6.S890: Topics in Multiagent Learning

Lecture 17 – Prof. Farina

Scalability-enhancing techniques

Fall 2023

Some practical solutions

- Utility computation can be very expensive (huge matrix-vector product) -> Use a sparse unbiased estimator
- … Also maybe your problem admits a small latent space that can help you
- Maybe the information you can receive, or the actions you can play, are too many -> Abstract, that is, bucket them and treat them the same
- Maybe your strategy is not very good -> Improve it locally as you play, for the situation you’re specifically encountering!

Sampling

Utility computation is too expensive!

Recall: How do we use no-external-regret algorithms in two-player zero-sum normal-form or extensive-form games?

$$\max_{x \in X} \min_{y \in Y} x^T A y$$

X, Y = Simplex for normal-form games

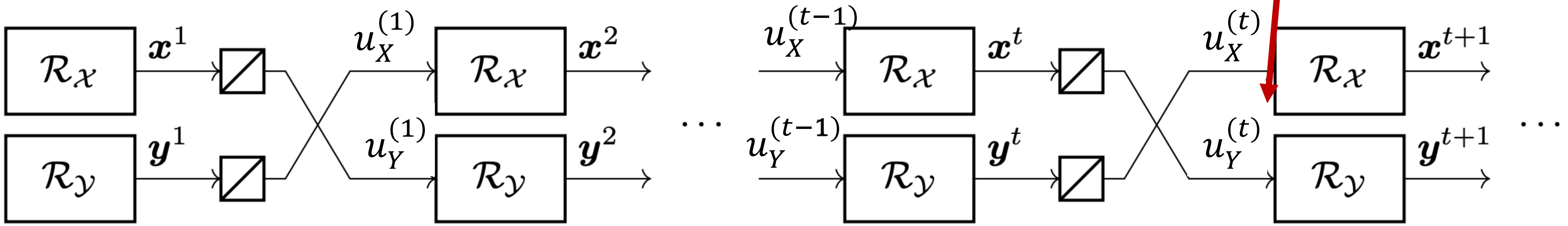
X, Y = sequence-form polytope for extensive-form games

Answer: we let the learners play against each other

Q: What utilities do we supply to the learners?

$$u_X^{(t)} := A y^{(t)} \qquad u_Y^{(t)} := -A^T x^{(t)}$$

(Gradients of the players' utility functions)



Idea: Monte-Carlo CFR

- Remember: CFR works by orchestrating a tree of local regret minimizers (one per decision point)
  - At each time t, each regret minimizers outputs a local behavioral strategy
  - Then, when a utility $u^{(t)}$ is received as feedback by CFR, $u^{(t)}$ is used to construct counterfactual utilities by considering the expected utility in each subtree
- At its core, Monte-Carlo CFR uses the observation that if the utility is very sparse, then the expected utilities in each subtree will almost always be 0
  - Therefore, no update of the strategy is necessary for those subtrees, and no regret is cumulated

Idea: Monte-Carlo CFR

The idea of MCCFR is to replace any incoming utility $u^{(t)}$ by a sparse unbiased estimator $\tilde{u}^{(t)}$, that is, a sparse vector $\tilde{u}^{(t)}$ whose expectation is $u^{(t)}$.

Unbiased estimators of Ay

- Warmup in normal-form games
- Extensive-form games:
  - Opponent sampling
  - Outcome sampling

Theoretical Guarantees

- We can bound the degradation in regret incurred by the sampling
- Regret with sampling:
  - $\tilde{R}^{(T)} = \max_x \sum \langle \tilde{u}^{(t)}, x - x^{(t)} \rangle$
- Regret without sampling
  - $R^{(T)} = \max_x \sum \langle u^{(t)}, x - x^{(t)} \rangle$

Bounds on the diameter of the utilities:

$$M = \max_{x,x'} \langle u^{(t)}, x - x' \rangle$$

$$\tilde{M} = \max_{x,x'} \langle \tilde{u}^{(t)}, x - x' \rangle$$

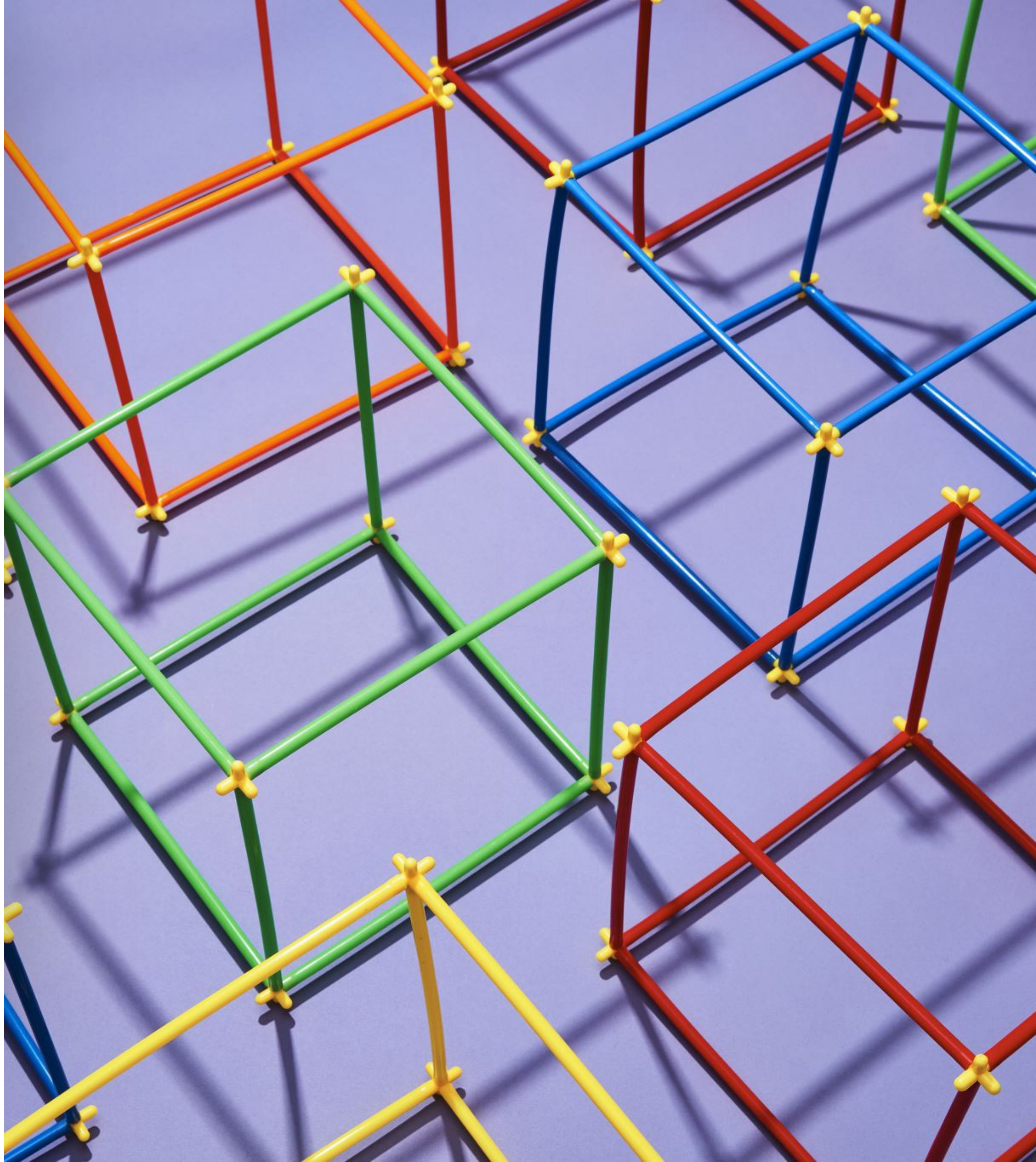
Theorem: No matter the sequence of utility vectors $u^{(t)}$, the difference between $R^{(T)}$ and $\tilde{R}^{(T)}$ is bounded as $\tilde{R}^{(T)} \leq R^{(T)} + (M + \tilde{M})\sqrt{2T \log \frac{1}{\delta}}$ with probability at least $1 - \delta$ for all $\delta \in (0,1)$.

Exploiting the structure of the payoff matrix

… Maybe you have a small latent space?

Sparsification

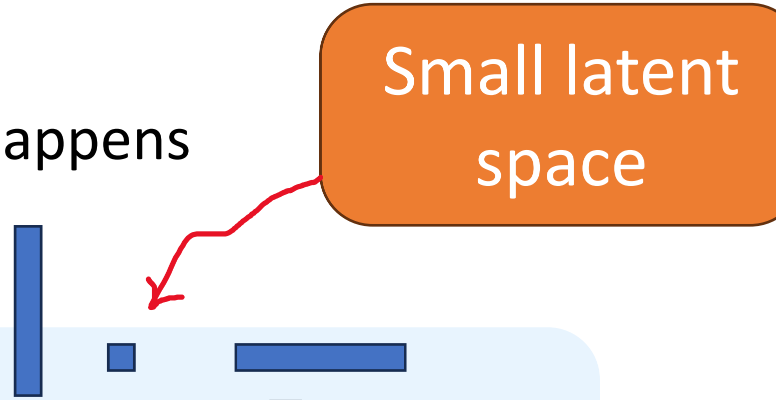
- Many games have a strong combinatorial structure
- This structure can inform opportunity for speedups
- Idea of **sparsification**: exploit a low-rank utility matrix



Example

- For example, it can be shown that in a game like poker, the payoff matrix can be written as a sum of Kronecker products
  - In other words, the payoff matrix has a low-rank block structure

- Intuition: it's a sum of two block matrices
  - First matrix controls the payoffs when a fold happens
  - Second matrix controls the payoffs when a showdown happens
  - The blocks correspond to the hands of the players



Bottom line: payoff matrix is $A = \hat{A} + UM^{-1}V^T$

Recall: LP Formulation

$$Q_1 = \begin{cases} F_1 x = f_1 \\ x \geq 0 \end{cases} \qquad Q_2 = \begin{cases} F_2 y = f_2 \\ y \geq 0 \end{cases}$$

$$\max_{x \in Q_1} \min_{y \in Q_2} x^T A y$$

① →

$$\begin{cases} \max & \begin{cases} \min & x^T A y \\ \text{s.t.} & y \in Q_2 \end{cases} \\ \text{s.t.} & x \in Q_1 \end{cases}$$

② →

$$\begin{cases} \max & \begin{cases} \min & x^T A y \\ \text{s.t.} & F_2 y = f_2 \\ & y \geq 0 \end{cases} \\ \text{s.t.} & F_1 x = f_1 \\ & x \geq 0 \end{cases}$$

③ Dualize!

$$\begin{cases} \max & \begin{cases} \max & f_2 v \\ \text{s.t.} & F_2^T v \leq A^T x \\ & v \in \mathbb{R} \end{cases} \\ \text{s.t.} & F_1 x = f_1 \\ & x \geq 0 \end{cases}$$

④ →

$$\begin{cases} \max & f_2 v \\ & F_1 x = f_1 \\ \text{s.t.} & F_2^T v \leq A^T x \\ & x \geq 0 \\ & v \in \mathbb{R} \end{cases}$$

Single linear program!

First application: Sparsification of LP

Payoff matrix is $A = \hat{A} + UM^{-1}V^T$

Original LP

$$\begin{cases} \max & f_2 v \\ \text{s.t.} & F_1 x = f_1 \\ & F_2^T v \leq A^T x \\ & x \geq 0 \\ & v \in \mathbb{R} \end{cases}$$

$$\begin{cases} \max & f_2 v \\ \text{s.t.} & F_1 x = f_1 \\ & F_2^T v \leq \hat{A}^T x + Vw \\ & M^T w - U^T x = 0 \\ & x \geq 0 \\ & v, w \in \mathbb{R} \end{cases}$$

Sparsified LP

Some data from poker

Game	Unsparsified size	Sparsification Size	Sparsification Time
River 7	5.09×10^7	2.74×10^5	318ms
River 6	6.03×10^7	2.70×10^5	454ms
River 8	9.59×10^7	3.93×10^5	436ms
River 2	1.77×10^8	6.72×10^5	567ms
River 4	2.21×10^8	7.76×10^5	624ms
River 1	4.47×10^8	1.60×10^6	699ms
River 3	4.76×10^8	1.65×10^6	722ms
River 5	4.79×10^8	1.65×10^6	733ms

Roughly 2 orders of magnitude reduction

Poker endgames can be solved in seconds

[Farina and Sandholm, "Fast Payoff Matrix Sparsification Techniques for Structured Extensive-Form Games", AAAI'22]

Second applicatio

Opportunity for speedup using low-rank decomposition of A?

Recall: How do we u
algorithms in two-pla
extensive-form games?

$$\max_{x \in X} \min_{y \in Y} x^T A y$$

X, Y = Simplex for normal-form games

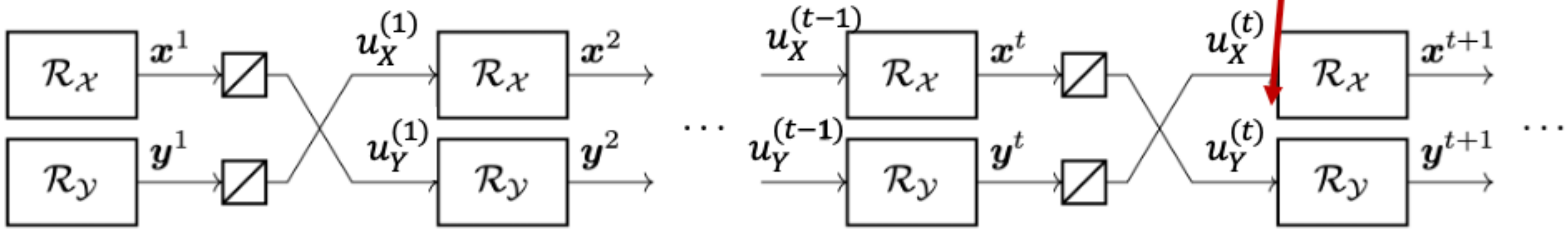
X, Y = sequence-form polytope for extensive-form games

Answer: we let the learners play against each other

Q: What utilities do we supply to the learners?

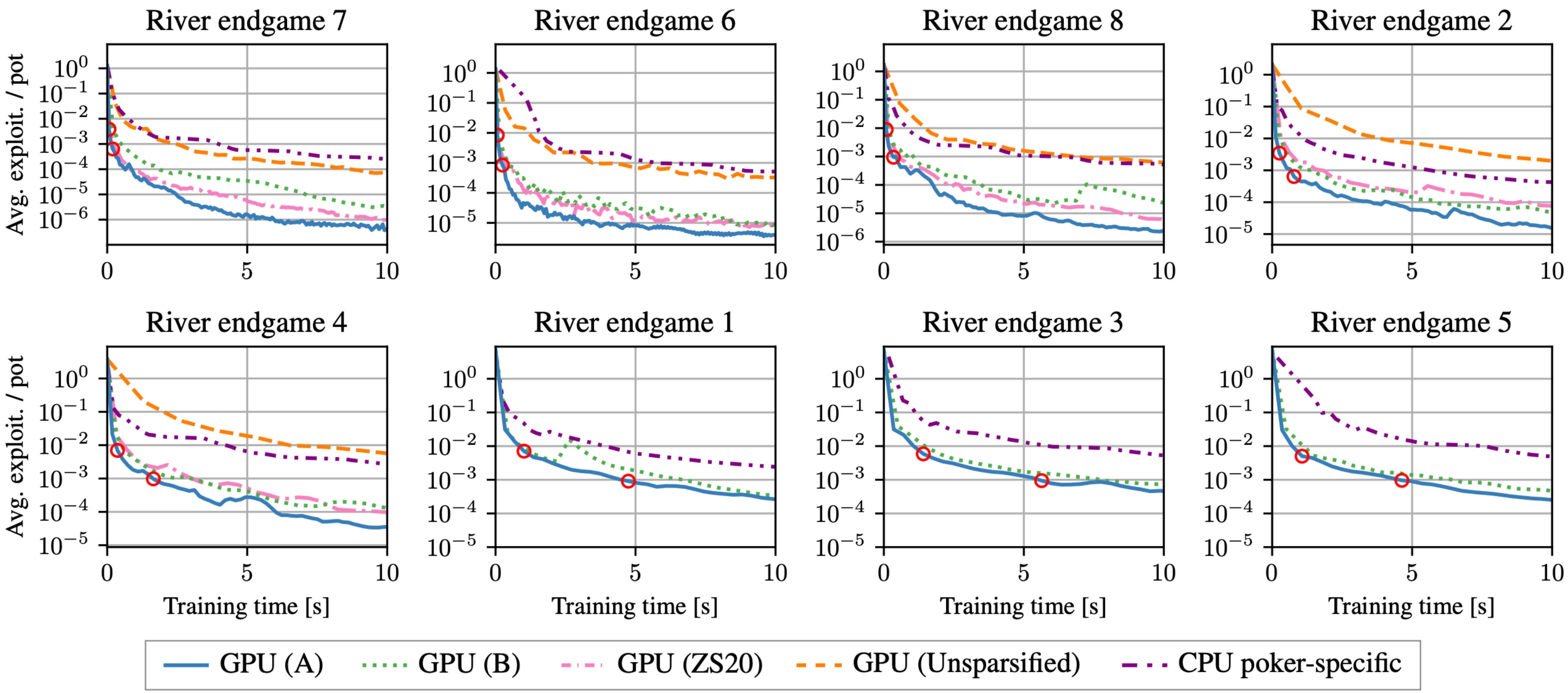
$$u_X^{(t)} := A y^{(t)} \qquad u_Y^{(t)} := -A^T x^{(t)}$$

(Gradients of the players' utility functions)



Second application

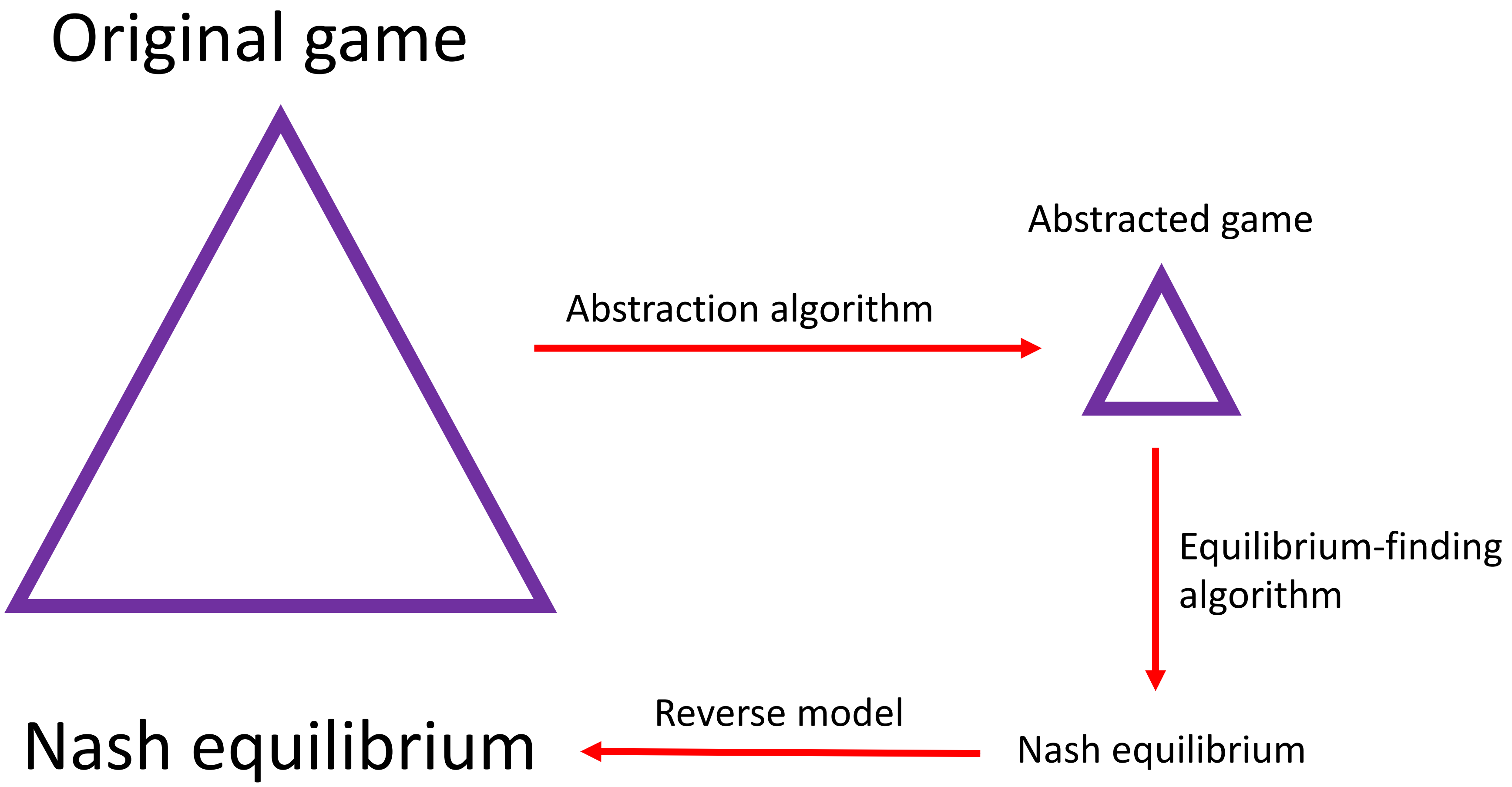
- Performs really well on the GPU too



Information and Action Abstraction

The game is too big! Make is smaller

The basic idea



[Gilpin & Sandholm EC-06, *J. of the ACM* 2007…] Foreshadowed by Shi & Littman 01 and Billings et al. IJCAI-03

Two approaches

Lossless Abstraction

Exploits structural properties of the game to compress the action space without changing the equilibria

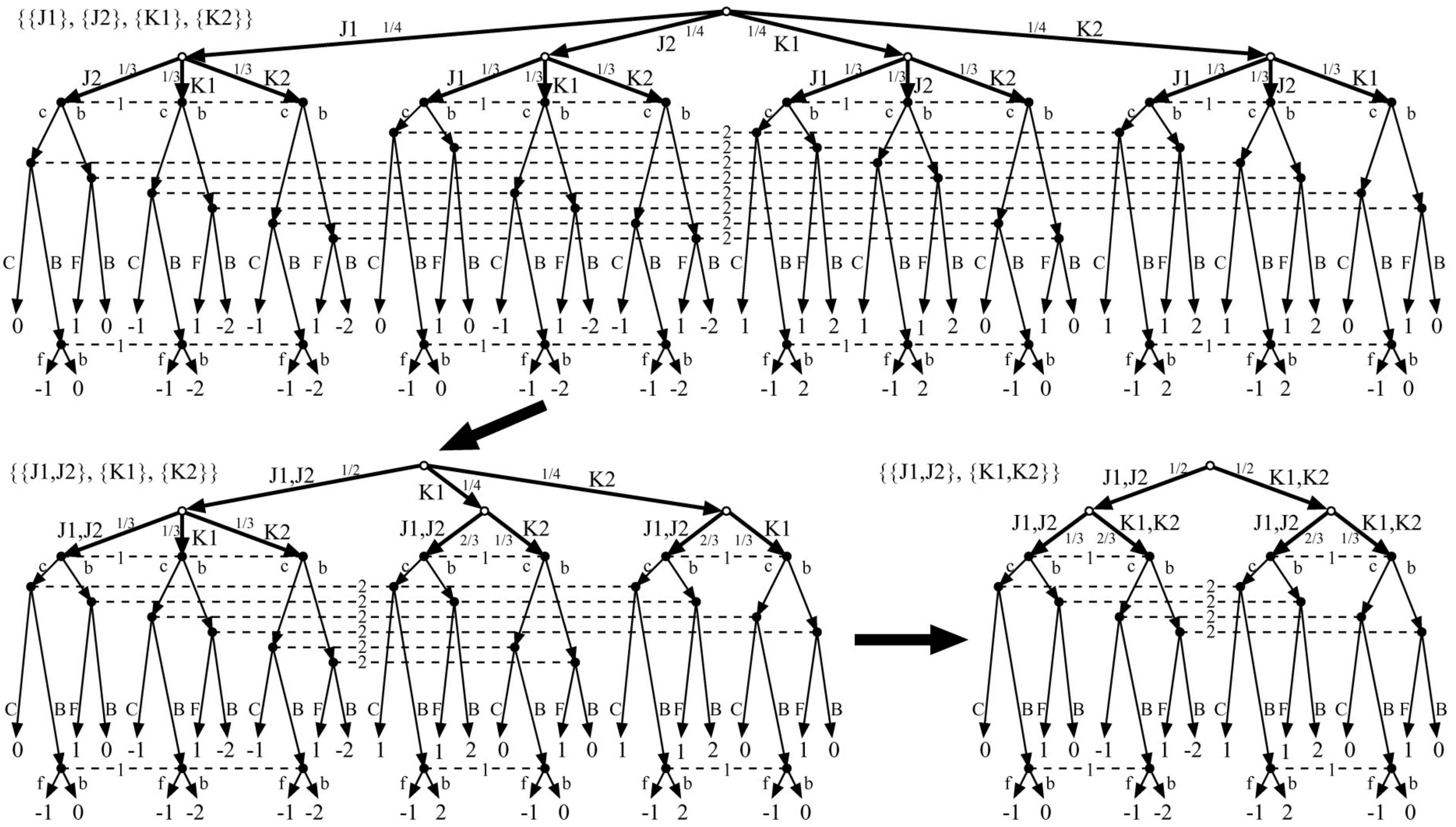
Lossy Abstraction

Compresses the game forcefully to a point where it can be solved

Lossless Abstraction

Lossless abstraction was mostly pioneered in an attempt to tackle poker

- Observation: We can make games smaller by filtering the *information* a player receives
- Instead of observing a specific signal exactly, a player instead observes a **filtered set** of signals
  - *E.g.* receiving signal {A♠,A♣,A♥,A♦} instead of A♥
- Fundamentally, lossless abstraction works by isolating **isomorphisms** between different scenarios
  - For example, your strategy should be blind to the specific suit of the cards, and only depends on whether the suits match or differ


{{J1}, {J2}, {K1}, {K2}}
{{J1,J2}, {K1}, {K2}}
{{J1,J2}, {K1,K2}}

GameShrink algorithm

[Gilpin and Sandholm, JACM'07]

- **Bottom-up pass**: Run dynamic programming to discover isomorphism in the game
- **Top-down pass**: Then, starting from top of the tree, perform the transformation where applicable

- Implementation details complex, but it is able to operate the passes implicitly with respect to the game tree, by constructing a succinct representation called a **signal tree**

Solved *Rhode Island Hold'em* poker

- AI challenge problem [Shi & Littman 01]
  - **3.1 billion nodes (!)** in game tree
- Without abstraction, LP has 91,224,226 rows and columns => unsolvable
- GameShrink runs in one second
- After that, LP has 1,237,238 rows and columns (50,428,638 non-zeros)
- Solved the LP
  - CPLEX barrier method took 8 days & 25 GB RAM
- Exact Nash equilibrium
- **Historical significance:** Largest incomplete-info game solved by then by over 4 orders of magnitude

Slide credits: Tuomas Sandholm

Solved *Rhode Island Hold'em* poker

- AI challenge problem [Shi & Littman 01]
  - **3.1 billion nodes (!)** in game tree
- Without abstraction, LP has 91,224,226 rows and columns => unsolvable
- GameShrink runs in one second
- After that, LP has 1,237,238 rows and columns (50,428,638 non-zeros)
- Solved the LP
  - CPLEX barrier method took 8 days & 25 GB RAM
- Exa
- **His** ... e solved by

Bottom line: lossless abstraction **reduced the size by 2 orders of magnitude** without losing any strategic property

Slide credits: Tuomas Sandholm

Texas Hold’em Poker

2-player Limit has ~1018 nodes

2-player No-Limit has ~10165 nodes

Lossless abstraction is (way) too big to solve

=> abstract more
=> **we need lossy abstraction**

Slide credits: Tuomas Sandholm

Attempts

- Different approaches to good lossy *information* abstractions:

2006:

- *GameShrink* can be made to abstract more by not requiring a perfect matching => lossy
- For speed of the matching, Gilpin & Sandholm [AAAI-06] used a faster matching heuristic
- Unfortunately the greedy nature of the heuristic results in **lopsided** (unbalanced) abstractions

Attempts

- Different approaches to good lossy abstractions:

<u>2007-2008:</u>

- Prior abstraction algorithms use winning probability as similarity metric
- Problem: Hands like flush draws where although the probability of winning is small, the payoff could be high
- Solution: people started investigating “potential-aware” abstraction

Attempts

- Different approaches to good lossy abstractions:

<u>2009:</u>

- People embraced the idea of imperfect-recall abstractions
  - Abstract by forgetting about past observations

<u>~2018-onward</u>:

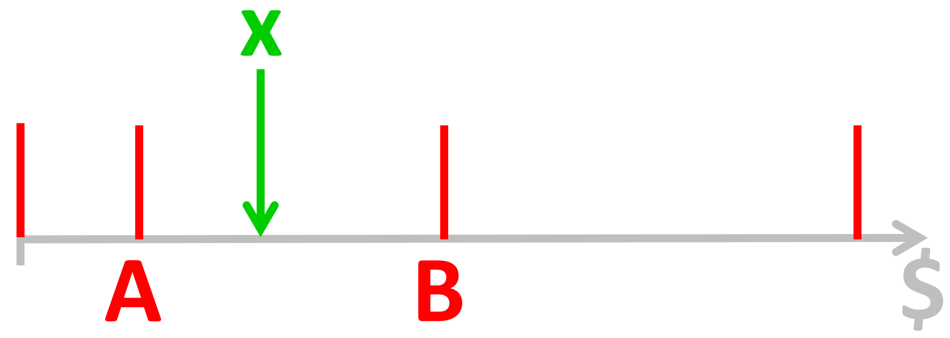
- Information abstraction is taken care implicitly by neural network architecture
  - It is the network that "decides" what information (rank, value, etc.) to retain

What about *action* abstraction?

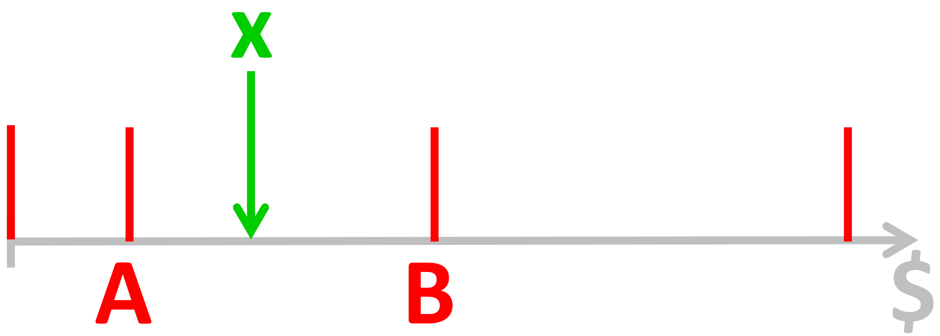
- Typically done manually
- Prior action abstraction algorithms for extensive games (even for just poker) have had no guarantees on solution quality [Hawkin et al. AAAI-11, 12]
- For stochastic games there is an action abstraction algorithm with bounds (based on discrete optimization) [Sandholm & Singh EC-12]

Problem: Action Translation

- Suppose in our abstraction we have discretized bet $ amounts for our opponents to only be A or B.
- But now in the game we see some different amount x. How should we play?



Action *translation*



f(x) ≡ probability we map x to A

Desiderata about f

1. f(A) = 1,   f(B) = 0
2. Monotonicity
3. Scale invariance
4. Small change in x doesn't lead to large change in f
5. Small change in A or B doesn't lead to large change in f

"Pseudo-harmonic mapping"

- $f(x) = [(B-x)(1+A)] / [(B-A)(1+x)]$
- Derived from Nash equilibrium of a simplified no-limit poker game
- Satisfies the desiderata
- Much less exploitable than prior mappings in simplified domains
- Performs well in practice in no-limit Texas Hold'em

Slide credits: Tuomas Sandholm

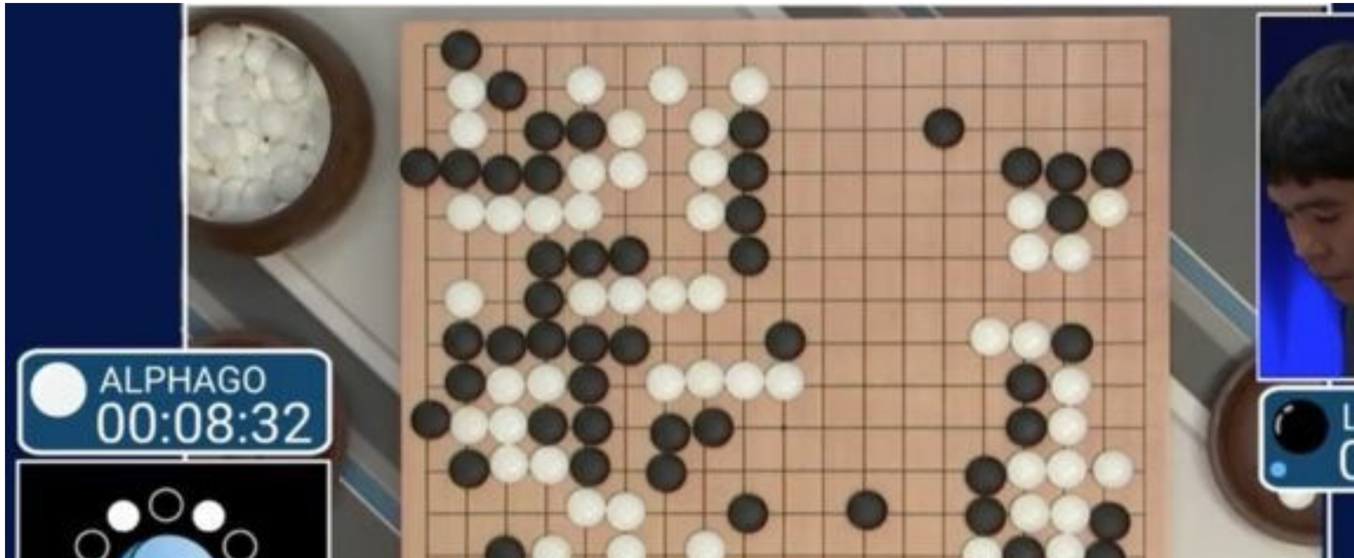
[Ganzfried & Sandholm IJCAI-13]

Decision-time planning (aka search)

The general idea

- In large games, we will never be able to compute exact equilibria
- In fact, we are lucky if we get somewhat close to equilibrium ("blueprint")

- Big idea: decision-time planning (e.g., Monte Carlo Tree Search)
  - Key technique for solving go
  - We refine, on the fly, the blueprint strategy in the subtree in which we are playing, just before playing the next move



Perfect-information games

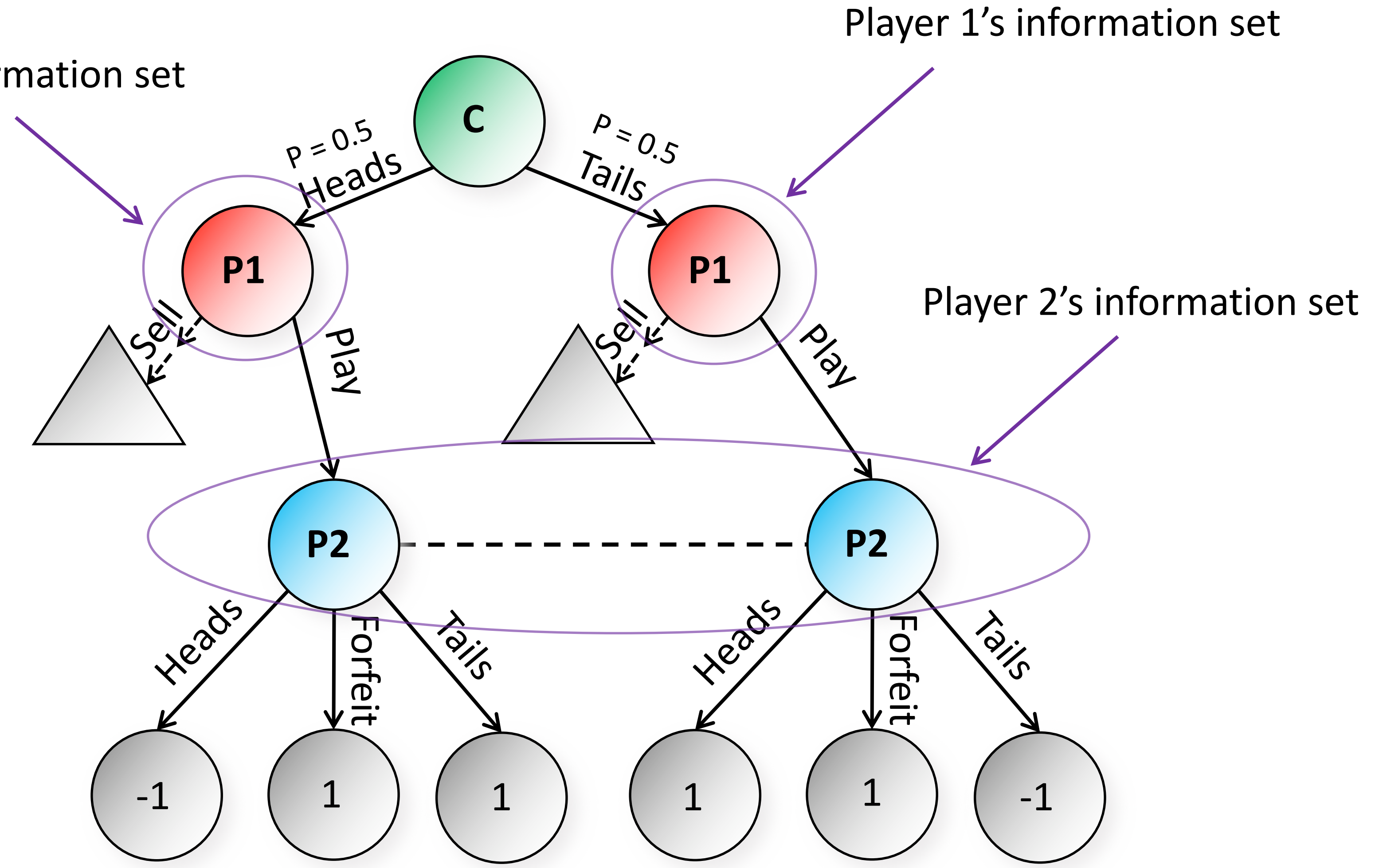
Sicilian Defense

Queen’s Gambit

- Subgames can be solved with information from the subgame only
- This is **not true** in imperfect-information games

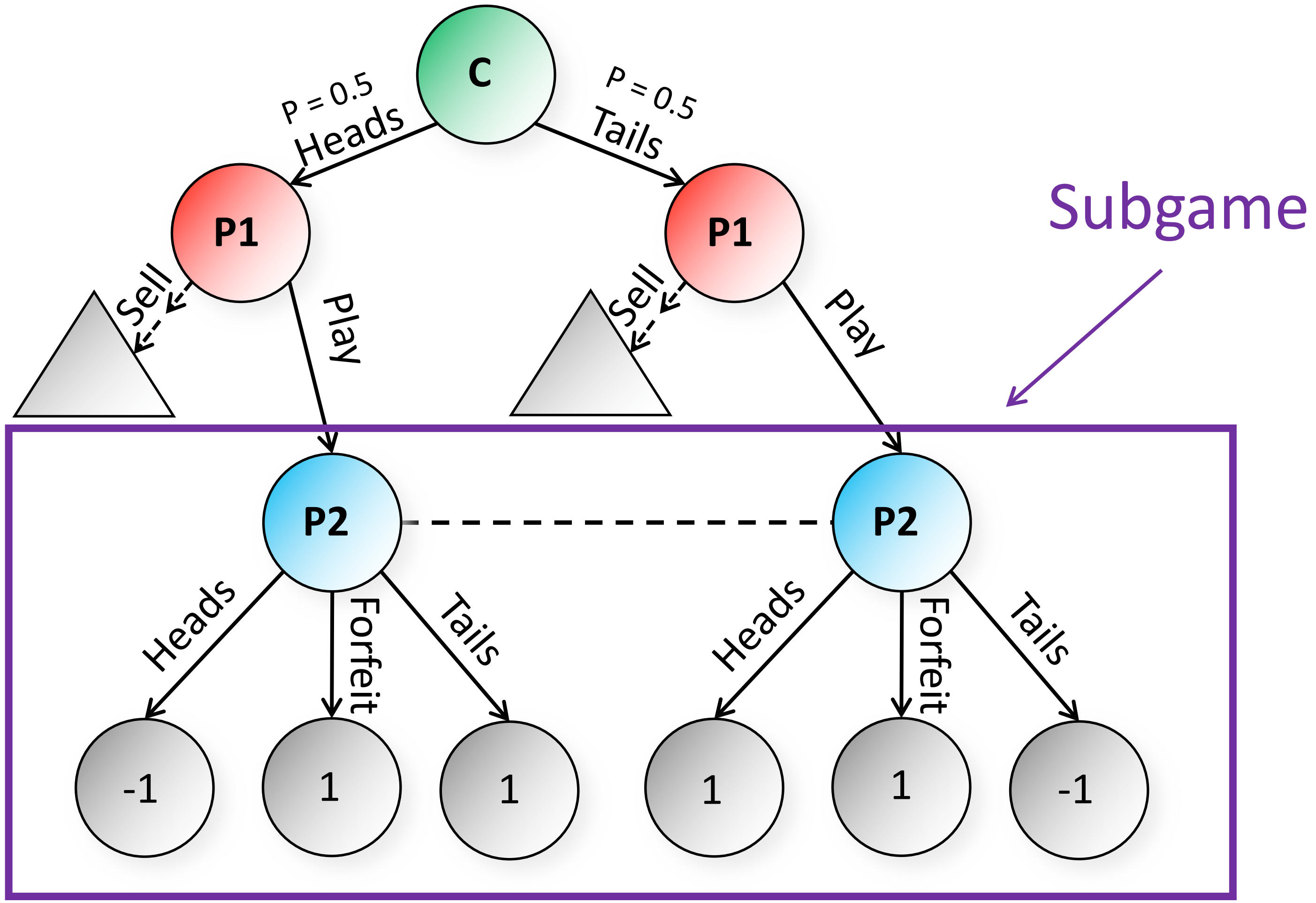
Slide credits: Tuomas Sandholm

Imperfect-information games
Example game: "Coin toss"



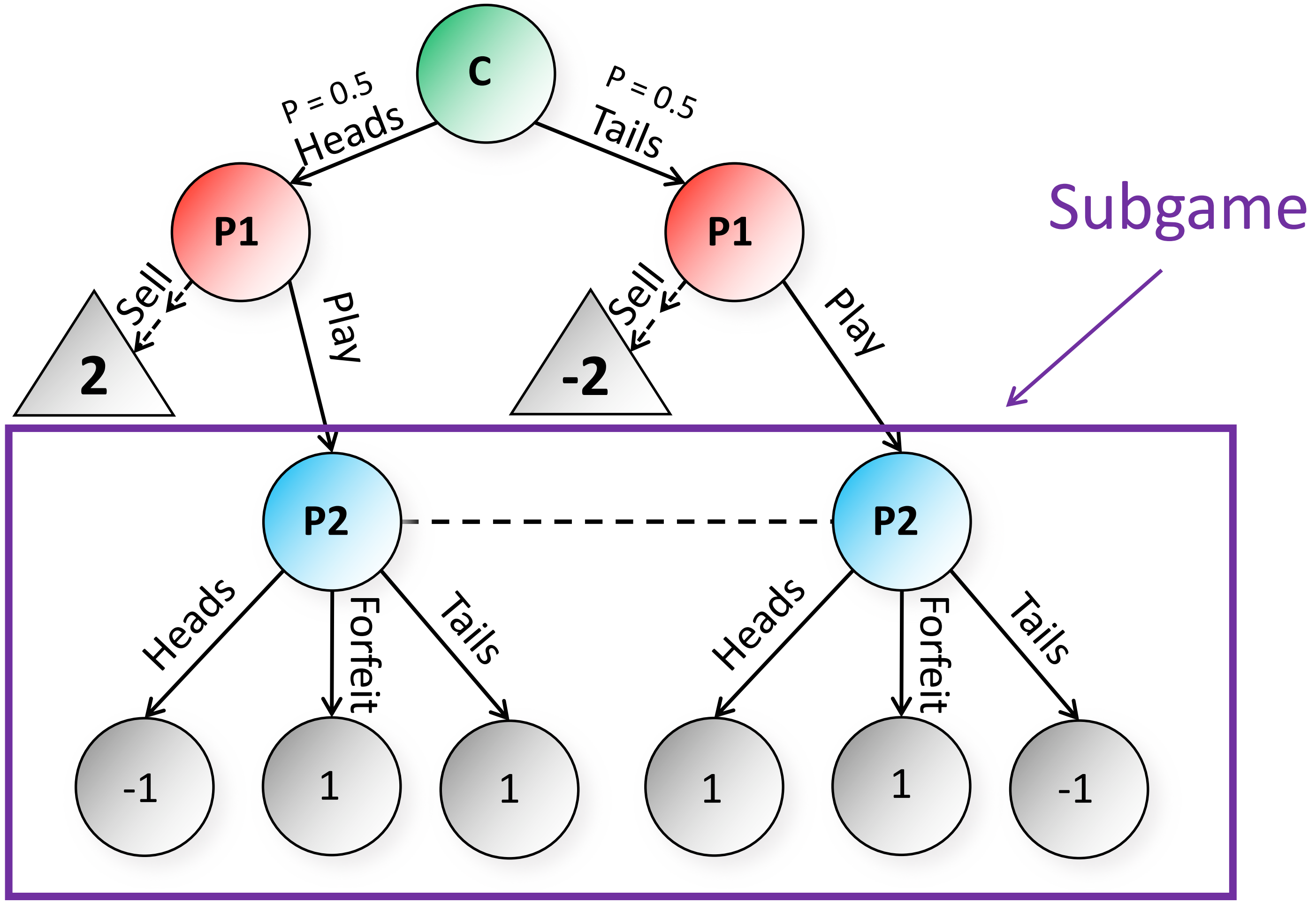
Slide credits: Tuomas Sandholm

Suppose P1 plays and it is P2’s turn



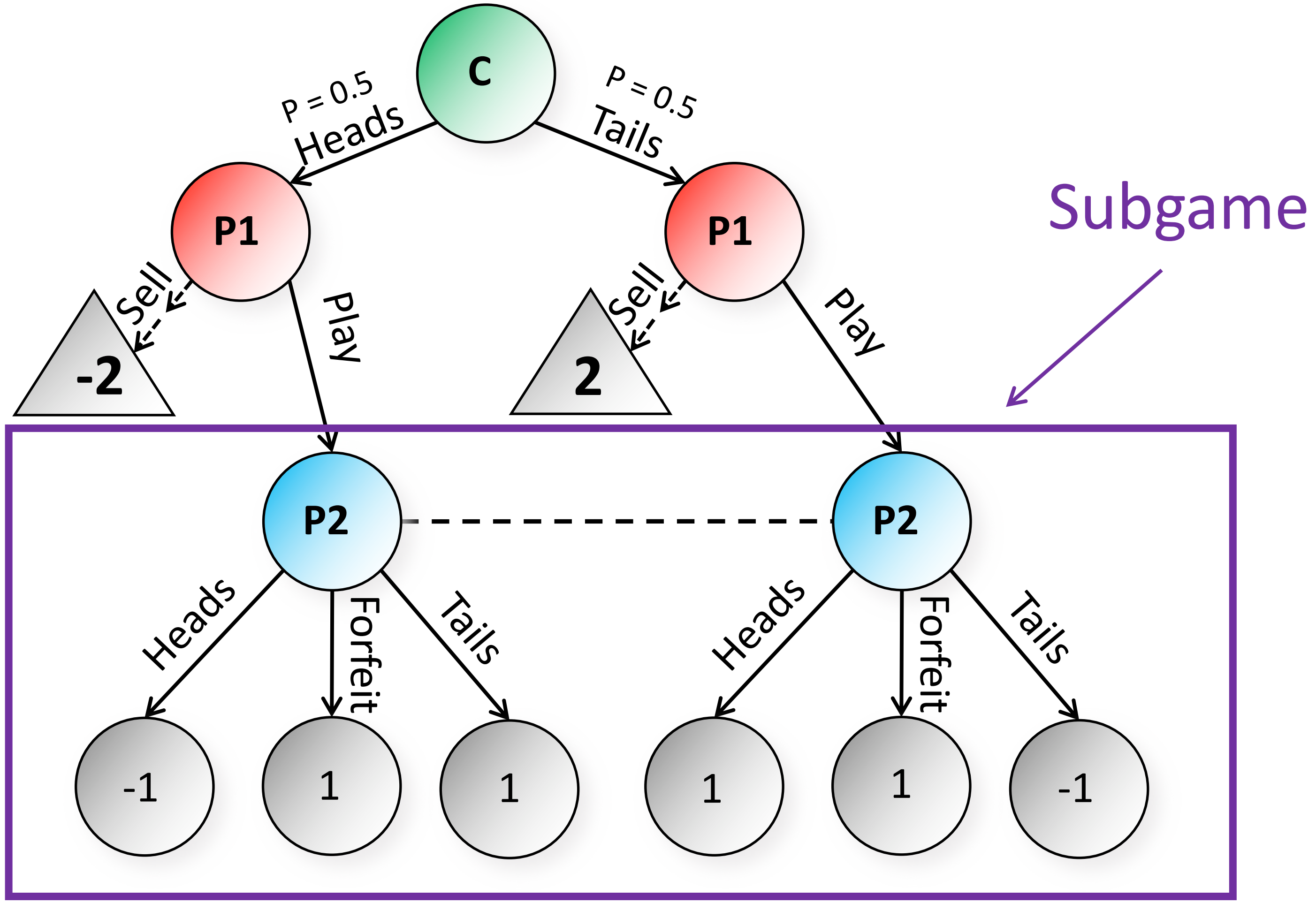
Slide credits: Tuomas Sandholm

First scenario: Heads sells for 2, Tails for -2



Slide credits: Tuomas Sandholm

But if the Sell payoffs are switched, then the optimal strategy in the subgame changes:

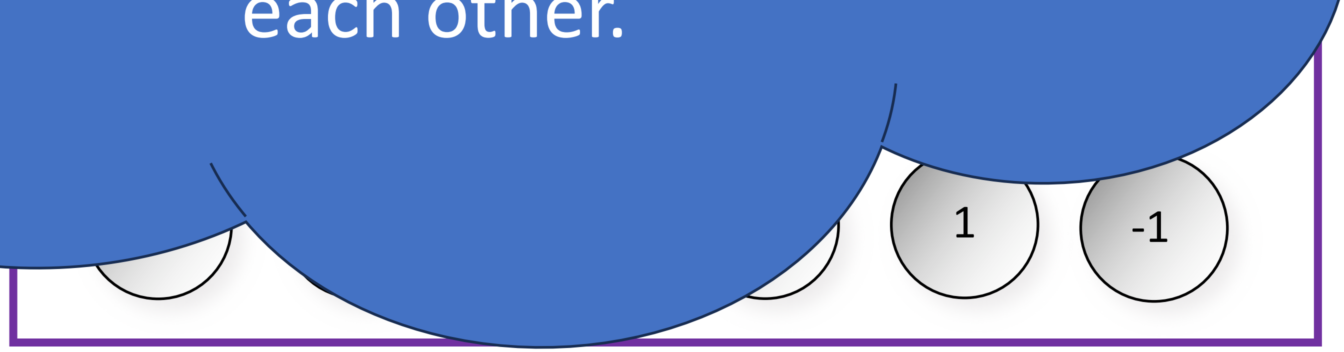


Slide credits: Tuomas Sandholm

But if the Sell … n the opti…

Conclusion: the optimal strategy in the subgame **depends on outcomes and strategies for situations that are not in the subgame, unlike perfect-information games**.

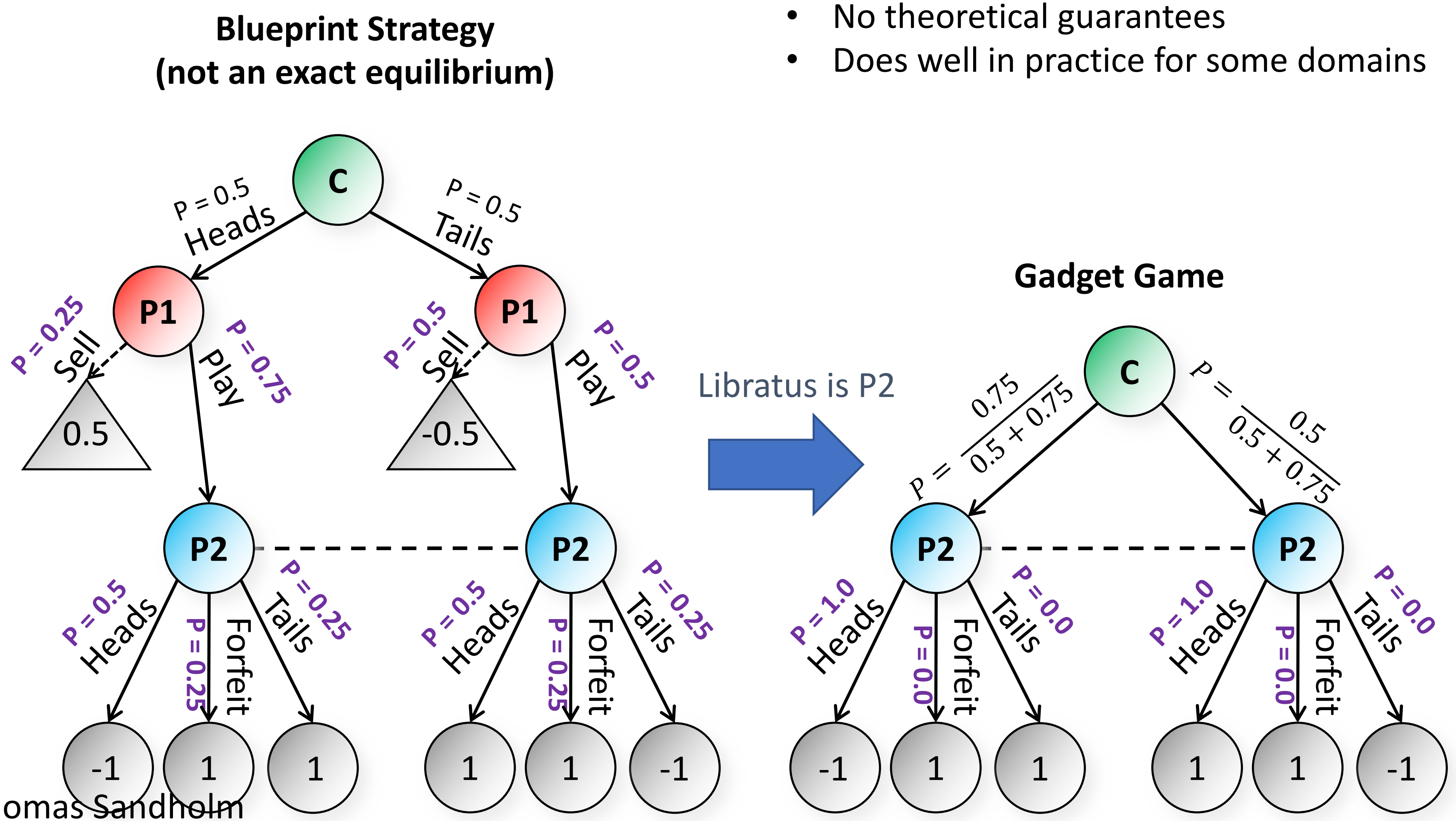
Two completely different parts of the game tree can affect the strategies of each other.



Unsafe subgame solving

[Ganzfried & Sandholm AAAMAS 2015]

- No theoretical guarantees
- Does well in practice for some domains

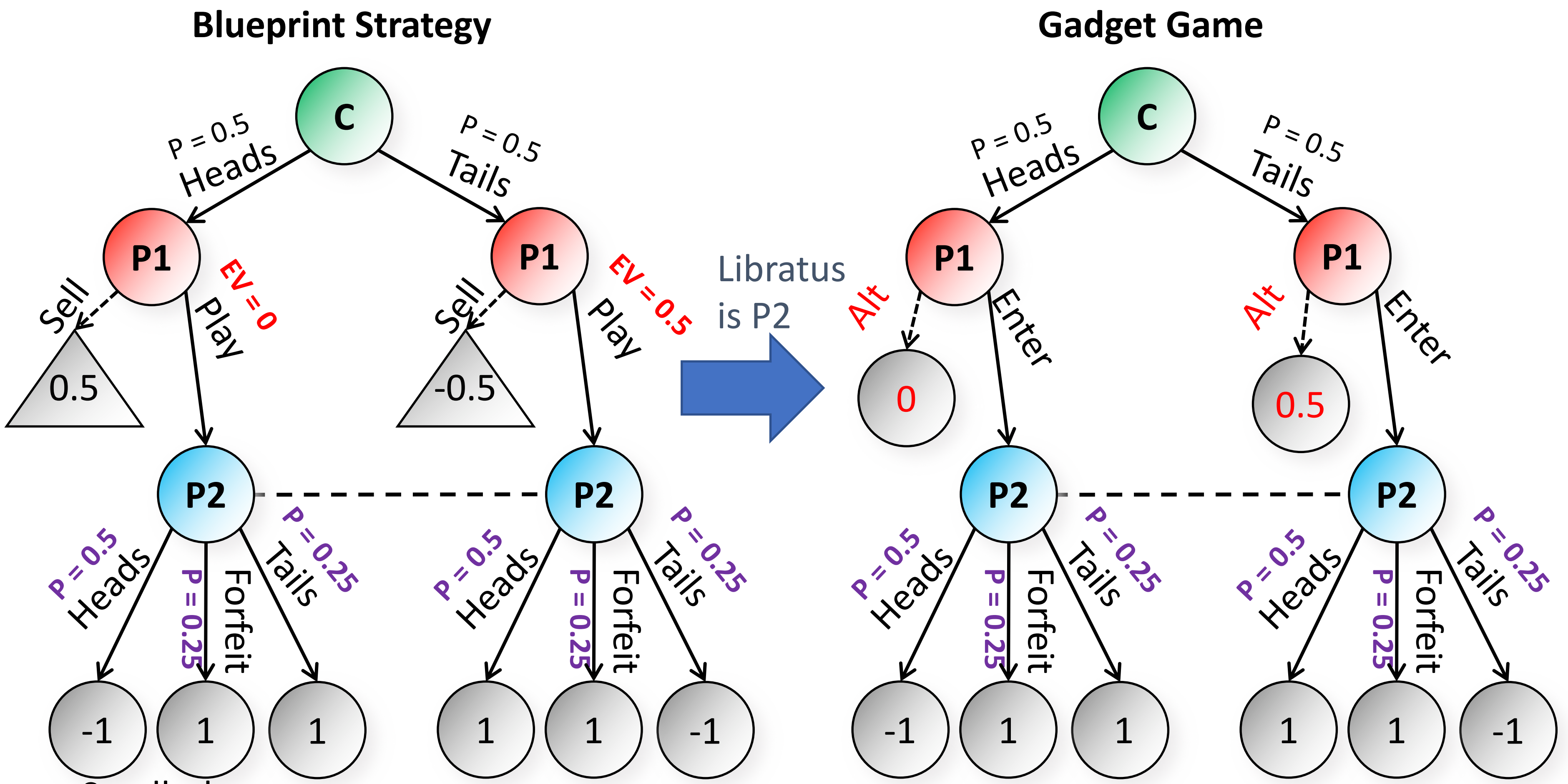


Slide credits: Tuomas Sandholm

Re-solve refinement

[Burch *et al*. AAAI 2014]

- P1 can choose between entering the subgame or taking the EV (according to the blueprint) of the subgame
- Makes sure opponent's EV for entering the subgame is no higher than in the blueprint strategy
  => Strategy provably no worse than blueprint strategy
- But may miss obvious opportunities for improvement (e.g., not forfeiting)



Slide credits: Tuomas Sandholm

Other decision-time planning techniques

- Much more involved techniques exist
- Decision-time planning in poker makes a big difference

	Exploitability
No decision-time planning	1465 mbb / hand
Nested Re-solve Refinement	**150.2 mbb / hand**
Nested Unsafe Refinement	**148.3 mbb / hand**
Nested Maxmargin Refinement	**122.0 mbb / hand**
Nested Reach-Maxmargin Refinement	**119.1 mbb / hand**

Libratus

- *Libratus* combined all these techniques against four of the ***best*** heads-up no-limit Texas Hold'em specialist pros



- 120,000 hands over 20 days in January 2017
- $200,000 divided among the pros based on performance
- Conservative experiment design

Slide credits: Tuomas Sandholm

Slide credits: Tuomas Sandholm

Slide credits: Tuomas Sandholm

Slide credits: Tuomas Sandholm

Systems structuring

- ***Bridges*** supercomputer
  - ~$17 million (including running it for its lifetime)
  - Architected by Hewlett Packard Enterprise (HPE) & Pittsburgh Supercomputing Center
  - Heterogeneous architecture
  - We used the part that has 800 HPE Apollo 2000 servers, each with 28 cores and 128GB RAM
  - We officially used ~24 million core hours for Libratus (Jan 2016-Jan 2017)
  - But we used only 14 of the 28 cores on each node because that was fastest
  - We were the biggest user of Bridges in that timeframe (used about half)
- Blueprint runs typically used 1 + 195 nodes
  - Typically ~1-8 weeks per run
- Each endgame solver used 50 nodes
  - Typically 30-60 seconds per run
- Each self-improver run used 196-600 nodes
  - Typically for 8-30 hours per run

- C++, Open-MP for parallelism within each server, MPI for distributed computing
- 2.6 PB disk storage

  Multiple strategies

  Snapshots (balance in snapshotting)

  Connections by Intel Omni-Path

  Intel Lustre file system

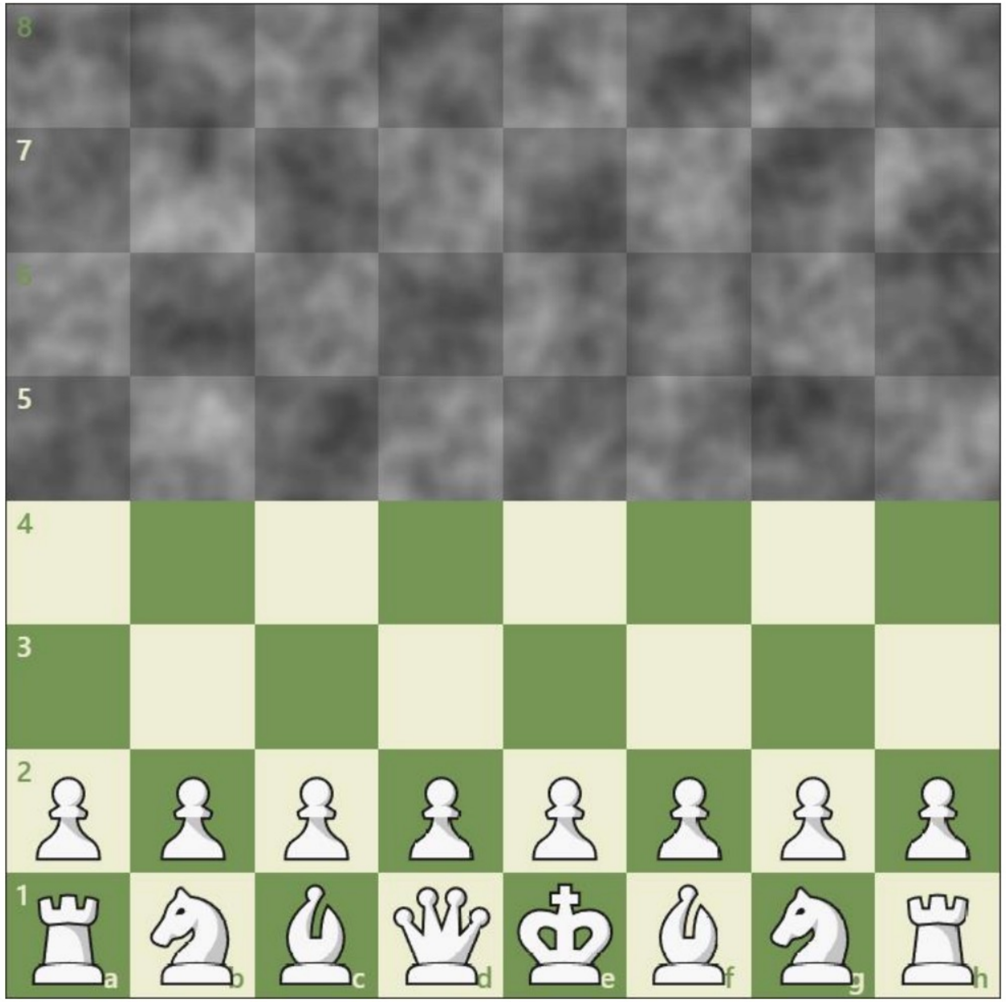
Slide credits: Tuomas Sandholm

Final result

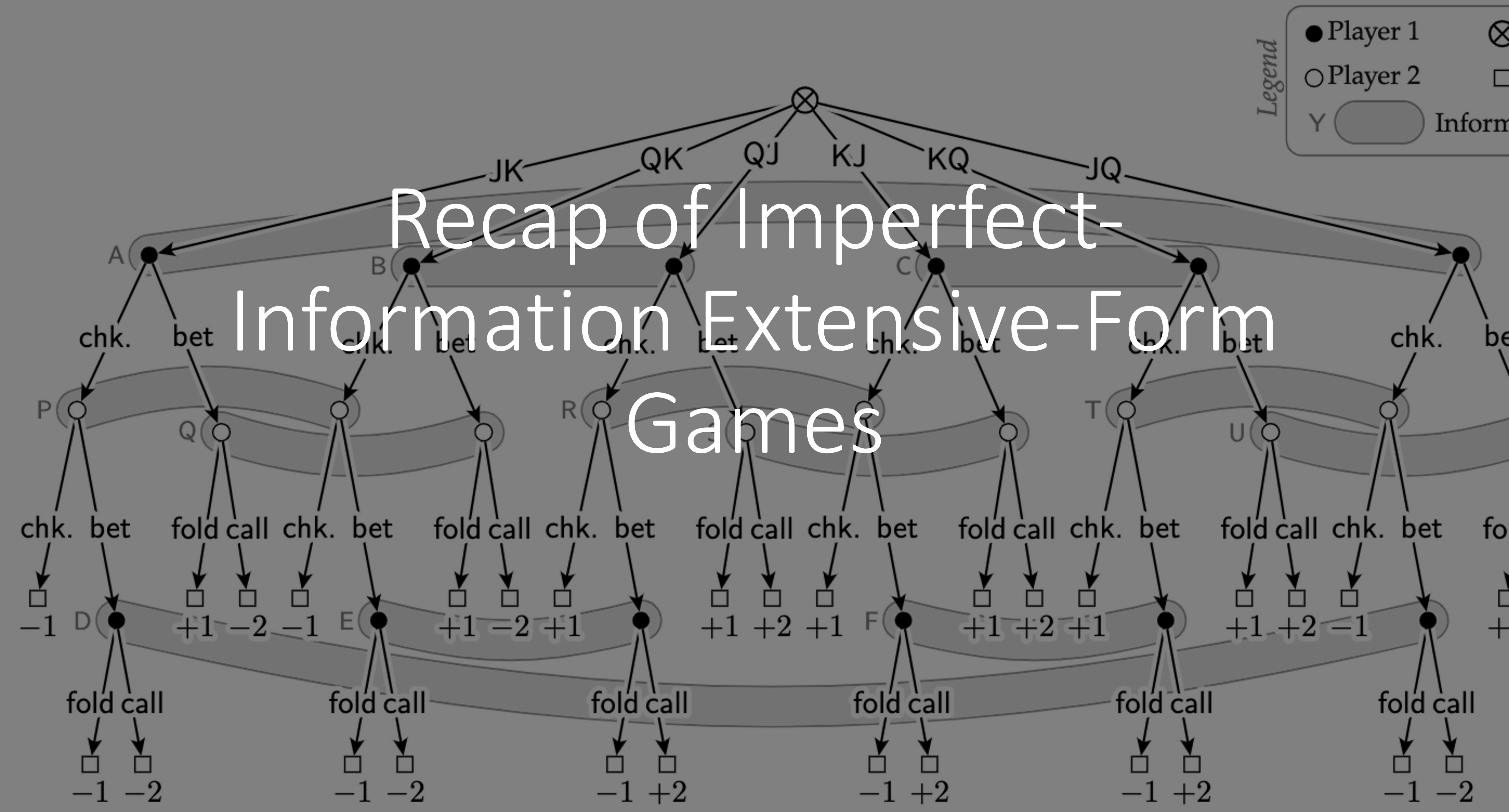
- Libratus beat the top humans in this game by a lot
  - 147 mbb/hand
  - Statistical significance 99.98%, i.e., 0.0002
  - Each human lost to Libratus

Remaining Challenges on DTP

- The amount of imperfect information in poker is relatively low
    - Only in the order of 1000s possible hands (two cards per player from a deck of 52)
- What about games with massive amounts of imperfect information?
    - Zhang and Sandholm, "Subgame solving without common knowledge", NeurIPS 2021
    - Liu, Fu, Fu, Yang, "Opponent-Limited Online Search for Imperfect Information Games", ICML 2023



Recap of Imperfect-Information Extensive-Form Games



The Important Messages

- Very flexible formalism for imperfect-information settings
- Many positive results
  - Convex structure (sequence-form)
  - Linear programming can be applied
  - Learning is possible
- Imperfect-information presents additional challenges
  - Combinatorial structure significantly more complicated than normal-form games
    - Exponentially many deterministic strategies as a function of edges in the tree
  - No clear notion of "subgame" or "endgame" -> no Markovian structure! Requires specialized care before RL-type techniques can be applied safely