Master Informatics Eng.

2019/20

A.J.Proença

Data Parallelism 1 (*vector & SIMD extensions*)

(most slides are borrowed)

Instruction and Data Streams

		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

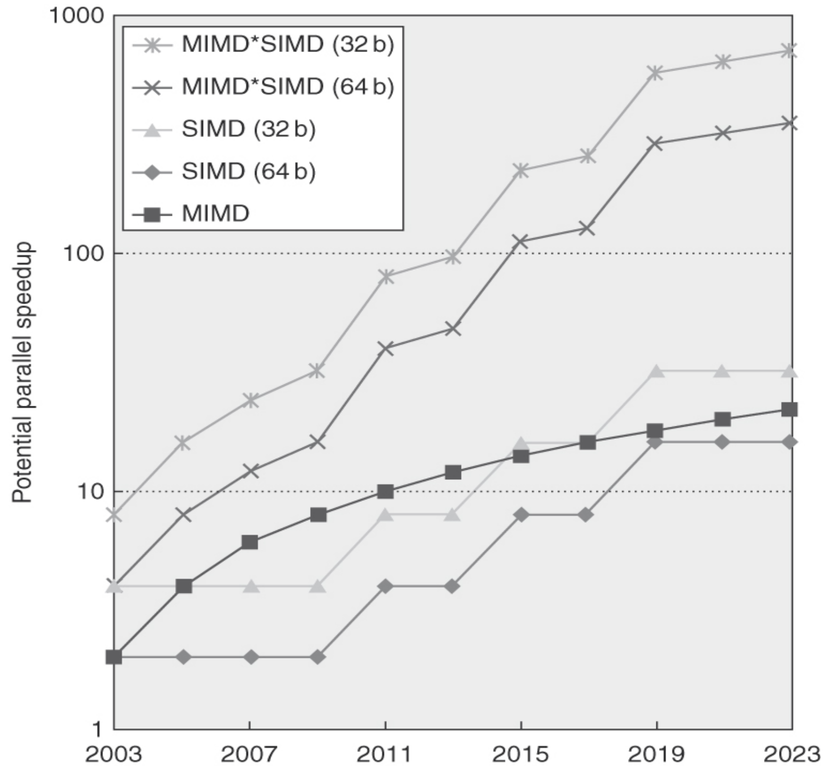
Introduction

- 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 *(slides 5 to 19)*
- SIMD & extensions *(slides 20 to 29 and next set)*
- Graphics Processor Units (GPUs) *(next set)*

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



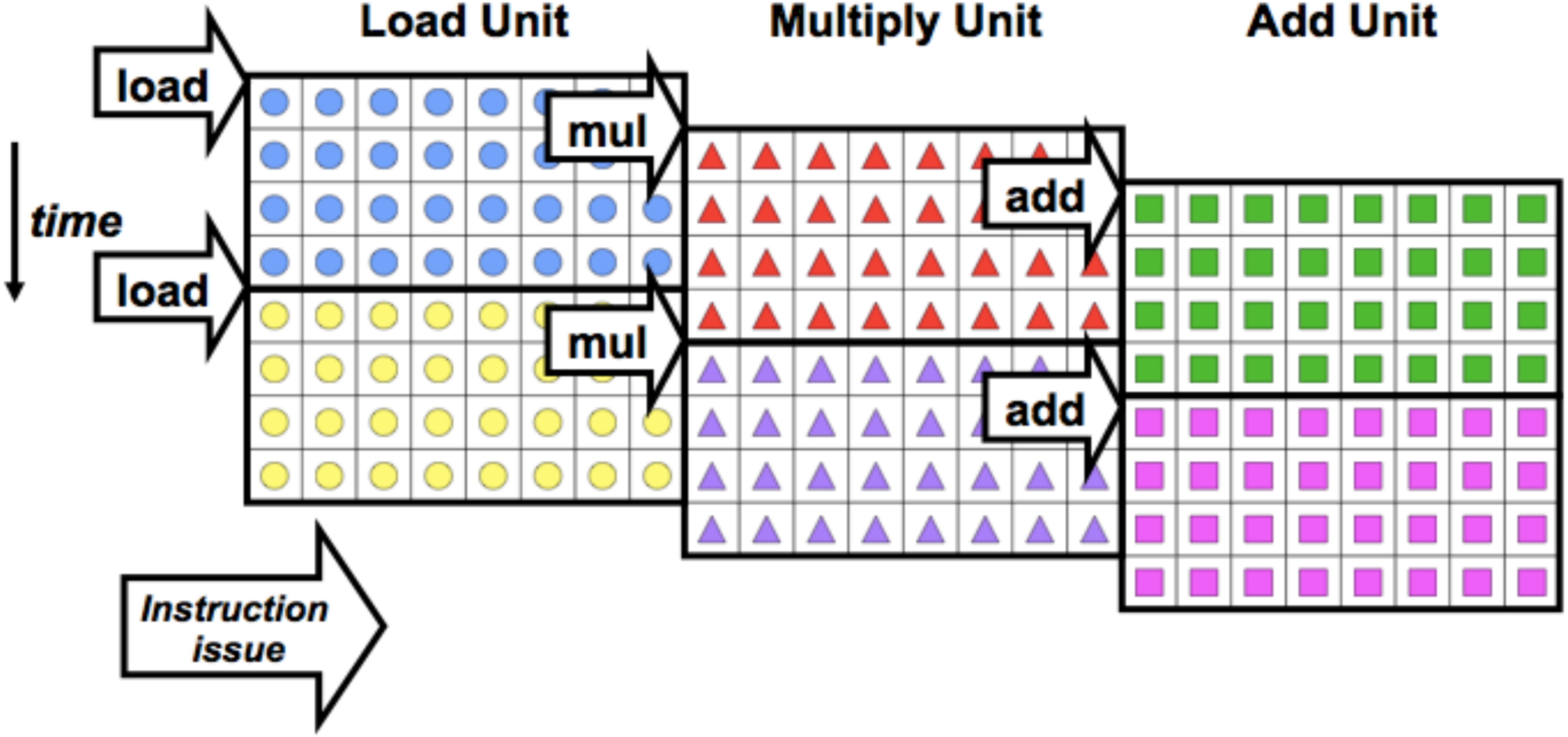
Vector Architectures

- 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

Vector Instruction Parallelism

Can overlap execution of multiple vector instructions

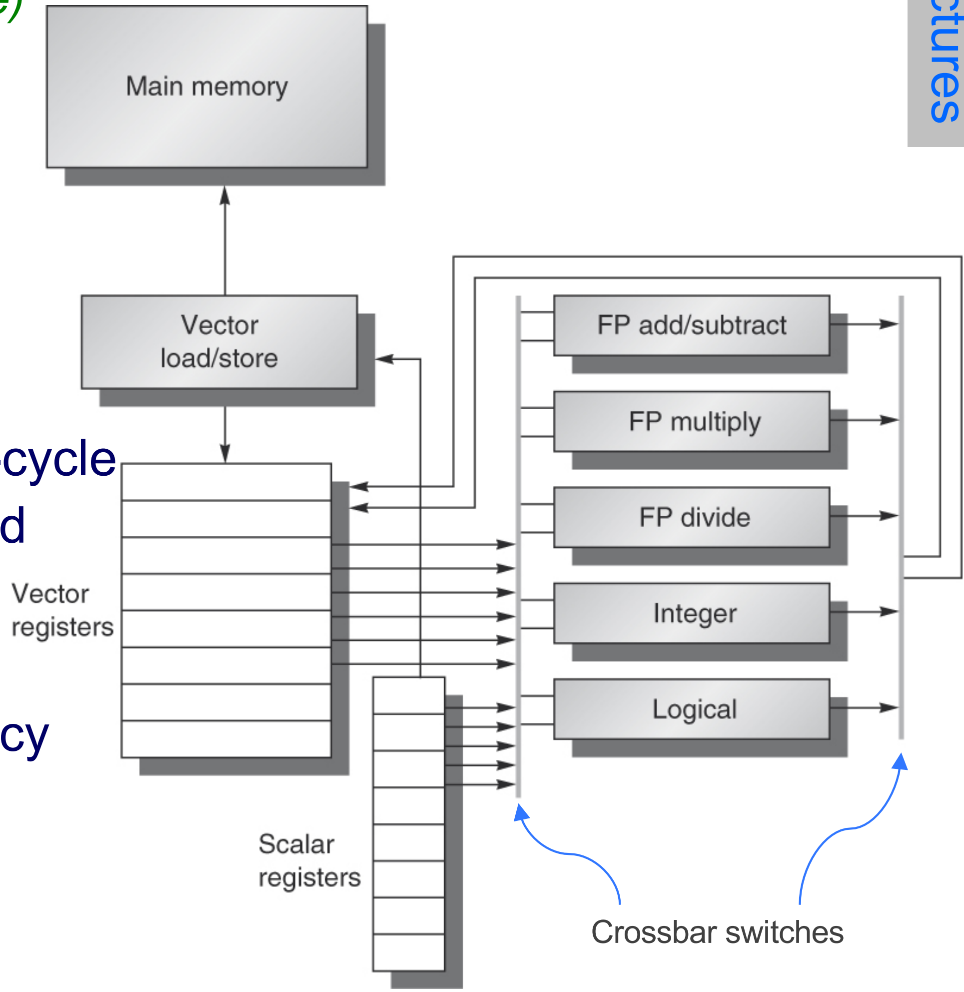
- Consider machine with 32 elements per vector register and 8 lanes:



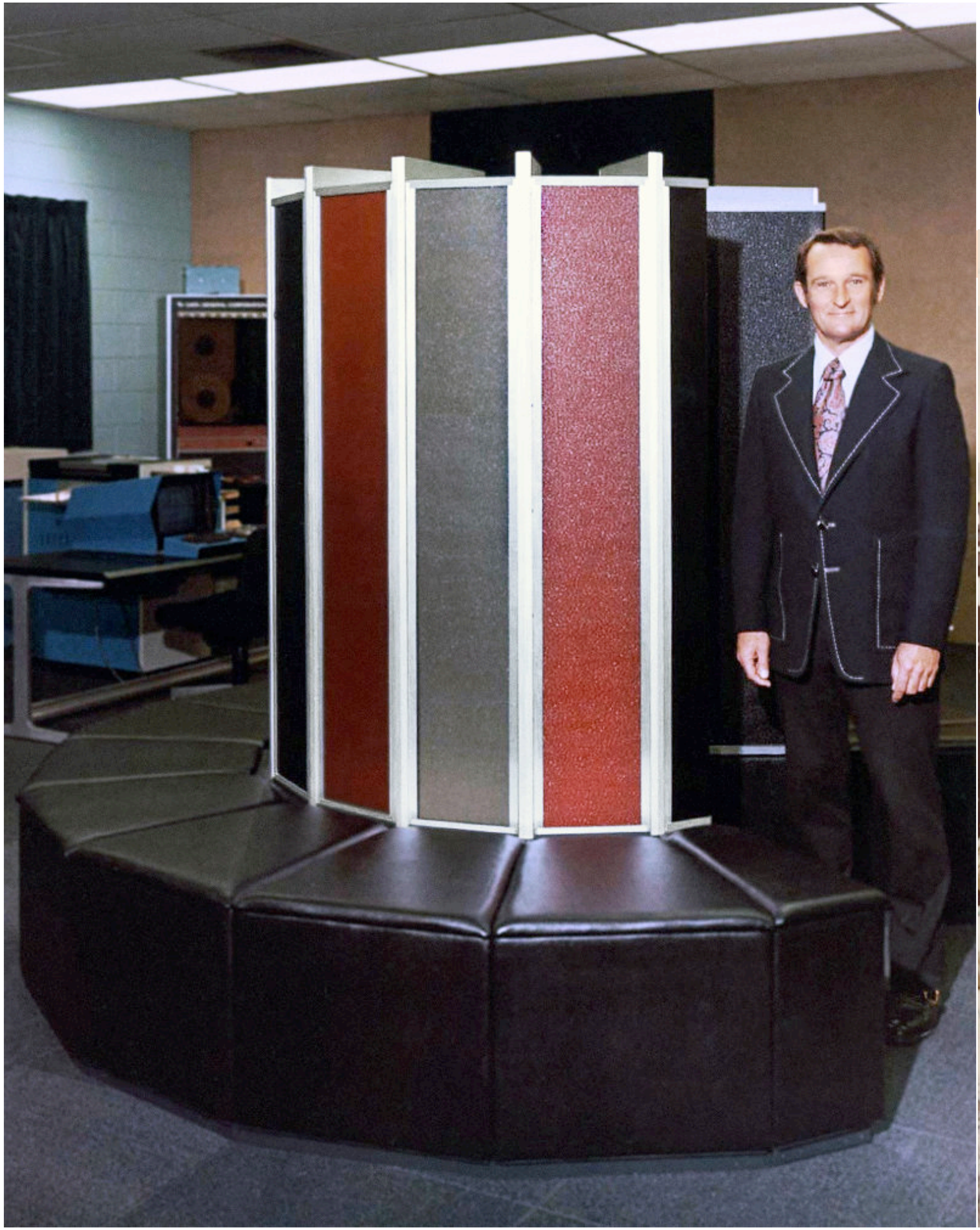
Complete 24 operations/cycle while issuing 1 short instruction/cycle

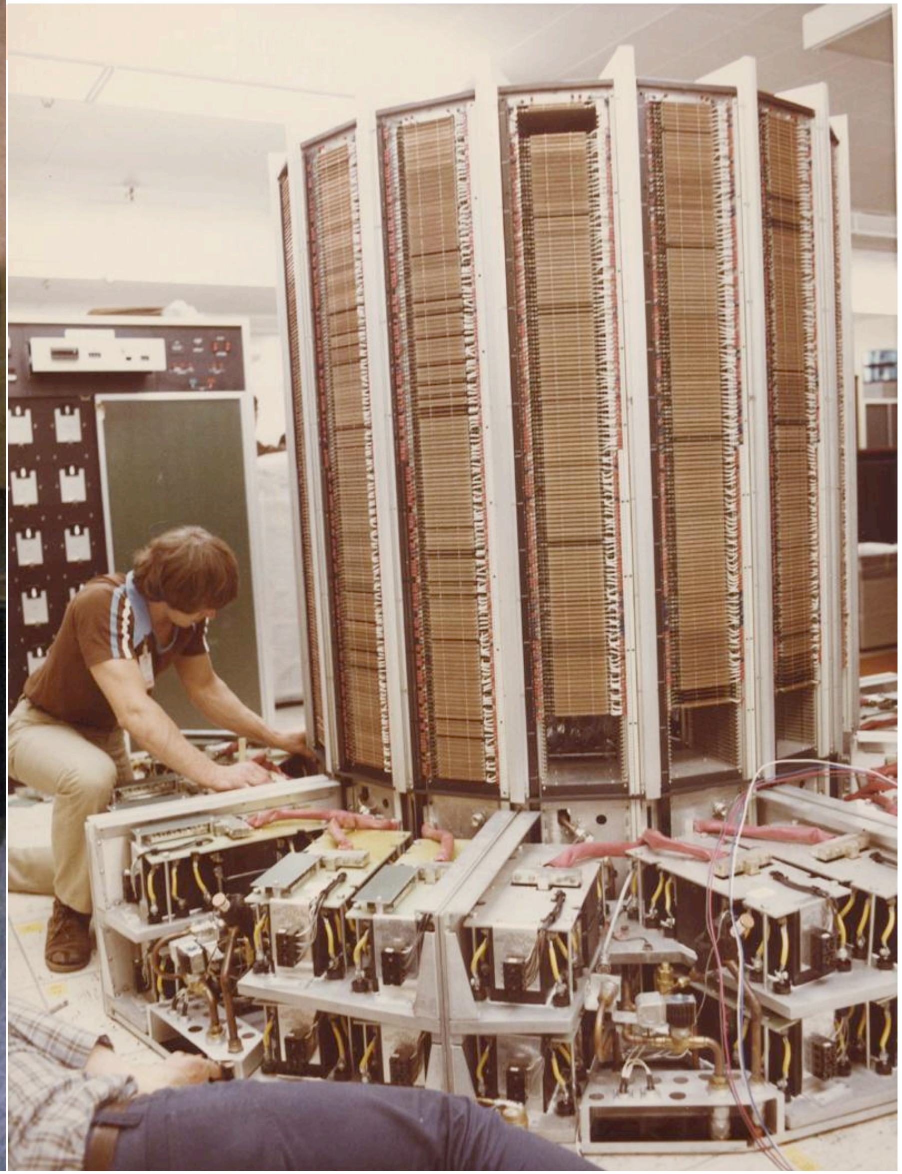
VMIPS

- Example architecture: VMIPS
  - Loosely based on Cray-1 *(next slide)*
  - Vector registers
    - Each register holds a 64-element, 64 bits/element vector
    - Register file has 16 read ports and 8 write ports
  - Vector functional units
    - Fully pipelined, new op each clock-cycle
    - Data & control hazards are detected
  - Vector load-store unit
    - Fully pipelined
    - 1 word/clock-cycle after initial latency
  - Scalar registers
    - 32 general-purpose registers
    - 32 floating-point registers



Cray-1 Supercomputer
(1976)





VMIPS Instructions

- ADDVV.D: add two vectors
- ADDVS.D: add vector to a scalar
- LV/SV: vector load and vector store from address

- Example: DAXPY *(Double-precision A x X Plus Y)*

```
L.D      F0,a       ; load scalar a
LV       V1,Rx      ; load vector X
MULVS.D  V2,V1,F0   ; vector-scalar multiply
LV       V3,Ry      ; load vector Y
ADDVV    V4,V2,V3   ; add
SV       Ry,V4      ; store the result
```

- Requires the execution of 6 instructions *versus* almost 600 for MIPS *(assuming DAXPY is operating on a vector with 64 elements)*

Vector Execution Time

- Execution time depends on three factors:
  - Length of operand vectors
  - Structural hazards
  - Data dependencies
- VMIPS functional units consume one element per clock cycle
  - Execution time is approximately the vector length
- *Convoy*
  - Set of vector instructions that could potentially execute together in one unit of time, *chime*

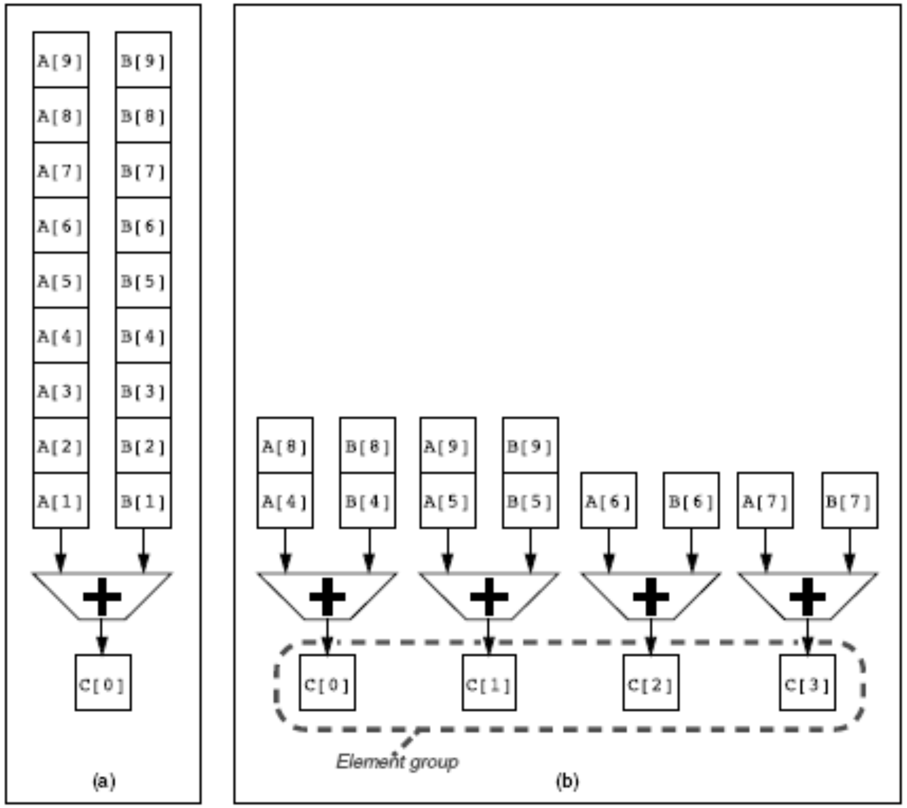
MORGAN KAUFMANN

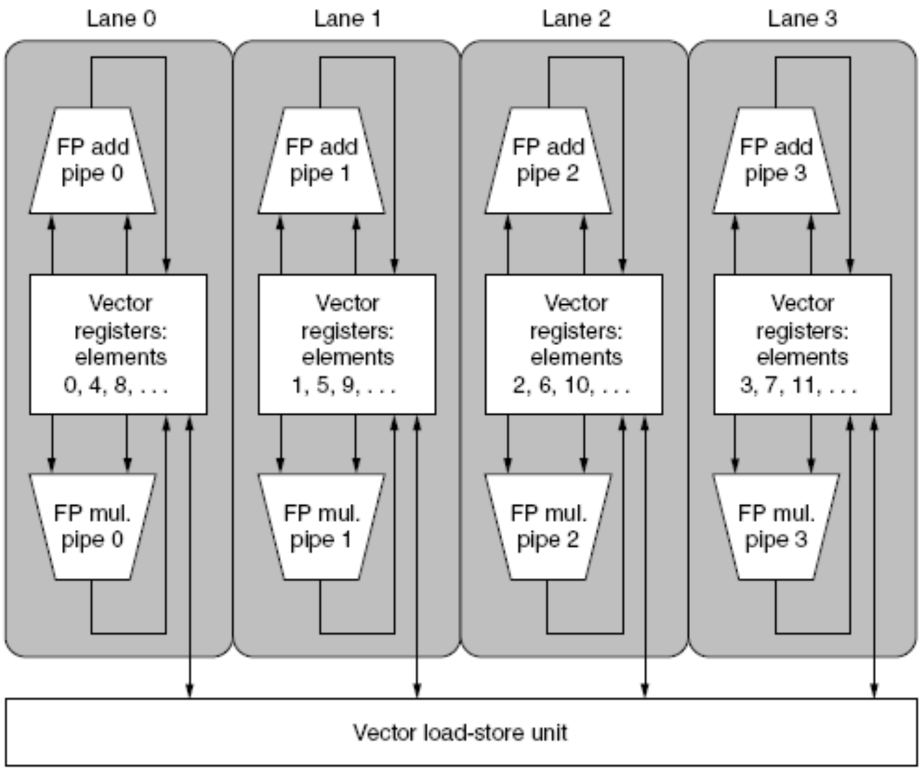
Challenges

- 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 *(1)*
  - Non-64 wide vectors *(2)*
  - IF statements in vector code *(3)*
  - Memory system optimizations to support vector processors *(4)*
  - Multiple dimensional matrices *(5)*
  - Sparse matrices *(6)*
  - Programming a vector computer *(7)*

Multiple Lanes *(1)*

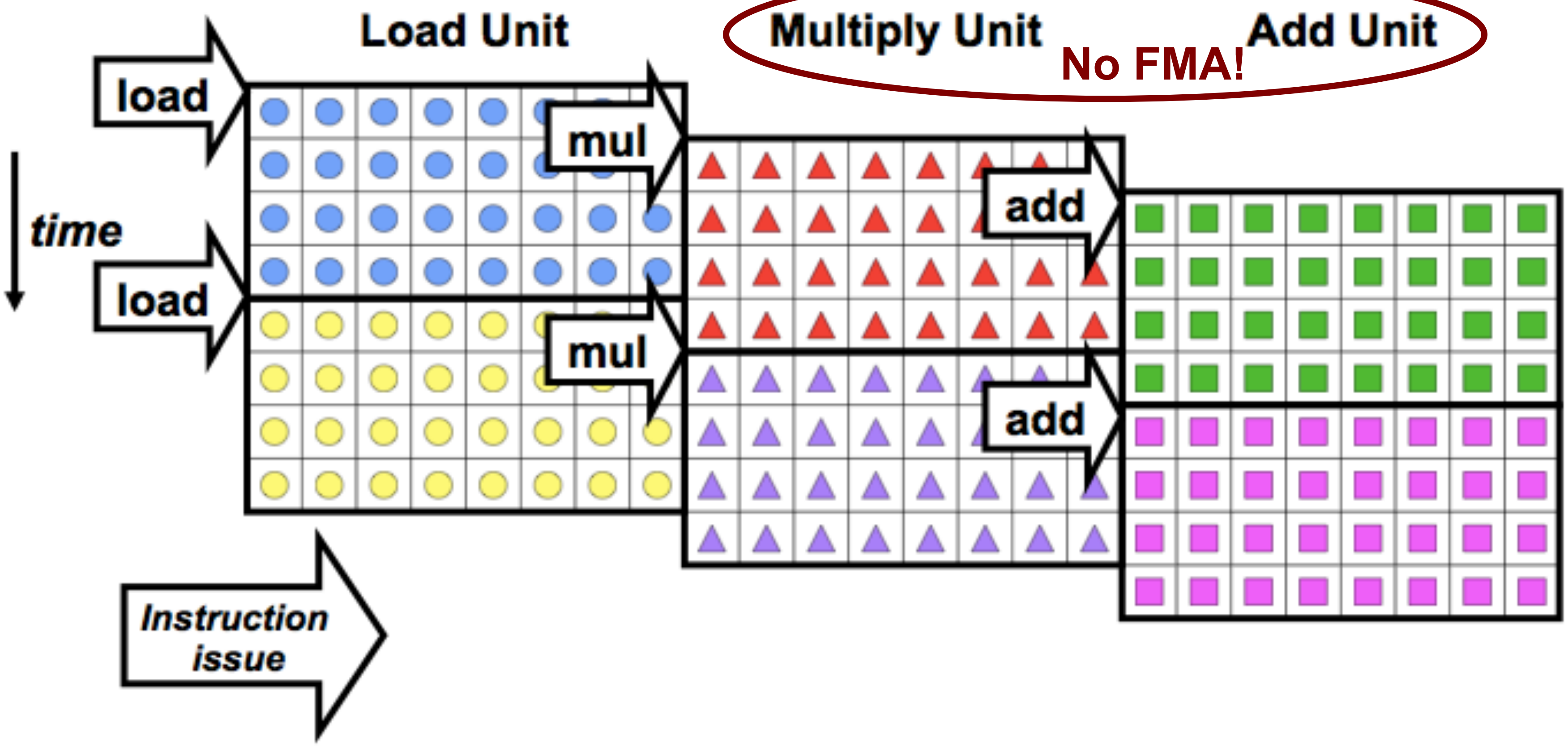
- Element *n* of vector register *A* is "hardwired" to element *n* of vector register *B*
  - Allows for multiple hardware lanes





Vector Instruction Parallelism

Can overlap execution of multiple vector instructions
- Consider machine with 32 elements per vector register and 8 lanes:

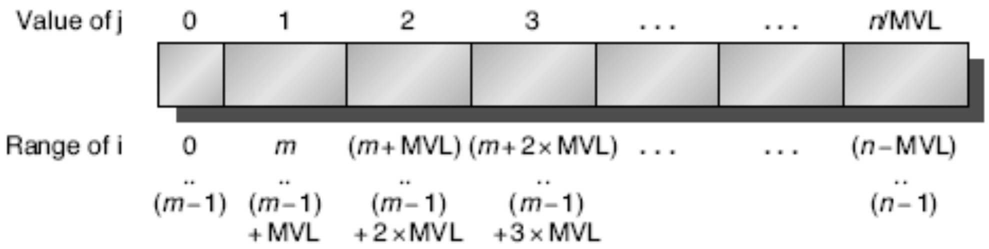


Complete 24 operations/cycle while issuing 1 short instruction/cycle

Vector Length Register *(2)*

- Handling vector length not known at compile time
- Use Vector Length Register (VLR)
- Use strip mining for vectors over the maximum length:

```
low = 0;
VL = (n % MVL); /*find odd-size piece using modulo op % */
for (j = 0; j <= (n/MVL); j=j+1) { /*outer loop*/
   for (i = low; i < (low+VL); i=i+1) /*runs for length VL*/
    Y[i] = a * X[i] + Y[i] ; /*main operation*/
   low = low + VL; /*start of next vector*/
   VL = MVL; /*reset the length to maximum vector length*/
}
```



Vector Mask Registers *(3)*

- Handling IF statements in Vector Loops:

```
for (i = 0; i < 64; i=i+1)
      if (X[i] != 0)
          X[i] = X[i] – Y[i];
```

- Use vector mask register to “disable” elements:

```
LV        V1,Rx       ;load vector X into V1
LV        V2,Ry       ;load vector Y
L.D       F0,#0       ;load FP zero into F0
SNEVS.D   V1,F0       ;sets VM(i) to 1 if V1(i)!=F0
SUBVV.D   V1,V1,V2    ;subtract under vector mask
SV        Rx,V1       ;store the result in X
```

- GFLOPS rate decreases!

Memory Banks *(4)*

- Memory system must be designed to support high bandwidth for vector loads and stores
- Spread accesses across multiple banks
  - Control bank addresses independently
  - Load or store non sequential words
  - Support multiple vector processors sharing the same memory
- Example (Cray T932, 1996; Ford acquired 1 out of 13, $39M):
  - 32 processors, each generating 4 loads and 2 stores per cycle
  - Processor cycle time is 2.167 ns, SRAM cycle time is 15 ns
  - How many memory banks needed?

Stride *(5)*

- Handling **multidimensional arrays** in Vector Architectures:

```
for (i = 0; i < 100; i=i+1) {
    for (j = 0; j < 100; j=j+1) {
    A[i][j] = 0.0;
    for (k = 0; k < 100; k=k+1)
        A[i][j] = A[i][j] + B[i][k] * D[k][j];
    }
}
```

- Must vectorize multiplication of rows of B with columns of D
- Use *non-unit stride* (in VMIPS: load/store vector with stride)
- Bank conflict (stall) occurs when the same bank is hit faster than bank busy time:
  - #banks / Least_Common_Multiple (stride, #banks) < bank busy time

Scatter-Gather *(6)*

- Handling **sparse matrices** in Vector Architectures:

```
for (i = 0; i < n; i=i+1)
    A[K[i]] = A[K[i]] + C[M[i]];
```

- Use index vector:

```
LV      Vk, Rk        ;load K
LVI     Va, (Ra+Vk)   ;load A[K[]]
LV      Vm, Rm        ;load M
LVI     Vc, (Rc+Vm)   ;load C[M[]]
ADDVV.D Va, Va, Vc;add them
SVI      (Ra+Vk), Va  ;store A[K[]]
```

Vector Programming *(7)*

- 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; 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 Ext started with 64-bit wide vectors and grew to wider vectors and more capabilities
  - Current AVX generation is 512-bit wide

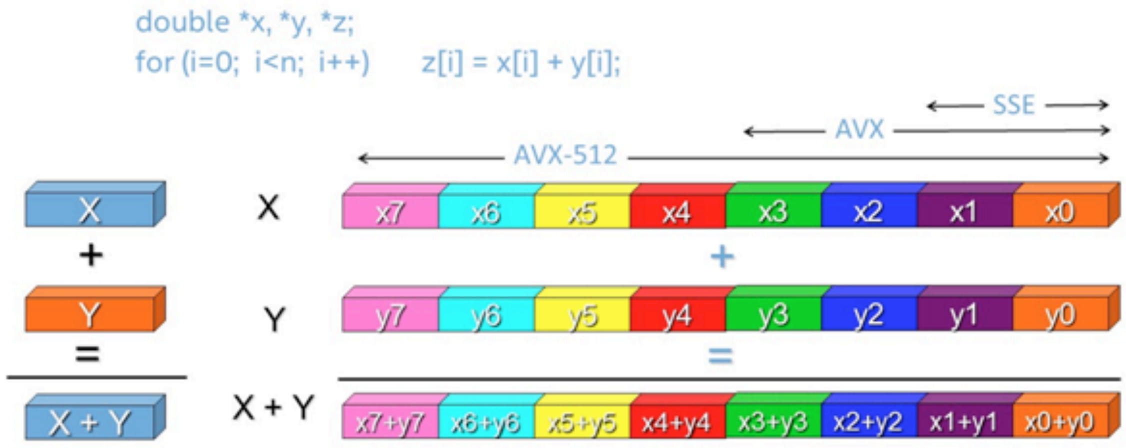


Figure 1 Scalar and vectorized loop versions with Intel® SSE, AVX and AVX-512.

- 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

Example SIMD Code

- Example DAXPY *(in MIPS SIMD)*:

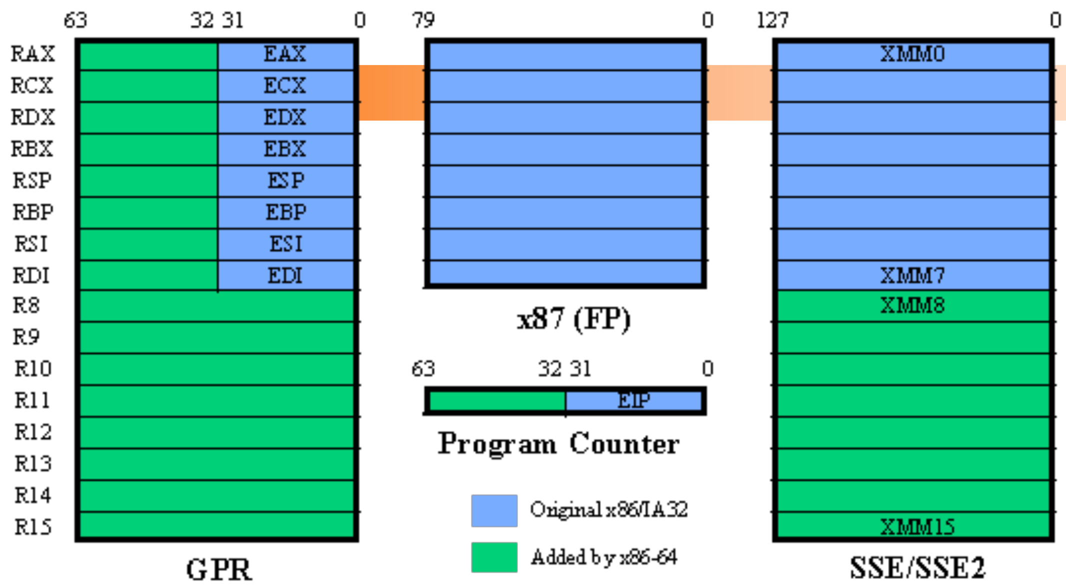
```
   L.D      F0,a         ;load scalar a
   MOV      F1, F0       ;copy a into F1 for SIMD MUL
   MOV      F2, F0       ;copy a into F2 for SIMD MUL
   MOV      F3, F0       ;copy a into F3 for SIMD MUL
   DADDIU   R4,Rx,#512   ;last address to load
Loop:
   L.4D     F4,0[Rx]     ;load X[i], X[i+1], X[i+2], X[i+3]
   MUL.4D   F4,F4,F0     ;a×X[i],a×X[i+1],a×X[i+2],a×X[i+3]
   L.4D     F8,0[Ry]     ;load Y[i], Y[i+1], Y[i+2], Y[i+3]
   ADD.4D   F8,F8,F4     ;a×X[i]+Y[i], ..., a×X[i+3]+Y[i+3]
   S.4D     0[Ry],F8     ;store into Y[i],Y[i+1],Y[i+2],Y[i+3]
   DADDIU   Rx,Rx,#32    ;increment index to X
   DADDIU   Ry,Ry,#32    ;increment index to Y
   DSUBU    R20,R4,Rx    ;compute bound
   BNEZ     R20,Loop     ;check if done
```

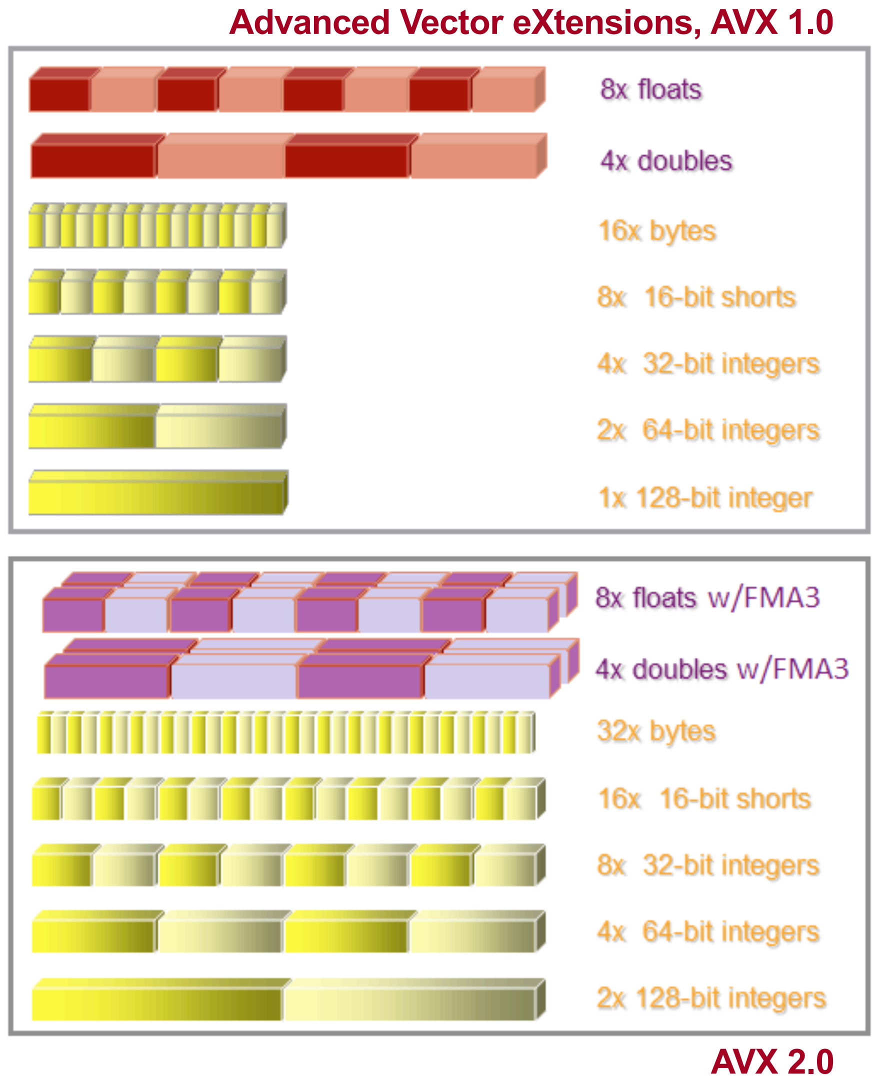
MORGAN KAUFMANN

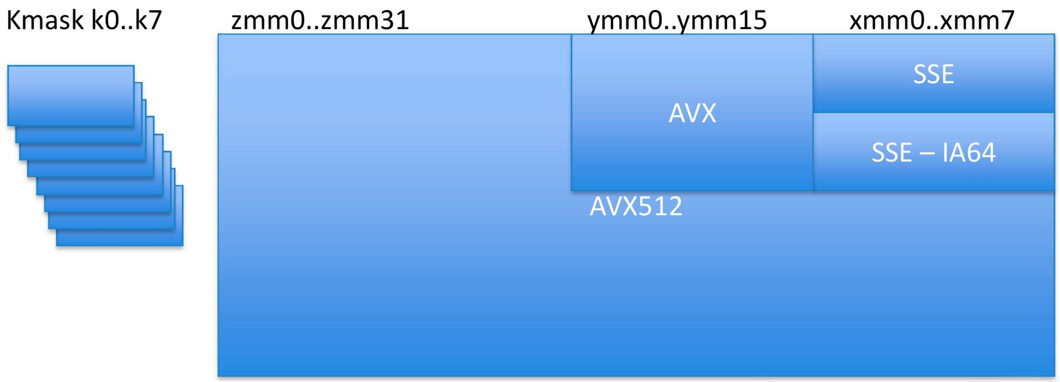
SIMD Implementations

- 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 ops or Four 64-bit fp ops (integers 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): with AVX-2
- ARM v8 (64-bit) implementations (next...)

Registers for vector processing in Intel 64







High amounts of compute need large amounts of state to compensate for memory BW
AVX512 has 8x state compared to SSE (commensurate with its 8x flops level)

Next: AVX-512 (Skylake...)

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

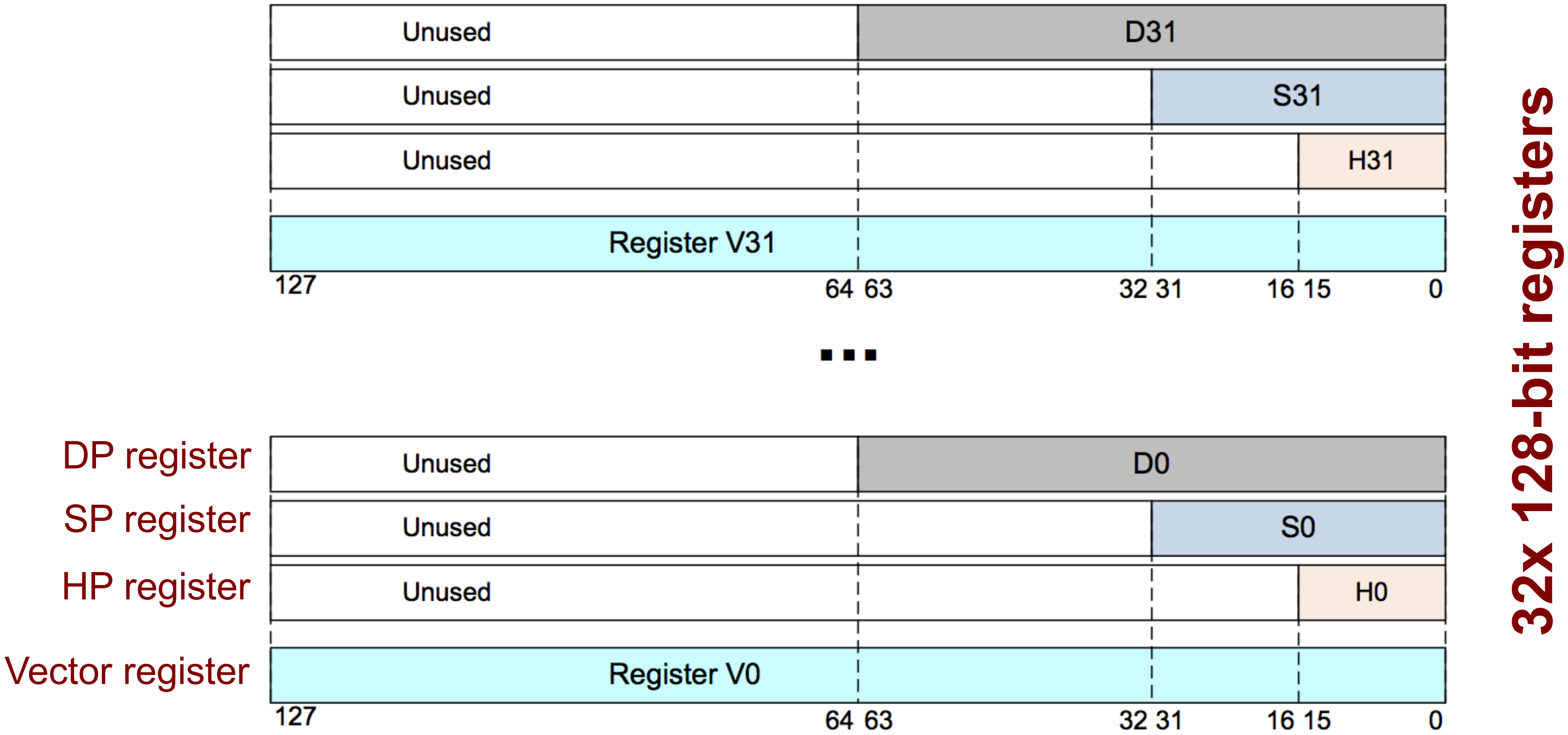


Figure 4-10 Arrangement of floating-point values

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 FP registers in ARMv8 (64-bit)

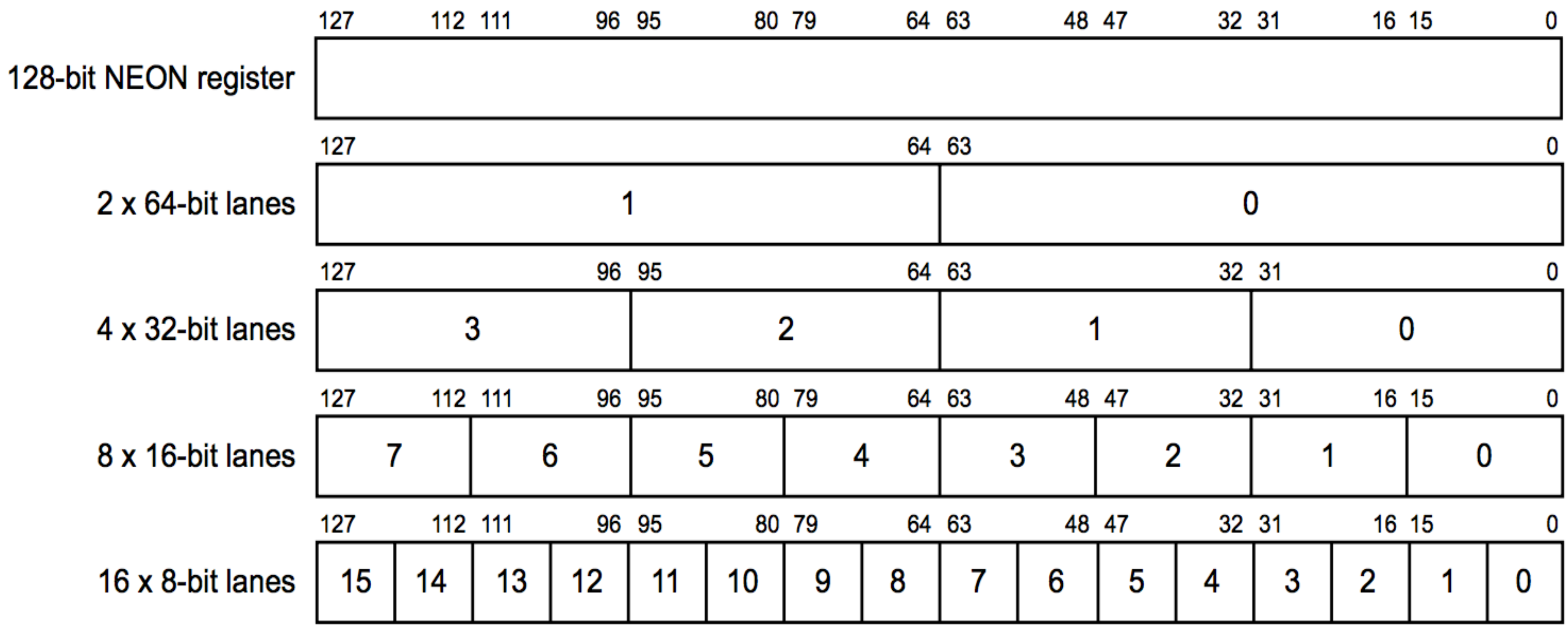
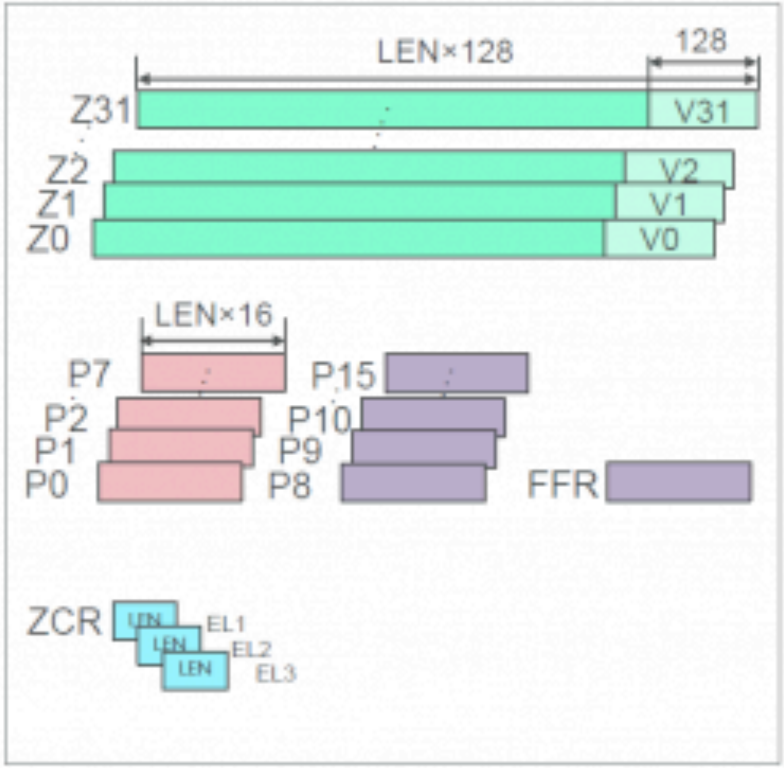


Figure 7-1 Divisions of the V register

ARMv8-A Scalable Vector Extension (SVE)

SVE architectural state

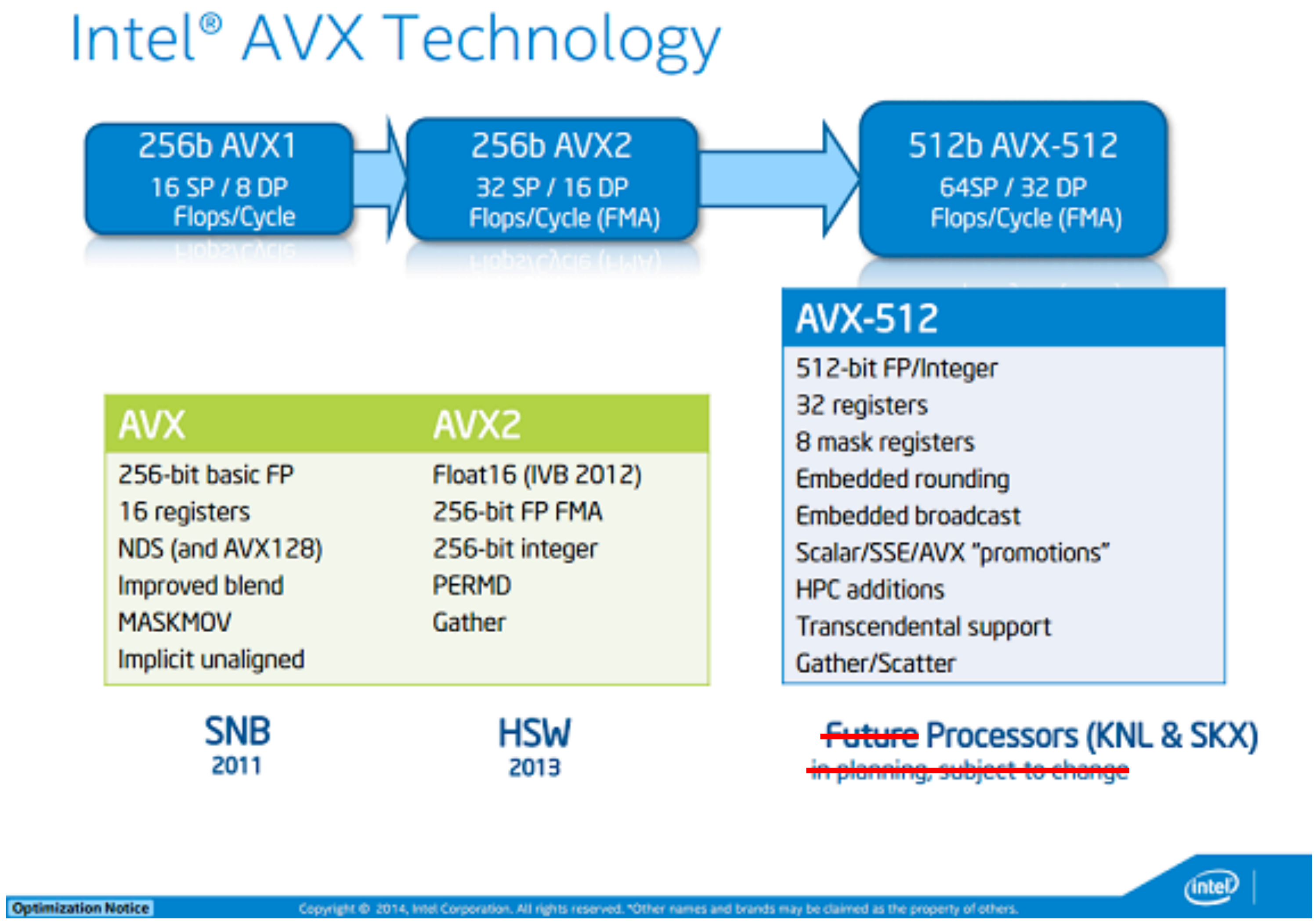
- Scalable vector registers
  - Z0-Z31 extending NEON's V0-V31
    - DP & SP floating-point
    - 64, 32, 16 & 8-bit integer
- Scalable predicate registers
  - P0-P7 lane masks for ld/st/arith
  - P8-P15 for predicate manipulation
  - FFR *first fault register*
- Scalable vector control registers
  -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



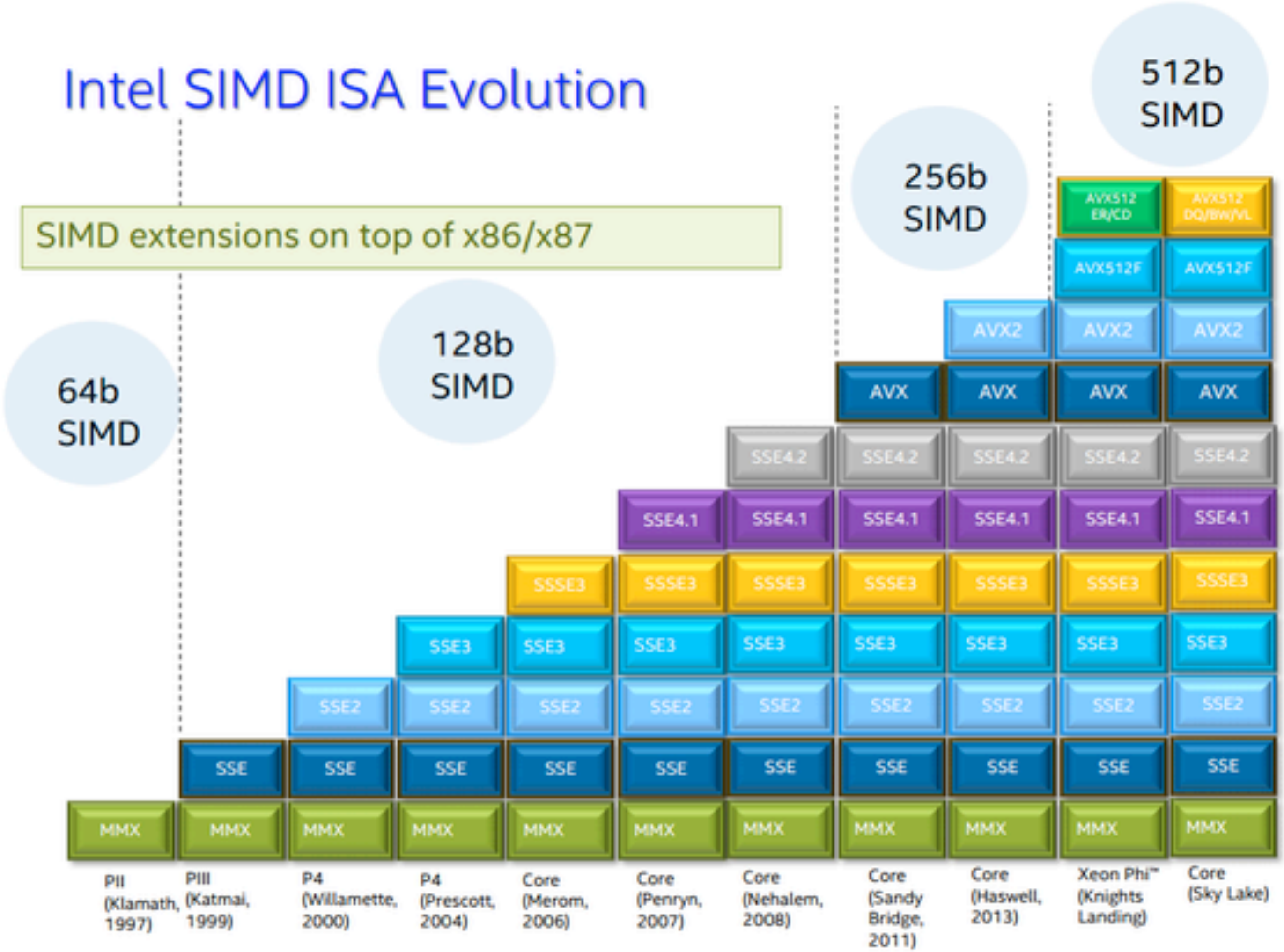
5 ©ARM 2016

ARM

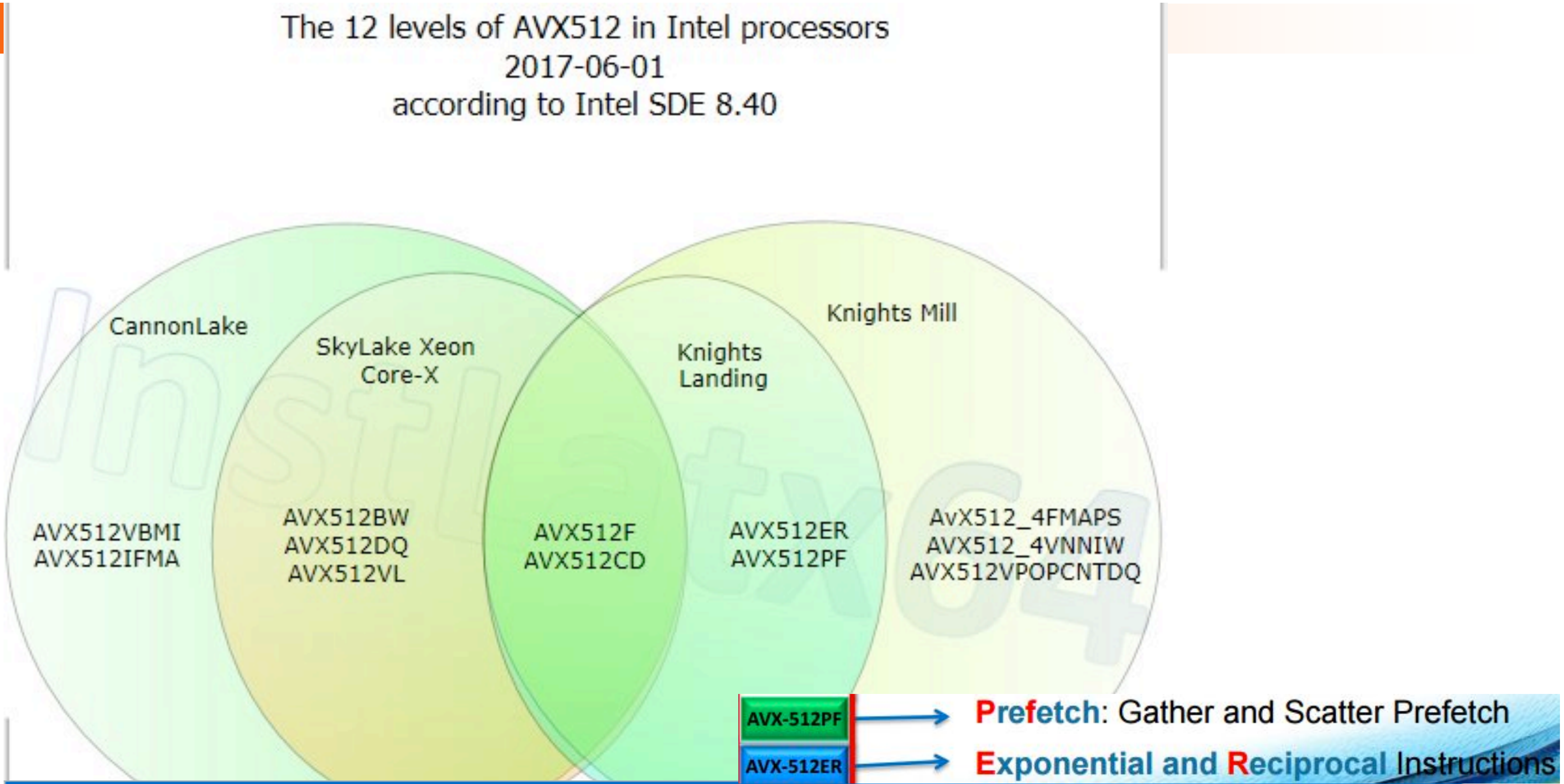
Intel evolution to the AVX-512



Intel SIMD ISA evolution

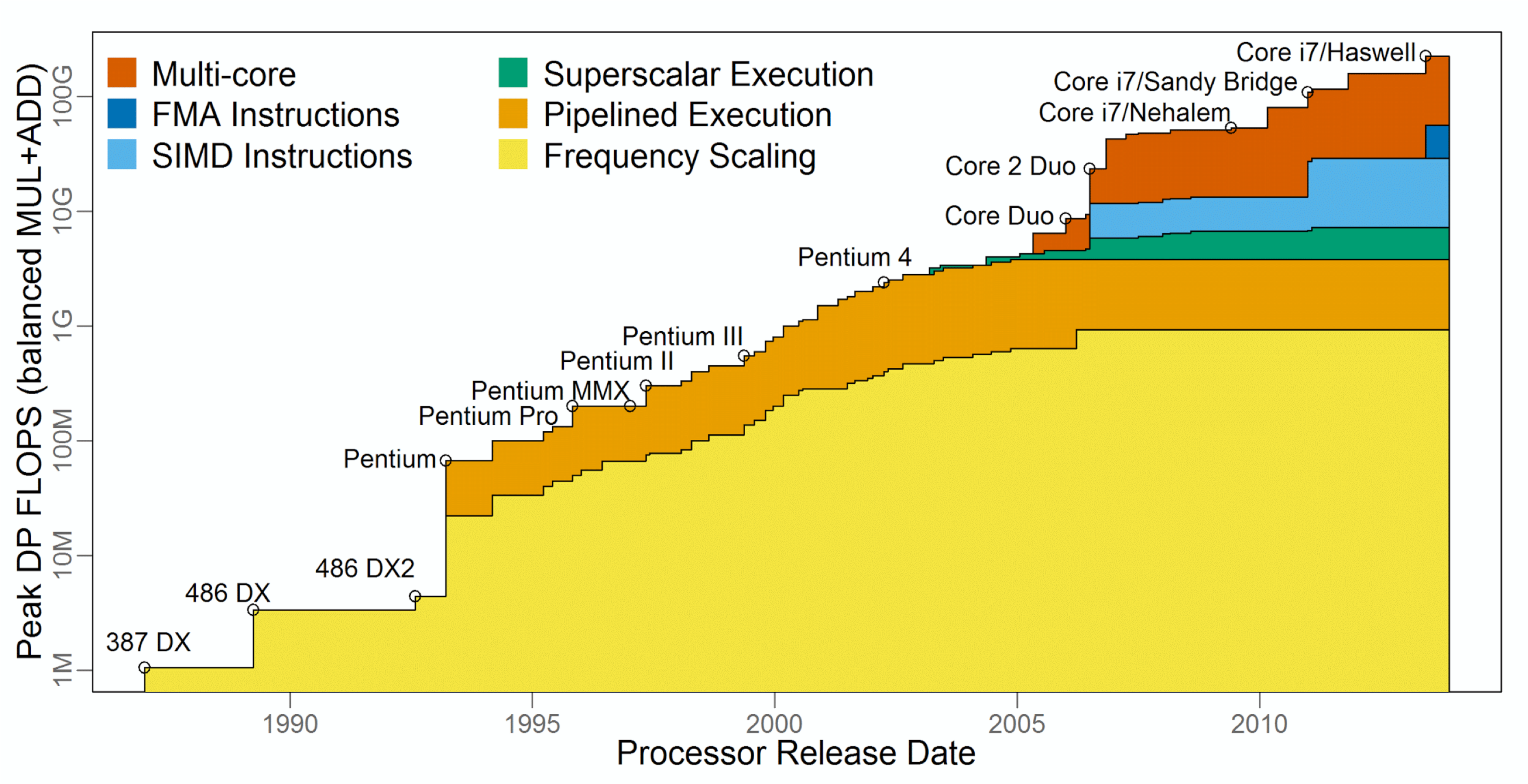


The AVX-512 across Intel devices



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

Additional features in Intel x86



from Marat Dukhan, 2014

https://developer.arm.com/technologies/neon

https://www.anandtech.com/show/11441/dynamiq-and-arms-new-cpus-cortex-a75-a55

https://www.anandtech.com/show/11213/arm-launches-dynamiq-biglittle-to-eight-cores-per-cluster

https://www.anandtech.com/show/10596/hot-chips-2016-nvidia-discloses-tegra-parker-details

https://www.anandtech.com/show/11185/nvidia-announces-jetson-tx2-parker

http://www.fudzilla.com/news/41404-nvidia-parker-has-denver-2-with-pascal

http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.den0024a/CFHEIDJH.html

http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.den0024a/BABIGHEB.html