Parallel and its Problems

Doing it Faster in Parallel

- Lets read an 8 page book
- 1 person starts with the whole book
- It takes 1 person 3 minutes to read 1 page
- It takes 1 minute rip a page out and hand it to another person
- What is the most amount of people we can use, beyond which adding further people will not speed up the reading?

Doing it Faster in Parallel

	1	2	3	4	5	6	7	8	9	10	11
Person 1	Send	Send	Send	Send	Send	Send	Send	Send			
Person 2	Receive	Read									
Person 3		Receive	Read								
Person 4			Receive	Read							
Person 5				Receive	Read						
Person 6					Receive	Read					
Person 7						Receive	Read				
Person 8							Receive	Read			
Person 9								Receive	Read		

Doing it Faster in Parallel

	1	2	3	4	5	6	7	8	9	10	11
Person 1	Send	Send	Send	Send	Send	Send	Send	Send			
Person 2	Receive	Read			Receive	Read					
Person 3		Receive	Read			Receive	Read				
Person 4			Receive	Read			Receive	Read			
Person 5				Receive	Read			Receive	Read		
Person 6											
Person 7											
Person 8											
Person 9											

Doing it Faster in Parallel

	1	2	3	4	5	6	7	8	9	10	11
Person 1	Send	Send	Send	Send	Send	Send	Send	Read			
Person 2	Receive	Read			Receive	Read					
Person 3		Receive	Read			Receive	Read				
Person 4			Receive	Read			Receive	Read			
Person 5				Receive	Read						
Person 6											
Person 7											
Person 8											
Person 9											

Doing it Faster in Parallel

- Lets write an 8 page book
- It takes 1 person 3 minutes to write 1 page
- It takes 1 minute copy a page into the finished book
- What is the most amount of people we can use, beyond which adding further people will not speed up the writing?

Doing it Faster in Parallel

- Moore’s Law is not really true any more.
- If we want faster systems we need to use parallel processing.
- But we need to think a little bit about how we do so.

Concurrency

- Concurrency means that two or more tasks are being undertaken and can each progress without depending on any other task.
- Not specific to computing.
- Doesn’t mean tasks are run at *literally* the same time however.
- For example, a single core PC running multiple programmes at once, a student undertaking multiple courses.

Parallelism

- Parallelism is when multiple tasks are split over more than one processor to be undertaken at *literally* the same time.
- Not specific to computing.
- For example, a multicore PC running multiple programmes at once, a car production line.

CSP

- Communicating Sequential Processes
- Formal language for concurrent and parallel programming
- Message passing
- Defines processes, channels (and several other things)

Processes

- A Process is a collection of sequential code that runs within a parallel system
- It might take some inputs, and might have some outputs.
- Each process will run independently of each other process

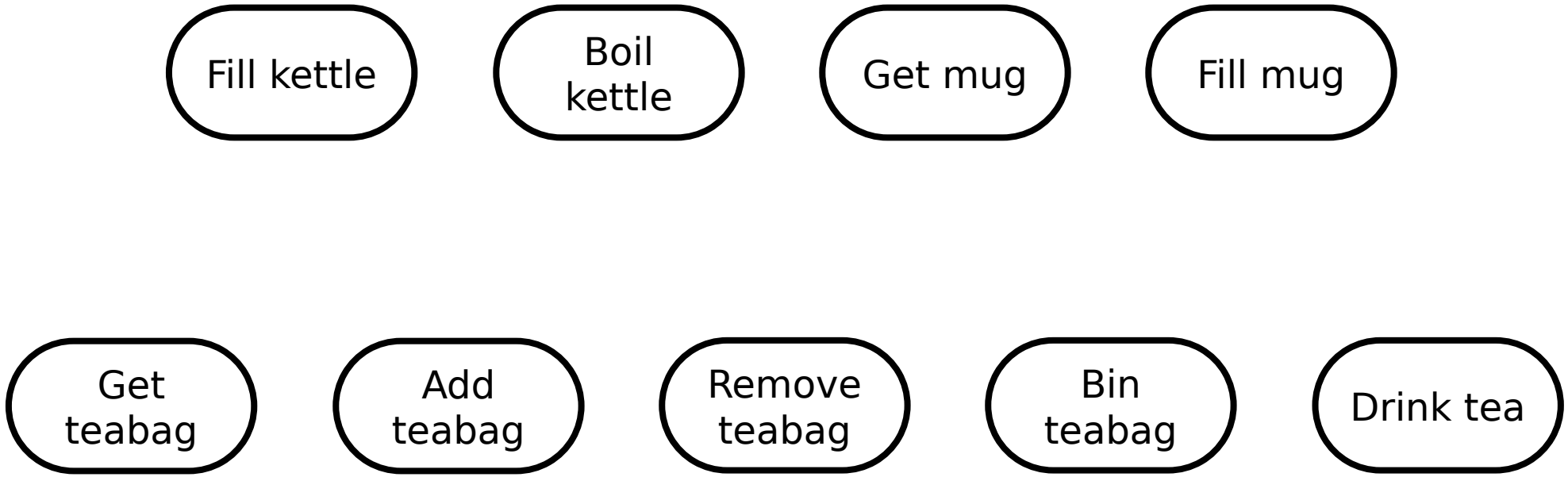
Channels

- Channels are used to communicate between processes
- Channels are one way, with an input and output end
- Multiple processes can connect to the same end (in PyCSP anyway)
- Communication is Synchronous, both processes need to be ready to read/write

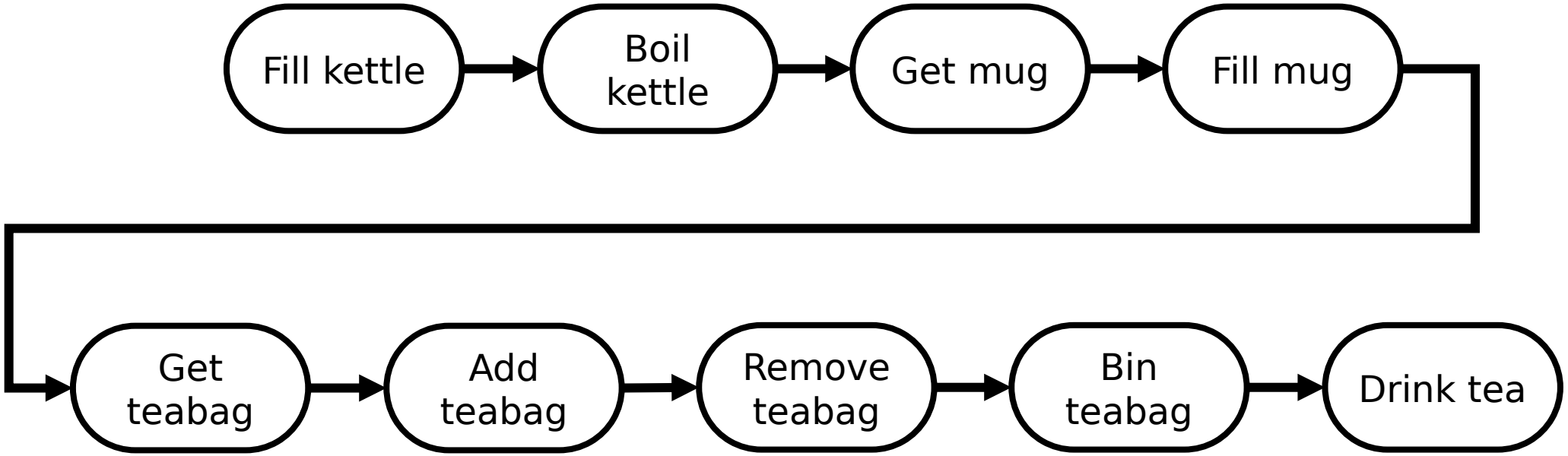
Drawing the procedure

- Parallel programs are difficult to reason about
- We **NEED** to draw diagrams of what we're going to do
- Diagrams should show all the different processes and how they are connected

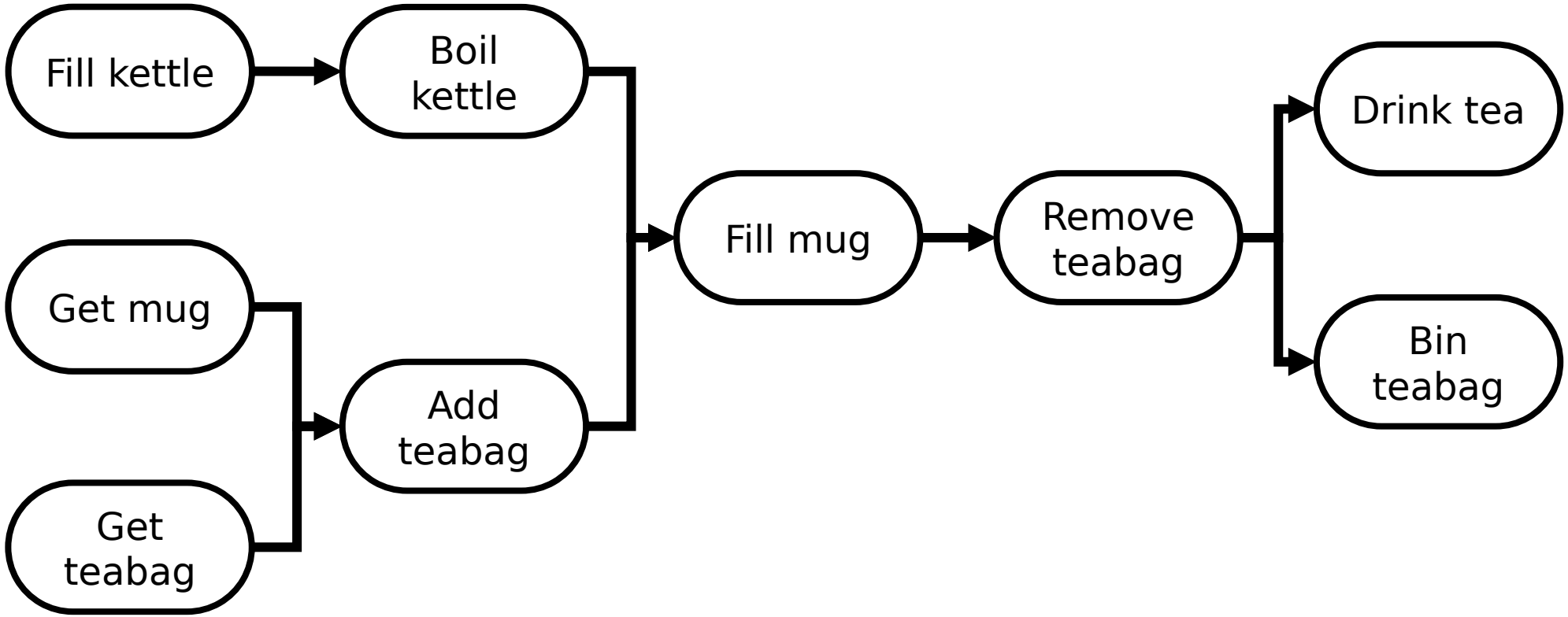
Drawing the procedure



Drawing the procedure



Drawing the procedure

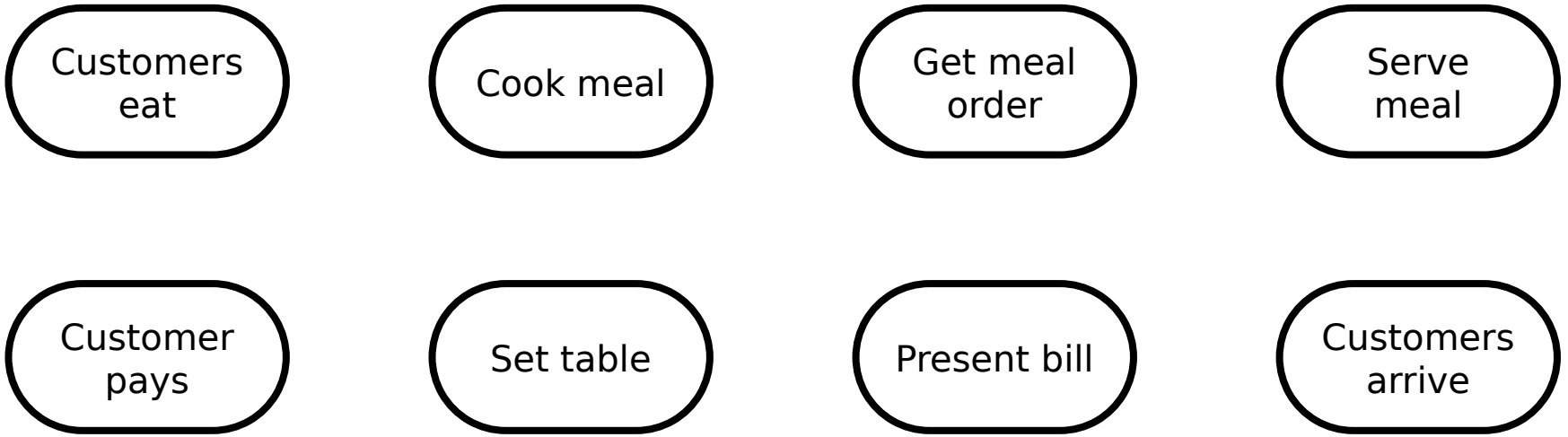


Drawing the procedure

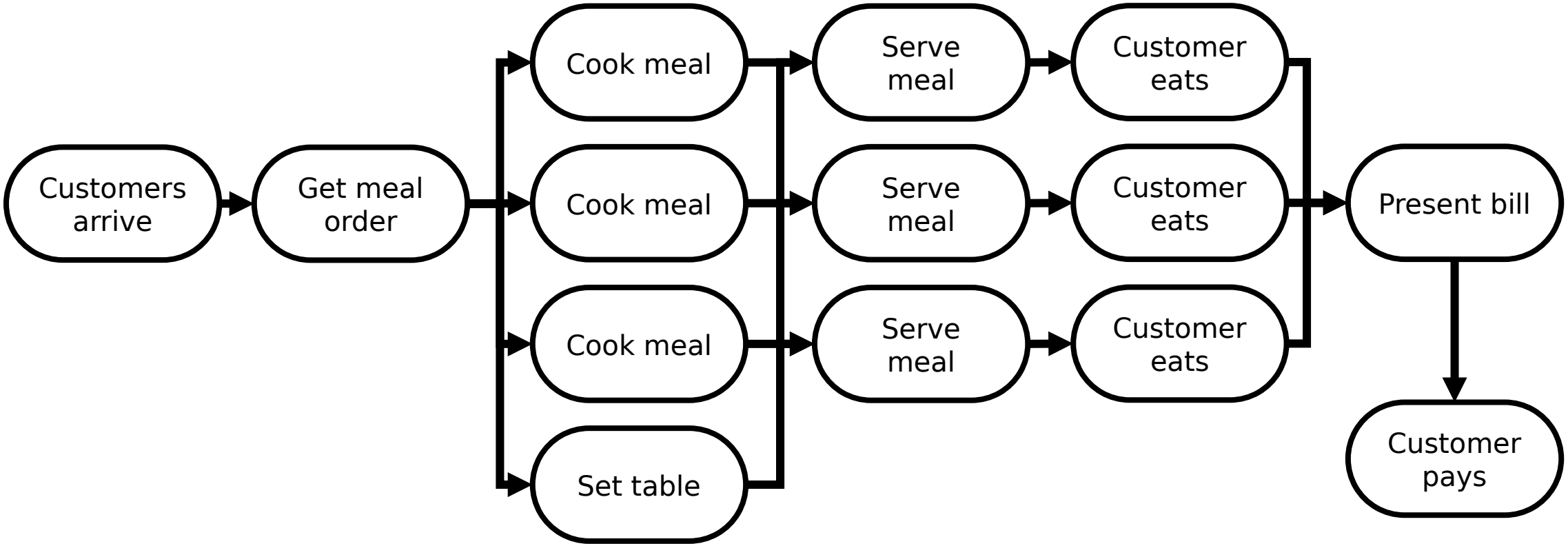
- Lets build a restaurant system to serve dinner for 3

Drawing the procedure

- Lets build a restaurant system to serve dinner for 3



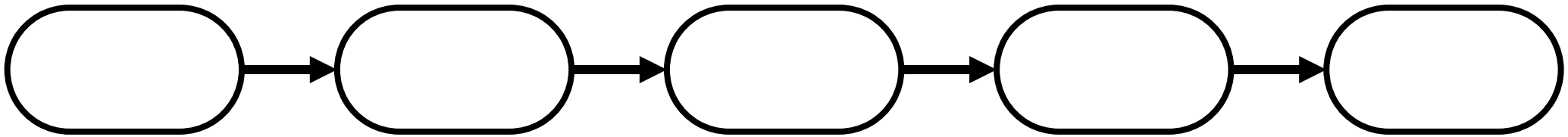
Drawing the procedure



Task Parallel

- Task Parallel is a model of parallelisation.
- Different processes (or tasks) are distributed across different processors
- These systems will typically have a pipeline structure
- For example, a production line

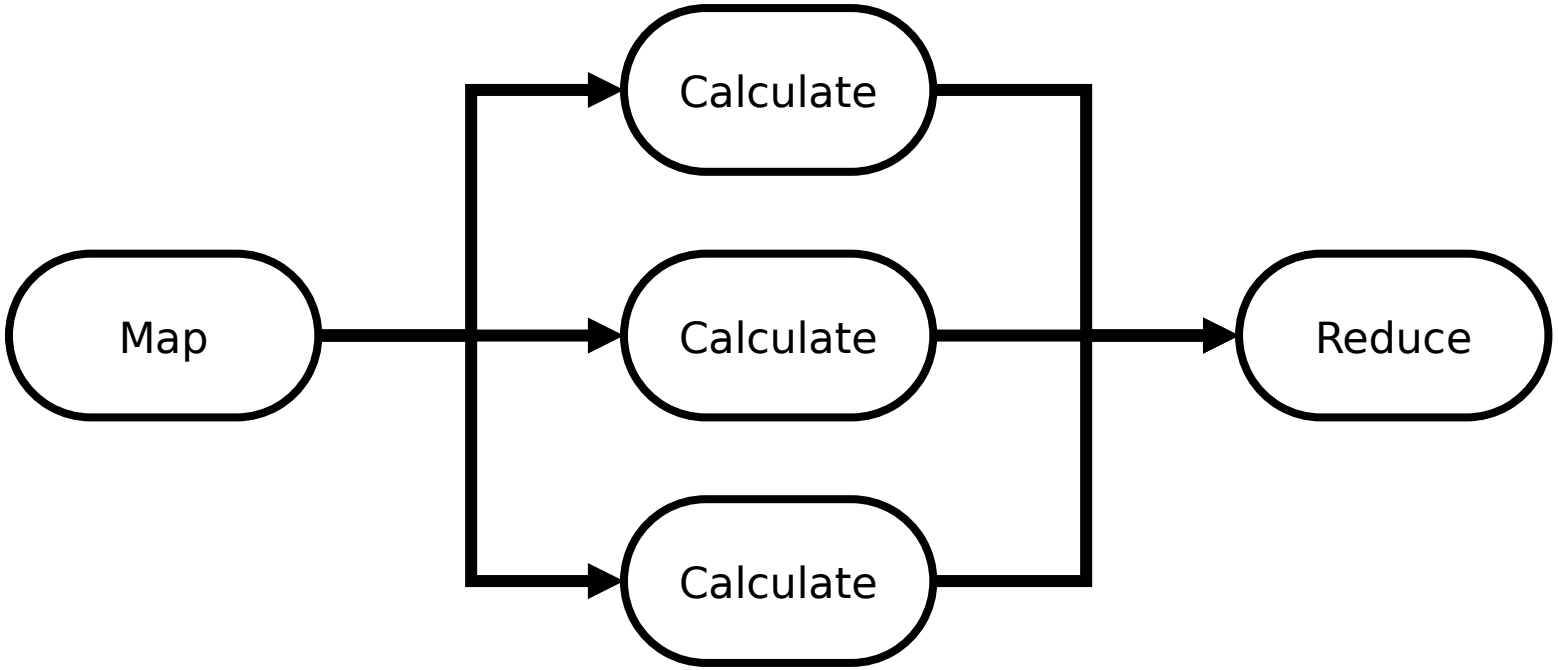
Task Parallel



Data Parallel

- Data Parallel is a model of parallelisation.
- A large chunk of data is broken up and split different processors all performing the same task
- These systems will typically have a parallel structure
- For example,

Data Parallel



Task and Data Parallel

- Data Parallelism is better suited to *embarrassingly parallel* problems. These are problems with no data dependency (Most data analysis).

- Task Parallelism is better suited to systems where the same process needs to be repeated on an ongoing basis (Hardware).

- Most systems will be a mix of the two, plus some old fashioned sequential code.