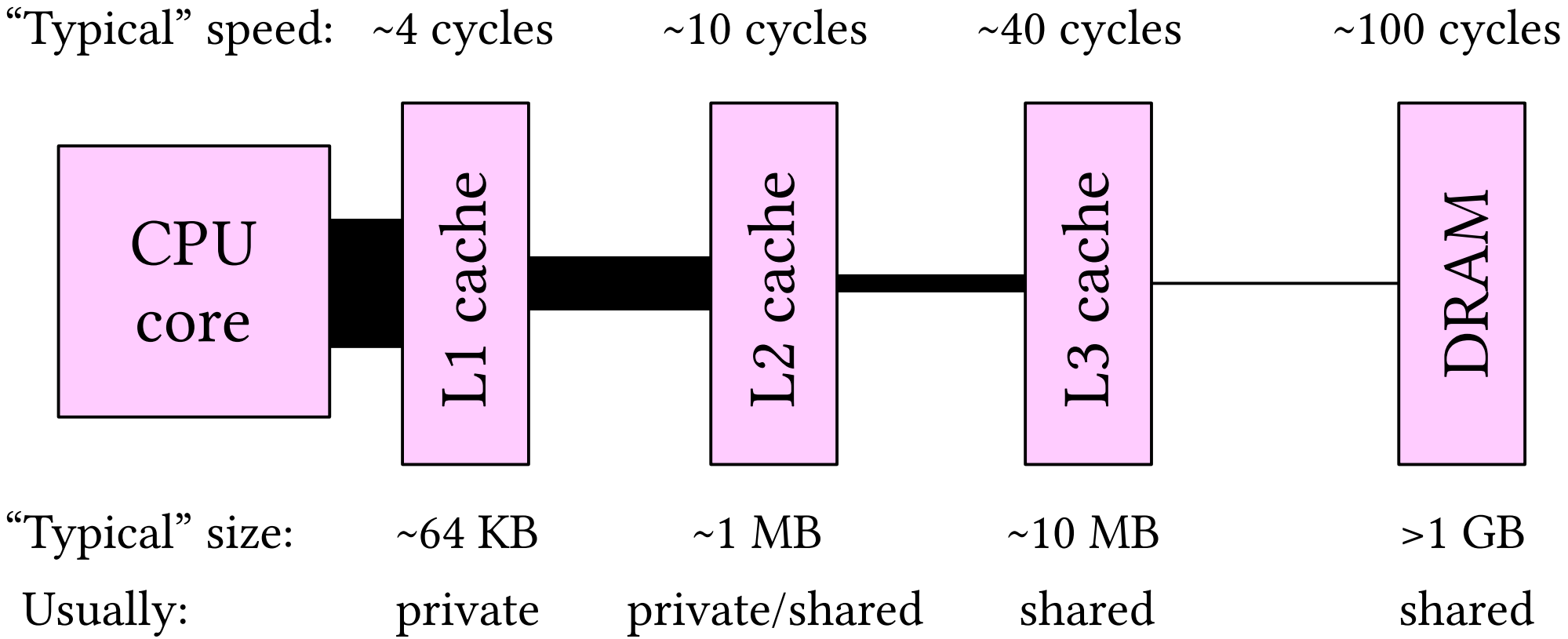
  ⟶ By storing (caching) data the processor *believes will be used* soon in a smaller and faster memory

  ⟶ Works well because typical programs exhibit temporal and spatial memory locality

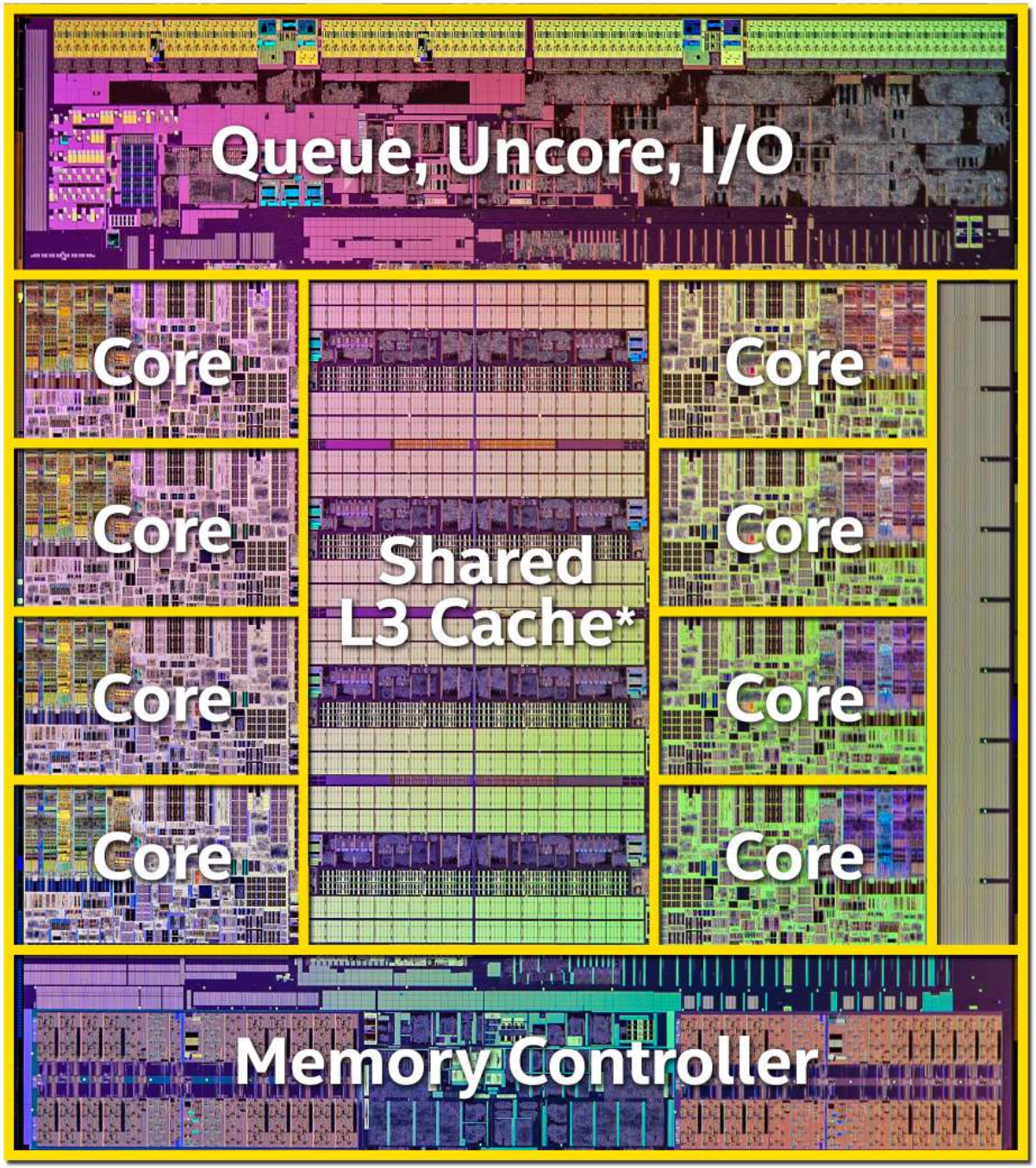
Memory hierarchy

- The further you go in the memory hierarchy, the slower your accesses will be

  → Caches are hugely important for performance



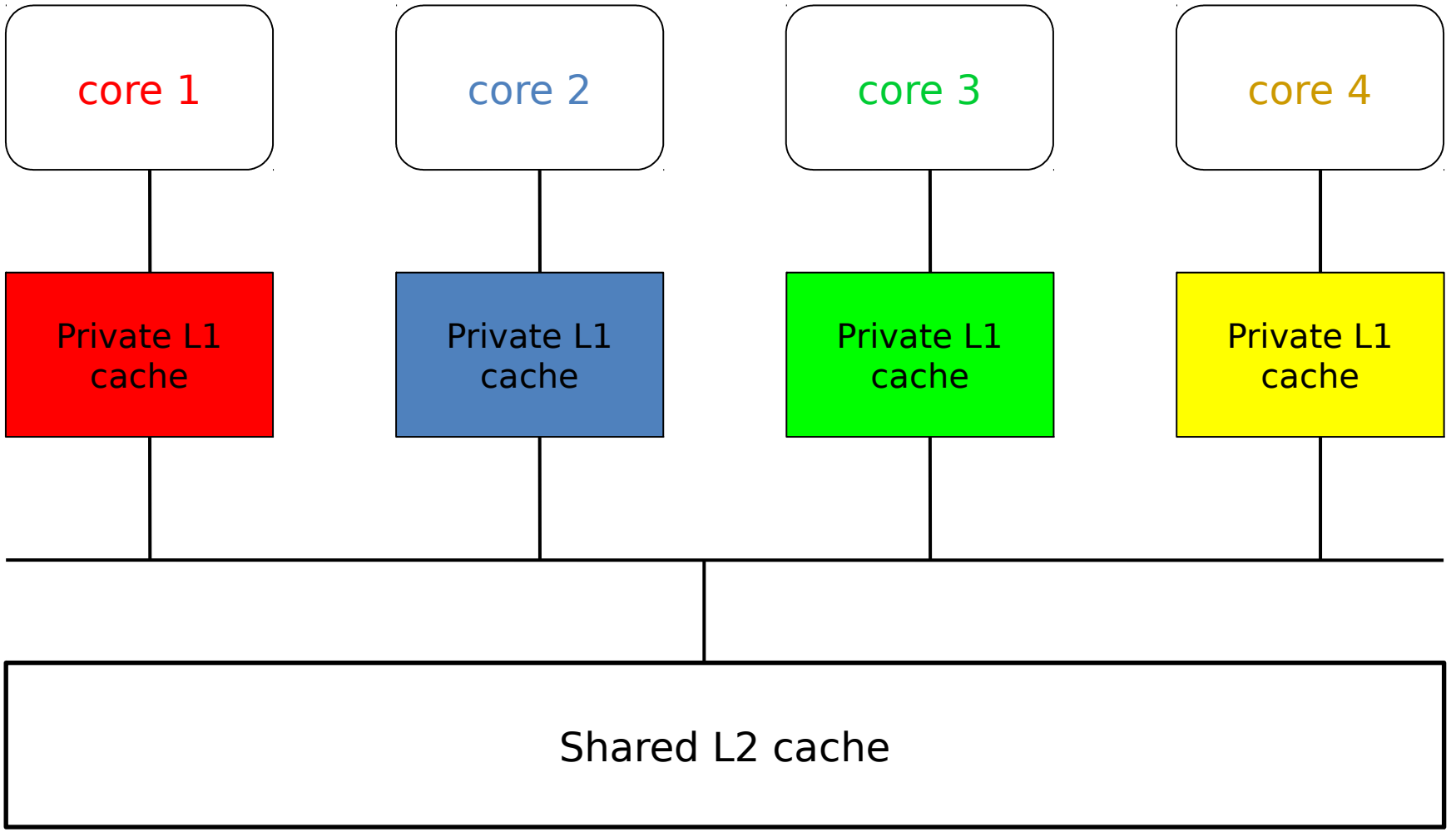
Intel Core i7 die



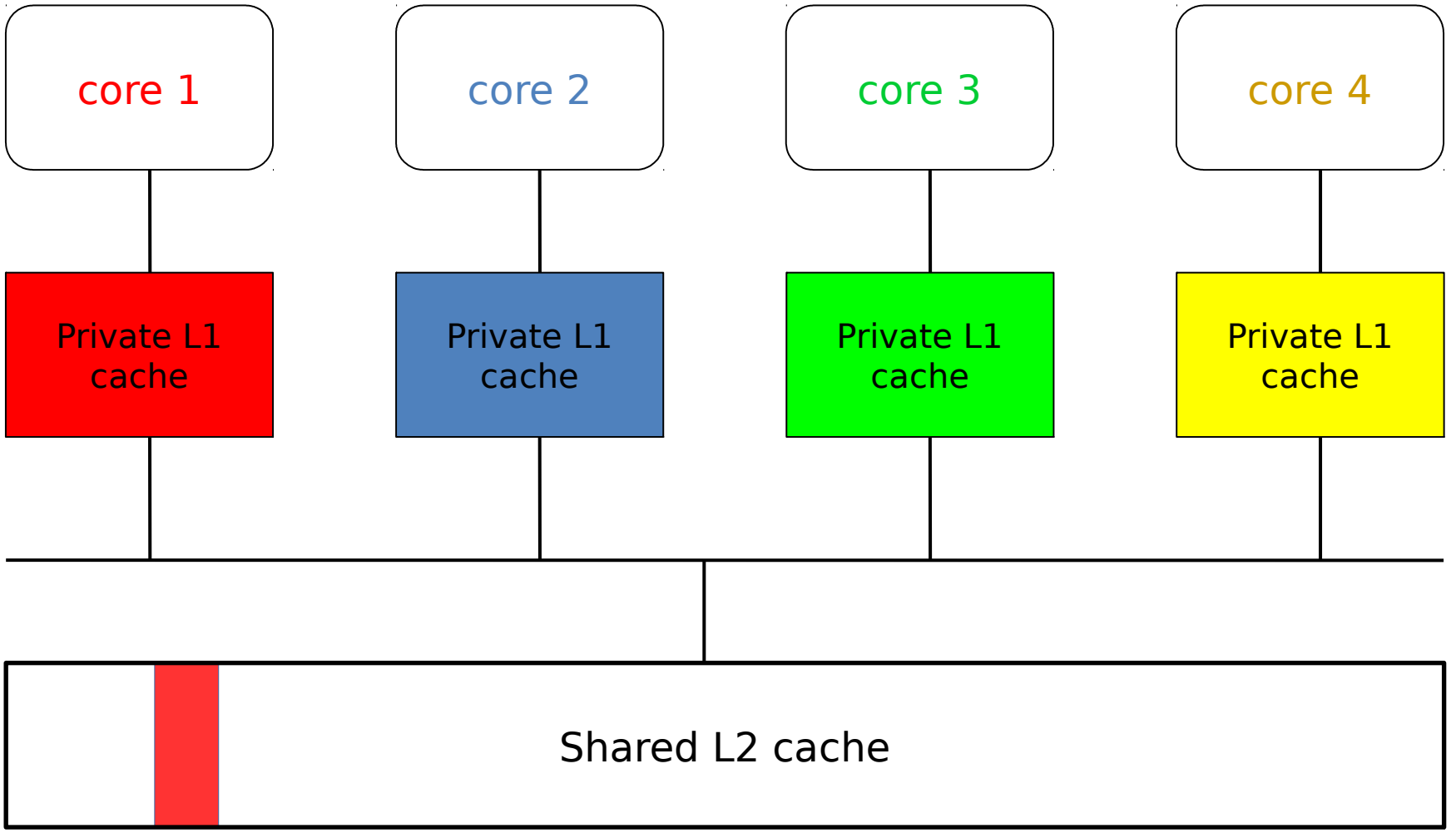
For real-time systems

- Caches are designed for good average-case performance, *not for predictability*
- Safely predicting cache hits in *private cache* is very difficult (though we have seen remarkable progress)
- Safely predicting cache hits in *shared cache* is practically impossible
- Other micro-architectural features (memory controller, hyper-threading, shared buses, I/O etc.) also make it *very difficult to predict WCET* on multicores

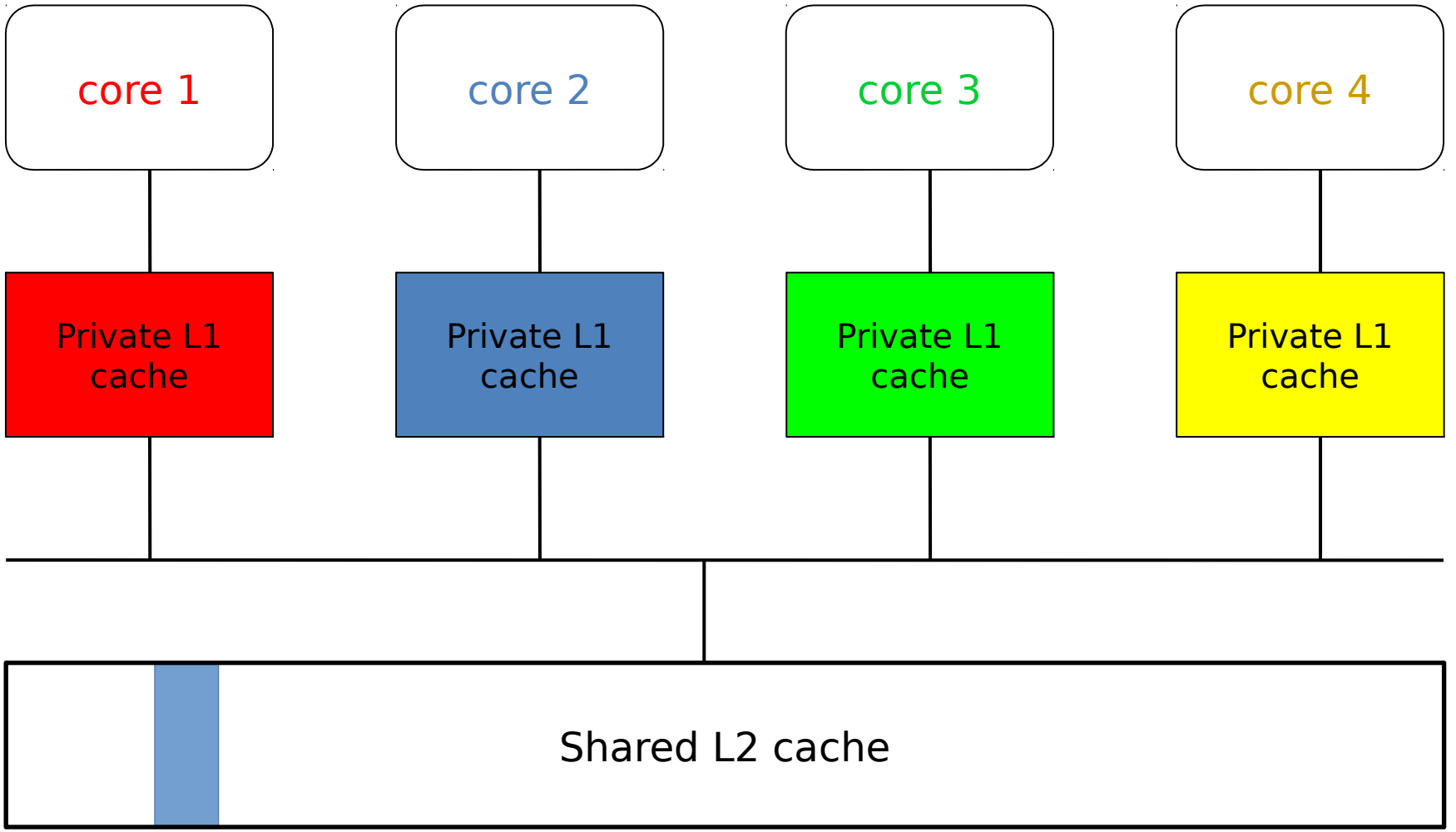
Interference on shared cache



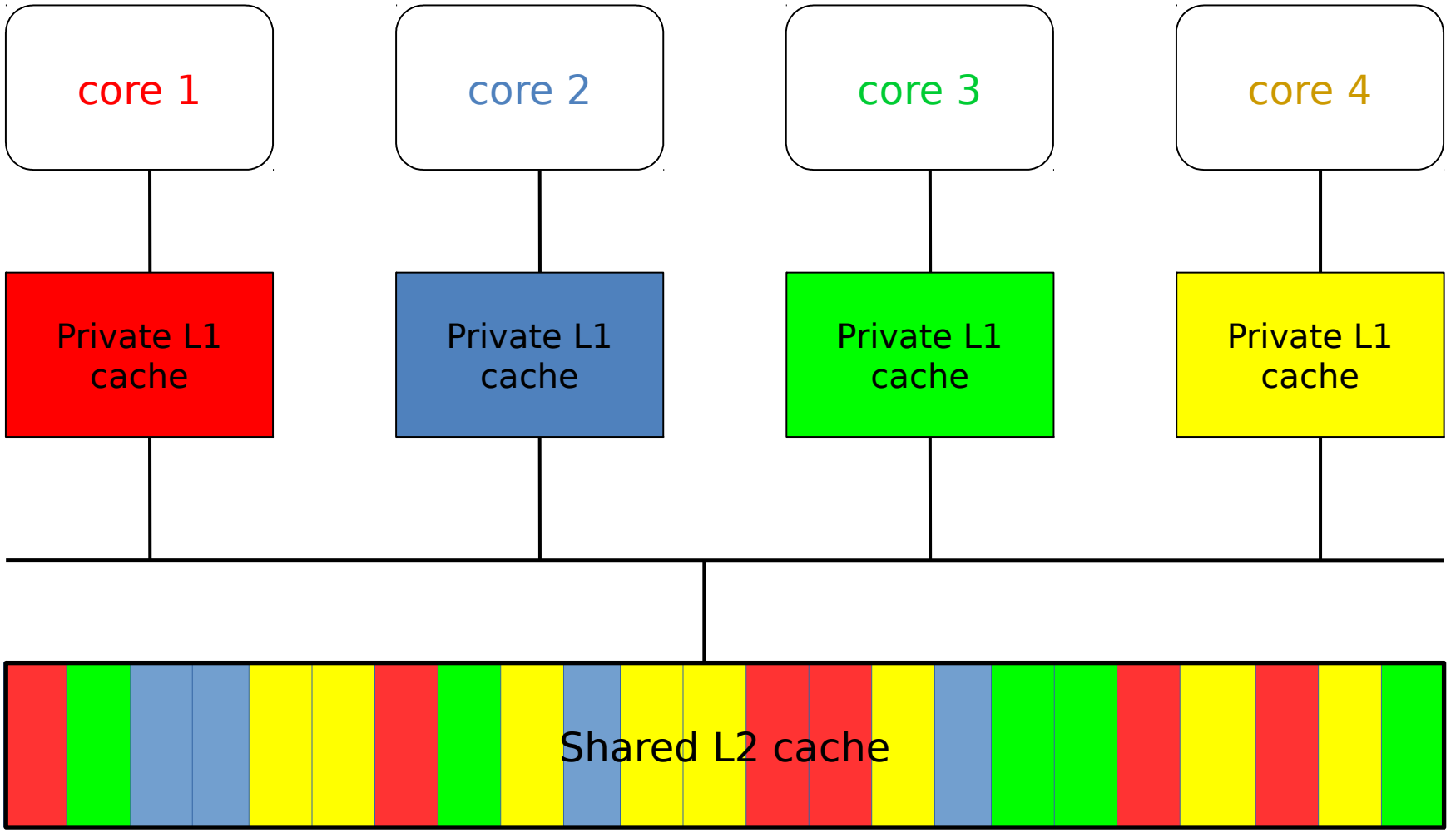
Interference on shared cache



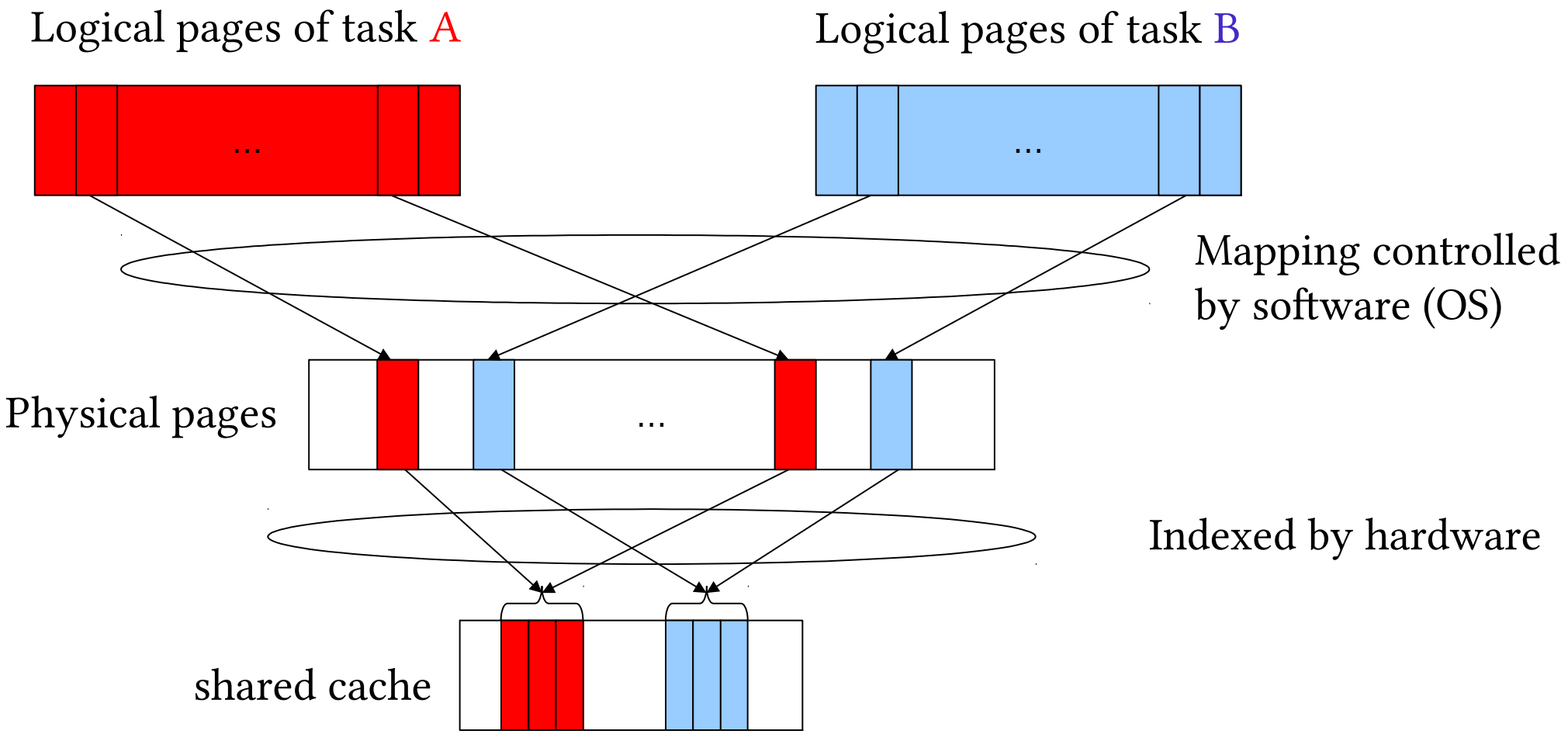
Interference on shared cache



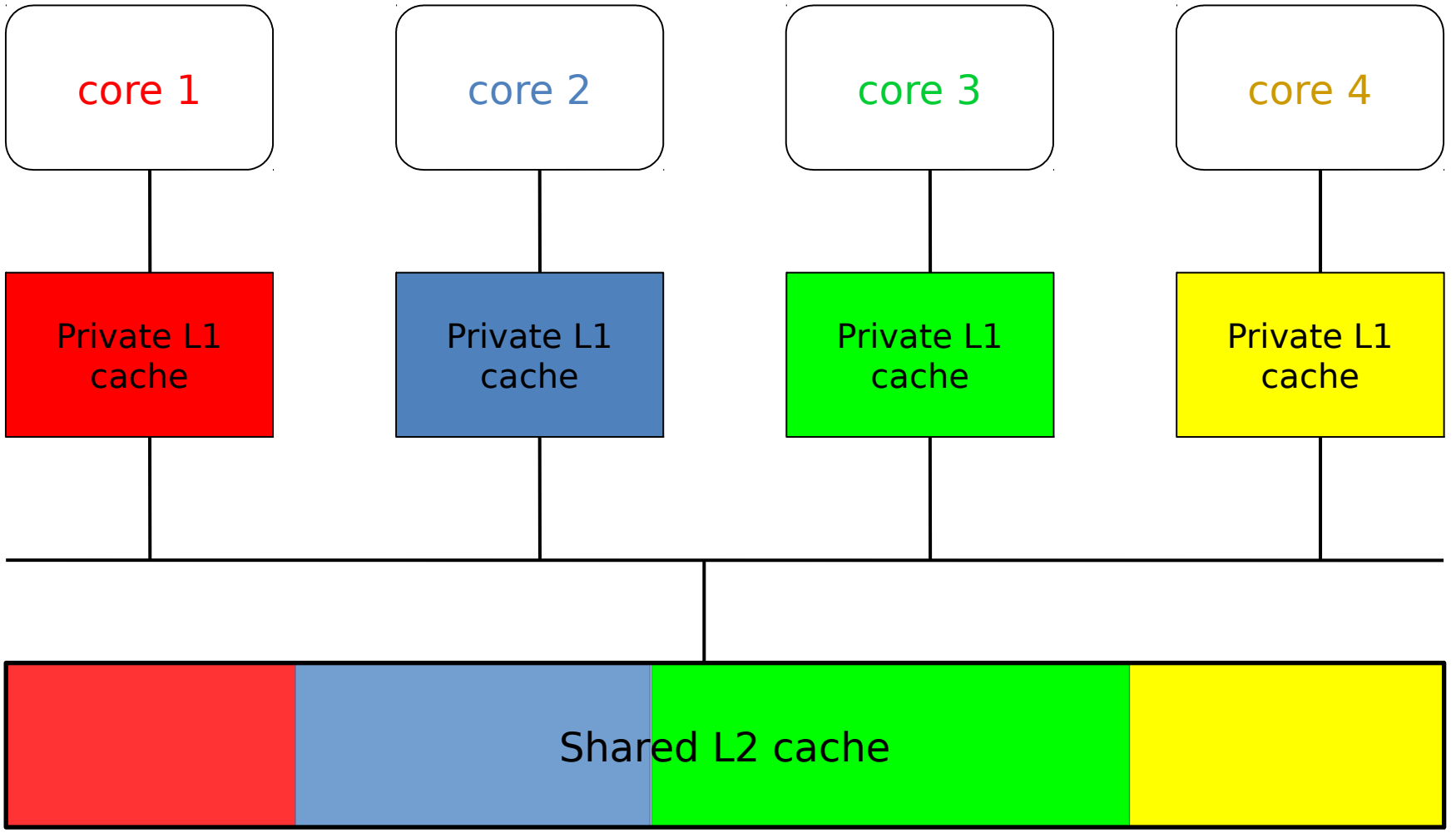
Interference on shared cache



Potential solution: cache coloring

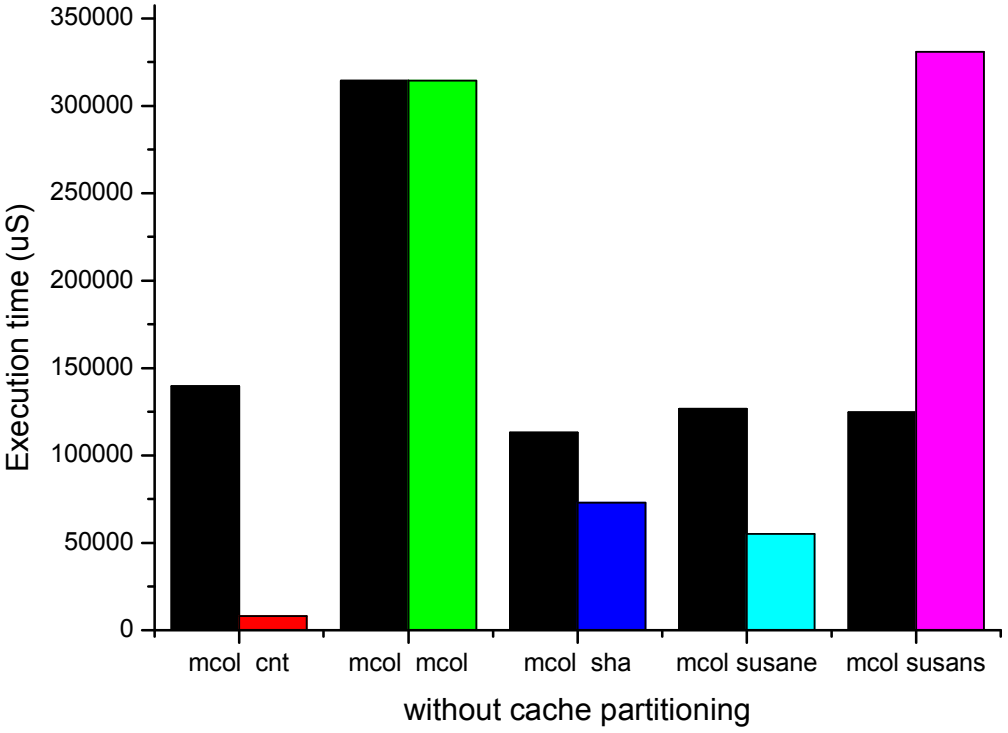


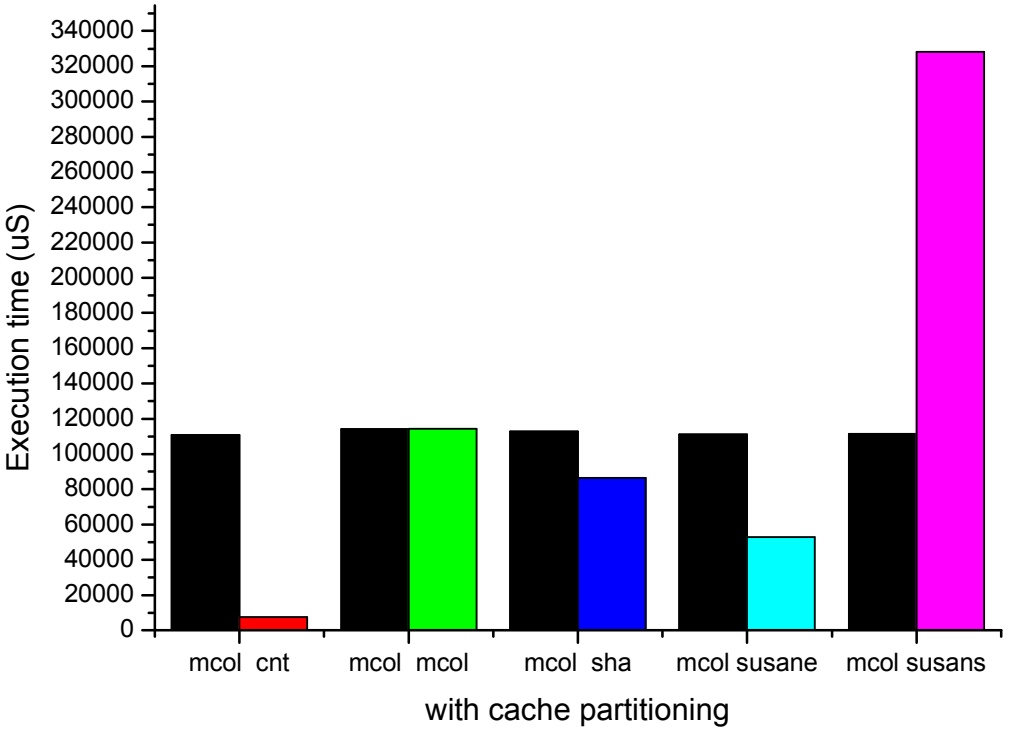
Potential solution: cache coloring



Case study

- A simple experiment on a dual-core Linux machine





A better cache solution?

- A better solution is likely to make the “cache” completely software controlled

  $\longrightarrow$ We know exactly what is in it!

A better cache solution?

- A better solution is likely to make the “cache” completely software controlled

  ⟶ We know exactly what is in it!
- A fast, small memory where the programmer (not HW) decides what goes into it is often called a *scratchpad memory* or *tightly coupled memory*

A better cache solution?

- A better solution is likely to make the “cache” completely software controlled

  ⟶ We know exactly what is in it!

- A fast, small memory where the programmer (not HW) decides what goes into it is often called a *scratchpad memory* or *tightly coupled memory*

- Not common in COTS CPUs, but there are exceptions, like the *ARM Cortex-R* series

  ⟶ Cortex-R also has other nice features: ECC on caches and buses, lock-step mode for cores etc.

Solutions to shared bus issues?

- Buses can often be made predictable using *time-division multiple access* (TDMA)

  ⟶ But this is often inefficient

Solutions to shared bus issues?

- Buses can often be made predictable using *time-division multiple access* (TDMA)

  ⟶ But this is often inefficient

- A better solution is to use a special-purpose memory controller, for example *priority-aware*

  ⟶ But then someone has to build it!

Solutions to shared bus issues?

- Buses can often be made predictable using *time-division multiple access* (TDMA)

  ⟶ But this is often inefficient

- A better solution is to use a special-purpose memory controller, for example *priority-aware*

  ⟶ But then someone has to build it!

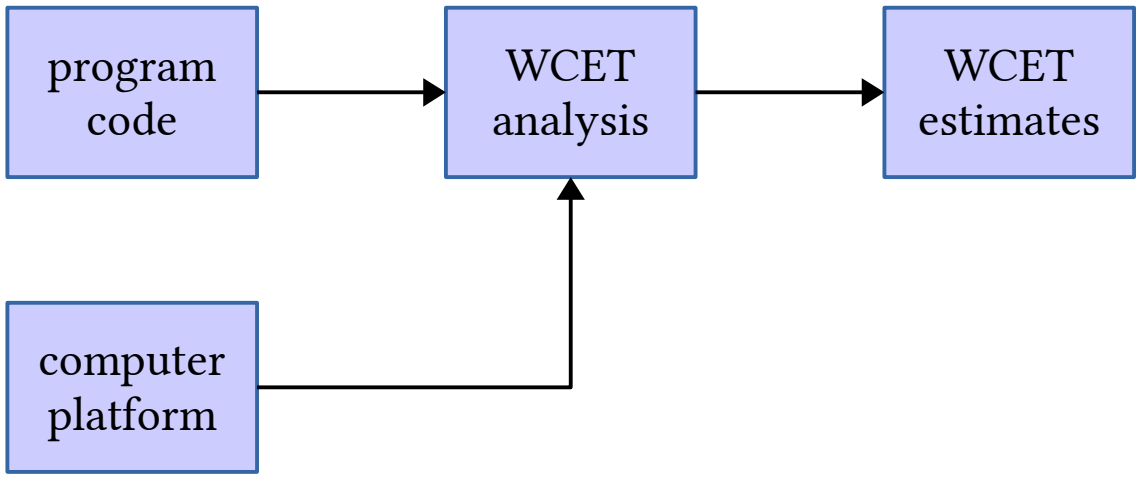
- By exploiting performance measurement registers, it is sometimes possible to implement memory accesses budgeting in software

The WCET-scheduling interface

Inter-core interference on multicores *break* the usual WCET-scheduling interface!

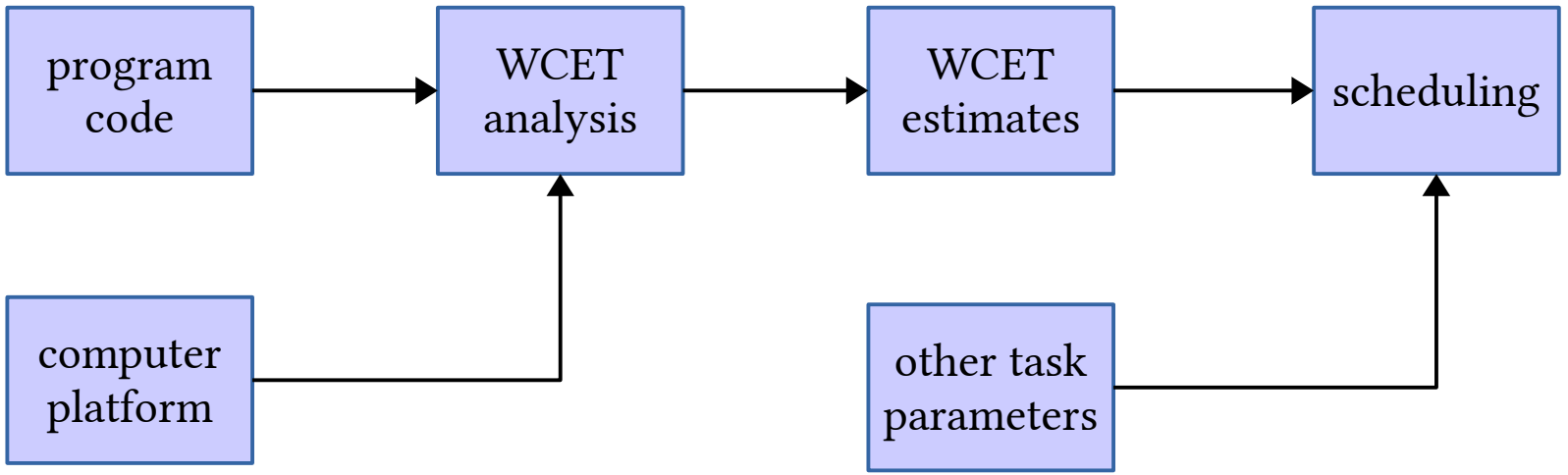
The WCET-scheduling interface

Inter-core interference on multicores *break* the usual WCET-scheduling interface!



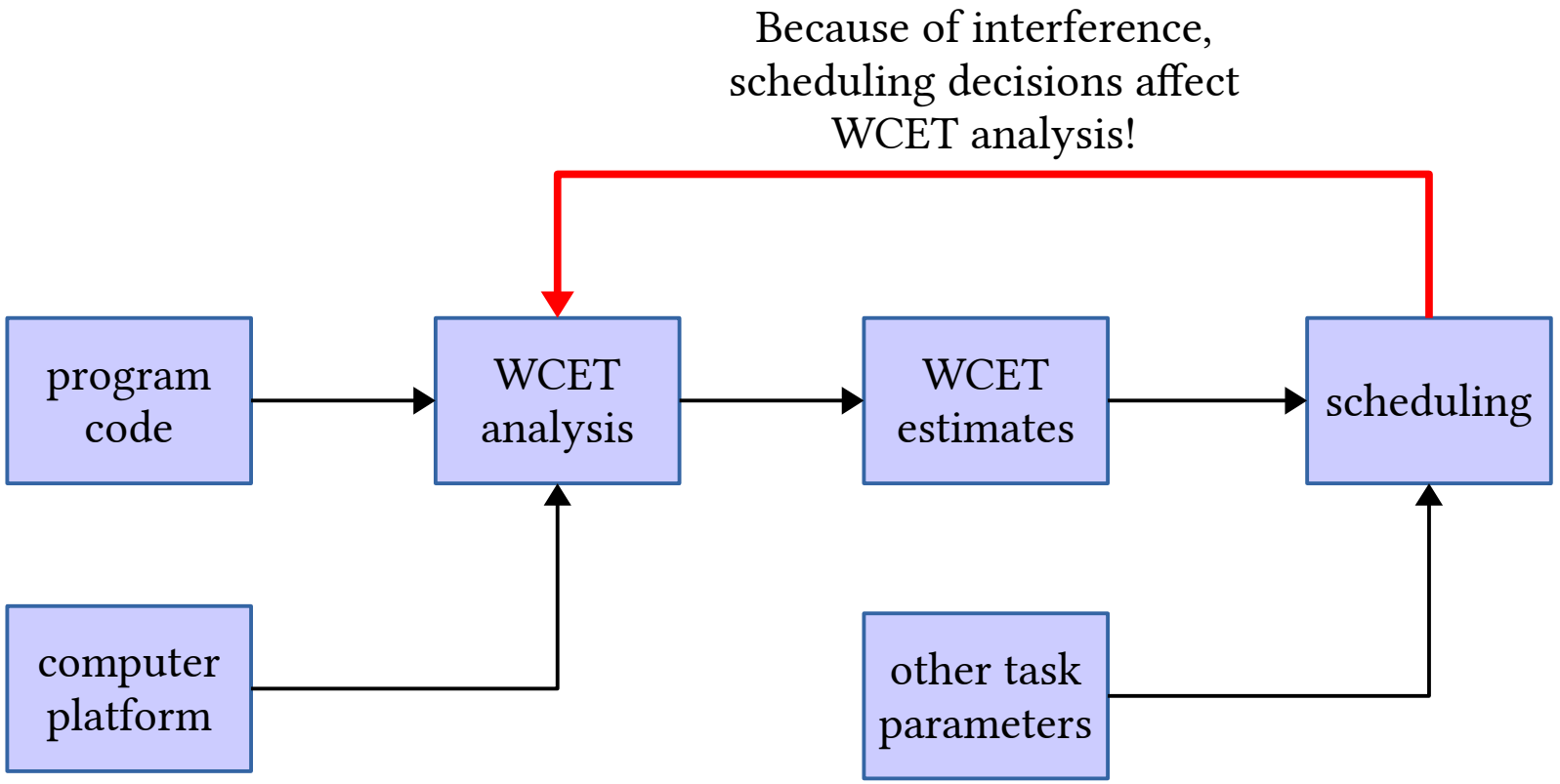
The WCET-scheduling interface

Inter-core interference on multicores *break* the usual WCET-scheduling interface!



The WCET-scheduling interface

Inter-core interference on multicores *break* the usual WCET-scheduling interface!



Multiprocessor scheduling: Themes