# Master Informatics Eng.

2020/21

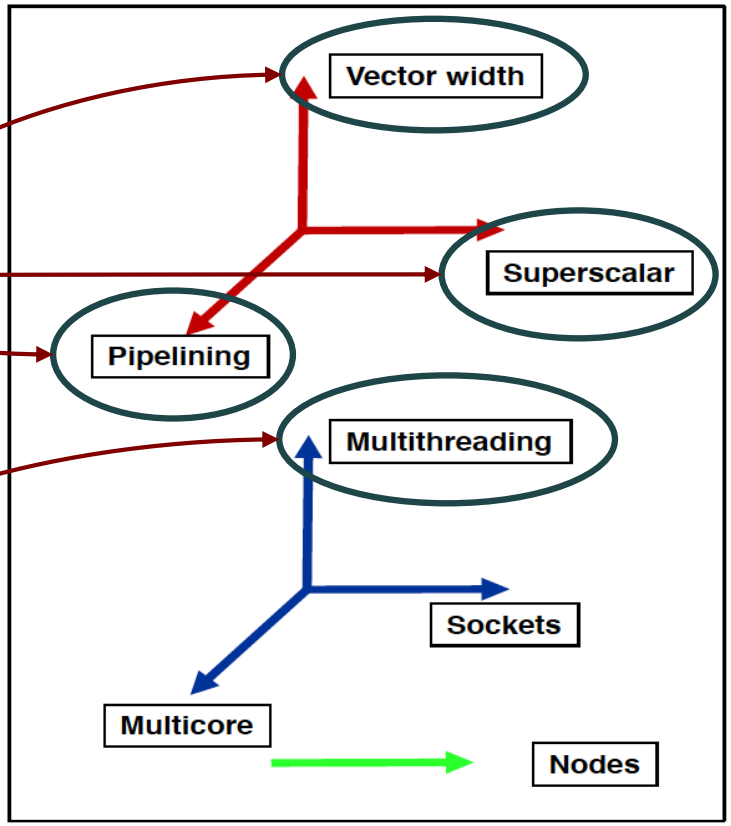
*A.J.Proença*

## SIMD extensions at Intel, AMD & ARM

***(some slides are borrowed)***

# *Key issues for parallelism in a single-core*

- **Currently under discussion:**
  - pipelining:
    reviewed in the `combine` example
  - superscalar:
    idem, but some more now
  - data parallelism:
    vector computers &
    vector extensions to scalar processors
  - multithreading:
    alternative approaches



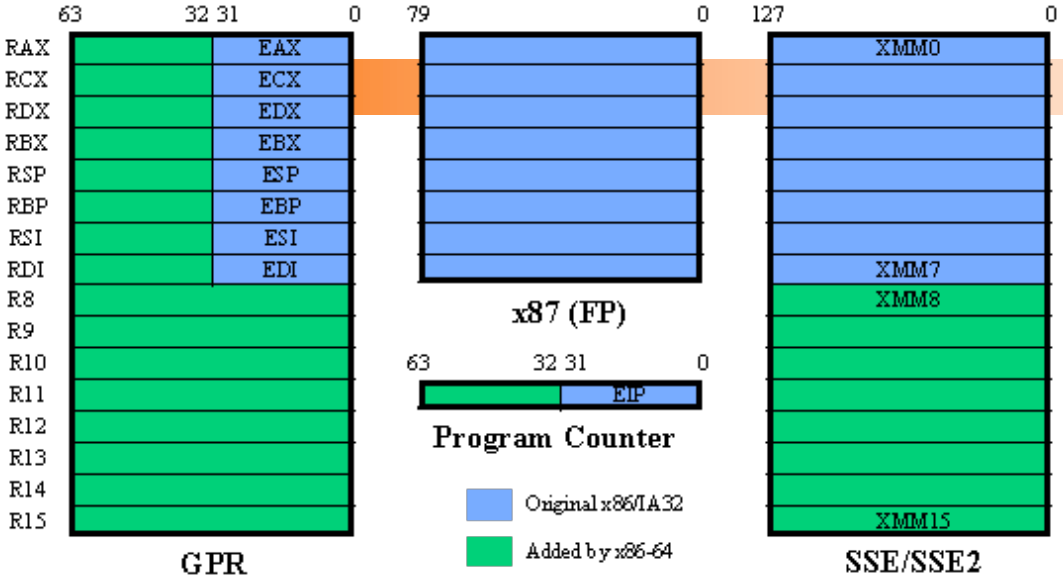
# SIMD Parallelism

- Vector architectures
  - Read sets of data elements *(gather from memory)* into "vector registers"
  - Operate on those registers
  - Store/scatter the results back into memory
- SIMD extensions to scalar architectures
  - Have limitations, compared to vector architectures (before AVX...)
  - Number of data operands encoded into op code
  - No sophisticated addressing modes (strided, scatter-gather)
  - No mask registers
- Graphics Processor Units (GPUs) *(next set of slides)*

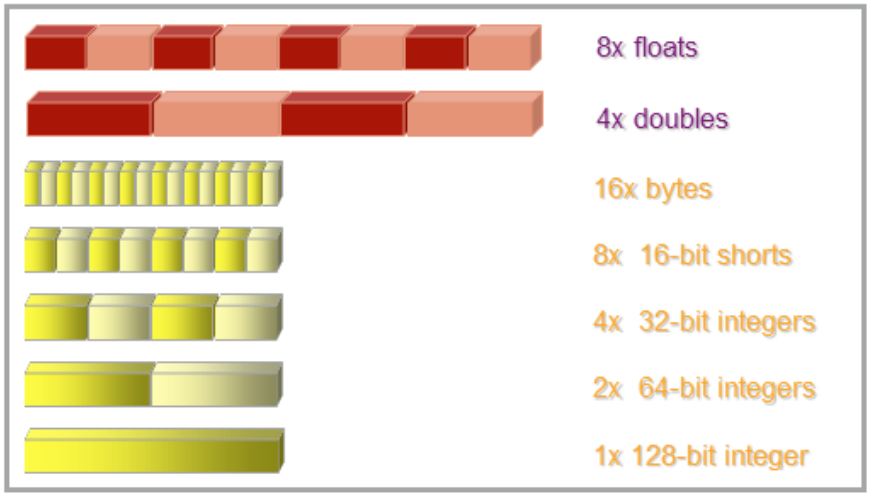
# SIMD Implementations

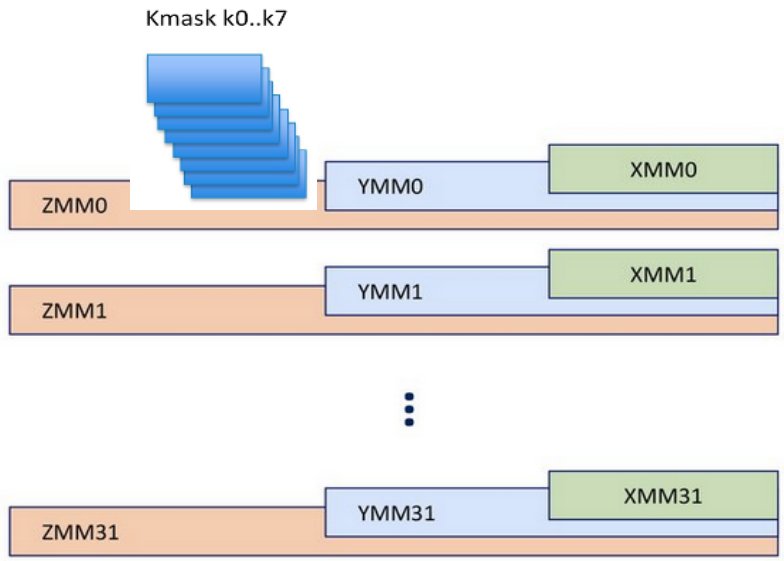
- Intel implementations:
  - MMX (1996)
    - Eight 8-bit integer ops or four 16-bit integer ops
  - Streaming SIMD Extensions (SSE) (1999)
    - Eight 16-bit integer ops
    - Four 32-bit integer/fp ops or two 64-bit integer/fp ops
  - Advanced Vector eXtensions (AVX) (2010...)
    - Eight 32-bit fp or four 64-bit fp ops (integers only in AVX-2)
    - 512-bits wide in AVX-512 (and also in Larrabee & Phi-KNC)
  - Operands **must / should be in consecutive and aligned** memory locations
- AMD Zen/Epyc (Opteron follow-up): up to AVX-2
- ARMv8 (64-bit) architecture: NEON & SVE

# Registers for vector processing in Intel 64

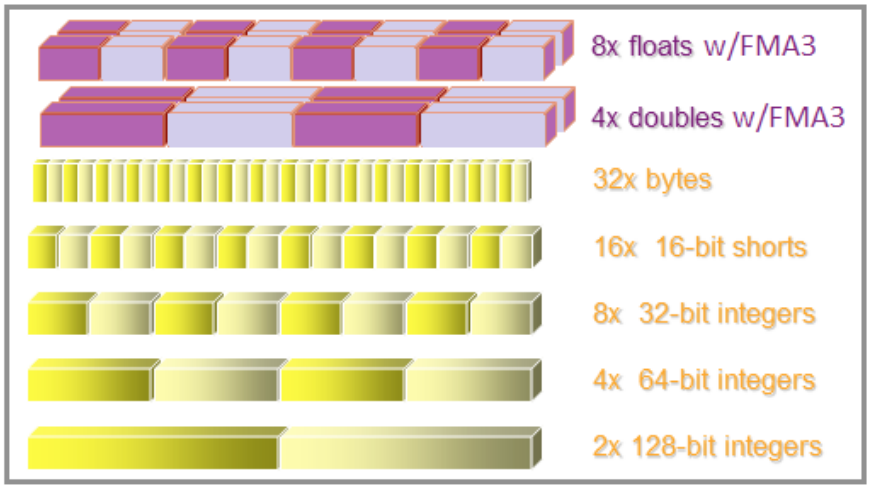



**Advanced Vector eXtensions, AVX 1.0**




- XMM0 – XMM15
  - 128-bit registers
  - SSE
- YMM0 – YMM15
  - 256-bit registers
  - AVX, AVX2
- ZMM0 – ZMM31
  - 512-bit registers
  - AVX-512




**AVX 2.0**

**AMD only supports AVX 2**

# Intel evolution to the AVX-512

## Intel® AVX Technology

256b AVX1
16 SP / 8 DP Flops/Cycle

256b AVX2
32 SP / 16 DP Flops/Cycle (FMA)

512b AVX-512
64SP / 32 DP Flops/Cycle (FMA)

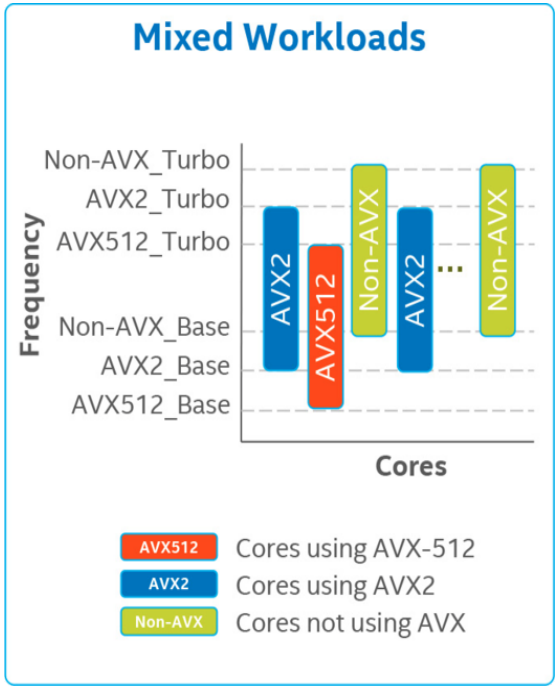
| AVX | AVX2 |
|---|---|
| 256-bit basic FP | Float16 (IVB 2012) |
| 16 registers | 256-bit FP FMA |
| NDS (and AVX128) | 256-bit integer |
| Improved blend | PERMD |
| MASKMOV | Gather |
| Implicit unaligned | |

**AVX-512**
- 512-bit FP/Integer
- 32 registers
- 8 mask registers
- Embedded rounding
- Embedded broadcast
- Scalar/SSE/AVX "promotions"
- HPC additions
- Transcendental support
- Gather/Scatter

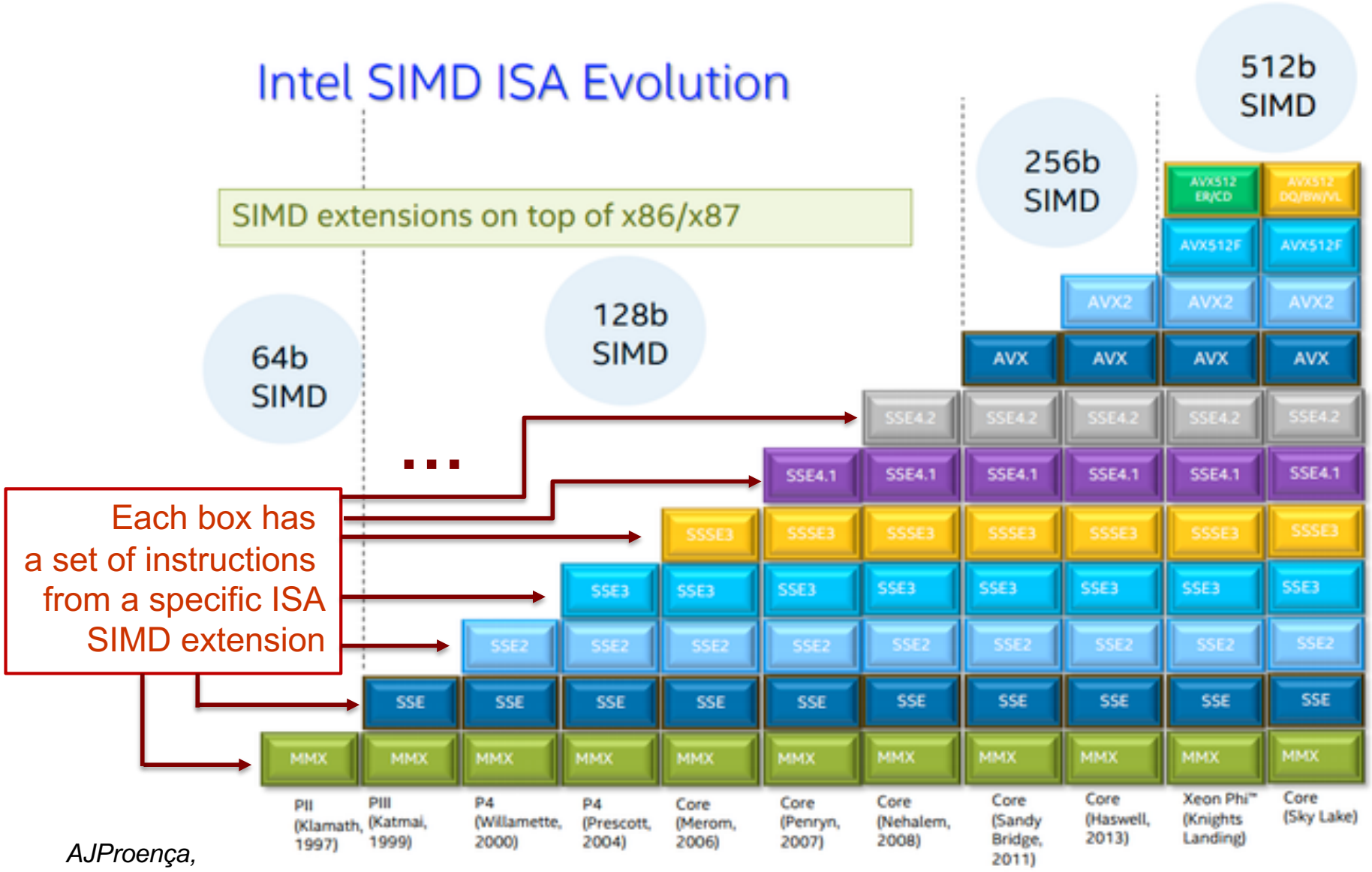
SNB 2011 — **Sandy Bridge**

HSW 2013 — **Haswell**

~~Future~~ Processors (KNL & SKX)
~~In planning, subject to change~~



# Intel SIMD ISA evolution



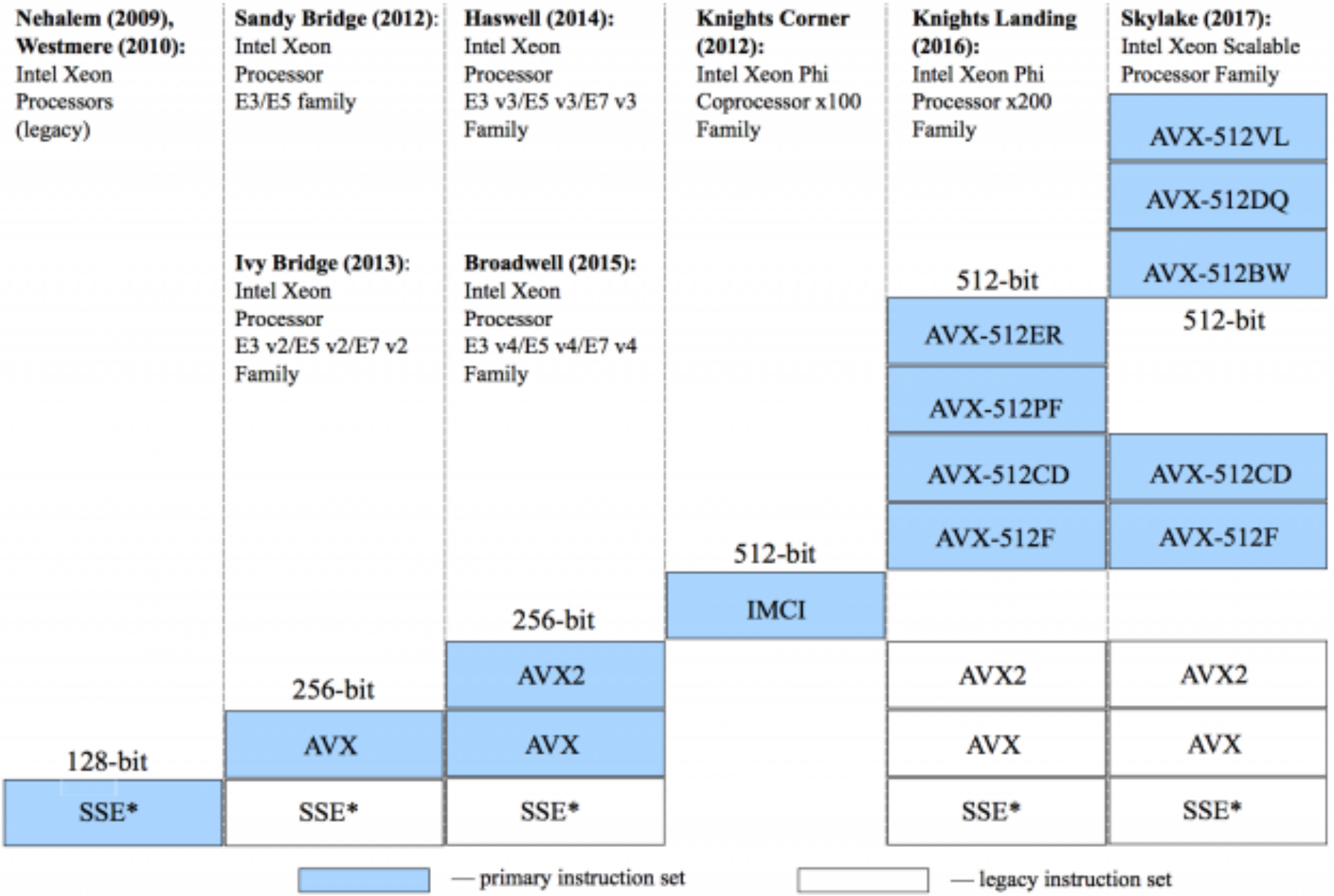
# The AVX-512 across Intel devices

| Microarchitecture | Support Level | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | F | CD | ER | PF | BW | DQ | VL | IFMA | VBMI | 4FMAPS | VNNI | 4VNNIW | VPOPCNTDQ | BF16 |
| Knights Landing | ✔ | ✔ | ✔ | ✔ | ✘ | ✘ | ✘ | ✘ | ✘ | ✘ | ✘ | ✘ | ✘ | ✘ |
| Knights Mill | ✔ | ✔ | ✔ | ✔ | ✘ | ✘ | ✘ | ✘ | ✘ | ✔ | ✔ | ✔ | ✔ | ✘ |
| Skylake (server) | ✔ | ✔ | ✘ | ✘ | ✔ | ✔ | ✔ | ✘ | ✘ | ✘ | ✘ | ✘ | ✘ | ✘ |
| Cascade Lake | ✔ | ✔ | ✘ | ✘ | ✔ | ✔ | ✔ | ✘ | ✘ | ✘ | ✔ | ✘ | ✘ | ✘ |
| Cannon Lake | ✔ | ✔ | ✘ | ✘ | ✔ | ✔ | ✔ | ✔ | ✔ | ✘ | | ✘ | ✘ | ✘ |
| Ice Lake (server) | ✔ | ✔ | ✘ | ✘ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✘ | ✘ | ✘ |
| Cooper Lake | ✔ | ✔ | ✘ | ✘ | ✔ | ✔ | ✔ | ✘ | ✘ | ✘ | ✔ | ✘ | ✘ | ✔ |

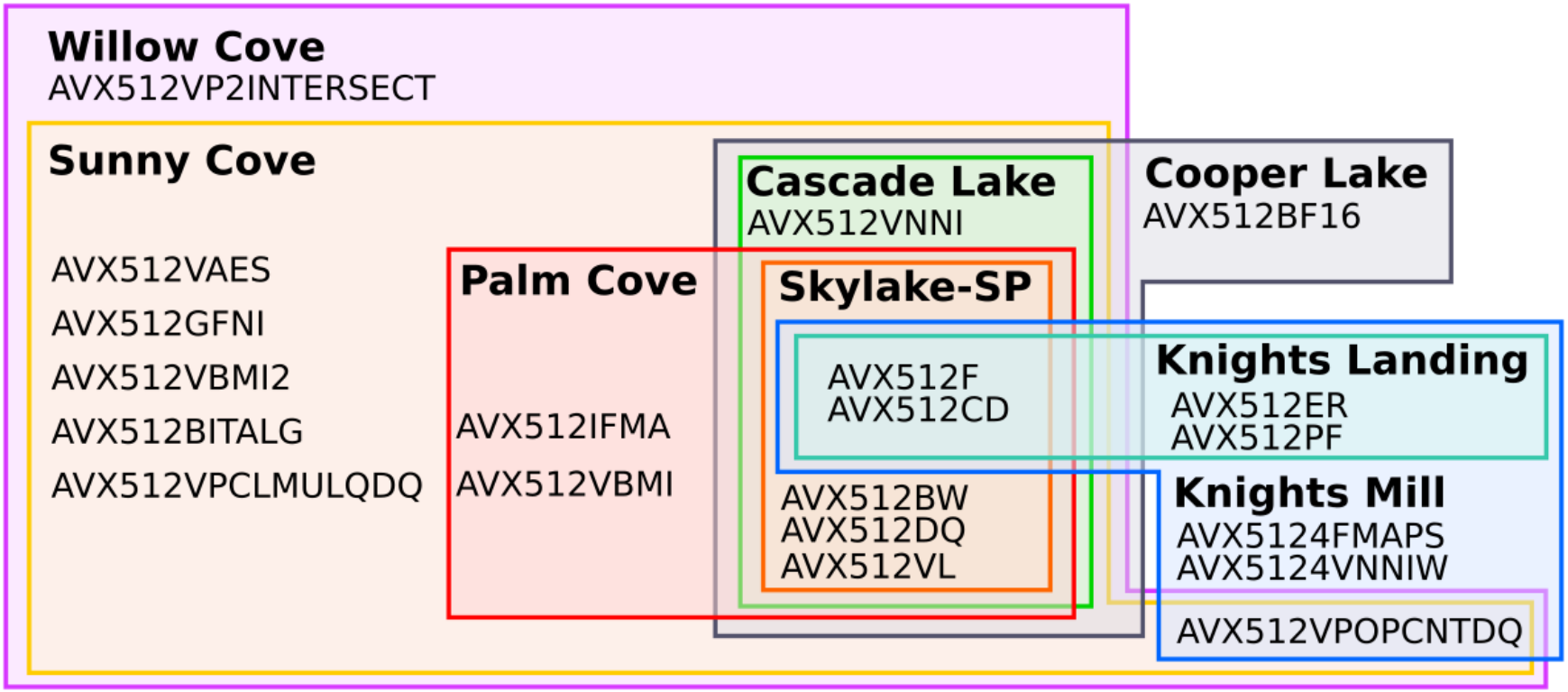
AVX512F - AVX-512 Foundation
AVX512CD - AVX-512 Conflict Detection
AVX512BW - AVX-512 Byte and Word
AVX512DQ - AVX-512 Doubleword and Quadword
AVX512VL - AVX-512 Vector Length
AVX512IFMA - AVX-512 Integer Fused Multiply-Add
AVX512_VNNI - AVX-512 Vector Neural Network Instructions
AVX512_BF16 - AVX-512 BFloat16 Instructions

*"I hope AVX512 dies a painful death, and that Intel starts fixing real problems…"*
Linus Torvalds

# The ISA chaos at Intel SIMD extensions…



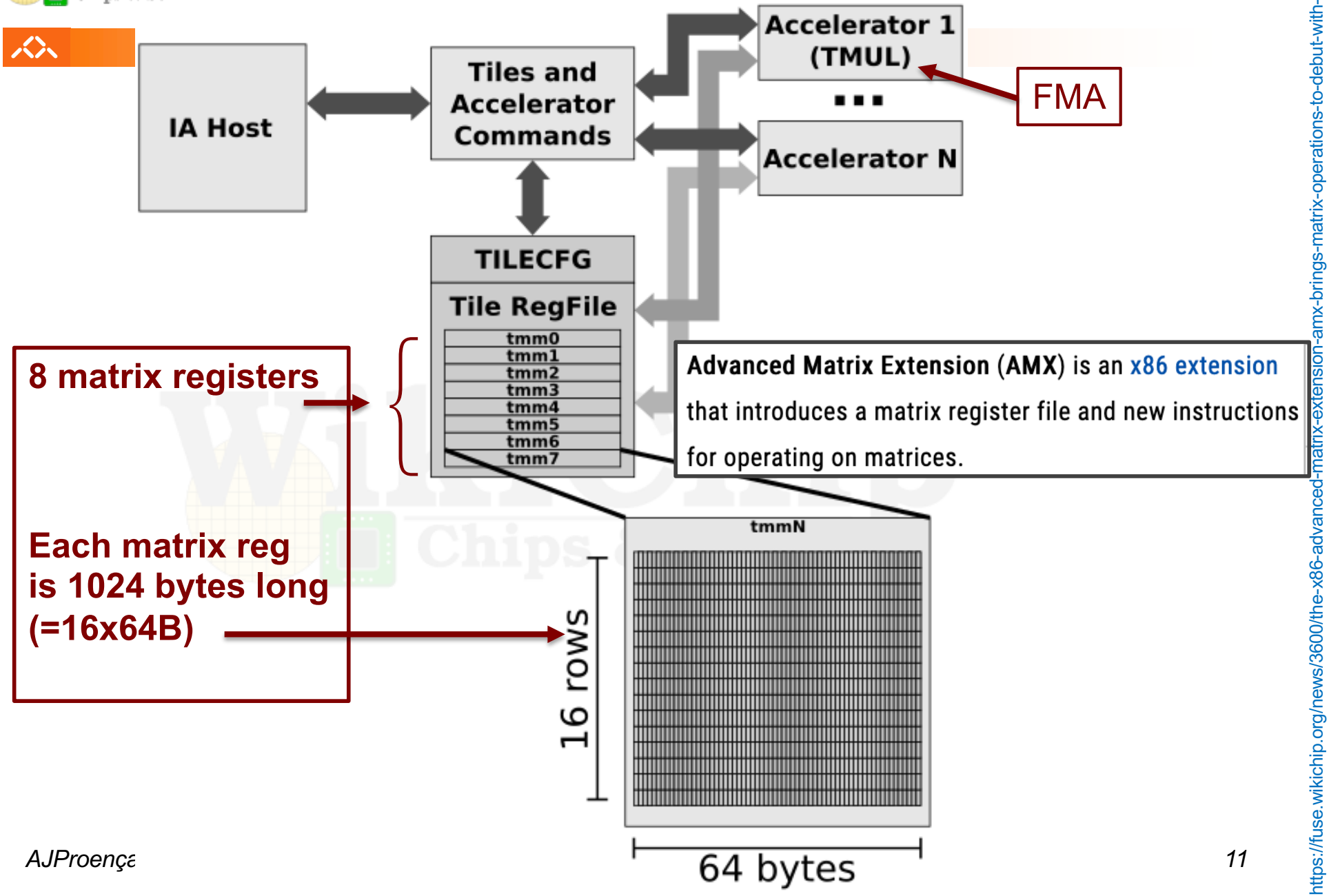
# The AVX-512 across Intel microarchitectures




| Intel AVX-512 Instruction Types | |
|---|---|
| AVX-512-F | AVX-512 Foundation Instructions |
| AVX-512-VL | Vector Length Orthogonality : ability to operate on sub-512 vector sizes |
| AVX-512-BW | 512-bit Byte/Word support |
| AVX-512-DQ | Additional D/Q/SP/DP instructions (converts, transcendental support, etc.) |
| AVX-512-CD | Conflict Detect : used in vectorizing loops with potential address conflicts |

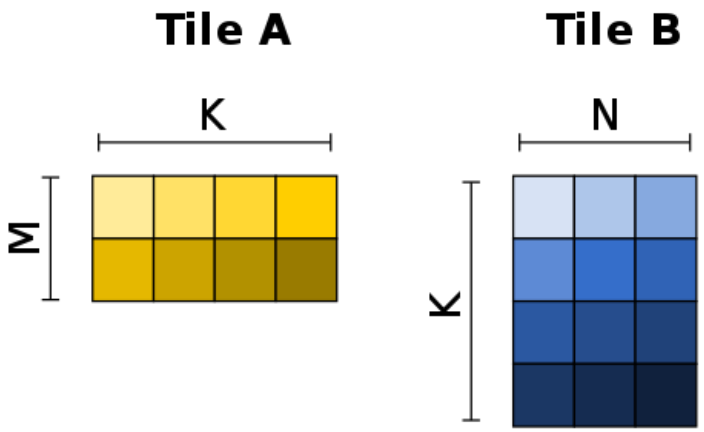
# Intel Advanced Matrix Extension (AMX)

*(expected in 2021)*



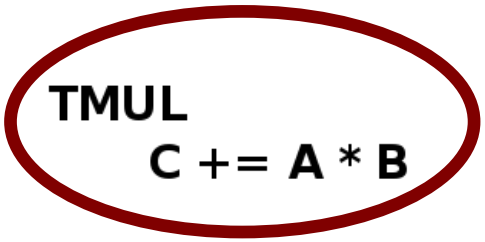

https://fuse.wikichip.org/news/3600/the-x86-advanced-matrix-extension-amx-brings-matrix-operations-to-debut-with-sapphire-rapids/

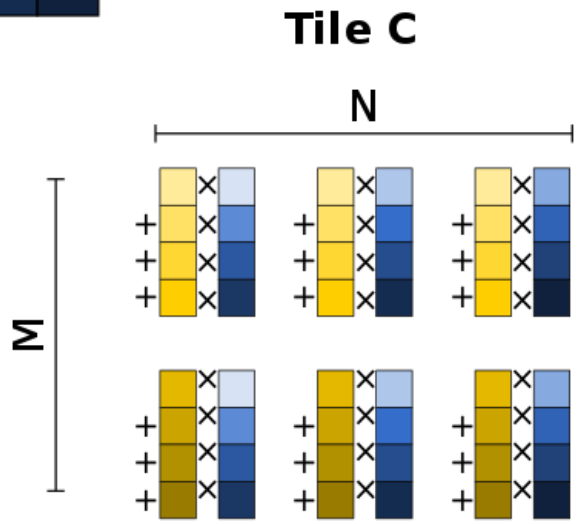
# AMX instructions in Accelerator 1

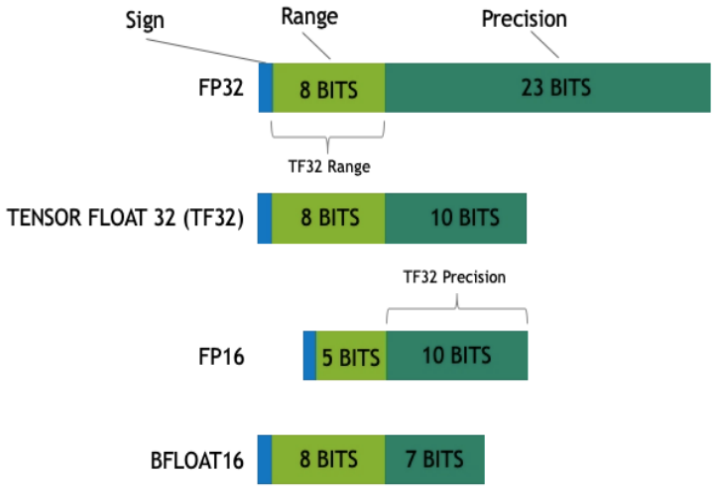



| AMX Extensions | | |
|---|---|---|
| Feature Set | Description | Instructions |
| AMX-TILE | The base matrix tile architecture support. | 7 instructions |
| AMX-INT8 | Dot-product of Int8 tiles. | 4 instructions |
| AMX-BF16 | Dot-product of BF16 tiles. | 1 instruction |

Data types



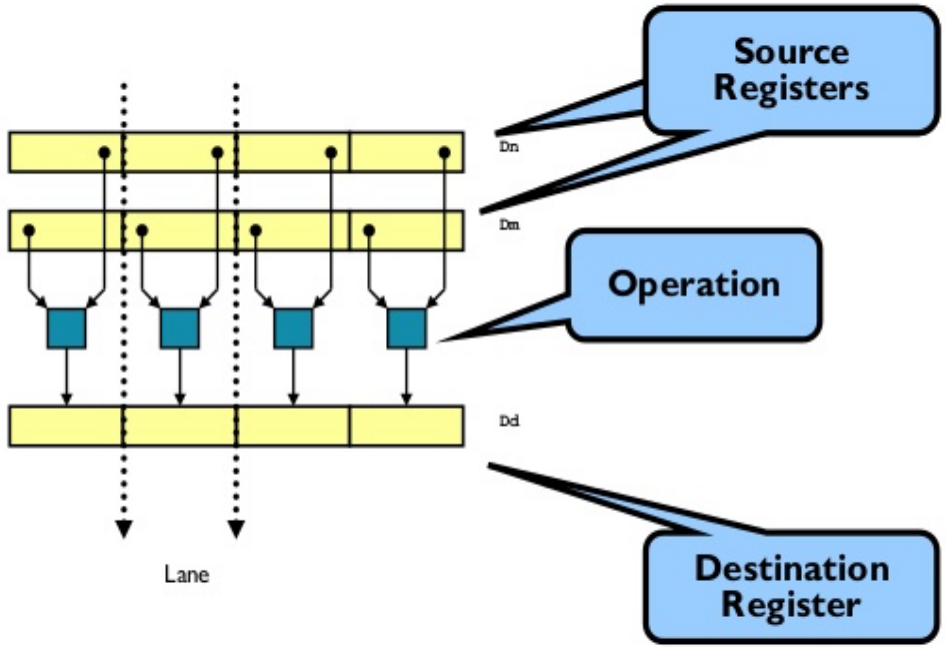



# NEON: the SIMD extension in ARM

## What is NEON?

- NEON™ is a wide SIMD data processing architecture
  - Extension of the ARM® instruction set
  - 32 registers, 64-bit wide
    (**AArch64: 32 registers**, 128-bit wide)
- NEON Instructions perform "Packed SIMD" processing
  - Registers are considered as vectors of elements of the same data type
  - Data types can be: signed/unsigned 8-bit, 16-bit, 32-bit, 64-bit, single precision float
    (**AArch64: Double precision float**)
  - Instructions perform the same operation in all lanes



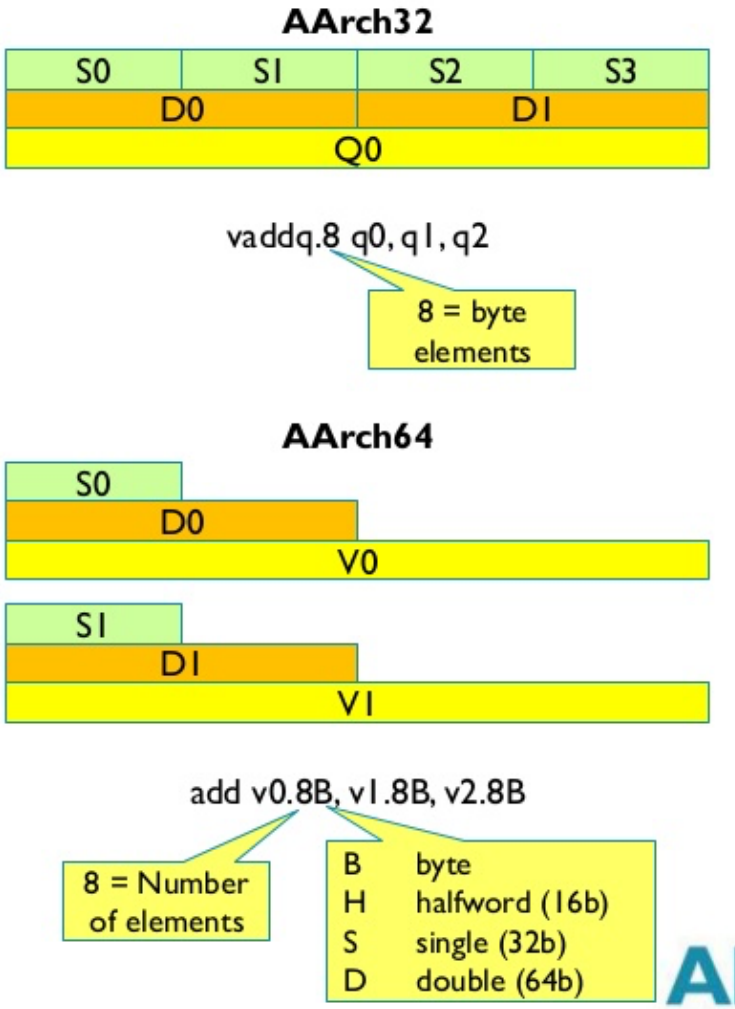

ARM

2

# NEON & FP registers: the move to ARMv8 (64-bit)

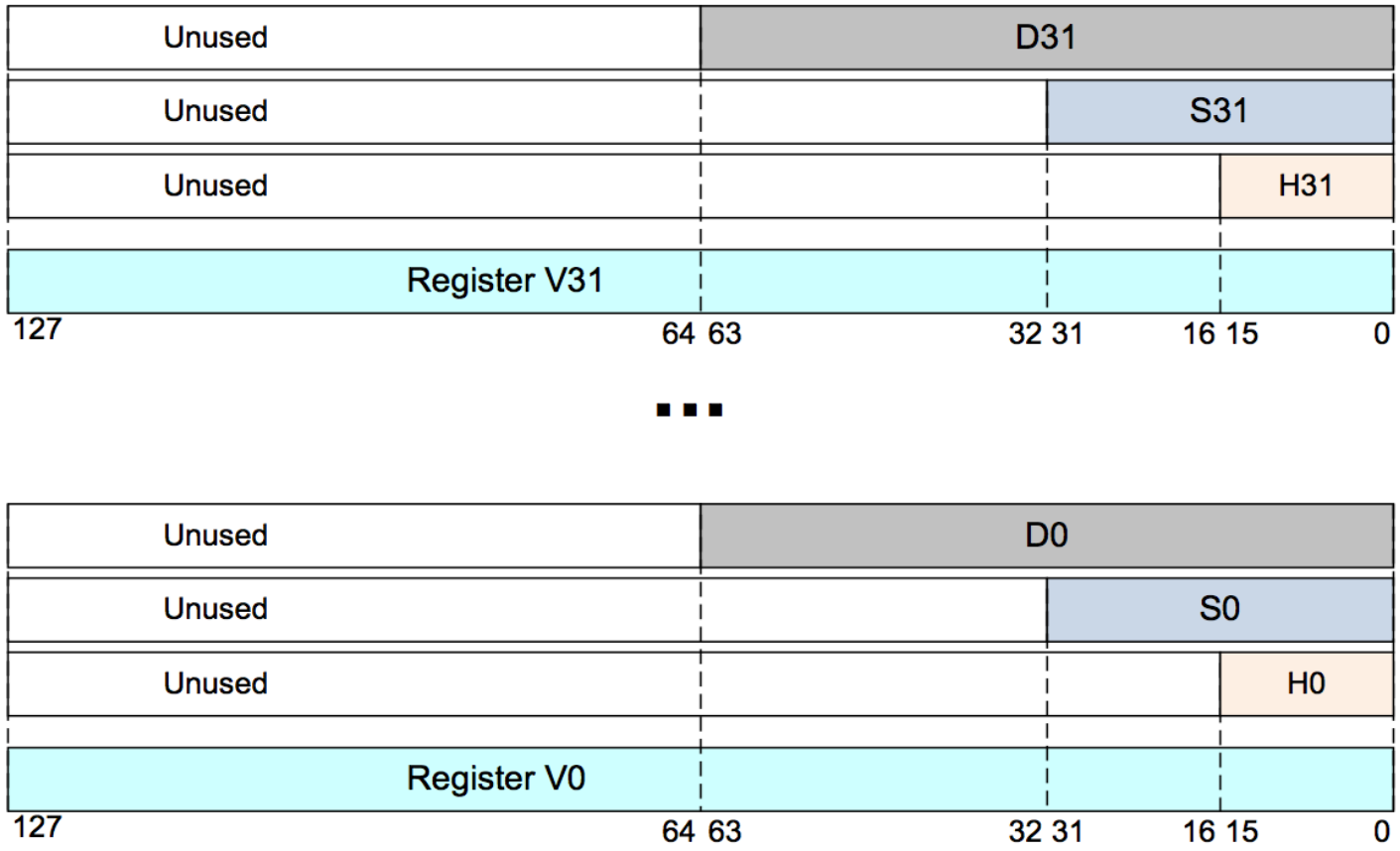
## Changes in AArch64

- More registers
  - AArch32: 16x128-bit "Q-regs" Q0-Q15
  - AArch64: 32x128-bit "V-regs" V0-V31
- 'Dual view' no longer packed
  - 32 registers of each type: S, D, V
  - Clearer mapping of overlap
- Asm language changes
  - No 'v' in mnemonics
  - Width specifier moved to register description
  - 128-bit Q-regs renamed V-regs




3

# Vector & FP register sizes in ARMv8 (64-bit)




32x 128-bit registers

**Figure 4-10 Arrangement of floating-point values**

**Note**

16-bit floating-point is supported, but only as a format to be converted from or to. It is not supported for data processing operations.

# NEON FP registers in ARMv8 (64-bit)

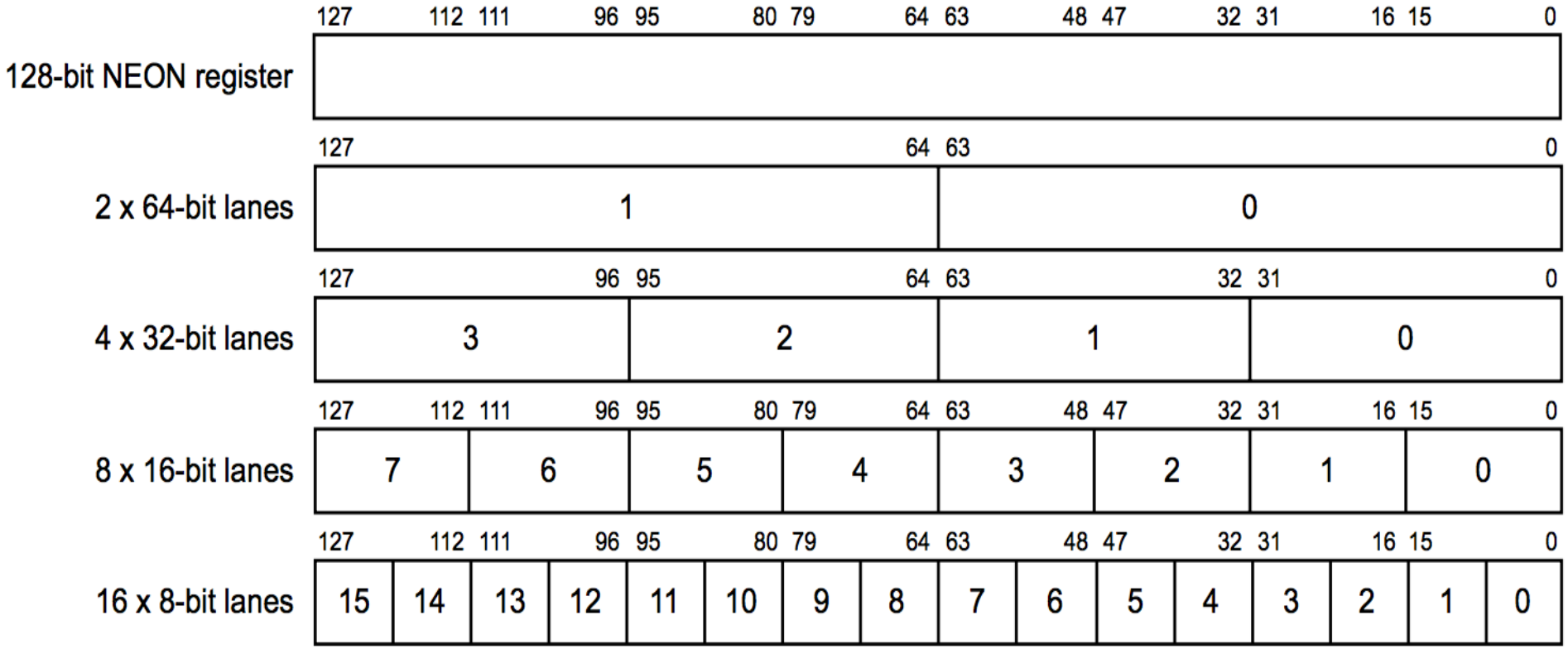
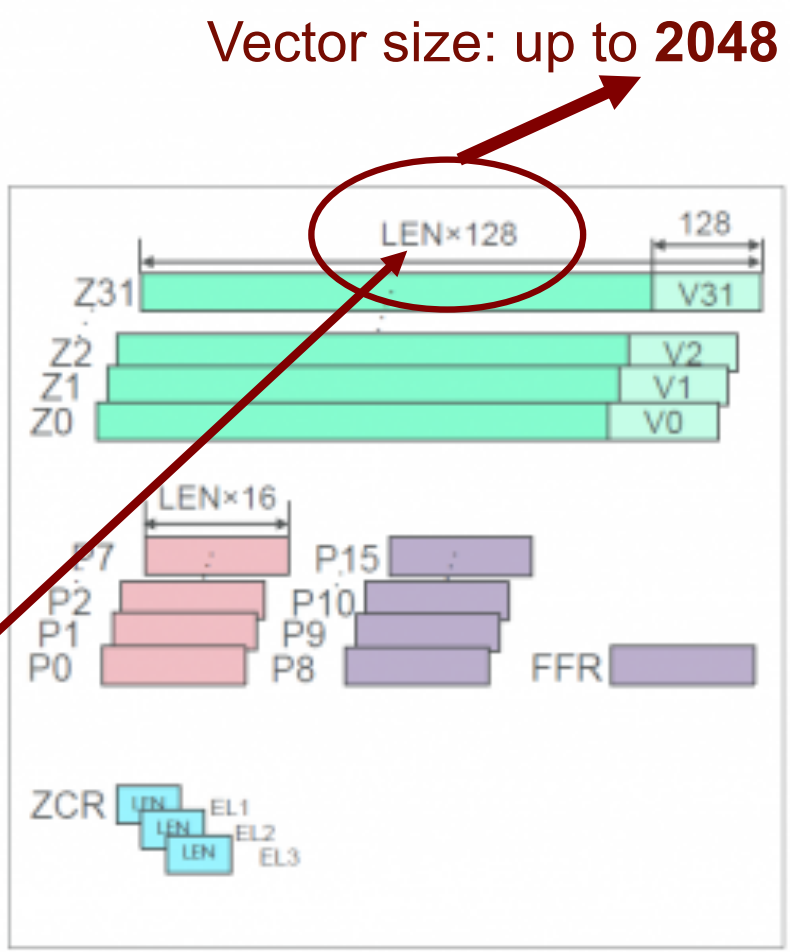



**Figure 7-1 Divisions of the V register**

# ARMv8 - A Scalable Vector Extension (SVE)

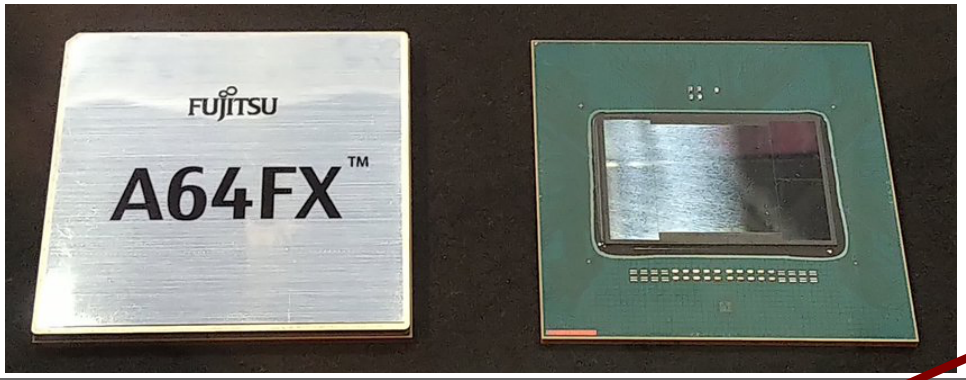
## SVE architectural state

- Scalable vector registers
  - Z0-Z31 extending NEON's V0-V31
    - DP & SP floating-point
    - 64, 32, 16 & 8-bit integer
- Scalable predicate registers
  - P0-P7 lane masks for ld/st/arith
  - P8-P15 for predicate manipulation
  - FFR *first fault register*
- Scalable vector control registers
  - ZCR_ELx vector length (LEN=1..16)
  - Exception / privilege level EL1 to EL3




5 ©ARM 2016 ARM

# The 1st implementation of SVE: Fujitsu A64FX




**ARM64**

**Fujitsu's A64FX Arm Chip:**
**SVE 4x128**

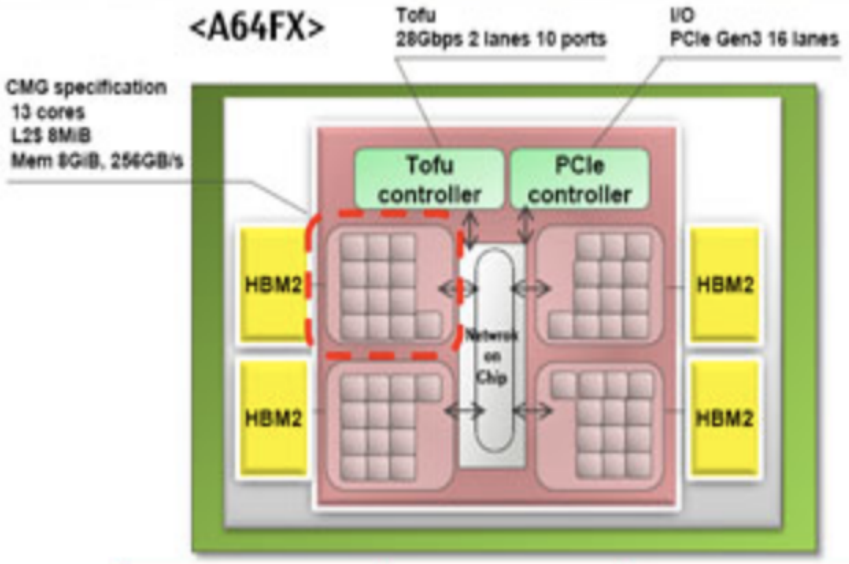
## A64FX Chip Overview

FUJITSU

- **Architecture Features**
  - ~~Armv8.2-A (AArch64 only)~~
  - SVE 512-bit wide SIMD
  - ~~48 computing cores~~ + 4 assistant cores*

*All the cores are identical

  - HBM2 32GiB
  - Tofu 6D Mesh/Torus
    28Gbps x 2 lanes x 10 ports
  - PCIe Gen3 16 lanes



# Beyond vector extensions

- Vector/SIMD-extended architectures are hybrid approaches
  - mix (super)**scalar + vector** op capabilities on a single device
  - **highly pipelined** approach to reduce memory access penalty
  - **tightly-closed access to shared memory**: lower latency
- Evolution of vector/SIMD-extended architectures
  - **computing accelerators optimized for number crunching (GPU)**
  - **add support for matrix multiply + accumulate operations**; **why?**
    - most scientific, engineering, AI & finance applications use matrix computations, namely the dot product: multiply and accumulate the elements in a row of a matrix by the elements in a column from another matrix
    - manufacturers typically call these extensions **Tensor Processing Unit** (TPU)
  - **support for half-precision FP & 8-bit integer; why?**
    - machine learning using neural nets is becoming very popular; to compute the model parameter during training phase, intensive matrix products are used and with very low precision (is adequate!)