NTNU – Trondheim
Norwegian University of
Science and Technology

Flynn’s taxonomy, shared & distributed memory

Jan.Christian.Meyer@ntnu.no

Classifications of parallel computers

- We have surveyed how performance has been automatically extracted from sequential computers
- Today, we start with the *visibly* parallel designs
- Flynn's taxonomy is one way to sort them
- *Shared* and *distributed* memory designs make another way to sort them
- Important performance characteristics come out of how the parallel units are wired up to communicate with each other

Flynn's taxonomy

- According to Flynn, there are two things to deal with
  - Instructions
  - Data

  and we can have either 1 or many of each at a time
  - Single
  - Multiple
- This gives us 4 combinations:

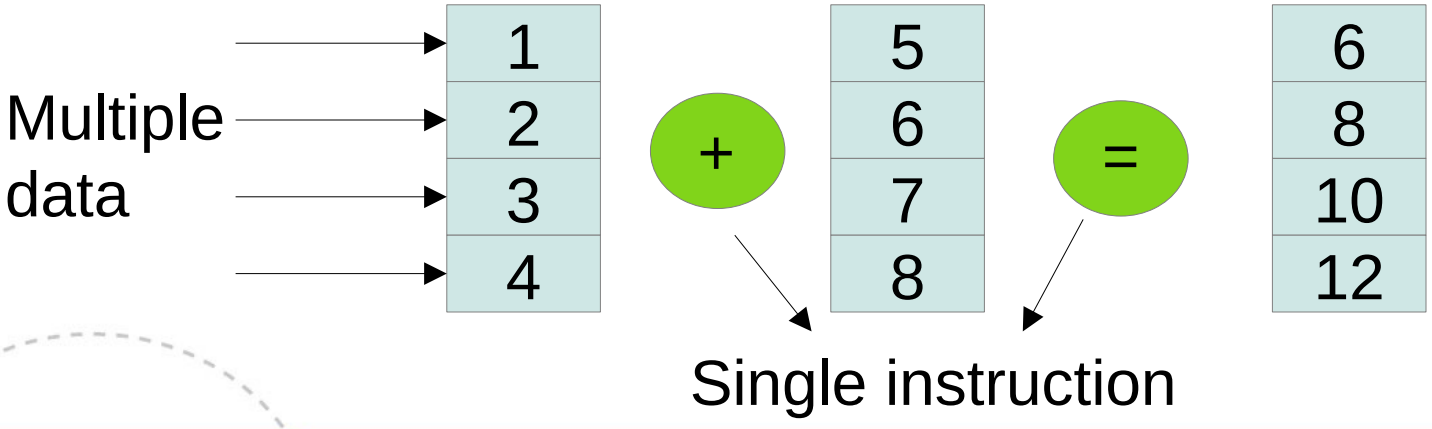
Instructions \ Data	Single	Multiple
Single	SISD	SIMD
Multiple	MISD	MIMD

SISD

- This is your vanilla-flavor regular von Neumann machine
- One instruction is run at a time
- One set of operands are affected
- Everything happens in sequence

SIMD

- This is a vector machine
- Single Instruction means we're only doing one thing at a time
- Multiple Data applies the same action to many sets of operands simultaneously



MISD

- This combination exists mostly because it's a valid combination of M and S in Flynn's system
- It would represent a machine that can load only one set of operands at a time, but apply lots of different instructions to them
  - That's ~~blooming useless~~ interesting
- You can make an argument that pipelines are a kind of MISD parallelism, but most references to MISD are purely of theoretical interest

MIMD

- With multiple instructions and multiple operands at the same time, we need fully independent processors
- Our programs will make them work towards the same goal, but they’re not synchronized by their construction
- This is the kind of parallelism we get with threads, processes, *etc.*

Flynn is kind of old-fashioned

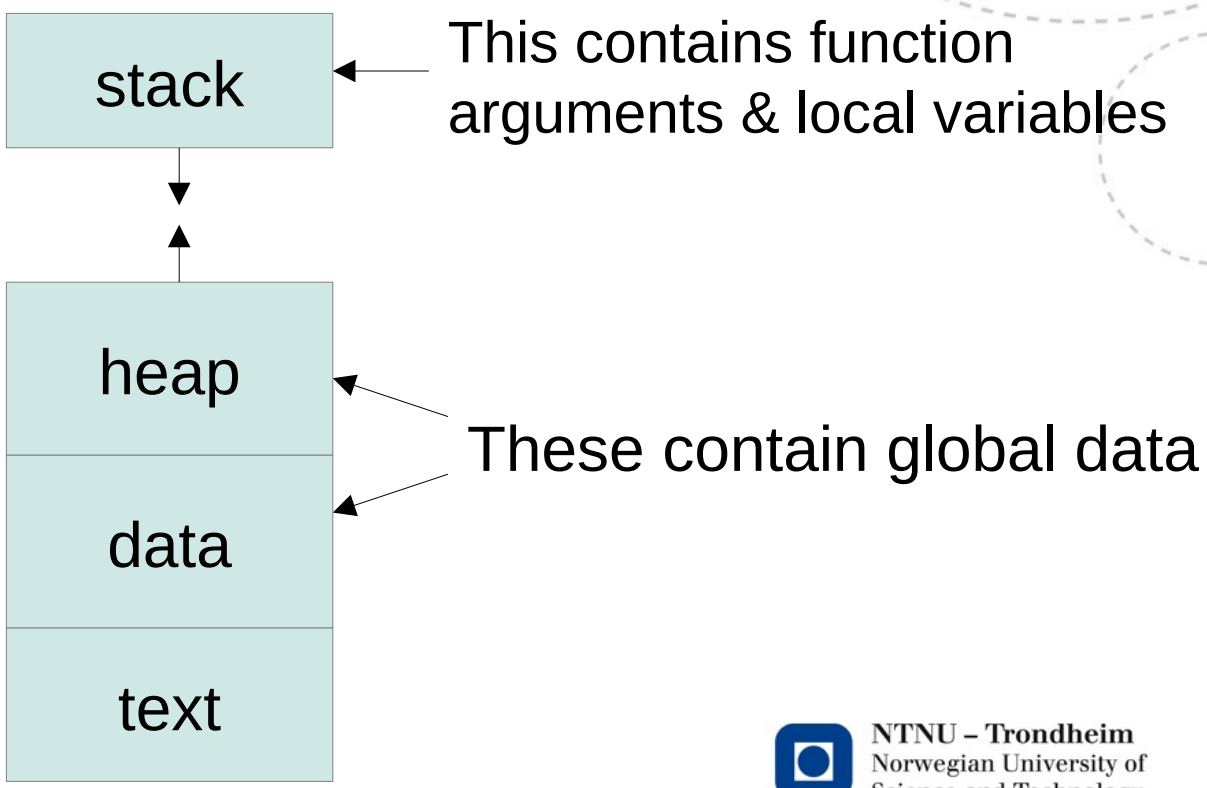
- This 4-way division of parallelism was originally meant to classify all parallel architectures
- In the years since it was introduced, we have
  - invented systems that don't fit neatly into it
  - developed a need for additional vocabulary that it doesn't cover
- Even though the terms aren't crystal clear descriptions of machinery, they still work in order to invoke an idea when we discuss particular systems
- They're so basic that everyone should know them

In tune with the times

- More pertinently, we can divide our parallel computers into *shared memory* and *distributed memory* variants
- The main difference is whether or not they can examine each others' values without asking
- *Shared memory* partitions the memory image of a process, *distributed memory* works with several processes

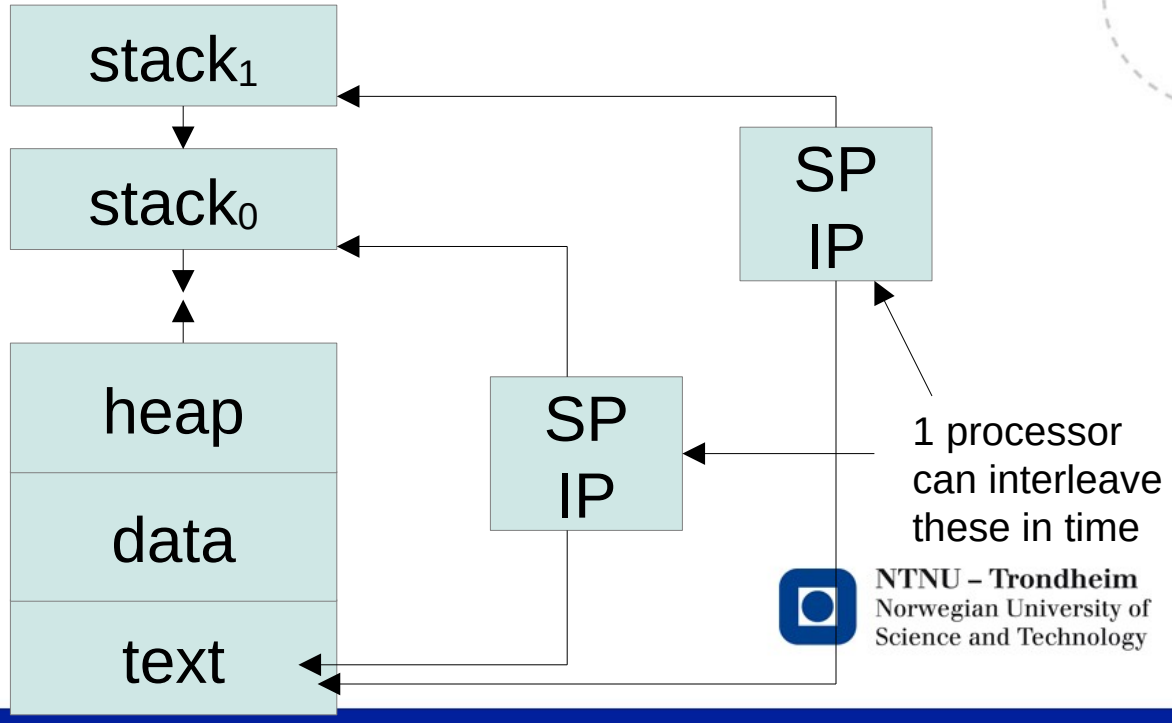
Shared memory

- Here's the memory image of a process again:



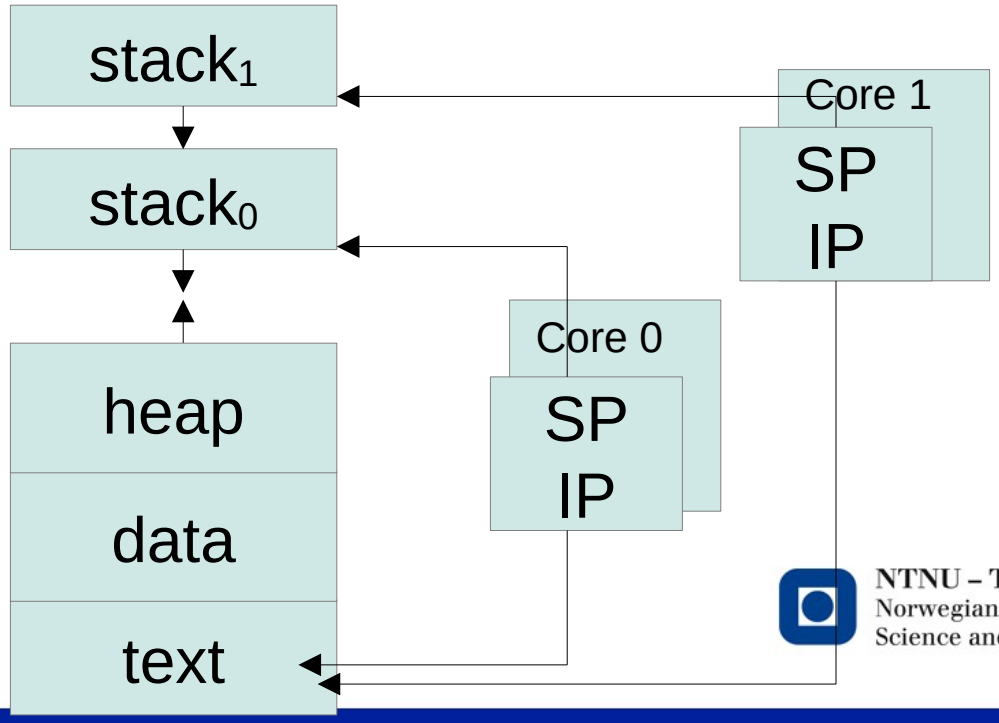
Threads

- Originally, threads were a mechanism meant to dispatch a concurrent function call until its result was required
- That requires additional stacks and IPs, but nothing more:



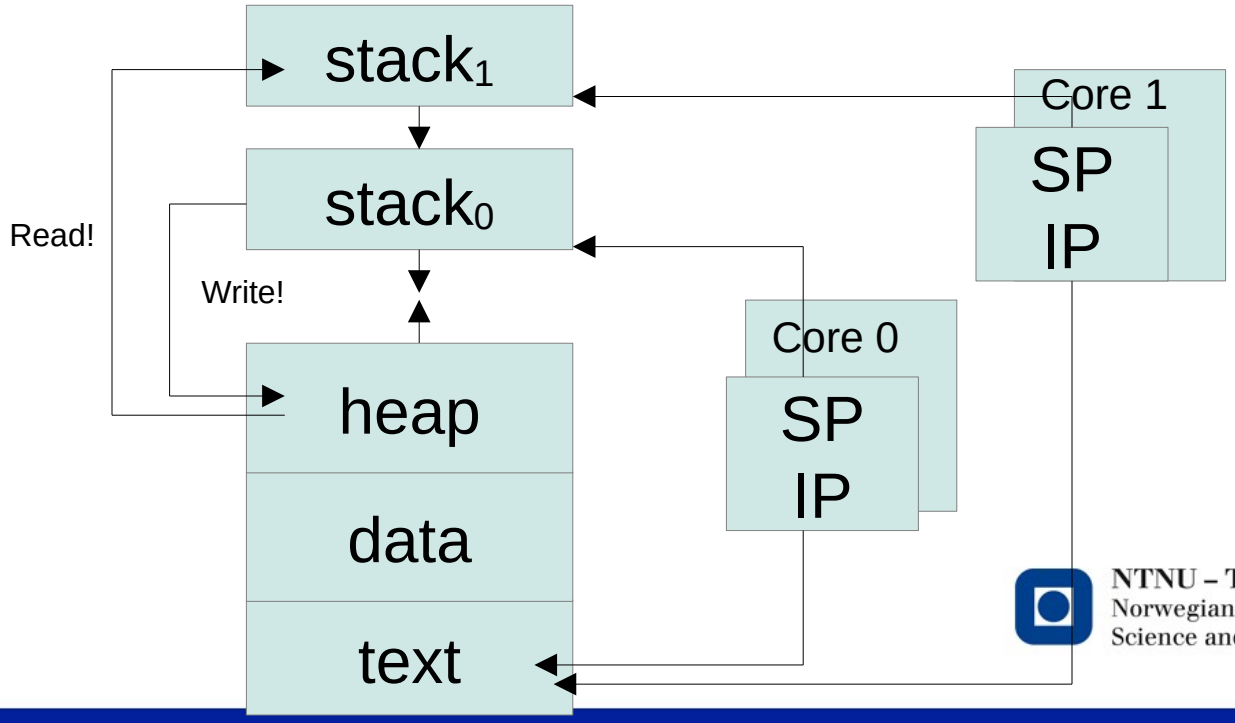
Threads & multiple cores

- If we have different processors with independent IP and SP registers, they can execute threads simultaneously:



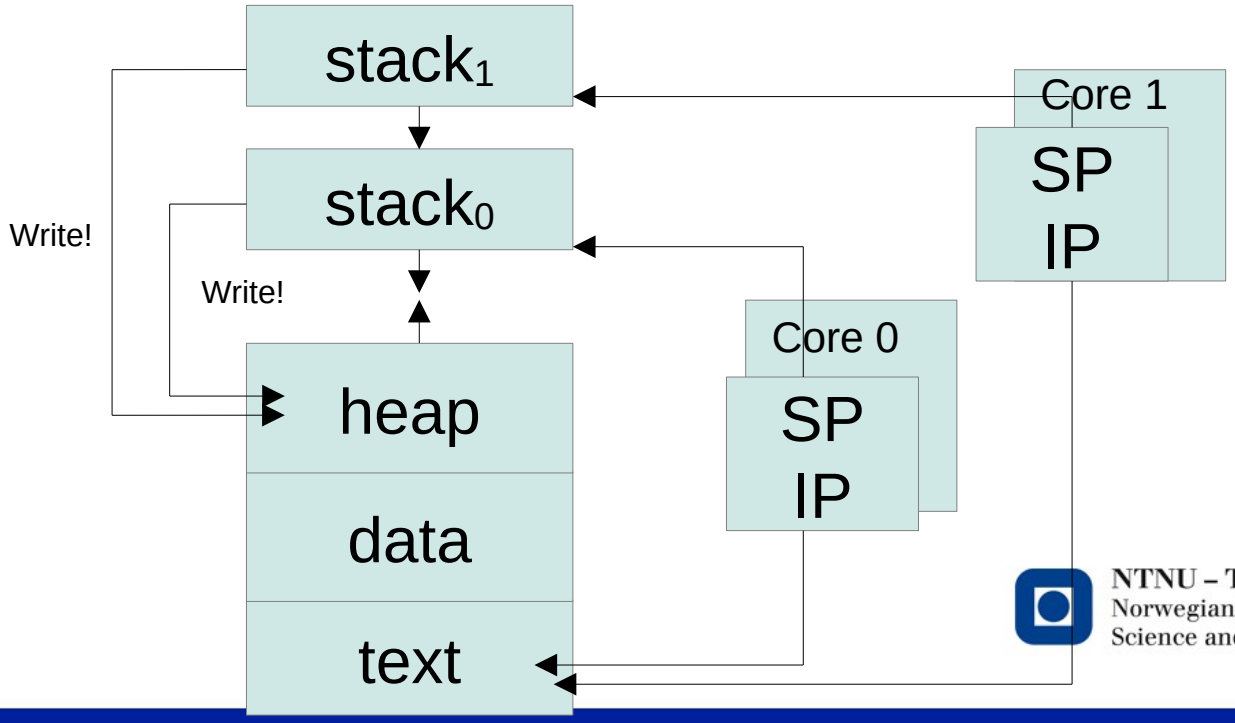
Threads & invisible communication

- If threads coordinate access to the same location outside of their own stacks, the result is instantly available to all of them:



Threads & race conditions

- If threads try to write to the same location simultaneously, the result will be determined by timing (last writer wins)

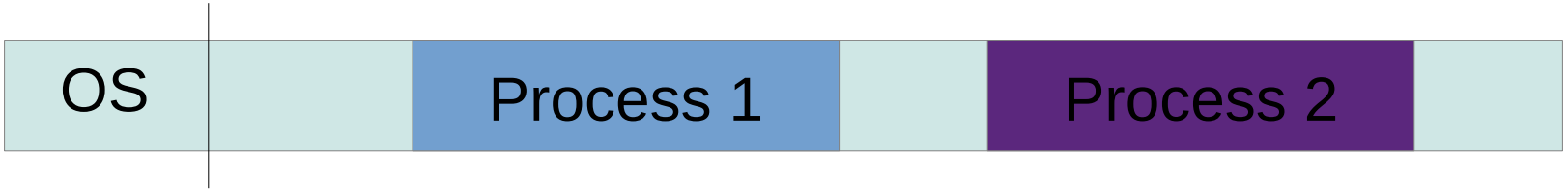


Shared memory pros & cons

- The good part:
  - Threads only need to keep 1 copy of their shared data, and they can all work on modifying it
  - ...as long as they coordinate who can write at any given time
- The bad part:
  - Threads have to live inside the address space of a single process, so a single OS must manage their memory, and therefore they can’t live on separate computers
  - We only get as many as we can fit into one machine

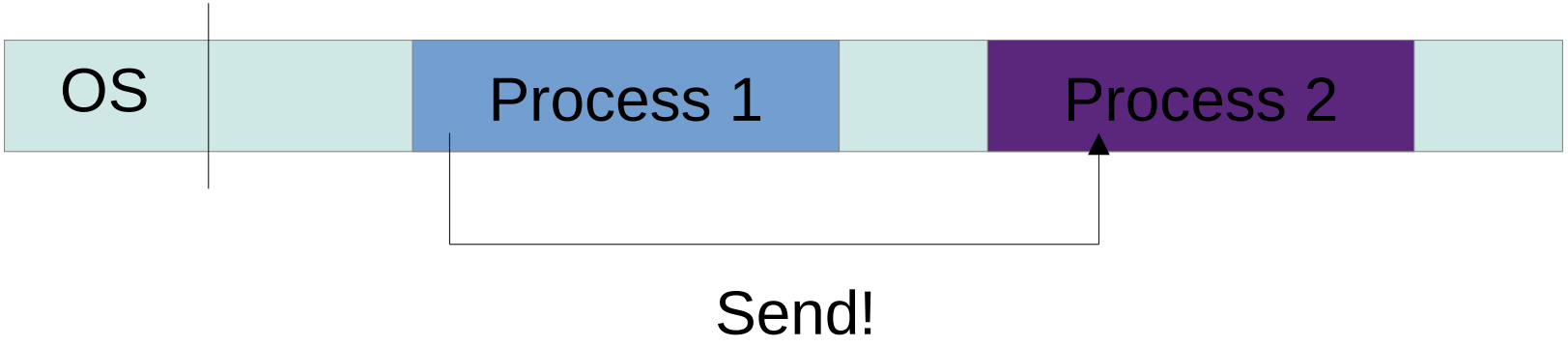
Distributed memory

- If we replicate *processes* instead, we get the situation from our discussion of virtual memory:



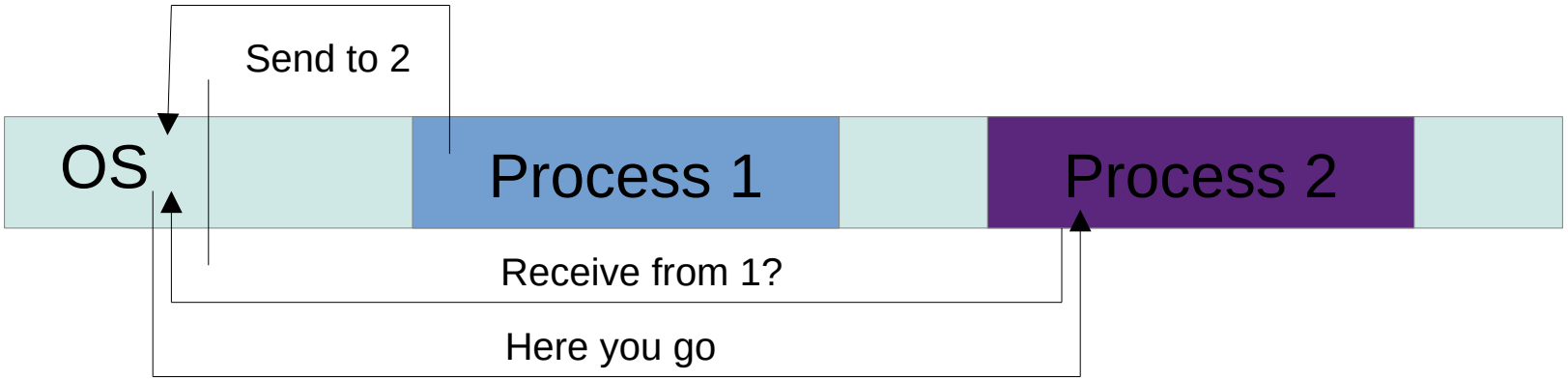
Not distributed memory communication

- This won't work, the page tables prevent process 1 from writing where process 2 can read



Distributed memory communication

- When processes are launched simultaneously, we can give them each other's ID numbers
- This allows them to establish some shared workspace under supervision
- Function calls can transmit data to and from it, but traffic has to be initiated by the process itself

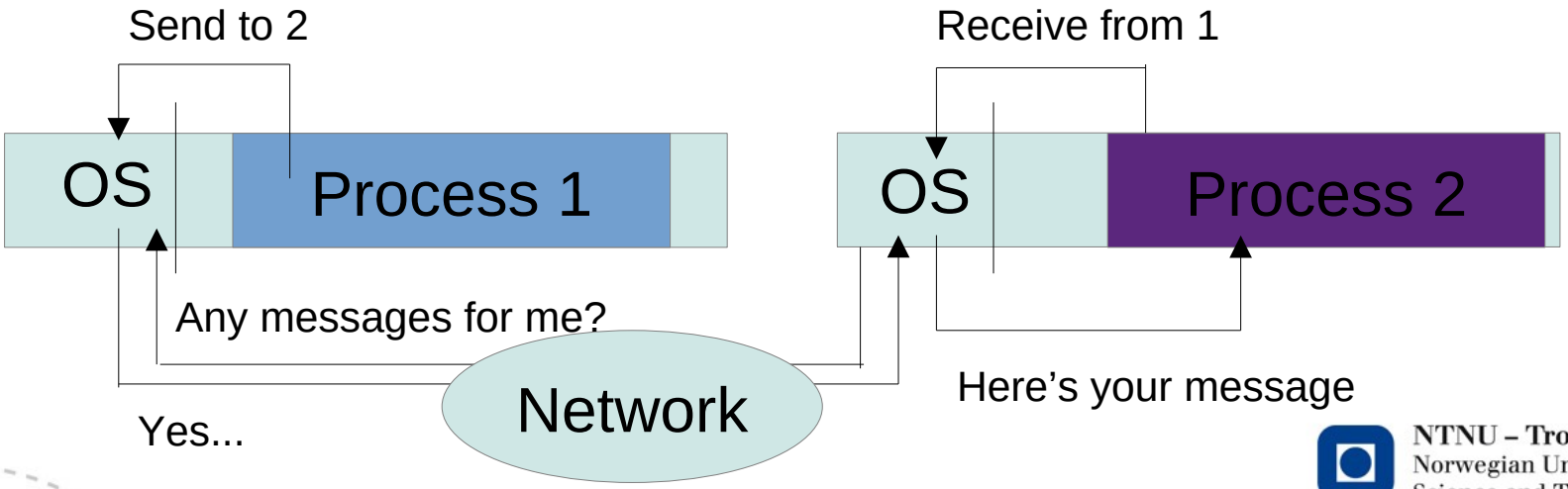


Distributed memory pros & cons

- The good part:
  - Processes can't involuntarily have their memory corrupted, they only receive what they have asked for in a designated place
  - There's one more thing, which we get to in a moment
- The bad part:
  - After data transmission, there are two copies of the thing you wanted to communicate, which occupies twice as much memory.

Distributed memory: the extra good part

- Processes are already assumed to have completely distinct address spaces, so it doesn't matter whether they run on the same computer or not:

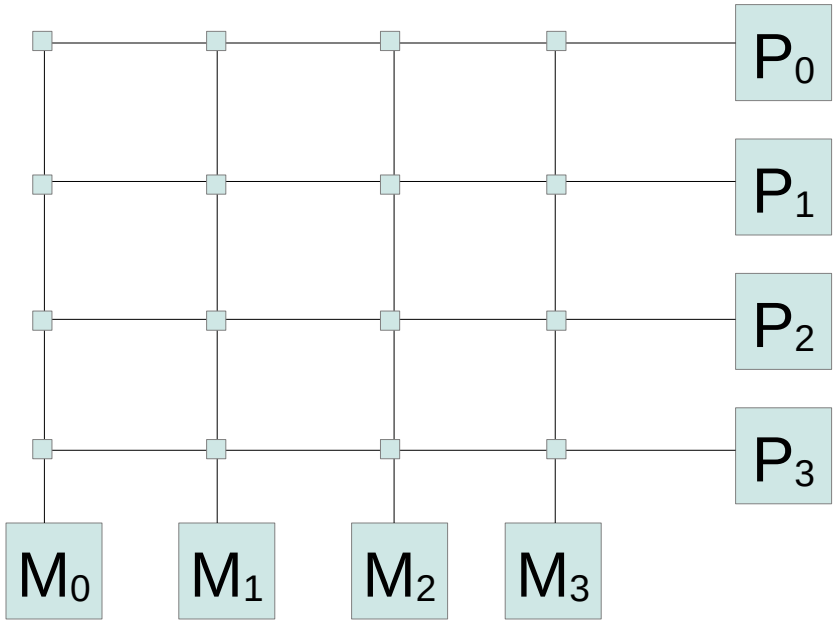


Multi-computer parallelism

- The difference between two processes on one computer and two processes on two computers is handled by communication libraries and OS
- Inside the processes, the send- and receive-calls can look exactly the same, the difference is just that it takes a little longer to shift data across a network than it takes to copy between memory banks
- If we run out of processors, we can just add another machine

Interconnects

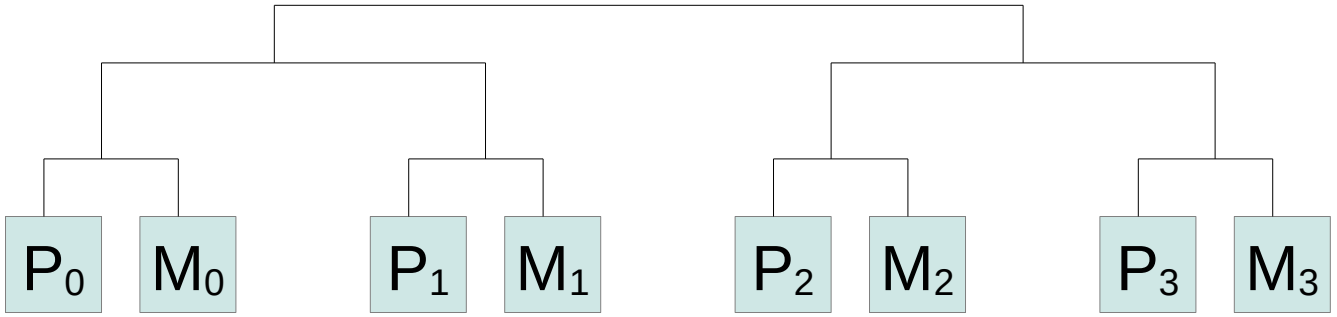
- In shared memory systems, one way to connect a set of processors to a set of memory banks is a *crossbar:*



Efficient, but impractical for large numbers,
cost grows as product of processors and memory

(Fat) trees

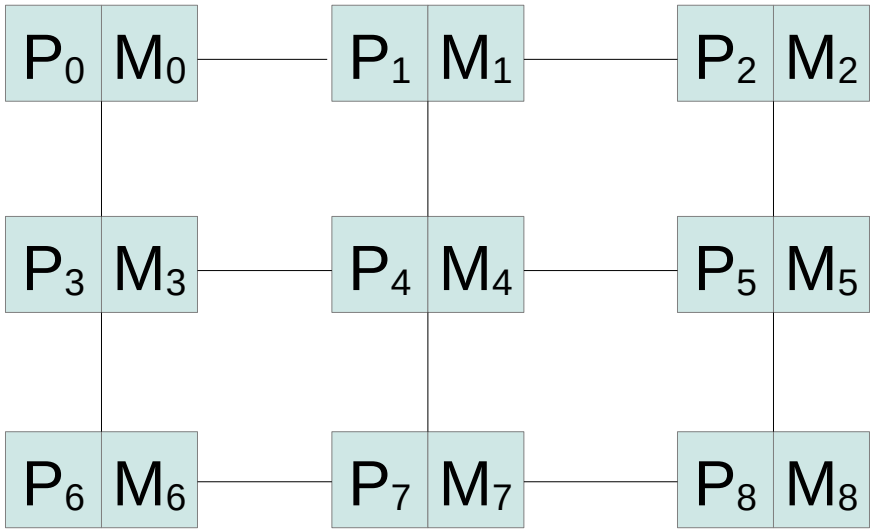
- Another is to associate processors with memory modules that are their responsibility:



- Less expensive at scale, but gives *non-uniform memory access* (NUMA) effects (remote is slower)
- *Fat* trees compensate for cross-section traffic volume by providing more bandwidth near the root of the tree structure

Mesh

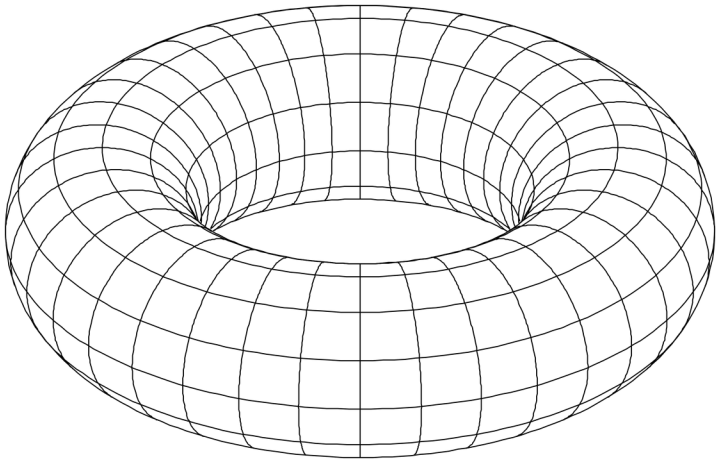
- Constant number of links per unit, messages routed throughout the network in several “hops”:



Very scalable, but communication latency grows linearly with distance

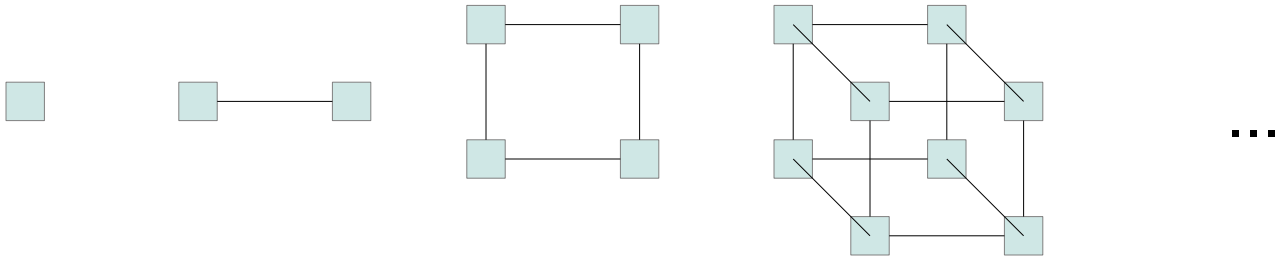
Torus

- This is just a mesh that wraps around the edges:



Hypercube

- Add 1 dimension by replicating what you had from before, and connecting all the matching points:



- Requires $\log_2 P$ links *per processor* for P processors
- Particularly good for *d-cube* algorithms, e.g. the Fast Fourier Transform

Interconnection fabrics

- Several of these graph shapes can typically be found at various levels of granularity in a large computer
- In combination, we call them the *interconnect fabric*
  - Some parts are memory logic
  - Some parts are network connections
  - To a suitably parallelized program, they combine into how much it costs to send data from A to B