OpenMP

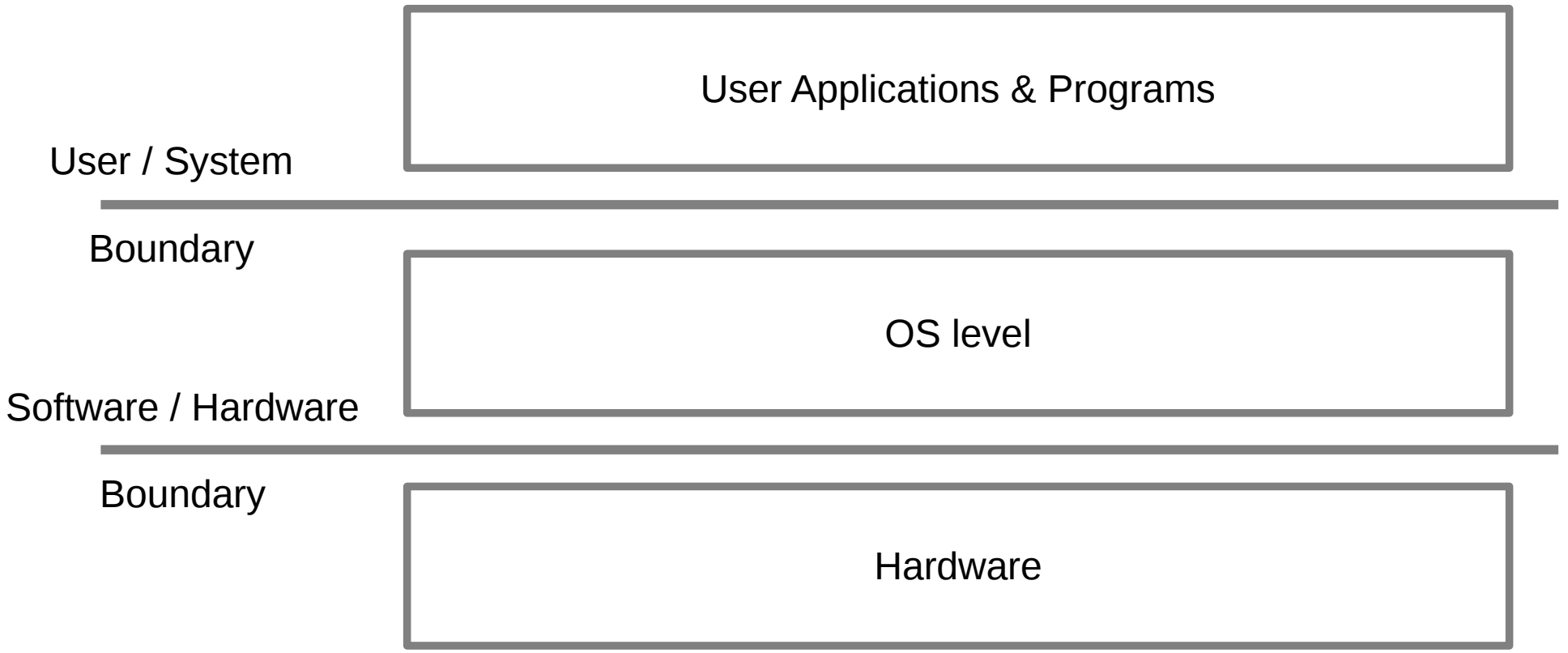
ECSE 420 - Tutorial 3

Dimitrios Stamoulis

TR 4110

October 6, 2014

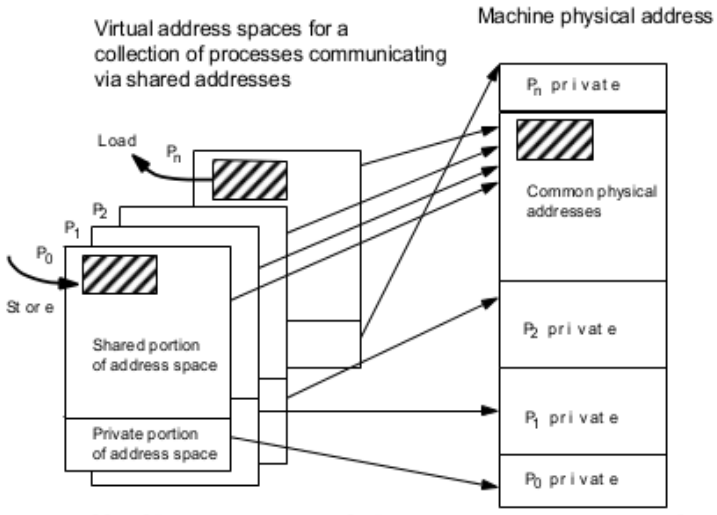
Tutorial 3 – Introduction



Tutorial 3 – Introduction

Structured Shared Address Space

From ECSE 420 lecture slides!!



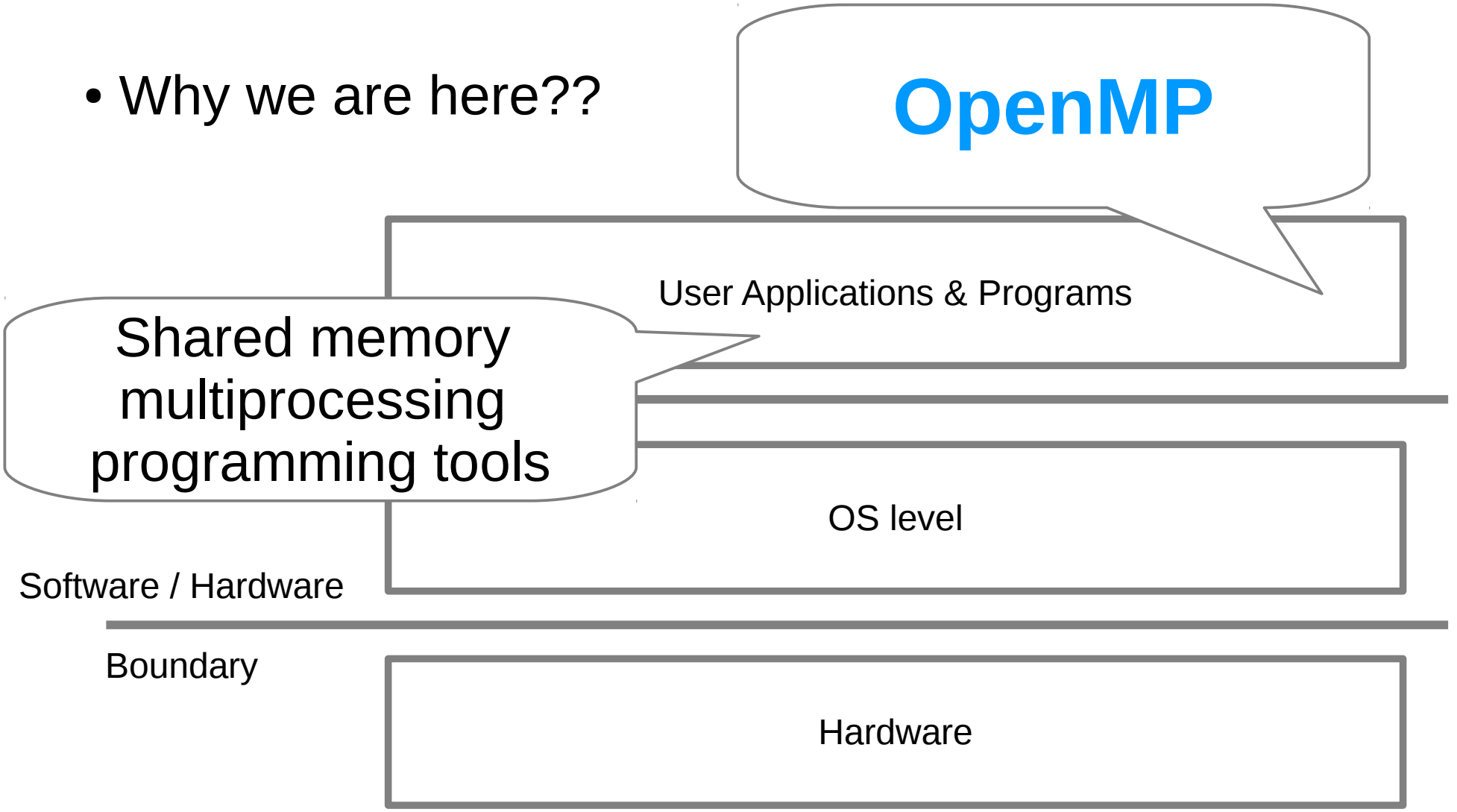
Softwar

Boundary

Hardware

Tutorial 3 – Introduction

- Why we are here??



Tutorial 3 - Material

- Linux machine :
  - Compile an OpenMP program :

```
$ gcc -fopenmp myprogram.c
$ ./a.out
```

Shared Variable - A simple example

```
int main()
{
   int sharedVar = 5;
   CREATE THREAD( thread1 )
   CREATE THREAD( thread2 )
   WAIT THREADS(..)
   printf ("shared var = %d\n", sharedVar);
   return 0;
}
```

Thread 1:

```
for (i=0; i<LOOPS; i++)
    *shared_var *= 2;
```

Thread 2:

```
for (i=0; i<LOOPS; i++)
    *shared_var -= 2;
```

```
stam@Stam:~/codes$ for i in $(seq 5); do ./sharedVar  ; done
shared variable=-36700160
shared variable=5242840
shared variable=5242840
shared variable=5242840
shared variable=-36700160
```

OpenMP (Open Multi-Processing)

- Application programming interface (API)

- Shared memory multiprocessing programming API:
  - Run-time library
  - Compiler directives (pragmas)
  - Environment variables

API → Sets the behaviors of the **runtime system**
→The runtime system implements the **execution model**

OpenMP (Open Multi-Processing)

- Application programming interface (API)

- Shared memory multiprocessing programming API:
  - Run-time library
  - Compiler directives (pragmas)
  - Environment variables

API → Sets the behaviors of the **runtime system**

→ The runtime system implements the **OpenMP execution model**

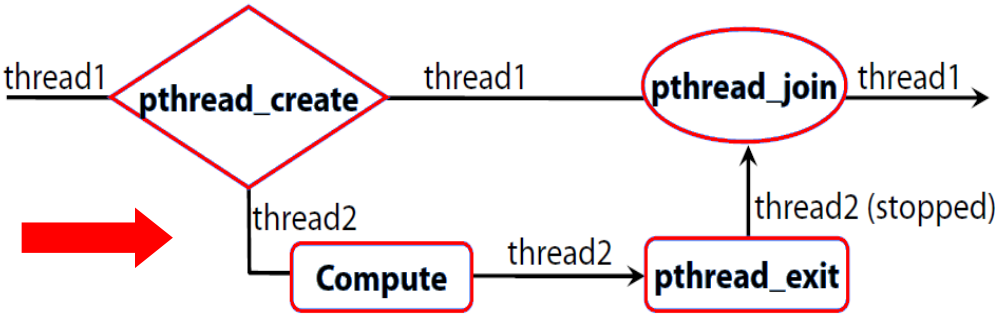
OpenMP (Open Multi-Processing)

- Application programming interface (API)

- Shared memory multiprocessing programming API:
  - Run-time library
  - Compiler directives (pragmas)
  - Environment variables

API → Sets the behaviors of the **runtime system**

→ The runtime system implements the **OpenMP execution model**

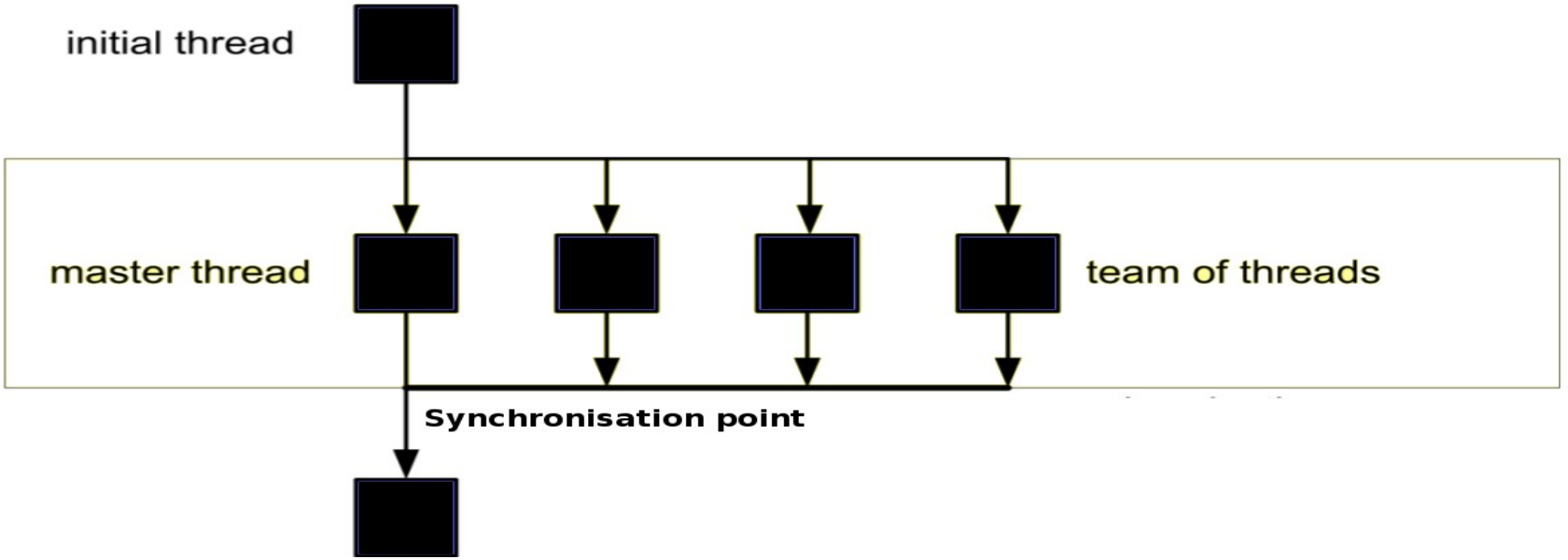


Lab 1 Revision

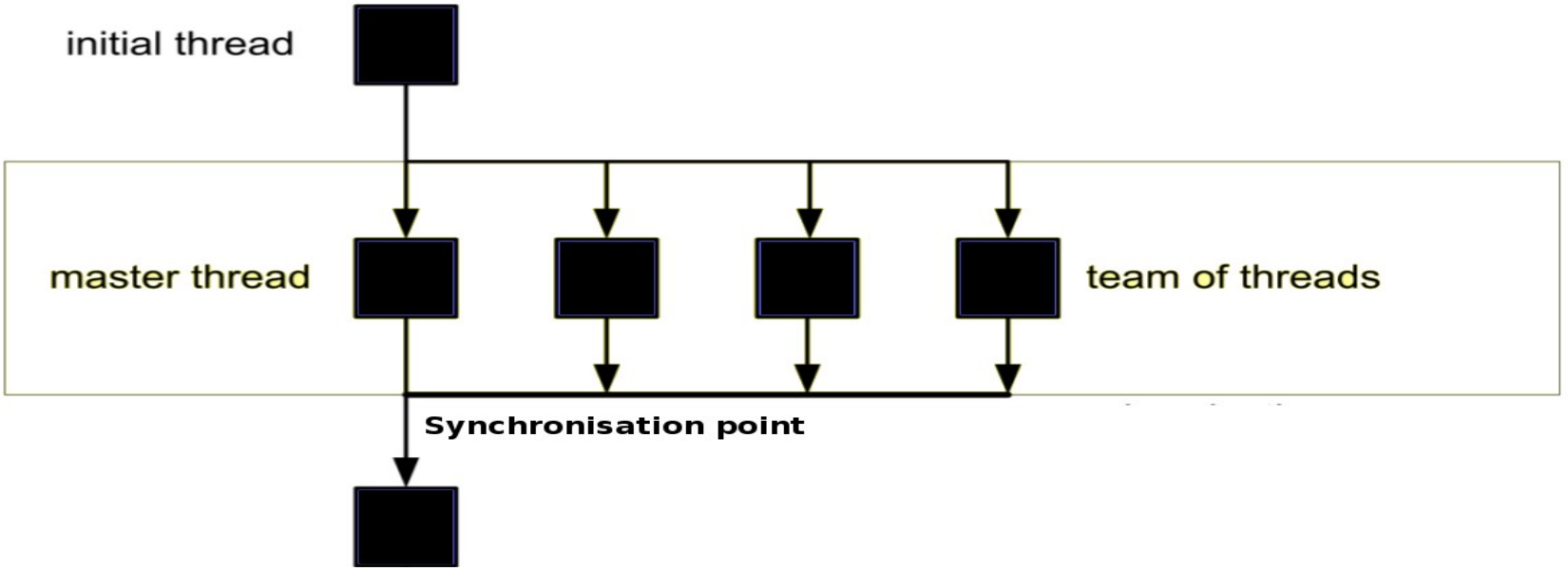
fork-join model

Execution model (1/2)

- **OpenMP uses a fork-join model of parallel execution :**
  - A thread encounters an 'omp parallel' section
  - The thread creates a team composed of itself and other additional threads
  - The encountering thread becomes the master of the new team
  - All team members execute the code inside the 'omp parallel' section



Execution model (1/2)



- **OpenMP uses a fork-join model of parallel execution.**
  - All threads finish their work and wait at the implicit barrier at the end of the 'omp parallel' section.
  - When **ALL** team members have arrived at the barrier, the threads can leave the barrier.
  - The master thread continues execution of user code beyond the end of the parallel section.

Basic Concepts

- OpenMP API programming tools:
  - Run-time library
  - Compiler directives
  - Environment variables

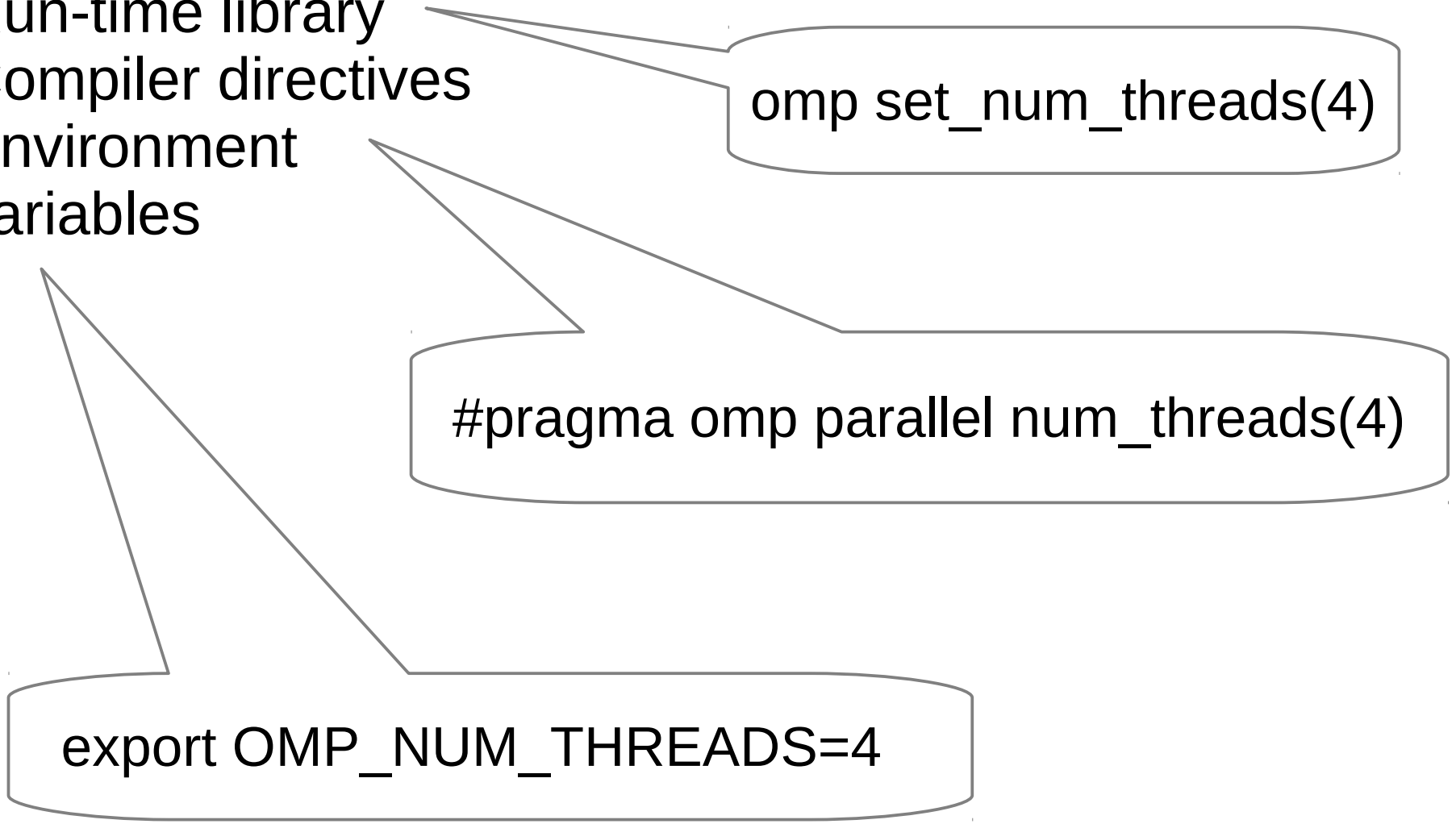
- Number of threads
- Thread ID
- Timers
- ...

- Parallel regions
- Work sharing
- Synchronization
- Specific attributes
  - private
  - shared
  - Reduction
- ...

- Number of threads
- Scheduling type
- Nested parallelism
- ...

Basic Concepts

- I can do the same thing with more that one way :
  - Run-time library
  - Compiler directives
  - Environment variables



Basic Concepts – Compiler directives

#pragma omp <directive> <clauses>

Different <directive> types to:

- Define Parallel region
  - #pragma omp parallel
- Orchestrate Work sharing
  - #pragma omp for
  - #pragma omp single
- Achieve Synchronization
  - #pragma omp barrier
  - #pragma omp master
  - #pragma omp critical
  - #pragma omp ordered

Basic Concepts – Compiler directives

#pragma omp <directive> <clauses>

Different <directive> types to:

- Define Parallel region
  - #pragma omp parallel
- Orchestrate Work sharing
  - #pragma omp for
  - #pragma omp single
- Achieve Synchronization
  - #pragma omp barrier
  - #pragma omp master
  - #pragma omp critical
  - #pragma omp ordered

Parallel region

#pragma omp parallel <clauses>
{ structured-block }

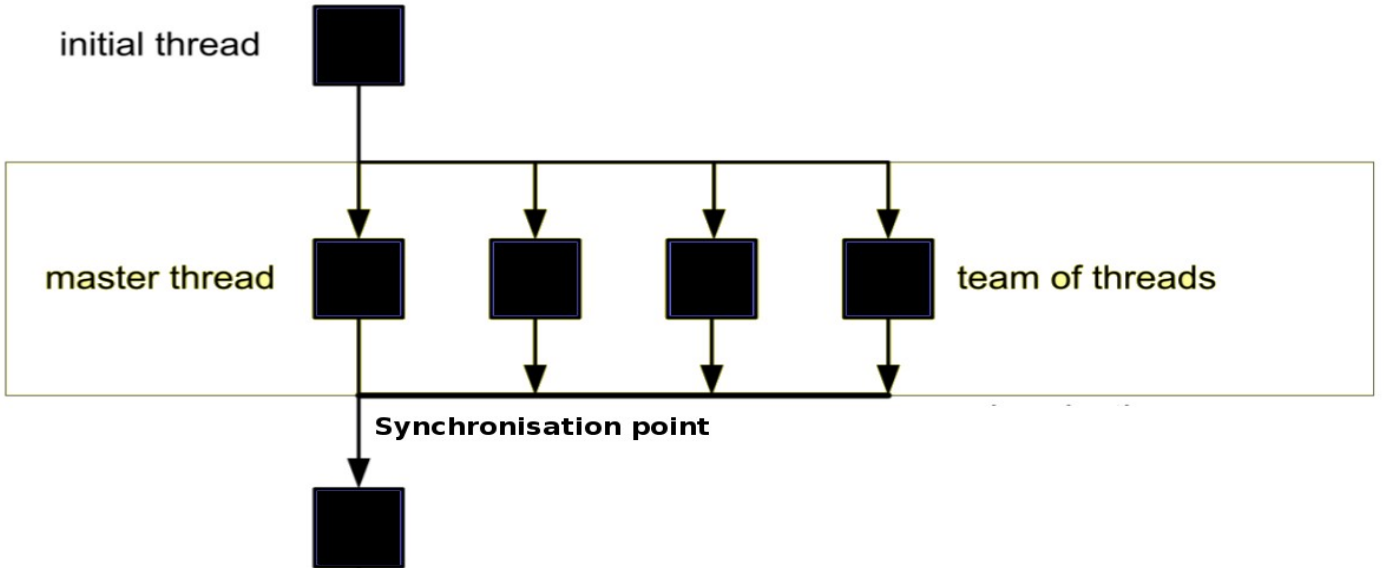
We can define different <clauses> :

- num_threads
- private variables
- shared variables
- ...

We have an implicit barrier at the end of the 'omp parallel' section → The barrier implies 'flush'

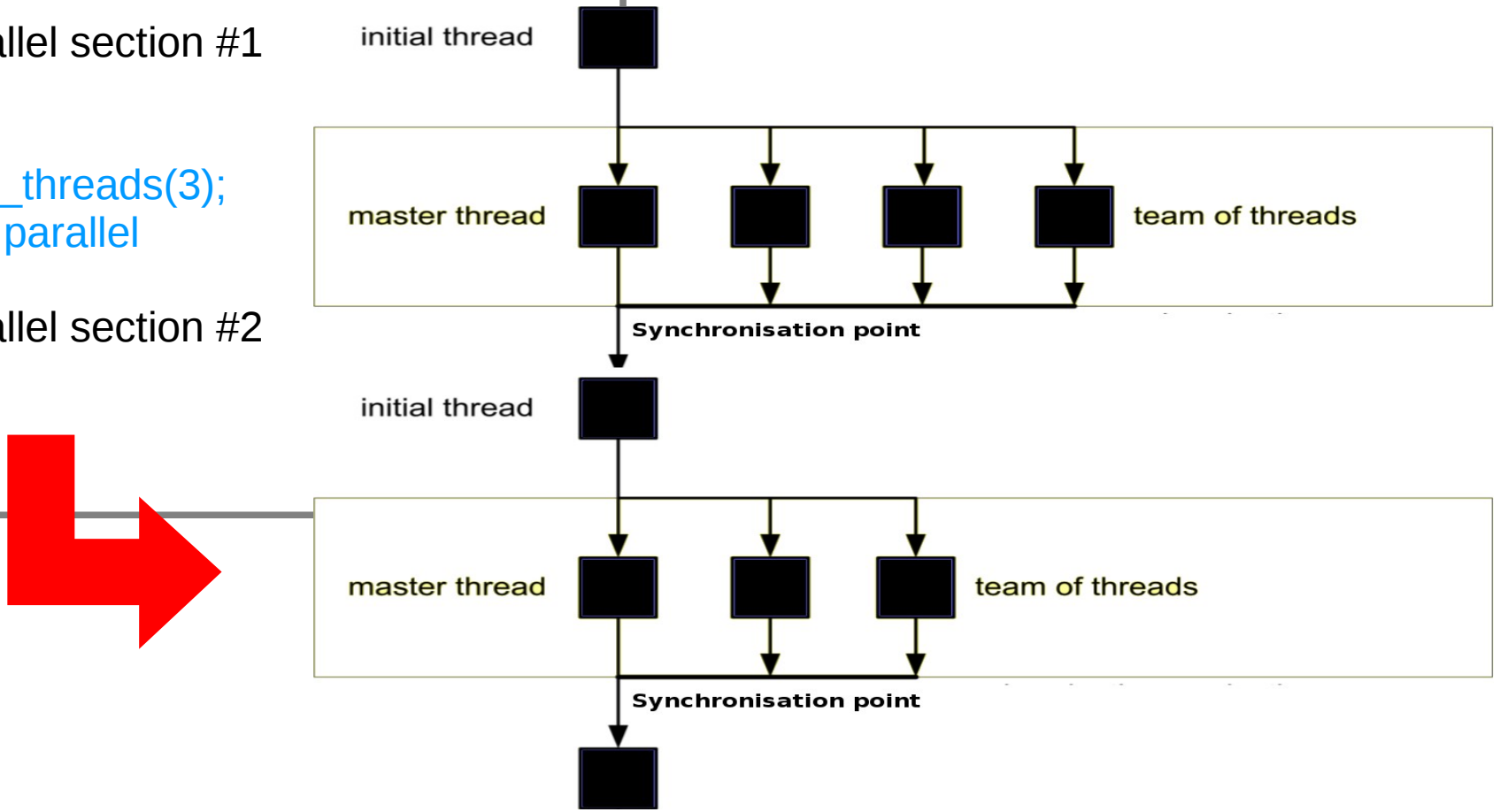
Parallel region - Our 1st example ...

```
#include <omp.h>

int main(){
  ...
  #pragma omp parallel num_threads(4)
  {
     ... // our parallel section
  }
  ...
}
```



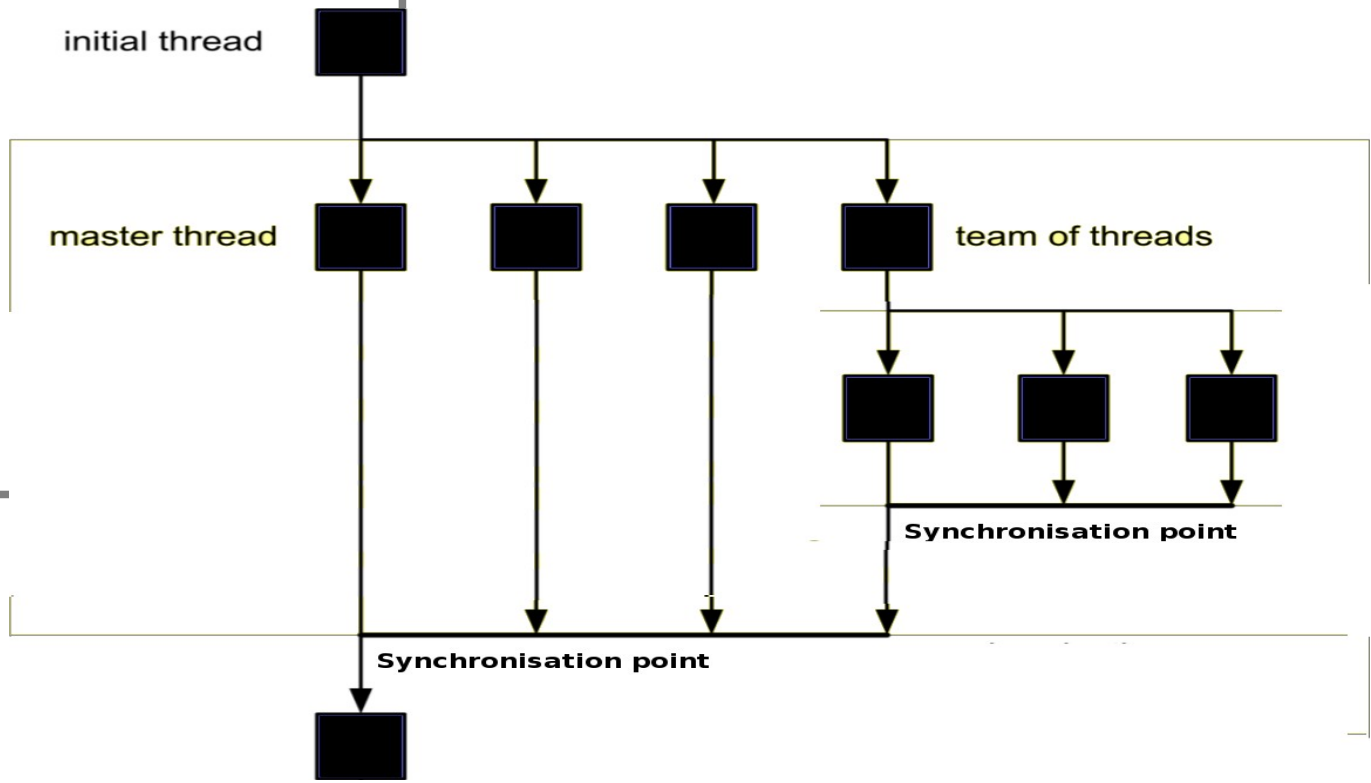
Parallel region - Our 2nd example ...

```
#include <omp.h>

int main(){
  ...
  #pragma omp parallel num_threads(4)
  {
     ... // our parallel section #1
  }
  ...
  omp_set_num_threads(3);
  #pragma omp parallel
  {
     ... // our parallel section #2
  }
  ...
}
```



Nested parallel regions (1/2)

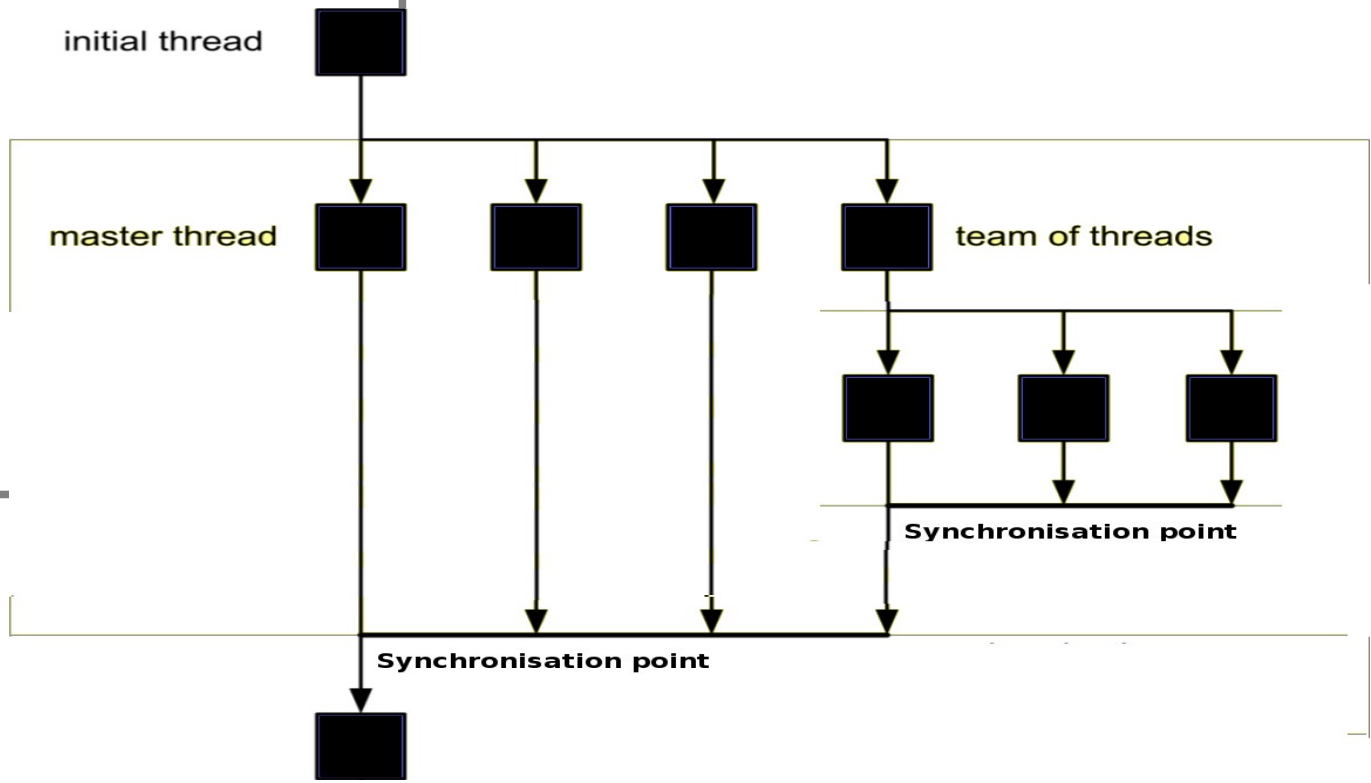
```
#include <omp.h>

int main(){
  ...
  omp_set_nested(1);
  #pragma omp parallel num_threads(4)
  {
     ... // our parallel section #1

    if (omp_get_thread_num()== 3){
     ...
     omp_set_num_threads(3);
     #pragma omp parallel
     {
        ...
     }
   }
  ...
}
```



Nested parallel regions (1/2)

```
#include <omp.h>

int main(){
  ...
  omp_set_nested(1);
  #pragma omp parallel num_threads(4)
  {
     ... // our parallel section #1

    if (omp_get_thread_num()== 3){
     ...
     omp_set_num_threads(3);
     #pragma omp parallel
     {
        ...
     }
    }
  ...
}
```

What if omp_set_nested(0); ????

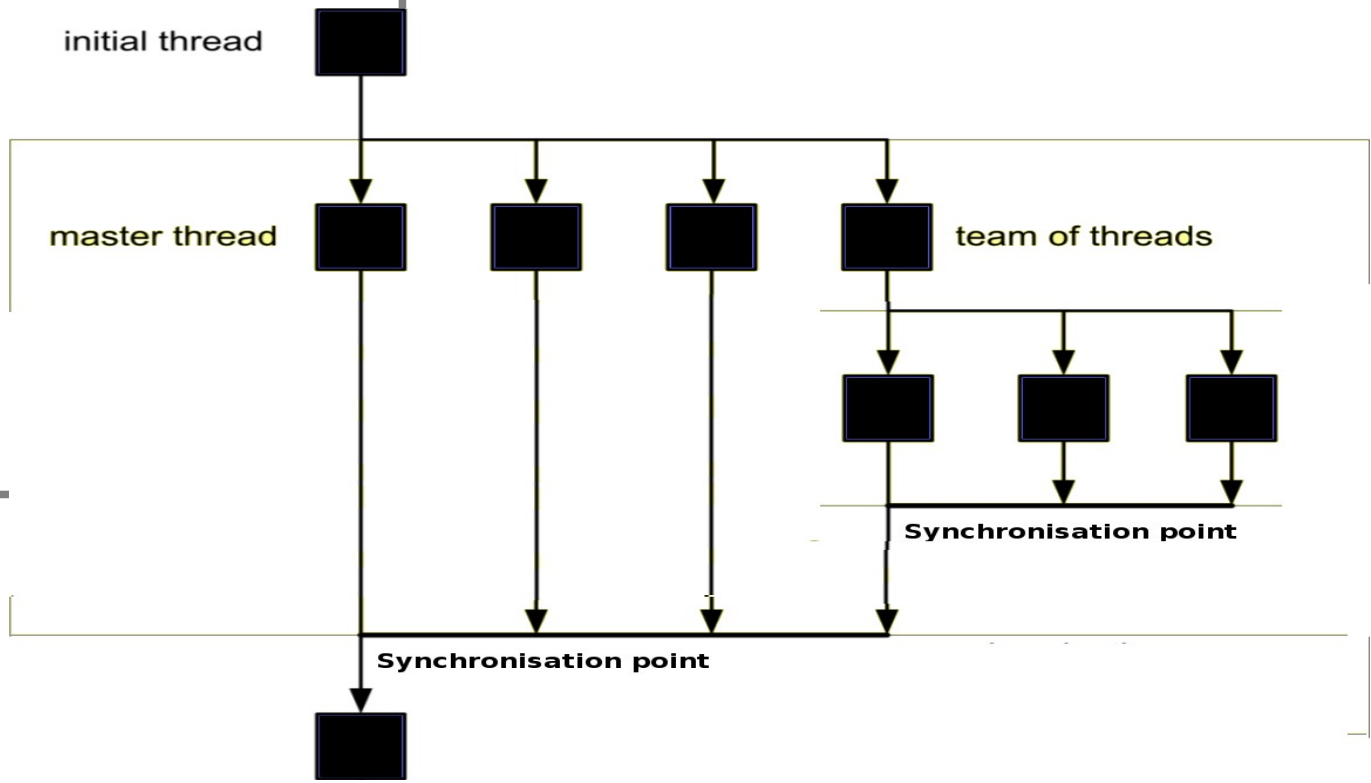


Nested parallel regions (2/2)

```
#include <omp.h>

int main(){
  ...
  omp_set_nested(1);
  #pragma omp parallel num_threads(4)
  {
     ... // our parallel section #1

    if (omp_get_thread_num()== 3){
     ...
     omp_set_num_threads(3);
     #pragma omp parallel
     {
        ...
     }
   }
  ...
}
```

What if omp_set_nested(0); ????

setenv OMP_NESTED FALSE

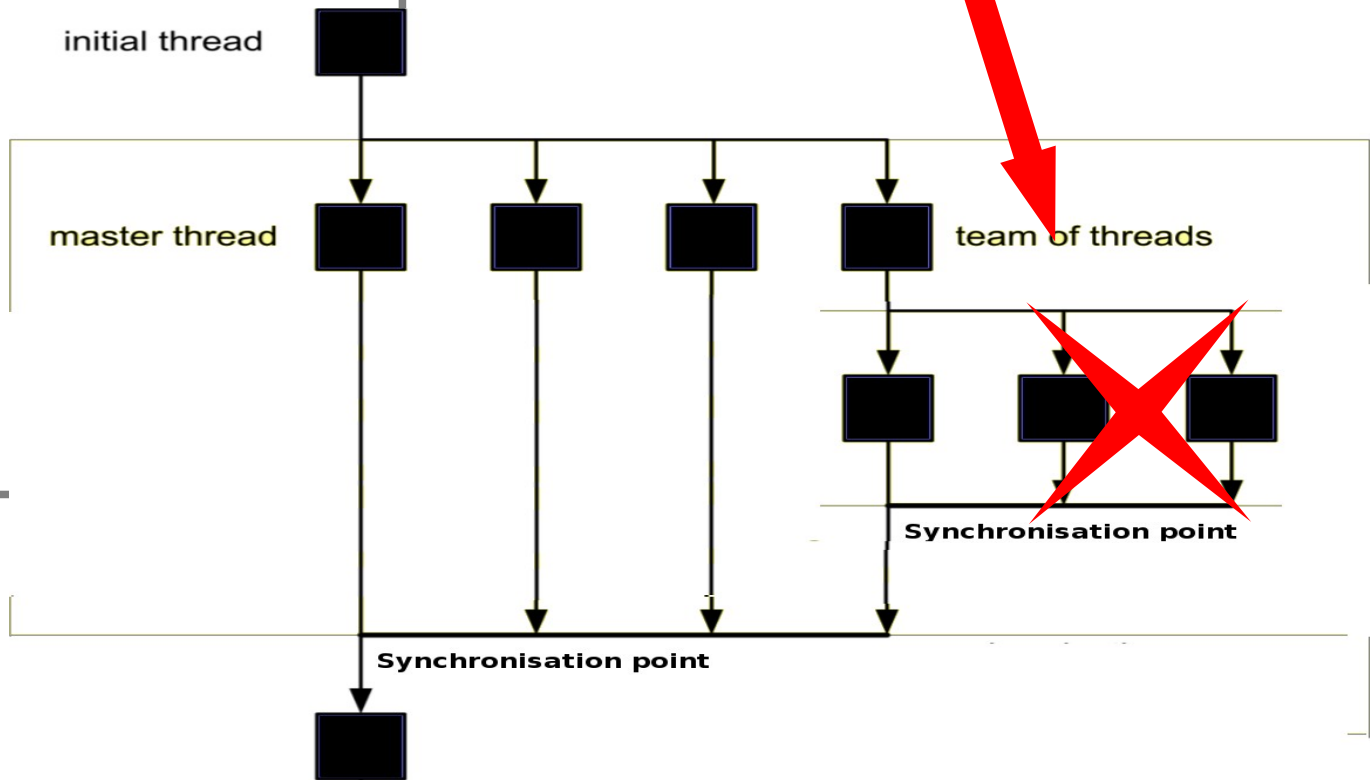


Nested parallel regions (2/2)

```
#include <omp.h>

int main(){
  ...
  omp_set_nested(1);
  #pragma omp parallel num_threads(4)
  {
     ... // our parallel section #1

    if (omp_get_thread_num()== 3){
     ...
     omp_set_num_threads(3);
     #pragma omp parallel
     {
        ...
     }
    }
  ...
}
```

What if omp_set_nested(0); ????

setenv OMP_NESTED FALSE



Basic Concepts – Compiler directives

#pragma omp <directive> <clauses>

Different <directive> types to:

- Define Parallel region
  - #pragma omp parallel
- Orchestrate Work sharing
  - #pragma omp for
  - #pragma omp single
- Achieve Synchronization
  - #pragma omp barrier
  - #pragma omp master
  - #pragma omp critical
  - #pragma omp ordered

Work sharing

#pragma omp for [schedule(...)] [nowait]
{ for - loop }

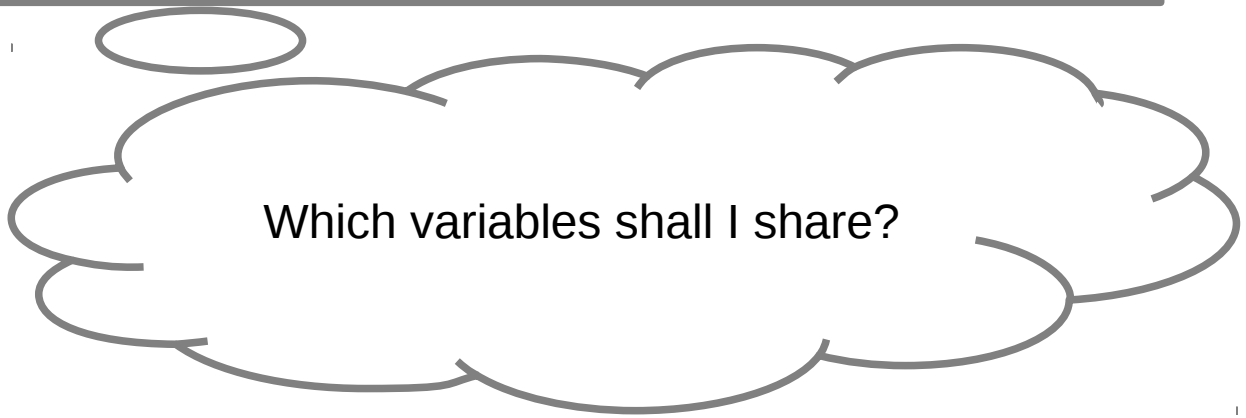
We can define different scheduling schemes :

- static : round-robin scheduling
- dynamic : Inactive threads are scheduled
- ...

We can prevent the synchronization at the implicit barrier at the end of the for-loop → nowait

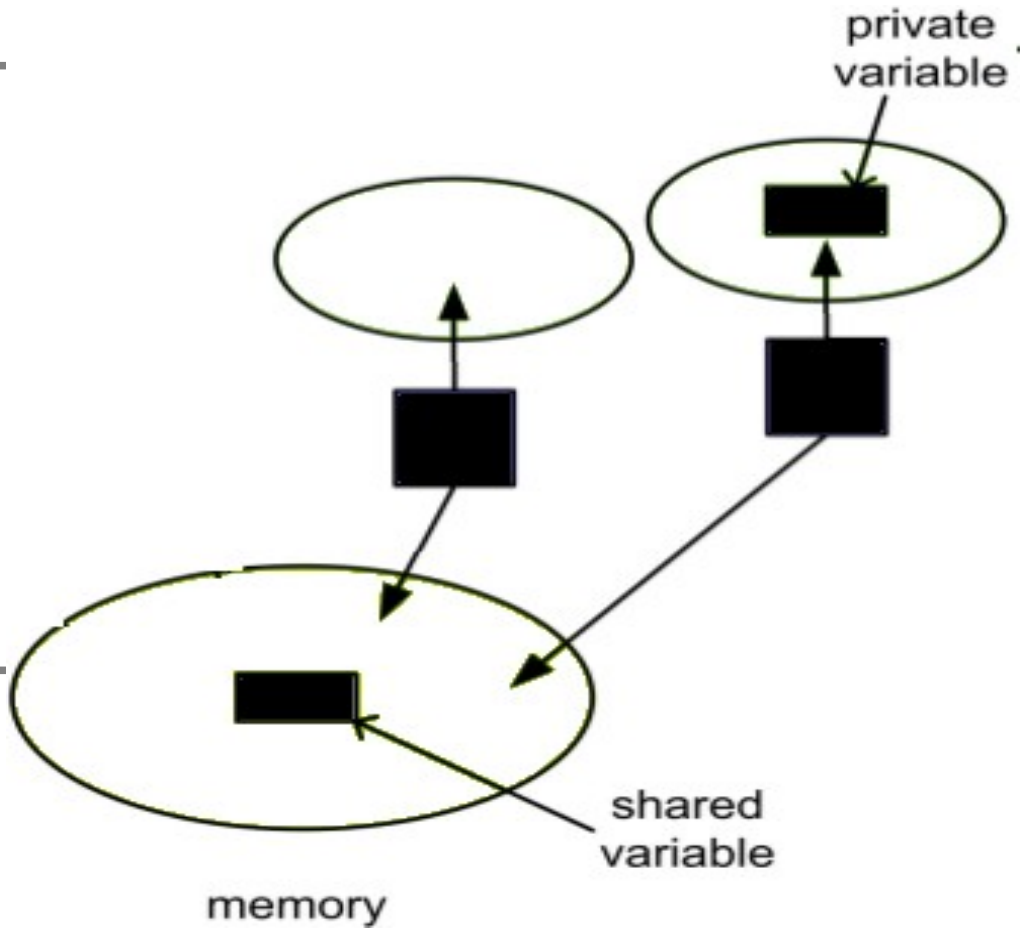
Parallel for

```
// Array addition example
int main(){
  ...

  // independent loor iterations
  for (i = 0; i < n; i++)
       c[i] = a[i] + b[i];
   ...
}
```



Shall I share everything?? (1/2)

```
  // Array addition example
int main(){
  ...

  // independent loor iterations
  for (i = 0; i < n; i++)
       c[i] = a[i] + b[i];
   ...
}
```



- Within a parallel section, we can have :
  - Shared variables
  - Private variables !!

Shall I share everything?? (2/2)

```
// independent loor iterations
for (i = 0; i < n; i++)
    c[i] = a[i] + b[i];
```

```
#include <omp.h>

int main(){
  ...
   // independent loor iterations
  #pragma omp parallel shared(n, a, b, c) private(i) num_threads(5)
  #pragma omp for
  for (i = 0; i < n; i++)
      c[i] = a[i] + b[i];
  ...
}
```

Parallel for - nowait

```
#include <omp.h>

int main(){
  ...
    #pragma omp parallel num_threads(2)
  {
    #pragma omp for nowait
    for (i = 0; i < 5; i++)
    {
      sleep(omp_get_thread_num()*2);
    }
     // verify that thread 0 did not wait
     printf("I am thread: %d\n", omp_get_thread_num() );
  }
 ...
}
```

Basic Concepts – Compiler directives

#pragma omp <directive> <clauses>

Different <directive> types to:

- Define Parallel region
  - #pragma omp parallel
- Orchestrate Work sharing
  - #pragma omp for
  - #pragma omp single
- Achieve Synchronization
  - #pragma omp barrier
  - #pragma omp master
  - #pragma omp critical
  - #pragma omp ordered

Synchronization

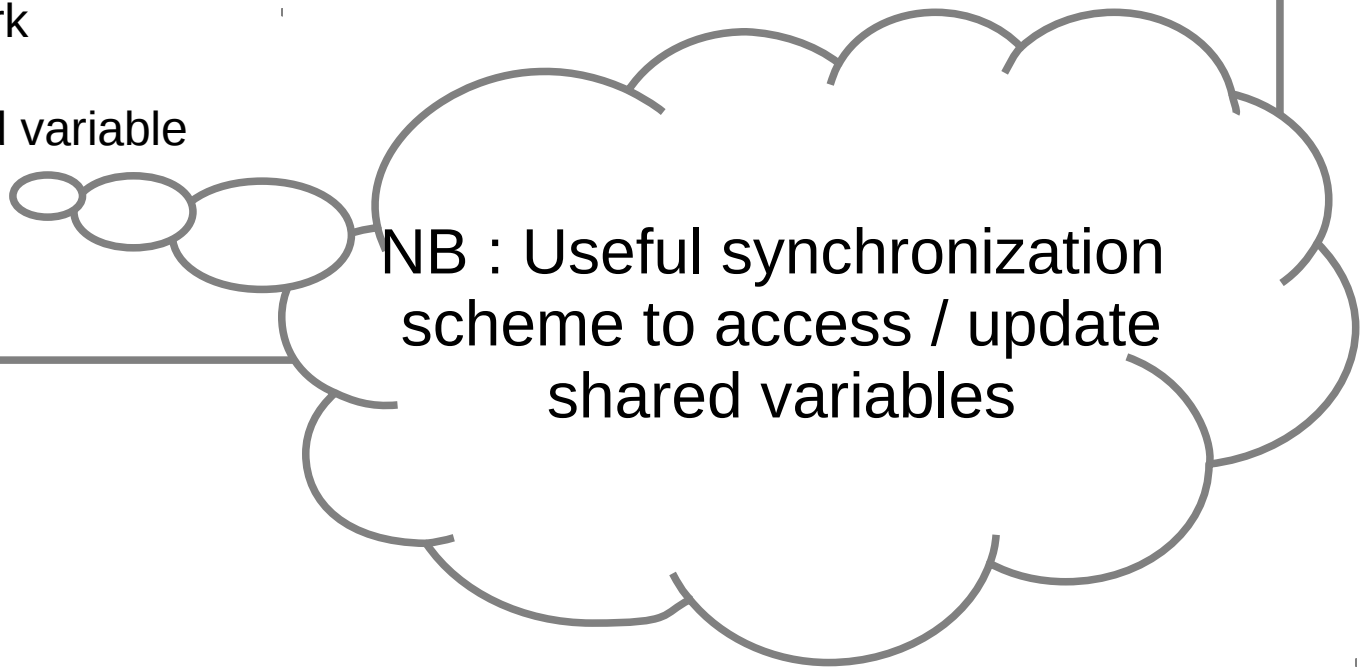
#pragma omp <directive>
{ structured-block }

We can define different <directives> :

- #pragma omp critical
- #pragma omp barrier
- #pragma omp master
- #pragma omp flush
- #pragma omp ordered

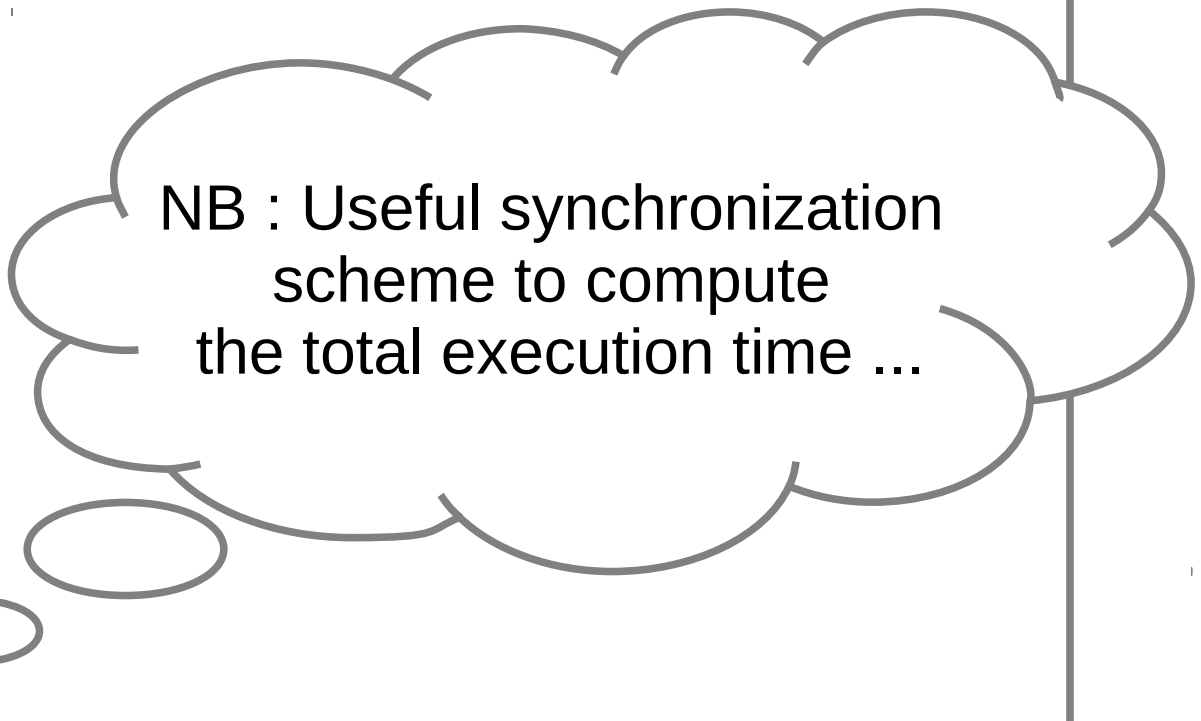
Example – critical

```
#include <omp.h>

int main(){
  ...
  #pragma omp parallel shared(TotalSum) private(mySum)
  {
    ...
    work();  //let's do our work
    #pragma omp critical
    // let's update our shared variable
    TotalSum += mySum;
   }
}
```



Example – barrier & master (1/2)

```
#include <omp.h>

int main(){
  ...
  #pragma omp parallel
  {
    #pragma omp barrier
    #pragma omp master
       Gettimeofday(...);

       work();  //let's do our work
      #pragma omp barrier
      #pragma omp master
      {
         gettimeofday(...);
         printf("execution time", ...);
      }
   }
}
```

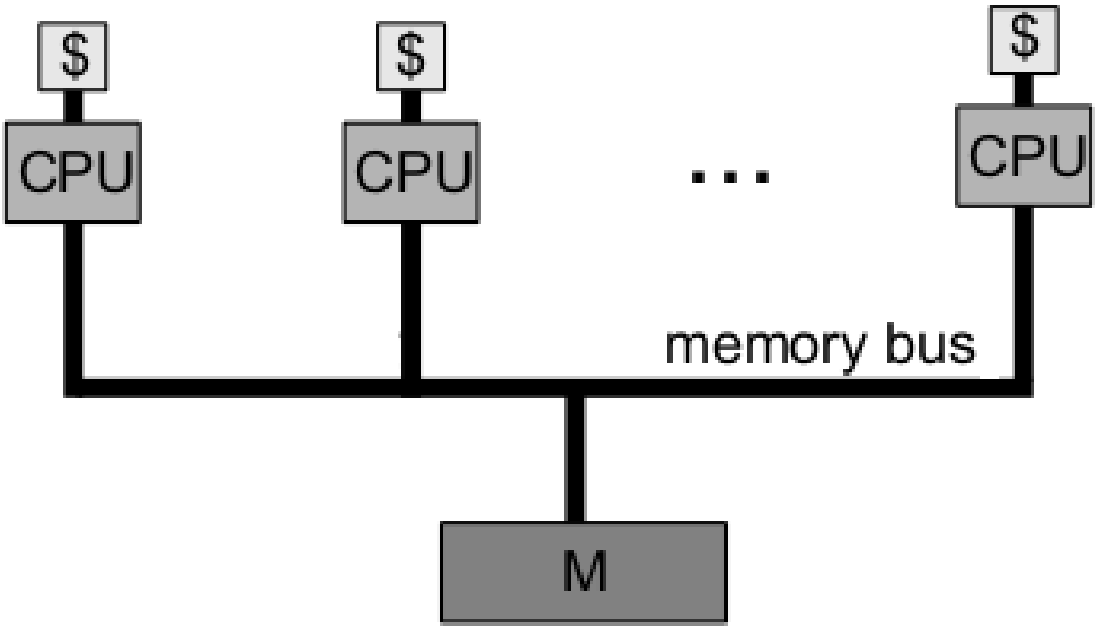


Example – barrier & master (2/2)

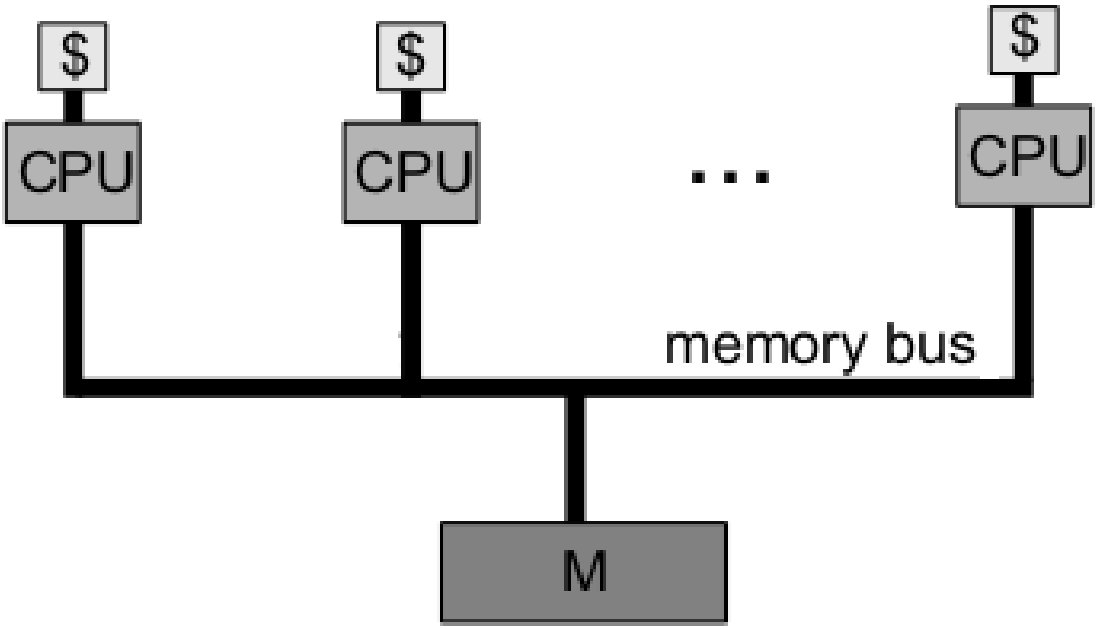
```
#include <omp.h>

int main(){
  ...
  #pragma omp parallel
  {
    #pragma omp barrier
    #pragma omp master
       Gettimeofday(...);

       work();  //let's do our wo
      #pragma omp barrier
      #pragma omp master
      {
         gettimeofday(...);
         printf("execution time", ...);
      }
   }
}
```

Suppose we have a program that can be divided into 5 concurrent tasks. One of these tasks requires twice the execution time than the other 4. **Max Speed-up=??**

Do I always share a shared variable !?!?

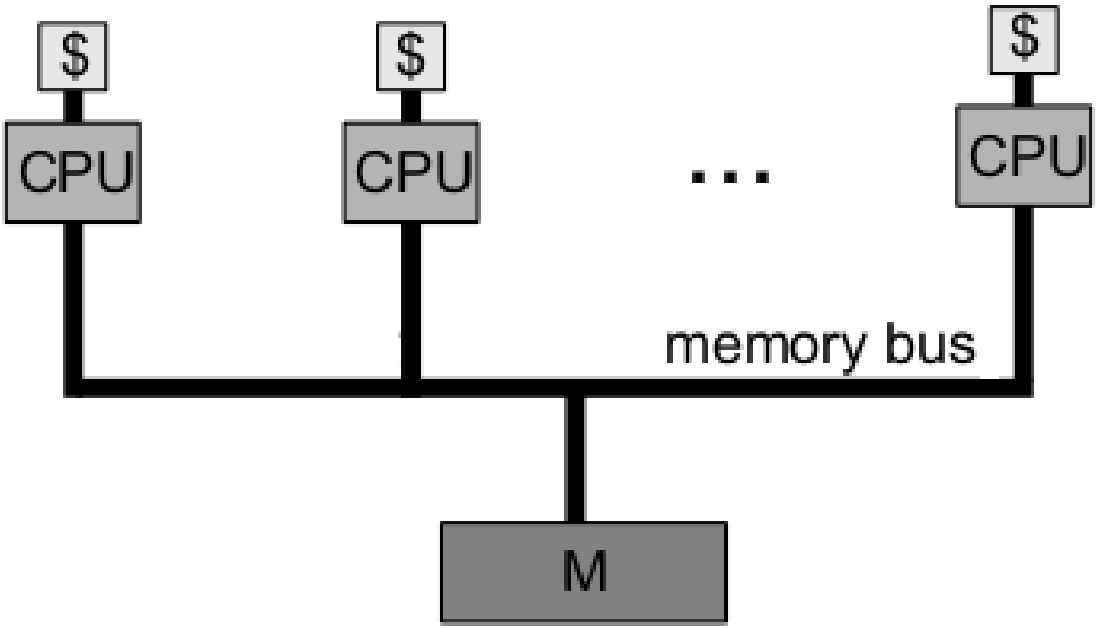


Do I always share a shared variable !?!?



- Keep in mind that we have machines with complex Memory hierarchy :
  - Multi-level caches.
  - False sharing cases.
  - ...

Do I always share a shared variable !?!?



- Keep in mind that we have machines with complex Memory hierarchy :
  - Multi-level caches.
  - False sharing cases.
  - ...

Example - flush()

```
#include <omp.h>

int main(){
  ...
  // initially a = b = 0 !!
  #pragma omp parallel num_threads(2)
  {
    // Thread 0 :
    if (omp_get_thread_num()== 0)
       a = 2;
    // Thread 1 :
    if (omp_get_thread_num()== 1)
       b = 2;

    #pragma omp flush(a,b)
    printf("Thread %d: a=%d,b=%d\n", omp_get_thread_num(), a, b );
  }
  ...
}
```