Introduction to OpenMP

Branislav Jansík

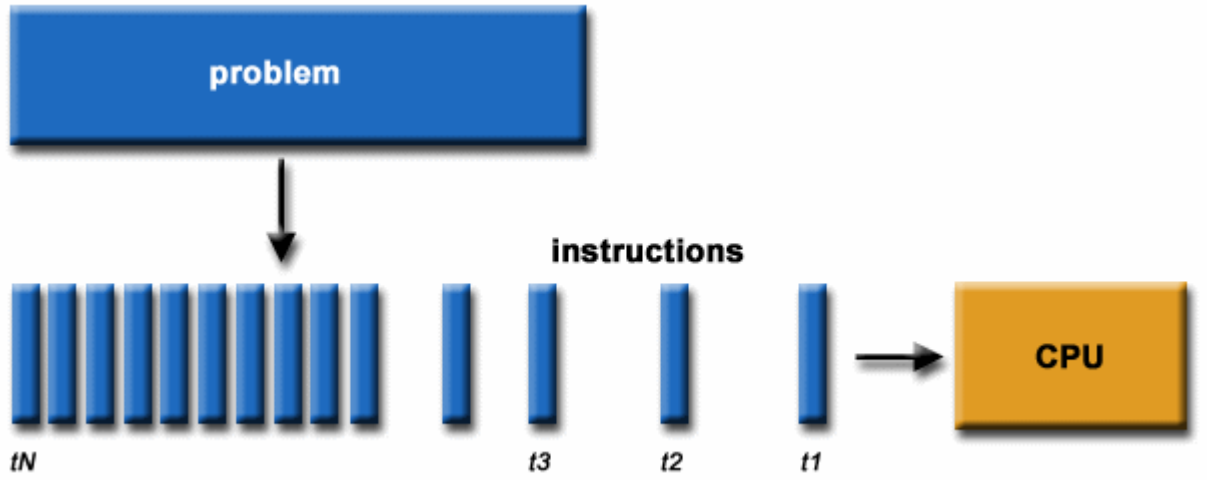
IT4Innovations
national10£$01
supercomputing
center0!£0#010

Resources

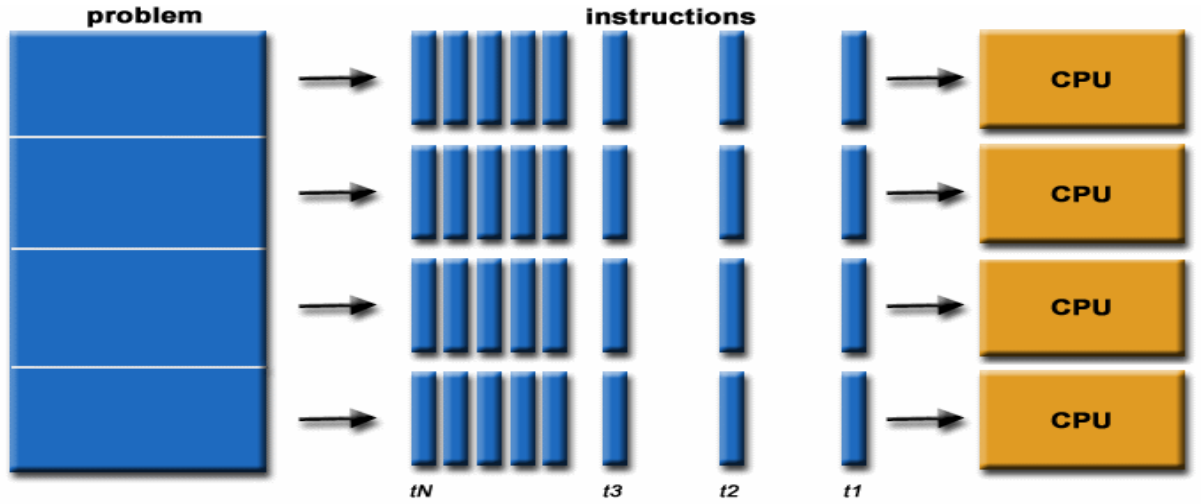
https://computing.llnl.gov/tutorials/openMP

Parallel computing recap

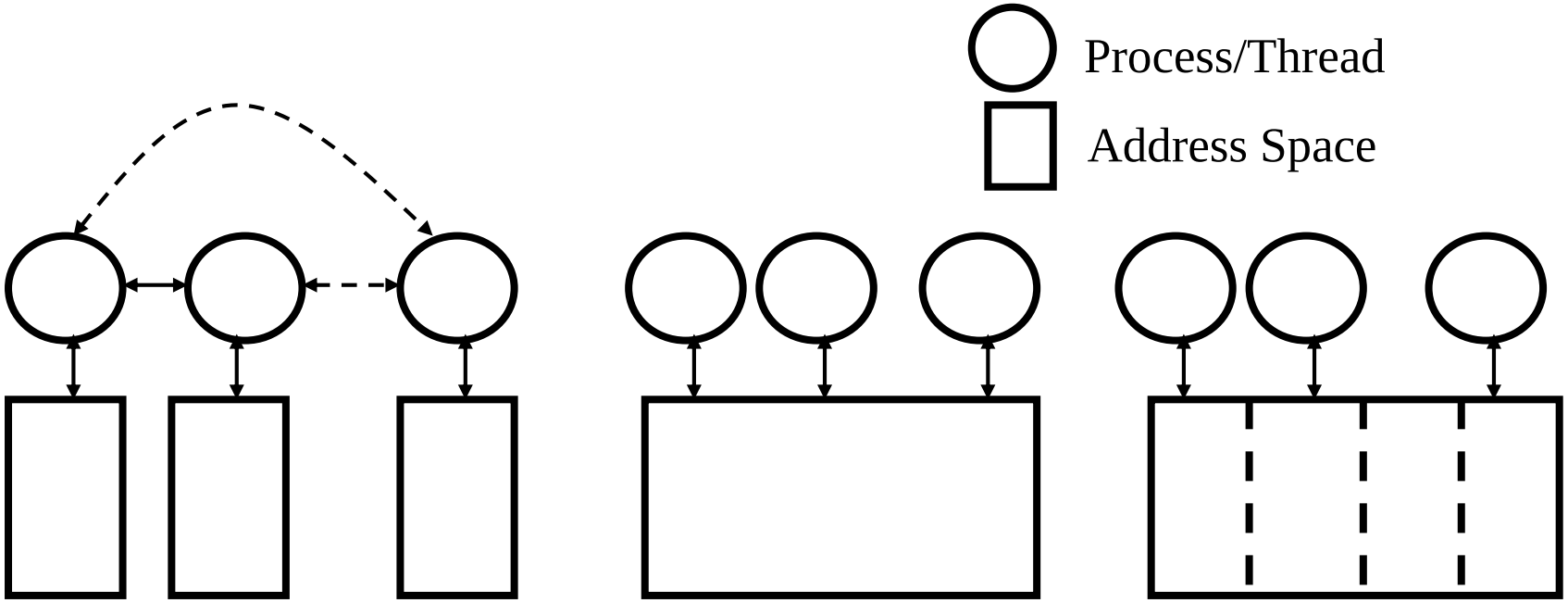
Serial



Parallel



Programming Models

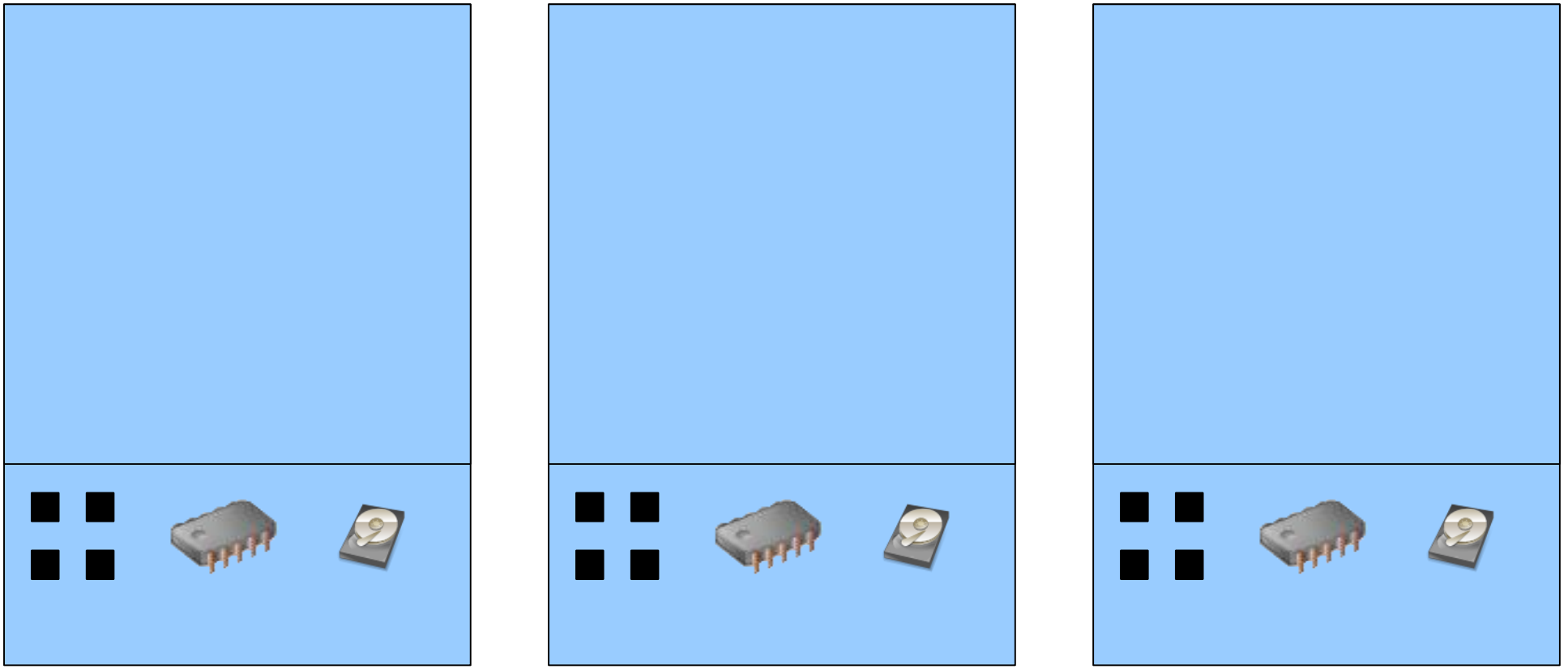


Message Passing
- MPI

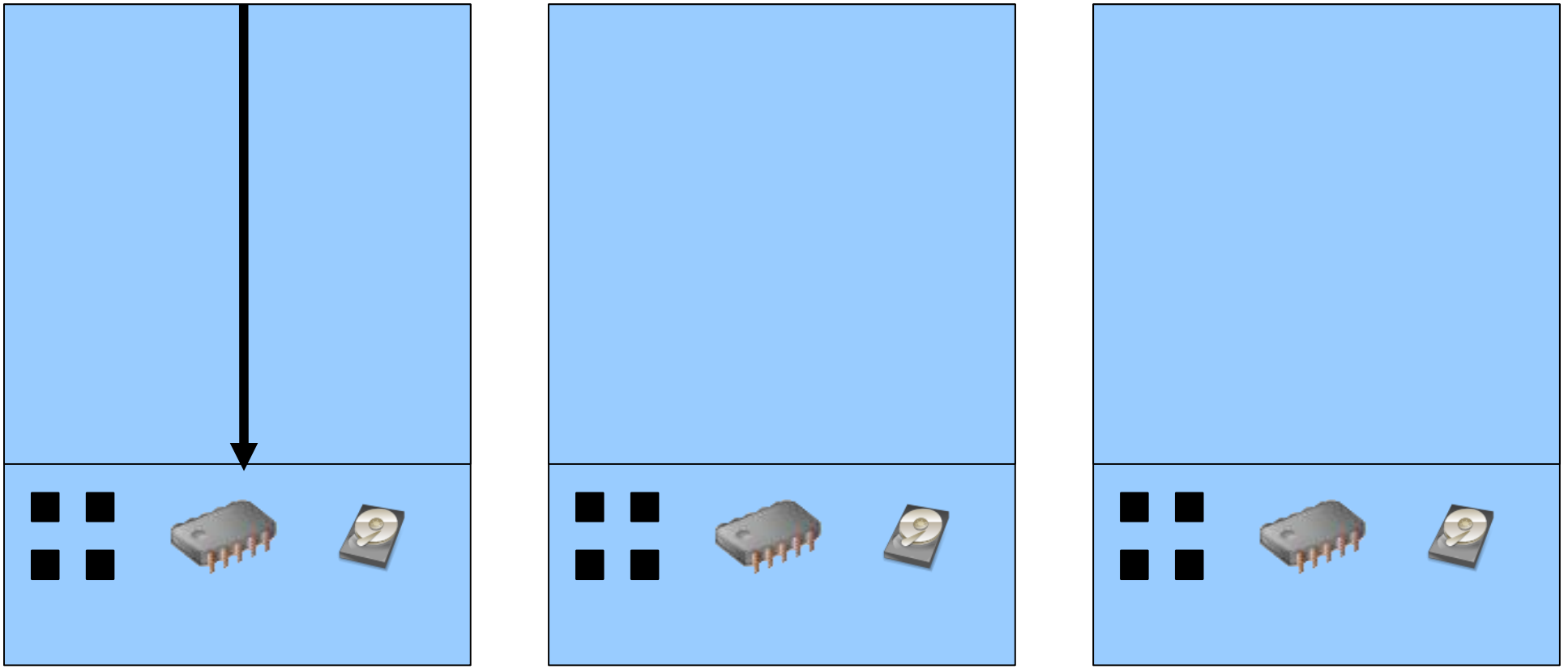
Shared Memory
- OpenMP

Distribute Shared Memory (DSM)
- PGAS
- UPC

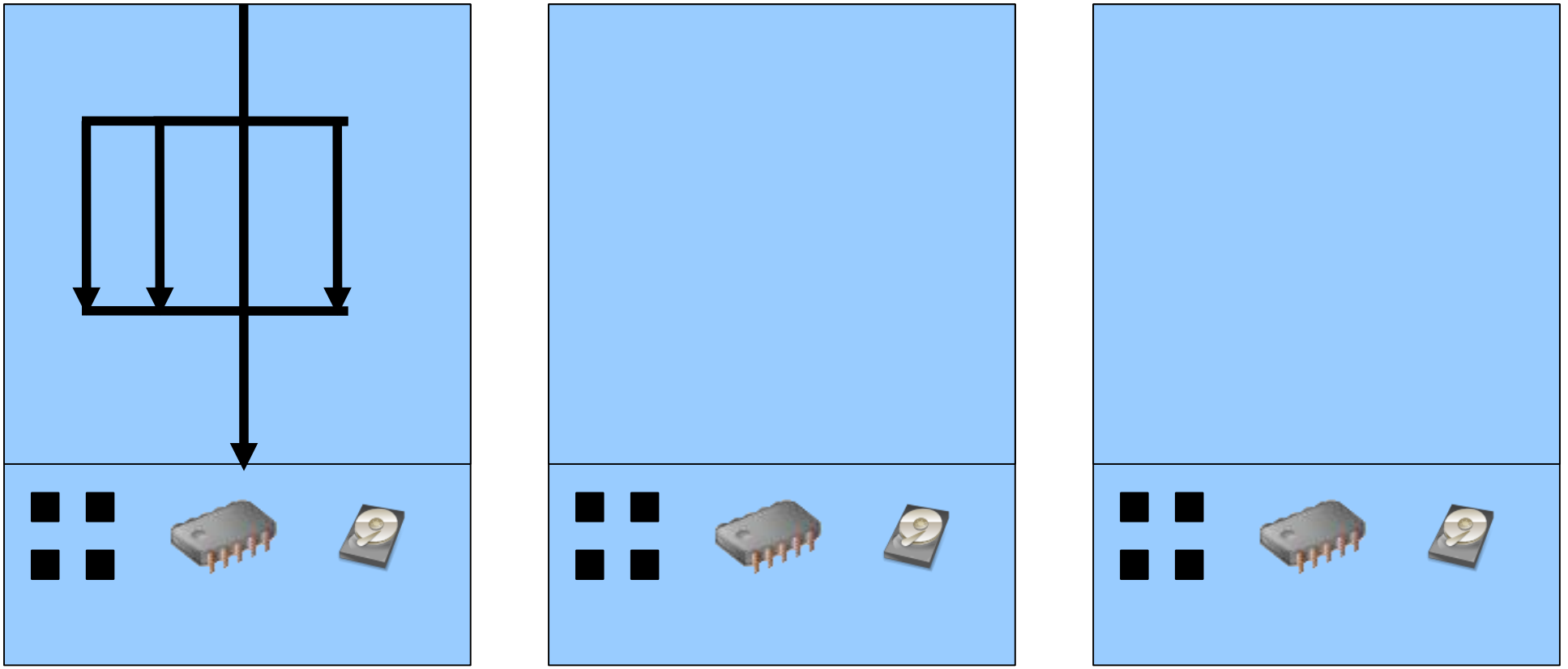
Node
Process
Thread
Shared memory
Task
Parallel Task
Rank



Node
Process
Thread
Shared memory
Task
Parallel Task
Rank

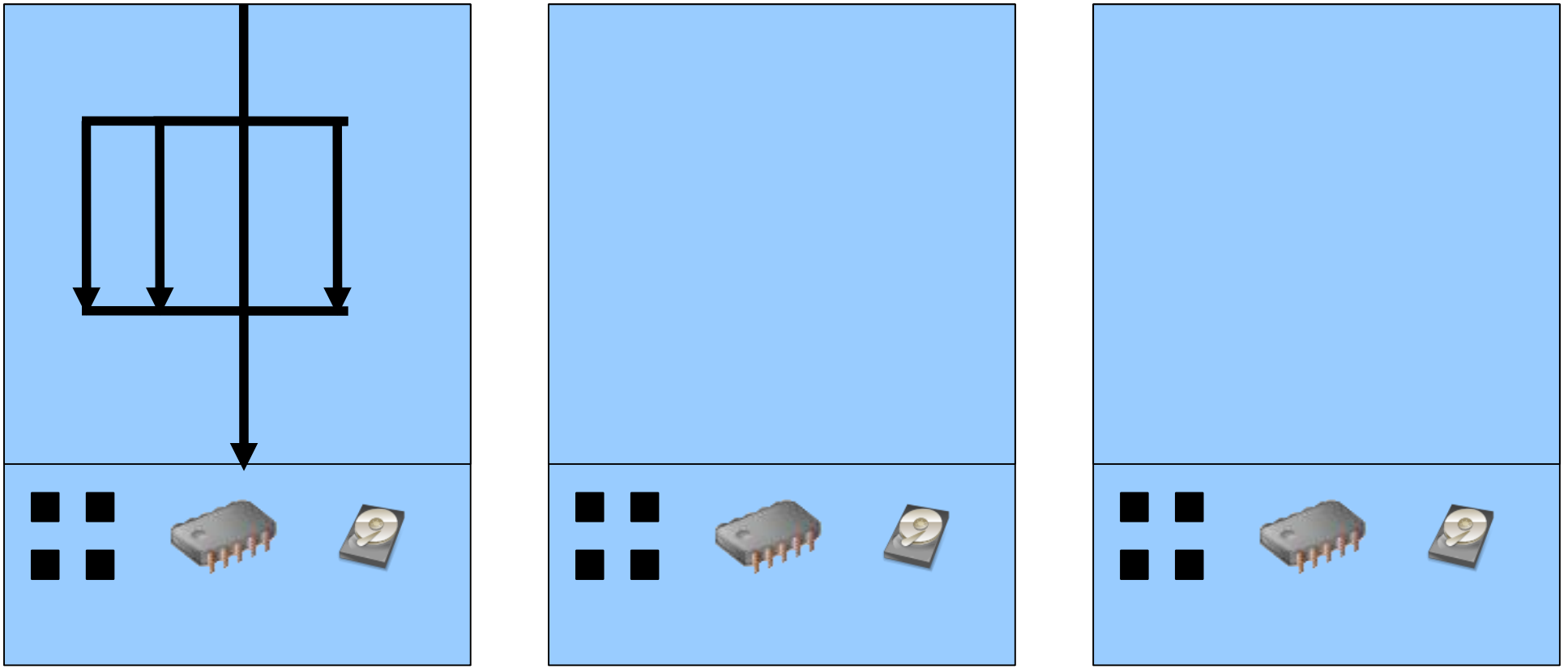


Node
Process
Thread



Open MP

multi-threaded, shared memory parallelism



How is OpenMP implemented?

OpenMP

is a set of compiler directives and that are included in the source code.

run time library that provides support functions
all comes along with compiler that offers OpenMP support

OpenMP

You give directives to the compiler
Compiler writes the code for you!

Compilation and linking OpenMP program

Compilation and linking - INTEL

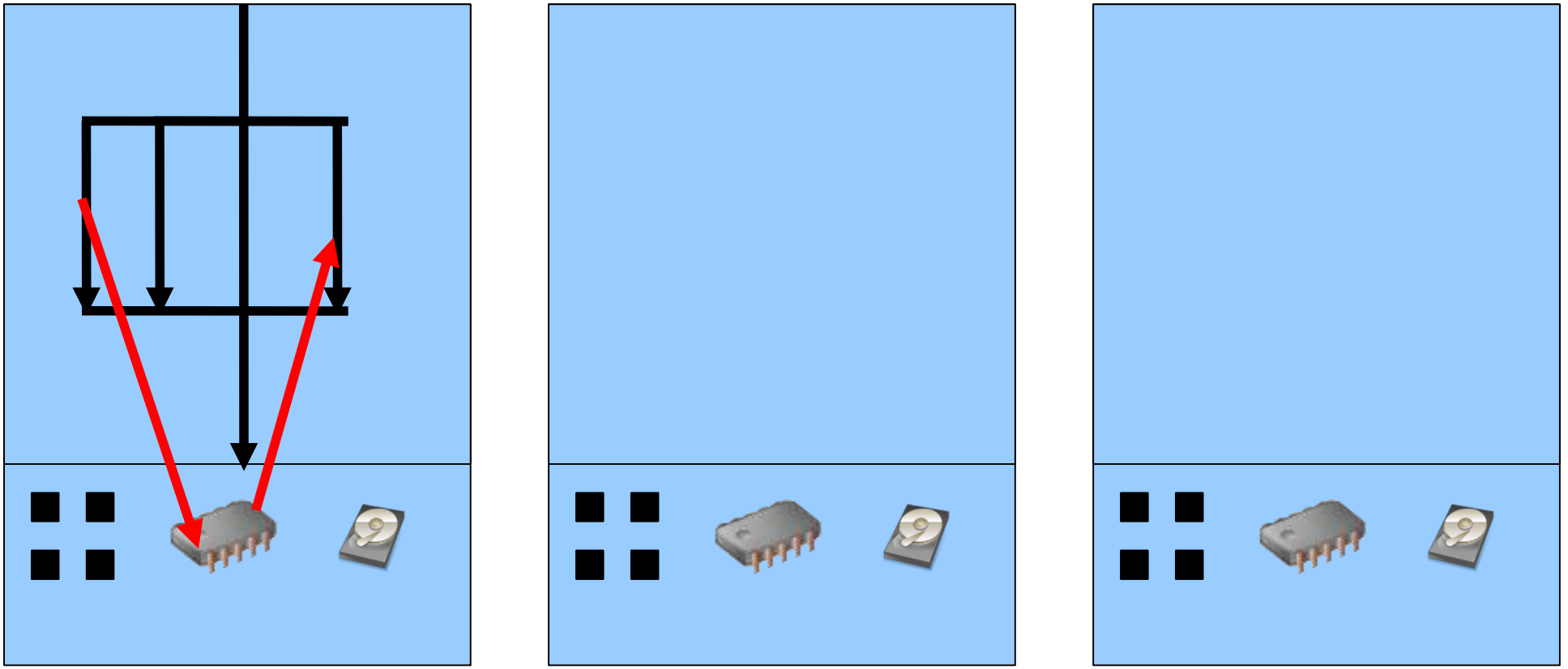
```
ifort -qopenmp myprog.F -o myprog.x

icc -qopenmp myprog.c -o myprog.x
```

Compilation and linking - GNU

```
gfortran -fopenmp myprog.F -o myprog.x

gcc -fopenmp myprog.c -o myprog.x
```

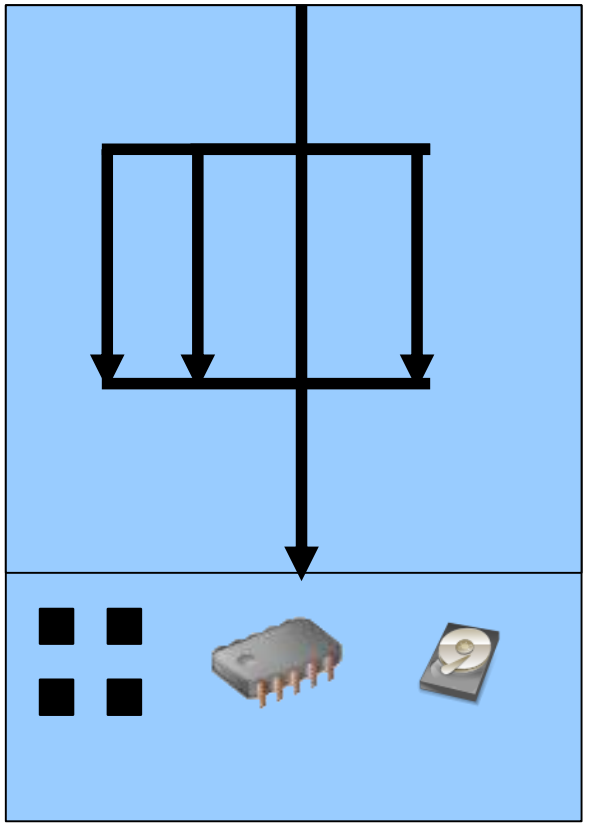
Open MP

Communication via shared memory



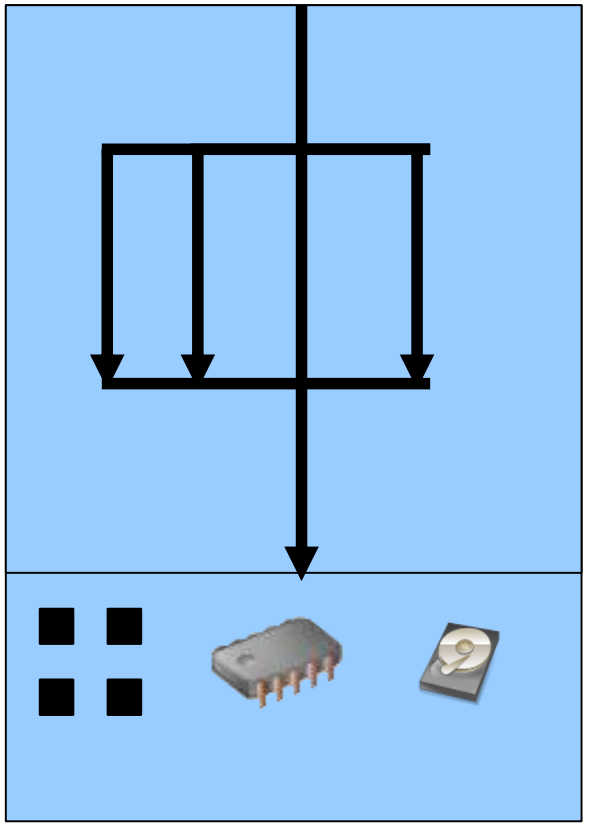
Basic OpenMP directives

```
Serial code

!$OMP PARALLEL PRIVATE(VAR1, VAR2) SHARED(VAR3)
  Parallel section executed by all threads
                .
                .
                .
  All threads join master thread and disband
!$OMP END PARALLEL

Resume serial code

#pragma omp parallel private(var1, var2) \
shared(var3)
{
  Parallel section executed by all threads
                .
                .
                .
  All threads join master thread and disband
}

Resume serial code
```



Basic OpenMP directives

```
Serial code

!$OMP PARALLEL PRIVATE(VAR1, VAR2) SHARED(VAR3)

tid = omp_get_thread_num()


!$OMP END PARALLEL

Resume serial code
```



Inside the structured block

- Exactly one entry point at the top
- Exactly one exit point at the bottom
- Branching in or out is not allowed
- Terminating the program is allowed

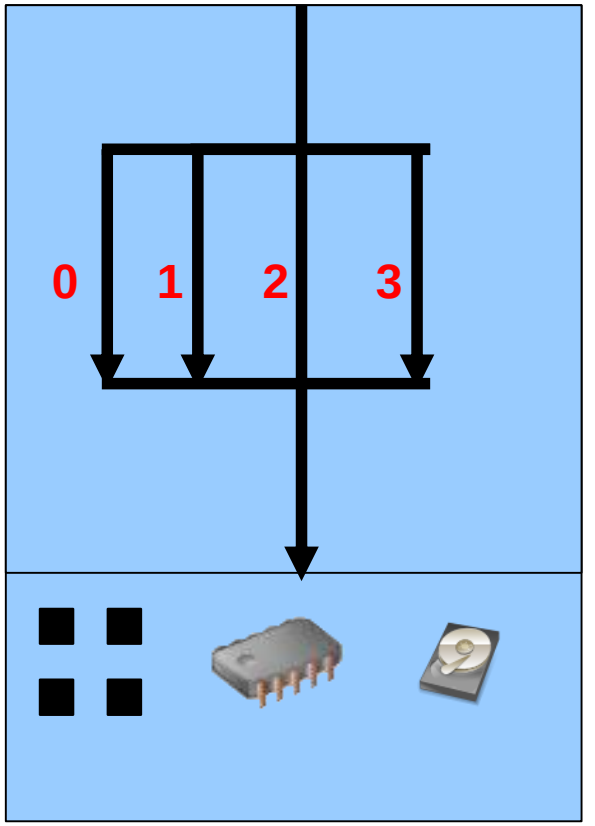
Controlling number of threads

- export OMP_NUM_THREADS= …
- num_threads(…) clause
- function omp_set_num_threads()

Basic OpenMP directives

```
Serial code

!$OMP PARALLEL PRIVATE(VAR1, VAR2) SHARED(VAR3)

tid = omp_get_thread_num()


!$OMP END PARALLEL

Resume serial code
```

Threads are identified by a unique thread-id (tid), returned by function omp_get_thread_num()

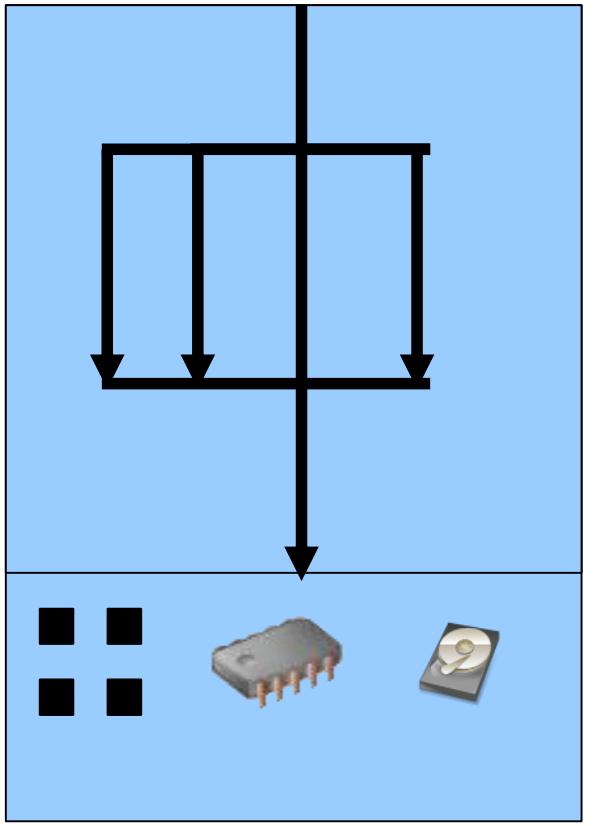


Basic OpenMP directives

Serial code

```
!$OMP PARALLEL FIRSTPRIVATE(VAR1, VAR2), &
!$OMP& REDUCTION(+:VAR3)



!$OMP END PARALLEL
```

Resume serial code



FIRSTPRIVATE

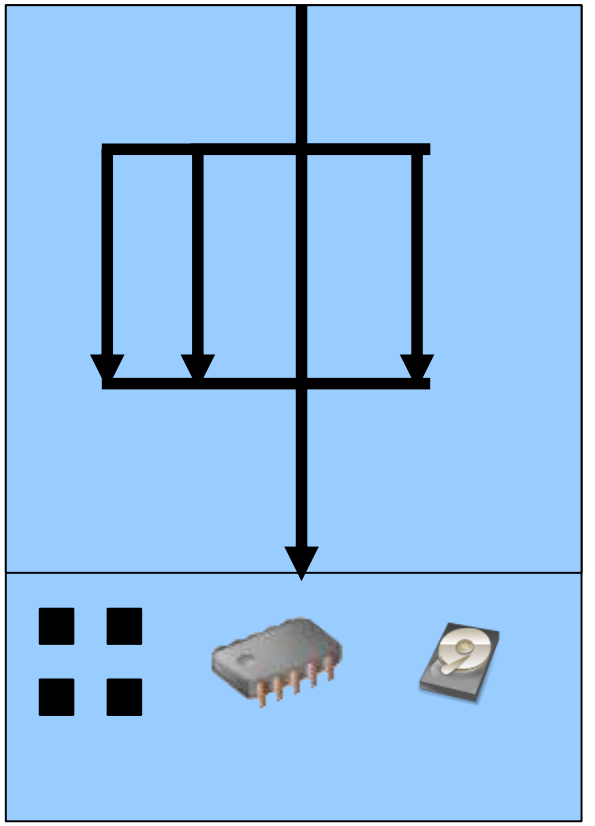
- Copy in data on entry to parallel region

REDUCTION

- Apply reduction on variables in the list
- Multiple operators (+,*,-,/,&,…)
- Variable must be scalar and shared
- Variable will be privatized within the parallel region

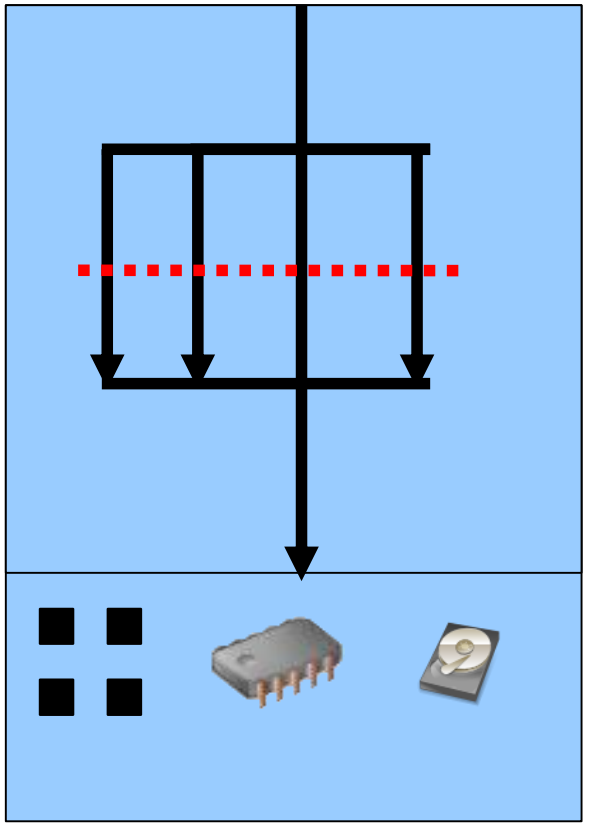
Basic OpenMP directives

```
Serial code

!$OMP PARALLEL PRIVATE(VAR1, VAR2) SHARED(VAR3)
  Parallel section executed by all threads
                .
!$OMP MASTER
  Code run by master thread only
!$OMP END MASTER
                .
                .
  All threads join master thread and disband
!$OMP END PARALLEL

Resume serial code
```



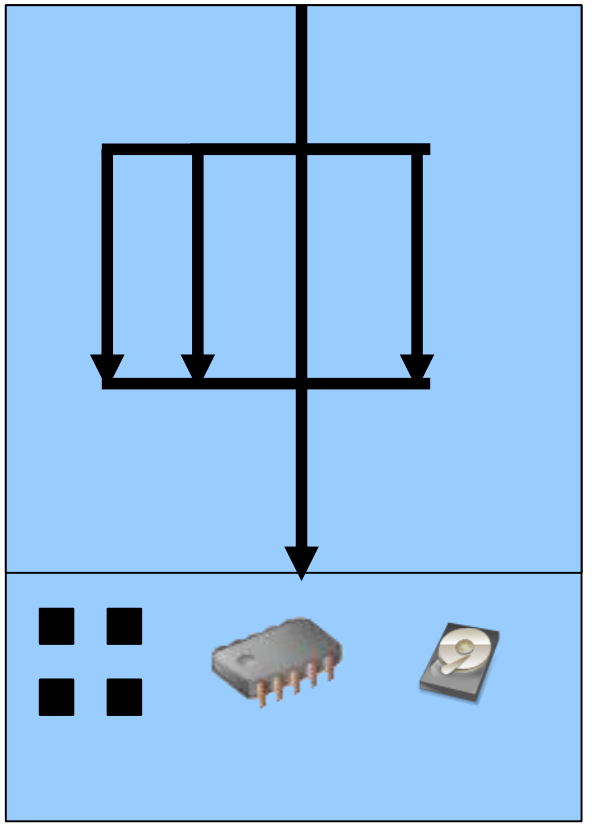
Basic OpenMP directives

```
Serial code

!$OMP PARALLEL PRIVATE(VAR1, VAR2) SHARED(VAR3)
  Parallel section executed by all threads
                .
!$OMP BARRIER

                .
                .
  All threads join master thread and disband
!$OMP END PARALLEL

Resume serial code
```



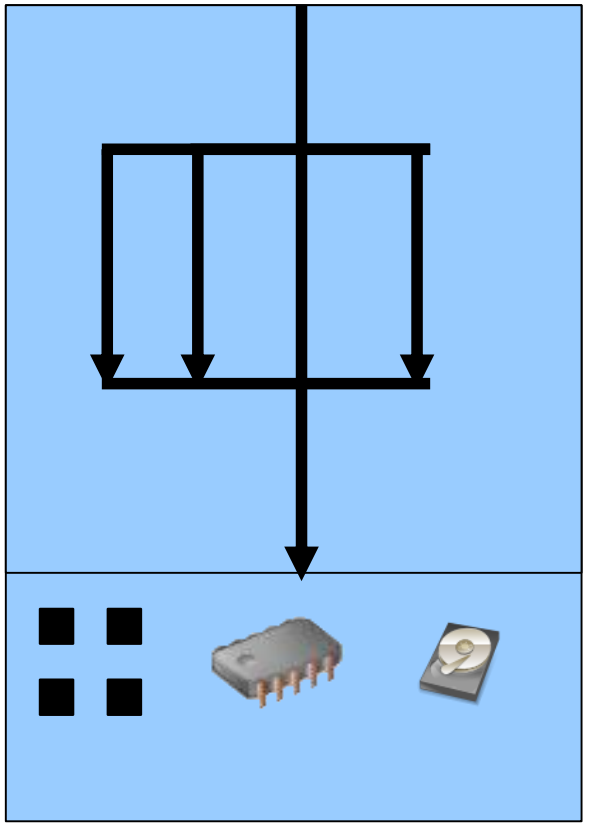
Basic OpenMP directives

```
Serial code

!$OMP PARALLEL PRIVATE(VAR1, VAR2) SHARED(VAR3)
  Parallel section executed by all threads
                .
!$OMP SINGLE
    .
    .
!$OMP END SINGLE
                .
                .
  All threads join master thread and disband
!$OMP END PARALLEL

Resume serial code
```



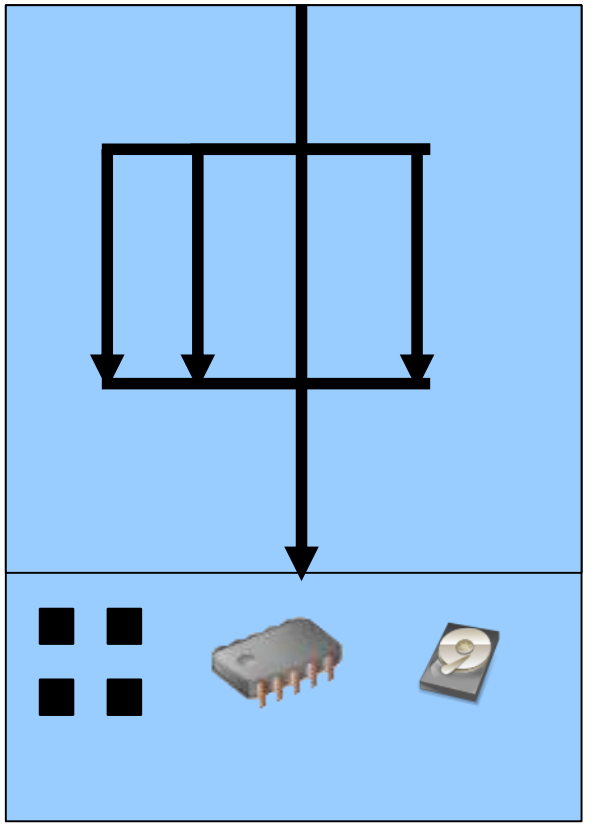
Basic OpenMP directives

```
Serial code

!$OMP PARALLEL PRIVATE(VAR1, VAR2) SHARED(VAR3)
  Parallel section executed by all threads
                .
!$OMP SECTIONS

!$OMP SECTION
 work in section 1
!$OMP END SECTION

!$OMP SECTION
 work in section 2
!OMP END SECTION
    .
    .
!$OMP END SECTIONS
                .
                .
  All threads join master thread and disband
!$OMP END PARALLEL

Resume serial code
```



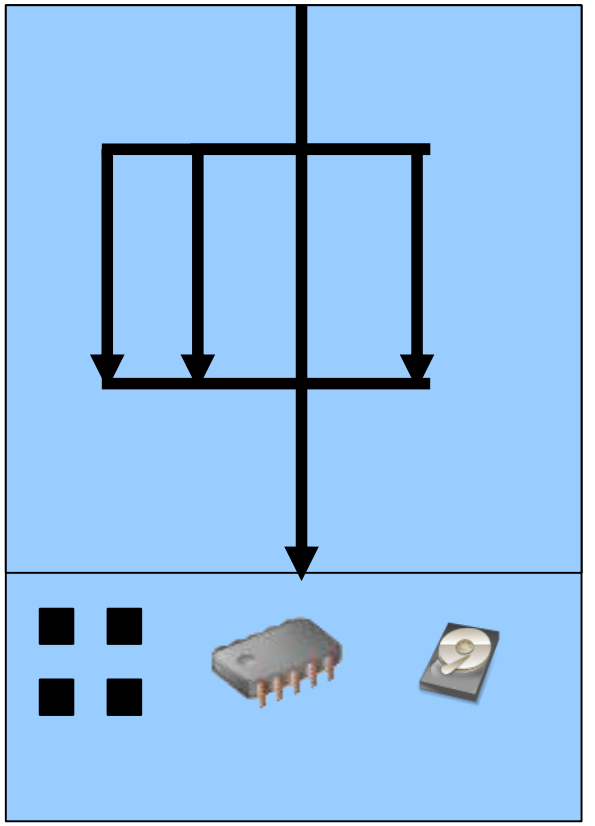
Basic OpenMP directives

```
Serial code

!$OMP PARALLEL PRIVATE(VAR1, VAR2) SHARED(VAR3)
  Parallel section executed by all threads
                .
!$OMP SECTIONS

!$OMP SECTION
 work in section 1
!$OMP END SECTION

!$OMP SECTION
 work in section 2
!OMP END SECTION
    .
    .
!$OMP END SECTIONS
                .
                .
  All threads join master thread and disband
!$OMP END PARALLEL

Resume serial code
```



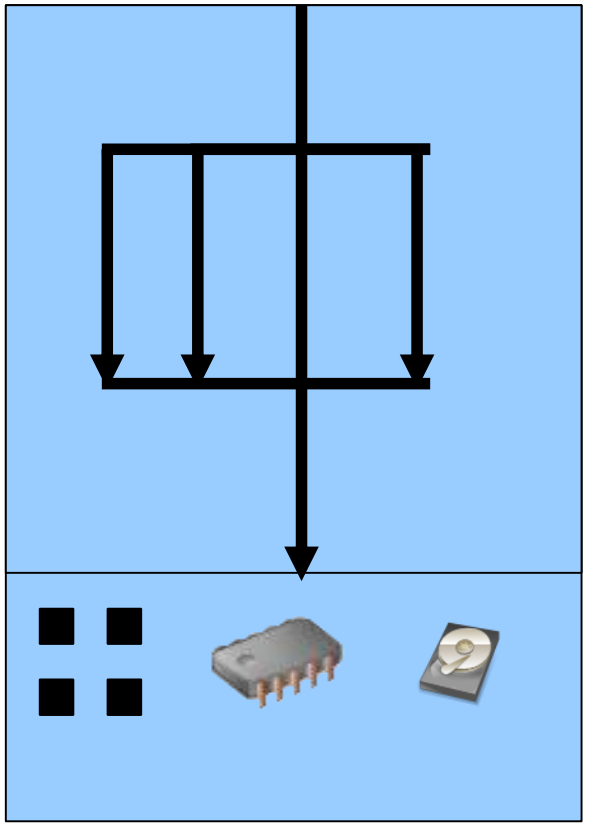
Basic OpenMP directives

```
Serial code

!$OMP PARALLEL PRIVATE(VAR1, VAR2) SHARED(VAR3)
  Parallel section executed by all threads
                .
!$OMP DO SCHEDULE(DYNAMIC,10) LASTPRIVATE(VAR4)
  do i=1,n
     body
  end do
!$OMP END DO
                .
                .
  All threads join master thread and disband
!$OMP END PARALLEL

Resume serial code
```



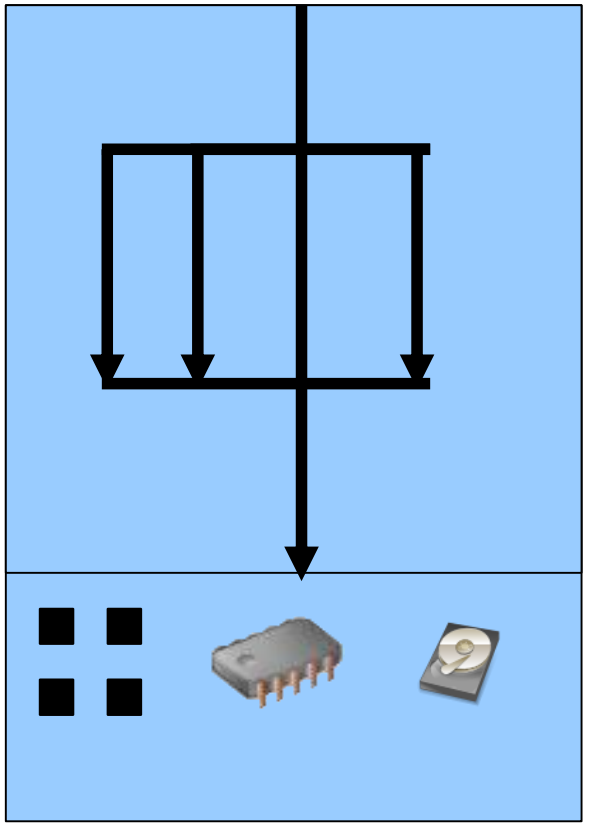
Basic OpenMP directives

```
Serial code

!$OMP PARALLEL DO PRIVATE(i,VAR1, VAR2)
!$OMP SHARED(VAR3) SCHEDULE(DYNAMIC,10)

  do i=1,n
     body
  end do

!$OMP END PARALLEL DO

Resume serial code
```



Basic OpenMP directives

```
Serial code

!$OMP PARALLEL PRIVATE(VAR1, VAR2) SHARED(VAR3)
  Parallel section executed by all threads
                .
!$OMP CRITICAL
  Some critical part, may only be accessed
  by one thread at time
!$OMP END CRITICAL
                .
                .
  All threads join master thread and disband
!$OMP END PARALLEL

Resume serial code
```



OpenMP runtime library functions

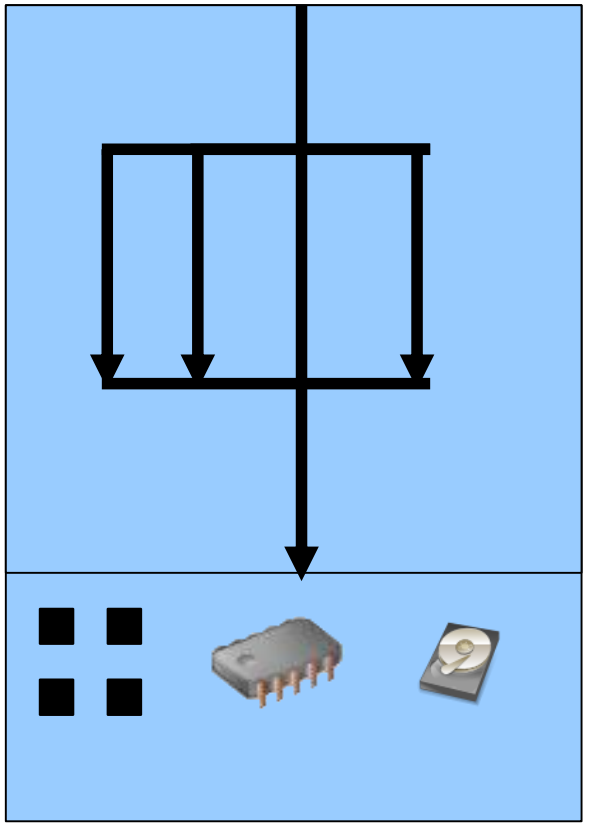
Routine	Purpose
OMP_SET_NUM_THREADS	Sets the number of threads that will be used in the next parallel region
OMP_GET_NUM_THREADS	Returns the number of threads that are currently in the team executing the parallel region from which it is called
OMP_GET_MAX_THREADS	Returns the maximum value that can be returned by a call to the OMP_GET_NUM_THREADS function
OMP_GET_THREAD_NUM	Returns the thread number of the thread, within the team, making this call.
OMP_GET_THREAD_LIMIT	Returns the maximum number of OpenMP threads available to a program
OMP_GET_NUM_PROCS	Returns the number of processors that are available to the program
OMP_GET_WTIME	Provides a portable wall clock timing routine
OMP_GET_WTICK	Returns a double-precision floating point value equal to the number of seconds between successive clock ticks

Clauses accepted in the Directives

Clause	Directive					
	PARALLEL	DO/for	SECTIONS	SINGLE	PARALLEL DO/for	PARALLEL SECTIONS
IF	●				●	●
PRIVATE	●	●	●	●	●	●
SHARED	●	●			●	●
DEFAULT	●				●	●
FIRSTPRIVATE	●	●	●	●	●	●
LASTPRIVATE		●	●		●	●
REDUCTION	●	●	●		●	●
COPYIN	●				●	●
SCHEDULE		●			●	
ORDERED		●			●	
NOWAIT		●	●	●		

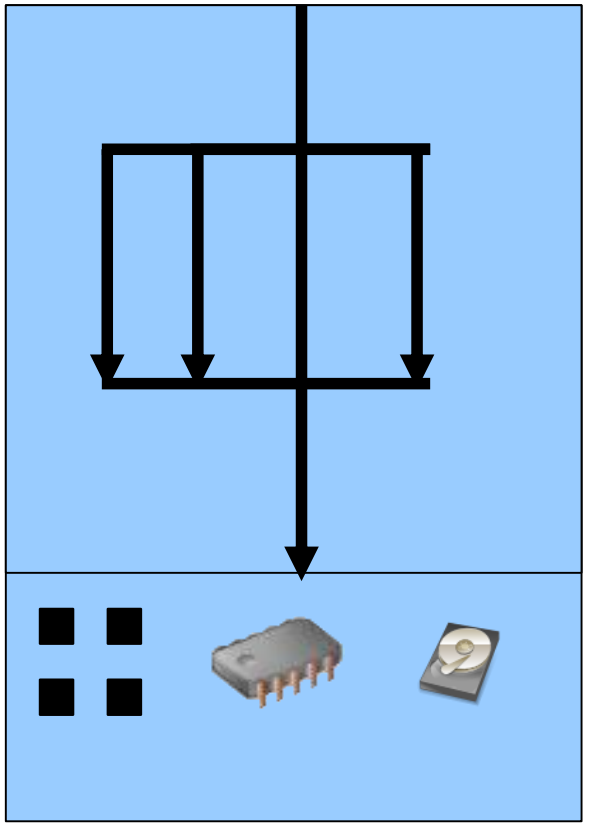
Common OpenMP pitfalls

```
VAR1 = -10

!$OMP PARALLEL PRIVATE(VAR1)
  Parallel section executed by all threads
                .
                .
                .
  All threads join master thread and disband
!$OMP END PARALLEL

Resume serial code
```



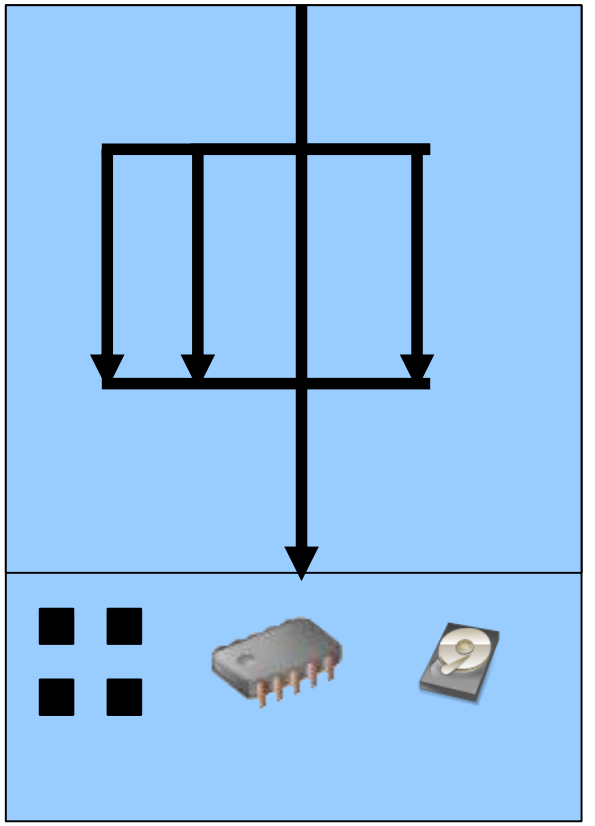
Common OpenMP pitfalls

```
VAR1 = -10

!$OMP PARALLEL FIRSTPRIVATE(VAR1)
  Parallel section executed by all threads
                .
                .
                .
  All threads join master thread and disband
!$OMP END PARALLEL

Resume serial code
```



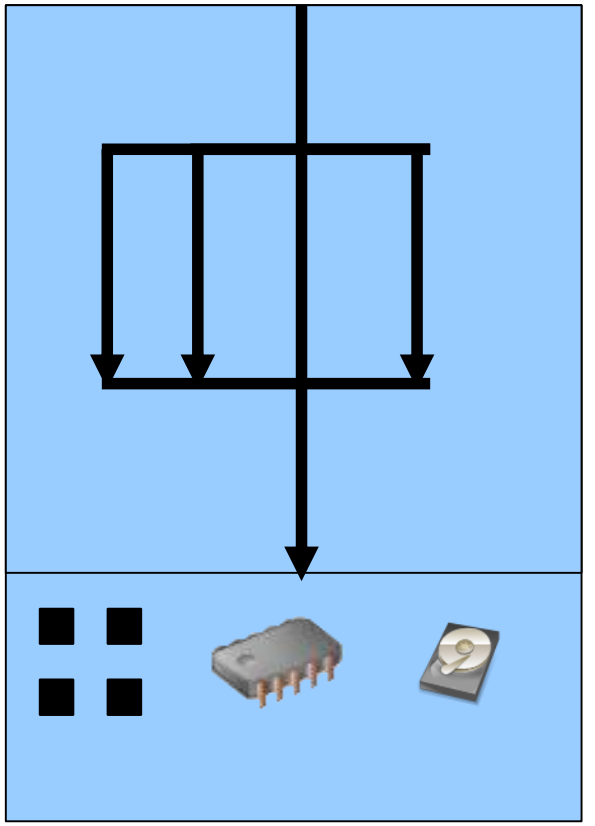
Common OpenMP pitfalls

```
!$OMP PARALLEL DEFAULT(SHARED)
  DO I=1,N
    ...
  ENDDO
!$OMP END PARALLEL

Resume serial code
```



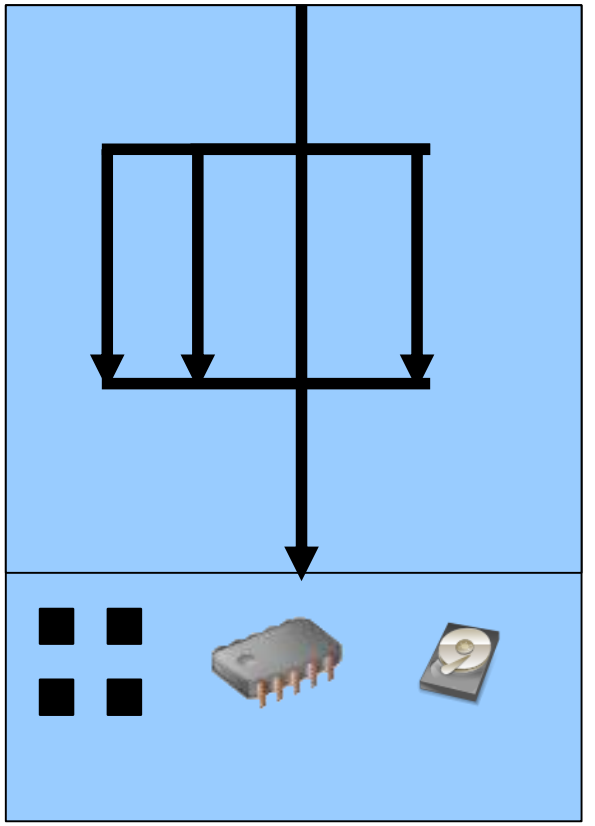
Common OpenMP pitfalls

```
!$OMP PARALLEL DEFAULT(PRIVATE)
  DO I=1,N
    ...
  ENDDO
!$OMP END PARALLEL

Resume serial code
```



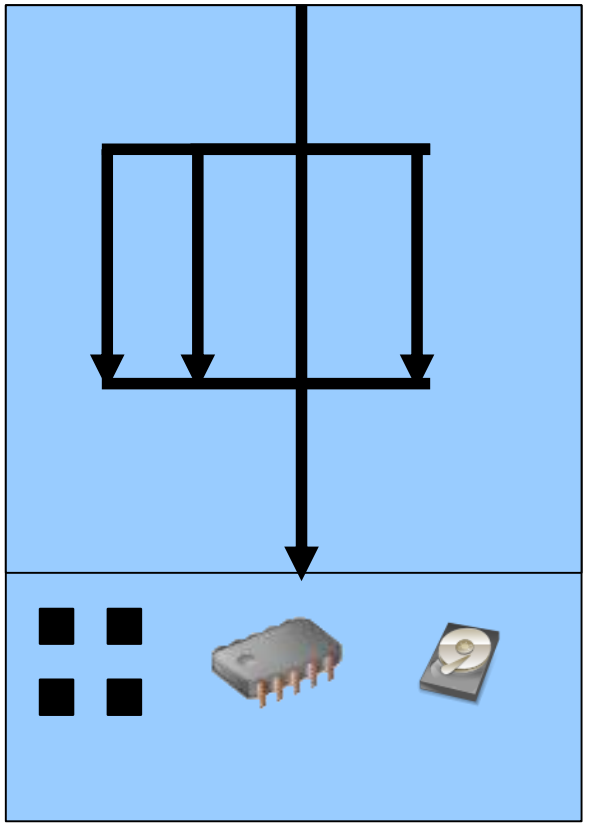
Common OpenMP pitfalls

```
!$OMP PARALLEL DEFAULT(NONE) PRIVATE(I), &
!$OMP& SHARED(N)
  DO I=1,N
    ...
  ENDDO
!$OMP END PARALLEL

Resume serial code
```



Common OpenMP pitfalls

```
VAR1 = 0

!$OMP PARALLEL DEFAULT(NONE) SHARED(VAR1)

VAR1 = VAR1 + 1


!$OMP END PARALLEL

Resume serial code
```



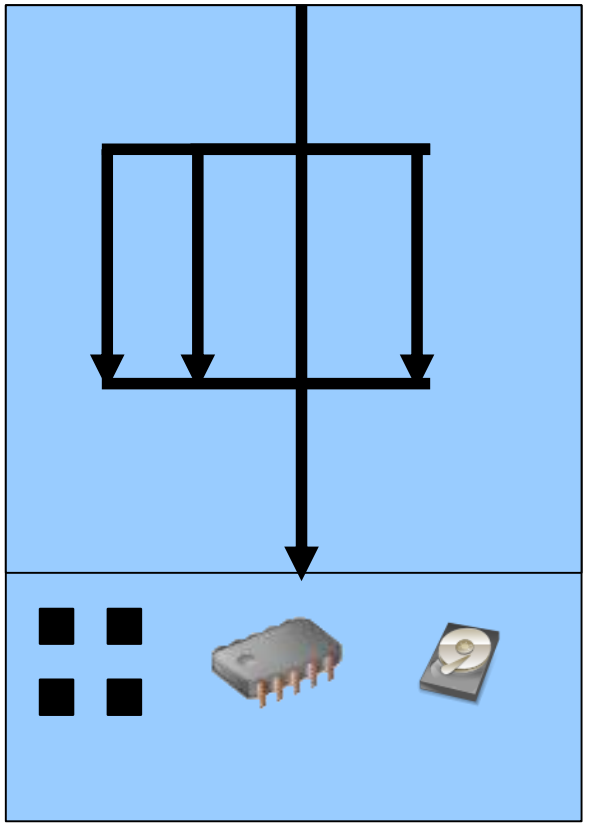
Each thread has its own temporary view on the data

- Applicable to shared data only
- Means different threads may temporarily not see the same value for the same variable ...

All threads have a consistent view of the memory after synchronization constructs

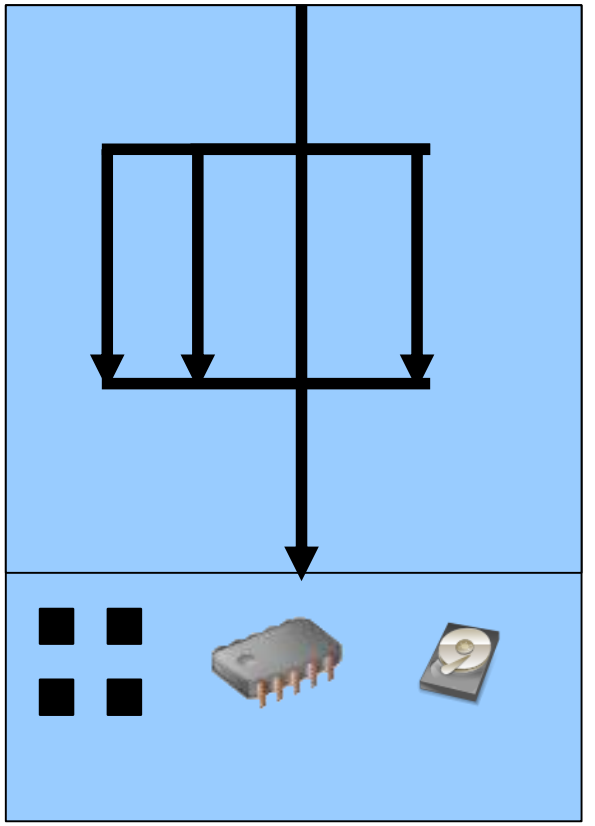
Common OpenMP pitfalls

```
DIMENSION VAR1(1000)
VAR1 = -10

!$OMP PARALLEL PRIVATE(VAR1)
  Parallel section executed by all threads
                .
                .
                .
  All threads join master thread and disband
!$OMP END PARALLEL

Resume serial code
```



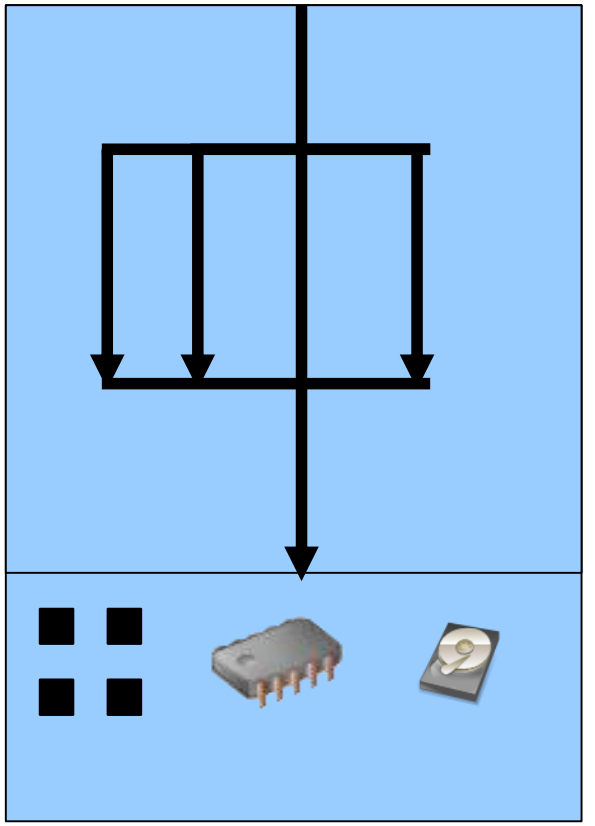
Common OpenMP pitfalls

```
POINTER VAR1
ALLOCATE(VAR1(1000))

!$OMP PARALLEL PRIVATE(VAR1)
  Parallel section executed by all threads
                .
                .
                .
  All threads join master thread and disband
!$OMP END PARALLEL

Resume serial code
```



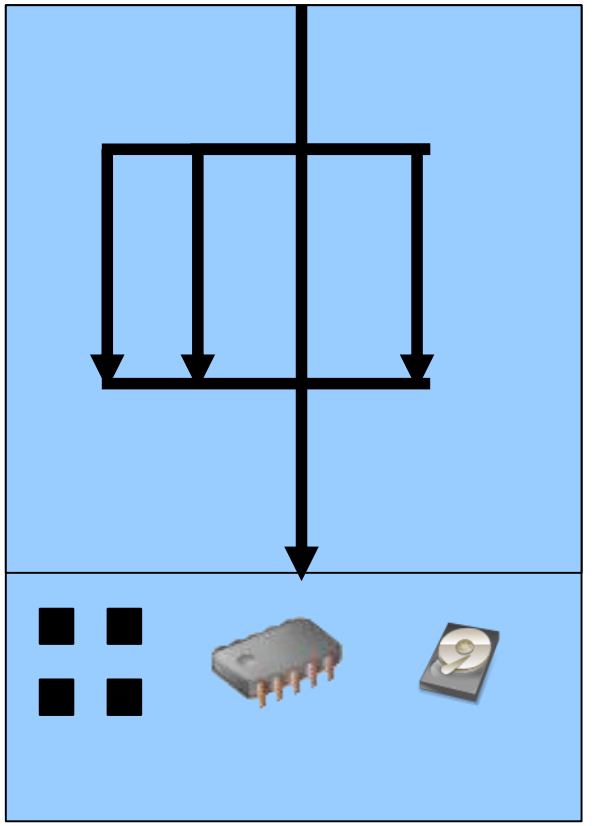
Common OpenMP pitfalls

```
POINTER VAR1
ALLOCATE(VAR1(1000))

!$OMP PARALLEL FIRSTPRIVATE(VAR1)
  Parallel section executed by all threads
                .
                .
                .
  All threads join master thread and disband
!$OMP END PARALLEL

Resume serial code
```



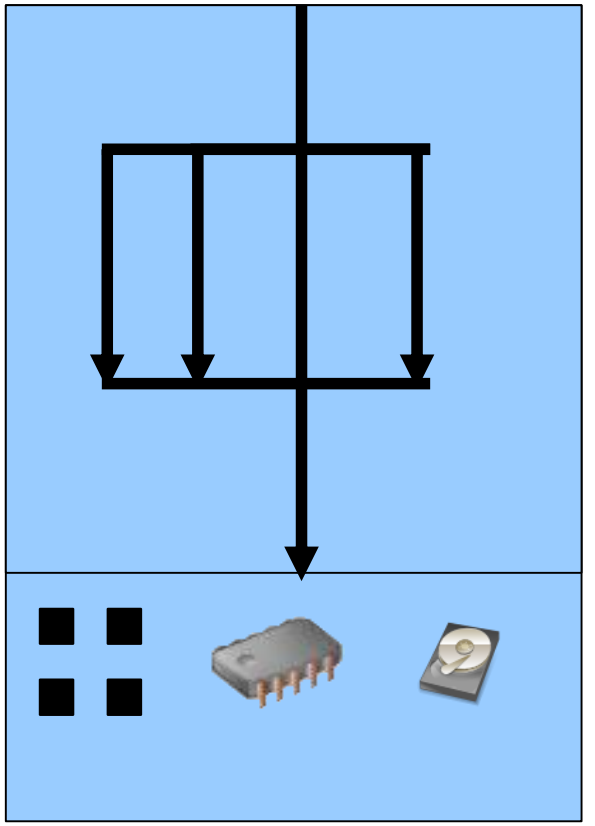
Common OpenMP pitfalls

```
POINTER VAR1


!$OMP PARALLEL PRIVATE(VAR1)
  Parallel section executed by all threads
  ALLOCATE(VAR1(1000))                .
              .
              .
  All threads join master thread and disband
!$OMP END PARALLEL

Resume serial code
DEALLOCATE(VAR1)
```



Common OpenMP pitfalls

```
Type MyStruct
 POINTER VAR1
 VAR2
 VAR3
End Type

MyStruct%VAR2 = 2; MyStruct%VAR3 = 3
ALLOCATE(MyStruct%VAR1(1000))

!$OMP PARALLEL FIRSTPRIVATE(VAR1)
  Parallel section executed by all threads
                .
                .
                .
  All threads join master thread and disband
!$OMP END PARALLEL

Resume serial code
```



False sharing

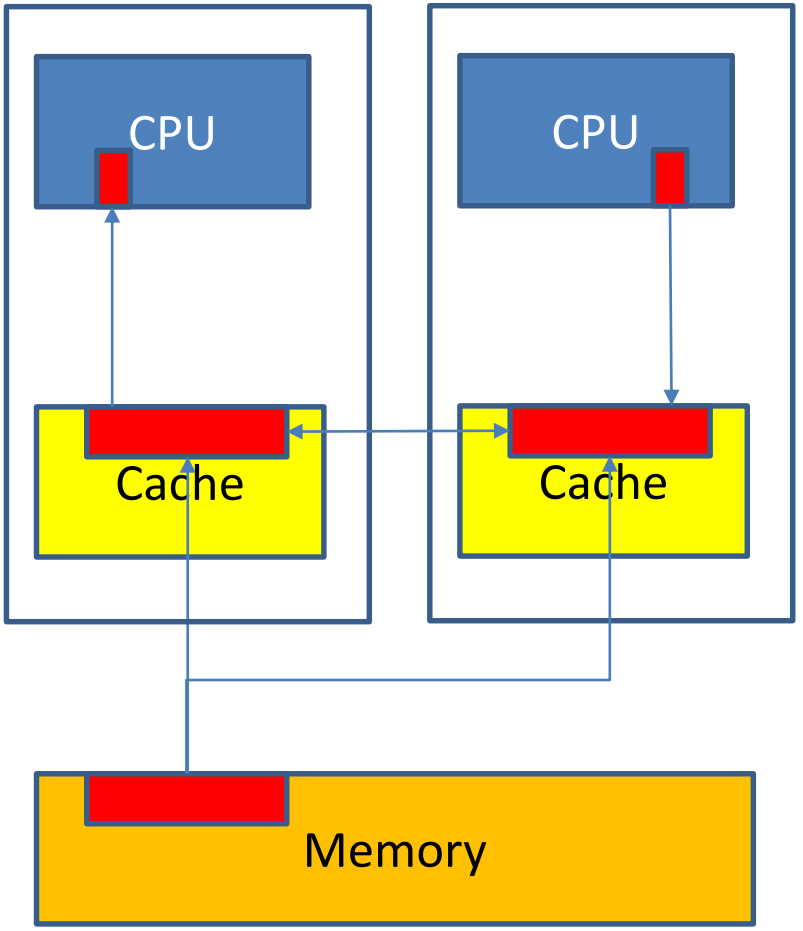
False sharing occurs when

- Different threads use the same cache line
- One of the threads writes the same cache line

As a result, cache line may be evicted by cache coherence protocol.

False sharing is a performance problem, not a correctness issue.

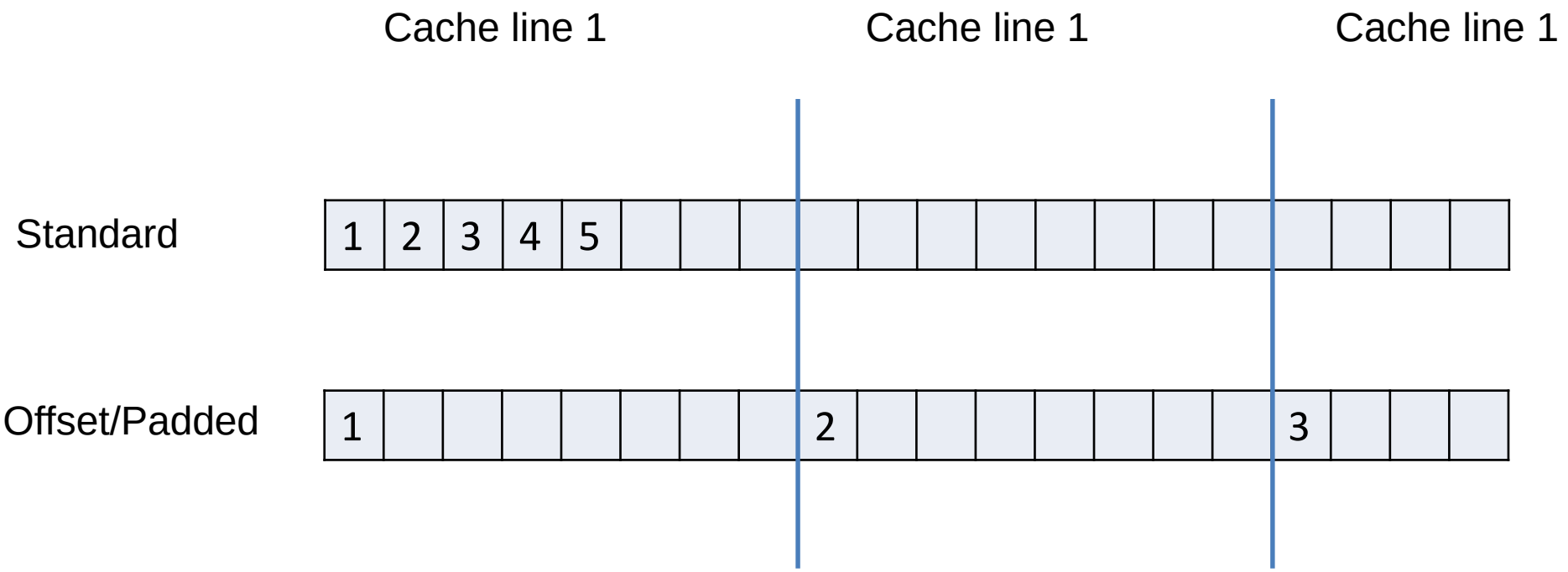
False sharing is an inherent artifact of automatically synchronized cache protocols



False sharing

```
struct foo {
    int x;
    int y;
};

static struct foo f;

/* The two following functions are running concurrently: */

int sum_a(void)
{
    int s = 0;
    int i;
    for (i = 0; i < 1000000; ++i)
        s += f.x;
    return s;
}

void inc_b(void)
{
    int i;
    for (i = 0; i < 1000000; ++i)
        ++f.y;
}
```

False sharing



Make sure to place unrelated (and often updated) variables on different cachelines

- Padding the data structures
- Offsetting the data access

Thank you!

Branislav Jansík

IT4Innovations
national10£$01
supercomputing
center0!£0#010