Adversarial Search and Game Playing

Where choosing actions means respecting your opponent

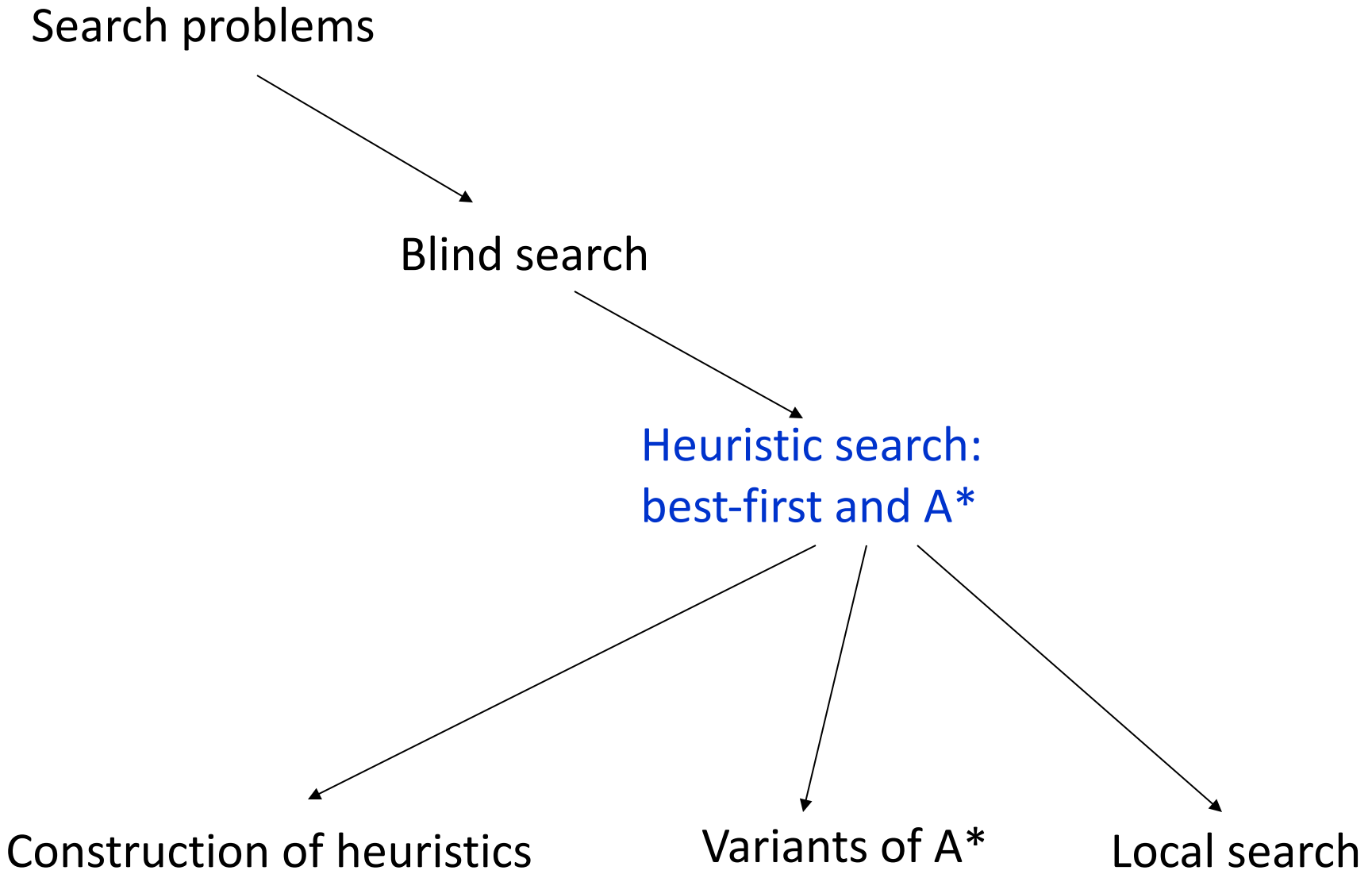
R&N: Chap. 6

With some slides from Dan Klein and Stuart Russell

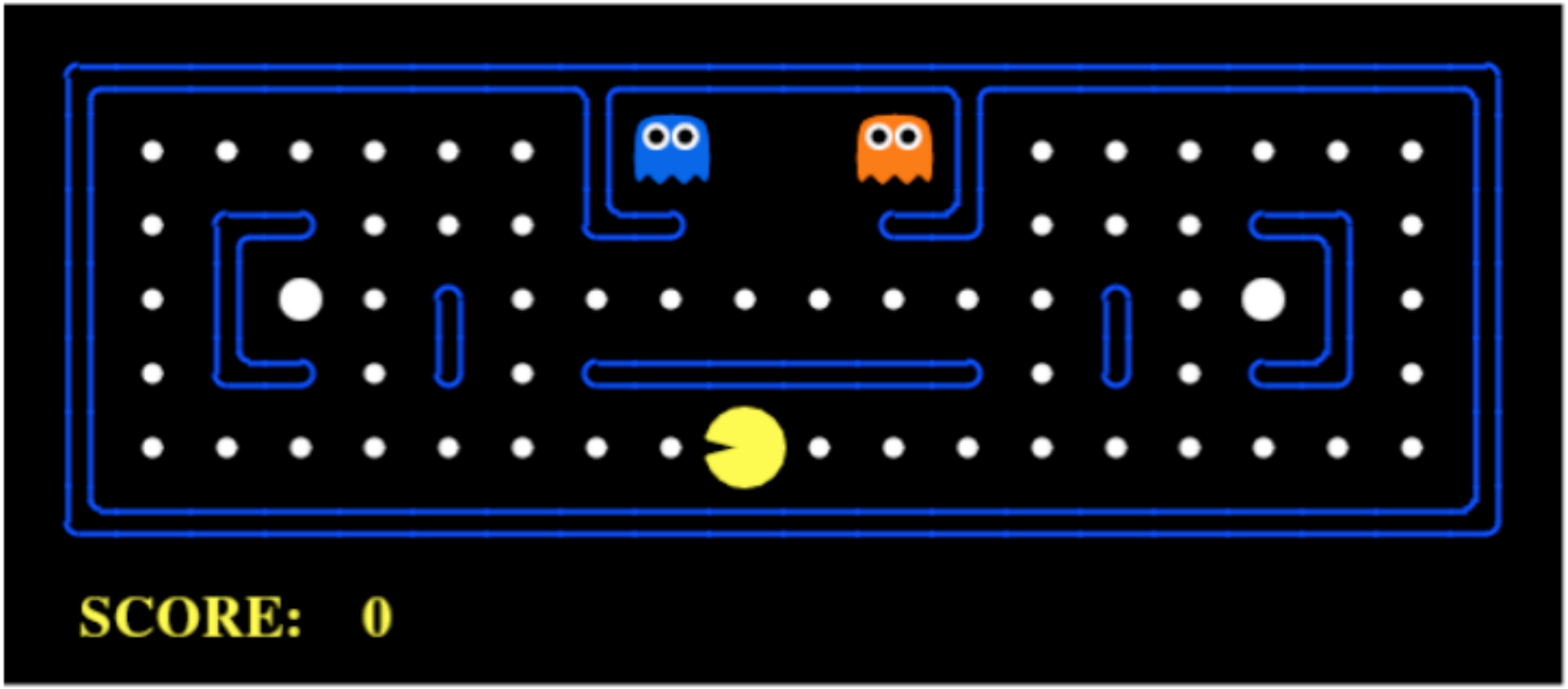
Project 1 Postmortem

- ☹ Almost 20% did not submit project 1 even 3 days after deadline (#FAIL)
- What were the major problems?
- What can we improve for Project 2 (assigned Thu?)

- Instructor solution overview
- Extra-credit approaches/solutions?


Search problems
Blind search
Heuristic search:
best-first and A*
Construction of heuristics
Variants of A*
Local search

Adversarial Search



Video of Demo Mystery Pacman

Why Study Games?

- Games mimic the natural world
- For centuries humans have used them for fun, education, and measuring their intelligence
- Making rules (and winning) explicit makes real-world problems possible to analyze

General Game Playing

IJCAI 2013 WORKSHOPS
August 3–5, 2013, Beijing, China

3rd International General Game Playing Workshop

General Intelligence in Game-Playing Agents (GIGA'13)

(http://giga13.ru.is)

General Information

Artificial Intelligence (AI) researchers have for decades worked on building game-playing agents capable of matching wits with the strongest humans in the world, resulting in several success stories for games like chess and checkers. The success of such systems has been partly due to years of relentless knowledge-engineering effort on behalf of the program developers, manually adding application-dependent knowledge to their game-playing agents. The various algorithmic enhancements used are often highly tailored towards the game at hand.

Research into general game playing (GGP) aims at taking this approach to the next level: to build intelligent software agents that can, given the rules of any game, automatically learn a strategy for playing that game at an expert level without any human intervention. In contrast to software systems designed to play one specific game, systems capable of playing arbitrary unseen games cannot be provided with game-specific domain knowledge a priori. Instead, they must be endowed with high-level abilities to learn strategies and perform abstract reasoning. Successful realization of such programs poses many interesting research challenges for a wide variety of artificial-intelligence sub-areas including (but not limited to):

- knowledge representation and reasoning
- heuristic search and automated planning
- computational game theory
- multi-agent systems
- machine learning

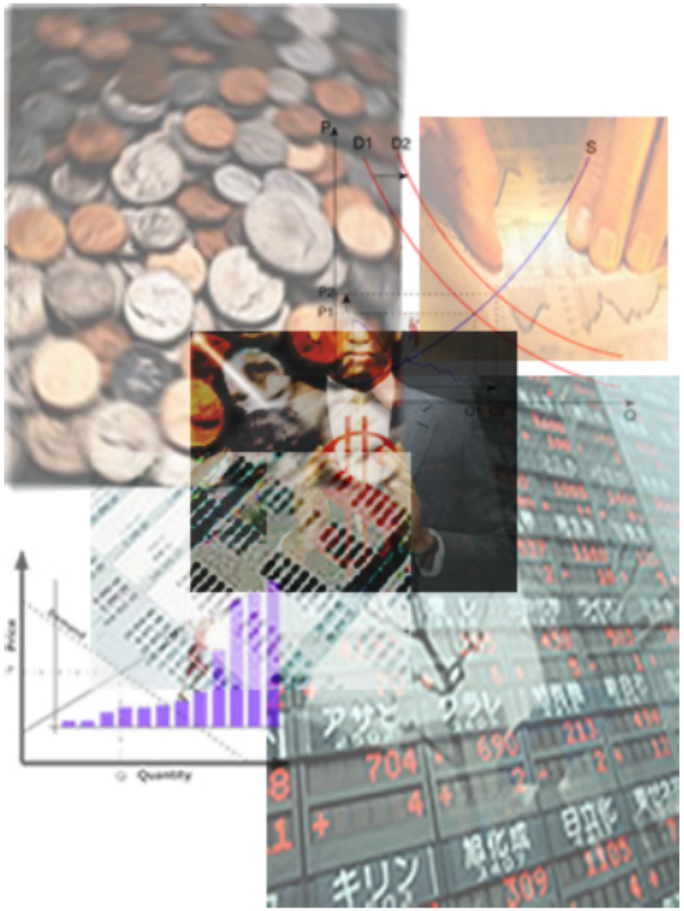
The aim of this workshop is to bring together researchers from the above sub-fields of AI to discuss how best to address the challenges of and further advance the state-of-the-art of general game-playing systems and generic artificial intelligence.

The workshop is one-day long and will be held onsite at IJCAI during the scheduled workshop period August 3rd-5th (exact day is to be announced later).

Real World: Serious Games

Some Fields where Game Theory is Used

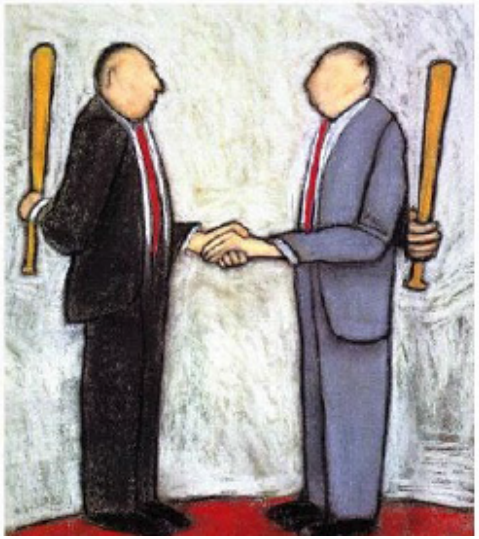
- Economics
  - Auctions
  - Markets
  - Bargaining
  - Fair division
  - Social networks
  - …

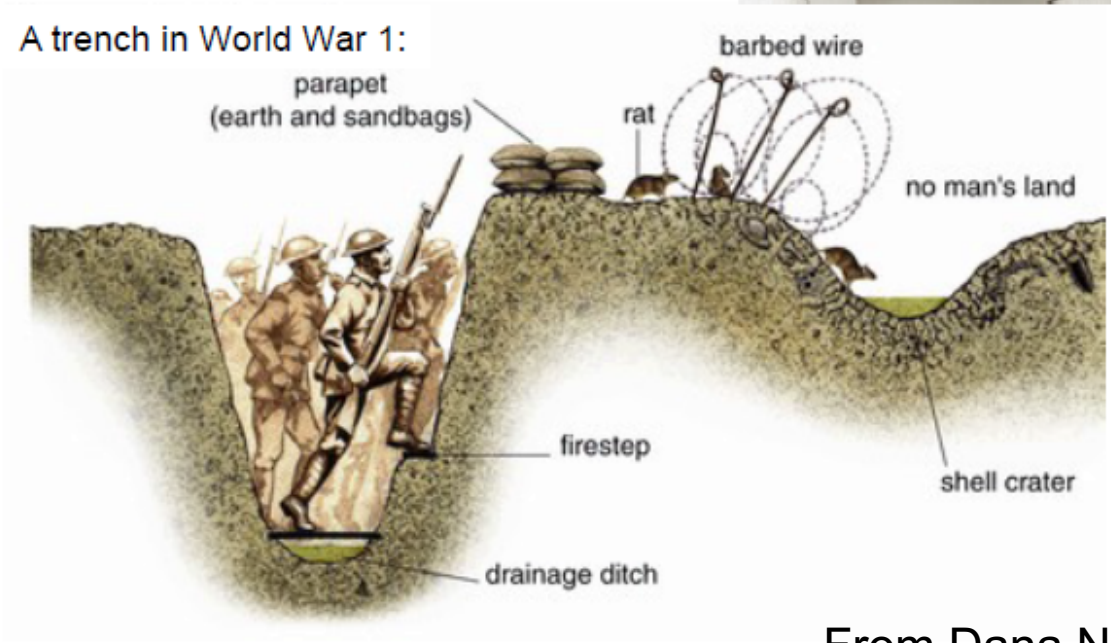


From Dana Nau

Some Fields where Game Theory is Used

- Government and Politics
  - Voting systems
  - Negotiations
  - International relations
  - War
  - Human rights



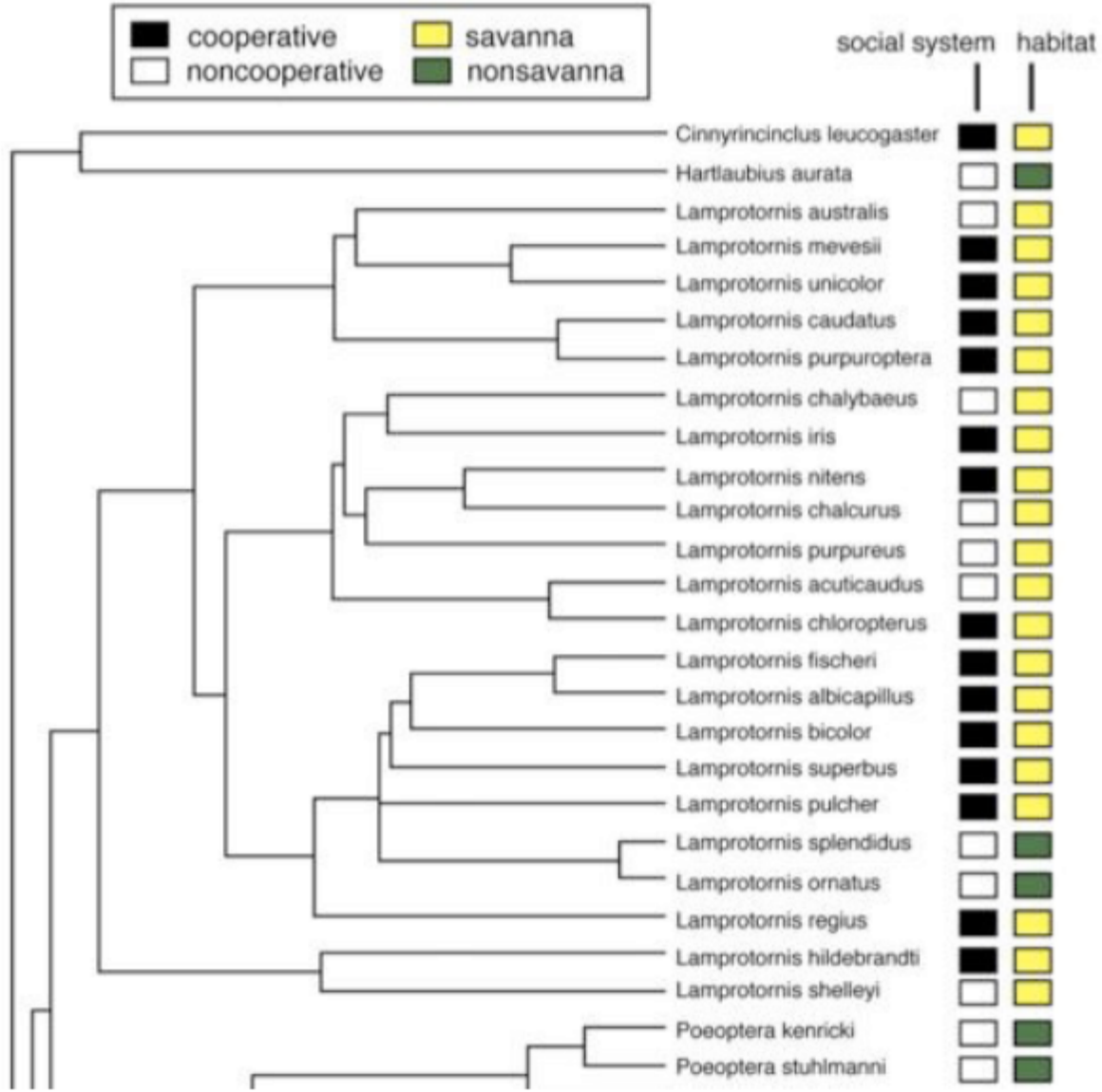


From Dana Nau

Some Fields where Game Theory is Used

- Evolutionary Biology
  - Communication
  - Population ratios
  - Territoriality
  - Altruism
  - Parasitism, symbiosis
  - Social behavior

From Dana Nau



TCP/IP Traffic

From Dana Nau, UMD

- Internet traffic is governed by the TCP protocol
- TCP's *backoff* mechanism
  - If the rate at which you're sending packets causes congestion, reduce the rate until congestion subsides
- Suppose that
  - You're trying to finish an important project
    - It's extremely important for you to have a fast connection
  - Only one other person is using the Internet
    - That person wants a fast connection just as much as you do
- You each have 2 possible actions:
  - C (use a *correct* implementation)
  - D (use a *defective* implementation that won't back off)

TCP/IP: Analysis

From Dana Nau, UMD

- An **action profile** is a choice of action for each agent
  - You both use C => average packet delay is 1 ms
  - You both use D => average delay is 3 ms (router overhead)
  - One of you uses D, the other uses C:
    - D user's delay is 0
    - C user's delay is 4 ms
- **Payoff matrix**:
  - Your options are the rows
  - The other agent's options are the columns
  - Each cell = an action profile
    - 1st number in the cell is your **payoff** or **utility** (I'll use those terms synonymously)
      - In this case, the negative of your delay
    - 2nd number in each cell is the other agent's payoff

How to analyze if payoffs are 2, 3, 4 moves away?

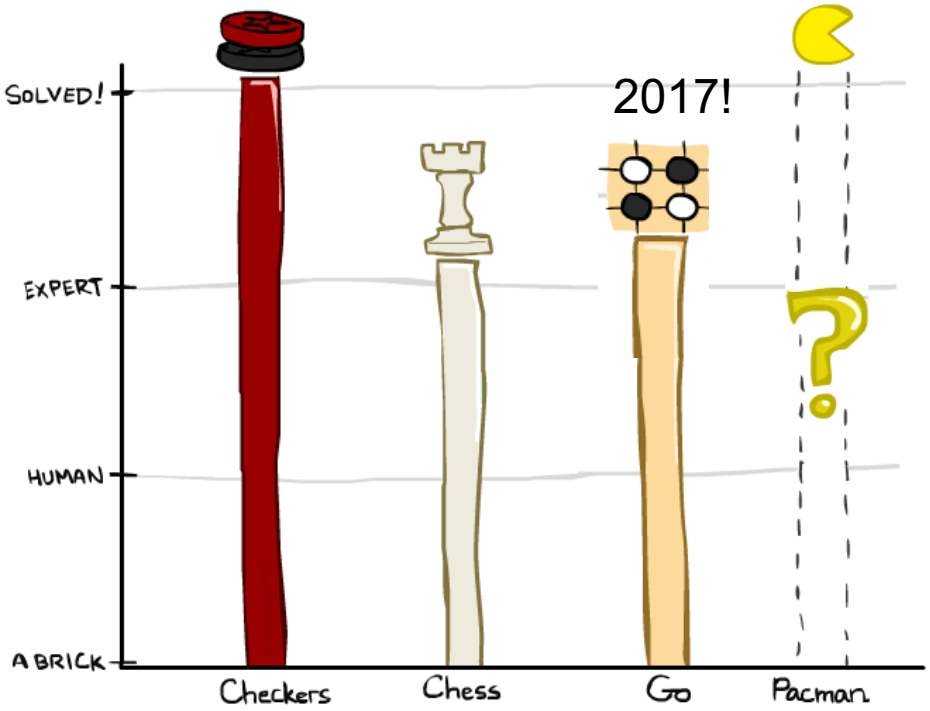
	C	*D*
C	–1, –1	–4, 0
D	0, –4	–3, –3

General Idea: Rational Agents

- To model games, assume agents are **rational**
- Rational = ?
- What is the rational thing to do in TCP/IP example?
- Rational agent will take a sequence of actions to maximize utility
  - $$
  - Winning a game
  - Minimize delay

Game Playing State-of-the-Art

- **Checkers:** 1950: First computer player. 1994: First computer champion: Chinook ended 40-year-reign of human champion Marion Tinsley using complete 8-piece endgame. 2007: Checkers solved!
- **Chess:** 1997: Deep Blue defeats human champion Gary Kasparov in a six-game match. Deep Blue examined 200M positions per second, used very sophisticated evaluation and undisclosed methods for extending some lines of search up to 40 ply. Current programs are even better, if less historic.
- **Go:** (updated for 2017): Google DeepMind beat human champion (but it was close). In Go, b > 300!. DeepMind uses Monte Carlo (randomized) expansion methods + deep learning for estimation + simulations for self-training.
- **Pacman**



Checkers: Tinsley vs. Chinook

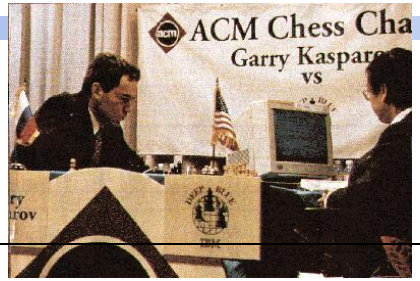
Name: Marion Tinsley
Profession: Teaches Math
Hobby: Checkers
Record: Over 42 years
lost only 3 games
of checkers
World champion for over 40 years

Mr. Tinsley suffered his 4th and 5th losses against Chinook

Chinook: World Checkers Champion

First computer to become official world champion of Checkers!

Chess: Kasparov vs. Deep Blue



Kasparov		Deep Blue
5’10”	**Height**	6’ 5”
176 lbs	**Weight**	2,400 lbs
34 years	**Age**	4 years
50 billion neurons	**Computers**	32 RISC processors
		+ 256 VLSI chess engines
2 pos/sec	**Speed**	200,000,000 pos/sec
Extensive	**Knowledge**	Primitive
Electrical/chemical	**Power Source**	Electrical
Enormous	**Ego**	None

1997: Deep Blue wins by 3 wins, 1 loss, and 2 draws

Chess: Kasparov vs. Deep Junior

Deep Junior

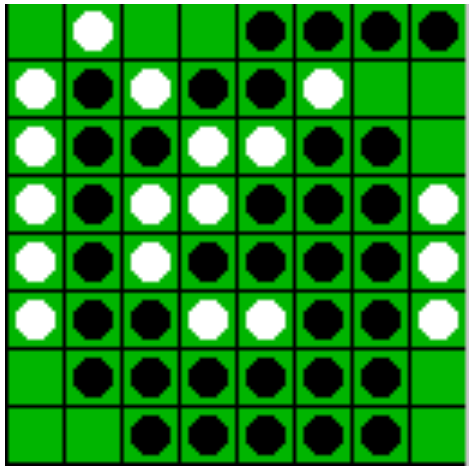
8 CPU, 8 GB RAM, Win 2000

2,000,000 pos/sec

Available for < $1000

August 2, 2003: Match ends in a 3/3 tie!

Othello: Murakami vs. Logistello



Takeshi Murakami
World Othello Champion

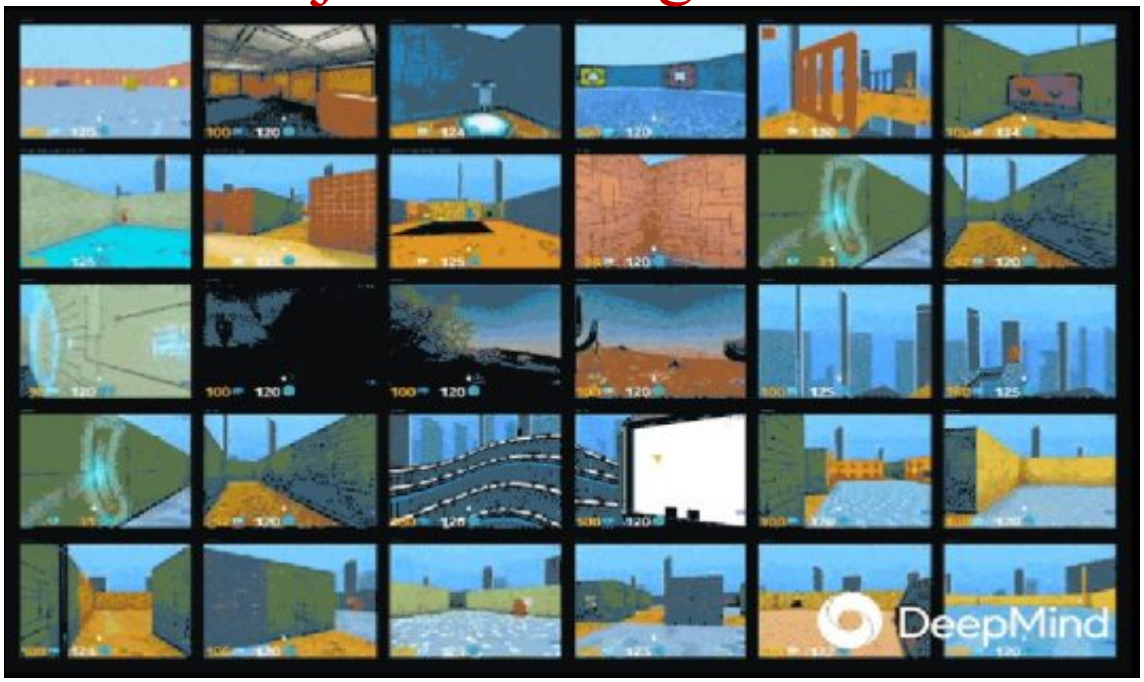
1997: The Logistello software crushed Murakami by 6 games to 0

Easy game? Try it: http://www.othelloonline.org/

Today in AI News

Google taught AI to multitasl

"We already knew DeepMind's AI was better than us at games, but now it's just showing off."

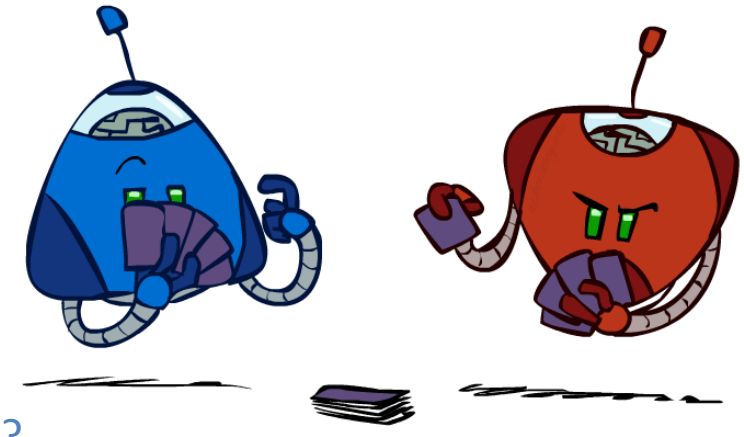


https://thenextweb.com/artificial-intelligence/2018/02/08/deepmind-taught-ai-how-to-multitask-using-video-games/

https://arxiv.org/pdf/1802.01561.pdf

Types of Games

- Many different kinds of games!
- Dimensions:
  - Deterministic or stochastic?
  - One, two, or more players?
  - Zero sum?
  - Perfect information (can you see the state)?

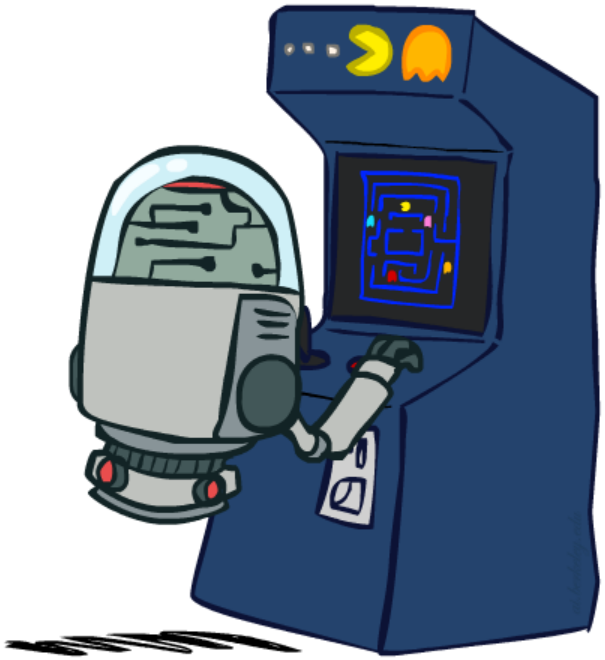


- Want algorithms for calculating a strategy (policy) which recommends a move from each state

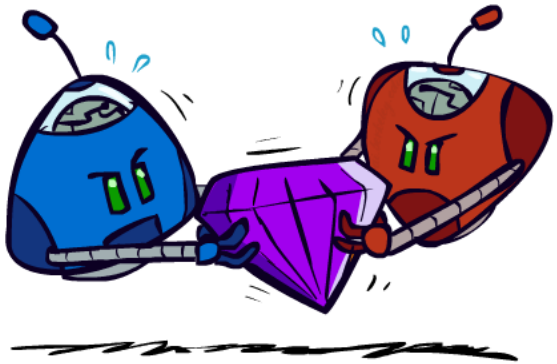
Deterministic Games

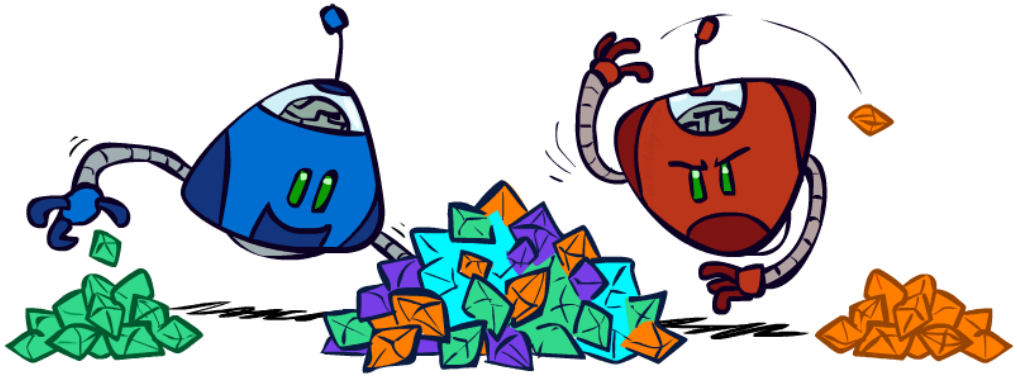
- Many possible formalizations, one is:
  - States: S (start at s_0)
  - Players: P={1...N} (usually take turns)
  - Actions: A (may depend on player / state)
  - Transition Function: SxA $\rightarrow$ S
  - Terminal Test: S $\rightarrow$ {t,f}
  - Terminal Utilities: SxP $\rightarrow$ R

- Solution for a player is a policy: S $\rightarrow$ A



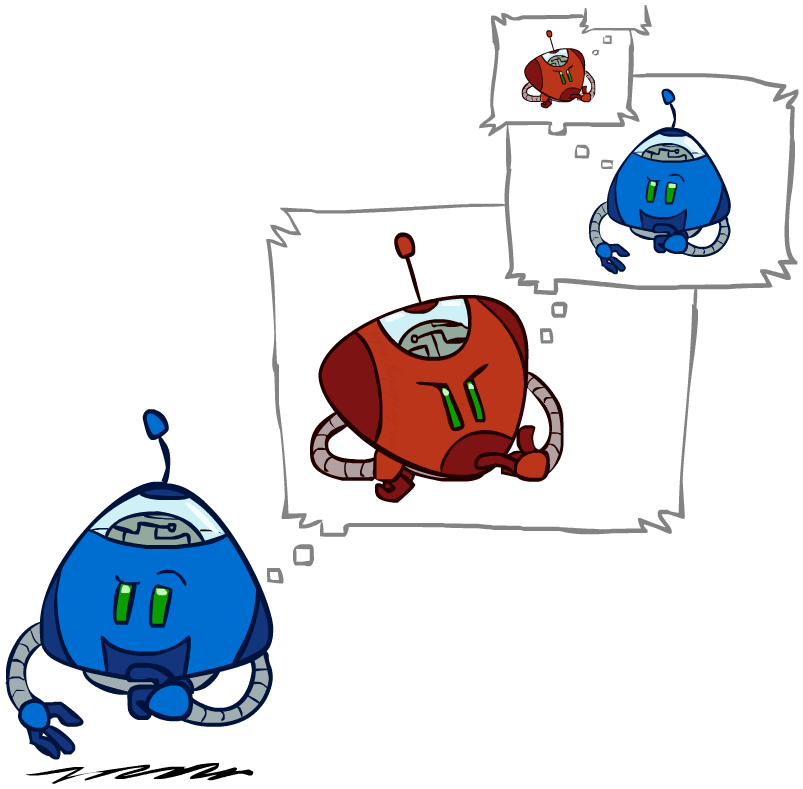
Zero-Sum Games



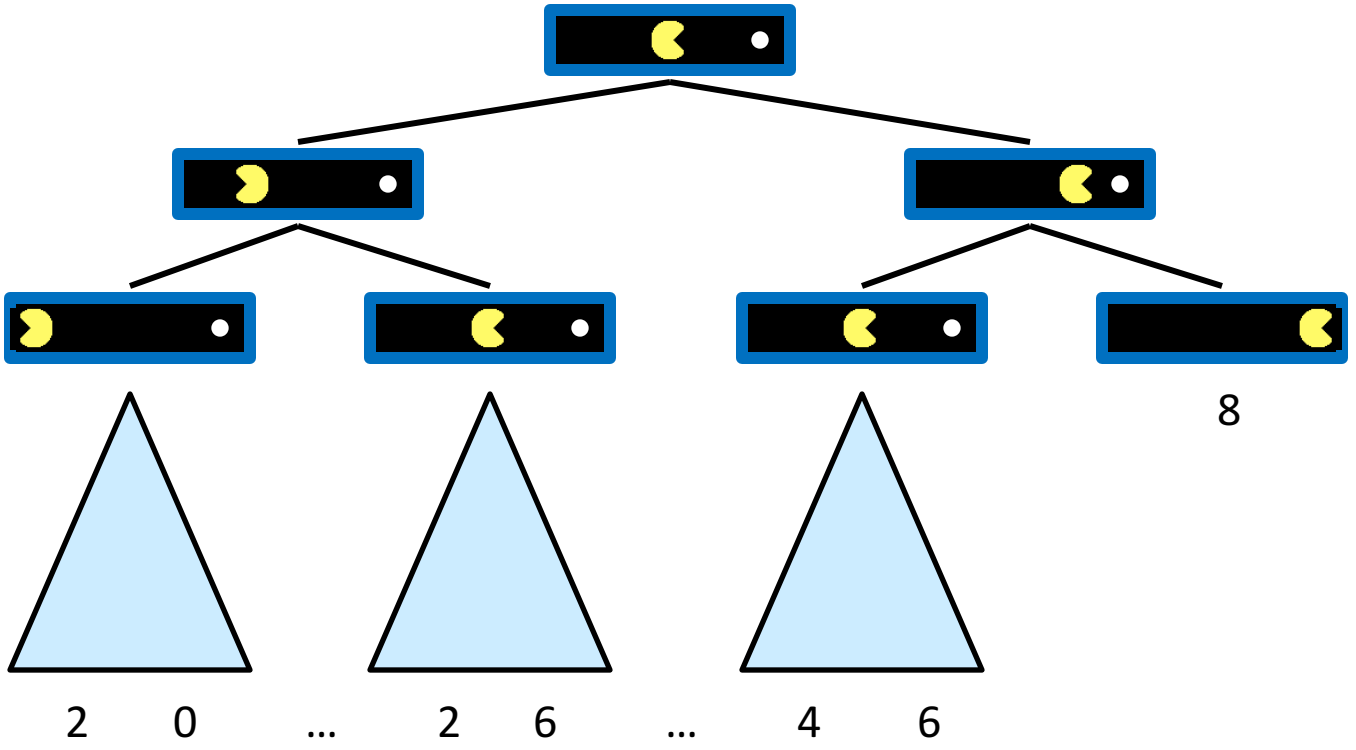


- **Zero-Sum Games**
  - Agents have opposite utilities (values on outcomes)
  - Lets us think of a single value that one maximizes and the other minimizes
  - Adversarial, pure competition
- General Games
  - Agents have independent utilities (values on outcomes)
  - Cooperation, indifference, competition, and more are all possible
  - More later on non-zero-sum games

Adversarial Search

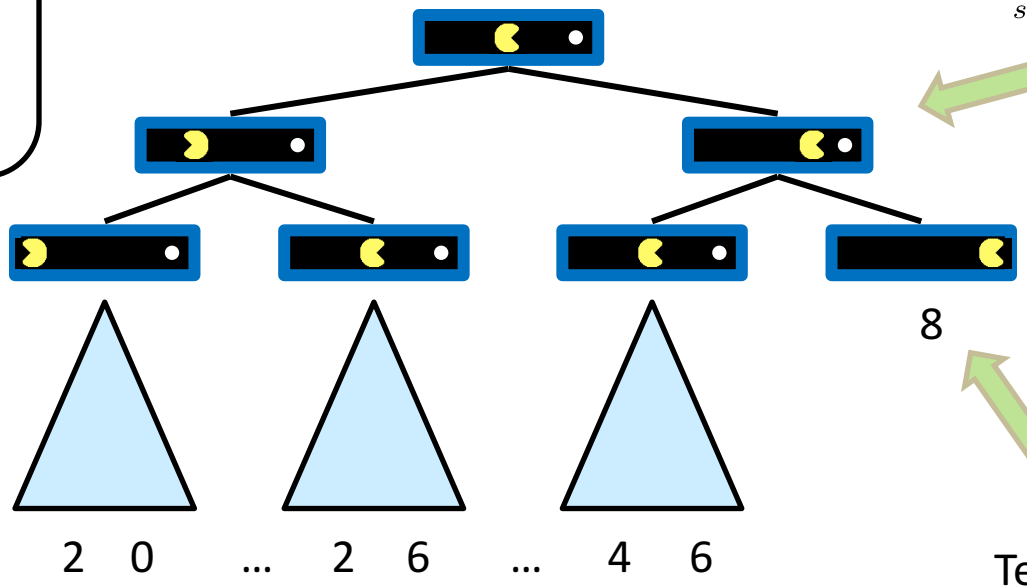


Single-Agent Trees

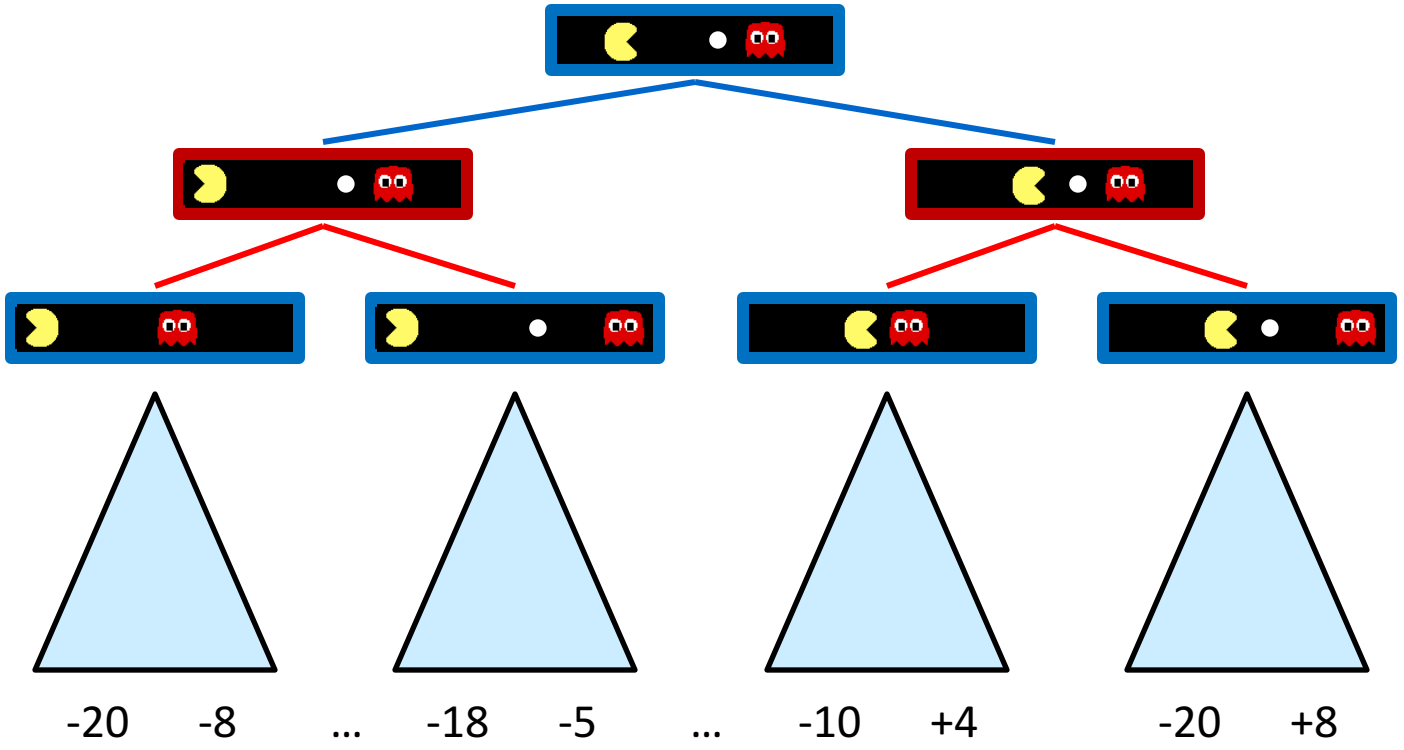


Value of a State

Value of a state:
The best achievable outcome (utility) from that state



Adversarial Game Trees

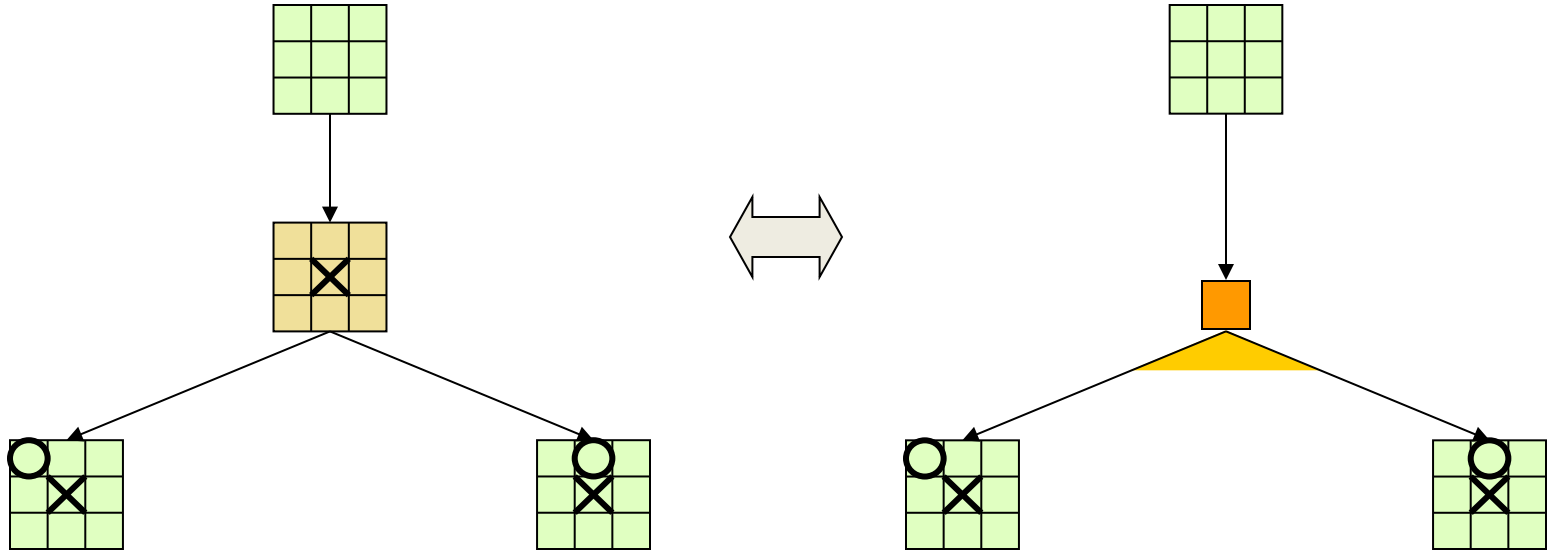


Setting: Two-player, turn-taking, deterministic, fully observable, zero-sum, time-constrained game

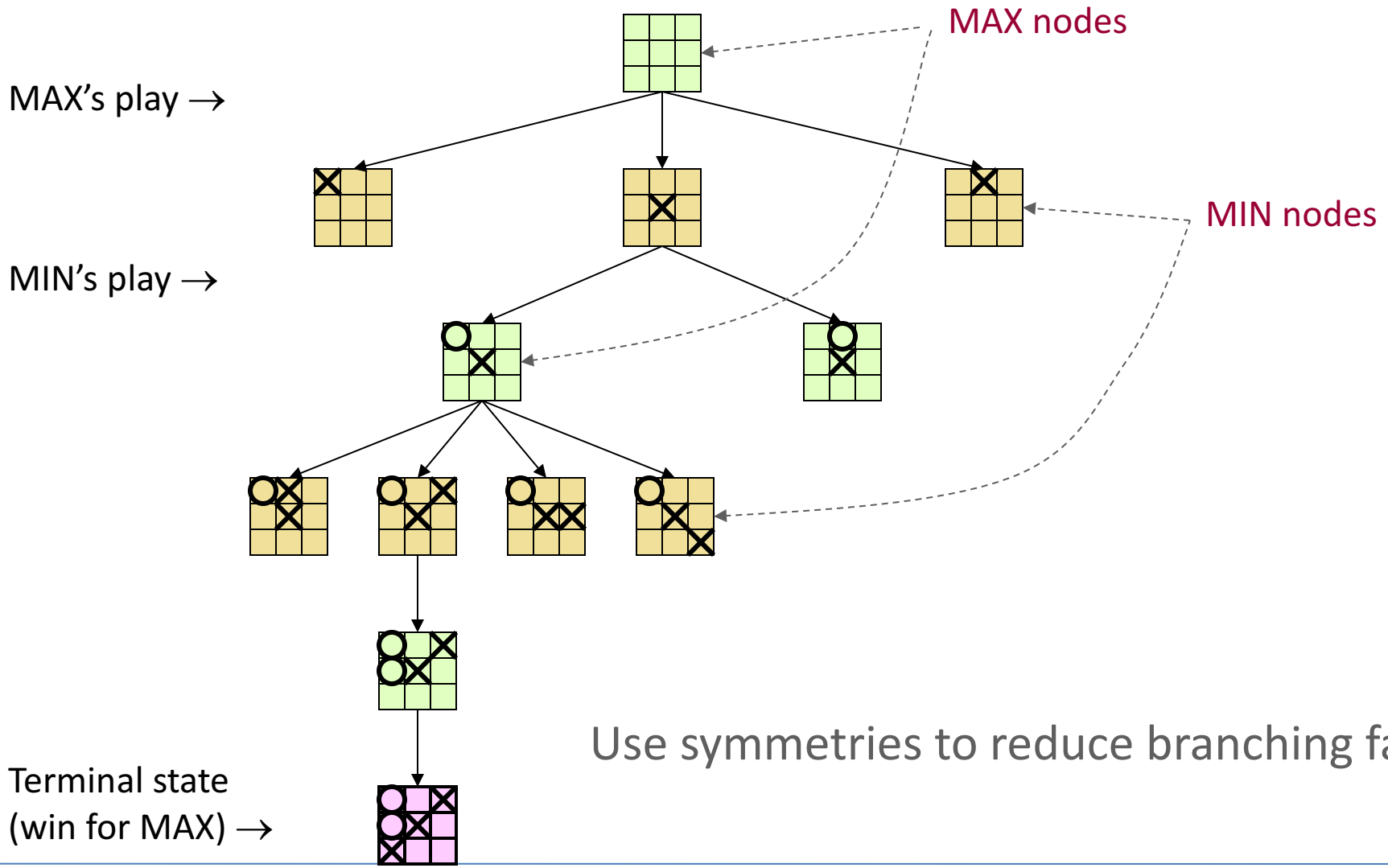
- State space
- Initial state
- Successor function: it tells which actions can be executed in each state and gives the successor state for each action
- MAX's and MIN's actions alternate, with MAX playing first in the initial state
- Terminal test: it tells if a state is terminal and, if yes, if it's a win or a loss for MAX, or a draw
- All states are fully observable

Why This Search is Harder?

- **Uncertainty** caused by the actions of another agent (MIN), who competes with our agent (MAX)

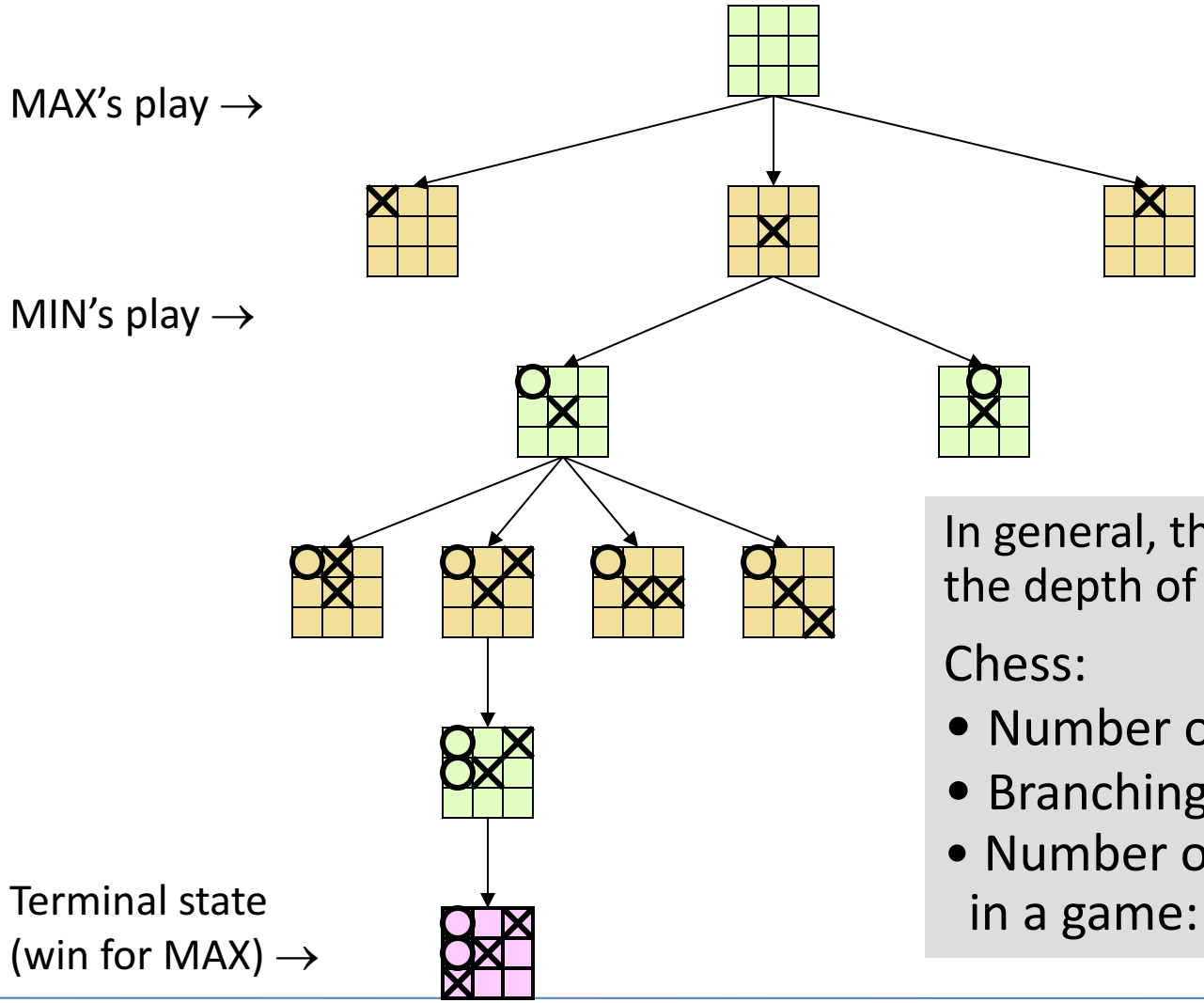


Game Tree



Use symmetries to reduce branching factor

Game Tree



In general, the branching factor and the depth of terminal states are large

Chess:

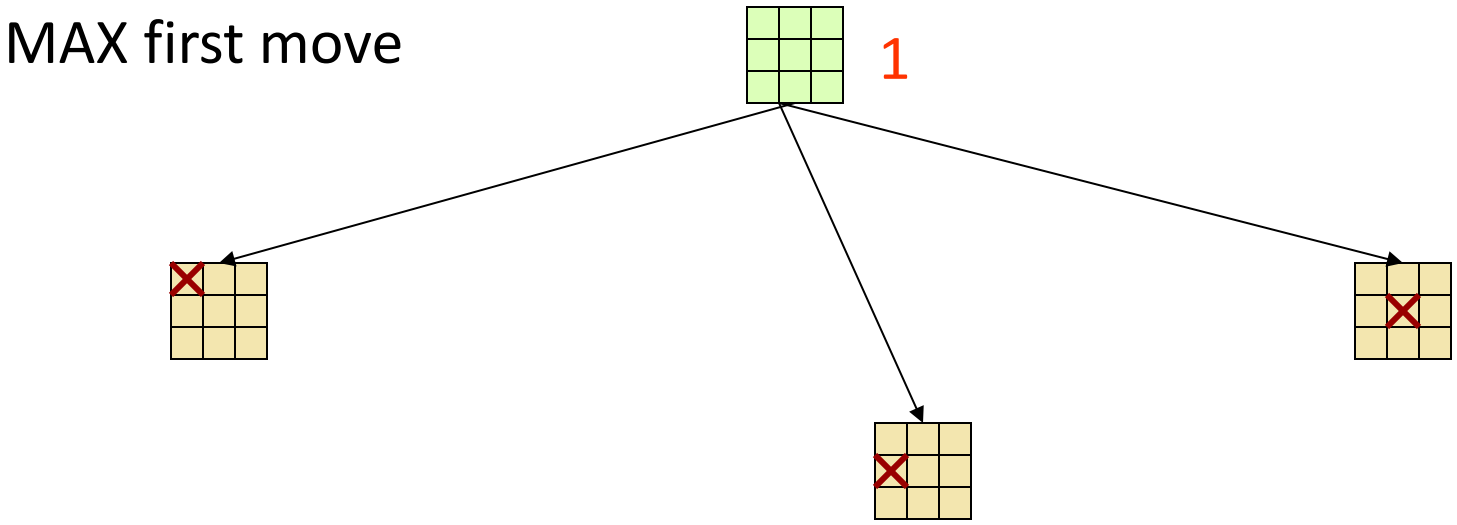
- Number of states: ~10^{40}
- Branching factor: ~35
- Number of total moves in a game: ~100

Choosing an Action: Basic Idea

1) Using the current state as the initial state, build the game tree uniformly to the maximal depth h (called horizon) feasible within the time limit
2) Evaluate the states of the leaf nodes
3) Back up the results from the leaves to the root and pick the best action assuming the worst (for us) from MIN

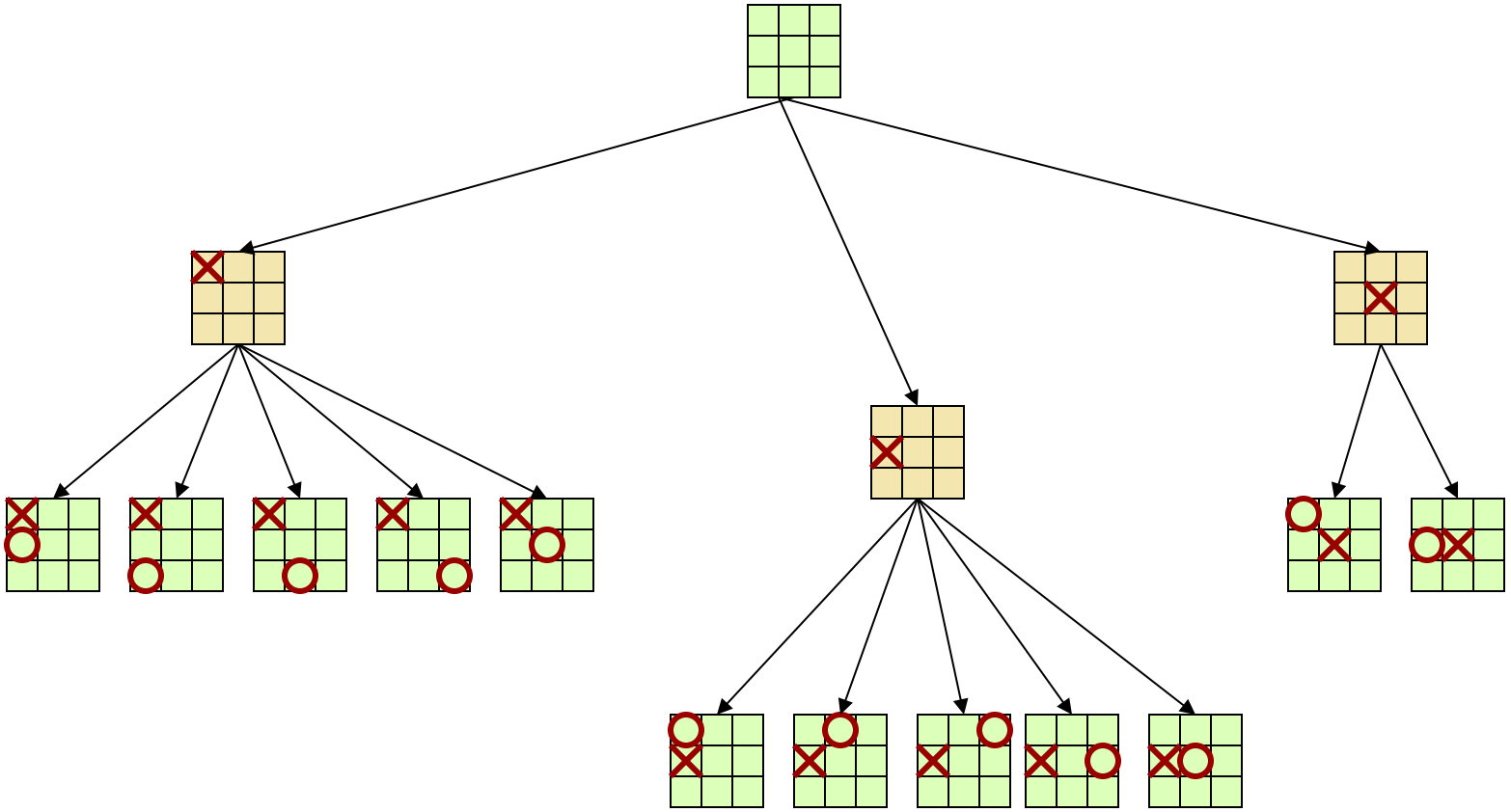
→ Minimax algorithm

Propagating e-values to root (first move)



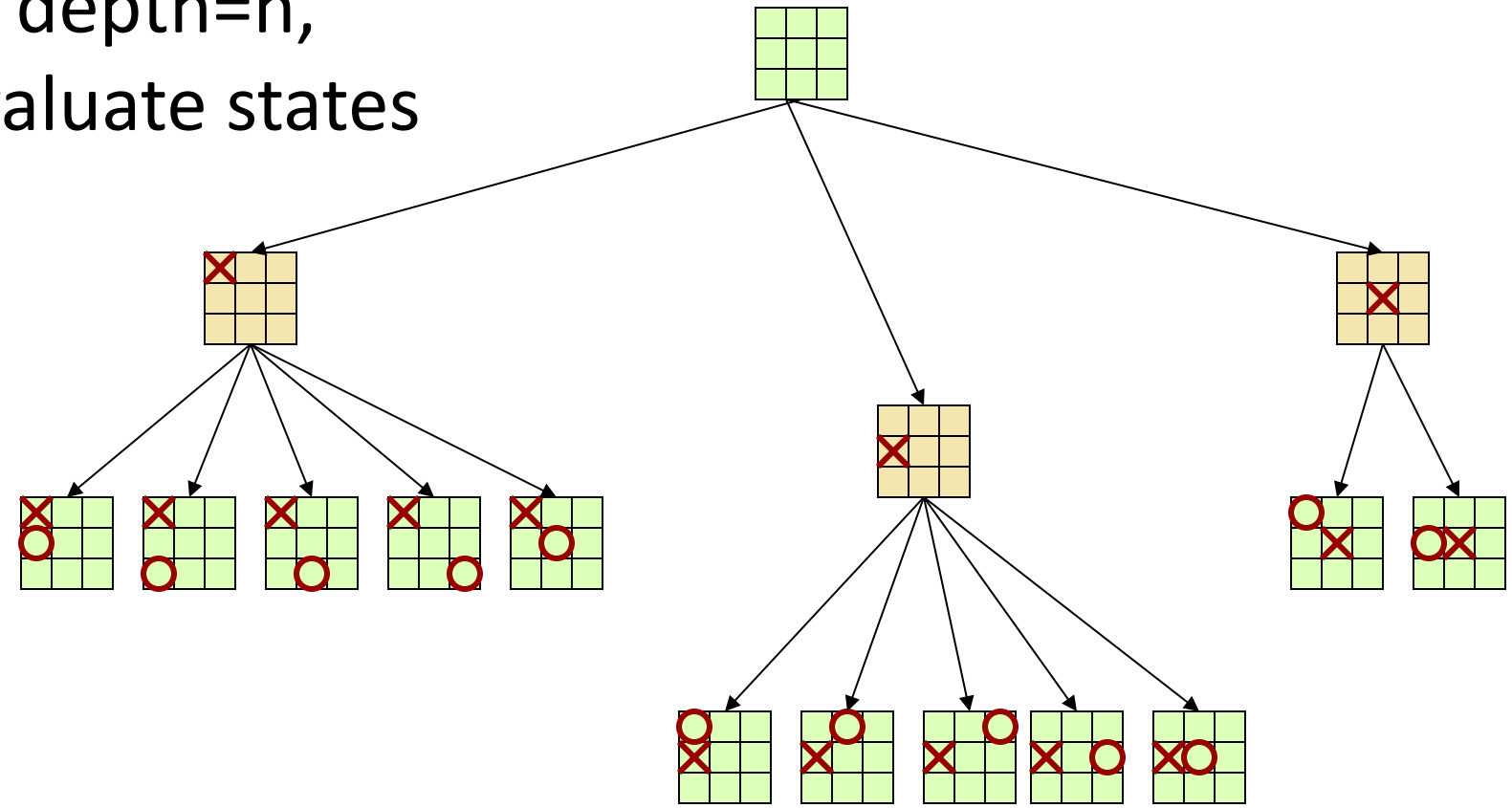
e(s) = (#rows + #columns + #diagonals for MAX)
- (#rows+#columns+#diagonals for MIN)

Propagating e-values to root (first move)



Propagating e-values to root (first move)

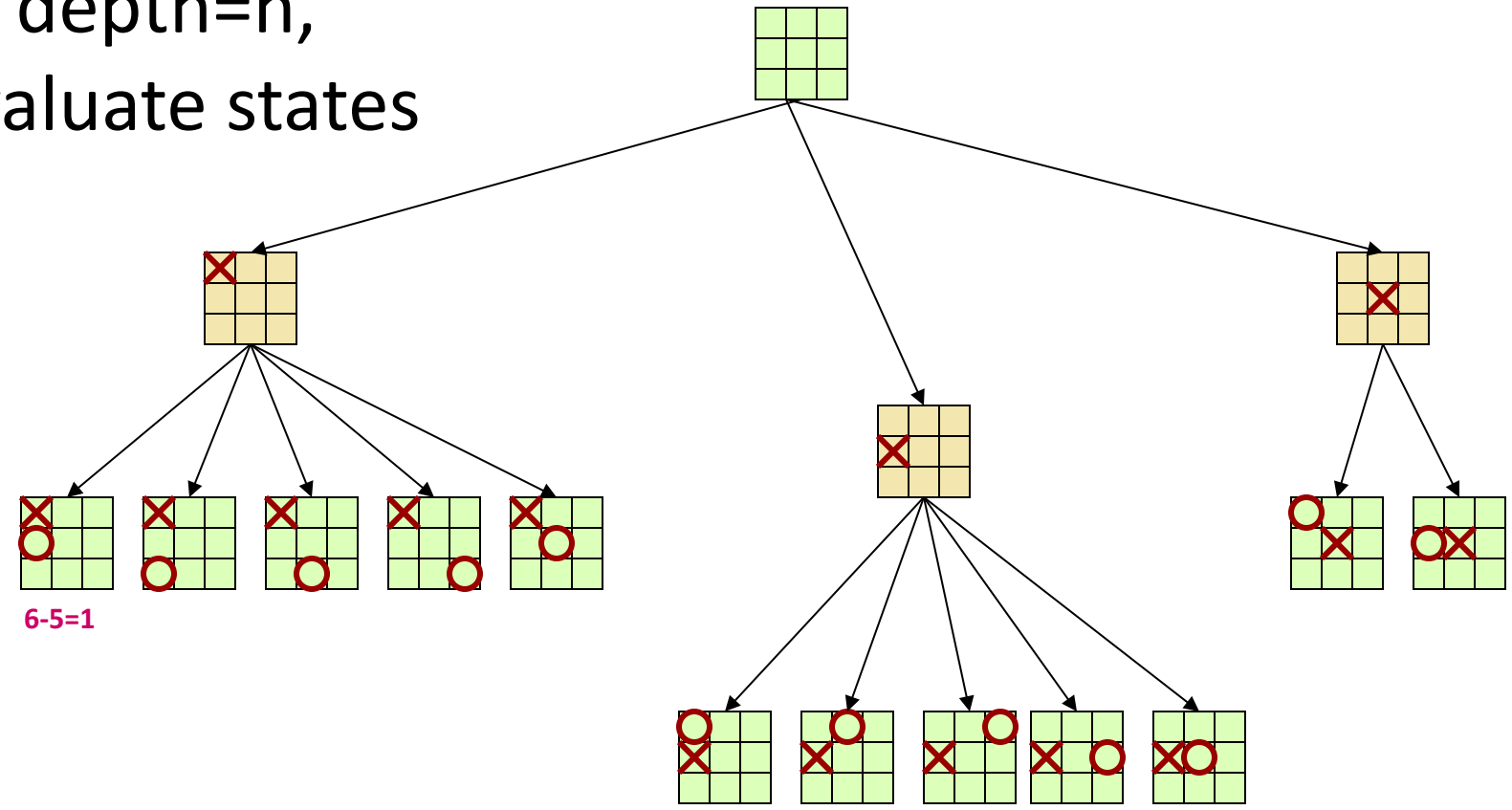
- At depth=h, evaluate states



e(s) = (#row + #col + #diag for MAX)- (#row+#col+#diag for MIN)

Propagating e-values to root (first move)

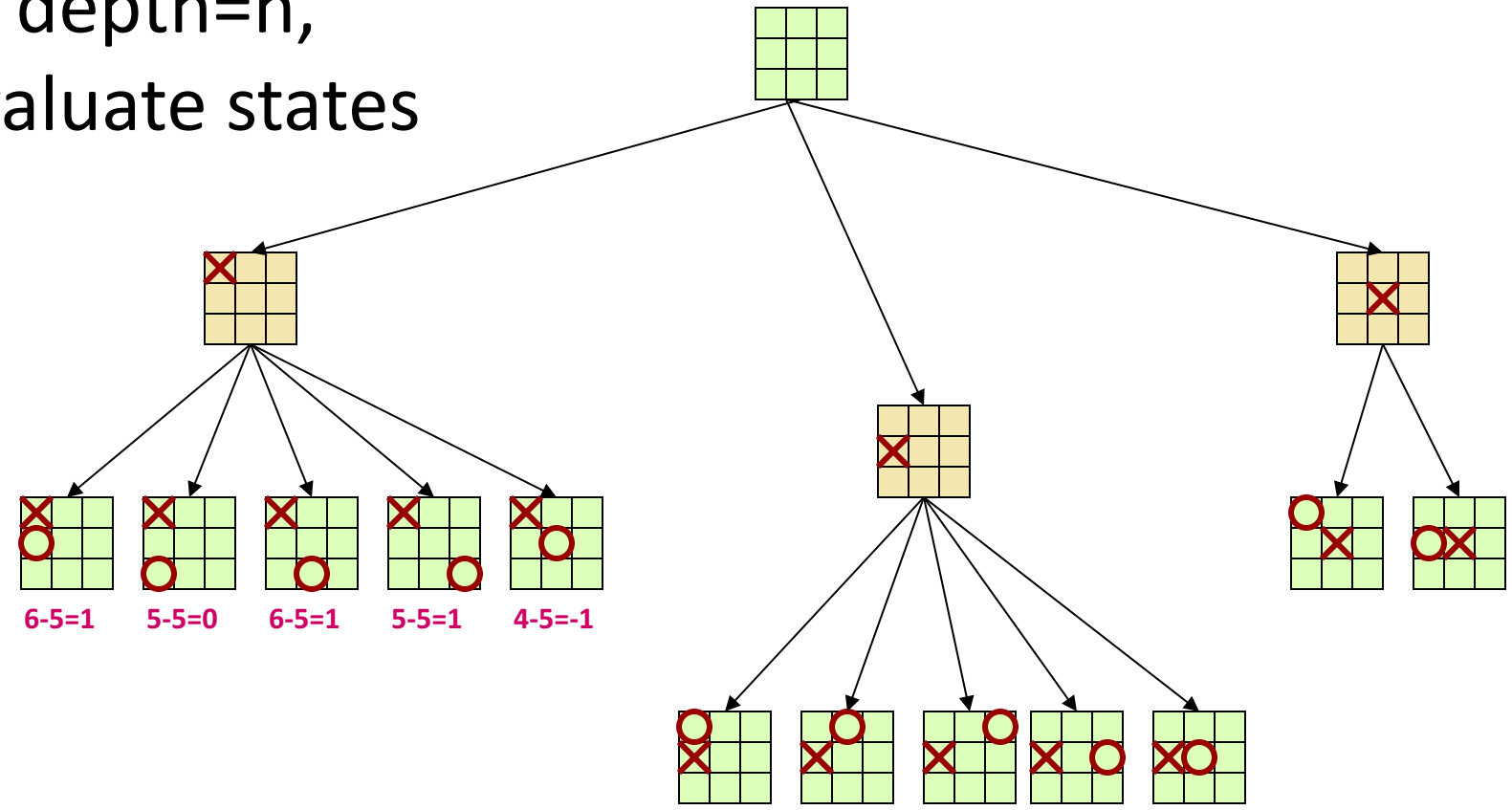
- At depth=h, evaluate states



e(s) = (#row + #col + #diag for MAX)- (#row+#col+#diag for MIN)

Propagating e-values to root (first move)

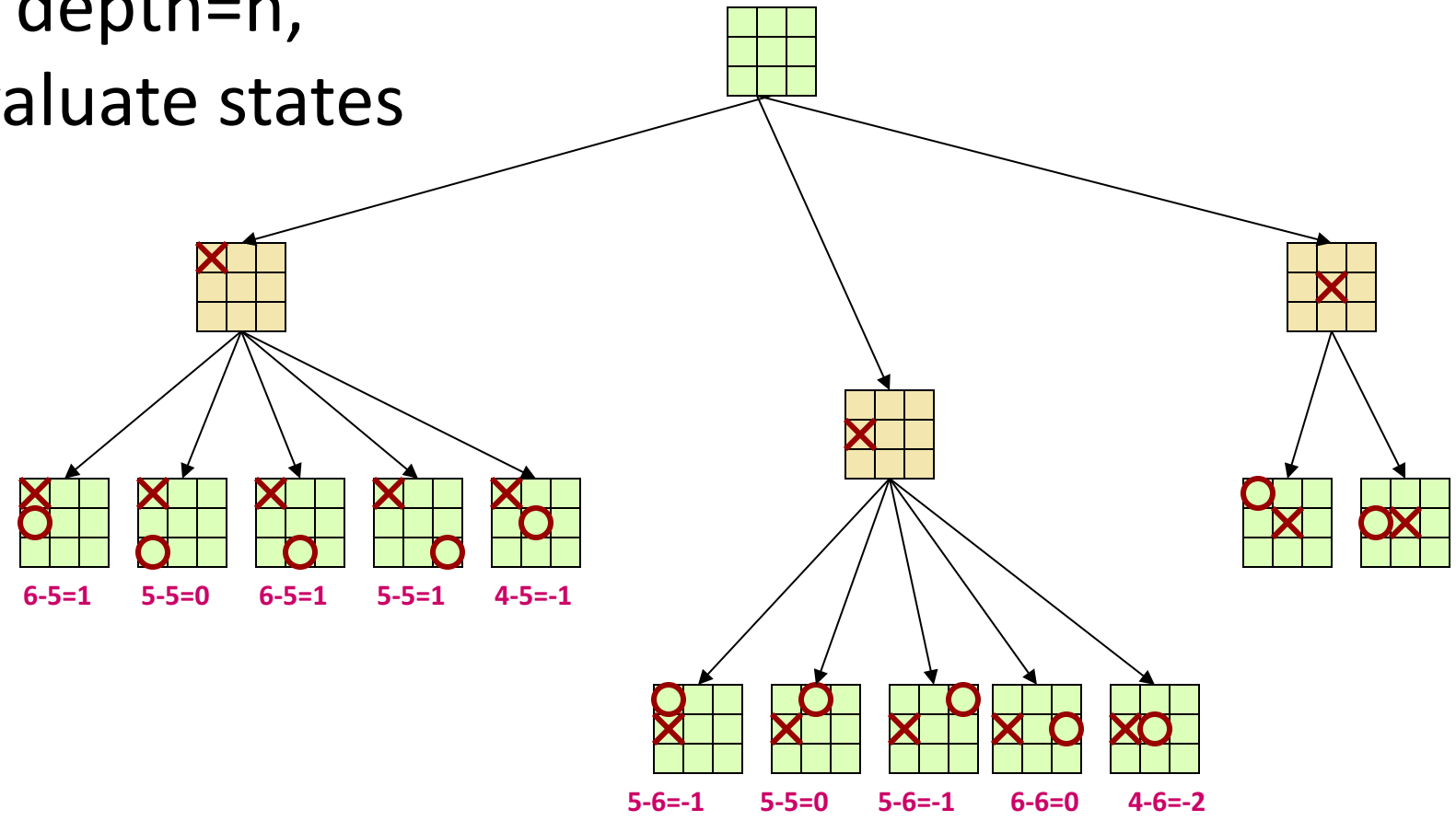
- At depth=h, evaluate states



e(s) = (#row + #col + #diag for MAX)- (#row+#col+#diag for MIN)

Propagating e-values to root (first move)

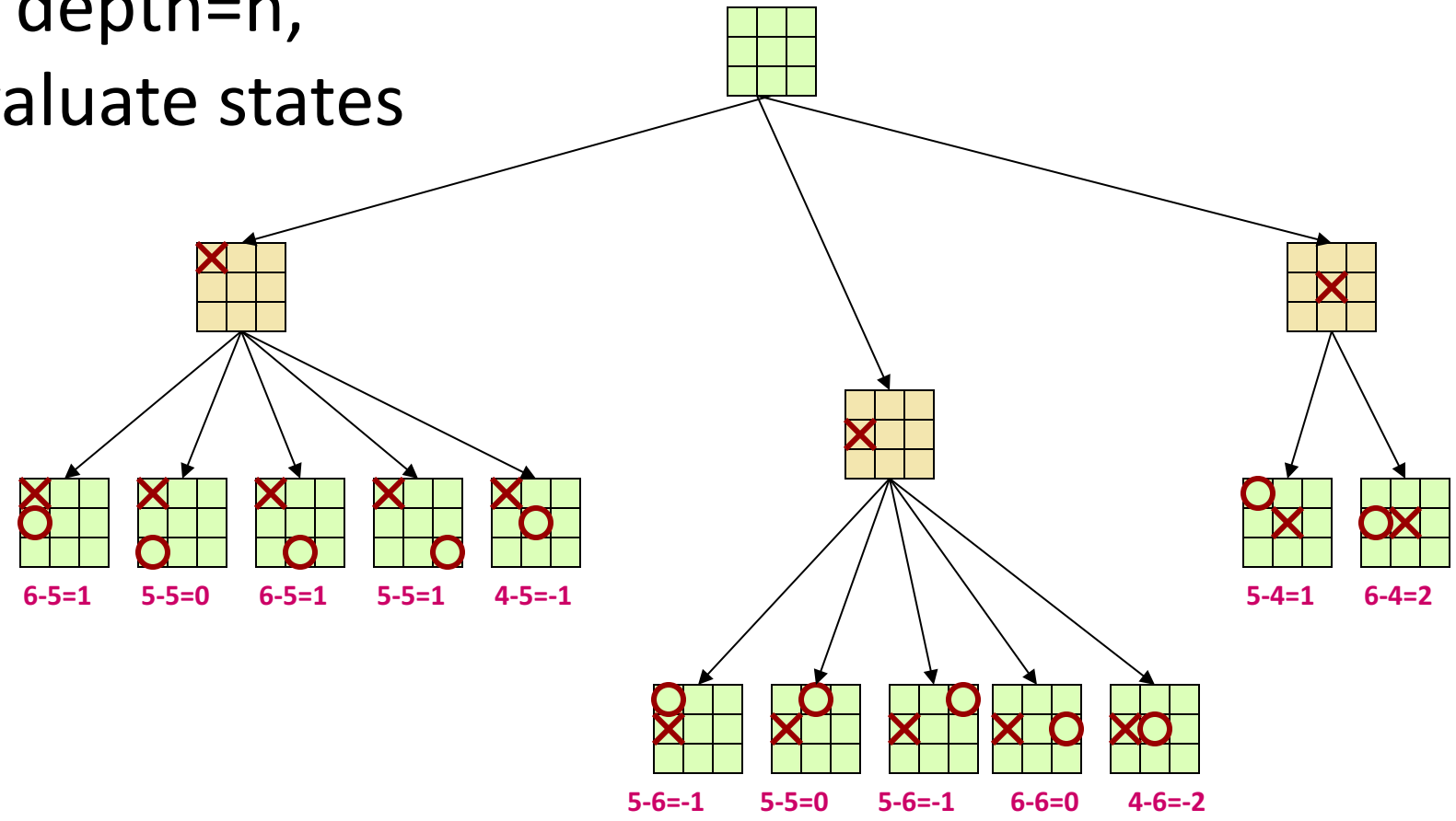
- At depth=h, evaluate states



e(s) = (#row + #col + #diag for MAX)- (#row+#col+#diag for MIN)

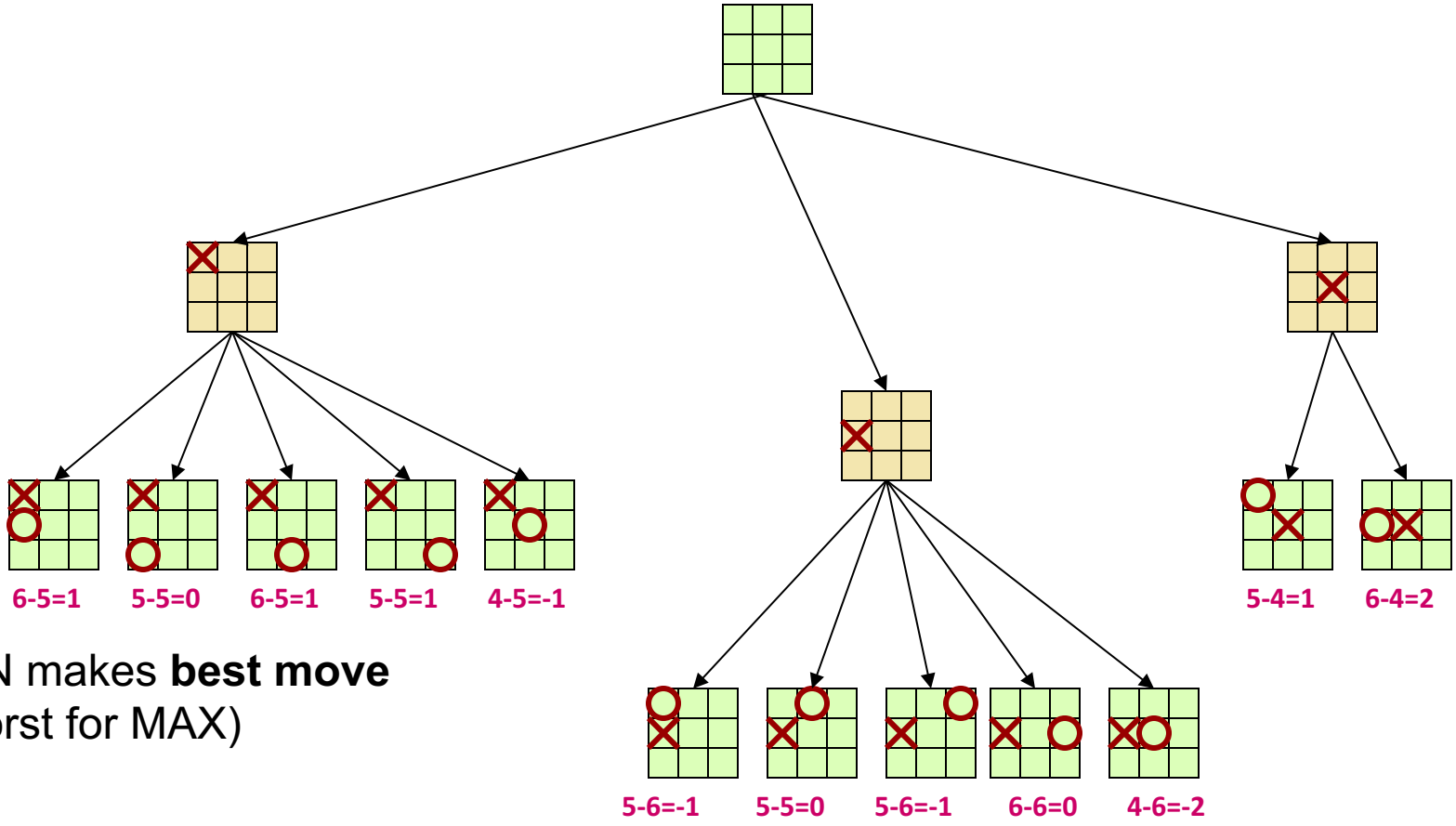
Propagating e-values to root (first move)

- At depth=h, evaluate states



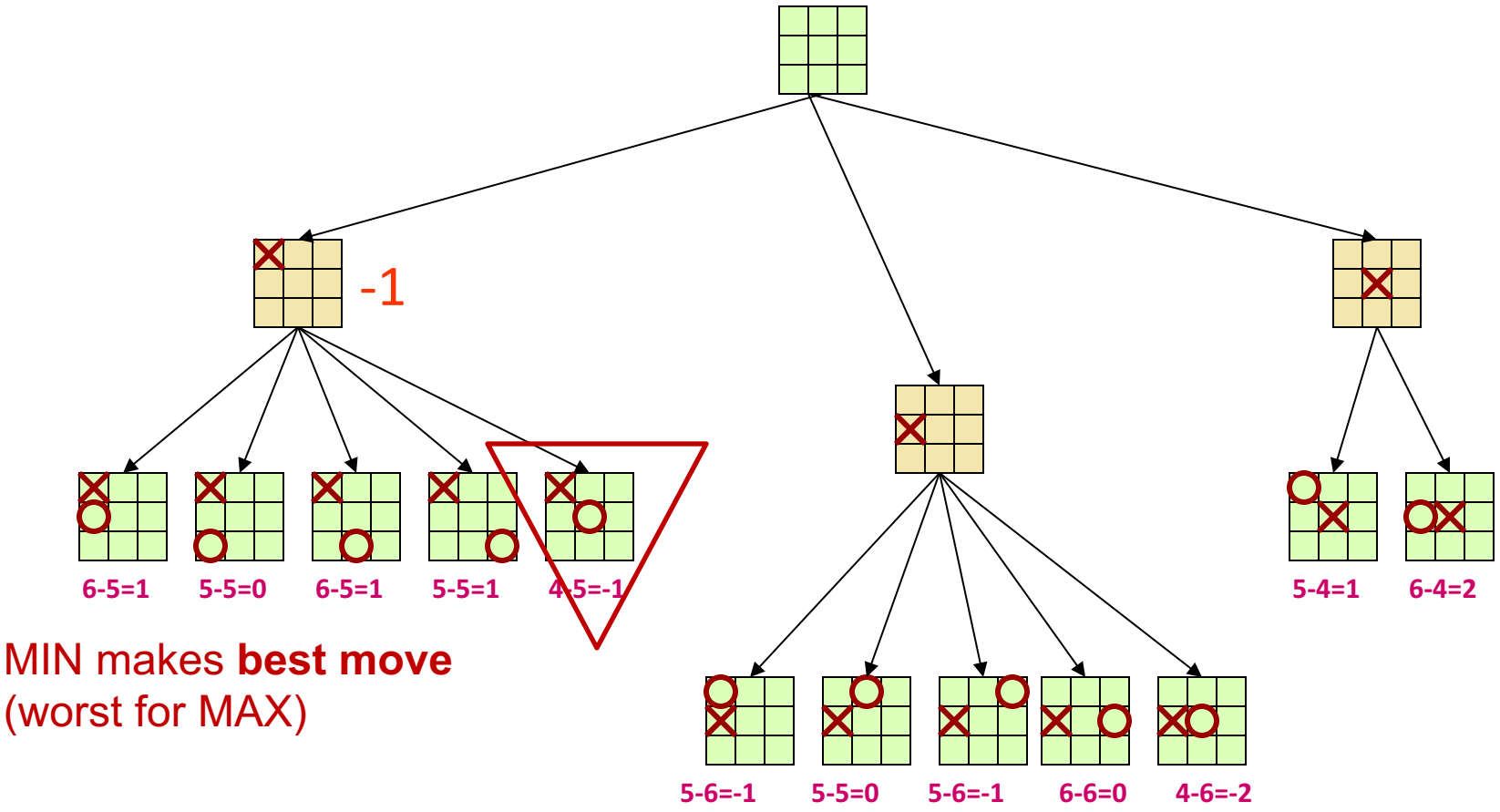
e(s) = (#row + #col + #diag for MAX)- (#row+#col+#diag for MIN)

Propagating e-values to root (first move)



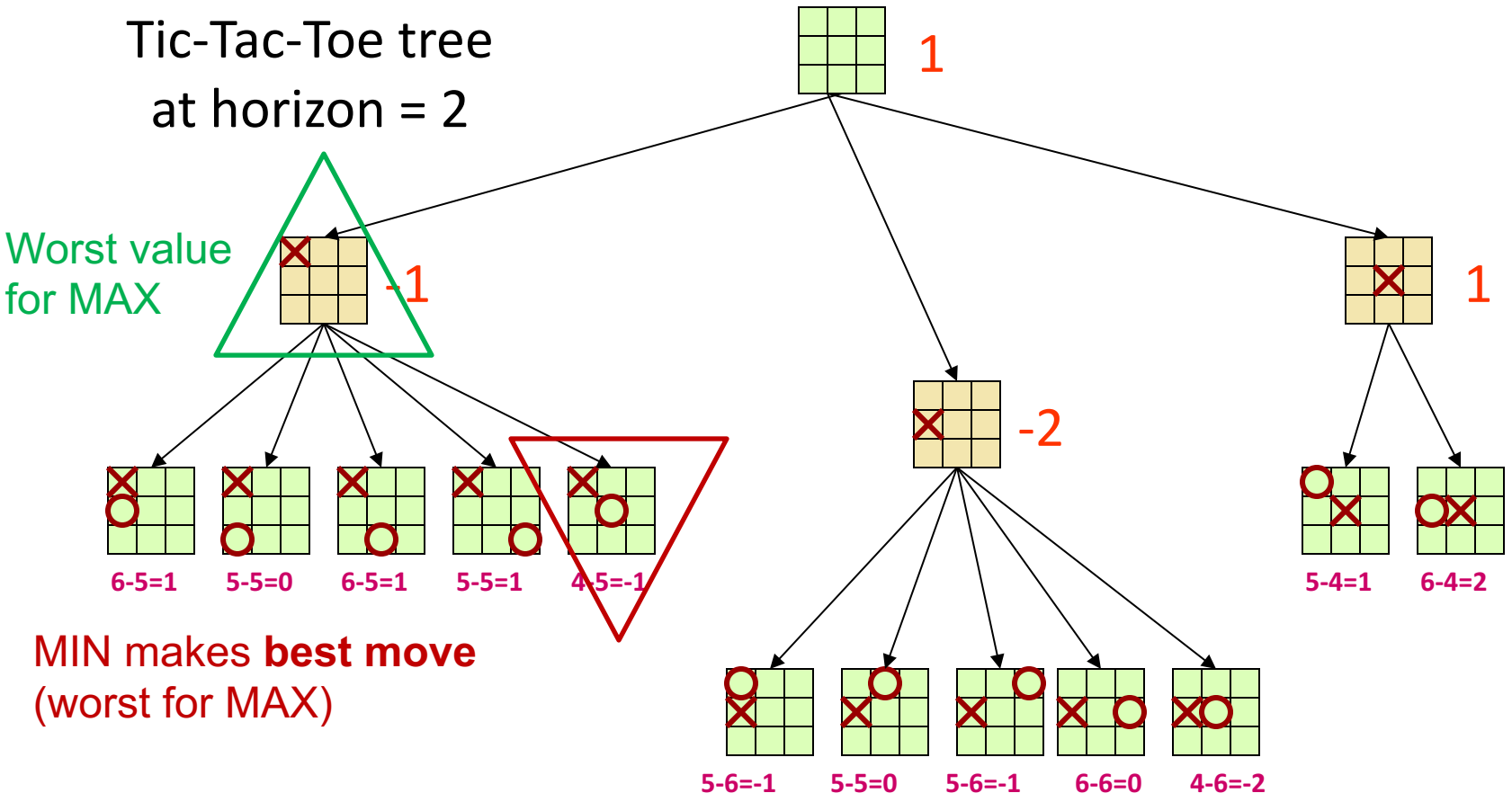
MIN makes **best move**
(worst for MAX)

Propagating e-values to root (first move)

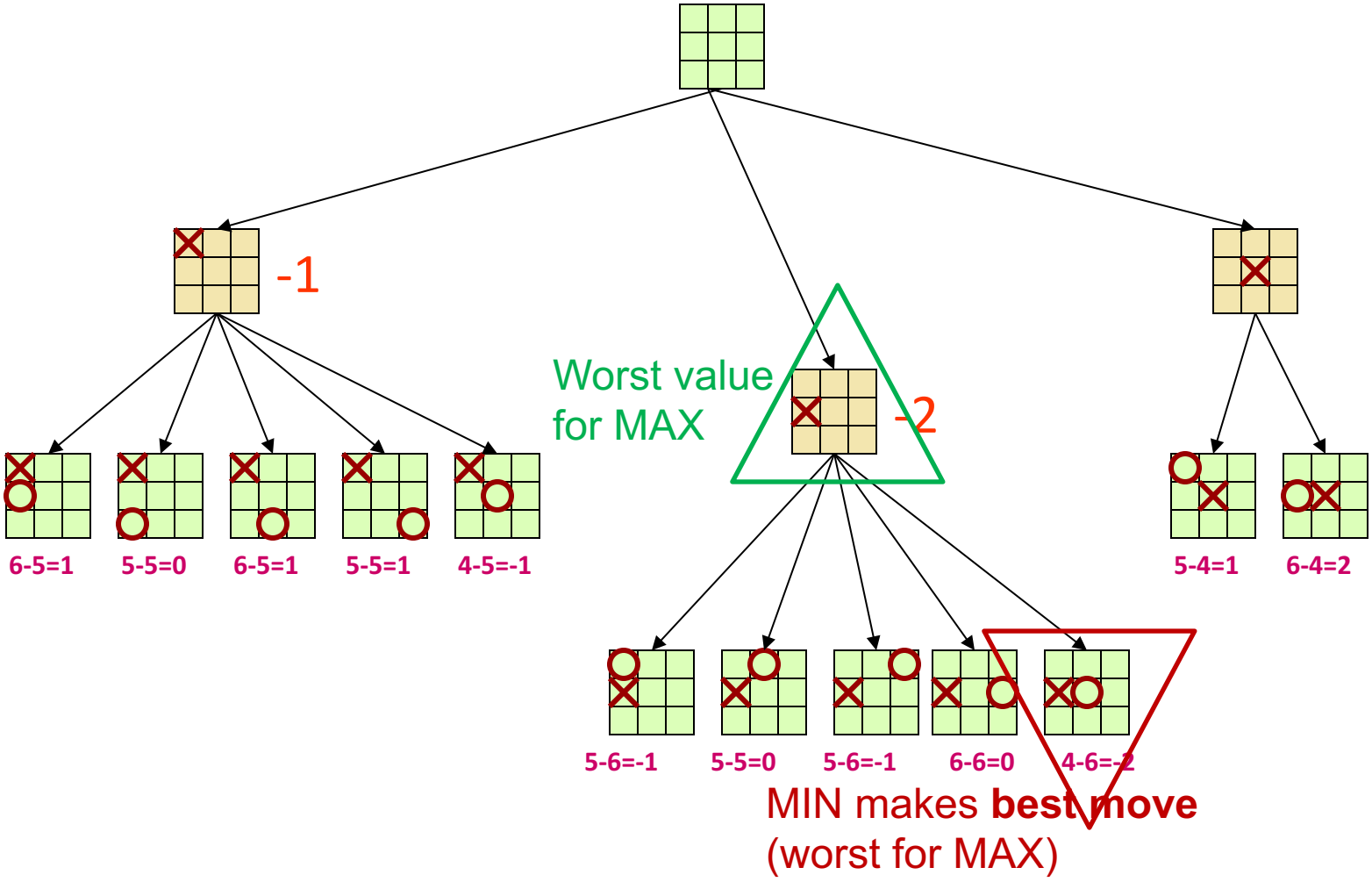


MIN makes **best move**
(worst for MAX)

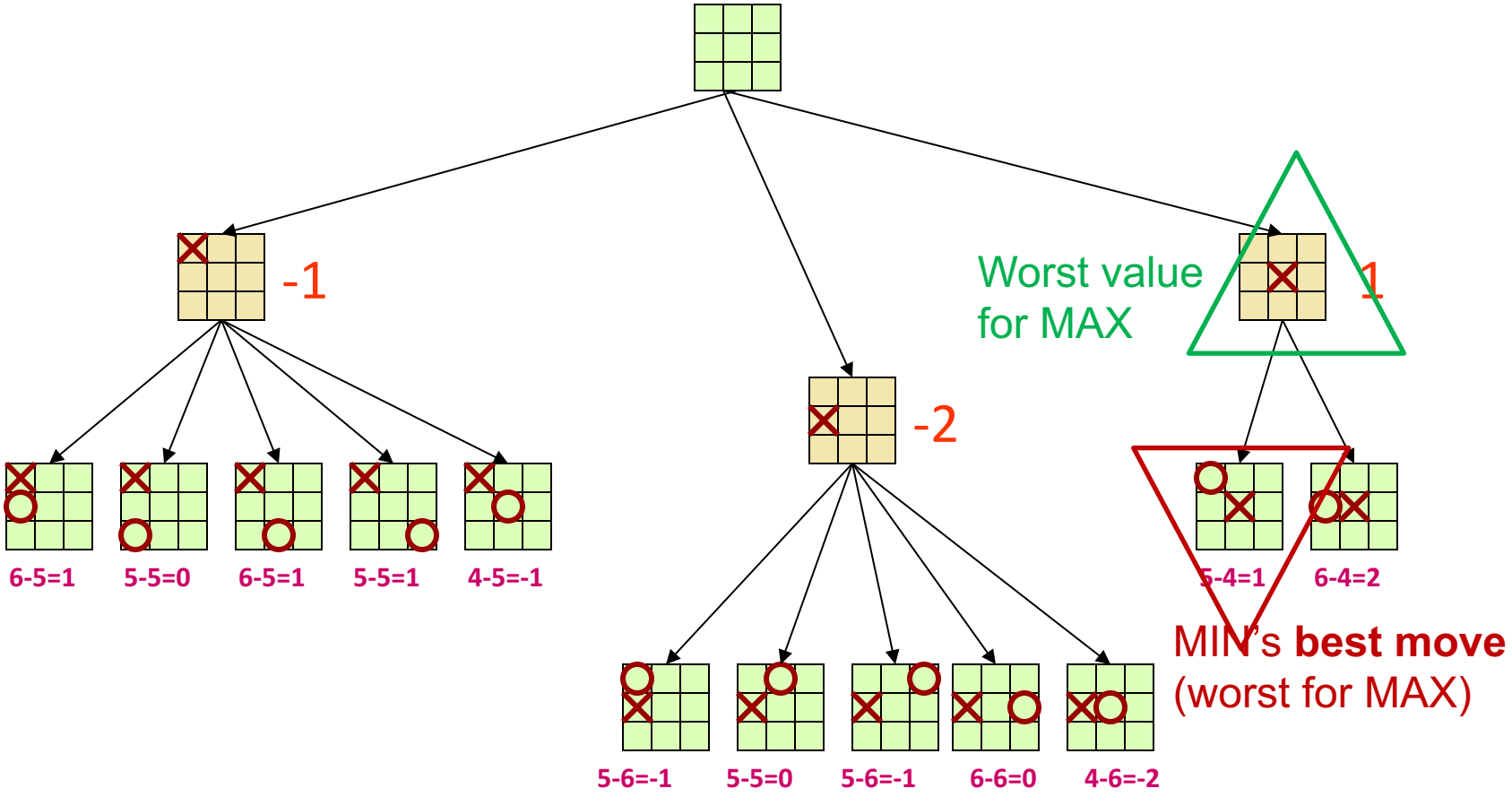
Propagating e-values to root (first move)



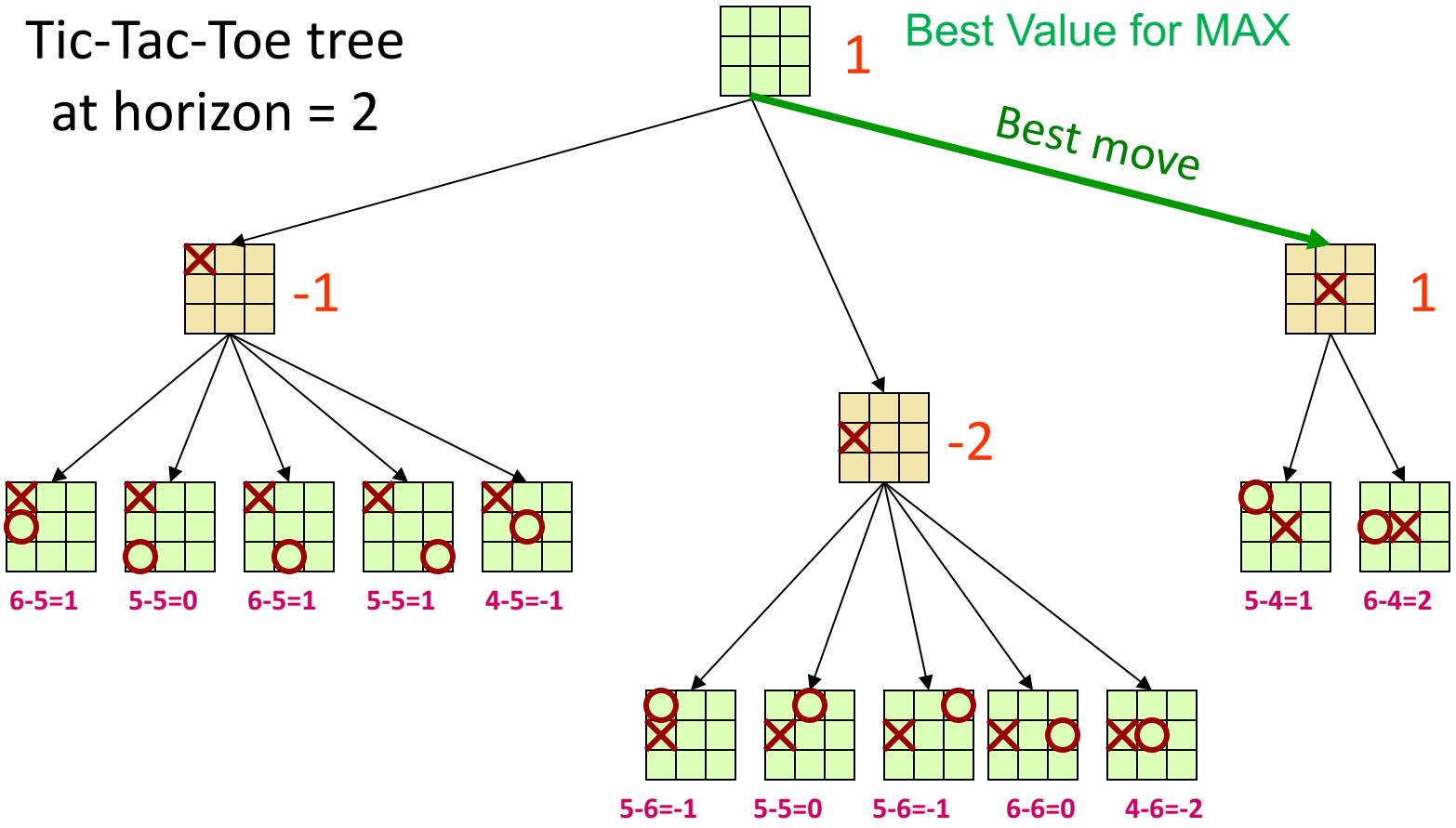
Propagating e-values to root (first move)



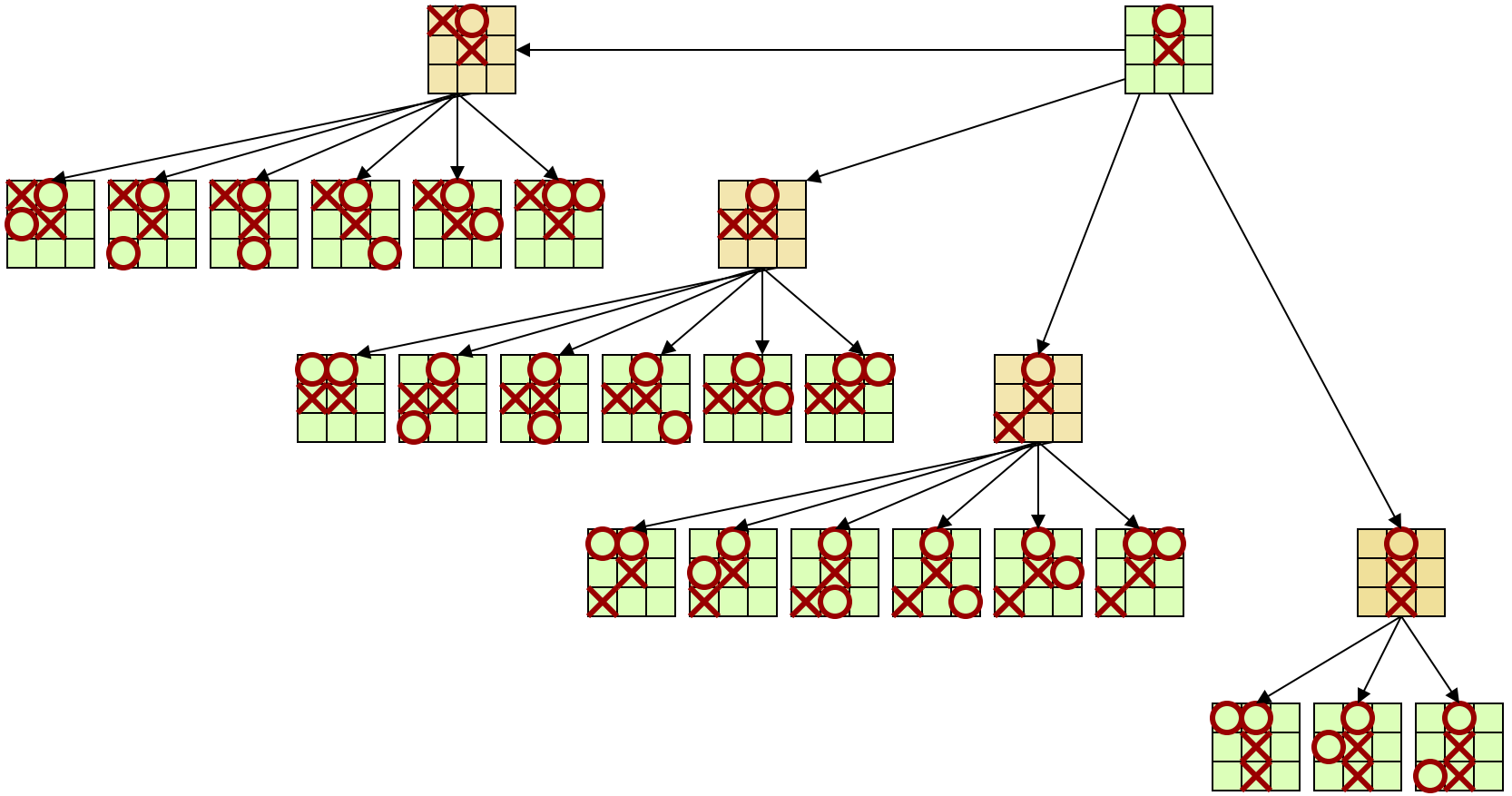
Propagating e-values to root (first move)



Propagating e-values to root (first move)

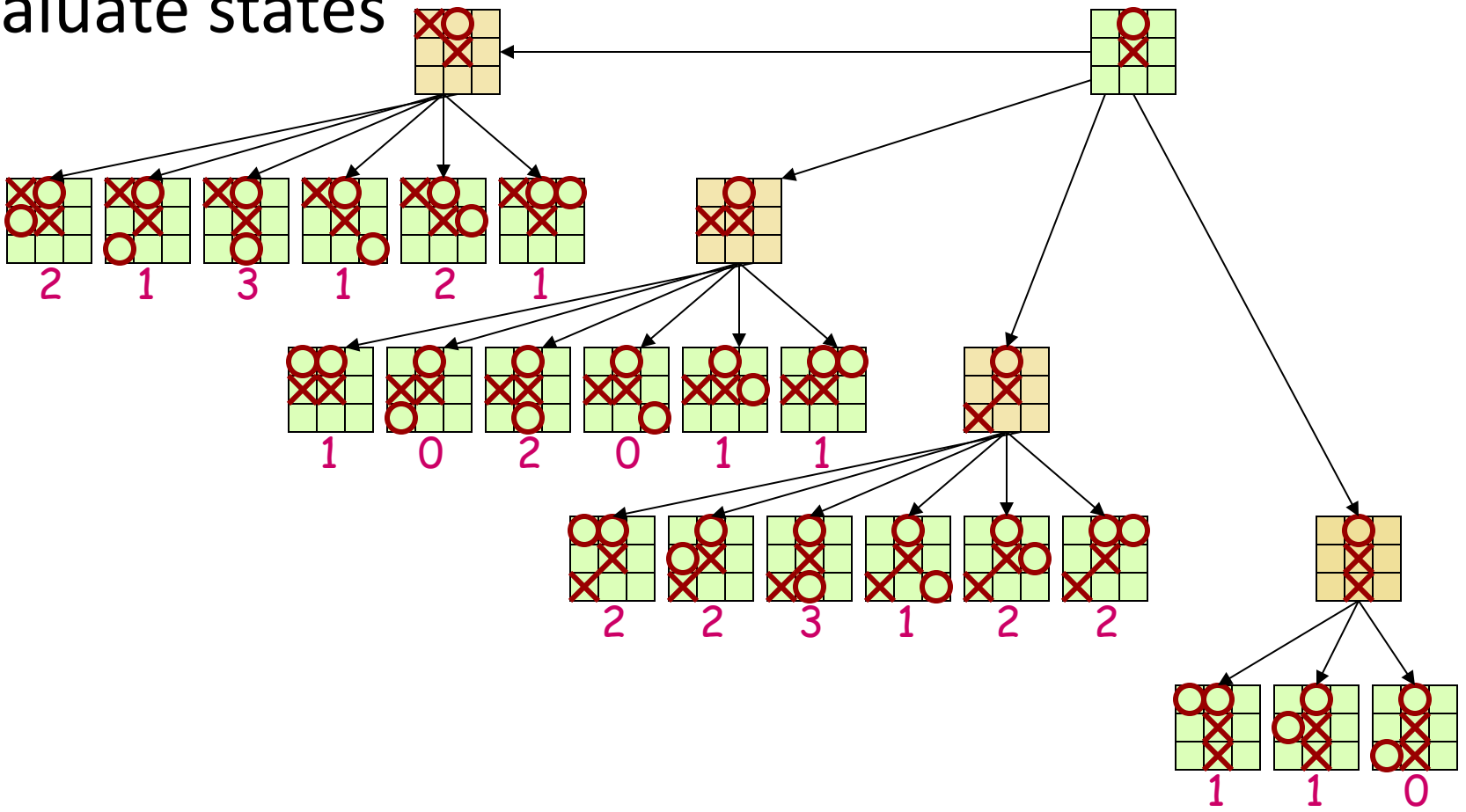


Propagating values (next move)

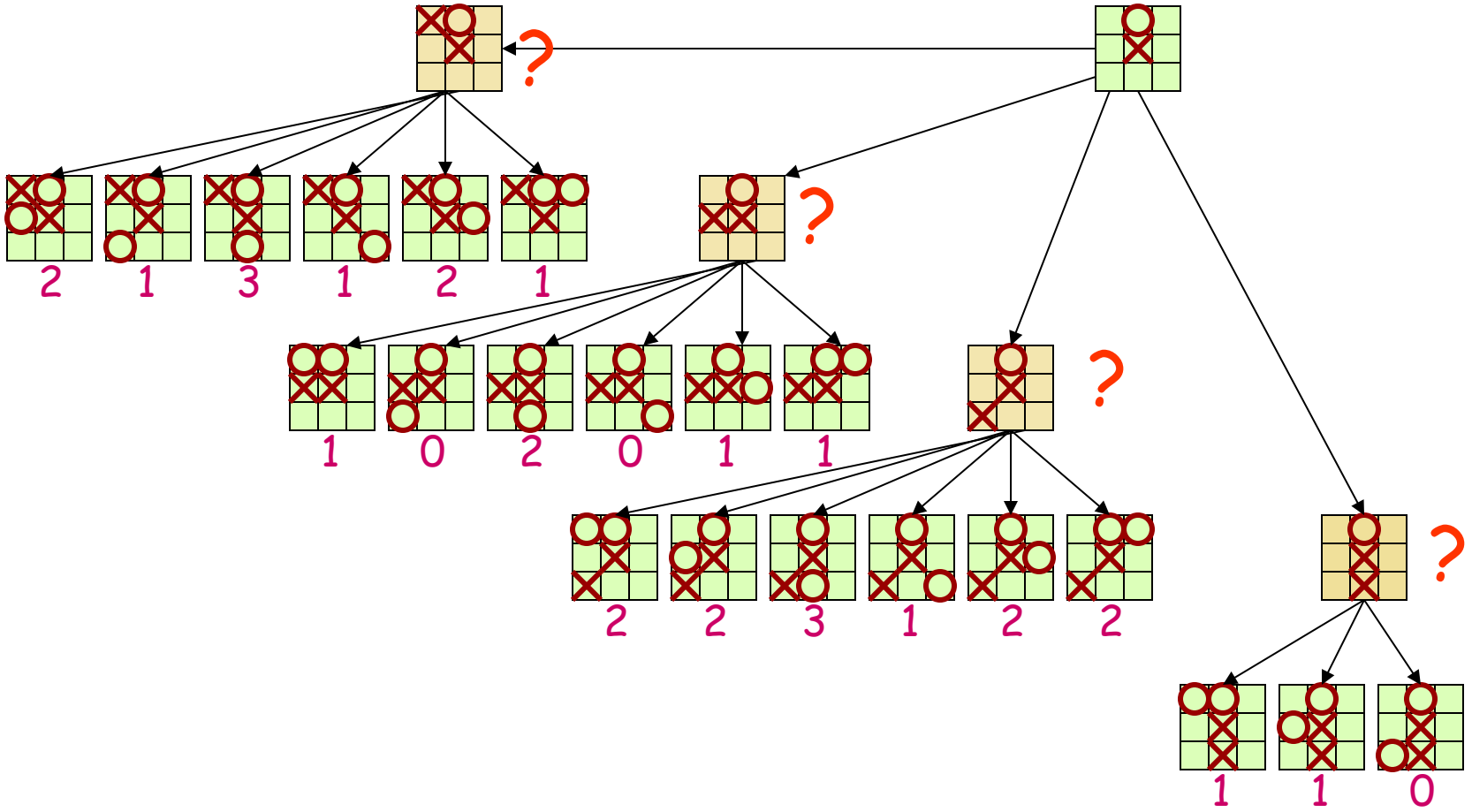


Propagating values (next move)

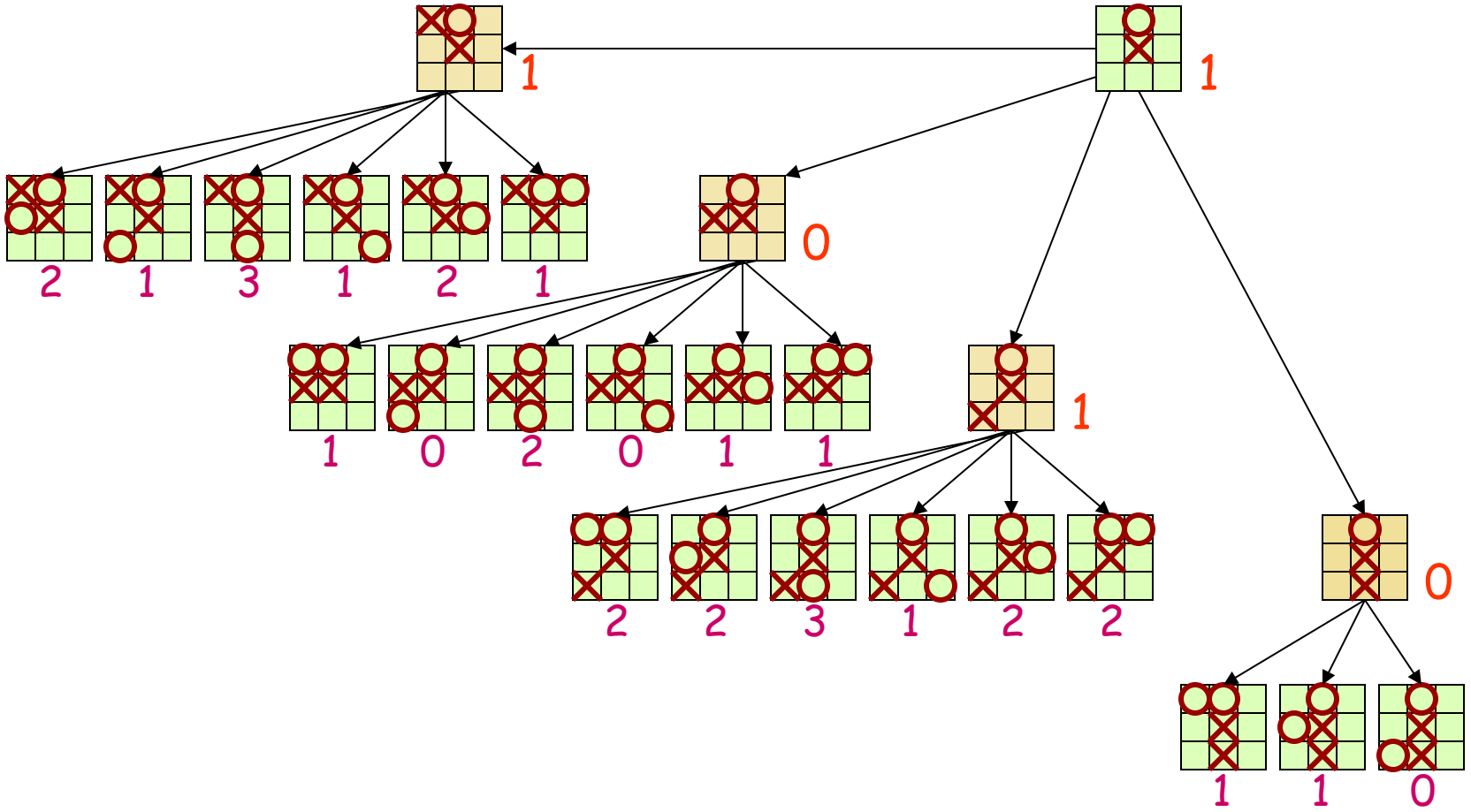
- Evaluate states



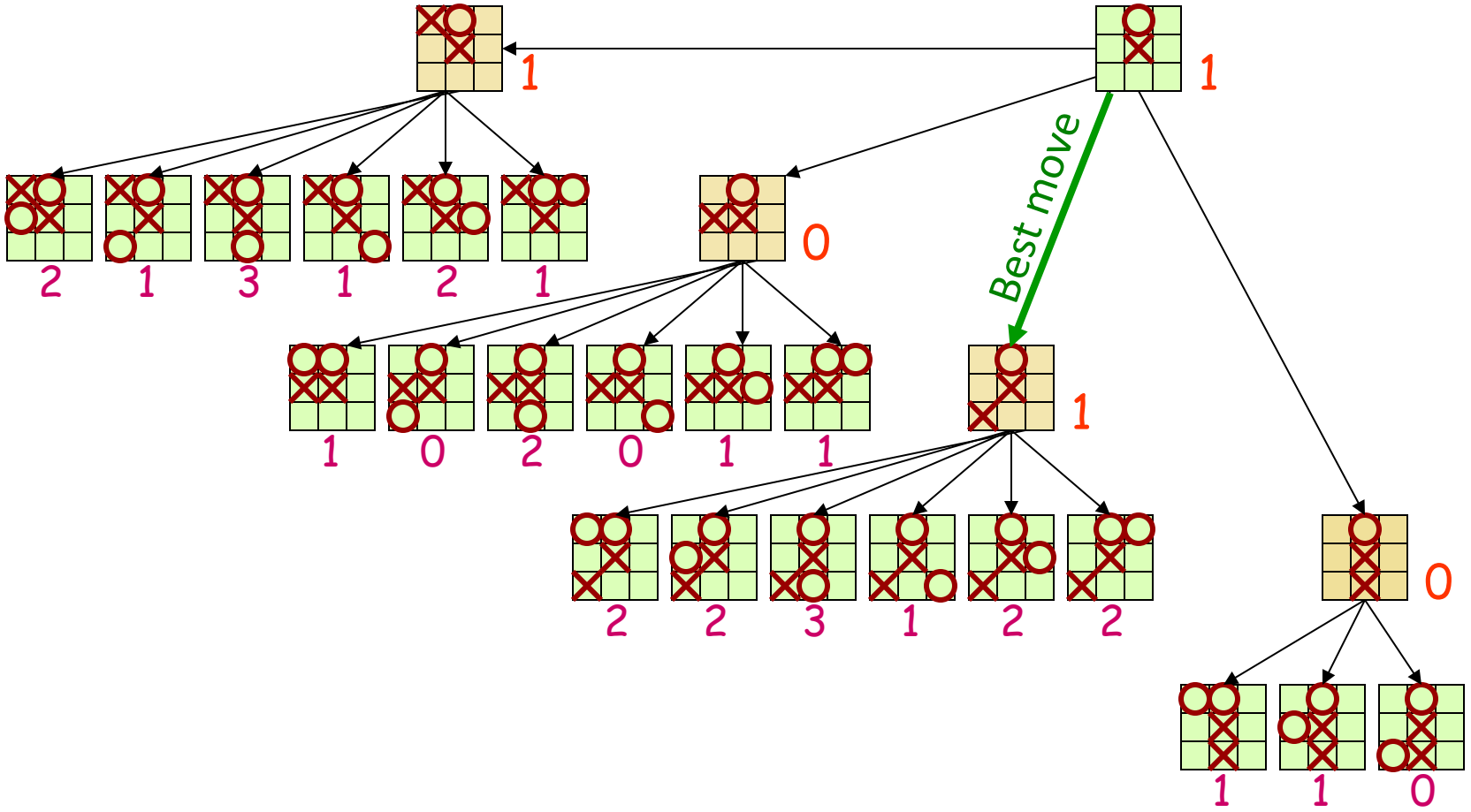
Propagating values (next move)



Propagating values (next move)



Propagating values (next move)



Why use propagated values instead of original *e*?

- At each non-leaf node N, the value is the highest value of the best state that MAX can reach at depth h, if MIN plays well (by the same criterion as MAX applies to itself)
- Evaluation function ***e*** (estimate) improves closer to terminal (win/lose) states
- If ***e*** is to be trusted in the first place, then the **backed-up value is a better estimate** of how favorable STATE(N) is than ***e***(STATE(N))

Minimax Algorithm

1. On each turn, expand the game tree uniformly (**BFS**) from the current state to depth **h**
2. Compute the evaluation function at **every leaf** of the tree
3. Back-up (propagate) the values from the leaves to the root of the tree as follows:
   a. A MAX node gets the maximum of the evaluation of its successors
   b. A MIN node gets the minimum of the evaluation of its successors
4. Select the move toward a MIN node that has the largest backed-up value