Summary: Introduction to OpenMP
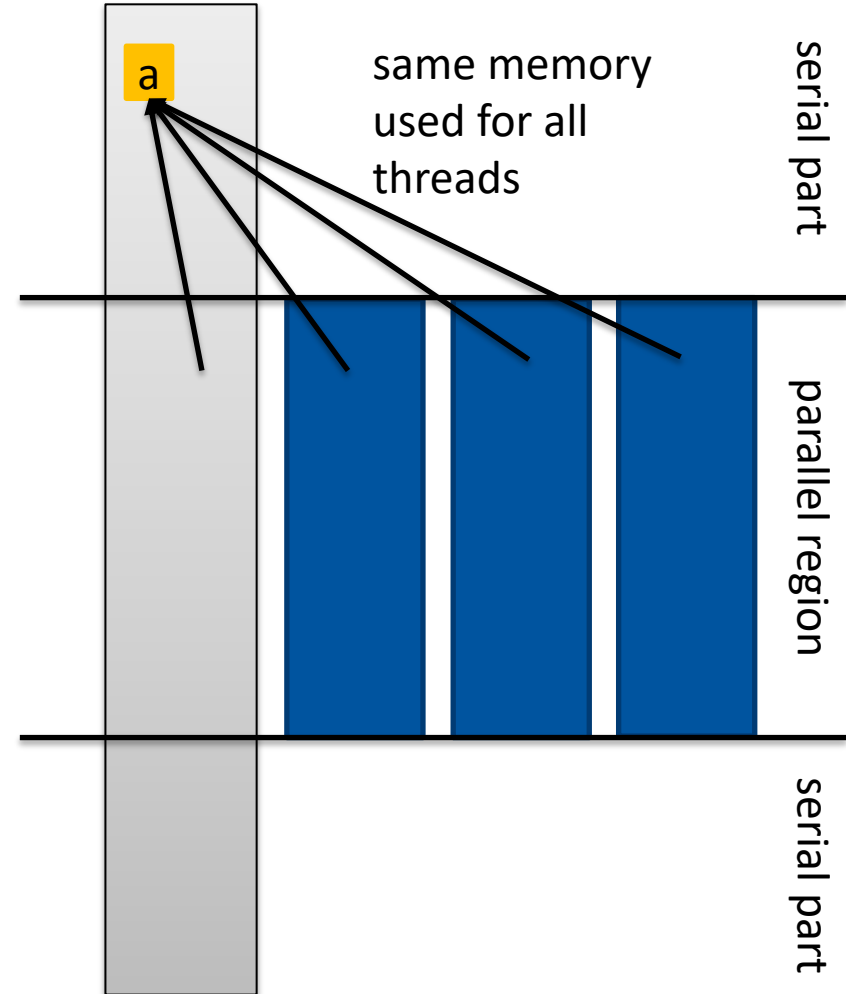
Fork-Join Execution Model

.

.

.

#pragma omp parallel

{

.

.

.

}

.

.

.
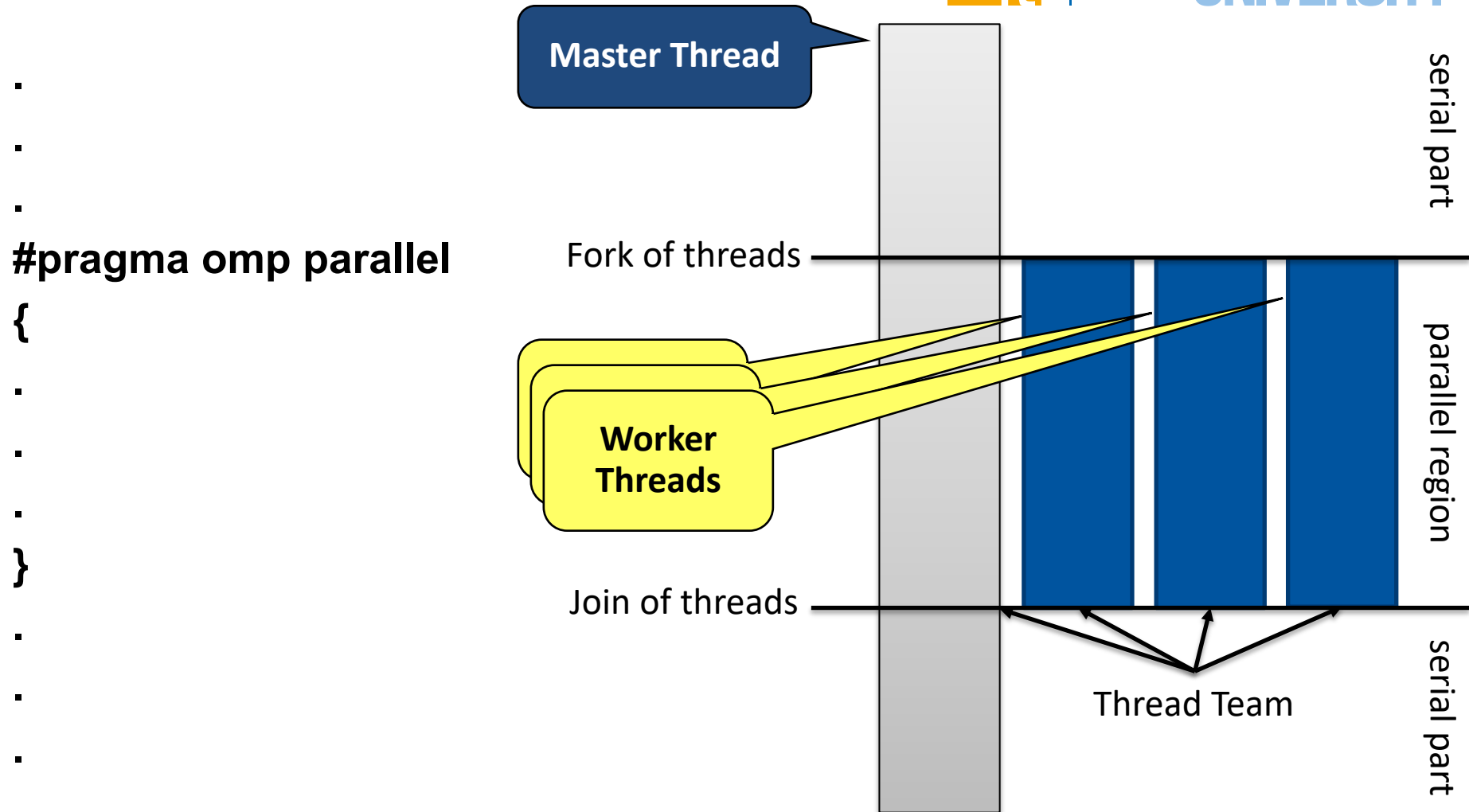
Data Sharing Attributes

int a;

.

.

#pragma omp parallel shared(a)

{

.

.

.

}

.

.

.

same memory used for all threads

serial part

parallel region

serial part

OpenMP and Performance
C. Terboven| IT Center der RWTH Aachen University

Data Sharing Attributes

int a,b;

.

#pragma omp parallel shared(a) //
  private(b)

{

.

int c;

.

}

.

.

.

uninitialized
private copies
of b and c for
every thread

OpenMP and Performance
C. Terboven | IT Center der RWTH Aachen University
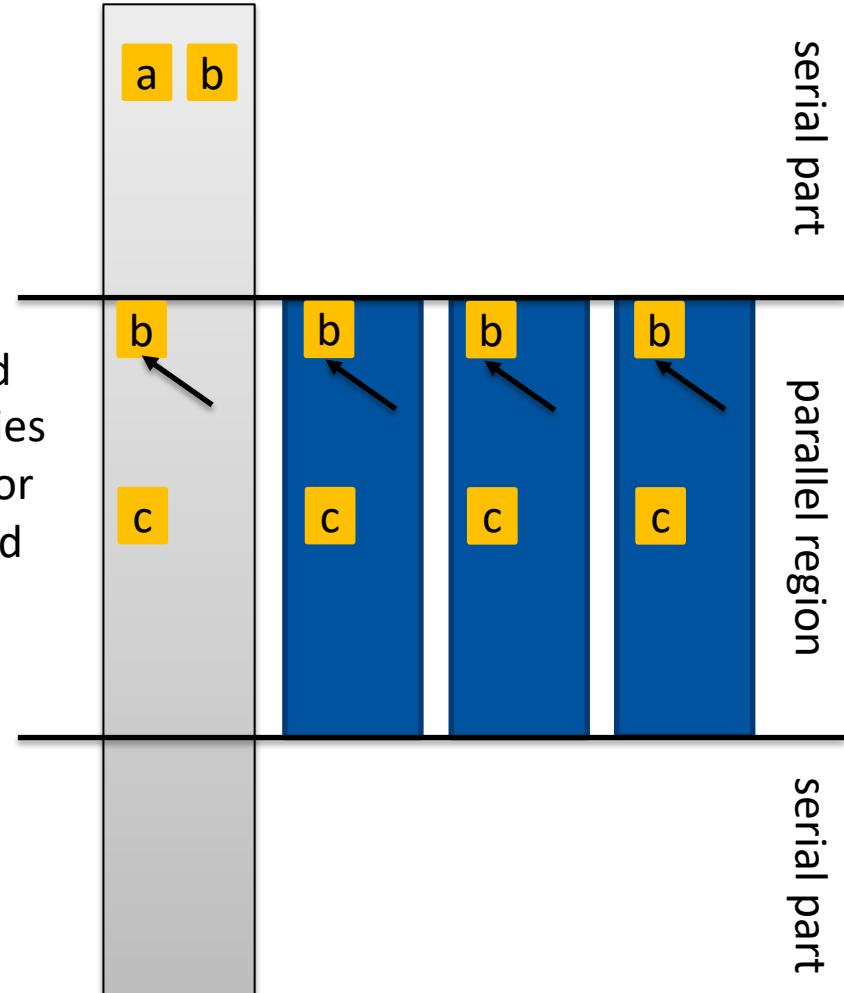
Data Sing Attributes

```
int d=2;

.

.

#pragma omp parallel firstprivate(d)

{

#pragma omp single

{d=6;}

.

}

.

.

.
```
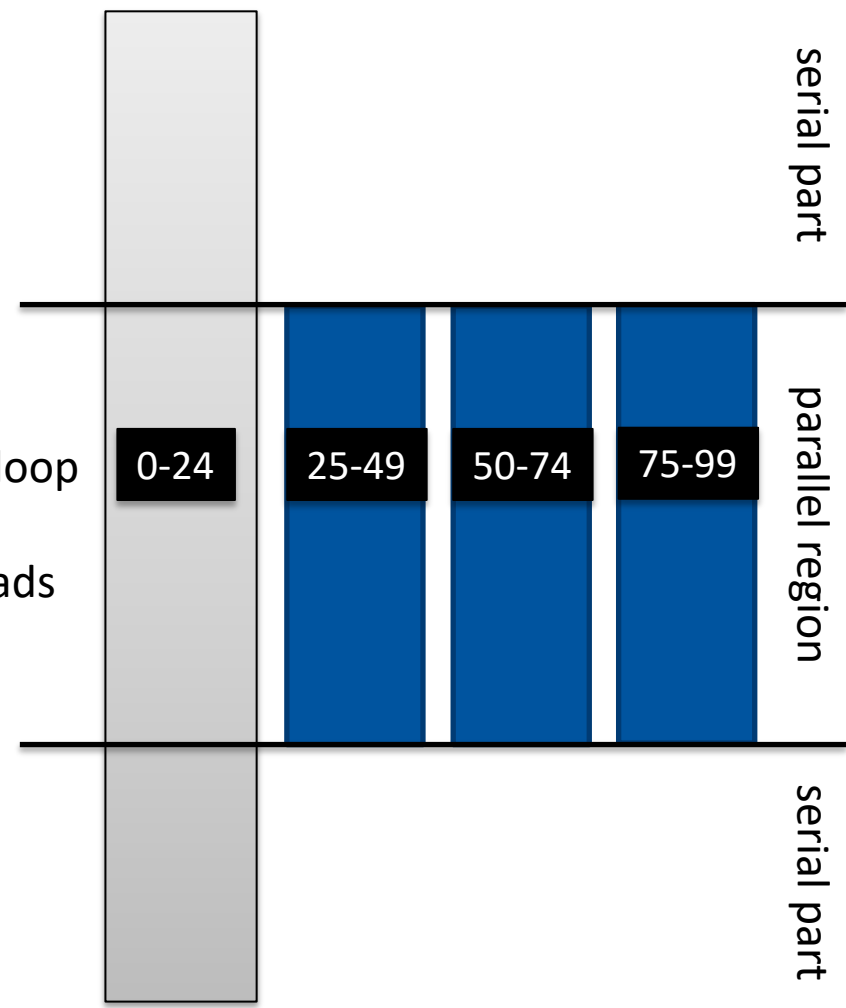

copy of the value of d

d=2

d=2   d=2   d=2   d=2

d=6

serial part

parallel region

serial part
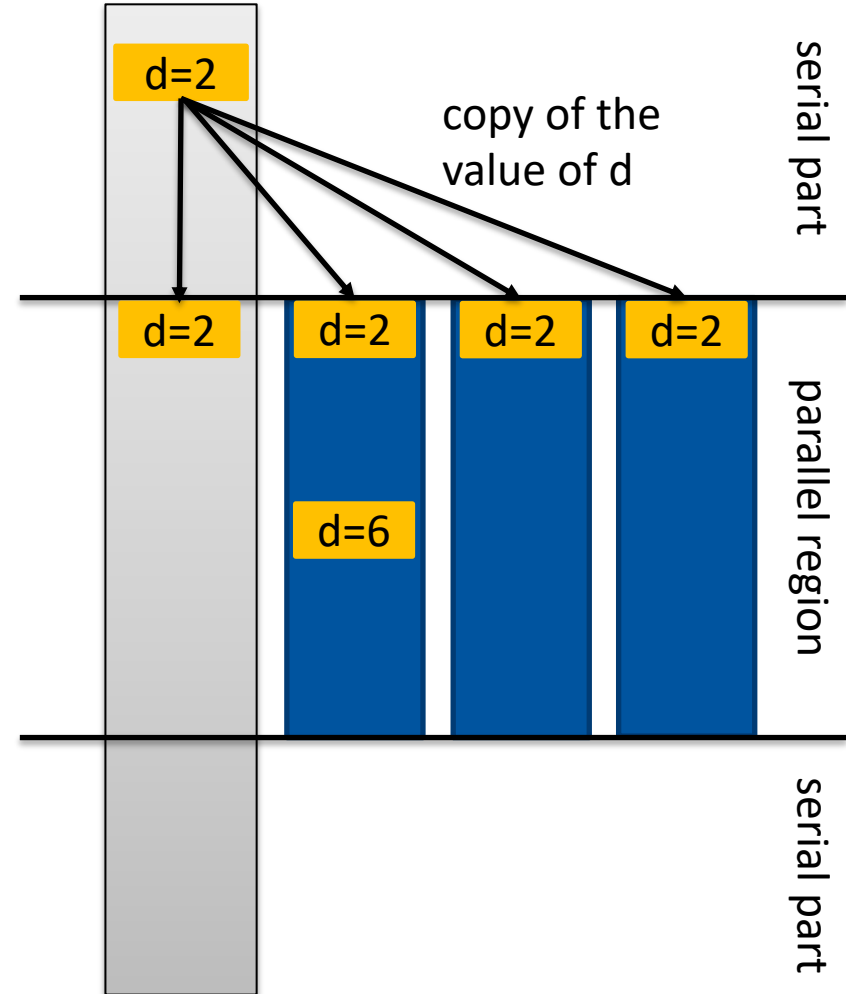
For Worksharing

-
-
-

#pragma omp parallel

#pragma omp for

for (int i=0; i<100; i++){

-
-
-

}

-
-
-

distributes loop
iterations
across threads

serial part

0-24	25-49	50-74	75-99

parallel region

serial part

OpenMP and Performance
C. Terboven| IT Center der RWTH Aachen University

Parallelizable Loops

- **Loop iterations must be independent to parallelize a loop!**

No loop dependencies => parallelizable

```
#pragma omp parallel for
for ( i=0 ; i<100 ; i++ ){
        a[i] = b[i] + c[i];
}
```

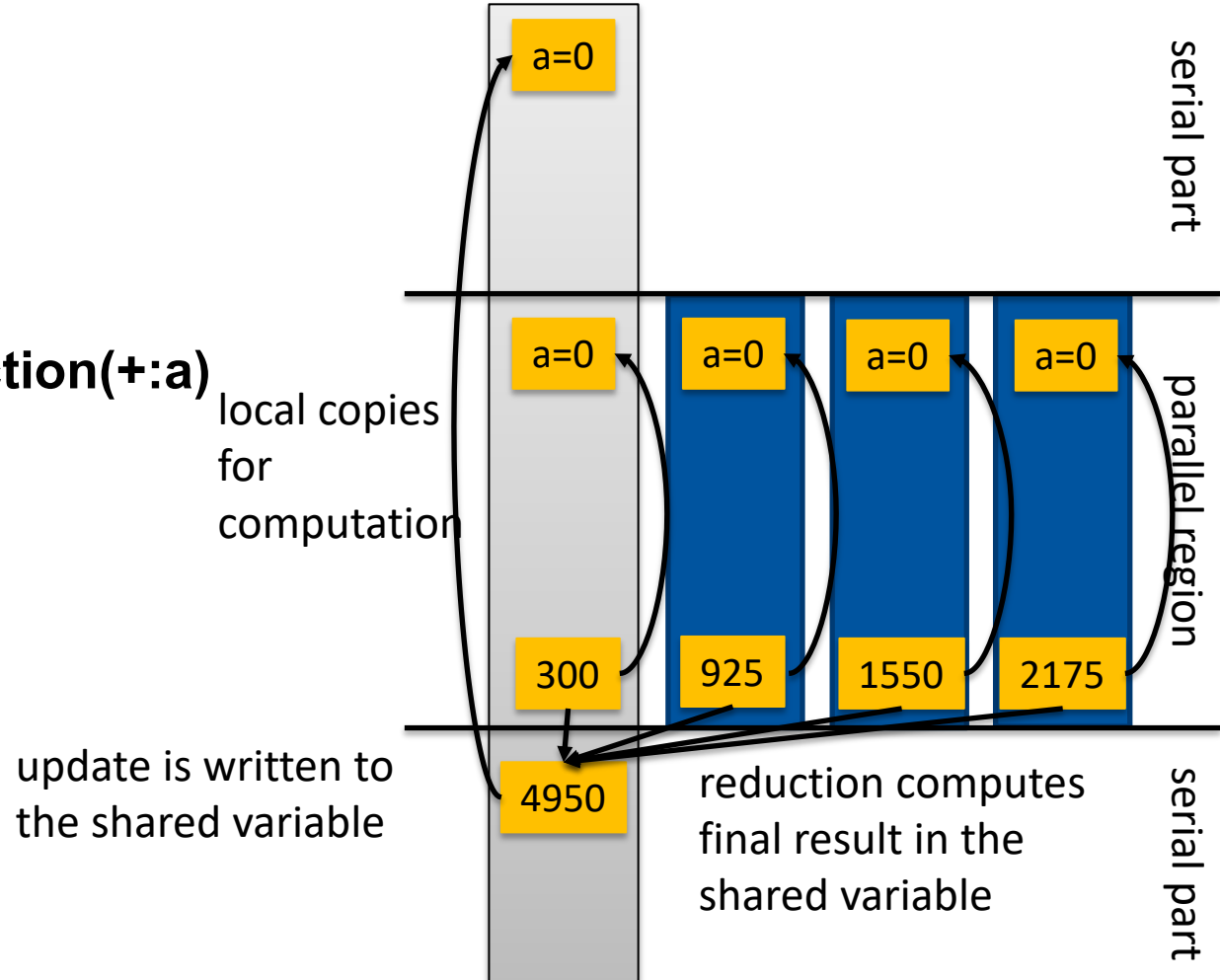Loop dependencies => **not** parallelizable

```
#pragma omp parallel for
for ( i=1 ; i<100 ; i++ ){
        a[i] = a[i] + a[i-1];
}
```

- **Simple test: If the results differ when the code is executed backwards, the loop iterations are not independent.**

  BUT: This test alone is not sufficient

OpenMP and Performance
C. Terboven| IT Center der RWTH Aachen University

Reduction Operations

```
int a=0;

.
.

#pragma omp parallel
#pragma omp for reduction(+:a)
for (int i=0; i<100; i++)
{
        a+=i;
}

.
.
.
```


serial part

a=0

local copies for computation

a=0   a=0   a=0   a=0

parallel region

300   925   1550   2175

update is written to the shared variable

4950

reduction computes final result in the shared variable

serial part

OpenMP and Performance
C. Terboven | IT Center der RWTH Aachen University

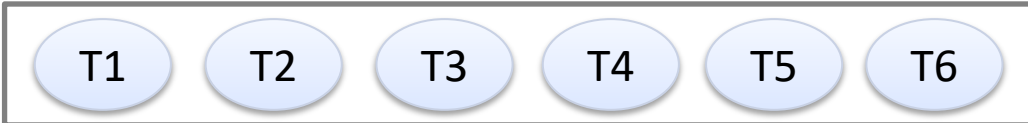# Tasks

```
#pragma omp parallel
#pragma omp single
while (work()){
        #pragma omp task
        {
        …
        }
}// implicit barrier here
```

.

.

Taskqueue

(T1) (T2) (T3) (T4) (T5) (T6)

serial part

parallel region

serial part
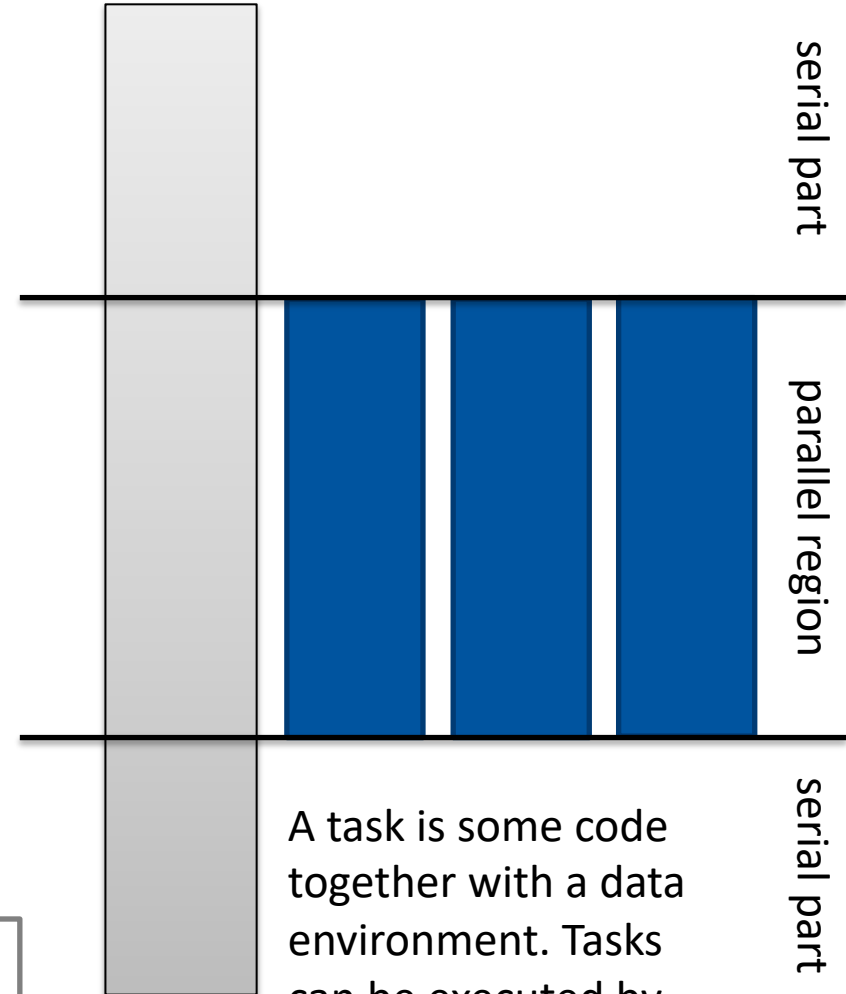
A task is some code together with a data environment. Tasks can be executed by any thread in any order.

OpenMP and Performance
C. Terboven| IT Center der RWTH Aachen University

The Barrier and Taskwait Constructs

- **OpenMP `barrier` (implicit or explicit)**

  → All tasks created by any thread of the current *Team* are guaranteed to be

  completed at barrier exit

C/C++
`#pragma omp barrier`

- **Task barrier: `taskwait`**

  → Encountering Task suspends until child tasks are complete

  → Only direct childs, not descendants!

C/C++
`#pragma omp taskwait`

Questions?

OpenMP and Performance
C. Terboven| IT Center der RWTH Aachen University