End to End Inference in HEP

Hammers & Nails 2023

Lukas Heinrich

Technische Universität München

ODSL ORIGINS Data Science Lab

The Core Problem in HEP: Our Nail

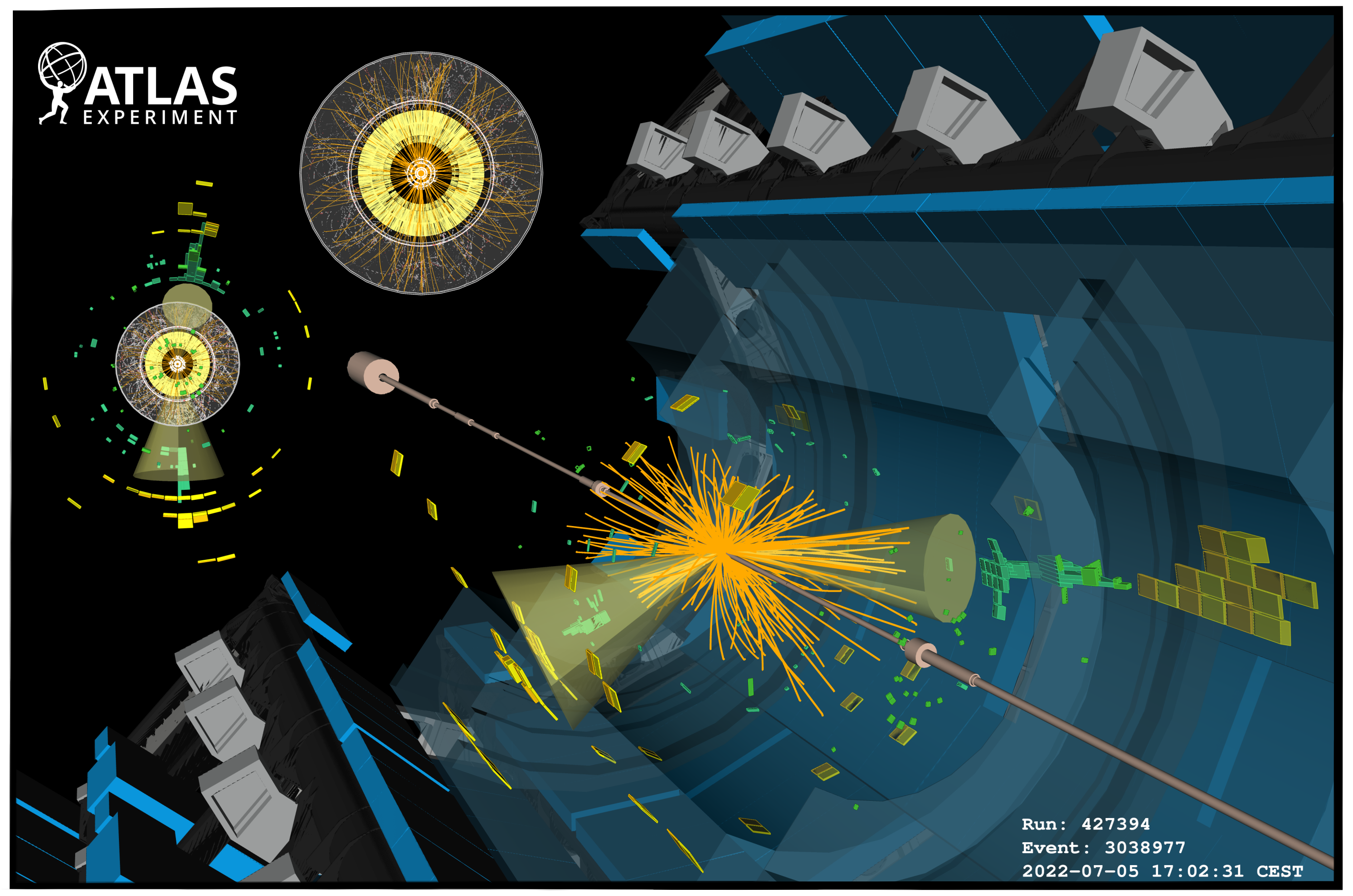
Theory

$$\mathcal{L} = -\frac{1}{4} F_{\mu\nu} F^{\mu\nu} + i \bar{\Psi} \not{D} \psi + h.c. + \bar{\psi}_i y_{ij} \psi_j \phi + h.c. + |D_\mu \phi|^2 - V(\phi)$$

O(10) Parameters of Interest

$$p(x|\theta) \longrightarrow$$

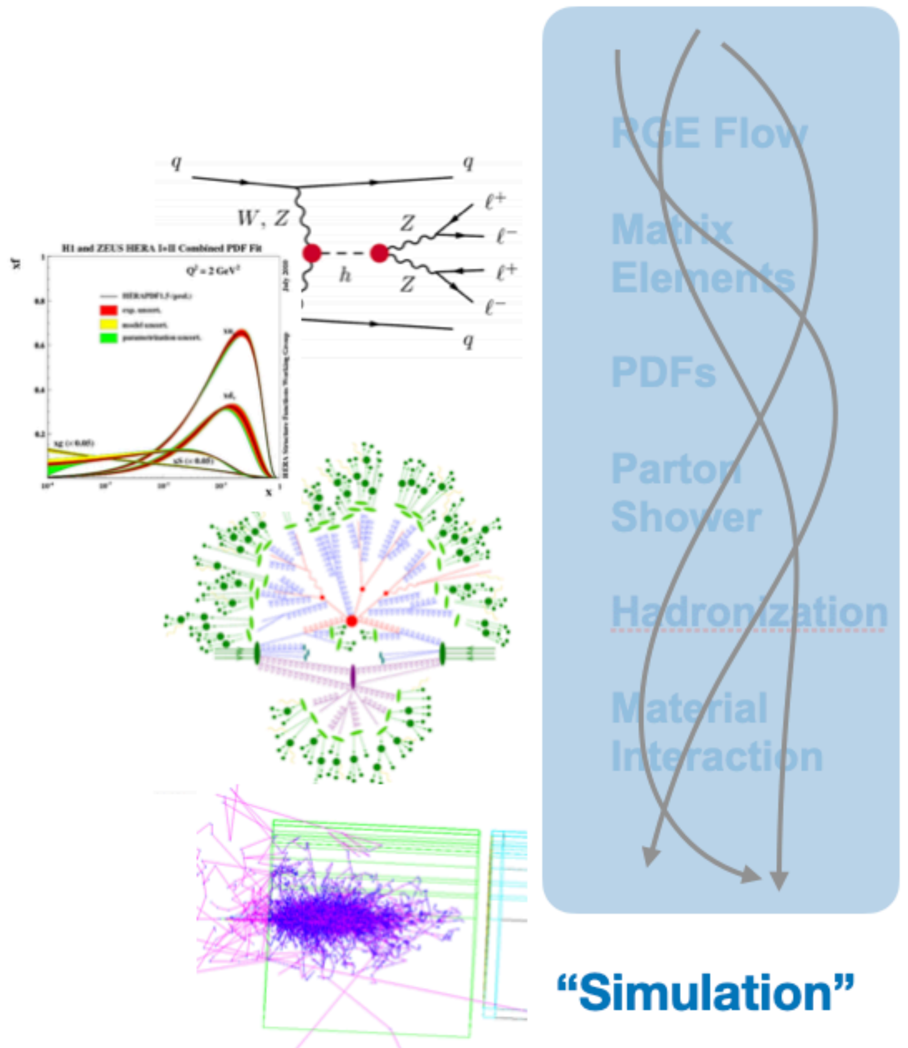
Data



100M Channels

Hypothesis

θ



"Simulation"

$$p(x|\theta) = \int dz \;\; p(x|z)p(z|\theta)$$

HEP is defined by an **intractable likelihood**

The Core Problem in HEP: Our Nail

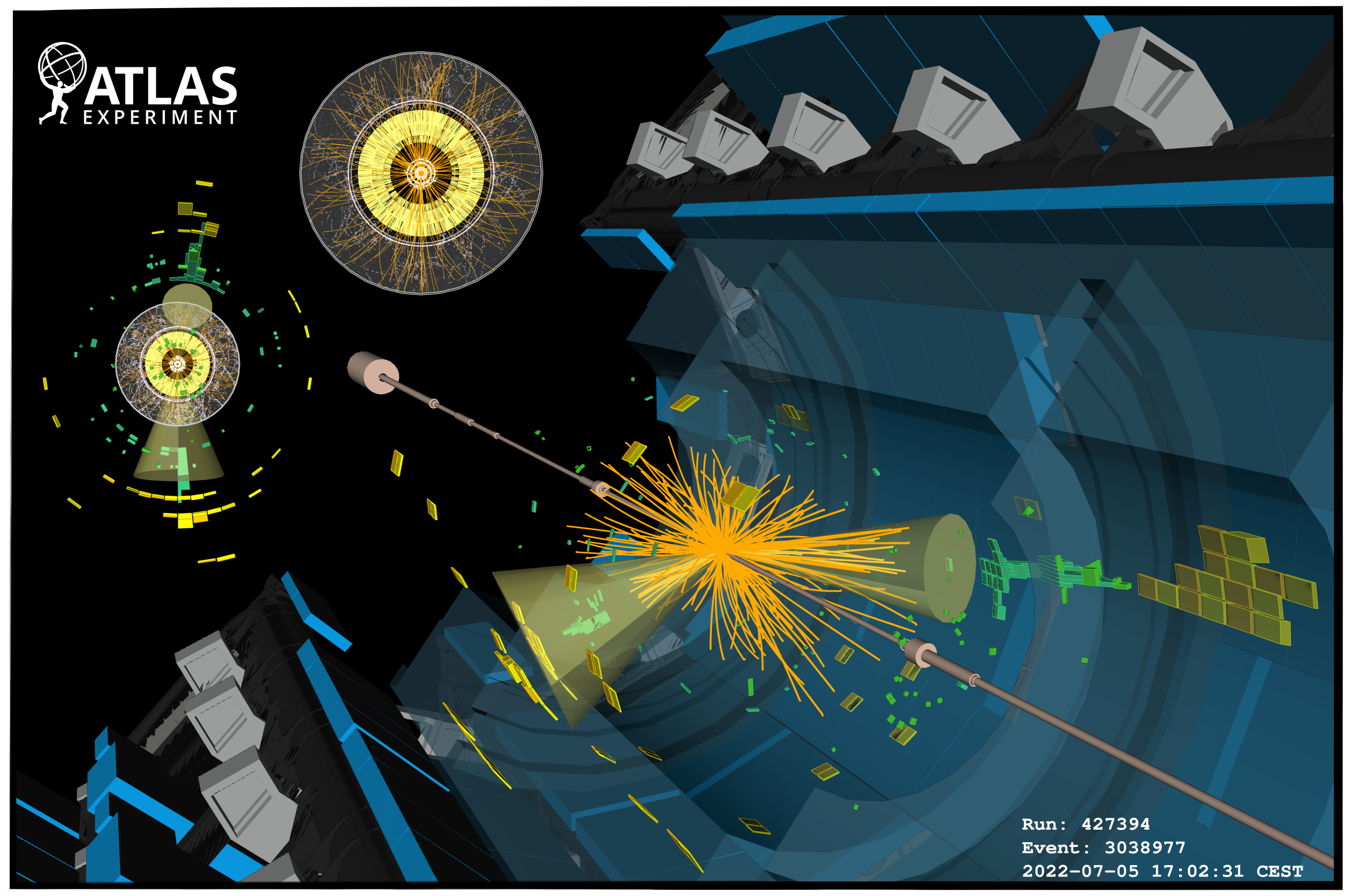
Theory

$$\mathcal{L} = -\frac{1}{4} F_{\mu\nu} F^{\mu\nu} + i\bar{\Psi} \not{D} \psi + h.c. + \bar{\Psi}_i y_{ij} \Psi_j \phi + h.c. + |D_\mu \phi|^2 - V(\phi)$$

O(10) Parameters of Interest

$$\xrightarrow{p(x\,|\,\theta)}$$

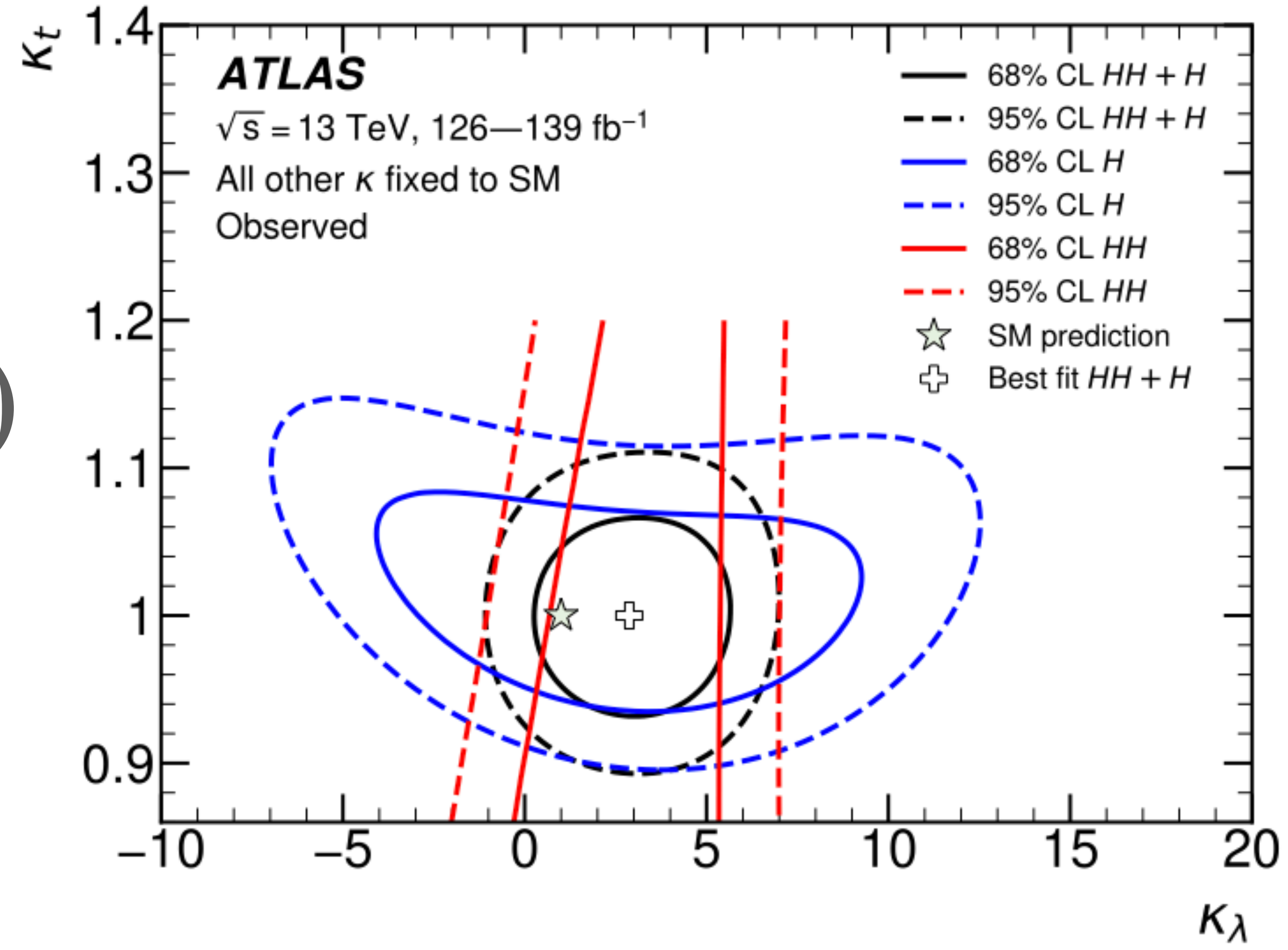
Data



100M Channels

$$\xrightarrow{p(\theta\,|\,x)}$$

Inference



O(10) Parameters of Interest

HEP is defined by an **intractable likelihood** and yet:
we **want to infer something** about nature

Our Hope/Hammer: ML should be able to help us

ML + HEP = ❤️

The ML and HEP setups are fortunately very aligned.

If you squint your eyes, you can recognize many of today's buzzwords already in our old, traditional HEP workflows

Amortized Simulation Based Inference

Multi-Modal Foundation Models with Attention

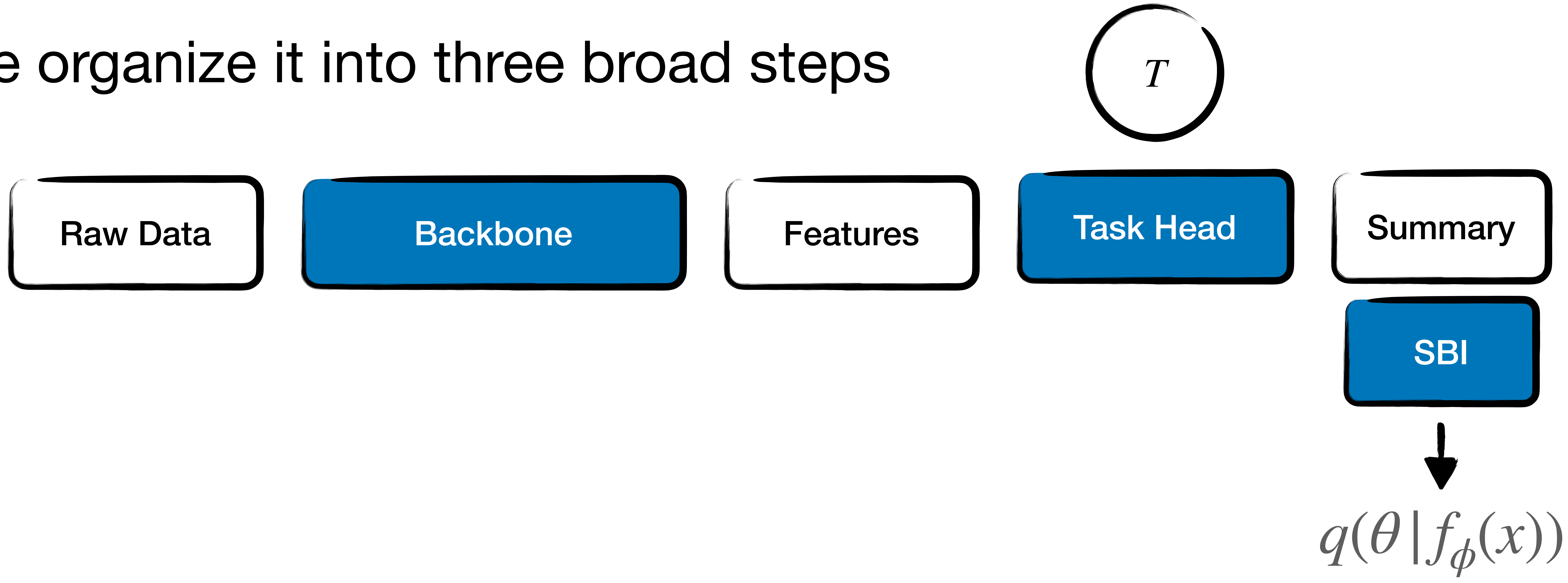
Finetuning

A (to me) useful - if distorted - lens to make connections and see how to move forward

HEP in the modern ML Language

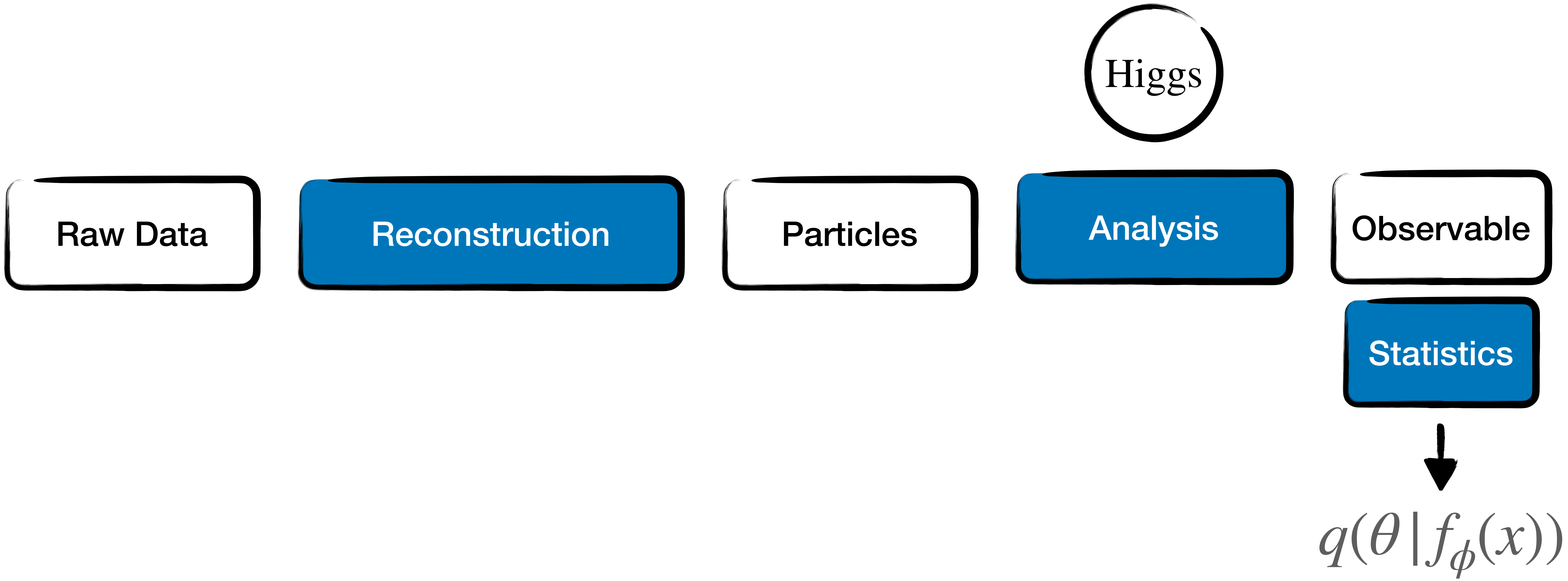
The raw data in HEP is useless, and the way we run our inference is through powerful, meaningful summary statistics

We organize it into three broad steps



HEP in the our usual Language

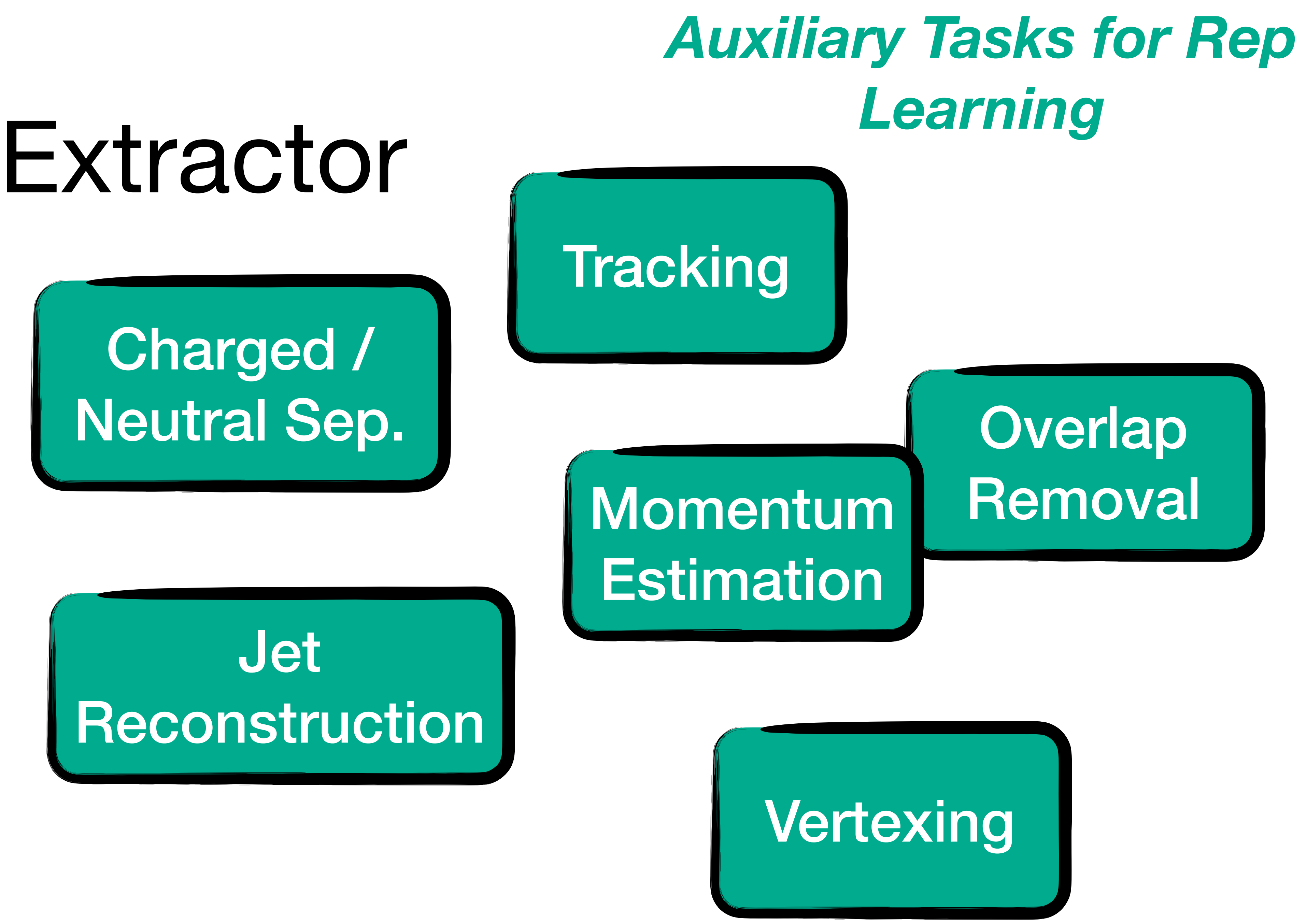
When talking to a physicist we'll label the boxes differently, but they are essentially the same.



Reconstruction = Our Foundation Model

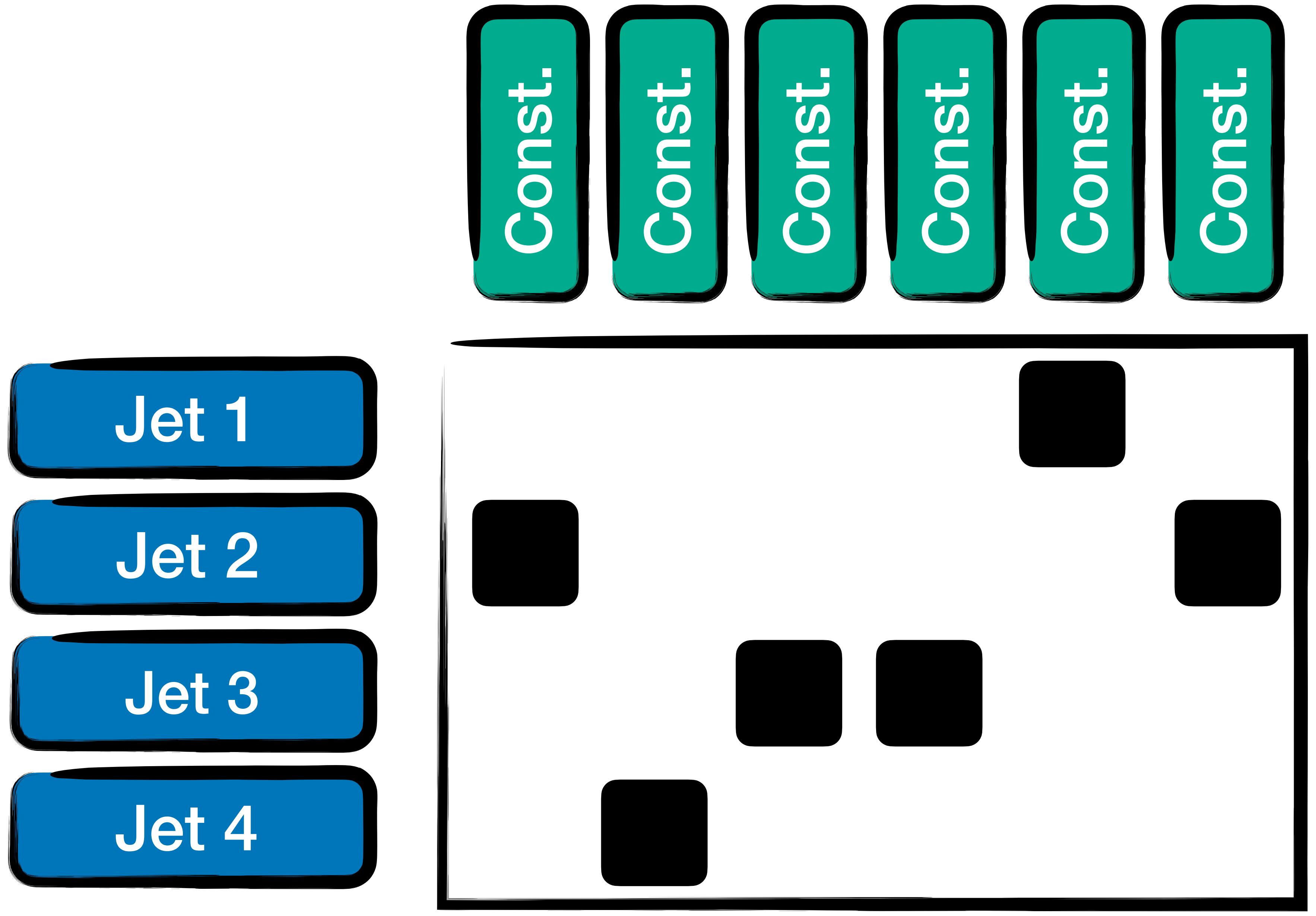
Very complex, optimized on a diverse set of auxiliary tasks, that aid in learning a representation that will eventually be useful for the main task (e.g. measure Higgs)

Sole Purpose: Act as a Universal Feature Extractor

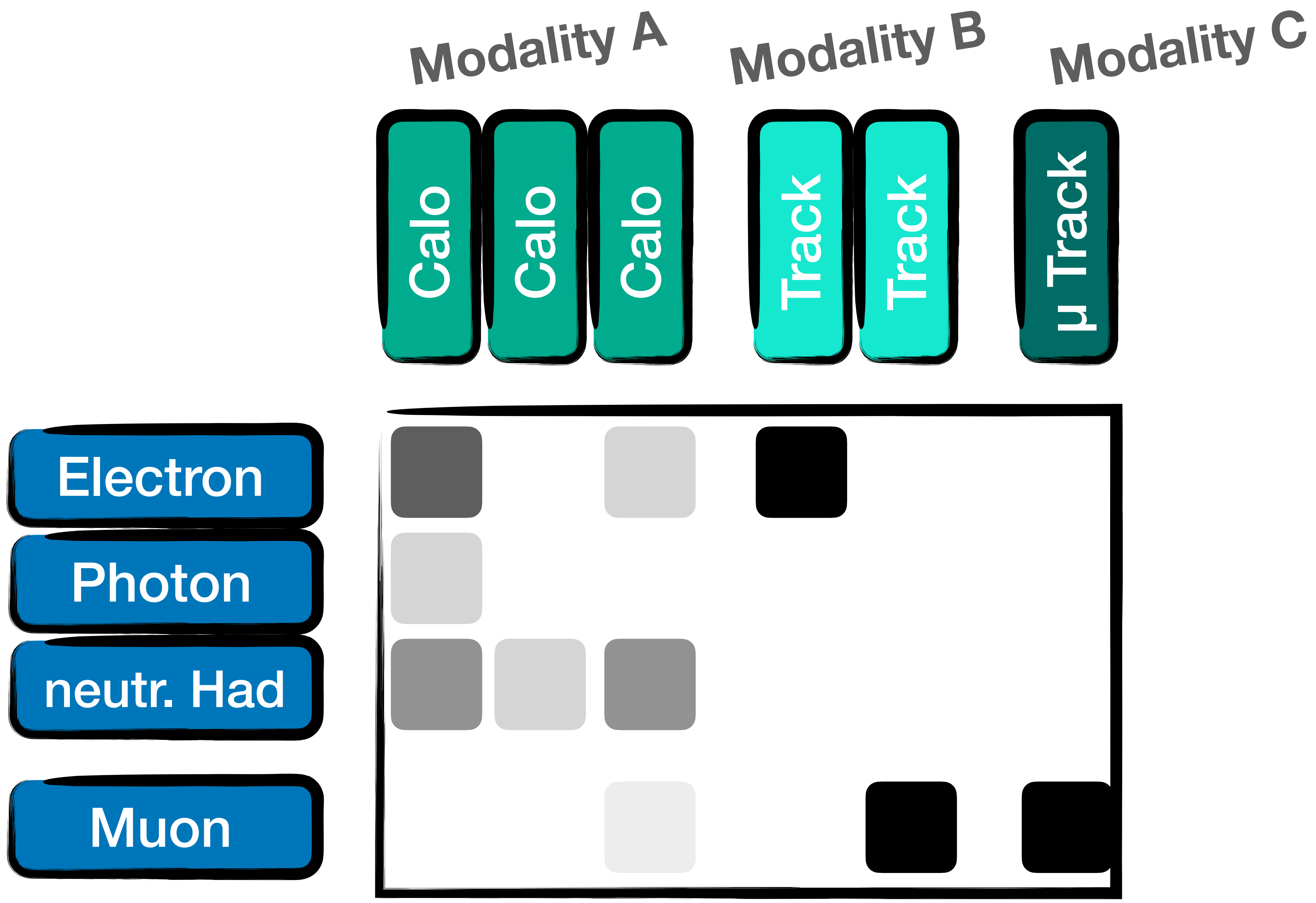


“Multi-Modal (Slot) Attention”

Inside the backbone, one of the key operations is a **data-dependent** (think: “attention”) grouping signals from **multiple data modalities** into higher-level objects (think “slots”)



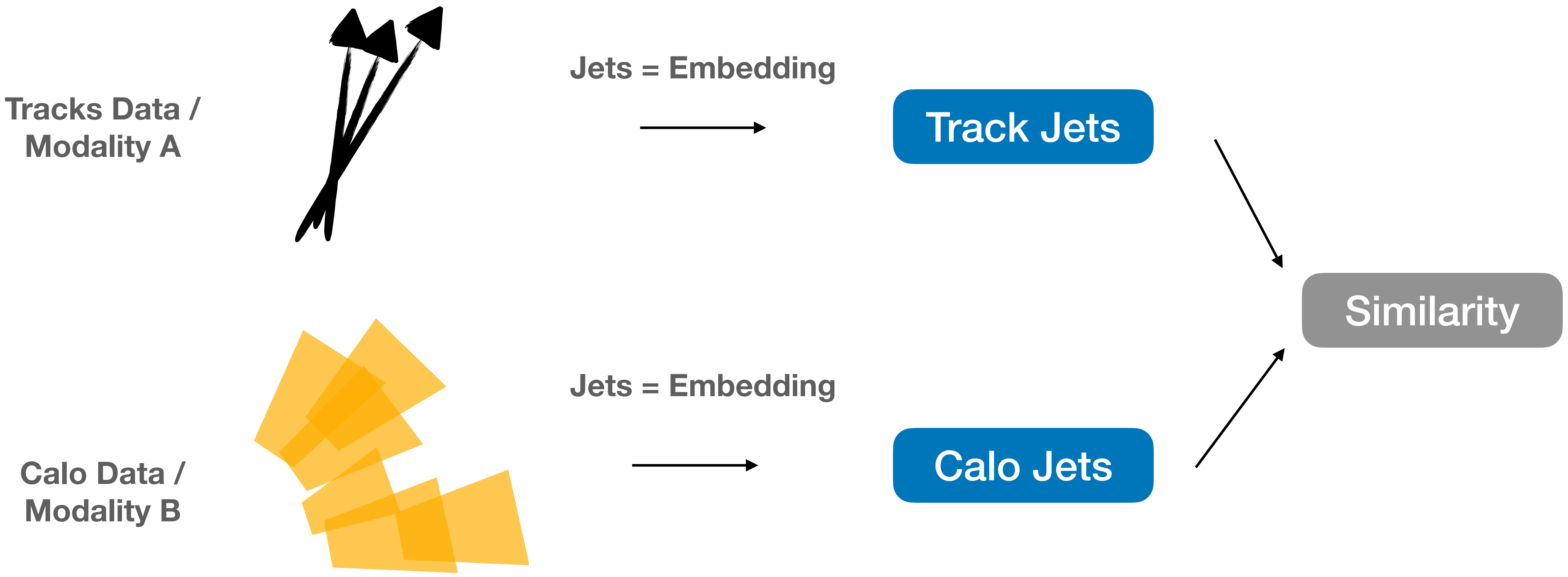
Hard Attention (e.g. Jet Clustering)



Soft Attention (e.g. Particle Flow)

Similarity in “Embedding Space”

We also take these “slots” (representations or low-level data) and compare them in the representation space…



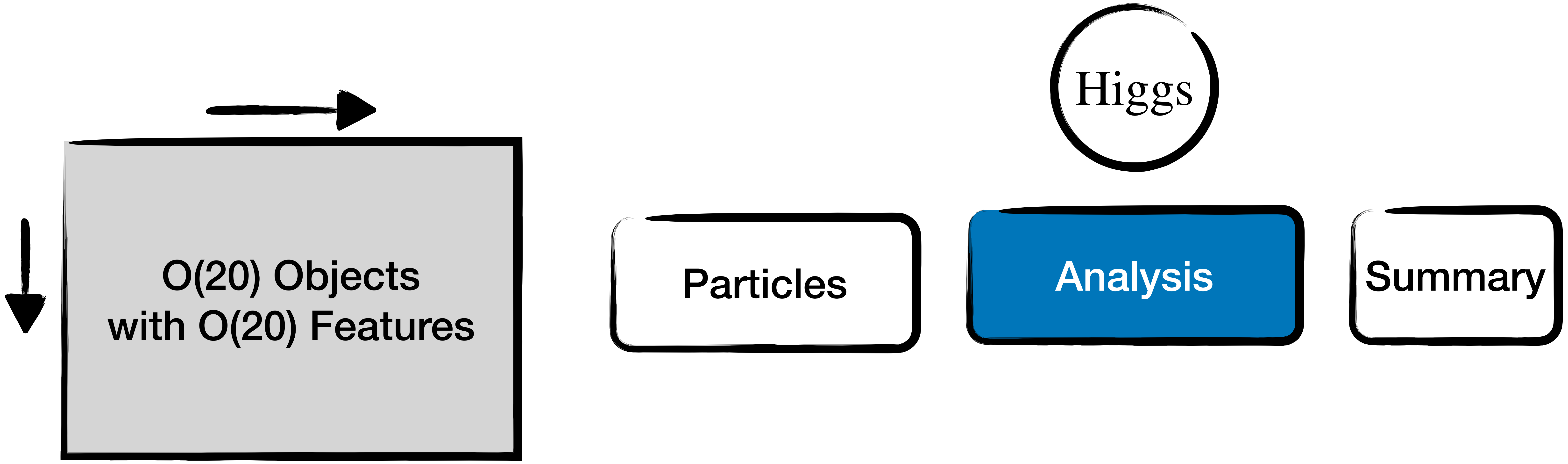
Eur. Phys. J. C (2013) 73:2304

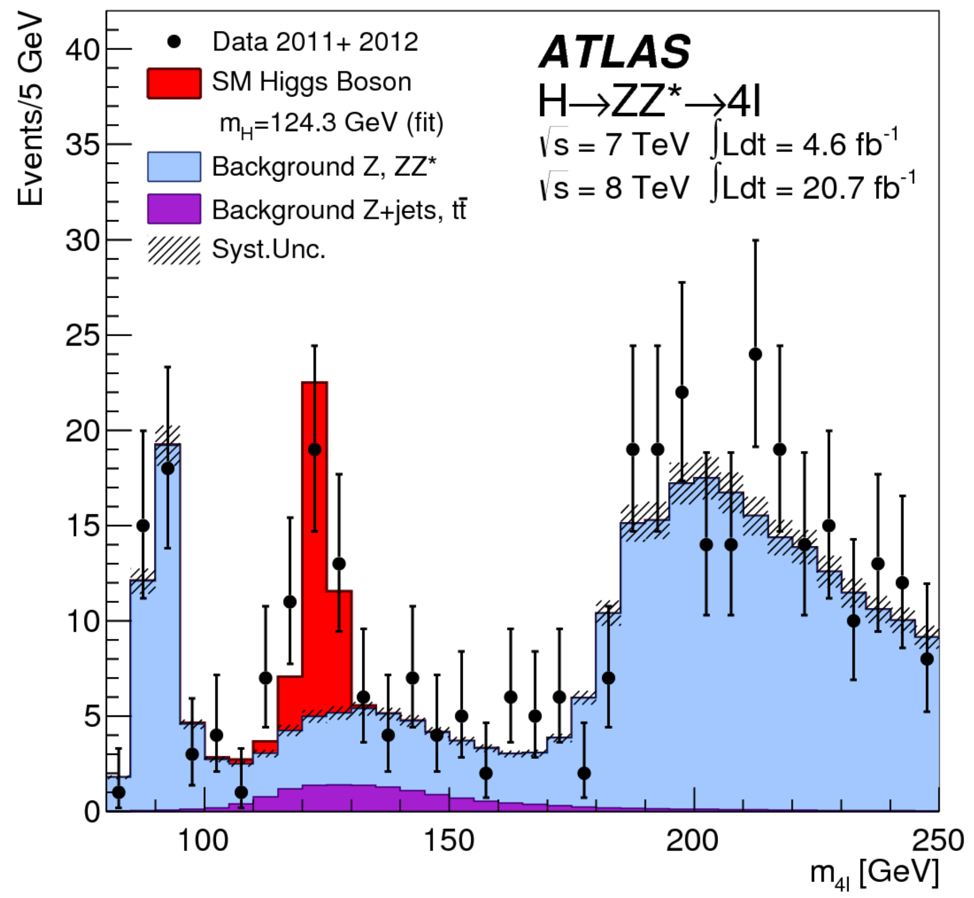
The JES systematic uncertainty is derived for isolated jets.[21] The response of jets as a function of the distance to the closest reconstructed jet needs to be studied and corrected for separately if the measurement relies on the absolute jet energy scale. The contribution to the JES uncertainty from close-by jets also needs to be estimated separately, since the jet response depends on the angular distance to the closest jet. This additional uncertainty can be estimated from the Monte Carlo simulation to data comparison of the p_T-ratio between calorimeter jets and matched track jets in inclusive jet events as a function of the isolation radius. This is discussed in more detail in Sect. 17.

[Link]

The “Head”

The “Downstream Task” is the what people are mostly working on





Observations

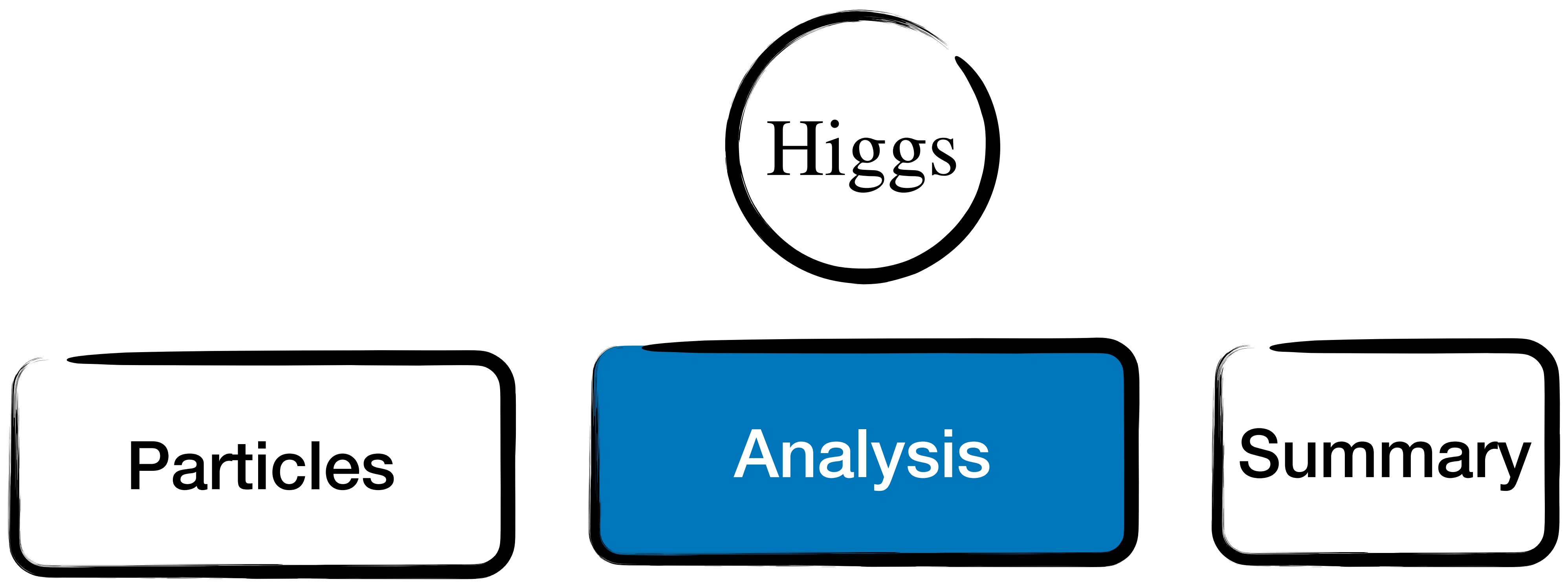
ML

“The Head is small & simple
compared to the Backbone”

“Optimizing the Head is
fast & cheap”

“New Foundation Model = $$$”

“Head is task dependent but works
reasonably well on **frozen** backbone”



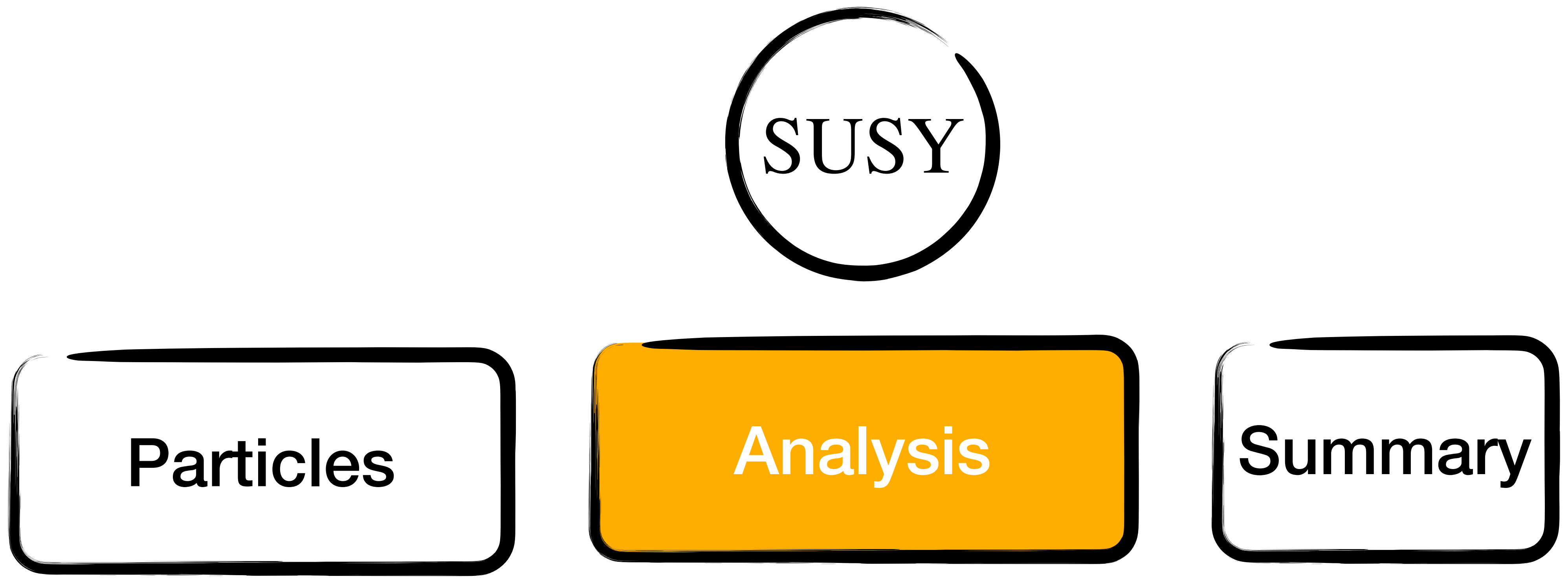
HEP

Analysis: Grad Student Effort
Reco: Full Collaboration Effort

Train a BDT / NN to separate
signal from background: easy!

Reprocessing Campaign

Nominal Reconstruction is good
as a starting point

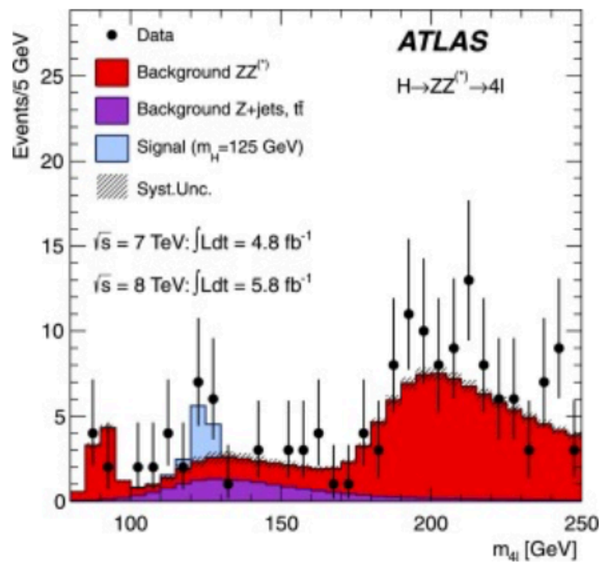


"The Inference"

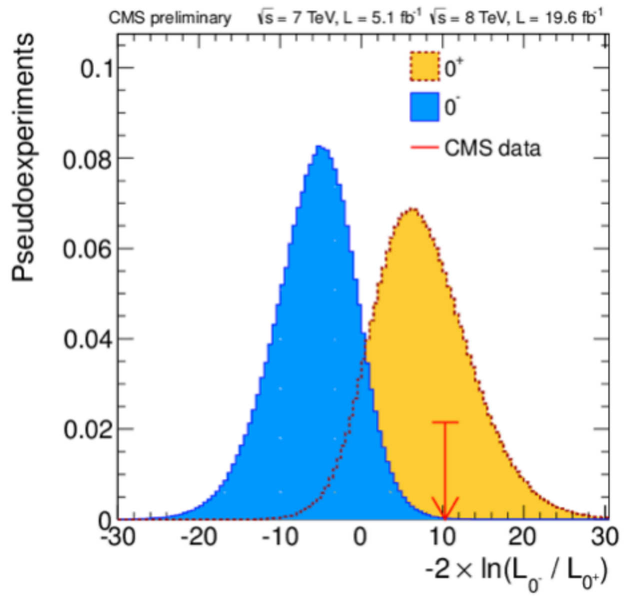
HEP has forever been "simulation-based inference". But:

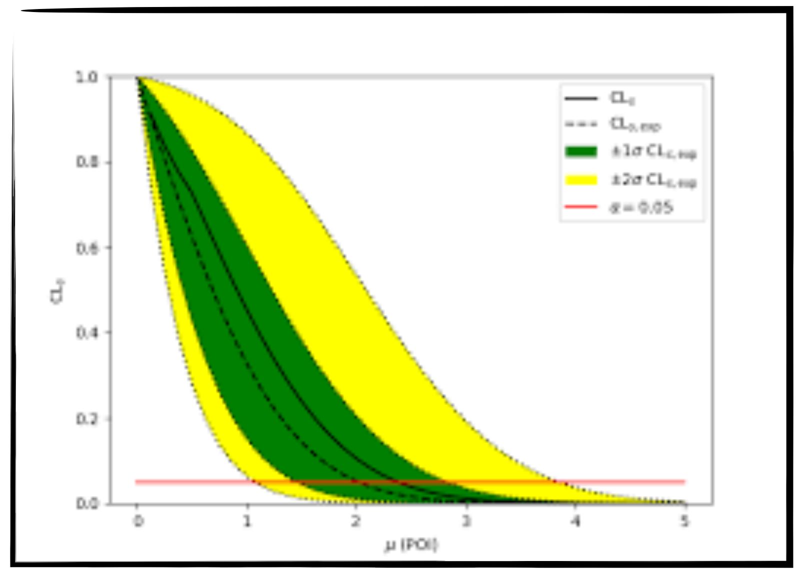
- using pre-ML techniques (e.g. histograms instead of flow)
- Frequentist School & i.i.d. data

Summary → Density → Likelihood Model → Test Statistic → Final Result



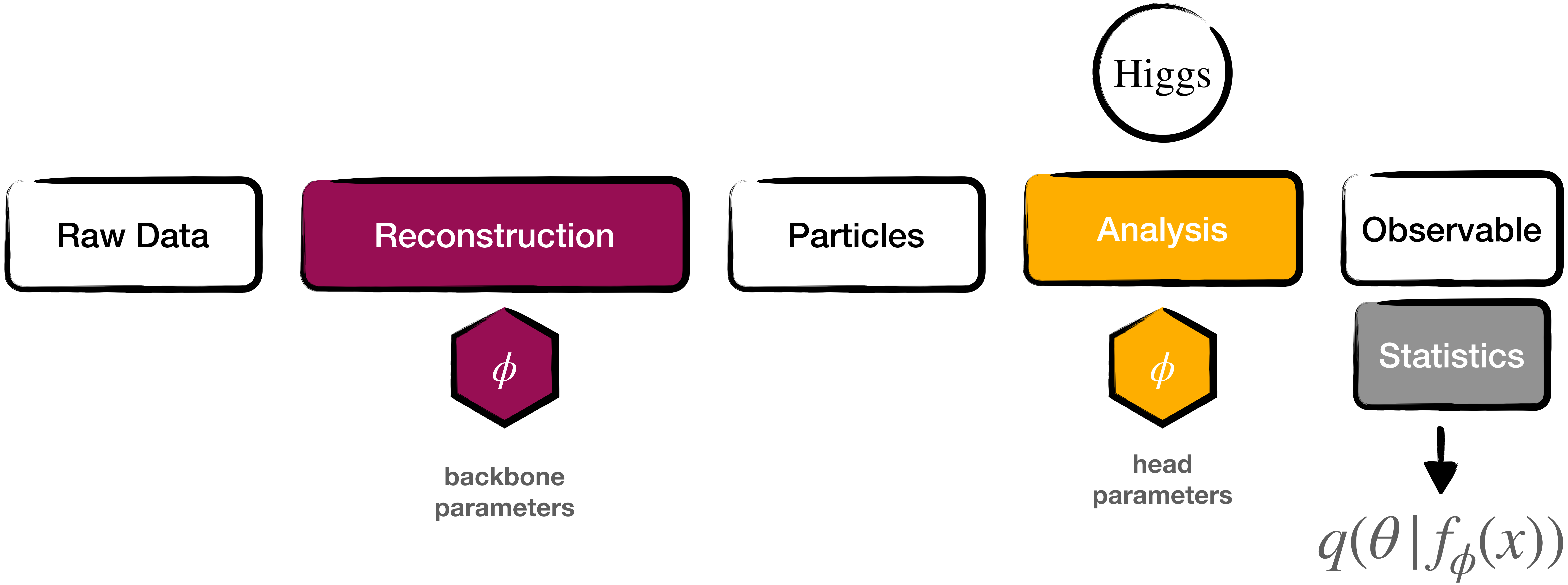
$$p(n,a|\theta) = \underbrace{\prod_{\text{channel } c}\prod_{\text{bin } b} \mathrm{Pois}(n_{cb}|\lambda_{cb}(\theta))}_{\text{main}} \underbrace{\prod_{\text{constraint } \chi} p_\chi(a_\chi|\chi)}_{\text{auxiliary}} \qquad \lambda_{cb}(\theta) = \sum_{\text{sample } s} \lambda_{cbs}(\theta)$$





Amortized Variational Inference

We **optimize the parameters** of the reconstruction **once** for and run these for all possible raw data inputs instead of trying to interpret any single event especially well



Amortized Variational Inference

With Variational Techniques we are worried about

- how efficient do we optimize within the variational family
- how close does this family come to the true posterior

True Result

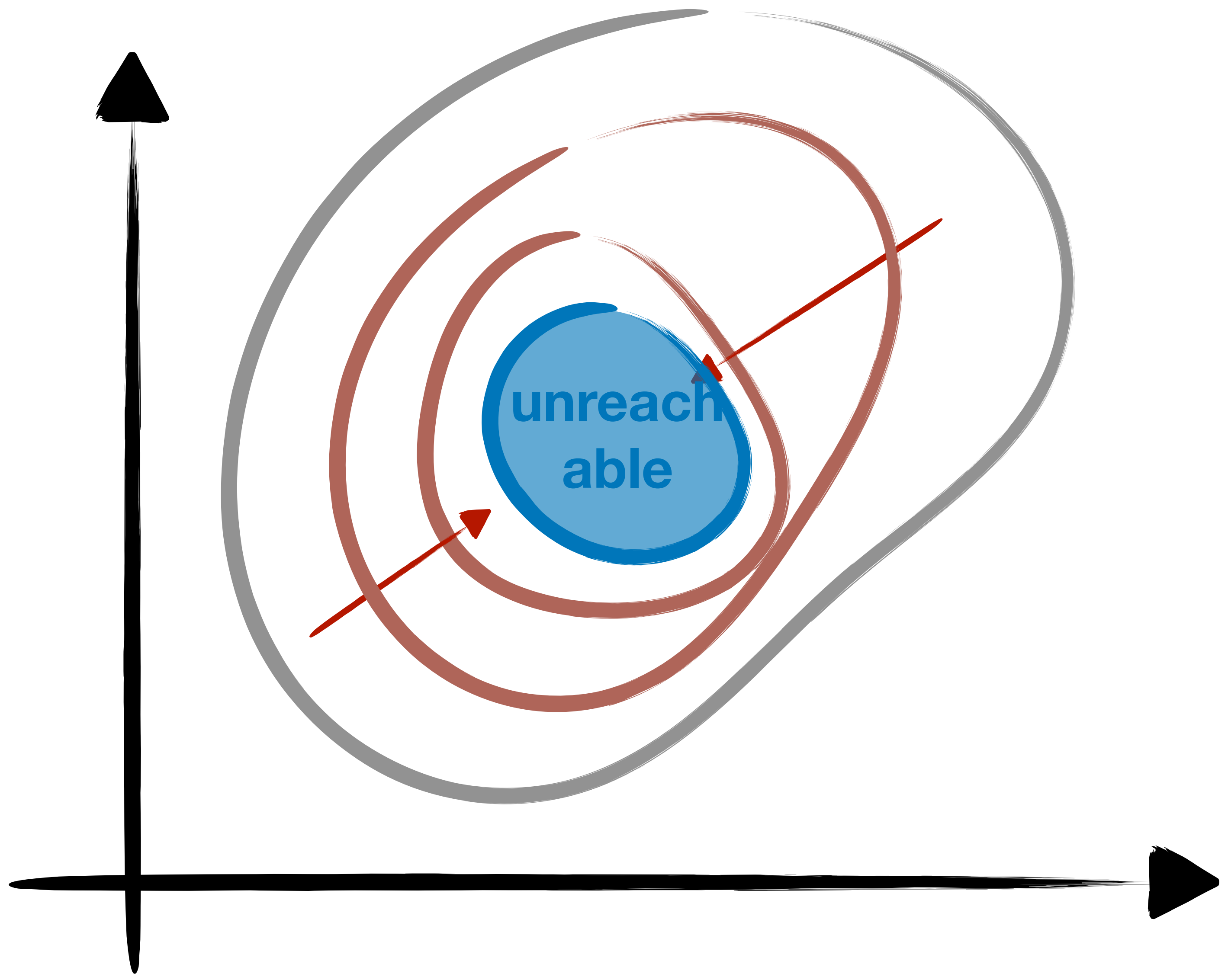
Best Possible Result
(given our pipeline)

Obtained Result

$$p(\theta|x) \leftrightarrow q_{\phi_{\min}}(\theta|x) \leftrightarrow q_{\hat{\phi}}(\theta|x)$$

Variational Gap

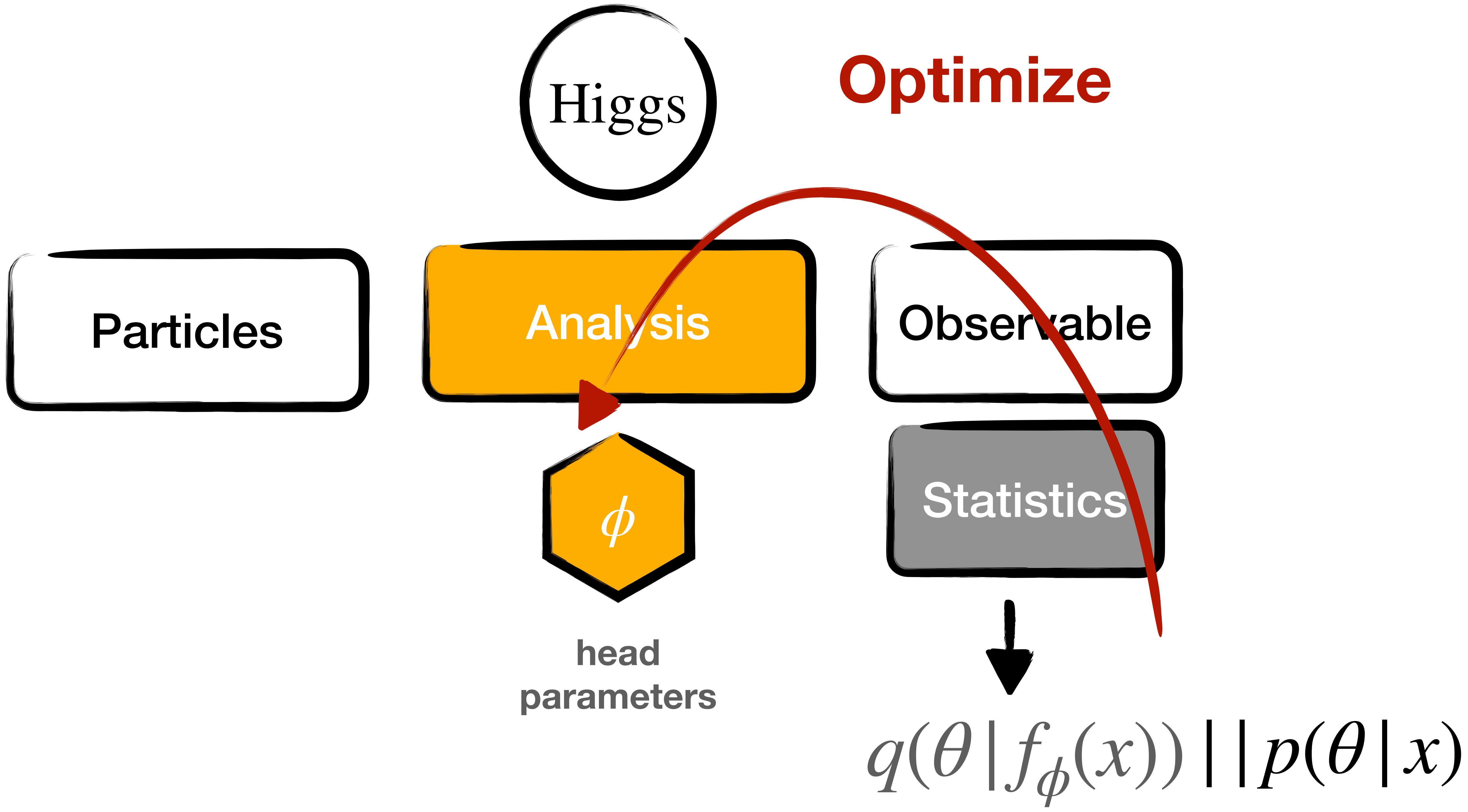
Efficient Optimization



Measurements
(e.g. Higgs Couplings)

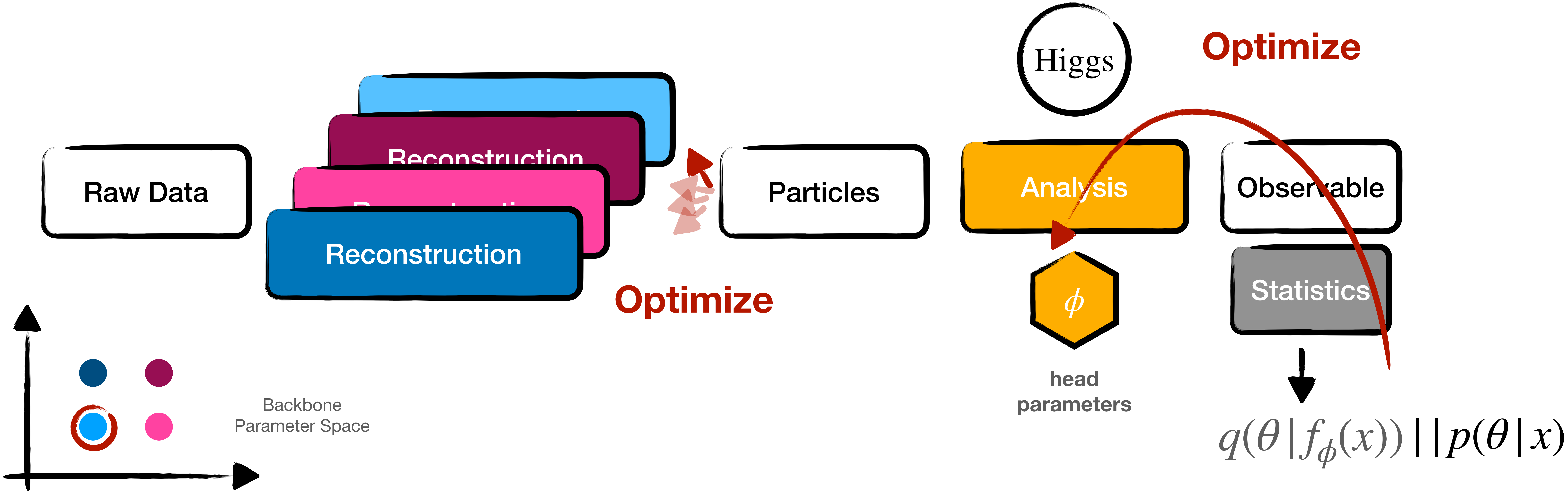
Optimization

The **optimization** of the sensitivity is primarily the job of the analysis.
→ i.e. you (=grad student) optimize the head for the task.



Finetuning

We *also already do* do “finetuning” as well! Every analysis has a choice of **possible backbones (“working points”)**
→ i.e. you (=grad student) optimize it by trial/error & received wisdom



Upshot

If you squint HEP already has a lot of similar workflows of modern ML built in (some analogies are perhaps a bit too stretched)

Foundation Models, Task Heads, (Slot) Attention, Pretraining, Finetuning, Object Representations, … a helpful analogy for me

So what's the role of actual ML?

Automating, Optimizing, Realizing this Pipeline to extract the most science

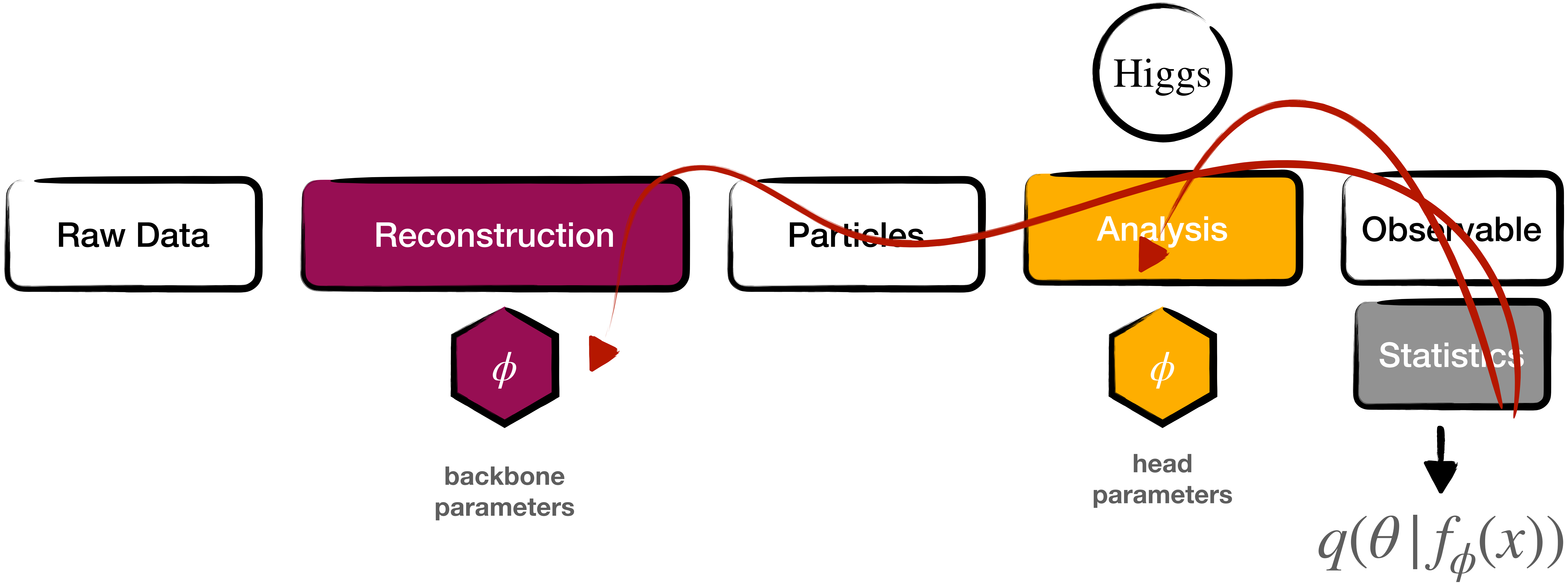
Obvious Idea: Gradient-Based Optimization

$$\nabla_\phi \mathrm{Science}(\phi)$$

Graduate Student / Collaborative Descent

Automatic Gradient Descent

Obvious Idea: Gradient-Based Optimization



The Issue

This would all work great *if* these were reality instead of analogy.

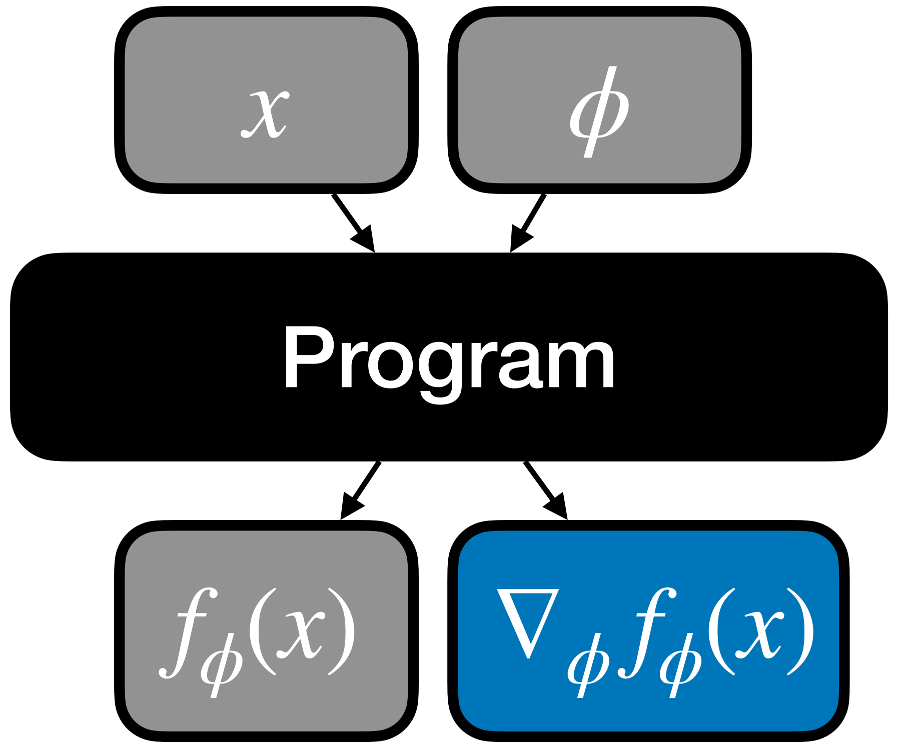
Our Backbone is not a Neural Network with weights & biases

Neither is our Analysis

Both are complex mixtures of (yes) NNs but also hand-written programs & control flow. Need to have gradients of programs!

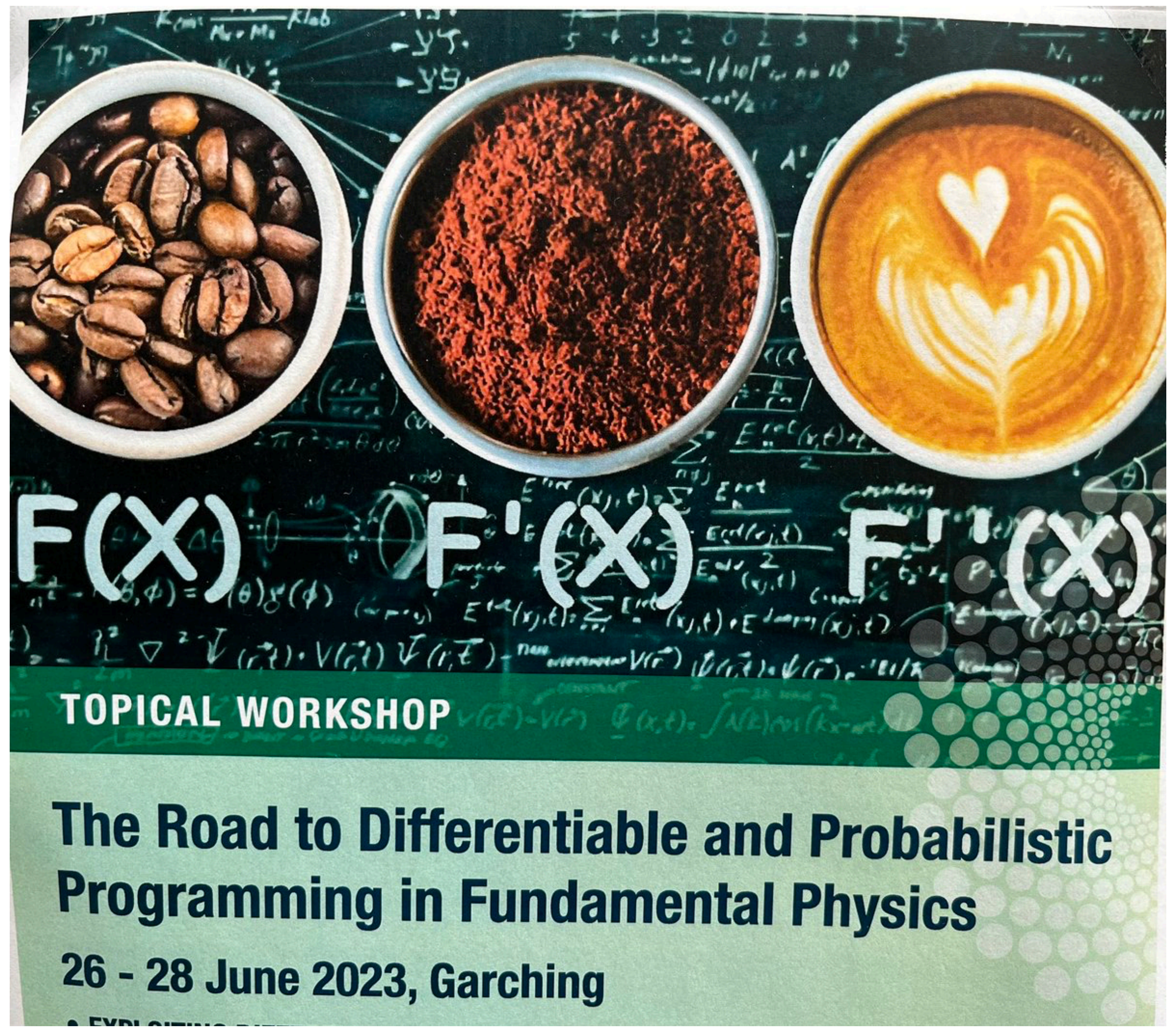
Automatic Differentiation (of course)

The technical solution is becoming clearer: ML lives & dies by automatic gradient estimation / computation.



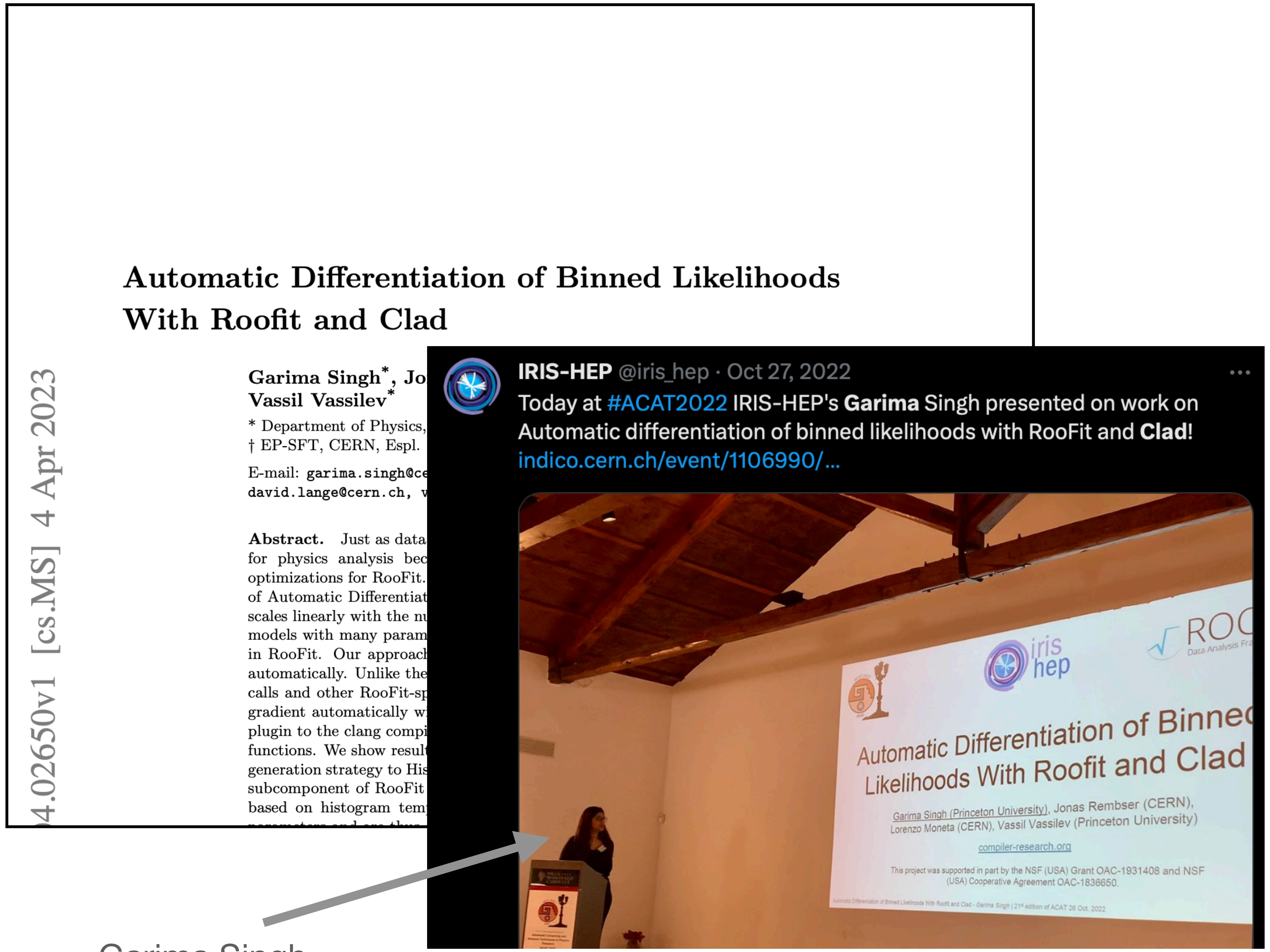
We’re starting to get the experience & students [know about / grow up with] it

Know how to go beyond Python, integrate deep into e.g. C++ or FORTRAN



Technical issue seems solv(ed/able)

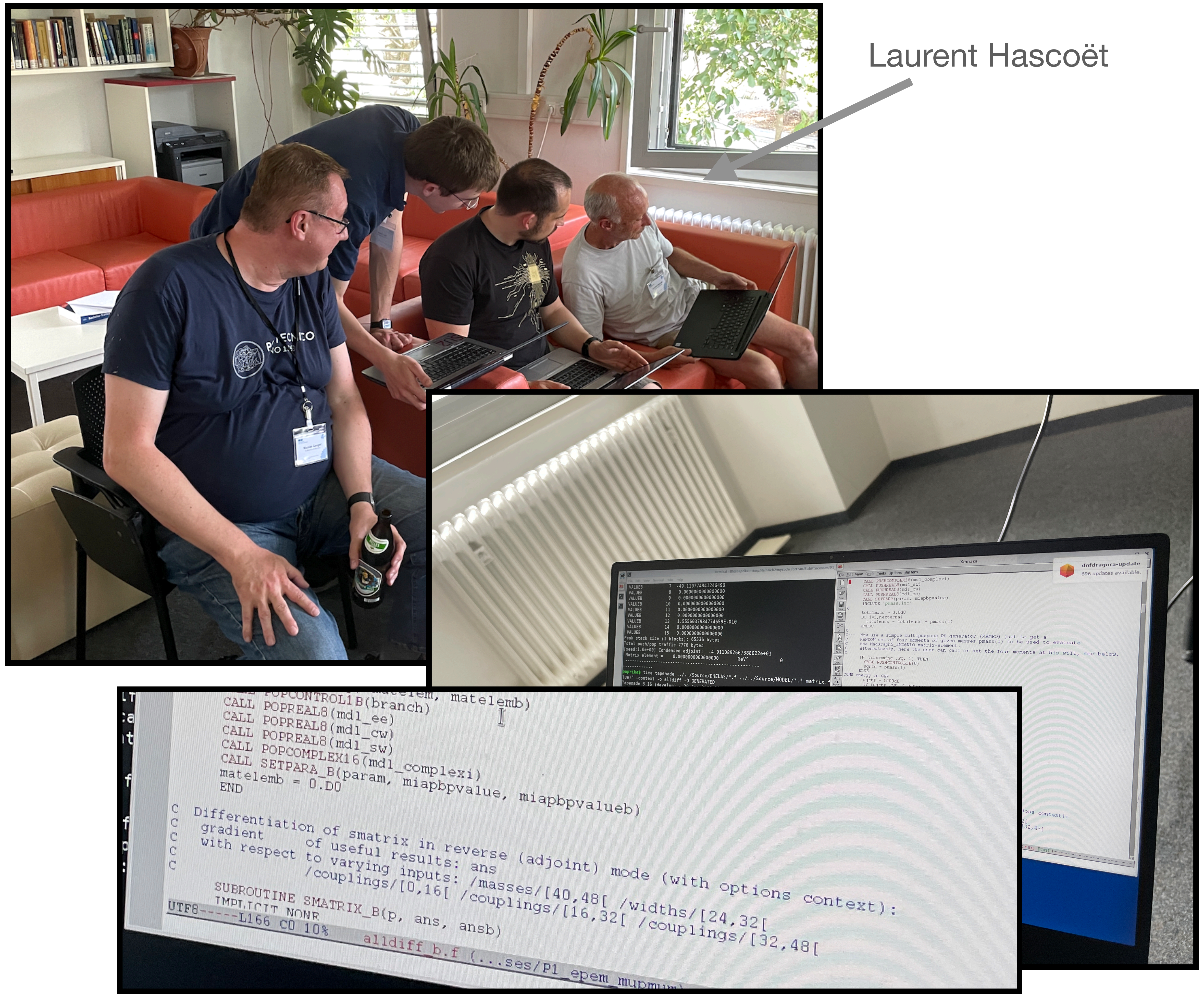
Two Examples



Garima Singh

Differentiating RooFit became a reality

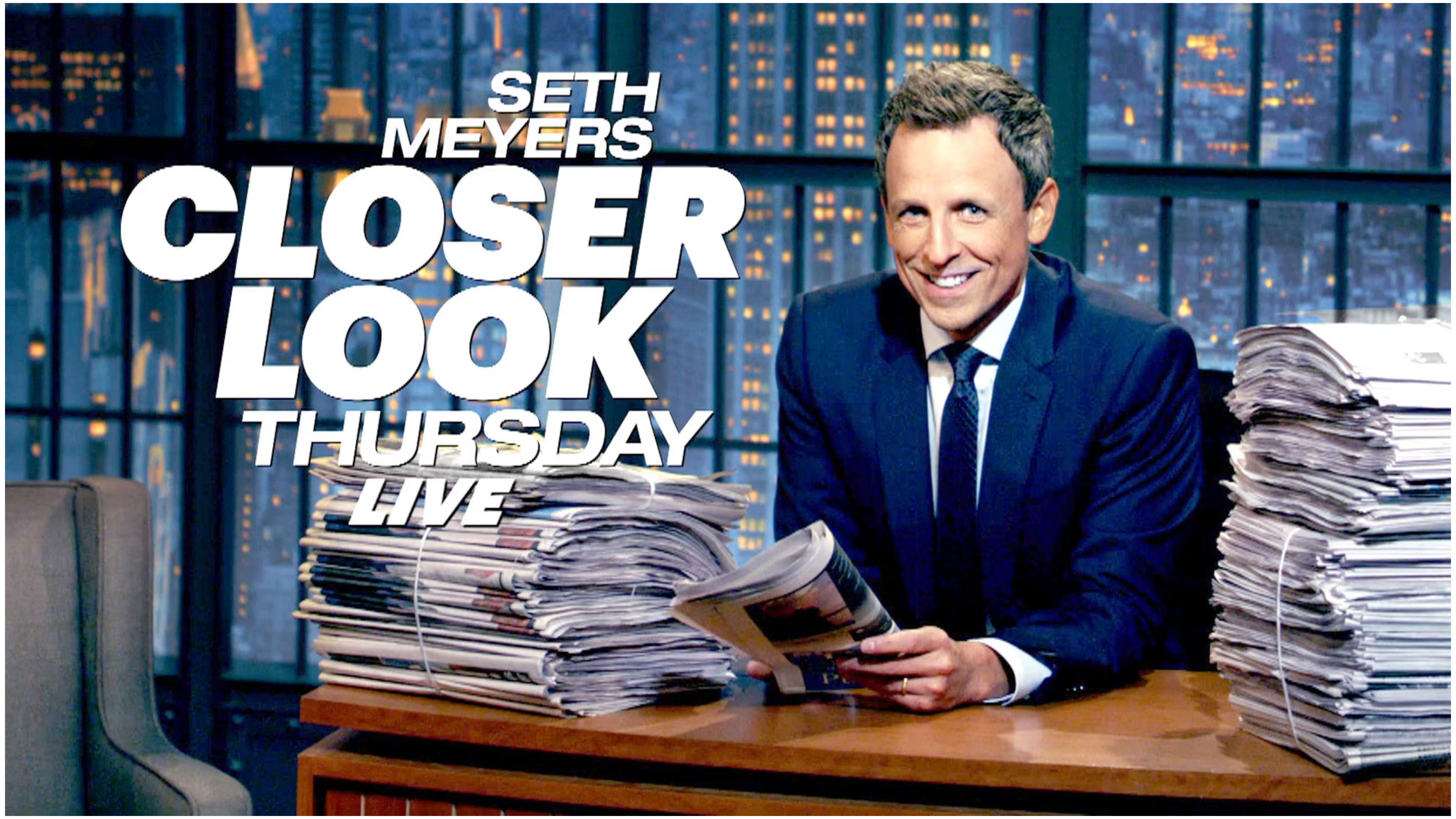
Laurent Hascoët



Differentiating MadGraph FORTRAN

More difficult question

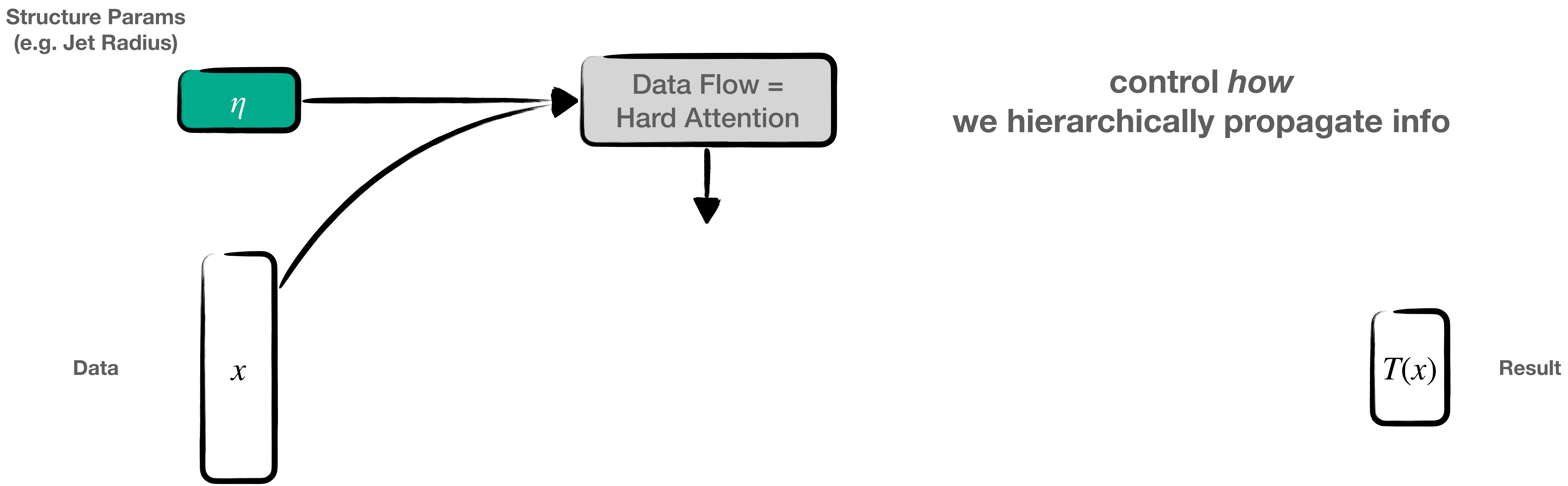
What’s a practical way to get towards deep optimization

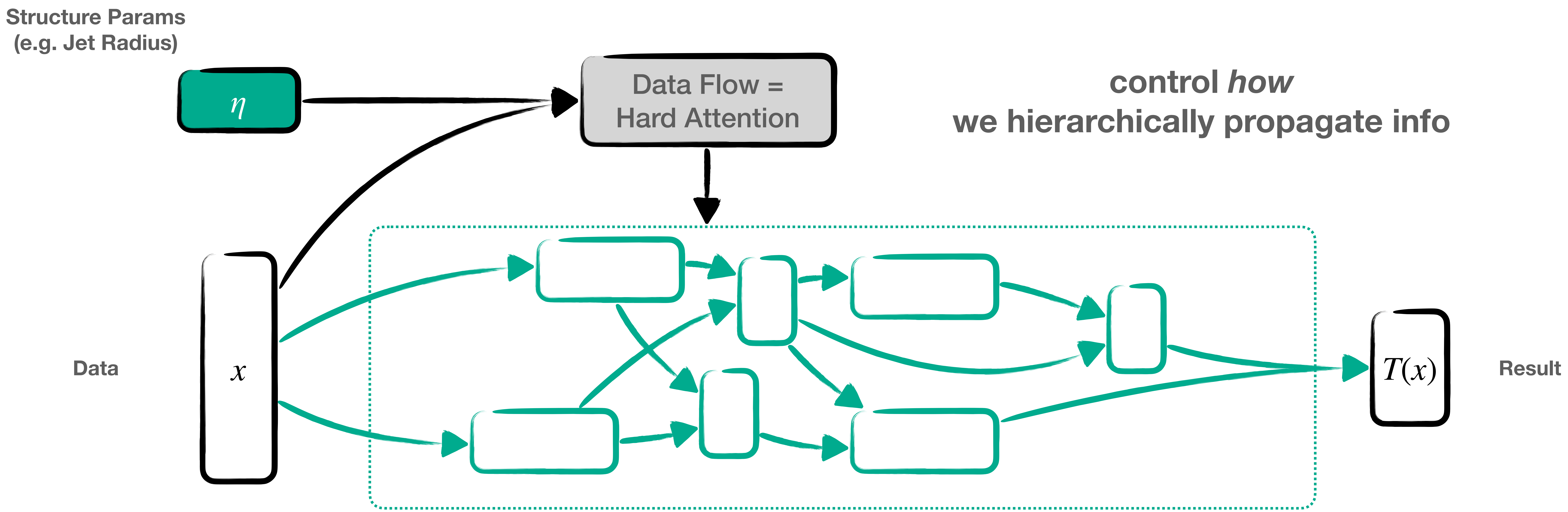


Structure (Hard Attention) vs Representation

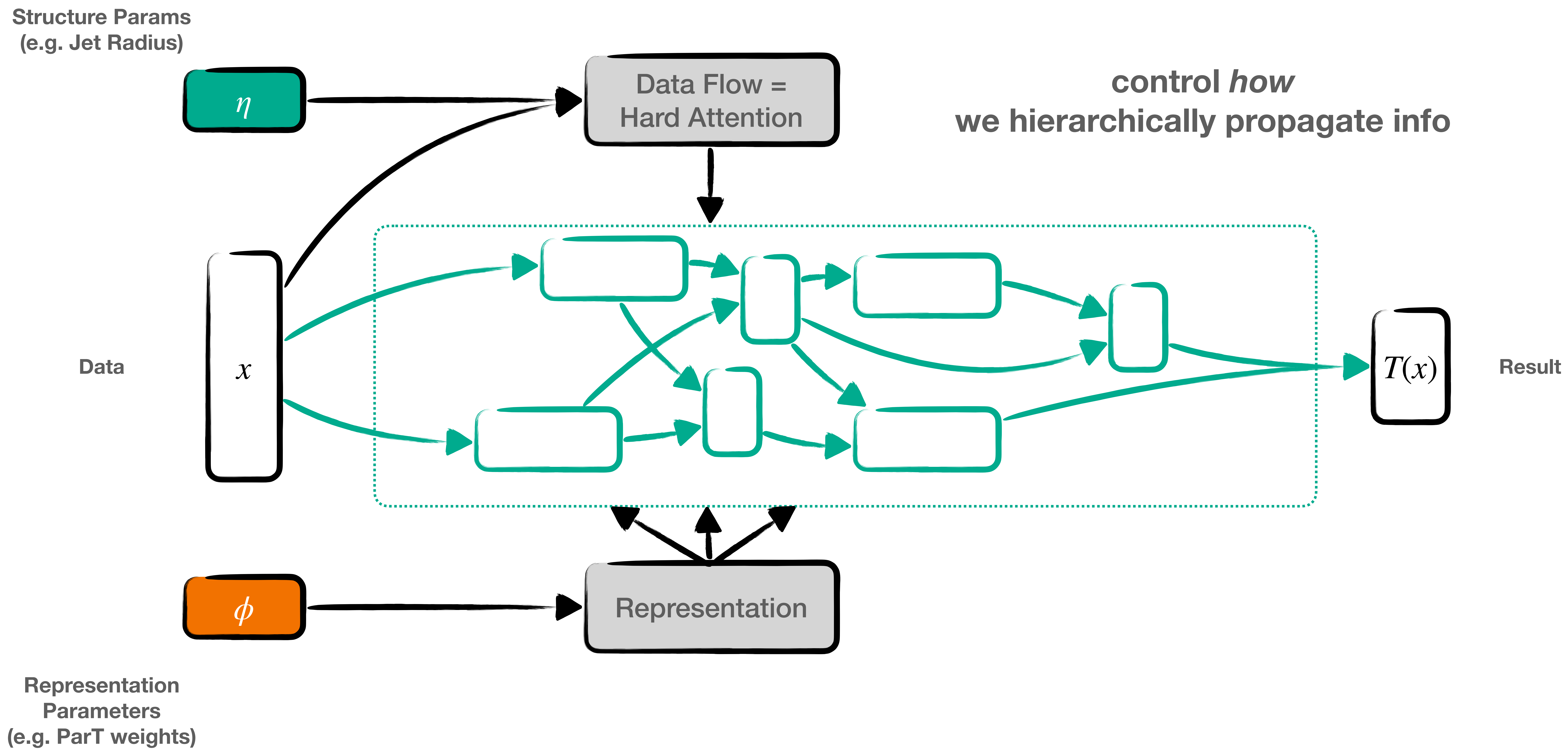
In our Backbone (and Analysis) there is usually a fairly well-defined split between structure-defining operations & representation

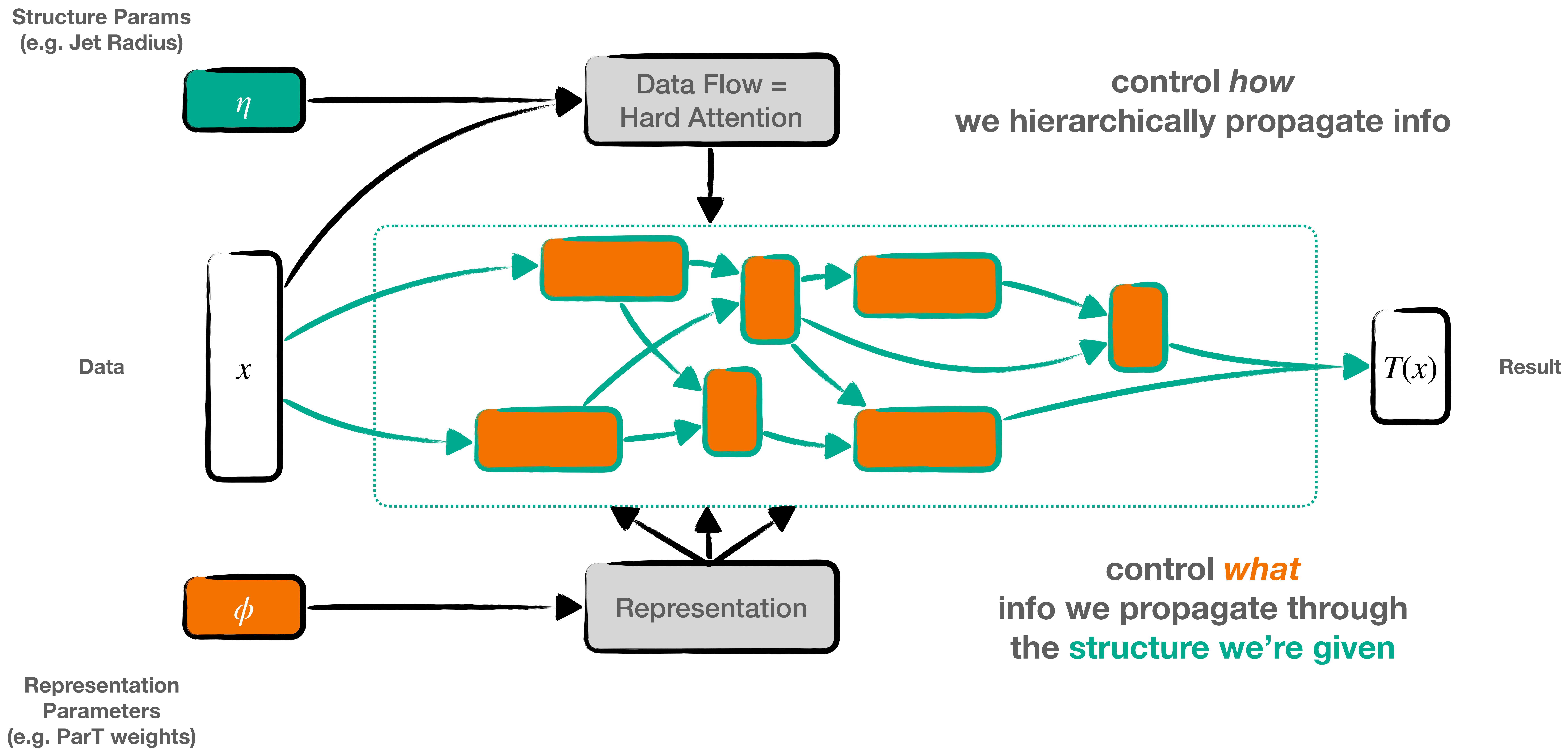
Structure	Representation
Track Finding	Track Fitting / Params
Topoclusters	Cluster Variables
Element Links (to e.g. Pflow Objects)	Particles Properties
Jet Definition	Jet Observables
Resonance System	χ^2, invariant mass, etc..


Structure Params
(e.g. Jet Radius)
η
Data Flow =
Hard Attention
control *how*
we hierarchically propagate info
Data
x
$T(x)$
Result



Structure Params
(e.g. Jet Radius)
η
Data Flow =
Hard Attention
control *how*
we hierarchically propagate info
Data
x
$T(x)$
Result

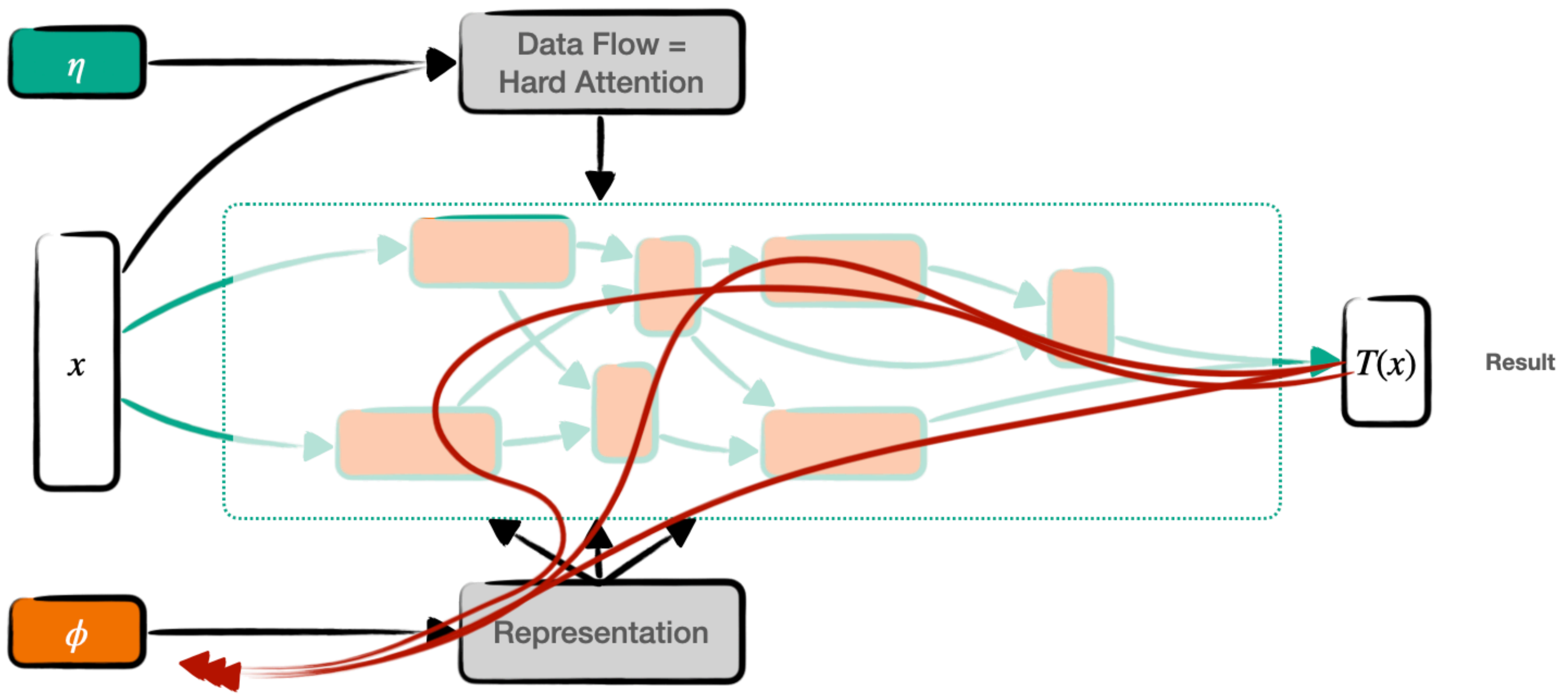

Structure Params
(e.g. Jet Radius)
η
Data Flow =
Hard Attention
control *how*
we hierarchically propagate info
Data
x
$T(x)$
Result
ϕ
Representation
Representation
Parameters
(e.g. ParT weights)


Structure Params
(e.g. Jet Radius)
η
Data Flow =
Hard Attention
control *how*
we hierarchically propagate info
Data
x
$T(x)$
Result
ϕ
Representation
control *what*
info we propagate through
the structure we're given
Representation
Parameters
(e.g. ParT weights)

Two ways to make progress

We should already be able to optimize what information we pass through the structure by standard backprop.

End-to-end Optimize Representations



Two ways to make progress

- We can coarse grain the structure of the data processing.
  Fewer but bigger tasks (possibly solved by ML).

- Go to bigger (learned / latent) representations at each step, instead of hand-picked observables per object

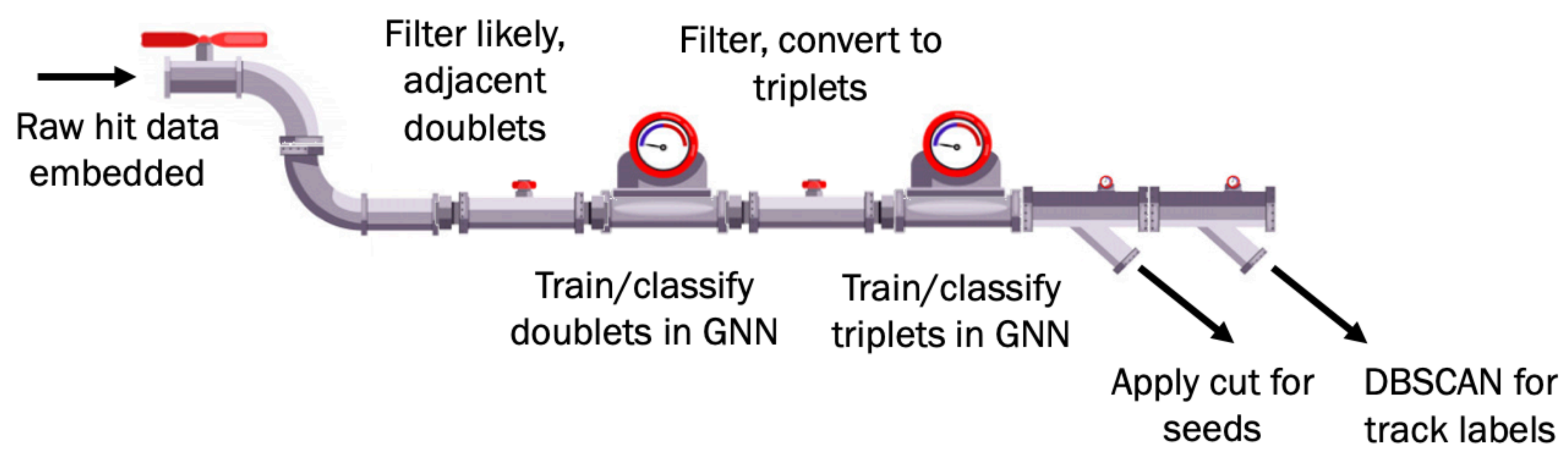
With <u>data flow fixed</u> we can at least still optimize representations

(defer diffing through hard structure for now)

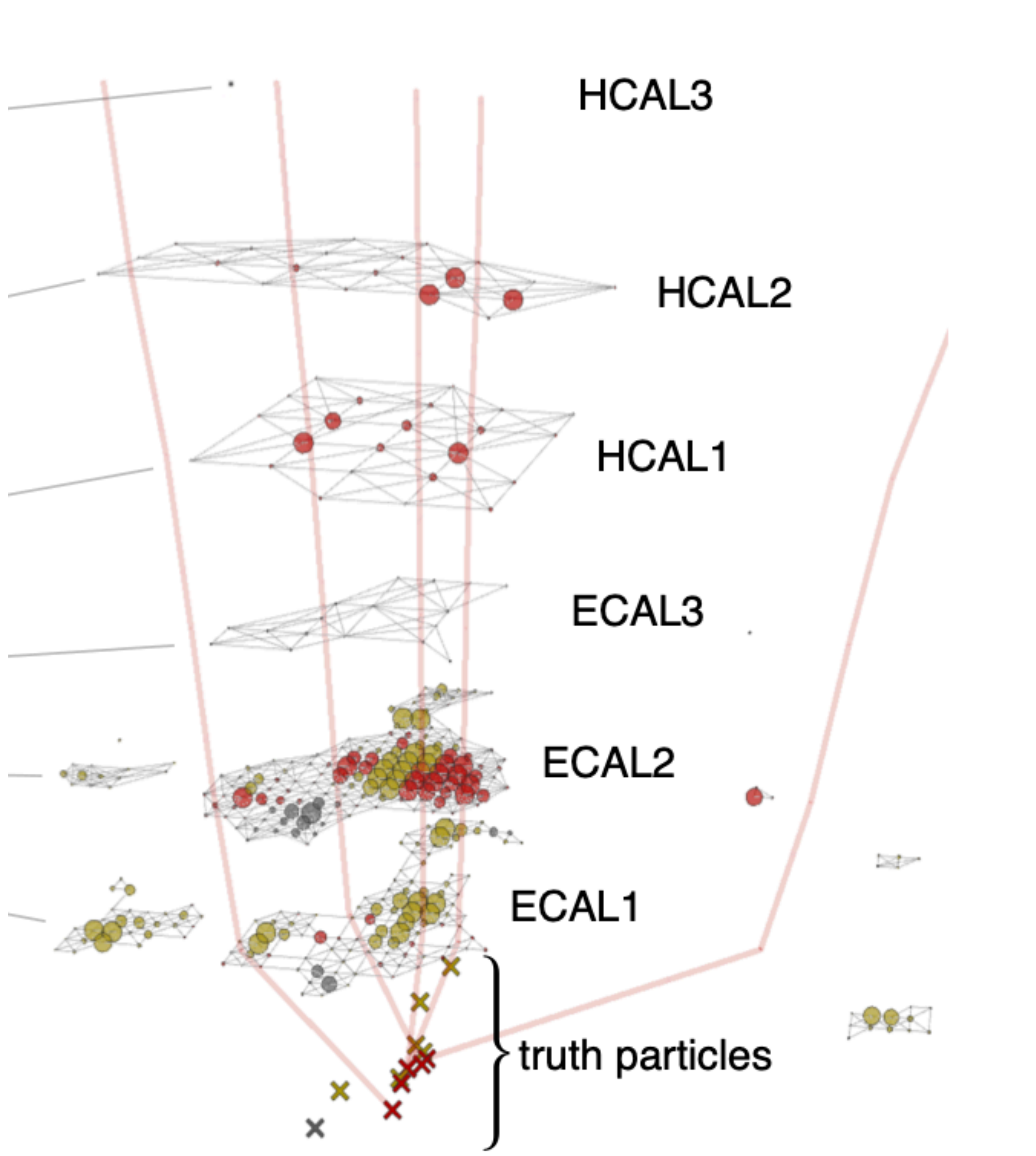
Removing Structure

A trend to bigger (ML) blocks solving more complex tasks and dropping intermediate (helper) representations

arxiv: 2103.06995

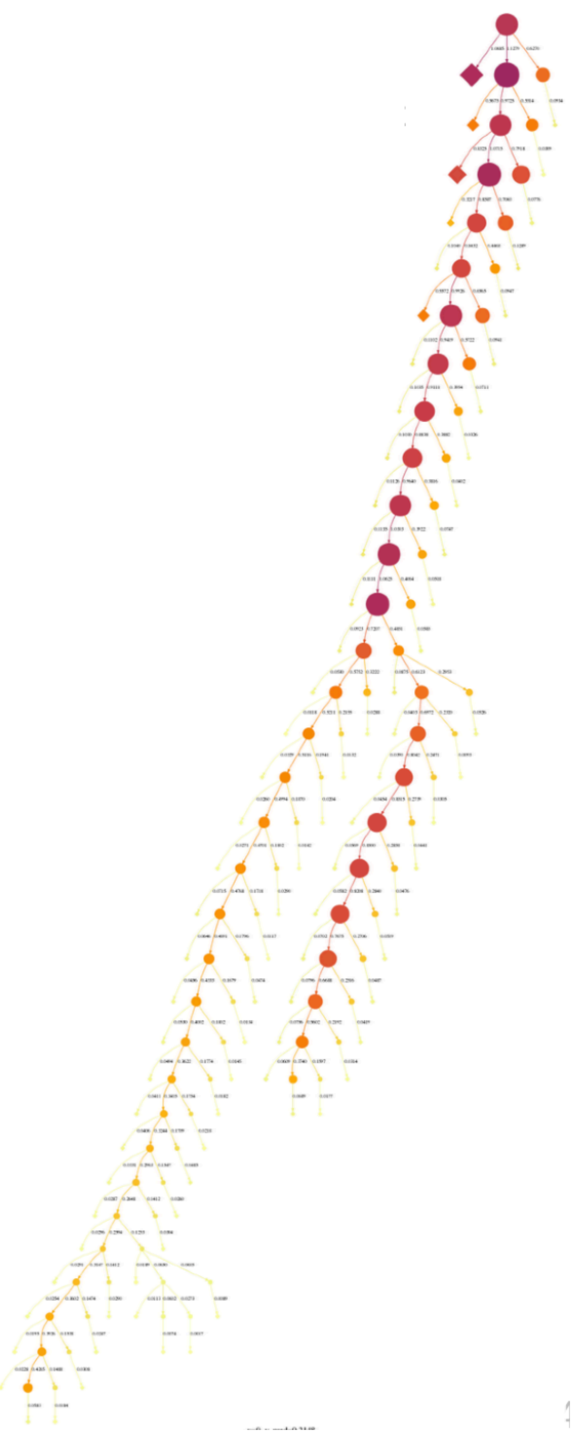


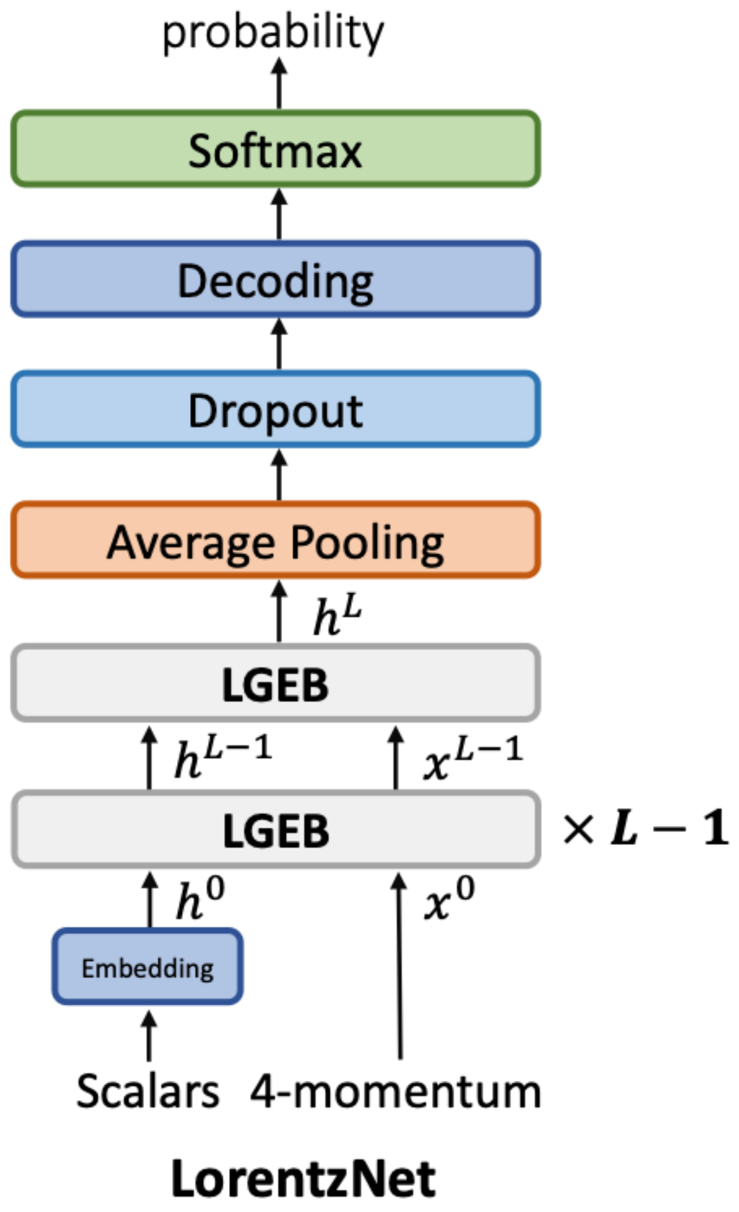
ML Tracking



arxiv: 2302.03583

ML Particle Flow





Jet Transformer vs QCD Aware

SBI vs Differentiable Inference

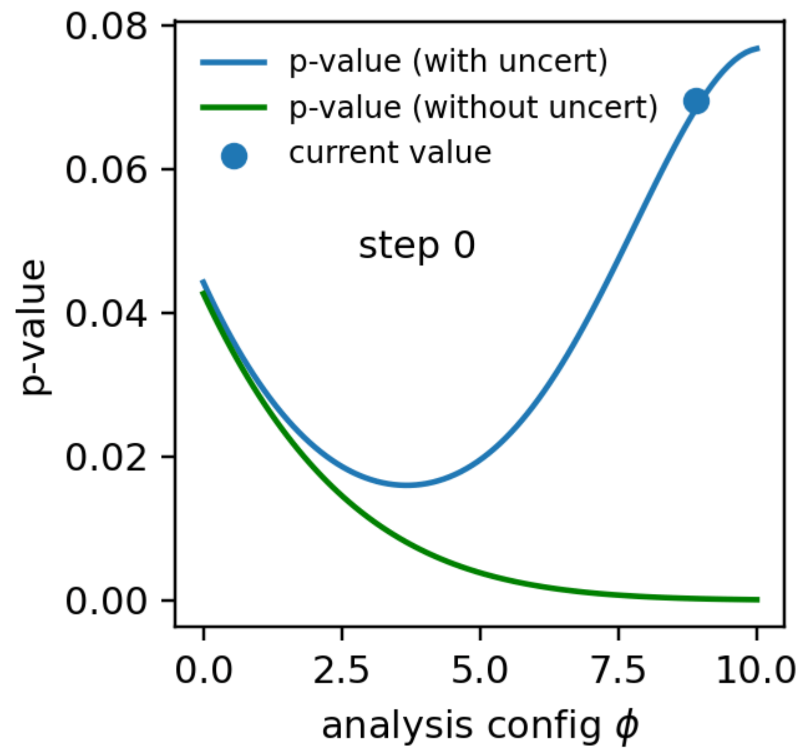
a) work hard to make classic stats differentiable

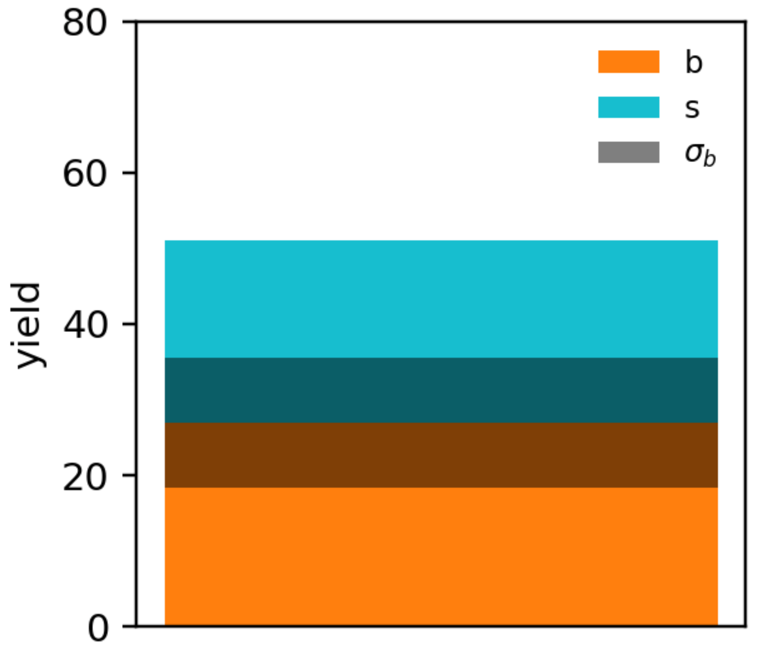
Inputs → Density Estimate → Parametrization → Likelihood Model → Likelihood Ratio → Test Statistic → Result

θ → Parametrization

[Simpson, Heinrich]

[de Castro, Dorigo]

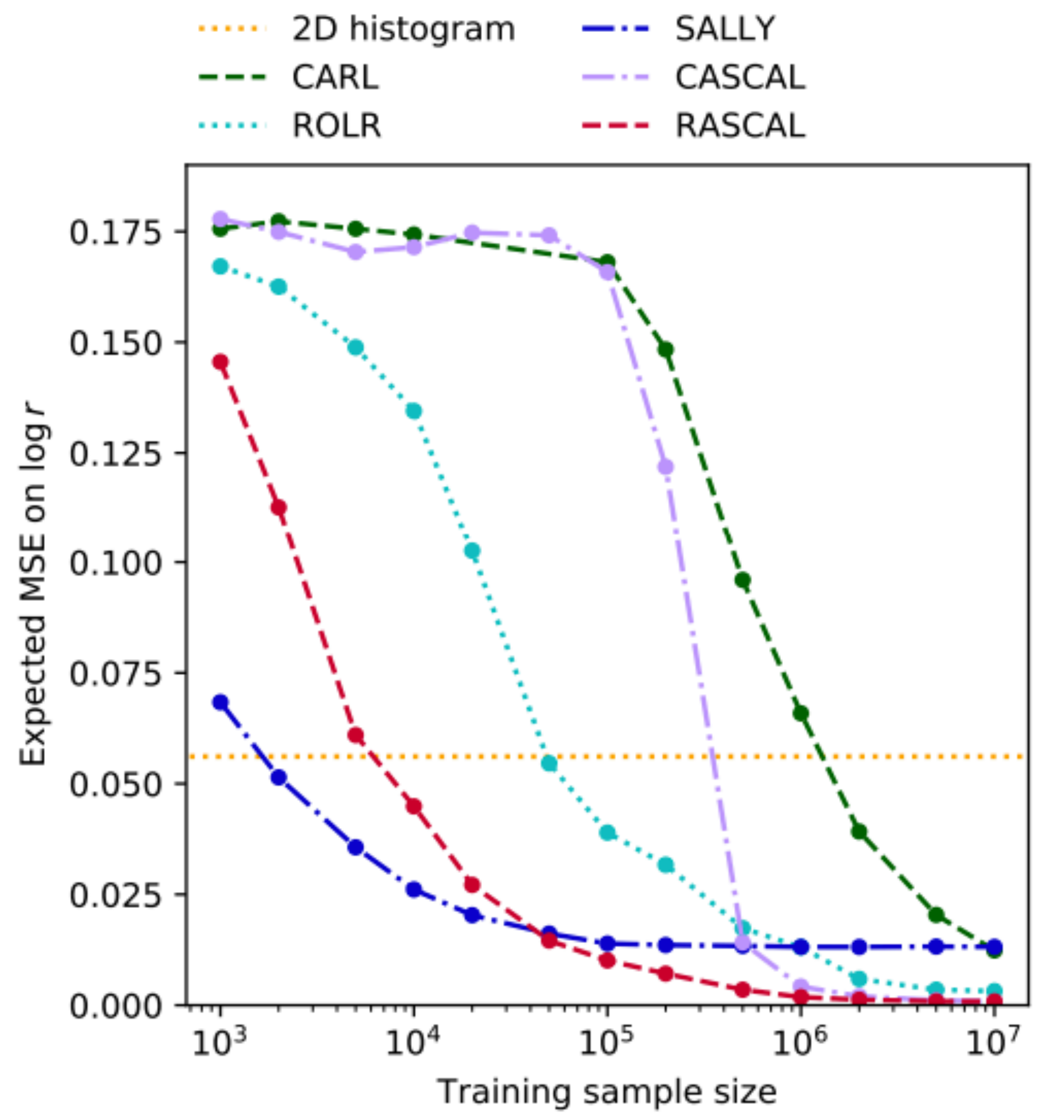




b) replace a lot (or ~all) of it by a clever NN training

Inputs → Modern / Neural SBI 🤯 → Result

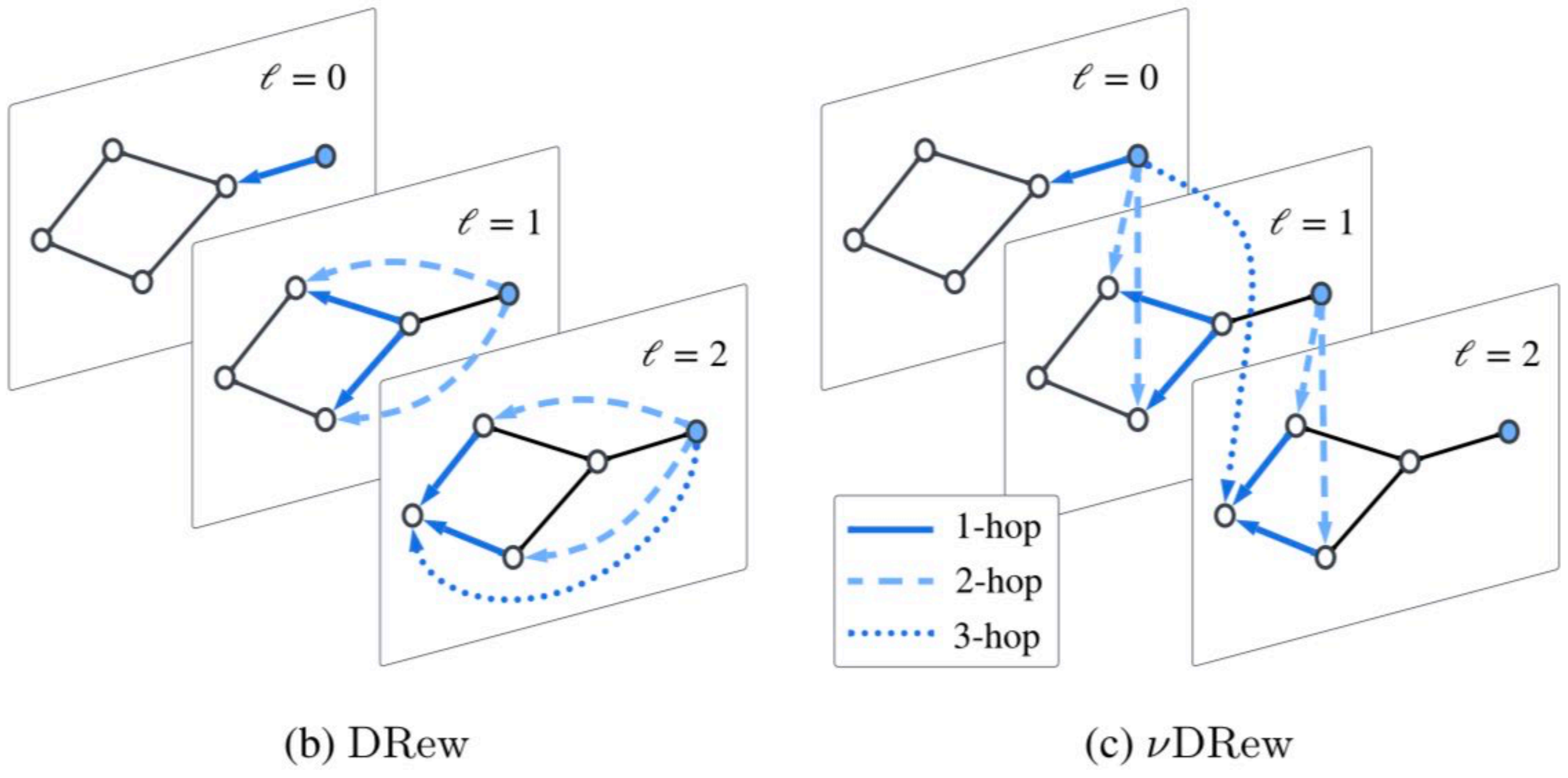
θ → Modern / Neural SBI



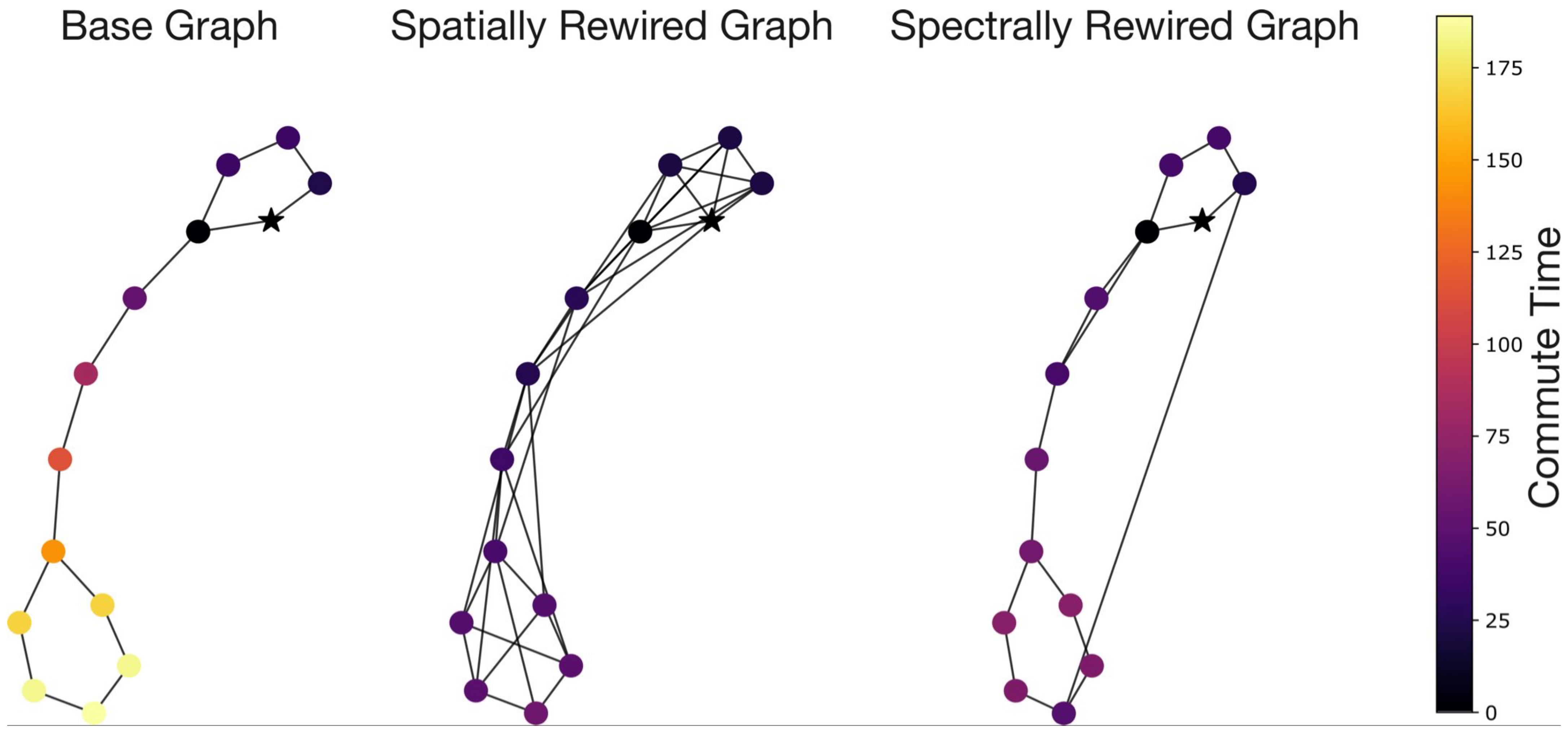
[Brehmer, Cranmer, +], [Louppe+++], [Weniger++], [LH, Mishra-Sharma, Windischhofer, Pollard] etc etc

Less Structure = Better Learning Dynamics?

Inductive Bias is not always a blessing. Information Flow must also be efficient & training dynamics favorable (see e.g. M. Bronstein's talk)

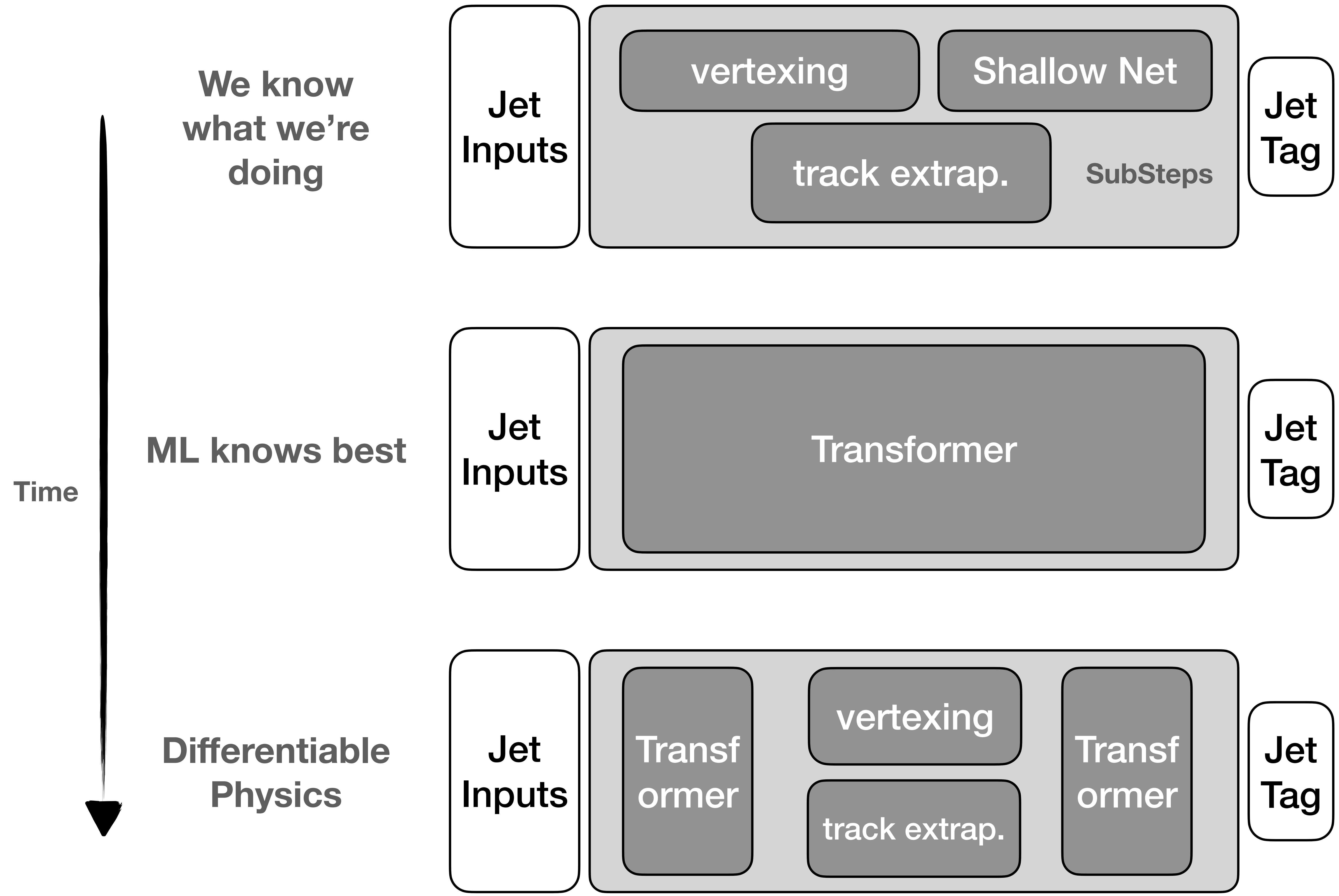


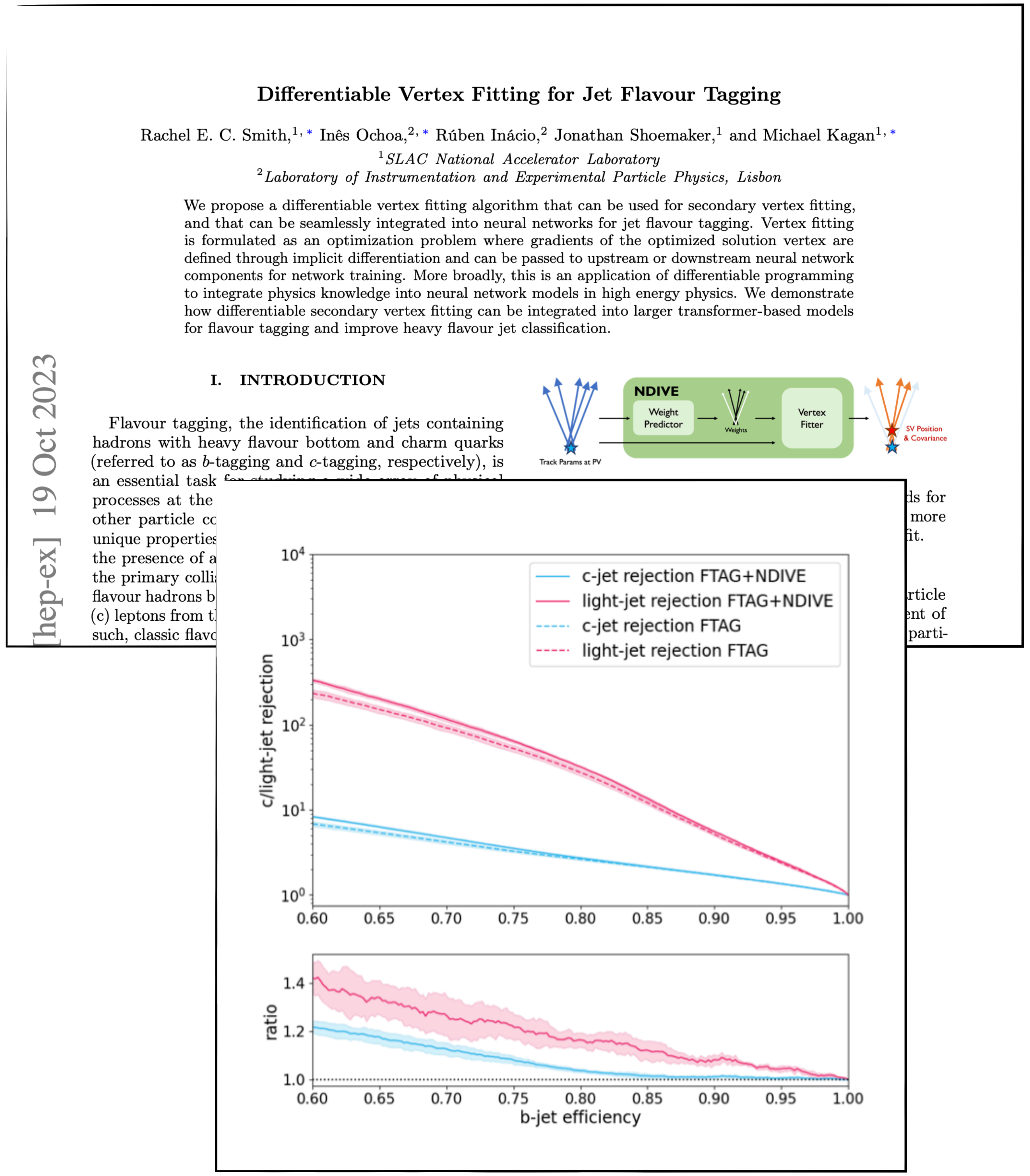
(b) DRew (c) νDRew



[M. Bronstein]

But no Structure isn't the Solution Either





Differentiable Vertex Fitting for Jet Flavour Tagging

Rachel E. C. Smith,[1,*] Inês Ochoa,[2,*] Rúben Inácio,[2] Jonathan Shoemaker,[1] and Michael Kagan[1,*]
[1]*SLAC National Accelerator Laboratory*
[2]*Laboratory of Instrumentation and Experimental Particle Physics, Lisbon*

We propose a differentiable vertex fitting algorithm that can be used for secondary vertex fitting, and that can be seamlessly integrated into neural networks for jet flavour tagging. Vertex fitting is formulated as an optimization problem where gradients of the optimized solution vertex are defined through implicit differentiation and can be passed to upstream or downstream neural network components for network training. More broadly, this is an application of differentiable programming to integrate physics knowledge into neural network models in high energy physics. We demonstrate how differentiable secondary vertex fitting can be integrated into larger transformer-based models for flavour tagging and improve heavy flavour jet classification.

[hep-ex] 19 Oct 2023

I. INTRODUCTION

Flavour tagging, the identification of jets containing hadrons with heavy flavour bottom and charm quarks (referred to as *b*-tagging and *c*-tagging, respectively), is an essential task ...

[M Kagan, R. Smith, I. Ochoa et al]

So what gives?

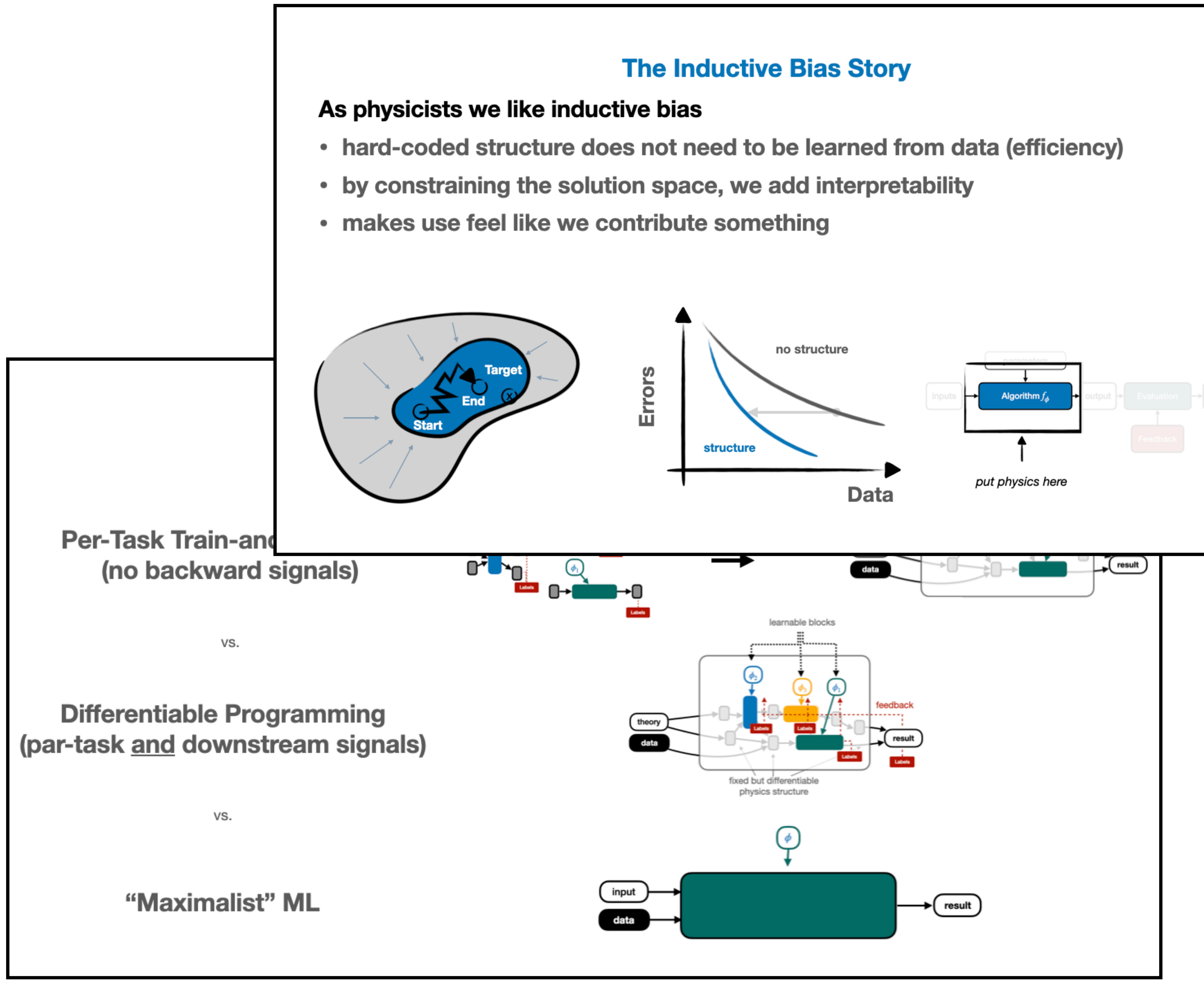
Questions from last H&N

Few Q’s from Last Year’s Discussion on Differentiable Programming

How much structure is important?

Are auxiliary tasks important or just optimize end to end?

What’s there to gain if we do end to end optimization?



A toy end-to-end Analysis

Nicole Hartman

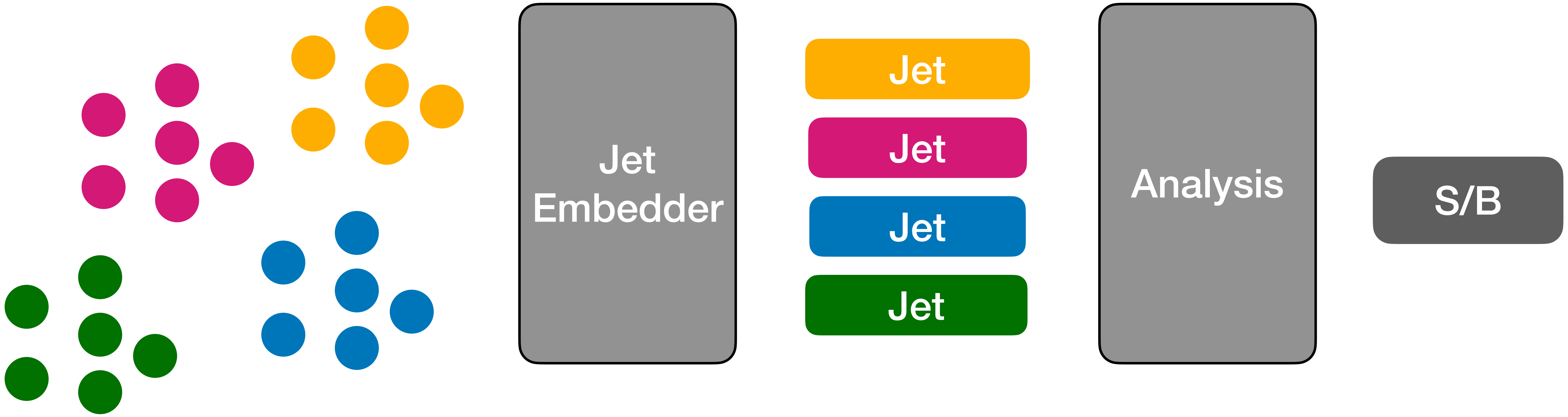
Matthias Vigl

Test-Setup: X → HH → 4b. Final state with Jets.

Q: could we just train from scrach? Does pretraining matter?

Q: Is finetuning a la modern ML worth it?

Q: do we see benefits of scale & adjacent pretraining tasks?

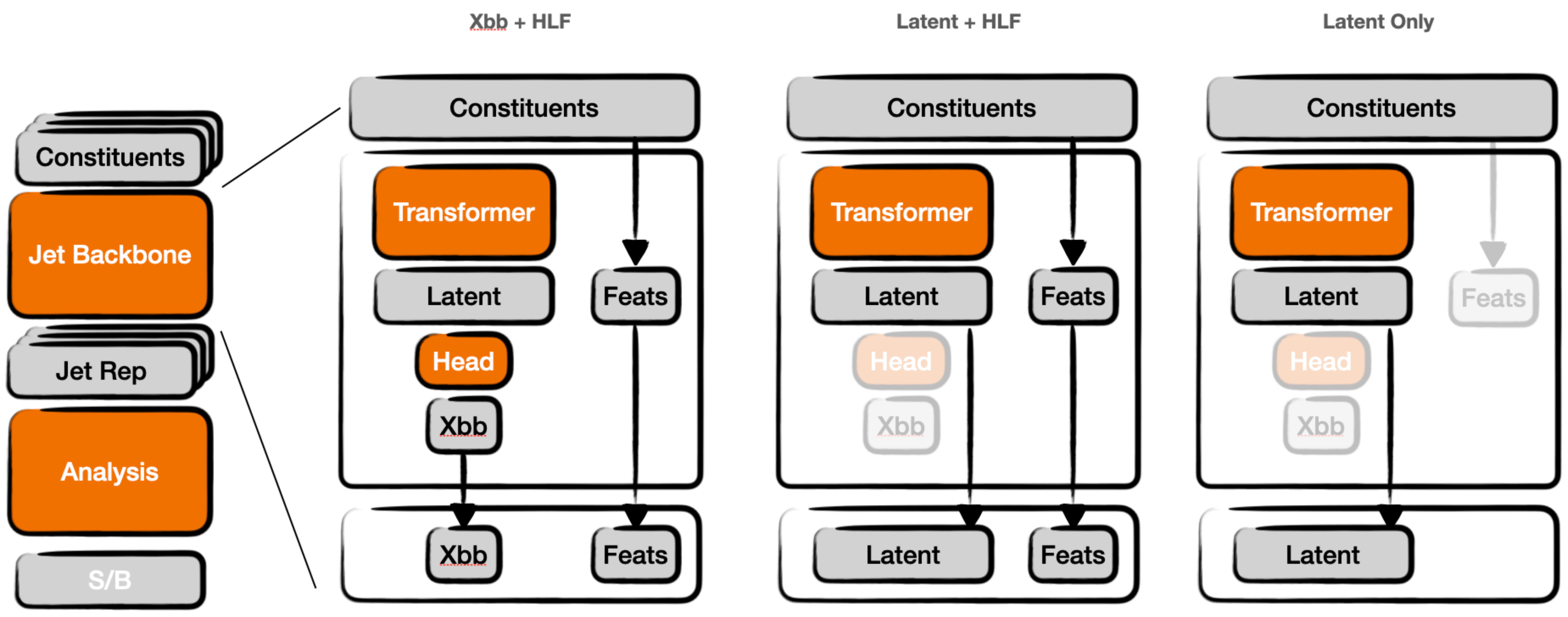


A small Experiment in End2End Optimization

“Foundation model”: Particle Transformer

“Analysis”: simple DeepSet + binary classification

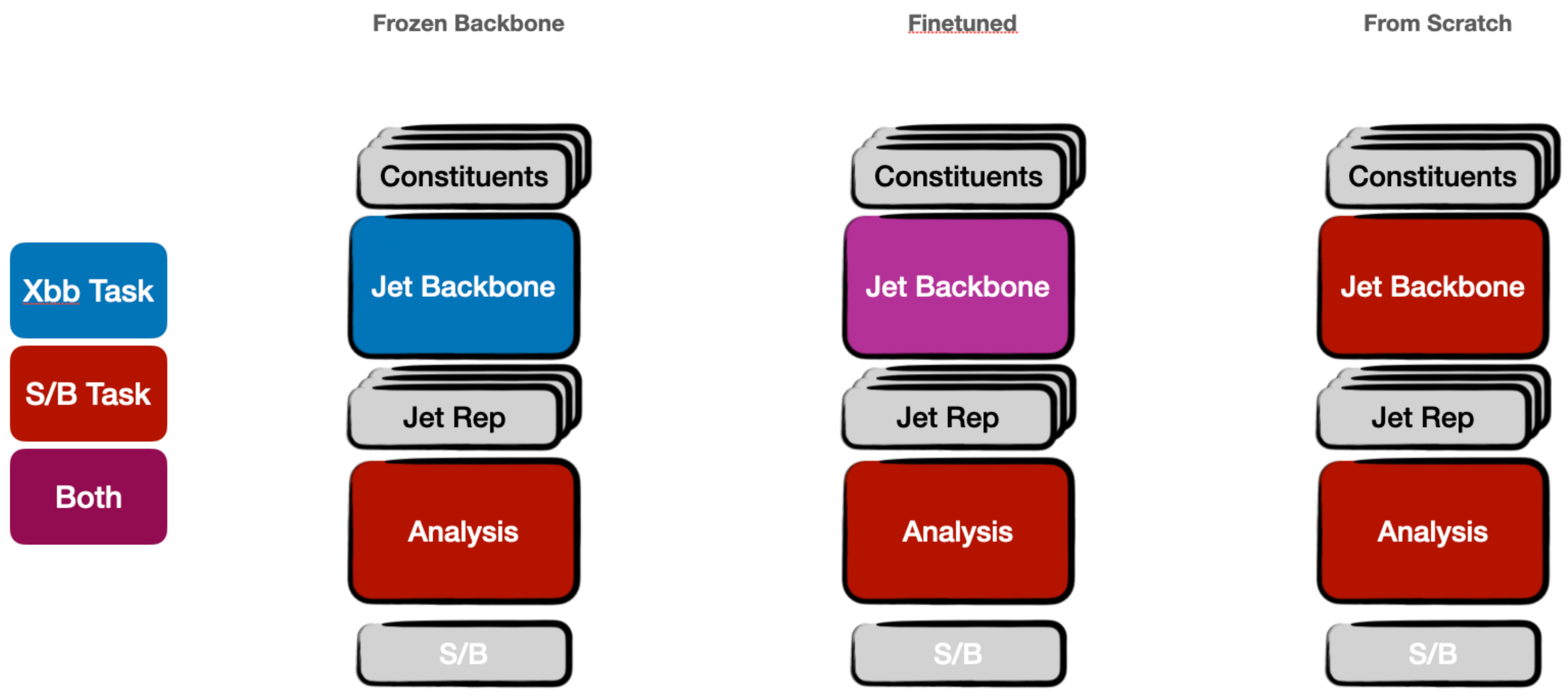
Various options on size of communication channel



A small Experiment in End2End Optimization

Three training setups:

- pretrained on Xbb then **frozen**
- pretrained on Xbb and then **finetuned** on di-Higgs resonanc
- **from scratch**: random ParT



Two Directions

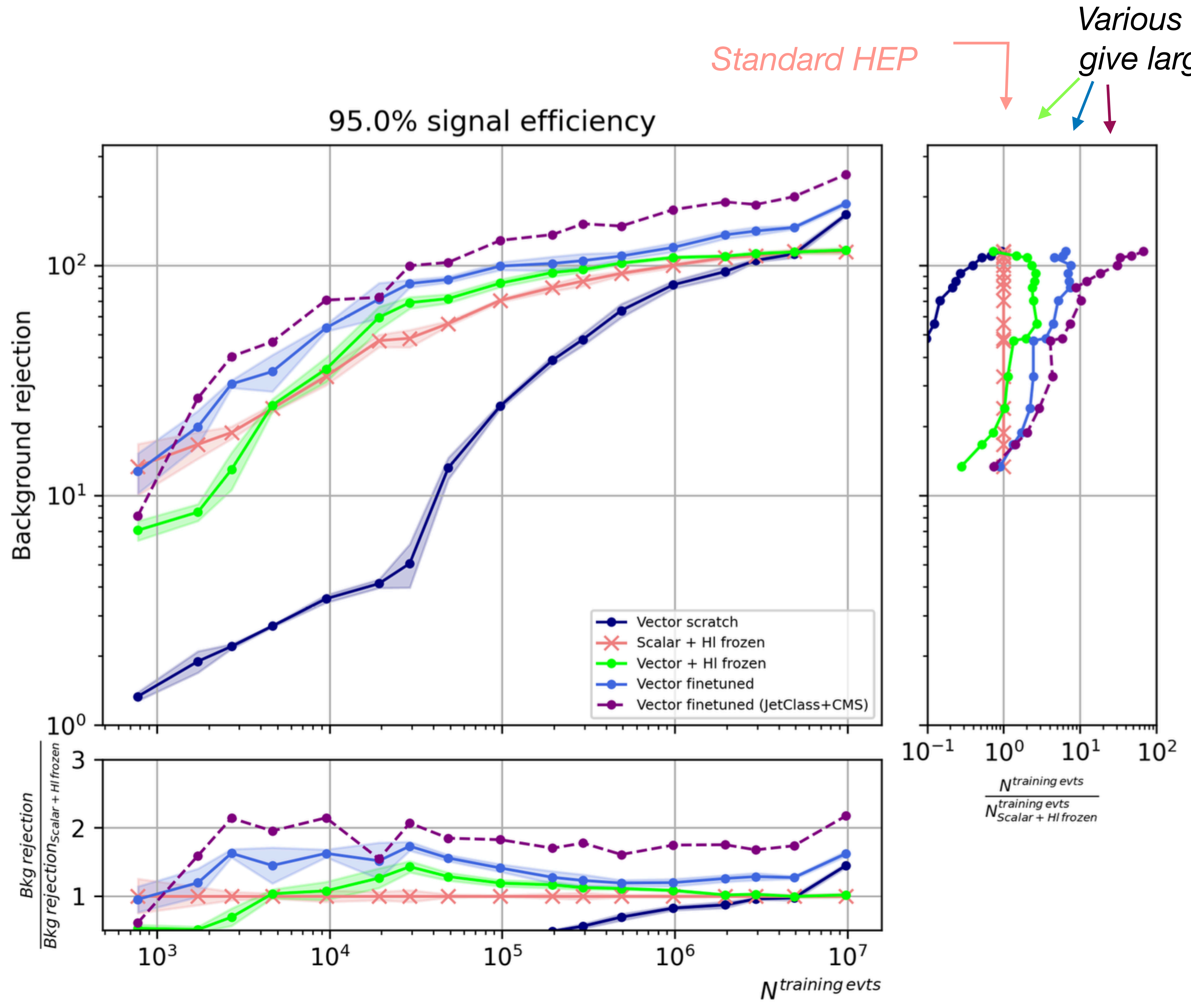
Structural Autonomy →

Representational Autonomy ↓

Training \ Architecture	Xbb + HLF	Latent + HLF	Latent Only
Frozen	**Standard HEP**		**Hope for a Sufficient Stat**
Finetuned		**ML-assisted HEP**	
From Scratch	**Inductive Bias is all you need**		**“Hits to Higgs”**

Interesting Results

Well-known patterns from ML seem to hold also in HEP



- **Pretraining (20M jets) helps and pretraining more (100M jets) helps more.**
- **Finetuning for Analysis extracts more info than just pretrained features**
- **higher-dim embeddings are better**
- **pretrain + finetune = 1000x over scratch "few" shot models**
- **(from scratch training works it's just slow)**

What about the Discrete / Hard Structure ?

Given fixed objects we can see that both directions help

- higher dimensional embeddings
- smart end-to-end training (i.e. pretraining + finetuning)

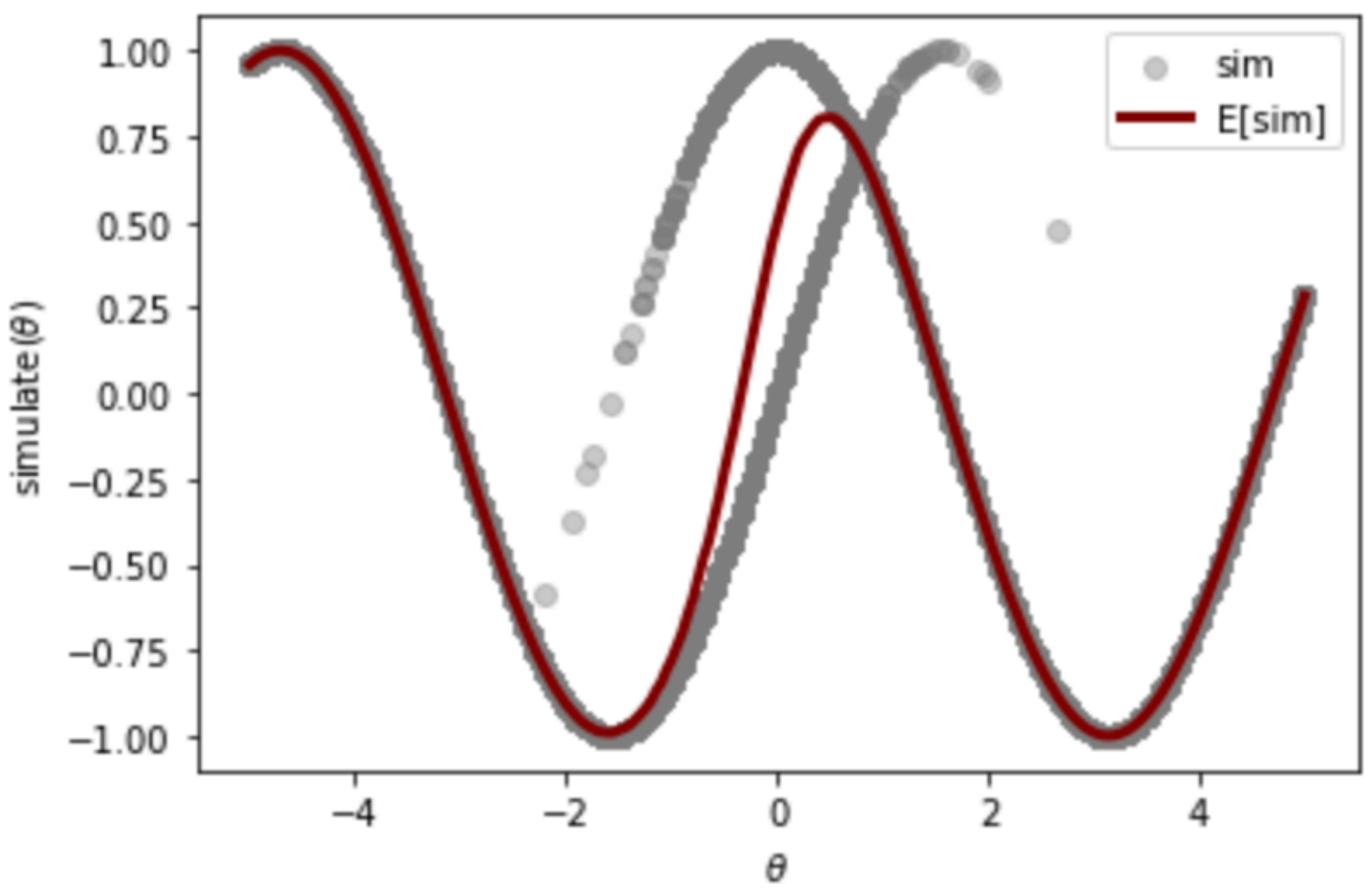
What about gradients for discrete structures?

Discrete Randomness

Differentiating discrete structures is easiest if it’s **discrete and probabilistic** → **smooth expectation value**

```
def simulate(theta):
    p = sigmoid(theta)
    x = bernoulli(p) #0 or 1
    if x == 0:
        eval = sin(theta)
    else:
        eval = cos(theta)
    return eval
```

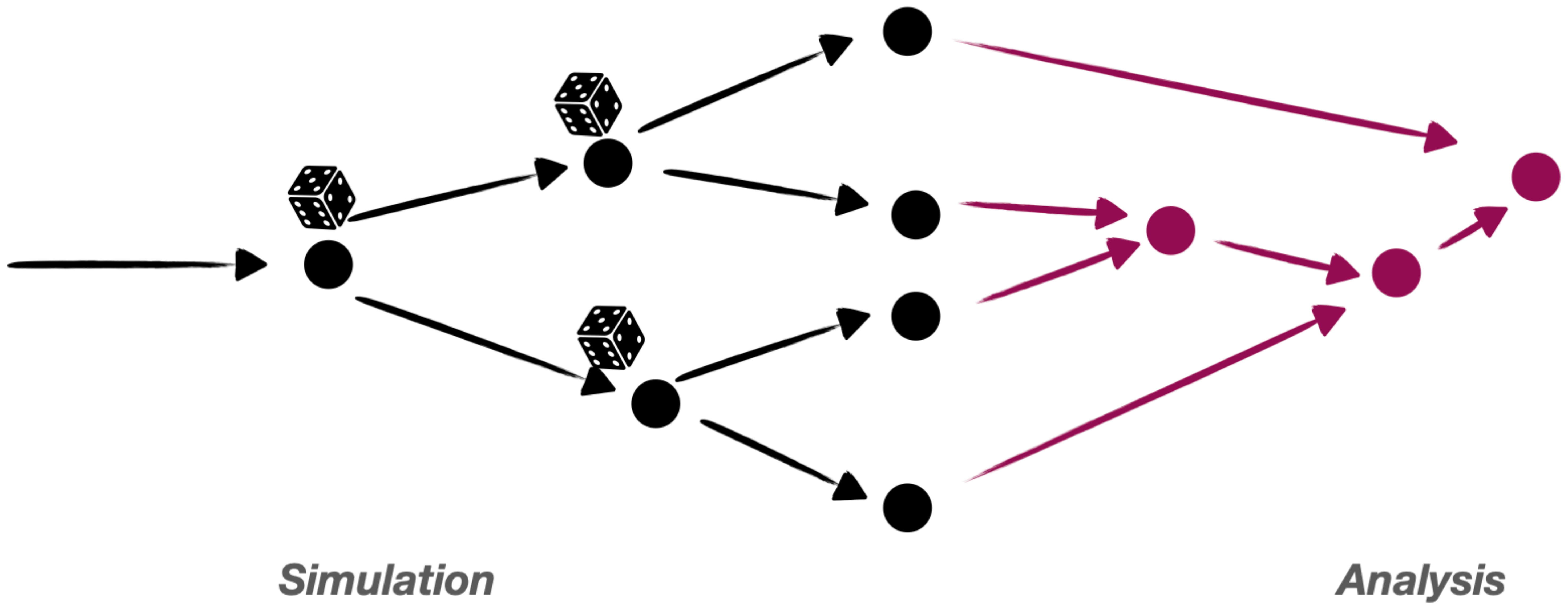
Discrete Jumps



Smooth Expectation Value

Natural Test Case: Differentiating Particle Showers

Stochastic Branching: the reason for the **ubiquitous clustering we've seen during inference**

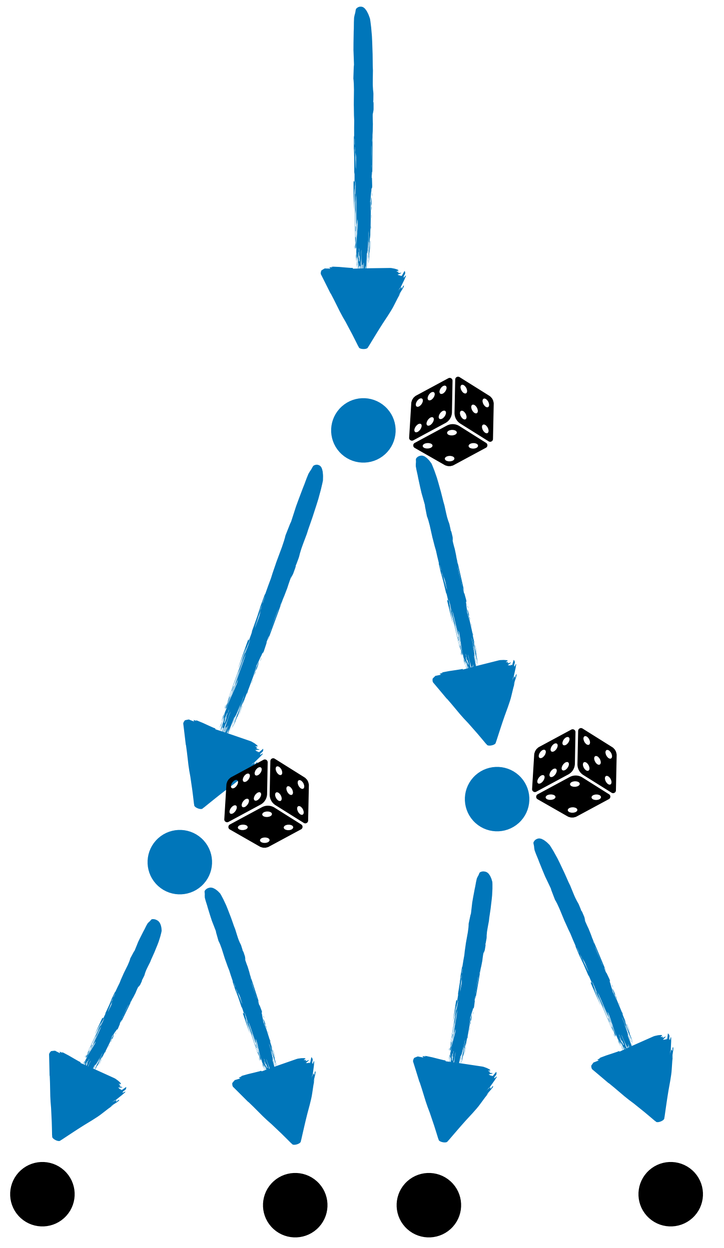


Differentiating through Particle Showers

The best known algorithm for gradient estimation is used a lot in *Reinforcement Learning - score function estimation*

$$\nabla \bar{f}(\phi) = \mathbb{E}_{p_\phi}[f(x)\,\nabla_\phi \log p_\phi(x)]$$

requires tracking probabilities (and their gradients) while running code → probabilistic programming



HEP Simulator: discrete processe

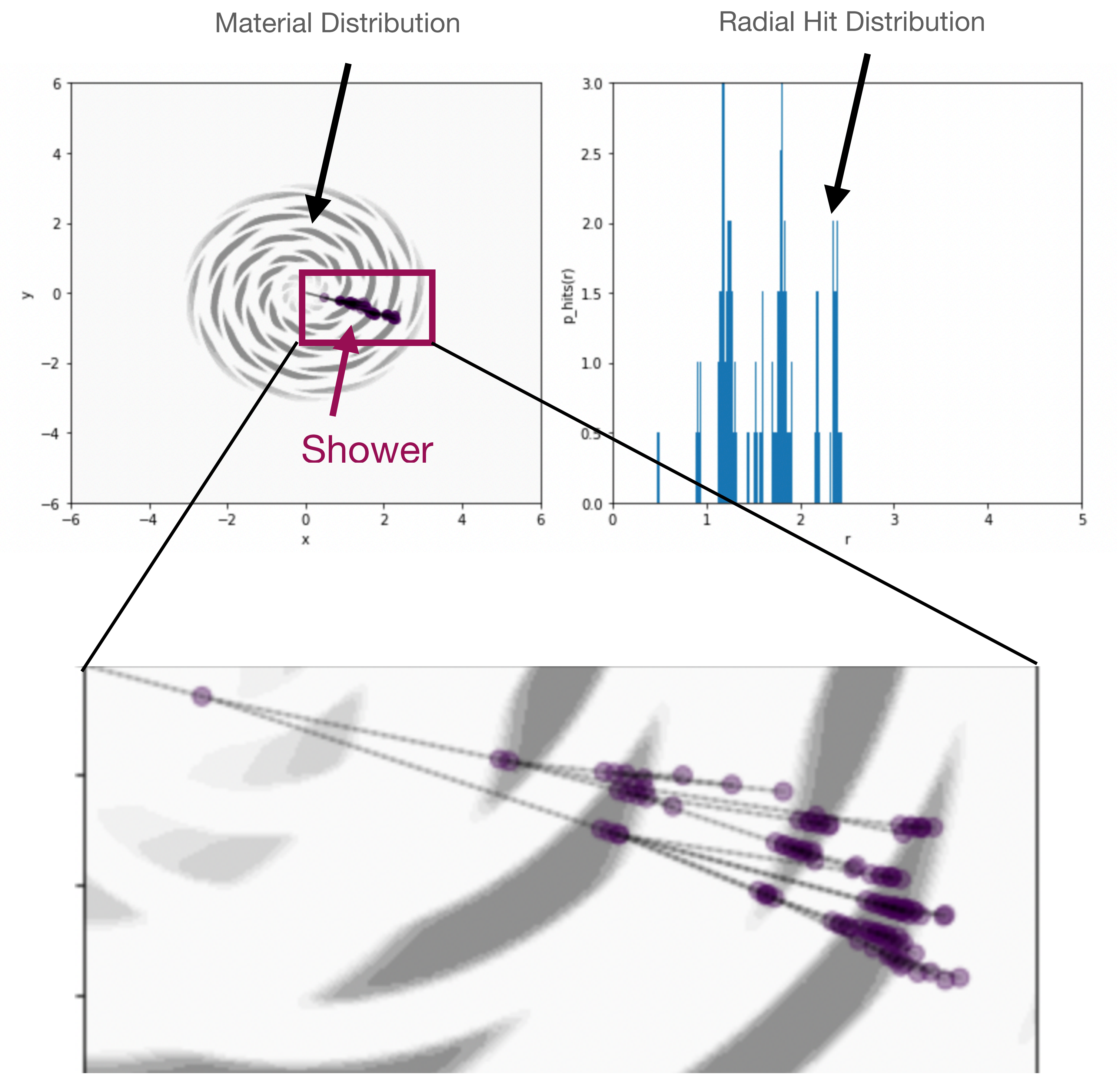
Atari Games discrete actions

Differentiating through Particle Showers

High Density: E-loss and splitting
Low Density: linear propagation

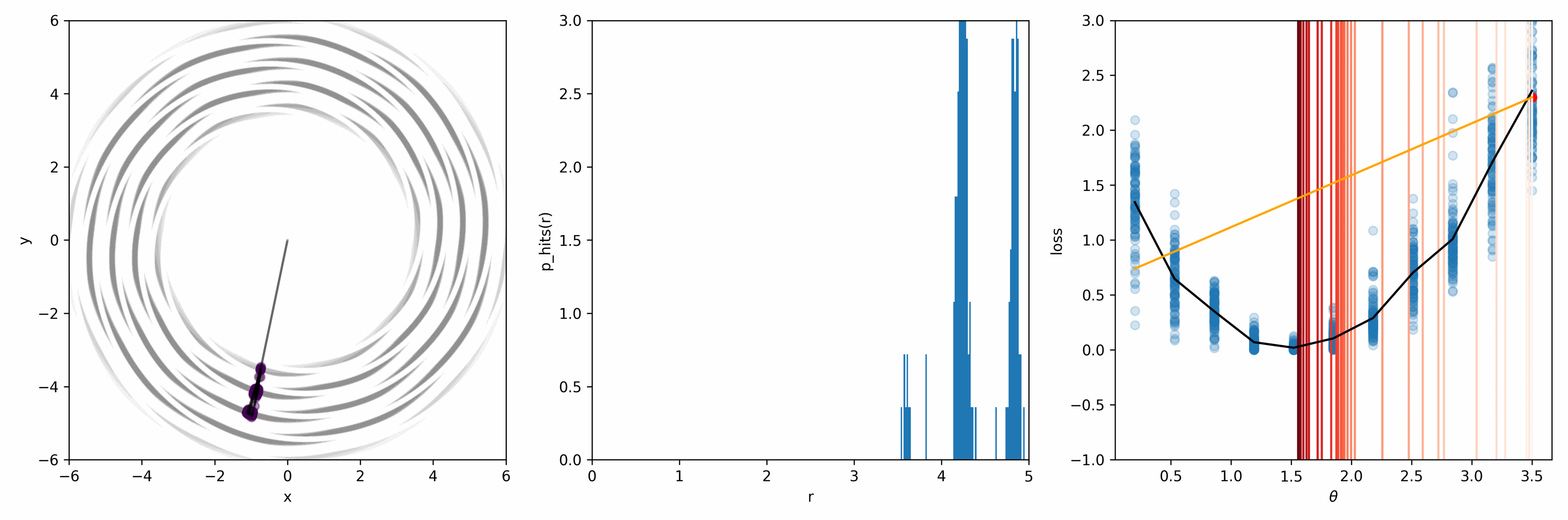
Design Parameter:
Radial Distance of Material

Design Goal: Shower Depth



Differentiating through Particle Showers

It Works! Can optimize layout from post-shower reward!



Branches of a Tree: Taking Derivatives of Programs with Discrete and Branching Randomness in High Energy Physics

Michael Kagan[1,*] and Lukas Heinrich[2,*]

[1] *SLAC National Accelerator Laboratory*
[2] *Technical University of Munich*

We propose to apply several gradient estimation techniques to enable the differentiation of programs with discrete randomness in High Energy Physics. Such programs are common in High Energy Physics due to the presence of branching processes and clustering-based analysis. Thus differentiating such programs can open the way for gradient based optimization in the context of detector design optimization, simulator tuning, or data analysis and reconstruction optimization. We discuss several possible gradient estimation strategies, including the recent Stochastic AD method, and compare them in simplified detector design experiments. In doing so we develop, to the best of our knowledge, the first fully differentiable branching program.

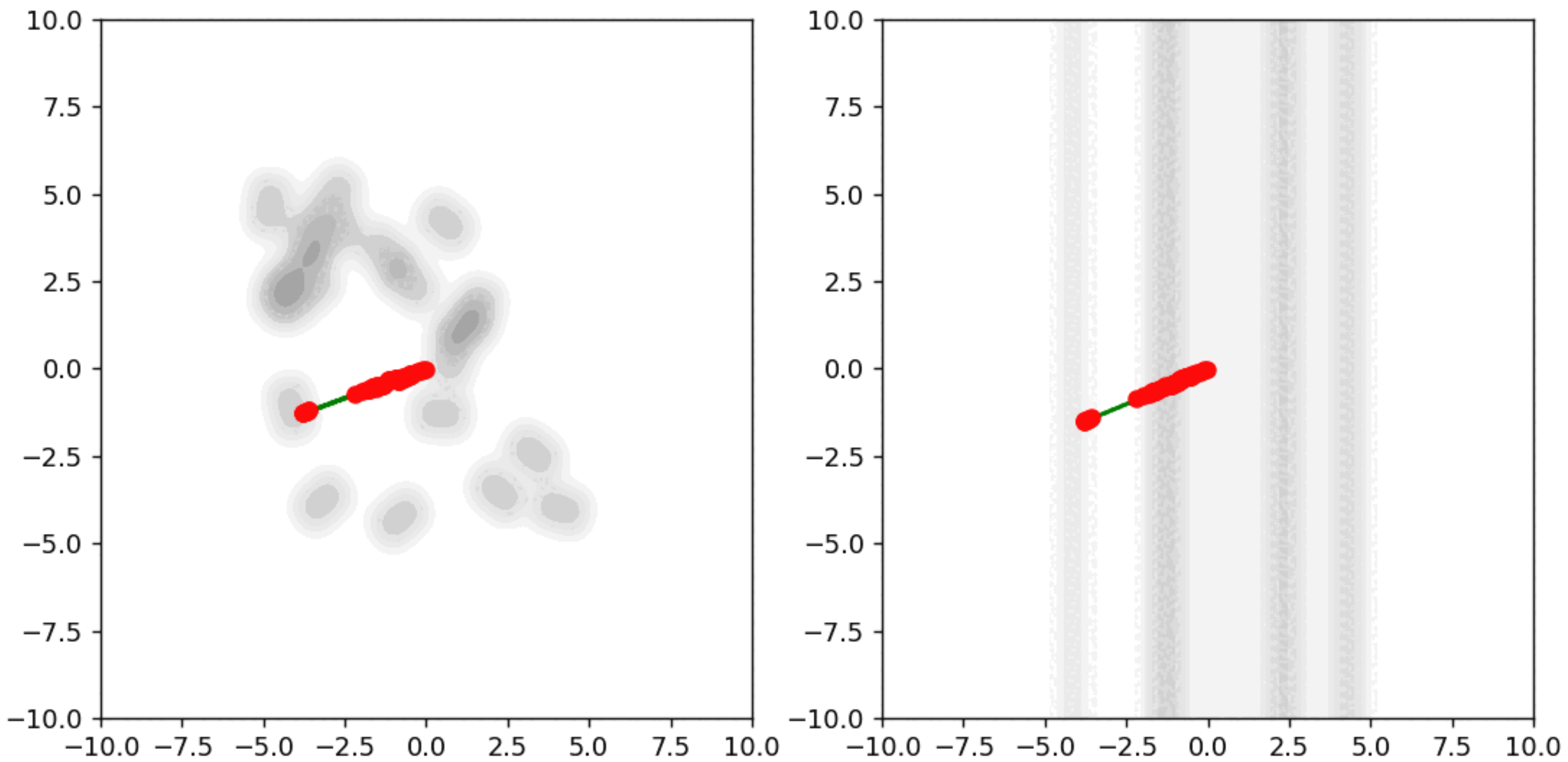
2023

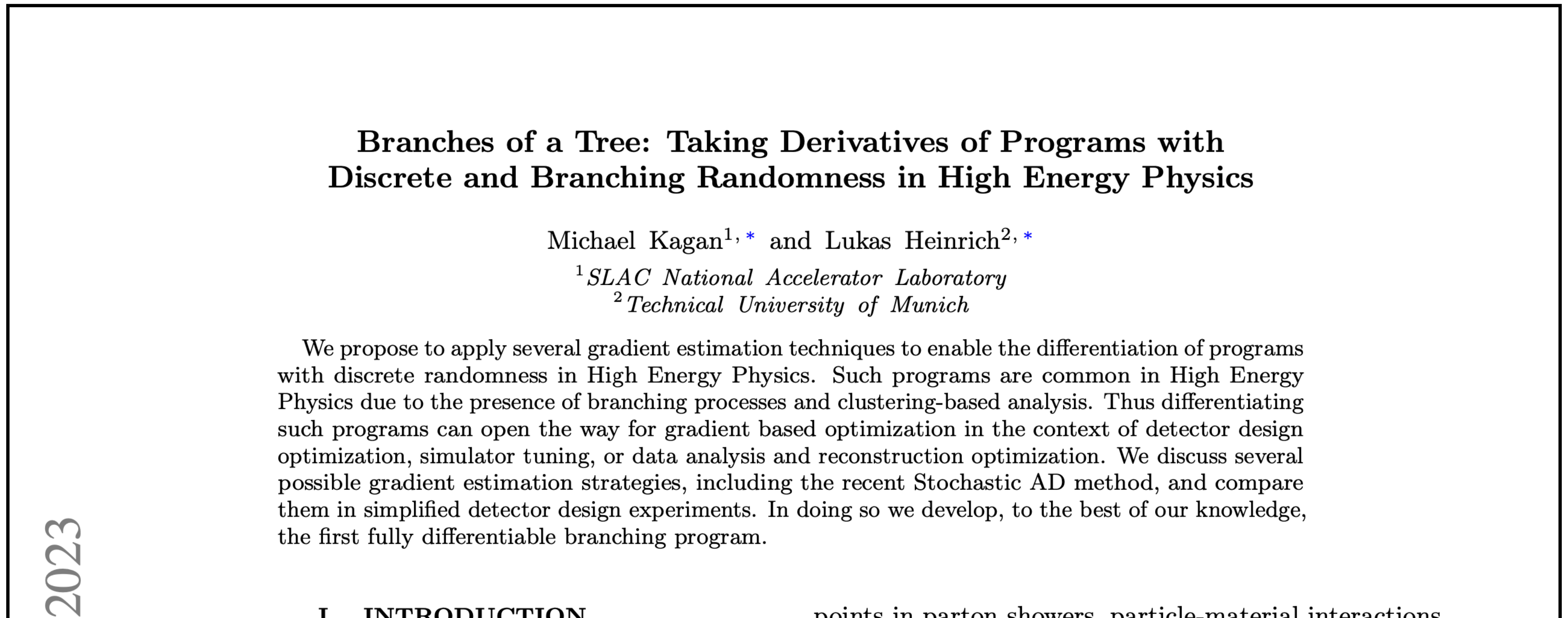
I. INTRODUCTION

points in parton showers, particle-material interactions

Differentiating through Particle Showers

It Works! Can optimize layout from post-shower reward!





Branches of a Tree: Taking Derivatives of Programs with Discrete and Branching Randomness in High Energy Physics

Michael Kagan[1,*] and Lukas Heinrich[2,*]

[1] SLAC National Accelerator Laboratory
[2] Technical University of Munich

We propose to apply several gradient estimation techniques to enable the differentiation of programs with discrete randomness in High Energy Physics. Such programs are common in High Energy Physics due to the presence of branching processes and clustering-based analysis. Thus differentiating such programs can open the way for gradient based optimization in the context of detector design optimization, simulator tuning, or data analysis and reconstruction optimization. We discuss several possible gradient estimation strategies, including the recent Stochastic AD method, and compare them in simplified detector design experiments. In doing so we develop, to the best of our knowledge, the first fully differentiable branching program.

2023

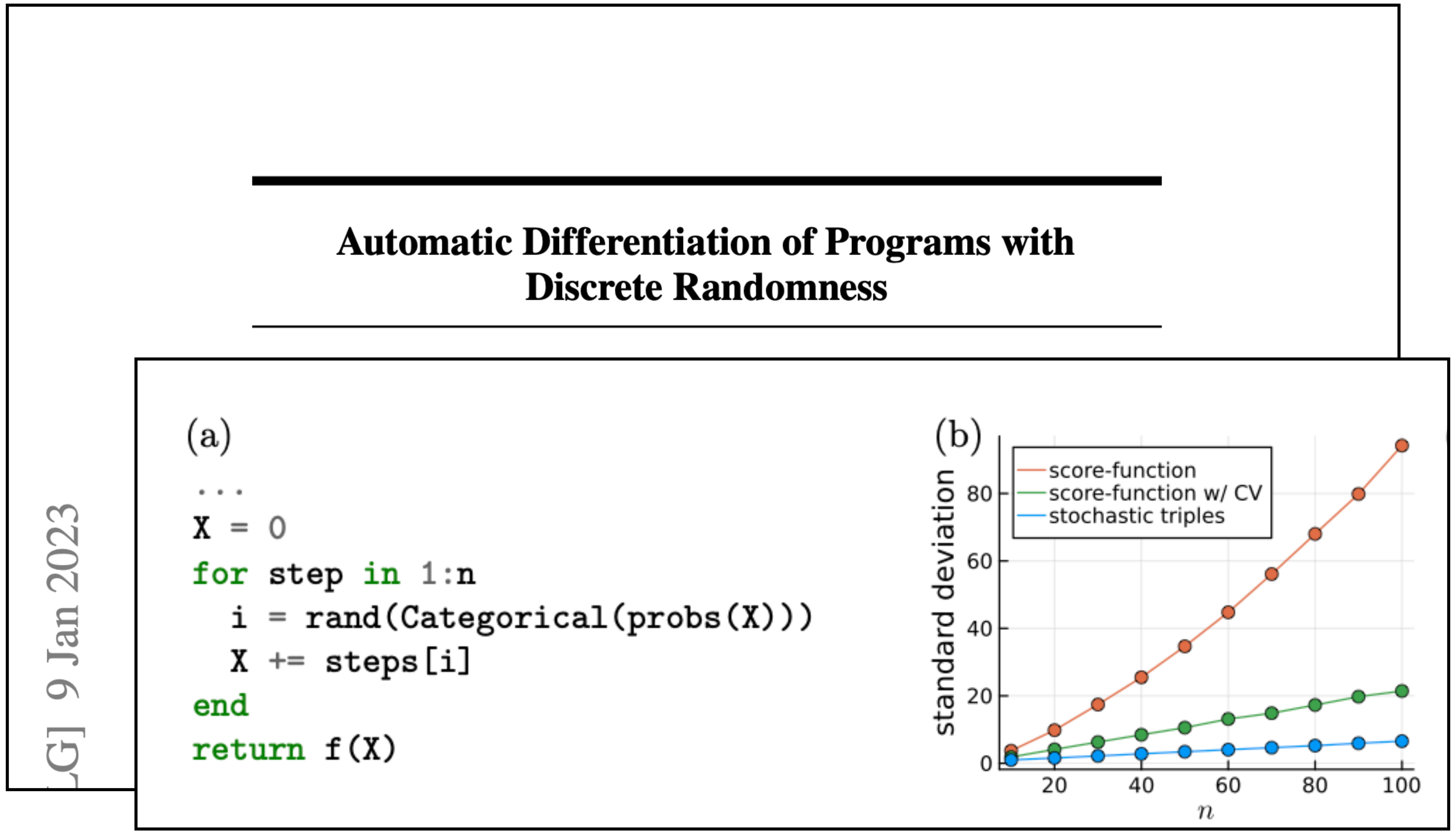
I. INTRODUCTION

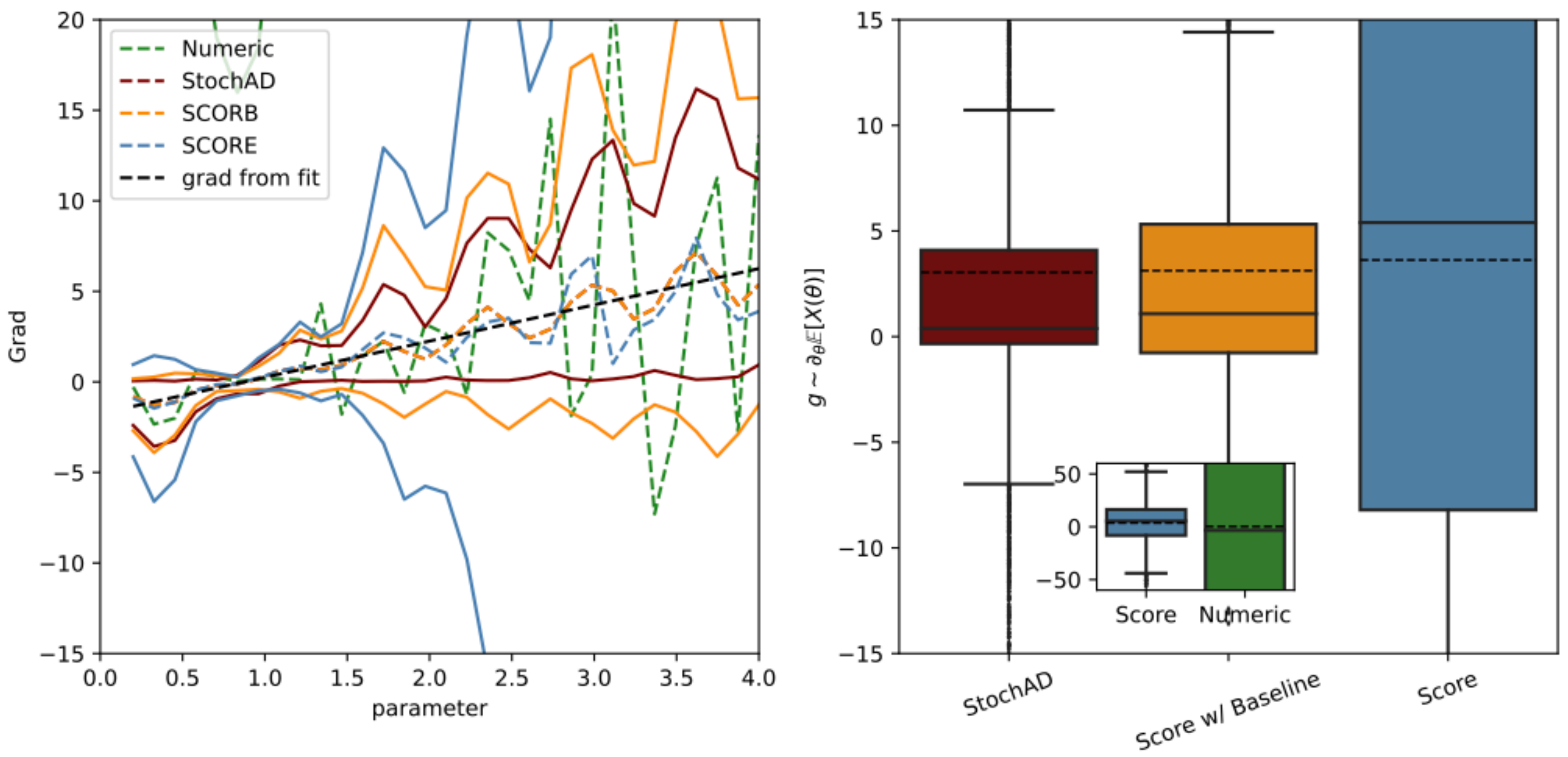
points in parton showers, particle-material interactions

Differentiating through Particle Showers

Also investigated new Stochastic Gradient Estimators.
→ Stochastic AD: gradients for stochastic domain.
→ Promises much lower variance: active R&D happening

(but our toy was prob. too simple)



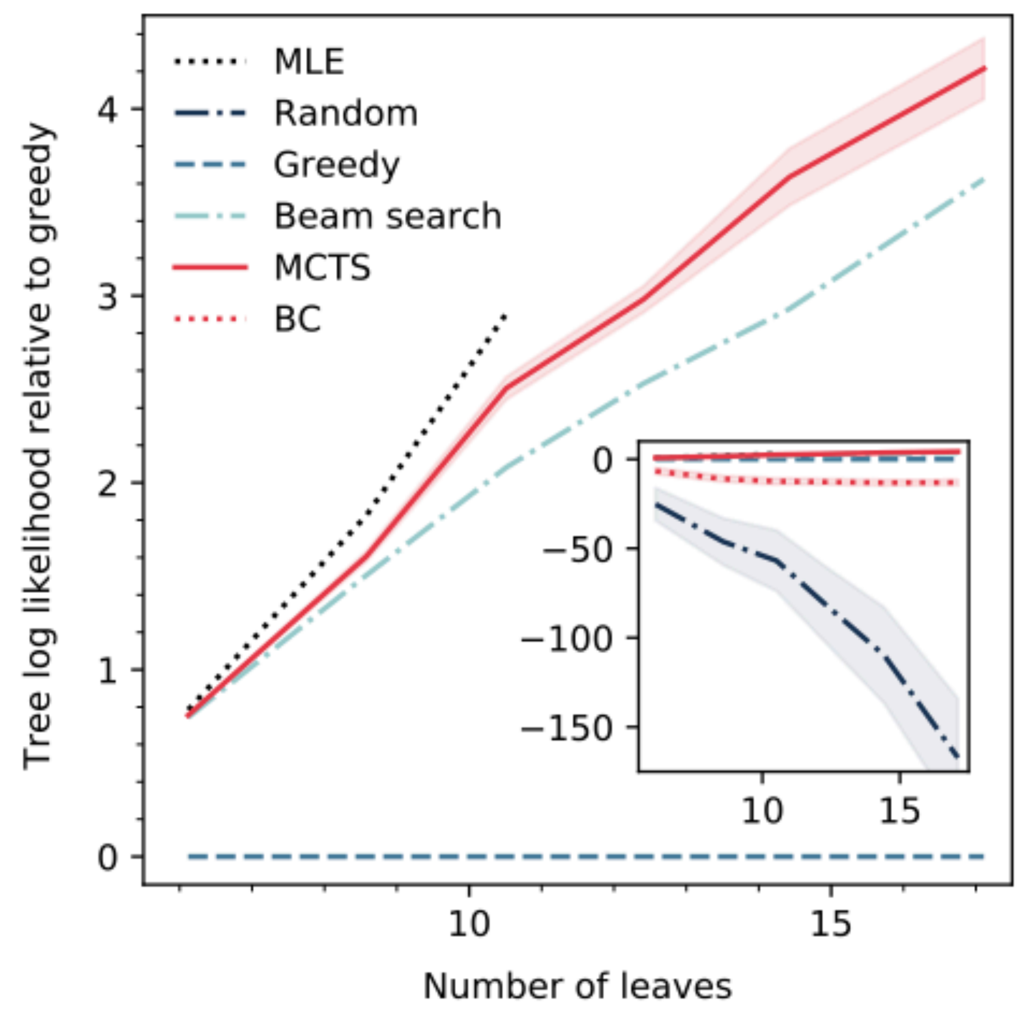


In our case (Showers) not a huge difference

Can we do the same for discrete Structures in Inference

Easiest would be to make discrete choices (cuts, clustering) etc.

probabilistic programs that we sample from



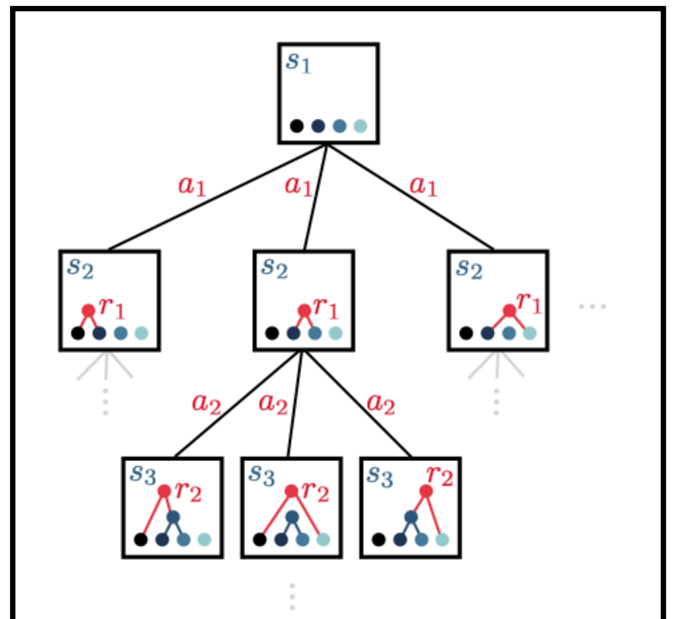
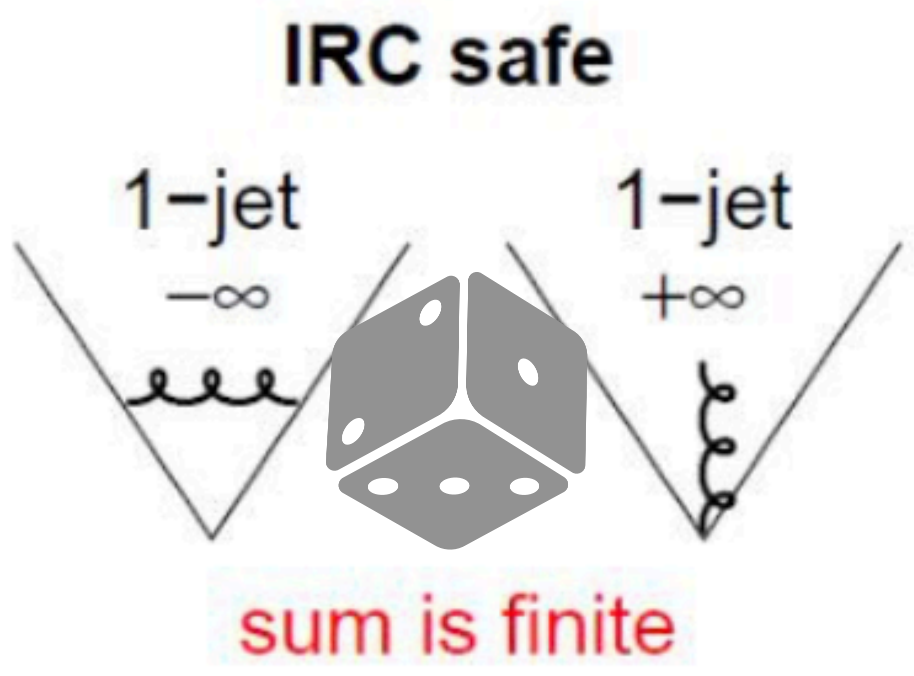


Figure 1: Jet clustering as a Markov Decision Process. States s (squares) represent (partial) clusterings of the original particles (small circles), the agent begins in the unclustered state (top square). Each action a chooses a pair of particles in the current state (which may be either part of the original particle set or the result of a partial clustering) to be merged next. The reward r is the log likelihood of the corresponding $1 \to 2$ splitting.



G. Salam

Early work from Kyle++
Jet Clusterings as RL

Q to Jesse (?) : can we formulate a stochastic Jet Clustering / Def. that is IRC safe?

Some Answers to last H&N

Fine-tuning workflow for end to end
analysis works and is useful even for simple examples

High-Dim Embeddings are a good idea

Gradients of Discrete Randomness
is a promising direction

Some new Questions for next H&N ?

How do we calibrate high-dim representation?

Will we get a "safe" calibrated fine tuning manifold?

Can we optimize structural pieces (e.g. jet definition) → stochastic reconstruction?

Supervised vs Self-supervised Backbones (JetCLR, ReSim, MPM,…)

Michael's Talk Next

Thanks!