UCLouvain
Institut de recherche en mathématique et physique
Centre de Cosmologie, Physique des Particules et Phénoménologie

MCnet

Going Parallel

Going Parallel
Going Parallel
Going Parallel
Going Parallel

Olivier Mattelaer

In Collaboration with

Taylor Childers, Nathan Nichols, Walter Hopkins

Argonne NATIONAL LABORATORY

Laurence Field, Stefan Roiser, David Smith, Andrea Valassi

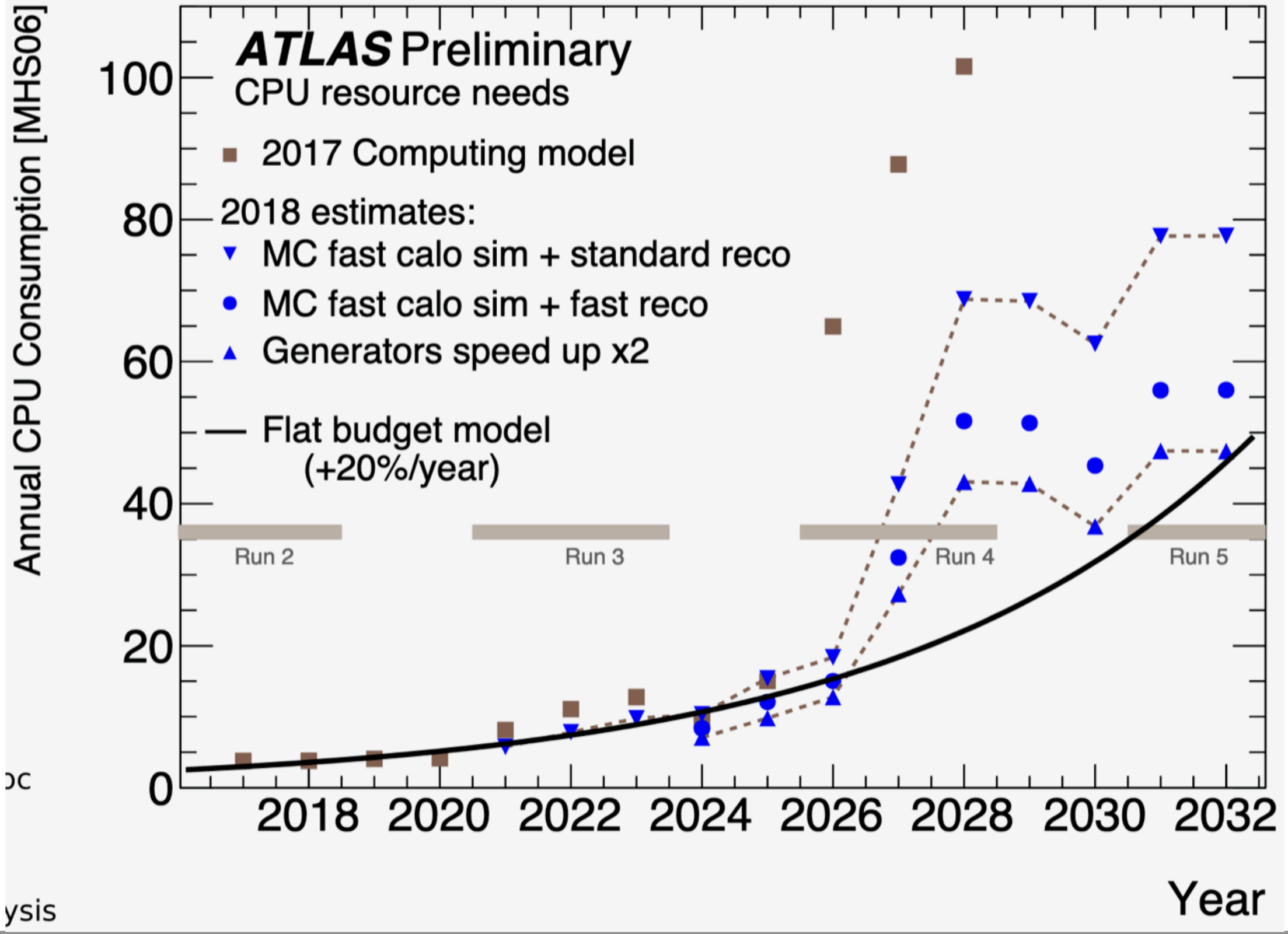
comp-ph 2106.12631

CERN

Goal of today

- Hardware theory:
  - Learn SIMD/vectorization
  - Learn the difference between CPU/GPU
- Two use case:
  - Matrix-Element evaluation
  - Phase-Space integration

Why speed is important?

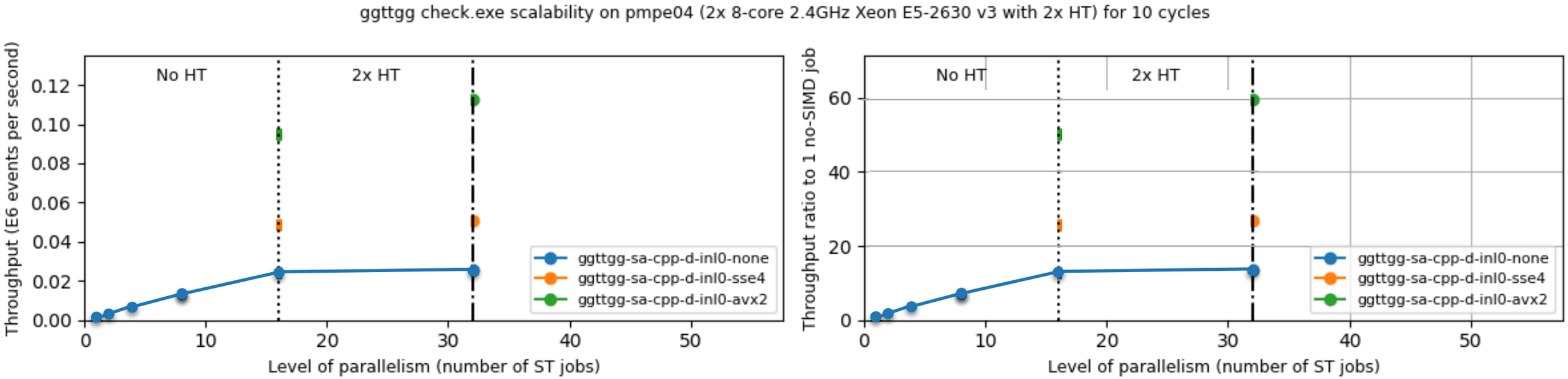


Going Parallel

- Multi-processes
  - ➡ For boringly parallel code
  - ➡ Cross-section computation is in this category
- OpenMP
  - ➡ Thread parallelism with shared memory
- OpenMPI
  - ➡ Multi-socket parallelism with communication protocol

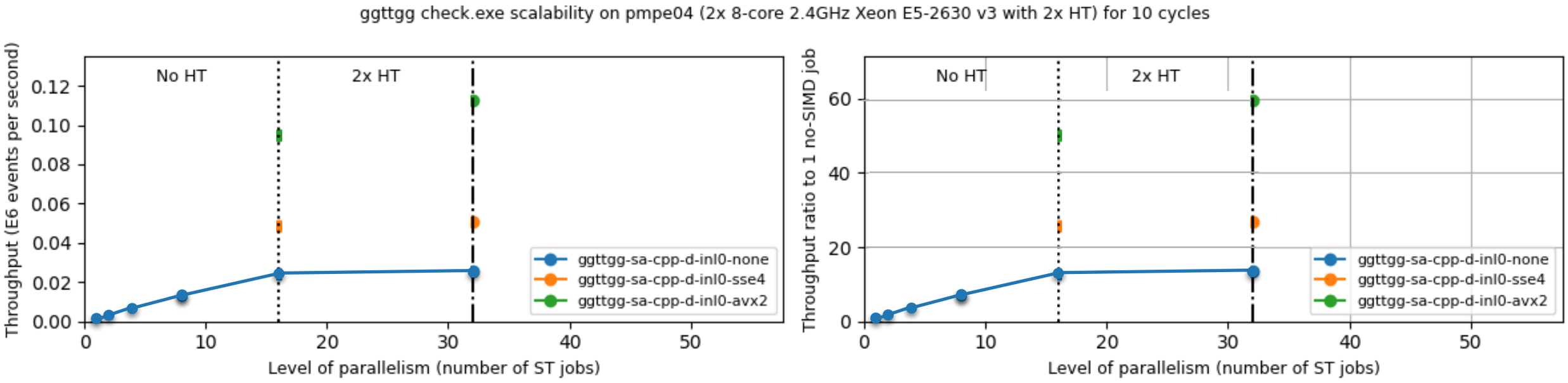
➡The more you use the less you wait

Haswell Computer



- This is for a “quite complex” matrix-element
- Twice the same plot (different normalization)
- X-axis number of process submitted on the node
  - Multi-process mode (borrongly parallel)
- Machine has 16 core
  - Above 16 the hyper-threading is used

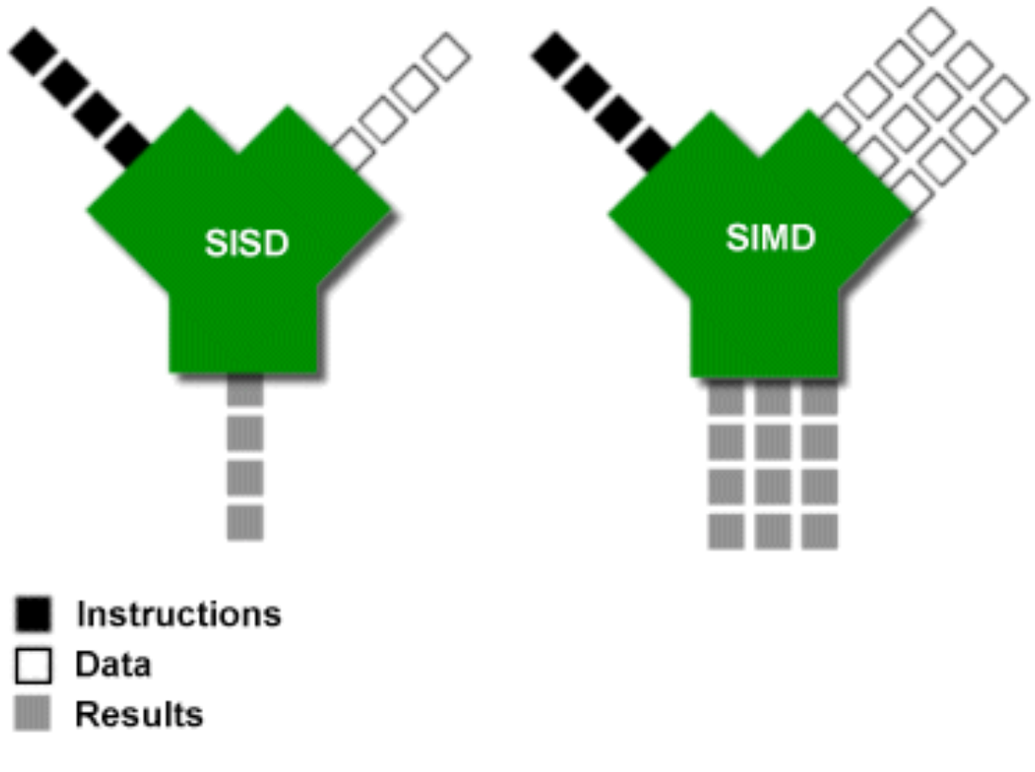
Haswell Computer



- Blue curve: “Old code”
- Not perfect scaling with number of thread use (x14 not x16)
  - Down-clocking of the cpu
  - Small impact of hyper-threading (at best 10%)

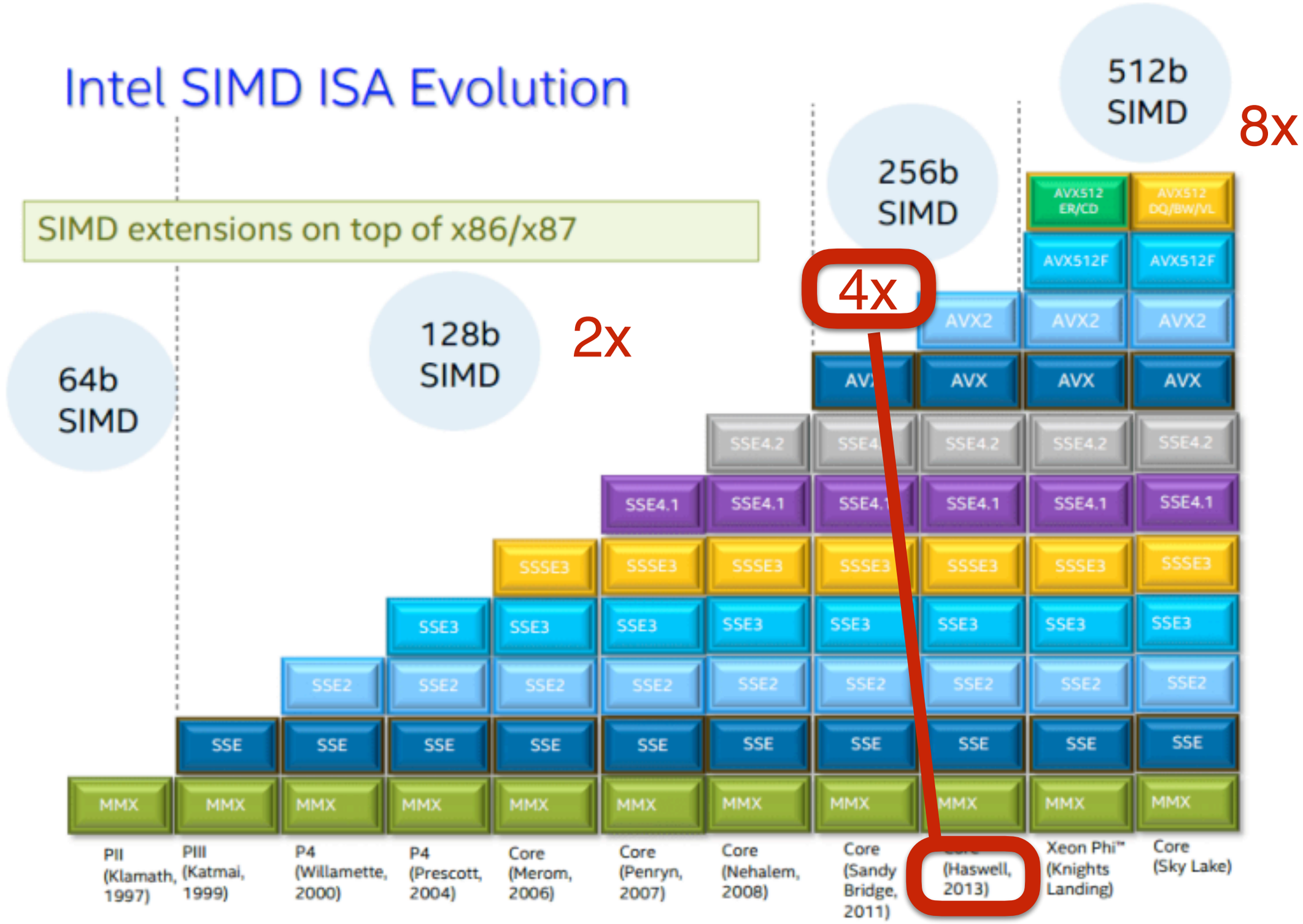
Data parallelism on cpu

Data parallelism



- SIMD (Single Instruction Multiple data):
  - ➡ Also named code vectorisation
  - ➡ Need dedicated memory pattern to allow it
  - ➡ Speed-up on the same hardware
    - ? All CPU have it

How much can you gain?



Computation

Calculate a given process (e.g. gluino pair)

- Determine the production mechanism

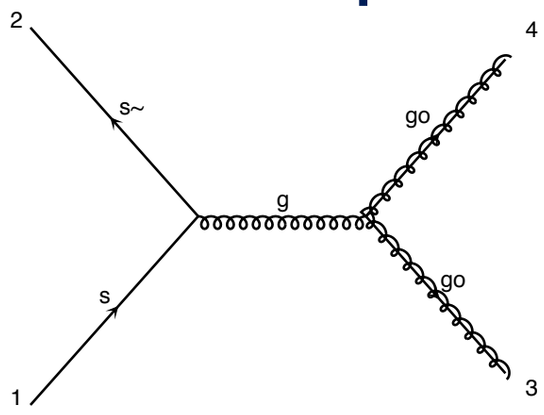


diagram 1 QCD=2, QED=0

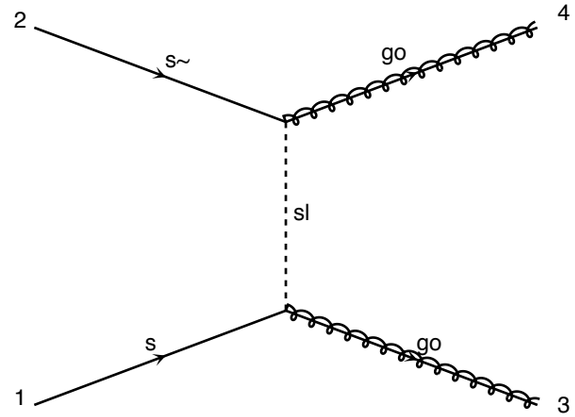


diagram 2 QCD=2, QED=0

- Evaluate the matrix-element

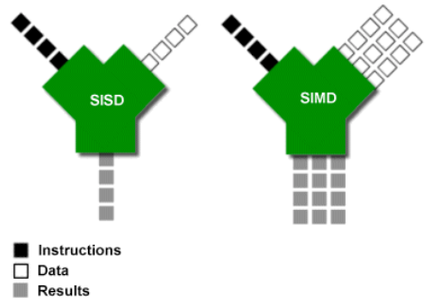
$$|\mathcal{M}|^2$$

➡Need Feynman Rules!

- Phase-Space Integration

$$\sigma = \frac{1}{2s} \int |\mathcal{M}|^2 d\Phi(n)$$

Implementation



- Helicity Amplitude Formalism
  - No helicity recycling so far (can be done)
- Parallelization at the event level
  - Evaluate N events simultaneously
  - Avoid ANY code divergence
  - Momenta (and the rest) set in AOSOA

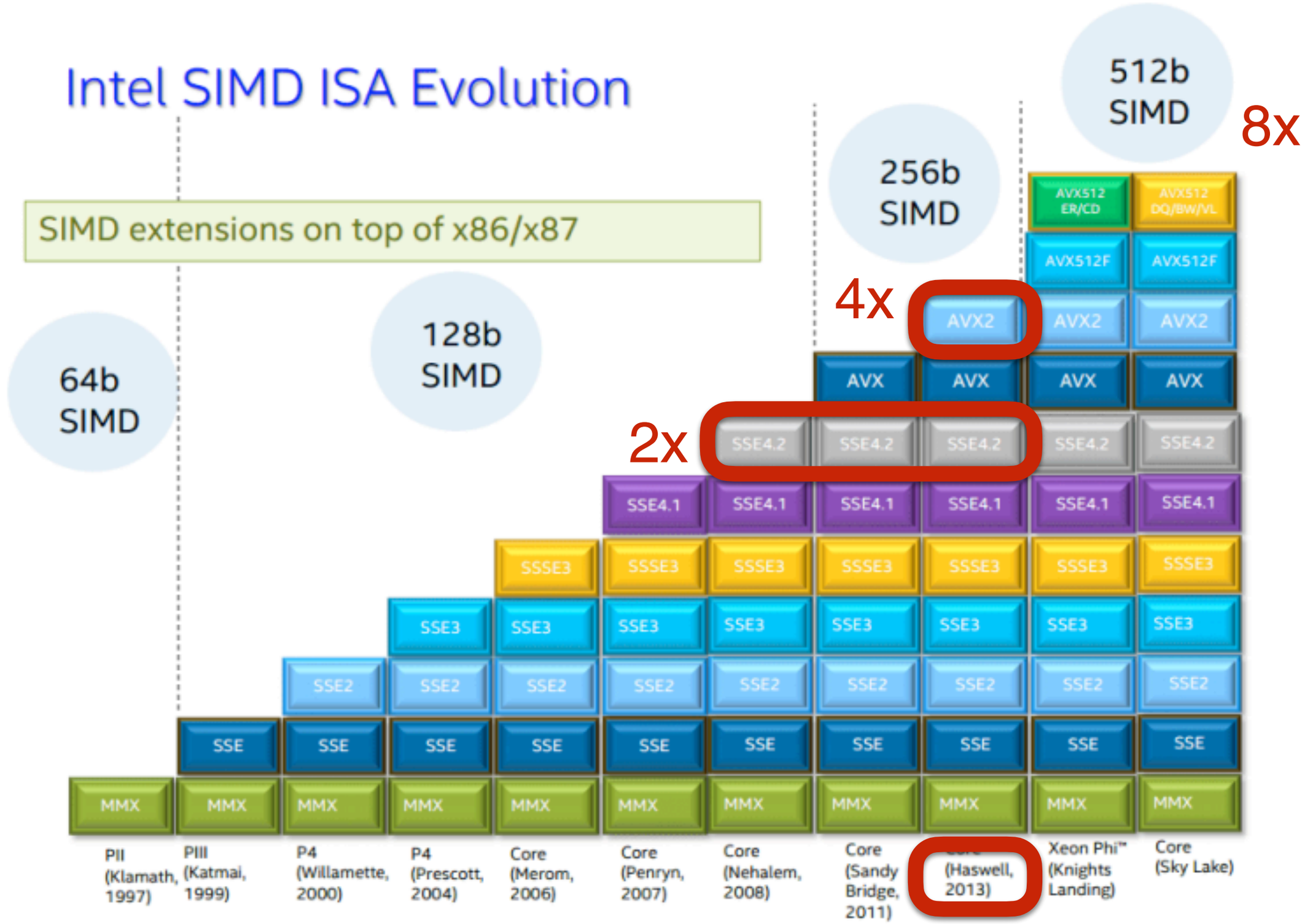
$$|E^1|p_x^1|p_y^1|P_z^1| \rightarrow |E^1|E^2|E^3|E^4|p_x^1|p_x^2|p_x^3|p_x^4|p_y^1|p_y^2|p_y^3|p_y^4|P_z^1|P_z^2|P_z^3|P_z^4|$$

  - Code in C++ with dedicated object
  - SIMD obtained by overwriting standard operation

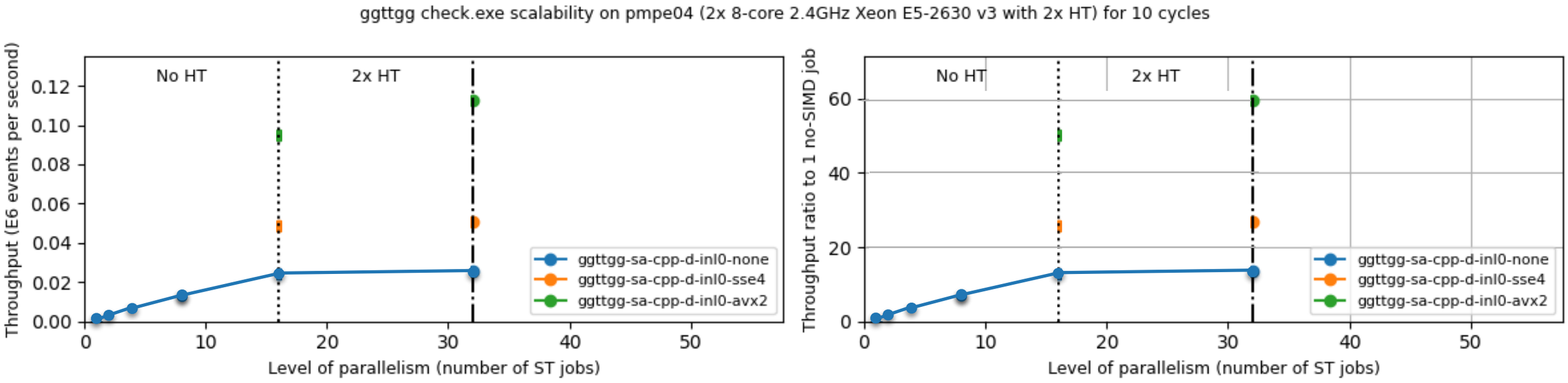
$$EE - p_x p_x - p_y p_y - p_z p_z \rightarrow EE - p_x p_x - p_y p_y - p_z p_z$$

All Multiplication/addition: hides a for loop
(Using vectorised code extension)

Haswell computer

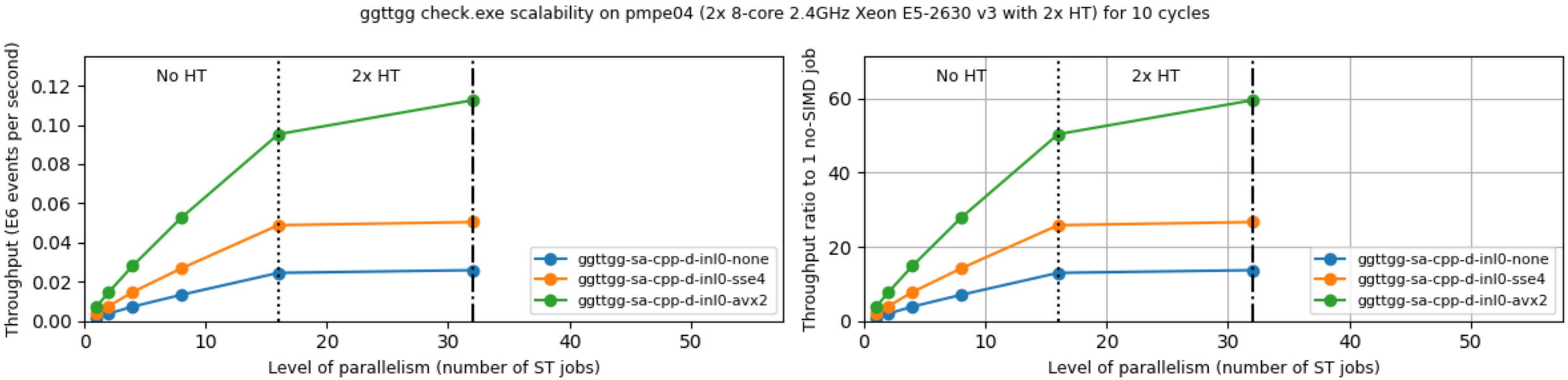


Haswell Computer



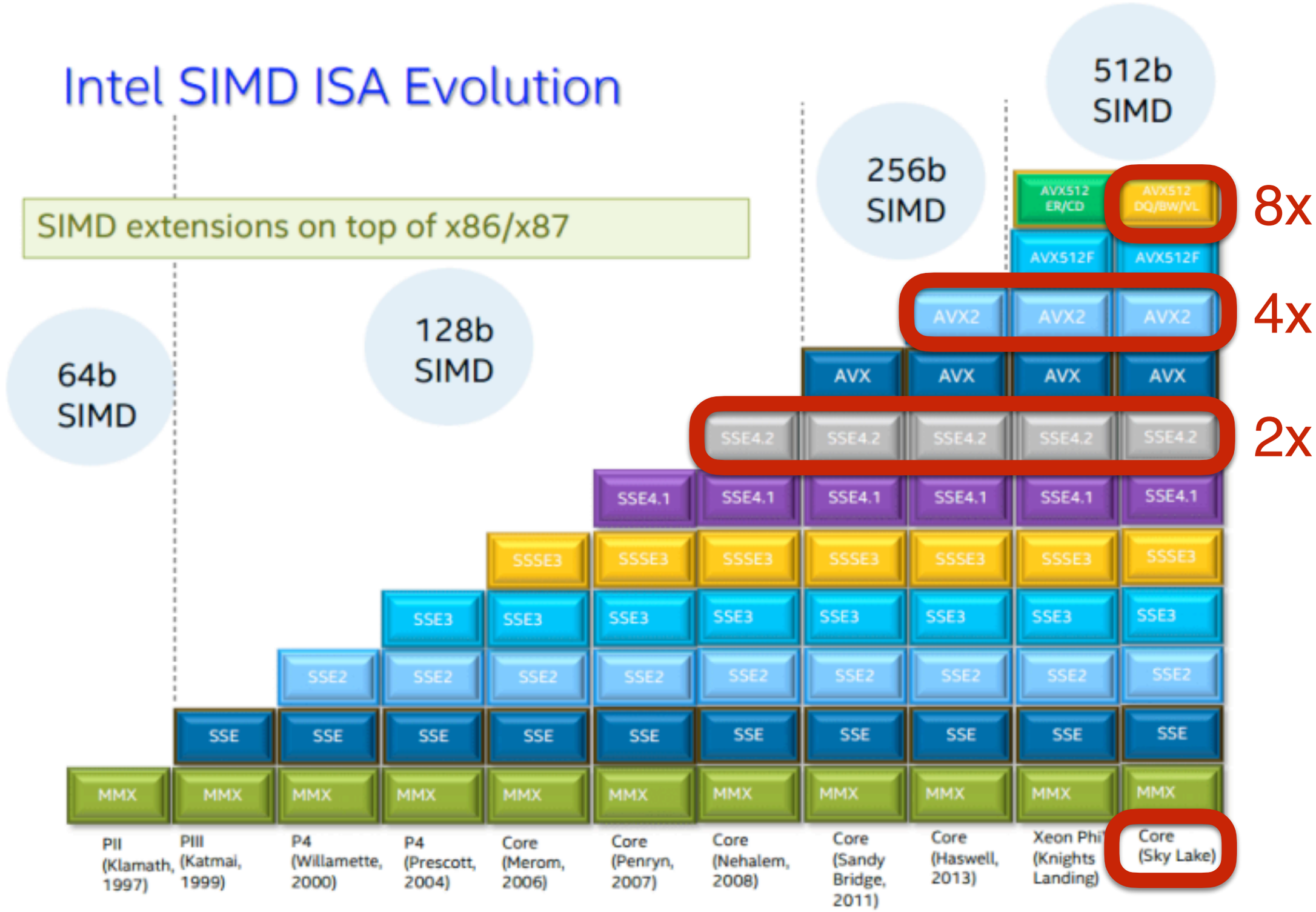
- This was the status without SIMD

Haswell Computer

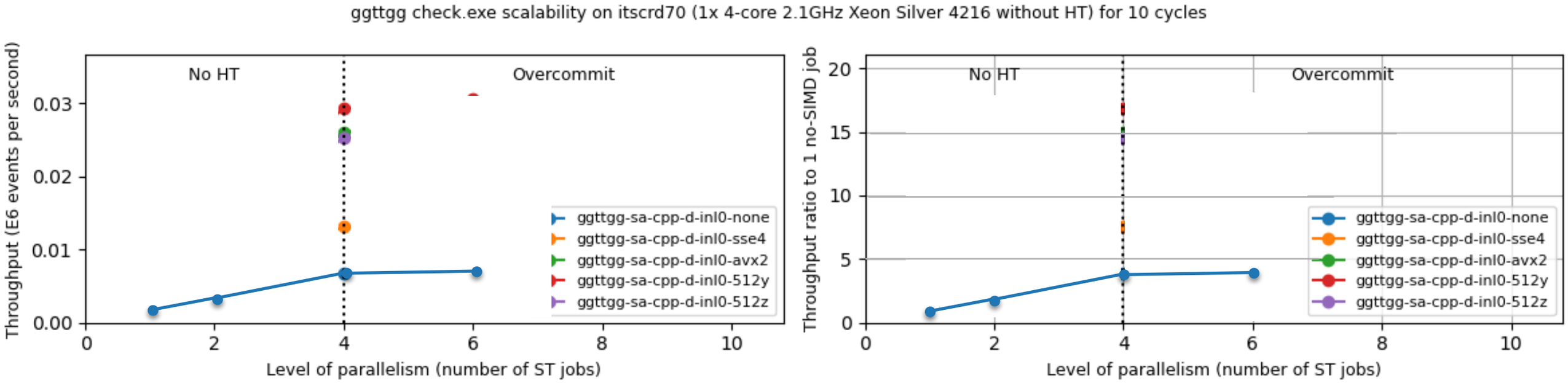


- Orange line (same with SSE4: expected speed-up 2x)
  - Expectation met
- Green line (same with AVX2: expected speed-up 4x)
  - Expectation met
  - HT helps significantly in this case
    - Hide memory latency (?)

Skylake Computer

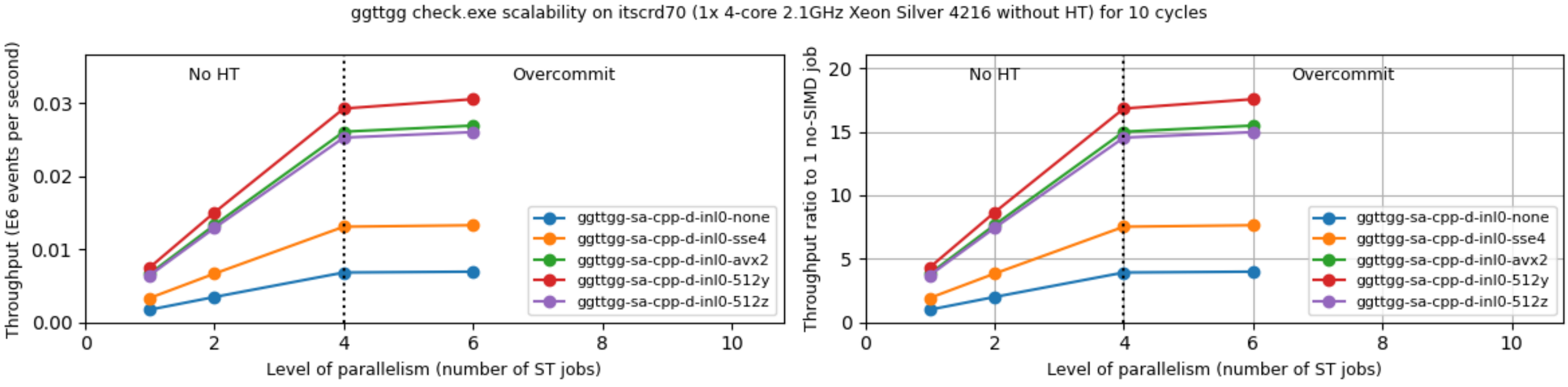


Skylake Computer (4 core)



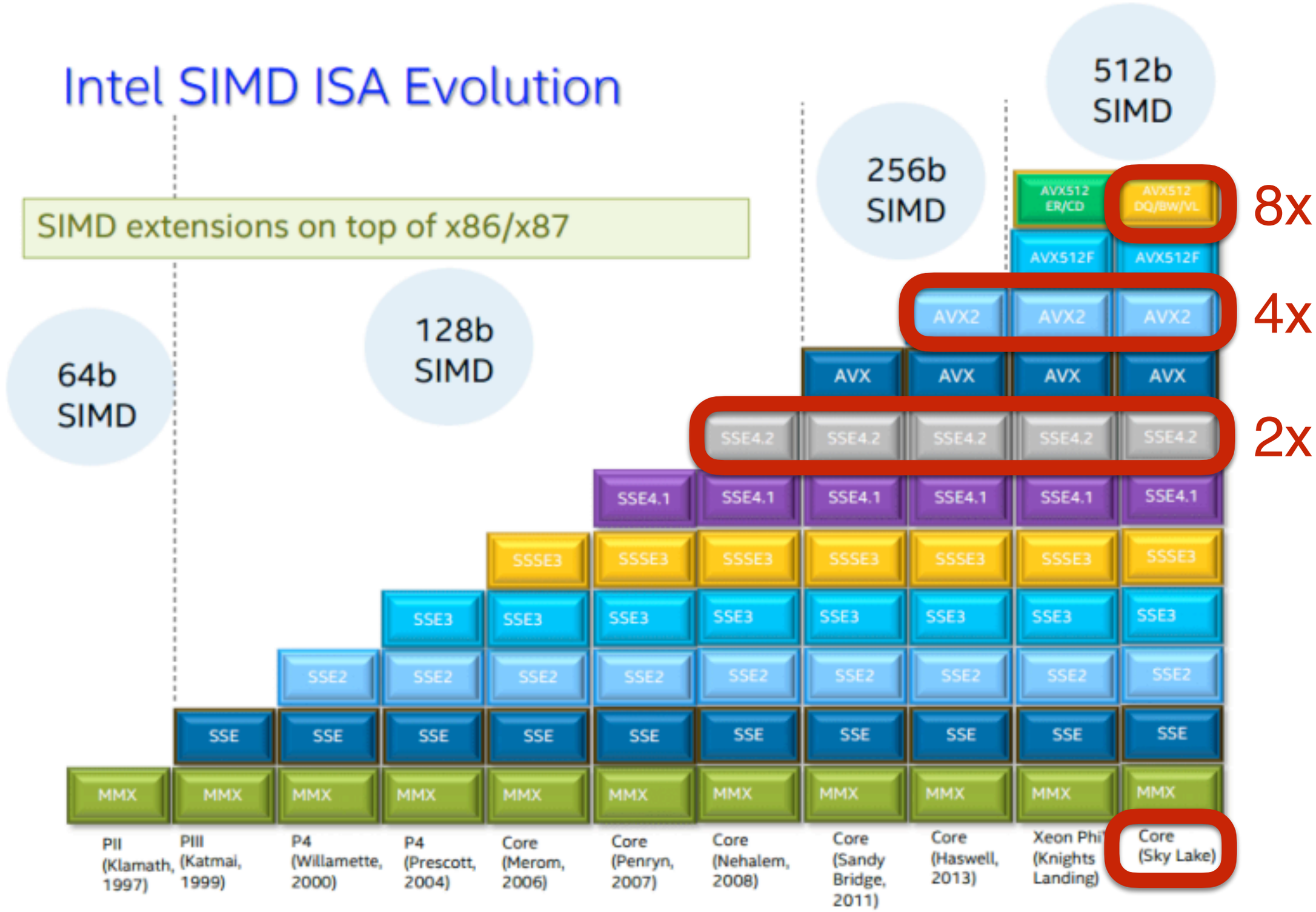
- Hyper threading was off.
- Only four core on that node
- Blue curve: Old code
  - Process parallelism close to perfect
    - Less down-clocking here (only 4 core)

Skylake Computer (4 core)

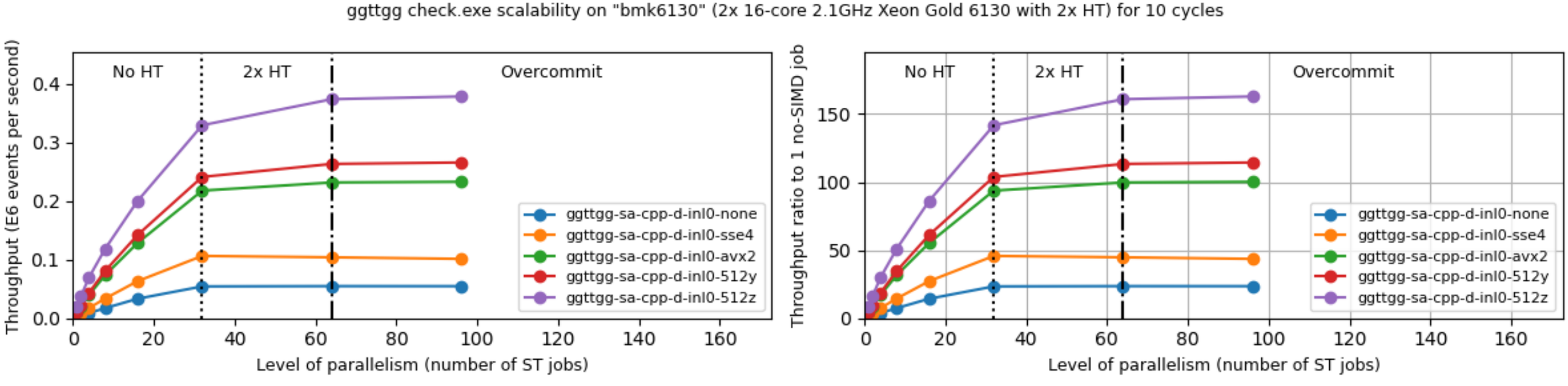


- Orange line: same with SSE4
  - Expected: 2x
  - Exception met
- Green line: same with AVX2
  - Expected 4x
  - Exception met
- Red line: AVX512y
  - Expected: 8x
  - Exception failed (4.5x)
- Purple line: AVX512z
  - Expected 8x
  - Exception failed (3.8x)

CascadeLake Computer



CascadeLake Computer (32 core)

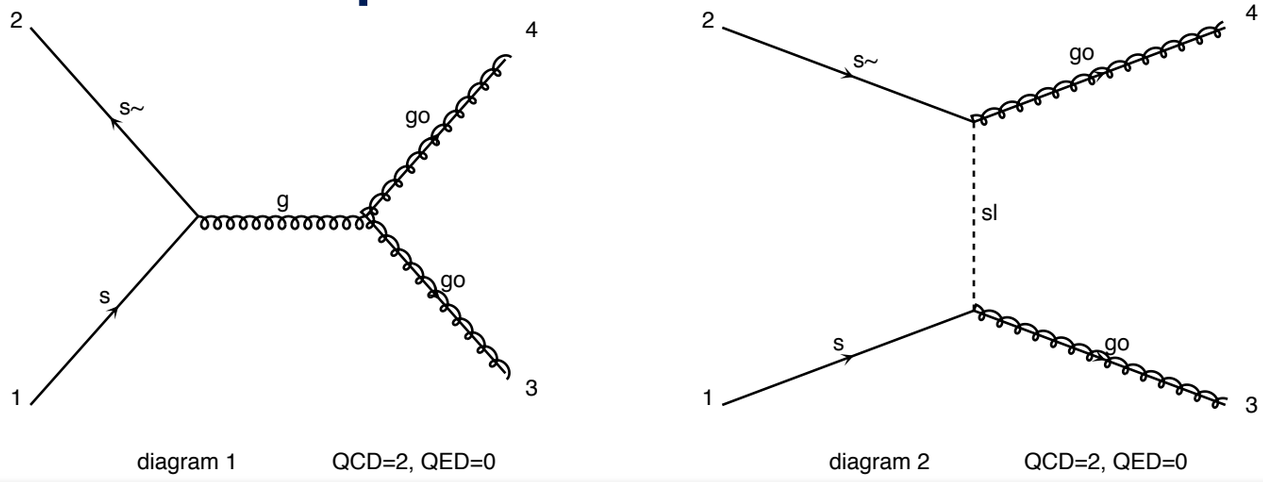


- Orange line: same with SSE4
  - Expected: 2x
  - Exception met
- Green line: same with AVX2
  - Expected 4x
  - Exception met
- Red line: AVX512y
  - Expected: 8x
  - Exception failed (4.5x)
- Purple line: AVX512z
  - Expected 8x
  - Exception failed (6x)
    - Memory latency?
    - down-cloacking?

Computation

Calculate a given process (e.g. gluino pair)

- Determine the production mechanism



- Evaluate the matrix-element

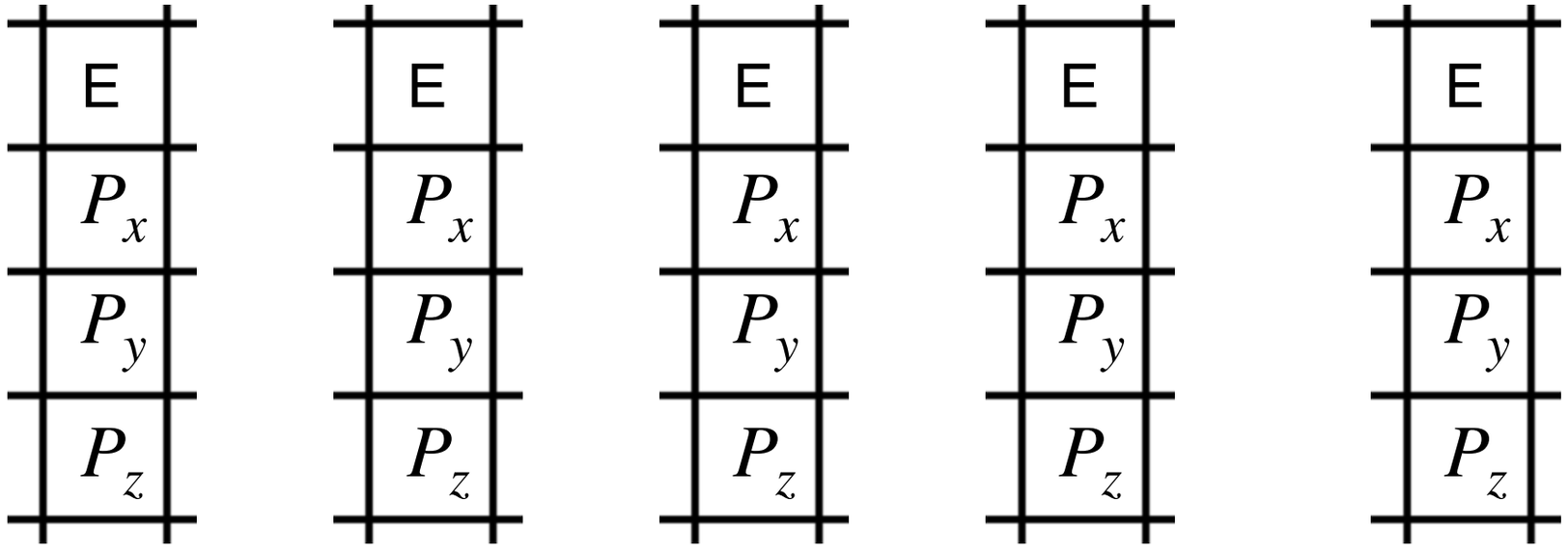
$$|\mathcal{M}|^2$$

➡Need Feynman Rules!

- Phase-Space Integration

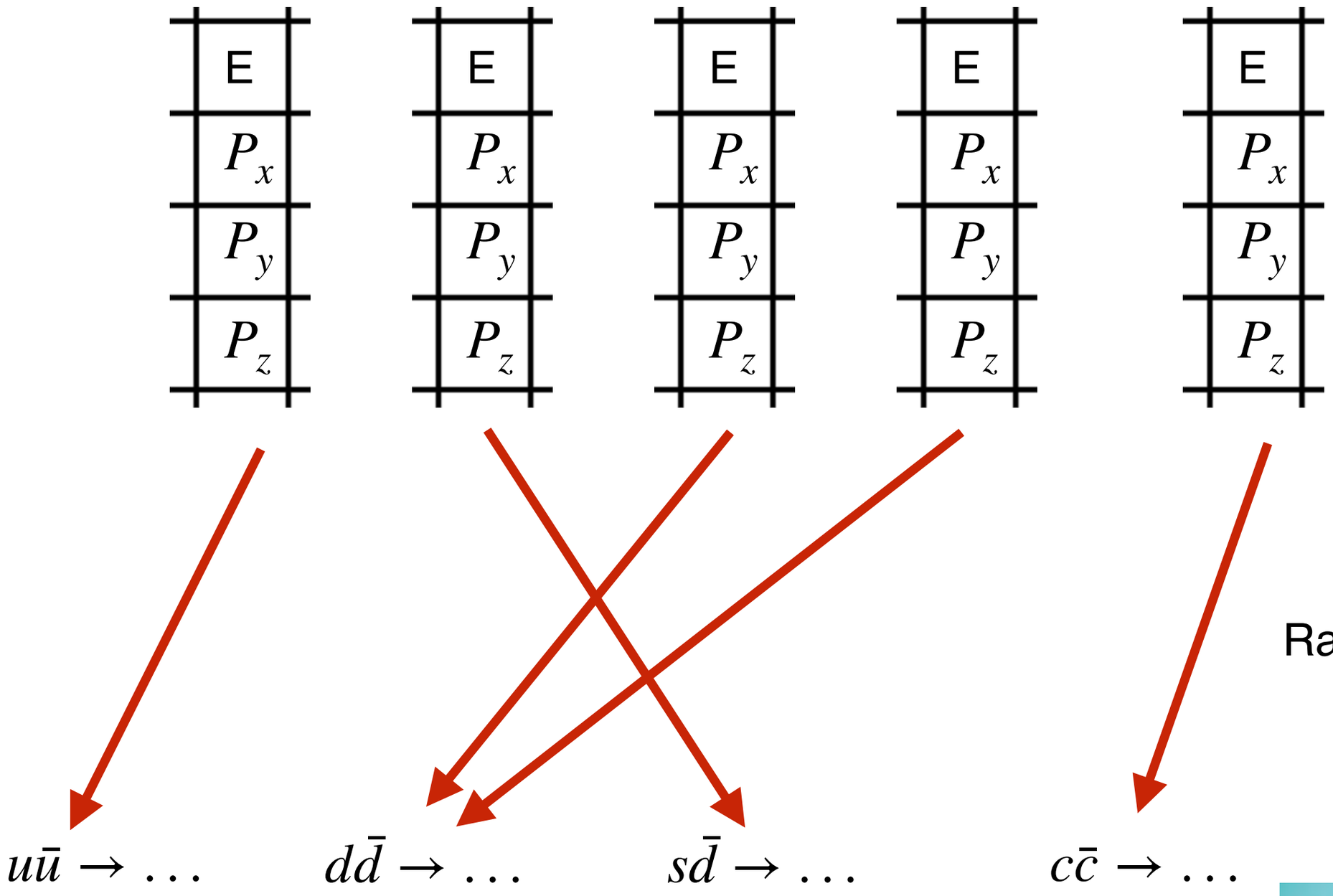
$$\sigma = \frac{1}{2s} \int |\mathcal{M}|^2 d\Phi(n)$$

Event and matrix-element



$u\bar{u} \to \ldots \qquad d\bar{d} \to \ldots \qquad s\bar{d} \to \ldots \qquad c\bar{c} \to \ldots$

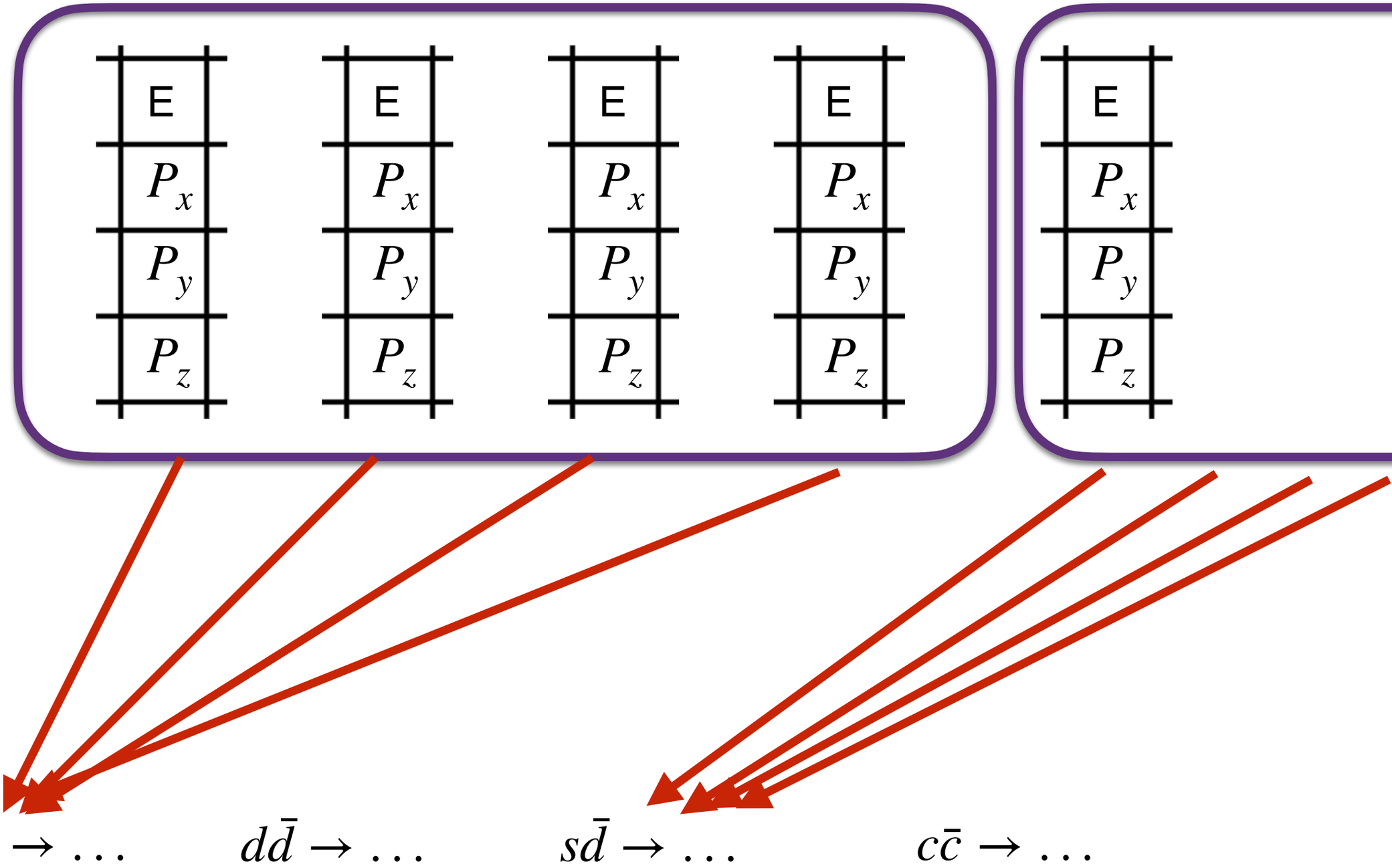
Event and matrix-element



Random assignment

Prevent SIMD/GPU !!!

Event and matrix-element

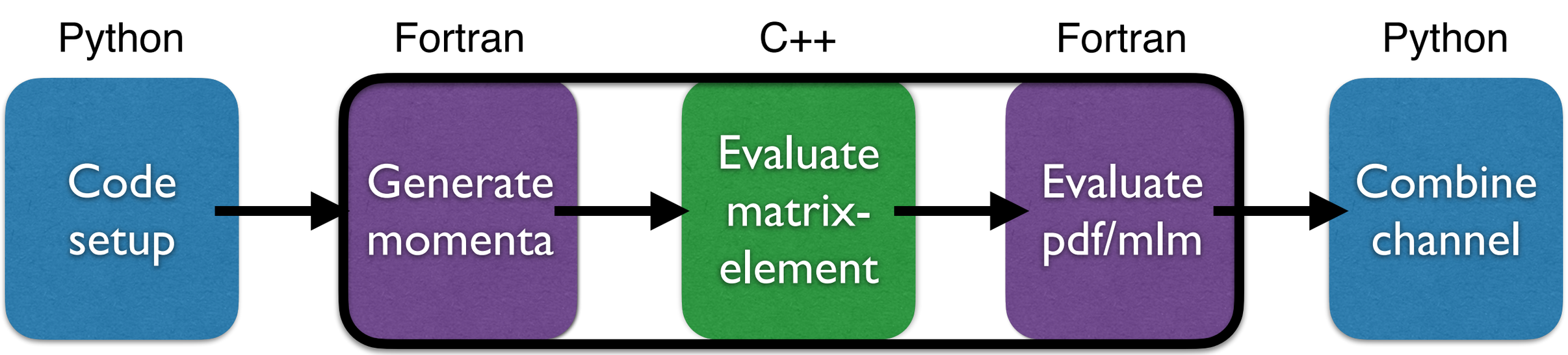


Still Random assignment but by block of N events
(For GPU we will need to split from the start go)

SIMD and integration

Technical issues

- Fortan/C++ linking
  - ➡ Python Interface is not impacted
- Need to compute multi-channel factor



Current status

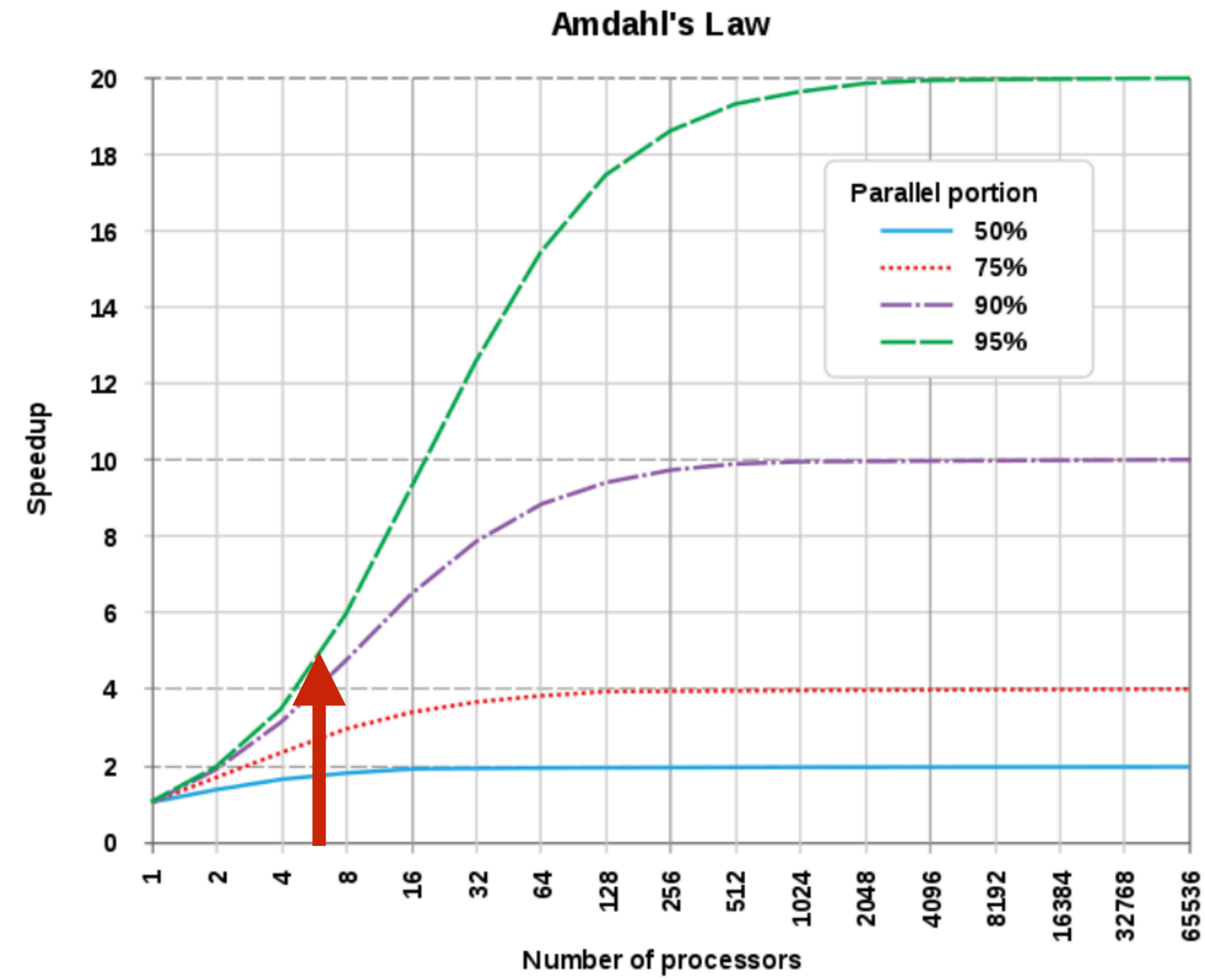
Warning still work in progress

- We can reproduce the (differential) cross-section
- We do not yet have helicity/color information (so no Parton-shower)
- latest optimisation (helicity-recycling) not yet supported

Potential gain

	$gg \to t\bar{t}$	$gg \to t\bar{t}gg$	$gg \to t\bar{t}ggg$
`madevent`	13G	470G	11T
`matrix1`	3.1G (23%)	450G (96%)	11T (>99%)

- Not full code is using SIMD
  - Gain limited by Amdahl's law
    - Around 5x



MadEvent result

Intel Gold 6148 CPU (Juwels Cluster HPC)

	mad (81952 MEs)	mad	mad	sa/brdg
ggttgg	[sec] tot = mad + MEs	[TOT/sec]	[MEs/sec]	[MEs/sec]
FORTRAN	41.82 = 3.23 + 38.60	1.96e+03 (= 1.0)	2.12e+03 (= 1.0)	---
CPP/none	47.78 = 3.56 + 44.22	1.72e+03 (x 0.9)	1.85e+03 (x 0.9)	1.90e+03
CPP/sse4	23.04 = 2.97 + 20.07	3.56e+03 (x 1.8)	4.08e+03 (x 1.9)	4.05e+03
CPP/avx2	12.19 = 2.88 + 9.32	6.72e+03 (x 3.4)	8.80e+03 (x 4.2)	9.24e+03
CPP/512y	11.57 = 2.86 + 8.71	7.08e+03 (x 3.6)	9.41e+03 (x 4.4)	1.01e+04
CPP/512z	8.26 = 2.88 + 5.38	9.92e+03 (x 5.1)	1.52e+04 (x 7.2)	1.60e+04

TIME Total = MadEvent (scalar) + MEs (parallel)

TIME MadEvent (scalar)

TIME MEs (parallel)

THROUGHPUT MadEvent + MEs (within madevent)

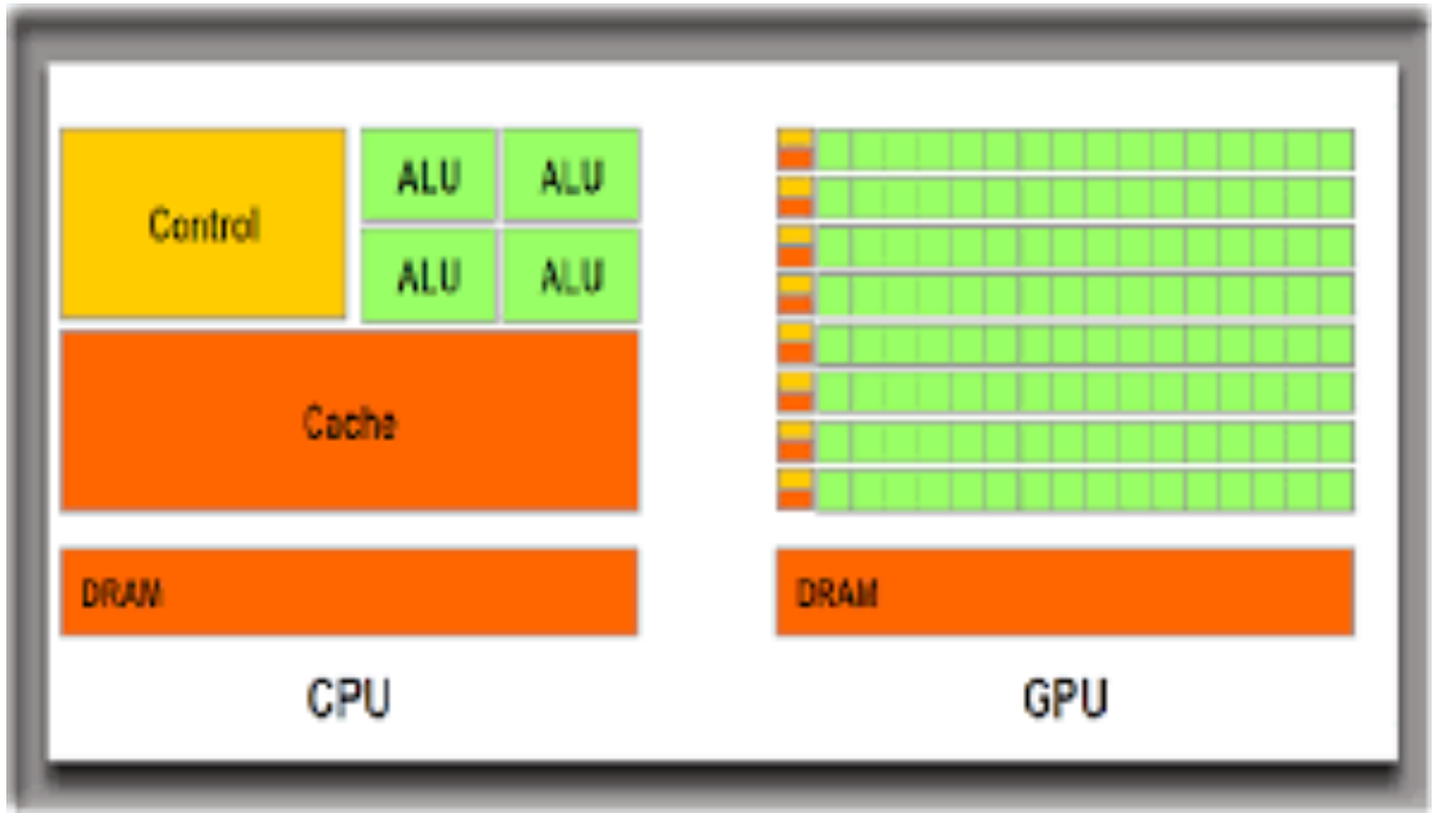
THROUGHPUT MEs (within madevent)

THROUGHPUT MEs (within standalone test application)

- No additional surprise here.
  - Have to fix the last missing feature

Data parallelism on GPU

CPU versus GPU



CPU	GPU
Central Processing Unit	Graphics Processing Unit
Several cores	Many cores
Low latency	High throughput
Good for serial processing	Good for parallel processing
Can do a handful of operations at once	Can do thousands of operations at once

Speed versus Latency

- Speed: number of operation per second
- Latency: delay in the first operation
  - ➡ $T = L + vD$
- How amazon transfer data from one cluster to another

- Speed: Large bandwidth
  - Fiber connection: Gb
- Latency: time of the travel between the two cluster.

- Latency is “reactivity”

Going Parallel (GPU)

- GPU are
  - Thread parallelism
  - Lock step operation by 32/64 thread
  - Memory management is critical

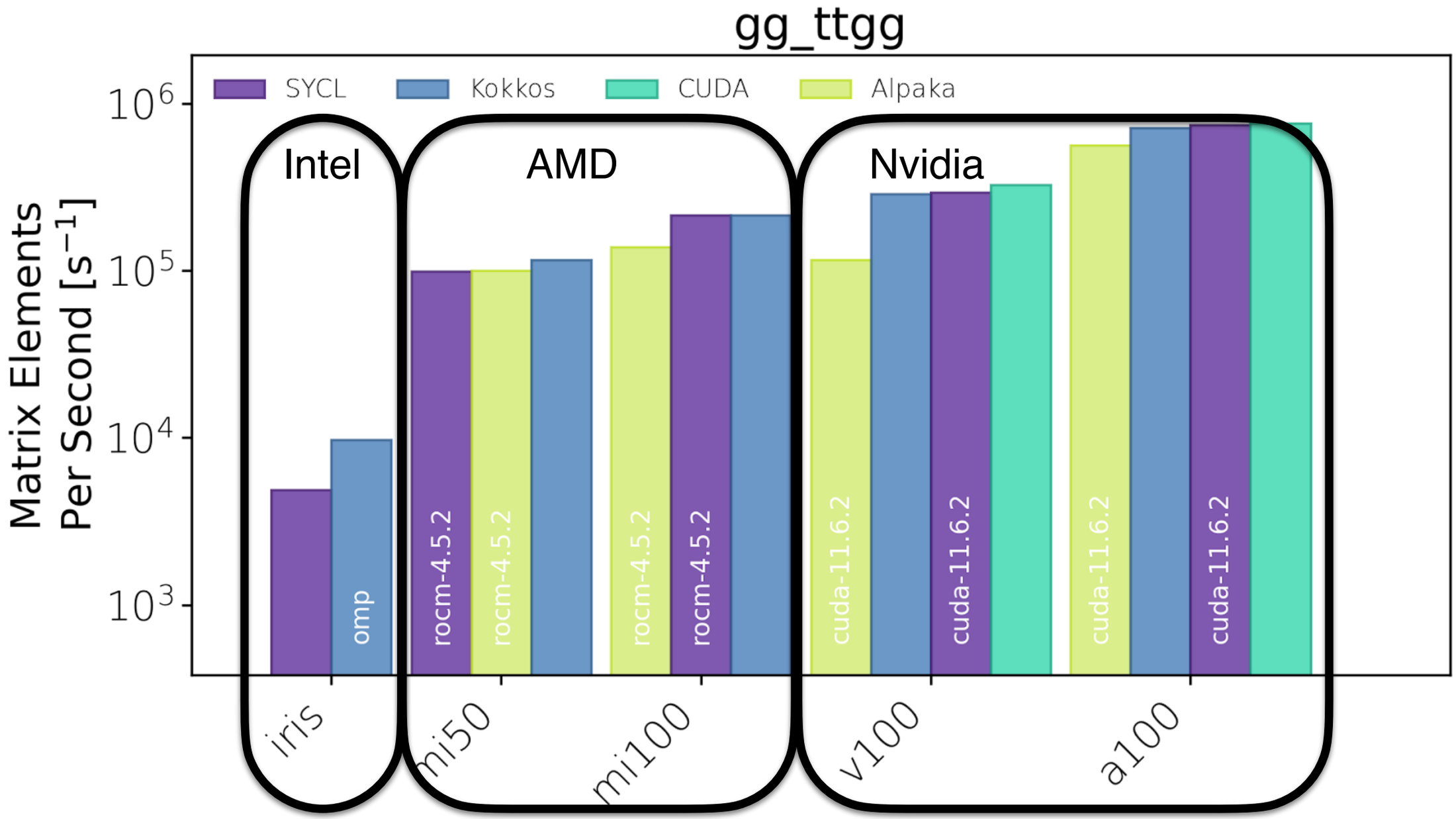
- CUDA implementation:
  - Same code as the SIMD C++
  - kernel is the FULL matrix-element

- Abstraction Layer:
  - Kokkos, sycl, alpaka
  - Allow portability

- Other work:
  - MadFlow
  - Old MadGPU

GPU result

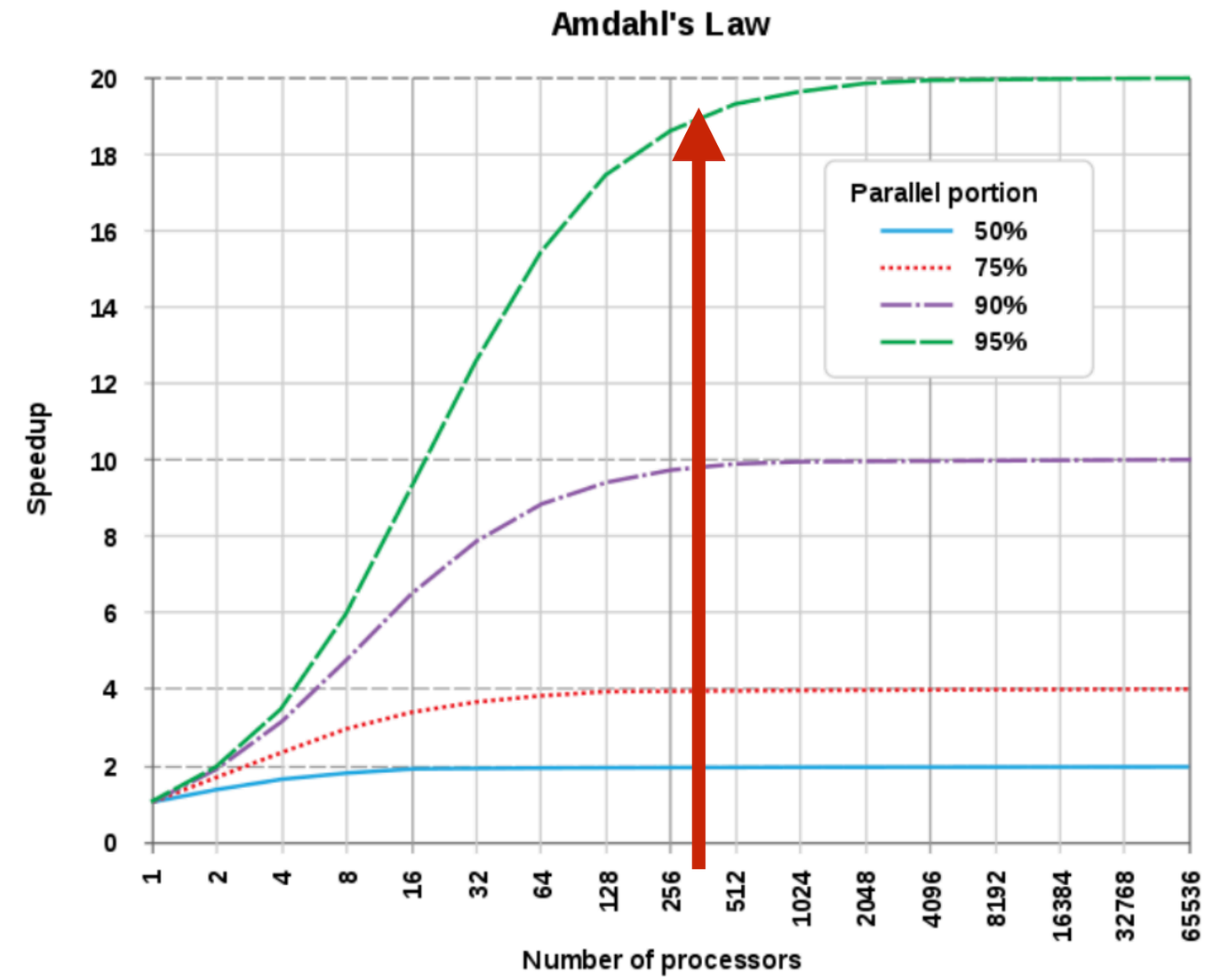
1-core Standalone C++ scalar	1.84E3 **(x1.00)**
Standalone CUDA NVidia V100S-PCIE-32GB (TFlops*: 7.1 FP64, 14.1 FP32)	4.89E5 **(x270)**



Potential gain

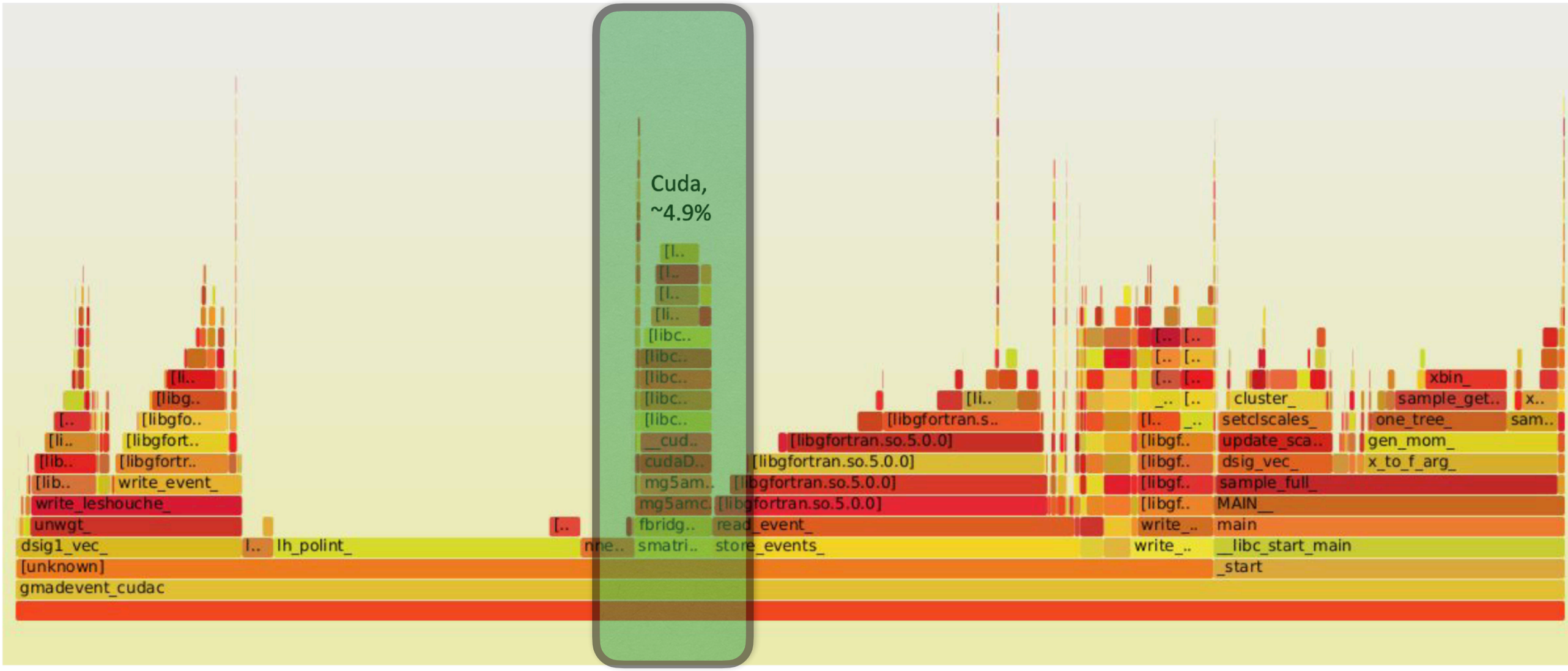
	$gg \to t\bar{t}$	$gg \to t\bar{t}gg$	$gg \to t\bar{t}ggg$
`madevent`	13G	470G	11T
`matrix1`	3.1G (23%)	450G (96%)	11T (>99%)

- Not full code is using GPU
  - Gain limited by Amdahl's law
    - Around 20x



Phase-Space Integration

- GPU is only used for 5% of the total time



- Waste of the GPU
  - Solution under-investigation (lhapdf, multi-process, un-weighting)

Conclusion

- Speed up can be achieved in multiple way
  - ➡ Better software
  - ➡ Better use of hardware
- Faster matrix-element
  - ➡ Using SIMD
  - ➡ Using GPU
    - ◆ Abstraction layer works
      - ◆ Sycl in particular
- First prototype of event generation
  - ➡ Still missing some pieces

Portability to CPU