High Performance Programming
SIMD Intro – MMX/SSE/AVX
FISE2-INFO2

Guillaume MULLER

April 12, 2021

Why having HPP lectures in FISE2? Why HPP?

Why having HPP lectures in FISE2? Why HPP?

Classical approach to HPP = distribute on cluster

- What if the initial code is inefficient?
- $\Rightarrow$ Optimize locally / Think parallel

Why having HPP lectures in FISE2? Why HPP?

Classical approach to HPP = distribute on cluster

- What if the initial code is inefficient?
- $\Rightarrow$ Optimize locally / Think parallel

Current machines already are massively parallel

- Multi-Core, Multi-Thread...

Why having HPP lectures in FISE2? Why HPP?

Classical approach to HPP = distribute on cluster

- What if the initial code is inefficient?
- $\Rightarrow$ Optimize locally / Think parallel

Current machines already are massively parallel

- Multi-Core, Multi-Thread...

A large part of optimizations can not be treated automatically

- $\Rightarrow$ impossible to rely on tools written by others
- $\Rightarrow$ as (future) engineers in CS: mandatory knowledge

Why SIMD/MMX/SSE/AVX?

Task Parallelism

- Execute coarse-grained pieces of code on $\neq$ pieces of hardware
- Multi-Cores, Multi-Threading...

Why SIMD/MMX/SSE/AVX?

Task Parallelism

- Execute coarse-grained pieces of code on $\neq$ pieces of hardware
- Multi-Cores, Multi-Threading...

Instruction/Data Parallelism

- Tiny pieces of code/data simultaneously on $=$ hardware
- SIMD

Why SIMD/MMX/SSE/AVX?

Task Parallelism

- Execute coarse-grained pieces of code on $\neq$ pieces of hardware
- Multi-Cores, Multi-Threading...

Instruction/Data Parallelism

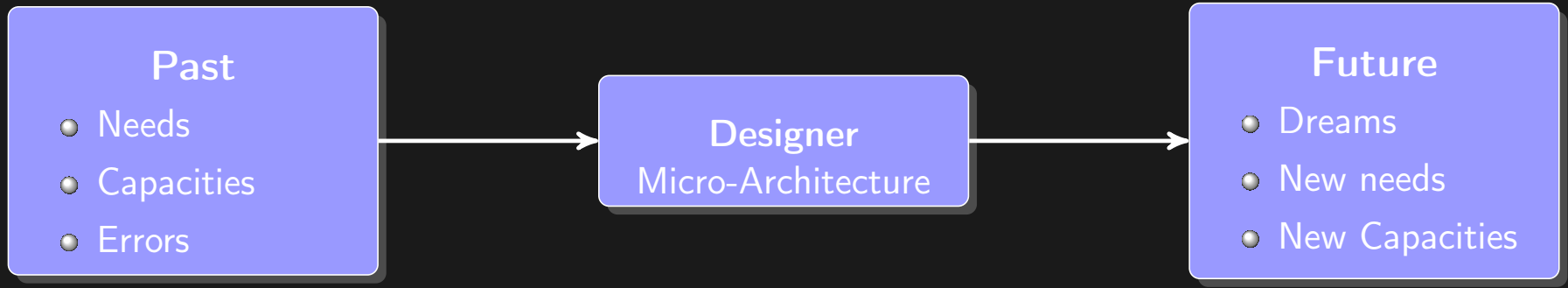
- Tiny pieces of code/data simultaneously on $=$ hardware
- SIMD

- *Who has already used/programmed a processor $\neq$ Intel?*

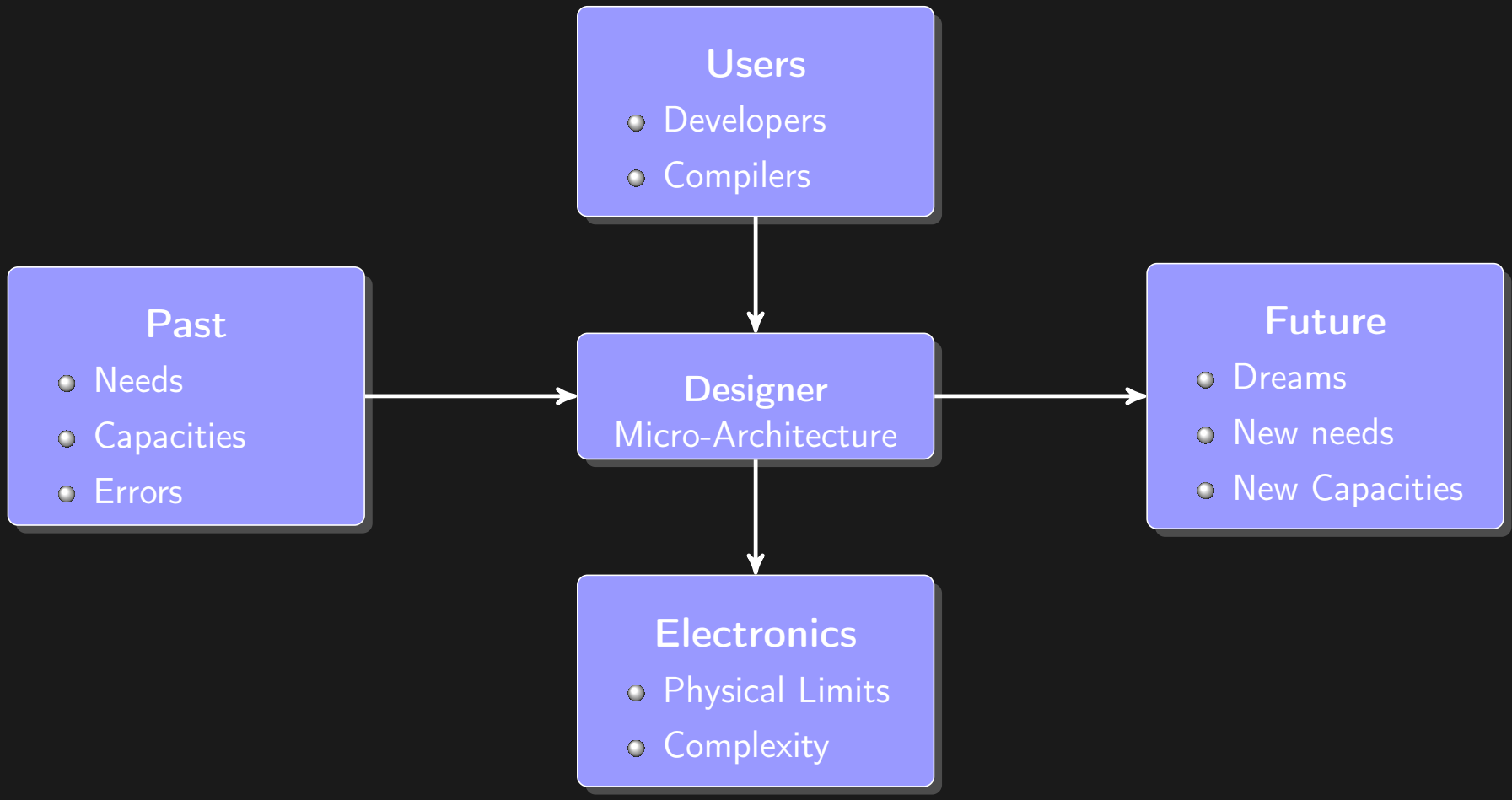
"Ça dépend" ("ça dépasse"?)

Designer
Micro-Architecture

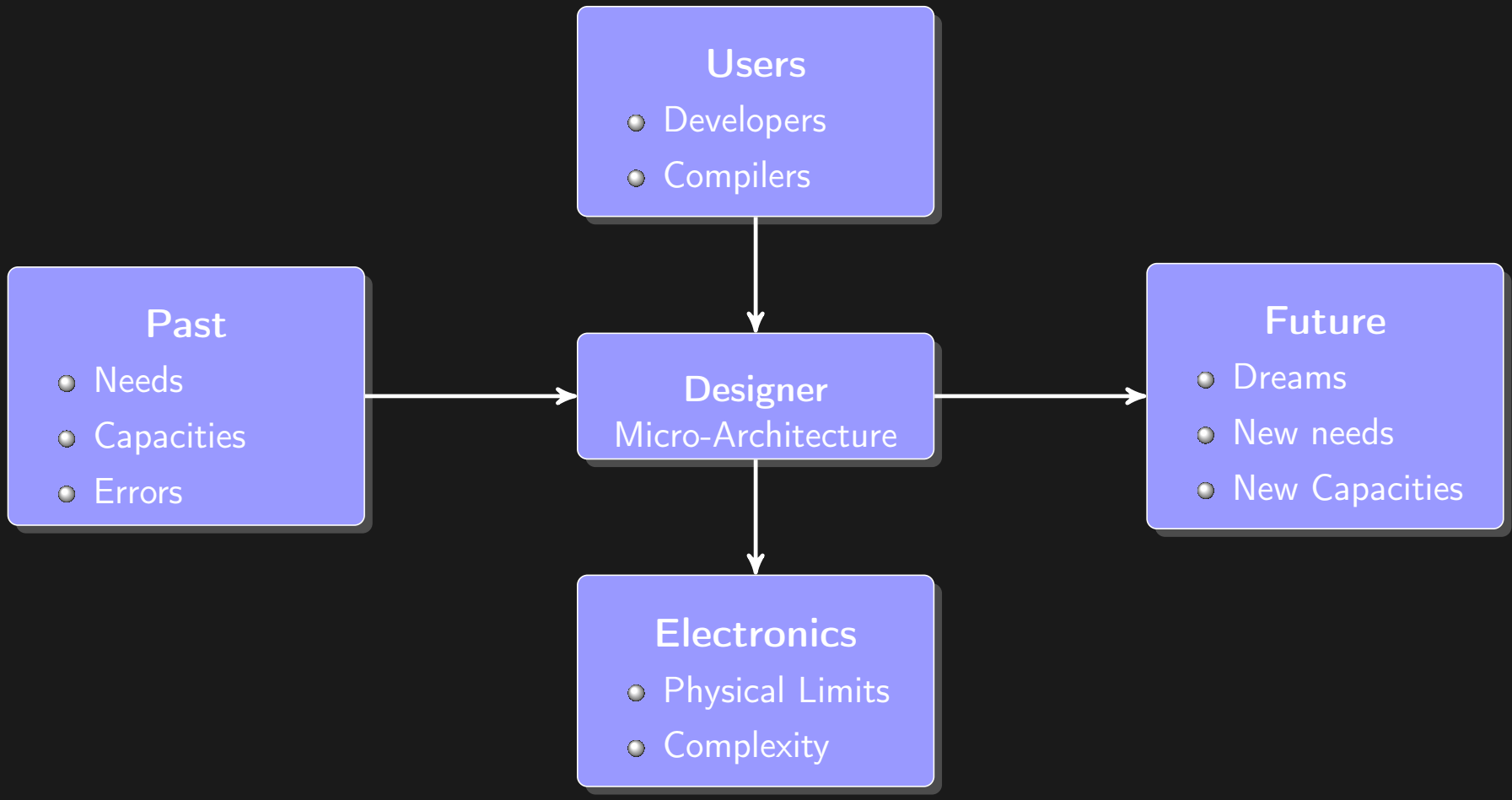
"Ça dépend" ("ça dépasse"?)



"Ça dépend" ("ça dépasse"?)



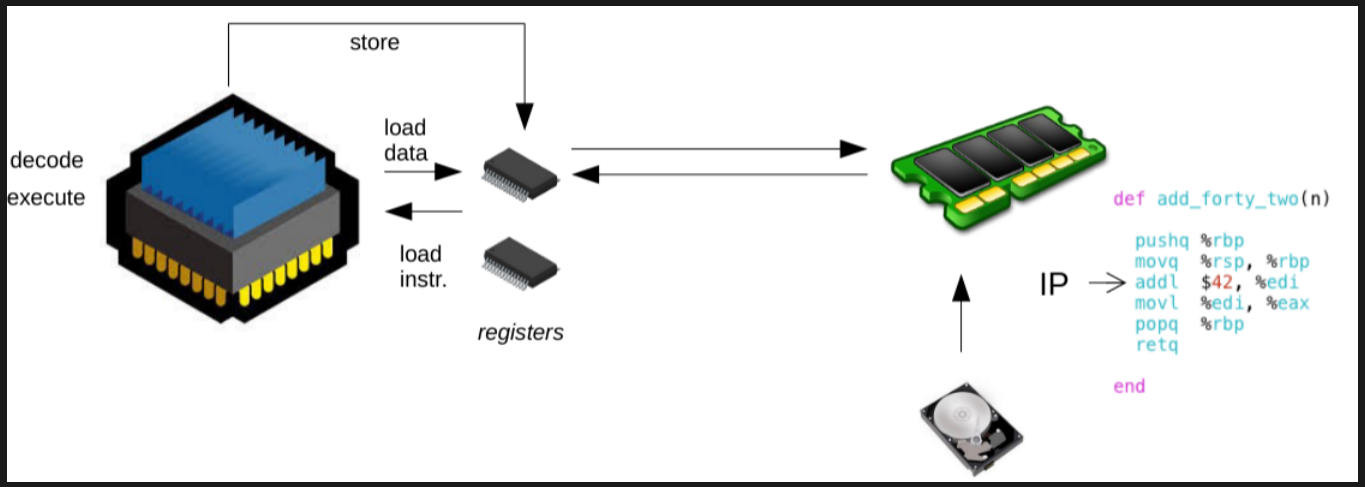
"Ça dépend" ("ça dépasse"?)



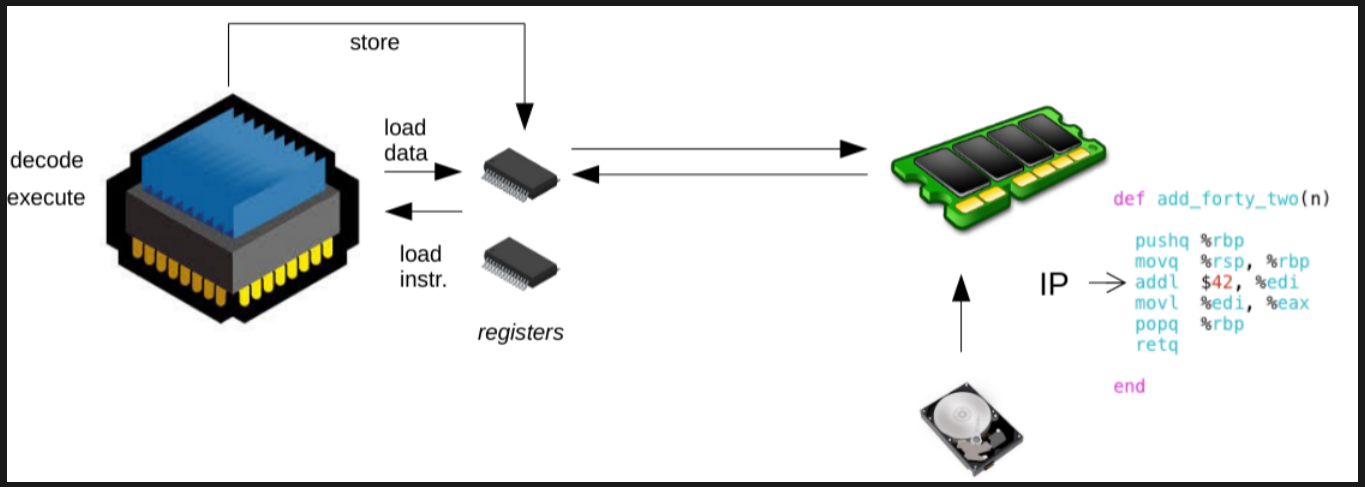
Typical answer in this course

"ça dépend"

"Reminder"


store
decode
execute
load
data
load
instr.
registers
IP
def add_forty_two(n)
pushq %rbp
movq %rsp, %rbp
addl $42, %edi
movl %edi, %eax
popq %rbp
retq
end

"Reminder"



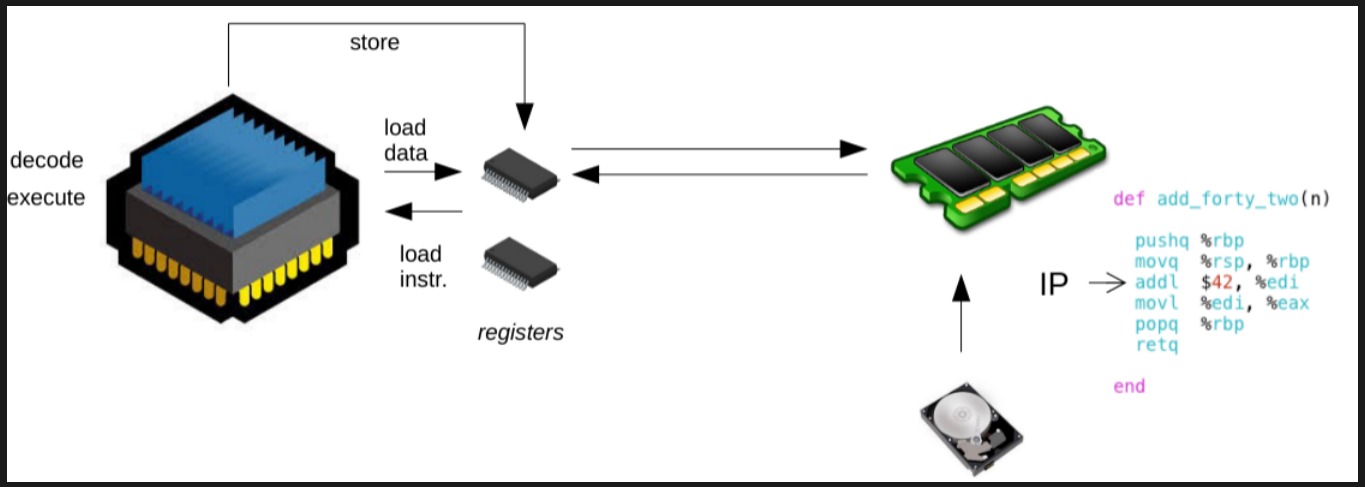
Instruction Types

Flow `IP / JMP ...`

Mem `LD / ST ...`

Calc `ADD / SUB / MULT / DIV ...`

"Reminder"



Instruction Types

Flow `IP / JMP ...`

Mem `LD / ST ...`

Calc `ADD / SUB / MULT / DIV ...`

Instruction Cycle

- Fetch
- Decode
- Execute
- ...