Barrier/1

Suppose we run each of these two loops in parallel over i:

```
for (i=0; i < N; i++)
   a[i] = b[i] + c[i];
```

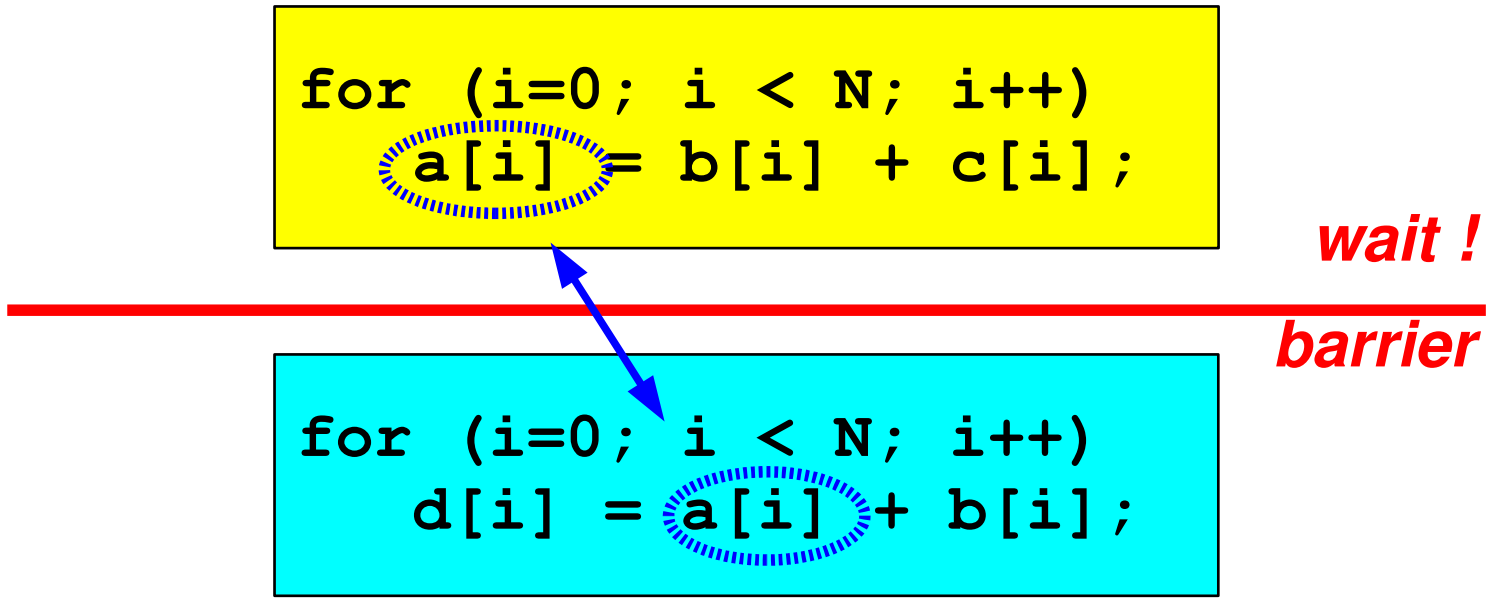
```
for (i=0; i < N; i++)
   d[i] = a[i] + b[i];
```

This may give us a wrong answer (one day)

Why ?

Barrier/2

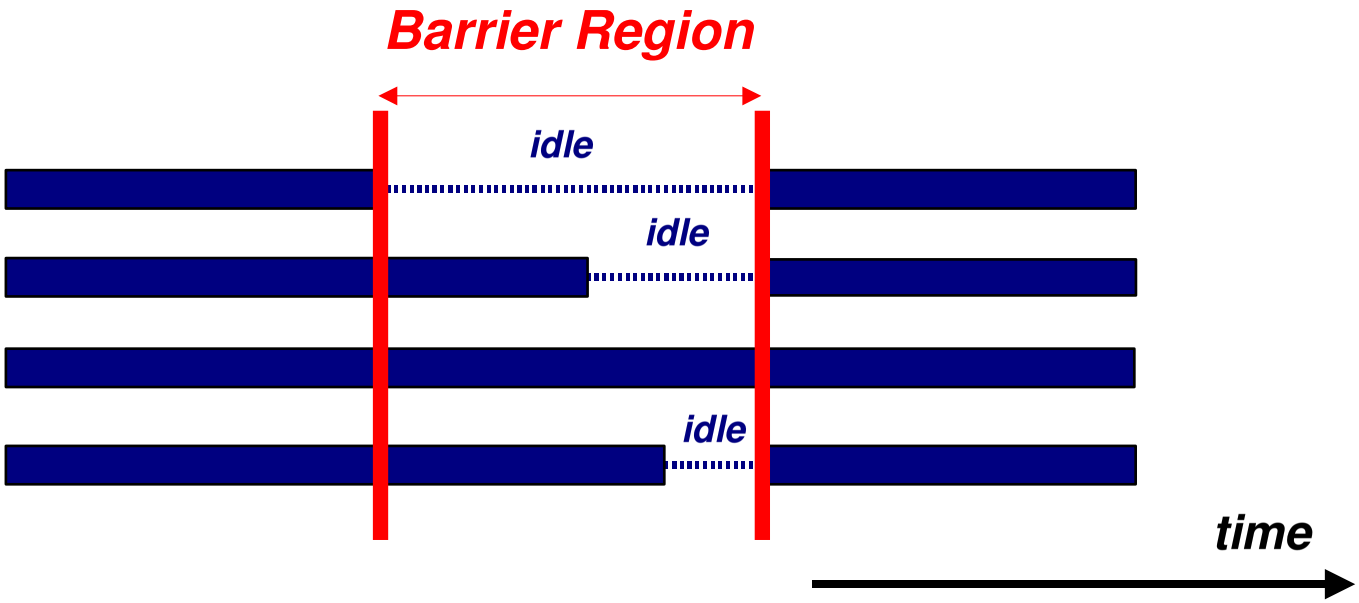
*We need to have updated all of a[ ] f rst, before using a[ ] **



All threads wait at the barrier point and only continue when all threads have reached the barrier point

**) If there is the guarantee that the mapping of iterations onto threads is identical for both loops, there will not be a data race in this case*

Barrier/3



Barrier syntax in OpenMP:

```
#pragma omp barrier
```

```
!$omp barrier
```

When to use barriers ?

- *If data is updated asynchronously and data integrity is at risk*
- *Examples:*
  - *Between parts in the code that read and write the same section of memory*
  - *After one timestep/iteration in a solver*
- *Unfortunately, barriers tend to be expensive and also may not scale to a large number of processors*
- *Therefore, use them with care*