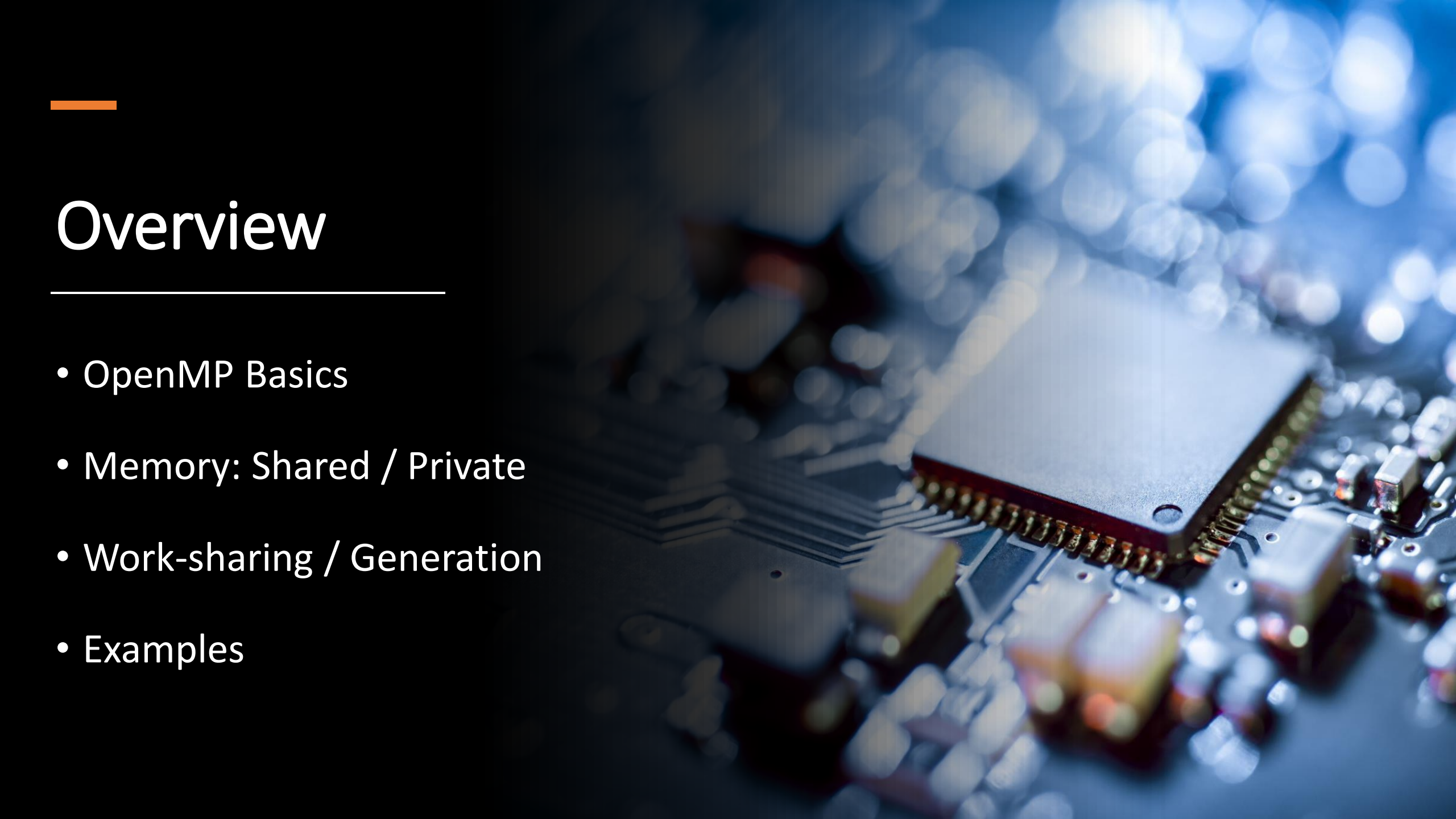
OpenMP
Open
Multi-Processing

From the Transistor to the Cluster
DERFAST - B.6.

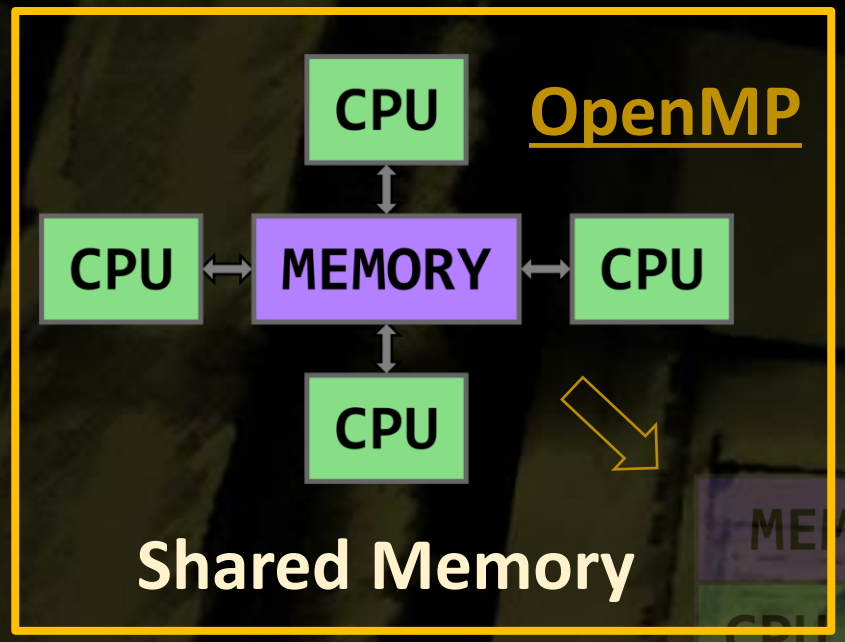
By Diego Roa

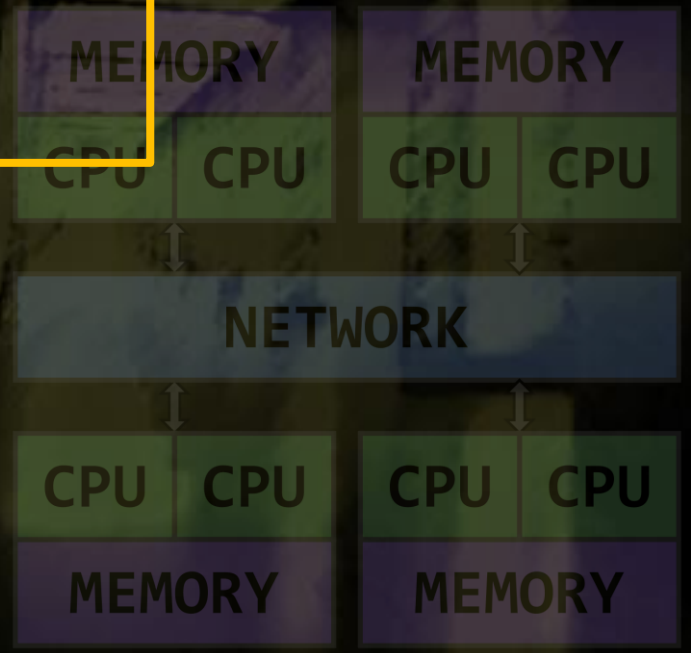
Overview

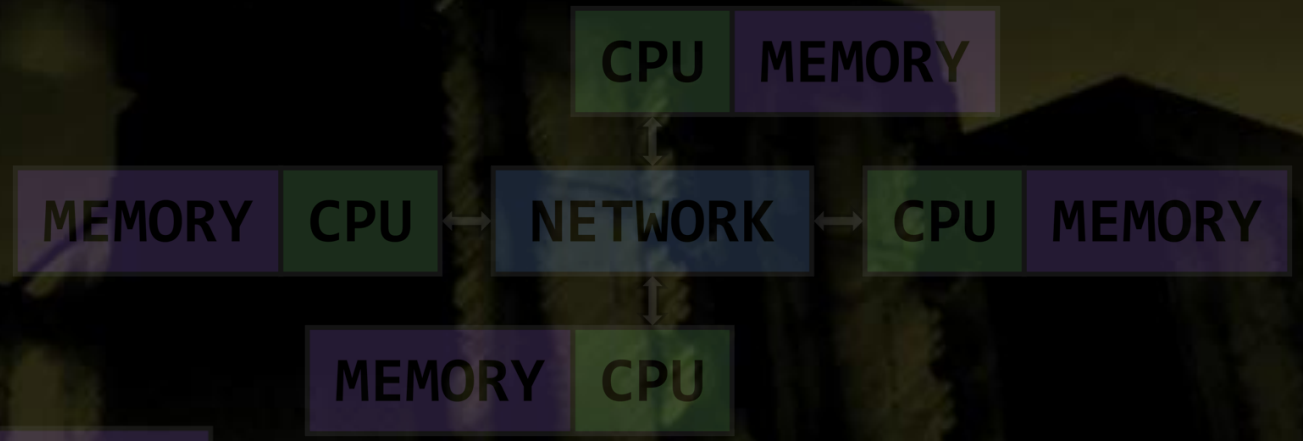
- OpenMP Basics
- Memory: Shared / Private
- Work-sharing / Generation
- Examples



Programming Models



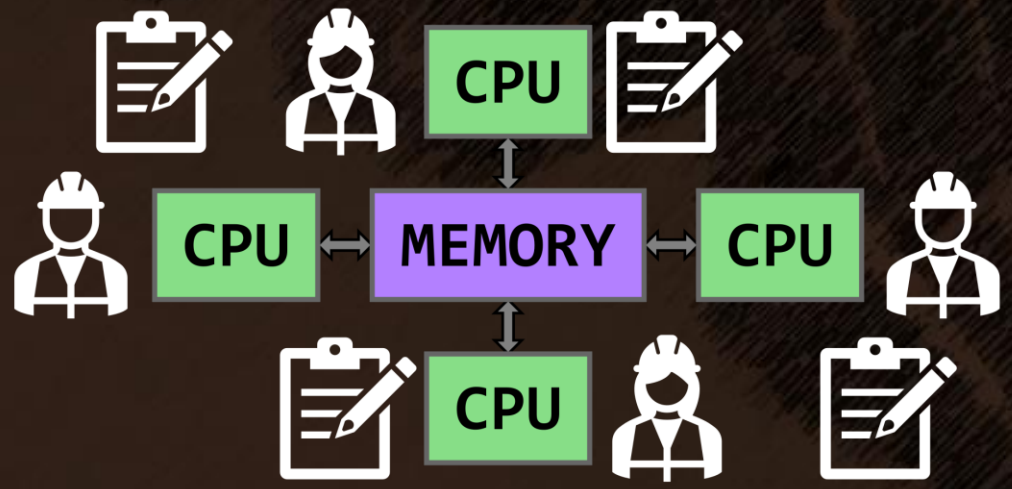


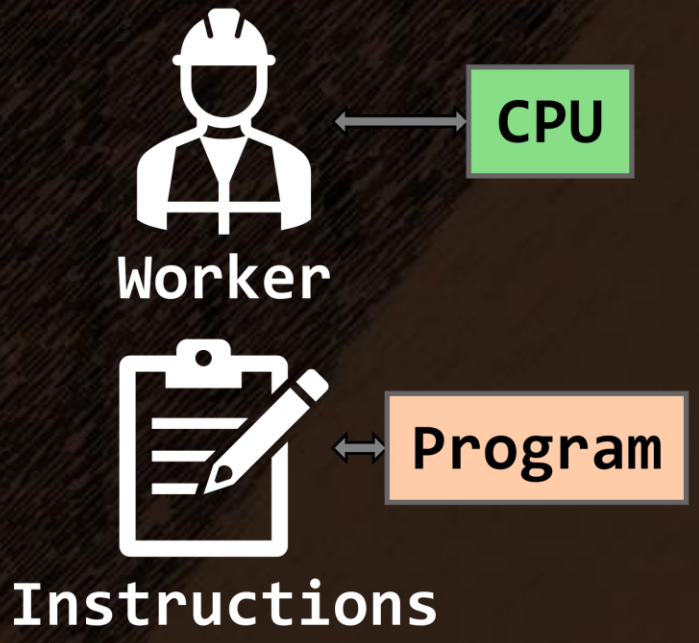


OpenMP

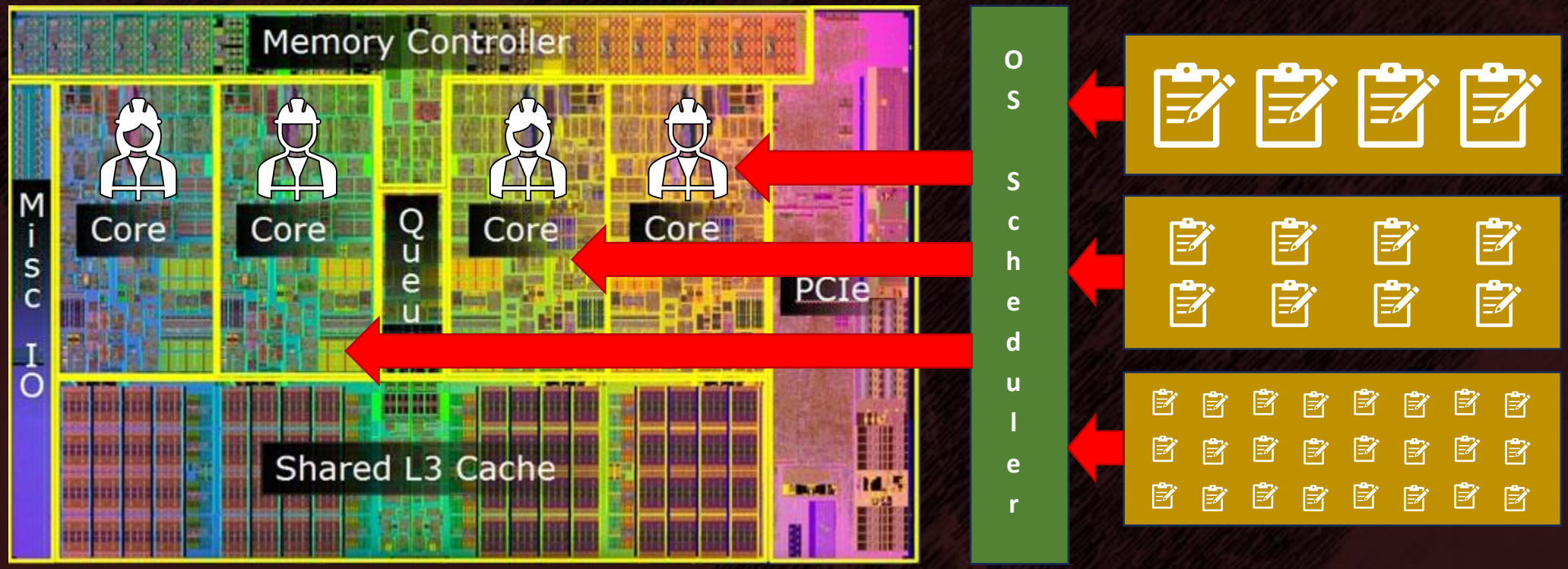
- **Not** a programming language
- An extension to C, C++ and Fortran
- Allows to express parallelism and workload distribution

- Worker Management
- Coordination / Communication
- Task assignments
- Resource sharing





Hardware vs Software

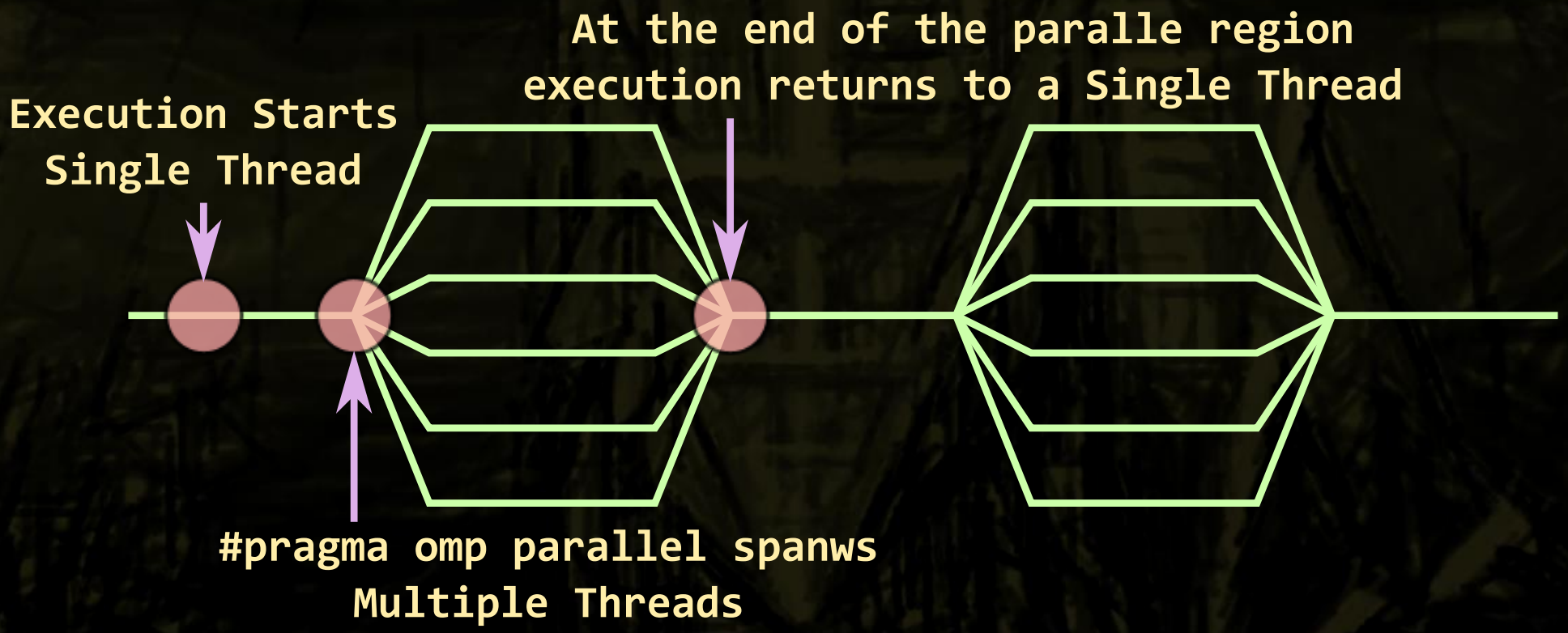


OS manages resource allocation - allows oversubscription of threads

OS creates process(es) - Threads live in a process

Threads define instructions and memory interaction

Fork-Join Model (Thread Creation)



OpenMP pragmas are compiler directives that manage parallelism in code by controlling thread behavior, workload distribution, and synchronization in multi-core environments.

Fork-Join Model - Parallel Directive

Thread creation:

- Spawn N threads

Work assignment:

- All threads get the same code

Synchronization:

- All threads wait at the end of the region

```
1 #pragma omp parallel
2 {
3   printf("Hi from %d\n", omp_get_thread_num());
4 }
```

Fork-Join Model - Primary Directive

Thread creation:

- Spawn N threads

Work assignment:

- Parallel: Region of code for all threads
- Master: Region of code for a single thread.

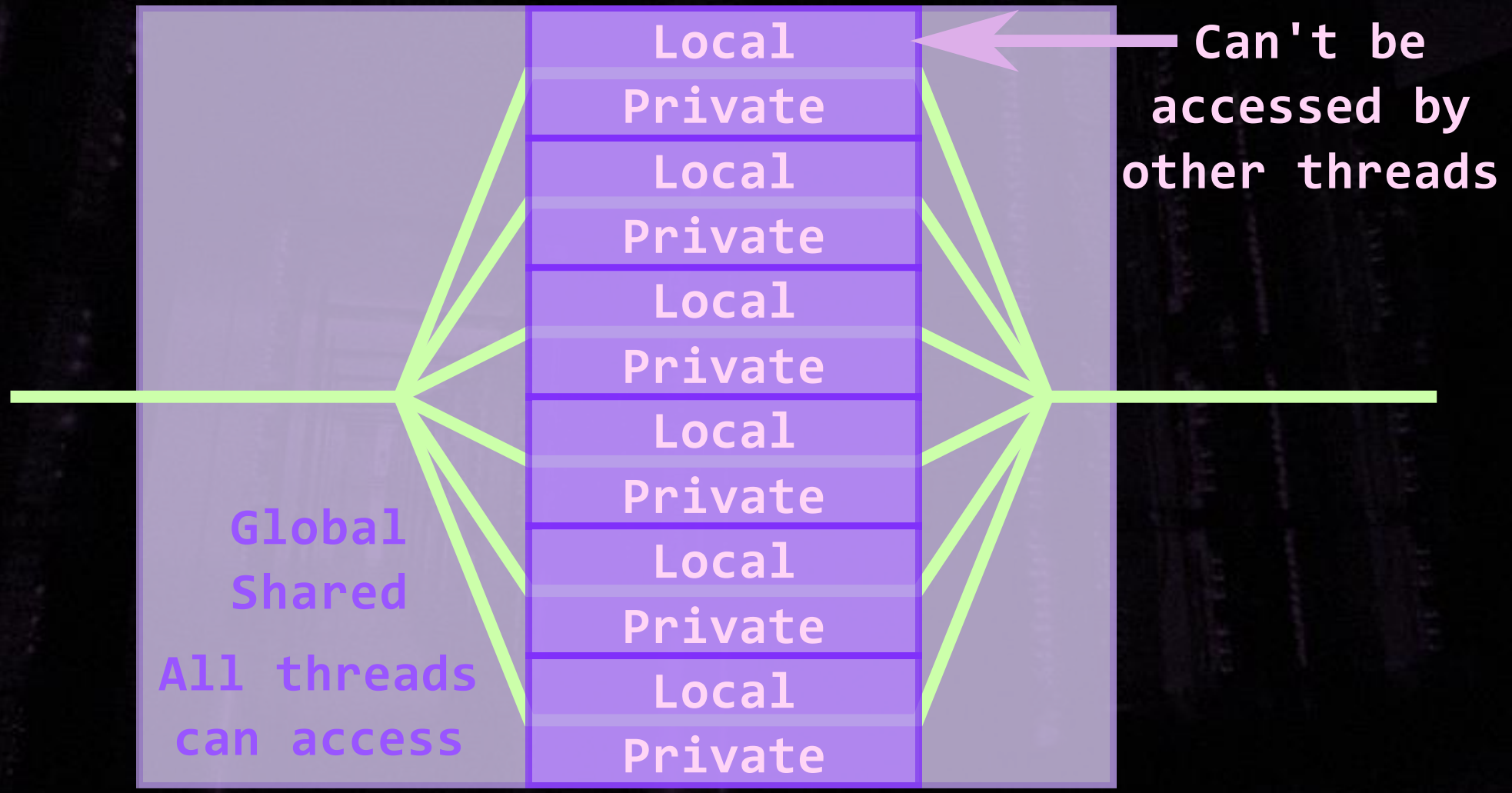
Synchronization:

- All threads wait at the end of the region

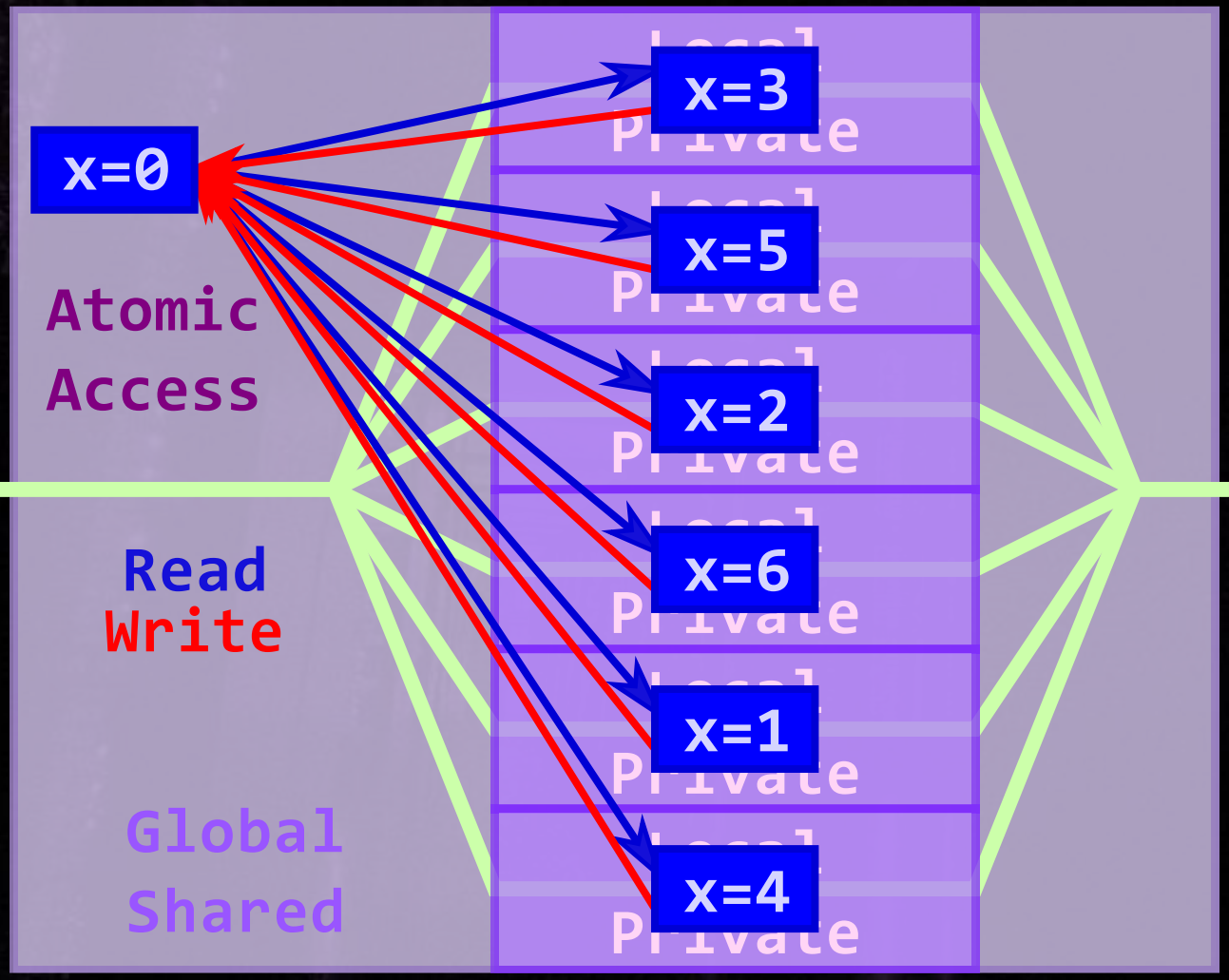
```
1 #pragma omp parallel
2 {
3   printf("Hi from %d\n", omp_get_thread_num());
4 #pragma omp master
5   {
6     printf("This only once\n");
7   }
```

Note:
Master was deprecated in favor of primary thread affinity. However, it is still widely used by applications

Memory: Private vs Shared



Memory Construct: Atomic



Ordered access NOT guaranteed

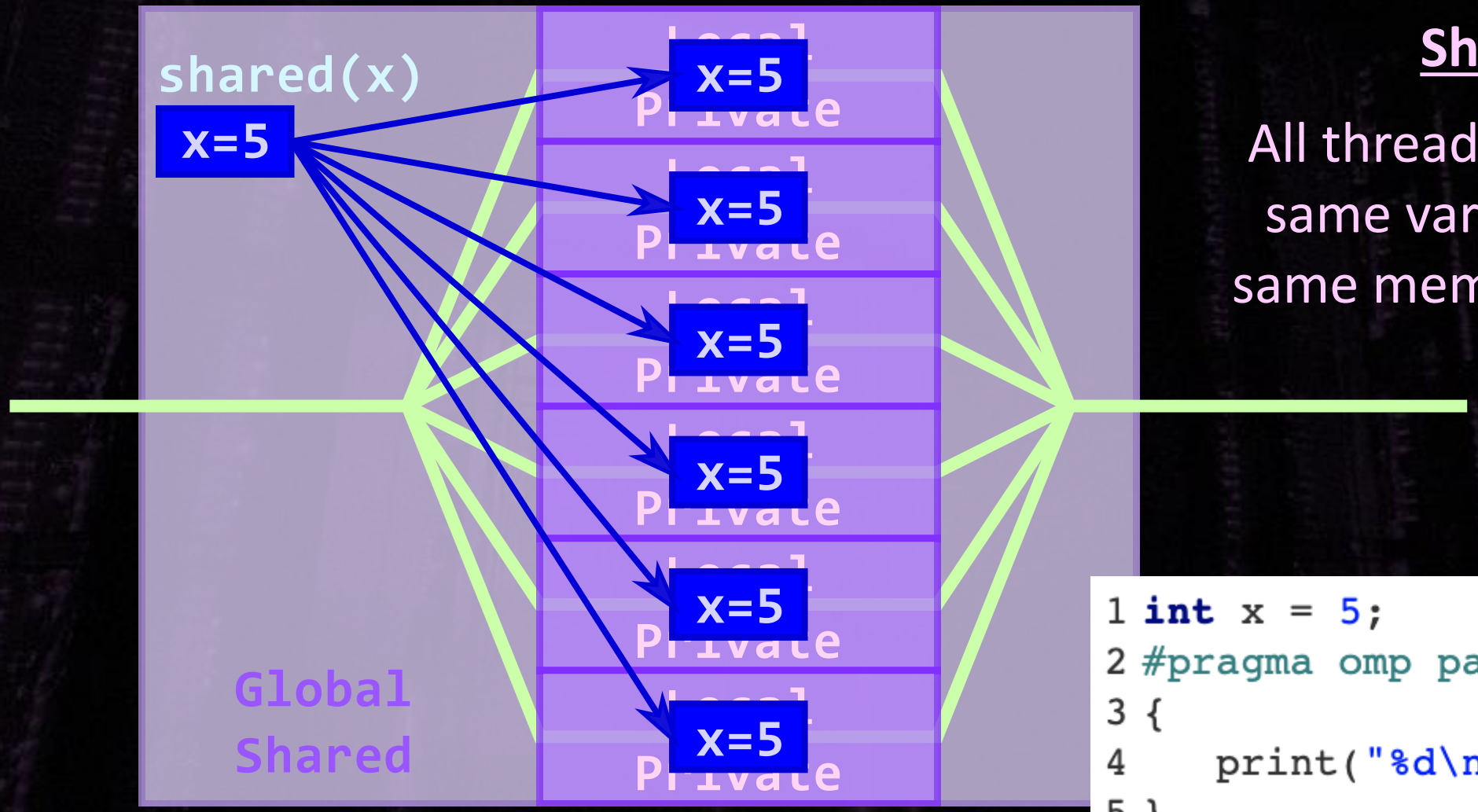
Atomic

All threads access the same variable in the same memory location.

Only one read/write operation at the time.

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

Memory Clauses: Shared

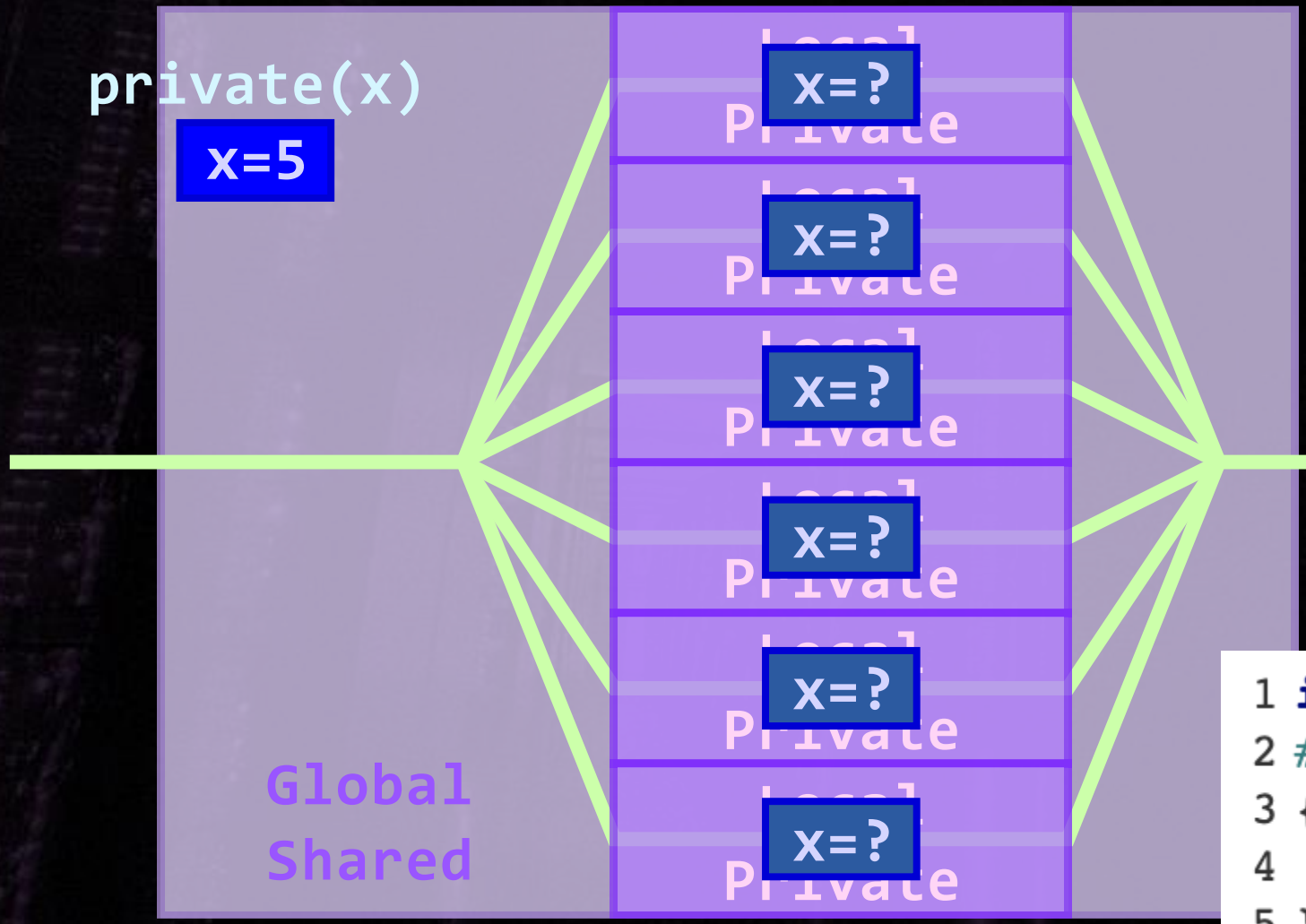


Shared

All threads access the same variable in the same memory location

```
1 int x = 5;
2 #pragma omp parallel shared(x)
3 {
4     print("%d\n", x);
5 }
```

Memory Clauses: Private



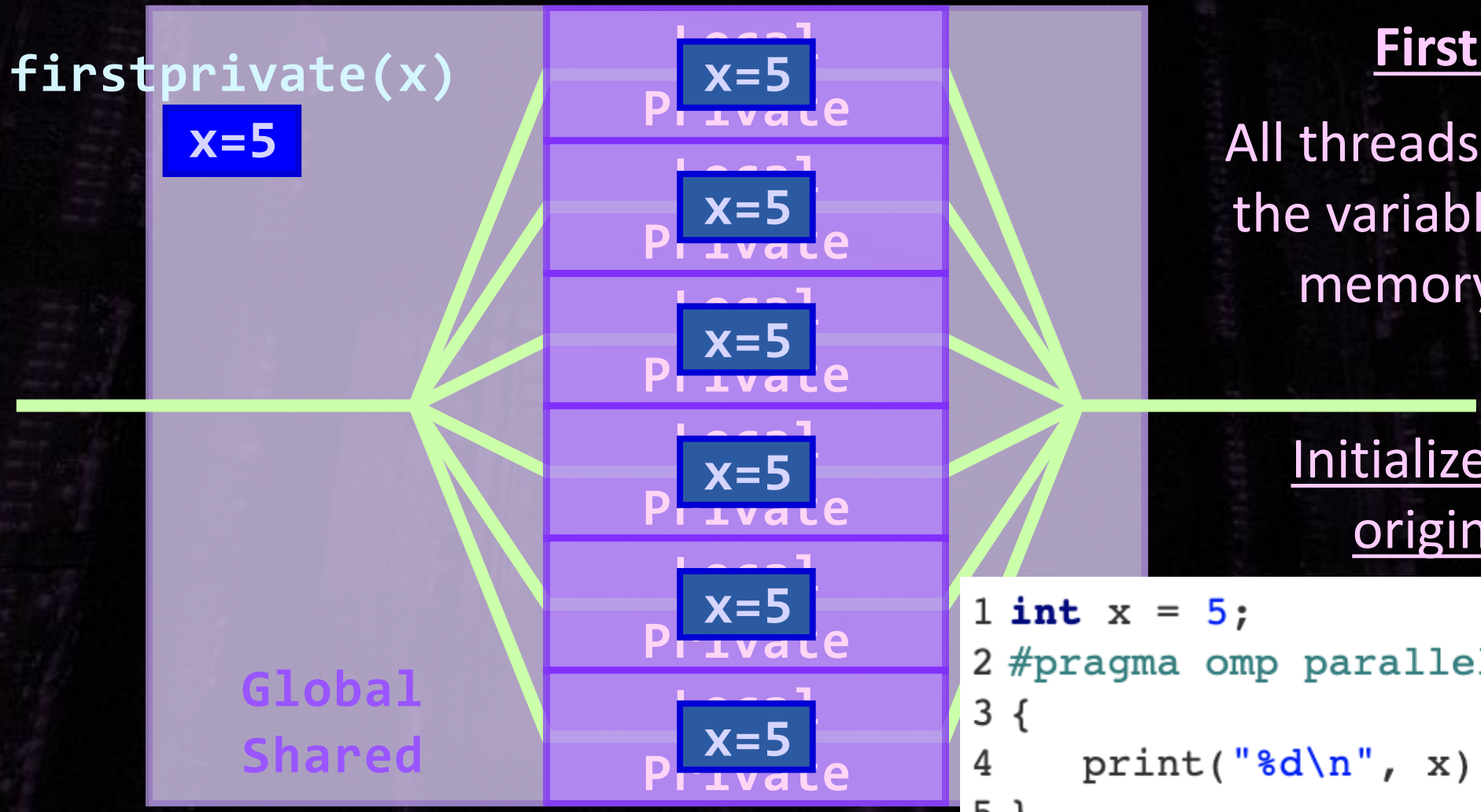
Private

All threads get a copy of the variable in different memory locations

Initialization value unknown

```
1 int x = 5;
2 #pragma omp parallel private(x)
3 {
4     print("%d\n", x);
5 }
```

Memory Clauses: Firstprivate



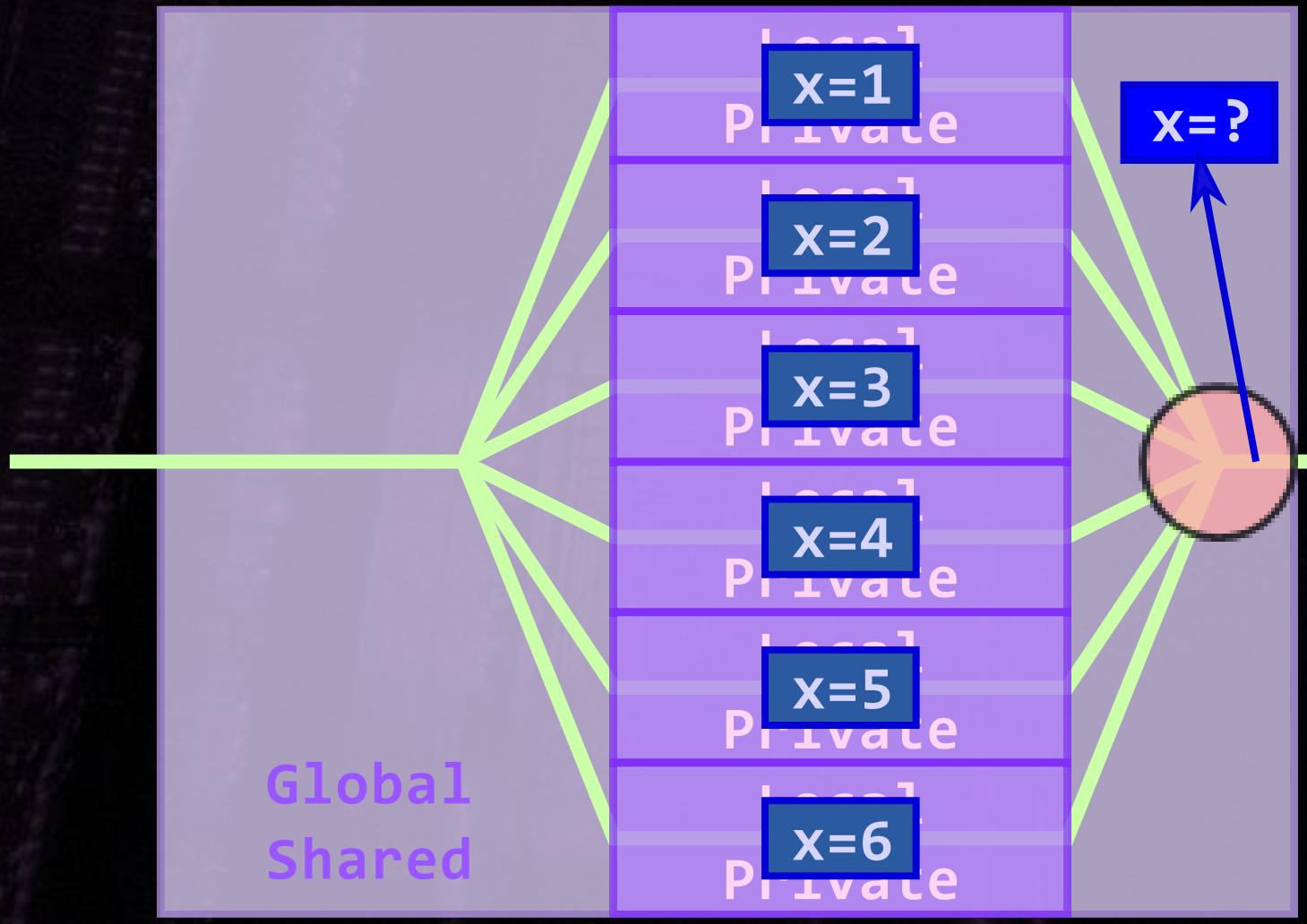
Firstprivate

All threads get a copy of the variable in different memory locations

Initialized with the original value

```
1 int x = 5;
2 #pragma omp parallel firstprivate(x)
3 {
4     print("%d\n", x);
5 }
```

Memory Clauses: What happens at the end of the parallel region?

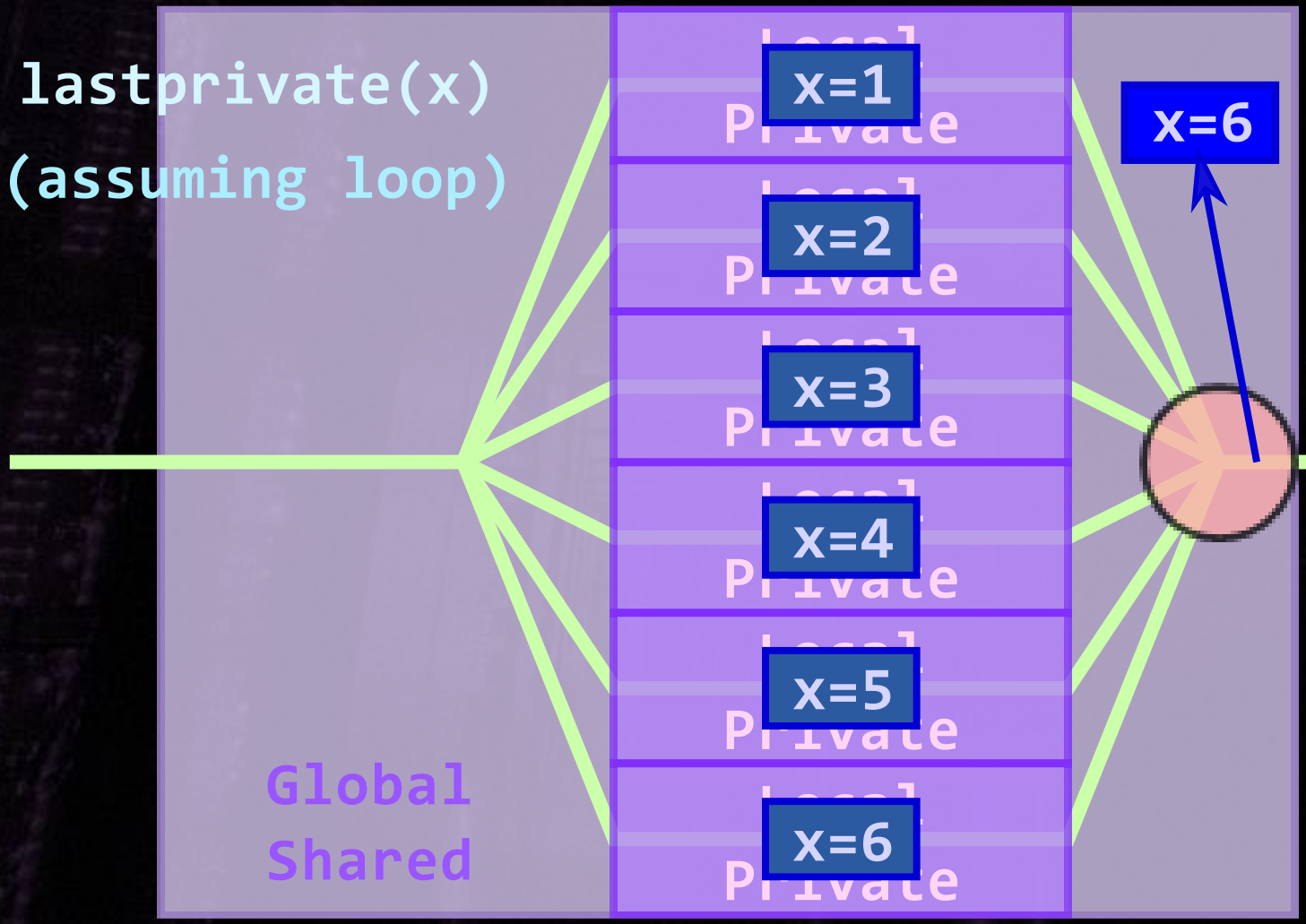


OpenMP provides a **relaxed-consistency shared-memory model**

Is the programmer's responsibility to define the behavior.

Ensure through **synchronization** that the lifetime of the variable does not end before completion of the **explicit** task region **sharing** it. Any other access by one task to the private variables of another task results in **unspecified behavior.**

Memory Clauses: Lastprivate

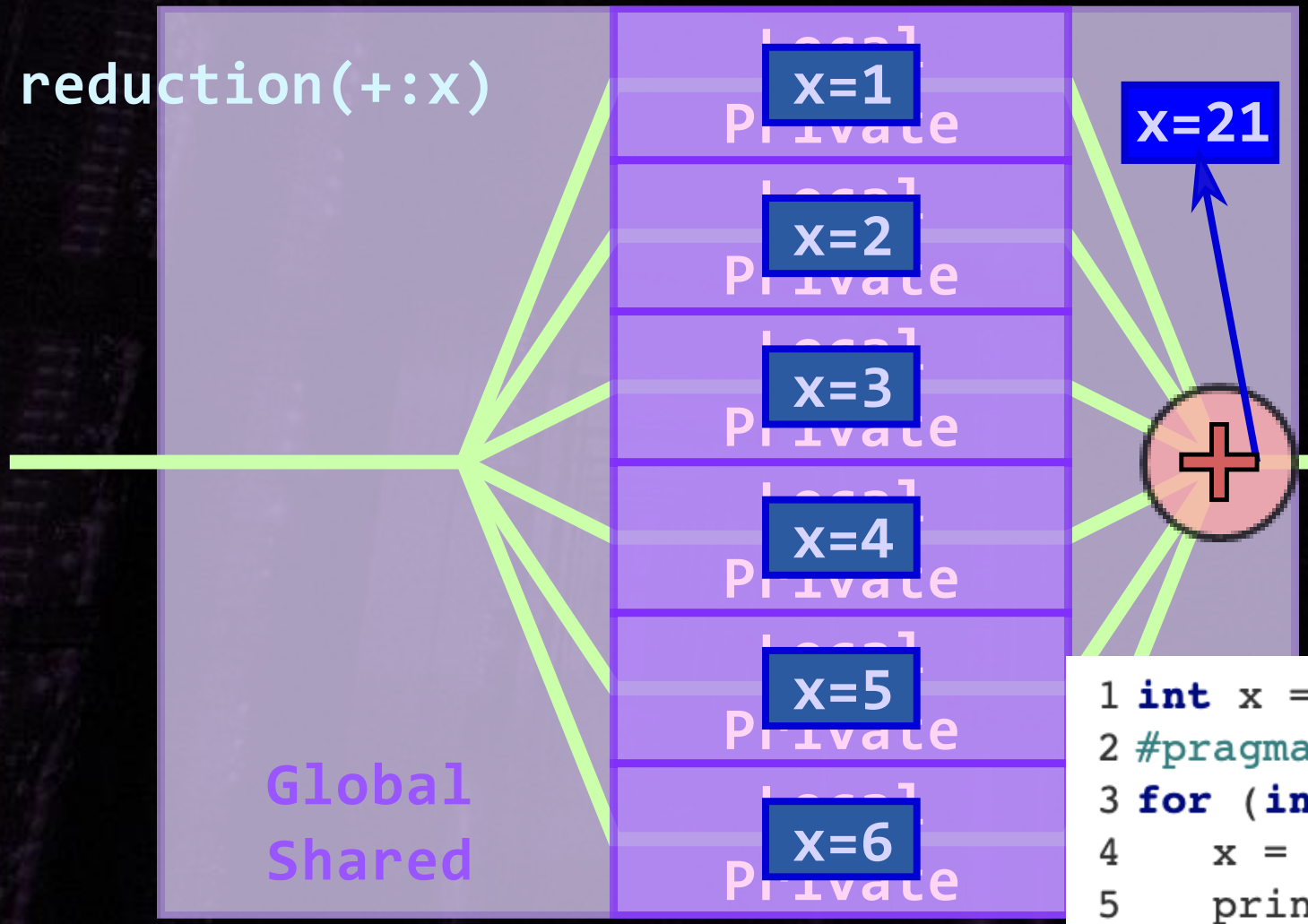


Lastprivate

All threads get a copy of the variable in different memory locations

Final value corresponds to last iteration's value

Memory Clauses: Reduction



Reduction

All threads get a copy of the variable in different memory locations

Final value corresponds to last iteration's value

```
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 }
```

Directives

C/C++ Macros:

#pragma omp …

_pragma(“omp …”)

C++:

[[omp :: directive(…)]] | [[using omp :: directive(…)]]

Fortran:

!$omp | c$omp | *$omp | !$omx | c$omx | *$omx

An OpenMP executable directive applies to the succeeding structured block or an OpenMP construct.

A structured-block is a single statement or a compound statement with a single entry at the top and a single exit at the bottom.

OpenMP directives tell the compiler what to do

Directives

Directive:

- A base language mechanism to specify OpenMP program behavior

Clause:

- An optional modifier that changes the directive's behavior

Construct:

- An OpenMP executable Directive