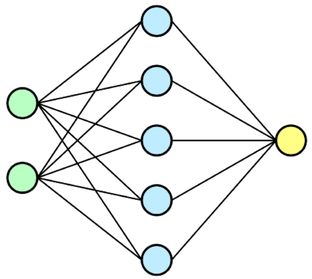
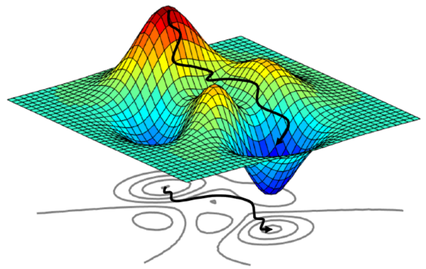
AlphaZero

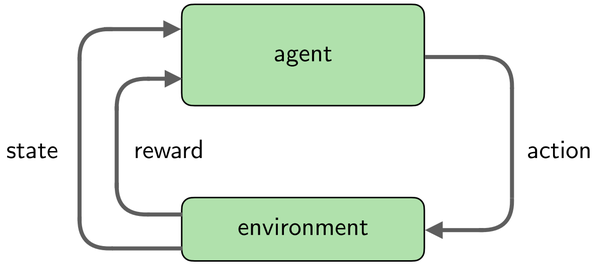
Janik Euskirchen

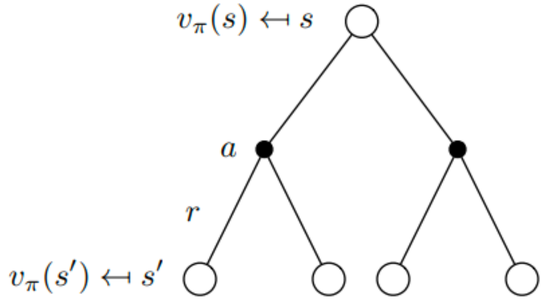
Neural networks and gradient descent



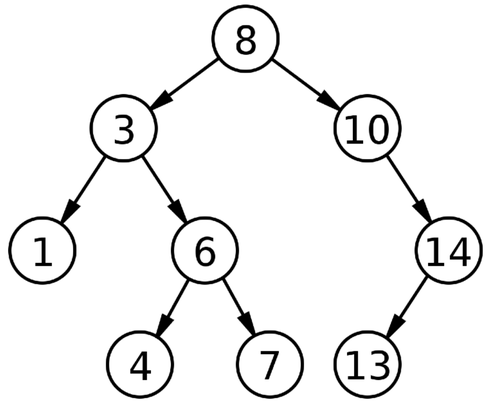


Reinforcement Learning (Basics)





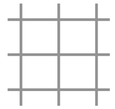
(Classical) Tree Search

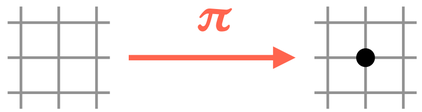


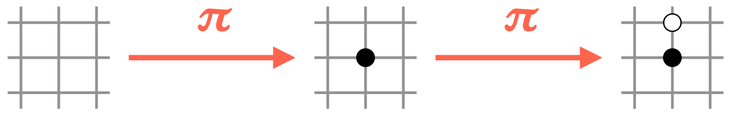
e.g. BFS, DFS

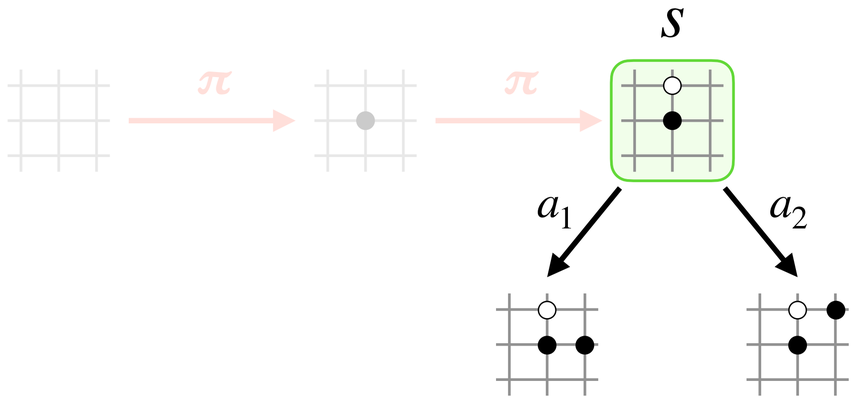
1

AlphaZero

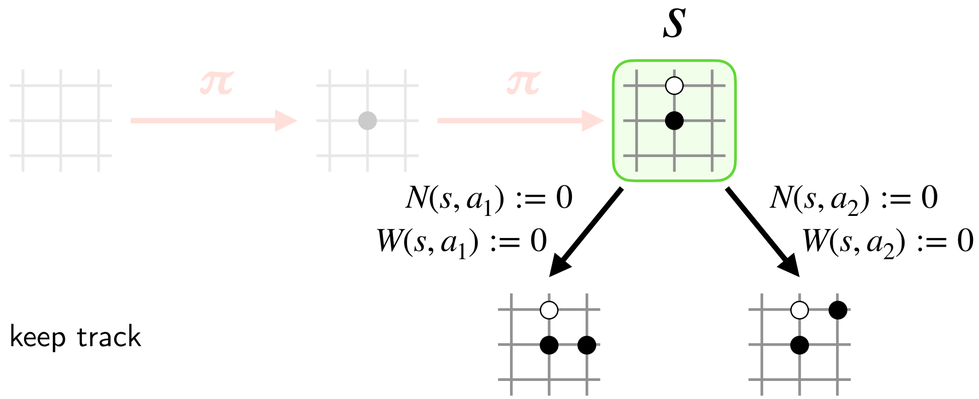



π


π
π



We can think through what the next steps would be, and whether it would result in a win.
This is called ***planning*** or ***lookahead***.



For each edge (s, a) in the tree, we keep track of two statistics:

- the visit count N
- the total value W

Take the action that maximizes the scoring function:

$$a^* := \arg\max_a U(s, a)$$

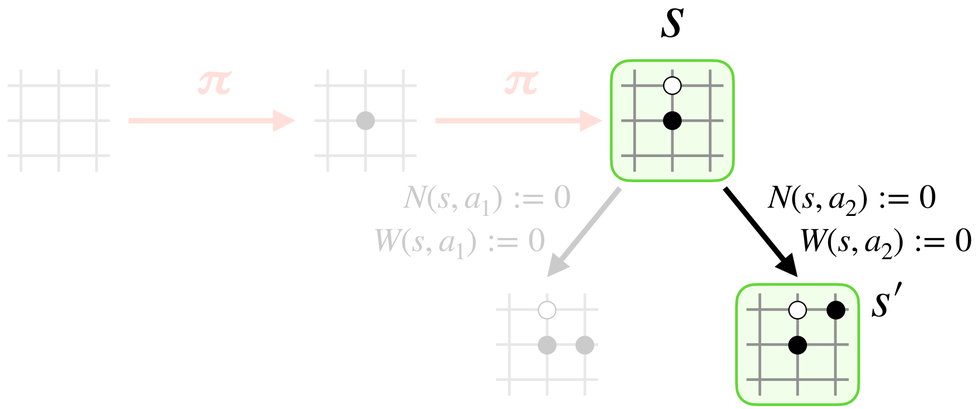
Scoring function

$$N(s) := \sum_b N(s, b)$$

$$U(s, a) := Q(s, a) + c \cdot p(s, a) \frac{N(s)}{N(s, a) + 1}$$

$$Q(s, a) = \frac{W(s, a)}{N(s, a)}$$

$$(\mathbf{p}, v) = f_\theta(s)$$



Take the action that maximizes the scoring function:

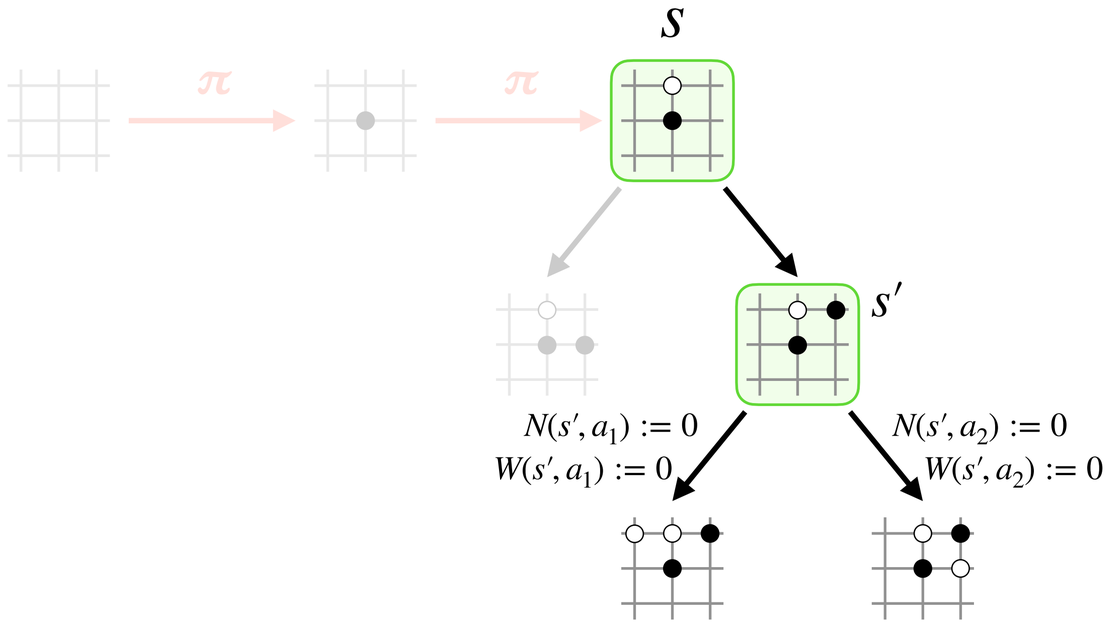
$$a^* := \arg\max_a U(s, a)$$

Scoring function

$$U(s, a) := Q(s, a) + c \cdot p(s, a) \frac{N(s)}{N(s, a) + 1}$$

$$N(s) := \sum_b N(s, b)$$

$$\frac{W(s, a)}{N(s, a)} \qquad f_\theta(s)$$



Take the action that maximizes the scoring function:

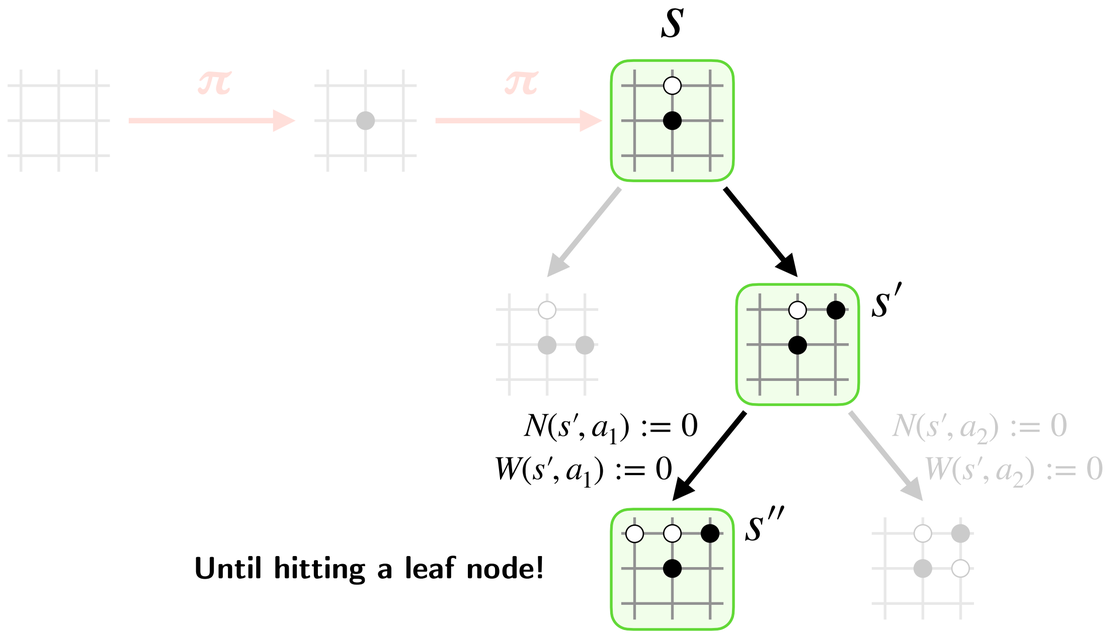
$$a^* := \arg\max_a U(s, a)$$

$$N(s) := \sum_b N(s, b)$$

Scoring function

$$U(s, a) := Q(s, a) + c \cdot p(s, a) \frac{N(s)}{N(s, a) + 1}$$

$\frac{W(s, a)}{N(s, a)}$ $f_\theta(s)$



Until hitting a leaf node!

In the first simulation, this will be right away.

But after some time, the simulations will consist of many steps.

Take the action that maximizes the scoring function:

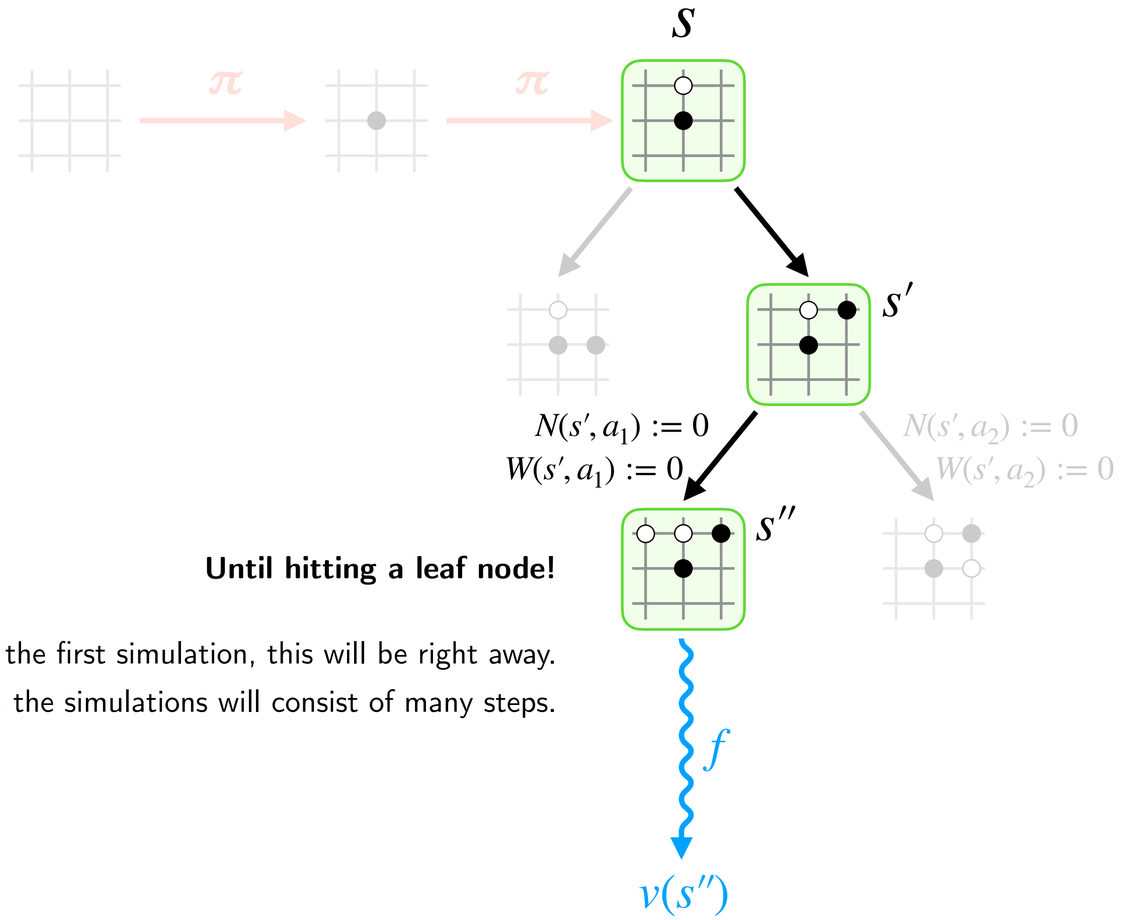
$$a^* := \arg\max_a U(s, a)$$

$N(s) := \sum_b N(s, b)$

Scoring function

$$U(s, a) := Q(s, a) + c \cdot p(s, a) \frac{N(s)}{N(s, a) + 1}$$

$\frac{W(s, a)}{N(s, a)}$ $f_\theta(s)$



Until hitting a leaf node!

In the first simulation, this will be right away.

But after some time, the simulations will consist of many steps.

Take the action that maximizes the scoring function:

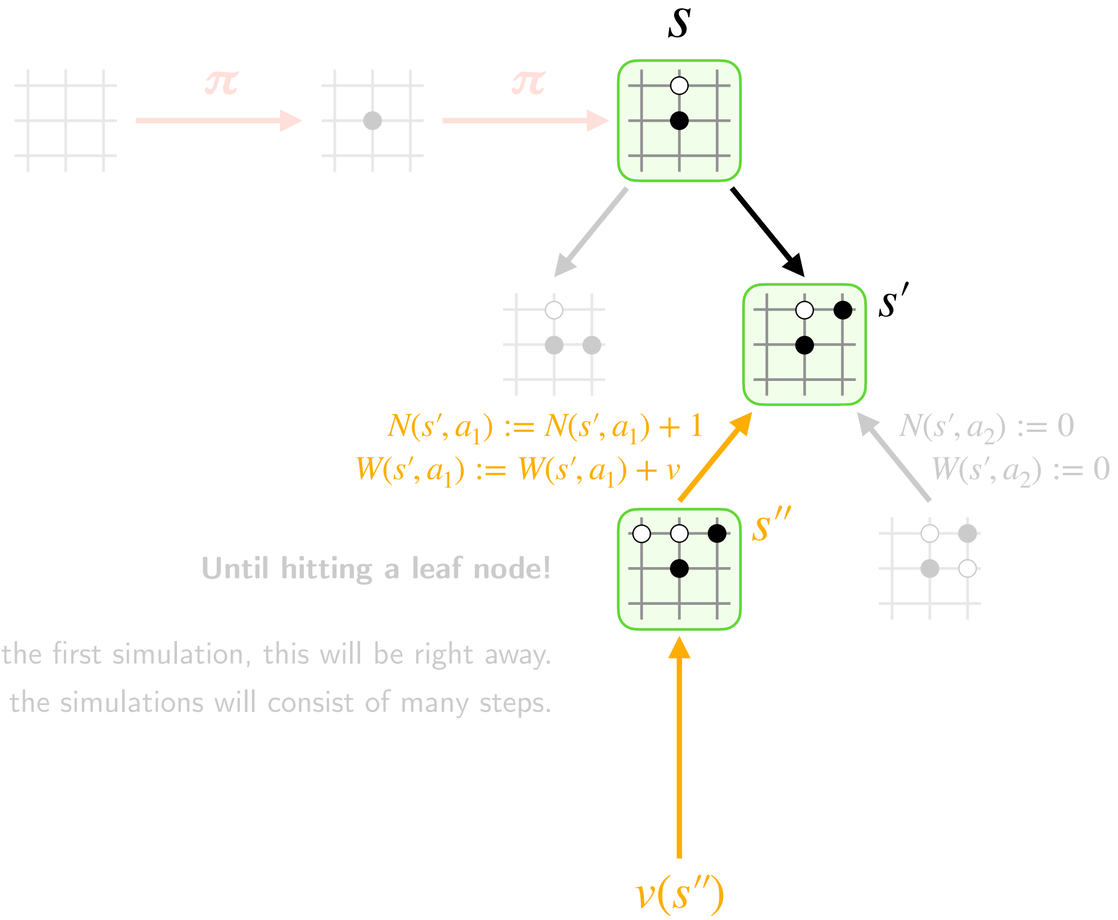
$$a^* := \arg\max_a U(s,a)$$

$N(s) := \sum_b N(s,b)$

Scoring function

$$U(s,a) := Q(s,a) + c \cdot p(s,a) \frac{N(s)}{N(s,a)+1}$$

$\frac{W(s,a)}{N(s,a)}$ $f_\theta(s)$



Take the action that maximizes the scoring function:

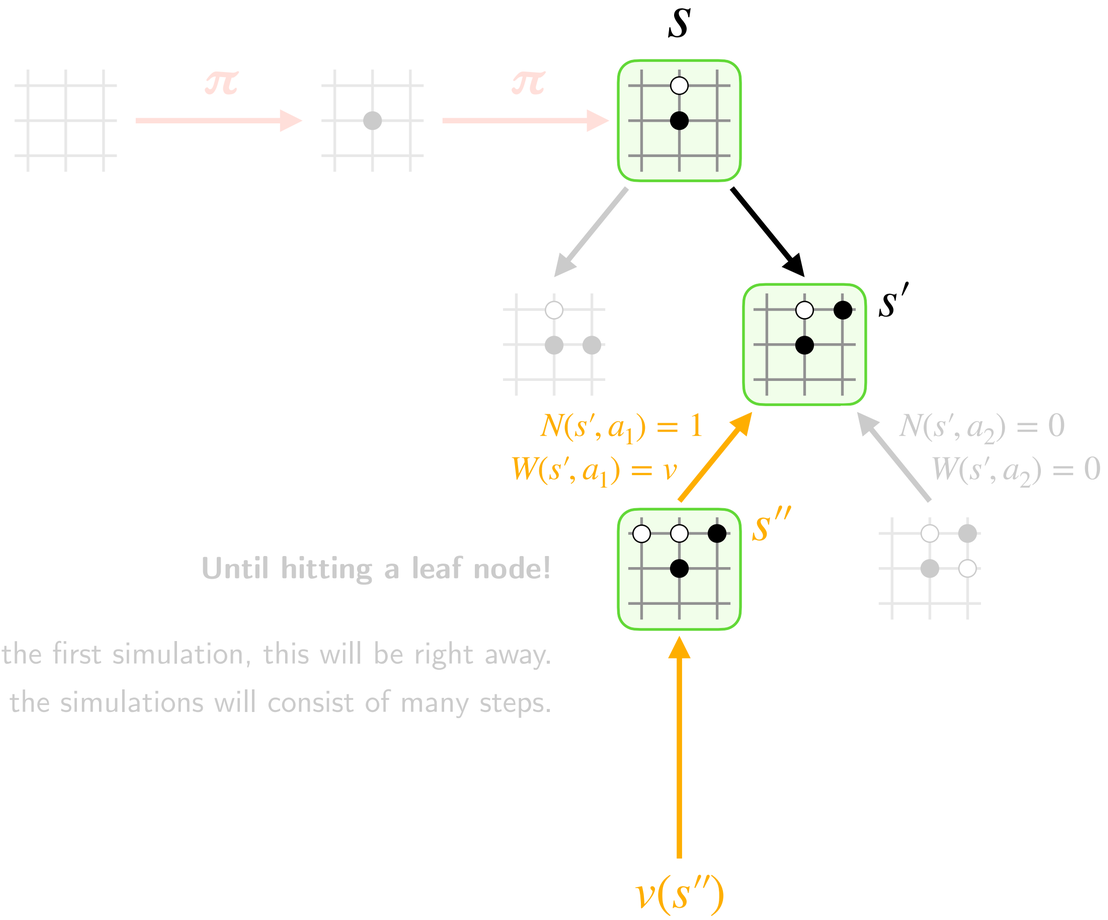
$$a^* := \arg\max_a U(s, a)$$

$N(s) := \sum_b N(s, b)$

Scoring function

$$U(s, a) := Q(s, a) + c \cdot p(s, a) \frac{N(s)}{N(s, a) + 1}$$

$\frac{W(s, a)}{N(s, a)}$ $f_\theta(s)$



Until hitting a leaf node!

In the first simulation, this will be right away.

But after some time, the simulations will consist of many steps.

Take the action that maximizes the scoring function:

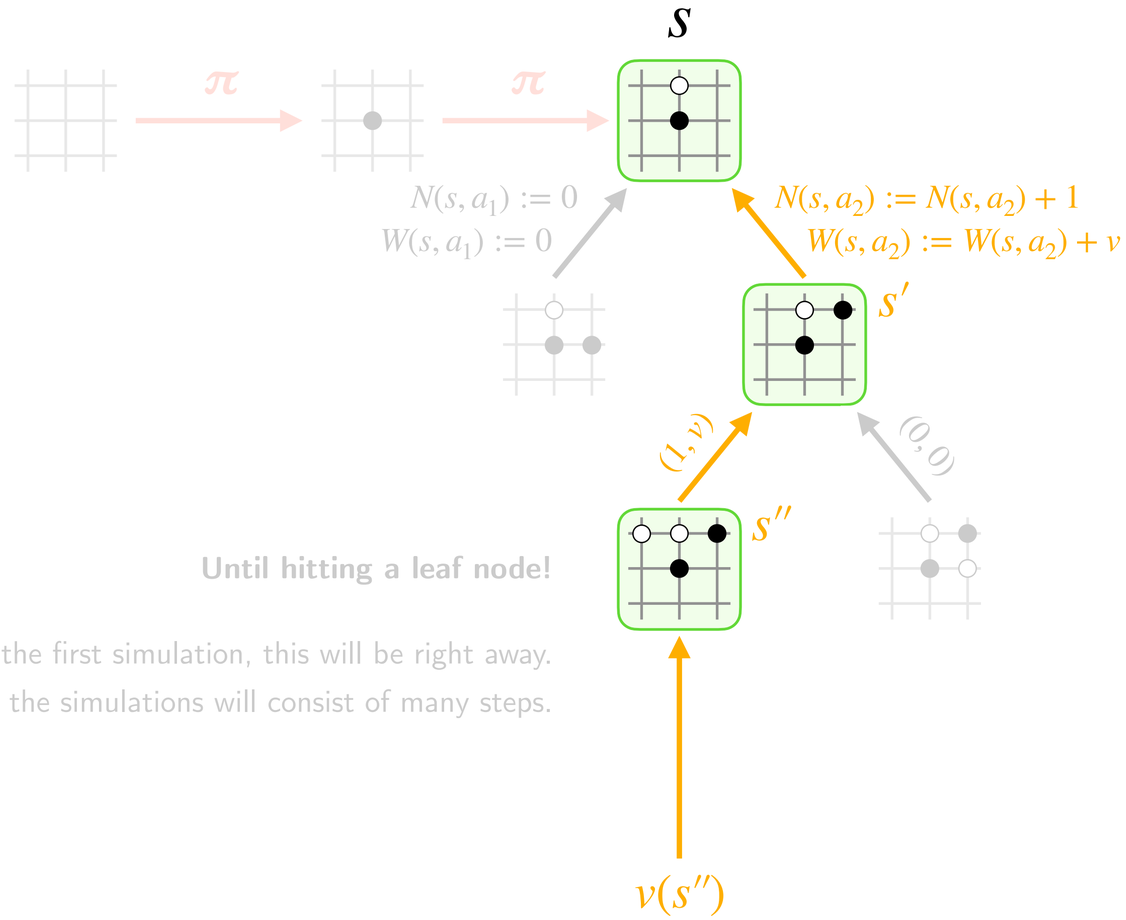
$$a^* := \arg\max_a U(s,a)$$

Scoring function

$$N(s) := \sum_b N(s,b)$$

$$U(s,a) := Q(s,a) + c \cdot p(s,a) \frac{N(s)}{N(s,a)+1}$$

$\frac{W(s,a)}{N(s,a)}$ $f_\theta(s)$



Until hitting a leaf node!

In the first simulation, this will be right away.

But after some time, the simulations will consist of many steps.

Take the action that maximizes the scoring function:

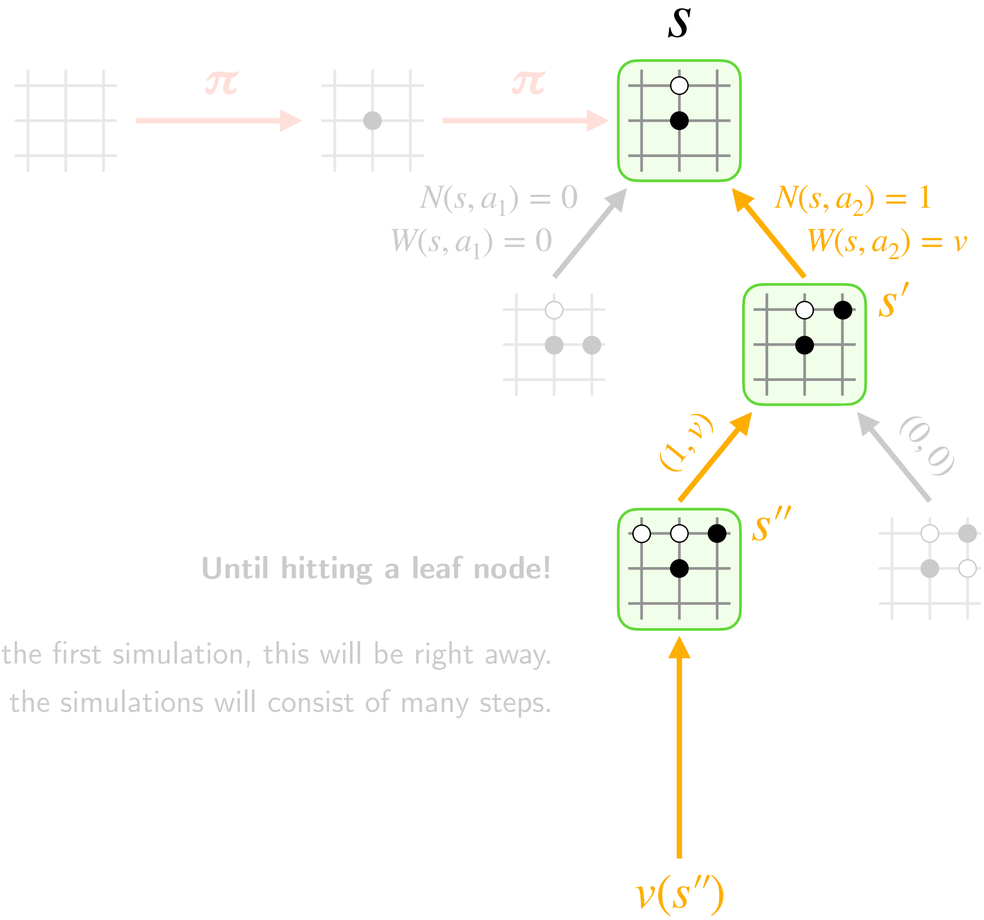
$$a^* := \arg\max_a U(s, a)$$

Scoring function

$$N(s) := \sum_b N(s, b)$$

$$U(s, a) := Q(s, a) + c \cdot p(s, a) \frac{N(s)}{N(s, a) + 1}$$

$\frac{W(s,a)}{N(s,a)}$ $f_\theta(s)$



Until hitting a leaf node!

In the first simulation, this will be right away.

But after some time, the simulations will consist of many steps.

Take the action that maximizes the scoring function:

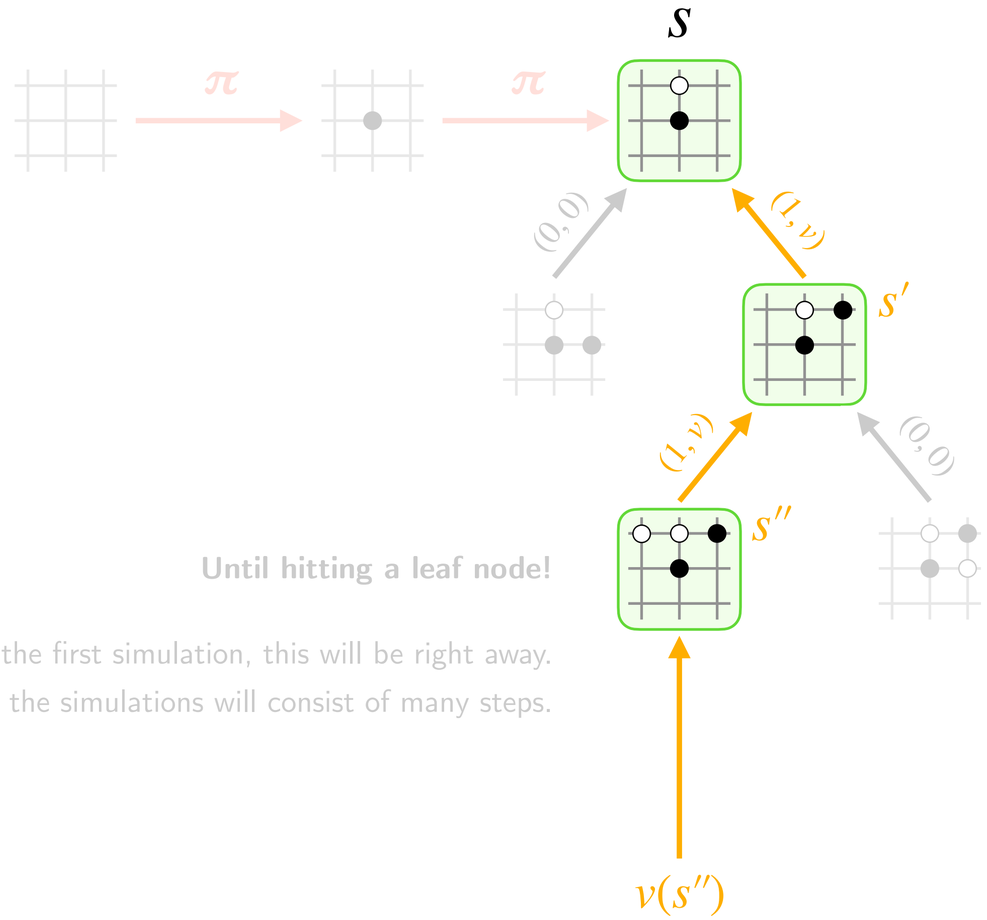
$$a^* := \arg\max_a U(s, a)$$

Scoring function

$$N(s) := \sum_b N(s, b)$$

$$U(s, a) := Q(s, a) + c \cdot p(s, a) \frac{N(s)}{N(s, a) + 1}$$

$\frac{W(s, a)}{N(s, a)}$ $f_\theta(s)$



Until hitting a leaf node!

In the first simulation, this will be right away.

But after some time, the simulations will consist of many steps.

Take the action that maximizes the scoring function:

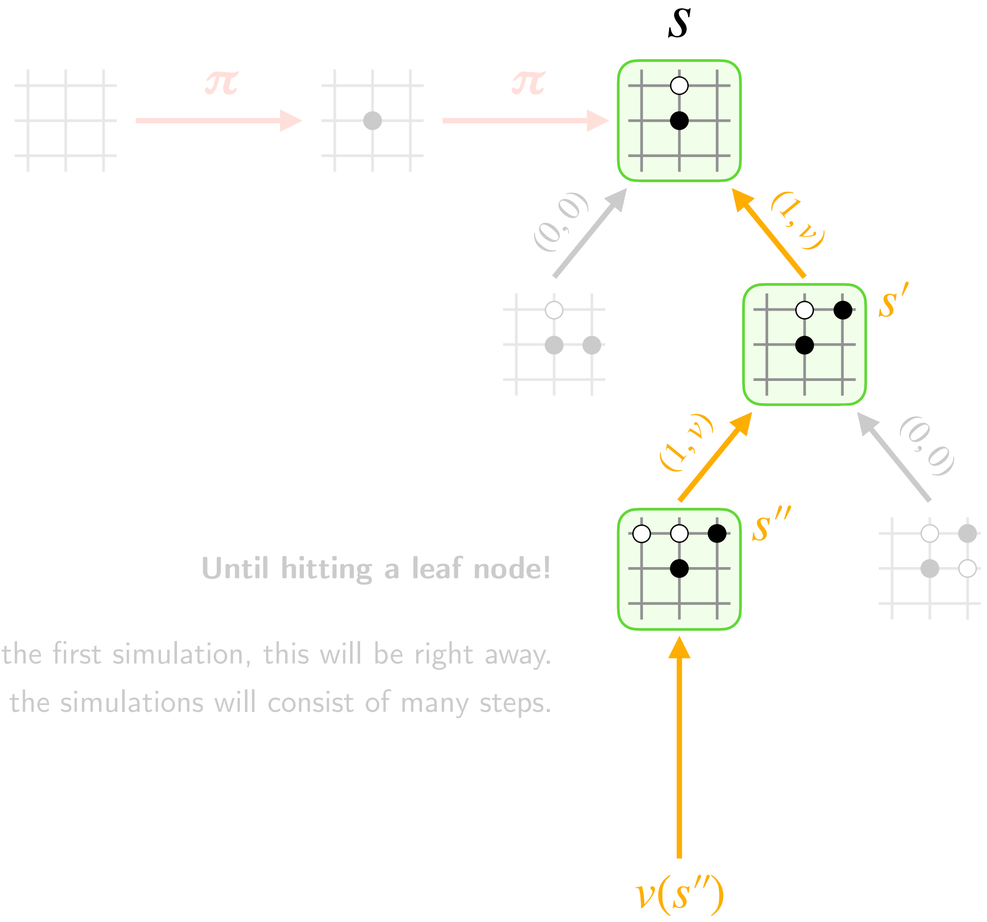
$$a^* := \arg\max_a U(s,a)$$

Scoring function

$$N(s) := \sum_b N(s,b)$$

$$U(s,a) := Q(s,a) + c \cdot p(s,a) \frac{N(s)}{N(s,a)+1}$$

$\frac{W(s,a)}{N(s,a)}$ $f_\theta(s)$



Until hitting a leaf node!

In the first simulation, this will be right away.

But after some time, the simulations will consist of many steps.

Take the action that maximizes the scoring function:

$$a^* := \arg\max_a U(s,a)$$

$$N(s) := \sum_b N(s,b)$$

Scoring function

$$U(s,a) := Q(s,a) + c \cdot p(s,a) \frac{N(s)}{N(s,a)+1}$$

$\frac{W(s,a)}{N(s,a)}$ $f_\theta(s)$

Run many simulations like this.

Each will be slightly different, as each time it uses slightly different tree statistics to traverse the tree.

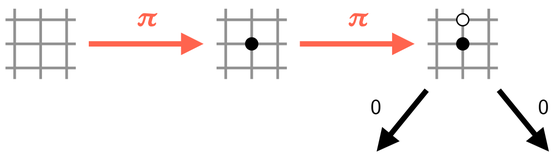
The **predicted policy *p*** guides the tree search so we only take reasonable trajectories.
The **predicted value *v*** truncates the tree so we don't have to play all the way to the end of the game.

Together, they make the tree search feasible.

Simulation 1

Simplified to demonstrate the flow of AlphaZero MCTS

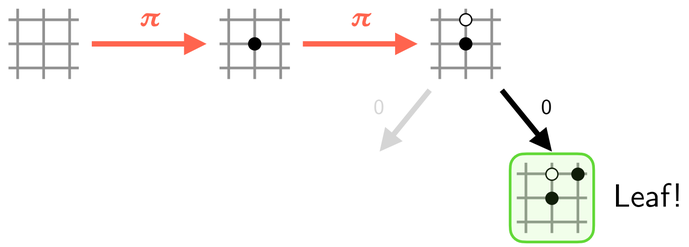
Simulation 1



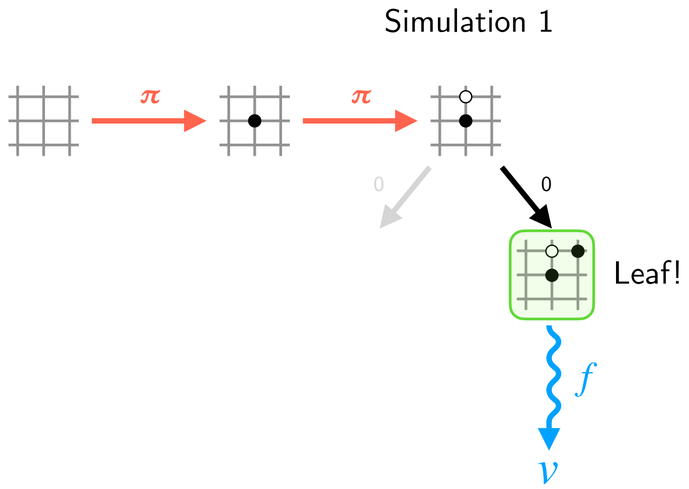
$\arg\max_a U(s,a)$

Simplified to demonstrate the flow of AlphaZero MCTS

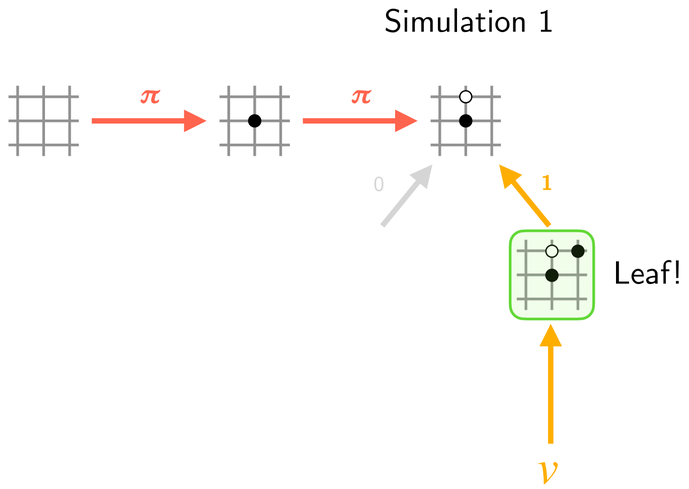
Simulation 1



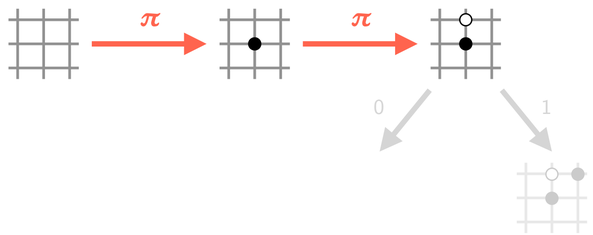
Simplified to demonstrate the flow of AlphaZero MCTS



Simplified to demonstrate the flow of AlphaZero MCTS

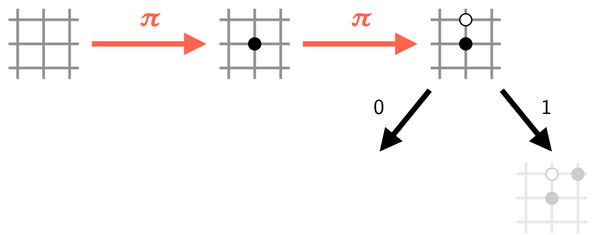


Simplified to demonstrate the flow of AlphaZero MCTS



Simplified to demonstrate the flow of AlphaZero MCTS

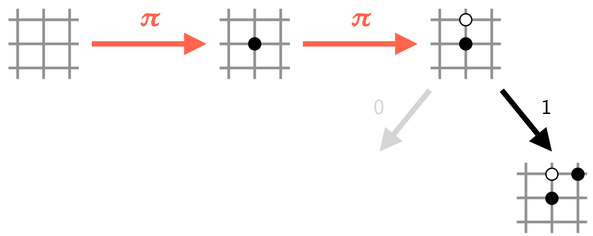
Simulation 2



$\arg\max_a U(s,a)$

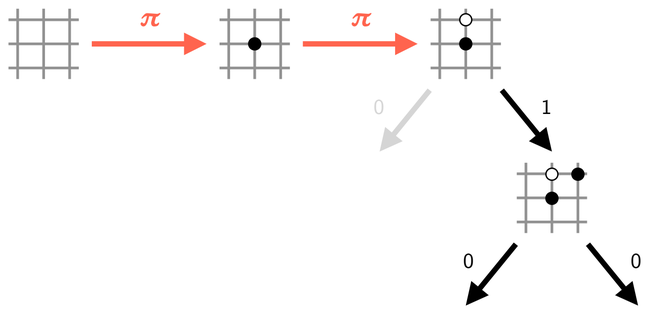
Simplified to demonstrate the flow of AlphaZero MCTS

Simulation 2



Simplified to demonstrate the flow of AlphaZero MCTS

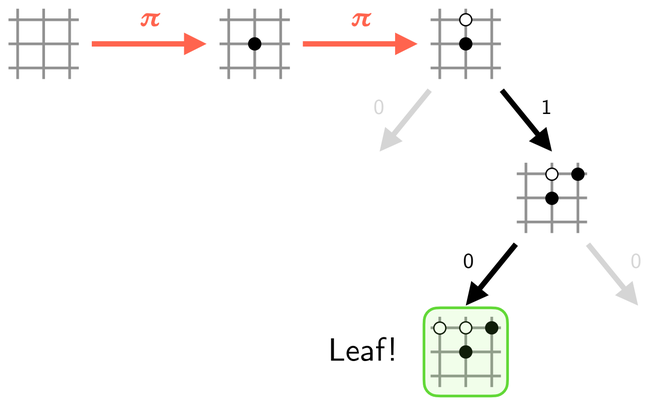
Simulation 2



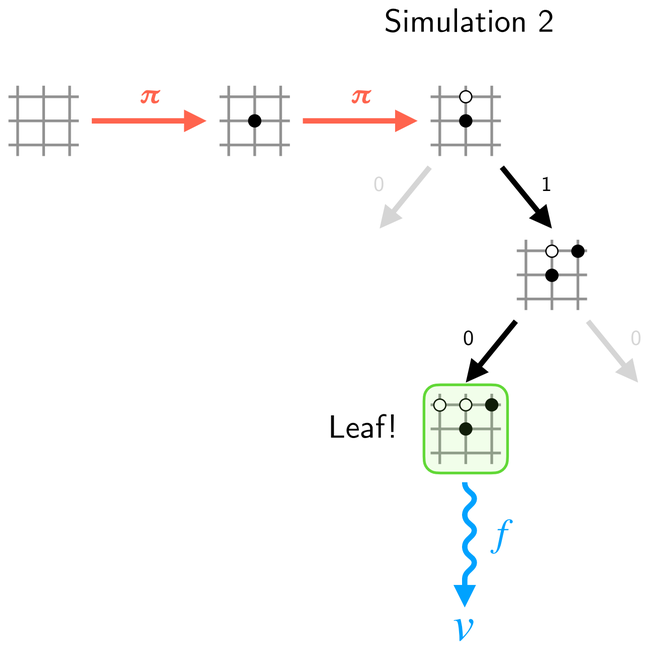
$\arg\max_a U(s, a)$

Simplified to demonstrate the flow of AlphaZero MCTS

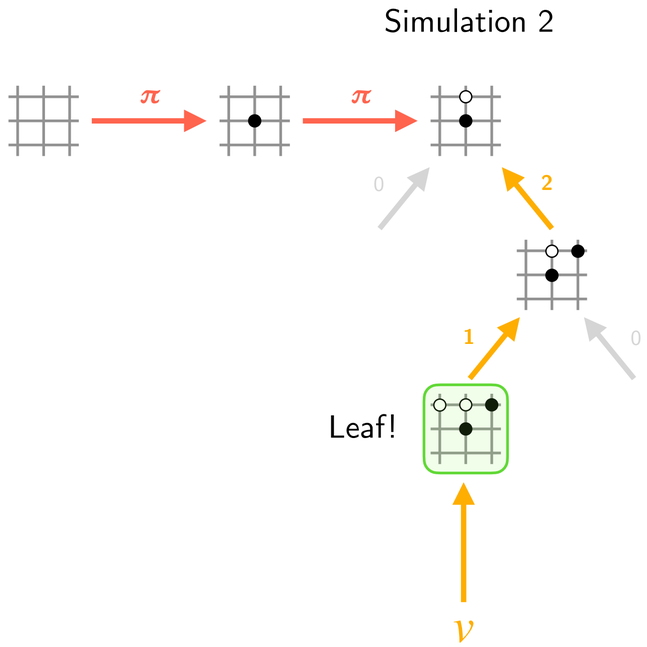
Simulation 2



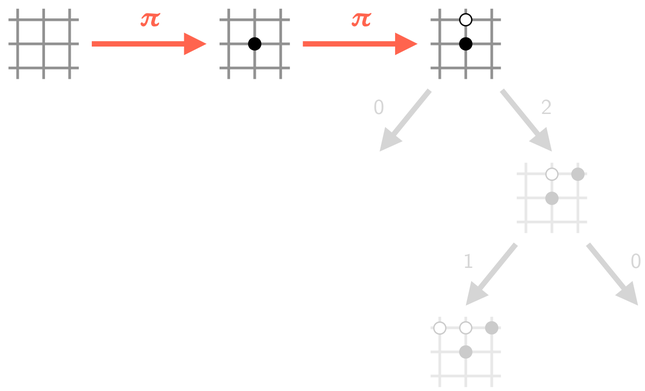
Simplified to demonstrate the flow of AlphaZero MCTS



Simplified to demonstrate the flow of AlphaZero MCTS

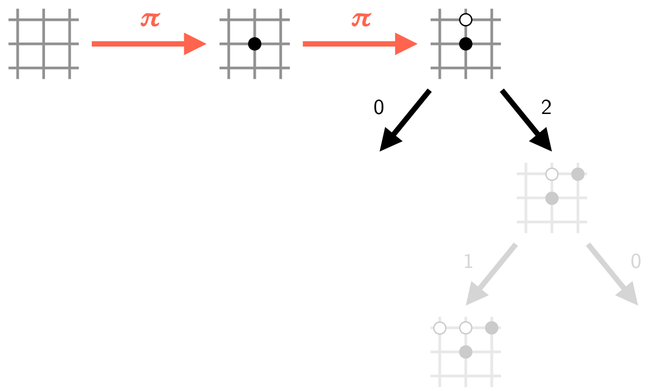


Simplified to demonstrate the flow of AlphaZero MCTS



Simplified to demonstrate the flow of AlphaZero MCTS

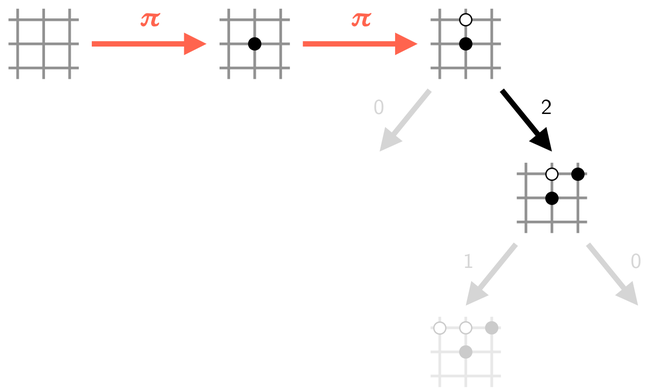
Simulation 3



$\arg\max_a U(s,a)$

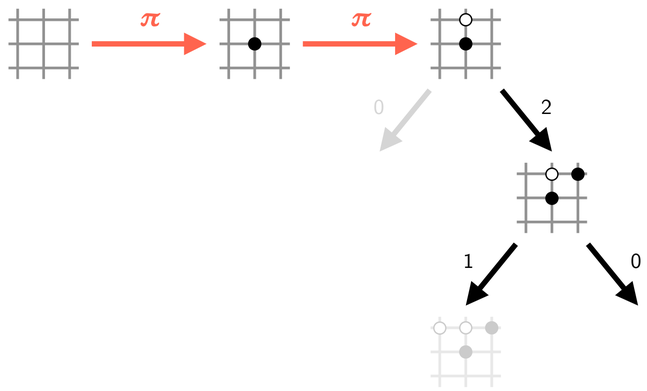
Simplified to demonstrate the flow of AlphaZero MCTS

Simulation 3



Simplified to demonstrate the flow of AlphaZero MCTS

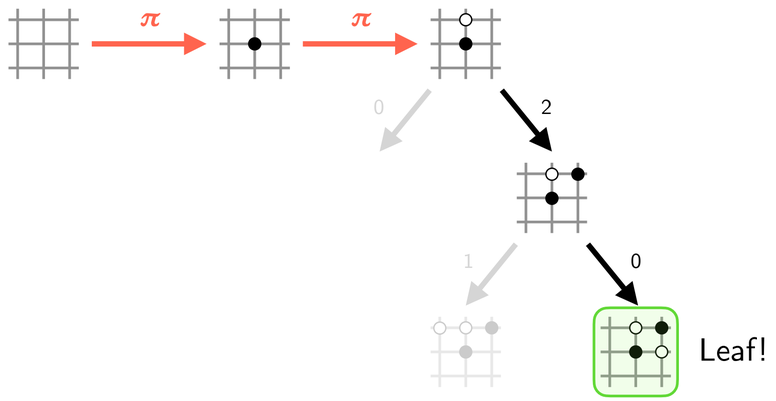
Simulation 3



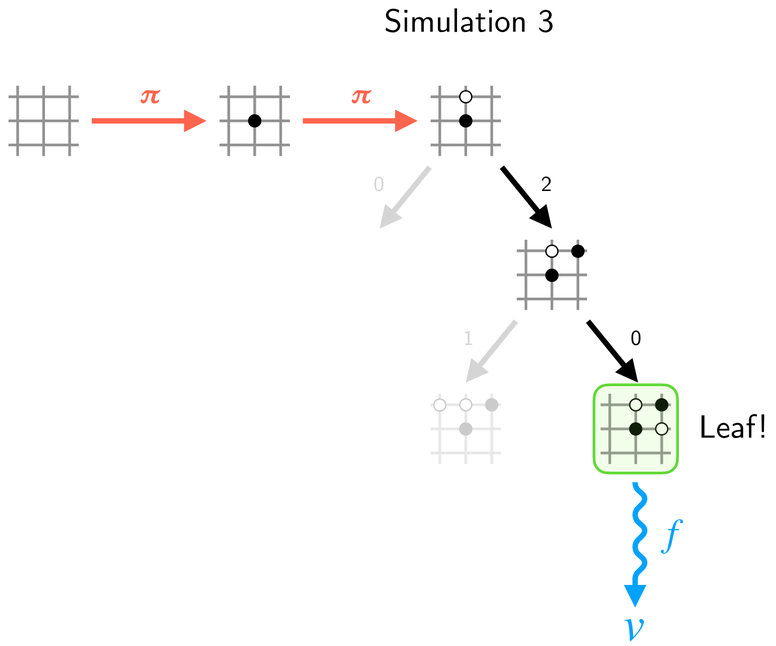
$\arg\max_a U(s, a)$

Simplified to demonstrate the flow of AlphaZero MCTS

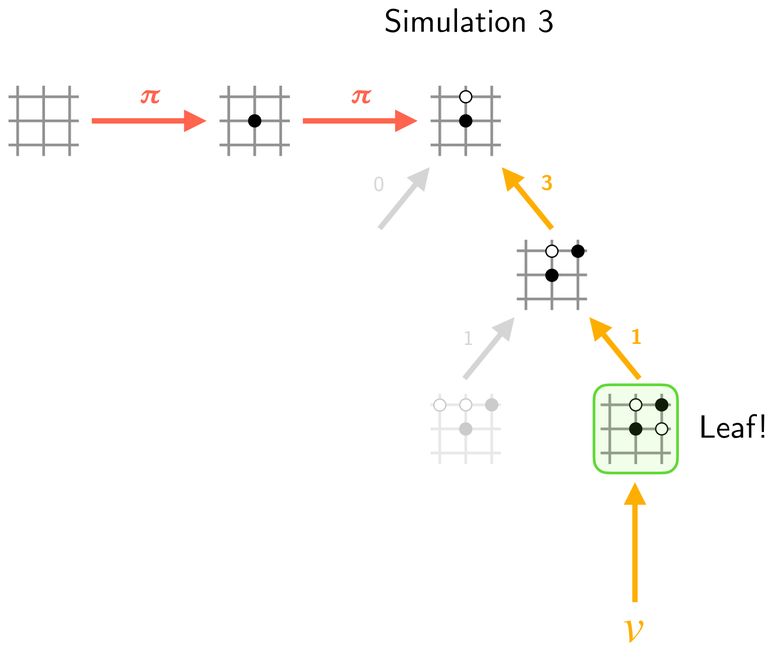
Simulation 3



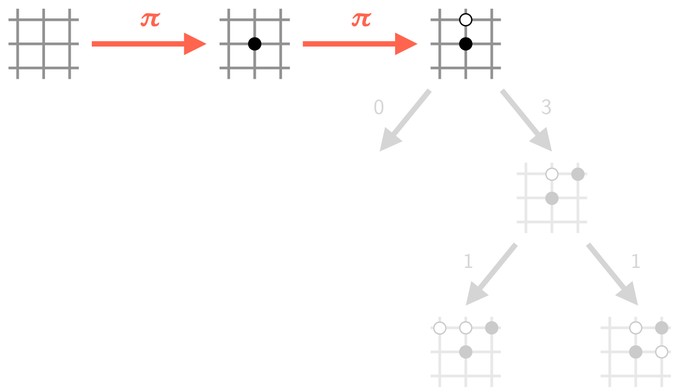
Simplified to demonstrate the flow of AlphaZero MCTS



Simplified to demonstrate the flow of AlphaZero MCTS

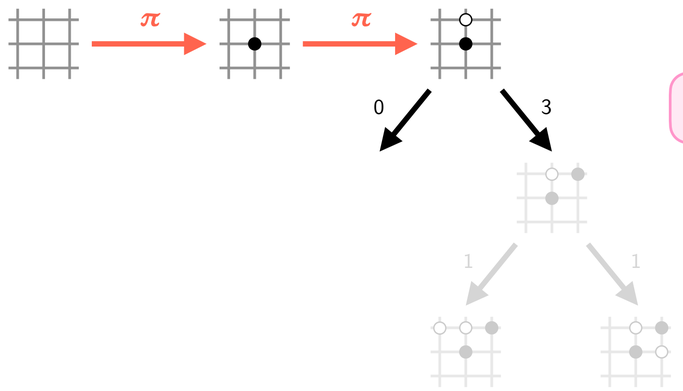


Simplified to demonstrate the flow of AlphaZero MCTS



Simplified to demonstrate the flow of AlphaZero MCTS

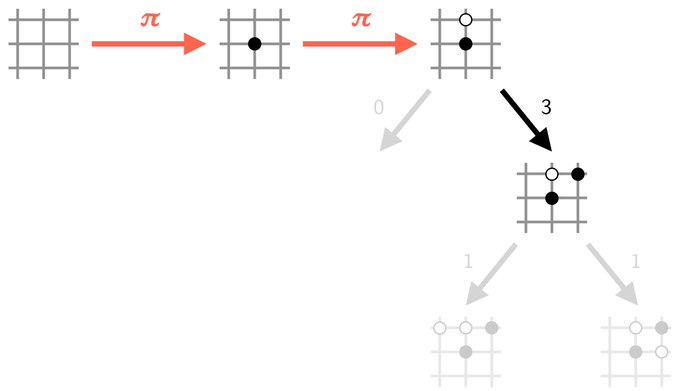
Simulation 4



$\arg\max_a U(s,a)$

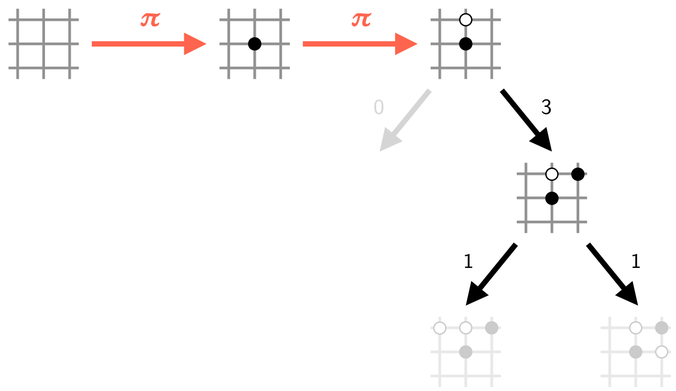
Simplified to demonstrate the flow of AlphaZero MCTS

Simulation 4



Simplified to demonstrate the flow of AlphaZero MCTS

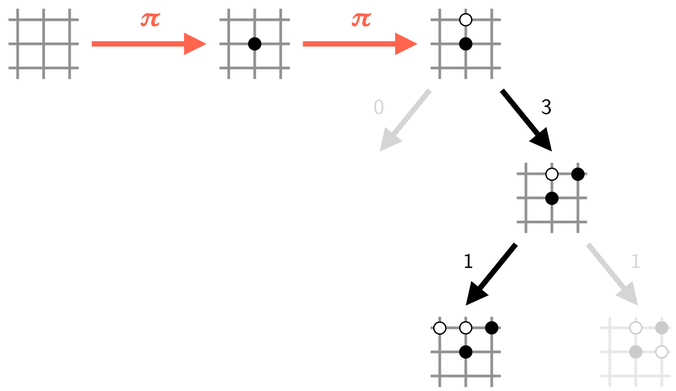
Simulation 4



$\arg\max_a U(s, a)$

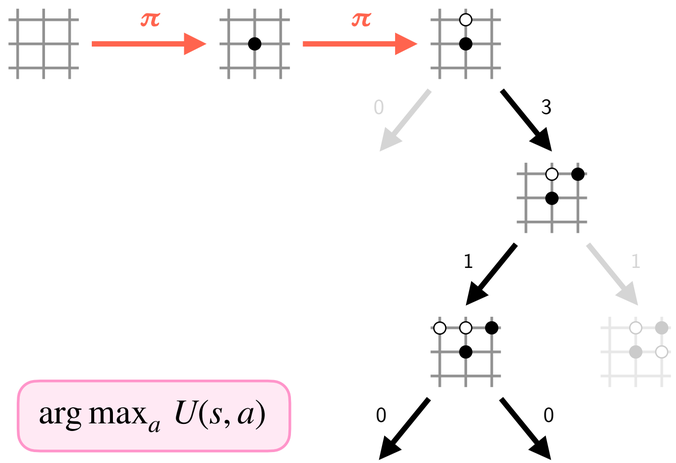
Simplified to demonstrate the flow of AlphaZero MCTS

Simulation 4



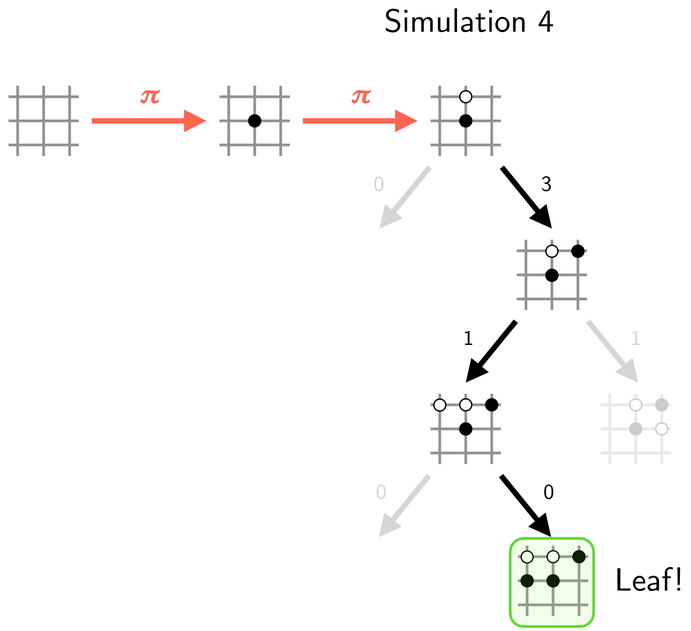
Simplified to demonstrate the flow of AlphaZero MCTS

Simulation 4



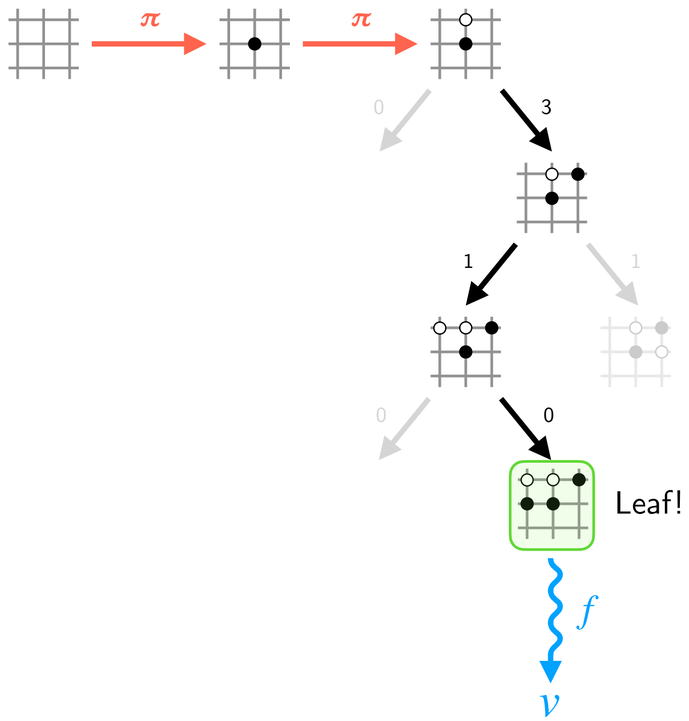
$\arg\max_a U(s, a)$

Simplified to demonstrate the flow of AlphaZero MCTS

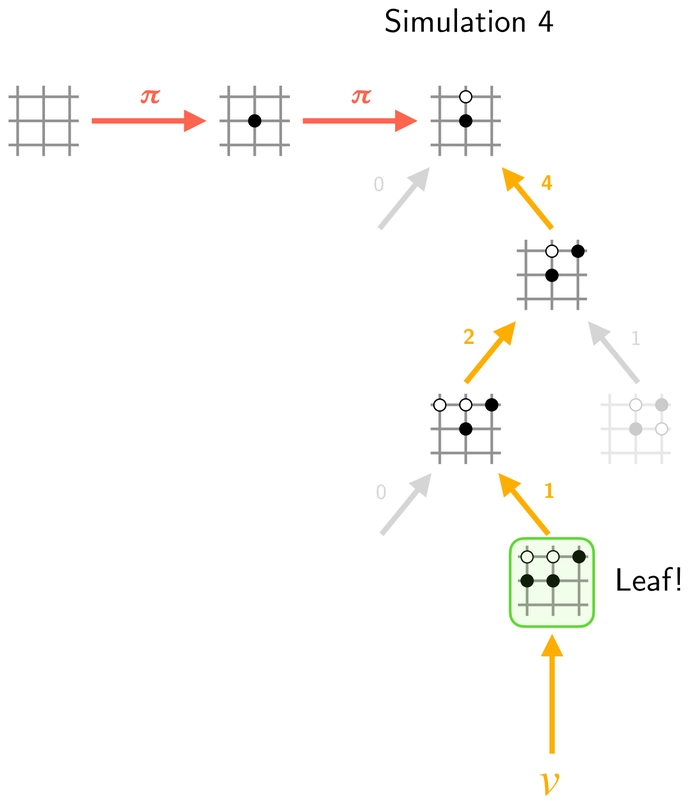


Simplified to demonstrate the flow of AlphaZero MCTS

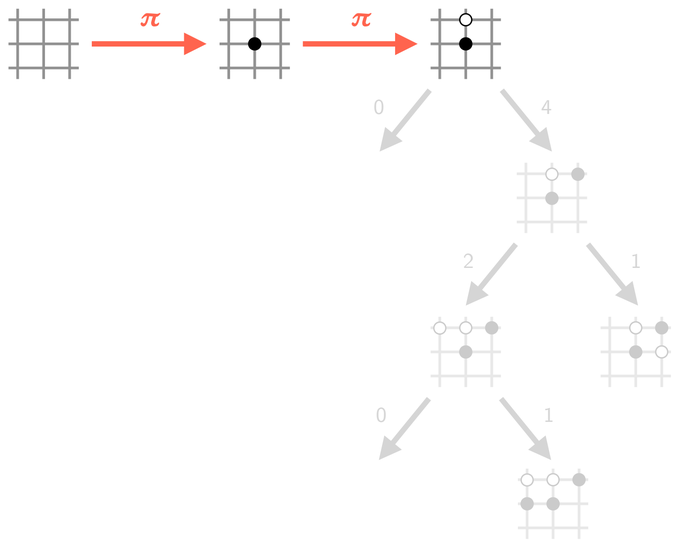
Simulation 4



Simplified to demonstrate the flow of AlphaZero MCTS

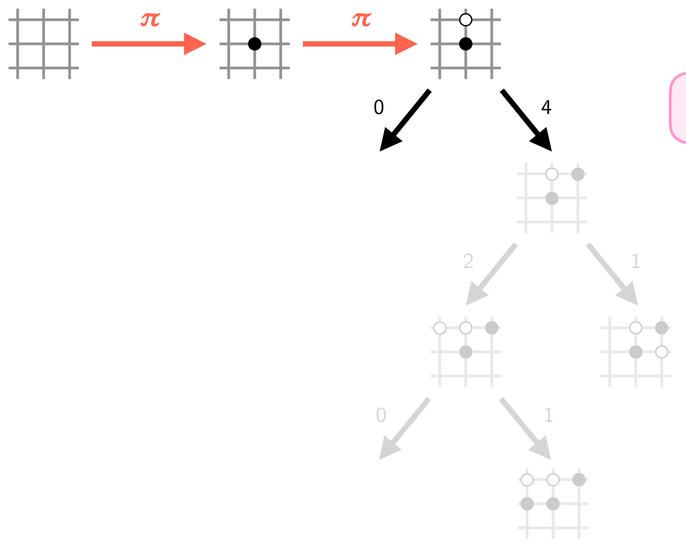


Simplified to demonstrate the flow of AlphaZero MCTS



Simplified to demonstrate the flow of AlphaZero MCTS

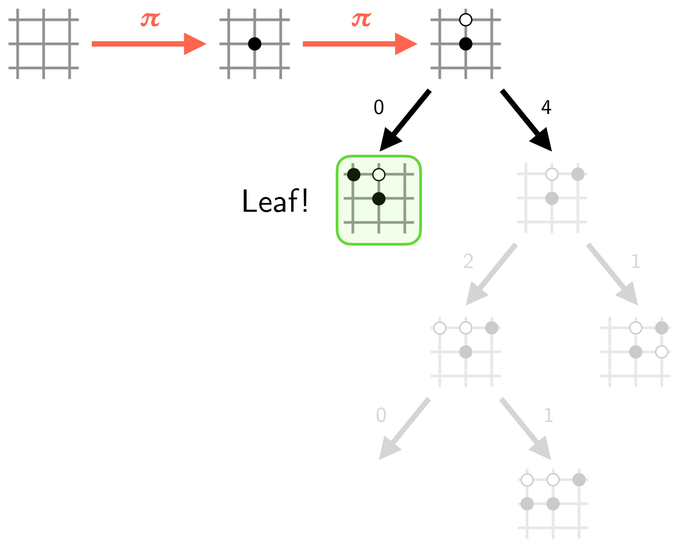
Simulation 5



$\arg\max_a U(s,a)$

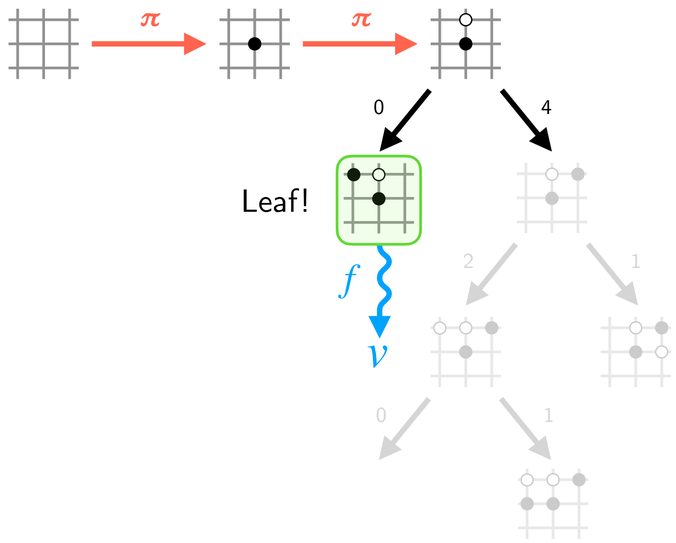
Simplified to demonstrate the flow of AlphaZero MCTS

Simulation 5



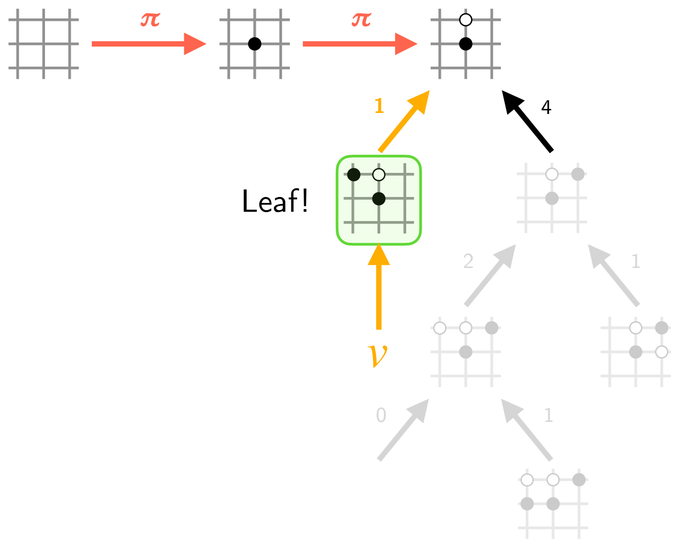
Simplified to demonstrate the flow of AlphaZero MCTS

Simulation 5



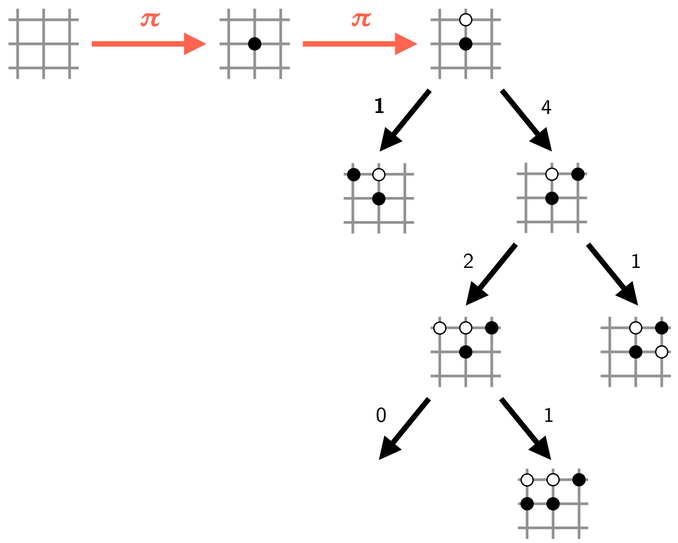
Simplified to demonstrate the flow of AlphaZero MCTS

Simulation 5



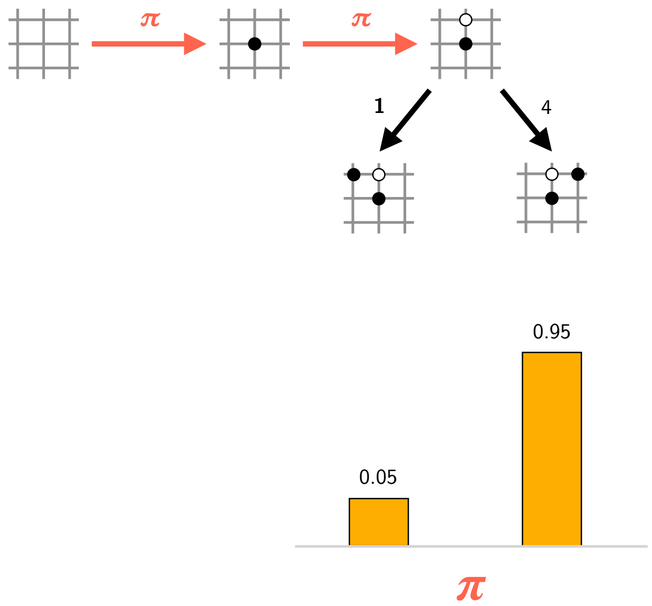
Simplified to demonstrate the flow of AlphaZero MCTS

Full tree

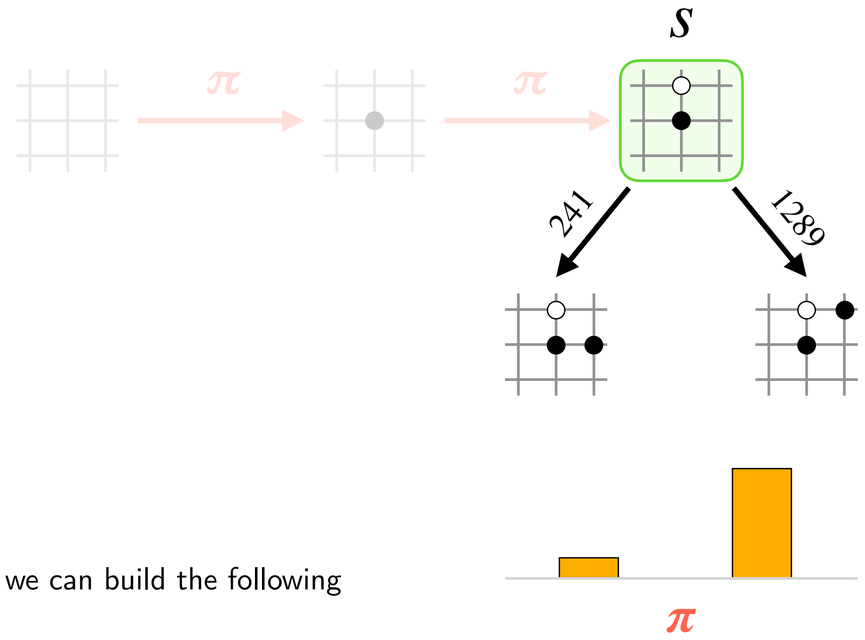


Simplified to demonstrate the flow of AlphaZero MCTS

Visit count distribution



Simplified to demonstrate the flow of AlphaZero MCTS



We can stop at any time.

From the collected visit counts, we can build the following policy reflecting the tree search:

Softmax of the visit counts:

Tree-search policy:

$$\pi(a \mid s) = \frac{\exp N(s,a)^\tau}{\sum_b \exp N(s,b)^\tau}$$

Take the action that maximizes the scoring function:

$$a^* := \arg\max_a U(s,a)$$

Scoring function

$$N(s) := \sum_b N(s,b)$$

$$U(s,a) := Q(s,a) + c \cdot p(s,a) \frac{N(s)}{N(s,a)+1}$$

$$\frac{W(s,a)}{N(s,a)} \qquad f_\theta(s)$$

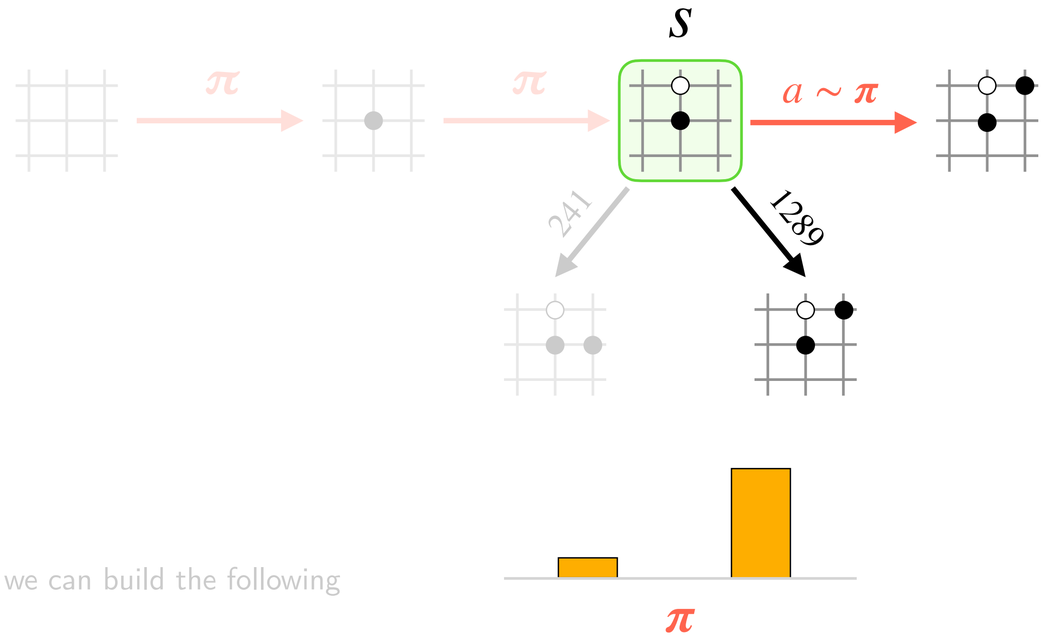
Run many simulations like this.

Each will be slightly different, as each time it uses slightly different tree statistics to traverse the tree.

The **predicted policy *p*** guides the tree search so we only take reasonable trajectories.

The **predicted value *v*** truncates the tree so we don't have to play all the way to the end of the game.

Together, they make the tree search feasible.



We can stop at any time.

From the collected visit counts, we can build the following policy reflecting the tree search:

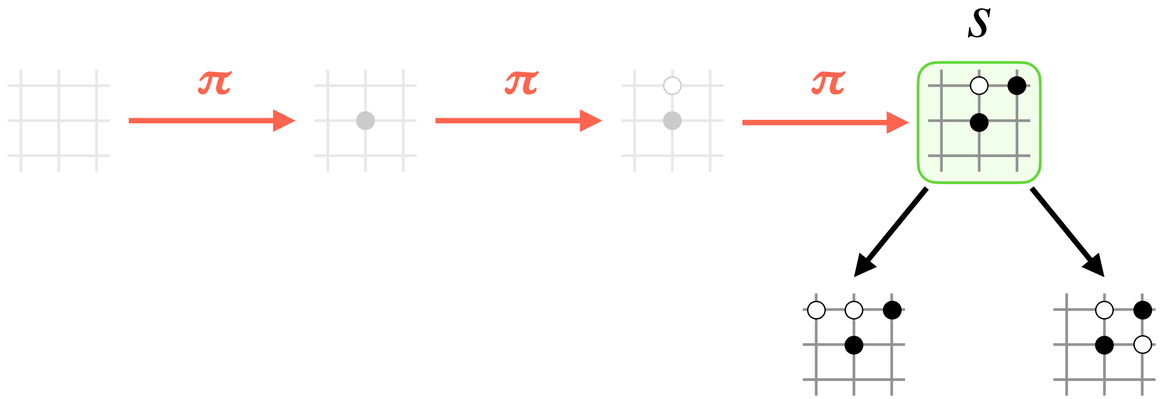
Softmax of the visit counts:

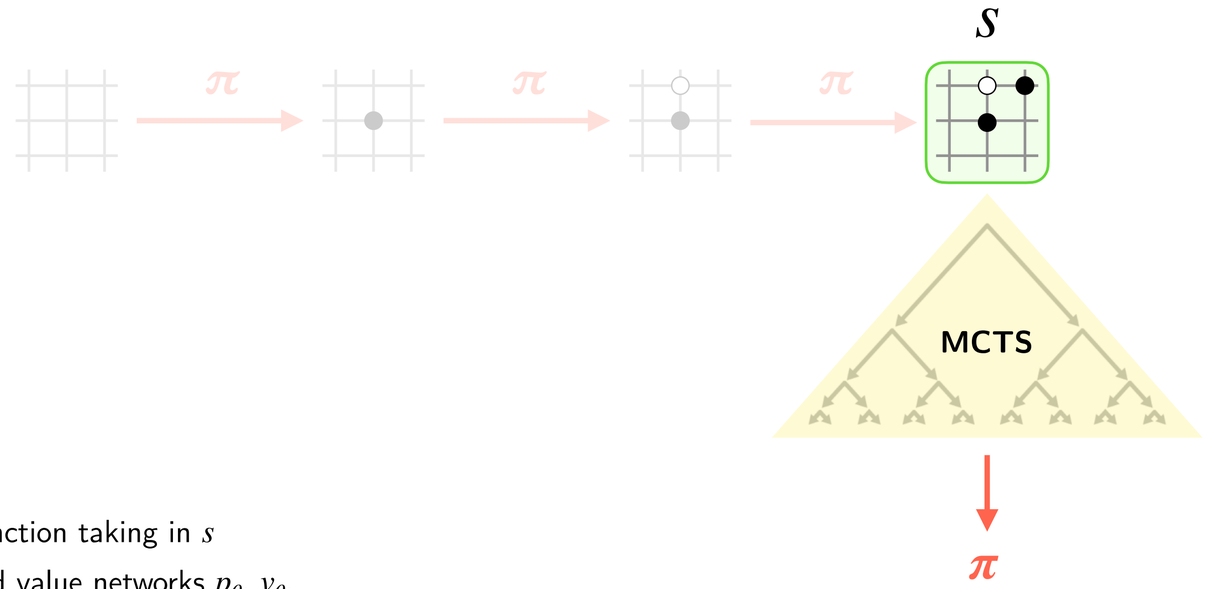
Tree-search policy:

$$\pi(a \mid s) = \frac{\exp N(s,a)^\tau}{\sum_b \exp N(s,b)^\tau}$$

During training, we can then sample from $\boldsymbol{\pi}$.

After training, we can simply always take the action with highest visit count.

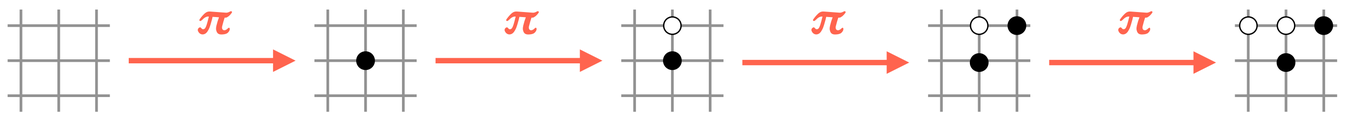

π
π
π
S



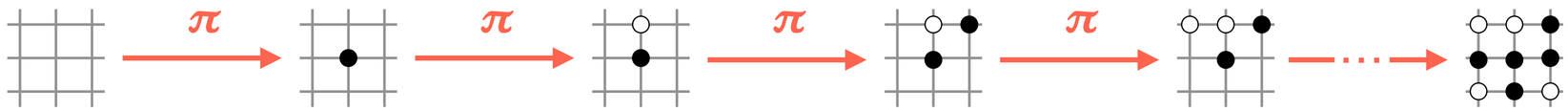
We can think of MCTS as a function taking in s
and parameterized by policy and value networks p_θ, v_θ.
It produces an improved policy $\boldsymbol{\pi}$:

Monte Carlo Tree Search:

$$\boldsymbol{\pi} = \mathrm{MCTS}(s; p_\theta, v_\theta)$$


π
π
π
π

Once this game is finished:



End of game!

z

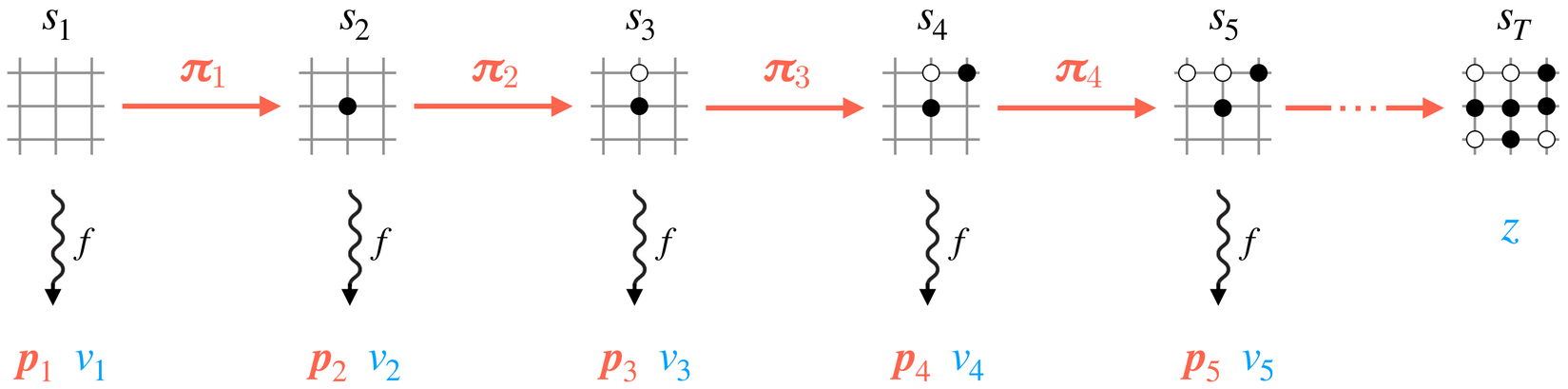
0 if loss

1 if win

This is the only learning signal!

Monte Carlo return

Once this game is finished:



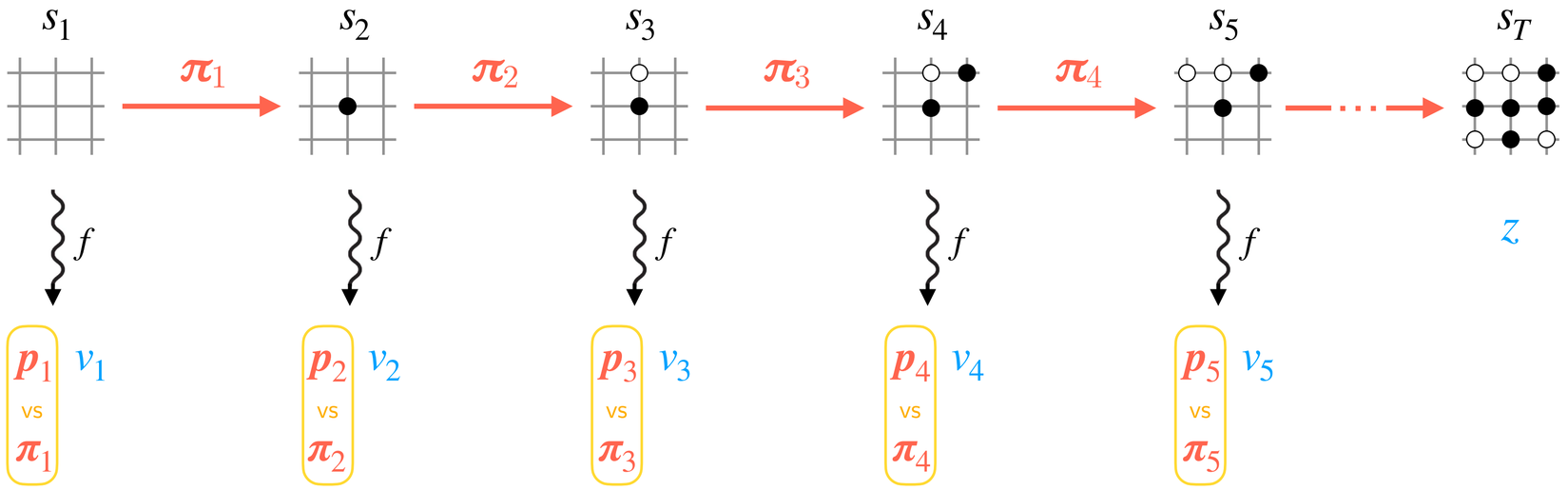
End of game!

0 if loss
1 if win

This is the only learning signal!

Monte Carlo return

Once this game is finished:



End of game!

0 if loss
1 if win

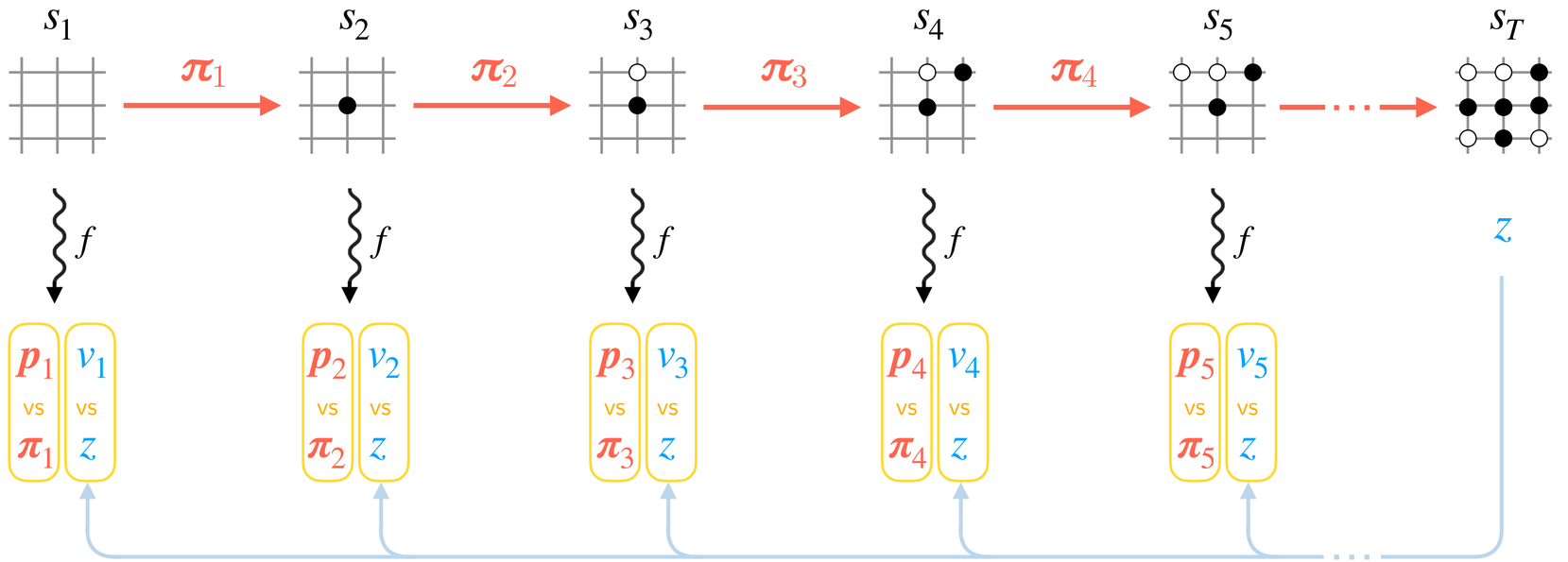
This is the only learning signal!

Monte Carlo return

Make neural network policy more like result of tree search!

$$J(\theta) = \boldsymbol{\pi}^\top \log \boldsymbol{p} + (z - v)^2 + c\|\theta\|^2$$

Once this game is finished:



End of game!

0 if loss

1 if win

This is the only learning signal!

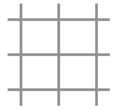
Monte Carlo return

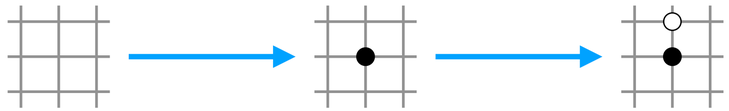
Make neural network policy more like result of tree search!

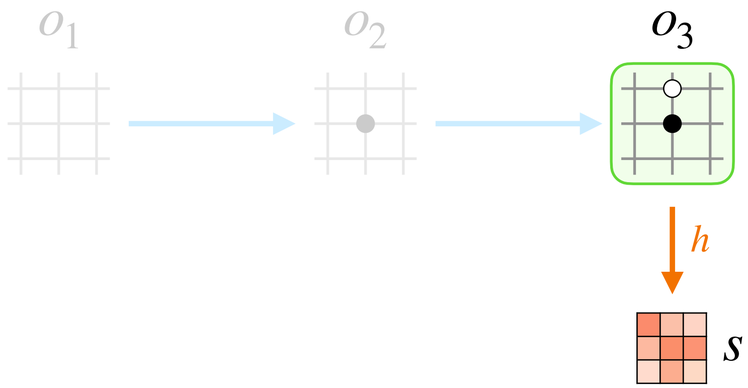
$$J(\theta) = \boldsymbol{\pi}^\top \log \boldsymbol{p} + (z - v)^2 + c\|\theta\|^2$$

2

MuZero

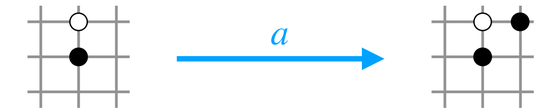


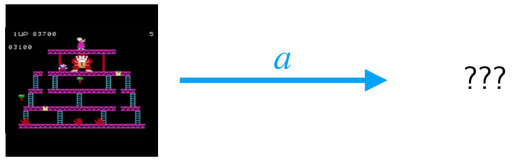




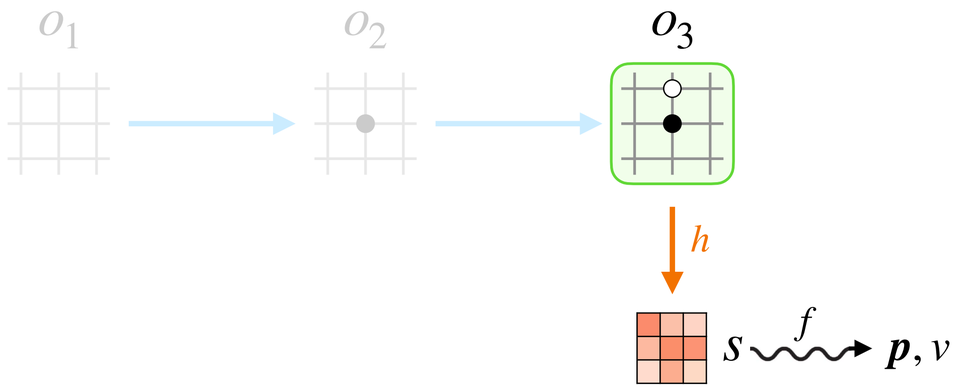
Instead of planning in **observation space**, we first create a representation s, and plan in **representation space**. Think of a neural network extracting all the relevant information from a high-dimensional observation into a low-dimensional abstract representation.

In many environments, we don't know what the next observation will be, so we need to learn the **dynamics** of going from one abstract state to the next.





$$h(o_1, \ldots, o_t) = s_t$$



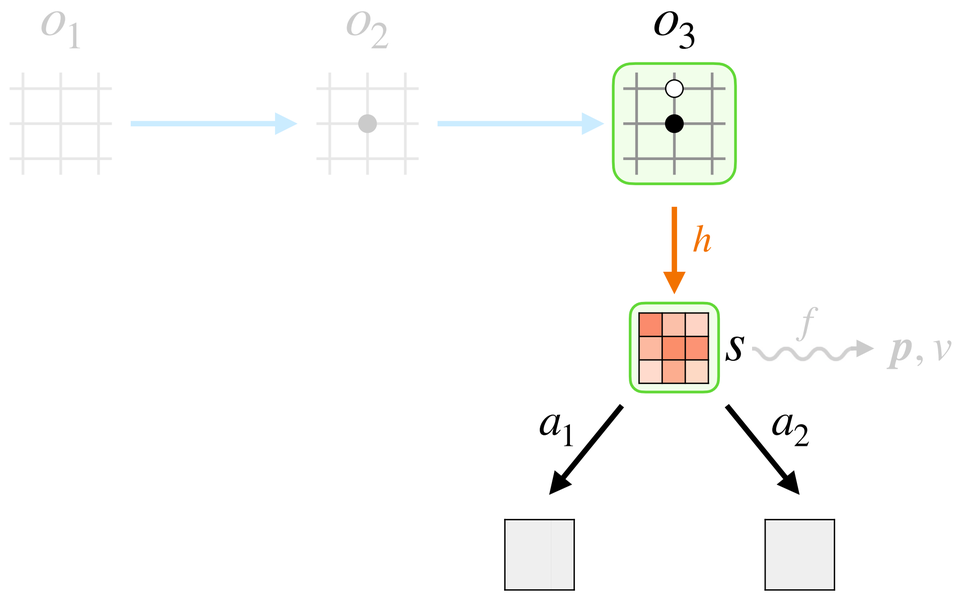
As before, we predict the policy $\boldsymbol{p}$ and value v for each state.

We now store node statistics, rather than edge statistics.
From these statistics, we derive the state value $V(s)$, ???
rather than the action value $Q(s, a)$.
We now predict the **immediate reward** r and add it to the state value to get the action value:

$$Q(s, a) = R(s, a) + \gamma V(s')$$

This is necessary as, unlike chess or Go, there can be immediate rewards, rather than just a single reward of 0 or 1 at the end of the game.

or is V(s) always the prediction?



Take the action that maximizes the scoring function:

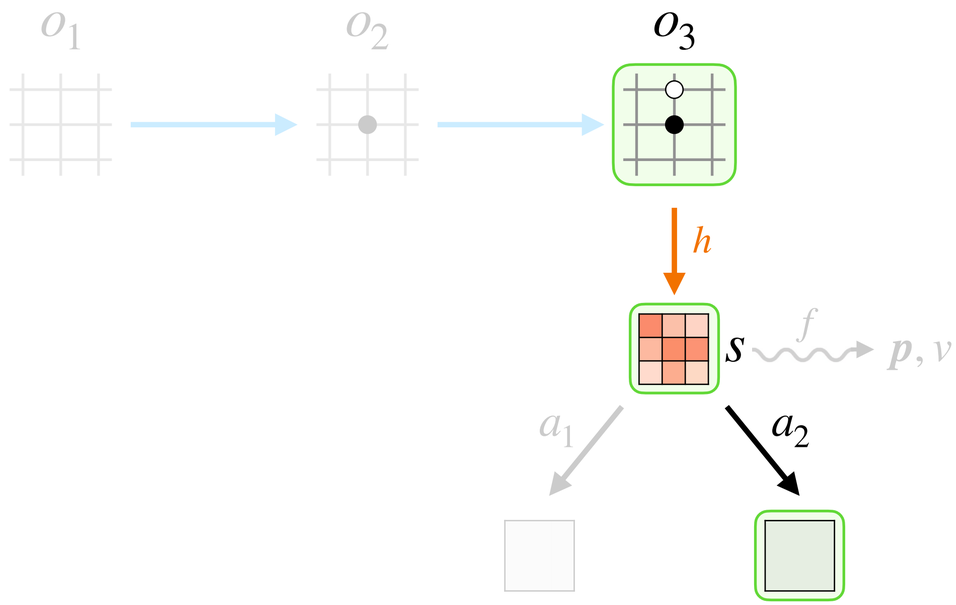
$$a^* := \arg\max_a U(s,a)$$

Scoring function

$$U(s,a) := R(s,a) + \gamma V(s') + p(s,a) \cdot c(N)$$

$Q(s,a)$

$g_\theta(s,a)$ $\quad \frac{W(s')}{N(s')}$ $\quad f_\theta(s)$



Take the action that maximizes the scoring function:

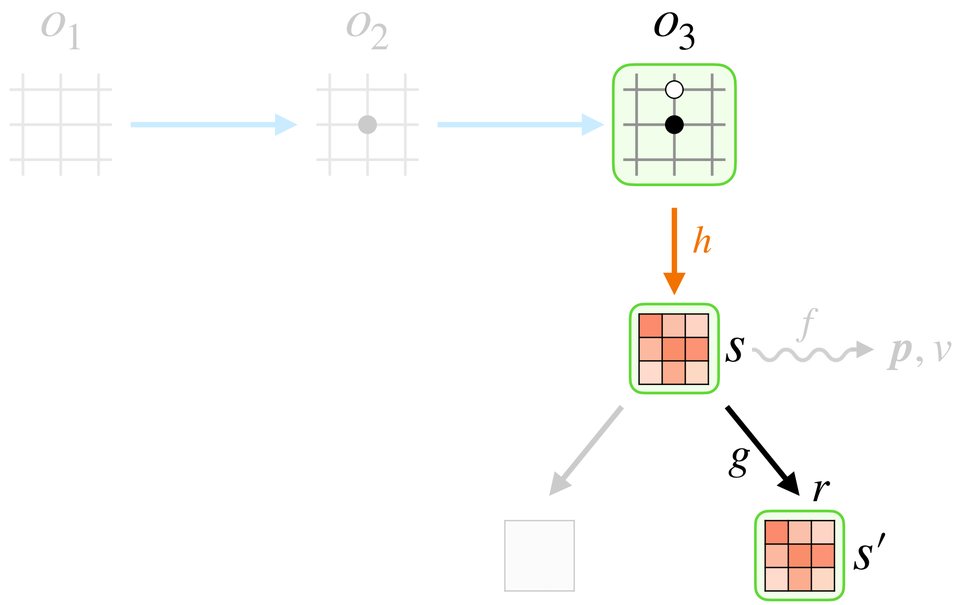
$$a^* := \arg\max_a U(s,a)$$

Scoring function

$$U(s,a) := R(s,a) + \gamma V(s') + p(s,a) \cdot c(N)$$

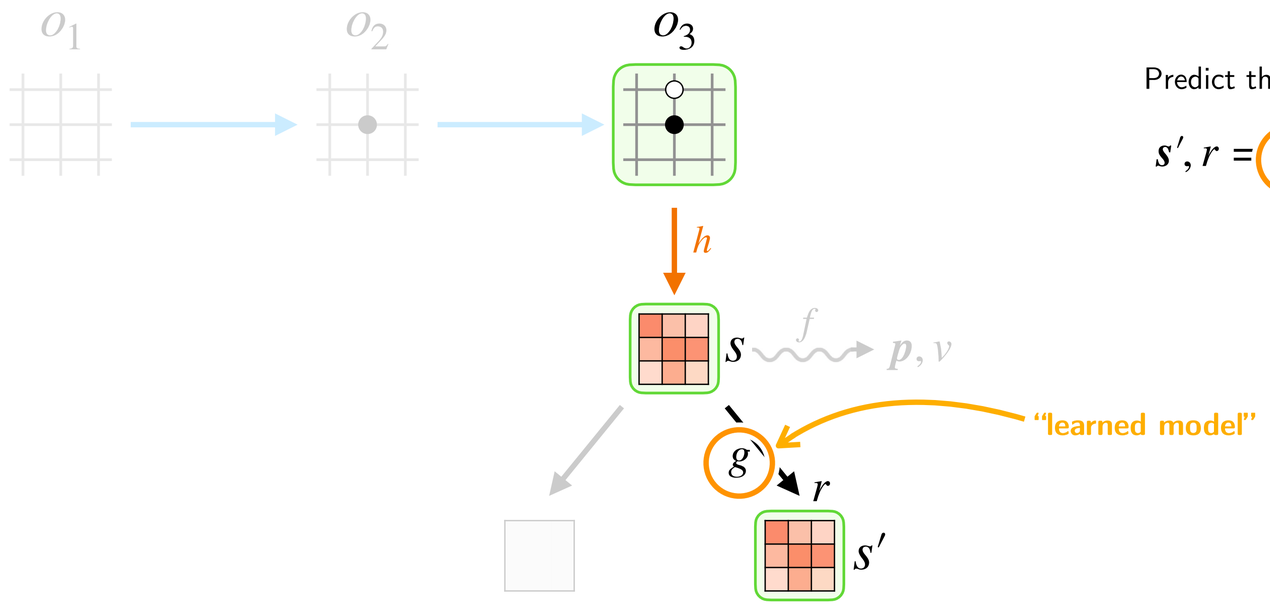
$Q(s,a)$

$g_\theta(s,a)$ $\quad \frac{W(s')}{N(s')}$ $\quad f_\theta(s)$



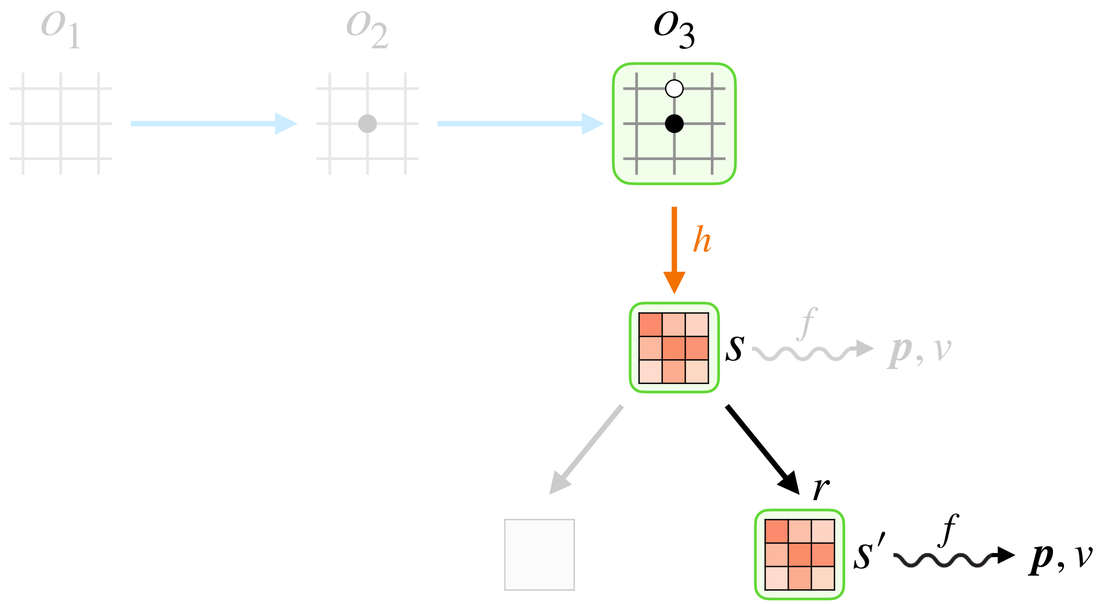
Predict the next abstract state and immediate reward.

$$s', r = g_\theta(s, a)$$



Predict the next abstract state and immediate reward.

$$s', r = g_\theta(s, a)$$



Take the action that maximizes the scoring function:

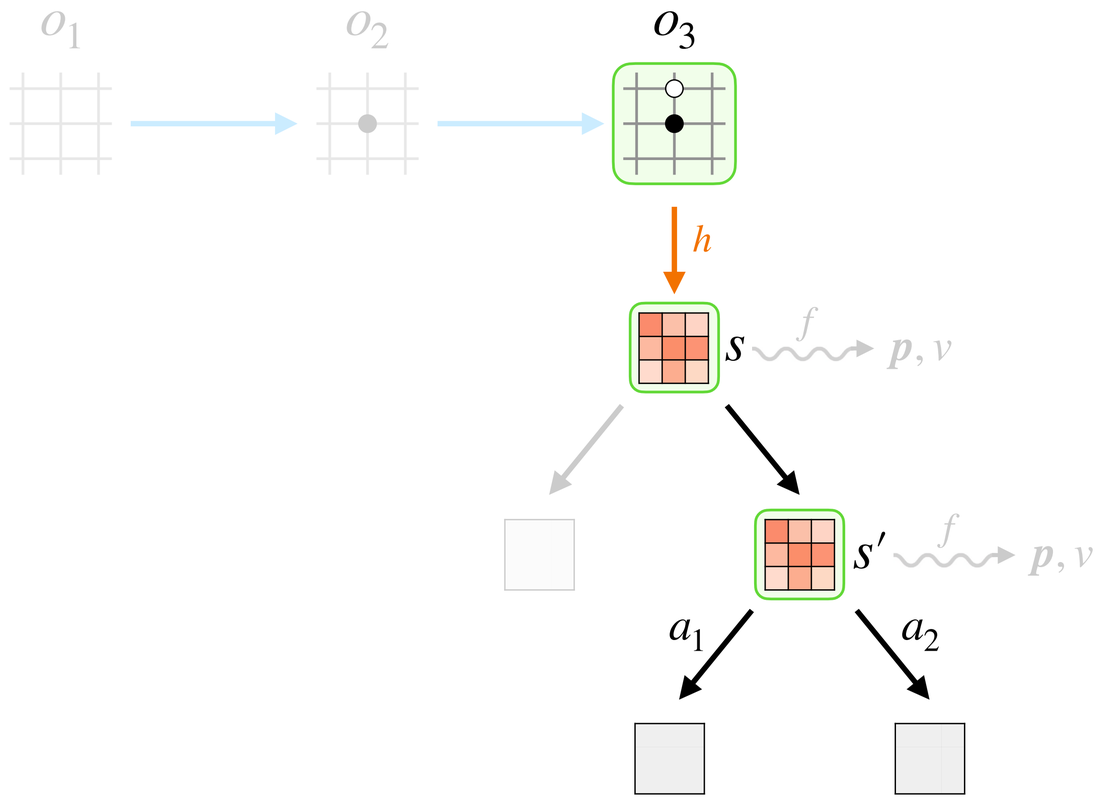
$$a^* := \arg\max_a U(s,a)$$

Scoring function

$$U(s,a) := R(s,a) + \gamma V(s') + p(s,a) \cdot c(N)$$

$Q(s,a)$

$g_\theta(s,a)$ $\quad \frac{W(s')}{N(s')}$ $\quad f_\theta(s)$



Take the action that maximizes the scoring function:

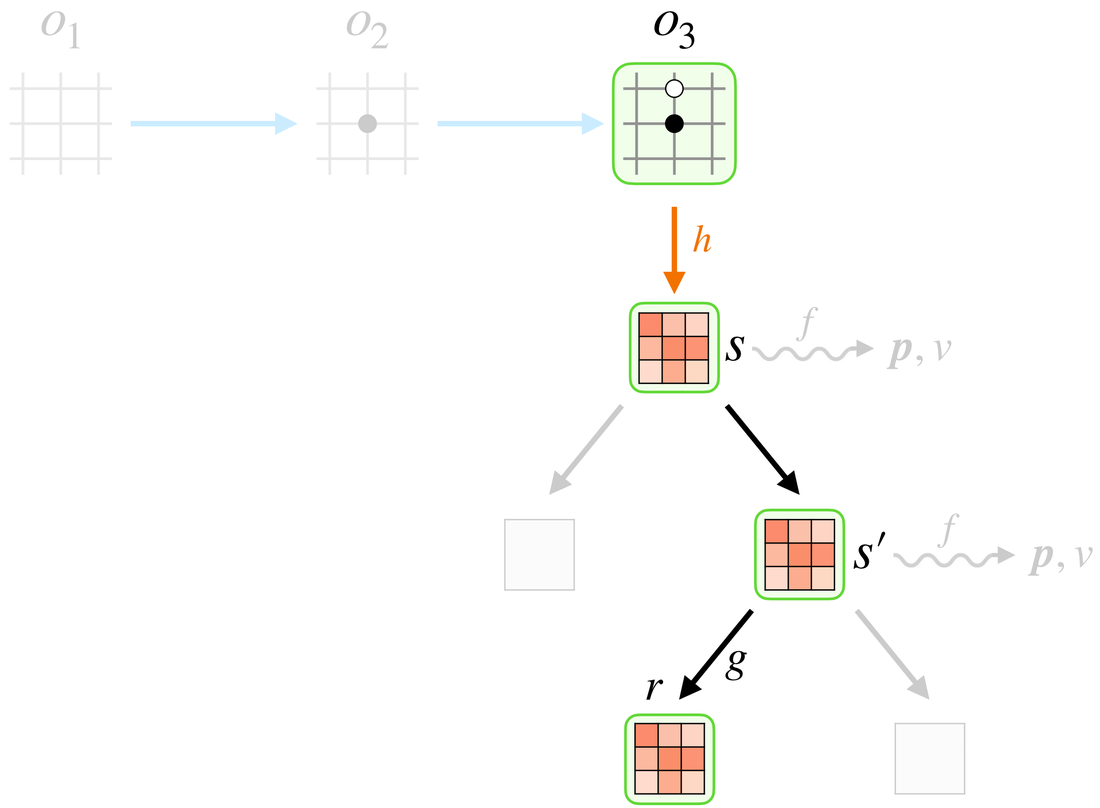
$$a^* := \arg\max_a U(s,a)$$

Scoring function

$$U(s,a) := R(s,a) + \gamma V(s') + p(s,a) \cdot c(N)$$

$Q(s,a)$

$g_\theta(s,a)$ $\quad \frac{W(s')}{N(s')}$ $\quad f_\theta(s)$



Take the action that maximizes the scoring function:

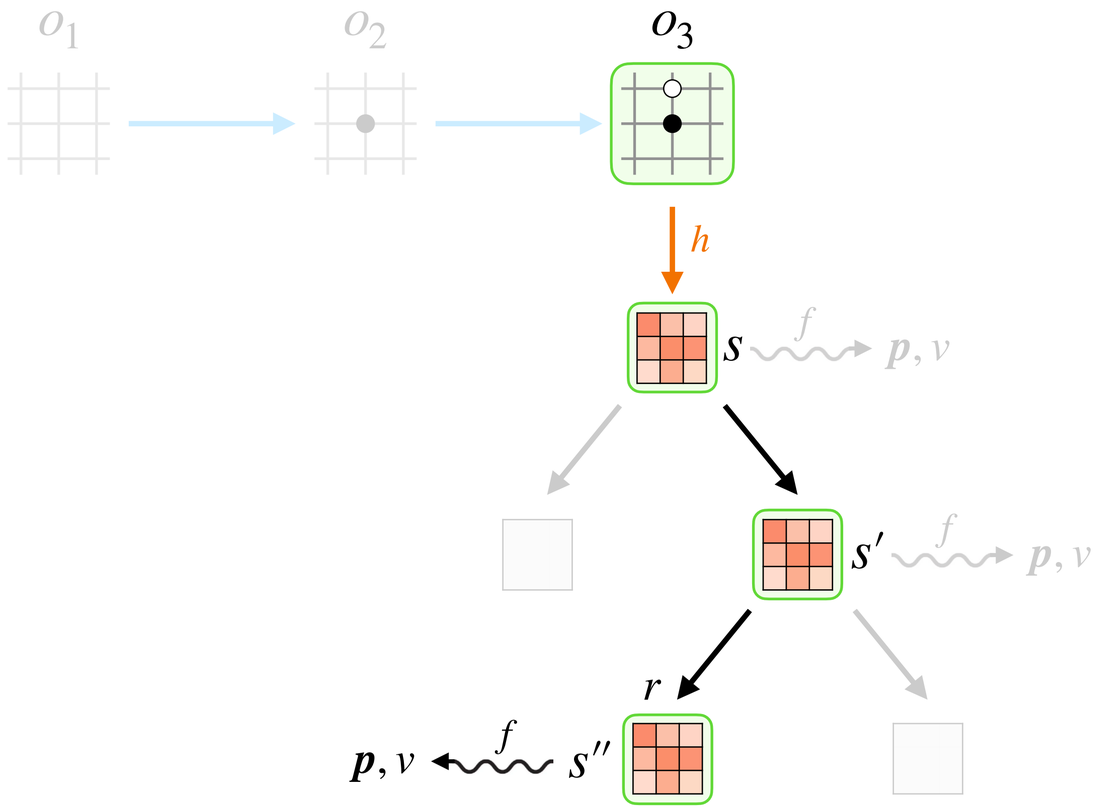
$$a^* := \arg\max_a U(s,a)$$

Scoring function

$$U(s,a) := R(s,a) + \gamma V(s') + p(s,a) \cdot c(N)$$

$Q(s,a)$

$g_\theta(s,a)$ $\quad \frac{W(s')}{N(s')}$ $\quad f_\theta(s)$



Take the action that maximizes the scoring function:

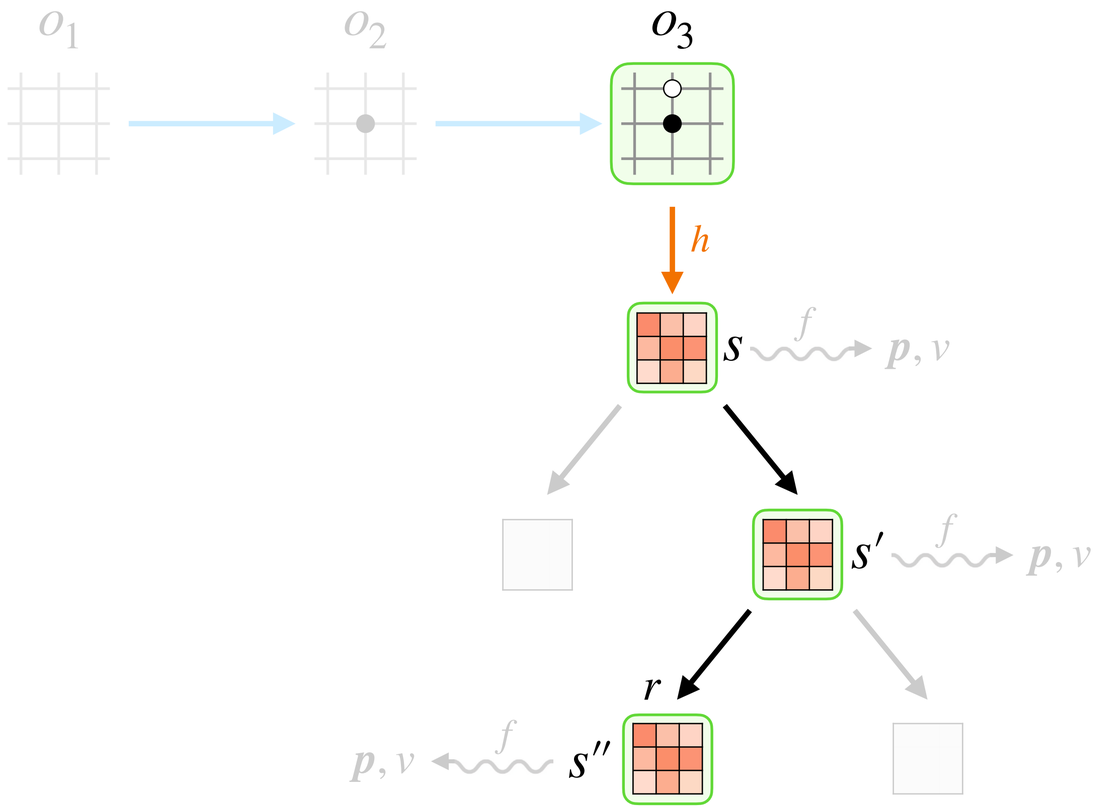
$$a^* := \arg\max_a U(s,a)$$

Scoring function

$$U(s,a) := R(s,a) + \gamma V(s') + p(s,a) \cdot c(N)$$

$Q(s,a)$

$g_\theta(s,a)$ $\quad \frac{W(s')}{N(s')}$ $\quad f_\theta(s)$



Take the action that maximizes the scoring function:

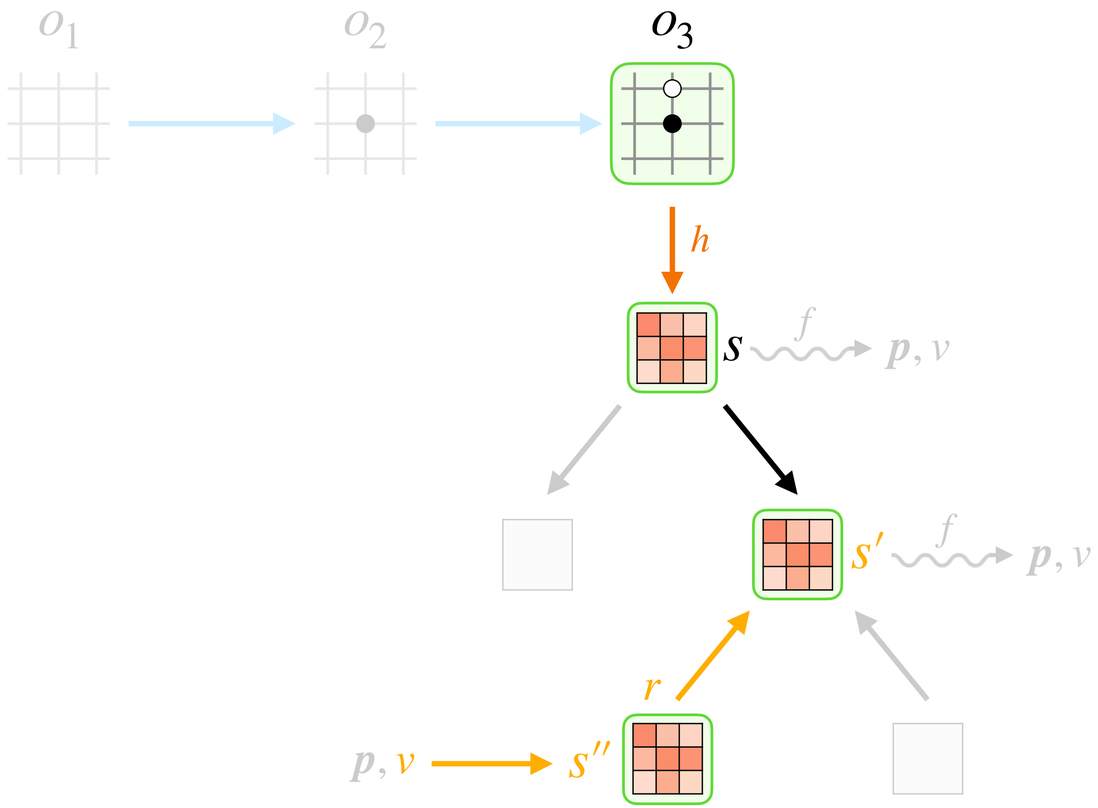
$$a^* := \arg\max_a U(s,a)$$

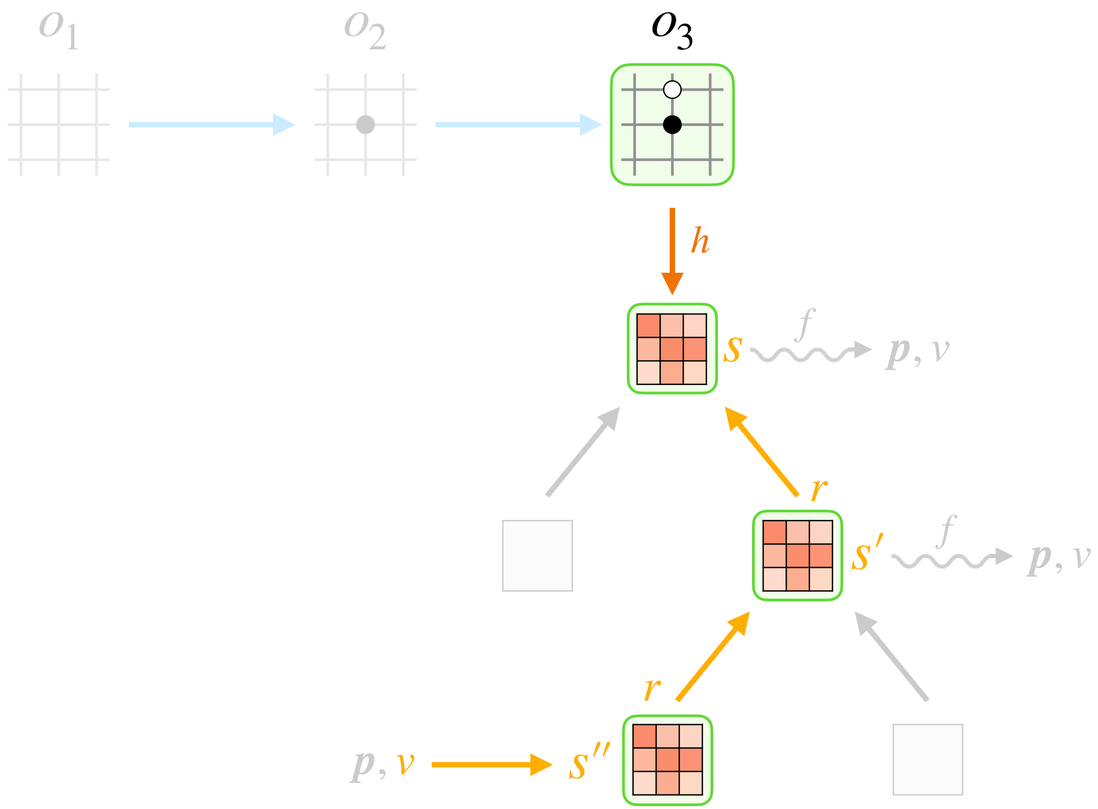
Scoring function

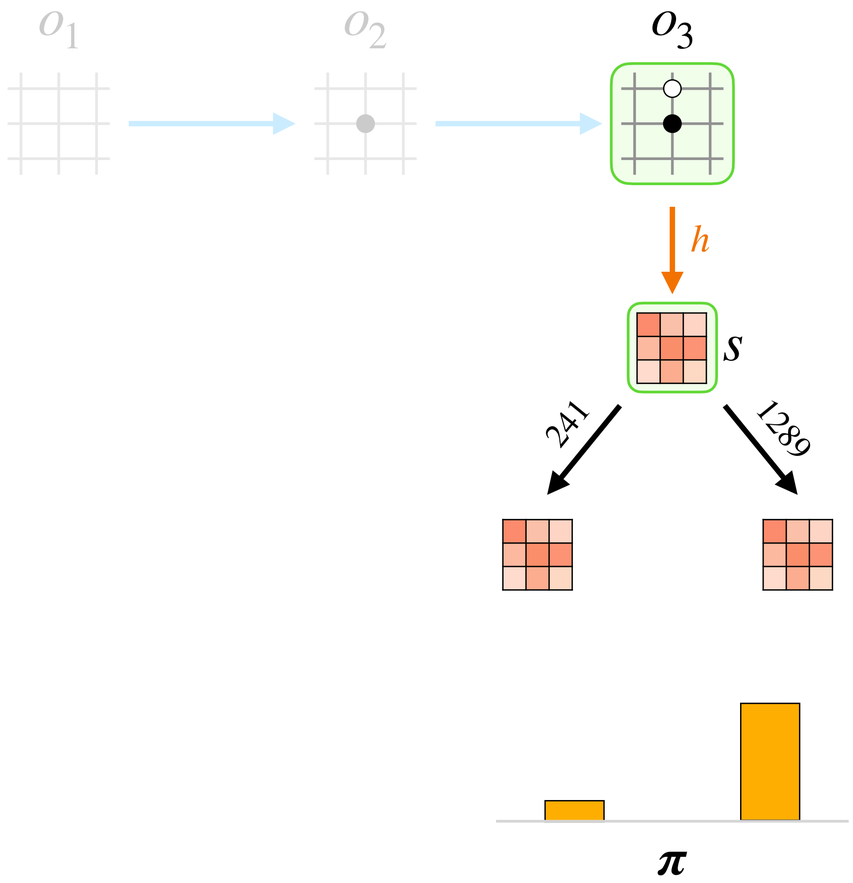
$$U(s,a) := R(s,a) + \gamma V(s') + p(s,a) \cdot c(N)$$

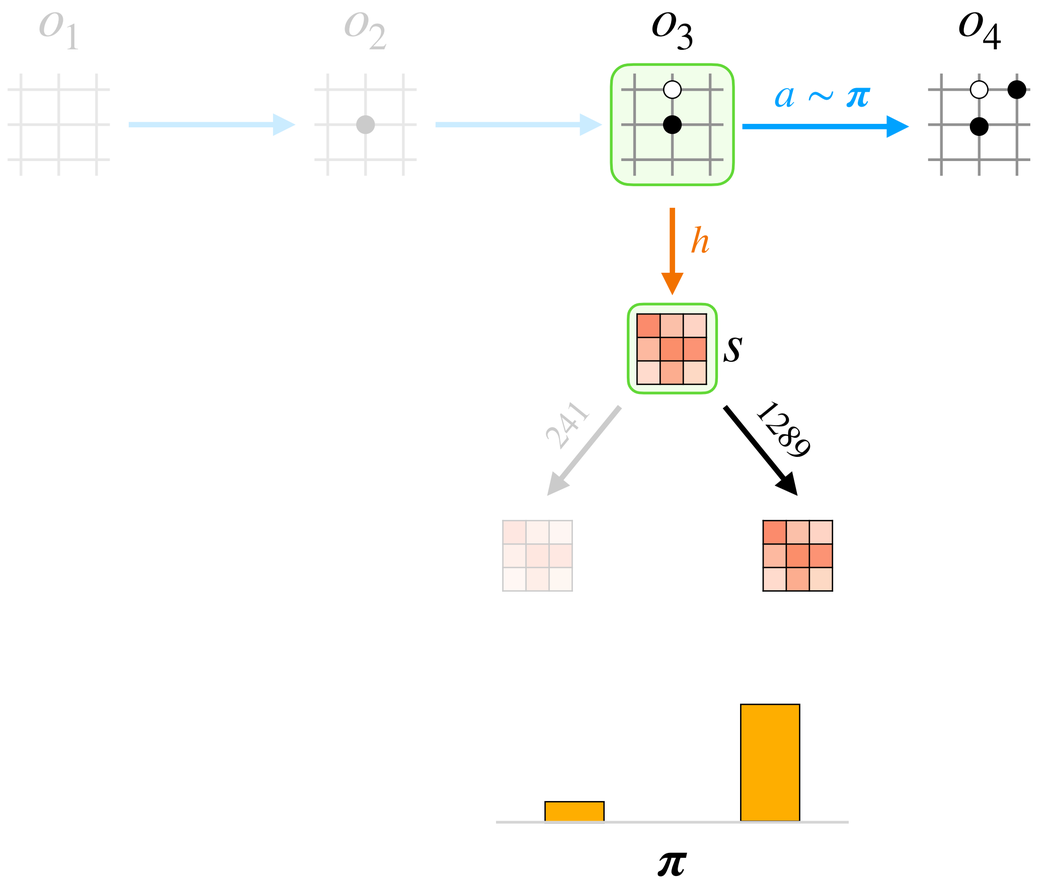
$Q(s,a)$

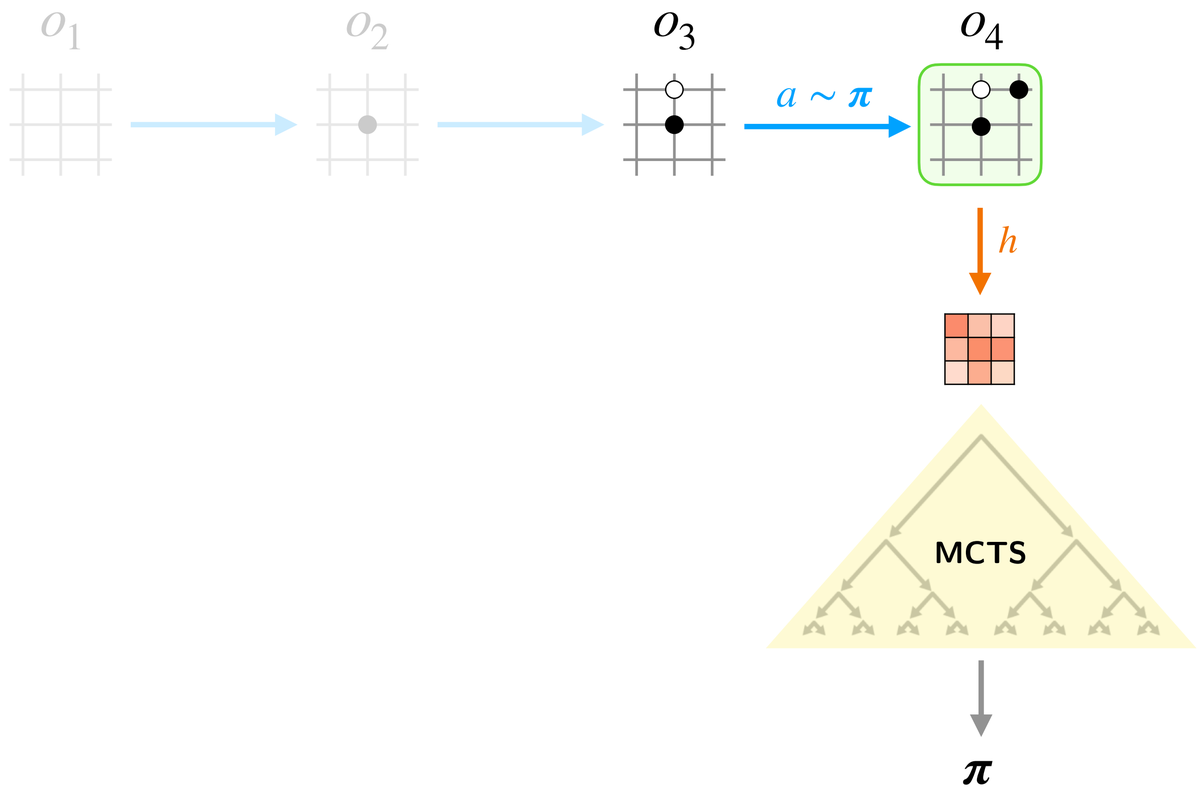
$g_\theta(s,a)$ $\quad \frac{W(s')}{N(s')}$ $\quad f_\theta(s)$

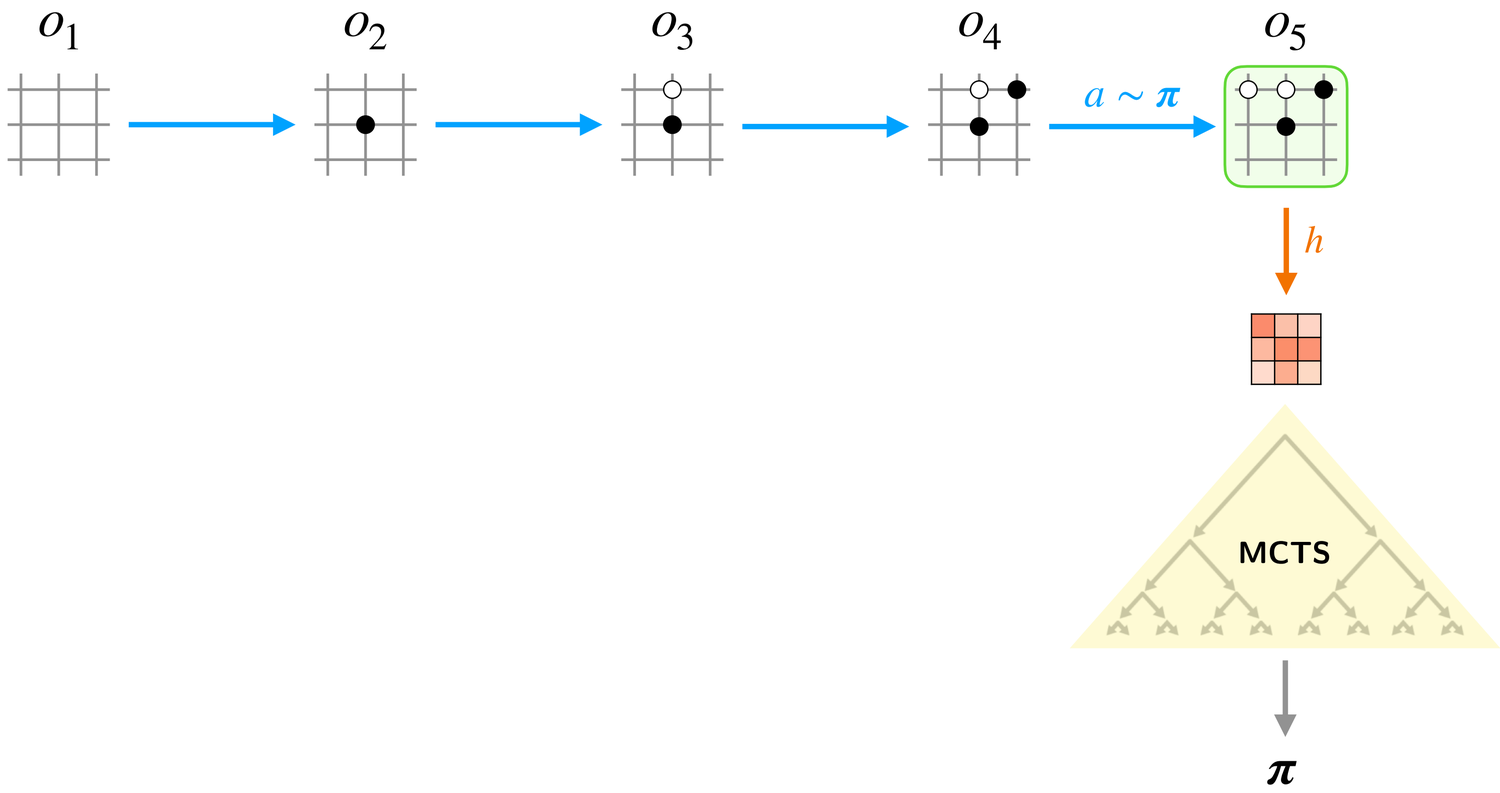

o_1
o_2
o_3
h
s
f
$\boldsymbol{p}, v$
s'
f
$\boldsymbol{p}, v$
r
$\boldsymbol{p}, v$
s''

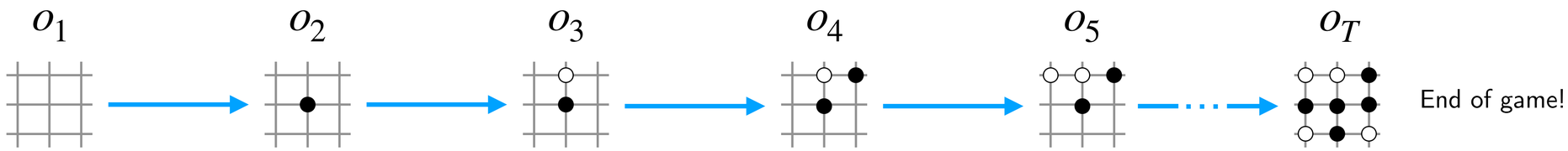

o_1
o_2
o_3
h
s
f
$\boldsymbol{p}, v$
r
s'
f
$\boldsymbol{p}, v$
r
$\boldsymbol{p}, v$
s''


o_1
o_2
o_3
h
s
241
1289
π

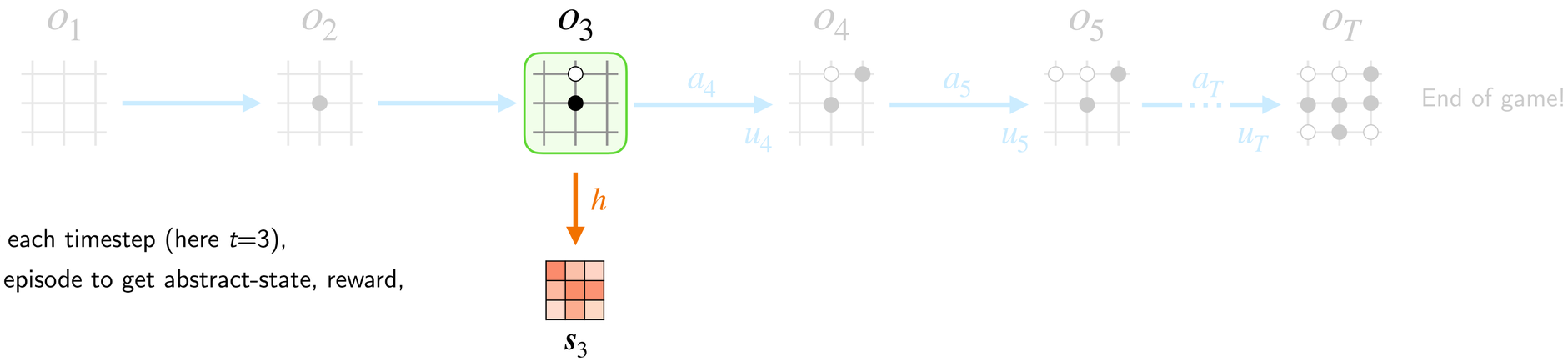

o_1
o_2
o_3
o_4
$a \sim \pi$
h
s
241
1289
π


o_1
o_2
o_3
o_4
$a \sim \pi$
h
MCTS
π

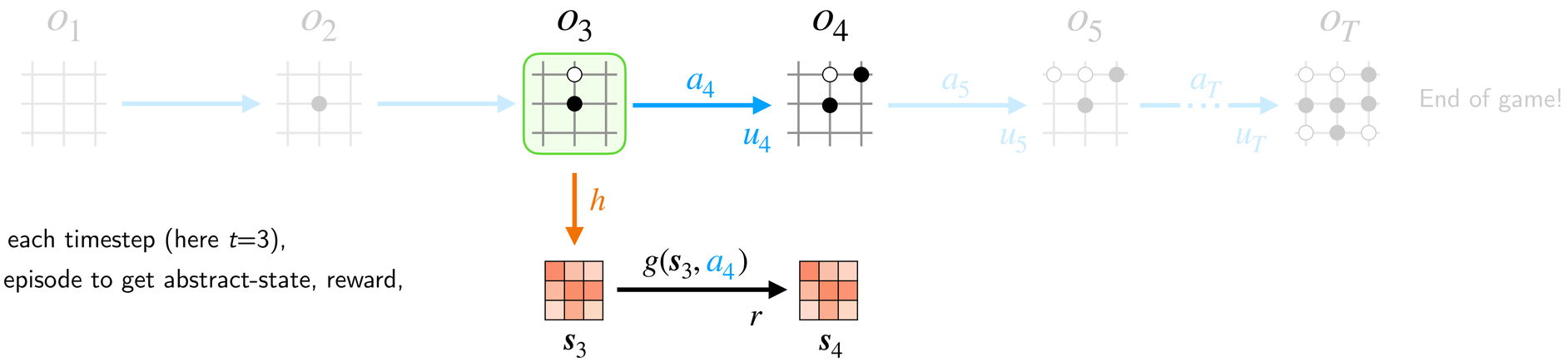

o_1
o_2
o_3
o_4
$a \sim \pi$
o_5
h
MCTS
π



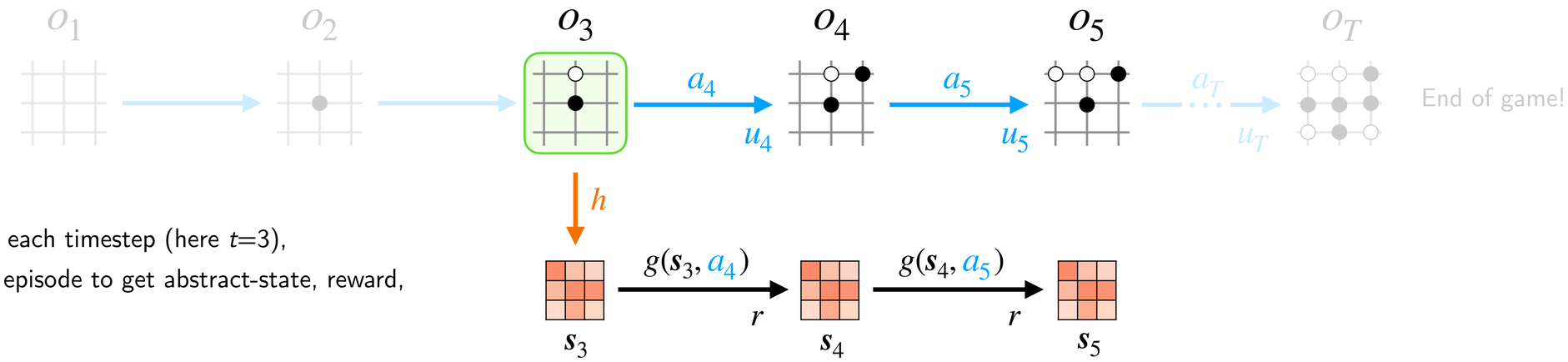
Once we have the full episode, for each timestep (here t=3),
use the action taken in the actual episode to get abstract-state, reward,
policy, and value predictions.



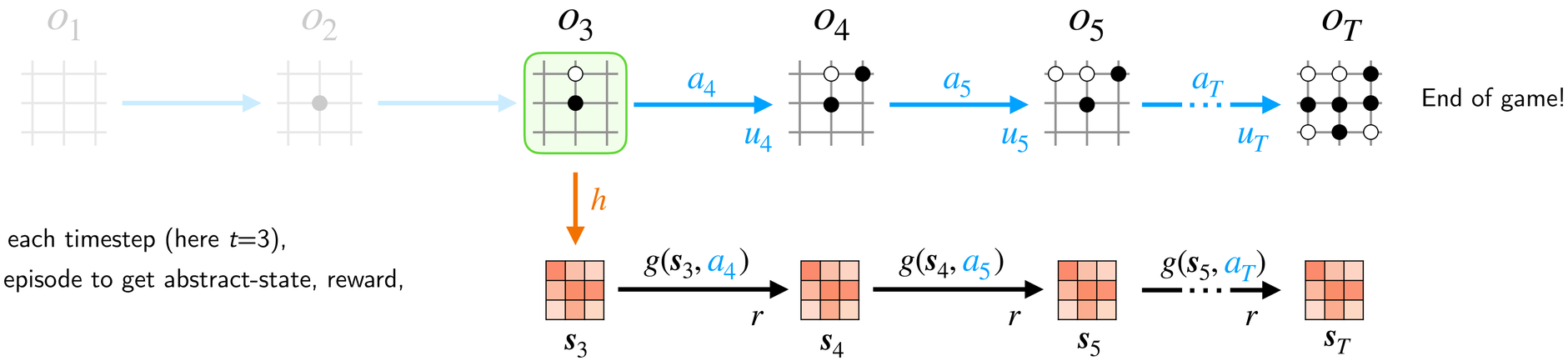
Once we have the full episode, for each timestep (here t=3),
use the action taken in the actual episode to get abstract-state, reward,
policy, and value predictions.



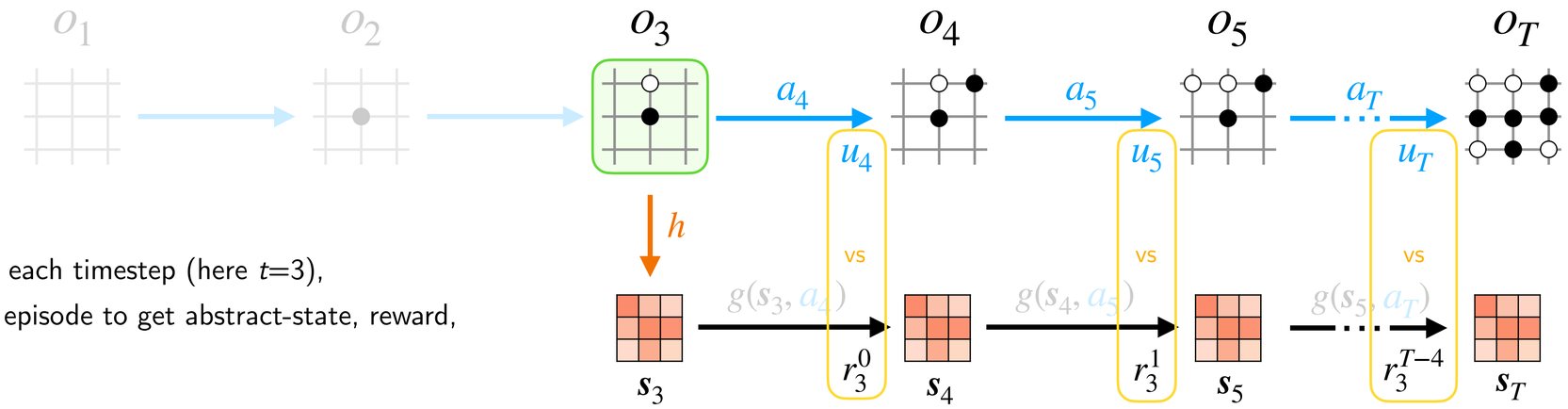
Once we have the full episode, for each timestep (here t=3),
use the action taken in the actual episode to get abstract-state, reward,
policy, and value predictions.



Once we have the full episode, for each timestep (here t=3),
use the action taken in the actual episode to get abstract-state, reward,
policy, and value predictions.



Once we have the full episode, for each timestep (here t=3),
use the action taken in the actual episode to get abstract-state, reward,
policy, and value predictions.



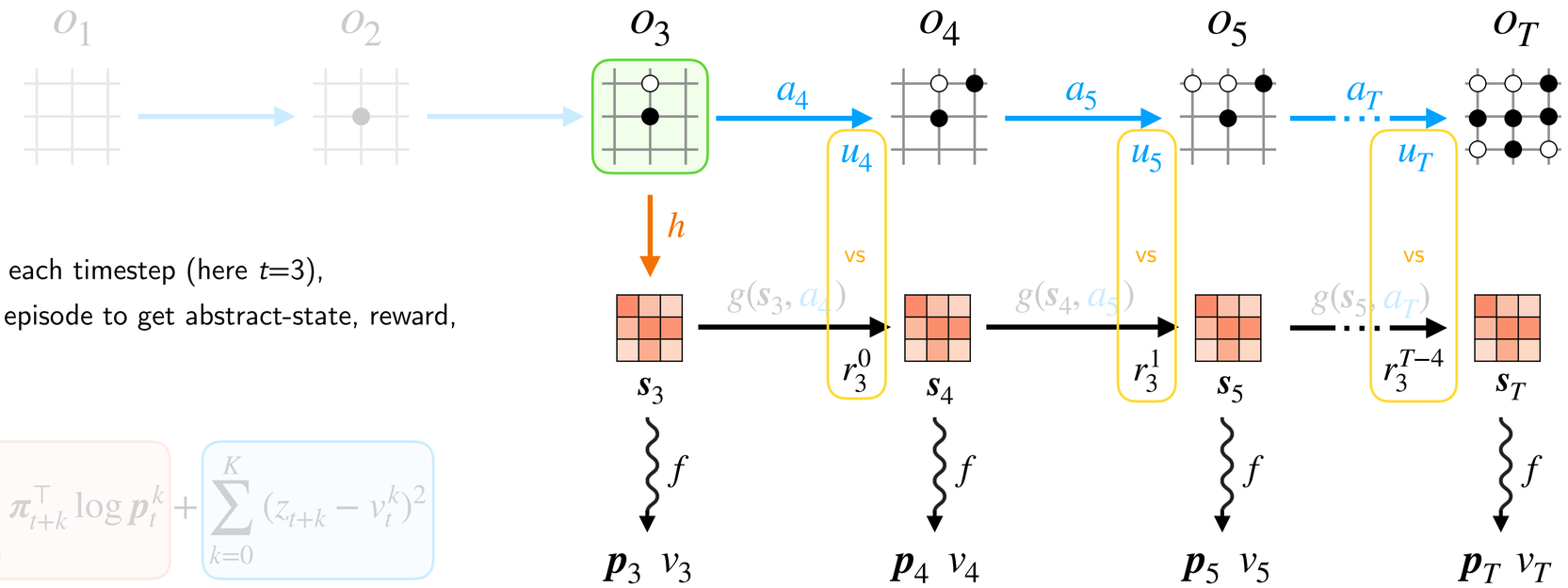
End of game!

Once we have the full episode, for each timestep (here t=3),
use the action taken in the actual episode to get abstract-state, reward,
policy, and value predictions.

$$J(\theta) = \sum_{k=1}^{K} (u_{t+k} - r_t^k)^2 + \sum_{k=0}^{K} \pi_{t+k}^\top \log p_t^k + \sum_{k=0}^{K} (z_{t+k} - v_t^k)^2$$

$$+ c\|\theta\|^2$$

Same as in AlphaGo, but now take the sum over the imagined trajectory.



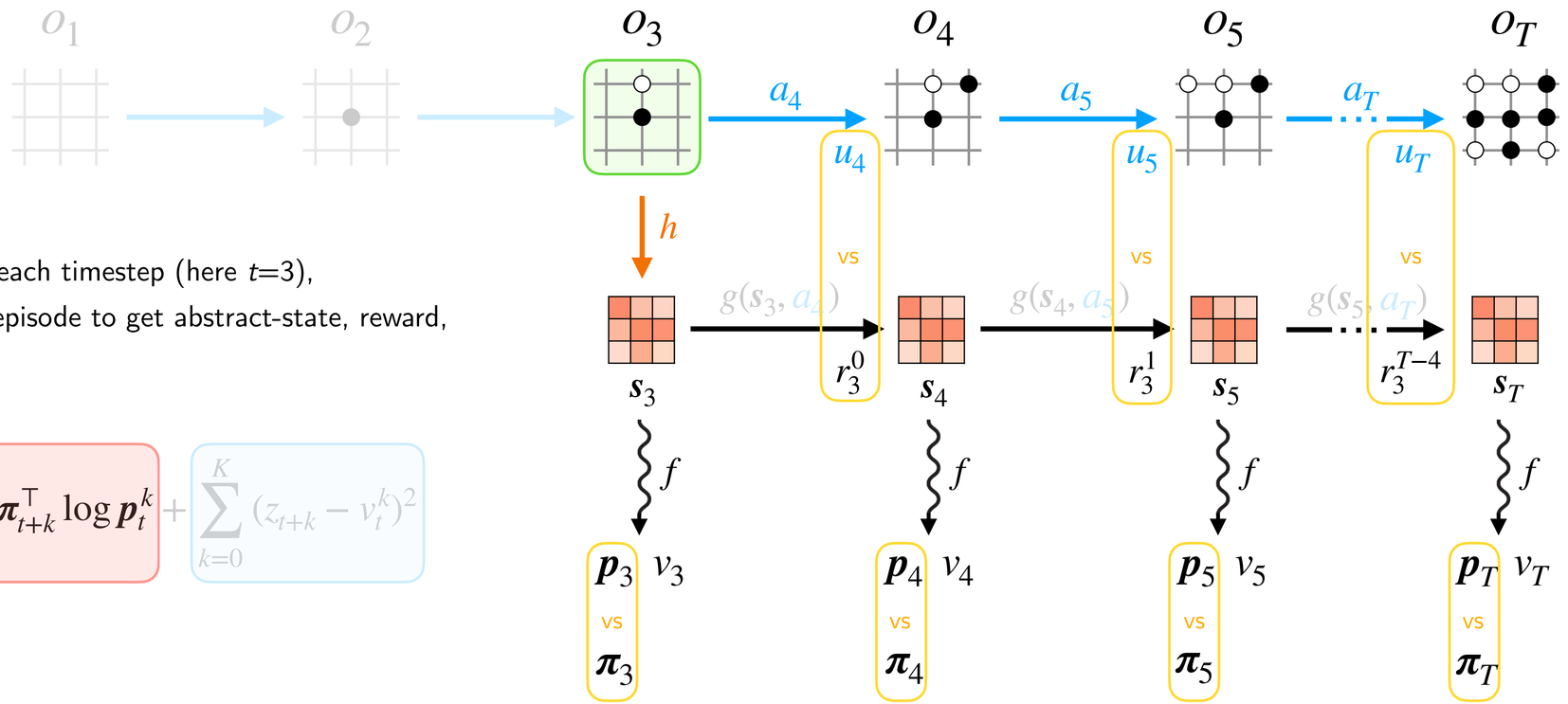
End of game!

Once we have the full episode, for each timestep (here t=3),
use the action taken in the actual episode to get abstract-state, reward,
policy, and value predictions.

$$J(\theta) = \sum_{k=1}^{K} (u_{t+k} - r_t^k)^2 + \sum_{k=0}^{K} \pi_{t+k}^\top \log p_t^k + \sum_{k=0}^{K} (z_{t+k} - v_t^k)^2$$

$$+ c\|\theta\|^2$$

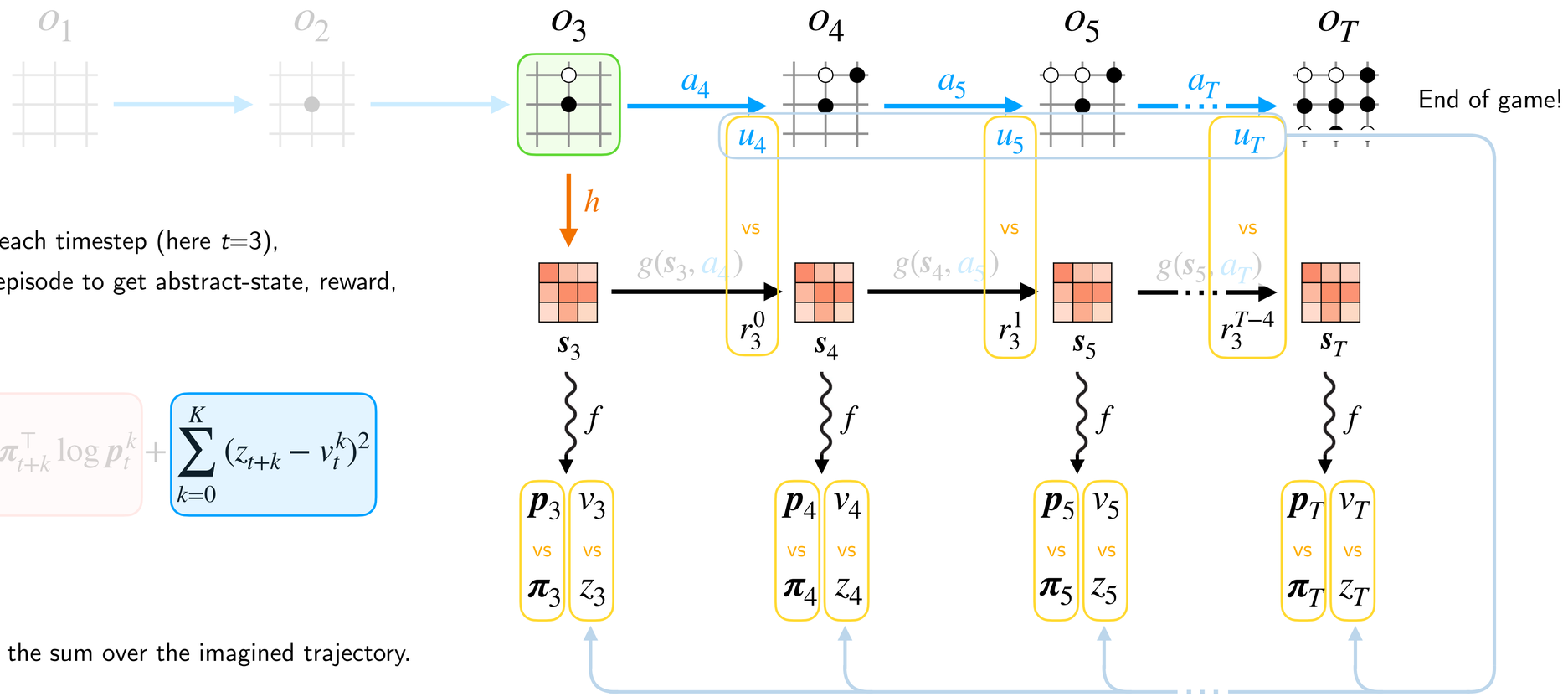
Same as in AlphaGo, but now take the sum over the imagined trajectory.



Once we have the full episode, for each timestep (here t=3),
use the action taken in the actual episode to get abstract-state, reward,
policy, and value predictions.

$$J(\theta) = \sum_{k=1}^{K} (u_{t+k} - r_t^k)^2 + \sum_{k=0}^{K} \boldsymbol{\pi}_{t+k}^\top \log \boldsymbol{p}_t^k + \sum_{k=0}^{K} (z_{t+k} - v_t^k)^2 + c\|\theta\|^2$$

Same as in AlphaGo, but now take the sum over the imagined trajectory.



Once we have the full episode, for each timestep (here t=3),
use the action taken in the actual episode to get abstract-state, reward,
policy, and value predictions.

$$J(\theta) = \sum_{k=1}^{K} (u_{t+k} - r_t^k)^2 + \sum_{k=0}^{K} \pi_{t+k}^\top \log p_t^k + \sum_{k=0}^{K} (z_{t+k} - v_t^k)^2$$

$$+ c\|\theta\|^2$$

Same as in AlphaGo, but now take the sum over the imagined trajectory.

***n*-step return (TD Learning):**

$$z_t = u_{t+1} + \gamma u_{t+2} + \ldots + \gamma^{n-1} u_{t+n} + \gamma^n v_{t+n}$$

3

EfficientZero

MuZero is ***sample inefficient***.

Even when re-using old trajectories, we can do more to make better use of our data.

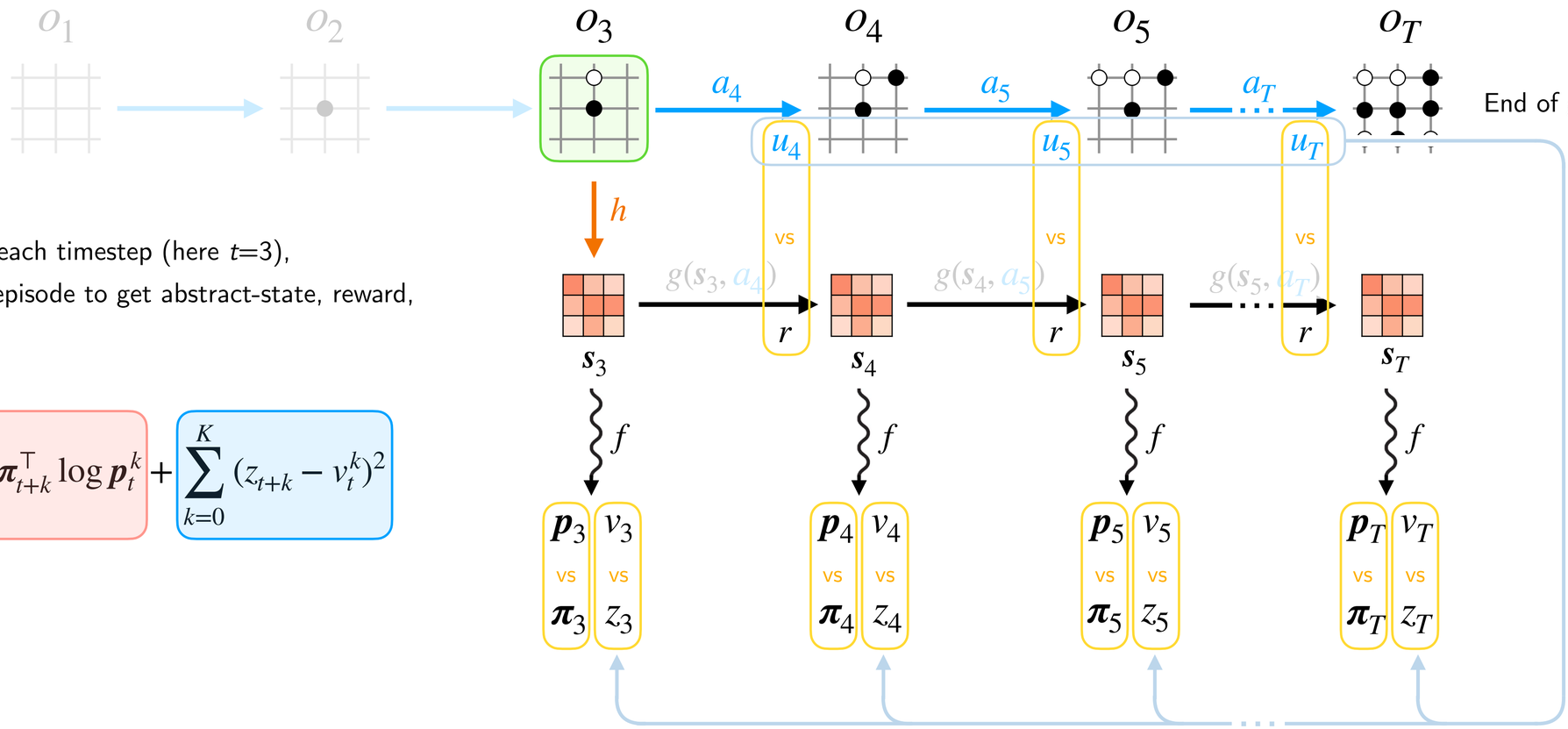
What the authors of ***EfficientZero*** propose:

(1) Self-supervised learning of state representation

(2) Prediction of the "value prefix"

(3) Off-policy correction

(1) Self-supervised learning of state representation



Once we have the full episode, for each timestep (here t=3),
use the action taken in the actual episode to get abstract-state, reward,
policy, and value predictions.

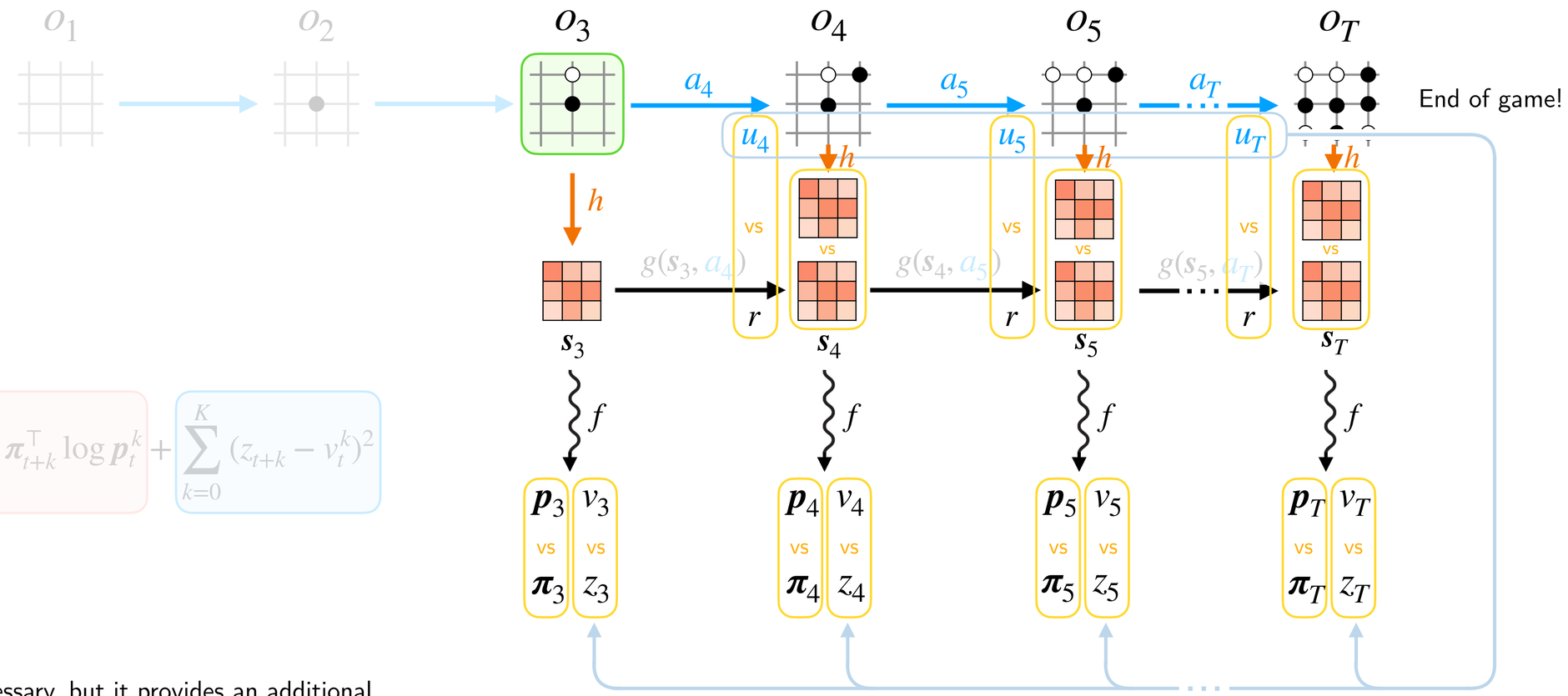
$$J(\theta) = \sum_{k=1}^{K} (u_{t+k} - r_t^k)^2 + \sum_{k=0}^{K} \boldsymbol{\pi}_{t+k}^{\top} \log \boldsymbol{p}_t^k + \sum_{k=0}^{K} (z_{t+k} - v_t^k)^2$$

$$+c\|\theta\|^2$$

n-step return (TD Learning):

$$z_t = u_{t+1} + \gamma u_{t+2} + \ldots + \gamma^{n-1} u_{t+n} + \gamma^n v_{t+n}$$

(1) Self-supervised learning of state representation



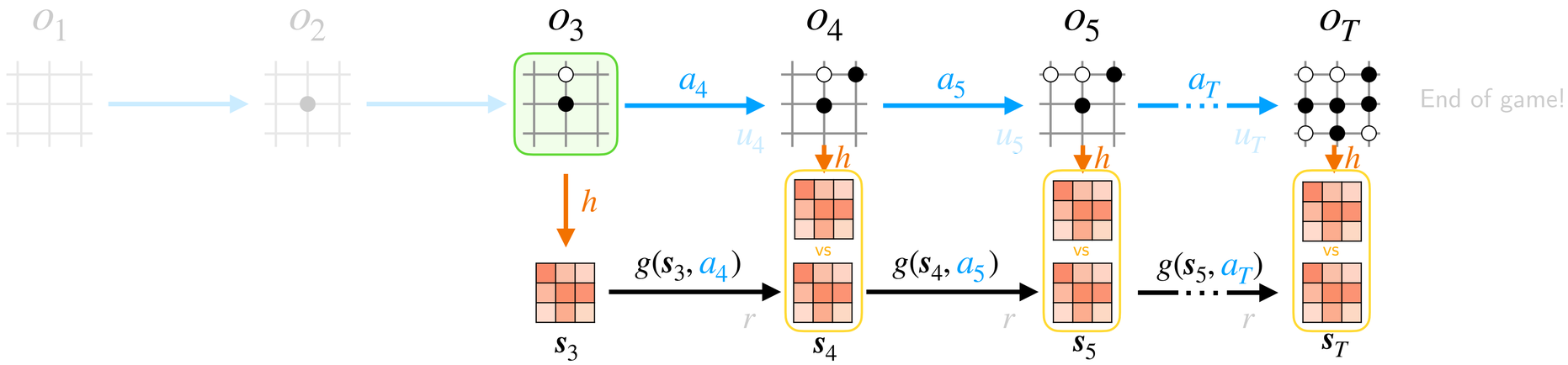
$$J(\theta) = \sum_{k=1}^{K} (u_{t+k} - r_t^k)^2 + \sum_{k=0}^{K} \boldsymbol{\pi}_{t+k}^\top \log \boldsymbol{p}_t^k + \sum_{k=0}^{K} (z_{t+k} - v_t^k)^2 + c\|\theta\|^2$$

The representation loss is not necessary, but it provides an additional learning signal and so it can help with training, especially if the data is limited. Think of it as an **auxiliary loss**.

n-step return (TD Learning):

$$z_t = u_{t+1} + \gamma u_{t+2} + \ldots + \gamma^{n-1} u_{t+n} + \gamma^n v_{t+n}$$

(1) Self-supervised learning of state representation

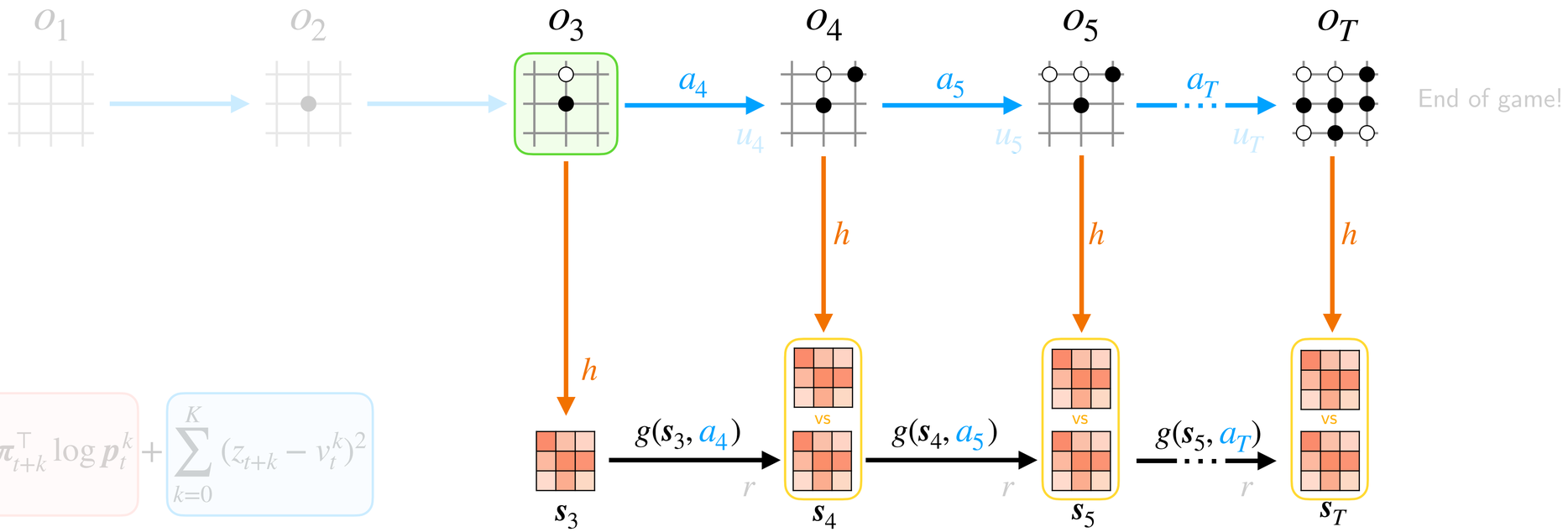


$$J(\theta) = \sum_{k=1}^{K} (u_{t+k} - r_t^k)^2 + \sum_{k=0}^{K} \boldsymbol{\pi}_{t+k}^\top \log \boldsymbol{p}_t^k + \sum_{k=0}^{K} (z_{t+k} - v_t^k)^2$$

$$+ c\|\theta\|^2$$

The representation loss is not necessary, but it provides an additional learning signal and so it can help with training, especially if the data is limited. Think of it as an **auxiliary loss**.

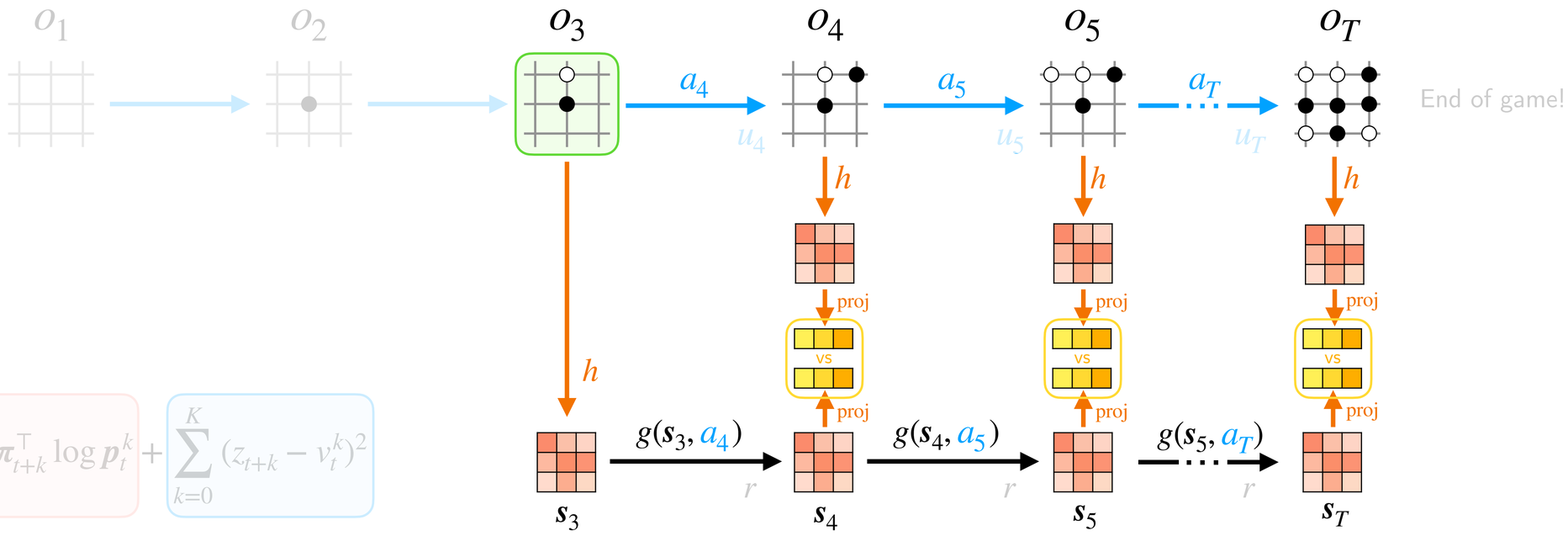
(1) Self-supervised learning of state representation



$$J(\theta) = \sum_{k=1}^{K} (u_{t+k} - r_t^k)^2 + \sum_{k=0}^{K} \boldsymbol{\pi}_{t+k}^\top \log \boldsymbol{p}_t^k + \sum_{k=0}^{K} (z_{t+k} - v_t^k)^2 + c\|\theta\|^2$$

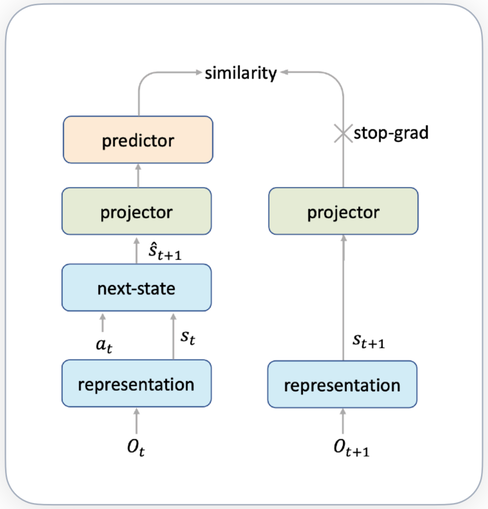
The representation loss is not necessary, but it provides an additional learning signal and so it can help with training, especially if the data is limited. Think of it as an **auxiliary loss**.

(1) Self-supervised learning of state representation

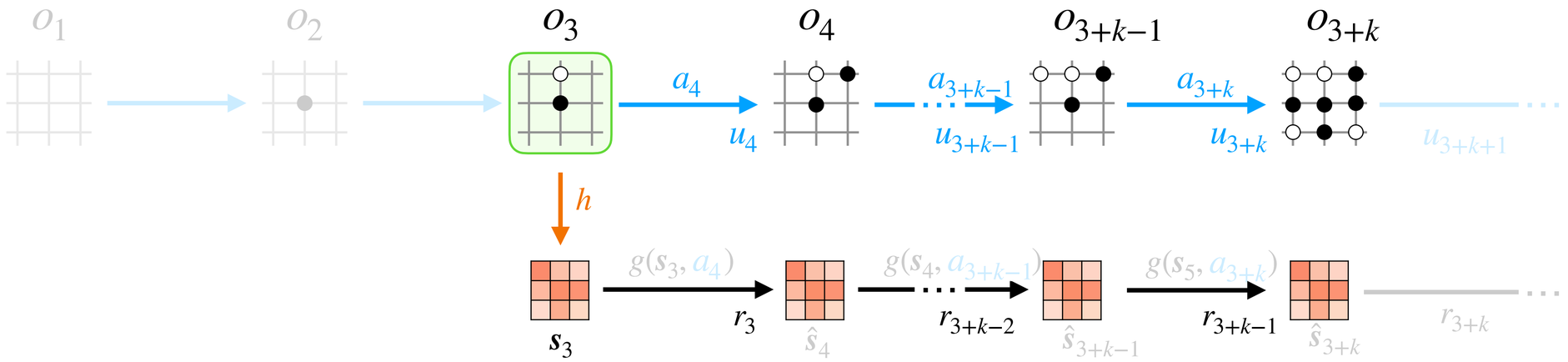


$$J(\theta) = \sum_{k=1}^{K} (u_{t+k} - r_t^k)^2 + \sum_{k=0}^{K} \boldsymbol{\pi}_{t+k}^\top \log \boldsymbol{p}_t^k + \sum_{k=0}^{K} (z_{t+k} - v_t^k)^2 + c\|\theta\|^2$$

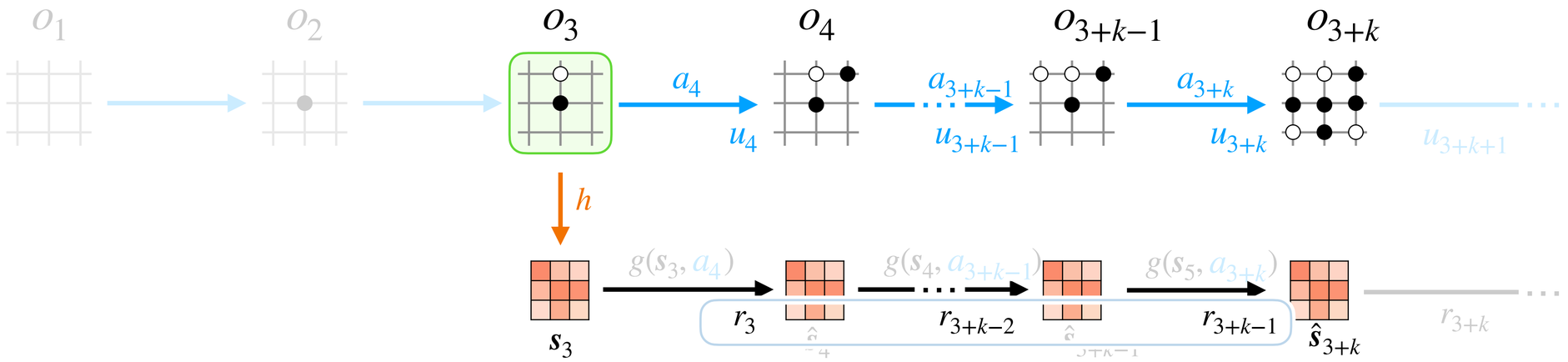
The representation loss is not necessary, but it provides an additional learning signal and so it can help with training, especially if the data is limited. Think of it as an **auxiliary loss**.



(2) Prediction of the "value prefix"



(2) Prediction of the "value prefix"



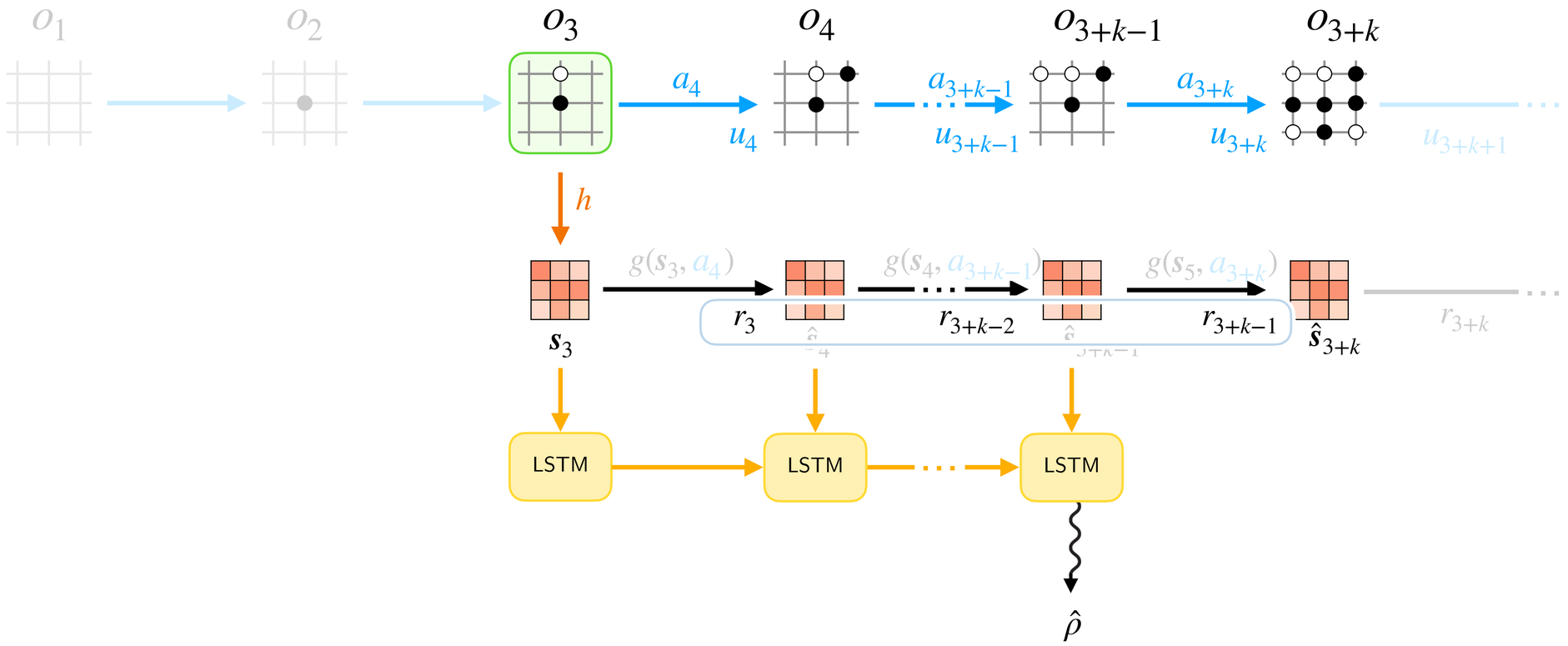
Instead of predicting the rewards at each exact point in time, only predict their sum for the next k steps.

$$Q(s_t, a) = \left[\sum_{i=0}^{k-1} \gamma^i r_{t+i}\right] + \gamma^k v_{t+k}$$

"value prefix"

ρ_t

(2) Prediction of the "value prefix"



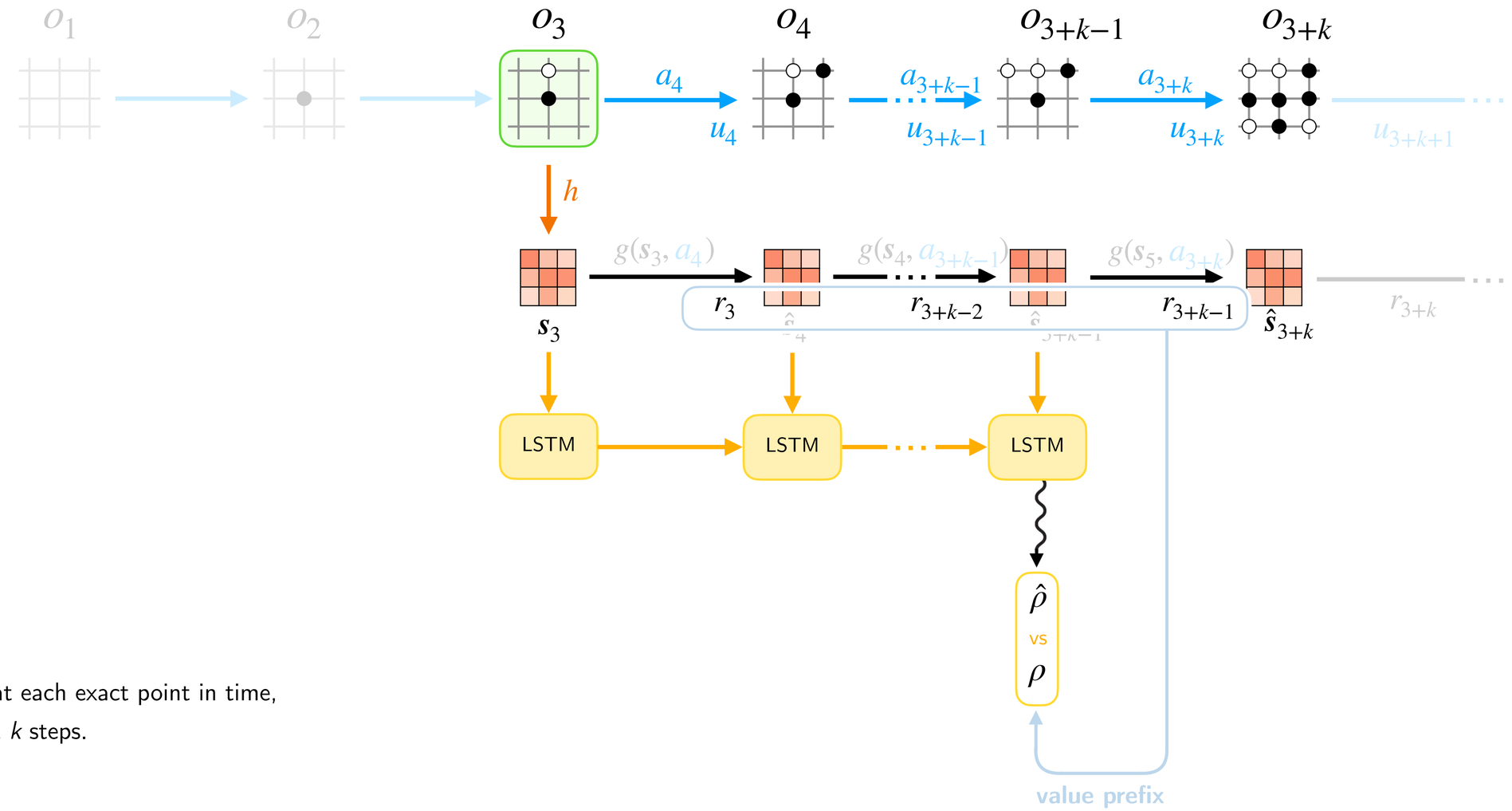
Instead of predicting the rewards at each exact point in time, only predict their sum for the next k steps.

$$Q(s_t, a) = \left[\sum_{i=0}^{k-1} \gamma^i r_{t+i}\right] + \gamma^k v_{t+k}$$

"value prefix"

ρ_t

(2) Prediction of the "value prefix"



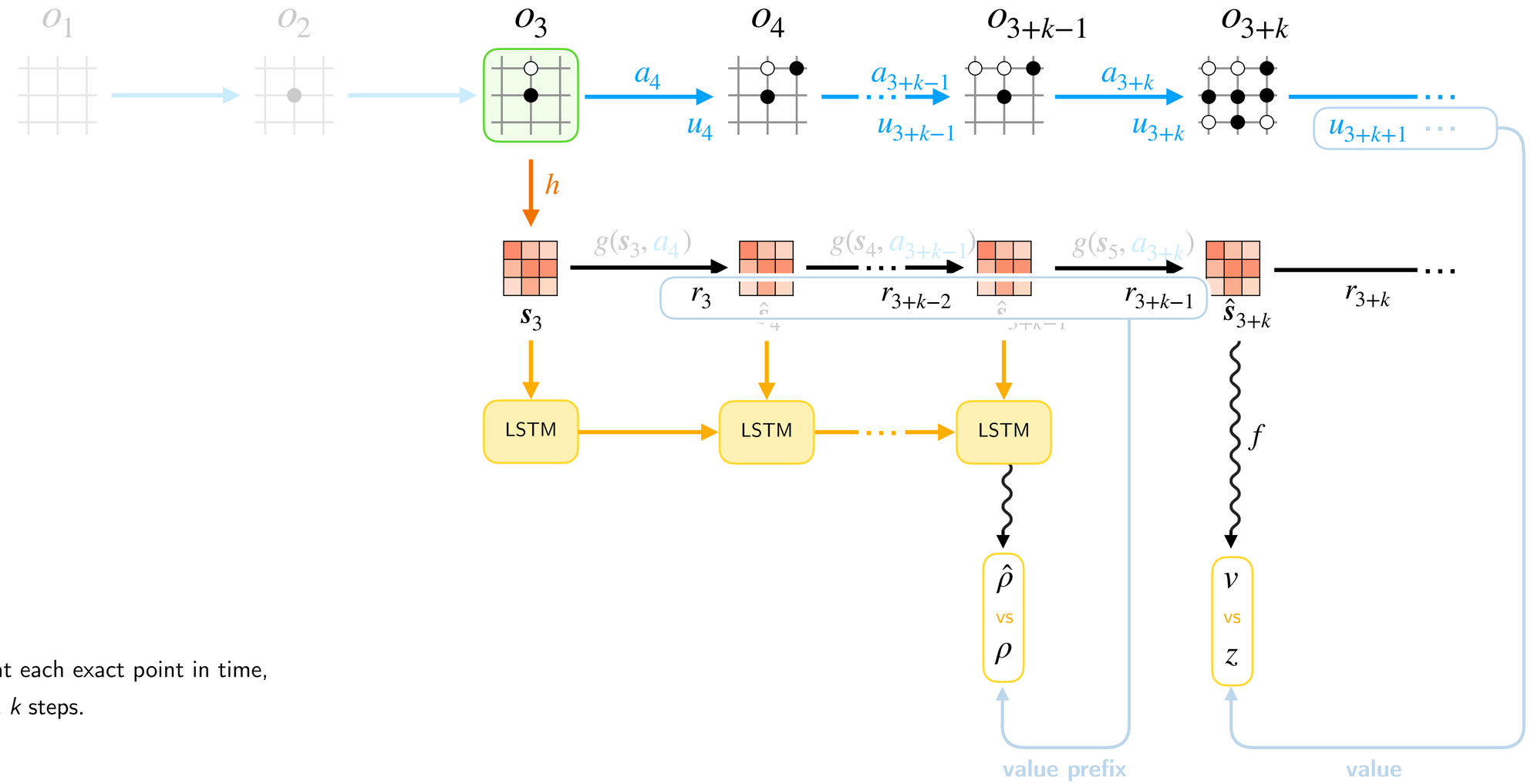
Instead of predicting the rewards at each exact point in time, only predict their sum for the next k steps.

$$Q(s_t, a) = \left[\sum_{i=0}^{k-1} \gamma^i r_{t+i}\right] + \gamma^k v_{t+k}$$

"value prefix"

ρ_t

(2) Prediction of the "value prefix"



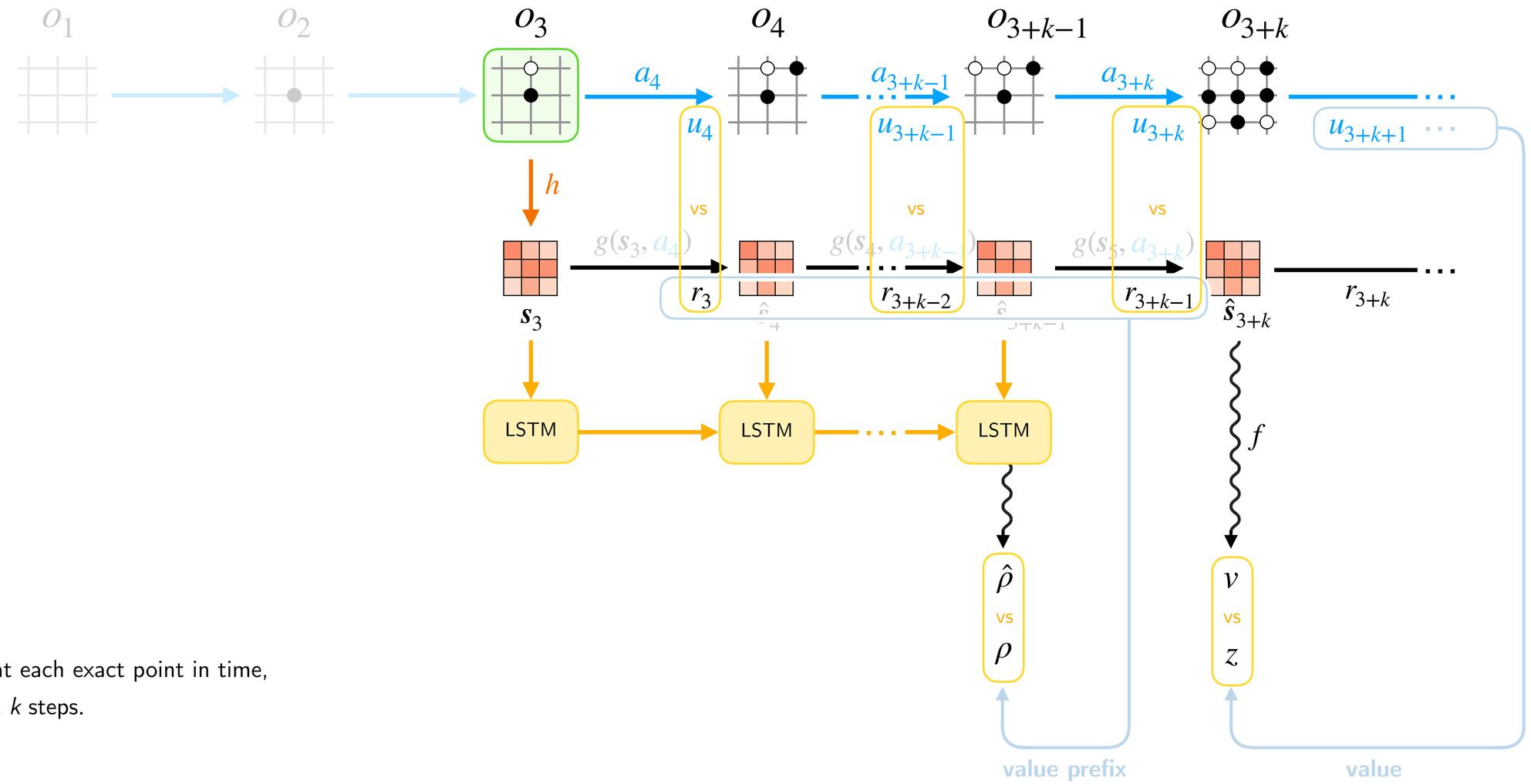
Instead of predicting the rewards at each exact point in time, only predict their sum for the next k steps.

$$Q(s_t, a) = \left[\sum_{i=0}^{k-1} \gamma^i r_{t+i}\right] + \gamma^k v_{t+k}$$

"value prefix"

ρ_t

(2) Prediction of the "value prefix"



Instead of predicting the rewards at each exact point in time, only predict their sum for the next k steps.

$$Q(s_t, a) = \left[\sum_{i=0}^{k-1} \gamma^i r_{t+i}\right] + \gamma^k v_{t+k}$$

"value prefix"

ρ_t

4

Stochastic MuZero

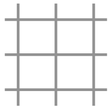
Stochastic MuZero

Let's generalize it even further, so it can be applied to ***stochastic state spaces***.

Stochastic MuZero

Pretend it's some kind of stochastic Go variant.

o_1

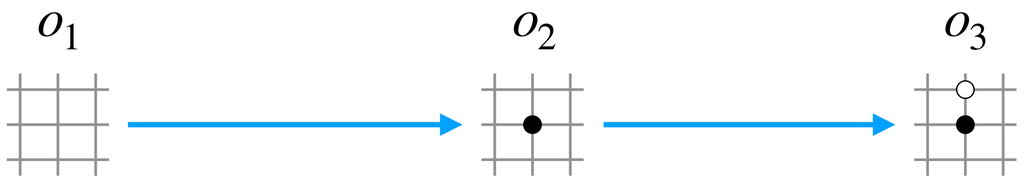


Stochastic MuZero

Pretend it's some kind of stochastic Go variant.

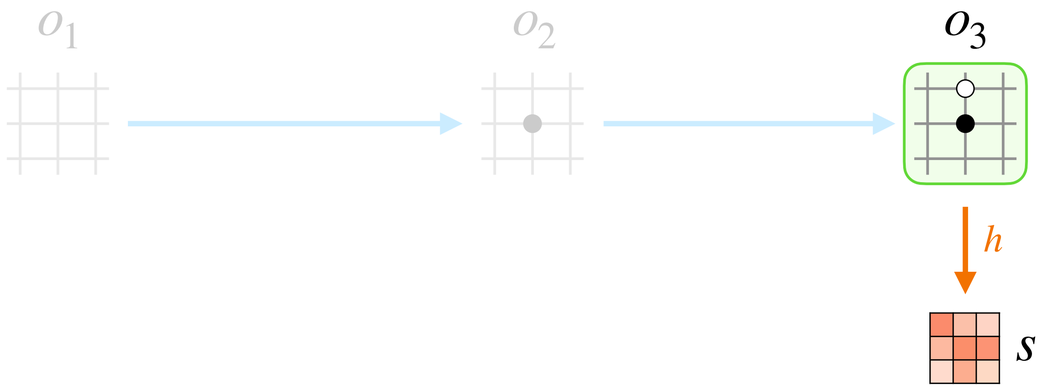
Stochastic MuZero

Pretend it's some kind of stochastic Go variant.



Stochastic MuZero

Pretend it's some kind of stochastic Go variant.

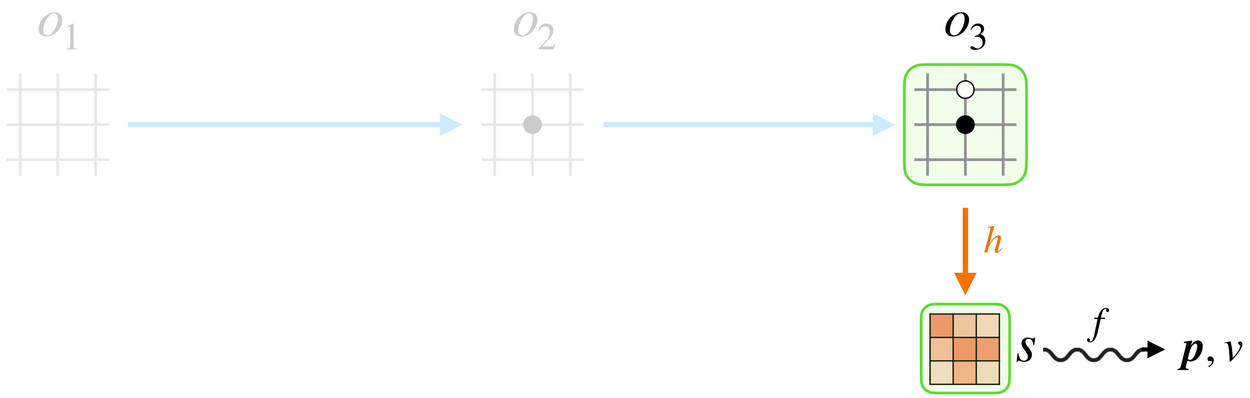


state s

representation h

Stochastic MuZero

Pretend it's some kind of stochastic Go variant.



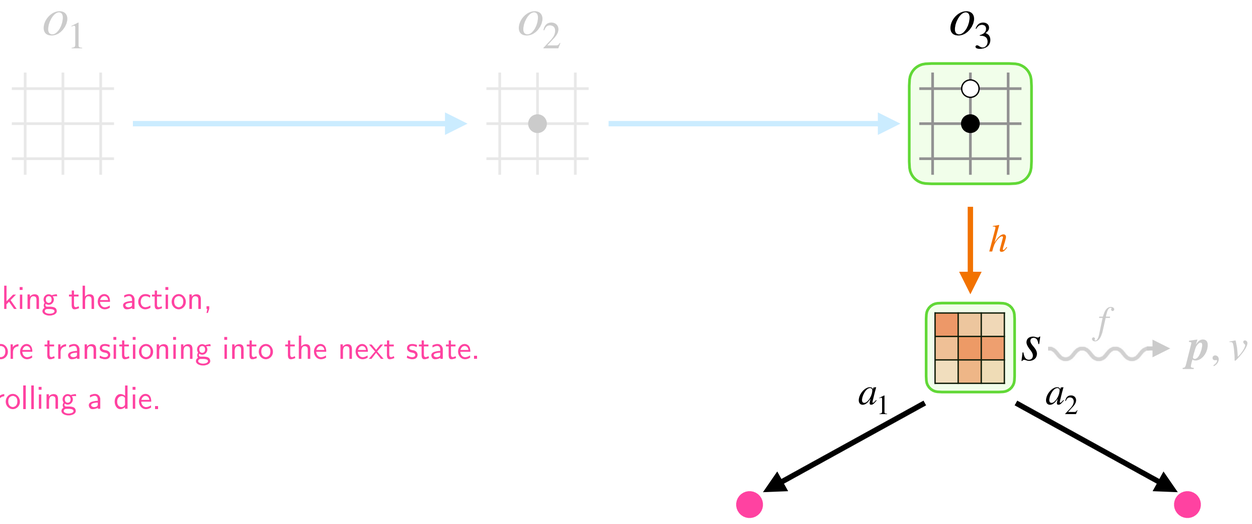
state s

representation h

prediction f

Stochastic MuZero

Pretend it's some kind of stochastic Go variant.



After taking the action,
but before transitioning into the next state.
Think: rolling a die.

state s

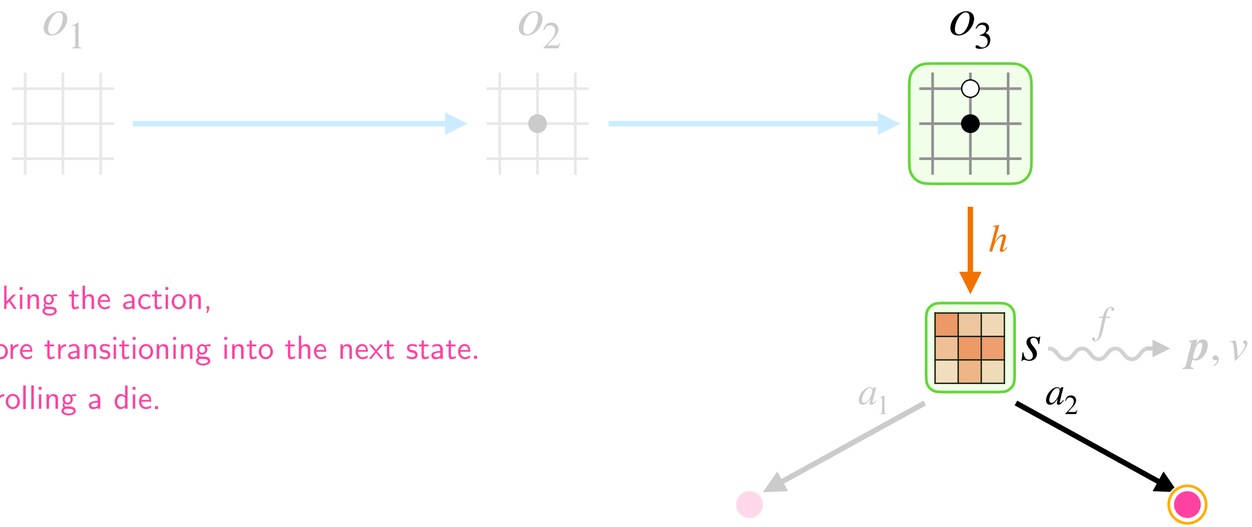
representation h

prediction f

afterstate α

Stochastic MuZero

Pretend it's some kind of stochastic Go variant.



After taking the action,
but before transitioning into the next state.
Think: rolling a die.

state s

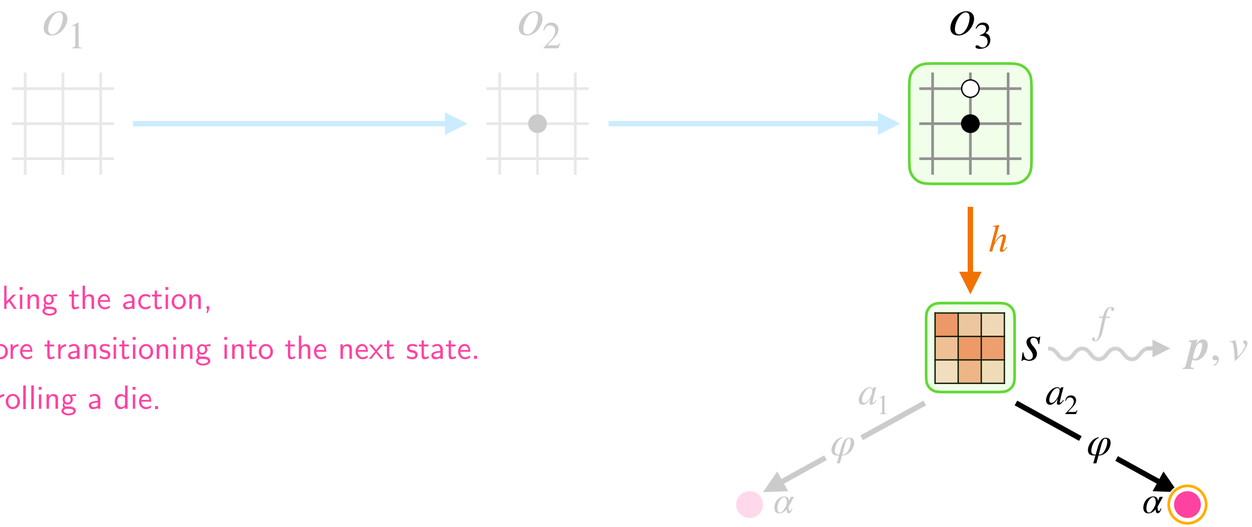
representation h

prediction f

afterstate α

Stochastic MuZero

Pretend it's some kind of stochastic Go variant.



After taking the action,
but before transitioning into the next state.
Think: rolling a die.

state s

representation h

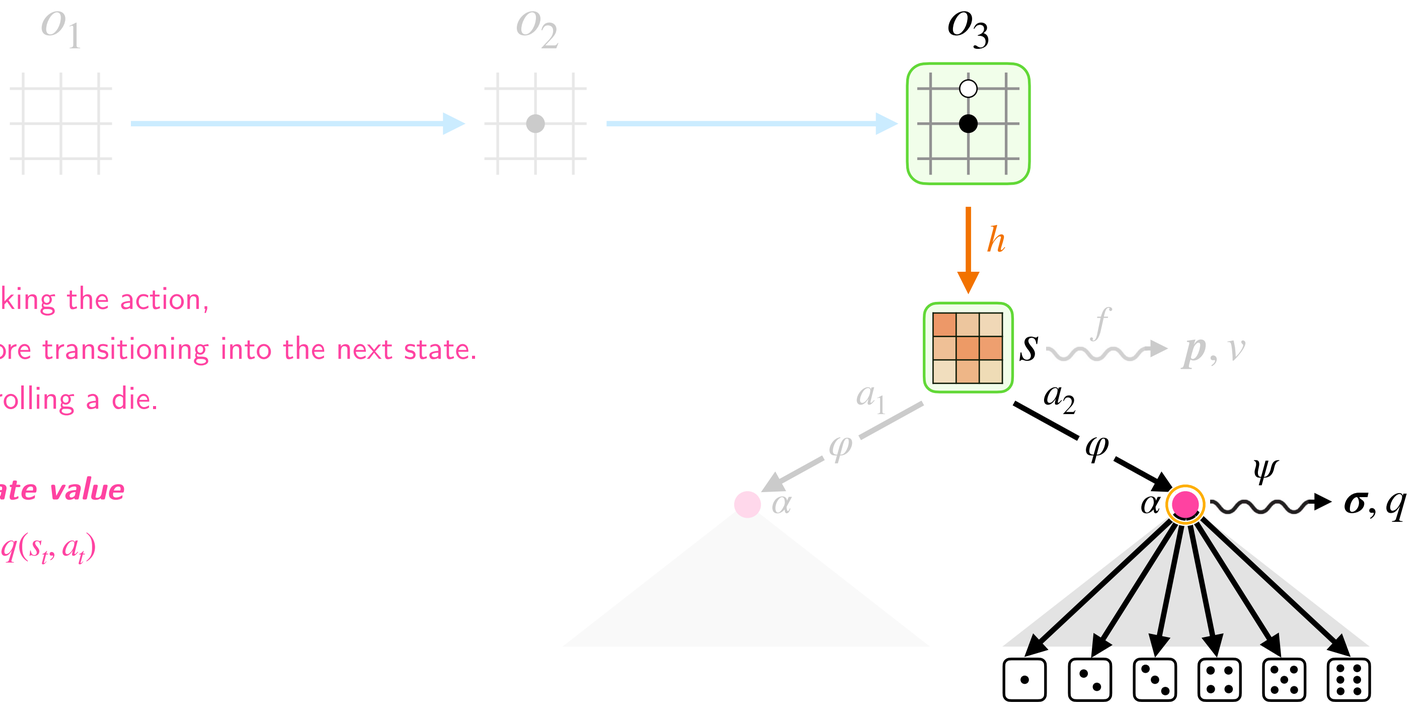
prediction f

afterstate α

afterstate dynamics φ

Stochastic MuZero

Pretend it's some kind of stochastic Go variant.



After taking the action,
but before transitioning into the next state.
Think: rolling a die.

afterstate value

$q(\alpha_t) = q(s_t, a_t)$

state s

representation h

prediction f

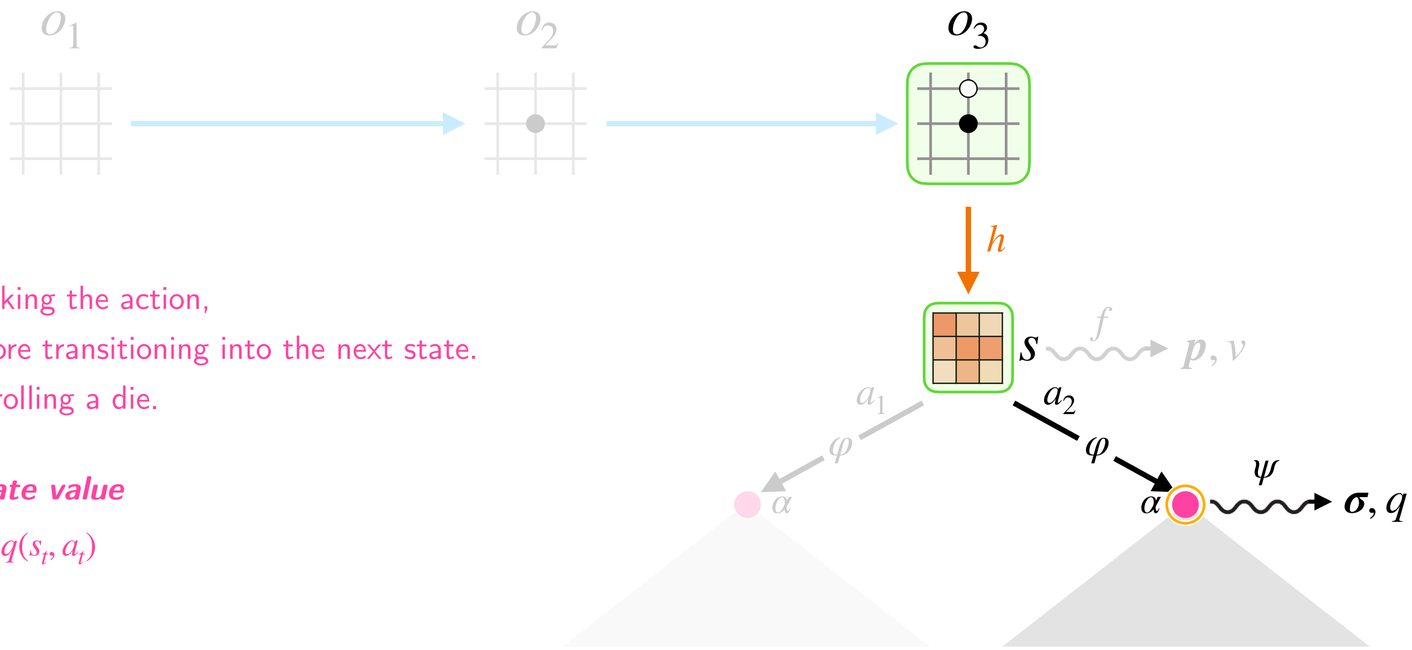
afterstate α

afterstate dynamics φ

afterstate prediction ψ

Stochastic MuZero

Pretend it's some kind of stochastic Go variant.



After taking the action,
but before transitioning into the next state.
Think: rolling a die.

afterstate value

$q(\alpha_t) = q(s_t, a_t)$

state s

representation h

prediction f

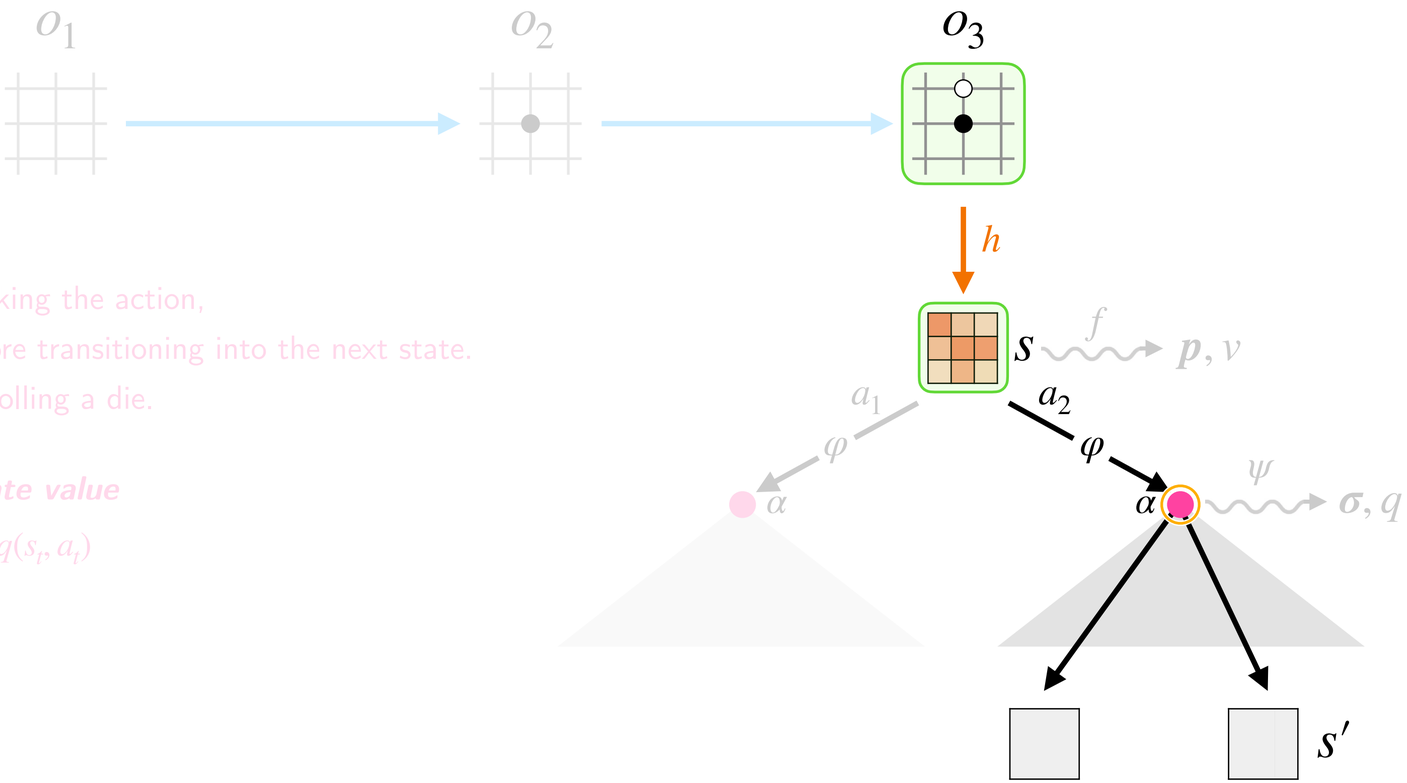
afterstate α

afterstate dynamics φ

afterstate prediction ψ

Stochastic MuZero

Pretend it's some kind of stochastic Go variant.



After taking the action,
but before transitioning into the next state.
Think: rolling a die.

afterstate value

$q(\alpha_t) = q(s_t, a_t)$

state s

representation h

prediction f

afterstate α

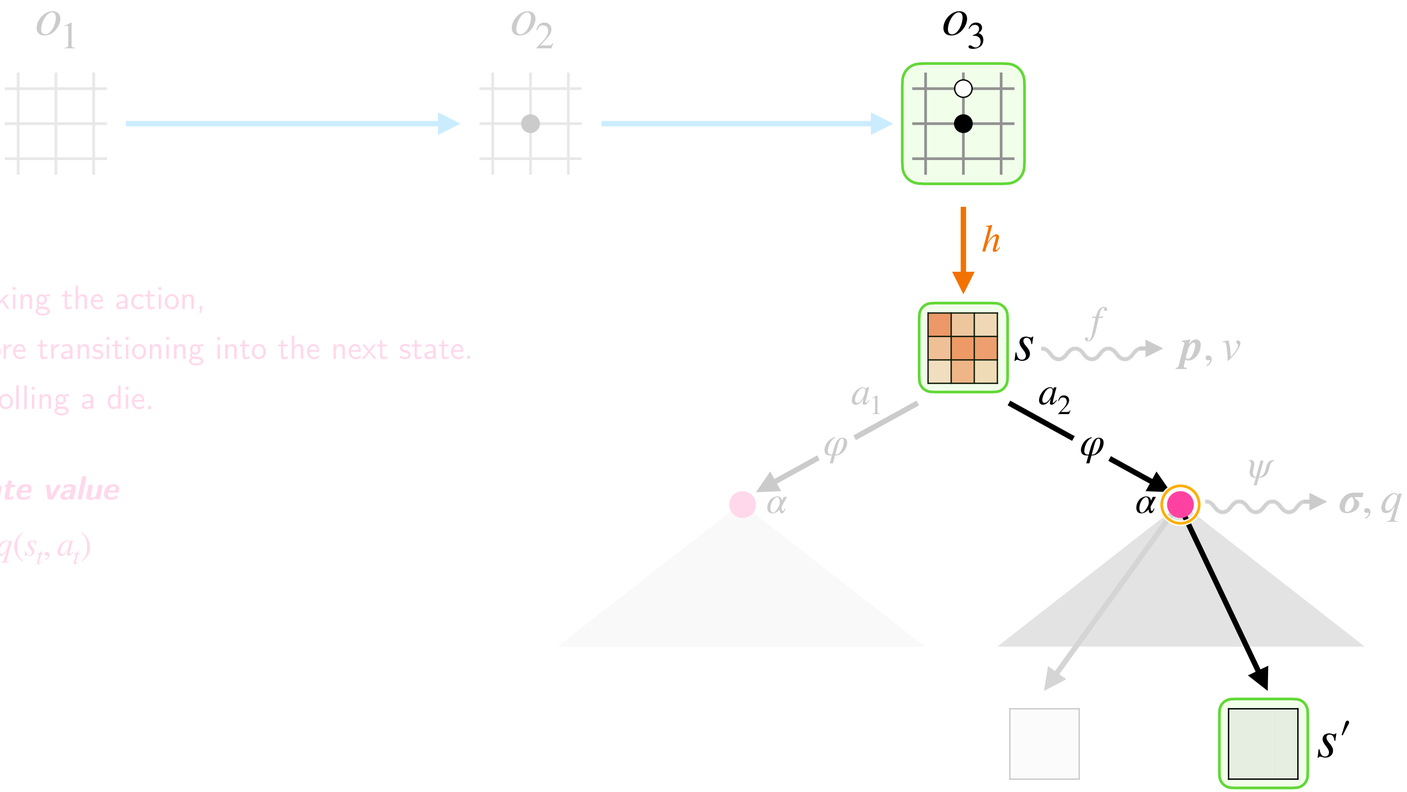
afterstate dynamics φ

afterstate prediction ψ

state s'

Stochastic MuZero

Pretend it's some kind of stochastic Go variant.



After taking the action,
but before transitioning into the next state.
Think: rolling a die.

afterstate value

$q(\alpha_t) = q(s_t, a_t)$

state s

representation h

prediction f

afterstate α

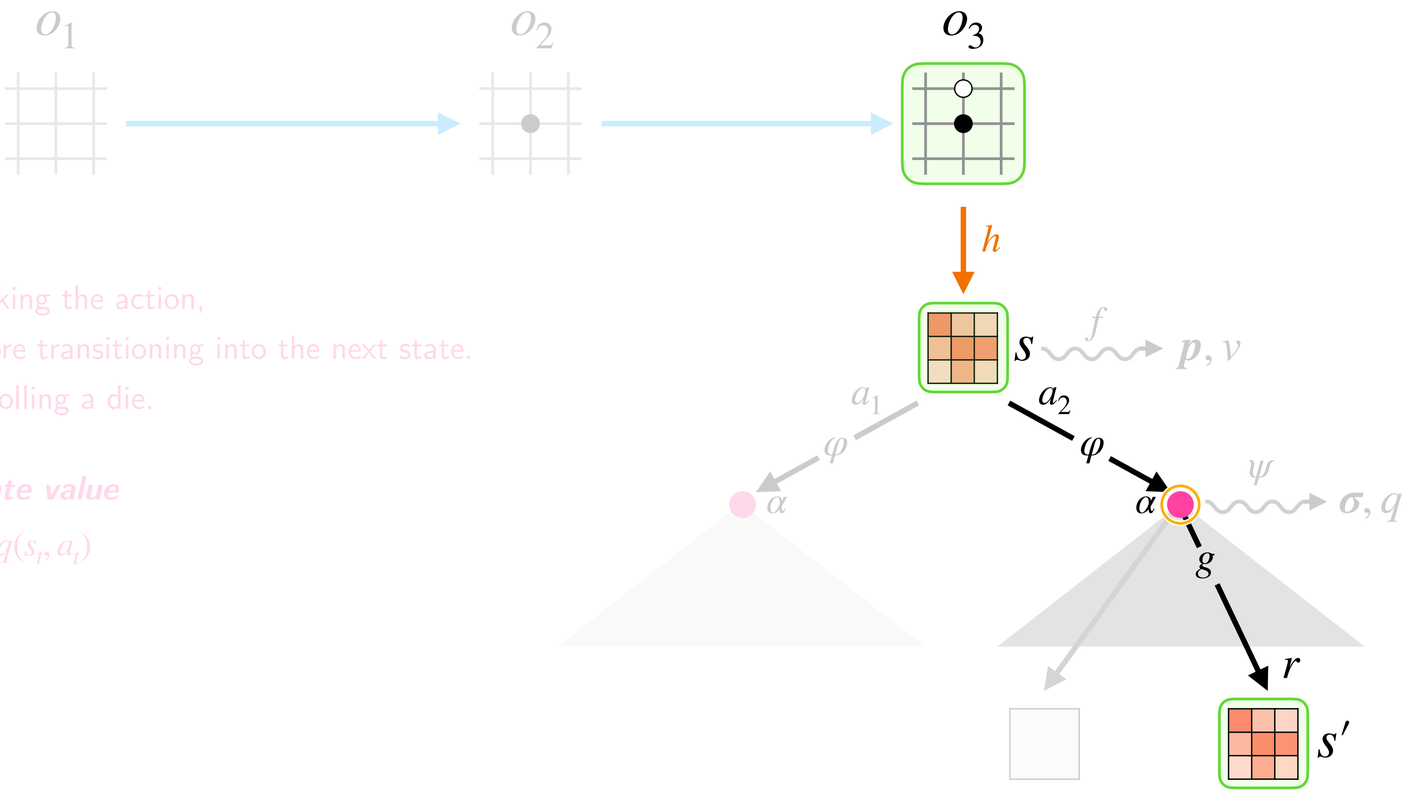
afterstate dynamics φ

afterstate prediction ψ

state s'

Stochastic MuZero

Pretend it's some kind of stochastic Go variant.



After taking the action,
but before transitioning into the next state.
Think: rolling a die.

afterstate value

$q(\alpha_t) = q(s_t, a_t)$

$c_3^1 \sim \sigma_3^0$

state s

representation h

prediction f

afterstate α

afterstate dynamics φ

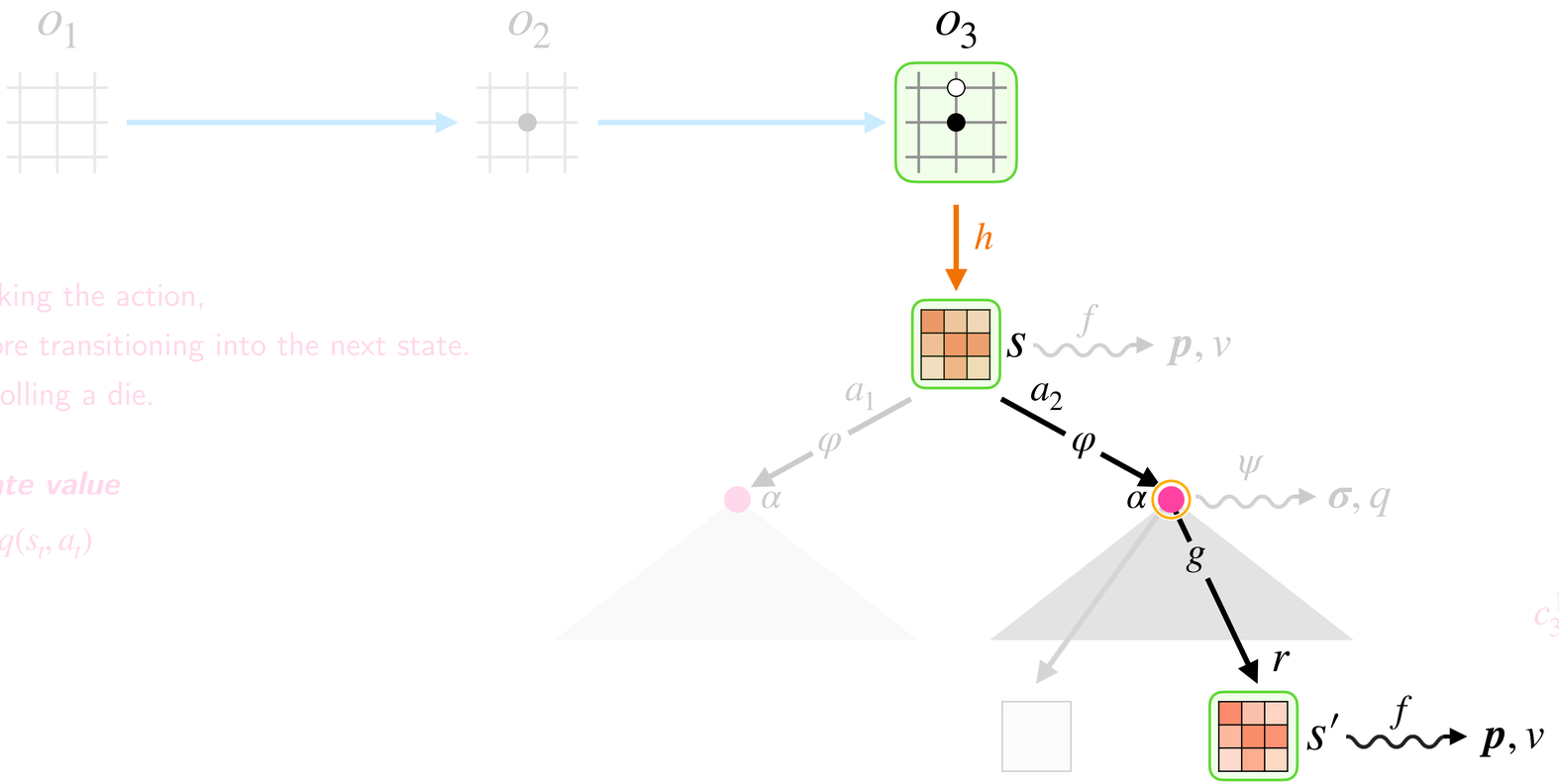
afterstate prediction ψ

state s'

dynamics g

Stochastic MuZero

Pretend it's some kind of stochastic Go variant.



After taking the action,
but before transitioning into the next state.
Think: rolling a die.

afterstate value

$q(\alpha_t) = q(s_t, a_t)$

$c_3^1 \sim \sigma_3^0$

state s

representation h

prediction f

afterstate α

afterstate dynamics φ

afterstate prediction ψ

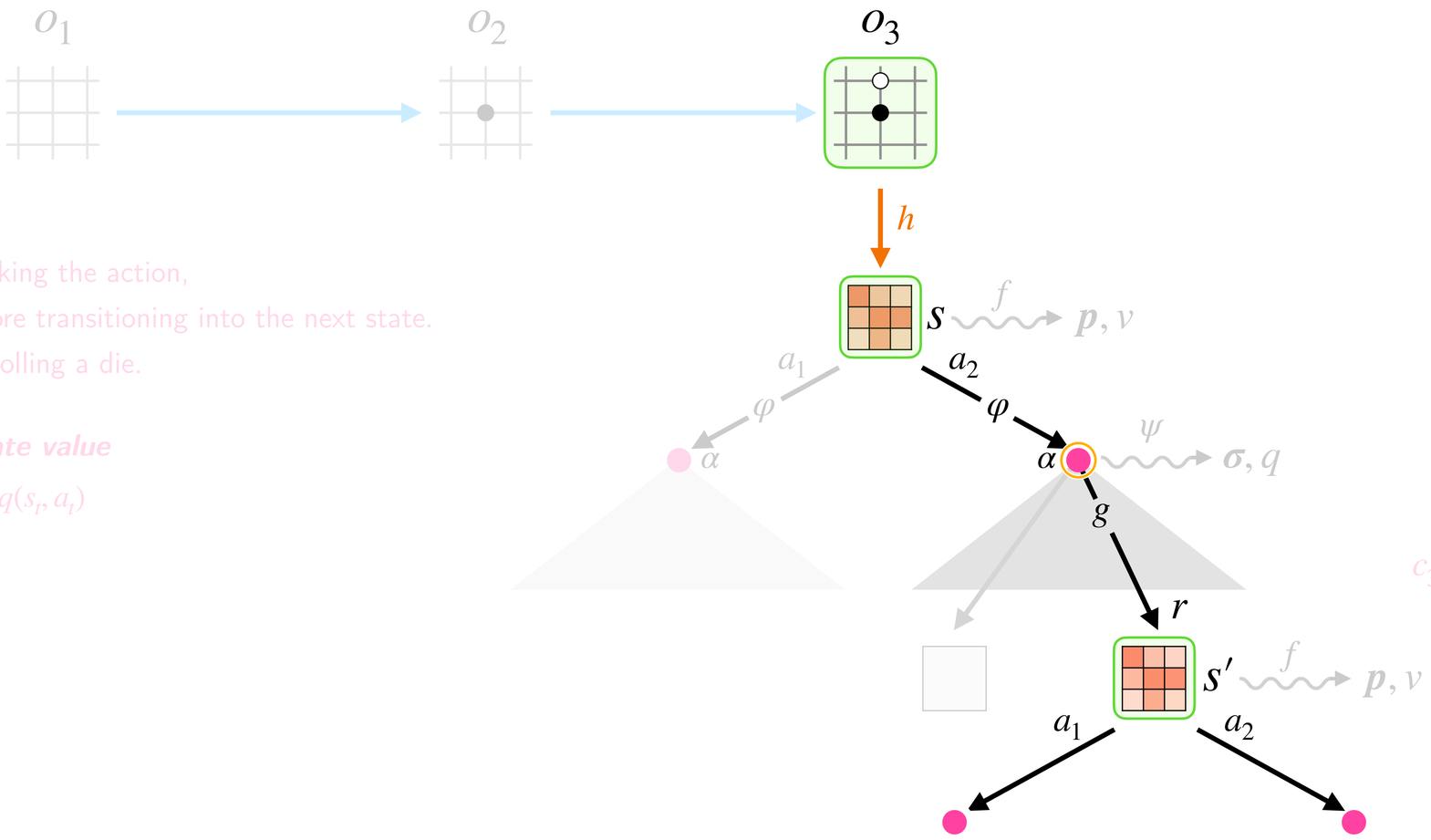
state s'

dynamics g

prediction f

Stochastic MuZero

Pretend it's some kind of stochastic Go variant.



After taking the action,
but before transitioning into the next state.
Think: rolling a die.

afterstate value

$q(\alpha_t) = q(s_t, a_t)$

$c_3^1 \sim \sigma_3^0$

state s

representation h

prediction f

afterstate α

afterstate dynamics φ

afterstate prediction ψ

state s'

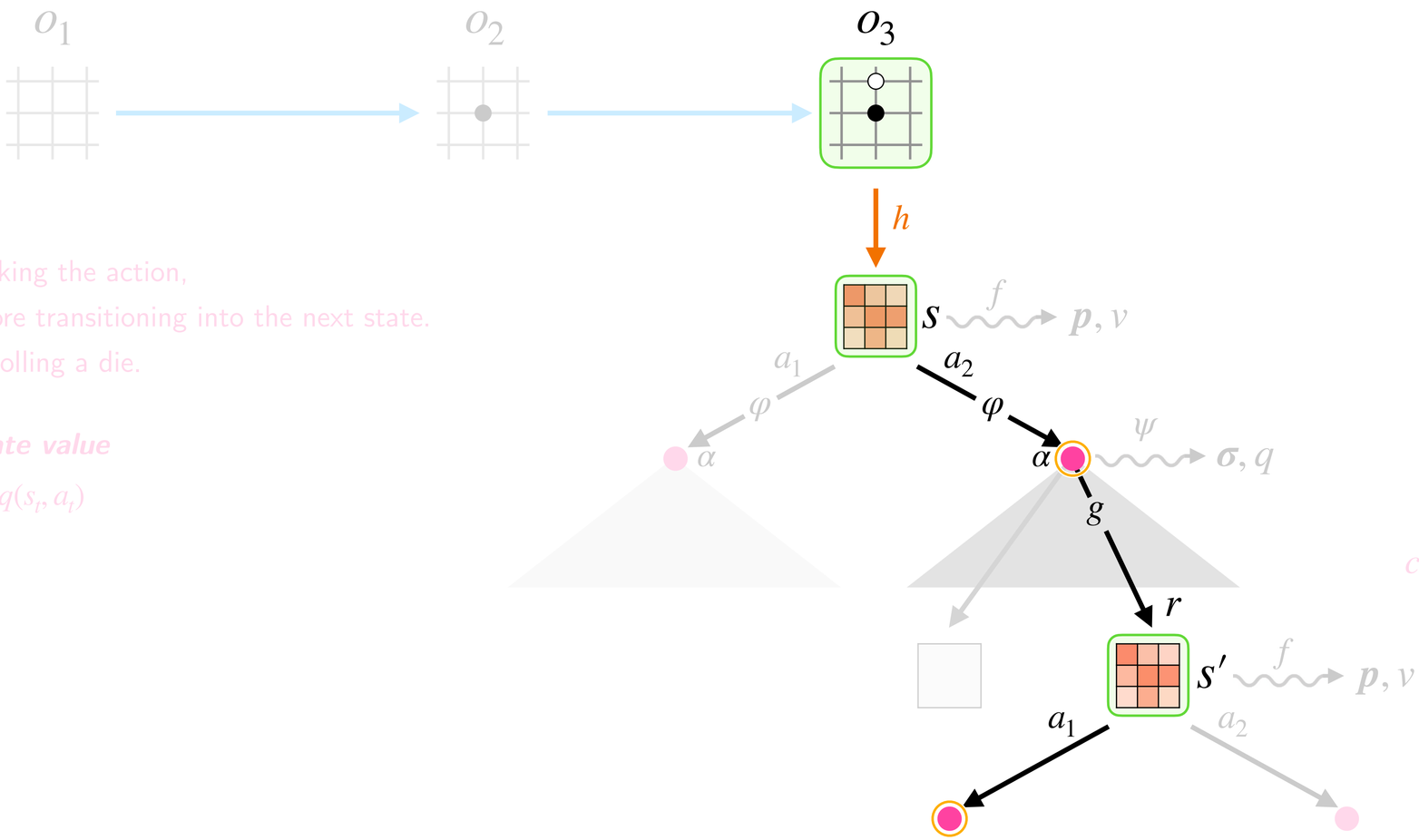
dynamics g

prediction f

afterstate α'

Stochastic MuZero

Pretend it's some kind of stochastic Go variant.



After taking the action,
but before transitioning into the next state.
Think: rolling a die.

afterstate value

$q(\alpha_t) = q(s_t, a_t)$

$c_3^1 \sim \sigma_3^0$

state s

representation h

prediction f

afterstate α

afterstate dynamics φ

afterstate prediction ψ

state s'

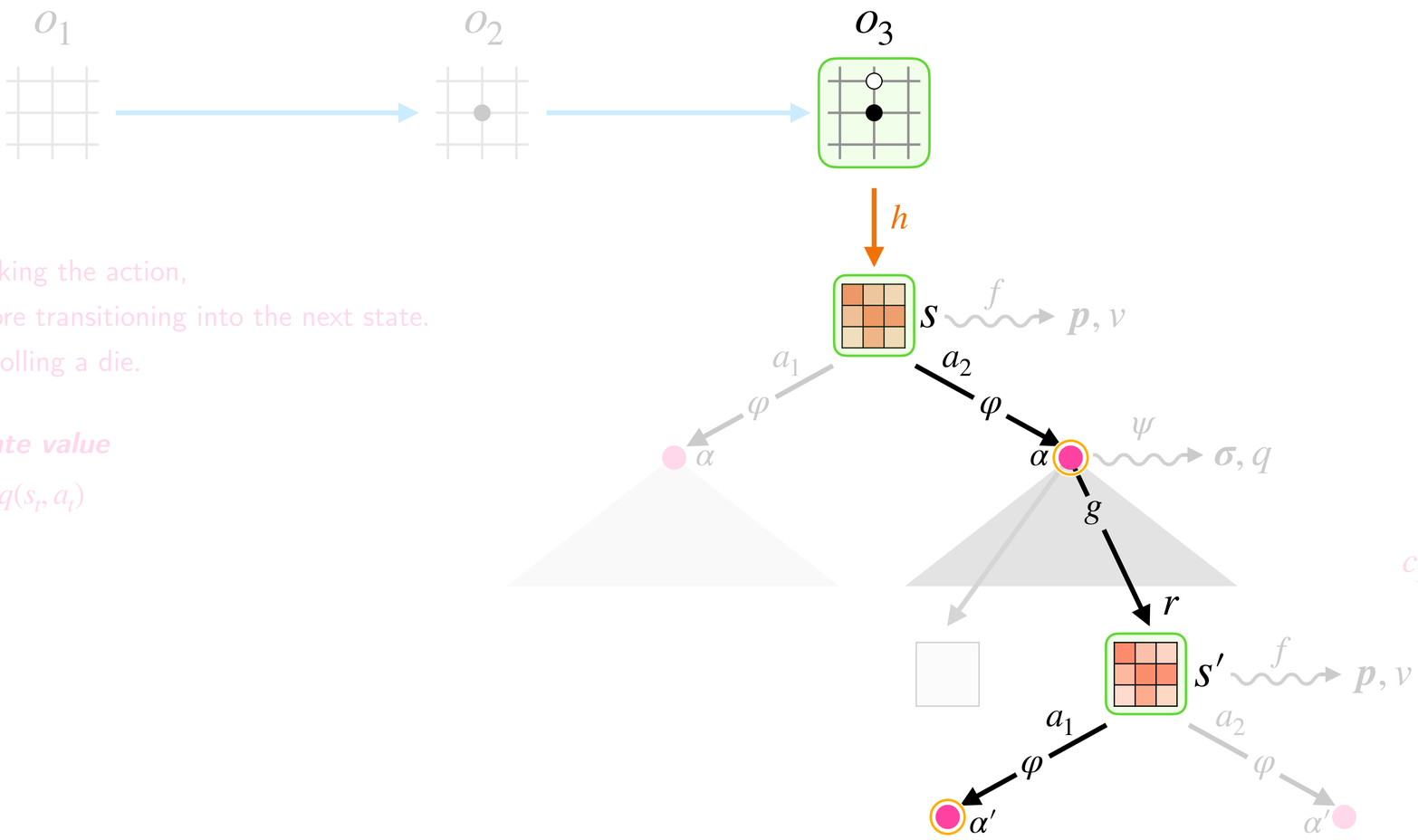
dynamics g

prediction f

afterstate α'

Stochastic MuZero

Pretend it's some kind of stochastic Go variant.



After taking the action,
but before transitioning into the next state.
Think: rolling a die.

afterstate value

$q(\alpha_t) = q(s_t, a_t)$

$c_3^1 \sim \sigma_3^0$

state s

representation h

prediction f

afterstate α

afterstate dynamics φ

afterstate prediction ψ

state s'

dynamics g

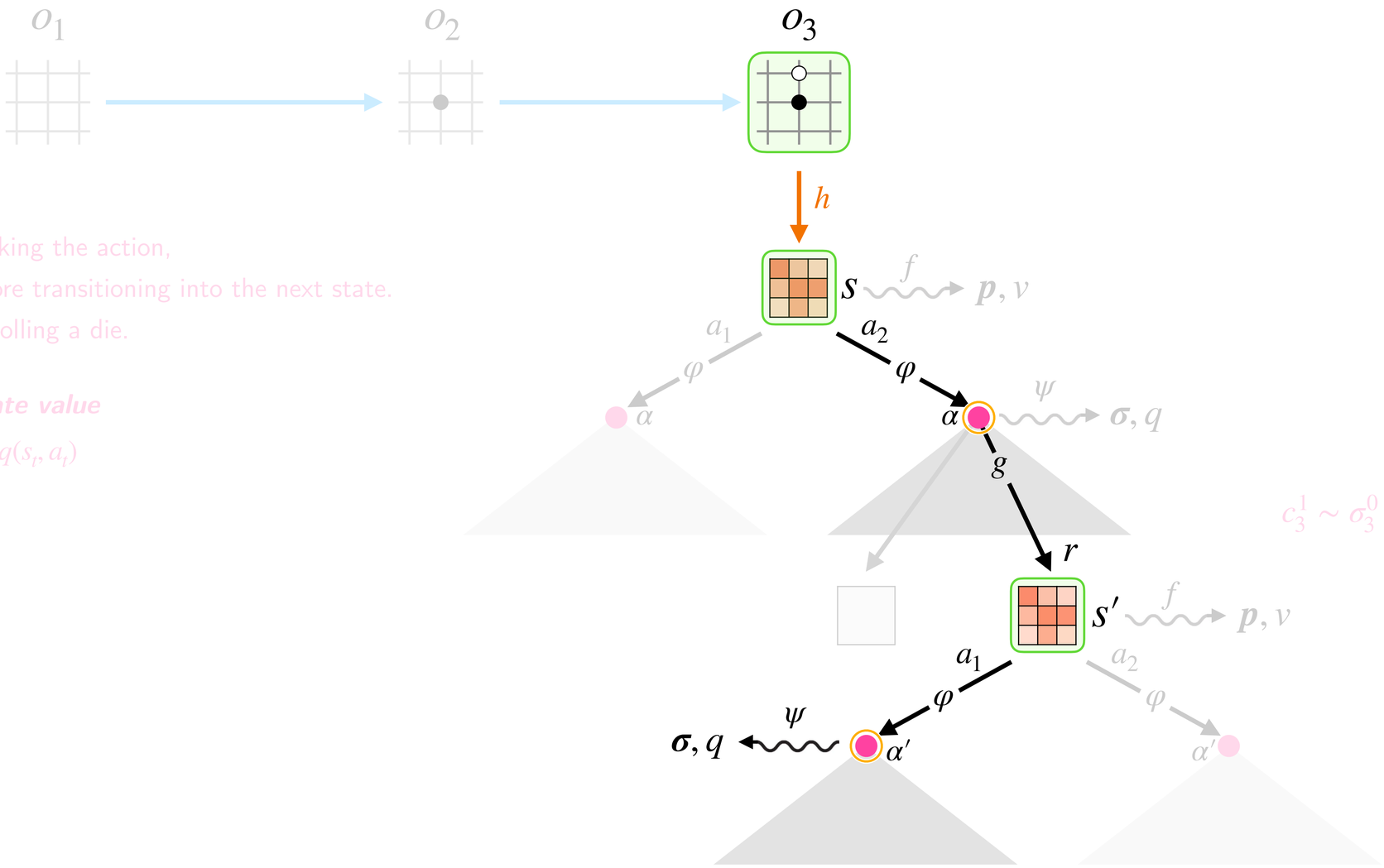
prediction f

afterstate α'

afterstate dynamics φ

Stochastic MuZero

Pretend it's some kind of stochastic Go variant.



After taking the action,
but before transitioning into the next state.
Think: rolling a die.

afterstate value

$q(\alpha_t) = q(s_t, a_t)$

$c_3^1 \sim \sigma_3^0$

state s
representation h
prediction f

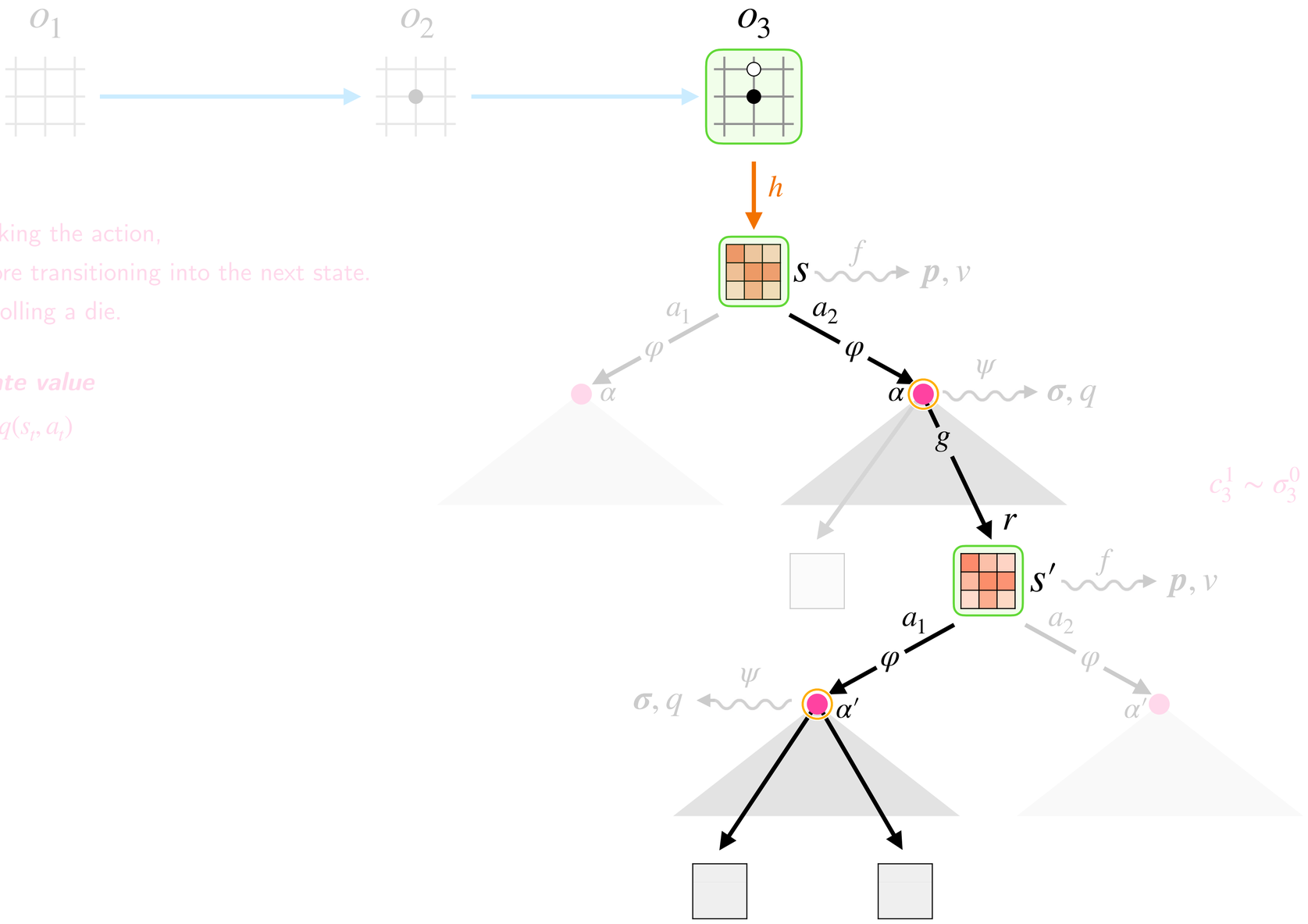
afterstate α
afterstate dynamics φ
afterstate prediction ψ

state s'
dynamics g
prediction f

afterstate α'
afterstate dynamics φ
afterstate prediction ψ

Stochastic MuZero

Pretend it's some kind of stochastic Go variant.



After taking the action,
but before transitioning into the next state.
Think: rolling a die.

afterstate value

$q(\alpha_t) = q(s_t, a_t)$

$c_3^1 \sim \sigma_3^0$

state s
representation h
prediction f

afterstate α
afterstate dynamics φ
afterstate prediction ψ

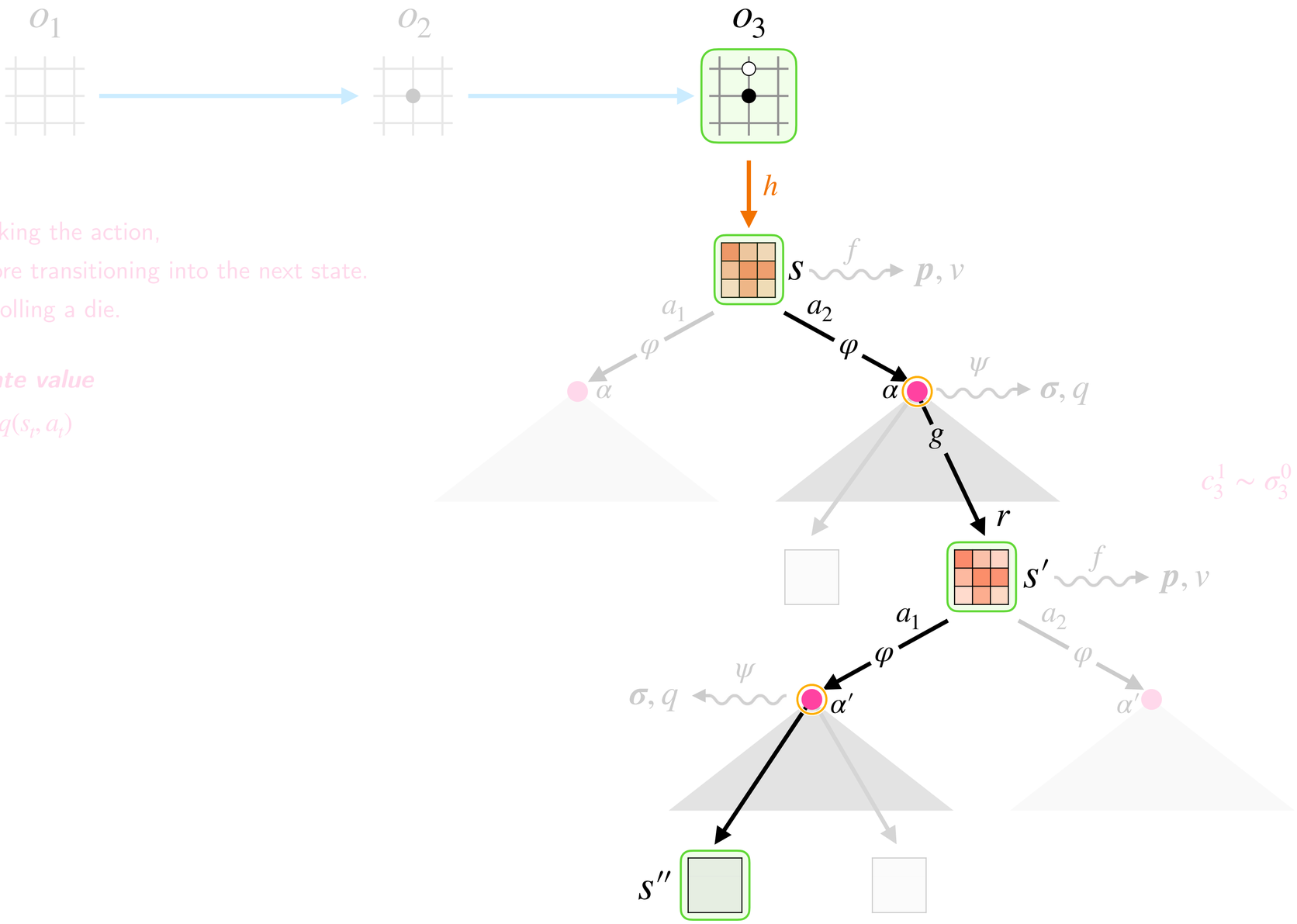
state s'
dynamics g
prediction f

afterstate α'
afterstate dynamics φ
afterstate prediction ψ

state s''
dynamics g
prediction f

Stochastic MuZero

Pretend it's some kind of stochastic Go variant.



After taking the action,
but before transitioning into the next state.
Think: rolling a die.

afterstate value

$q(\alpha_t) = q(s_t, a_t)$

$c_3^1 \sim \sigma_3^0$

state s
representation h
prediction f

afterstate α
afterstate dynamics φ
afterstate prediction ψ

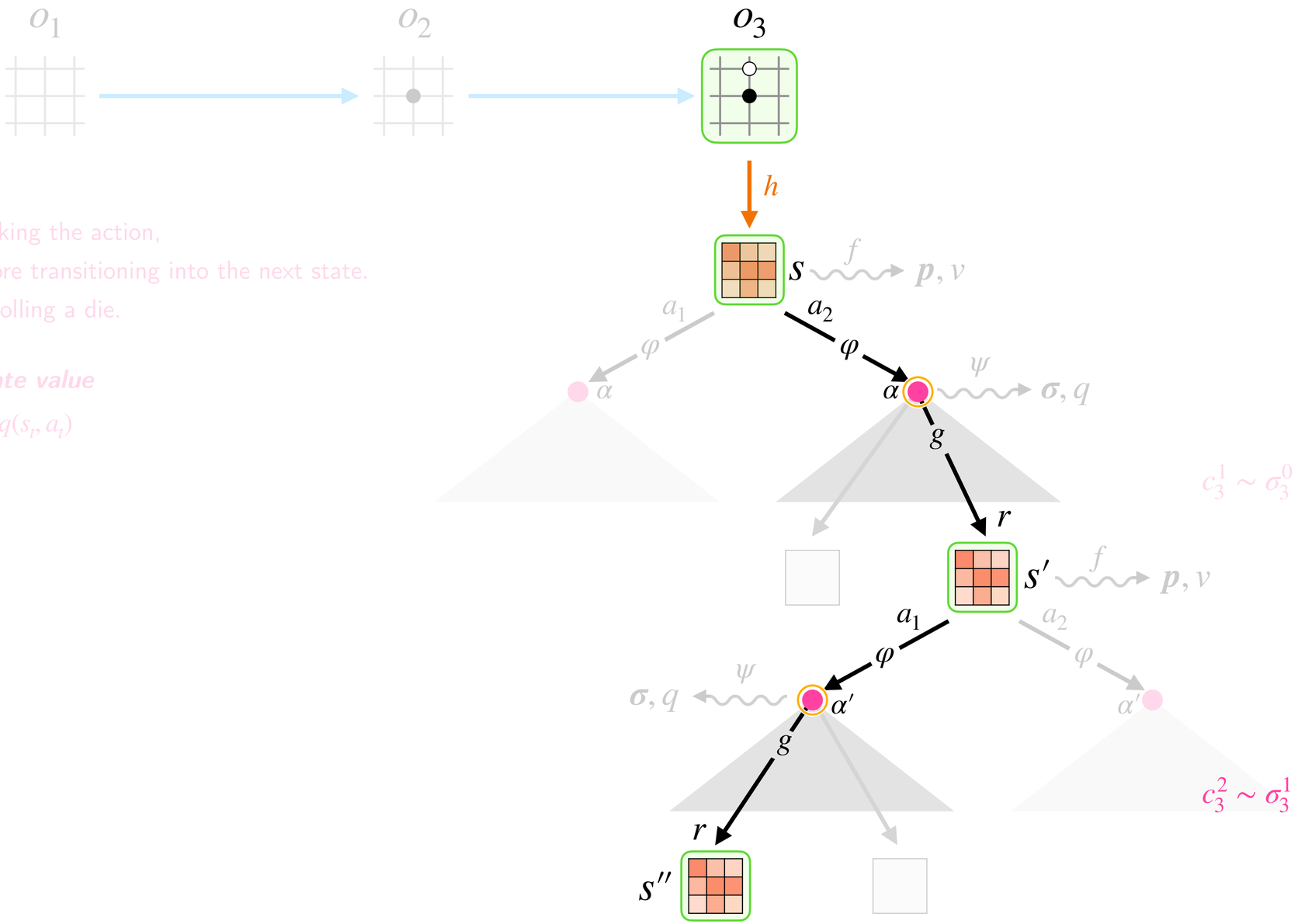
state s'
dynamics g
prediction f

afterstate α'
afterstate dynamics φ
afterstate prediction ψ

state s''

Stochastic MuZero

Pretend it's some kind of stochastic Go variant.



After taking the action,
but before transitioning into the next state.
Think: rolling a die.

afterstate value

$q(\alpha_t) = q(s_t, a_t)$

state s

representation h

prediction f

afterstate α

afterstate dynamics φ

afterstate prediction ψ

state s'

dynamics g

prediction f

afterstate α'

afterstate dynamics φ

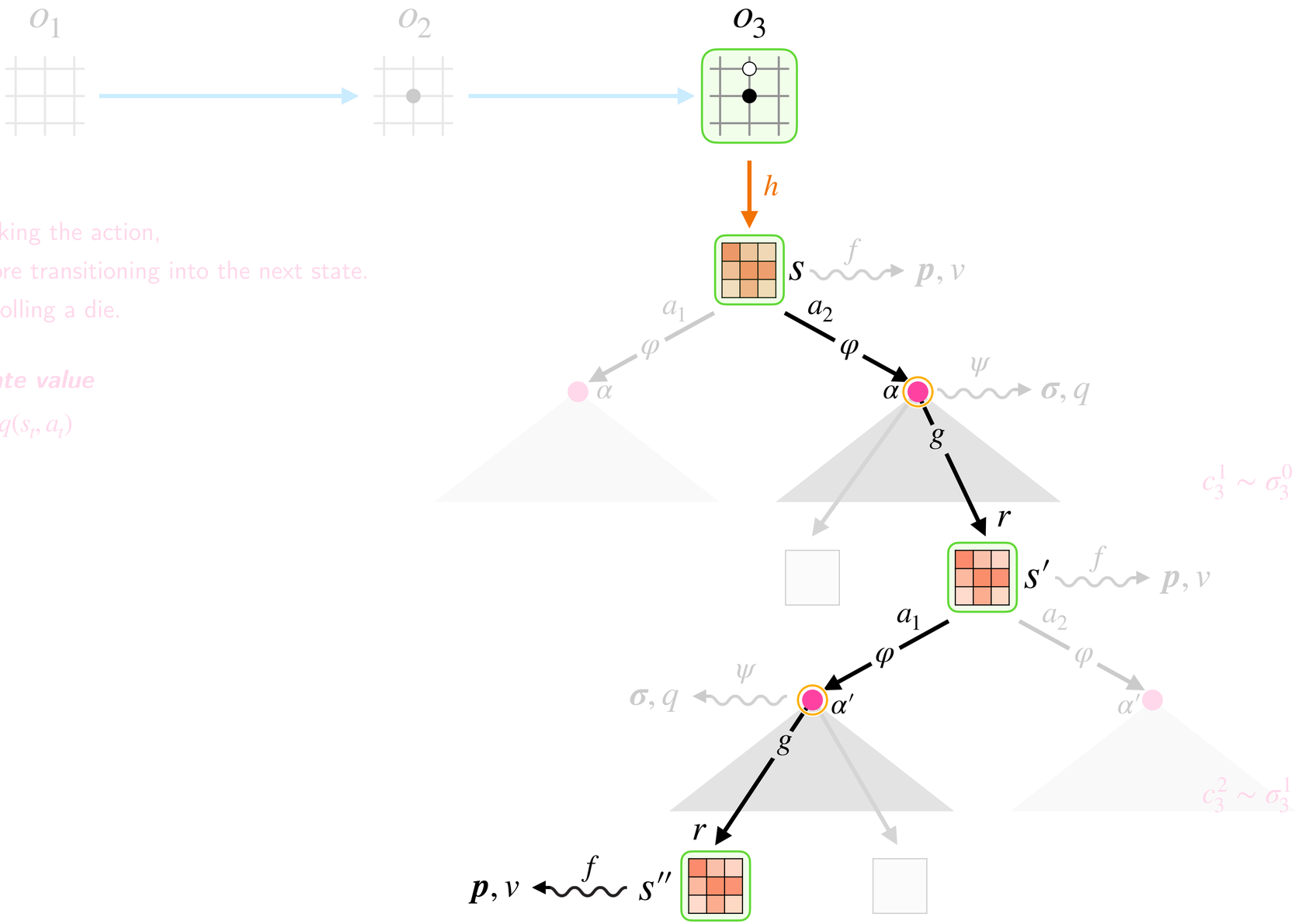
afterstate prediction ψ

state s''

dynamics g

Stochastic MuZero

Pretend it's some kind of stochastic Go variant.



After taking the action,
but before transitioning into the next state.
Think: rolling a die.

afterstate value

$q(\alpha_t) = q(s_t, a_t)$

state s
representation h
prediction f

afterstate α
afterstate dynamics φ
afterstate prediction ψ

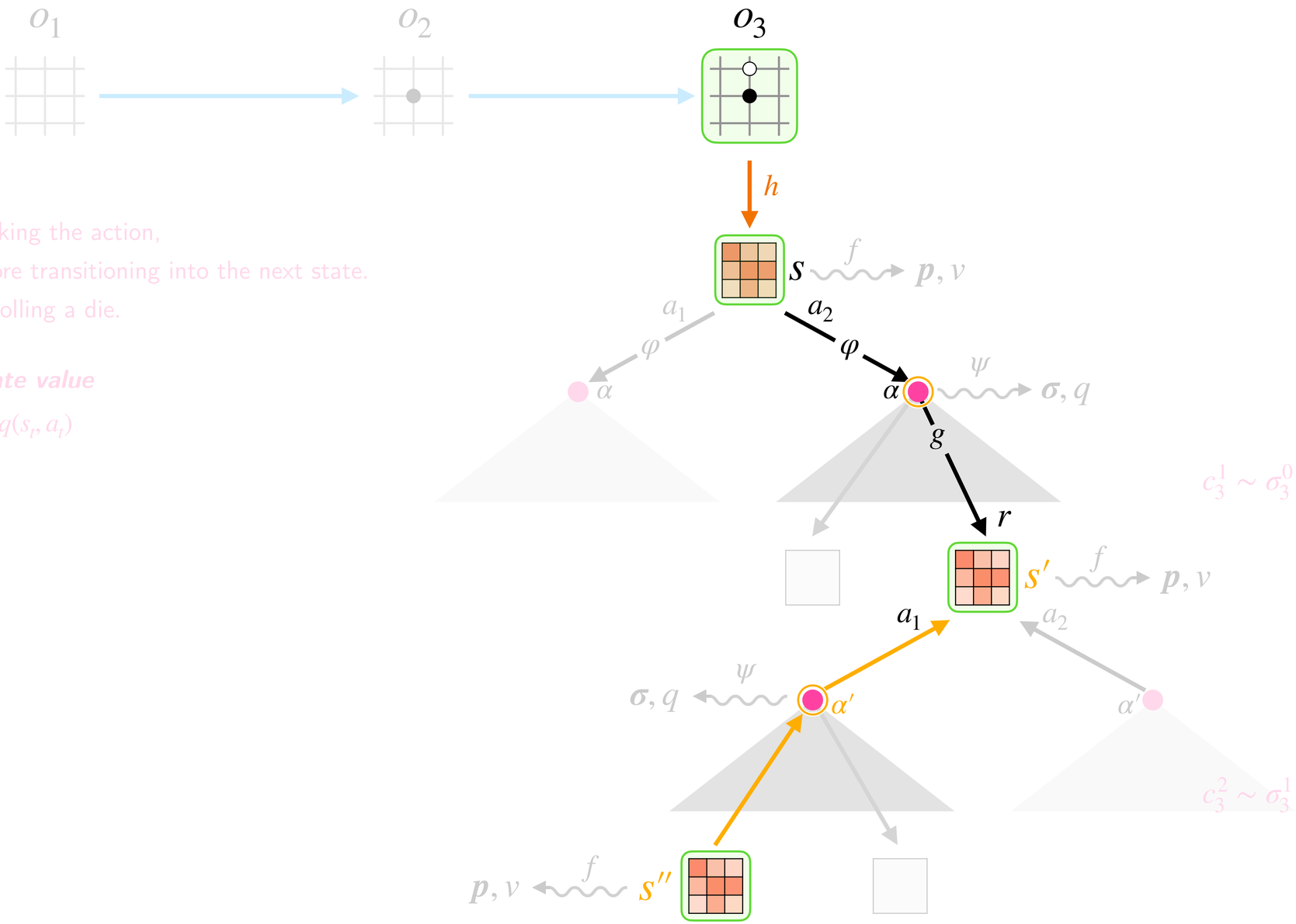
state s'
dynamics g
prediction f

afterstate α'
afterstate dynamics φ
afterstate prediction ψ

state s''
dynamics g
prediction f

Stochastic MuZero

Pretend it's some kind of stochastic Go variant.



After taking the action,
but before transitioning into the next state.
Think: rolling a die.

afterstate value

$q(\alpha_t) = q(s_t, a_t)$

state s
representation h
prediction f

afterstate α
afterstate dynamics φ
afterstate prediction ψ

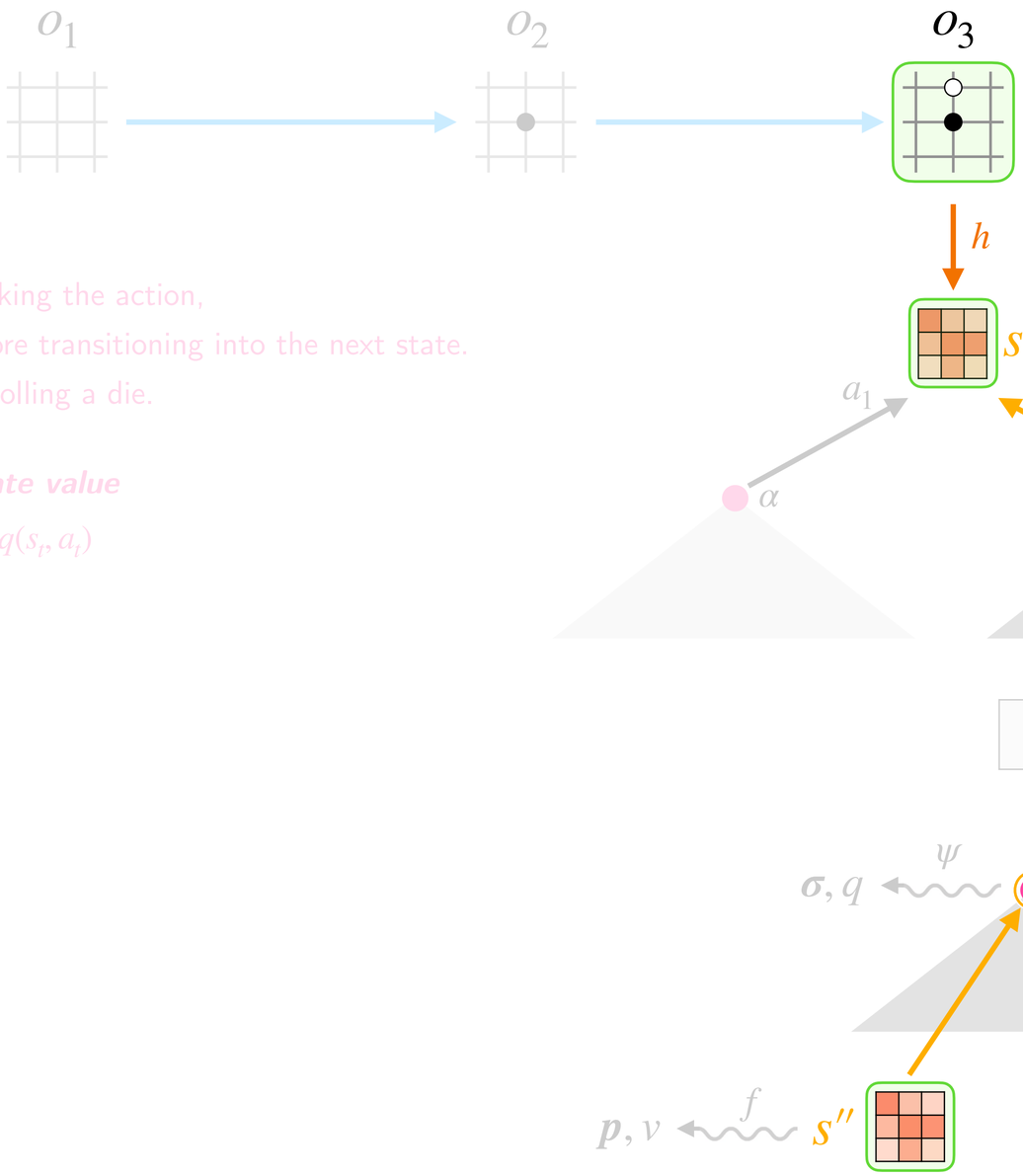
state s'
dynamics g
prediction f

afterstate α'
afterstate dynamics φ
afterstate prediction ψ

state s''
dynamics g
prediction f

Stochastic MuZero

Pretend it's some kind of stochastic Go variant.



After taking the action,
but before transitioning into the next state.
Think: rolling a die.

afterstate value

$q(\alpha_t) = q(s_t, a_t)$

state s
representation h
prediction f

afterstate α
afterstate dynamics φ
afterstate prediction ψ

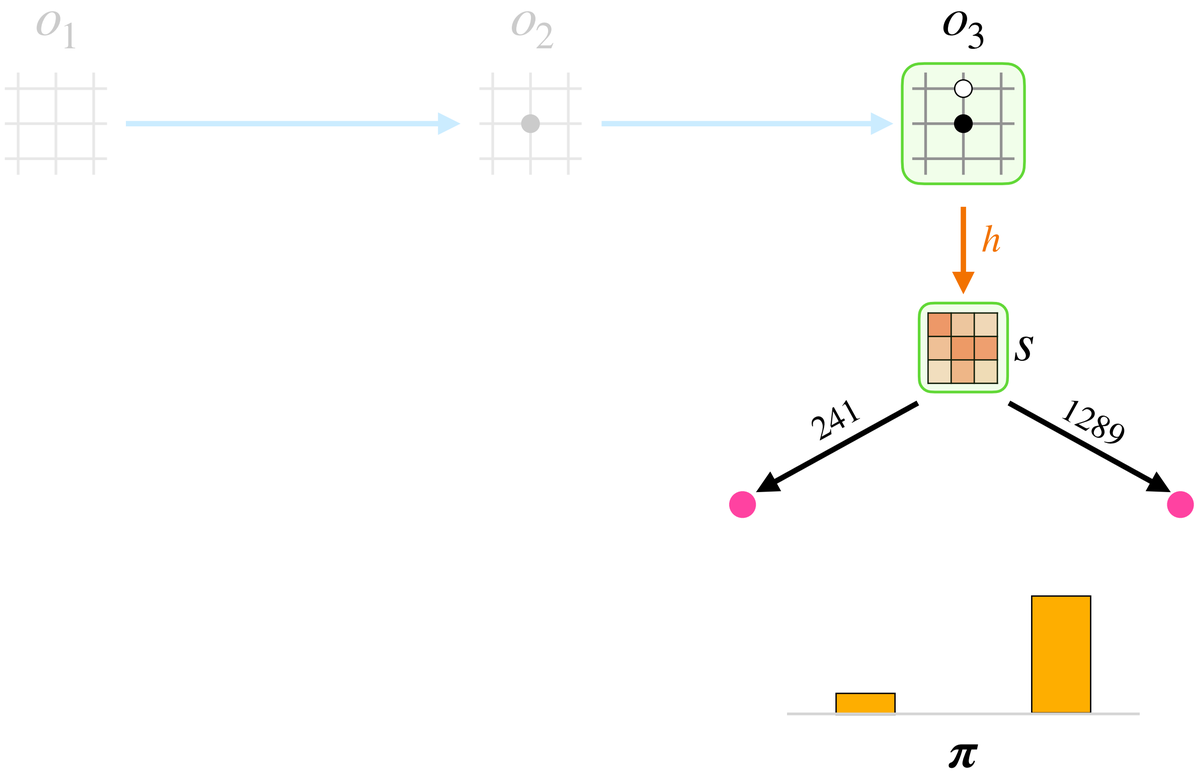
state s'
dynamics g
prediction f

afterstate α'
afterstate dynamics φ
afterstate prediction ψ

state s''
dynamics g
prediction f

Stochastic MuZero

Pretend it's some kind of stochastic Go variant.



state s

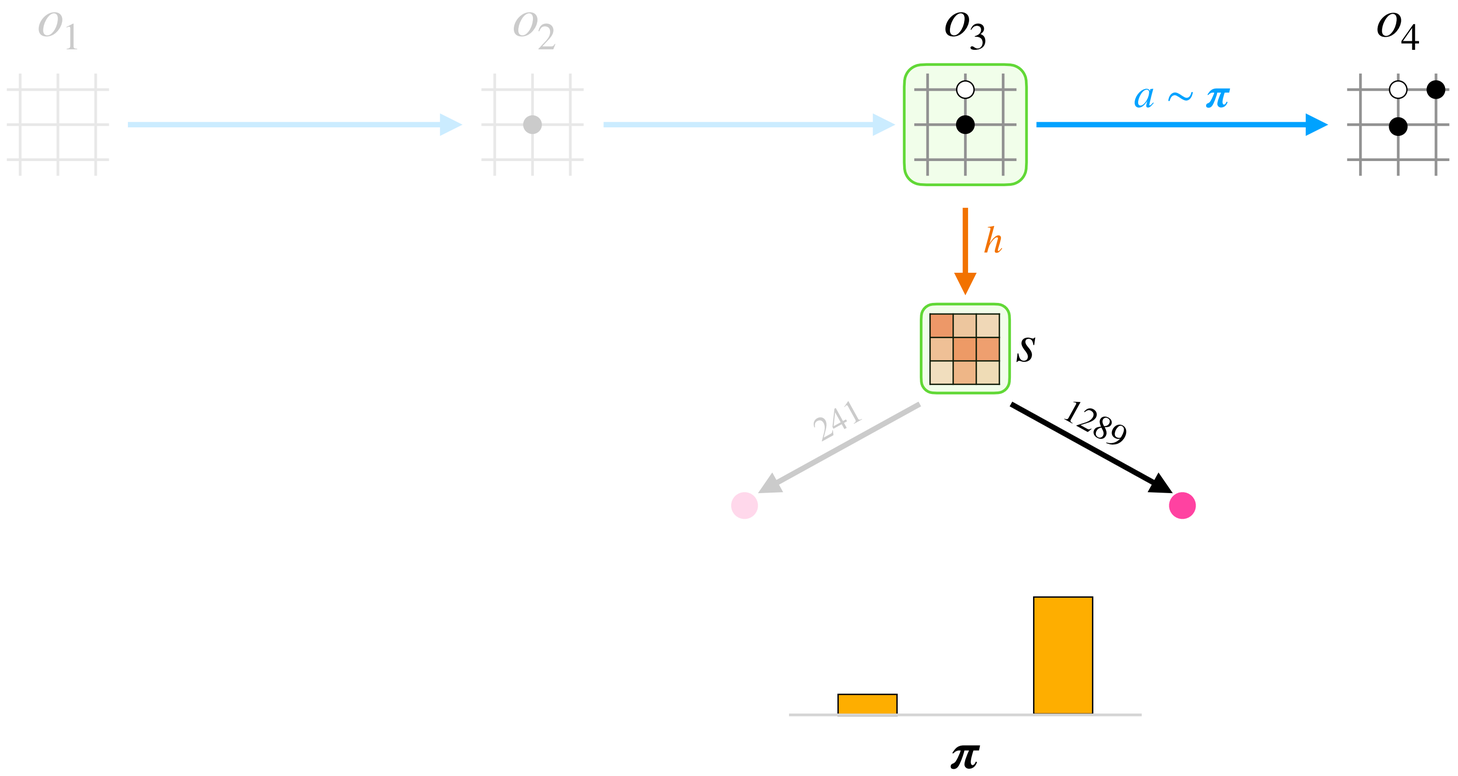
representation h

prediction f

afterstate α

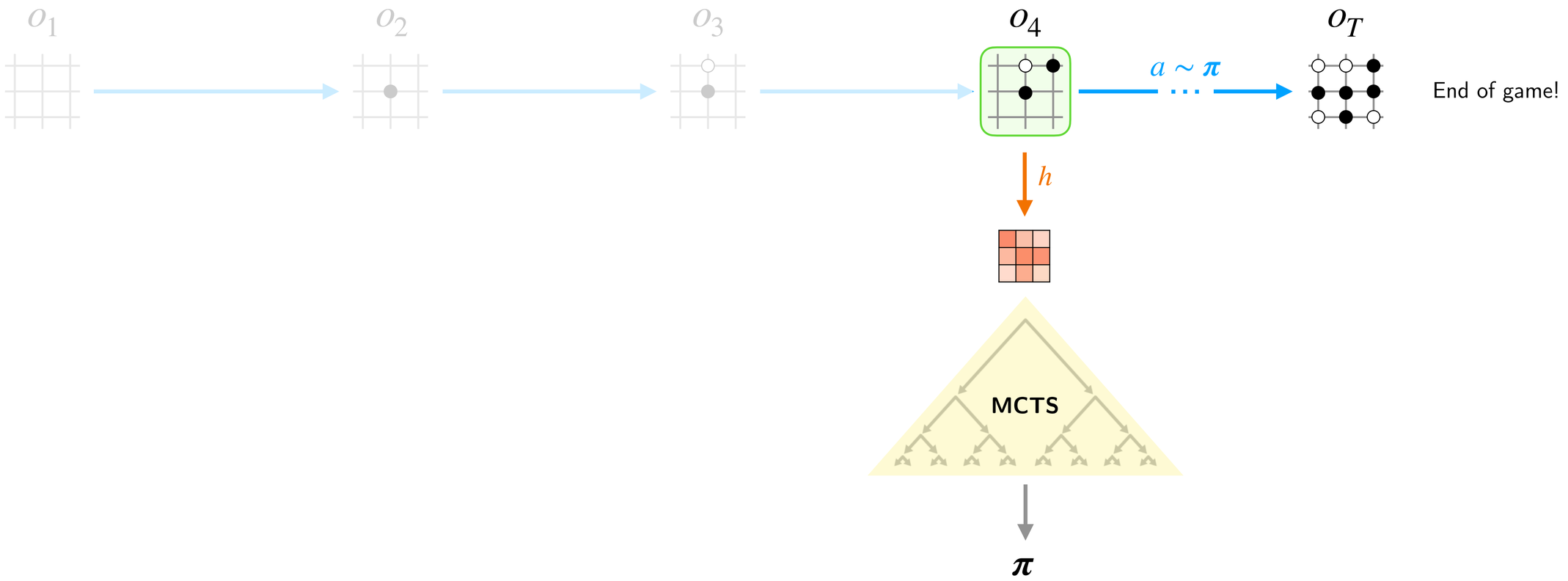
Stochastic MuZero

Pretend it's some kind of stochastic Go variant.



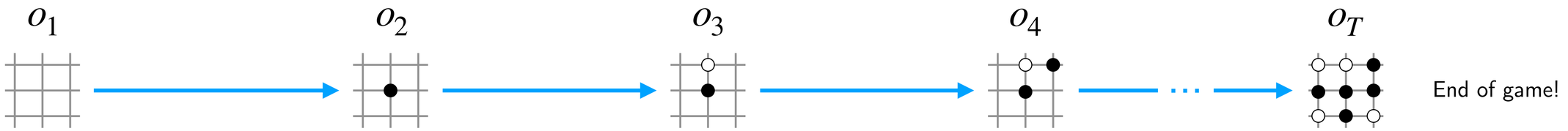
Stochastic MuZero

Pretend it's some kind of stochastic Go variant.



Stochastic MuZero

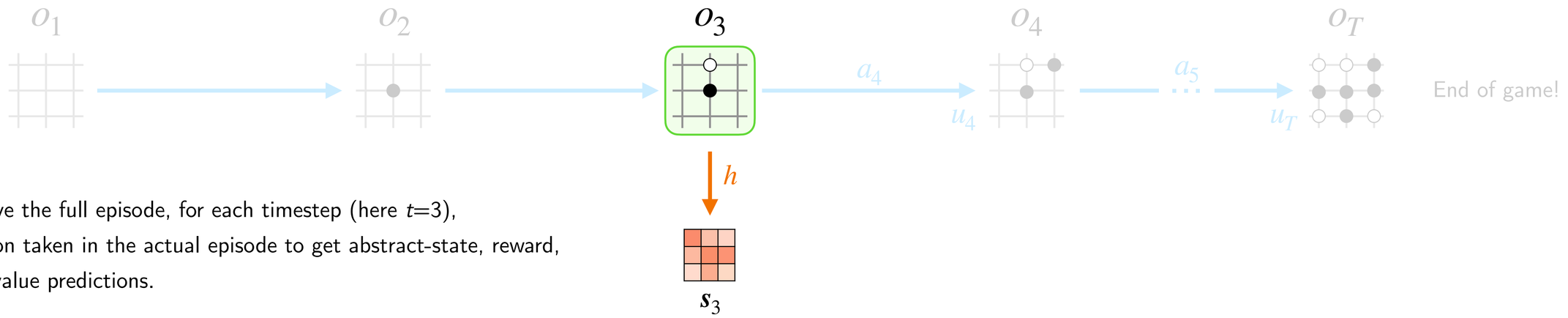
Pretend it's some kind of stochastic Go variant.



Once we have the full episode, for each timestep (here t=3),
use the action taken in the actual episode to get abstract-state, reward,
policy, and value predictions.

Stochastic MuZero

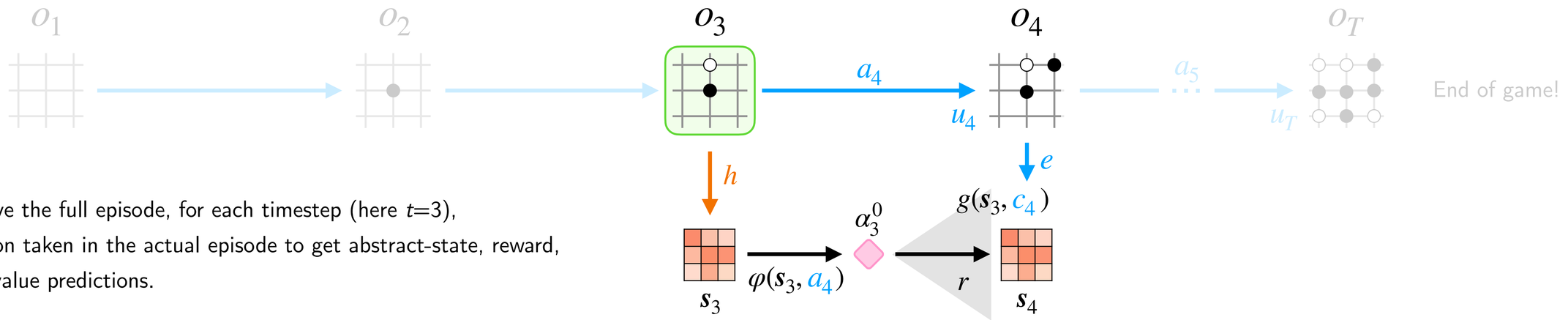
Pretend it's some kind of stochastic Go variant.



Once we have the full episode, for each timestep (here t=3),
use the action taken in the actual episode to get abstract-state, reward,
policy, and value predictions.

Stochastic MuZero

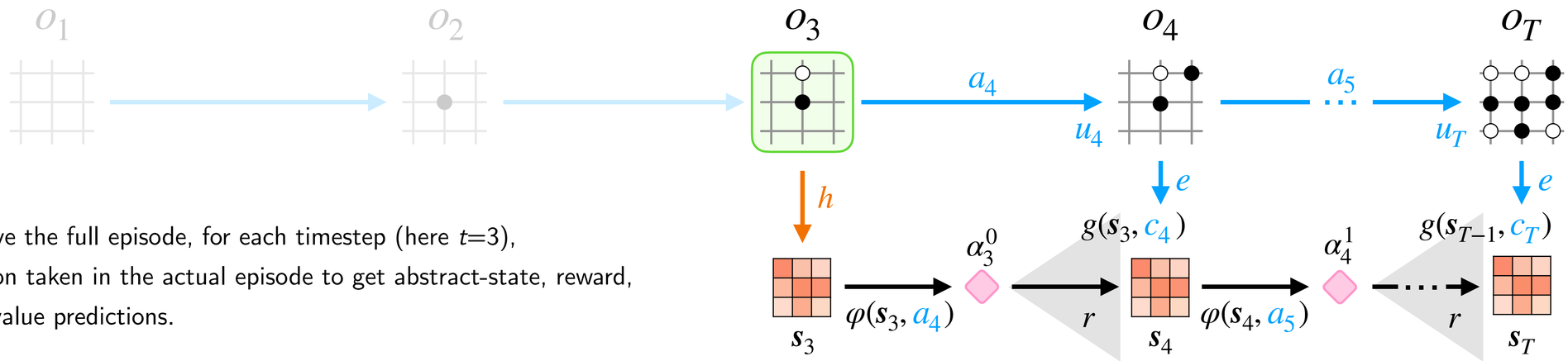
Pretend it's some kind of stochastic Go variant.



Once we have the full episode, for each timestep (here t=3),
use the action taken in the actual episode to get abstract-state, reward,
policy, and value predictions.

Stochastic MuZero

Pretend it's some kind of stochastic Go variant.

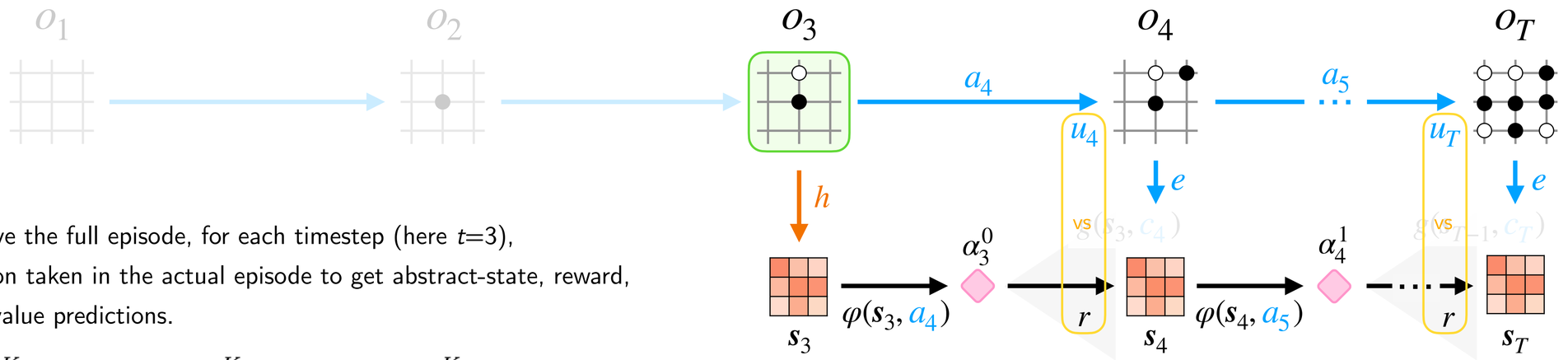


End of game!

Once we have the full episode, for each timestep (here t=3),
use the action taken in the actual episode to get abstract-state, reward,
policy, and value predictions.

Stochastic MuZero

Pretend it's some kind of stochastic Go variant.



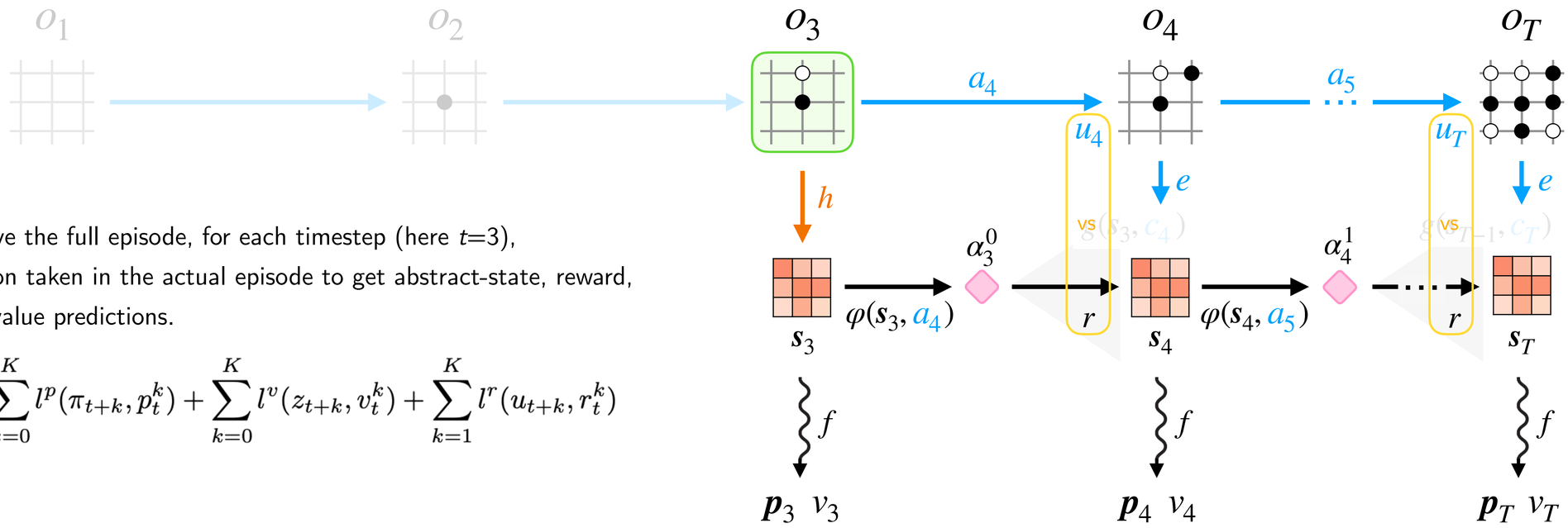
End of game!

Once we have the full episode, for each timestep (here t=3),
use the action taken in the actual episode to get abstract-state, reward,
policy, and value predictions.

$$J(\theta) = \sum_{k=0}^{K} l^p(\pi_{t+k}, p_t^k) + \sum_{k=0}^{K} l^v(z_{t+k}, v_t^k) + \sum_{k=1}^{K} l^r(u_{t+k}, r_t^k)$$

Stochastic MuZero

Pretend it's some kind of stochastic Go variant.



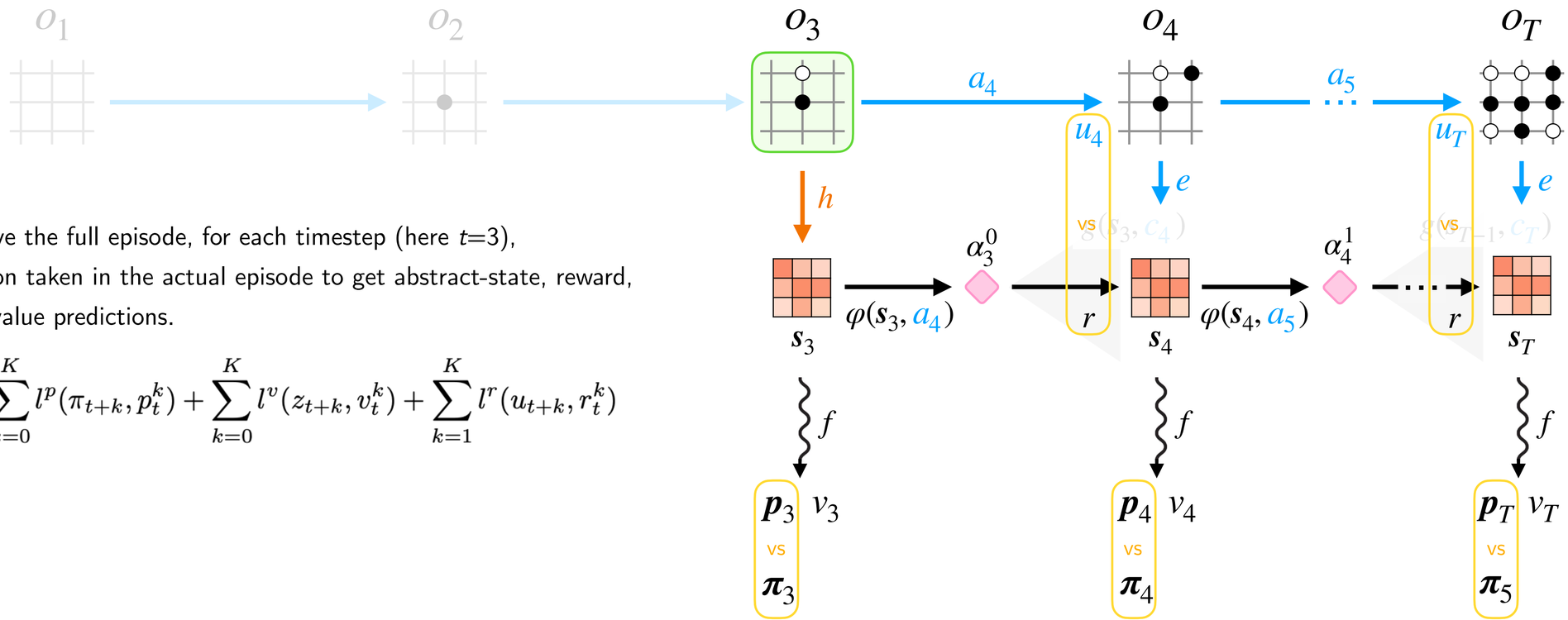
End of game!

Once we have the full episode, for each timestep (here t=3),
use the action taken in the actual episode to get abstract-state, reward,
policy, and value predictions.

$$J(\theta) = \sum_{k=0}^{K} l^p(\pi_{t+k}, p_t^k) + \sum_{k=0}^{K} l^v(z_{t+k}, v_t^k) + \sum_{k=1}^{K} l^r(u_{t+k}, r_t^k)$$

Stochastic MuZero

Pretend it's some kind of stochastic Go variant.



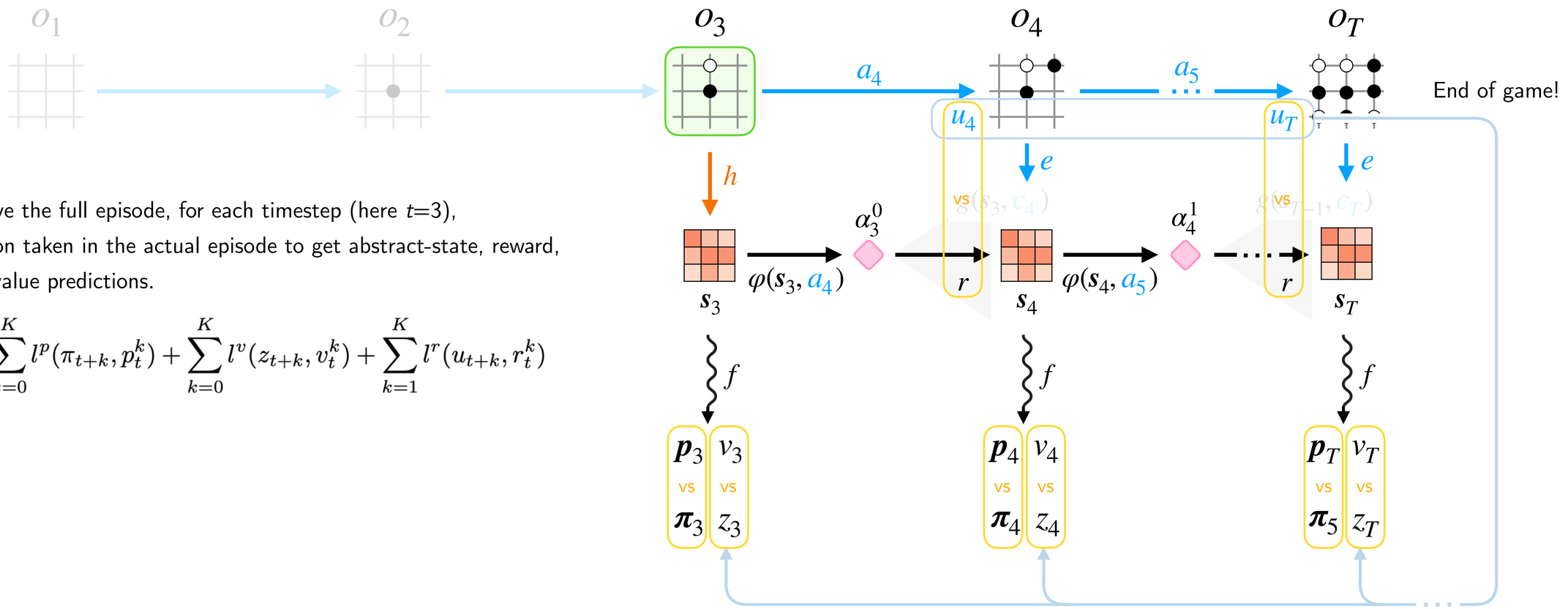
End of game!

Once we have the full episode, for each timestep (here t=3),
use the action taken in the actual episode to get abstract-state, reward,
policy, and value predictions.

$$J(\theta) = \sum_{k=0}^{K} l^p(\pi_{t+k}, p_t^k) + \sum_{k=0}^{K} l^v(z_{t+k}, v_t^k) + \sum_{k=1}^{K} l^r(u_{t+k}, r_t^k)$$

Stochastic MuZero

Pretend it's some kind of stochastic Go variant.

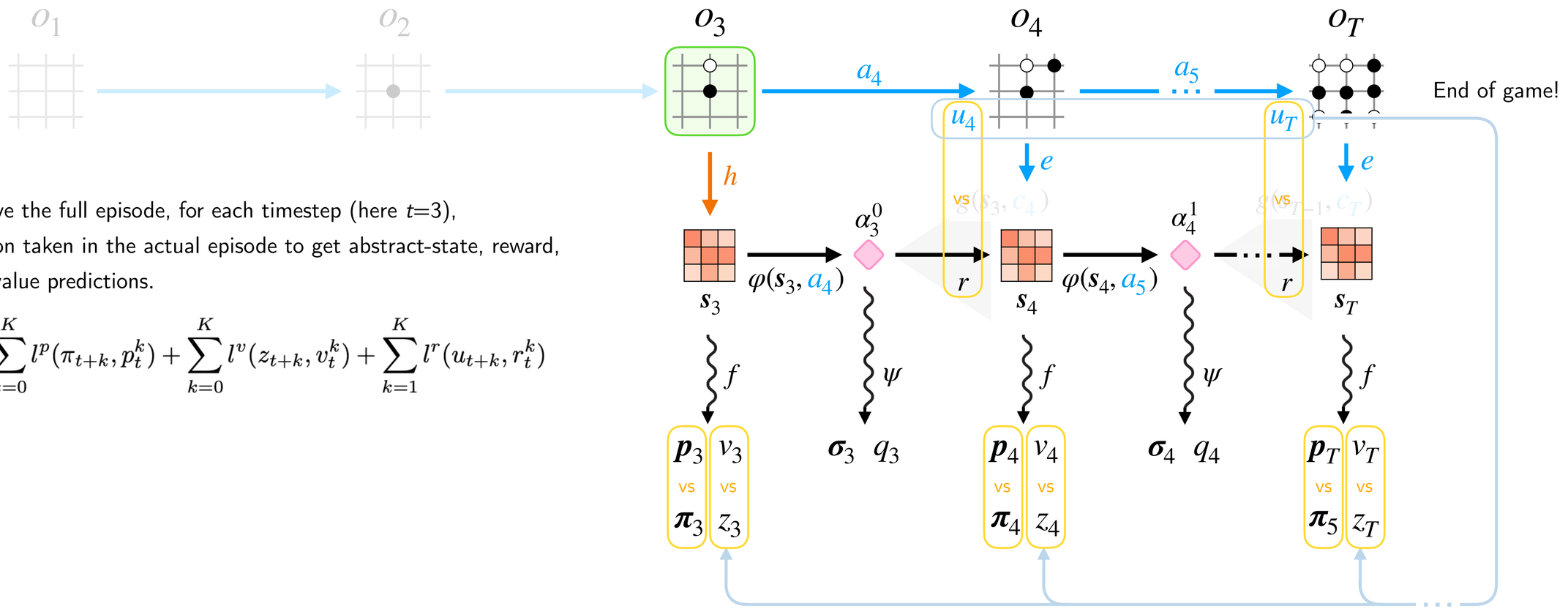


Once we have the full episode, for each timestep (here t=3),
use the action taken in the actual episode to get abstract-state, reward,
policy, and value predictions.

$$J(\theta) = \sum_{k=0}^{K} l^p(\pi_{t+k}, p_t^k) + \sum_{k=0}^{K} l^v(z_{t+k}, v_t^k) + \sum_{k=1}^{K} l^r(u_{t+k}, r_t^k)$$

Stochastic MuZero

Pretend it's some kind of stochastic Go variant.

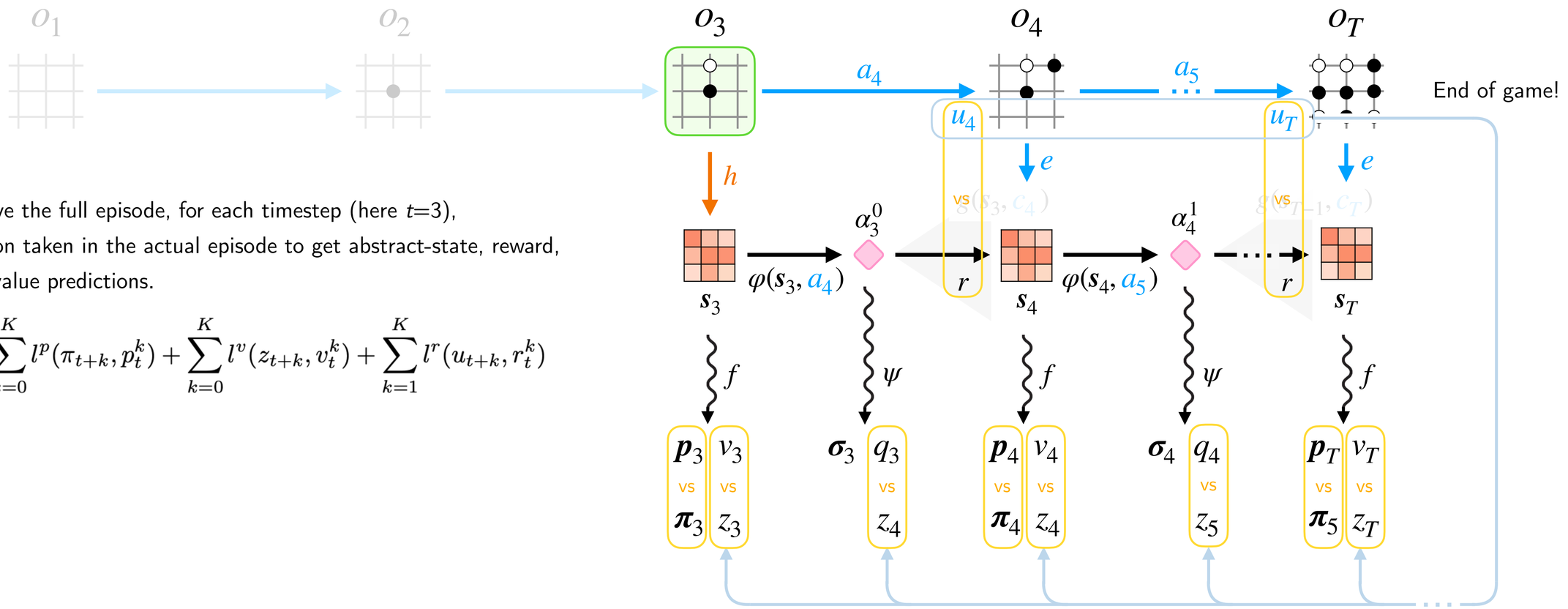


Once we have the full episode, for each timestep (here t=3),
use the action taken in the actual episode to get abstract-state, reward,
policy, and value predictions.

$$J(\theta) = \sum_{k=0}^{K} l^p(\pi_{t+k}, p_t^k) + \sum_{k=0}^{K} l^v(z_{t+k}, v_t^k) + \sum_{k=1}^{K} l^r(u_{t+k}, r_t^k)$$

Stochastic MuZero

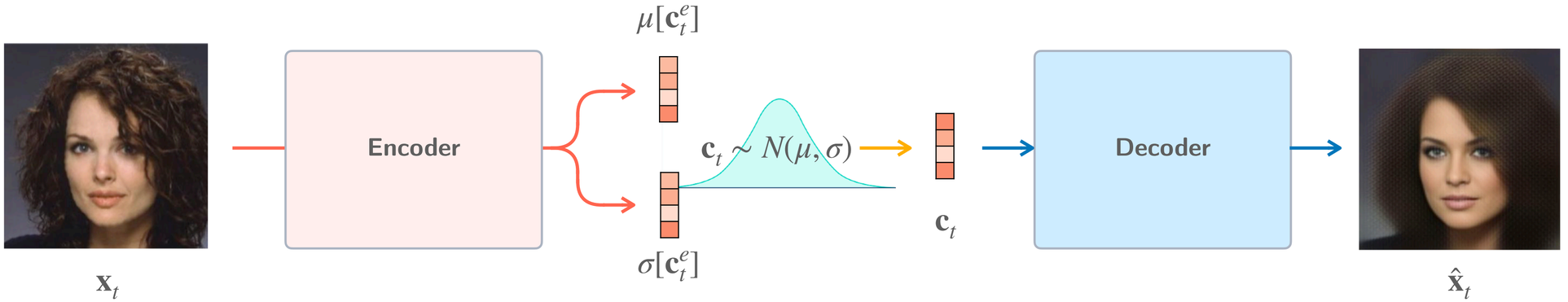
Pretend it's some kind of stochastic Go variant.



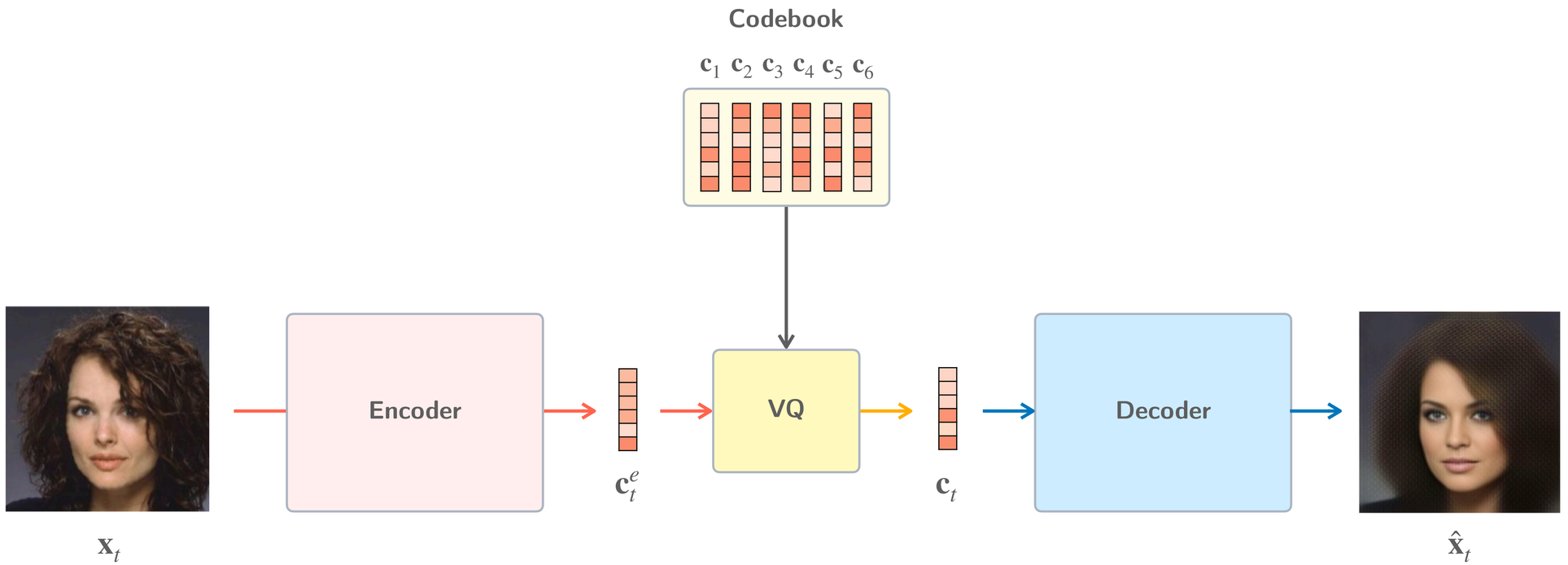
Once we have the full episode, for each timestep (here t=3),
use the action taken in the actual episode to get abstract-state, reward,
policy, and value predictions.

$$J(\theta) = \sum_{k=0}^{K} l^p(\pi_{t+k}, p_t^k) + \sum_{k=0}^{K} l^v(z_{t+k}, v_t^k) + \sum_{k=1}^{K} l^r(u_{t+k}, r_t^k)$$

Variational Autoencoder (VAE)



Vector-Quantized Variational Autoencoder (VQ-VAE)

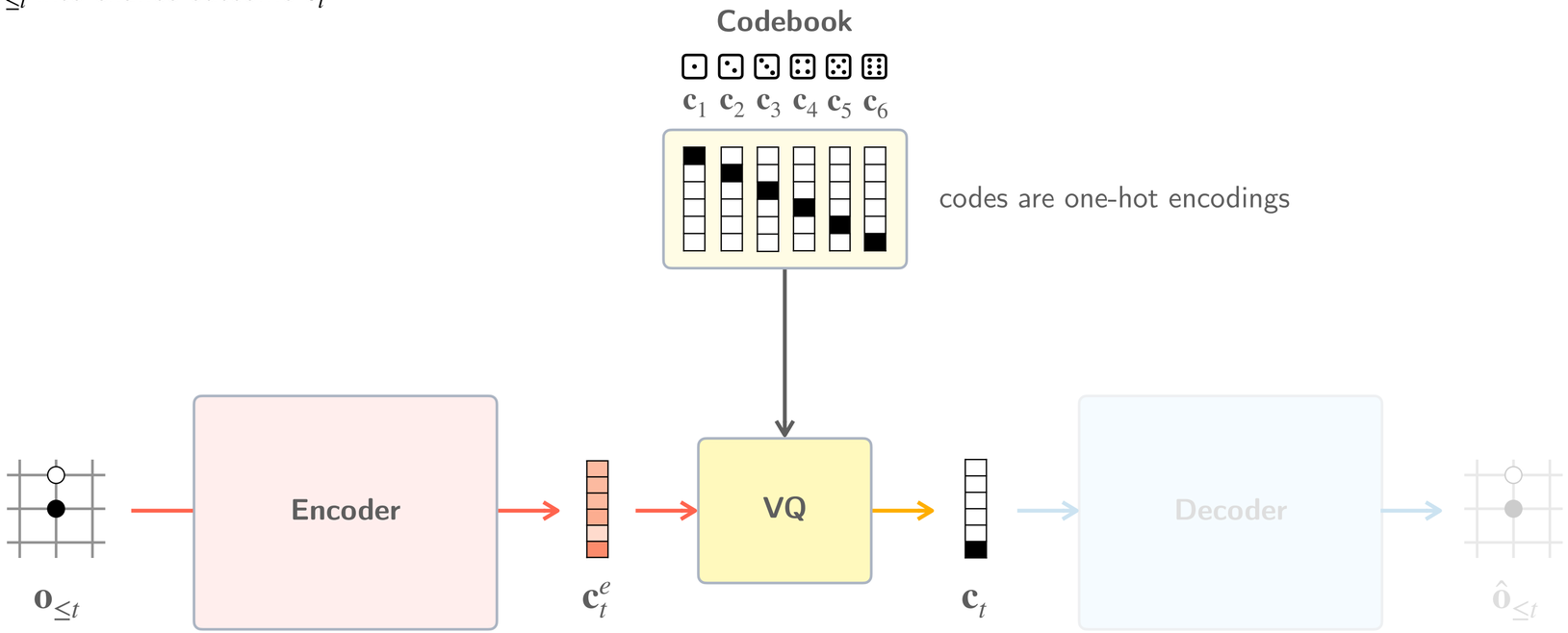


$$\mathbf{c}_t = \mathbf{c}_{k_t}$$

where $k_t = \arg\min_i \|\mathbf{c}_i - \mathbf{c}_t^e\|$

Vector-Quantized Variational Autoencoder (VQ-VAE)

VQ-VAE to encode observation $\mathbf{o}_{\leq t}$ into chance outcome $\mathbf{c}_t$



$\mathbf{c}_t^e$ is probability vector over codes

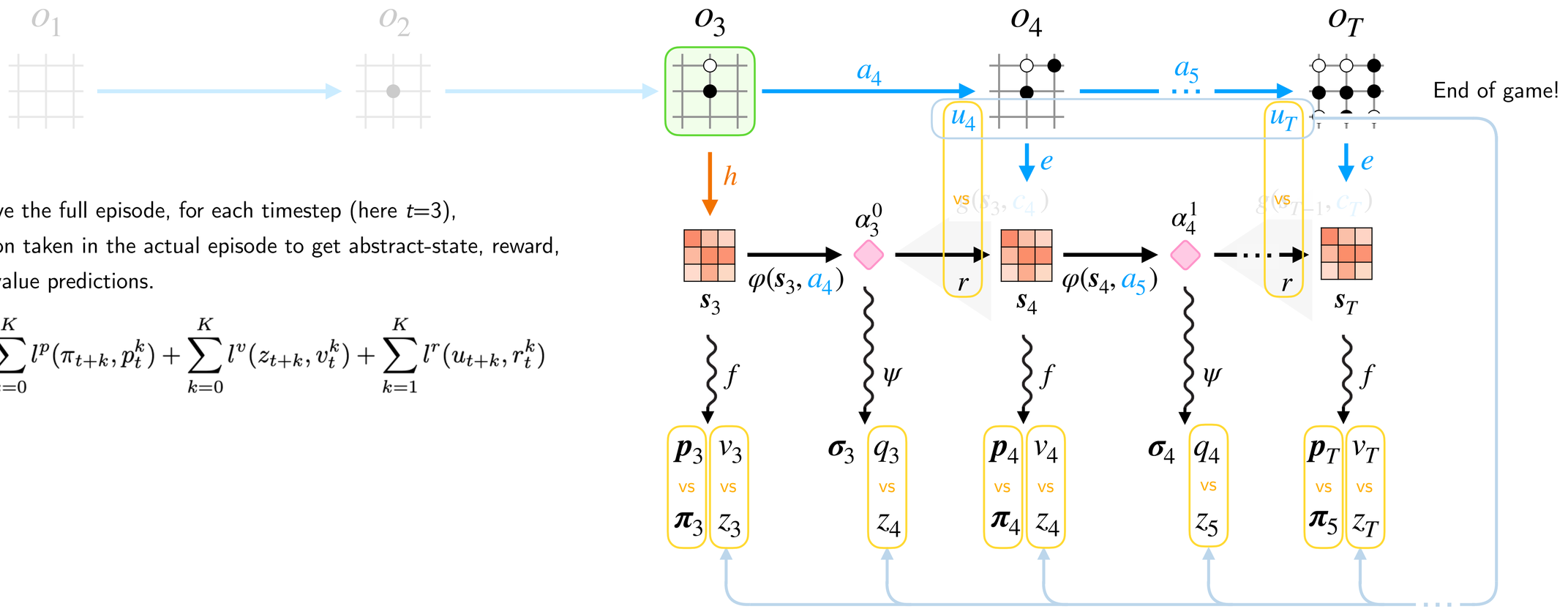
$\mathbf{c}_t$ is the corresponding one-hot vector

$$\mathbf{c}_t = \mathbf{c}_{k_t}$$

where $k_t = \arg\min_i \|\mathbf{c}_i - \mathbf{c}_t^e\|$

Stochastic MuZero

Pretend it's some kind of stochastic Go variant.

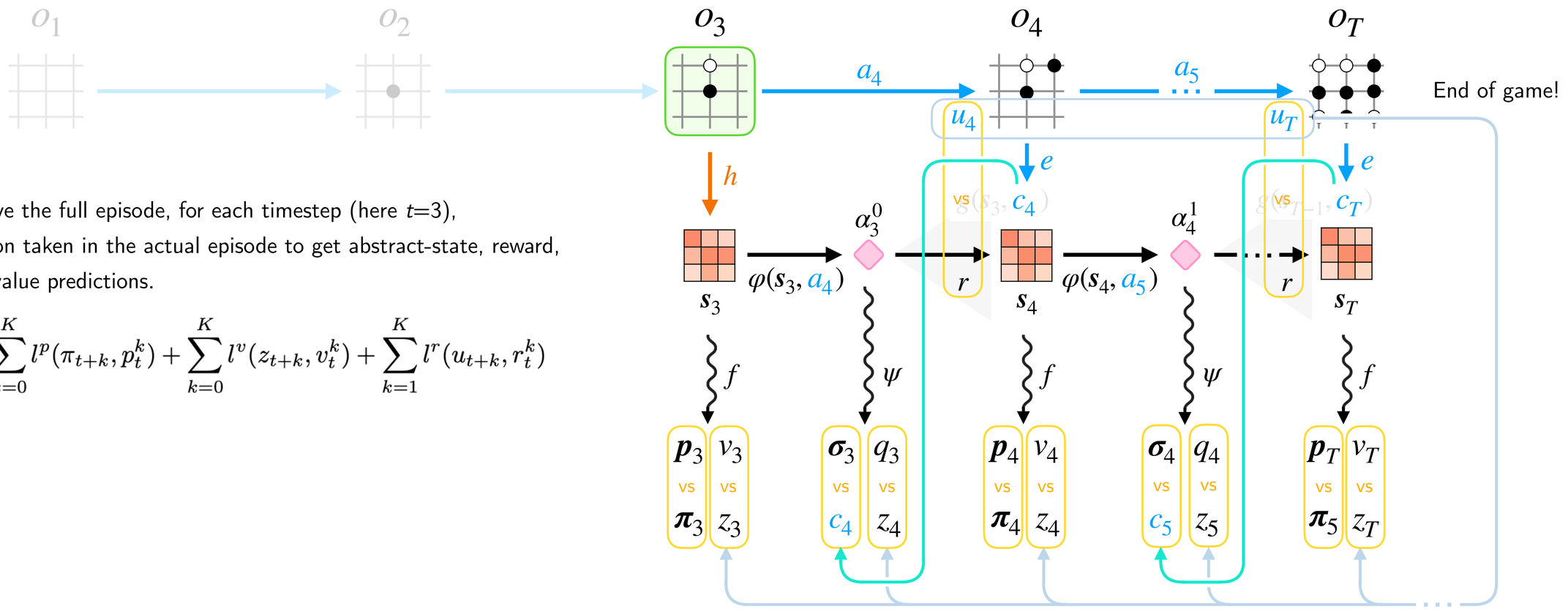


Once we have the full episode, for each timestep (here t=3),
use the action taken in the actual episode to get abstract-state, reward,
policy, and value predictions.

$$J(\theta) = \sum_{k=0}^{K} l^p(\pi_{t+k}, p_t^k) + \sum_{k=0}^{K} l^v(z_{t+k}, v_t^k) + \sum_{k=1}^{K} l^r(u_{t+k}, r_t^k)$$

Stochastic MuZero

Pretend it's some kind of stochastic Go variant.



Once we have the full episode, for each timestep (here t=3),
use the action taken in the actual episode to get abstract-state, reward, policy, and value predictions.

$$J(\theta) = \sum_{k=0}^{K} l^p(\pi_{t+k}, p_t^k) + \sum_{k=0}^{K} l^v(z_{t+k}, v_t^k) + \sum_{k=1}^{K} l^r(u_{t+k}, r_t^k)$$