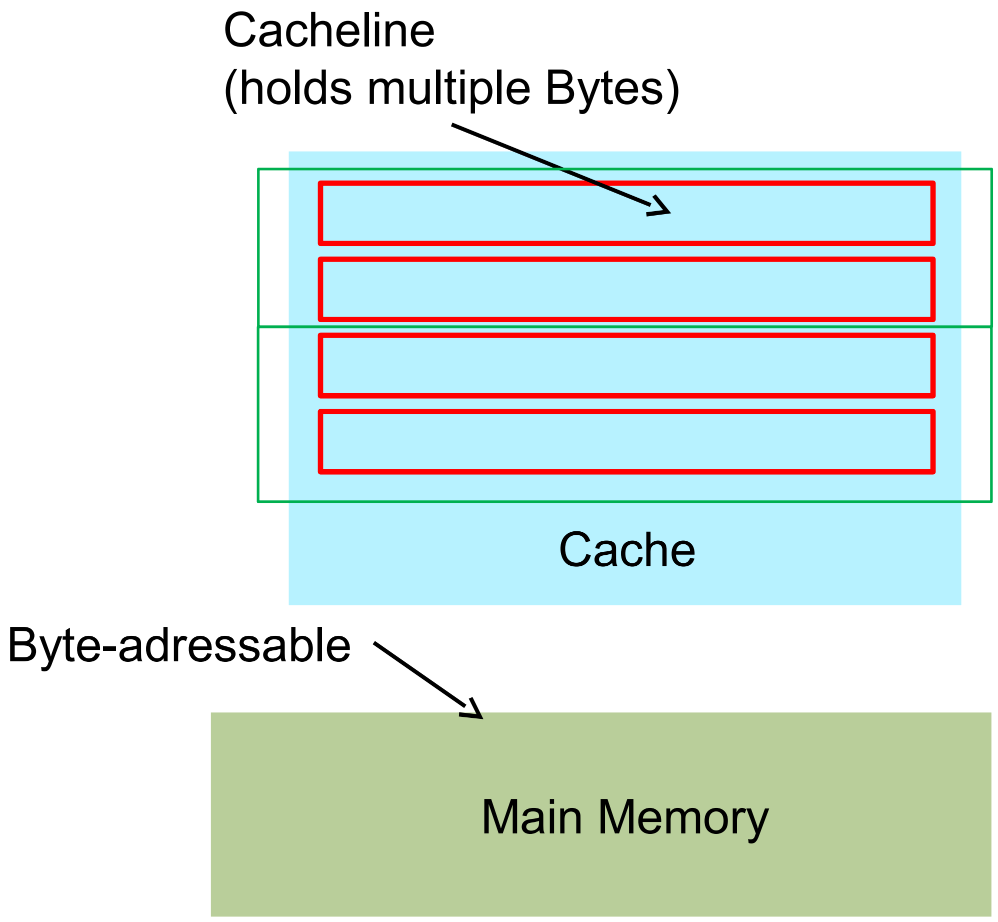
TIMO SCHNEIDER <TIMOS@INF.ETHZ.CH>

DPHPC Recitation Session 4
Sequential Consistency

Homework – False Sharing



int a;
int b;

What if core 0 writes a and core 1 wrtites b and both variables are in the same cache line?

CL has to be flushed for each write, even though no data is shared!

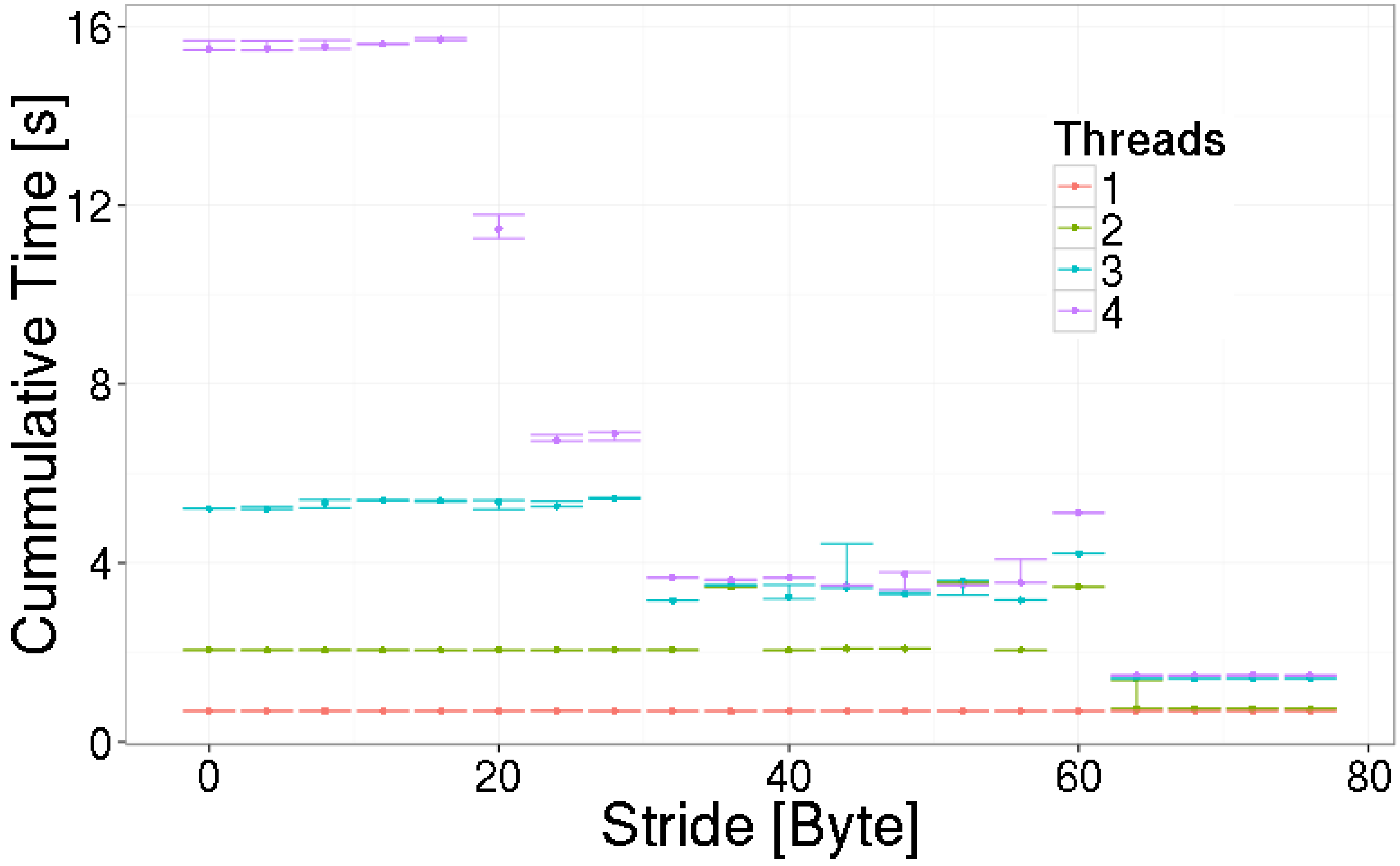
This phenomenon is called False Sharing.

Homework – False Sharing Benchmark

- **Idea: Allocate uint8_t array a, let core 0 write to a[0] and core 1 to a[x]**
- **If x is larger than the size of one CL this should be “fast” because both cores operate on their on cached copy of different CLs**
- **If x is smaller than one CL it will be slow, due to false sharing**

- **In practice it is a bit harder to get it right :)**
  - If we write only once it might not really be parallel -> do it in a large enough loop
  - If we write only one Byte in each iteration we will not see much because of loop overhead (incrementing counter, jump) -> write 8 bytes in inner loop
  - Make sure the compiler does not “optimize” your loop by removing it!

Homework – False Sharing Benchmark



Machine: Intel Core i5 3230M
Compiler: gcc 4.9.1 –O3 –fopenmp –std=gnu11

On Benchmarking / Plots

- **Make sure you can explain your data!**
- **Plots should**
  - have labels + units on x and y axis
  - have legends or a description of each line/color
  - some indication of accuracy of measurements
  - do not measure only once or show only the minimum!

- **You can make plots with many different software packages, my personal favorite is GNU R**
  - Free Software
  - Includes many statistic / data-analysis functions
  - Is probably harder to learn than Excel/Gnuplot, but generates nicer plots

Sequential Consistency

Consistency vs. Coherence

- Remember: Cache coherence guarantees:
  - Writes are eventually seen by other processes
  - All processes see writes to the same location in the same order
- So what does cache coherence tell us about writes to different locations?
  - Nothing!
  - That's why we need something more

Consistency: Why we need it

Consider this implementation of a lock: Does it work?

Process 1:

```
A = 1
if (B == 0)
    Enter CS
```

Process 2:

```
B = 1
if (A == 0)
    Enter CS
```