


CLASSIFYING
COMPUTERS –
HOW, WHEN AND
WHY

FLYNN'S TAXONOMY

WHAT IS A TAXONOMY?

taxonomy [tak-son-uh-mee] [SHOW IPA](#) 

[SEE SYNONYMS FOR taxonomy ON THESAURUS.COM](#)

noun, plural tax-on-o-mies.

- 1 the science or technique of classification.
- 2 a classification into ordered categories:
a proposed taxonomy of educational objectives.
- 3 *Biology.* the science dealing with the description, identification, naming, and classification of organisms.

WHY SHOULD
WE CLASSIFY
COMPUTERS?

1.To understand what has already been achieved

Computers have come a long way today from Babbage's Analytical engine, and we need to establish how we can systematically study them.



2.To reveal possible configurations that may not have occurred to a systems designer

Only after classifying the existing solutions and studying them, the gaps can be identified which can suggest other possibilities.

3.To allow useful models of performance to be built and used

Classification on a distinct basis may reveal why a particular architecture is likely to improve performance.

Characterization in scrutiny serves as a model for performance analysis.

Our primary interest in a parallel computer taxonomy is to :

Help understand pertinent issues raised by parallel computing hardware

Quantify completeness of argument – By analyzing every leaf in a taxonomy, we can quantify completeness of argument (in applicability of technology, etc.)

Michael J. Flynn

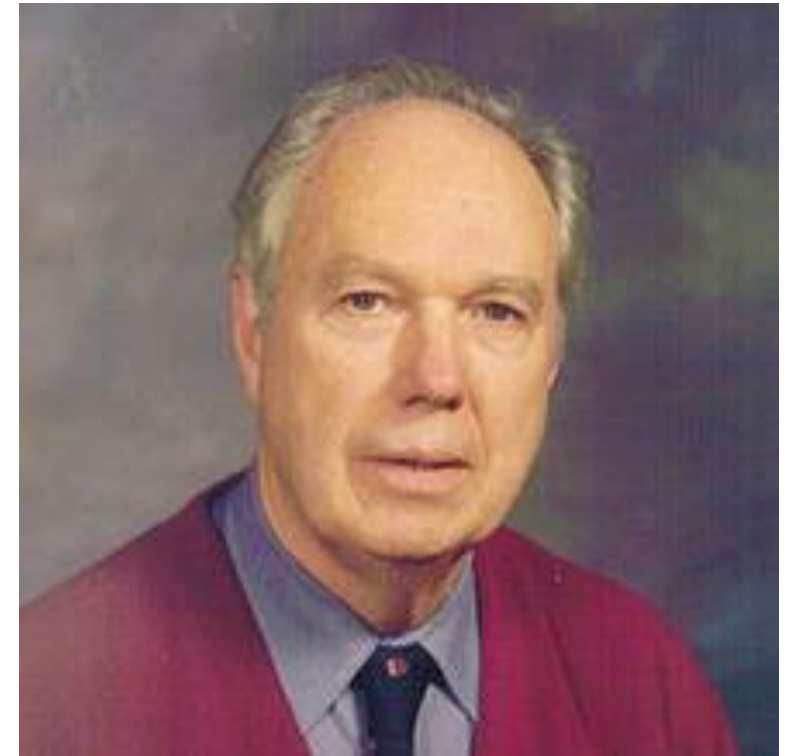
American Professor emeritus, Stanford University

Founded IEEE Computer Society's technical committee on Computer Architecture

Founded ACM's Special Interest group on Computer Architecture

Winner of prestigious Harry H. Goode Memorial Award, Eckert-Mauchley Award and Tesla Medal

Most known for proposing **Flynn's Taxonomy**.



What is Flynn's Taxonomy?

- A stream is defined as a sequence of items.
- On the basis of the instruction stream and the data stream, Flynn classifies computers into 4 categories :



SISD

SIMD

MISD

MIMD

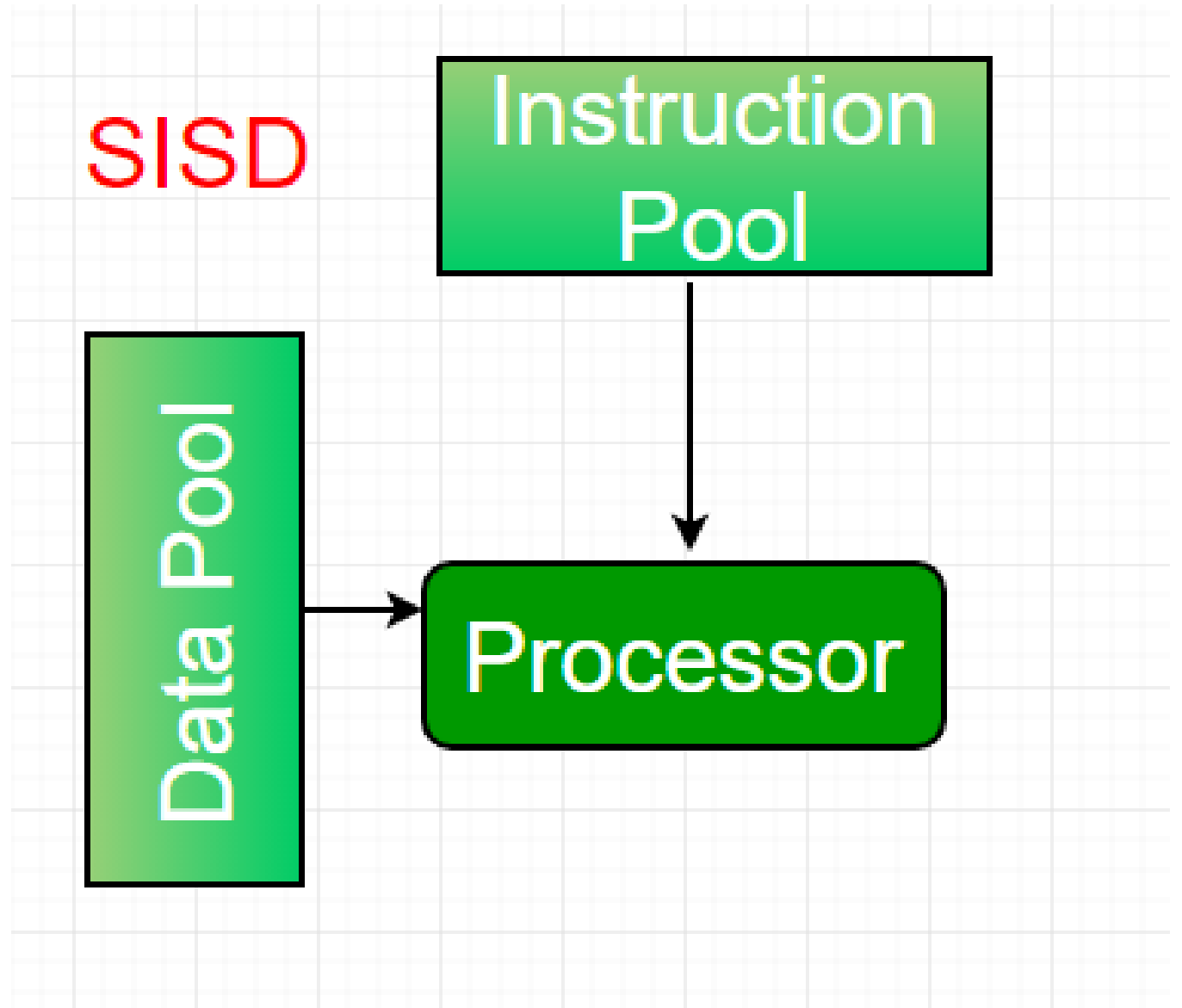
SISD – Single Instruction, Single Data

Uniprocessor machine

Processes instructions sequentially

All data has to be stored in the primary memory

E.g. Von Neumann Architecture



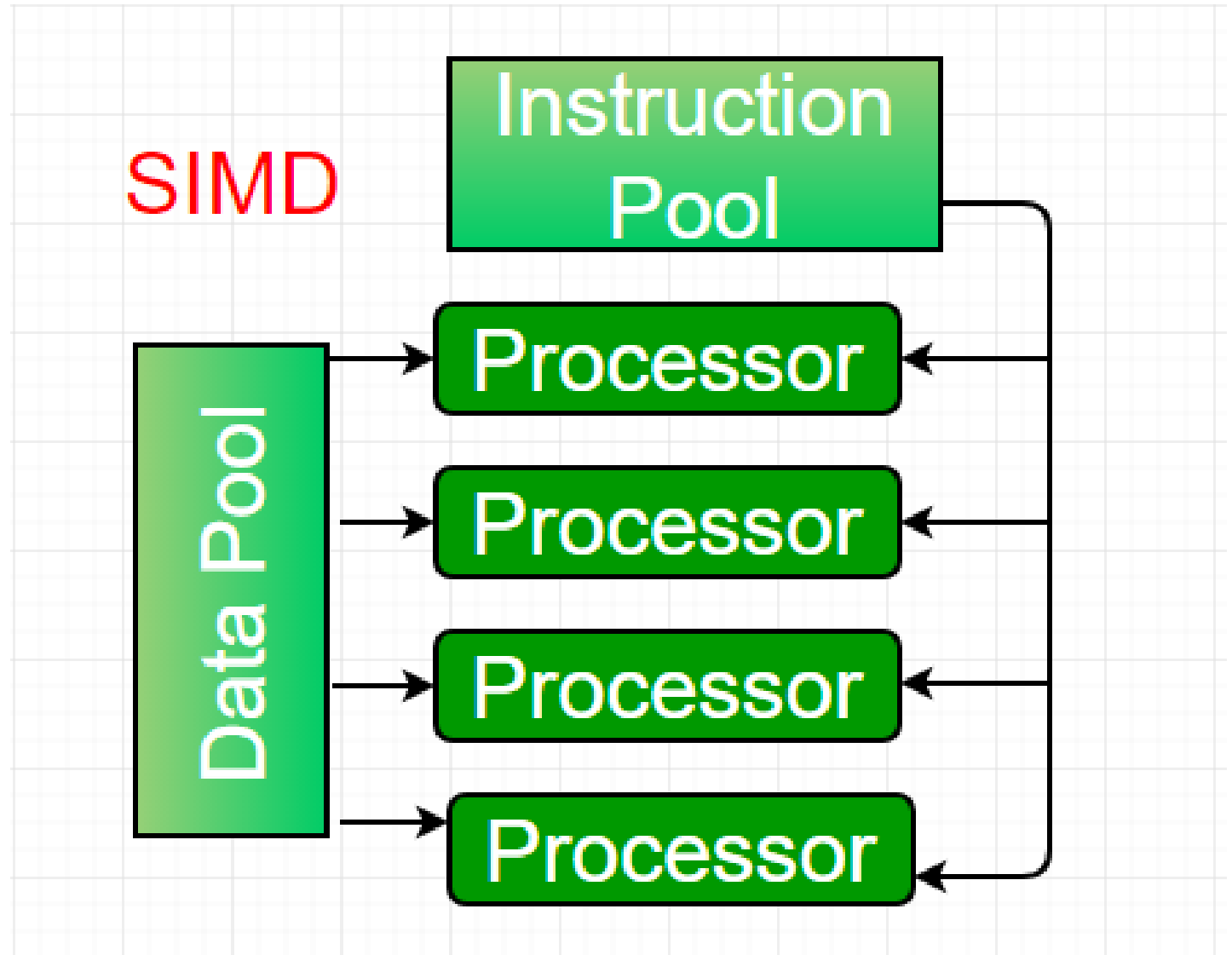
SIMD – Single Instruction, Multiple Data

Multiprocessor machine

Multiple processing elements perform same operation on multiple data points simultaneously

Well-suited for vector and matrix operations

E.g. CRAY's Vector Processing Machine



How SIMD works

Instead of concurrency, SIMD machines exploit data-level parallelism



Data-level parallelism is a form of parallel computing where data is distributed across different parallel nodes for simultaneous computation



Since only one instruction is executed across multiple data, it is used prevalently in audio/image processing, improving performance in multimedia usage, etc.

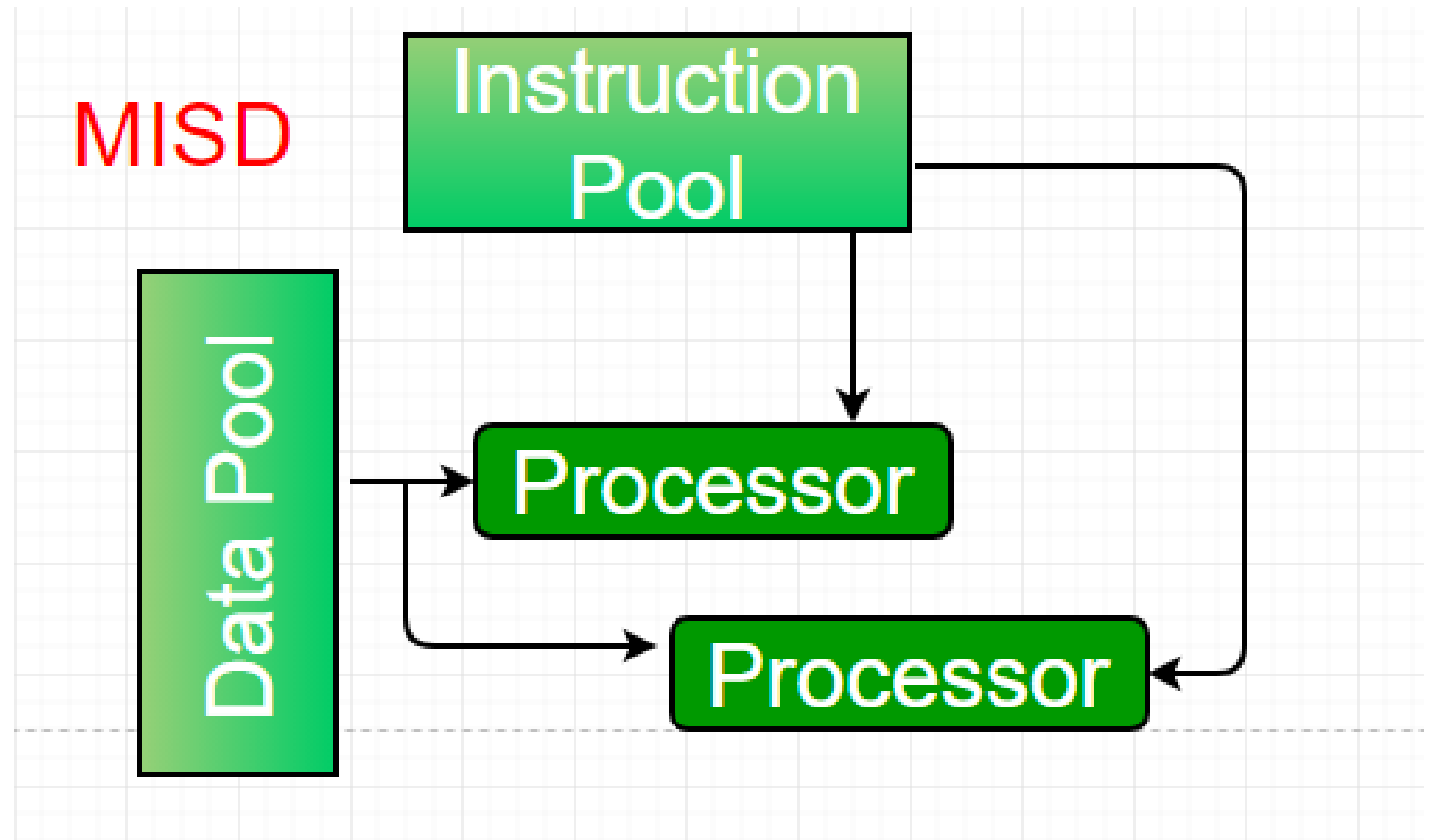
MISD – Multiple Instruction, Single Data

Multiprocessor machine

Same dataset, different instructions, e.g. $z = \sin(x) + \cos(x) + \tan(x)$

Non-existent in any commercial usage

Added to quantify completeness



MIMD – Multiple Instruction, Multiple Data

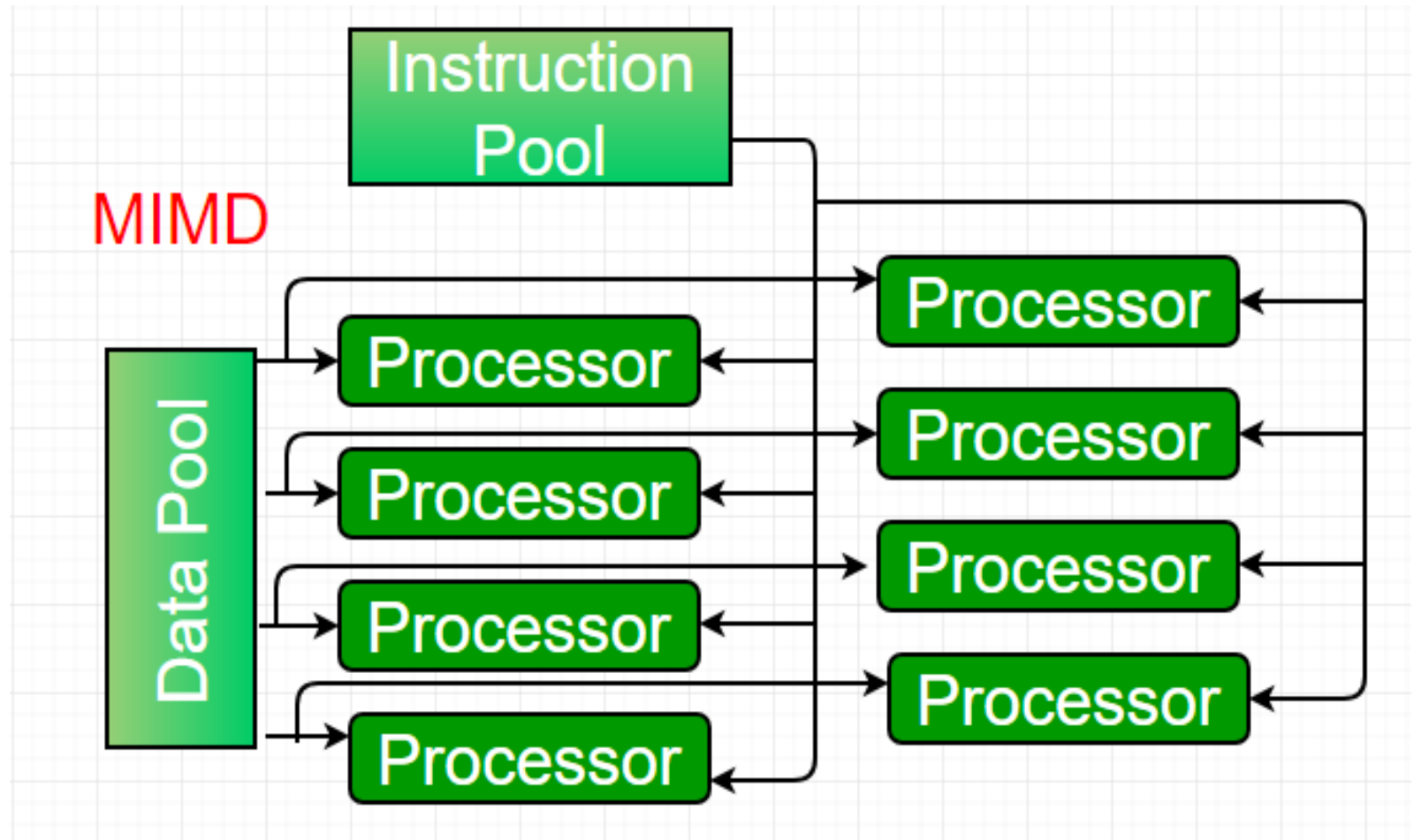
Multiprocessor machine

Processing elements work
asynchronously and
independently

At any point of time, different
processing elements perform
different operations on different
data

Capable to any kind of
application

E.g. Intel Xeon Phi




MIMD is further subcategorized into :

Shared-
Memory
MIMD

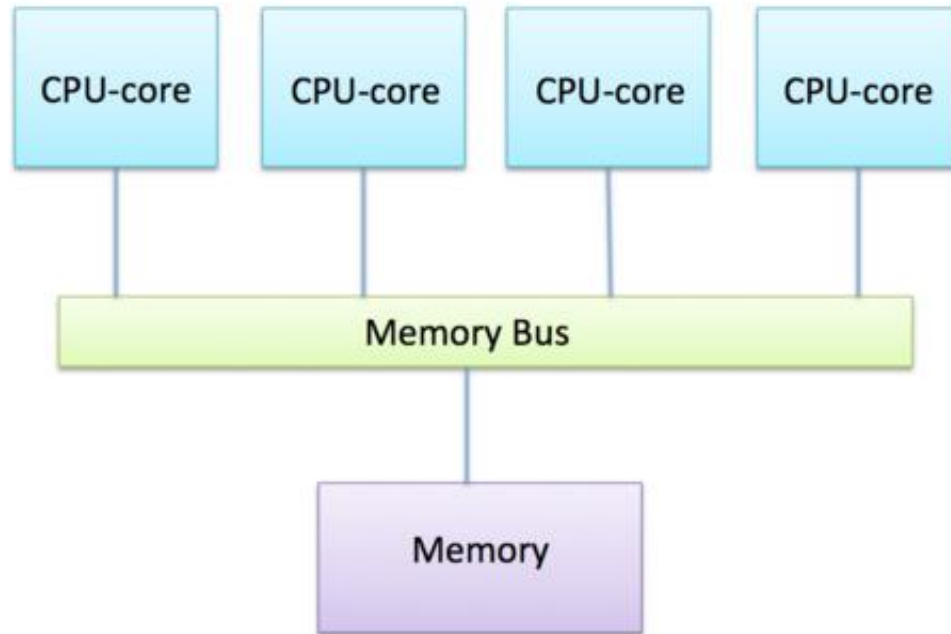
Distributed
Memory
MIMD

Shared-Memory Architecture

Multiple processors operate independently, but share the same memory resources



Changes made in a memory location by one processor is visible to all other processors



How Shared-Memory Architecture works

A special fast inter-connection network allows any processor to access any part of the memory in parallel

Processors communicate via main-memory, making it very efficient

Since all processing elements come under the same Operating System, load balancing is also taken care of

Shared-Memory
Architecture can
be further
subcategorized
into :

Uniform Memory
Access (UMA)

Non-Uniform Memory
Access (NUMA)

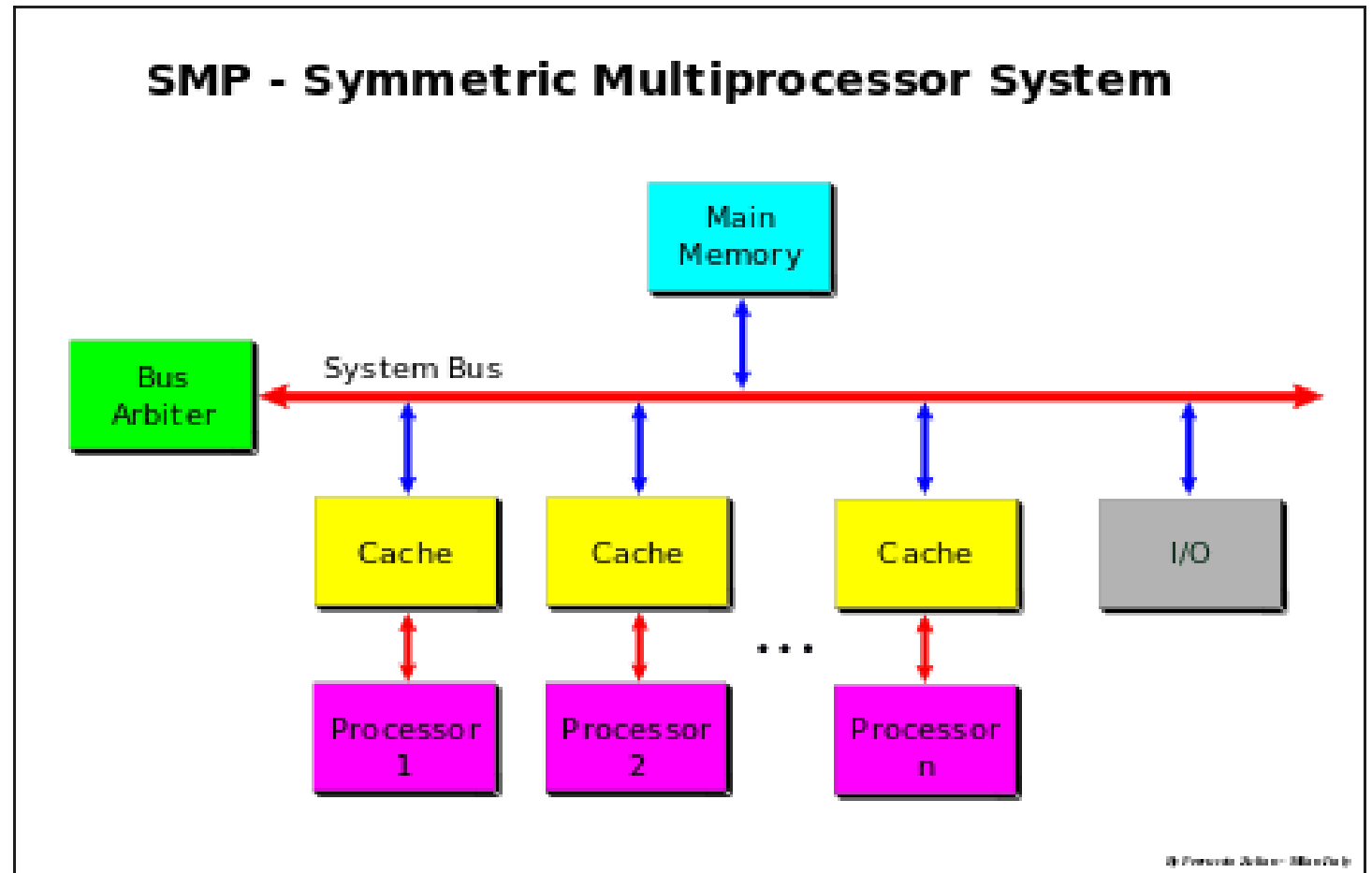
SMP – Symmetric Multiprocessing

A computer hardware/software architecture where multiple cores/processors are :

Connected to a single, shared main-memory

Have full access to all input/output devices

Are connected to a single Operating System that treats all cores/processors equally

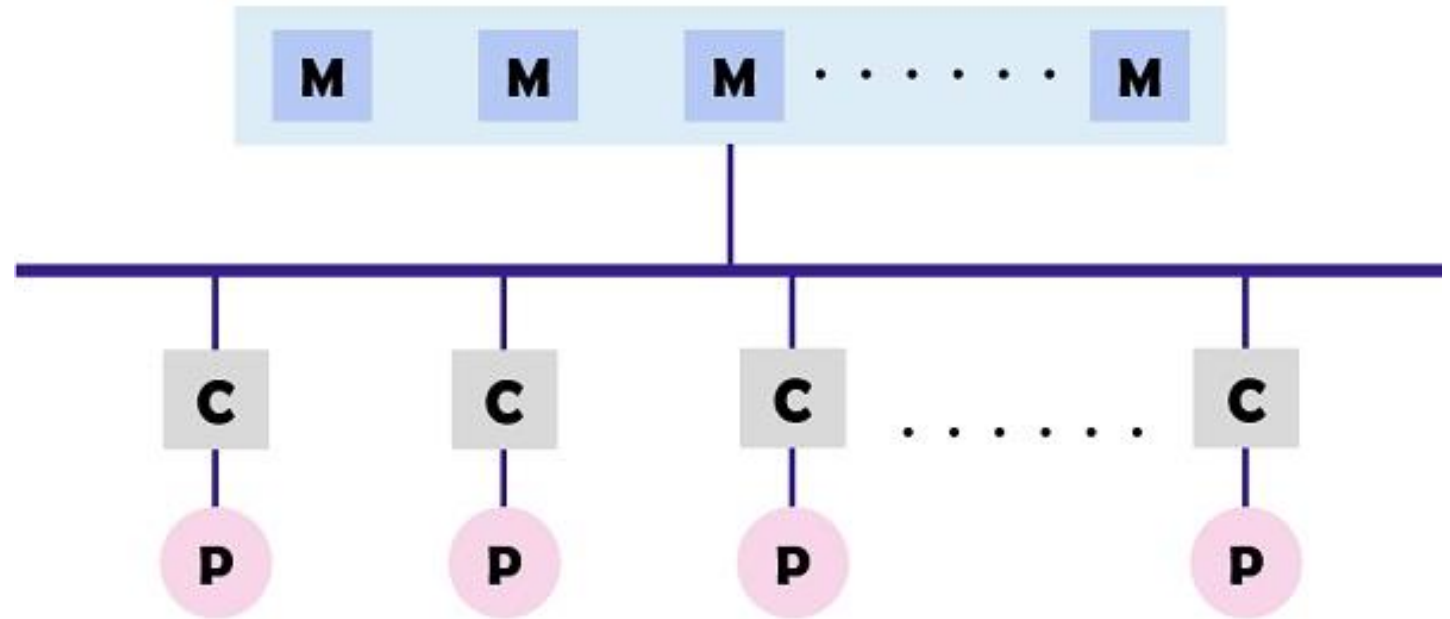


Uniform Memory Access (UMA)

Most prevalently implemented
as bus-based SMP machines

Memory Access time is
balanced/equal

UMA is cache coherent – If one
processor updates a location in
shared memory, all other
processors know about the
update



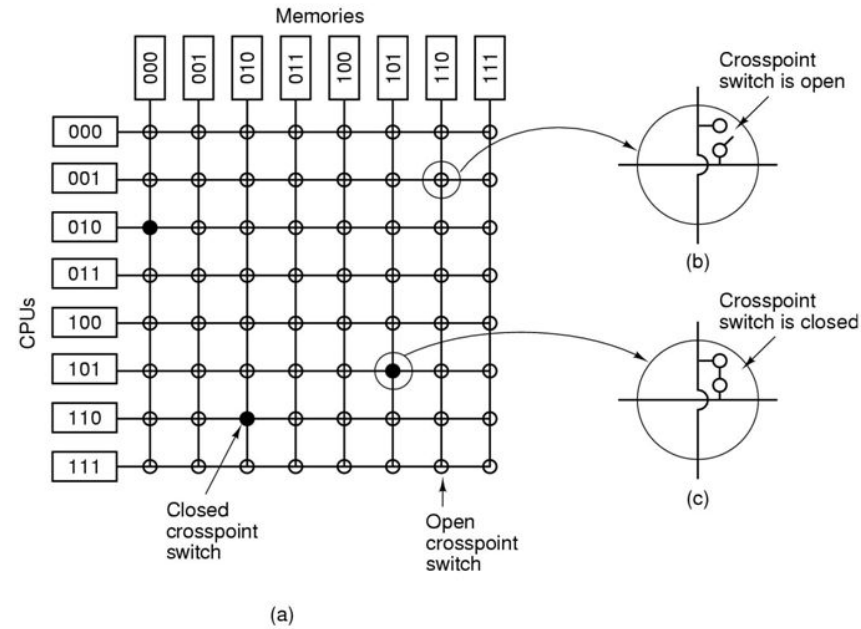
Bus-based UMA (SMP) Shared Memory

UMA can also be implemented by :

Using Crossbar
switches

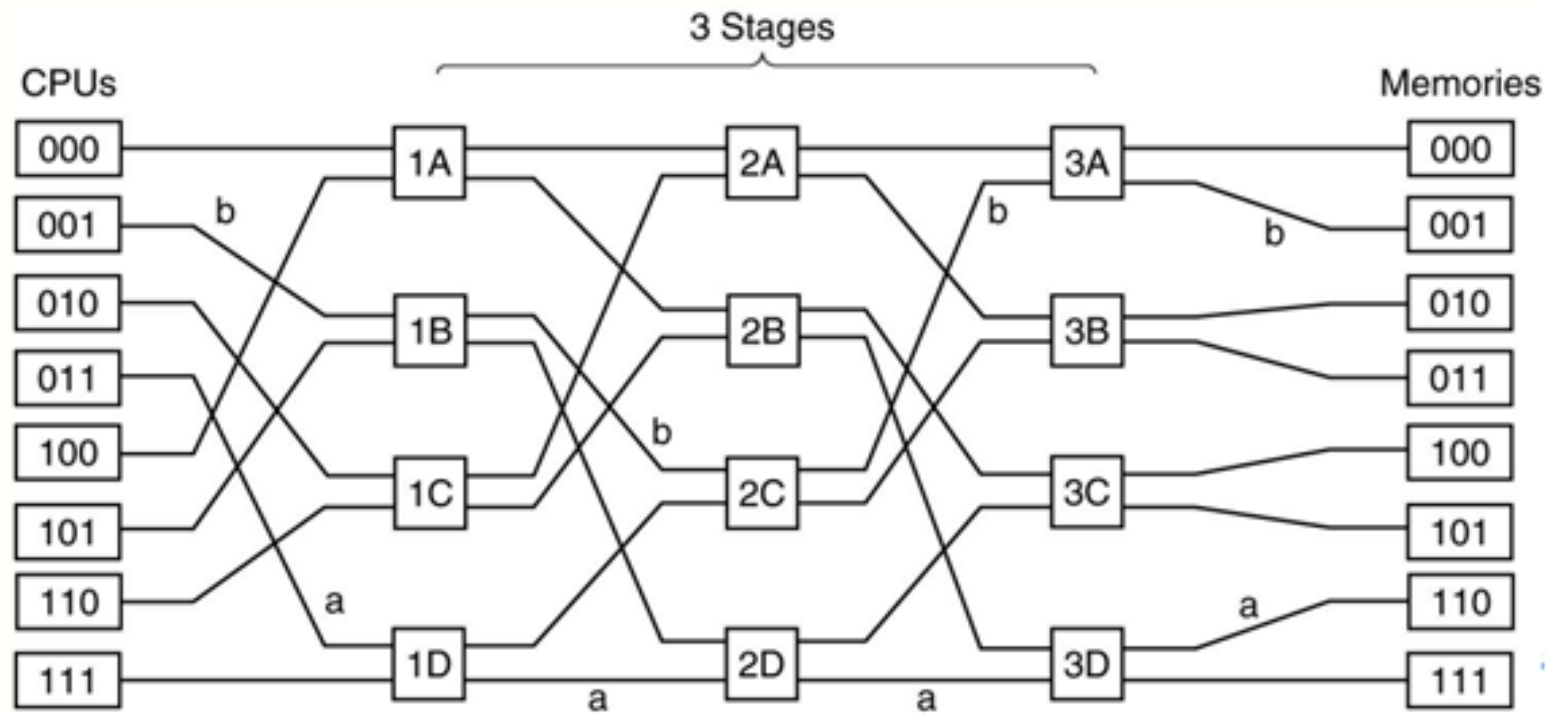
Using Multistage
interconnection
networks

Non-blocking network



UMA Multiprocessor using a crossbar switch

UMA using crossbar switches



UMA using Multistage Interconnection Networks

hUMA – Heterogeneous UMA

Relatively new concept, surfaced around 2013

Refers to CPU and GPU sharing same system memory with cache coherency

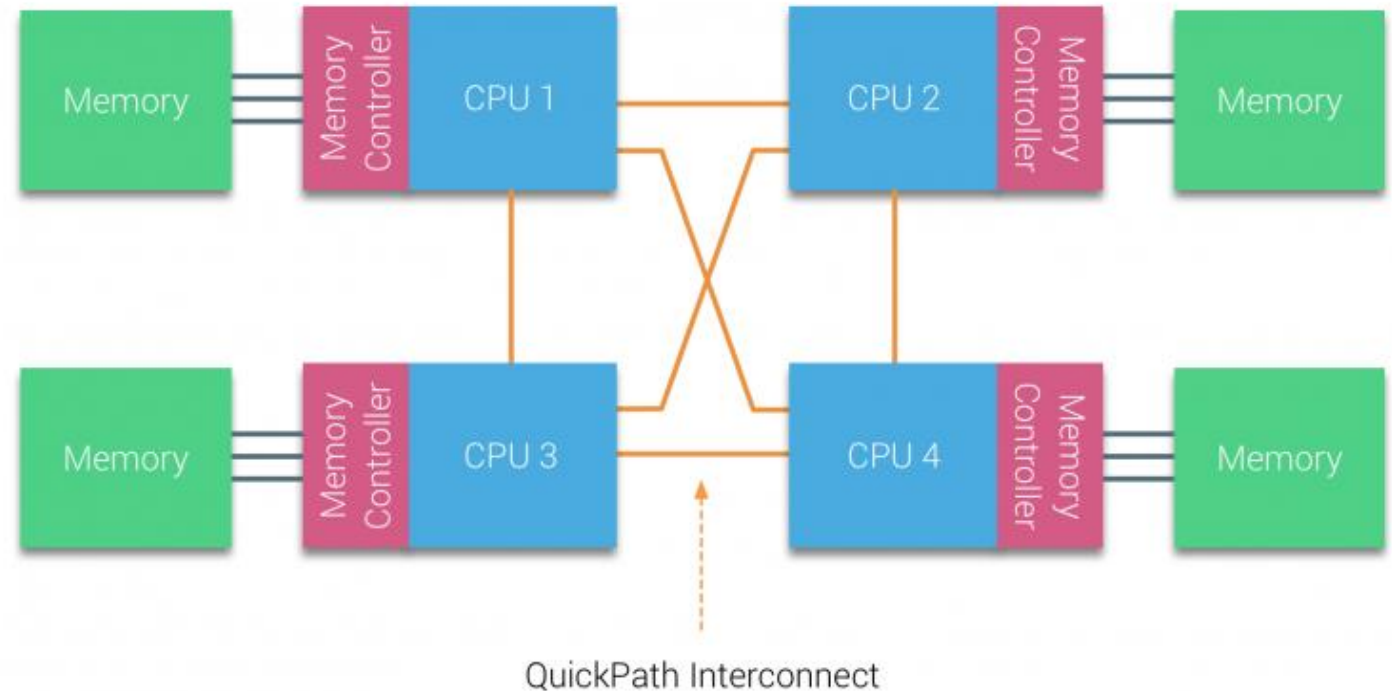
Advantageous because of an easier programming model and less copying of data between memory pools

Non-Uniform Memory Access (NUMA)

Physical linking of 2 or more
SMPs, one can directly access
the other

Memory Access time is
dependent on memory location
relative to the processor

If cache coherency is
maintained, it is called CC-
NUMA



Shared-Memory Architecture

PROS

- User-friendly programming perspective
- Data sharing between tasks is fast and uniform

CONS

- Lack of scalability in memory and CPUs
- Programmer is responsible for synchronization

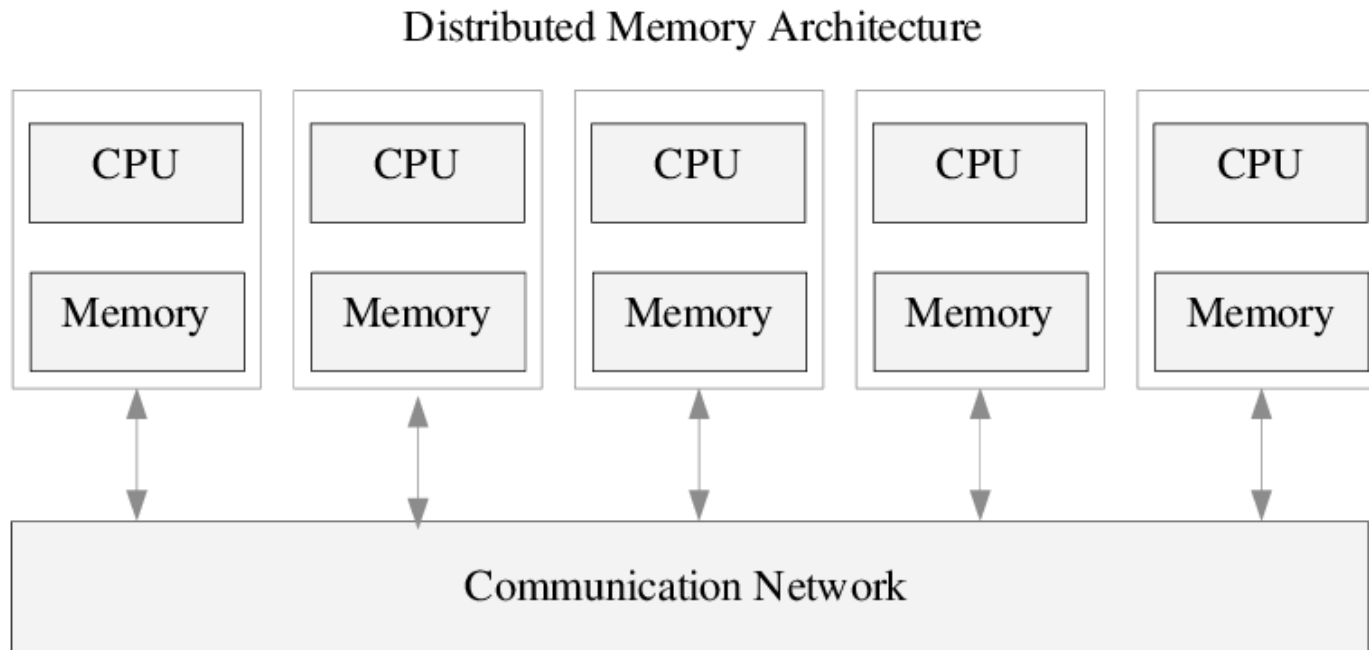
Distributed-Memory Architecture

Multiprocessor system where processors have own local memory and operate independently



A communication network is used to access inter-connected memory

How Distributed-Memory works



Computational tasks can only use local data

If remote data is required, the program must communicate with the remote processors

Processors need not be aware of where data resides

The interconnect is in the form of point-to-point links, or a switching network

Distributed-Memory Architecture

PROS

- Memory is scalable with number of processors
- Processors can rapidly access own memory without interference
- No overhead of maintaining global cache coherency
- Cost Effective

CONS

- Programmer is responsible for details of communication between processors
- Difficult to map existing data structures based on global memory to this memory organization
- Data on remote node has longer access time than local data

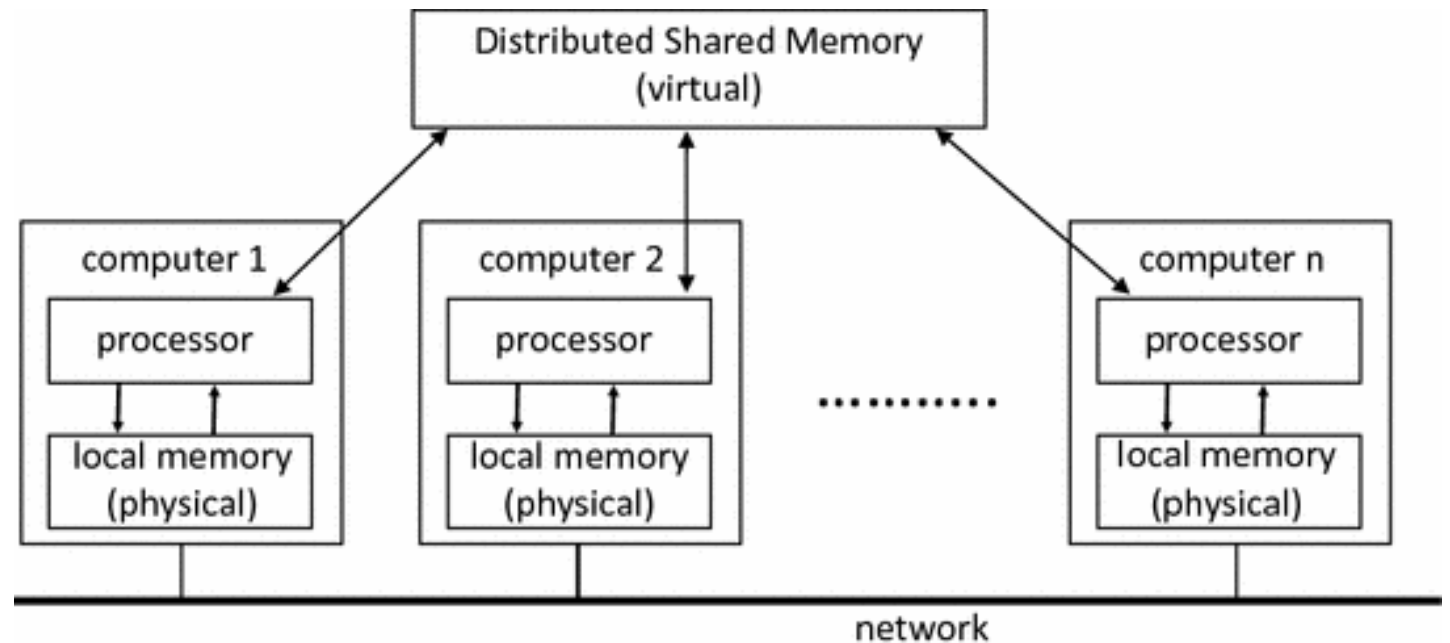
Distributed-Shared Memory Architecture

There is also a hybrid Distributed-Shared Memory Architecture

Shared-Memory component may be a Shared-Memory machine/GPU

Distributed-Memory component may be networking of multiple Shared-Memory machines/GPUs, that only know of their own memory

Hides mechanism of communication, but does not hide latency of communication



Takeaway from using each Memory Architecture

Shared-Memory

- Unified memory address space where all data can be found
- Highly efficient data sharing
- Programmer-friendly

Distributed-Memory

- Excludes race conditions
- Programmer has to think about data distribution as well as communication between processors

Shared-Distributed Memory

- Very easy to design a machine that scales with the algorithm
- Very high complexity for programmer

Modification And Updation of Flynn's Taxonomy

AS COMPUTERS
EVOLVED, THERE
CAME SOME
DESIGNS THAT
COULD NOT BE
ACCOMODATED. SO
FLYNN'S TAXONOMY
WAS MODIFIED WITH
2 MORE
CATEGORIES...

- The most important and most significant modification to Flynn's taxonomy was appending synchronization at the instruction level with synchronization at the program level
- A program is defined as a set of instructions followed to perform a particular task
- In this context, two more categories stemmed out in MIMD :

SPMD

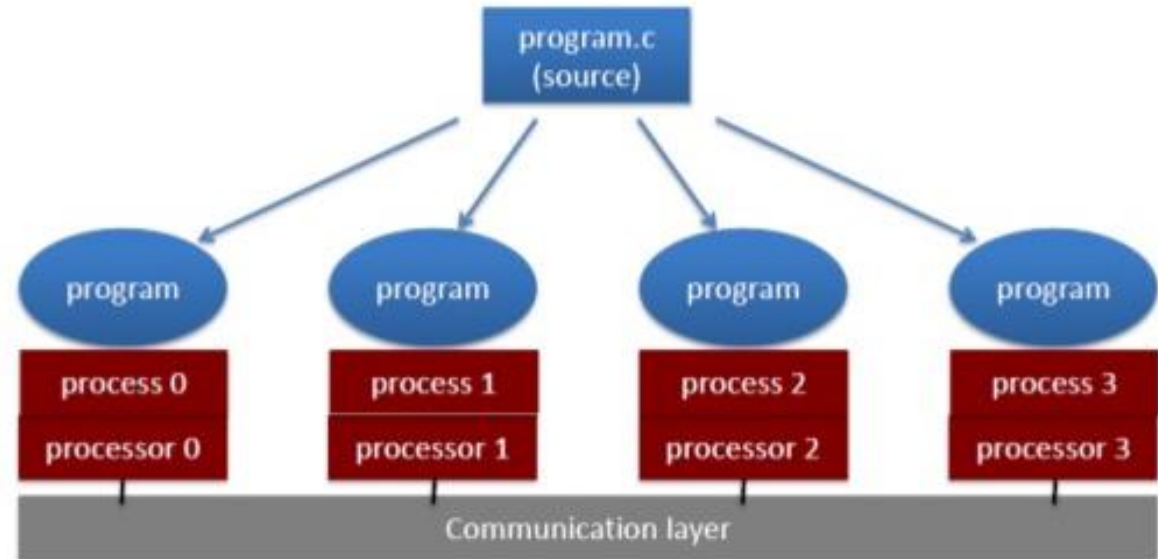
MPMD

SPMD – Single Program, Multiple Data

Synchronization is established at the level of programs

While multiple cores/CPU's run the same program, they can execute different instructions of a program at the same time

Most common style of parallel programming

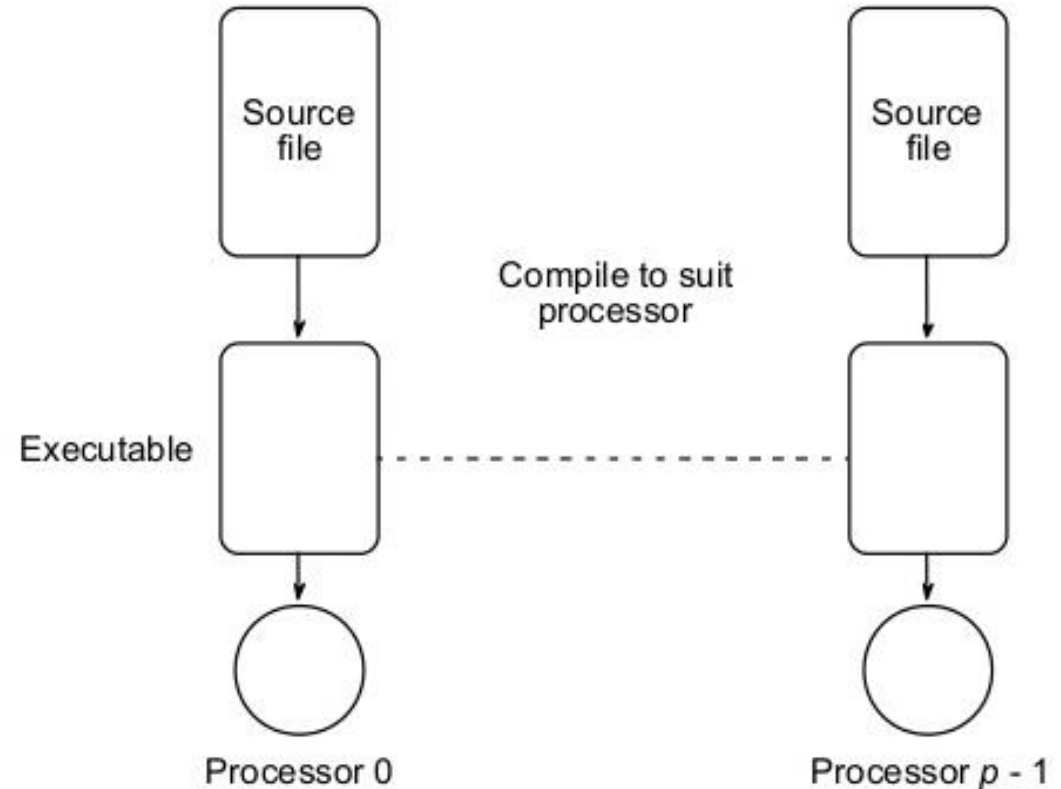


MPMD – Multiple Program, Multiple Data

Multiple autonomous processors operate on at least two independent programs simultaneously

Typically, one node acts as the "master" that runs one program, farming out data to all other "worker" nodes, which run a different program and return the results to the master node

- Different programs executed by each processor



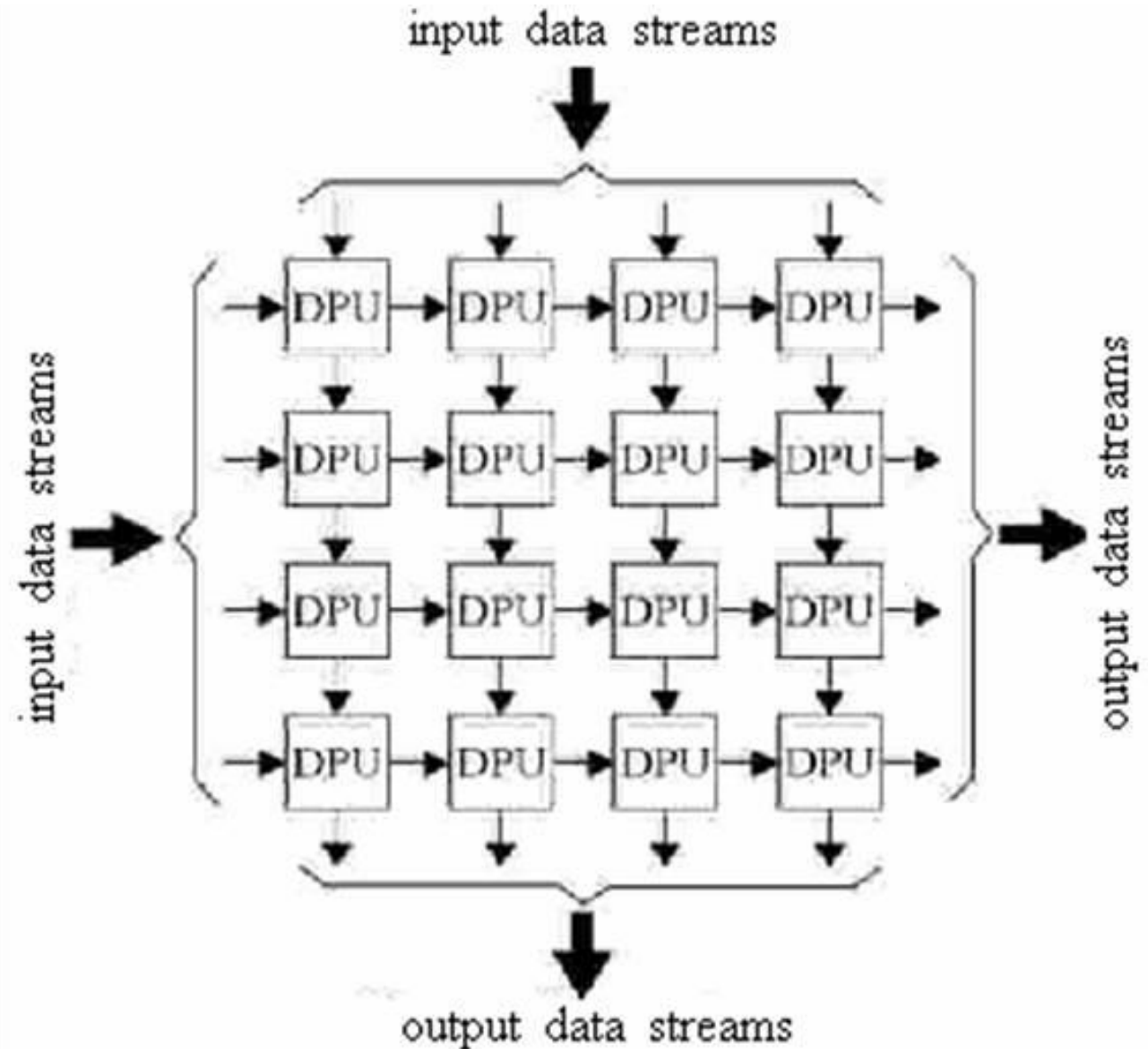
Ambiguities in Flynn's Taxonomy

JUST LIKE ANY
TAXONOMY, FLYNN'S
TAXONOMY HAS A
FEW DEBATABLE OR
CONTROVERSIAL
ARGUMENTS...

Systolic Arrays

Homogeneous network of tightly-coupled processors/nodes

Each node receives data from upstream, independently computes a partial result, stores the result within itself, and passes the data downstream



MISD – problematic classifications

Systolic Arrays are prevalently classified as MISD



This classification is problematic because since the data swarm is transformed as it passes through each node



Therefore the processors do not act on the same data, making the classification a misnomer

What do Systolic Arrays come under?

Since the input is typically a vector of independent values, the systolic array is definitely not SISD

Since these values are merged and combined into the result(s) and do not maintain their independence as they would in SIMD, the array cannot be classified as such

Consequently, the array cannot be classified as a MIMD either, because MIMD can be viewed as a mere collection of smaller SISD and SIMD machines

This kind of a misnomer also exists with pipeline architecture being classified as MISD

Systems classified as MIMD

Technically, every standard desktop computer today is an MIMD system, with each core/processor being SISD or SIMD

But **purists** argue for MIMD as only when parallelism is implemented at the hardware level(with thousands of cores/processors)

But such types of systems are not commercially available in any market, and only exist as research projects

Timelines of Computing Systems

AND HOW FLYNN'S
TAXONOMY HAS
HELPED US CLASSIFY
AND STUDY THEM

1. SISD Systems

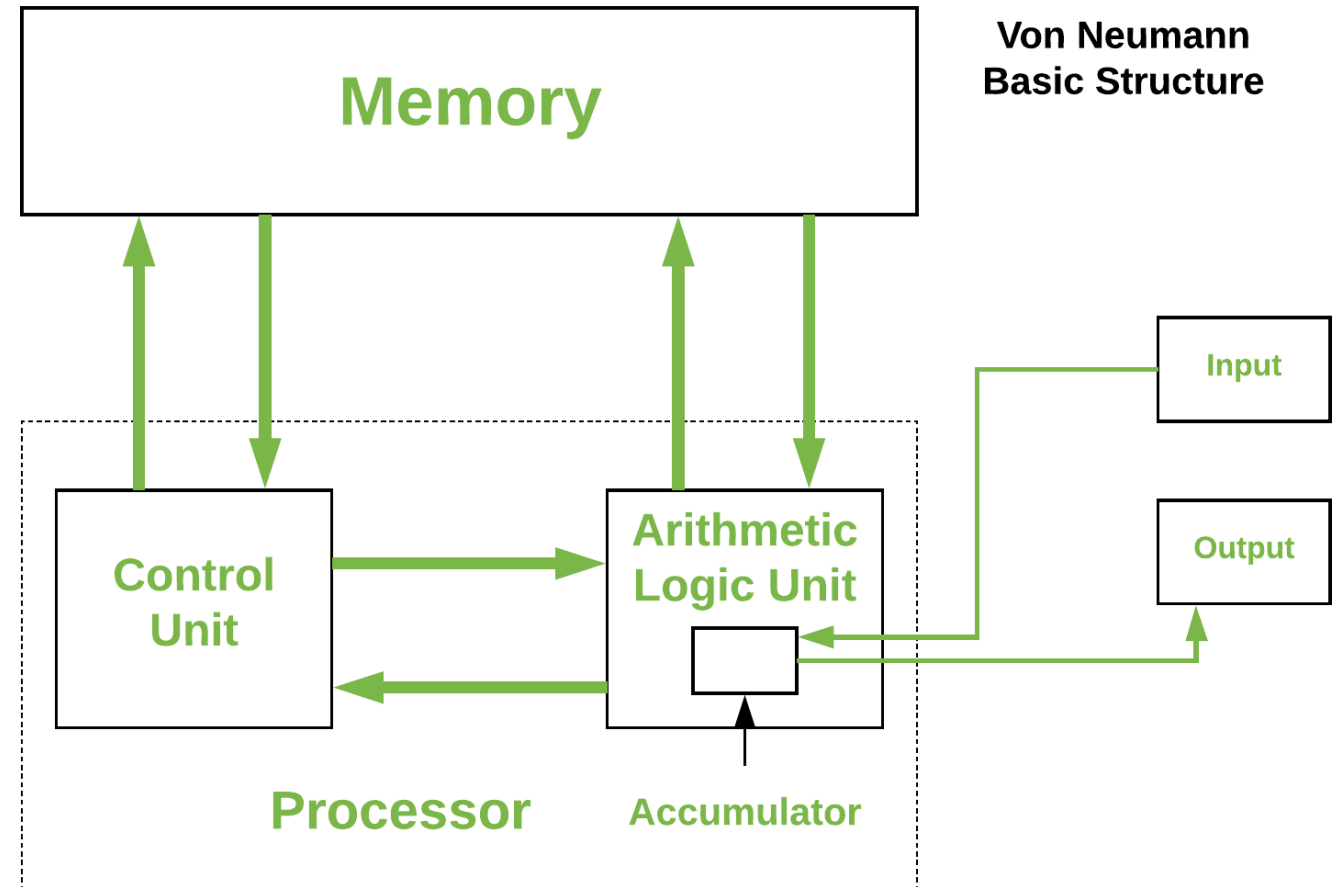
A BRIEF OVERVIEW

Von Neumann Architecture

Proposed in 1945

Architecture where the stored-program concept was first introduced

Refers to a stored program computer where instruction fetch and data operation cannot take place simultaneously as they share a common bus



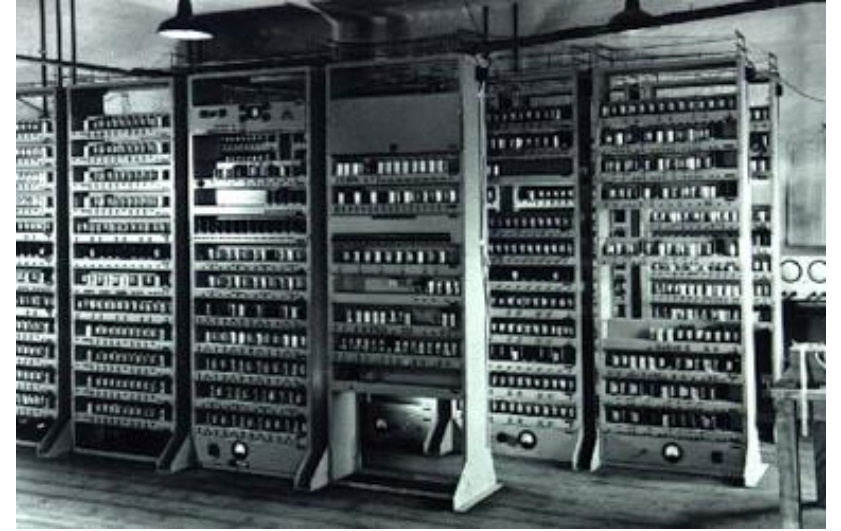
Few systems that
stemmed out
from this
architecture are :

EDVAC, 1946

One of the earliest electronic computers

Designed to be a stored-program computer

Was a binary serial computer, unlike its predecessor ENIAC

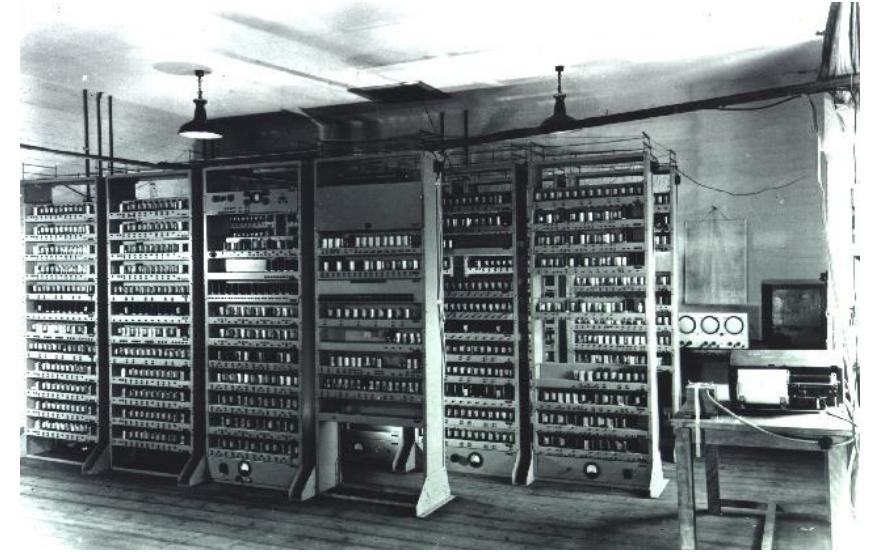


EDSAC, 1949

The second electronic, digital stored-program computer

Designed by Maurice Wilkes and team at University of Cambridge Mathematical Laboratory, England

Winners of 3 Nobel prizes have benefitted from the EDSAC's computing power

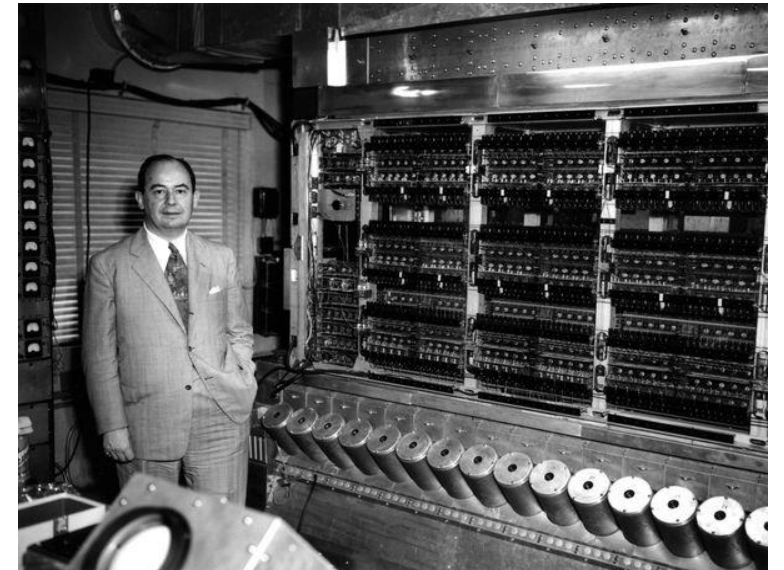
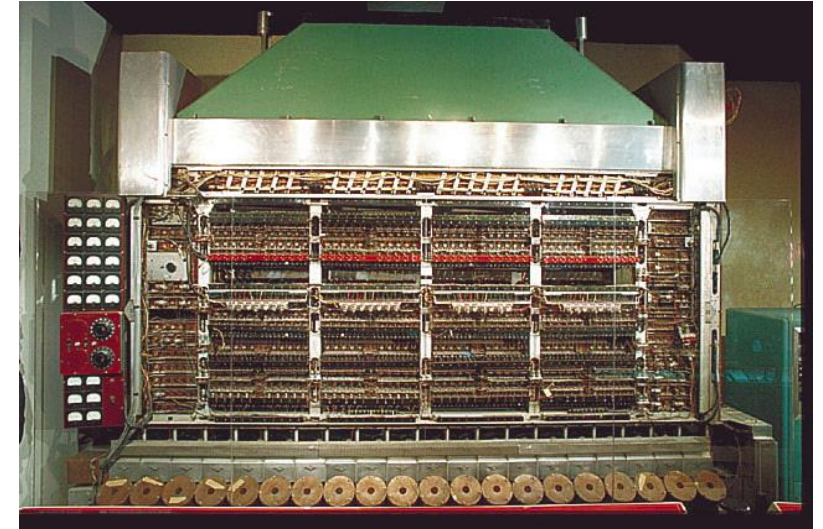


IAS Machine, 1952

First electronic computer to be built in Institute of Advanced Study, Princeton

Generally called Von Neumann Machine, since paper describing its design was edited by Von Neumann

The man on the right, posing with the IAS machine, is Von Neumann himself!



2. SIMD Systems

AND HOW THEY HAVE EVOLVED



ILLIAC IV, 1966

THIS WAS THE MACHINE
THAT FIRST USED THE
CONCEPT OF SIMD

CDC STAR-100, 1974

Was one of the first machines to use vector processors to speed up scientific applications

Was the first supercomputer to use ICs and the first supercomputer that was equipped with a million words of computer memory



CDC STAR-100 1975

TEXAS INSTRUMENTS ASC, 1973

A first-generation vector
processing machine

Was early to implement
but inefficient in vector
processing, much like the
CDC STAR 100, which was
introduced in the same
year

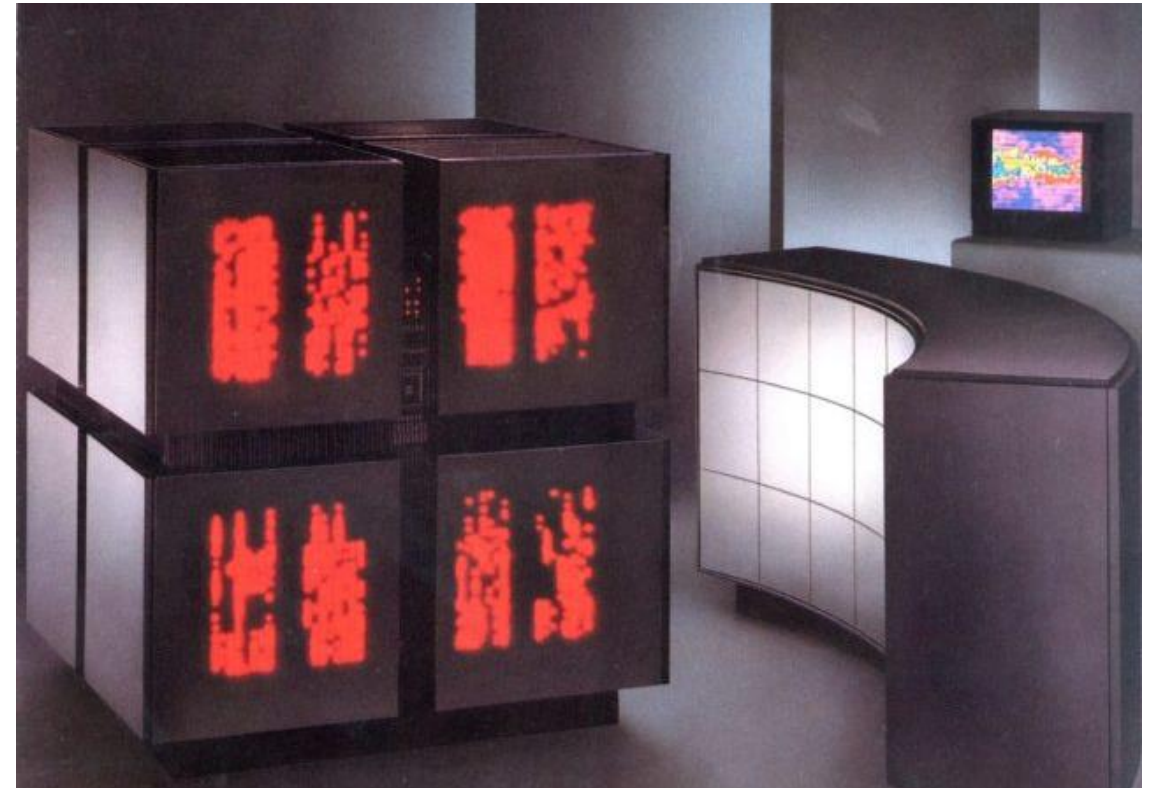


CRAY - 1, 1976

WAS ONE OF THE FIRST
AND MOST SUCCESSFUL
SUPERCOMPUTERS

REMOVED LIMITATIONS
OF CDC STAR-100 AND
TEXAS INSTRUMENTS ASC,
TO MULTIPLY
PERFORMANCE BY
SEVERAL TIMES





THINKING MACHINES CM-1 and CM-2, early 80's

WERE MASSIVELY PARALLEL SUPERCOMPUTERS THAT EMULATED SIMD

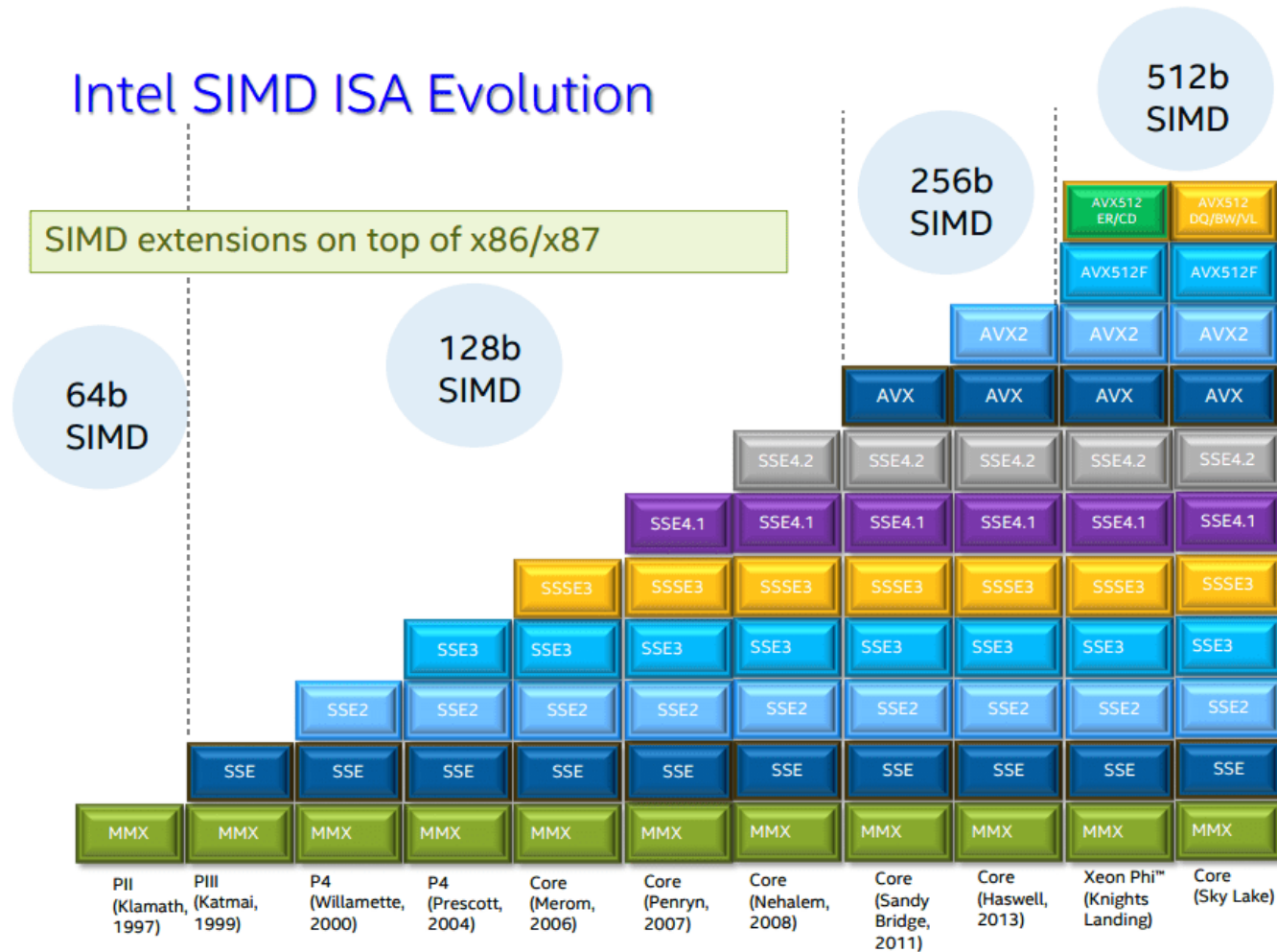
Intel's MMX extensions to the x86 architecture - 1996

In the 90's, desktop computers which used SIMD had become powerful enough to support real-time gaming & audio/video processing

Intel's MMX extensions to the x86 architecture were the first widely deployed SIMD in desktop computers



Intel SIMD ISA Evolution



How Intel has developed SIMD extensions on x86/x87 till today

SIMD in today's world

Modern **desktop computers** are multiprocessor MIMD computers where each processor executes short-vector SIMD instructions

Modern **supercomputers** are a cluster of MIMD computers, each implementing short-vector SIMD instructions

3. MISD systems

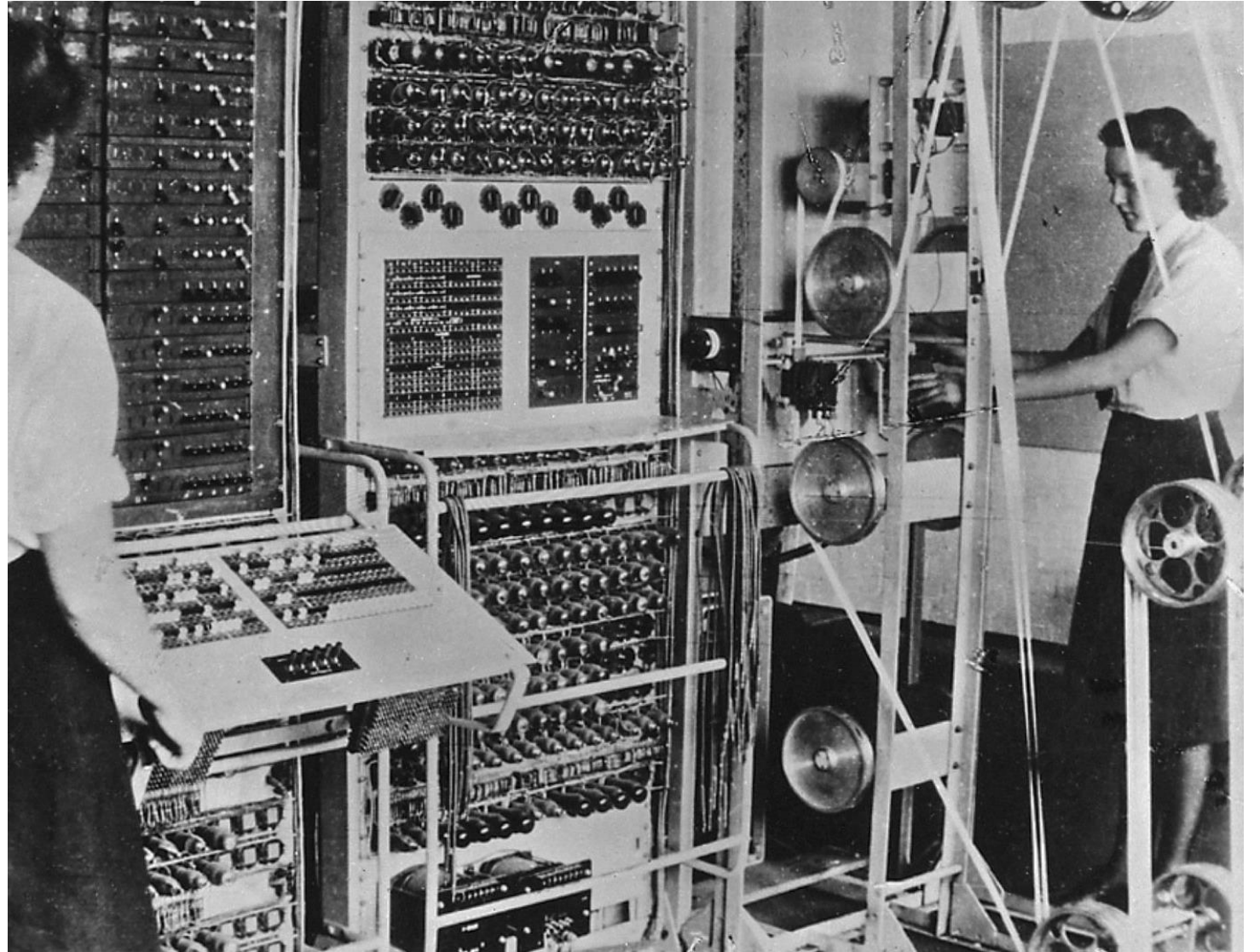
A BRIEF ANALYSIS OF REAL-WORLD EXISTENCE OF MISD

MISD Systems

Pipeline Architectures and Fault tolerance(for error detection) using task replication are examples of MISD

Systolic Arrays, published in 1979 by Kung and Leicerson, emulates MISD

However, the first machine to use a concept similar to systolic arrays dates back to the Colossus Mark II, in 1944!



IBM System/4 Pi, 1967

Was used in the family of Space Shuttle Flight computers

Is one of the very few machines that actually implemented MISD



4. MIMD Systems

A BRIEF LOOK AT THE MOST PROMISING CATEGORY OF
COMPUTERS

DEC-PDP-11, early 70's

One of the most successful minicomputers of all time

The experimental C.mmp and CM* projects (1971) at Carnegie Mellon used DEC-PDP-11s

Samuel C.C. Ting used a DEC-PDP-11 for his experiments in discovering J/ψ meson, which fetched him his Nobel prize in '76



Goodyear MPP, 1983

Massively parallel processing computer built by Goodyear Aerospace, for the NASA Goddard Space Flight Center

Designed to deliver enormous computational power at a lower cost than other existing supercomputer architectures at that time



FROSTBURG - THINKING MACHINES CM-5, 1991

Was the first massively parallel supercomputer bought by the NSA, to perform higher-level mathematical calculations

The OS CMost was based on UNIX, but optimized for parallel processing



Intel Xeon Phi, 2010

Series of x86 manycore processors, designed for supercomputers and high-end workstations

Based on Intel's earlier GPU design and thus shares areas of applicability with GPUs



OWING TO THEIR
DIVERSITY AND
HIGH
PERFORMANCE,

Most parallel
computers today
are **MIMD**
computers

What about other taxonomies?

FLYNN WAS NOT THE ONLY ONE TO PROPOSE A TAXONOMY FOR COMPUTERS. HERE IS A LOOK AT A FEW OTHER TAXONOMIES PROPOSED...

Other Taxonomies

Shore – Based on how computer is organized from its constituent parts. 6 different types distinguished by a numerical designator

Bell – A taxonomy exclusive for MIMD machines

Feng – Based on the number of bits processed simultaneously

Hockney and Jesshope – Most ambitious taxonomy where processors were classified similar to chemical notation for organic compounds. Was successfully exhaustive of all systems but too complex

EVEN WITH SO
MANY OTHER
OPTIONS,

Why is Flynn's
Taxonomy most
successful?

Perks of Flynn's Taxonomy

It is an exhaustive taxonomy

When computing systems could not be accommodated within, it could be expanded incrementally without modifying the core basis(SPMD/MPMD)

It is a fairly simple taxonomy, easy to understand

It is very well established and most widely used

SUMMARY

SO, TO WRAP IT UP...

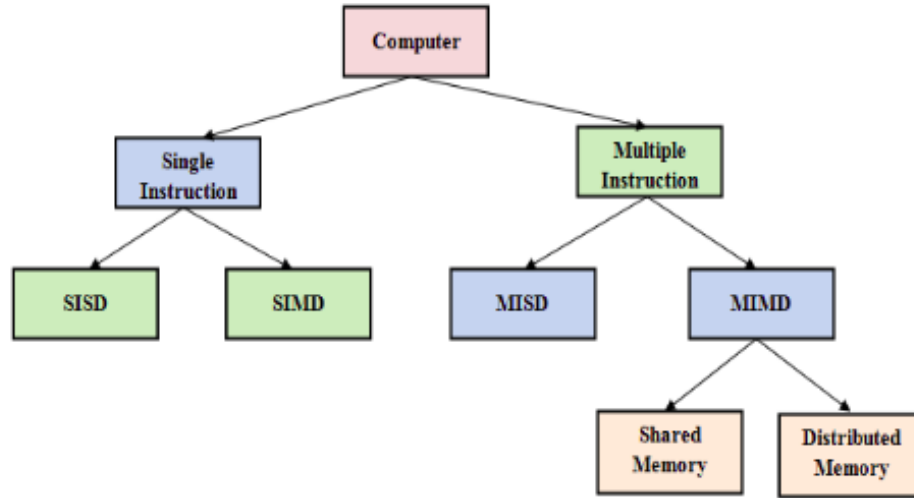
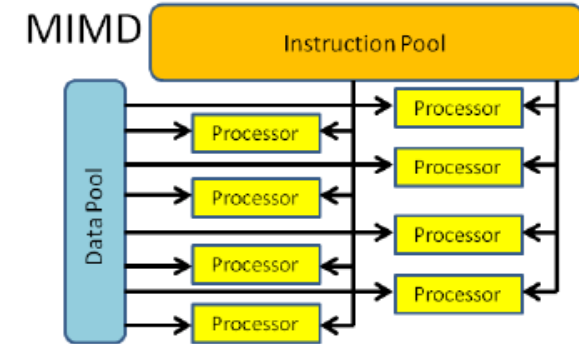
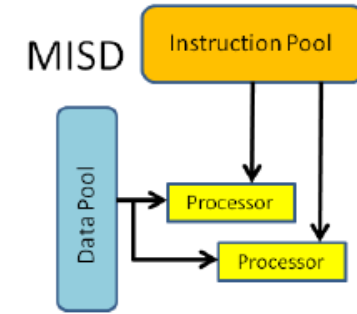
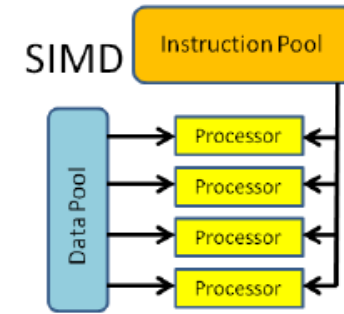
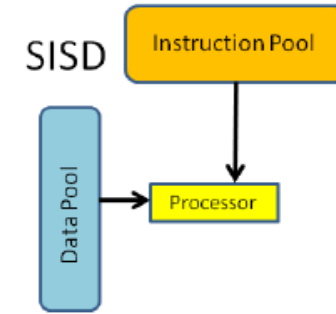
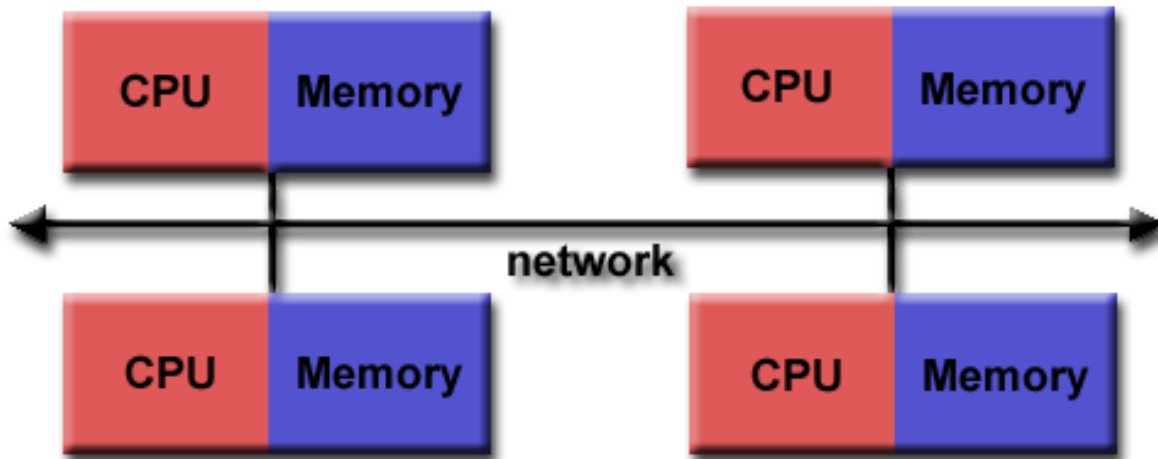


Fig 1-: Flynn's Taxonomy

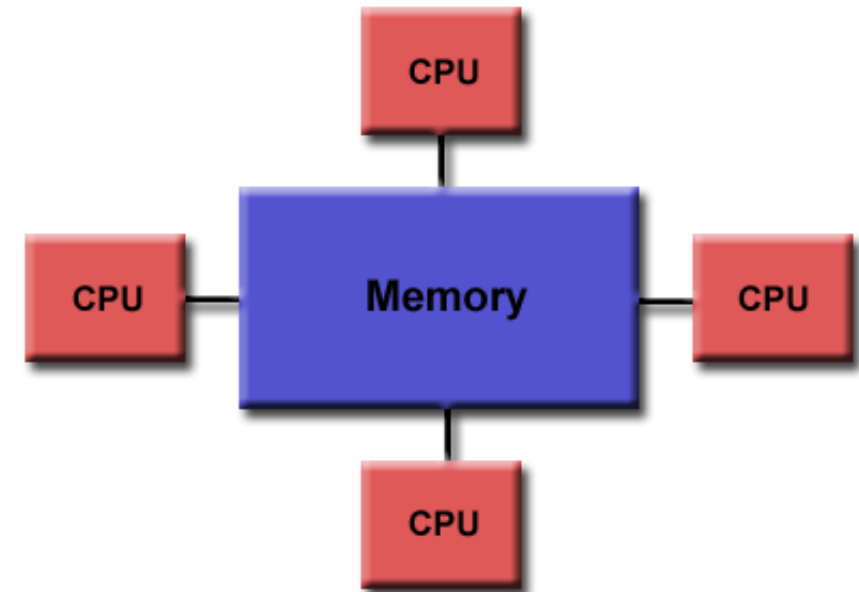


The essence of Flynn's Taxonomy

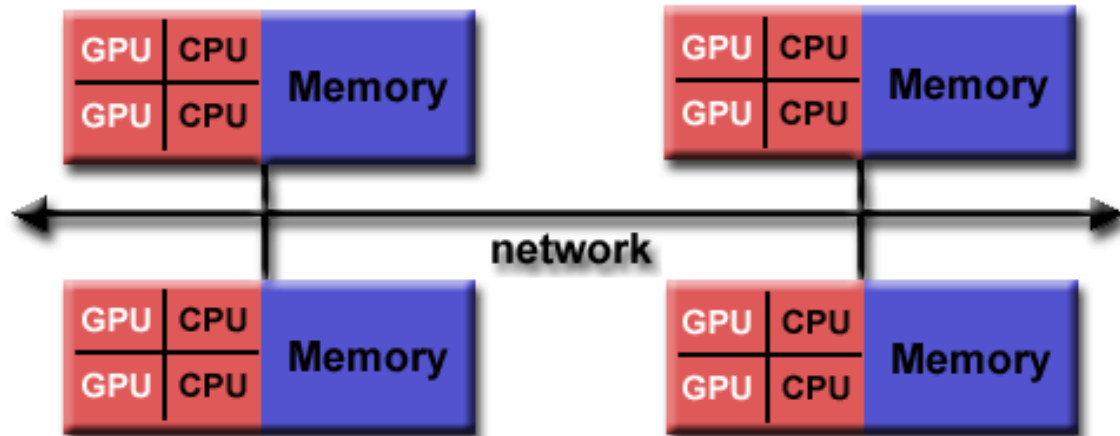
DISTRIBUTED-MEMORY



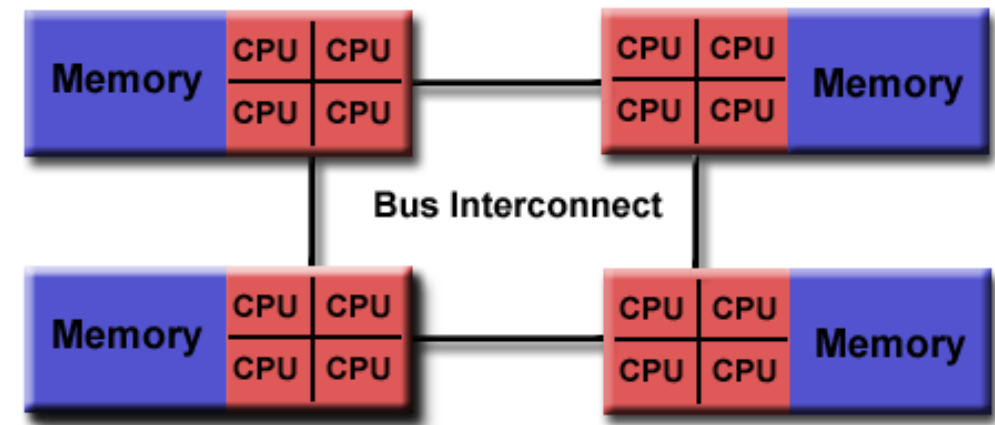
SHARED-MEMORY UMA



HYBRID DISTRIBUTED-SHARED MEMORY



SHARED-MEMORY NUMA





THANK YOU!

- PRATYUSH V MOORTHY

COE18B042