MadNIS

A journey towards the first ML event generator

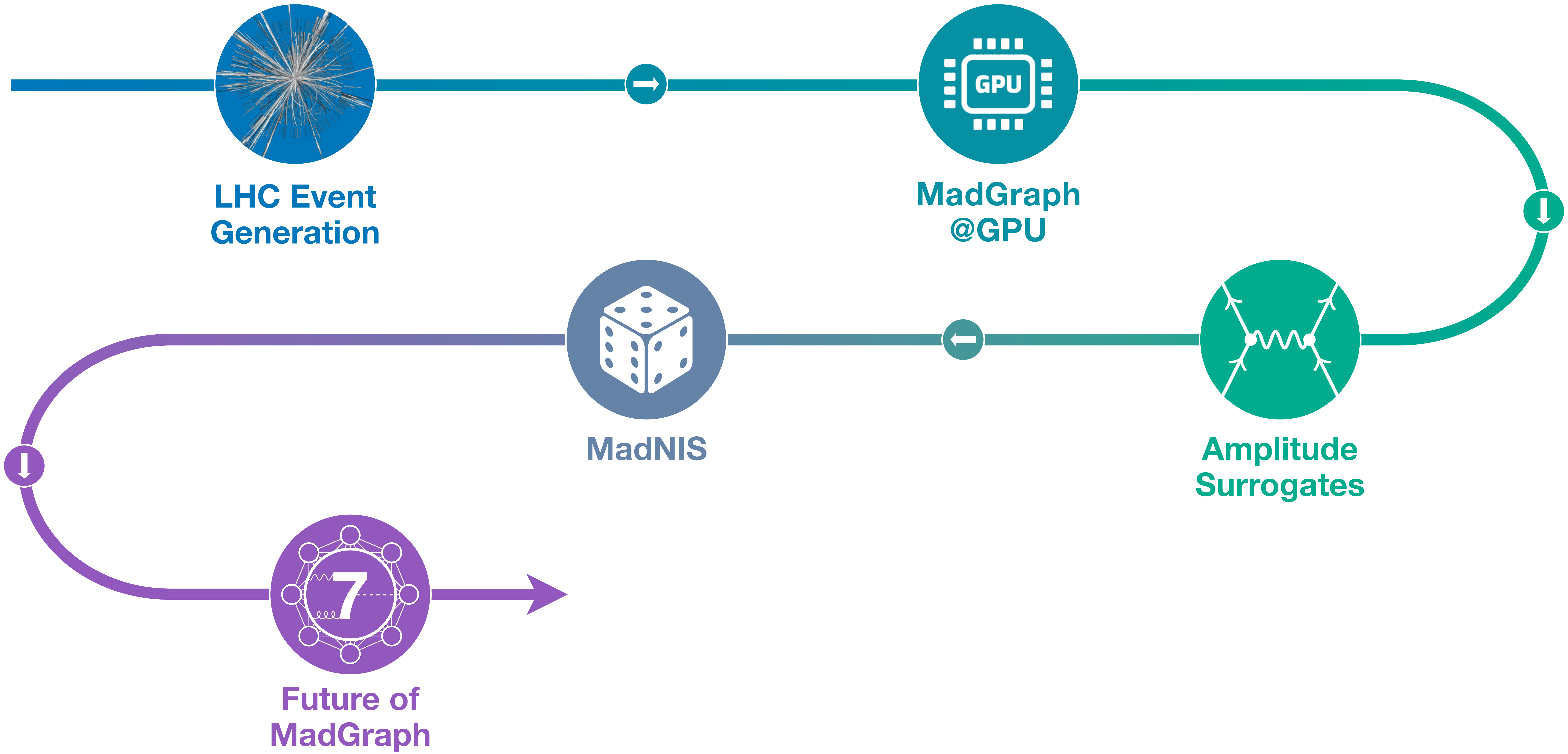
UNIVERSITÀ DEGLI STUDI DI MILANO

INFN MILANO

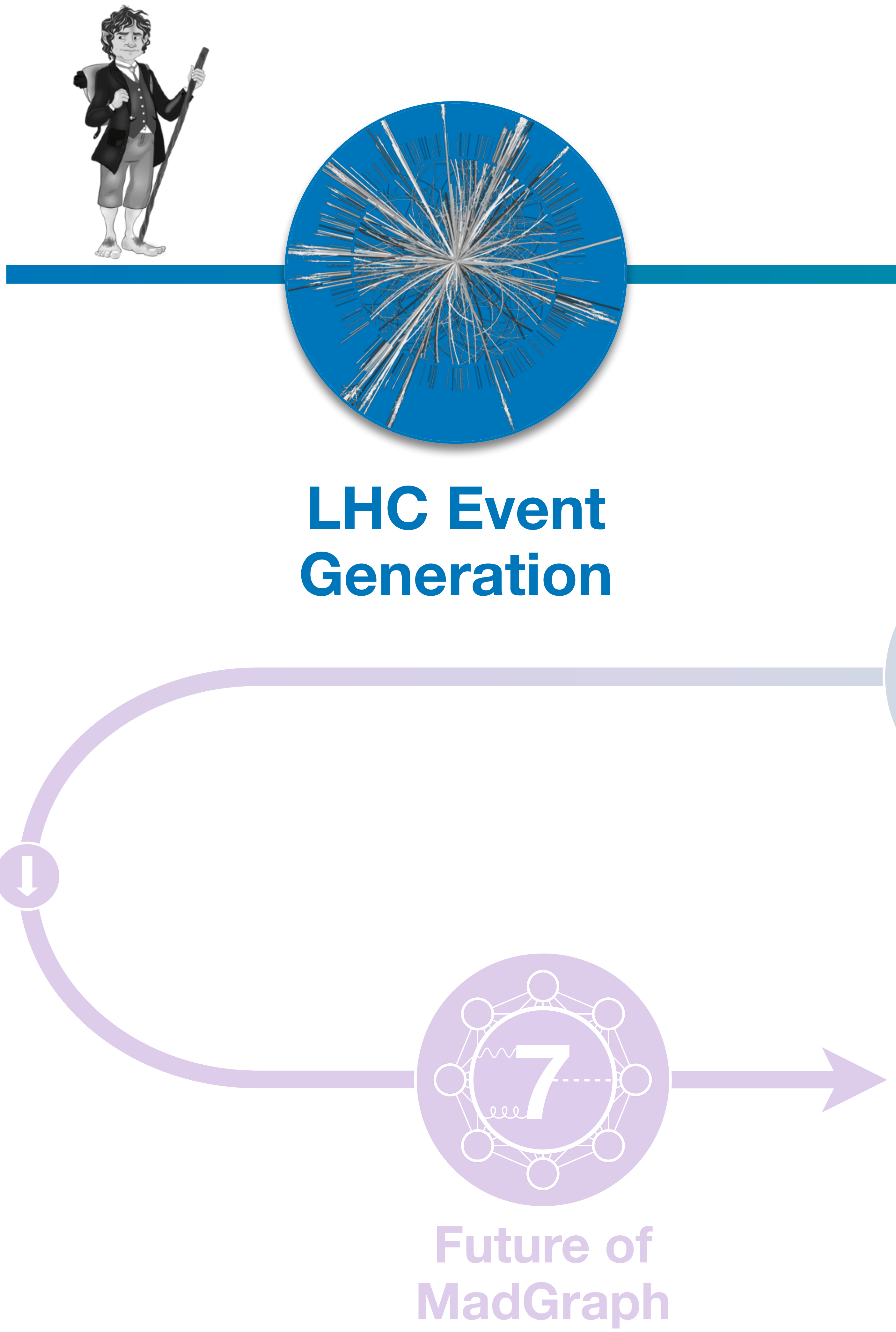
ML4Jets 2024 | Paris Sorbonne Campus

Ramon Winterhalder

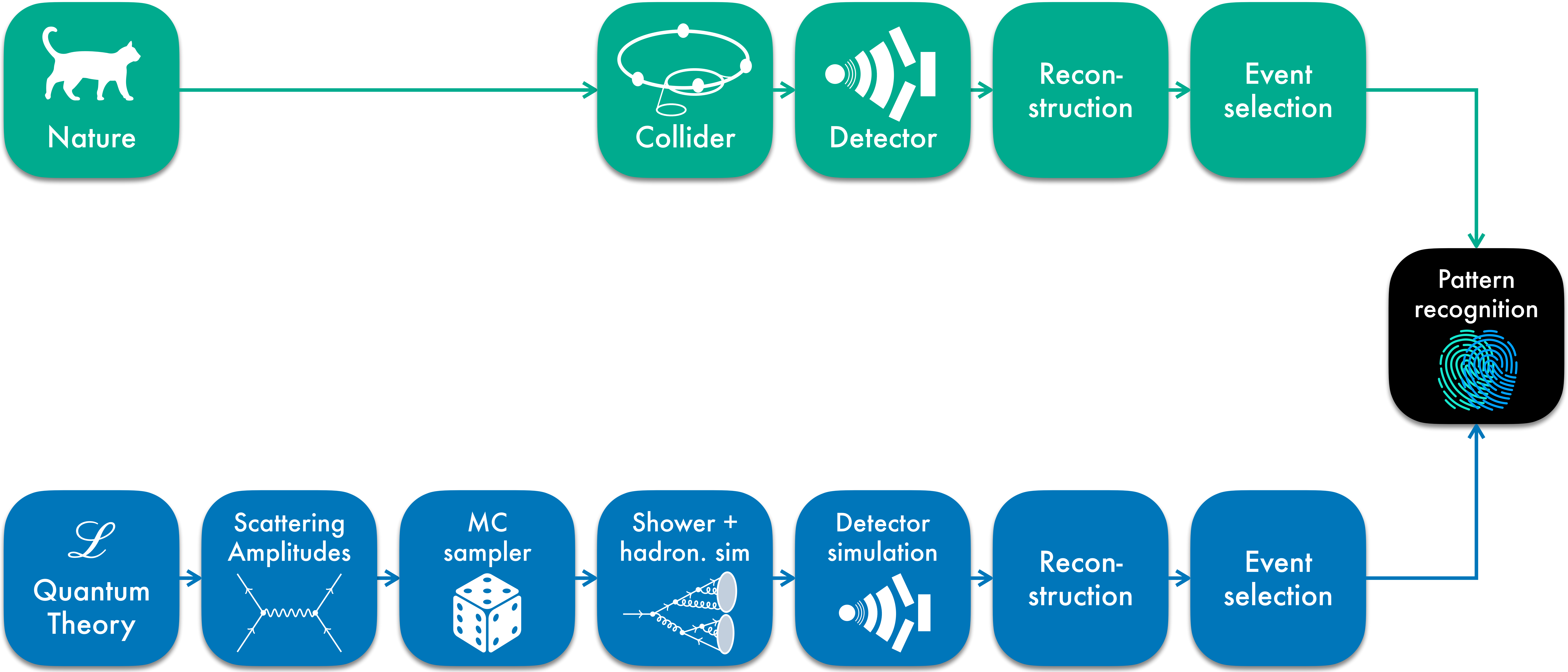
A journey towards the first ML event generator



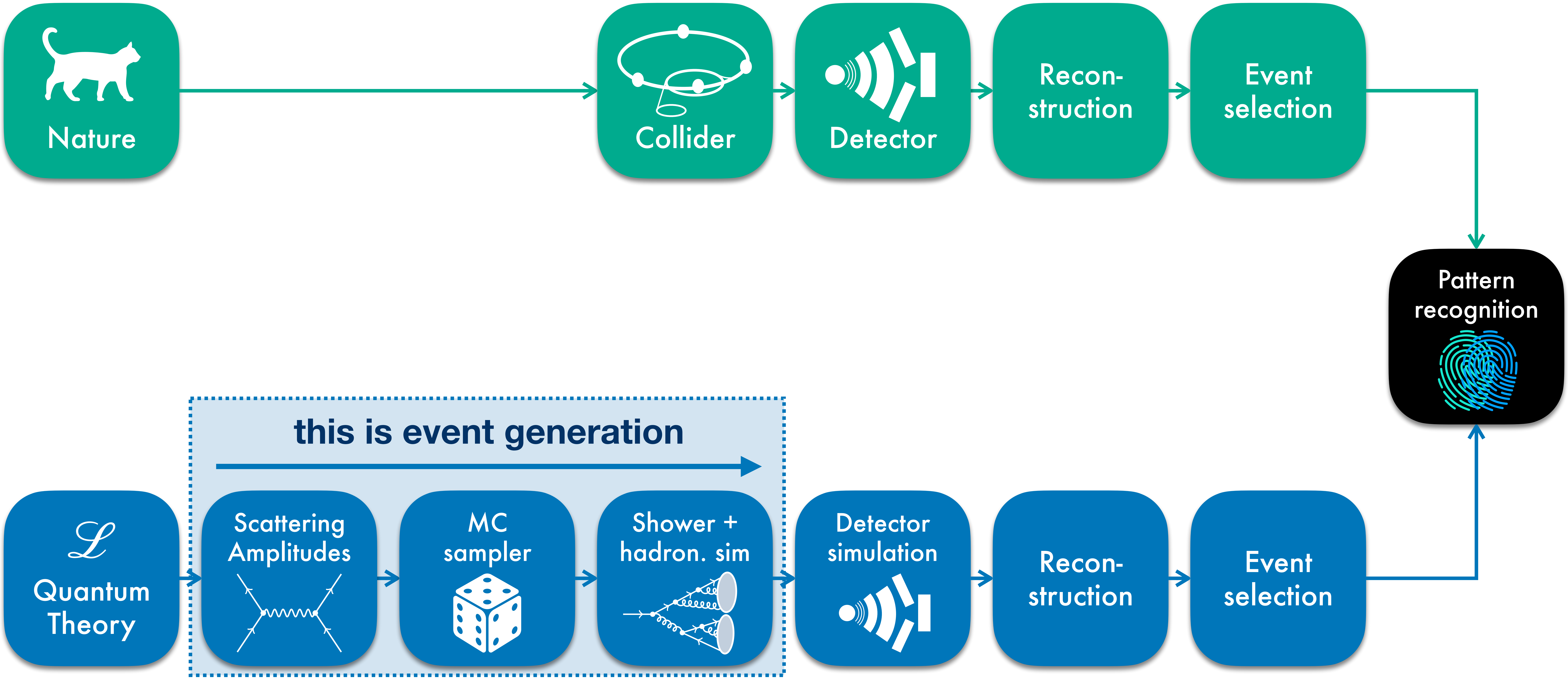
LHC event generation



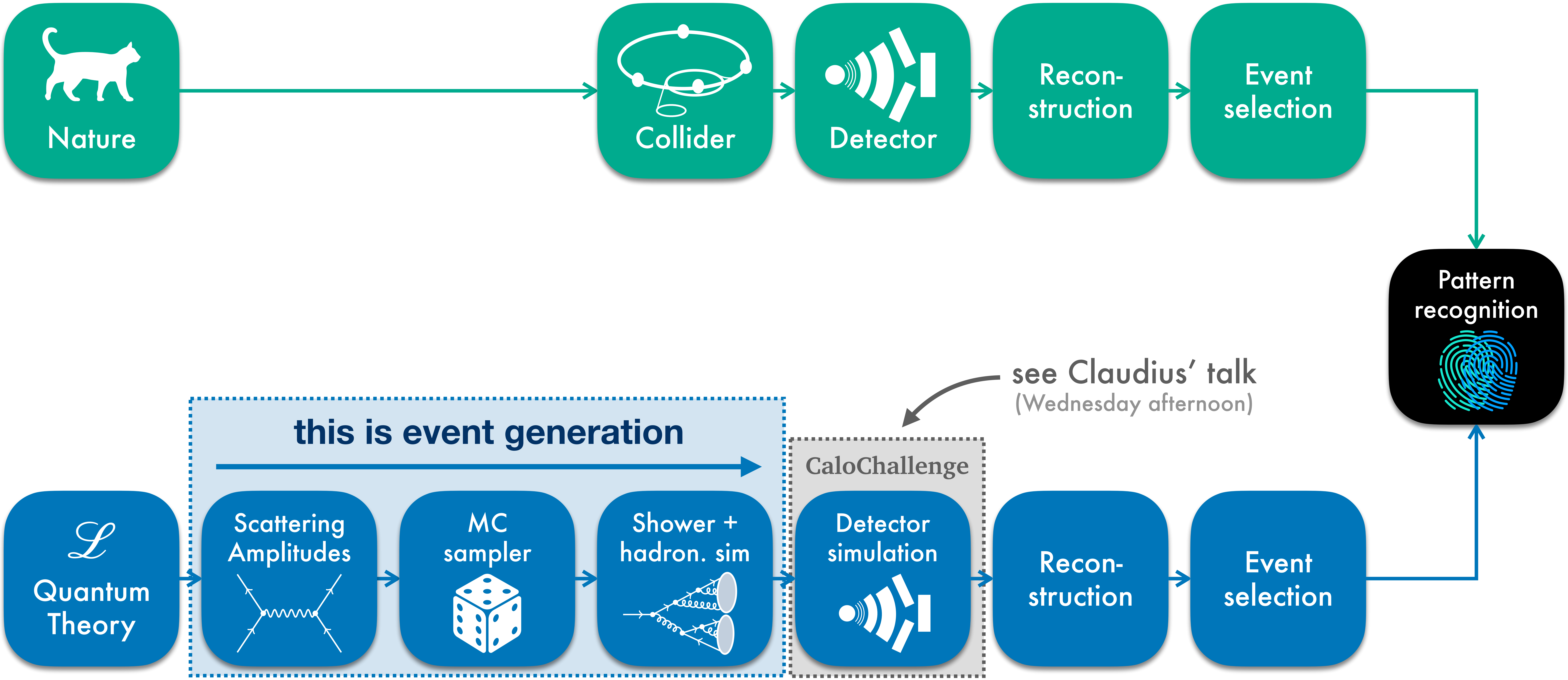
LHC analysis in a nutshell



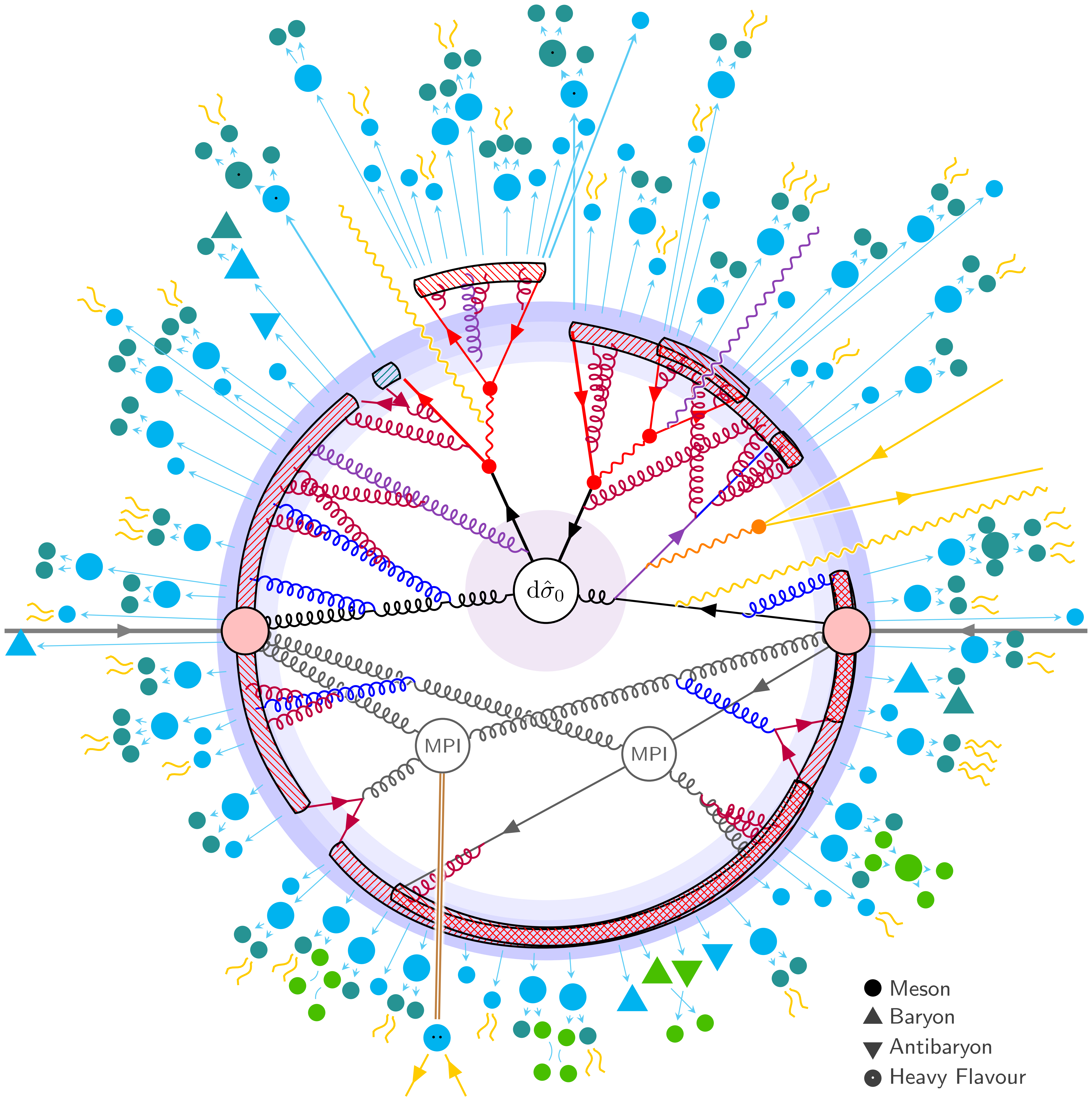
LHC analysis in a nutshell



LHC analysis in a nutshell



The devil is in the details

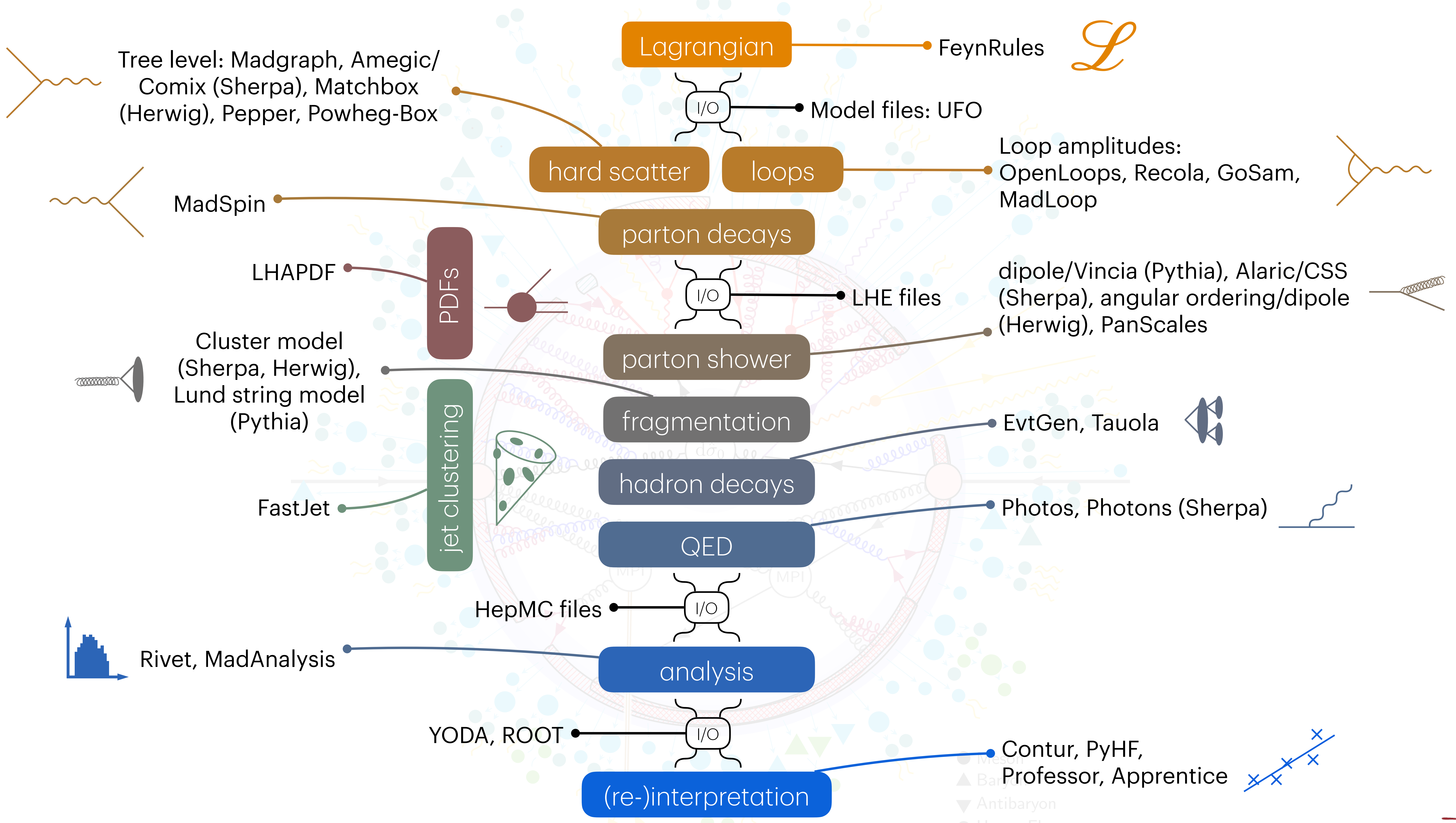


- Hard Interaction
- Resonance Decays
- MECs, Matching & Merging
- FSR
- ISR*
- QED
- Weak Showers
- Hard Onium
- Multiparton Interactions
- Beam Remnants*
- Strings
- Ministrings / Clusters
- Colour Reconnections
- String Interactions
- Bose-Einstein & Fermi-Dirac
- Primary Hadrons
- Secondary Hadrons
- Hadronic Reinteractions

(: incoming lines are crossed)*

Pythia8 Manual [2203.11601]

The devil is in the details



Pythia8 Manual [2203.11601]

Why do we need the LHC simulation chain to run faster?

Ask ChatJesseT

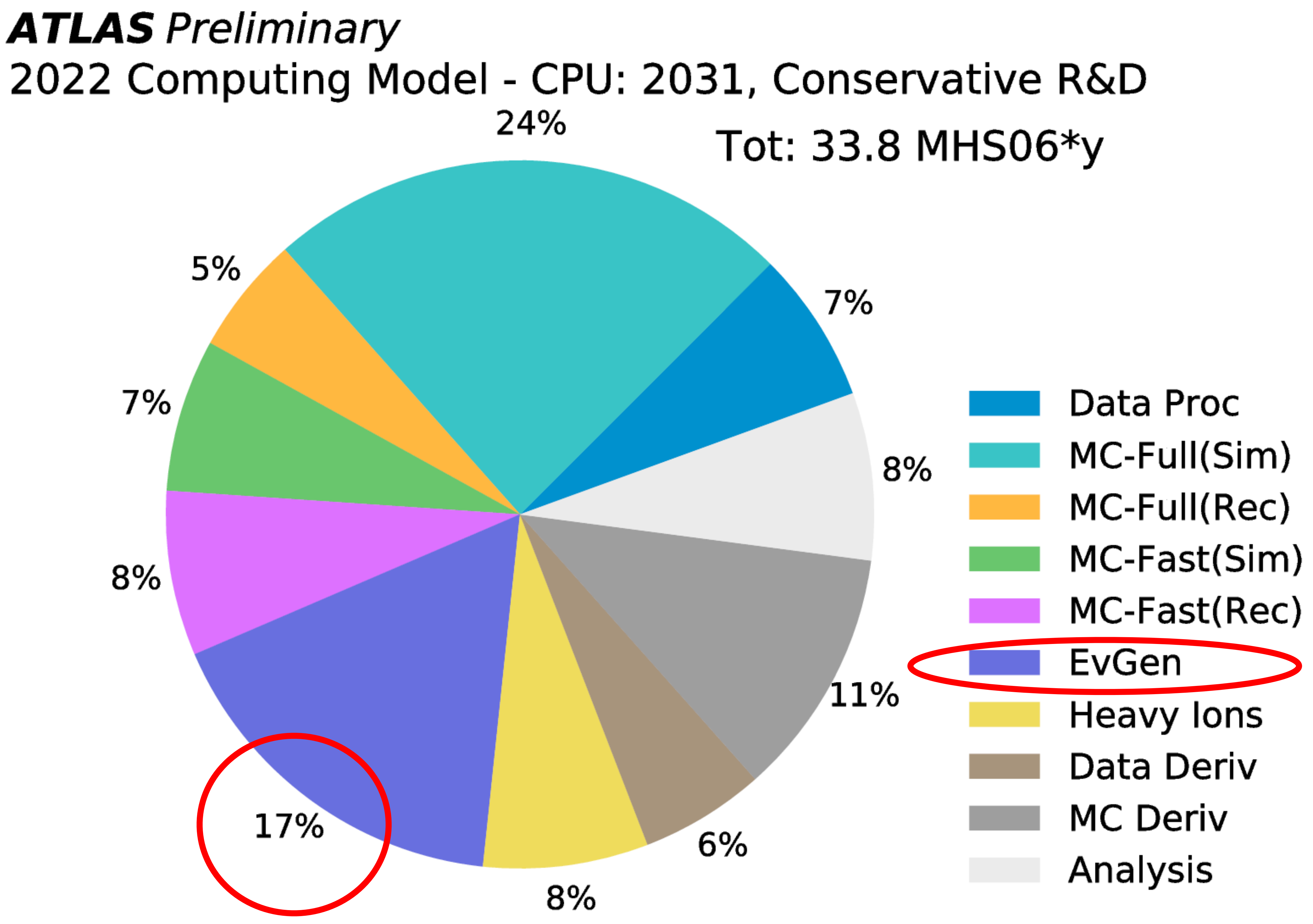
ChatJesseT

Question: *Why do we need the LHC simulation chain to run faster?*

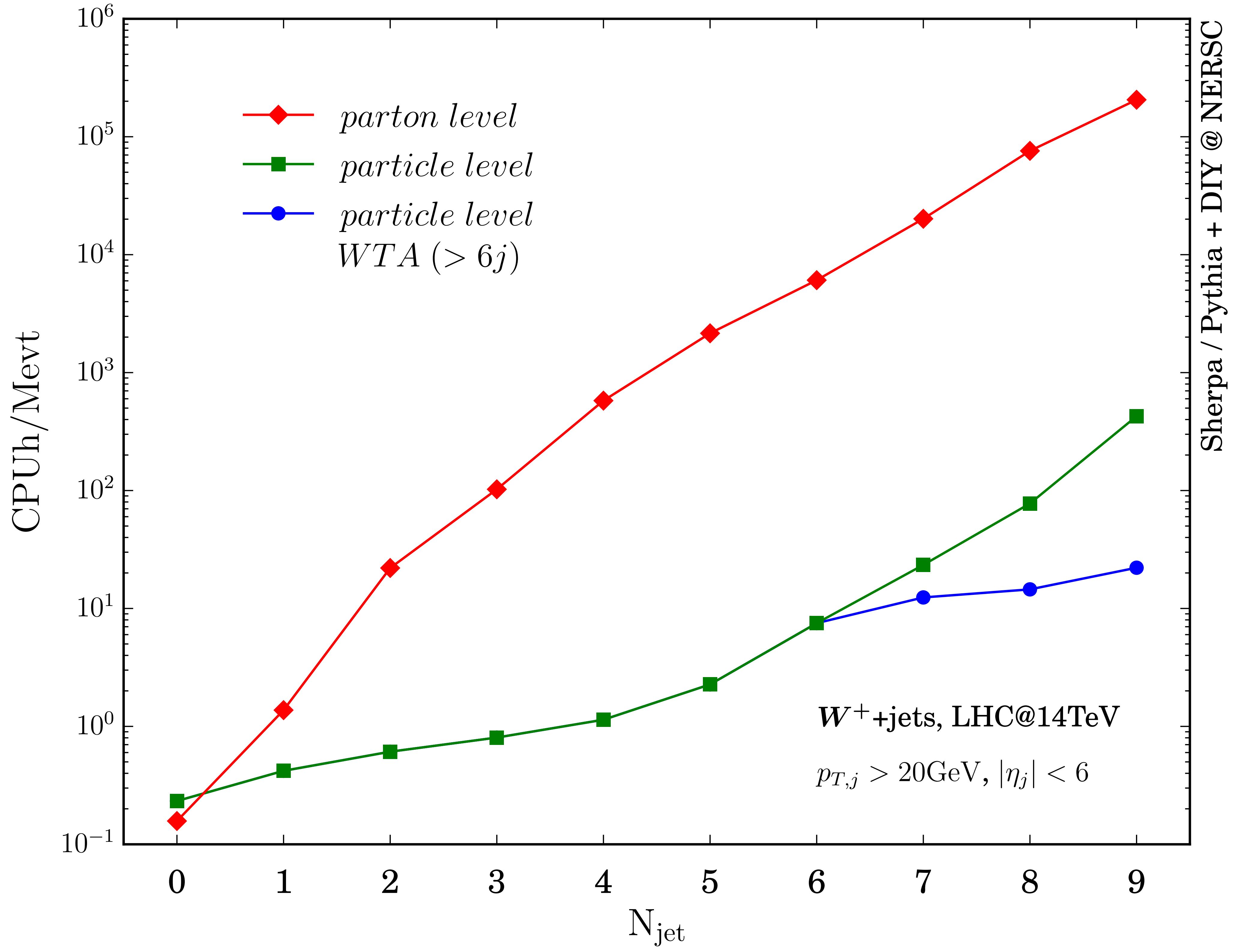
Ah, a question right up my particle accelerator alley! I'm Jesse Thaler, theoretical physicist extraordinaire and Director of the NSF AI Institute for Artificial Intelligence and Fundamental Interactions (IAIFI). When it comes to the need for the LHC simulation chain to run faster, it's all about striking while the iron is hot, or should I say, while the particles are colliding at near-light speeds! You see, in the high-energy world of particle physics, time is of the essence. The faster we can simulate the complex interactions happening at the Large Hadron Collider (LHC), the quicker we can analyze the data, test our theories, and uncover the secrets of the universe. Precision and speed are like two entangled particles – you can't have one without the other! By harnessing the power of modern machine learning and first-principles physics, we can accelerate the simulation chain, paving the way for groundbreaking discoveries at the LHC Run 3, the HL-LHC, and beyond. So, to put it simply, in the fast-paced realm of particle physics, a speedy simulation chain is the key to unlocking the mysteries of the cosmos in record time!

chatjesset.com

Computing Budget

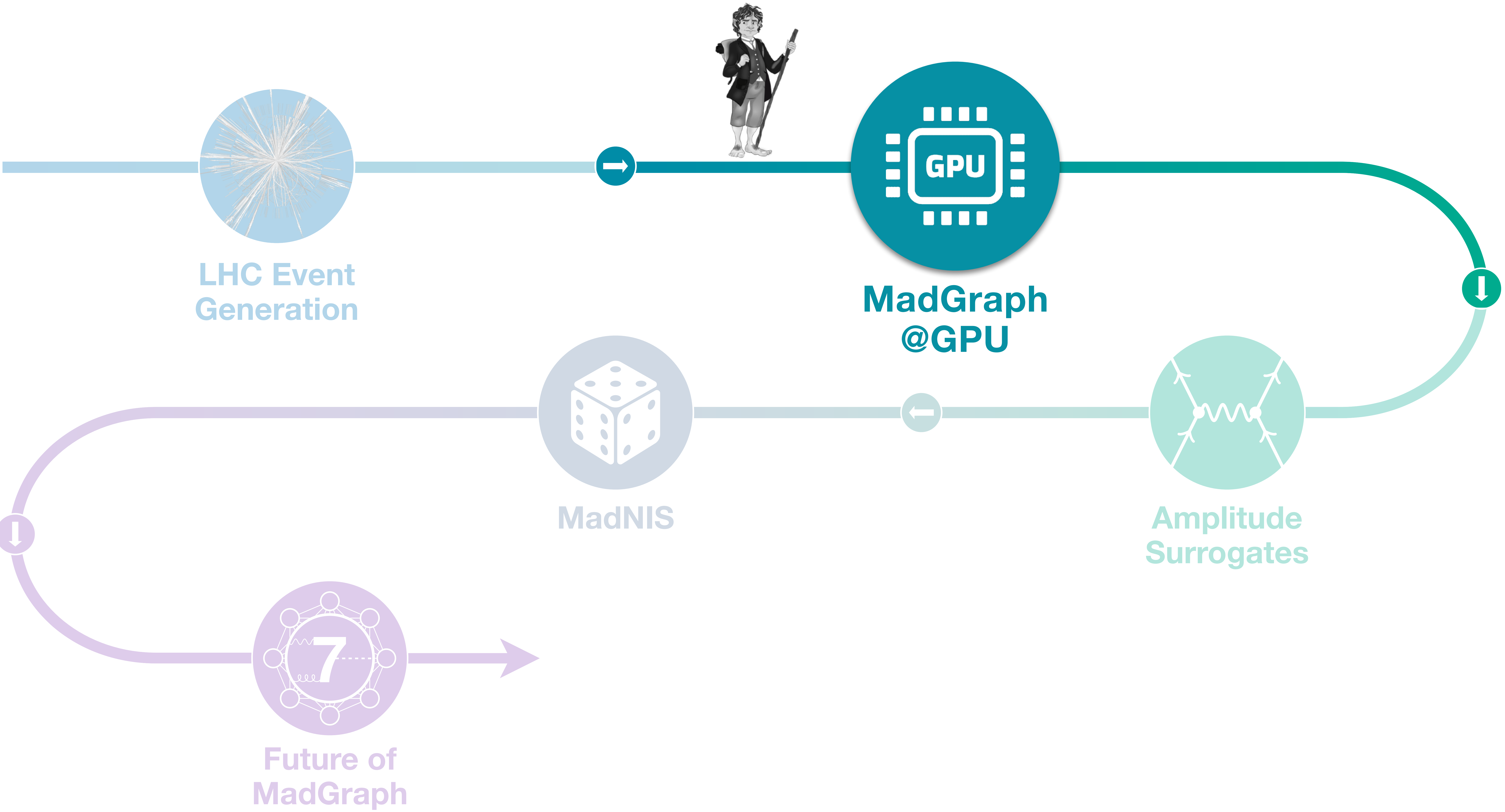


[CERN-LHCC-2022-005]



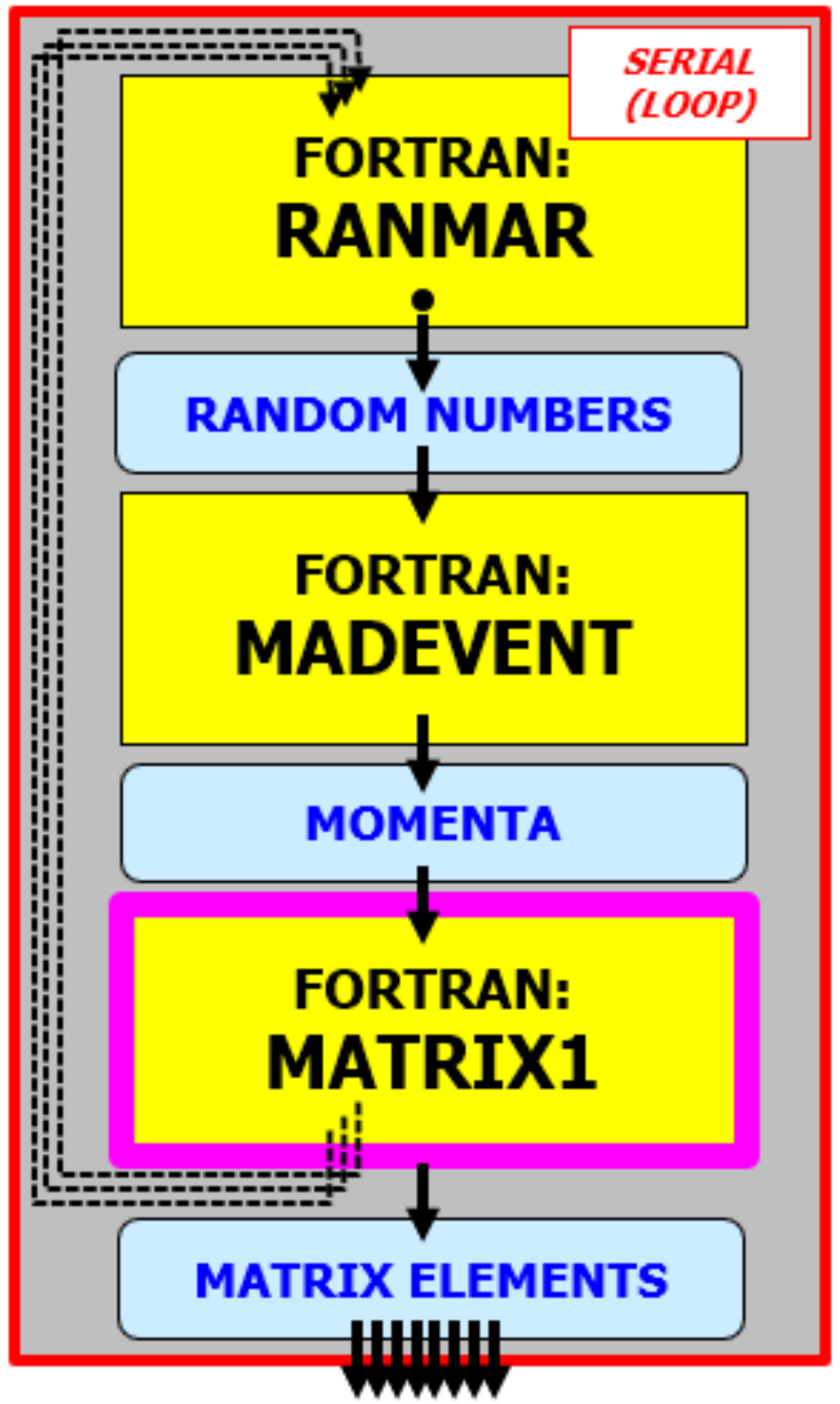
[Höche *et al.*, 1905.05120]

MadGraph4GPU



Towards aMG5aMC at LO

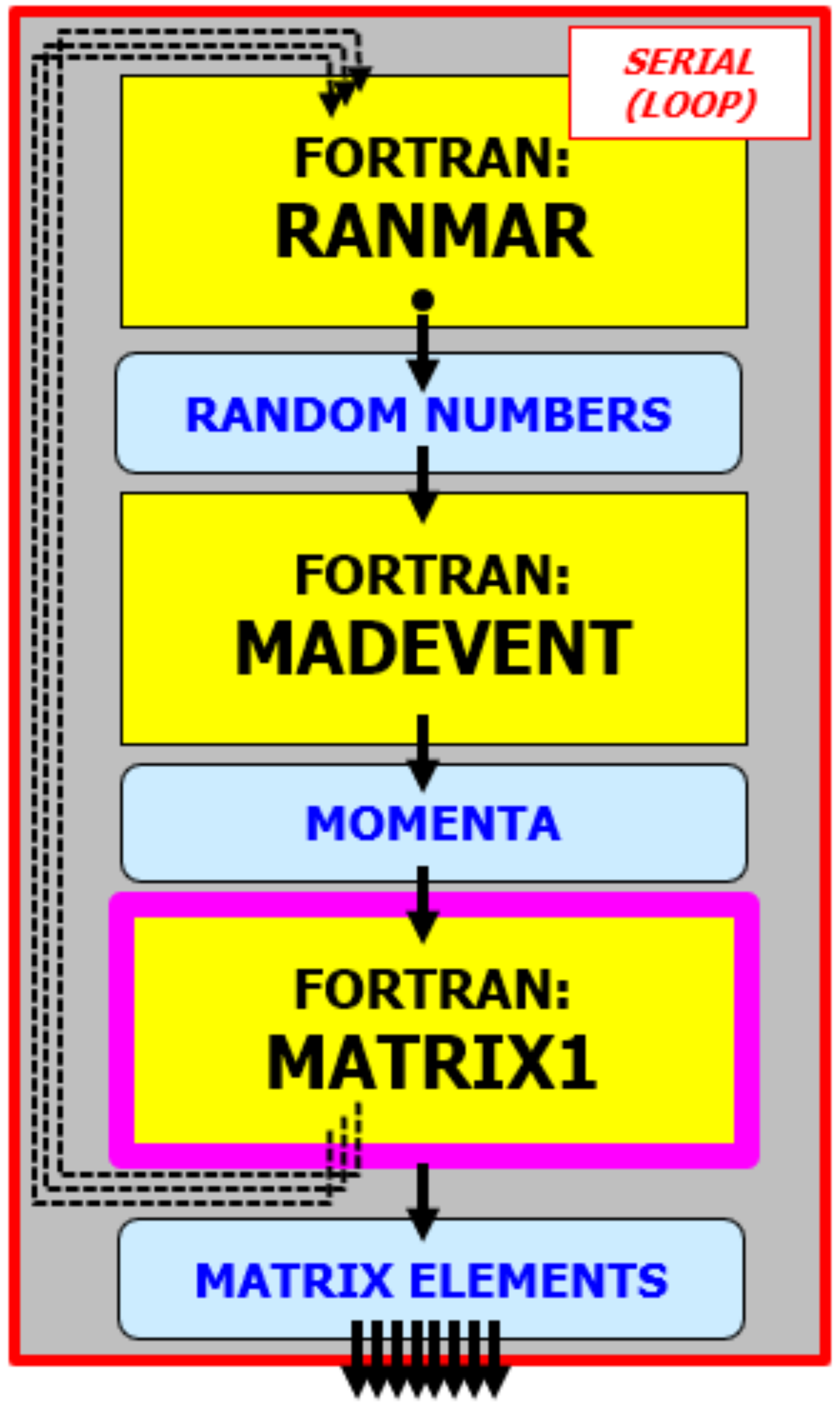
OLD madevent
Single-event MEs
(< 2020 or < MG 3.6.0)



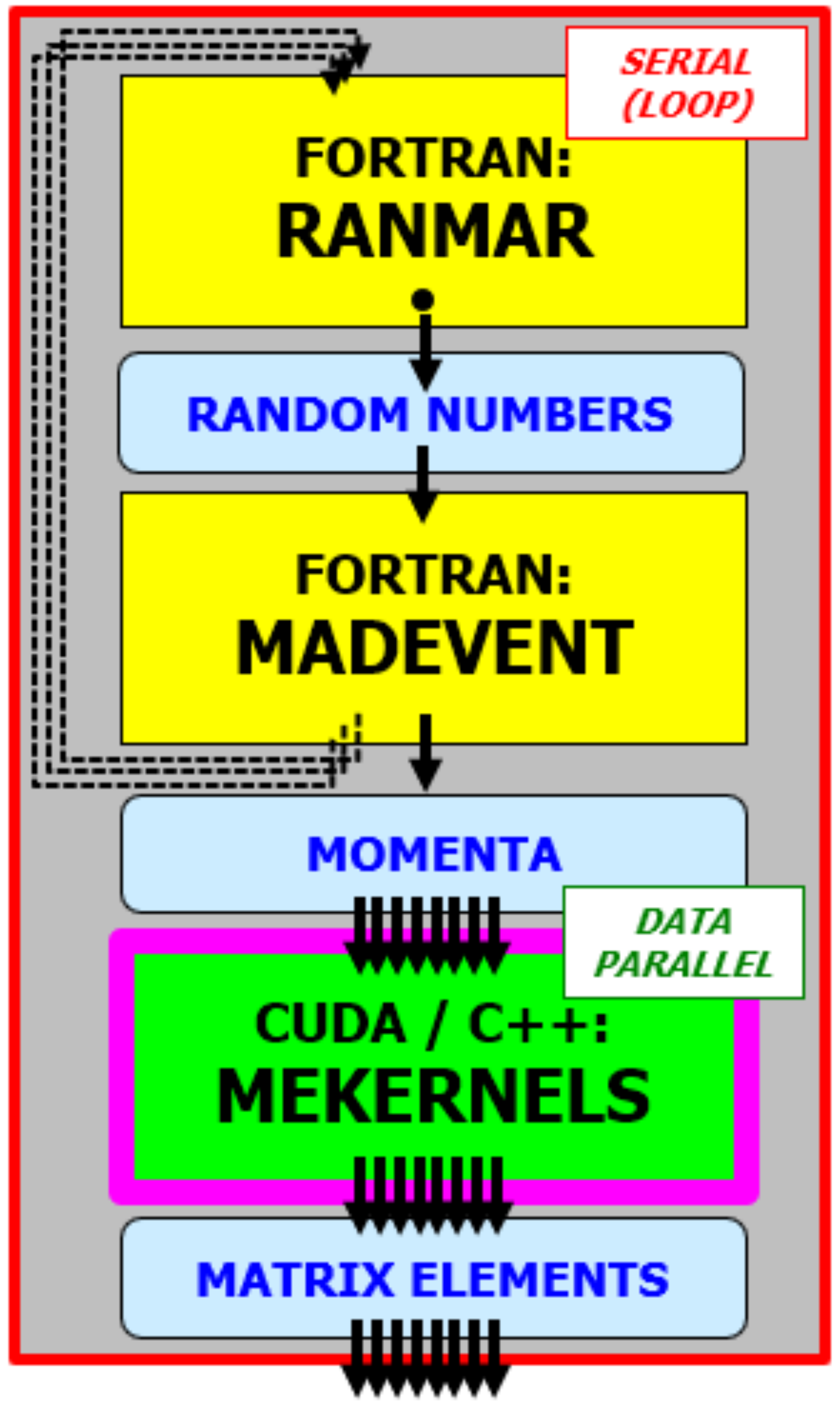
[2303.18244, 2312.02898]

Towards aMG5aMC at LO

OLD madevent
Single-event MEs
(< 2020 or < MG 3.6.0)



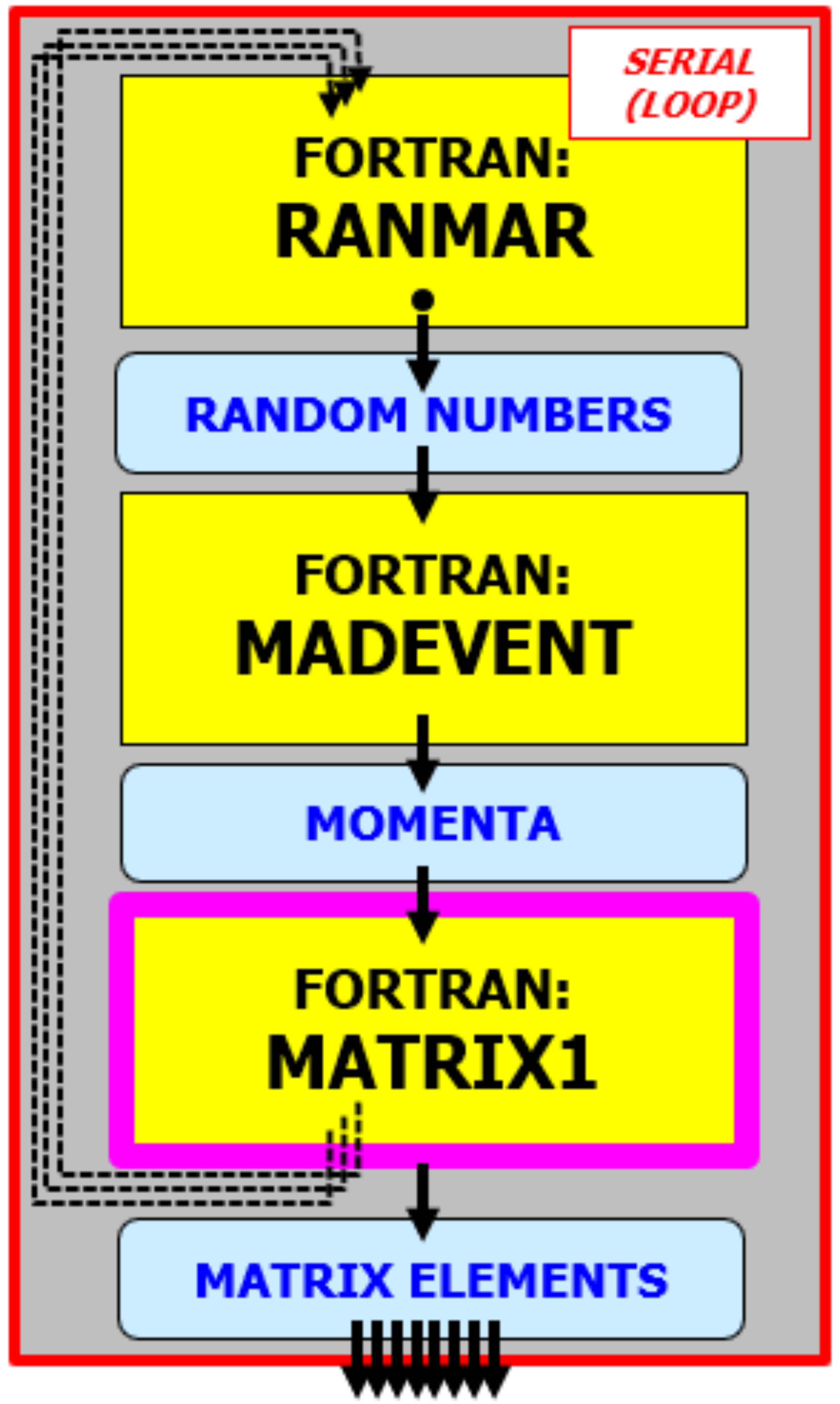
SINGLE madevent
Application
(2022)



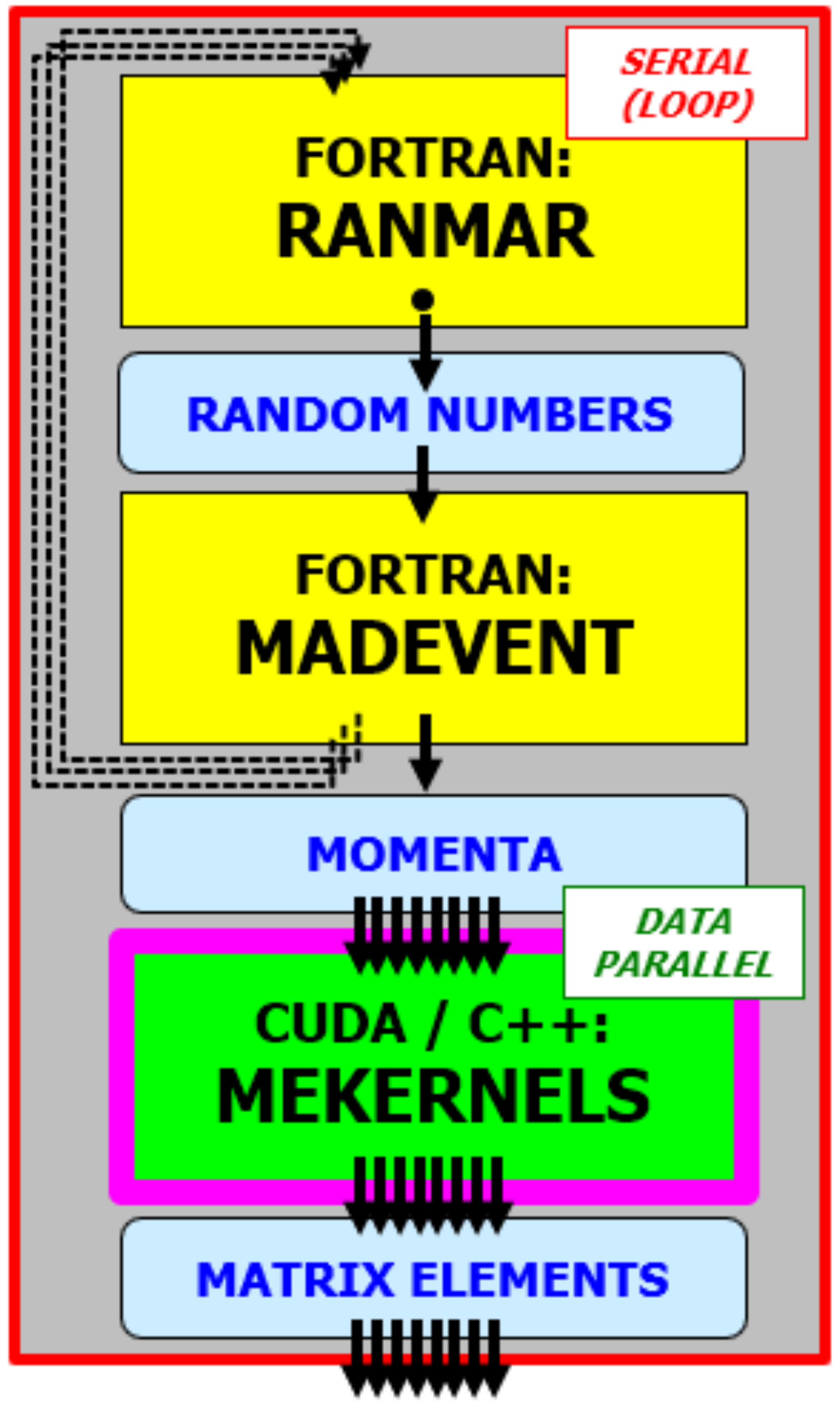
[2303.18244, 2312.02898]

Towards aMG5aMC at LO

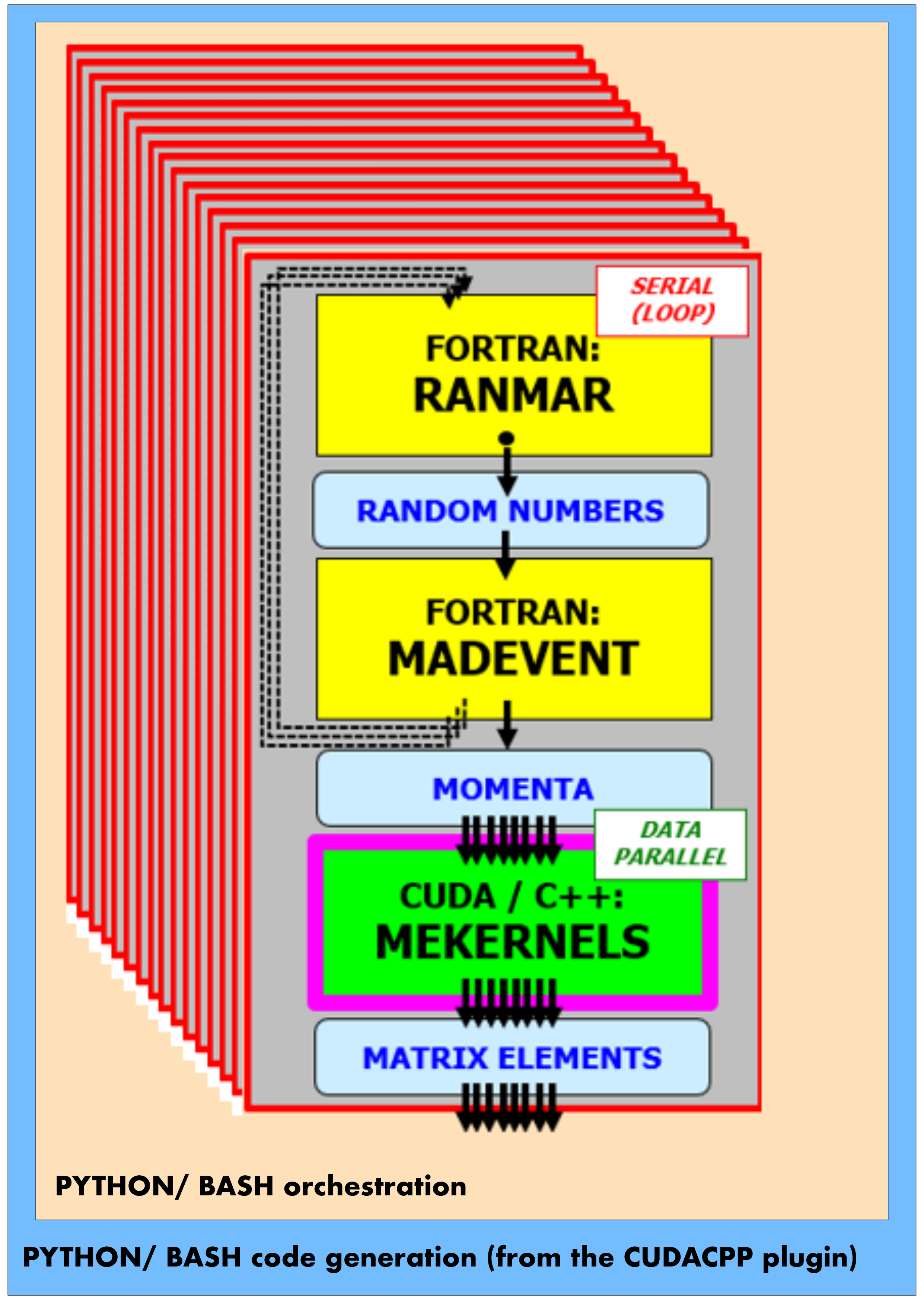
OLD madevent
Single-event MEs
(< 2020 or < MG 3.6.0)



SINGLE madevent
Application
(2022)



FULL workflow
./bin/mg5_aMC
install cudacpp;
(2024)



[2303.18244, 2312.02898]

Towards aMG5aMC at LO

OLD madevent
Single-event MEs
(< 2020 or < MG 3.6.0)

SINGLE madevent
Application
(2022)

FULL workflow
./bin/mg5_aMC
install cudacpp;
(2024)

Oct 3
github-actions
cudacpp_for...
262b845
Compare

Tarball install.
Release tag!
(Oct 2024)

cudacpp_for3.6.0_v1.00.00

Version tag cudacpp_for3.6.0_v1.00.00 (changelog)

Validated for mg5amcnlo version 3.6.0 (commit 55a291d)

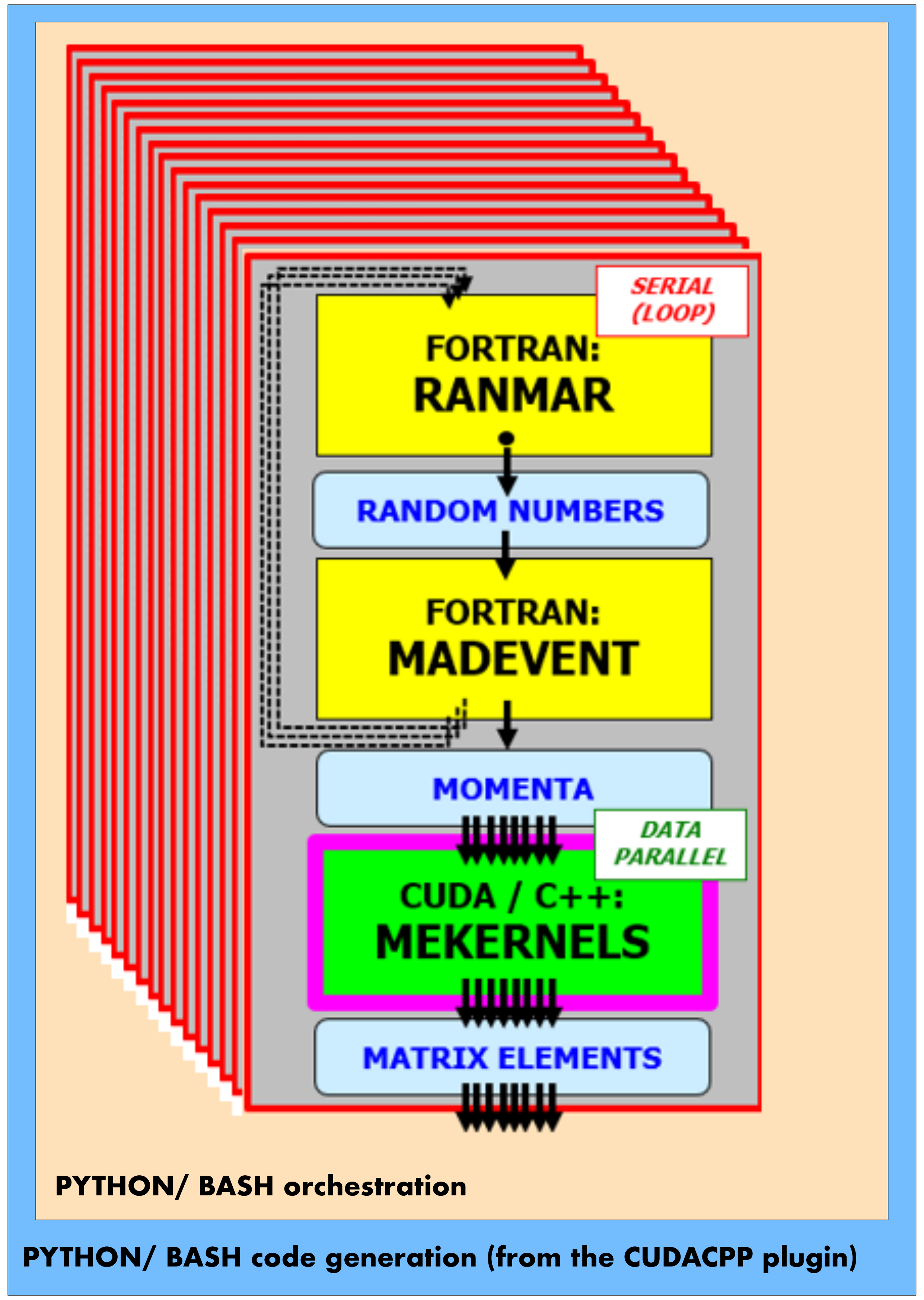
This is a release tag: you may install the *latest* release tag cudacpp_for3.6.0_latest as follows

```
MG5_aMC>install cudacpp
```

Alternatively, you may install directly this specific tag as follows

```
MG5_aMC>install cudacpp --
cudacpp_tarball=https://github.com/madgraph5/madgraph4gpu/releases/download/cudacpp_for3.6.0_v1.00.00/cudacpp.tar.gz
```

TARBALL DATE: 2024-10-03_10:10:55 UTC
commit 262b845



[2303.18244, 2312.02898]

Towards aMG5aMC at LO

OLD madevent
Single-event MEs
(< 2020 or < MG 3.6.0)

SINGLE madevent
Application
(2022)

FULL workflow
./bin/mg5_aMC
install cudacpp;
(2024)

Oct 3
github-actions
cudacpp_for...
262b845
Compare

**Tarball install.
Release tag!
(Oct 2024)**

cudacpp_for3.6.0_v1.00.00

Version tag cudacpp_for3.6.0_v1.00.00 (changelog)

Validated for mg5amcnlo version 3.6.0 (commit 55a291d)

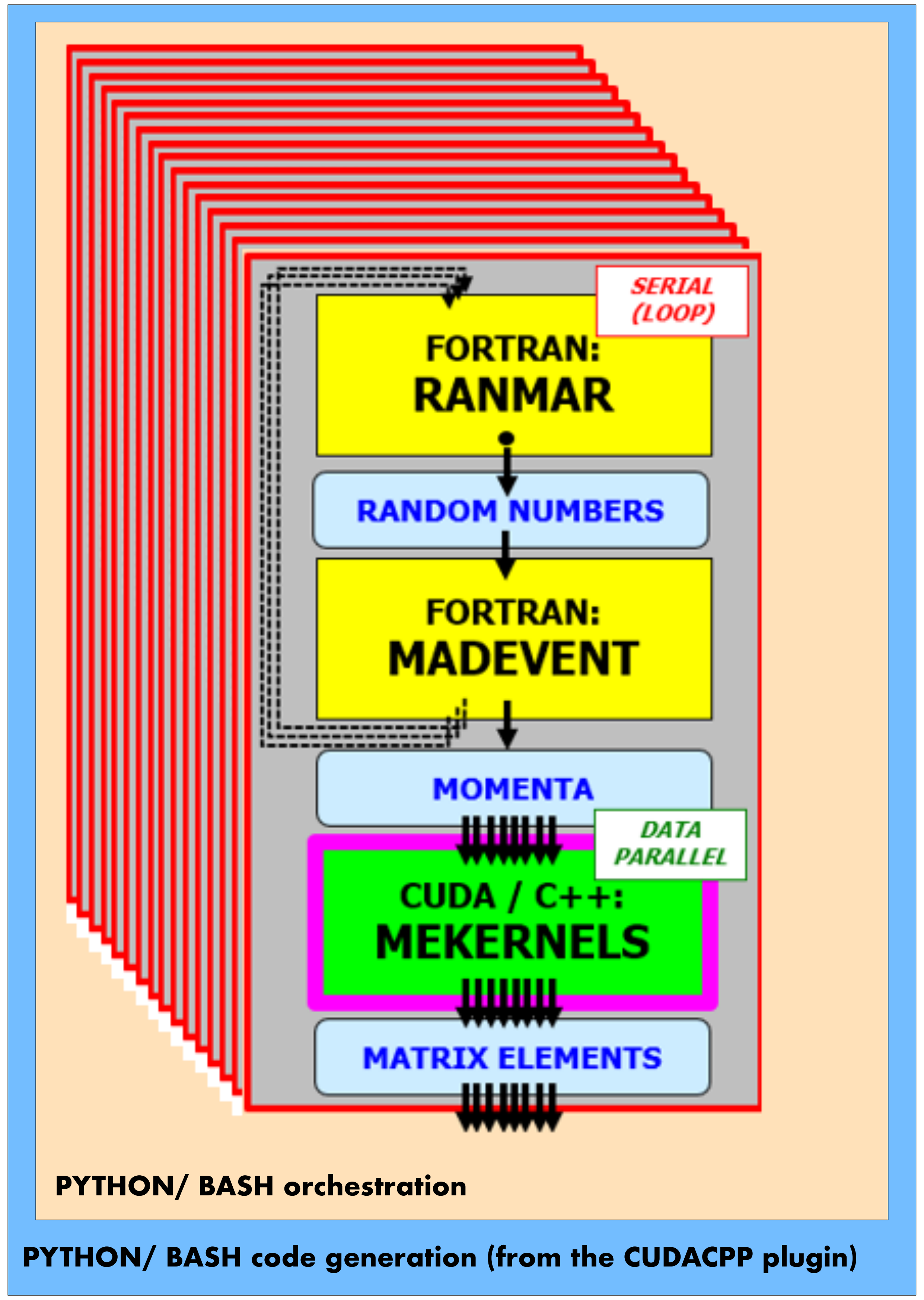
This is a release tag: you may install the *latest* release tag cudacpp_for3.6.0_latest as follows

```
MG5_aMC>install cudacpp
```

Alternatively, you may install directly this specific tag as follows

```
MG5_aMC>install cudacpp --
cudacpp_tarball=https://github.com/madgraph5/madgraph4gpu/releases/download/cudacpp_for3.6.0_v1.00.00/cudacpp.tar.gz
```

TARBALL DATE: 2024-10-03_10:10:55 UTC
commit 262b845



[2303.18244, 2312.02898]

see also Pepper [Bothmann *et al.*, 2311.06198]

What do we gain?

Process	Matrix elm	Total	Momenta+ unweight	Matrix elm
$e^+e^- \to \mu^+\mu^-$	Fortran	9.93± 0.05s	9.75± 0.05s	0.185± 0.001s
	C++ AVX2	9.93± 0.02s	9.89± 0.02s	0.045± 0.001s
		1.00± 0.01×	0.99± 0.01×	4.12 ± 0.02 ×
	Cuda Tesla A100	10.33± 0.02s	10.32± 0.02s	0.008± 0.001s
		0.96± 0.01×	0.94± 0.01×	24.3 ± 0.4 ×
$gg \to t\bar{t}gg$	Fortran	106.6 ± 0.2 s	4.55± 0.01s	102.0 ± 0.2 s
	C++ AVX2	29.01± 0.05s	4.56± 0.01s	24.45 ± 0.04 s
		3.67± 0.01×	1.00± 0.01×	4.17 ± 0.01 ×
	Cuda Tesla A100	5.78± 0.01s	4.87± 0.01s	0.91 ± 0.02 s
		18.44± 0.04×	0.93± 0.01×	112.3 ± 2.1 ×
$gg \to t\bar{t}ggg$	Fortran	2233.6 ± 1.9 s	8.81± 0.07s	2224.8 ± 1.9 s
	C++ AVX2	697.2 ± 1.2 s	8.71± 0.01s	688.5 ± 1.2 s
		3.20± 0.01×	1.01± 0.01×	3.23 ± 0.01 ×
	Cuda Tesla A100	27.78± 0.05s	9.12± 0.05s	18.66 ± 0.02 s
		80.40± 0.16×	0.97± 0.01×	119.23 ± 0.14 ×

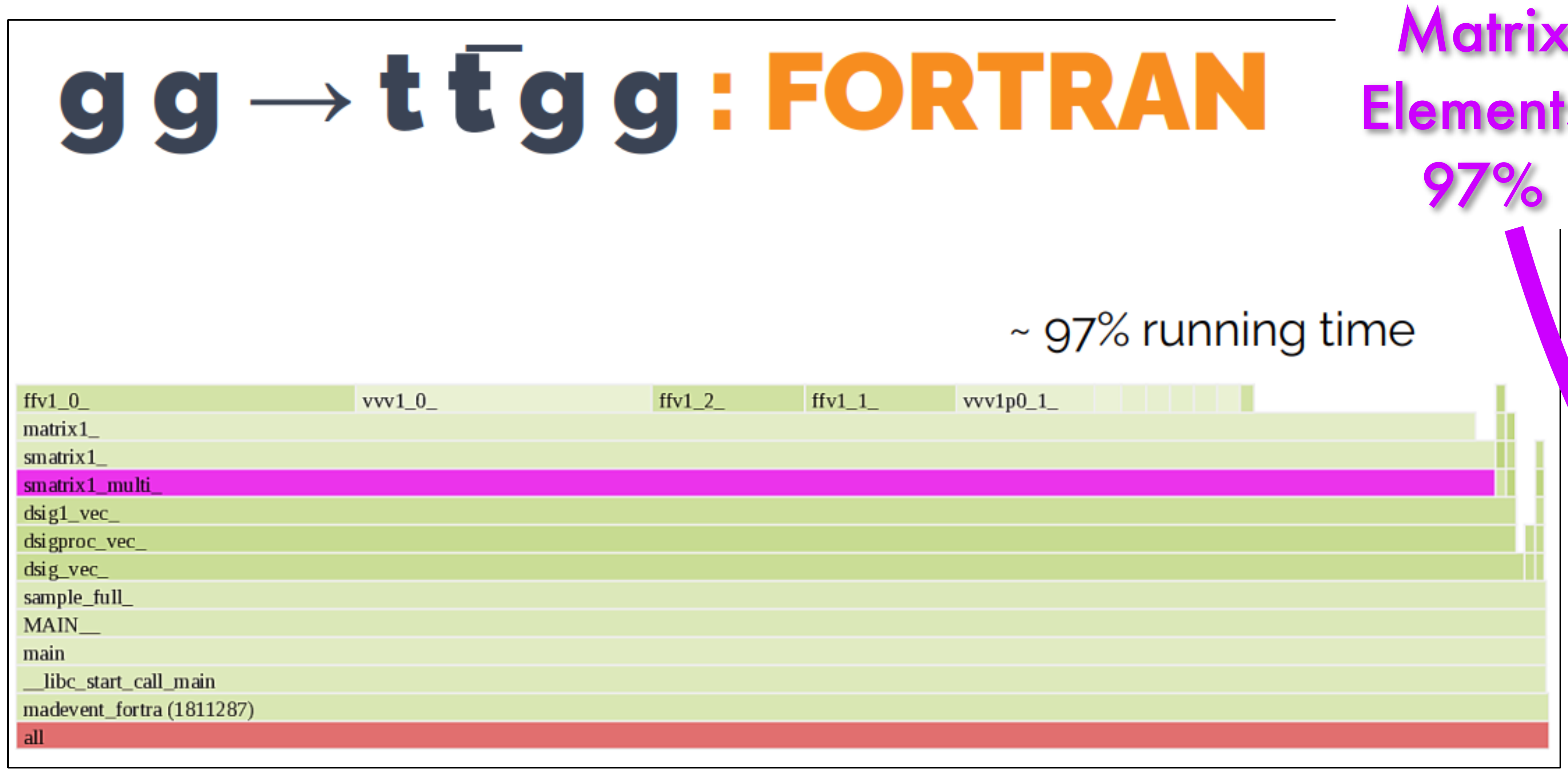
No gain for simple processes
$t_{\mathrm{Mad}} \gg t_{\mathrm{ME}}$

Medium gain for more complicatd processes
$t_{\mathrm{Mad}} < t_{\mathrm{ME}}$

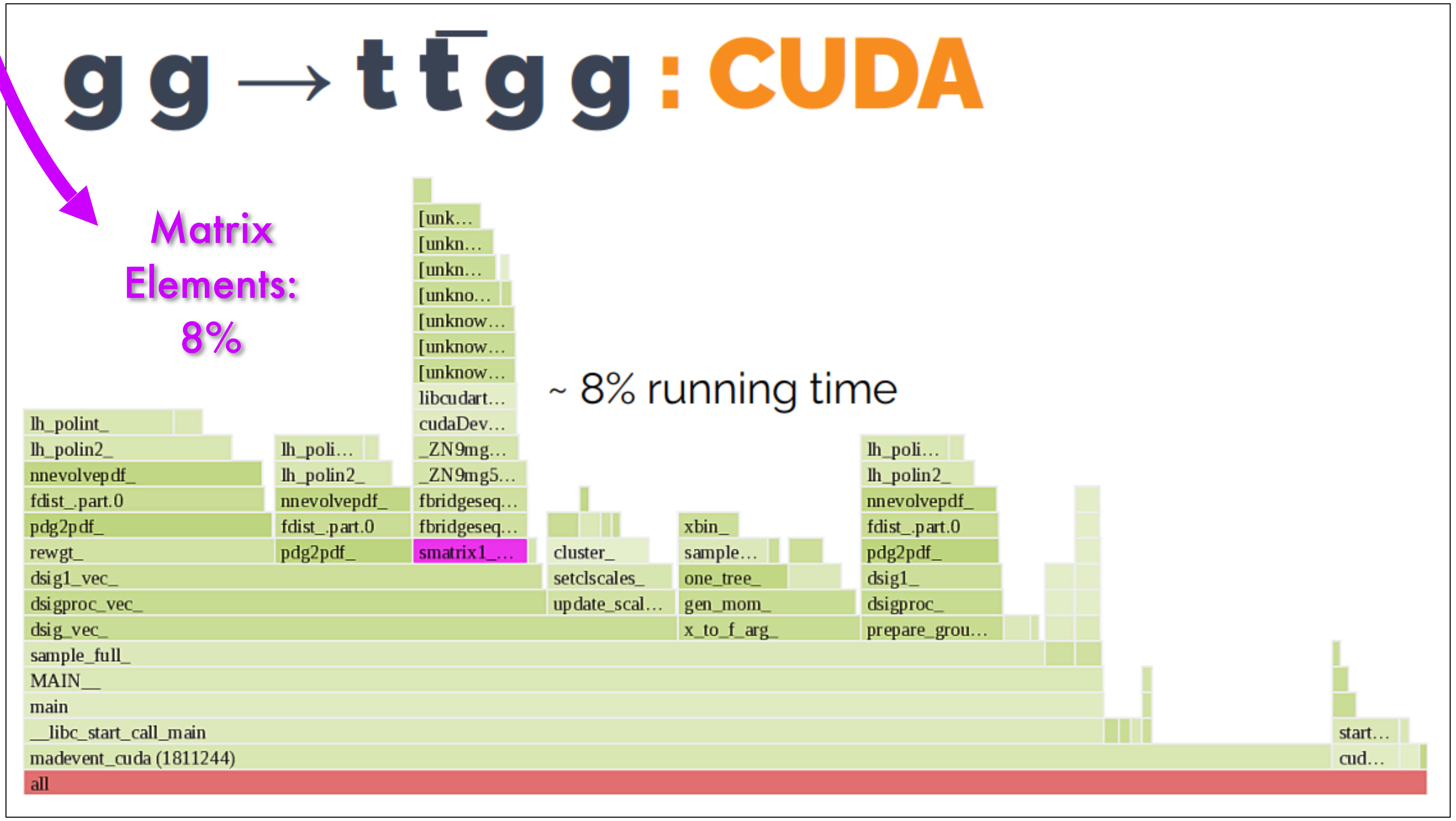
High gain for very complicatd processes
$t_{\mathrm{Mad}} \ll t_{\mathrm{ME}}$

[2303.18244, 2312.02898]

Understanding the bottleneck



Matrix Elements: 97%

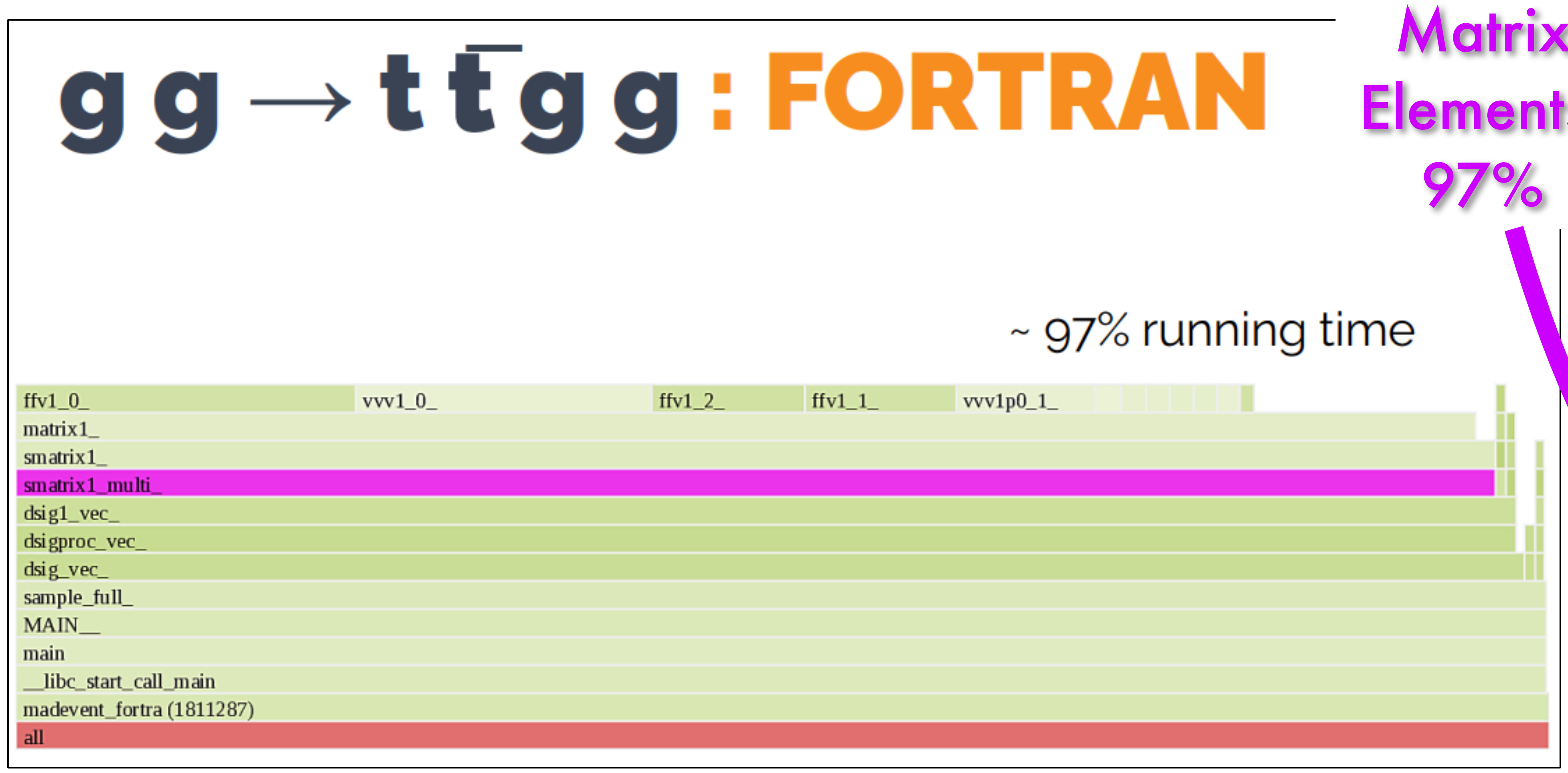


- With **Fortran**
  → **MEs are the bottleneck**

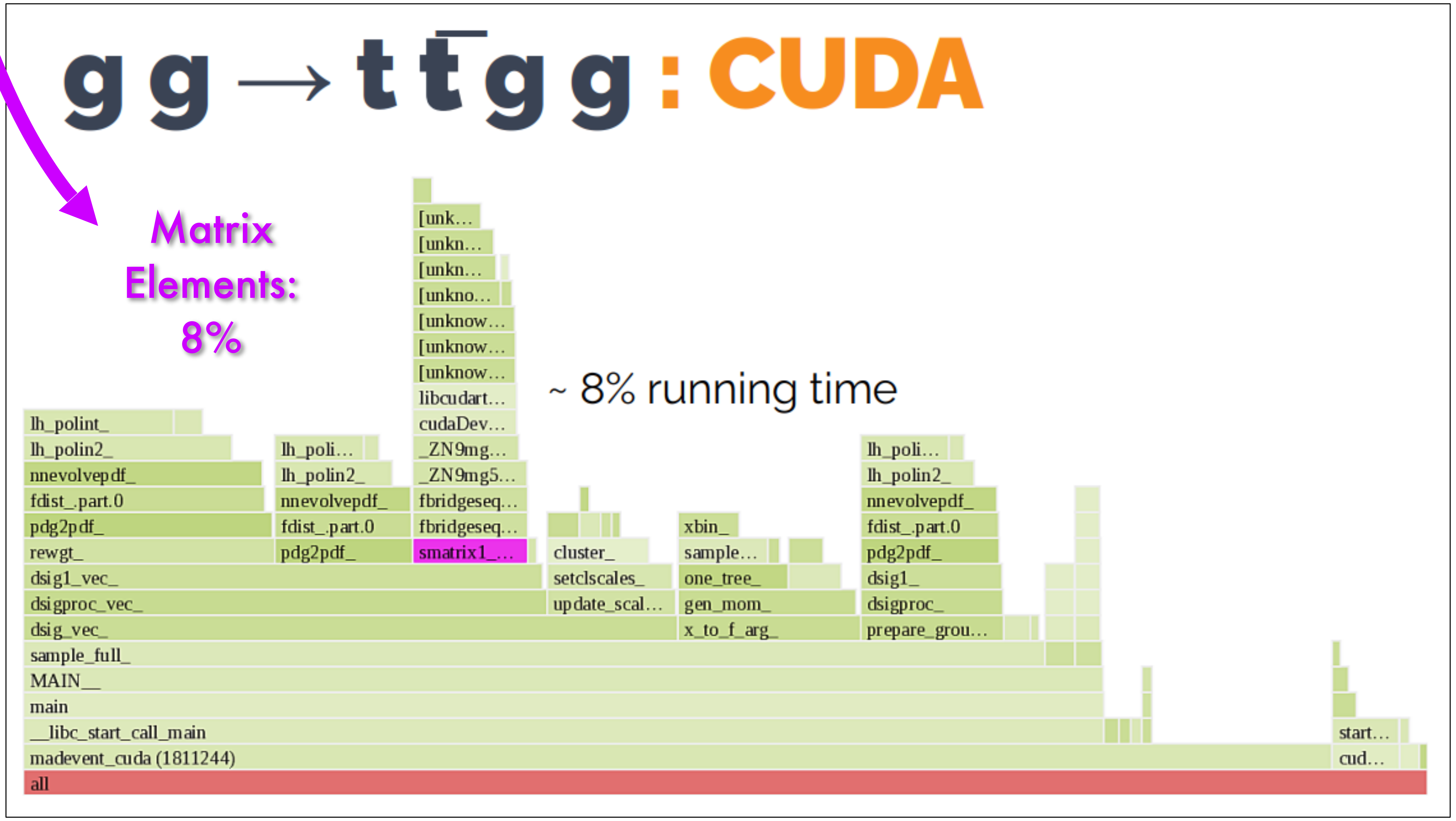
- With **GPUs/SIMD**
  → **MEs outpace other components**
  → New bottlenecks:
    - Phase-space sampling, PDFs, …

[2303.18244, 2312.02898]

Understanding the bottleneck



Matrix Elements: 97%



Matrix Elements: 8%

- With **Fortran**
  → **MEs are the bottleneck**
- With **GPUs/SIMD**
  → **MEs outpace other components**
  → New bottlenecks:
    - Phase-space sampling, PDFs, …

→ use MadNIS and **new** MadEvent7 (see future of MG5)

[2303.18244, 2312.02898]

Beyond leading order?

$$\sigma_{\mathrm{NLO}} = \int \mathrm{d}\Phi_n \,(B + V) + \int \mathrm{d}\Phi_{n+1}\, R$$

[2312.07440]

Beyond leading order?

$$\sigma_{\mathrm{NLO}} = \int \mathrm{d}\Phi_n \left(B + V\right) + \int \mathrm{d}\Phi_{n+1}\, R$$

Tree-level amplitudes

- Recycle from LO plugin
- Can reduce total runtime by **~50-70%**

[2312.07440]

Beyond leading order?

$$\sigma_{\mathrm{NLO}} = \int \mathrm{d}\Phi_n \,(B + V) + \int \mathrm{d}\Phi_{n+1}\, R$$

Tree-level amplitudes

- Recycle from LO plugin
- Can reduce total runtime by **~50-70%**

Loop amplitudes

- MG5 calls external ME libraries → **unaddresed so far**
- Often need **quadruple precision** → bad for GPUs

[2312.07440]

Beyond leading order?

$$\sigma_{\mathrm{NLO}} = \int \mathrm{d}\Phi_n \, (B + V) + \int \mathrm{d}\Phi_{n+1} \, R$$

Tree-level amplitudes (B, R)

- Recycle from LO plugin
- Can reduce total runtime by **~50-70%**

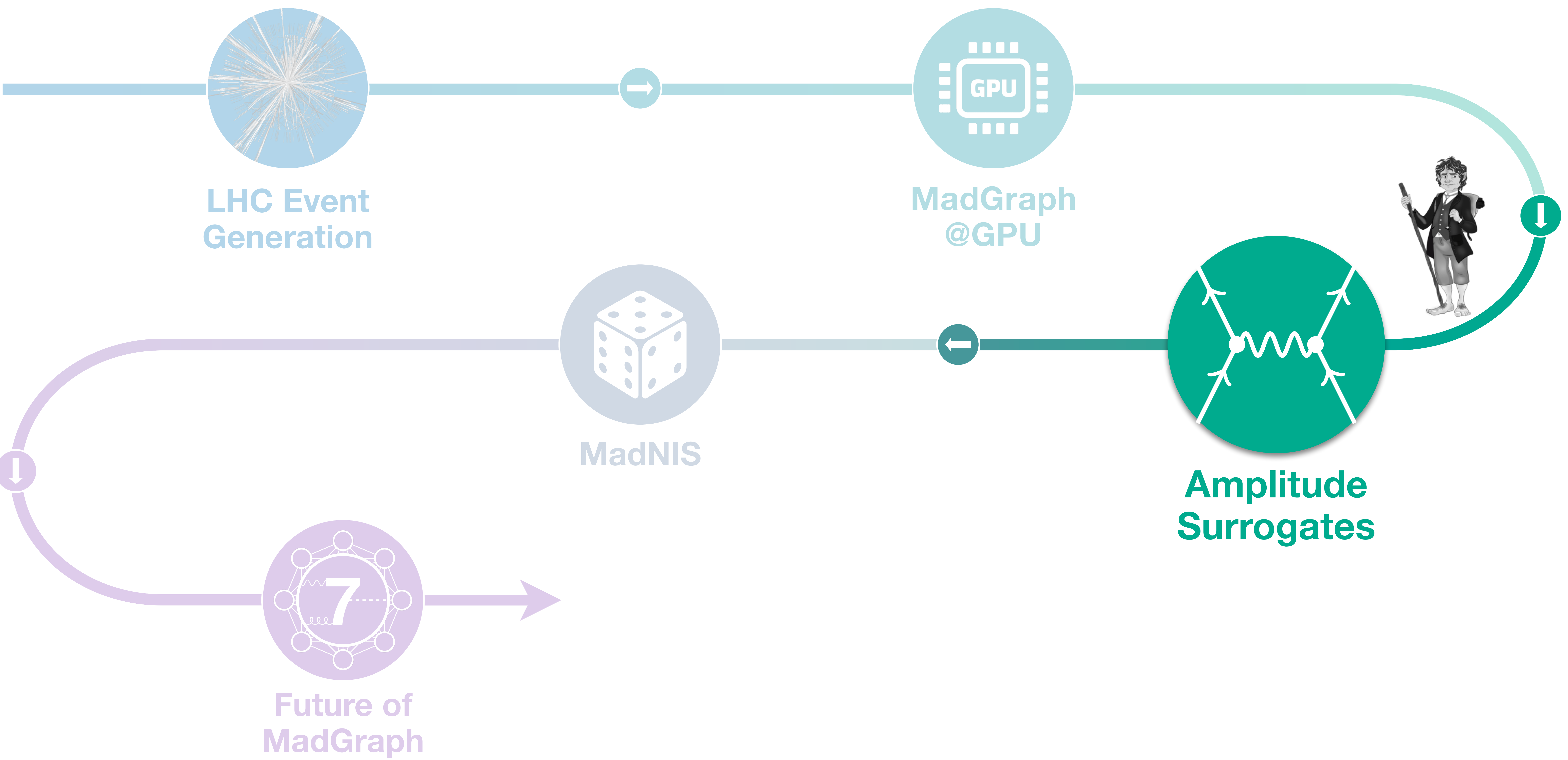
Loop amplitudes (V)

- MG5 calls external ME libraries → **unaddresed so far**
- Often need **quadruple precision** → bad for GPUs

→ Use ML to approximate loop MEs

[2312.07440]

Precision amplitude surrogates



Approximating loop amplitudes

- Instead of evaluating loop amplitude for each phase-space point, use an approximator $\tilde{V}$

$$\sigma_{\mathrm{NLO}}^{(V)} = \int \mathrm{d}\Omega \; V = \underbrace{\int \mathrm{d}\Omega \; \tilde{V}}_{1} + \underbrace{\int \mathrm{d}\Omega \; (V - \tilde{V})}_{2}$$

Approximating loop amplitudes

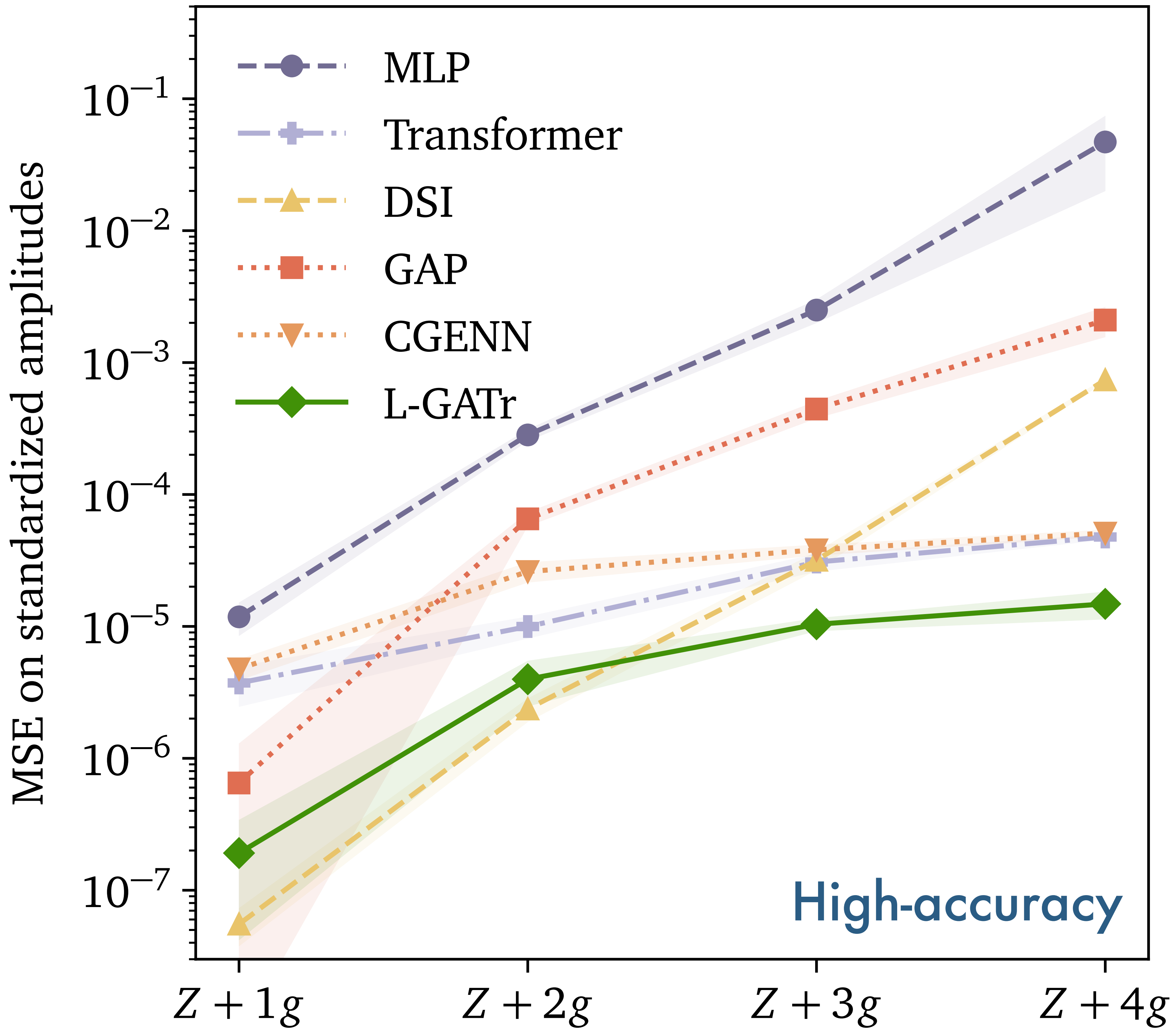
- Instead of evaluating loop amplitude for each phase-space point, use an approximator $\tilde{V}$

$$\sigma_{\mathrm{NLO}}^{(V)} = \int \mathrm{d}\Omega\ V = \underbrace{\int \mathrm{d}\Omega\ \tilde{V}}_{①} + \underbrace{\int \mathrm{d}\Omega\ (V - \tilde{V})}_{②}$$

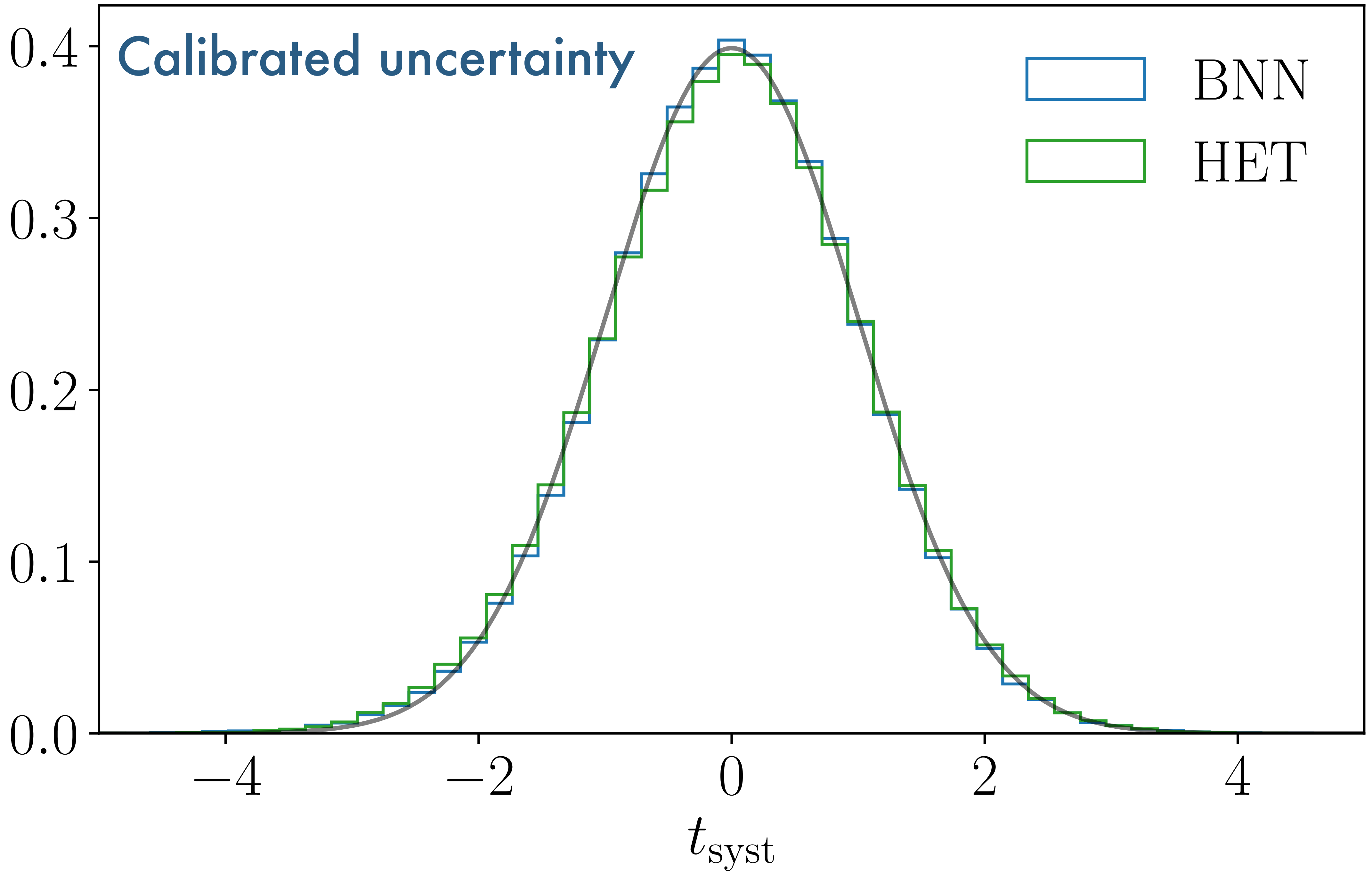
- MC over **(1)** + **(2)** → if $(V - \tilde{V}) \ll V$ only evaluate **(2)** for $n \ll N$
  → less expensive evaluations **but still** precise observable

Precise and calibrated ML amplitudes

J. Spinner et al. [2405.14806, 2411.00446]

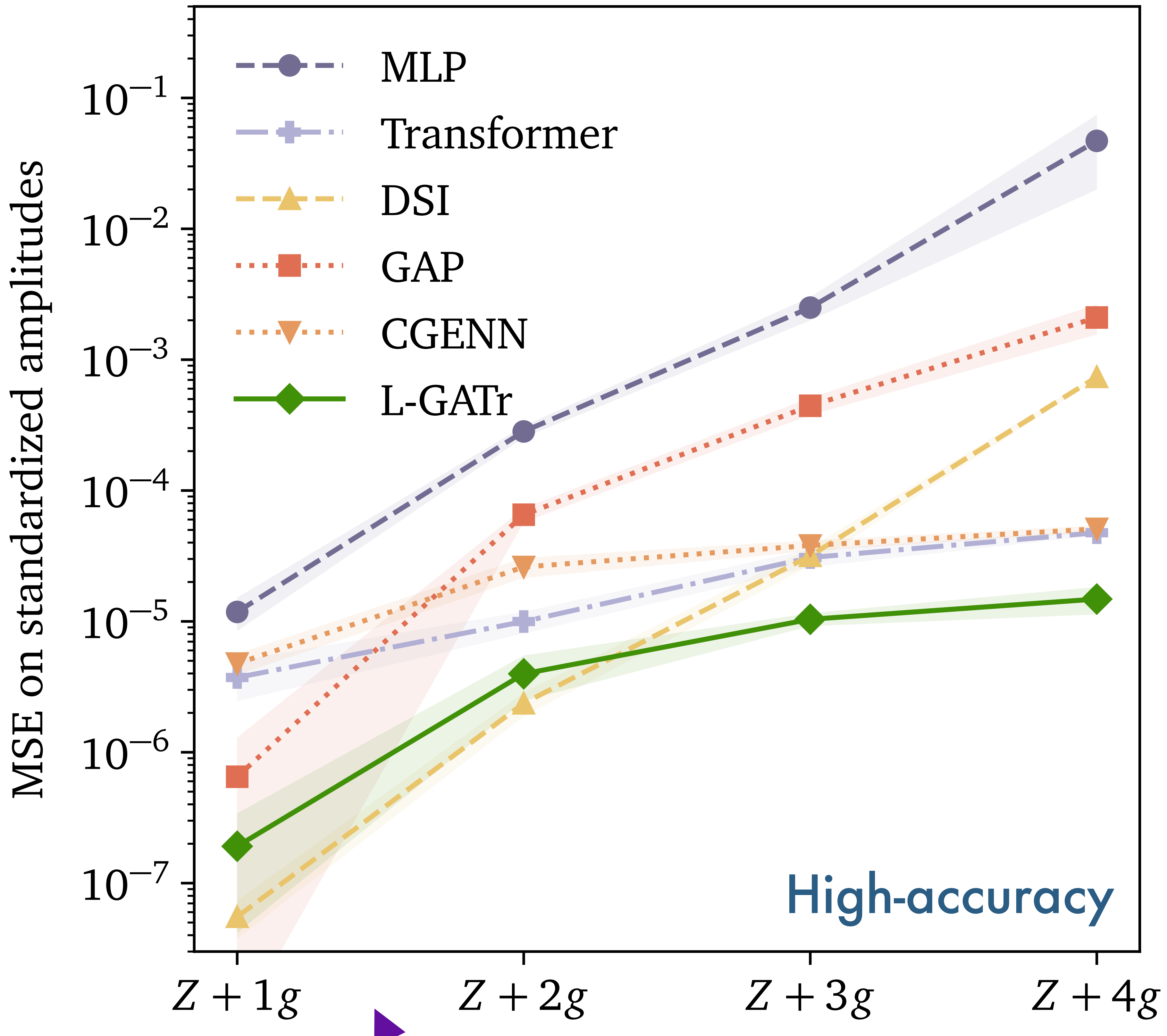


N. Elmer et al. [2412.xxxx]

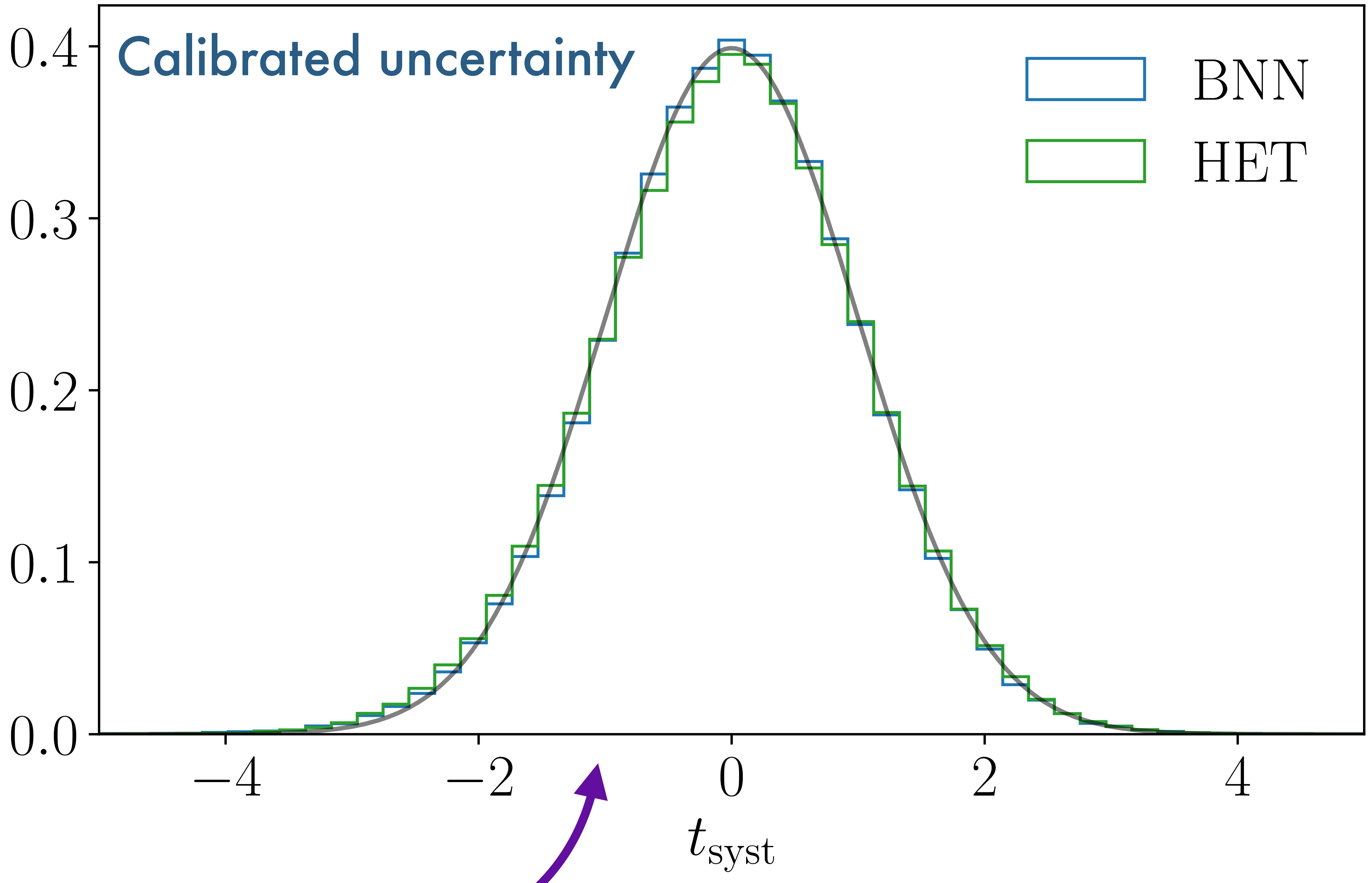


Precise and calibrated ML amplitudes

J. Spinner et al. [2405.14806, 2411.00446]



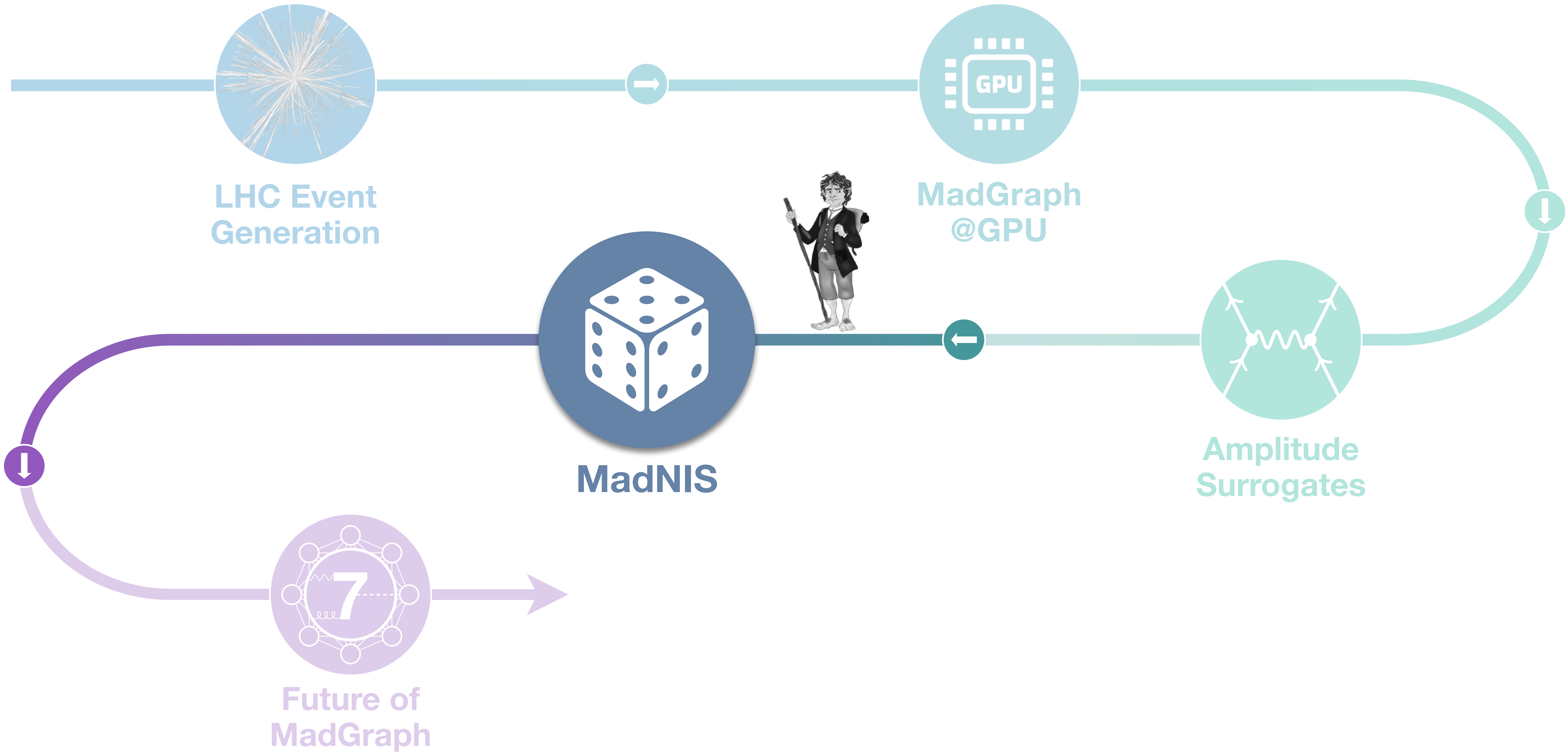
N. Elmer et al. [2412.xxxx]



See Jonas' talk
(Tuesday morning)

See Nina's talk
(Thursday morning)

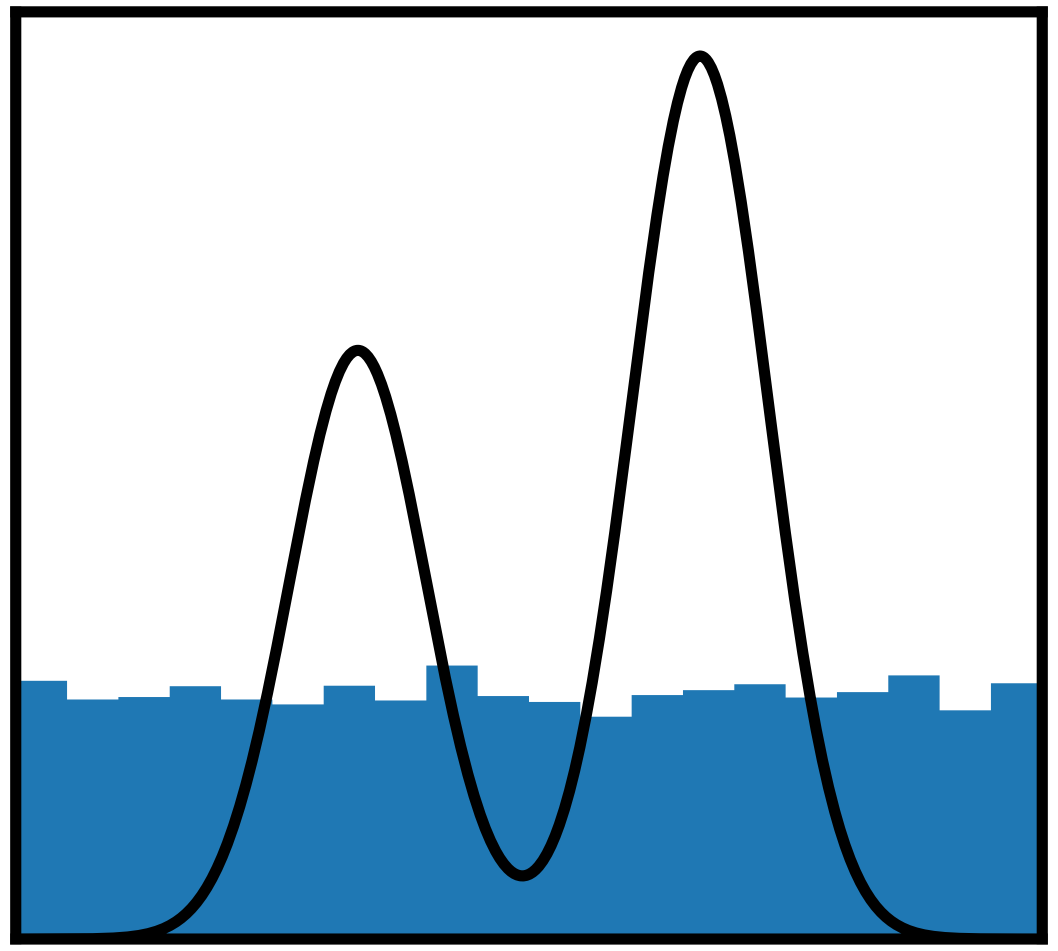
MadNIS — Neural importance sampling



Monte Carlo integration

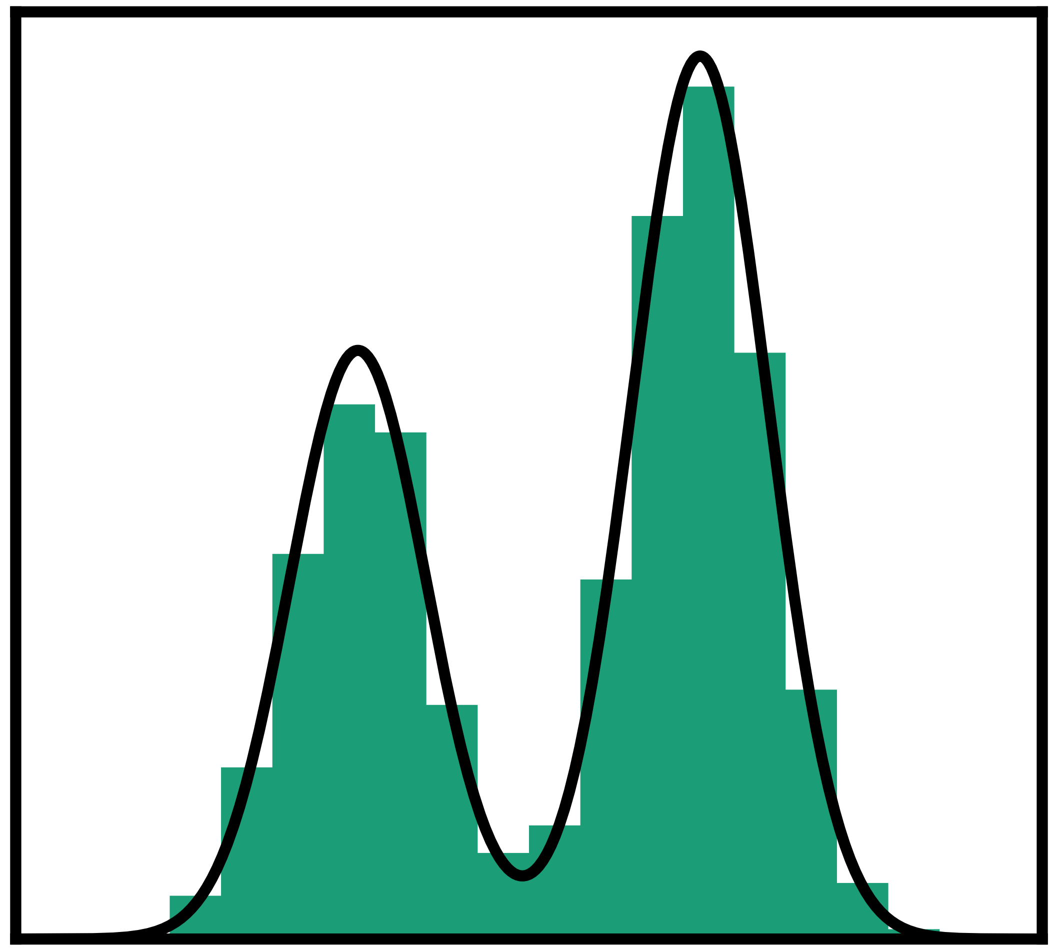
Calculate (differential) cross sections

$$d\sigma = \frac{1}{\text{flux}} dx_a dx_b\, f(x_a) f(x_b)\, d\Phi_n \left\langle\, |M_{\lambda,c,\ldots}(p_a, p_b \,|\, p_1, \ldots, p_n)|^2 \,\right\rangle$$



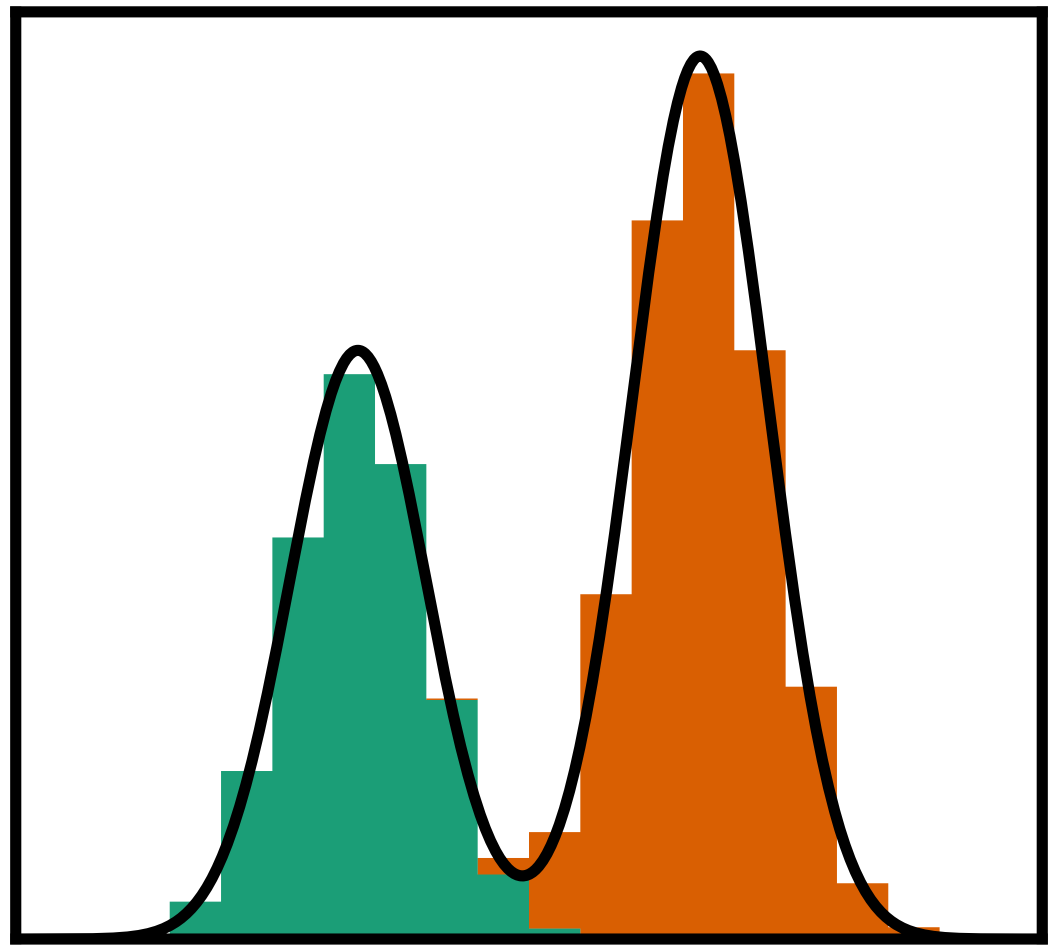
Flat sampling:
inefficient

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



Importance sampling:
find p close to f

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



Multi-channel:
one map for each channel

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

Event generation

Calculate (differential) cross sections

$$d\sigma = \frac{1}{\text{flux}} dx_a dx_b\, f(x_a) f(x_b)\, d\Phi_n \left\langle\, |M_{\lambda,c,\ldots}(p_a, p_b \,|\, p_1, \ldots, p_n)|^2 \,\right\rangle$$

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

Sum over channels

MadGraph: build channels from Feynman diagrams

Integrand

MadGraph: $d\sigma/dx$

Channel weights

MadGraph: $\alpha_i^{\text{MG}}(x) \sim |M_i|^2$

Channel mappings

MadGraph: use amplitude structure, …

Analytic mappings + refine with **VEGAS** (factorized, histogram based importance sampling)

Event generation + MadNIS

Calculate (differential) cross sections

$$\mathrm{d}\sigma = \frac{1}{\text{flux}} \mathrm{d}x_a \mathrm{d}x_b\, f(x_a) f(x_b)\, \mathrm{d}\Phi_n \left\langle\, |M_{\lambda,c,\ldots}(p_a, p_b \mid p_1, \ldots, p_n)|^2 \,\right\rangle$$

Sum over channels

MadGraph: build channels from Feynman diagrams

Integrand

MadGraph: $\mathrm{d}\sigma/\mathrm{d}x$

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

Channel weights

MadGraph: $\alpha_i^{\mathrm{MG}}(x) \sim |M_i|^2$

Learned channel mappings

MadGraph: use amplitude structure, …

Analytic mappings + ~~refine with VEGAS~~

refine with NF

Flows for NIS:
[Gao *et al*, 2001.05486+2001.10028] [Bothmann *et al*, 2001.05478]
[Winterhalder *et al*, 2112.09145] [Deutschmann *et al*, 2401.09069]

Event generation + MadNIS

Calculate (differential) cross sections

$$d\sigma = \frac{1}{\text{flux}} dx_a dx_b\, f(x_a) f(x_b)\, d\Phi_n \left\langle\, |M_{\lambda,c,\ldots}(p_a, p_b \,|\, p_1, \ldots, p_n)|^2 \,\right\rangle$$

Sum over channels

MadGraph: build channels from Feynman diagrams

Integrand

MadGraph: $d\sigma/dx$

$$I = \sum_i \left\langle \alpha_i^{\xi}(x) \frac{f(x)}{p_i^{\omega}(x)} \right\rangle_{x \sim p_i^{\omega}(x)}$$

Learned channel weights

MadGraph: $\alpha_i^{\text{MG}}(x) \sim |M_i|^2$

$$\alpha_i(x) \rightarrow \alpha_i^{\xi}(x) = \alpha_i^{\text{MG}}(x) \cdot K_i^{\xi}(x)$$

parametrize with **NN**

Learned channel mappings

MadGraph: use amplitude structure, …

Analytic mappings + ~~refine with VEGAS~~

refine with **NF**

MadNIS — Overview

MadNIS

Basic functionality

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

Improved multi-channeling

- Stratified sampling/ training
- Symmetries between channels
- Channel Dropping
- Partial weight buffering

Improved training

- VEGAS Initialization
- Buffered Training
- Surrogate Integrand

LHC Examples

MadNIS [2212.06172, 2311.01548, 2408.01486]

MadNIS — Overview

Basic functionality

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

MadNIS

Improved multi-channeling

- Stratified sampling/ training
- Symmetries between channels
- Channel Dropping
- Partial weight buffering

Improved training

- VEGAS Initialization
- Buffered Training
- Surrogate Integrand

LHC Examples

VEGAS initialization

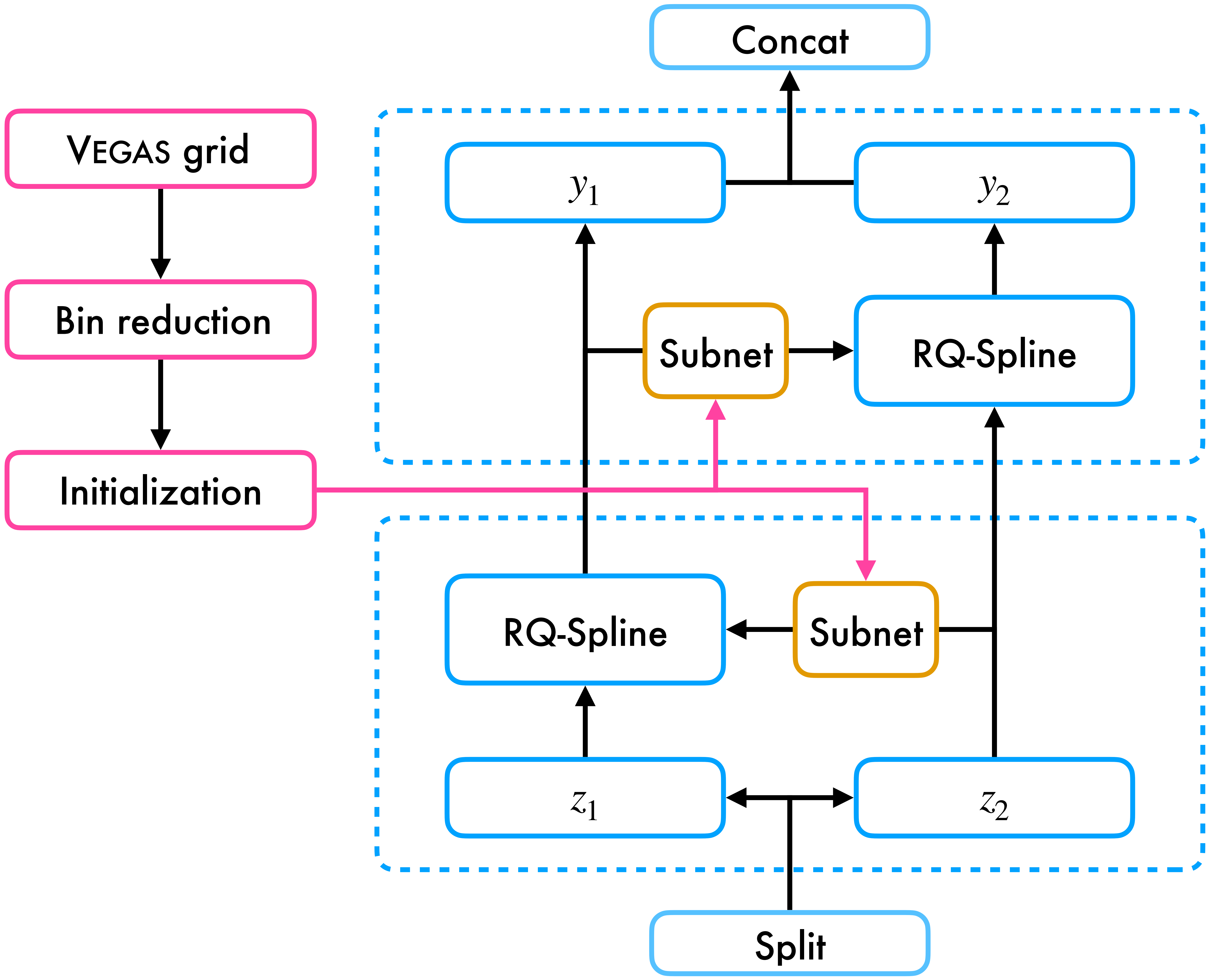
	VEGAS	Flow
Training	Fast	Slow
Correlations	No	Yes

VEGAS initialization

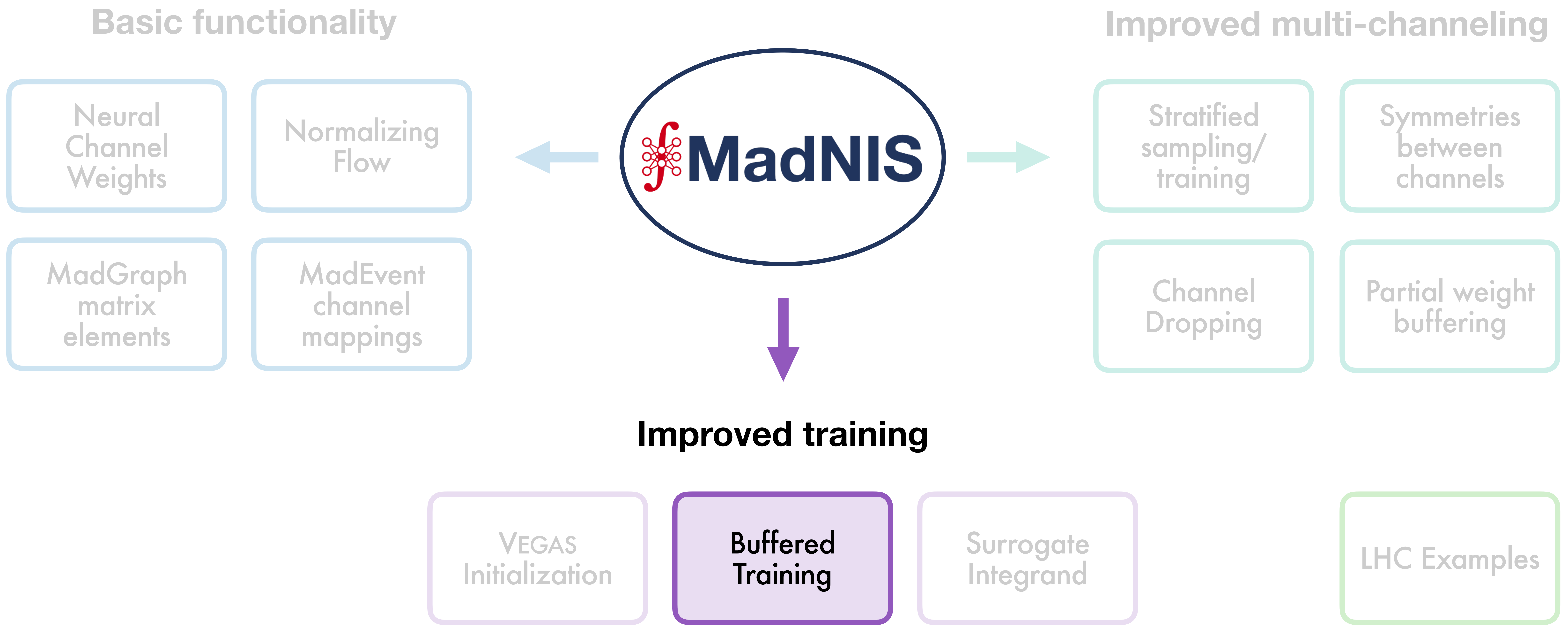
	VEGAS	Flow
Training	Fast	Slow
Correlations	No	Yes

Combine advantages:

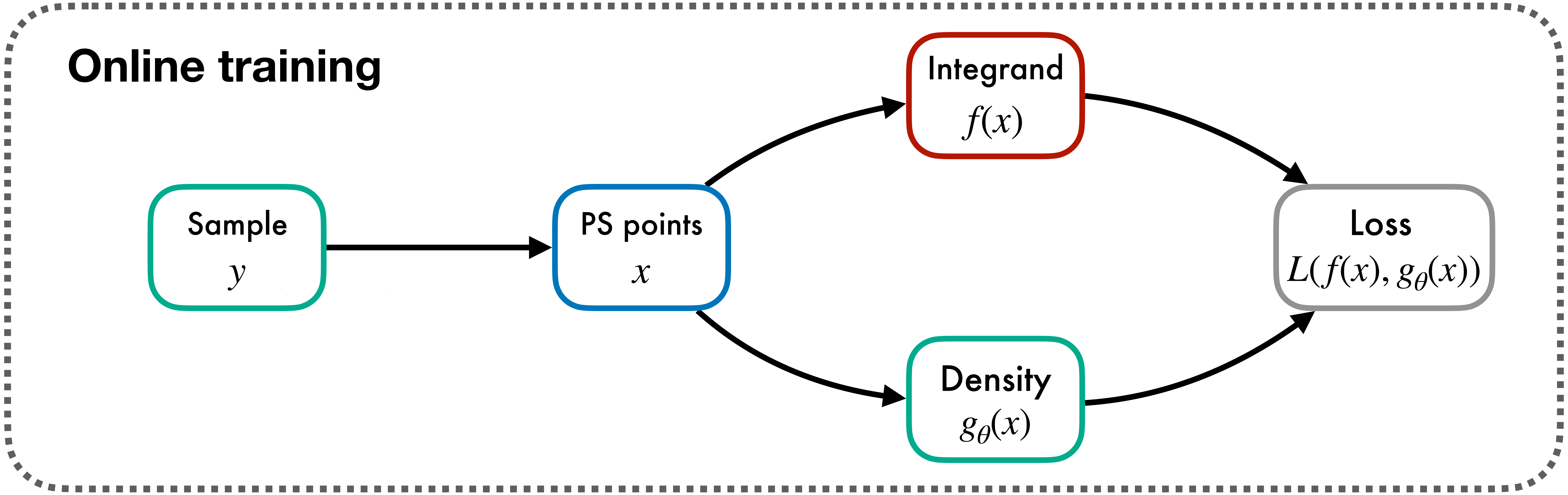
Pre-trained VEGAS grid as starting point for flow training



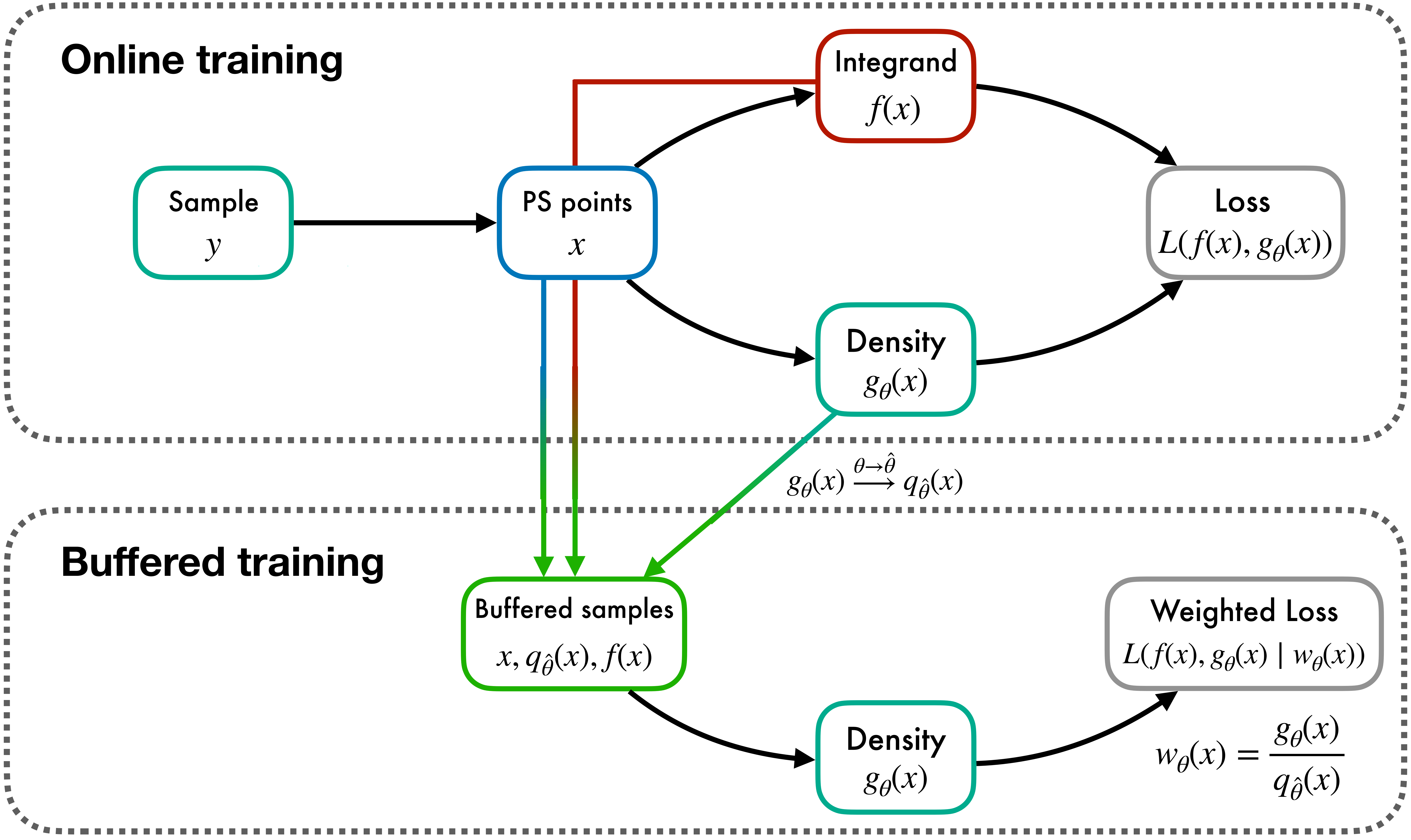
MadNIS — Overview



Buffered training



Buffered training



Buffered training

Training algorithm

generate new samples, train on them, save samples

↓

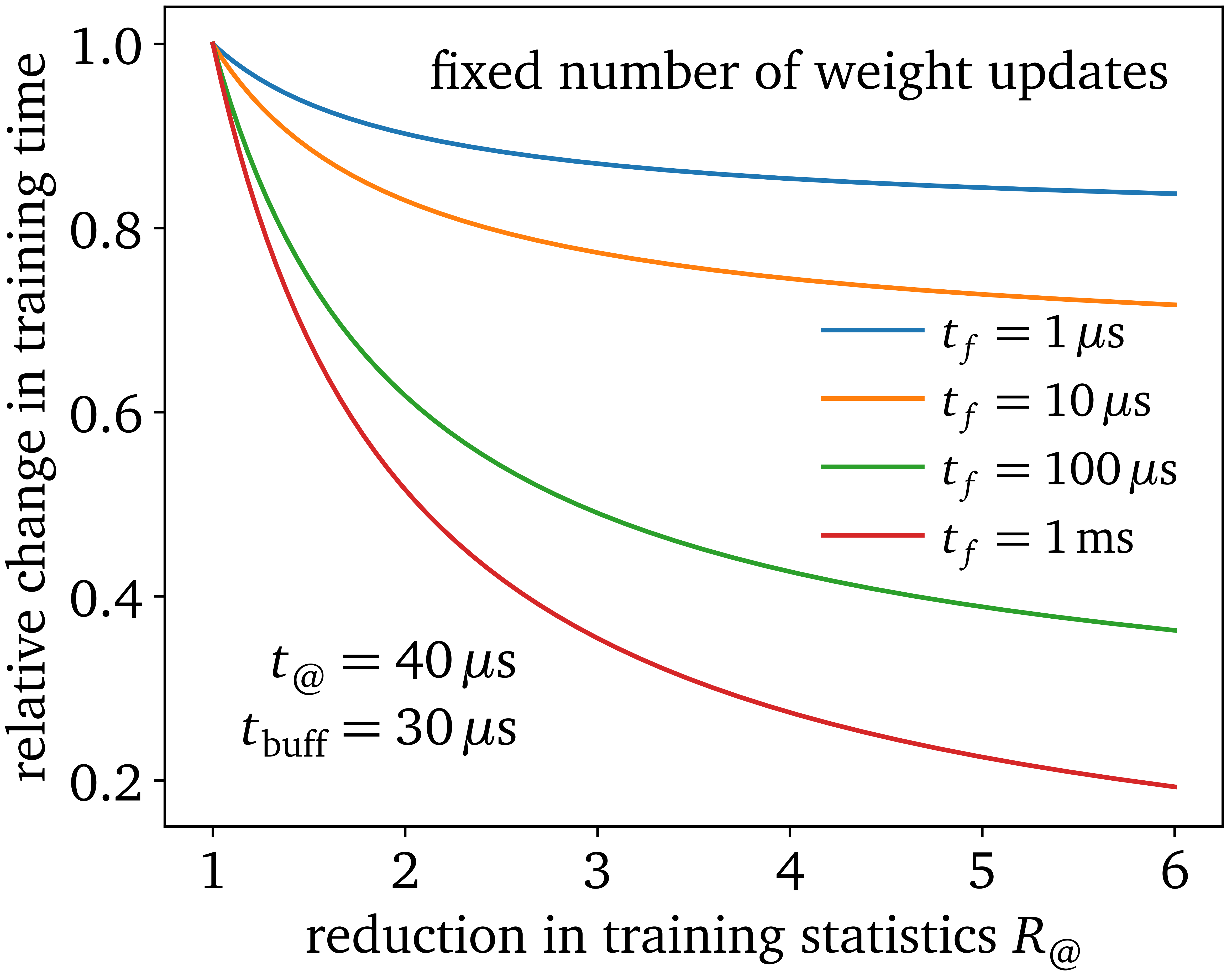
train on saved samples n times

↓

repeat

Reduction in training statistics by

$$R_@ = n + 1$$



MadNIS — Overview

Basic functionality

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

MadNIS

Improved multi-channeling

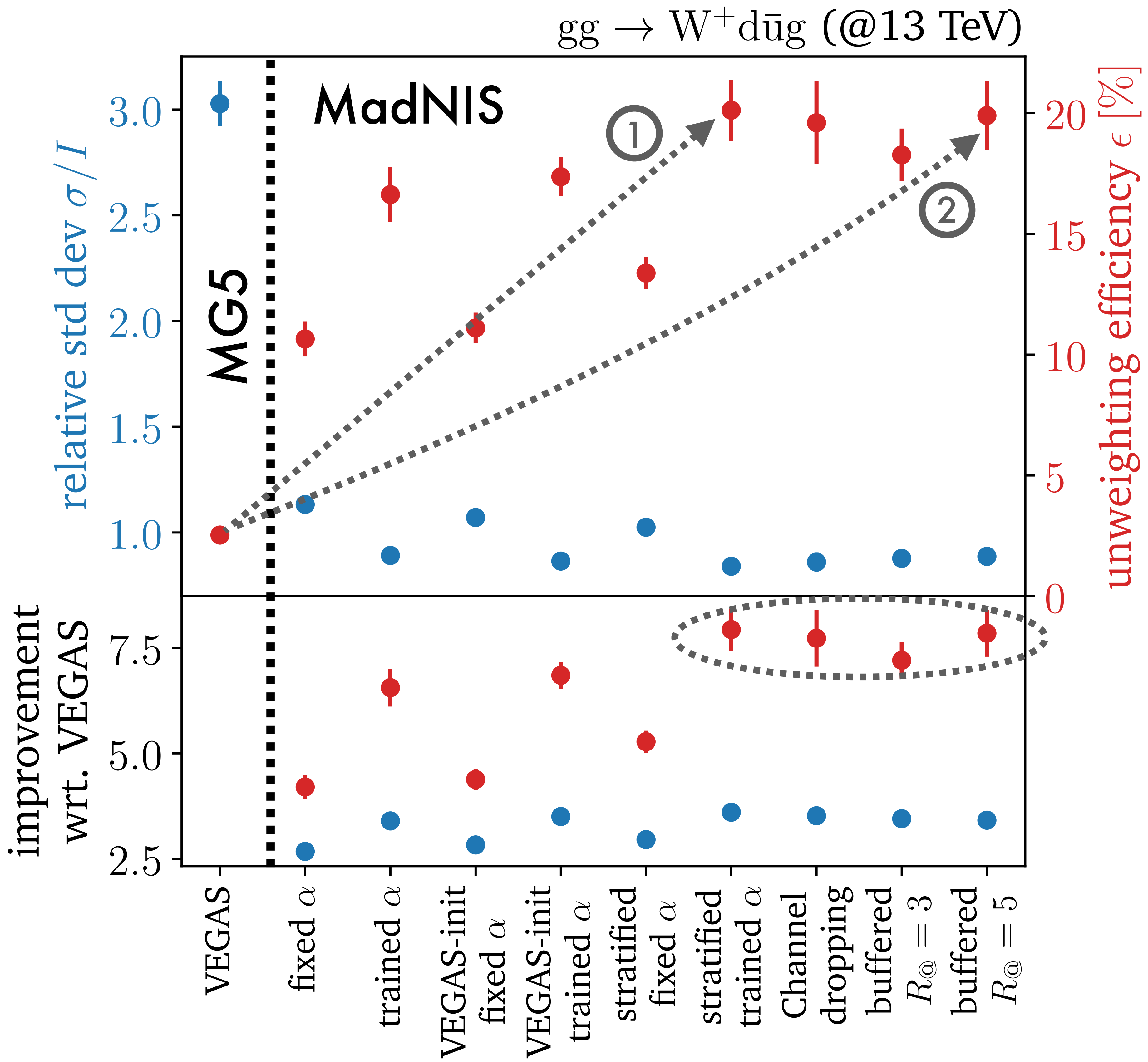
- Stratified sampling/ training
- Symmetries between channels
- Channel Dropping
- Partial weight buffering

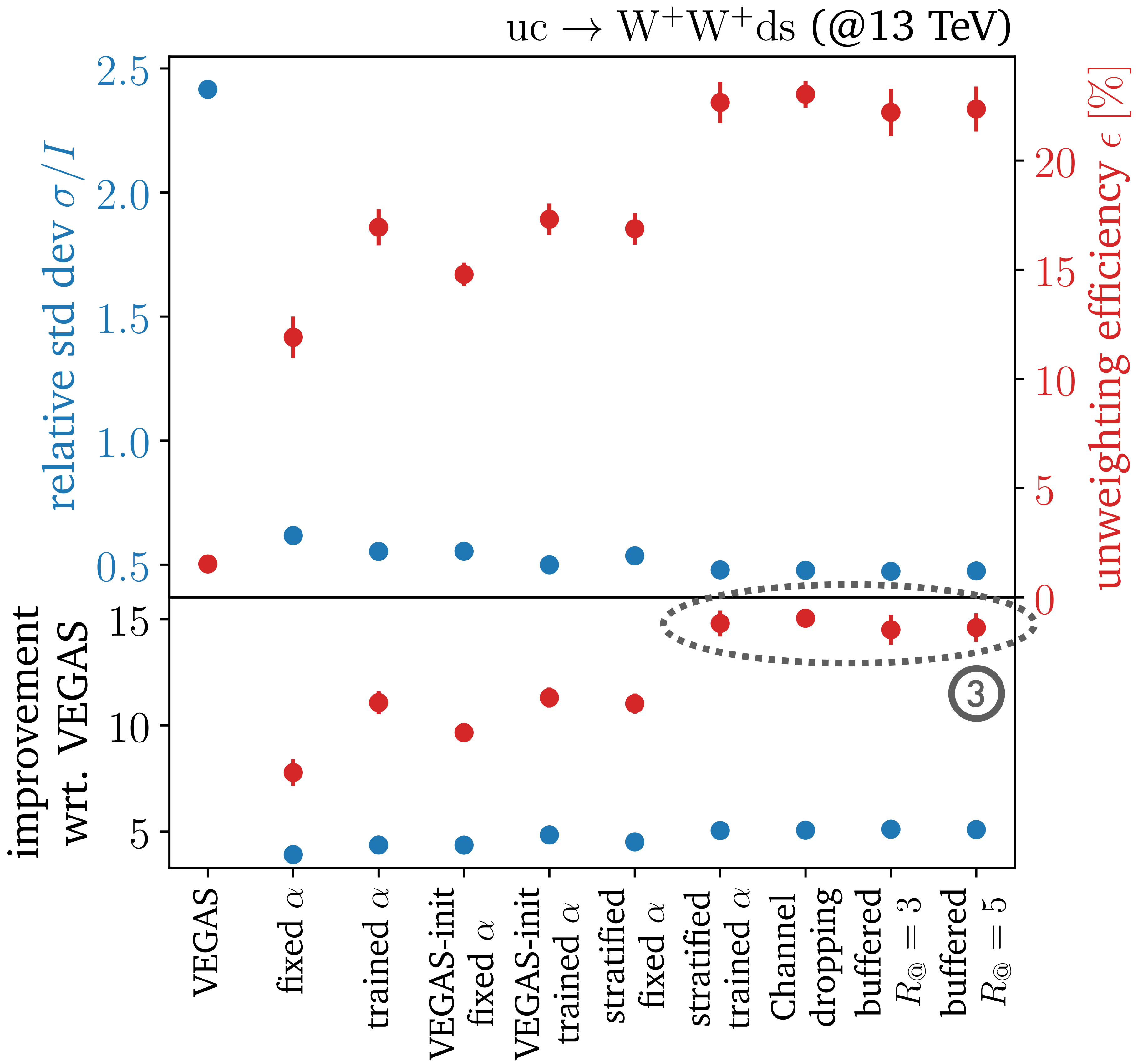
Improved training

- VEGAS Initialization
- Buffered Training
- Surrogate Integrand

LHC Examples

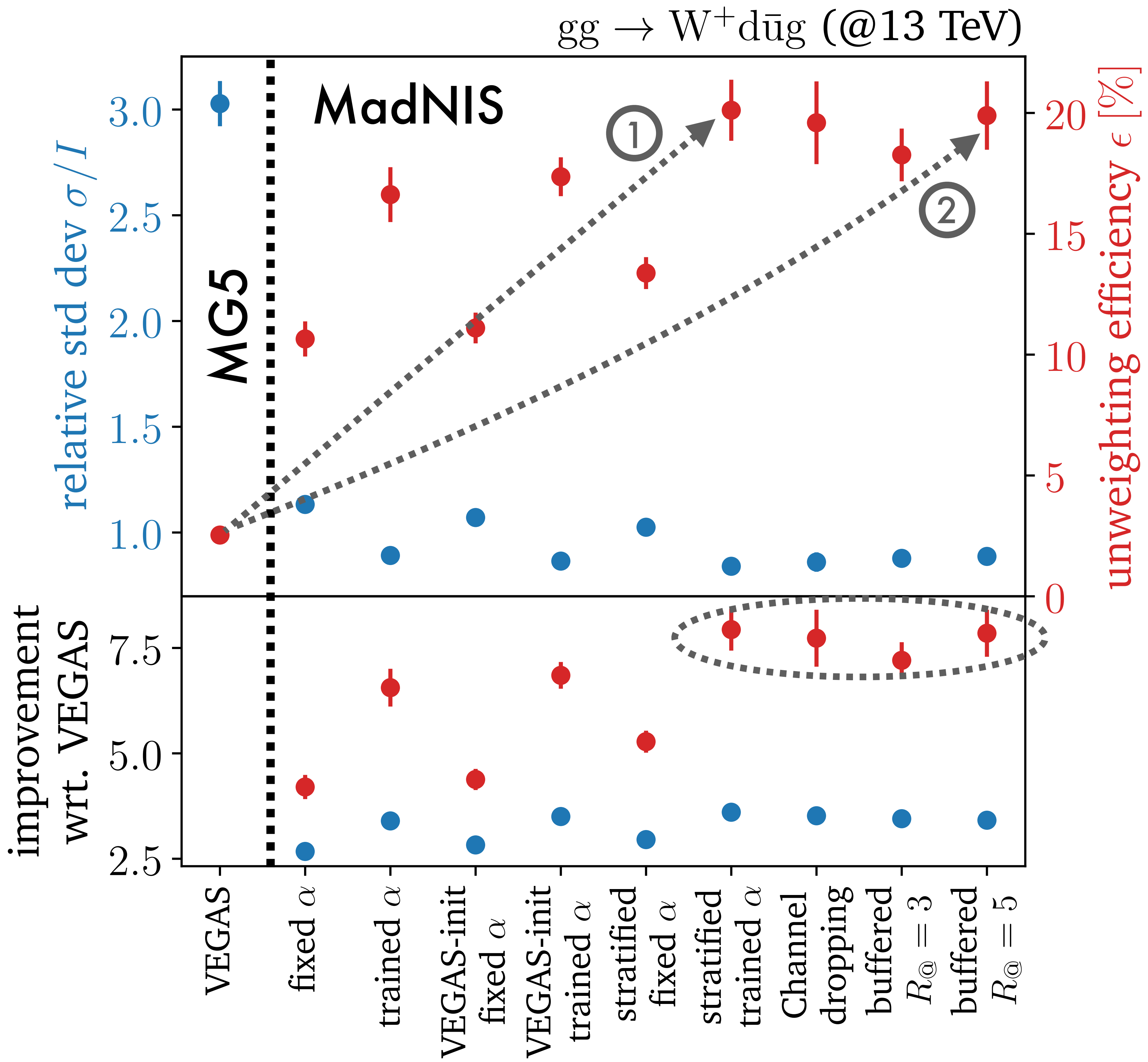
LHC processes

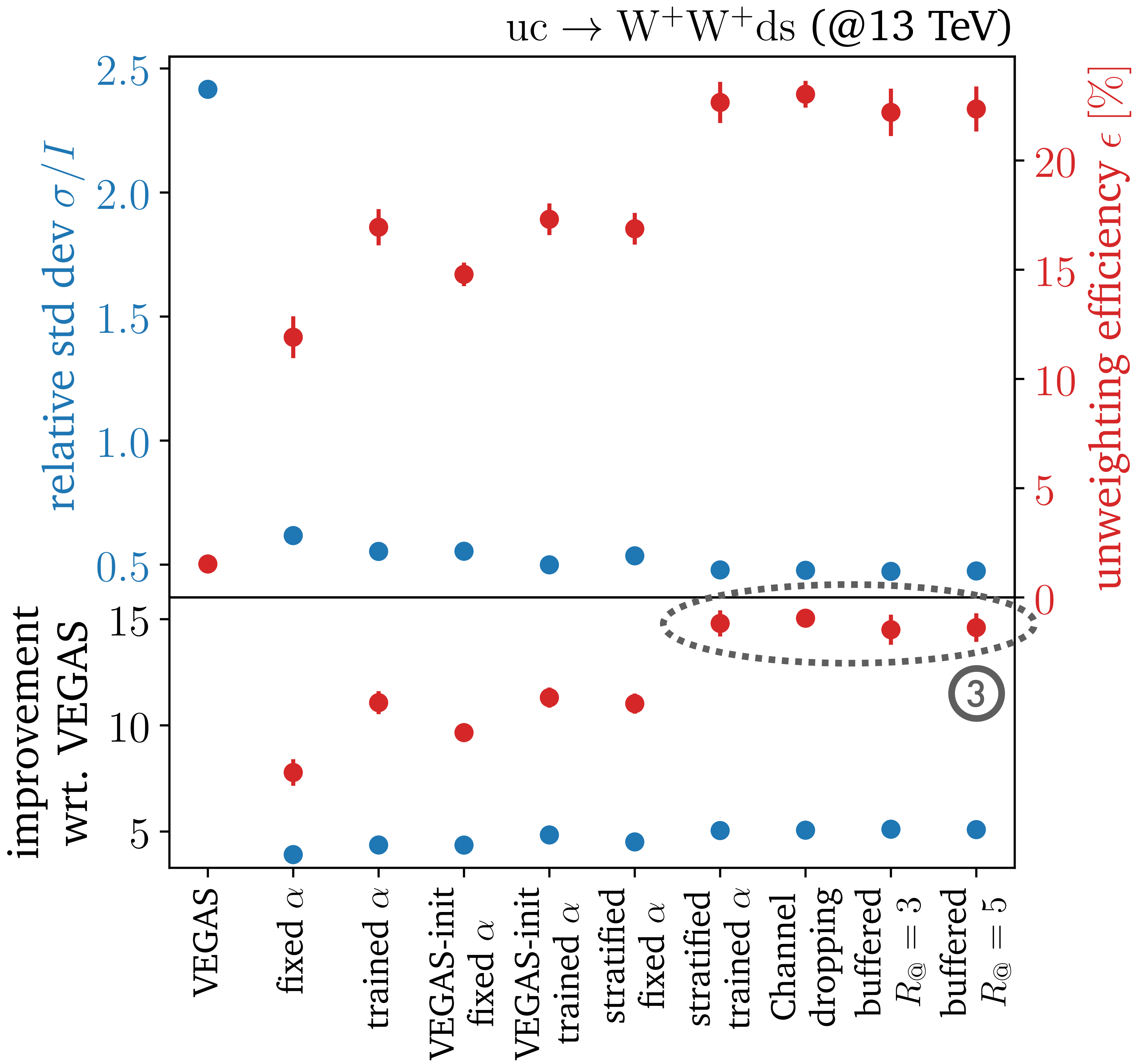




1. excellent results with all improvements
2. same performance with buffered training
3. Larger improvements for processes with large interference terms

LHC processes



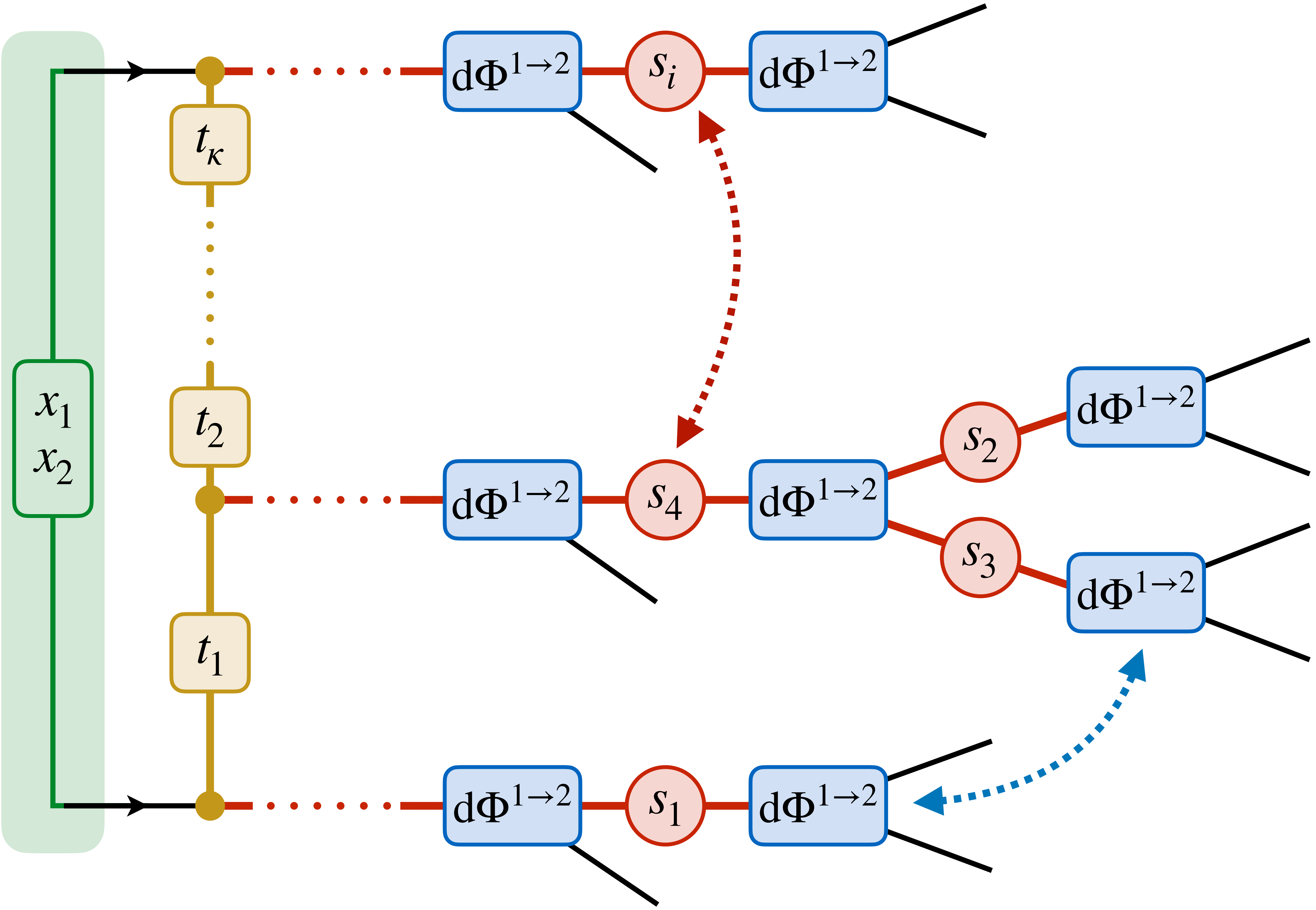


1. excellent results with all improvements
2. same performance with buffered training
3. Larger improvements for processes with large interference terms

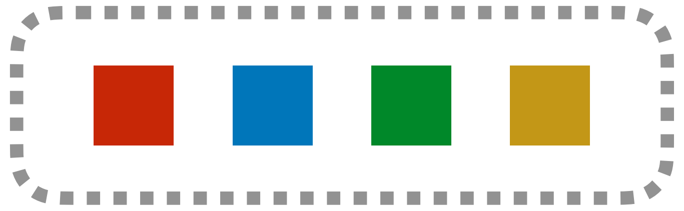
Likelihoods (SFitter)

→ see Nikita's talk
(Tuesday morning)

Differentiable MadNIS-Lite

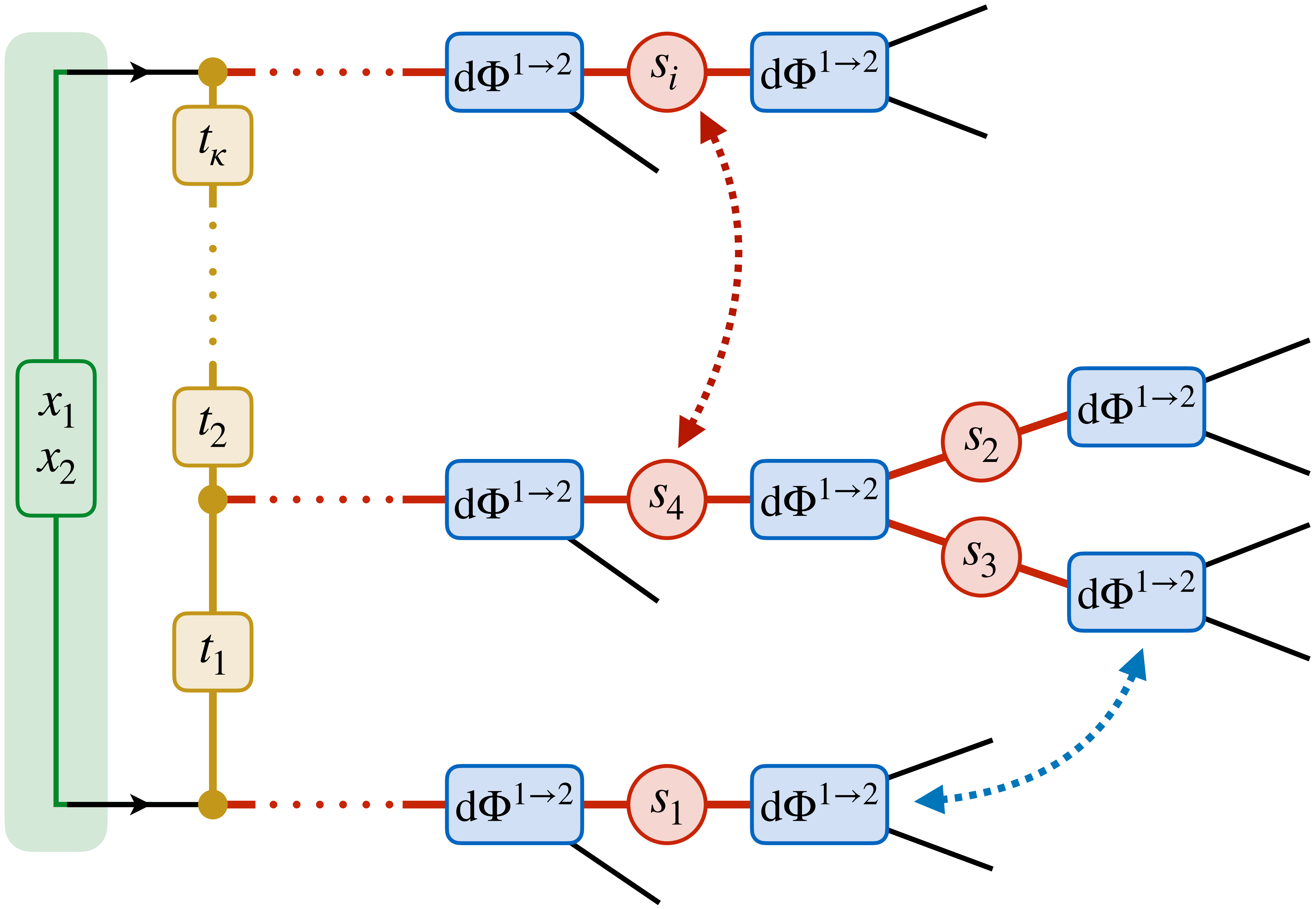


- Feynman diagram inspired PS-mappings
- Phase-space library based on PyTorch
  → fully **differentiable + invertible**
- built-in trainable components

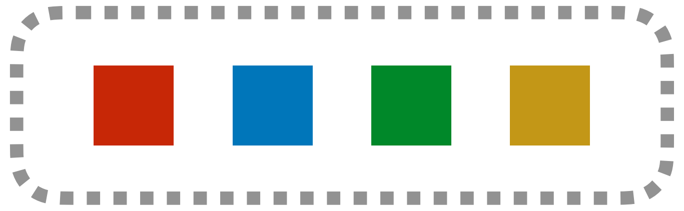


- Tiny number of parameter: shared
  ⇢ between all components of **same type**
  → between all channels

Differentiable MadNIS-Lite



- Feynman diagram inspired PS-mappings
- Phase-space library based on PyTorch
  → fully **differentiable + invertible**
- built-in trainable components

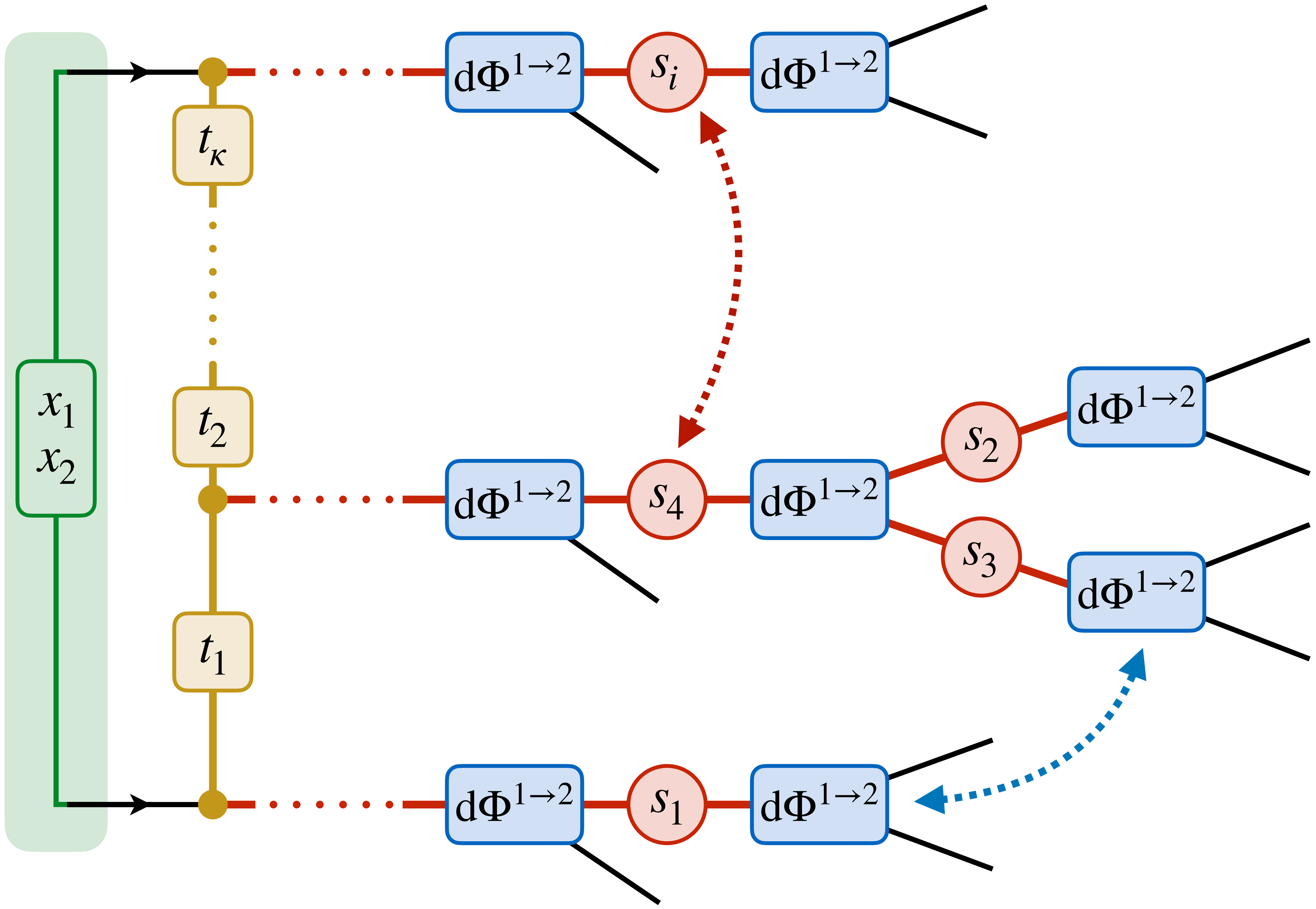


- Tiny number of parameter: shared
  ⇢ between all components of **same type**
  → between all channels

→ details in Theo's talk
(Tuesday morning)

MadNIS-Lite [2408.01486]

Differentiable MadNIS-Lite



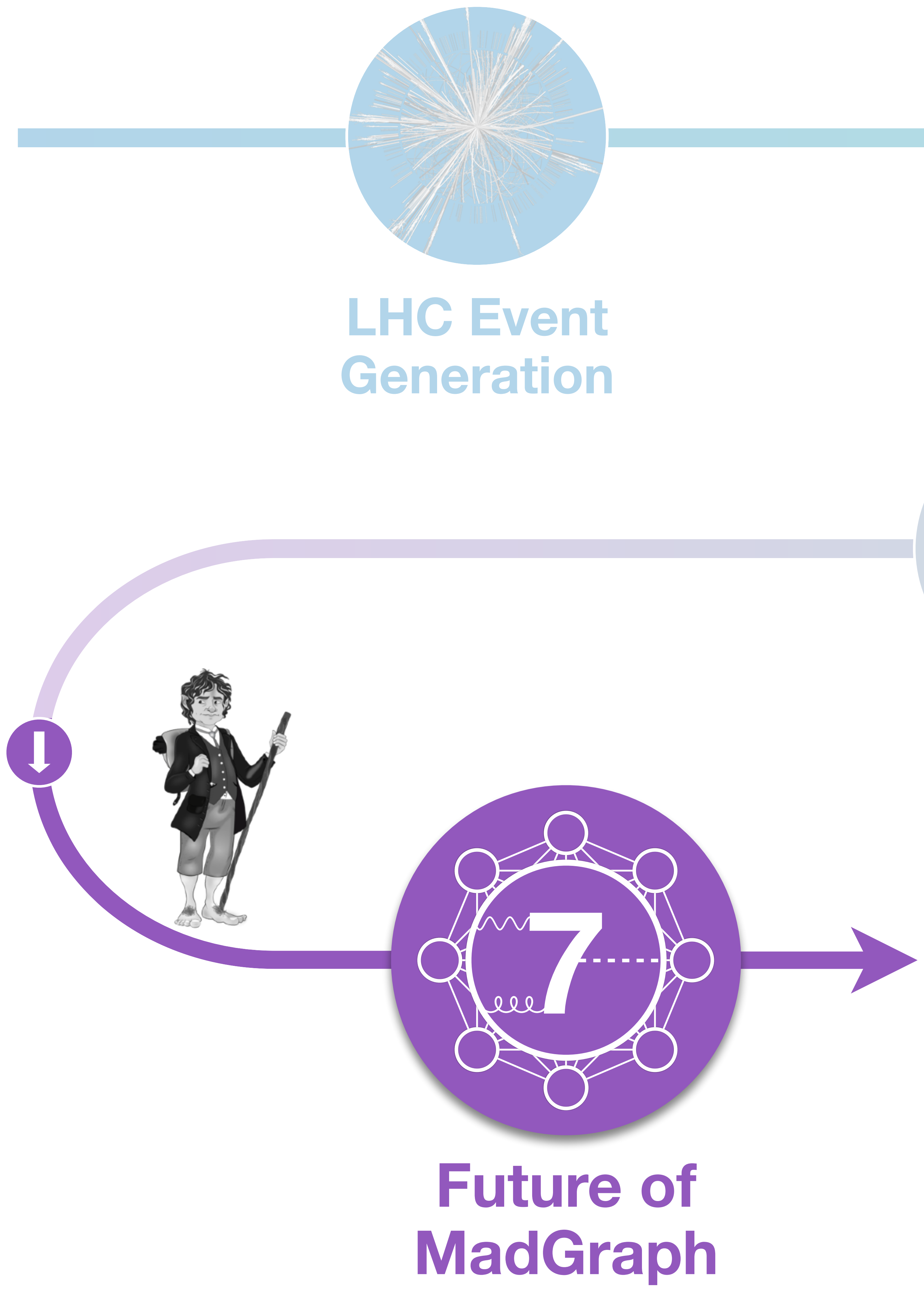
- Feynman diagram inspired PS-mappings
- Phase-space library based on PyTorch
  → fully **differentiable + invertible**
- built-in trainable components
- Tiny number of parameter: shared
  ⇢ between all components of **same type**
  → between all channels

⟶ details in Theo's talk (Tuesday morning)

MadNIS-Lite [2408.01486]

⟶ see also: PDFFlow [2009.06635], MadFlow [2106.10279], MadJax [2203.00057]

The future of MadGraph



The future of MadGraph

Nov
2024

Release of MadNIS package
- python library
- easy install with 'pip install'

→ **December 2024**

Fully integrate into MG5aMC
- multiple partonic processes
- optimized API
- merge with MG@GPU

New MadEvent7
- fully vectorized mappings
- multiple backends (c++, cuda, python,…)

MadNIS@NLO
- subtraction-aware sampling
- fast ML amplitudes (NLO)

Release of MadGraph7
- rigorous testing
- reliable default settings dep. on hardware/process/…

Open Discussion

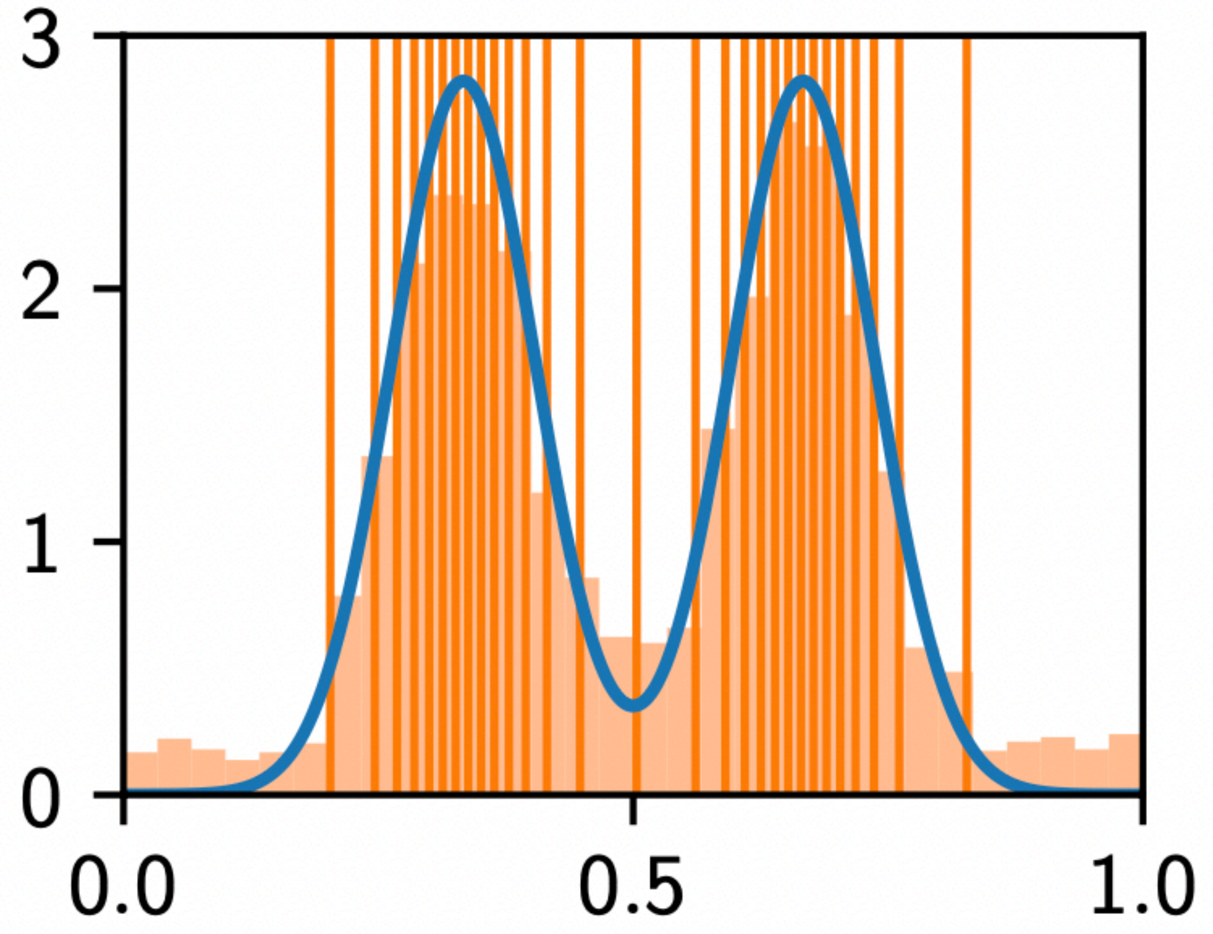
Backup

Importance sampling — VEGAS

Factorize probability

$$p(x) = p(x_1) \cdots p(x_n)$$

Fit bins with equal probability
and varying width

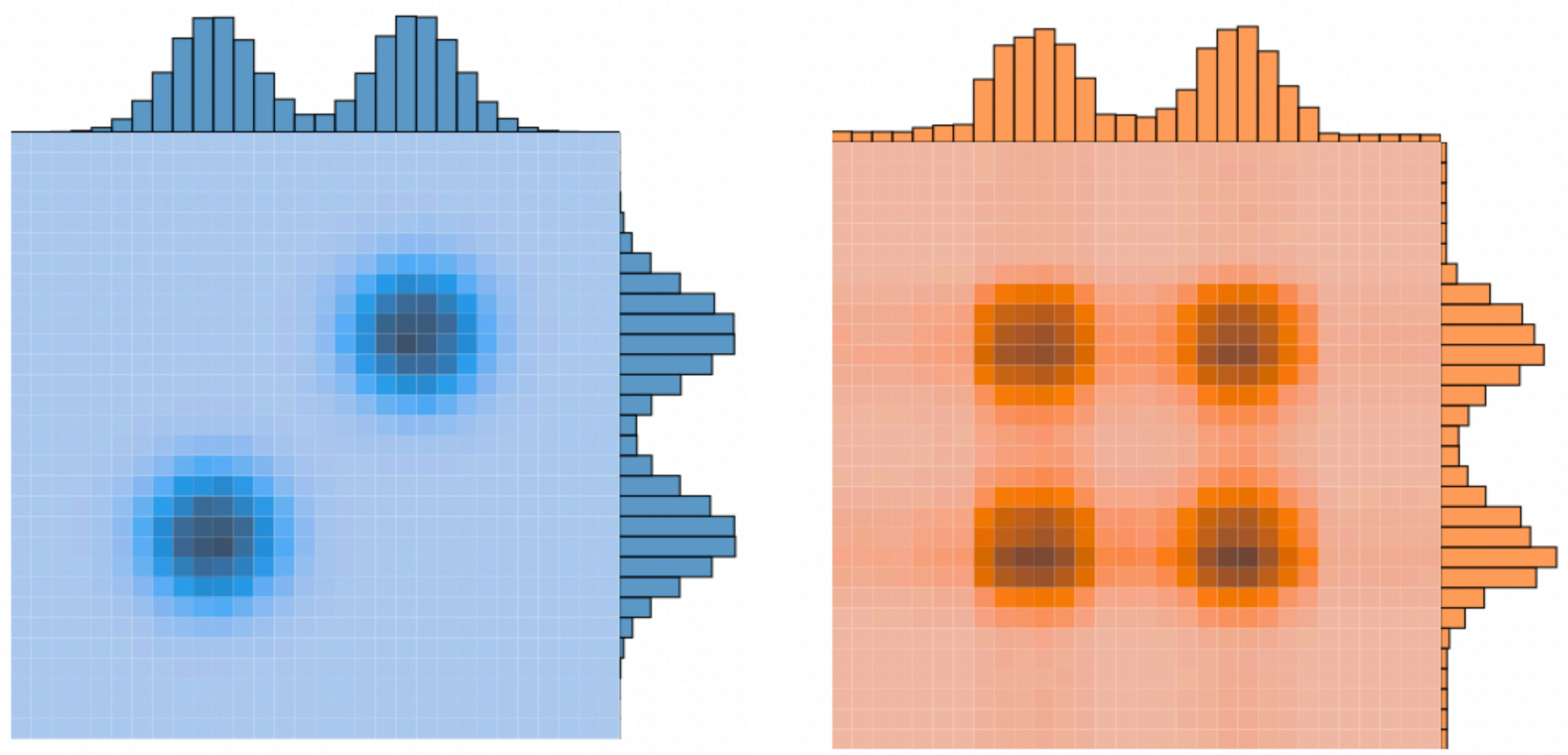


[G. P. Lepage, 1978]

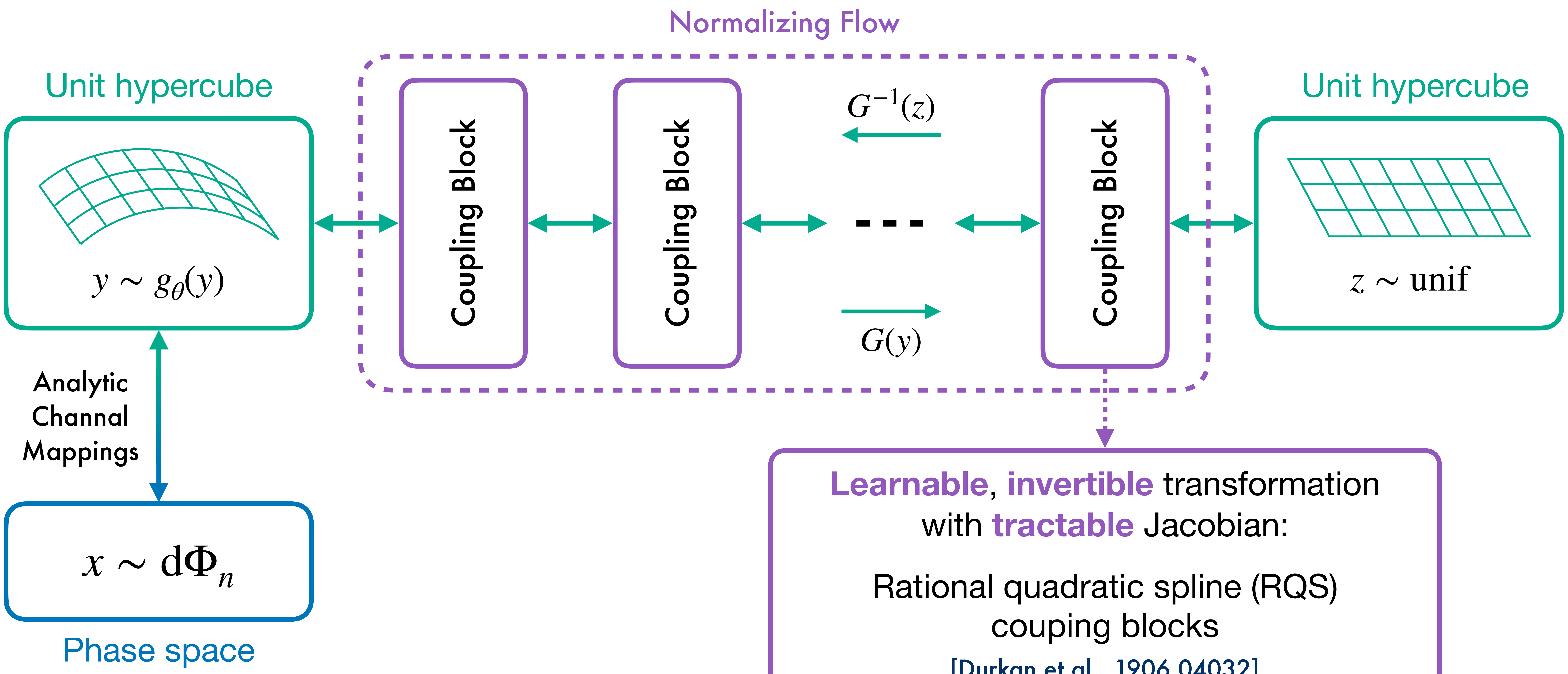
⊕ Computationally cheap

⊖ High-dim and rich peaking functions
→ **slow convergence**

⊖ Peaks not aligned with grid axes
→ **phantom peaks**



Neural importance sampling



Learnable, **invertible** transformation with **tractable** Jacobian:

Rational quadratic spline (RQS) couping blocks

[Durkan et al., 1906.04032]

Flows for NIS: [Gao *et al*, 2001.05486+2001.10028] [Bothmann *et al*, 2001.05478] [Winterhalder *et al*, 2112.09145] [Deutschmann *et al*, 2401.09069]