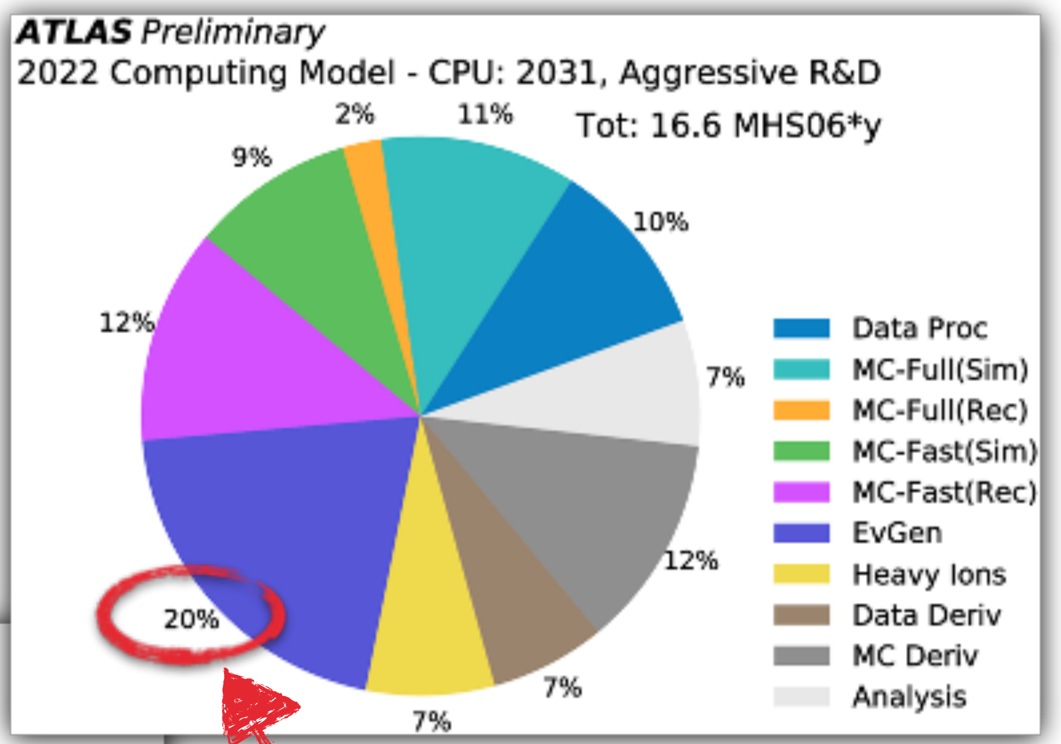
MADGRAPH5_AMC@NLO ON GPUS (AND VECTORIZED CPUS)

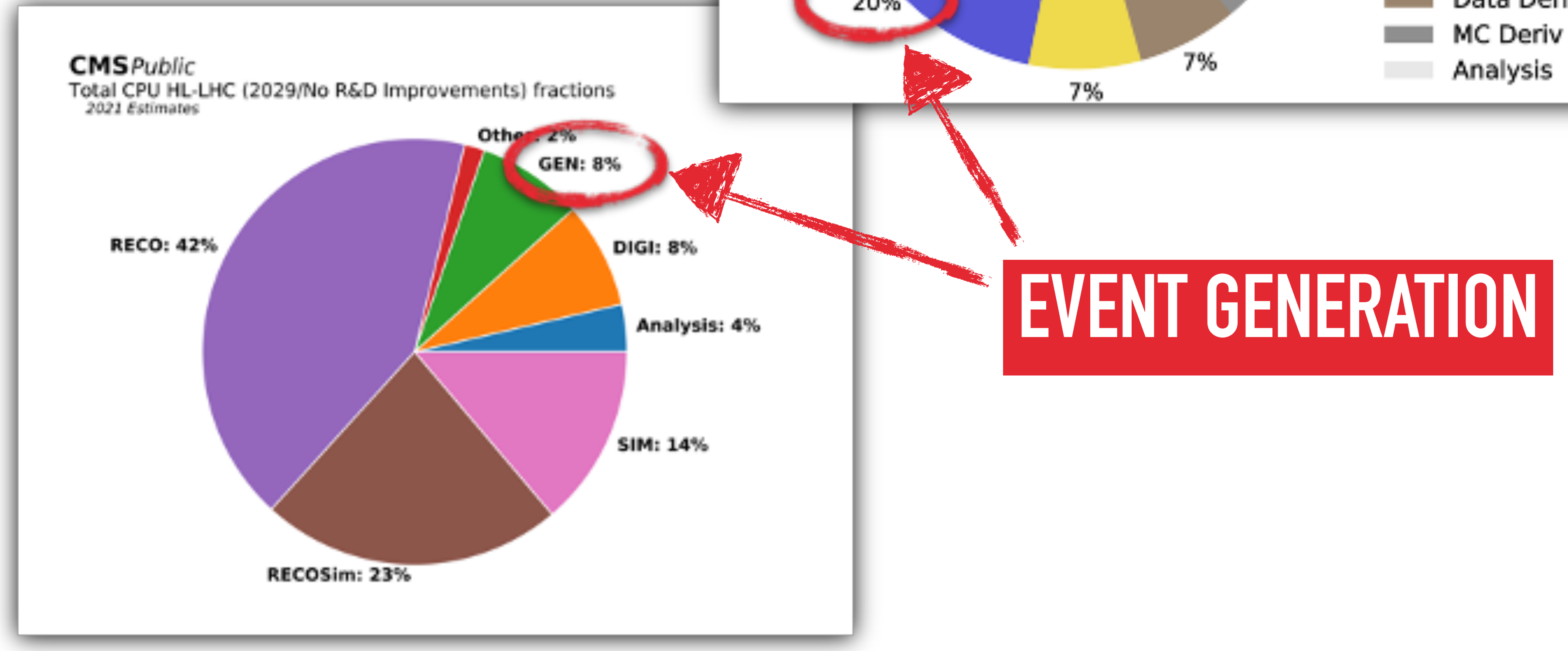
STEFAN ROISER, CERN

WLCG WORKSHOP, 8 NOV 2022

MADGRAPH5_AMC@NLO (MG5)

- MG5 is an important event generator package for HEP experiments
  - ATLAS & CMS HL-LHC usage for event generators is forecasted to 8 - 20 % of their computing budgets





- MG5 is a code generator, written in Python, to produce source code to calculate physics processes
  - Several backend languages for the generated code exist: Fortran, C, C++, Python

- The current production version of the package uses Fortran on (single-threaded) CPUs

MADGRAPH4GPU

- The Madgraph4GPU project aims to speed up the application execution by
  - offloading the parallelisable compute intensive parts of the workflow to GPUs
  - parallelise the execution on CPUs by leveraging on vector instructions
  - using of “abstraction layers” for compute accelerators
- Current and past contributions by:

Taylor Childers
Walter Hopkins
Nathan Nichols
Argonne National Laboratory

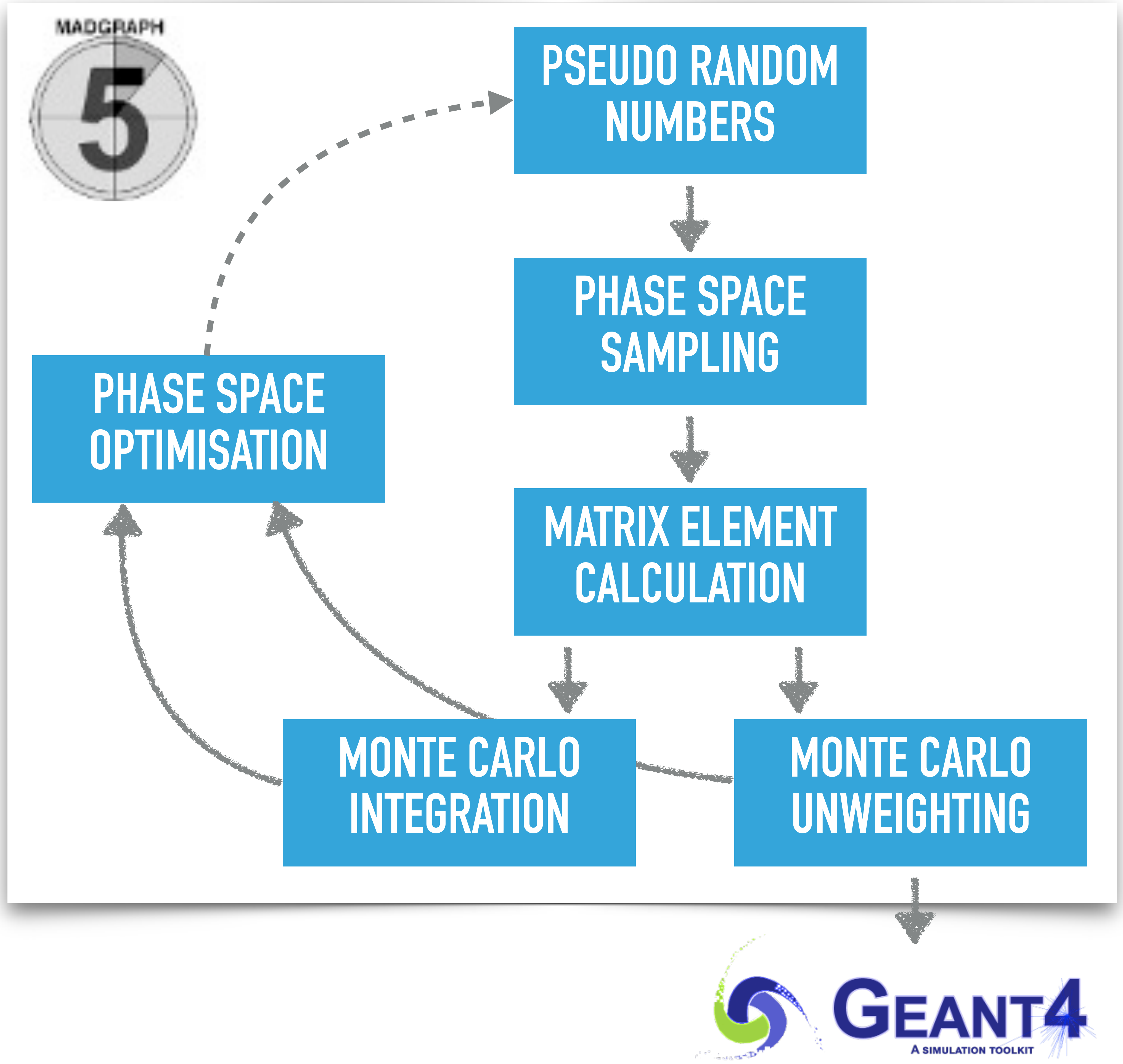
Laurence Field
Stephan Hageboeck
Stefan Roiser
David Smith
Jorgen Teig
Andrea Valassi
Zenny Wettersten
CERN

Olivier Mattelaer
UCL Université catholique de Louvain

Carl Vuosalo
Wisconsin University of Wisconsin-Madison

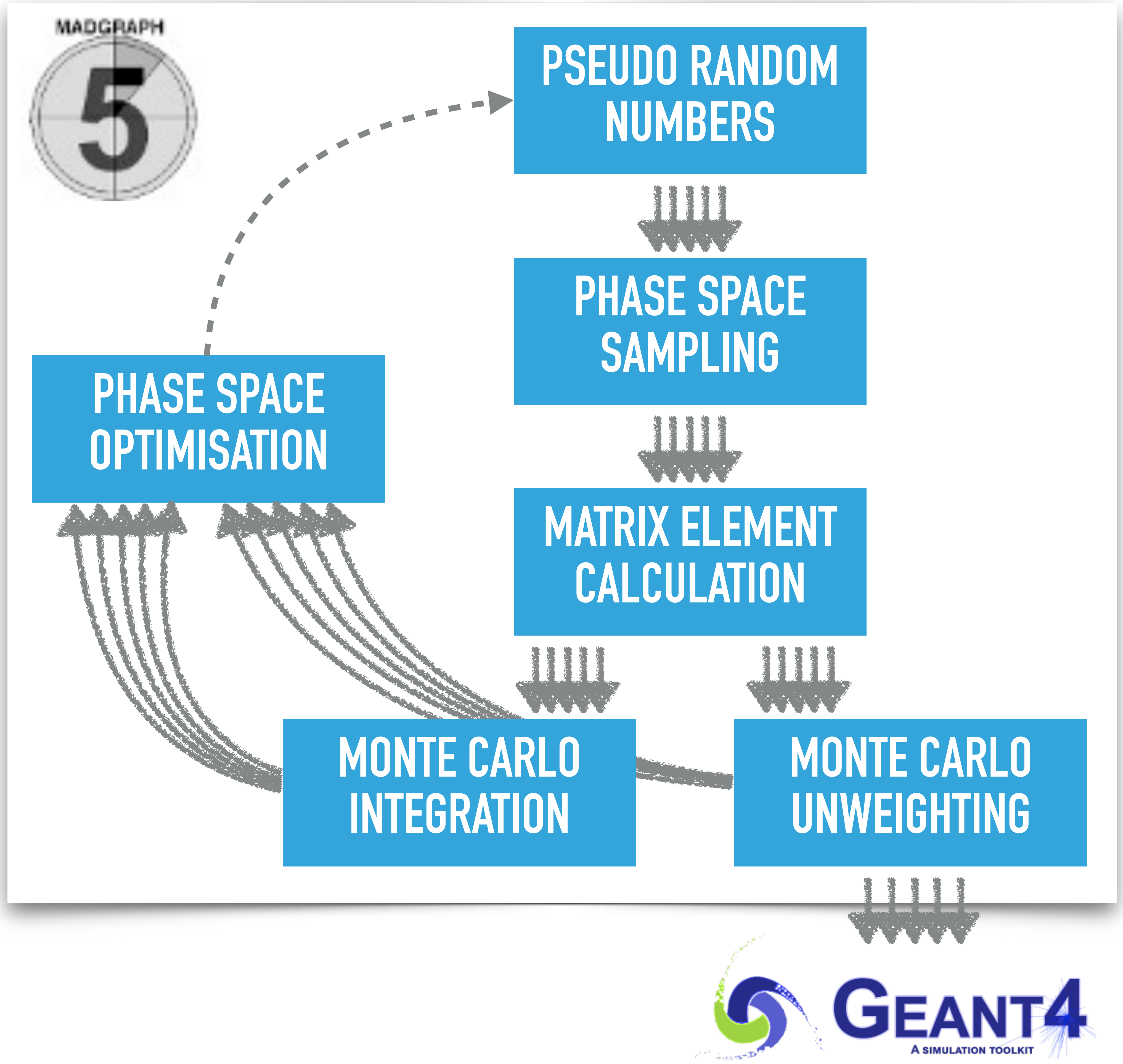
ANATOMY OF THE MADGRAPH EVENT GENERATOR

- No input data –> starts from random numbers



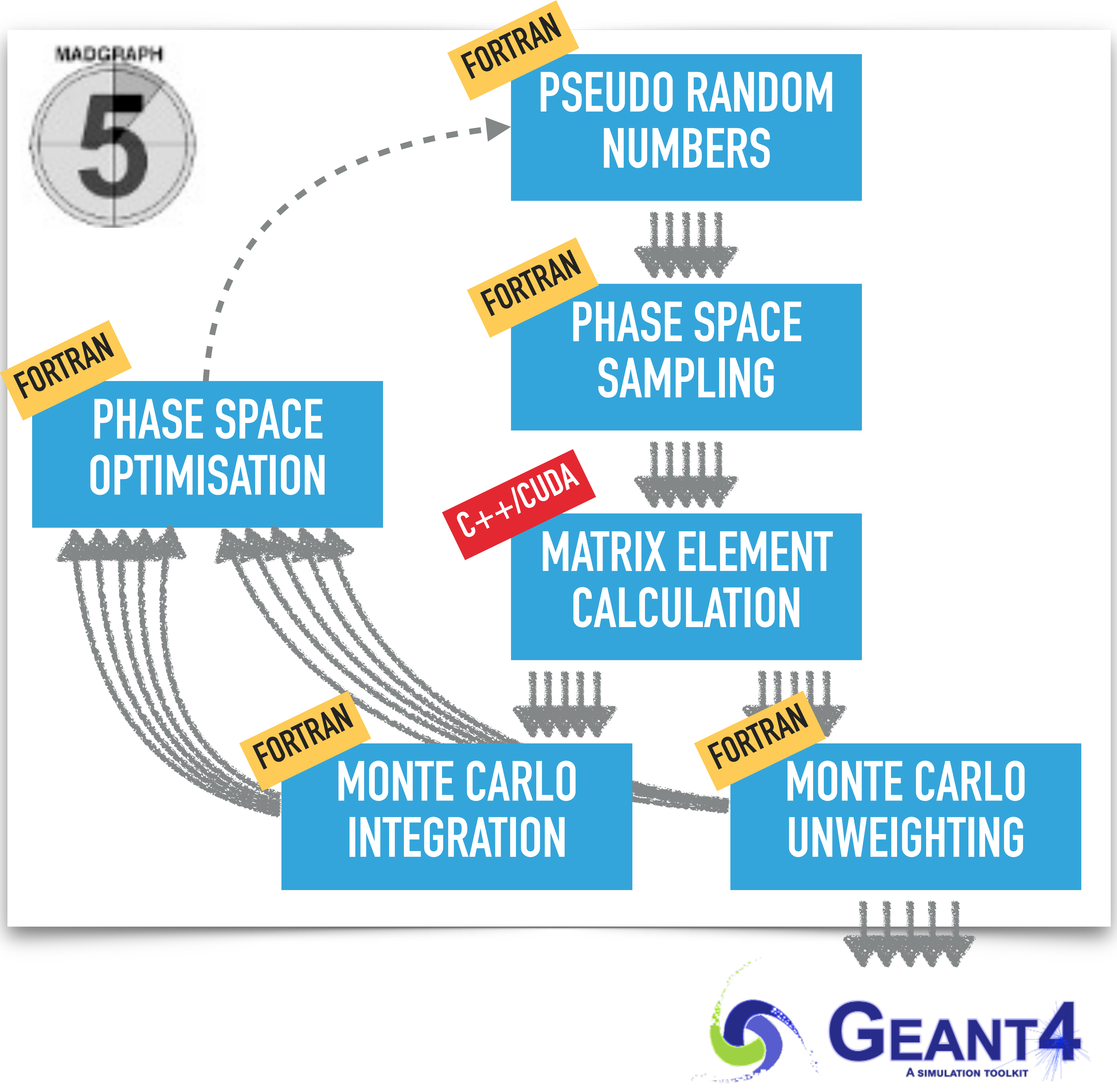
ANATOMY OF THE MADGRAPH EVENT GENERATOR

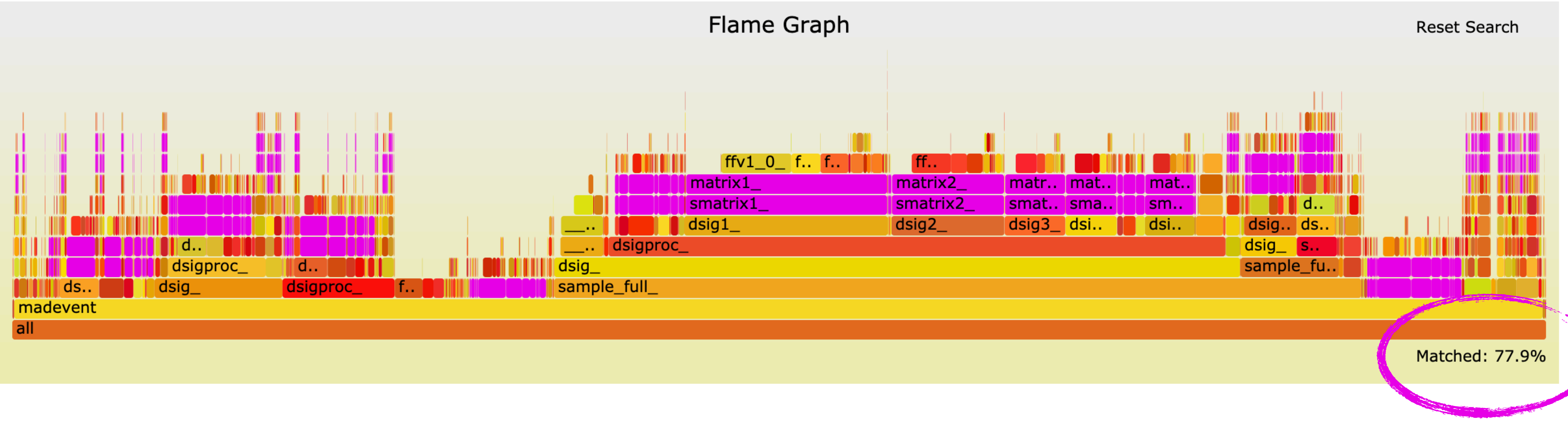
- No input data –> starts from random numbers
- Parallelise on the event level
  - Use vector registers on CPU
  - Massive parallelisation on GPUs



ANATOMY OF THE MADGRAPH EVENT GENERATOR

- No input data –> starts from random numbers
- Parallelise on the event level
- Bulk of compute time spent in matrix element calculations
- Speedup MEs in C++/Cuda
  - Rest stays in Fortran (for now)

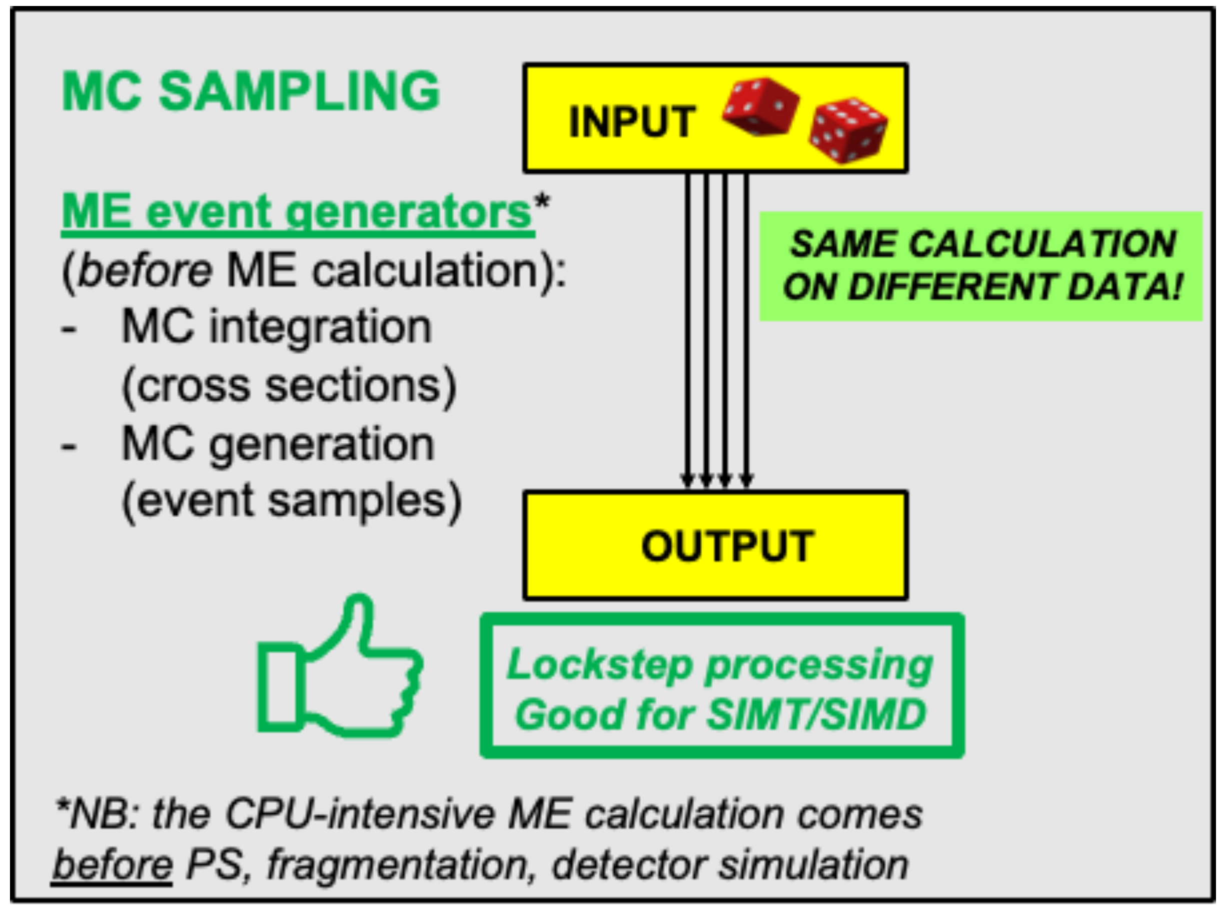


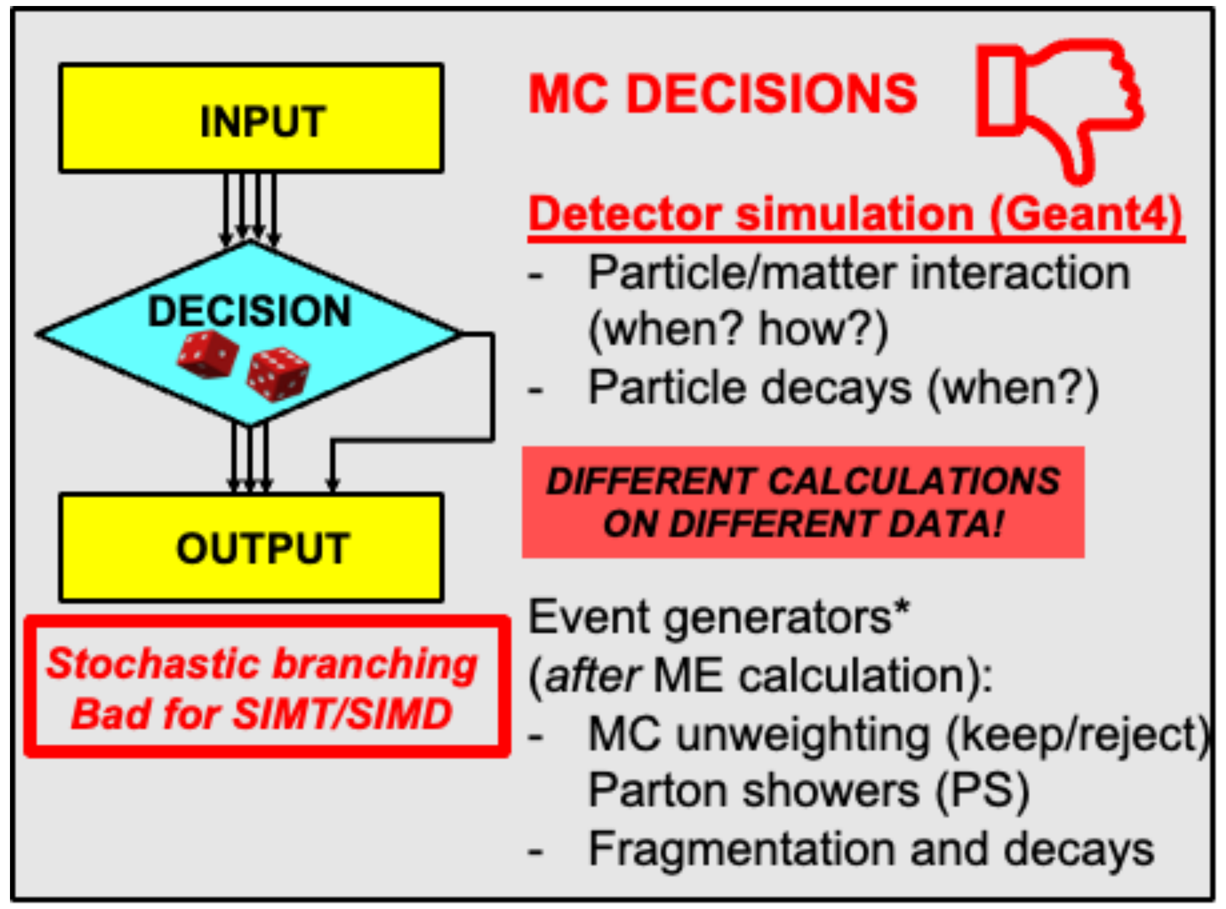


MATRIX ELEMENT CALCULATION

ADVANTAGES OF EVENT GENERATION PACKAGES FOR PERFORMANCE ENGINEERING

- No input data –> starts from random numbers
  - Event generation is the first application in the simulation chain
- No stochastic branching
  - Same execution path in every thread allows lockstep processing
- Easy refactoring of code
  - Matrix element calculations are a rather simple chain of function calls

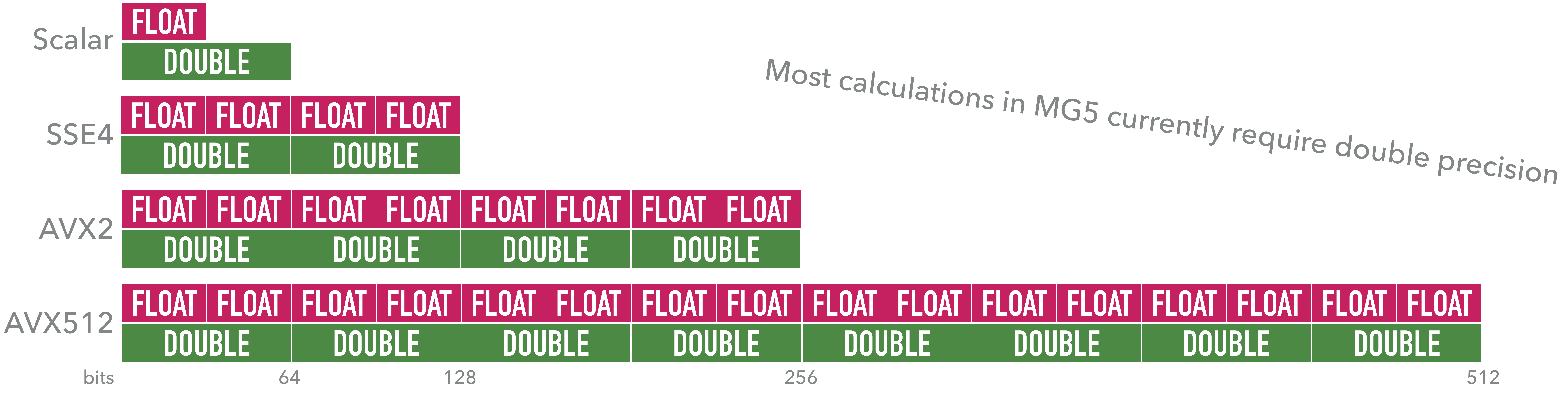




CPU VERSION

CPU DEVELOPMENTS

- CPU and GPU (Cuda) version share the same code base, differentiated by C++ macros
  - For the CPU implementation we parallelise on the even level via vector registers
- Main development platform are intel compatible CPUs
  - Depending on the CPU architecture capabilities the code will be compiled for different vector sizes (SSE4, AVX2, AVX512)



CURRENT RESULTS

- Current production version in double precision
  - float precision shown for information
- "Minimal" speedup for CPU execution at a factor x1.9 in double precision
- Potential for more speedups on machines with wider register sizes of up to a factor x6 to x8

$gg \to t\bar{t}gg$	MEs precision	madevent $t_{TOT} = t_{Mad} + t_{MEs}$ [sec]	N_{events}/t_{TOT} [events/sec]	N_{events}/t_{MEs} [MEs/sec]
Fortran(scalar)	double	37.3 = 1.7 + 35.6	2.20E3 (=1.0)	2.30E3 (=1.0)
C++/none(scalar)	double	37.8 = 1.7 + 36.0	2.17E3 (x1.0)	2.28E3 (x1.0)
C++/sse4(128-bit)	double	19.4 = 1.7 + 17.8	4.22E3 (x1.9)	4.62E3 (x2.0)
C++/avx2(256-bit)	double	9.5 = 1.7 + 7.8	8.63E3 (x3.9)	1.05E4 (x4.6)
C++/512y(256-bit)	double	8.9 = 1.8 + 7.1	9.29E3 (x4.2)	1.16E4 (x5.0)
C++/512z(512-bit)	double	6.1 = 1.8 + 4.3	1.35E4 (x6.1)	1.91E4 (x8.3)
C++/none(scalar)	float	36.6 = 1.8 + 34.9	2.24E3 (x1.0)	2.35E3 (x1.0)
C++/sse4(128-bit)	float	10.6 = 1.7 + 8.9	7.76E3 (x3.6)	9.28E3 (x4.1)
C++/avx2(256-bit)	float	5.7 = 1.8 + 3.9	1.44E4 (x6.6)	2.09E4 (x9.1)
C++/512y(256-bit)	float	5.3 = 1.8 + 3.6	1.54E4 (x7.0)	2.30E4 (x10.0)
C++/512z(512-bit)	float	3.9 = 1.8 + 2.1	2.10E4 (x9.6)	3.92E4 (x17.1)

Intel Gold 6148, gcc 11.2

  - Further reductions of the Fortran part (madevent overhead) seem to be feasible
  - Can we draw advantage vectorisation on e.g. WLCG resources?

GPU VERSION

CURRENT RESULTS

- Development of the GPU version in Cuda
    - Started from C/C++ output of MG5
    - Matrix element calculations offloaded to GPU
- Depending on underlying physics process (gg->ttgg, gg->ttggg) speedups x19 to x63
    - Similarly to the CPU version further improvements seem possible
    - Overall execution dominated by the Fortran part

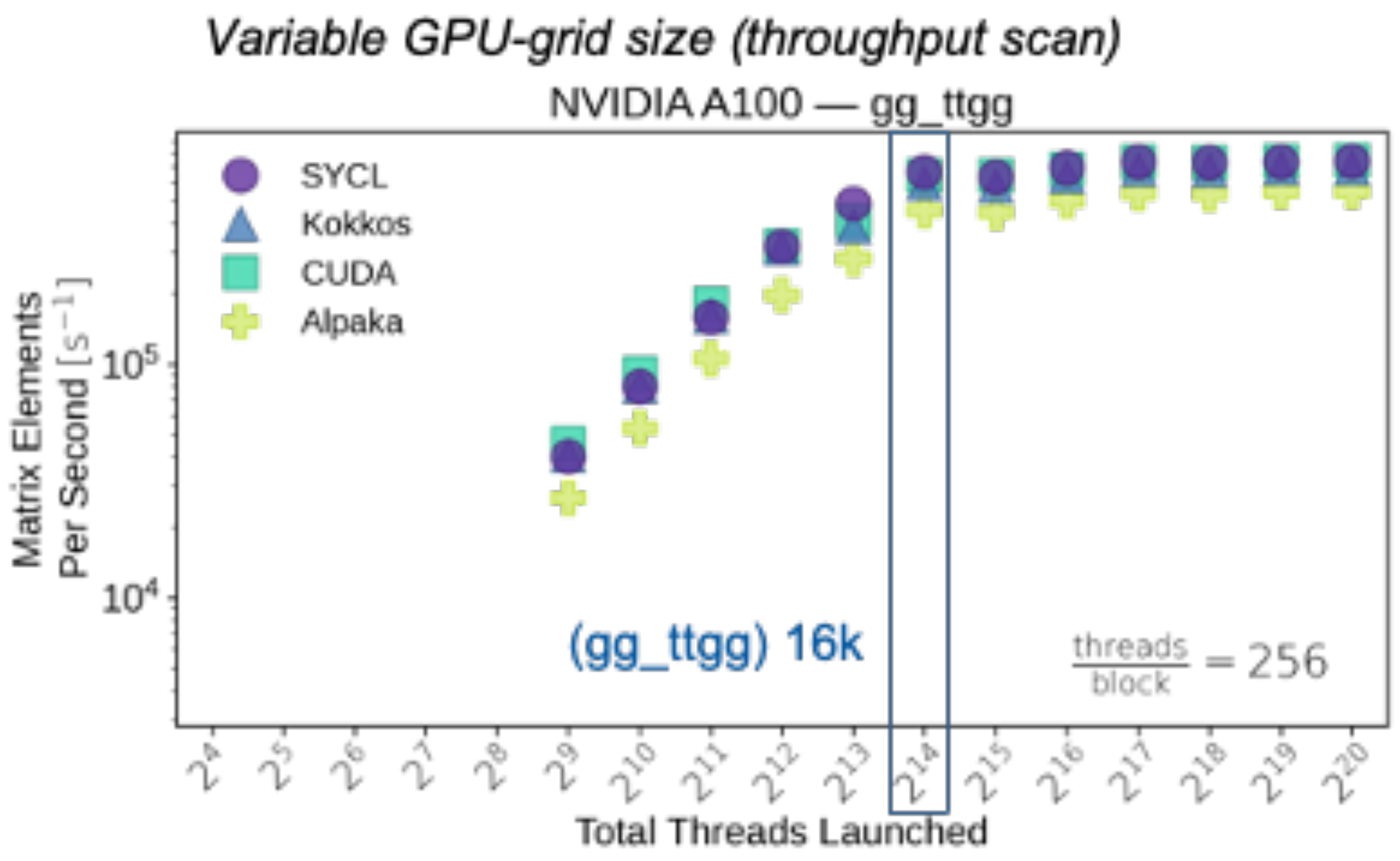
		madevent		
CUDA grid size		8192		
$gg \to t\bar{t}gg$	MEs precision	$t_{TOT} = t_{Mad} + t_{MEs}$ [sec]	N_{events}/t_{TOT} [events/sec]	N_{events}/t_{MEs} [MEs/sec]
Fortran	double	55.4 = 2.4 + 53.0	1.63E3 (=1.0)	1.70E3 (=1.0)
CUDA	double	2.9 = 2.6 + 0.35	3.06E4 (x18.8)	2.60E5 (x152)
CUDA	float	2.8 = 2.6 + 0.24	3.24E4 (x19.9)	3.83E5 (x225)

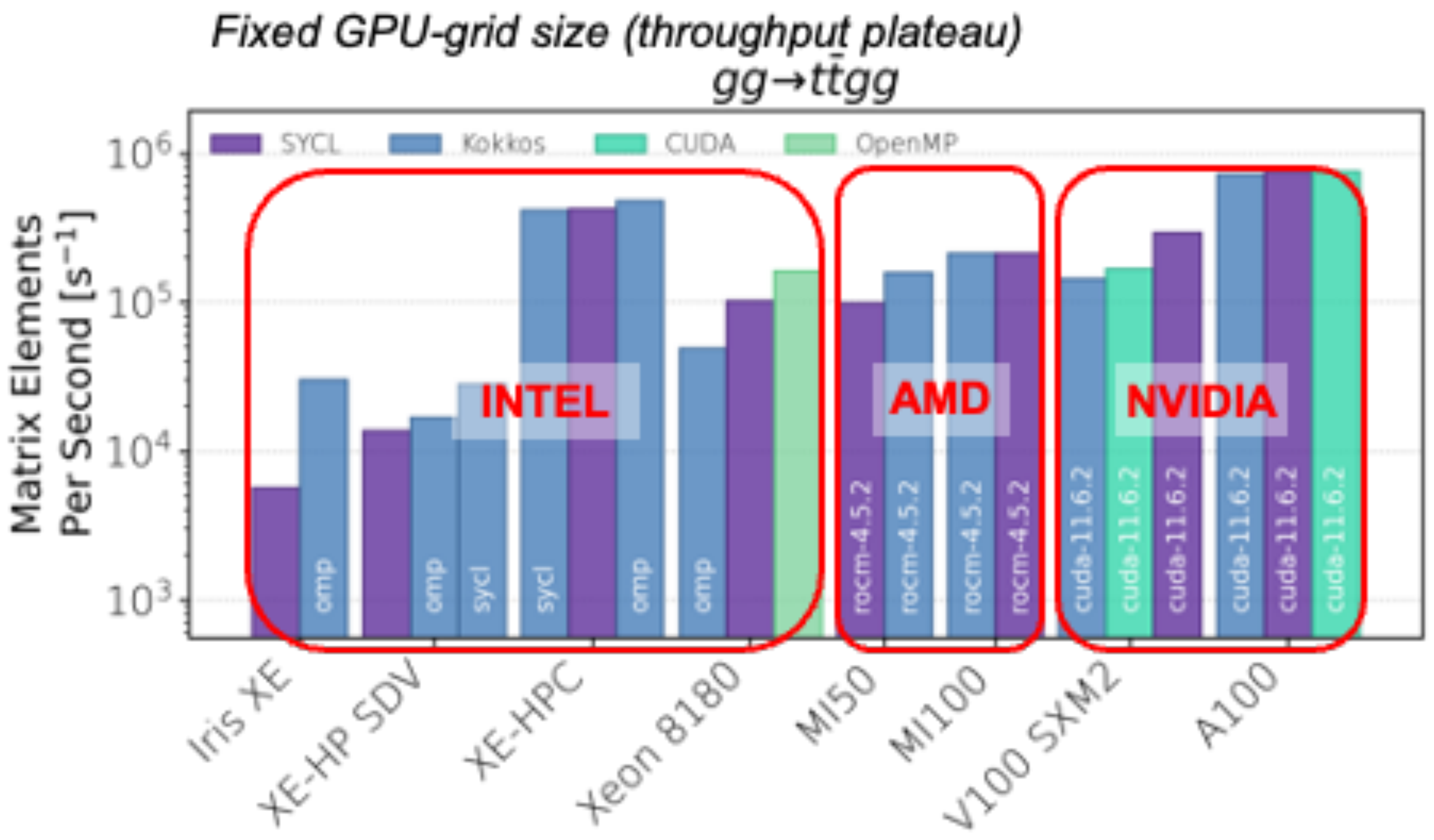
NVidia V100, Cuda 11.7, gcc 11.2

		madevent		
CUDA grid size		8192		
$gg \to t\bar{t}ggg$	MEs precision	$t_{TOT} = t_{Mad} + t_{MEs}$ [sec]	N_{events}/t_{TOT} [events/sec]	N_{events}/t_{MEs} [MEs/sec]
Fortran	double	1228.2 = 5.0 + 1223.2	7.34E1 (=1.0)	7.37E1 (=1.0)
CUDA	double	19.6 = 7.4 + 12.1	4.61E3 (x63)	7.44E3 (x100)
CUDA	float	11.7 = 6.2 + 5.4	7.73E3 (x105)	1.66E4 (x224)
CUDA	mixed	16.5 = 7.0 + 9.6	5.45E3 (x74)	9.43E3 (x128)

NVidia V100, Cuda 11.7, gcc 11.2

USE OF ABSTRACTION LAYERS

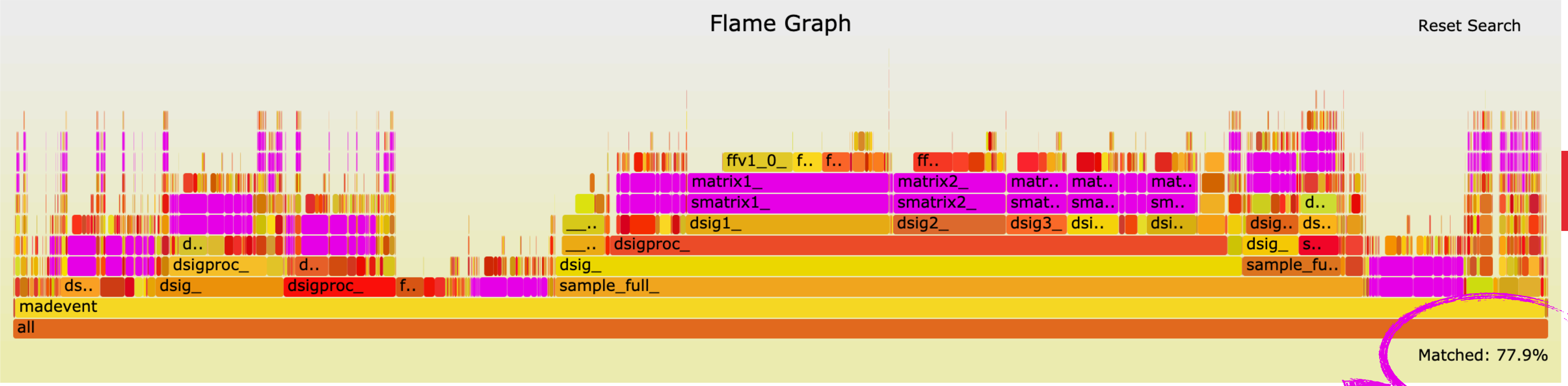




- Performance of SYCL and Kokkos are comparable to the CUDA implementation
- SYCL and Kokkos run out of the box also on AMD and Intel GPUs
  - They also run out of the box on CPUs – performance still under investigation

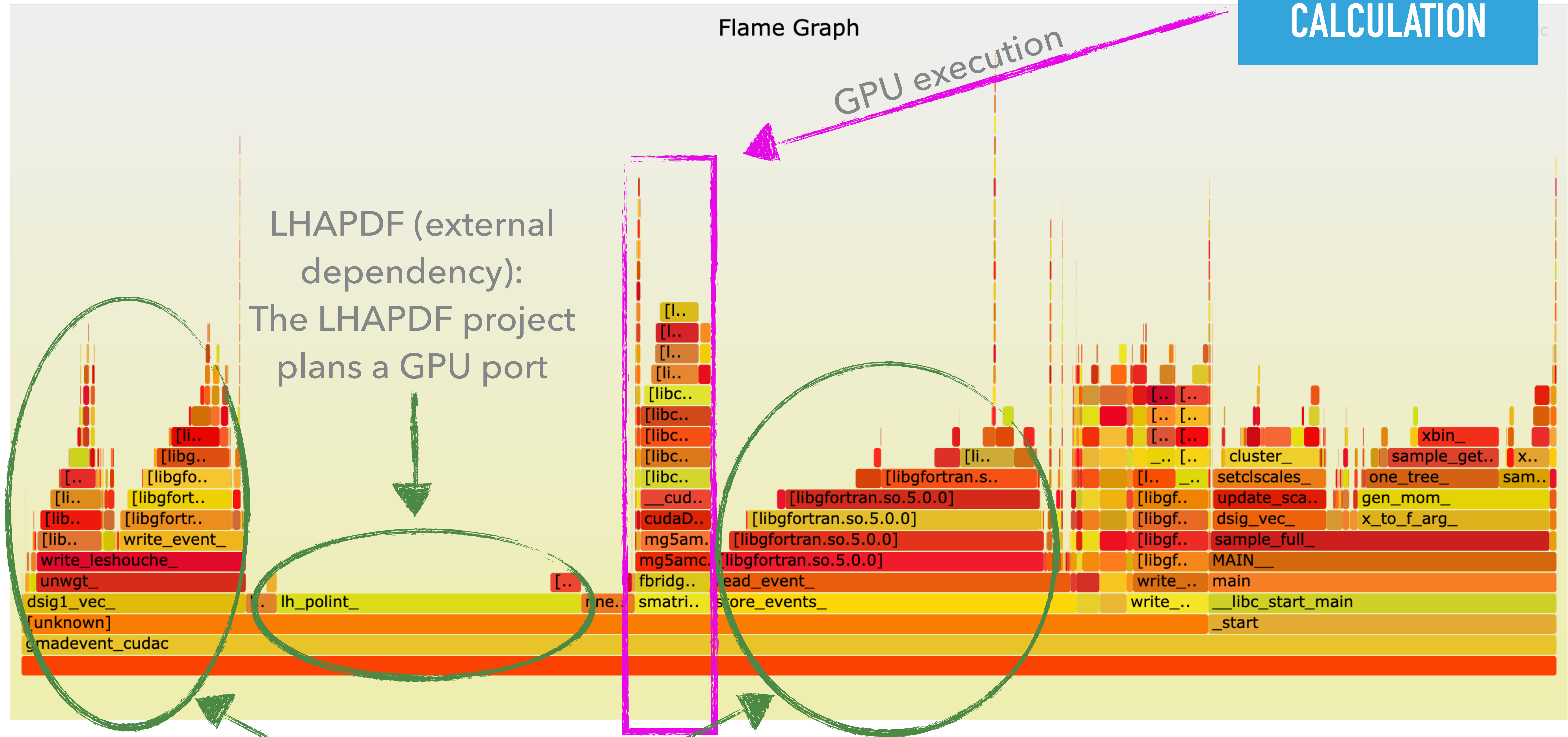
Xe-HP is a software development vehicle for functional testing only - currently used at Argonne and other customer sites to prepare their code for future Intel data centre GPUs
XE-HPC is an early implementation of the Aurora GPU

MORE IDEAS FOR IMPROVEMENTS



BEFORE – FORTRAN/CPU

MATRIX ELEMENT CALCULATION



CURRENT – CUDA/GPU

Monte Carlo unweighting – ideas on how to port it to GPU

NB: The two flamegraphs don't represent the same physics processes. The comparisons here show orders of magnitude

FUTURE DIRECTIONS

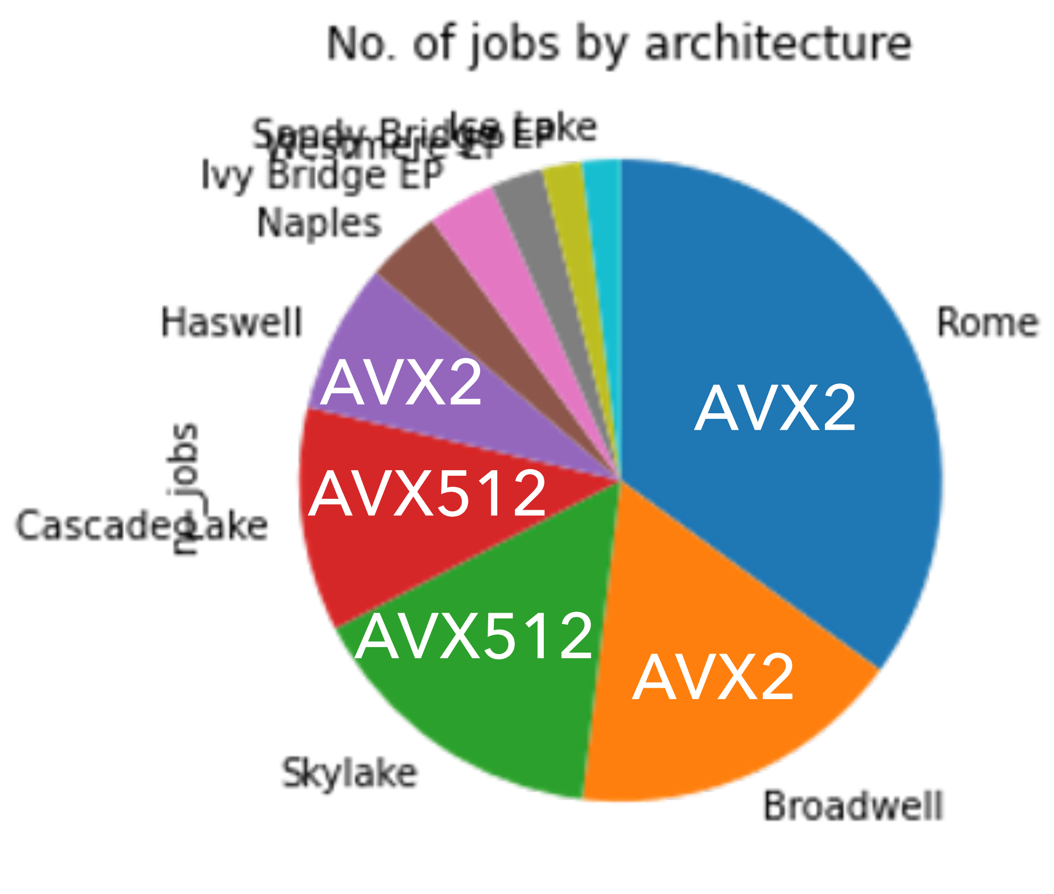
- Preparation of a production version
  - Two minor features still missing (extract random color, random helicity)
  - Aiming for a first alpha version usable by experiments in the coming months
- Developments for further speedups
  - Port / use more GPU ported parts of the workflow
  - Use of NVidia/cublas for a matrix multiplication inside the matrix element code
  - Efficient hybrid host / device execution
- Also working on a re-weighting version of Madgraph based on the new code
- Include also NLO calculations (the current version uses LO)

BONUS SLIDE

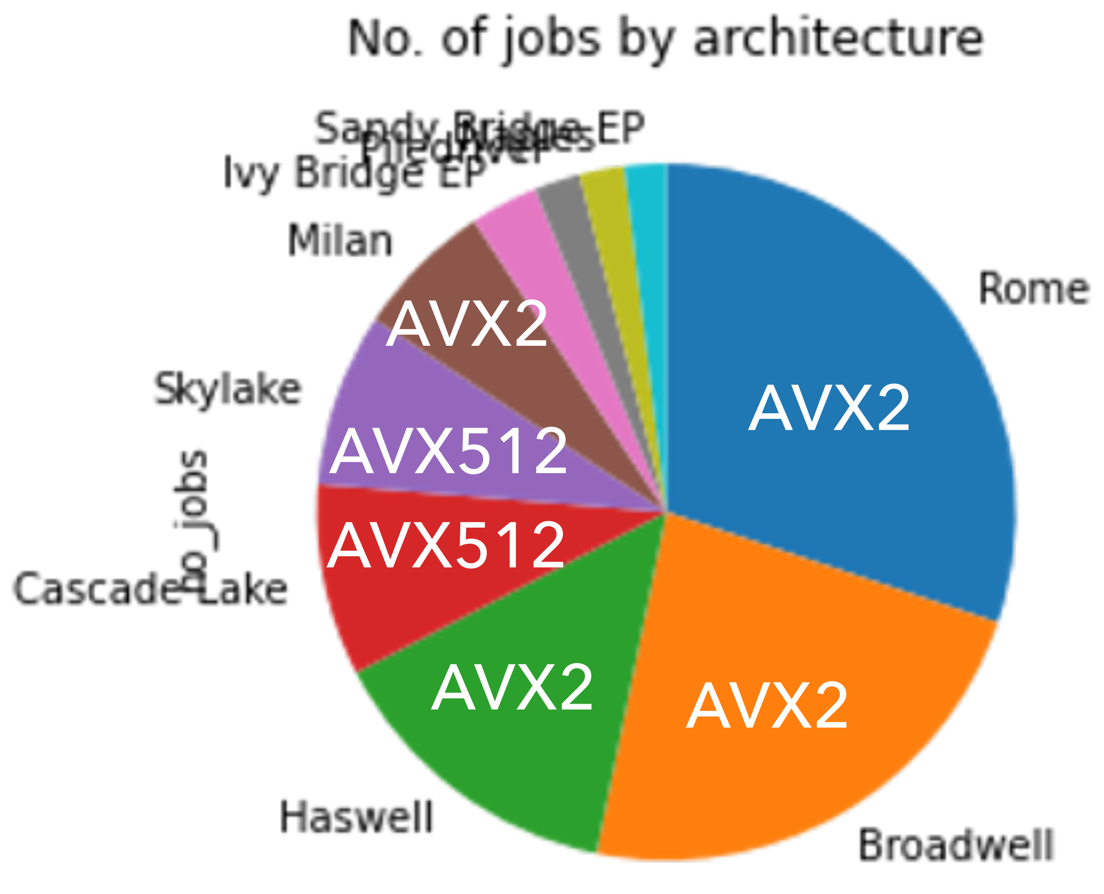
IS IT WORTH TO USE VECTOR INSTRUCTIONS ON WLCG FOR MG5?

BONUS SLIDE – IS IT WORTH TO USE VECTOR INSTRUCTIONS ON WLCG FOR MG5?

ATLAS



CMS



representative sample of WLCG grid jobs in 2022
A. Sciaba, https://indico.cern.ch/event/1072141/

		madevent		
$gg \to t\bar{t}gg$	MEs precision	$t_{TOT} = t_{Mad} + t_{MEs}$ [sec]	N_{events}/t_{TOT} [events/sec]	N_{events}/t_{MEs} [MEs/sec]
Fortran(scalar)	double	37.3 = 1.7 + 35.6	2.20E3 (=1.0)	2.30E3 (=1.0)
C++/none(scalar)	double	37.8 = 1.7 + 36.0	2.17E3 (x1.0)	2.28E3 (x1.0)
C++/sse4(128-bit)	double	19.4 = 1.7 + 17.8	4.22E3 (x1.9)	4.62E3 (x2.0)
C++/avx2(256-bit)	double	9.5 = 1.7 + 7.8	8.63E3 (x3.9)	1.05E4 (x4.6)
C++/512y(256-bit)	double	8.9 = 1.8 + 7.1	9.29E3 (x4.2)	1.16E4 (x5.0)
C++/512z(512-bit)	double	6.1 = 1.8 + 4.3	1.35E4 (x6.1)	1.91E4 (x8.3)
C++/none(scalar)	float	36.6 = 1.8 + 34.9	2.24E3 (x1.0)	2.35E3 (x1.0)
C++/sse4(128-bit)	float	10.6 = 1.7 + 8.9	7.76E3 (x3.6)	9.28E3 (x4.1)
C++/avx2(256-bit)	float	5.7 = 1.8 + 3.9	1.44E4 (x6.6)	2.09E4 (x9.1)
C++/512y(256-bit)	float	5.3 = 1.8 + 3.6	1.54E4 (x7.0)	2.30E4 (x10.0)
C++/512z(512-bit)	float	3.9 = 1.8 + 2.1	2.10E4 (x9.6)	3.92E4 (x17.1)

- Vast majority of ATLAS and CMS jobs from 2022 provide AVX2 vectorisation or better
  - The Madgraph the throughput on those nodes would potentially increase ~ x 4

SUMMARY

- A port for LO calculations of the Madgraph5_aMC@NLO event generator for vectorized CPUs and GPUs will be provided in the coming months
  - In double precision maximum speedups of x8 (CPU) and x60 (GPU) have been achieved
  - Further optimisations for the Madgraph5_aMC@NLO workflow are being worked on
  - A software version using an abstraction layer is also being prepared
- The majority of WLCG grid worker nodes currently provide AVX2 vectorisation or better
  - Using those worker nodes would provide a factor x4 throughput increase for Madgraph5_aMC@NLO workflow execution