Computer Architecture

Multicore Processors, Vector Processors and GPUs

Dr. Mohammad Reza Selim

Lecture Outline

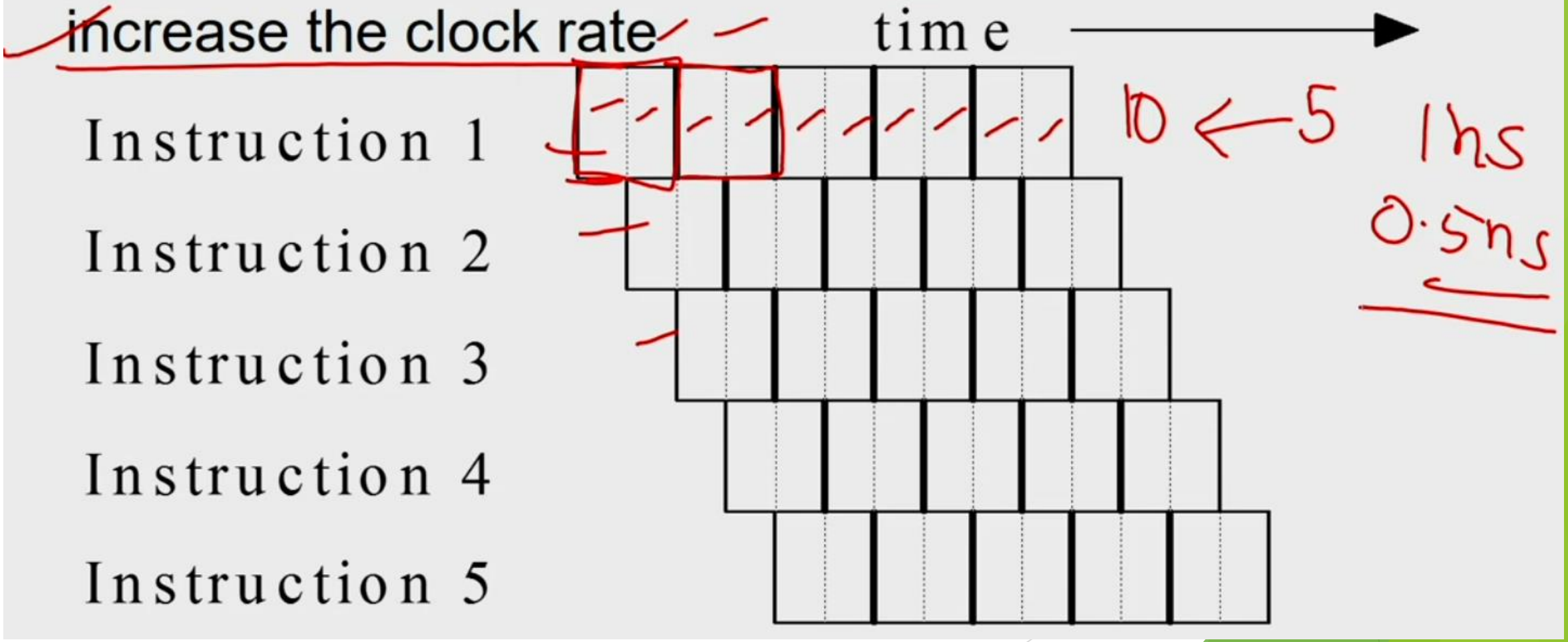
- Multi core processors
  - Paradigm shift to Multicore
  - Tiled Chip Multicore processors
  - Routers and Tiles
  - On chip address mapping
- Vector Processors
  - SIMD Architecture
  - Vector Architectures
  - VMIPS architecture and Its programming
- GPUs
  - Key Features
  - GPU vs CPU

Advanced Pipelining

- Increase the depth of the pipeline to increase the clock rate – **superpipelining**
- Fetch (and execute) more than one instructions at one time (expand every pipeline stage to accommodate multiple instructions) – **multiple-issue (super scalar)**
- Launching multiple instructions per stage allows the instruction execution rate, CPI, to be less than 1

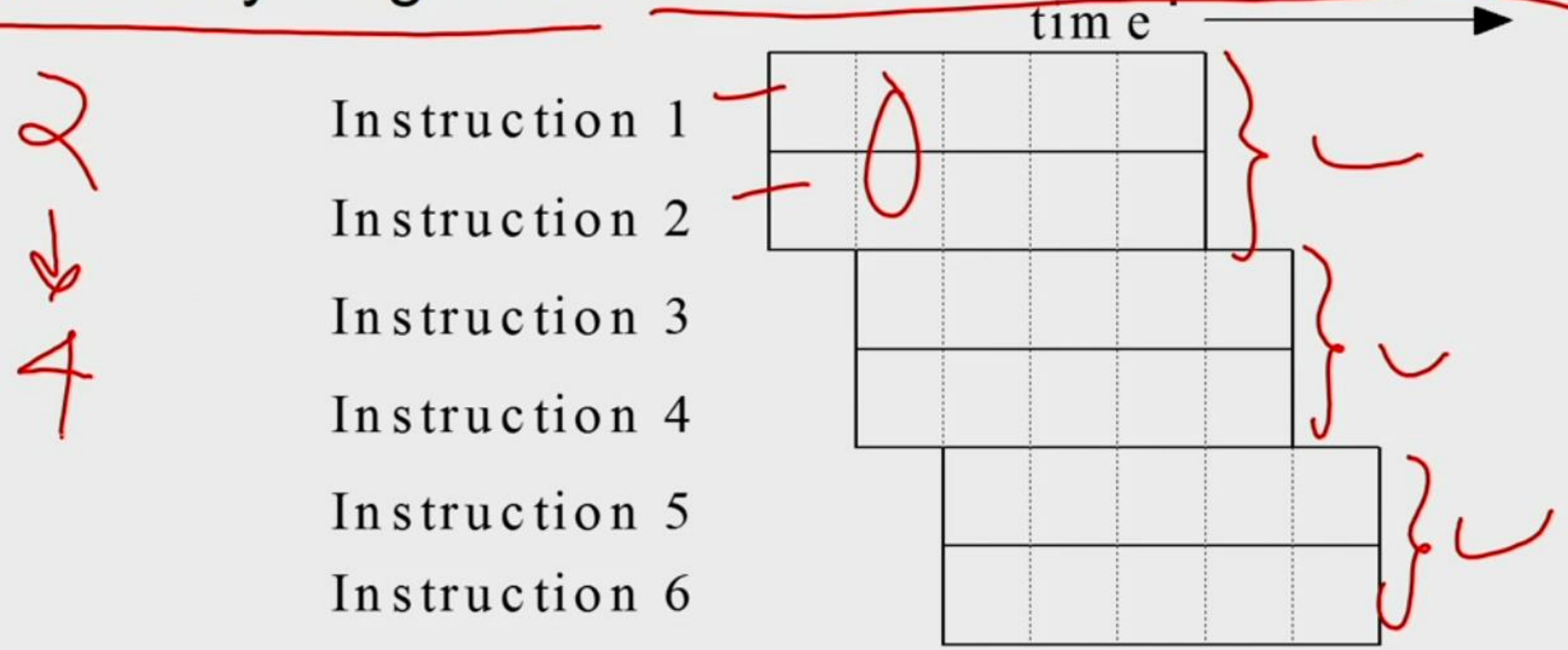
Superpipelining

❖superpipelining -Increase the depth of the pipeline to increase the clock rate



Superscalar - Multiple Issue

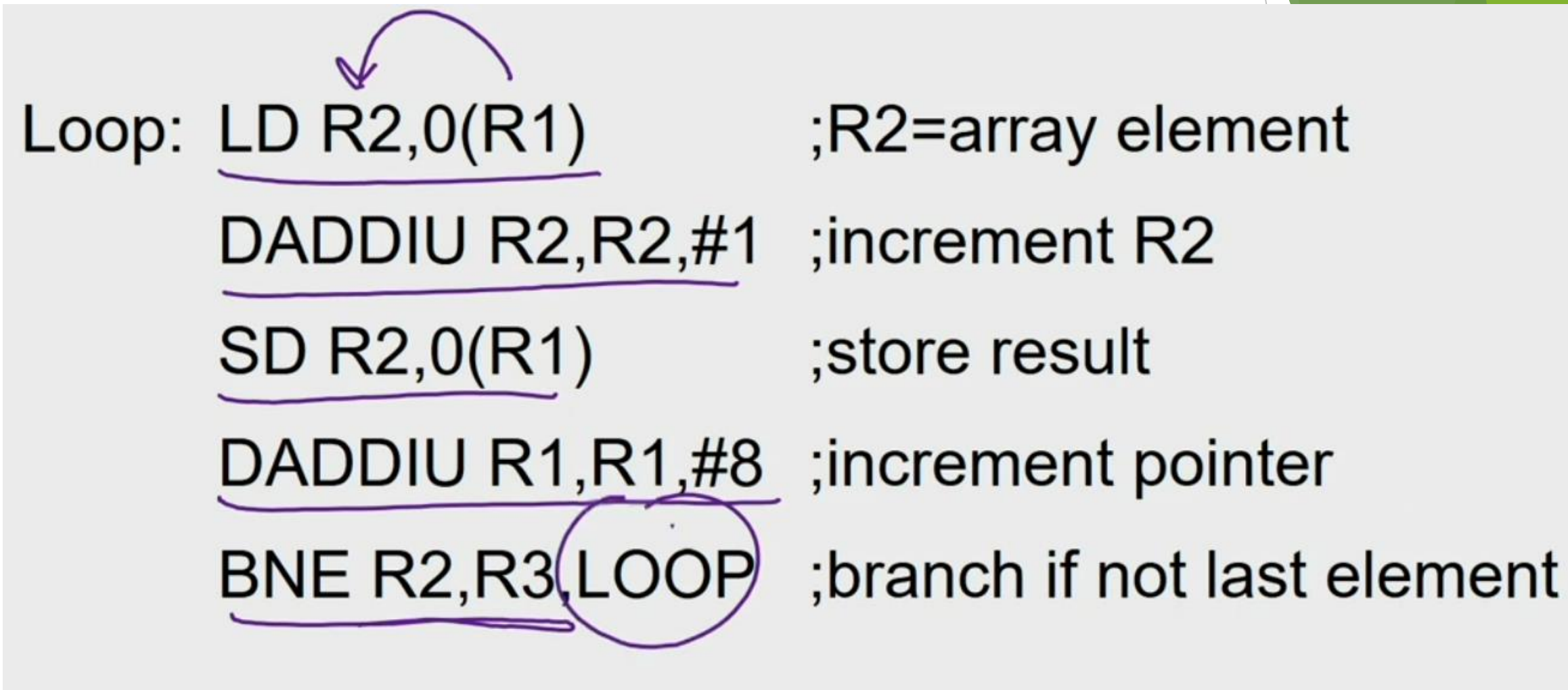
- Fetch (and execute) more than 1 instructions at one time
- Expand every stage to accommodate multiple instructions



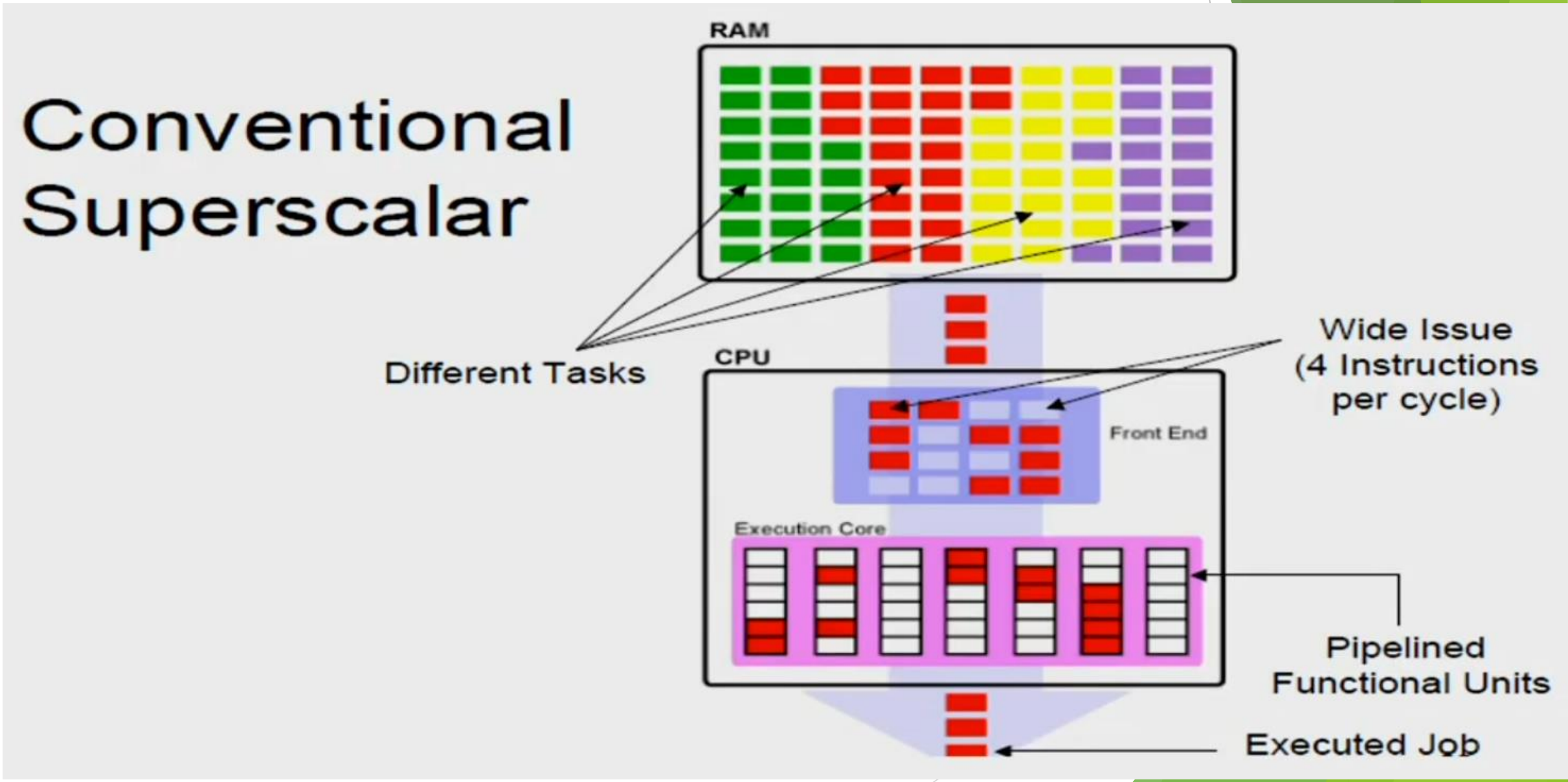
Extreme Optimization

- Modern micro-architectures uses this triple combination:
  - Dynamic scheduling + multiple issue + speculation
- Dependency between instructions issued in same clock.
- Register read for multiple instructions in parallel.
- Complex Issue logic to check dependencies and hazards.
- Assign reservation stations and ROB entries in a cycle.

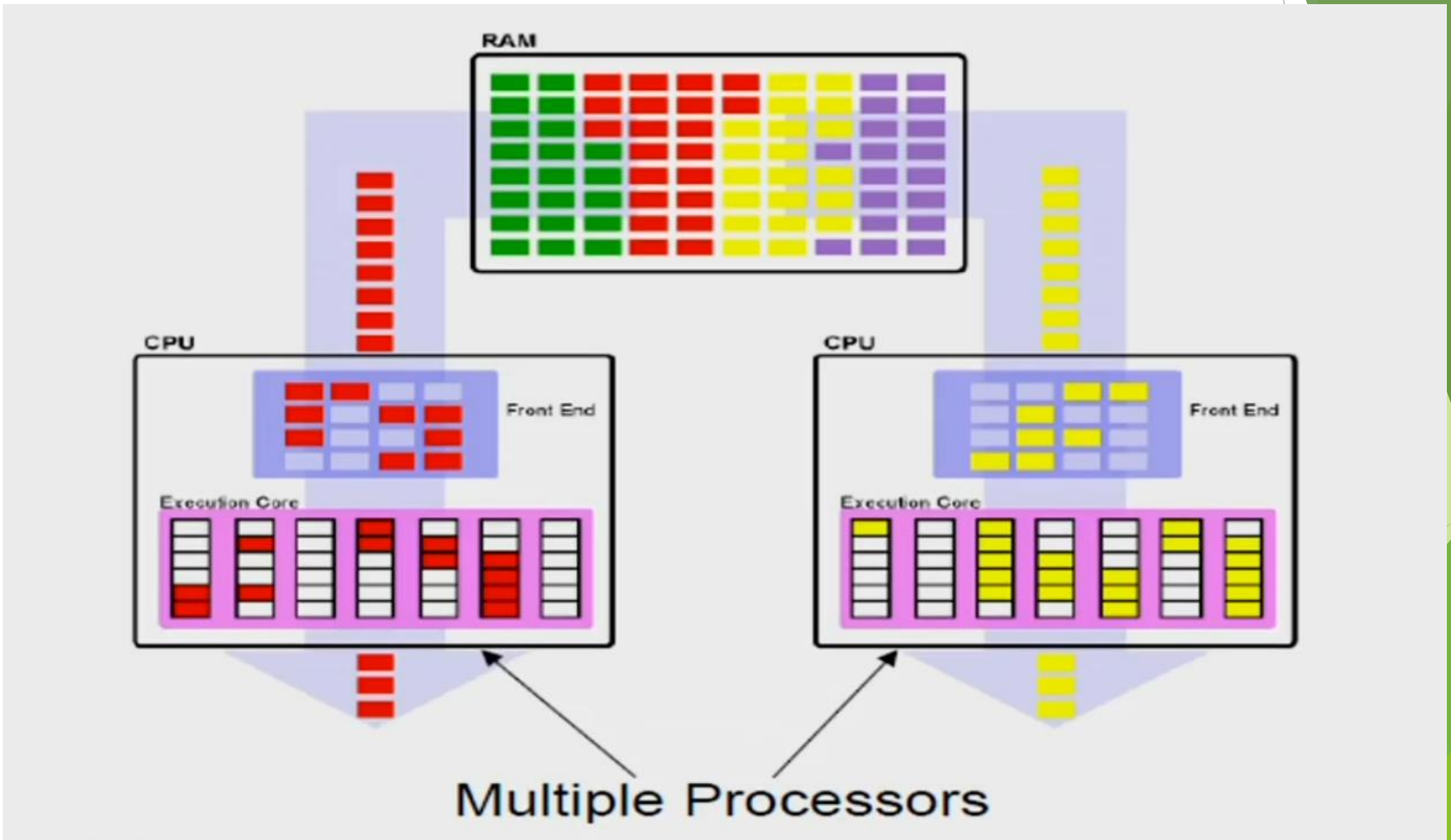
Example


Loop: LD R2,0(R1) ;R2=array element
DADDIU R2,R2,#1 ;increment R2
SD R2,0(R1) ;store result
DADDIU R1,R1,#8 ;increment pointer
BNE R2,R3,LOOP ;branch if not last element

Superscalar Processor



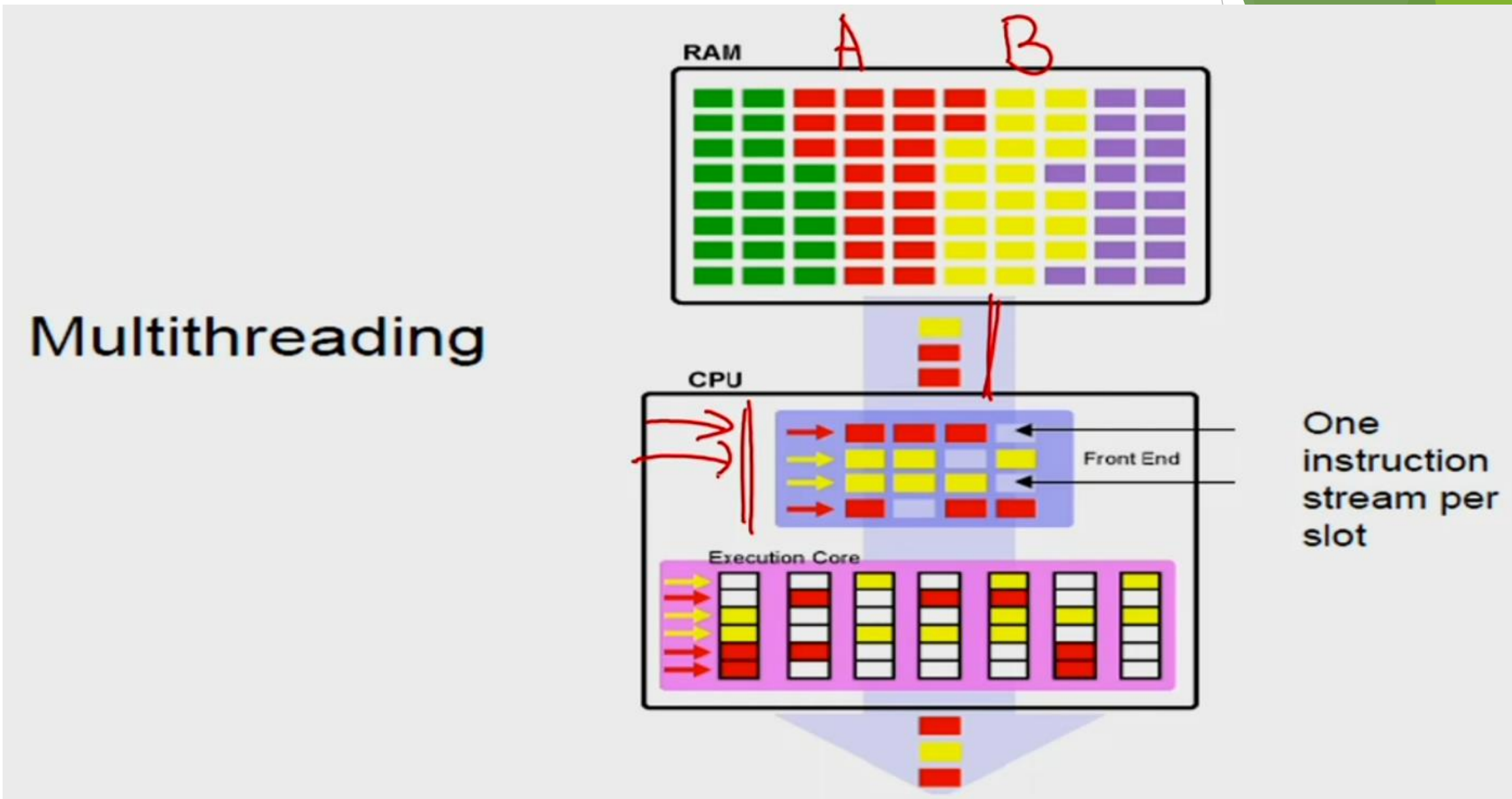
Symmetric Multiprocessor



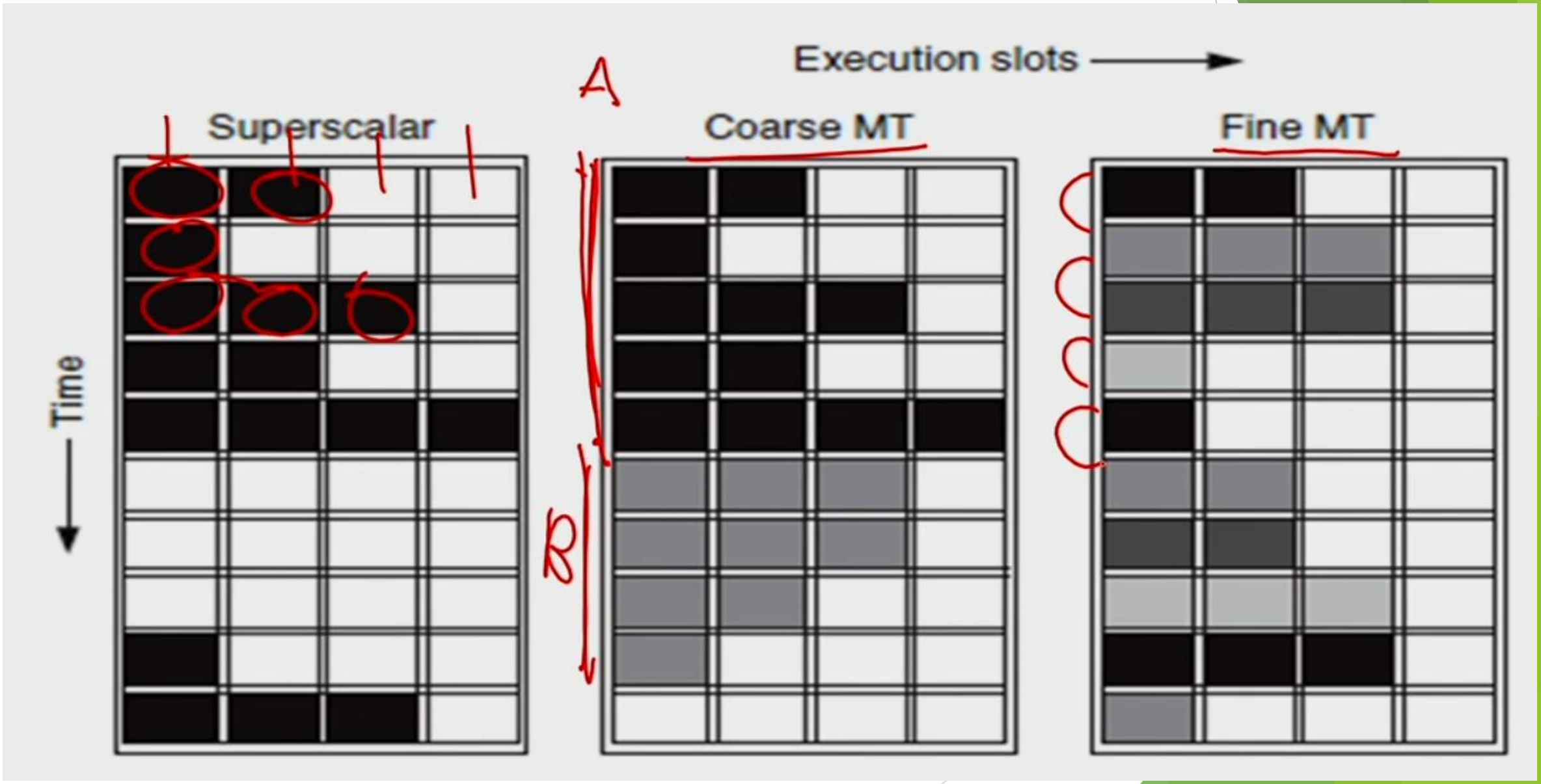
Is that Enough?

- We have tried Tomasulo's superscalar approach with ROB, speculation with aggressive branch prediction and multi issue
- At very high issue rates cache misses that go to L2 and off chip can not be hidden by ILP.
- How to cover such long memory stalls ?
- **Multithreading allows multiple threads to share the functional units of a single processor in an overlapping fashion**

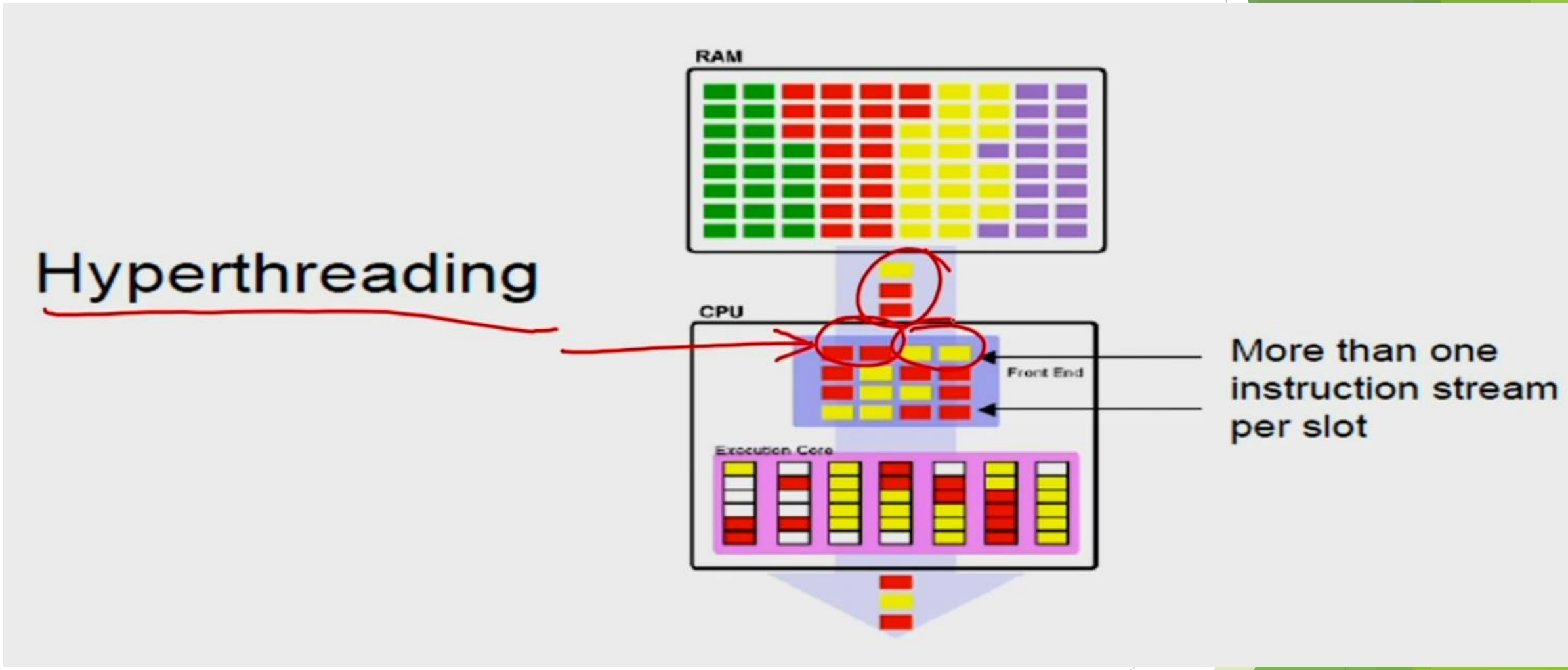
Multithreading



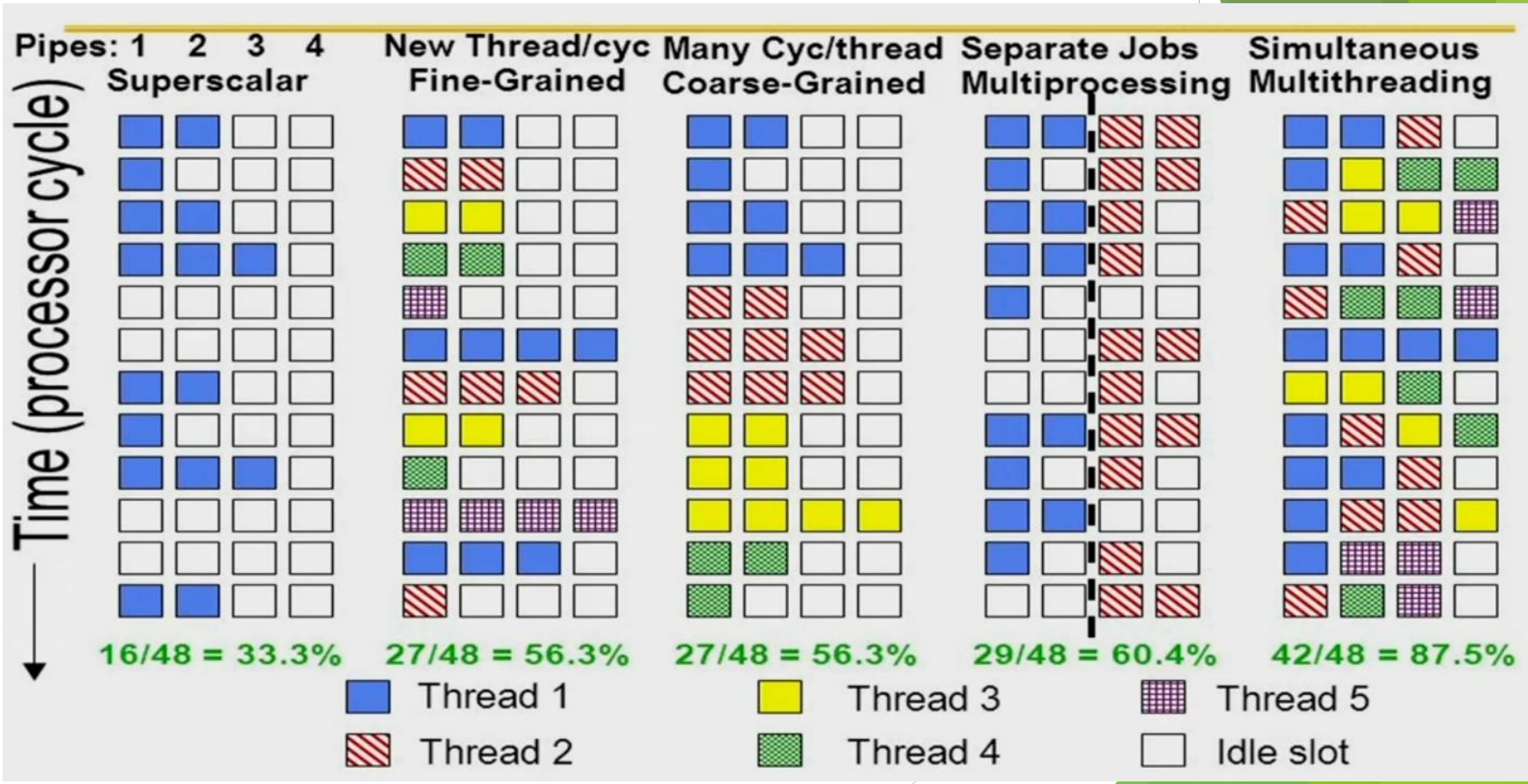
Multithreading



Hyperthreading /SMT

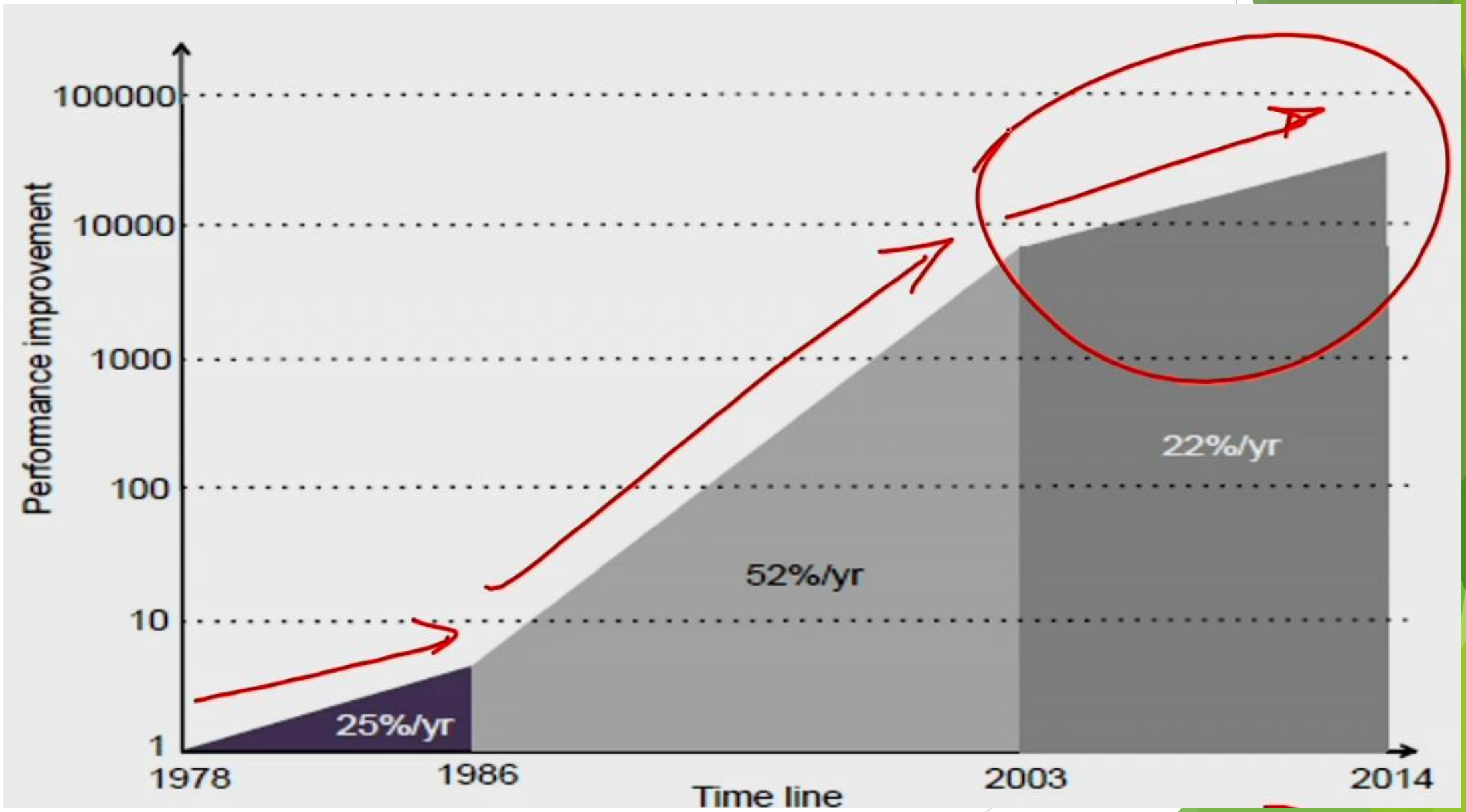


Comparison of Resource Utilization

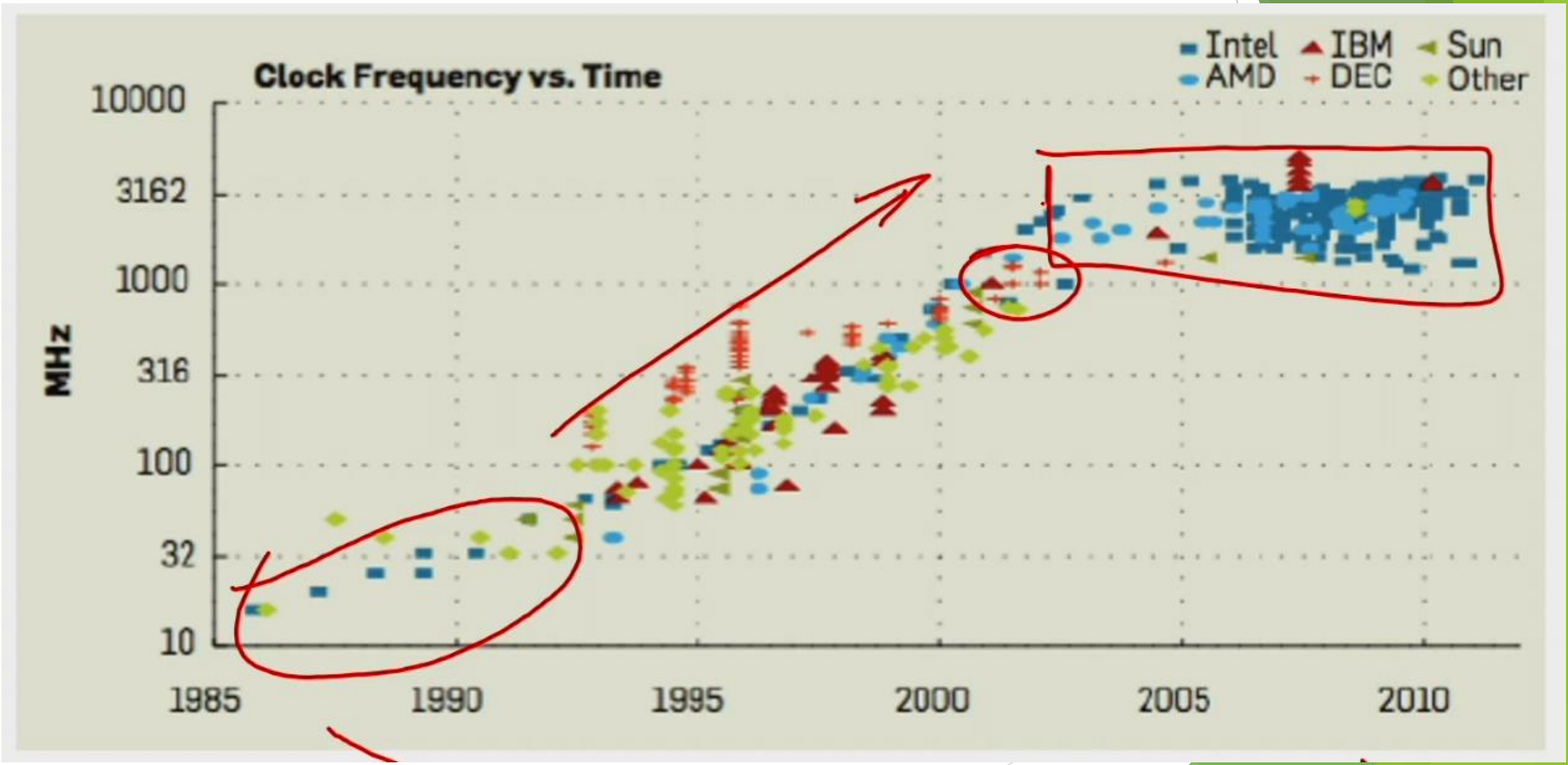


Multicore Processors

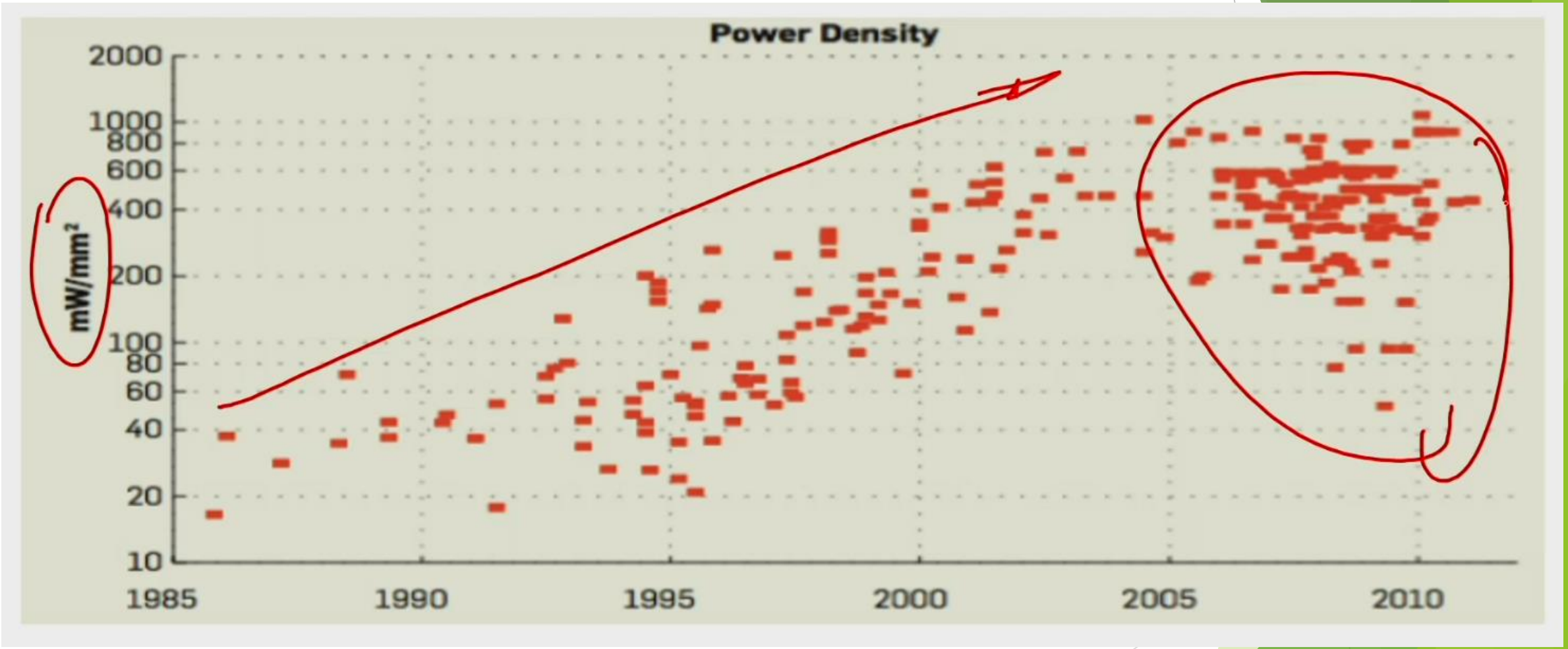
Processor Performance



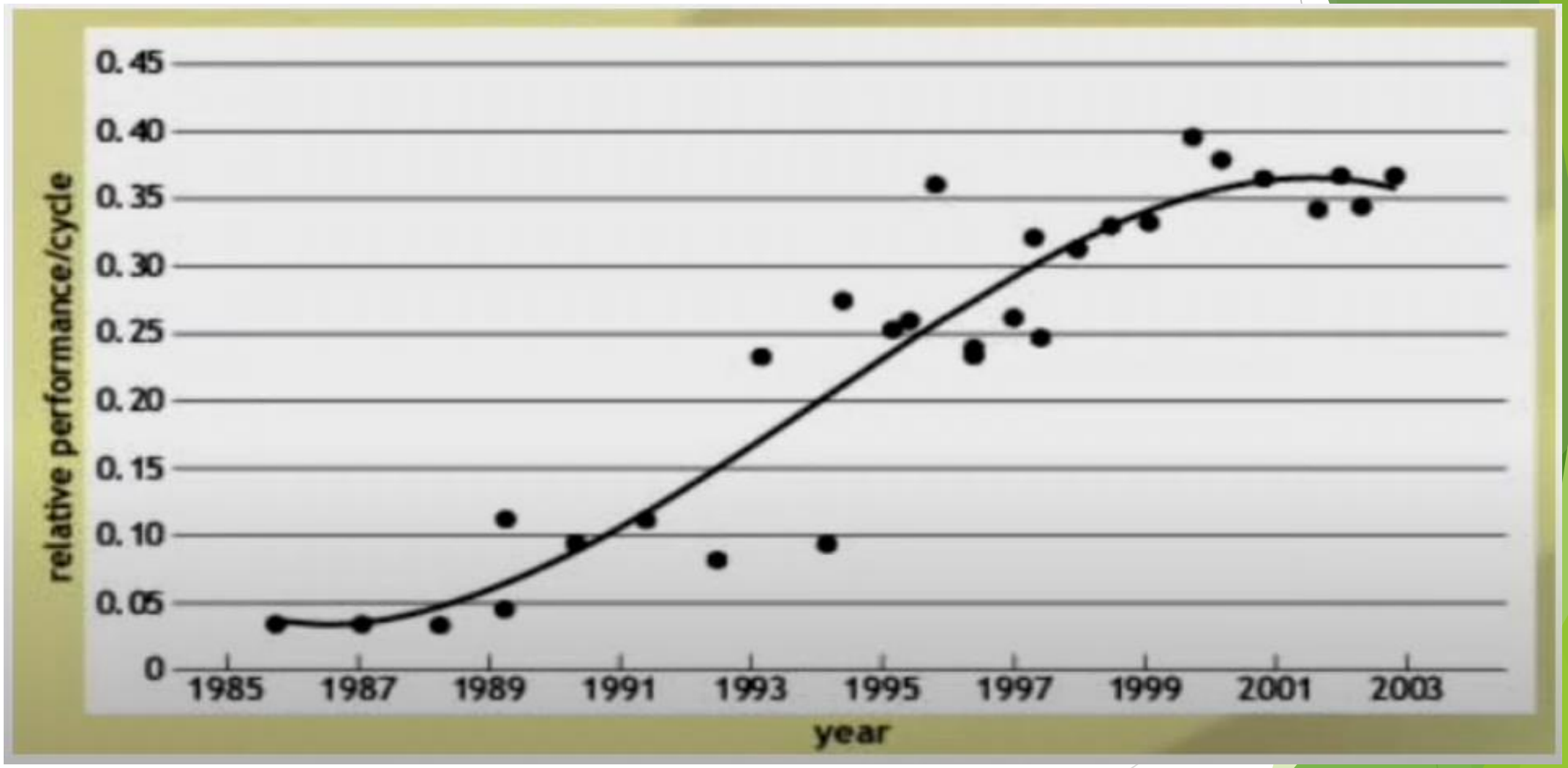
Frequency Wall



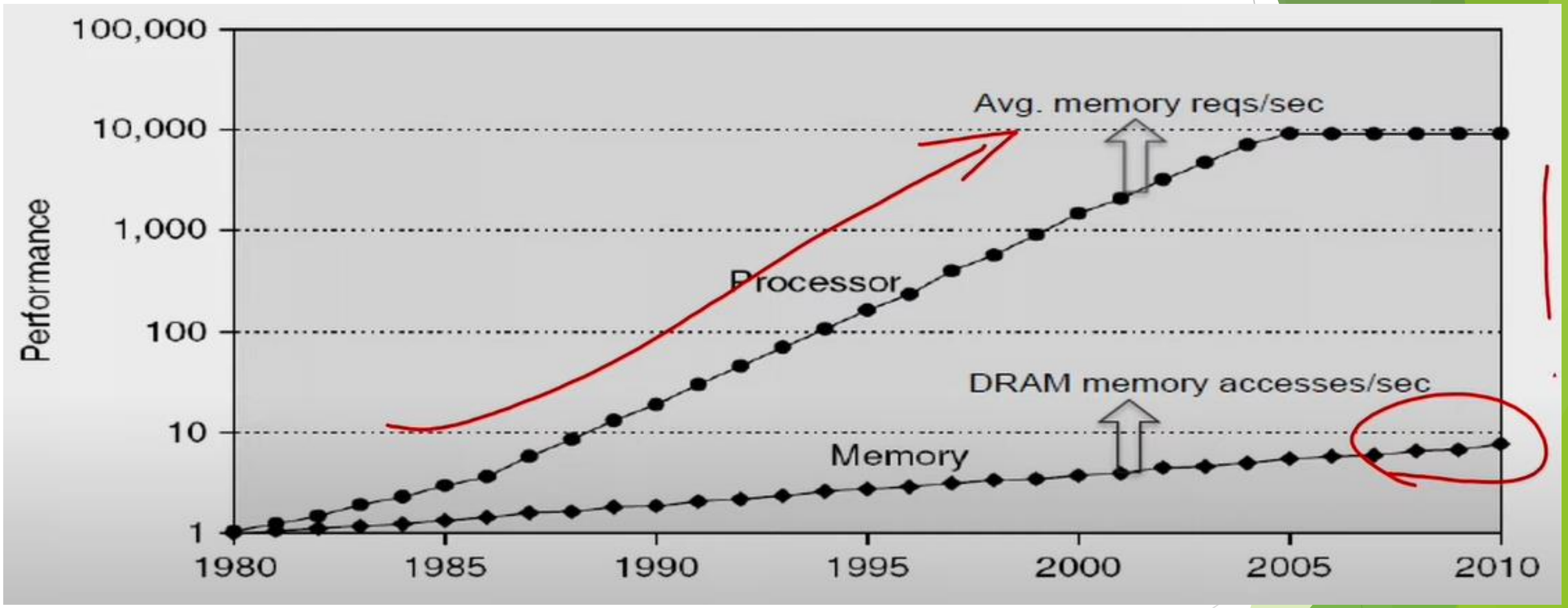
Power Wall


Power Density
2000
1000
800
600
400
200
100
80
60
40
20
10
mW/mm²
1985
1990
1995
2000
2005
2010

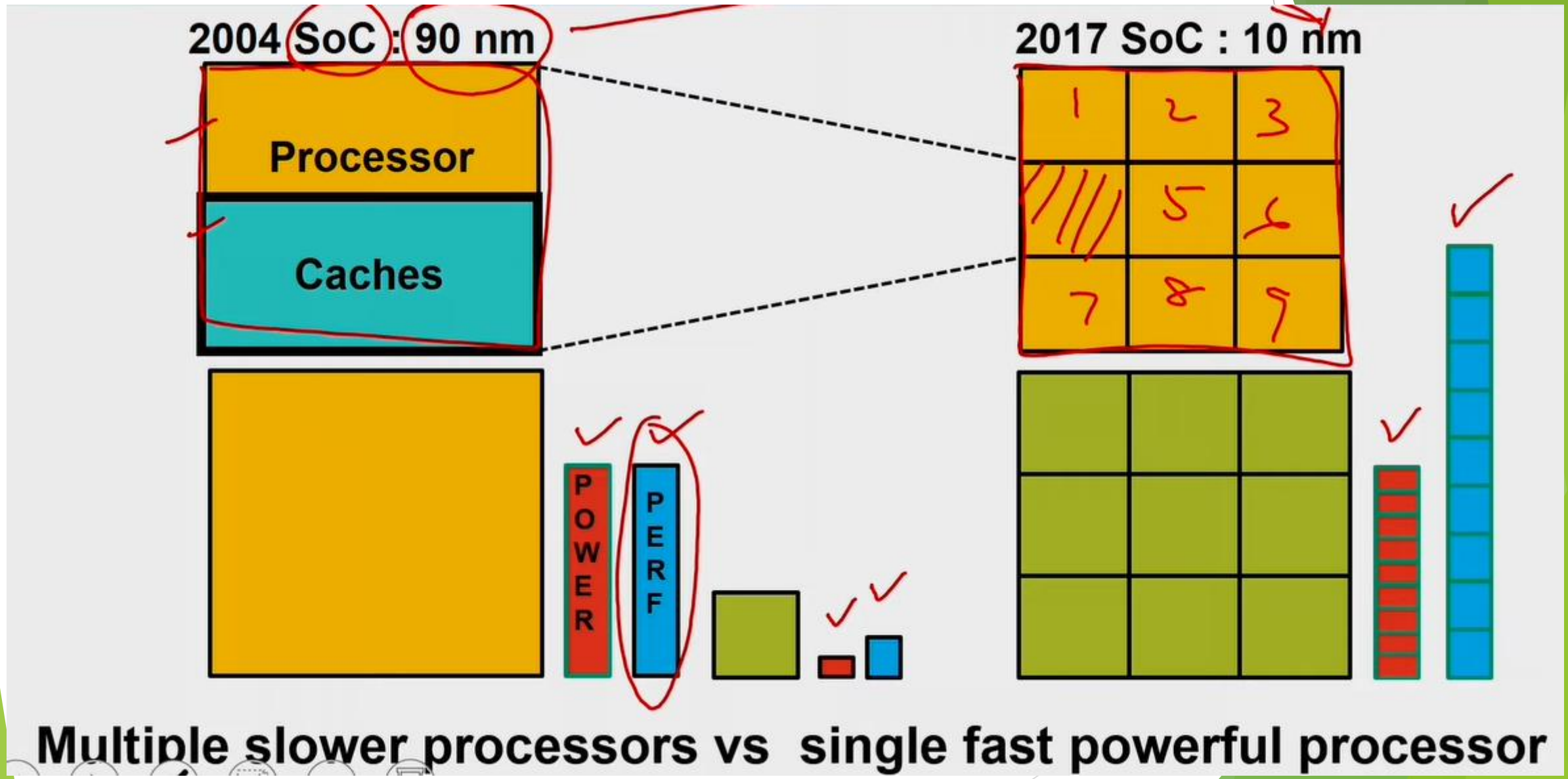
ILP Wall


relative performance/cycle
0.45
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0
1985
1987
1989
1991
1993
1995
1997
1999
2001
2003
year

Memory Wall

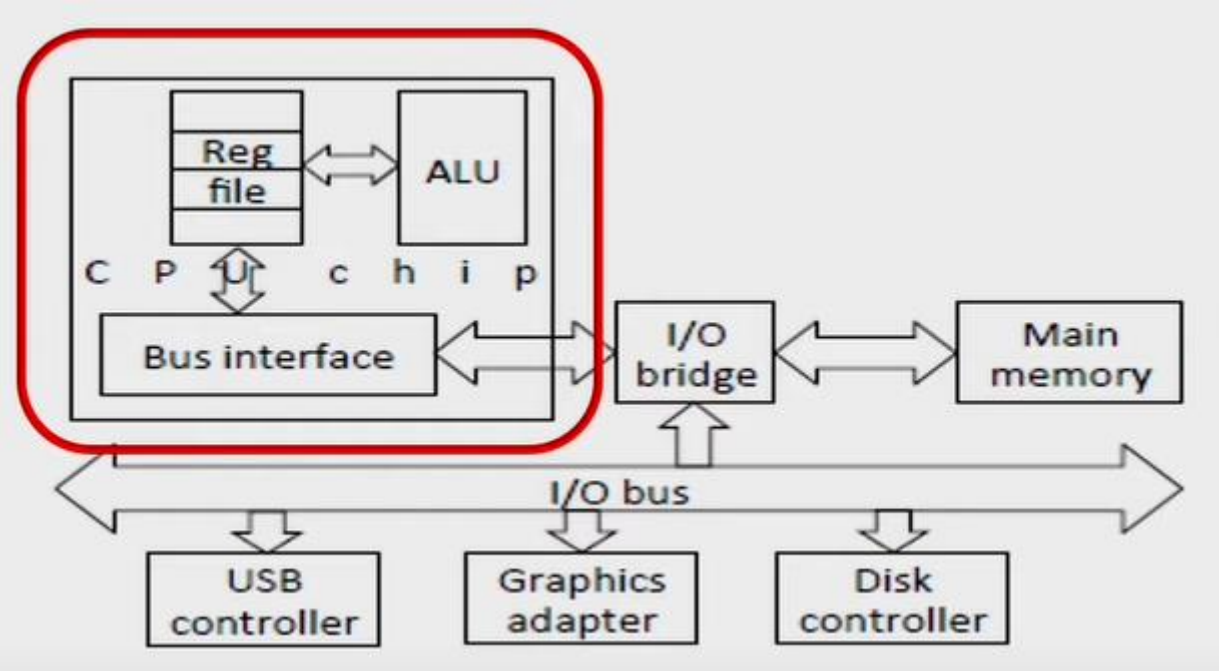

100,000
10,000
1,000
100
10
1
Performance
1980
1985
1990
1995
2000
2005
2010
Avg. memory reqs/sec
Processor
DRAM memory accesses/sec
Memory

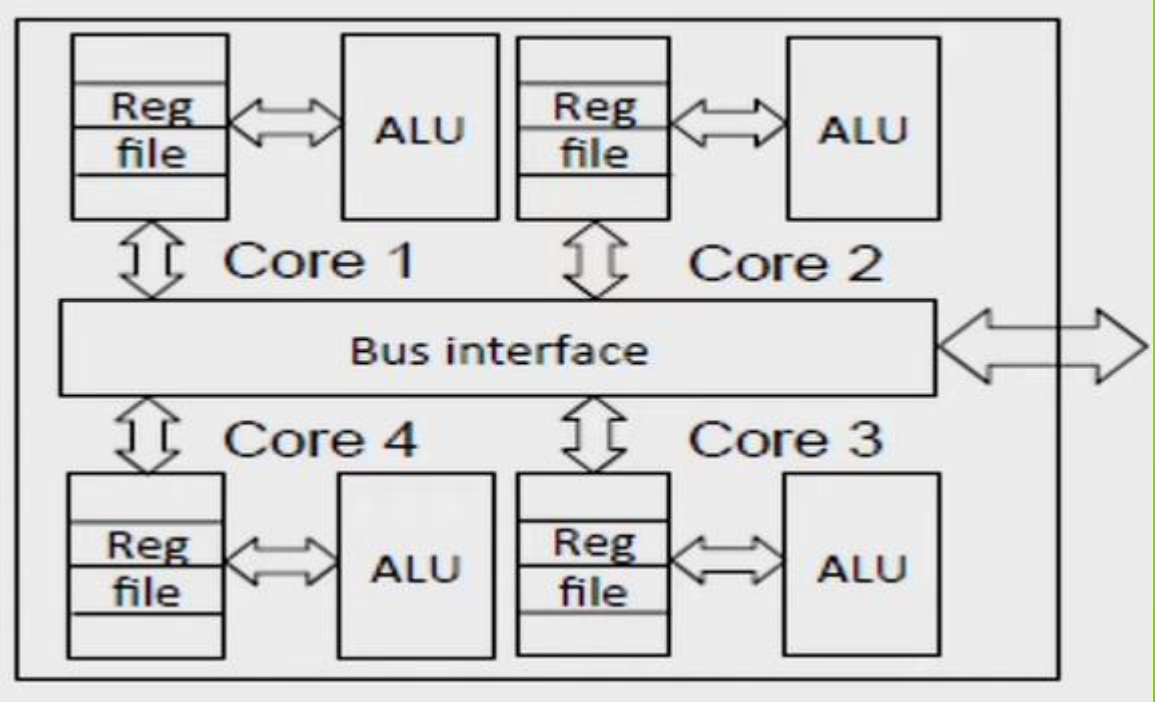
Paradigm Shift to Multicore



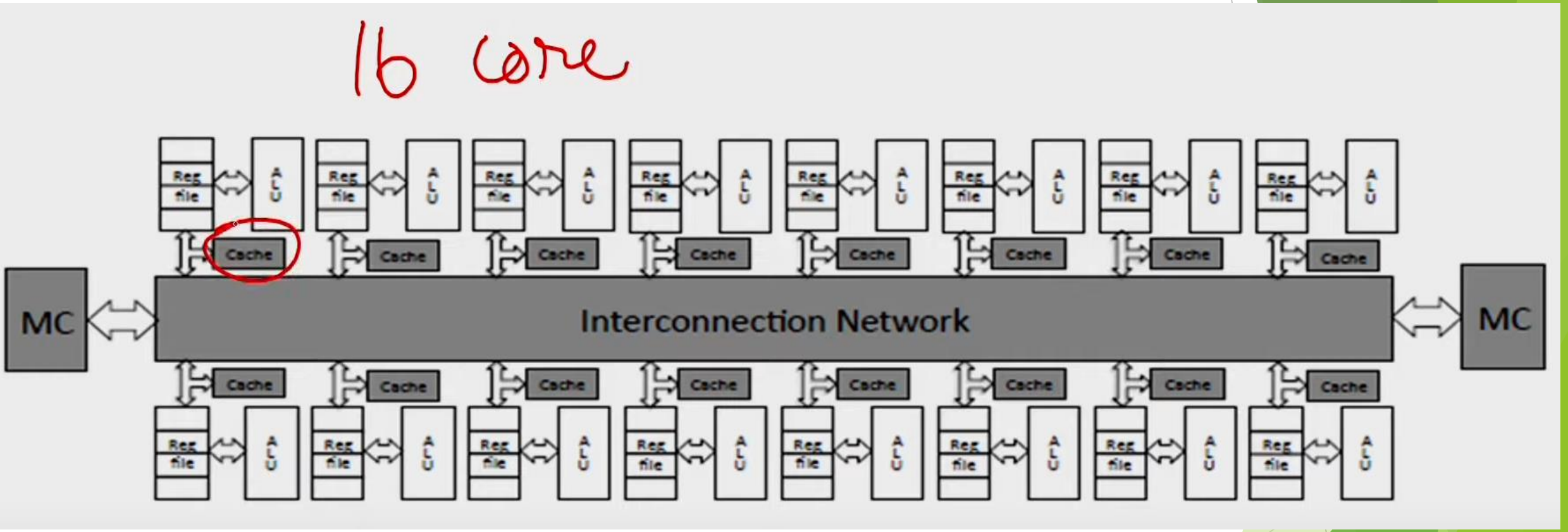
Paradigm Shift to Multicore

What is Multicore

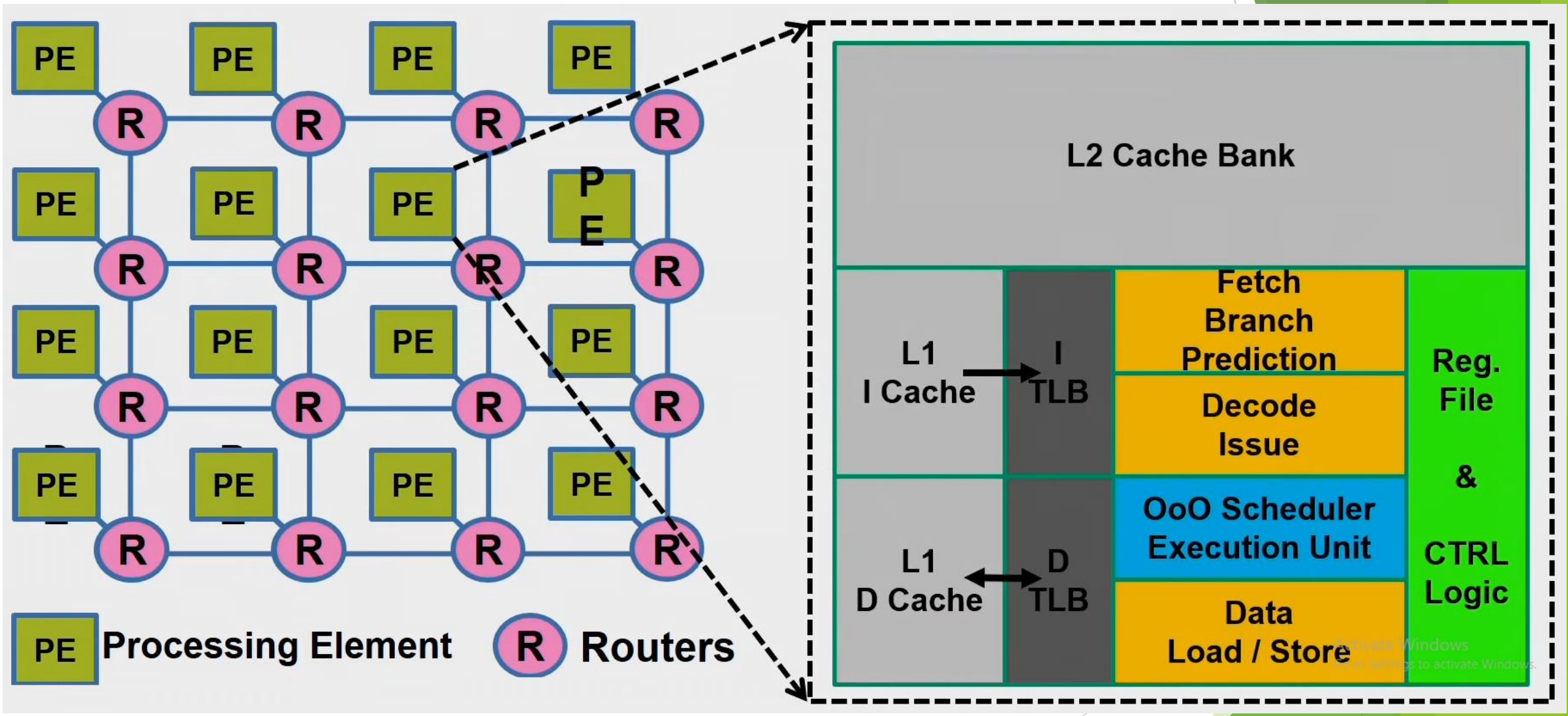




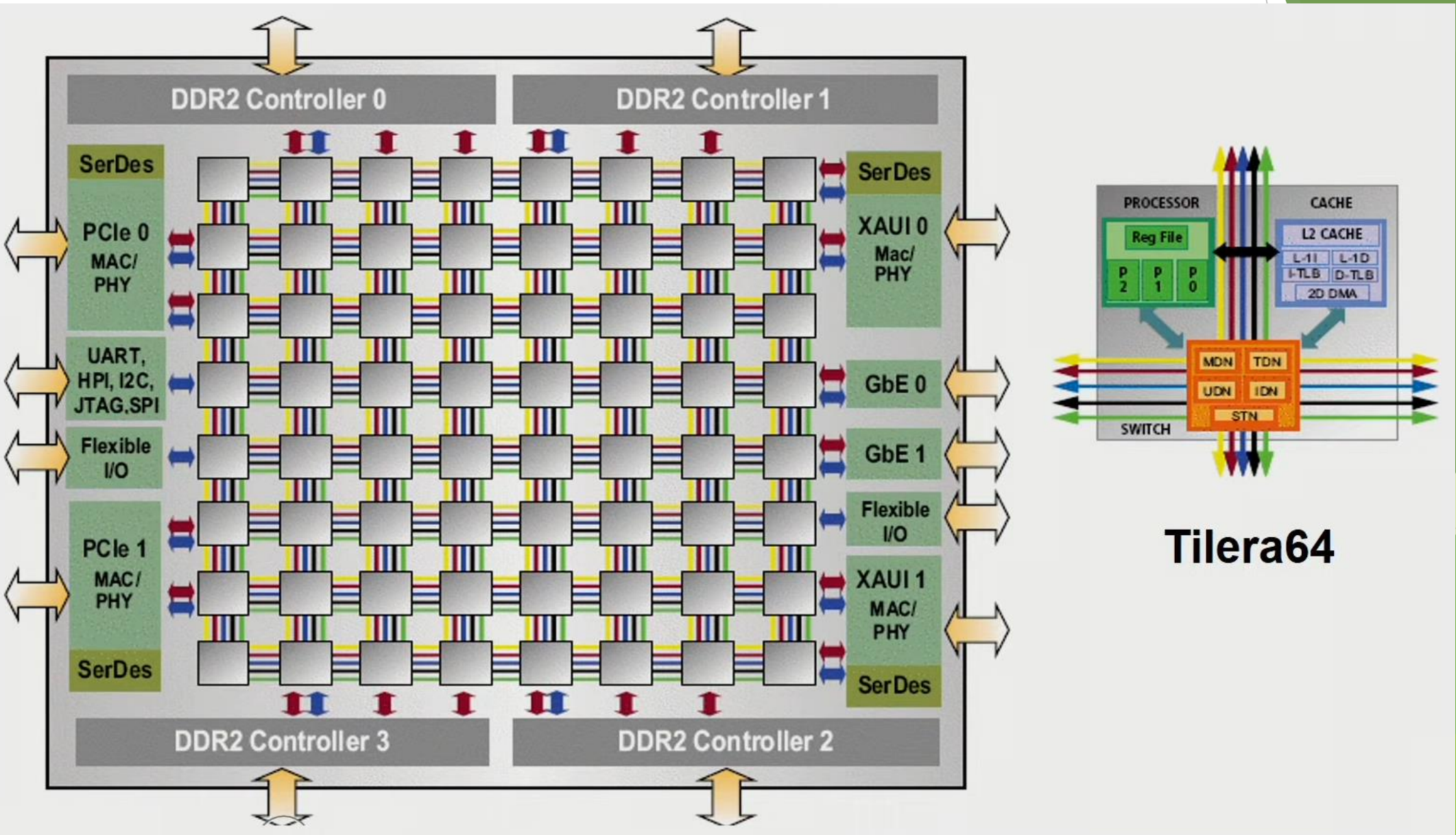
What is Multicore



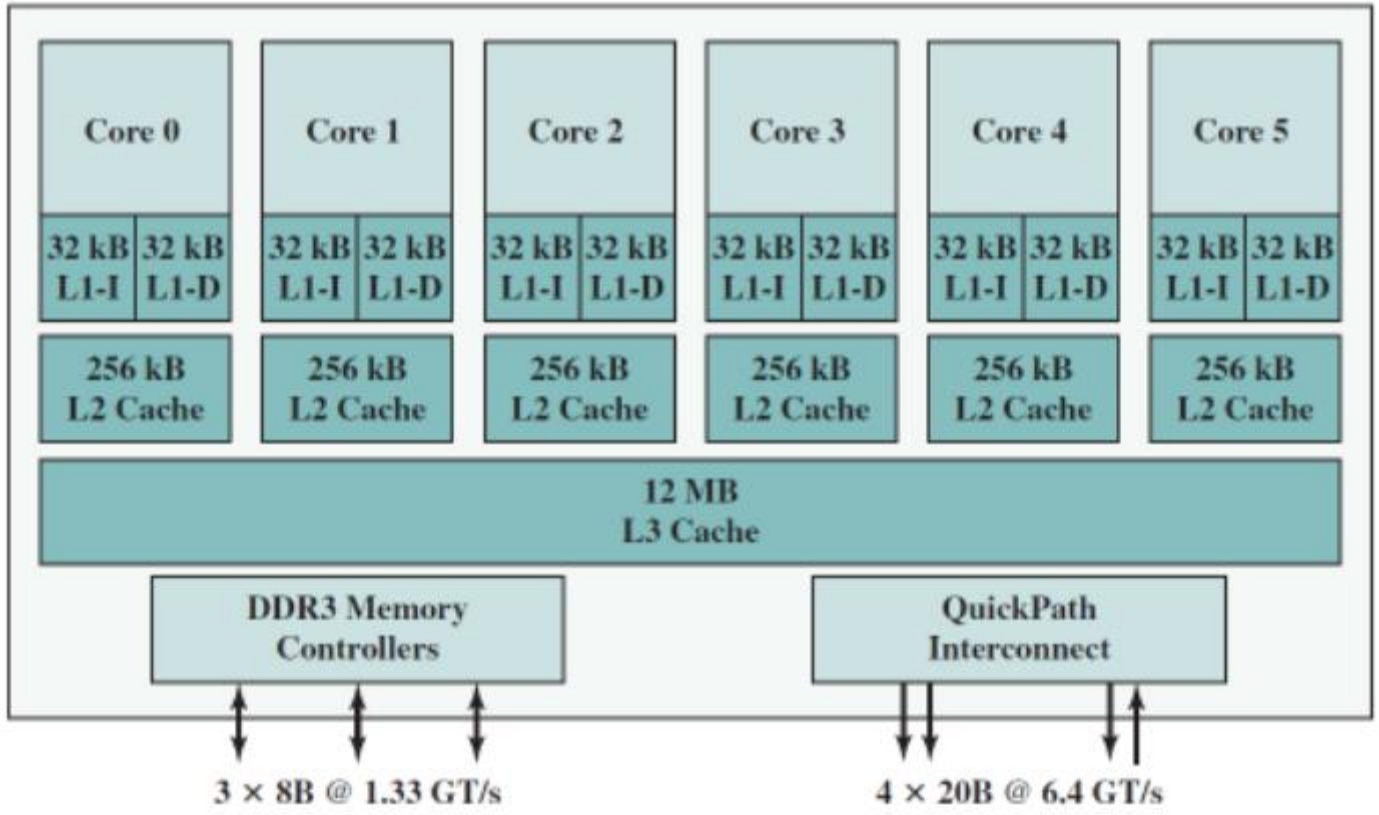
Tiled Chip Manycore Processor (TCMP)



State of the Art Architectures

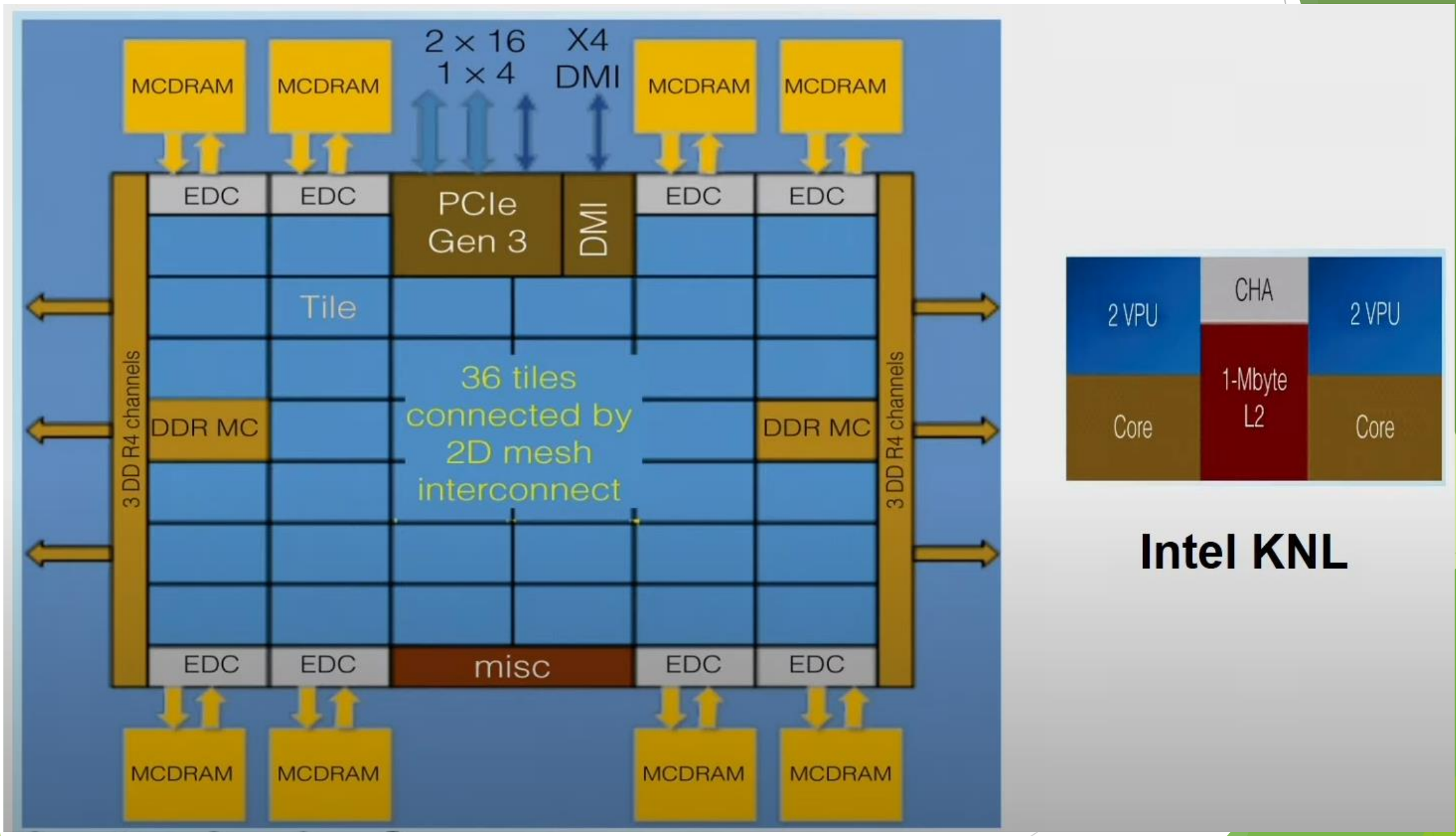


What is State of the Art Architectures

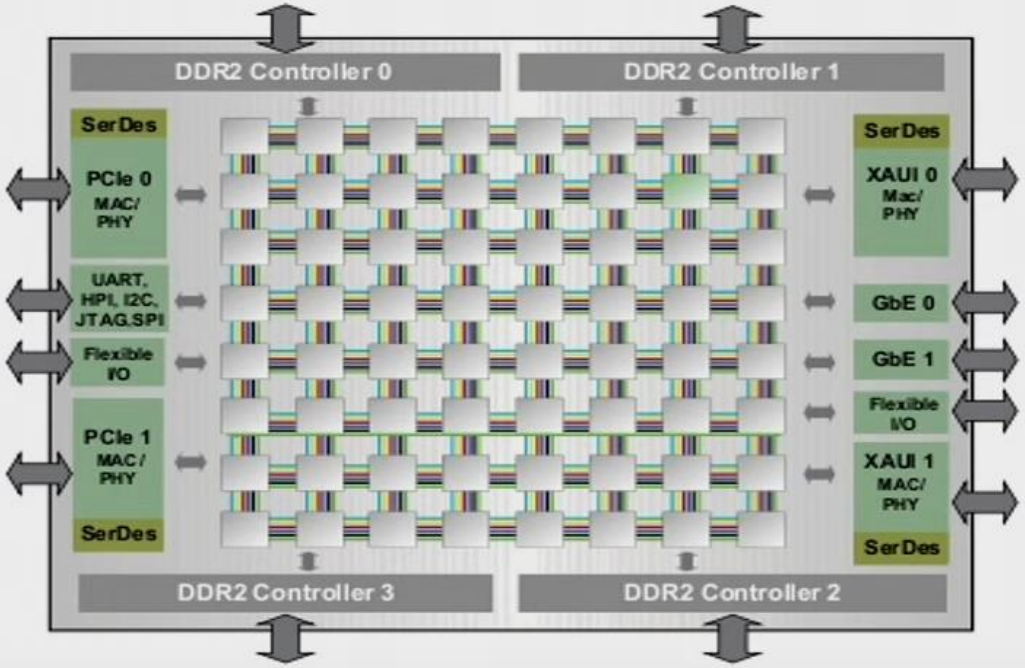


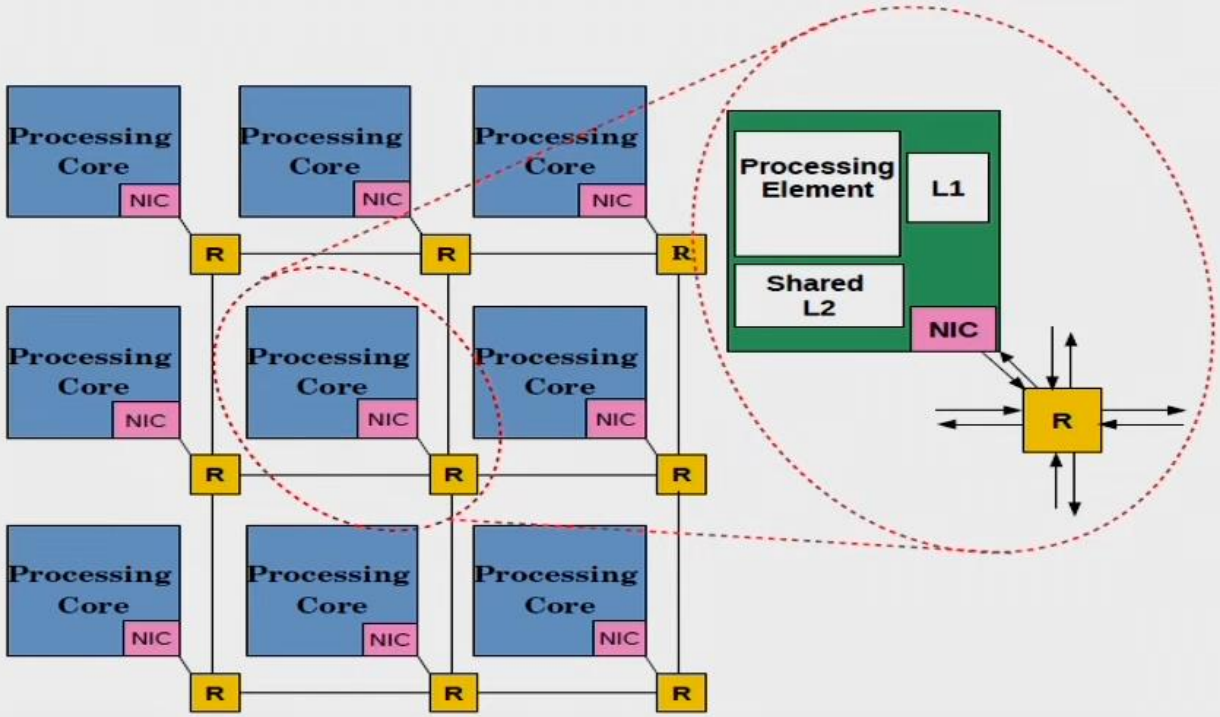
Intel Core i7 990X

State of the Art Architectures



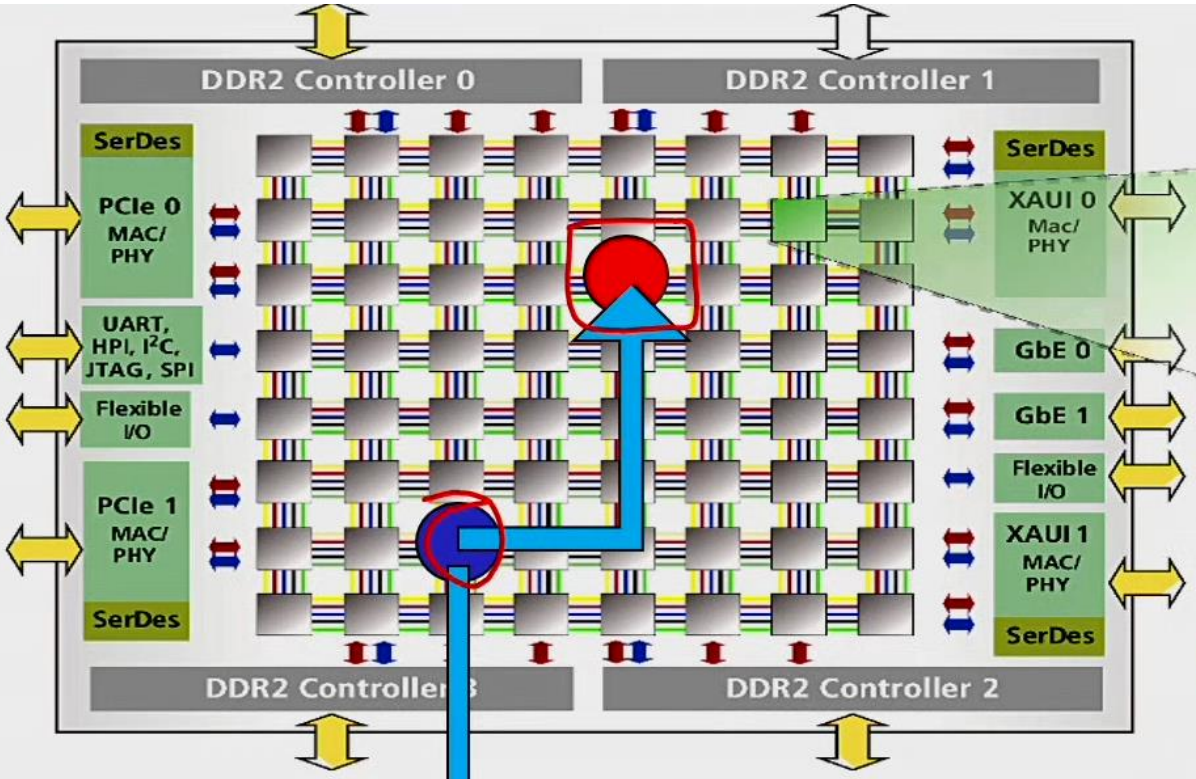
Routers and Tiles

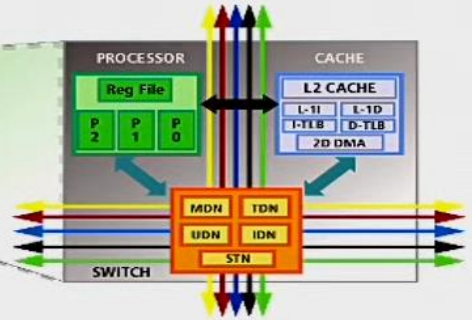




- East, West, North and South neighbors
- Packets are divided into flow control units called flits
- L1 and L2 cache misses create NoC traffic packets

On Chip Cache Address Mapping





512 x 64 = 32MB

Per core: 512 KB

16-way set associative mapping

64 Bytes/block

Consider a 4GB RAM (2^{32} bits)

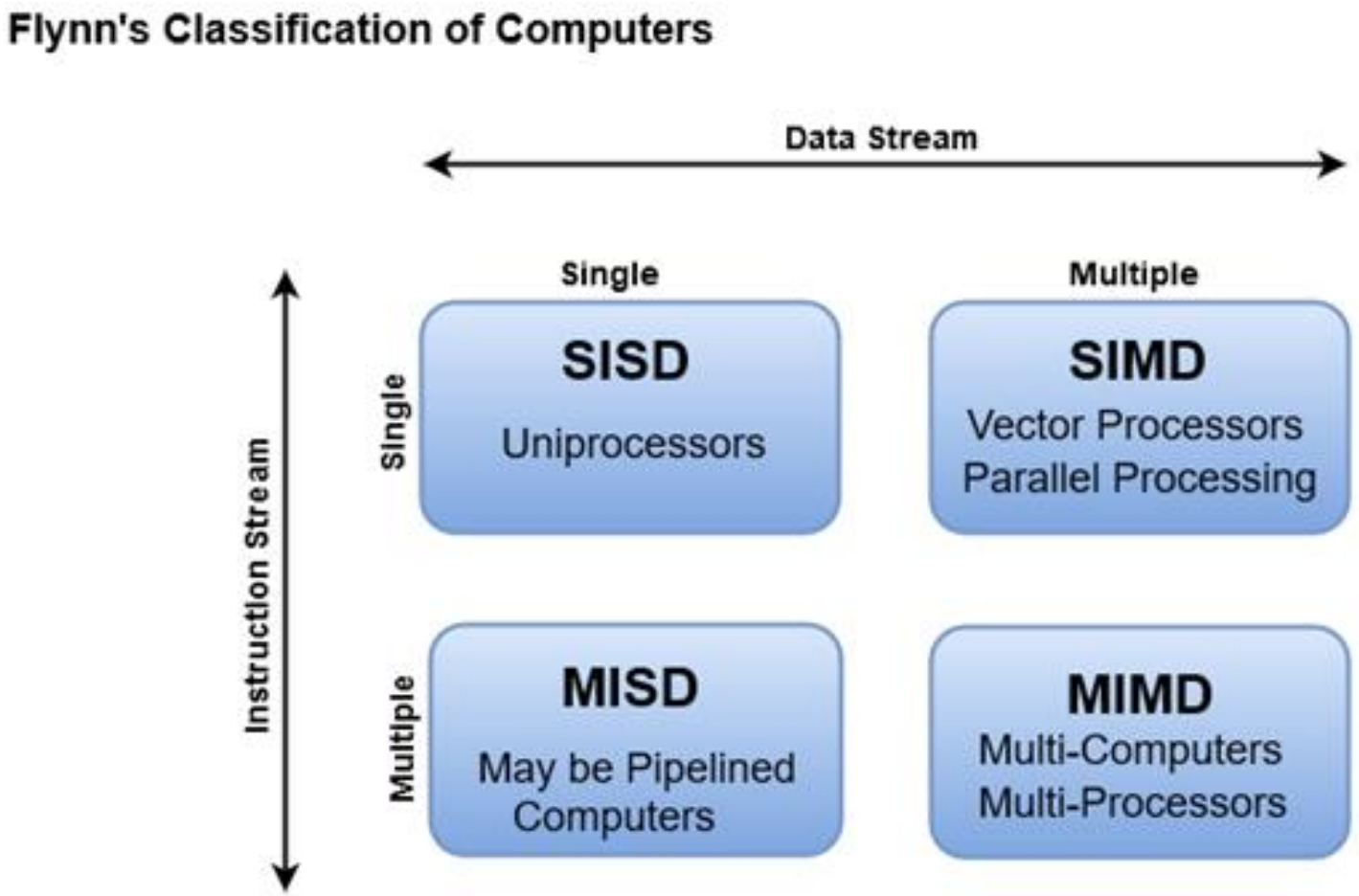
Hex Address : 0x764254

11 bits	15 bits	6 bits
Tag	Index	Block offset

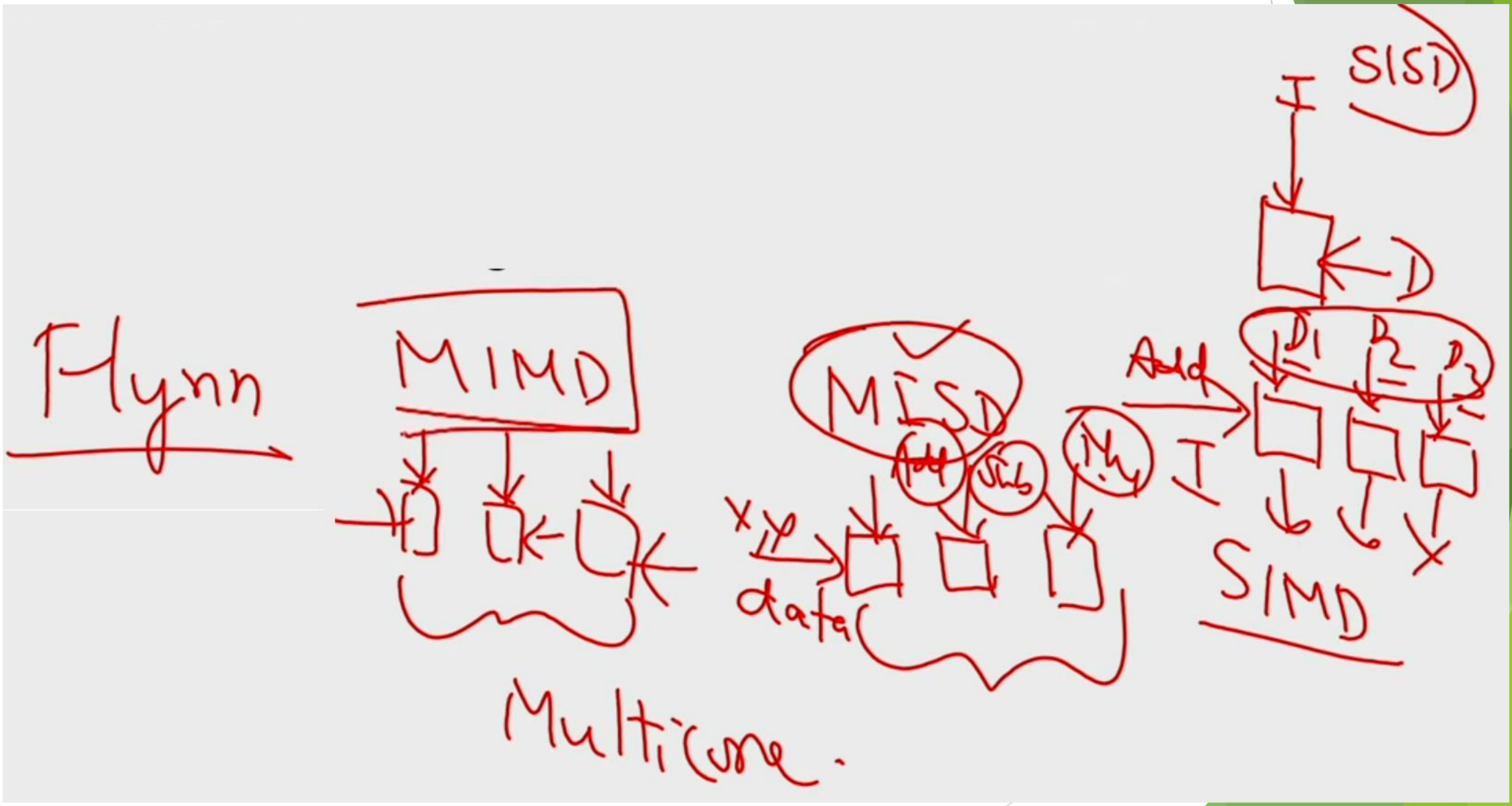
11 15 6

Vector Processors

Flynn’s Classification



Flynn's Classification

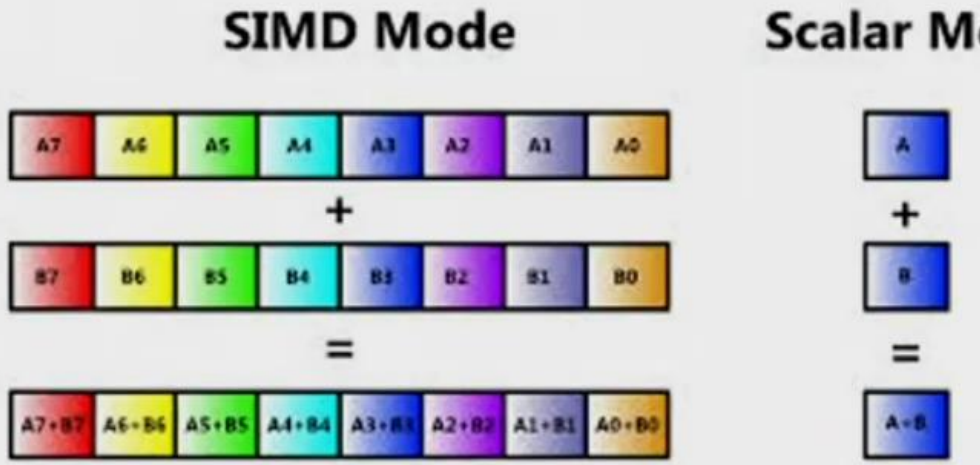


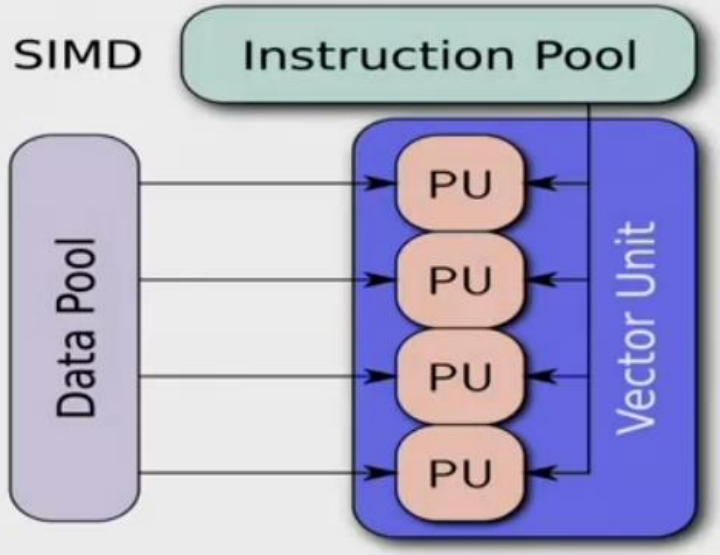
SIMD Architectures

- **SIMD architectures exploit data-level parallelism**
  - matrix-oriented scientific computing
  - media-oriented image and sound processing
- **SIMD is more energy efficient than MIMD**
  - Only needs to fetch one instruction per data operation
  - Makes SIMD attractive for personal mobile devices

SIMD Architectures

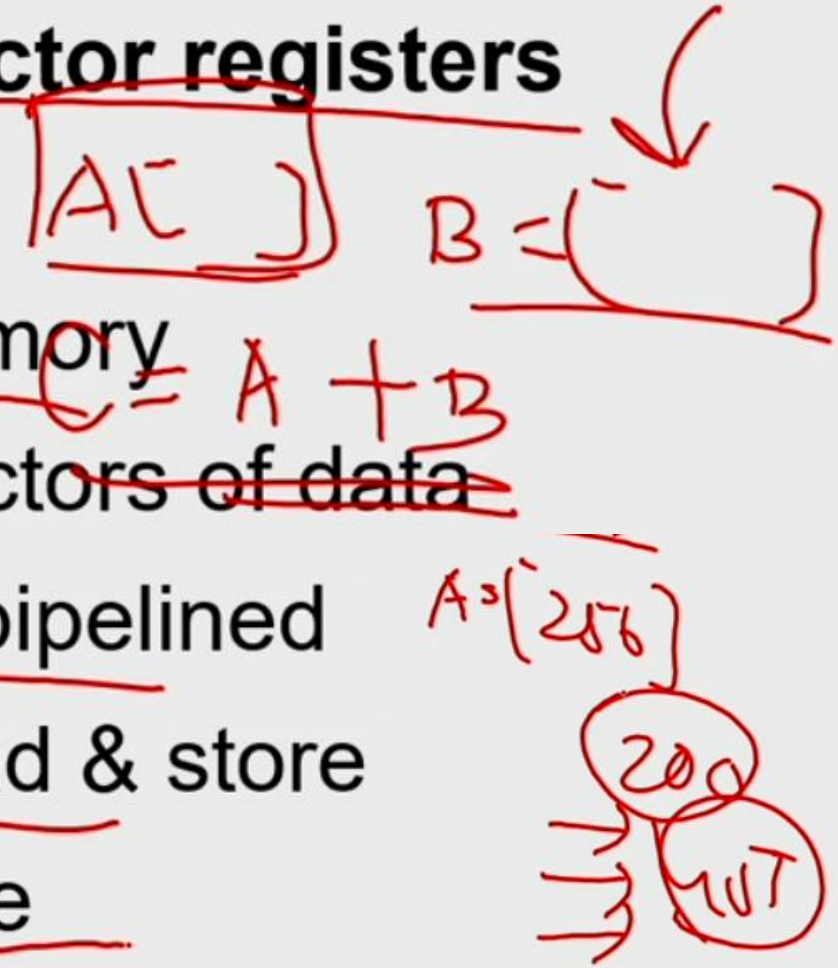
- Vector Processors (VPs)
- Graphics Processor Units (GPUs)





Vector Architectures

- Read set of data elements into **vector registers**
- Operate on **vector registers**
- Disperse the results back into memory
- Single instruction operate on a vectors of data
- Vector load and store are deeply pipelined
- High memory latency once per load & store
- Amortize the latency over load size

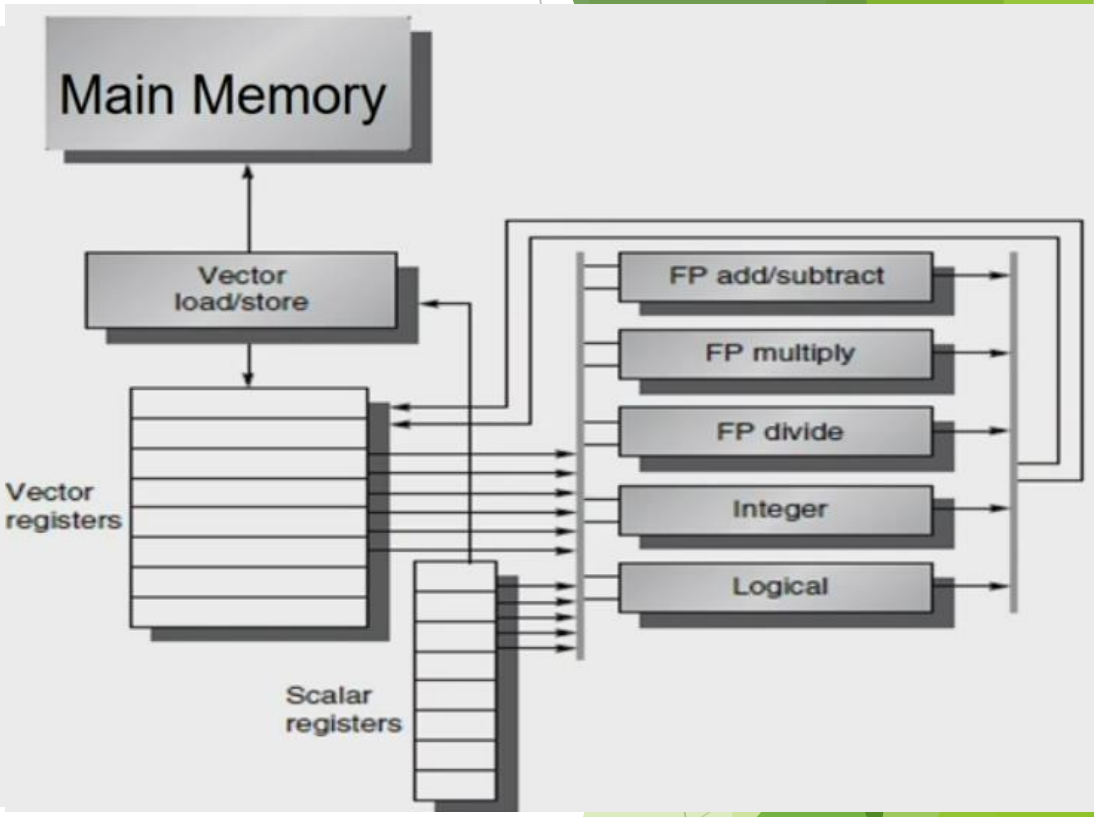


VMIPS Architecture

- **Vector registers**
  - 8 vector registers; each register holds 64-elements
  - 64 bits/vector element **[1 VR=512 Bytes]**
  - Register file has 16 read ports and 8 write ports
- **Scalar registers**
  - 32 GPRs, 32 FPRs : Used by VFU / VLSU
- **Vector Functional Units [VFU]**
  - Fully pipelined, can start new operation on every cycle
  - Capability for data and control hazards detection
- **Vector Load-Store Unit [VLSU]**

VMIPS Architecture

- **Vector registers**
  - 8 vector registers; each register holds 64-elements
  - 64 bits/vector element **[1 VR=512 Bytes]**
  - Register file has 16 read ports and 8 write ports
- **Scalar registers**
  - 32 GPRs, 32 FPRs : Used by VFU / VLSU
- **Vector Functional Units [VFU]**
  - Fully pipelined, can start new operation on every cycle
  - Capability for data and control hazards detection
- **Vector Load-Store Unit [VLSU]**



Programming with VMIPS Instructions

- ADDVV.D: add two vectors
- ADDVS.D: add vector to a scalar
- LV/SV: vector load and vector store from address

```
L.D      F0,a      ; load scalar a
LV       V1,Rx     ; load vector X
MULVS.D  V2,V1,F0  ; vector-scalar multiply
LV       V3,Ry     ; load vector Y
ADDVV    V4,V2,V3  ; add
SV       Ry,V4     ; store the result
```

- Requires 6 instructions vs. almost 600 for MIPS

Programming with VMIPS Instructions

AX+Y

MIPS

```
        L.D     F0,a
        DADDIU  R4,Rx,#512
oop:    L.D     F2,0(Rx)
        MUL.D   F2,F2,F0
        L.D     F4,0(Ry)
        ADD.D   F4,F4,F2
        S.D     F4,9(Ry)
        DADDIU  Rx,Rx,#8
        DADDIU  Ry,Ry,#8
        DSUBU   R20,R4,Rx
        BNEZ    R20,Loop
```

VMIPS

```
L.D     F0,a
LV      V1,Rx
MULVS.D V2,V1,F0
LV      V3,Ry
ADDVV.D V4,V2,V3
SV      V4,Ry
```

- 600 vs 6 instructions,
- Avoiding loop overhead instructions
- Loop Vectorization (if no dependency between iterations)

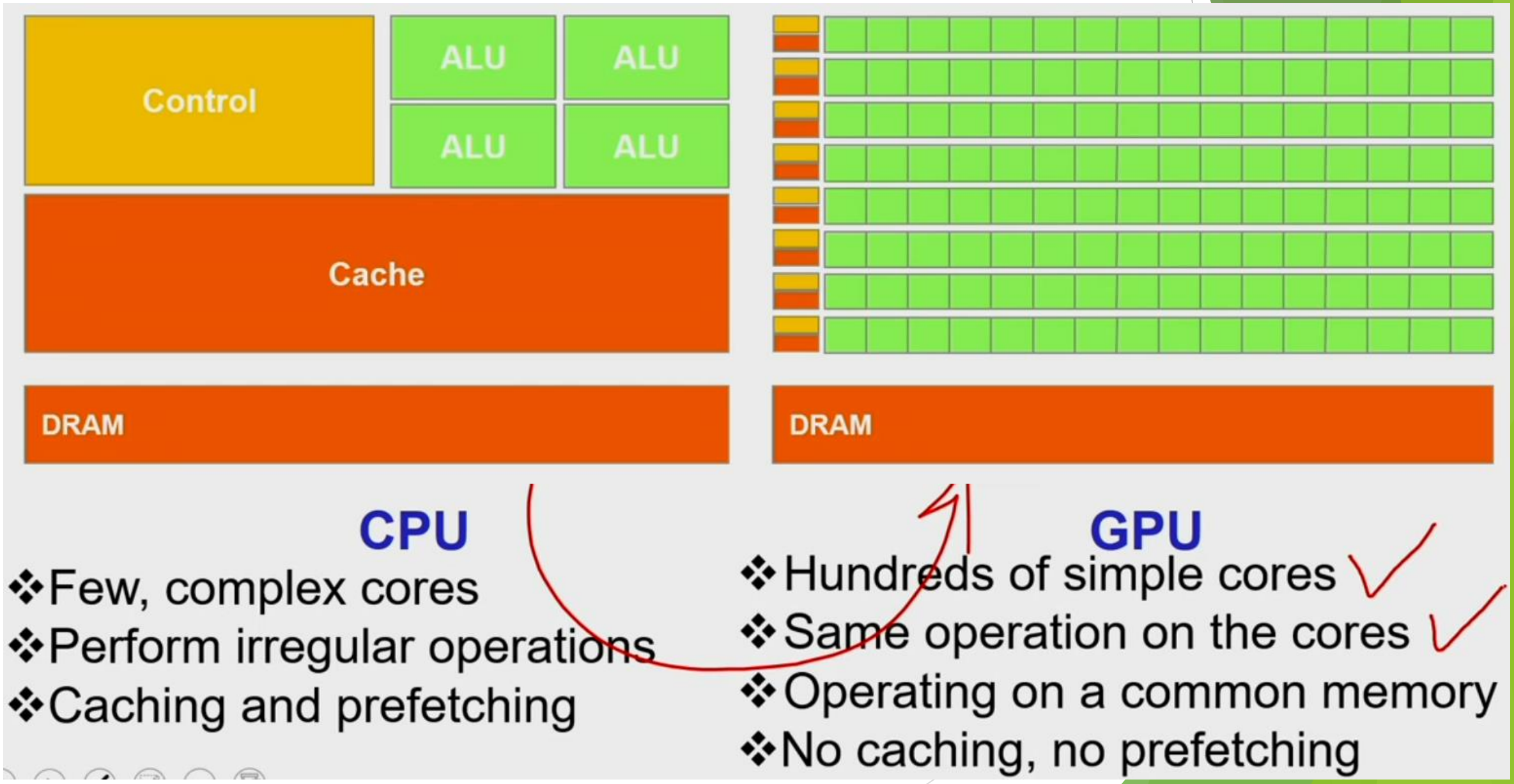
Graphics Processing Unit (GPU)

Graphics Processing Unit (GPU)

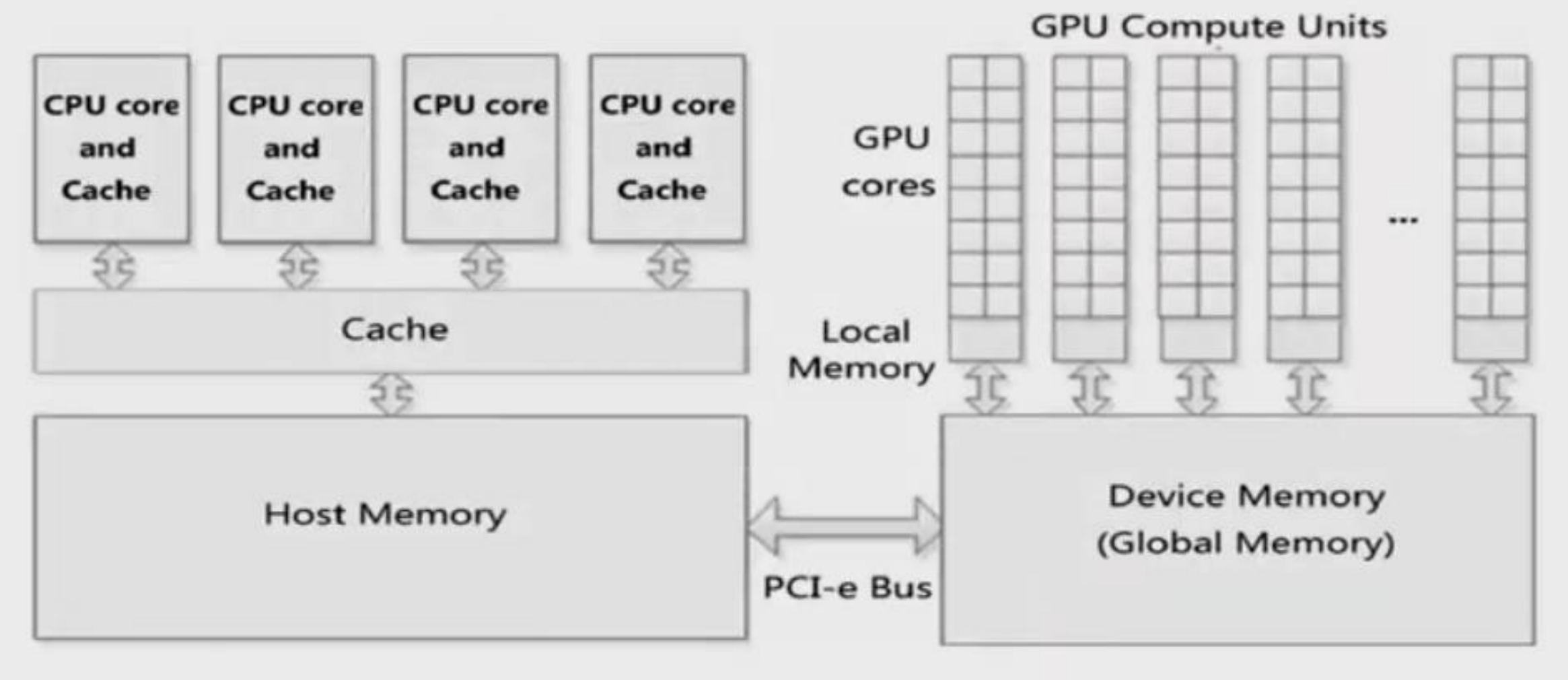
- The GPU is viewed as a compute device that:
  - Is a coprocessor to the CPU or host
  - Has its own DRAM (device memory)
  - Runs many threads in parallel
- Data-parallel portions of an application are executed on the device as kernels which run in parallel on many light weight threads
- GPU gives best performance if the application is:
  - Computation intensive
  - Many independent computations
  - Many similar computations

GPU CPU
device host

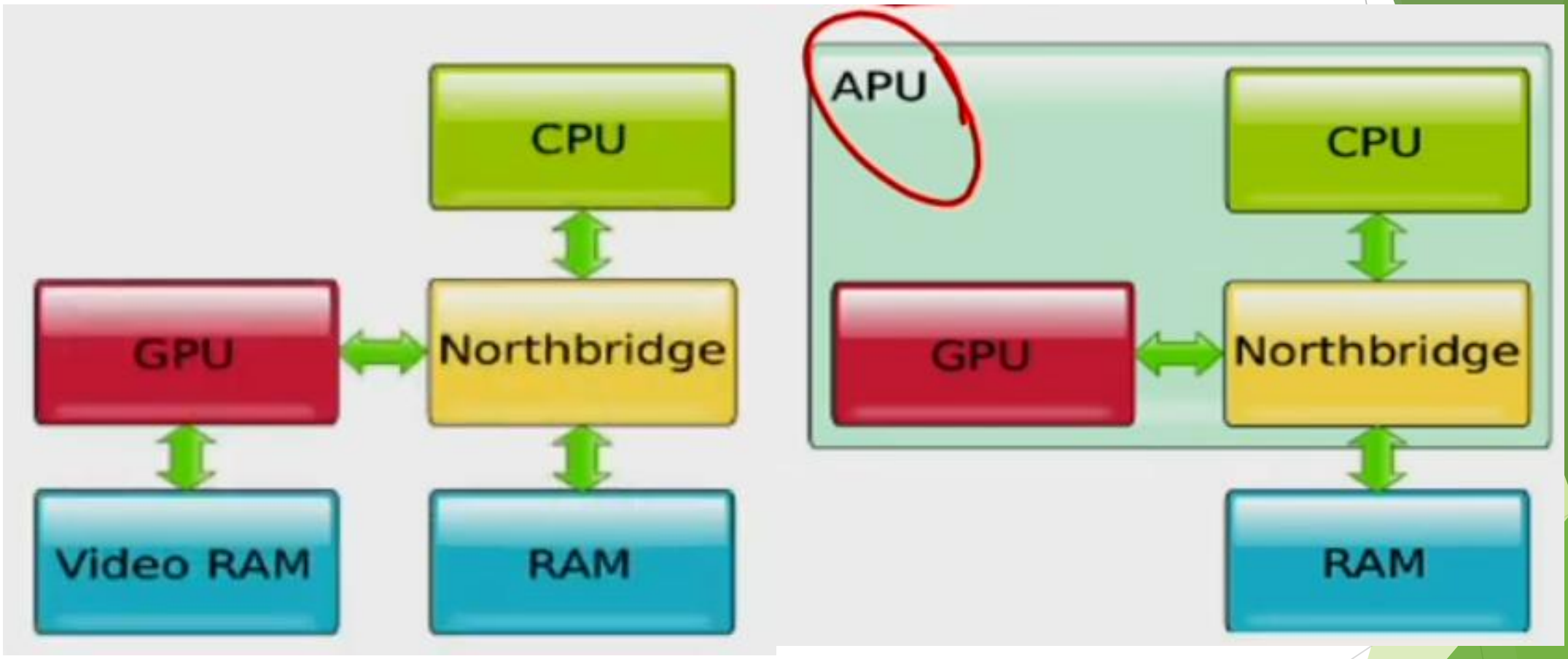
CPU vs GPU


Control
ALU
ALU
ALU
ALU
Cache
DRAM
DRAM
CPU
❖Few, complex cores
❖Perform irregular operations
❖Caching and prefetching
GPU
❖Hundreds of simple cores
❖Same operation on the cores
❖Operating on a common memory
❖No caching, no prefetching

CPU vs GPU

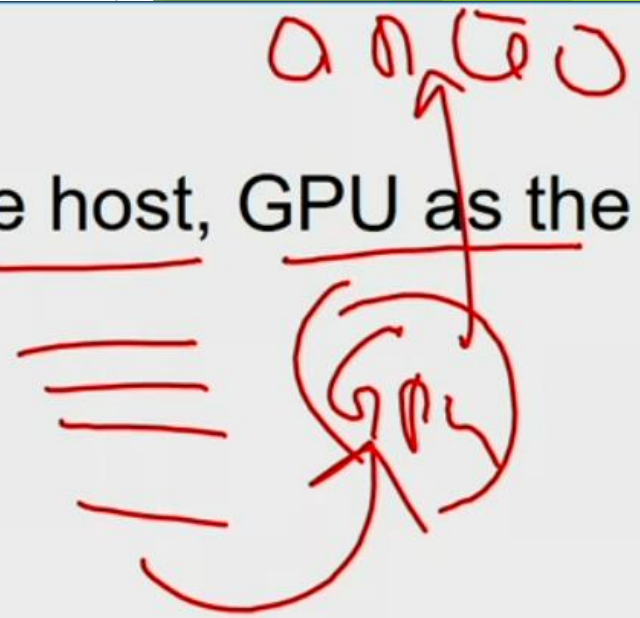

GPU Compute Units
CPU core and Cache
CPU core and Cache
CPU core and Cache
CPU core and Cache
GPU cores
...
Cache
Local Memory
Host Memory
Device Memory (Global Memory)
PCI-e Bus

CPU vs GPU


CPU
GPU
Northbridge
Video RAM
RAM
APU
CPU
GPU
Northbridge
RAM

Key Features of GPU

- Working on SIMD model
- Heterogeneous execution model with CPU as the host, GPU as the device
- CUDA - programming language for GPU
  (CUDA - Compute Unified Device Architecture)
- A thread is associated with each data element
- Threads are organized into blocks, blocks into a grid
- GPU hardware handles thread management.



Key Features of GPU

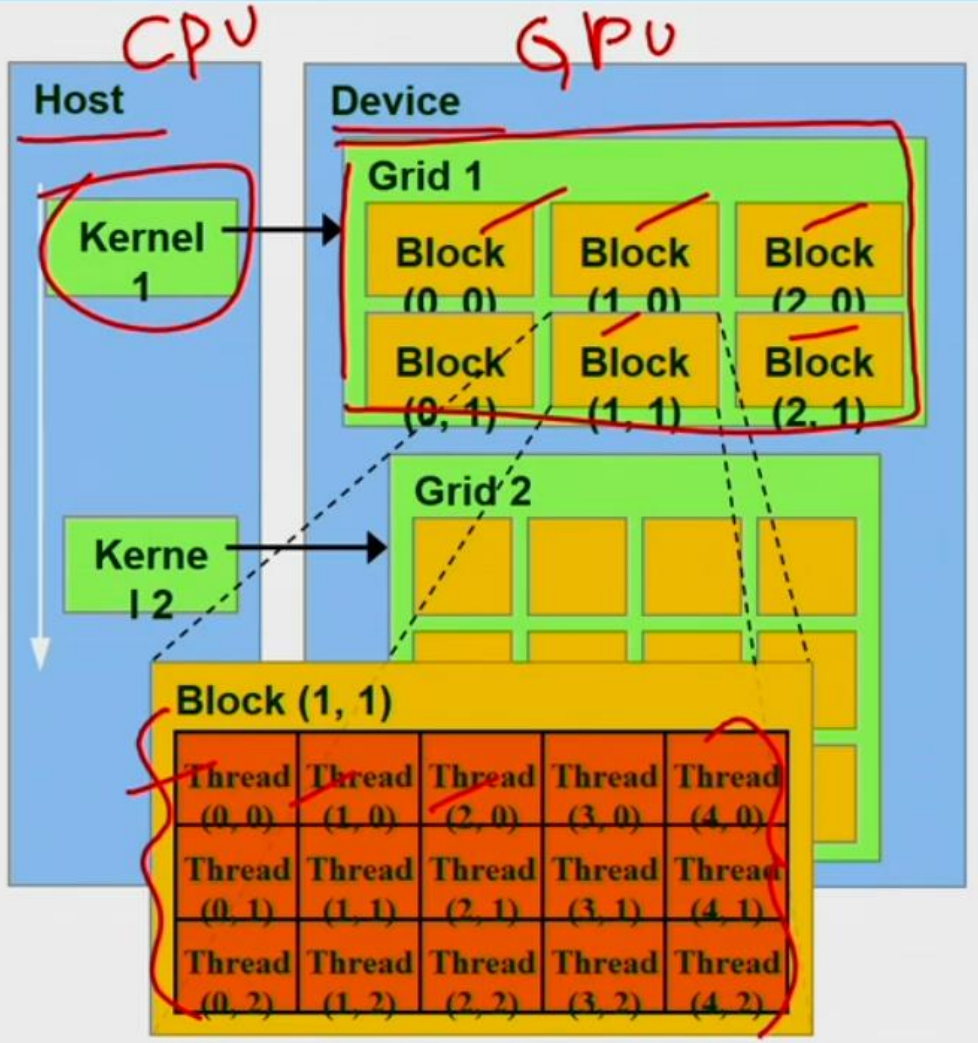
- The GPU is good at
  - data-parallel processing when the same computation is executed on many data elements in parallel. It has low control flow overhead.
  - high floating point arithmetic intensity operations that has many calculations per memory access and high floating point to integer ratio.

Key Features of GPU

- **Similarities to vector machines:**
  - Works well with data-level parallel problems
  - Scatter-gather transfers
  - Mask registers
  - Large register files
- **Differences:**
  - No scalar processor
  - Uses multithreading to hide memory latency
  - Has many functional units, as opposed to a few deeply pipelined units like a vector processor

Key Features of GPU

- A kernel is executed as a grid of thread blocks
- Threads share memory space
- A thread block is a batch of threads that can cooperate with each other by:
  - Synchronizing their execution for hazard-free shared memory accesses
  - Efficiently sharing data through a low latency shared memory



Example

- Multiply two vectors of length 8192
- Grid – entire code
- Grid is divided into blocks
- Grid size = 16 blocks
- Block is assigned to few multithreaded SIMD processor
- 16 multithreaded SIMD processor
- Each SIMD instruction executes 32 elements at a time
- Here 512 threads per block