To Process or Not To Process

Compartmentalisation

- Compartmentalisation is the method of breaking a problem into smaller problems
- Within CSP this refers to breaking a sequential program down into numerous processes which can be scheduled concurrently or in parallel.
- So when do we do it?

Compartmentalisation

- Lets read an 8 page 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?

Compartmentalisation

	1	2	3	4	5	6	7	8	9	10
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					

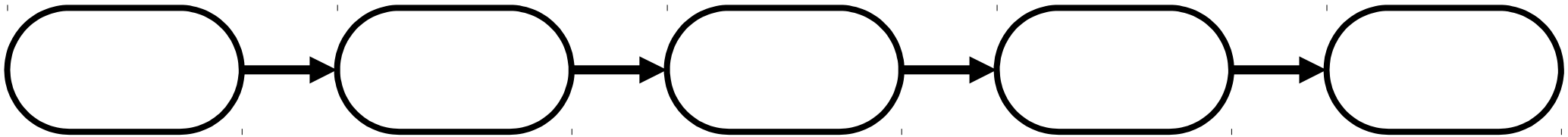
Compartmentalisation

- 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?

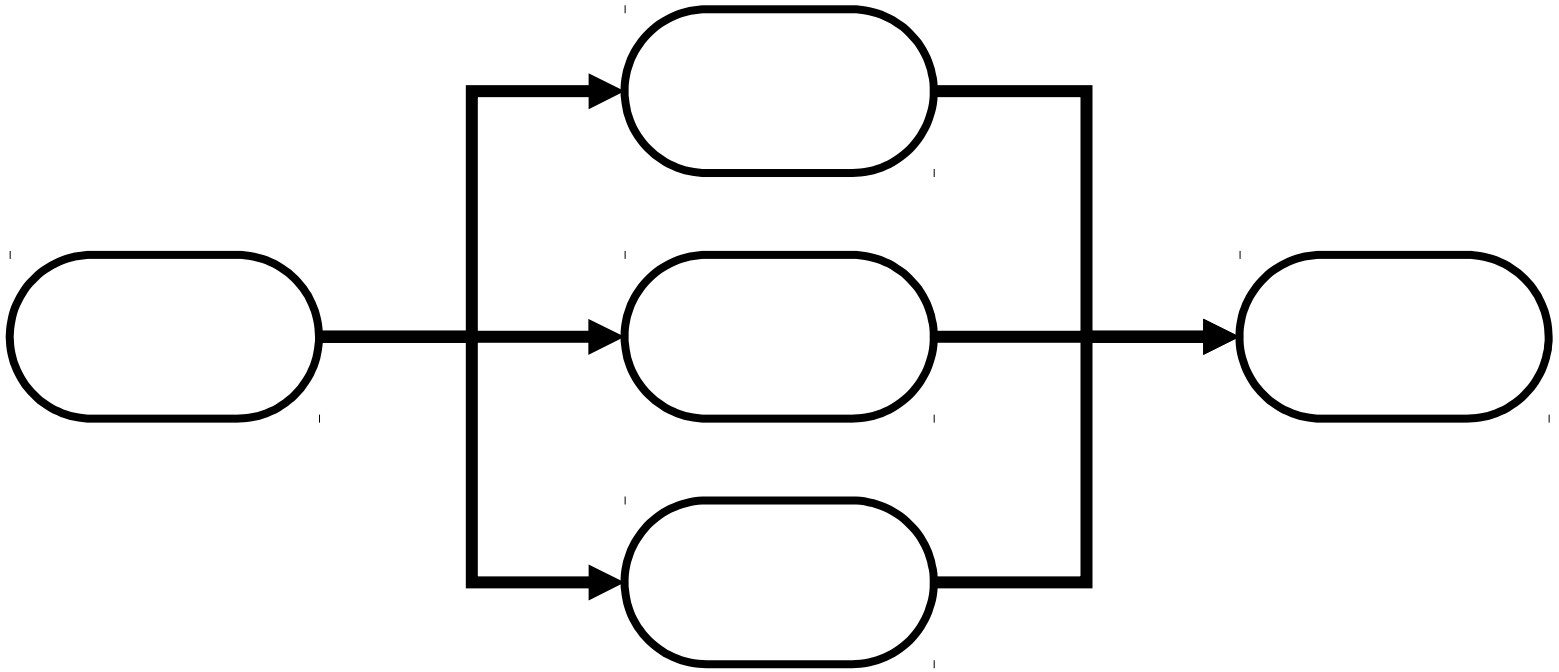
Compartmentalisation

- Typically you want the minimum amount of processes necessary to get the job done.
- But with too few processes you won’t be able to take advantage of available hardware
- All hardware will have a finite limit on the amount of processes that can be run in parallel
- But there is no limit on the amount that can be run concurrently

Task Parallel



Data Parallel



Contexts Switches

- Operating Systems allow for far more processes to be scheduled than can be physically run at once.
- Each process will be run for a short amount of time and then switched out *almost* instantly for another.
- This switch does take time, and the more processes we schedule at once the more often these switches will occur.
- More time switching means less time processing

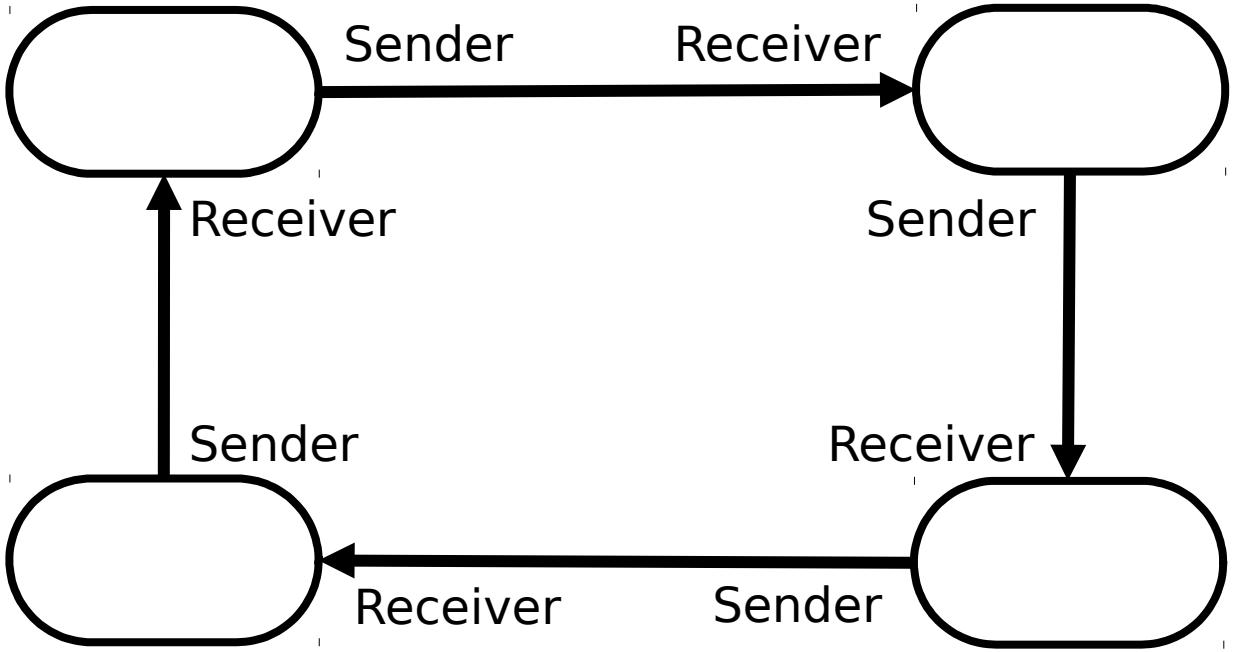
Contexts Switches

- But sometimes its worth putting up with lots of context switches.
- For instance continuously running even based systems such as robotics or IoT devices
- A good rule of thumb is that when using a data parallel approach, don’t use more processes that you have physical processors.
- As for task parallel, break up only what *needs* to be broken up
- Makes most sense across physical machines / processors

Asynchronous Communications

- So far all communications have been synchronous.
- We’ve gotten around this by building buffers.
- Why is this actually needed?

Asynchronous Communications



Asynchronous Communications

- This system may deadlock if all 4 processes try to send at once.
- This means the system has a message capacity of 4.
- A buffer process can increase this capacity.

Asynchronous Communications

- We could build a buffer straight into a channel

```
channel = Channel(buffer=10)
```

- This will read in values until the buffer is full. It will wait until there is space to read in additional values.

- It looks very, very handy. So why only introduce it now?