# Single Construct (14A)

- Loop
-

Please send corrections (or suggestions) to youngwlim@hotmail.com.

This document was produced by using OpenOffice and Octave.

# Master thread and a team of threads

Immediately preceding the **parallel block**,
one **thread** will be executing the code.

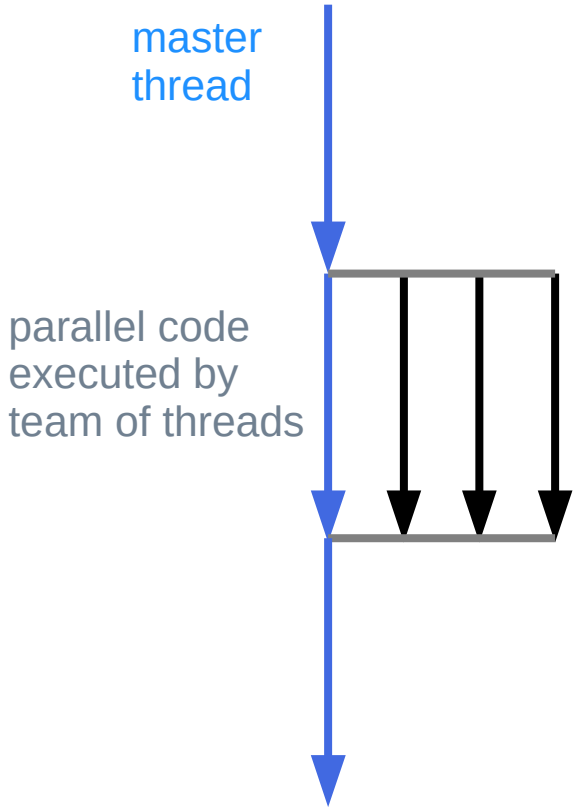
In the **main** program this is the **initial thread**.

At the start of the block, a new team of **threads** is created,
and the **thread** that was active before the block becomes
the **master thread** of that team.

After the block, only the **master thread** is active.

Inside the block there is **team** of **threads**:

each **thread** in the **team**
- executes the body of the block,
- can access all variables of the surrounding environment.




https://pages.tacc.utexas.edu/~eijkhout/pcse/html/omp-parallel.html

# **single** construct

- the specified structured block is executed
  - by only one of the threads in the team
  - in the context of its implicit task.

- The executing thread need not be the master thread

- the other threads in the team
  - do not execute the block
  - wait at an implicit barrier at the end of the **single** construct unless a **nowait** clause is specified.

https://pages.tacc.utexas.edu/~eijkhout/pcse/html/omp-parallel.html

# **single** construct

#pragma omp **single** [clause[ [,] clause] ... ] *new-line*
*structured-block*

where clause is one of the following:

**private**(list)
**firstprivate**(list)
**copyprivate**(list)
**allocate**([allocator :] list)
**nowait**

https://www.openmp.org/spec-html/5.0/openmpsu38.html#x60-1090002.8.2

# Single

The **single** construct specifies that
the associated structured block is
executed by only one of the threads in the team
(not necessarily the master thread),
in the context of its **implicit task**.

The other threads in the team,
which do not execute the block,
wait at an **implicit barrier**
at the end of the single construct
unless a **nowait** clause is specified.

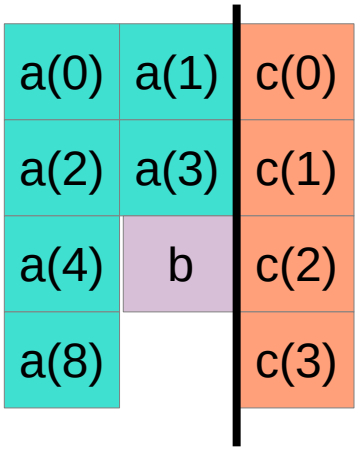
https://www.openmp.org/spec-html/5.0/openmpsu38.html

# Single

denotes block of code
to be executed by only one thread
- first thread to arrive is chosen
- **implicit barrier** at end

```
#pragma omp parallel
{
   a();
   #pragma omp single
   {
      b();
   }  // threads wait here for single
   c();
}
```

chosen

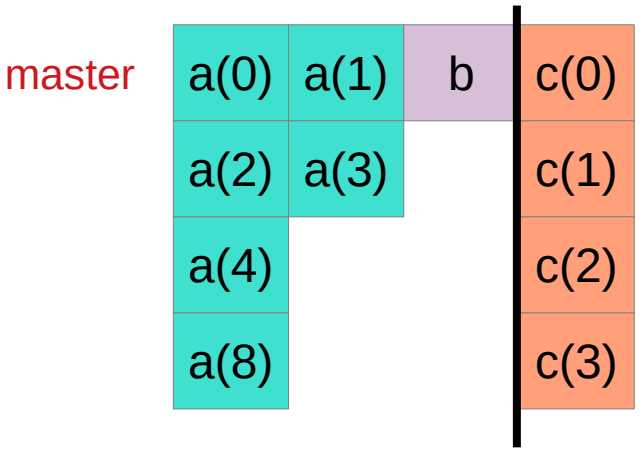



https://www.intel.com/content/dam/www/public/apac/xa/en/pdfs/ssg/Programming_with_OpenMP-Linux.pdf

# Master

Denotes block of code to be executed only by the master thread
No **implicit barrier** at end

```
#pragma omp parallel
{
   a();
   #pragma omp master
   {     // if not master skip to next stmtp
         b();
   }
   c();
|
```




https://www.intel.com/content/dam/www/public/apac/xa/en/pdfs/ssg/Programming_with_OpenMP-Linux.pdf

# Single (1)

```
int main()
{
    int salaries1 = 0;
    int salaries2 = 0;

    for (int employee = 0; employee < 25000; employee++)
    {
        salaries1 += fetchTheSalary(employee, Co::Company1);
    }

    std::cout << "Salaries1: " << salaries1 << std::endl;

    for (int employee = 0; employee < 25000; employee++)
    {
        salaries2 += fetchTheSalary(employee, Co::Company2);
    }

    std::cout << "Salaries2: " << salaries2 << std::endl;

    return 0;
}
```

http://jakascorner.com/blog/2016/06/omp-single.html

# Single (2)

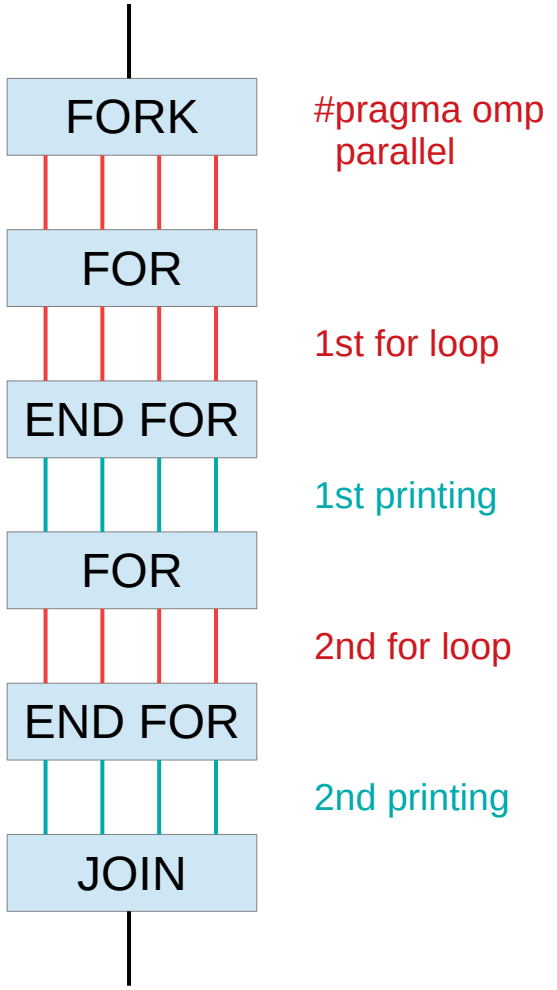
```
int salaries1 = 0;
int salaries2 = 0;

#pragma omp parallel shared(salaries1, salaries2)
{
   #pragma omp for reduction(+: salaries1)
   for (int employee = 0; employee < 25000; employee++)
   {
      salaries1 += fetchTheSalary(employee, Co::Company1);
   }

   std::cout << "Salaries1: " << salaries1 << std::endl;

   #pragma omp for reduction(+: salaries2)
   for (int employee = 0; employee < 25000; employee++)
   {
      salaries2 += fetchTheSalary(employee, Co::Company2);
   }

   std::cout << "Salaries2: " << salaries2 << std::endl;
}
```




http://jakascorner.com/blog/2016/06/omp-single.html

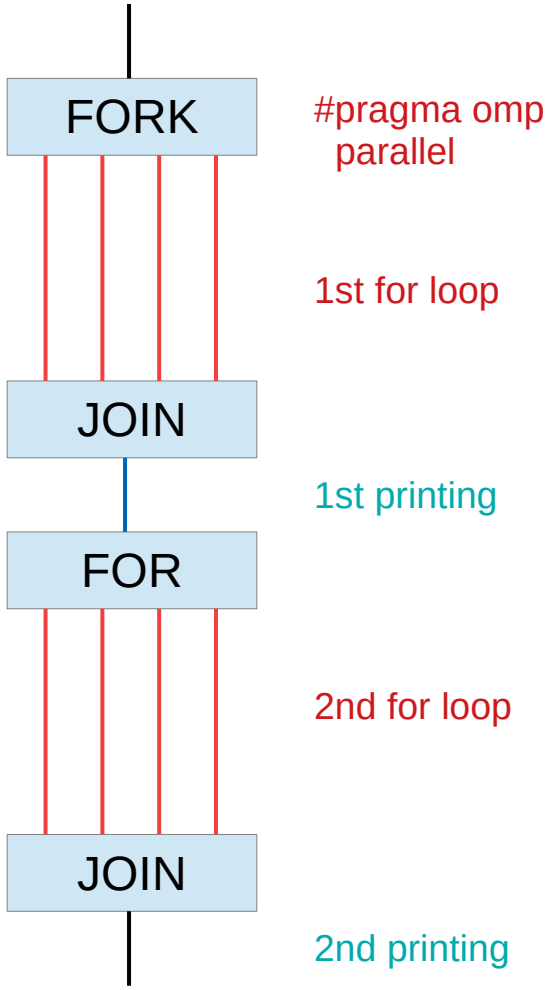
# Single (v1)

```
#pragma omp parallel for reduction(+: salaries1)
for (int employee = 0; employee < 25000; employee++)
{
    salaries1 += fetchTheSalary(employee, Co::Company1);
}

std::cout << "Salaries1: " << salaries1 << std::endl;

#pragma omp parallel for reduction(+: salaries2)
for (int employee = 0; employee < 25000; employee++)
{
    salaries2 += fetchTheSalary(employee, Co::Company2);
}

std::cout << "Salaries2: " << salaries2 << std::endl;
```

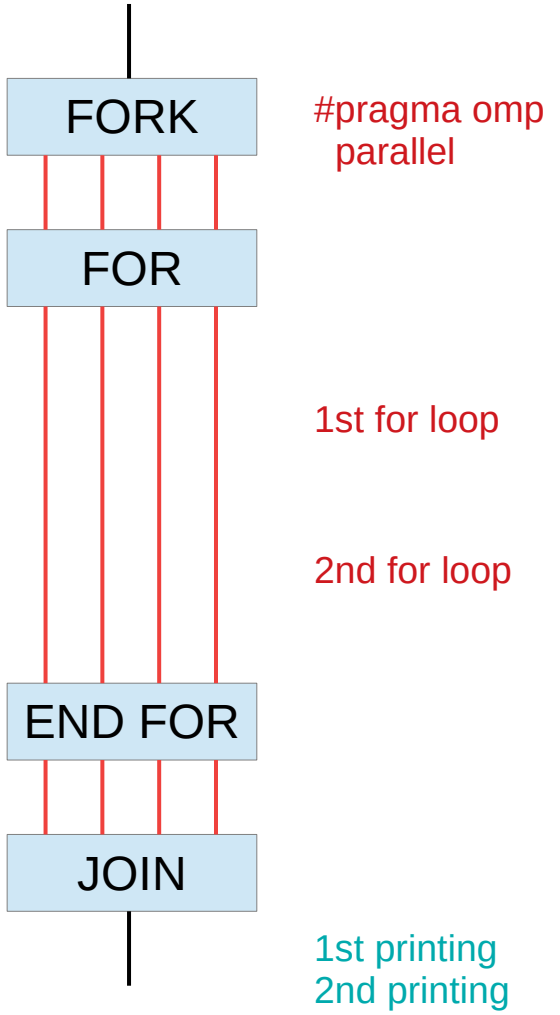



http://jakascorner.com/blog/2016/06/omp-single.html

# Single (v2)

```
#pragma omp parallel for reduction(+: salaries1, salaries2)
for (int employee = 0; employee < 25000; employee++)
{
    salaries1 += fetchTheSalary(employee, Co::Company1);
    salaries2 += fetchTheSalary(employee, Co::Company2);
}
```

```
std::cout << "Salaries1: " << salaries1 << "\n"
          << "Salaries2: " << salaries2 << std::endl;
```




http://jakascorner.com/blog/2016/06/omp-single.html

# Single (v3)

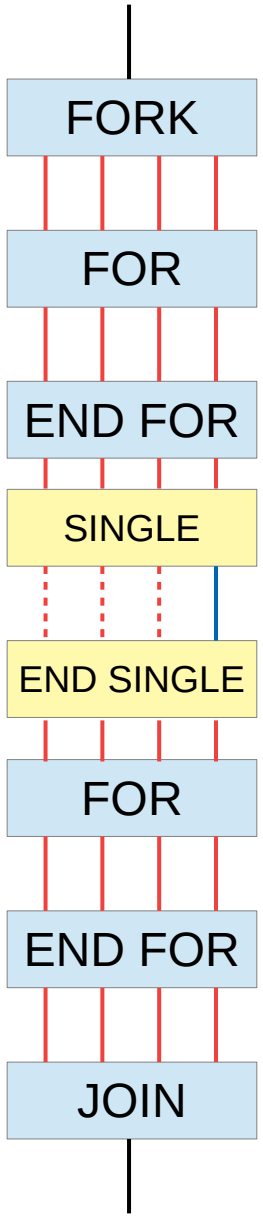
```
#pragma omp parallel shared(salaries1, salaries2)
{
    #pragma omp for reduction(+: salaries1)
    for (int employee = 0; employee < 25000; employee++)
    {
        salaries1 += fetchTheSalary(employee, Co::Company1);
    }

    #pragma omp single
    {
        std::cout << "Salaries1: " << salaries1 << std::endl;
    }

    #pragma omp for reduction(+: salaries2)
    for (int employee = 0; employee < 25000; employee++)
    {
        salaries2 += fetchTheSalary(employee, Co::Company2);
    }
}

std::cout << "Salaries2: " << salaries2 << std::endl;
```

http://jakascorner.com/blog/2016/06/omp-single.html




#pragma omp parallel

1st for loop

1st printing

2nd for loop

2nd printing

## References

[1] en.wikipedia.org
[2] M Harris, http://beowulf.lcs.mit.edu/18.337-2008/lectslides/scan.pdf