Understanding
Parallel Computers - Paradigms and Programming Models

Sofien GANNOUNI
Computer Science
E-mail: gnnosf@ksu.edu.sa ; gansof@yahoo.com

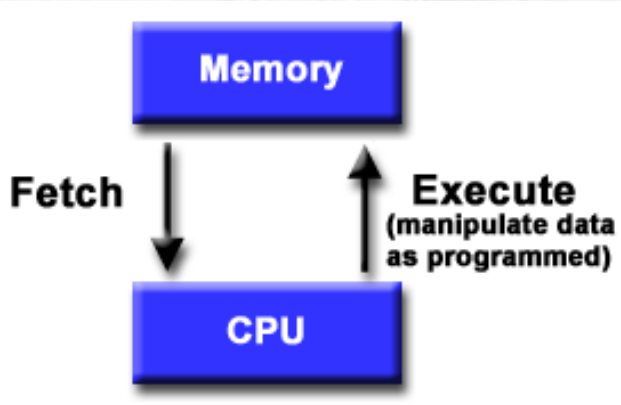
Von Neumann Architecture

- For over 40 years, virtually all computers have followed a common machine model known as the von Neumann computer. Named after the Hungarian mathematician John von Neumann.

- A von Neumann computer uses the stored-program concept. The CPU executes a stored program that specifies a sequence of read and write operations on the memory.

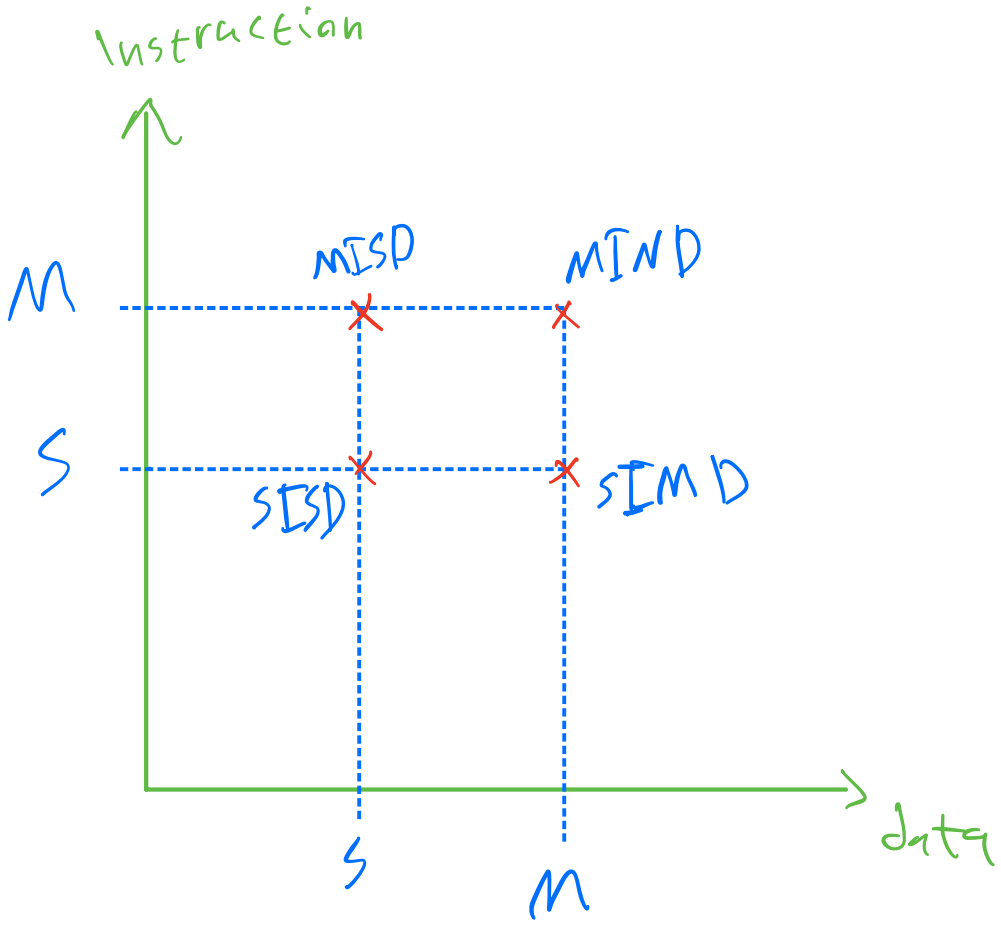
Basic Design

- **Basic design**
  - Memory is used to store both program and data instructions
  - Program instructions are coded data which tell the computer to do something
  - Data is simply information to be used by the program
- **A central processing unit (CPU) gets instructions and/or data from memory, decodes the instructions and then *sequentially* performs them.**



Flynn's Classical Taxonomy

- There are different ways to classify parallel computers. One of the more widely used classifications, in use since 1966, is called Flynn's Taxonomy.
- Flynn's taxonomy distinguishes multi-processor computer architectures according to how they can be classified along the two independent dimensions of *Instruction* and *Data*. Each of these dimensions can have only one of two possible states: *Single* or *Multiple*.



M: mutiple

S: single

I: instruction

D: Data

Flynn Matrix

- The matrix below defines the 4 possible classifications according to Flynn

SISD Single Instruction, Single Data	**SIMD** Single Instruction, Multiple Data
MISD Multiple Instruction, Single Data	**MIMD** Multiple Instruction, Multiple Data

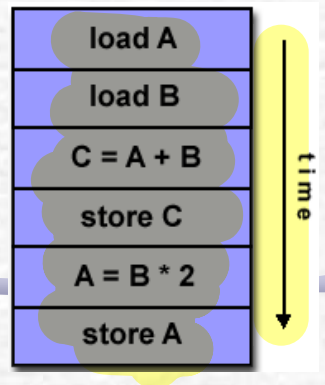
- a stream of instructions (the algorithm) tells the computer what to do.
- a stream of data (the input) is affected by these instructions.

Single Instruction, Single Data (SISD)

Sequential / single processor

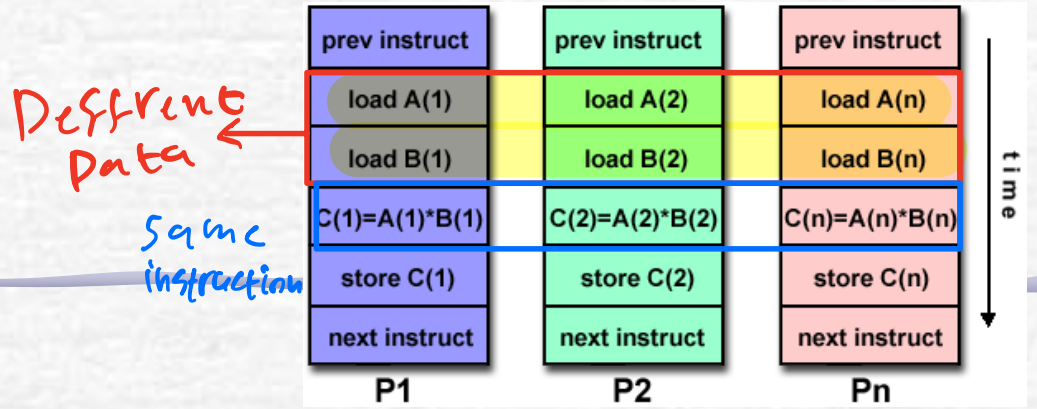
- A serial (non-parallel) computer
- Single instruction: only one instruction stream is being acted on by the CPU during any one clock cycle
- Single data: only one data stream is being used as input during any one clock cycle
- This is the oldest and until recently, the most prevalent form of computer
- Examples: most PCs, single CPU workstations and mainframes

Eg: Von Neuman architecture



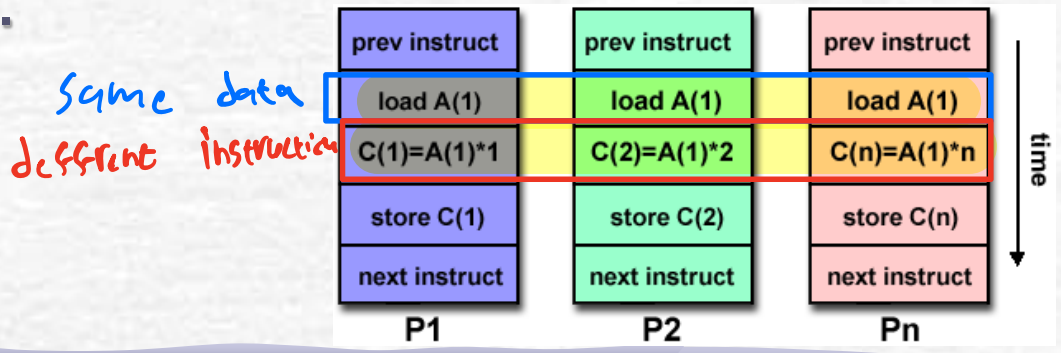
Single Instruction, Multiple Data (SIMD)

- Single instruction: All processing units execute the same instruction at any given clock cycle
- Multiple data: Each processing unit can operate on a different data element
- Best suited for specialized problems characterized by a high degree of regularity, such as image processing.
- Examples:
  - Processor Arrays: Connection Machine CM-2, Maspar MP-1, MP-2
  - Vector Pipelines: IBM 9000, Cray C90, Fujitsu VP, NEC SX-2, Hitachi S820



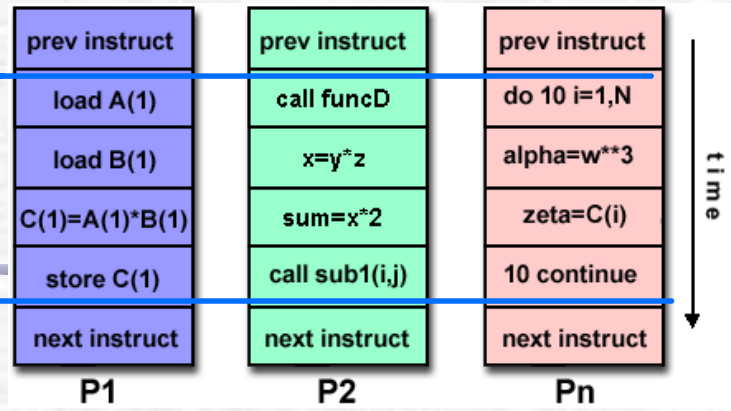
Multiple Instruction, Single Data (MISD)

- A single data stream is fed into multiple processing units.
- Each processing unit operates on the data independently via independent instruction streams.
- Some conceivable uses might be:
  - multiple frequency filters operating on a single signal stream
  - multiple cryptography algorithms attempting to crack a single coded message.

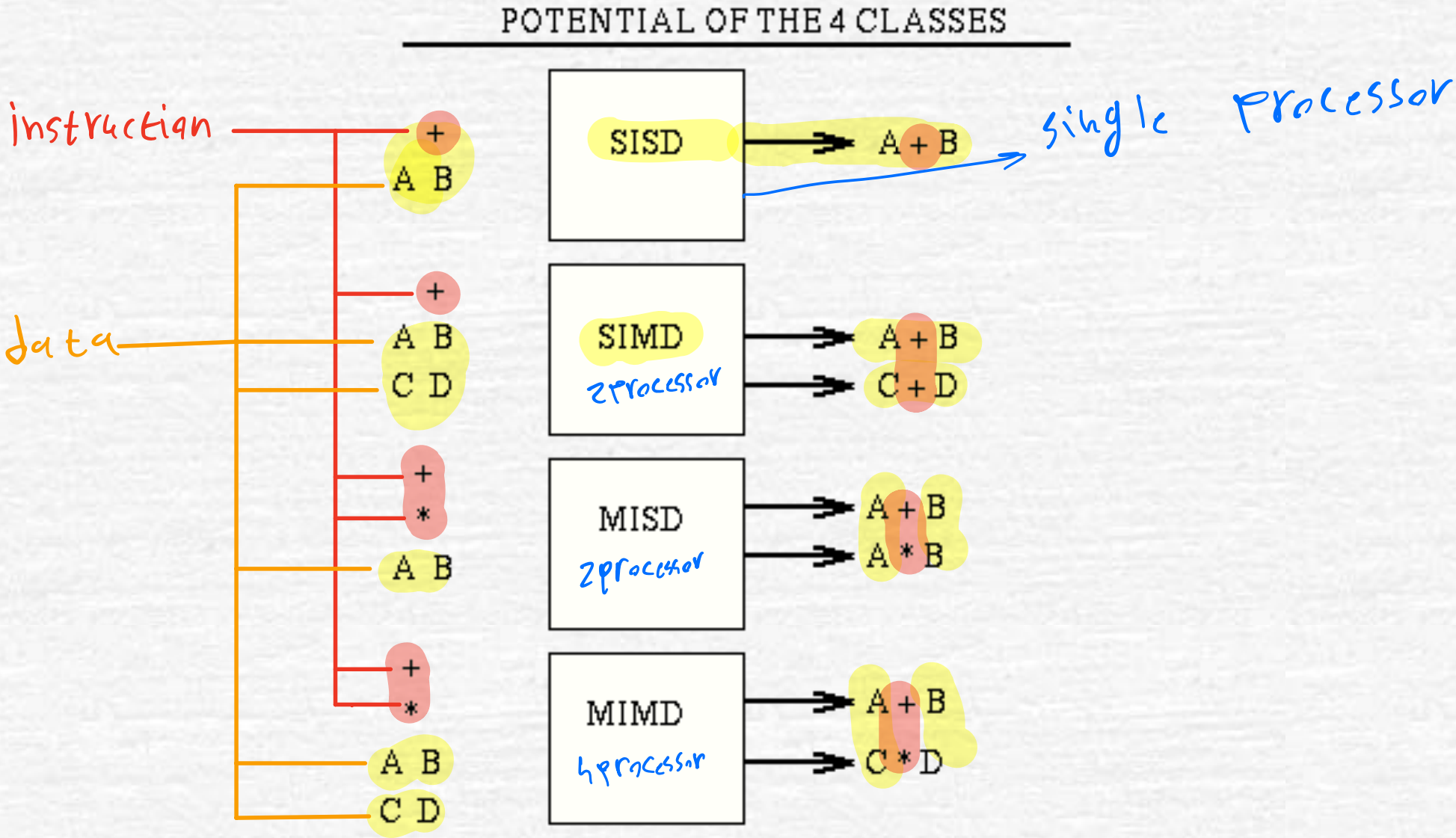


Multiple Instruction Multiple Data (MIMD)

- Currently, the most common type of parallel computer. Most modern computers fall into this category.
- Multiple Instruction: every processor may be executing a different instruction stream
- Multiple Data: every processor may be working with a different data stream
- Examples: most current supercomputers, networked parallel computer "grids" and multi-processor SMP computers - including some types of PCs.



Potential of the 4 Classes

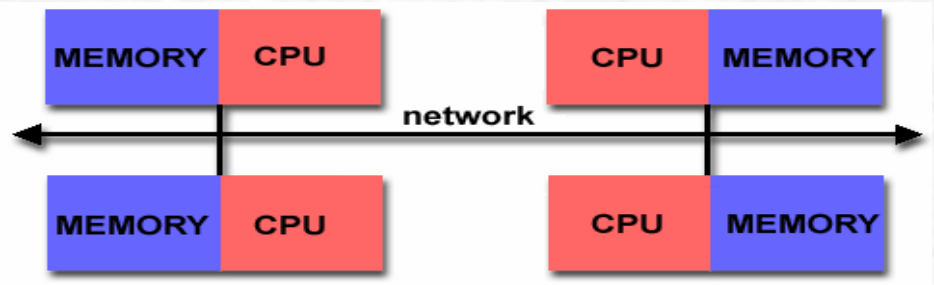


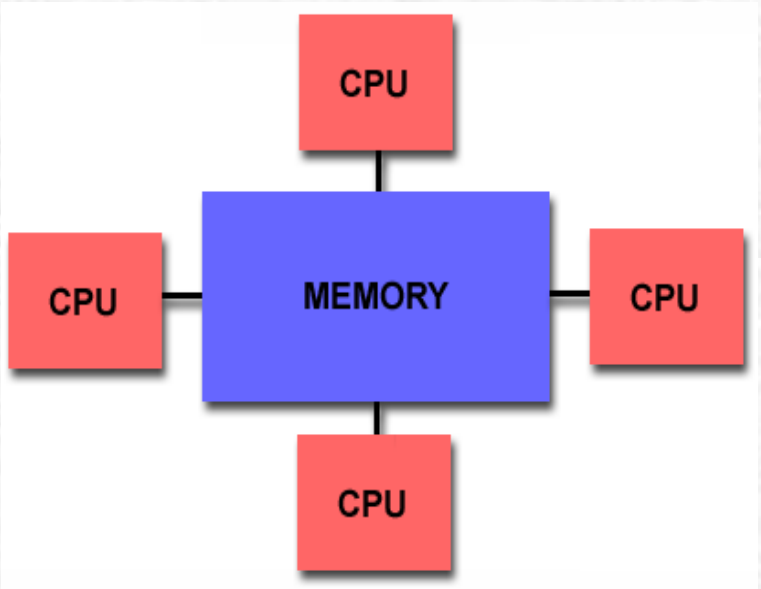
Parallel Paradigms

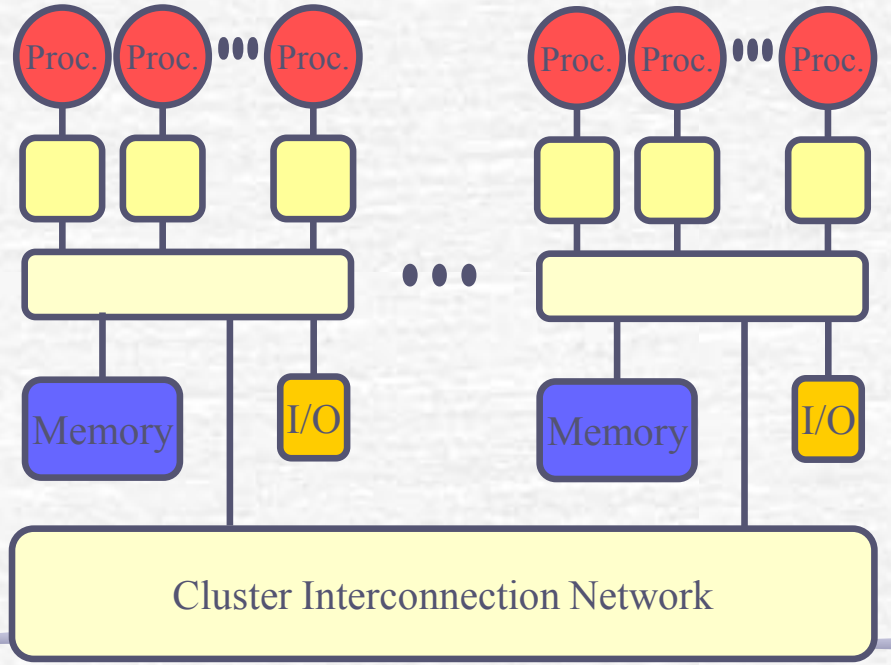
- Shared Memory
- Message Passing
- Multi-threading

Shared Memory Paradigm

1. Centralized Shared memory
2. Distributed memory
3. Hybrid systems





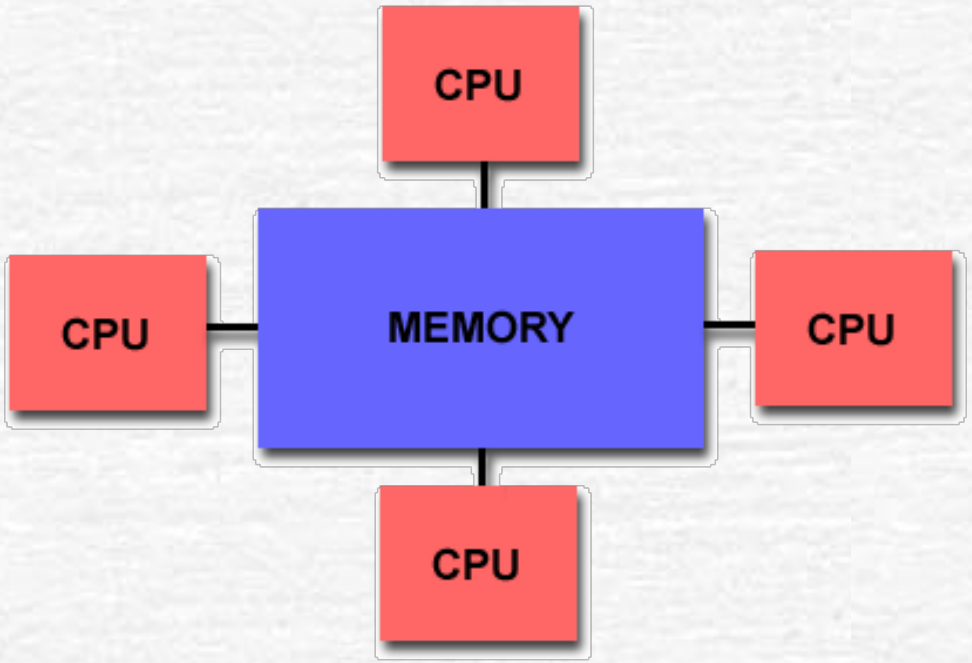


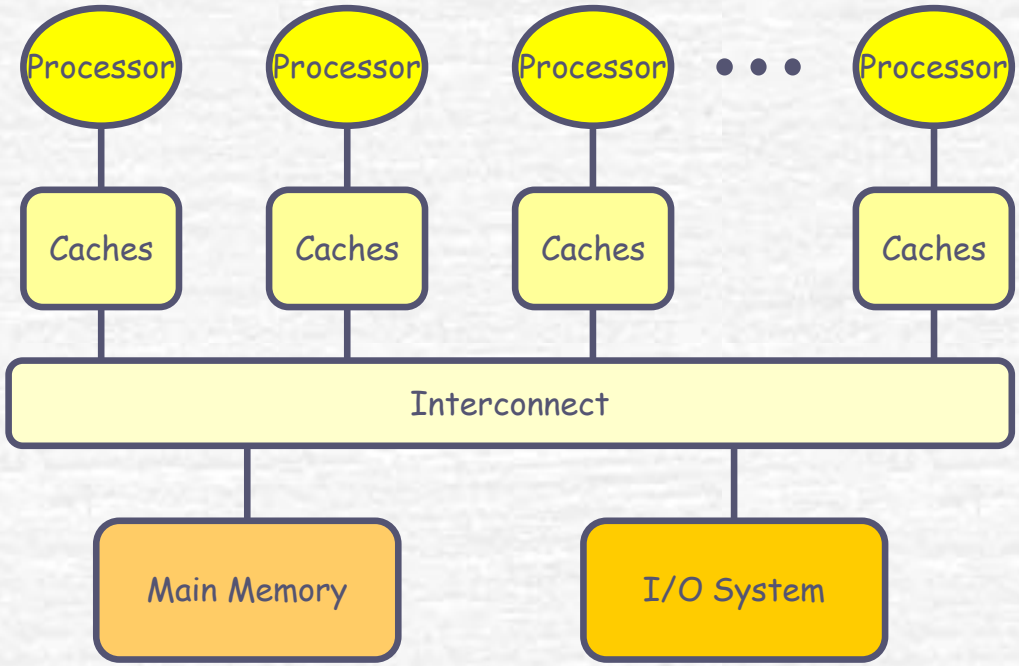
Memory architectures

- **Shared Memory**
  - is memory that may be simultaneously accessed by multiple programs with an intent to provide communication among them or avoid redundant copies. Shared memory is an efficient means of passing data between programs.
- **Distributed Memory**
  - refers to a multiprocessor computer system in which each processor has its own private memory. Computational tasks can only operate on local data, and if remote data is required, the computational task must communicate with one or more remote processors.
- **Hybrid Distributed-Shared Memory**
  - hybrid programming techniques combining the best of distributed and shared memory programs are becoming more popular.

Centralized Shared Memory

- Multiple processors can operate independently but share the same memory resources.
- Changes in a memory location effected by one processor are visible to all other processors.
- Shared memory machines can be divided into two main classes based upon memory access times: *UMA* and *NUMA*.





Shared Memory : UMA vs. NUMA

- **Uniform Memory Access (UMA):**
  - Most commonly represented today by Symmetric Multiprocessor (SMP) machines
  - Identical processors
  - Equal access and access times to memory
  - Sometimes called CC-UMA - Cache Coherent UMA. Cache coherent means if one processor updates a location in shared memory, all the other processors know about the update. Cache coherency is accomplished at the hardware level.

- **Non-Uniform Memory Access (NUMA):**
  - Often made by physically linking two or more SMPs
  - One SMP can directly access memory of another SMP
  - Not all processors have equal access time to all memories
  - Memory access across link is slower
  - If cache coherency is maintained, then may also be called CC-NUMA - Cache Coherent NUMA

hardware level cc-UMA:

1 - shared lock ⟹ read

2 - exlusive lock ⟹ write

lock problem

dead lock: two processor wating
eatch other with two
defrent data item

UMA vs NUMA

UMA

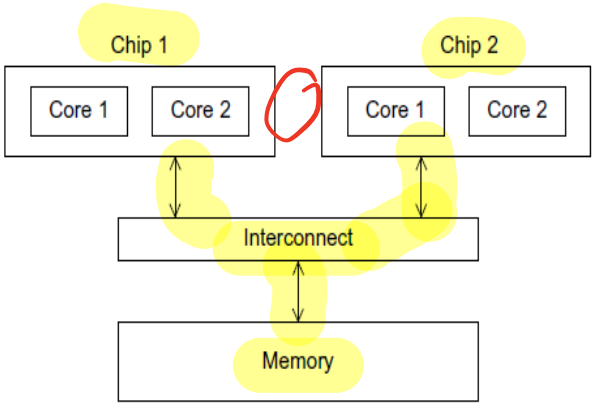


FIGURE 2.5

A UMA multicore system

NUMA

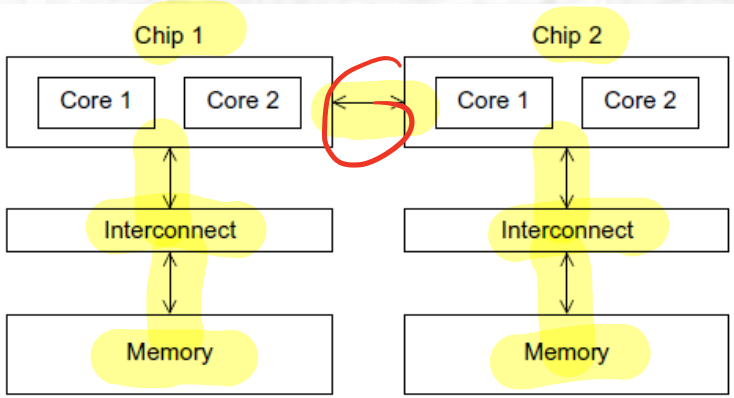


FIGURE 2.6

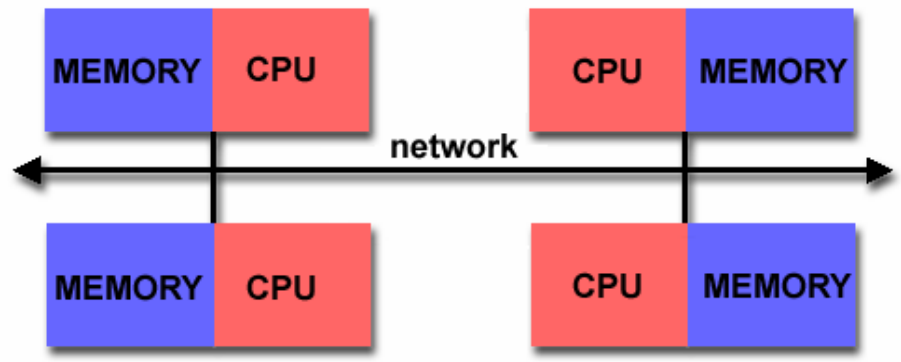
A NUMA multicore system

Centralized Shared Memory

- **Advantages**
  - Global address space provides a user-friendly programming perspective to memory
  - Data sharing between tasks is both fast and uniform due to the proximity of memory to CPUs
- **Disadvantages:**
  - Lack of scalability between memory and CPUs. Adding more CPUs can increases traffic on the shared memory-CPU path, and for cache coherent systems, increase traffic associated with cache/memory management.
  - Programmer responsibility for synchronization constructs that insure "correct" access of global memory.
  - Expense: it becomes increasingly difficult and expensive to design and produce shared memory machines with ever increasing numbers of processors.

Distributed Memory

- **Processors have their own local memory.**
  - Memory addresses in one processor do not map to another processor, so there is no concept of global address space across all processors.
- **Because each processor has its own local memory, it operates independently.**
  - Changes it makes to its local memory have no effect on the memory of other processors.
  - Hence, the concept of cache coherency does not apply.
- **When a processor needs access to data in another processor, it is usually the task of the programmer to explicitly define how and when data is communicated.**
  - Synchronization between tasks is likewise the programmer's responsibility.

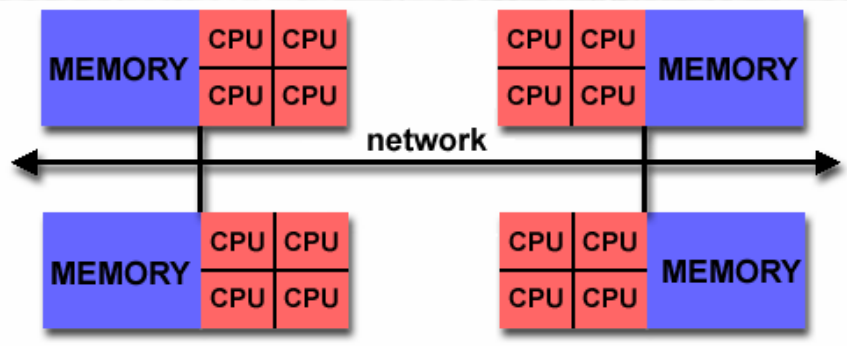


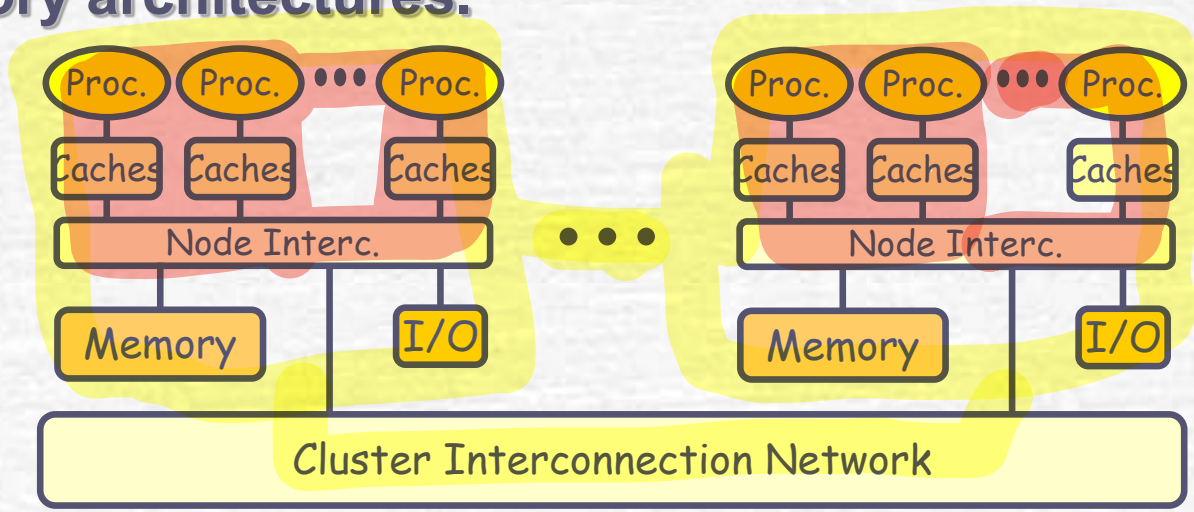
Distributed Memory: Pro and Con

- **Advantages**
  - Memory is scalable with number of processors.
    - Increase the number of processors and the size of memory increases proportionately.
  - Each processor can access its own memory without interference and without the overhead incurred with trying to maintain cache coherency.
  - Cost effectiveness: can use commodity, off-the-shelf processors and networking.
- **Disadvantages**
  - The programmer is responsible for many of the details associated with data communication between processors.

Hybrid Distributed-Shared Memory

- The largest and fastest computers in the world today employ both shared and distributed memory architectures.





- The shared memory component is usually a cache coherent SMP machine.
  - Processors on a given SMP can address that machine's memory as global.
- The distributed memory component is the networking of multiple SMPs.
  - SMPs know only about their own memory - not the memory on another SMP. Therefore, network communications are required to move data from one SMP to another.
- Current trends seem to indicate that this type of memory architecture will continue to prevail and increase.
- Advantages and Disadvantages:
  - whatever is common to both shared and distributed memory architectures.

Message Passing

Point - Point
collective

- Allows for communication between a set of processors.
- Each processor is required to have a local memory, no global memory is required.
- The whole address space of the system consists of multiple private address spaces.
- Communication occurs between processors by sending and receiving messages.

في الكود نرسل الـ task لـ processor ثاني بعدين (send)

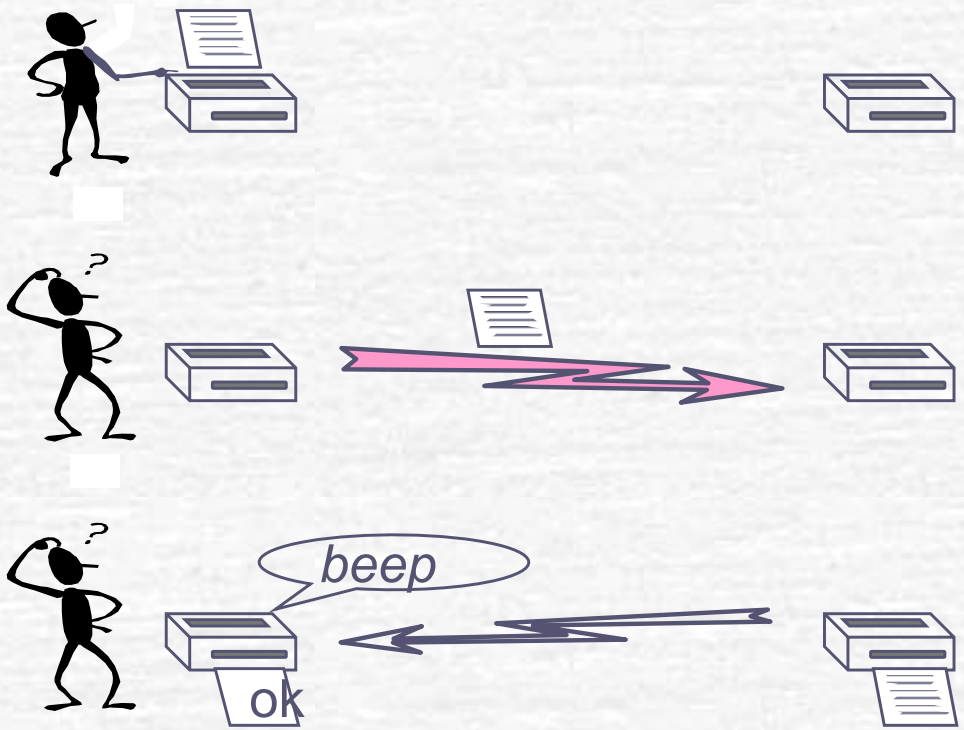
نسوي receiving

Point-to-Point Communication

- Simplest form of message passing.
- One process sends a message to another.
- Different types of point-to-point communication:
  - synchronous send
  - Asynchronous (buffered) send

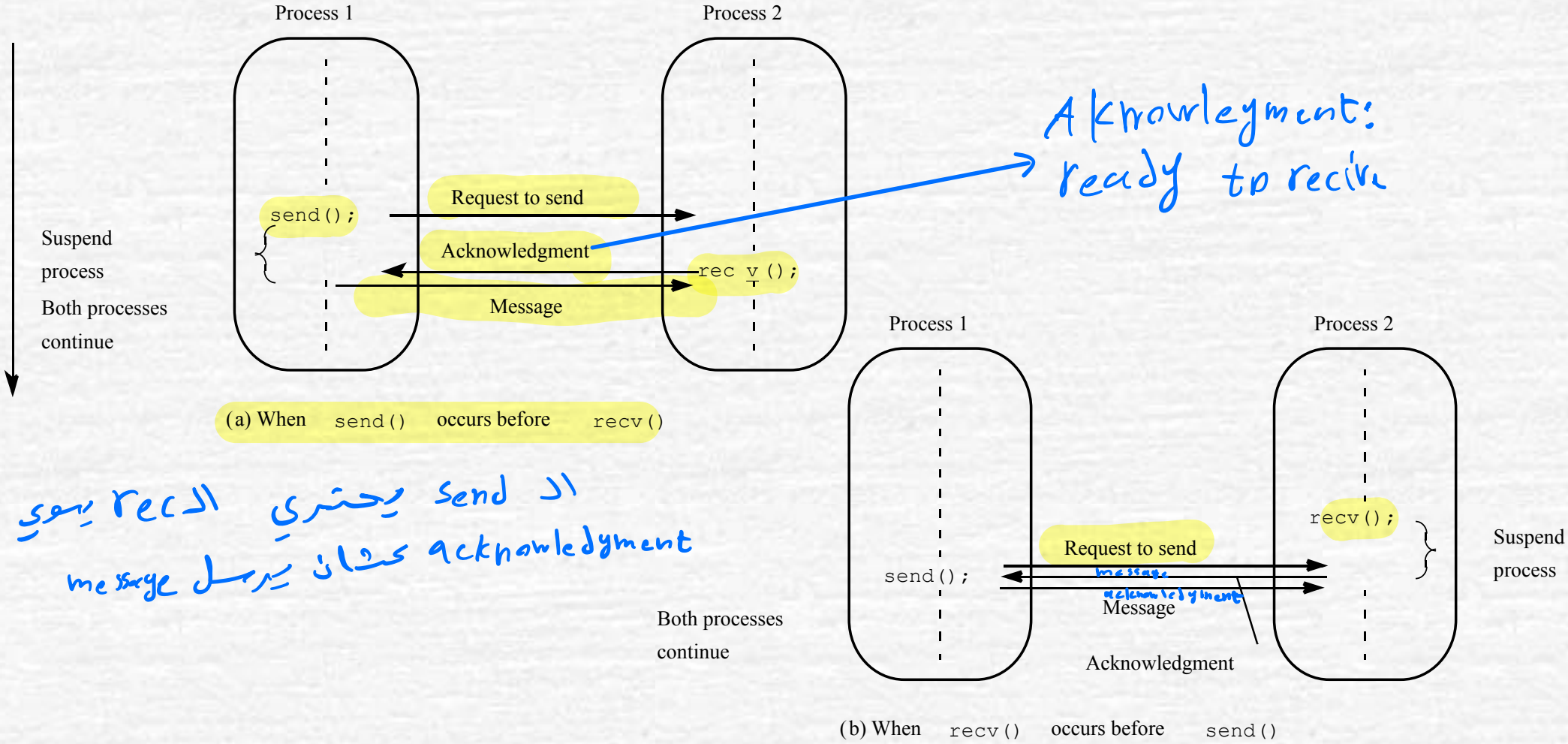
Synchronous Sends

- The sender gets an information that the message is received.
- Analogue to the *beep* or *okay-sheet* of a fax.



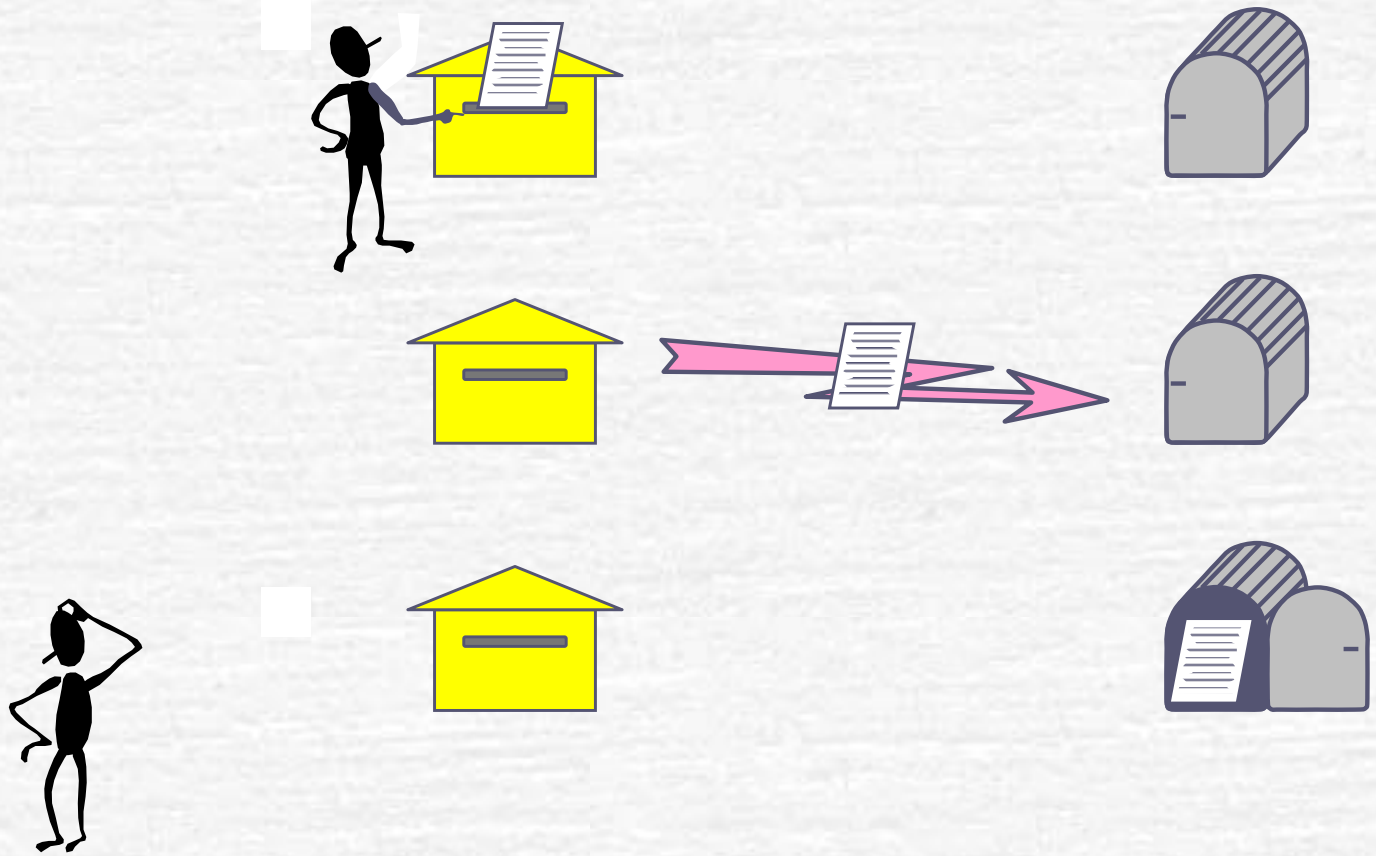
Synchronous Sends

- Synchronous send() and recv() library calls using a three-way protocol



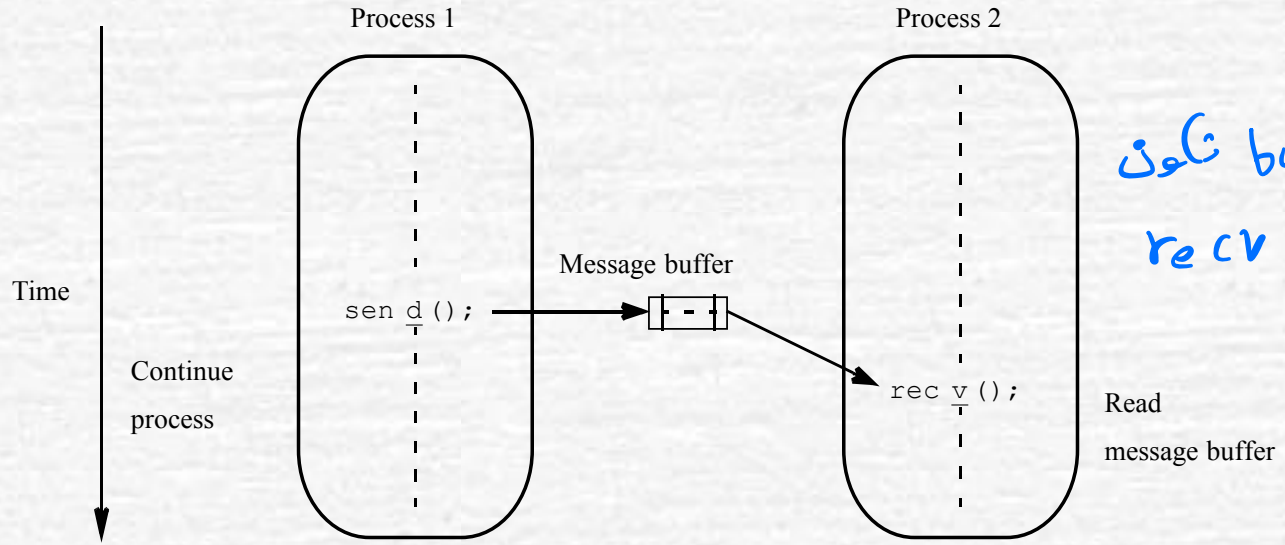
Buffered = Asynchronous Sends

- Only know when the message has left.



Buffered = Asynchronous Sends

- when the sender process reaches a *send* operation it copies the data into the buffer on the receiver side and can proceed without waiting.
- At the receiver side it is not necessary that the received data will be stored directly at the designated location.
- When the receiver process encounters a receive operations it checks the buffer for data.



عادة ال buffer اللي بتكون بتخزن في ال recv

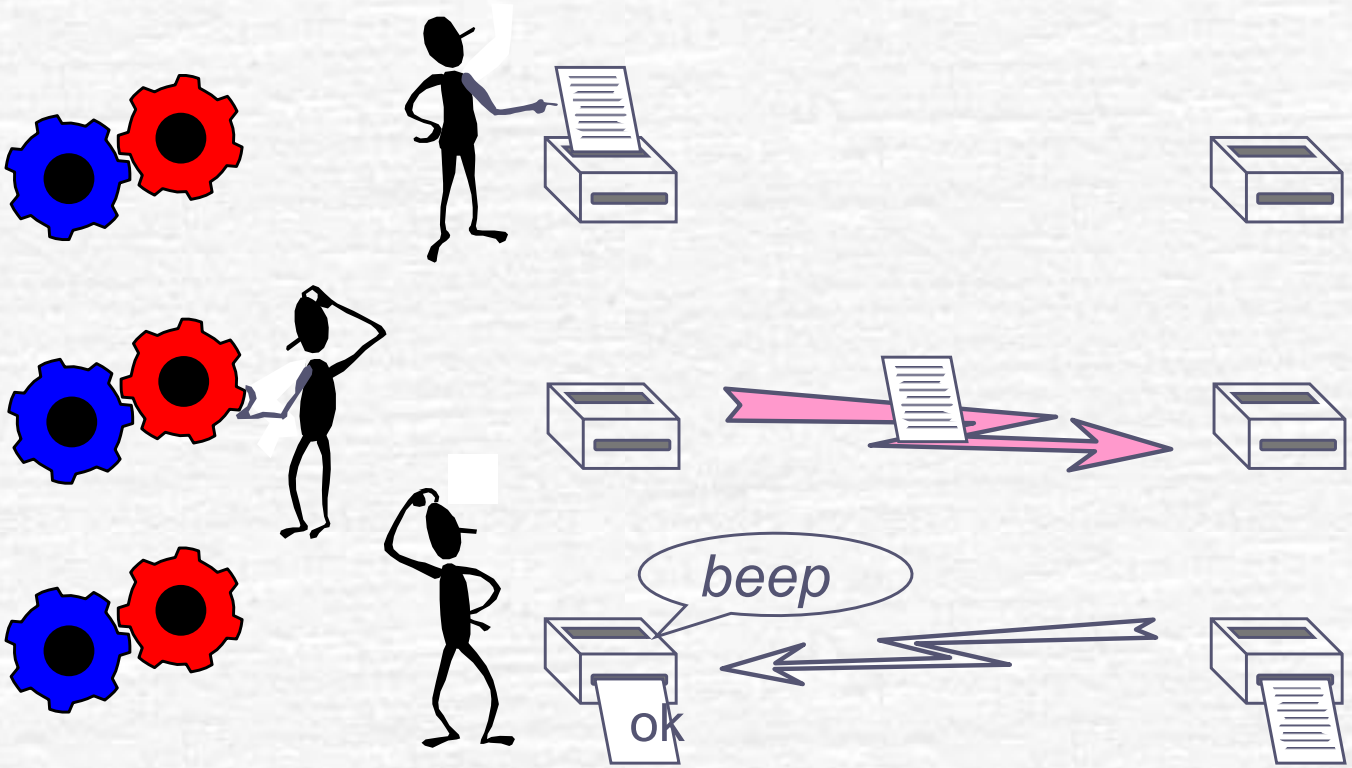
Blocking Operations

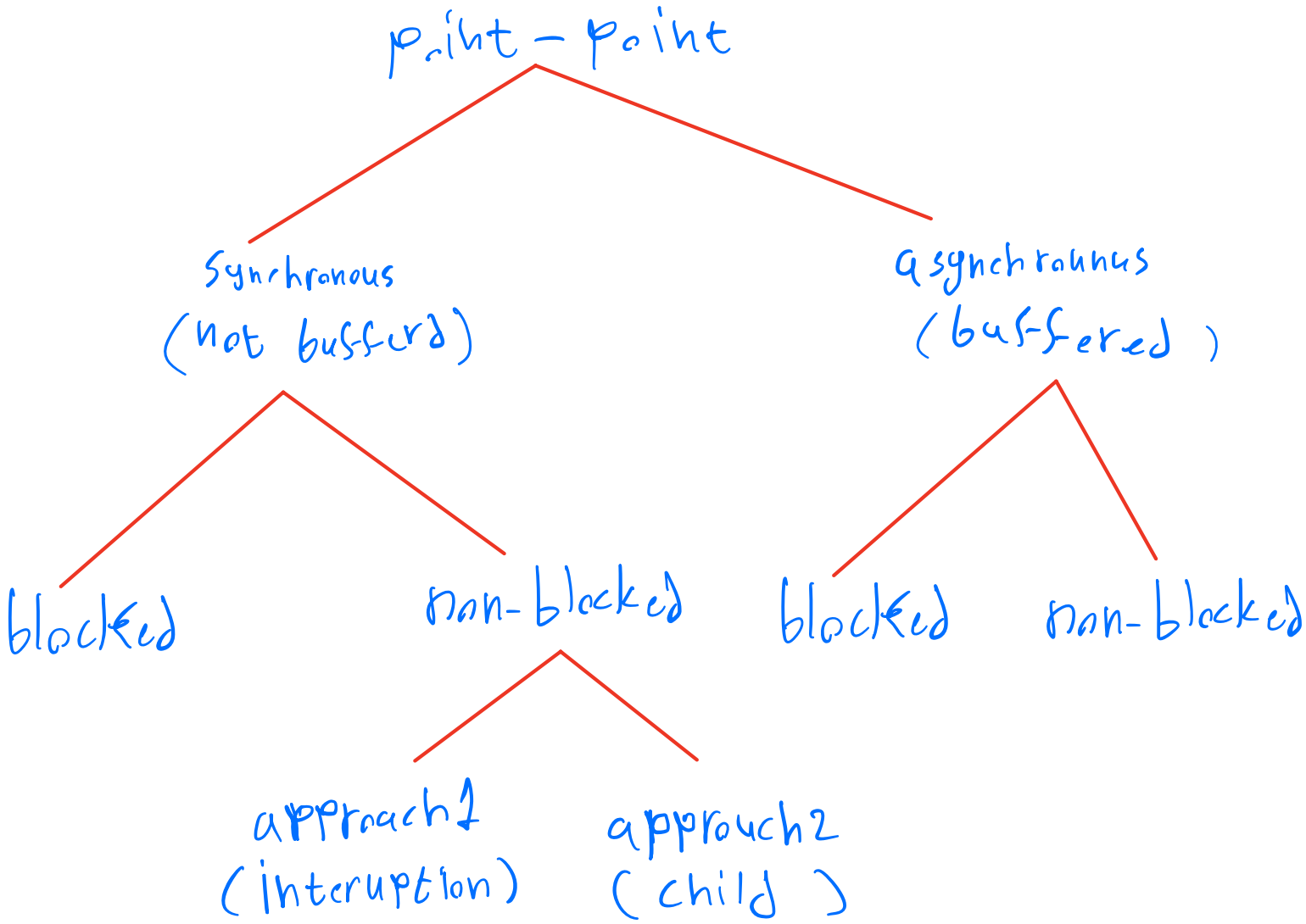
- **Blocking subroutine returns only when the operation has completed.**

- **Some sends/receives may block until another process acts:**
  - synchronous send operation **blocks until** receive is issued;
  - receive operation **blocks until** message is sent.

Non-Blocking Operations

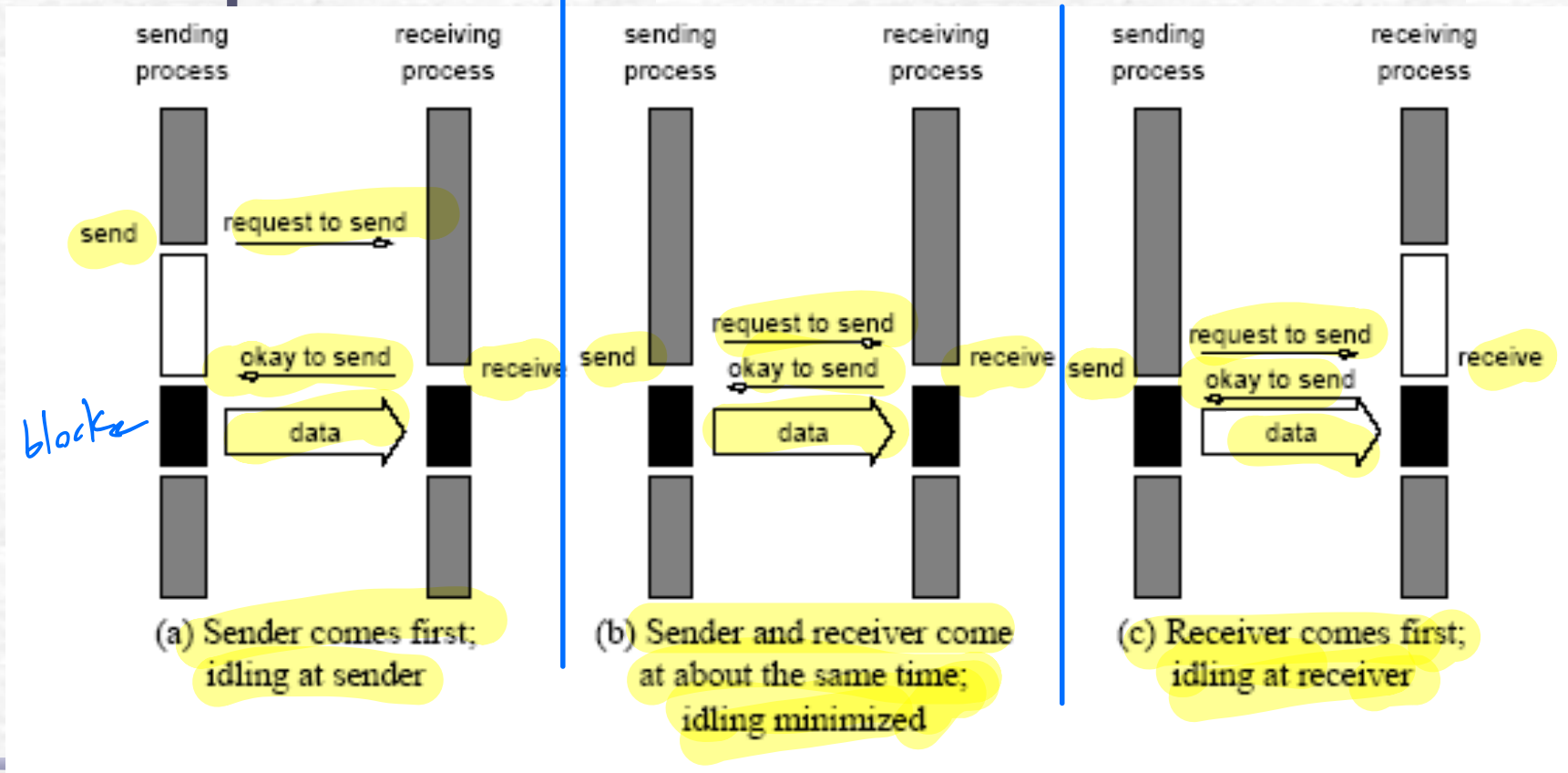
- Non-blocking operations return immediately and allow the sub-program to perform other work.




point - point
Synchronous
(not bussfurd)
asynchronous
(buffered)
blocked
non-blocked
blocked
non-blocked
approach1
(interuption)
approuch2
(child)

Blocking non-buffered send/ receive

- The sender issues a *send* operation and cannot proceed until a matching receive at the receiver's side is encountered and the operation is complete.



(a) Sender comes first; idling at sender

(b) Sender and receiver come at about the same time; idling minimized

(c) Receiver comes first; idling at receiver

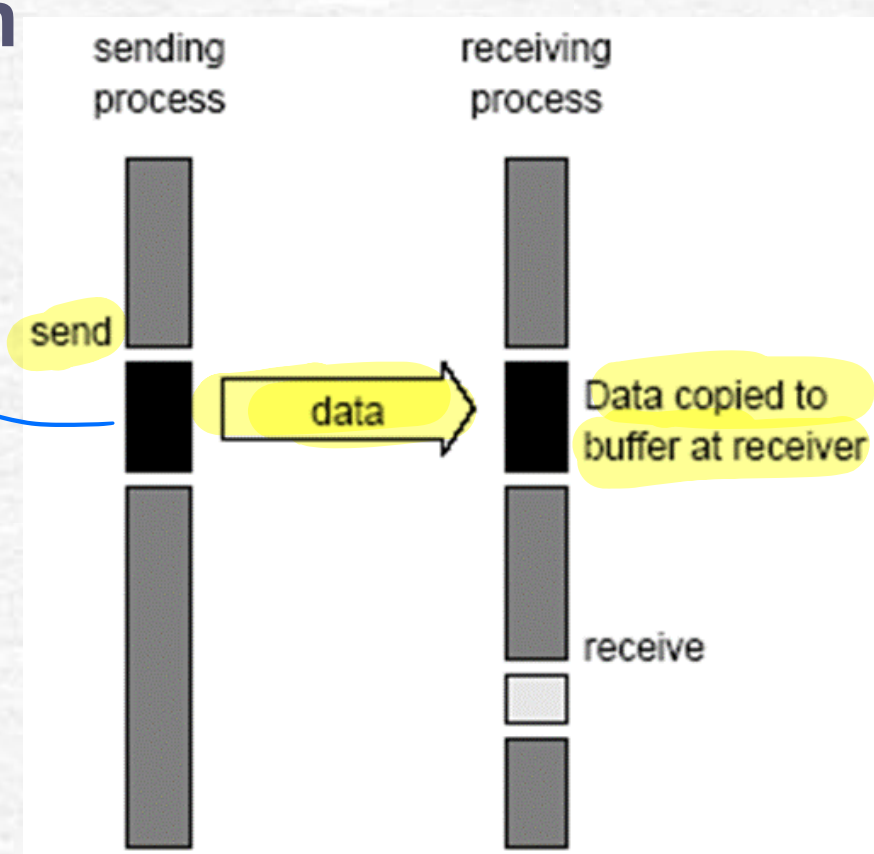
Blocking buffered send/ receive

- When the sender process reaches a *send* operation it copies the data into the buffer on the receiver side and can proceed without waiting.

اذا كان الـ buffer فاضي
الـ recive راح ينتظر

block until data send to buffer

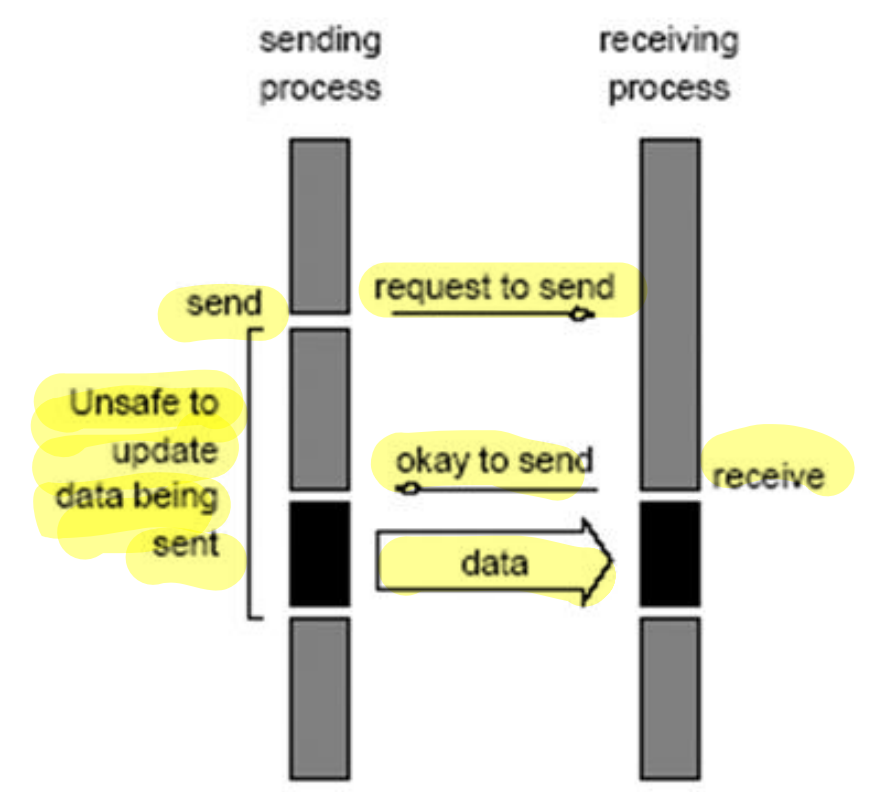
- When the receiver process encounters a receive operations it checks the buffer for data.



Non-Blocking non-buffered send/ receive

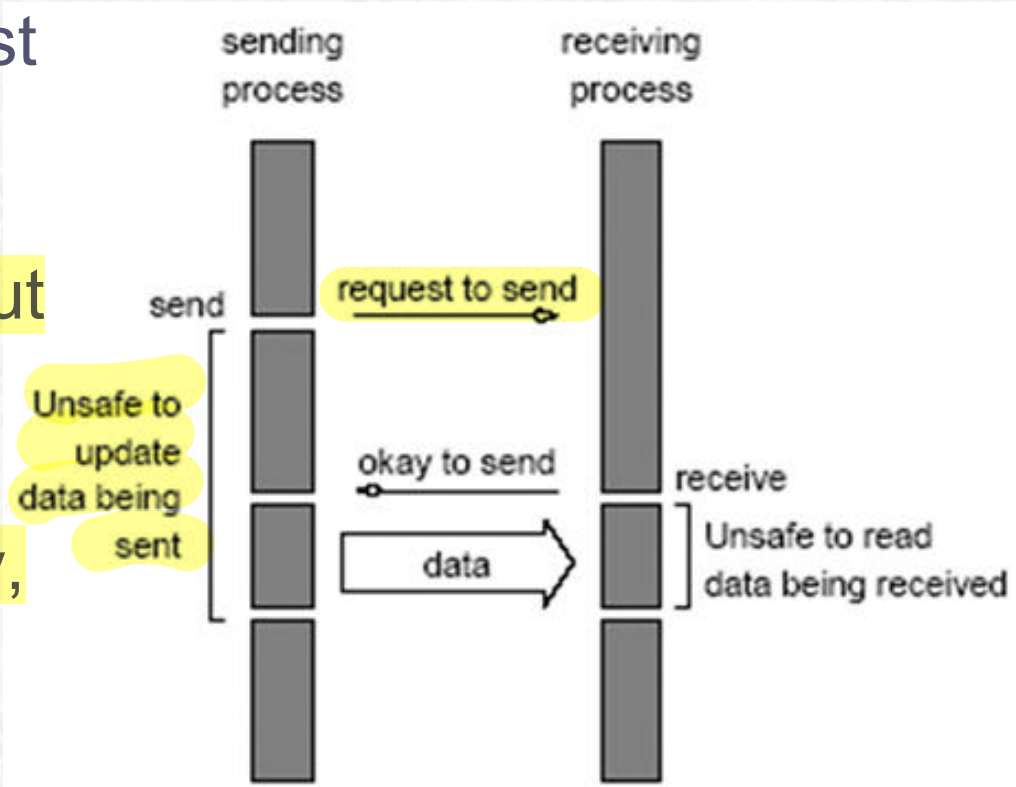
- The sender process needs not to be idle but instead can do useful computations while waiting for the send / receive operation to complete.

- **Approach 1:** interrupte
  - The sender issues a request to send and can proceed with its computations without waiting the receiver to be ready.
  - When the receiver is ready, interruption signals trigger the sender to start sending the data.



Non-Blocking non-buffered send/ receive

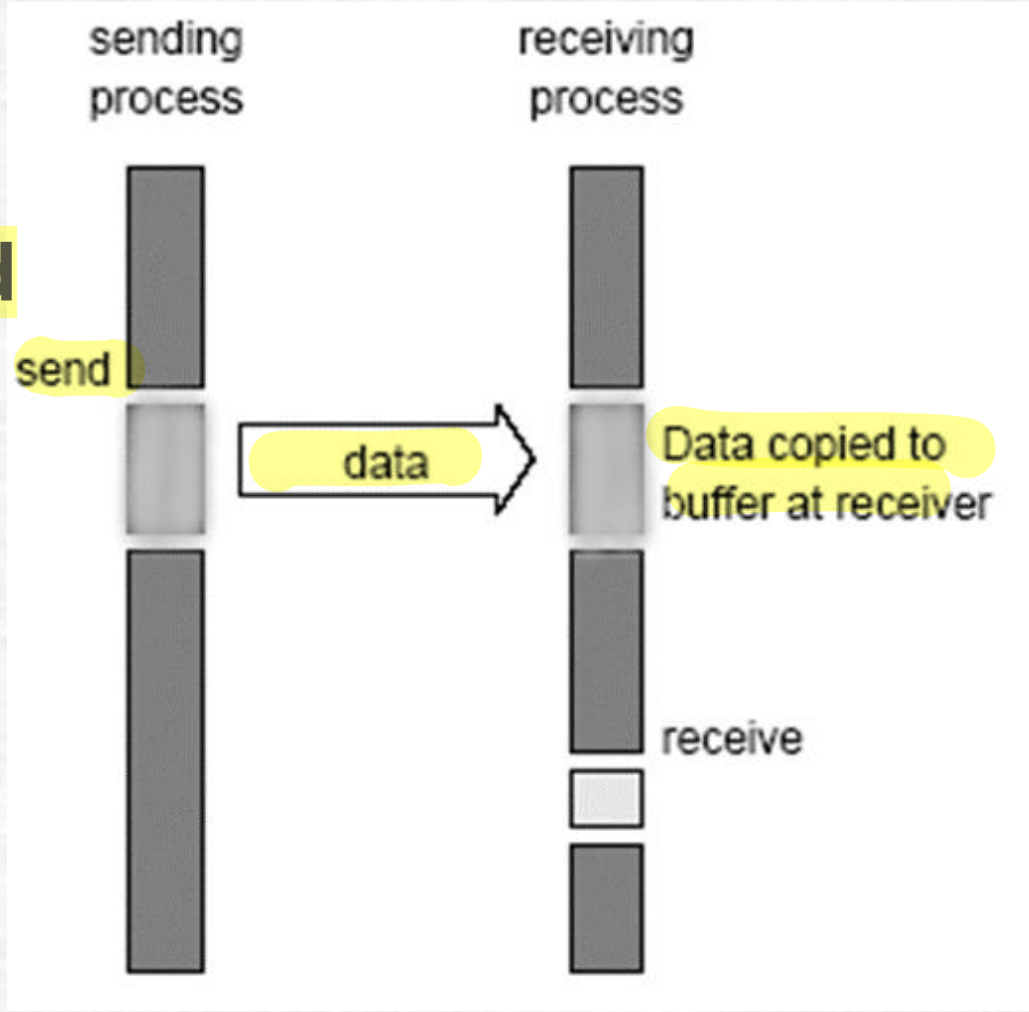
- The sender process needs not to be idle but instead can do useful computations while waiting for the send / receive operation to complete.
- **Approach 2:**
  - The sender issues a request to send, creates a child process, and .can proceed with its computations without waiting the receiver to be ready.
  - When the receiver is ready, the child process starts sending the data.



الـchild هو الي يرسل

Non-Blocking buffered send/ receive

- The sender issues a direct memory access operation (DMA) to the buffer.
- The sender can proceed with its computations.
- At the receiver side, when a receive operation is encountered the data is transferred from the buffer to the memory location.

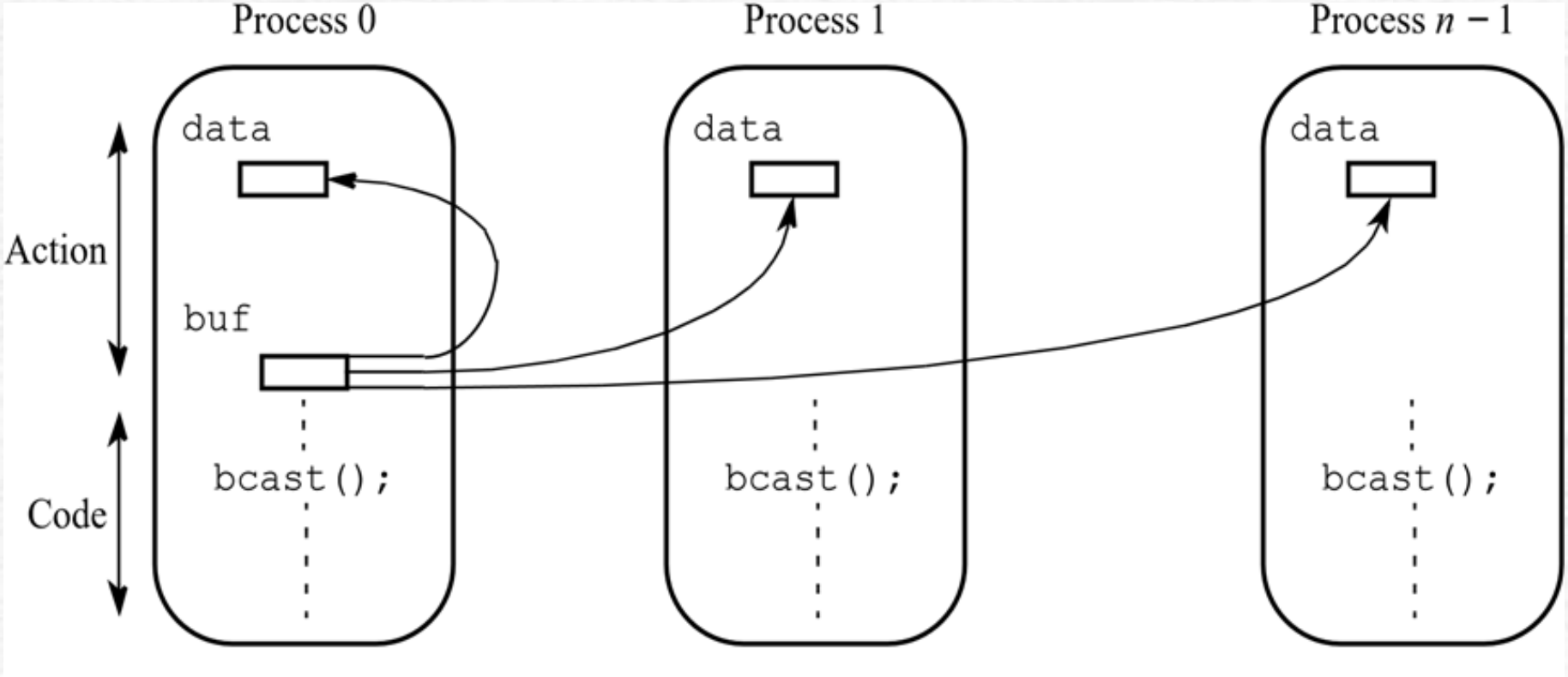


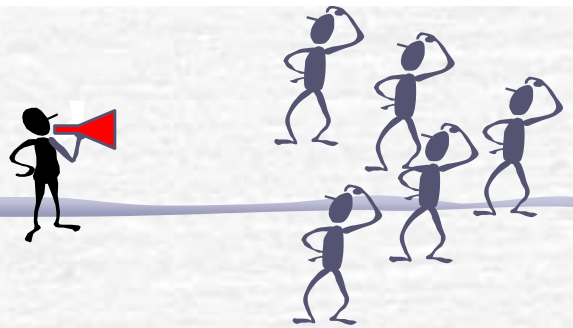
Collective Communications

- **Broadcast**
  - This function allows one process (called the root) to send the same data to all communicator members
- **Scatter**
  - Allows one process to give out the content of its send buffer to all processes in a communicator.
- **Gather**
  - Each process gives out the data in its send buffer to the root process which stores them according to their ranks.

Broadcast

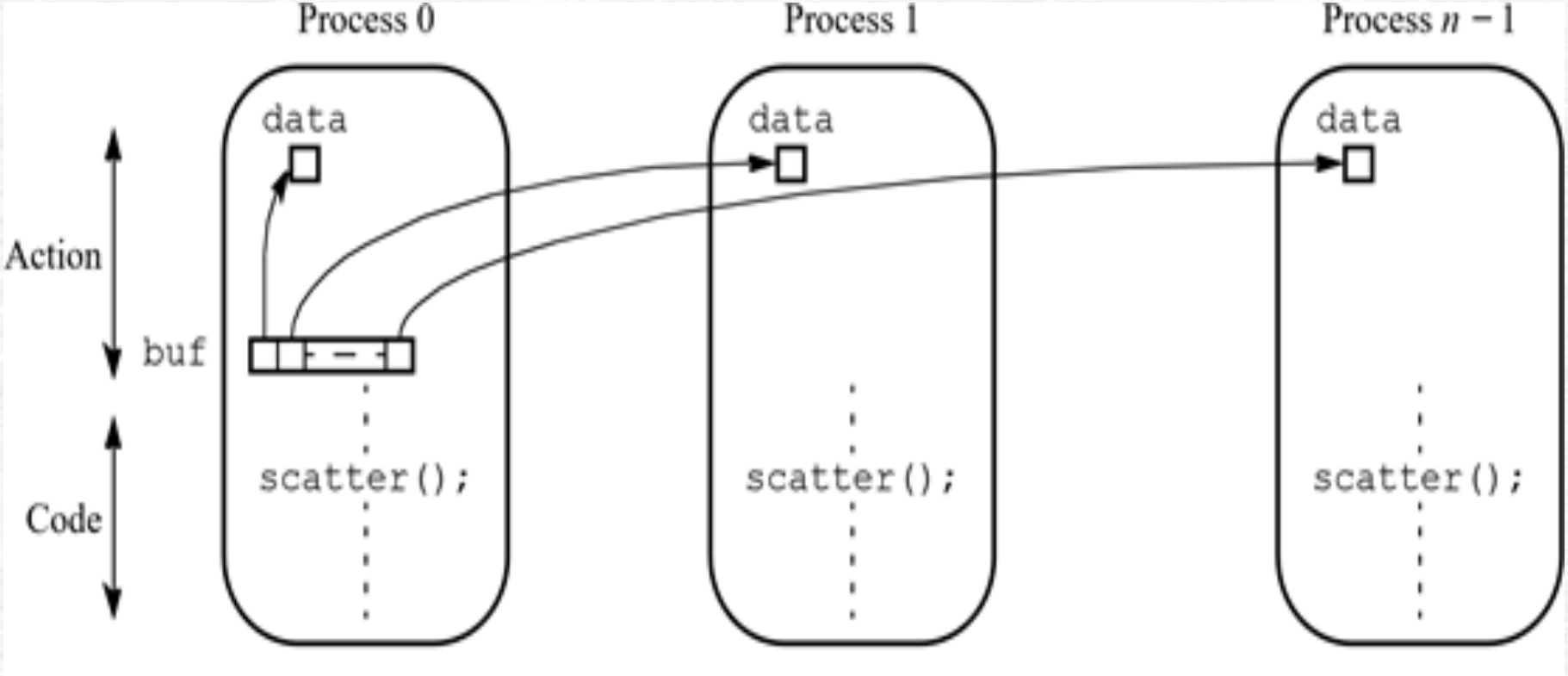
- Sending each element of an array of data in the root to a separate process.





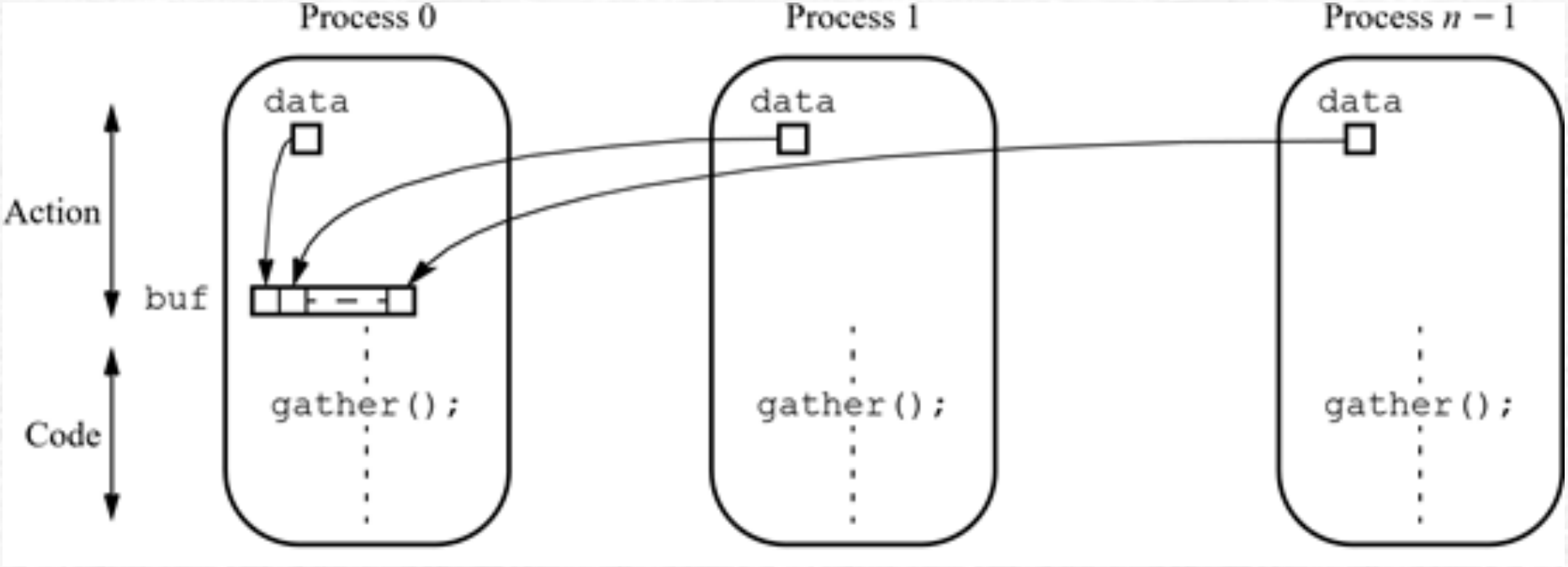
Scatter

- A one-to-many communication.
- Sending each element of an array of data in the root to a separate process.



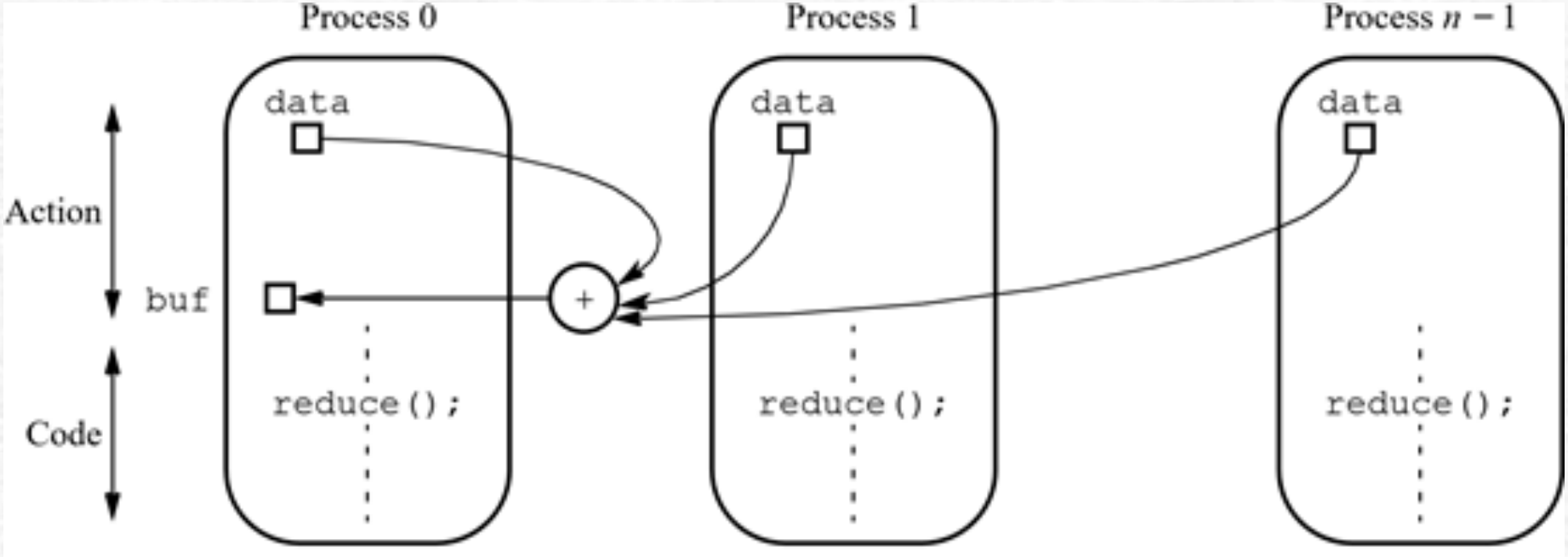
Gather

- Having one process collect individual values from a set of processes.



Reduction

- Gather operation combined with a specified arithmetic or logical operation.



Multi-threading Paradigm

- **In a single-core (superscalar) system,**
  - we can define multithreading as the ability of the processor's hardware to run two or more threads in an overlapping fashion by allowing them to share the functional units of that processor.
- **in a multi-core system,**
  - we can define multithreading as the ability of two or more processors to run two or more threads simultaneously (in parallel) where each thread run on a separate processor
- **Modern systems combine both multithreading approaches.**