You have a problem you wish to solve faster. What do you do?
You have a problem you wish to solve faster. What do you do?

1. You think hard and devise a better solution
You have a problem you wish to solve faster. What do you do?

1. You think hard and devise a better solution

Clearly this is a stupid thing to do. Programmers are much too lazy to do this
You have a problem you wish to solve faster. What do you do?

1. You think hard and devise a better solution

Clearly this is a stupid thing to do. Programmers are much too lazy to do this

2. You get a faster processor
You have a problem you wish to solve faster. What do you do?

1. You think hard and devise a better solution

Clearly this is a stupid thing to do. Programmers are much too lazy to do this

2. You get a faster processor

Better. This used to work, but not any more: processors have pretty much levelled off at around the 3-5GHz mark and don’t seem to be getting faster
Background

3. You get a multicore machine and run the problem in parallel
3. You get a multicore machine and run the problem in parallel

This *must* be the solution!
3. You get a multicore machine and run the problem in parallel

This *must* be the solution!

Isn’t it?
3. You get a multicore machine and run the problem in parallel

This *must* be the solution!

Isn’t it?

One purpose of this Unit is to make you realise this is actually the *hardest* way of doing it!
3. You get a multicore machine and run the problem in parallel

This *must* be the solution!

Isn’t it?

One purpose of this Unit is to make you realise this is actually the *hardest* way of doing it!

In reality, No. 1 is best, then No. 2, lastly No. 3
Background

Consider the following:

- it takes one person ten weeks to build one house
Consider the following:

- it takes one person ten weeks to build one house
- it takes ten people one week to build one house

Why is the last so implausible?
Background

Consider the following:

- it takes one person ten weeks to build one house
- it takes ten people one week to build one house
- it takes 100 people one-tenth of a week to build one house

Why is the last so implausible?
Background

Consider the following:

- it takes one person ten weeks to build one house
- it takes ten people one week to build one house
- it takes 100 people one-tenth of a week to build one house

Why is the last so implausible?
Background

When there are 100 people running about they will get in each others’ way; fight over limited resources like bricks; some will have to sit and twiddle their thumbs while they wait for others to finish: you can’t plumb a bathroom until the bathroom has been built.
When there are 100 people running about they will get in each others’ way; fight over limited resources like bricks; some will have to sit and twiddle their thumbs while they wait for others to finish: you can’t plumb a bathroom until the bathroom has been built

And so on
When there are 100 people running about they will get in each others’ way; fight over limited resources like bricks; some will have to sit and twiddle their thumbs while they wait for others to finish: you can’t plumb a bathroom until the bathroom has been built

And so on

Simply adding more people won’t get it done faster
When there are 100 people running about they will get in each others’ way; fight over limited resources like bricks; some will have to sit and twiddle their thumbs while they wait for others to finish: you can’t plumb a bathroom until the bathroom has been built

And so on

Simply adding more people won’t get it done faster

Sometimes adding more people will make it go slower as they get in each others’ way
Background

But we can scale in a different way:
Background

But we can scale in a different way:

- it takes one person ten weeks to build one house
But we can scale in a different way:

- it takes one person ten weeks to build one house
- it takes ten people ten weeks to build ten houses
But we can scale in a different way:

- it takes one person ten weeks to build one house
- it takes ten people ten weeks to build ten houses
- it takes one person 100 weeks to build ten houses
But we can scale in a different way:

- it takes one person ten weeks to build one house
- it takes ten people ten weeks to build ten houses
- it takes one person 100 weeks to build ten houses
- it takes ten people 100 weeks to build 100 houses
Background

But we can scale in a different way:

- it takes one person ten weeks to build one house
- it takes ten people ten weeks to build ten houses
- it takes one person 100 weeks to build ten houses
- it takes ten people 100 weeks to build 100 houses

This is much more believable
Most people think parallel computing is about making things go faster
Most people think parallel computing is about making things go faster.

Up to a point, but they will soon be disappointed.
Most people think parallel computing is about making things go faster

Up to a point, but they will soon be disappointed

Much more likely to succeed is to make things larger
Most people think parallel computing is about making things go *faster*

Up to a point, but they will soon be disappointed

Much more likely to succeed is to make things *larger*

This scales much better
The first is process parallelism, also called task parallelism
The first is *process* parallelism, also called *task* parallelism.

The second is *data parallelism*.
Background

The first is *process* parallelism, also called *task* parallelism.

The second is *data parallelism*.

Two very different ways of getting more in a given amount of time.
Background

You all have had the situation where someone tries to help you do something and it’s ended up taking *longer*
You all have had the situation where someone tries to help you do something and it’s ended up taking \textit{longer}

There is the basic time it takes to solve the problem: then there are substantial overheads in the coordination of the parts of the solution.
You all have had the situation where someone tries to help you do something and it’s ended up taking longer.

There is the basic time it takes to solve the problem: then there are substantial overheads in the coordination of the parts of the solution.

The overheads can easily be larger than the problem itself.
You all have had the situation where someone tries to help you do something and it’s ended up taking longer.

There is the basic time it takes to solve the problem: then there are substantial overheads in the coordination of the parts of the solution.

The overheads can easily be larger than the problem itself.

This is the reality of parallel computing.
You all have had the situation where someone tries to help you do something and it’s ended up taking longer.

There is the basic time it takes to solve the problem: then there are substantial overheads in the coordination of the parts of the solution.

The overheads can easily be larger than the problem itself.

This is the reality of parallel computing.

Often a parallel version of a small problem will be slower than the sequential version.
You all have had the situation where someone tries to help you do something and it’s ended up taking longer.

There is the basic time it takes to solve the problem: then there are substantial overheads in the coordination of the parts of the solution.

The overheads can easily be larger than the problem itself.

This is the reality of parallel computing.

Often a parallel version of a small problem will be slower than the sequential version.

Only when the problem is made large enough to overcome the overheads will it become faster than doing it sequentially.
So cost (the number of cpu cycles) of a parallel computation =
cost of computation + cost of management of parallelism
Background

So cost (the number of cpu cycles) of a parallel computation = cost of computation + cost of management of parallelism

Another huge issue is that people have enough difficulties with programming sequential machines
So cost (the number of cpu cycles) of a parallel computation = cost of computation + cost of management of parallelism

Another huge issue is that people have enough difficulties with programming sequential machines

Some would say that sequential programming is not yet a “solved” problem
So cost (the number of cpu cycles) of a parallel computation =
cost of computation + cost of management of parallelism

Another huge issue is that people have enough difficulties with
programming sequential machines

Some would say that sequential programming is not yet a
“solved” problem

Parallel programming is *much* harder
Background

So cost (the number of cpu cycles) of a parallel computation =
\[ \text{cost of computation} + \text{cost of management of parallelism} \]

Another huge issue is that people have enough difficulties with
programming sequential machines

Some would say that sequential programming is not yet a
“solved” problem

Parallel programming is *much* harder

If you think you understand parallel programing, you clearly
don’t
You have all the issues of sequential programs **plus** lots more
You have all the issues of sequential programs plus lots more

And they are issues that many programmers have difficulty even understanding
You have all the issues of sequential programs plus lots more.

And they are issues that many programmers have difficulty even understanding.

Particularly as they have been trained to program for sequential machines and have habits and assumptions that are simply invalid for parallel machines.
Background

Have I convinced you that parallel programming is difficult yet?
Have I convinced you that parallel programming is difficult yet?
Well, it’s worse than you can imagine!
You will see the terms *parallel* and *concurrent*, often being used interchangeably.
You will see the terms *parallel* and *concurrent*, often being used interchangeably.

It is important to make a distinction between the two.
You will see the terms *parallel* and *concurrent*, often being used interchangeably.

It is important to make a distinction between the two.

**concurrent** means your computation is in separately executable parts.
You will see the terms *parallel* and *concurrent*, often being used interchangeably.

It is important to make a distinction between the two:

- **concurrent** means your computation is in separately executable parts.
- **parallel** means those parts are being executed *at the same time*.
You will see the terms *parallel* and *concurrent*, often being used interchangeably.

It is important to make a distinction between the two:

- **concurrent** means your computation is in separately executable parts.
- **parallel** means those parts are being executed *at the same time*.

Concurrency is about structure, parallelism is about execution.
So, “concurrent” means in parts, and those parts may or may not be running simultaneously (e.g., being scheduled one by one on a single core CPU).

Exercise. Read about async programming as an example of non-parallel concurrency. Also futures, promises, coroutines, generators and others.
So, “concurrent” means in parts, and those parts may or may not be running simultaneously (e.g., being scheduled one by one on a single core CPU)

And “parallel” when we are explicitly talking about stuff running at the same time on multiple pieces of hardware
Background

So, “concurrent” means in parts, and those parts may or may not be running simultaneously (e.g., being scheduled one by one on a single core CPU)

And “parallel” when we are explicitly talking about stuff running at the same time on multiple pieces of hardware

Concurrency is about dealing with lots of things at once.
Parallelism is about doing lots of things at once.
Rob Pike
Background

So, “concurrent” means in parts, and those parts may or may not be running simultaneously (e.g., being scheduled one by one on a single core CPU)

And “parallel” when we are explicitly talking about stuff running at the same time on multiple pieces of hardware

-Concurrency is about dealing with lots of things at once.
Parallelism is about doing lots of things at once.
Rob Pike

Exercise. Read about async programming as an example of non-parallel concurrency. Also futures, promises, coroutines, generators and others
In contrast, you might hear of *serial* and *sequential* both being used to describe non-concurrent/non-parallel systems.
In contrast, you might hear of *serial* and *
sequential* both being used to describe non-concurrent/non-parallel systems.

They mean the same thing.