Admin

HW1 and proposal feedback released

- Check your grades
- Generally, not regrading unless there are calculation errors

HW2 still being finalized

- Due last day of reading week

Project due last day of reading weeks

Lecture 10: Scaling up Part II

But where's the **machine learning**?

Not the **focus** of this class,
But we will **use ML to solve games**

Scaling up 2p0s game solvers

Games with perfect information

- A little bit of history
- AlphaGo, Muzero, AlphaZero

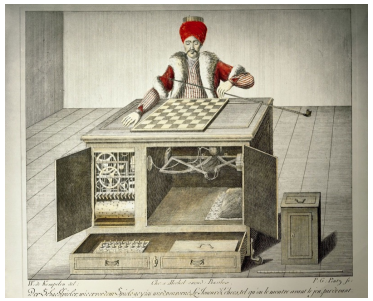
Games with imperfect information

- **Subgame solving/search [Focus of Today's Lecture]**

2p0s Perfect Information Games

2p0s Perfect Information Games

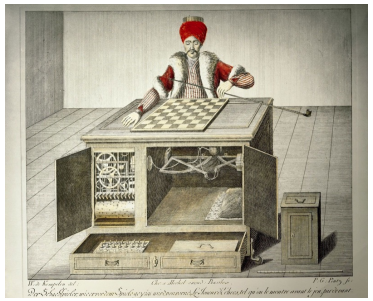
Mech Turk
[Chess]



~1770

2p0s Perfect Information Games

Mech Turk
[Chess]

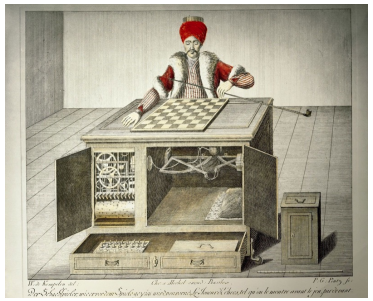


~1770

look, magic!

2p0s Perfect Information Games

Mech Turk
[Chess]



Chinook
[Checkers]

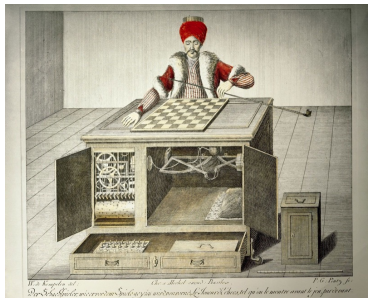
~1770

~1994

look, magic!

2p0s Perfect Information Games

Mech Turk
[Chess]



Chinook
[Checkers]

Deep Blue
[Chess]

~1770

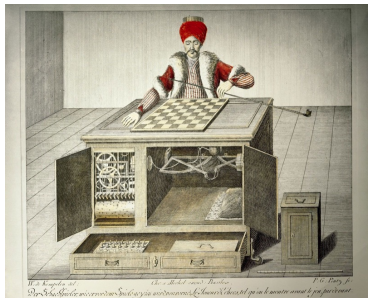
~1994

1996/7

look, magic!

2p0s Perfect Information Games

Mech Turk [Chess]



Chinook [Checkers]

Deep Blue [Chess]

~1770

~1994

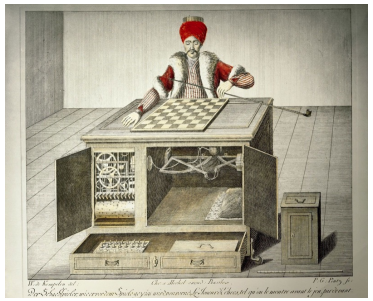
1996/7

look, magic!

Harnessing the power of compute

2p0s Perfect Information Games

Mech Turk [Chess]



Chinook [Checkers]

Deep Blue [Chess]

AlphaGo [Go]

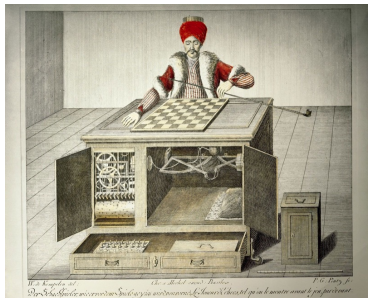
~1770 ~1994 1996/7 2016

look, magic!

Harnessing the power of compute

2p0s Perfect Information Games

Mech Turk [Chess]



Chinook [Checkers]

Deep Blue [Chess]

AlphaGo [Go]

~1770

~1994

1996/7

2016

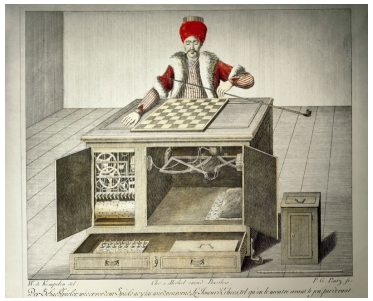
look, magic!

Harnessing the power of compute

Pattern recognition

2p0s Perfect Information Games

Mech Turk [Chess]

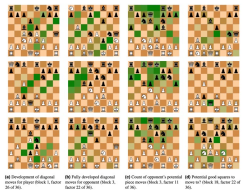


Chinook [Checkers]

Deep Blue [Chess]

AlphaGo [Go]

MuZero [many games]



~1770

~1994

1996/7

2016

2019

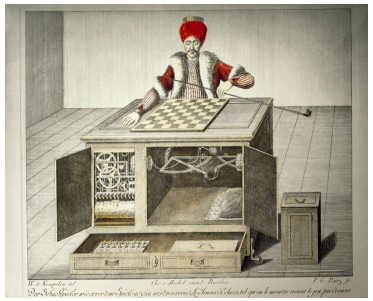
look, magic!

Harnessing the power of compute

Pattern recognition

2p0s Perfect Information Games

Mech Turk [Chess]

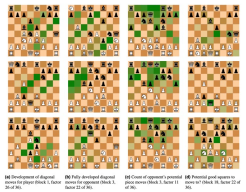


Chinook [Checkers]

Deep Blue [Chess]

AlphaGo [Go]

MuZero [many games]



~1770

look, magic!

~1994

Harnessing the power of compute

1996/7

2016

Pattern recognition

2019

General game-playing

What happens with huge game trees?

Full tree search requires time linear in _____ ?

What happens with huge game trees?

Full tree search requires time linear in _____ ?

- Exponential in the size of the depth of the tree

What happens with huge game trees?

Full tree search requires time linear in _____ ?

- Exponential in the size of the depth of the tree

Chess has a branching factor in the 20-30s, impossible to search for depths of more than 10+

- Chinese Chess, Shogi, Go

What happens with huge game trees?

Full tree search requires time linear in ____ ?

- Exponential in the size of the depth of the tree

Chess has a branching factor in the 20-30s, impossible to search for depths of more than 10+

- Chinese Chess, Shogi, Go

Rely on **evaluation function** $\tilde{V}(s)$

- At every state, tell me how "good" a state is from P1's perspective, assuming both players play optimally from here onwards
- $\tilde{V}(s)$ approximates $V(s)$ using some features of s
  - Material, Pawn structure, math

What happens with huge game trees?

Heuristics: be selective in depth/branching or early termination

- Quiescence search
- Null-moves
- Endgame tables
- …

What happens with huge game trees?

Heuristics: be selective in depth/branching or early termination

- Quiescence search
- Null-moves
- Endgame tables
- …

Tradeoff between “high-tech” evaluation functions and searching deeper

What happens with huge game trees?

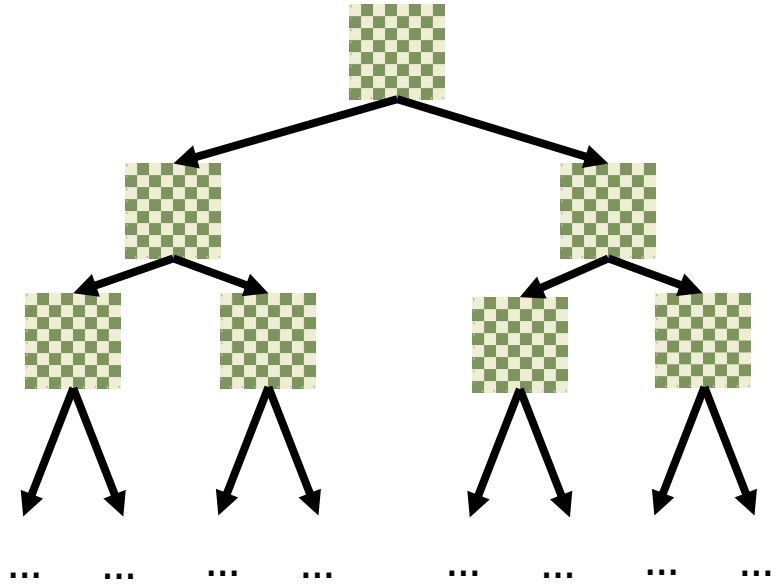
Heuristics: be selective in depth/branching or early termination

- Quiescence search
- Null-moves
- Endgame tables
- …

Tradeoff between “high-tech” evaluation functions and searching deeper

Monte Carlo Tree Search

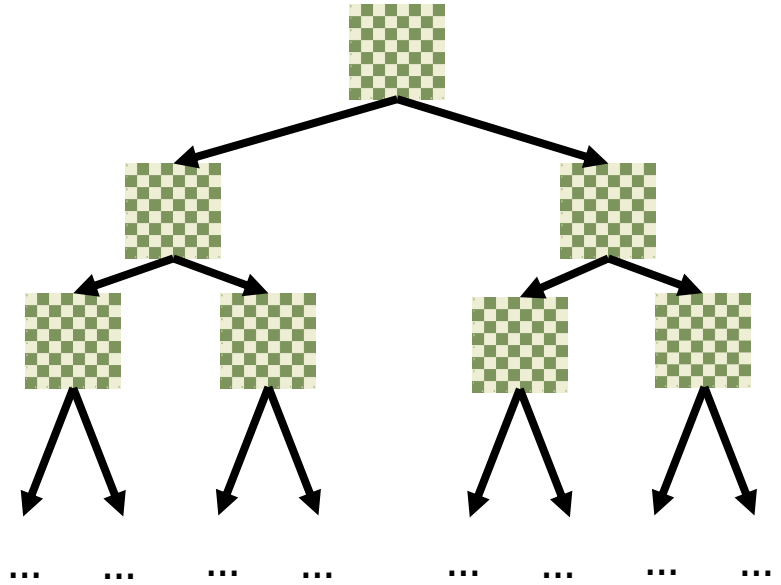
Game Solving with Function Approximation



Game Solving with Function Approximation

Consider zero-sum games

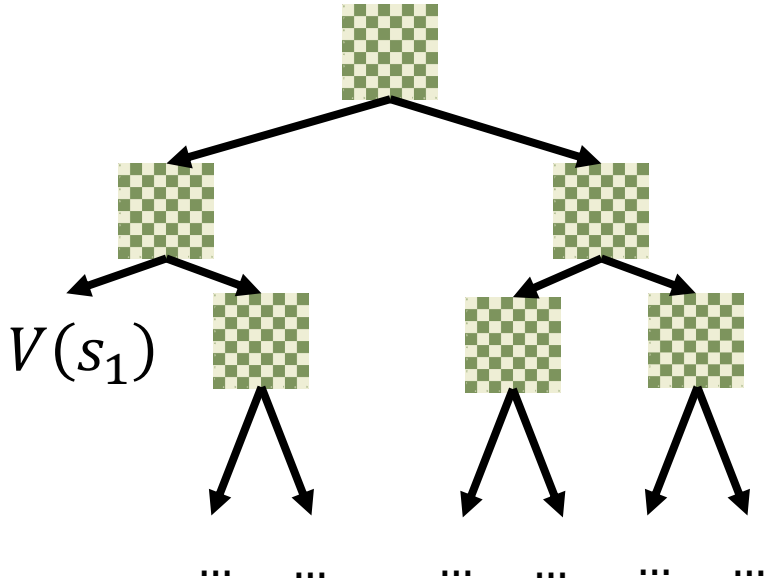
- Game tree too large to traverse explicitly
- **Value Function** $V(s)$ approximates how "good or bad" each state is



Game Solving with Function Approximation

Consider zero-sum games

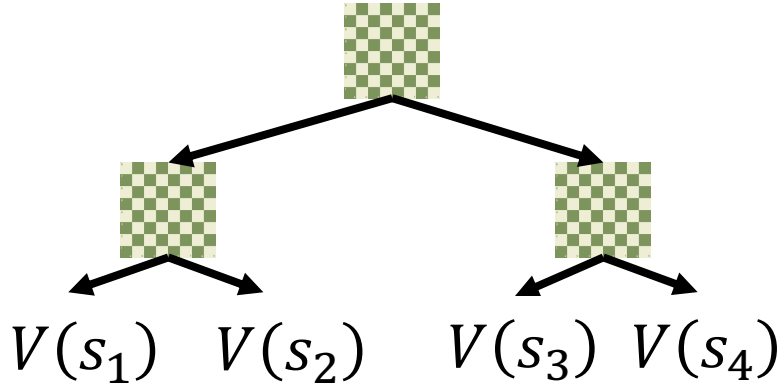
- Game tree too large to traverse explicitly
- **Value Function** $V(s)$ approximates how "good or bad" each state is



Game Solving with Function Approximation

Consider zero-sum games

- Game tree too large to traverse explicitly
- **Value Function** $V(s)$ approximates how "good or bad" each state is



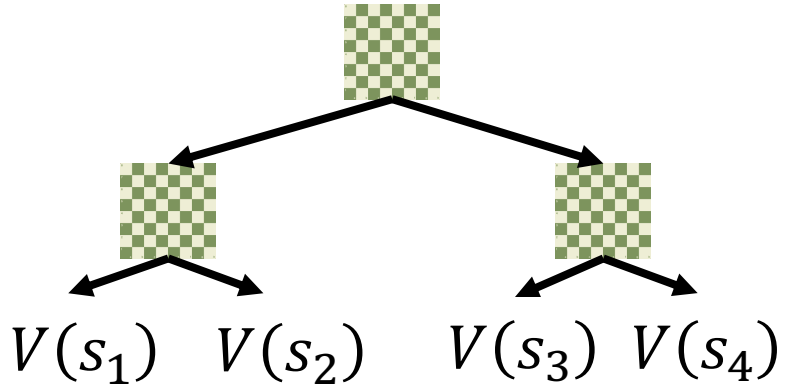
Game Solving with Function Approximation

Consider zero-sum games

- Game tree too large to traverse explicitly
- **Value Function** $V(s)$ approximates how "good or bad" each state is

Previously (e.g., Deep Blue)

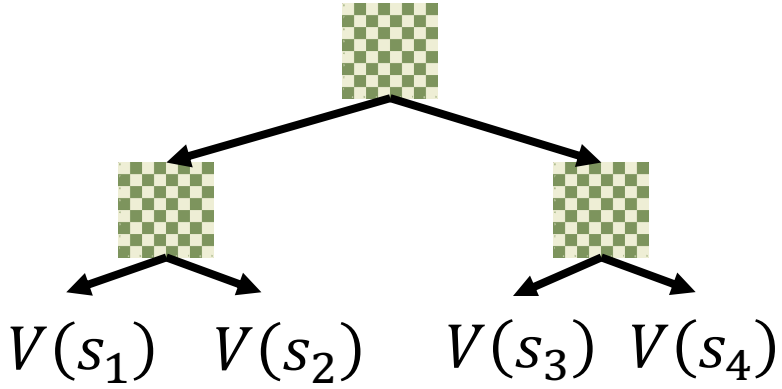
- Handcrafted evaluation function $V(s)$ based on heuristics from experts



Game Solving with Function Approximation

Consider zero-sum games

- Game tree too large to traverse explicitly
- **Value Function** $V(s)$ approximates how "good or bad" each state is



Previously (e.g., Deep Blue)

- Handcrafted evaluation function $V(s)$ based on heuristics from experts

Today (e.g., AlphaGo/Zero, DeepStack)

- **Learn** how good each state s is, generalize to states not seen before
- $V_\phi(s)$ is a network parameterized by ϕ approximating the value of s

Some observations

Relies on the idea of a value $V(s)$ or $Q(s, a)$ and their approximates

- Where did these come from, practically?
- Should we even be as accurate as possible?
  - Example: V(s) for chess is technically restricted to be in {-1, 0, 1}, yet in practice we still evaluate positions in terms of “centipawns”. Same goes for most win/loss games

Some observations

Relies on the idea of a value $V(s)$ or $Q(s,a)$ and their approximates

- Where did these come from, practically?
- Should we even be as accurate as possible?
  - Example: V(s) for chess is technically restricted to be in {-1, 0, 1}, yet in practice we still evaluate positions in terms of "centipawns". Same goes for most win/loss games

Warning: hard to be true in practice

What happens if value function is "slightly off?", i.e., $\left|\tilde{V}(s) - V(s)\right| < \epsilon$?

- Will a deeper search always yield a better immediate action at the root?
  - No (see Nau 1982, Wilson et. al., 2018) but very rare in practice
  - Looking ahead deeper is probably the right thing to do
- Are there any other guarantees?
  - Yes, though a lot weaker, we'll see that later

Subgame Decomposition

What about imperfect information EFGs?

Let's start with the simplest task

Can we even **store** or summarize "values" in subgames?

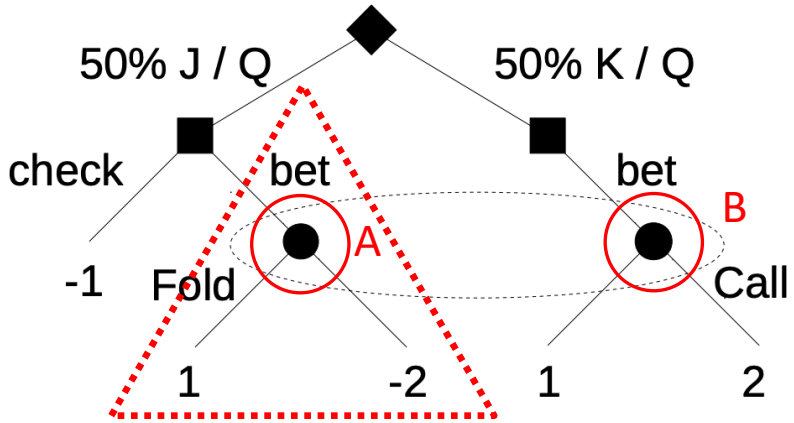
Attempt 1: Replacing w/ subtree value

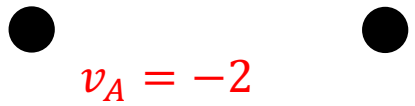
Consider subtree rooted at A (or B)

- Take EFG formed by the sub-EFG G' rooted at A (or B)
- If s, s' in same infoset in G and are both in G', then they are the same infoset in G' (and vice versa)
- Solve G' to get value of vertex v

Replace payoffs at by v_A (and v_B)

Does this work?



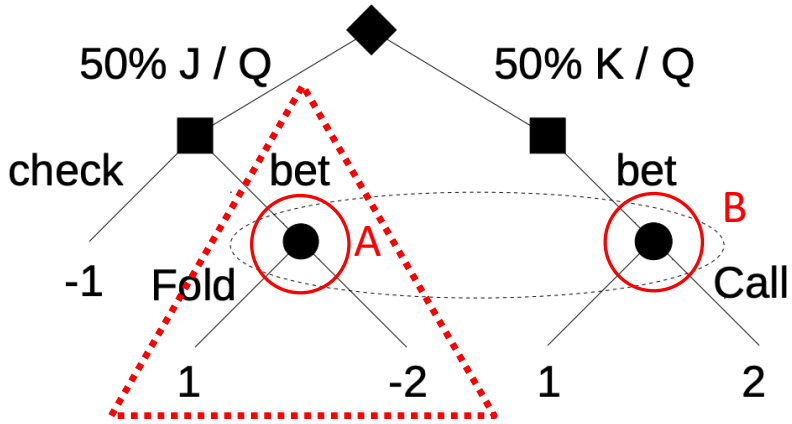


Attempt 1: Replacing w/ subtree value

Consider subtree rooted at A (or B)

- Take EFG formed by the sub-EFG G' rooted at A (or B)
- If s, s' in same infoset in G and are both in G', then they are the same infoset in G' (and vice versa)
- Solve G' to get value of vertex v

Replace payoffs at by v_A (and v_B)

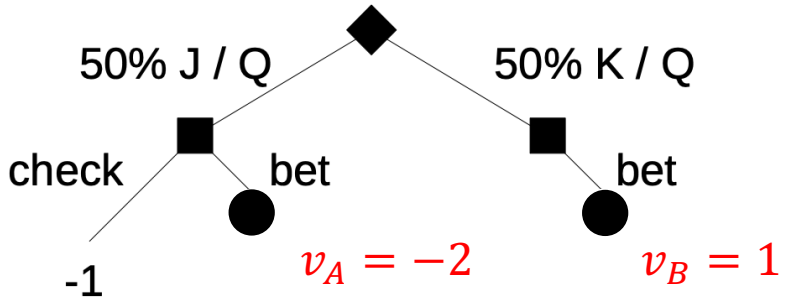


Does this work?

Recall true Nash has value 1/3

No. New value under "truncated" game is 0

- Information set at the root is not respected
- In this instance, P1 is "respecting" P2 too much



Attempt 2: Replace w/ "Nash node value"

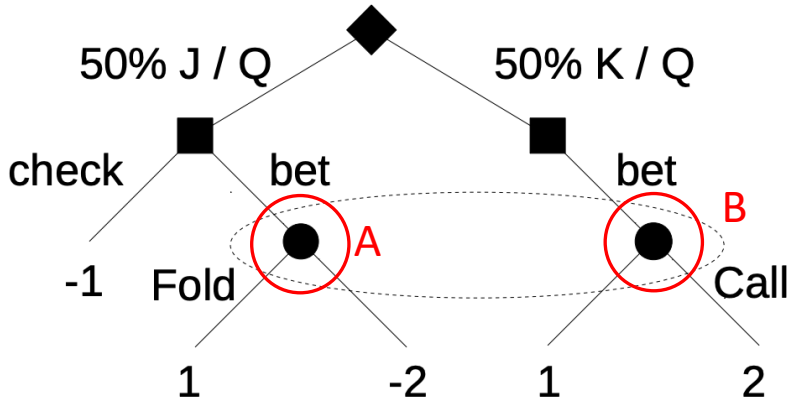
P2's strategy (at both A, B) is

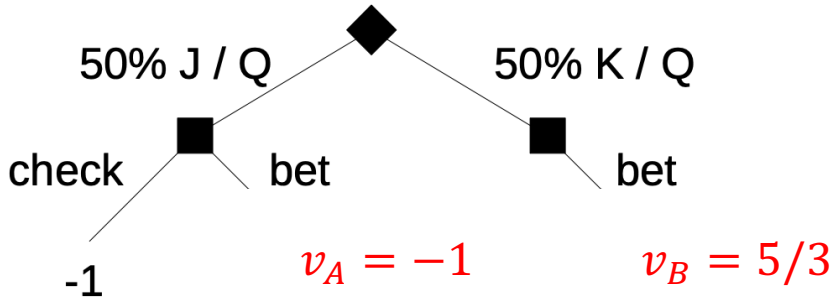
- Fold w.p. $1/3$, Call w.p. $2/3$ (make sure you know why!)
- "Value" at A is $1/3 - 4/3 = -1$

By this logic, P1 can do *anything* if it got J

- Value calculated here is $2/3$
- This is **higher** than in the true game!

What went wrong here?





Attempt 2: Replace w/ "Nash node value"

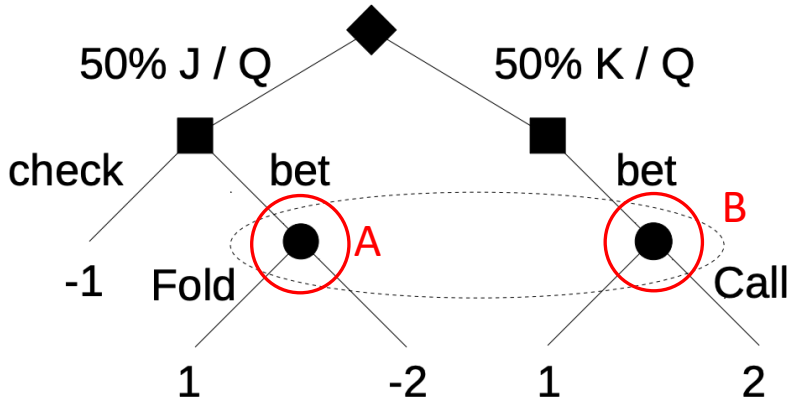
P2's strategy (at both A, B) is

- Fold w.p. $1/3$, Call w.p. $2/3$ (make sure you know why!)
- "Value" at A is $1/3 - 4/3 = -1$

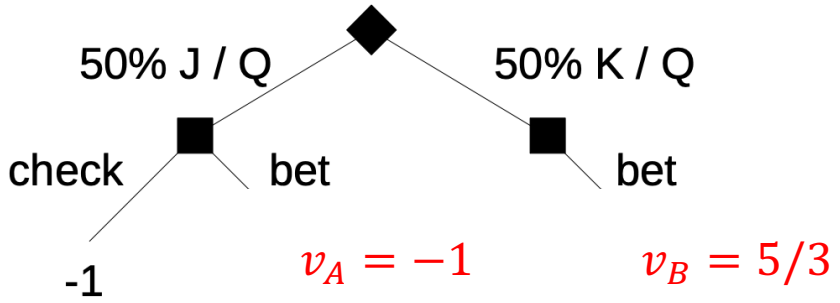
By this logic, P1 can do *anything* if it got J

- Value calculated here is $2/3$
- This is **higher** than in the true game!

What went wrong here?



We assumed that P2 continues playing its NE



Important reminder!

That a strategy does well against the opponent's NE is **not** a sign that your strategy is optimal, or even good

- Holds even for normal-form games
- Need to consider **exploitability**

Important reminder!

That a strategy does well against the opponent's NE is **not** a sign that your strategy is optimal, or even good

- Holds even for normal-form games
- Need to consider **exploitability**

Example: RPS

- Any strategy does "well" against the opponent playing uniformly
- But 100% Rock is bad in terms of exploitability

Important reminder!

That a strategy does well against the opponent's NE is **not** a sign that your strategy is optimal, or even good

- Holds even for normal-form games
- Need to consider **exploitability**

Example: RPS

- Any strategy does "well" against the opponent playing uniformly
- But 100% Rock is bad in terms of exploitability

Very, very, very common mistake that I see in papers all the time

- "My agent's performance is good against an opponent that is not exploiting me"
- "My agent's performance is good against an opponent that … I myself trained"

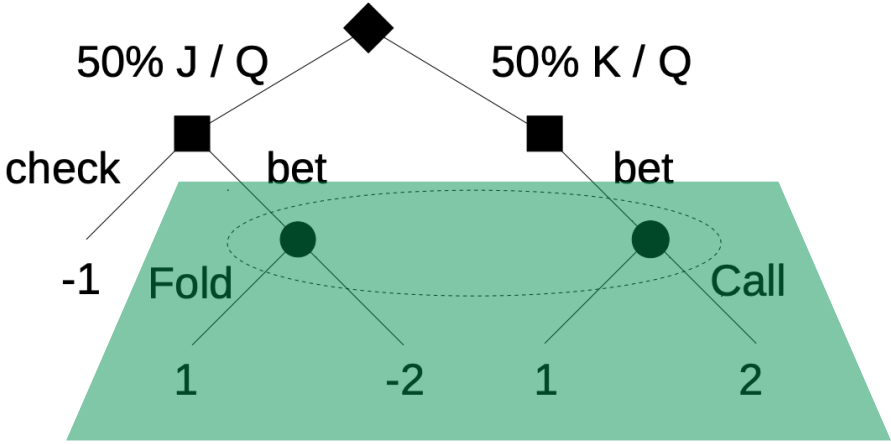
<u>What makes subgame solving in EFG tricky</u>

Need to deal with both imperfect information **and** the fact that there is another player is taking advantage of this!

Insight #1: Subtrees are not enough

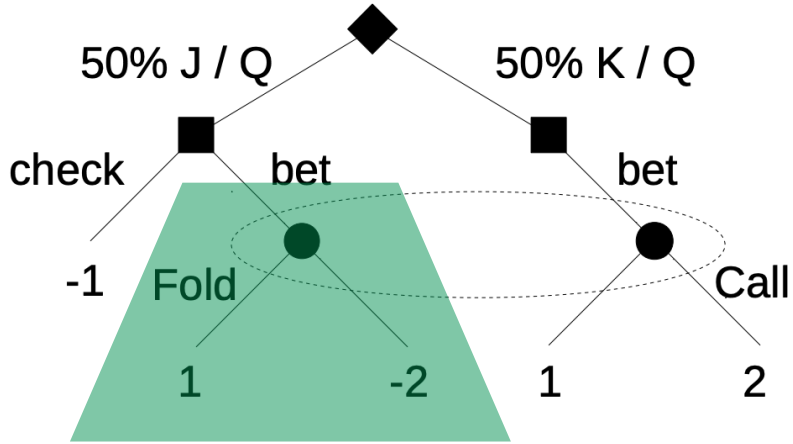
We need to look at **subgames**

- Closed under "in-same-infoset" relationship
- Collection of subtrees (forest) such that
  - all descendants are included AND
  - If s, s' are in same infoset, and s is in subgame, then s' is also in same subgame
- Intuitively: subgame cannot have infosets "poking out"



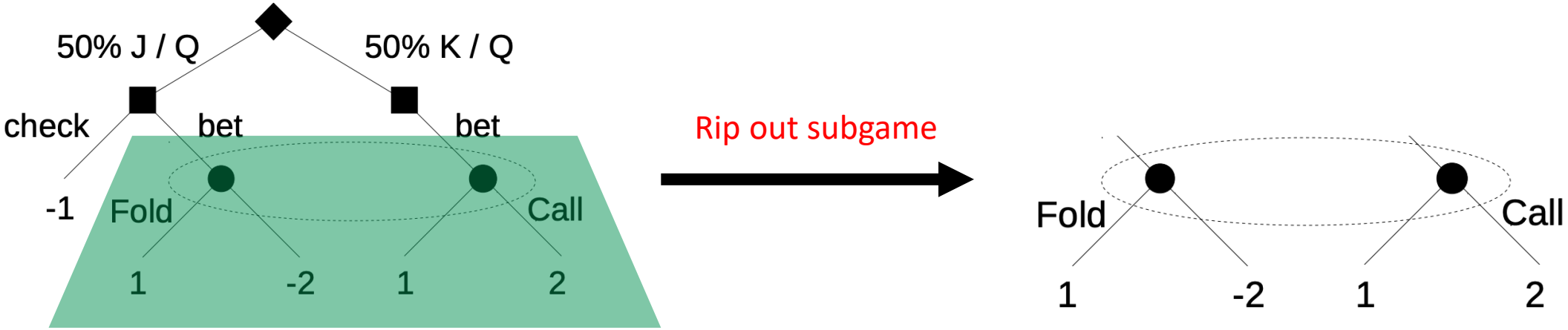
Subgame now "respects" infosets

- But not yet enough --- we can't solve a subgame "independently" from the rest of the game!

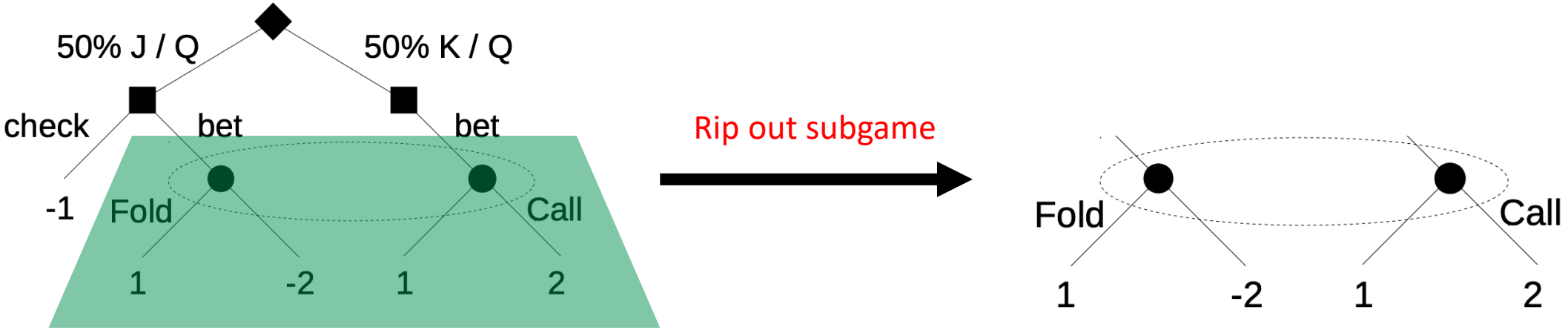


Not allowed!

Can we “solve” subgames alone?

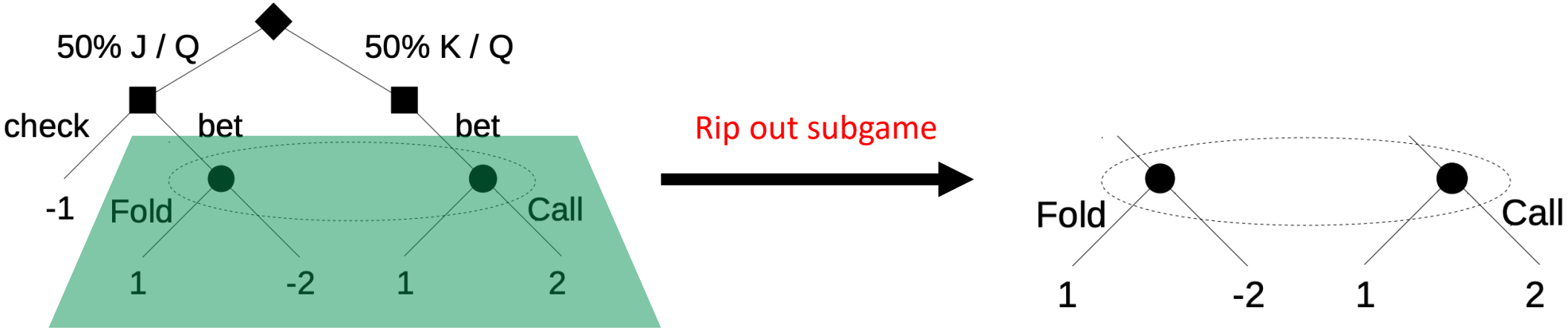


Can we “solve” subgames alone?



No. Why?

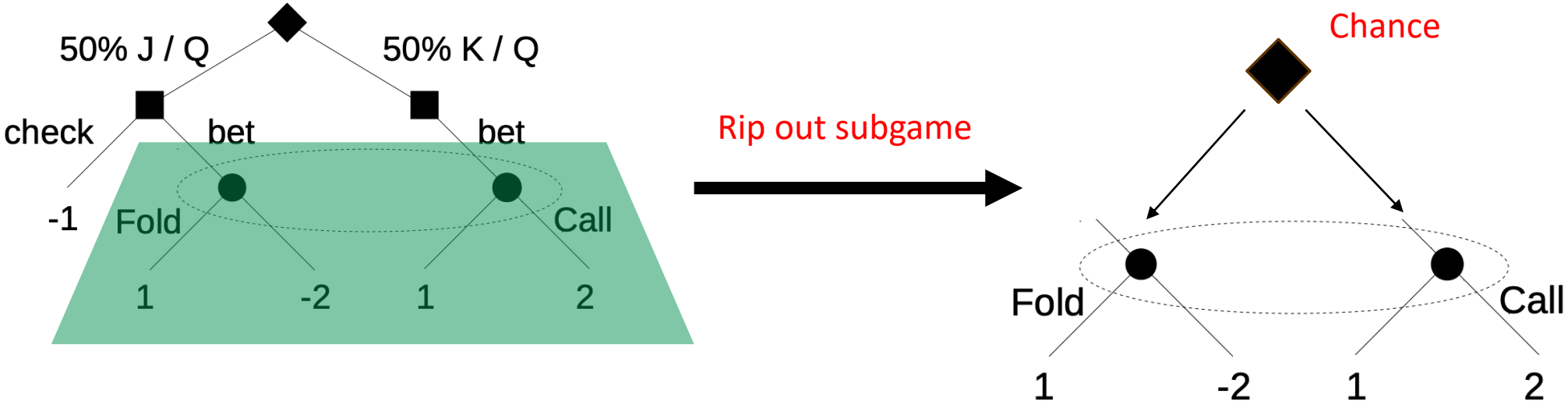
Can we “solve” subgames alone?



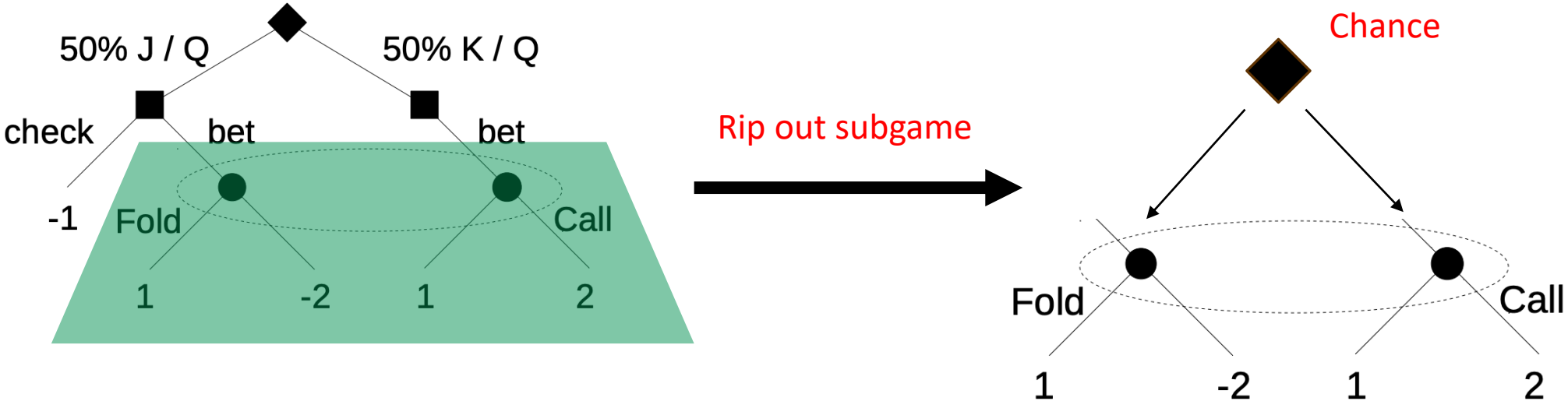
No. Why?

Where is the root?

Adding in chance nodes

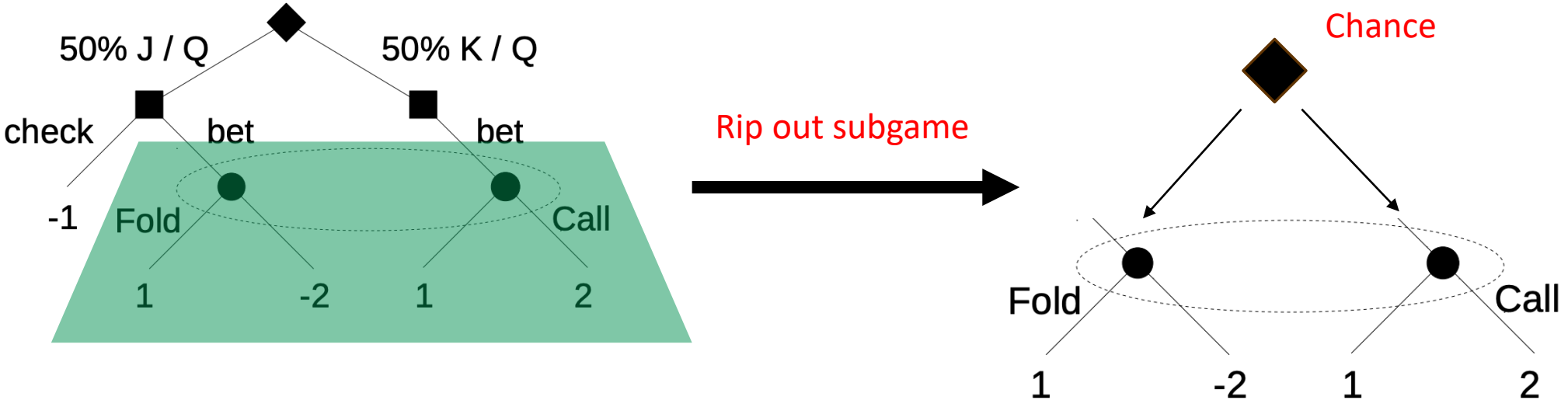


Adding in chance nodes



Not yet. Why?

Adding in chance nodes

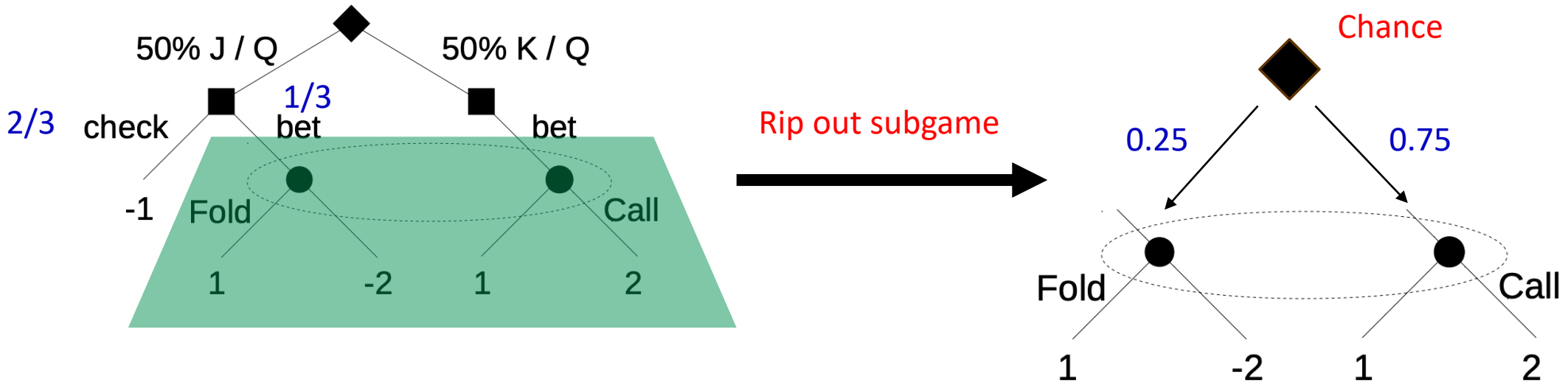


Not yet. Why?

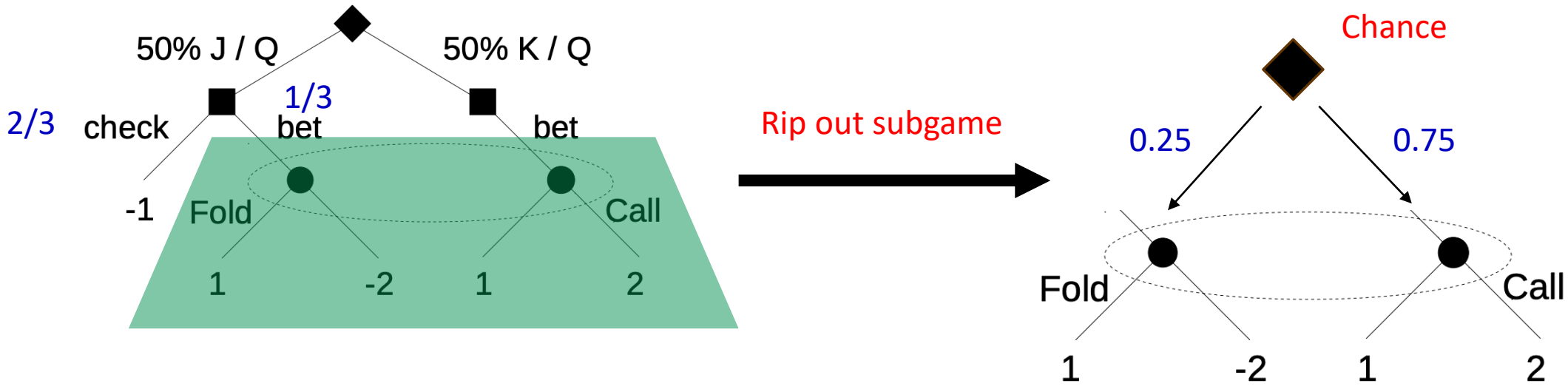
What are the chance probabilities?

- The probabilities dramatically affect solution of subgame!
- Do we use the probabilities of reaching **under Nash?**
- Or…?

Let's try using NE as probabilities



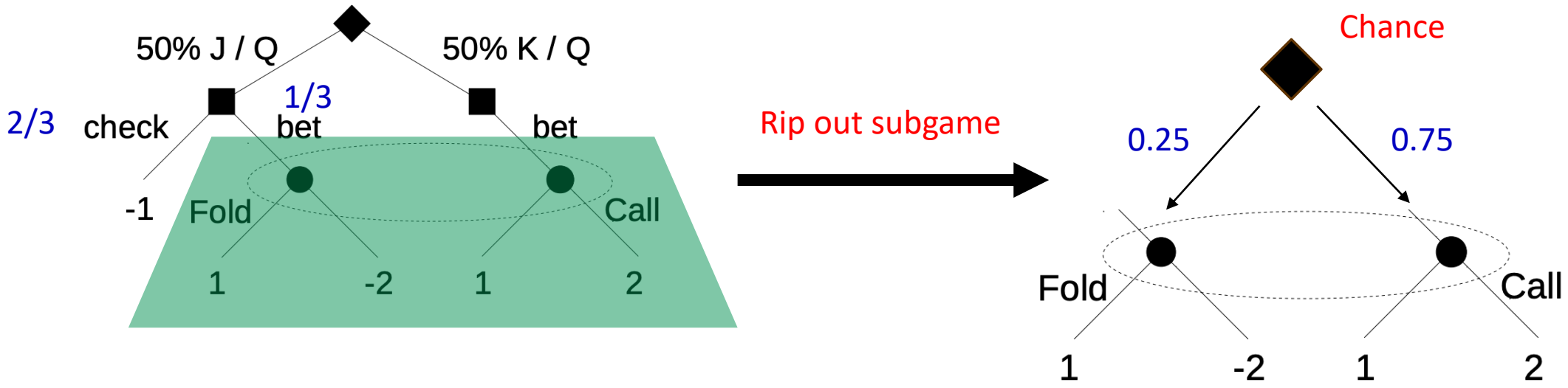
Let's try using NE as probabilities



P2 is now indifferent between folding as calling (as expected)

- But... What can the solution of this game useful for? Not much…
- **Any strategy** of P2 in subgame is a Nash!

Let's try using NE as probabilities



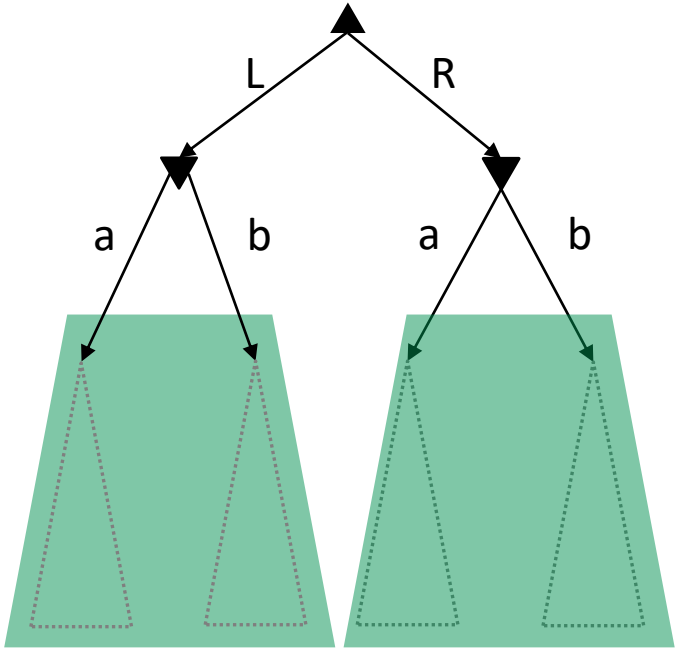
P2 is now indifferent between folding as calling (as expected)

- But... What can the solution of this game useful for? Not much…
- **Any strategy** of P2 in subgame is a Nash!

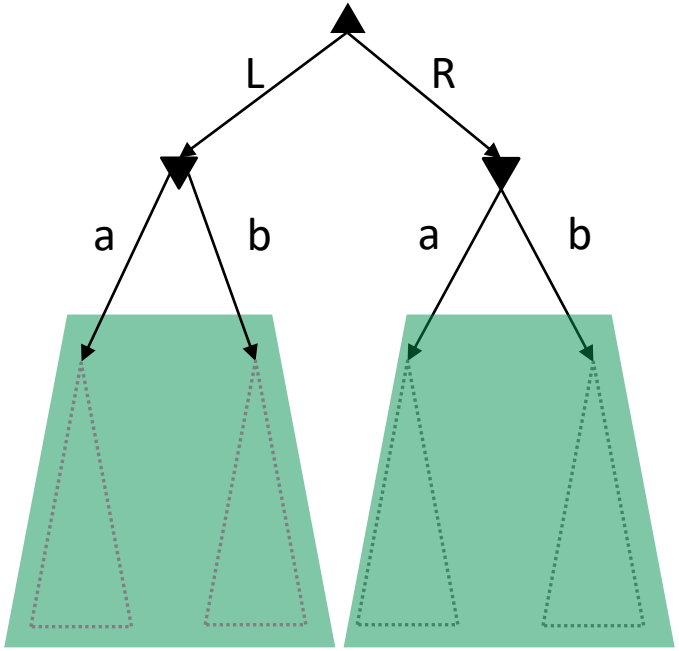
Still a useful idea to keep in mind!

Subgame Solving

Framework for Subgame solving

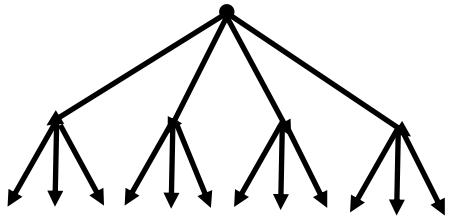


Framework for Subgame solving

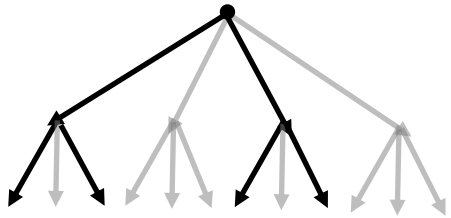


Blueprint

- Can be any baseline strategy
- Typically, the `best' strategy which can be computed offline based on some small, abstracted variant of game

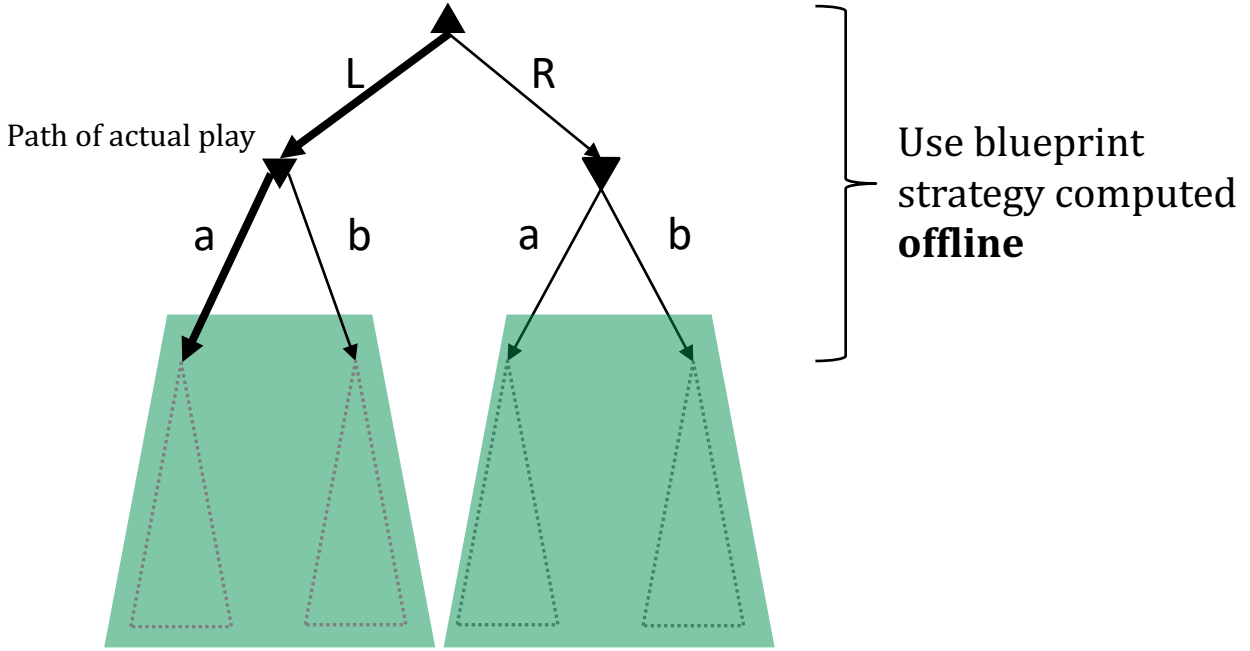


Full Game



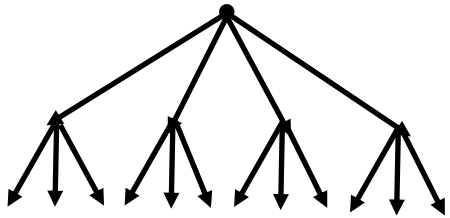
Abstraction to compute blueprint

Framework for Subgame solving

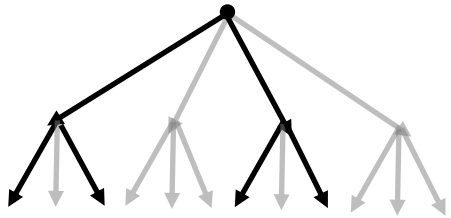


Blueprint

- Can be any baseline strategy
- Typically, the `best' strategy which can be computed offline based on some small, abstracted variant of game

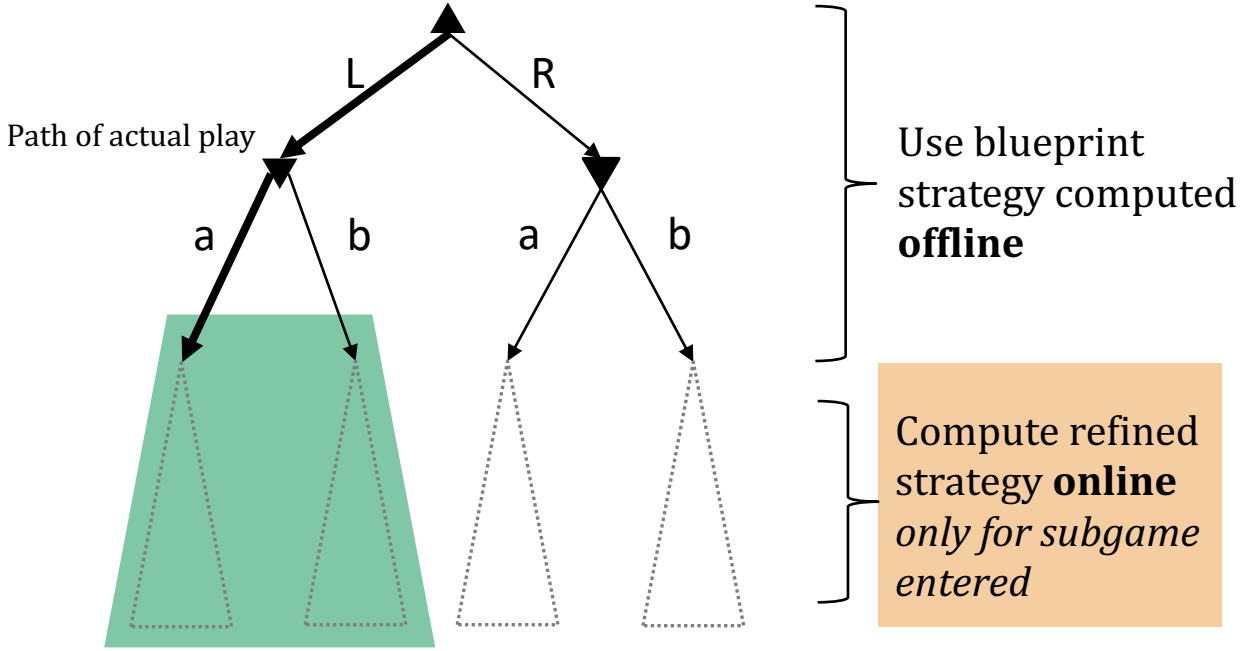


Full Game



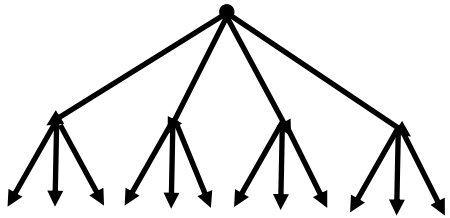
Abstraction to compute blueprint

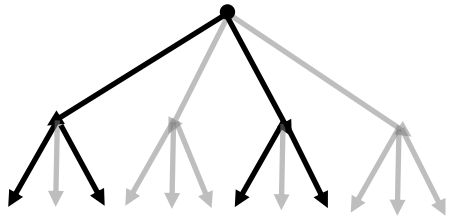
Framework for Subgame solving



Blueprint

- Can be any baseline strategy
- Typically, the `best' strategy which can be computed offline based on some small, abstracted variant of game

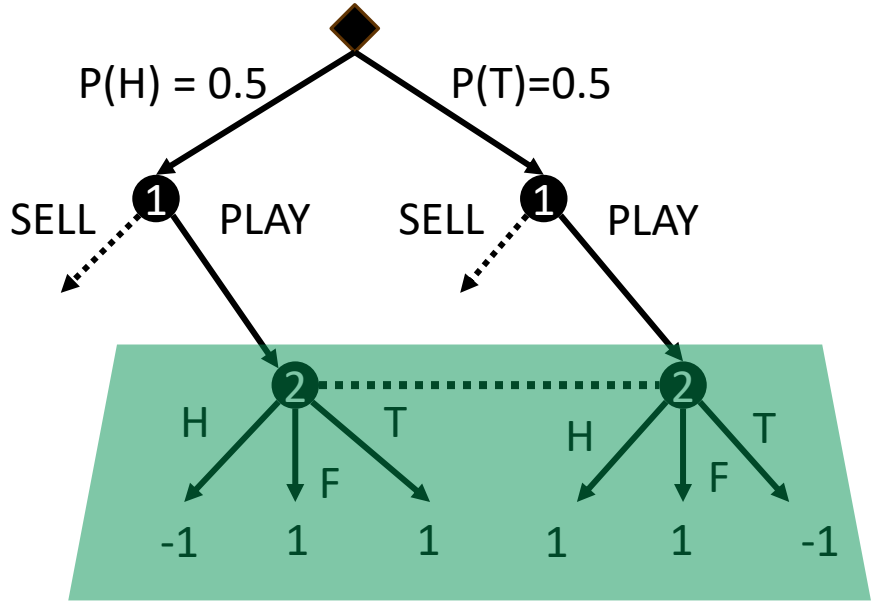

Full Game


Abstraction to compute blueprint

Refinement (of a blueprint)

- An attempt at improving P1's strategy in a subgame from the blueprint
- Followed for remainder of game

Example: Coin Toss

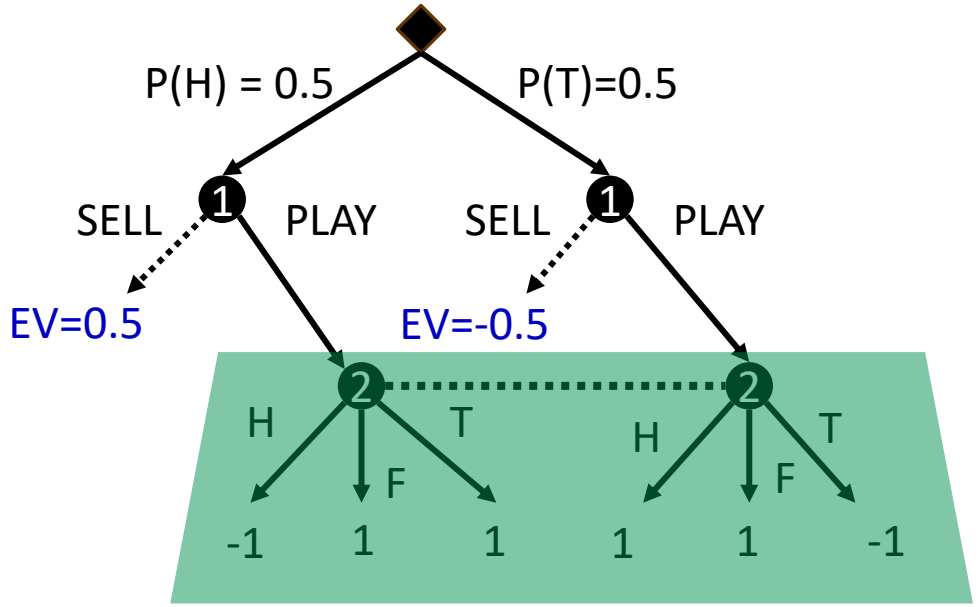


H=Heads
T=Tails
F=Forfeit

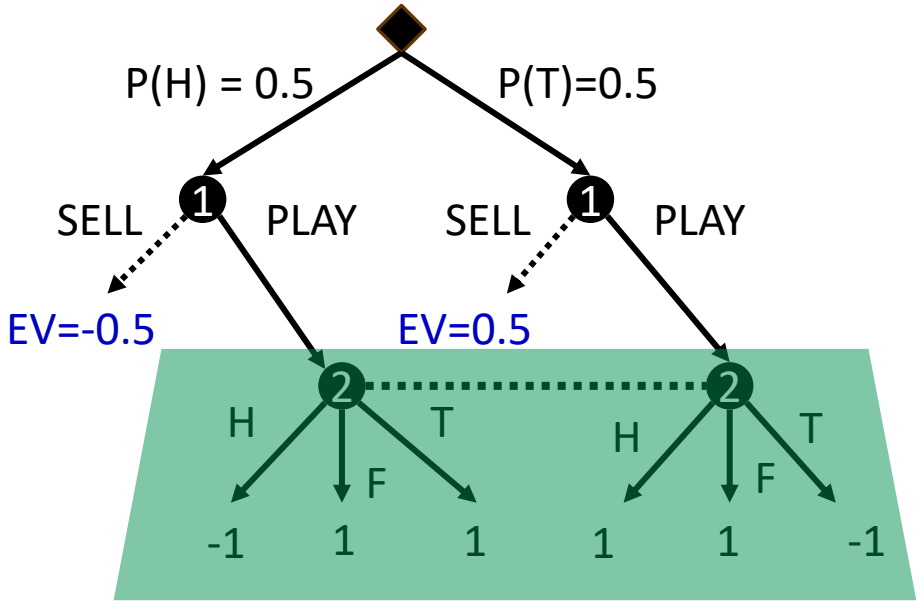
Rewards for for P1
NOTE: P2 is doing the subgame solving here (to follow the paper's convention)

Subgame cannot be solved independently

H=Heads
T=Tails
F=Forfeit

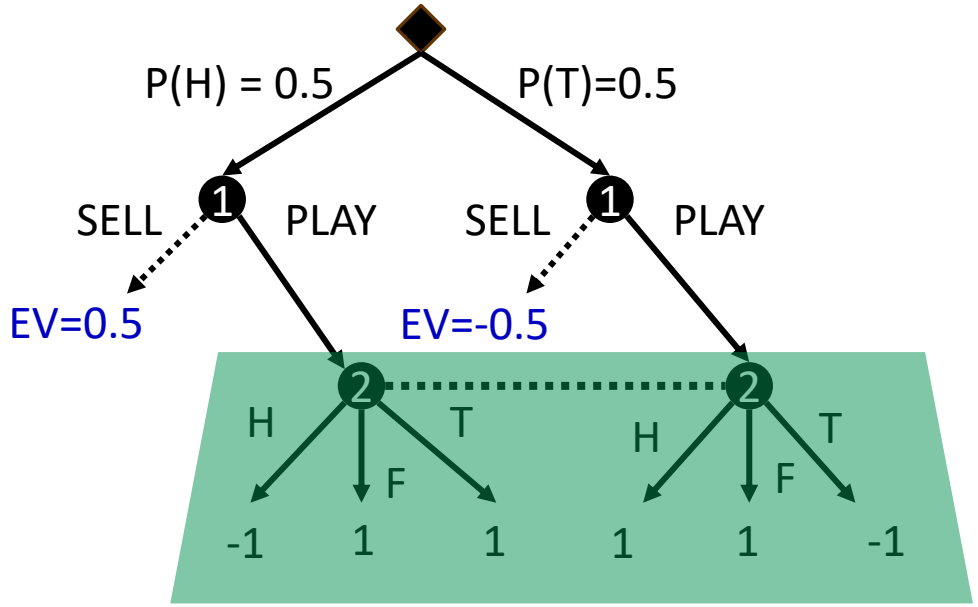


Game 1



Game 2

Subgame cannot be solved independently II

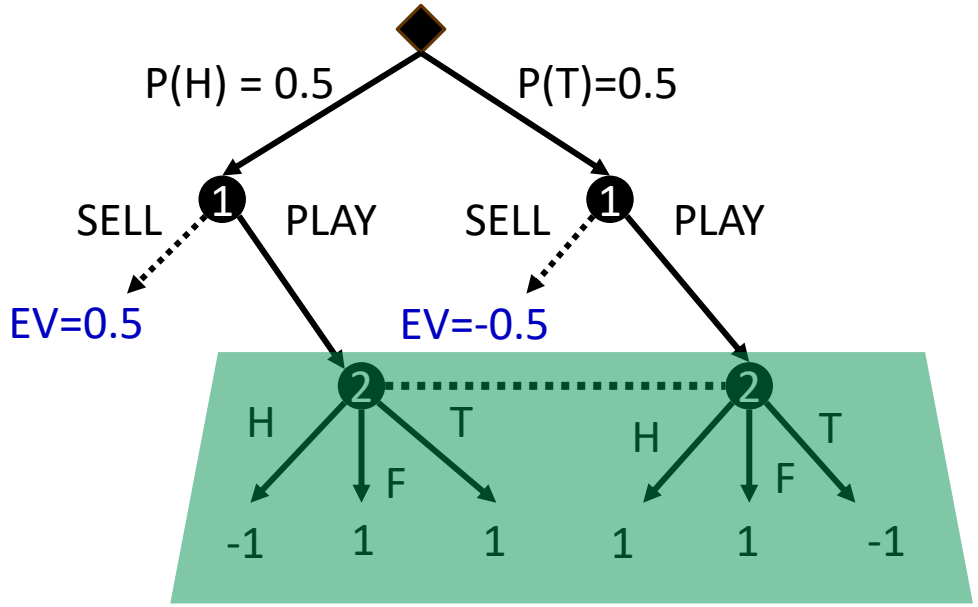


Game 1

H=Heads
T=Tails
F=Forfeit

Subgame cannot be solved independently II

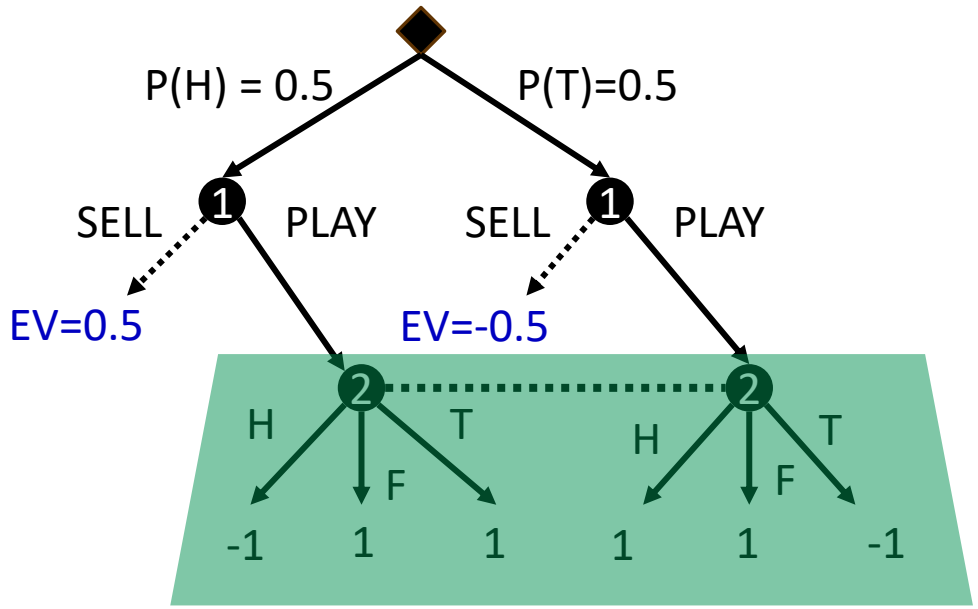
What is the NE of this game?



H=Heads
T=Tails
F=Forfeit

Game 1

Subgame cannot be solved independently II



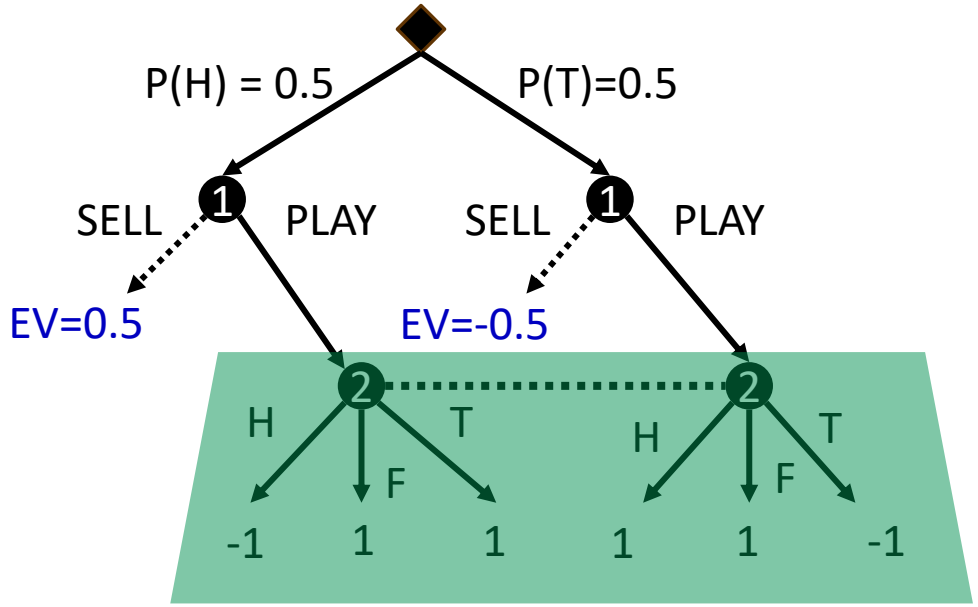
H=Heads
T=Tails
F=Forfeit

Game 1

What is the NE of this game?

- If P2 plays heads, P1's BR will sell if coin is heads, and play if tails →0.75 on avg

Subgame cannot be solved independently II



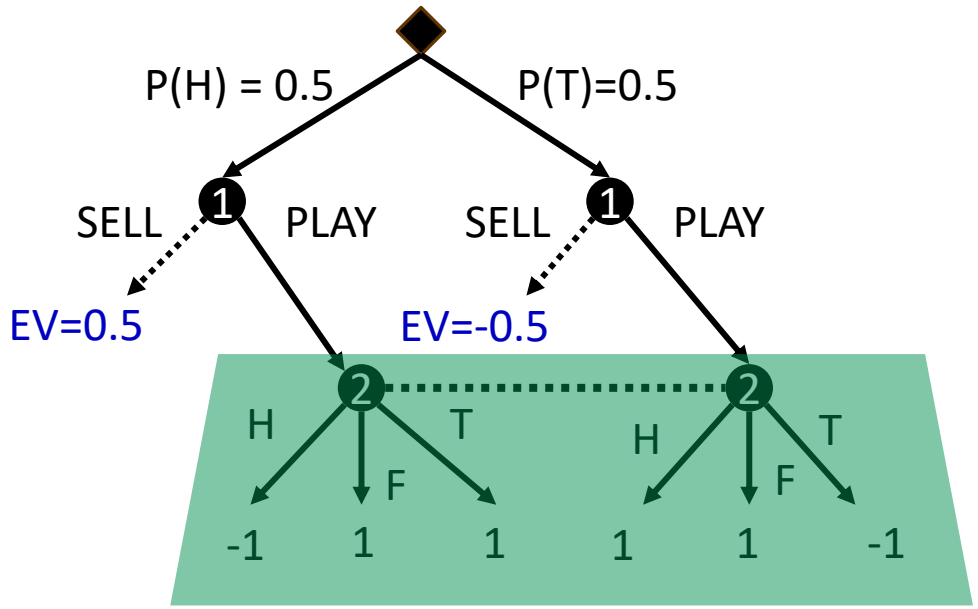
H=Heads
T=Tails
F=Forfeit

Game 1

What is the NE of this game?

- If P2 plays heads, P1's BR will sell if coin is heads, and play if tails →0.75 on avg
- If P2 plays tails, P1's BR will sell if coin is tails, and play if heads →0.25 on avg

Subgame cannot be solved independently II



H=Heads
T=Tails
F=Forfeit

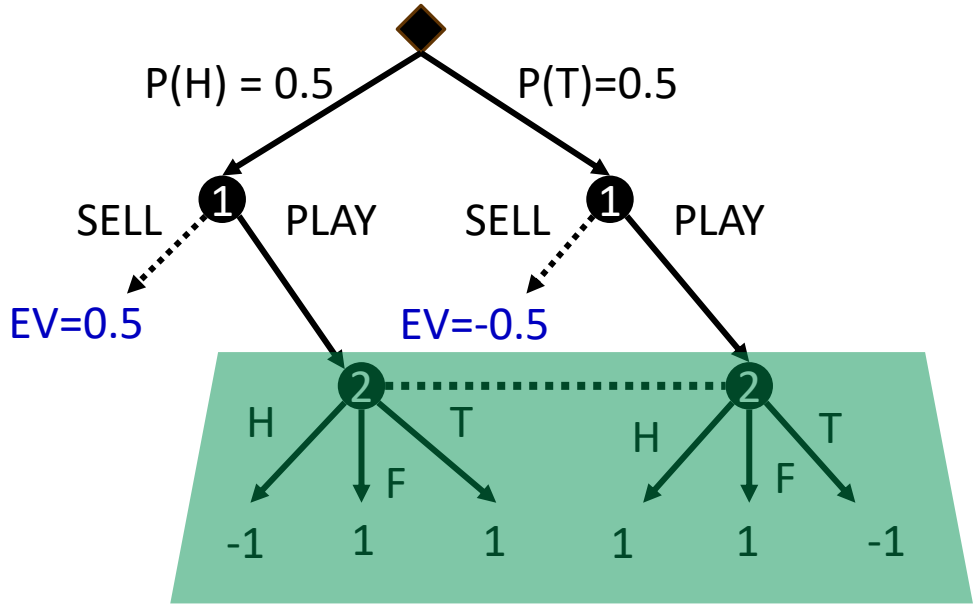
Game 1

What is the NE of this game?

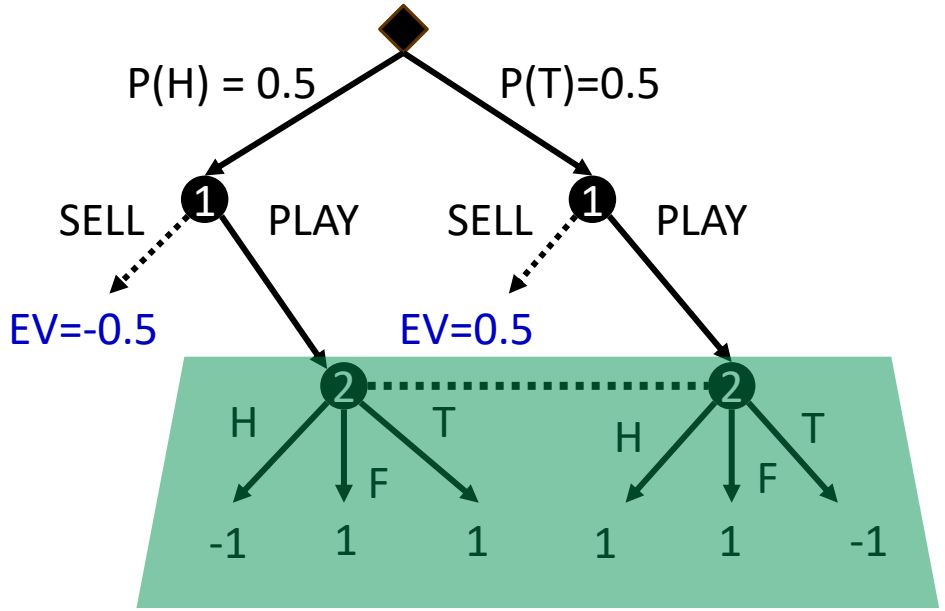
- If P2 plays heads, P1's BR will sell if coin is heads, and play if tails →0.75 on avg
- If P2 plays tails, P1's BR will sell if coin is tails, and play if heads →0.25 on avg
- NE is for P2 to play heads wp 0.25, tails 0.75→0 on avg (show!), better for P2

Subgame cannot be solved independently III

H=Heads
T=Tails
F=Forfeit

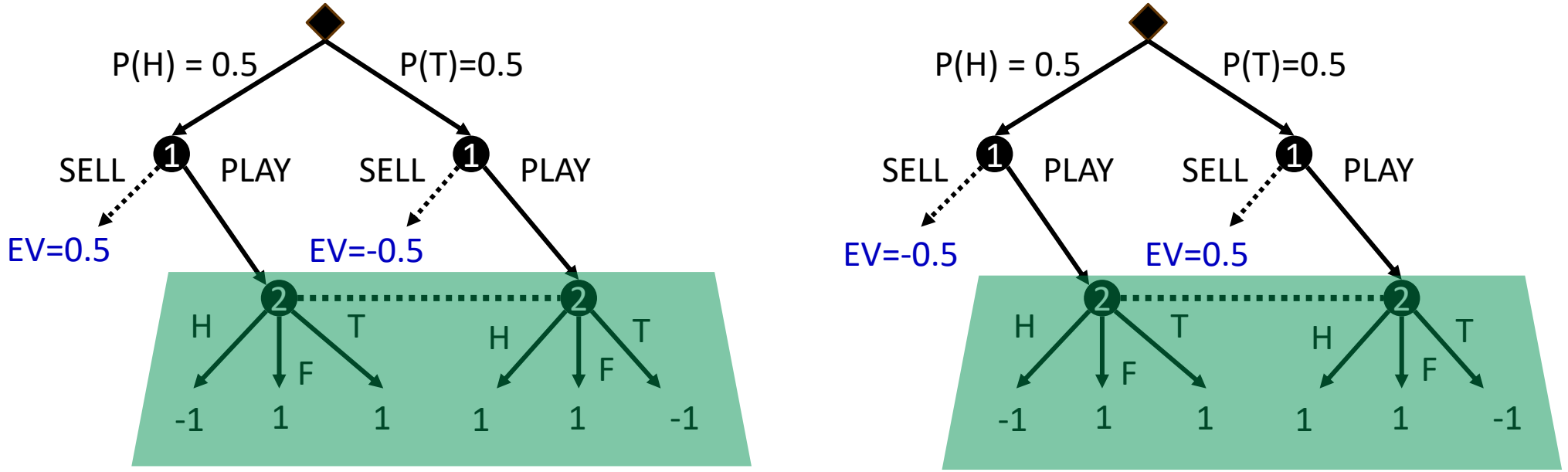


Game 1



Game 2

Subgame cannot be solved independently III



H=Heads
T=Tails
F=Forfeit

Game 1

NE = 0.25 H, 0.75 T

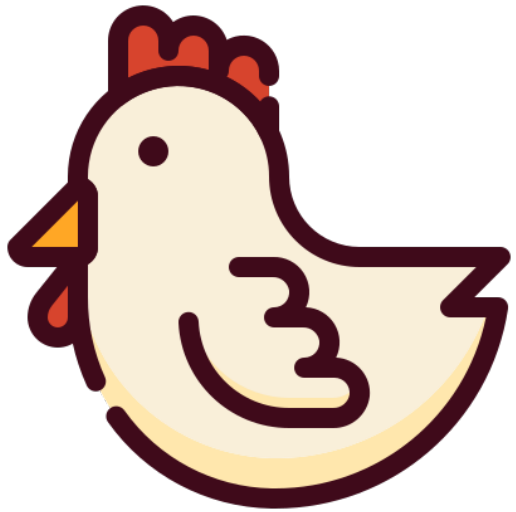
Game 2

NE = 0.75 H, 0.25 T

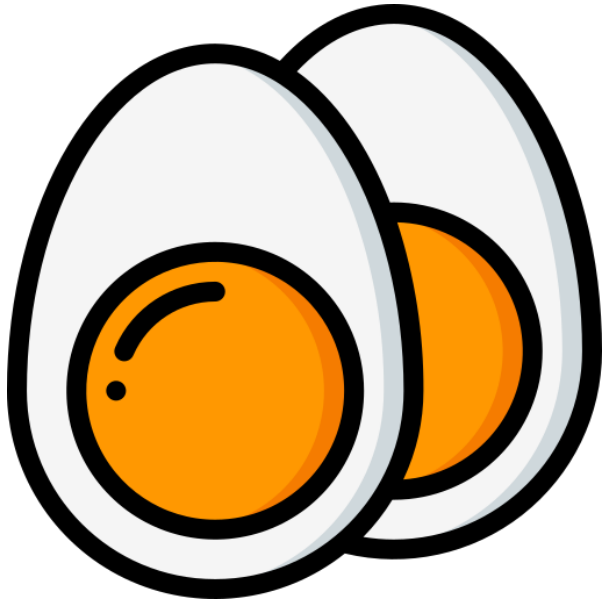
Subgame is identical, but solution is different

Parts of the game **outside** subgame affects solution within subgame

A chicken and egg problem…?



Depends on

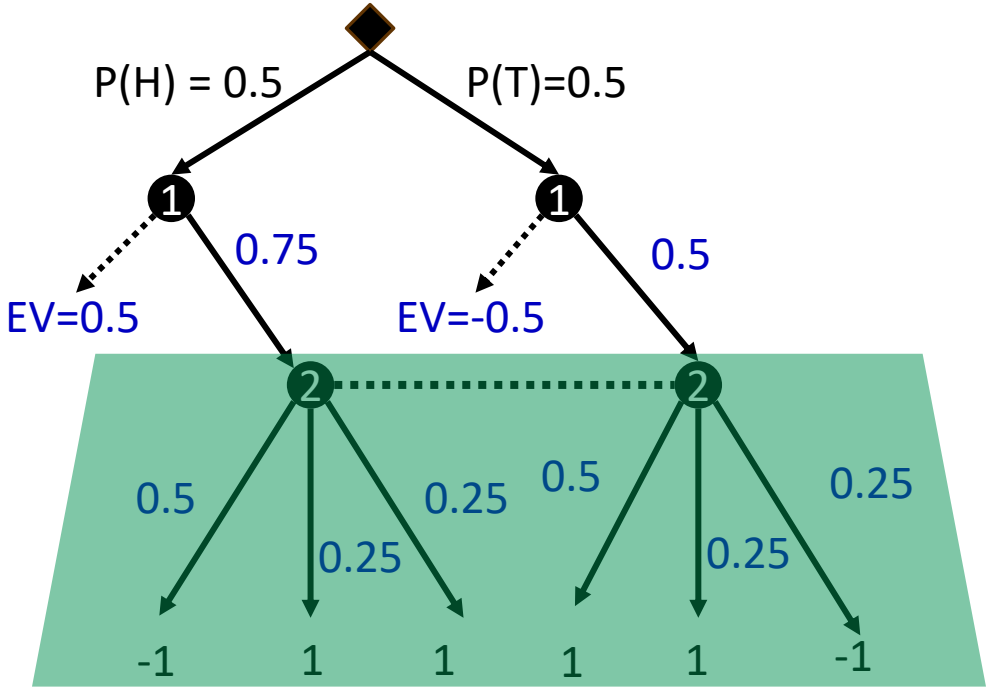


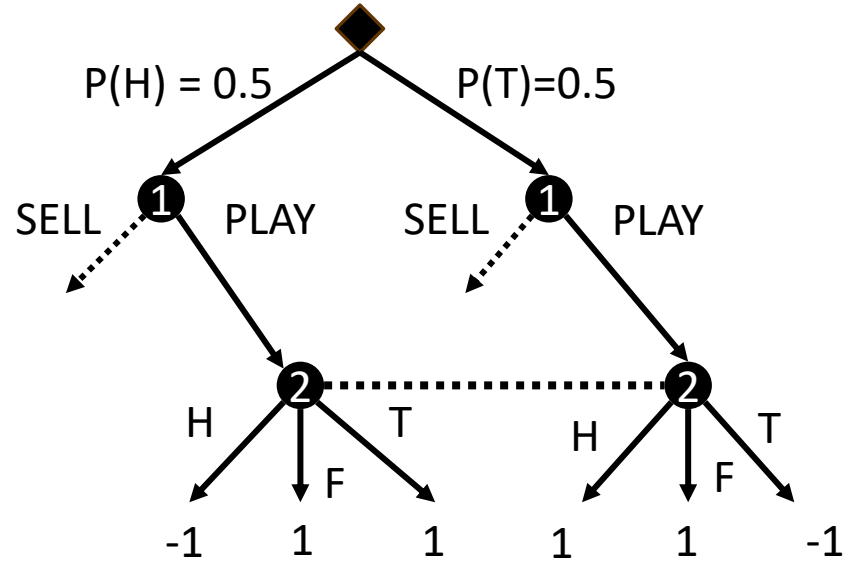
Optimal strategy in subgame

Best response of opponent
(including in pre-subgame)

Naïve/unsafe subgame solving

Coin Toss Blueprint

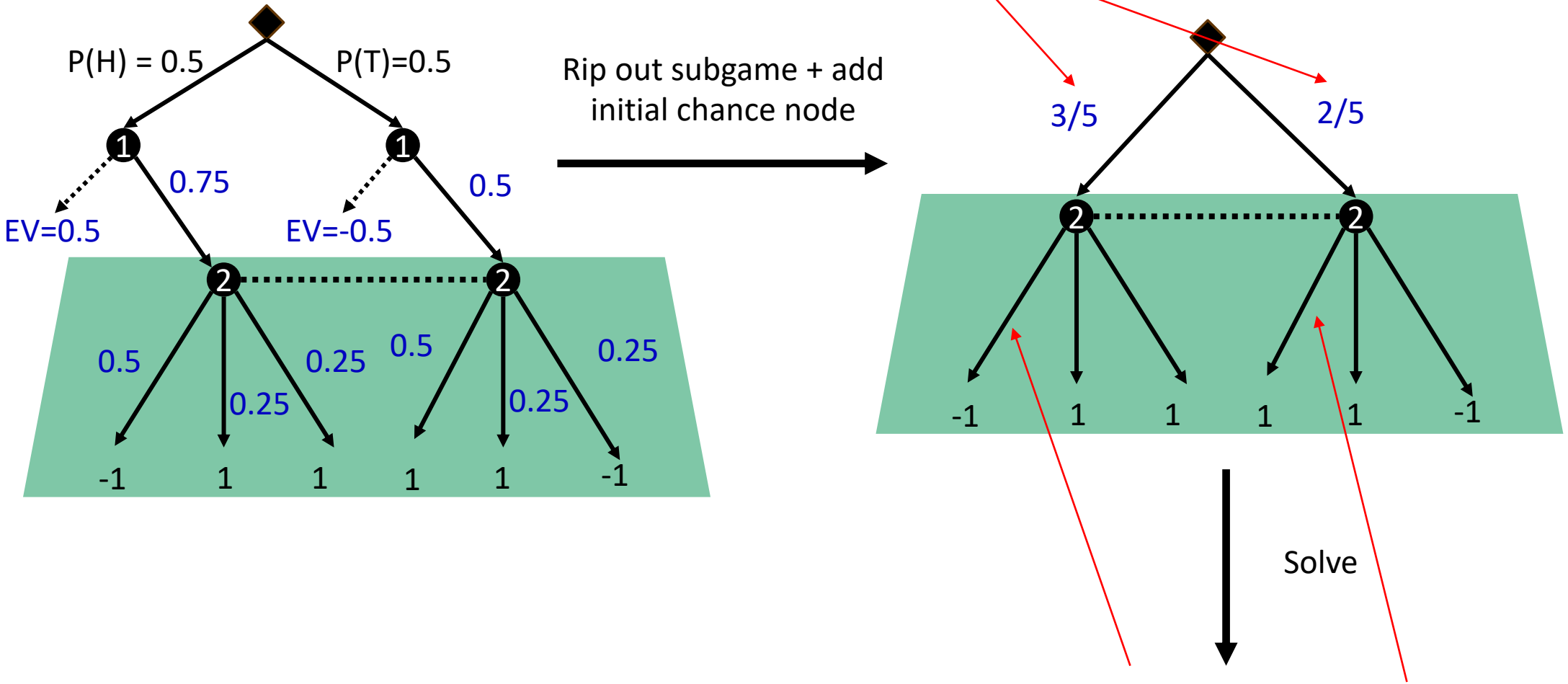




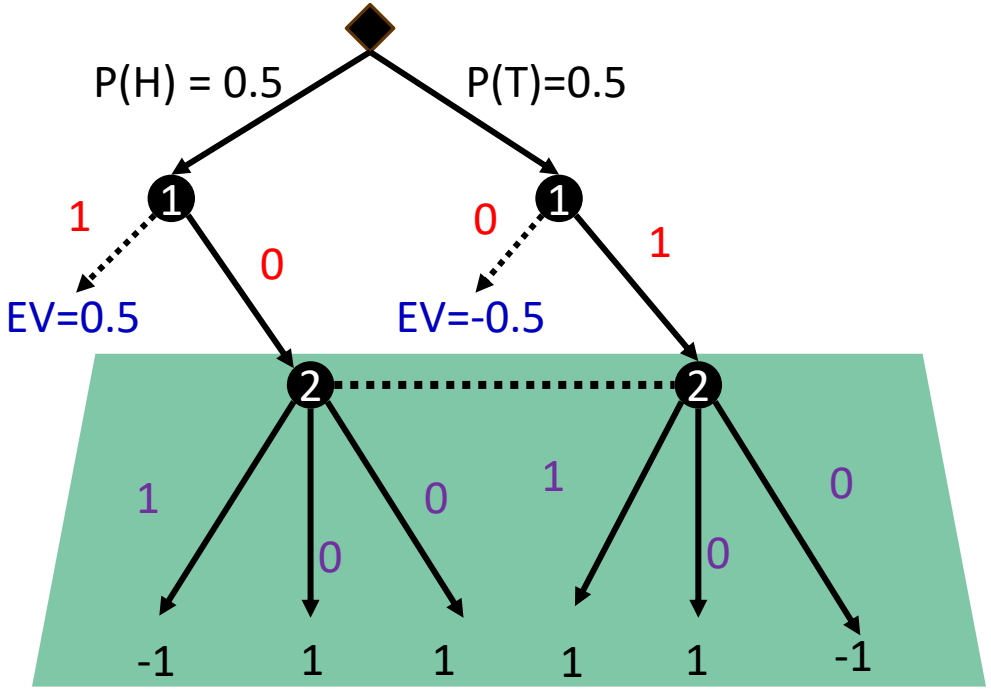
"Annotated Game"

Note: Blueprint is obviously not optimal (it chooses to forfeit), but that is besides the point

Naïve subgame solving



But what does your opponent think?



What would P1 do, **if it knew you are going to do a refinement this way**?

- P1 best responds to the refined strategy!

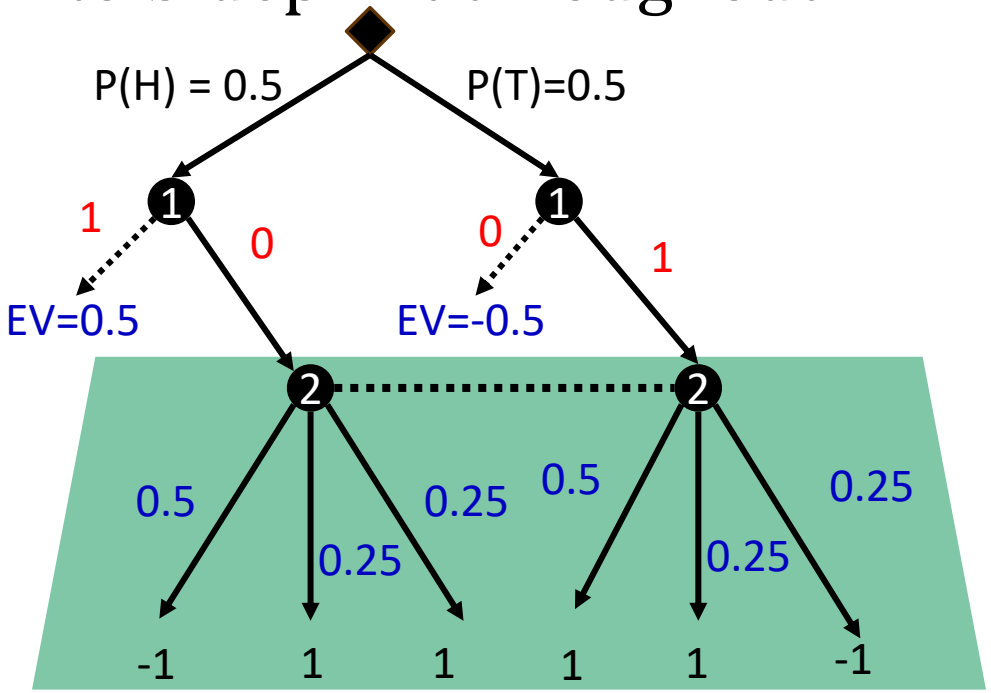
Plays only when coin toss is tails

Expected utility = P(H) * 0.5 + P(T) * 1 = 0.75

- Reminder: P1 (the opponent) is maximizing

What if you ditched subgame solving?

Stick to blueprint throughout



Note: as P2, you cannot control what P1 plays

What would P1 do, **if it knew you were sticking to blueprint?**

- Best responds to blueprint

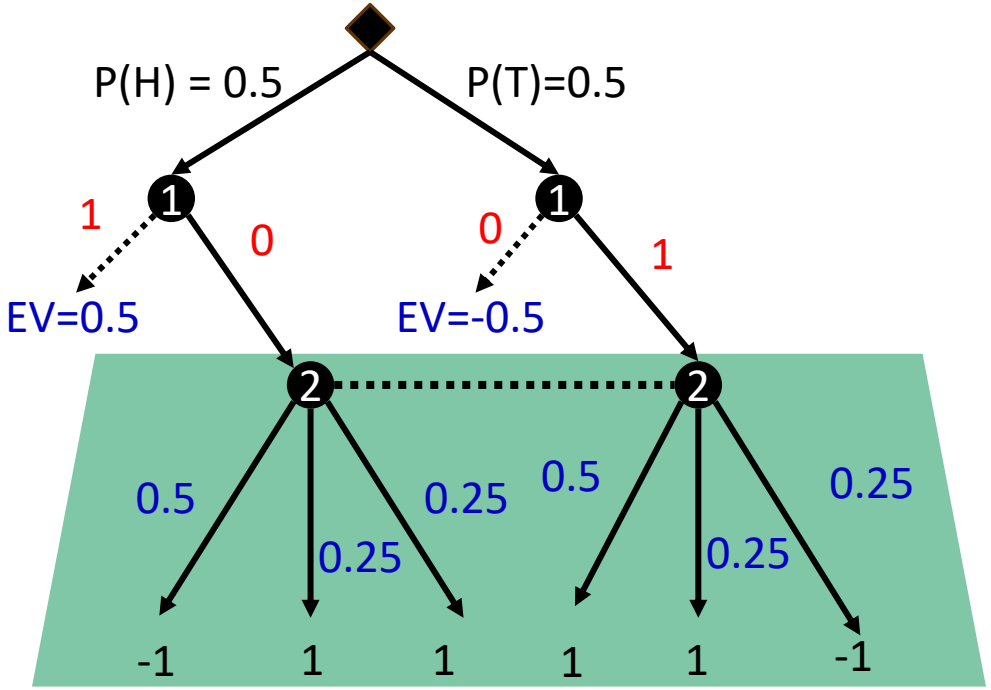
Plays only when coin toss is tails

- Same as before

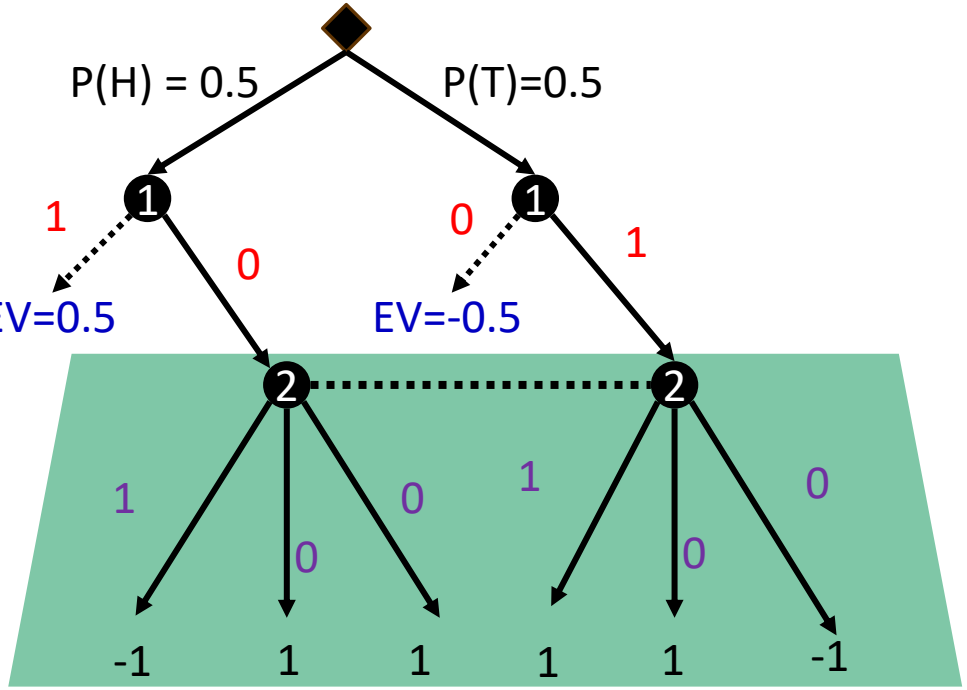
Expected utility = P(H) * 0.5 + P(T) *(0.5+0.25-0.25)=0.5

- Reminder: P1 (the opponent) is maximizing

Naïve subgame solving is counterproductive!

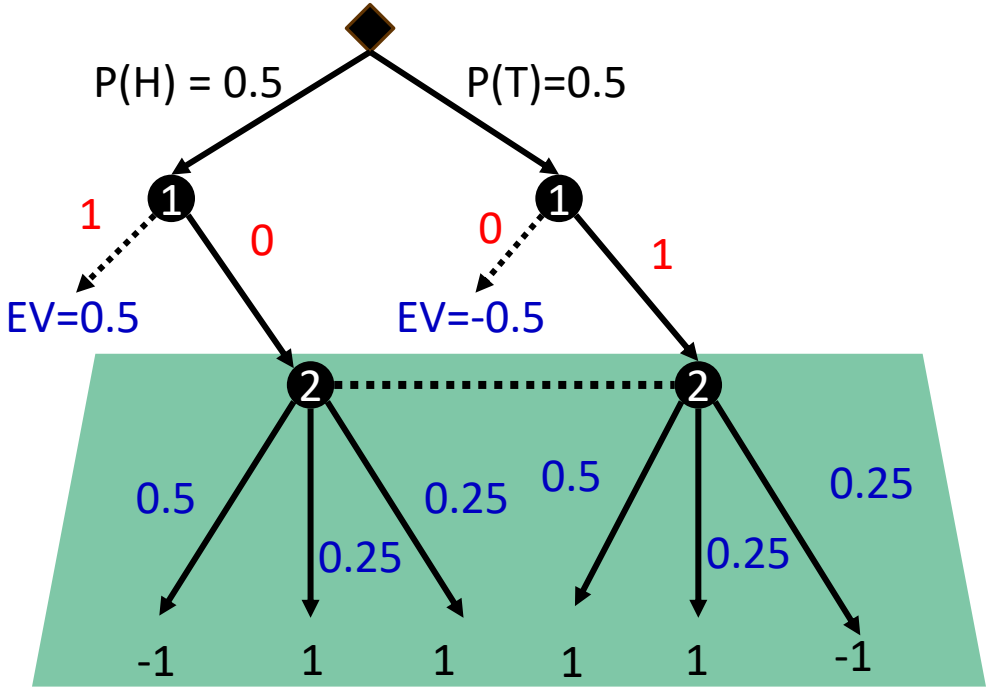


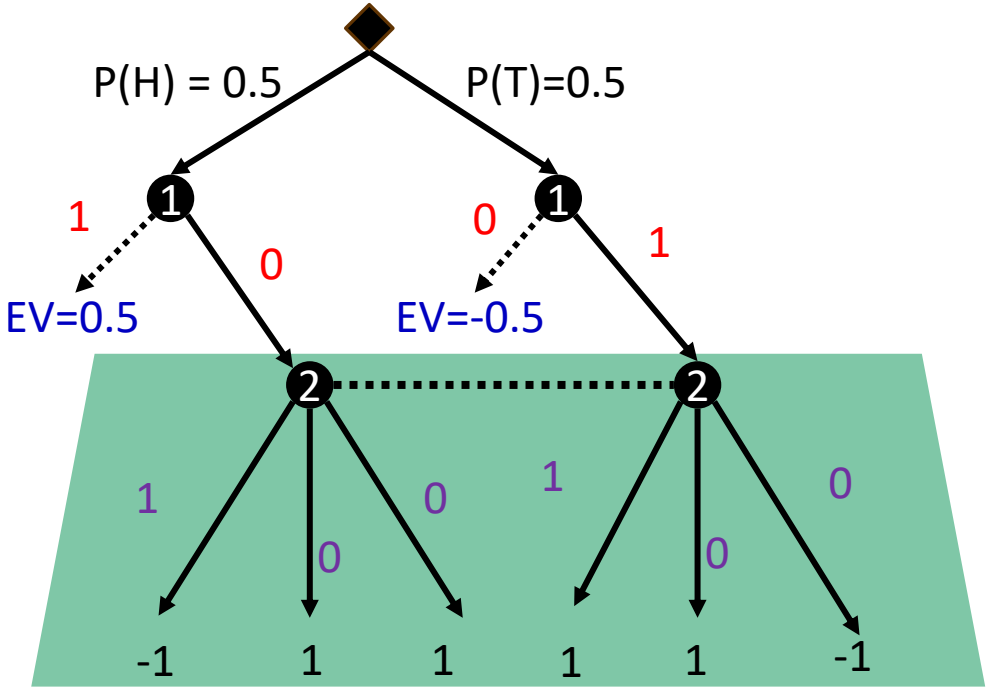
If we stuck to the blueprint



If we did refinement based on the BP

Naïve subgame solving is counterproductive!



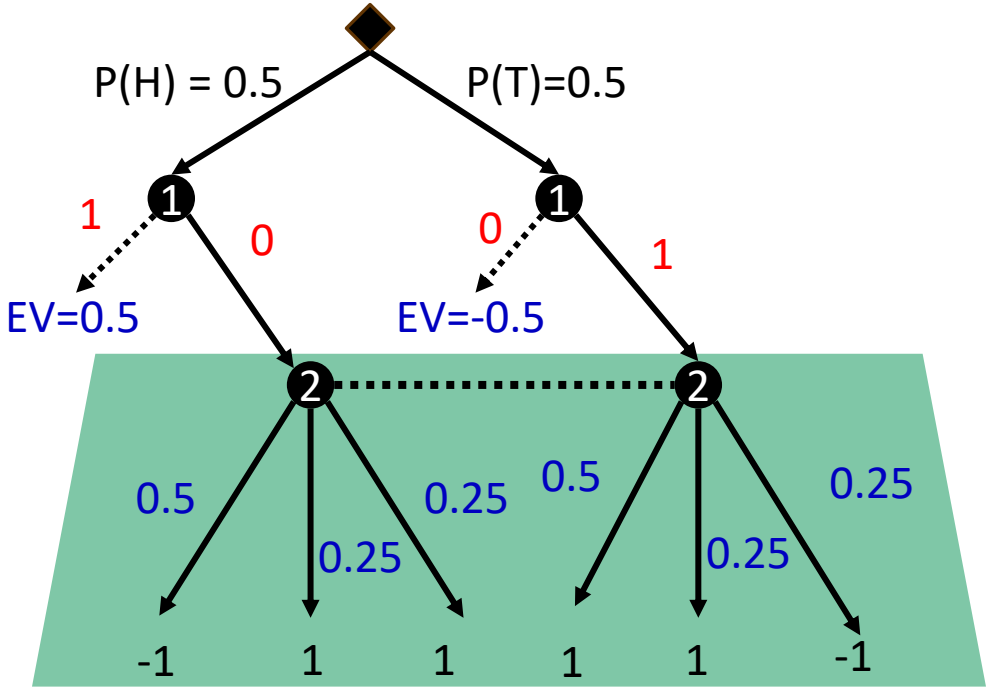


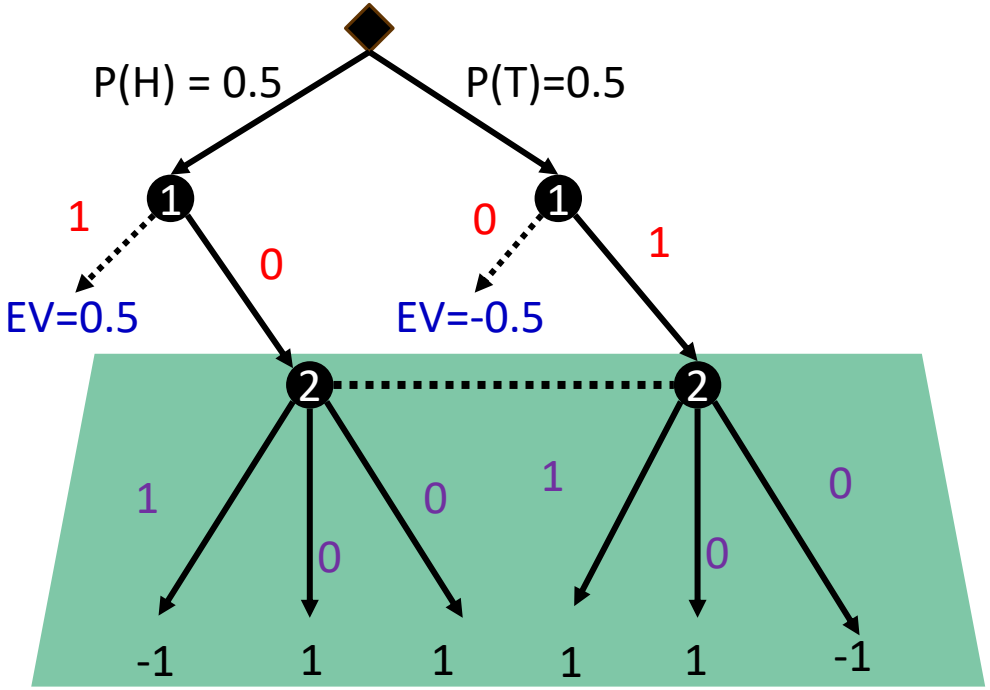
If we stuck to the blueprint

0.5

If we did refinement based on the BP

Naïve subgame solving is counterproductive!





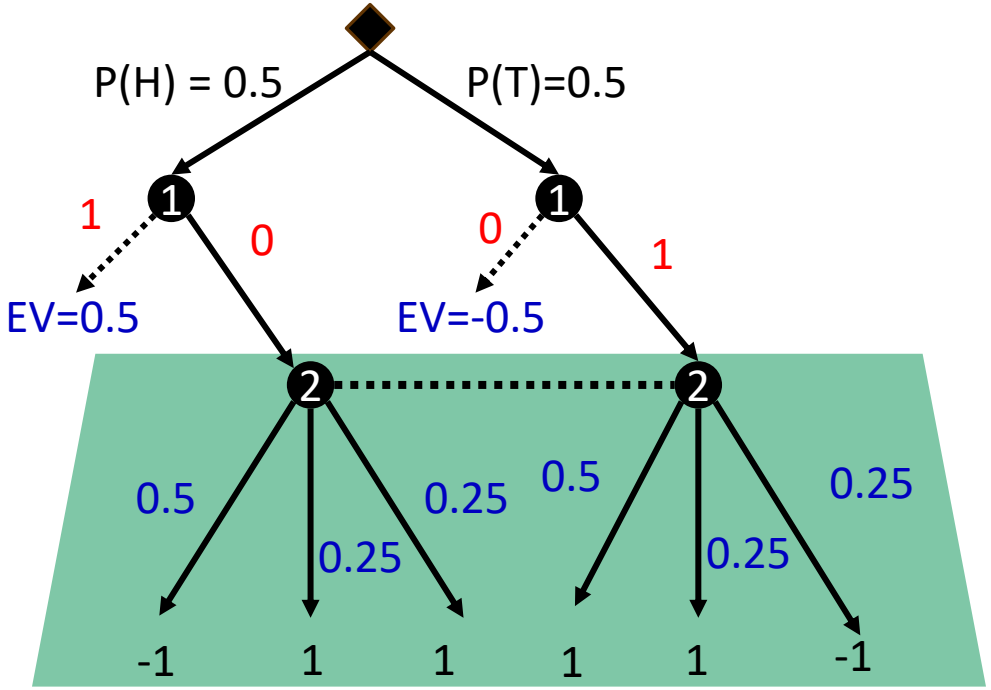
If we stuck to the blueprint

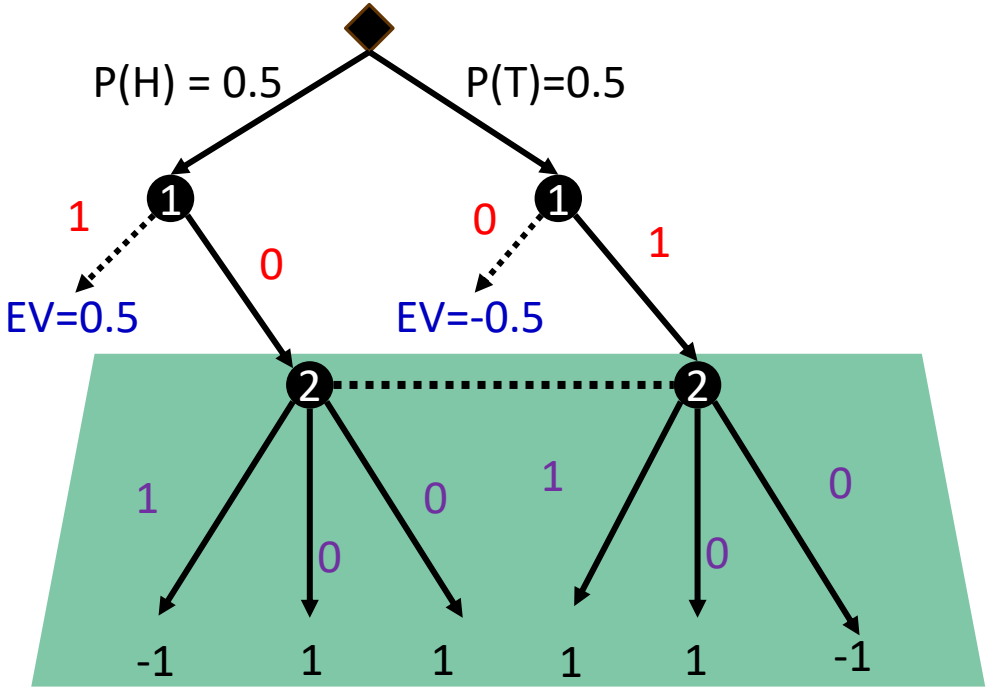
0.5

If we did refinement based on the BP

0.75

Naïve subgame solving is counterproductive!





If we stuck to the blueprint

If we did refinement based on the BP

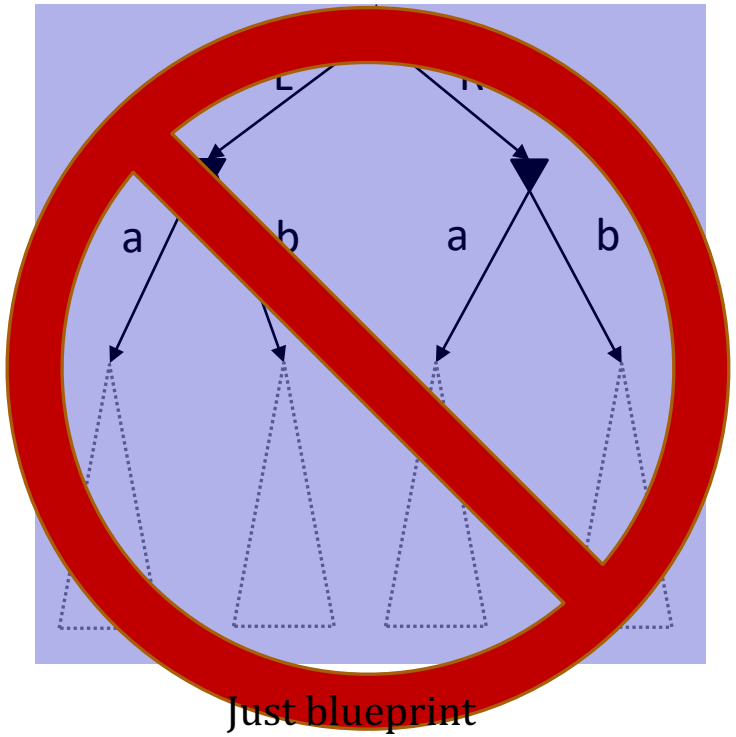
0.5 < 0.75

Is better than
(from P2's perspective)

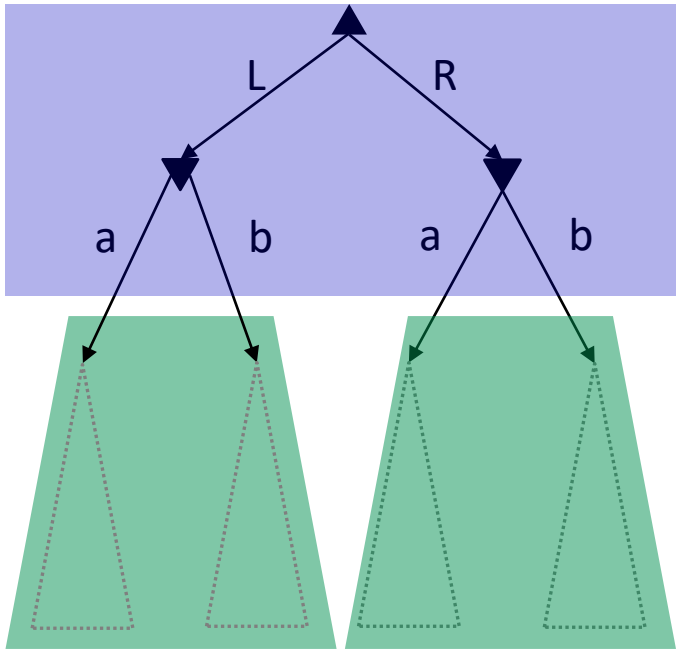
Insight #2: Opponent BRs to refined strategy

Since we are dealing with Nash, we are thinking of the worst-case opponent. From the opponent's viewpoint, you are not playing the blueprint, but rather, **blueprint + refined strategy**

- As if refinement "source code" was uploaded online



Just blueprint



Blueprint + refinement

Safe Refinements

Safe Refinements

Safety: Refined strategy performs **no worse than blueprint**

- Refinements are never counterproductive
- No harm in “dumping” extra compute on refinements

Safe Refinements

Safety: Refined strategy performs **no worse than blueprint**

- Refinements are never counterproductive
- No harm in “dumping” extra compute on refinements

Other player best-responds to blueprint **+ refined strategy**

- As if refinement “source code” was uploaded online

Safe Refinements

Safety: Refined strategy performs **no worse than blueprint**

- Refinements are never counterproductive
- No harm in “dumping” extra compute on refinements

Other player best-responds to blueprint **+ refined strategy**

- As if refinement “source code” was uploaded online

Naïve subgame solving is **unsafe**

Safe Refinements

Safety: Refined strategy performs **no worse than blueprint**

- Refinements are never counterproductive
- No harm in "dumping" extra compute on refinements

Other player best-responds to blueprint **+ refined strategy**

- As if refinement "source code" was uploaded online

Naïve subgame solving is **unsafe**

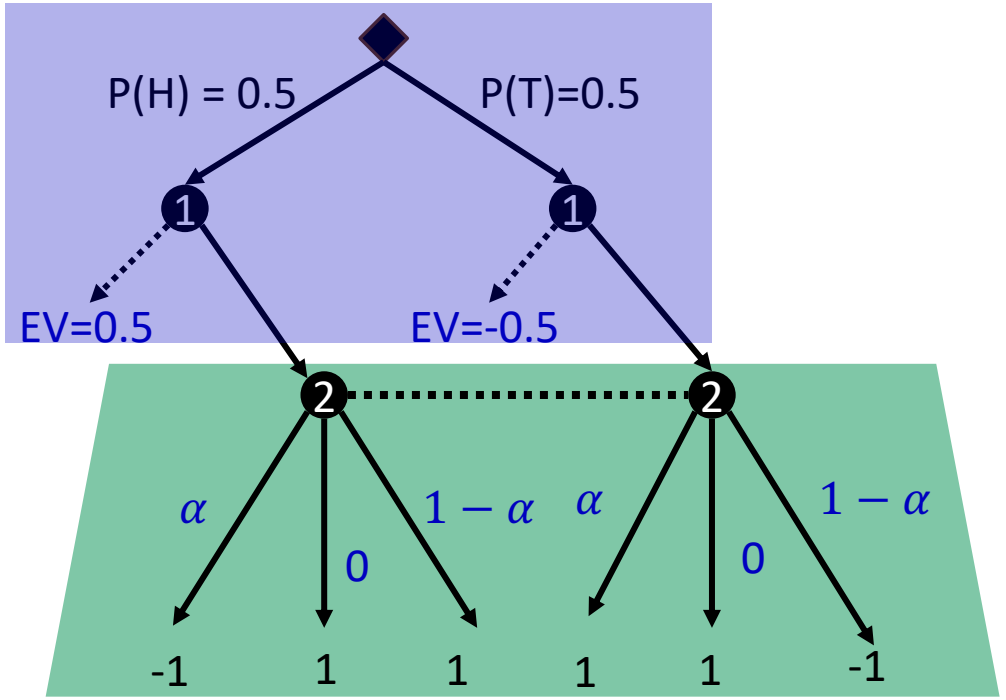
Note: in **perfect information 2p0s games** naïve subgame solving is **trivially safe**. Why?

Safe subgame solving

Most material from: Safe and Nested Subgame Solving for Imperfect-Information Games (Brown and Sandholm 2017)

Why is naïve subgame solving unsafe?

Go back to the basics: opponent is **best responding**

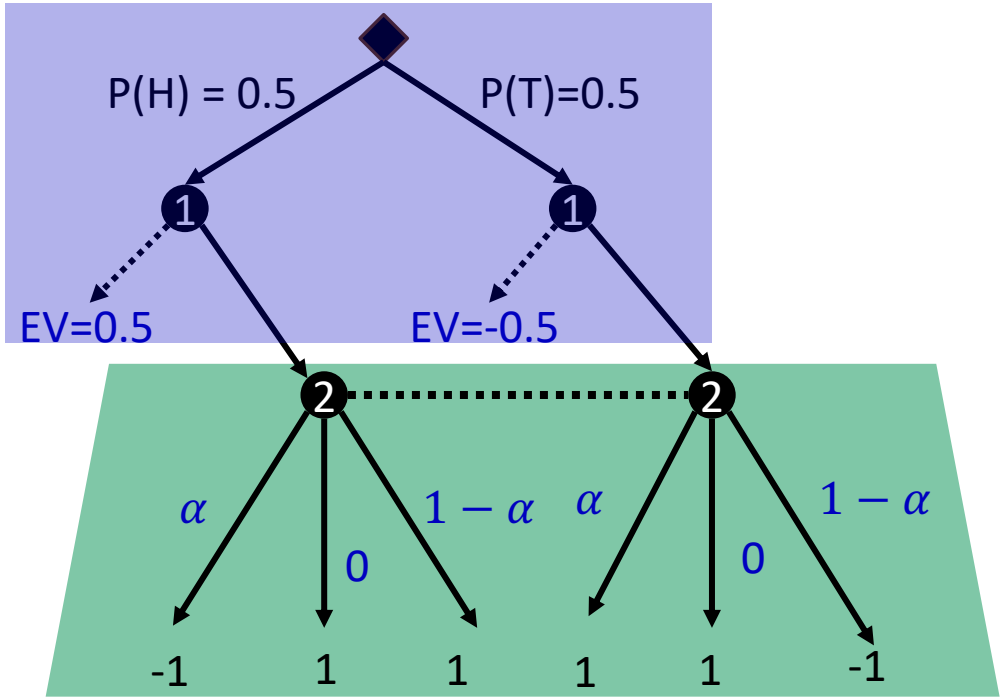


- When $\alpha = 0$
  - H→PLAY, T→SELL
- When $\alpha = 1$
  - H→SELL, T→PLAY
- When $\alpha = 0.25$,
  - H→INDIFFERENT, T→INDIFFERENT

As α increases, Heads (left branch) improves but Tails (right branch) deteriorates from P2's perspective

Why is naïve subgame solving unsafe?

Go back to the basics: opponent is **best responding**



- When $\alpha = 0$
  - H→PLAY, T→SELL
- When $\alpha = 1$
  - H→SELL, T→PLAY
- When $\alpha = 0.25$,
  - H→INDIFFERENT, T→INDIFFERENT

As α increases, Heads (left branch) improves but Tails (right branch) deteriorates from P2's perspective

Pre-subgame actions of opponent will change "in response" to strategy refinement!

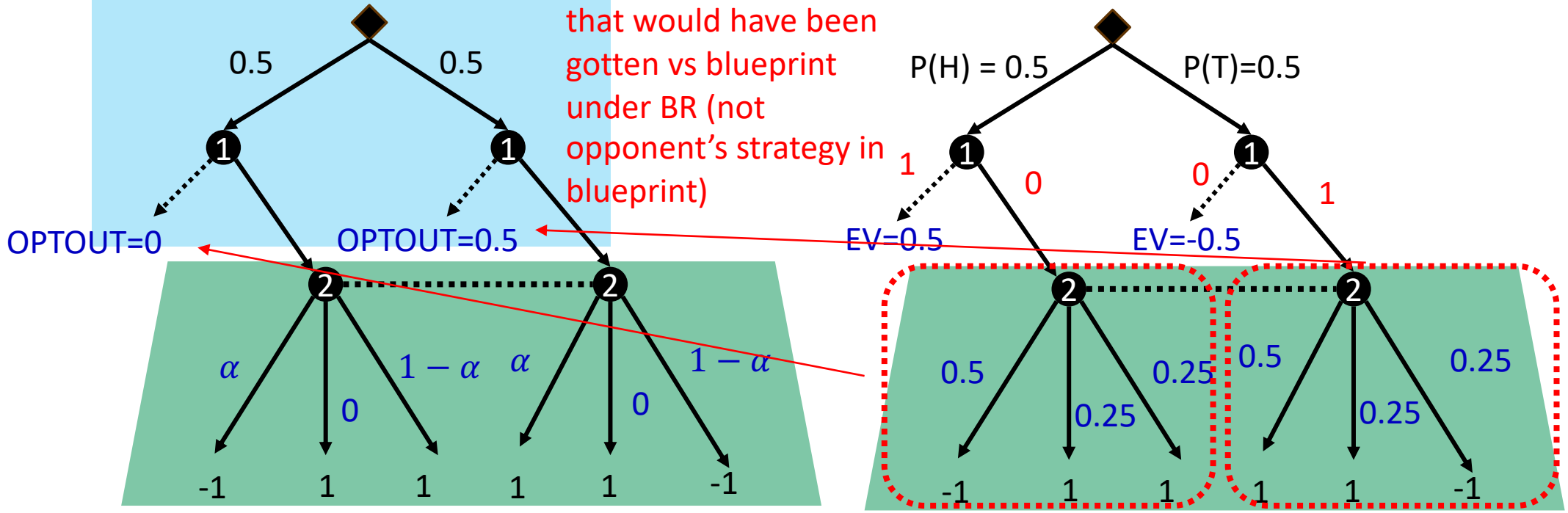
Resolve Subgame Solving

Idea: use gadget to add in an "opt-out" option for the opponent

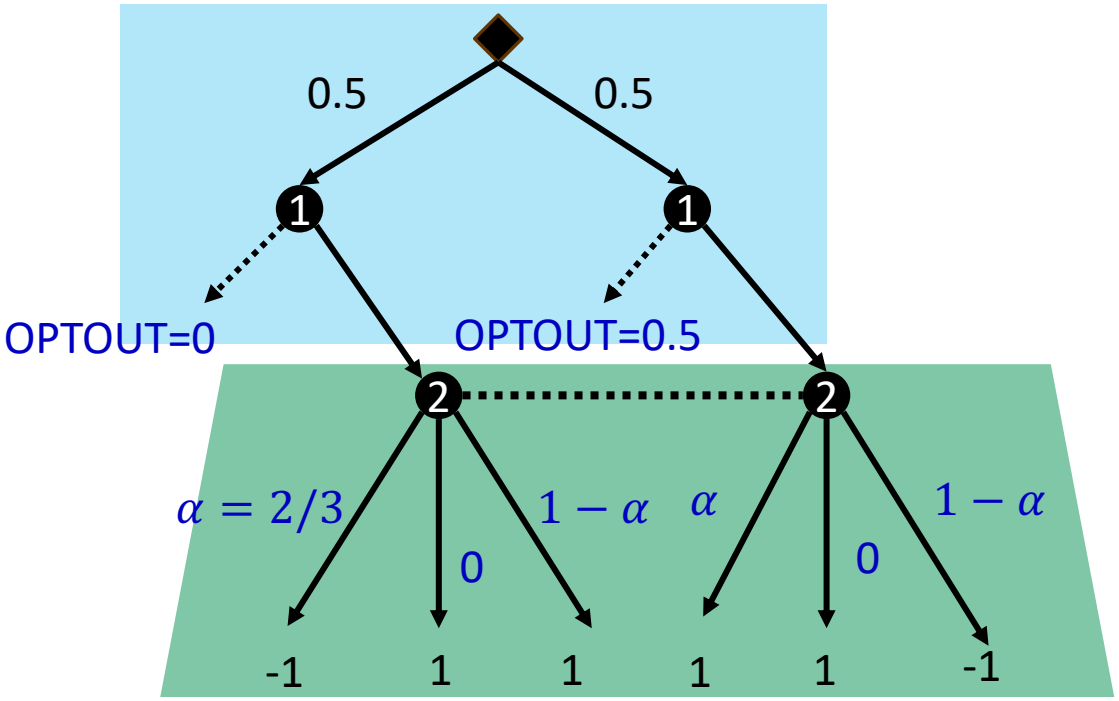
- Value of opting out is based on value under BR to blueprint
- Ensures that we cannot "hallucinate" rewards even though opponent would have changed his best response such that subgame isn't even entered

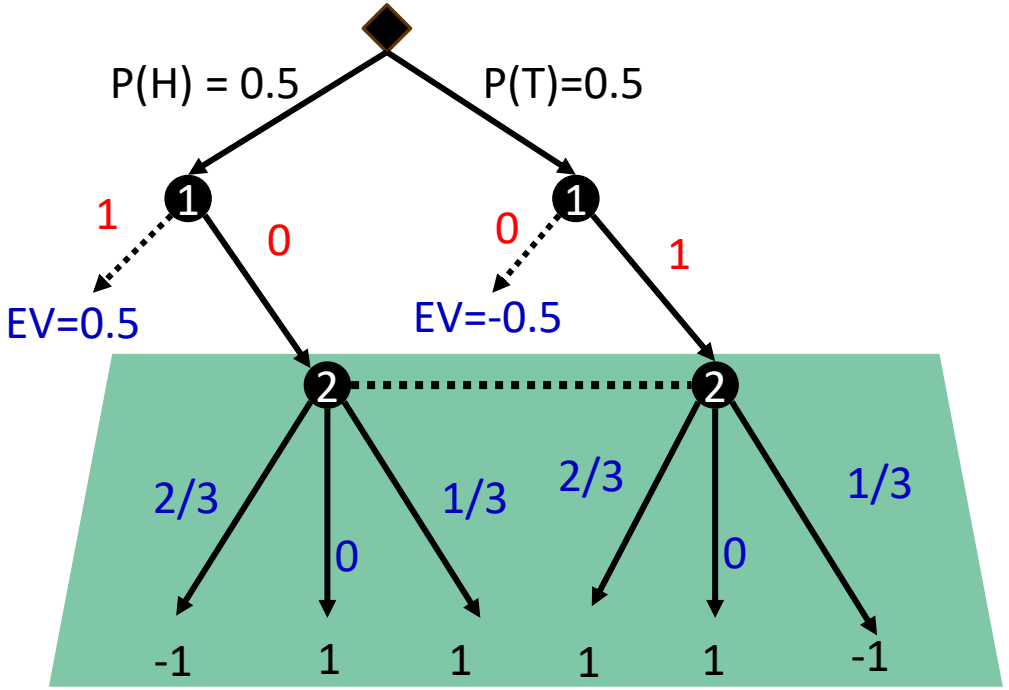
Warning: This is **not** the same game (the "pre-subgame part" in this gadget are additional vertices added and have nothing to do with the pre-subgame part in the original game)

Note: these are values that would have been gotten vs blueprint under BR (not opponent's strategy in blueprint)

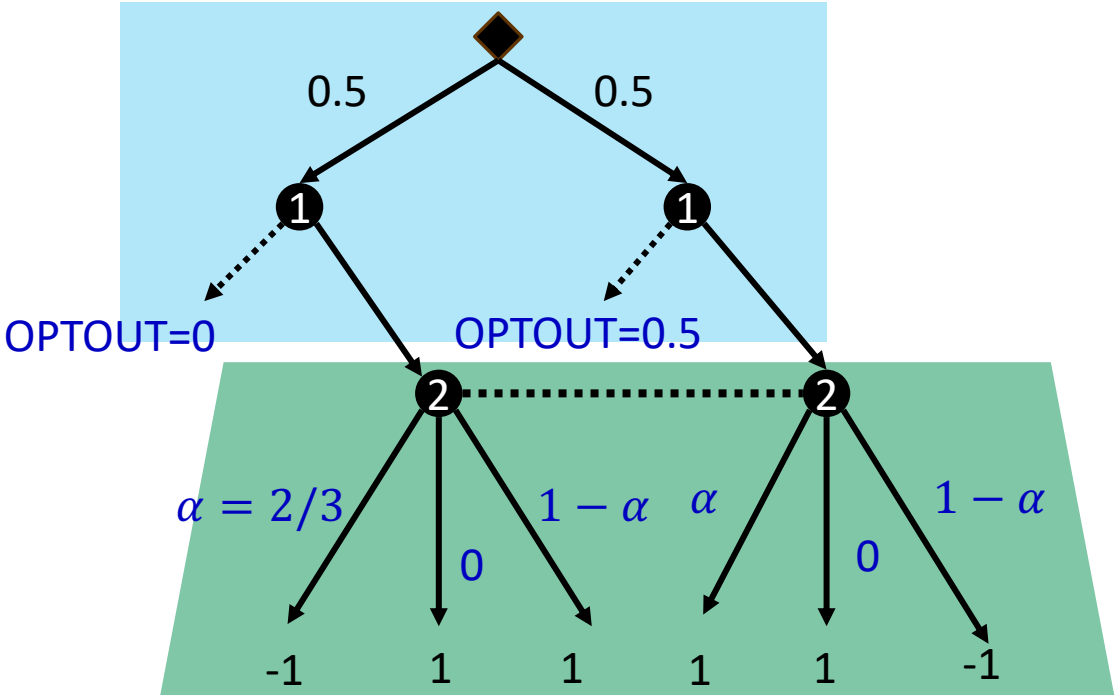


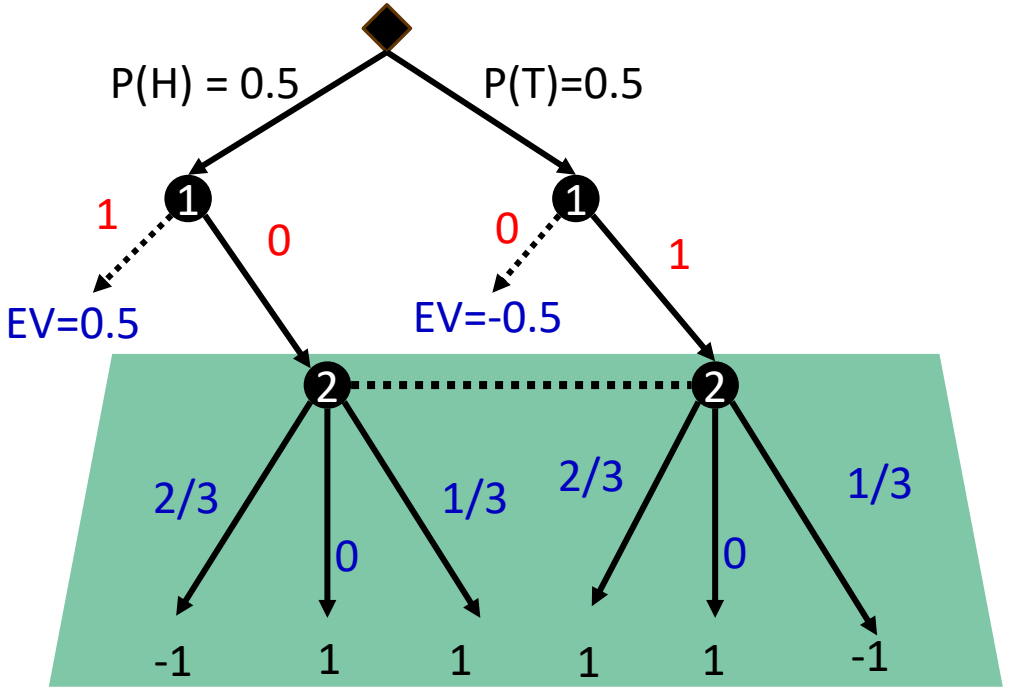
Examples of refined strategy I





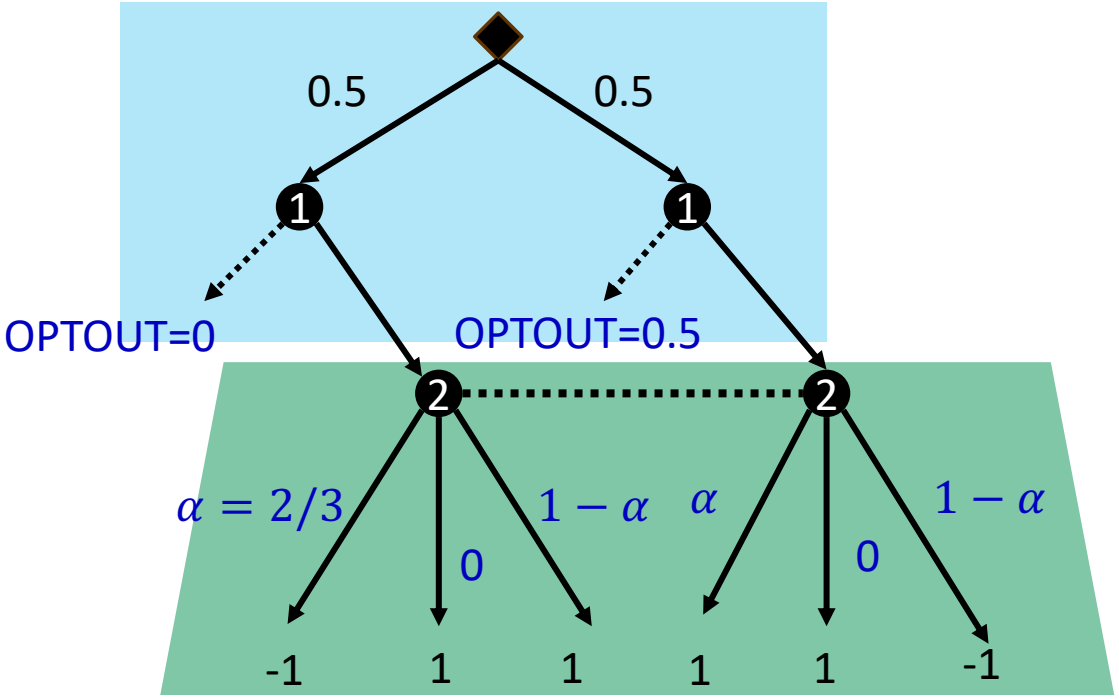
Examples of refined strategy I

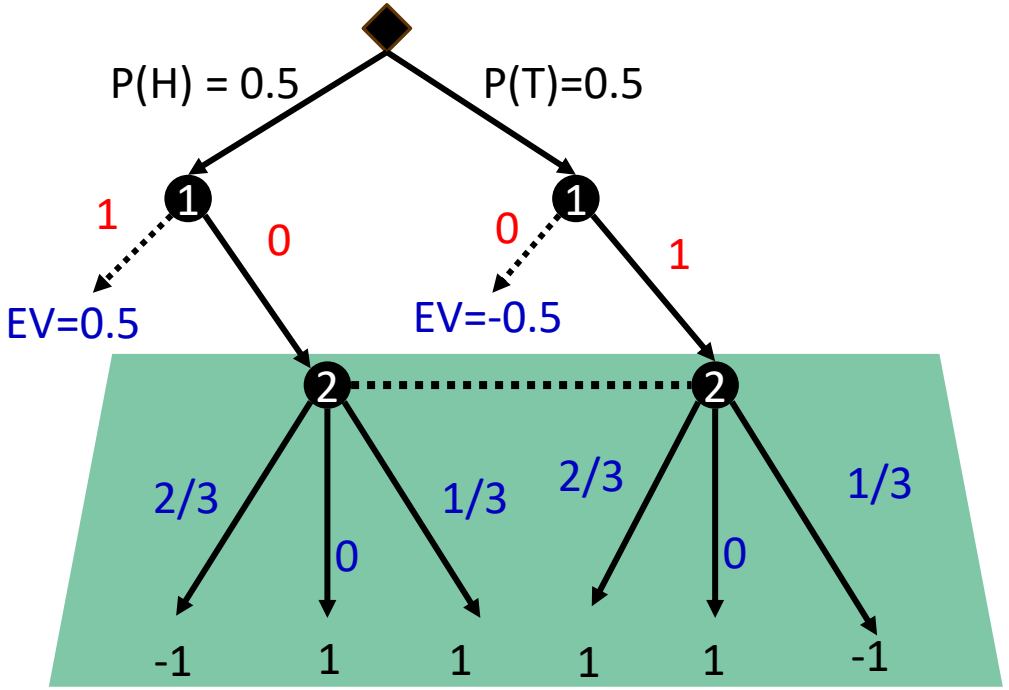




Nash of original game (taking components in subgame) is also a Nash in this game (why?)
What is player 1's BR here?

Examples of refined strategy I

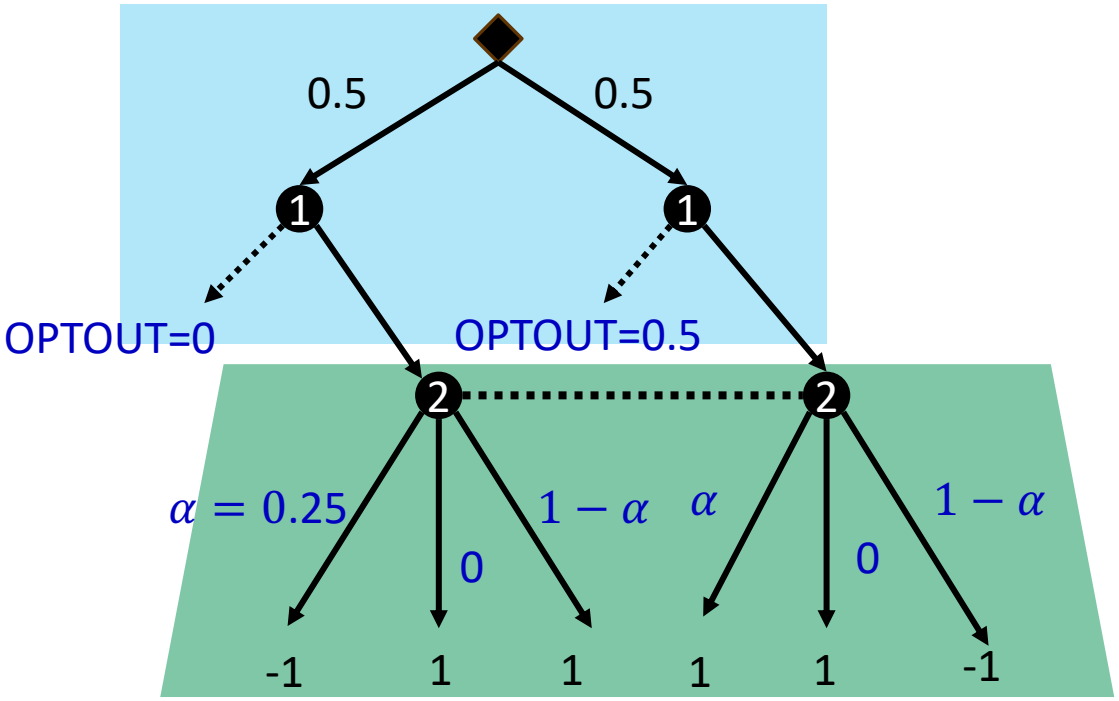


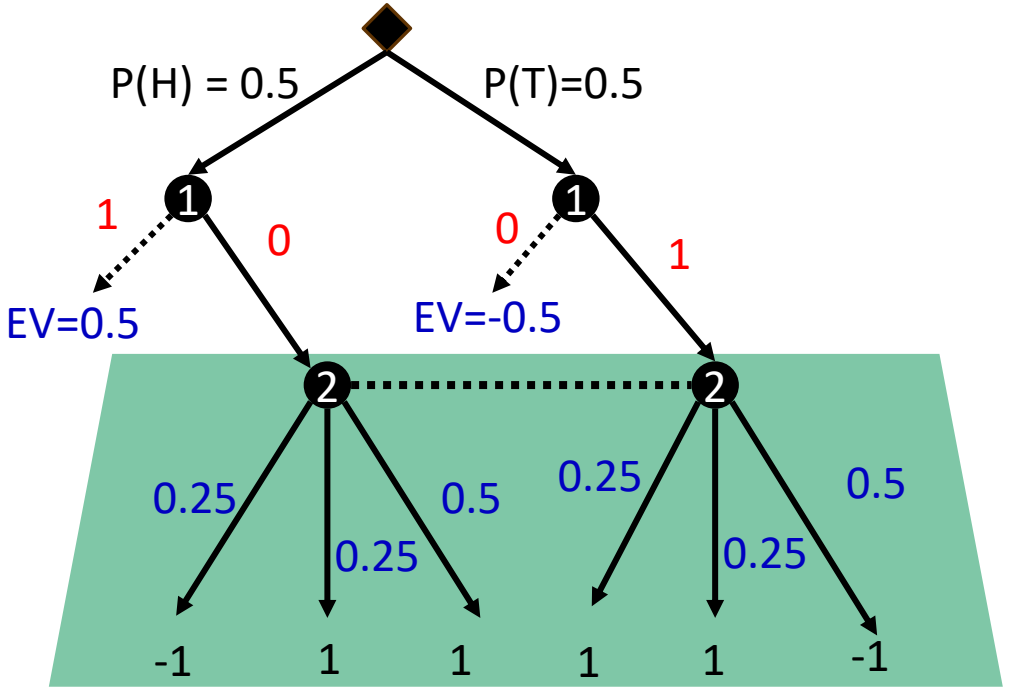


Nash of original game (taking components in subgame) is also a Nash in this game (why?)
What is player 1's BR here?

Payoff under refined strategy is: 0.5 * 0.5 + 0.5 * 1/3 = 0.416 < Blueprint
Strict improvement from blueprint!

Examples of refined strategy II





The blueprint itself is also a Nash of the gadget game…→No improvement!

Resolve Subgame Solving is Safe

Why?

- Guarantees that probability distribution for refined strategy at “head” of subgame is the same as blueprint

But usually won’t lead to meaningful improvements

- Too stringent!
- Gadget game doesn’t allow for much improvement, not explicitly at least
- We solved the problem of ”sacrificing one branch for the other”, but in turn made it very hard to improve

Resolve Subgame Solving is Safe

Why?

- Guarantees that probability distribution for refined strategy at “head” of subgame is the same as blueprint

But usually won’t lead to meaningful improvements

- Too stringent!
- Gadget game doesn’t allow for much improvement, not explicitly at least
- We solved the problem of ”sacrificing one branch for the other”, but in turn made it very hard to improve

How about allowing cases where **every** “head subgame” vertex’s EV is improved?

Maxmargin Subgame Solving

Caveat! We can relax this

Idea (from before): if values of all starting subtrees in subgame (under BR of opponent) do not increase, then we are **safe**

- The problem of non-safety came about because we didn't know which branch we will end up in, **since opponent BRs to us even pre-subgame**
- But if all branches are not worse than blueprint (under opponent BR), then we can't have performed worse

Maxmargin Subgame Solving

Caveat! We can relax this

Idea (from before): if values of all starting subtrees in subgame (under BR of opponent) do not increase, then we are **safe**

- The problem of non-safety came about because we didn't know which branch we will end up in, **since opponent BRs to us even pre-subgame**
- But if all branches are not worse than blueprint (under opponent BR), then we can't have performed worse

What if we try to maximize the *margin*, i.e., minimum improvement from blueprint across all branches?

- Margin will never be negative because blueprint itself is feasible refinement →safe!
- If margin is strictly positive, whichever branch we end up in, we will always do strictly better than before!

Maxmargin Subgame Solving

Caveat! We can relax this

Idea (from before): if values of all starting subtrees in subgame (under BR of opponent) do not increase, then we are **safe**

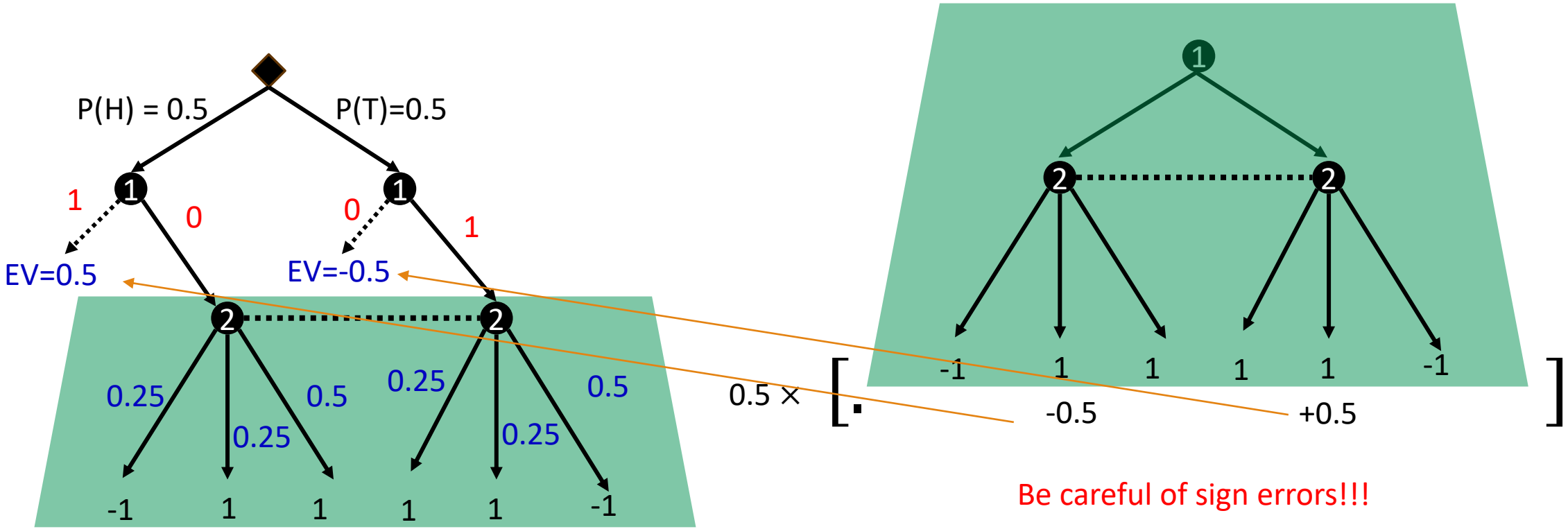
- The problem of non-safety came about because we didn't know which branch we will end up in, **since opponent BRs to us even pre-subgame**
- But if all branches are not worse than blueprint (under opponent BR), then we can't have performed worse

What if we try to maximize the *margin*, i.e., minimum improvement from blueprint across all branches?

- Margin will never be negative because blueprint itself is feasible refinement →safe!
- If margin is strictly positive, whichever branch we end up in, we will always do strictly better than before!

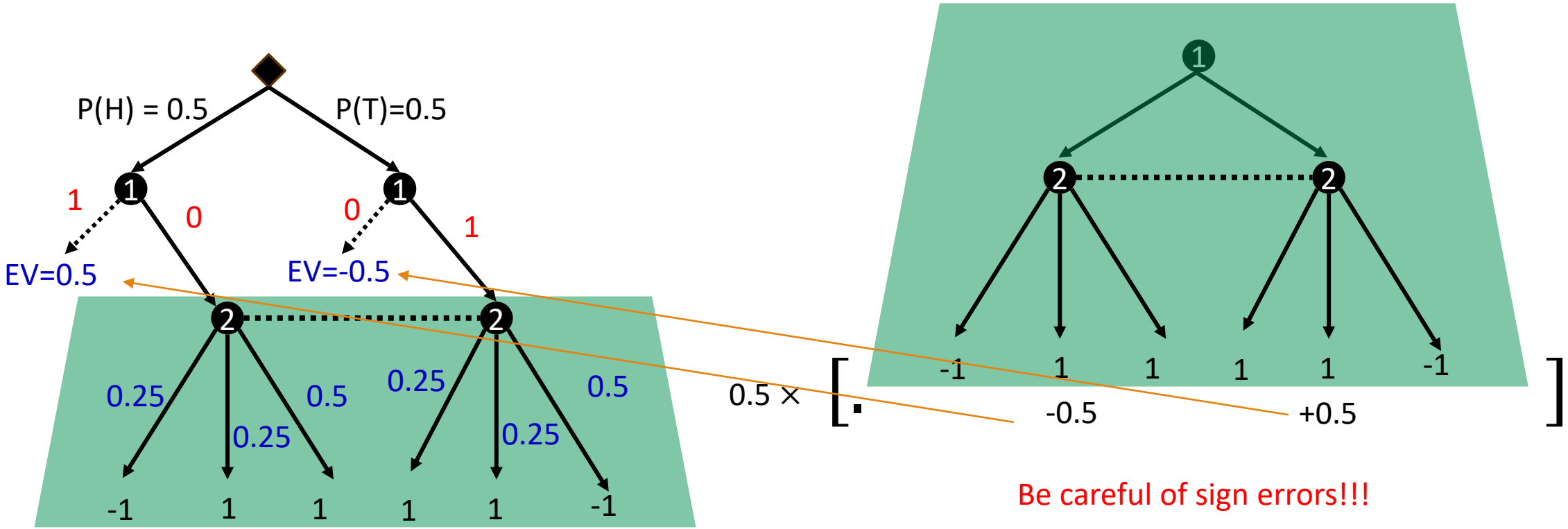
Refinement can be done via linear programming

Maxmargin Gadget I



Be careful of sign errors!!!

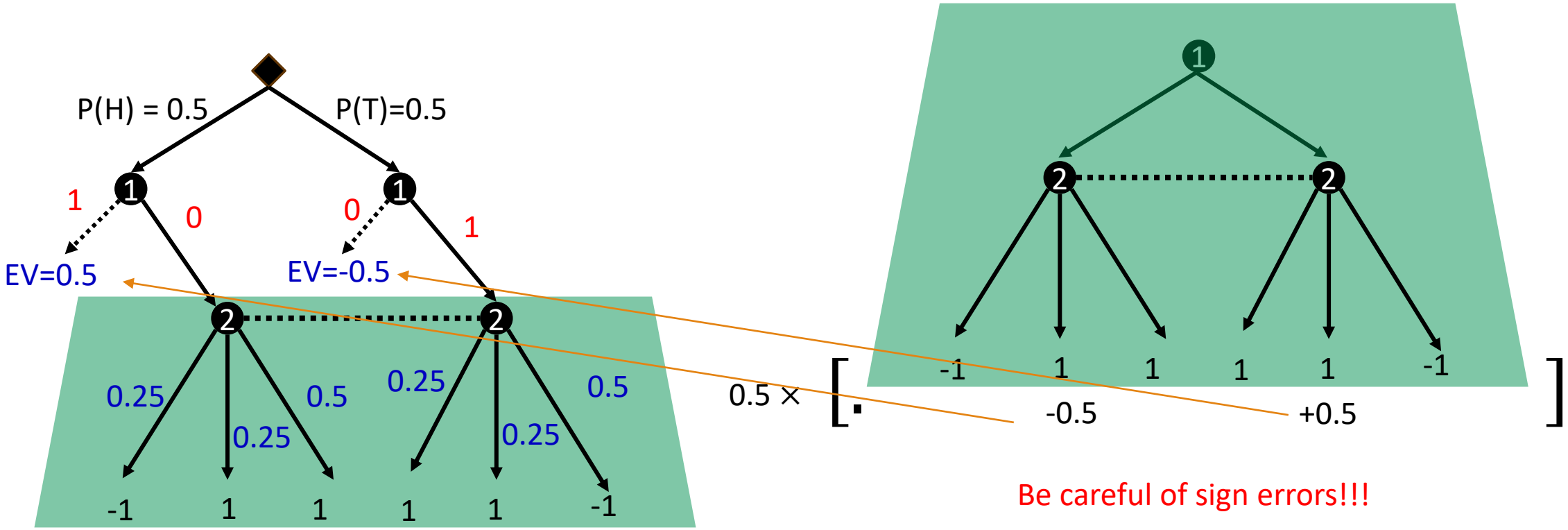
Maxmargin Gadget I



Be careful of sign errors!!!

Why does this work?

Maxmargin Gadget I



Be careful of sign errors!!!

Why does this work?

Think of maxmargin as a min-max problem, opponent now chooses the branch with the lowest margin (improvement).

Maxmargin Gadget II

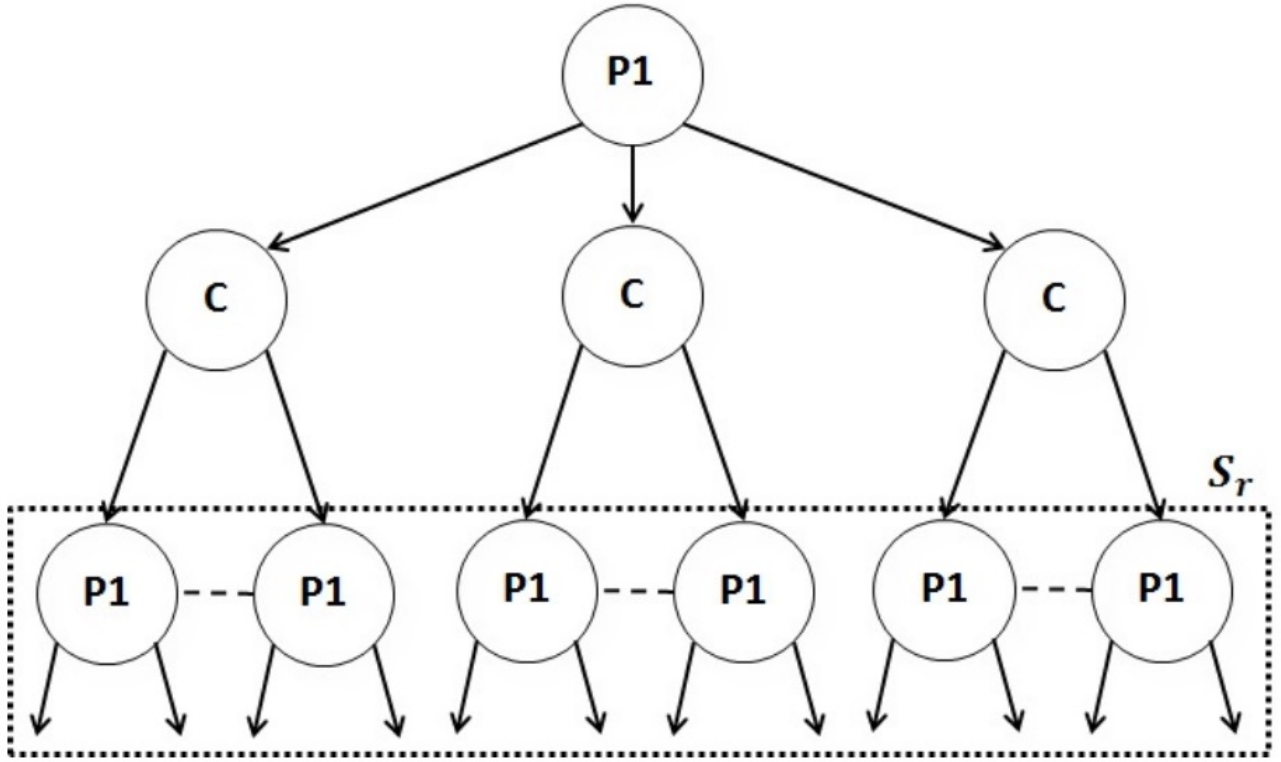
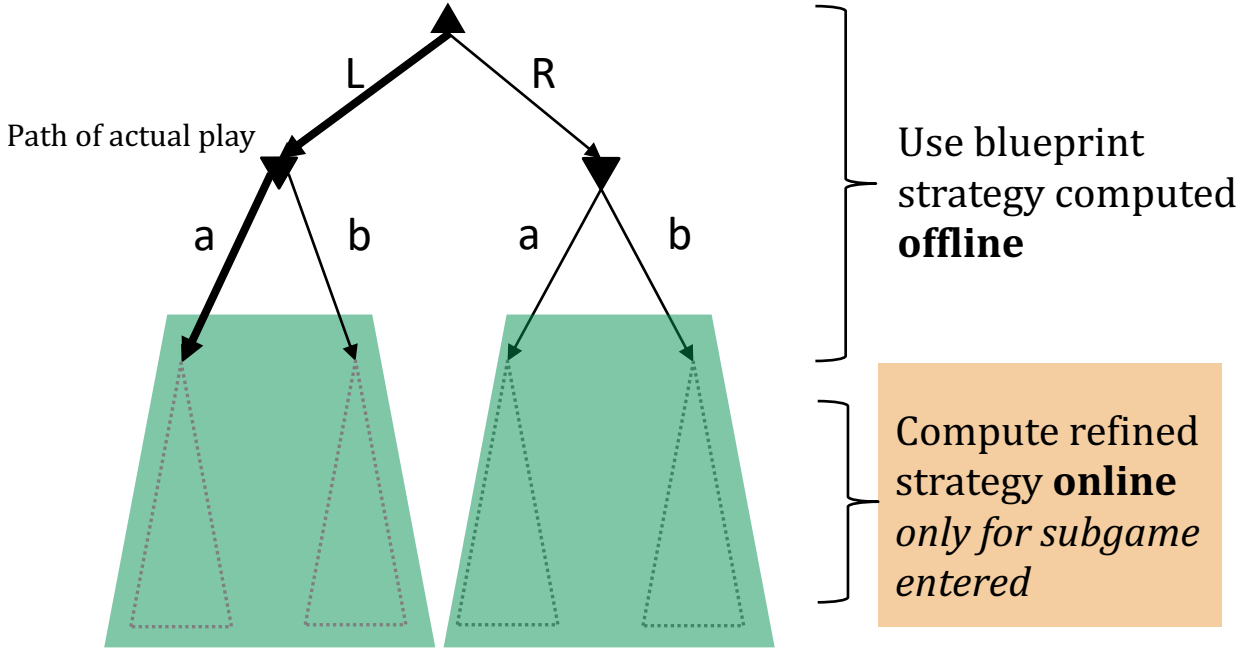


Figure 6: An example of a gadget game in Maxmargin refinement. P_1 picks the initial infoset she wishes to enter S_r in. Chance then picks the particular node of the infoset, and play then proceeds identically to the augmented subgame, except all P_1 payoffs are shifted by the size of the alternative payoff and the alternative payoff is then removed from the augmented subgame.

Implementation can be very tricky

Reach subgame solving/maxmargin

Intuition: So far, only considered a single subgame at any point.

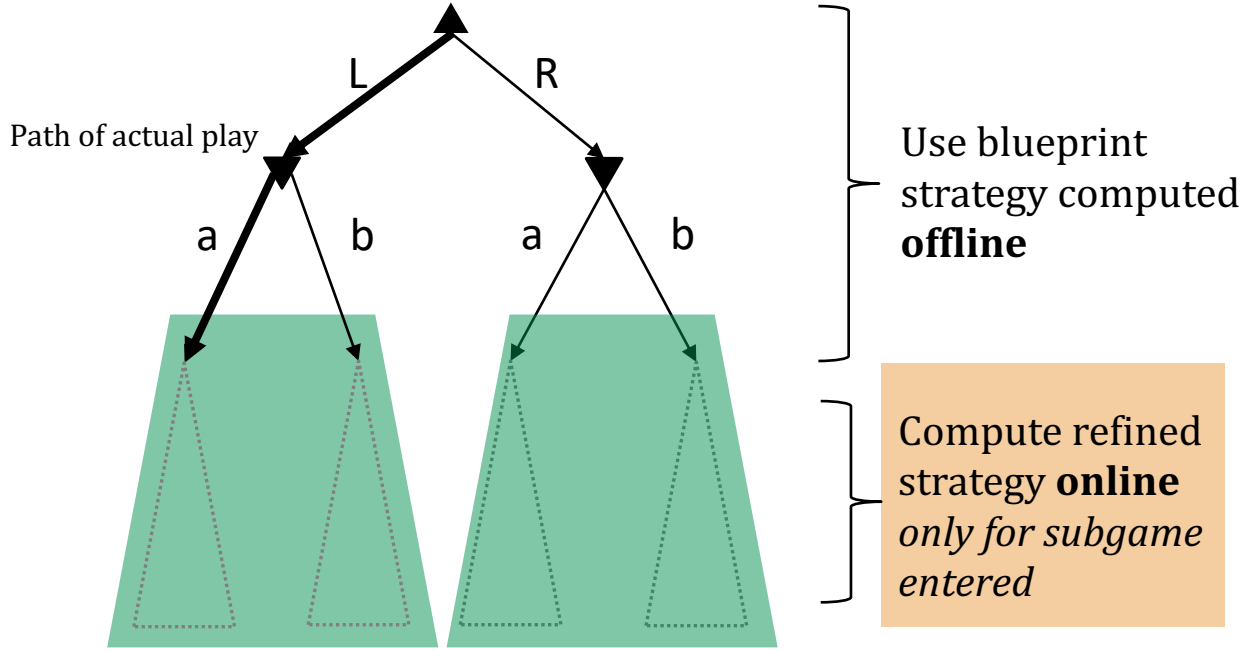


Not covered due to lack of time

In reality, many subgames will exist!

Reach subgame solving/maxmargin

Intuition: So far, only considered a single subgame at any point.



Not covered due to lack of time

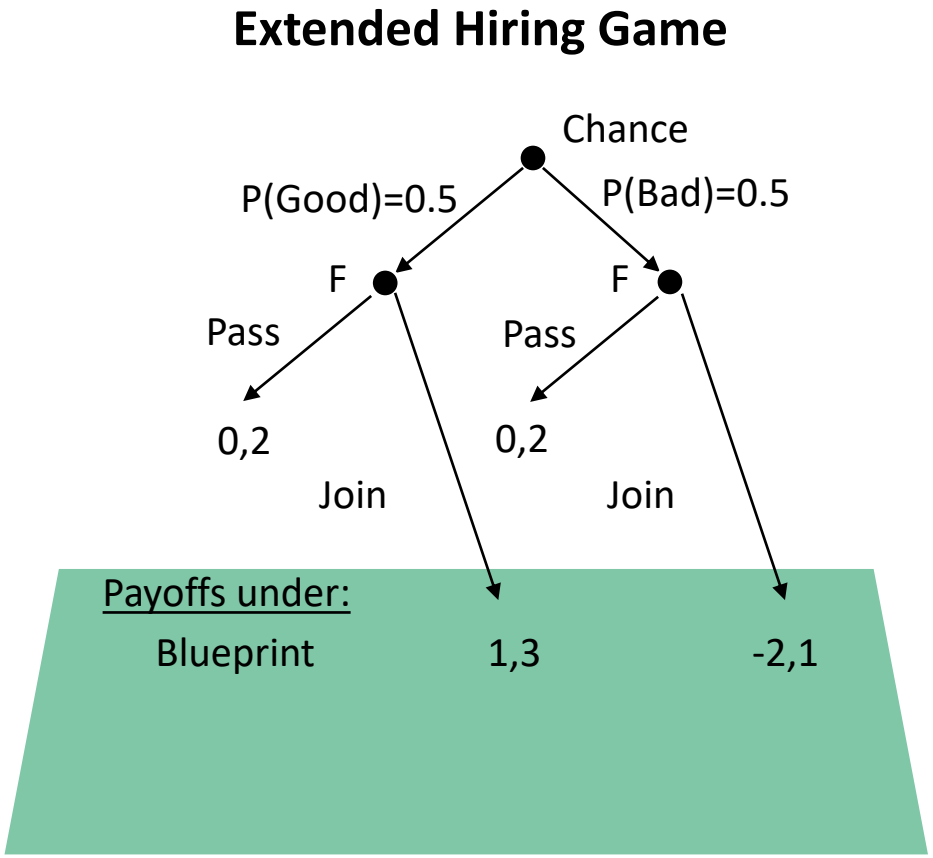
In reality, many subgames will exist!

From opponent's perspective, when finding BR, refinement is done on **all** subgames, not just one, even though from the refiner's POV, only one subgame's refinement is computed

Subgame solving for Stackelberg eqm

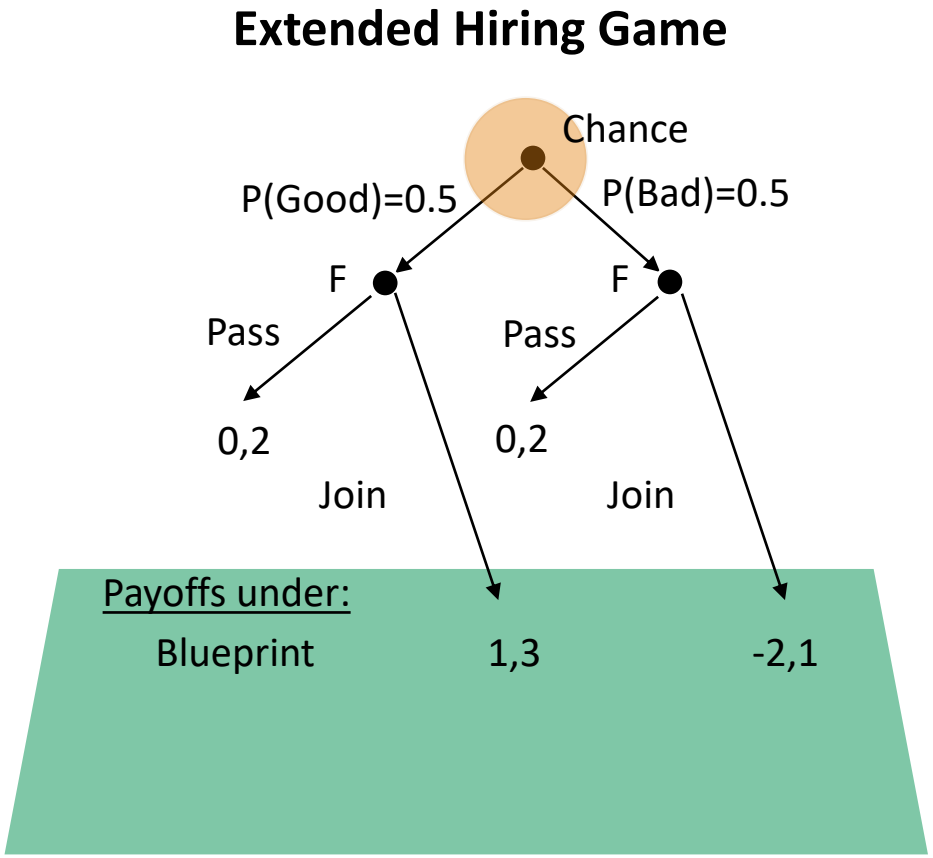
[Optional]

Carrots and Sticks in Resolving



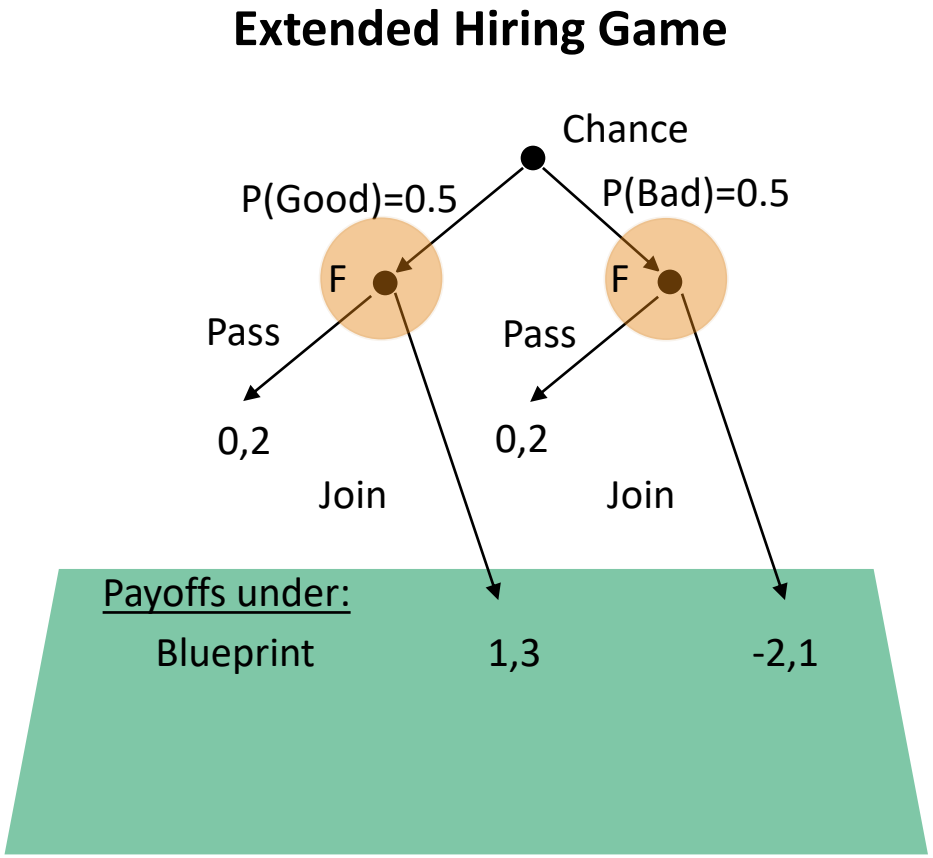
Follower: Applicants applying for a job
Leader: Hiring committee
Game: How should leader conduct the hiring process?

Carrots and Sticks in Resolving



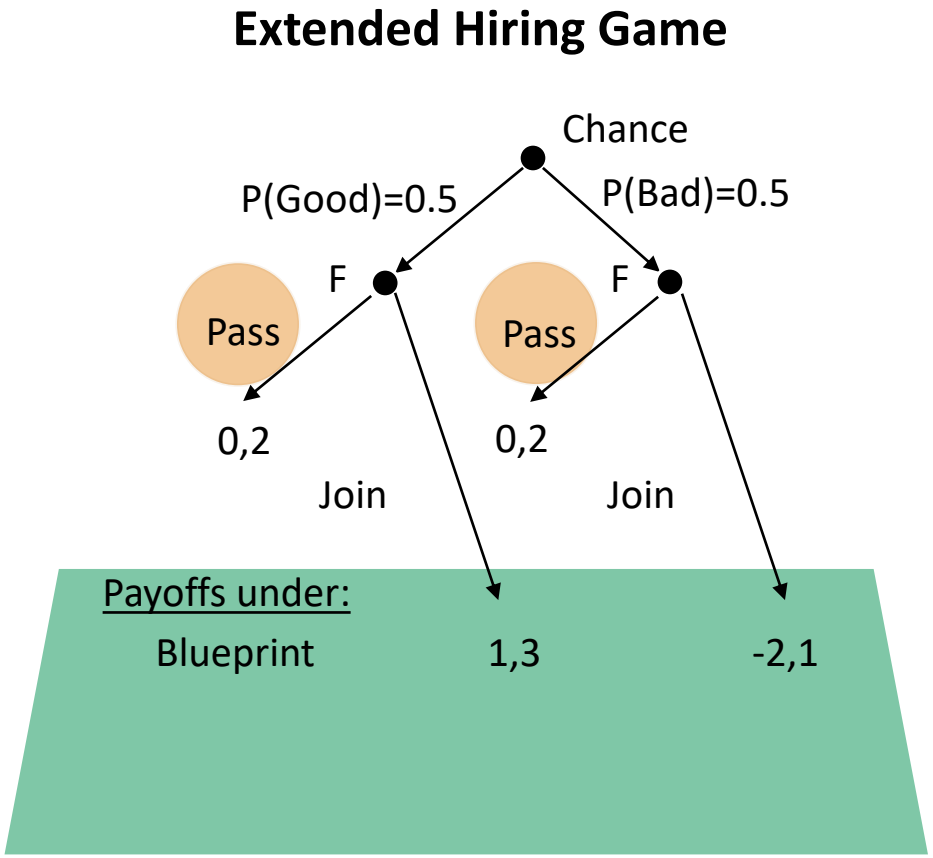
Follower: Applicants applying for a job
Leader: Hiring committee
Game: How should leader conduct the hiring process?

Carrots and Sticks in Resolving



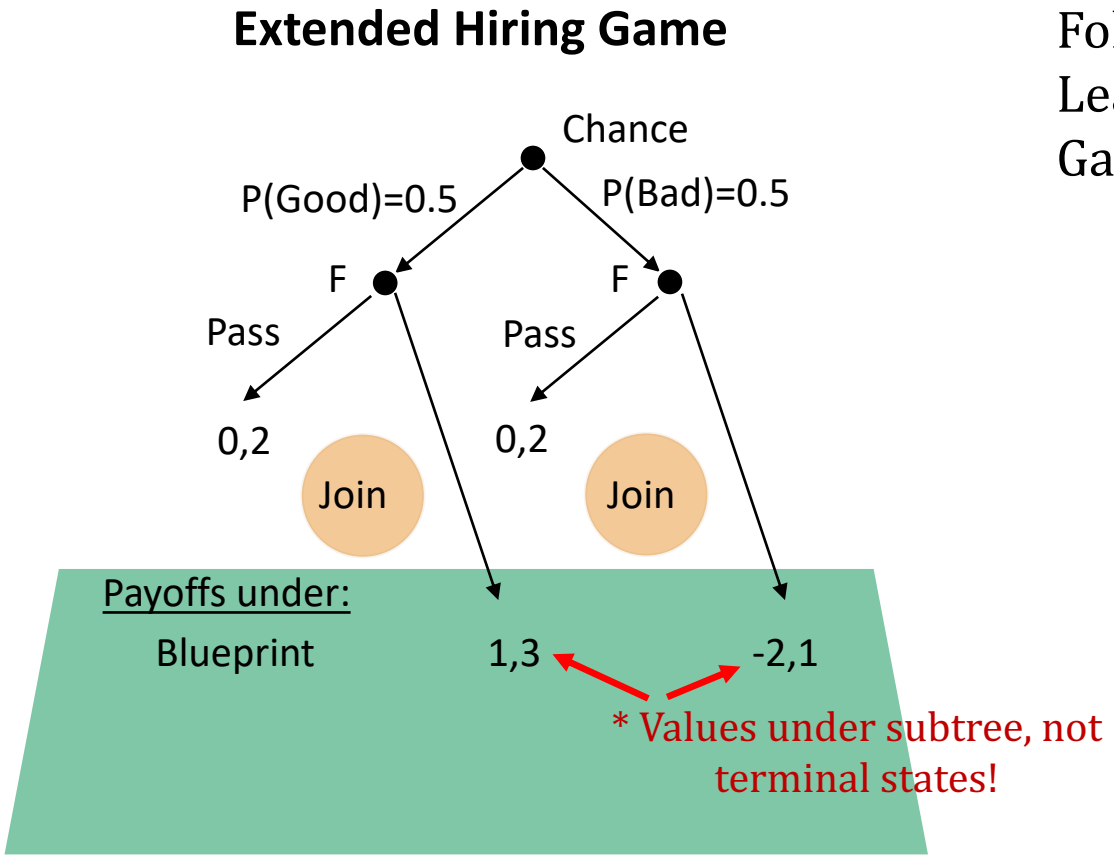
Follower: Applicants applying for a job
Leader: Hiring committee
Game: How should leader conduct the hiring process?

Carrots and Sticks in Resolving



Follower: Applicants applying for a job
Leader: Hiring committee
Game: How should leader conduct the hiring process?

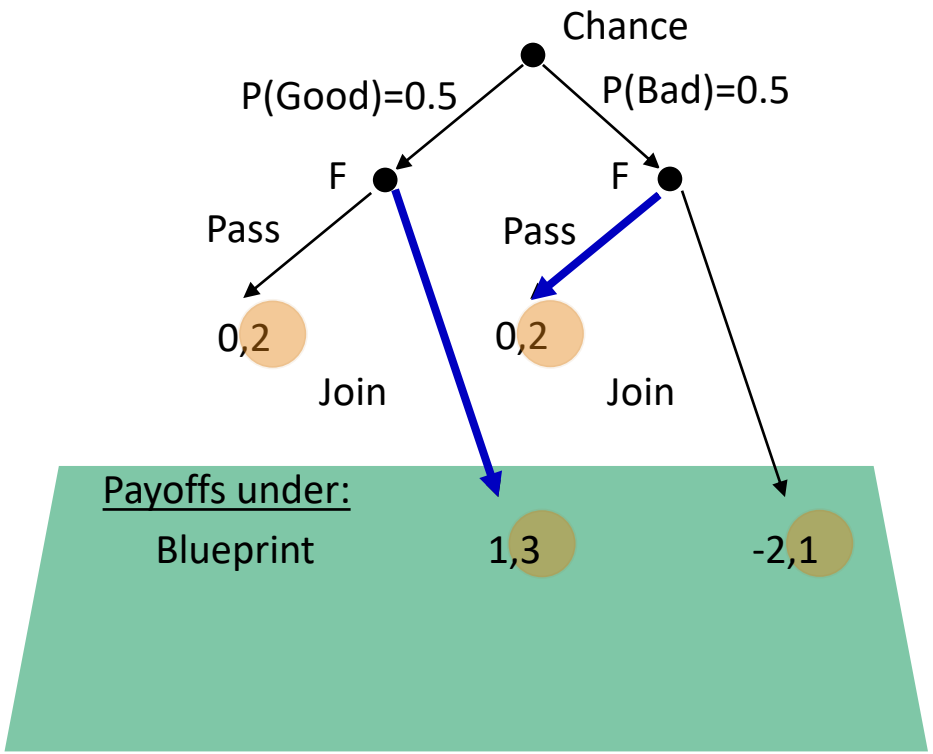
Carrots and Sticks in Resolving



Follower: Applicants applying for a job
Leader: Hiring committee
Game: How should leader conduct the hiring process?

Carrots and Sticks in Resolving

Extended Hiring Game



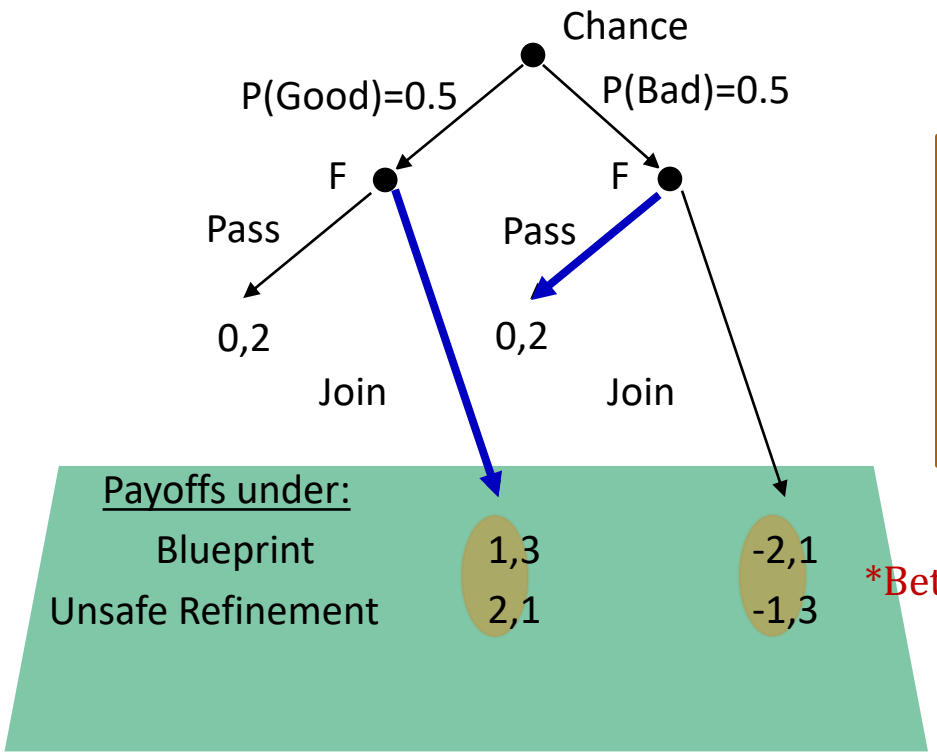
Follower: Applicants applying for a job
Leader: Hiring committee
Game: How should leader conduct the hiring process?

Blueprint

Candidates follow the blue paths.
Good candidates apply, bad candidates pass. 🥳
Leader expected payoff = **0.5**

Carrots and Sticks in Resolving

Extended Hiring Game



*Better for leader regardless

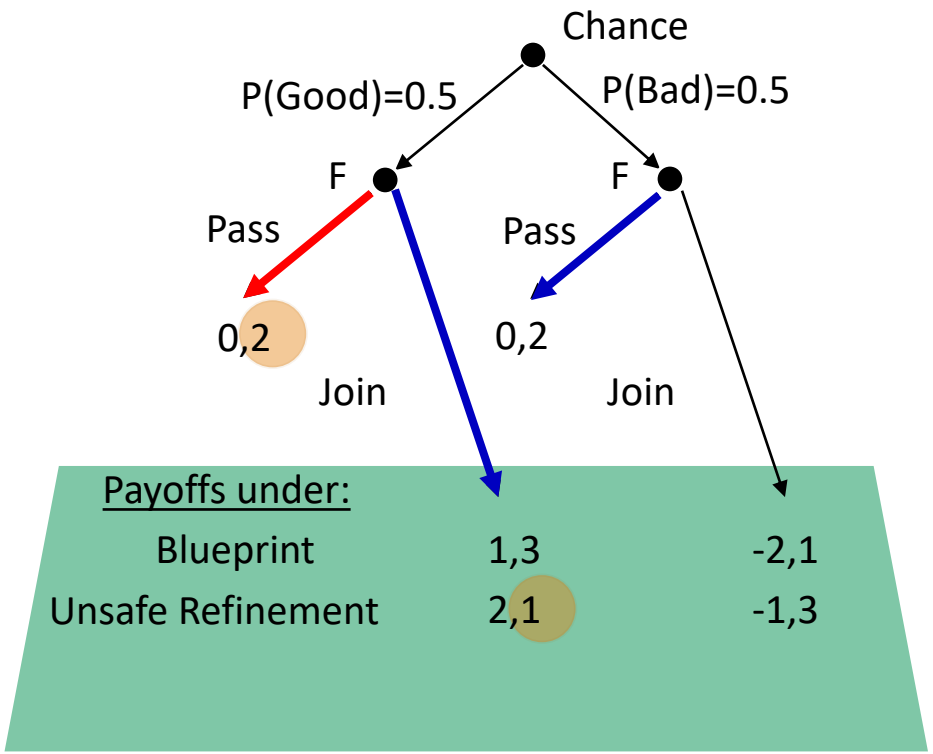
Follower: Applicants applying for a job
Leader: Hiring committee
Game: How should leader conduct the hiring process?

Blueprint

Candidates follow the blue paths.
Good candidates apply, bad candidates pass.
Leader expected payoff = **0.5**

Carrots and Sticks in Resolving

Extended Hiring Game



Follower: Applicants applying for a job
Leader: Hiring committee
Game: How should leader conduct the hiring process?

Blueprint

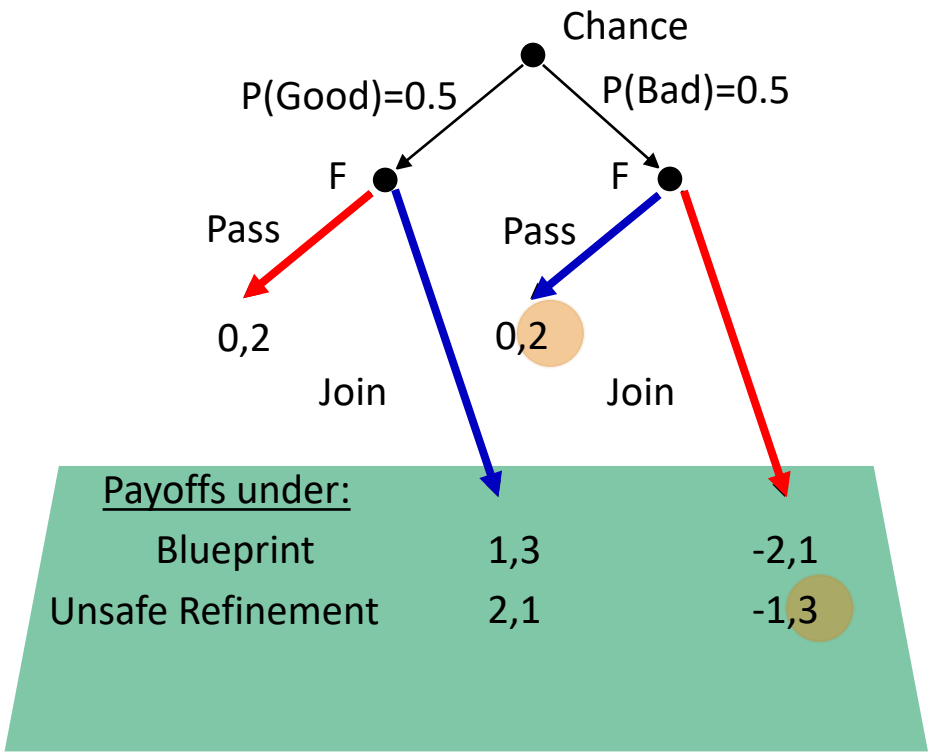
Candidates follow the blue paths.
Good candidates apply, bad candidates pass.
Leader expected payoff = **0.5**

Naïve Refinement

Candidates follow the red paths.
Good candidates pass, bad candidates apply.
Leader expected payoff = **-0.5**

Carrots and Sticks in Resolving

Extended Hiring Game



Follower: Applicants applying for a job
Leader: Hiring committee
Game: How should leader conduct the hiring process?

Blueprint

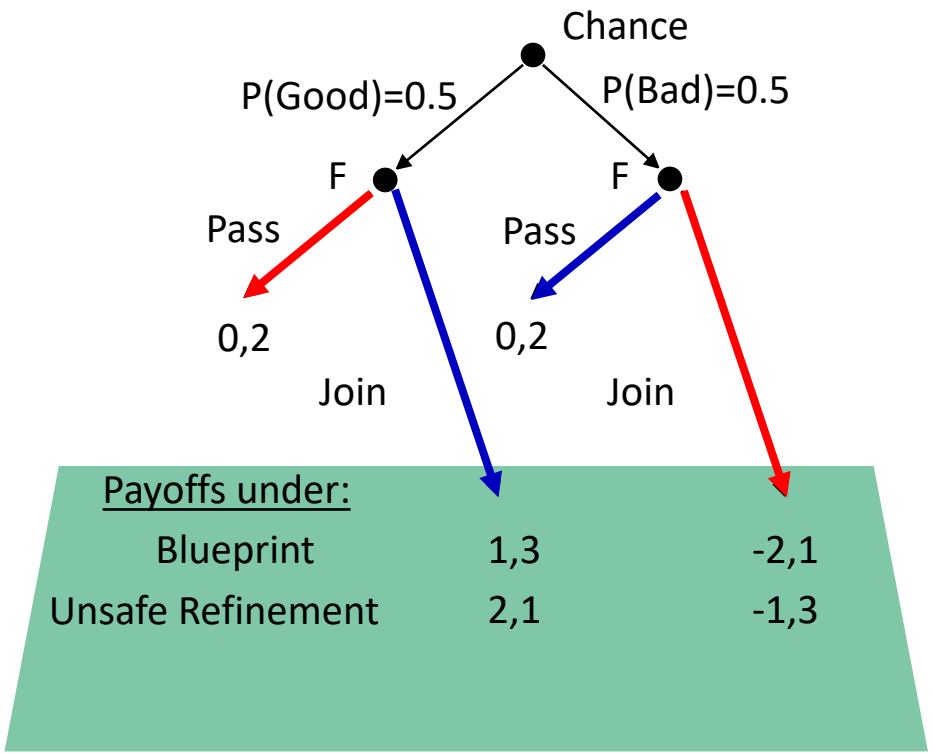
Candidates follow the blue paths.
Good candidates apply, bad candidates pass.
Leader expected payoff = **0.5**

Naïve Refinement

Candidates follow the red paths.
Good candidates pass, bad candidates apply.
Leader expected payoff = **-0.5**

Carrots and Sticks in Resolving

Extended Hiring Game



Follower: Applicants applying for a job
Leader: Hiring committee
Game: How should leader conduct the hiring process?

Blueprint

Candidates follow the blue paths.
Good candidates apply, bad candidates pass.
Leader expected payoff = **0.5**

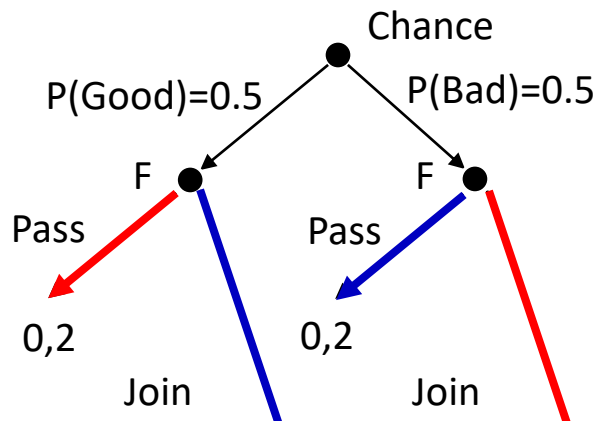
Naïve Refinement

Candidates follow the red paths.
Good candidates pass, bad candidates apply.
Leader expected payoff = **-0.5**

> UNSAFE

Carrots and Sticks in Resolving

Extended Hiring Game



Payoffs under:		
Blueprint	1,3	-2,1
Unsafe Refinement	2,1	-1,3
Safe Bounds	, ≥ 2	, ≤ 2

Follower: Applicants applying for a job
Leader: Hiring committee
Game: How should leader conduct the hiring process?

Blueprint

Candidates follow the blue paths.
Good candidates apply, bad candidates pass.
Leader expected payoff = **0.5**

Naïve Refinement

Candidates follow the red paths.
Good candidates pass, bad candidates apply.
Leader expected payoff = **-0.5**

> UNSAFE

Safe Bounds

Candidates follow the blue paths
Blueprint obeys bounds ⇒ expected payoff: at least that of blueprint!

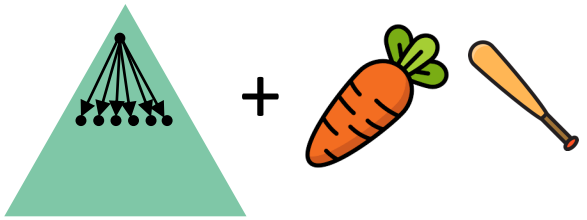
Our algorithm

Step 1

Compute safe value bounds for follower information sets

Step 2 (only for subgame)

Solve



Apply naïve refinement with the enforcement of bounds

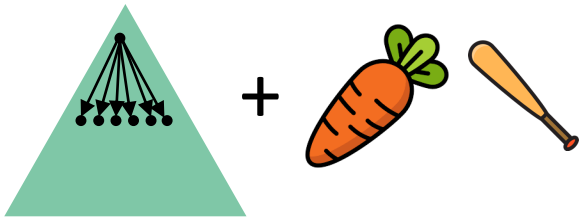
Our algorithm

Step 1

Compute safe value bounds for follower information sets

Step 2 (only for subgame)

Solve



Apply naïve refinement with the enforcement of bounds

Not a standard Stackelberg problem

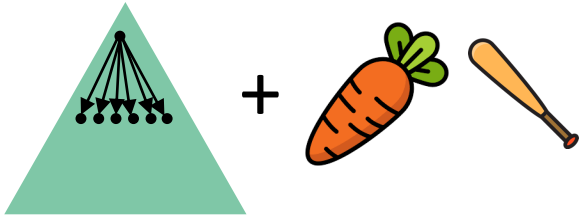
Our algorithm

Step 1

Compute safe value bounds for follower information sets

Step 2 (only for subgame)

Solve



Apply naïve refinement with the enforcement of bounds

Not a standard Stackelberg problem

Step 2 can be solved by

- Mathematical programming (integer linear program)
- Transform into another Stackelberg problem using **gadget game**