MPI is used extensively out there in the big world of Real Science
MPI is used extensively out there in the big world of Real Science

It is very well suited for when there is so much computation needed that the overhead of a bunch of messages is well worth paying
MPI is used extensively out there in the big world of Real Science.

It is very well suited for when there is so much computation needed that the overhead of a bunch of messages is well worth paying.

The large (100k core) clusters will be running jobs using MPI.
MPI

MPI is used extensively out there in the big world of Real Science.

It is very well suited for when there is so much computation needed that the overhead of a bunch of messages is well worth paying.

The large (100k core) clusters will be running jobs using MPI.

MPI scales very well to large systems.
And, of course, you can mix shared and distributed memory: running OpenMP tasks communicating across nodes via MPI
And, of course, you can mix shared and distributed memory: running OpenMP tasks communicating across nodes via MPI

Don’t use OpenMP in the coursework: that should be pure MPI
And, of course, you can mix shared and distributed memory: running OpenMP tasks communicating across nodes via MPI.

Don’t use OpenMP in the coursework: that should be pure MPI.

MPI requires you to make sure all your function calls are coordinated across the processes: every processor must call the appropriate matching functions at the appropriate times.
And, of course, you can mix shared and distributed memory: running OpenMP tasks communicating across nodes via MPI

Don’t use OpenMP in the coursework: that should be pure MPI

MPI requires you to make sure all your function calls are coordinated across the processes: every processor must call the appropriate matching functions at the appropriate times

This the programmer’s problem: it’s a bug if you get it wrong
For example, you can still easily deadlock. Suppose A and B wish to exchange messages:

A
MPI_Recv(...);
MPI_Send(...);

B
MPI_Recv(...);
MPI_Send(...);
For example, you can still easily deadlock. Suppose A and B wish to exchange messages:

**A**

```c
MPI_Recv(...);
MPI_Send(....);
```

**B**

```c
MPI_Recv(...);
MPI_Send(....);
```

This is slightly more obvious when it happens since MPI is SPMD and has a single program source.
For example, you can still easily deadlock. Suppose A and B wish to exchange messages:

A
MPI_Recv(...);
MPI_Send(...);

B
MPI_Recv(...);
MPI_Send(...);

This is slightly more obvious when it happens since MPI is SPMD and has a single program source.

Careful use of message tags helps structuring.
For example, you can still easily deadlock. Suppose A and B wish to exchange messages:

A
MPI_Recv(...);
MPI_Send(...);

B
MPI_Recv(...);
MPI_Send(...);

This is slightly more obvious when it happens since MPI is SPMD and has a single program source.

Careful use of message tags helps structuring.

As is common, MPI provides easy mechanism but no analysis.
In fact, for this case, MPI provides `MPI_Sendrecv` which combines a send with a receive that is guaranteed not to deadlock.
In fact, for this case, MPI provides MPI_Sendrecv which combines a send with a receive that is guaranteed not to deadlock

A
MPI_Sendrecv(...);

B
MPI_Sendrecv(...);
In fact, for this case, MPI provides MPI_Sendrecv which combines a send with a receive that is guaranteed not to deadlock

A
MPI_Sendrecv(...);

B
MPI_Sendrecv(...);

This function is recommended in cases of swapping data
In fact, for this case, MPI provides `MPI_Sendrecv` which combines a send with a receive that is guaranteed not to deadlock.

```
A
 MPI_Sendrecv(...);

B
 MPI_Sendrecv(...);
```

This function is recommended in cases of swapping data.

And it can pair any send and receive; is not limited to simple swapping between two processes. For example, A sends to B but receives from C; while B sends to C but receives from A; etc.
Using MPI requires careful thought about messages to get the maximum efficiency out of the system.
Using MPI requires careful thought about messages to get the maximum efficiency out of the system.

For example, we might be able to overcome message latency by judicious use of non-blocking sends and receives.
Using MPI requires careful thought about messages to get the maximum efficiency out of the system.

For example, we might be able to overcome message latency by judicious use of non-blocking sends and receives.

Rather than waiting for a receive to complete, we carry on working on some other part of the computation: later, when the receive has completed, we can go back to that part of the computation.
This requires careful programming, but can give good results
This requires careful programming, but can give good results

Sometimes not
This requires careful programming, but can give good results

Sometimes not

In general (not just distributed computing), overlapping communication and computation is a good thing to do
This requires careful programming, but can give good results

Sometimes not

In general (not just distributed computing), overlapping communication and computation is a good thing to do

But hard to program and easy to make errors
This requires careful programming, but can give good results
Sometimes not

In general (not just distributed computing), overlapping communication and computation is a good thing to do

But hard to program and easy to make errors

Exercise. You wish to make a cup of tea and a sandwich. Do you

(a) make the sandwich then boil the kettle; or
(b) start boiling the kettle then make the sandwich?
MPI

Also:

- messaging has a high overhead, so MPI only really works well on very large programs
Also:

- messaging has a high overhead, so MPI only really works well on very large programs
- it is hard to program effectively: simple programs are easy to write, but efficient programs usually need experienced programmers
MPI

Also:

- messaging has a high overhead, so MPI only really works well on very large programs
- it is hard to program effectively: simple programs are easy to write, but efficient programs usually need experienced programmers
- it is not naturally dynamic: the number of processors is effectively fixed and cannot vary during the execution of the program. This excludes efficient execution of some kinds of program (later version of MPI do include MPI_Comm_spawn but it’s not easy to use)
MPI has succeeded for many reasons

- An open standard, inviting several competing implementations
- Thus implementations tend to be optimised and efficient
- MPI is simple in concept, so straightforward to program (not necessarily easy)
- MPI is flexible as it contains lots of kinds of communication
- MPI is supported by many languages and environments
- MPI scales well to very large problems

The MPI standard is still being developed and updated
MPI

MPI has succeeded for many reasons

- An open standard, inviting several competing implementations
MPI

MPI has succeeded for many reasons

- An open standard, inviting several competing implementations
- Thus implementations tend to be optimised and efficient

The MPI standard is still being developed and updated
MPI

MPI has succeeded for many reasons

- An open standard, inviting several competing implementations
- Thus implementations tend to be optimised and efficient
- MPI is simple in concept, so straightforward to program (not necessarily easy to program...)
MPI has succeeded for many reasons

- An open standard, inviting several competing implementations
- Thus implementations tend to be optimised and efficient
- MPI is simple in concept, so straightforward to program (not necessarily easy to program . . .)
- MPI is flexible as it contains lots of kinds of communication

The MPI standard is still being developed and updated
MPI has succeeded for many reasons

- An open standard, inviting several competing implementations
- Thus implementations tend to be optimised and efficient
- MPI is simple in concept, so straightforward to program (not necessarily easy to program . . .)
- MPI is flexible as it contains lots of kinds of communication
- MPI is supported by many languages and environments
MPI has succeeded for many reasons

- An open standard, inviting several competing implementations
- Thus implementations tend to be optimised and efficient
- MPI is simple in concept, so straightforward to program (not necessarily easy to program...)
- MPI is flexible as it contains lots of kinds of communication
- MPI is supported by many languages and environments
- MPI scales well to very large problems
MPI has succeeded for many reasons

- An open standard, inviting several competing implementations
- Thus implementations tend to be optimised and efficient
- MPI is simple in concept, so straightforward to program (not necessarily easy to program...)
- MPI is flexible as it contains lots of kinds of communication
- MPI is supported by many languages and environments
- MPI scales well to very large problems

The MPI standard is still being developed and updated
Exercise. Read about UPC, a (not popular) alternative to MPI, that presents a virtual shared NUMA architecture.