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

MG5aMC: offloading to GPU

Olivier Mattelaer

Plan of the talk/co-author

Data Parallelism

Taylor Childers,
Nathan Nichols,
Walter Hopkins

Argonne NATIONAL LABORATORY

Stefan Roiser,
Andrea Valassi,
Stephan Hageboek,
Zenny Wetterson

CERN

Machine Learning

MADNIS

Theo Heimel,
Tilman Plehn

Heidelberg UNIVERSITY

Ramon Winterhalder,
Luca Beccatini

Why do we need speed?

chatjesset.com

Question: *Why do you need MadGraph to run faster?*

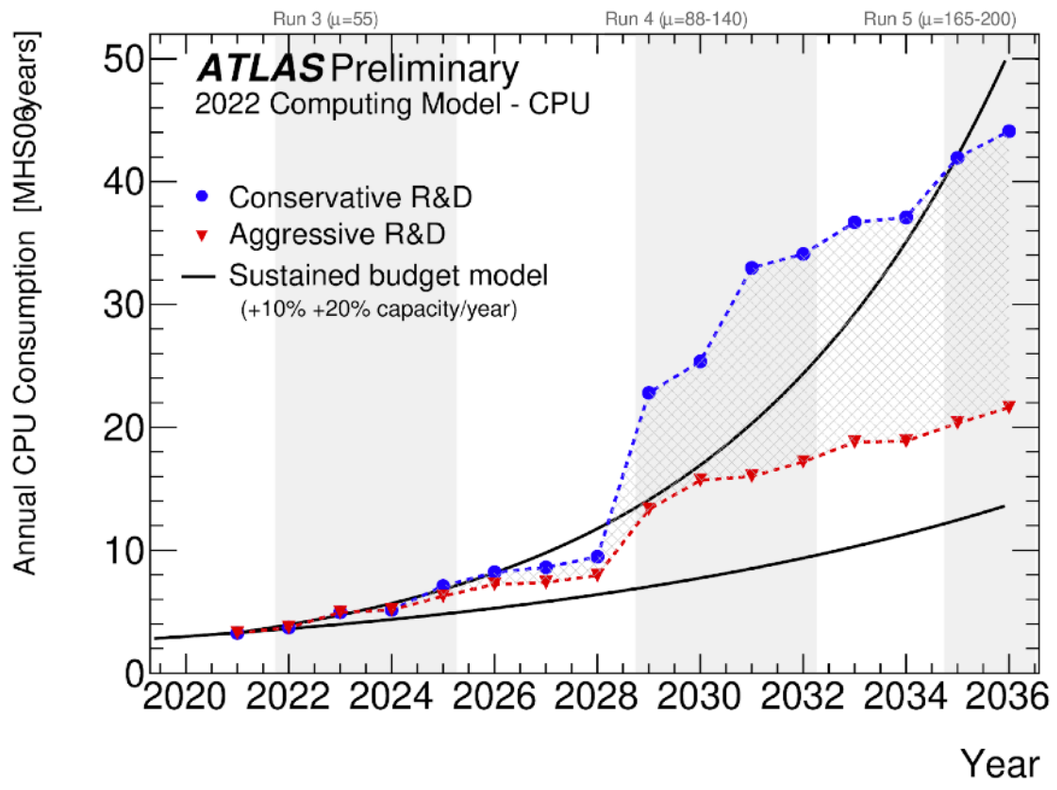
As a theoretical physicist, I can tell you that MadGraph is a powerful tool for simulating hard-scattering event generation in particle collisions. However, as the rate of data streaming in high-energy physics experiments increases, the need for faster computing becomes crucial. That's why researchers are constantly working on improving the speed and efficiency of MadGraph, so that it can keep up with the demands of modern particle physics. And as the Director of the NSF AI Institute for Artificial Intelligence and Fundamental Interactions (IAIFI), I can say that machine learning-supported smart physics trigger systems, such as those integrating FPGA and ASIC advancements, will be critical for the future of HEP experiments, including DUNE and HL-LHC.

Why do we need speed?

chatjesset.com

Question: *Why do you need MadGraph to run faster?*

As a theoretical physicist, I can tell you that MadGraph is a powerful tool for simulating hard-scattering event generation in particle collisions. However, as the rate of data streaming in high-energy physics experiments increases, the need for faster computing becomes crucial. That's why researchers are constantly working on improving the speed and efficiency of MadGraph, so that it can keep up with the demands of modern particle physics. And as the Director of the NSF AI Institute for Artificial Intelligence and Fundamental Interactions (IAIFI), I can say that machine learning-supported smart physics trigger systems, such as those integrating FPGA and ASIC advancements, will be critical for the future of HEP experiments, including DUNE and HL-LHC.

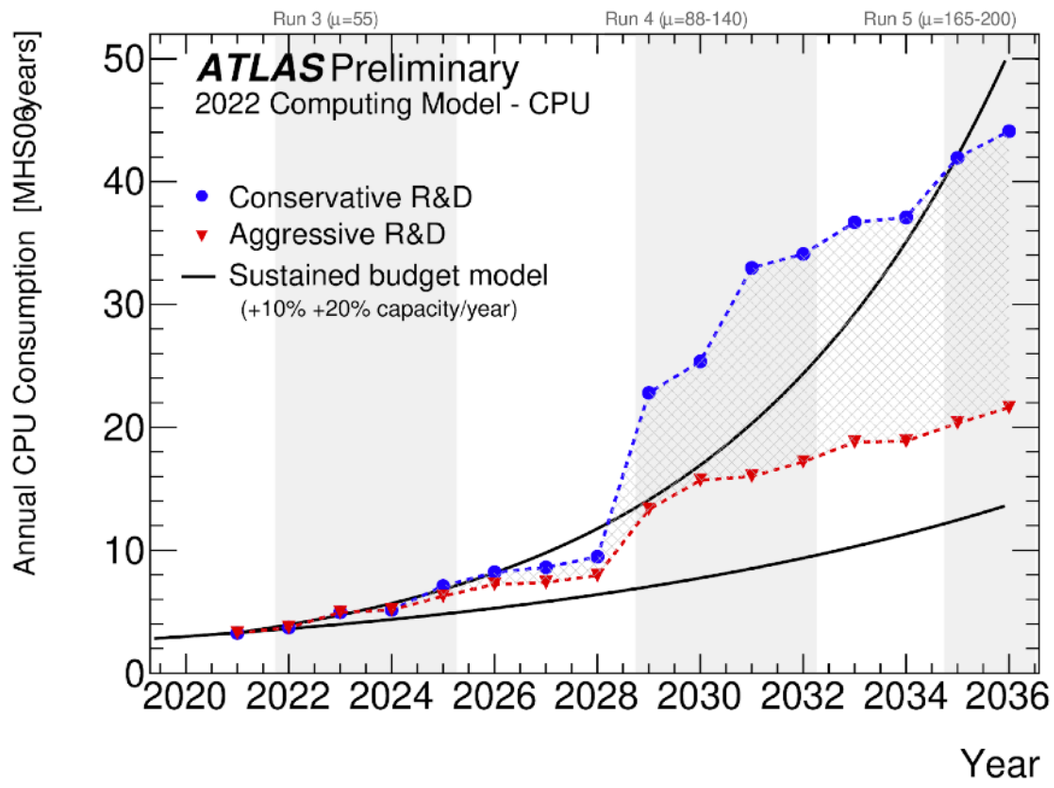


Why do we need speed?

chatjesset.com

Question: *Why do you need MadGraph to run faster?*

As a theoretical physicist, I can tell you that MadGraph is a powerful tool for simulating hard-scattering event generation in particle collisions. However, as the rate of data streaming in high-energy physics experiments increases, the need for faster computing becomes crucial. That's why researchers are constantly working on improving the speed and efficiency of MadGraph, so that it can keep up with the demands of modern particle physics. And as the Director of the NSF AI Institute for Artificial Intelligence and Fundamental Interactions (IAIFI), I can say that machine learning-supported smart physics trigger systems, such as those integrating FPGA and ASIC advancements, will be critical for the future of HEP experiments, including DUNE and HL-LHC.



Truth is …

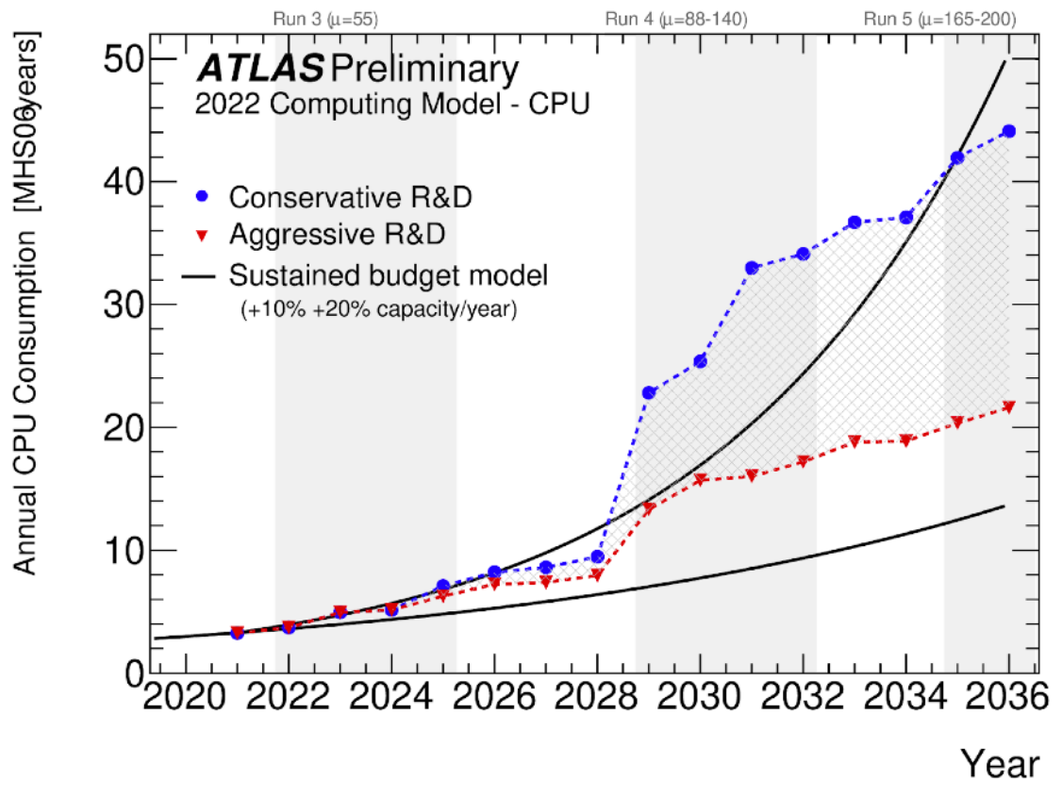
- ➡We have no choice, GPU will be the norm
- ➡HPC center are GPU dominated

Why do we need speed?

chatjesset.com

Question: *Why do you need MadGraph to run faster?*

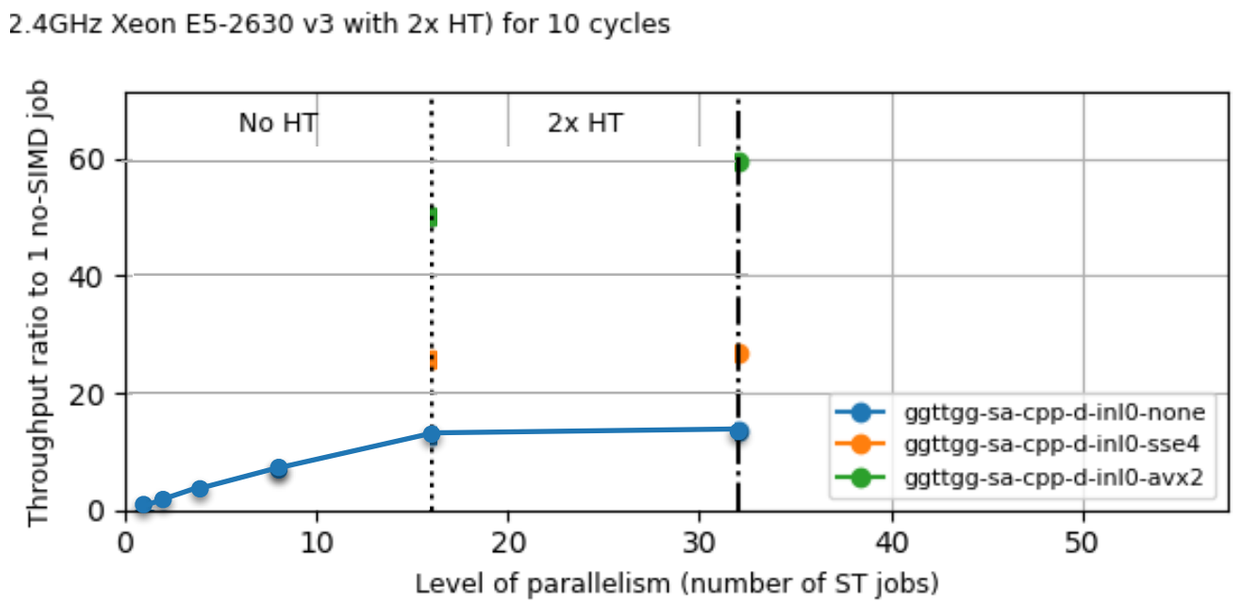
As a theoretical physicist, I can tell you that MadGraph is a powerful tool for simulating hard-scattering event generation in particle collisions. However, as the rate of data streaming in high-energy physics experiments increases, the need for faster computing becomes crucial. That's why researchers are constantly working on improving the speed and efficiency of MadGraph, so that it can keep up with the demands of modern particle physics. And as the Director of the NSF AI Institute for Artificial Intelligence and Fundamental Interactions (IAIFI), I can say that machine learning-supported smart physics trigger systems, such as those integrating FPGA and ASIC advancements, will be critical for the future of HEP experiments, including DUNE and HL-LHC.



Truth is …

- ➡We have no choice, GPU will be the norm
- ➡HPC center are GPU dominated
- ➡This is an IT project

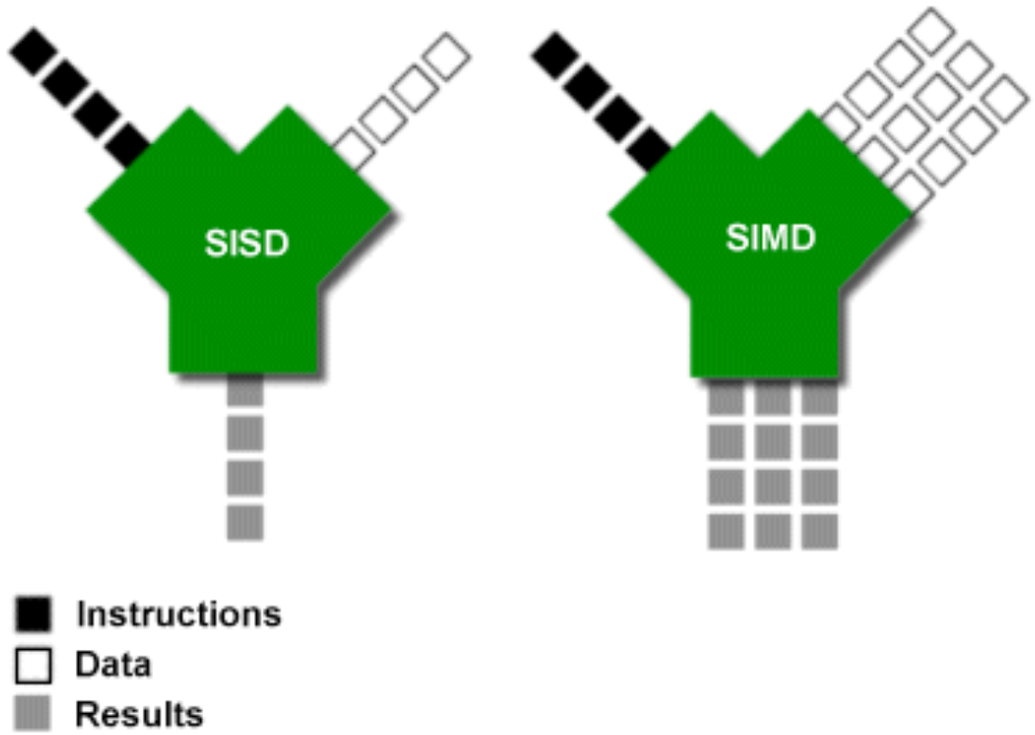
Multi-process



- Pure matrix-element evaluation (no pdf/...)
- 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
  - Small gain from hyper-threading

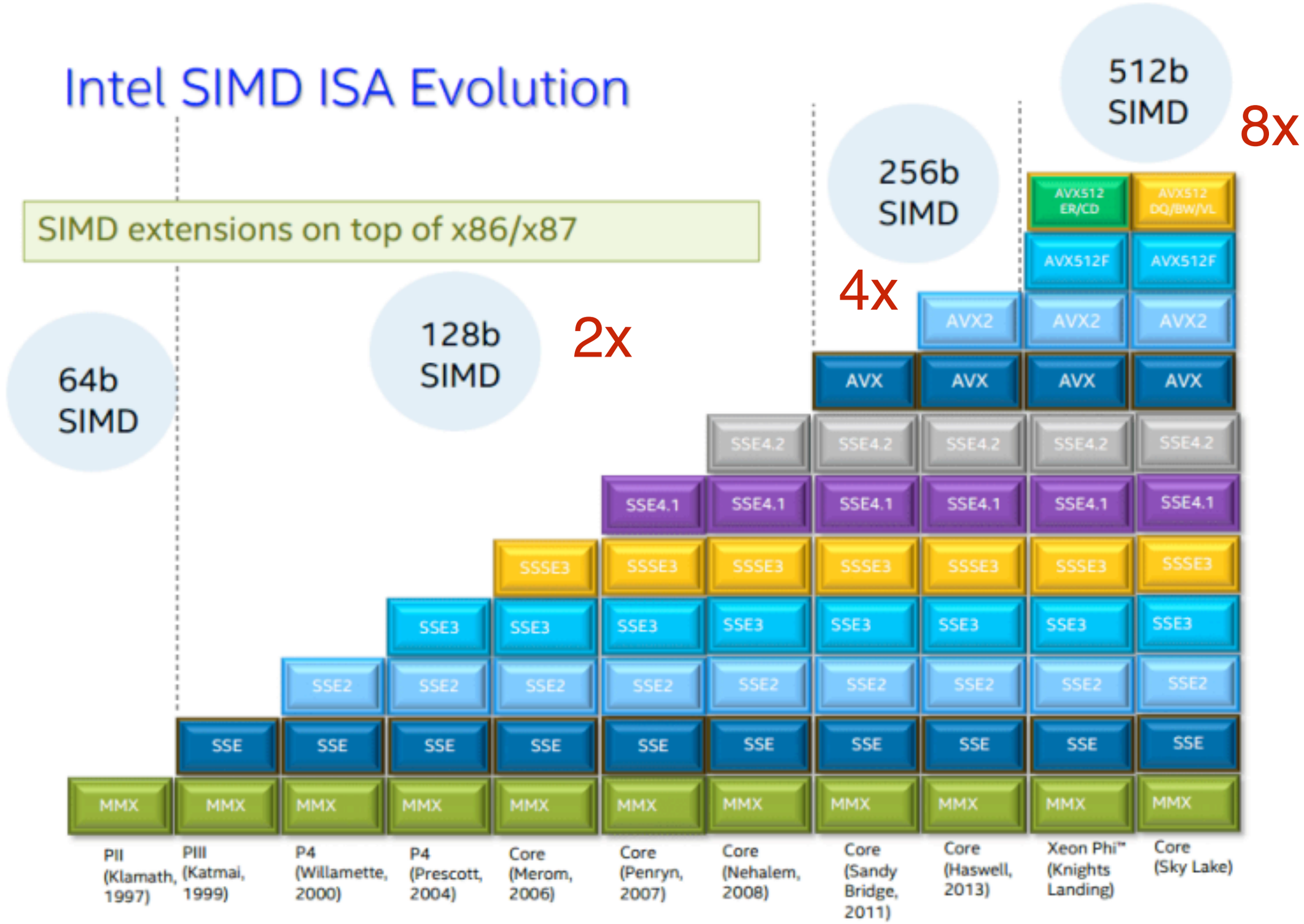
➡The more you use the less you wait

Data parallelism

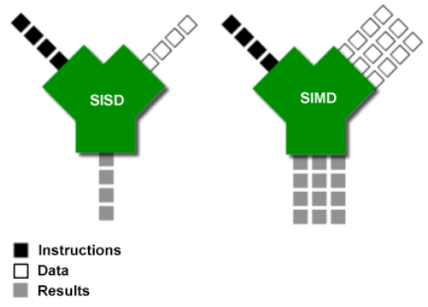


- SIMD (Single Instruction Multiple data):
  - ➡ Also named code vectorisation
  - ➡ Need dedicated memory pattern to allow it
  - ➡ Speed-up on the same hardware
    - All CPU have it

How much can you gain?



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

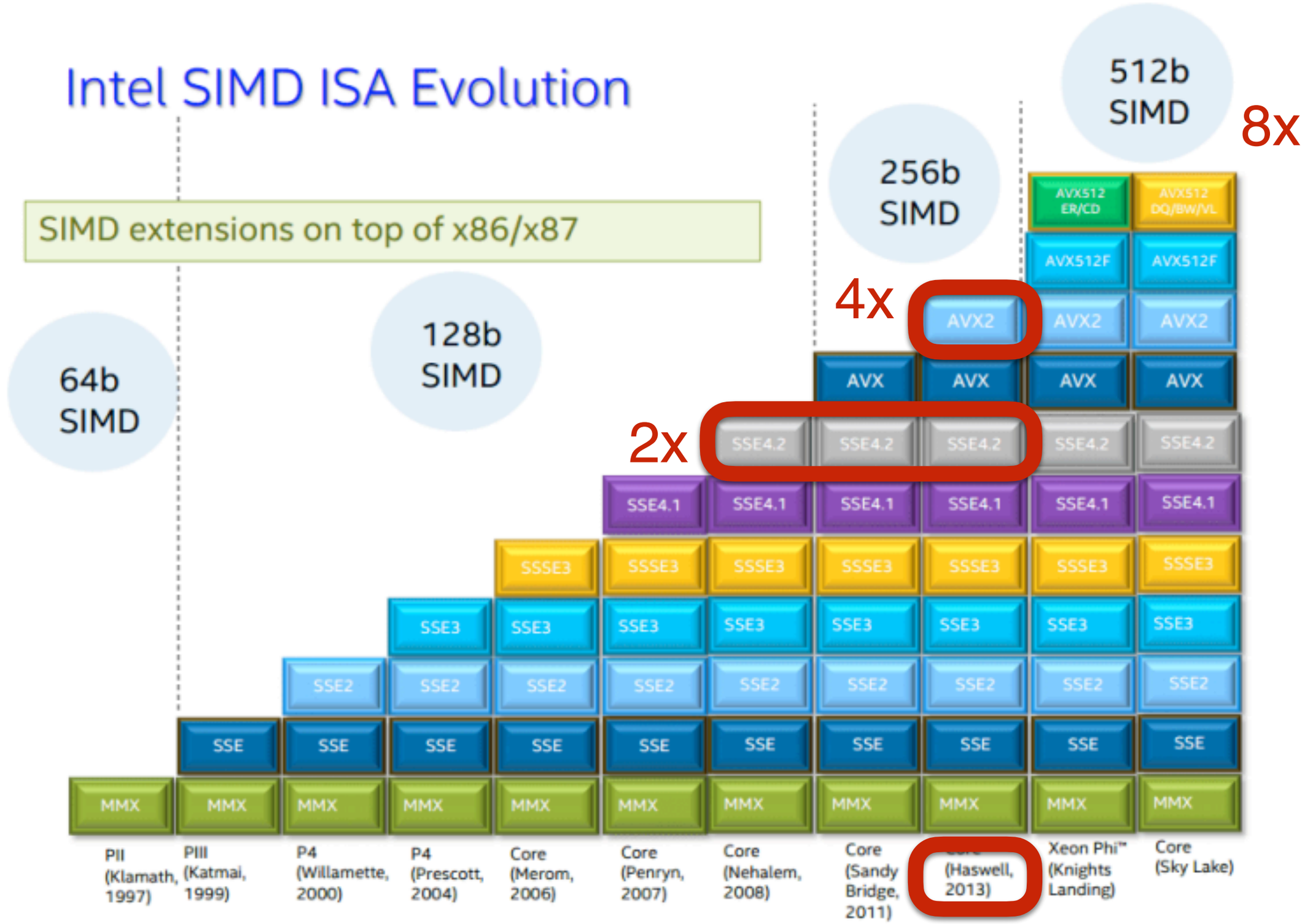
$$|E^1|p_x^1|p_y^1|P_z^1| \rightarrow |E^1|E^2|E^3|E^4|p_x^1|p_x^2|p_x^3|p_x^4|p_y^1|p_y^2|p_y^3|p_y^4|P_z^1|P_z^2|P_z^3|P_z^4|$$

  - Code in C++ with dedicated object
  - SIMD obtained by overwriting sum/multiplication operation (no code change)

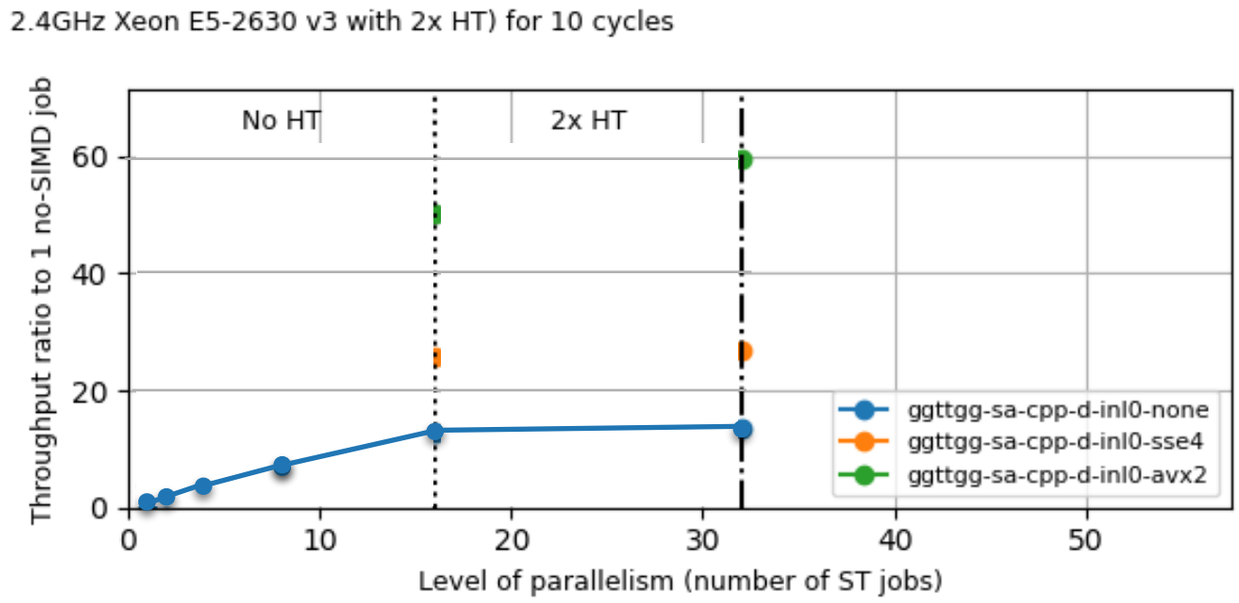
$$EE - p_x p_x - p_y p_y - p_z p_z \rightarrow EE - p_x p_x - p_y p_y - p_z p_z$$

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

Haswell computer

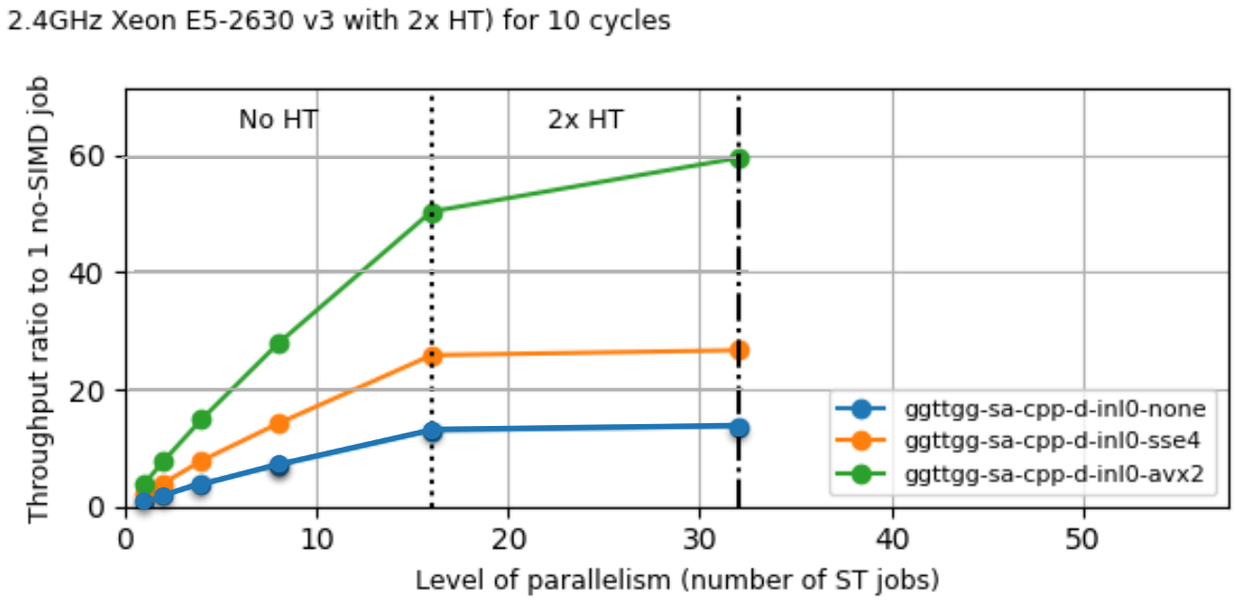


Haswell Computer



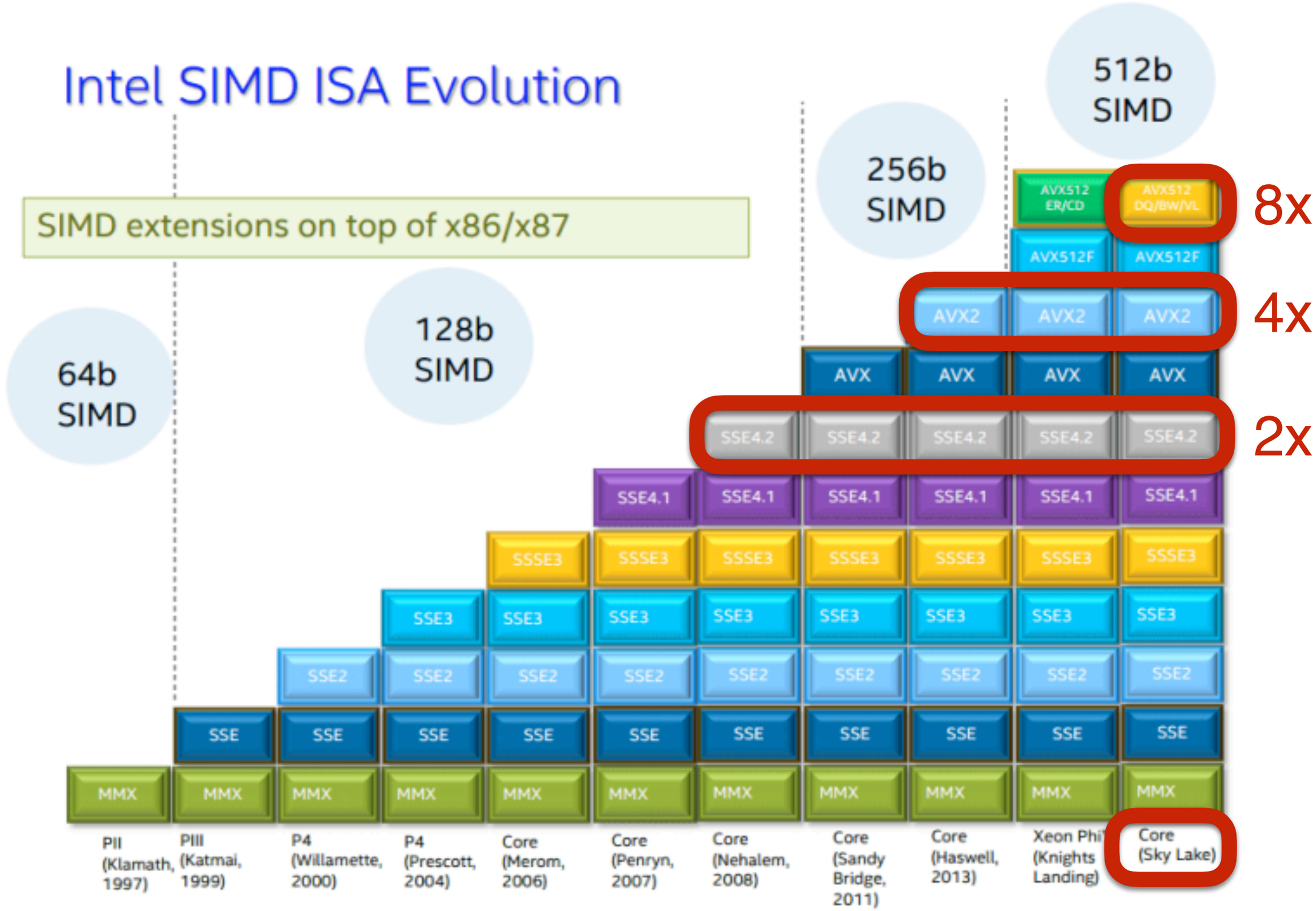
- This was the status without SIMD

Haswell Computer

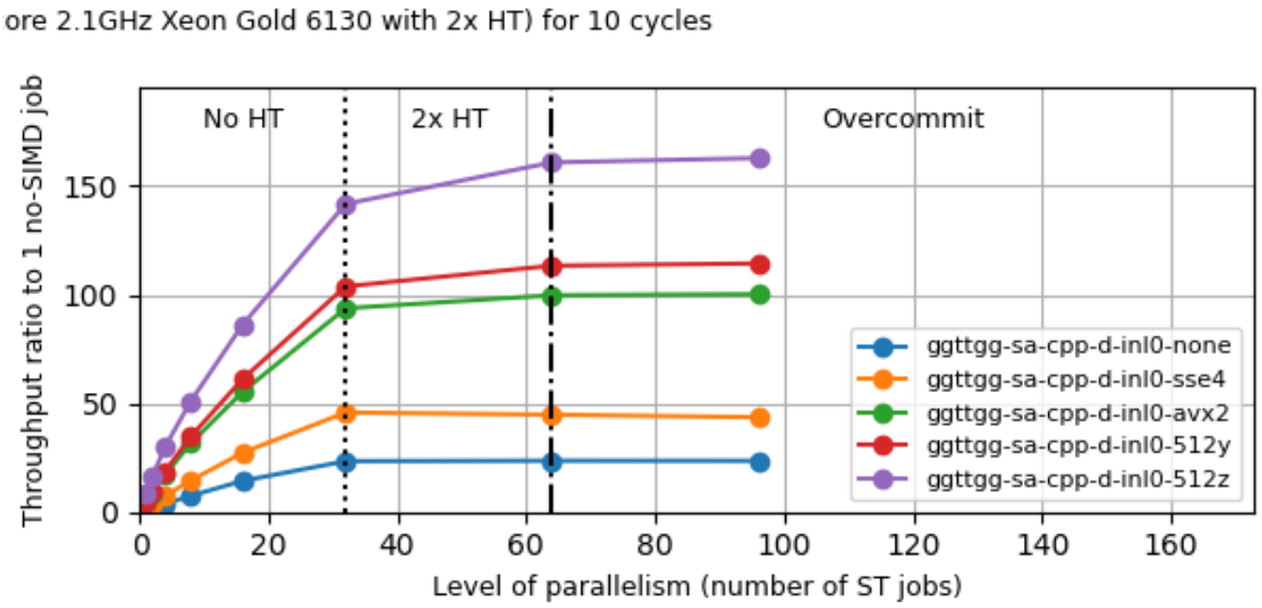


- Orange line (SSE4.2: expected speed-up 2x)
  - Expectation met
- Green line (AVX2: expected speed-up 4x)
  - Expectation met
  - HT helps significantly in this case
    - Hide memory latency (?)

CascadeLake Computer



CascadeLake Computer (32 core)



- Orange line: same with SSE4
  - Expected: 2x
  - Exception met
- Green line: same with AVX2
  - Expected 4x
  - Exception met
- Purple line: AVX512z
  - Expected 8x
  - Exception failed (6-7x)
    - down-cloacking

Computation

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

- Determine the production mechanism

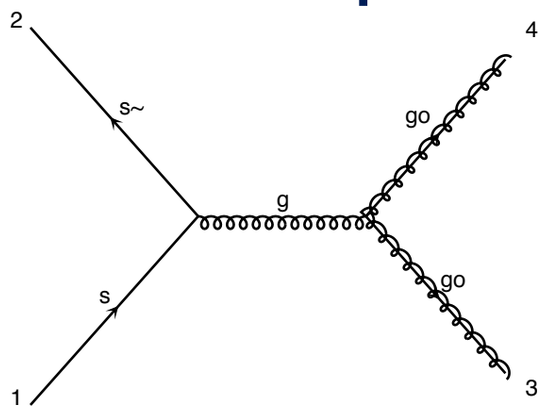


diagram 1 QCD=2, QED=0

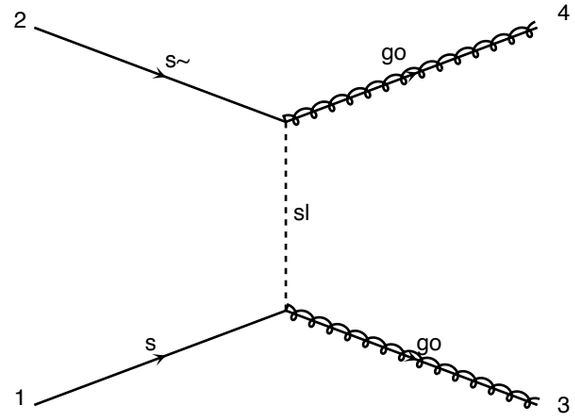


diagram 2 QCD=2, QED=0

- Evaluate the matrix-element

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

➡Need Feynman Rules!

- Phase-Space Integration

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

Computation

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

- Determine the production mechanism

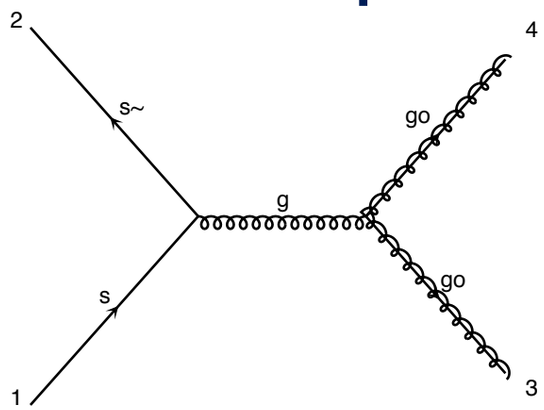


diagram 1 QCD=2, QED=0

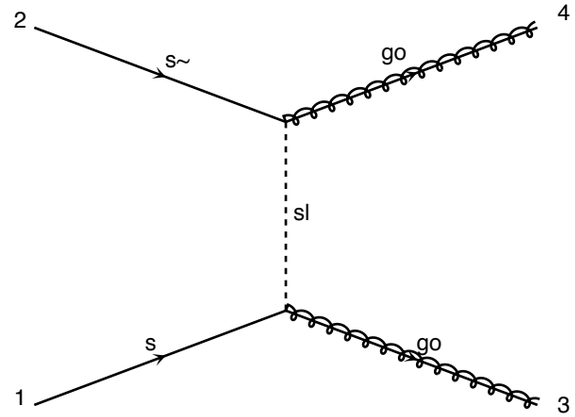


diagram 2 QCD=2, QED=0

- Evaluate the matrix-element

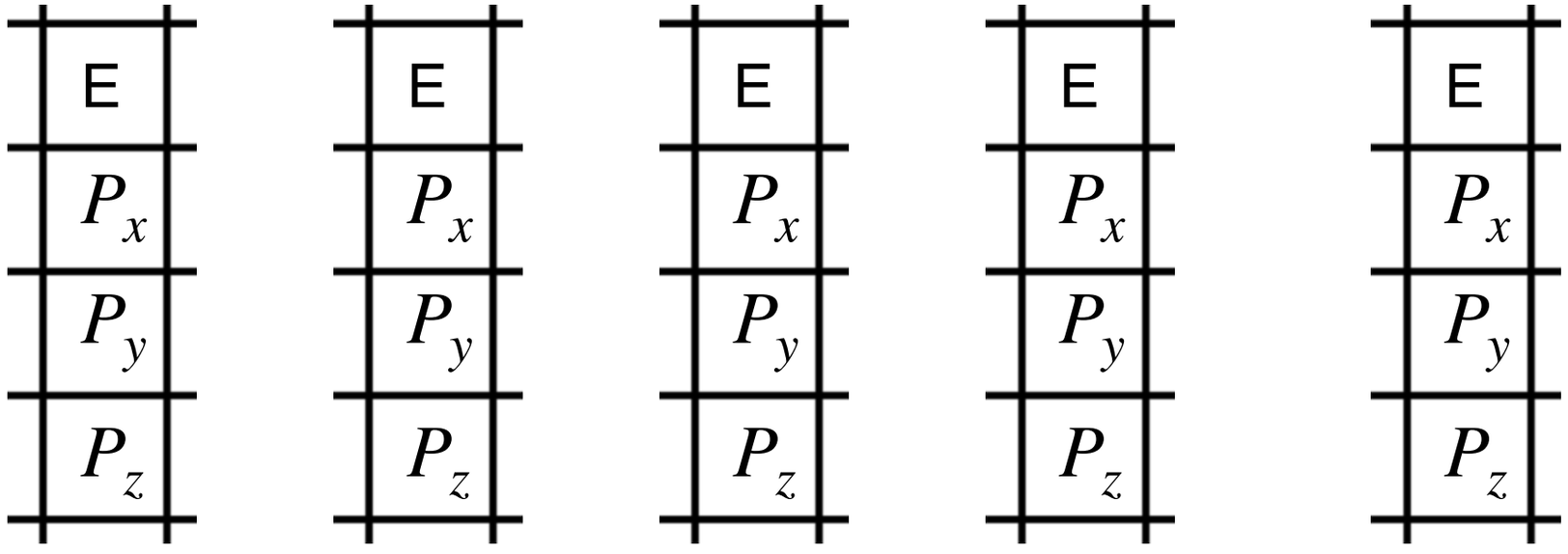
$$|\mathcal{M}|^2$$

➡Need Feynman Rules!

- Phase-Space Integration

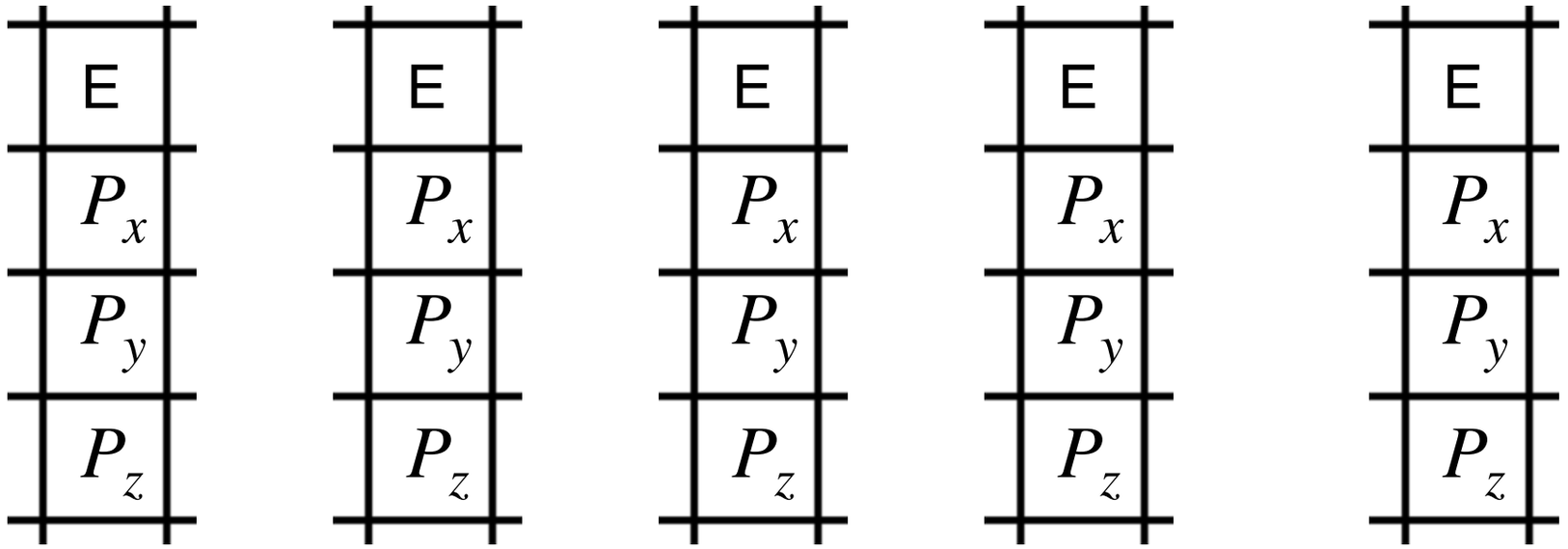
$$\sigma = \frac{1}{2s} \int |\mathcal{M}|^2 d\Phi(n)$$

Event and matrix-element



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

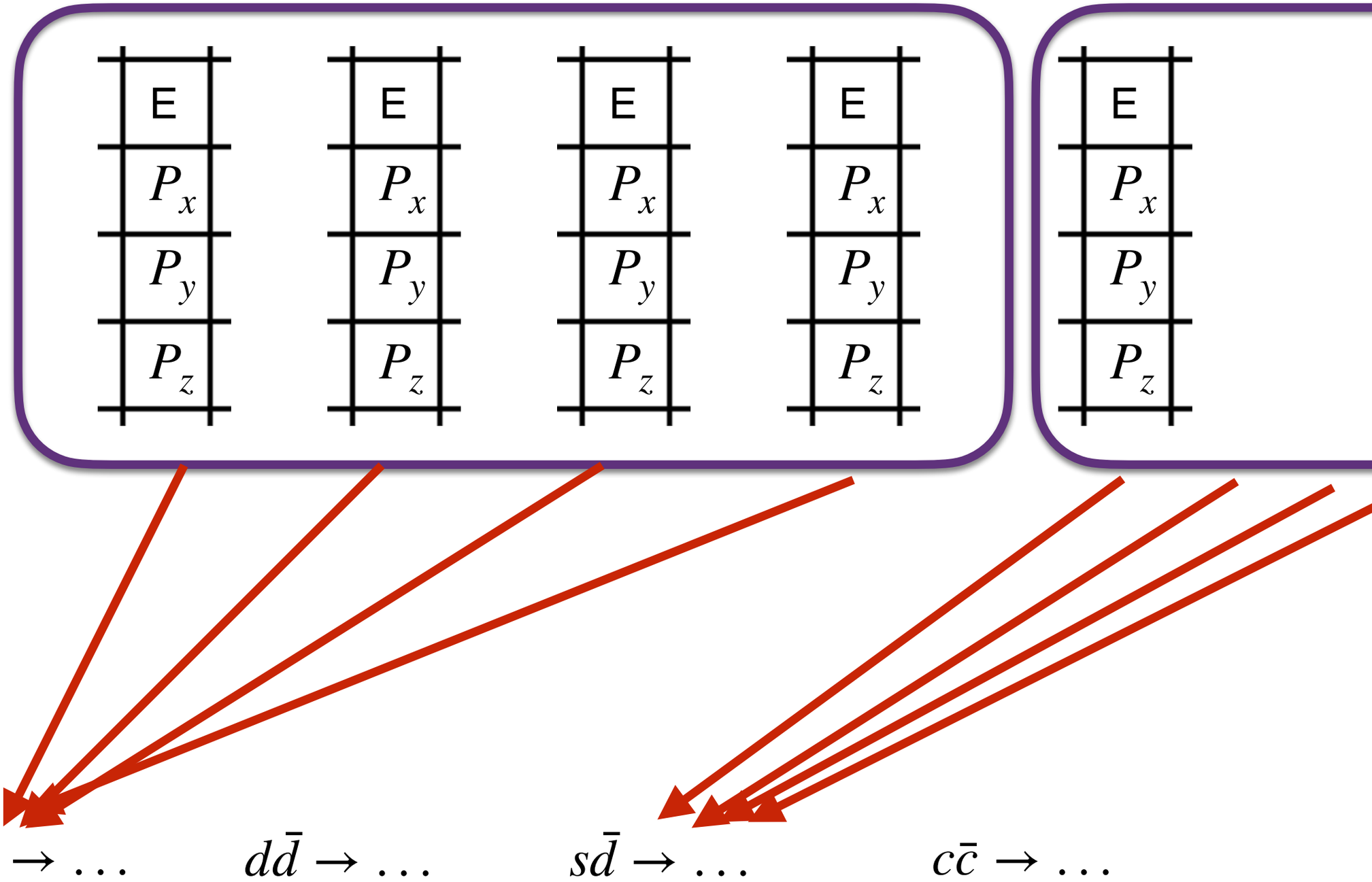
Event and matrix-element



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)

Current status

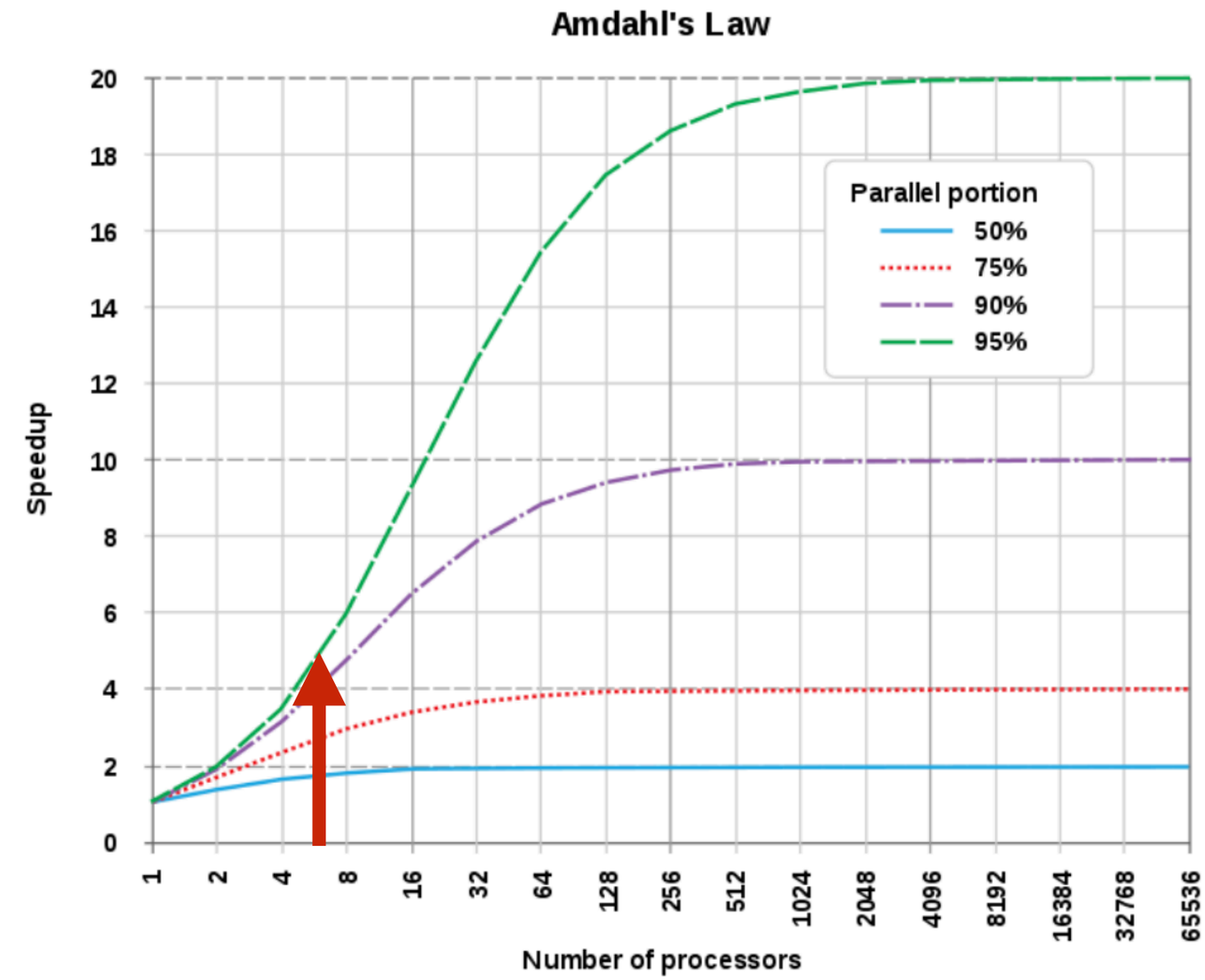
Warning still work in progress

- We can reproduce the (differential) cross-section (some issue with MLM)
- We have event generation
- 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 finish validation

Going Parallel (GPU)

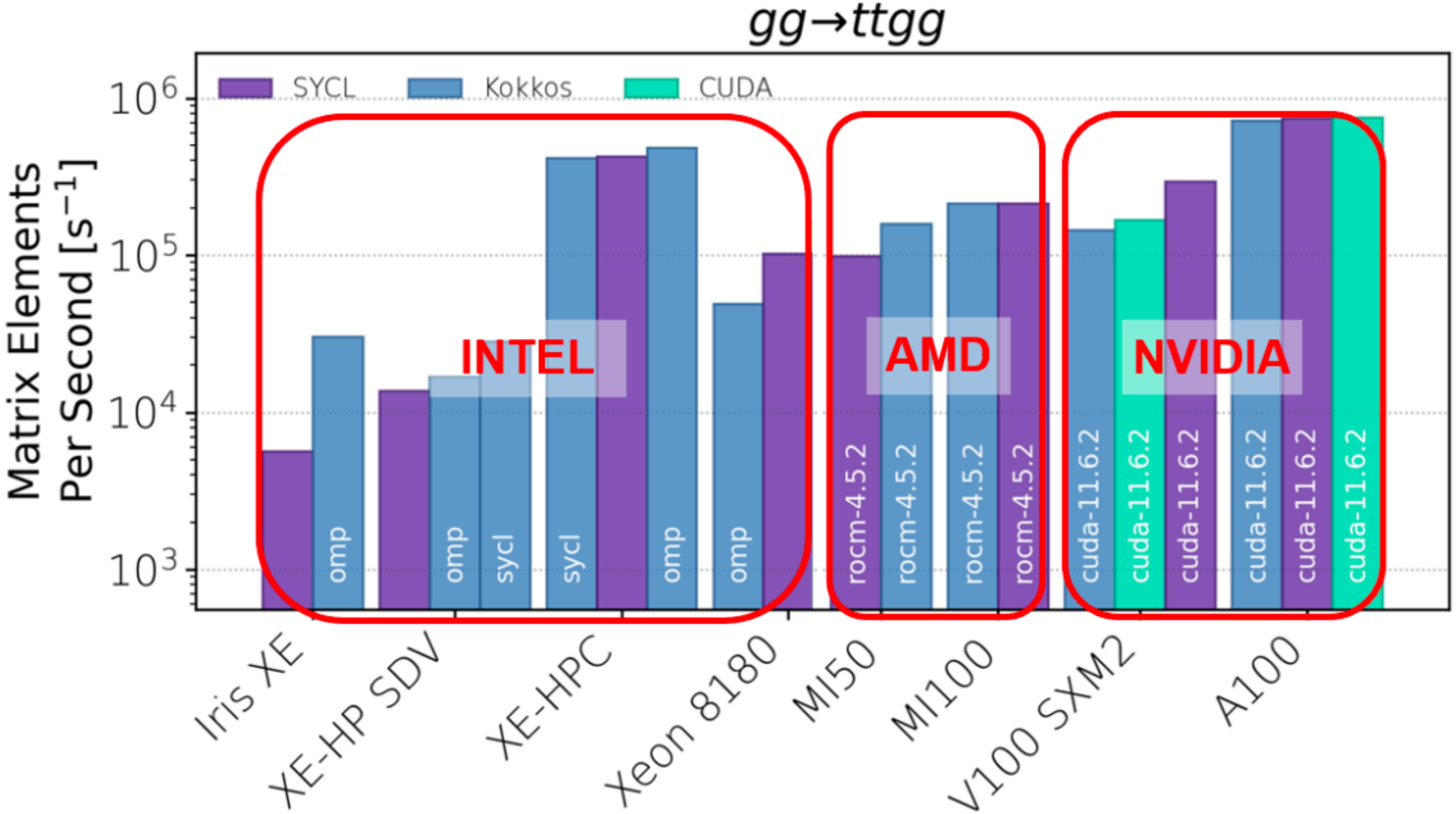
- GPU are
  - Thread parallelism
  - Lock step operation by 32/64 thread
  - Memory management is critical

- 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

Hardware portability

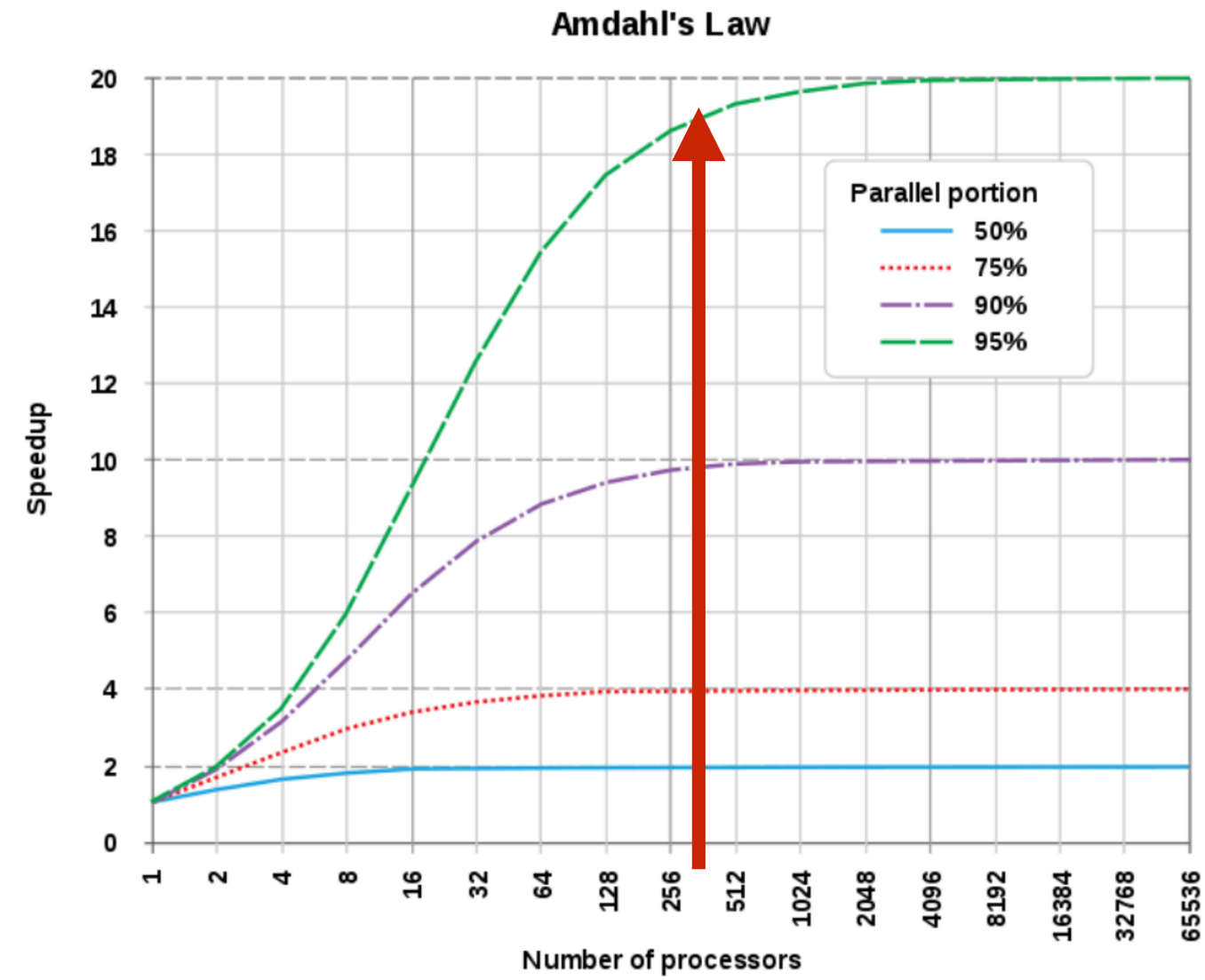


Speed up of ~300x faster than CPU

Potential gain

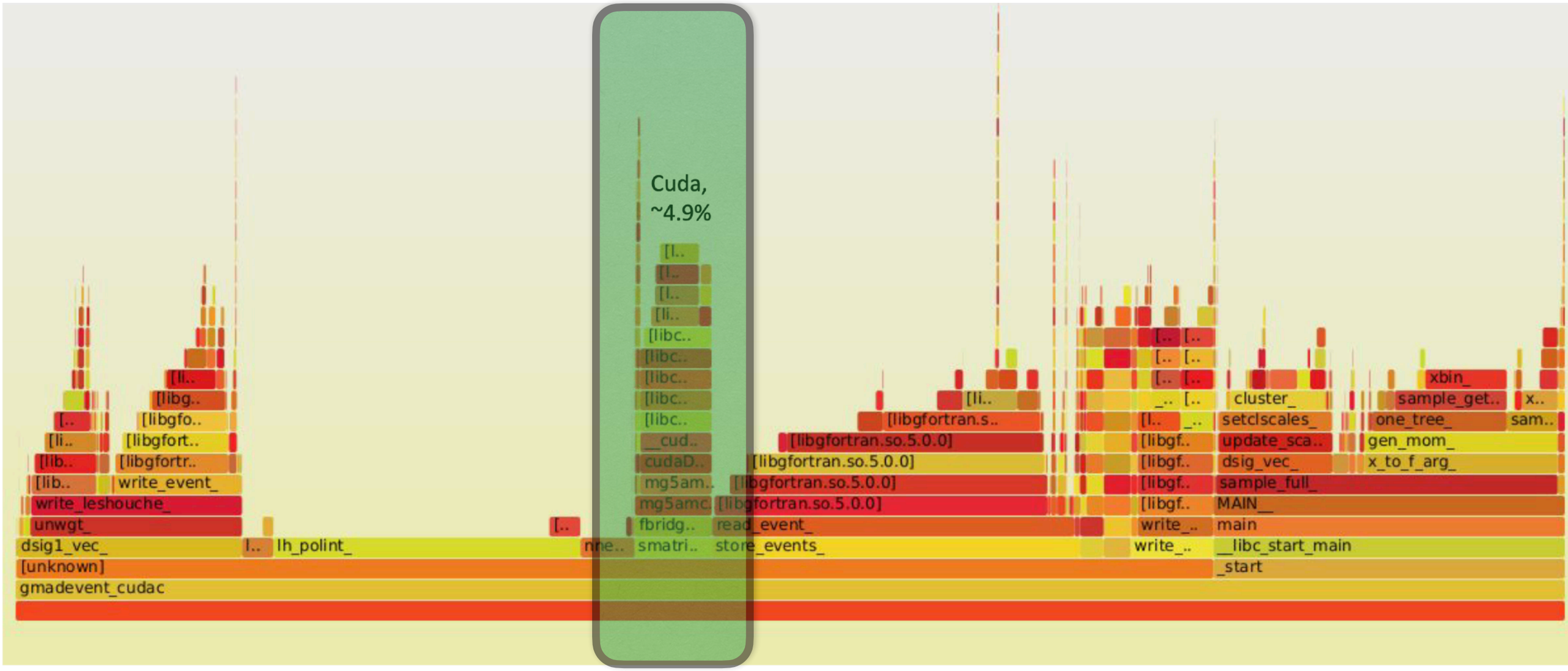
	$gg \to t\bar{t}$	$gg \to t\bar{t}gg$	$gg \to t\bar{t}ggg$
`madevent`	13G	470G	11T
`matrix1`	3.1G (23%)	450G (96%)	11T (>99%)

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



Phase-Space Integration

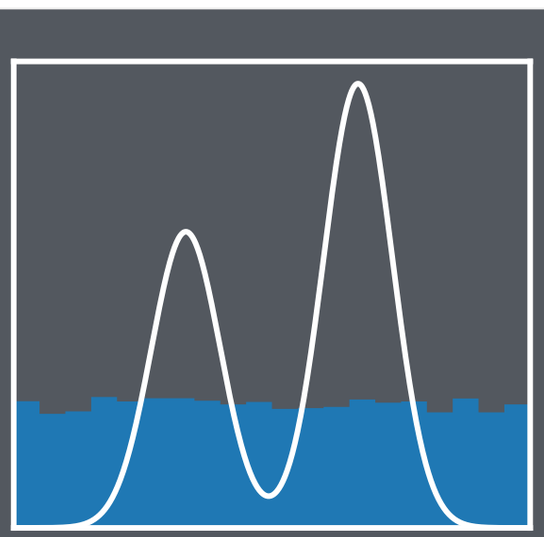
- GPU is only used for 5% of the total time



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

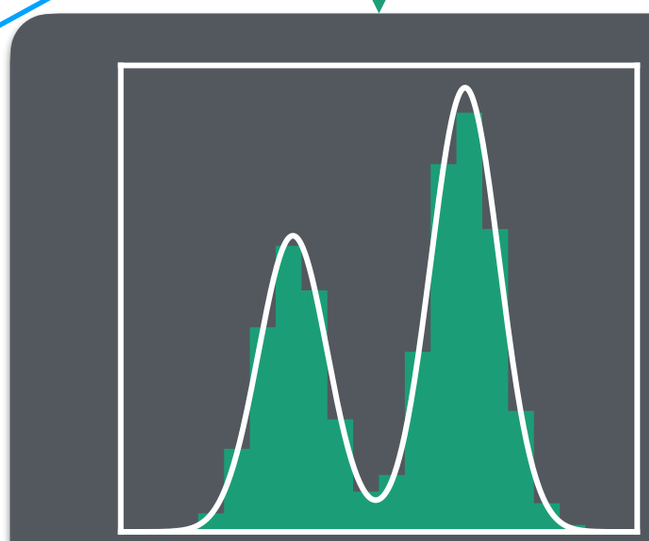
Monte Carlo integration

$$\int f(x)dx$$



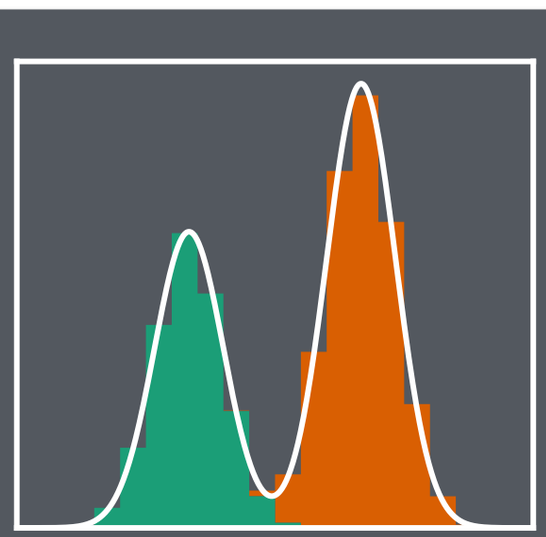
Flat sampling:

$$I = \langle f(x) \rangle_{x \sim \text{unif}}$$



Importance

$$I = \left\langle \frac{f(x)}{g(x)} \right\rangle_{x \sim g(x)}$$



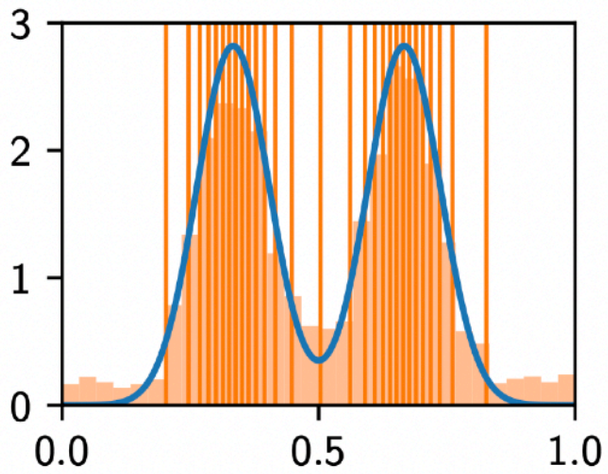
Multi-channel:

$$I = \sum_i \left\langle \alpha_i(x) \frac{f(x)}{g_i(x)} \right\rangle_{x \sim g_i(x)}$$

Importance sampling — Vegas

Factorize probability

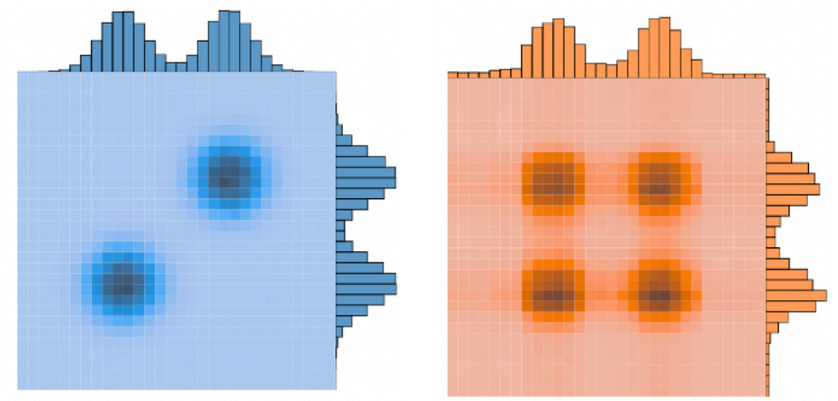
Fit bins with equal probability and varying width



Computationally cheap

High-dim and rich peaking functions
→ slow convergence

Peaks not aligned with grid axes
→ phantom peaks



Importance sampling — Flow

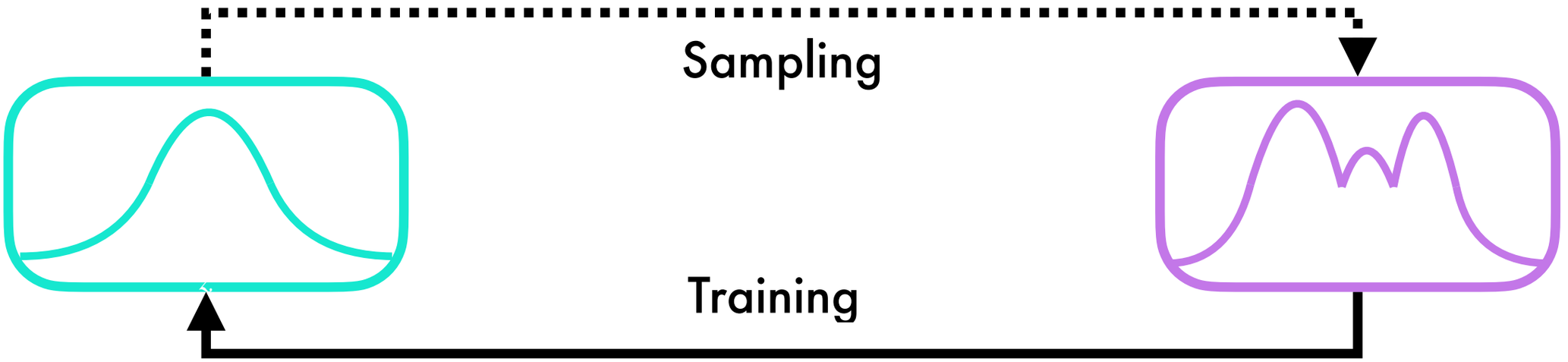
Using a Normalizing Flow

⊕ Invertibility

→ **bijective mapping**

⊕ tractable Jacobians

→ **fast training and evaluation**



[2001.05478, 2001.05486, 2001.10028,2005.12719, 2112.09145]

MadNIS

Basic Functionality

- Neural Channel Weights
- Normalizing Flow
- MadGraph matrix elements
- MadEvent channel mappings

Improved multi-channeling

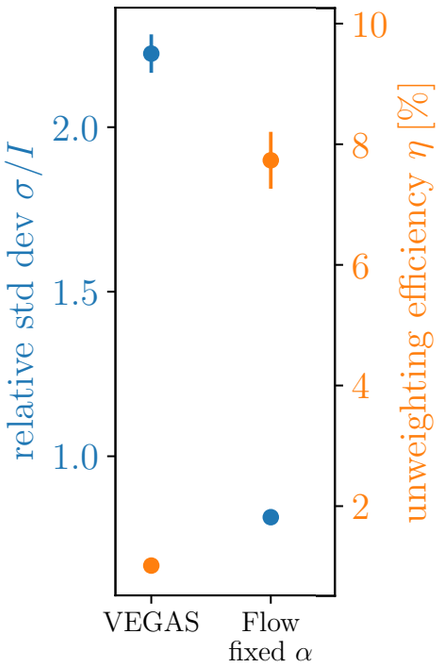
- Conditional flows
- Overflow Channels
- Symmetries between channels
- Stratified Sampling/ Training

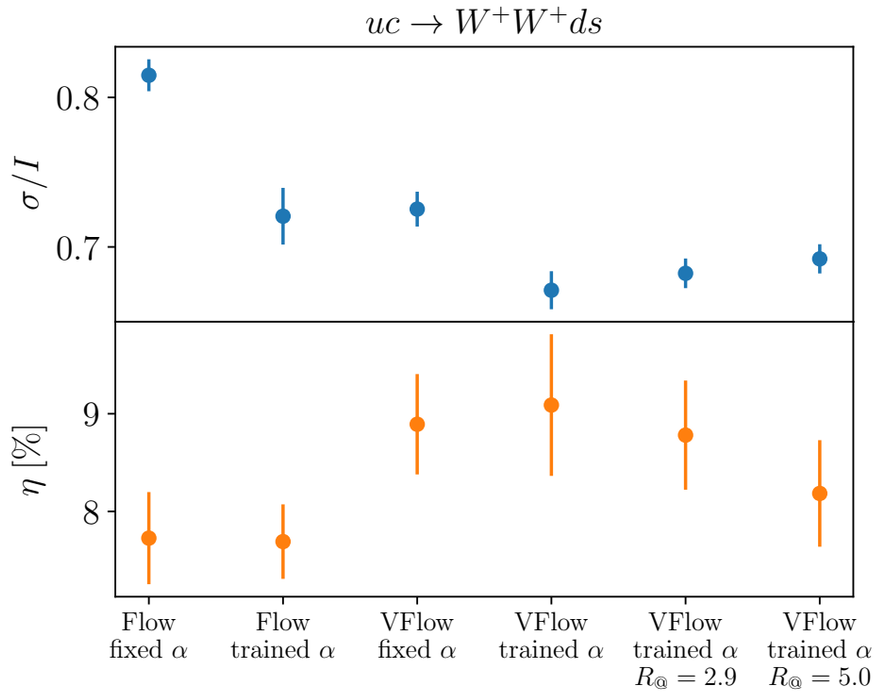
Improved training

- Vegas Initialization
- Buffered Training
- Trainable Rotations

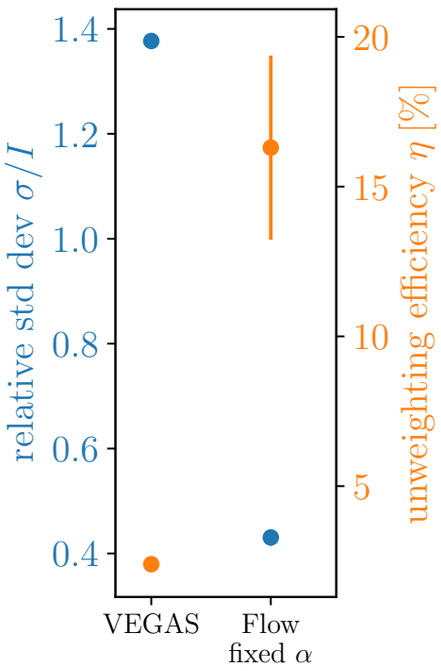
LHC Example (preliminary)

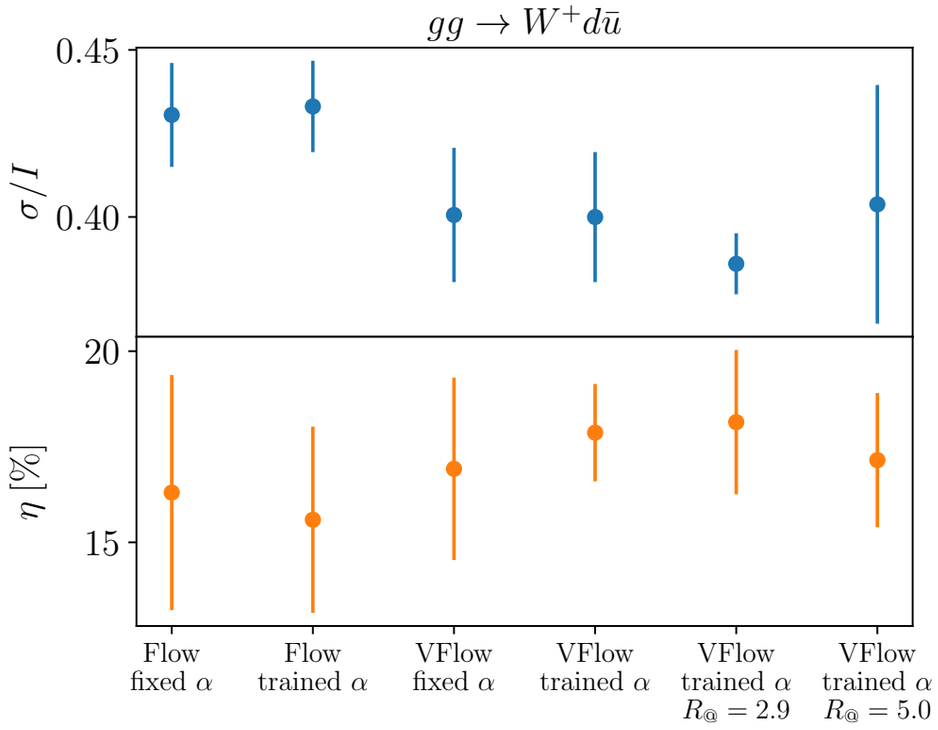
VBS





Wjets

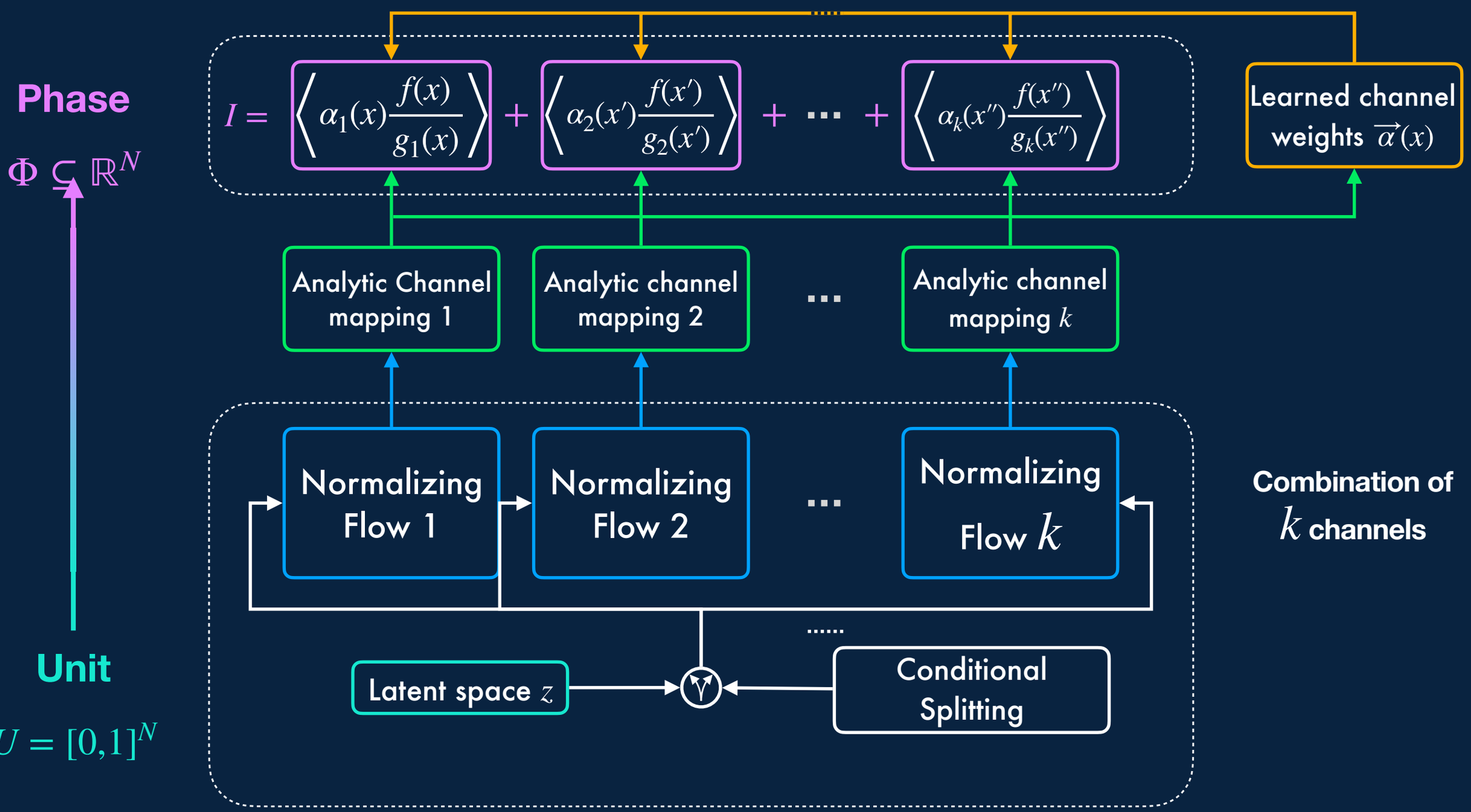




Conclusion

- Speed up can be achieved in multiple way
  - ➡ Better software (madnis)
  - ➡ Better use of hardware
- Matrix-Element can be evaluated with
  - ➡ SIMD
  - ➡ GPU
- Event generation will be released soon
  - ➡ Likely SIMD only first
- MadNis is also coming soon
  - ➡ Normalising Flow helps a lot

MadNIS — Basic functionality



Portability to CPU