Thinking Compositionally about Inference

Jan-Willem van de Meent

University of Amsterdam

Heiko Zimmermann

Babak Esmaeili

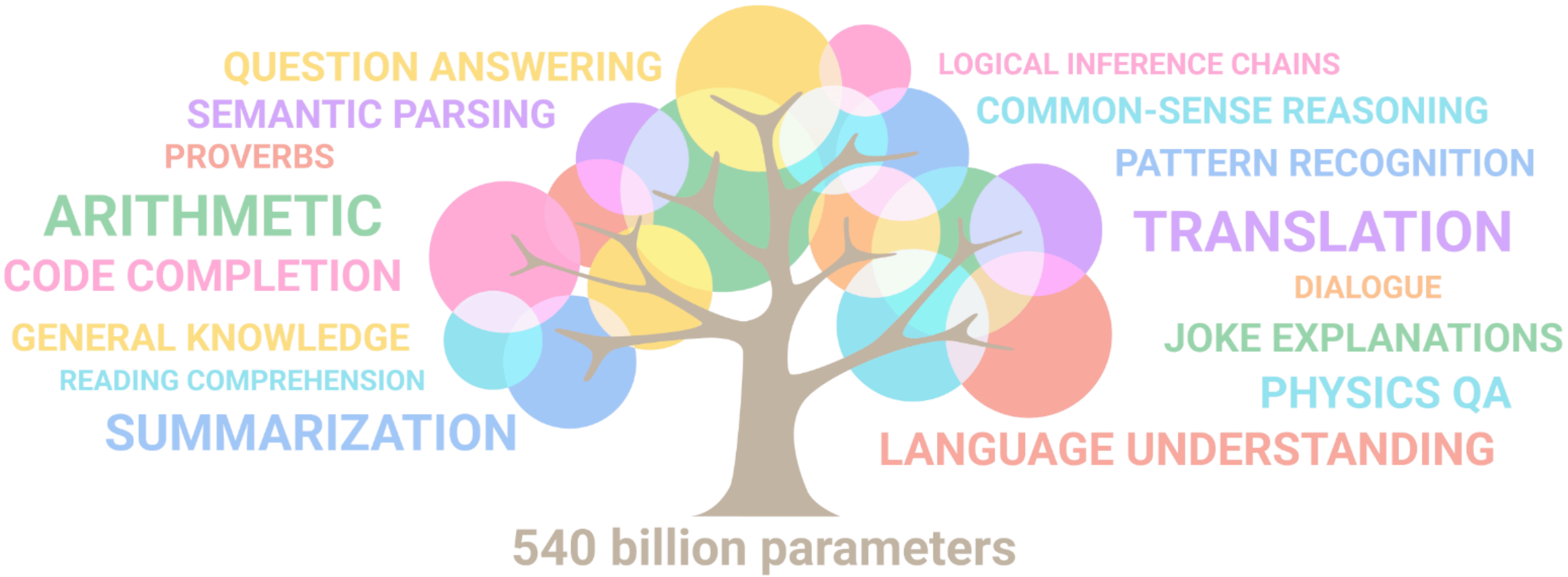
Northeastern University

Sam Stites

Hao Wu

Eli Sennesh

Deep Learning Success Stories



PaLM

Deep Learning Success Stories

Explaining a joke

Prompt

```
Explain this joke:

Joke: Did you see that Google just hired an eloquent whale for
their TPU team? It showed them how to communicate between two
different pods!
```

Model Response

```
Prediction: TPUs are a type of computer chip that Google uses
for deep learning. A "pod" is a group of TPUs. A "pod" is also a
group of whales. The joke is that the whale is able to
communicate between two groups of whales, but the speaker is
pretending that the whale is able to communicate between two
groups of TPUs.
```

PaLM

AMLAB Amsterdam Machine Learning Lab

Deep Learning Success Stories

Painting of the **orange cat Otto von Garfield**, Count of Bismarck-Schönhausen, Duke of Lauenburg, Minister-President of Prussia. Depicted **wearing a Prussian Pickelhaube** and **eating his favorite meal - lasagna**.

A photo of the **back of a wombat** wearing a **backpack** and holding a **walking stick**. It is next to a **waterfall** and is staring at a **distant mountain**.

Dall-E 2, Imagen, Parti, Stable Diffusion

Amsterdam Machine Learning Lab

Is Scale All We Need?

The Bitter Lesson

Rich Sutton, March 13, 2019

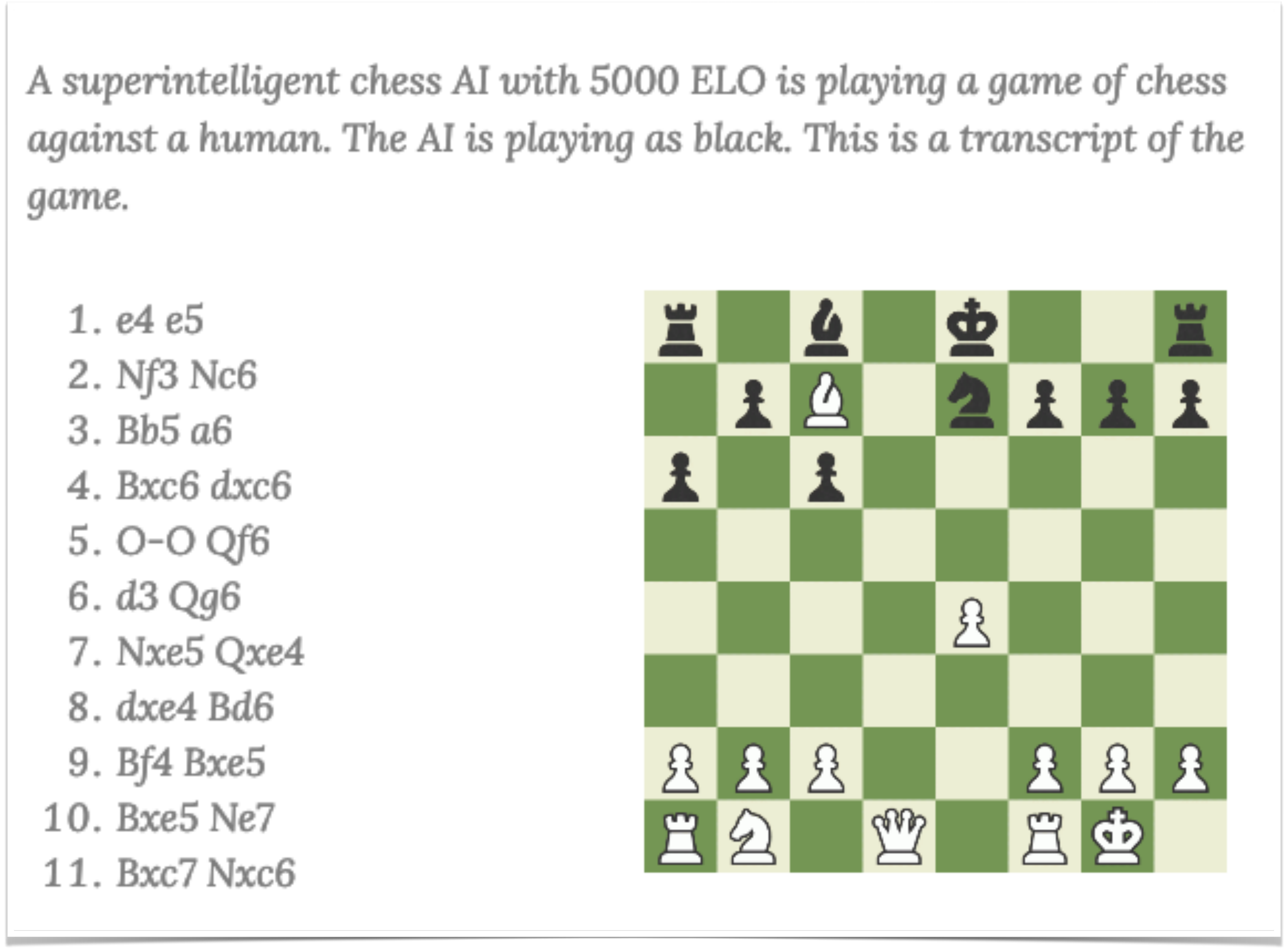
The biggest lesson that can be read from 70 years of AI research is that general methods that leverage computation are ultimately the most effective, and by a large margin.

http://www.incompleteideas.net/IncIdeas/BitterLesson.html

- Scale can lead to abstractions and generalization across tasks
- Still difficult to know when a model will succeed or fail.
- How can we scale up to more diverse application domains?

AMLAB Amsterdam Machine Learning Lab

Is Scale All We Need?

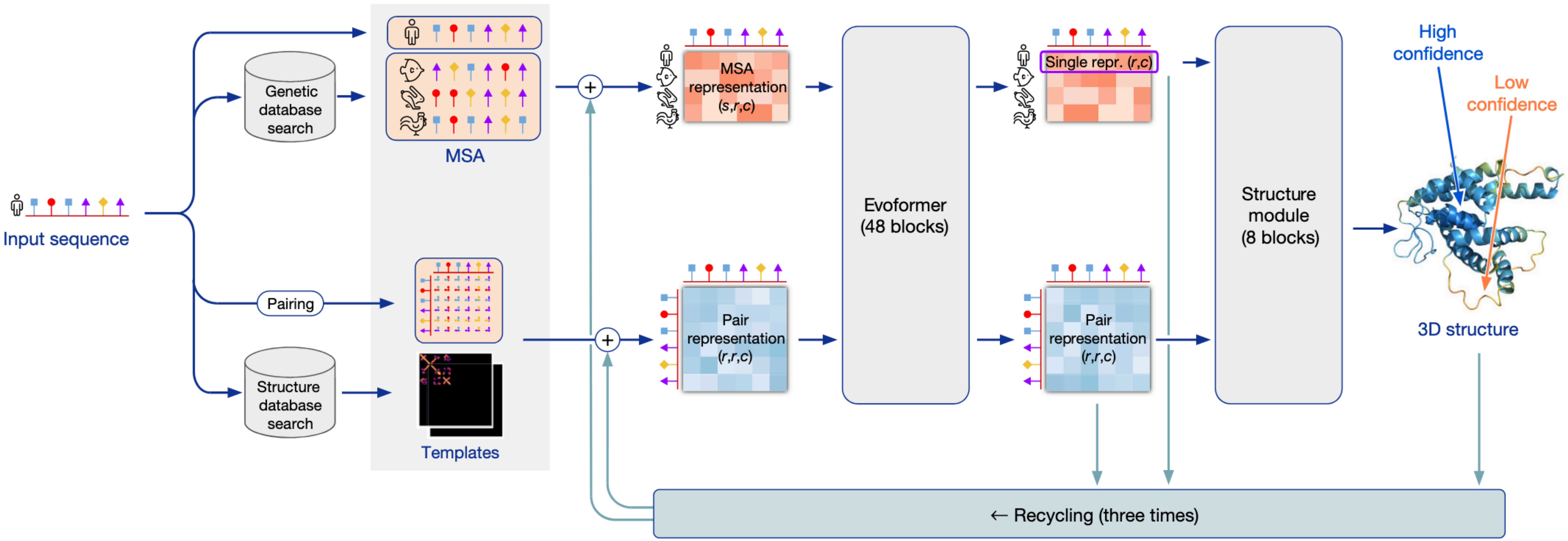


https://jacobbuckman.com/2022-06-14-an-actually-good-argument-against-naive-ai-scaling/

- Scale can lead to abstractions and generalization across tasks
- Still difficult to know when a model will succeed or fail.
- How can we scale up to more diverse application domains?

AMLAB Amsterdam Machine Learning Lab

Adapting Deep Learning to New Domains



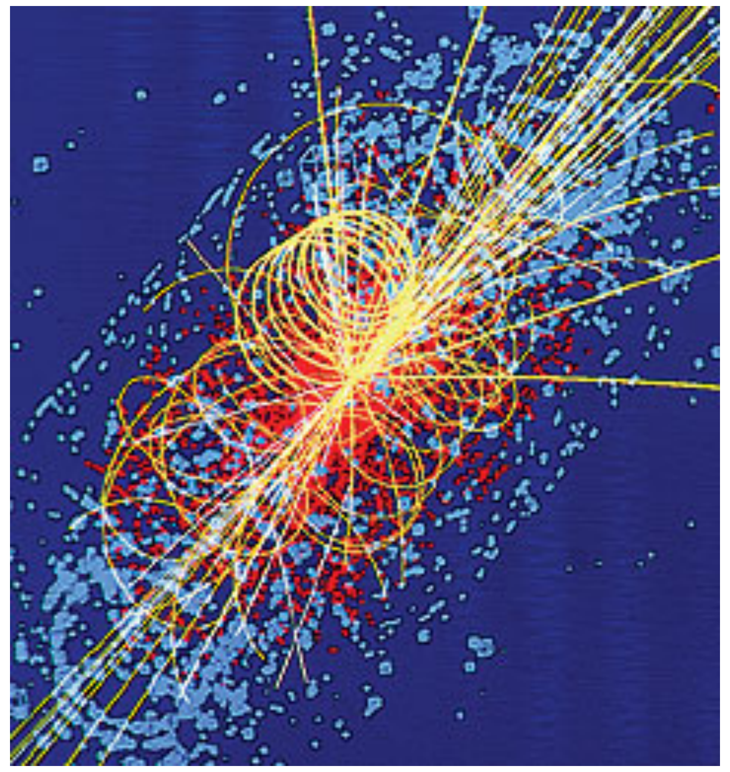
Lessons from deep learning

1. Gradient descent scales *really* well
2. Model engineering scales *pretty* well

AMLAB Amsterdam Machine Learning Lab

Horizons of AI Research

Science & Engineering

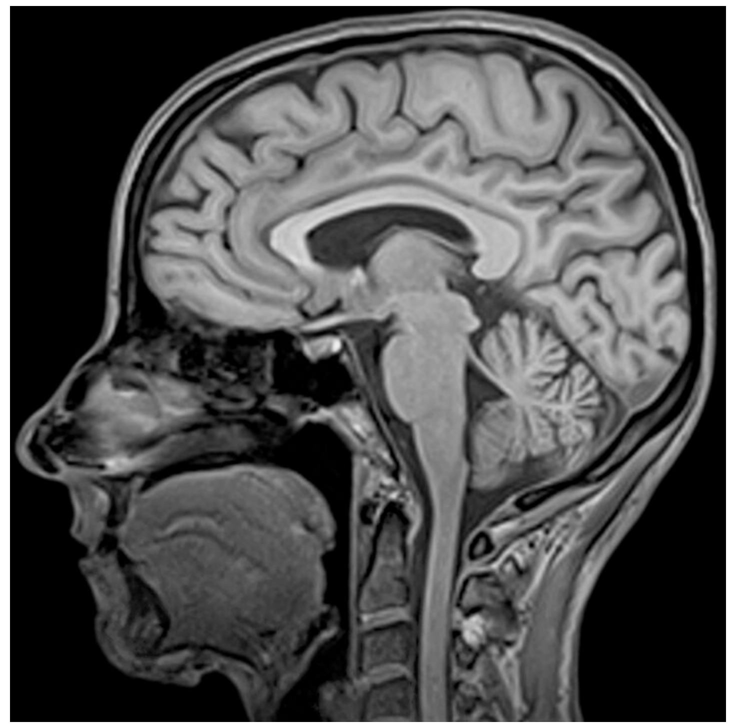


Deep domain knowledge
but limited data

Autonomous Vehicles

Generalization to
long tail events

Healthcare



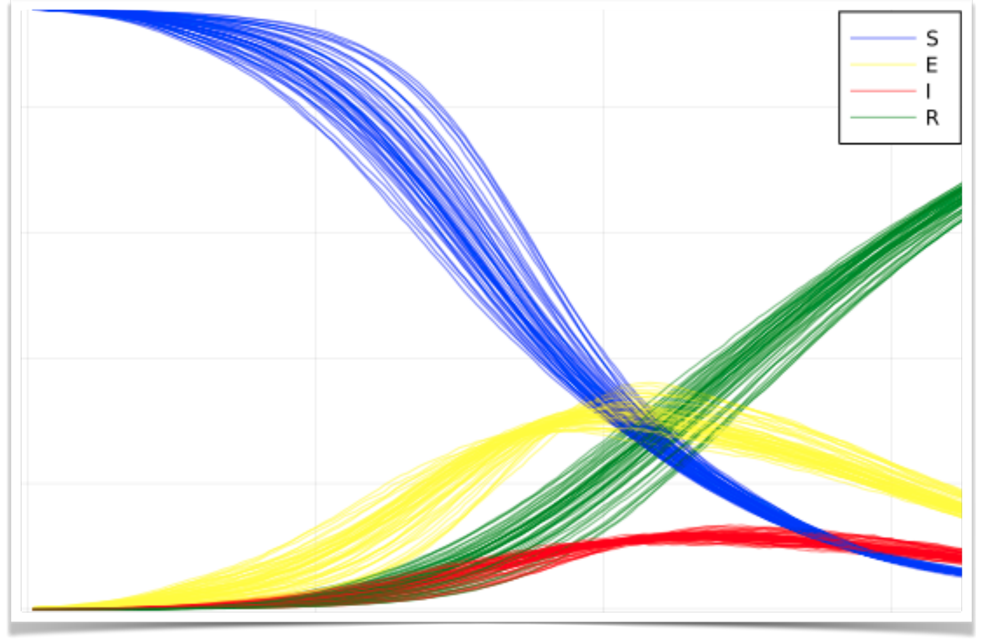
Many prediction tasks,
imbalanced data

Challenges in emerging domains

1. Incorporating (enough) domain knowledge
2. Reliable generalization across related tasks
3. Avoiding overconfident predictions

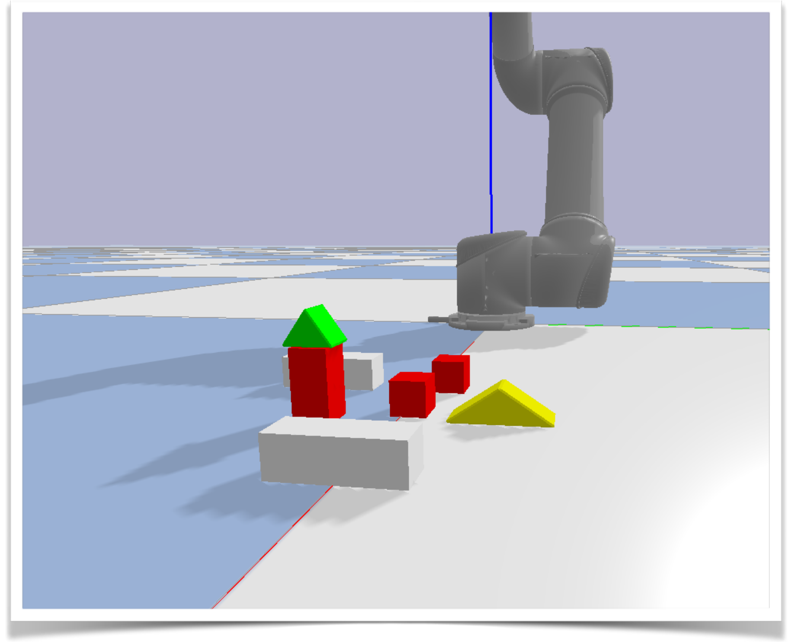
What Models are Useful?

Simulation-based Modeling



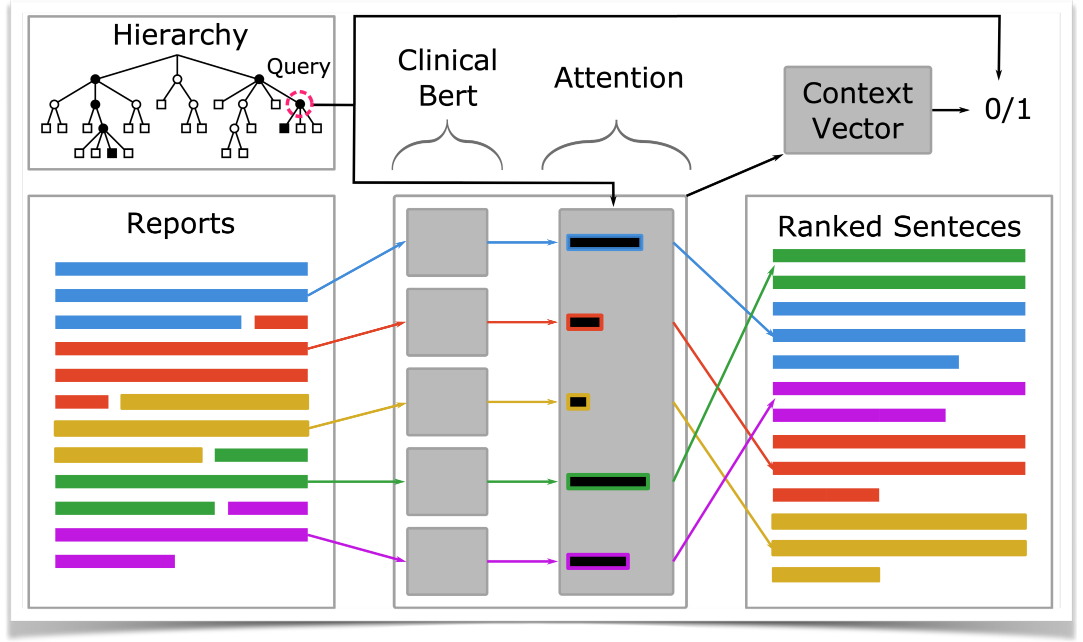
[Smedemark-Margulies et al., 2021]

Planning and Robotics



[Biza et al., 2021]

Vision & Language



[McInerney et al., 2020]

Stronger assumptions ⟵ ⟶ *Weaker assumptions*

Known dynamics (e.g. PDEs) for system

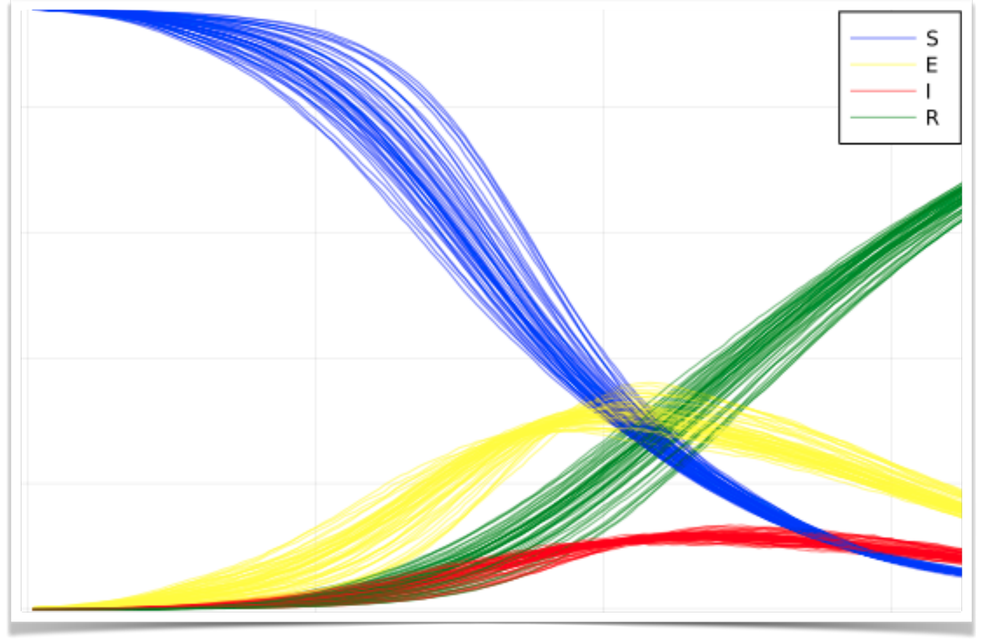
More knowledge (and edge cases)

Some domain knowledge (e.g. structure)

AMLAB Amsterdam Machine Learning Lab

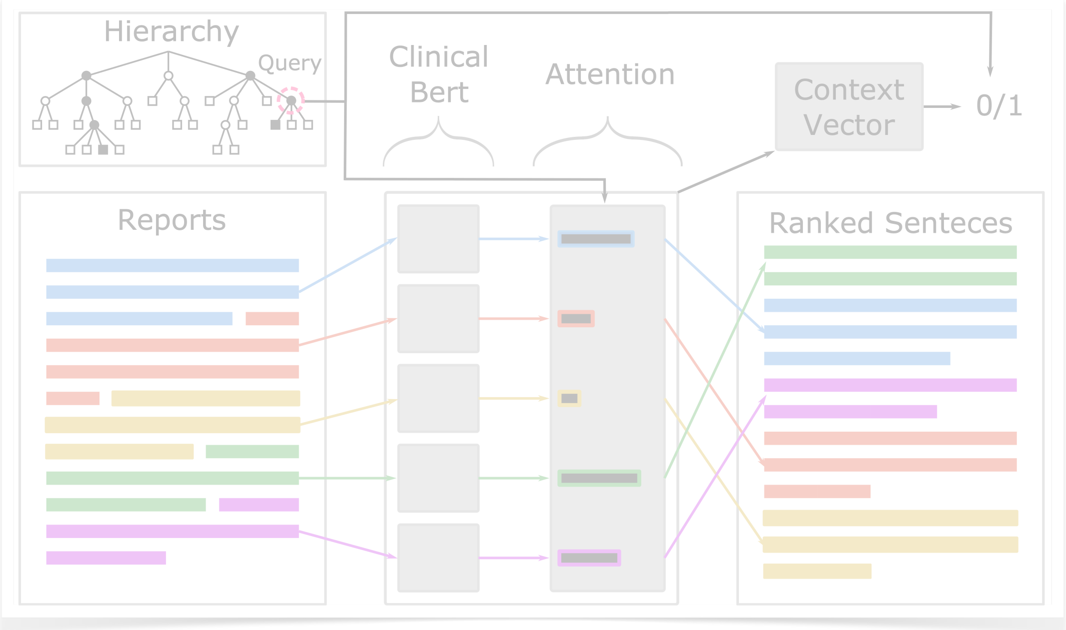
What Models are Useful?

Simulation-based Modeling	Planning and Robotics	Vision & Language



[Smedemark-Margulies et al., 2021]

[Biza et al., 2021]



[McInerney et al., 2020]

Stronger assumptions ⟵ ⟶ *Weaker assumptions*

Known dynamics (e.g. PDEs) for system	More knowledge (and edge cases)	Some domain knowledge (e.g. structure)

Amsterdam Machine Learning Lab

The Next 700
AI Domains

Amsterdam
Machine Learning Lab

The Next 700 Programming Languages

P. J. Landin

Univac Division of Sperry Rand Corp., New York, New York

"... today ... 1,700 special programming languages used to 'communicate' in over 700 application areas."—*Computer Software* Issues, an American Mathematical Association Prospectus, July 1965.

Volume 9 / Number 3 / March, 1966 Communications of the ACM 157

Two Ingredients for a Language

1. Core operations / abstractions
2. Mechanisms for composition into program

Differentiable Programming

TensorFlow PyTorch

1. **Abstractions:** Differentiation, Tensor Calculus, Layers
2. **Composition:** Networks, Objectives, Optimization

Probabilistic Programming

Stan
https://mc-stan.org

Anglican
https://probprog.github.io/anglican/

Gen.jl
https://www.gen.dev/

Pyro
https://pyro.ai/

Birch
https://www.birch.sh/

1. **Abstractions:** Distributions, Conditioning, Inference
2. **Composition:** Programs as Probabilistic Models

AMLAB
Amsterdam
Machine Learning Lab

Programs as Models

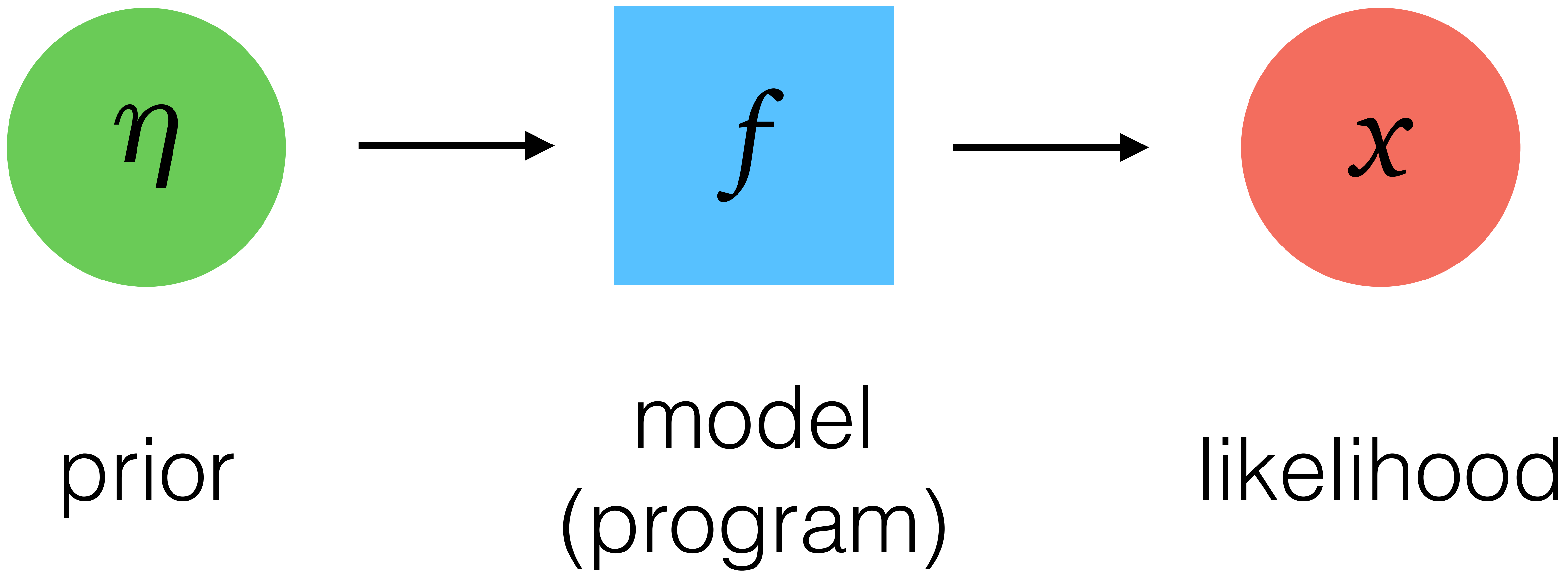
Forward Simulation

$$\eta \longrightarrow \boxed{f} \longrightarrow \hat{x}$$

inputs (parameters)	model (program)	outputs (predictions)

Programs as Models

Forward Simulation

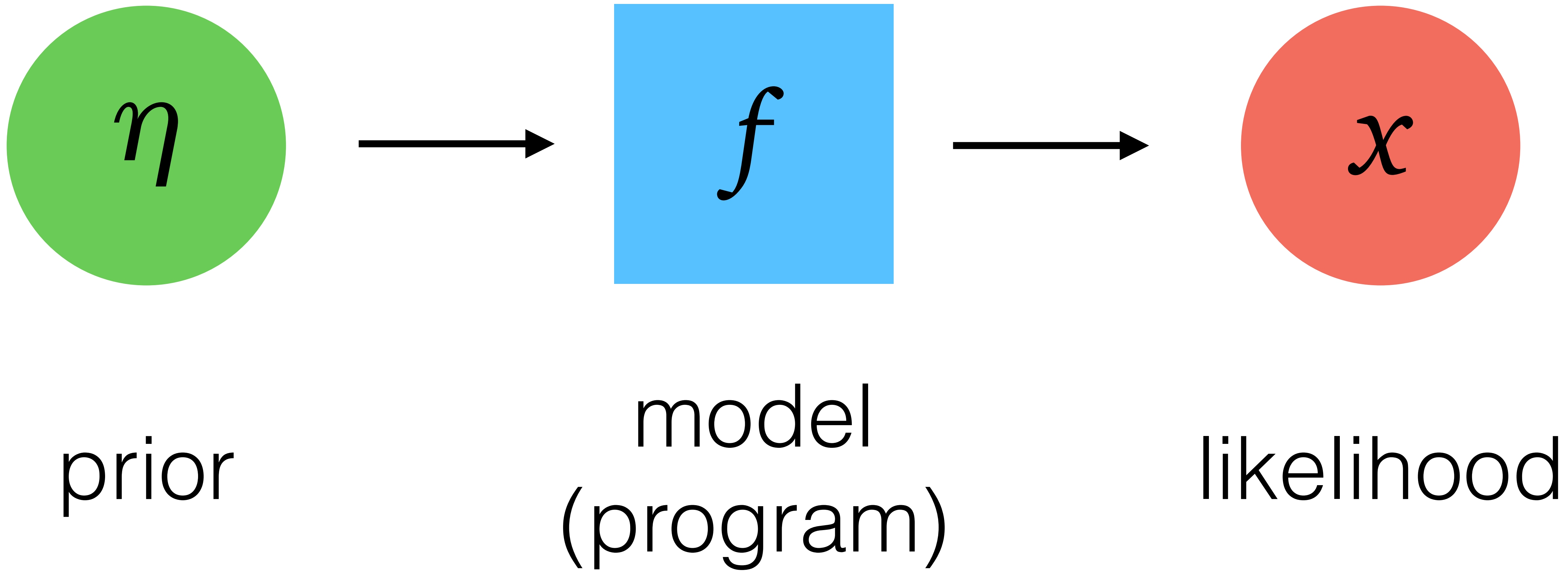


Generative Model

$$p(x, \eta) = p(\eta)\, p(x \mid f(\eta))$$

Programs as Models

Forward Simulation

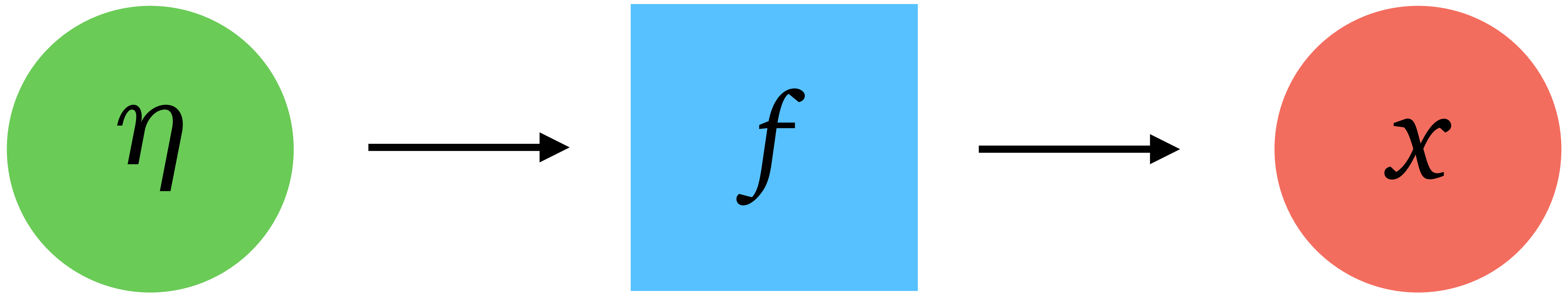


Inference (Inverse Reasoning)

$$p(\eta \mid x) = \frac{p(\eta)\, p(x \mid f(\eta))}{p(x)}$$

Programs as Models

Forward Simulation



prior

model (program)

likelihood

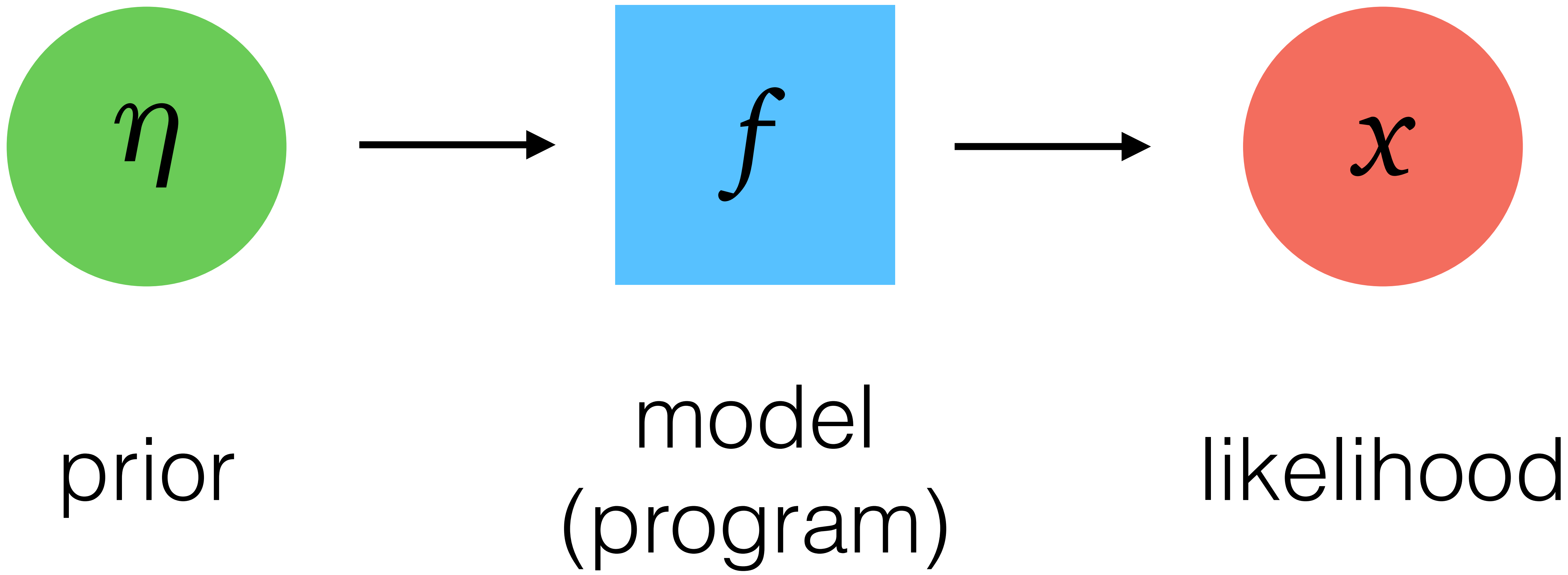
Inference (Inverse Reasoning)

$$p(\eta \mid x) = \frac{p(\eta)\, p(x \mid f(\eta))}{\boxed{p(x)}}$$

Intractable integral

Programs as Models

Forward Simulation



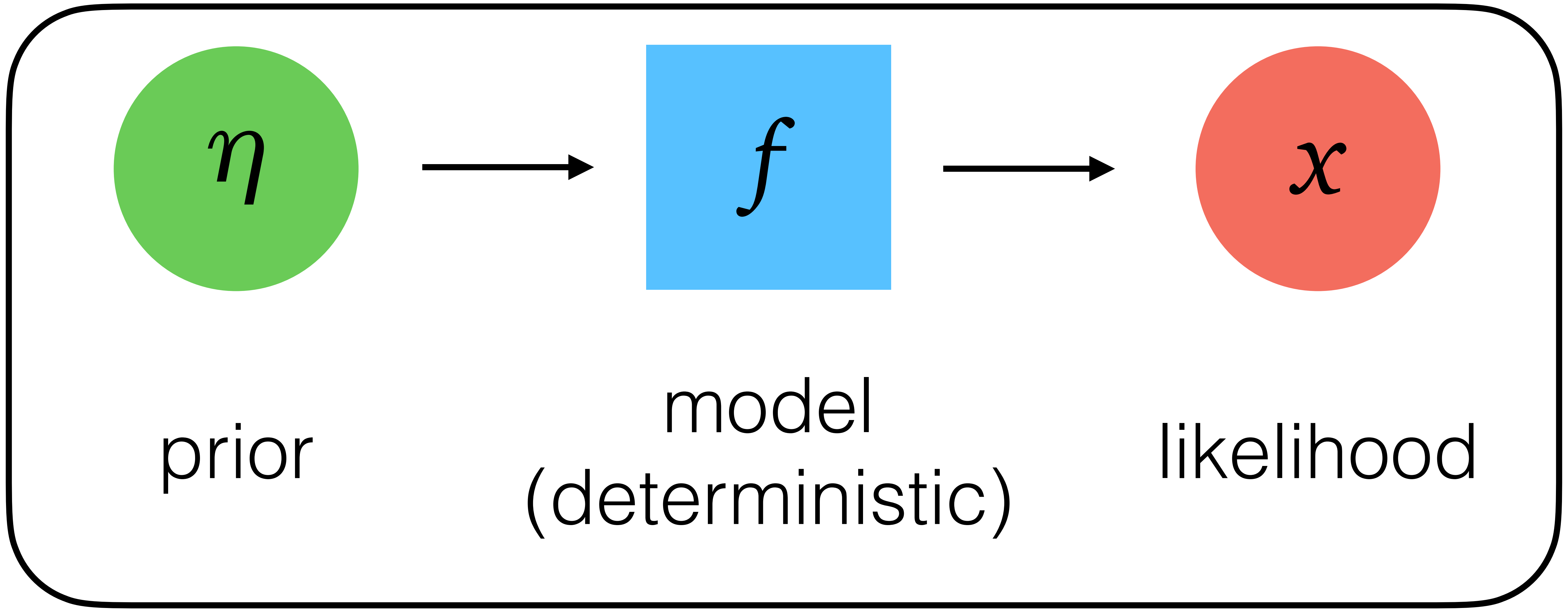
Inference (Inverse Reasoning)

$$p(\eta \mid x) = \frac{p(\eta)\, p(x \mid f(\eta))}{p(x)}$$

Can have intractable likelihood (ABC/SBI)

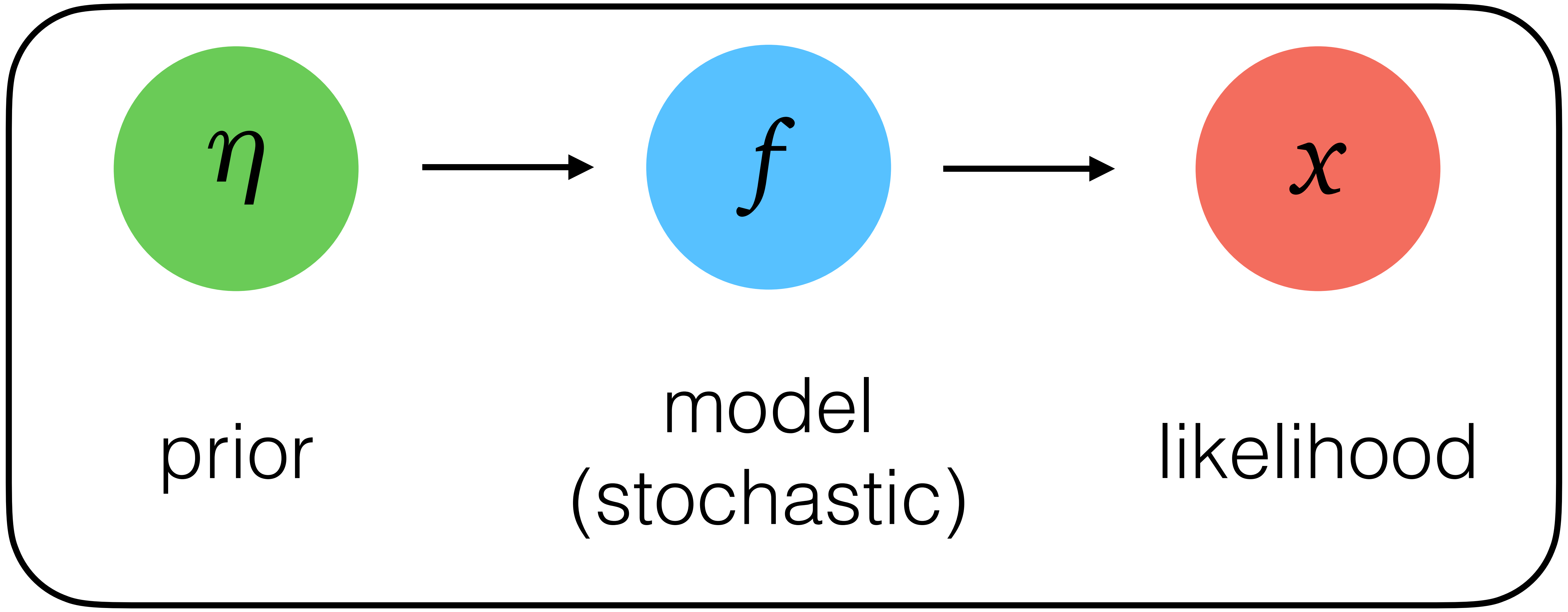
Programs as Models

Probabilistic Program (User Defined)



Programs as Models

Probabilistic Program (User Defined)



Inference (Implemented by System)

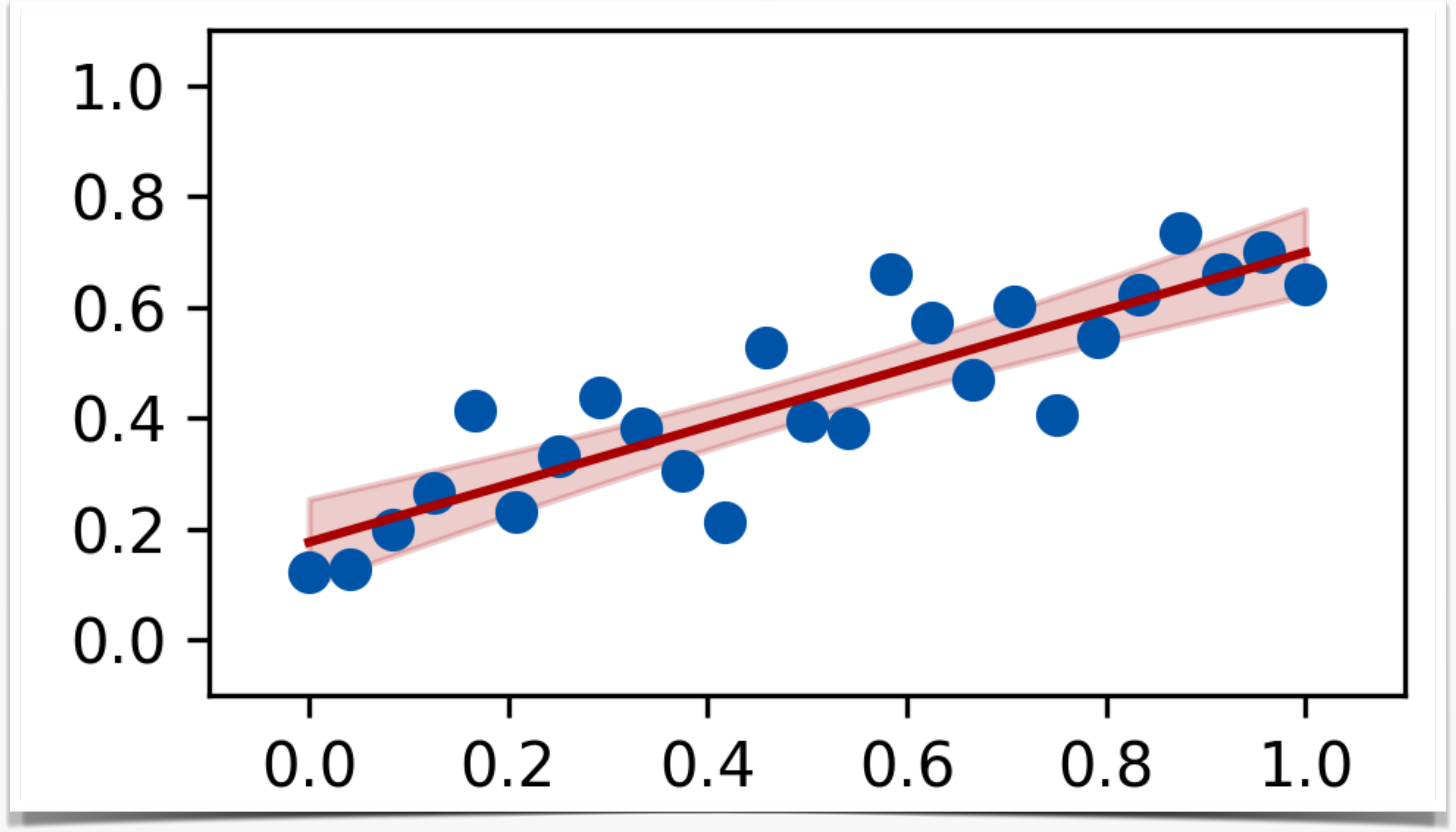
$$\eta \sim p(\eta \mid x)$$

Programs as Probabilistic Models

Probabilistic Program

```
def regression(y_vals, x_vals):
  a = sample(normal(0, 10))
  b = sample(normal(0, 10))
  f = lambda x: a * x + b
  for y, x in zip(y_vals, x_vals):
    observe(normal(f(x), 0.1), y)
  return f, a, b
```

Inference (MCMC)



Probabilistic Special Forms

sample *random* value

observe *condition* on a value

$$p(a, b \mid y) = \frac{p(y, a, b)}{p(y)}$$

*Infer **sample** values that are in agreement with **observe** values (using Bayesian statistics)*

Programs as Probabilistic Models (~2013)

Probabilistic Program

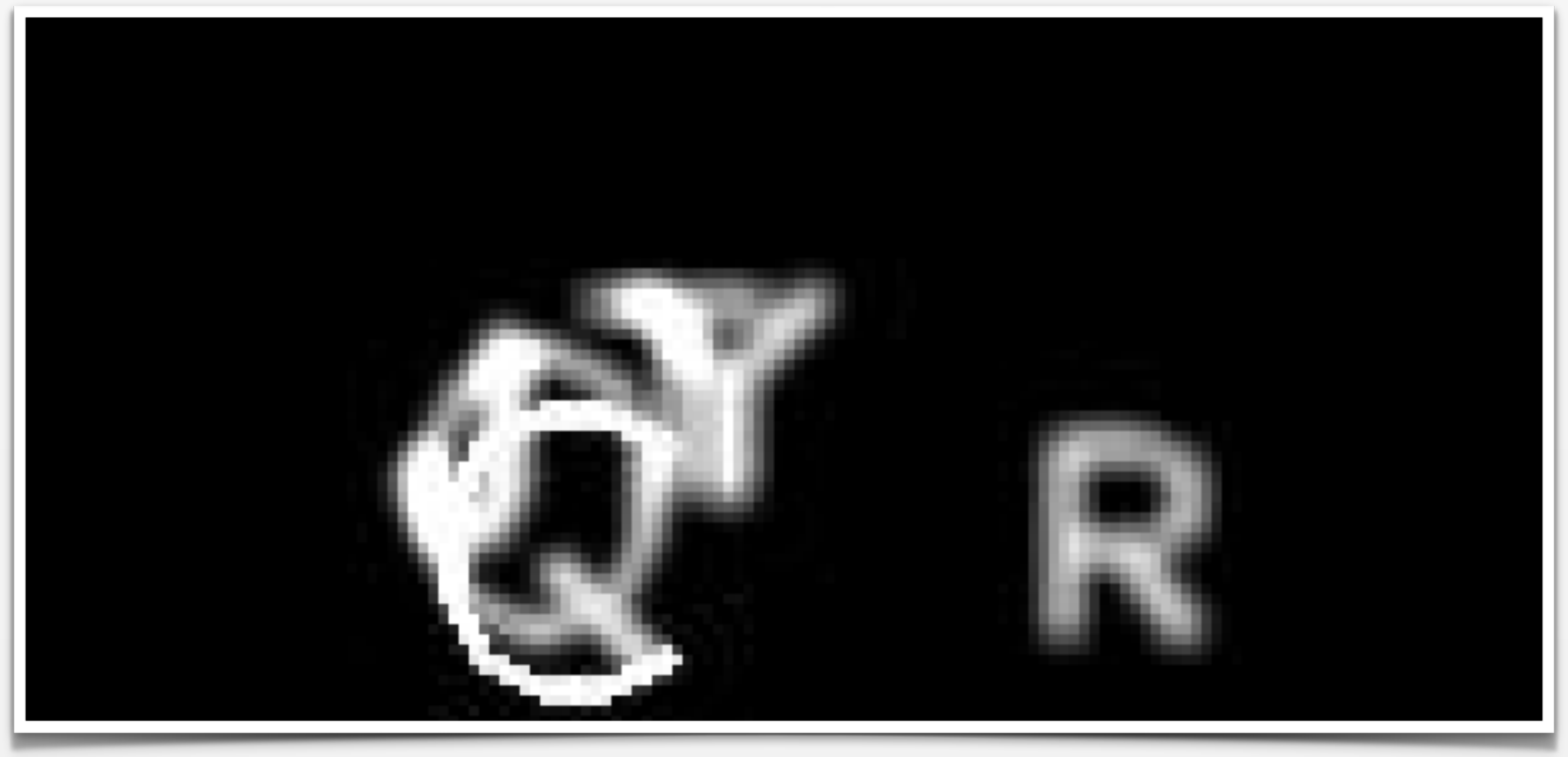
```
def render(chars, shape):
 ...

def char():
 x = sample(uniform(0.0, 1.0))
 y = sample(uniform(0.0, 1.0))
 size = sample(uniform(0.5, 1.5))
 weight = sample(uniform(0.8, 1.2))
 return x, y, size, weight

def captcha(image):
 K = sample(uniform(range(1, 6)))
 chars = [char() for k in range(K)]
 captcha = render(chars, image.shape)
 observe(normal(captcha, 0.1), image)
 return captcha, chars
```

Data

Inference (MCMC)



[Mansinghka, Kulkarni, Perov, Tenenbaum, NeurIPS 2013]

Programs as Probabilistic Models (~2013)

Probabilistic Program

```
def render(chars, shape):
 ...

def char():
 x = sample(uniform(0.0, 1.0))
 y = sample(uniform(0.0, 1.0))
 size = sample(uniform(0.5, 1.5))
 weight = sample(uniform(0.8, 1.2))
 return x, y, size, weight

def captcha(image):
 K = sample(uniform(range(1, 6)))
 chars = [char() for k in range(K)]
 captcha = render(chars, image.shape)
 observe(normal(captcha, 0.1), image)
 return captcha, chars
```

Data

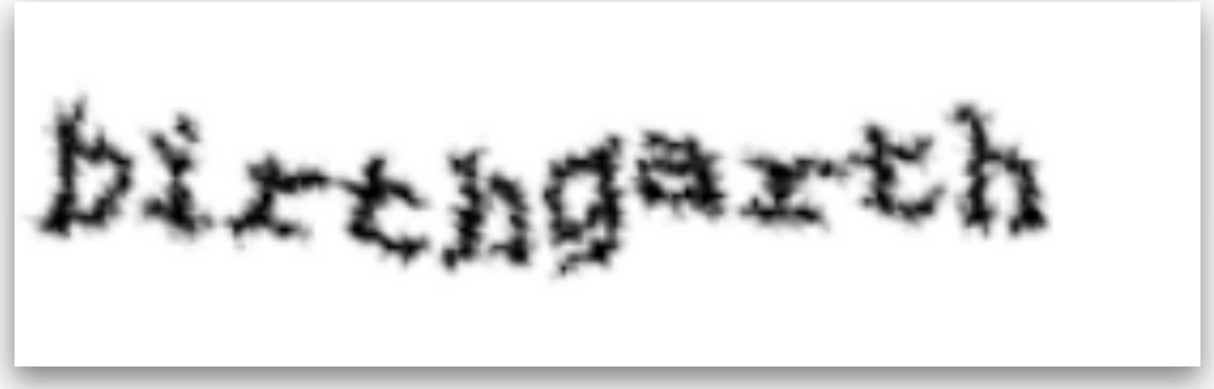
Inference (MCMC)

[Mansinghka, Kulkarni, Perov, Tenenbaum, NeurIPS 2013]

Programs as Probabilistic Models (~2013)

Probabilistic Program

Actual CAPTCHAs



- Writing realistic simulators can be difficult
  (might need source code for CAPTCHA generator)
- Inference can be prohibitively slow
  (MCMC takes ~15 mins for a single image)

Deep Probabilistic Programming

Edward2

https://github.com/google/edward2

Probabilistic Torch

PROB TORCH

https://github.com/probtorch/probtorch

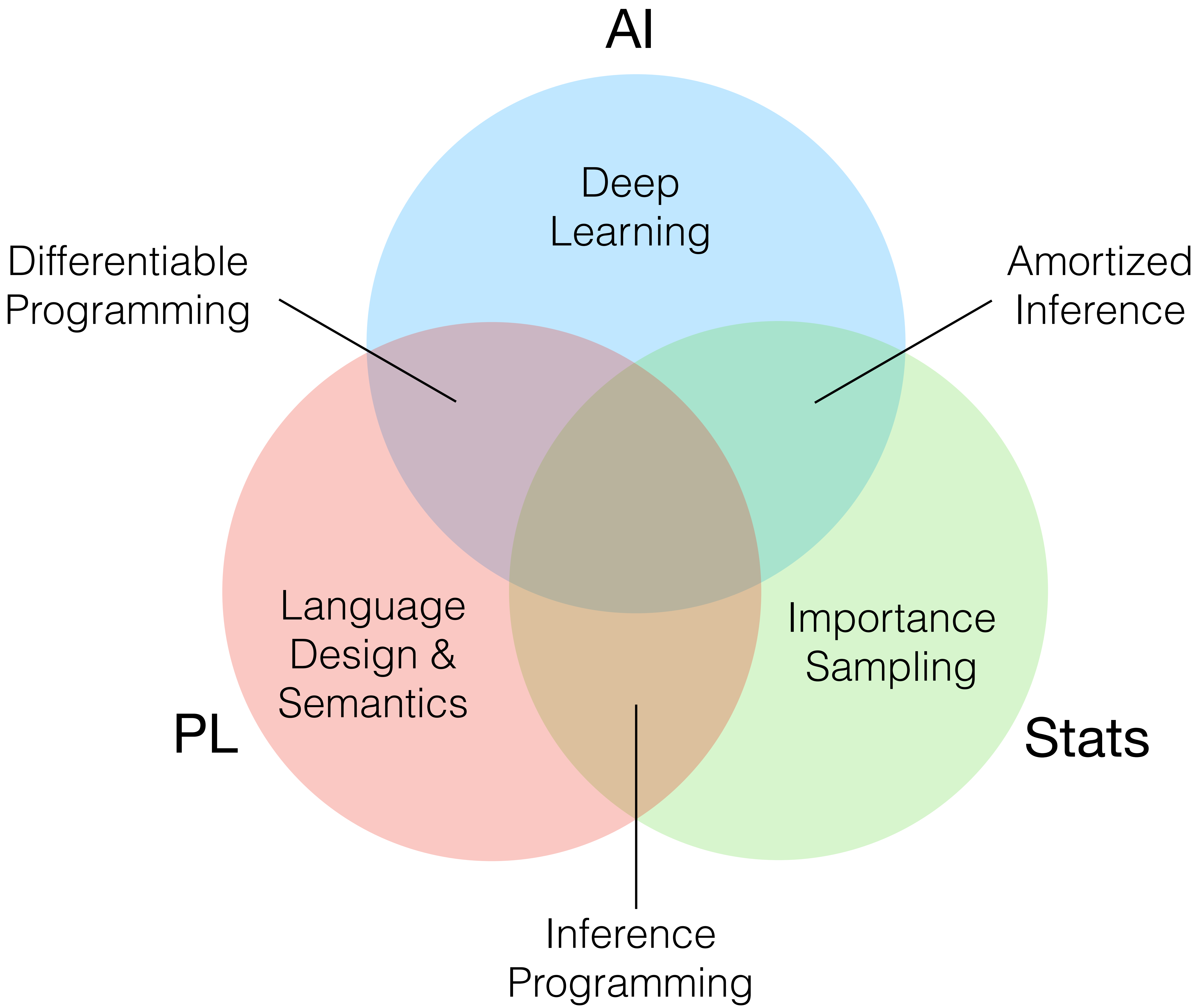
Pyro / NumPyro

https://pyro.ai

1. **Abstractions:** Differentiable Programming, Densities, Sampling, Variational Objectives
2. **Composition:** Deep Generative Models, Amortized Importance Samplers

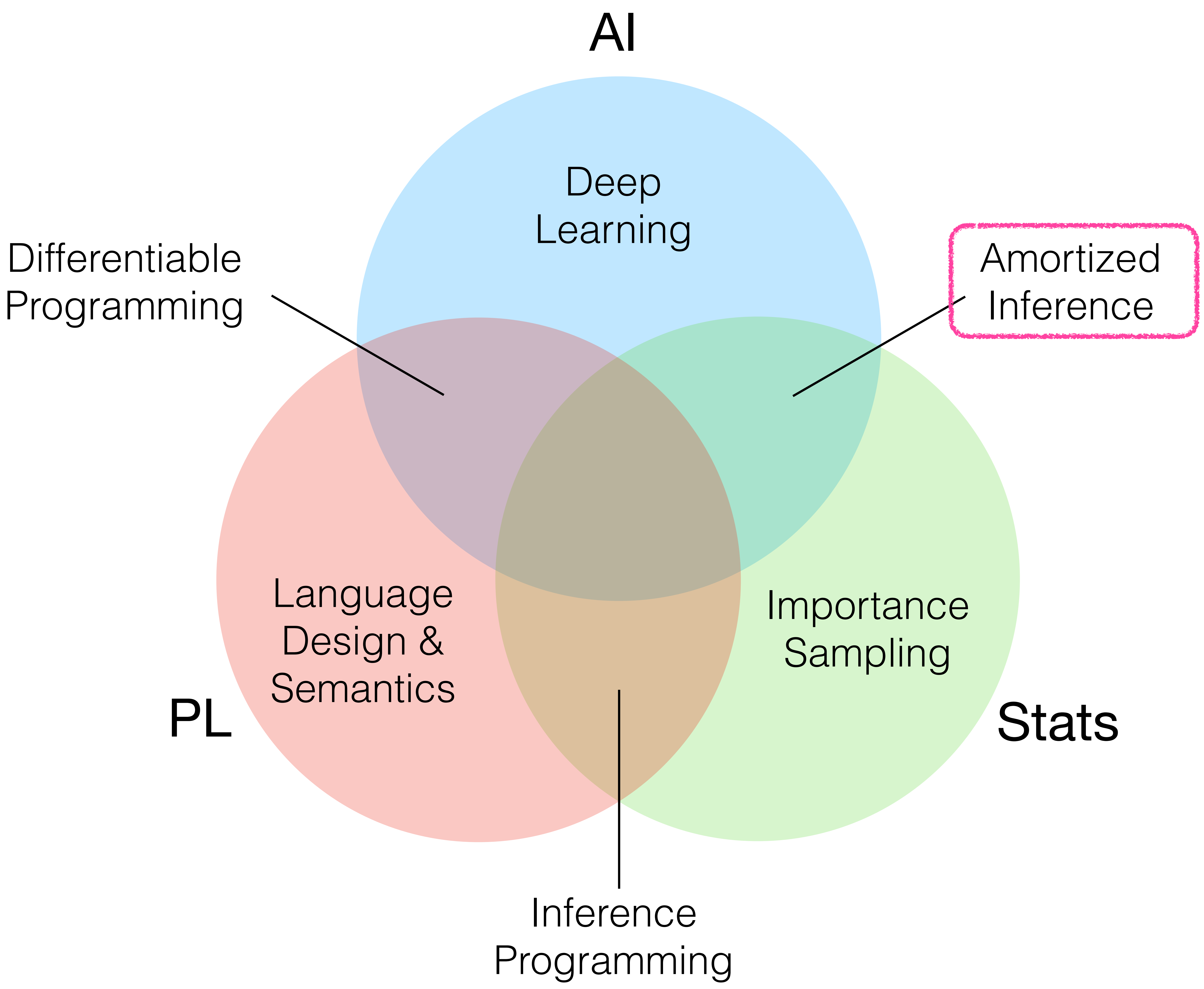
AMLAB Amsterdam Machine Learning Lab

Deep Probabilistic Programming



AMLAB
Amsterdam
Machine Learning Lab

Deep Probabilistic Programming



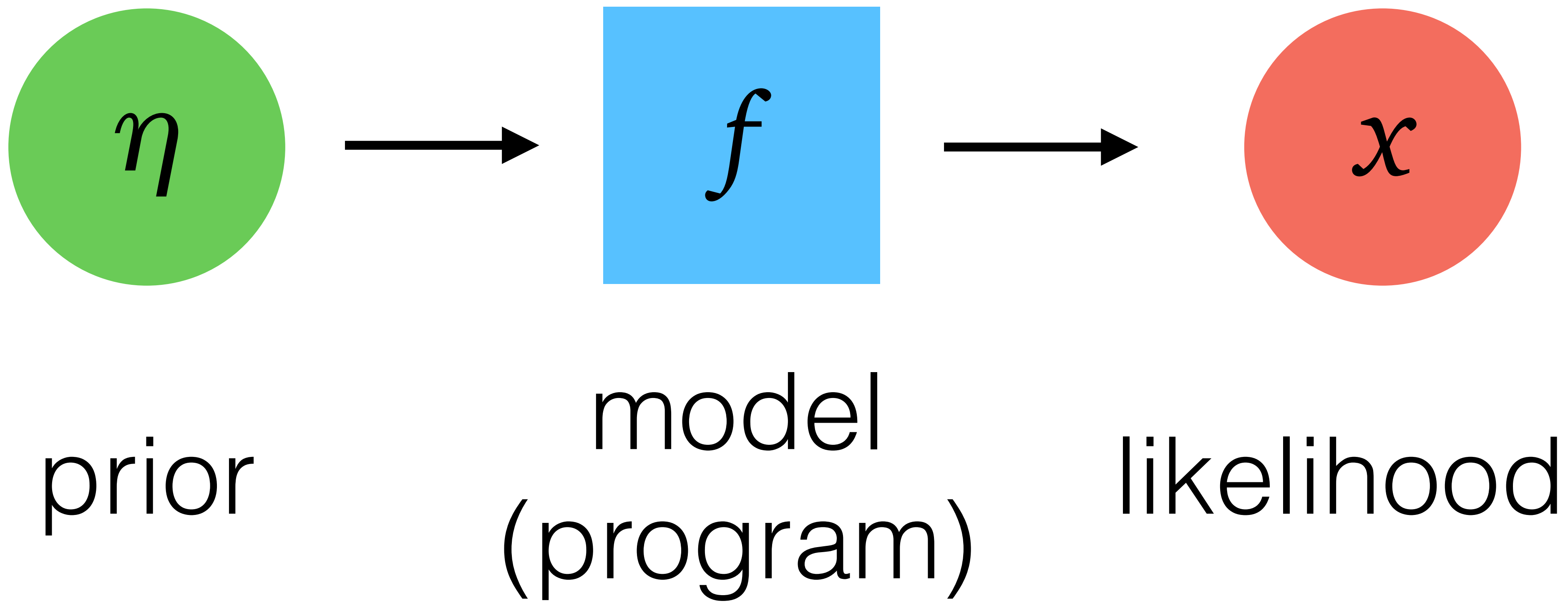
Amortized Inference

Generative Model
(*stochastic simulator*)

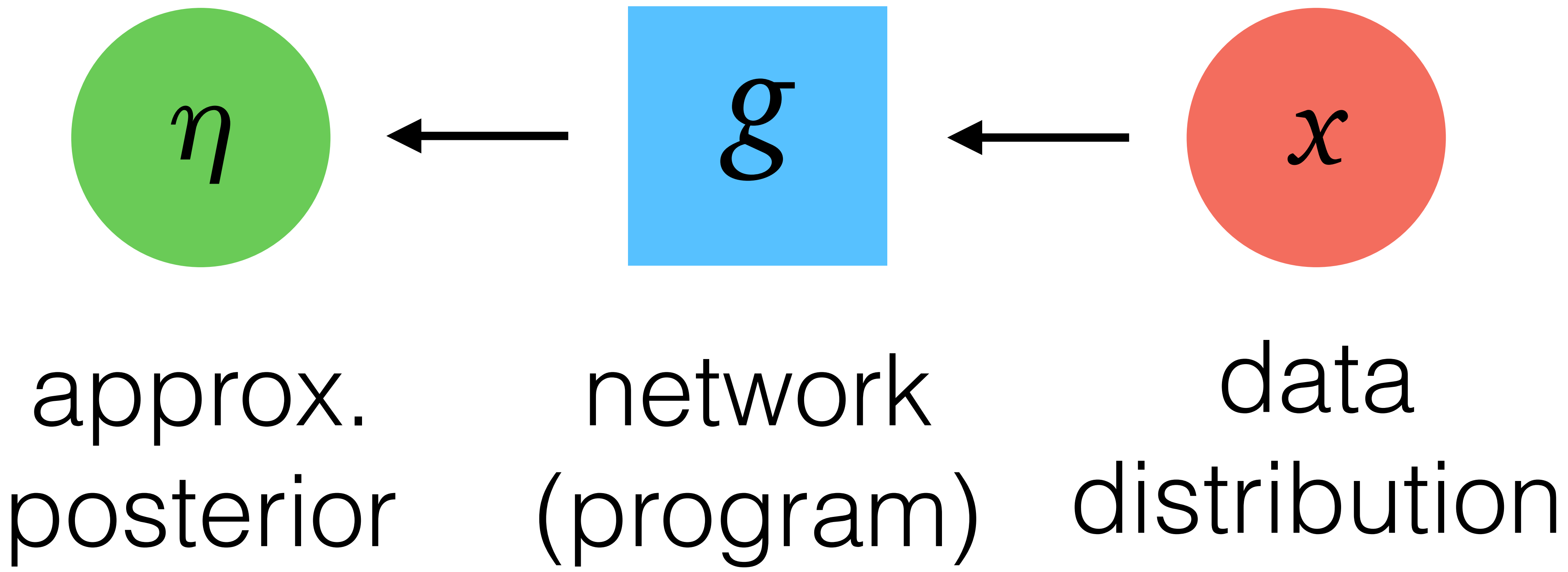
Inference Model
(*approximate inverse*)

Amortized Inference

Generative Model
(*stochastic simulator*)



Inference Model
(*approximate inverse*)



Model Learning

$$\min_{\theta} \boxed{D(q(x) \,||\, p_\theta(x))}$$

Amortized Inference

$$\min_{\phi} \boxed{\mathbb{E}_{x\sim q}\left[D(q_\phi(\eta\,|\,x)\,||\,p_\theta(\eta\,|\,x))\right]}$$

Amsterdam Machine Learning Lab

Reparameterized Variational Inference

Variational Lower Bound

$$\max_{\theta,\phi} \mathscr{L} = \max_{\theta,\phi} \mathop{\mathbb{E}}_{\eta,x\sim q} [\log\ w]$$

Importance Weight

$$w = \frac{p_\theta(x,\eta)}{q_\phi(\eta \mid x)}$$

$$= \min_{\theta,\phi} \left\{ \boxed{D_{\mathrm{KL}}(q_\phi(x) \,||\, p_\theta(x)}) + \boxed{\mathop{\mathbb{E}}_{x\sim q} \left[D_{\mathrm{KL}}(q_\phi(\eta \mid x) \,||\, p_\theta(\eta \mid x)) \right]} \right\}$$

Combines model learning and amortized inference

[Kingma, Welling, ICLR 2014; Rezende, Mohamed, Wierstra, ICML 2014]

AMLAB Amsterdam Machine Learning Lab

Reparameterized Variational Inference

Variational Lower Bound

$$\max_{\theta,\phi} \mathscr{L} = \max_{\theta,\phi} \mathop{\mathbb{E}}_{\eta,x\sim q} [\log\ w]$$

Importance Weight

$$w = \frac{p_\theta(x,\eta)}{q_\phi(\eta \mid x)}$$

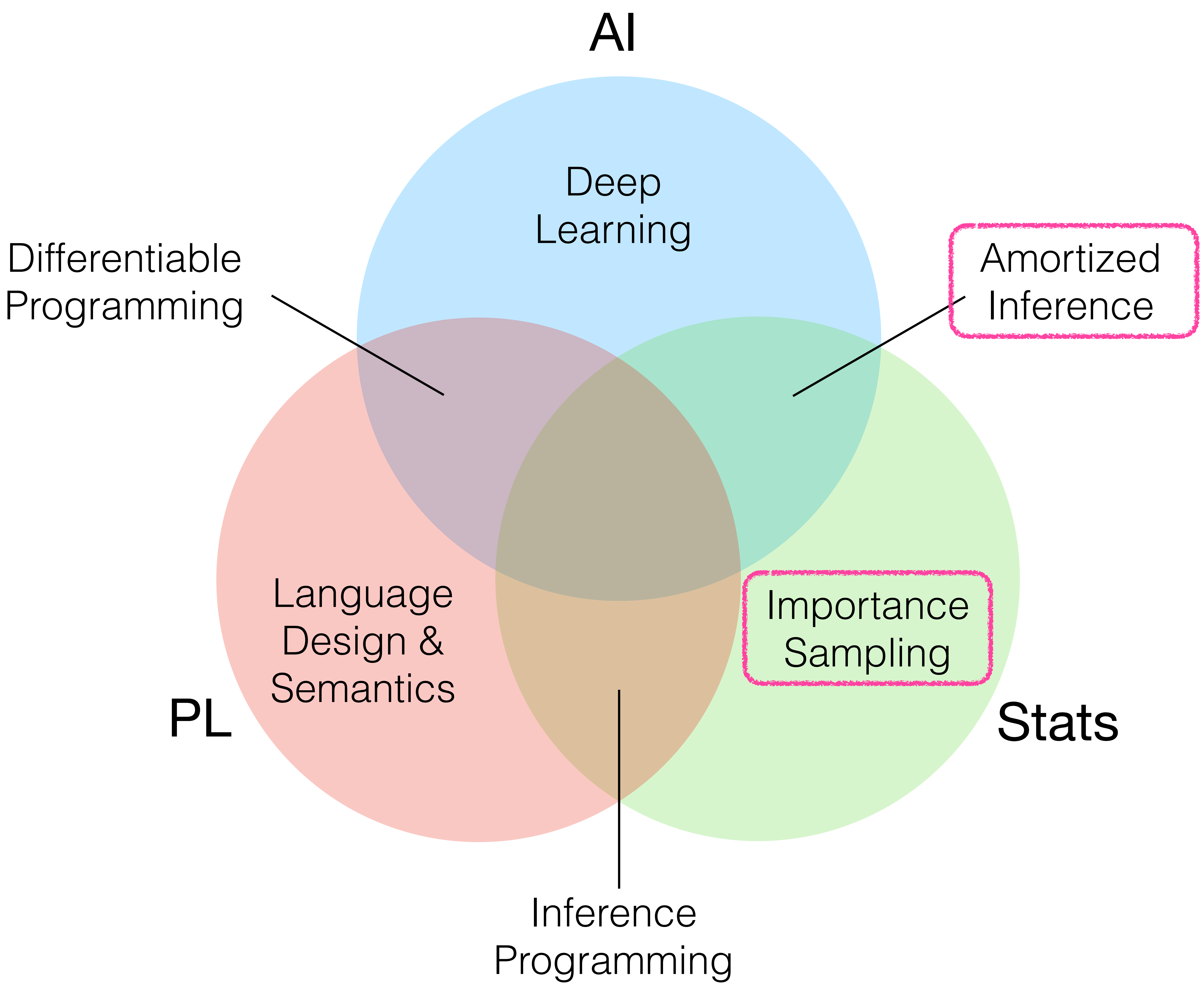
Main Requirement: Fully differentiable model

$$\nabla_\phi \mathop{\mathbb{E}}_{\eta\sim q_\phi} [\log w] = \mathop{\mathbb{E}}_{\epsilon\sim q_0} \Big[(\nabla_\eta \log w)^\top \nabla_\phi \eta(\epsilon) + \nabla_\phi \log w \Big]$$

[Kingma, Welling, ICLR 2014; Rezende, Mohamed, Wierstra, ICML 2014]

AMLAB Amsterdam Machine Learning Lab

Deep Probabilistic Programming



Amsterdam Machine Learning Lab

Minimizing the Inclusive KL divergence

Idea 1: Minimize *inclusive* KL (rather than *exclusive* KL)

$$\min_{\phi} \mathop{\mathbb{E}}_{x \sim q}\Big[D_{\mathrm{KL}}(q_\phi(\eta \mid x) \,||\, p_\theta(\eta \mid x))\Big] \;\rightarrow\; \min_{\phi} \mathop{\mathbb{E}}_{x \sim q}\Big[D_{\mathrm{KL}}(p_\theta(\eta \mid x) \,||\, q_\phi(\eta \mid x))\Big]$$

Idea 2: Use importance sampling to approximate gradient

$$-\nabla_\phi\, D_{\mathrm{KL}}(p_\theta(\eta \mid x) \,||\, q_\phi(\eta \mid x)) = \mathop{\mathbb{E}}_{\eta \sim p_\theta(\cdot \mid x)}\Big[\nabla_\phi \log q_\phi(\eta \mid x)\Big]$$

Use importance sampling

$$w^l = \frac{p_\theta(x, \eta^l)}{q_\phi(\eta^l \mid x)} \qquad \eta^l \sim q_\phi(\eta \mid x)$$

[Bornschein and Bengio, ICLR 2015] [Le, Kosiorek, Siddarth, Teh, Wood, UAI 2019]

AMLAB Amsterdam Machine Learning Lab

Minimizing the Inclusive KL divergence

Idea 1: Minimize *inclusive* KL (rather than *exclusive* KL)

$$\min_{\phi} \mathop{\mathbb{E}}_{x \sim q}\left[D_{\mathrm{KL}}(q_\phi(\eta \mid x) \,||\, p_\theta(\eta \mid x))\right] \;\rightarrow\; \min_{\phi} \mathop{\mathbb{E}}_{x \sim q}\left[D_{\mathrm{KL}}(p_\theta(\eta \mid x) \,||\, q_\phi(\eta \mid x))\right]$$

Idea 2: Use importance sampling to approximate gradient

$$-\nabla_\phi \, D_{\mathrm{KL}}(p_\theta(\eta \mid x) \,||\, q_\phi(\eta \mid x)) = \mathop{\mathbb{E}}_{\eta \sim p_\theta(\cdot \mid x)}\left[\nabla_\phi \log q_\phi(\eta \mid x)\right]$$

$$\simeq \sum_{l=1}^{L} \frac{w^l}{\sum_{l'} w^{l'}} \nabla_\phi \log q_\phi(\eta^l \mid x)$$

Use importance sampling

$$w^l = \frac{p_\theta(x, \eta^l)}{q_\phi(\eta^l \mid x)} \qquad \eta^l \sim q_\phi(\eta \mid x)$$

[Bornschein and Bengio, ICLR 2015] [Le, Kosiorek, Siddarth, Teh, Wood, UAI 2019]

Amortized Importance Samplers

Better gradient estimates

Importance Sampling

Use proposals $q_\phi(\eta \mid x)$
to sample from $p_\theta(\eta \mid x)$

Variational Inference

Learn proposals $q_\phi(\eta \mid x)$
using samples from $p_\theta(\eta \mid x)$

Better proposals

- Does not rely on differentiable models / reparameterization
- Often works as well as, or better than, maximizing a lower bound

[Bornschein and Bengio, ICLR 2015] [Le, Kosiorek, Siddarth, Teh, Wood, UAI 2019]

Amortized Importance Samplers

Better gradient estimates

Importance Sampling

Use proposals $q_\phi(\eta \mid x)$ to sample from $p_\theta(\eta \mid x)$

Variational Inference

Learn proposals $q_\phi(\eta \mid x)$ using samples from $p_\theta(\eta \mid x)$

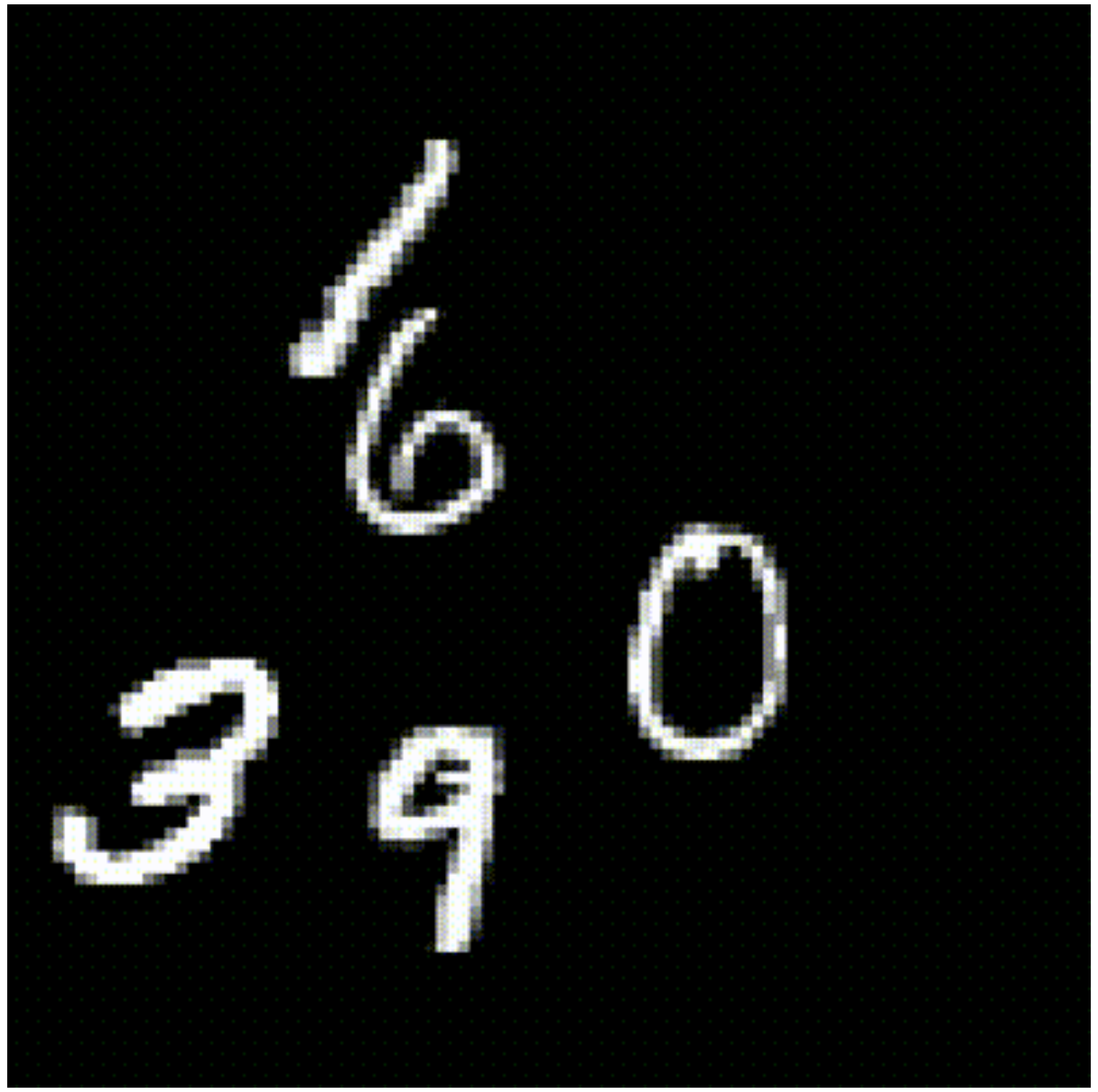
Better proposals

Opportunity: New VI methods based on SMC samplers, nested importance samplers, etc

[Bornschein and Bengio, ICLR 2015] [Le, Kosiorek, Siddarth, Teh, Wood, UAI 2019]

Example (~2019): Amortized Population Gibbs

Task: Unsupervised Tracking



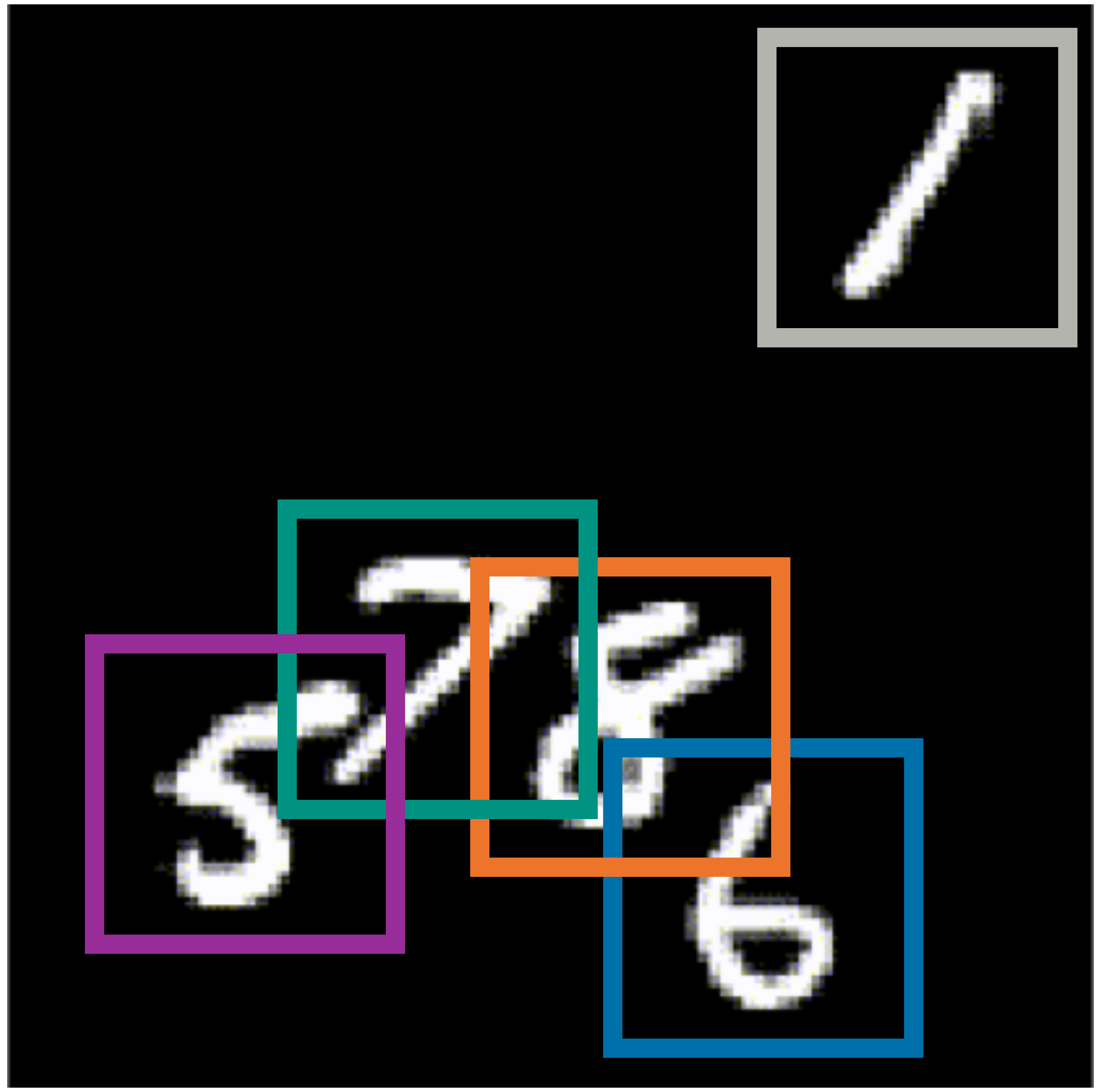
- Corpus level (*many videos*)
  Digit shapes
  Transition dynamics

- Instances (*single videos*)
  Object representations

- Data-points (*single frames*)
  Object positions

[Wu, Zimmermann, Sennesh, Le, van de Meent, ICML 2020]

Example (~2019): Amortized Population Gibbs



Classic Chicken-and-Egg Problem

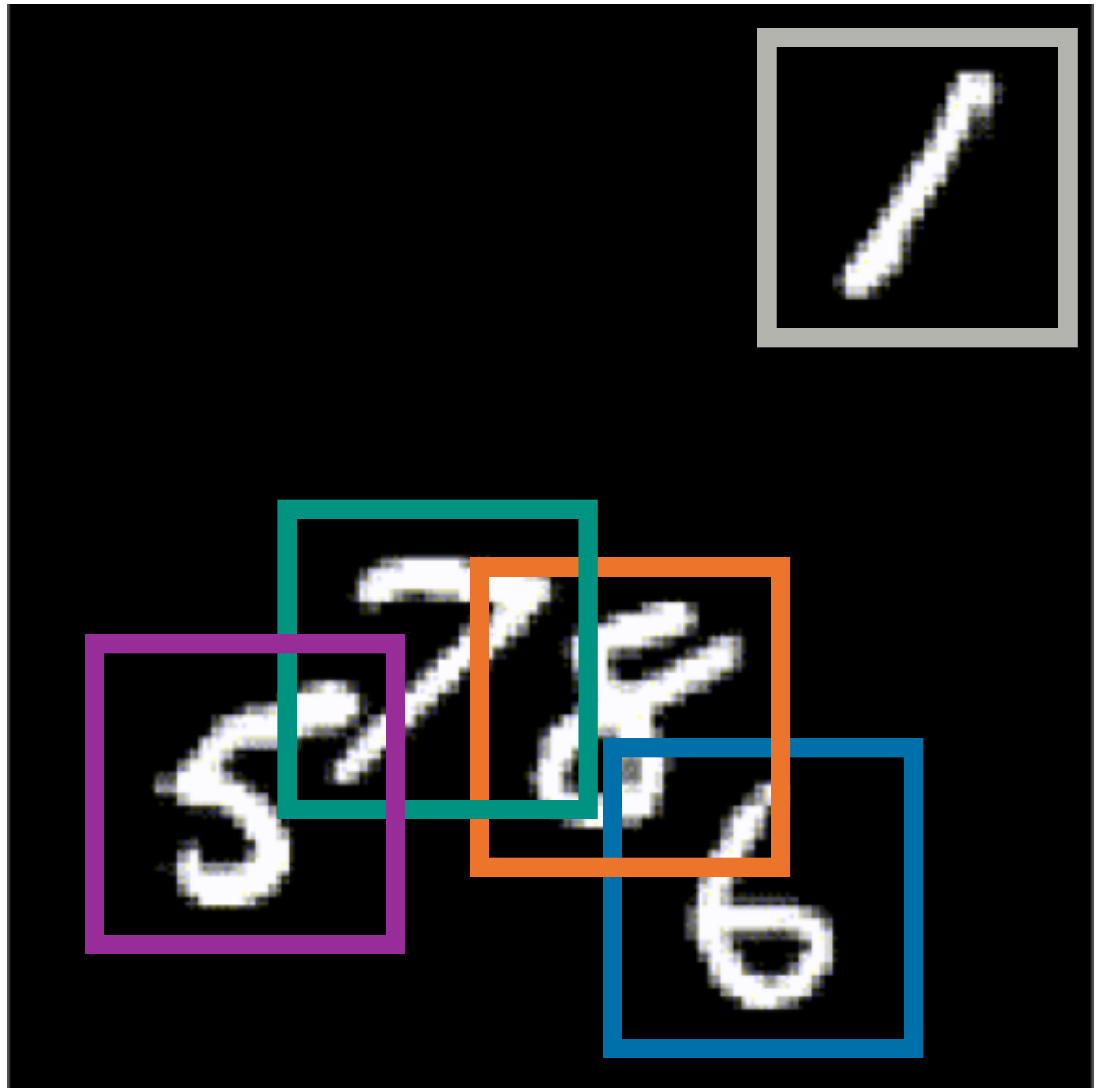
- **Easy:** Infer object representations given object positions

- **Also Easy:** Infer positions given object representations

- **Not Easy:** Joint inference of positions and representations

[Wu, Zimmermann, Sennesh, Le, van de Meent, ICML 2020]

Example (~2019): Amortized Population Gibbs



Classic Solution: Iterate

- **Step 0:** Initialize representations and positions.
- **Update 1:** Infer object representations given object positions

$\eta \sim p(\eta \mid x, z)$

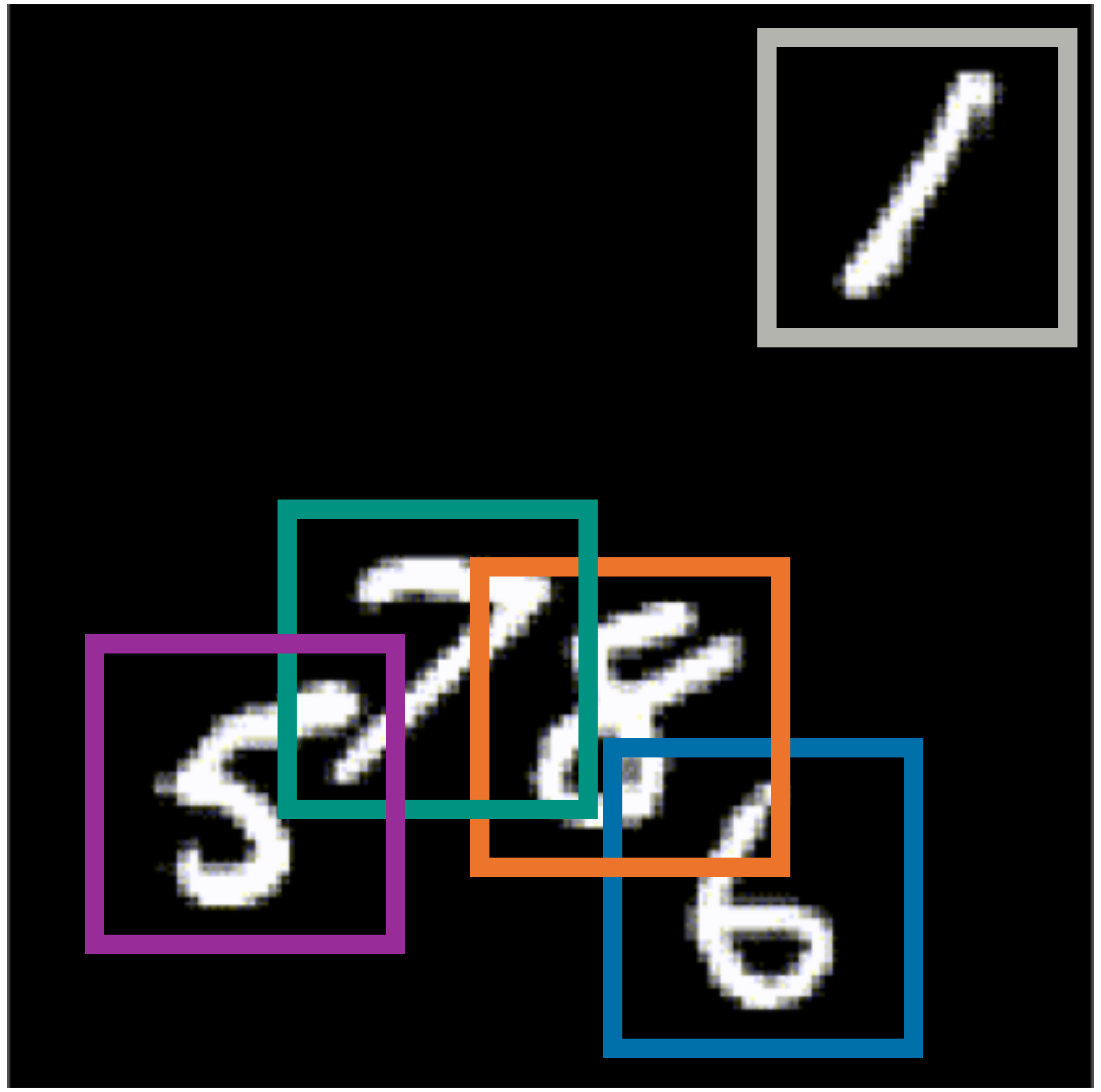
- **Update 2:** Infer object representations given object positions

$z \sim p(z \mid x, \eta)$

Problem:
Only computable in conjugate exponential family models

[Wu, Zimmermann, Sennesh, Le, van de Meent, ICML 2020]

Example (~2019): Amortized Population Gibbs



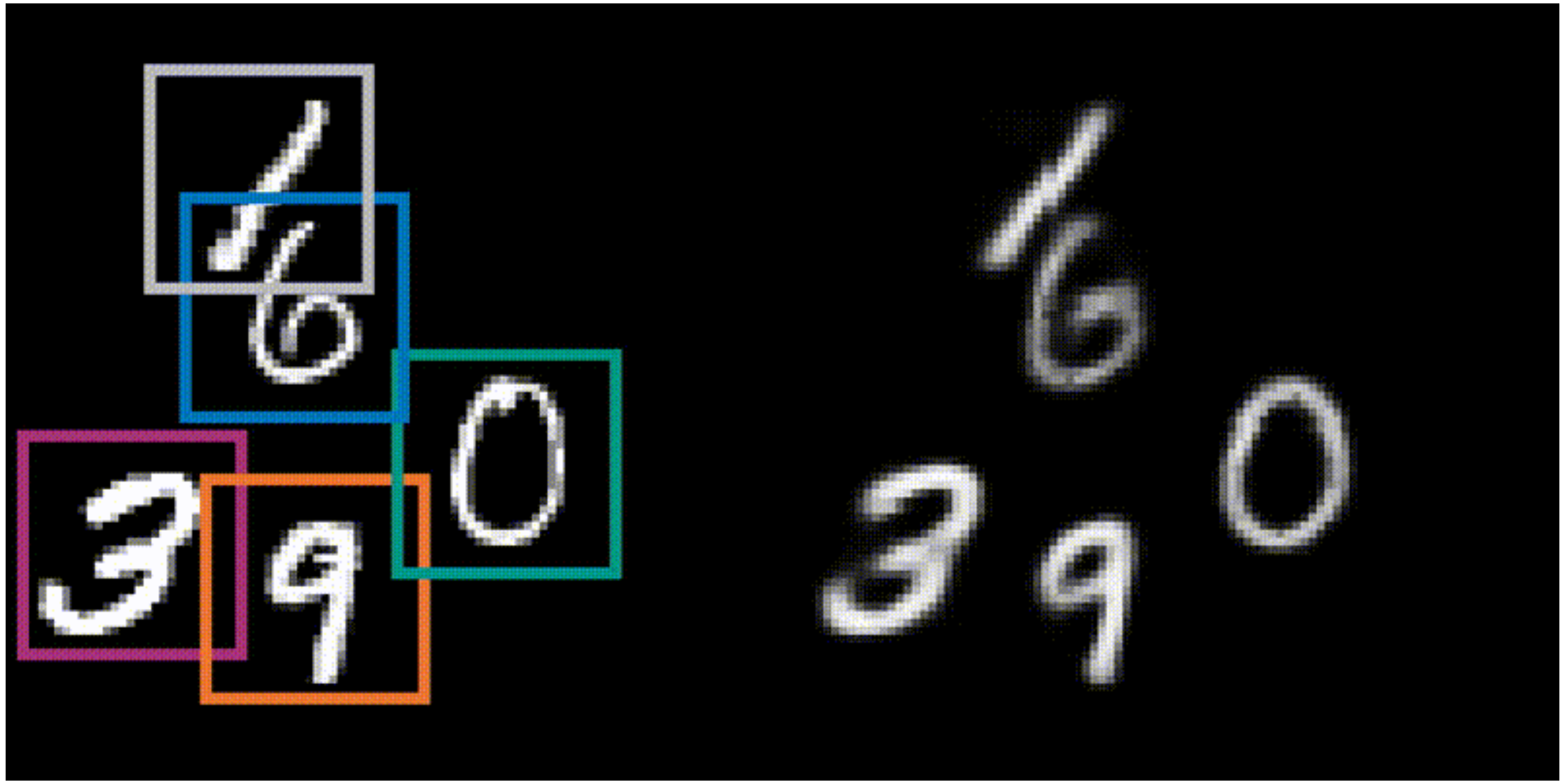
Modern solution: Learn Updates

- **Step 0:** Initialize representations and positions.
- **Update 1:** Infer object representations given object positions
  $\eta \sim q_\phi(\eta \mid x, z)$
- **Update 2:** Infer object representations given object positions
  $z \sim q_\phi(z \mid x, \eta)$

[Wu, Zimmermann, Sennesh, Le, van de Meent, ICML 2020]

Example (~2019): Amortized Population Gibbs

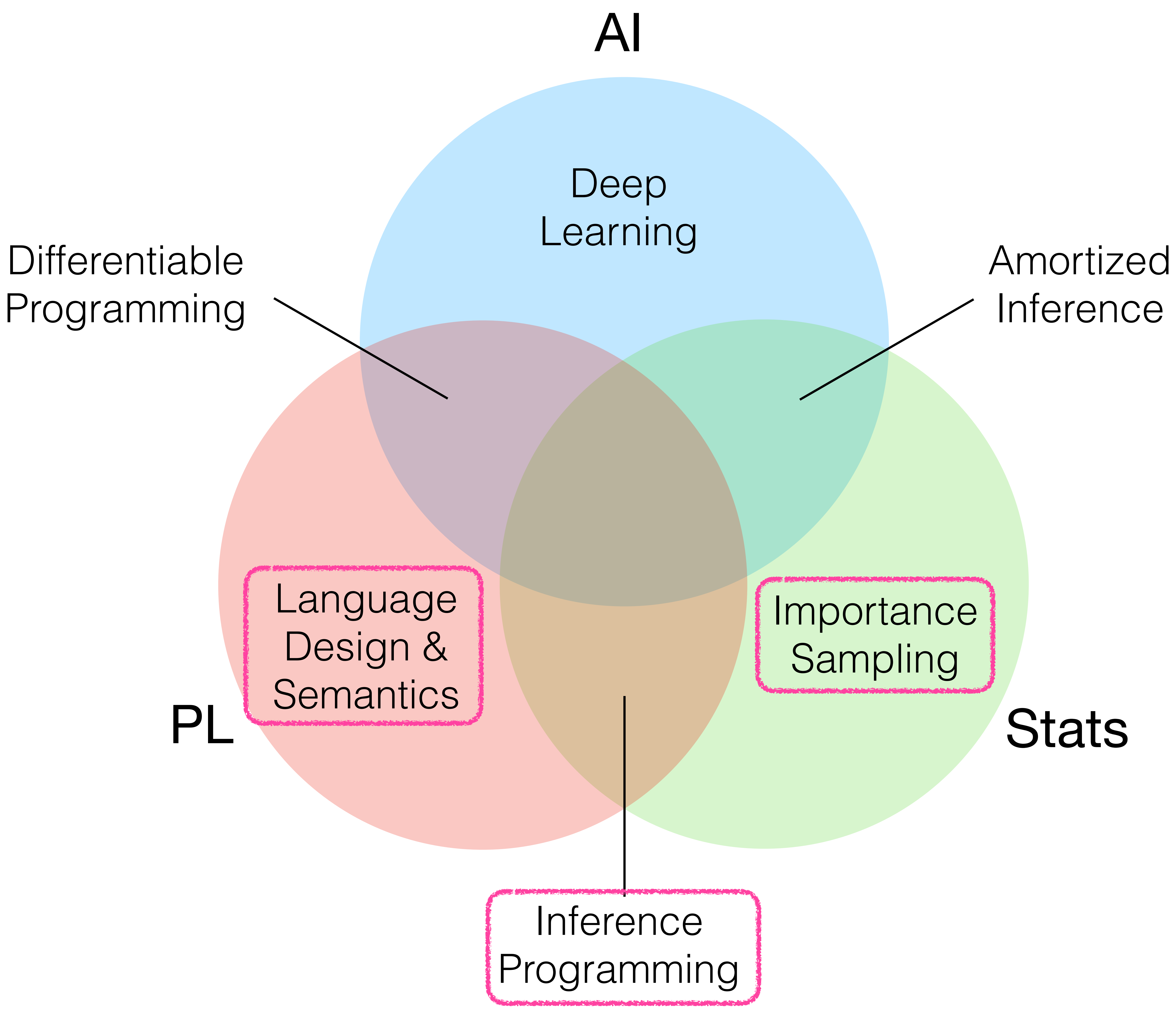
Inferred Positions　　　Reconstructions



- Completely unsupervised
- Computationally efficient (~5 updates, ~10 particles)

[Wu, Zimmermann, Sennesh, Le, van de Meent, ICML 2020]

Deep Probabilistic Programming



AMLAB
Amsterdam Machine Learning Lab

Reasoning Compositionally About Inference

What (inference) DSL could define this sampler / variational method?

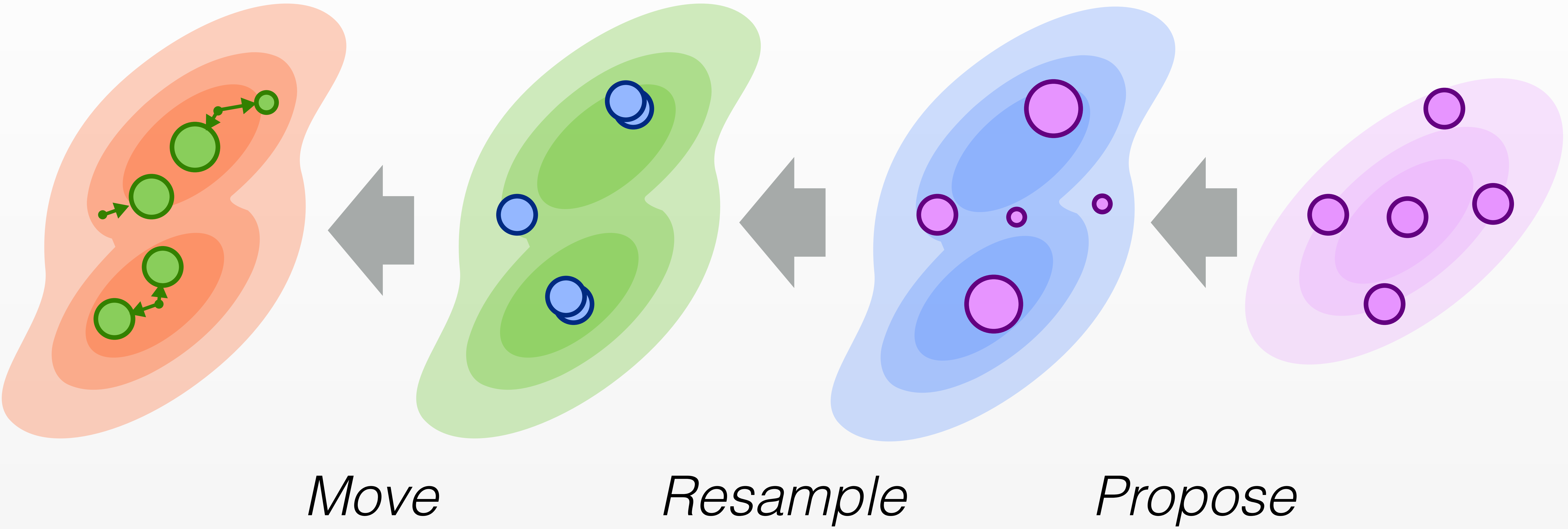
Algorithm 1 Amortized Population Gibbs Sampling

---

for n **in** $1, \ldots, N$ **do**
  $G_\phi = 0$
  $x^n \sim p^{\text{DATA}}(x)$
  for l **in** $1, \ldots, L$ **do**
    $z^{n,1,l} \sim q_\phi(z \mid x^n)$
    $w^{n,1,l} \leftarrow p_\theta(x^n, z^{n,1,l}) \,/\, q_\phi(z^{n,1,l})$
  for k **in** $2, \ldots, K$ **do**
    $\tilde{z}, \tilde{w} = z^{n,k-1}, w^{n,k-1}$
    for b **in** $1, \ldots, B$ **do**
      $\tilde{z}, \tilde{w} = \text{RESAMPLE}(\tilde{z}, \tilde{w})$
      for l **in** $1, \ldots, L$ **do**
        $\tilde{z}_b'^{\,l} \sim q_\phi(\cdot \mid x^n, \tilde{z}_{-b}^l)$
        $\tilde{w}^l = \frac{p_\theta(x^n, \tilde{z}_b'^{\,l}, \tilde{z}_{-b}^l)\, q_\phi(\tilde{z}_b^l \mid x^n, \tilde{z}_{-b}^l)}{p_\theta(x^n, \tilde{z}_b^{\,l}, \tilde{z}_{-b}^l)\, q_\phi(\tilde{z}_b'^{\,l} \mid x^n, \tilde{z}_{-b}^l)}\, \tilde{w}^l$
        $\tilde{z}_b^l = \tilde{z}_b'^{\,l}$
      $G_\phi = G_\phi + \sum_{l=1}^{L} \frac{\tilde{w}^l}{\sum_{l'} \tilde{w}^{l'}} \frac{d}{d\phi} \log q_\phi(\tilde{z}_b^l \mid x^n, \tilde{z}_{-b}^l)$
    $z^{n,k}, w^{n,k} = \tilde{z}, \tilde{w}$
return G_ϕ, z, w ▷ Output: Grac

- APG is an example of a amortized SMC sampler
- Known building blocks, but not trivial to combine correctly
- Can we define compositional methods for importance sampling and gradient estimation?

Combinators: A DSL for Inference



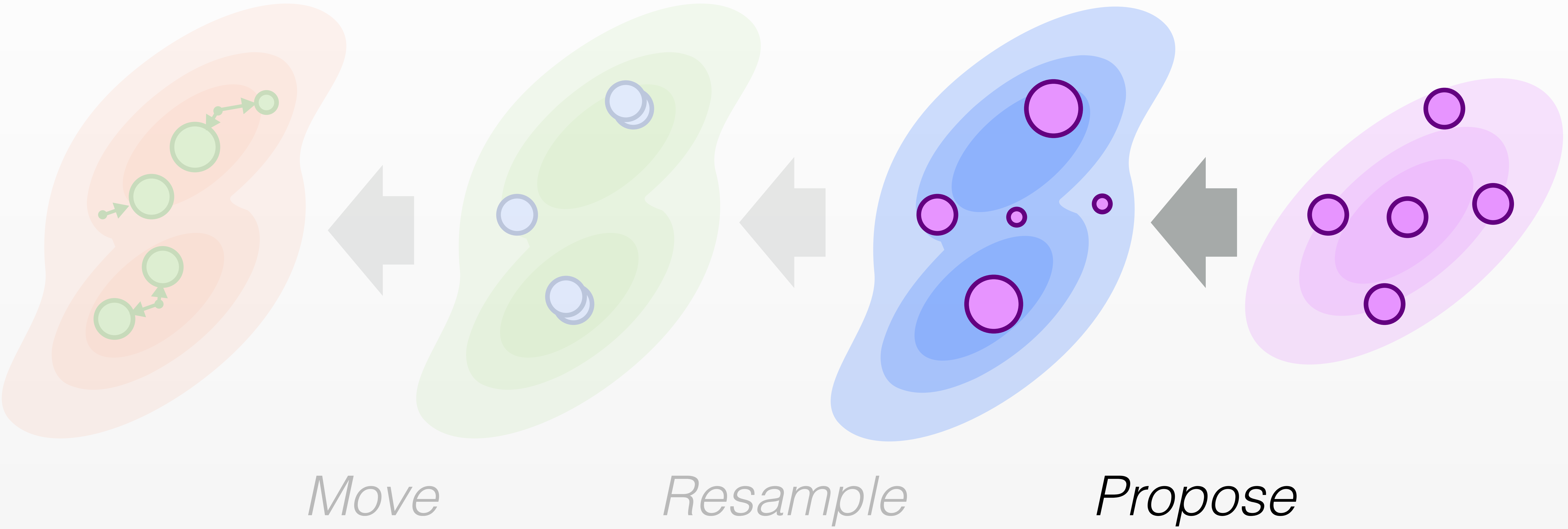
$\mathtt{f} ::=$ A primitive program

$\mathtt{p} ::= \mathtt{f} \mid \mathtt{extend(p,\ f)}$

$\mathtt{q} ::= \mathtt{p} \mid \mathtt{resample(q)} \mid \mathtt{compose(q',\ q)} \mid \mathtt{propose(p,\ q)}$

[Stites, Zimmerman, Wu, Sennesh, van de Meent, UAI 2021]

Combinators: A DSL for Inference



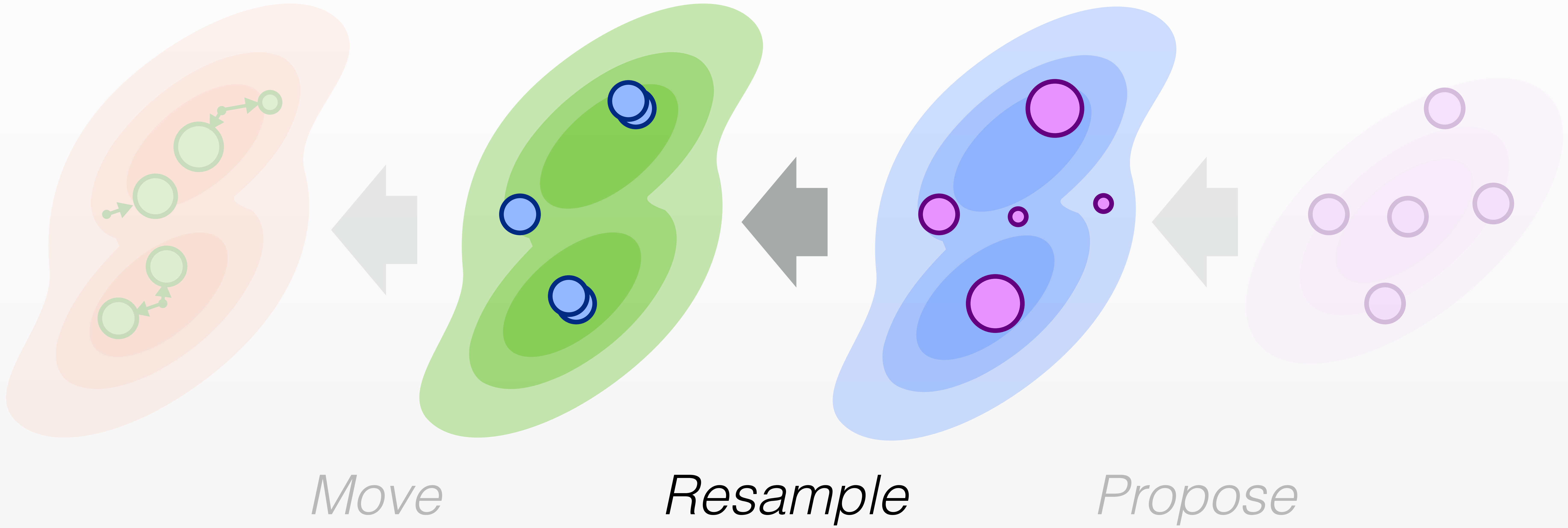
f ::= A primitive program

p ::= f | `extend(p, f)`

q ::= p | `resample(q)` | `compose(q′, q)` | `propose(p, q)`

[Stites, Zimmerman, Wu, Sennesh, van de Meent, UAI 2021]

Combinators: A DSL for Inference



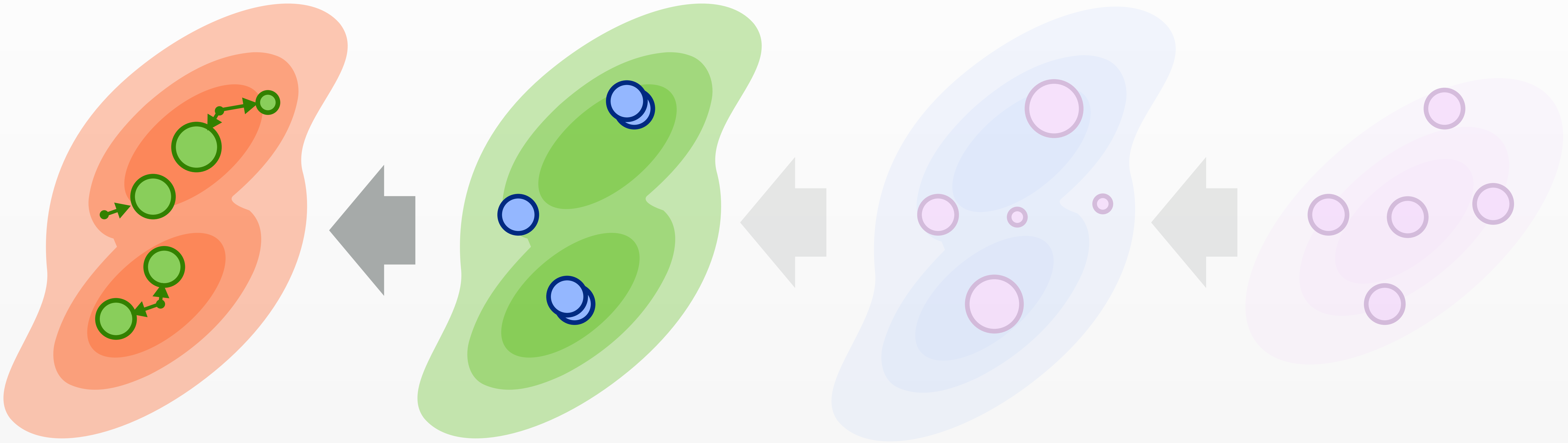
f ::= A primitive program

p ::= f | `extend(p, f)`

q ::= p | `resample(q)` | `compose(q′, q)` | `propose(p, q)`

[Stites, Zimmerman, Wu, Sennesh, van de Meent, UAI 2021]

Combinators: A DSL for Inference



Move *Resample* *Propose*

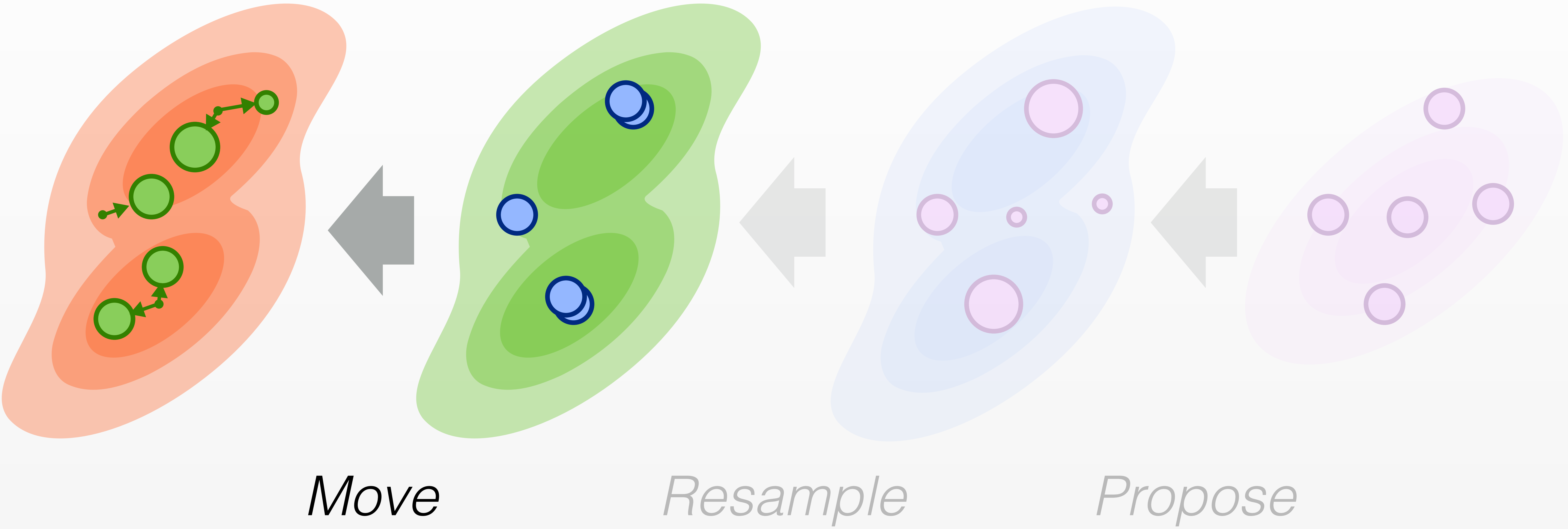
$f ::=$ A primitive program

$p ::= f \mid$ `extend(p, f)`

$q ::= p \mid$ `resample(q)` $\mid$ `compose(q′, q)` $\mid$ `propose(p, q)`

[Stites, Zimmerman, Wu, Sennesh, van de Meent, UAI 2021]

Combinators: A DSL for Inference



f ::= A primitive program

p ::= f | `extend(p, f)`

q ::= p | `resample(q)` | `compose(q′, q)` | `propose(p, q)`

[Stites, Zimmerman, Wu, Sennesh, van de Meent, UAI 2021]

Core Property: Proper Weighting

Definition: A pair **w, z** is properly weighted with respect to a density **π(z)** when, for all measurable **h(z)**,

$$\mathbb{E}_{w,z\sim\Pi}\Big[w\,h(z)\Big] = \mathscr{Z}\,\mathbb{E}_{z\sim\pi}\Big[h(z)\Big]$$

Sampler
(*can be a black box*)

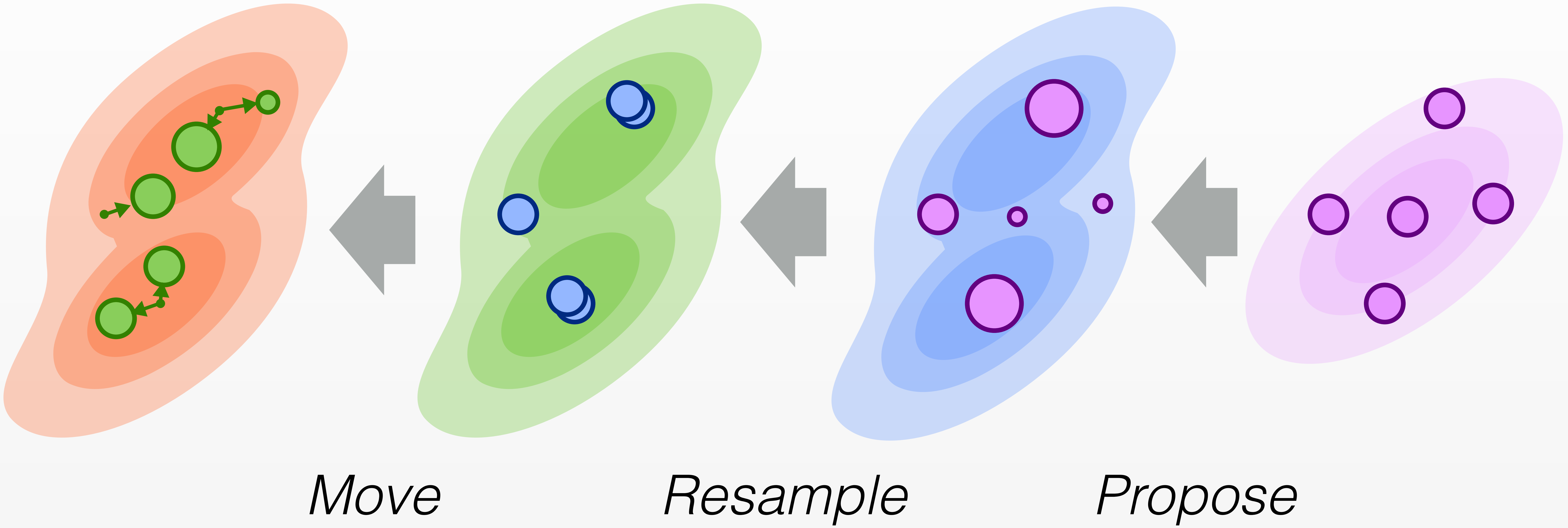
Quantity of interest
(*return value of program*)

Constant of proportionality
(*marginal likelihood*)

Density of interest
(*program posterior*)

[Naesseth, Lindsten, Schön, Foundations and Trends in Machine Learning, 2019]

Combinators: A DSL for Inference



Semantics: Composition preserves proper weighting

https://github.com/probtorch/combinators

(Pyro implementation forthcoming)

[Stites, Zimmerman, Wu, Sennesh, van de Meent, UAI 2021]

Example: Amortized Gibbs Samplers

```
def pop_gibbs(target, proposal, kernels, sweeps):
  q = propose(partial(target, suffix=0),
              partial(proposal, suffix=0))
  for s in range(sweeps):
    for k in kernels:
      q = propose(
          extend(partial(target, suffix=s+1),
                 partial(k, suffix=s)),
          compose(partial(k, suffix=s+1),
                  resample(q, dim=0)))
    return q
```

Algorithm 1 Amortized Population Gibbs Sampling

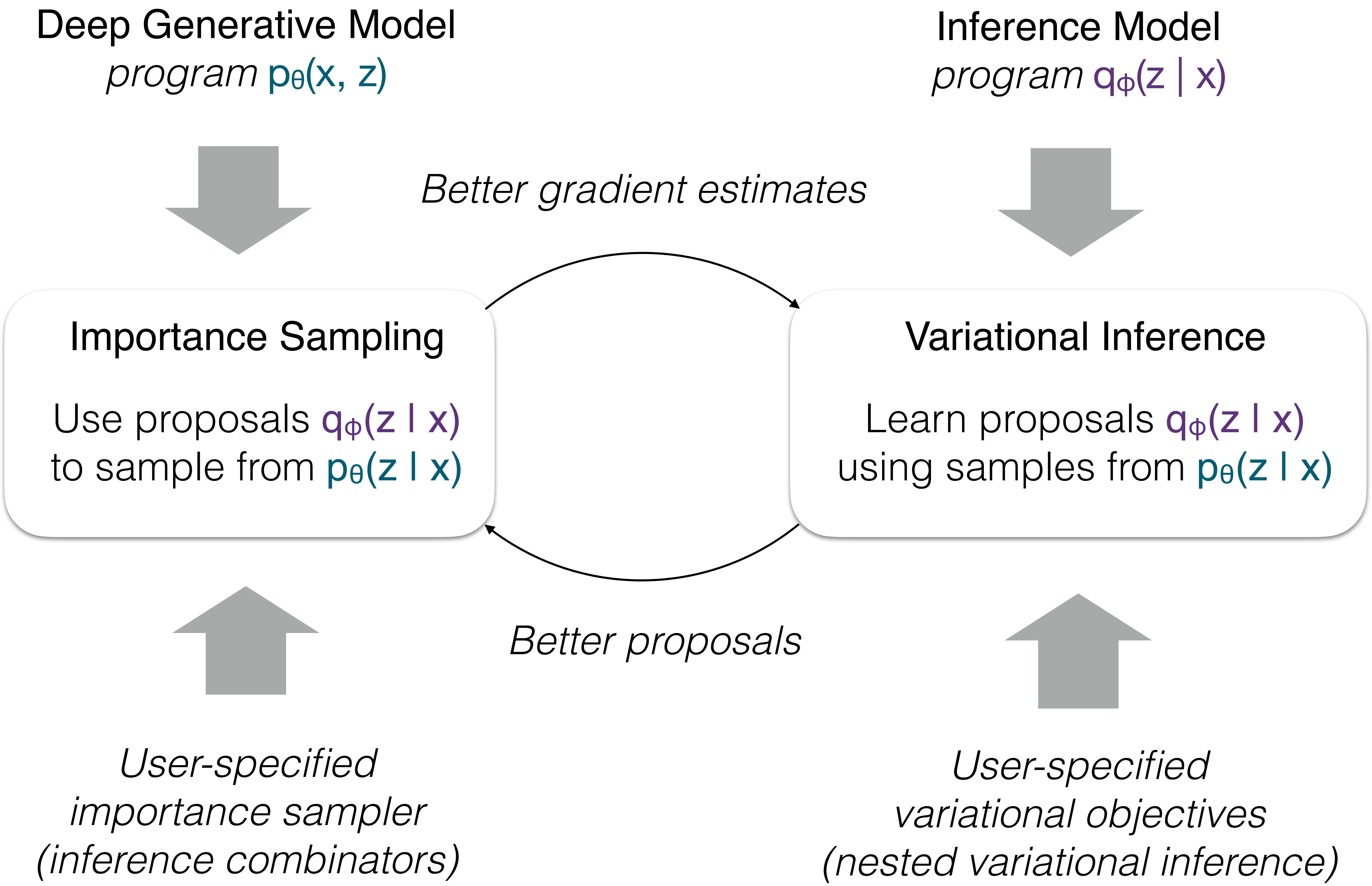
for n **in** $1, \ldots, N$ **do**
  $G_\phi = 0$
  $x^n \sim p^{\text{DATA}}(x)$
  for l **in** $1, \ldots, L$ **do**
    $z^{n,1,l} \sim q_\phi(z \mid x^n)$
    $w^{n,1,l} \leftarrow p_\theta(x^n, z^{n,1,l}) \,/\, q_\phi(z^{n,1,l})$
  for k **in** $2, \ldots, K$ **do**
    $\tilde{z}, \tilde{w} = z^{n,k-1}, w^{n,k-1}$
    for b **in** $1, \ldots, B$ **do**
      $\tilde{z}, \tilde{w} = \text{RESAMPLE}(\tilde{z}, \tilde{w})$
      for l **in** $1, \ldots, L$ **do**
        $\tilde{z}'^{\,l}_b \sim q_\phi(\cdot \mid x^n, \tilde{z}^l_{-b})$
        $\tilde{w}^l = \frac{p_\theta(x^n, \tilde{z}'^{\,l}_b, \tilde{z}^l_{-b})\, q_\phi(\tilde{z}^l_b \mid x^n, \tilde{z}^l_{-b})}{p_\theta(x^n, \tilde{z}^l_b, \tilde{z}^l_{-b})\, q_\phi(\tilde{z}'^{\,l}_b \mid x^n, \tilde{z}^l_{-b})}\, \tilde{w}^l$
        $\tilde{z}^l_b = \tilde{z}'^{\,l}_b$
      $G_\phi = G_\phi + \sum_{l=1}^{L} \frac{\tilde{w}^l}{\sum_{l'} \tilde{w}^{l'}} \frac{d}{d\phi} \log q_\phi(\tilde{z}^l_b \mid x^n, \tilde{z}^l_{-b})$
    $z^{n,k}, w^{n,k} = \tilde{z}, \tilde{w}$
return G_ϕ, z, w ▷ Output: Grad

High-level algorithm description
(transition operators, resampling)

Low-level algorithm description
(weight and gradient computations)

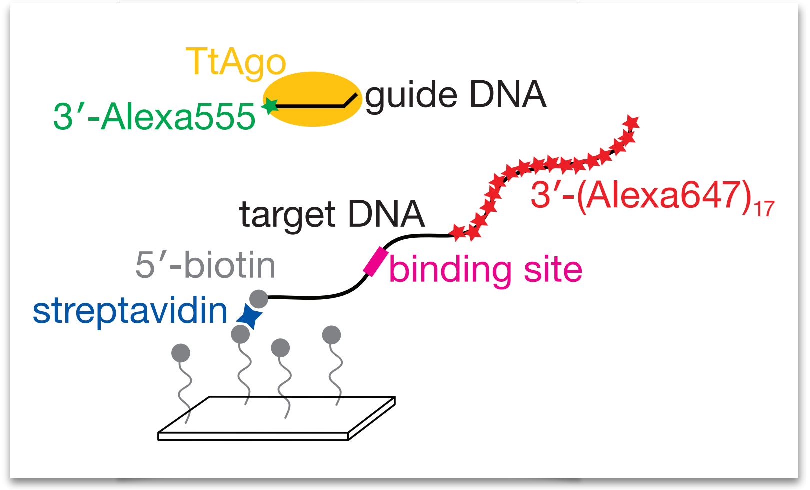
[Stites, Zimmerman, Wu, Sennesh, van de Meent, *UAI*, 2021]

Differentiable + Probabilistic + Inference Programming

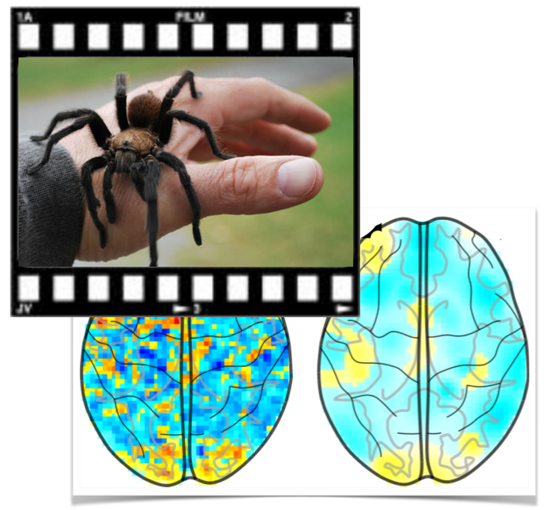


The Next 700 Models in AI

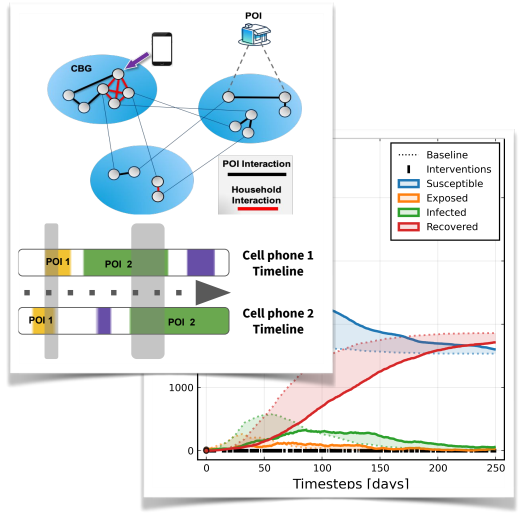
Biophysics



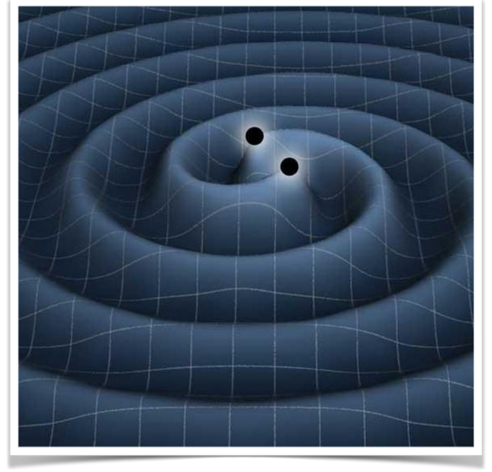
Neuroimaging



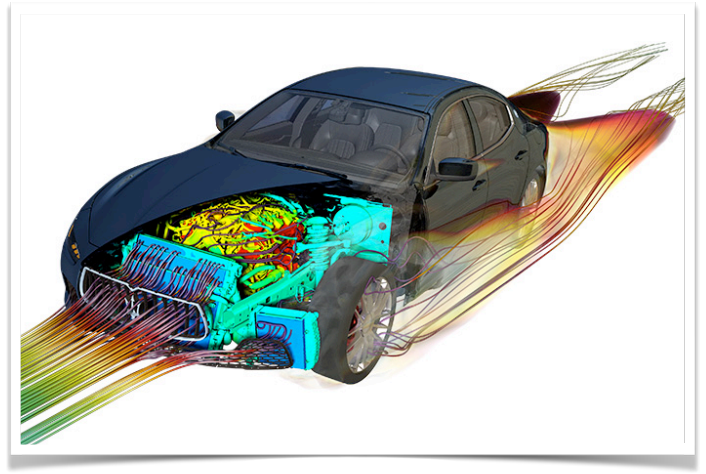
Epidemiology



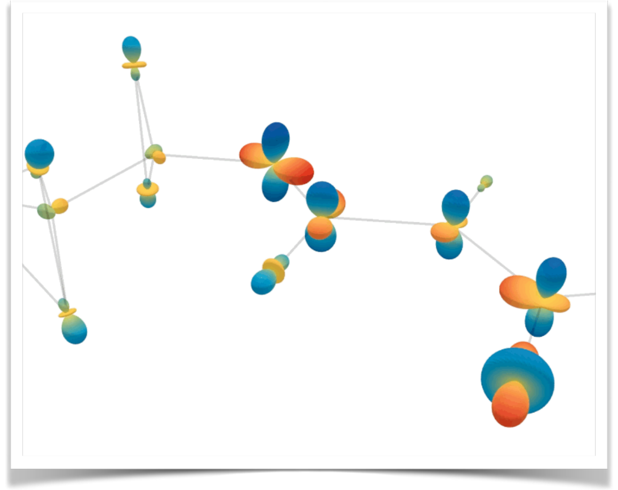
Astrophysics



Computational Fluid Dynamics



Molecular Design



Manufacturing

Abstractions for Emerging Problems:

- Learning surrogate models
- Modeling search spaces
- Inferring differential equations

AMLAB Amsterdam Machine Learning Lab

Thank You!

University of Amsterdam

Northeastern University

Heiko Zimmermann

Babak Esmaeili

Sam Stites

Hao Wu

Eli Sennesh

Nested Variational Inference
H. Zimmermann, H. Wu, Babak Esmaeili, J.-W. van de Meent
NeurIPS 2021 [https://arxiv.org/abs/2103.00668]

Learning Proposals for Probabilistic Programs with Inference Combinators
S. Stites*, H. Zimmermann*, H. Wu, E. Sennesh, J.-W. van de Meent
UAI 2021 [https://arxiv.org/abs/2103.00668]

Amortized Population Gibbs Samplers with Neural Sufficient Statistics
H. Wu, H. Zimmermann, E. Sennesh, Tuan Anh Le, J.-W. van de Meent
ICML 2020 [https://proceedings.icml.cc/static/paper_files/icml/2020/5881-Paper.pdf]

An Introduction to Probabilistic Programming
J.-W. van de Meent, B. Paige, H. Yang, F. Wood
ArXiv 2018 [https://arxiv.org/abs/1809.10756]