CAP-REU program
Brief introduction and tutorial on parallel computing

Joaquín E. Drut

University of North Carolina at Chapel Hill

THE UNIVERSITY
of NORTH CAROLINA
at CHAPEL HILL

(Biased) Overview of parallel computing paradigms

What do you imagine parallel computing is?
(hint: it’s exactly what you think it is)

(Biased) Overview of parallel computing paradigms

What do you imagine parallel computing is?
(hint: it’s exactly what you think it is)

(Biased) Overview of parallel computing paradigms

What do you imagine parallel computing is?
(hint: it’s exactly what you think it is)

Suppose you are given the hardware...
How do you take advantage of it?

(Biased) Overview of parallel computing paradigms

What do you imagine parallel computing is?
(hint: it's exactly what you think it is)

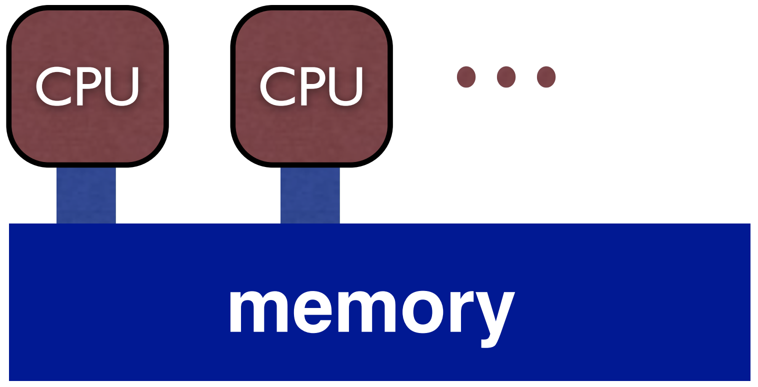
Suppose you are given the hardware...
How do you take advantage of it?

It depends...

Different kinds of hardware allow different kinds of parallelism

Your computer (laptop or desktop):

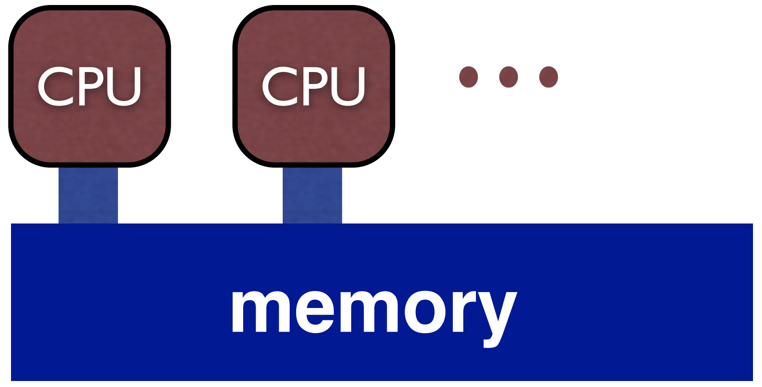
- two **processors** or more (CPUs)
- **shared memory** between them (RAM)
-



Different kinds of hardware allow different kinds of parallelism

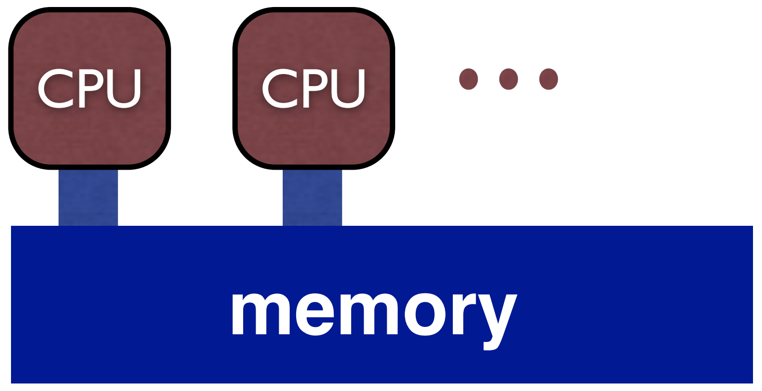
Your computer (laptop or desktop):

- two **processors** or more (CPUs)
- **shared memory** between them (RAM)
- general purpose graphics processing unit (GPGPU)



Different kinds of hardware allow different kinds of parallelism

Your computer (laptop or desktop):
- two **processors** or more (CPUs)
- **shared memory** between them (RAM)
- general purpose graphics processing unit (GPGPU)

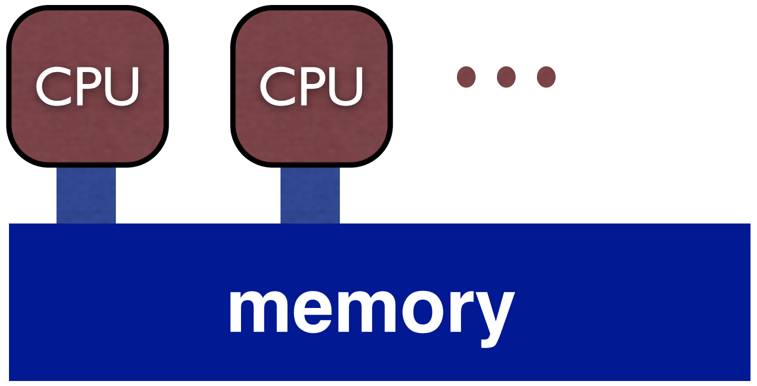


The collective memory is **shared**, so we speak of a **shared-memory architecture.**

Different kinds of hardware allow different kinds of parallelism

Your computer (laptop or desktop):

- two **processors** or more (CPUs)
- **shared memory** between them (RAM)
- general purpose graphics processing unit (GPGPU)

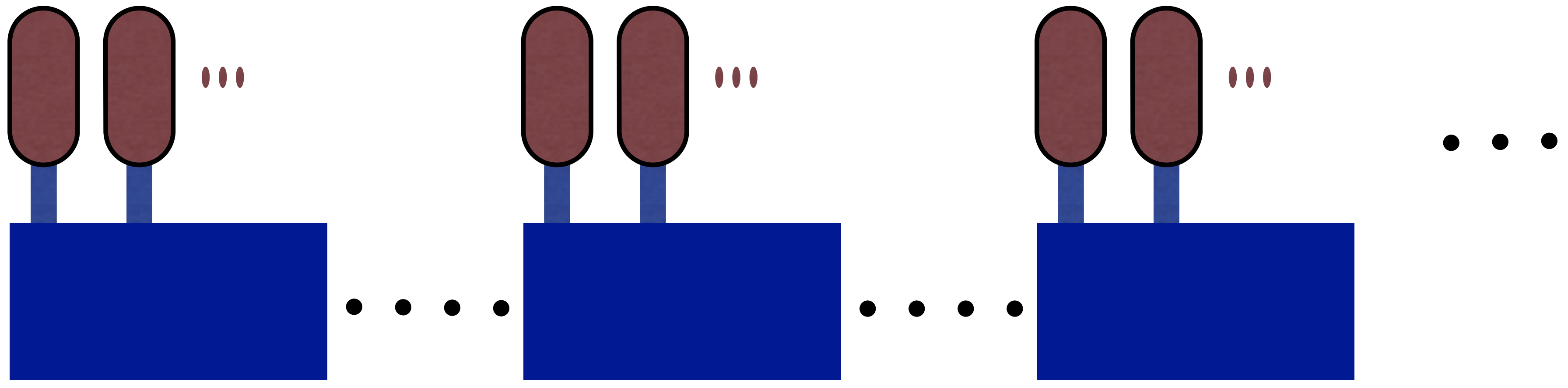


The collective memory is **shared**, so
we speak of a **shared-memory architecture.**

Every CPU has access to the same memory...
What kinds of problems can we expect to have?

Different kinds of hardware allow different kinds of parallelism

A typical cluster: **nodes**
- twelve processors or more per node
- shared memory within the node
- **interconnects**: Infiniband (fast), or at least ethernet



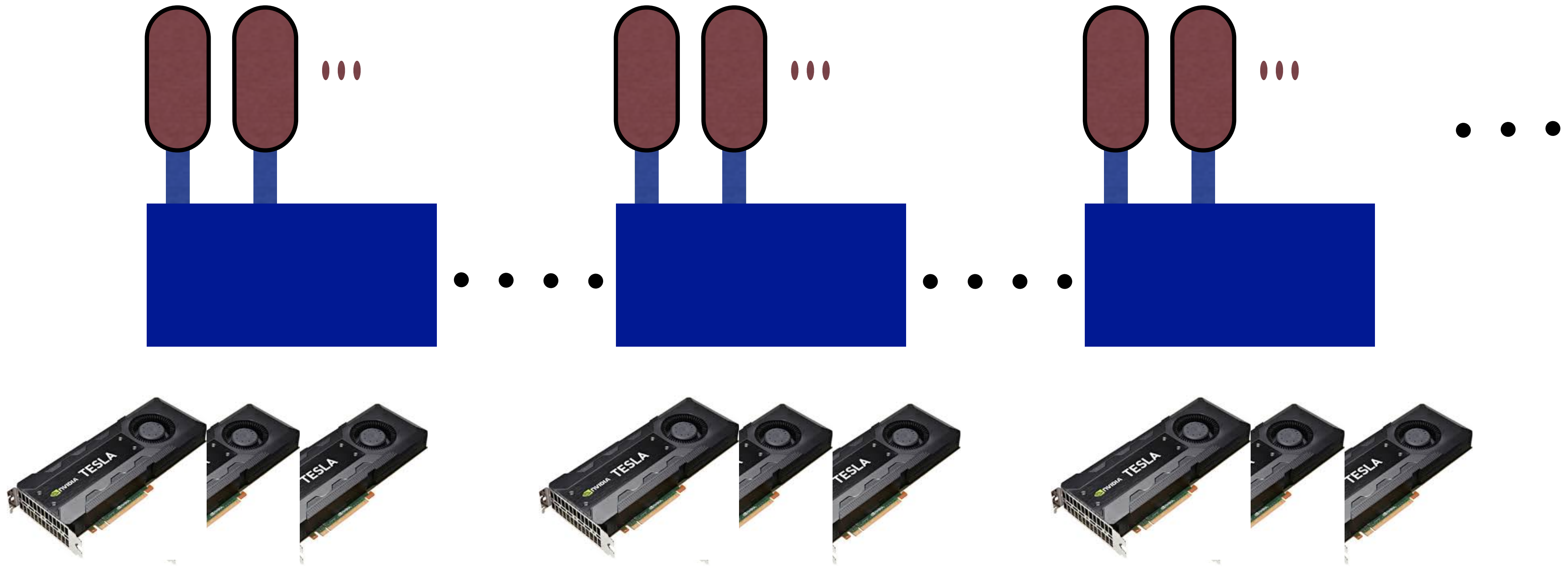
Memory on each node is **not shared** with other nodes.
The collective memory is said to be **distributed**, so
we speak of a **distributed-memory architecture.**

When is this architecture (obviously) better than the previous one?

Different kinds of hardware allow different kinds of parallelism

A modern high-performance cluster

- lots of processors or more per node
- lots of shared memory within the node
- fast interconnects
- multiple GPUs per node



How do we use all this?

Questions so far?

So... how do we use this in practice? (biased again!)

Shared memory: OpenMP (*not to be confused with OpenMPI*, see below!)

- It's the easiest one to work with (ideally computers would have enough memory and bus capacity that this would be all you'd need)

- It can do many simple things automatically (like loops), by creating **threads** on the fly, all of which run **concurrently**.

So... how do we use this in practice? (biased again!)

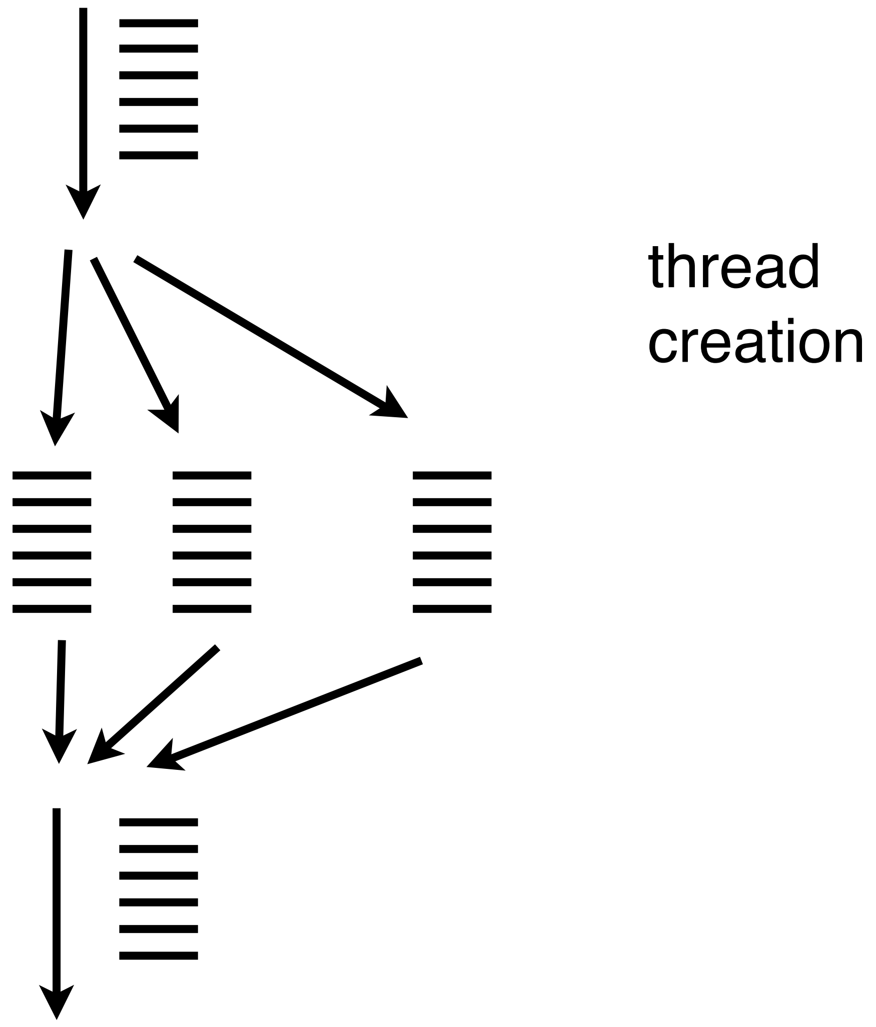
Shared memory: OpenMP (*not to be confused with OpenMPI*, see below!)

- It's the easiest one to work with (ideally computers would have enough memory and bus capacity that this would be all you'd need)

- It can do many simple things automatically (like loops), by creating **threads** on the fly, all of which run **concurrently**.

Your code execution looks like this:

No messaging, only writing to memory, and forking and un-forking events



Example: Parallelizing a loop with OpenMP

To parallelize a loop...

```
!$OMP PARALLEL DO
DO i = 1, 1000
        !... your code here...



END DO
!$OMP END PARALLEL DO
```

Example: Parallelizing a loop with OpenMP

To parallelize a loop...

```
!$OMP PARALLEL DO
DO i = 1, 1000
    !... your code here...



END DO
!$OMP END PARALLEL DO
```

Wait… but… how?!

Example: Parallelizing a loop with OpenMP

To parallelize a loop...

```
!$OMP PARALLEL DO
DO i = 1, 1000
      !... your code here...



END DO
!$OMP END PARALLEL DO
```

OpenMP is prepared to understand that the values of “i” will be distributed among threads.

Different values of “i” will automatically go to different threads.

How many threads will be created? Can we change that number? What problems do you foresee? Questions?

Example: Parallelizing a loop with OpenMP

To parallelize a loop… for matrix-vector multiplication: Ax = y

```
!$OMP PARALLEL DO
DO i = 1, 1000
    y(i) = 0
    DO j = 1, 1000
        y(i) = y(i) + A(i,j)*x(j)
    END DO
END DO
!$OMP END PARALLEL DO
```

Do you see any problems?

So... how do we use this in practice? (biased again!)

Distributed memory: MPI ("Message Passing Interface")

- It's a standard, of which there are several implementations: MPICH2, OpenMPI, MVAPICH, ...
- It's all about sending and receiving messages between **ranks**, and there are various ways to do that.
- In practice, multiple instances of your code are run at the same time, and each has its own **rank** identifier.

So... how do we use this in practice? (biased again!)

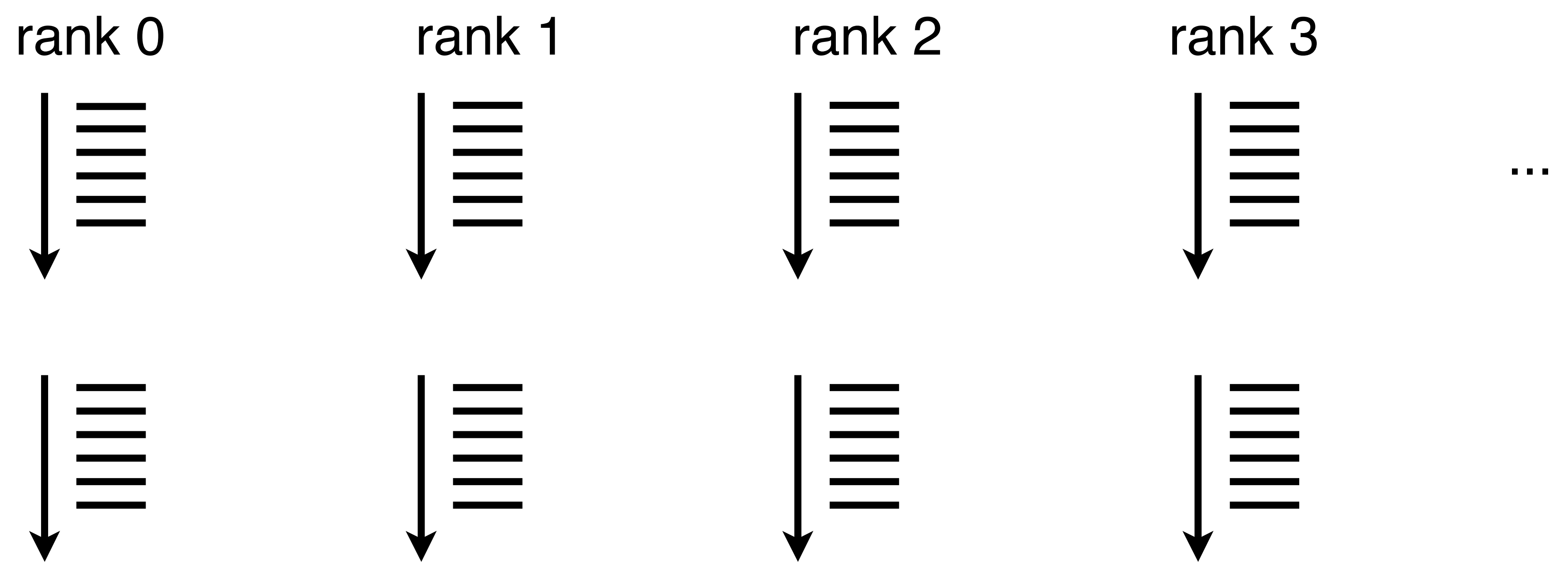
Distributed memory: MPI ("Message Passing Interface")

- It's a standard, of which there are several implementations: MPICH2, OpenMPI, MVAPICH, ...

- It's all about sending and receiving messages between **ranks**, and there are various ways to do that.

- In practice, multiple instances of your code are run at the same time, and each has its own **rank** identifier.

Your code execution looks like this:



So... how do we use this in practice? (biased again!)

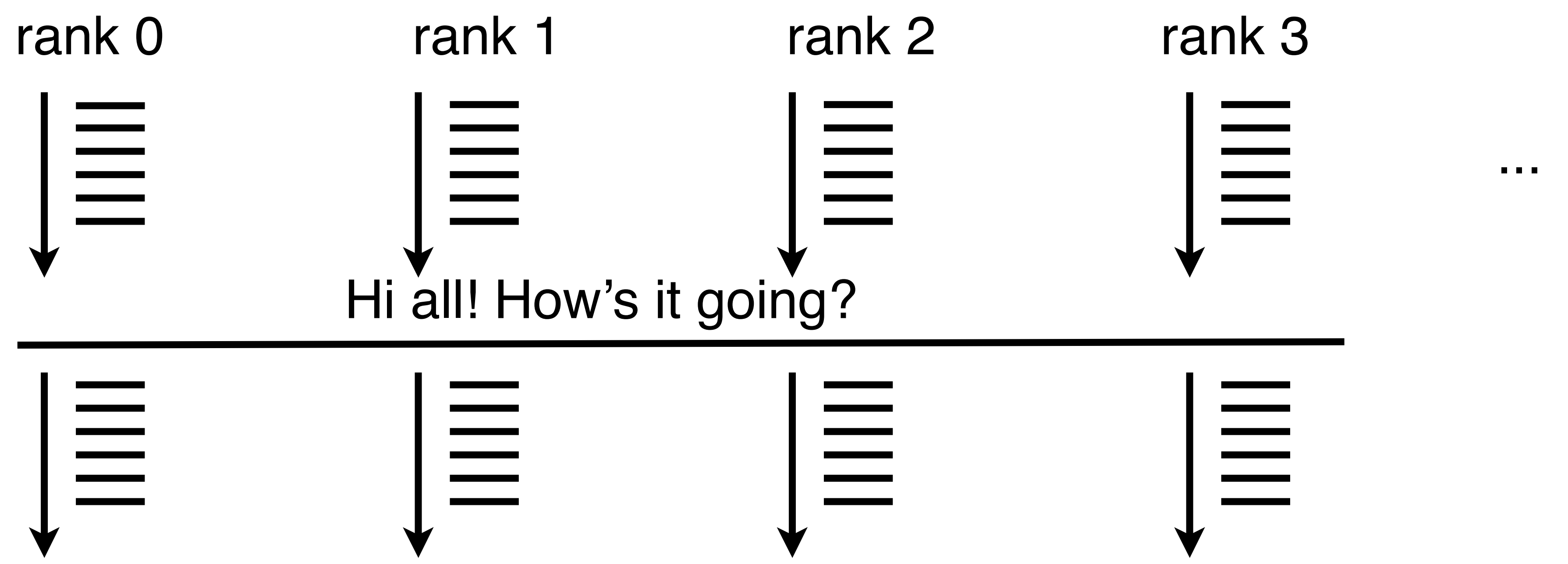
Distributed memory: MPI ("Message Passing Interface")

- It's a standard, of which there are several implementations:
  MPICH2, OpenMPI, MVAPICH, ...

- It's all about sending and receiving messages between **ranks**, and there are various ways to do that.

- In practice, multiple instances of your code are run at the same time, and each has its own **rank** identifier.

Your code execution looks like this:



So... how do we use this in practice? (biased again!)

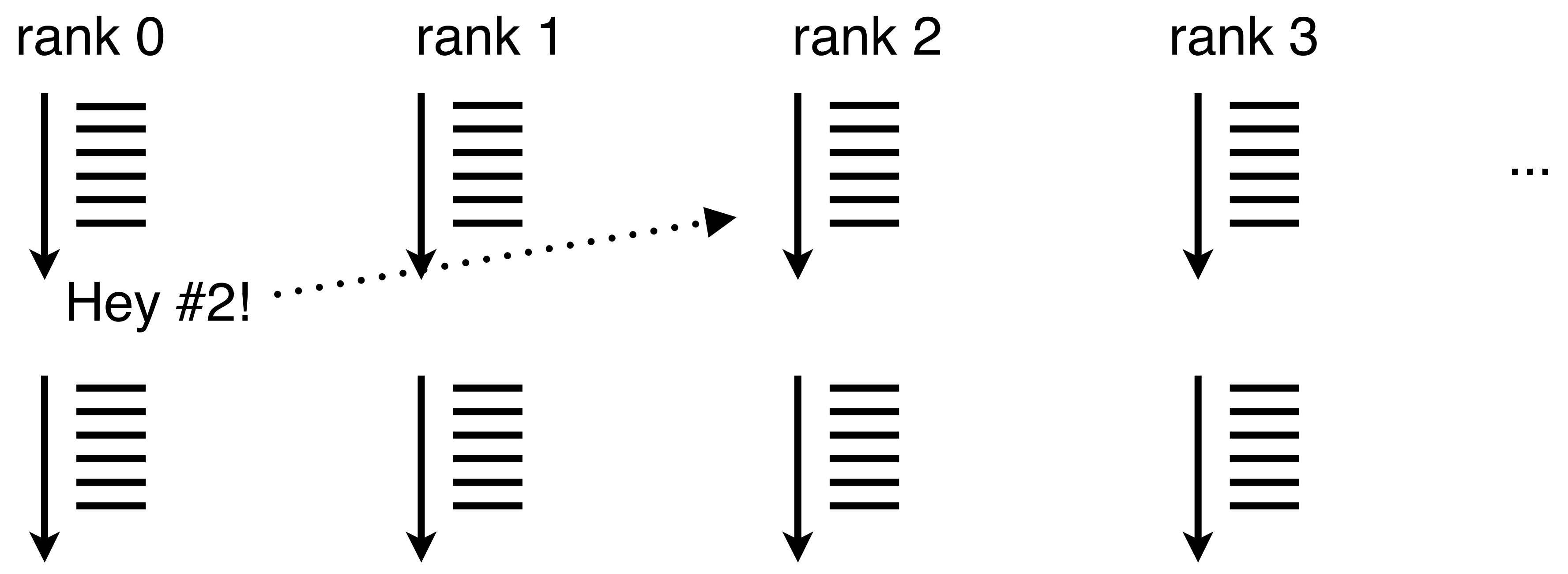
Distributed memory: MPI ("Message Passing Interface")

- It's a standard, of which there are several implementations: MPICH2, OpenMPI, MVAPICH, ...

- It's all about sending and receiving messages between **ranks**, and there are various ways to do that.

- In practice, multiple instances of your code are run at the same time, and each has its own **rank** identifier.

Your code execution looks like this:



Questions?

Quick comments on GPUs

- GPUs are programmed with languages like CUDA or OpenCL.

- They can be very useful for many applications (speedups can be substantial (factors of 10 or more, depending on the problem).

- They offer a hierarchy of memories, some small and fast, some larger but slower.

- Therefore GPUs are somewhat more challenging to work with, but not much. However, it does take some work to get good performance from them once you get your code going.

Quick comments on GPUs

- GPUs are programmed with languages like CUDA or OpenCL.

- They can be very useful for many applications (speedups can be substantial (factors of 10 or more, depending on the problem).

- They offer a hierarchy of memories, some small and fast, some larger but slower.

- Therefore GPUs are somewhat more challenging to work with, but not much. However, it does take some work to get good performance from them once you get your code going.

Yes, it is possible to combine MPI with OpenMP and GPUs. That's how you program for the big machines...

Tutorial time!

Pre-tutorial…

- Use
```
cat /proc/cpuinfo
```
to find out the number and specs of your processors.

- Use
```
top
```
on a separate terminal window to monitor the creation of processes.
Each thread will open up a new process.

- Compile with
```
gfortran -fopenmp defs.f90 example[#].f90 -o a.out
```

- Run with
```
./a.out
```

Extra stuff

Example I : Using OpenMP with Fortran 90

To parallelize a loop...

```
!$OMP PARALLEL DO
DO i = 1, 1000
        !... your code here...

END DO
!$OMP END PARALLEL DO
```

Example I : Using OpenMP with Fortran 90

To parallelize a loop...

```
!$OMP PARALLEL DO
DO i = 1, 1000
      !... your code here...

END DO
!$OMP END PARALLEL DO
```

To parallelize a chunk of code (i.e. create threads)...

```
!$OMP PARALLEL PRIVATE ([your thread-private variables])

 !... your code here...



!$OMP END PARALLEL
```

Example I : Using OpenMP with Fortran 90

To get the thread identifier:

```
USE omp_lib   ! somewhere at the top

!$OMP PARALLEL

    id = OMP_GET_THREAD_NUM()


!$OMP END PARALLEL
```

Example I : Using OpenMP with Fortran 90

To get the thread identifier:

```
USE omp_lib   ! somewhere at the top

!$OMP PARALLEL

     id = OMP_GET_THREAD_NUM()


!$OMP END PARALLEL
```

To set the number of threads:

```
USE omp_lib   ! somewhere at the top
   CALL OMP_SET_NUM_THREADS(Num_of_threads)
```

THE END

…actually, one last thing:
google “Amdahl’s law”

…and happy parallel
programming!