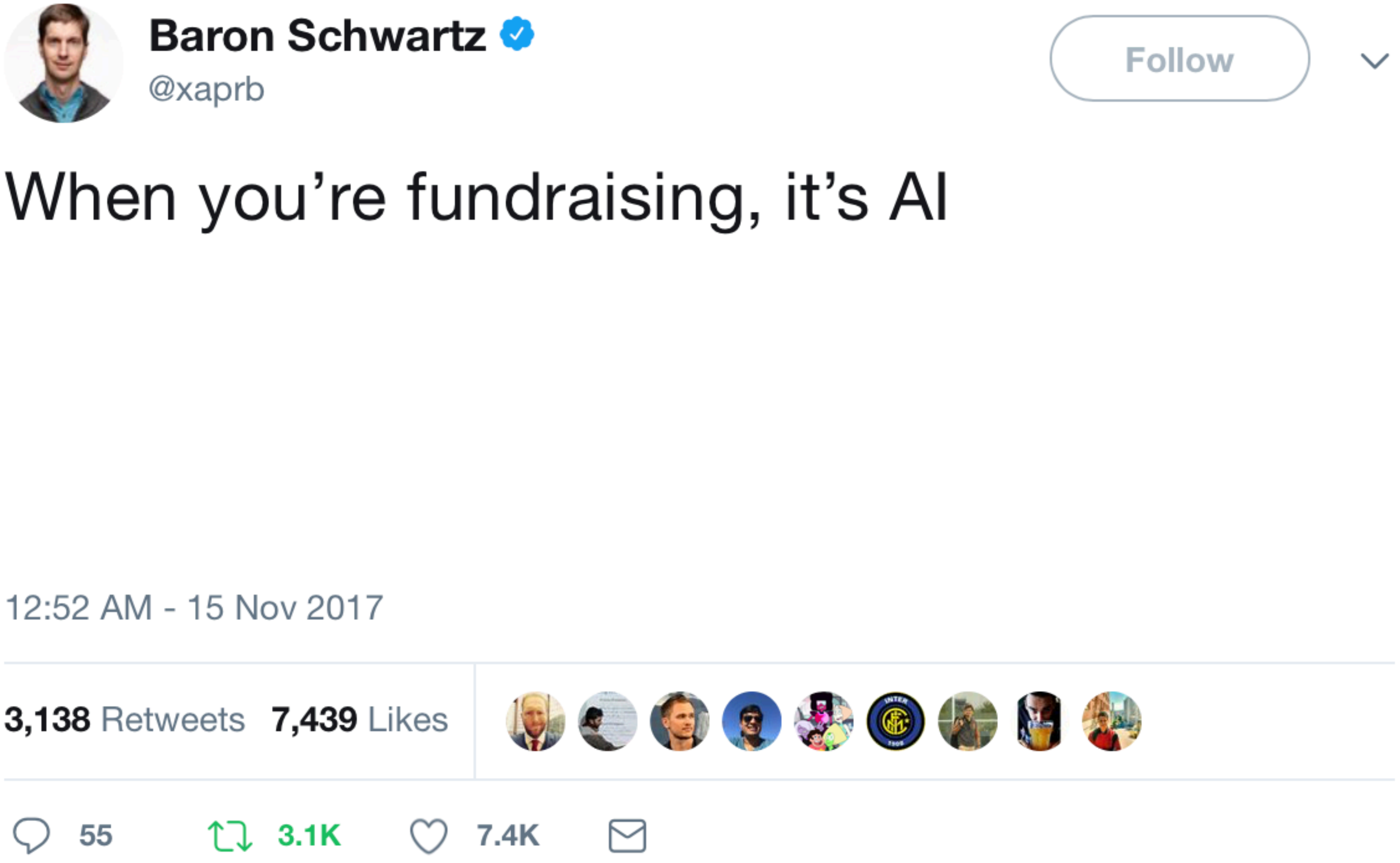
AI for MC

Gabriel Perdue // @gnperdue

Fermi National Accelerator Laboratory // @Fermilab

ECT, Trento, July 2018

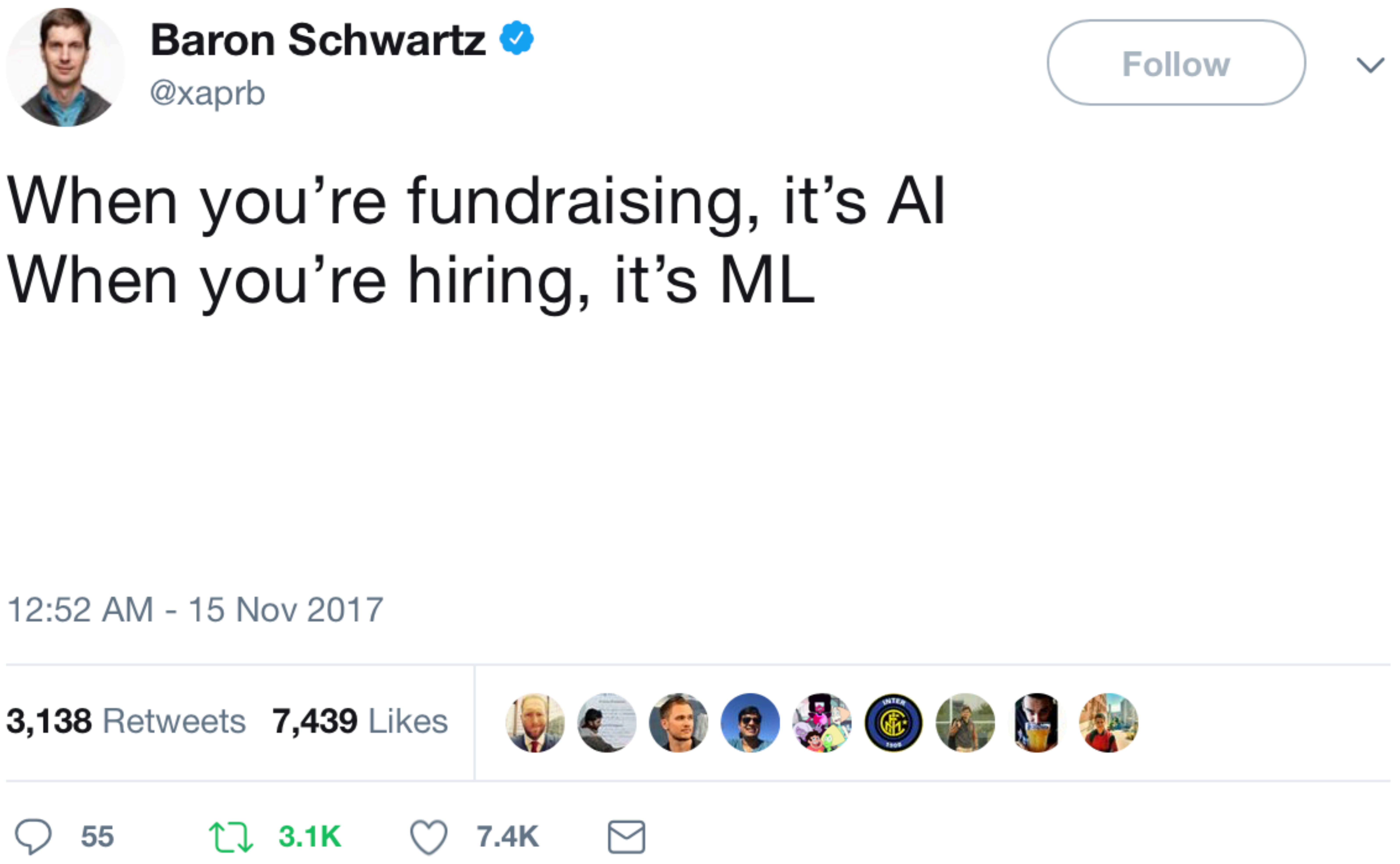


Baron Schwartz
@xaprb

When you’re fundraising, it’s AI

12:52 AM - 15 Nov 2017

3,138 Retweets 7,439 Likes



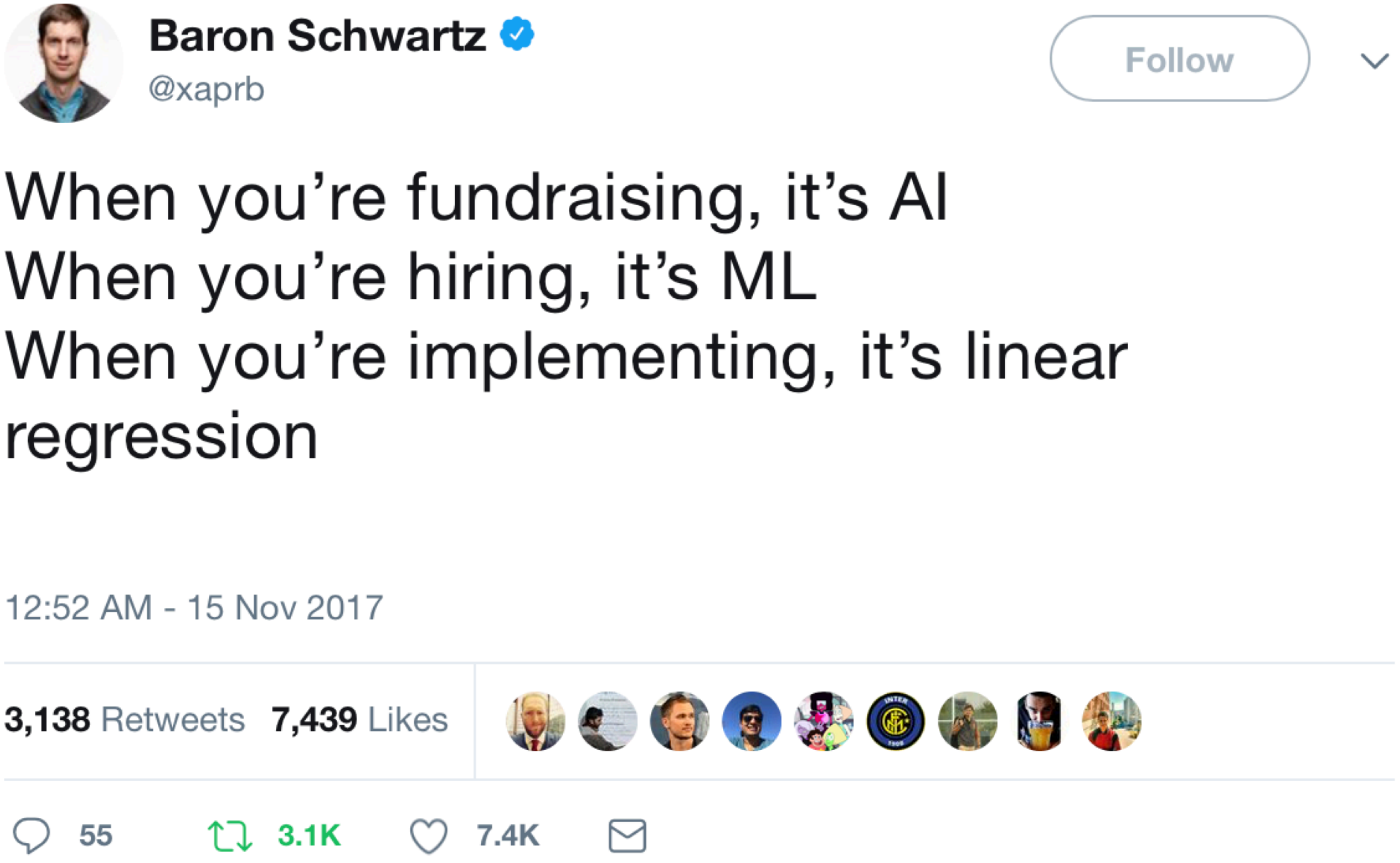
Baron Schwartz
@xaprb

When you’re fundraising, it’s AI
When you’re hiring, it’s ML

12:52 AM - 15 Nov 2017

3,138 Retweets **7,439** Likes

55 3.1K 7.4K

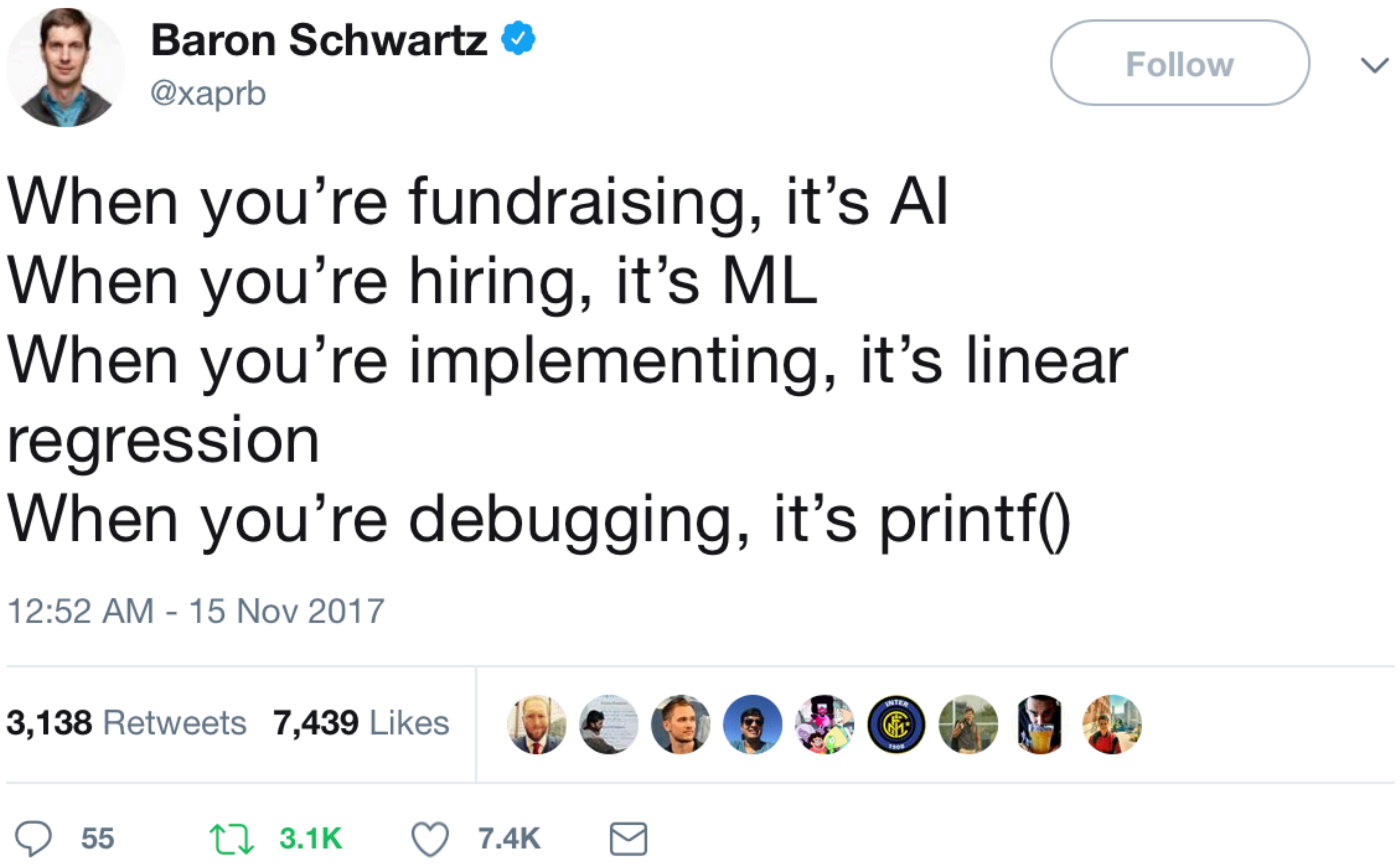


Baron Schwartz
@xaprb

When you’re fundraising, it’s AI
When you’re hiring, it’s ML
When you’re implementing, it’s linear regression

12:52 AM - 15 Nov 2017

3,138 Retweets **7,439** Likes



Baron Schwartz
@xaprb

When you’re fundraising, it’s AI
When you’re hiring, it’s ML
When you’re implementing, it’s linear regression
When you’re debugging, it’s printf()

12:52 AM - 15 Nov 2017

3,138 Retweets **7,439** Likes

55 3.1K 7.4K

Outline

- Neural networks and deep learning (and machine learning, more broadly) for event generators.
  - Two "levels":
    - "That sounds promising. I see where you are going there..."
    - "Why the heck would you do *that*!?"
- Conclusions

ALEX DAVIES TRANSPORTATION 10.25.16 06:00 AM

UBER'S SELF-DRIVING TRUCK MAKES ITS FIRST DELIVERY: 50,000 BEERS

Workfolio
GET MORE INTERVIEWS
SEE HOW A PERSONAL WEBSITE CAN HELP YOU STAND OUT TO EMPLOYERS
GET S
MOST POPUL
TRANSPORTA
What Does
Automated
Truckers?
AARIAN MAR
TRANSPORTA
Elon Musk

Jet Substructure Classification in High-Energy Physics with Deep Neural Networks

Pierre Baldi,[1] Kevin Bauer,[2] Cl

[1] Department of Computer Sci

[2] Department of Physics and Ast

[3] Department of Chemical Engineerin

(I

WALL STREET

FINANCE | BANKS | INVESTING | WALL STREET | HEDGE FUNDS | M&A | INSURANC

Who needs Wall Street? New robot-powered ETF beating the market so far

TECHNOLOGY

Artificial intelligence now powers all of Facebook's translation

The new system is more accurate than the old method.

By Rob Verger August 4, 2017

Deep-learning algorithm recommends ingredients and recipes based on a photo of food

July 21, 2017 by Adam Conner-Simons

JANUARY 25, 2017

Deep learning algorithm does as well as dermatologists in identifying skin cancer

In hopes of creating better access to medical care, Stanford researchers have trained an algorithm to diag cancer.

PUBLIC RELEASE: 27-JUL-2017

First molecules discovered by next-generation artificial intelligence to be developed into drugs

INSILICO MEDICINE, INC.

Thursday, July 27, 2017, Baltimore, Md., Insilico Medicine ("Insilico"), a Baltimore-based leader in artificial intelligence ("AI")

The sane... and the more... speculative.

- Various ML techniques show real promise for speeding up slow calculations. There are many cases where we have solve very complex expressions using algorithms with terrible (even exponential) scaling but where a ML algorithm can approximate the result to very tolerable precision in fixed time.
  - Will show examples for MC Integration, and
  - parametric regression for LQCD (won't have time, but see PRD 97, 094506).
- More speculatively, we may also use ML algorithms as event generators directly:
  - Demonstrated success for simple detector geometries for electromagnetic showers using Generative Adversarial Networks,
  - Success with recurrent and other sequence-based models on language problems suggest we could produce event generators based on a mixture of theory and learned models.
- Bonus topic - also won't have time, but see arXiv 1805.00905 for an interesting application of NNs to form factor fitting (not exactly MC work, but probably interesting to this audience).

MC integration

- https://indico.cern.ch/event/568875/ (MC4BSM 2017), J. Bendavid
- arXiv 1707.00028, J. Bendavid
- arXiv 1603.02754 (xgboost)
- arXiv 1406.2661 (GAN foundation paper, *many* others)
- (plus more GAN references in the next section)

MC Sampling

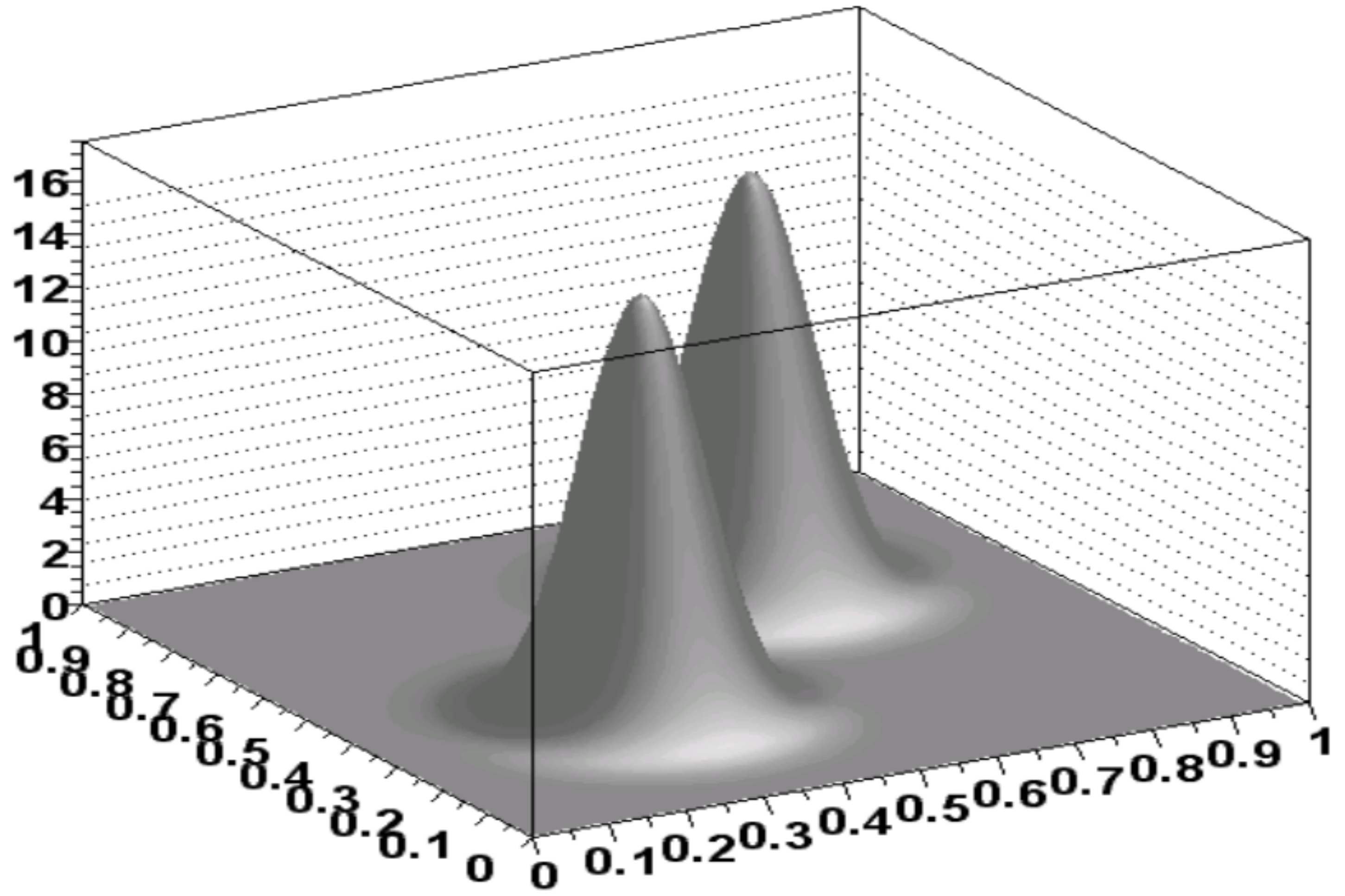
- Simplest possible approach:
  - randomly sample the function with a uniform distribution,
  - compute a weight for each sample based on the value and the sampling size,
  - integral is the weighted sum of all samples.
  - Generation with accept-reject.
- Naive implementations are very inefficient.
  - Some regions of the function are much more important than others for computing an integral - *importance sampling*:
    - instead of sampling from a uniform distribution, use some generating probability,
    - now weights are modified by the value of the generating distribution,
    - maximally efficient case - the generating distribution is a good approximation of the integral we'd like to perform (and is easy to sample from).

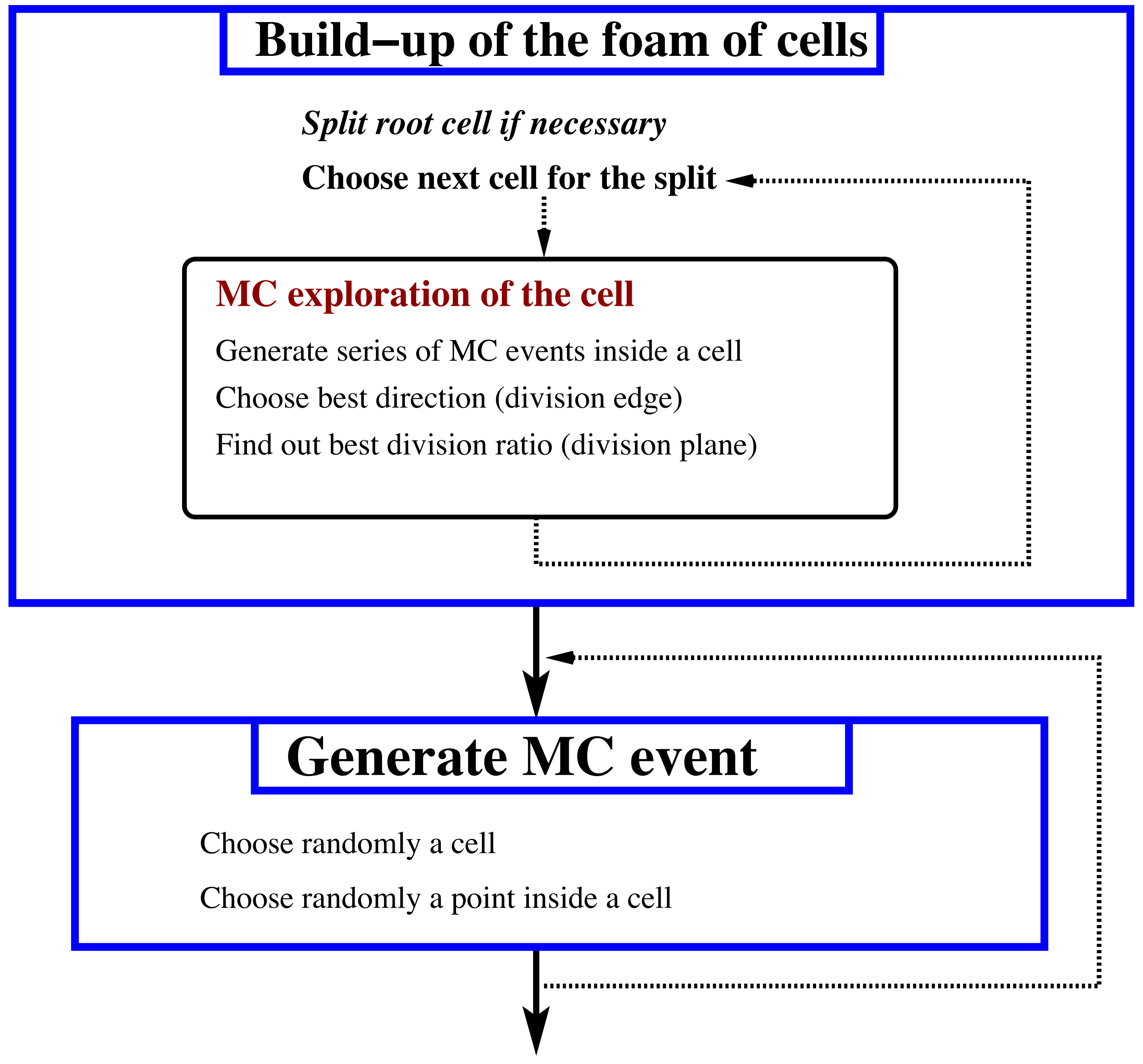
VEGAS

- Smart integration algorithm - Lepage, "A New Algorithm for Adaptive Multidimensional Integration", Journal of Computational Physics 27, 192-203 (1978)
- Iterative - at each iteration build an adaptive-binned histogram that better approximates the target function.
- Multidimensional functions are treated as products of one-dimensional histograms - so, fast and simple, but the algorithm has trouble when there are complex correlations across dimensions.
- Sometimes we can change basis or transform the function to make the problem easier, but not always.

FOAM

- S. Jadach, physics/0203033
- Truly multidimensional sampling function.
  - Use a **decision tree** to divide the phase space into optimized hyper-rectangles.
  - Sample uniformly within each hyper-rectangle to determine whether and where to perform a binary split (until a stopping condition is finally met within the cell).
  - Weights are proportional the integrals over hyper-rectangles.
  - Weighted sampling by hyper-rectangle, then randomly within the volume.





Build–up of the foam of cells

Split root cell if necessary

Choose next cell for the split

MC exploration of the cell

Generate series of MC events inside a cell

Choose best direction (division edge)

Find out best division ratio (division plane)

Generate MC event

Choose randomly a cell

Choose randomly a point inside a cell

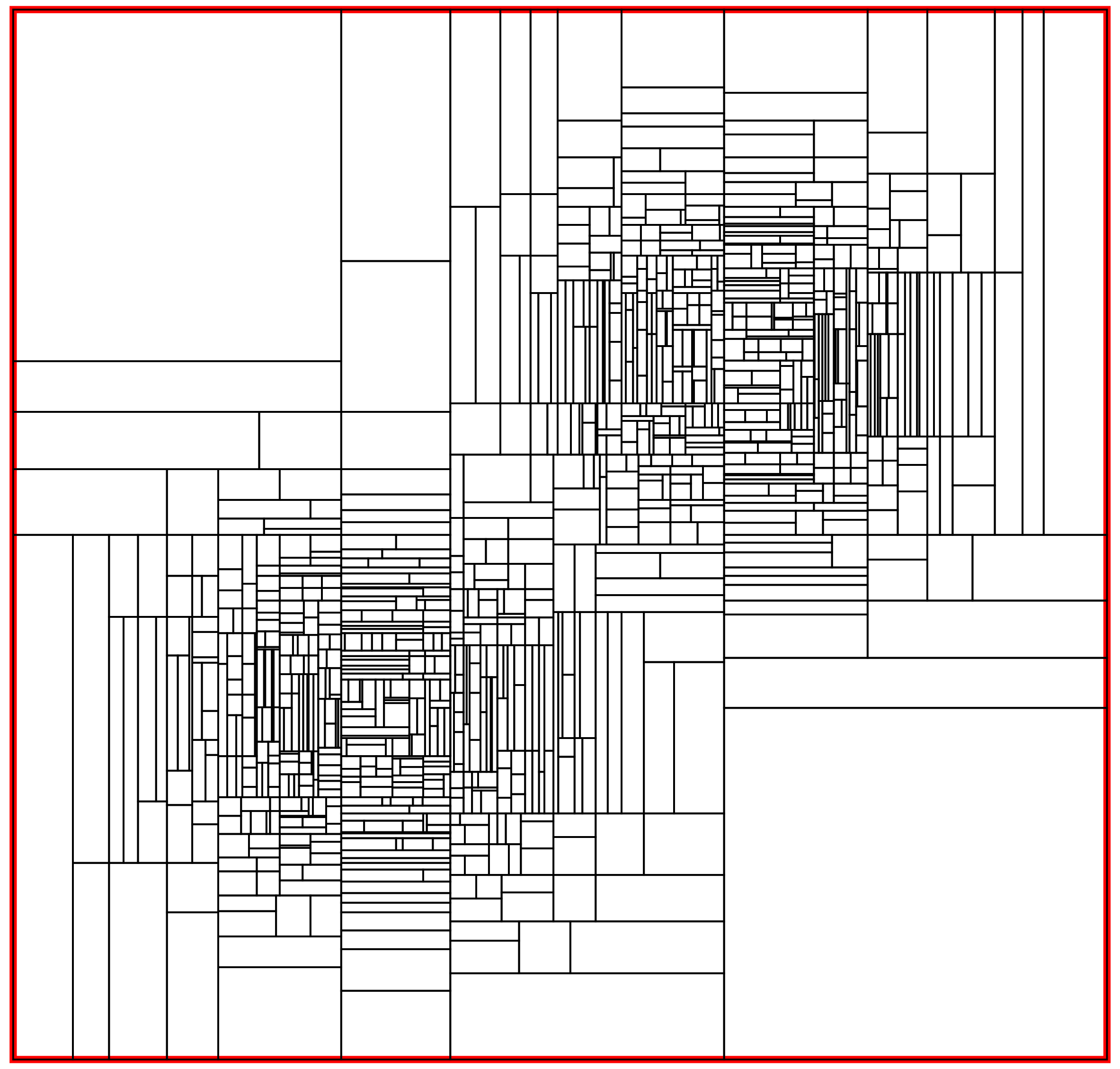


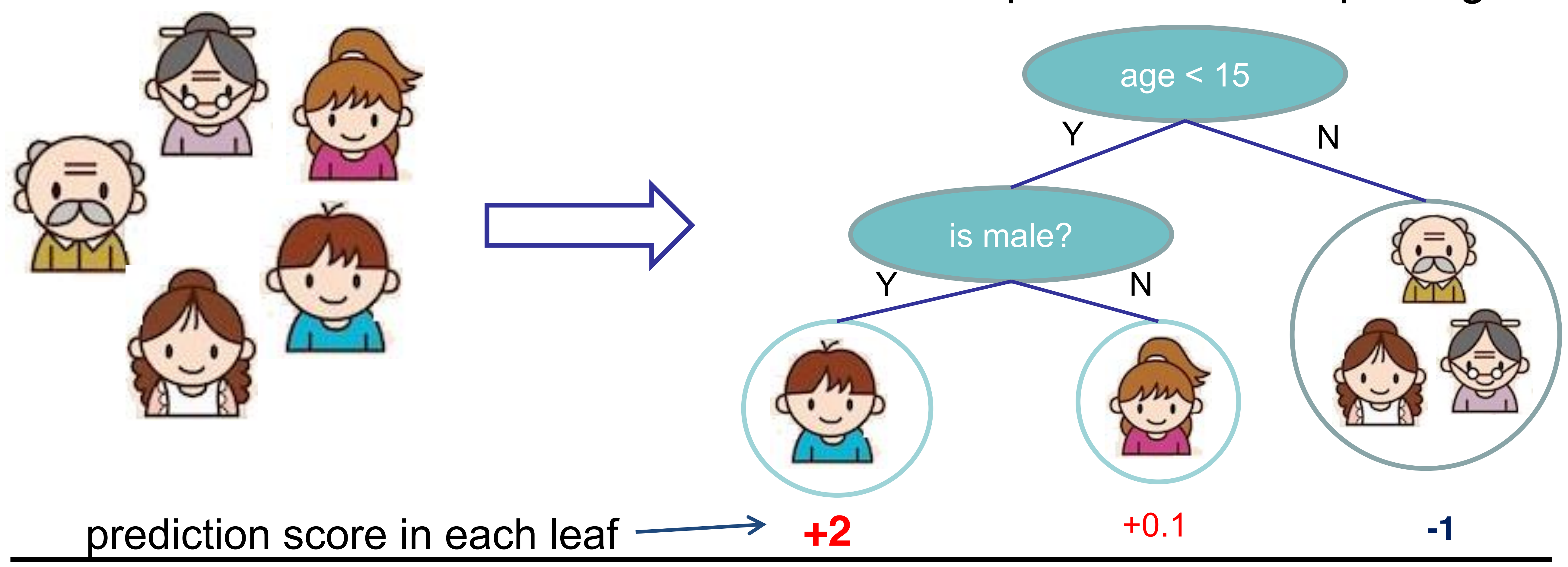
Figure 1: Two stages in the cellular algorithm of Foam.

Regression Tree (CART)

- Decision rules for each branch.
- Leaf nodes contain scores.
- Useful in classification also.
- Function approximation.

Input: age, gender, occupation, …

Does the person like computer games



age < 15

Y

N

is male?

Y

N

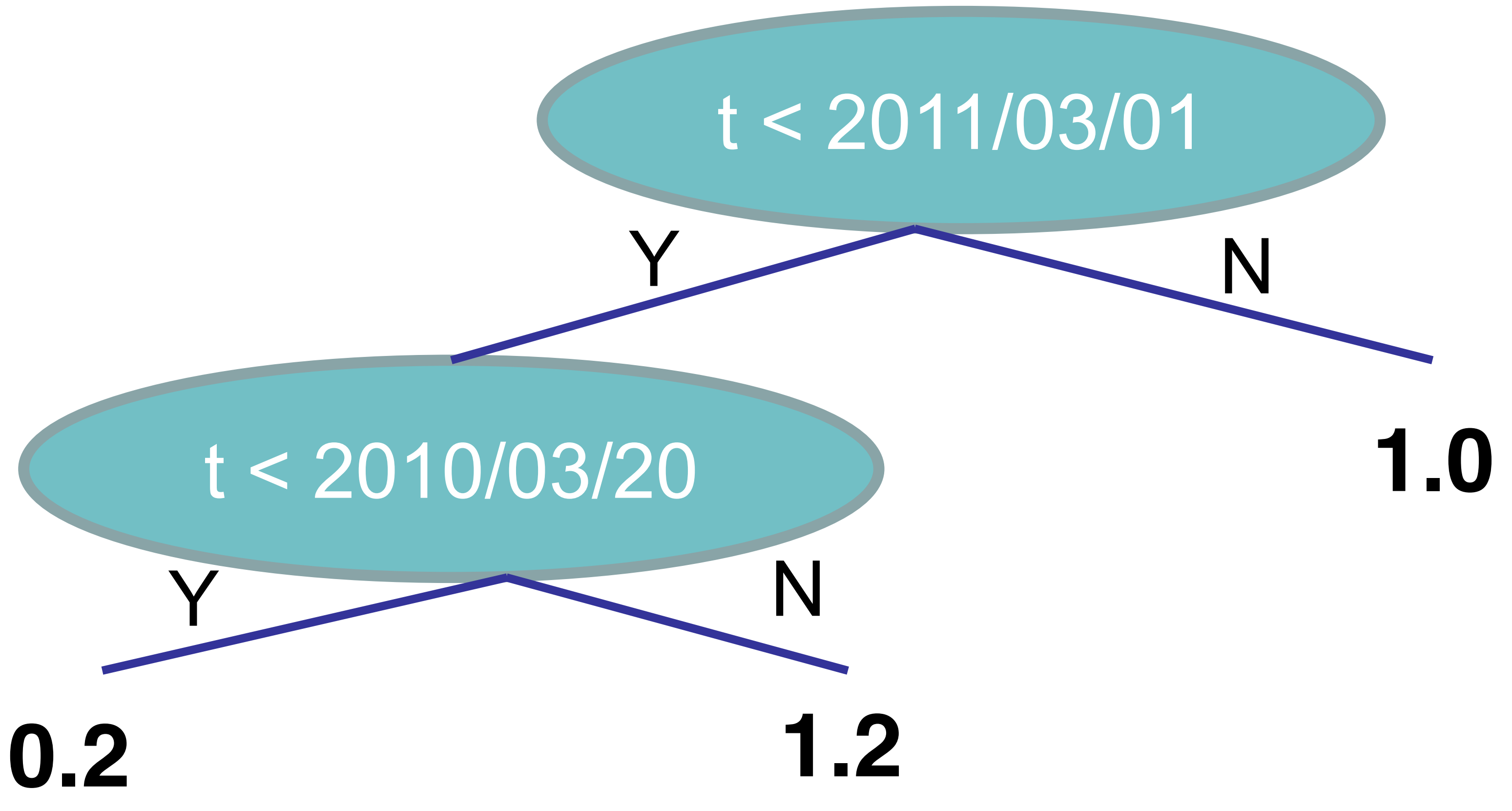
prediction score in each leaf

+2

+0.1

-1

The model is regression tree that splits on time



t < 2011/03/01

Y

N

1.0

t < 2010/03/20

Y

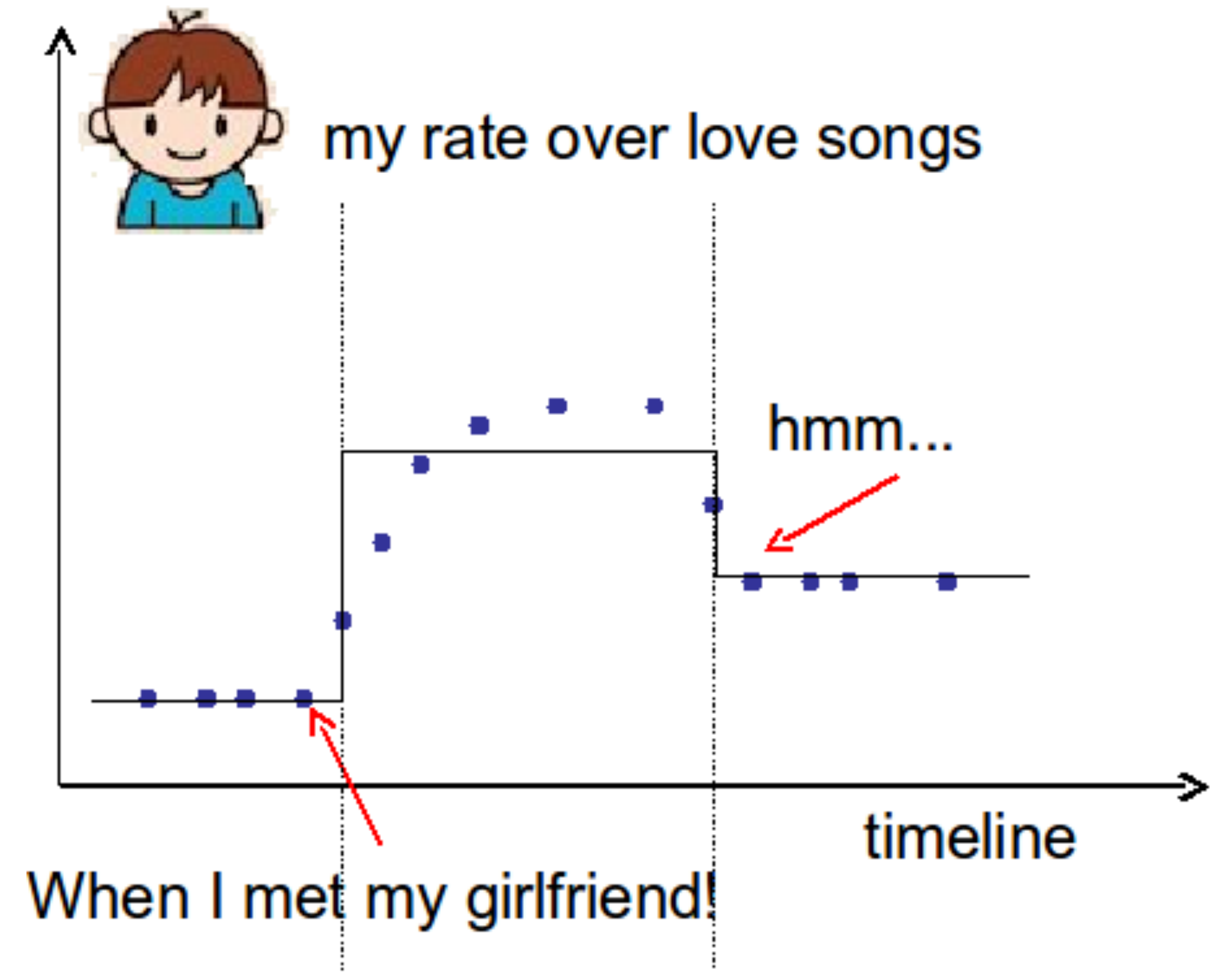
N

0.2

1.2

Equivalently

Piecewise step function over time



my rate over love songs

hmm...

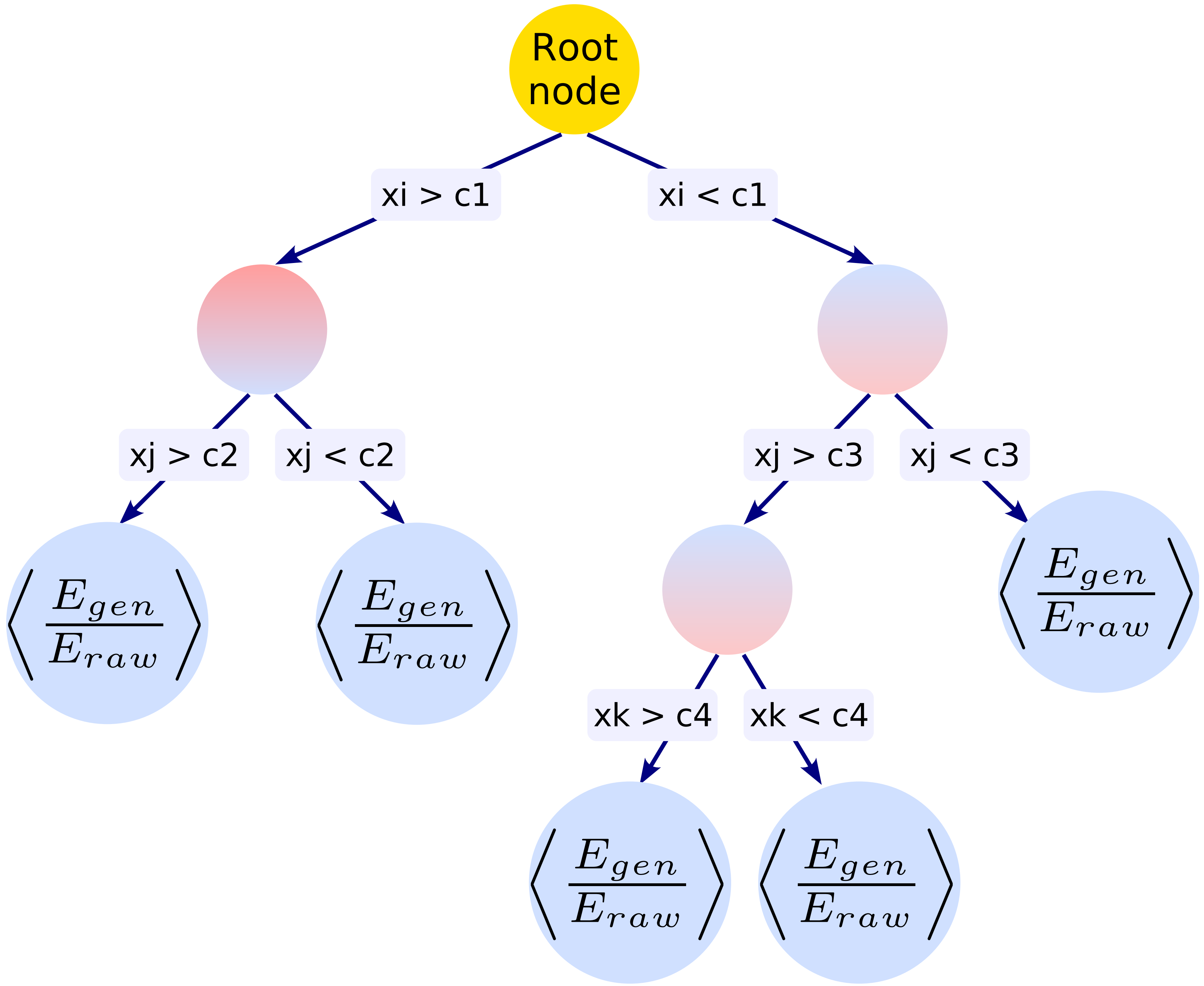
When I met my girlfriend!

timeline

Tianqi Chen
University of Washington
tqchen@cs.washington.edu

Carlos Guestrin
University of Washington
guestrin@cs.washington.edu

Boosted trees for regression

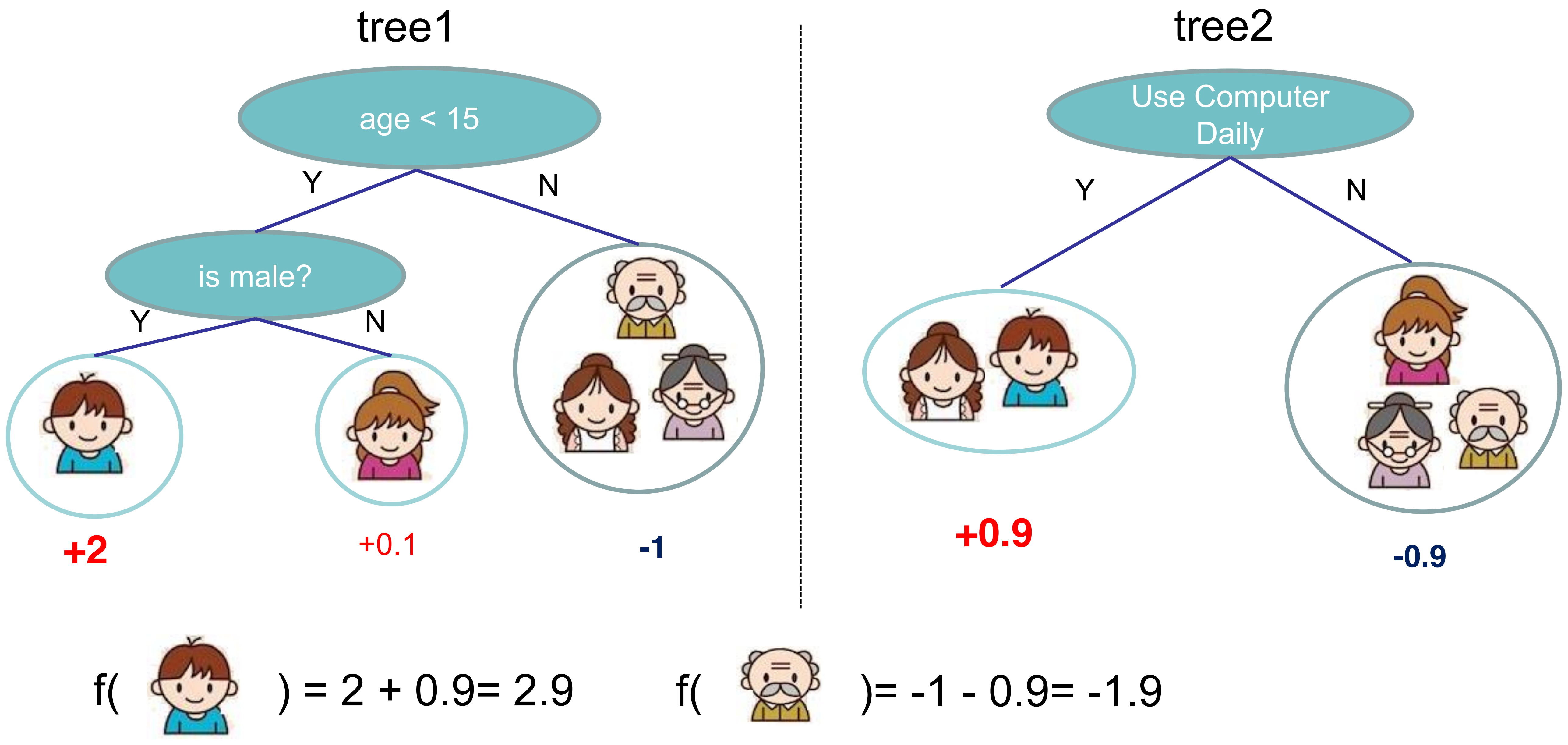

Root node
xi > c1
xi < c1
xj > c2
xj < c2
xj > c3
xj < c3
xk > c4
xk < c4
$\left\langle \frac{E_{gen}}{E_{raw}} \right\rangle$
$\left\langle \frac{E_{gen}}{E_{raw}} \right\rangle$
$\left\langle \frac{E_{gen}}{E_{raw}} \right\rangle$
$\left\langle \frac{E_{gen}}{E_{raw}} \right\rangle$
$\left\langle \frac{E_{gen}}{E_{raw}} \right\rangle$

- Boosted Decision Trees can also be used for multivariate regression problem
- Replace log likelihood ratio with generic function $f(\bar{x})$
- Minimize deviation between training sample and regression function
- Decision trees form a series of piecewise continuous approximations for the function $f(\bar{x})$ in the multidimensional input space

Josh Bendavid (Caltech/LPC)

Tree ensemble

- Ensembling (combining many ML models and voting/summing) is widely known to improve accuracy.
- Diversity of trees allows exploitation of combinatorics to express very complex outcomes with relatively simple model components.



Prediction of is sum of scores predicted by each of the tree

Tianqi Chen
University of Washington
tqchen@cs.washington.edu

Carlos Guestrin
University of Washington
guestrin@cs.washington.edu

Tianqi Chen
University of Washington
tqchen@cs.washington.edu

- Model: assuming we have K trees

$$\hat{y}_i = \sum_{k=1}^{K} f_k(x_i), \quad f_k \in \mathcal{F}$$

- Objective

$$Obj = \sum_{i=1}^{n} l(y_i, \hat{y}_i) + \sum_{k=1}^{K} \Omega(f_k)$$

Training loss

Complexity of the Trees

- Using Square loss $l(y_i, \hat{y}_i) = (y_i - \hat{y}_i)^2$
  - Will results in common gradient boosted machine
- Using Logistic loss $l(y_i, \hat{y}_i) = y_i \ln(1 + e^{-\hat{y}_i}) + (1 - y_i) \ln(1 + e^{\hat{y}_i})$
  - Will results in LogitBoost

Tianqi Chen
University of Washington
tqchen@cs.washington.edu

- Objective: $\sum_{i=1}^{n} l(y_i, \hat{y}_i) + \sum_k \Omega(f_k), f_k \in \mathcal{F}$
- We can not use methods such as SGD, to find f (since they are trees, instead of just numerical vectors)
- Solution: **Additive Training (Boosting)**
  - Start from constant prediction, add a new function each time

$$
\begin{aligned}
\hat{y}_i^{(0)} &= 0 \\
\hat{y}_i^{(1)} &= f_1(x_i) = \hat{y}_i^{(0)} + f_1(x_i) \\
\hat{y}_i^{(2)} &= f_1(x_i) + f_2(x_i) = \hat{y}_i^{(1)} + f_2(x_i) \\
&\dots \\
\hat{y}_i^{(t)} &= \sum_{k=1}^{t} f_k(x_i) = \hat{y}_i^{(t-1)} + f_t(x_i)
\end{aligned}
$$

New function

Model at training round t

Keep functions added in previous round

- Goal $Obj^{(t)} = \sum_{i=1}^{n} l\left(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)\right) + \Omega(f_t) + constant$
  - Seems still complicated except for the case of square loss
- Take Taylor expansion of the objective
  - Recall $f(x + \Delta x) \simeq f(x) + f'(x)\Delta x + \frac{1}{2} f''(x)\Delta x^2$
  - Define $g_i = \partial_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)}), \quad h_i = \partial^2_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)})$

$$Obj^{(t)} \simeq \sum_{i=1}^{n} \left[l(y_i, \hat{y}_i^{(t-1)}) + g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i) \right] + \Omega(f_t) + constant$$

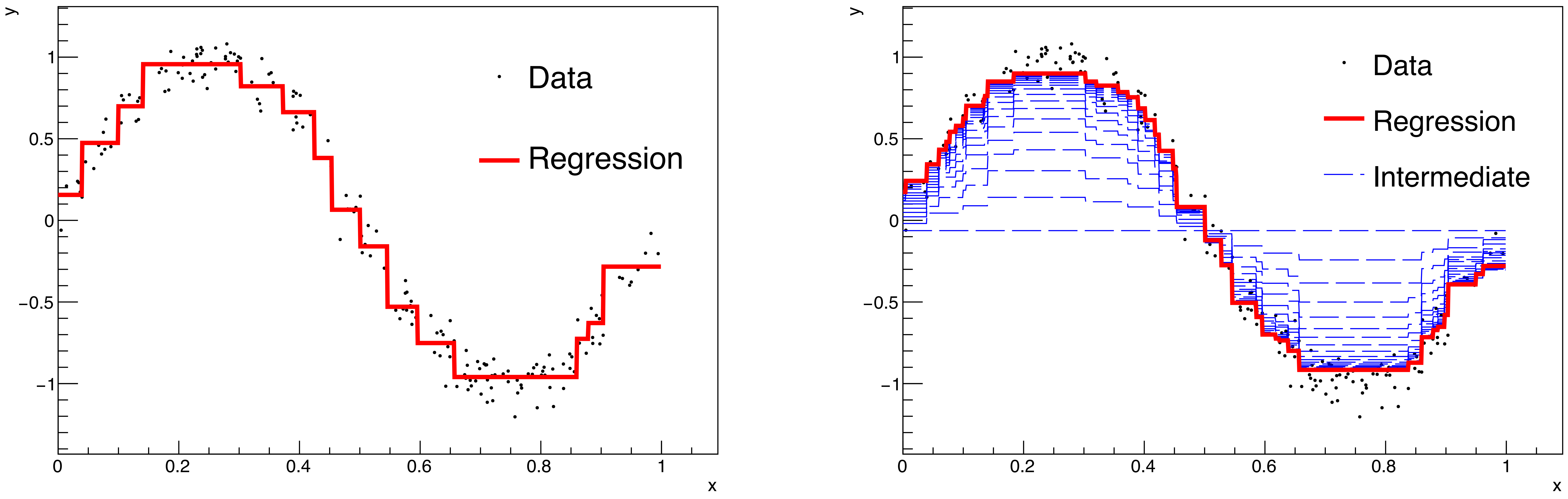
This is why we call this method "gradient" boosting.

Compare to, e.g. adaptive boosting - weight examples to highlight losses from the previous round choose a split in a "decision stump" based on the maximum information gain in a split of the newly weighted sample.

Tianqi Chen
University of Washington
tqchen@cs.washington.edu

Gradient boosting for function approximation

- Build an additive series of piecewise continuous approximations.
- Can represent more complex functions than a single tree (ensemble advantages).
- Allows for more efficient MC sampling / integration?



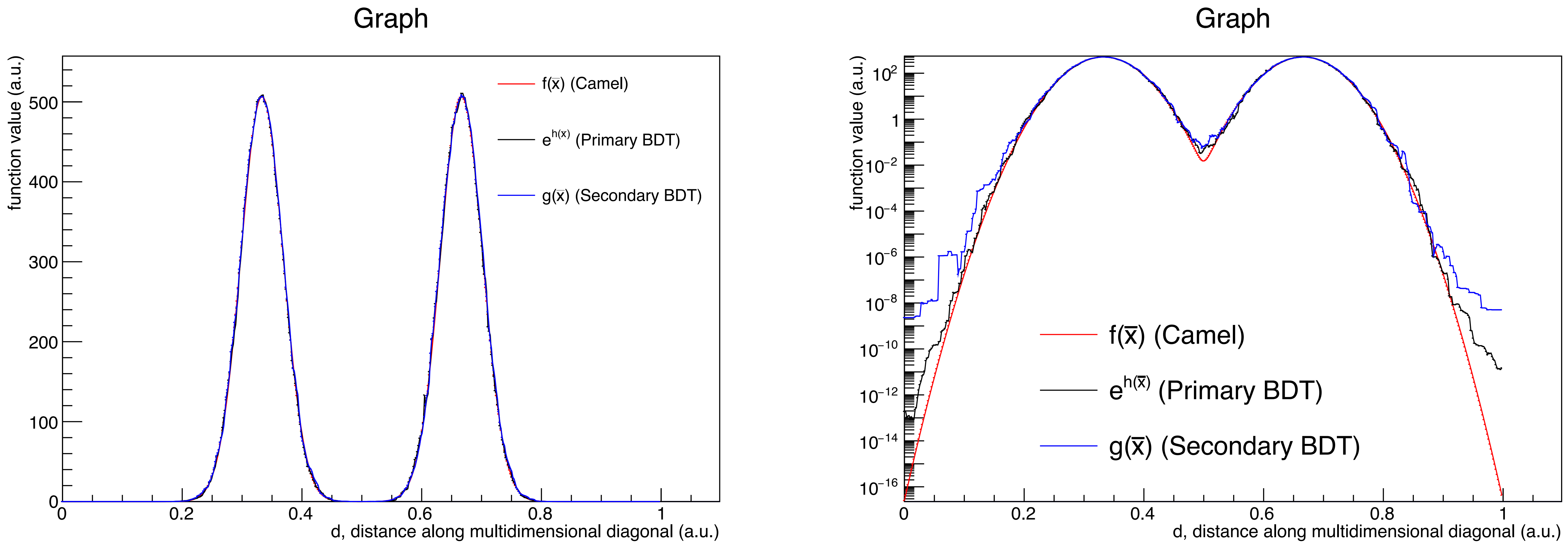
(a) Single Tree

(b) Gradient Boosted ($\sim$ 20 trees)

Josh Bendavid (Caltech/LPC)

Gradient boosting - 4D Camel

Algorithm	# of Func. Evals	$\sigma_w / < w >$	σ_I / I
			(2e6 add. evts)
VEGAS	300,000	2.820	$\pm 2.0 \times 10^{-3}$
Foam	3,855,289	0.319	$\pm 2.3 \times 10^{-4}$
GBRIntegrator	300,000	0.082	$\pm 5.8 \times 10^{-5}$
GBRIntegrator (staged)	300,000	0.077	$\pm 5.4 \times 10^{-5}$



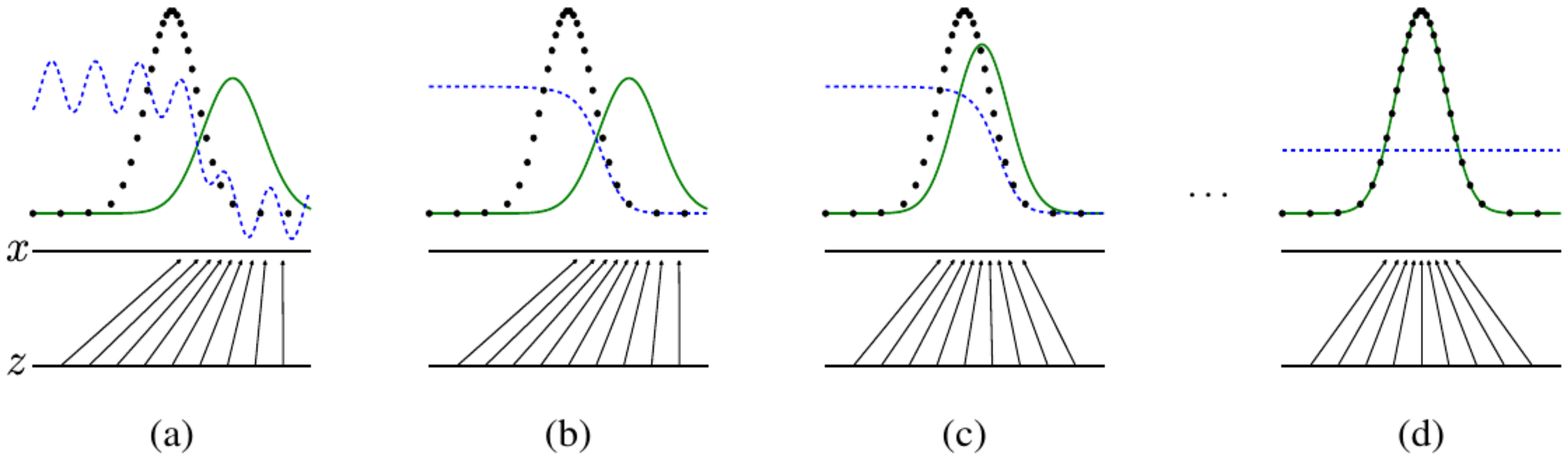
(a) linear

(b) log

Josh Bendavid (Caltech/LPC)

Generative adversarial neural networks

- Generative adversarial networks (GANs) pair a *generator* network and a *discriminator* network. The training is adversarial in that the loss functions are coupled - the discriminator attempts to label "real" and "generated" data and the generator attempts to fool the discriminator.
- Convergence is challenging, and the networks can be very difficult to train, but the results are promising...
  - And the horrifying source of the fake news apocalypse, e.g. arXiv 1805.11714,
  - https://www.youtube.com/watch?reload=9&v=qc5P2bvfl44



arXiv:1406.2661

$$\boxed{\mathcal{L} =} \boxed{\mathbb{E}[\log(\mathbb{P}(D(G(z)) = 0 \mid z \sim (\mu, \sigma^2)))]} + \boxed{\mathbb{E}[\log(\mathbb{P}(D(I) = 1 \mid I \in \mathcal{N}))]}$$

- First term, probability that a fake sample is classified as fake

- Second term, probability that a real sample is classified as real

- Generator wants to minimize this, discriminator wants to maximize this

- If we allow D, G, to be from space of all continuous functions, then
  - There exists a unique Nash equilibrium (no “player” incentivize to deviate off path)
  - G **exactly recovers** $\mathcal{N}$, the data distribution

$$D(I) = \frac{1}{2} \; \forall I \in \mathcal{N} \cup \{G(z) | z \sim (\mu, \sigma^2)\}$$

- Standard formulation is highly unstable

LOTS of tricks! Very active area of research...

Luke de Oliveira

Probability sampling

- Given a generating network G and input space z and output space x (and assuming z and x have the same dimension) we may compute the generating probability density g from the sampling prior and the Jacobian determinant:

$$p(\bar{z}) = g(\bar{x}) \left|\left| \frac{\partial \bar{G}(\bar{z})}{\partial \bar{z}} \right|\right|$$

- Then, if the function to be integrated has some probability density pf = f/Int(f), the KL divergence between f and the generating pdf is

$$D_{KL} = \int g(\bar{x}) \ln \frac{g(\bar{x})}{p_f(\bar{x})} d\bar{x} \qquad\qquad p_f(\bar{x}) = f(\bar{x})/I_f$$

Josh Bendavid (Caltech/LPC)

Network objective

- KL divergence may be approximated from a set of samples from p

$$D_{KL} = \sum_{p(\bar{z})} \left[\ln p(\bar{z}) - \ln \left|\left| \frac{\partial \bar{G}(\bar{z})}{\partial \bar{z}} \right|\right| - \ln f(\bar{x}) \right] + NI_f$$

- ... where p is the sampling prior, G is the generative network, and f is the function to be integrated. NIf is a constant we can ignore in training.
- For deep networks, G (using proper activation functions, etc.), we may use this relationship for the loss function in stochastic gradient descent provided f is differentiable and a function we can evaluate at a point, etc.
  - Plus a trick for getting a differentiable representation of the determinant!
- Sampling from the trained network is straightforward.

Josh Bendavid (Caltech/LPC)

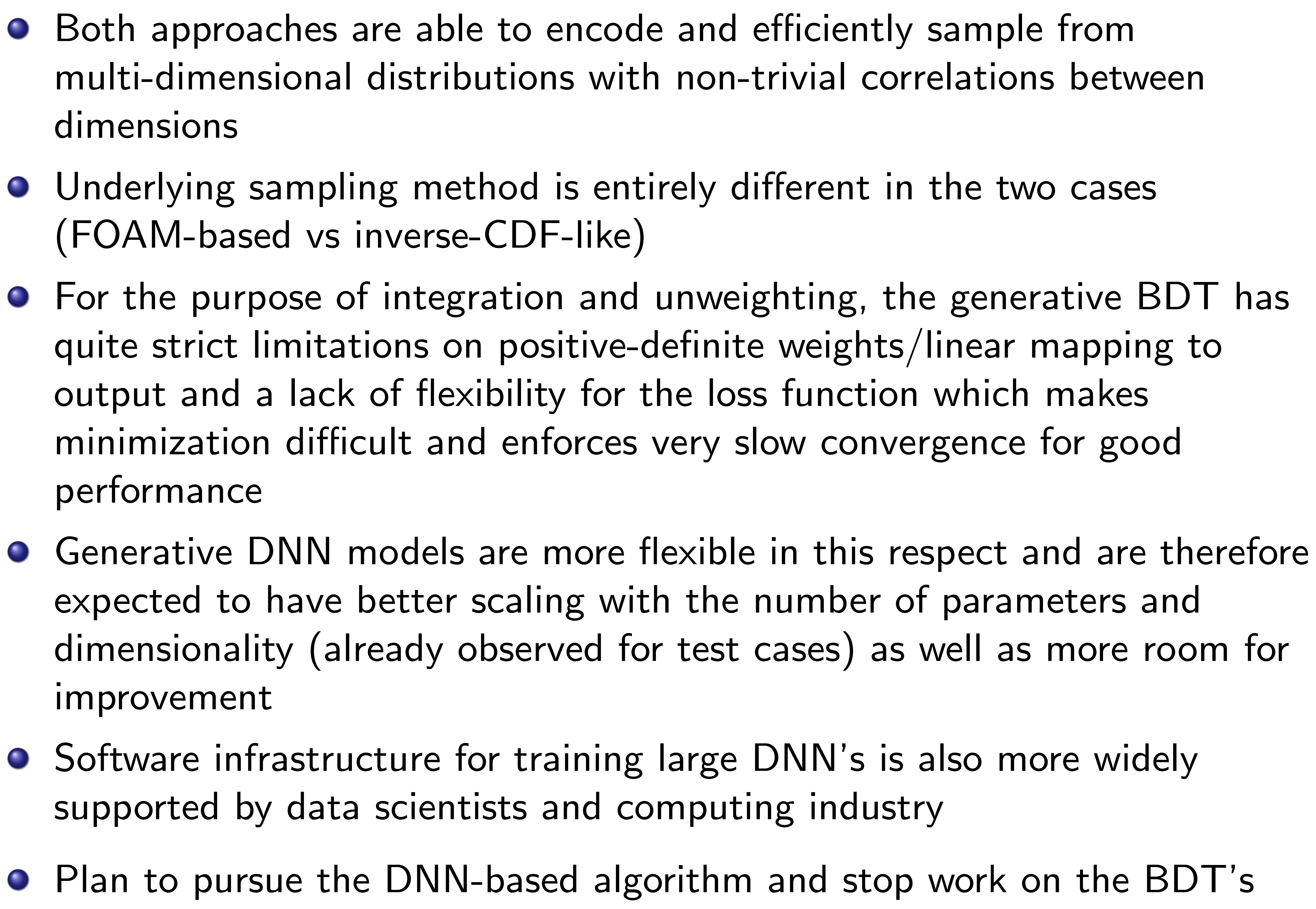
Function comparisons

Algorithm	# of Func. Evals	$\sigma_w / < w >$	σ_I / I
			(2e6 add. evts)
VEGAS	300,000	2.820	$\pm 2.0 \times 10^{-3}$
Foam	3,855,289	0.319	$\pm 2.3 \times 10^{-4}$
GBRIntegrator	300,000	0.082	$\pm 5.8 \times 10^{-5}$
GBRIntegrator (staged)	300,000	0.077	$\pm 5.4 \times 10^{-5}$
Generative DNN	294,912	0.083	$\pm 5.9 \times 10^{-5}$
Generative DNN (staged)	294,912	0.030	$\pm 2.1 \times 10^{-5}$

- Interesting limitation: Probability density for generative DNN model can **not** be evaluated for an arbitrary phase space point $\bar{x}$, since one needs to know the corresponding point in the prior space $\bar{z}$, and the model is not trivially invertible
- Not a problem for integration or unweighting where all the phase space points are anyways generated by sampling from the prior

Josh Bendavid (Caltech/LPC)

BDT vs NNs



- Both approaches are able to encode and efficiently sample from multi-dimensional distributions with non-trivial correlations between dimensions
- Underlying sampling method is entirely different in the two cases (FOAM-based vs inverse-CDF-like)
- For the purpose of integration and unweighting, the generative BDT has quite strict limitations on positive-definite weights/linear mapping to output and a lack of flexibility for the loss function which makes minimization difficult and enforces very slow convergence for good performance
- Generative DNN models are more flexible in this respect and are therefore expected to have better scaling with the number of parameters and dimensionality (already observed for test cases) as well as more room for improvement
- Software infrastructure for training large DNN's is also more widely supported by data scientists and computing industry
- Plan to pursue the DNN-based algorithm and stop work on the BDT's

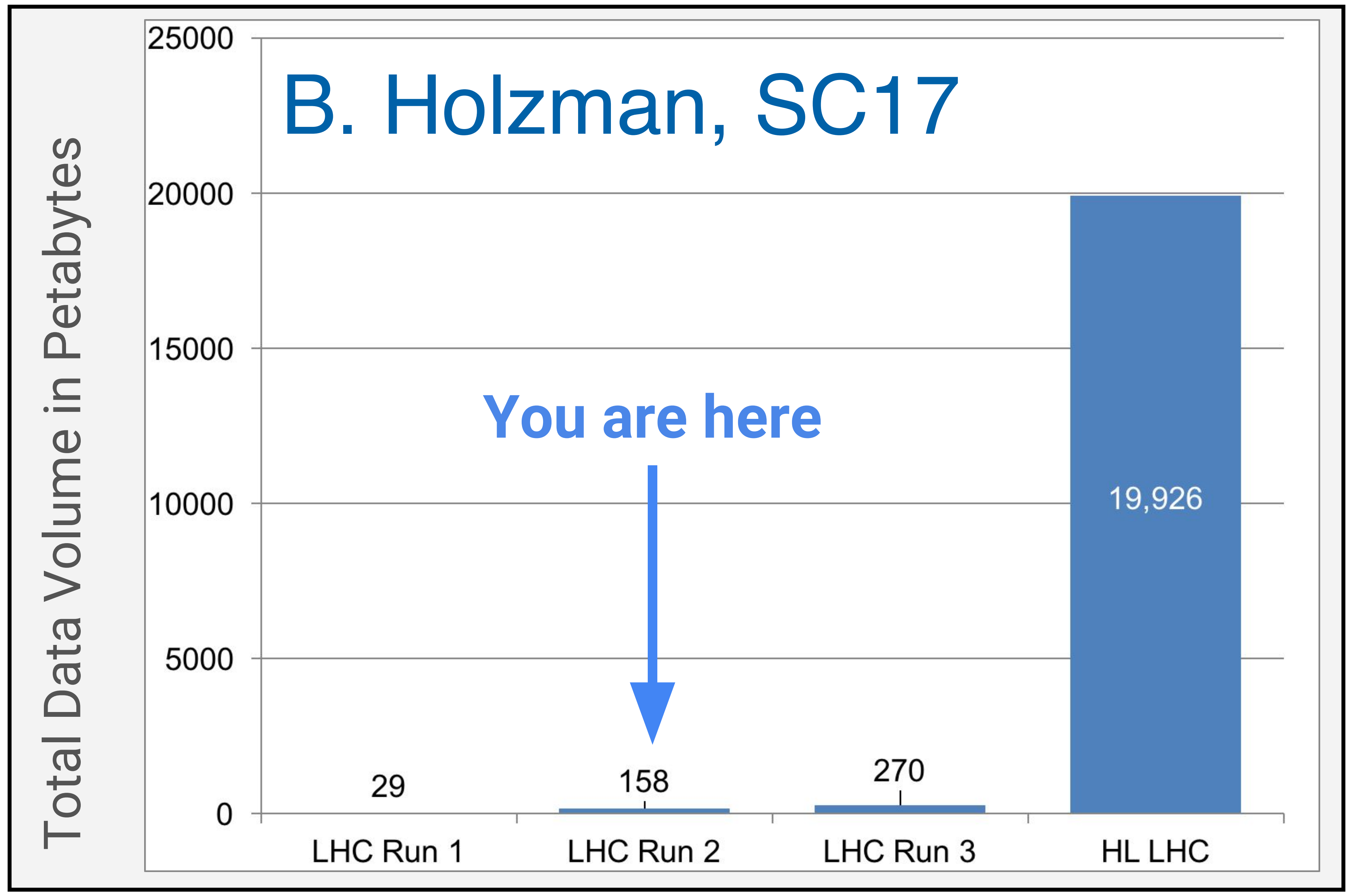
Josh Bendavid (Caltech/LPC)

Learned generators

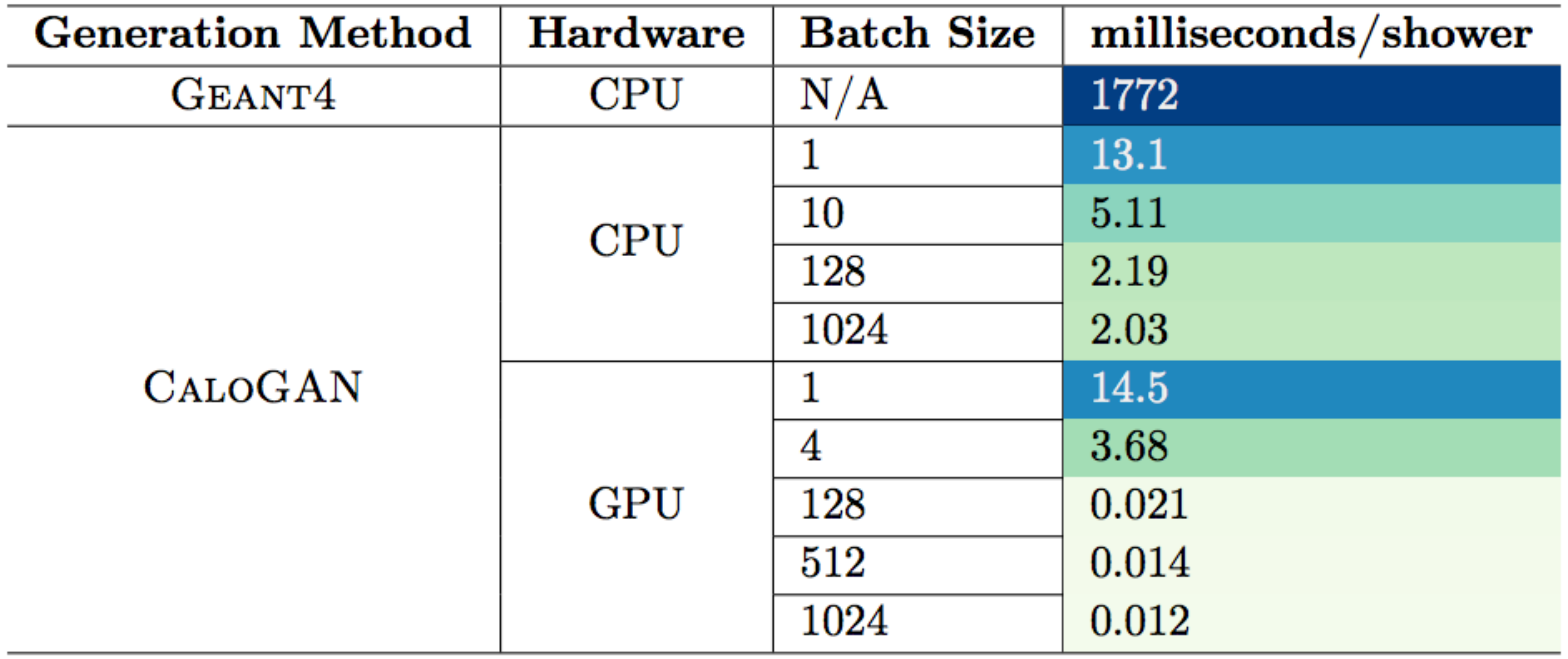
- http://colah.github.io/posts/2015-08-Understanding-LSTMs/
- http://karpathy.github.io/2015/05/21/rnn-effectiveness/
- https://arxiv.org/abs/1705.02355
- https://arxiv.org/abs/1702.00748
- https://arxiv.org/abs/1505.07818

Generative models for simulation

- Future simulation needs (e.g., HL-LHC) appear likely to outstrip even optimistic resource projections.
  - Requires creative, "outside the box" thinking.
- Shower libraries face problems rooted in incompleteness and heavy data access.
- Generative models offer a potentially incredible speed-up along with better flexibility by modeling very complex distributions.

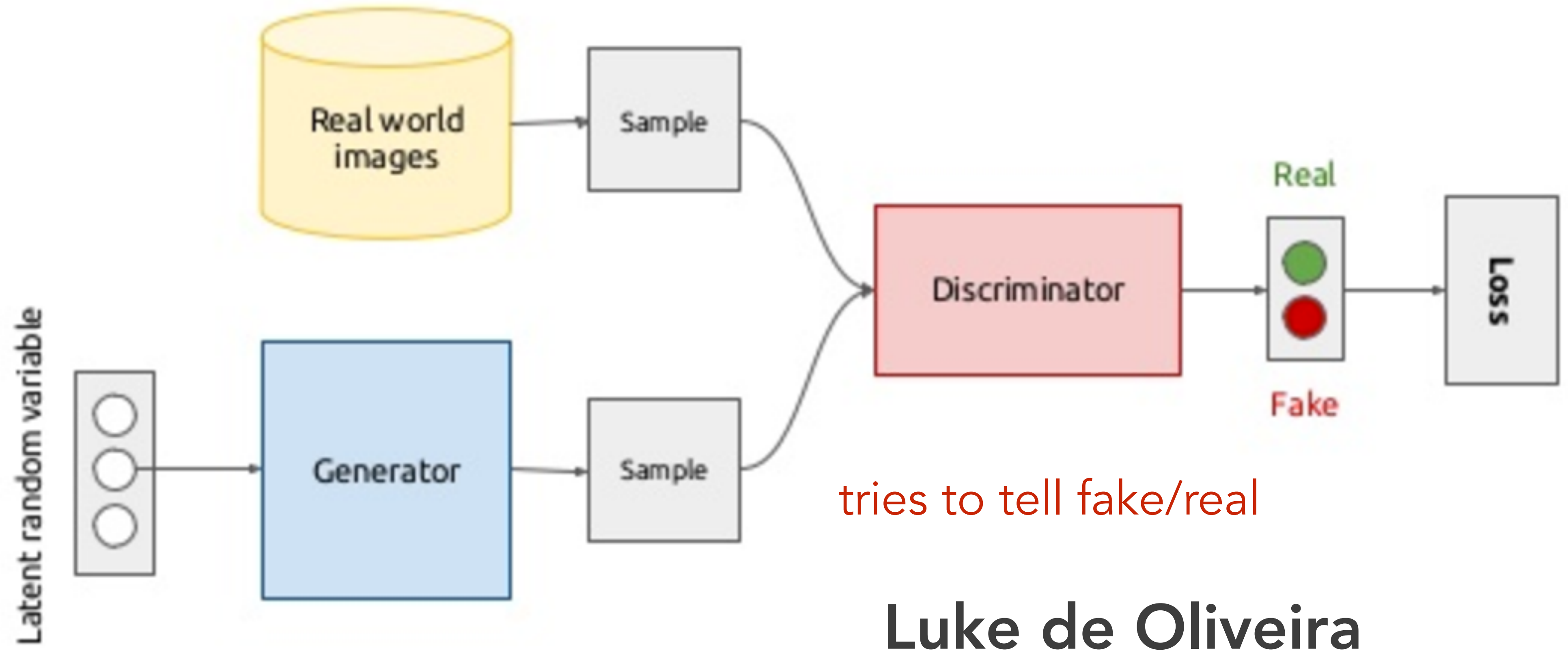

B. Holzman, SC17

You are here



Generation Method	Hardware	Batch Size	milliseconds/shower
GEANT4	CPU	N/A	1772
CALOGAN	CPU	1	13.1
		10	5.11
		128	2.19
		1024	2.03
	GPU	1	14.5
		4	3.68
		128	0.021
		512	0.014
		1024	0.012

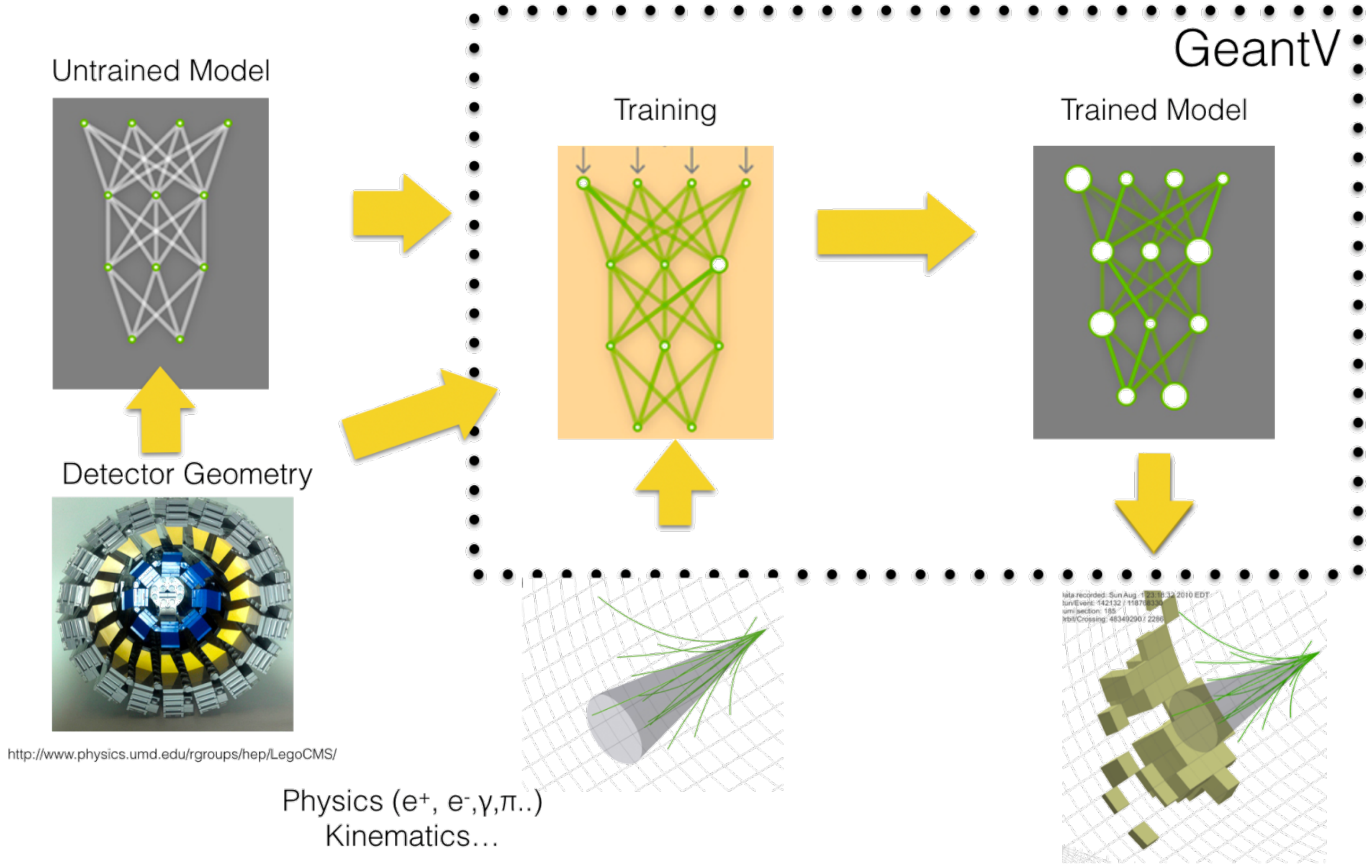
Michela Paganini*, Luke de Oliveira, Ben Nachman DS@HEP 2017



tries to tell fake/real

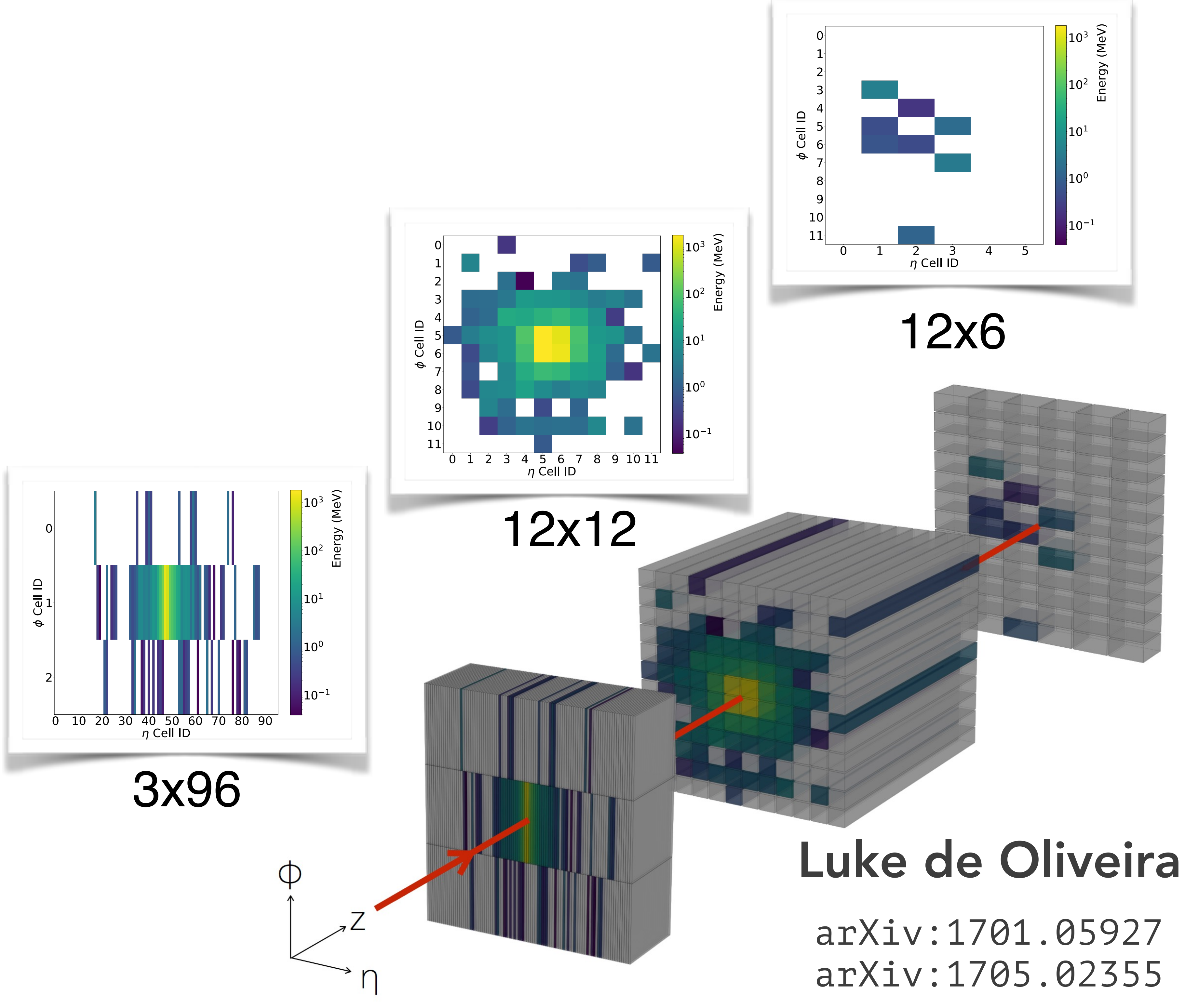
tries to produce real looking samples

Luke de Oliveira



Sofia Vallecorsa*
for the GeantV project

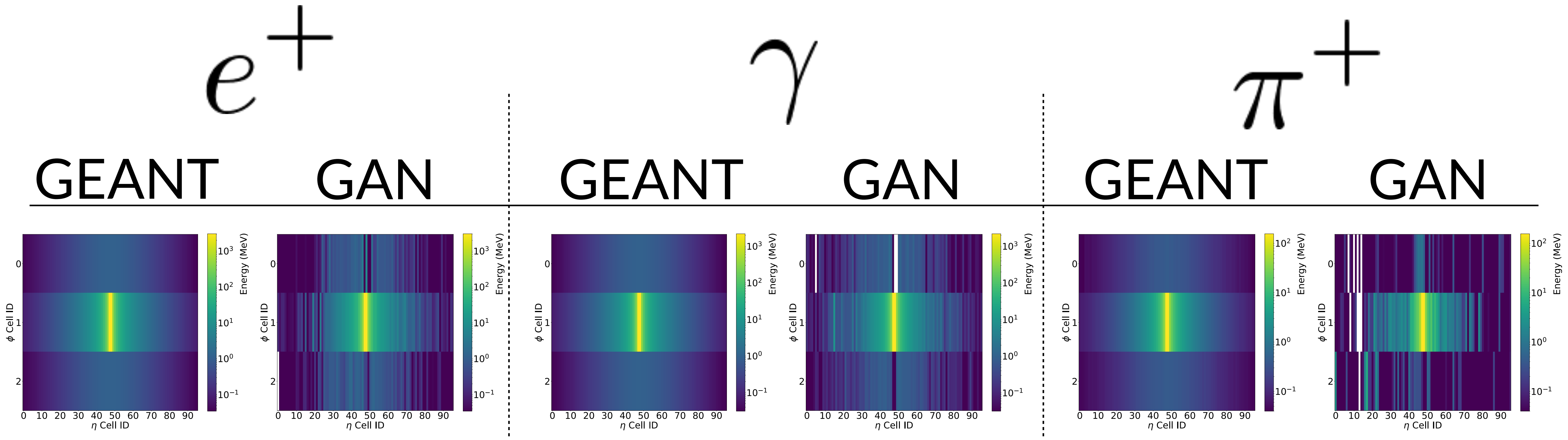
ACAT 2017



Luke de Oliveira

arXiv:1701.05927
arXiv:1705.02355

ACAT 2017



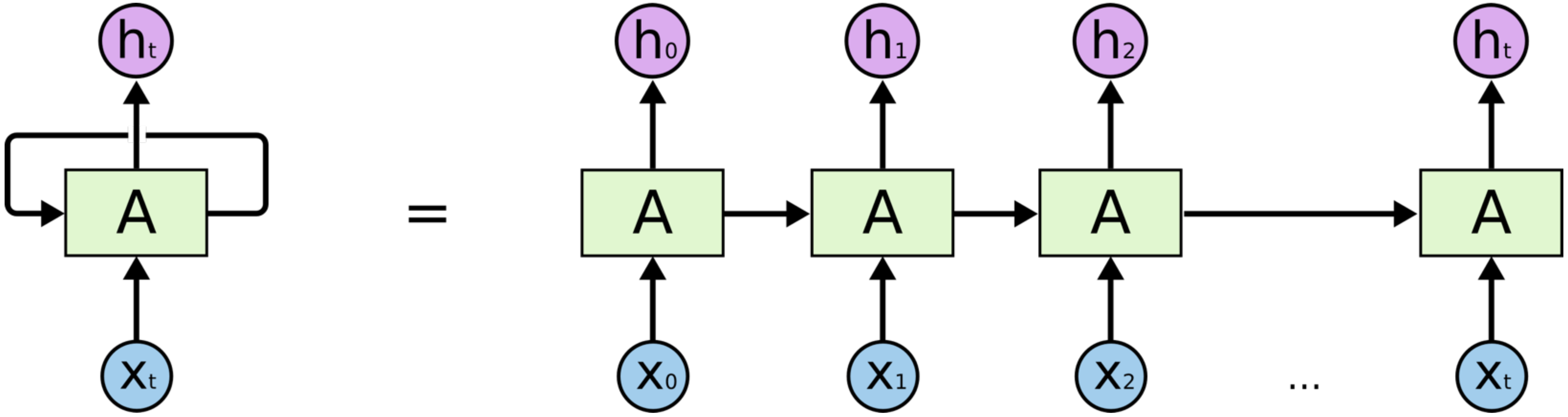
Michela Paganini*, Luke de Oliveira, Ben Nachman DS@HEP 2017

Also: arXiv 1806.11484

Language models == particle models?

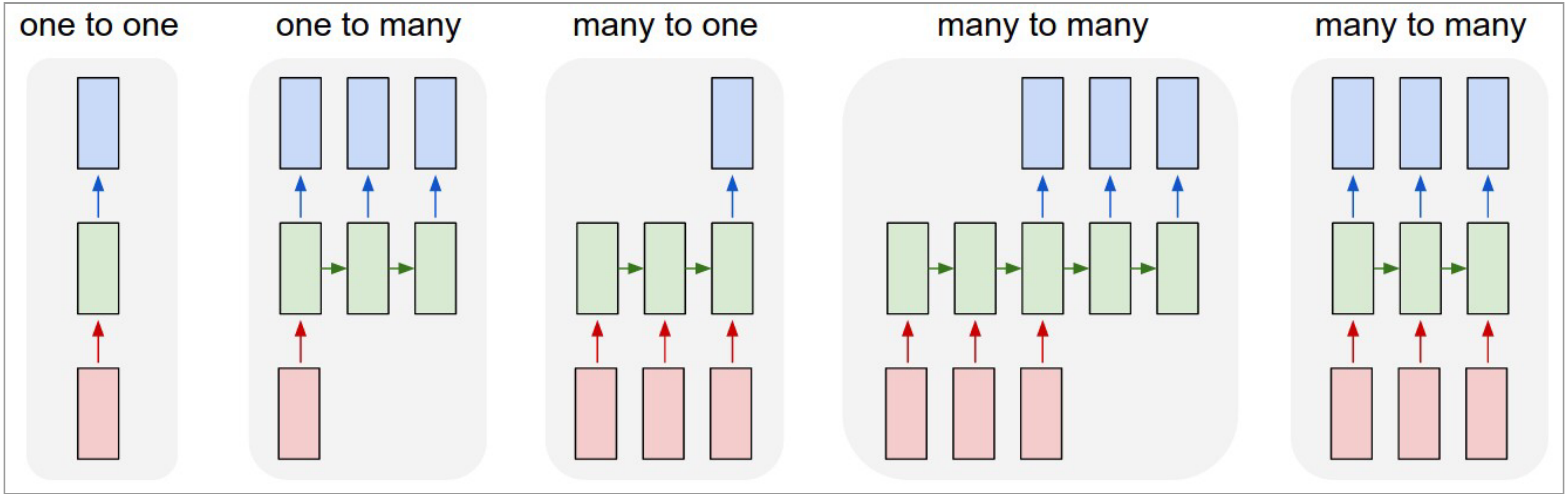
- Recent success in machine learning is dominated by two kinds of data:
  - visual data - images, video, etc. - we now have algorithms for parsing and analyzing this sort of very high-dimensional data, and
  - sequence data - speech recognition, language processing, game playing, etc. - all revolve around sequences of data and the patterns buried in these data.
- Can we map problems in physics into these domains?
  - Yes!
  - Image data is obvious and easy - see, for example, convolutional neural nets for event reconstruction and classification (e.g., J. Nowak's presentation yesterday on event classification in LArTPCs).
  - Sequence data is trickier... but several inspired applications at the LHC suggest we may be able to treat particle sequences in an event like words in a sentence. The semantics of ordering are different, but there are successful examples of leveraging this paradigm for event classification.

Recurrent networks for sequence analysis



- Recurrent neural networks can operate over sequences of vectors (in principle, of arbitrary length).
- The network structure explicitly contains input from its own output - it contains loops. These loops could theoretically extend back into infinity for all the examples the network is asked to operate on, but in practice, we truncate the series.
- This means we can effectively "unroll" the network, which makes the loop structure less mysterious.

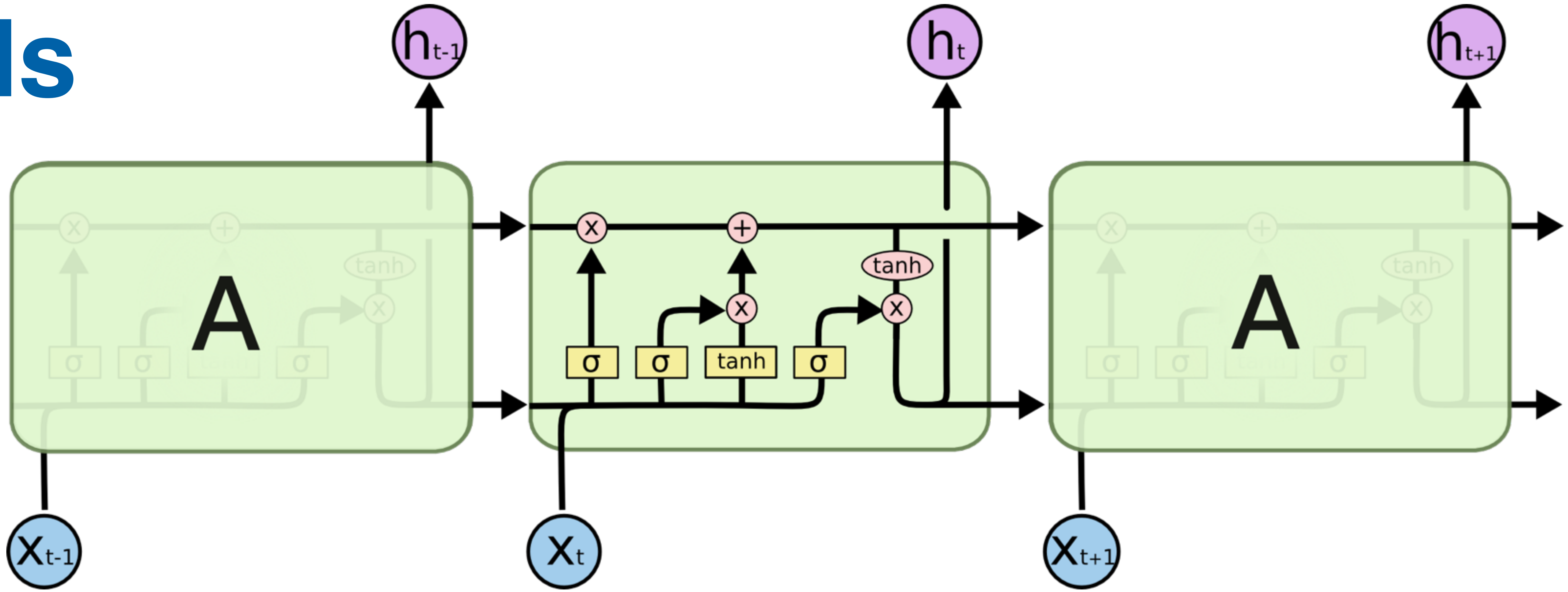
http://colah.github.io/posts/2015-08-Understanding-LSTMs/



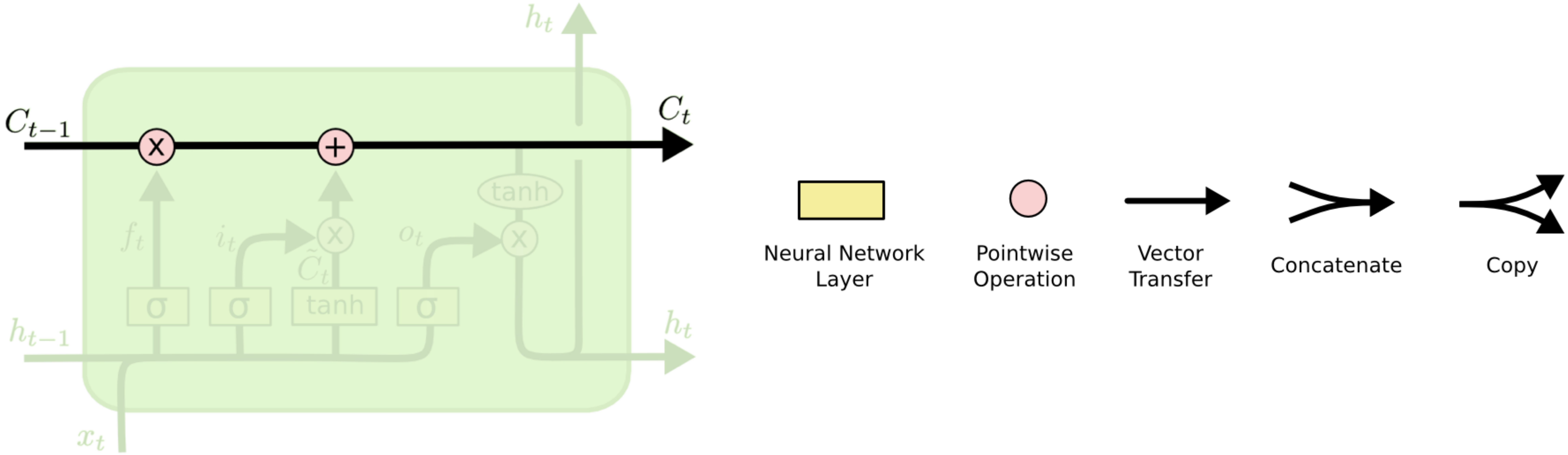
Each rectangle is a vector and arrows represent functions (e.g. matrix multiply). Input vectors are in red, output vectors are in blue and green vectors hold the RNN's state (more on this soon). From left to right: **(1)** Vanilla mode of processing without RNN, from fixed-sized input to fixed-sized output (e.g. image classification). **(2)** Sequence output (e.g. image captioning takes an image and outputs a sentence of words). **(3)** Sequence input (e.g. sentiment analysis where a given sentence is classified as expressing positive or negative sentiment). **(4)** Sequence input and sequence output (e.g. Machine Translation: an RNN reads a sentence in English and then outputs a sentence in French). **(5)** Synced sequence input and output (e.g. video classification where we wish to label each frame of the video). Notice that in every case are no pre-specified constraints on the lengths sequences because the recurrent transformation (green) is fixed and can be applied as many times as we like.

http://karpathy.github.io/2015/05/21/rnn-effectiveness/

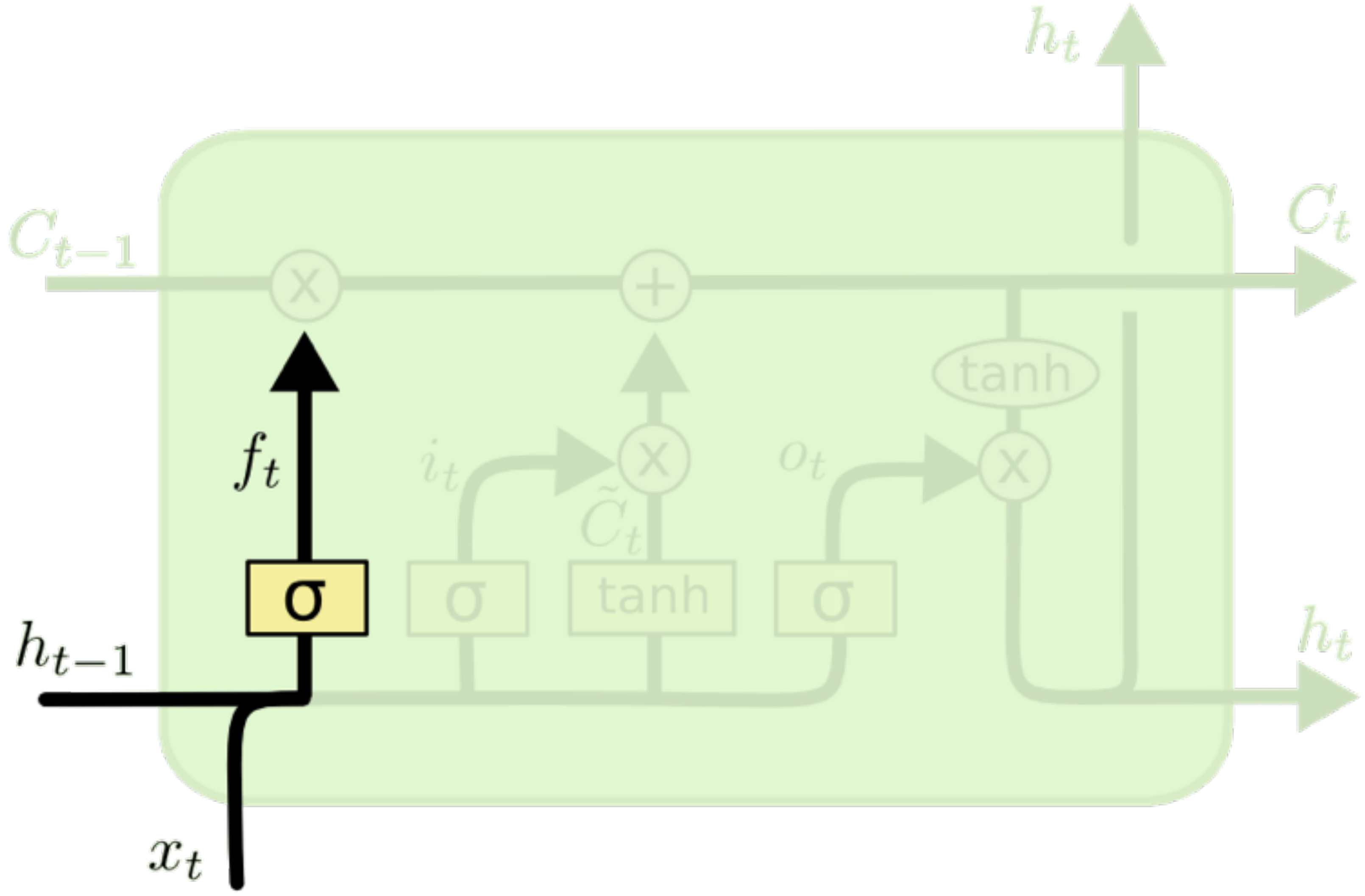
LSTMs



- In general, recurrent neural networks have trouble "remembering" details from more than a few steps back.
- This matters for us (eventually) because we would like to generate a sequence of particles (of arbitrary length). We need the network to remember what we've already generated.
- There are *many* solutions to this problem built around structures like the one above (LSTM - long short-term memory) - "memory" cells that bring extra copies of the information from previous steps in the sequence. LSTMs are important in language processing where order and context matter - this particular structure is not necessarily right for an event generator producing lists of particles, but we'll look at it as an example anyway...

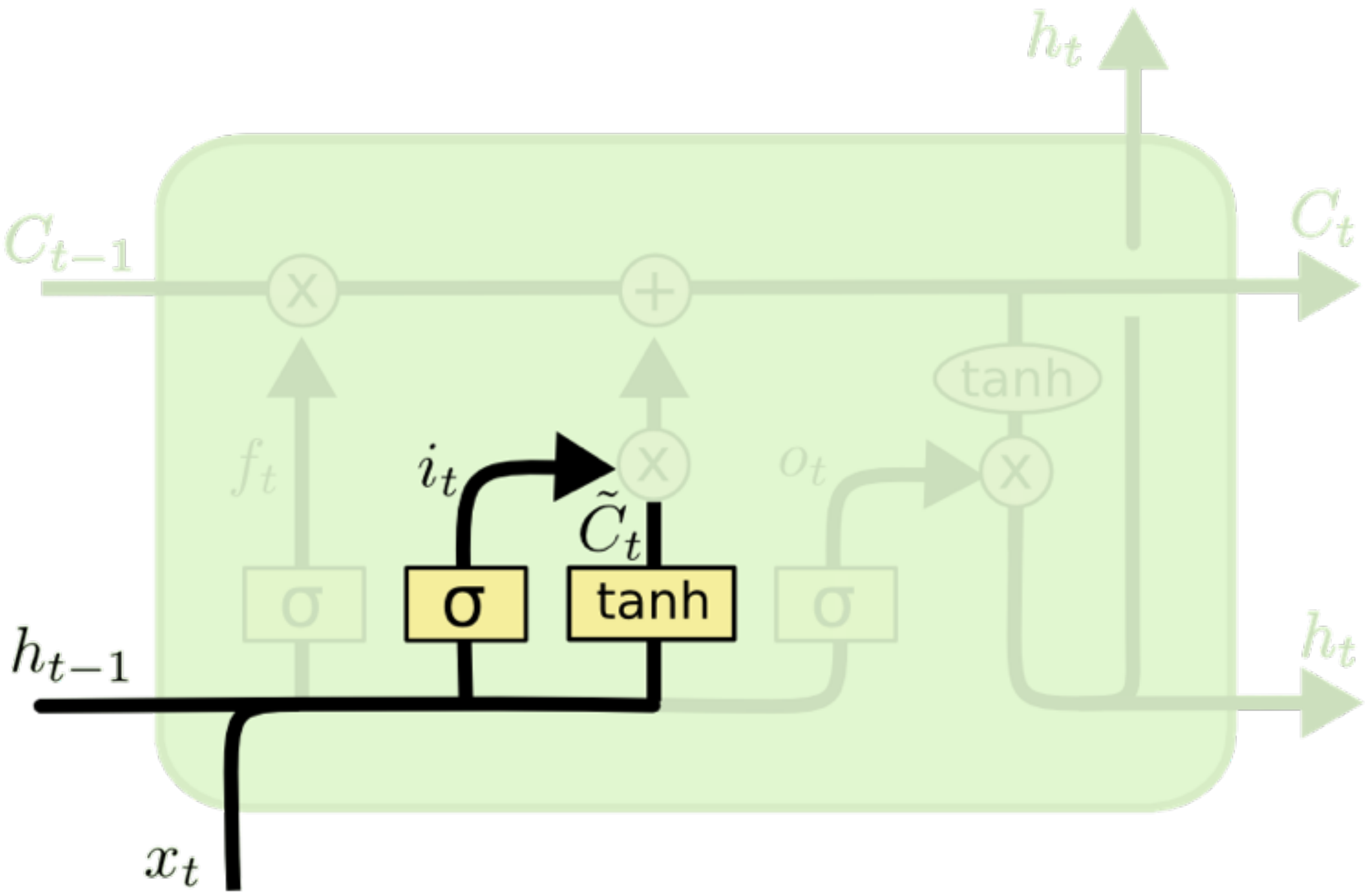


- The cell state (usually initialized to zero) runs through the entire chain, possibly with some linear interactions.
- Information can flow along unchanged - but the LSTM has the ability to add or subtract information passing along through the cell state.
- This is regulated by structures called *gates*.



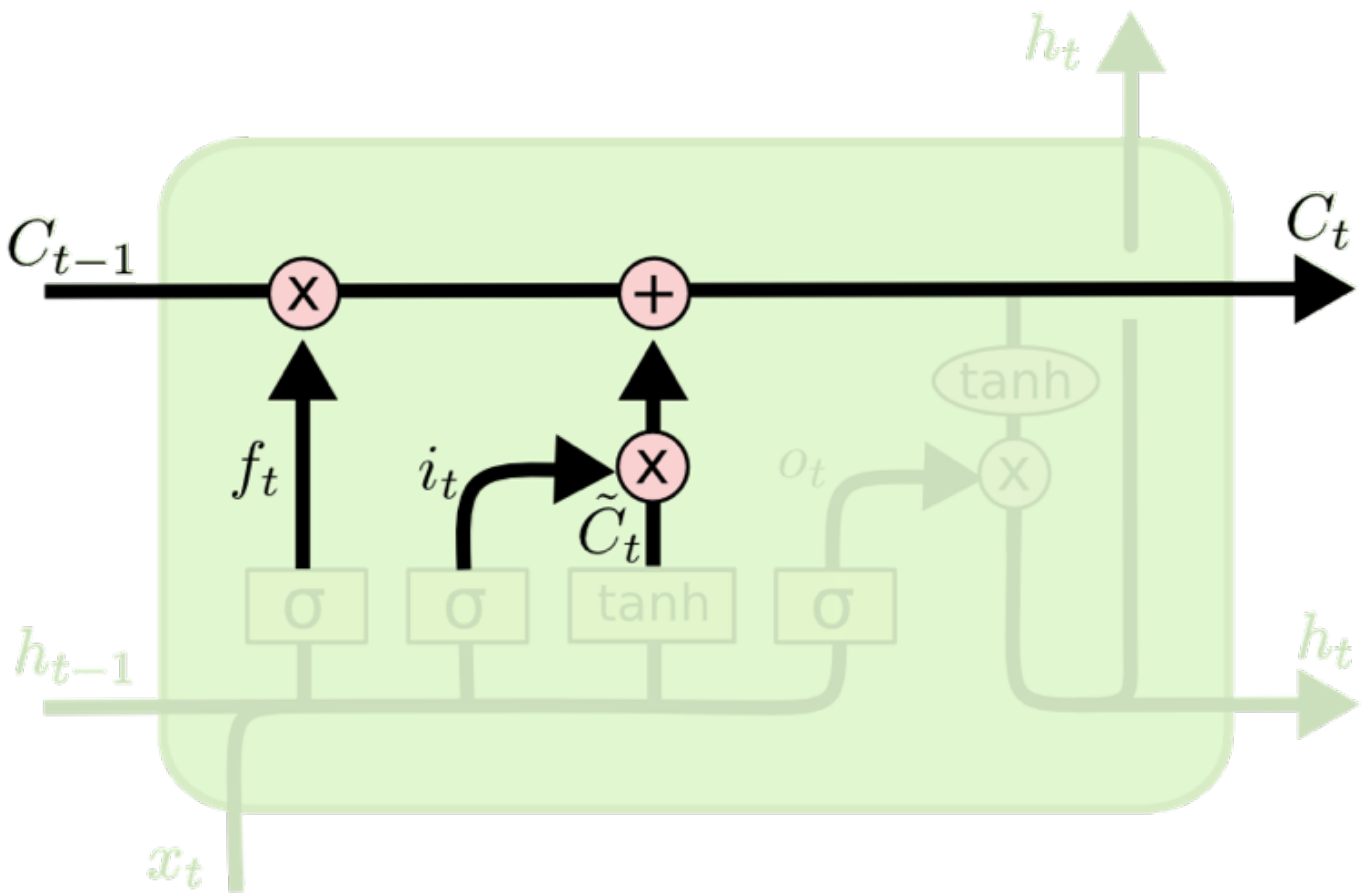
$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \ + \ b_f\right)$$

- Gates are composed of sigmoid activation functions (value from 0 to 1 controls how much of the component should come through - from nothing to everything) and point-wise multiplication.
- The first of these gates is the "forget gate layer", and consumes the output from the pervious recurrent cell call.
- In a language model this controls when the LSTM can forget about earlier words for context.
- In a particle context, the semantics are less obvious, but some information must be learned to always be retained (four momentum sums, for example!).



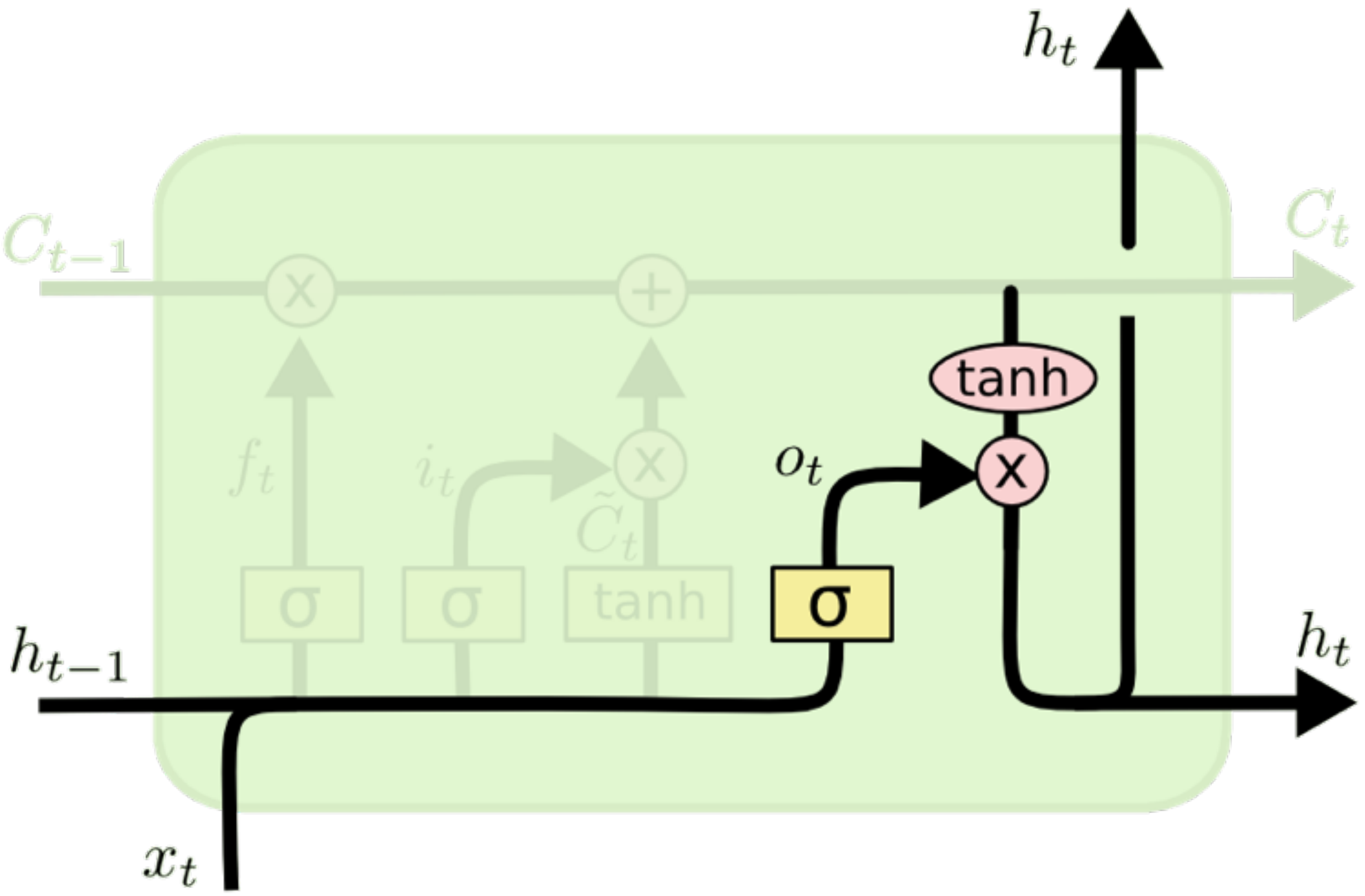
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$$

- Next an LSTM needs to decide what "new" information it wants to keep in the cell state.
- First, a sigmoid layer decides how *much* information to propagate into the state vector. Then, a tanh layer creates new candidate values based on the immediately preceding output that could be included in the state.
- Very often in a language model these gates control information replacement when old information is forgotten. In a particle context, they represent a change in focus as we process the sequence.



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

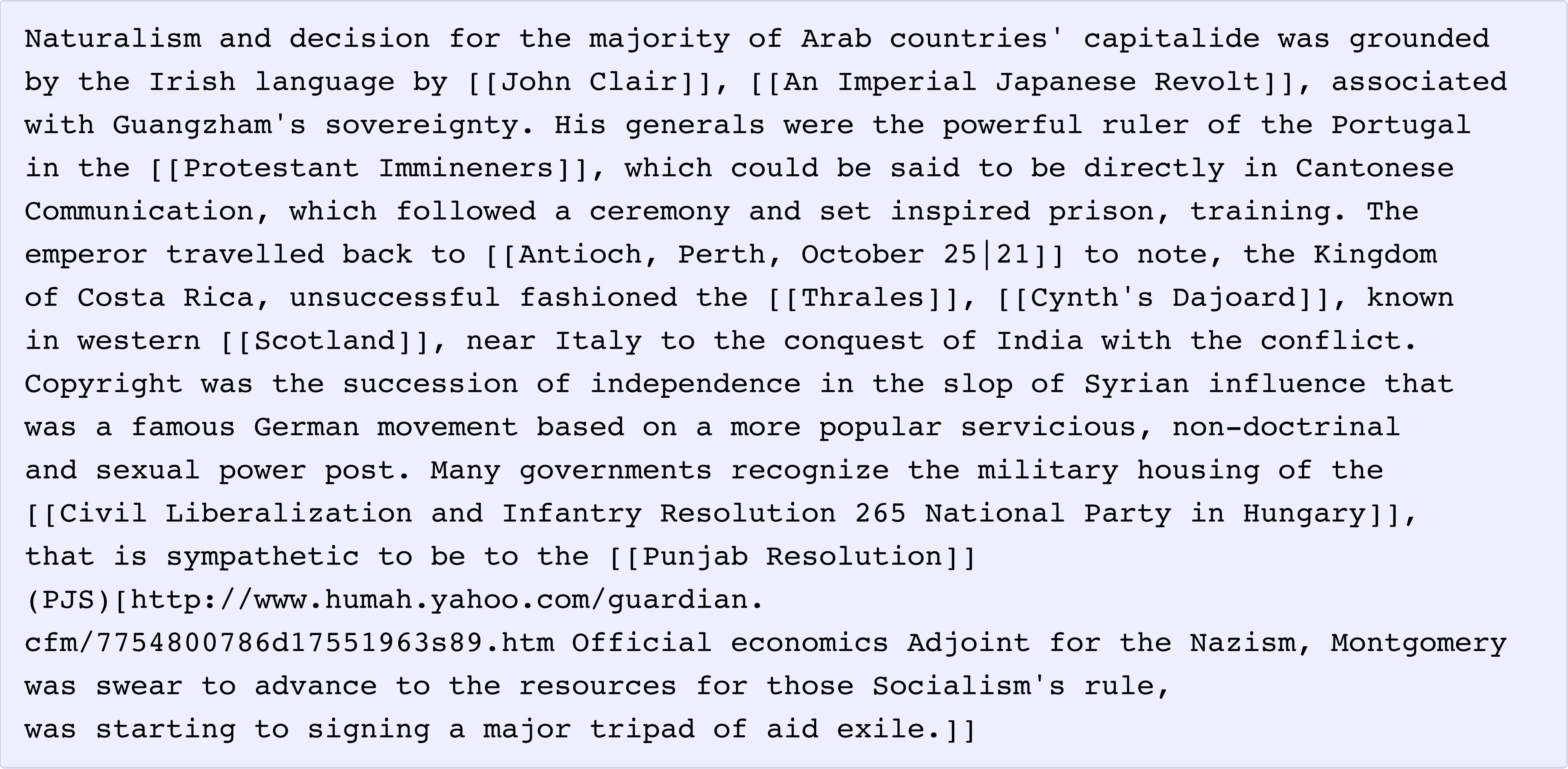
- Once we have decided what to drop and what to add to the running cell state, we must apply the changes to the state vector.
- We multiply the cell state by the output of a sigmoid to modulate older information, then we add new information (which was multiplied by sigmoid and tanh layers to modulate new information and allow additive and subtractive changes).



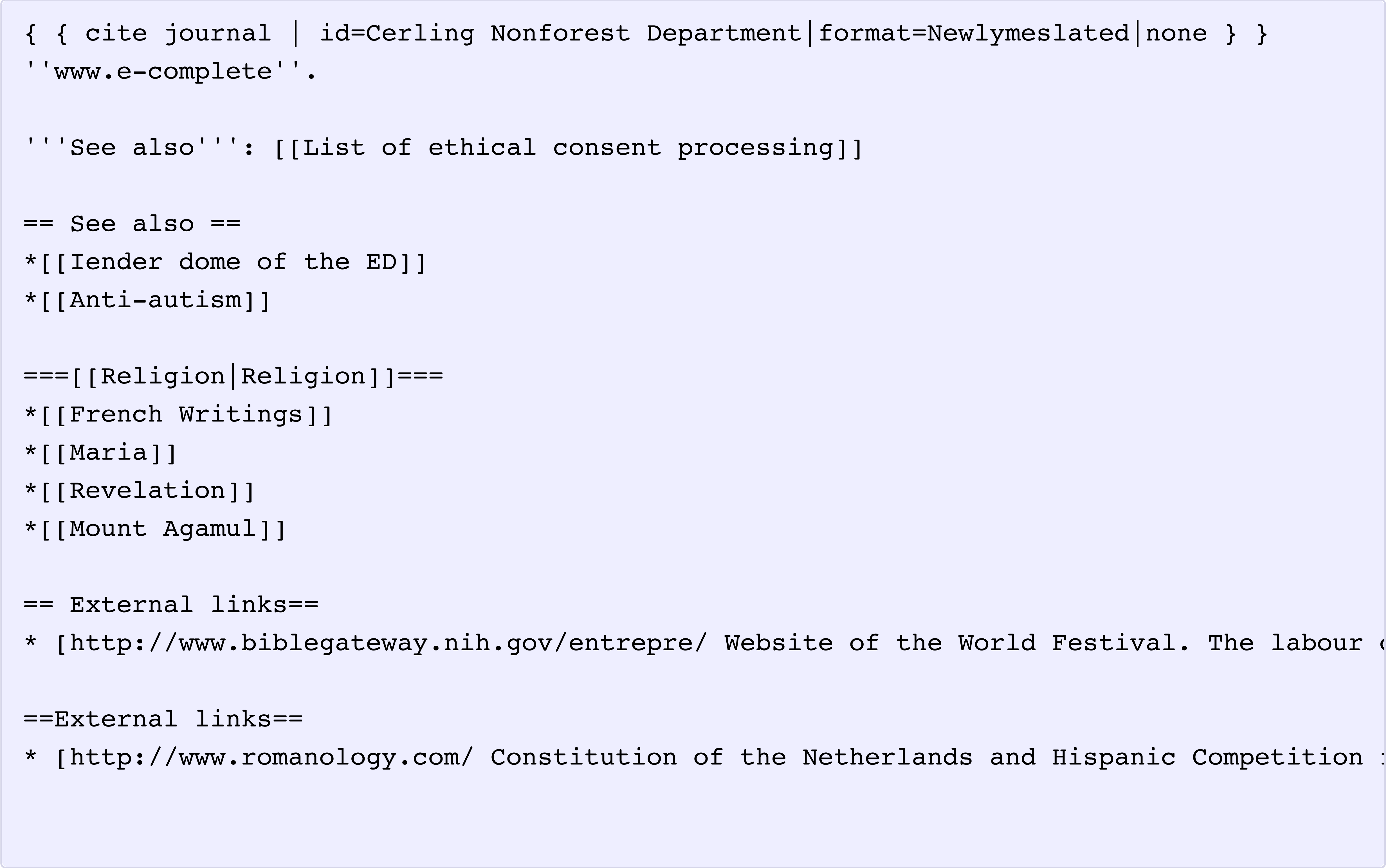
$$o_t = \sigma\left(W_o \ [h_{t-1}, x_t] \ + \ b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

- Then, we regulate output by using a sigmoid layer to decide which values make it to the output, and combine it with the cell state, after passing those values through a tanh layer.
- This output is split and passed out as a classifier value or a generated value and also passed back into the LSTM.

RNNs can be easily trained to predict certain types of sequences based on an input seed / sequence. Here, for example is an RNN trained by consuming text from Wikipedia:

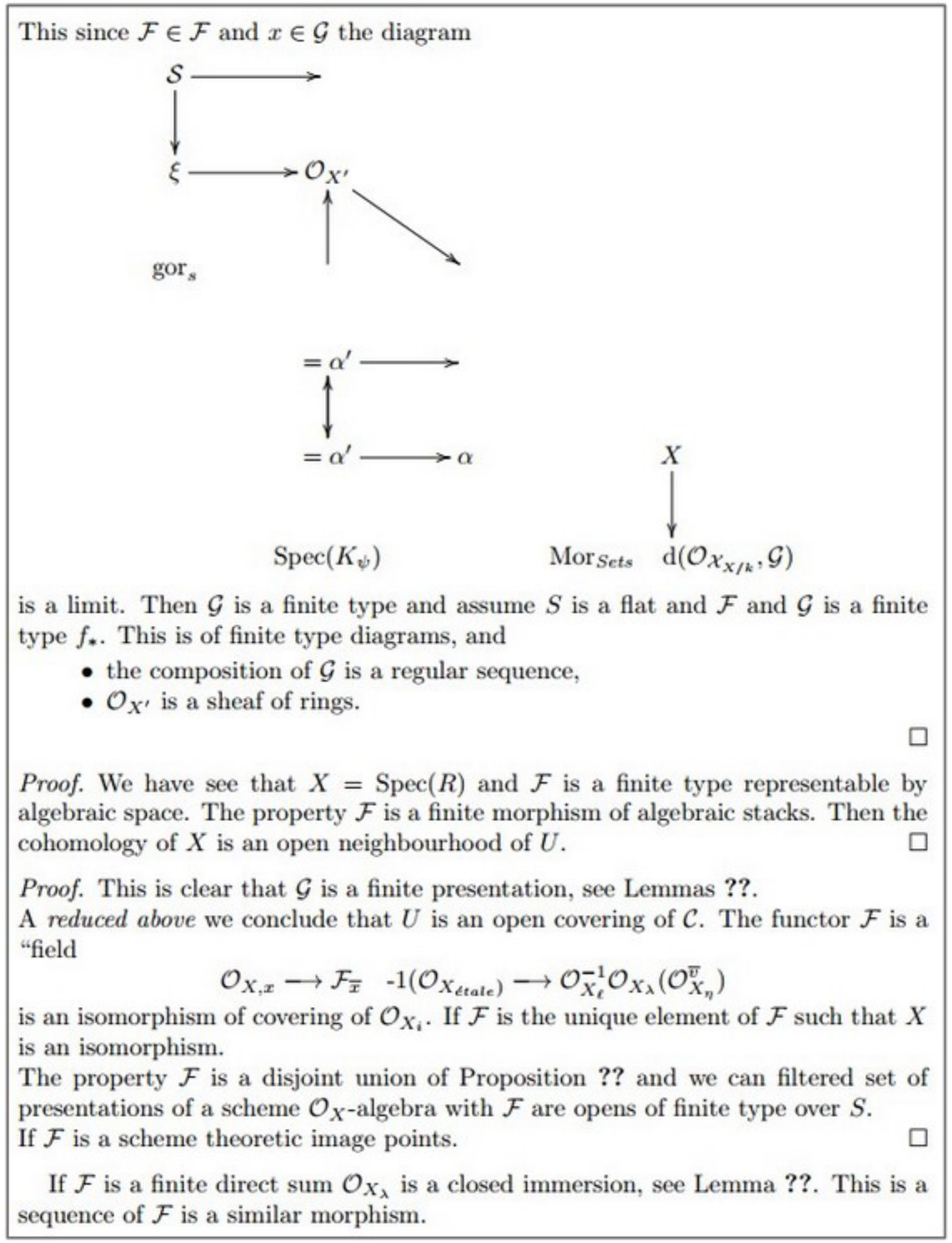


```
Naturalism and decision for the majority of Arab countries' capitalide was grounded
by the Irish language by [[John Clair]], [[An Imperial Japanese Revolt]], associated
with Guangzham's sovereignty. His generals were the powerful ruler of the Portugal
in the [[Protestant Immineners]], which could be said to be directly in Cantonese
Communication, which followed a ceremony and set inspired prison, training. The
emperor travelled back to [[Antioch, Perth, October 25|21]] to note, the Kingdom
of Costa Rica, unsuccessful fashioned the [[Thrales]], [[Cynth's Dajoard]], known
in western [[Scotland]], near Italy to the conquest of India with the conflict.
Copyright was the succession of independence in the slop of Syrian influence that
was a famous German movement based on a more popular servicious, non-doctrinal
and sexual power post. Many governments recognize the military housing of the
[[Civil Liberalization and Infantry Resolution 265 National Party in Hungary]],
that is sympathetic to be to the [[Punjab Resolution]]
(PJS)[http://www.humah.yahoo.com/guardian.
cfm/7754800786d17551963s89.htm Official economics Adjoint for the Nazism, Montgomery
was swear to advance to the resources for those Socialism's rule,
was starting to signing a major tripad of aid exile.]]
```



```
{ { cite journal | id=Cerling Nonforest Department|format=Newlymeslated|none } }
''www.e-complete''.

'''See also''': [[List of ethical consent processing]]

== See also ==
*[[Iender dome of the ED]]
*[[Anti-autism]]

===[[Religion|Religion]]===
*[[French Writings]]
*[[Maria]]
*[[Revelation]]
*[[Mount Agamul]]

== External links==
* [http://www.biblegateway.nih.gov/entrepre/ Website of the World Festival. The labour

==External links==
* [http://www.romanology.com/ Constitution of the Netherlands and Hispanic Competition
```

Here is an example based on LaTeX files for a mathematics textbook.



This since $\mathcal{F} \in \mathcal{F}$ and $x \in \mathcal{G}$ the diagram

$$\begin{tikzcd}
\mathcal{S} \arrow[r] \arrow[d] & {} & & \\
\xi \arrow[r] & \mathcal{O}_{X'} \arrow[rd] & & \\
\mathrm{gor}_s & {} \arrow[u] & {} & \\
& =\alpha' \arrow[r] \arrow[d, leftrightarrow] & {} & \\
& =\alpha' \arrow[r] & \alpha & X \arrow[d] \\
& \mathrm{Spec}(K_\psi) & & \mathrm{Mor}_{Sets} \quad \mathrm{d}(\mathcal{O}_{\mathcal{X}_{X/k}}, \mathcal{G})
\end{tikzcd}$$

is a limit. Then $\mathcal{G}$ is a finite type and assume S is a flat and $\mathcal{F}$ and $\mathcal{G}$ is a finite type f_*. This is of finite type diagrams, and

- the composition of $\mathcal{G}$ is a regular sequence,
- $\mathcal{O}_{X'}$ is a sheaf of rings.

□

Proof. We have see that $X = \mathrm{Spec}(R)$ and $\mathcal{F}$ is a finite type representable by algebraic space. The property $\mathcal{F}$ is a finite morphism of algebraic stacks. Then the cohomology of X is an open neighbourhood of U. □

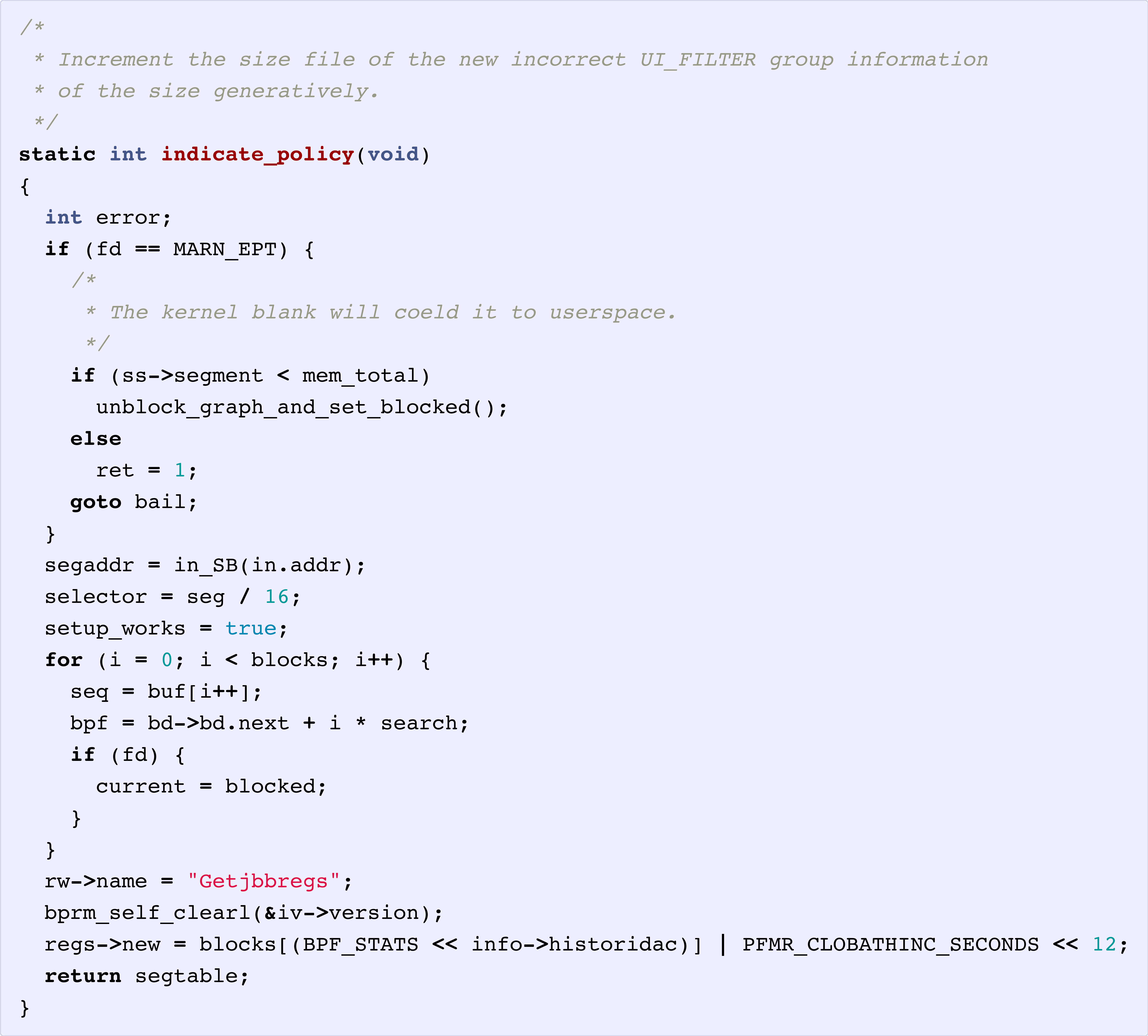
Proof. This is clear that $\mathcal{G}$ is a finite presentation, see Lemmas ??.
A *reduced above* we conclude that U is an open covering of $\mathcal{C}$. The functor $\mathcal{F}$ is a "field

$$\mathcal{O}_{X,x} \longrightarrow \mathcal{F}_{\overline{x}} \quad \text{-1}(\mathcal{O}_{X_{\acute{e}tale}}) \longrightarrow \mathcal{O}_{X_\ell}^{-1}\mathcal{O}_{X_\lambda}(\mathcal{O}_{X_\eta}^{\overline{v}})$$

is an isomorphism of covering of $\mathcal{O}_{X_i}$. If $\mathcal{F}$ is the unique element of $\mathcal{F}$ such that X is an isomorphism.
The property $\mathcal{F}$ is a disjoint union of Proposition ?? and we can filtered set of presentations of a scheme $\mathcal{O}_X$-algebra with $\mathcal{F}$ are opens of finite type over S.
If $\mathcal{F}$ is a scheme theoretic image points. □

If $\mathcal{F}$ is a finite direct sum $\mathcal{O}_{X_\lambda}$ is a closed immersion, see Lemma ??. This is a sequence of $\mathcal{F}$ is a similar morphism.

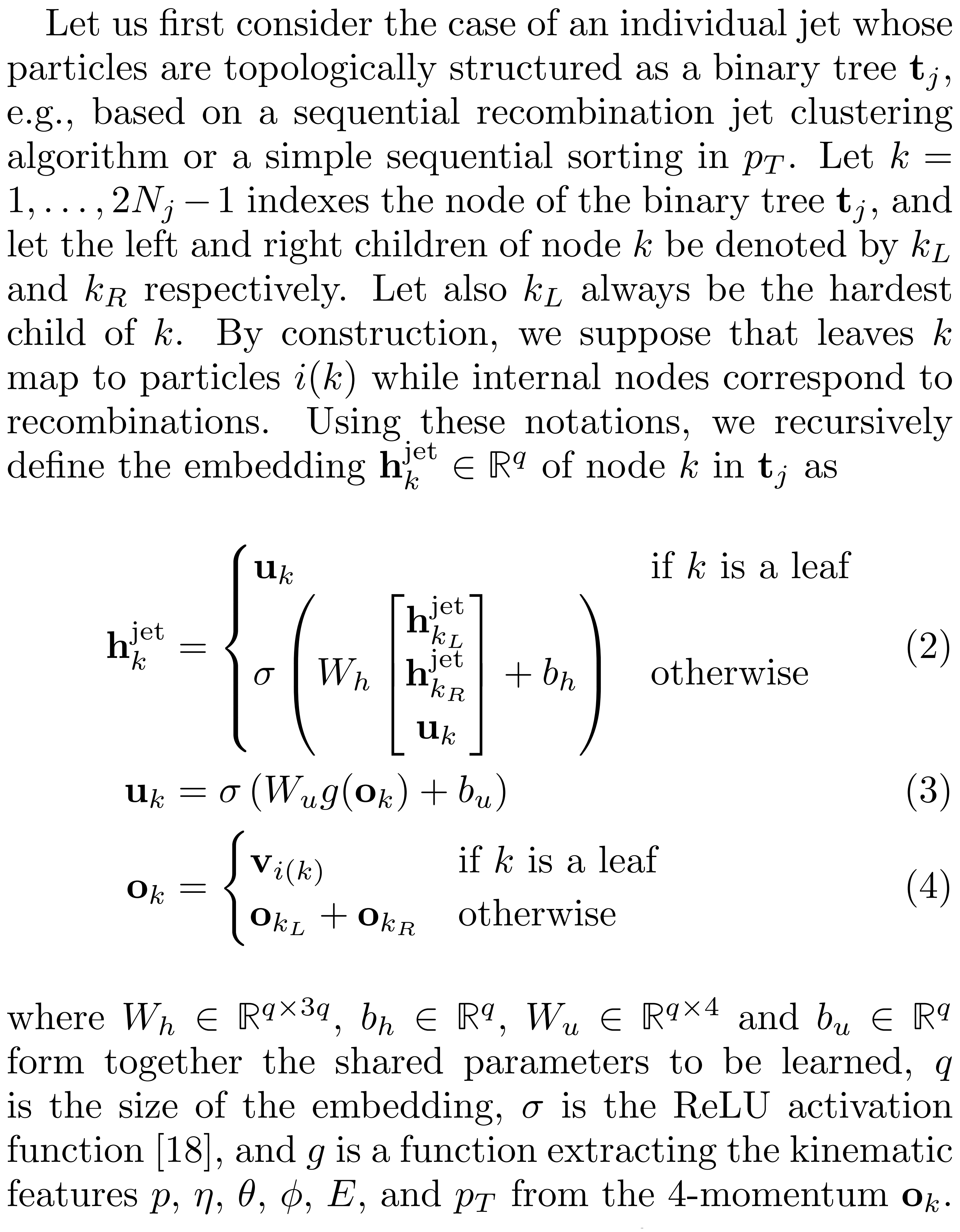
```
\begin{proof}
We may assume that $\mathcal{I}$ is an abelian sheaf on $\mathcal{C}$.
\item Given a morphism $\Delta : \mathcal{F} \to \mathcal{I}$
is an injective and let $\mathfrak q$ be an abelian sheaf on $X$.
Let $\mathcal{F}$ be a fibered complex. Let $\mathcal{F}$ be a category.
\begin{enumerate}
\item \hyperref[setain-construction-phantom]{Lemma}
\label{lemma-characterize-quasi-finite}
Let $\mathcal{F}$ be an abelian quasi-coherent sheaf on $\mathcal{C}$.
Let $\mathcal{F}$ be a coherent $\mathcal{O}_X$-module. Then
$\mathcal{F}$ is an abelian catenary over $\mathcal{C}$.
\item The following are equivalent
\begin{enumerate}
\item $\mathcal{F}$ is an $\mathcal{O}_X$-module.
\end{lemma}
```



```c
/*
 * Increment the size file of the new incorrect UI_FILTER group information
 * of the size generatively.
 */
static int indicate_policy(void)
{
  int error;
  if (fd == MARN_EPT) {
    /*
     * The kernel blank will coeld it to userspace.
     */
    if (ss->segment < mem_total)
      unblock_graph_and_set_blocked();
    else
      ret = 1;
    goto bail;
  }
  segaddr = in_SB(in.addr);
  selector = seg / 16;
  setup_works = true;
  for (i = 0; i < blocks; i++) {
    seq = buf[i++];
    bpf = bd->bd.next + i * search;
    if (fd) {
      current = blocked;
    }
  }
  rw->name = "Getjbbregs";
  bprm_self_clearl(&iv->version);
  regs->new = blocks[(BPF_STATS << info->historidac)] | PFMR_CLOBATHINC_SECONDS << 12;
  return segtable;
}
```

Here is an example based on the source code for the Linux kernel.

Particle sequences are like words in a language model?



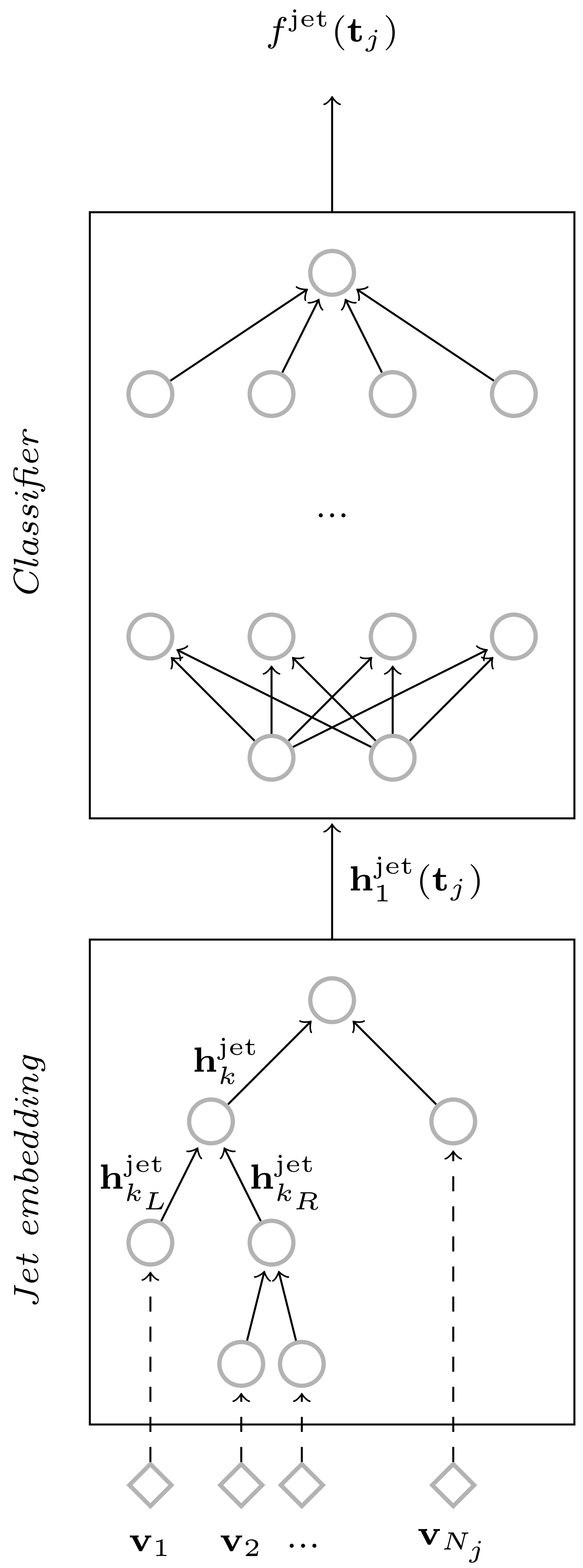
Let us first consider the case of an individual jet whose particles are topologically structured as a binary tree $\mathbf{t}_j$, e.g., based on a sequential recombination jet clustering algorithm or a simple sequential sorting in p_T. Let $k = 1, \dots, 2N_j - 1$ indexes the node of the binary tree $\mathbf{t}_j$, and let the left and right children of node k be denoted by k_L and k_R respectively. Let also k_L always be the hardest child of k. By construction, we suppose that leaves k map to particles $i(k)$ while internal nodes correspond to recombinations. Using these notations, we recursively define the embedding $\mathbf{h}_k^{\text{jet}} \in \mathbb{R}^q$ of node k in $\mathbf{t}_j$ as

$$\mathbf{h}_k^{\text{jet}} = \begin{cases} \mathbf{u}_k & \text{if } k \text{ is a leaf} \\ \sigma\left(W_h \begin{bmatrix} \mathbf{h}_{k_L}^{\text{jet}} \\ \mathbf{h}_{k_R}^{\text{jet}} \\ \mathbf{u}_k \end{bmatrix} + b_h \right) & \text{otherwise} \end{cases} \quad (2)$$

$$\mathbf{u}_k = \sigma\left(W_u g(\mathbf{o}_k) + b_u \right) \quad (3)$$

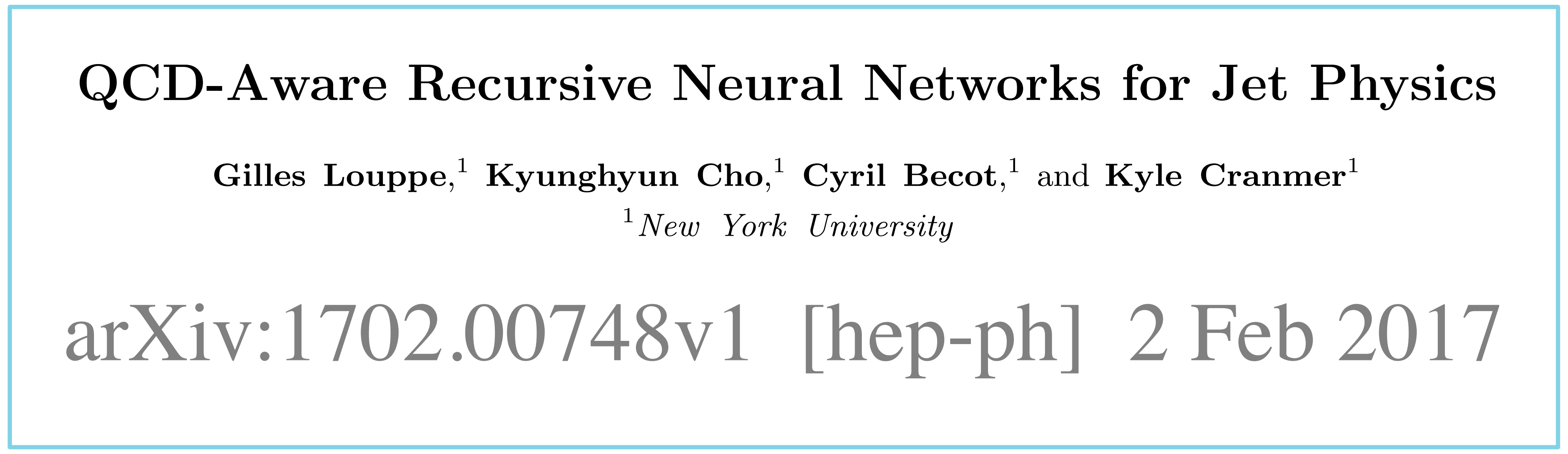
$$\mathbf{o}_k = \begin{cases} \mathbf{v}_{i(k)} & \text{if } k \text{ is a leaf} \\ \mathbf{o}_{k_L} + \mathbf{o}_{k_R} & \text{otherwise} \end{cases} \quad (4)$$

where $W_h \in \mathbb{R}^{q \times 3q}$, $b_h \in \mathbb{R}^q$, $W_u \in \mathbb{R}^{q \times 4}$ and $b_u \in \mathbb{R}^q$ form together the shared parameters to be learned, q is the size of the embedding, σ is the ReLU activation function [18], and g is a function extracting the kinematic features p, η, θ, ϕ, E, and p_T from the 4-momentum $\mathbf{o}_k$.



$f^{\text{jet}}(\mathbf{t}_j)$
Classifier
...
$\mathbf{h}_1^{\text{jet}}(\mathbf{t}_j)$
Jet embedding
$\mathbf{h}_k^{\text{jet}}$
$\mathbf{h}_{k_L}^{\text{jet}}$
$\mathbf{h}_{k_R}^{\text{jet}}$
$\mathbf{v}_1$
$\mathbf{v}_2$
...
$\mathbf{v}_{N_j}$

- The analogy is admittedly a bit strained... but it seems to work in classifiers.
- QCD inspired recursive network definition based on a sequence of particles encoded as a binary tree.



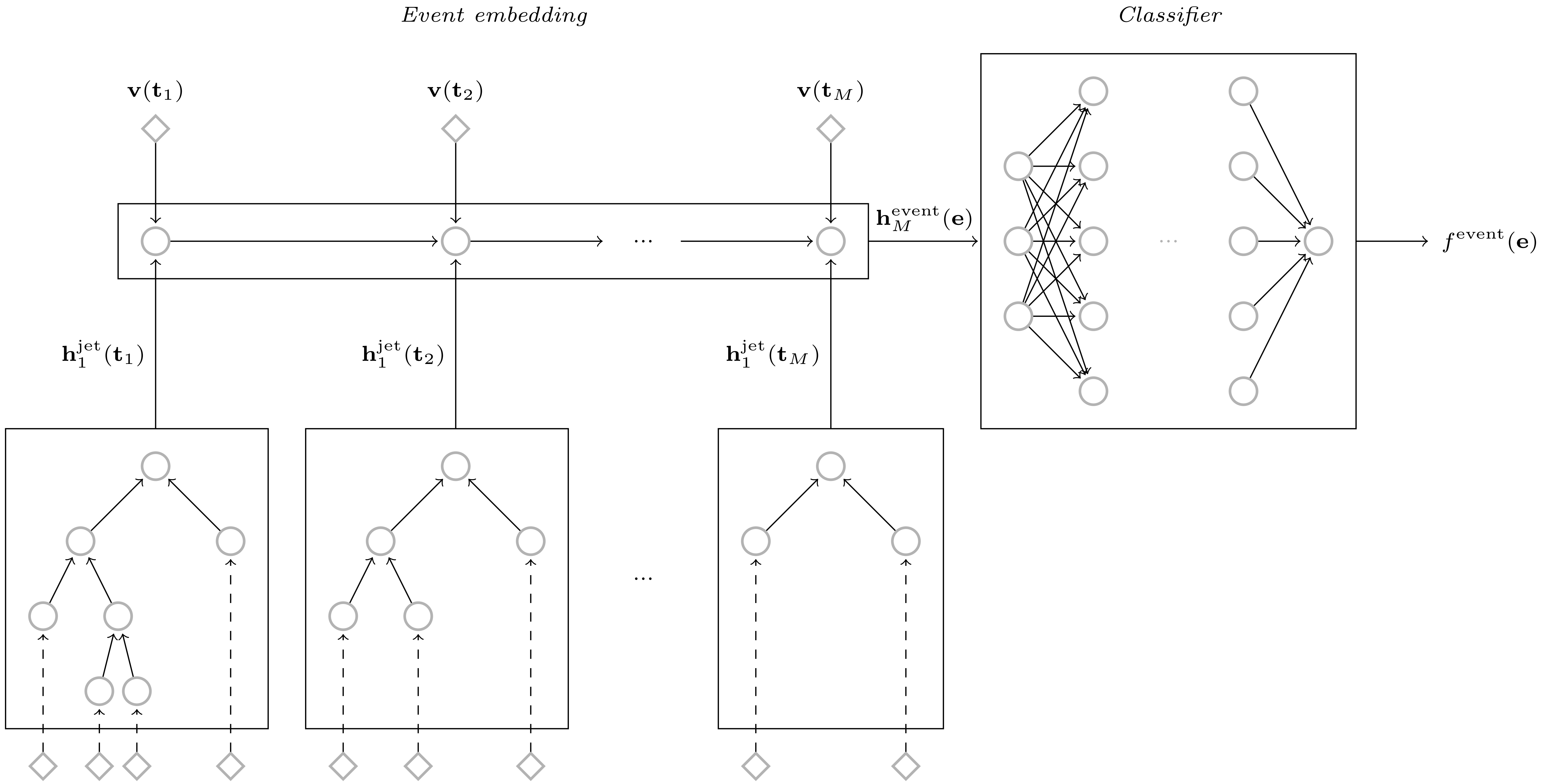
QCD-Aware Recursive Neural Networks for Jet Physics

Gilles Louppe,[1] Kyunghyun Cho,[1] Cyril Becot,[1] and Kyle Cranmer[1]

[1] *New York University*

arXiv:1702.00748v1 [hep-ph] 2 Feb 2017

Sequences of sequences



Event embedding
Classifier
$\mathbf{v}(\mathbf{t}_1)$
$\mathbf{v}(\mathbf{t}_2)$
$\mathbf{v}(\mathbf{t}_M)$
$\mathbf{h}_M^{\text{event}}(\mathbf{e})$
$f^{\text{event}}(\mathbf{e})$
$\mathbf{h}_1^{\text{jet}}(\mathbf{t}_1)$
$\mathbf{h}_1^{\text{jet}}(\mathbf{t}_2)$
$\mathbf{h}_1^{\text{jet}}(\mathbf{t}_M)$

FIG. 2. QCD-motivated event embedding for classification. The embedding of an event is computed by feeding the sequence of pairs $(\mathbf{v}(\mathbf{t}_j), \mathbf{h}_1^{\text{jet}}(\mathbf{t}_j))$ over the jets it is made of, where $\mathbf{v}(\mathbf{t}_j)$ is the unprocessed 4-momentum of the jet $\mathbf{t}_j$ and $\mathbf{h}_1^{\text{jet}}(\mathbf{t}_j)$ is its embedding. The resulting event-level embedding $\mathbf{h}_M^{\text{event}}(\mathbf{e})$ is chained to a subsequent classifier, as illustrated in the right part of the figure.

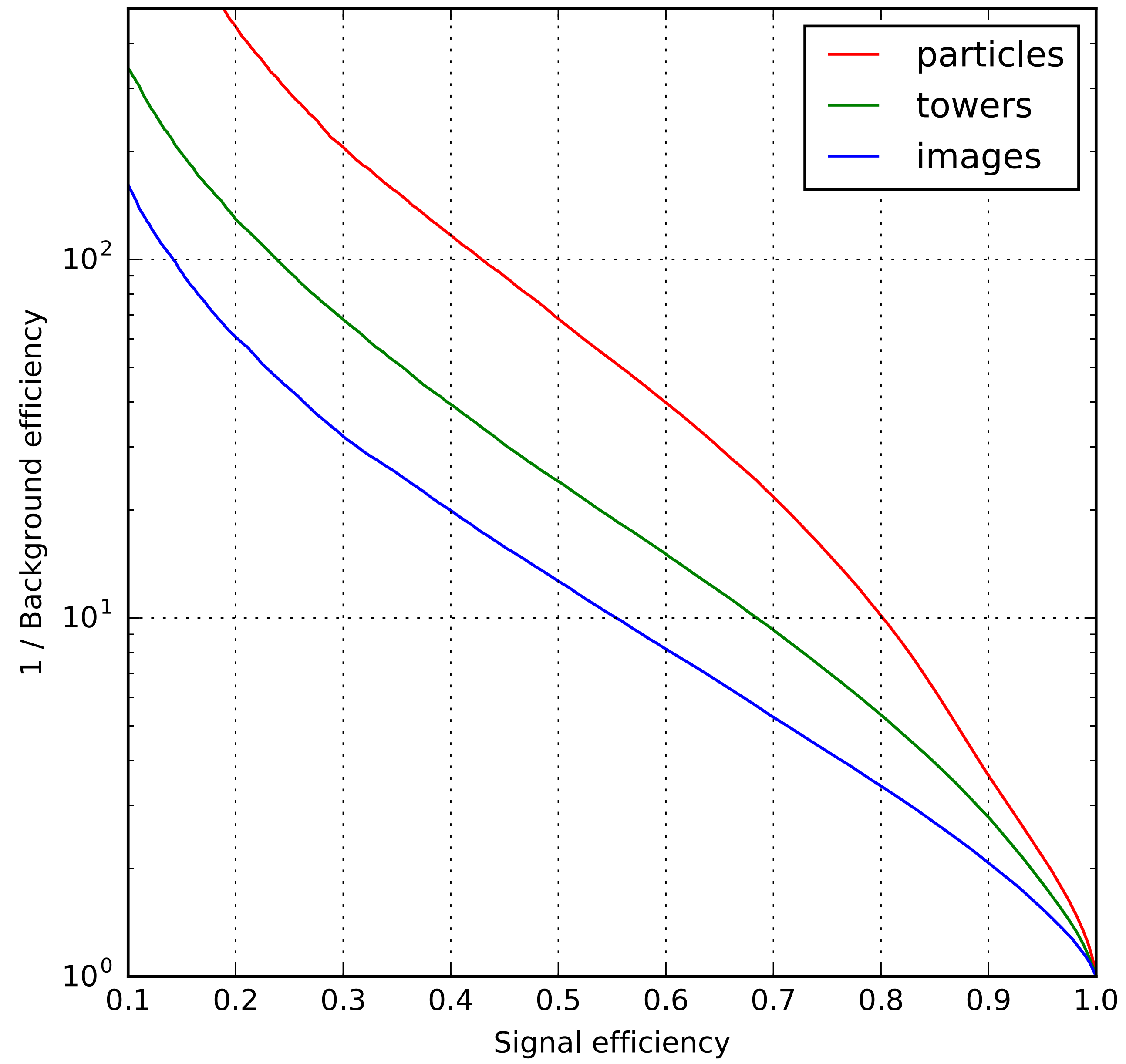


FIG. 3. Jet classification performance for various input representations of the RNN classifier, using k_t topologies for the embedding. The plot shows that there is significant improvement from removing the image processing step and that significant gains can be made with more accurate measurements of the 4-momenta.

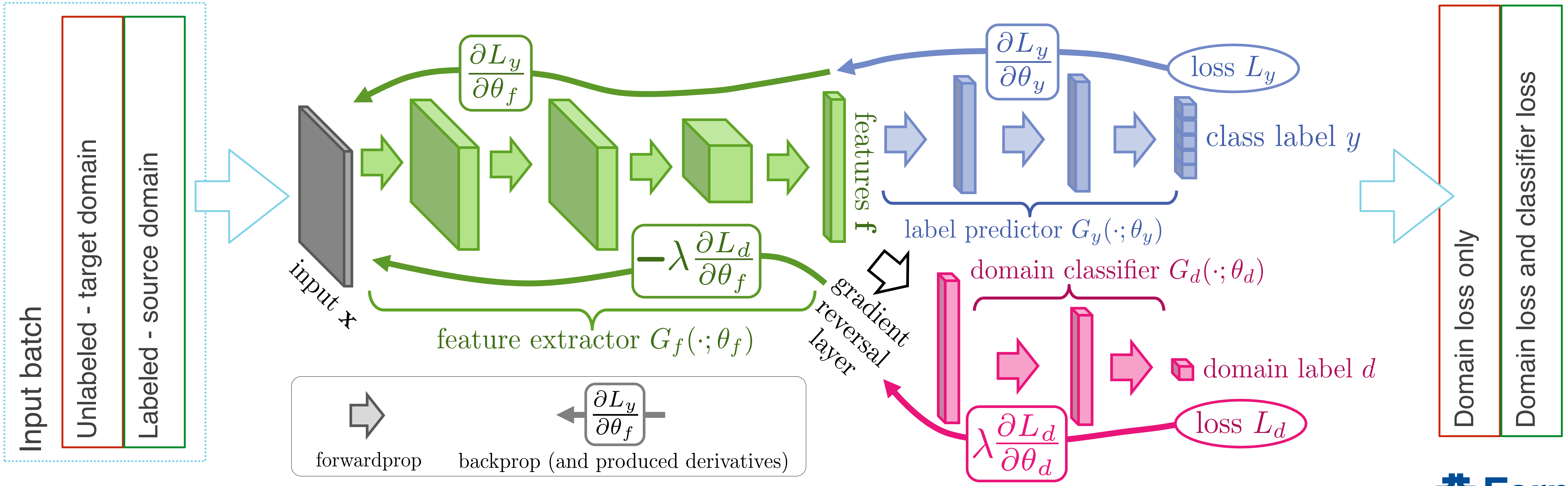
Also: https://indico.cern.ch/event/722319/ (K. Cranmer)

Can we learn from data?

- If we can build a classifier, we can build a generator.
- But generate based on... what?
- Training on the output of a MC does not really make sense - except in the cases where we are worried about simulation speed and are willing to trade off some accuracy and some information about the underlying event for speed.
  - Plausible scenario for detector simulation, e.g., EM showers in a calorimeter, but not really needed in an event generator except, perhaps, in an upstream calculation where we need to speed up some MC integration technique as per the first part of this talk.
- But, if we could train on data somehow, this approach becomes much more exciting:
  - "Capture" real distributions of particles and correlations.
  - Limited insight into the underlying dynamics, but if we can peg to total energy, maybe this is okay.

Domain adaption

- Consider a network innovation called a **Domain Adversarial Neural Network (DANN)** to minimize bias coming from a MC.
  - Journal of Machine Learning Research 17 (2016) 1-35
  - https://arxiv.org/abs/1505.07818
  - Jointly **maximize domain classifier loss and minimize feature classifier loss**.
  - Are the simulation and detector data domains **too similar** for this too work? **Too different**?


Input batch
Unlabeled - target domain
Labeled - source domain
$\frac{\partial L_y}{\partial \theta_f}$
$\frac{\partial L_y}{\partial \theta_y}$
loss L_y
class label y
features f
input x
label predictor $G_y(\cdot;\theta_y)$
$-\lambda \frac{\partial L_d}{\partial \theta_f}$
feature extractor $G_f(\cdot;\theta_f)$
gradient reversal layer
domain classifier $G_d(\cdot;\theta_d)$
domain label d
$\lambda \frac{\partial L_d}{\partial \theta_d}$
loss L_d
forwardprop
backprop (and produced derivatives)
Domain loss only
Domain loss and classifier loss

Sketch of a generative RNN as event generator...

- Fix the total cross section with a well-grounded theoretical calculation.
- Train on a mixture of data and simulation - use the simulation to teach the network how to produce particle signatures of a given energy in the detector, and use data to learn the distribution of produced particles (along with all correlations). Regulate data / MC exposure into the loss function for training using a domain adversarial network.
- Condition the generative model for the output sequence on neutrino energy (one to many mapping) - could also use target nucleus.
- *Not at all* obvious that this would work...
  - Guaranteed to generate some stream of particles... but would they be "gibberish"?
  - Even if the output stream were not gibberish, would using a domain adversarial component to condition the loss actually produce a particle stream that mimicked the distributions found in data?
    - Every reason to be skeptical... but it might be fun to try! The result would be a data-driven final state particle generator you could use to build neutrino energy reconstruction algorithms.

Conclusions

- ML methods show significant promise for speeding up certain calculations.
- Generally, this sort of method is probably not important for generators like the ones considered in this workshop - but these techniques could matter for upstream calculations that form inputs to "event-level" MC generators.
- There is potential for using generative models directly as "event-level" MC generators, but a great deal of work remains to be done:
  - Consider semantics of a particle sequence. ML research focuses on language problems, and we have different dynamics, symmetries, concerns, etc.
  - Research popping up that encodes symmetries in the network structure - maybe we could exploit that here.
  - Need to connect particle description to a detector description. The pieces are all there, but this hasn't actually been done yet (that I know of, but maybe in some corner of LHC-land).
  - Need to explore techniques for conditioning distributions on data so we may connect true, underlying information to an set of observables we don't know how to otherwise correctly simulate. Challenging to do correctly, but we have some unlikely allies interested in this problem - they're working on something just as crazy, namely ... self-driving cars!

Thanks!

now...

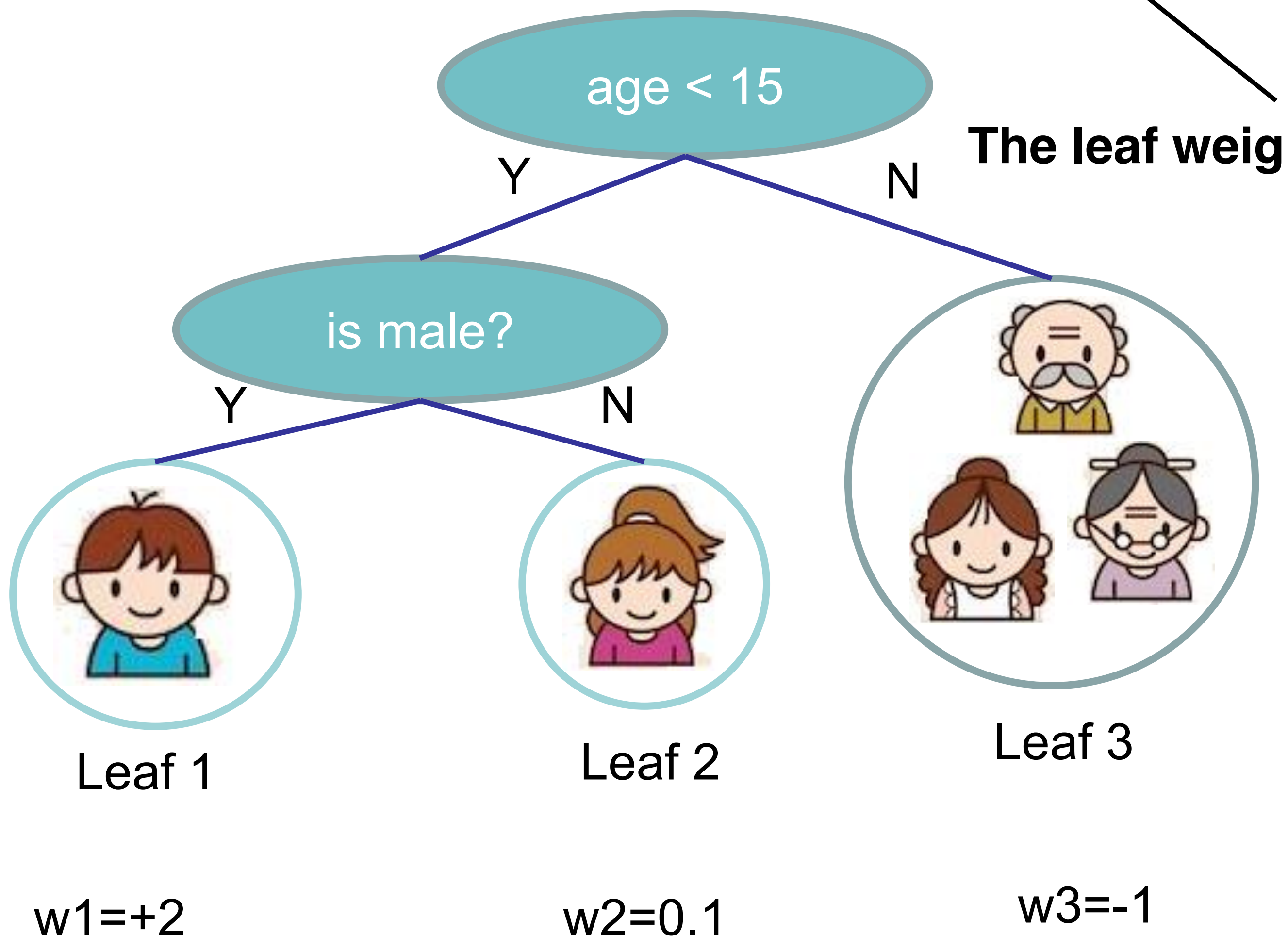
Back up!

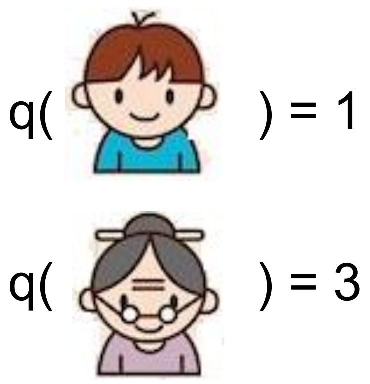
- We define tree by a vector of scores in leafs, and a leaf index mapping function that maps an instance to a leaf

$$f_t(x) = w_{q(x)}, \quad w \in \mathbf{R}^T, q : \mathbf{R}^d \to \{1, 2, \cdots, T\}$$

The structure of the tree

The leaf weight of the tree

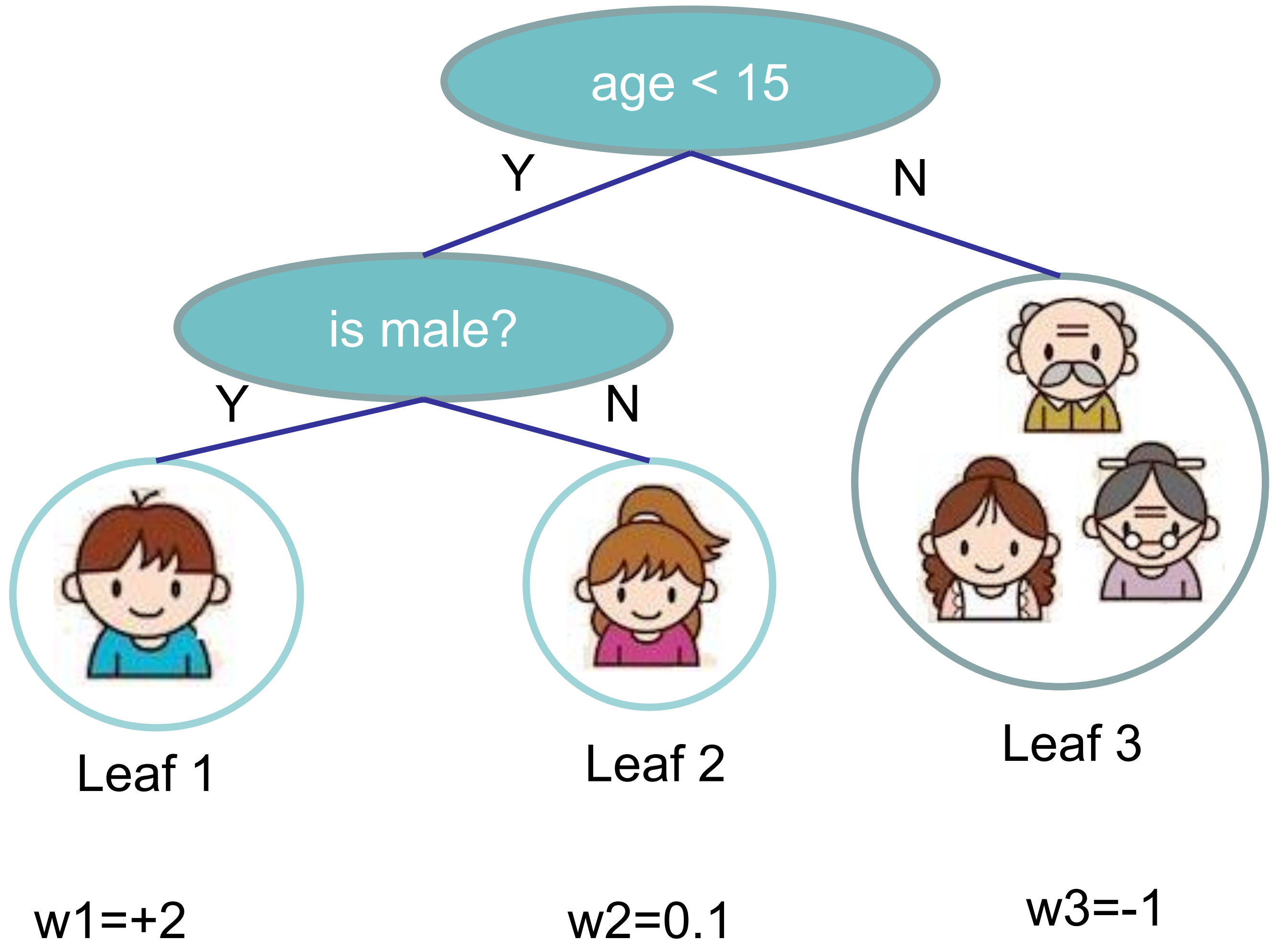




- Define complexity as (this is not the only possible definition)

$$\Omega(f_t) = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2$$

Number of leaves **L2 norm of leaf scores**



$$\Omega = \gamma 3 + \frac{1}{2}\lambda(4 + 0.01 + 1)$$

Tianqi Chen
University of Washington
tqchen@cs.washington.edu

- Define the instance set in leaf j as $I_j = \{i | q(x_i) = j\}$
- Regroup the objective by each leaf

$$
\begin{aligned}
Obj^{(t)} &\simeq \sum_{i=1}^{n} \left[g_i f_t(x_i) + \tfrac{1}{2} h_i f_t^2(x_i) \right] + \Omega(f_t) \\
&= \sum_{i=1}^{n} \left[g_i w_{q(x_i)} + \tfrac{1}{2} h_i w_{q(x_i)}^2 \right] + \gamma T + \lambda \tfrac{1}{2} \sum_{j=1}^{T} w_j^2 \\
&= \sum_{j=1}^{T} \left[(\textstyle\sum_{i \in I_j} g_i) w_j + \tfrac{1}{2} (\textstyle\sum_{i \in I_j} h_i + \lambda) w_j^2 \right] + \gamma T
\end{aligned}
$$

- This is sum of T independent quadratic functions

- Two facts about single variable quadratic function

$$
argmin_x \ Gx + \tfrac{1}{2} Hx^2 = -\tfrac{G}{H}, \ H > 0 \qquad \min_x \ Gx + \tfrac{1}{2} Hx^2 = -\tfrac{1}{2} \tfrac{G^2}{H}
$$

- Let us define $G_j = \sum_{i \in I_j} g_i \quad H_j = \sum_{i \in I_j} h_i$

$$
\begin{aligned}
Obj^{(t)} &= \sum_{j=1}^{T} \left[(\textstyle\sum_{i \in I_j} g_i) w_j + \tfrac{1}{2} (\textstyle\sum_{i \in I_j} h_i + \lambda) w_j^2 \right] + \gamma T \\
&= \sum_{j=1}^{T} \left[G_j w_j + \tfrac{1}{2} (H_j + \lambda) w_j^2 \right] + \gamma T
\end{aligned}
$$

- Assume the structure of tree (q(x)) is fixed, the optimal weight in each leaf, and the resulting objective value are

$$
w_j^* = -\frac{G_j}{H_j + \lambda} \qquad Obj = -\frac{1}{2} \sum_{j=1}^{T} \frac{G_j^2}{H_j + \lambda} + \gamma T
$$

This measures how good a tree structure is!

- Enumerate the possible tree structures q
- Calculate the structure score for the q, using the scoring eq.

$$
Obj = -\frac{1}{2} \sum_{j=1}^{T} \frac{G_j^2}{H_j + \lambda} + \gamma T
$$

- Find the best tree structure, and use the optimal leaf weight

$$
w_j^* = -\frac{G_j}{H_j + \lambda}
$$

- But… there can be infinite possible tree structures..

- In practice, we grow the tree greedily
  - Start from tree with depth 0
  - For each leaf node of the tree, try to add a split. The change of objective after adding the split is

The complexity cost by introducing additional leaf

$$
Gain = \frac{1}{2} \left[\frac{G_L^2}{H_L + \lambda} + \frac{G_R^2}{H_R + \lambda} - \frac{(G_L + G_R)^2}{H_L + H_R + \lambda} \right] - \gamma
$$

the score of left child

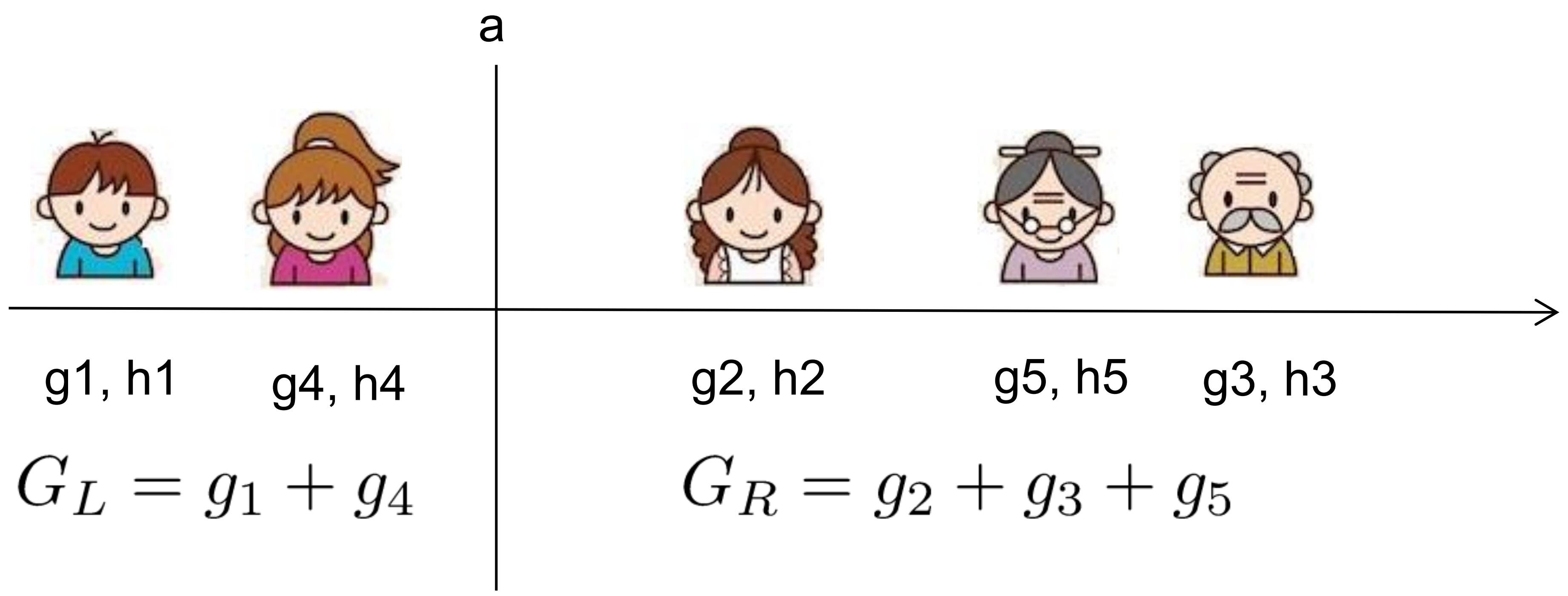
the score of if we do not split

the score of right child

  - Remaining question: how do we find the best split?

Tianqi Chen
University of Washington
tqchen@cs.washington.edu

- What is the gain of a split rule $x_j < a$? Say x_j is age



$$G_L = g_1 + g_4 \qquad G_R = g_2 + g_3 + g_5$$

- All we need is sum of g and h in each side, and calculate

$$Gain = \frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{(G_L+G_R)^2}{H_L+H_R+\lambda} - \gamma$$

- Left to right linear scan over sorted instance is enough to decide the best split along the feature

- For each node, enumerate over all features
  - For each feature, sorted the instances by feature value
  - Use a linear scan to decide the best split along that feature
  - Take the best split solution along all the features

- Time Complexity growing a tree of depth K
  - It is O(n d K log n): or each level, need O(n log n) time to sort There are d features, and we need to do it for K level
  - This can be further optimized (e.g. use approximation or caching the sorted features)
  - Can scale to very large dataset

Tianqi Chen
University of Washington
tqchen@cs.washington.edu