Parallel Computing

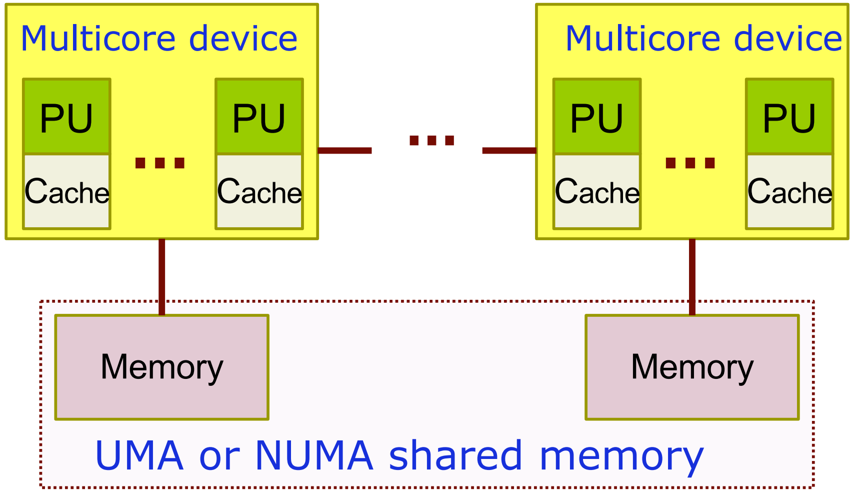
João Luís Ferreira Sobral
www.di.uminho.pt/~jls
jls@...

Web: Elearning

Explicit parallel computing (1)

Shared Memory (SM) parallel systems

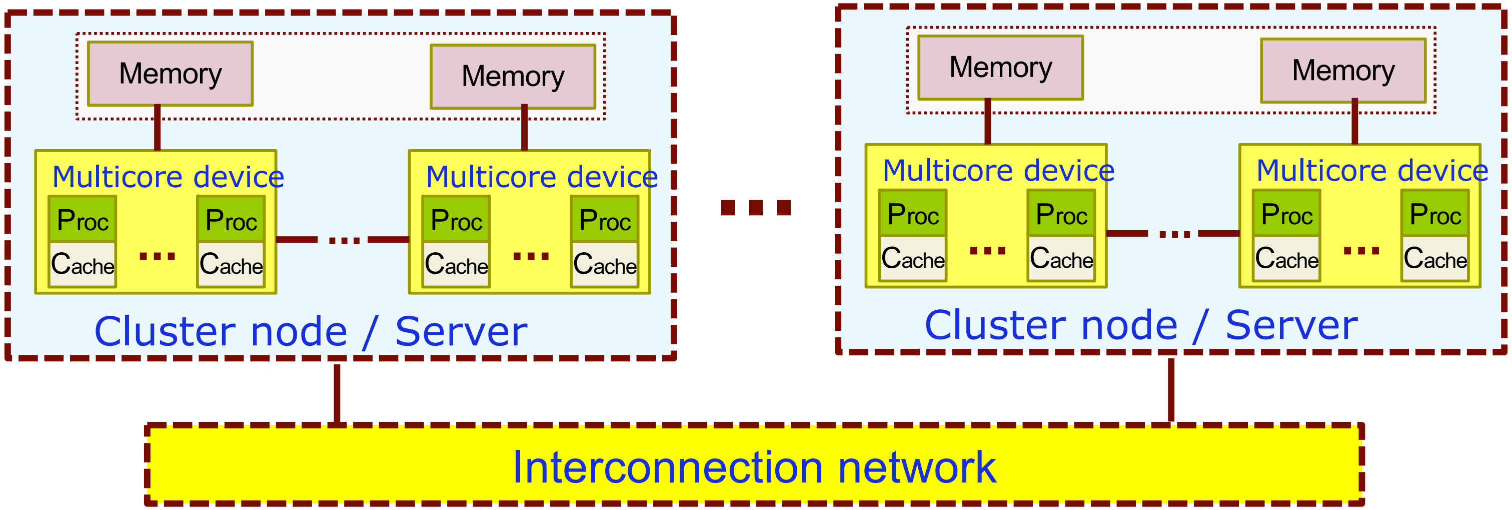
- parallelism on single or multiple devices *(same motherboard)*
  - **single physical memory address space**
  - each core (PU) can support multiple threads (SMT)
  - **memory bandwidth is shared by all cores**
  - **coherence of data in multiple caches?**



Explicit parallel computing (2)

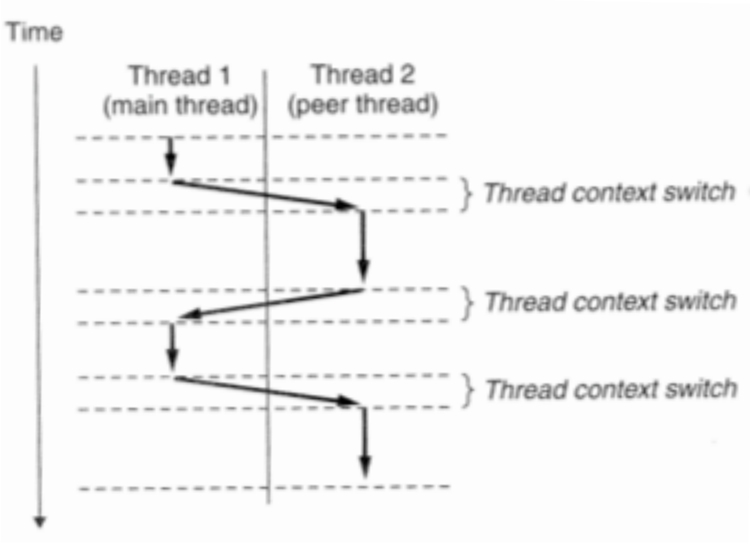
Distributed memory parallel systems

- on multiple boards *(or multiple nodes/servers)*
  - each node with its private memory space
    - **memory bandwidth is proportional to the number of nodes**

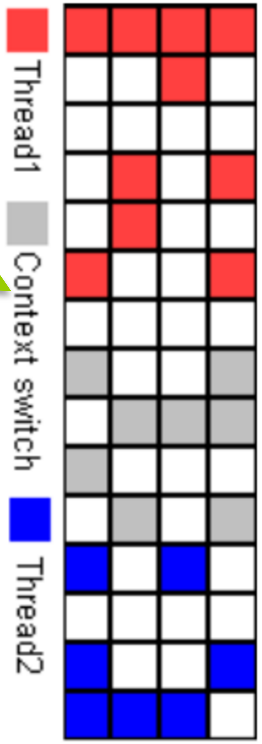


Thread Scheduling
(on shared memory parallel systems)

Review: OS thread scheduling and context switch



Superscalar PU (4-way)



Thread scheduling at the Operating System (OS) level:

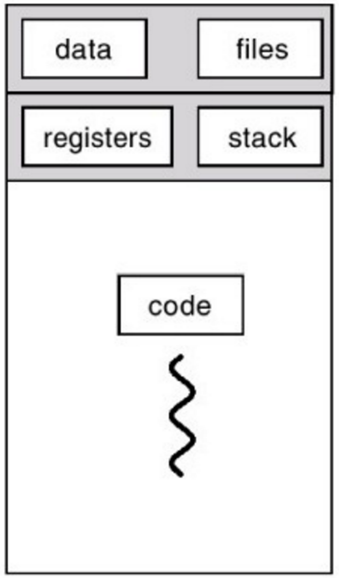
1. Interrupts execution of Thread1 & saves Thread1 context information (e.g., PU registers)
2. Selects the next thread to execute (Thread2)
3. OS restores the state of Thread2 and gives control to it
4. Thread2 continues execution from its previous execution point (saved Program Counter register)

Programming model: Process vs Threads

Processes

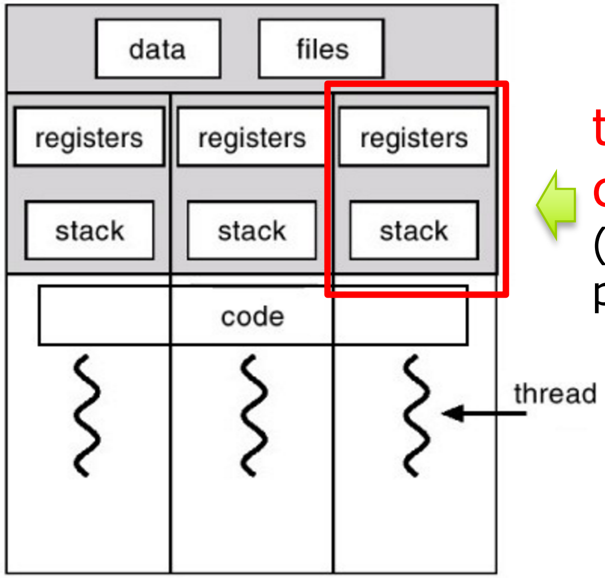
- Used for unrelated tasks
  - (e.g., a program)
- Own address space
  - Address space is protected from other process
- Switching at the kernel (OS) level

Every process has at lest one thread



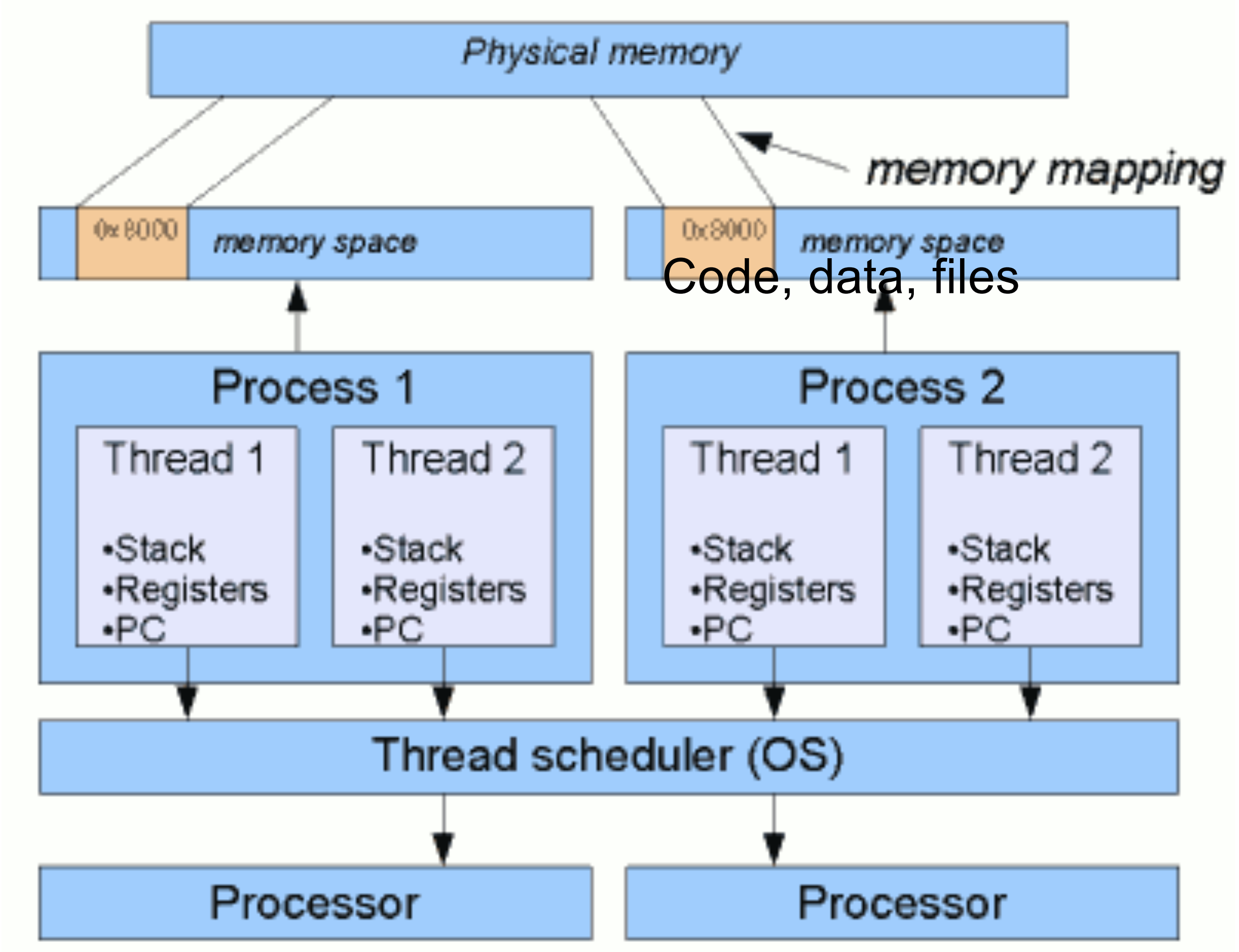
Threads

- Are part from the same job
- Share address space, code, data and files
- Switching at the user or kernel level



thread context
(one stack per thread)

Thread vs Process

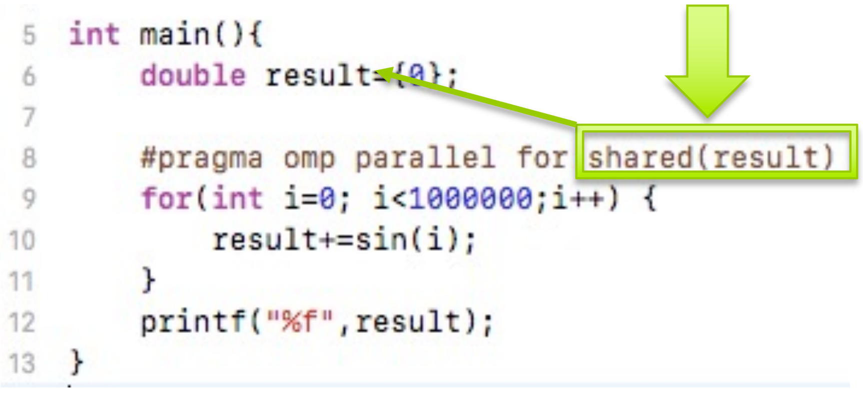


process memory space is shared by all threads of a process

Threads from all processes are scheduled into available PUs

Data races

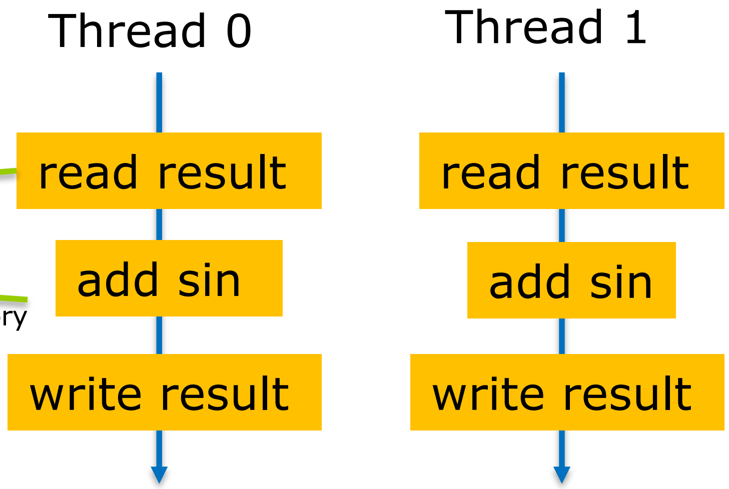
- A data race **can happen** when two or more threads access (write!) to a shared memory position



```
int main(){
    double result={0};

    #pragma omp parallel for shared(result)
    for(int i=0; i<1000000;i++) {
        result+=sin(i);
    }
    printf("%f",result);
}
```

```
.L4:
    ...
    call sin        ; after sin call, xmm0 holds the sin value
    vmovsd  ~~8(%rsp~~%r12), %xmm1        ; read result into %xmm1
    vaddsd  %xmm0, %xmm1, %xmm1   ; add sin value to %xmm1
    vmovsd  %xmm1, ~~8(%rsp~~%r12)        ; write new result back to memory
    ...
    cmpl    %ebx, %ebp                    ; i<1000000?
    jne .L4
```

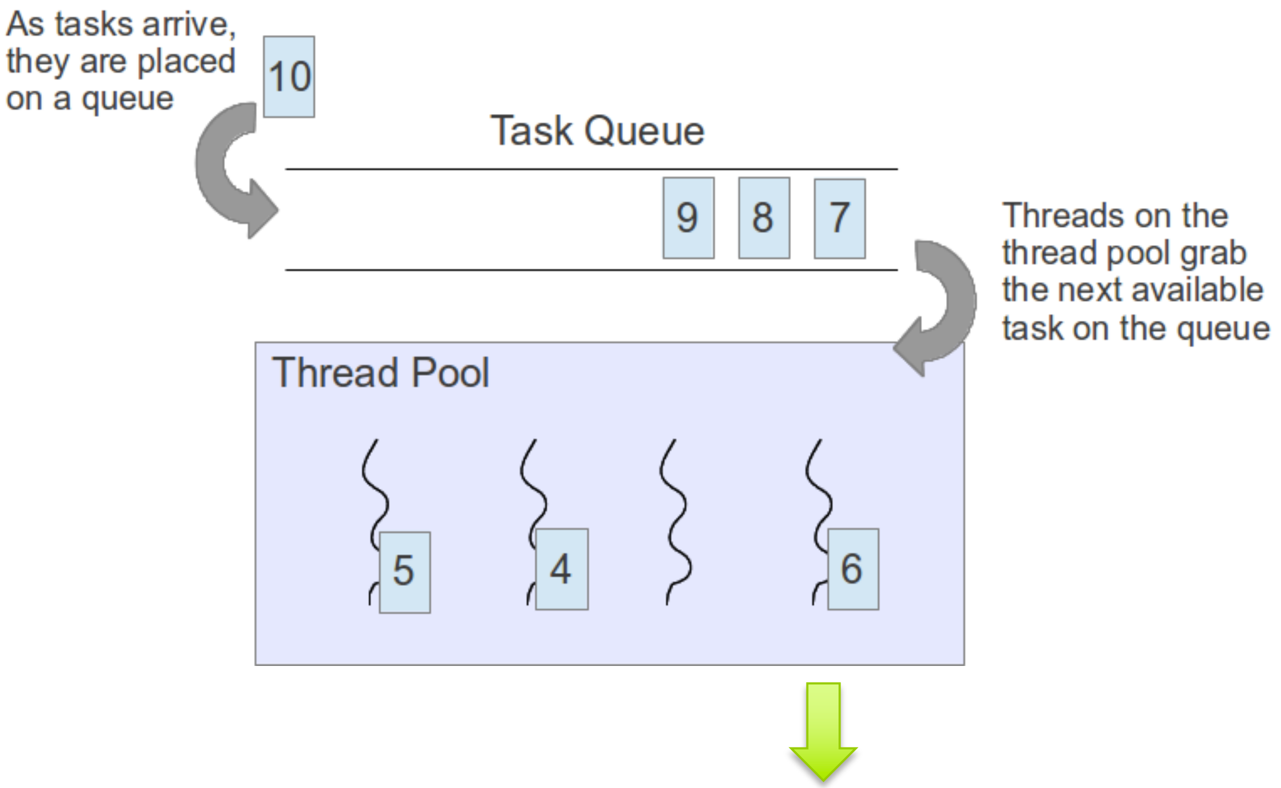


Process/Thread vs Tasks

- **Task**: sequence of instructions
- **Thread/process**: execution context for a task
- **Processor/core**: hardware that runs a thread/process

In Java

- Runnable object
- Thread
- Processor core


As tasks arrive, they are placed on a queue
10
Task Queue
9
8
7
Threads on the thread pool grab the next available task on the queue
Thread Pool
5
4
6

Threads are scheduled on available cores