NHR FAU

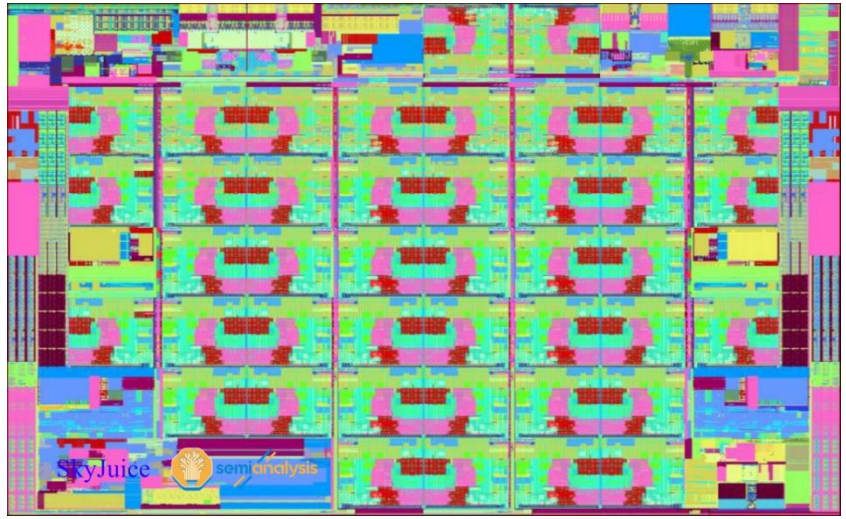
FAU Friedrich-Alexander-Universität Erlangen-Nürnberg

Modern compute node architecture (multicore)

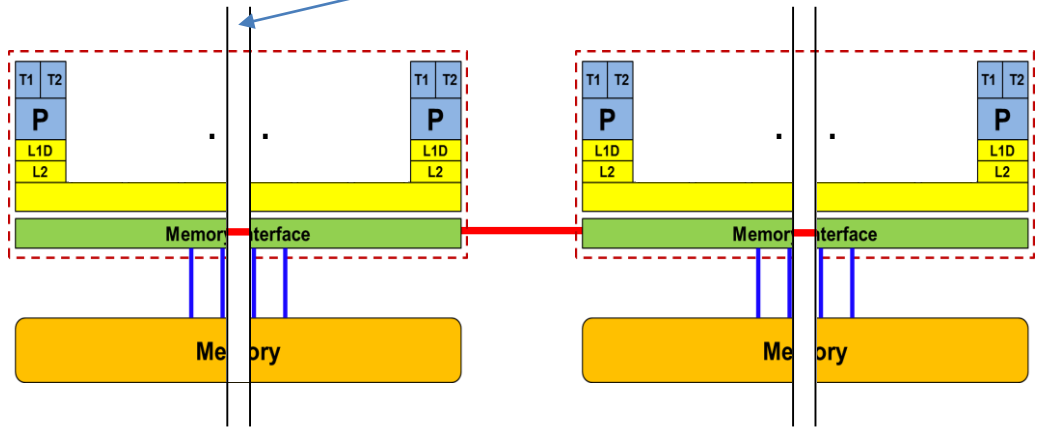
An introduction for software developers

Multi-core today: Intel Xeon Ice Lake (2021)

- Xeon “Ice Lake SP” (Platinum/Gold/Silver/Bronze):
  Up to 40 cores running at 2+ GHz (+ “Turbo Mode” 3.7 GHz),
- Simultaneous Multithreading
  → reports as 80-way chip
- ~15 Billion Transistors / ~10 nm / up to 270 W
- Die size: up to ~600 mm²



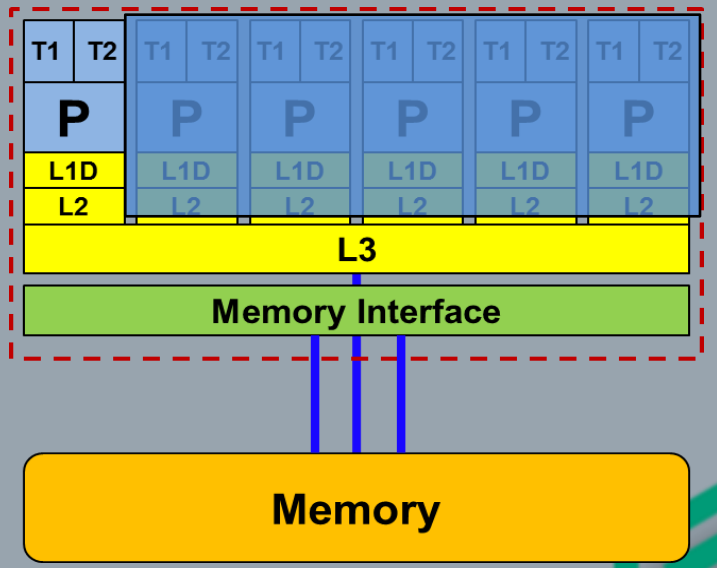
Optional: “Sub-NUMA Clustering” (SNC) mode



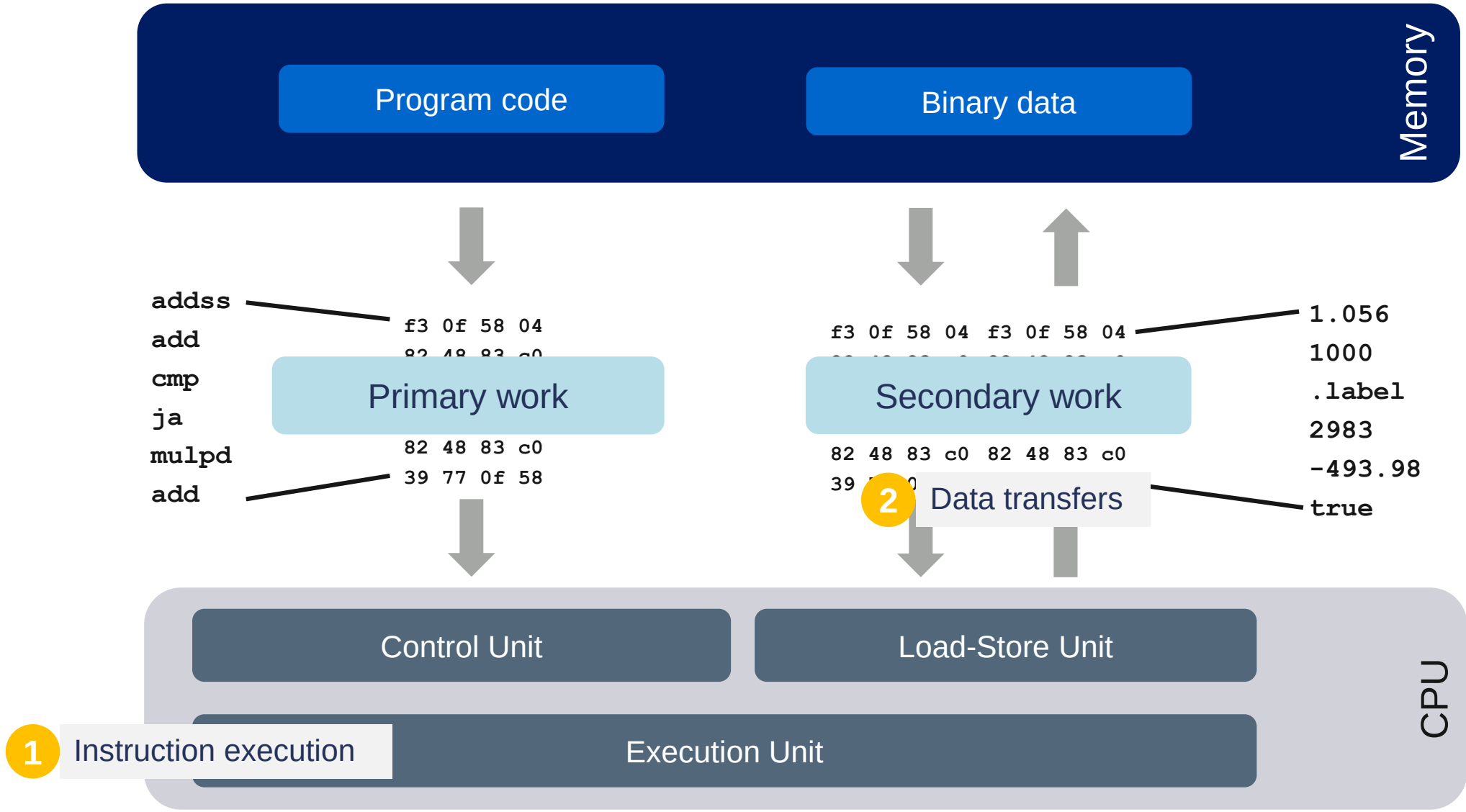
2-socket server

https://ark.intel.com/content/www/us/en/ark.html#@PanelLabel595

A deeper dive into core architecture



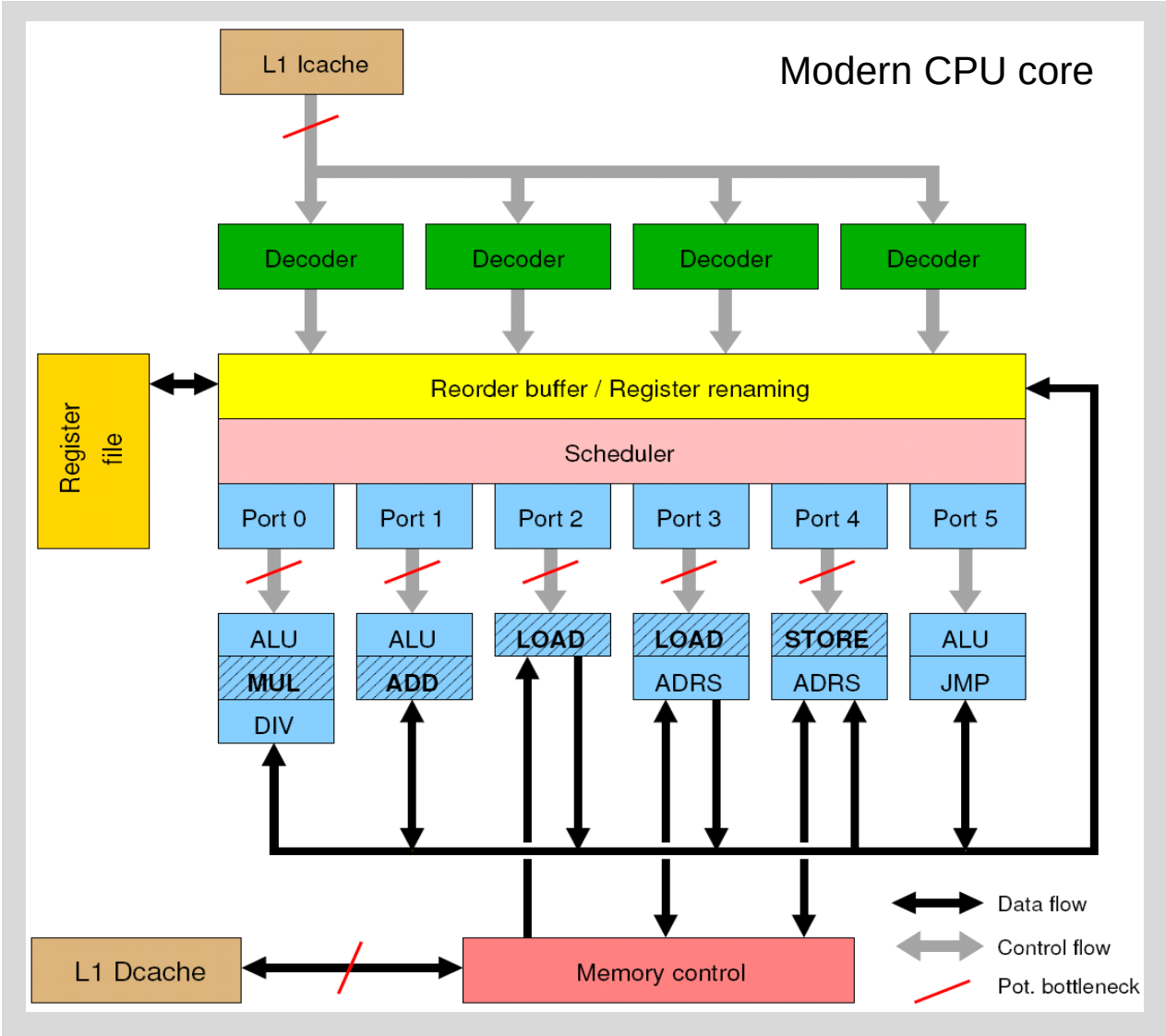
Stored Program Computer



General-purpose cache based microprocessor core

- Implements “Stored Program Computer” concept
- Similar designs on all modern systems
- (Still) multiple potential bottlenecks

The clock cycle is the “heartbeat” of the core



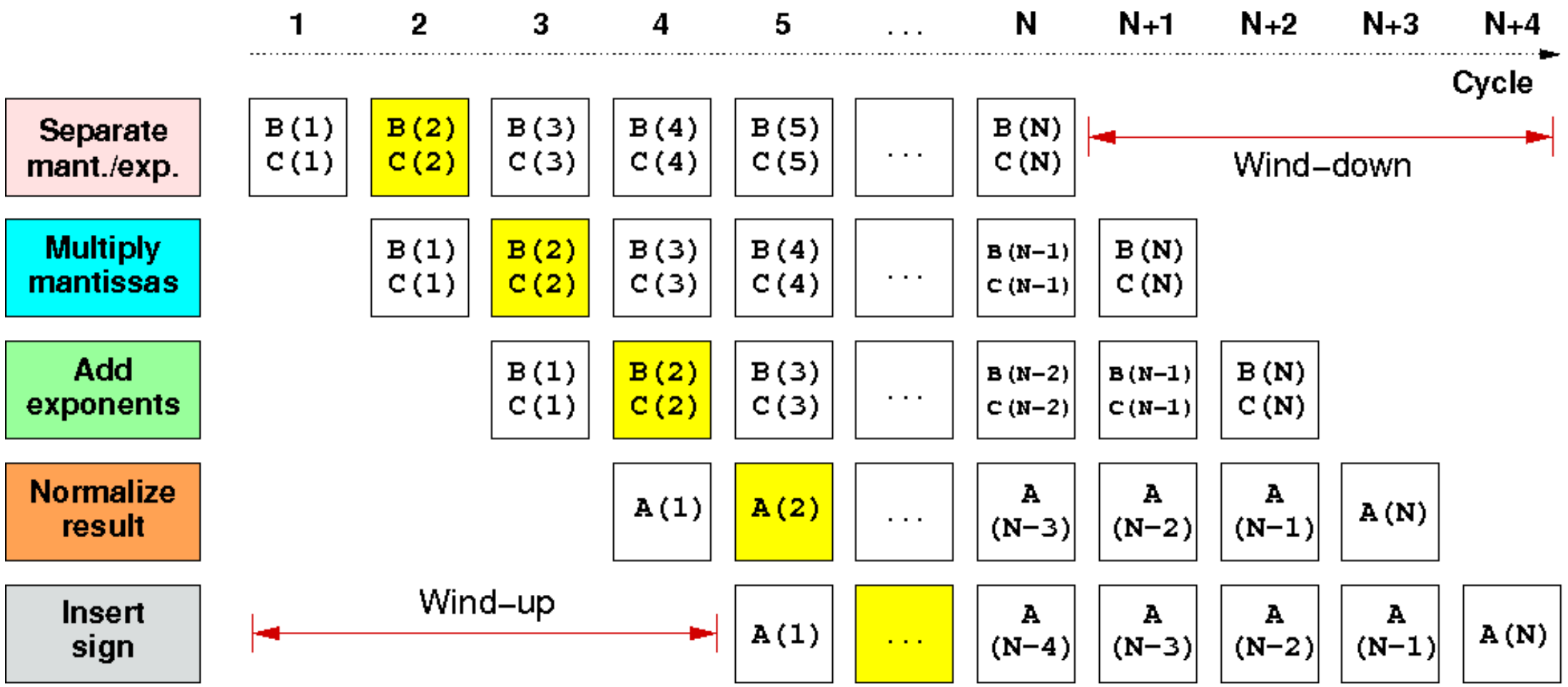
NHR FAU

FAU Friedrich-Alexander-Universität Erlangen-Nürnberg

In-core features

Pipelining, Superscalarity, SIMD

5-stage multiplication pipeline: `A(i)=B(i)*C(i) ; i=1,...,N`

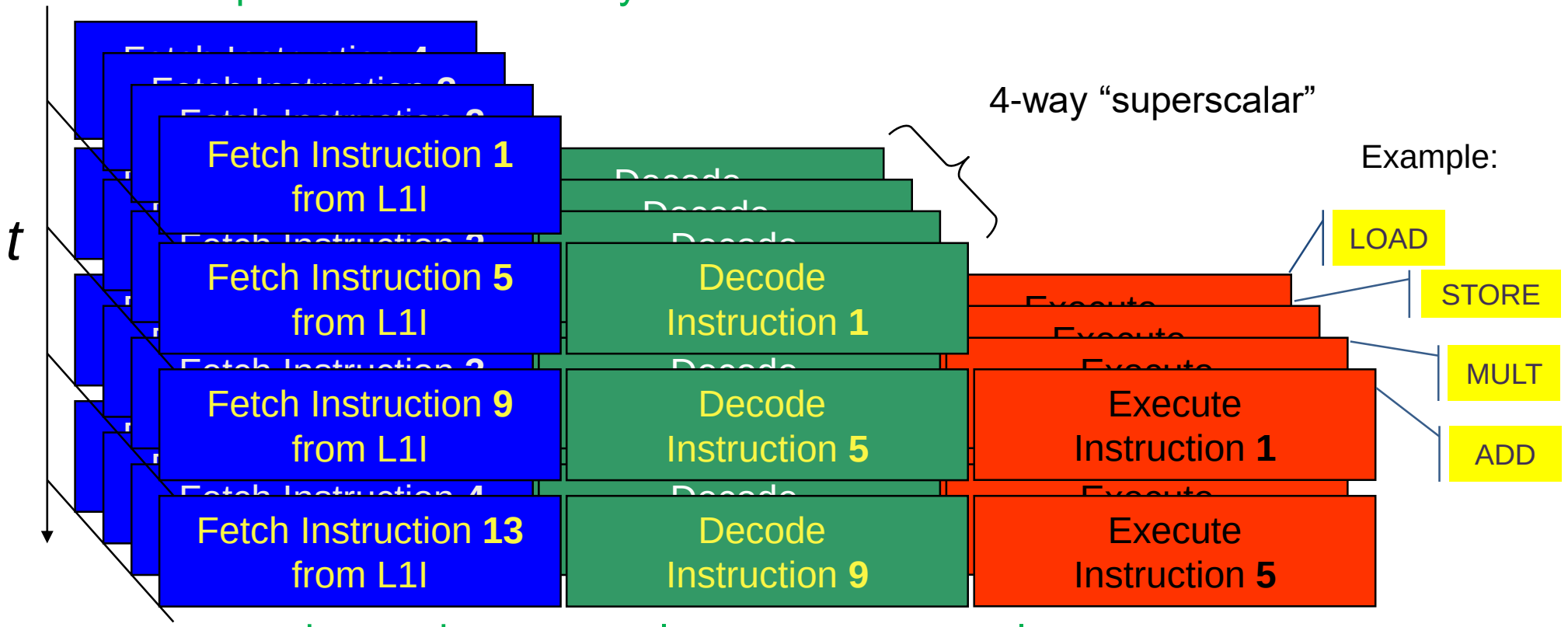


First result is available after 5 cycles (=latency of pipeline)!

Wind-up/-down phases: Empty pipeline stages

Instruction-level parallelism: Superscalar execution

Multiple units enable use of Instruction Level Parallelism (ILP):
Instruction stream is “parallelized” on the fly



- Issuing m concurrent instructions per cycle: m-way superscalar
- Modern processors are 3- to 6-way superscalar & can perform 2 floating point instructions per cycles

Superscalar/OoO execution and steady state

Instruction execution

STORE (Latency: 2 cy)

LOAD (Latency: 4 cy)

ADD (Latency: 3cy)

```
for(int i=1; i<n; ++i)
        a[i] = a[i] + c;
```

Cycle 1	load a[1]		
Cycle 2	load a[2]		
Cycle 3	load a[3]		
Cycle 4	load a[4]		
Cycle 5	load a[5]	add a[1]=c,a[1]	
Cycle 6	load a[6]	add a[2]=c,a[2]	
Cycle 7	load a[7]	add a[3]=c,a[3]	
Cycle 8	load a[8]	add a[4]=c,a[4]	store a[1]
Cycle 9	load a[9]	add a[5]=c,a[5]	store a[2]
Cycle 10	load a[10]	add a[6]=c,a[6]	store a[3]
Cycle 11	load a[11]	add a[7]=c,a[7]	store a[4]
Cycle 12	load a[12]	add a[8]=c,a[8]	store a[5]
Cycle 13	load a[13]	add a[9]=c,a[9]	store a[6]
Cycle 14	load a[14]	add a[10]=c,a[10]	store a[7]
Cycle 15	load a[15]	add a[11]=c,a[11]	store a[8]
Cycle 16	load a[16]	add a[12]=c,a[12]	store a[10]
…	…	…	…

"Steady state:"
3 instructions/cy
("3-way superscalar execution")

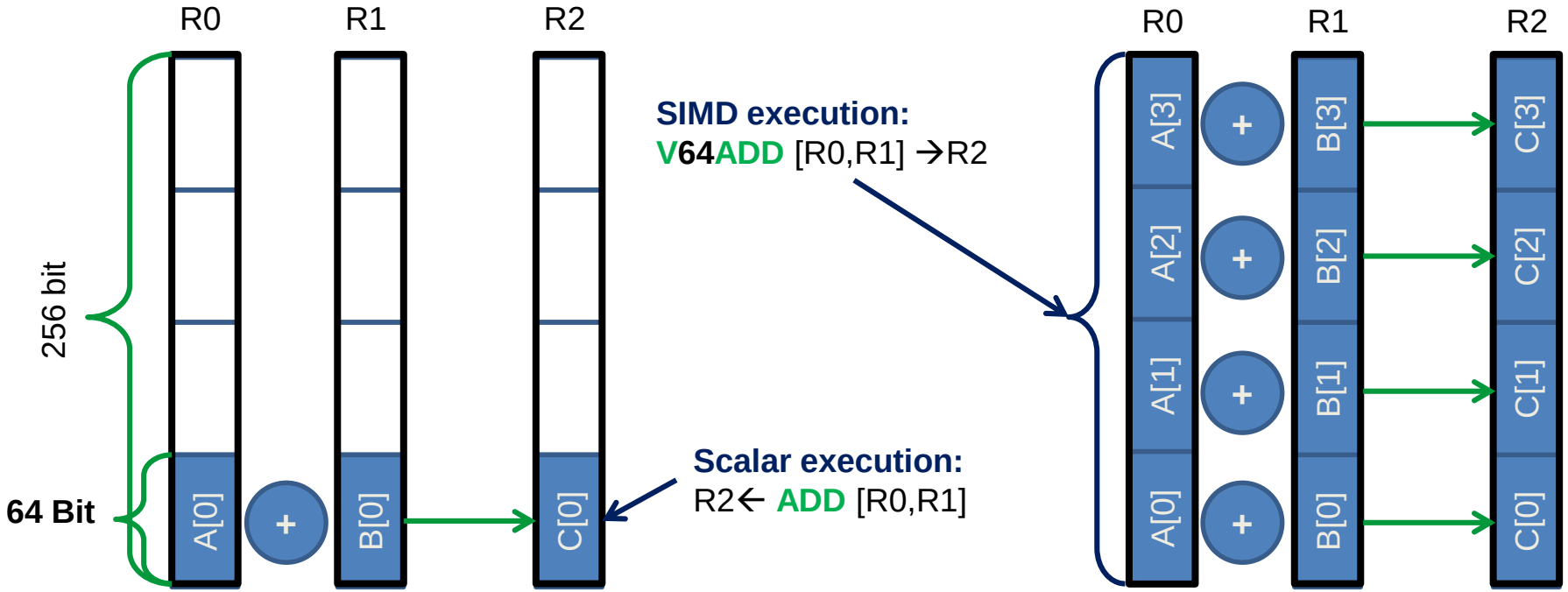
Instructions Per Cycle: IPC=3

Cycles Per Instruction: CPI=0.33

Hardware takes care of executing instructions as soon as their operands are available:
Out-Of-Order (OOO) execution

SIMD processing

- Single Instruction Multiple Data (SIMD) operations allow the execution of the same operation on “wide” registers from a single instruction
- x86 SIMD instruction sets:
  - SSE: register width = 128 Bit → 2 double precision floating point operands
  - AVX: register width = 256 Bit → 4 double precision floating point operands
  - AVX-512: … you guessed it!
- Adding two registers holding double precision floating point operands:

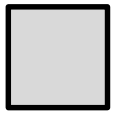


Scalar (non-SIMD) execution

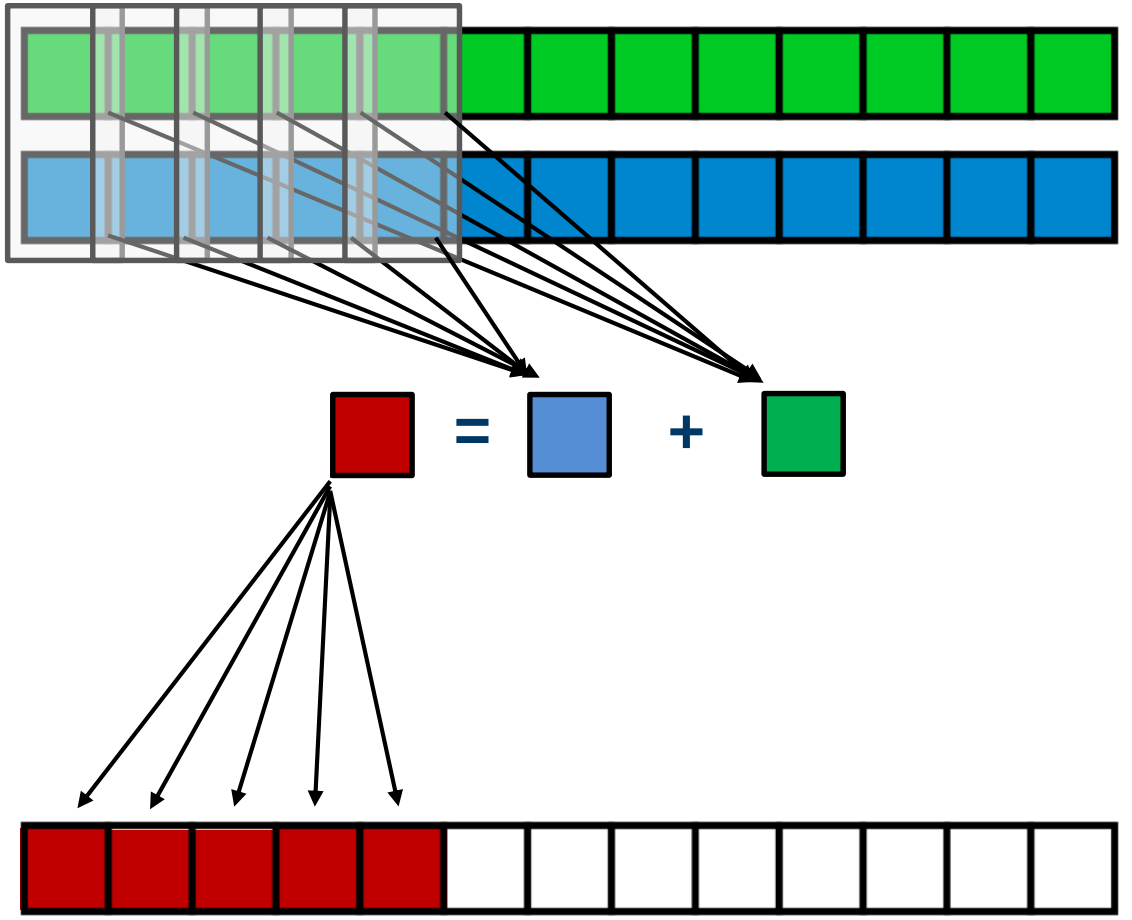
```
double *A, *B, *C;
for (int j=0; j<size; j++){
    A[j] = B[j] + C[j];
}
```

Register width:

- 1 operand (scalar)



Scalar execution



Data-parallel (SIMD) execution

```
double *A, *B, *C;
for (int j=0; j<size; j++){
    A[j] = B[j] + C[j];
}
```

Register widths (double prec.):

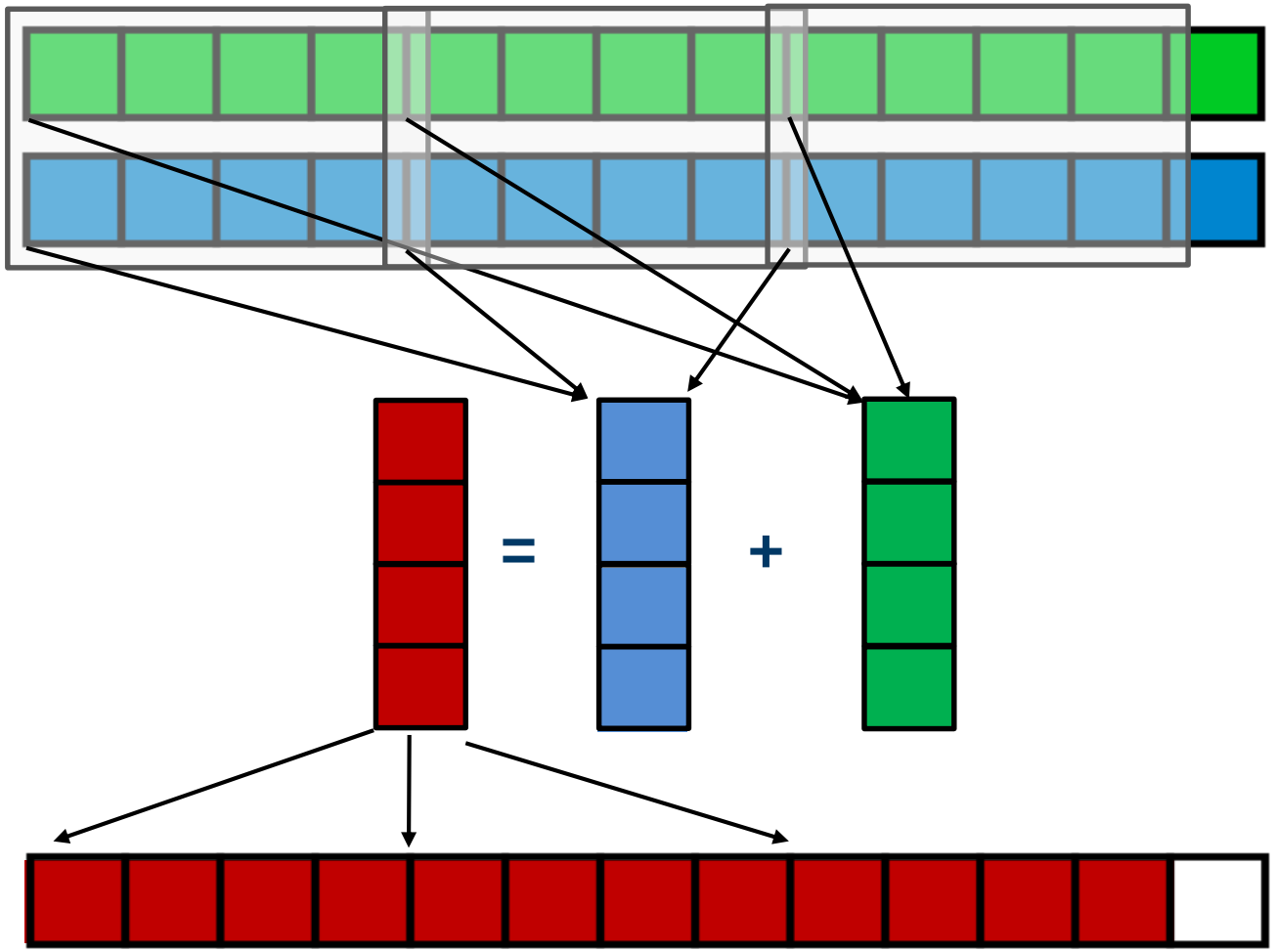
- 1 operand

- 2 operands (SSE)

- 4 operands (AVX)

- 8 operands (AVX512)

SIMD execution



SIMD by compiler

Steps (done by the compiler) for "SIMD processing"

```
for(int i=0; i<n;i++)
        C[i]=A[i]+B[i];
```

"Loop unrolling"

```
for(int i=0; i<n;i+=4){
        C[i]  =A[i]  +B[i];
        C[i+1]=A[i+1]+B[i+1];
        C[i+2]=A[i+2]+B[i+2];
        C[i+3]=A[i+3]+B[i+3];}
//remainder loop omitted
```

This should not be done by hand!

Load 256 bits starting from address of `A[i]` to register `xmm0`

Add the corresponding 64-bit entries in `xmm0` and `xmm1` and put the 4 results into `xmm2`

Store `xmm2` (256 bit) to address starting at `C[i]`

```
LABEL1:
        vmovupd xmm0 ← A[i]
        vmovupd xmm1 ← B[i]
        vaddpd  xmm0,xmm1 → xmm2
        vmovupd xmm2 → C[i]
        inc i,4
        cmp i,N
        jb LABEL1
//remainder loop omitted
```

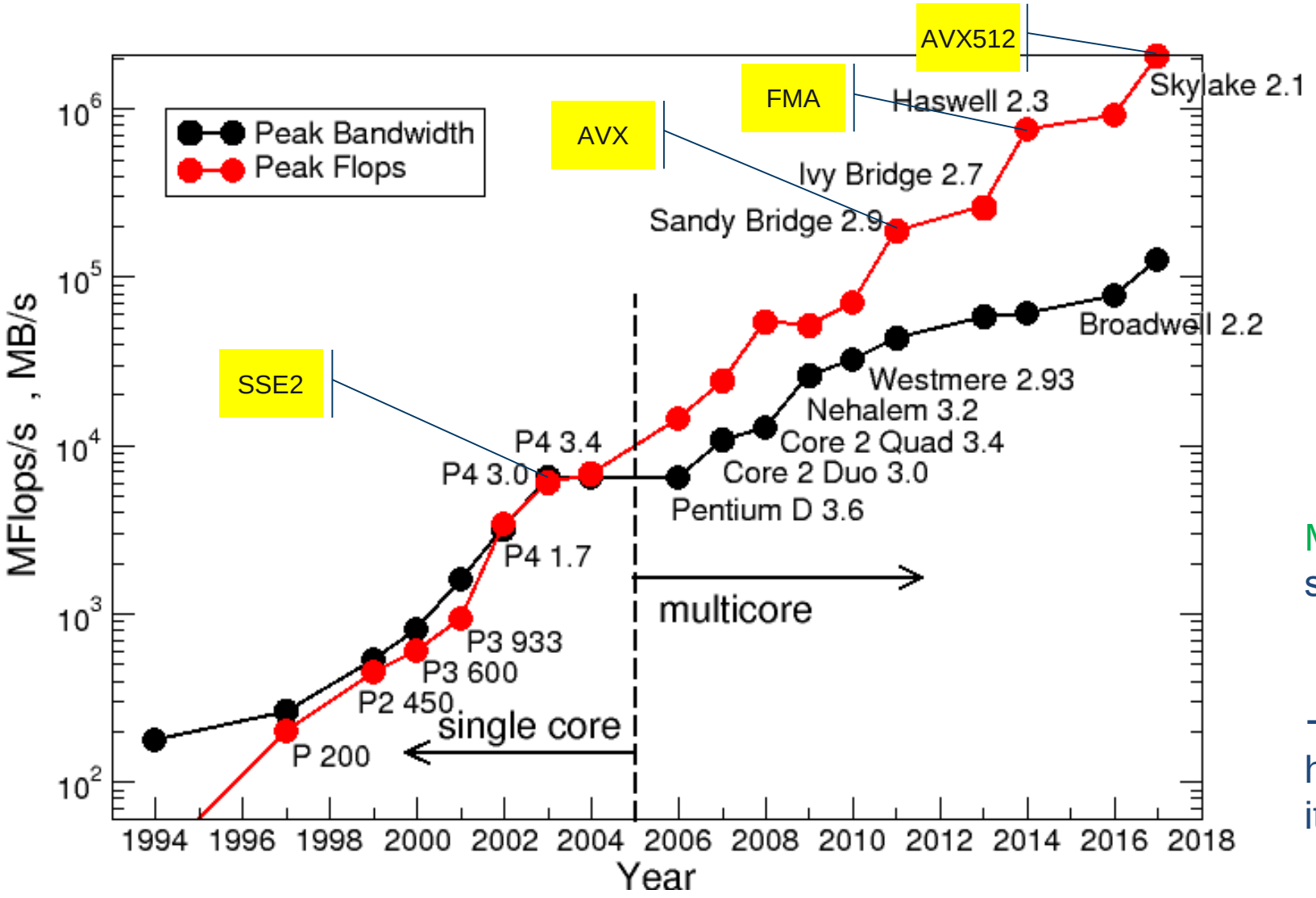
NHR FAU

FAU Friedrich-Alexander-Universität Erlangen-Nürnberg

Memory Hierarchy

The “DRAM gap”

DP peak performance and peak main memory bandwidth for a single Intel processor (chip)



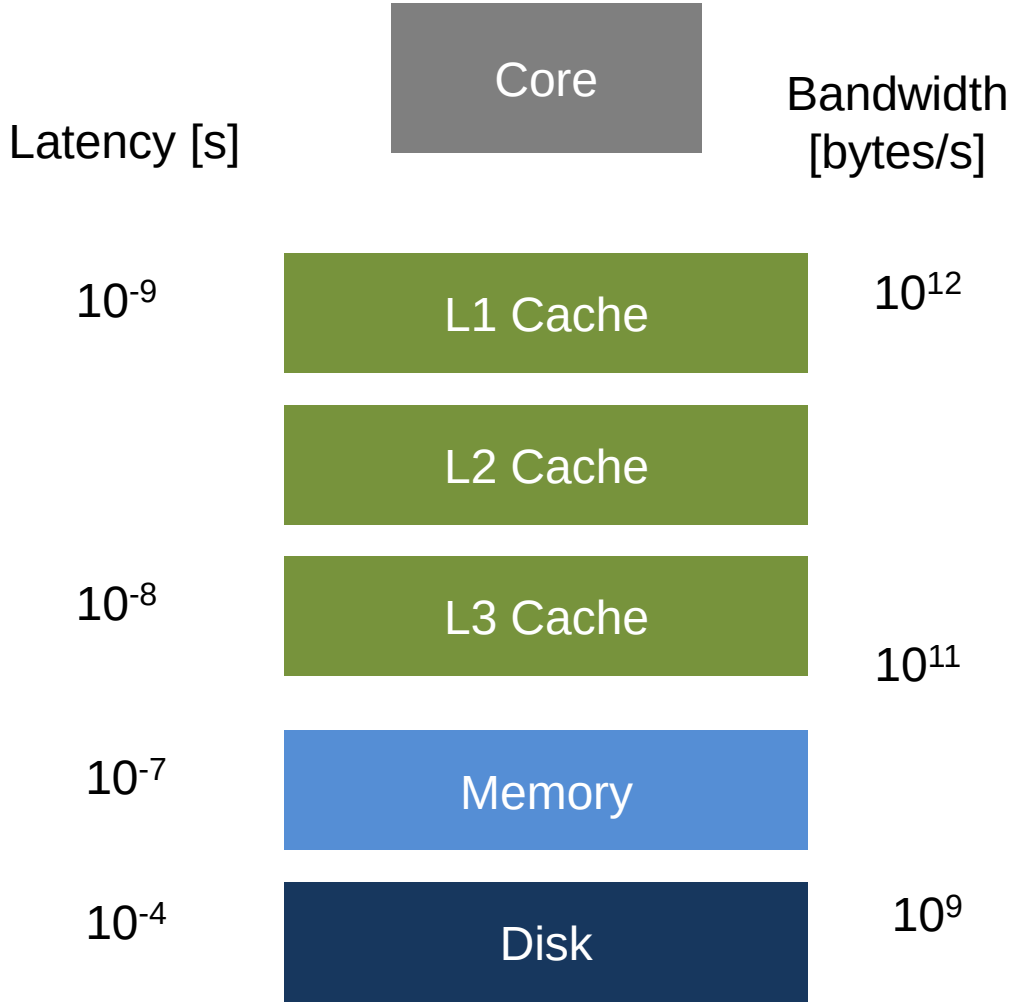
Approx.
15 F/B

Main memory access speed not sufficient to keep CPU busy…

→ Introduce fast on-chip caches, holding copies of recently used data items

Memory hierarchy

You can either build a ***small*** and ***fast*** memory or a ***large*** and ***slow*** memory.

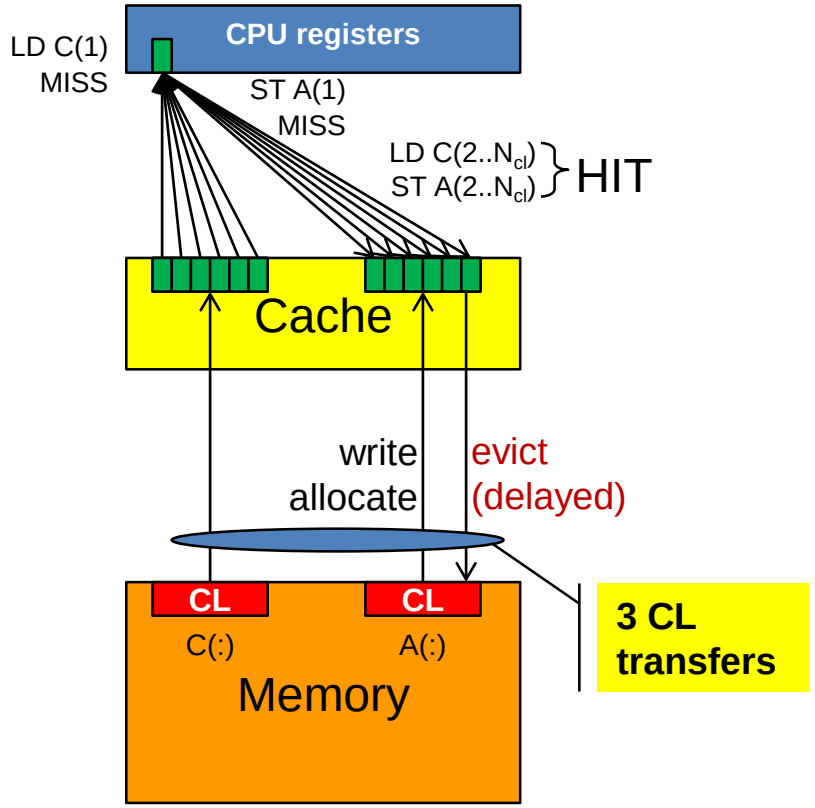


Purpose of many optimizations is to load data from fast memory layers.

Data transfers in a memory hiararchy

Caches help with getting instructions and data to the CPU “fast”

How does data travel from memory to the CPU and back?



- Remember: Caches are organized in cache lines (e.g., 64 bytes)
- Only complete cache lines are transferred between memory hierarchy levels (except registers)
- Registers can only “talk” to the L1 cache
- MISS: Load or store instruction does not find the data in acache level
  → CL transfer required

- Example: Array copy `A(:)=C(:)`

The parallel vector triad benchmark
A "swiss army knife" for microbenchmarking

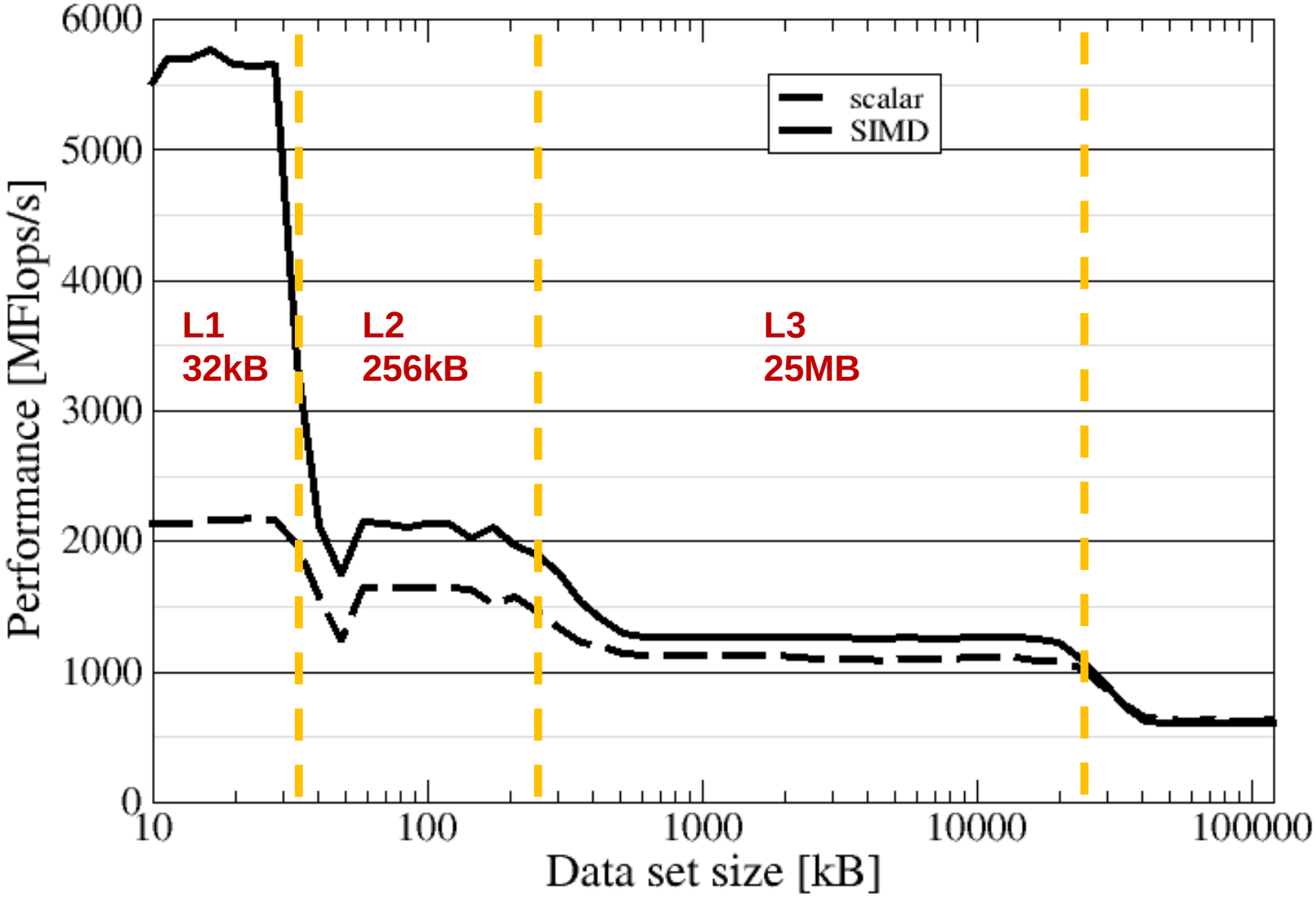
```
double striad_seq(double* restrict a, double* restrict b,
double* restrict c, double* restrict d, int N, int iter) {
    double S, E;
    S = getTimeStamp();
    for(int j = 0; j < iter; j++) {
    #pragma vector aligned
        for (int i = 0; i < N; i++) {
            a[i] = b[i] + d[i] * c[i];
        }
        if (a[N/2] > 2000) printf("Ai = %f\n",a[N/2]);
    }
    E = getTimeStamp();
    return E-S;
}
```

Required to get optimal code with Intel compiler!

Prevents smarty-pants compilers from doing "clever" stuff

1. **Report performance for different `N`, choose `iter` so that accurate time measurement is possible**
2. **This kernel is limited by data transfer performance for all memory levels on all architectures, ever!**

Schönauer triad on **one** IvyBridge **core** 2.2GHz



NHR FAU

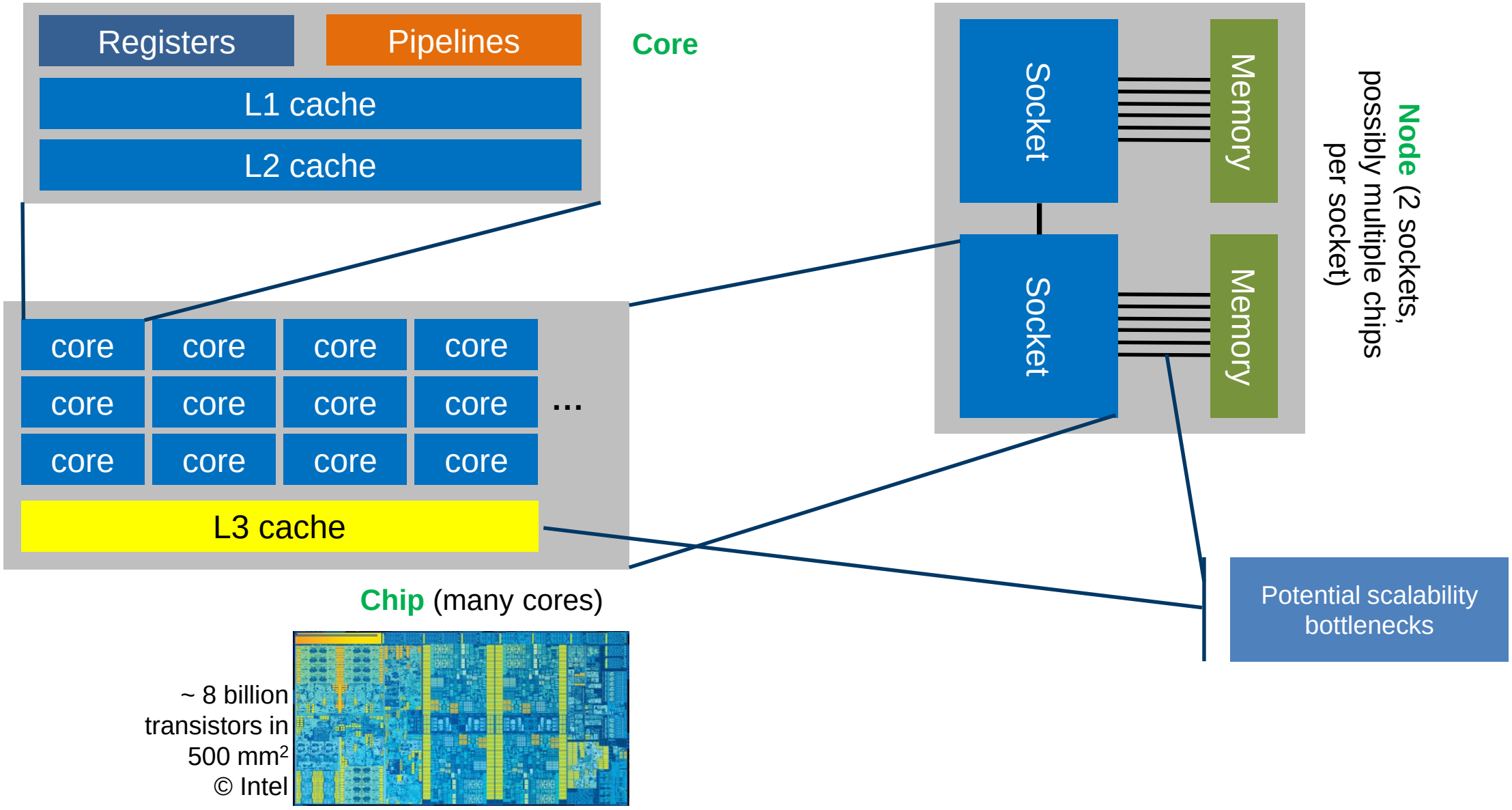
FAU Friedrich-Alexander-Universität Erlangen-Nürnberg

Multicore Chips

Memory bandwidth scaling

Node topology and performance

Node topology of HPC systems



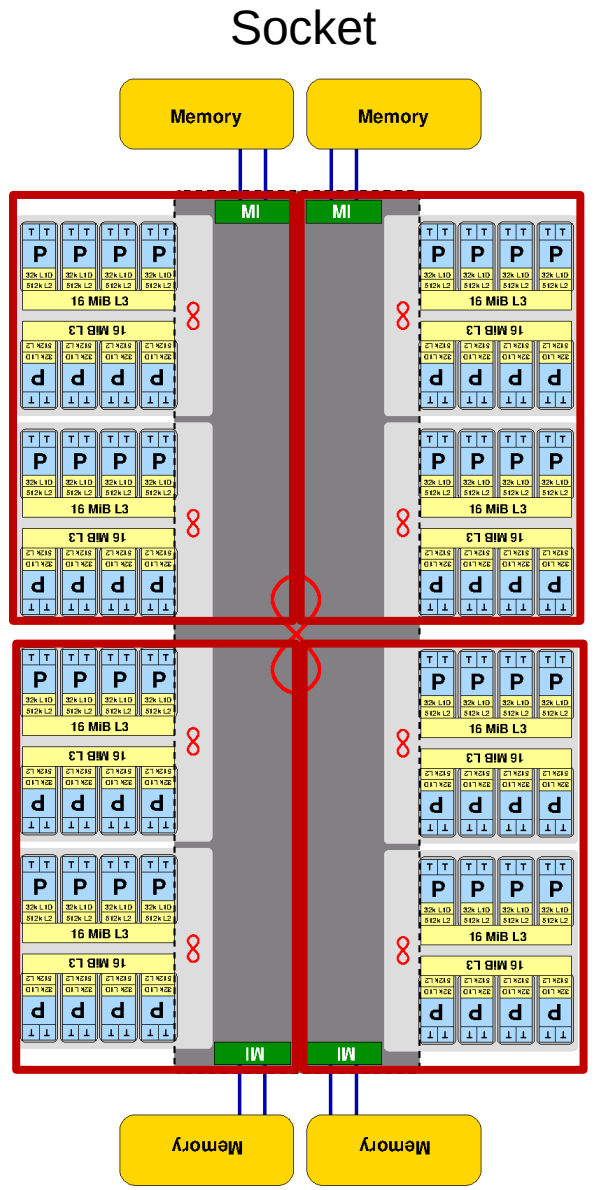
Putting the cores & caches together
AMD Epyc 7742 64-Core Processor («Rome»)

- Core features:
  - Two-way SMT
  - Two 256-bit SIMD FMA units (AVX2)
    →16 flops/cycle
  - 32 KiB L1 data cache per core
  - 512 KiB L2 cache per core

- 64 cores per socket hierarchically built up from
  - 16 CCX with 4 cores and 16 MiB of L3 cache
  - 2 CCX form 1 CCD (silicon die)
  - 8 CCDs connected to IO device “Infinity Fabric” (memory controller & PCIe)

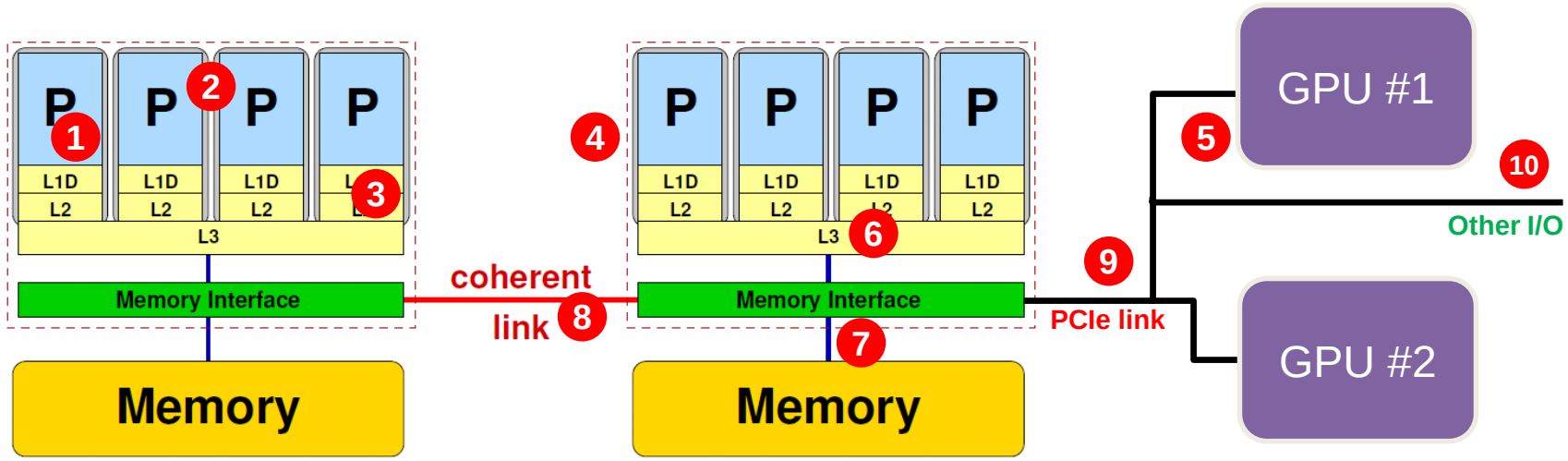
- 8 channels of DDR4-3200 per IO device
  - MemBW: 8 ch x 8 byte x 3.2 GHz = 204.8 GB/s

- ccNUMA-feature (Boot time option):
  - Node Per Socket (NPS)=1 , 2 or 4
  - **NPS=4 → 4 ccNUMA domains**



Parallelism in a modern compute node

Parallel and shared resources within a shared-memory node



Parallel resources:

- Execution/SIMD units 1
- Cores 2
- Inner cache levels 3
- Sockets / ccNUMA domains 4
- Multiple accelerators 5

Shared resources:

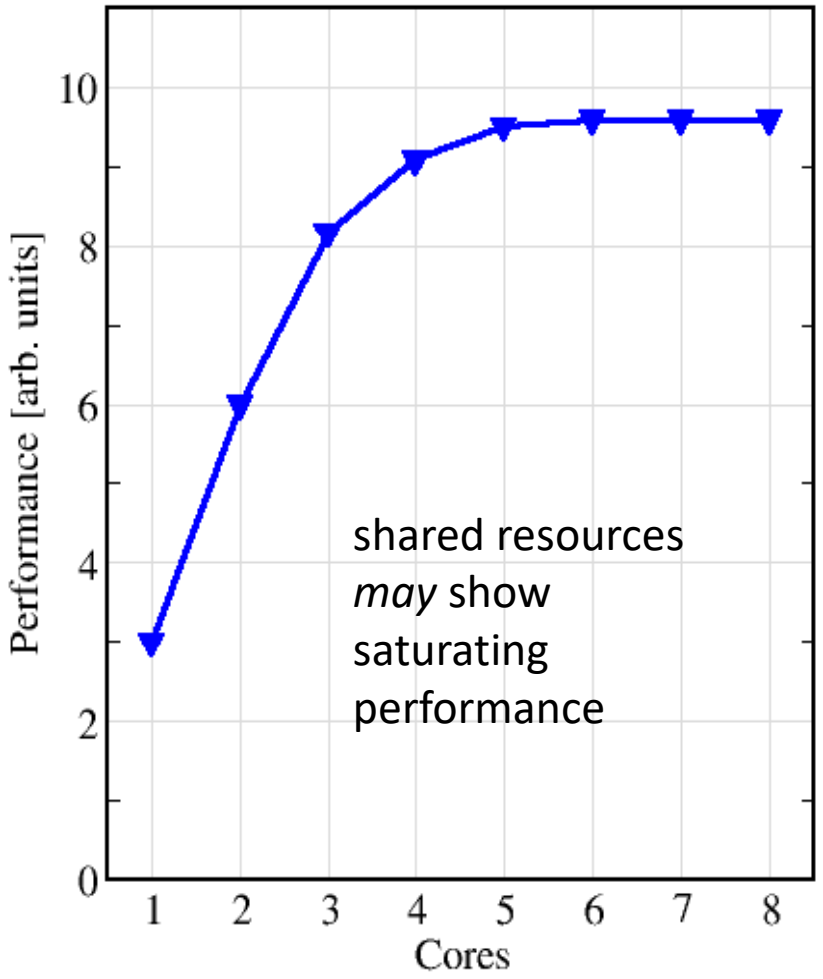
- Outer cache level per socket 6
- Memory bus per socket 7
- Intersocket link 8
- PCIe bus(es) 9
- Other I/O resources 10

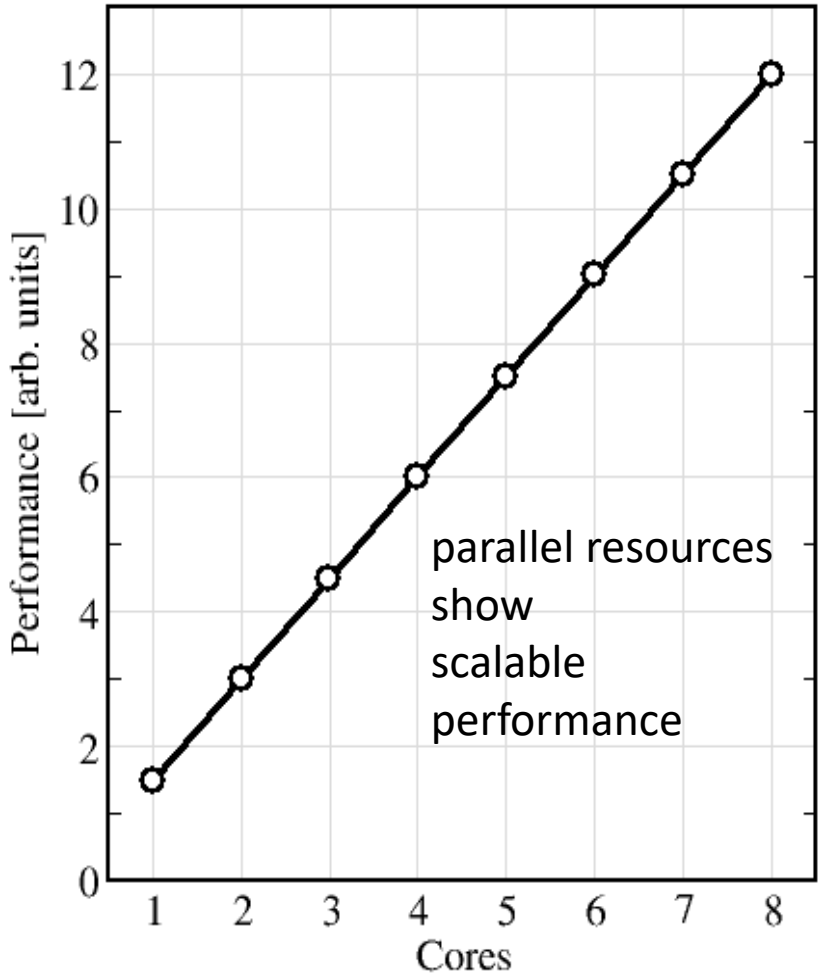
How does your application react to all of those details?

Scalable and saturating behavior

Clearly distinguish between "**saturating**" and "**scalable**" performance on the chip level

One of the most important performance signatures





Conclusions about architecture

- Performance is a result of
  - How **many instructions** you require to implement an algorithm
  - How **efficiently** those instructions are **executed** on a processor
  - Runtime contribution of the triggered **data transfers**

- Modern computer architecture has a rich “topology”

- Node-level hardware parallelism takes many forms
  - Sockets/devices – CPU: 1-4 or more
  - Cores – moderate/large (CPU: 4-64)
  - SIMD – moderate (CPU: 2-16)
  - Superscalarity (CPU: 2-6)

- Performance of programs is sensitive to architecture
  - “Performance portability” is a tough goal to achieve