Reinforcement Learning in Age of Large Scale Foundation Models

Invited Lecturer :
Zarif Ikram

National University of Singapore

Lecture 6

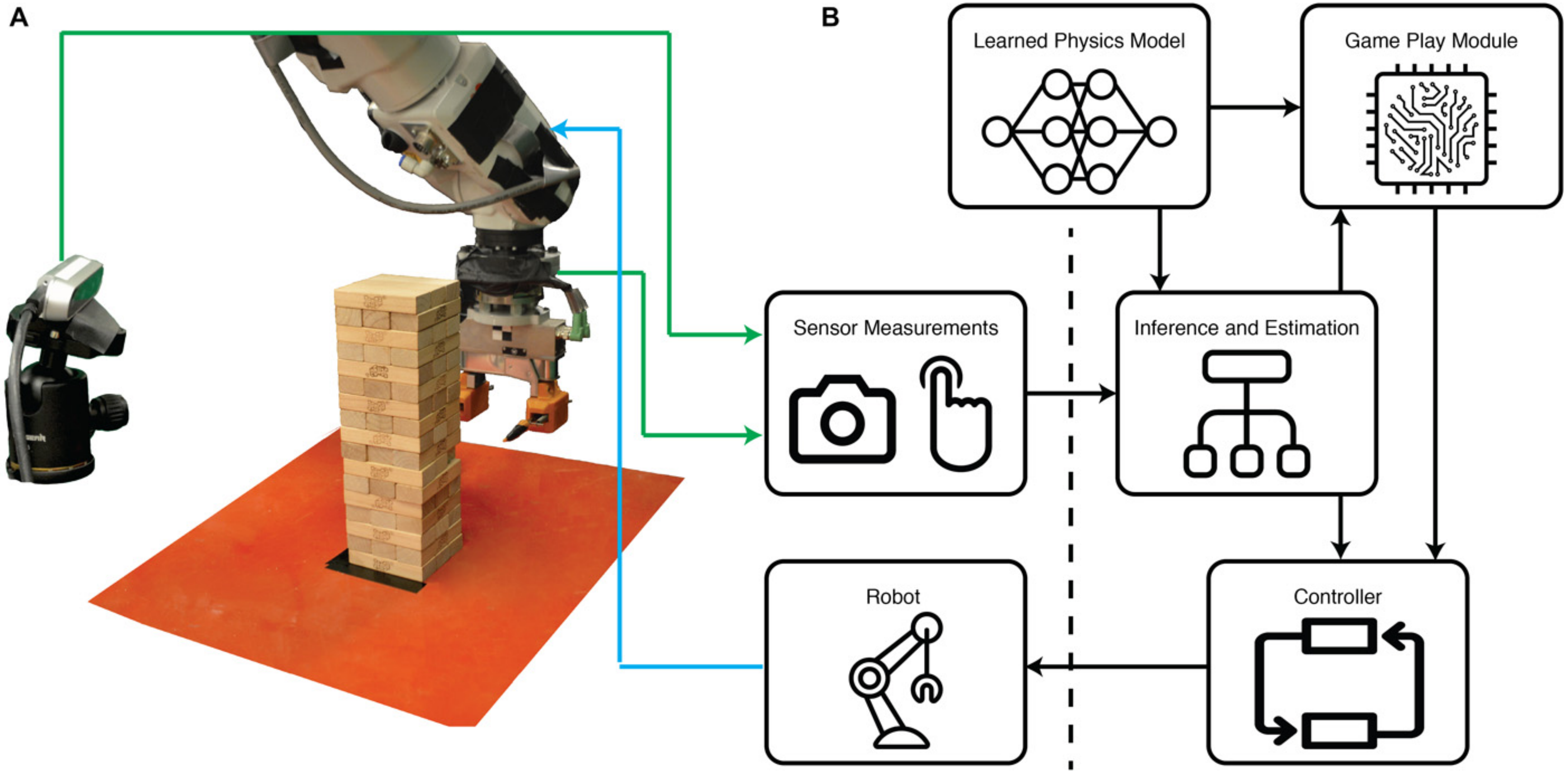
How to Train Your World Models

Introduction

Why World Models?

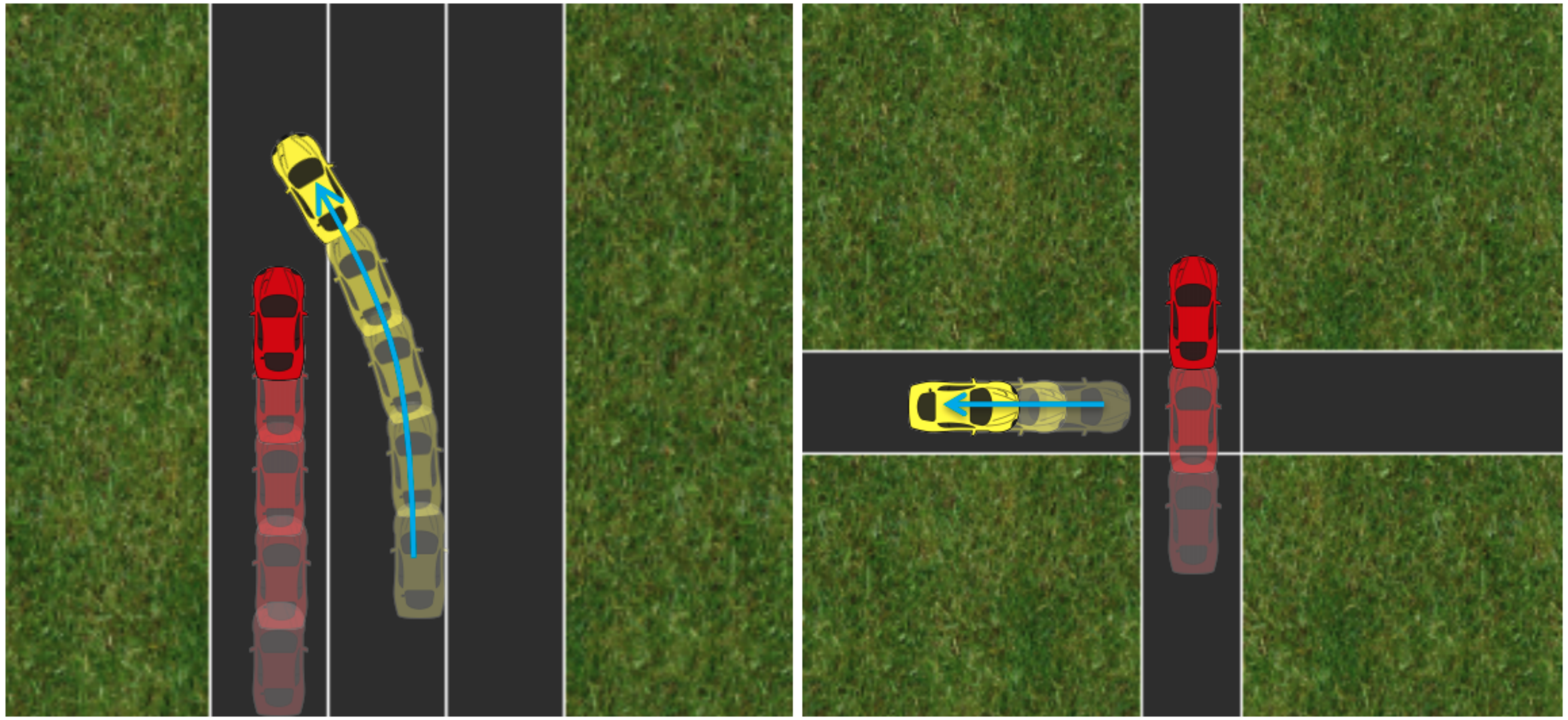
- Model Predictive Control (MPC)
- Model Based Reinforcement Learning (MBRL)
- Model Based Reasoning

Model-based reasoning for **robotic control**



Fazeli et al. (2019). See, feel, act: Hierarchical learning for complex manipulation skills with multisensory fusion. Science Robotics, 4(26).

Model-based reasoning for **human-AI interaction**

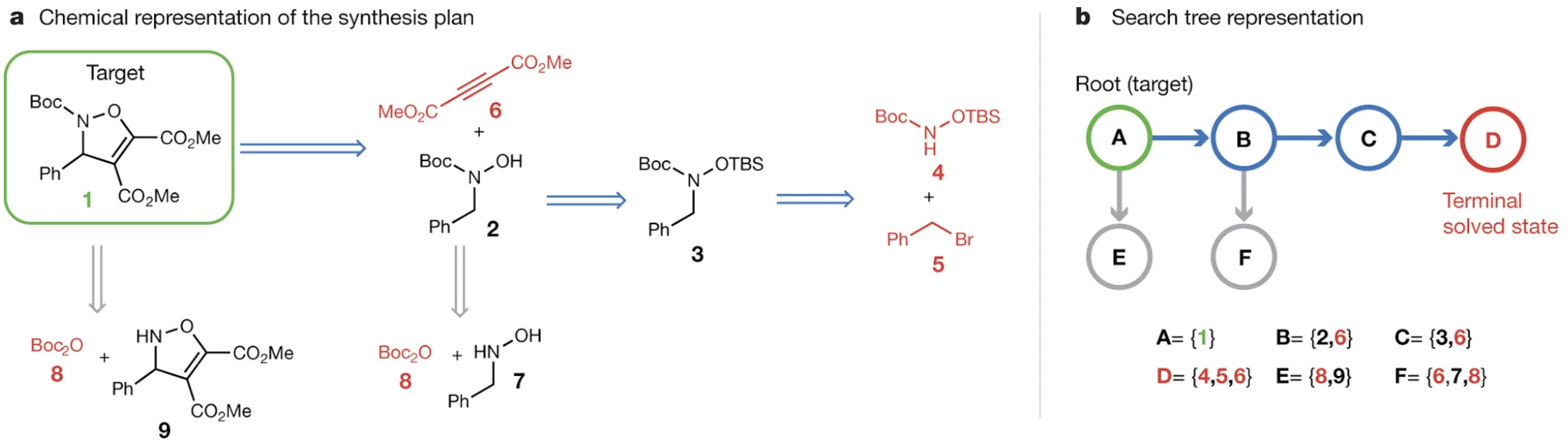


(a) Car merges *ahead* of human; anticipates human *braking*

(b) Car *backs up* at 4way stop; anticipates human *proceeding*

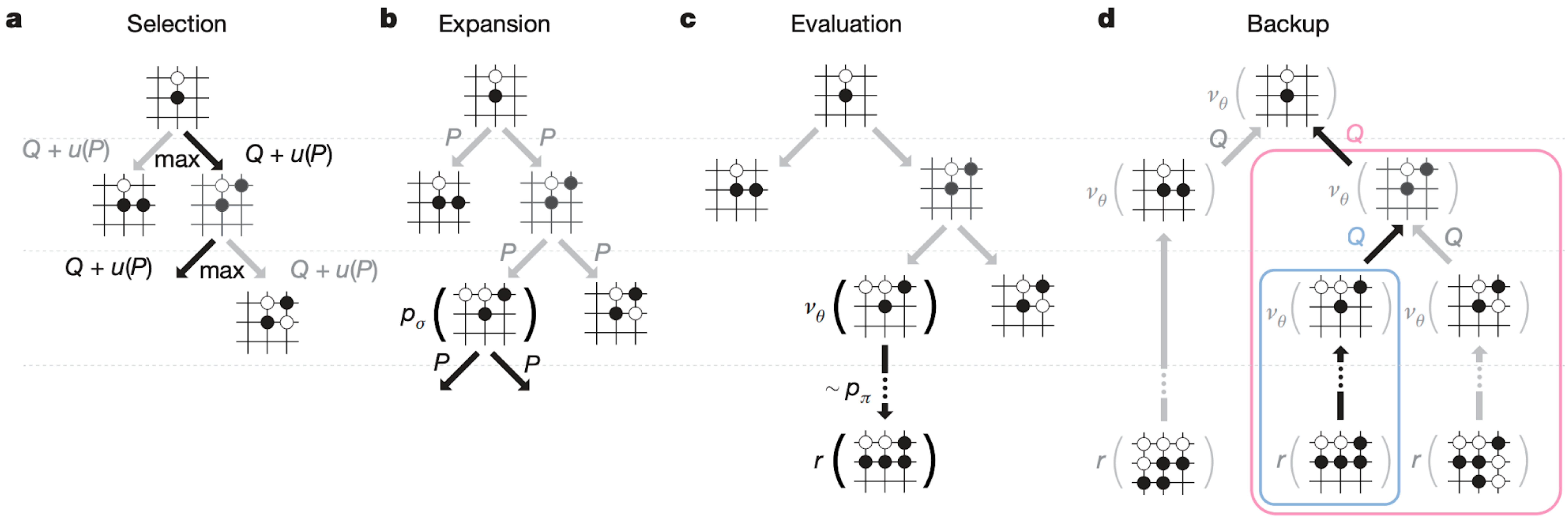
Sadigh et al. (2016). Planning for autonomous cars that leverage effects on human actions. RSS 2016.

Model-based reasoning for **science**



Segler, Preuss, & Waller (2018). Planning chemical syntheses with deep neural networks and symbolic AI. Nature, 555(7698).

Model-based reasoning for **games**



Silver et al. (2016). Mastering the game of Go with deep neural networks and tree search. Nature, 529(7587), 484.

Model, View, Controller

At each time step, our agent receives an **observation** from the environment.

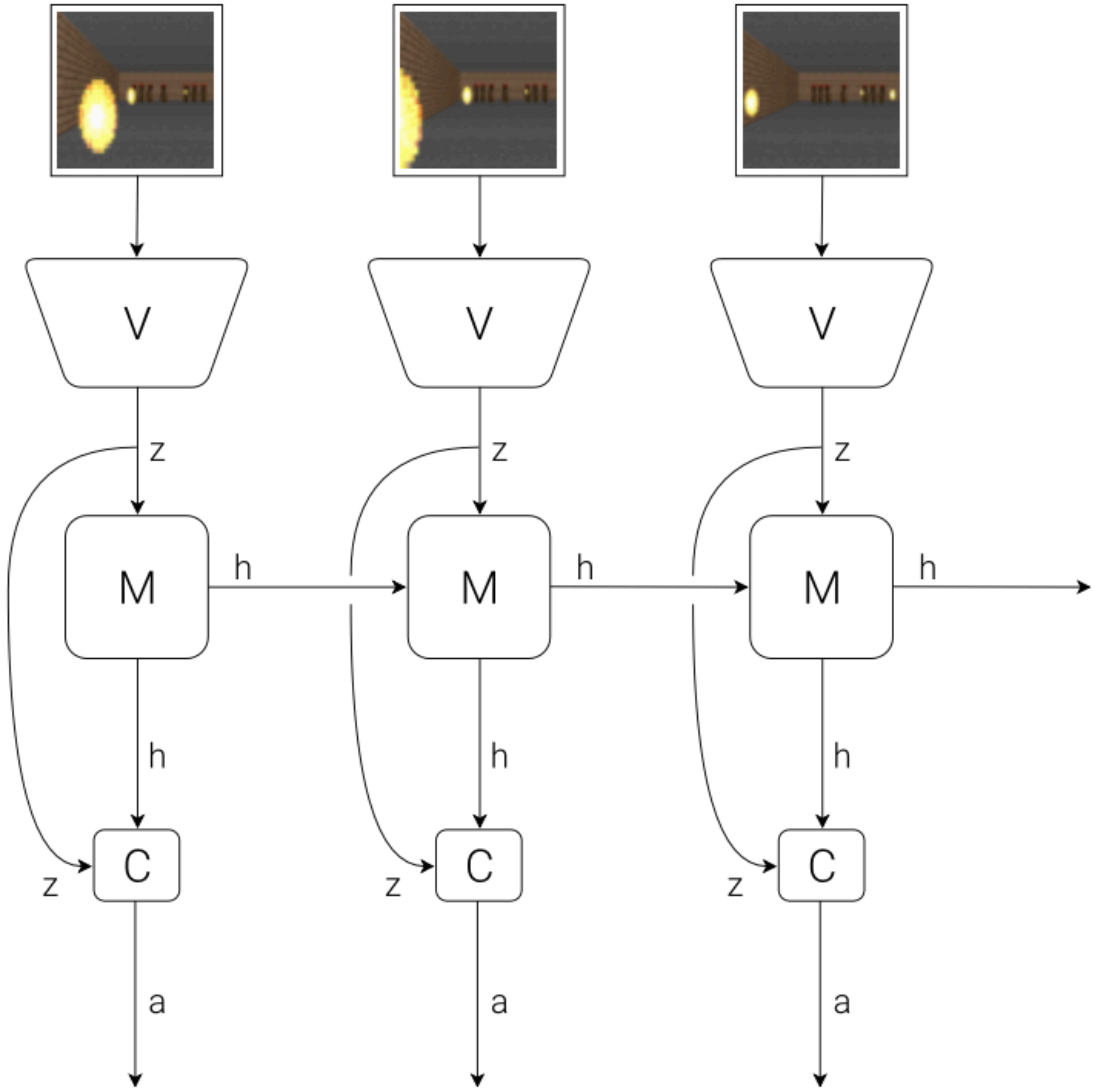
World Model

The **Vision Model (V)** encodes the high-dimensional observation into a low-dimensional latent vector.

The **Memory RNN (M)** integrates the historical codes to create a representation that can predict future states.

A small **Controller (C)** uses the representations from both **V** and **M** to select good actions.

The agent performs **actions** that go back and affect the environment.



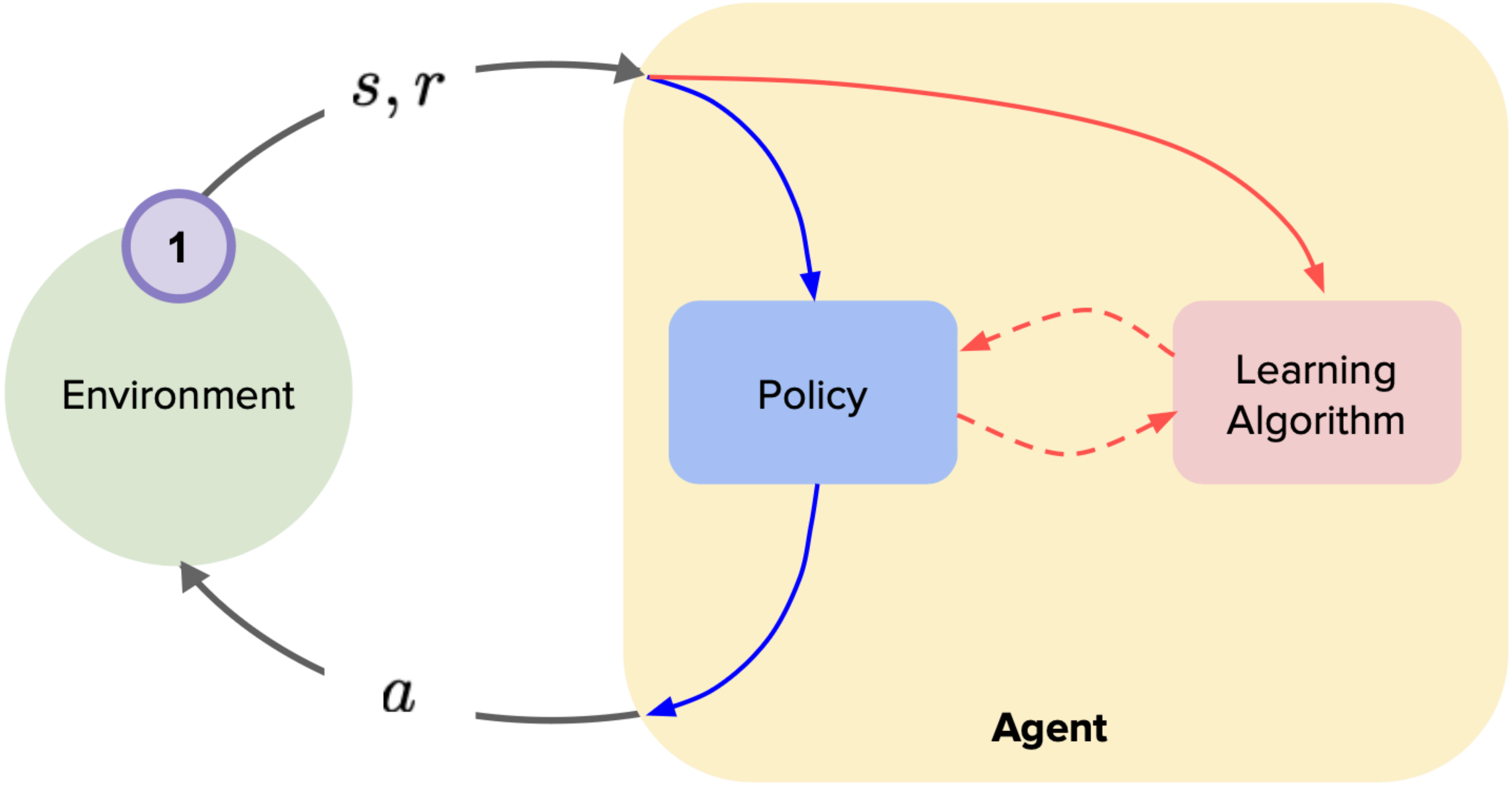
What is a model?

- Definition: a model is a representation that explicitly encodes knowledge about the structure of the environment and task.
  - A transition/dynamics model: $s_{t+1} = f_s(s_t, a_t)$
  - A model of rewards: $r_{t+1} = f_r(s_t, a_t)$
  - An inverse transition/dynamics model: $a_t = f_s^{-1}(s_t, s_{t+1})$
  - A model of distance: $d_{ij} = f_d(s_i, s_j)$
  - A model of future returns: $G_t = Q(s_t, a_t)$ or $G_t = V(s_t)$

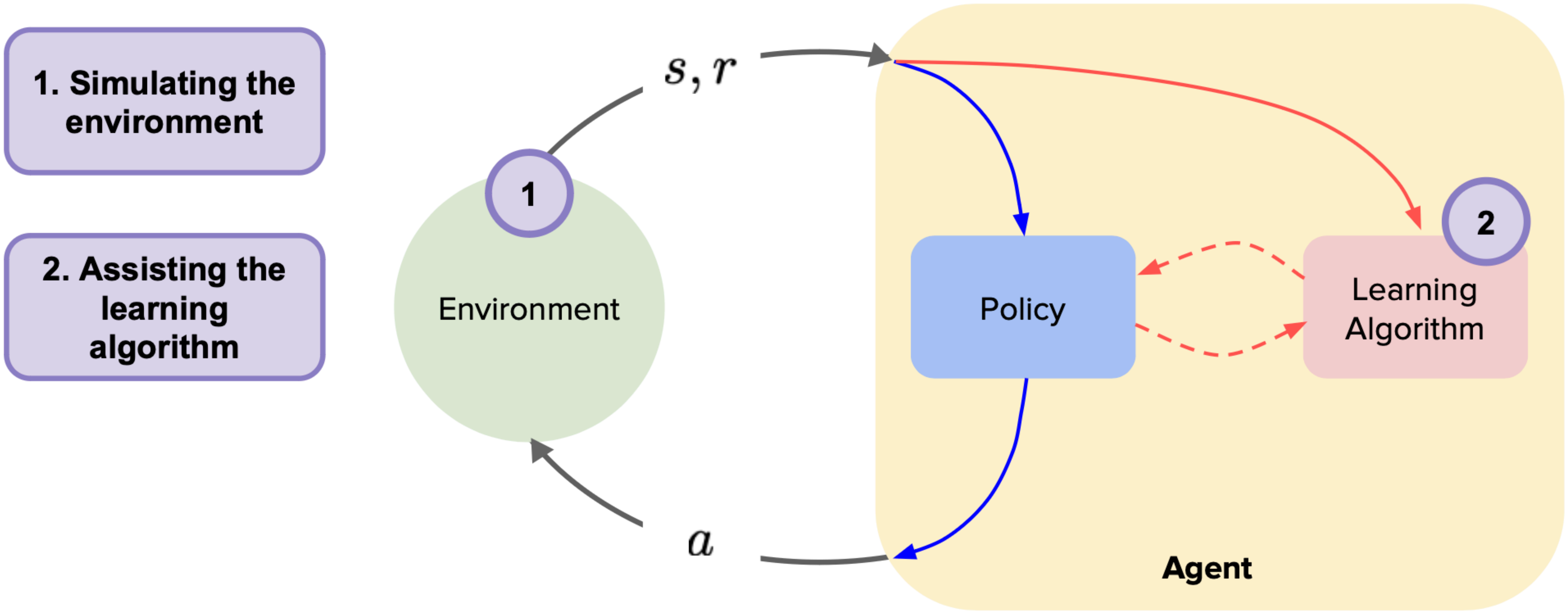
Typically what is meant by the model in model-based RL

Where does the model fit into the picture?

1. Simulating the environment



Where does the model fit into the picture?

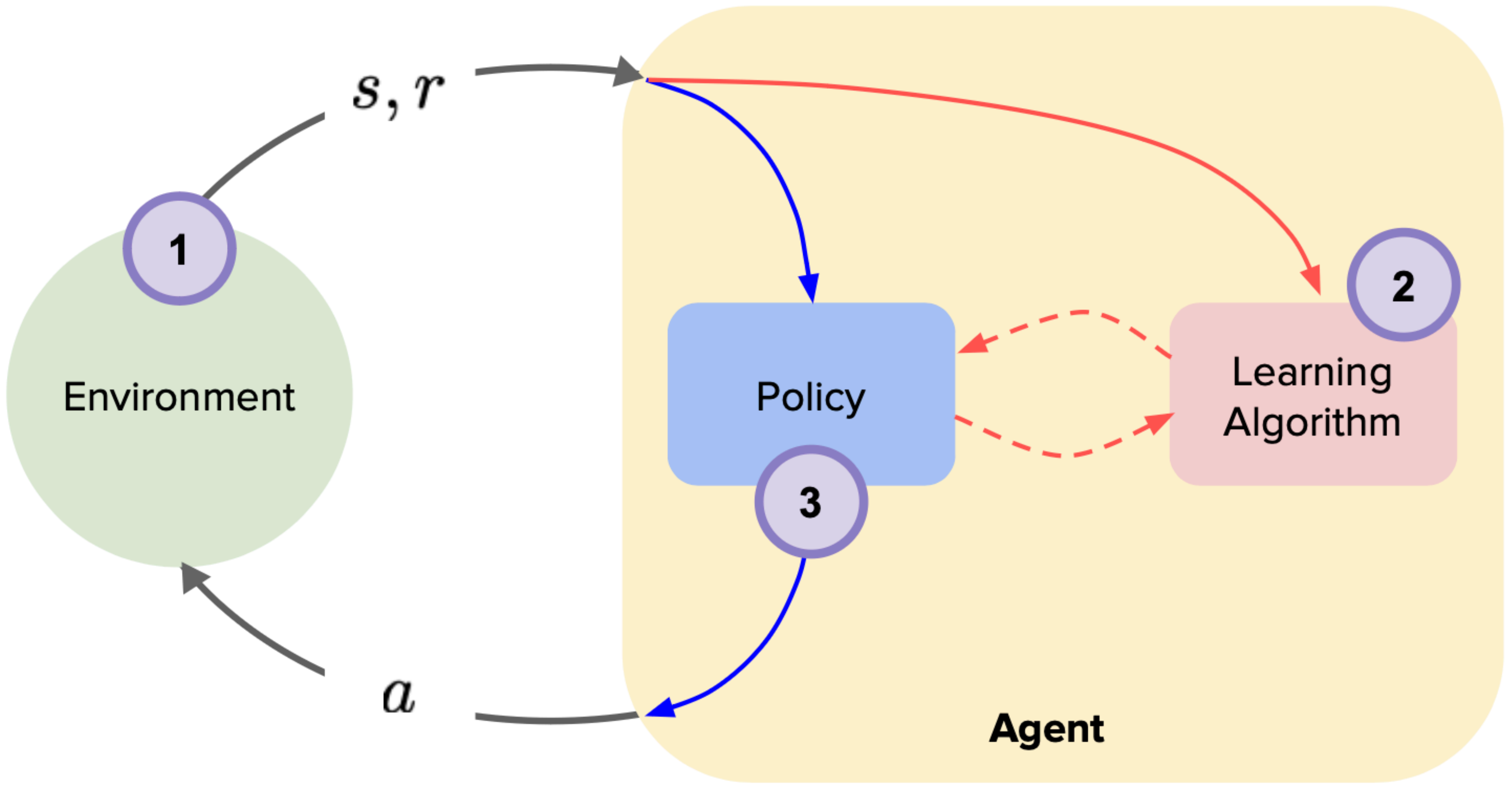


Where does the model fit into the picture?

1. Simulating the environment

2. Assisting the learning algorithm

3. Strengthening the policy



Why do we want to learn a model?

Planning with real robots
(too expensive, too risky)

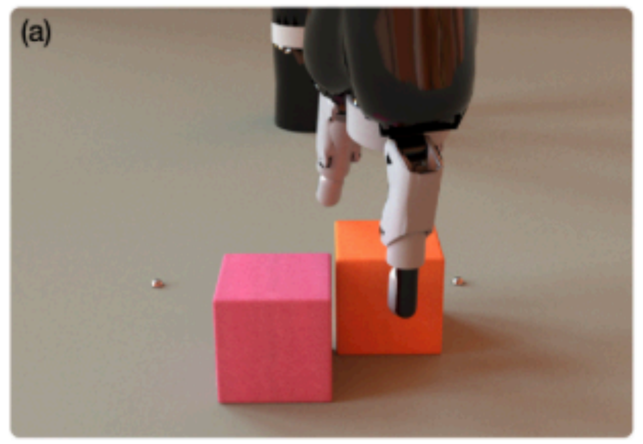
Simulating complex physical dynamics
(too expensive)

Interactions with humans
(no access)

V-View

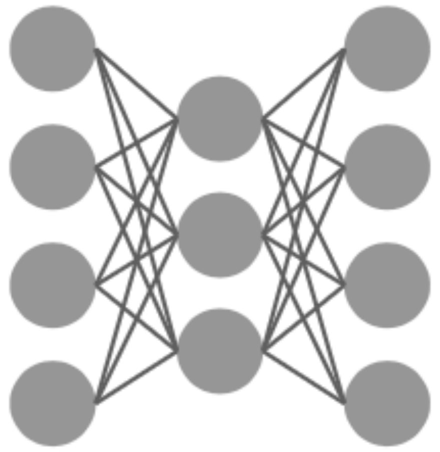
States vs. **Observations** vs. **Latent States**

$s = [x, \dot{x}, \phi]$

$o =$ 

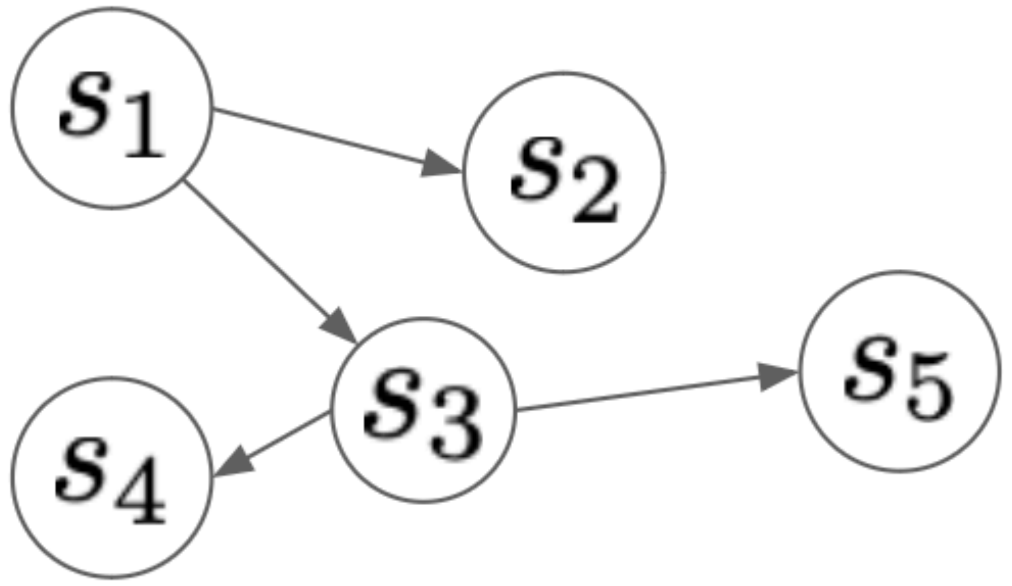
$z \in \mathbb{R}^{d_z}$

s_t
a_t



s_{t+1}

Parametric vs. **Non-Parametric**



Why View?

- Observation o_t can high dimensional
- Compresses each o_t it receives at time step t into a low dimensional latent vector z_t
- This compressed representation can be used to reconstruct the original $\hat{o}_t$

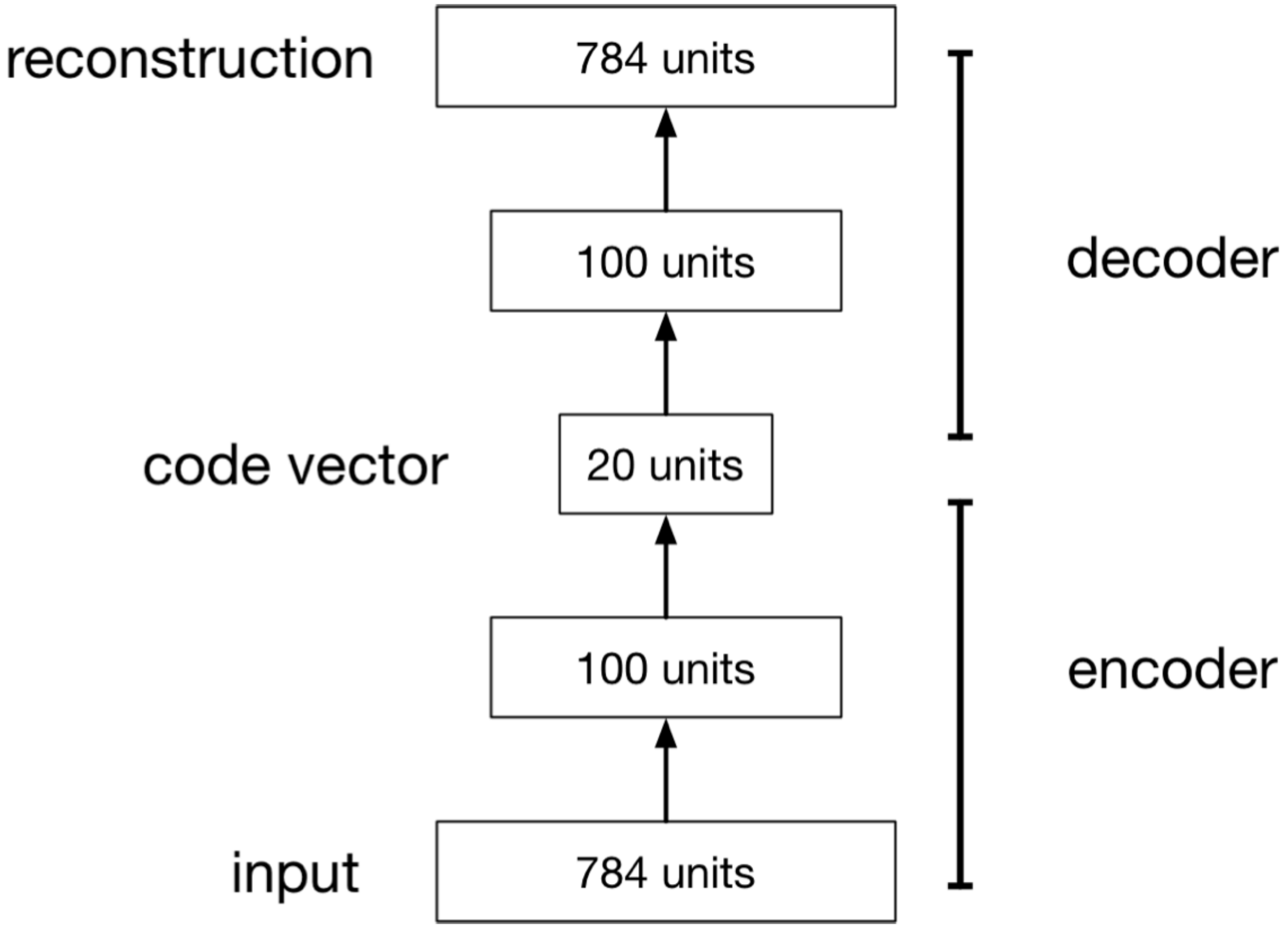
C-Controler

- The Controller (C) model is responsible for determining the course of actions to take in order to maximize the expected cumulative reward of the agent during a rollout of the environment.
- It uses V and M to rollout the environment
  - Like a dream!
- Can be (almost) any RL algorithm

Preliminaries

Autoencoders

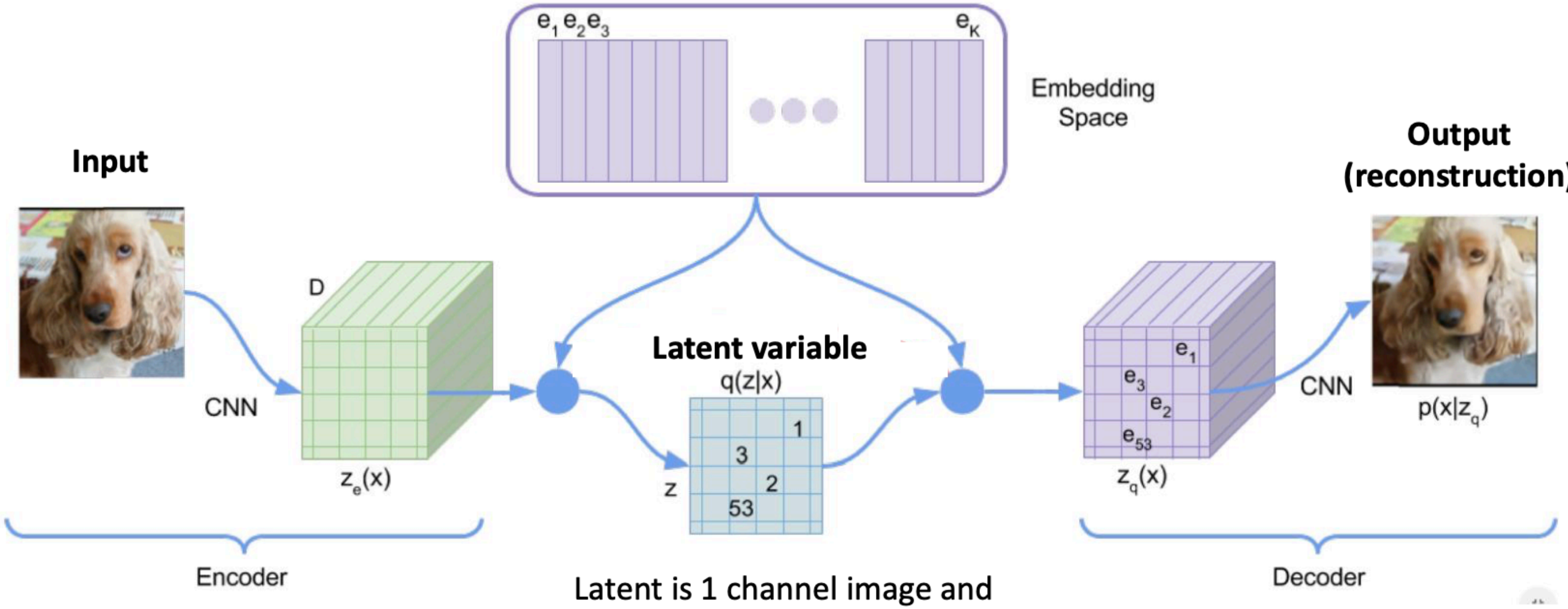
- An autoencoder is a feed-forward neural net whose job it is to take an input $\mathbf{x}$ and predict $\mathbf{x}$.
- To make this non-trivial, we need to add a bottleneck layer whose dimension is much smaller than the input.



Why Autoencoders?

- Map high-dimensional data to two dimensions for visualization
- Compression (i.e. reducing the file size)
  - Note: this requires a VAE, not just an ordinary autoencoder.
- Learn abstract features in an unsupervised way so you can apply them to a supervised task
  - Unlabled data can be much more plentiful than labeled data
- Learn a semantically meaningful representation where you can, e.g., interpolate between different images.

VQ-VAE

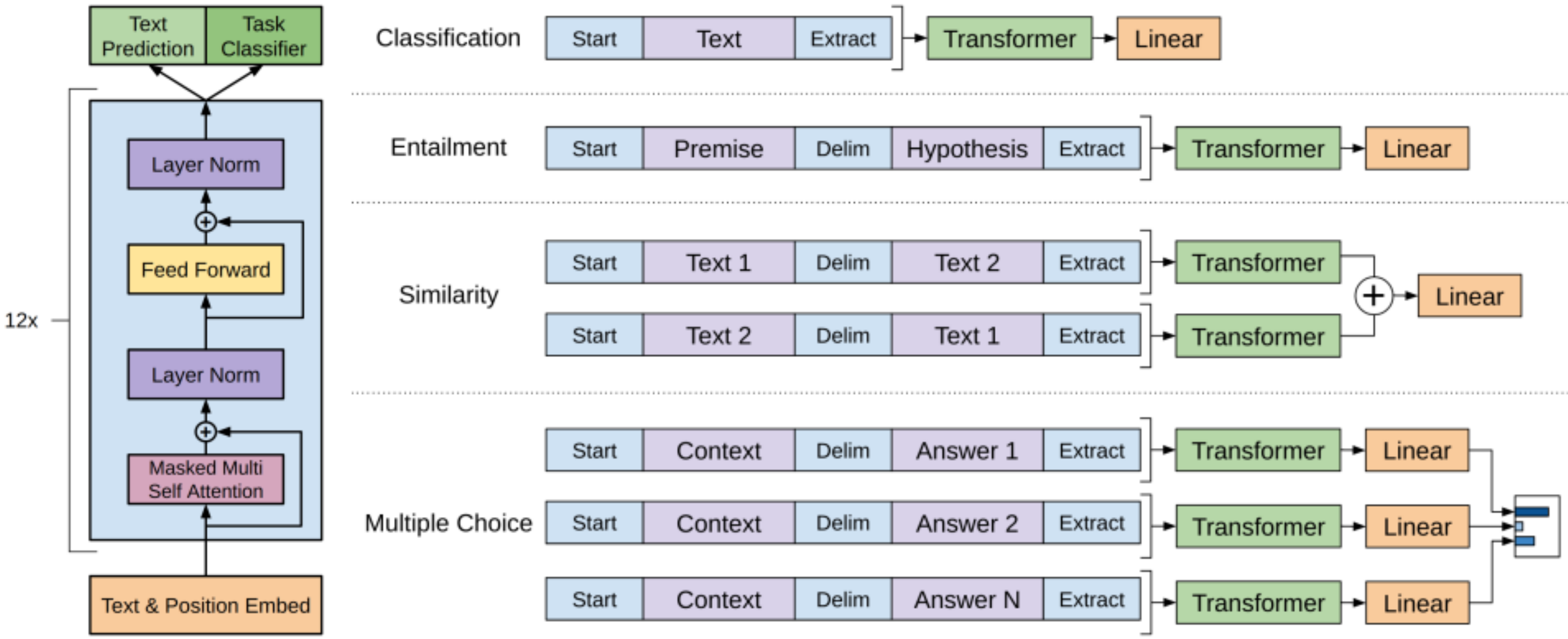


Latent is 1 channel image and contains the id of each e for each pixel (**discrete**).

Why are they useful?

- Provides low dimensional latent
  - Useful for V!
- Provides discrete representations
  - Useful for M
  - We'll see soon

GPT



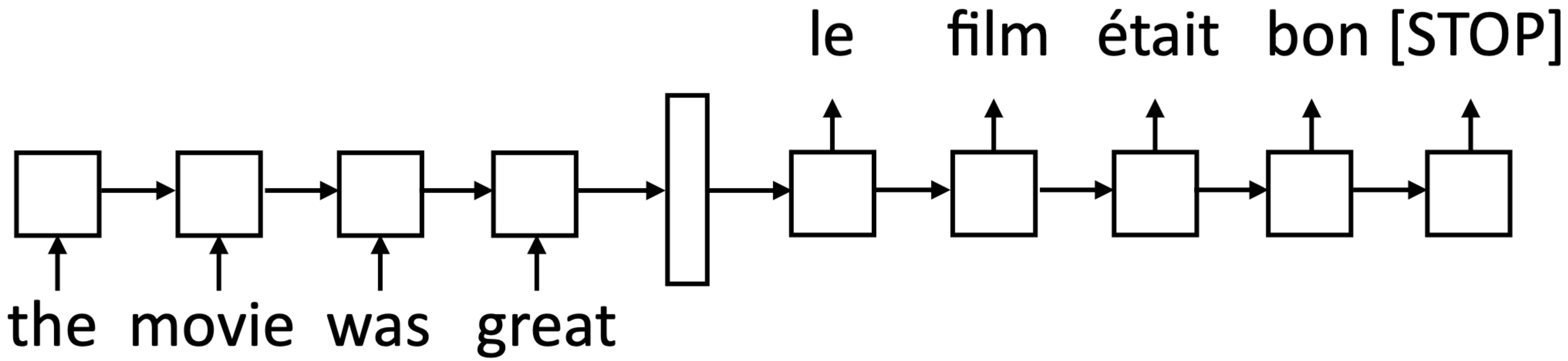
Text Prediction
Task Classifier
Layer Norm
Feed Forward
Layer Norm
Masked Multi Self Attention
12x
Text & Position Embed
Classification
Start
Text
Extract
Transformer
Linear
Entailment
Start
Premise
Delim
Hypothesis
Extract
Transformer
Linear
Similarity
Start
Text 1
Delim
Text 2
Extract
Transformer
Start
Text 2
Delim
Text 1
Extract
Transformer
Linear
Multiple Choice
Start
Context
Delim
Answer 1
Extract
Transformer
Linear
Start
Context
Delim
Answer 2
Extract
Transformer
Linear
Start
Context
Delim
Answer N
Extract
Transformer
Linear

GPT Training

- Assume a set of N tokens
- Given a T length sequence
  - Take a 0:t-1 seqeuence
  - Pass it through the model
  - Predict 1:t sequence through a softmax layer

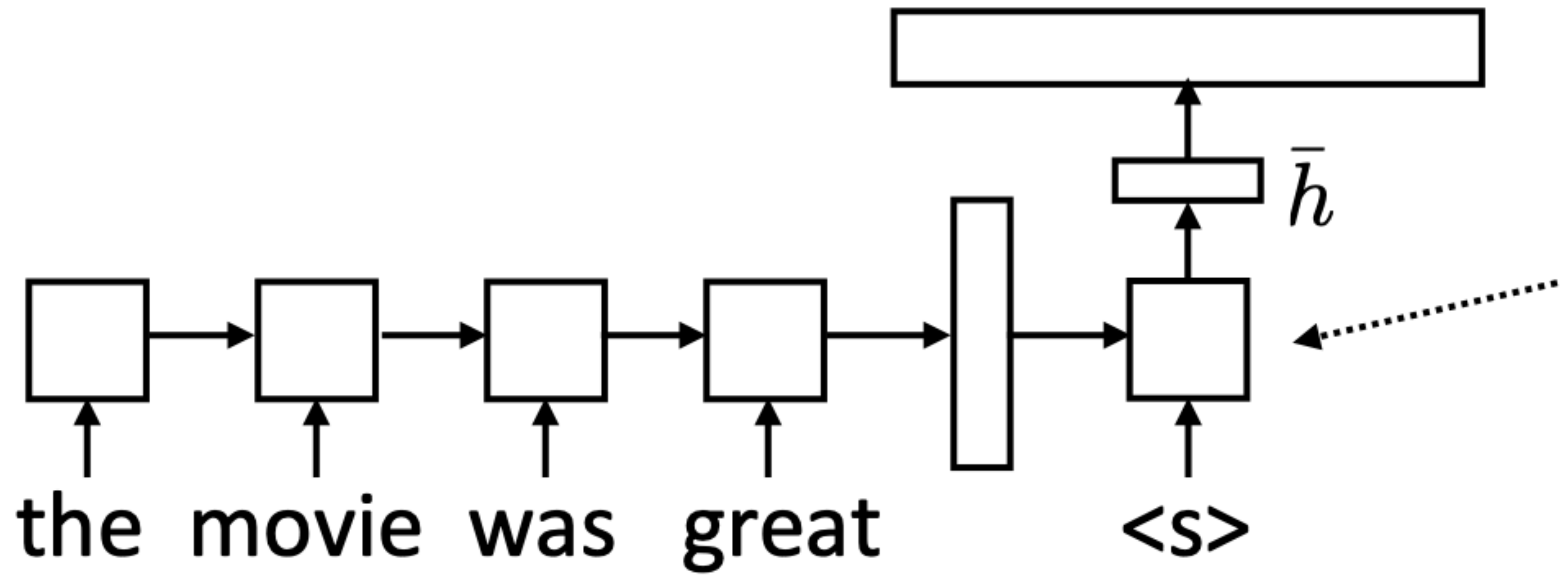
GPT From Encoder-Decoder View

Encode a sequence into a fixed-sized vector



GPT From Encoder-Decoder View

- Generate next word conditioned on previous word as well as hidden state
- W size is |vocab| x |hidden state|, softmax over entire vocabulary



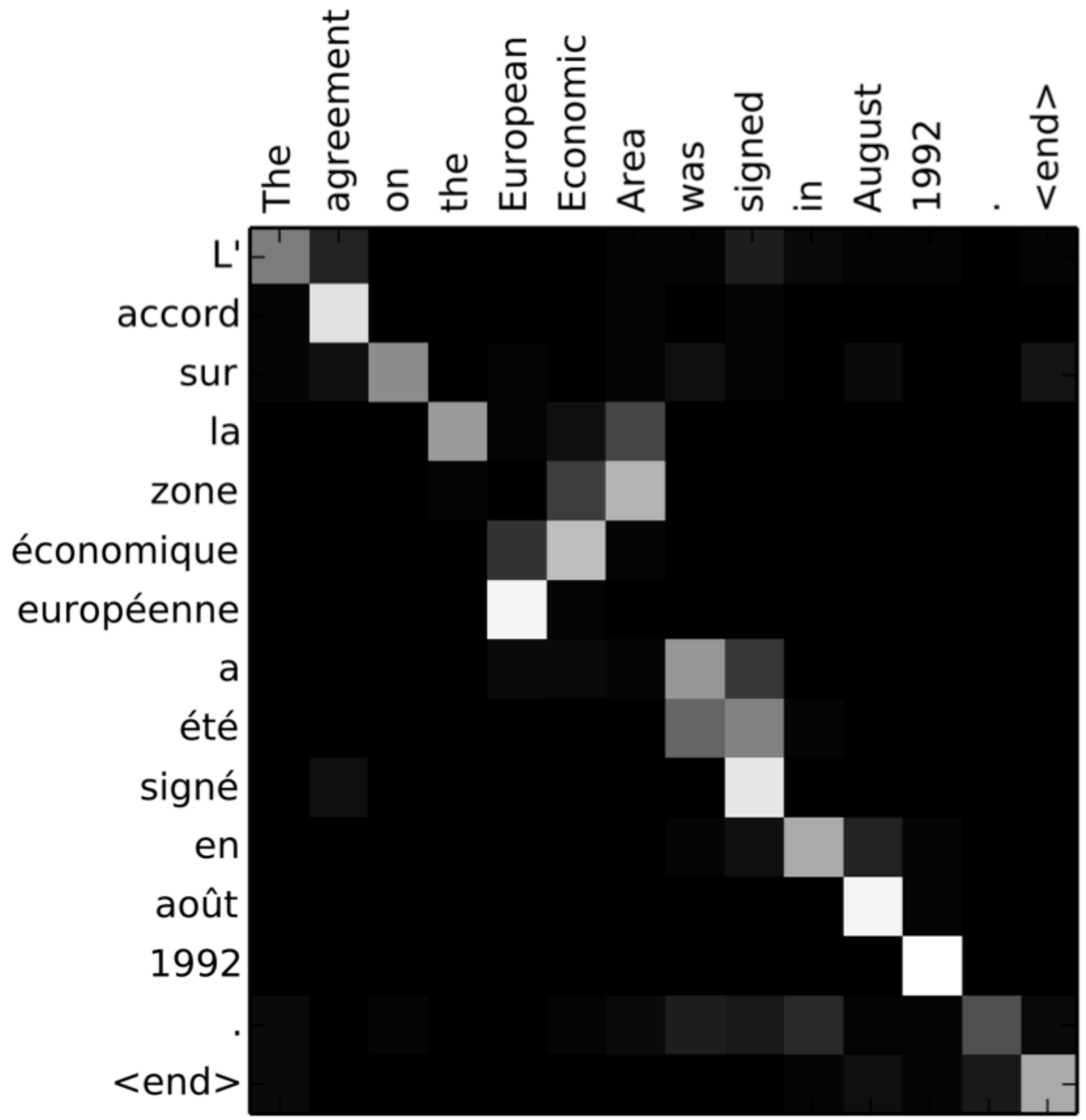
$$P(y_i|\mathbf{x}, y_1, \ldots, y_{i-1}) = \mathrm{softmax}(W\bar{h})$$

$$P(\mathbf{y}|\mathbf{x}) = \prod_{i=1}^{n} P(y_i|\mathbf{x}, y_1, \ldots, y_{i-1})$$

Decoder has separate parameters from encoder, so this can learn to be a language model (produce a plausible next word given current one)

Inference - Let's talk about attention

- Encoder hidden states capture contextual source word identity
- Decoder hidden states are now mostly responsible for selecting what to attend to
- Doesn't take a complex hidden state to walk monotonically through a sentence and spit out word-by-word translations

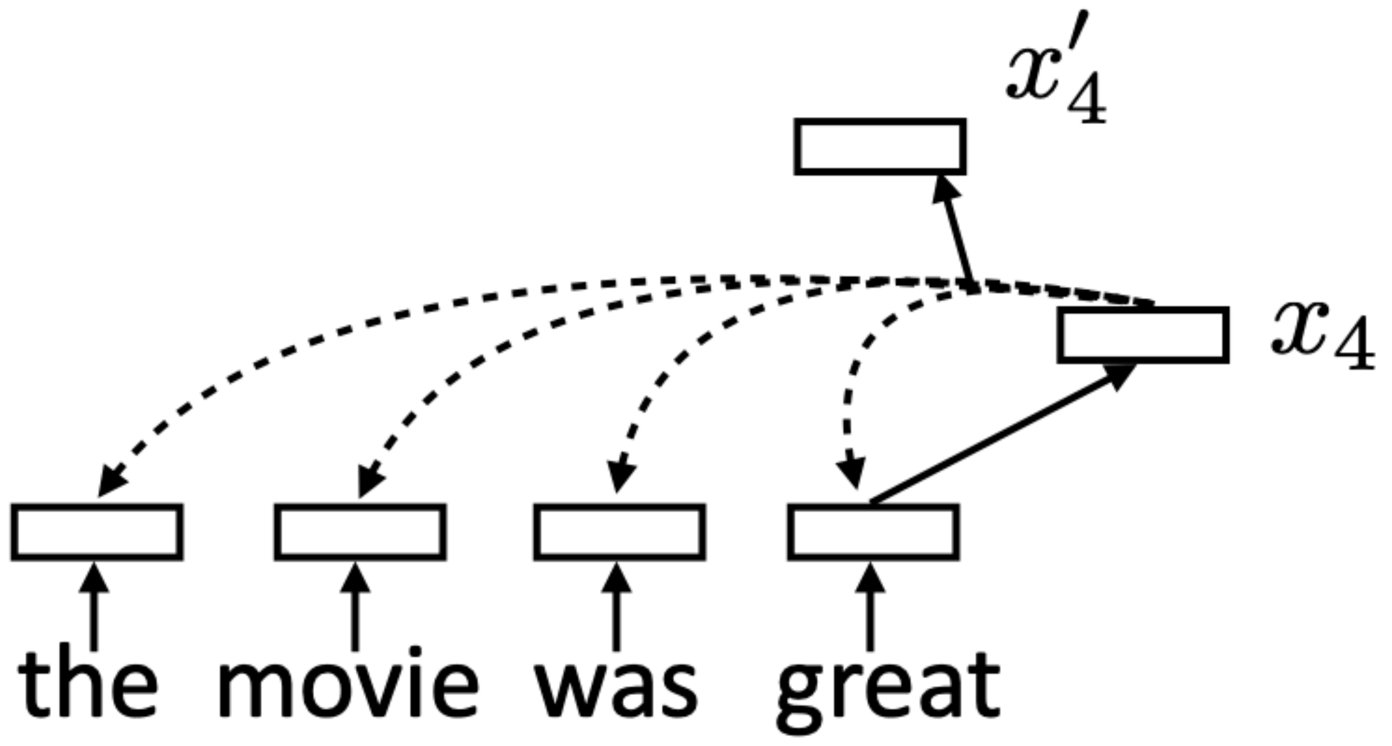


Self-attention

- Each word forms a "query" which then computes attention over each word

$$\alpha_{i,j} = \text{softmax}(x_i^\top x_j) \quad \text{scalar}$$

$$x'_i = \sum_{j=1}^{n} \alpha_{i,j} x_j \quad \text{vector = sum of scalar * vector}$$

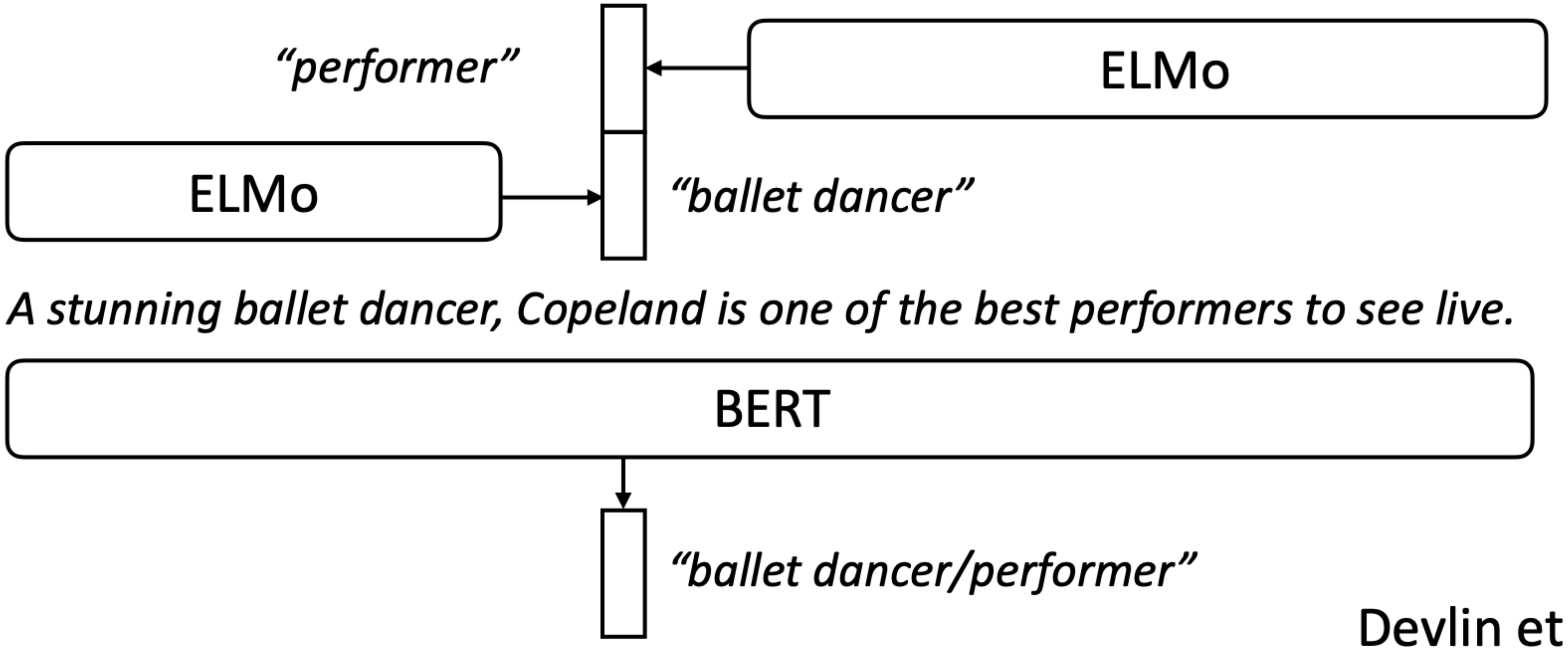


- Multiple "heads" analogous to different convolutional filters. Use parameters W_k and V_k to get different attention values + transform vectors

Vaswani et al. (2017)

Bidirectional Encoder Representations from Transformers(BERT)

- ELMo is a unidirectional model (as is GPT): we can concatenate two unidirectional models, but is this the right thing to do?
- ELMo reprs look at each direction in isolation; BERT looks at them jointly

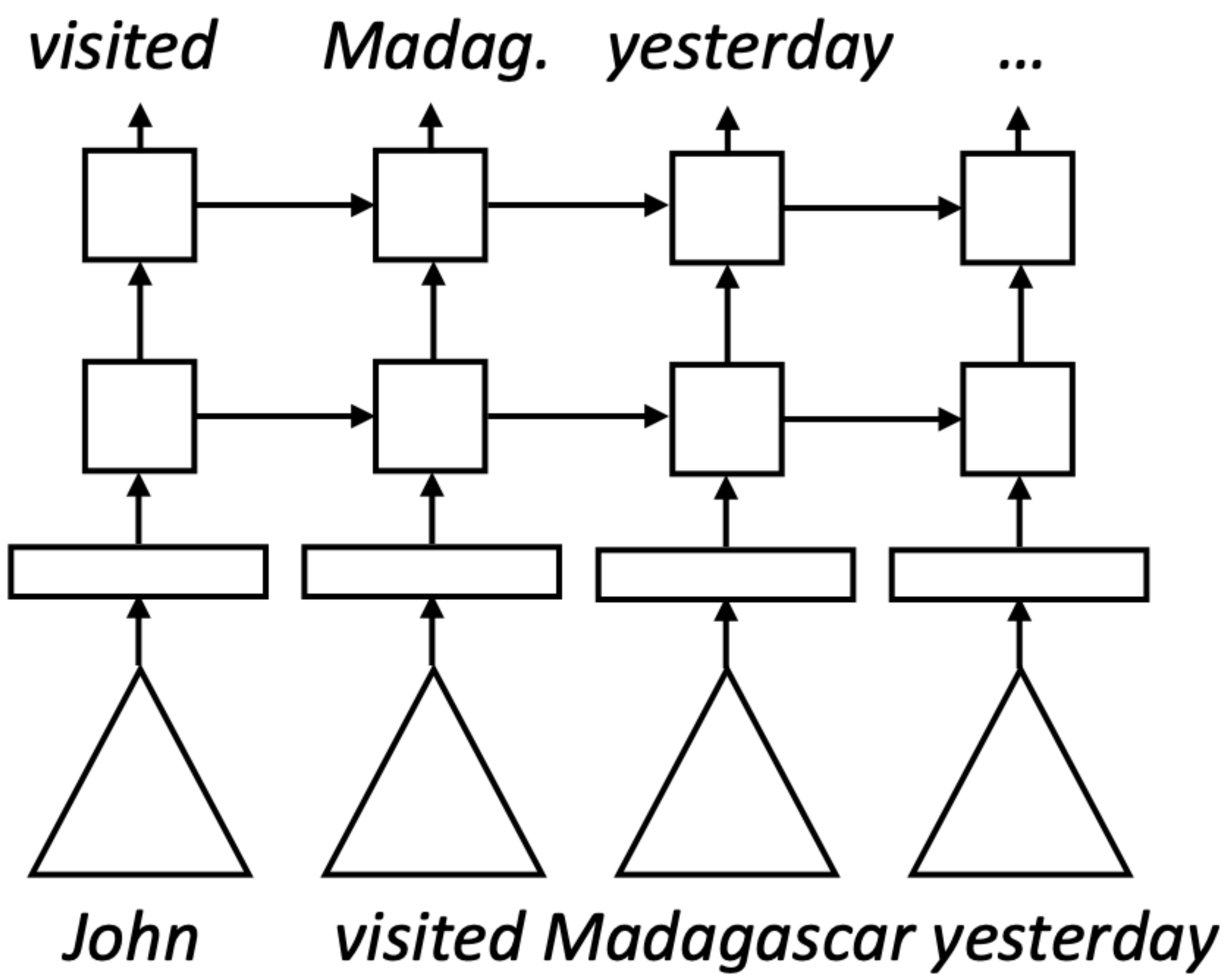


Devlin et al. (2019)

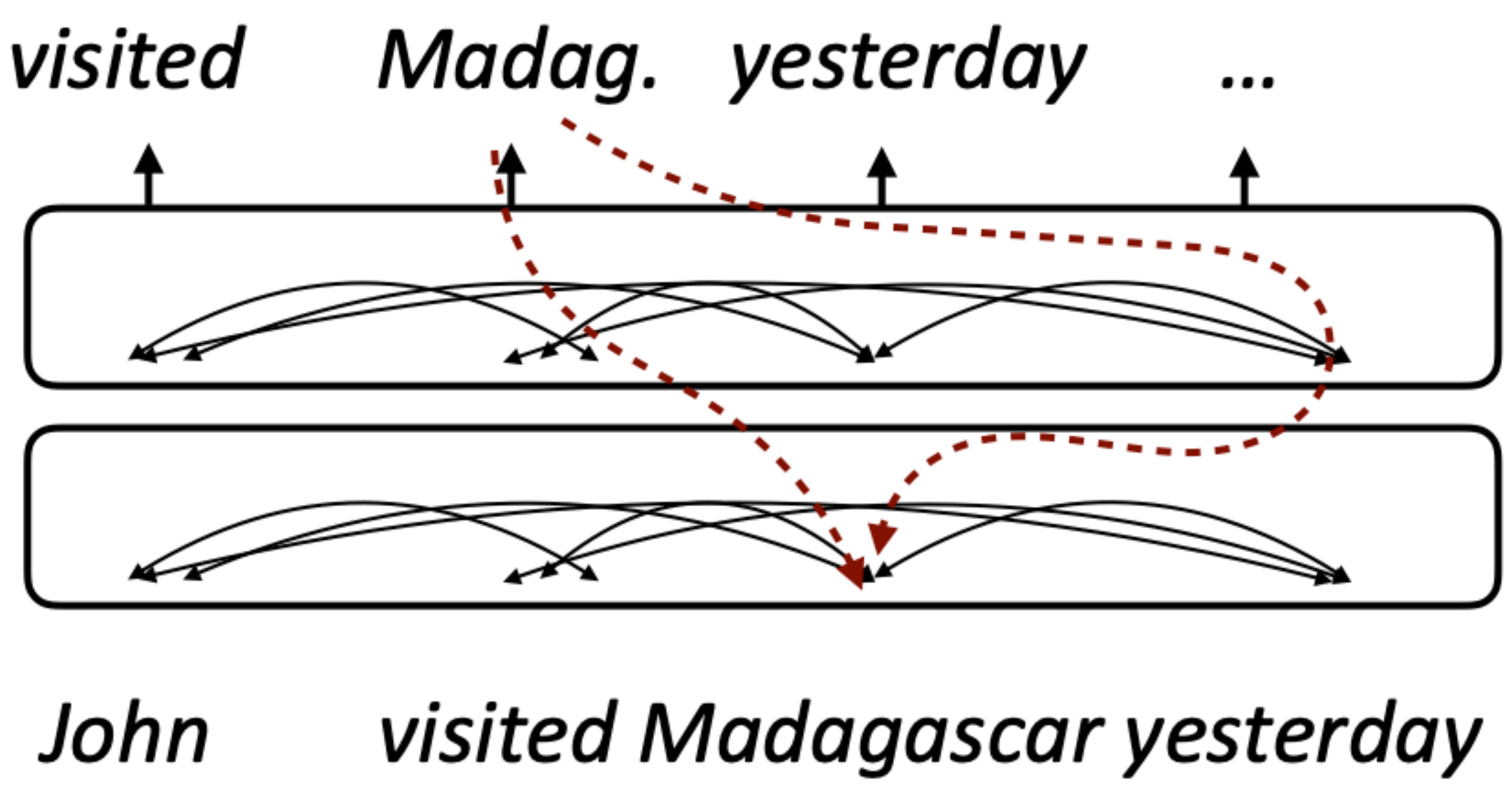
Bidirectional Encoder Representations from Transformers(BERT)

- How to learn a "deeply bidirectional" model? What happens if we just replace an LSTM with a transformer?

ELMo (Language Modeling)



BERT

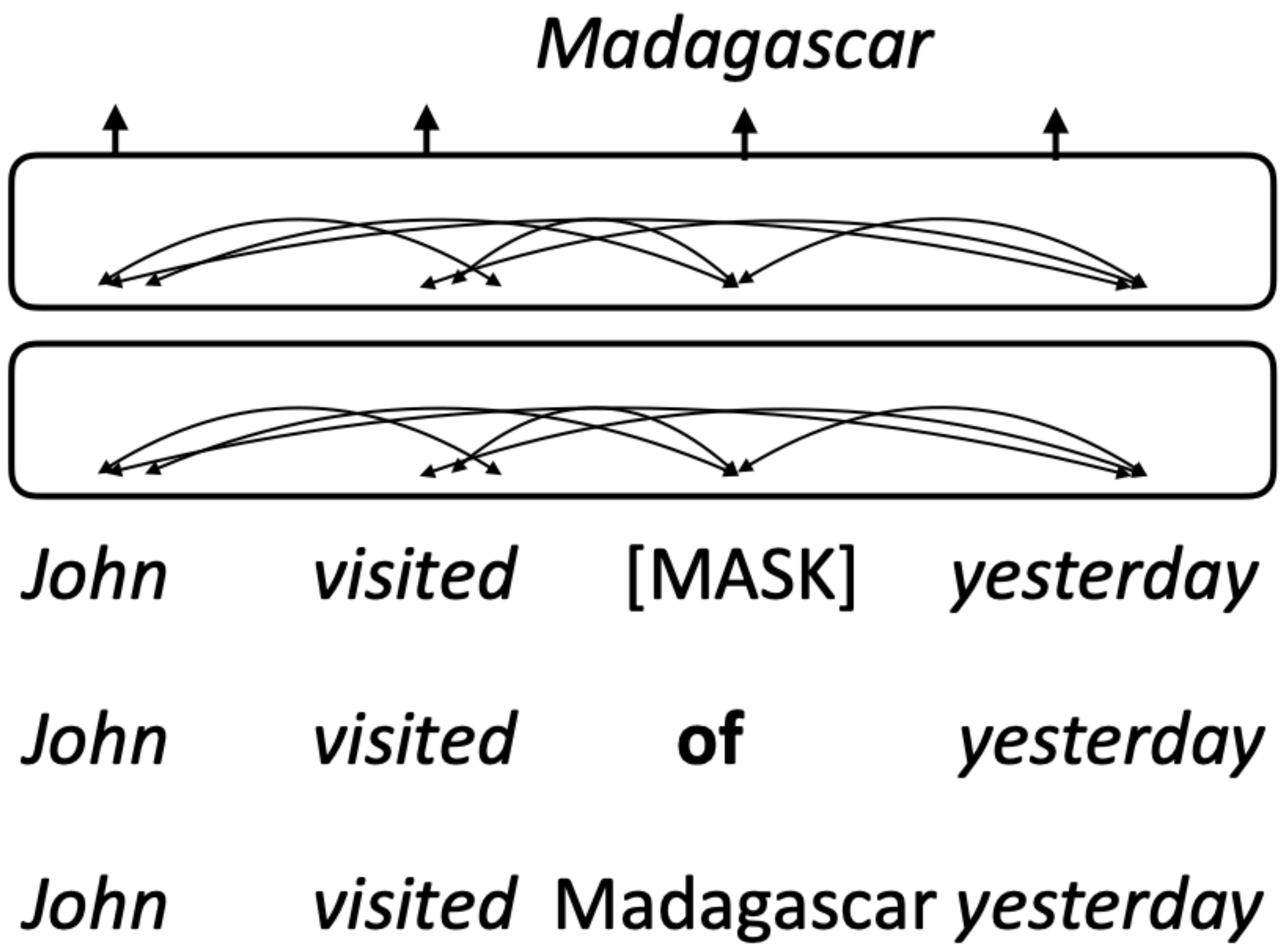


- Transformer LMs have to be "one-sided" (only attend to previous tokens), not what we want

Masked Sequence Modelling

- How to prevent cheating? Next word prediction fundamentally doesn't work for bidirectional models, instead do *masked language modeling*

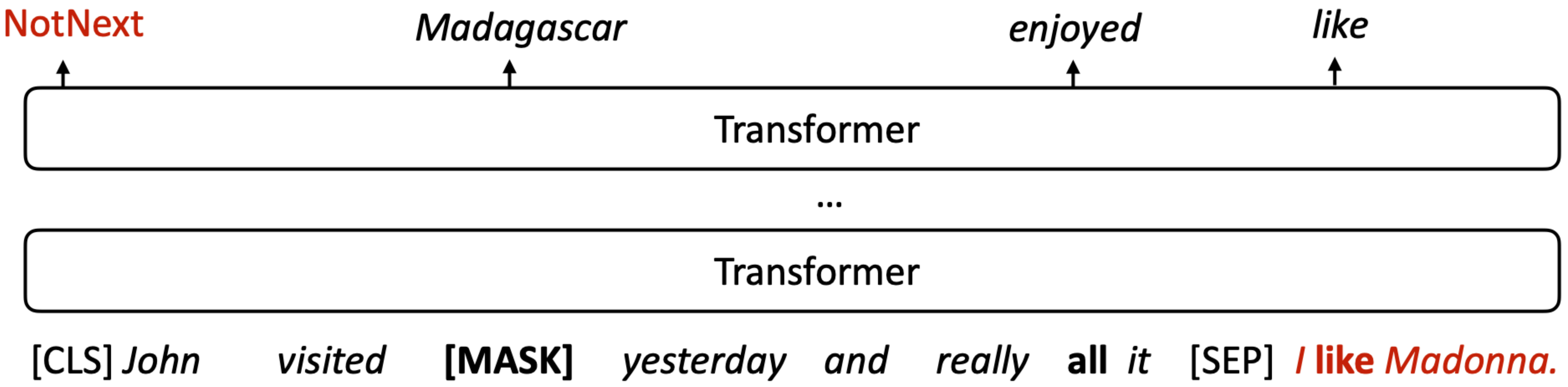
- BERT formula: take a chunk of text, predict 15% of the tokens
  - For 80% (of the 15%), replace the input token with [MASK]
  - For 10%, replace w/random
  - For 10%, keep same



Devlin et al. (2019)

Masked Sequence Modelling

- Input: [CLS] Text chunk 1 [SEP] Text chunk 2
- 50% of the time, take the true next chunk of text, 50% of the time take a random other chunk. Predict whether the next chunk is the “true” next
- BERT objective: masked LM + next sentence prediction



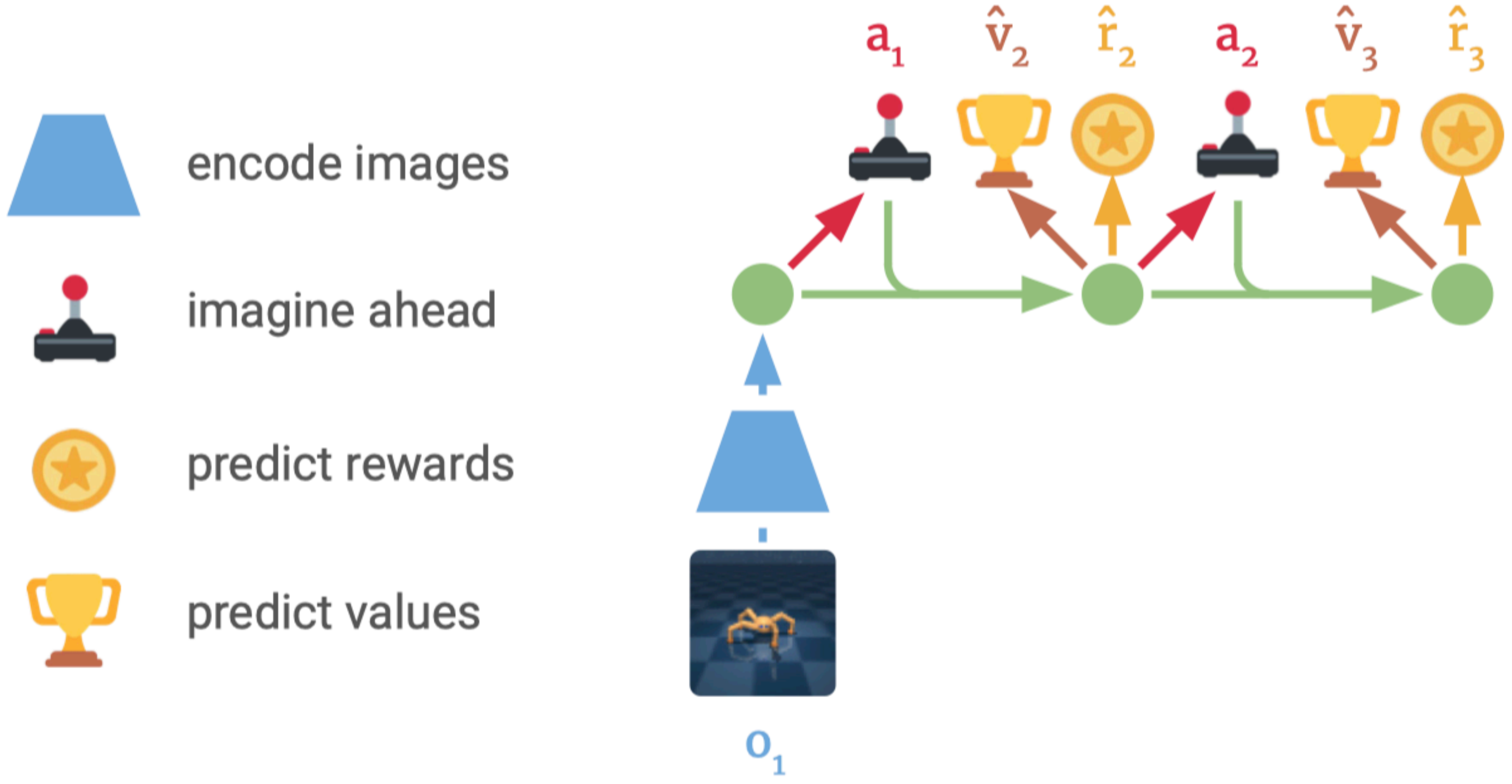
Devlin et al. (2019)

Why are we learning this?

- Different ways of sequence modeling
  - Important for model - M
- Different ways have different drawbacks
  - Engineering decisions!

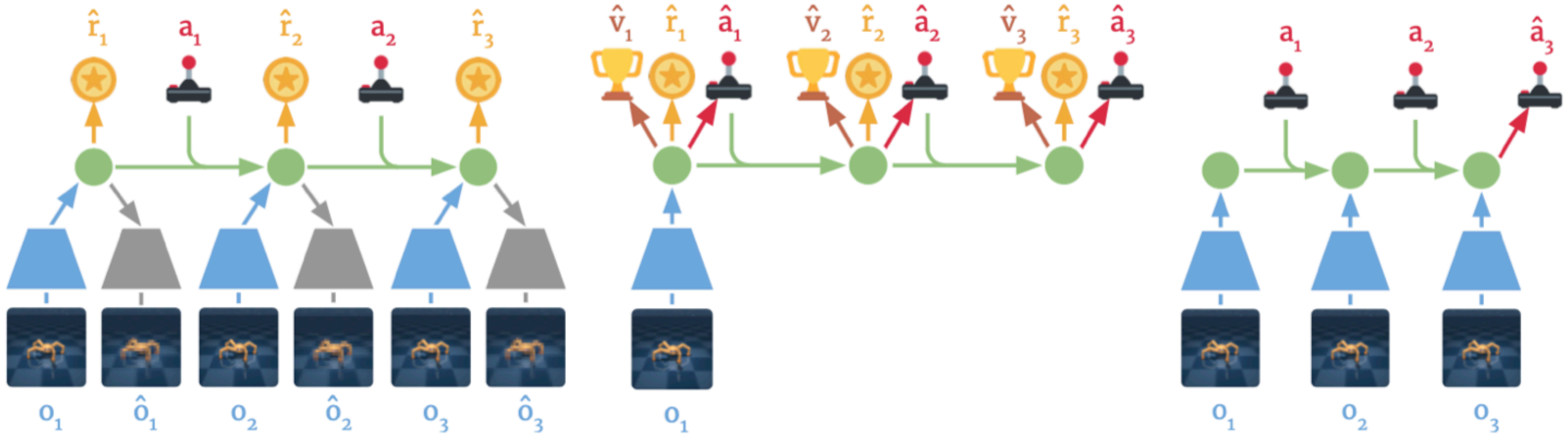
How to train your world model?

Dreamer



encode images
imagine ahead
predict rewards
predict values
a_1
$\hat{v}_2$
$\hat{r}_2$
a_2
$\hat{v}_3$
$\hat{r}_3$
o_1

Dreamer



1. Learning the dynamics model   2. Learning policy   3. Collect experience

Dreamer

Initialize policy θ, ϕ, ψ randomly and D_{env} with random trajectories $\{(o_t^{(i)}, a_t^{(i)}, r_t^{(i)})_{t=1}^{T}\}$.

1. **Dynamics learning:**
    1. Sample trajectories from D_{env} and infer states from observations using the representation model: $s_t \sim p_\theta(s_t \mid s_{t-1}, a_{t-1}, o_t)$.
    2. Train the dynamics model using variational inference and update θ.
2. **Actor-critic learning from imagined rollouts:**
    1. Imagine trajectories seeded from s_t : $\{s_\tau, r_\tau, a_\tau, v_\tau\}_{\tau=t}^{t+H}$
    2. Compute value targets $V_\lambda(s_\tau)$.
    3. Update actor $\phi \rightarrow \phi + \alpha \nabla_\phi \sum_\tau^H V_\lambda(s_\tau)$
    4. Update critic: $\psi \rightarrow \psi - \alpha \nabla_\psi \sum_\tau^H \|v_\psi - V_\lambda(s_\tau)\|$
3. **Environment interaction**
    1. Deploy the actor in the environment adding exploration noise to the predicted actions and update D_{env}

Dreamer

Initialize policy θ, ϕ, ψ randomly and D_{env} with random trajectories $\{(o_t^{(i)}, a_t^{(i)}, r_t^{(i)})_{t=1}^{T}\}$.

1. **Dynamics learning:**
    1. Sample trajectories from D_{env} and infer states from observations using the representation model: $s_t \sim p_\theta(s_t \mid s_{t-1}, a_{t-1}, o_t)$.
    2. Train the dynamics model using variational inference and update θ.
2. **Actor-critic learning from imagined rollouts:**
    1. Imagine trajectories seeded from s_t : $\{s_\tau, r_\tau, a_\tau, v_\tau\}_{\tau=t}^{t+H}$
    2. Compute value targets $V_\lambda(s_\tau)$.
    3. Update actor $\phi \rightarrow \phi + \alpha \nabla_\phi \sum_\tau^H V_\lambda(s_\tau)$
    4. Update critic: $\psi \rightarrow \psi - \alpha \nabla_\psi \sum_\tau^H \| v_\psi - V_\lambda(s_\tau) \|$
3. **Environment interaction**
    1. Deploy the actor in the environment adding exploration noise to the predicted actions and update D_{env}

Dreamer

Algorithm 1: Dreamer

Initialize dataset $\mathcal{D}$ with S random seed episodes.
Initialize neural network parameters θ, ϕ, ψ randomly.
while *not converged* **do**
  for *update step* $c = 1..C$ **do**
    `// Dynamics learning`
    Draw B data sequences $\{(a_t, o_t, r_t)\}_{t=k}^{k+L} \sim \mathcal{D}$.
    Compute model states $s_t \sim p_\theta(s_t \mid s_{t-1}, a_{t-1}, o_t)$.
    Update θ using representation learning.
    `// Behavior learning`
    Imagine trajectories $\{(s_\tau, a_\tau)\}_{\tau=t}^{t+H}$ from each s_t.
    Predict rewards $\mathrm{E}\big(q_\theta(r_\tau \mid s_\tau)\big)$ and values $v_\psi(s_\tau)$.
    Compute value estimates $\mathrm{V}_\lambda(s_\tau)$ via Equation 6.
    Update $\phi \leftarrow \phi + \alpha \nabla_\phi \sum_{\tau=t}^{t+H} \mathrm{V}_\lambda(s_\tau)$.
    Update $\psi \leftarrow \psi - \alpha \nabla_\psi \sum_{\tau=t}^{t+H} \frac{1}{2}\big\| v_\psi(s_\tau) - \mathrm{V}_\lambda(s_\tau) \big\|^2$.
  `// Environment interaction`
  $o_1 \leftarrow$ `env.reset()`
  for *time step* $t = 1..T$ **do**
    Compute $s_t \sim p_\theta(s_t \mid s_{t-1}, a_{t-1}, o_t)$ from history.
    Compute $a_t \sim q_\phi(a_t \mid s_t)$ with the action model.
    Add exploration noise to action.
    $r_t, o_{t+1} \leftarrow$ `env.step(`a_t`)`.
  Add experience to dataset $\mathcal{D} \leftarrow \mathcal{D} \cup \{(o_t, a_t, r_t)_{t=1}^{T}\}$.

Model components

Representation	$p_\theta(s_t \mid s_{t-1}, a_{t-1}, o_t)$
Transition	$q_\theta(s_t \mid s_{t-1}, a_{t-1})$
Reward	$q_\theta(r_t \mid s_t)$
Action	$q_\phi(a_t \mid s_t)$
Value	$v_\psi(s_t)$

Hyper parameters

Seed episodes	S
Collect interval	C
Batch size	B
Sequence length	L
Imagination horizon	H
Learning rate	α

Dreamer - What's there?

- M - latent state space model
- V - latent state space model
- C - actor critic model

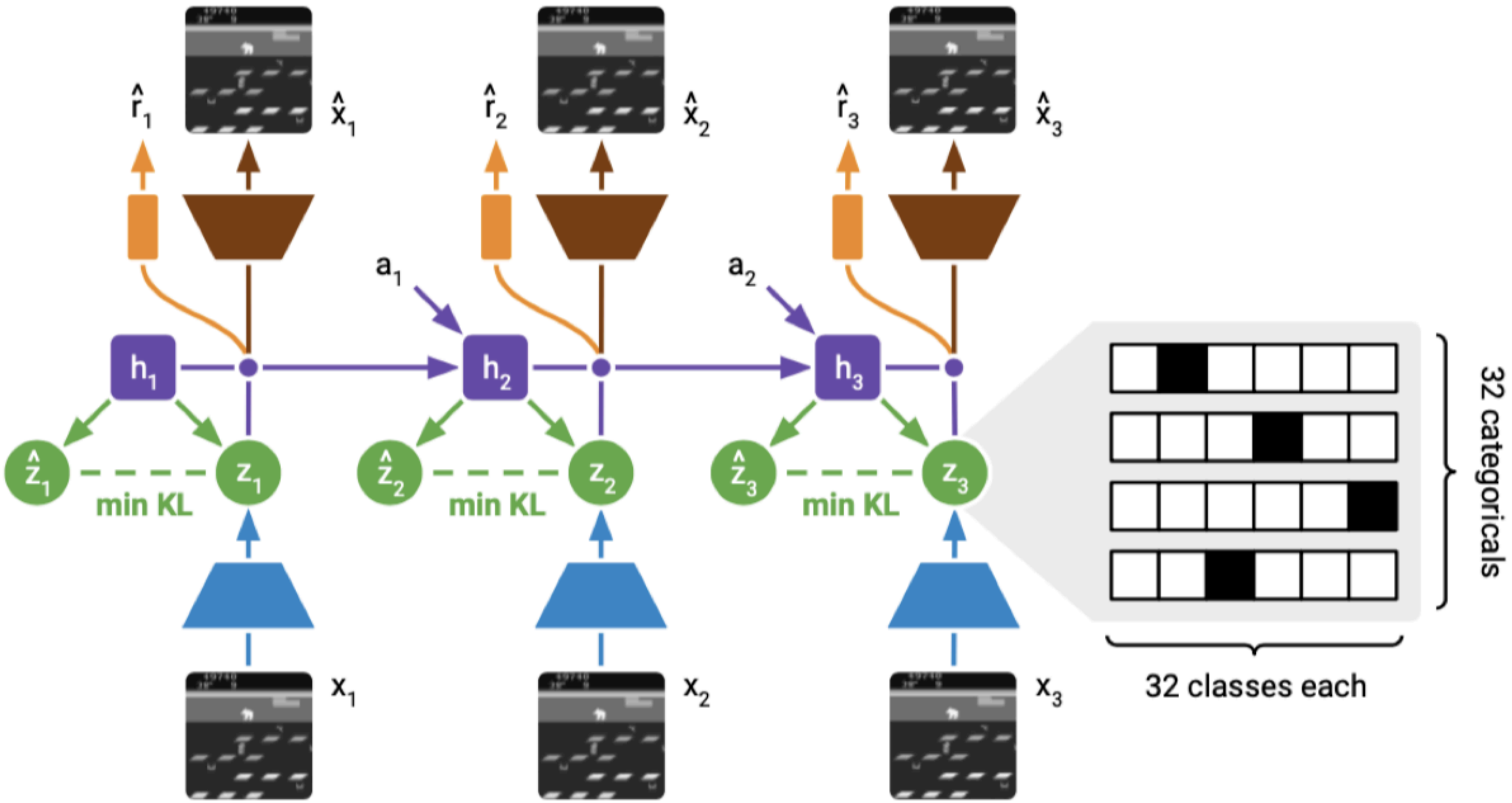
Dreamer - What's missing?

- No discrete representation
  - We saw discrete representation in VQ-VAE!

Dreamer V2

- Adds stochasticity through discrete latents
- Two representation for states s_t
  - Deterministic part (from previous approaches)
  - Stochastic part
    - 32 vectors from 32 values
- Combination of reinforce and straight-through gradients
- KL balancing

Dreamer V2



r̂1
x̂1
r̂2
x̂2
r̂3
x̂3
a1
a2
h1
h2
h3
ẑ1
z1
ẑ2
z2
ẑ3
z3
min KL
min KL
min KL
x1
x2
x3
32 categoricals
32 classes each

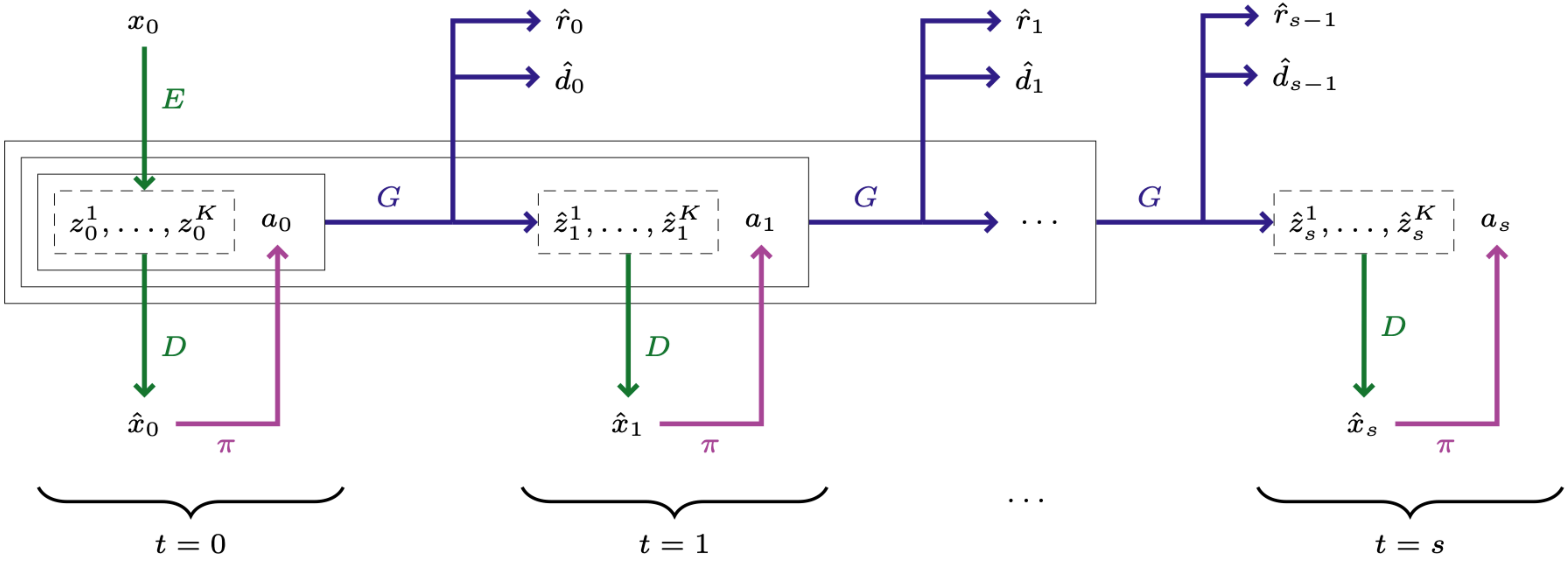
IRIS

- We saw discrete representations
- Can we use GPT? 🤔

Transformers are sample efficient world models

- Encode observations in a discrete space
- Use 0,..T-1 discrete tokens and actions to generate token at T, and predict reward and termination
- Using the WM, learn a policy on the decoded observation

Transformers are sample efficient world models



IRIS - What's there?

- M - GPT2
- V - VAE
- C - Actor critic model

IRIS - Issues

- Slow Imagine step
  - Imagine step has N * H steps where
  - Next token prediction for N tokens (1 obs / state = N tokens)
  - Next state prediction for H states
- Does not retain any state information

Faster imagination - two approaches

- Less number of tokens for imagination
- Non-autoregressive generation

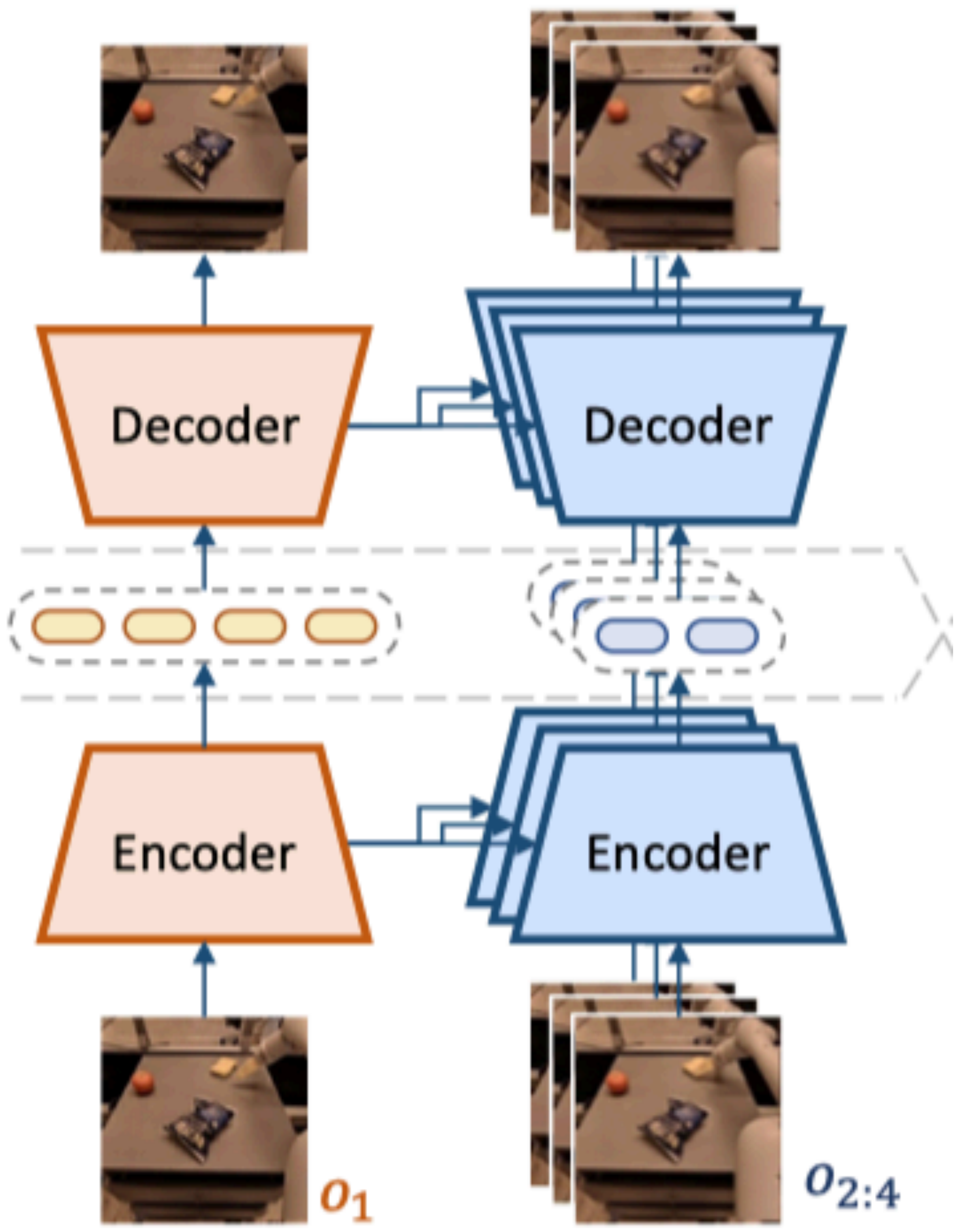
iVideoGPT | View (V)

- Context encoder and decoder
  - N = 16 tokens for $1 : T_0$ observation
  - $z_t^{1:N} = E_c(o_t)$
  - Uses larger token numbers to learn the underlying structure (e.g., physics, motion, etc.)
- Dynamics encoder and decoder
  - Used for tokenizing observations for $t > T_0$
  - N = 4 tokens for $T_0 + 1 : T$ observations
  - $z_t^{1:n} = E_p(o_t \,|\, o_{1:T_0})$, we condition on context observations
  - Uses cross-attention to achieve conditioning

iVideoGPT | View (V)

- Context encoder and decoder
  - N = 16 tokens for $1 : T_0$ observation
  - $z_t^{1:N} = E_c(o_t)$
  - Uses larger token numbers to learn the underlying structure (e.g., physics, motion, etc.)
- Dynamics encoder and decoder
  - Used for tokenizing observations for $t > T_0$
  - N = 4 tokens for $T_0 + 1 : T$ observations $\mathcal{L}_{\text{tokenizer}} = \sum_{t=1}^{T_0} \mathcal{L}_{\text{VQGAN}}(o_t; E_c(\cdot), D_c(\cdot)) + \sum_{t=T_0+1}^{T} \mathcal{L}_{\text{VQGAN}}(o_t; E_p(\cdot|o_{1:T_0}), D_p(\cdot|o_{1:T_0}))$
  - $z_t^{1:n} = E_p(o_t \,|\, o_{1:T_0})$, we condition on context observations
  - Uses cross-attention to achieve conditioning

iVideoGPT | View (V)

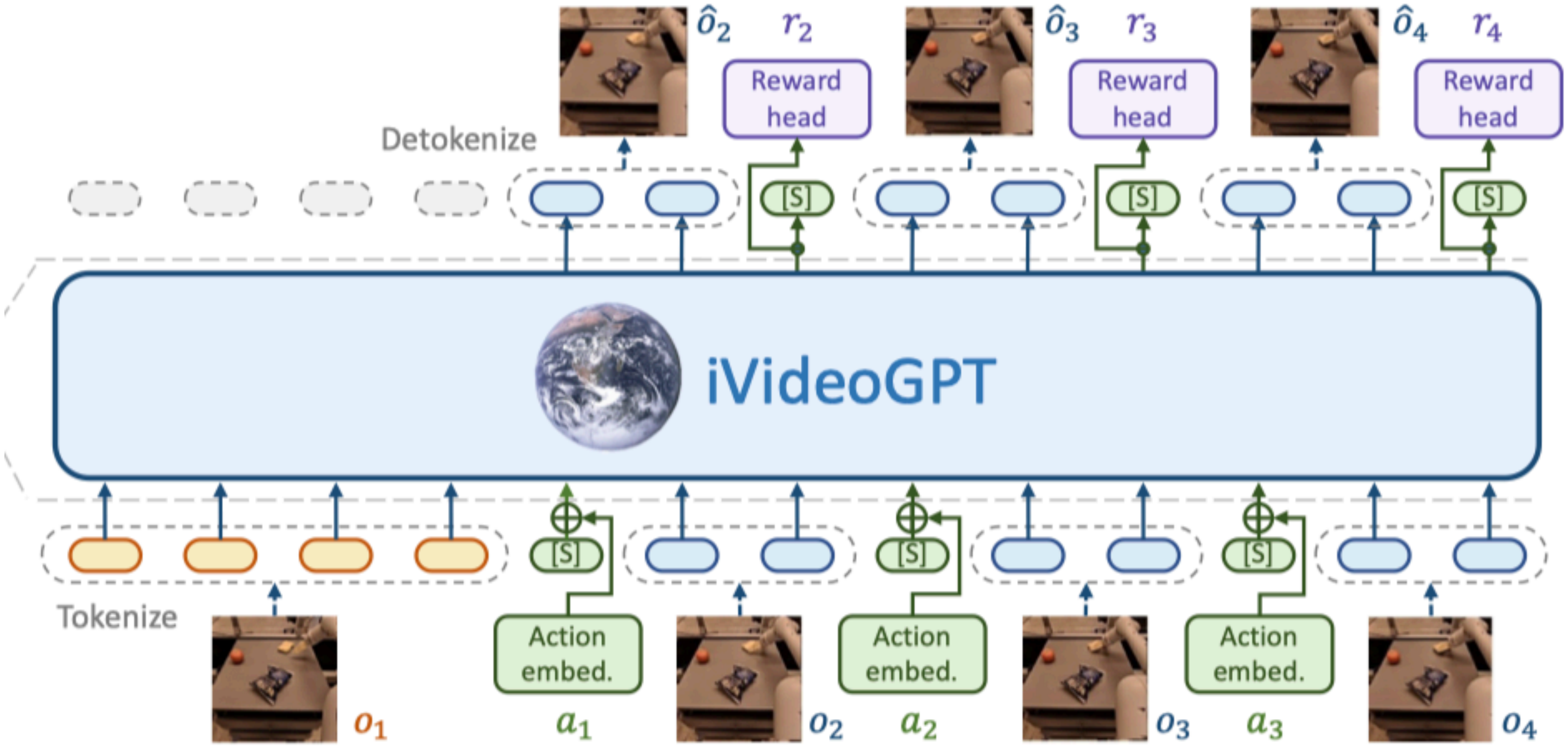


(a) Compressive tokenization

iVideoGPT | Model (M)

- N tokens for each context observation
- n tokens for each dynamics observation
- One extra 'slot' token for each observation
- Total tokens $= (N + 1)T_0 + (n + 1)(T - T_0) - 1$

iVideoGPT | Model (M)



iVideoGPT - What's there?

- V - VQGAN for image tokenization
  - Context encoder and decoder
  - Dynamics encoder and decoder
- M - GPT
  - LLAMA architecture
  - GPT-2 model size

MaskGIT

- Autoregressive token generation is *not* the best for image
- Can we generate tokens taking advantage of the spatial multi-dimensionality?
    - Faster to sample
    - Better fidelity
- Can we use ideas from BERT? 🤔

MaskGIT Training

- Learn to tokenize using VQ-GAN
- Mask n out of N tokens and predict the tokens
  - There is a special [MASK] token
  - Loss -> CE only on the masked tokens
  - $n \in [0,N]$ and monotonous function of some ratio r

MaskGIT Inference

- Start with N [MASK] tokens
- For t = 0 to T
    - Predict the tokens using bidirectional transformer
    - Take n_t out N high confidence tokens $(n_T = N)$
    - Replace the mask tokens with these tokens

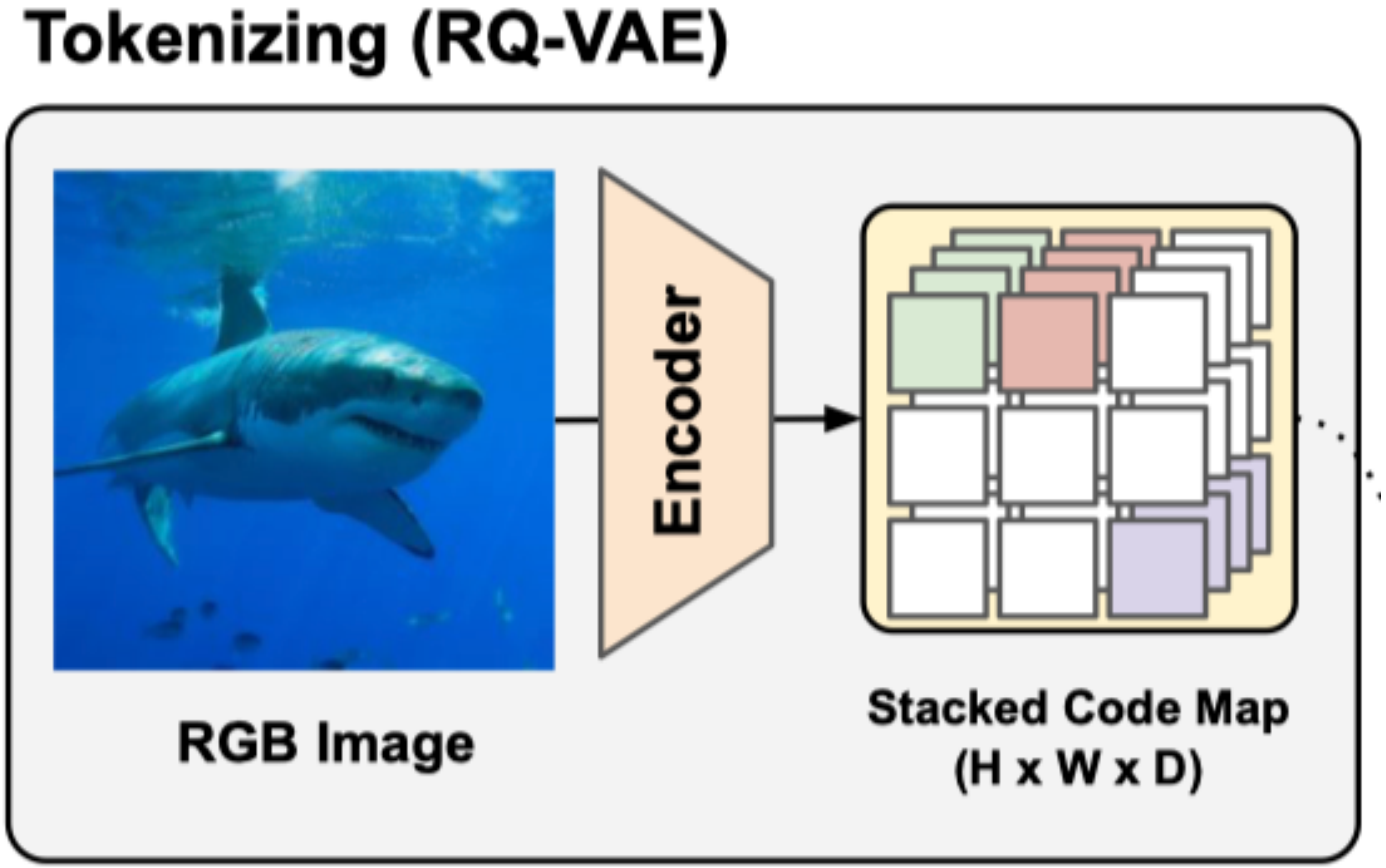
Draft and Revise

- Uses MaskGIT for model inference

Draft and Revise View (V)

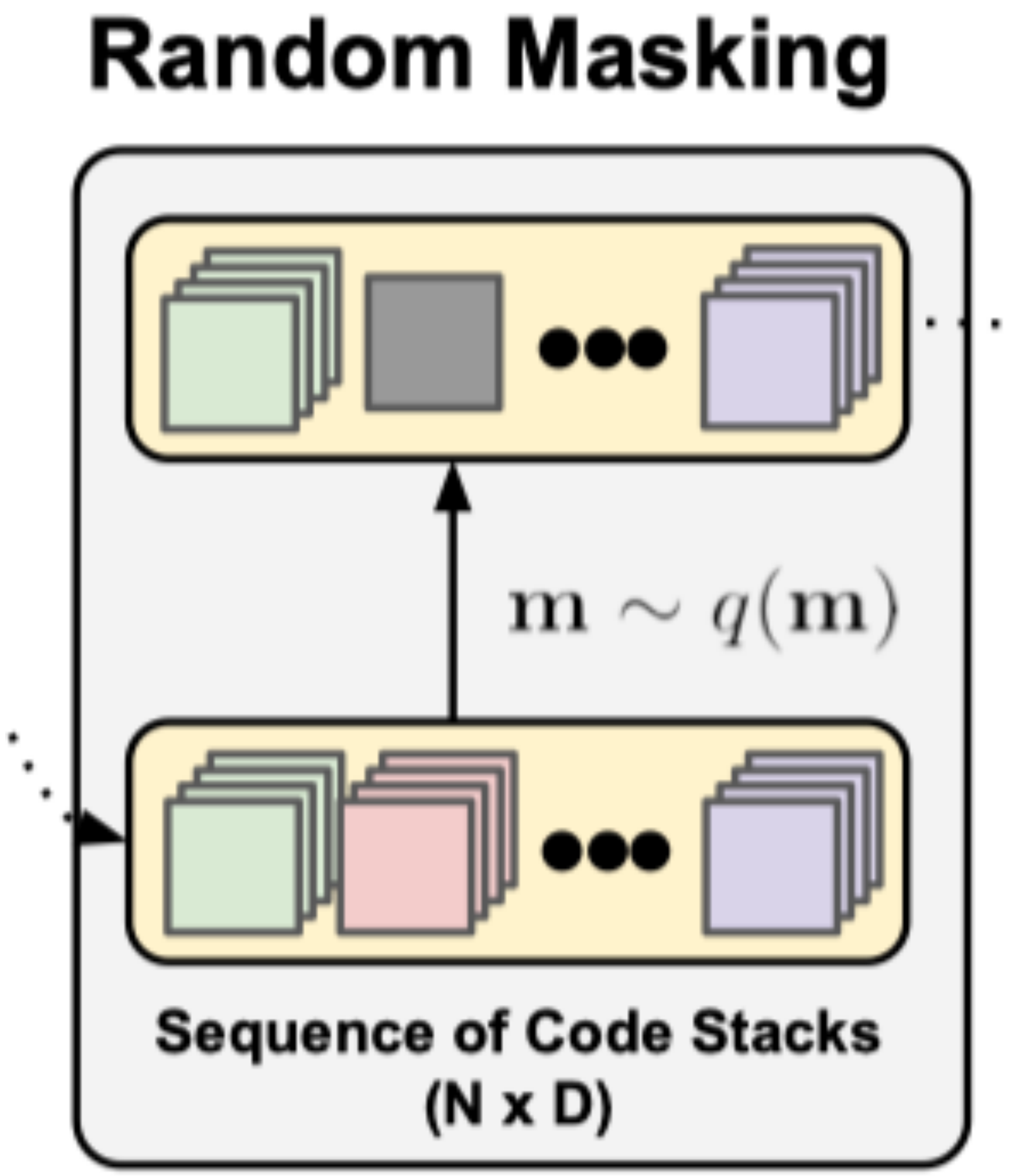
- Step 1:Turn a image into $h \times w$ continuous latents
- Step 2: Turn those into discrete latents from codebook $\mathscr{C}$
- Step 3: Find the difference.
  - Go to step 2 if numbers of codes < D

Draft and Revise View (V)



Draft and Revise Model (M)

- Similar to BERT training
- We construct a masked embedding sequence over time dimension
- The mask scheduling function is a strictly increasing function from $0 \rightarrow 1$
- The input to the transformer $\mathbf{u}_n = \mathrm{PE}_N(n) + \begin{cases} \sum_{d=1}^{D} \mathbf{e}(\mathbf{S}_{nd}) & \text{if } \mathbf{m}_n = 0 \\ \mathbf{e}_{[\mathrm{MASK}]} & \text{if } \mathbf{m}_n = 1 \end{cases}$
- The output $(\mathbf{h}_1, \cdots, \mathbf{h}_N) = f_\theta^{\mathrm{spatial}}(\mathbf{u}_1, \cdots, \mathbf{u}_N).$



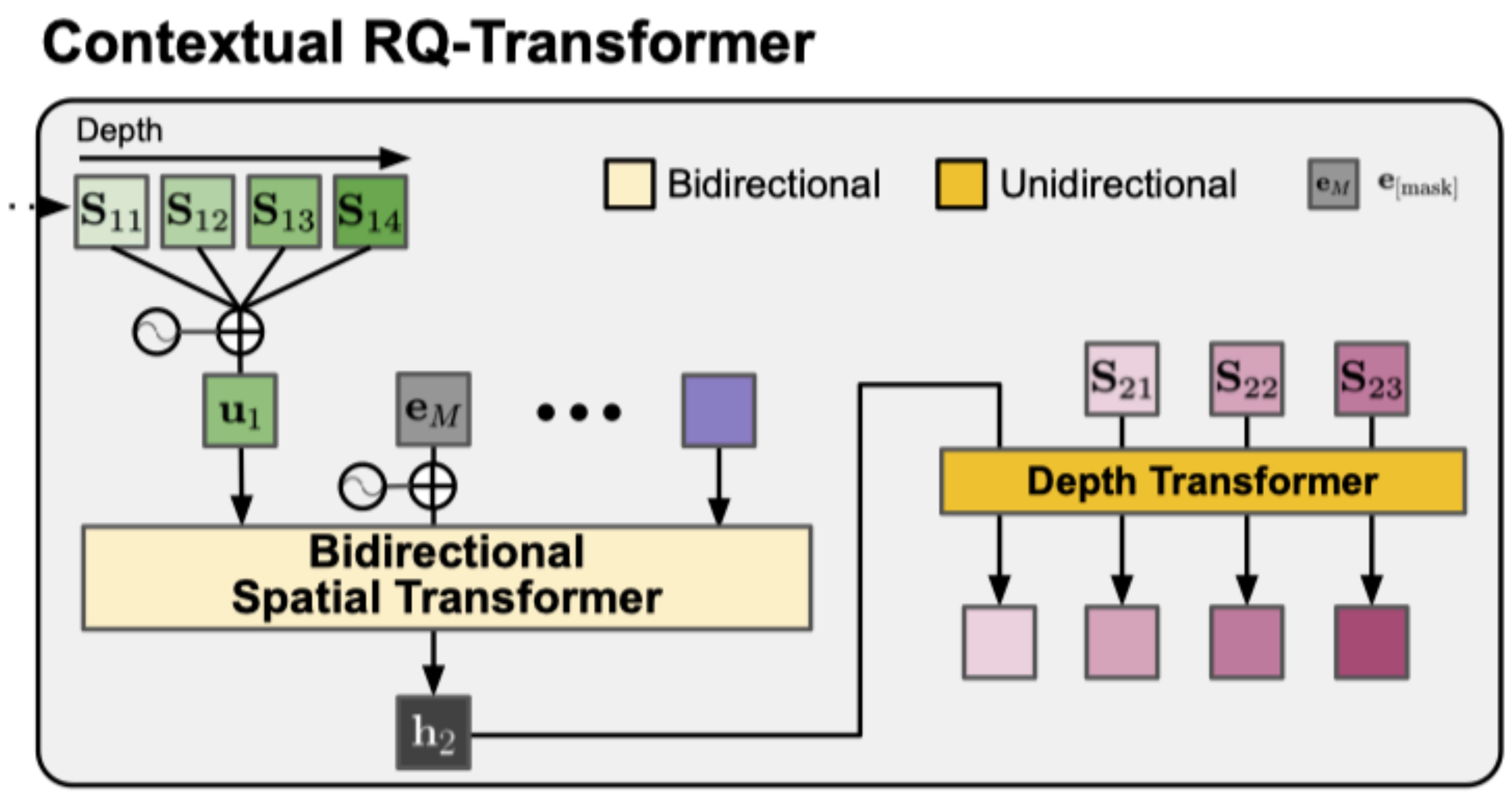
Draft and Revise Model (M)

Depth Transformer

- Autoregressive training
- We predict v_{nd} from $v_{n1,\ldots,n(d-1)}$
- Input to the transformer $\mathbf{v}_{nd} = \mathrm{PE}_D(d) + \begin{cases} \mathbf{h}_n & \text{if } d = 1 \\ \sum_{d'=1}^{d-1} \mathbf{e}(\mathbf{S}_{nd'}) & \text{if } d > 1 \end{cases}$
- Output of the transformer $\mathbf{p}_{nd} = f_\theta^{\text{depth}}(\mathbf{v}_{n1}, \cdots, \mathbf{v}_{nd})$
- We find S_{nd} from sampling the softmax of p_{nd}

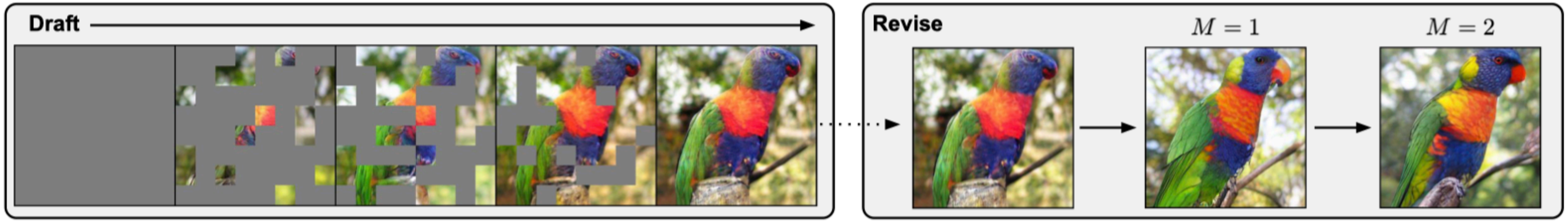
Draft and Revise Training

- Tokenize samples
- Sample masks
- Pass masked tokens and retrieve $\hat{S}_{11,\ldots,N1}$
- From $\hat{S}_{11,\ldots,N1}, \ldots \hat{S}_{1(t-1),\ldots,N(t-1)}$, predict $\hat{S}_{1t,\ldots,Nt}$
- Perform CE loss on masked code-stacks



Draft and Revise Inference

- Start with all masked positions
- **Draft:** Make a draft code-stack using bidirectional spatial transformer and depth transformer
- **Revise:** Update the code-stack conditioned on the previous one



Draft and Revise - What's there?

- V - RQ-VAE
- M - GPT and BERT

More…

- Better Model
  - Newer SSMs (S4, S5, etc.)
- Better representation for observations
  - CURL
  - STORM

More...

- Hierarchical World Modeling
  - Multi time scale world models using Gaussian marginalization and conditioning (MTS3)
- Hierarchical actors using goal-conditioning
  - Similar ideas as before
  - But, for more levels

Thank you

- Zarif Ikram
  - Visiting Researcher, NUS AI Institute
- Email : zzzarif.ikram@gmail.com
- Web: zarifikram.github.io

☺ ☺