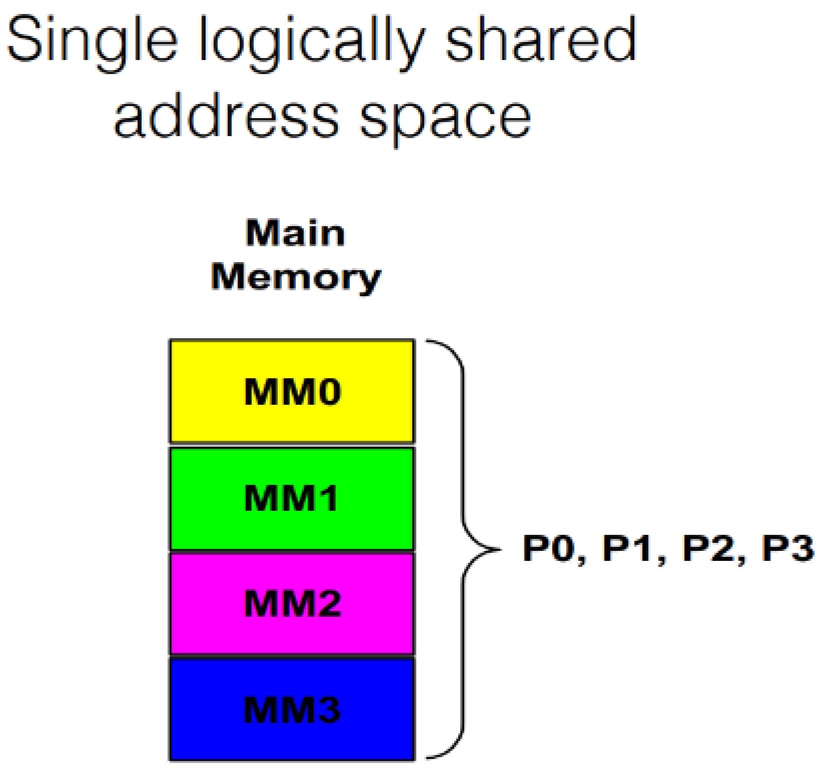
14.Parallel Processors-MIMD Architectures

Memory Address Space Model

Single logically shared address space: A memory reference can be made by any processor to any memory location ⇒Shared Memory Architectures.

- The address space is shared among processors: The same physical address on 2 processors refers to the same location in memory.
  Multiple and private address spaces: The processors communicate among them through send/receive primitives ⇒ Message Passing Architectures.
- The address space is logically disjoint and cannot be addressed by different processors: the same physical address on 2 processors refers to 2 different locations in 2 different memories



Shared Address

The processors communicate among them through shared variables in memory.
Implicit management of the communication through load/store operations to access any memory locations.

- Oldest and most popular model
  Shared memory does not mean that there is a single centralized memory.

Multiple and Private Addresses

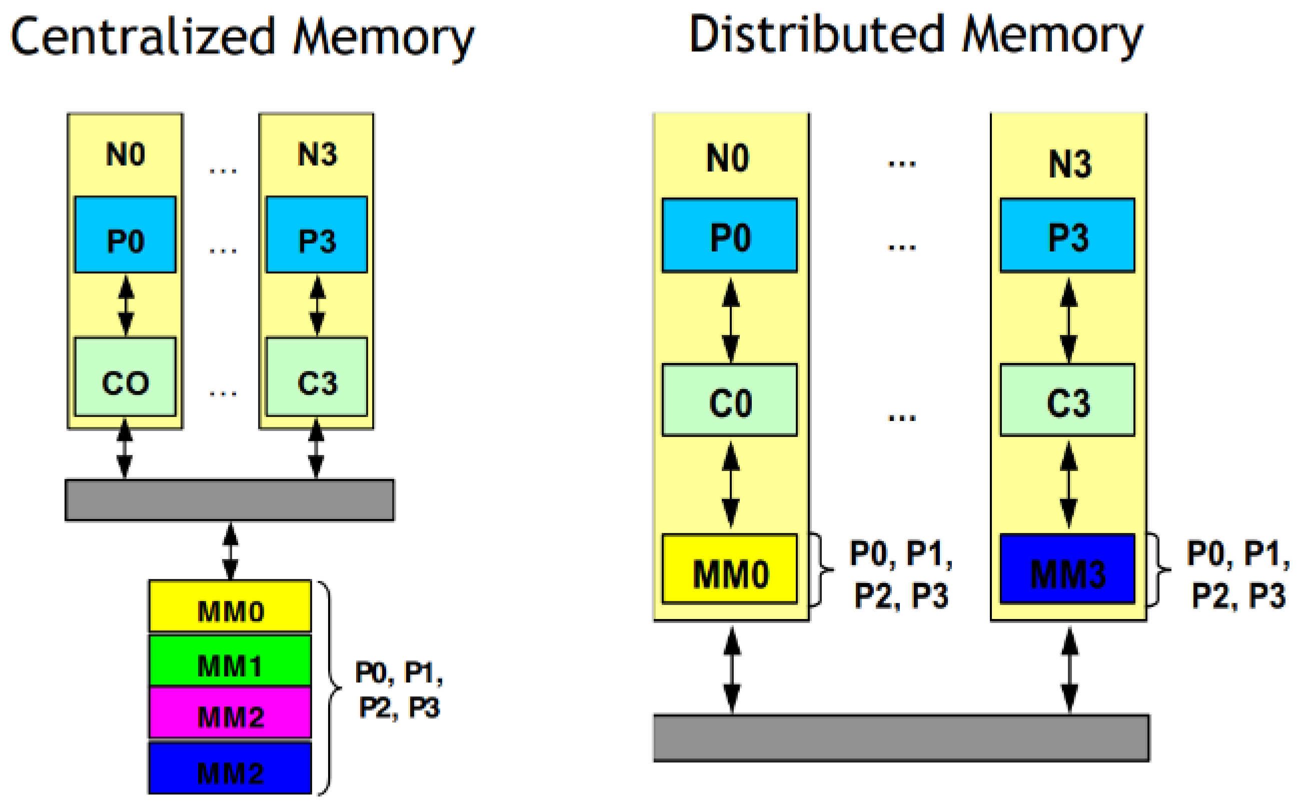
The processors communicate among them through sending/receiving messages.
Explicit management of the communication through send/receive primitives to access private memory locations.
No cache coherency problem among processors.

Physical Memory Organization

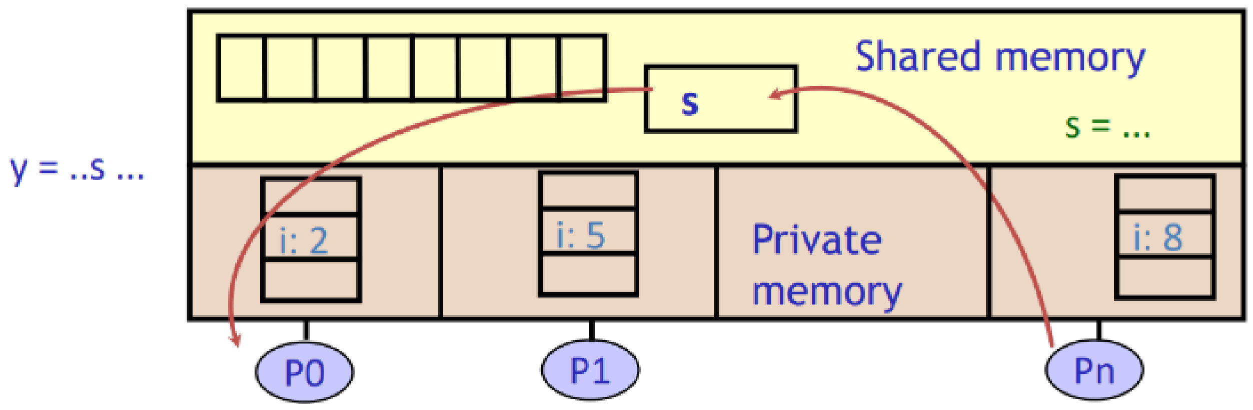
Centralized shared-memory architectures:

- at most few dozen processor chips (< 100 cores)
- Large caches, single memory multiple banks
- Often called symmetric multiprocessors (SMP) and the style of architecture called Uniform Memory Access (UMA)
  Distributed memory architectures:
- To support large processor counts
- Requires high-bandwidth interconnect
- Cons: data communication among processors
- Non Uniform Memory Access (NUMA)
  The concepts of addressing space (single/multiple) and the physical memory organization (centralized/distributed) are orthogonal to each other.
  Multiprocessor systems can have single addressing space and distributed physical memory.

Single Logically Shared Address Space (Shared-Memory Architectures)

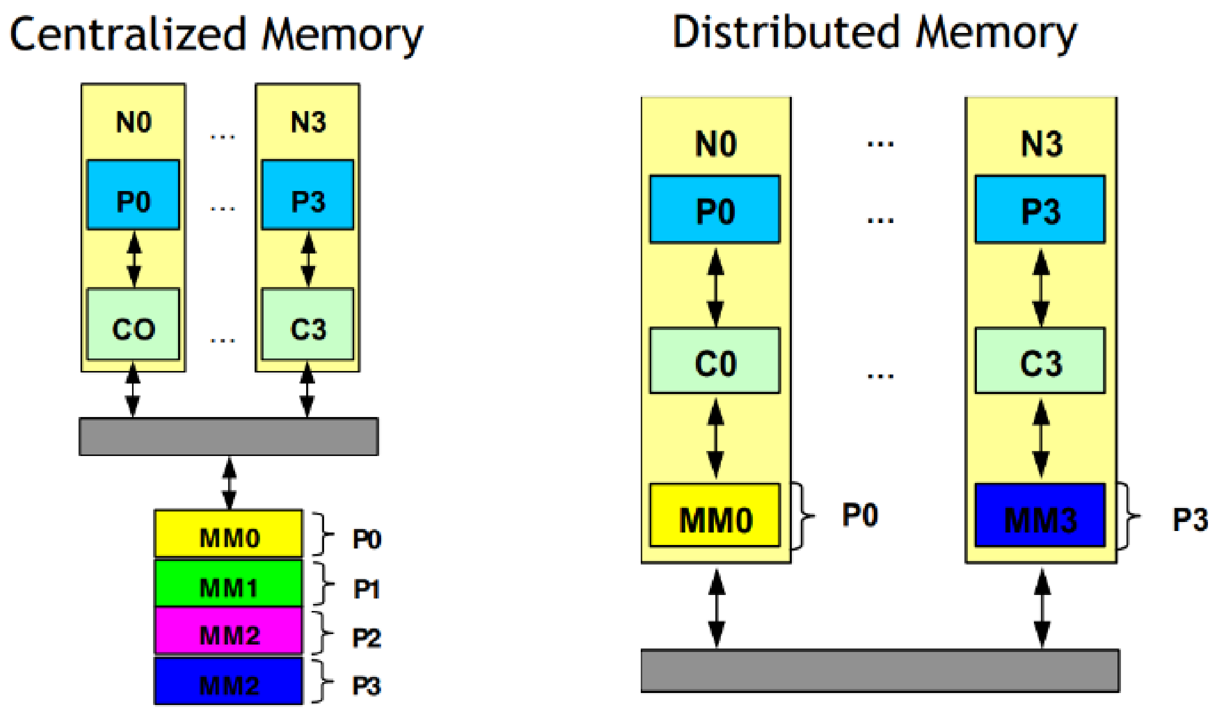


Programming Model 1: Shared Memory

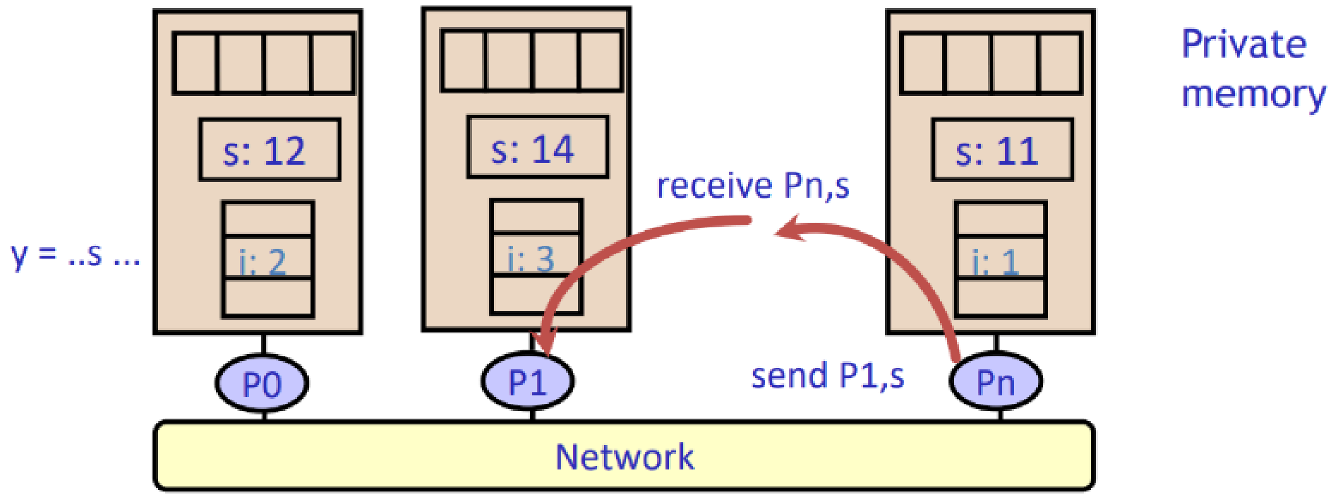


- Program is a collection of threads of control.
    - Can be created dynamically, mid-execution, in some languages
- Each thread has a set of private variables, e.g., local stack variables
- Also a set of shared variables, e.g., static variables, shared common blocks, or global heap.
    - Threads communicate implicitly by writing and reading shared variables.
    - Threads coordinate by synchronizing on shared variables

Multiple and Private Address Space (Message Passing Architectures)



Programming Model 2: Message Passing



Program consists of a collection of named processes.
- Usually fixed at program startup time
- Thread of control plus local address space -- NO shared data.
- Logically shared data is partitioned over local processes.

Processes communicate by explicit send/receive pairs
- Coordination is implicit in every communication event.
- MPI (Message Passing Interface) is the most commonly used SW

Which is better? SM or MP?

Depends on the program!
Both are “communication Turing complete”

- i.e. can build Shared Memory with Message Passing and vice-versa
  Advantages of Shared Memory:
- Implicit communication (loads/stores)
- Low overhead when cached
  Disadvantages of Shared Memory:
- Complex to build in way that scales well
- Requires synchronization operations
- Hard to control data placement within caching system
  Advantages of Message Passing
- Explicit Communication (sending/receiving of messages)
- Easier to control data placement (no automatic caching)
  Disadvantages of Message Passing
- Message passing overhead can be quite high
- More complex to program
- Introduces question of reception technique (interrupts/polling)

The Problem of Cache Coherence

Shared-Memory Architectures cache both private data (used by a single processor) and shared data (used by multiple processors to provide communication).
When shared data are cached, the shared value may be replicated in multiple caches.
In addition to the reduction in access latency and required memory bandwidth, this replication provides a reduction of shared data contention read by multiple processors simultaneously.
Private processor caches create a problem

- Copies of a variable can be present in multiple caches
- A write by one processor may not become visible to others
  The use of multiple copies of same data introduces a new problem: cache coherence.

What Does Coherency Mean?

Informally:

- “Any read must return the most recent write”
- Too strict and too difficult to implement
  Better:
- “Any write must eventually be seen by a read”
- All writes are seen in proper order (“serialization”)
  Two rules to ensure this:

1. If P writes x and P1 reads it, P’s write will be seen by P1 if the read and write are sufficiently far apart and no other writes to x occur between the two accesses
2. Writes to a single location are serialized: two writes to the same location by any two processors are seen in the same order by all processors.
    - Latest write will be seen
    - Otherwise could see writes in illogical order (could see older value after a newer value)