



# Mestrado em Eng. Informática

2024/25

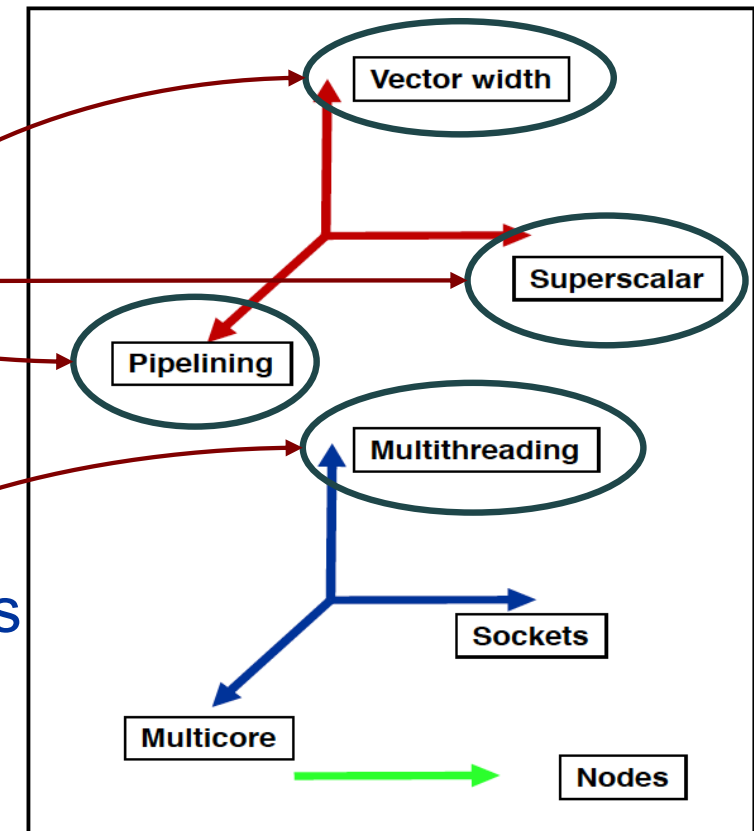
*J.L. Sobral*

## Uniprocessor Multithreading

# Key issues for parallelism in a single-core

- **Currently under discussion:**

- pipelining: reviewed in the combine example
- superscalar: idem, but some more now
- data parallelism: vector computers & vector extensions to scalar processors
- multithreading: alternative approaches

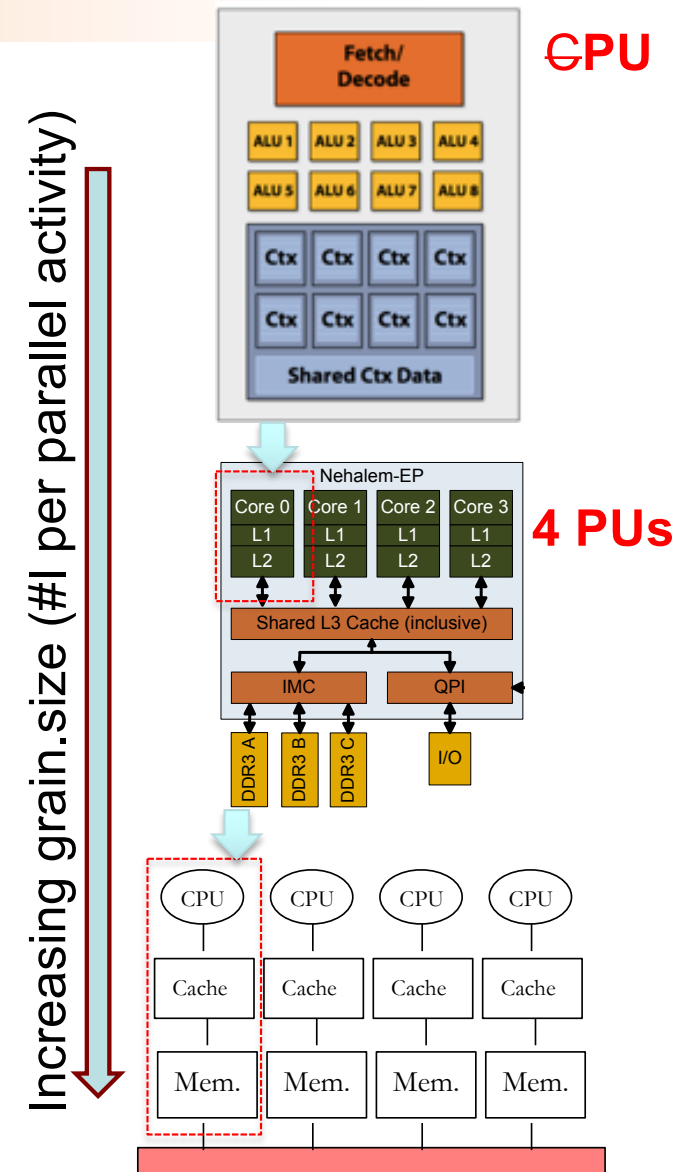


# Parallel Computing



## Parallelism level (Software vs Hardware)

- **Instruction-Level Parallelism (ILP)**
  - Parallel execution of multiple instructions
  - Vector processing (SIMD)
  - Automatic exploration by the hardware
  - Limited by the data and control dependencies
- **Threads [tasks]**
  - Several threads (tasks) executed in parallel
  - Limited by program dependencies (and by the algorithm)
- **Processes**
  - Multiple process from the same program or from several programs

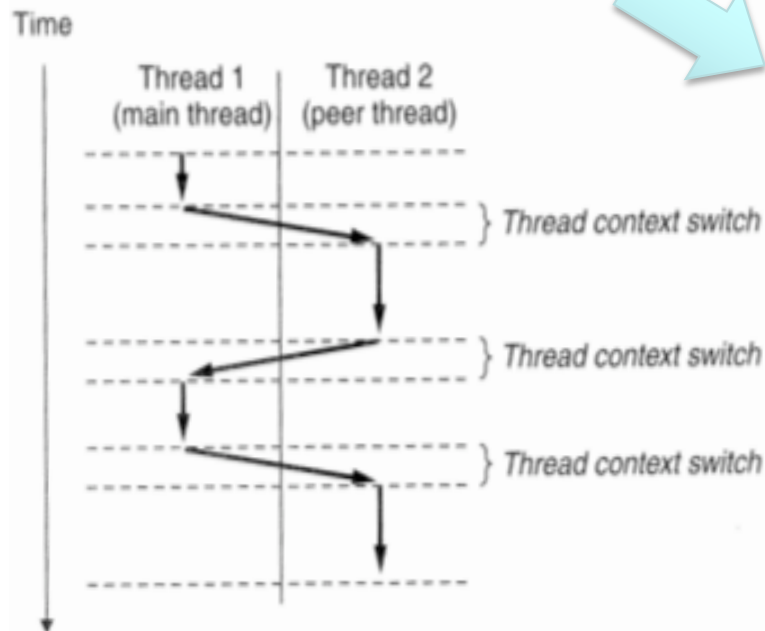


# Logic vs physical parallelism

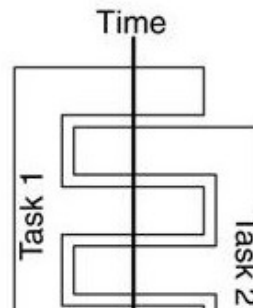


## [Thread] Scheduling

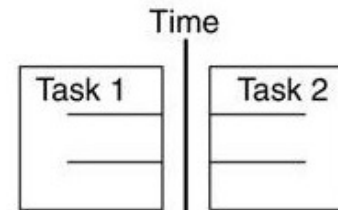
- The PU executes a set of threads [tasks], by scheduling instructions of each thread in-between time-slices (e.g. 50ms)
- It gives the illusion that threads are running in parallel => **logic parallelism**



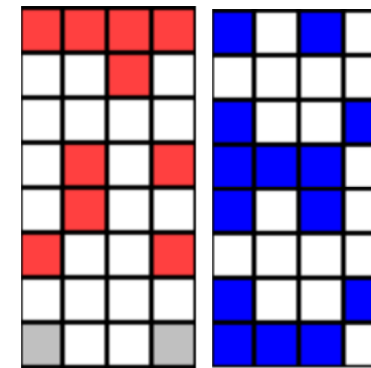
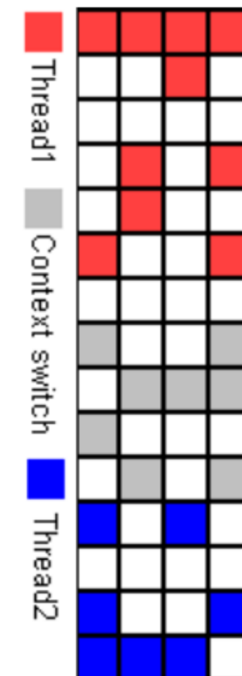
## Concurrency



## Parallelism



2 PUs



## Physical parallelism

- At each clock cycle, instructions from multiple threads are running in parallel (on the same or in different processors)

# Unicore Multithreading

(hardware-based [thread] scheduling)

## Executing multiple threads on a single core (PU)

- Share all resources but replicate registers, PC/IP, etc.
- Fast switching between threads  
(*fast thread context switch*)

### 1. Coarse-grain multithreading

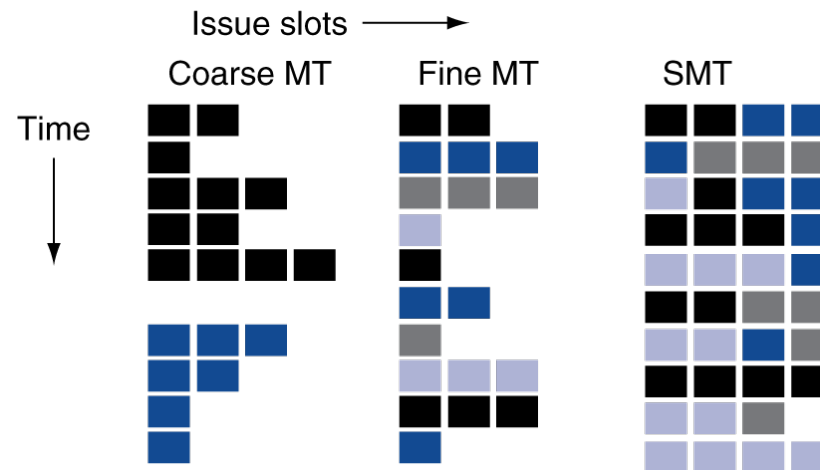
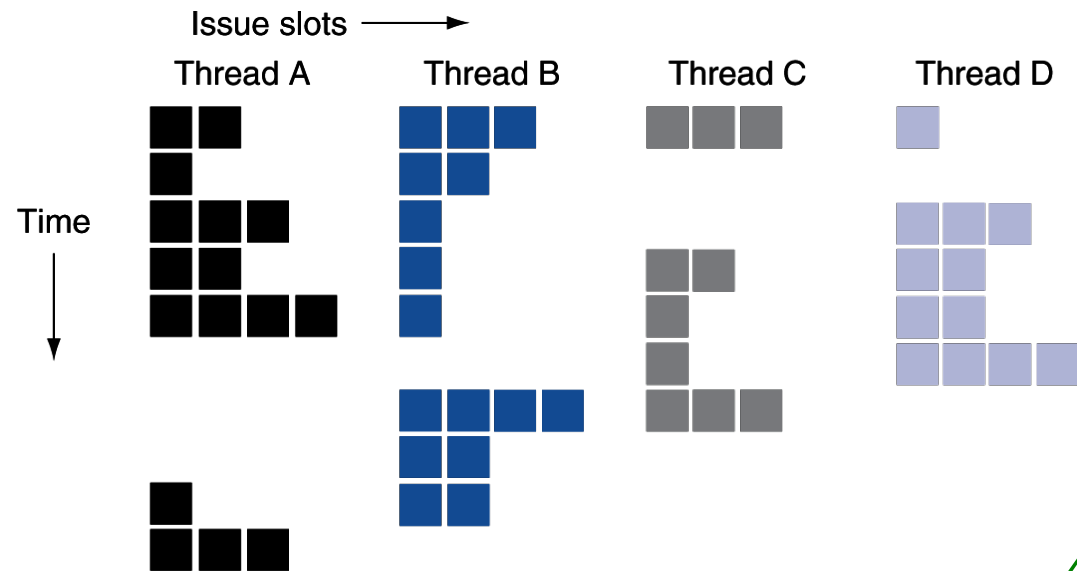
- Only switch on long stall (e.g., L2-cache miss)
- Simplifies hardware, but doesn't hide short stalls (e.g., data hazards)

### 2. Fine-grain multithreading / time-multiplexed MT

- Switch threads after each clock cycle
- Interleave instruction execution
- If one thread stalls, others are executed

### 3. Simultaneous multithreading (*SMT - next slides...*)

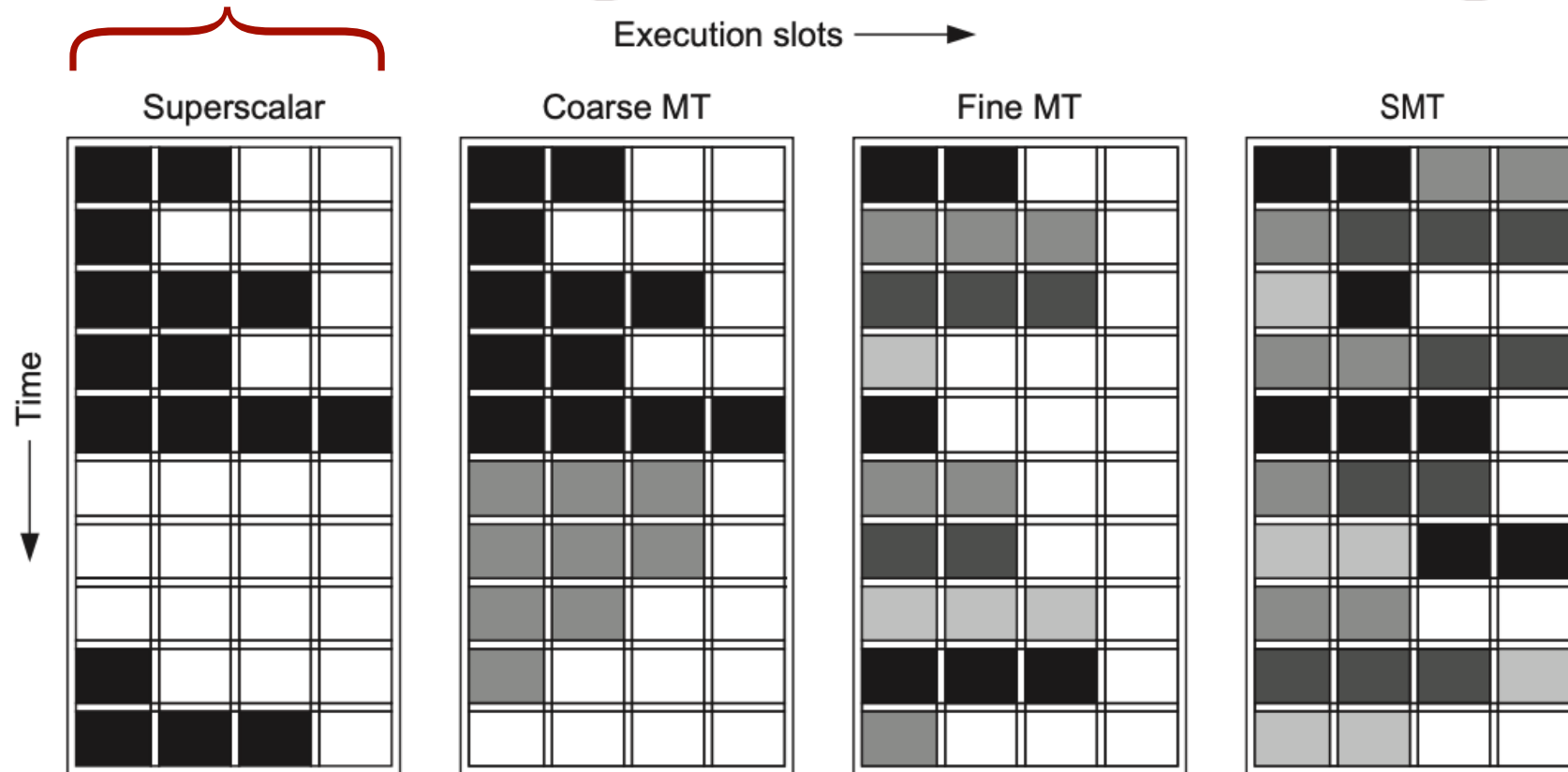
# Multithreading Example



# Sharing superscalar resources

4-way superscalar  
w/out MultiThreading

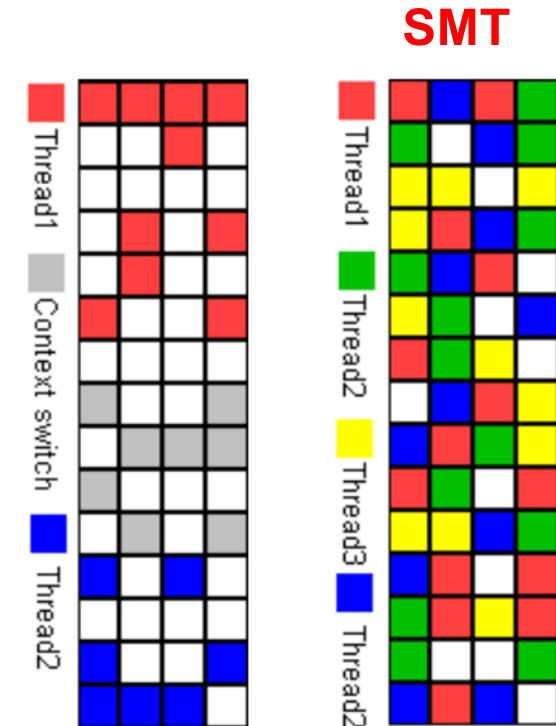
4-way superscalar & MultiThreading



**Figure 3.31** How four different approaches use the functional unit execution slots of a superscalar processor. The

### 3. Simultaneous Multithreading (SMT)

- In multiple-issue dynamically scheduled processor
  - Schedule instructions from multiple threads
  - Instructions from independent threads execute when function units are available
  - Within threads, dependencies handled by scheduling and register renaming
- Example: Intel from Pentium-4 HT
  - Two threads: duplicated registers, shared function units and caches



*HT: Hyper-Threading, Intel trade mark for their SMT implementation*



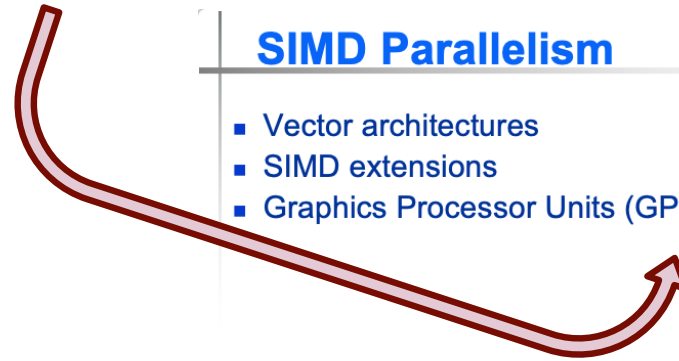
# Graphical Processing Units

## Hardware view

(programming will be addressed later)

### SIMD Parallelism


- Vector architectures
- SIMD extensions
- Graphics Processor Units (GPUs) *(in another set of slides)*



## ■ Basic idea:

- Heterogeneous execution model
  - CPU is the *host*, GPU is the *device*
- Develop a C-like programming language for GPU
- Unify all forms of GPU parallelism as *CUDA threads*
- Programming model follows **SIMT**:  
“*Single Instruction Multiple Threads*”

# [NVIDIA] GPU Architecture

- Similarities to **vector machines**:
  - Works well with data-level parallel problems
  - Scatter-gather transfers
  - Mask registers
  - **Large register files**  **Hardware-based thread scheduling**
    - Registers are assigned to threads
    - constrains the maximum number of threads**
    - Fine-grain multithreading
- Differences:
  - No scalar processor
  - **Uses multithreading to hide [memory] latency**
  - Has many functional units, as opposed to a few deeply pipelined units, **behaving as a vector processor**

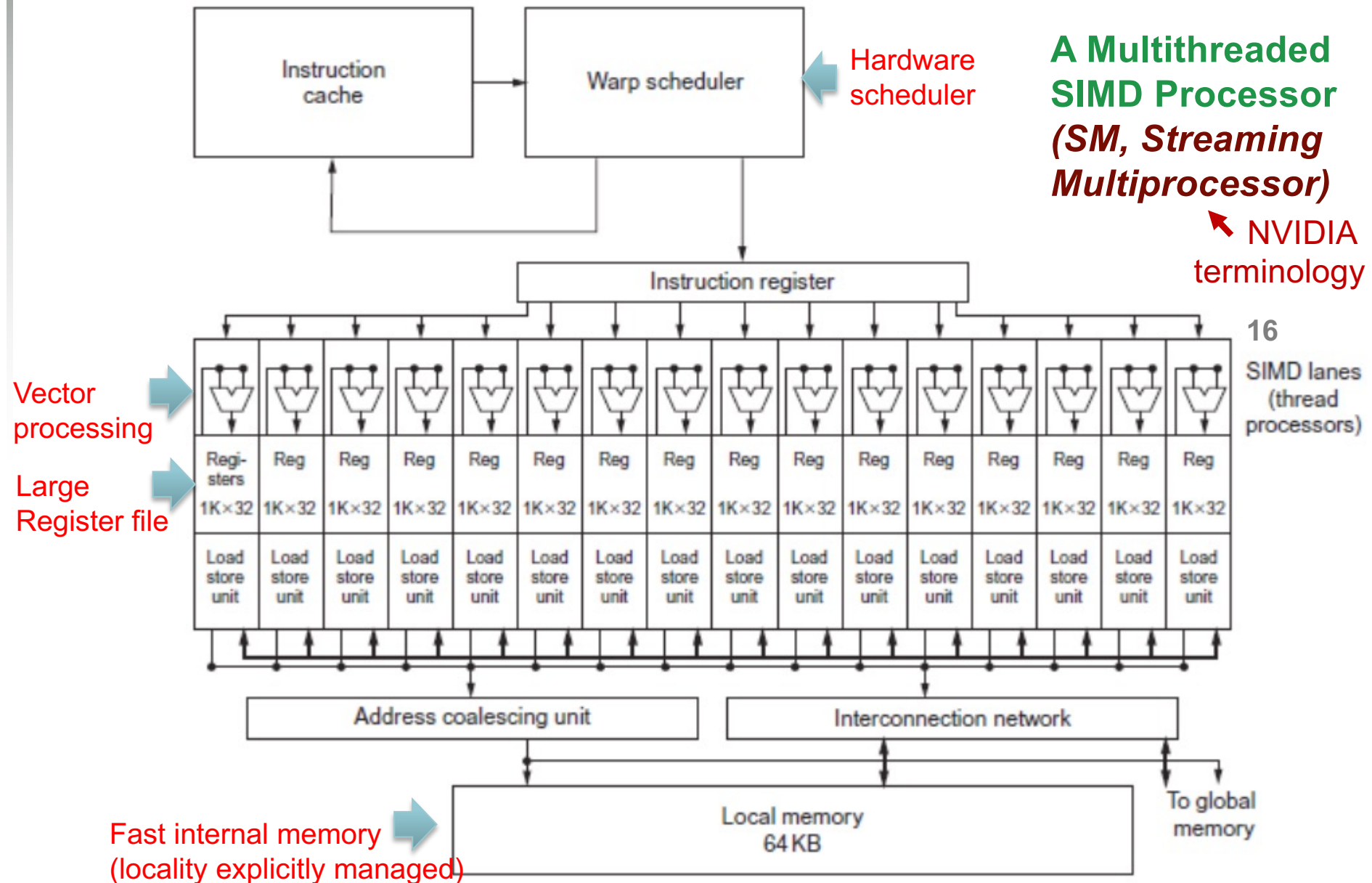
# GK110 Kepler Architecture

**Kepler:**  
15 SMX  
2880 CUDA-cores  
*October'13*

Each **SMX** is  
like a PU on a  
multicore  
processor  
**but SMXs**  
**internally rely**  
**on vector**  
**processing**  
(see next slide)



# GPU Organization (NVIDIA)





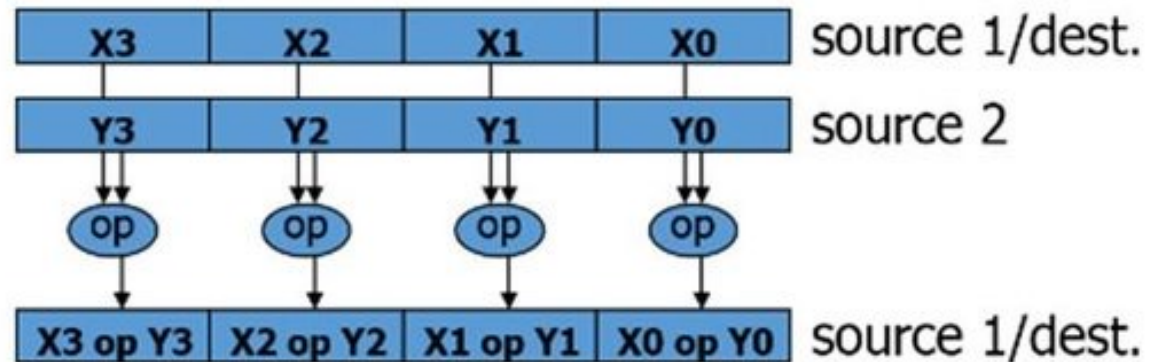
# SIMD Data Parallelism: CPU vs. GPU

~~CPU~~

SIMD

1 instruction – multiple data

SSE2/3/4 – Neon – AltiVec  
AVX – AVX2...



GPU

SIMT

1 instruction – multiple threads

