



Lic. Physics Eng.

2021/22

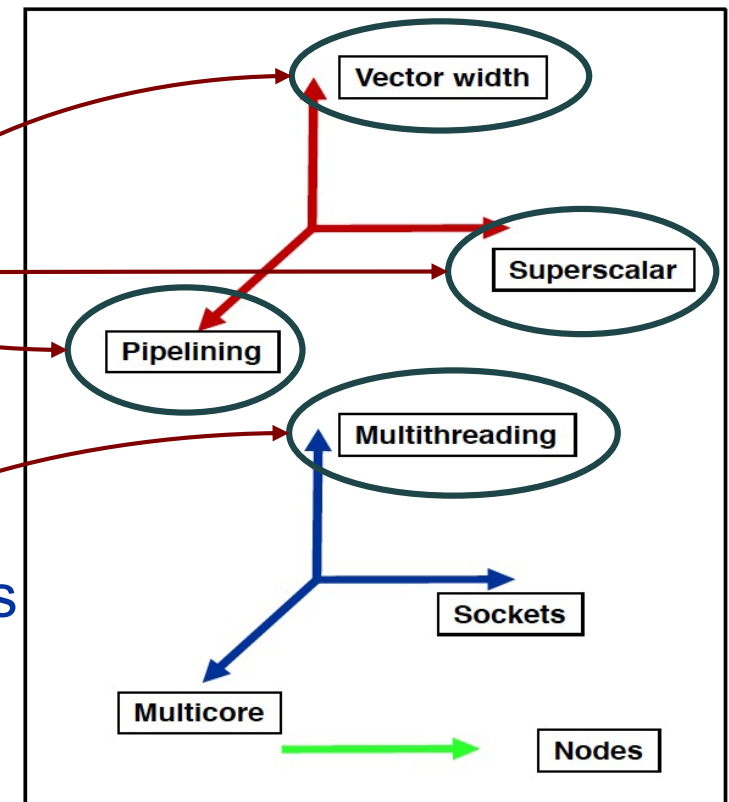
A.J.Proença

**At core level:
Multithreading and Data Parallelism**

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



Multithreading

Performing multiple threads of execution in parallel

- Share all resources but replicate registers, PC/IP, etc.
- Fast switching between threads

1. Fine-grain multithreading / time-multiplexed MT

- Switch threads after each cycle
- Interleave instruction execution
- If one thread stalls, others are executed

2. Coarse-grain multithreading

- Only switch on long stall (e.g., L2-cache miss)
- Simplifies hardware, but doesn't hide short stalls (eg, data hazards)

3. Simultaneous multithreading (*next slide...*)

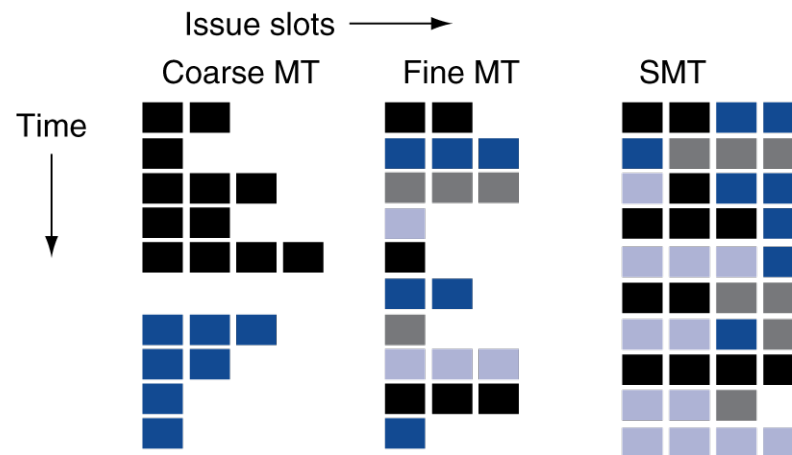
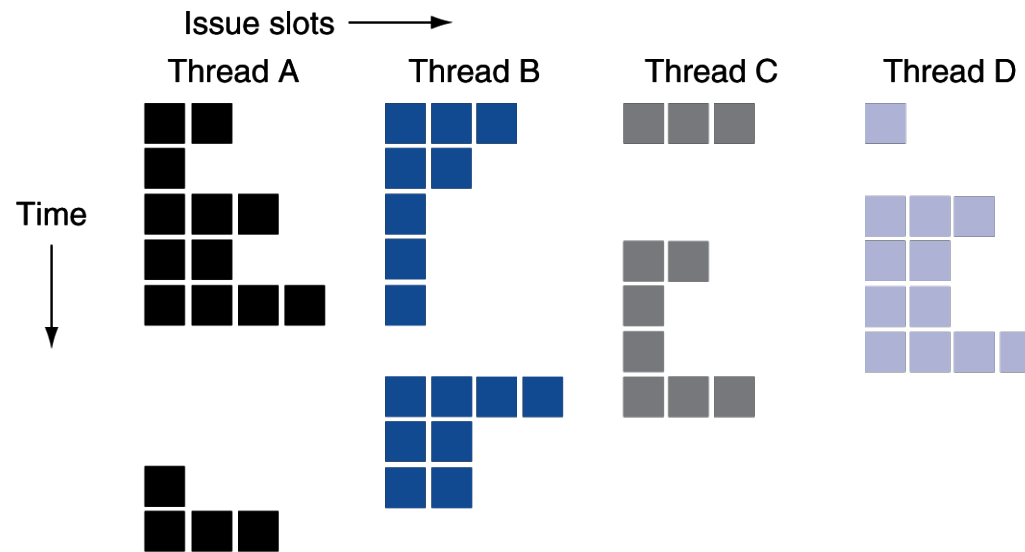
3. Simultaneous Multithreading

- In multiple-issue dynamically scheduled processor
 - Schedule instructions from multiple threads
 - Instructions from independent threads execute when function units are available
 - Within threads, dependencies handled by scheduling and register renaming
- Example: Intel from Pentium-4 HT
 - Two threads: duplicated registers, shared function units and caches

*HT: Hyper-Threading, Intel trade mark for their SMT implementation
MT in Xeon Phi KNC: 4-way SMT with time-mux MT, **not HT!***



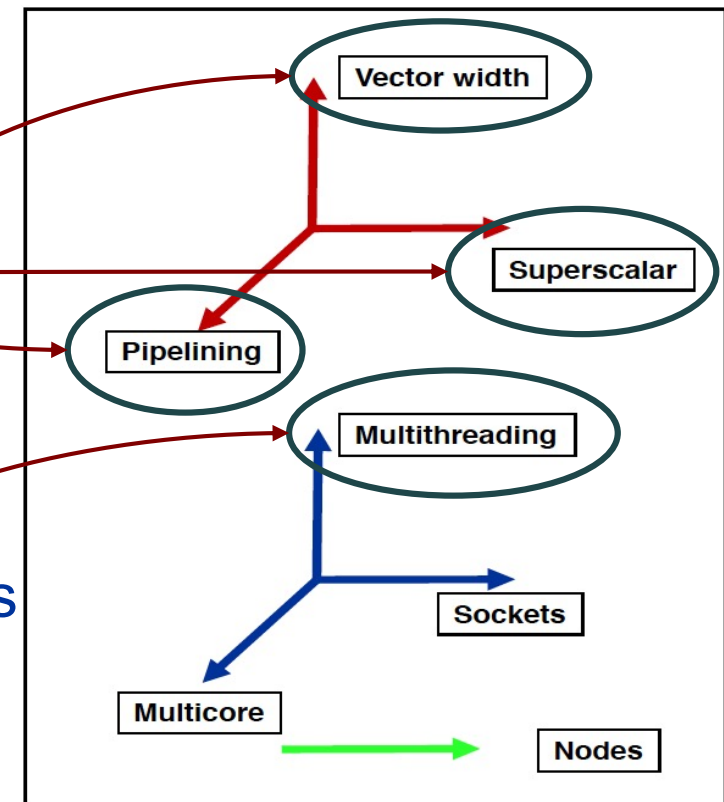
Multithreading Example



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



Instruction and Data Streams

Flynn's Taxonomy of Computers *

		Data Streams	
		Single	Multiple
Instruction Streams	Single	SISD: Intel Pentium 4	SIMD: SSE instructions of x86
	Multiple	MISD: No examples today	MIMD: Intel Xeon e5345

- **SPMD: Single Program Multiple Data**
 - A parallel program on a MIMD computer
 - Conditional code for different processors

* Mike Flynn, "Very High-Speed Computing Systems", *Proc. of IEEE*, 1966



Introduction

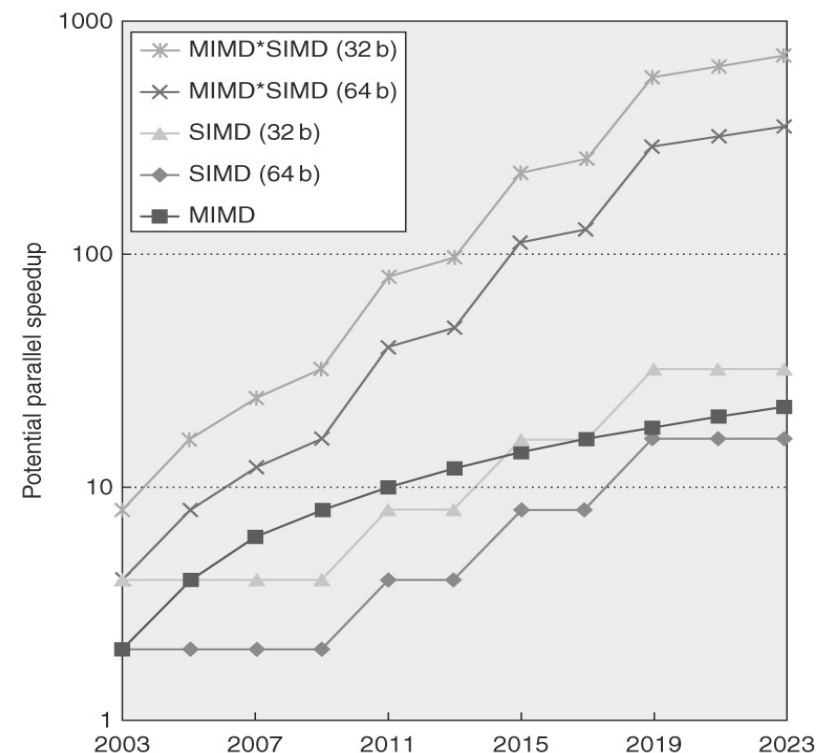
- SIMD architectures can exploit significant data-level parallelism for:
 - matrix-oriented scientific computing
 - media-oriented image and sound processing
- SIMD is more energy efficient than MIMD
 - only needs to fetch one instruction per data operation
 - makes SIMD attractive for personal mobile devices
- SIMD allows programmers to continue to think sequentially



SIMD Parallelism

- Vector architectures
- SIMD & extensions
- Graphics Processor Units (GPU) *(another set of slides)*

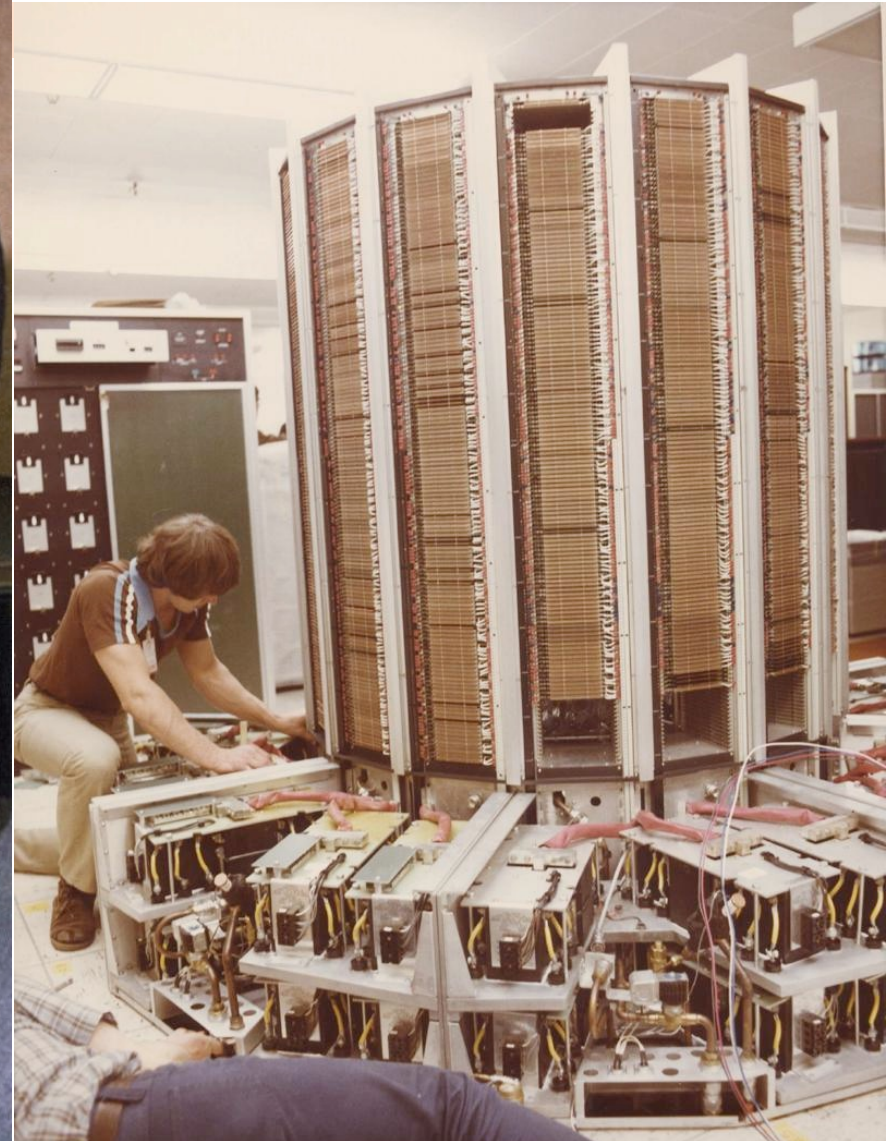
- For x86 processors:
 - Expected grow:
2 more cores/chip/year
 - SIMD width:
2x every 4 years
 - Potential speedup:
SIMD 2x that from MIMD!



Vector Architectures

- Basic idea:
 - Read sets of data elements (*gather from memory*) into “vector registers”
 - Operate on those registers
 - Store/scatter the results back into memory
- Registers are controlled by the compiler
 - Used to hide memory latency
 - Leverage memory bandwidth

Cray-1 Supercomputer (1976)



Challenges

- Start up time
 - Latency of vector functional unit
 - Assume the same as Cray-1
 - Floating-point add => 6 clock cycles
 - Floating-point multiply => 7 clock cycles
 - Floating-point divide => 20 clock cycles
 - Vector load => 12 clock cycles
- Improvements:
 - > 1 element per clock cycle
 - Non-64 wide vectors
 - IF statements in vector code
 - Memory system optimizations to support vector processors
 - Multiple dimensional matrices
 - Sparse matrices
 - Programming a vector computer

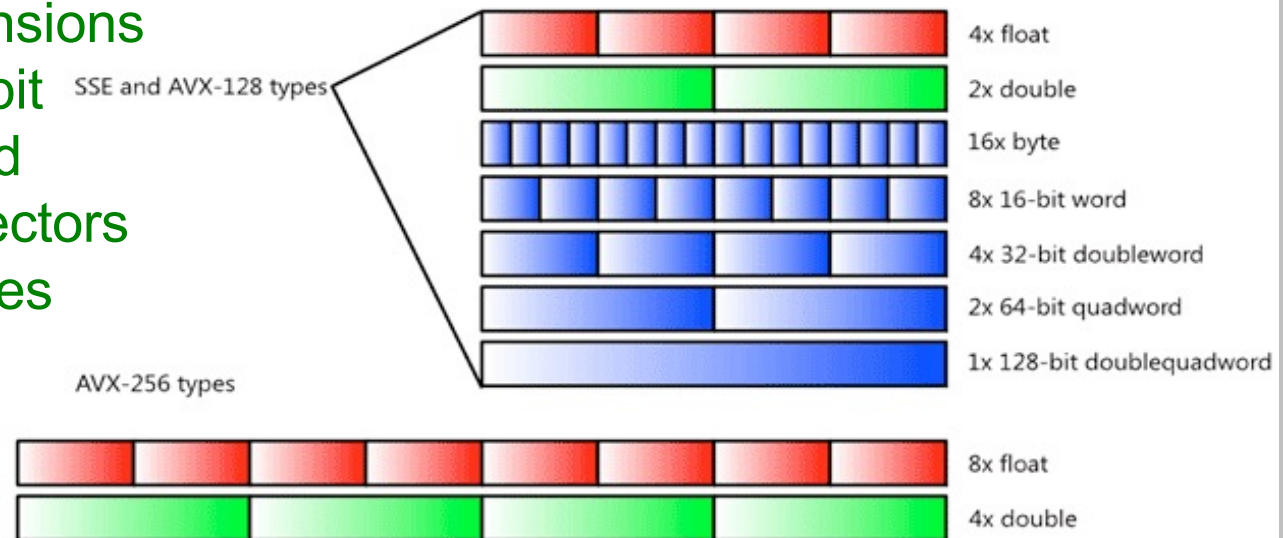
Vector Programming

- Compilers are a key element to give hints on whether a code section will vectorize or not
- Check if loop iterations have data dependencies, otherwise vectorization is compromised
- Vector Architectures have a too high cost, but simpler variants are currently available on off-the-shelf devices, as extensions to the scalar processor; however:
 - most do not support non-unit stride => care must be taken in the design of data structures
 - same applies for gather-scatter...

SIMD Extensions

- Media applications operate on data types narrower than the native word size

- Intel SIMD extensions started with 64-bit wide vectors and grew to wider vectors and more facilities
- Current AVX generation is 512-bit wide

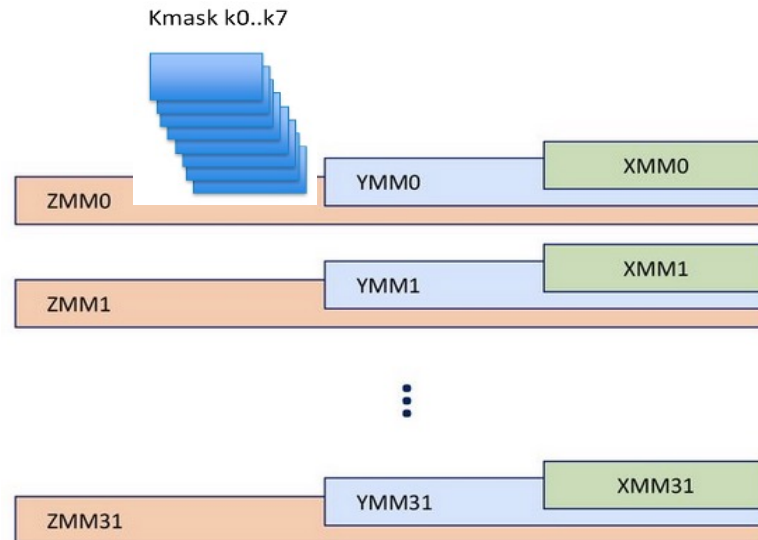
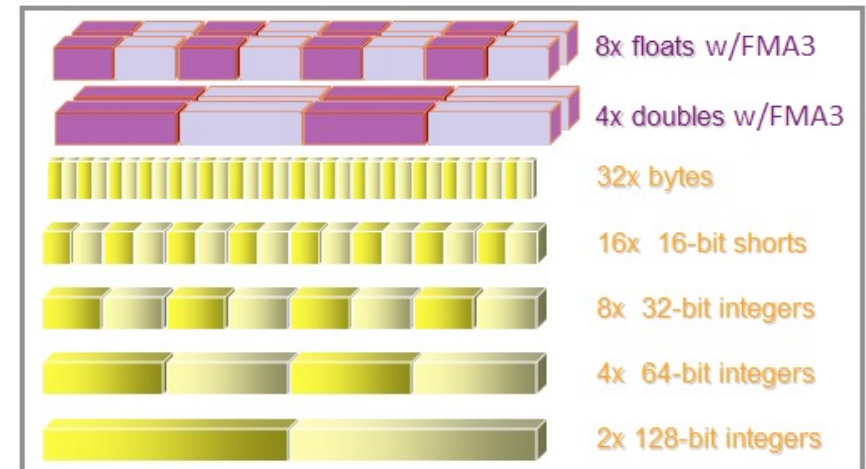
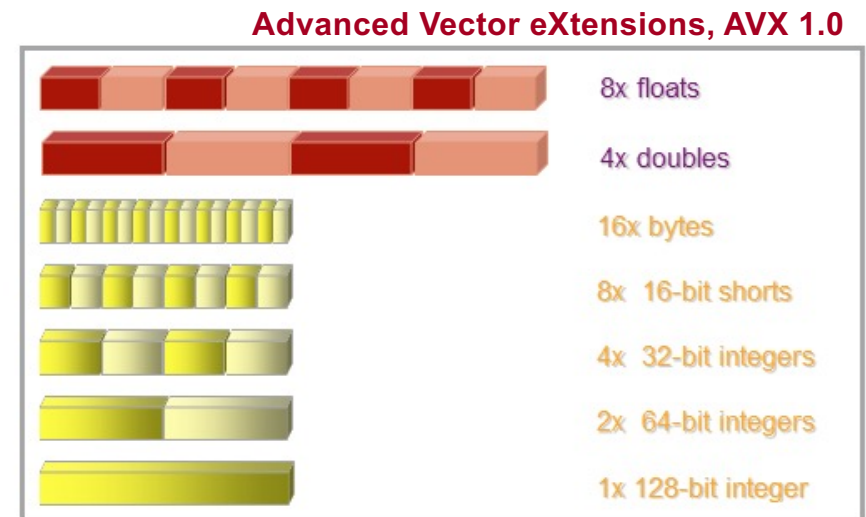
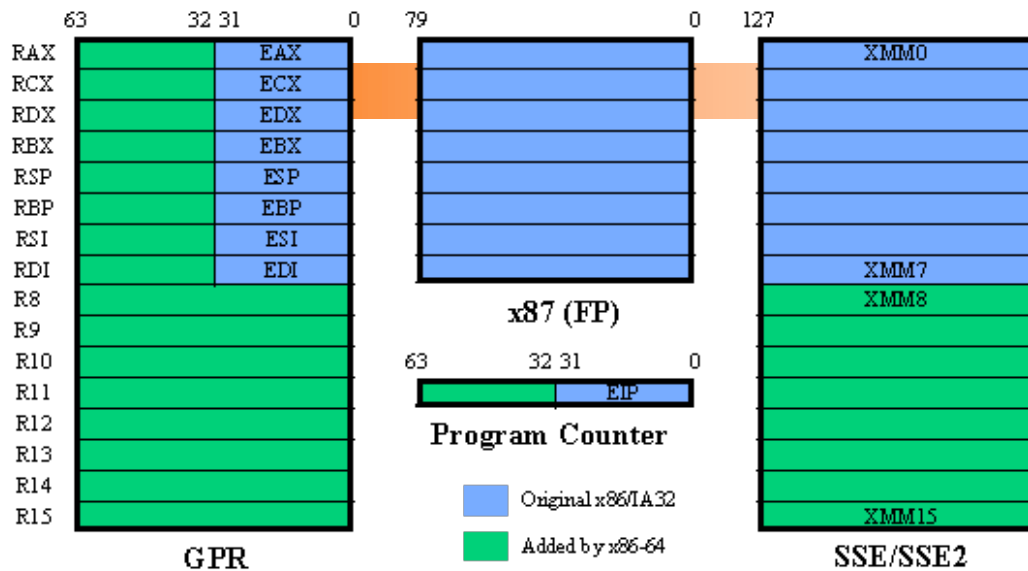


- Limitations, compared to vector architectures:
 - Number of data operands encoded into op code
 - No sophisticated addressing modes (strided, scatter-gather, but...)
 - No mask registers

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

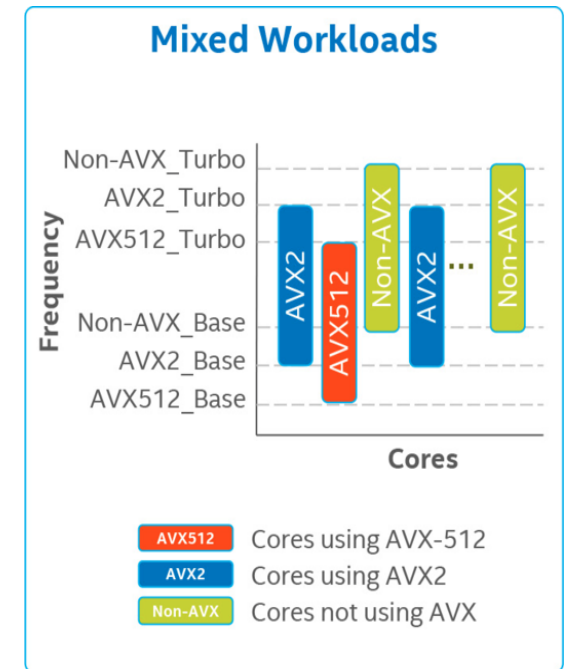
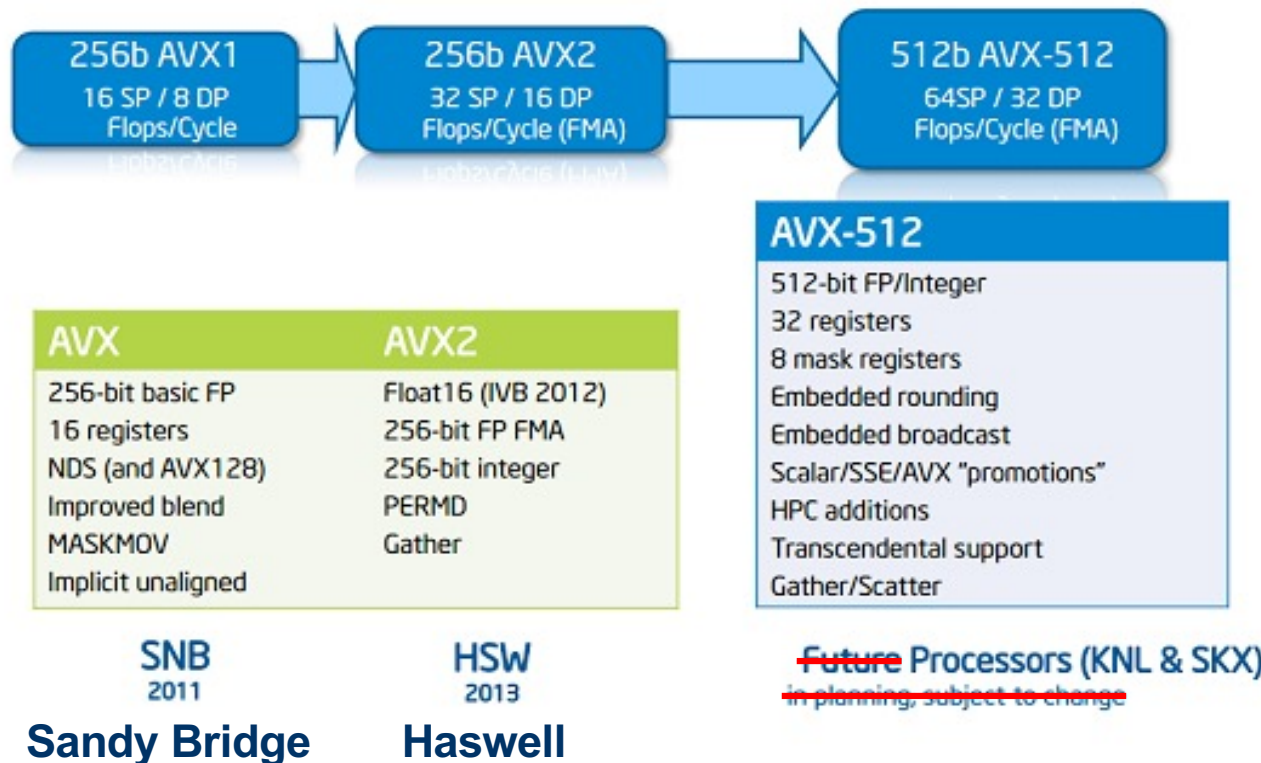


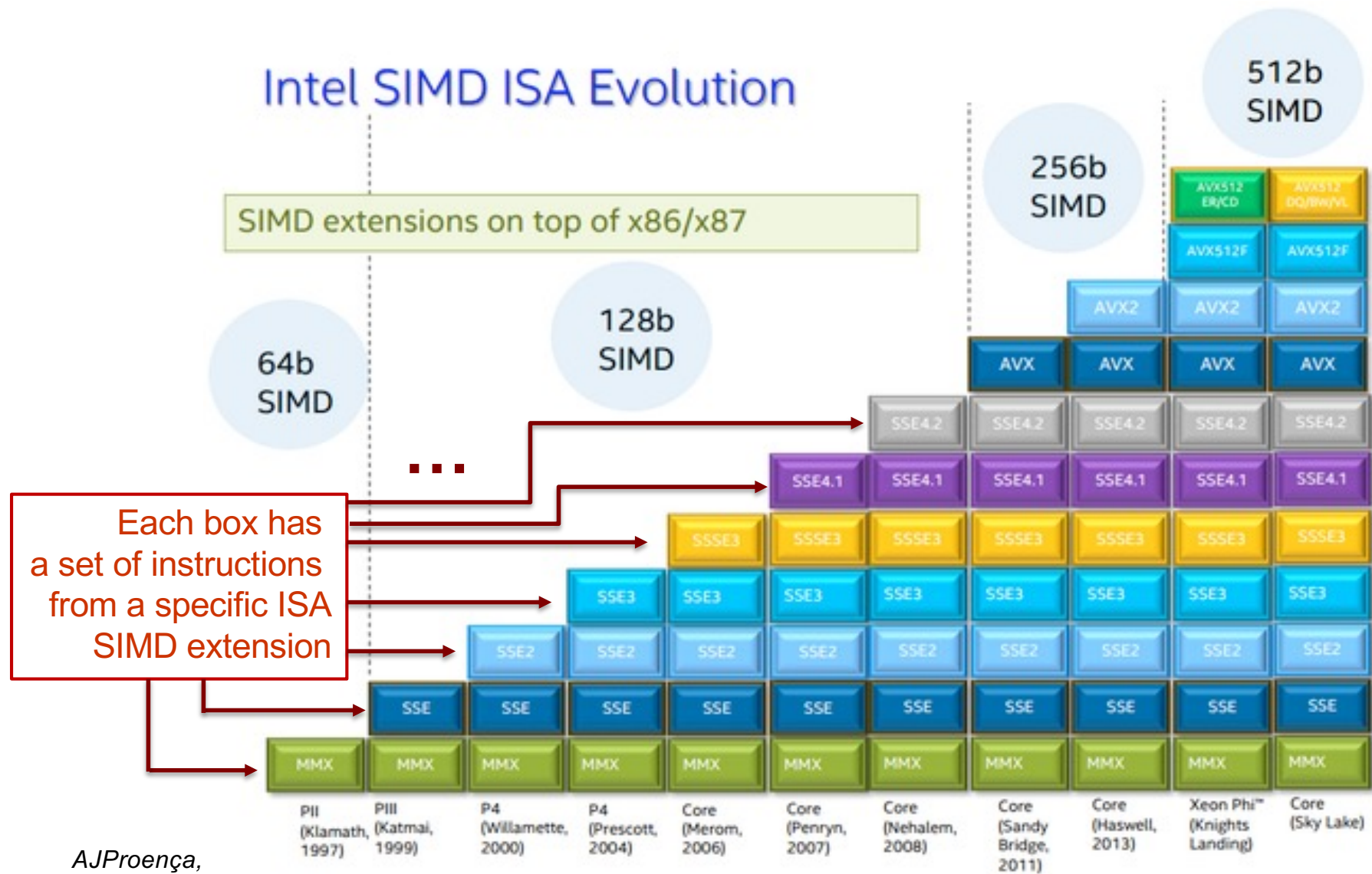
- ☐ XMM0 – XMM15
 - o 128-bit registers
 - o SSE
- ☐ YMM0 – YMM15
 - o 256-bit registers
 - o AVX, AVX2
- ☐ ZMM0 – ZMM31
 - o 512-bit registers
 - o AVX-512

Intel evolution to the AVX-512



Intel® AVX Technology





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](#)

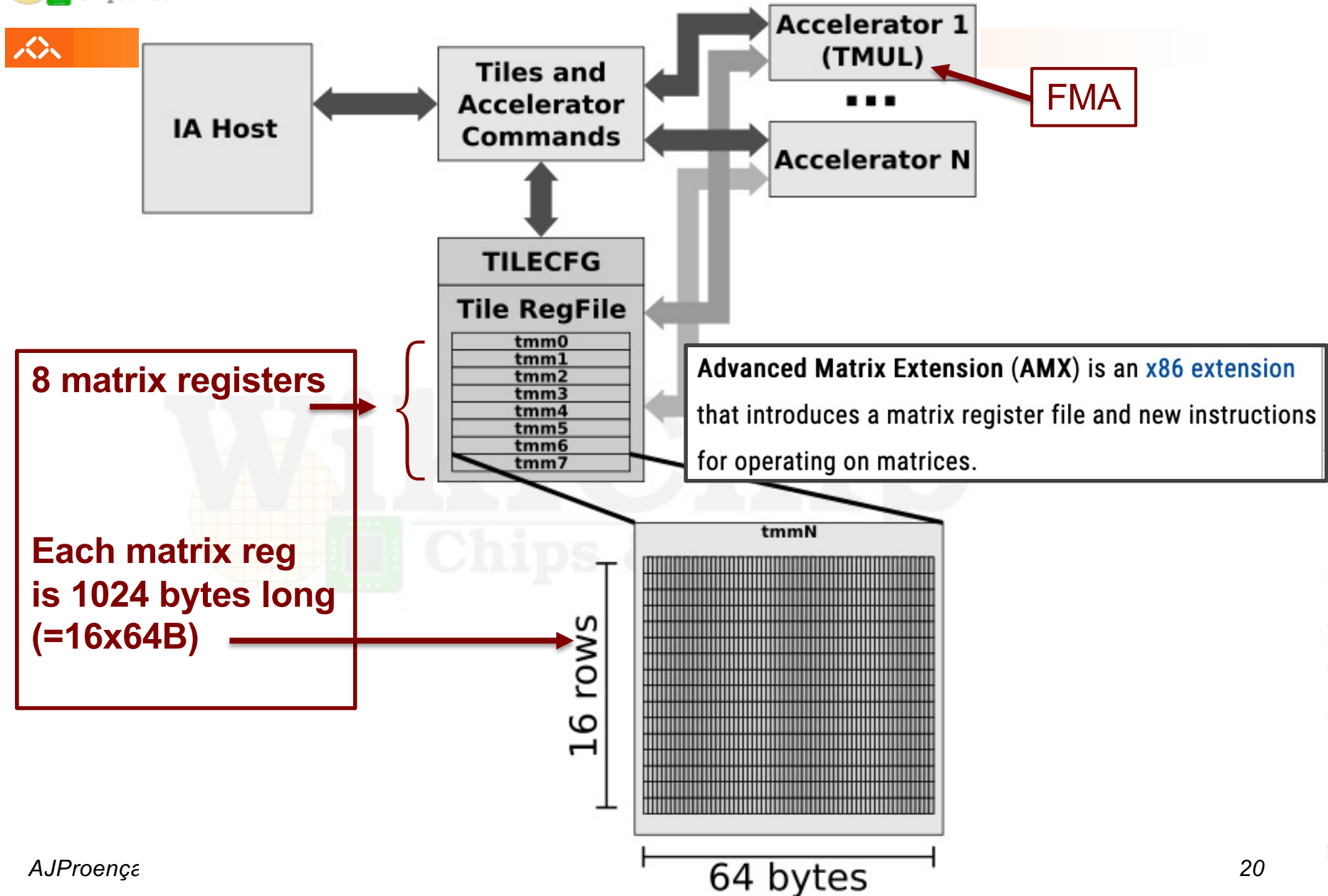
AVX512_VNNI - [AVX-512 Vector Neural Network Instructions](#)

AVX512_BF16 - [AVX-512 BFloat16 Instructions](#)

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

Intel Advanced Matrix Extension (AMX)

(expected in 2021)





ARM architecture

ARM architecture

ARM (stylised in lowercase as **arm**, previously an acronym for **Advanced RISC Machines** and originally **Acorn RISC Machine**) is a family of [reduced instruction set computing](#) (RISC) [architectures](#) for [computer processors](#), configured for various environments. [Arm Ltd.](#) develops the architecture and licenses it to other companies, who design their own products that implement one of those architectures—including [systems-on-chips](#) (SoC) and [systems-on-modules](#) (SoM) that

History [\[edit \]](#)

BBC Micro [\[edit \]](#)

Main article: [BBC Micro](#)

[Acorn Computers](#)' first widely successful design was the [BBC Micro](#), introduced in December 1981. This was a relatively conventional machine based on the [MOS 6502](#) CPU but ran at roughly double the performance of competing designs like the [Apple II](#) due to its use of faster [DRAM](#). Typical DRAM of the era ran at about 2 MHz; Acorn arranged a deal with [Hitachi](#) for a supply of faster 4 MHz parts.^[16]

NEON vector & FP registers in Armv8 (64-bit)

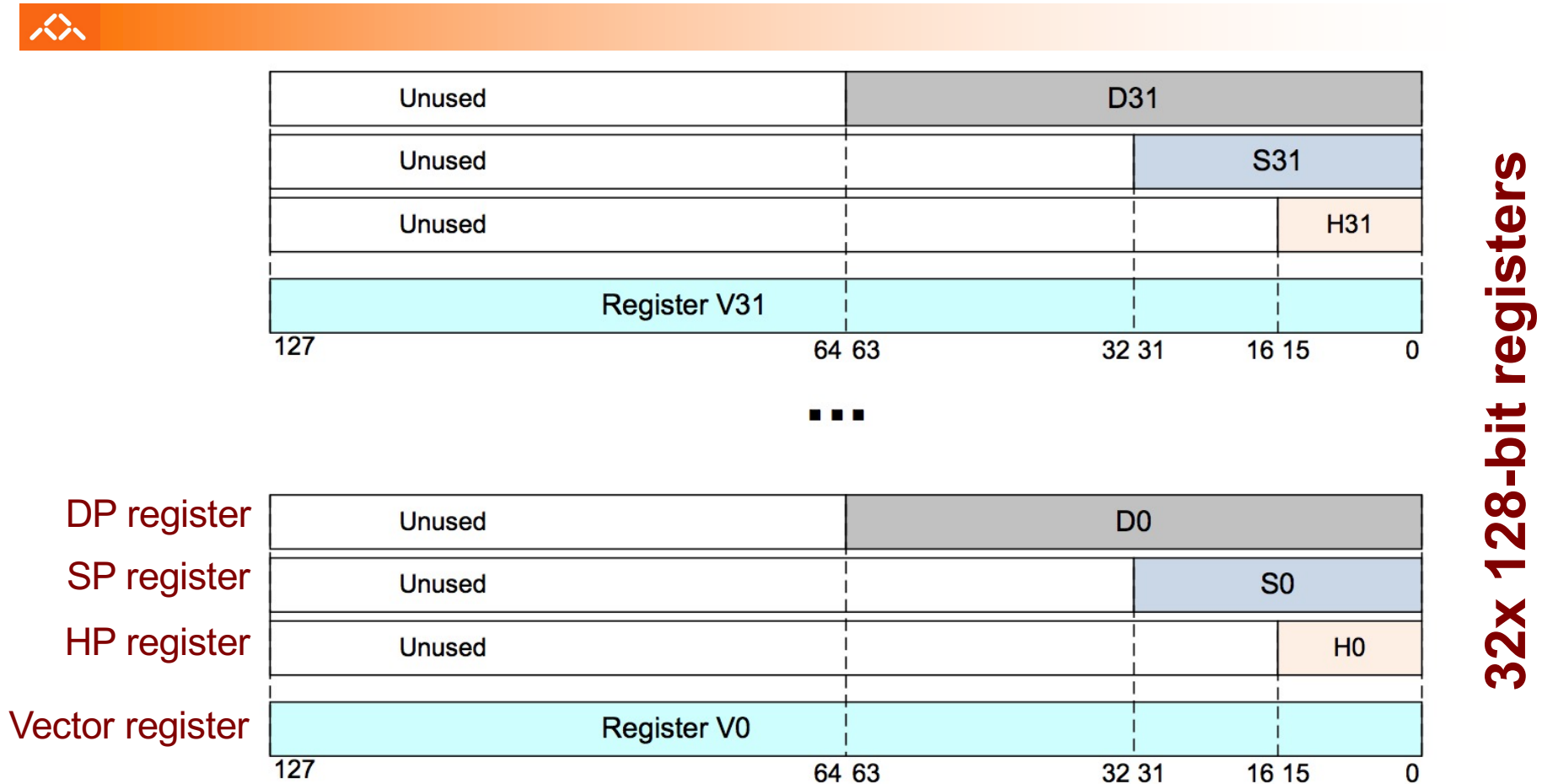


Figure 4-10 Arrangement of floating-point values

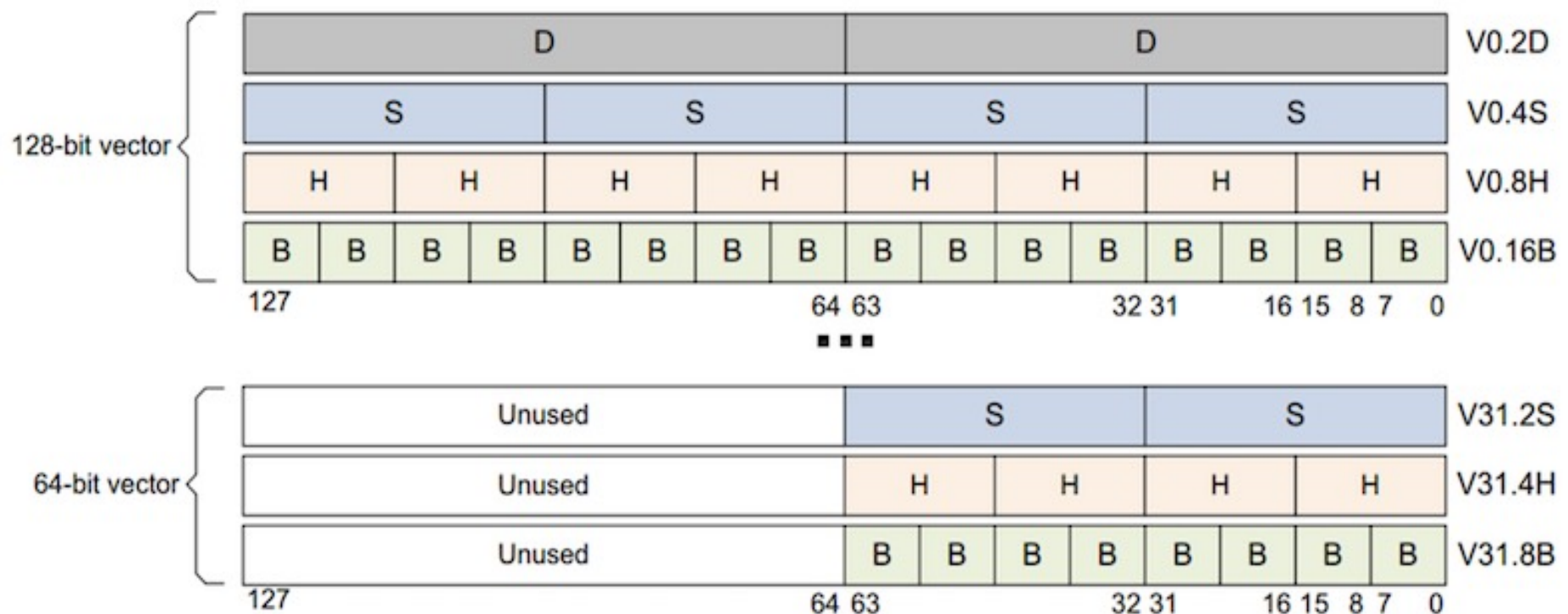
Note

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 vector & FP registers in Armv8 (64-bit)



Fundamentals of Armv8 Neon technology



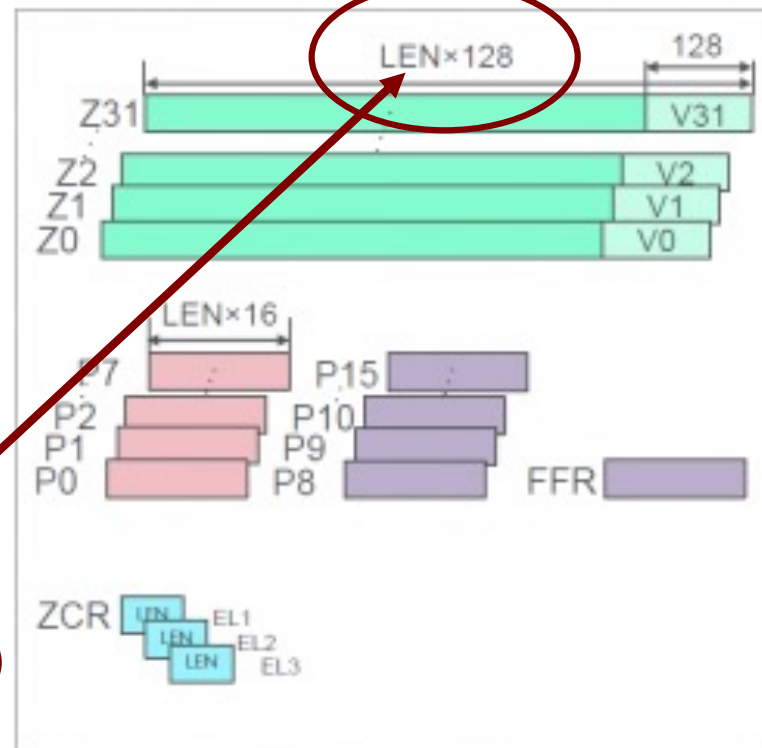
<https://developer.arm.com/documentation/102474/0100/Fundamentals-of-Armv8-Neon-technology>

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



Vector size: up to **2048**

The 1st implementation of SVE: Fujitsu A64FX



ARM64

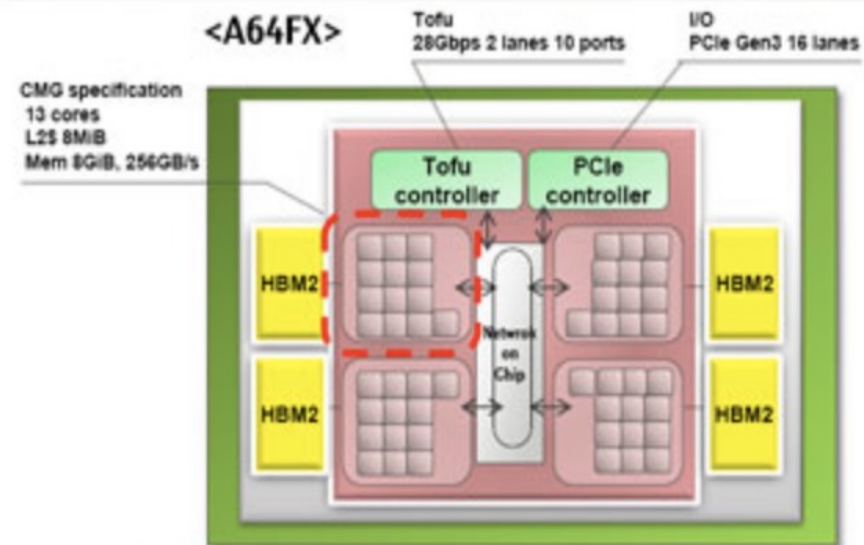
Fujitsu's A64FX Arm Chip:
SVE 4x128

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 extension **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!)

Reading suggestions



From CAQA 5th Ed

- Concepts and challenges in ILP: section 3.1
- Pipelining: basic and intermediate concepts: App. C
- Exploiting ILP w/ multiple issue & static scheduling: 3.7
- Exploiting ILP w/ dyn sched, multiple issue & specul: 3.8
- Multithread: exploiting TLP on uniprocessors: 3.12
- Vector architecture: 4.2
- SIMD instruction set extensions for multimedia: 4.3
- Graphic processing units: *(later...)* 4.4
- Detecting and enhancing loop-level parallelism: 4.5

Slides on Vector Computing from previous year

- in <http://gec.di.uminho.pt/lef/pp/T13-ArqVect.pdf>