What is Vectorization?

Georg Zitzlsberger

georg.zitzlsberger@vsb.cz

IT4Innovations
national supercomputing
center

5th of July 2017

Agenda

- What is "Vectorization"?
- History of SIMD
- How does SIMD work?
- SIMD for Intel Architecture
- SIMD Instructions (in a nutshell)
- Where is SIMD?

What is "Vectorization"?

What is a "vector"?

- This: $\vec{x} = \begin{bmatrix} x_1 & x_2 & \dots & x_m \end{bmatrix}^T$?
- This: `std::vector<int> x`?
- This: $x(:)$?
- This: `%ymm`?
- ...

Depending on your background your perception might vary...
But two properties are common to all:

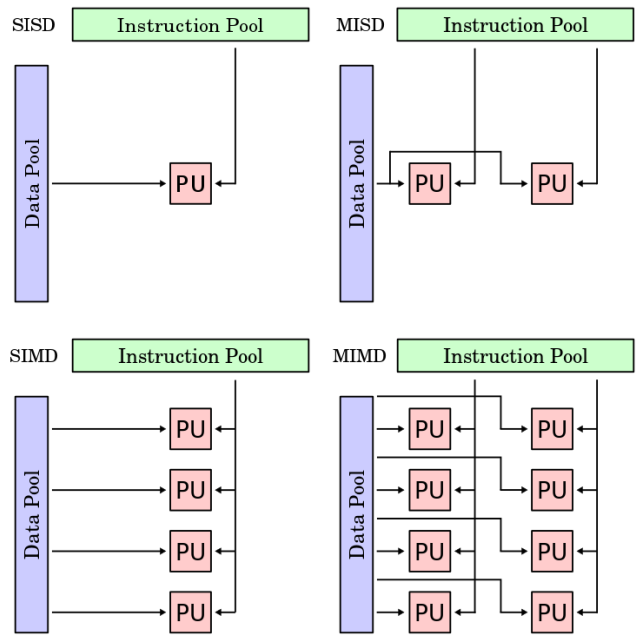
- Cover multiple data items
- Allow transformations/operations

Isn't that related to parallelism somehow?

Flynn's Taxonomy

Classifying parallelism since 1966:

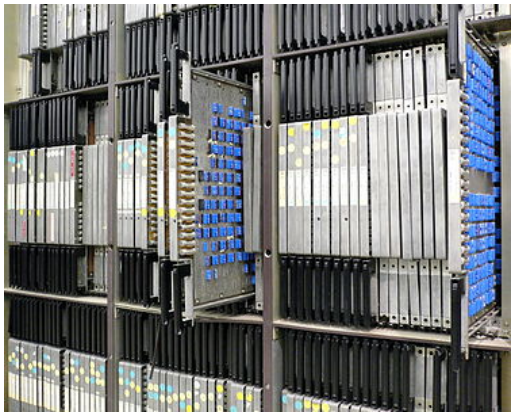
- Single data stream:
  - SISD:
    Single Instruction Single Data
  - MISD:
    Multiple Instruction Single Data
- Multiple data streams:
  - **SIMD:**
    Single Instruction Multiple Data
  - MIMD:
    Multiple Instruction Multiple Data
  - SPMD:
    Single Program Multiple Data
  - MPMD:
    Multiple Program Multiple Data

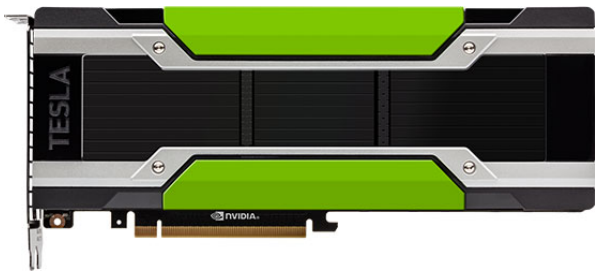


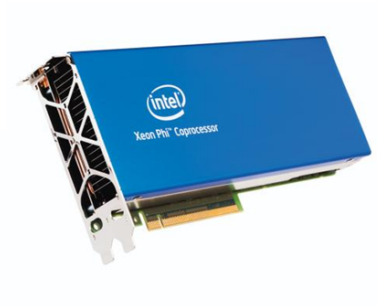
(Images: Wikipedia)

History of SIMD

- Idea goes back to 1960s with ILLIAC IV 1972 ($\sim 150 MFLOPS$)
- Cray dramatically improved and was renown for that for a long time (Cray-1 1976 $\sim 240 MFLOPS$)
- Nowadays it's pervasive and delivers $\sim 1 - 20 + TFLOPS$ per node





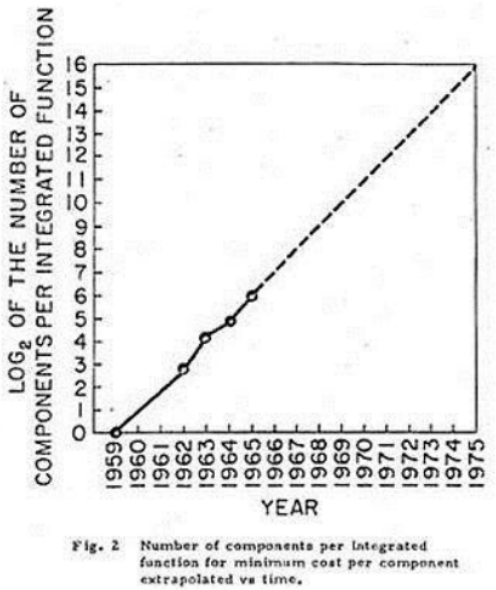


(Images: Wikipedia, Intel & NVidia)

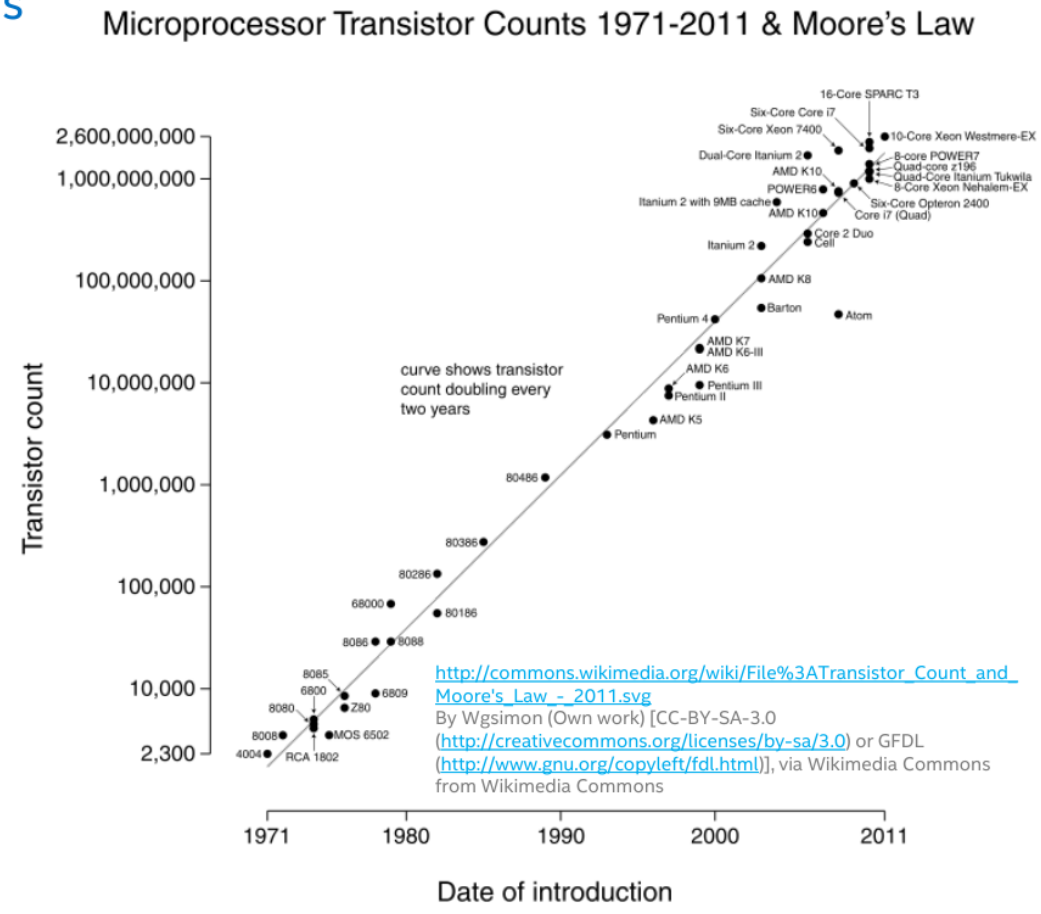
Moore's "Law"

"The number of transistors on a chip will **double** approximately **every two years.**"

[Gordon Moore]



Moore's Law graph, 1965



http://commons.wikimedia.org/wiki/File%3ATransistor_Count_and_Moore's_Law_-_2011.svg
By Wgsimon (Own work) [CC-BY-SA-3.0 (http://creativecommons.org/licenses/by-sa/3.0) or GFDL (http://www.gnu.org/copyleft/fdl.html)], via Wikimedia Commons
from Wikimedia Commons

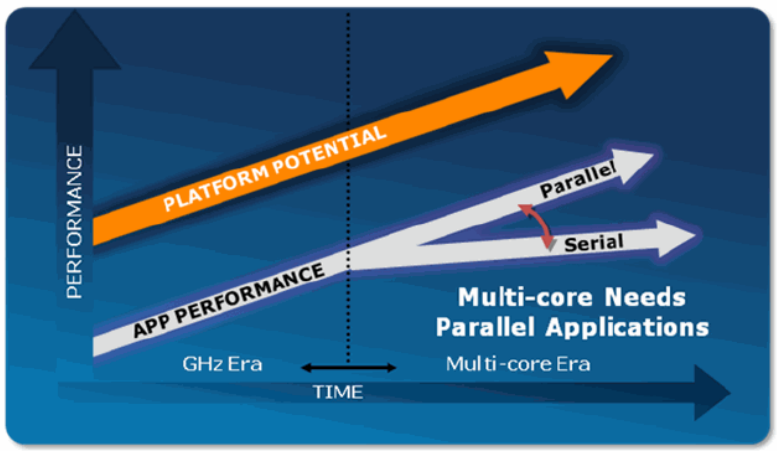
(Image: Intel)

Parallelism

Problem:
Economical operation **frequency of (CMOS) transistors is limited**.
⇨ **No free lunch anymore!**

Solution:
More transistors allow more gates/logic on the same die space and power envelop, **improving parallelism**:

- **Thread level** parallelism (TLP):
  Multi- and many-core
- **Data level** parallelism (DLP):
  Wider vectors (SIMD)
- **Instruction level** parallelism (ILP):
  Microarchitecture improvements, e.g. threading, superscalarity, ...

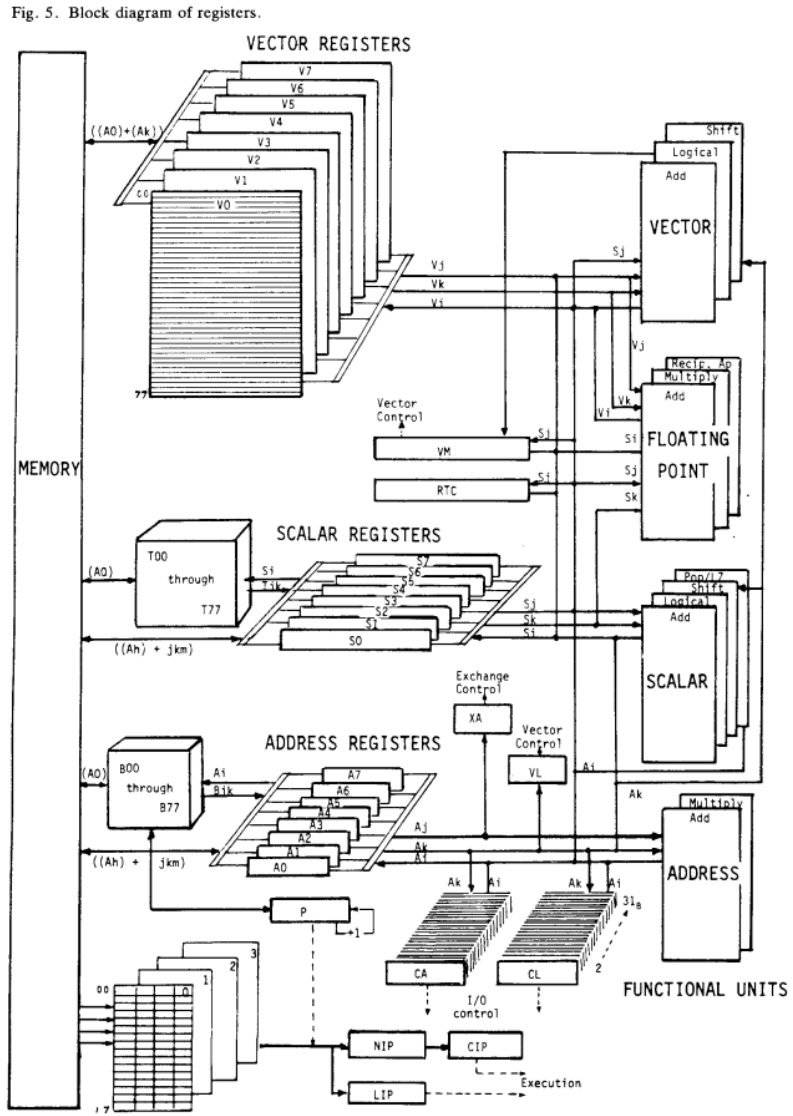


(Image: Intel)

How does SIMD work?

Depends on the architecture...

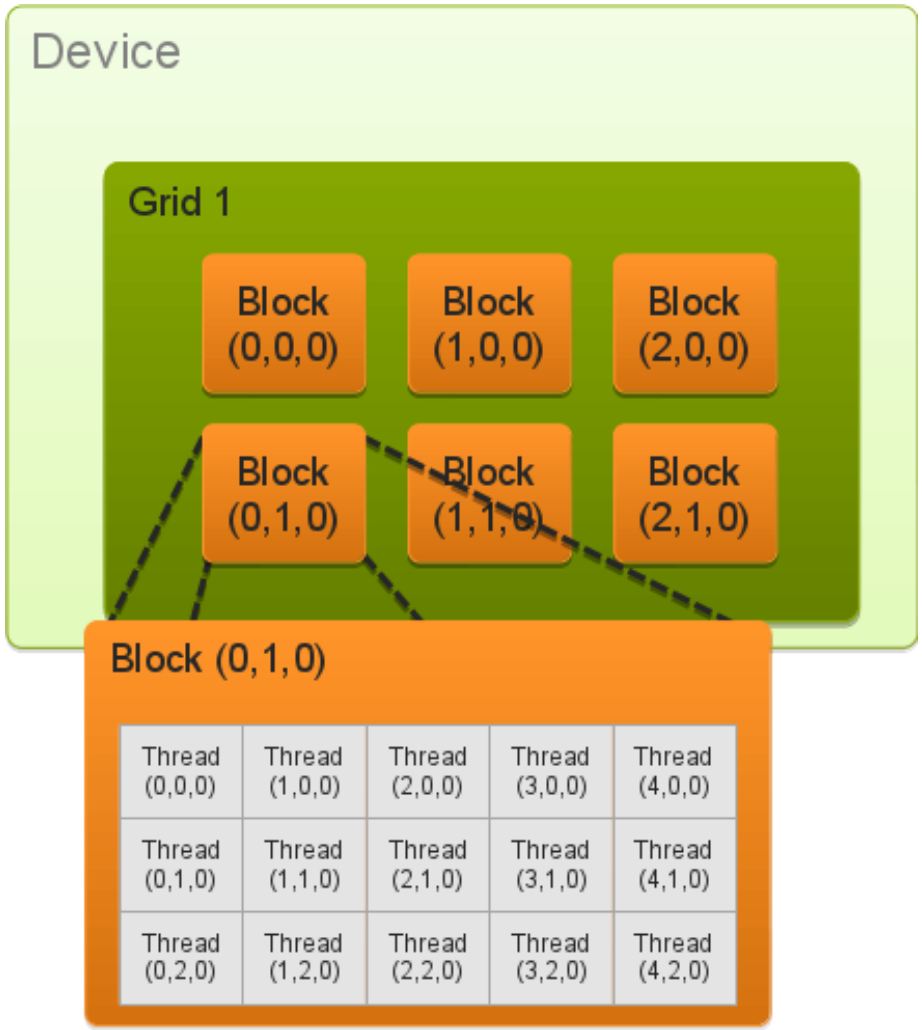
- Cray-1:



(Image: Cray)

How does SIMD work?

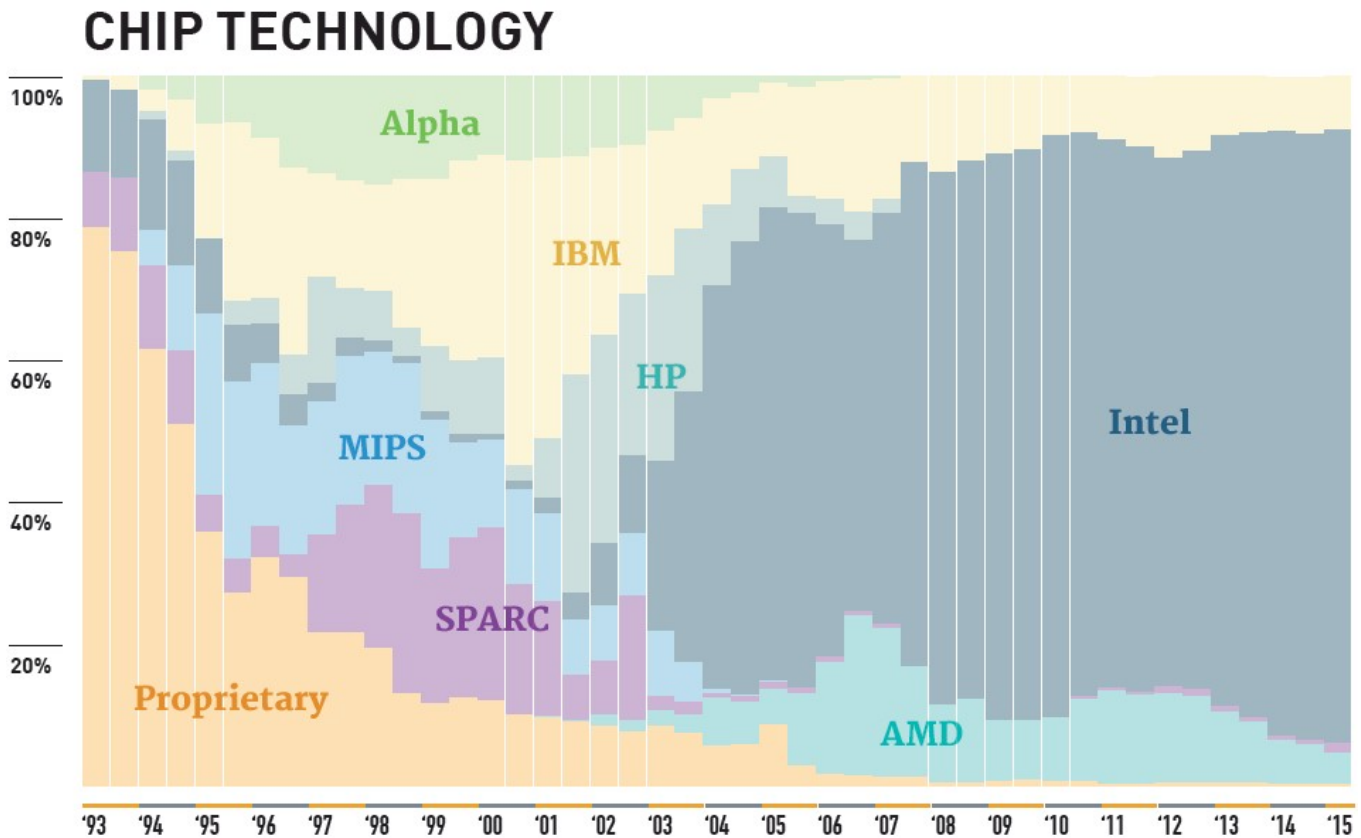
- GPUs (e.g. NVidia):



(Image: NVidia)

How does SIMD work?

- Intel Architecture: In more detail in the following...
  We focus on Intel Architectures used on the $> 80\%$ of all supercomputers from TOP500.



(Image: nextplatform.com)

How does SIMD work?

There are other architectures too:

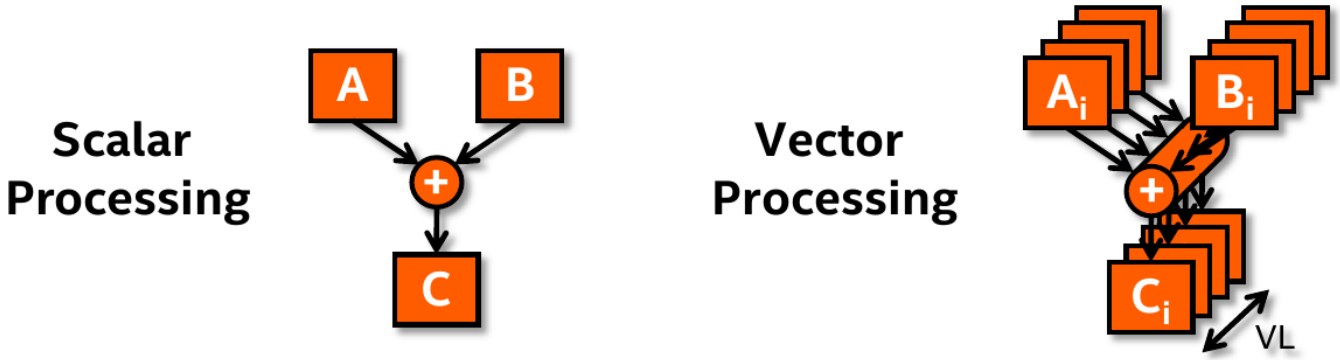
- IBM PowerPC: AltiVec

- ARM: NEON

...those work close to Intel Architecture we focus on throughout the day (but only have smaller 128 bit vectors).

SIMD for Intel Architecture

- SIMD:

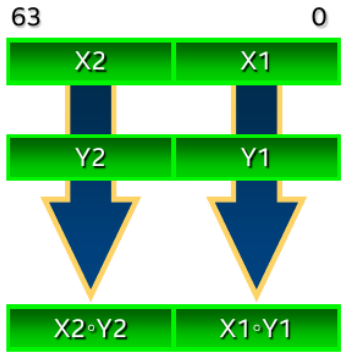


- Terminology:
  - SIMD: Data Level Parallelism (DLP)
  - Vector: Consists of more than one element
  - Elements of Vector: All of same primitive type (e.g. float, integer, . . .)
  - Vector Length (VL): Number of elements of a vector (subject to underlying type)
  - Vector Size: In bits (type-less)
  - SIMD lane: Data flow sliced by elements of a vector

History of SIMD Types for Intel Architecture

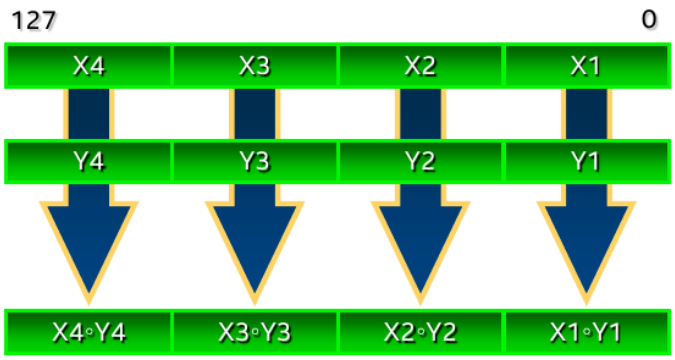
- 1996: MMX [+57 instructions]
- 1998: 3DNow! (AMD)
- 1999: Streaming SIMD Extensions (SSE) [+70 instructions]
  Only support for single & double precision FP
- 2001: SSE2 [+144 instructions]
  Added integer support
- 2004: SSE3 [+13 instructions]
- 2006: Supplemental SSE3 (SSSE3) [+16 instructions]
- 2006: SSE4.1 & 4.2 [+ 54 instructions]
- 2008: Advanced Vector Extensions (AVX)
  Only support for single & double precision FP
- 2011/2012: FMA4/FMA3 (AMD)
- 2012: Intel Many Integrated Core Architecture (MIC)
- 2013: AVX2 [+283 instructions]
  Added integer support + FMA for FP
- 2015: AVX-512

SIMD Types for Intel Architecture I



MMX™
Vector size: **64 bit**
Data types:
- 8, 16 and 32 bit integer

VL: 2, 4, 8



SSE
Vector size: **128 bit**
Data types:
- 8, 16, 32, 64 bit integer
- 32 and 64 bit float

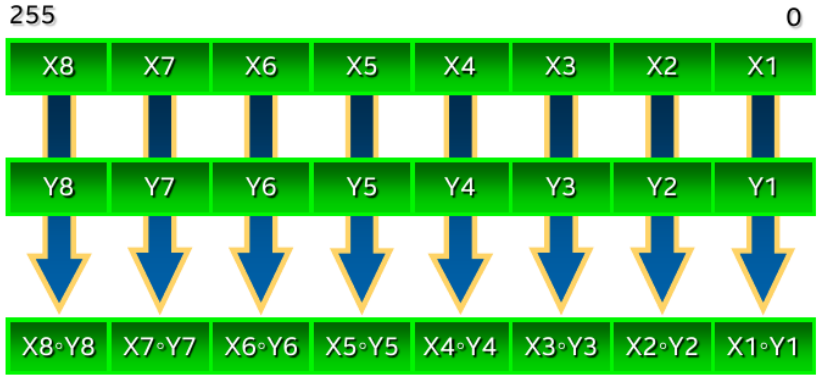
VL: 2, 4, 8, 16

Illustrations: Xi, Yi & results 32 bit integer

(Image: Intel)

SSE means all SSE and SSSE versions

SIMD Types for Intel Architecture II



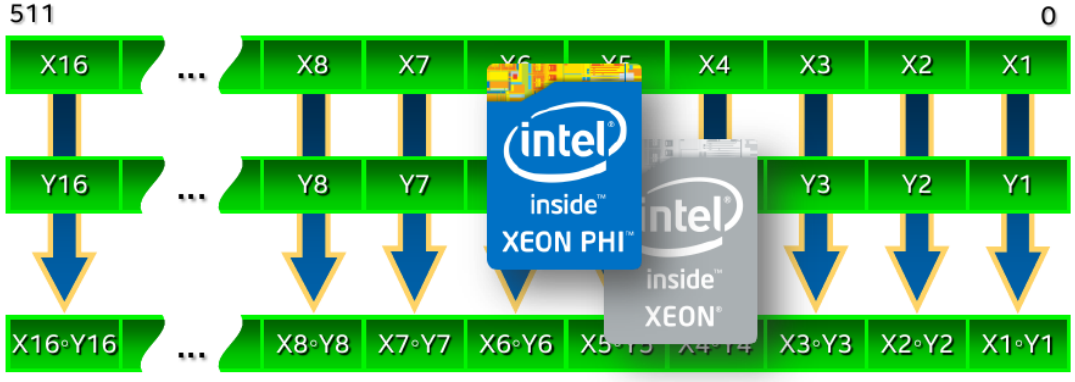
AVX

Vector size: **256 bit**

Data types:

- 8, 16, 32, 64 bit integer
- 32 and 64 bit float

VL: 4, 8, 16, 32



Intel® AVX-512 & Intel® MIC Architecture

Vector size: **512 bit**

Data types:

- 8, 16, 32, 64 bit integer
- 32 and 64 bit float

VL: 8, 16, 32, 64

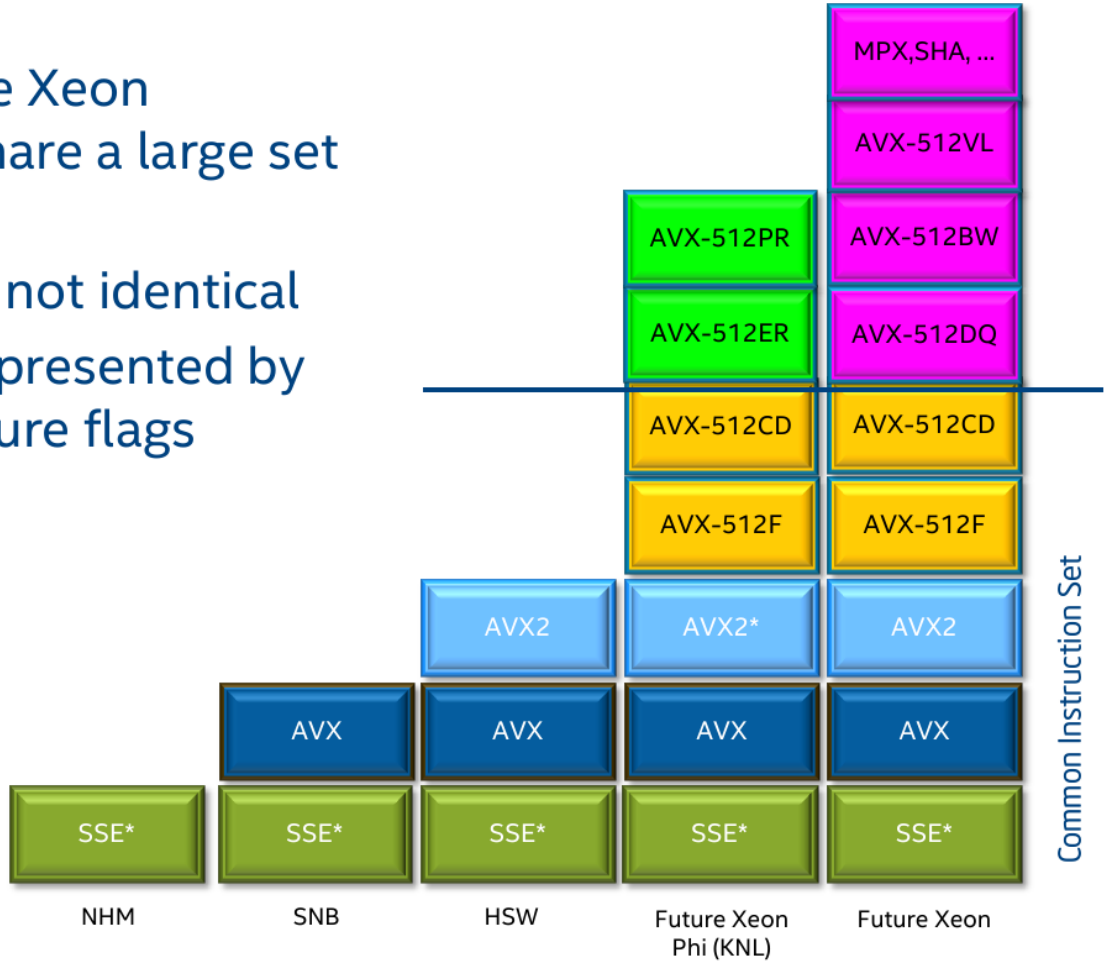
Illustrations: Xi, Yi & results 32 bit integer

(Image: Intel)

AVX means both AVX and AVX2

SIMD Types for Intel Architecture III

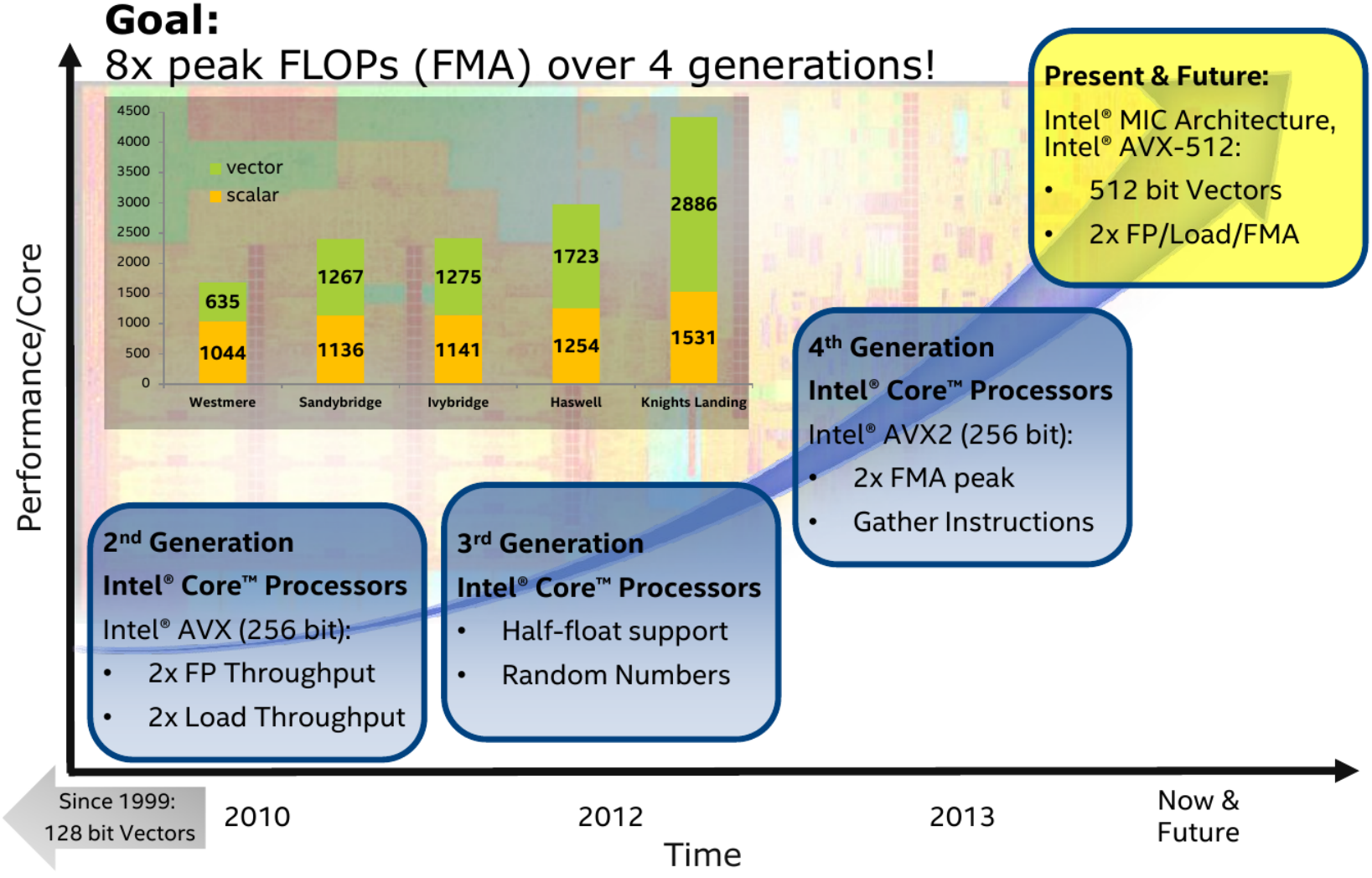
- KNL and future Xeon architecture share a large set of instructions
  - but sets are not identical
- Subsets are represented by individual feature flags (CPUID)



(Image: Intel)

Notice the different subsets!

Performance Evolution of SIMD for Intel Architect



(Image: Intel)

SIMD Instructions (in a nutshell)

Registers[1] depending on SIMD type:

- SSE 128 bit: *XMM0-15*
- AVX 256 bit: *YMM0-15*
- AVX-512 512 bit: *ZMM0-32* plus mask registers *k0-7*

511 256	255 128	127 0
ZMM0	YMM0	XMM0
ZMM1	YMM1	XMM1
ZMM2	YMM2	XMM2
ZMM3	YMM3	XMM3
ZMM4	YMM4	XMM4
ZMM5	YMM5	XMM5
ZMM6	YMM6	XMM6
ZMM7	YMM7	XMM7
ZMM8	YMM8	XMM8
ZMM9	YMM9	XMM9
ZMM10	YMM10	XMM10
ZMM11	YMM11	XMM11
ZMM12	YMM12	XMM12
ZMM13	YMM13	XMM13
ZMM14	YMM14	XMM14
ZMM15	YMM15	XMM15
ZMM16	YMM16	XMM16
ZMM17	YMM17	XMM17
ZMM18	YMM18	XMM18
ZMM19	YMM19	XMM19
ZMM20	YMM20	XMM20
ZMM21	YMM21	XMM21
ZMM22	YMM22	XMM22
ZMM23	YMM23	XMM23
ZMM24	YMM24	XMM24
ZMM25	YMM25	XMM25
ZMM26	YMM26	XMM26
ZMM27	YMM27	XMM27
ZMM28	YMM28	XMM28
ZMM29	YMM29	XMM29
ZMM30	YMM30	XMM30
ZMM31	YMM31	XMM31

[1] All limited to 8 for 32 bit mode

SIMD Instructions (in a nutshell)

SIMD instruction categorization[2]

E.g.: `vmovapd %ymm0, (%rdx,%rax)`

- Packed [p] or scalar [s] execution?
- Double [d] or single precision [s] floating point?

`vec.cpp`

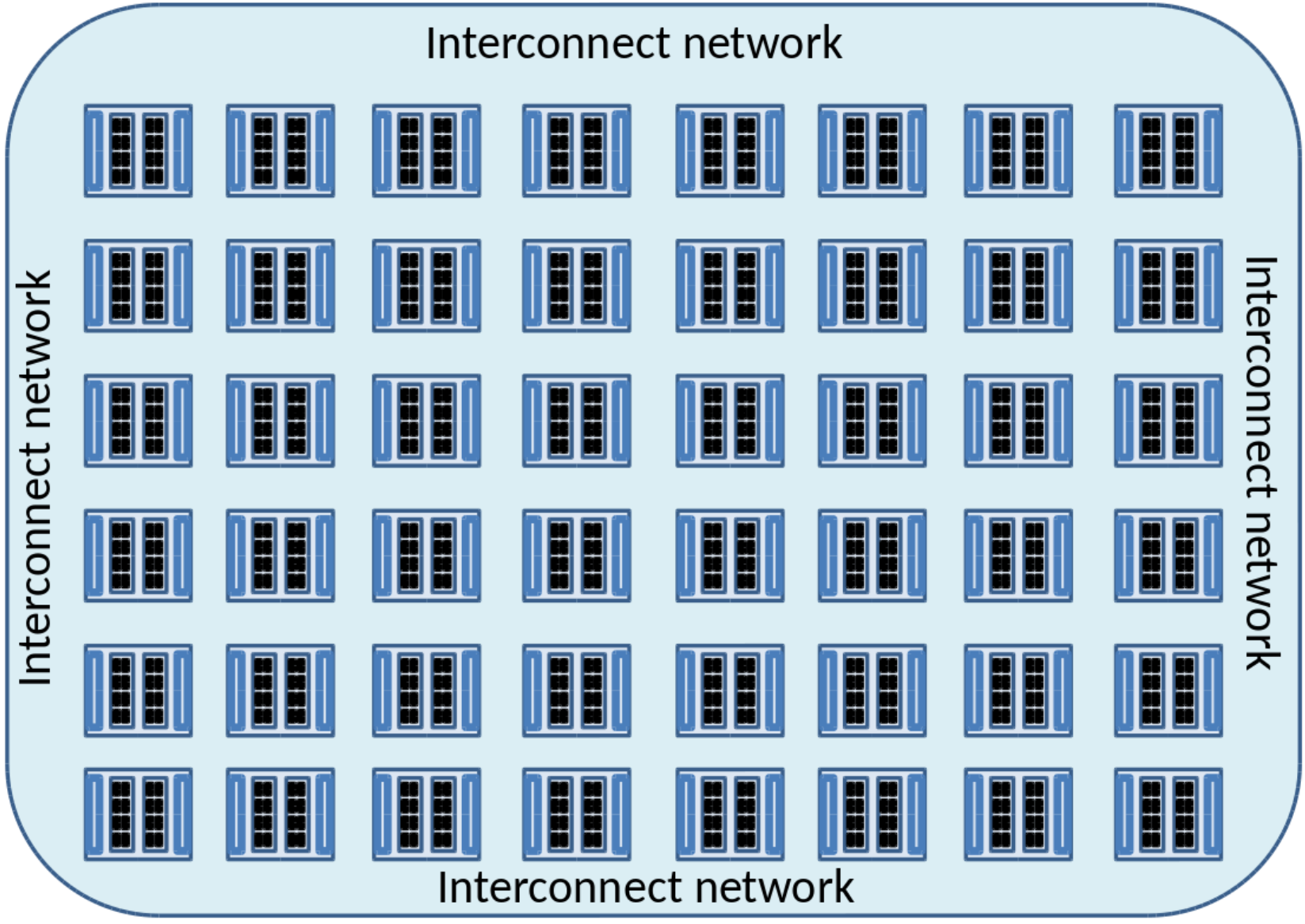
```
void vec(double *a, double *b,
         double * __restrict__ c)
{
    double *aa = (double *)
        __builtin_assume_aligned(a, 64);
    double *ba = (double *)
        __builtin_assume_aligned(b, 64);
    double *ca = (double *
        )__builtin_assume_aligned(c, 64);
    for(int i = 0; i < N; ++i) {
        ca[i] = aa[i] + ba[i];
    }
}
```

`vec.s` (GAS style)

```
.L4:
vmovapd (rdi,%rax), %ymm0
vaddpd (%rsi,%rax), %ymm0, %ymm0
vmovapd %ymm0, (%rdx,%rax)
addq $32, %rax
cmpq $80000, %rax
jne .L4
vzeroupper
ret
```

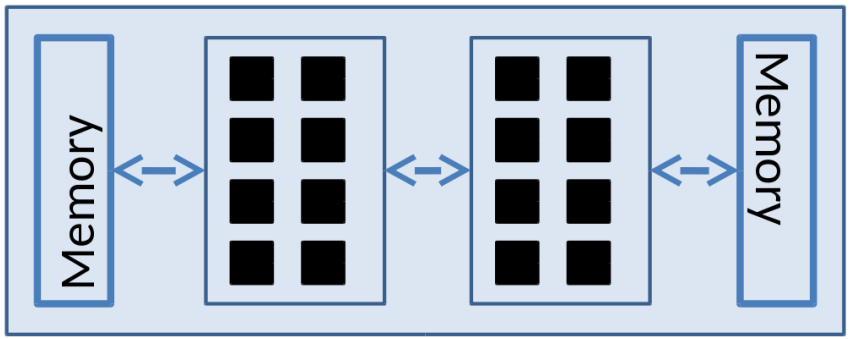
[2] Simplified - please refer to the processor manuals for full information

Where is SIMD?



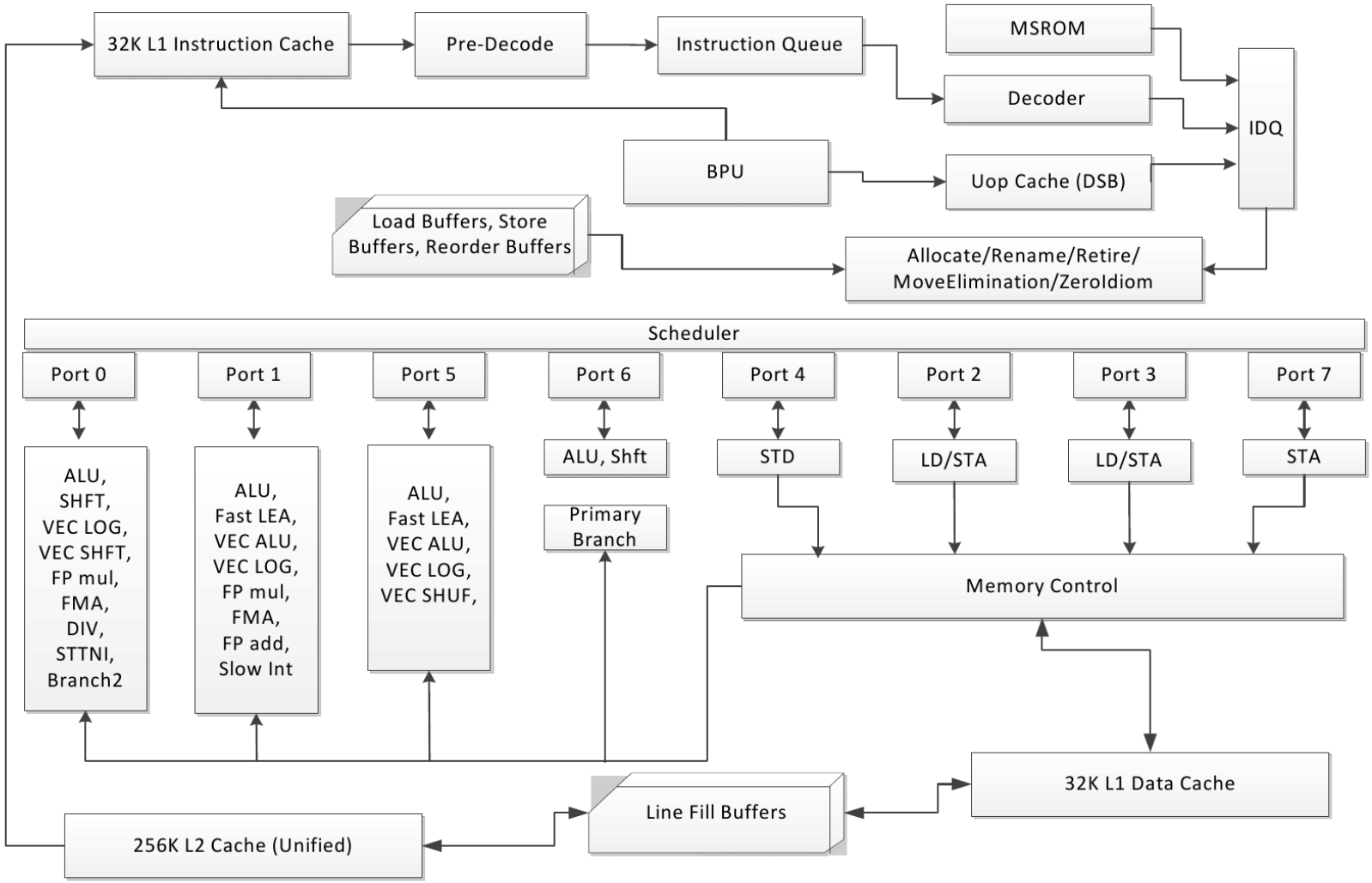
Where is SIMD?

Every core has SIMD capabilities:



- Every SW-Thread or MPI Process to use SIMD
- In a multicore environment SIMD will be influenced by memory (cache coherency)

Where is SIMD?



(Image: Intel)