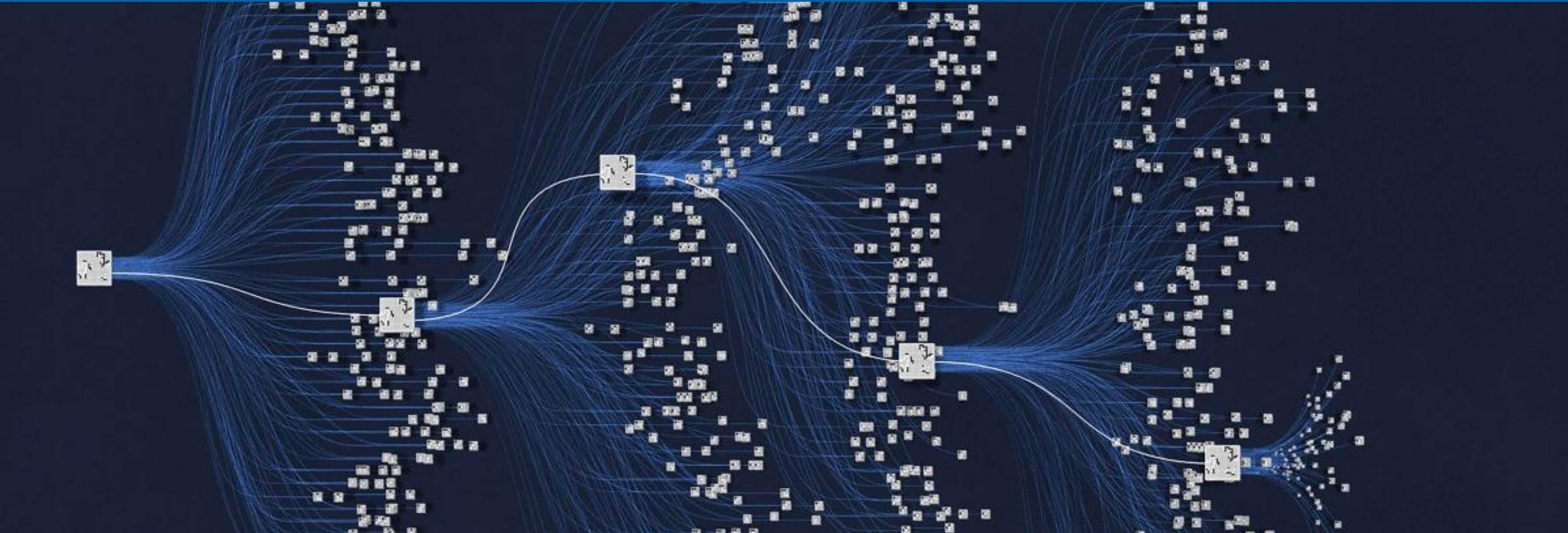
universität wien

ALPHAZERO

Timo Klein | 01.03.2023



Model-Based Reinforcement Learning

- Last time: *Model-free* RL
  ► Learning purely from trial and error
- This time: *Model-based* RL
  ► We know how the environment works

Model-Based Reinforcement Learning

- Last time: *Model-free* RL
  ► Learning purely from trial and error
- This time: *Model-based* RL
  ► We know how the environment works

- Given an MDP $\mathcal{M}=\langle \mathcal{S},\ \mathcal{A},\ P,\ R,\ \gamma \rangle$

Model-Based Reinforcement Learning

- Last time: *Model-free* RL
  ▶ Learning purely from trial and error
- This time: *Model-based* RL
  ▶ We know how the environment works

- Given an MDP $\mathcal{M}=\langle \mathcal{S},\ \mathcal{A},\ P,\ R,\ \gamma \rangle$

Dynamics are known!

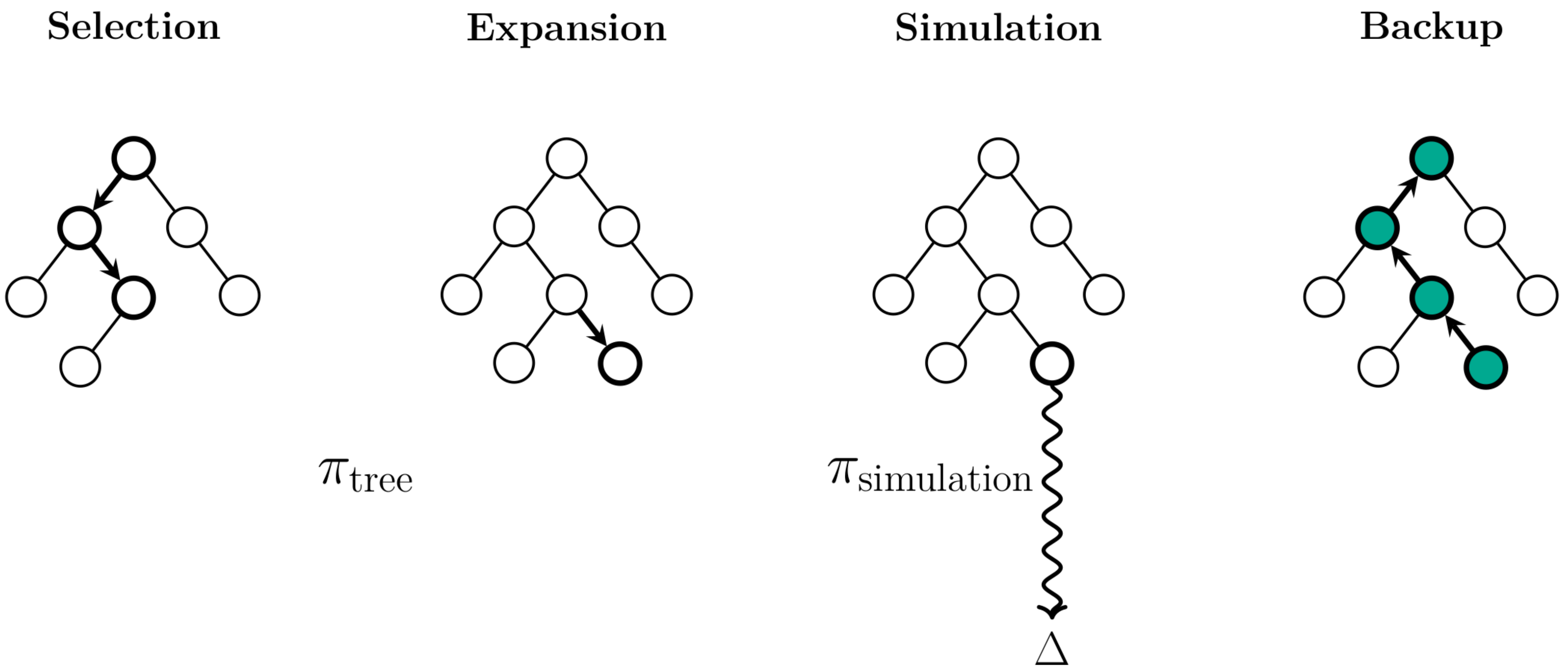
Monte Carlo Tree Search

- Decision-time *planning* algorithm
- Uses heuristic search to build an *asymmetric* search tree

Monte Carlo Tree Search (MCTS)

- Decision-time *planning* algorithm
- Uses heuristic search to build an *asymmetric* search tree
- **Nice properties**
  Anytime (Can always stop and get something)
  Best-first (Selects the best known action)
  Human-like planning (Gets better with more thinking)
  Diminishing returns (Already pretty good with few iterations)

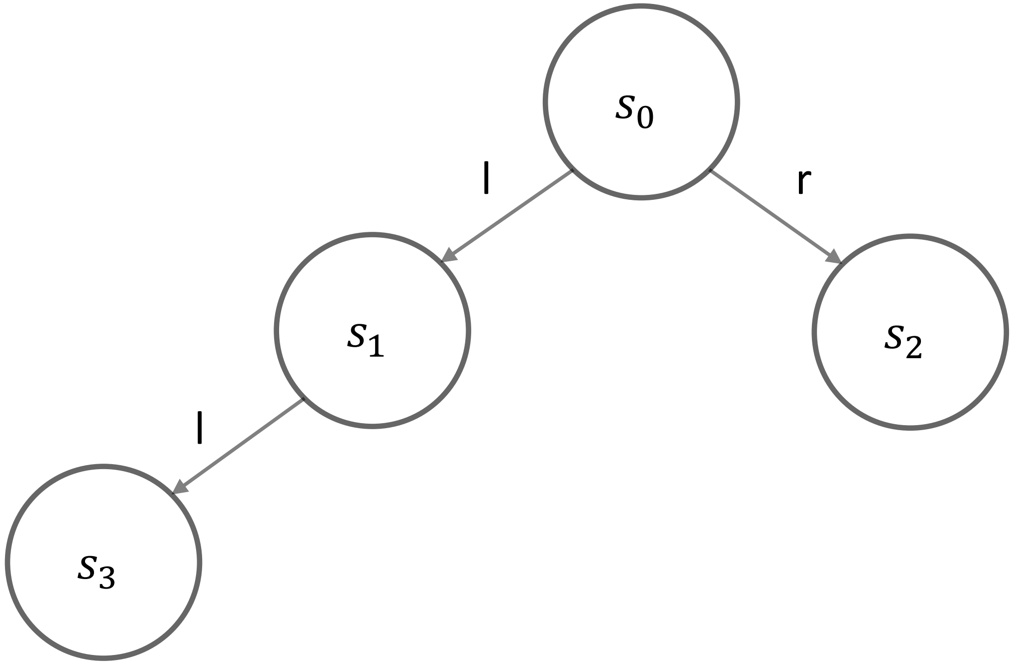
MCTS: PHASES



Selection
Expansion
Simulation
Backup
π_{tree}
$\pi_{\text{simulation}}$
Δ

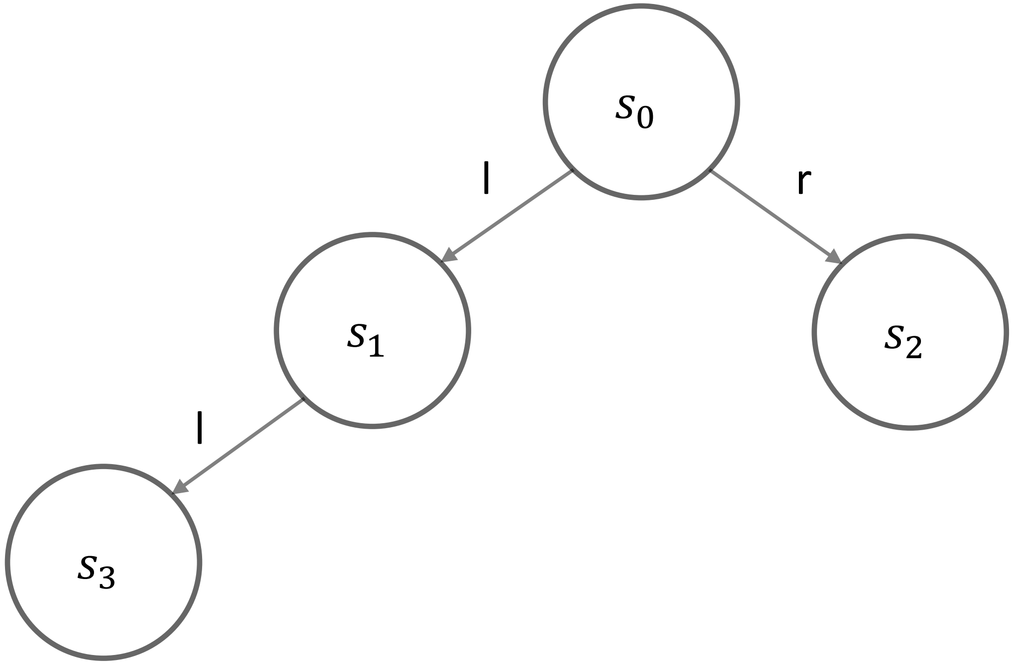
MCTS: Preliminaries

- Nodes: States
- Edges: State-action pairs



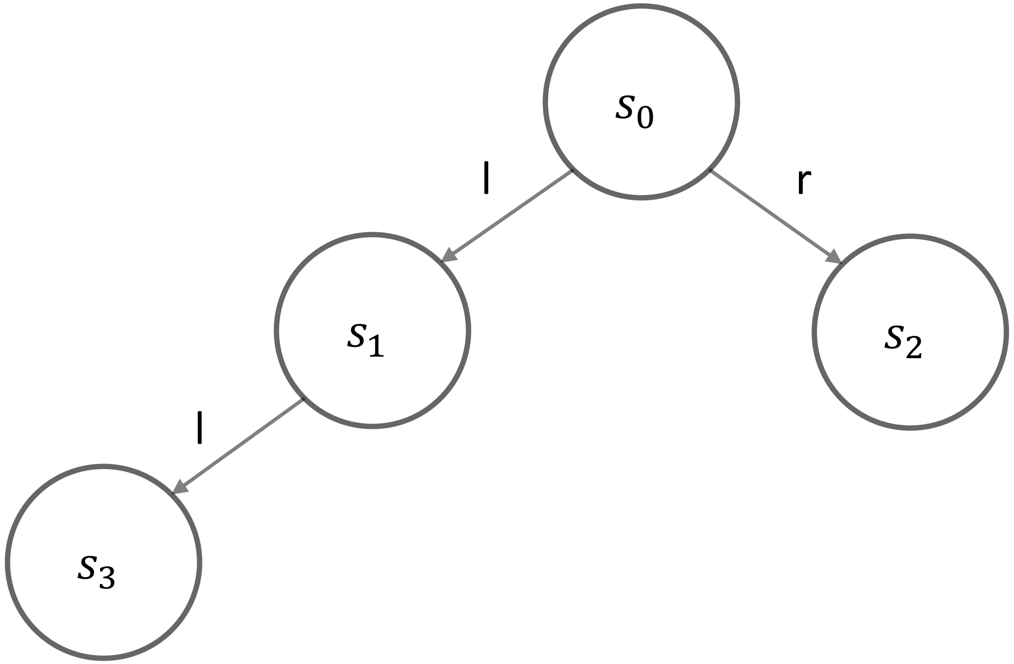
MCTS: PRELIMINARIES

- Nodes: States
- Edges: State-action pairs
- s_0: Root node (actual current environment state)



MCTS: Preliminaries

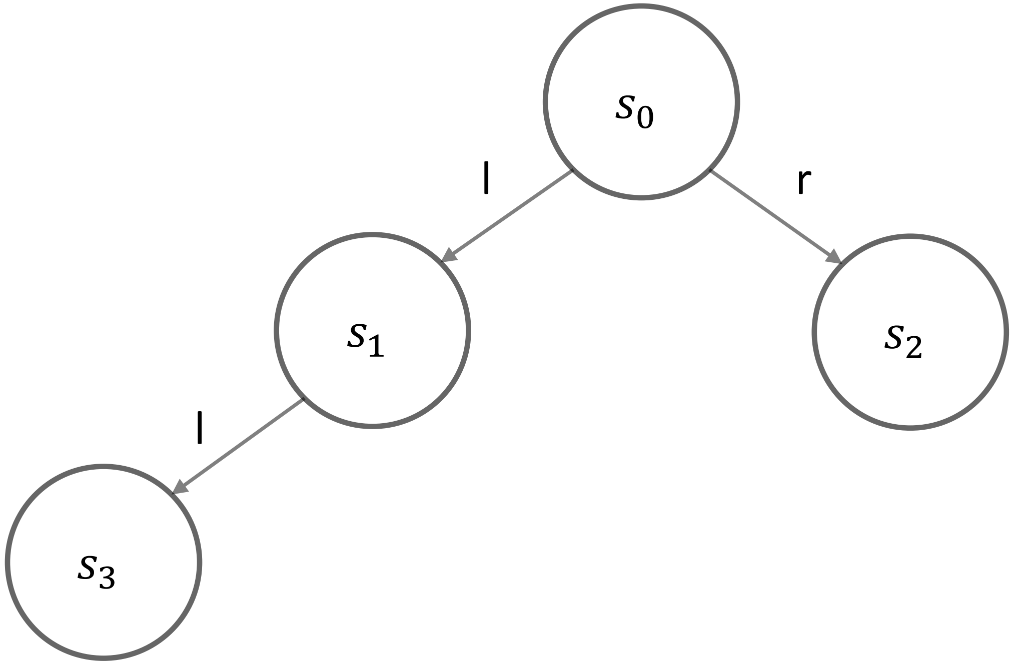
- Nodes: States
- Edges: State-action pairs
- s_0: Root node (actual current environment state)
- Each edge stores
  $N(s,a)$: Visitation count
  $W(s,a)$: Total action value
  $Q(s,a)$: Mean action value



MCTS: SELECTION

- At each node, select actions according to

$$UCT(s,a) = Q(s,a) + c\frac{\sqrt{\sum_b N(s,b)}}{1 + N(s,a)}$$



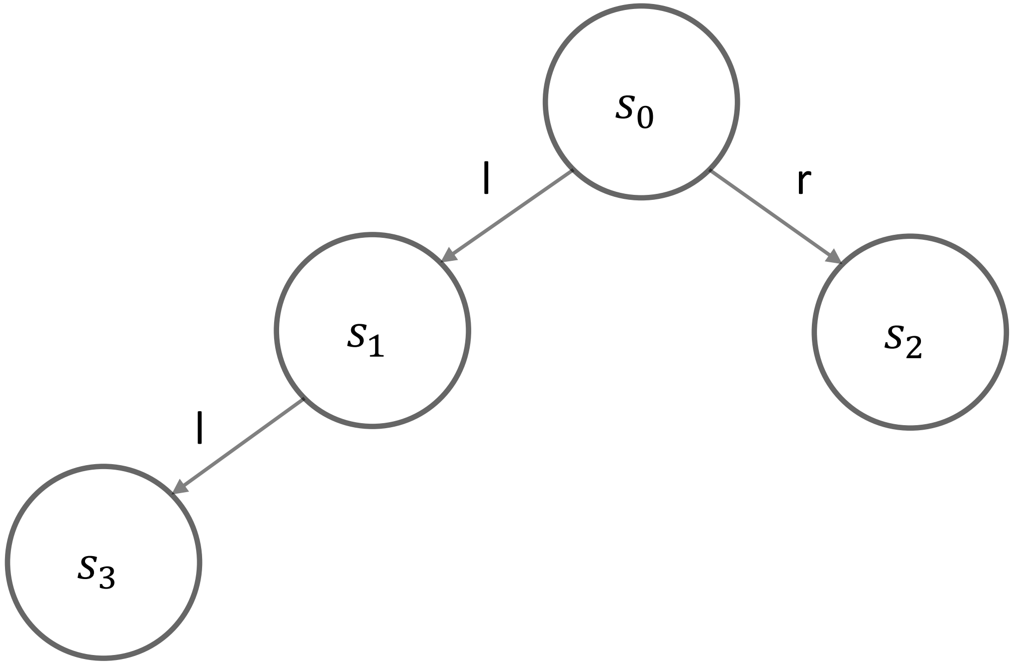
universität wien

MCTS: Selection

- At each node, select actions according to

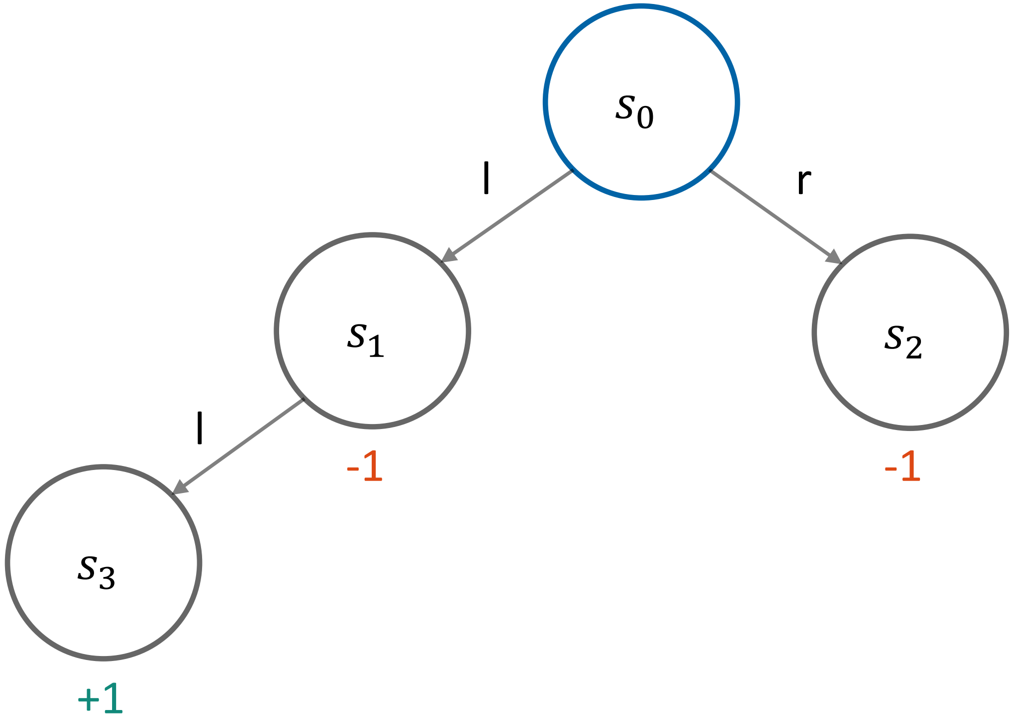
$$UCT(s,a) = Q(s,a) + c\frac{\sqrt{\sum_b N(s,b)}}{1+N(s,a)}$$

Upper confidence bound based on Hoeffding’s inequality



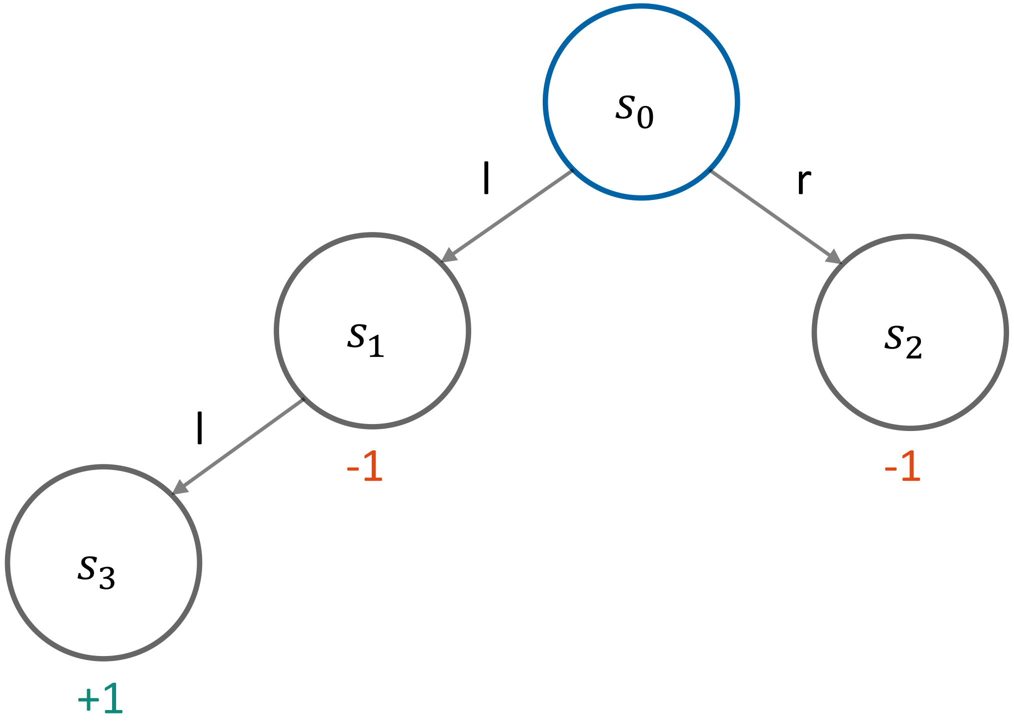
MCTS: Selection Example

Statistic	Value
$N(s_0, l)$	2
$N(s_0, r)$	1
$W(s_0, l)$	0
$W(s_0, r)$	-1
$Q(s_0, l)$	$1\ (0)$
$Q(s_0, r)$	$0\ (-1)$



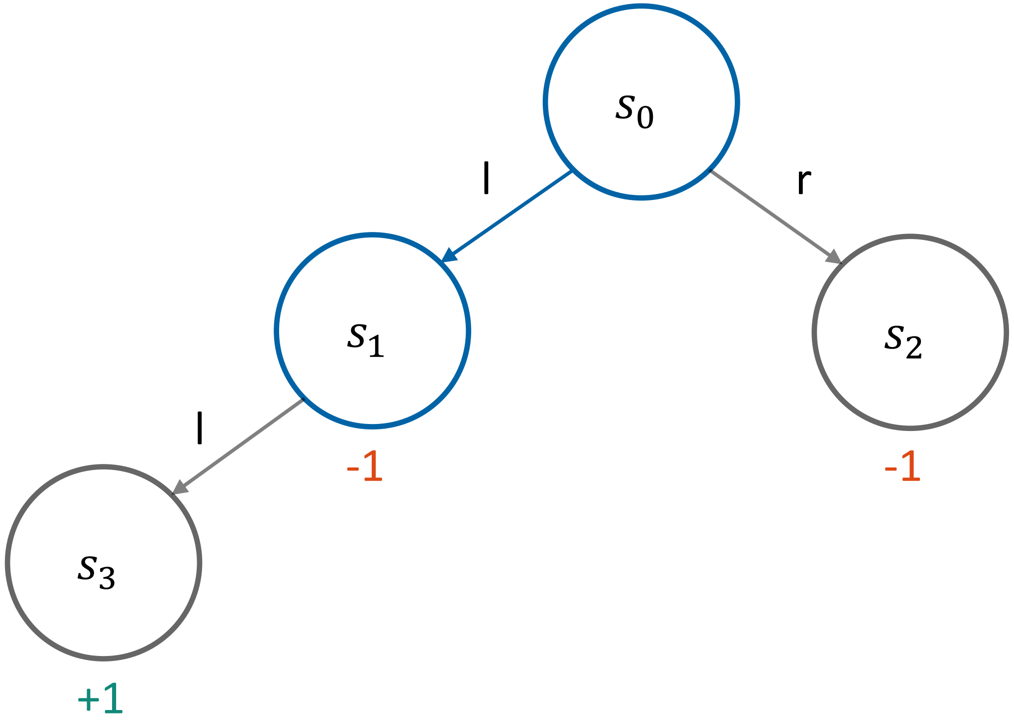
MCTS: SELECTION EXAMPLE

Statistic	Value
$N(s_0, l)$	2
$N(s_0, r)$	1
$W(s_0, l)$	0
$W(s_0, r)$	-1
$Q(s_0, l)$	1
$Q(s_0, r)$	0
$UCT(s_0, l)$	$1 + \frac{\sqrt{3}}{3} \approx 1.577$
$UCT(s_0, r)$	$0 + \frac{\sqrt{3}}{2} \approx 0.866$



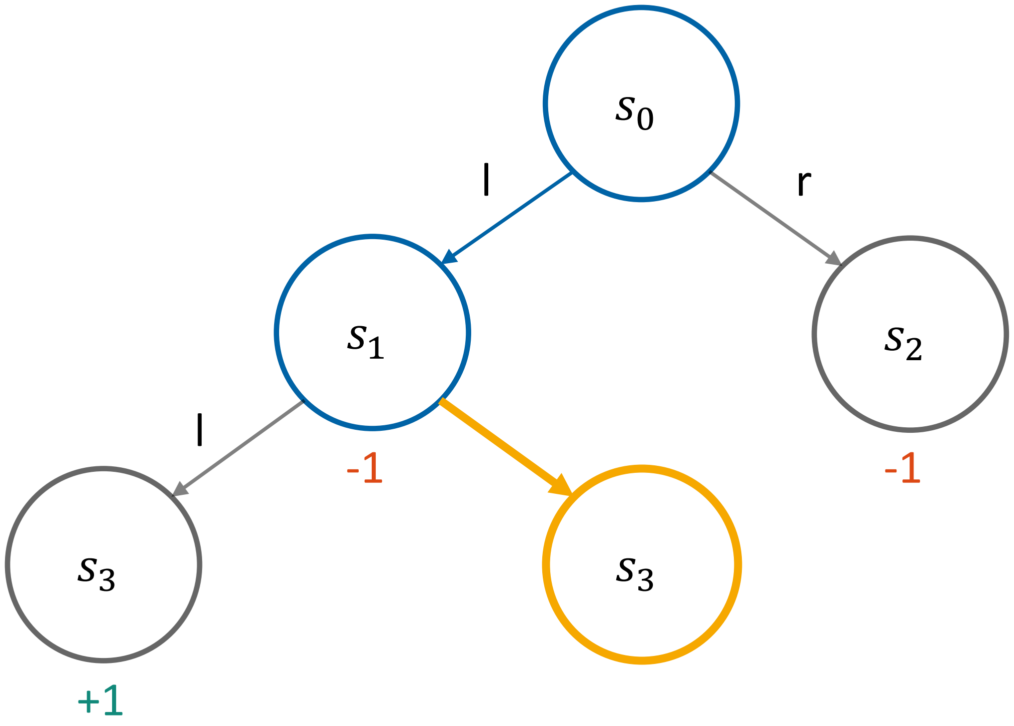
MCTS: Expansion Example

- Action r has not been taken in s_1



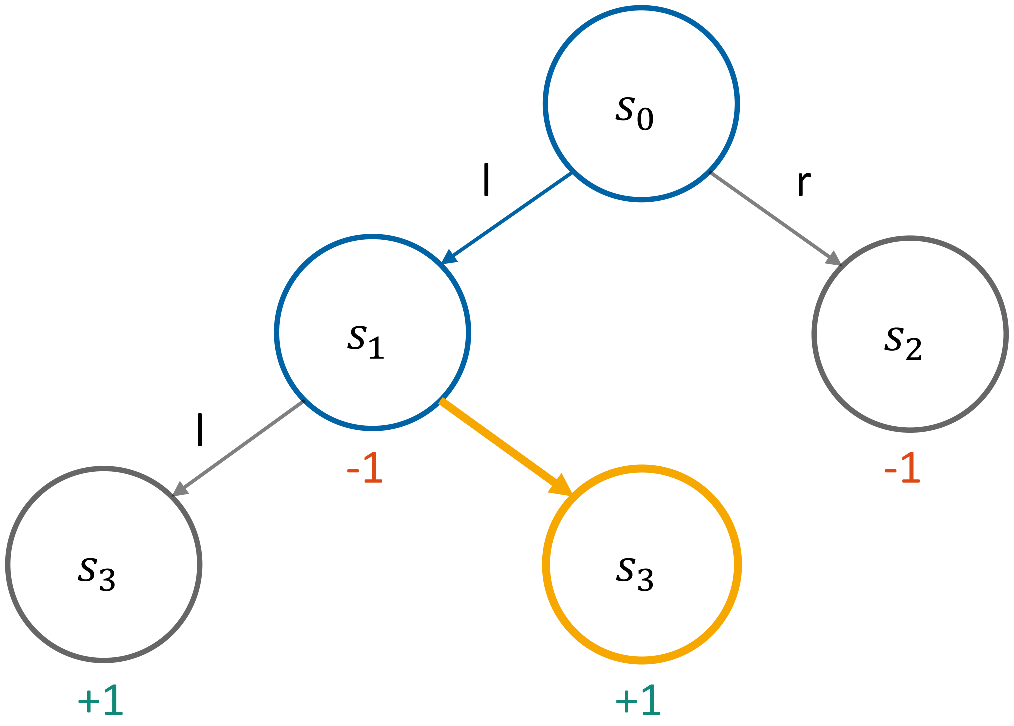
MCTS: Expansion Example

- Action r has not been taken in s_1 ▶ Expand it!



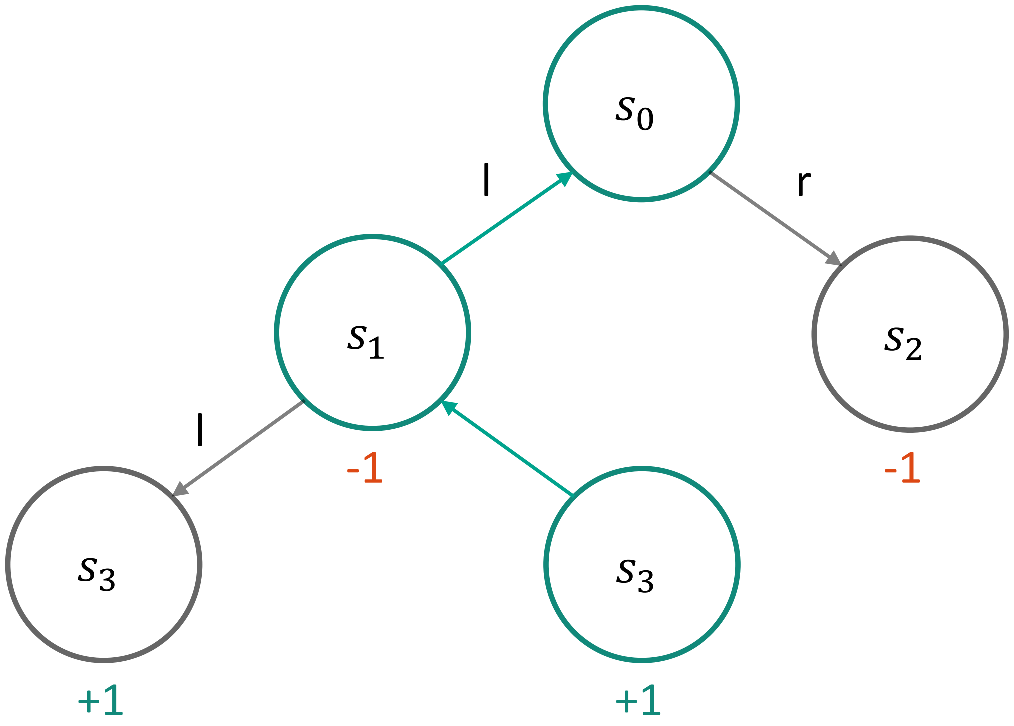
MCTS: Simulation Example

- Run simulation from s_3 until a terminal state is reached
- Cheapest possible way: Uniform random selection



MCTS: Backup Example

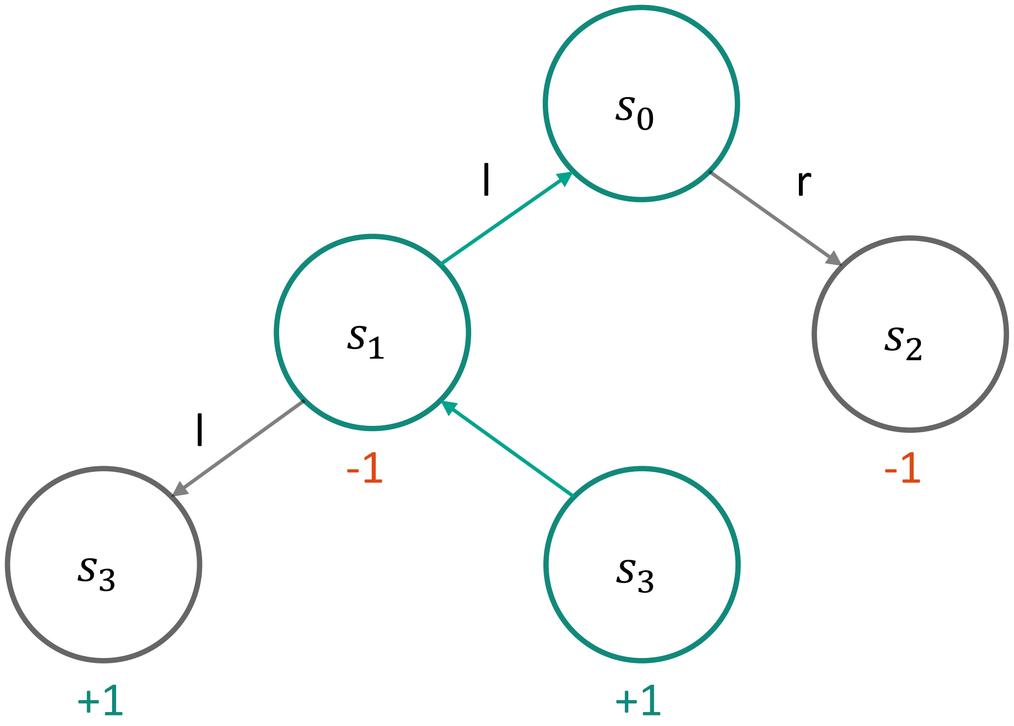
Statistic	Value
$N(s_0, l)$	$2 + 1$
$N(s_0, r)$	1
$W(s_0, l)$	$0 + 1$
$W(s_0, r)$	-1
$Q(s_0, l)$	$1 \left(\frac{1}{3}\right)$
$Q(s_0, r)$	$0\ (-1)$



MCTS: Backup Example

Statistic	Value
$N(s_0, l)$	$2 + 1$
$N(s_0, r)$	1
$W(s_0, l)$	$0 + 1$
$W(s_0, r)$	-1
$Q(s_0, l)$	$1 \left(\frac{1}{3}\right)$
$Q(s_0, r)$	$0\ (-1)$

Action selection: Use action with most visits



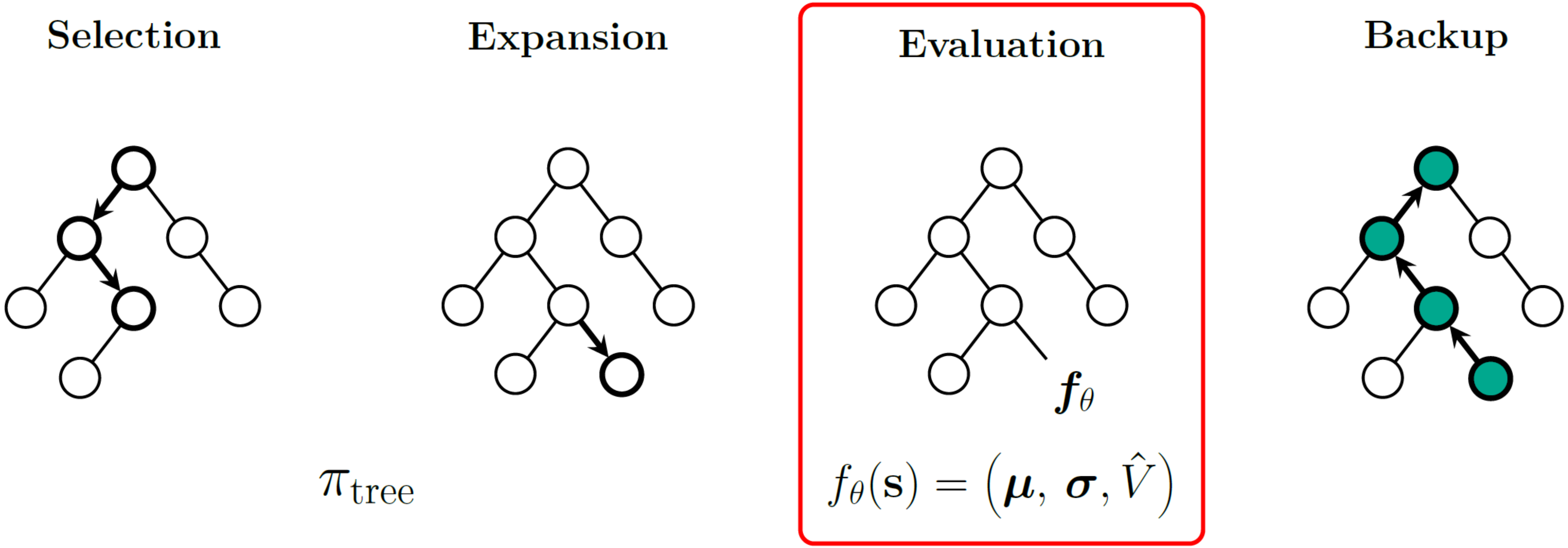
AlphaZero Motivation

- Why doesn't MCTS work for Go?
  - ►Branching factor too large (Chess 35, Go 250)
- High-quality simulations are too costly

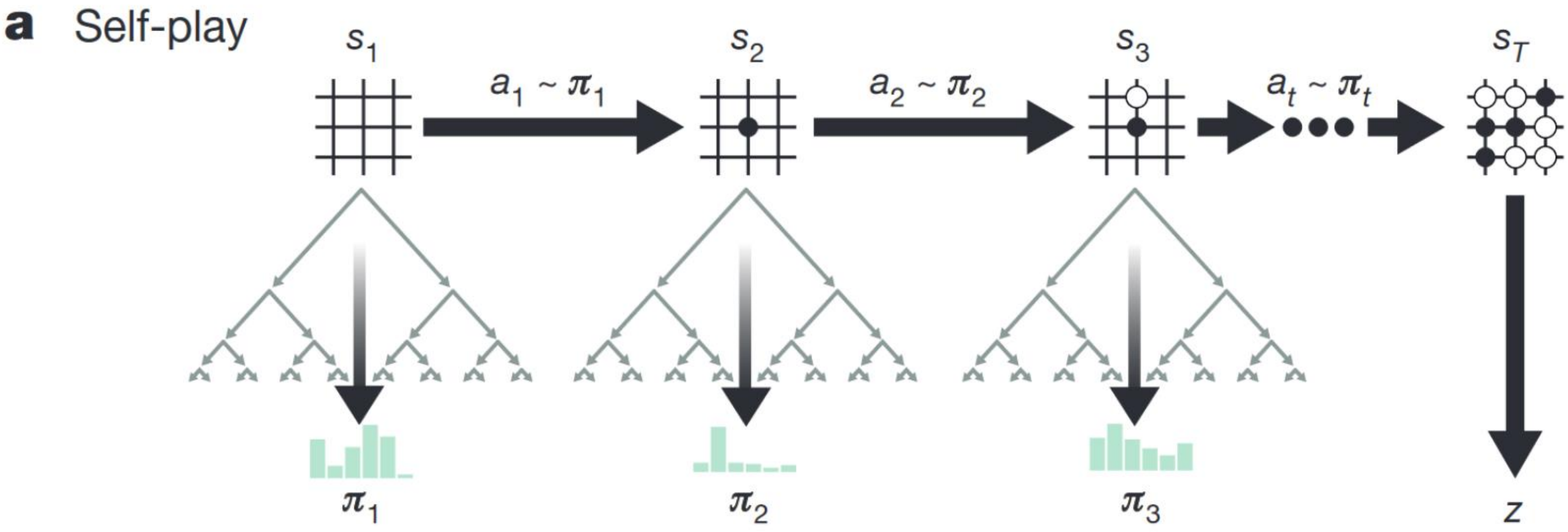
AlphaZero Motivation

- Why doesn't MCTS work for Go?
  ► Branching factor too large (Chess 35, Go 250)
- High-quality simulations are too costly
- *How can the tree search be made more efficient?*
- **Solution: Incorporate "prior" knowledge about move quality with NN**

AlphaZero: Integrating Neural Networks

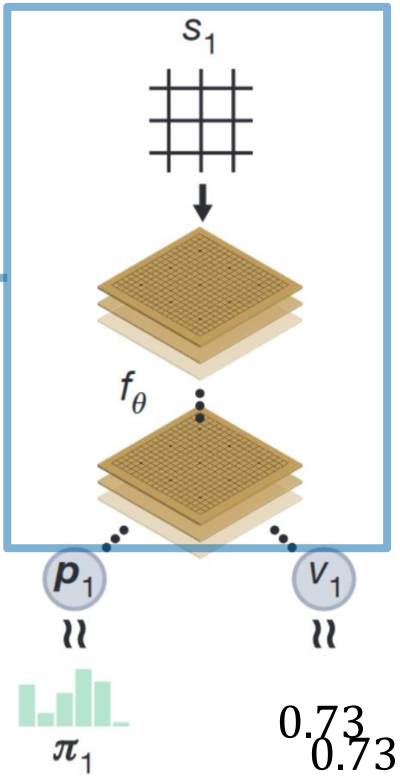


ALPHAZERO: SELF-PLAY



AlphaZero: Network Architecture

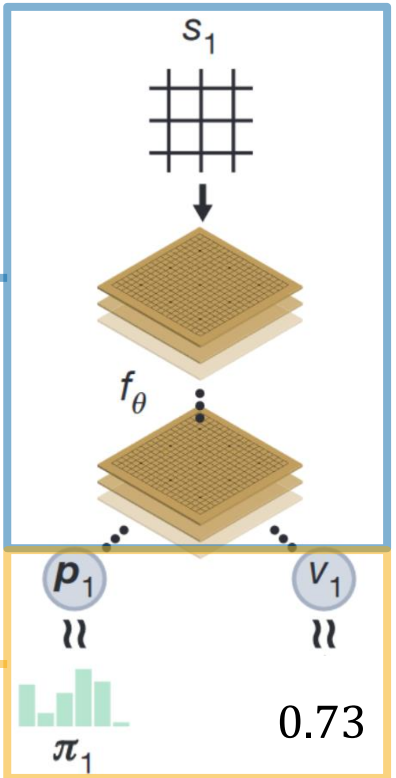
Trunk: ResNet with 20 blocks



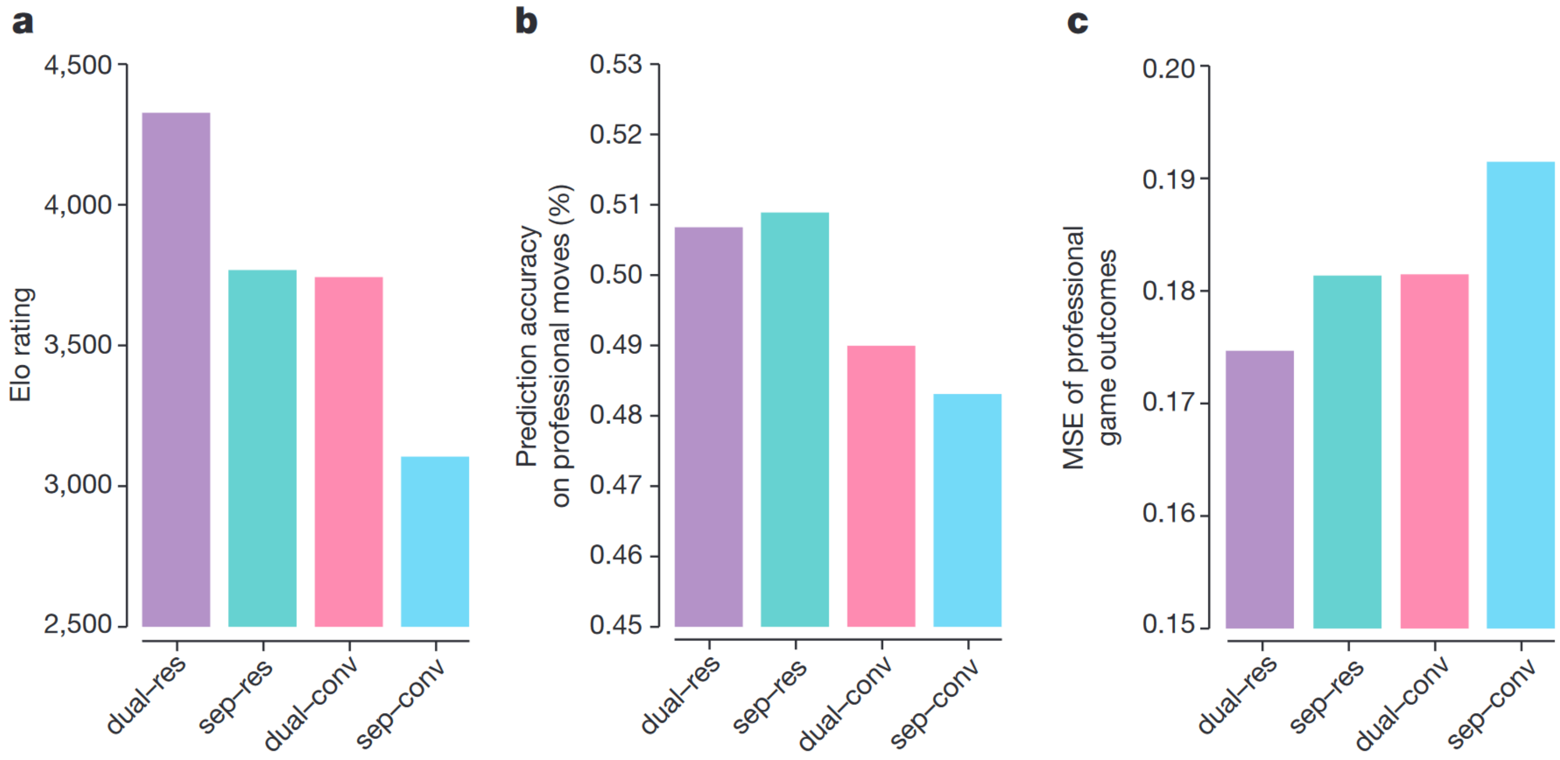
AlphaZero: Network Architecture

Trunk: ResNet with 20 blocks

Policy head: Outputs distribution over moves
Value head: Outputs probability of victory



AlphaZero: Shared Network Improvement

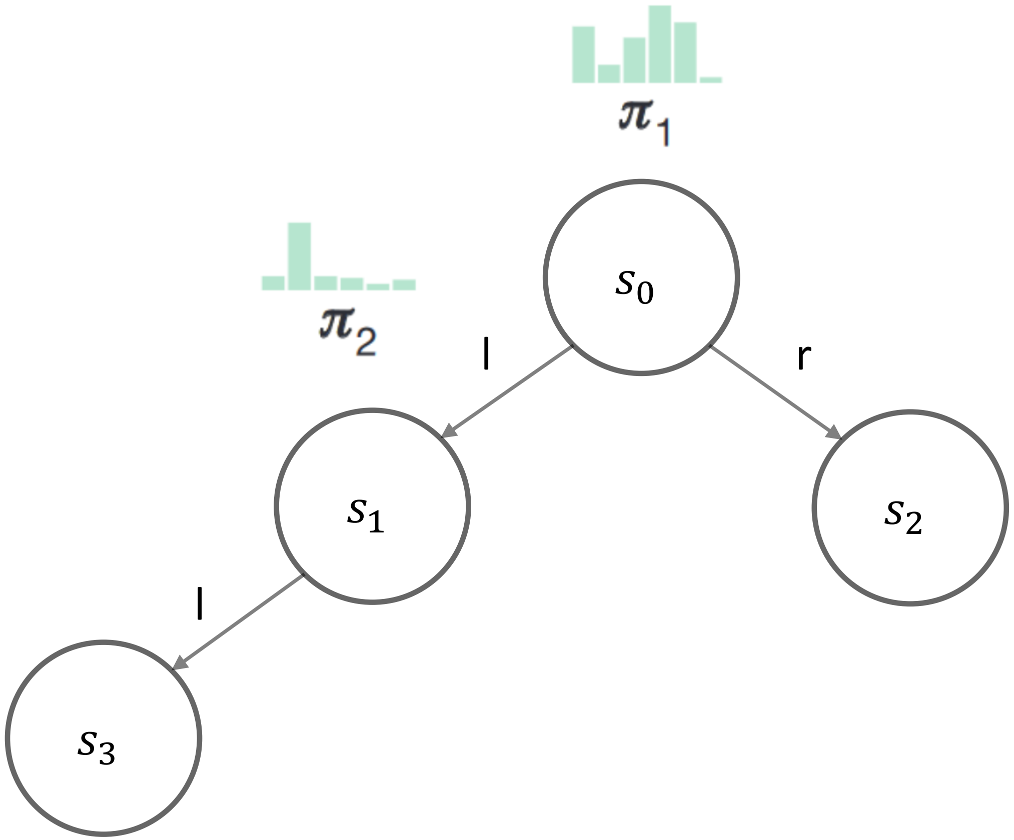


universität wien

ALPHAZERO: SELECTION

- At each node, select actions according to

$$UCT(s,a) = Q(s,a) + c\mathbf{P(s,a)}\frac{\sqrt{\sum_b N(s,b)}}{1 + N(s,a)}$$



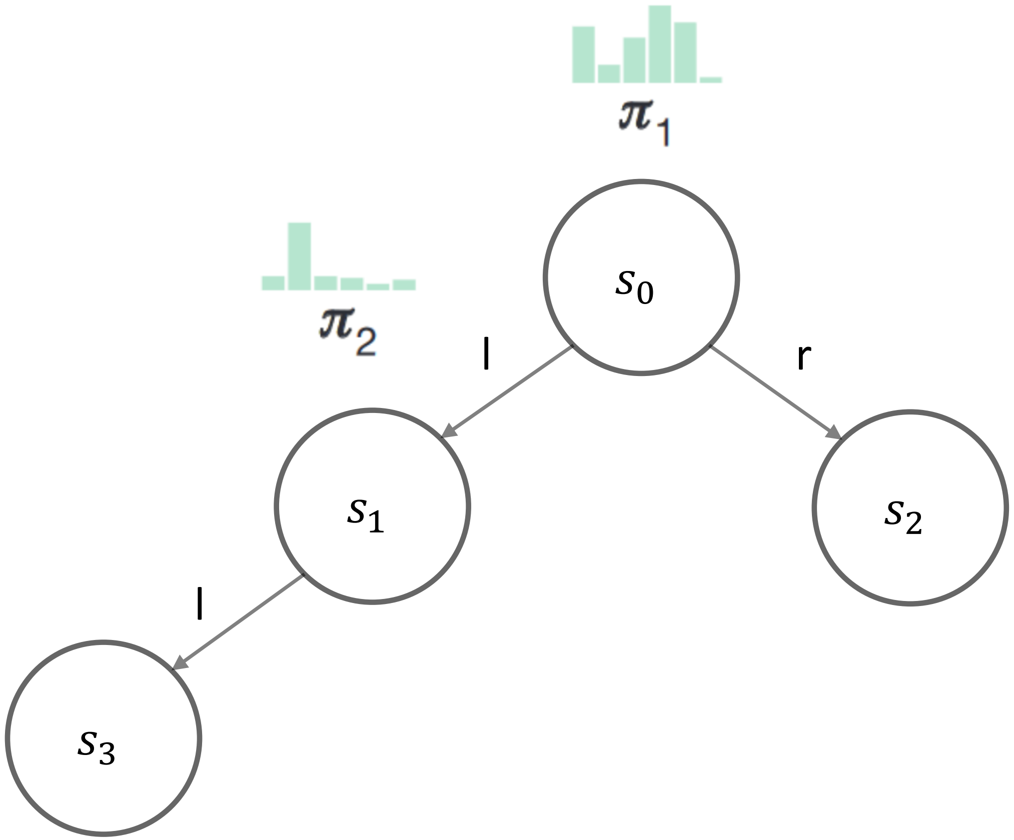
universität wien

ALPHAZERO: SELECTION

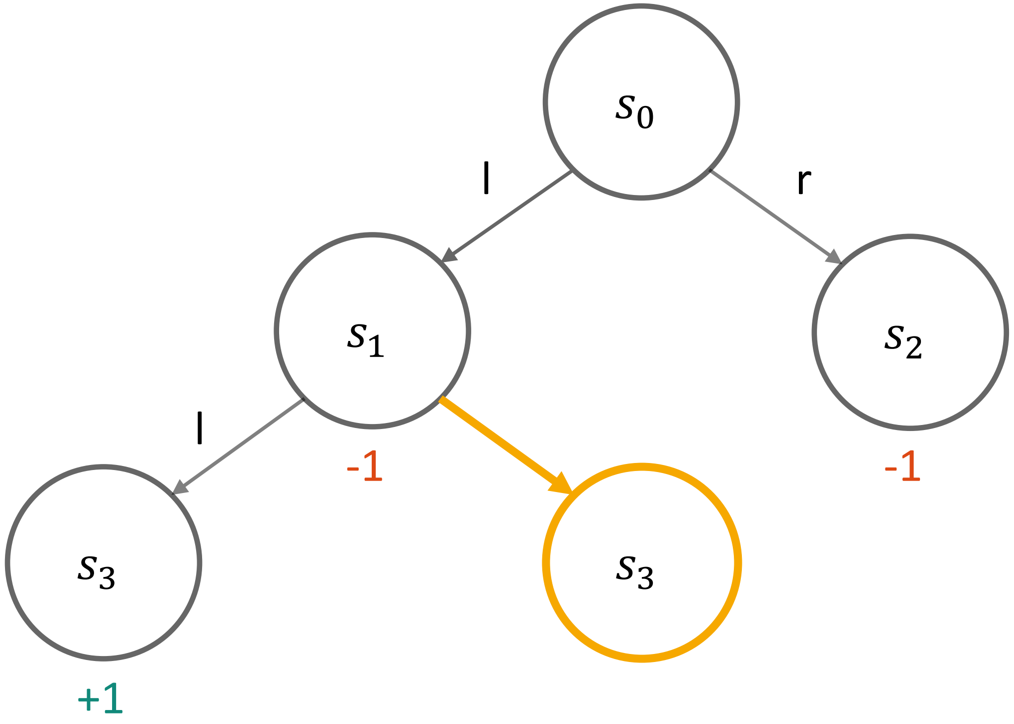
- At each node, select actions according to

$$UCT(s,a) = Q(s,a) + c\boldsymbol{P(s,a)}\frac{\sqrt{\sum_b N(s,b)}}{1+N(s,a)}$$

Move probability from the network's policy head for edge (s, a)



AlphaZero: Evaluation/Simulation

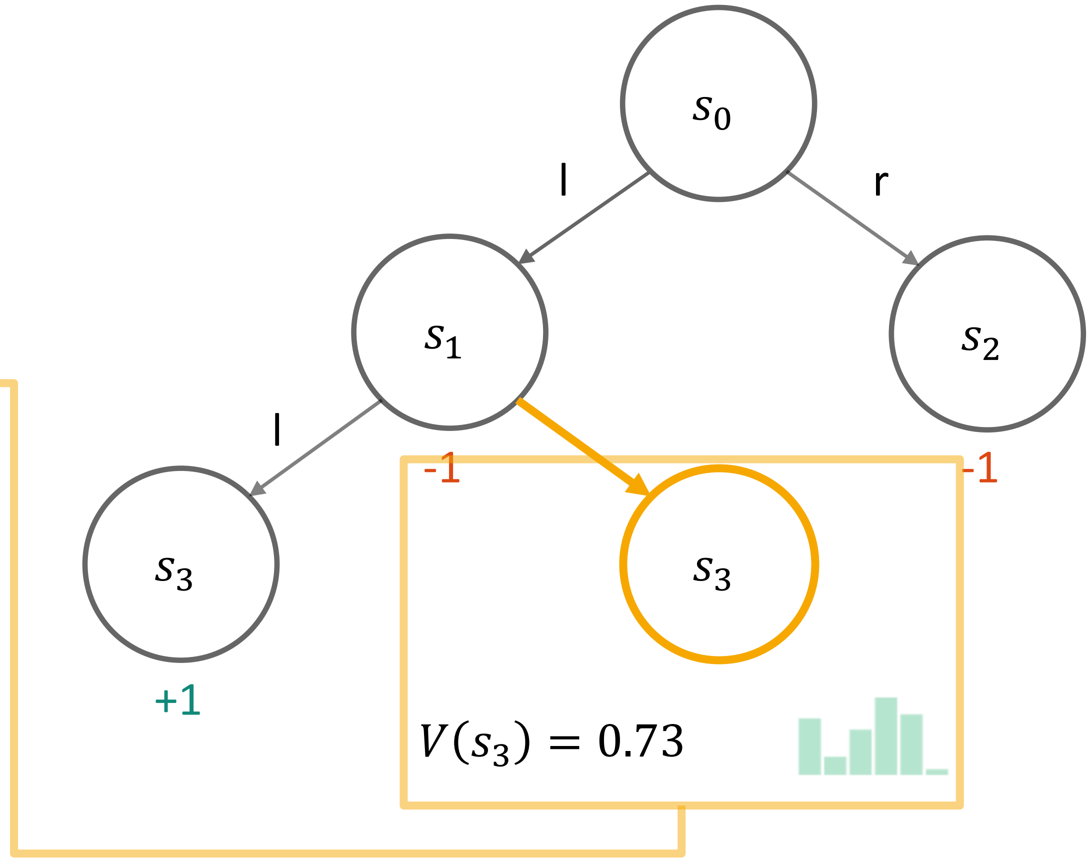


AlphaZero: Evaluation/Simulation

Neural network adds

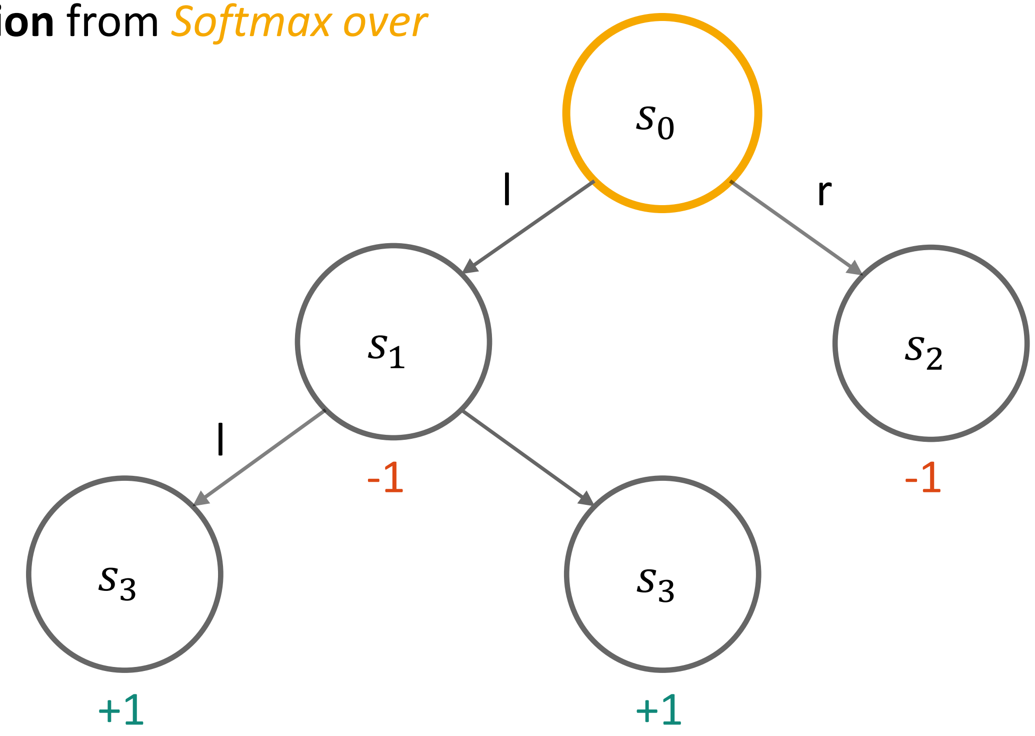
- Probability distribution over moves
- Win probability (= Value estimate)

▶ **No simulation is run!**



AlphaZero: Played Action (Data Generation)

Sample actual **game action** from *Softmax over root s_0 visitation counts*



AlphaZero Training

- **Network outputs**

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

$\boldsymbol{p}$: Move distribution
v: Win probability

AlphaZero Training

- **Network outputs**

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

$\boldsymbol{p}$: Move probabilities
v: Win probability

- **Training data**

$$(s_t, \boldsymbol{\pi}_t, z_t)$$

s_t: Actual game state
$\boldsymbol{\pi}_t$: MCTS selection probabilities
z_t: Game outcome from view of current player

AlphaZero Training

- Loss function

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

- **Network outputs**

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

$\boldsymbol{p}$: Move probabilities
v: Win probability

- **Training data**

$$(s_t, \boldsymbol{\pi}_t, z_t)$$

s_t: Game state
$\boldsymbol{\pi}_t$: MCTS selection probabilities
z_t: Game outcome from view of current player

AlphaZero Training

- Loss function

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

- MSE between value prediction and winner
- Cross-Entropy between policy and MCTS output
- Weight decay

- **Network outputs**

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

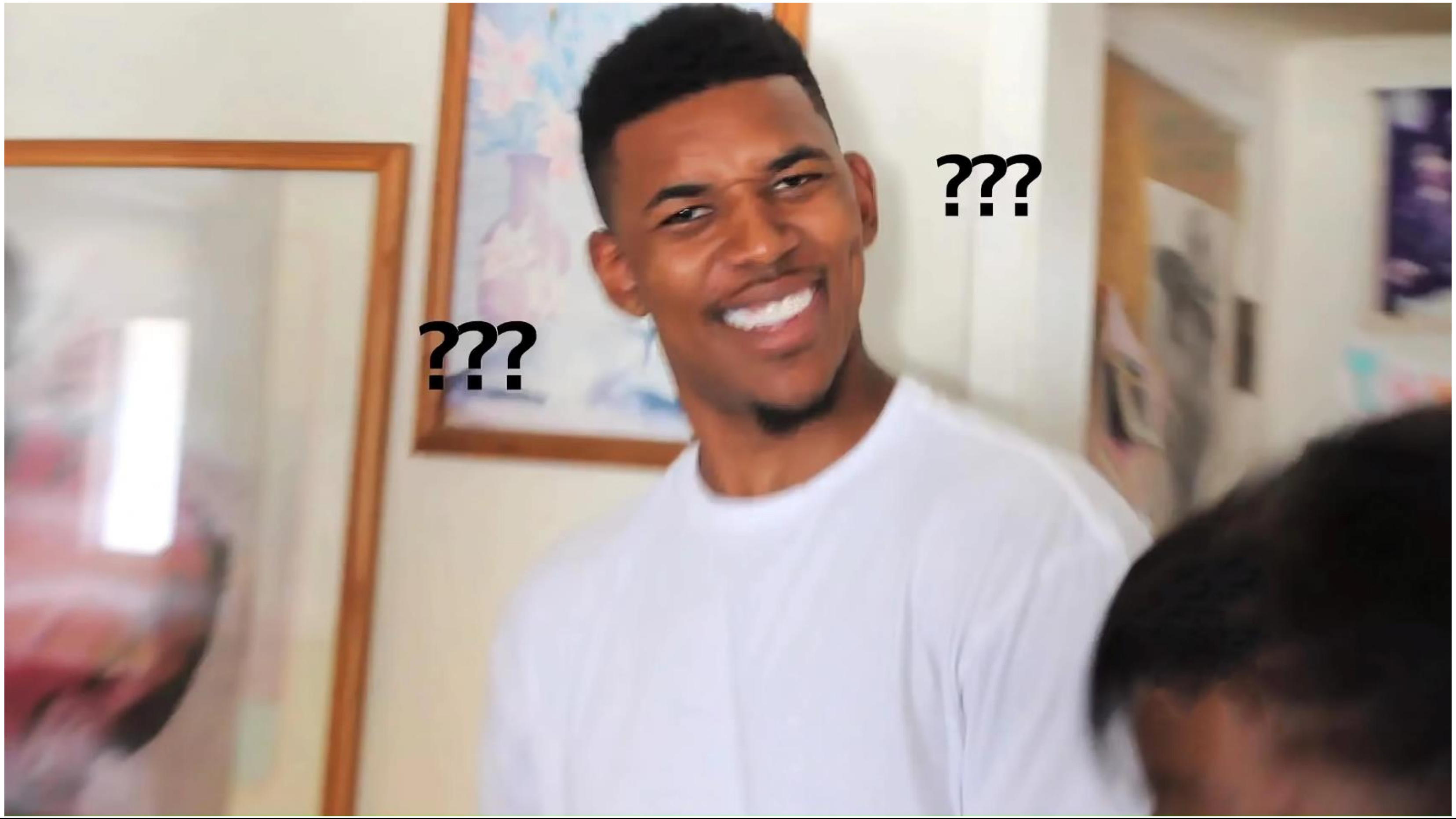
$\boldsymbol{p}$: Move probabilities
v: Win probability

- **Training data**

$$(s_t, \boldsymbol{\pi}_t, z_t)$$

s_t: Game state
$\boldsymbol{\pi}_t$: MCTS selection probabilities
z_t: Game outcome from view of current player

Why does this work?



Why does this work?

- At the start, policy outputs $\boldsymbol{p}_t$ will be complete garbage
- BUT: MCTS outputs $\boldsymbol{\pi}_t$ will be a *little better*
- Starts a **Self-Improving Loop**

Why does this work?

- At the start, policy outputs $\boldsymbol{p_t}$ will be complete garbage
- BUT: MCTS outputs $\boldsymbol{\pi}_t$ will be a *little better*
- Starts a **Self-Improving Loop**

1. Policy generates outputs
2. These are improved by MCTS
3. Policy is trained to match improved action probabilities
4. Repeat until superhuman

Engineering & Tricks

- Asynchronous data collection and training
- 5000 TPUs for data collection
- 4 days of training (for Go)
- Distributed network training
- MCTS parallelization with Virtual Loss
- Additional action noise at match start to avoid degeneration
- …

Limitations of AlphaZero

- Requires a *given* model of the environment
- Works only for discrete action spaces
- Environment must be fully observable
- Chess, Go etc. are deterministic environments
- Self-play works for zero-sum games only
- Crazy compute requirements for training

► Addressed by follow-up work

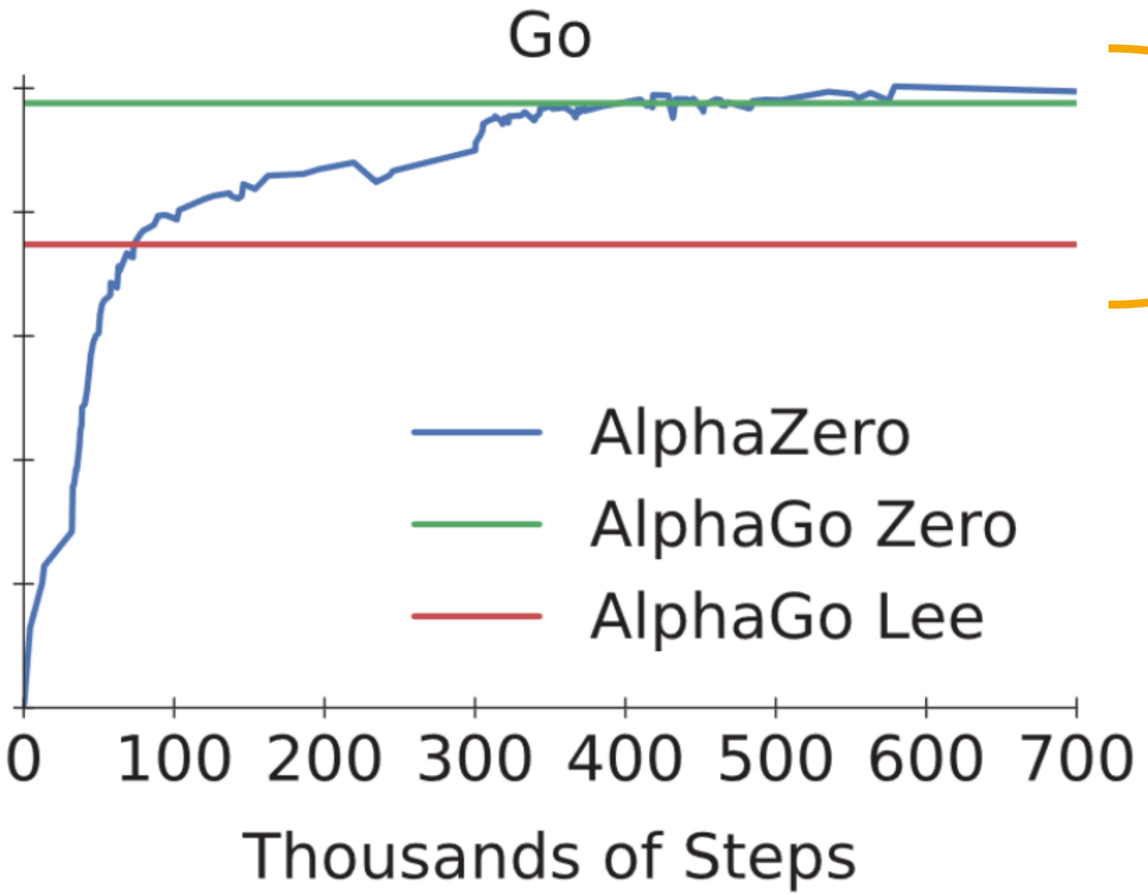
AlphaZero vs AlphaGo

- AlphaGo uses separate policy and value networks
- AlphaGo pre-trains policy and value nets on human data
- AlphaGo still uses simulations with a simulation network
- AlphaGo uses many Go-specific heuristics

AlphaZero vs AlphaGo Zero

- AlphaGo Zero uses Go-specific data augmentations
- AlphaGo Zero uses more rollouts (1600 vs 800)
- AlphaGo Zero uses tournament selection to select network

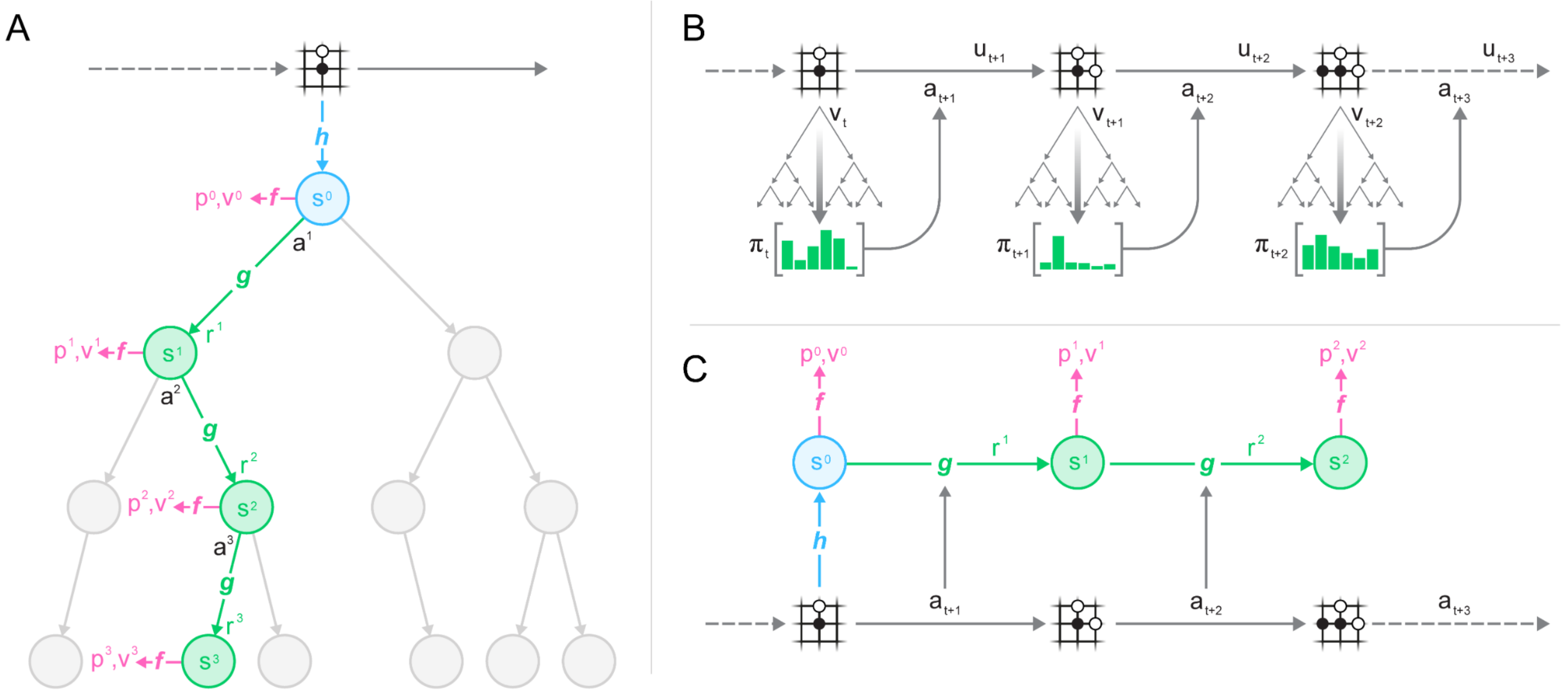
ALPHAGO VS ALPHAGO ZERO VS ALPHAZERO



Less human “expert ”knowledge
►Better algorithm ☺

AlphaZero vs MuZero

- h: Encoder
- g: Dynamics function
- f: Prediction function
- **Training**
  Functions are jointly trained to predict K steps from real trajectory



SAINT

- Social Artificial Intelligence Night
- 24.03.2023 | 16:30 | FH St. Pölten

SAINT

- 24.03.2023 | 16:30 | FH St. Pölten

References

- Browne, C. B., Powley, E., Whitehouse, D., Lucas, S. M., Cowling, P. I., Rohlfshagen, P., ... & Colton, S. (2012). A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games*, *4*(1), 1-43.
- Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., ... & Hassabis, D. (2016). Mastering the game of Go with deep neural networks and tree search. *nature*, *529*(7587), 484-489.
- Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., ... & Hassabis, D. (2017). Mastering the game of go without human knowledge. *nature*, *550*(7676), 354-359.
- Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., ... & Hassabis, D. (2018). A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science*, *362*(6419), 1140-1144.
- Schrittwieser, J., Antonoglou, I., Hubert, T., Simonyan, K., Sifre, L., Schmitt, S., ... & Silver, D. (2020). Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, *588*(7839), 604-609.

HOEFFDING'S INEQUALITY

Theorem (Hoeffding's Inequality)

Let $X_1, \ldots, X_t$ *be i.i.d. random variables in [0,1], and let* $\overline{X}_t = \frac{1}{\tau}\sum_{\tau=1}^{t} X_\tau$ *be the sample mean. Then*

$$\mathbb{P}\left[\mathbb{E}\left[X\right] > \overline{X}_t + u\right] \leq e^{-2tu^2}$$

universität
wien

ALPHAZERO CHESS EXAMPLE

- Shows 10 most visited states
- Estimated value from white's perspective, scaled by factor of 100
- Thickness of node border represents visit counts