Multithreaded Architectures and Programming on Multithreaded Architectures

Hung-Wei Tseng

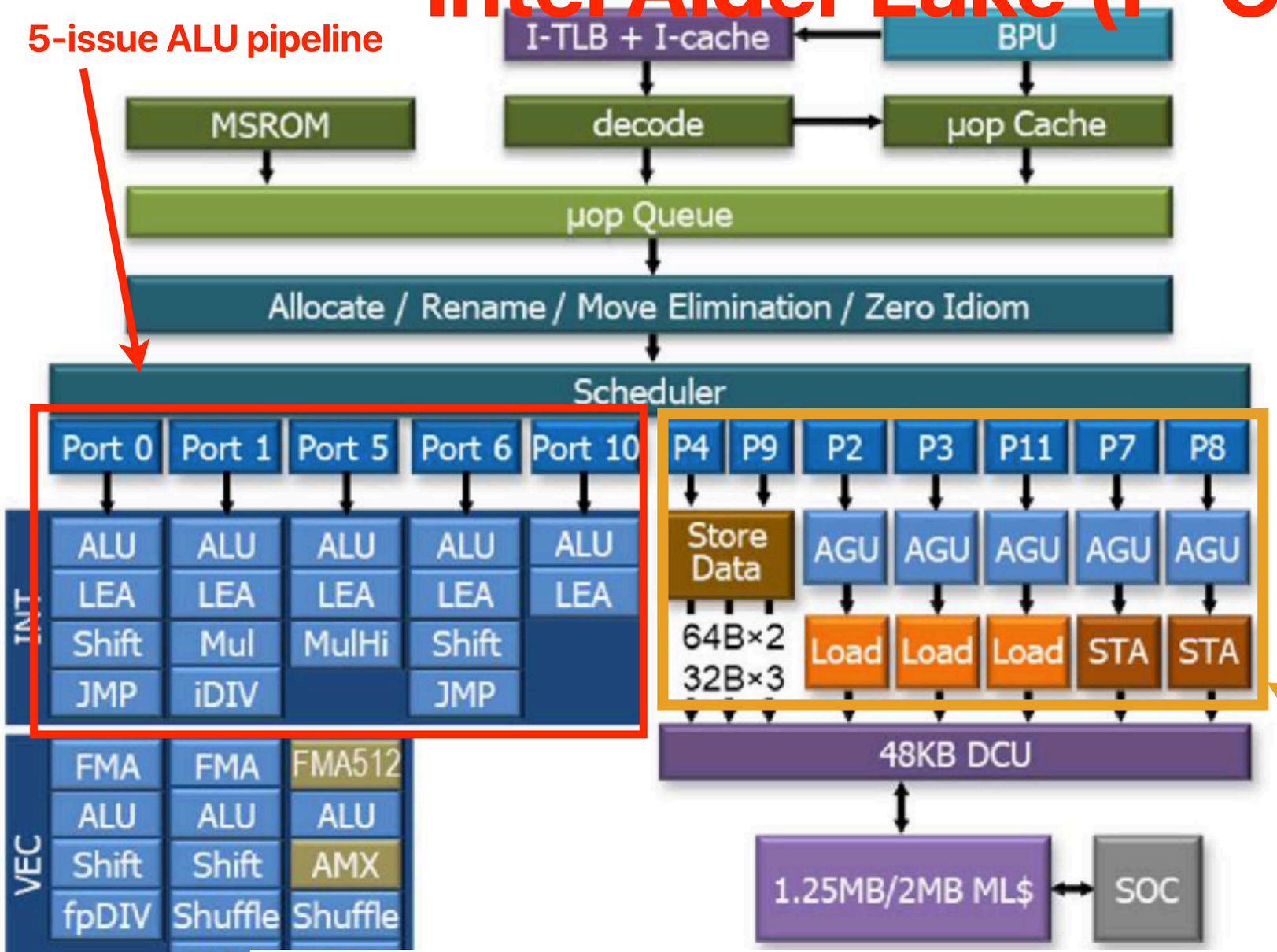
Intel Alder Lake (P-Core)

$$MinCPI = \frac{1}{12}$$

$$MinINTInst.CPI = \frac{1}{5}$$

$$MinMEMInst.CPI = \frac{1}{7}$$

$$MinBRInst.CPI = \frac{1}{2}$$



https://download.intel.com/newsroom/2021/client-computing/intel-architecture-day-2021-presentation.pdf

Summary: Characteristics of modern processor architectures

- Multiple-issue pipelines with multiple functional units available
  - Multiple ALUs
  - Multiple Load/store units
  - Dynamic OoO scheduling to reorder instructions whenever possible
- Cache — very high hit rate if your code has good locality
  - Very matured data/instruction prefetcher
- Branch predictors — very high accuracy if your code is predictable
  - Perceptron
  - Variable history predictors

Recap: popcount

- Which of the following implementations will perform the b
processors?

A

```
inline int popcount(uint64_t x){
  int c=0;
  while(x)  {
        c += x & 1;
        x = x >> 1;
    }
    return c;
}
```

B

```
inline int popcount(uint64_t x) {
   int c = 0;
   while(x)     {
     c += x & 1;
     x = x >> 1;
     c += x & 1;
     x = x >> 1;
     c += x & 1;
     x = x >> 1;
     c += x & 1;
     x = x >> 1;
   }
   return c;
}
```

C

```
inline int popcount(uint64_t x) {
     int c = 0;
     int table[16] = {0, 1, 1, 2, 1,
2, 2, 3, 1, 2, 2, 3, 2, 3, 3, 4};
     while(x)     {
         c += table[(x & 0xF)];
         x = x >> 4;
     }
     return c;
}
```

D

```
inline int popcount(uint64_t x) {
     int c = 0;
     int table[16] = {0, 1, 1, 2, 1,
2, 2, 3, 1, 2, 2, 3, 2, 3, 3, 4};
     for (uint64_t i = 0; i < 16; i++)
     {
         c += table[(x & 0xF)];
         x = x >> 4;
     }
     return c;
 }
```

E

```
inline int popcount(uint64_t x) {
     int c = 0;
     for (uint64_t i = 0; i < 16; i++)
     {
         switch((x & 0xF))
         {
             case 1: c+=1; break;
             case 2: c+=1; break;
             case 3: c+=2; break;
             case 4: c+=1; break;
             case 5: c+=2; break;
             case 6: c+=2; break;
             case 7: c+=3; break;
             case 8: c+=1; break;
             case 9: c+=2; break;
             case 10: c+=2; break;
             case 11: c+=3; break;
             case 12: c+=2; break;
             case 13: c+=3; break;
             case 14: c+=3; break;
             case 15: c+=4; break;
             default: break;
         }
         x = x >> 4;
     }
     return c;
 }
```

F

```
#include <smmintrin.h>
inline int popcount(uint64_t x) {
     int c = _mm_popcnt_u64(x);
     return c;
}
```

Recap: Tips of programming on modern processors

- Minimize the critical path operations
  - Don’t forget about optimizing cache/memory locality first!
    - Memory latencies are still way longer than any arithmetic instruction
    - Can we use arrays/hash tables instead of lists?
  - Branch can be expensive as pipeline get deeper
    - Sorting
    - Loop unrolling
  - Still need to carefully avoid long latency operations (e.g., mod)
- Since processors have multiple functional units — code must be able to exploit instruction-level parallelism
  - Hide as many instructions as possible under the “critical path”
  - Try to use as many different functional units simultaneously as possible
- Modern processors also have accelerated instructions
- Compiler can do fairly go optimizations, but with limitations

Recap: Summary of popcounts

	ET	IC	IPC/ILP	# of branches	Branch mis-prediction rate
A	22.21	332 Trillions	2.88	65 Trillions	1.13%
B	12.29	287 Trillions	4.52	17 Trillions	0.04%
C	5.01	102 Trillions	3.95	17 Trillions	0.04%
D	3.73	80 Trillions	4.13	1 Trillions	~0%
E	54.4	173 Trillions	0.61	44 Trillions	18.6%
SSE4.2	1.57	22 Trillions	2.7	1 Trillions	~0%

Best at 4.5

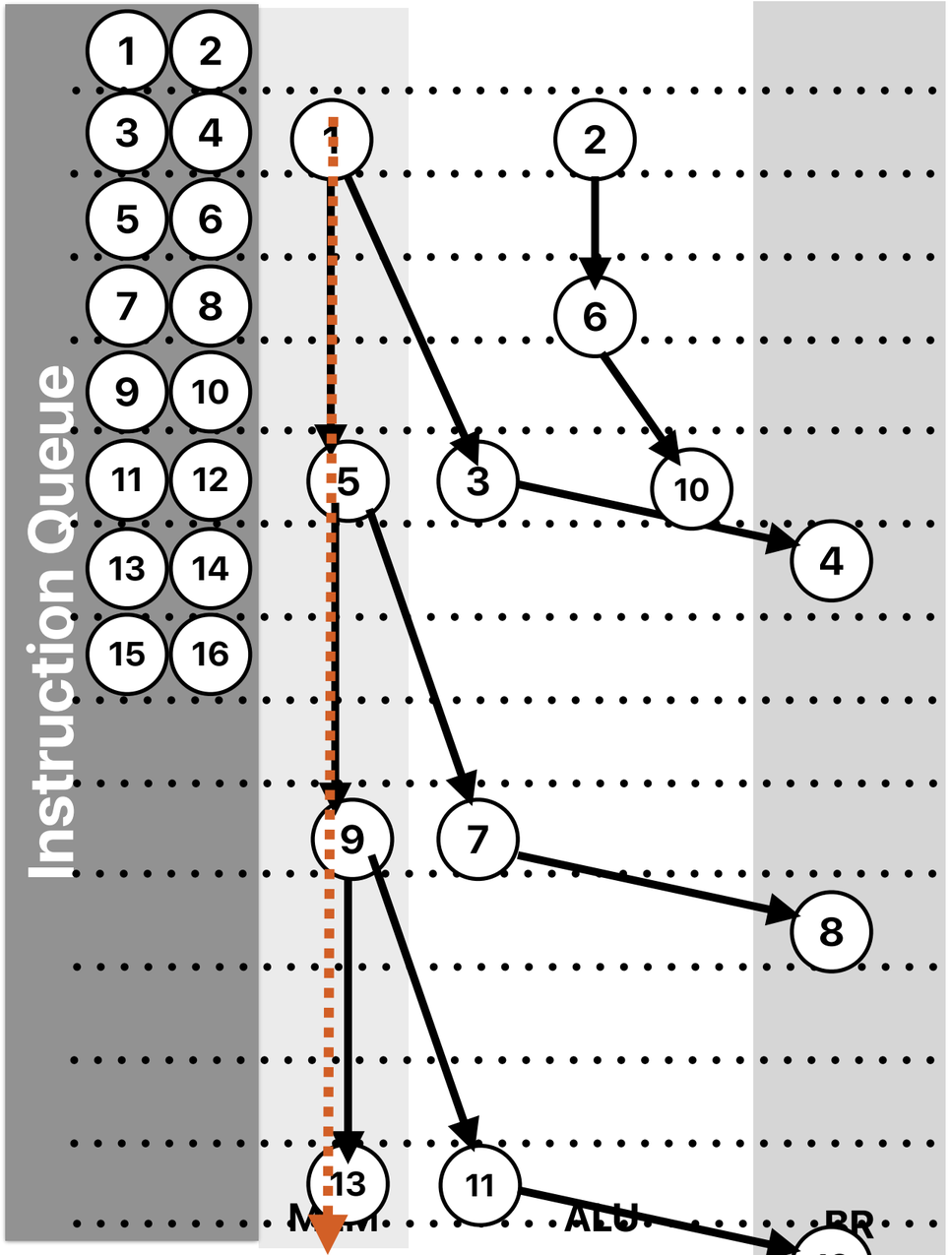
Best performing one at 2.7

What if we have “unlimited” fetch/issue width — “linked list”

Doesn’t help that much!
— It’s important that the programmer should write code that can exploit “ILP”
— But — there’re always cases we cannot do further in ILP

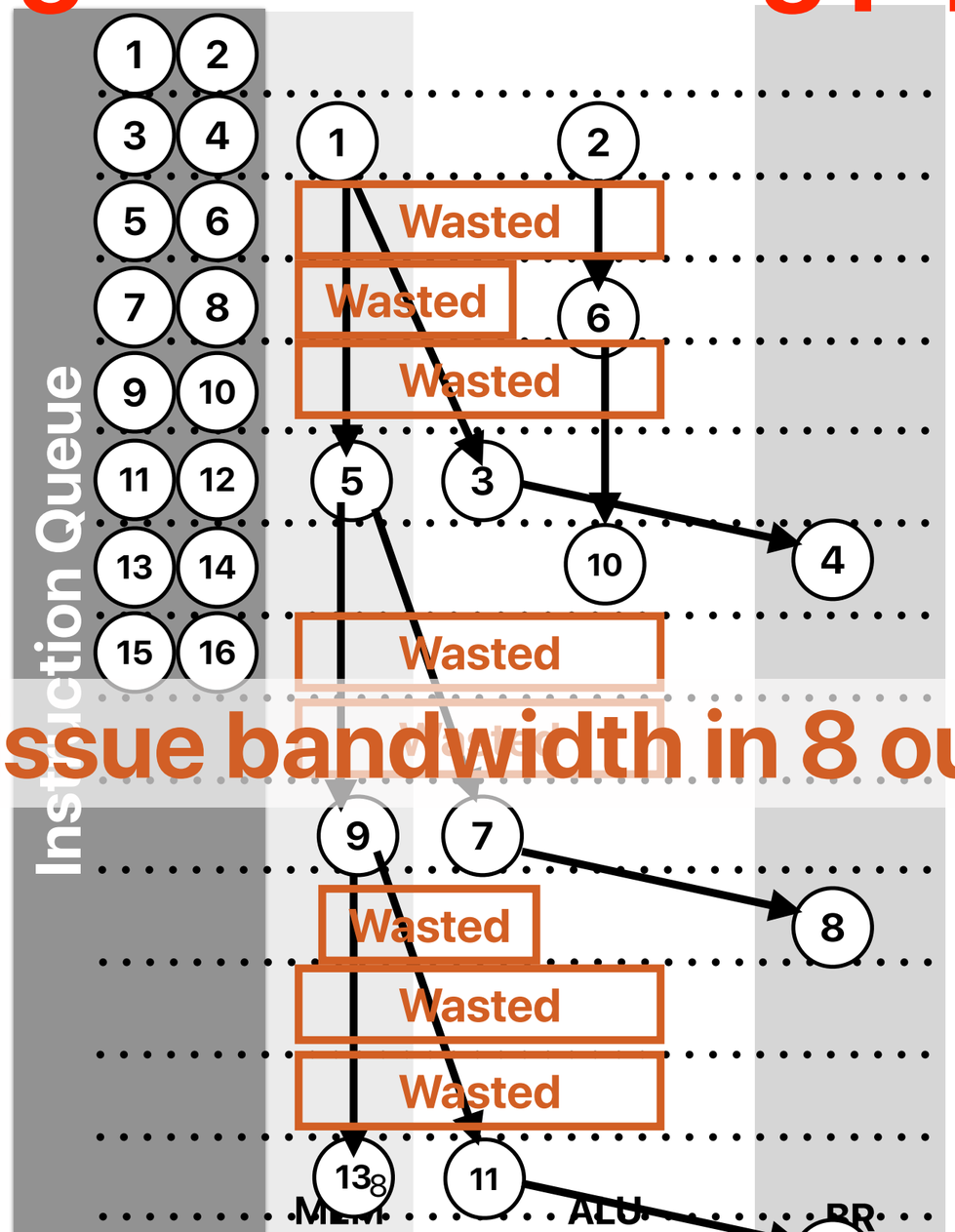
```
do {
    number_of_nodes++;
    current = current->next;
} while ( current != NULL );
```

```
① .L3:    movq    8(%rdi), %rdi
②         addl    $1, %eax
③         testq   %rdi, %rdi
④         jne     .L3
```



2-issue register renaming pipeline

```
① movq    8(%rdi), %rdi
② addl    $1, %eax
③ testq   %rdi, %rdi
④ jne     .L3
⑤ movq    8(%rdi), %rdi
⑥ addl    $1, %eax
⑦ testq   %rdi, %rdi
⑧ jne     .L3
⑨ movq    8(%rdi), %rdi
⑩ addl    $1, %eax
⑪ testq   %rdi, %rdi
⑫ jne     .L3
```

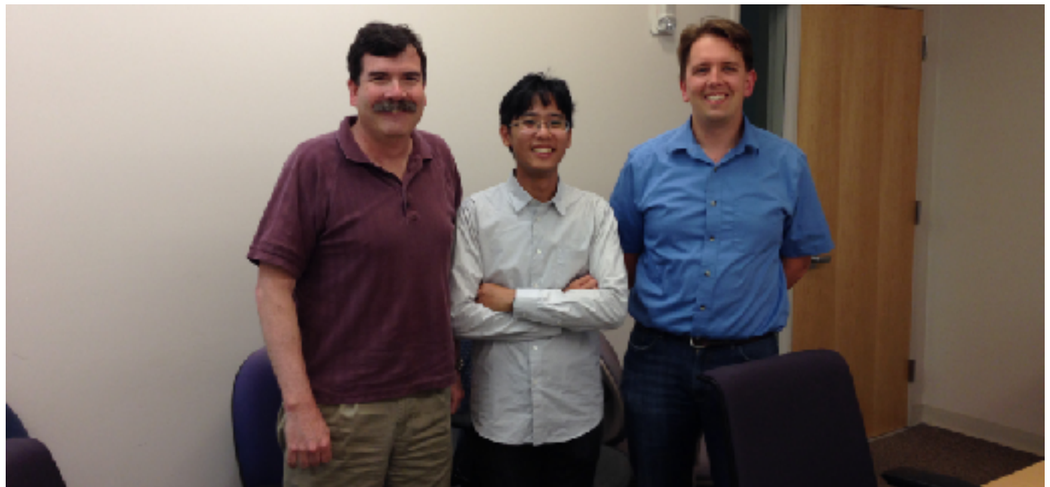


We're wasting the issue bandwidth in 8 out of 12 cycles

Outline

- Multithreaded processors
- Programming multithreaded processors & necessary architectural support

Simultaneous multithreading

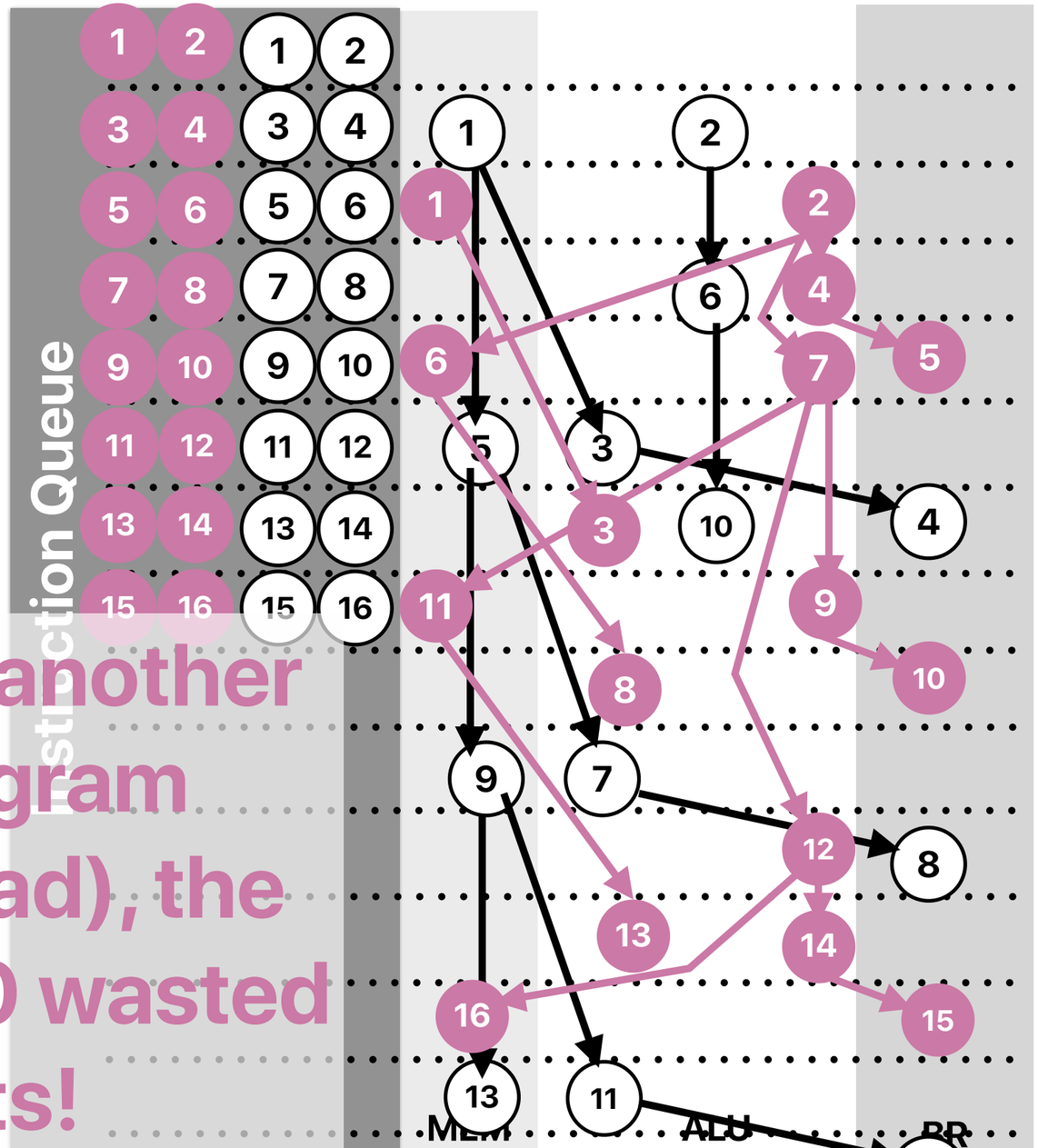


Simultaneous multithreading

- Invented by Dean Tullsen (Now a professor at **UCSD CSE**)
- The processor can schedule instructions from different threads/processes/programs
- Fetch instructions from different threads/processes to fill the not utilized part of pipeline
  - Exploit “thread level parallelism” (TLP) to solve the problem of insufficient ILP in a single thread
  - You need to create an illusion of multiple processors for OSs

Concept: Simultaneous Multithreading (SMT)

```
①  movq    8(%rdi), %rdi
②  addl    $1, %eax
③  testq   %rdi, %rdi
④  jne     .L3
⑤  movq    8(%rdi), %rdi
⑥  addl    $1, %eax
⑦  testq   %rdi, %rdi
⑧  jne     .L3
⑨  movq    8(%rdi), %rdi
⑩  addl    $1, %eax
⑪  testq   %rdi, %rdi
⑫  jne     .L3
```



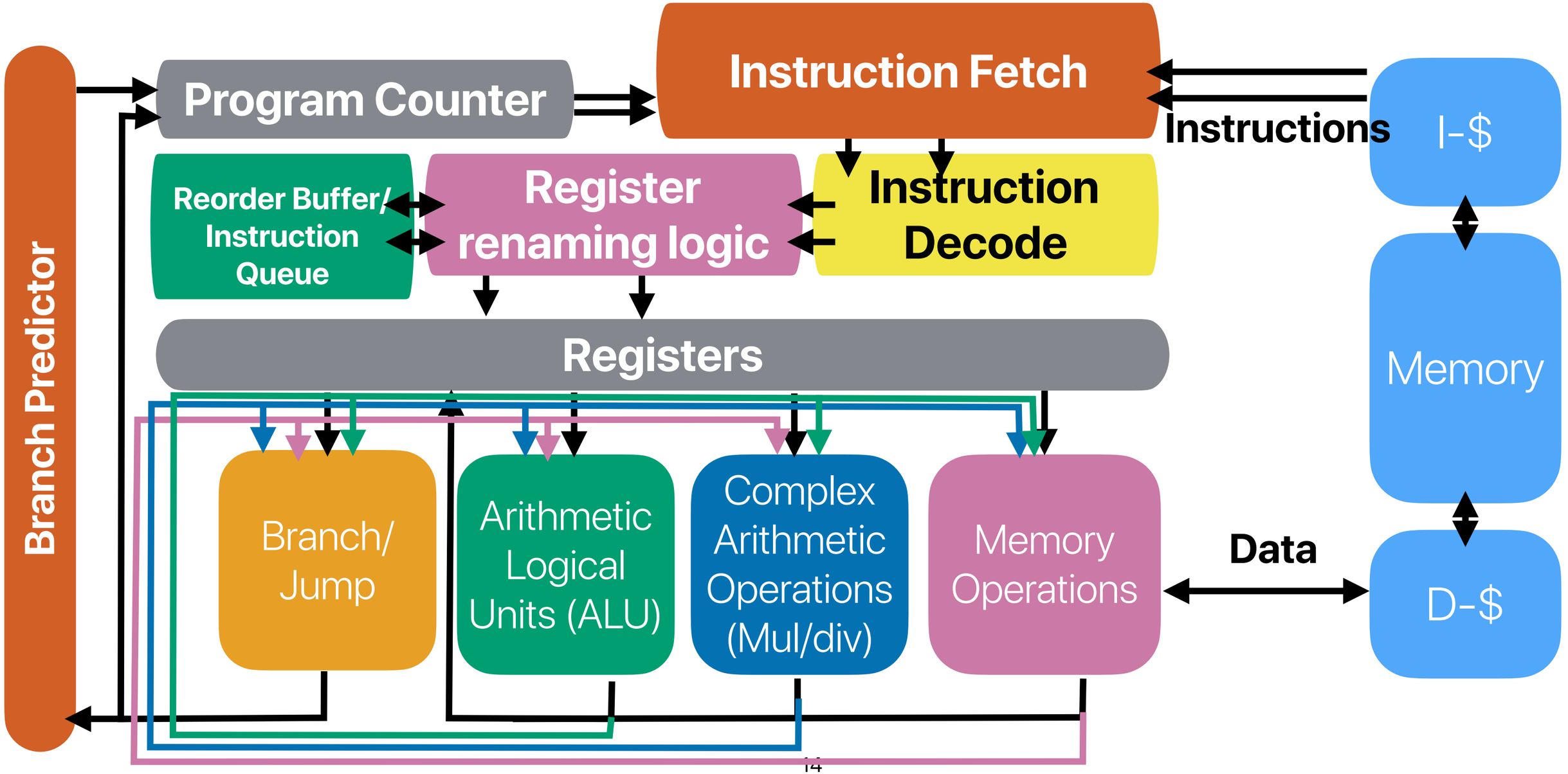
By scheduling another running program instance (thread), the processor has 0 wasted issue slots!

```
①  movl (%rdi), %ecx
②  addq $4, %rdi
③  addl %ecx, %eax
④  cmpq %rdx, %rdi
⑤  jne  .L3
⑥  movl (%rdi), %ecx
⑦  addq $4, %rdi
⑧  addl %ecx, %eax
⑨  cmpq %rdx, %rdi
⑩  jne  .L3
⑪  movl (%rdi), %ecx
⑫  addq $4, %rdi
⑬  addl %ecx, %eax
⑭  cmpq %rdx, %rdi
⑮  jne  .L3
⑯  movl (%rdi), %ecx
```

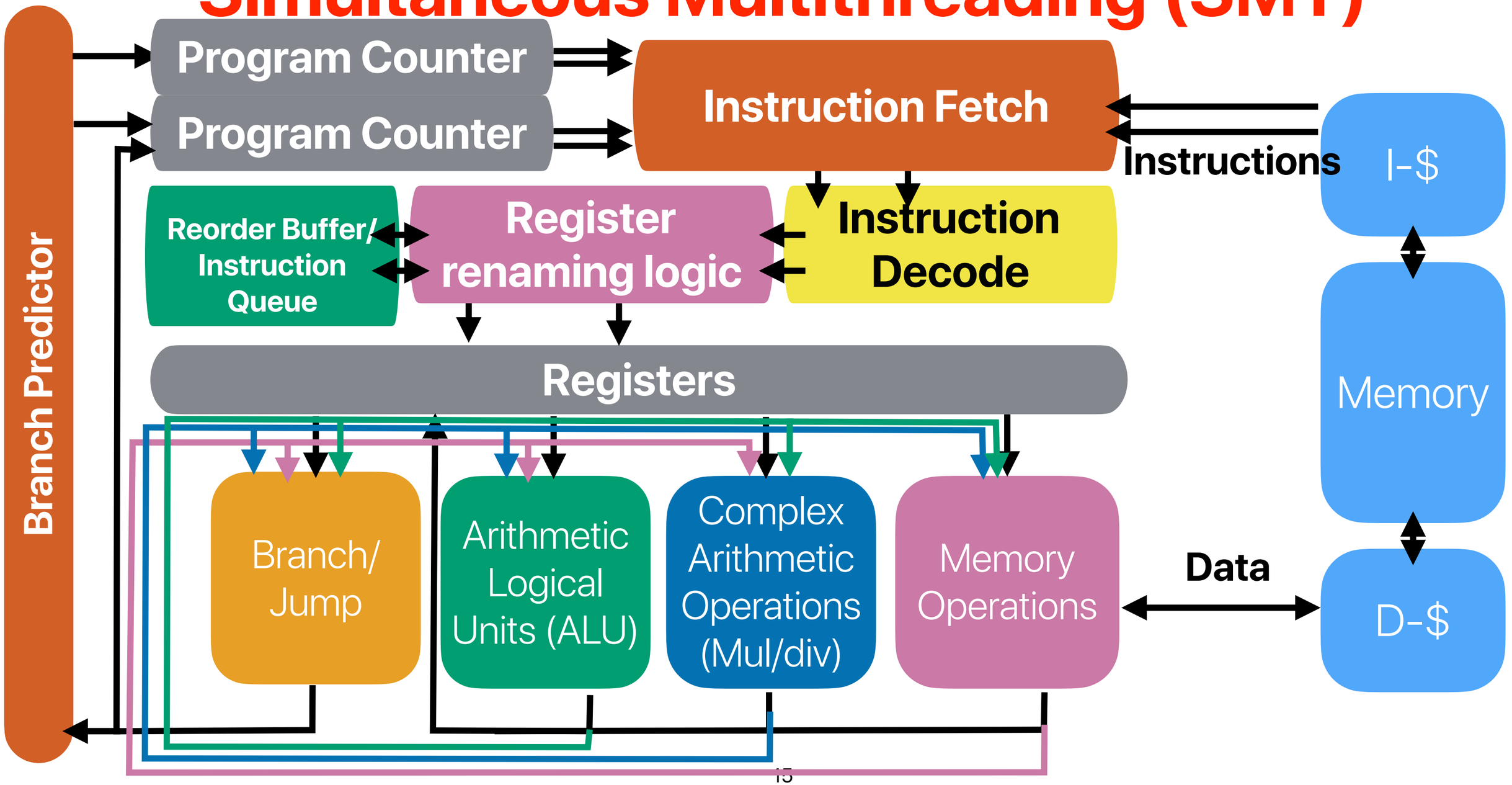
How do we support two running programs in one pipeline?

- We need two program counters
- We need two sets of architectural to physical register mappings
- We do not need
  - Duplicated cache — virtually indexed, physically tagged cache already addressed that
  - Duplicated pipeline functional units — isn’t sharing the whole purpose?
  - Duplicated reorder buffer — you simply need to tag which process the instruction belongs to

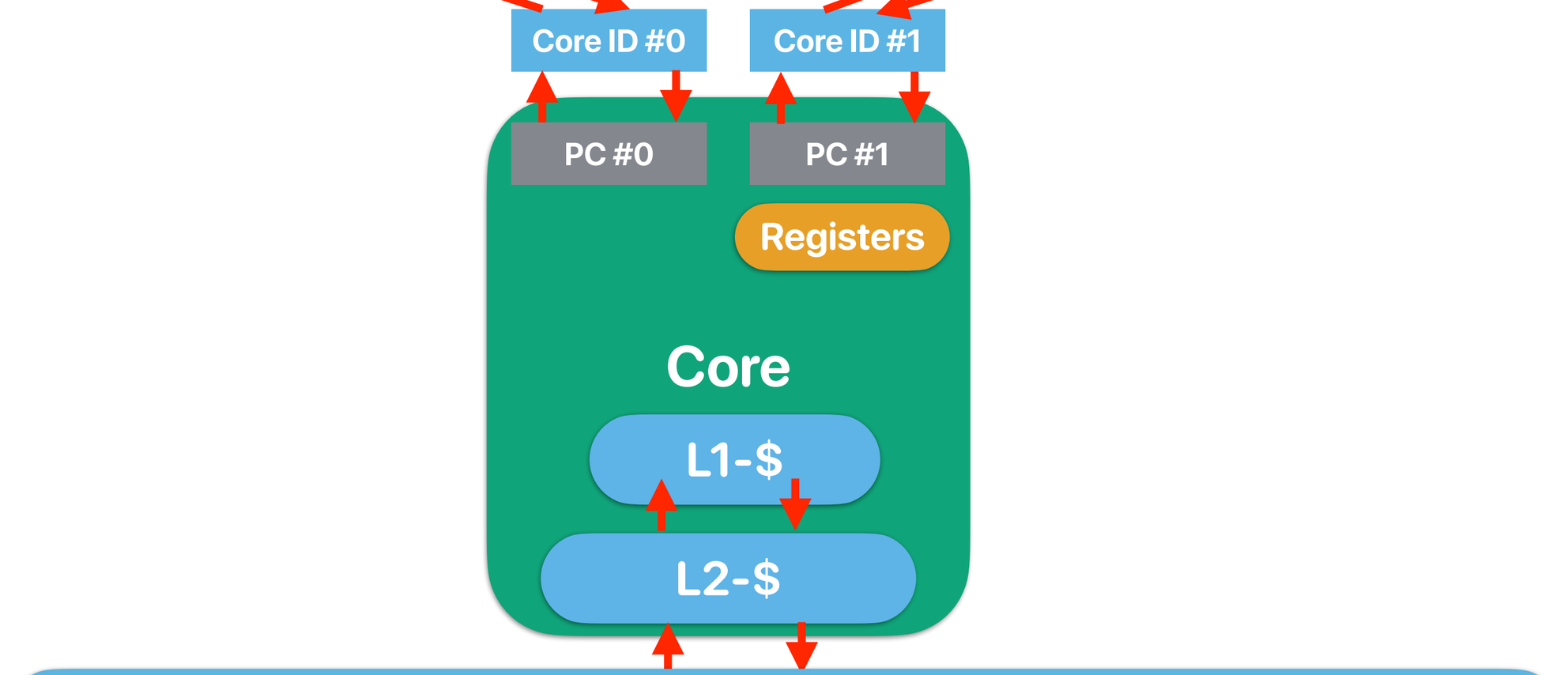
Recap: Register renaming



Simultaneous Multithreading (SMT)



SMT from the user/OS' perspective



https://www.pollev.com/hungweitseng close in 1:30

Pros/Cons of SMT

- How many of the following statements about SMT is/are correct?
  - ① SMT makes processors with deep pipelines more tolerable to mis-predicted branches
  - ② SMT can improve the throughput of a single-threaded application
  - ③ SMT processors can better utilize hardware during cache misses comparing with superscalar processors with the same issue width
  - ④ SMT processors can have higher cache miss rates comparing with superscalar processors with the same cache sizes when executing the same set of applications.

  A. 0
  B. 1
  C. 2
  D. 3
  E. 4

Pros/Cons of SMT

- How many of the following statements about SMT is/are correct?
  - ① SMT makes processors with deep pipelines more tolerable to mis-predicted branches
  - ② SMT can ~~improve~~ the throughput of a single-threaded application
  - ③ SMT processors can better utilize hardware during cache misses comparing with superscalar processors with the same issue width
  - ④ SMT processors can have higher cache miss rates comparing with superscalar processors with the same cache sizes when executing the same set of applications.

  A. 0
  B. 1
  C. 2
  D. 3
  E. 4

We can execute from other threads/contexts instead of the current one

hurt, b/c you are sharing resource with other threads.

We can execute from other threads/ contexts instead of the current one

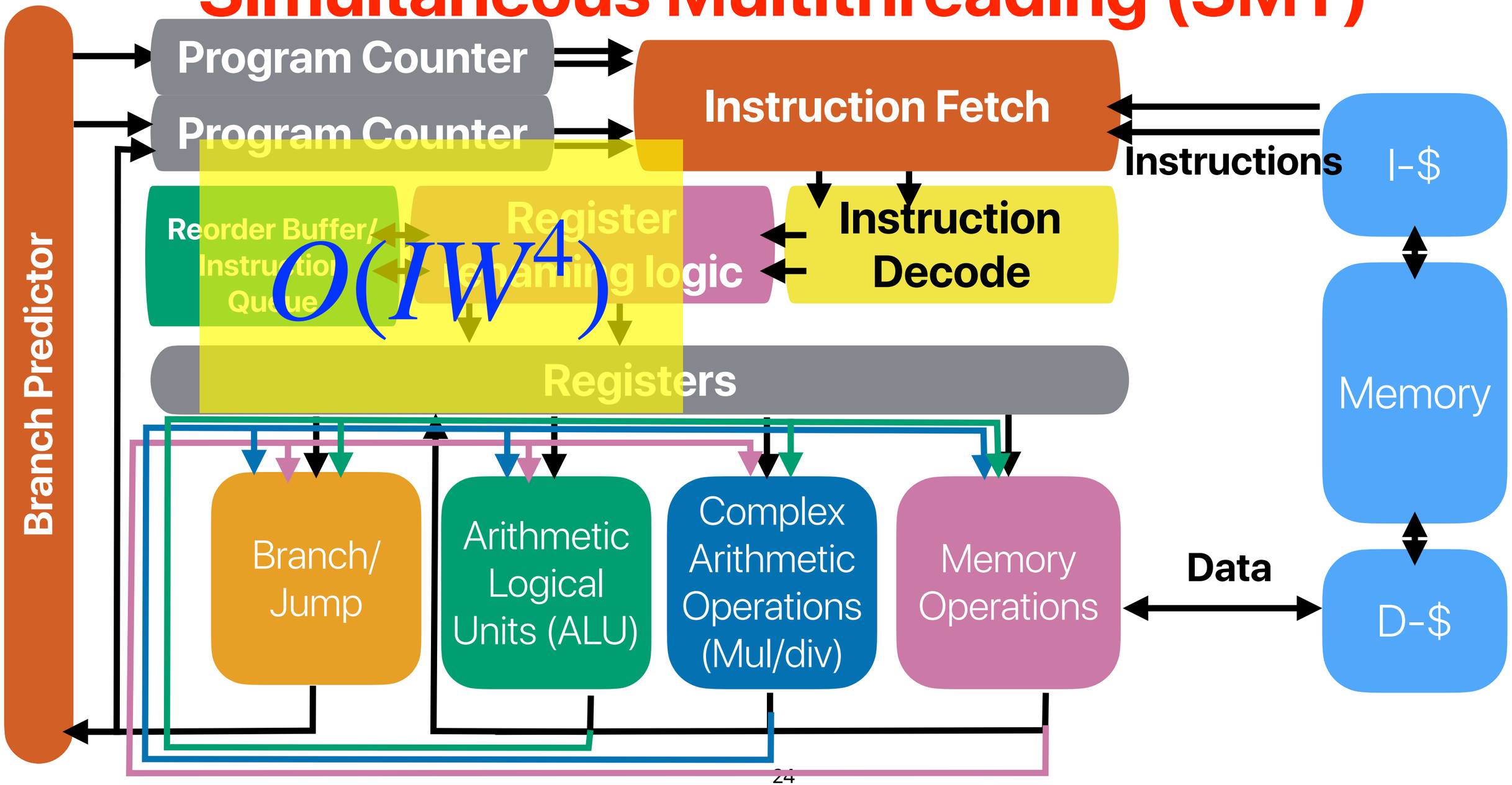
b/c we're sharing the cache

SMT

- Improve the throughput of execution
  - May increase the latency of a single thread
- Less branch penalty per thread
- Increase hardware utilization
- Simple hardware design: Only need to duplicate PC/Register Files
- Real Case:
  - Intel HyperThreading (supports up to two threads per core)
    - Intel Pentium 4, Intel Atom, Intel Core i7
  - AMD RyZen (Zen microarchitecture)
  - If you see a processor with “threads” more than “cores”, that must because of SMT!

```
Architecture:                    x86_64
CPU op-mode(s):                  32-bit, 64-bit
Byte Order:                      Little Endian
Address sizes:                   39 bits physical, 48 bits virtual
CPU(s):                          8
On-line CPU(s) list:             0-7
Thread(s) per core:              2
Core(s) per socket:              4
Socket(s):                       1
NUMA node(s):                    1
Vendor ID:                       GenuineIntel
CPU family:                      6
Model:                           151
Model name:                      12th Gen Intel(R) Core(TM) i3-12100F
Stepping:                        5
CPU MHz:                         3300.000
CPU max MHz:                     5500.0000
CPU min MHz:                     800.0000
BogoMIPS:                        6604.80
Virtualization:                  VT-x
L1d cache:                       192 KiB
L1i cache:                       128 KiB
```

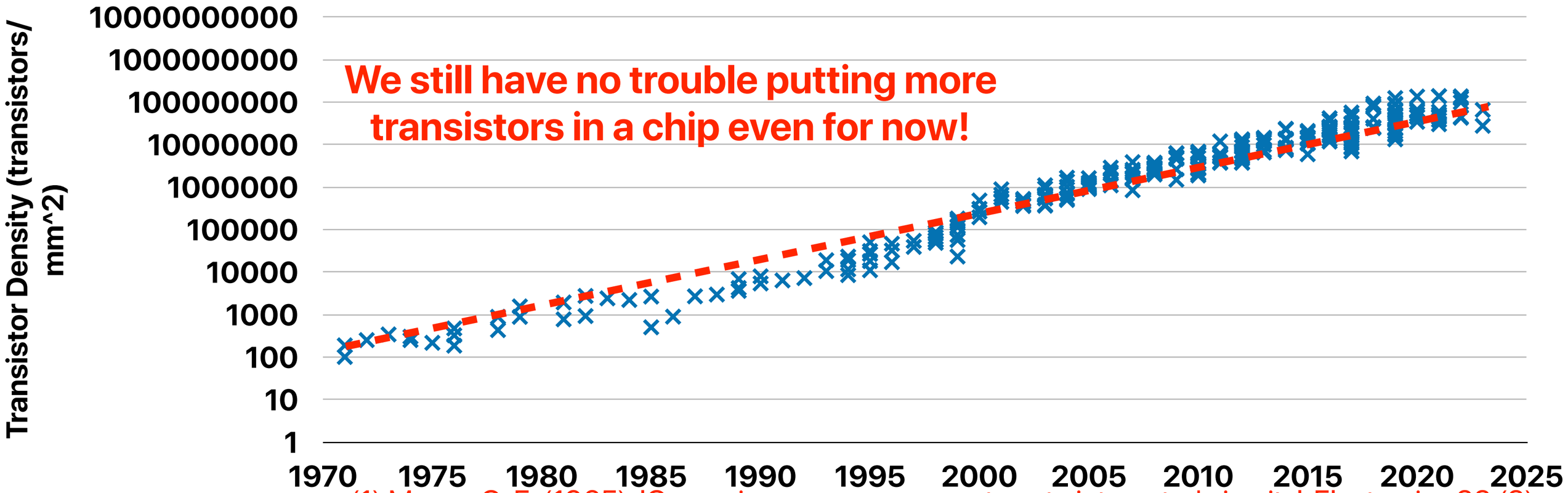
Simultaneous Multithreading (SMT)



Chip-Multiprocessors (CMP) or Multi-core processors

Recap: Moore's Law(1)

- The number of transistors we can build in a fixed area of silicon doubles every 12 ~ 24 months.
- Moore's Law "was" the most important driver for historic CPU performance gains

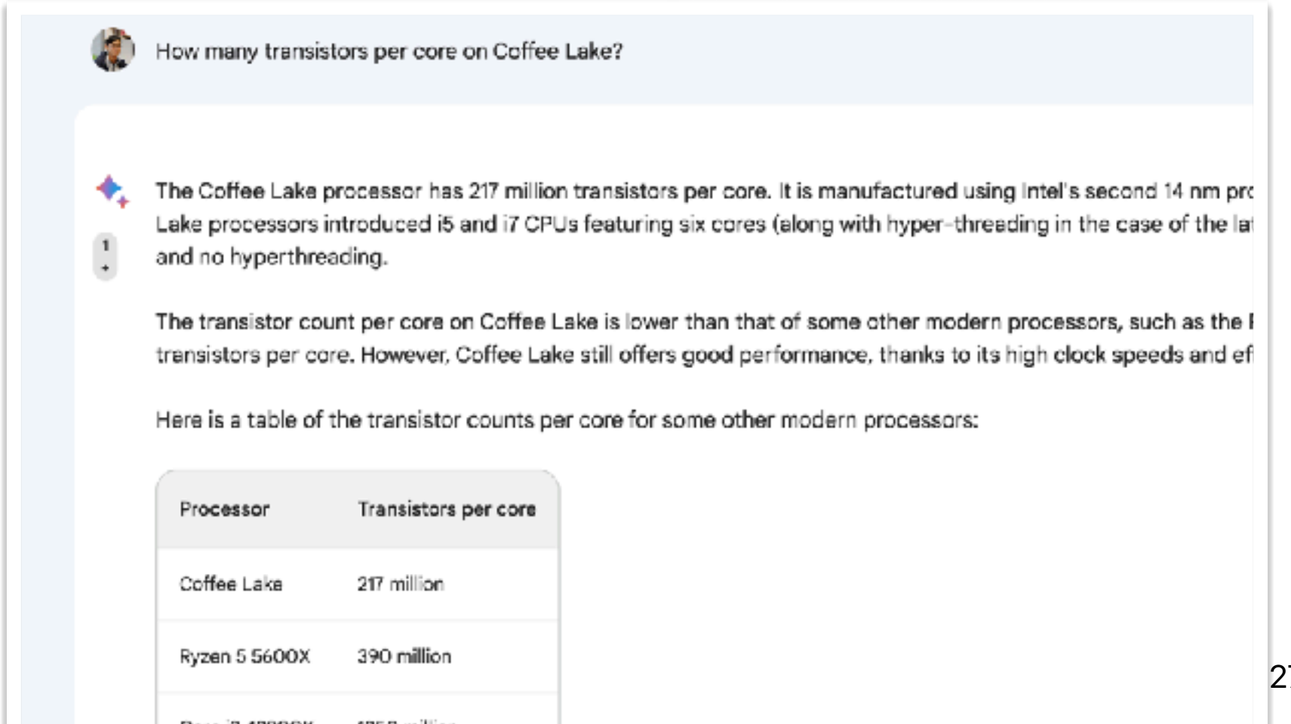


(1) Moore, G. E. (1965), 'Cramming more components onto integrated circuits', Electronics 38 (8) .

Plot based on https://en.wikipedia.org/wiki/Transistor_count by Hung-Wei Tseng

Transistor counts

Microarchitecture	Transistor Count	Issue-width	Year
Alder Lake	325 M	5x ALU, 7x Memory	2021
Coffee Lake	217 M	4x ALU, 4x Memory	2017
Sandy Bridge	290 M	3x ALU, 3x Memory	2011
Nehalem	182.75 M	3x ALU, 3x Memory	2008



How many transistors per core on Coffee Lake?

The Coffee Lake processor has 217 million transistors per core. It is manufactured using Intel's second 14 nm pro
Lake processors introduced i5 and i7 CPUs featuring six cores (along with hyper-threading in the case of the lat
and no hyperthreading.

The transistor count per core on Coffee Lake is lower than that of some other modern processors, such as the I
transistors per core. However, Coffee Lake still offers good performance, thanks to its high clock speeds and ef

Here is a table of the transistor counts per core for some other modern processors:

Processor	Transistors per core
Coffee Lake	217 million
Ryzen 5 5600X	390 million

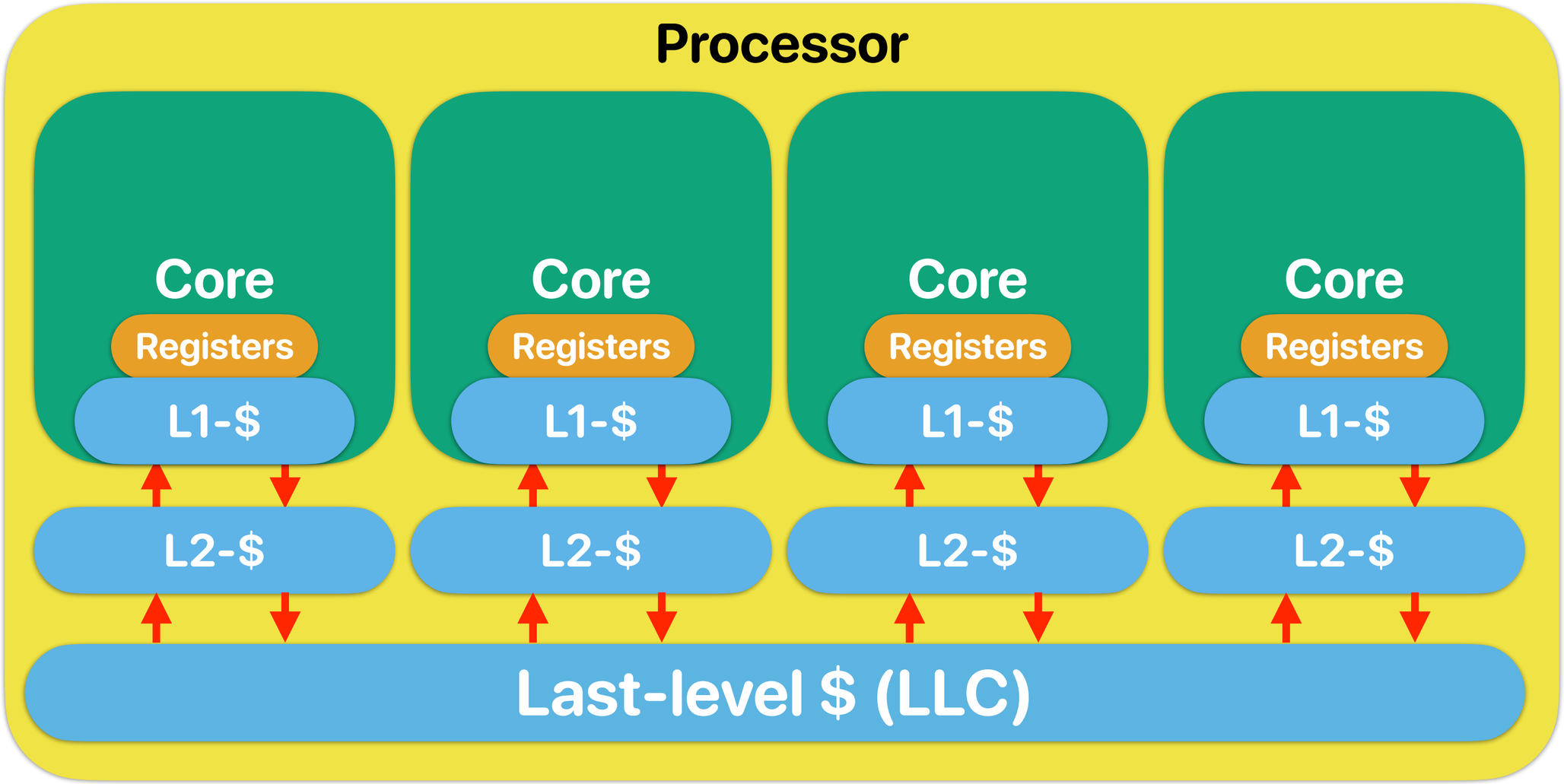
2x 3-issue ALUs Nehalem

Nehalem 6-issue | Alder Lake 12-issue | Nehalem 6-issue

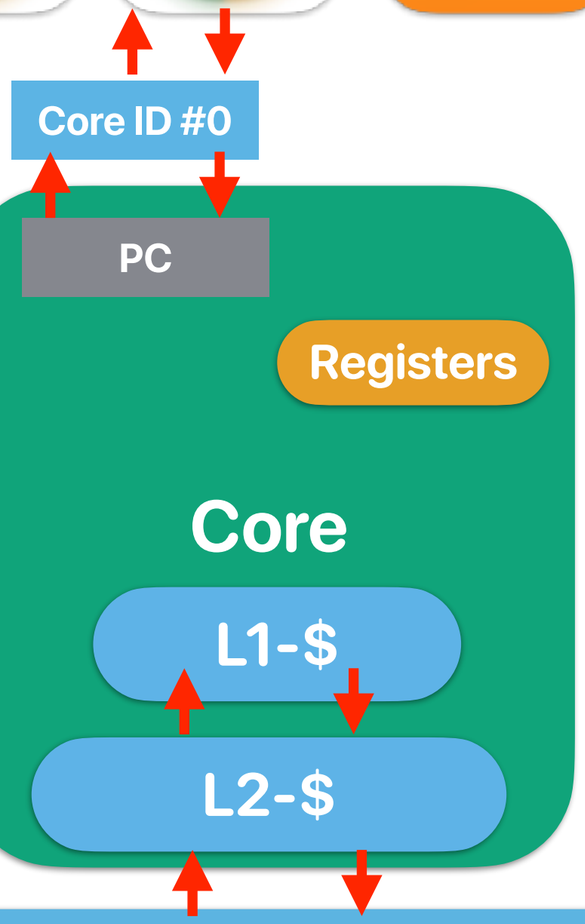
1x 5-issue ALUs Alder Lake

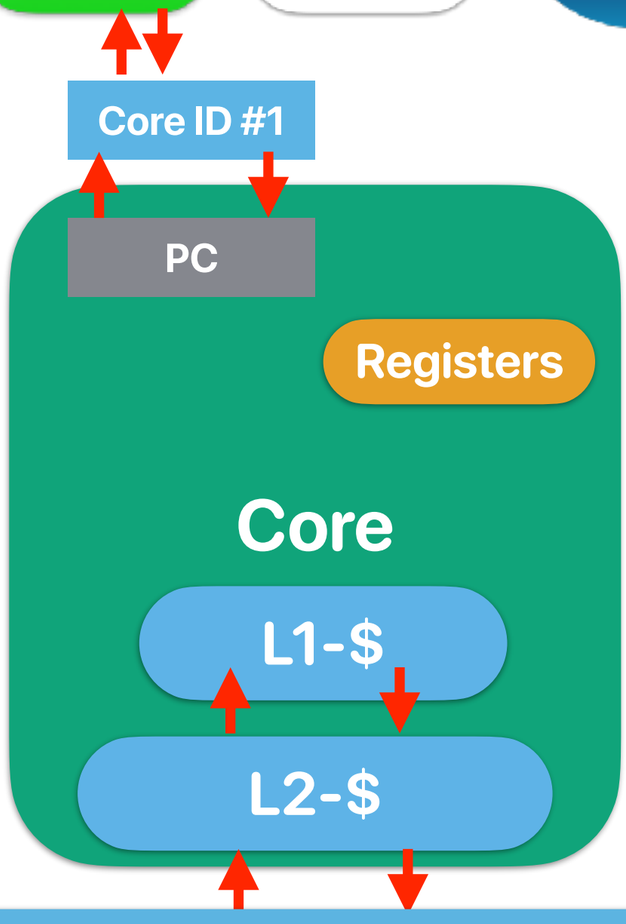
Based on https://en.wikipedia.org/wiki/Transistor_count

Concept of CMP



CMP from the user/OS' perspective





Last-level $ (LLC)

https://www.pollev.com/hungweitseng close in 1:30

SMT v.s. CMP

- An SMT processor is basically a SuperScalar processor with multiple instruction front-end. Assume within the same chip area, we can build an SMT processor supporting 4 threads, with 6-issue pipeline, 64KB cache or — a CMP with 4x 2-issue pipeline & 16KB cache in each core. Please identify how many of the following statements are/is correct when running programs on these processors.
  - ① If we are just running one program in the system, the program will perform better on an SMT processor
  - ② If we are running 4 applications simultaneously, the cache miss rates will be higher in the SMT processor
  - ③ If we are running 4 applications simultaneously, the branch mis-prediction will be higher in the SMT processor
  - ④ If we are running one program with 4 parallel threads, the cache miss rates will be higher in the SMT processor
  - ⑤ If we are running one program with 4 parallel threads simultaneously, the branch mis-prediction will be longer in the SMT processor

  A. 1
  B. 2
  C. 3
  D. 4
  E. 5

SMT v.s. CMP

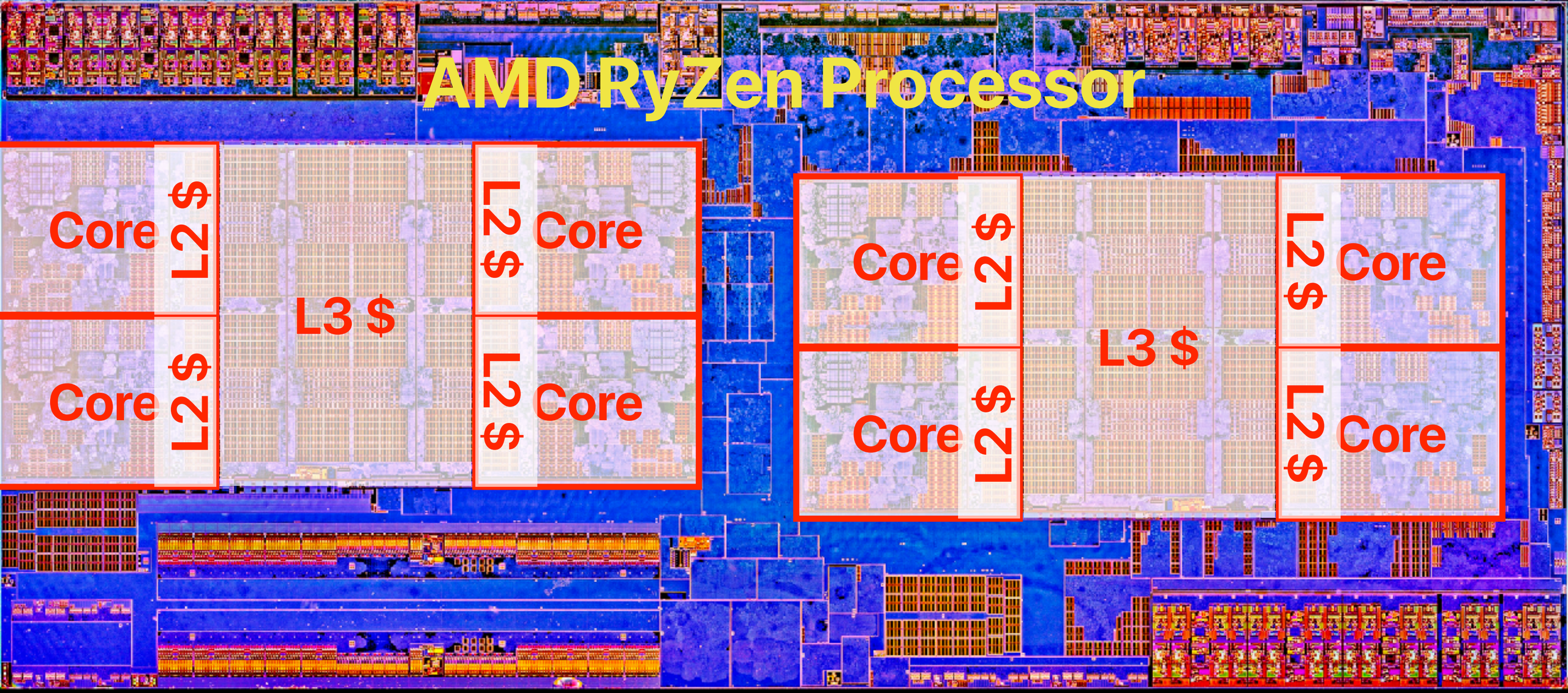
- An SMT processor is basically a SuperScalar processor with multiple instruction front-end. Assume within the same chip area, we can build an SMT processor supporting 4 threads, with 6-issue pipeline, 64KB cache or — a CMP with 4x 2-issue pipeline & 16KB cache in each core. Please identify how many of the following statements are/is correct when running programs on these processors.
  - ① If we are just running one program in the system, the program will perform better on an SMT processor — you have more resources for the program
  - ② If we are running 4 applications simultaneously, the cache miss rates will be higher in the SMT processor
  - ③ If we are running 4 applications simultaneously, the branch mis-prediction will be higher in the SMT processor — it depends!
  - ④ If we are running one program with 4 parallel threads, the cache miss rates will be higher in the SMT processor — it depends! — it depends!
  - ⑤ If we are running one program with 4 parallel threads simultaneously, the branch mis-prediction will be longer in the SMT processor — it depends!

A. 1
B. 2
C. 3
D. 4
E. 5

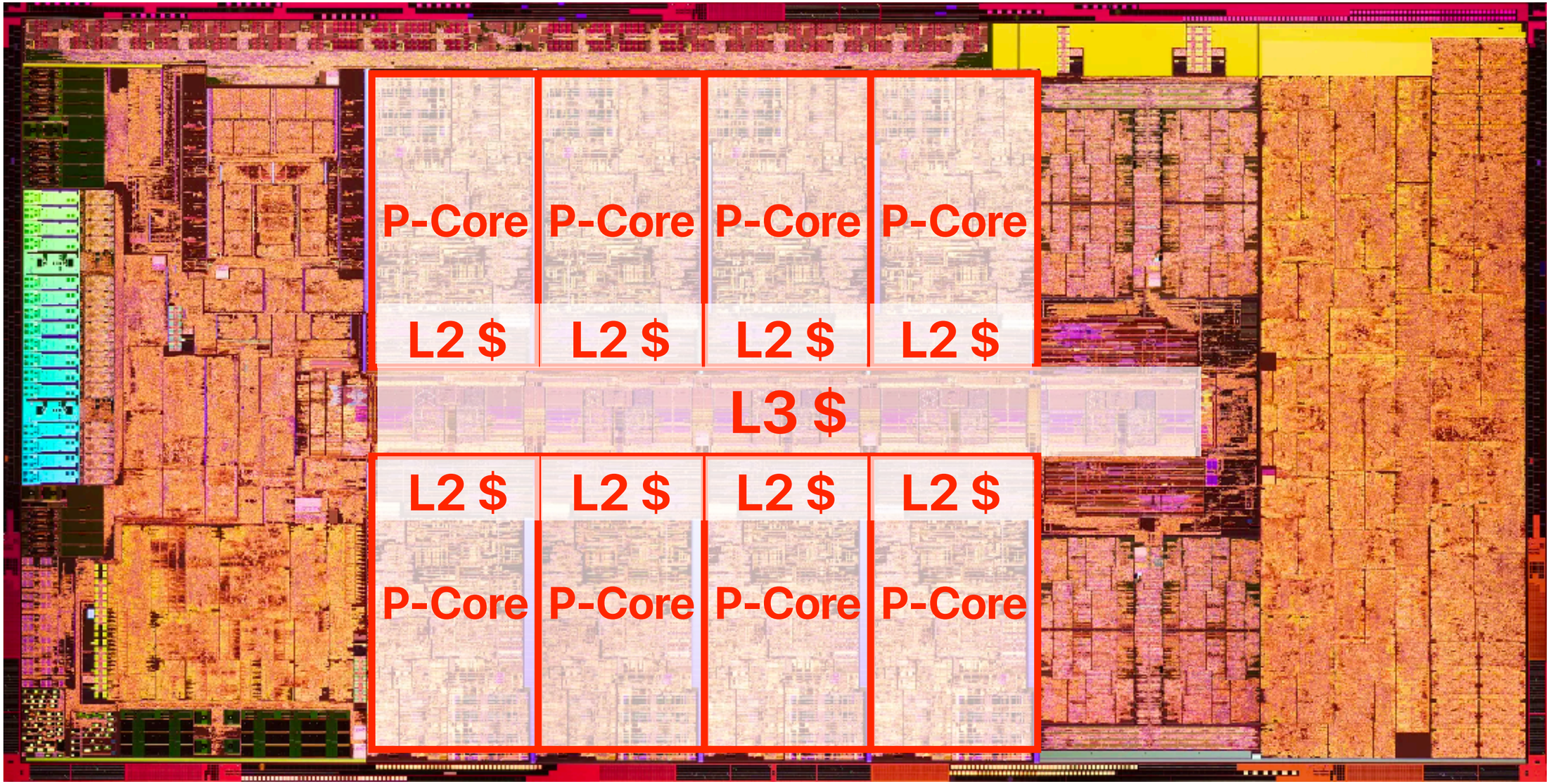
There is no clear win on each — why not having both?

The only thing we know for sure — if we don't parallel the program, it won't get any faster on CMP

Modern processors have both CMP/SMT


AMD RyZen Processor
Core
L2 $
L3 $
Core
L2 $
Core
L2 $
Core
L2 $
Core
L2 $
L3 $
Core
L2 $
Core
L2 $
AMD
RYZEN

Intel Alder Lake



SMT v.s. CMP

- An SMT processor is basically a SuperScalar processor with multiple instruction front-end. Assume within the same chip area, we can build an SMT processor supporting 4 threads, with 6-issue pipeline, 64KB cache or — a CMP with 4x 2-issue pipeline & 16KB cache in each core. Please identify how many of the following statements are/is correct when running programs on these processors.
  - ① If we are just running one program in the system, the program will perform better on an SMT processor **— you have more resources for the program**
  - ② If we are running 4 applications simultaneously, the cache miss rates will be higher in the SMT processor **— it depends!**
  - ③ If we are running 4 applications simultaneously, the branch mis-prediction will be higher in the SMT processor **— it depends!**
  - ④ If we are running one program with 4 parallel threads, the cache miss rates will be higher in the SMT processor **— it depends!**
  - ⑤ If we are running one program with 4 parallel threads simultaneously, the branch mis-prediction will be longer in the SMT processor **— it depends!**

  A. 1
  B. 2
  C. 3
  D. 4
  E. 5

There is no clear win on each — why not having both?

The only thing we know for sure — if we don't parallel the program, it won't get any faster on CMP

Announcements

- Assignment #4 — due this Sunday
- No lecture next Monday — Labor Day!

Computer
Science &
Engineering

142

つづく