

# Parallel and Distributed Systems / 2



Pavel Krömer,  
Dept. of Computer Science,  
VSB – Technical University of  
Ostrava

# Agenda

- Major topics
  - Types, basic concepts, classification
- Literature
  - Bhujade Moreshwar R., M.R. Bhujade; Parallel Computing, New Age International, 2009
  - Mattson, Timothy G., Rasmussen, Craig E., Sottile, Matthew J; Introduction to Concurrency in Programming Languages, Chapman & Hall/CRC Press, 2009
  - David Padua; Encyclopedia of Parallel Computing, Springer US, 2011

# Parallel platforms and programs

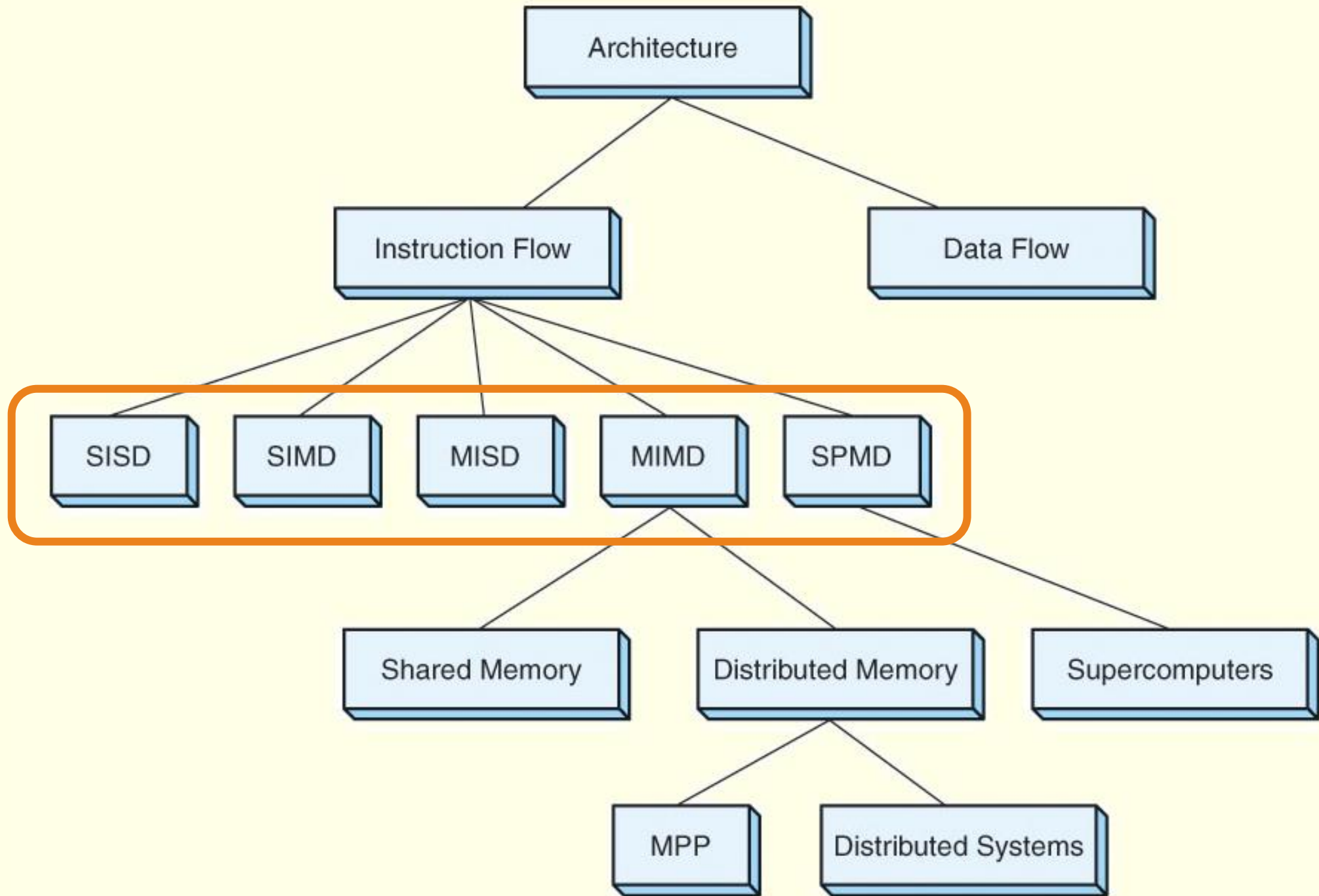
# Motivation

*The main purpose of a classification/taxonomy is to provide a common language in which **parallel and distributed architectures** can be discussed.*

(cw. {design | integration | architectural} patterns)

## Caveats

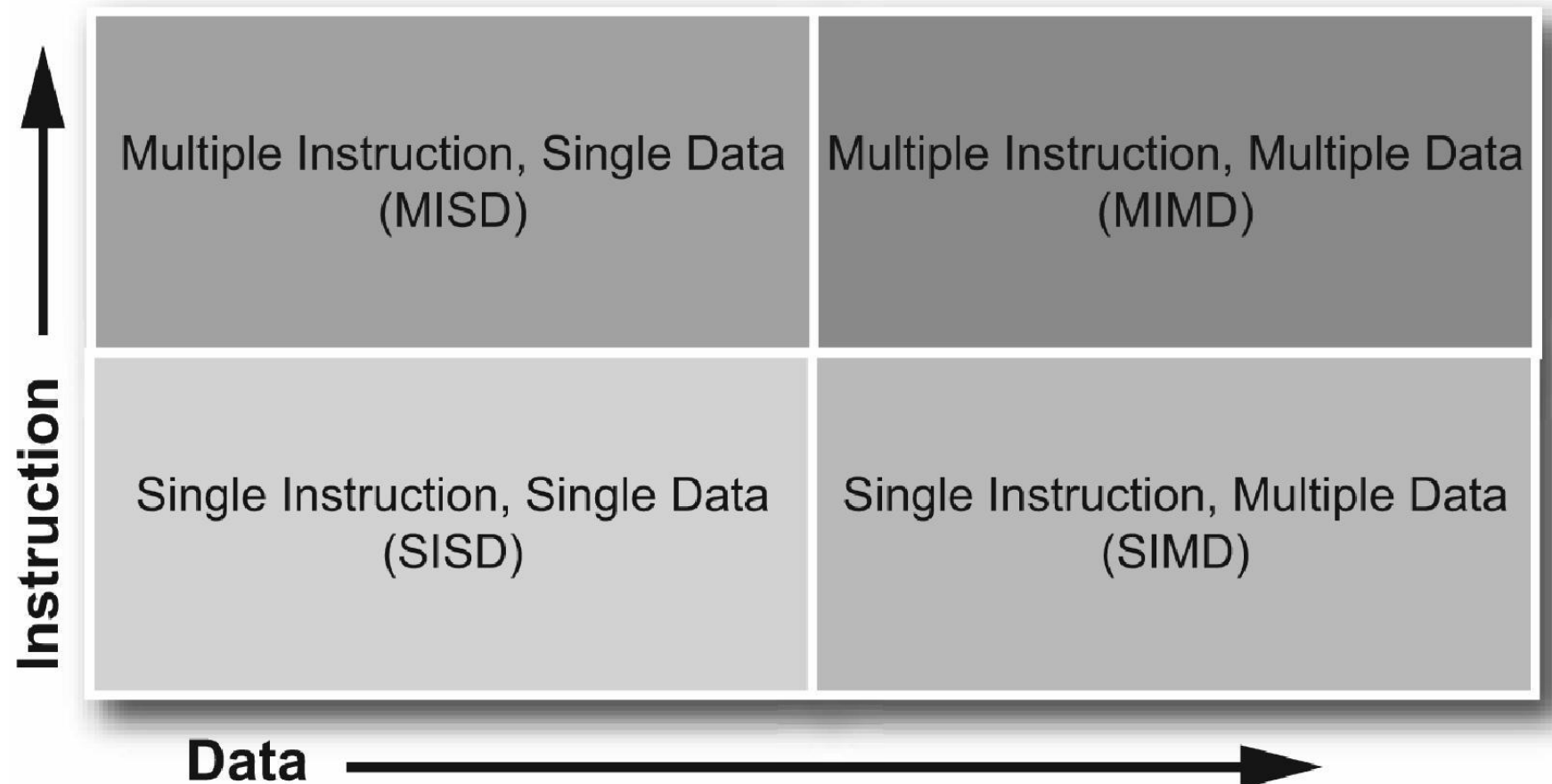
- not a single classification agreed upon
- age quickly



# Flynn's 1967 taxonomy

- Flynn, Michael. (1967). Very High-Speed Computing Systems. Proceedings of the IEEE. 54. 1901 - 1909. 10.1109/PROC.1966.5273.

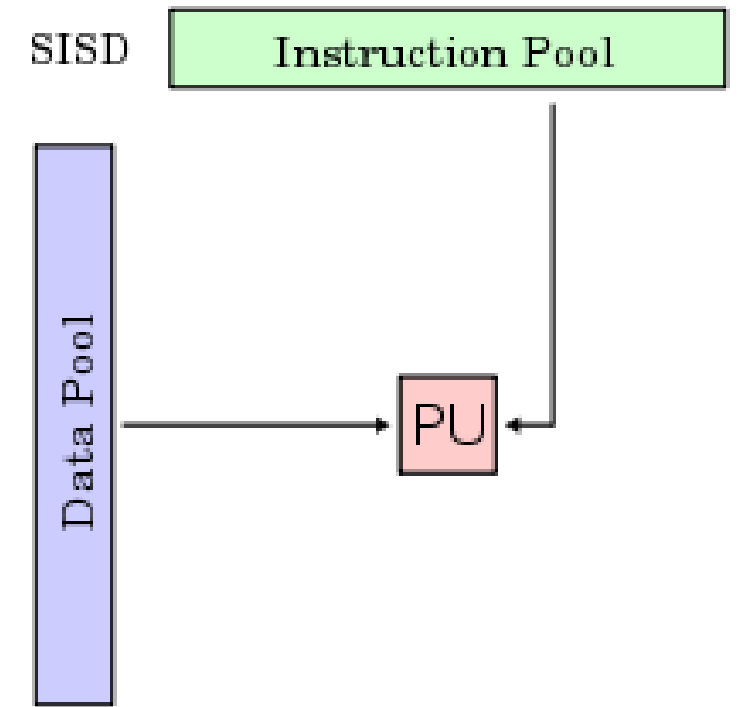
- **Dimensions:** instruction and data streams



# Single-instruction single-data

# SISD

- Serial (non-parallel) computer
- ‘Von Neumann architecture\*’, deterministic execution
- Only one instruction stream executed by the CPU
- Only one data stream used as an input during any clock cycle



UNIVAC1



IBM 360



CRAY1



CDC 7600



PDP1



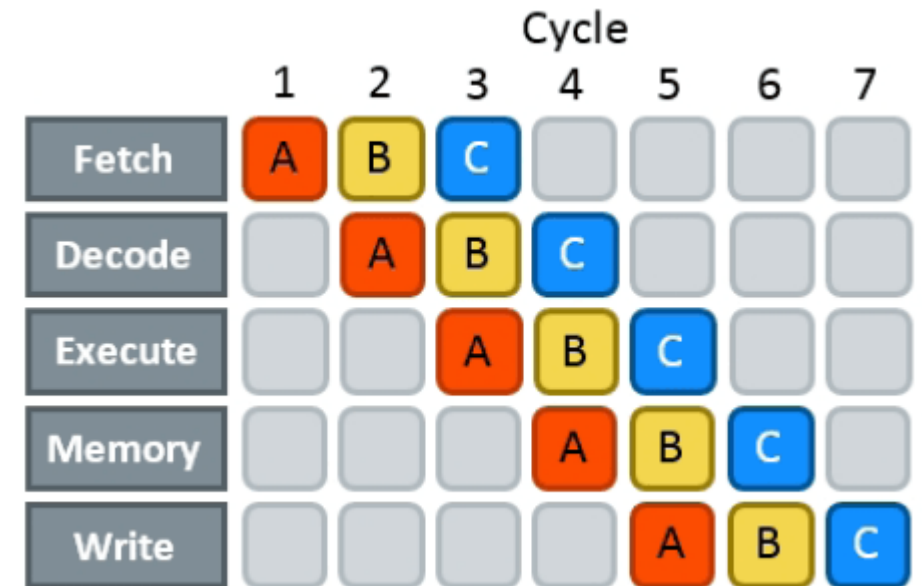
Dell Laptop

# Single-instruction single-data

# SISD

Parallel extensions of SISD architectures

- Instruction pipelining



UNIVAC1



IBM 360



CRAY1



CDC 7600



PDP1



Dell Laptop



# Single-instruction single-data



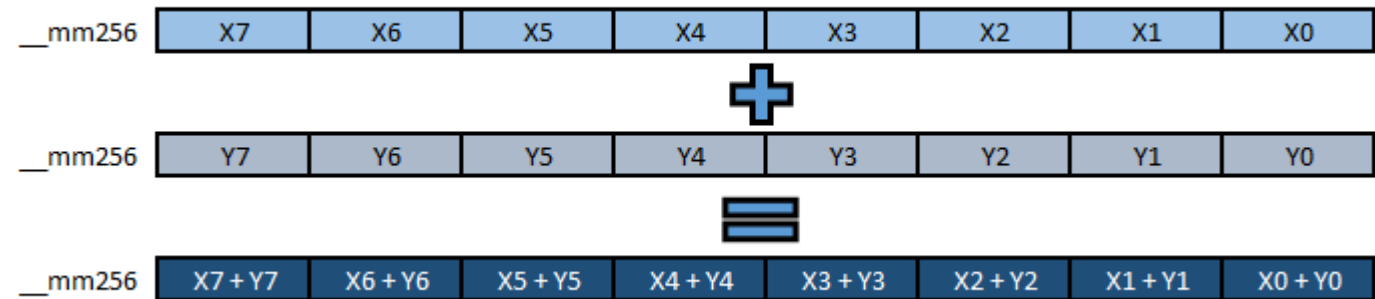
## Parallel extensions of SISD architectures

- Vector registers, extended instruction sets
  - Same instructions applied to a vector of data
  - MMX (1997, 80-bit), SSE (128-bit), AVX (2011, 256-bit), AVX-512 (2016)

### AVX Data Types (16 YMM Registers)

|                       |   |       |        |       |        |       |        |       |                  |
|-----------------------|---|-------|--------|-------|--------|-------|--------|-------|------------------|
| <code>__mm256</code>  | Float   | Float | Float  | Float | Float  | Float | Float  | Float | 8x 32-bit float  |
| <code>__mm256d</code> | Double  |       | Double |       | Double |       | Double |       | 4x 64-bit double |
| <code>__mm256i</code> | <i>256-bit Integer registers. It behaves similarly to <code>__m128i</code>. Out of scope in AVX, useful on AVX2</i> |       |        |       |        |       |        |       |                  |

### AVX Operation



UNIVAC1



IBM 360



CRAY1



CDC 7600



PDP1



Dell Laptop

# Single-instruction single-data

# SISD

## Parallel extensions of SISD architectures

- **Superscalar** execution
  - multiple instruction pipelines
  - in-order execution
    - Very-long instruction word (VLIW);
    - One instruction encodes more operations (for each pipeline one)
    - Compiler finds instructions that can be **speculatively** executed in parallel
  - out-of-order execution
    - Hardware finds independent instructions



UNIVAC1



IBM 360



CRAY1



CDC 7600



PDP1



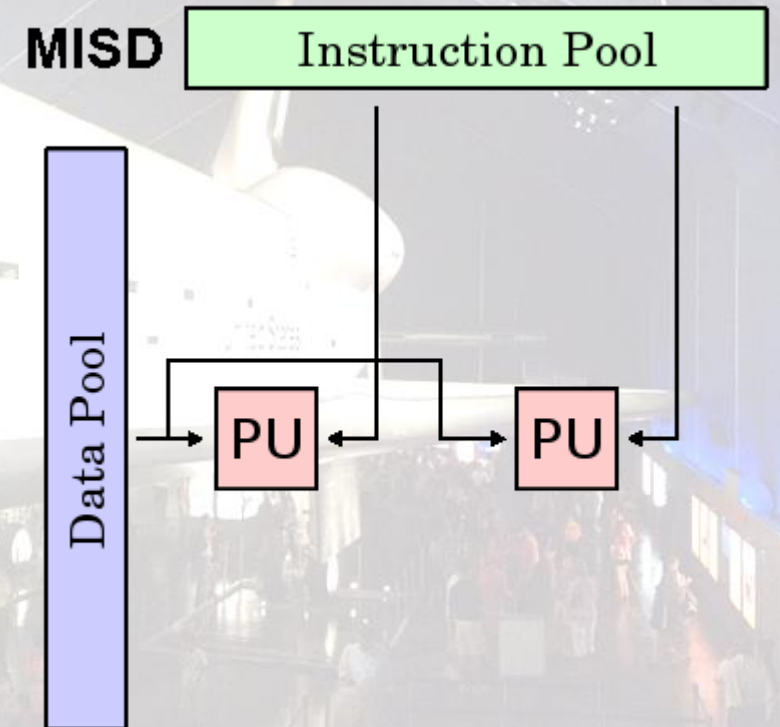
Dell Laptop

# Multiple-instruction single-data

# MISD

Rather rare parallel architecture

- Each processing unit operates on the data independently via **separate instruction streams**
- A **single data stream** is fed into multiple processing units
- Example usage
  - Redundant execution for fault tolerance (Space Shuttle flight control, Falcon 9/heavy avionics - “flight strings”)
  - Multiple frequency filters operating on a single signal stream
  - Multiple cryptography algorithms attempting to crack a single coded message

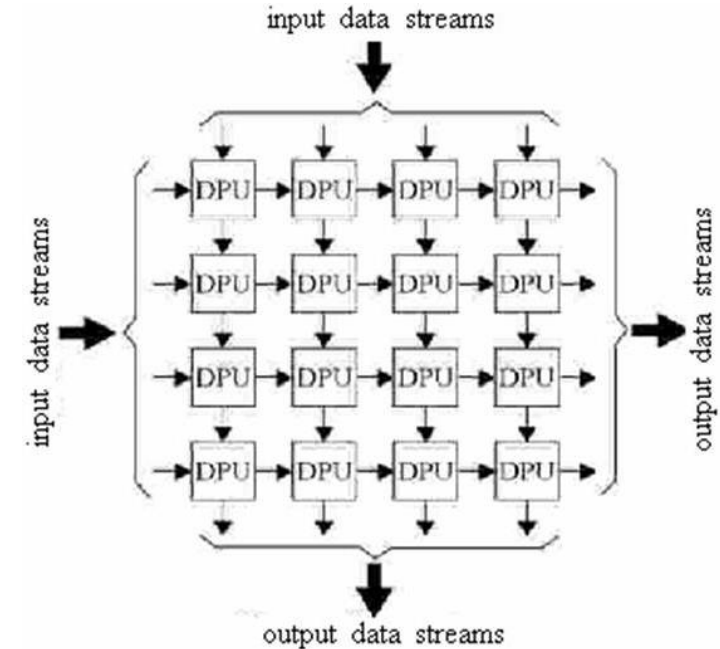
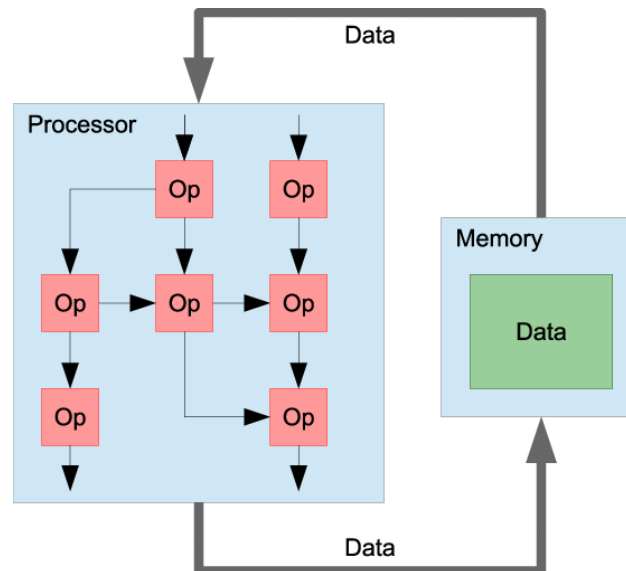


# Multiple-instruction single-data

# MISD

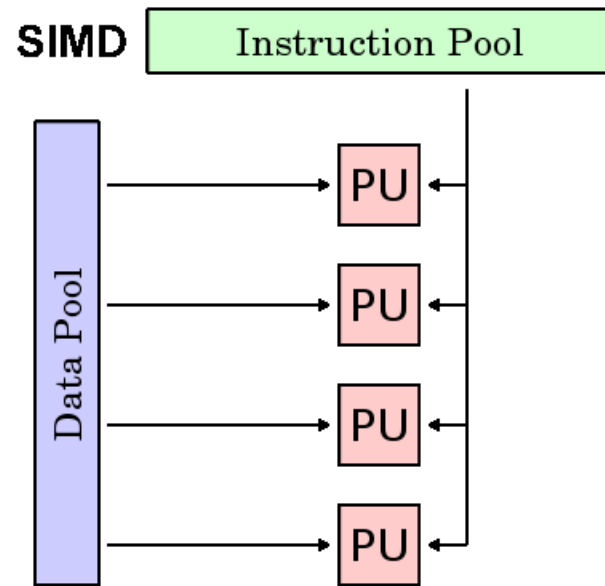
## Modern MISD-like architectures

- Systolic arrays
  - Computational networks with distributed data storage and distributed processing units
  - Matrix operations, convolution
  - Use: FPGAs, ASICs
- Data-flow computers
  - operations executed when their operands are available
  - Maxeler Technologies



# Single-instruction multiple-data

# SIMD



- AKA: data-parallelism, vector operations
- All processing units execute the **same instruction** at any given clock cycle
- Each processing unit can operate on a **different data element**
- Good for specialized problems with a high degree of regularity,
  - Matrix ops, graphics/image processing.
- Synchronous (lockstep) and deterministic execution



ILLIAC IV



MasPar



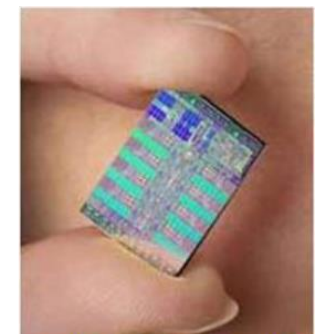
Cray X-MP



Cray Y-MP



Thinking Machines CM-2



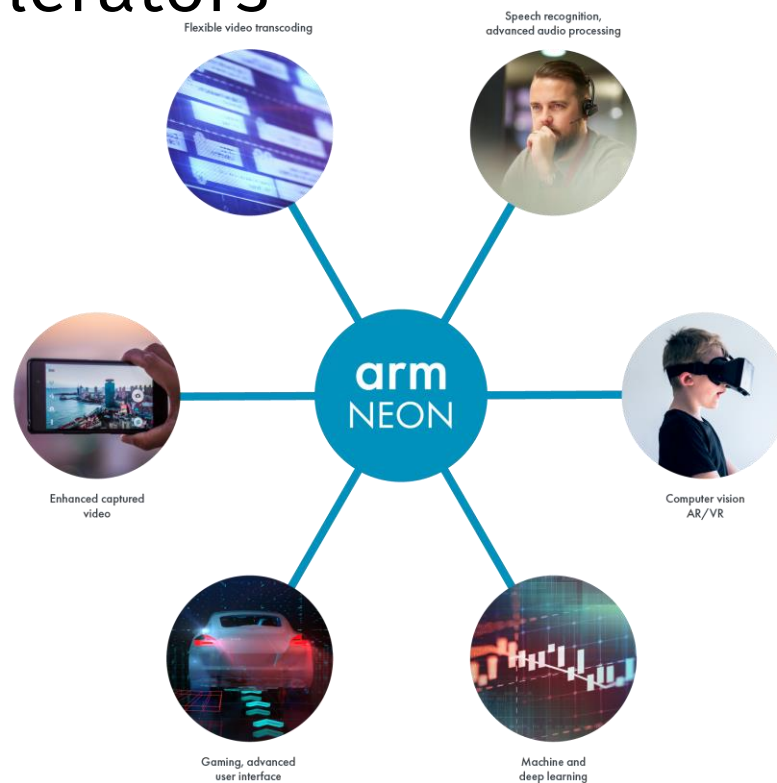
Cell Processor (GPU)









# Single-instruction multiple-data

# SIMD

## Modern examples

- GPGPUs,  
FP accelerators
- CPUs

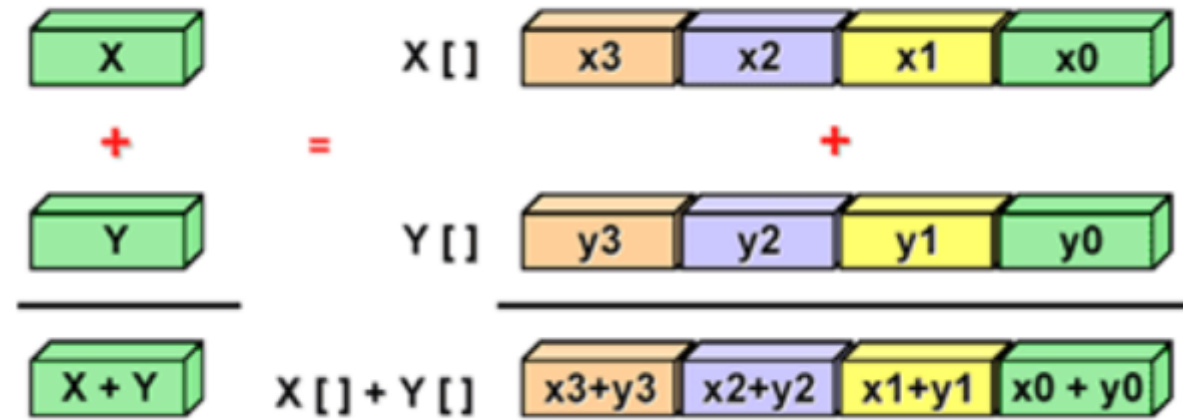


| Software       | Jetpack / Deepstream  |   |   |  |   |
|----------------|---|---|---|--|---|
| Modules        | <br>Jetson TX1               | <br>Jetson TX2<br>8 GB       | <br>Jetson TX2<br>Industrial | <br>Jetson TX2<br>4 GB              | <br>Jetson AGX<br>Xavier |
| Developer Kits | <br>Jetson TX1 Developer Kit | <br>Jetson TX2 Developer Kit |   | <br>Jetson AGX Xavier Developer Kit |   |

# Single-instruction multiple-data

# SIMD

A SIMD computation



Caveat

- branching

# Multiple-instruction multiple-data

**MIMD**

General parallel (task parallel) computation without the restrictions of SIMD



# Beyond Flynn's taxonomy

## Extensions and alternatives

- **SPMD** - Single **process** multiple data
- **MPMD** - Multiple **process** multiple data
- Multiple SIMD - hybrid architectures of multiple SIMD systems
- Superscalar SIMD

## Feng's classification (1972)

- serial vs parallel processing

## Handler's classification (1977)

- based on the degree of parallelism/pipelining ability of computer subsystems

# Beyond Flynn's taxonomy

## Feng's classification

### Vocabulary

Word Serial and Bit Serial (WSBS)

Word Parallel and Bit Serial (WPBS)

Word Serial and Bit Parallel (WSBP)

Word Parallel and Bit Parallel (WPBP)

### System classification

- **WSBS** has been called bit parallel processing because one bit is processed at a time.
- **WPBS** has been called bit slice processing because m-bit slice is processed at a time.
- **WSBP** is found in most existing computers and has been called as Word Slice processing because one word of n bit is processed at a time.
- **WPBP** is known as fully parallel processing in which an array of n x m bits is processed at one time.

# Beyond Flynn's taxonomy

## Handler's classification

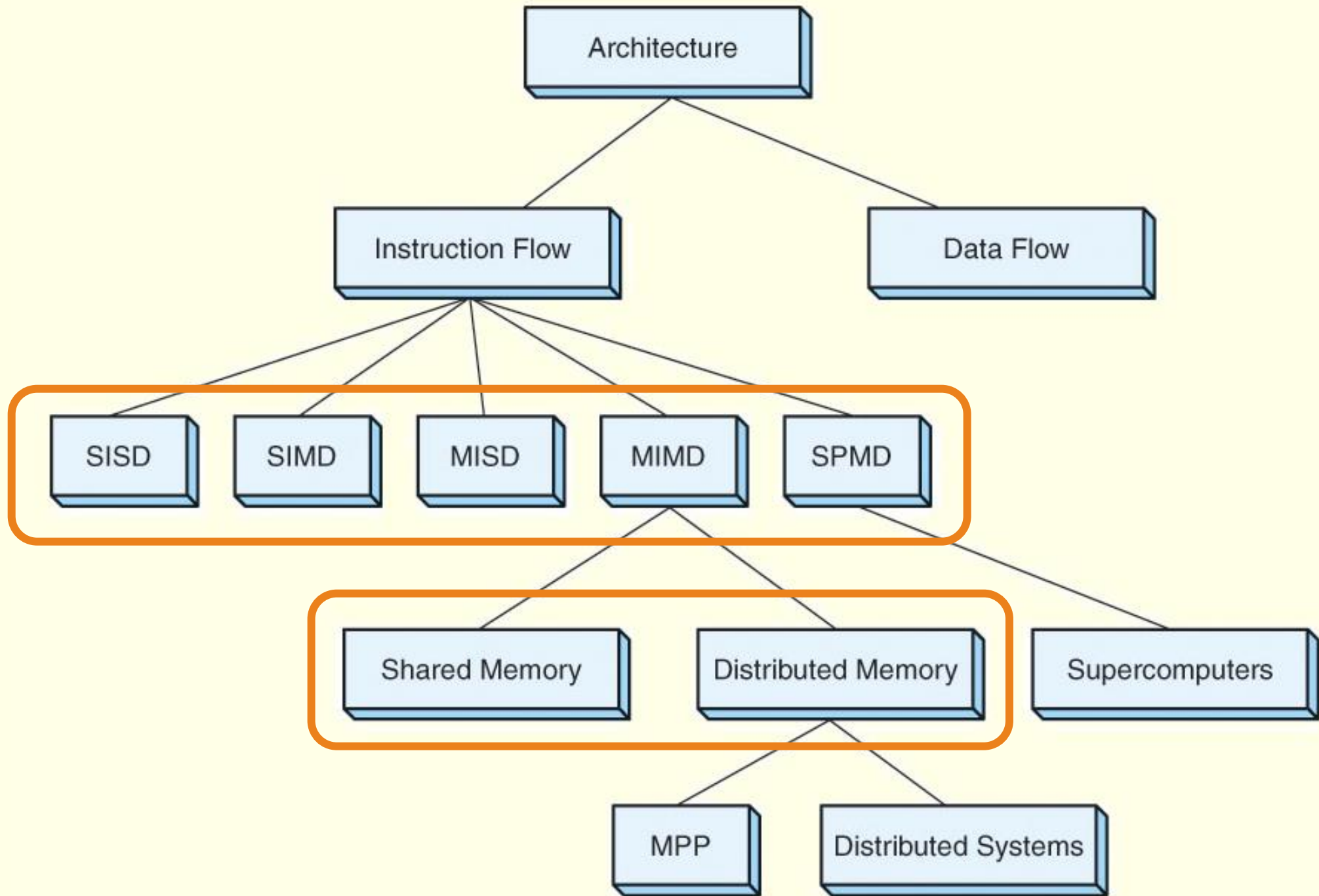
- Evaluates 3 levels of systems architecture
  - processor control unit (PCU [= CPU]); Arithmetic logic unit (ALU); Bit-level circuit (BLC)

$$\text{Computer} = (p * p', a * a', b * b')$$

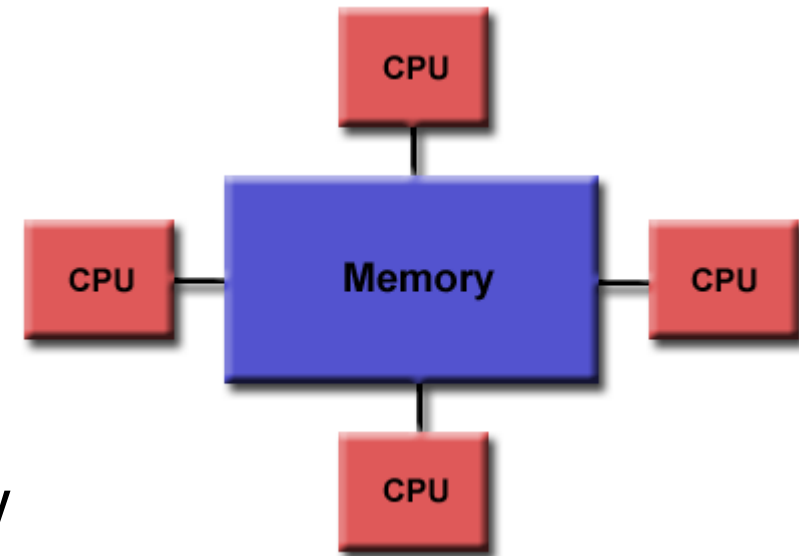
- $p$  = number of PCUs
- $p'$  = number of PCUs that can be pipelined
- $a$  = number of ALUs controlled by each PCU
- $a'$  = number of ALUs that can be pipelined
- $b$  = number of bits in ALU or processing element (PE) word
- $b'$  = number of pipeline segments on all ALUs or in a single PE
- set of well-defined operators ( $*$ ,  $+$ ,  $v$ ,  $\sim$ )

Example: Cray-1 = (1, 12 \* 8, 64 \* (1 ~ 14))





# Shared memory systems



Processors have access to the complete memory (as a global [shared] address space)

Processors can operate independently but share memory resources

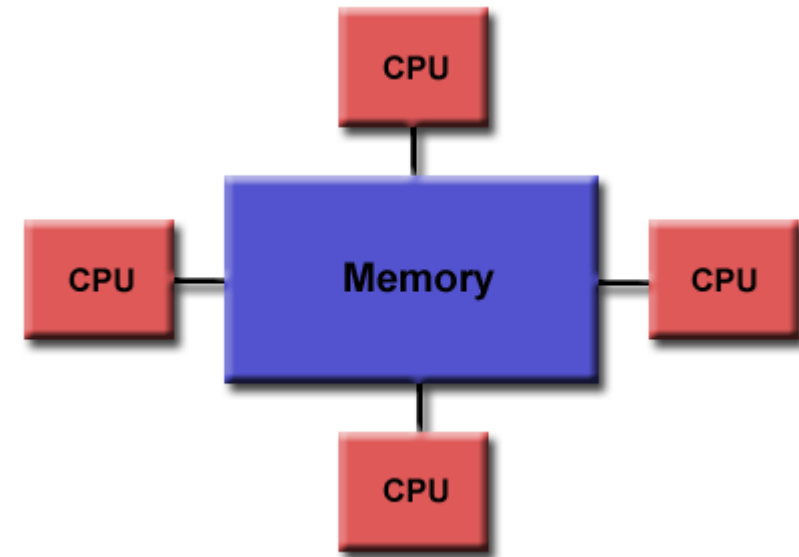
Changes in memory caused by one processor are visible to all other

Also **tightly coupled systems**

# Shared memory systems

## Uniform memory access (UMA)

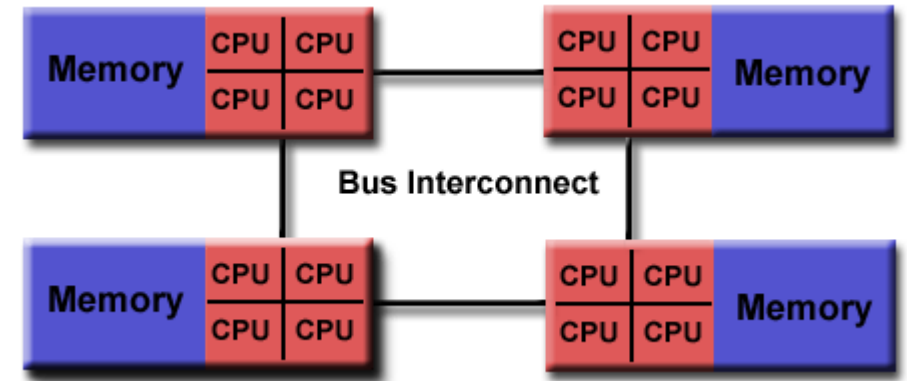
- AKA **Symmetric Multiprocessors (SMPs)**
- Systems of identical processors with equal access and access times to memory
- Sometimes called **Cache Coherent UMA (CC-UMA)**
  - if one processor updates a location in shared memory, all the other processors know about the update
  - cache memories that provide access to these variables are kept consistent
  - accomplished at the hardware level (snoopy/sniffy bus protocol)



# Shared memory systems

## Non-uniform memory access (NUMA)

- 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
- Cache Coherent NUMA (CC-NUMA) if cache coherency is achieved





# Shared memory systems



## Pros and cons

### • Advantages

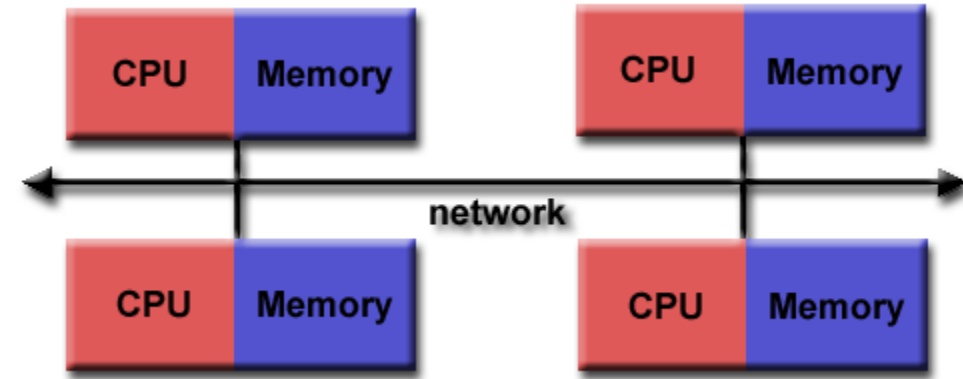
- Global address space provides a **user-friendly** programming access to memory
- **Data sharing** between tasks is both **fast and uniform** due to the proximity of memory to CPUs

### • Disadvantages

- **Lack of scalability between memory and CPUs.** Adding more CPUs can geometrically increase traffic on the shared memory-CPU path, and for cache coherent systems, geometrically increase traffic associated with cache/memory management
- **Programmer responsibility** for synchronization constructs that ensure "correct" access of global memory.

| Feature                         | SPARC M8 Processor  |
|---------------------------------|---|
| CPU frequency                   | 5.0 GHz   |
| Out-of-order execution          | Yes   |
| Instruction issue width         | 4   |
| Data/instruction prefetch       | Yes   |
| SPARC core                      | Fifth generation  |
| Cores per processor             | 32  |
| Threads per core                | 8   |
| Threads per processor           | 256   |
| Sockets in systems              | Up to 8   |
| Memory per processor            | Up to 16 DDR4 DIMMs   |
| Caches                          | 32 KB L1 four-way instruction cache<br>16 KB L1 four-way data cache<br>Shared 256 KB L2 four-way instruction cache (per quad cores)<br>128 KB L2 eight-way data cache (per core)<br>Shared 64 MB (L3) cache |
| Large page support <sup>1</sup> | 16 GB   |
| Power management granularity    | Half of the chip  |
| Technology                      | 20 nm technology  |

# Distributed memory systems



Use a communication network to connect inter-processor memory.

Processors have own local memory and memory addresses in one processor do not map to another processor. No global address space exists.

Processors operate independently, local changes do not affect other processors (and their memory). Cache coherency out of question.

IPC (data sharing + synchronization) explicitly defined by program/programmer.

Also **loosely coupled systems**

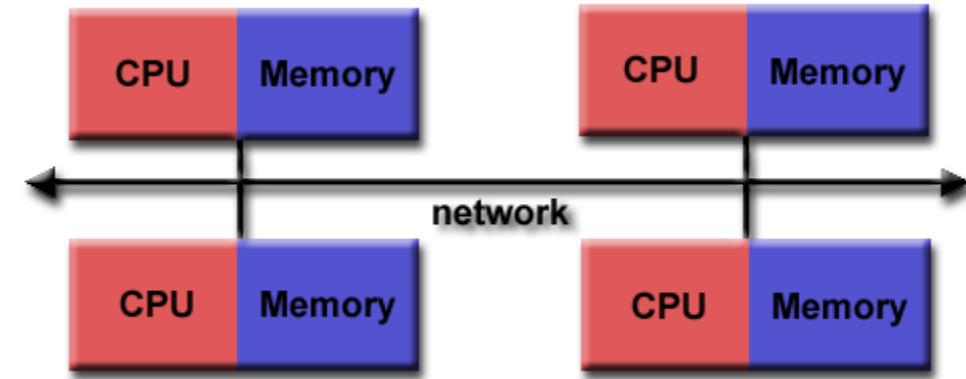
# Distributed memory systems

## Challenging IPC

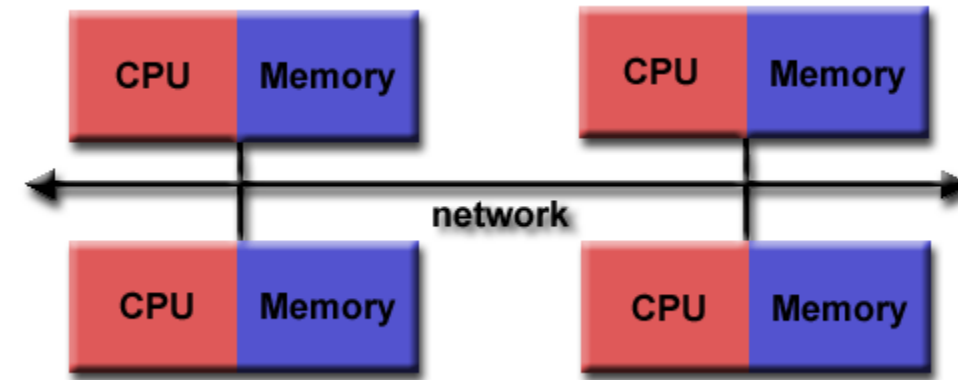
- Explicit IPC via message passing
- Communication usually a significant bottleneck
- Communication-driven algorithm design and considerations

## Advantages

- Memory scalable with the number of processors
- Each processor can rapidly access its own memory without interference and without any special overhead wrt. cache coherency
- Cost effective, can use commodity, off-the-shelf processors and networking



# Distributed memory systems

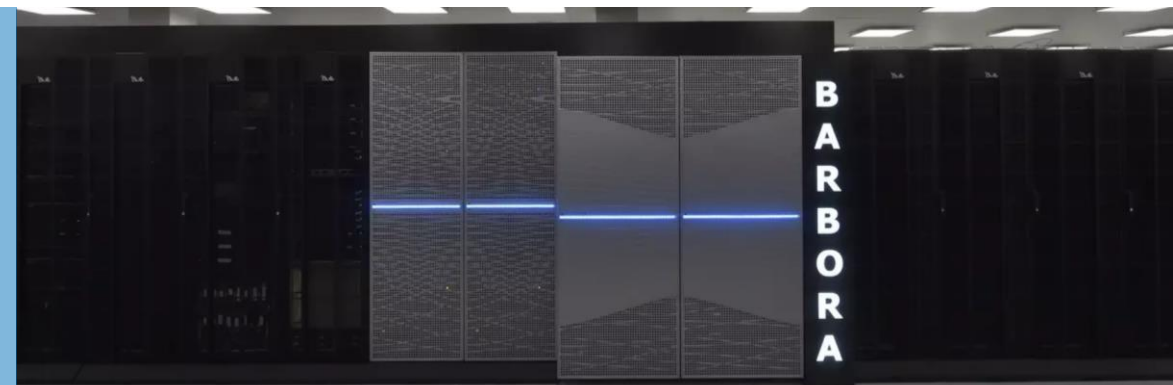


## Disadvantages

- Program/programmer responsible for all details associated with data communication between processors
- Difficult mapping of existing data structures to distributed memory for an **efficient execution**
- Non-uniform memory access times - data residing on a remote node takes longer to access than local data

## Barbora

- 2. 10. 2019, VSB-TUO / IT4Innovations
- 192x compute node w. 18-core CPU, 192 GiB RAM
- 8x GPU node w. 12-core CPU, 192 GiB RAM, 4x NVIDIA V100
- 1x fat node w. 8x16-core CPU, 6 TiB RAM

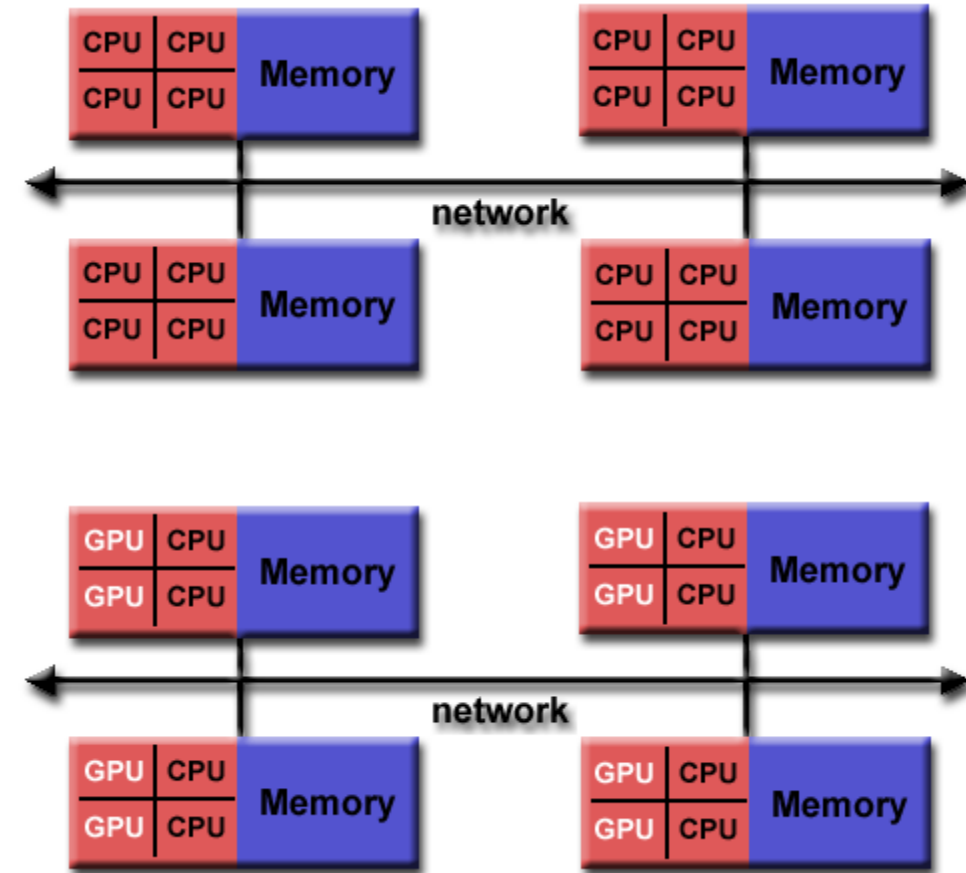


# Hybrid systems

Real-world systems combine SMPs and distributed memory

Combine the advantages/disadvantages of both

- Increased scalability traded for increased program complexity





# System description can be based on

- Parallelism type
  - Data vs. task parallelism
- Granularity
  - The ratio of communication and computation
  - How many instructions in a process\*
  - Fine (<20) vs. medium (<500) vs. coarse grained programs
- Degree of parallelism
  - high vs. low



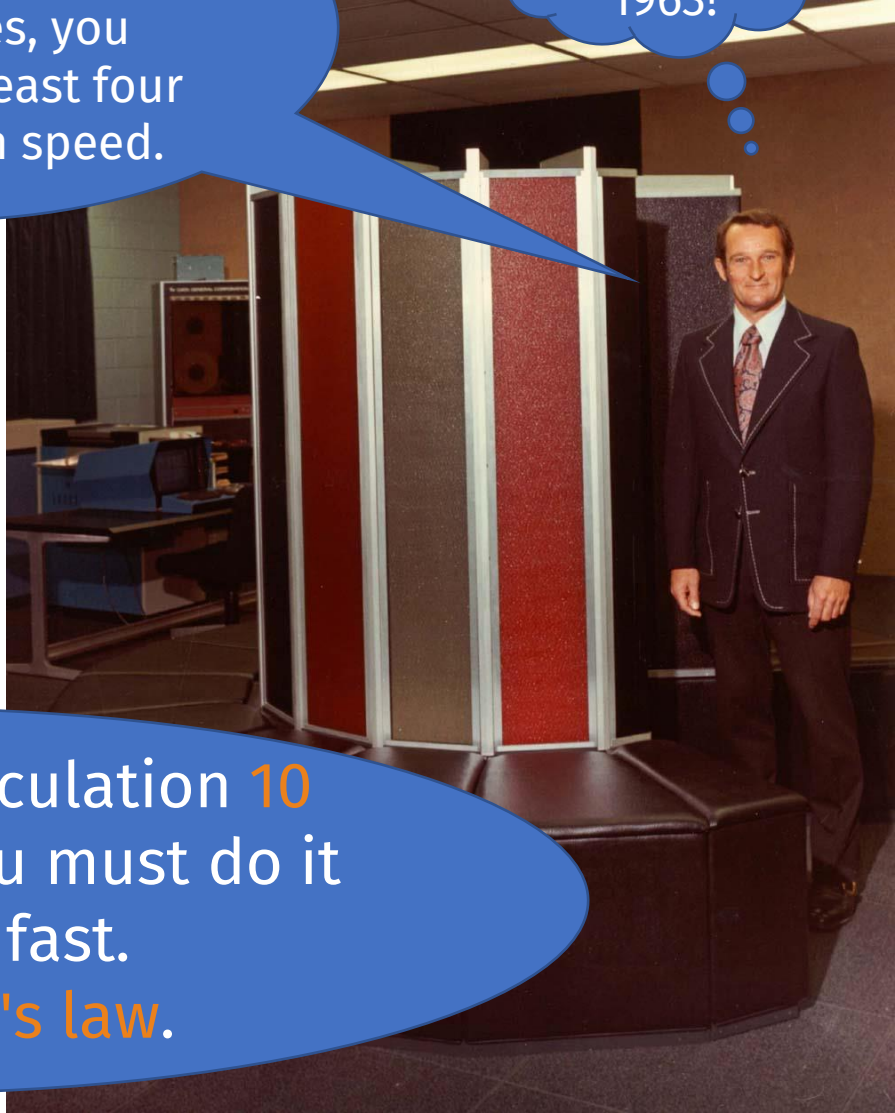


# Herb Grosch's law and Seymour Cray's observation



Computers should obey a **square law** — when the price doubles, you should get at least four times as much speed.

I said that in 1963!



1965: ... to do a calculation **10** times as cheaply you must do it **100** times as fast. I call it **Grosch's law**.

# Marvin Minsky's conjecture

Oliver Selfridge

Ray Solomonoff

Marvin Minsky

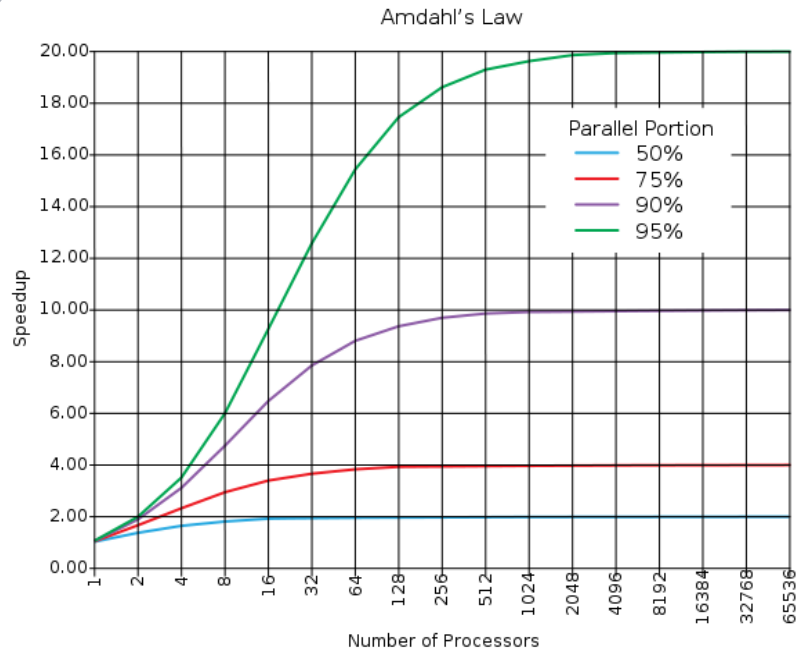
John McCarthy

Claude Shannon

IN THIS BUILDING DURING THE SUMMER OF 1956  
JOHN MCCARTHY (DARTMOUTH COLLEGE), MARVIN L. MINSKY (MIT)  
NATHANIEL ROCHESTER (IBM), AND CLAUDE SHANNON (BELL LABORATORIES)  
CONDUCTED  
THE DARTMOUTH SUMMER RESEARCH PROJECT  
ON ARTIFICIAL INTELLIGENCE  
FIRST USE OF THE TERM "ARTIFICIAL INTELLIGENCE"  
FOUNDING OF ARTIFICIAL INTELLIGENCE AS A RESEARCH DISCIPLINE  
"To proceed on the basis of the conjecture  
that every aspect of learning or any other feature of intelligence  
can in principle be so precisely described that a machine can be made to simulate it."  
IN COMMEMORATION OF THE PROJECT'S 50th ANNIVERSARY  
JULY 13, 2006

1970: For a parallel computer with  $n$  processors, the speedup  $S$  shall be proportional to  $\log_2(n)$

# Gene Amdahl's law



# What that means ?!

Grosch speaks about the economy of IT

- Grosch's Law appears to be a result of marketing policy more than of manufacturing expenses
- At some point of time, manufacturers priced their computers according to it

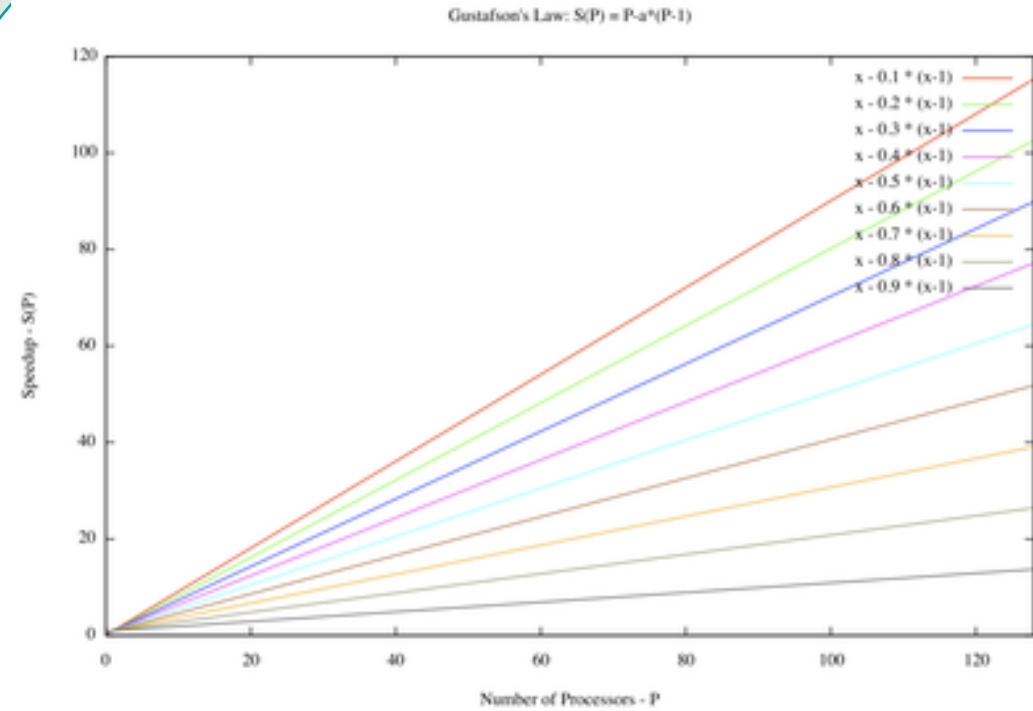
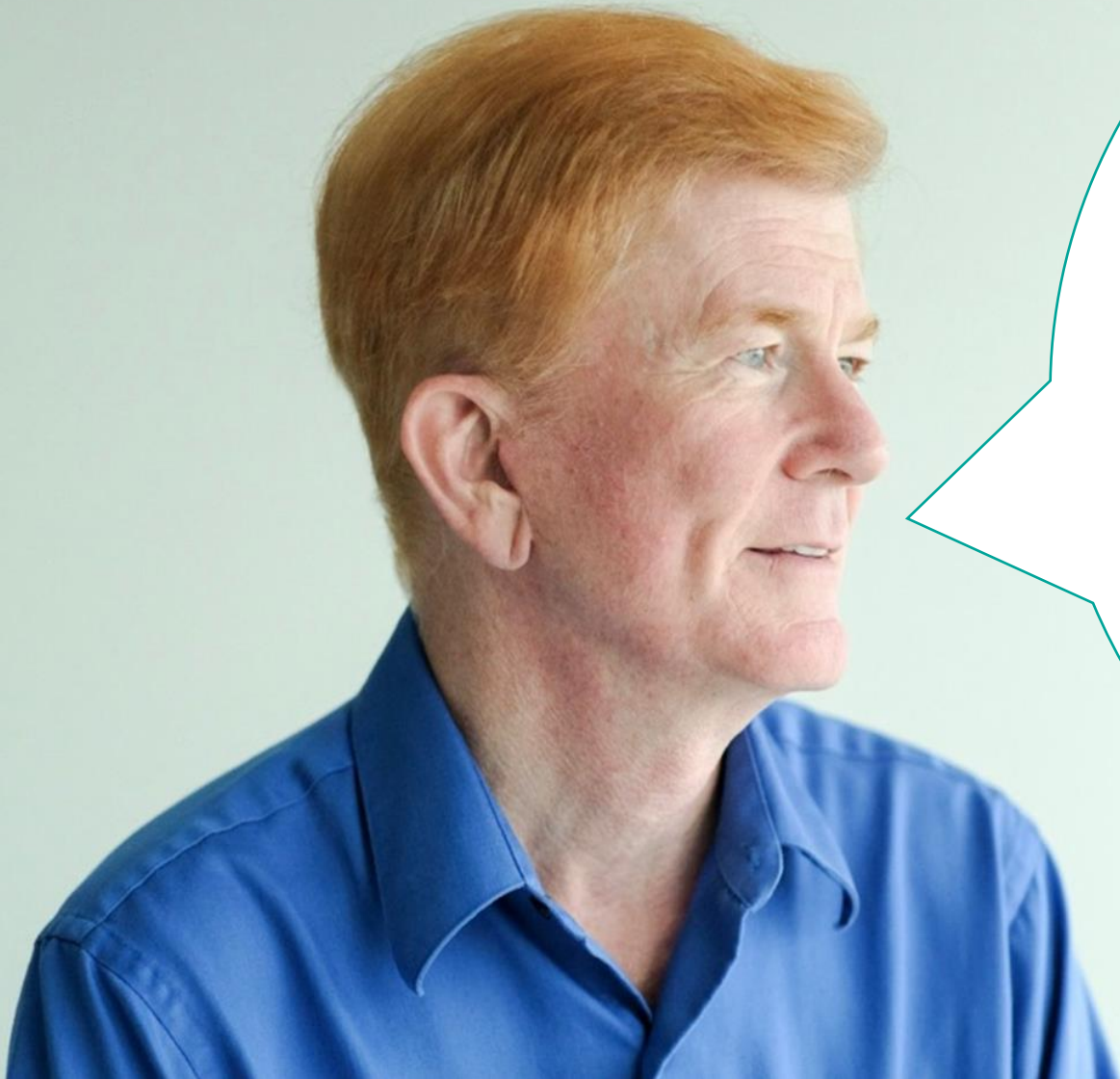
Amdahl's law was a strong argument against parallel computers

- The development of parallel algorithms would be too inefficient if it held
- Experimentally proven wrong [Sandia lab, 40-80% serial and scales linear]

Minsky's conjecture was based on technology of 60's and 70's

- Technologies like DMA and multitasking made it obsolete
- Can be still seen as worst-case scenario

# John Gustafson's law



# What that means (II) ?!

Gustafson overcomes the **fixed problem size** assumption by Amdahl

- Problems usually do not have fixed size
- Size of problems set to fully exploit the computing power
- Parallelism seen as an **opportunity** to solve larger-scale problems
- My favourite example: boot/load time