Въведение в OpenMP.

Курс „Паралелно програмиране“

ИНСТИТУТ *за* СЪВРЕМЕННИ
ФИЗИЧЕСКИ ИЗСЛЕДВАНИЯ

Стоян Мишев

План

Определение на OpenMP

`https://www.openmp.org/`

Стандарт за писане на програми за създаване и управление на нишки.

Определение на OpenMP

`https://www.openmp.org/`

Стандарт за писане на програми за създаване и управление на нишки.

Има варианти за C, C++, Фортран и др.

Определение на OpenMP

`https://www.openmp.org/`

Стандарт за писане на програми за създаване и управление на нишки.

Има варианти за C, C++, Фортран и др.

Нишка е част от програма, която има достъп до *собствена* част от паметта, както и до общи ресурси за компютъра - памет, външни устройства, мрежа и т.н.

Определение на OpenMP

`https://www.openmp.org/`

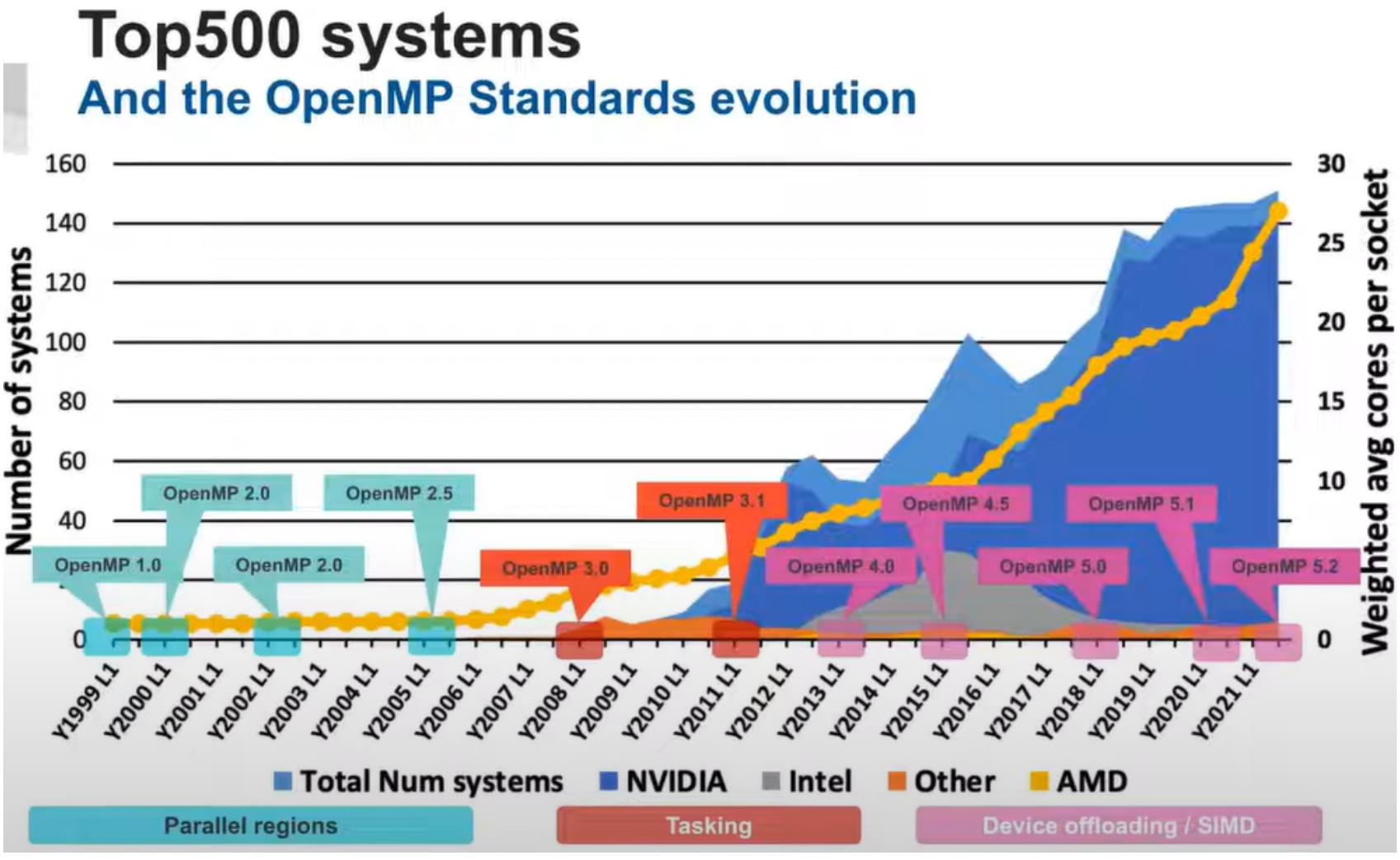
Стандарт за писане на програми за създаване и управление на нишки.

Има варианти за C, C++, Фортран и др.

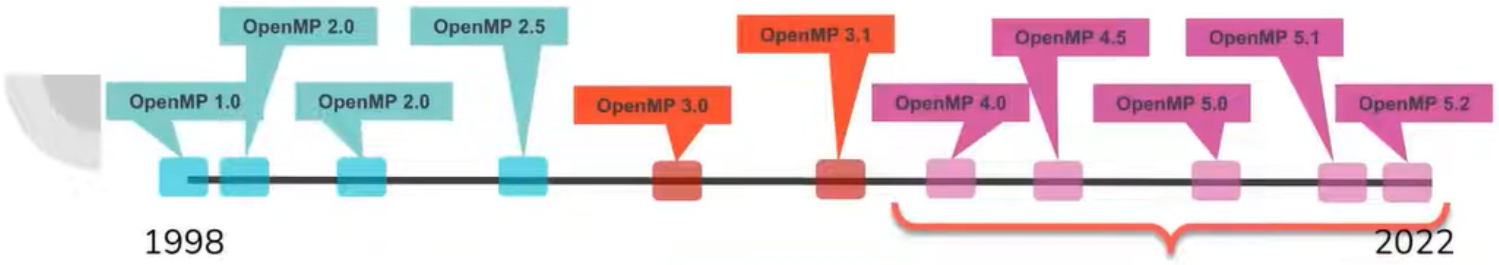
Нишка е част от програма, която има достъп до *собствена* част от паметта, както и до общи ресурси за компютъра - памет, външни устройства, мрежа и т.н.

Лекцията е на основата на клипа

`https://www.youtube.com/watch?v=IsBgW7-yldA`


Top500 systems
And the OpenMP Standards evolution
Number of systems
Weighted avg cores per socket
160
140
120
100
80
60
40
0
30
25
20
15
10
0
OpenMP 1.0
OpenMP 2.0
OpenMP 2.0
OpenMP 2.5
OpenMP 3.0
OpenMP 3.1
OpenMP 4.0
OpenMP 4.5
OpenMP 5.0
OpenMP 5.1
OpenMP 5.2
Y1999 L1
Y2000 L1
Y2001 L1
Y2002 L1
Y2003 L1
Y2004 L1
Y2005 L1
Y2006 L1
Y2007 L1
Y2008 L1
Y2009 L1
Y2010 L1
Y2011 L1
Y2012 L1
Y2013 L1
Y2014 L1
Y2015 L1
Y2016 L1
Y2017 L1
Y2018 L1
Y2019 L1
Y2020 L1
Y2021 L1
Total Num systems
NVIDIA
Intel
Other
AMD
Parallel regions
Tasking
Device offloading / SIMD

Версии на OpenMP



OpenMP 4.0
- target [data]
- declare target
- target update
- simd
- declare simd
- loop simd
- Teams
- distribute

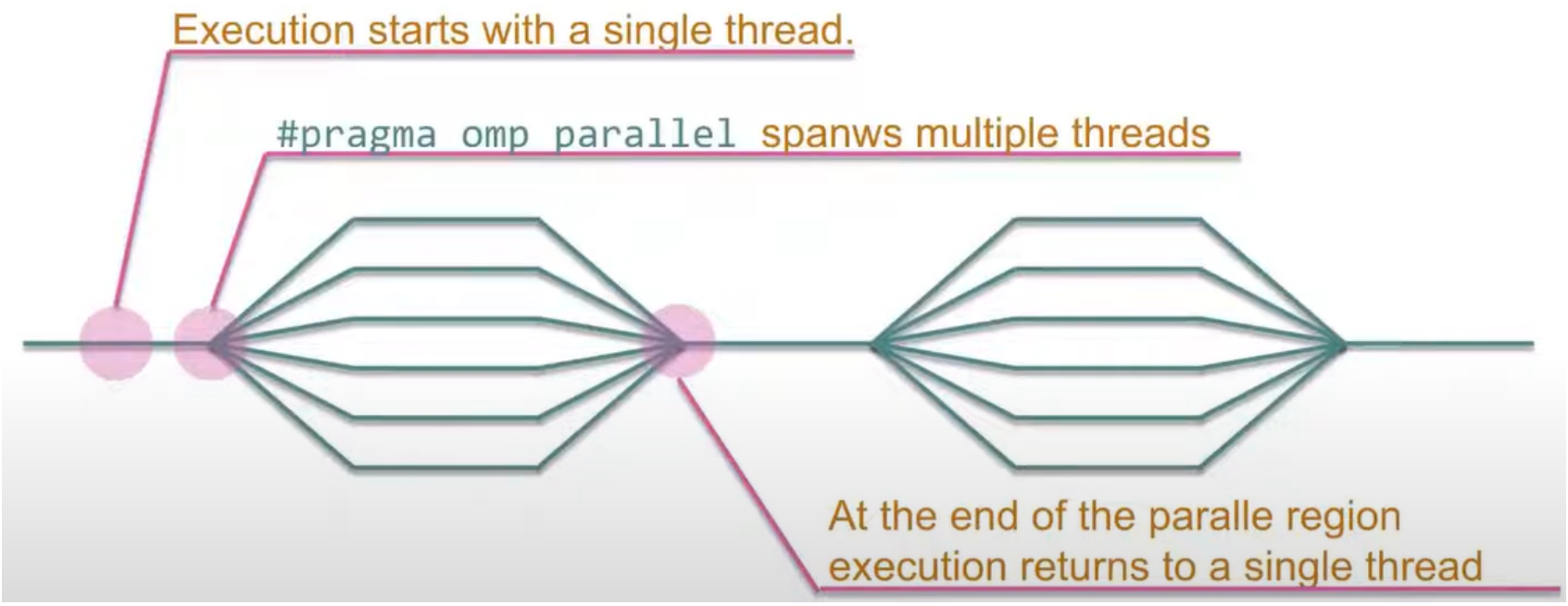
OpenMP 4.5
- taskloop
- taskloop simd
- target enter data
- target exit data
- target simd

OpenMP 5.0
- allocate
- declare mapper
- declare variant
- Memory spaces
- parallel loop
- teams loop
- requires
- scan

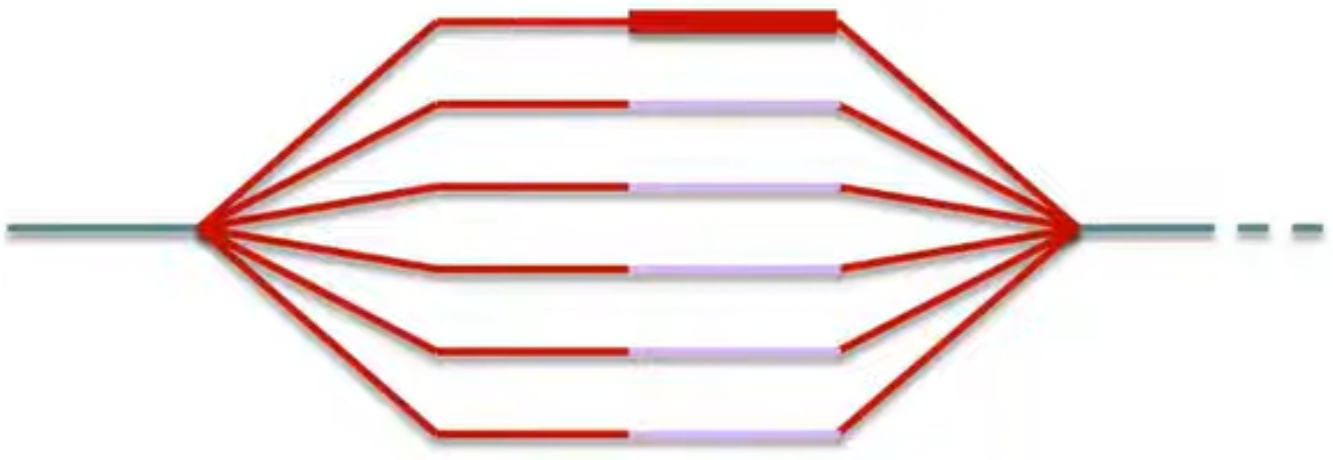
OpenMP 5.1
- Variant dispatch
- declare mapper
- Memory spaces
- parallel loop
- teams loop
- Interop
- masked
- scope
- tile/unroll

OpenMP 5.2
- ompx
- Reorganization
- …

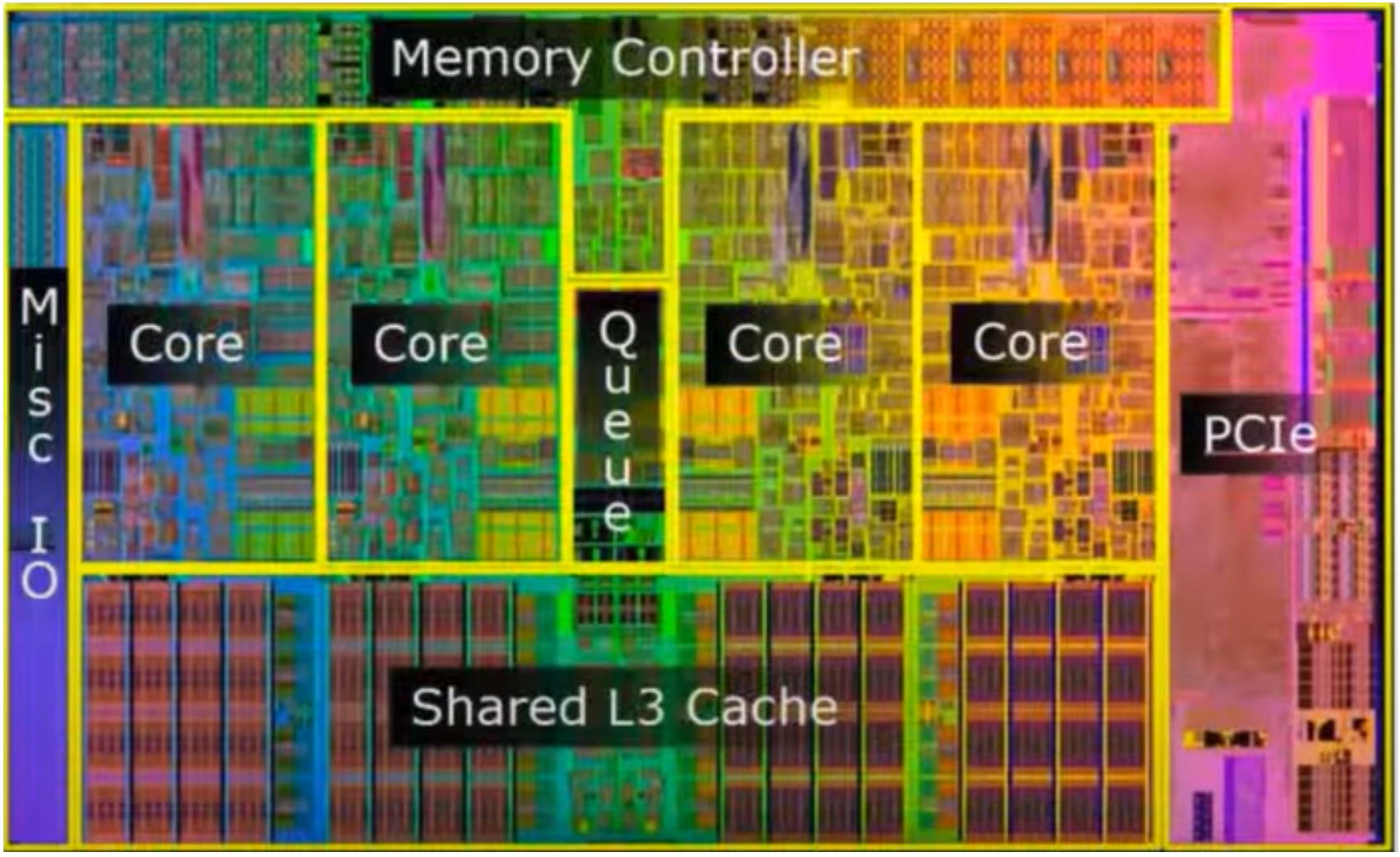
Създаване и *обединяване* на нишки

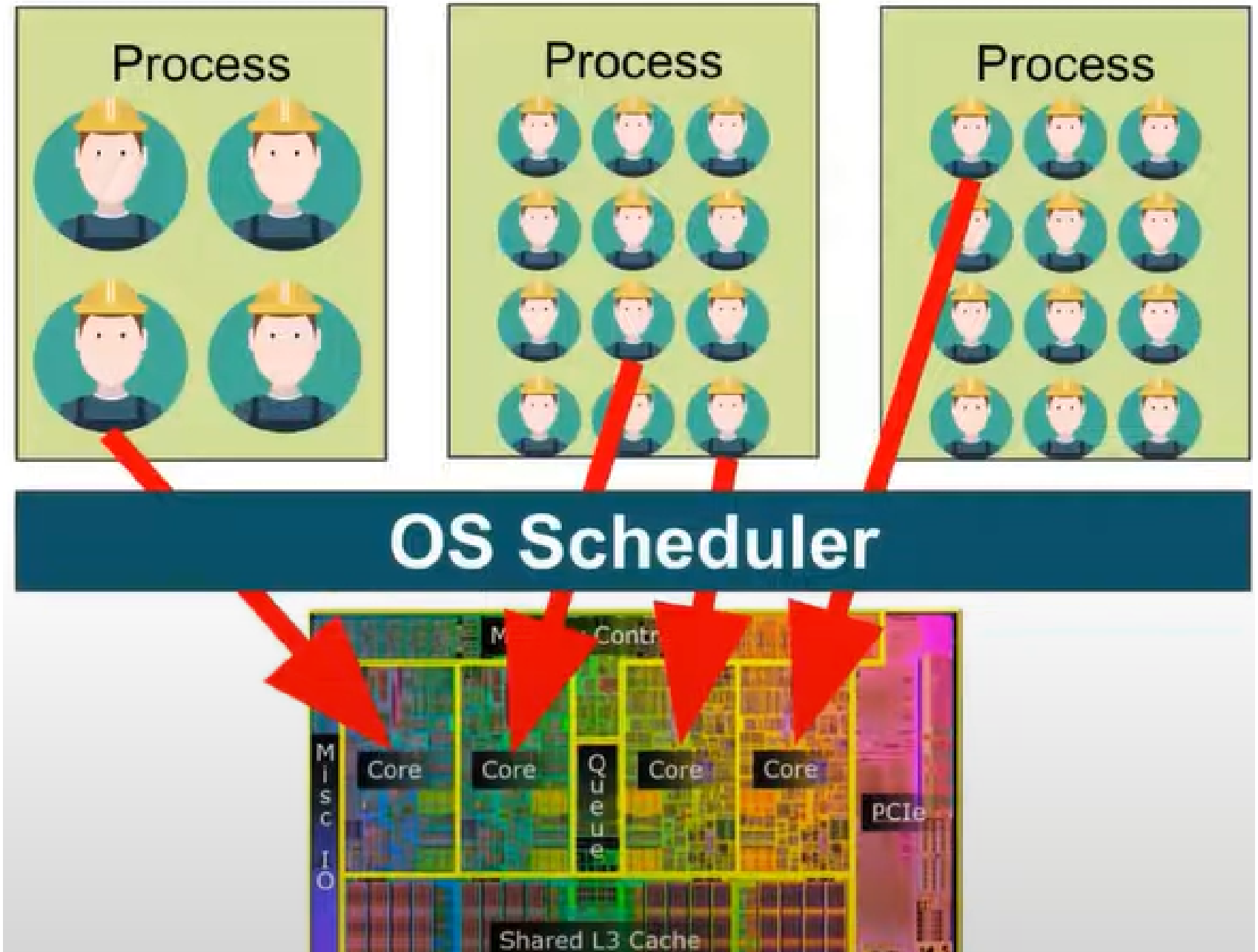


Създаване на нишки

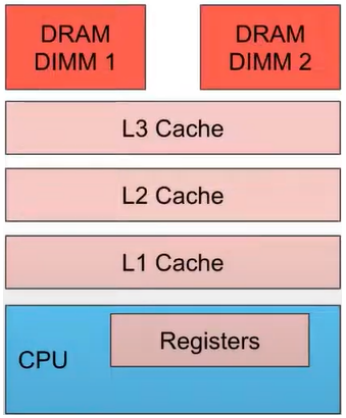


```
1        #pragma omp parallel{
2           printf("hi from %d\n", omp_get_num_thread());
3
4           #pragma omp master{
5              printf("This shows once \n");
6           }
7
8        }
```

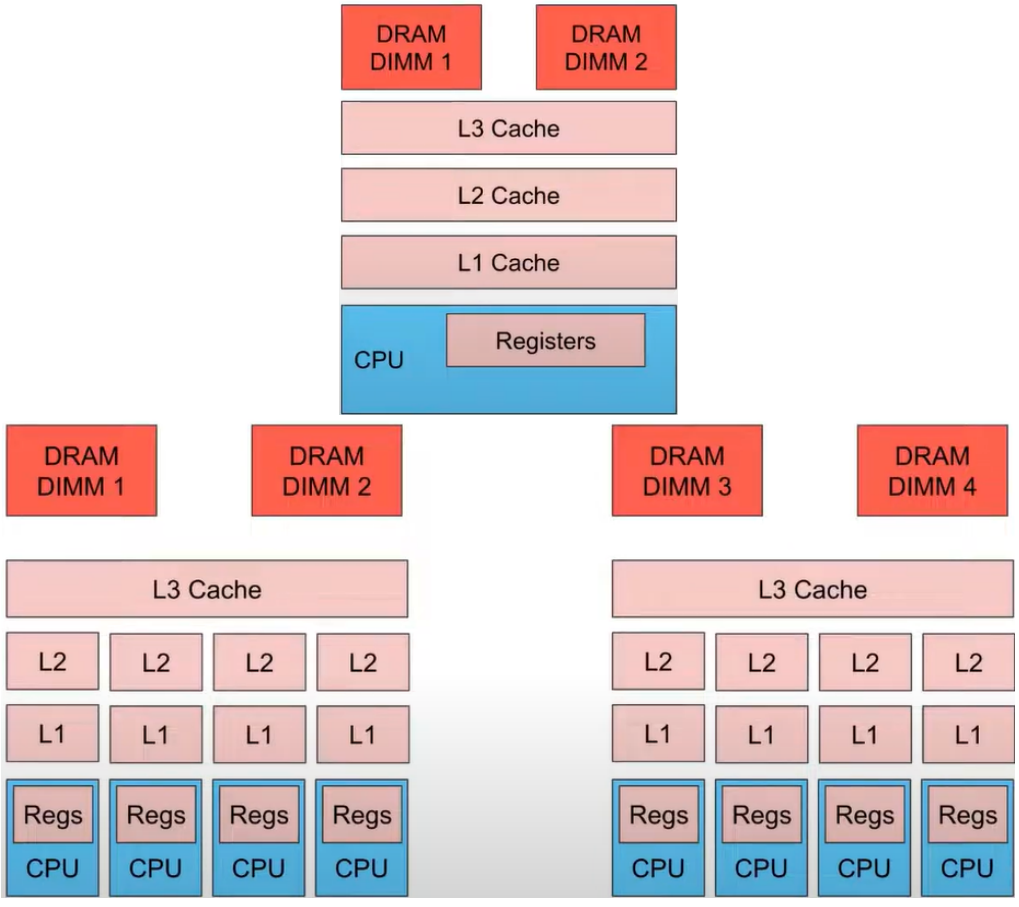

Memory Controller
Misc IO
Core
Core
Queue
Core
Core
PCIe
Shared L3 Cache


Process
Process
Process
OS Scheduler
Core
Core
Core
Core
Queue
Misc IO
PCIe
Shared L3 Cache

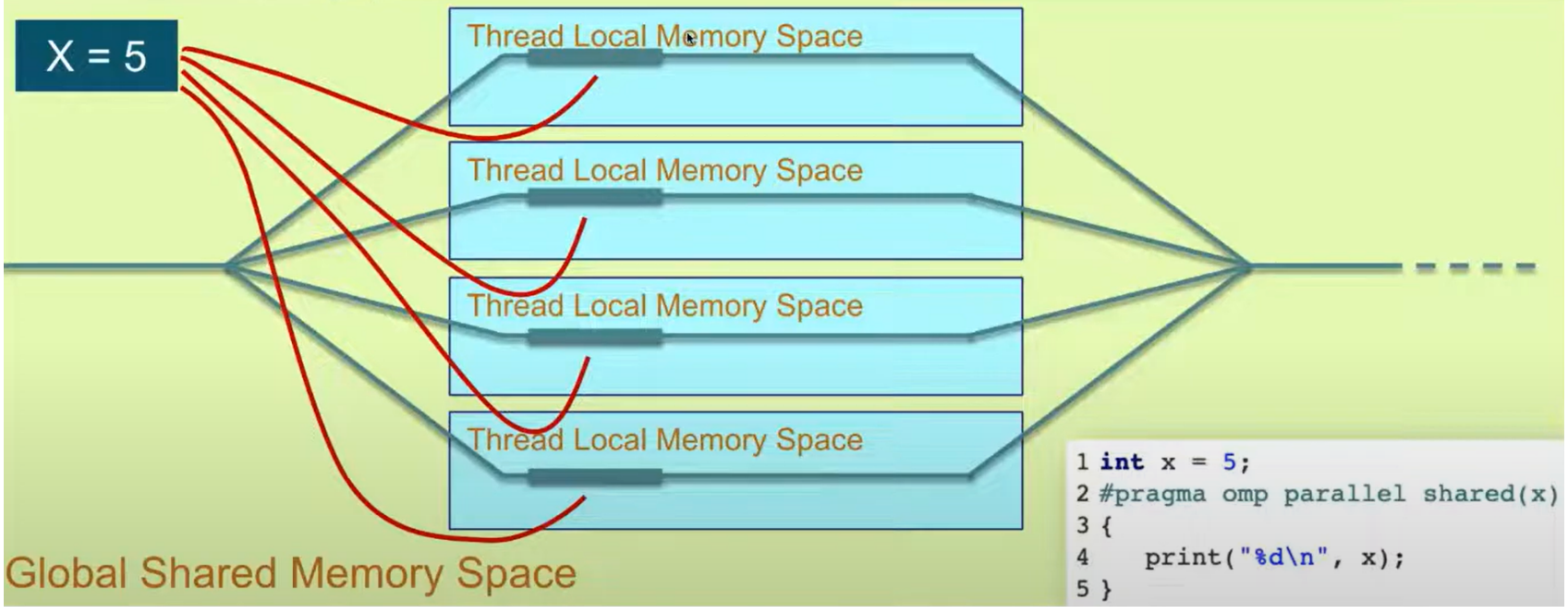
Една и много нишки



Една и много нишки



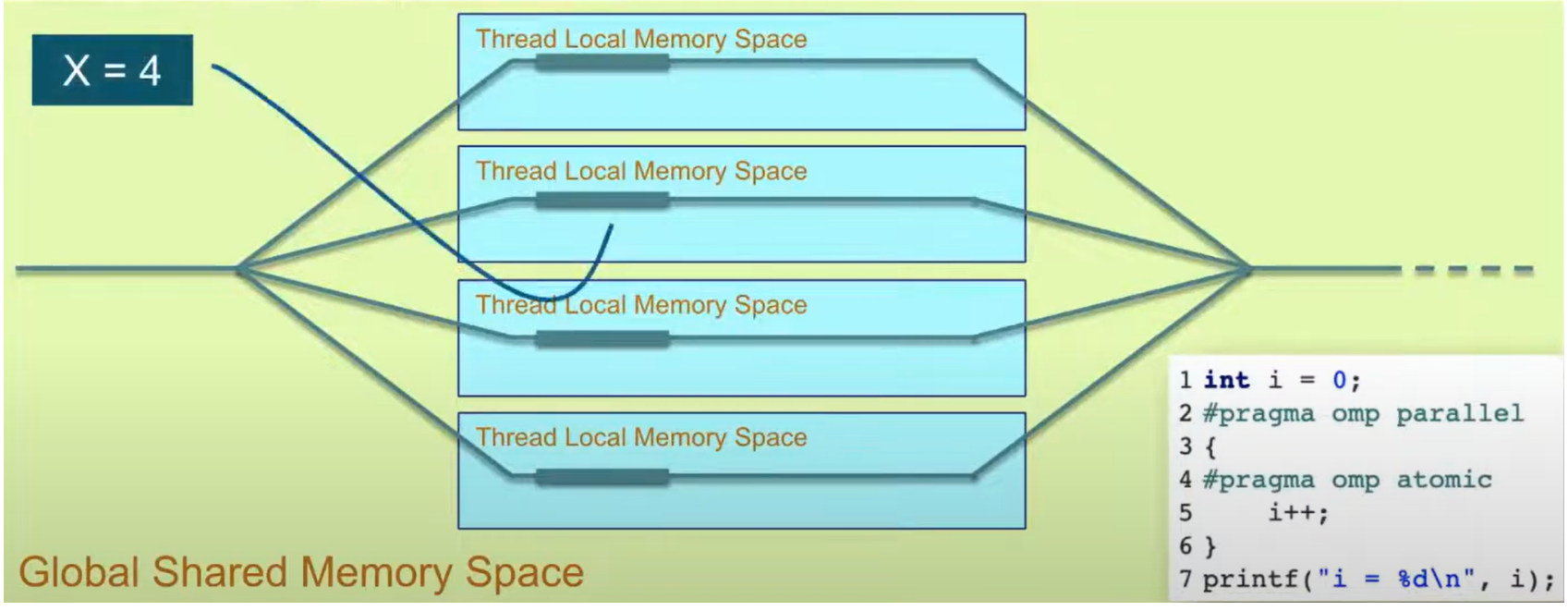
Shared


X = 5
Thread Local Memory Space
Thread Local Memory Space
Thread Local Memory Space
Thread Local Memory Space
Global Shared Memory Space
1 int x = 5;
2 #pragma omp parallel shared(x)
3 {
4 print("%d\n", x);
5 }

```
1  #include <stdio.h>
2  #include <omp.h>
3
4
5  int main() {
6      int i = 0;
7
8      #pragma omp parallel shared(i) num_threads(10000)
9      {
10         i++;
11     }
12
13     printf("i = %d\n",i);
14     return 0;
15 }
```

THE OPENMP MEMORY CLAUSES

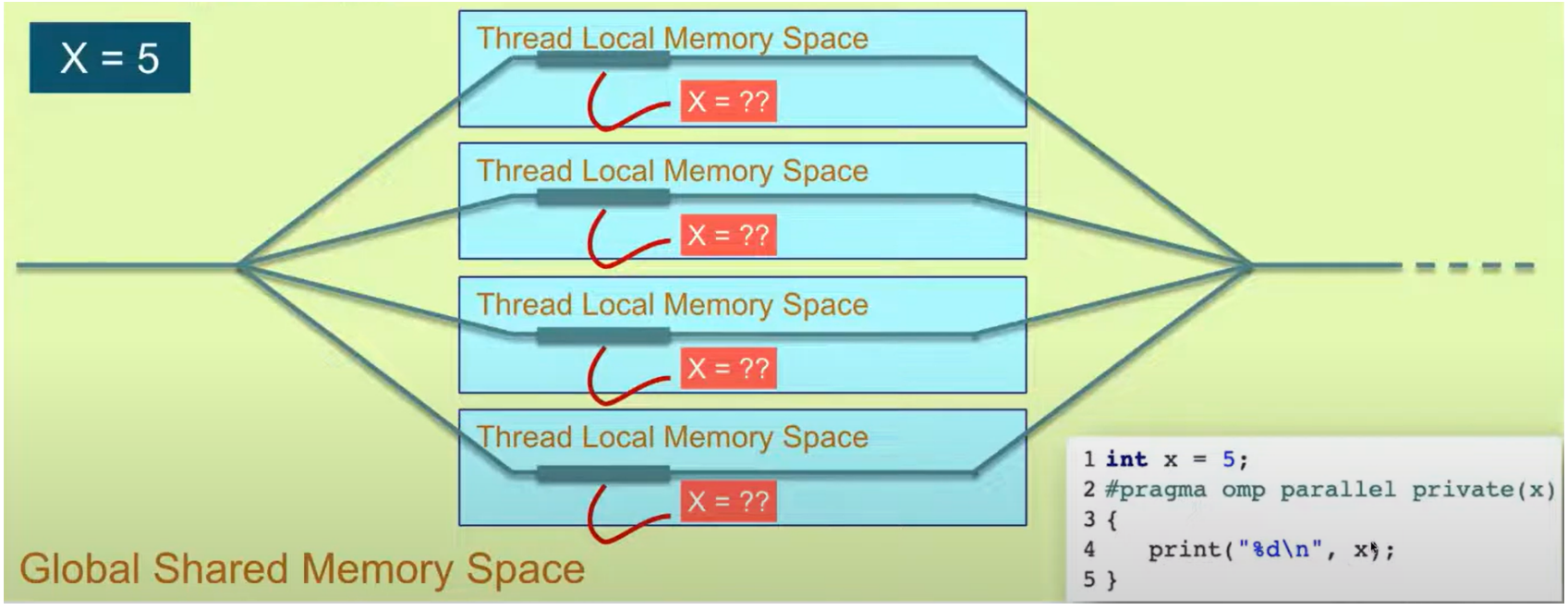
Atomic operations




```
1 int i = 0;
2 #pragma omp parallel
3 {
4 #pragma omp atomic
5     i++;
6 }
7 printf("i = %d\n", i);
```

```
1 #include <stdio.h>
2 #include <omp.h>
3
4
5 int main() {
6     int i = 0;
7
8     #pragma omp parallel shared(i) num_threads(10000)
9     {
10        #pragma omp atomic
11        i++;
12    }
13
14    printf("i = %d\n",i);
15    return 0;
16 }
```

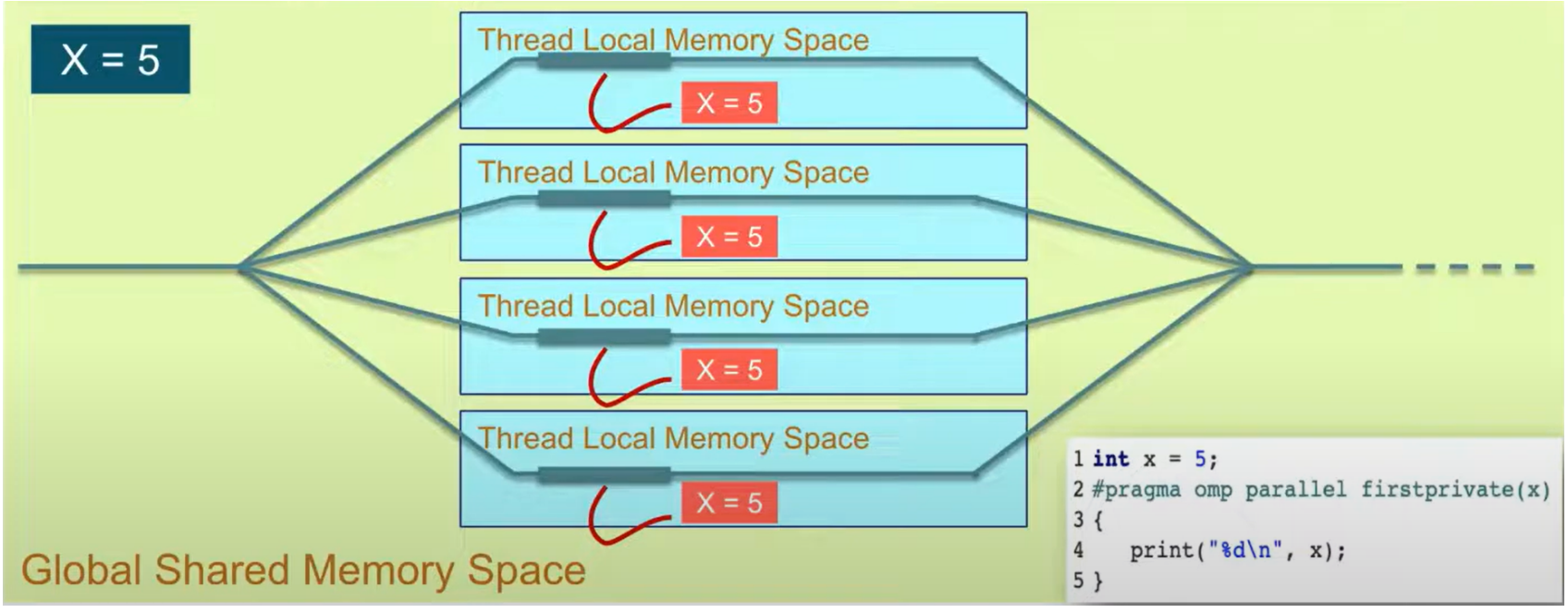
Private


X = 5
Thread Local Memory Space
X = ??
Thread Local Memory Space
X = ??
Thread Local Memory Space
X = ??
Thread Local Memory Space
X = ??
Global Shared Memory Space
1 int x = 5;
2 #pragma omp parallel private(x)
3 {
4 print("%d\n", x);
5 }

Private

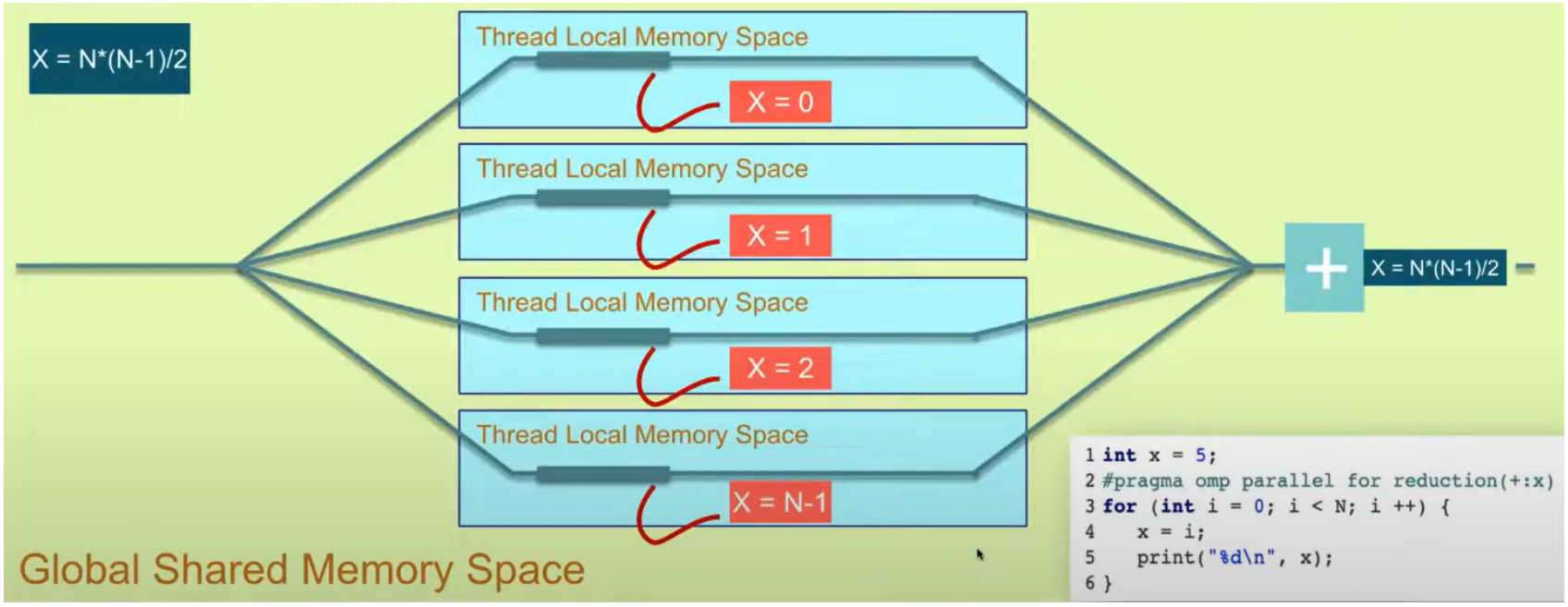
```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <omp.h>
4
5
6  int main() {
7      int i = 999;
8
9      printf("i is %d before parallel region\n",i);
10
11     #pragma omp parallel private(i) num_threads(10)
12     {
13         printf("Thread %d sees %d on memory %lx\n",
               omp_get_thread_num(), i, (unsigned long)&i)
               ;
14     }
15
16     return 0;
17 }
```

Firstprivate



```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <omp.h>
4
5
6 int main() {
7     int i[3] = {999,888,666};
8
9     printf("i is [%d,%d,%d] before parallel region\n",i
          [0],i[1],i[2]);
10
11    #pragma omp parallel firstprivate(i) num_threads
          (10)
12    {
13        printf("Thread %d sees [%d,%d,%d] on memory %lx
              \n", omp_get_thread_num(), i[0],i[1],i[2],
              (unsigned long)i);
14    }
15
16    return 0;
17 }
```

Reduction

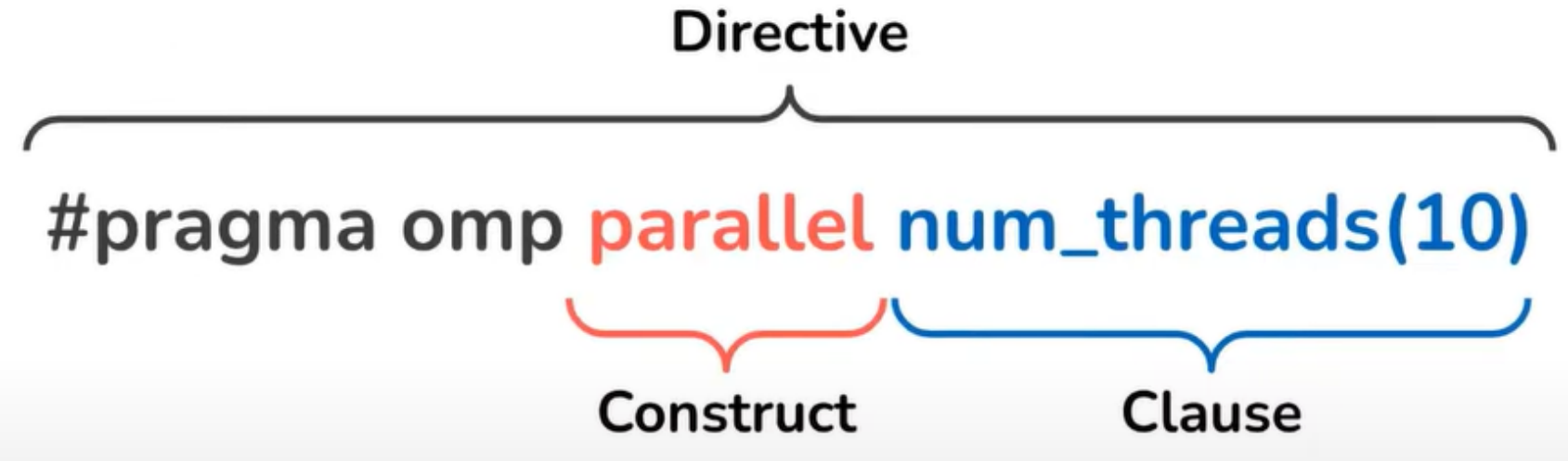

X = N*(N-1)/2
Thread Local Memory Space
X = 0
Thread Local Memory Space
X = 1
Thread Local Memory Space
X = 2
Thread Local Memory Space
X = N-1
Global Shared Memory Space
+
X = N*(N-1)/2
1 int x = 5;
2 #pragma omp parallel for reduction(+:x)
3 for (int i = 0; i < N; i ++) {
4 x = i;
5 print("%d\n", x);
6 }

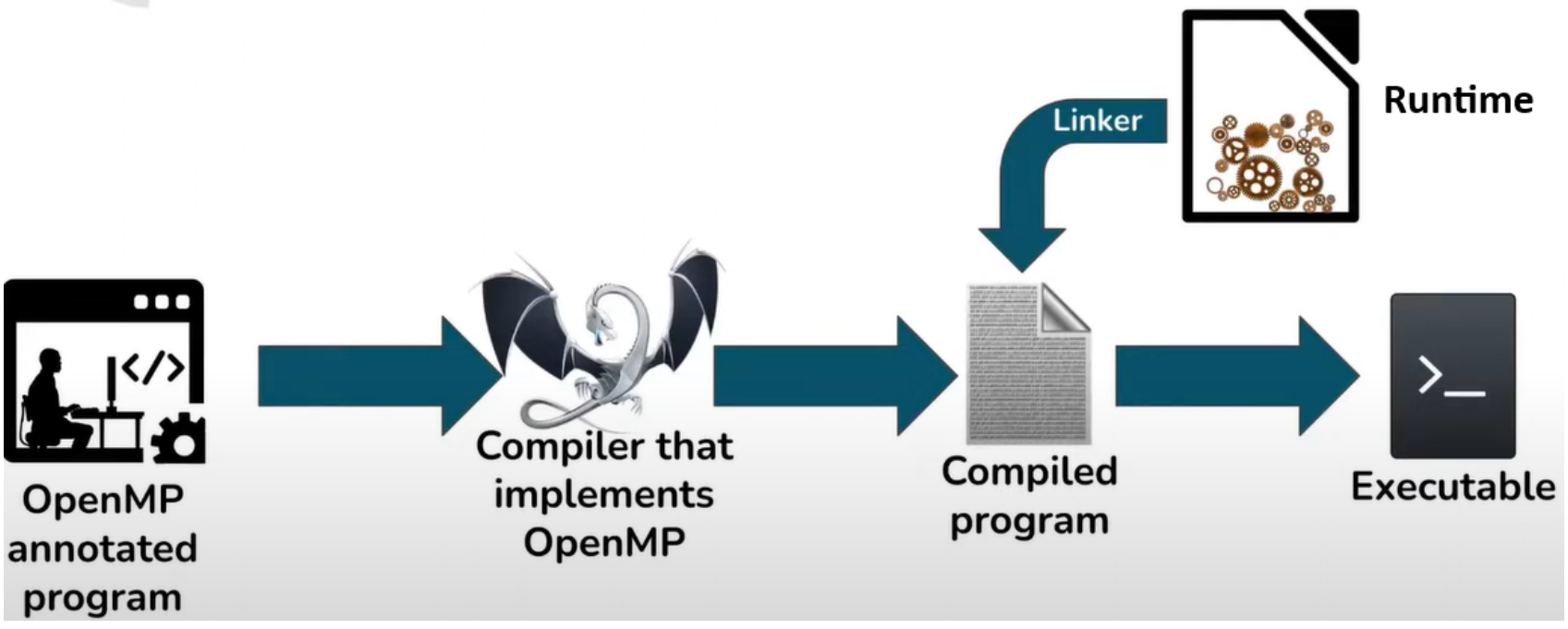
```
1 #include <stdio.h>
2 #include <omp.h>
3
4
5 int main() {
6     int i = 99;
7
8     printf("Value if i prior to parallel region is %d\n
          ",i);
9
10    // Private values are not transferred back
11    #pragma omp parallel private(i)
12    {
13        i=1000;
14    }
15    printf("Value if i after parallel region with
          private(i) is %d\n",i);
16
17    i = 0;
18    // Reductions for addition:
19    #pragma omp parallel reduction(+:i) num_threads(10)
```

Домашна работа

1. Прегледайте видеото на `https://www.youtube.com/watch?v=IsBgW7-yldA`.
2. Компилирайте “copc”-овете от `https://github.com/josemonsalve2/openmp_tutorial/blob/main/Labs/Lab1/C` на `neutronstar.iaps.institute` или другаде.

https://www.openmp.org/resources/openmp-compilers-tools/


Directive
#pragma omp parallel num_threads(10)
Construct
Clause


Linker
Runtime
OpenMP annotated program
Compiler that implements OpenMP
Compiled program
Executable

Function outlining


```
1  #pragma omp parallel{
2    printf("hello world\n");
3  }
```

```
1  void outlined_function(void* params)
2  {
3    printf("hello world\n");
4  }
5
6  __runtime_omp_parallel(outlined_function, params);
```