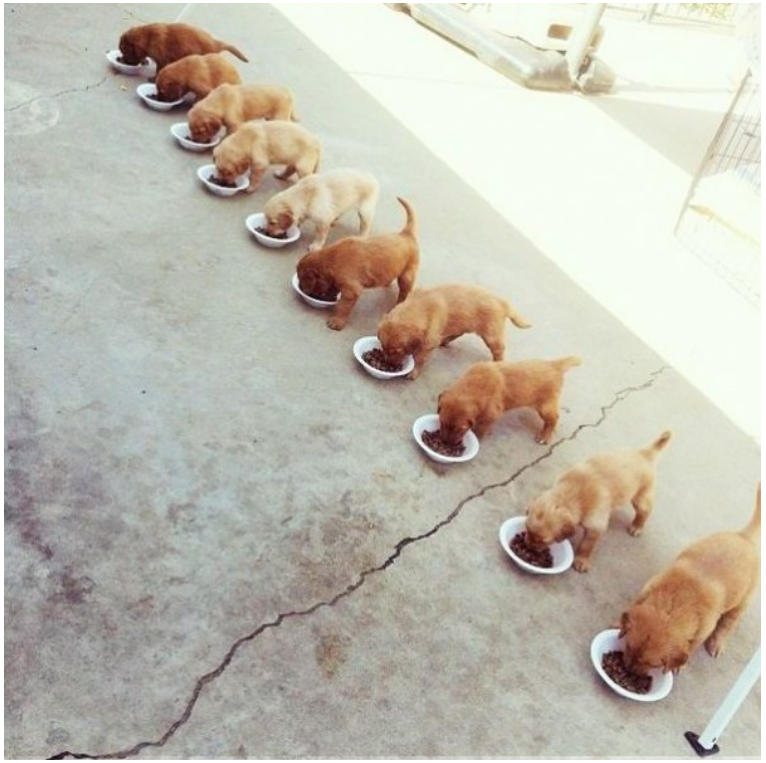
OpenMP 2

Multi-threaded Programming



theory

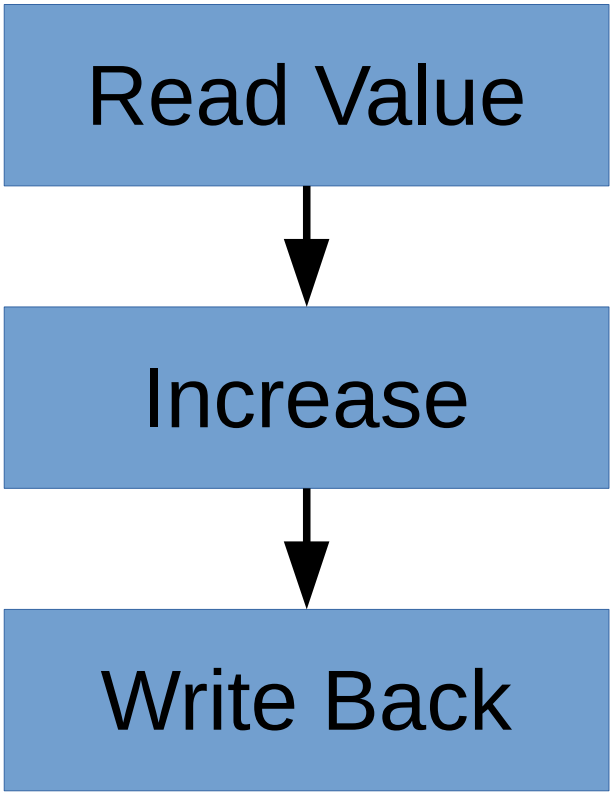
practice

University of BRISTOL

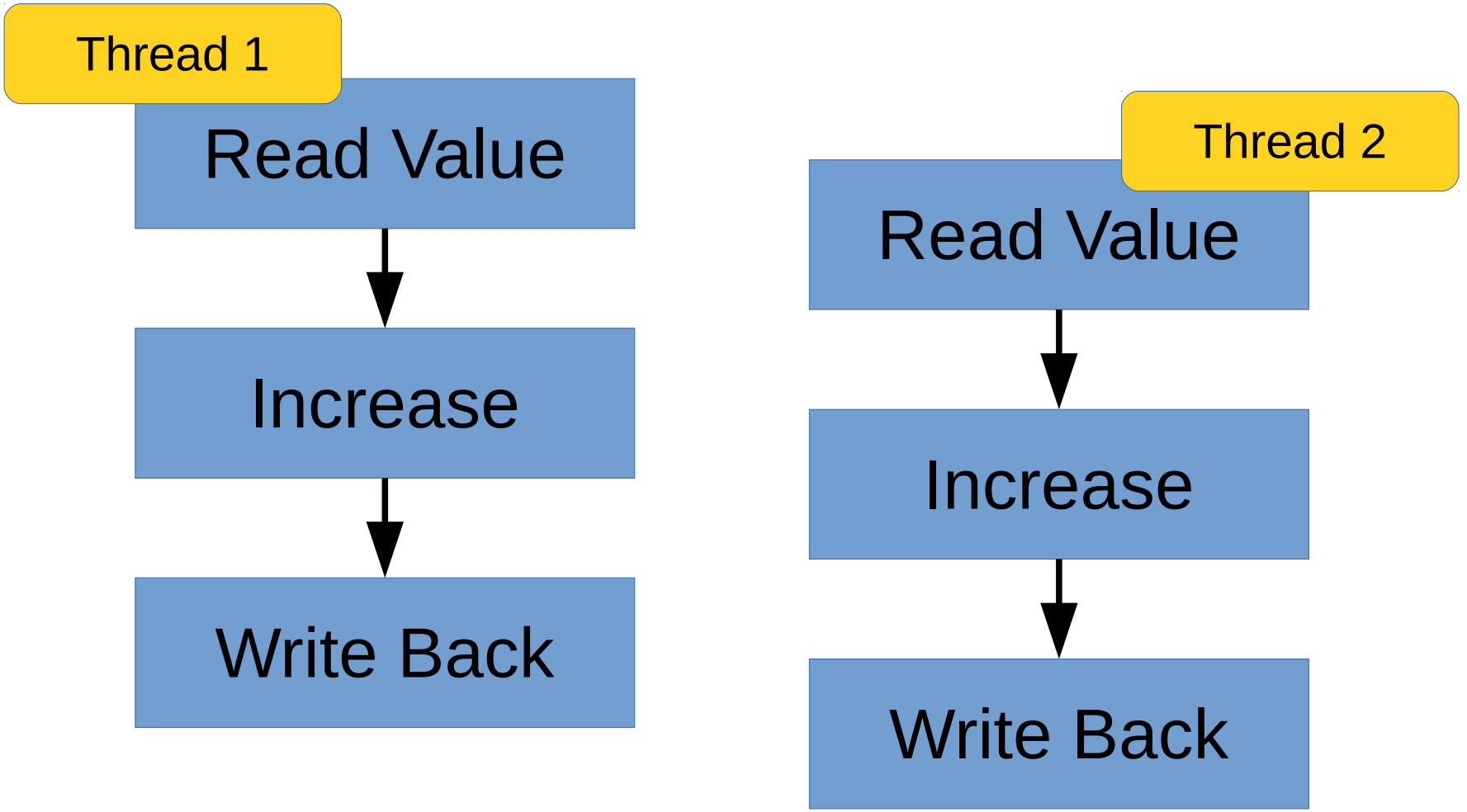
Overview

- Parallel programming – new opportunities! But also new bugs and new performance gotchas:
  - Race condition bugs, barriers and load balance.
- We'll consider several versions of a numerical integration example and review performance.
- Avoiding unwanted barriers.

A Serial Code Sequence



A Threaded Code Sequence



Parallel Bugs: Race Conditions

Thread 1	Thread 2		Integer Value
			0
Read value		←	0
Increase value			0
Write back		→	1
	Read value	←	1
	Increase value		1
	Write back	→	2

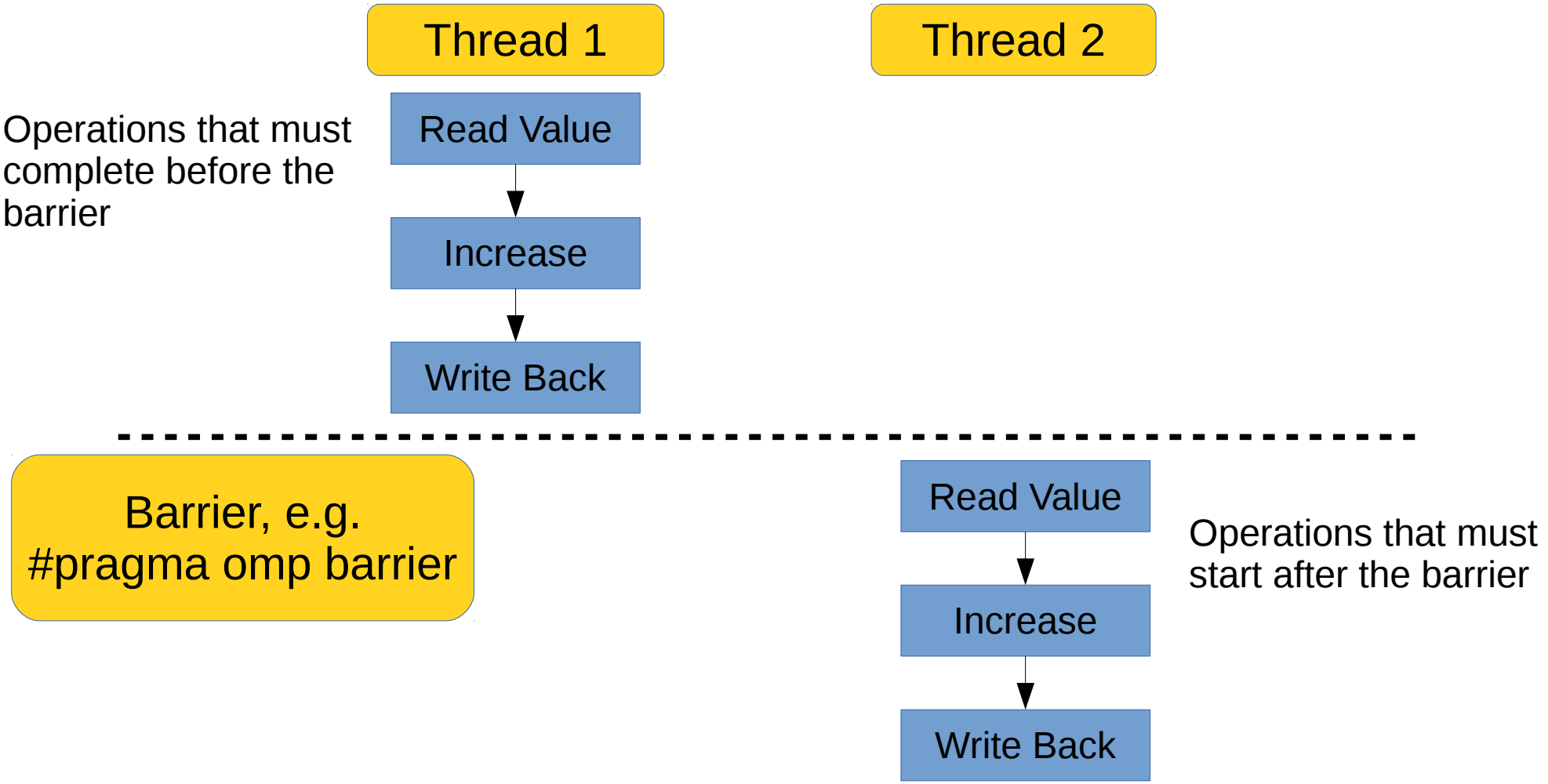
Intended

Thread 1	Thread 2		Integer Value
			0
Read value		←	0
	Read Value	←	0
Increase value			0
	Increase value		0
Write back		→	1
	Write back	→	1

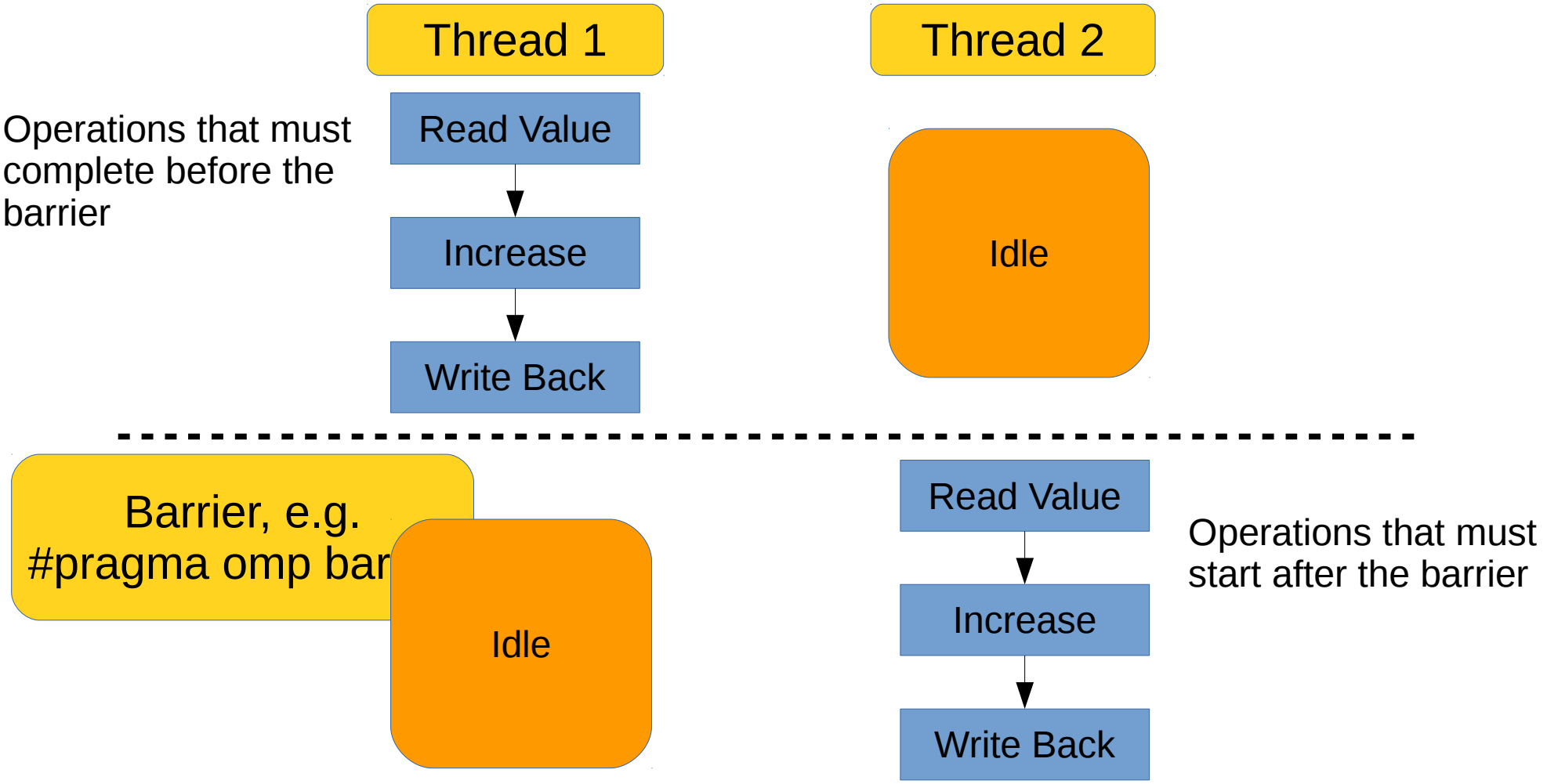
What we got

University of BRISTOL

Introducing a Barrier

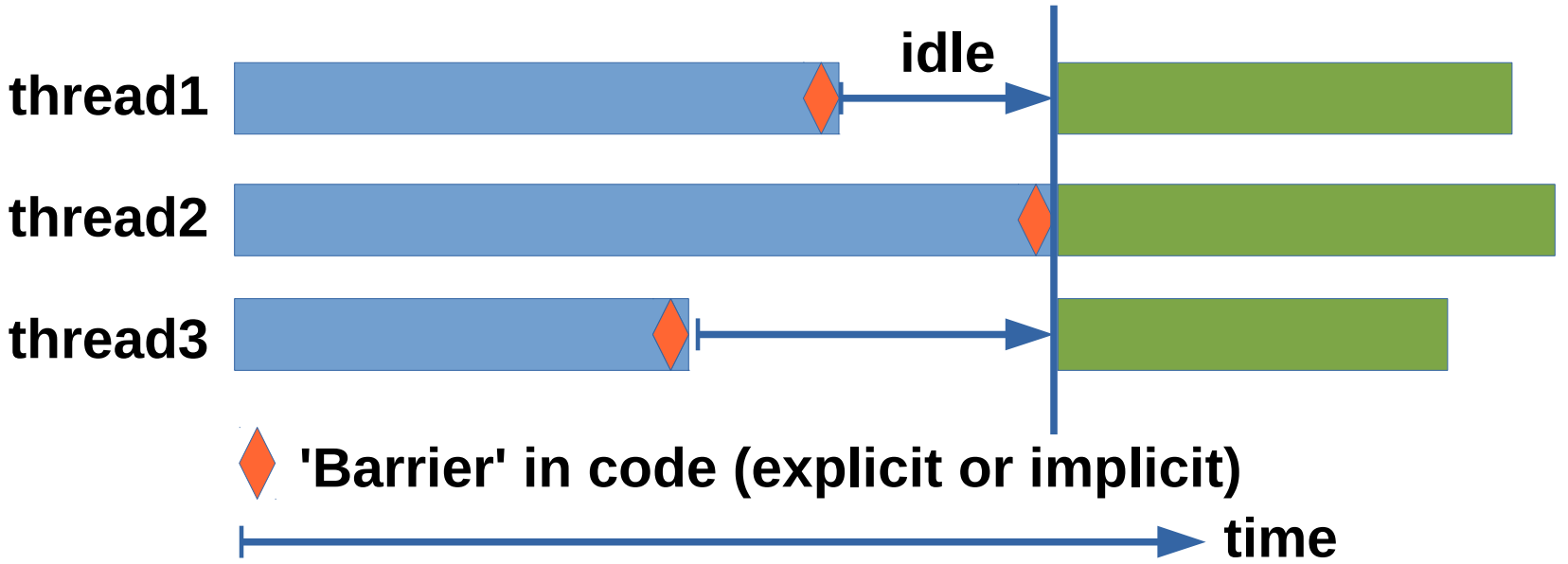


Introducing a Barrier



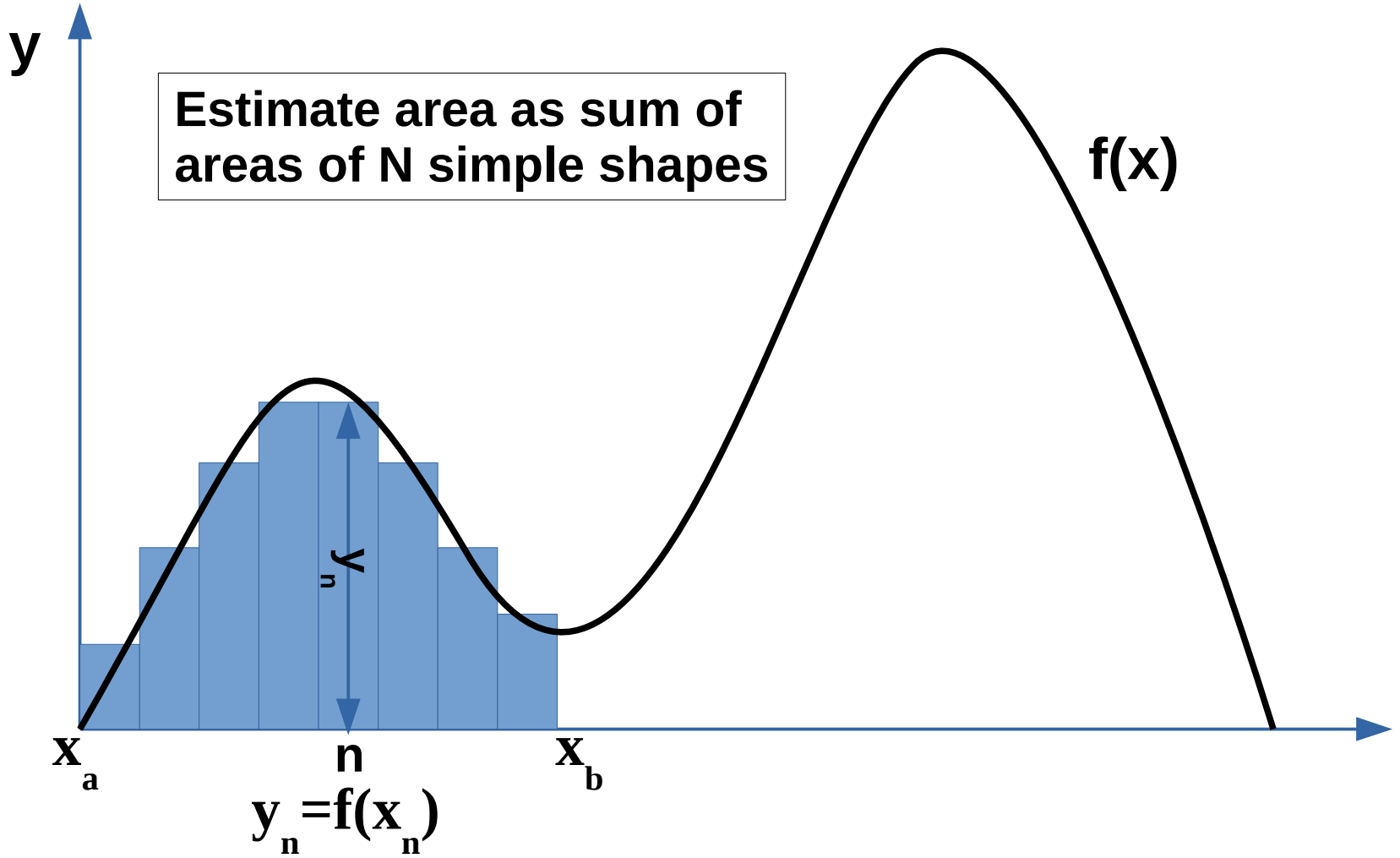
University of BRISTOL

Performance: Synchronisation & Load Balance



- We might use a barrier to avoid a '**race condition**' error. (e.g. https://en.wikipedia.org/wiki/Race_condition)
- **Parallel for loops have an implicit terminating barrier.**
- **A critical region or 'mutex' (perhaps to compute a total) will force a synchronisation**.

Example: numerical integration



(e.g. estimate pi by numerically integrating $4/1+x^2$)

University of BRISTOL

Shared Accumulator

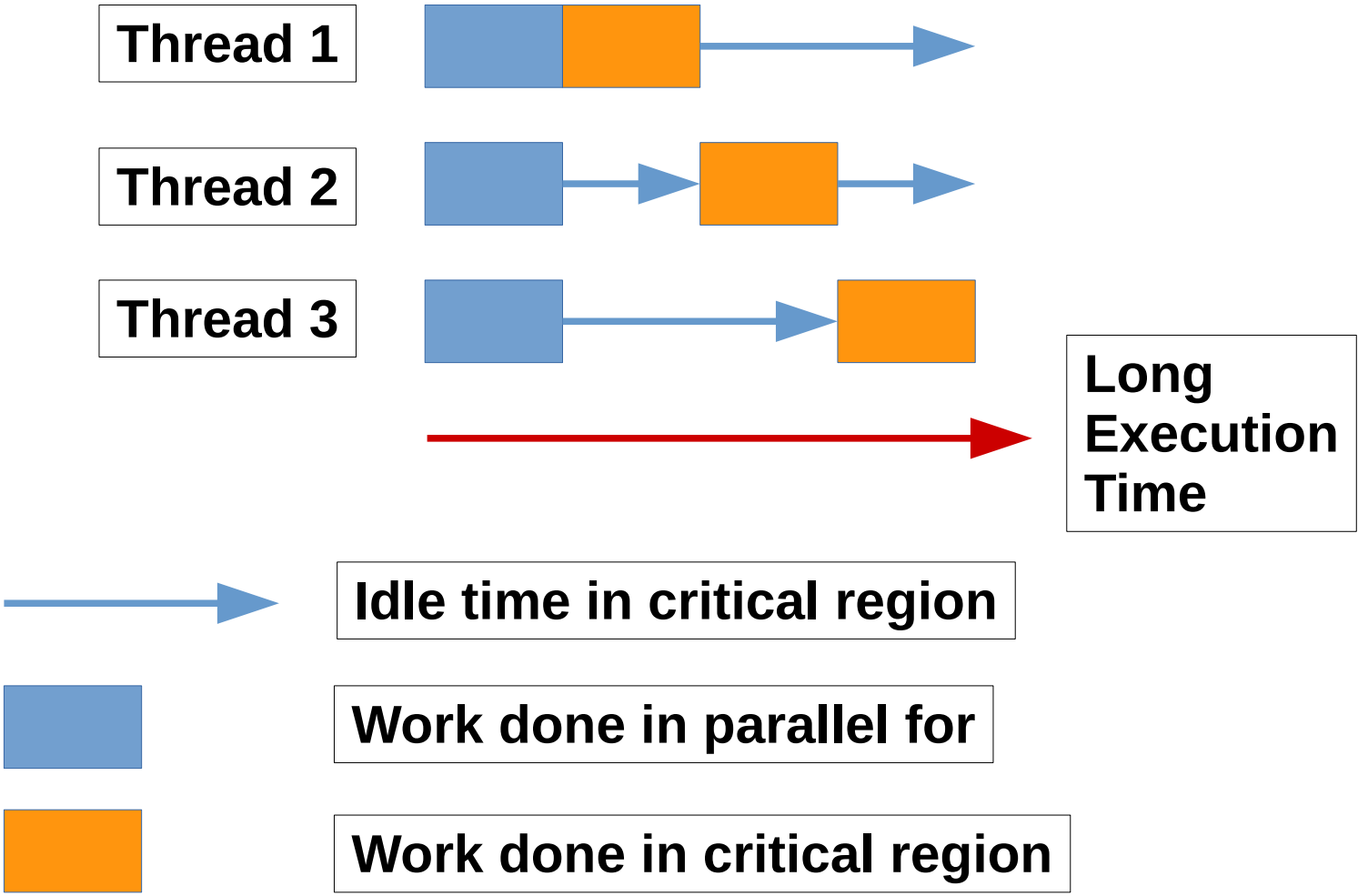
```
#pragma omp parallel for shared(sum) private(x)
  for (ii=0; ii<num_steps; ii++) {
    x = (ii+0.5)*step;
#pragma omp critical
    sum += 4.0/(1.0+x*x); /* all threads access! */
  }

  /* master thread only */
  pi = step * sum;

  printf("pi is:\t\t%.16f\n",pi);
  printf("error is:\t%.16f\n",fabs(pi – PI25DT));
```

Syntactically fine, but what's the problem?

Shared Accumulator



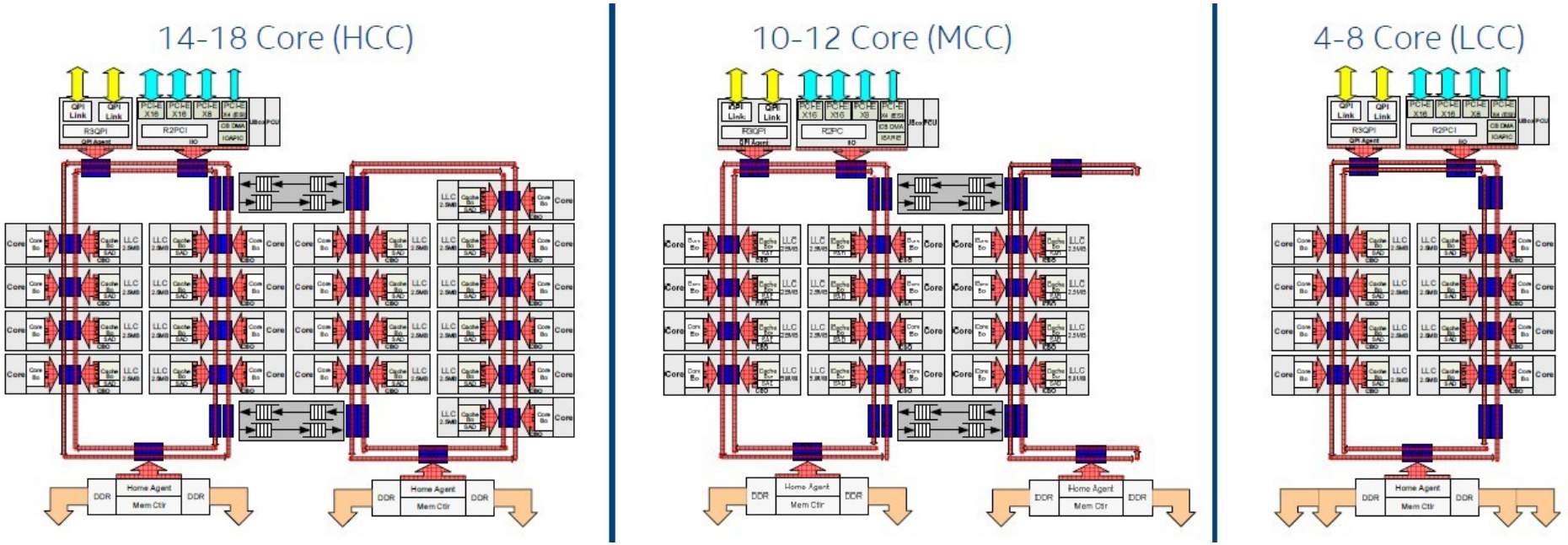
University of BRISTOL

Possible Reasons for Load Imbalance?

- Unequal division of labour.
- Independent clocking of cores (to remain inside thermal limits). For example, interaction between AVX instructions and turbo frequencies,
  - see e.g.: http://www.anandtech.com/show/8423/intel-xeon-e5-version-3-up-to-18-haswell-ep-cores-/5

Possible Reasons for Load Imbalance?

- NUMA: Current processors use bus architectures to access memory, and they are less symmetric than you might expect!



University of BRISTOL

Performance: Schedule

- **`Static`**
  - Iterations are divided evenly among threads. Extra control via chunk size. Chunks assigned on a 'round-robin' basis.
- **`Dynamic`**
  - Threads will come back for more once they have completed their block—faster threads free to acquire more work.
- **`Guided`**
  - As per dynamic but with initial large blocks (can save some overheads).

Performance: Schedule

```
#define CHUNKSIZE   10
#define N           100
int main (int argc, char *argv[]) {
  chunk = CHUNKSIZE;
#pragma omp parallel shared(a,chunk) \
                     private(i,tid,nthreads)
  {
    ...
#pragma omp for schedule(static,chunk)
    for (i=0; i<N; i++)
      {
        a[i] = i * 1.0;
        printf("Thread %d: a[%d]= %f\n",tid,i,a[i]);
      }
    printf("Thread %d done.\n",tid);
  }  /* end of parallel section */
```

The 'nowait' Clause

```
#pragma omp parallel
  {
     #pragma omp for nowait
          for (ii=1; ii<N; ii++)
               a[ii] = (a[ii] + a[ii-1]) /2.0;
     #pragma omp for
          for (ii=1; ii<N; ii++)
               c[ii] = d[ii]/e[ii];
   }
```

- Allows you to avoid the implicit barrier at the end of a worksharing for loop.
- Only possible if your loops are independent (note different data structures).

More on Load (Im)Balance

- Consider that you have 100 cores at your disposal:
  - Uncommon even a few years ago, but now close to routine..
  - Each core will do 1% of the work (given a task which we can parallelise well).
  - What if one processor was running just over 1% slower than the rest?

More on Load (Im)Balance

- Consider that you have 100 cores at your disposal:
  - Uncommon even a few years ago, but now close to routine..
  - Each core will do 1% of the work (given a task which we can parallelise well).
  - What if one processor was running just over 1% slower than the rest?

You'd be better off without it!

More on Load (Im)Balance

- This example gives us a feel for how important load balance can be, as core counts rise..
- Happily load tends to be balanced in (data parallel) work-sharing loops.
- But this is not the case for task parallel patterns.

Summary

- Use OpenMP wisely:
  - Concurrent threads gives us lots of opportunities
  - But also introduces new potential pitfalls
    - Race Condition Bugs
    - Load Imbalance
    - And methods to combat:
      - Good design
      - Schedule
      - nowait