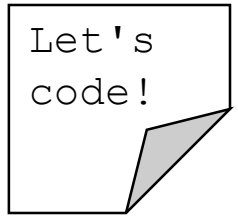
Data Sharing in OpenMP

Paolo Burgio
paolo.burgio@unimore.it

Outline

- Expressing parallelism
  - Understanding parallel threads
- Memory Data management
  - Data clauses
- Synchronization
  - Barriers, locks, critical sections
- Work partitioning
  - Loops, sections, single work, tasks…
- Execution devices
  - Target

Exercise

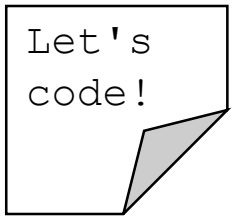


- Declare and initialize a variable outside the parallel region
- Spawn a team of parallel Threads
  - Each printing the value of the variable
- What do you see?

shared variables

- The variable is `shared` among the parallel threads
  - If one thread modifies it, then all threads see the new value

- Let's see this!
  - Let (only) Thread 0 modify the value of the variable



- What's happening?
  - (probably|might be that) Thread 0 modifies the value after the other threads read it
  - The more thread you have, the more probably you see this…

As opposite to… `private` variables

- Threads might wants to own a private copy of a datum
  - Other threads cannot modify it
- Two ways
  - They can declare it inside the parallel region
  - Or, they can use *data sharing attribute clauses*
- `private | firstprivate`
  - Create a storage for the specified datum (variable or param) in each threads' stack

Data sharing clauses in parregs

```
#pragma omp parallel [clause [[,]clause]...] new-line
  structured-block
```

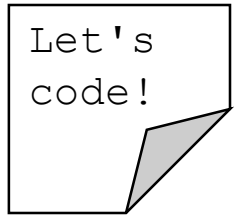
Where clauses can be:

```
if([parallel :] scalar-expression)
num_threads (integer-expression)
default(shared | none)
firstprivate (list)
private (list)
shared (list)
copyin (list)
reduction(reduction-identifier : list)
proc_bind(master | close | spread)
```

Initial value for `(first)private` data

- How is the private data initialized?
  - `firstprivate` initializes it with the value of the enclosing context
  - `private` does not initialize it / initializes it with 0

Exercise



- Declare and initialize a variable outside the parallel region
- Spawn a team of parallel Threads
  - Mark the variable as `private` using data sharing clause
  - Each printing the value of the variable
  - Let (only) Thread 0 modify the value of the variable
- What do you see?
  - Now, mark the variable as `firstprivate`

`shared` data-sharing clause

- All variables specified are shared among all threads
- Programmer is in charge of consistency!
  - OpenMP philosophy..

Multiple variables in a single clause

- Do not need to repeat the clause always
  - If you don't want..
- Separated by commas

```
int a = 11, b = 1, c;

#pragma omp parallel num_threads(16) \
                     private(a, b)
private(c)
{
  …
```

`private` vs. parreg-local variables

- Find the difference between…

```
#pragma omp parallel num_threads(4)
{
    int a = ...
}
```

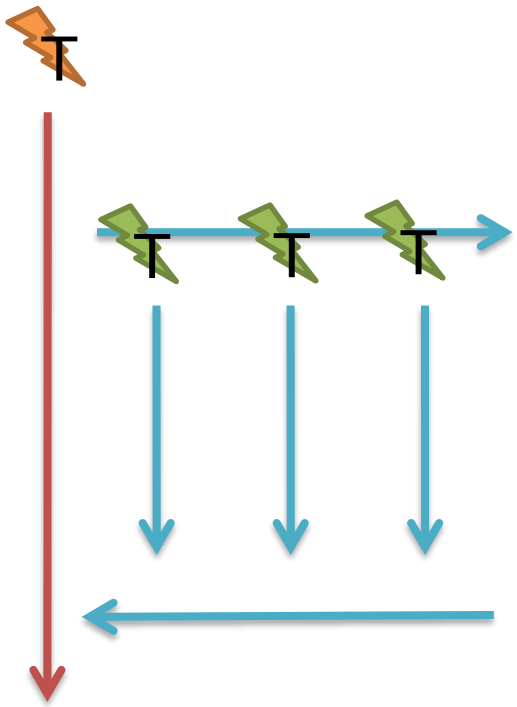
```
int a = ...;

#pragma omp parallel private(a) \
                     num_threads(4)
{
    a = ...
}
```

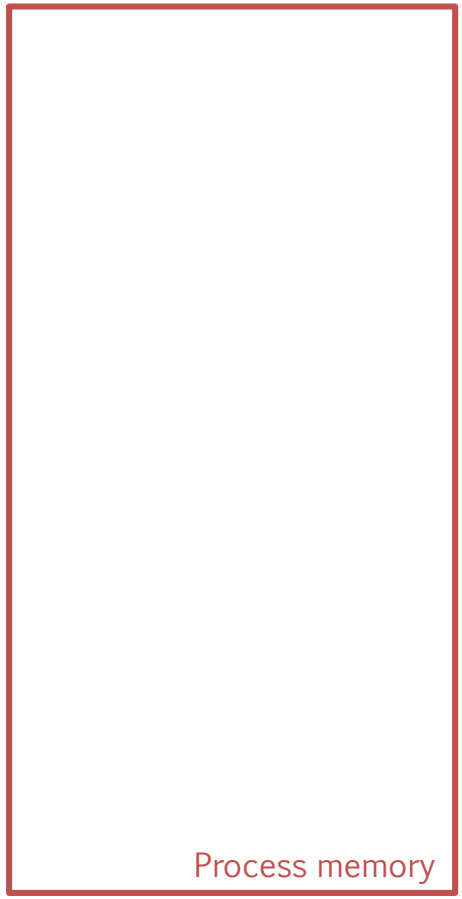
- *"A new storage is created as we enter the region, and destroyed after"*
- On the right (`private`)
  - There is also a storage that exists before and after parreg

Variables and memory (1)

› "The traditional way"

```
#pragma omp parallel num_threads(4)
{
    int a = ...



}
```

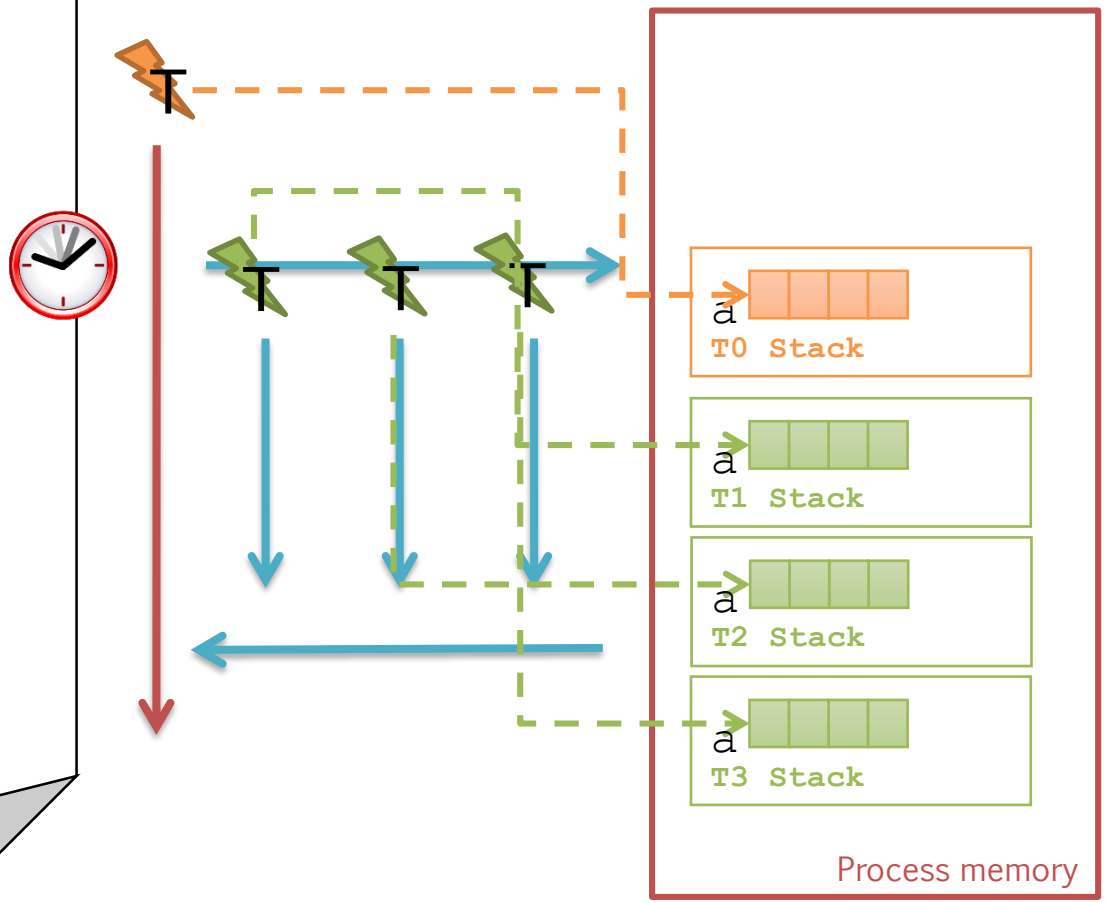




Variables and memory (1)

› "The traditional way"

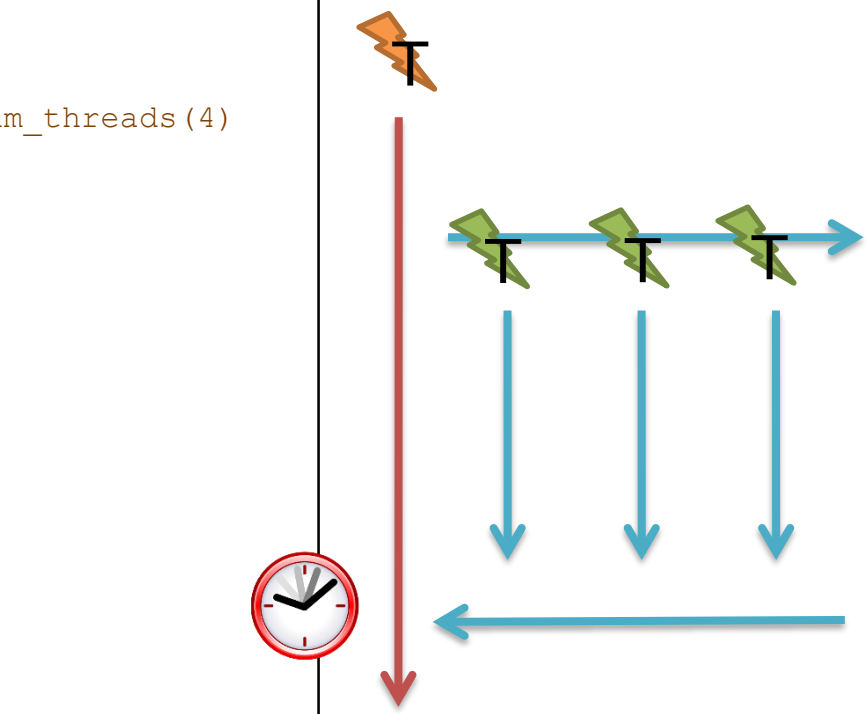
```
#pragma omp parallel num_threads(4)
{
    int a = ...



}
```

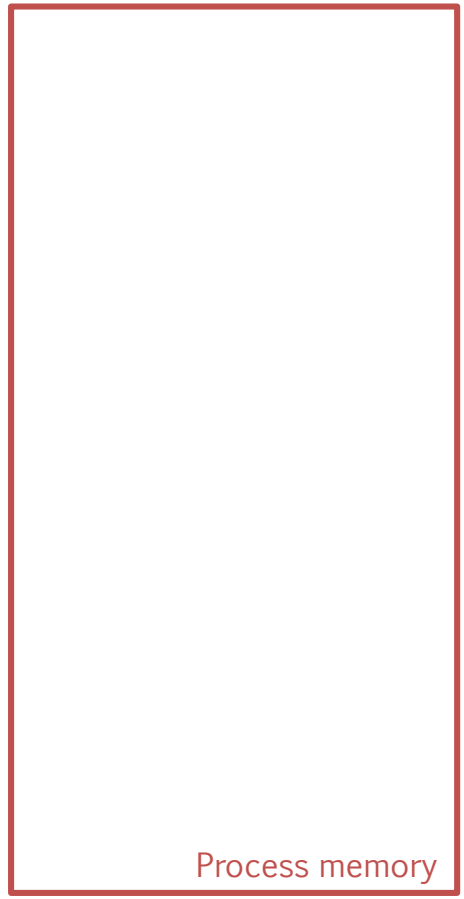


Variables and memory (1)

› "The traditional way"

```
#pragma omp parallel num_threads(4)
{
    int a = ...
}
```

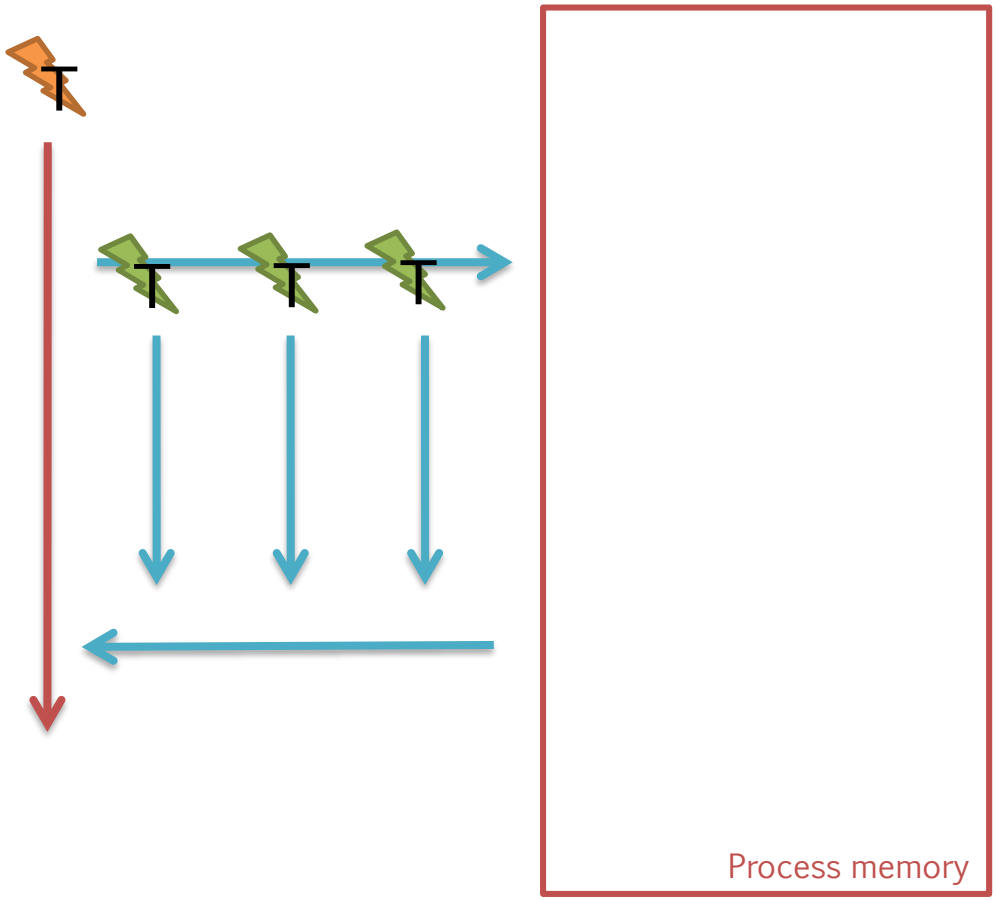




Variables and memory (2)

› Create a new storage for the variables, local to threads

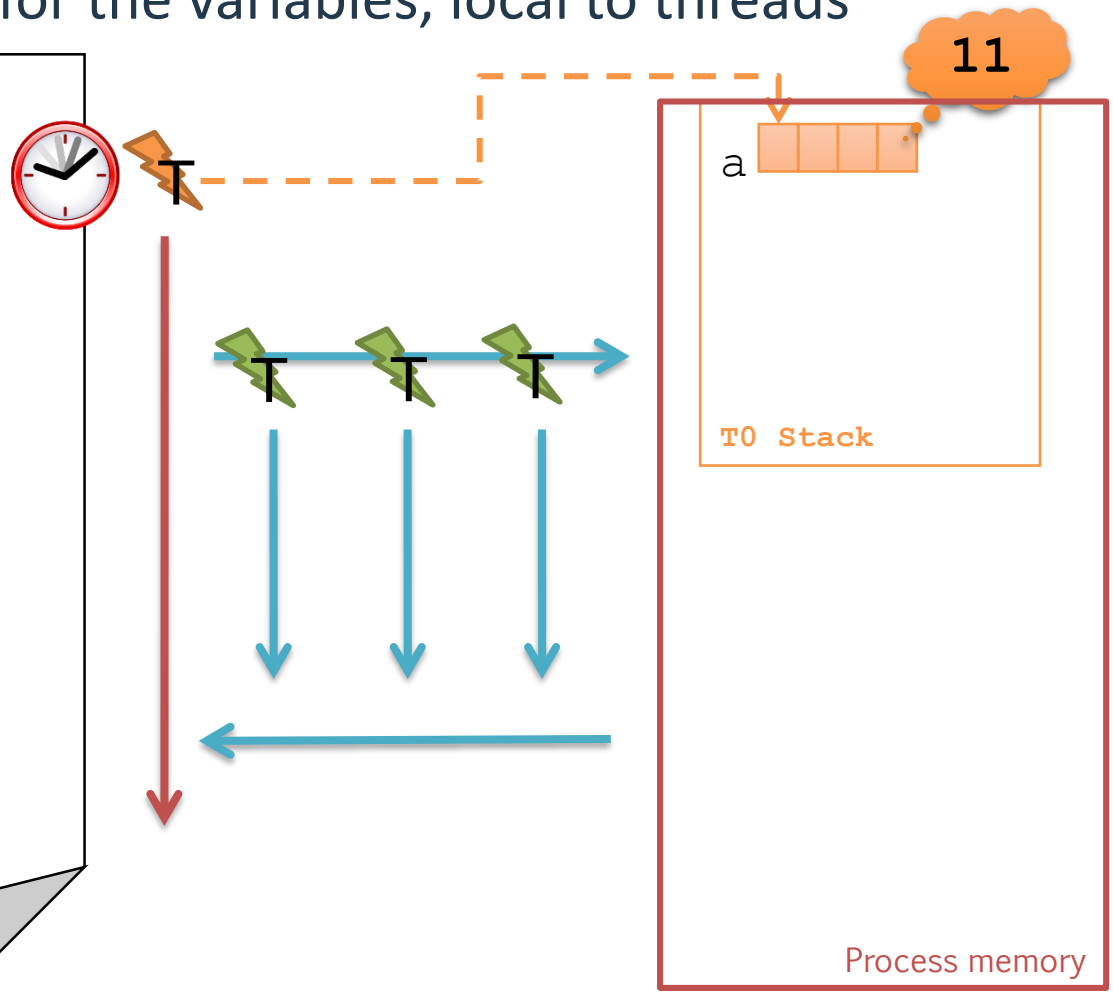
```
int a = 11;


#pragma omp parallel private(a) \
                    num_threads(4)
{

    a = ...




}
```



Variables and memory (2)

› Create a new storage for the variables, local to threads

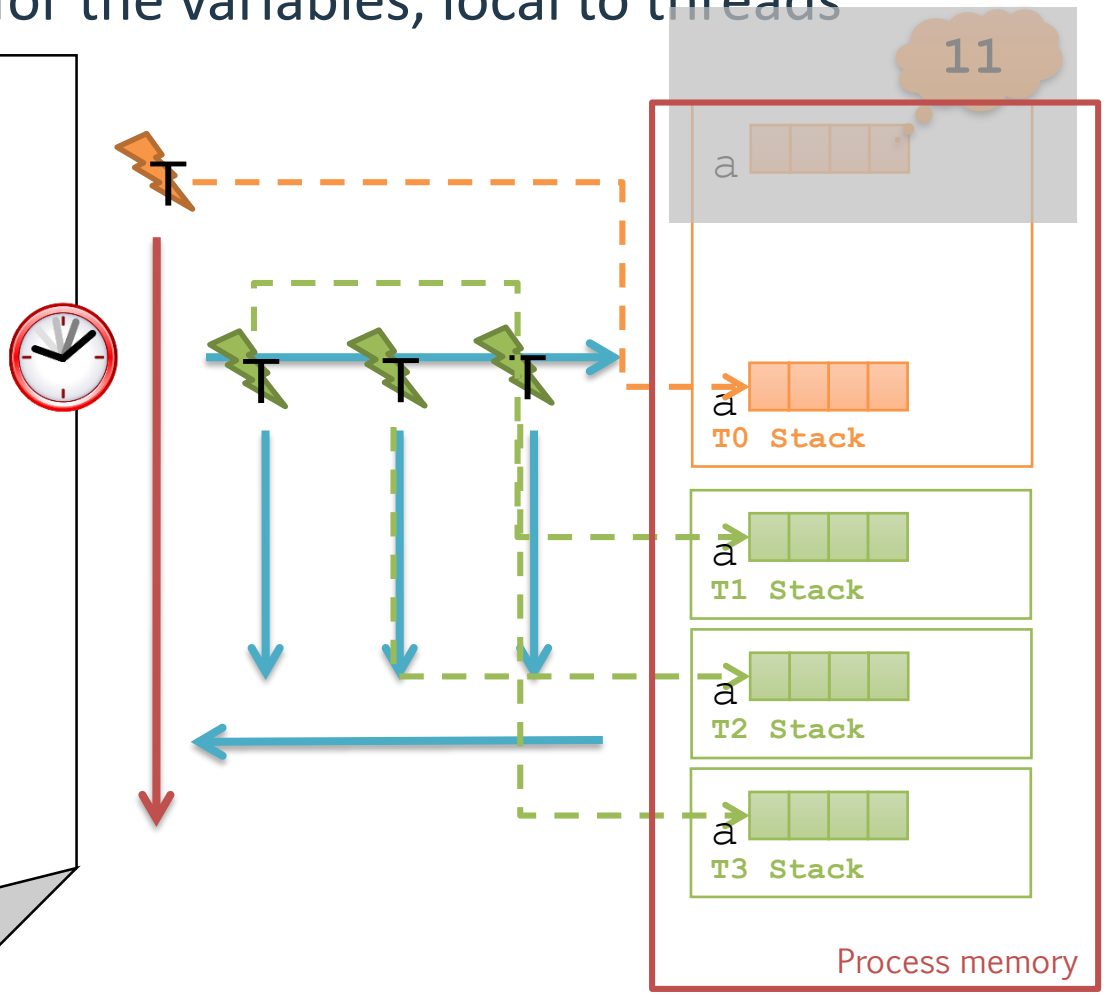
```
int a = 11;


#pragma omp parallel private(a) \
                    num_threads(4)
{

    a = ...




}
```



Variables and memory (2)

› Create a new storage for the variables, local to threads

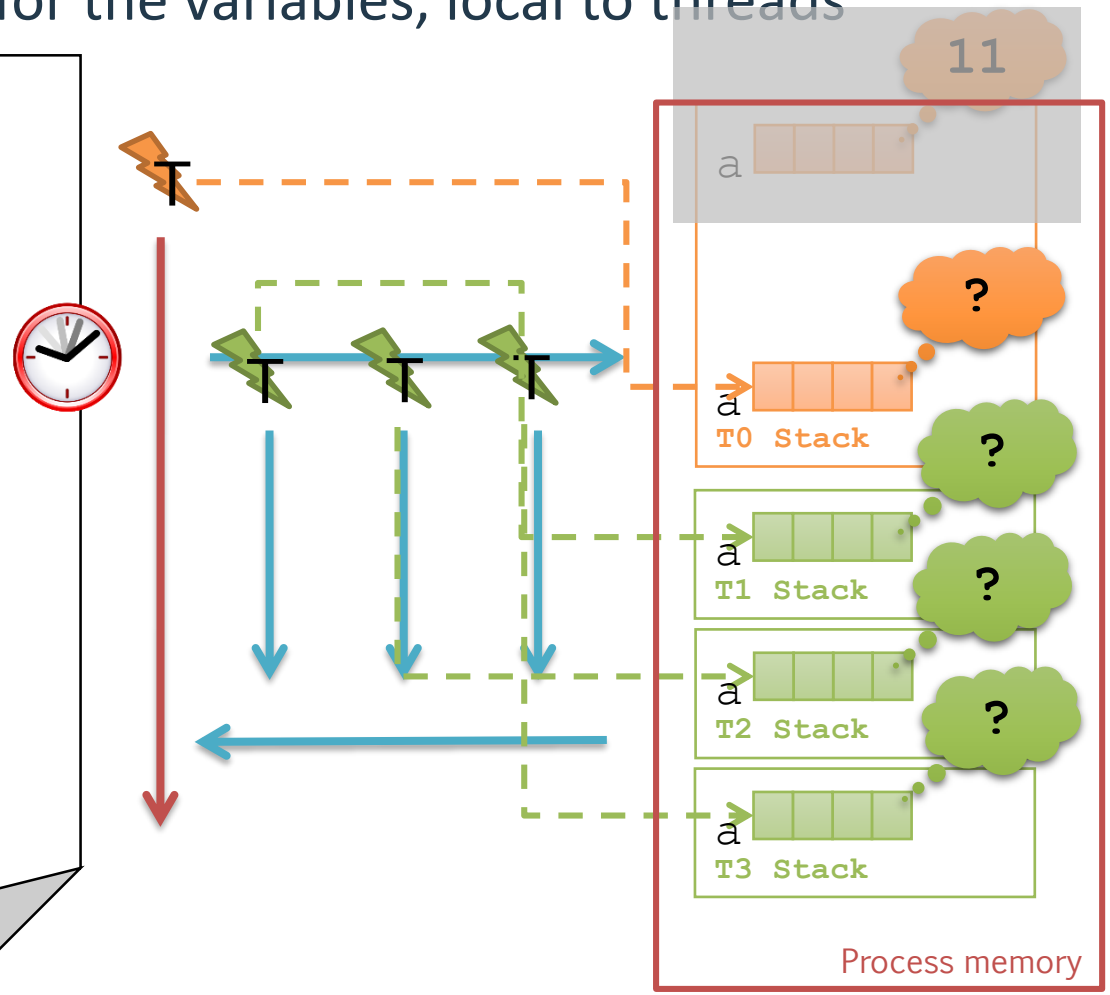
```
int a = 11;


#pragma omp parallel private(a) \
                     num_threads(4)
{

    a = ...




}
```



Variables and memory (2)

› Create a new storage for the variables, local to threads

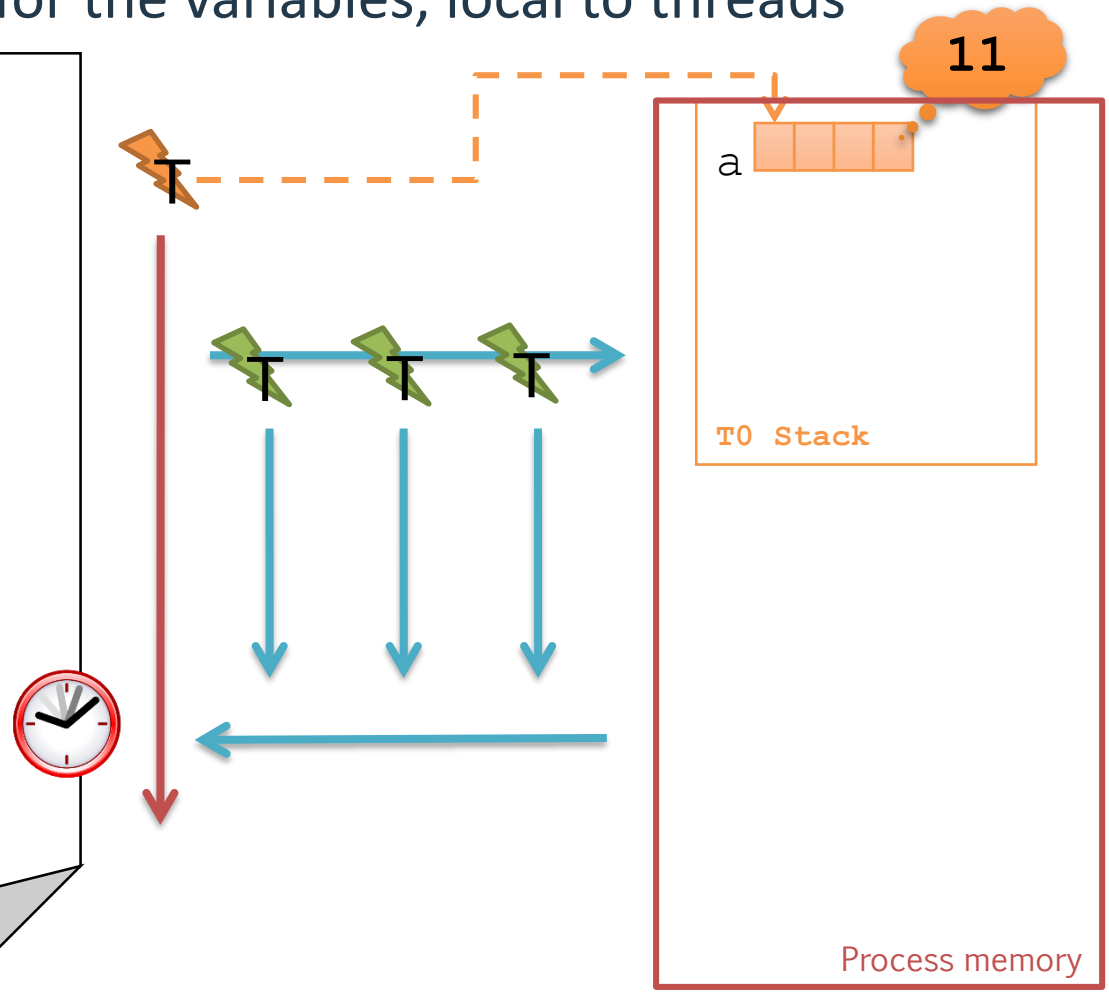
```
int a = 11;


#pragma omp parallel private(a) \
                     num_threads(4)
{

    a = ...




}
```



Variables and memory (2)

› Create a new storage for the variables, local to threads

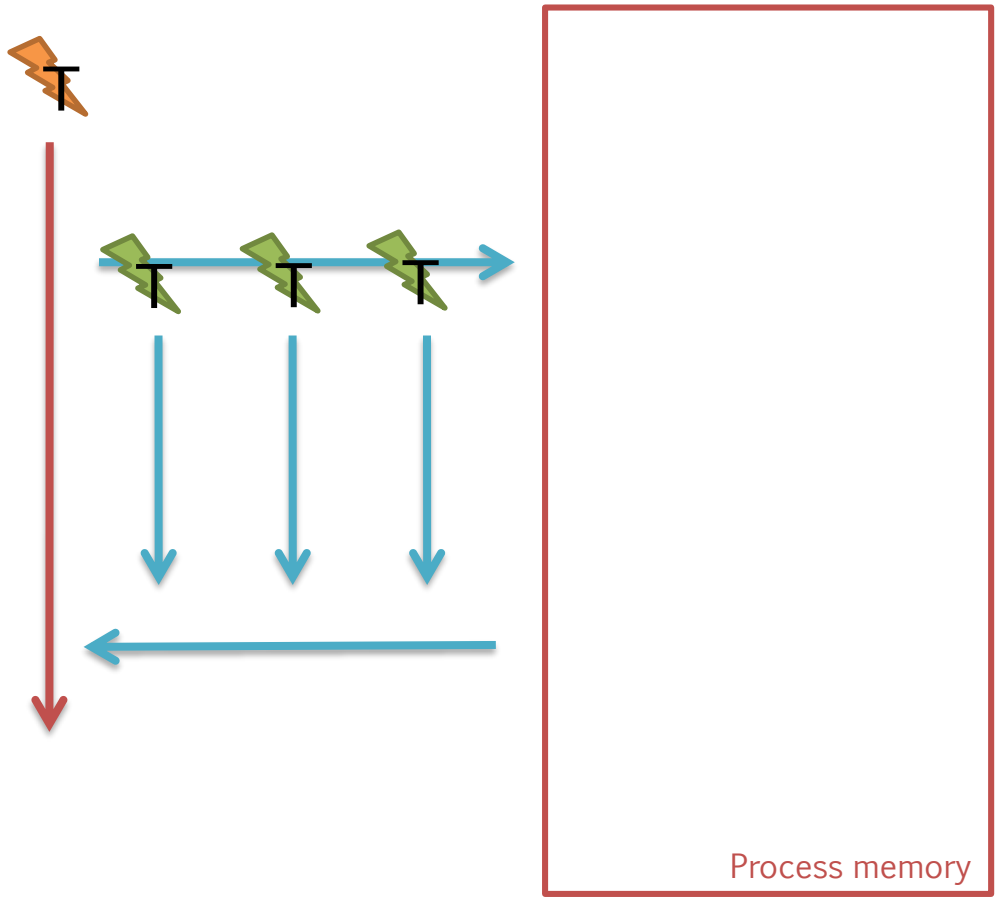
```
int a = 11;


#pragma omp parallel private(a) \
                    num_threads(4)
{

    a = ...




}
```



Variables and memory (3)

› Create a new storage for the variables, local to threads, and initialize

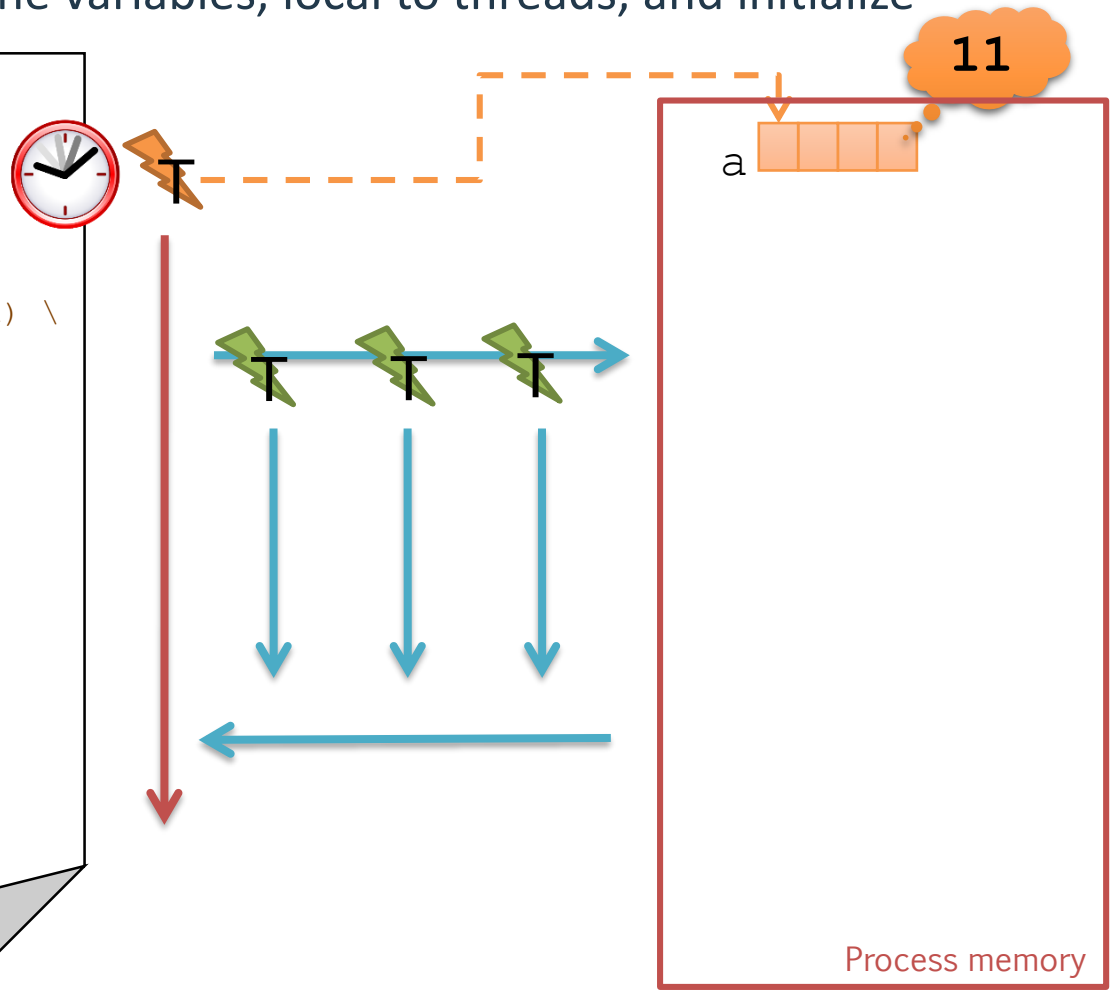
```
int a = 11;


#pragma omp parallel firstprivate(a) \
                     num_threads(4)
{

    a = ...




}
```



Variables and memory (3)

› Create a new storage for the variables, local to threads, and initialize

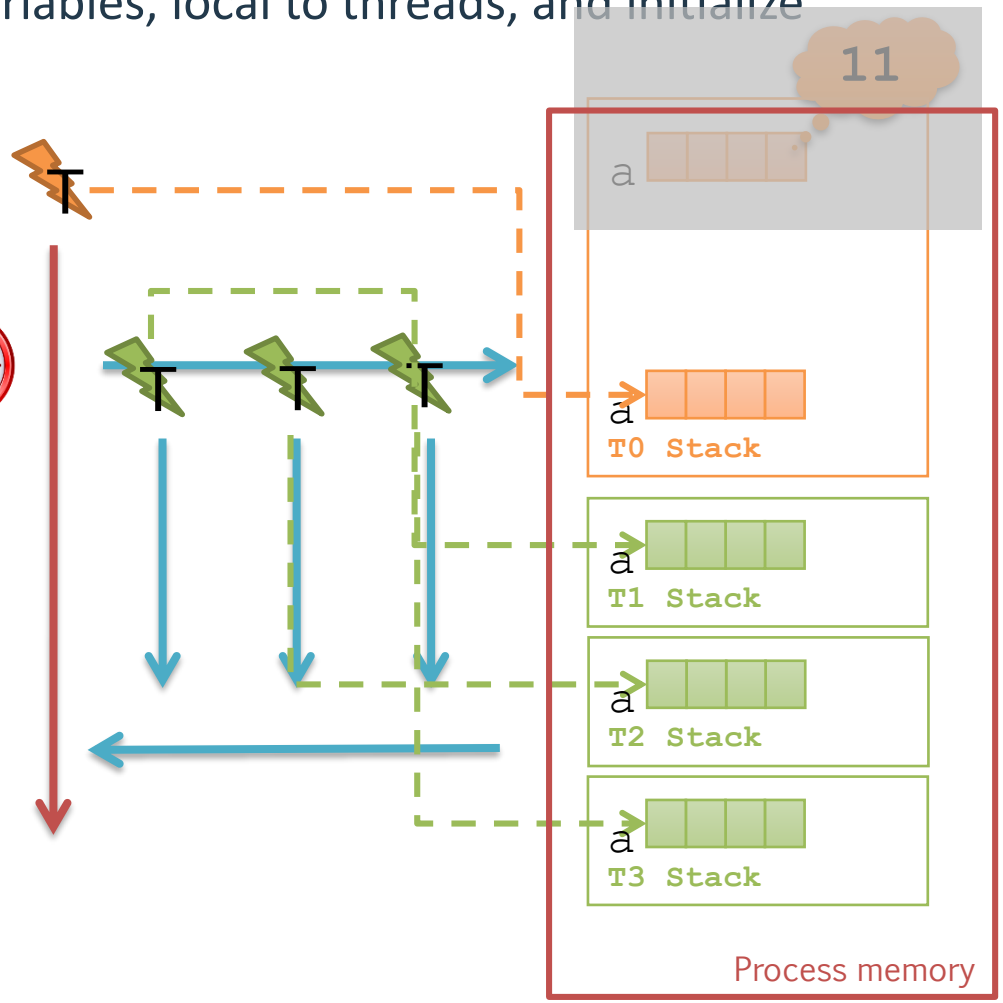
```
int a = 11;


#pragma omp parallel firstprivate(a) \
                     num_threads(4)
{

    a = ...




}
```



Variables and memory (3)

› Create a new storage for the variables, local to threads, and initialize

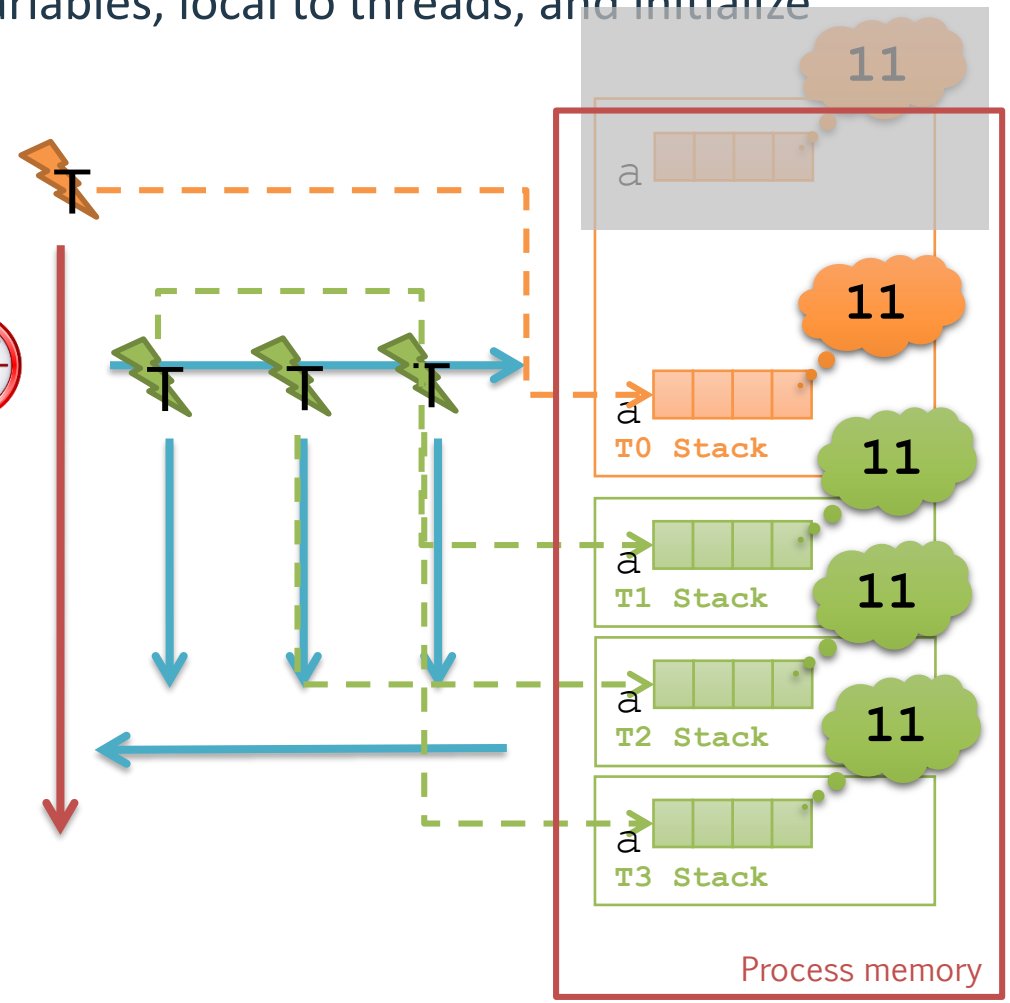
```
int a = 11;


#pragma omp parallel firstprivate(a)
                     num_threads(4)
{

    a = ...




}
```



Variables and memory (3)

› Create a new storage for the variables, local to threads, and initialize

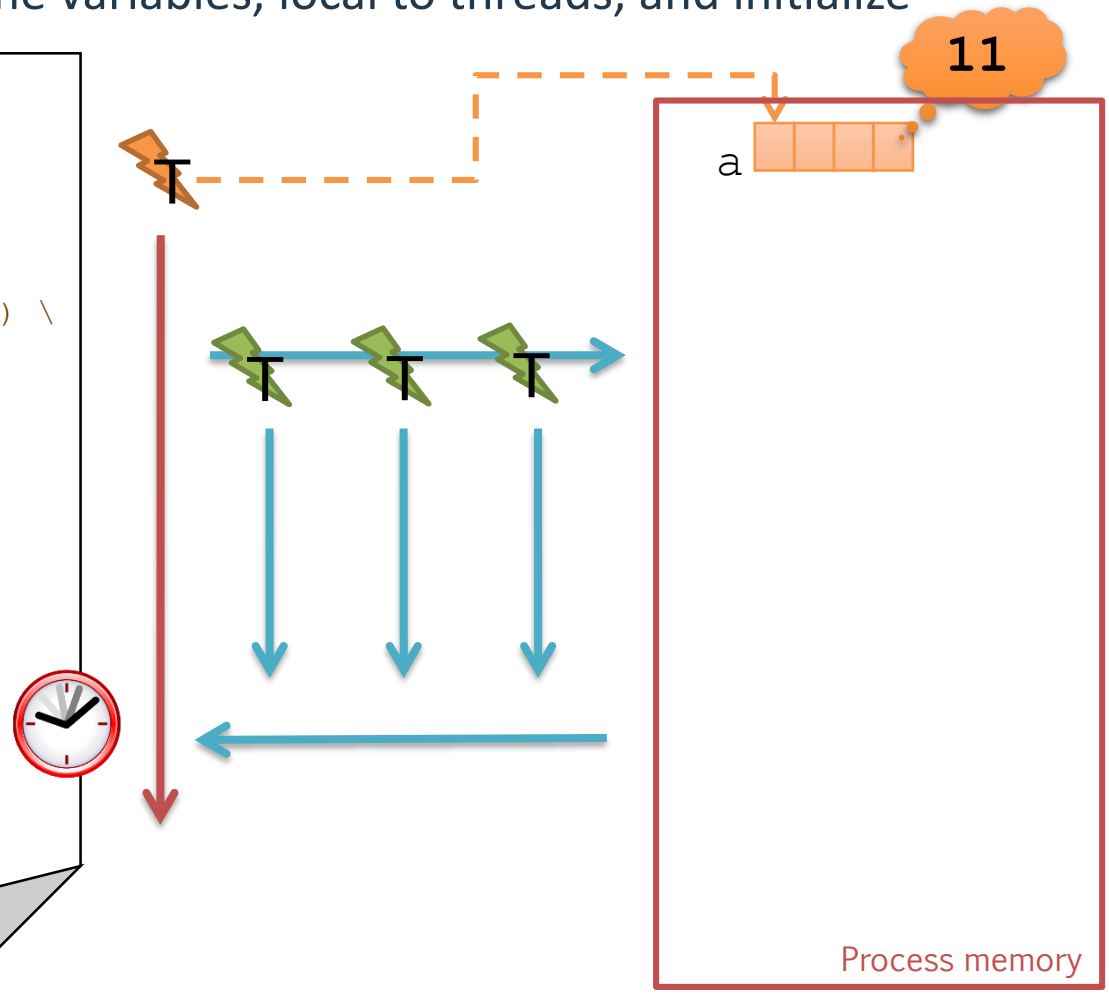
```
int a = 11;


#pragma omp parallel firstprivate(a)
                     num_threads(4)
{

    a = ...




}
```



Variables and memory (3)

› Create a new storage for the variables, local to threads, and initialize

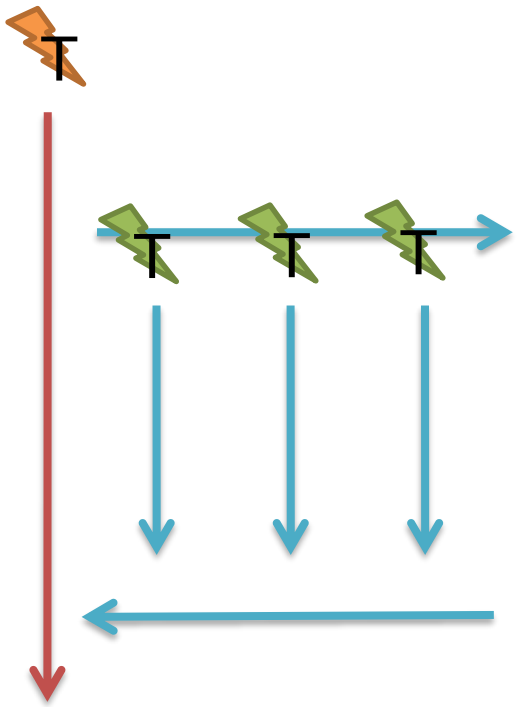
```
int a = 11;


#pragma omp parallel firstprivate(a) \
                     num_threads(4)
{

    a = ...




}
```

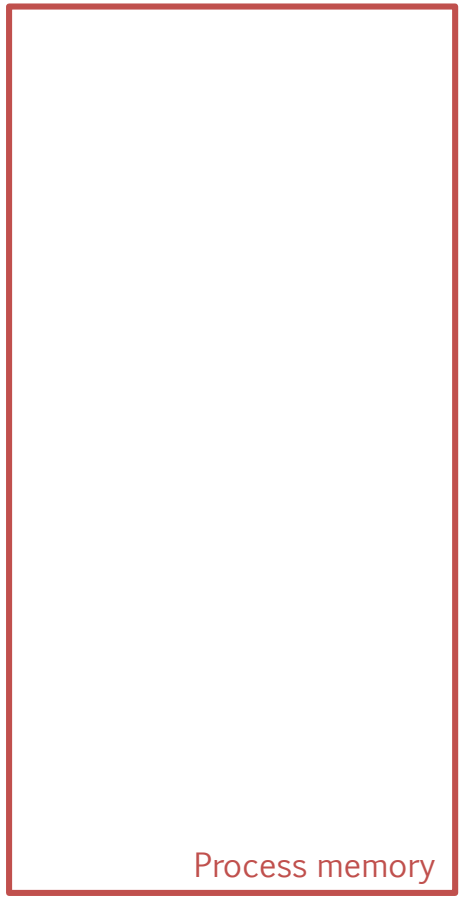


Variables and memory (4)

› Every slave Thread refers to master's storage

```
int a = 11;


#pragma omp parallel shared(a) \
                    num_threads(4)
{

    a = ...




}
```

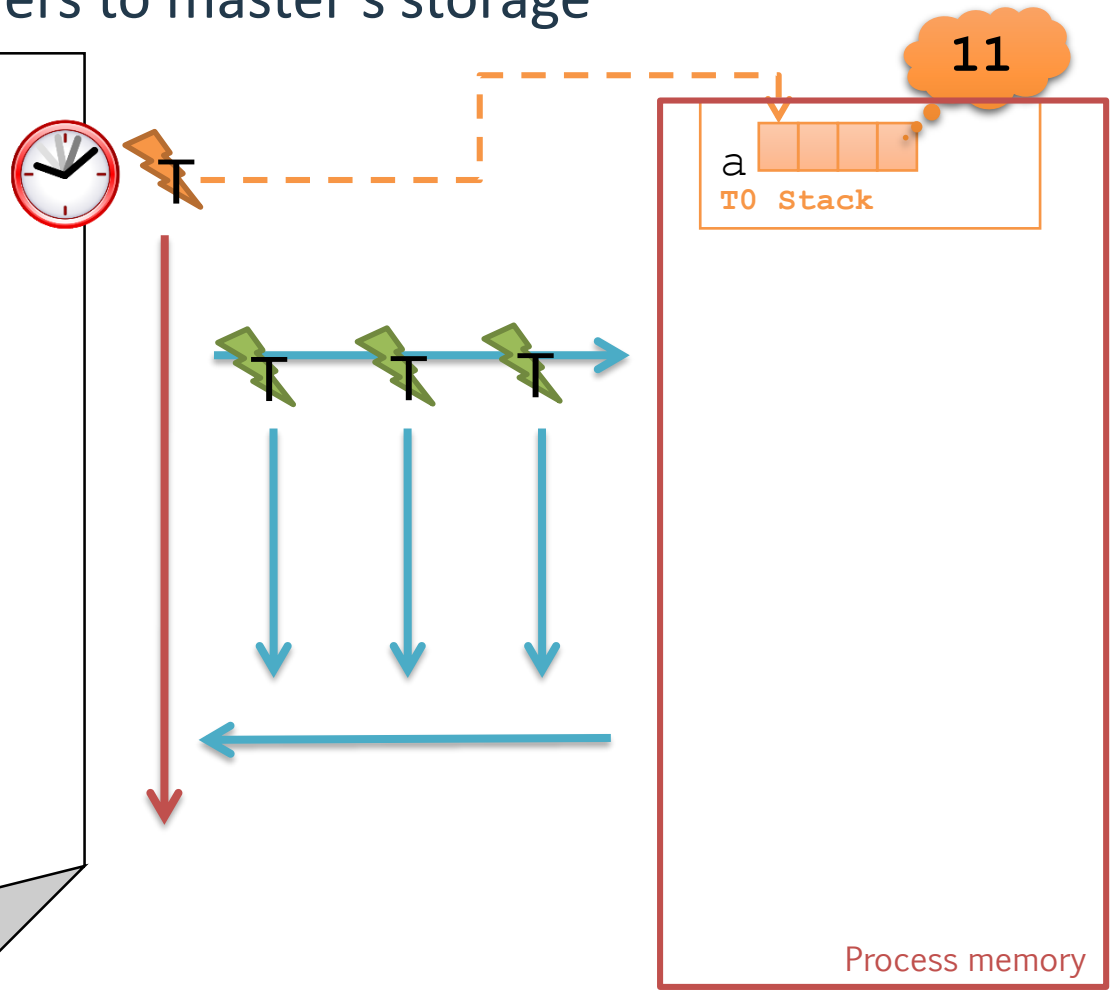




Variables and memory (4)

› Every slave Thread refers to master's storage

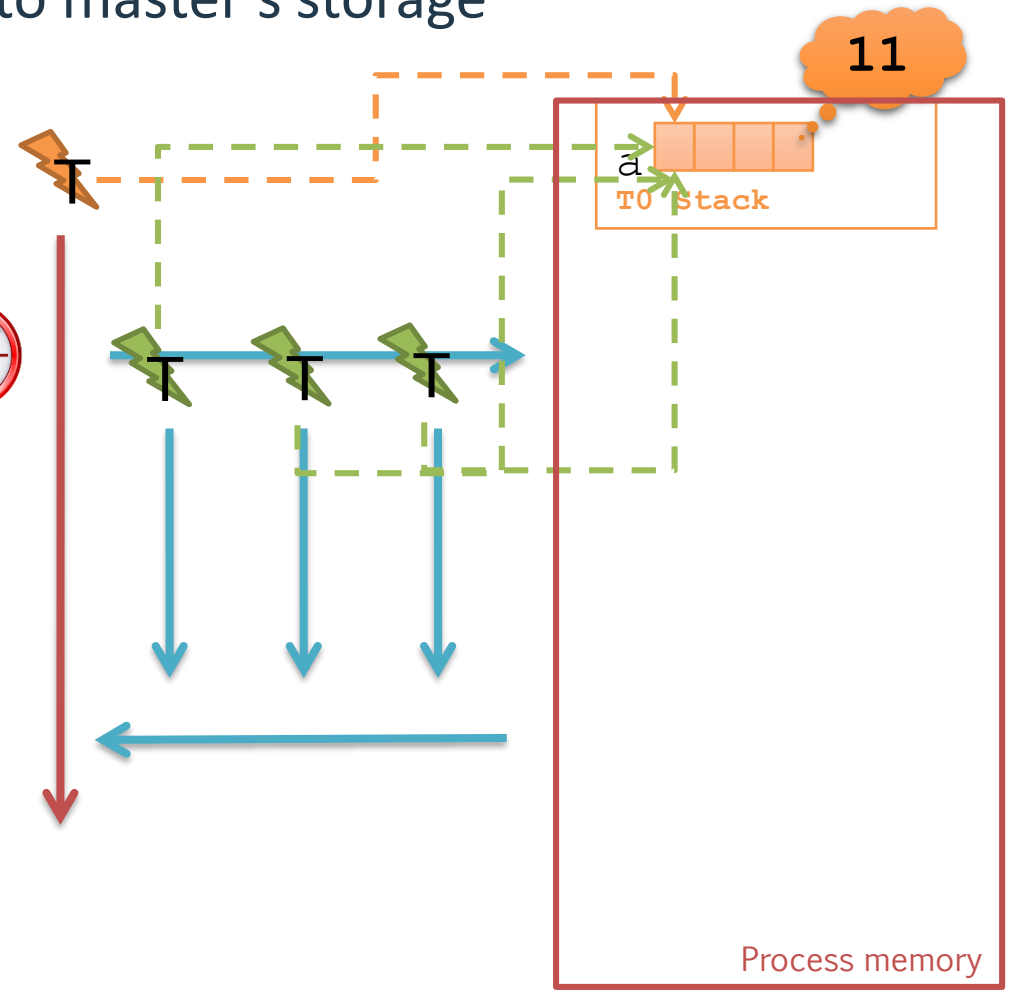
```
int a = 11;


#pragma omp parallel shared(a) \
                    num_threads(4)
{

    a = ...




}
```



Variables and memory (4)

› Every slave Thread refers to master's storage

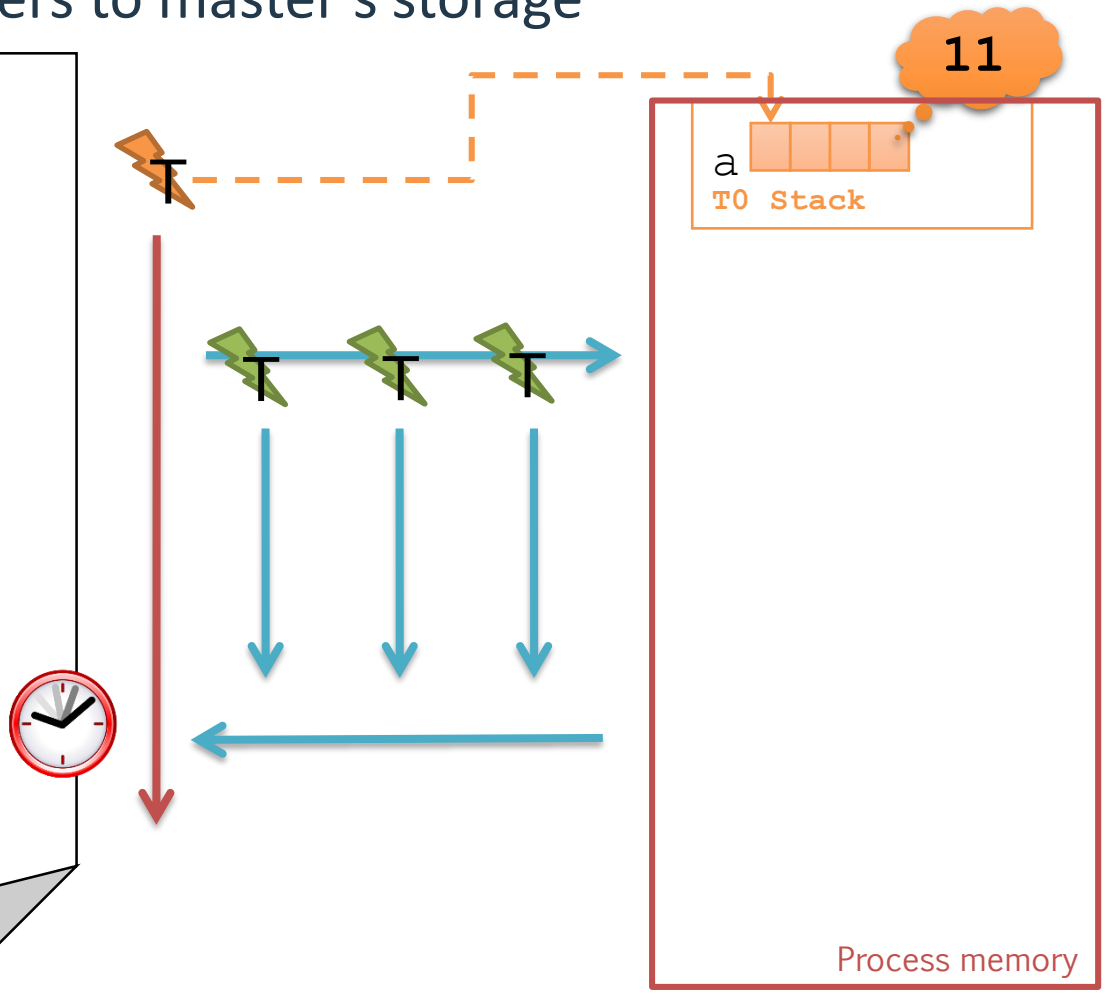
```
int a = 11;


#pragma omp parallel shared(a) \
                    num_threads(4)
{

    a = ...




}
```



Variables and memory (4)

› Every slave Thread refers to master's storage

```
int a = 11;


#pragma omp parallel shared(a) \
                    num_threads(4)
{

    a = ...




}
```



reduction clause in parregs

```
#pragma omp parallel [clause [[,]clause]...] new-line
  structured-block
```

Where clauses can be:

```
if([parallel :] scalar-expression)
num_threads (integer-expression)
default(shared | none)
firstprivate (list)
private (list)
shared (list)
copyin (list)
reduction(reduction-identifier : list)
proc_bind(master | close | spread)
```

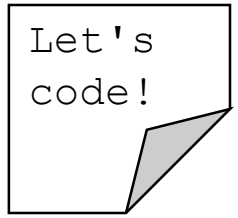
Reduction

OpenMP specifications

The reduction clause can be used to perform some forms of recurrence calculations (involving mathematically associative and commutative operators) in parallel. For parallel [...], a private copy of each list item is created, one for each implicit task, as if the private clause had been used. [...] The private copy is then initialized as specified above. At the end of the region for which the reduction clause was specified, the original list item is updated by combining its original value with the final value of each of the private copies, using the combiner of the specified reduction-identifier.

- In a nutshell
  - For each variable specified, create a `private` storage
  - At the end of the region, update master thread's value according to `reduction-identifier`
  - The variable must be qualified for that operation

Exercise



- Declare and initialize a variable outside the parallel region
  - `int a = 11`
- Spawn a team of parallel Threads
  - Mark the variable as `reduction(+:a)`
  - Increment variable `a`
  - Print the value of the variable before, inside, and after the parreg
- What do you see?
  - (at home) repeat with other reduction-identifiers

Reduction identifiers

+ - * & | ^ && || max min

- Mathematical/logical identifiers
  - Each has a default initializer, and a combiner
  - Minus (–) is more or less the same as plus (+)

OpenMP specifications

Identifier	Initializer	Combiner
+	omp_priv = 0	omp_out += omp_in
*	omp_priv = 1	omp_out *= omp_in
-	omp_priv = 0	omp_out += omp_in
&	omp_priv = 0	omp_out &= omp_in
|	omp_priv = 0	omp_out |= omp_in
^	omp_priv = 0	omp_out ^= omp_in
&&	omp_priv = 1	omp_out = omp_in && omp_out
||	omp_priv = 0	omp_out = omp_in || omp_out
max	omp_priv = *Least representable number in the reduction list item type*	omp_out = omp_in > omp_out ? omp_in : omp_out
min	omp_priv = *Largest representable number in the reduction list item type*	omp_out = omp_in < omp_out ? omp_in : omp_out

Data sharing clauses in parregs

```
#pragma omp parallel [clause [[,]clause]...] new-line
  structured-block
```

Where clauses can be:

```
if([parallel :] scalar-expression)
num_threads (integer-expression)
default(shared | none)
firstprivate (list)
private (list)
shared (list)
copyin (list)
reduction(reduction-identifier : list)
proc_bind(master | close | spread)
```

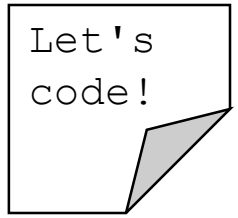
`default` data sharing clause

OpenMP specifications

The `default` clause explicitly determines the data-sharing attributes of variables that are referenced in a parallel, teams, or task generating construct and would otherwise be implicitly determined (see Section 2.15.1.1 on page 179).

- Can be
  - `shared`: all variables referenced in the construct that are not present in a data sharing clause are shared
  - `none`: each variable that is referenced in the construct, *and that does not have a predetermined data-sharing attribute*, must have its data-sharing attribute explicitly determined using a data-sharing clause
- (Yes, we can have predetermined attributes)
  - We won't see this

Exercise



- Declare and initialize a variable outside the parallel region
- Spawn a team of parallel Threads
  - Use the `default(none)` using data sharing clause
  - Do not use any other data sharing clause
  - Each thread prints the value of the variable
- What do you see?

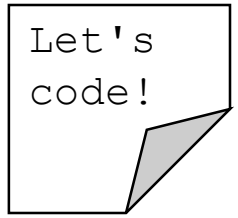
Watch out!

- We haven't seen everything..
  - Rules determining default sharing attributes are complex
  - For instance, `auto`matic variables within a parreg are implicitly private
  - `static` variables within a parallel are implicitly shared!!

- Stay on the safe side:
  - Use the `default` clause for variables you care about!!
  - Use `shared` clauses
  - If you can, declare variables inside parreg, instead of marking them as `private`

- …informatics is the ~~art~~ science of managing data

How to run the examples



- Download the `Code/` folder from the course website

- Compile
- `$ gcc -fopenmp code.c -o code`

- Run (Unix/Linux)

```
$ ./code
```

- Run (Win/Cygwin)

```
$ ./code.exe
```

References

- "Calcolo parallelo" website
  - http://hipert.unimore.it/people/paolob/pub/PhD/index.html
- My contacts
  - paolo.burgio@unimore.it
  - http://hipert.mat.unimore.it/people/paolob/
- Useful links
  - http://www.google.com
  - http://www.openmp.org
  - https://gcc.gnu.org/
- A "small blog"
  - http://www.google.com