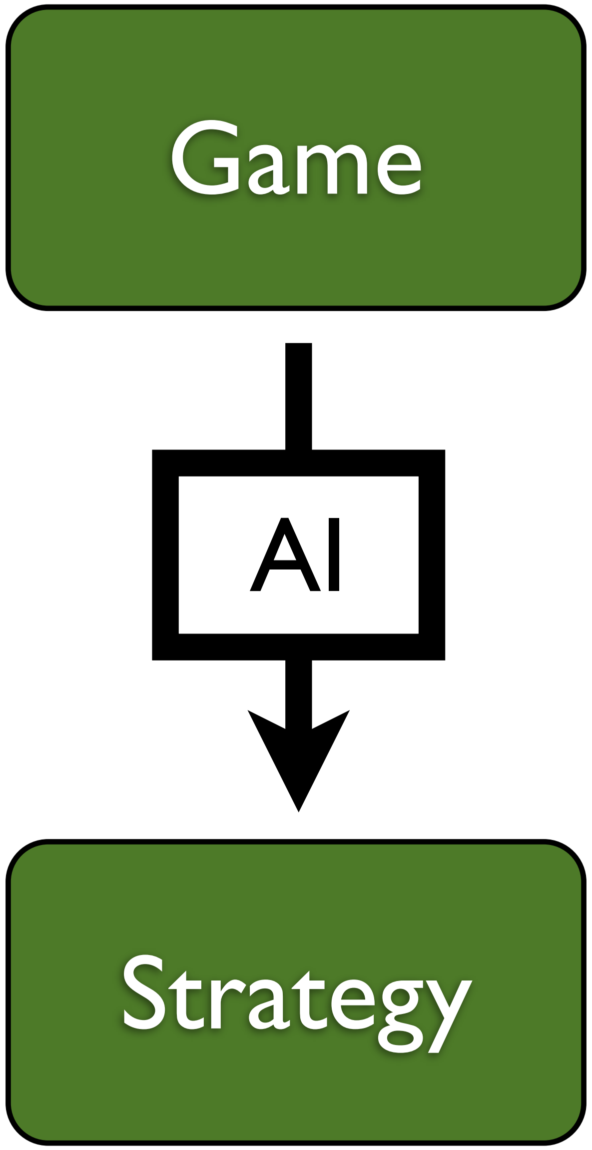
Accelerating Best Response Calculation in Large Extensive Games

University of Alberta
Computer Poker Research Group

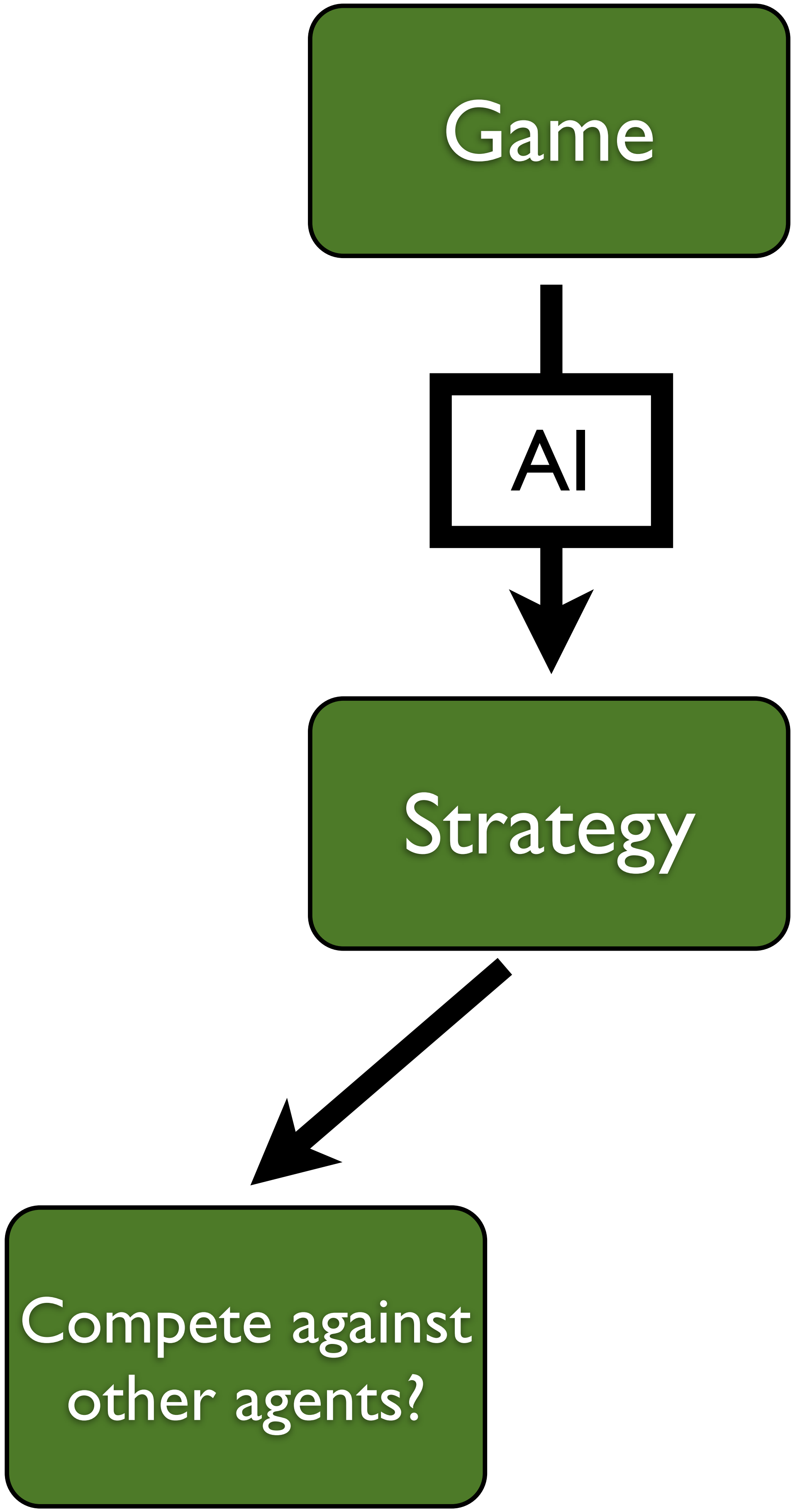
July 21, 2011

Michael Johanson, Kevin Waugh,
Michael Bowling, Martin Zinkevich



Suppose you have
a 2-player game.

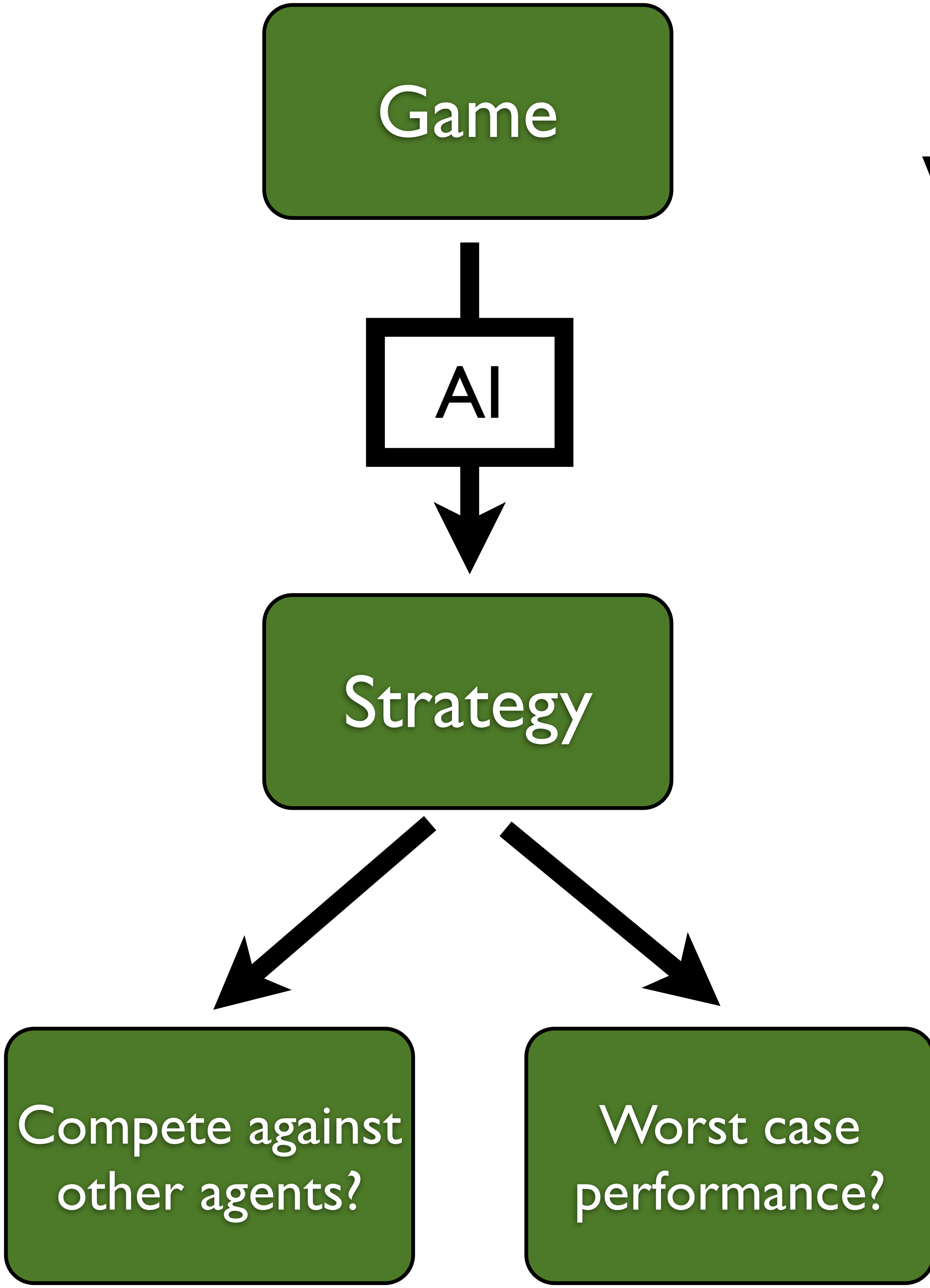
You can use an
algorithm for learning
a strategy in this space.



There are several ways to evaluate a strategy.

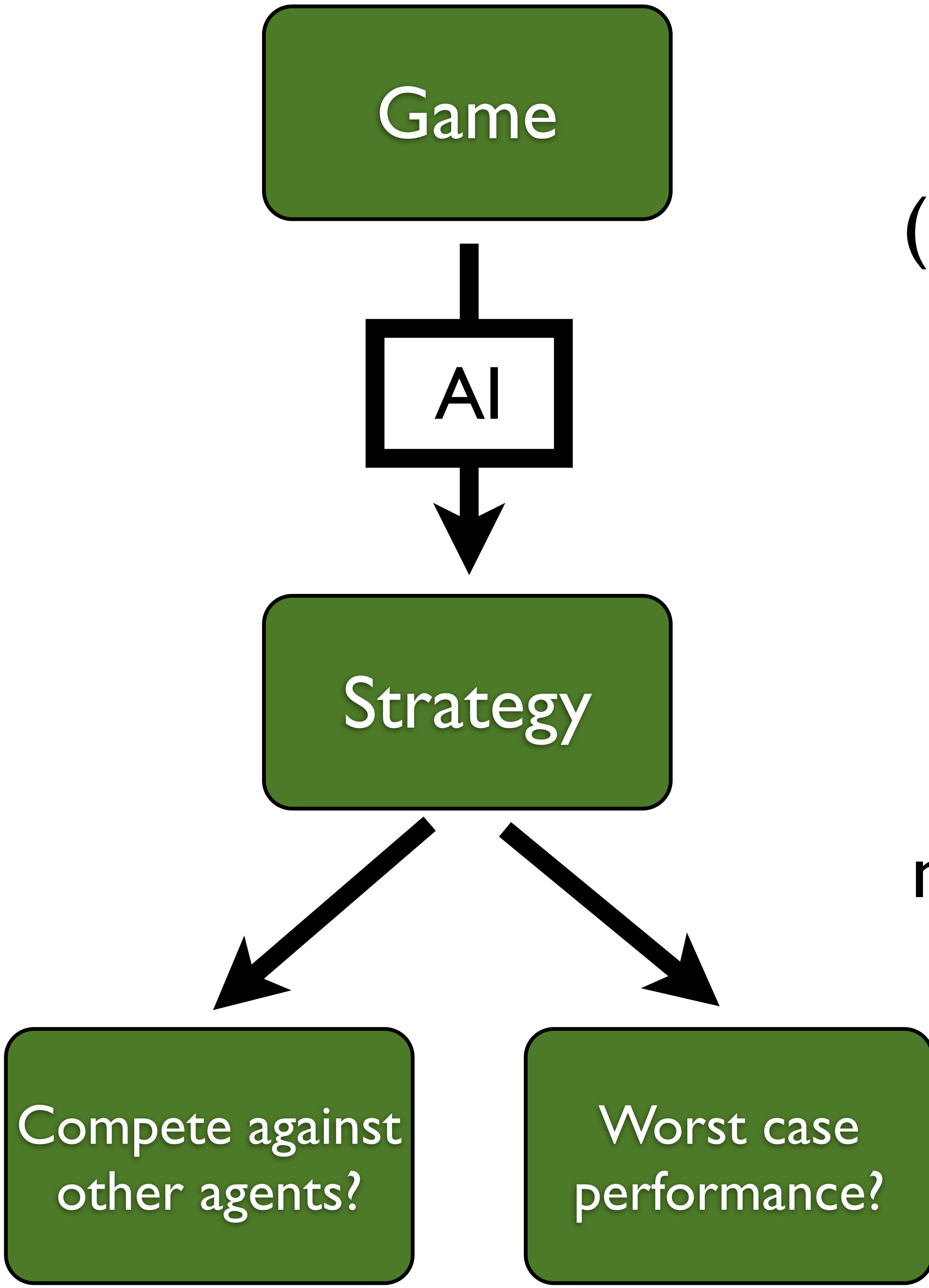
You could run a competition against other agents.

- ♥ Computer Poker Competition
- ♣ RoboCup
- ♦ Computer Olympiad
- ♠ Trading Agent Competition



Worst-case performance is another useful metric.

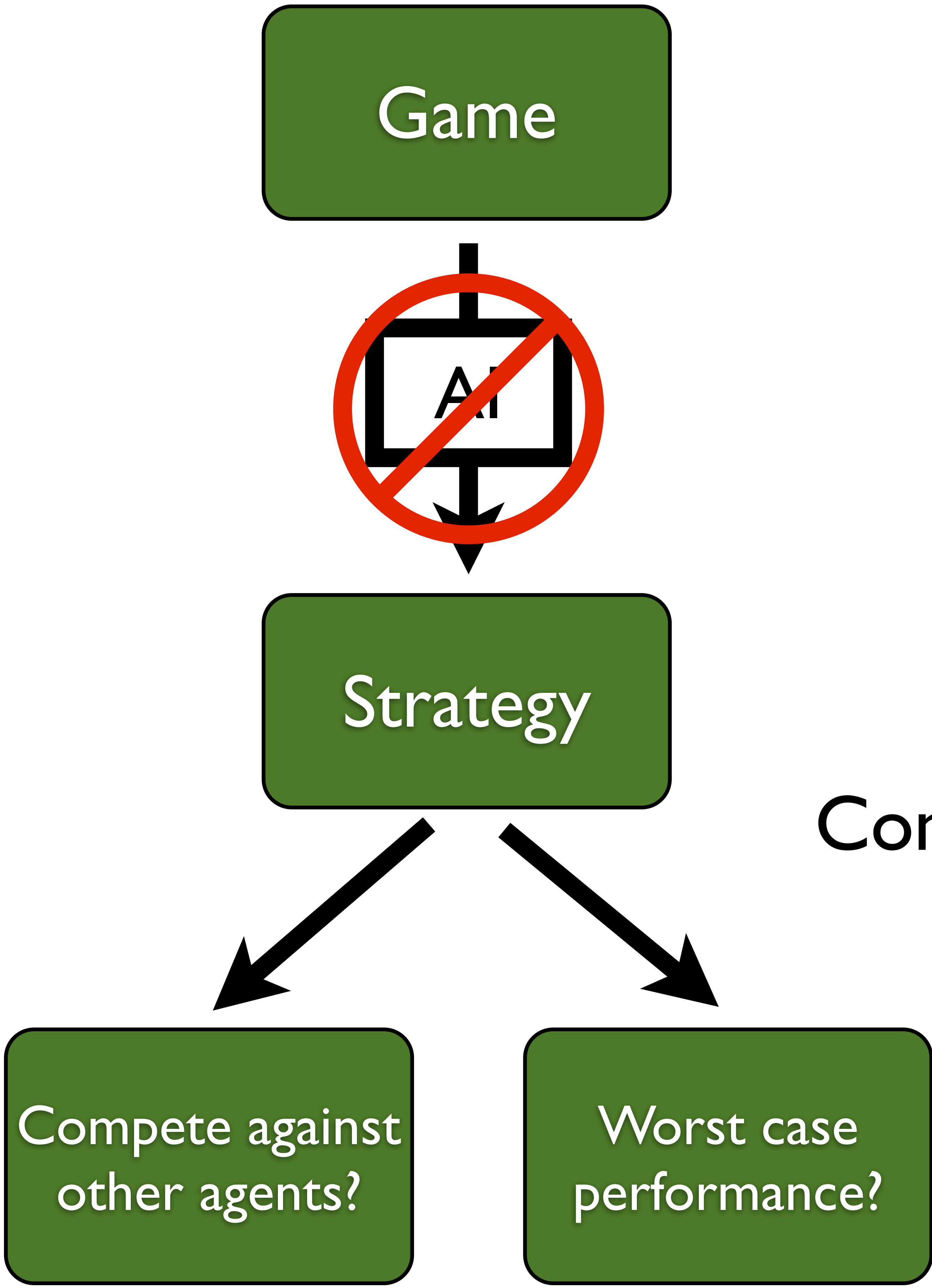
2-player games:
Use Expectimax to find a best-response counterstrategy.



Optimal Strategy
(2-player, zero-sum game):

Nash Equilibrium,
maximize worst-case
performance.

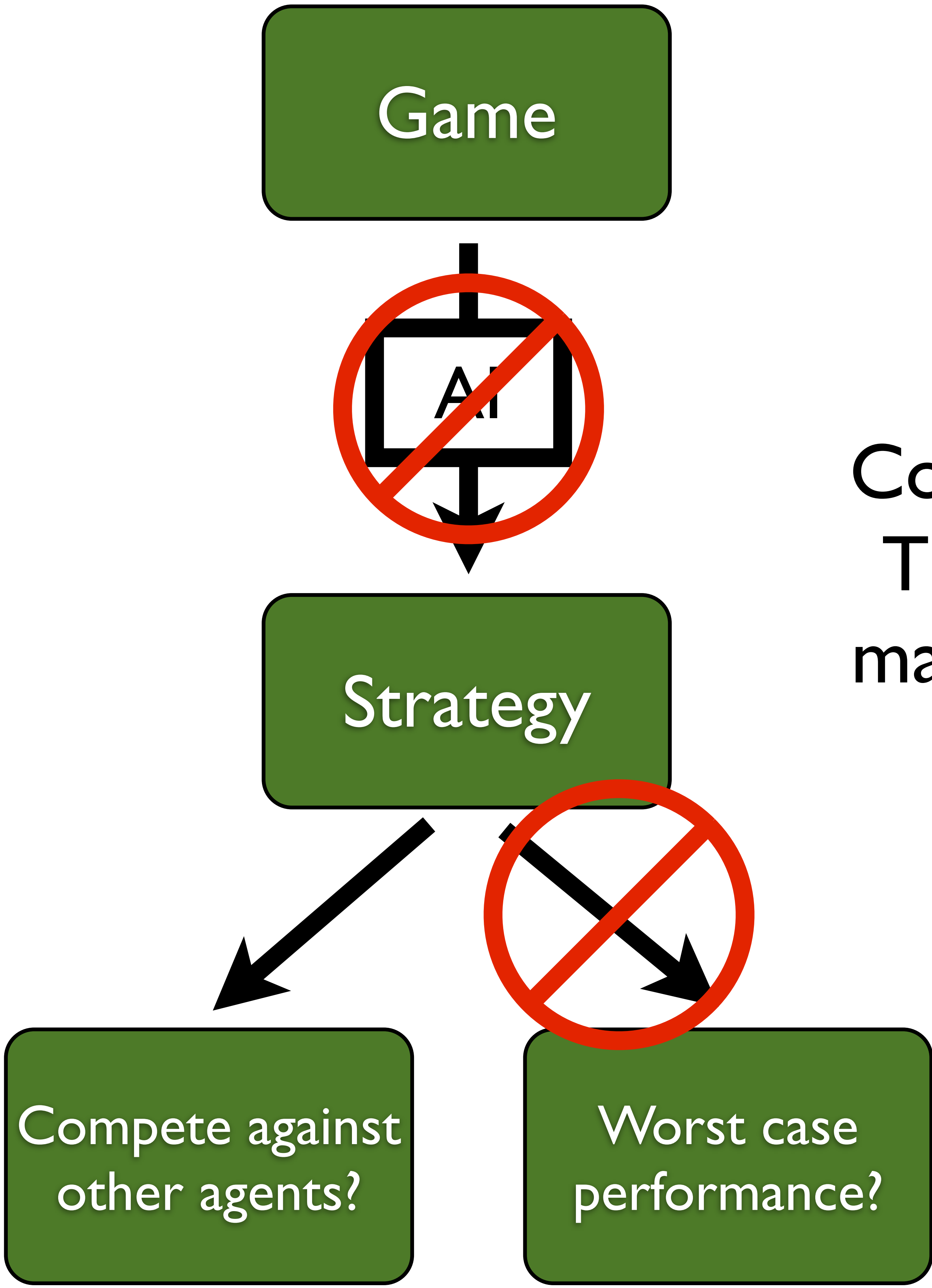
Or, equivalently,
minimize worst-case loss.



2-player Limit
Texas Hold'em Poker:

~10^{18} game states
~10^{14} information sets
(decision points)

Computing an optimal strategy:
4 PB of RAM,
1400 cpu-years

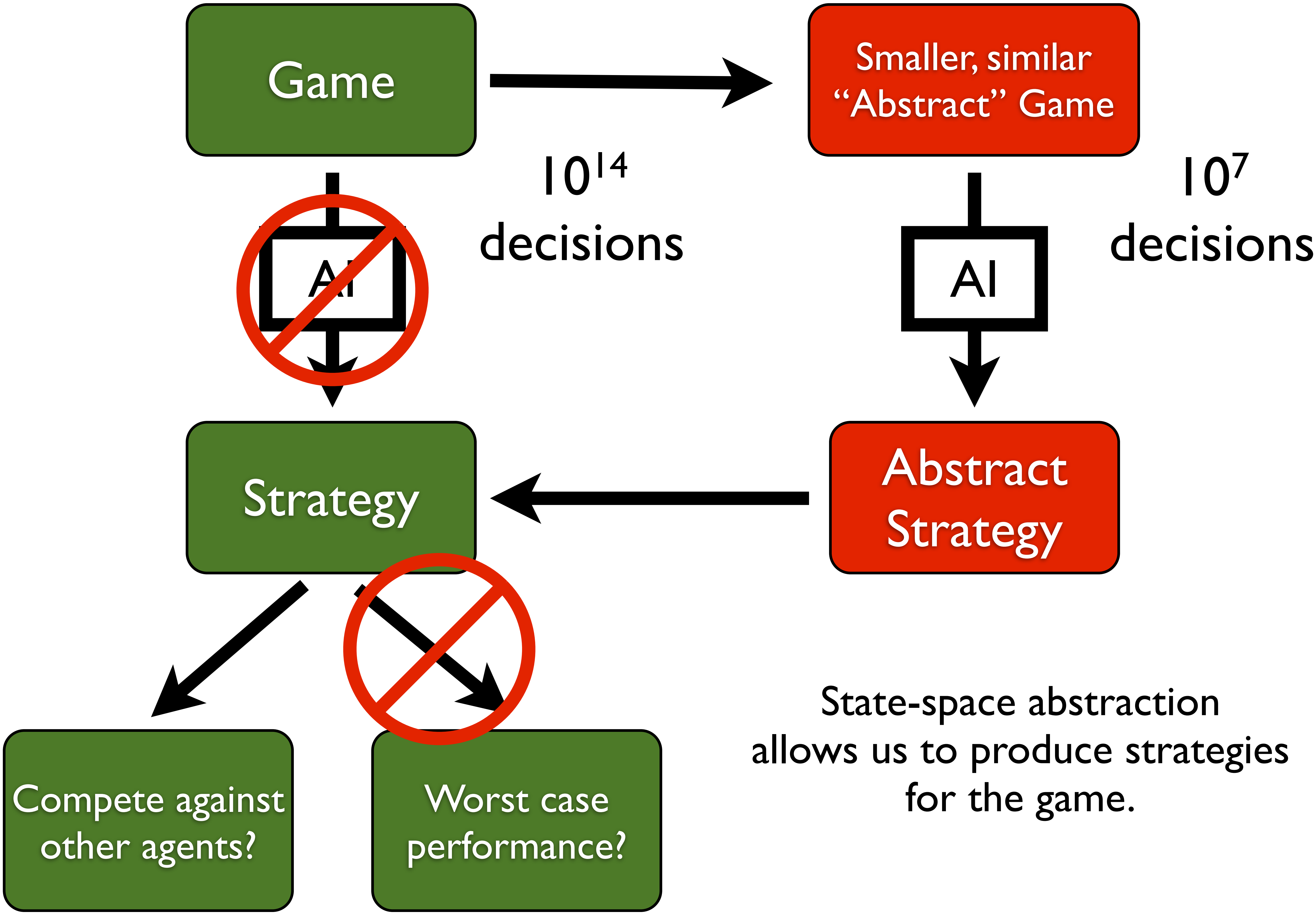


2-player Limit
Texas Hold'em Poker:

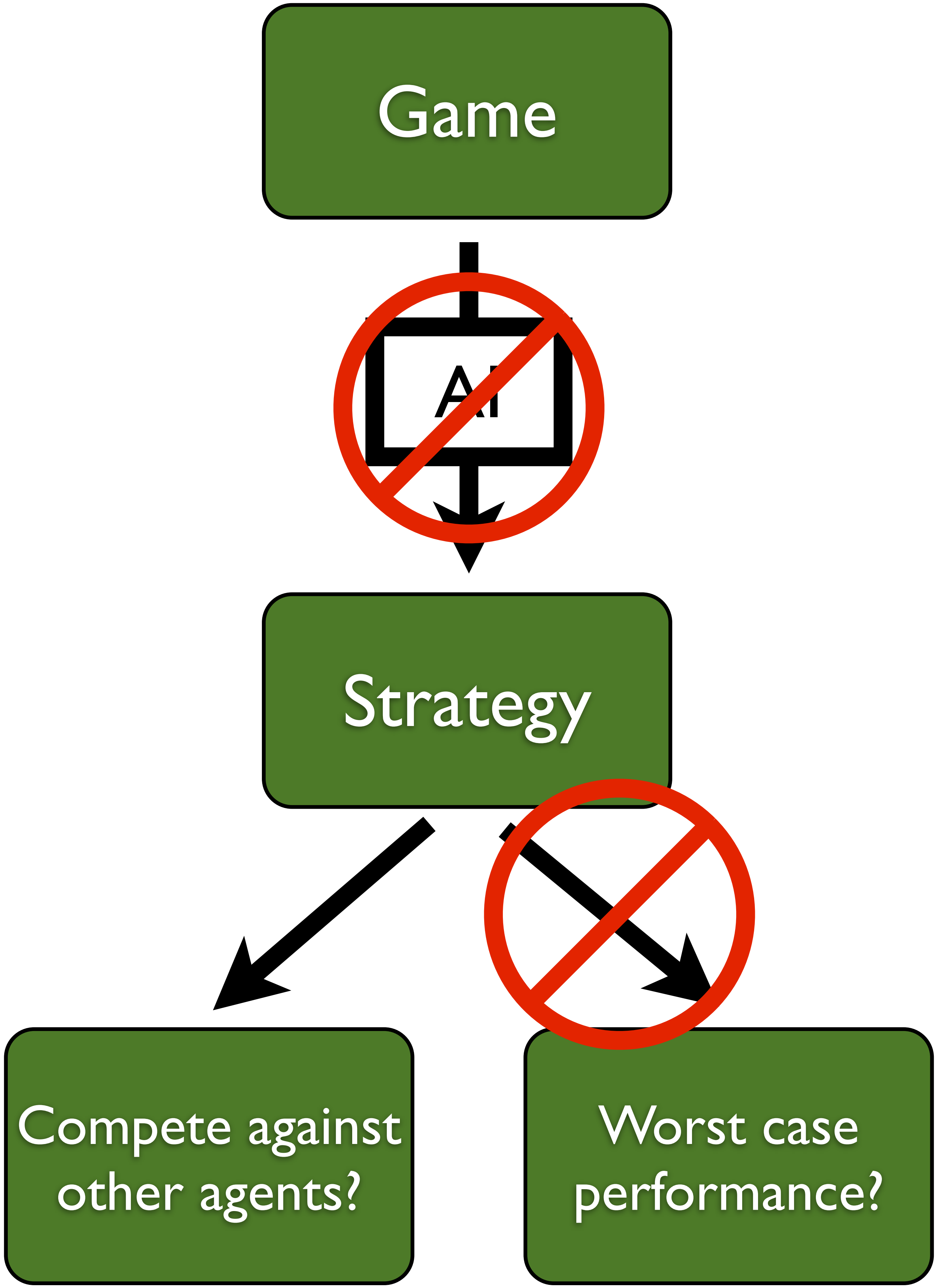
~10^{18} game states

Computing a best response:
Thought to be intractable,
may require a full game tree
traversal.

At 3 billion states/sec,
would take 10 years.



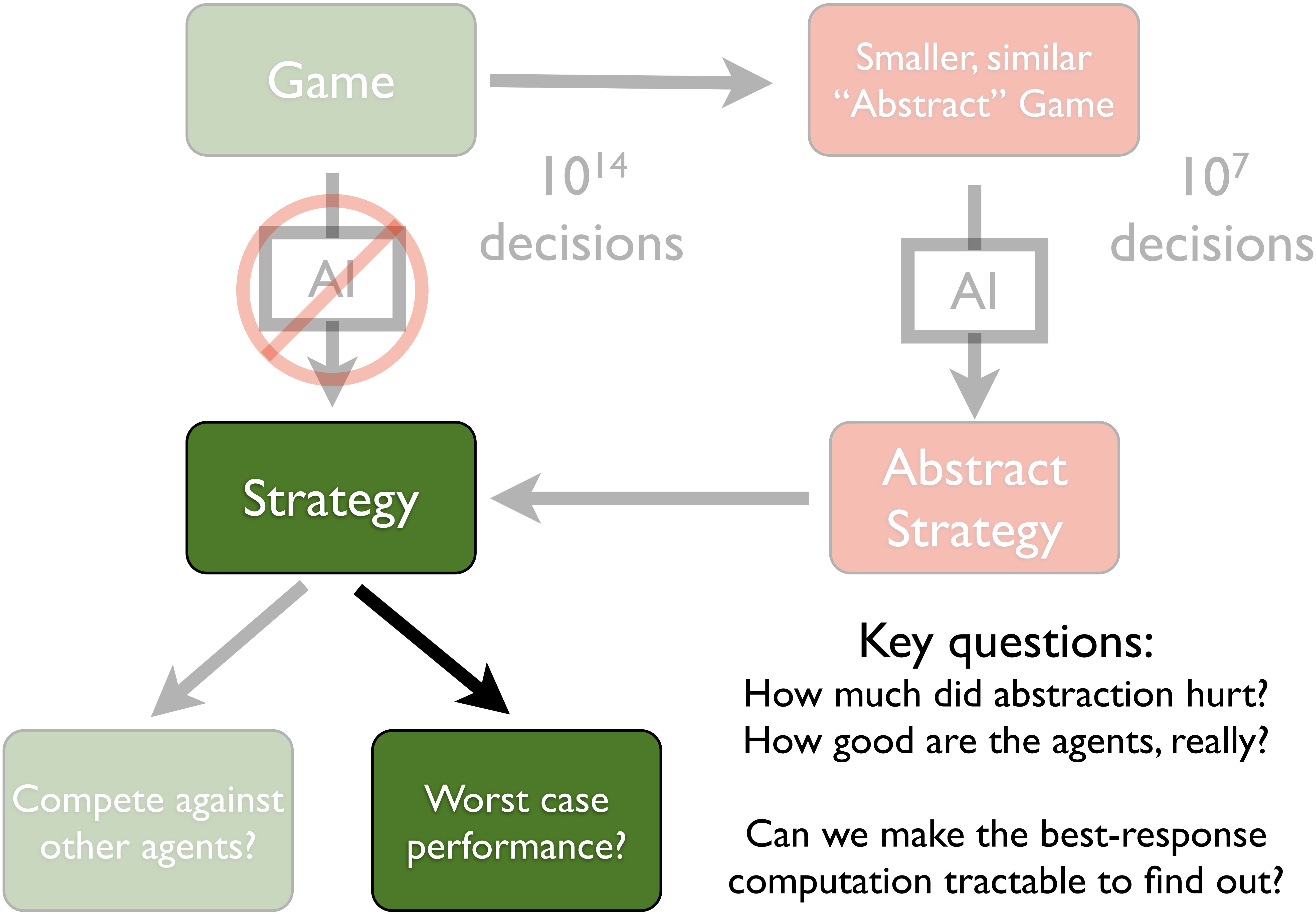
State-space abstraction
allows us to produce strategies
for the game.



Evaluation has relied on tournaments.

AAAI 2007 - Phil Laak
First Man-Machine
Poker Championship

Annual Computer
Poker Competition:
2006-2011
(at AAAI next month!)



Key questions:

How much did abstraction hurt?
How good are the agents, really?

Can we make the best-response computation tractable to find out?

Accelerating Best Response Computation

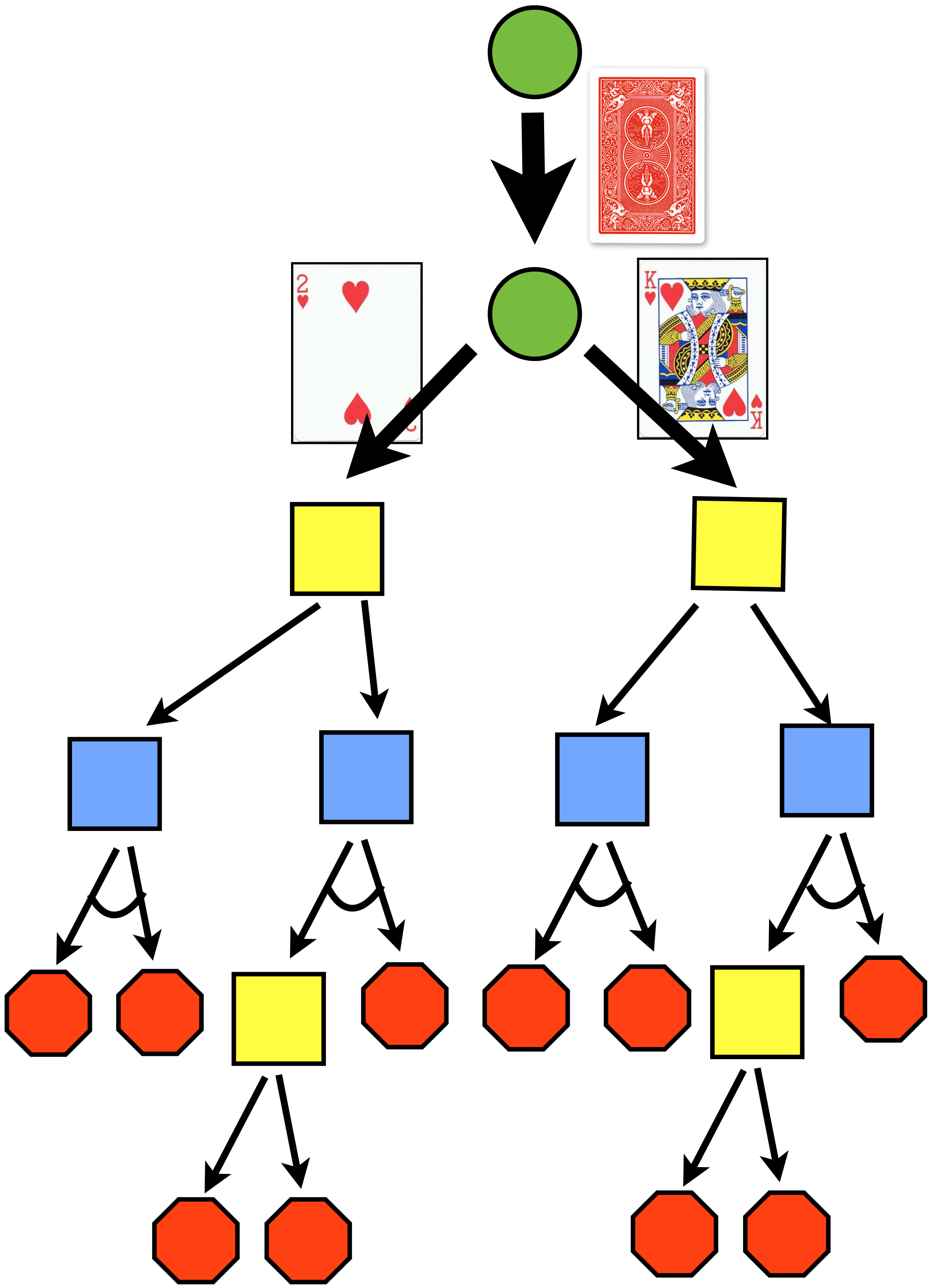
- ♥ Four ways to speed up best response calculation in imperfect information games
- ♣ Formerly intractable computations are now run in one day
- ♦ Solving an 8 year old evaluation problem
- ♠ How good are state-of-the-art computer poker programs?

Expectimax Search

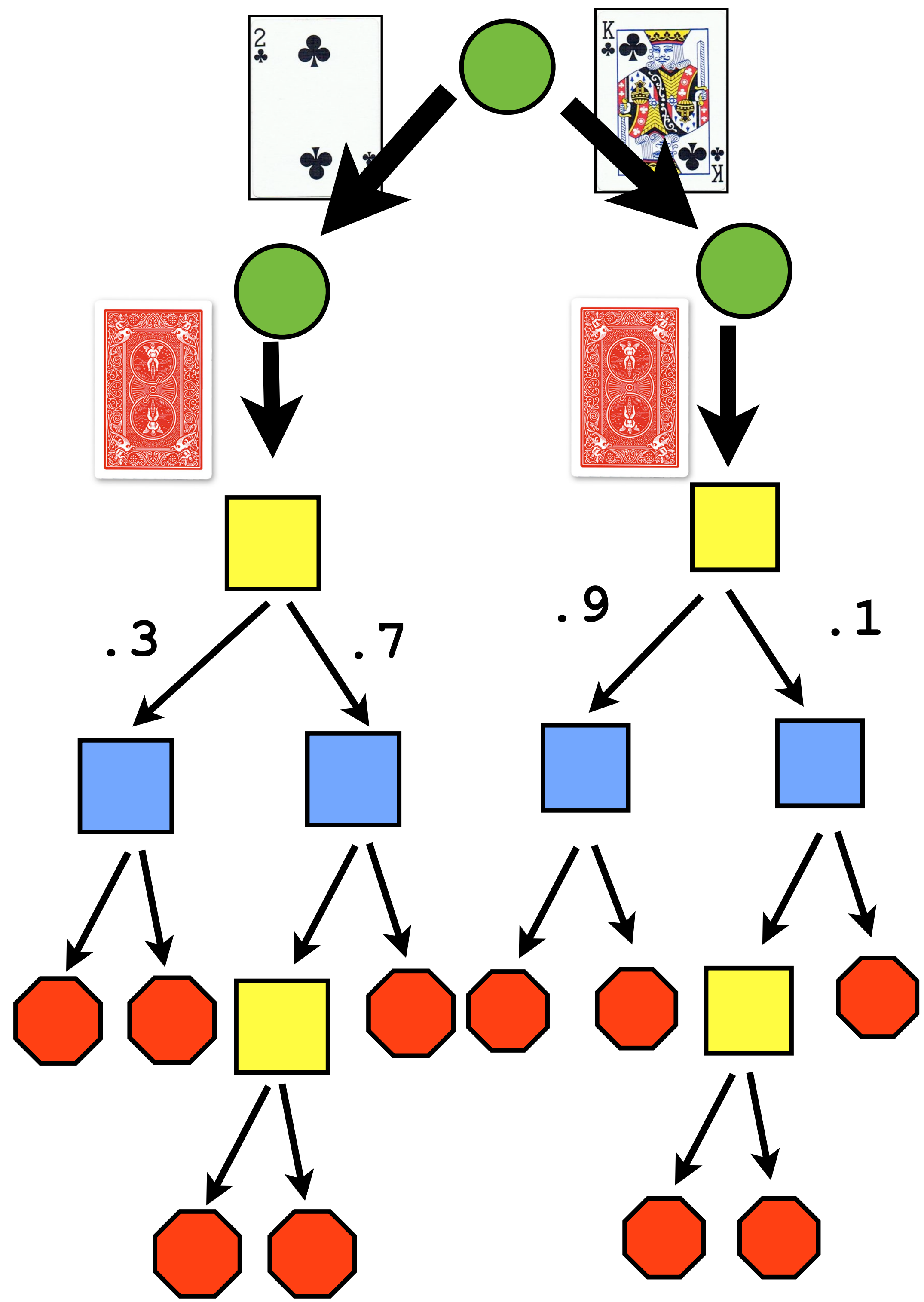
The Best Response Task:

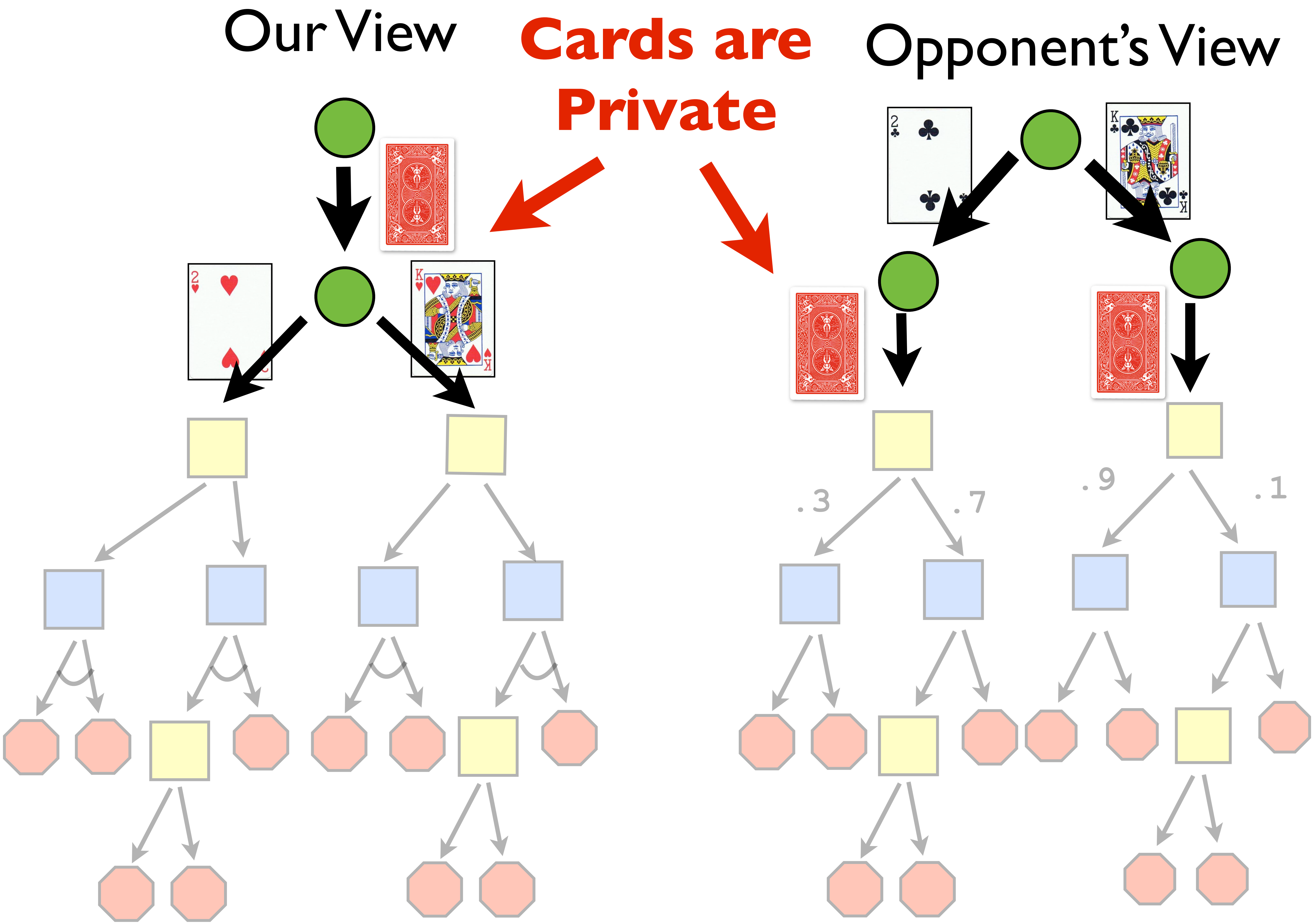
Given an opponent's entire strategy, choose actions to maximize our expected value.

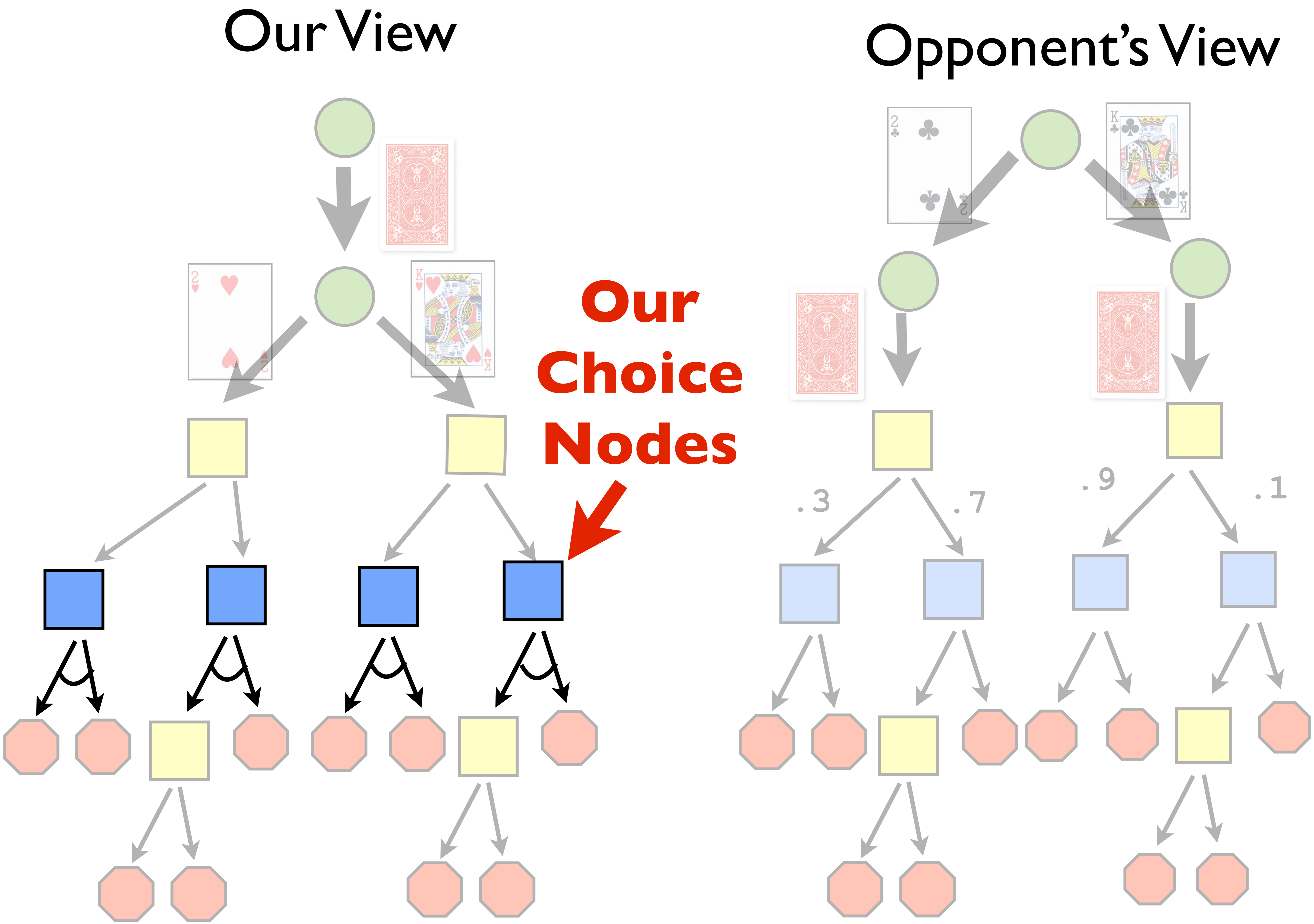
Our View

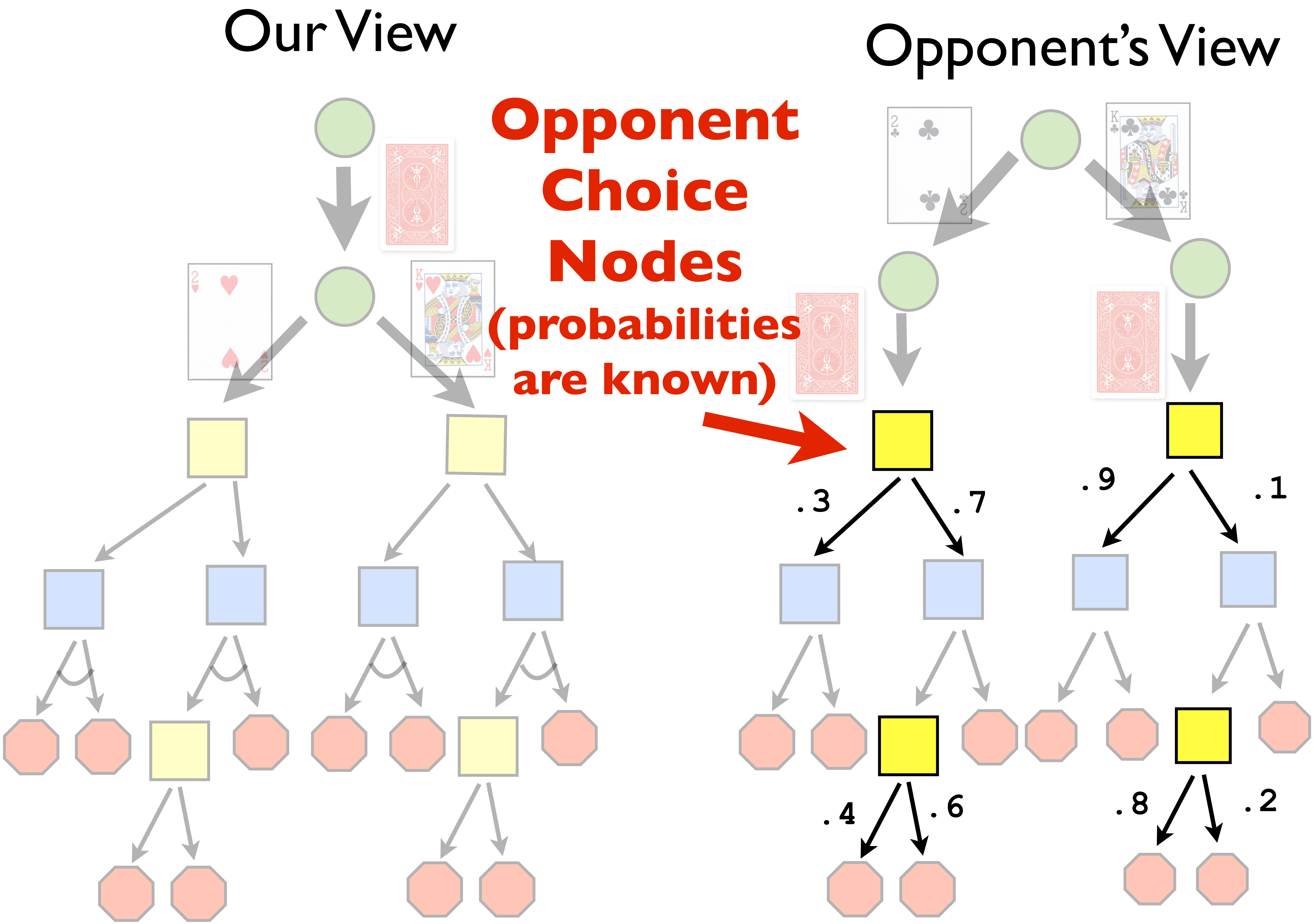


Opponent's View



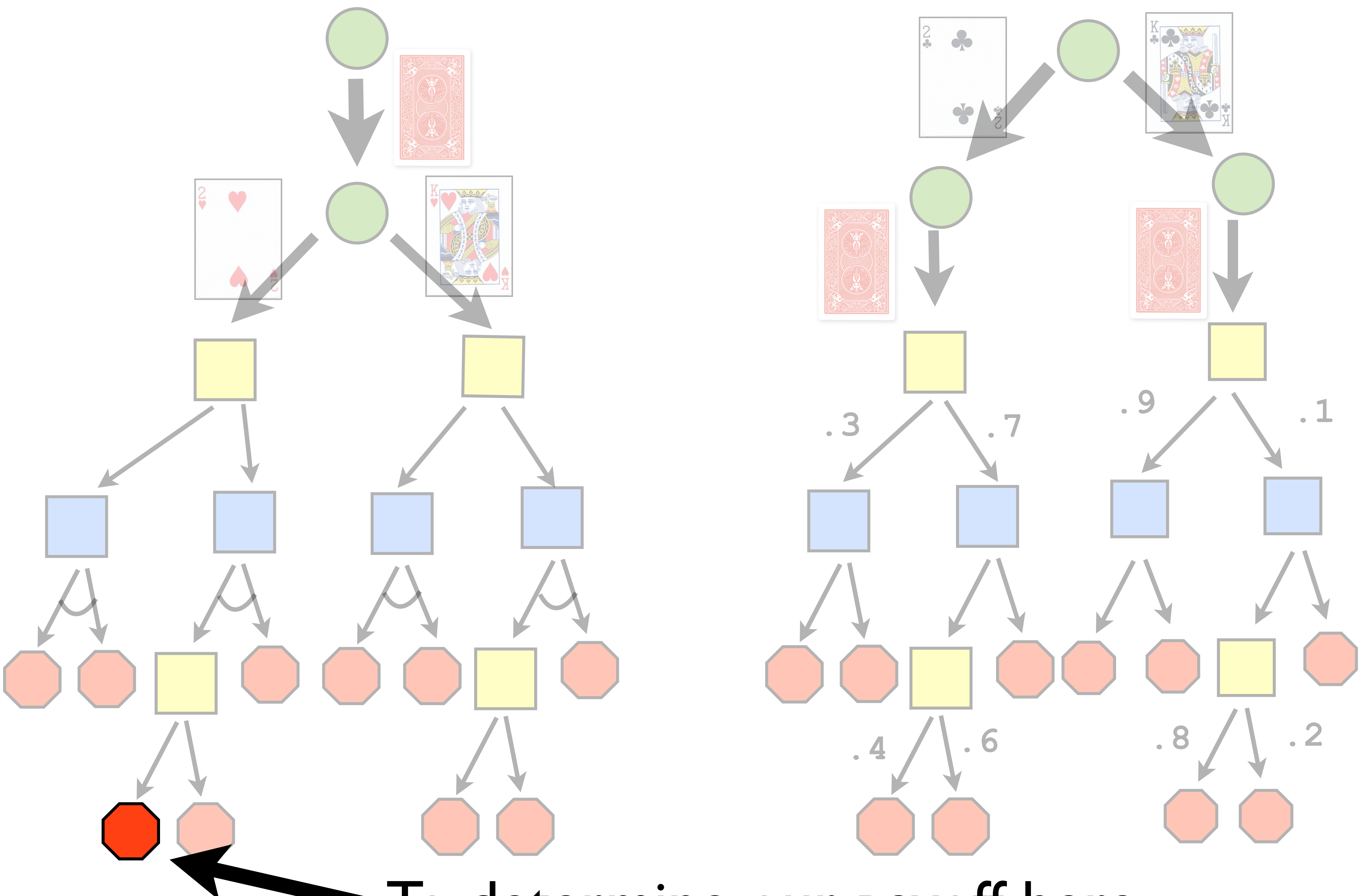

Our View
Cards are Private
Opponent's View
.3
.7
.9
.1


Our View
Opponent's View
Our Choice Nodes
.3
.7
.9
.1


Our View
Opponent's View
Opponent Choice Nodes (probabilities are known)
.3
.7
.9
.1
.4
.6
.8
.2

Our Tree

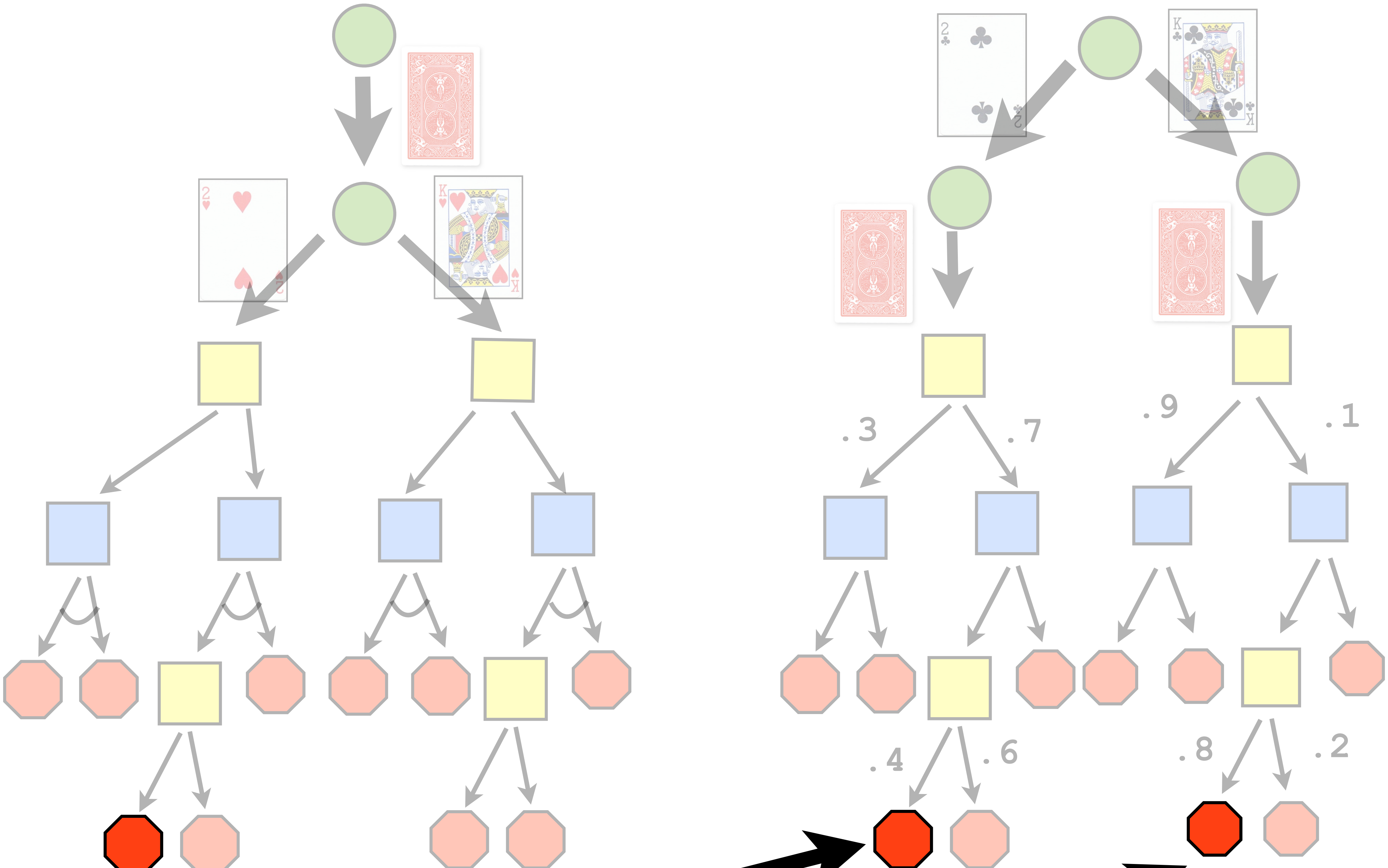
Opponent's Tree



To determine our payoff here...

Our Tree

Opponent's Tree



..we need to compute the distribution over these states

Expectimax Search

Simple recursive tree walk:

♥ Pass forward:
  Probability of opponent being in their private states

♣ Return:
  Expected value for our private state

Expectimax Search

Simple recursive tree walk:

- Pass forward:
  Probability of opponent being in their private states

- Return:
  Expected value for our private state

- Visits each state just once!
  But 10^{18} states is still intractable.

Accelerated Best Response

Four ways to accelerate this computation:

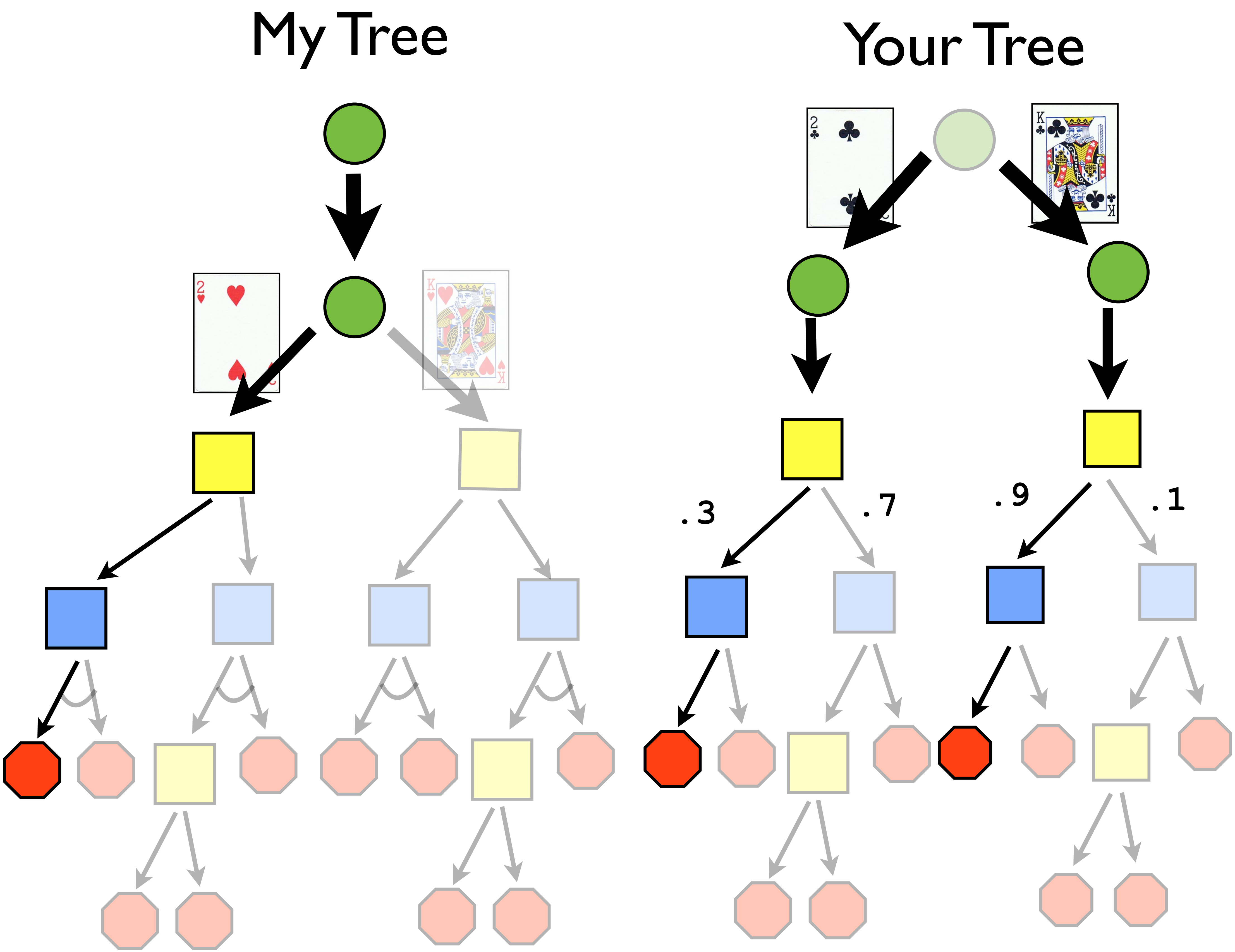
1) Take advantage of what the opponent doesn't know

2) Do O(n^2) work in O(n) time

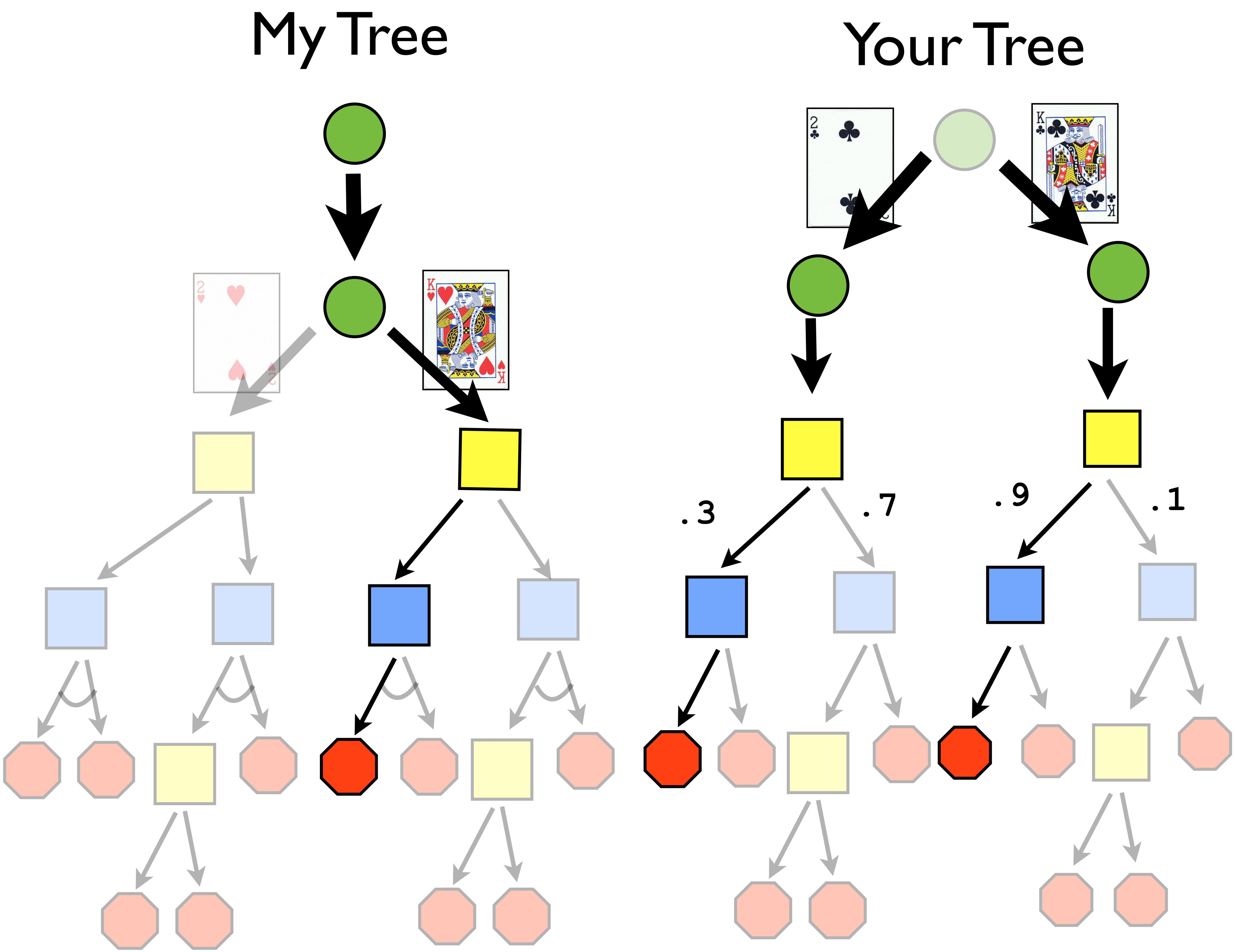
3) Avoid isomorphic game states

4) Parallel computation

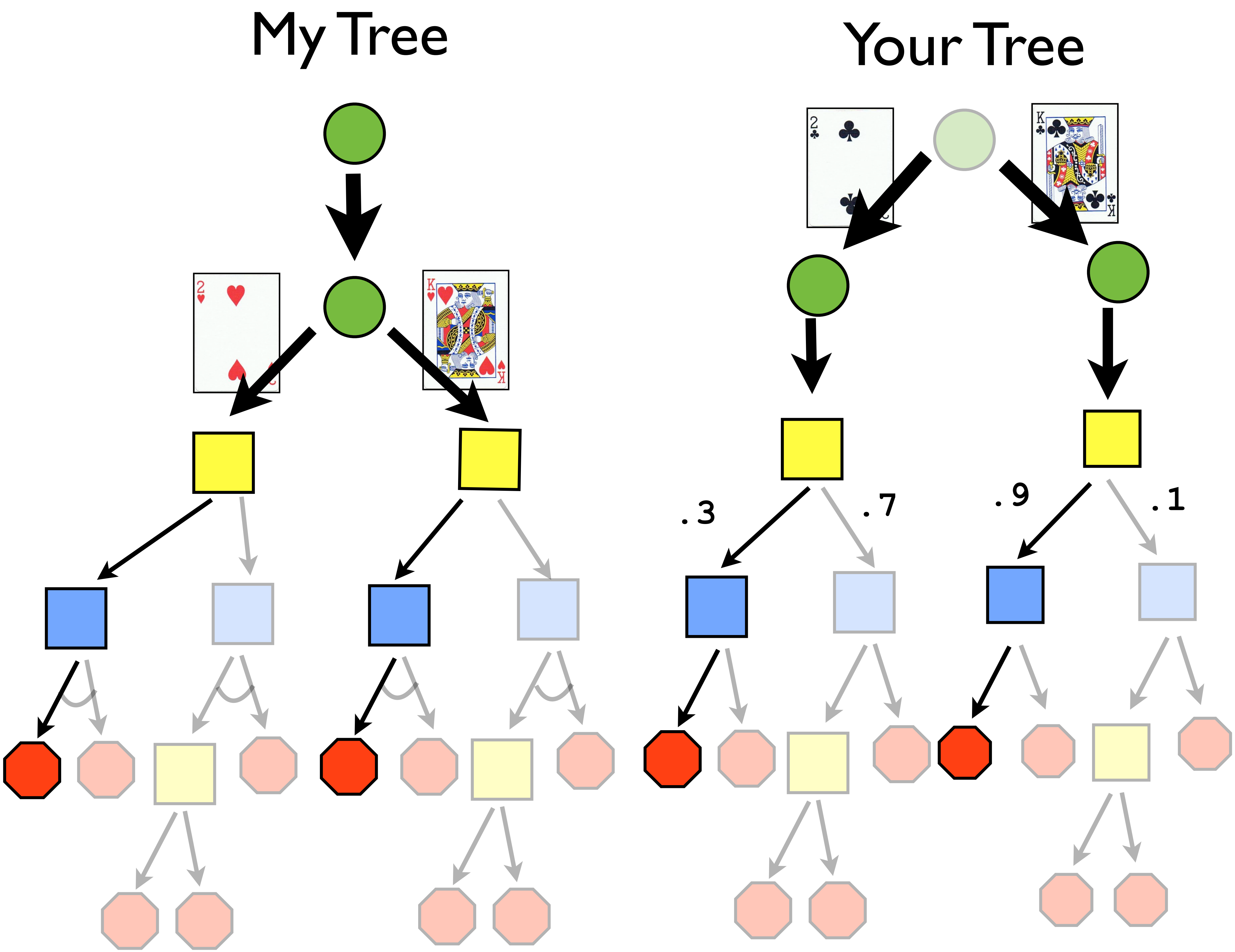
What the opponent doesn't know



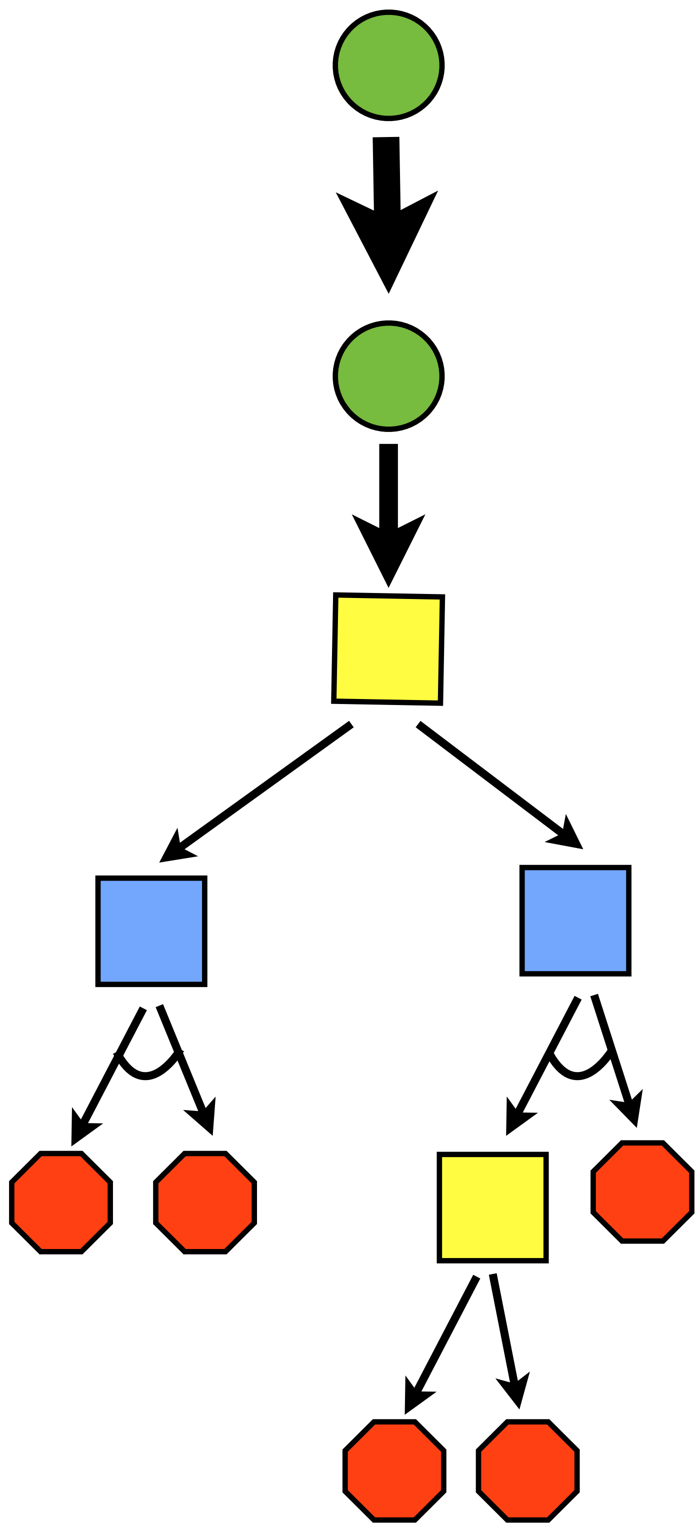
What the opponent doesn't know



What the opponent doesn't know



The Public Tree



We can instead walk this much smaller tree of public information.

At each node, we choose actions for all of the states our opponent cannot tell apart.

More work per node, but we reuse queries to the opponent's strategy!

~110x speedup in Texas hold'em

Accelerated Best Response

The new technique has four orthogonal improvements:

1) Take advantage of what the opponent doesn't know

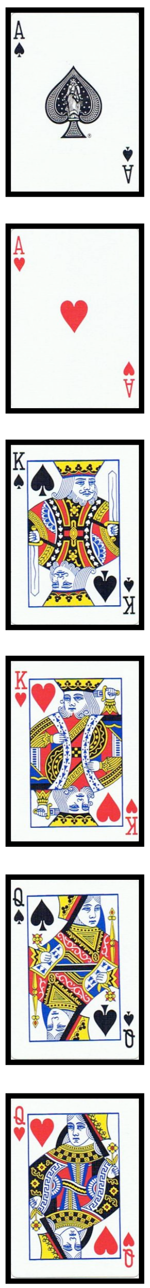
2) Do O(n^2) work in O(n) time.

3) Avoid isomorphic game states

4) Parallel computation

Fast Terminal Node Evaluation

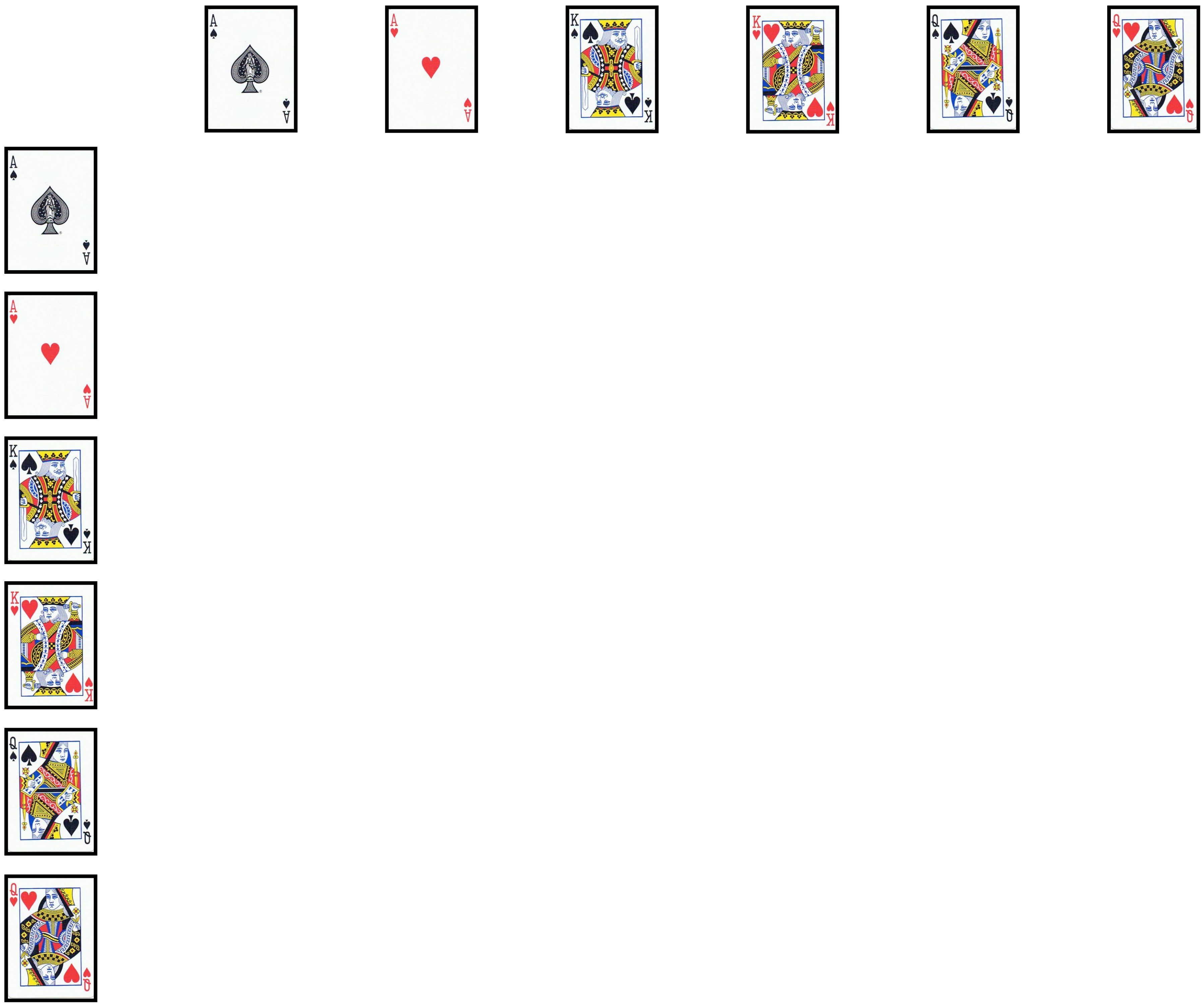
My n States



Fast Terminal Node Evaluation

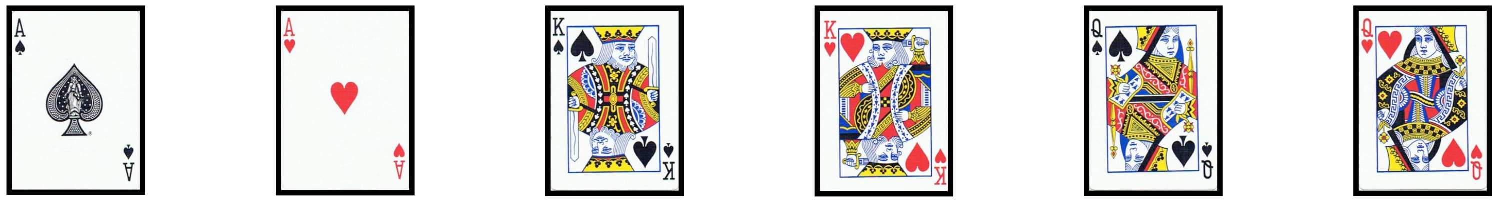
Opponent's *n* States

My *n* States

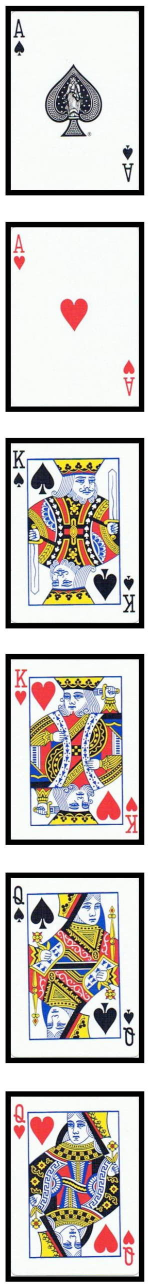


Fast Terminal Node Evaluation

Opponent's *n* States



My *n* States



$= p_1 * u_1 + p_2 * u_2 + p_3 * u_3 + p_4 * u_4 + p_5 * u_5 + p_6 * u_6$

$= p_1 * u_1 + p_2 * u_2 + p_3 * u_3 + p_4 * u_4 + p_5 * u_5 + p_6 * u_6$

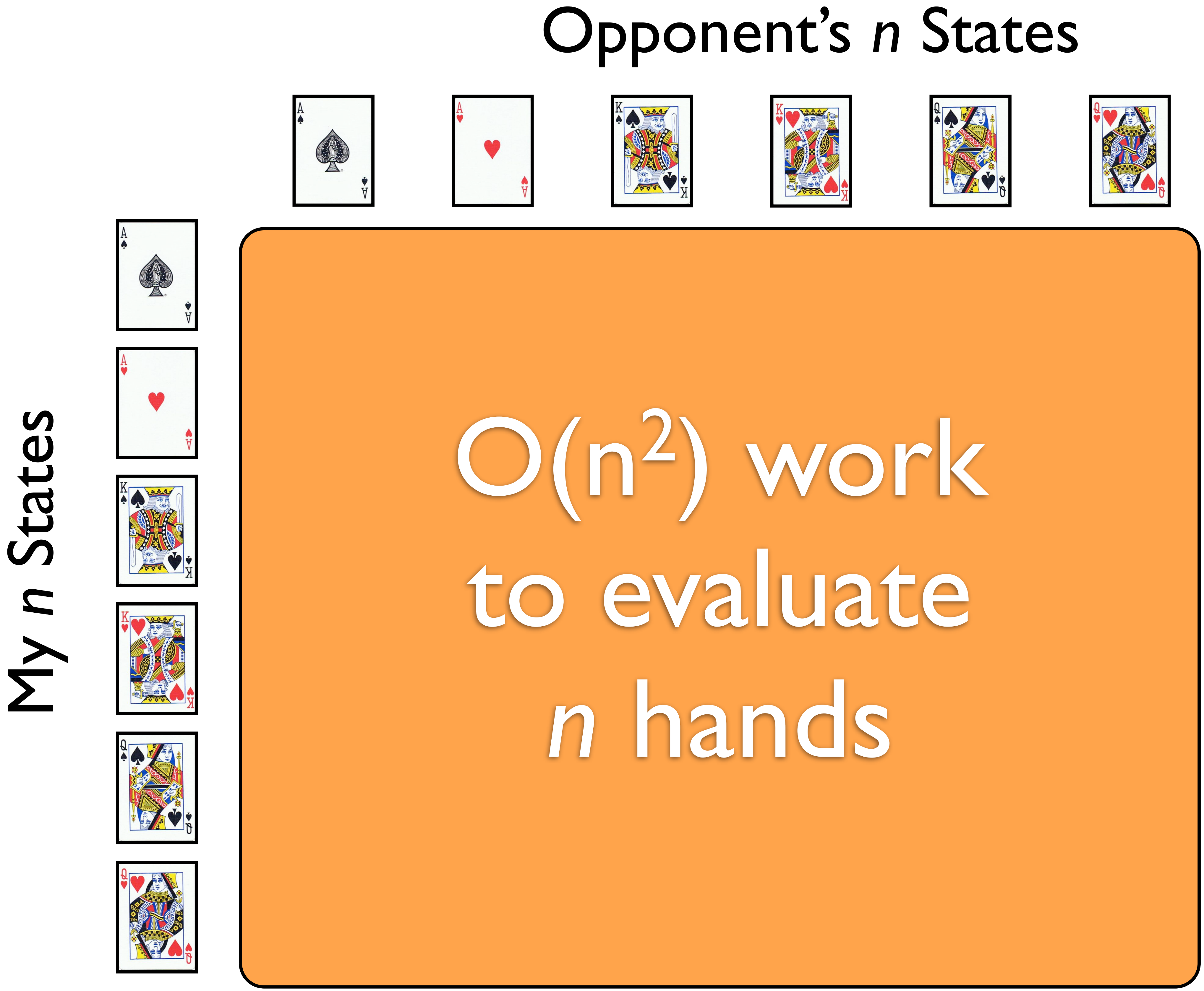
$= p_1 * u_1 + p_2 * u_2 + p_3 * u_3 + p_4 * u_4 + p_5 * u_5 + p_6 * u_6$

$= p_1 * u_1 + p_2 * u_2 + p_3 * u_3 + p_4 * u_4 + p_5 * u_5 + p_6 * u_6$

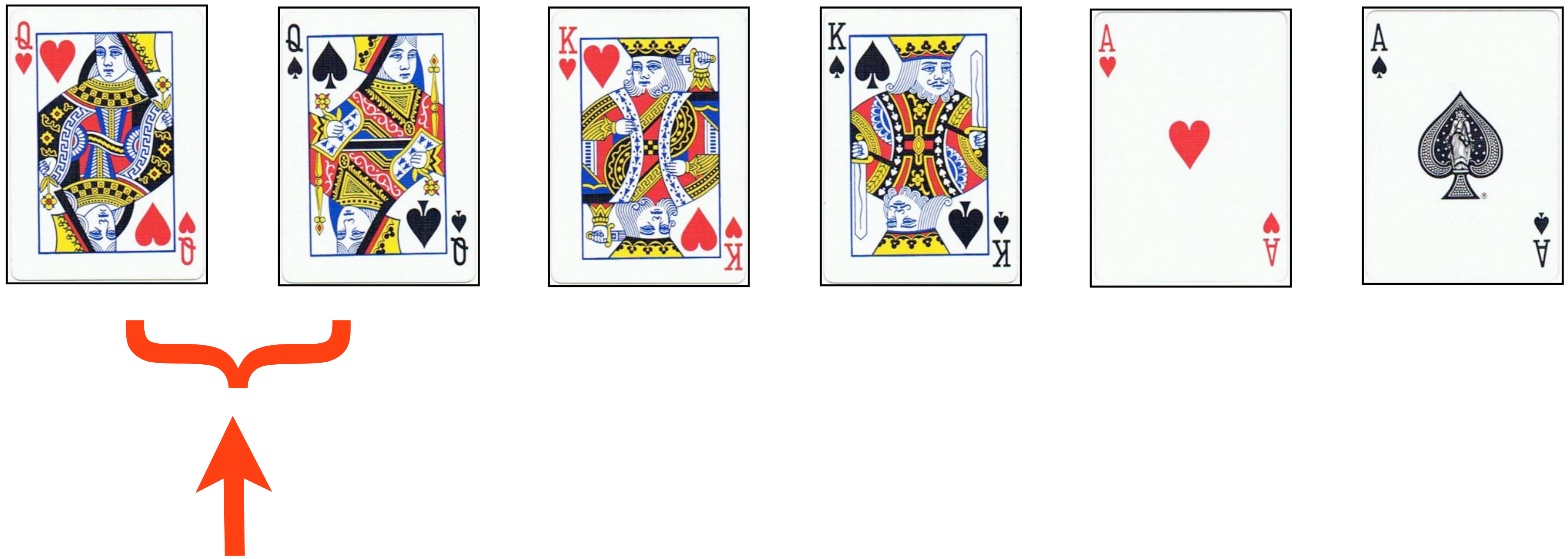
$= p_1 * u_1 + p_2 * u_2 + p_3 * u_3 + p_4 * u_4 + p_5 * u_5 + p_6 * u_6$

$= p_1 * u_1 + p_2 * u_2 + p_3 * u_3 + p_4 * u_4 + p_5 * u_5 + p_6 * u_6$

Fast Terminal Node Evaluation

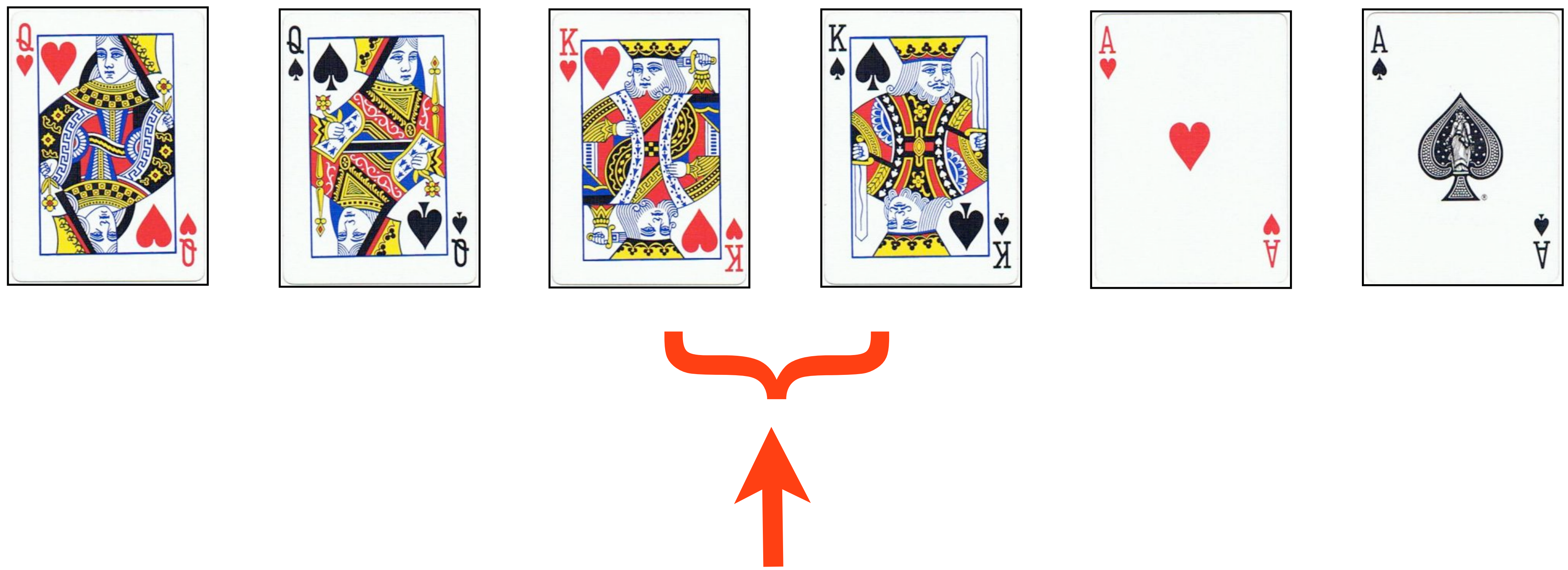


Fast Terminal Node Evaluation



Most games have structure that can be exploited.
In Poker, states are ranked, and the highest rank wins.

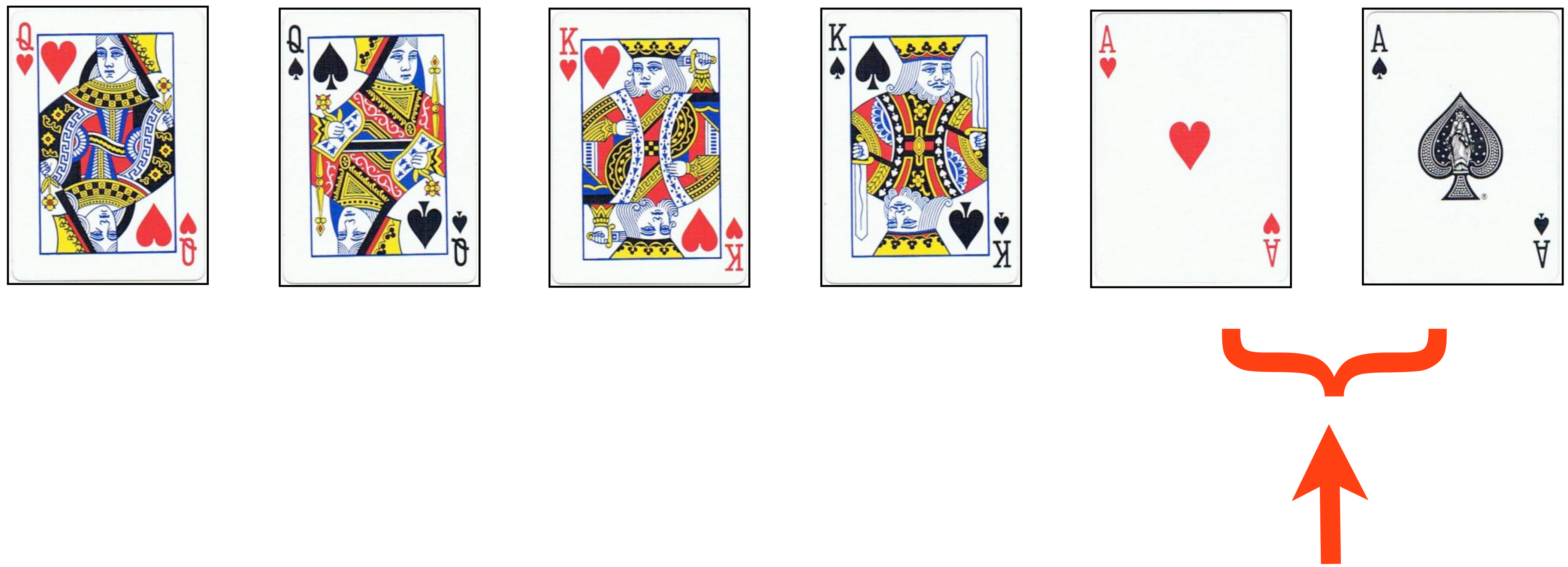
Fast Terminal Node Evaluation



To calculate one state's EV, we only need:
- Probability of opponent reaching weaker states
- Probability of opponent reaching stronger states

EV[i] = p(lose) * util(lose) + p(win) * util(win)

Fast Terminal Node Evaluation



By exploiting the game's structure, we can use two for() loops instead of two nested for() loops.

$O(n^2)$ to $O(n)$. 7.7x speedup in Texas hold'em.

(Some tricky details resolved in the paper)

Accelerated Best Response

The new technique has four orthogonal improvements:

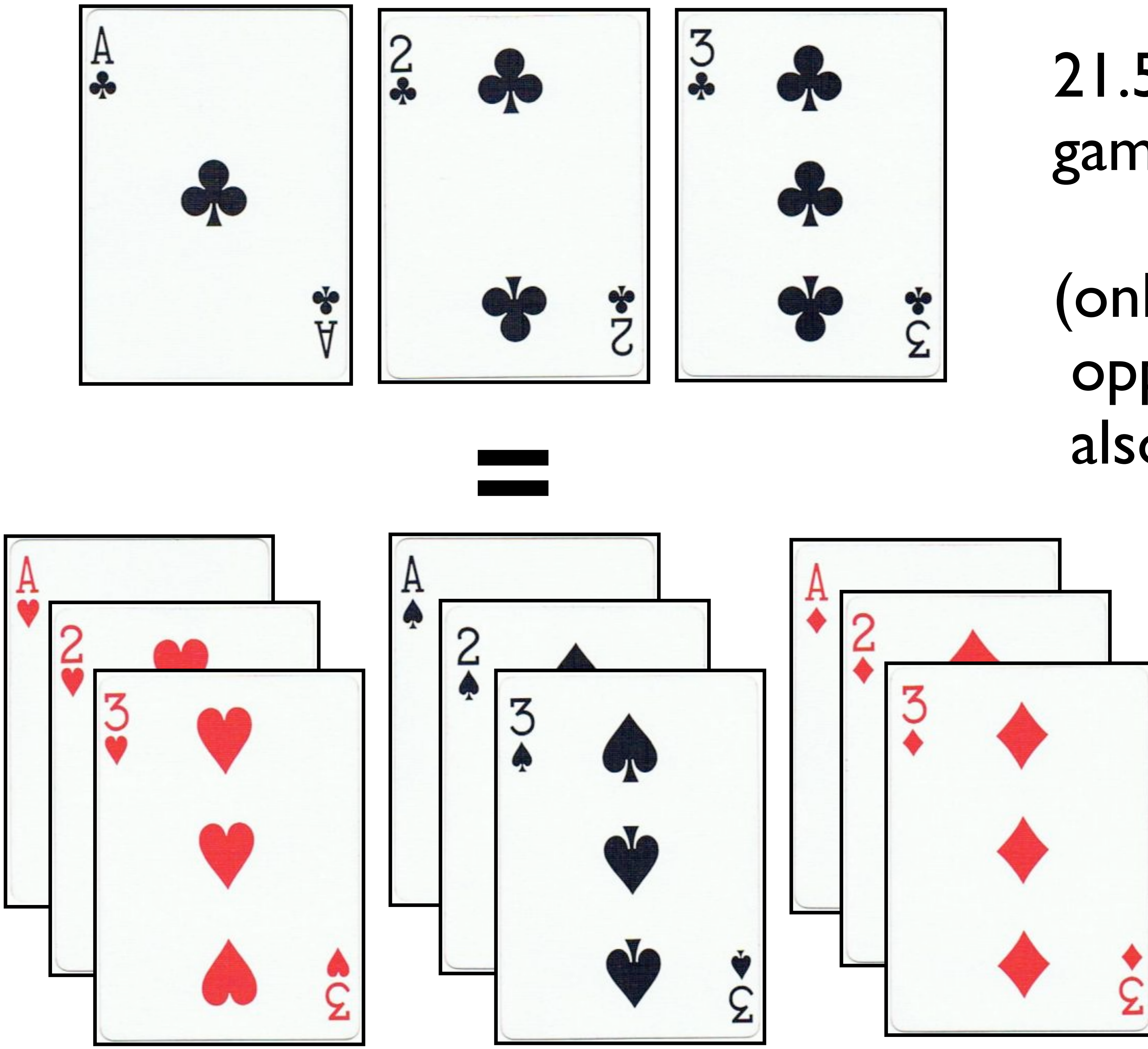
1) Take advantage of what the opponent doesn't know

2) Do O(n^2) work in O(n) time.

3) Avoid isomorphic game states

4) Parallel computation

Avoid Isomorphic States



21.5x reduction in game size

(only correct if opponent's strategy also does this)

Accelerated Best Response

The new technique has four orthogonal improvements:

1) Take advantage of what the opponent doesn’t know, to walk the much smaller public tree

2) Use a fast terminal node evaluation to do O(n^2) work in O(n) time.

3) Avoid isomorphic game states

4) Parallel computation

Parallel Computation

- ♥ 24,570 equal sized independent subtrees.
- ♣ Takes 4m30s to solve each one.
- ♦ 24,570 * 4.5 minutes = 76 cpu-days

Parallel Computation

- ♥ 24,570 equal sized independent subtrees.
- ♣ Takes 4m30s to solve each one.
- ♦ 24,570 * 4.5 minutes = 76 cpu-days
- ♠ 72 processors on a cluster: 1 day computation!

Evaluating the Progress of Computer Poker Research

Evaluating Computer Poker Agents

- Annual Computer Poker Competition (ACPC)
  - Started in 2006
  - Hosted at AAAI this year
  - 2-player Limit: Strongest agents are competitive with world's best human pros

- Most successful approach (U of A, CMU, many others):
  - Approximate a Nash equilibrium, worst case loss of $0 per game

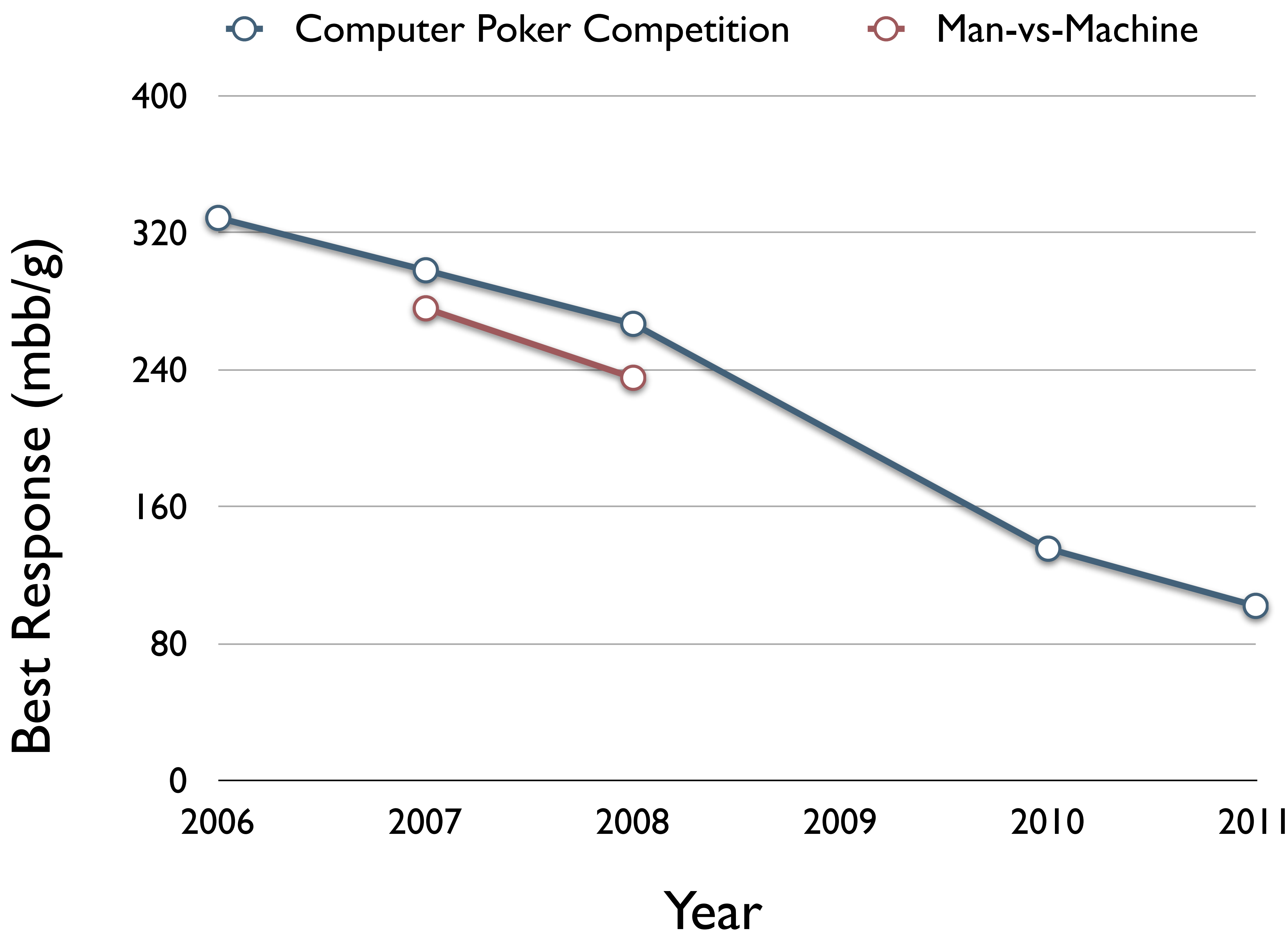
- For the first time, we can now tell how close we are to this goal!

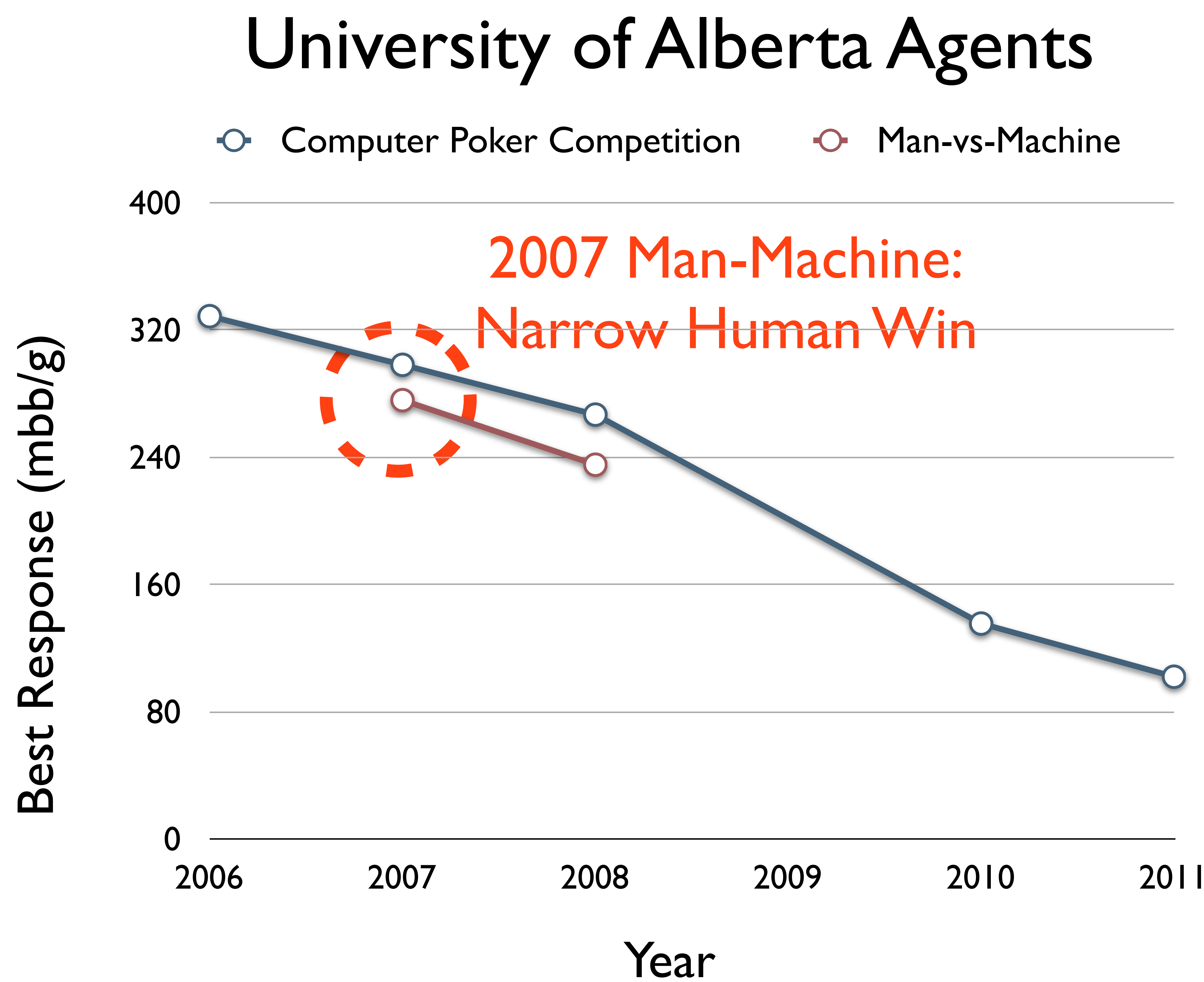
Trivial Opponents

	Value for Best Response
Always-Fold	750
Always-Call	1163.48
Always-Raise	3697.69
Uniform Random	3466.32

A human professional's goal is to win 50.
An optimal strategy would lose 0.
(Units are milli-big-blinds per game)

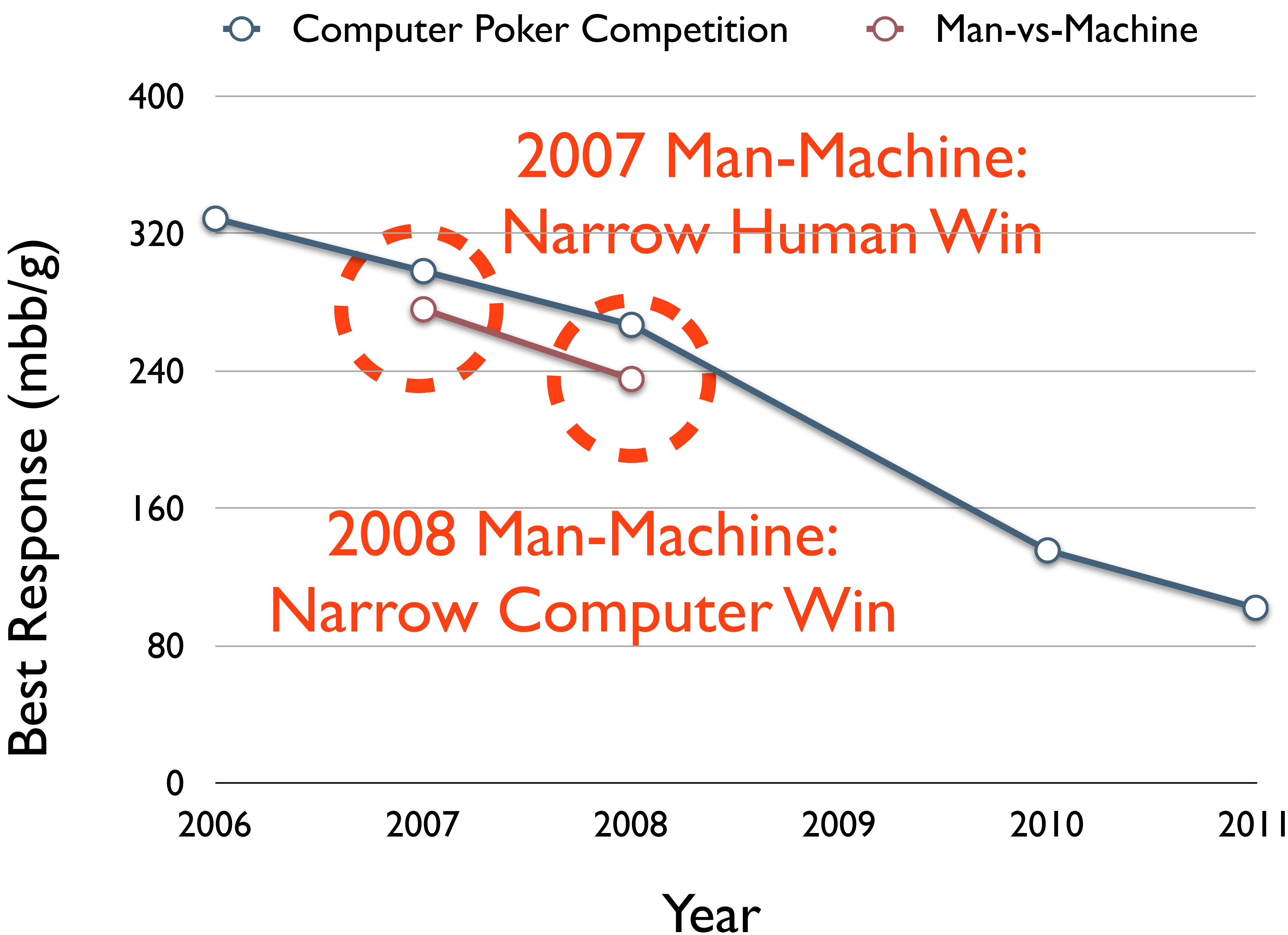
University of Alberta Agents





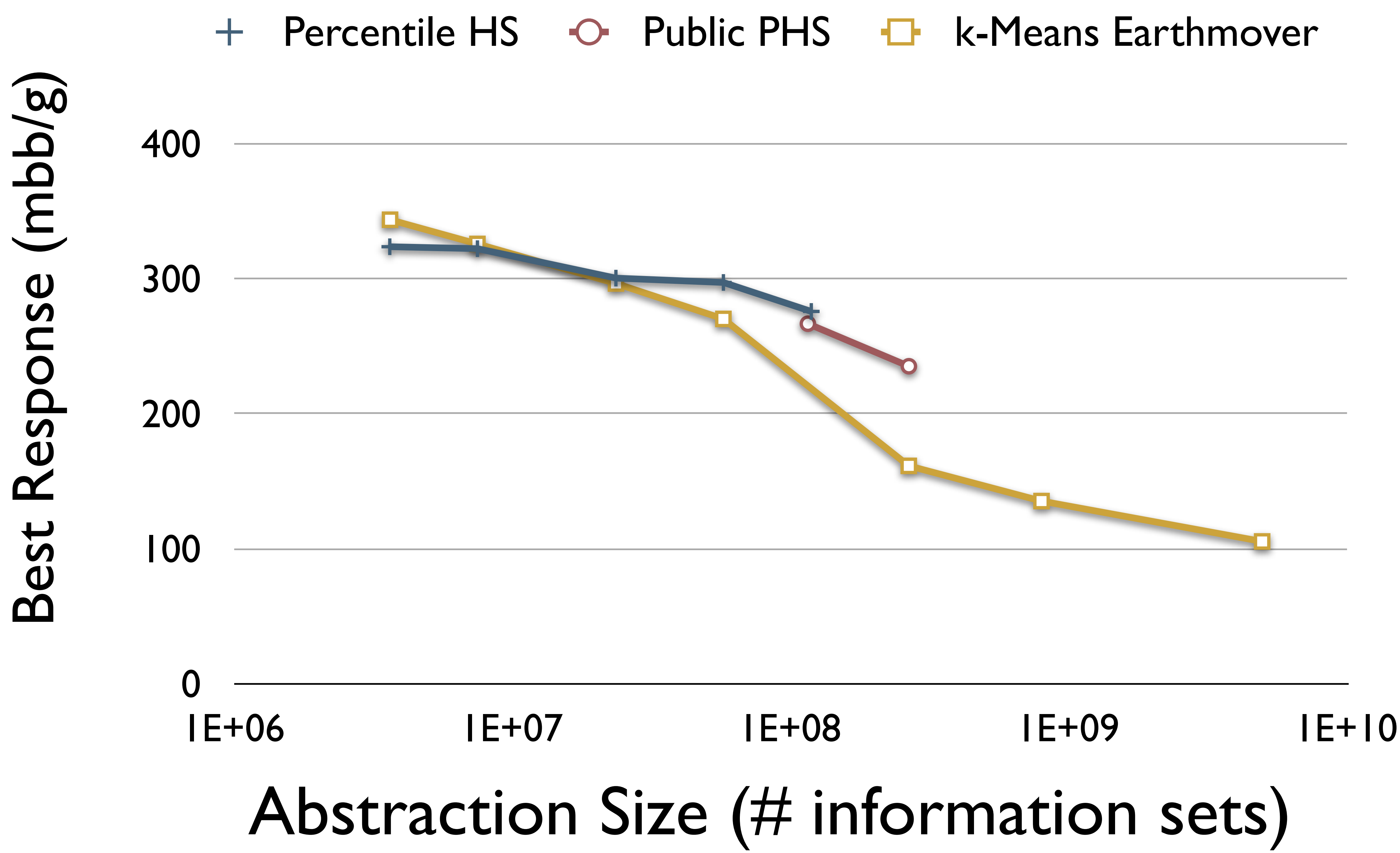
University of Alberta Agents
Computer Poker Competition
Man-vs-Machine
2007 Man-Machine:
Narrow Human Win
Best Response (mbb/g)
400
320
240
160
80
0
2006
2007
2008
2009
2010
2011
Year

University of Alberta Agents



Evaluating the University of Alberta agents

Comparing Abstraction Techniques:



Evaluating Computer Poker Agents: 2010 Competition

	Rock hopper	GGValuta	HyperB (UofA)	PULPO	GS6 (CMU)	Littlerock	Best Response
Rock hopper		6	3	7	37	77	300
GGValuta	-6		3	1	31	77	237
HyperB (UofA)	-3	-3		2	31	70	135
PULPO	-7	-1	-2		32	125	399
GS6 (CMU)	-37	-31	-31	-32		47	318
Littlerock	-77	-77	-70	-125	-47		421

Conclusion

- ♥ Fast best-response calculation in imperfect information games
- ♣ The previously intractable computation can now be run in a day!
- ♦ Computer poker community is making steady progress towards robust strategies
- ♠ Many additional exciting results in the paper and at the poster!

More details at our poster!
Today, 4:00 - 5:20, Room 120-121

Additional Slides:

Expectimax

Public Tree

n^2 to n

Abstraction

Pathologies

CFR

Polaris

Hyperborean 2009

Tilting

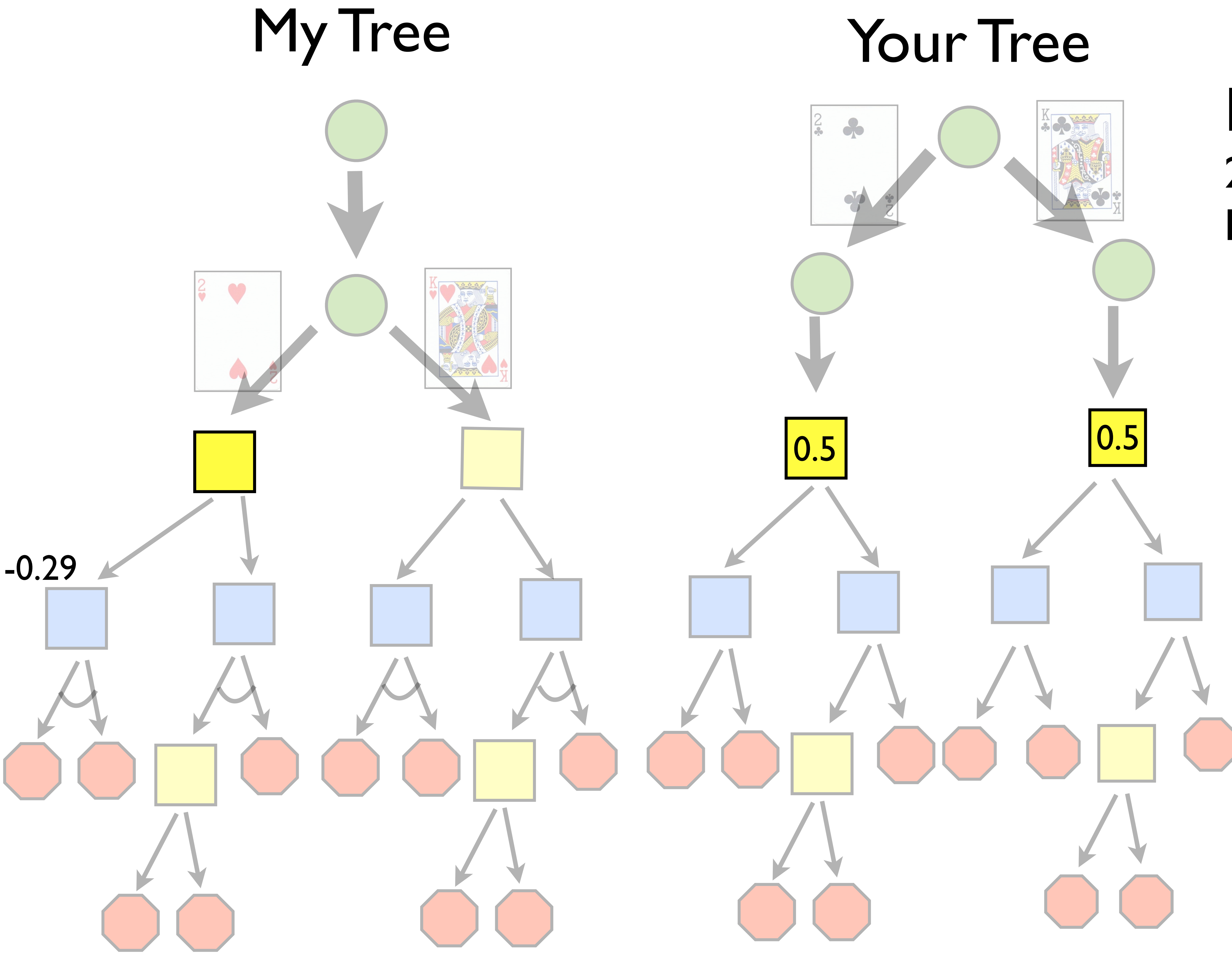
Additional Graphs

Leduc Hold'em Pathologies

Abstraction	Best Response
Real Game vs Real Game	**0**
J.Q.K vs Real Game	**55.2**
[JQ].K vs Real Game	**69.0**
J.[QK] vs Real Game	**126.3**
[JQK] vs Real Game	**219.3**
[JQ].K vs [JQ].K	**272.2**
[JQ].K vs J.Q.K	**274.1**
Real Game vs J.[QK]	**345.7**
Real Game vs [JQ].K	**348.9**
J.Q.K vs J.Q.K	**359.9**
J.Q.K vs [JQ].K	**401.3**
J.[QK] vs J.[QK]	**440.6**
Real Game vs [JQK]	**459.5**
Real Game vs J.Q.K	**491.0**
[JQK] vs [JQK]	**755.8**

Home

Expectimax



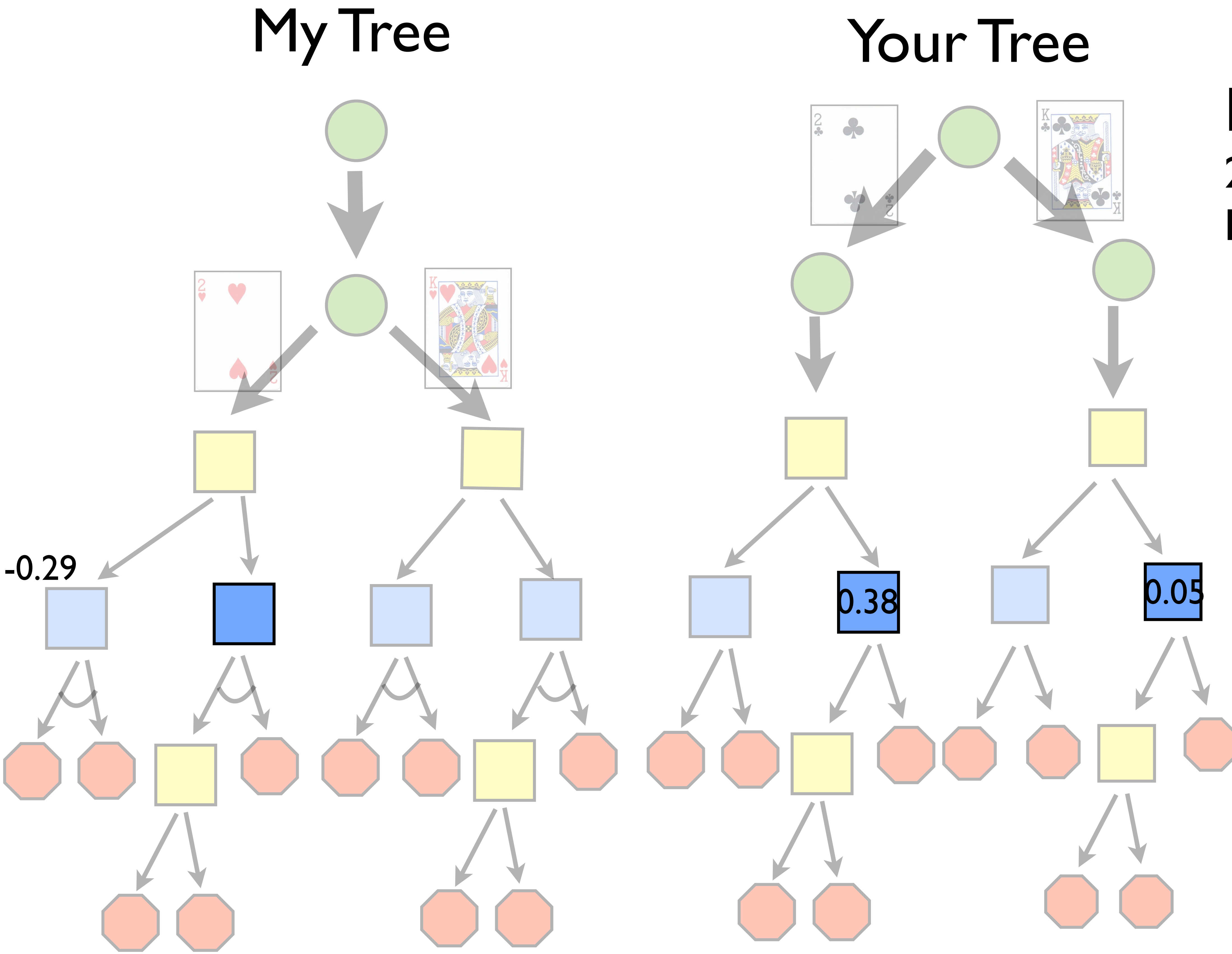
Reach:

2: 0.5

K: 0.5

Home

Conventional Best Response in one tree walk



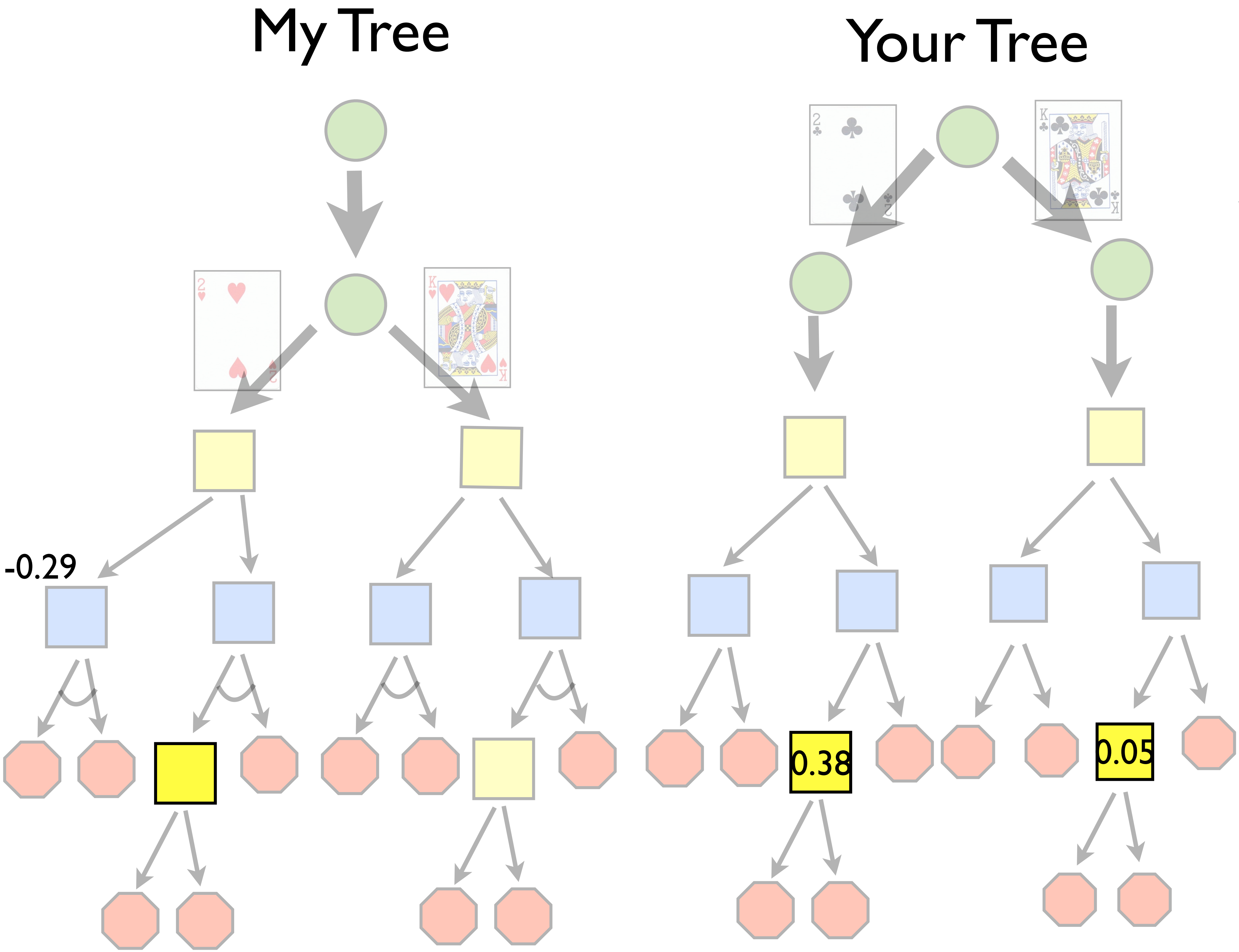
Reach:

2: 0.5*0.75

K: 0.5*0.1

Home

Conventional Best Response in one tree walk



Reach:

2: 0.5*0.75

K: 0.5*0.1

Home

Conventional Best Response in one tree walk

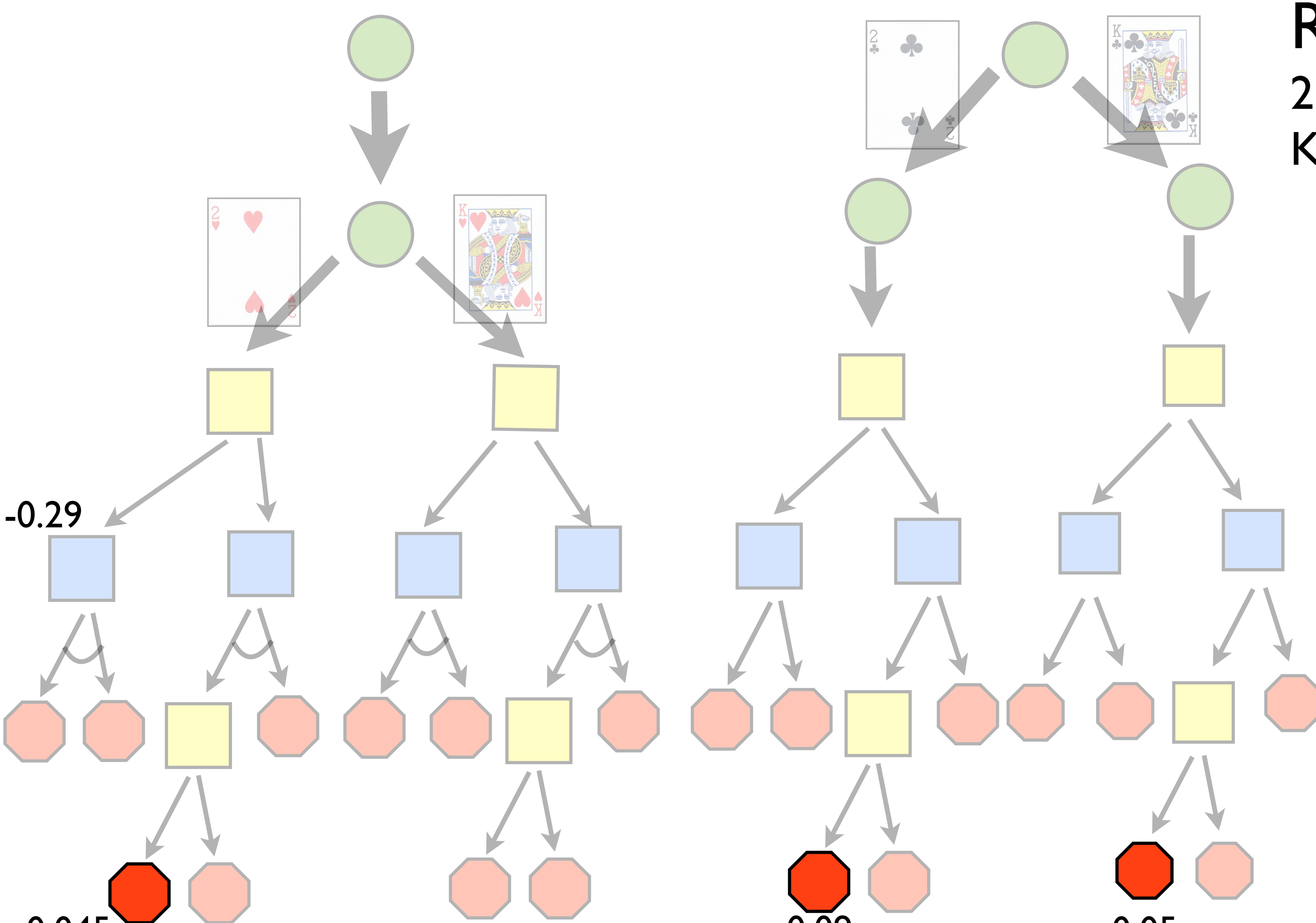
My Tree

Your Tree

Reach:

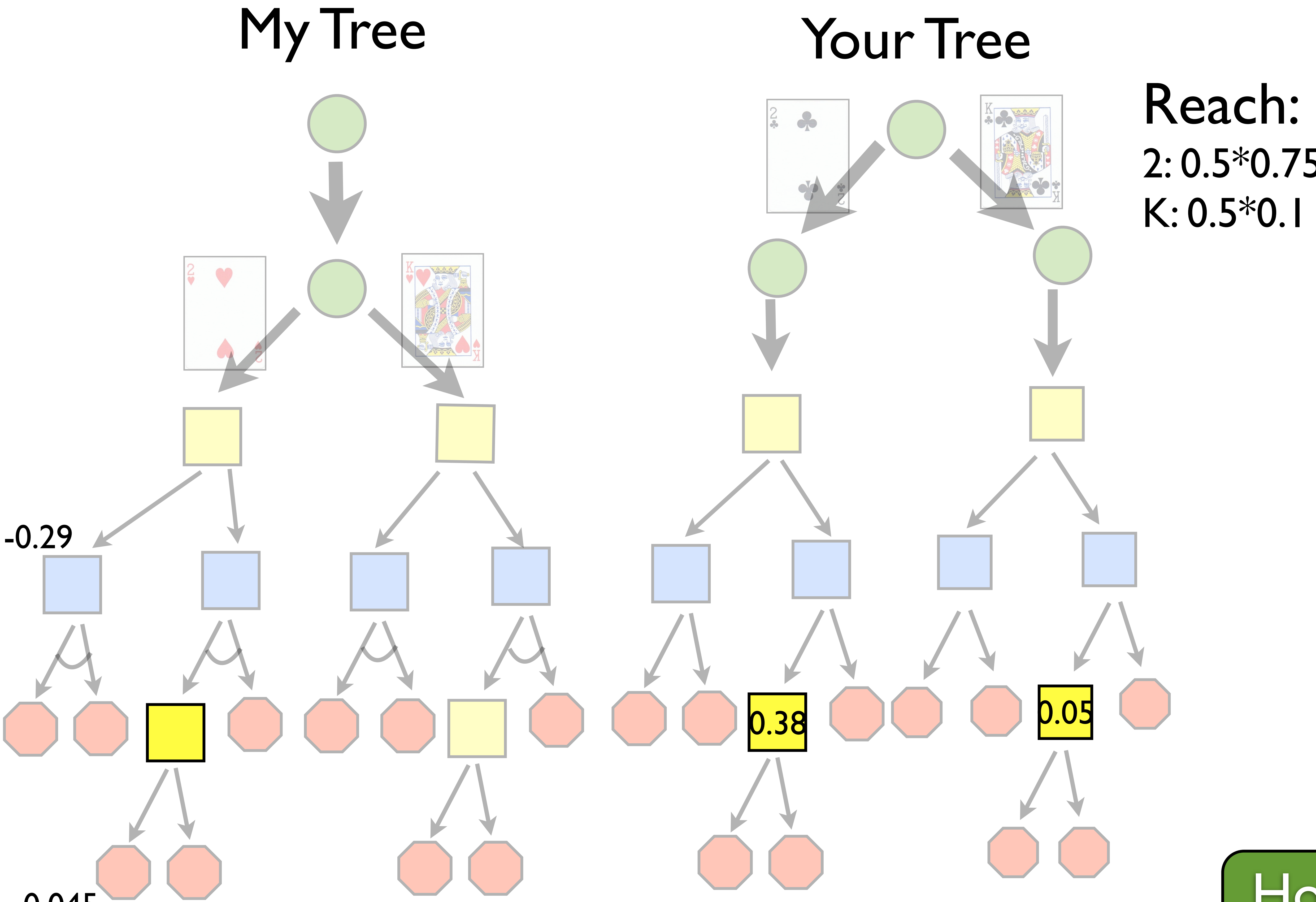
2: 0.5*0.75*0.25

K: 0.5*0.1*0.9



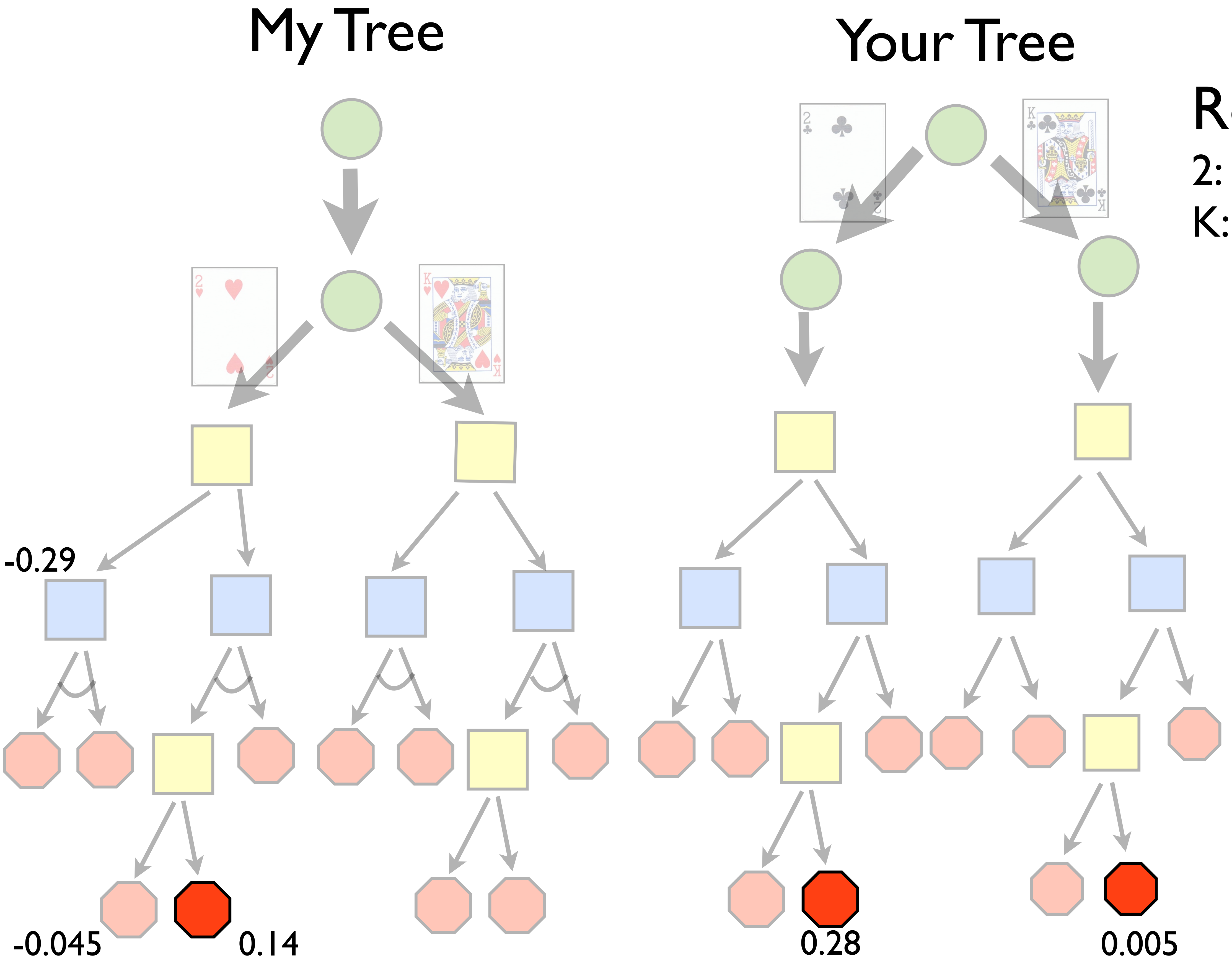
Home

Conventional Best Response in one tree walk



Home

Conventional Best Response in one tree walk



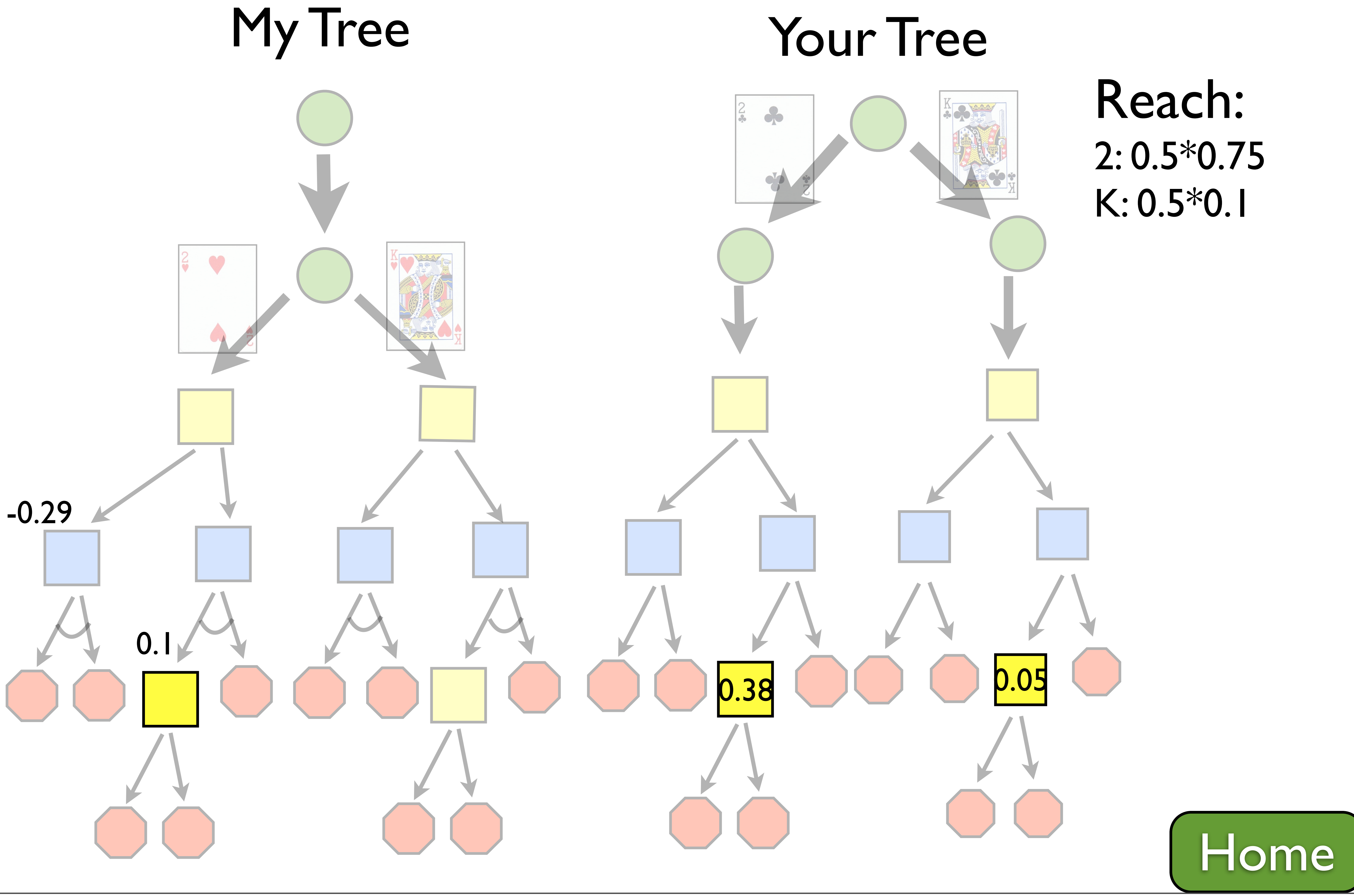
Reach:

2: 0.5*0.75*0.75

K: 0.5*0.1*0.1

Home

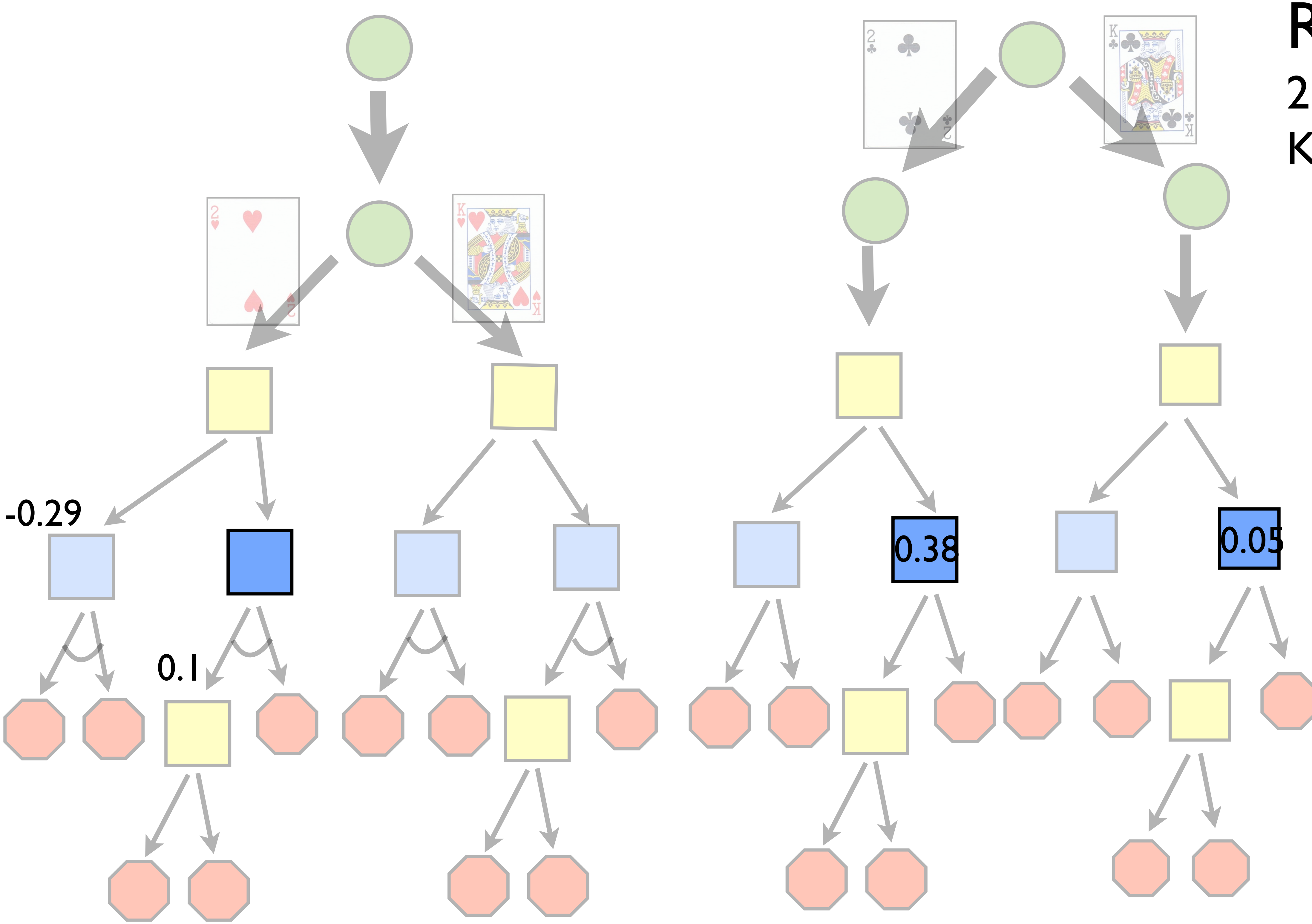
Conventional Best Response in one tree walk



Conventional Best Response in one tree walk

My Tree

Your Tree



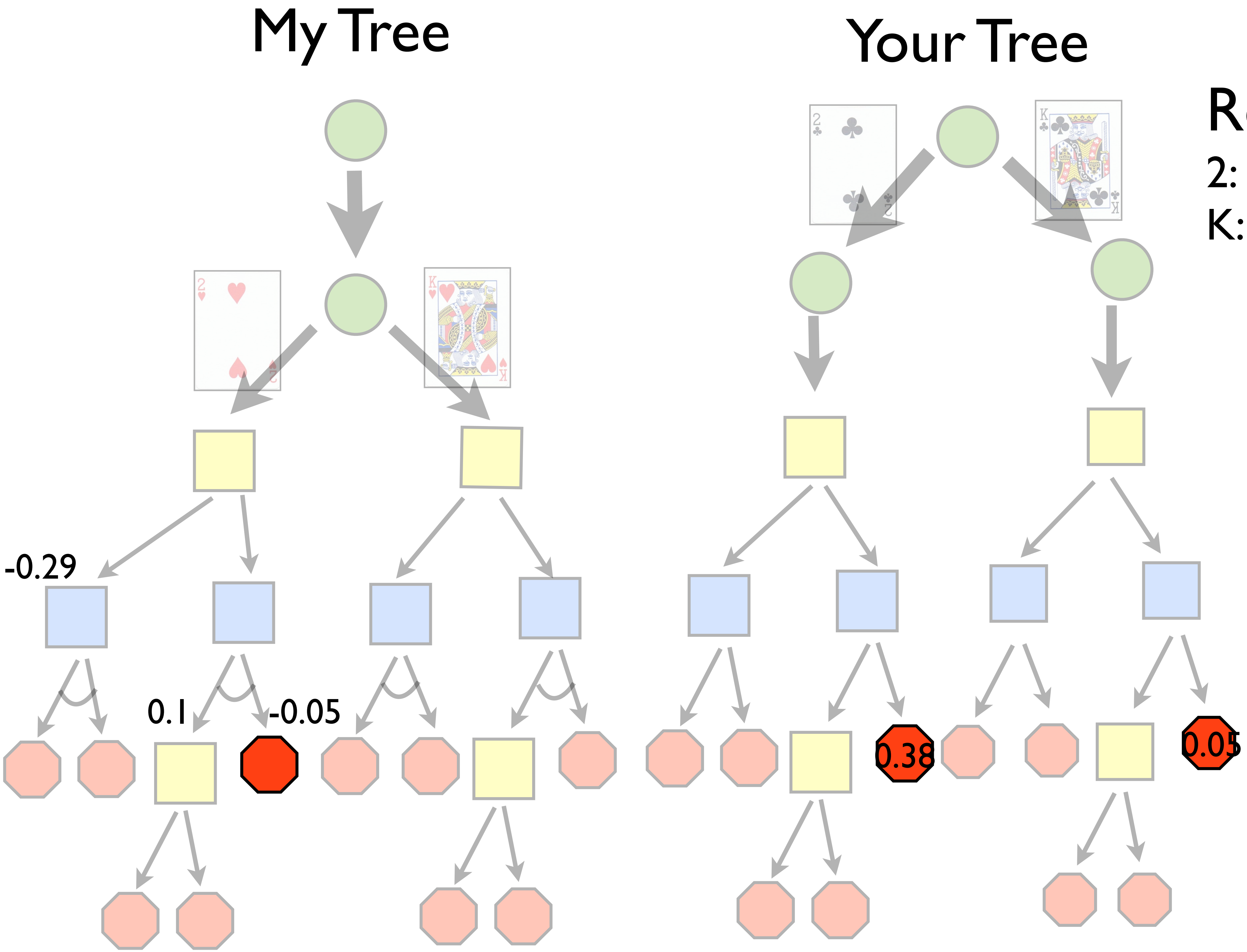
Reach:

2: 0.5*0.75

K: 0.5*0.1

Home

Conventional Best Response in one tree walk



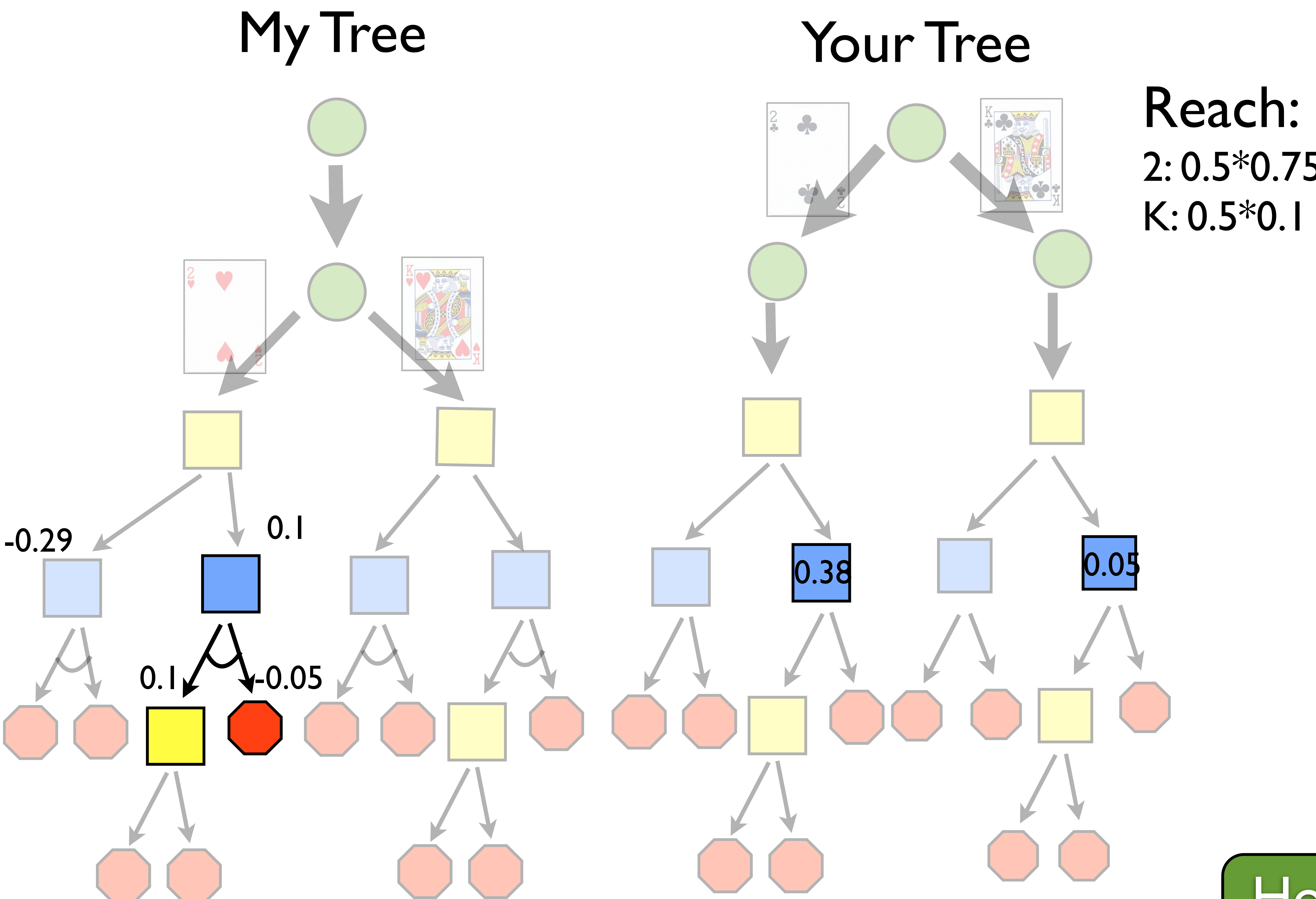
Reach:

2: 0.5*0.75

K: 0.5*0.1

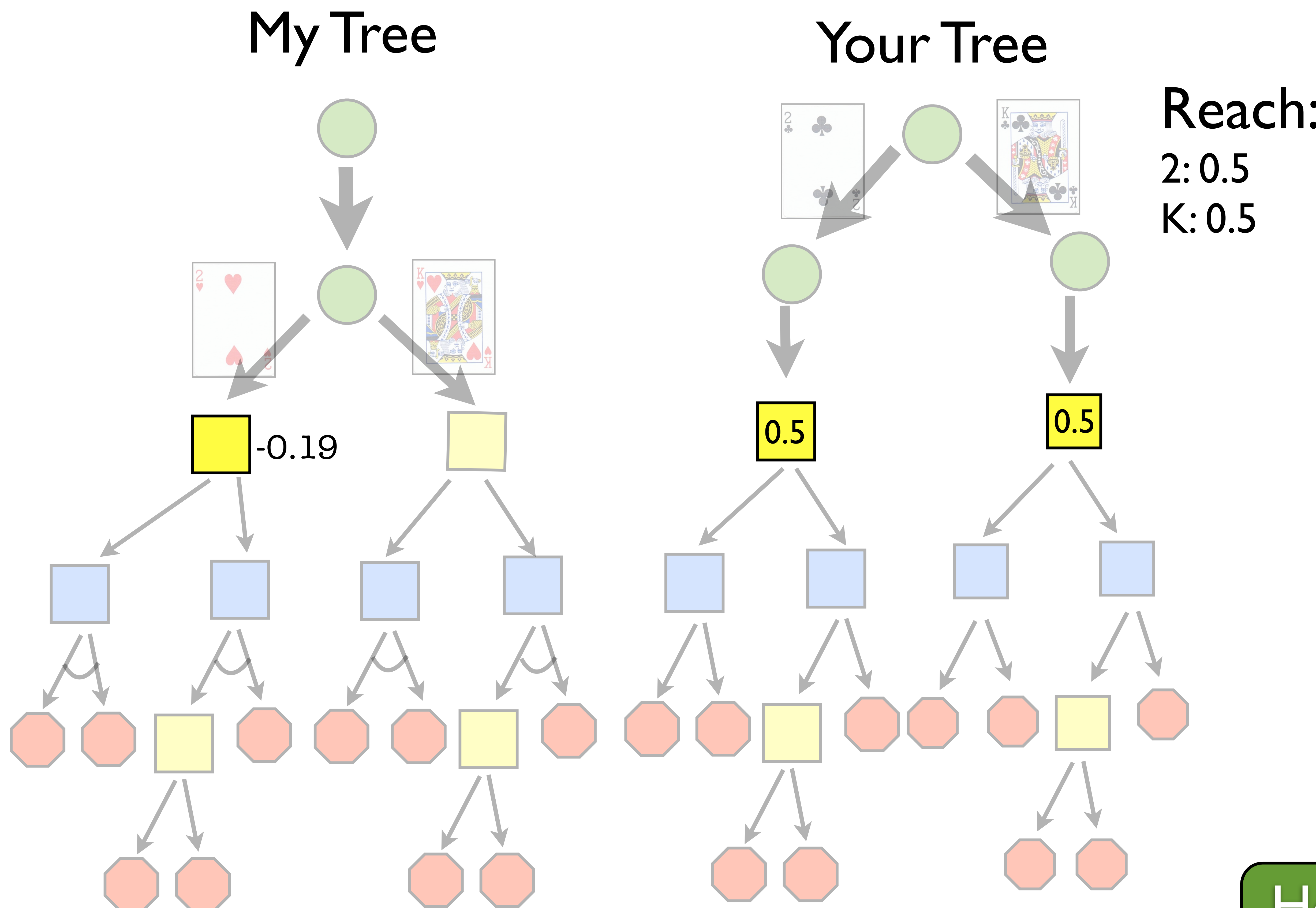
Home

Conventional Best Response in one tree walk



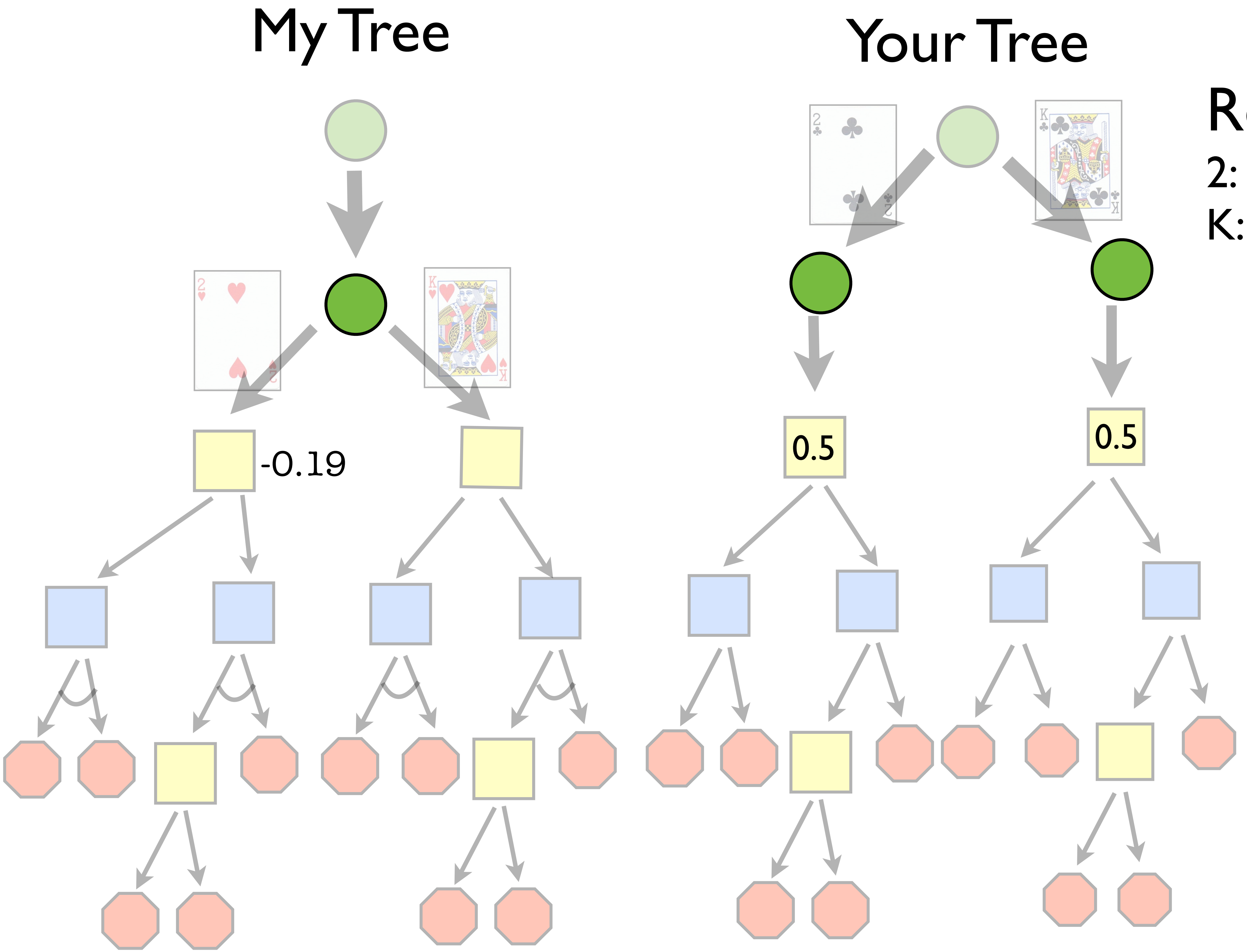
Home

Conventional Best Response in one tree walk



Home

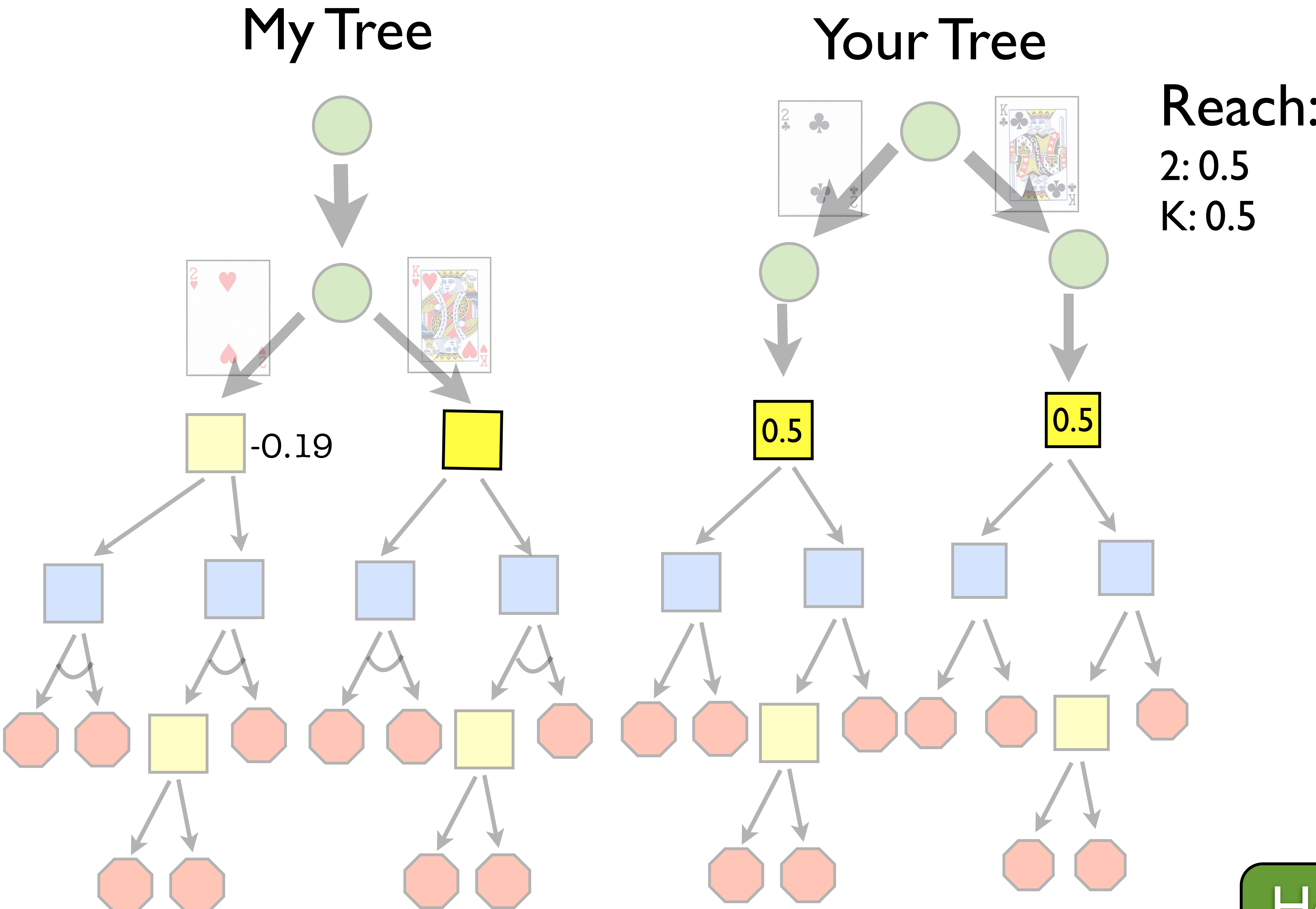
Conventional Best Response in one tree walk



Reach:
2: 0.5
K: 0.5

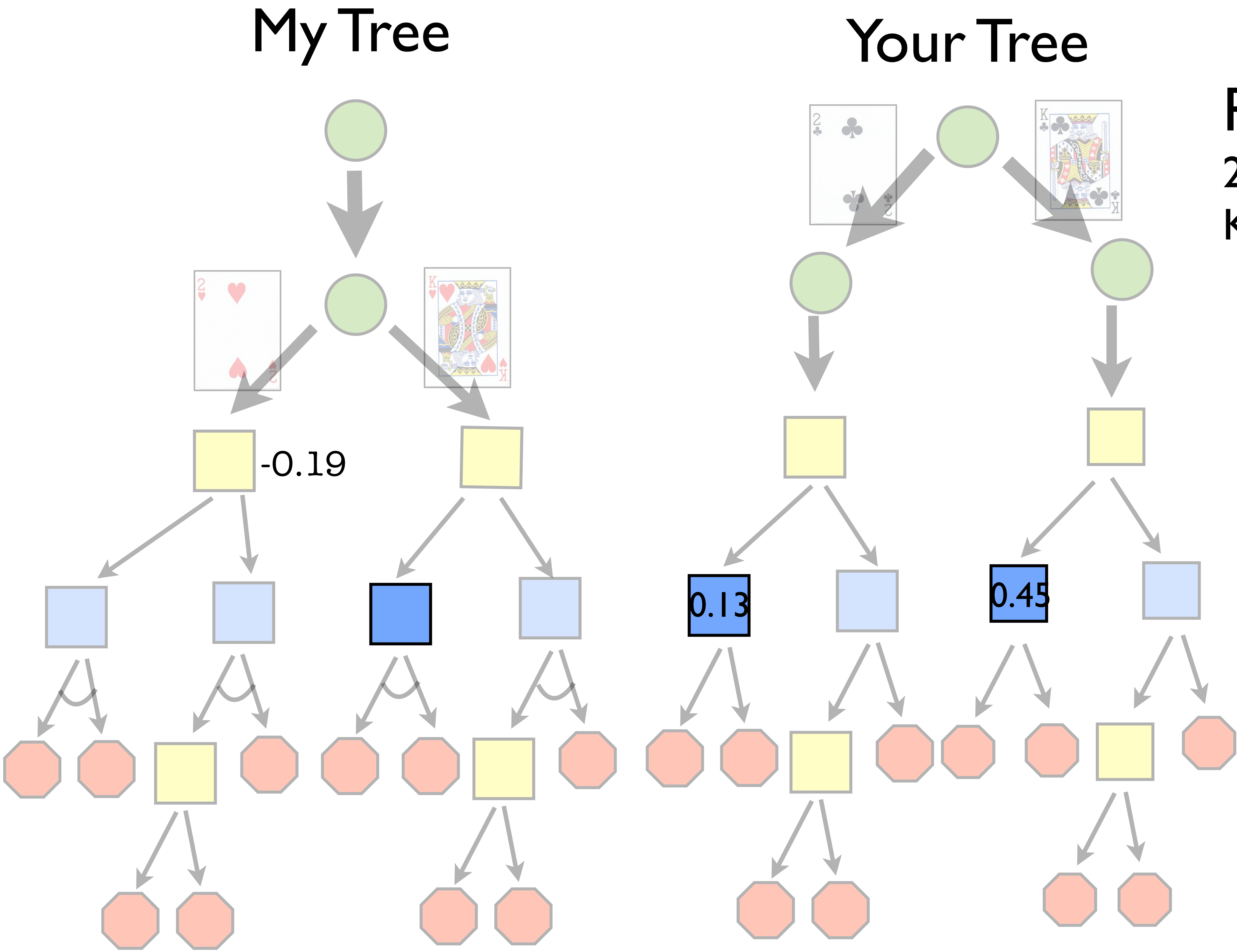
Home

Conventional Best Response in one tree walk



Home

Conventional Best Response in one tree walk



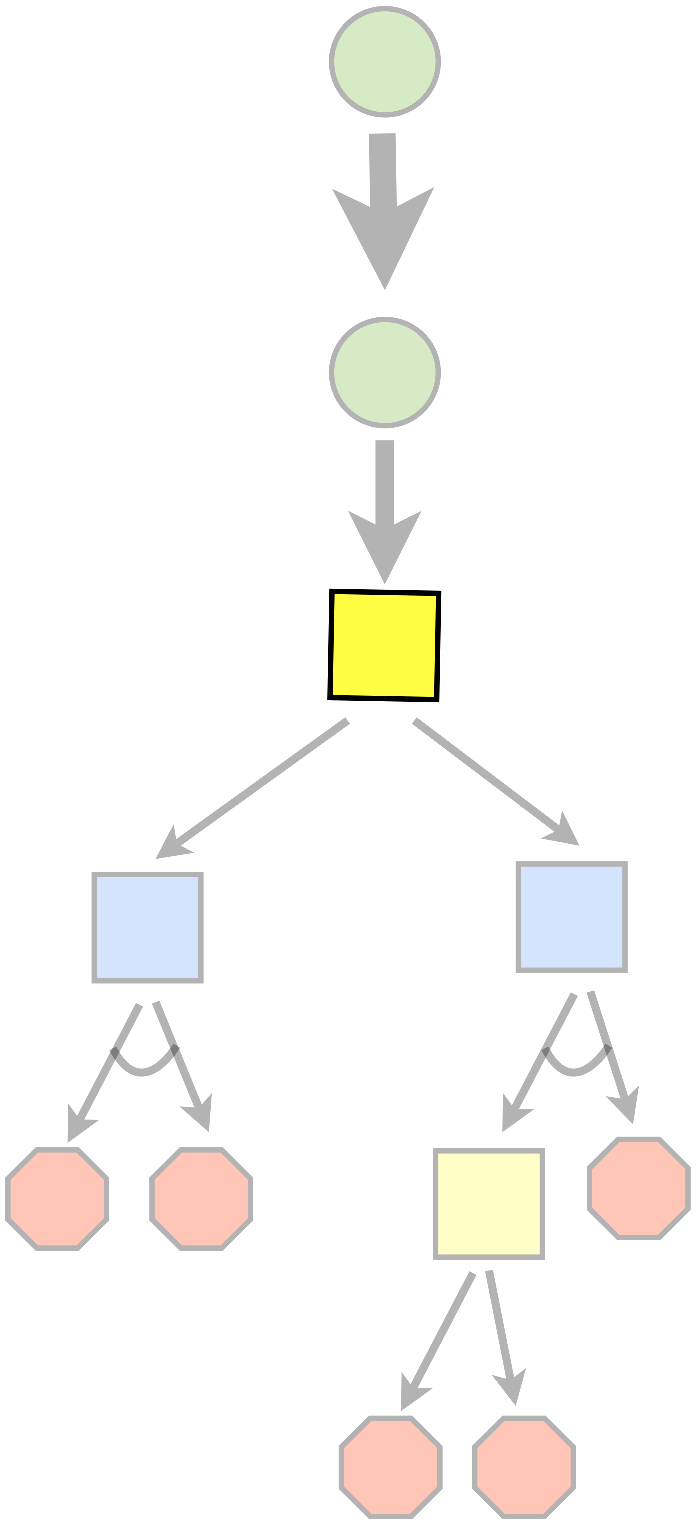
Reach:

2: 0.5*0.25

K: 0.5*0.9

Home

1: Walking the Public Tree



Their Reach Prob:

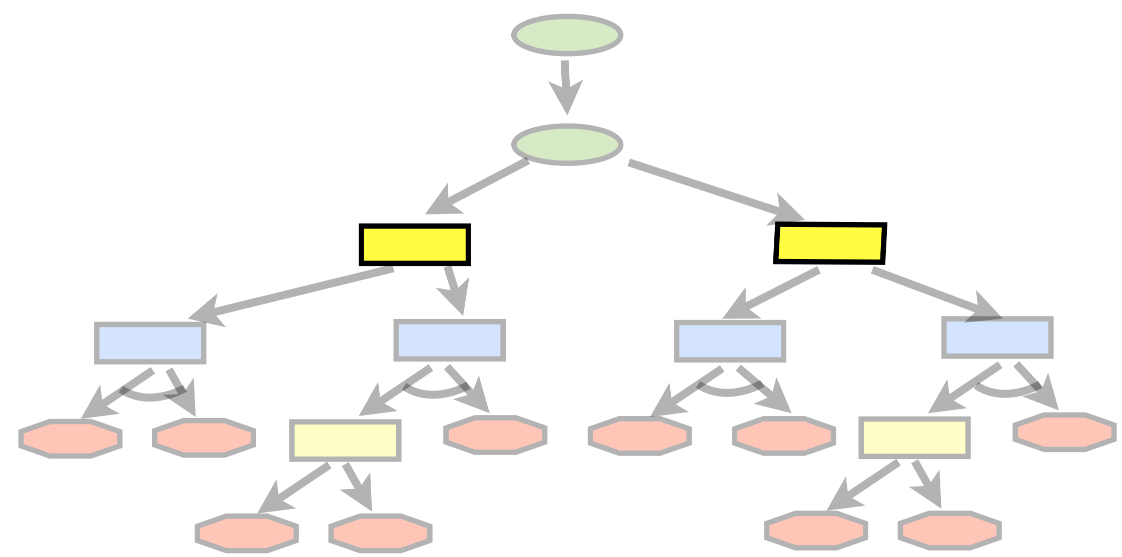
2: 0.5

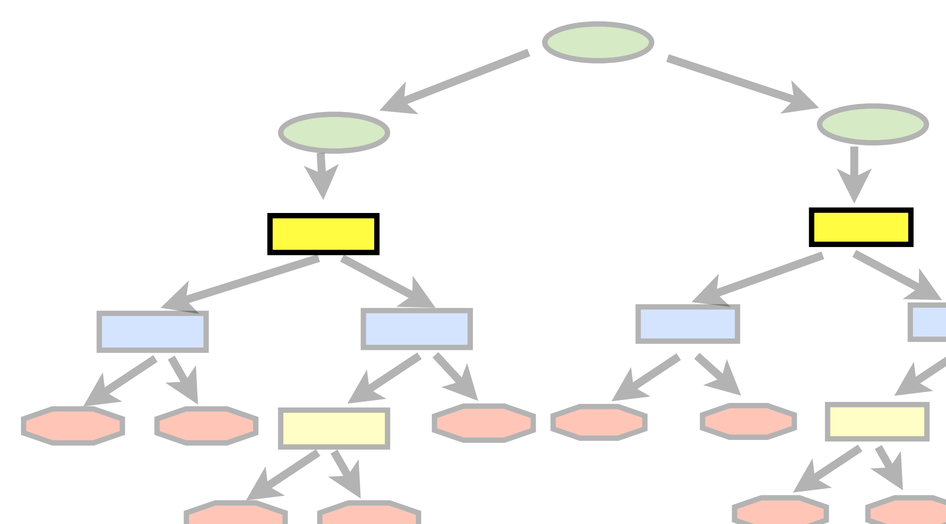
K: 0.5

My Value:

2:

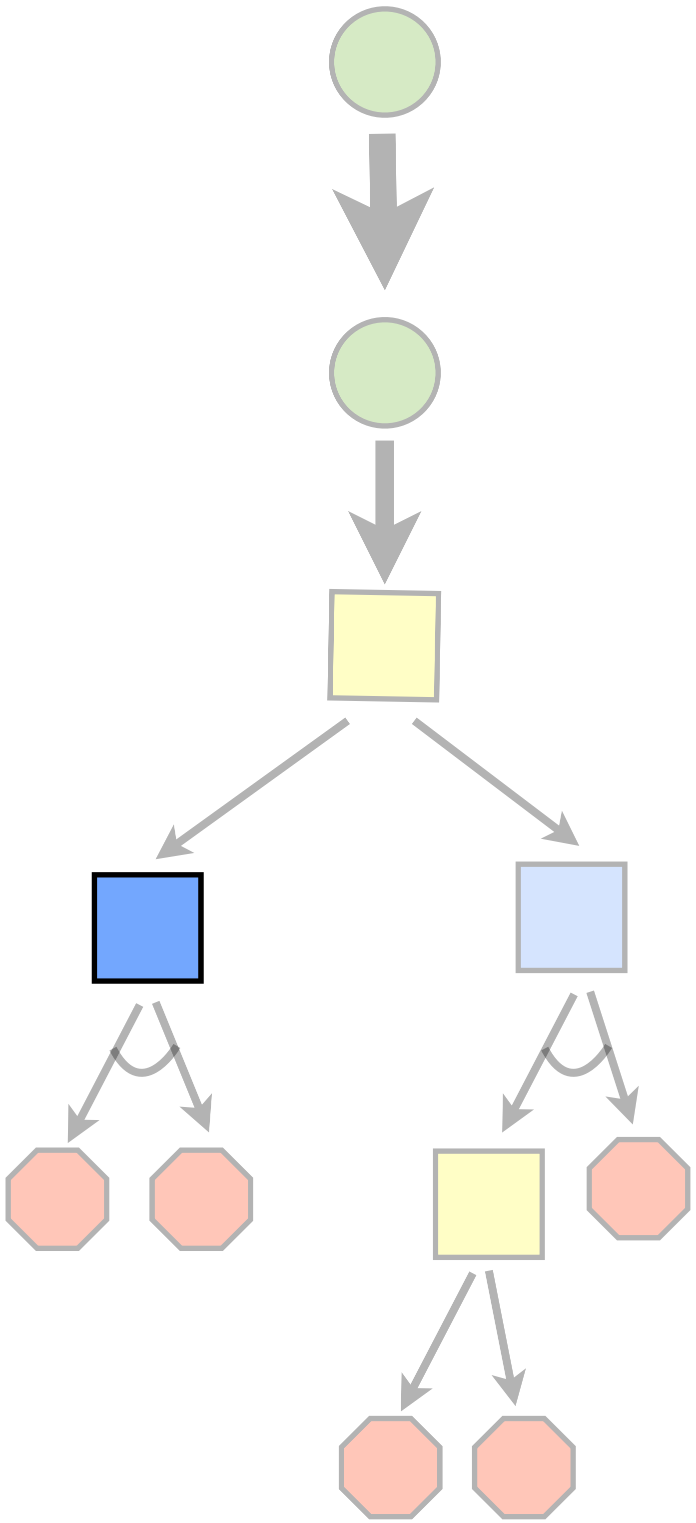
K:





Home

1: Walking the Public Tree



Their Reach Prob:

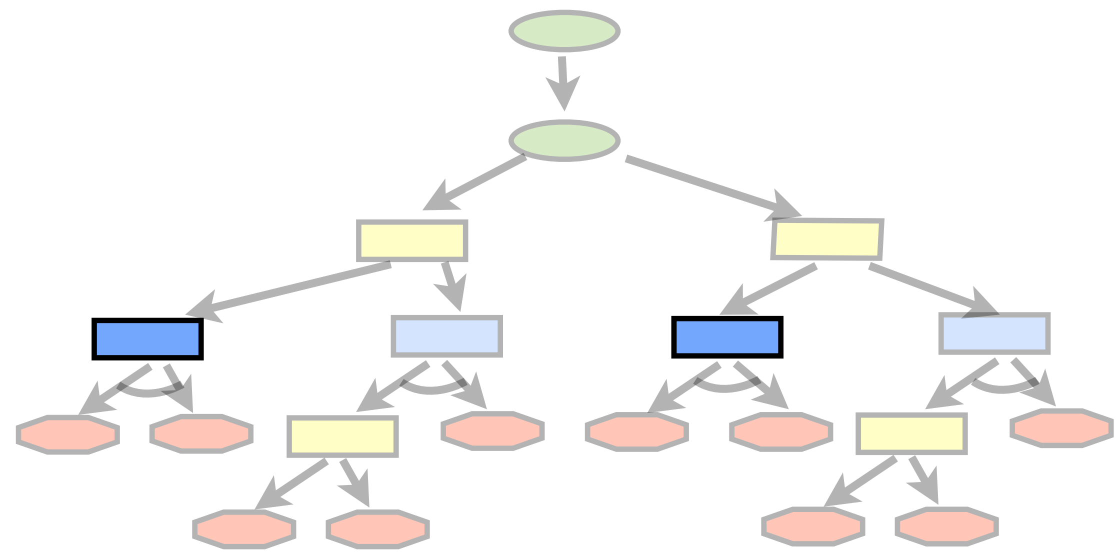
2: 0.5*0.25

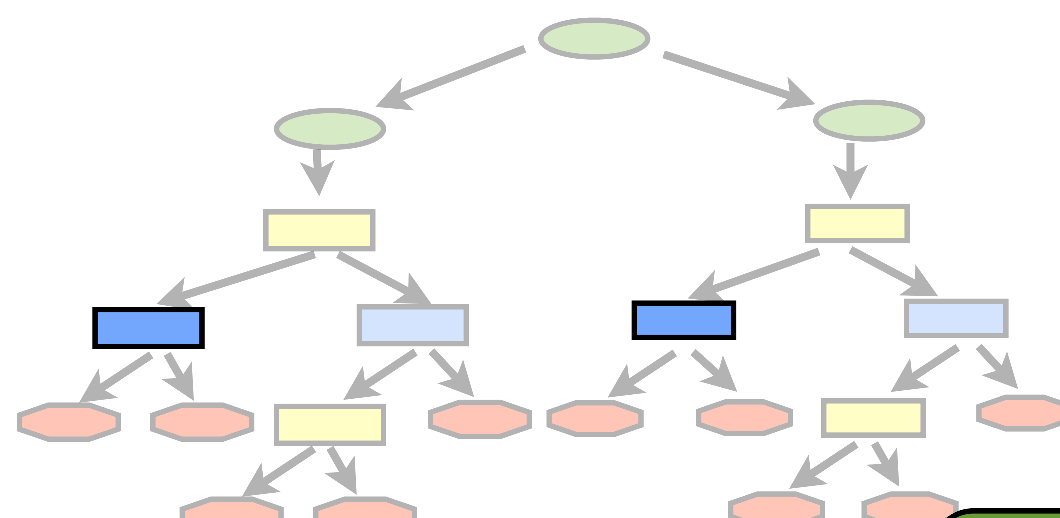
K: 0.5*0.9

My Value:

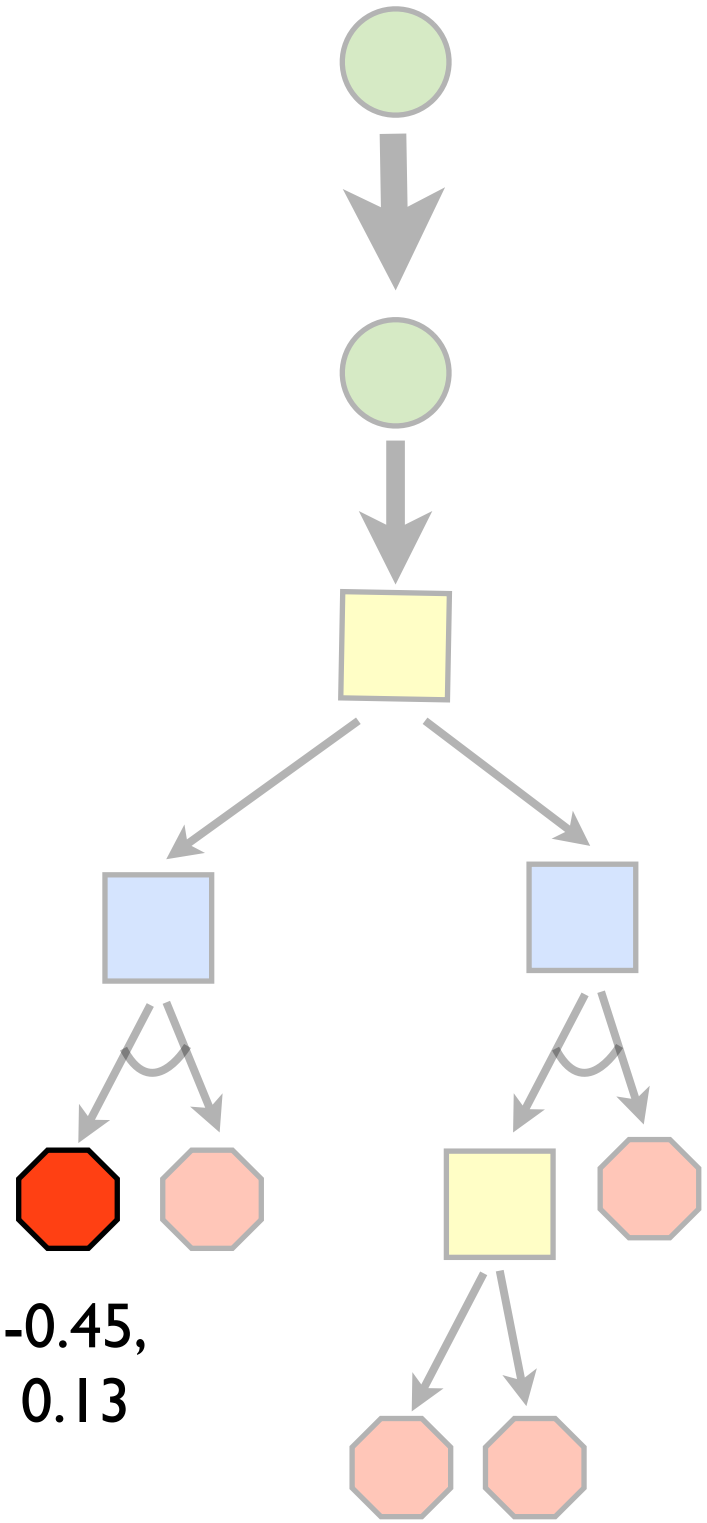
2:

K:





1: Walking the Public Tree



Their Reach Prob:

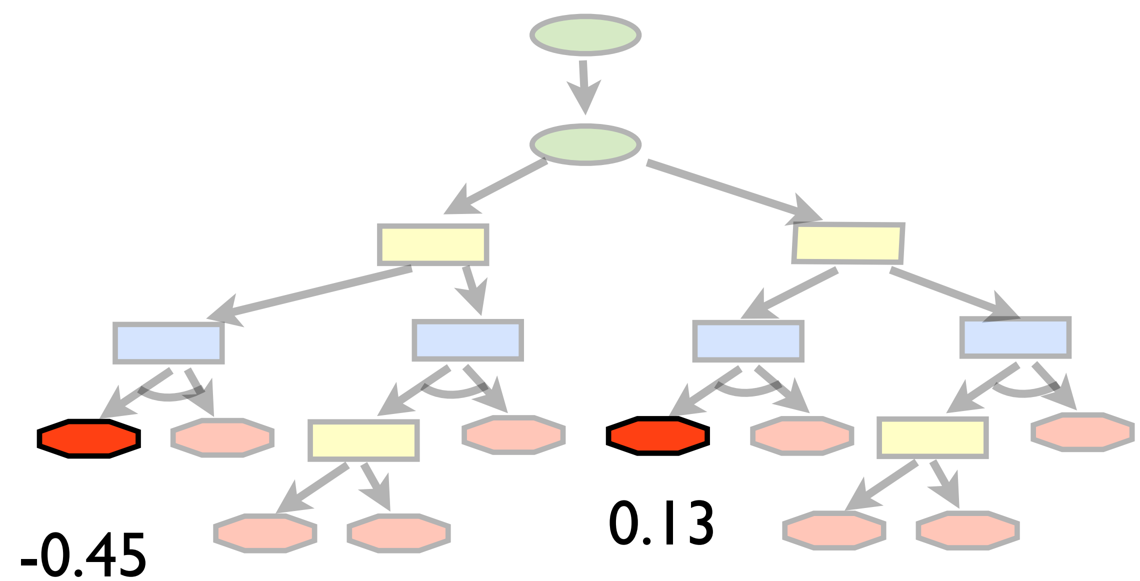
2: 0.5*0.25

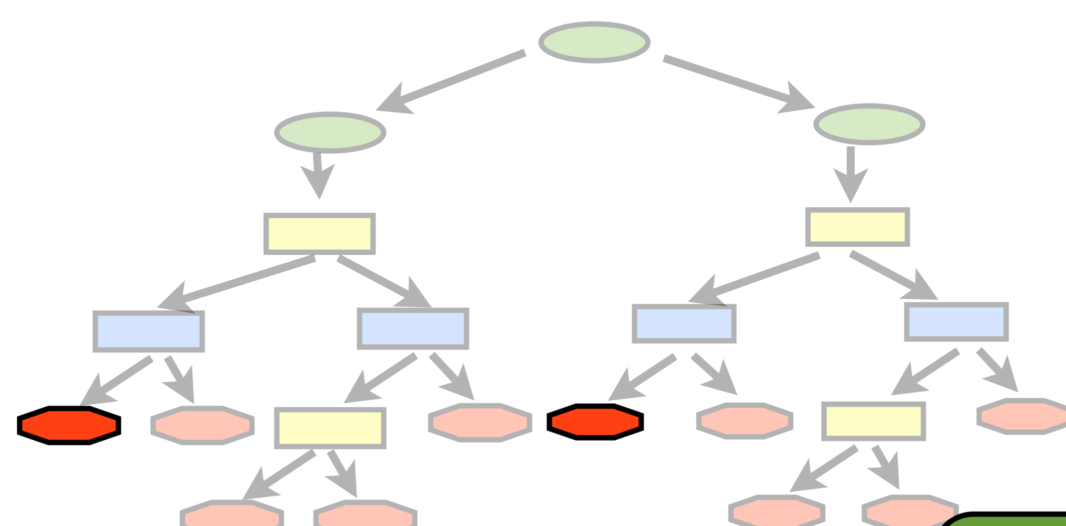
K: 0.5*0.9

My Value:

2: -0.45

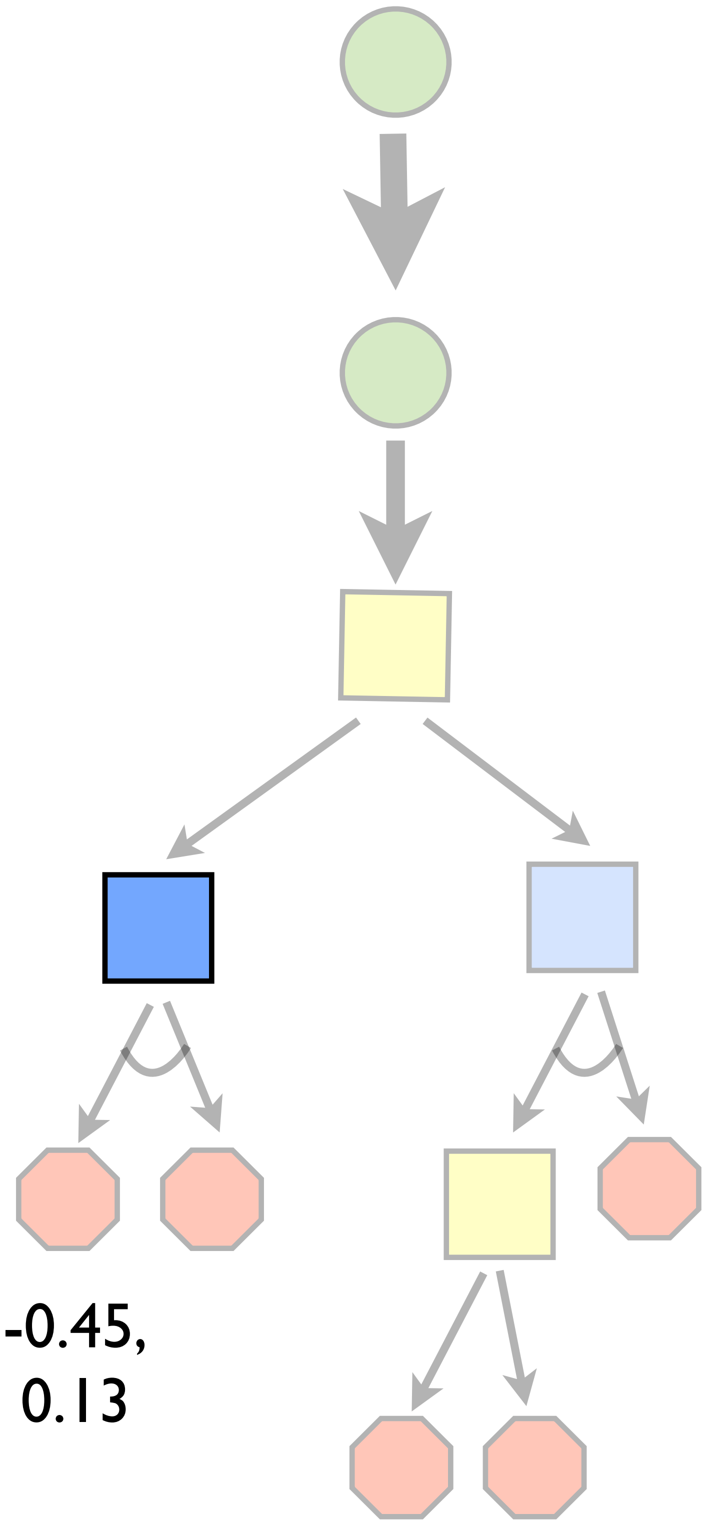
K:0.13





Home

1: Walking the Public Tree



Their Reach Prob:

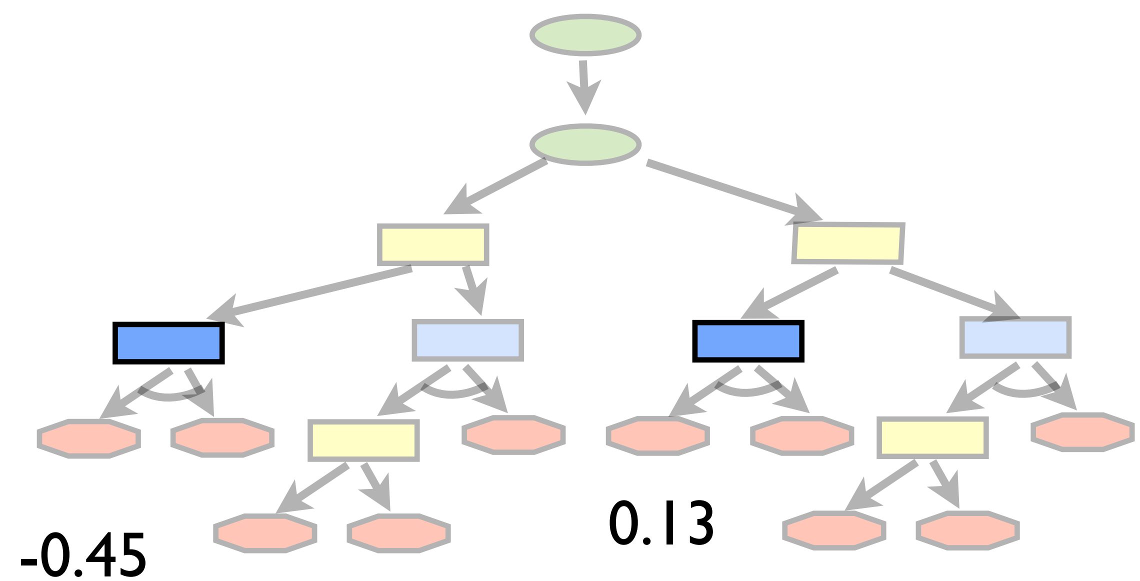
2: 0.5*0.25

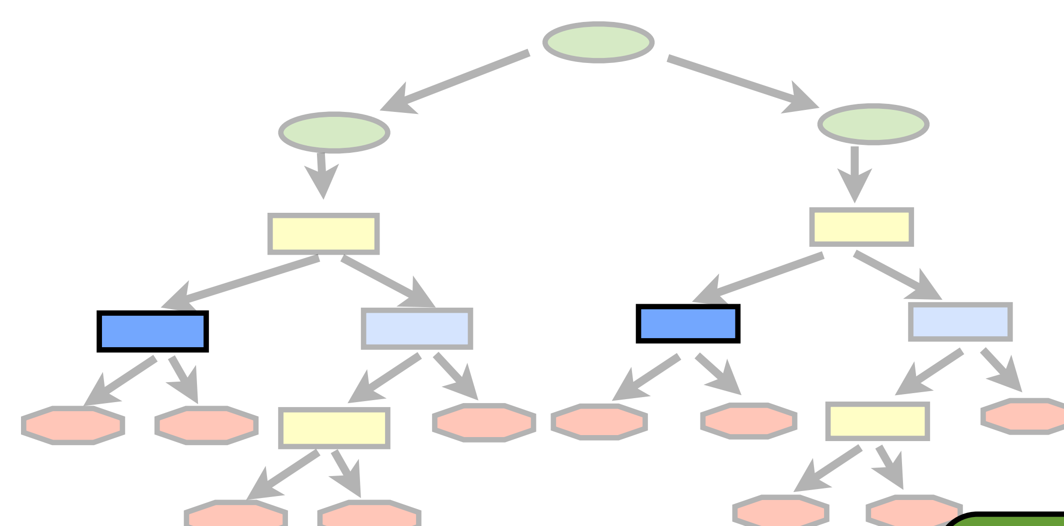
K: 0.5*0.9

My Value:

2: -0.45

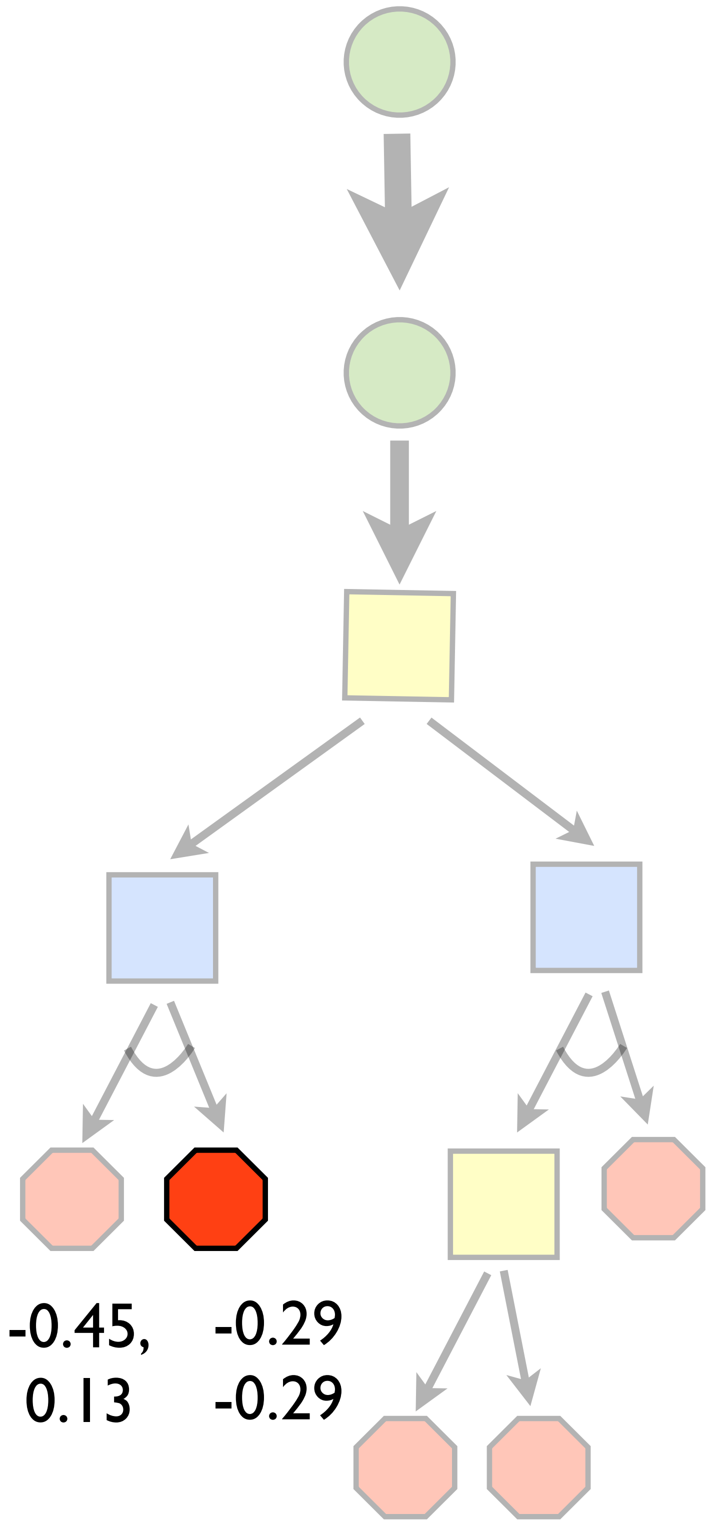
K:0.13





Home

1: Walking the Public Tree



Their Reach Prob:

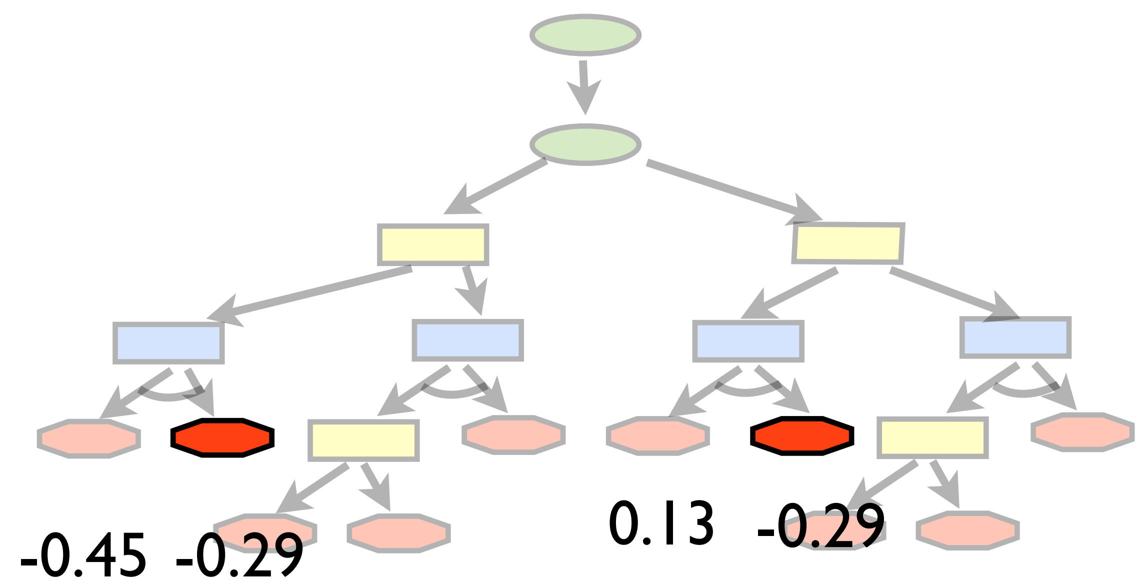
2: 0.5*0.25

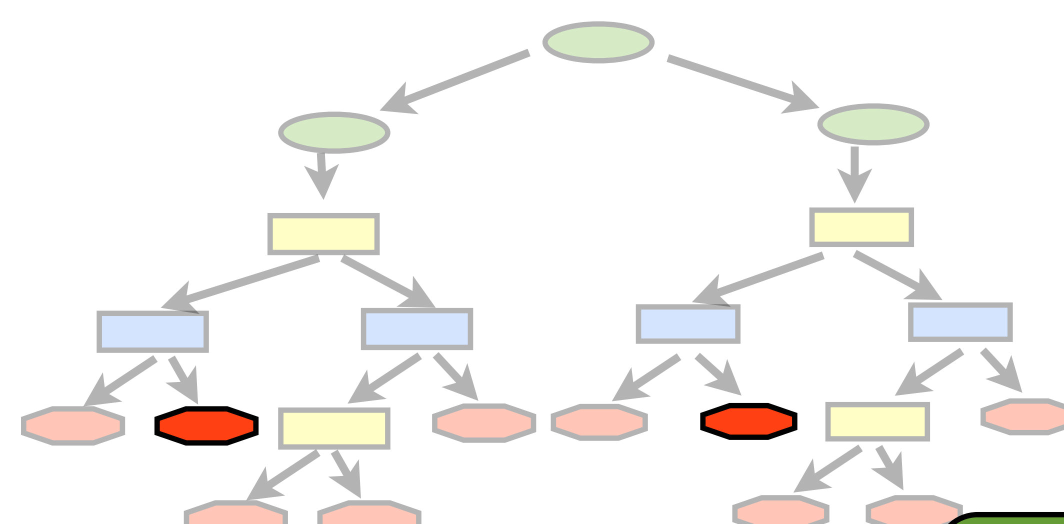
K: 0.5*0.9

My Value:

2: -0.29

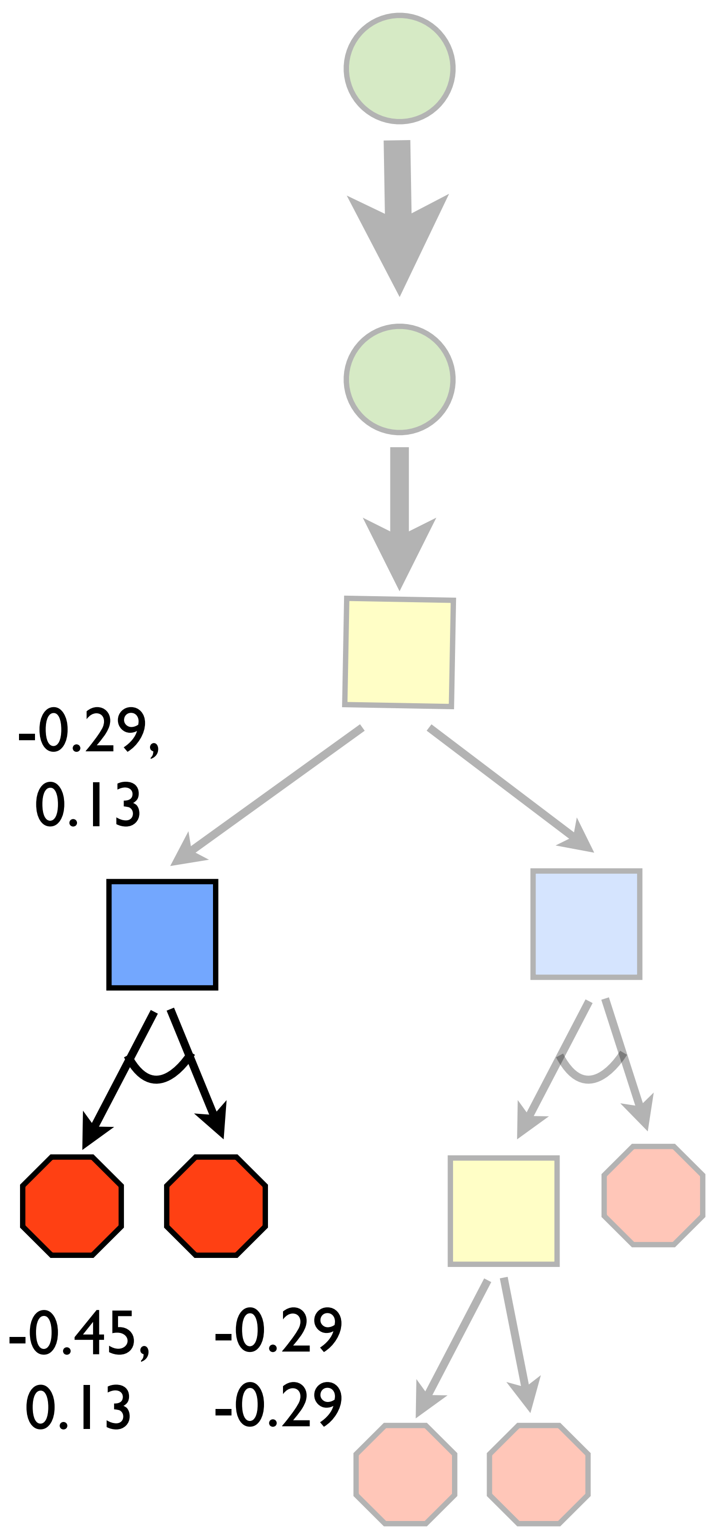
K: -0.29





Home

1: Walking the Public Tree



Their Reach Prob:

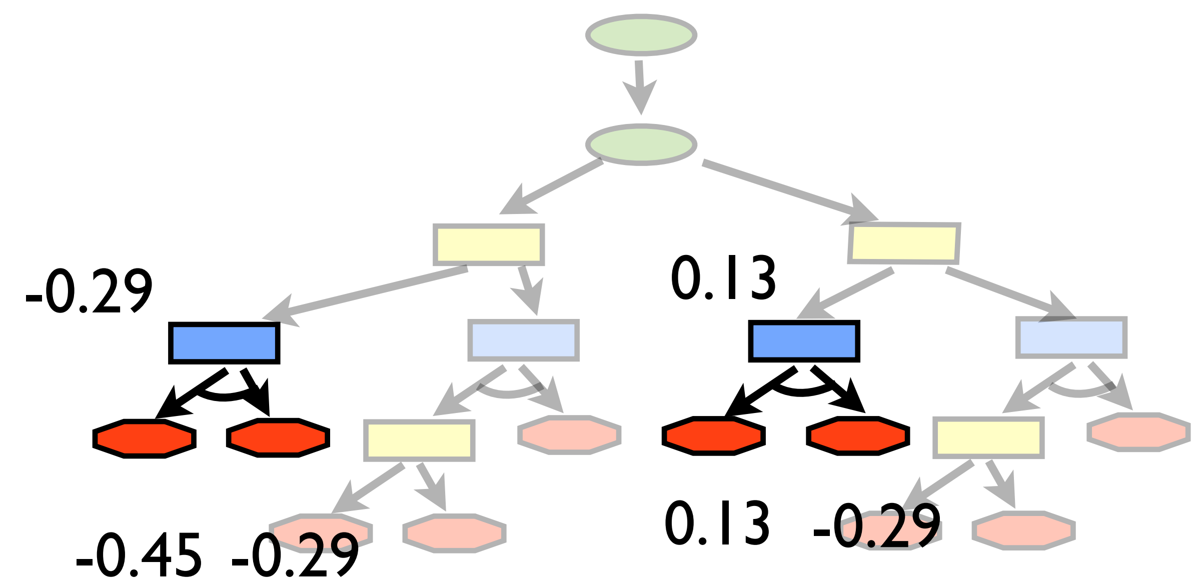
2: 0.5*0.25

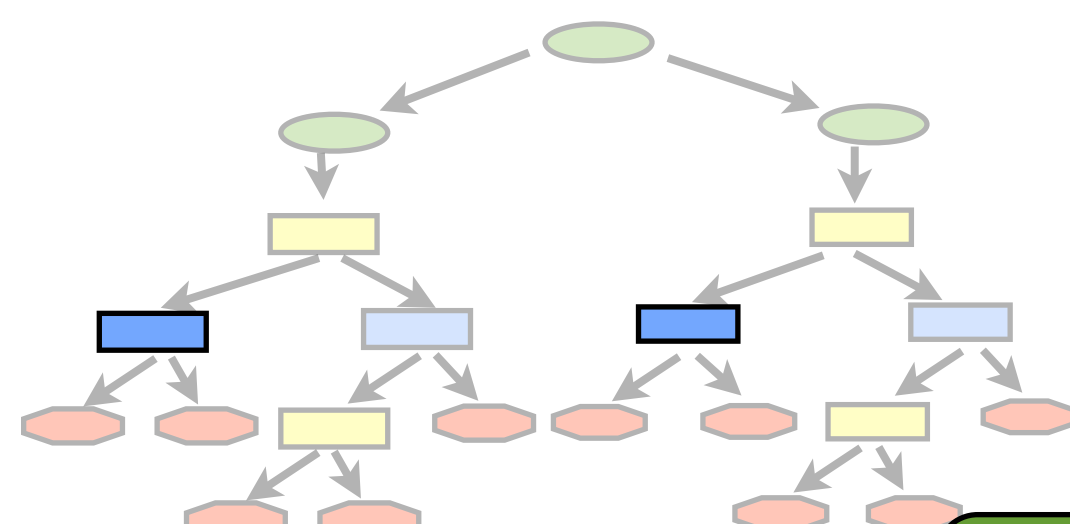
K: 0.5*0.9

My Value:

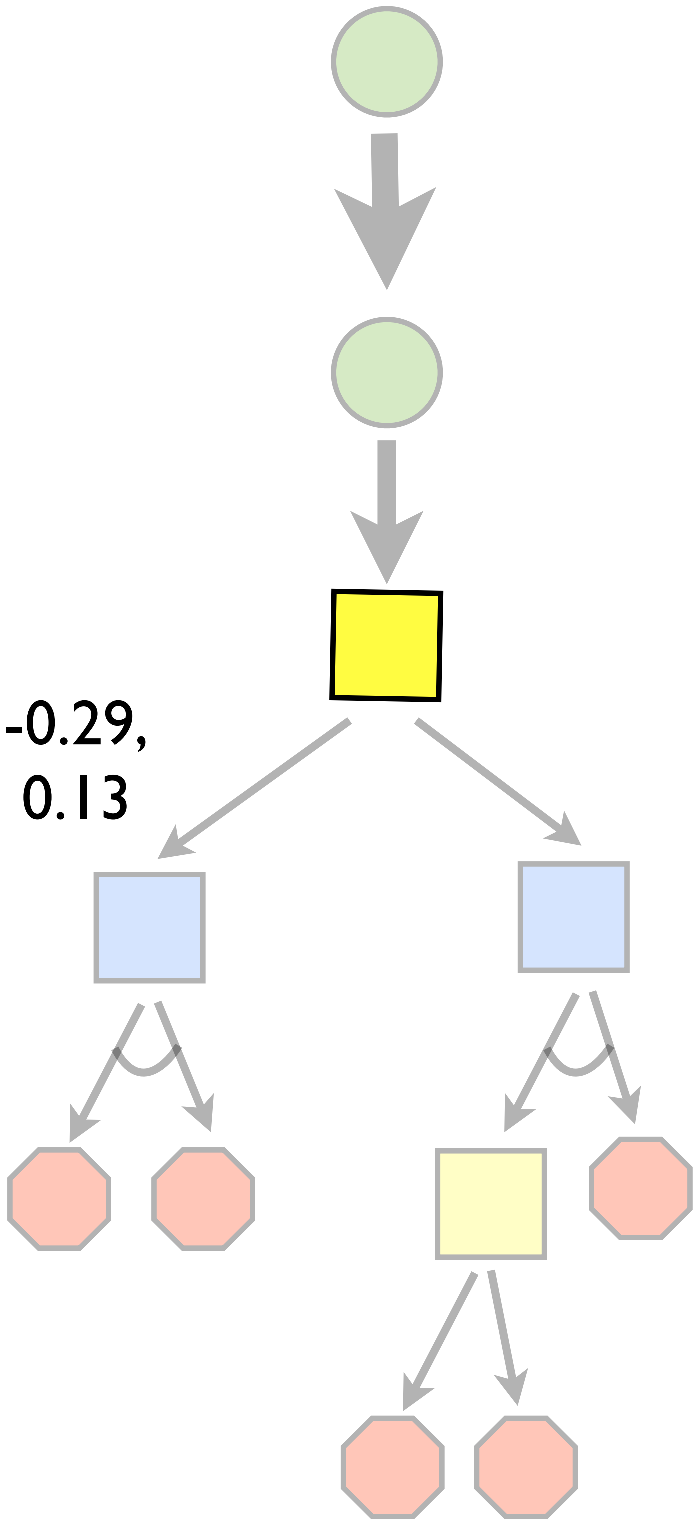
2: -0.29

K: 0.13





1: Walking the Public Tree



Their Reach Prob:

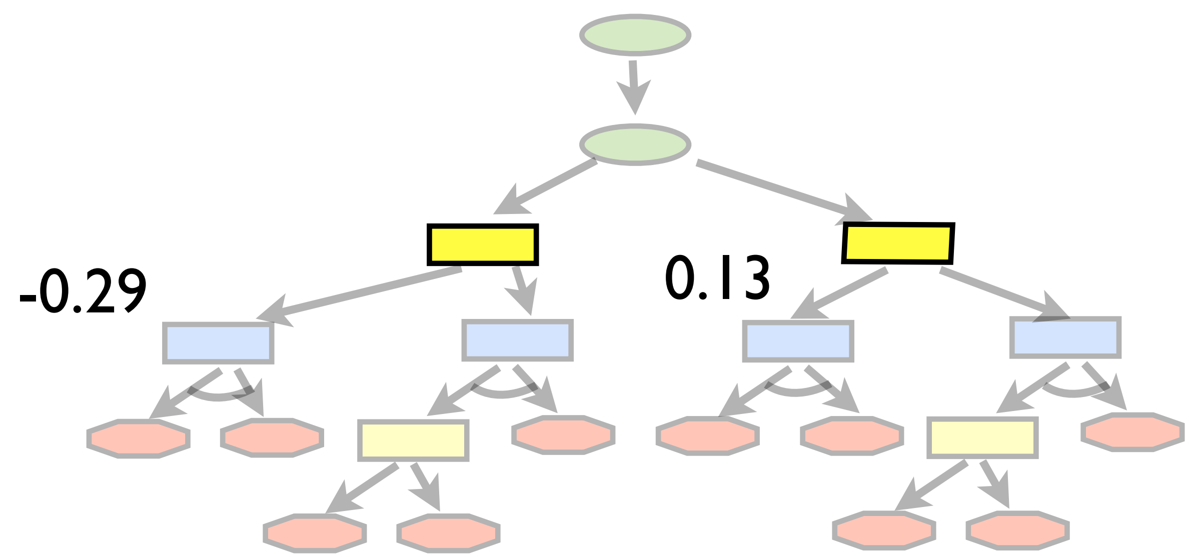
2: 0.5

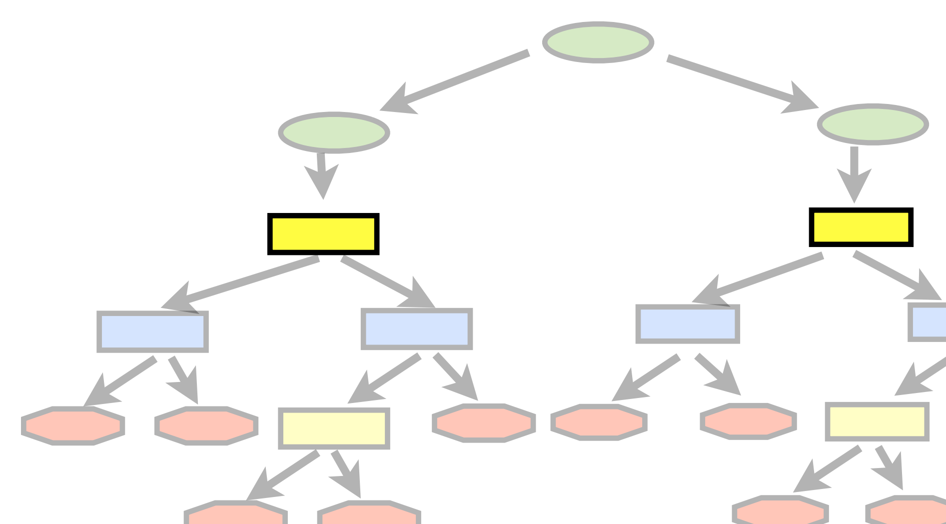
K: 0.5

My Value:

2: -0.29

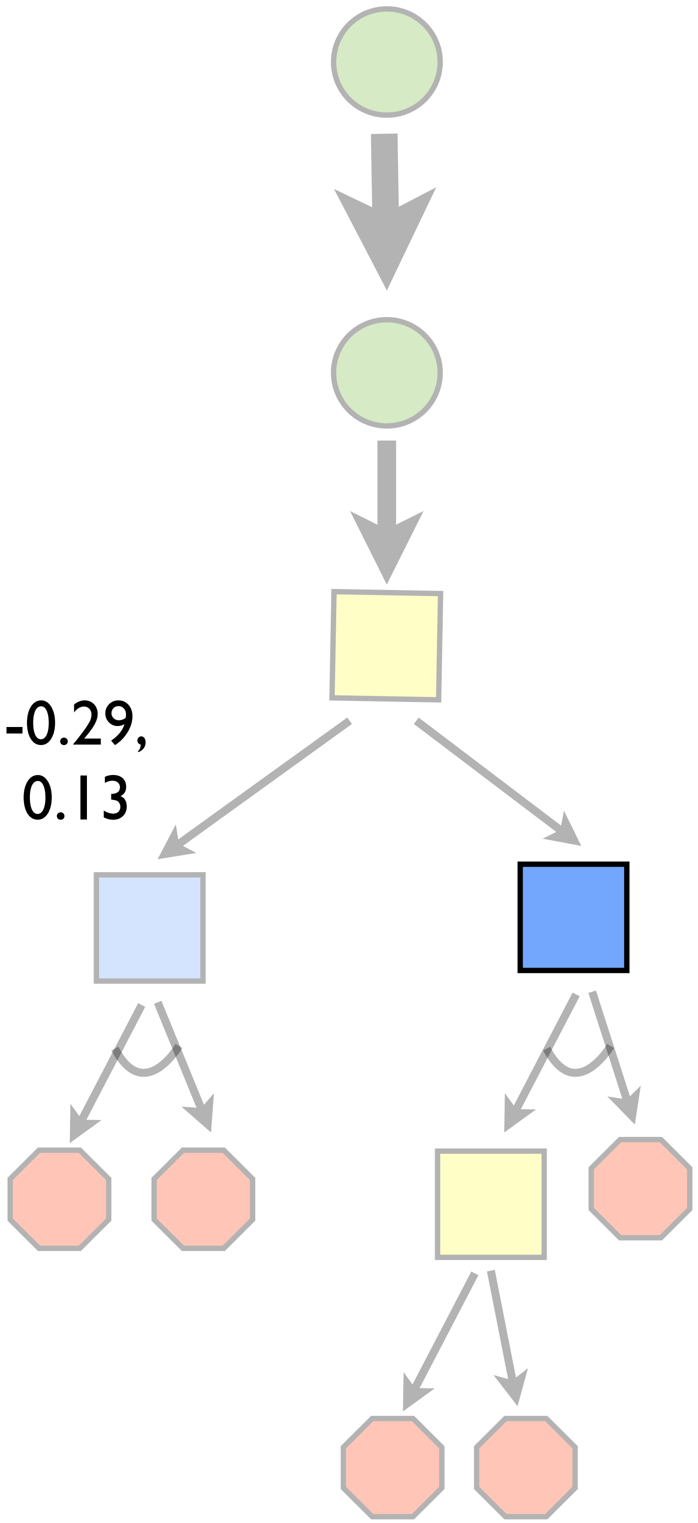
K: 0.13





Home

1: Walking the Public Tree



Their Reach Prob:

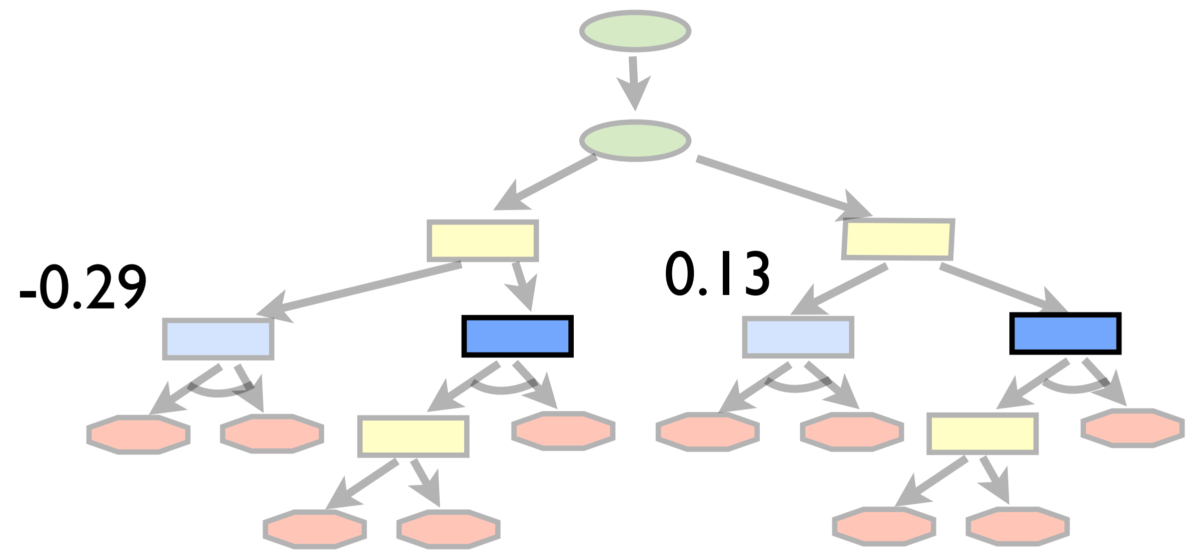
2: 0.5*0.75

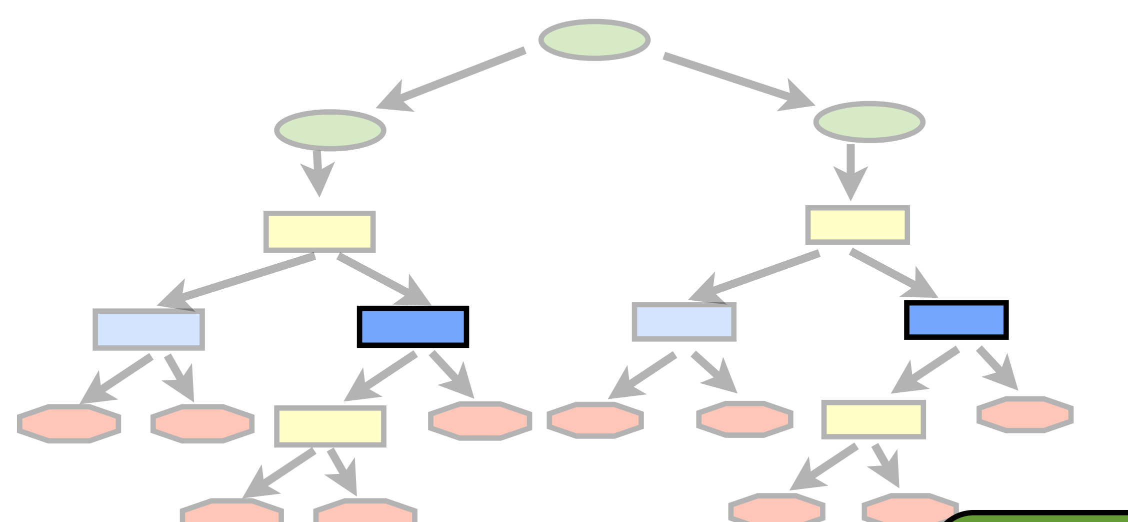
K: 0.5*0.1

My Value:

2:

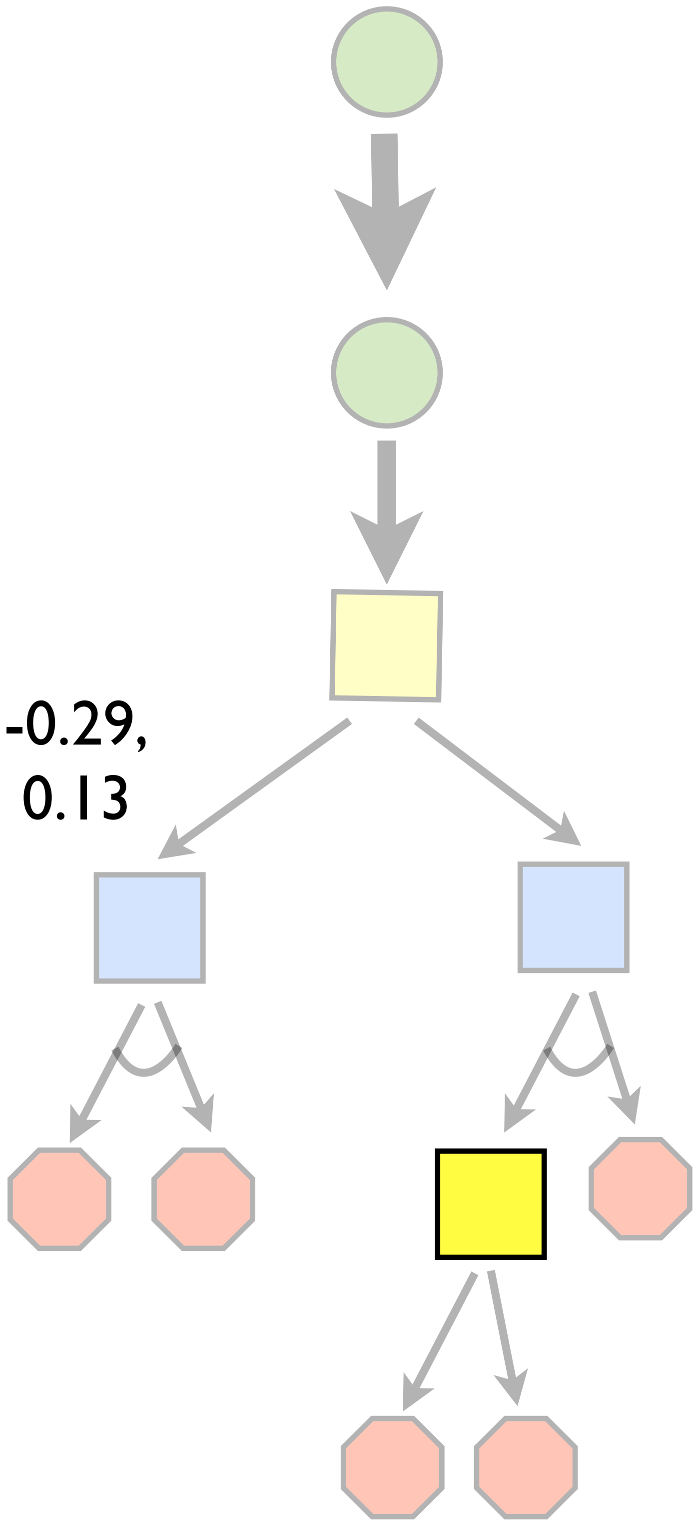
K:





Home

1: Walking the Public Tree



Their Reach Prob:

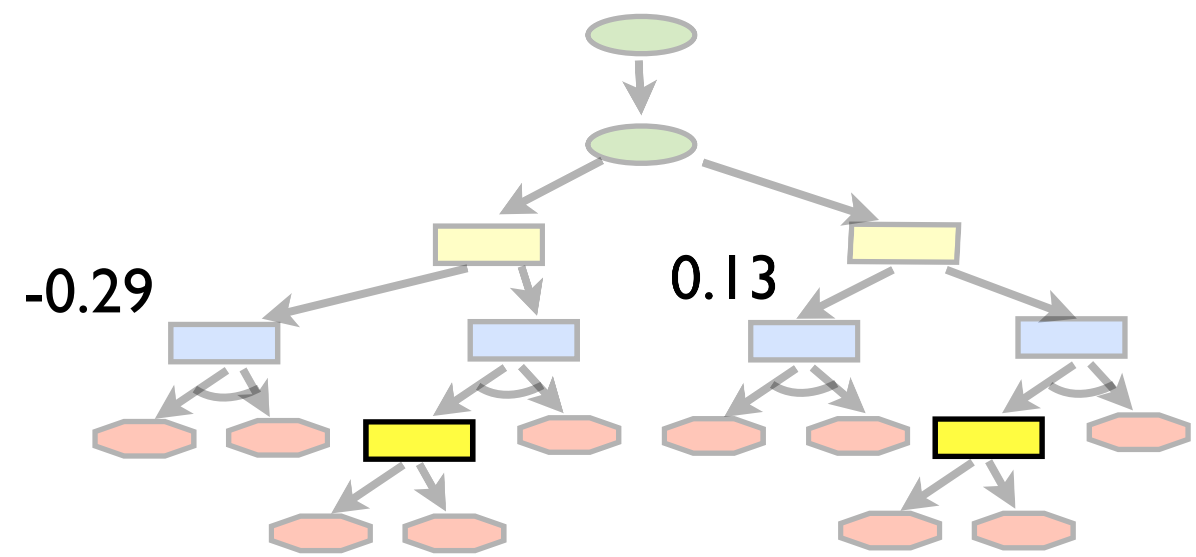
2: 0.5*0.75

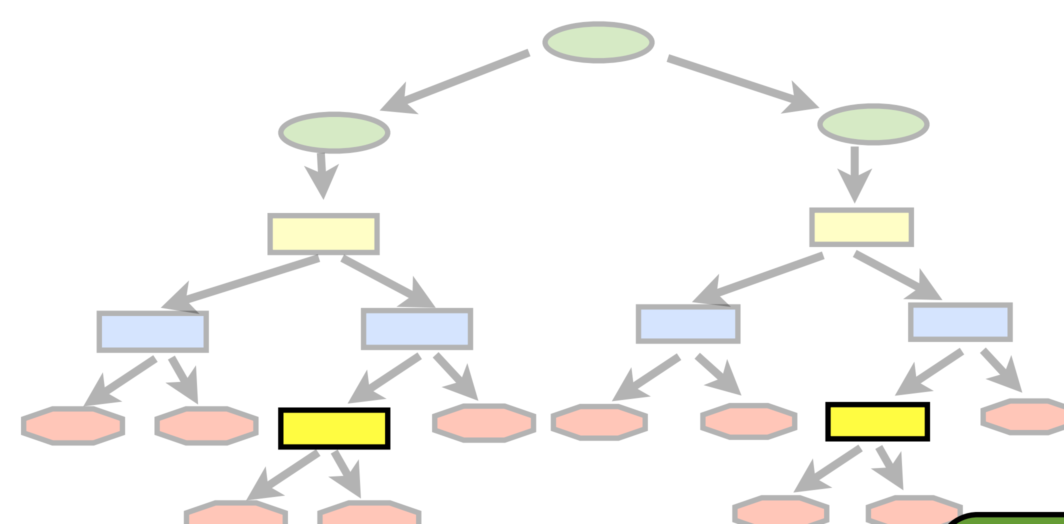
K: 0.5*0.1

My Value:

2:

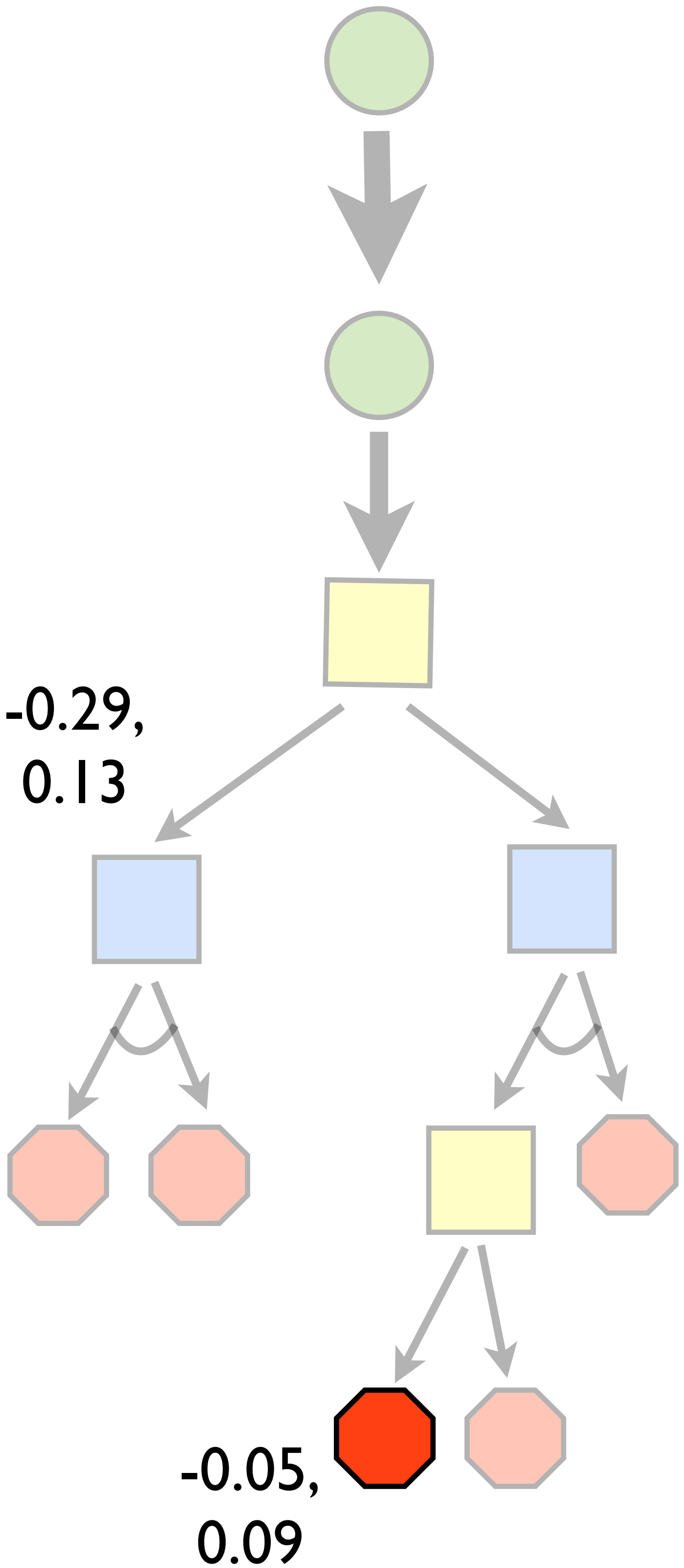
K:





Home

1: Walking the Public Tree



Their Reach Prob:

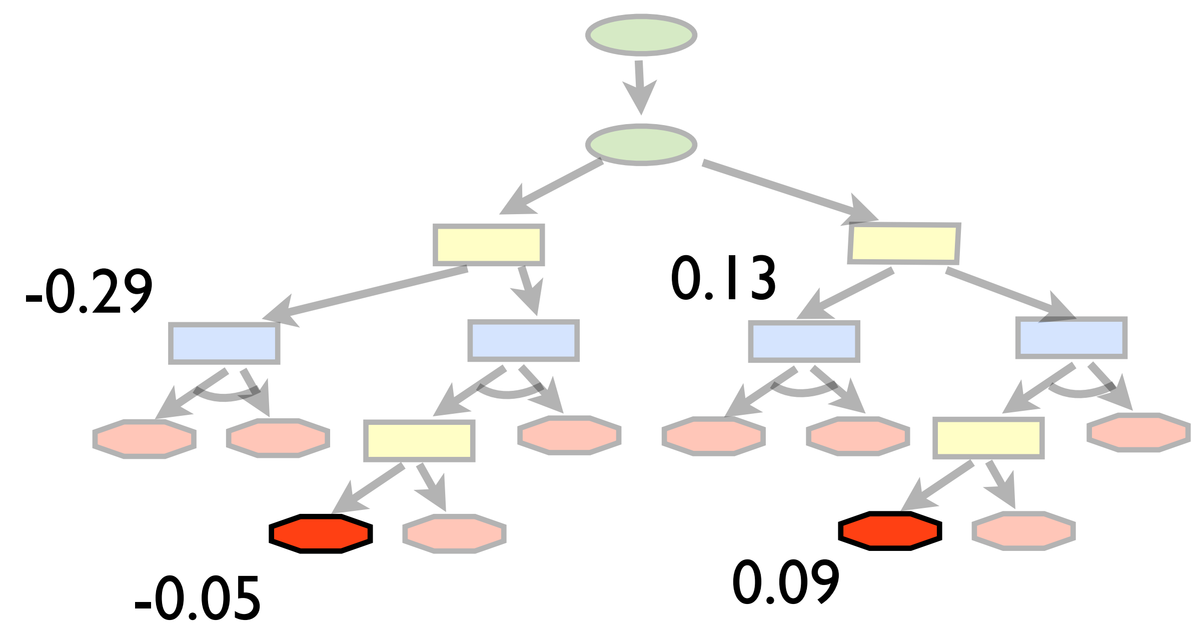
2: 0.5*0.75

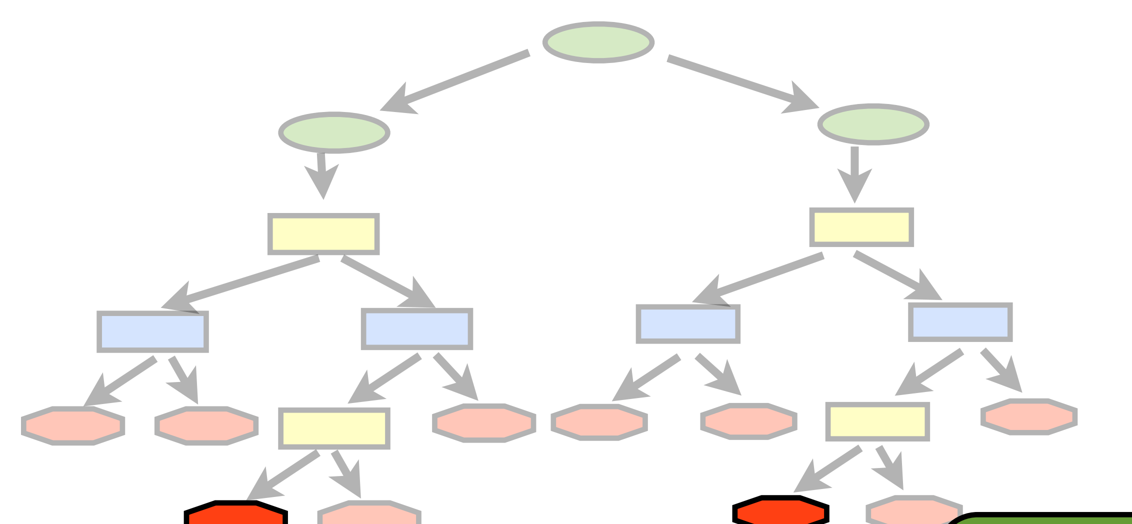
K: 0.5*0.1

My Value:

2: -0.05

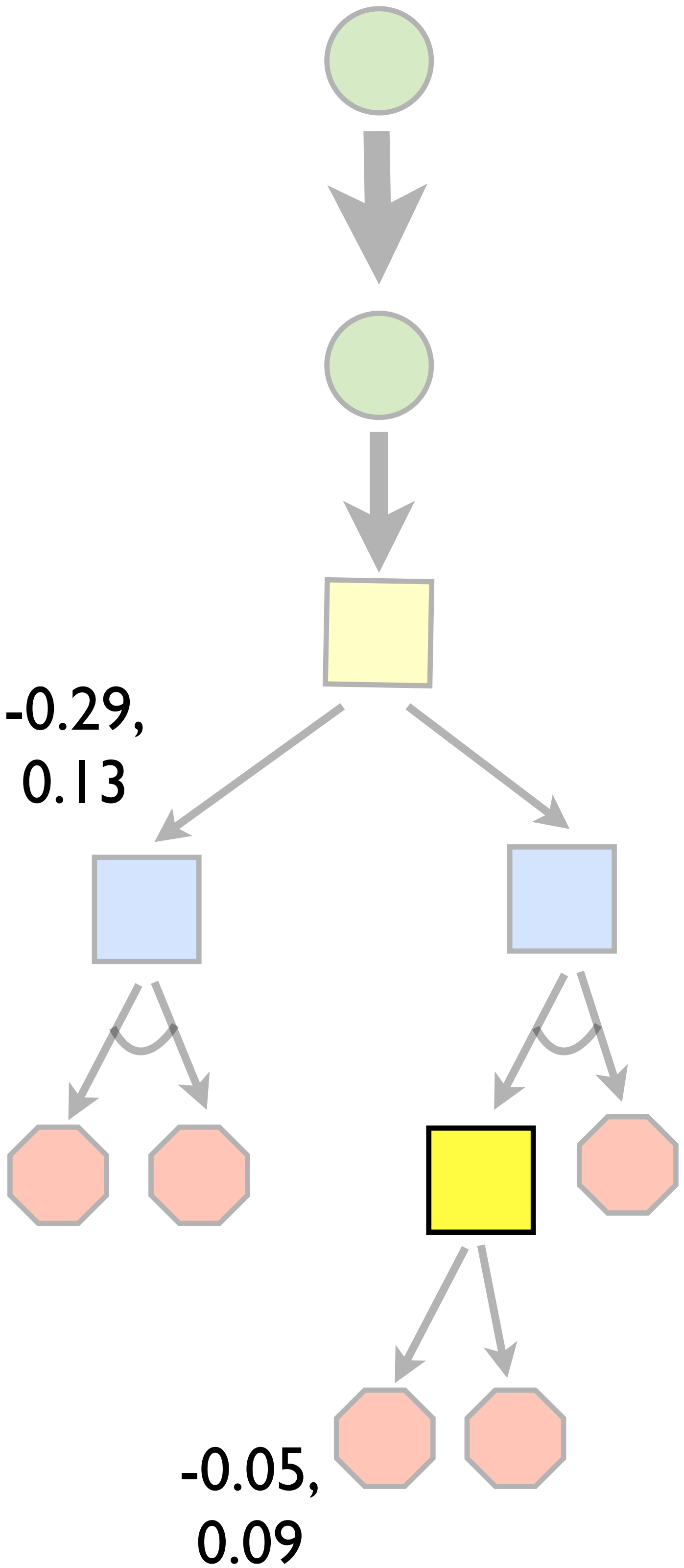
K: 0.09





Home

1: Walking the Public Tree



Their Reach Prob:

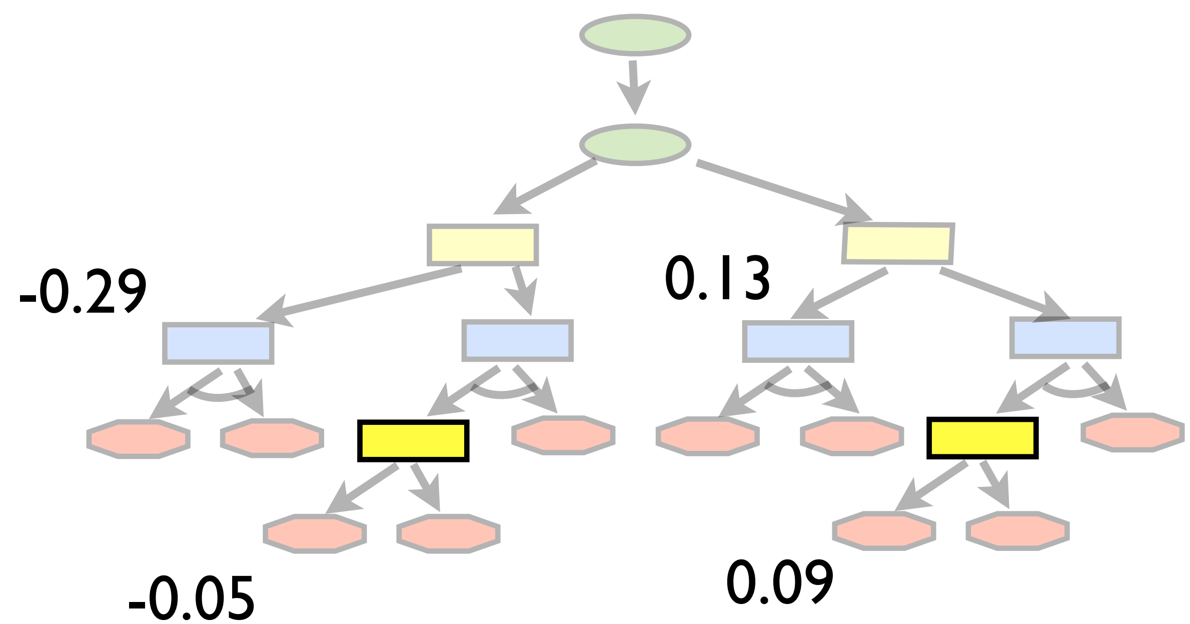
2: 0.5*0.75

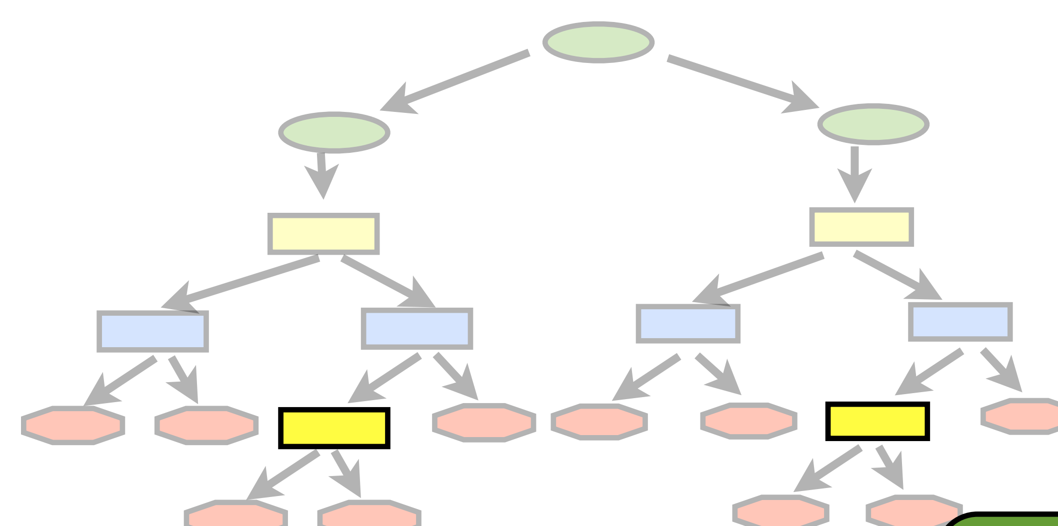
K: 0.5*0.1

My Value:

2: -0.05

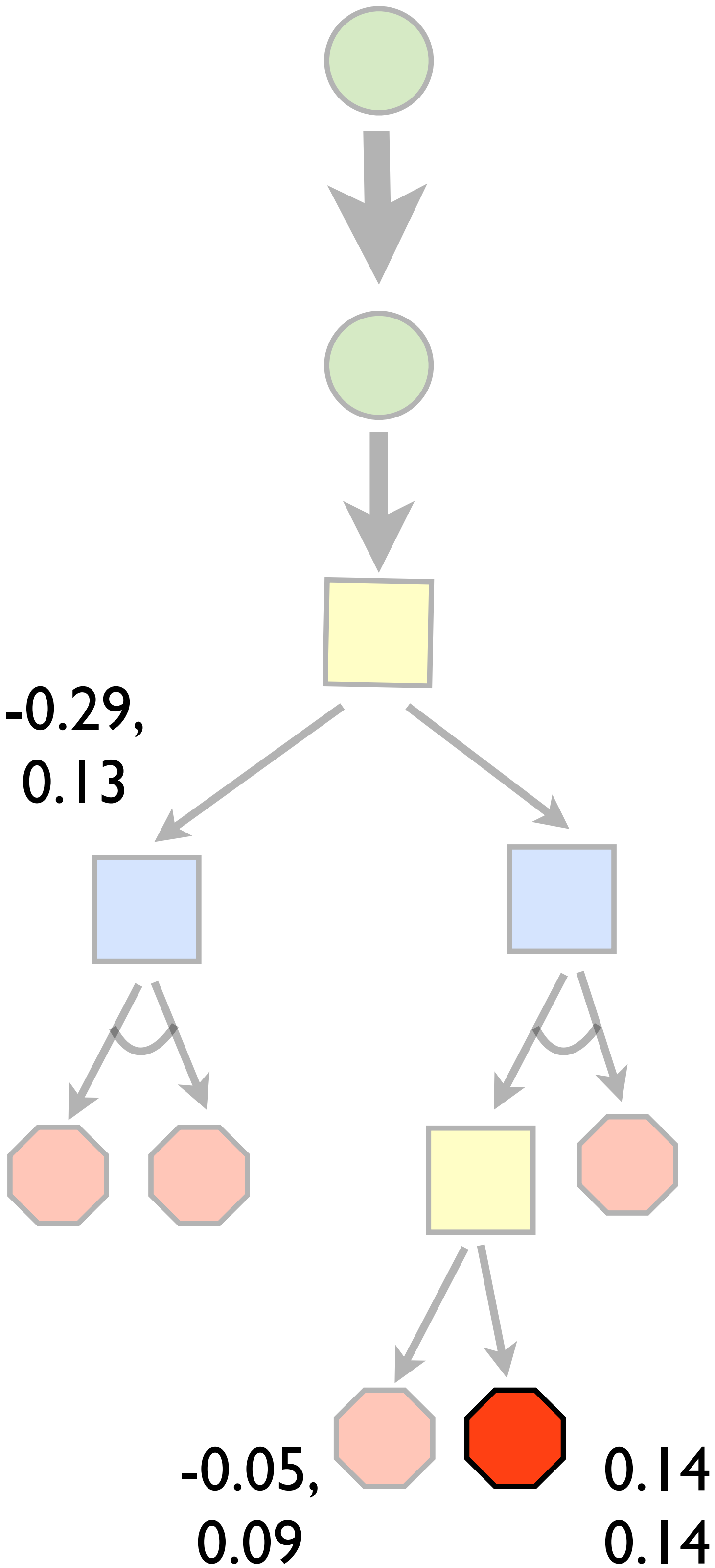
K: 0.09





Home

1: Walking the Public Tree

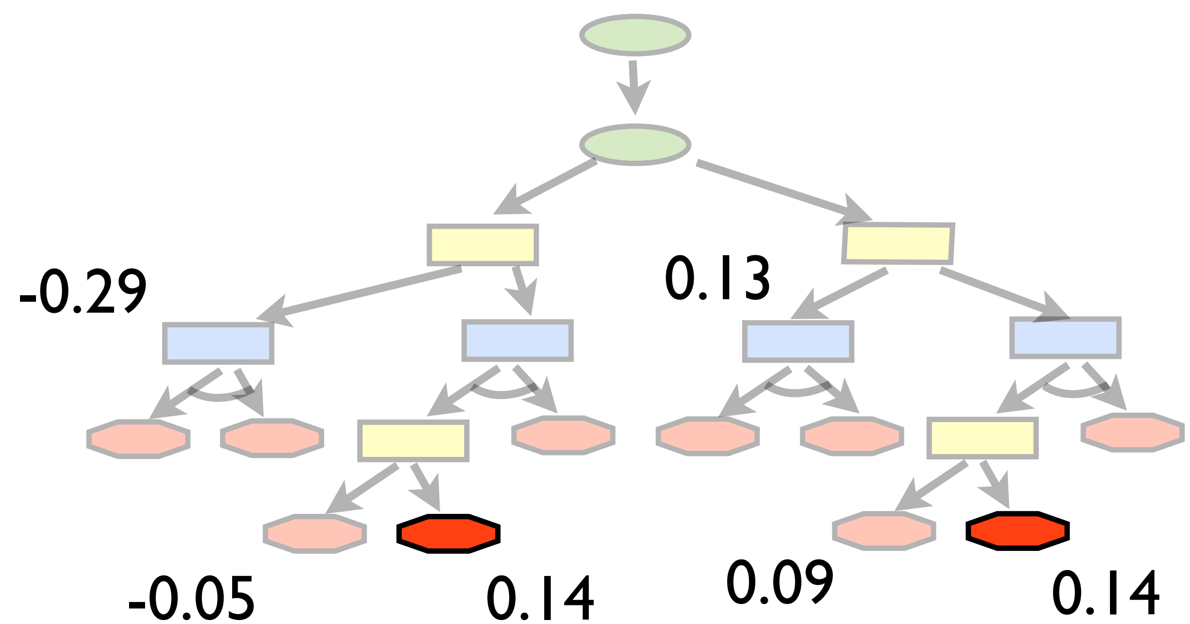


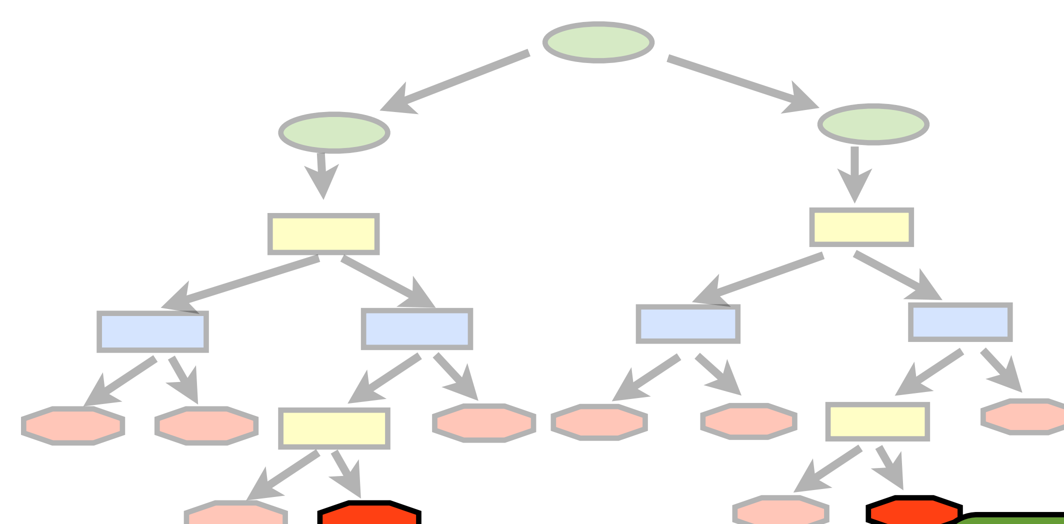
Their Reach Prob:

2: 0.5*0.75
K: 0.5*0.1

My Value:

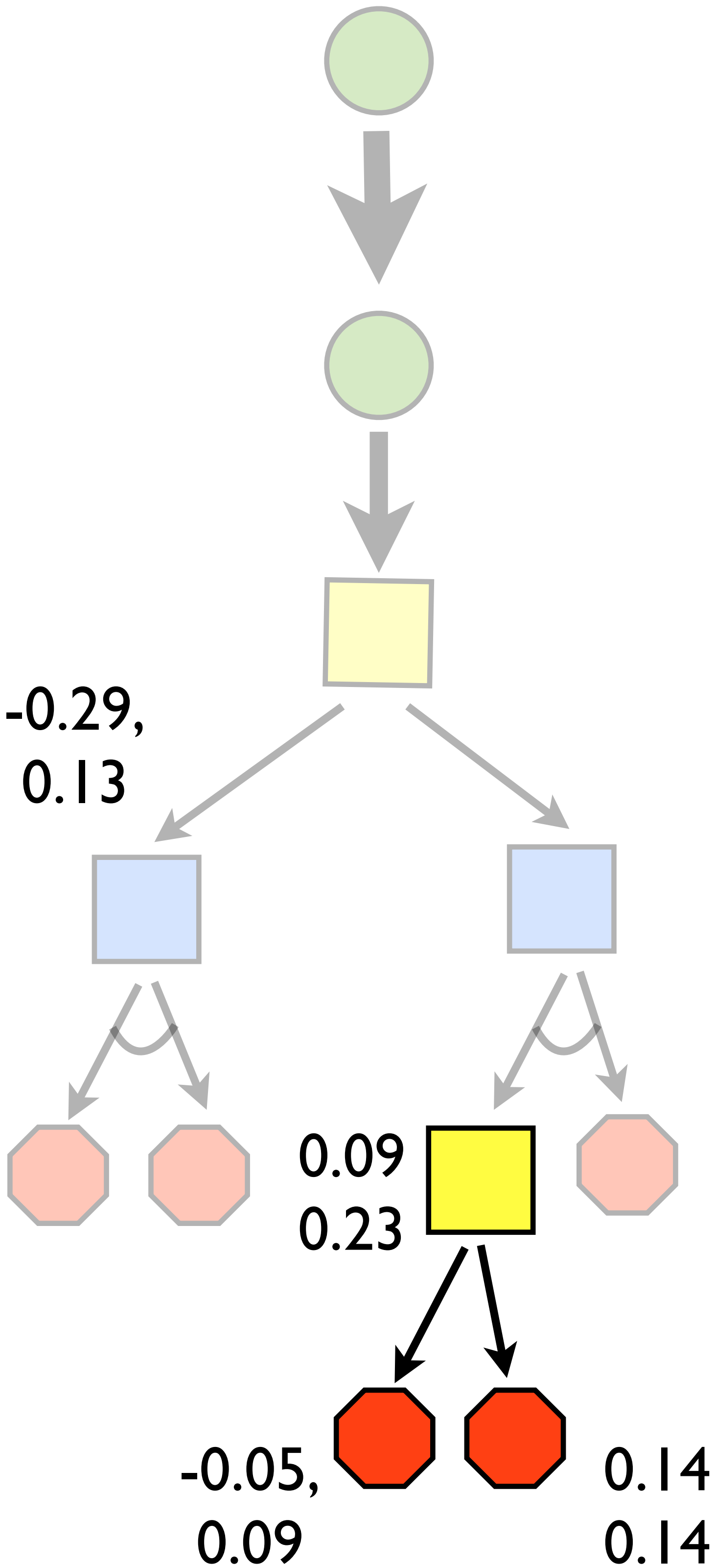
2: 0.14
K: 0.14





Home

1: Walking the Public Tree



Their Reach Prob:

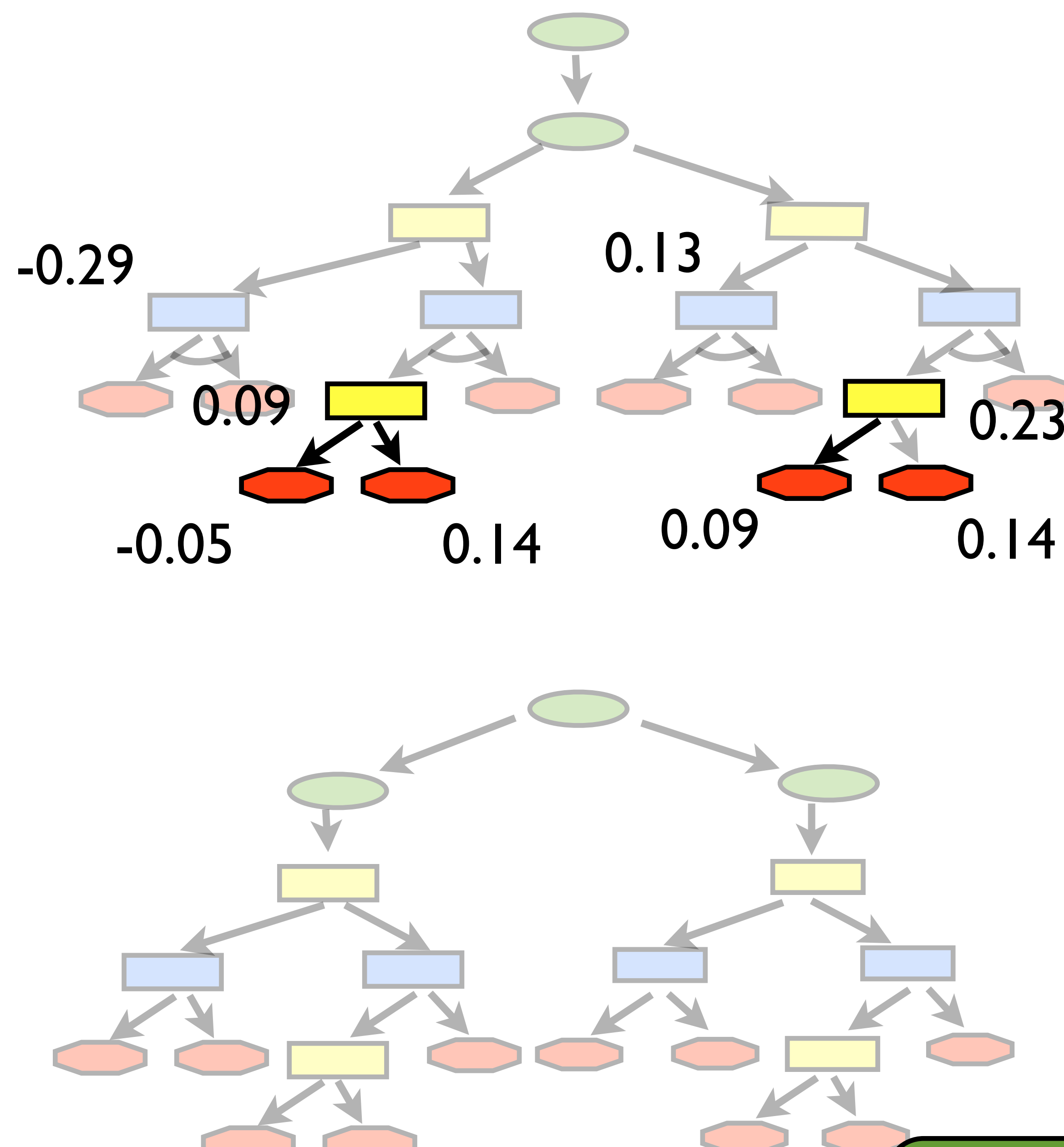
2: 0.5*0.75

K: 0.5*0.1

My Value:

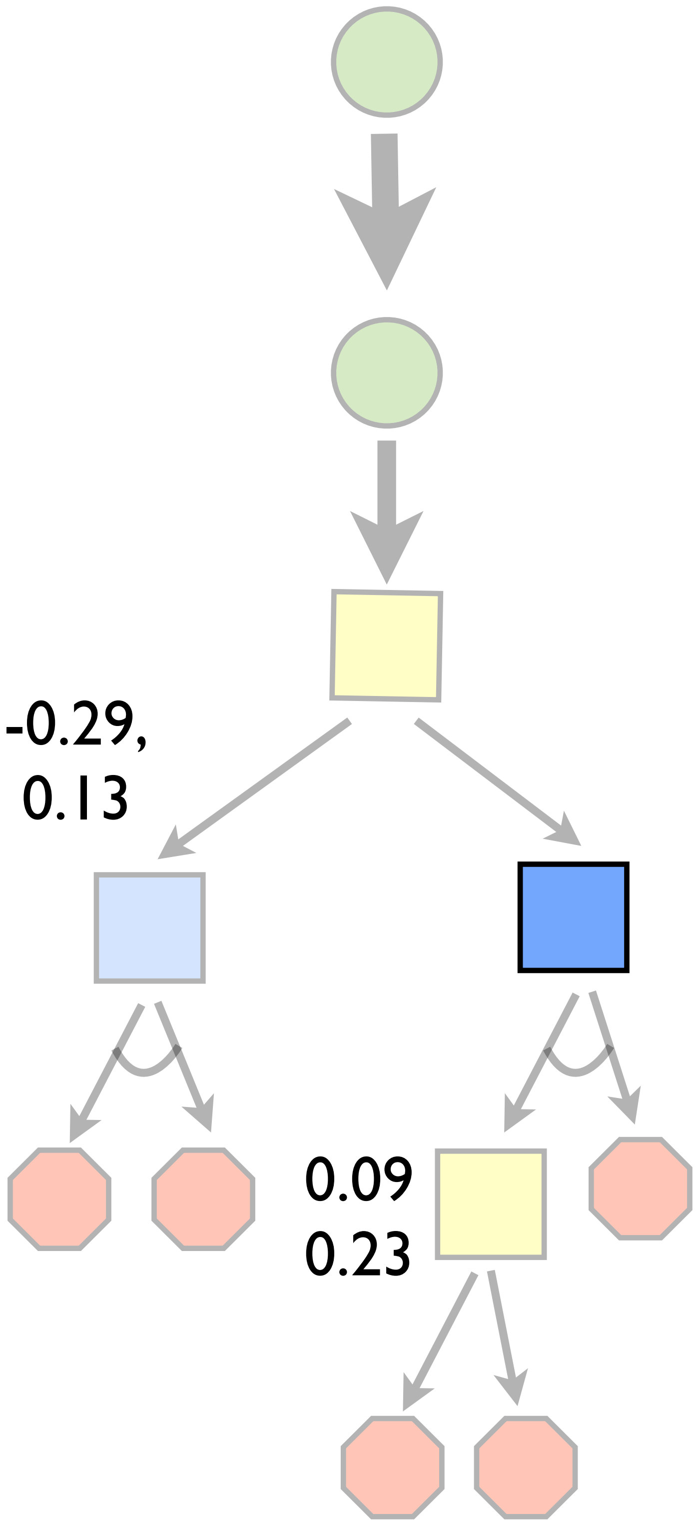
2: 0.14

K: 0.14



Home

1: Walking the Public Tree



Their Reach Prob:

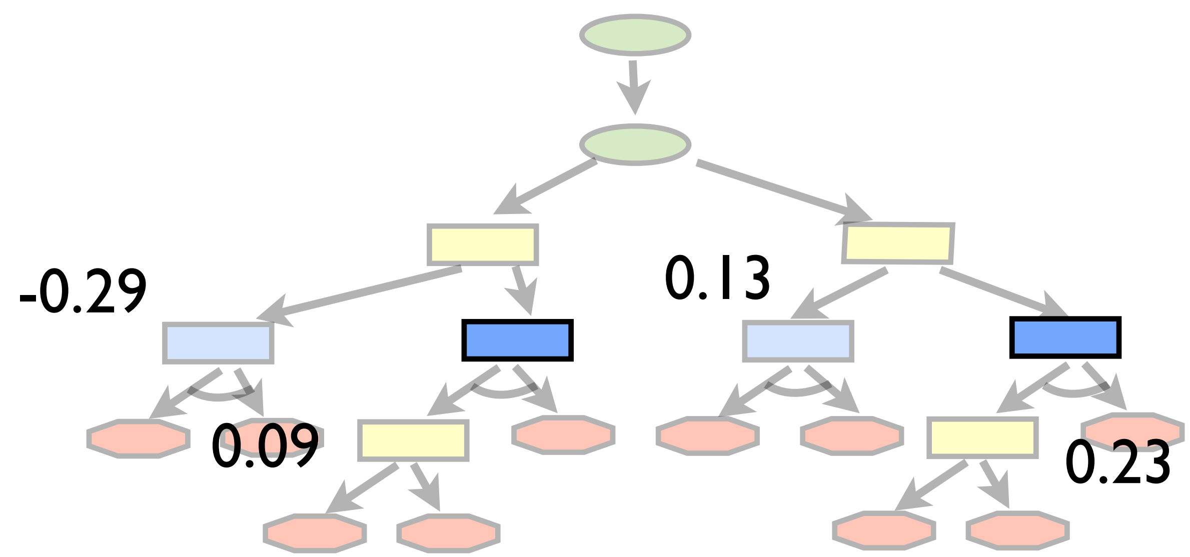
2: 0.5*0.75

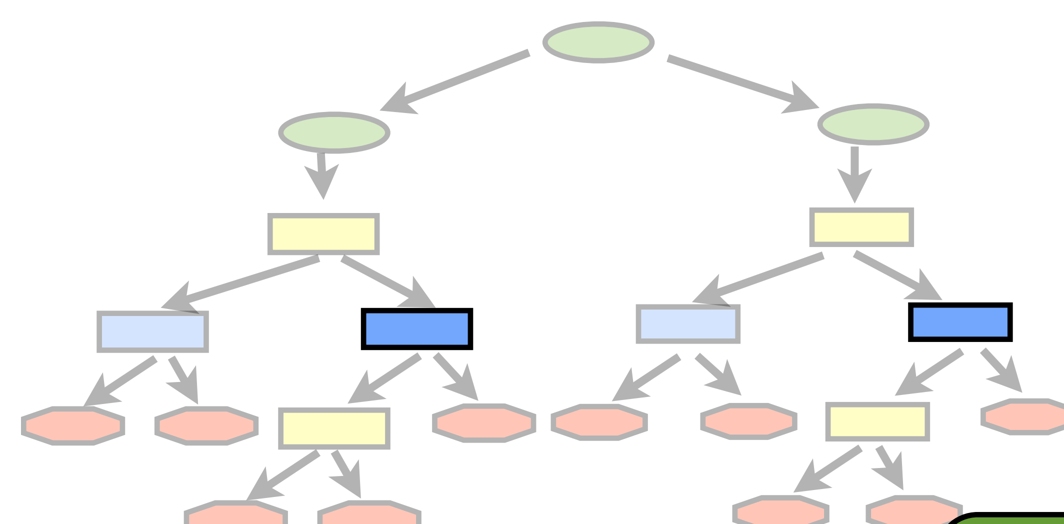
K: 0.5*0.1

My Value:

2: 0.14

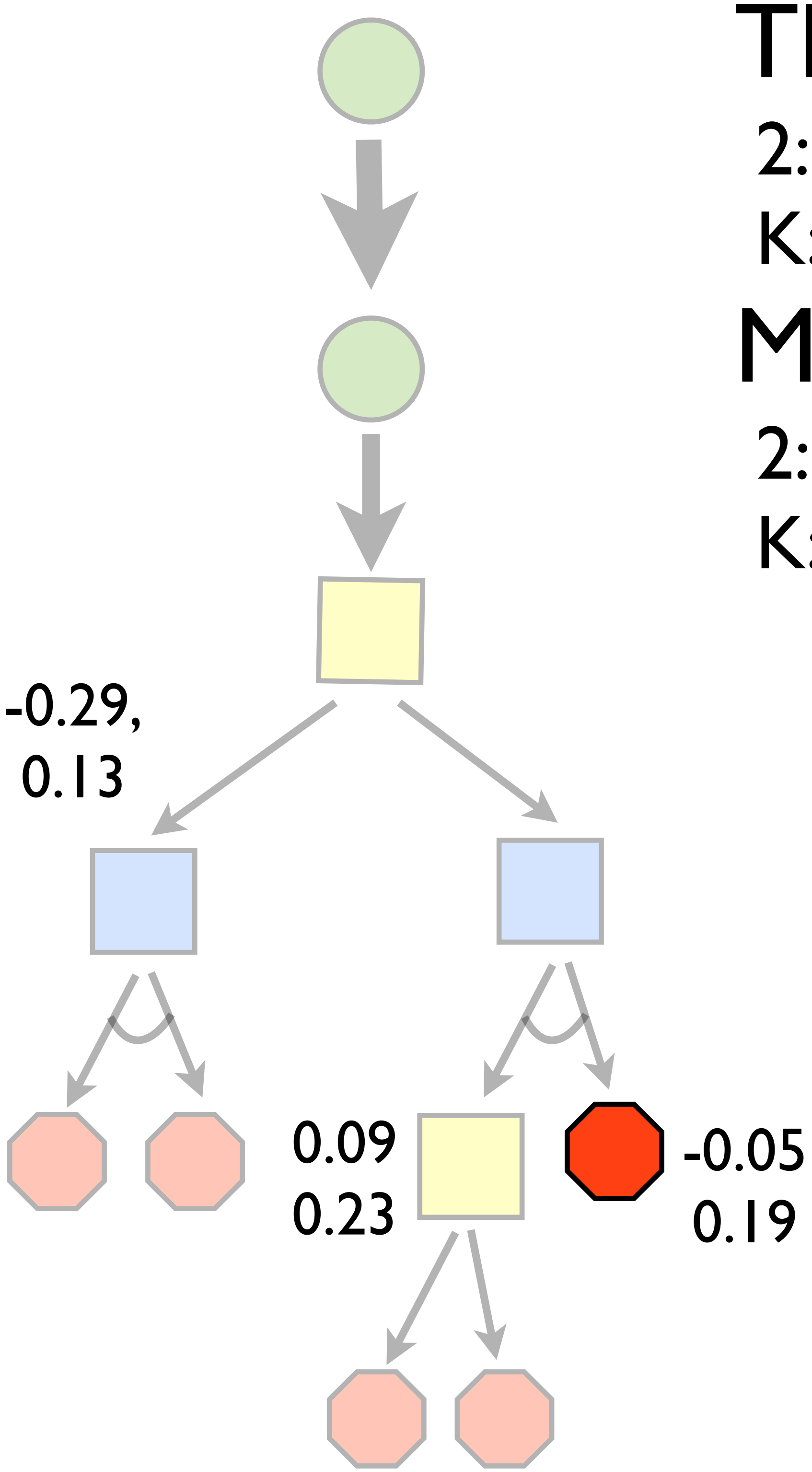
K: 0.14





Home

1: Walking the Public Tree



Their Reach Prob:

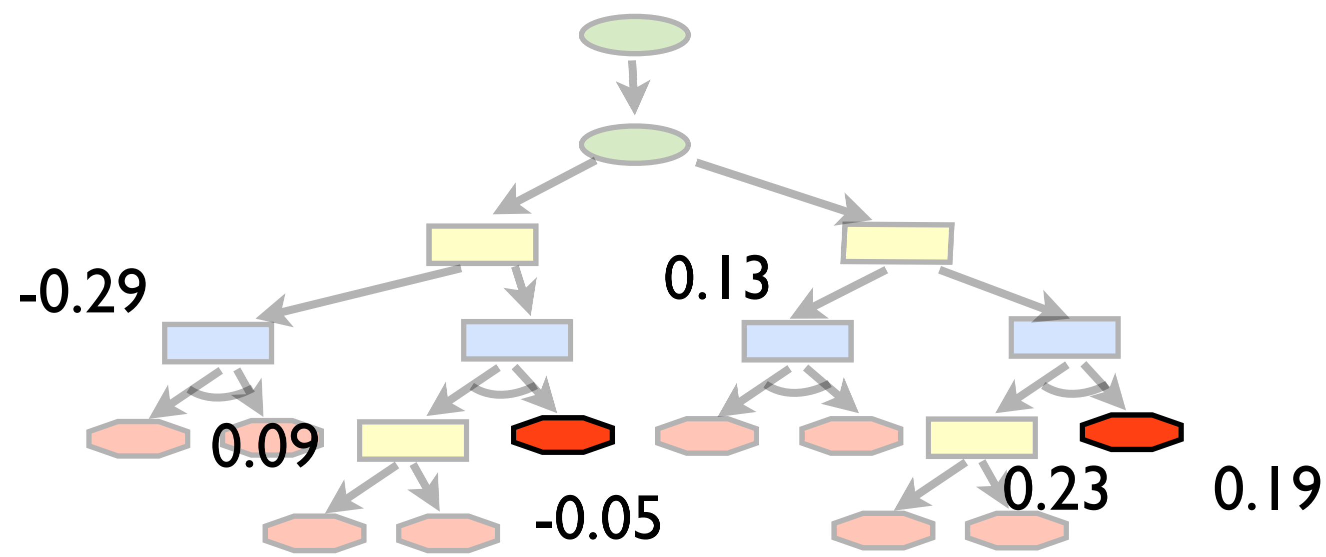
2: 0.5*0.75

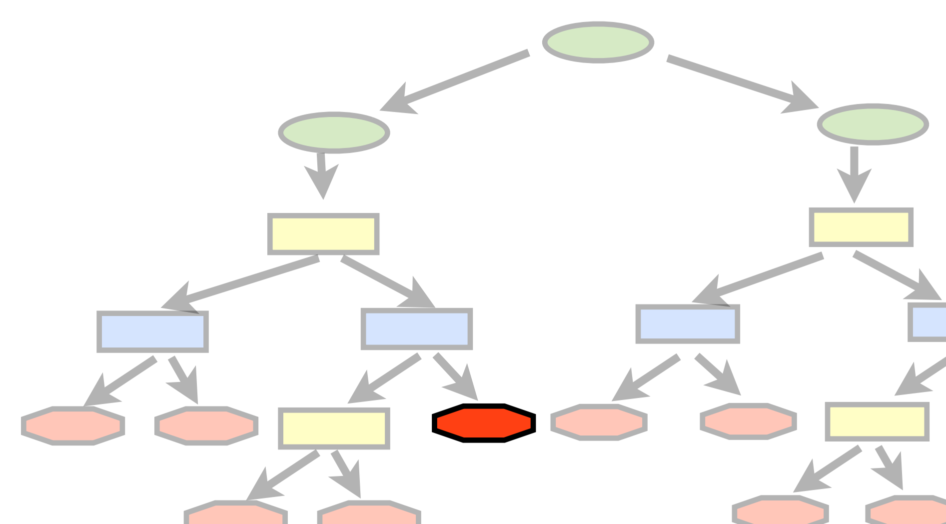
K: 0.5*0.1

My Value:

2: -0.05

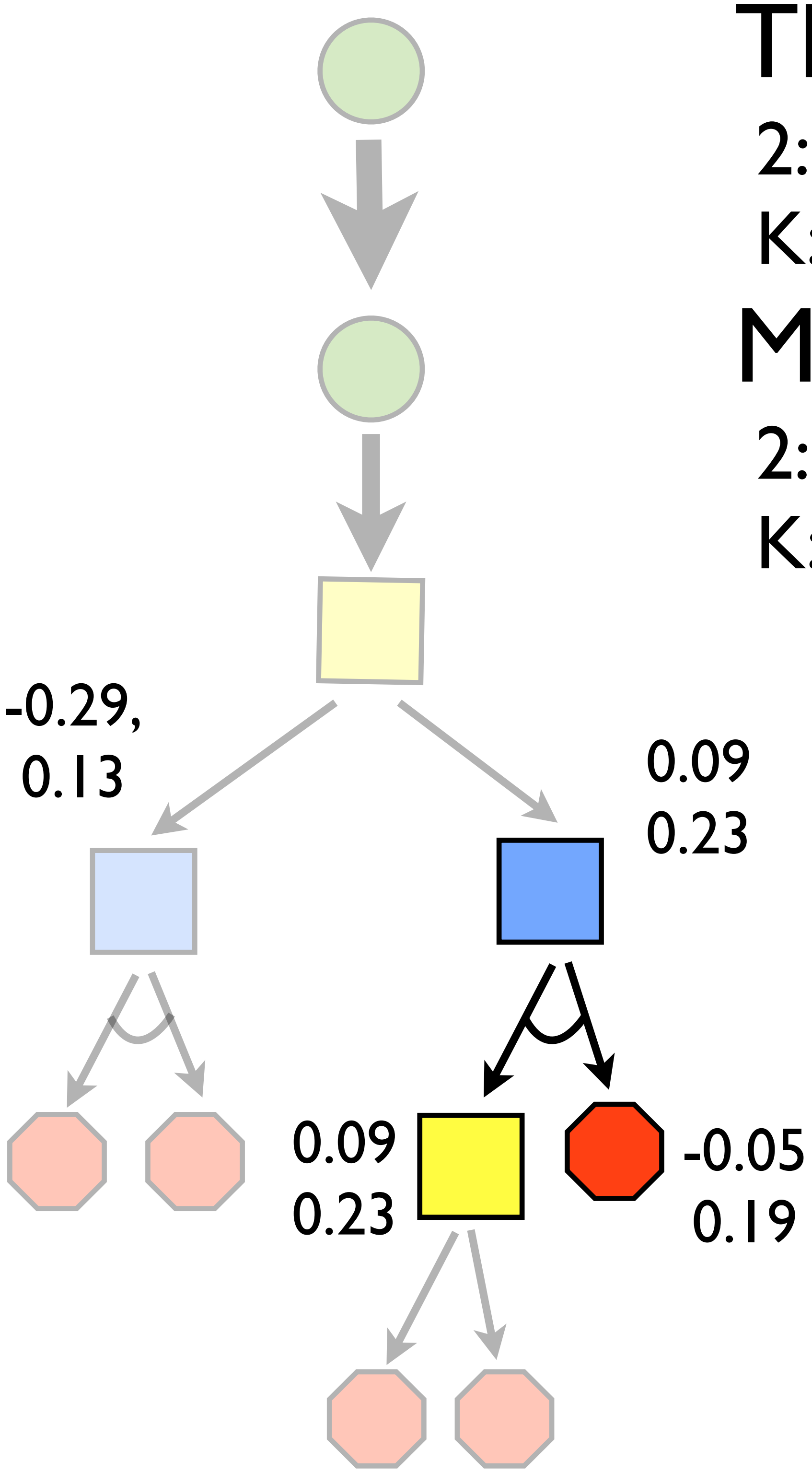
K: 0.19





Home

1: Walking the Public Tree

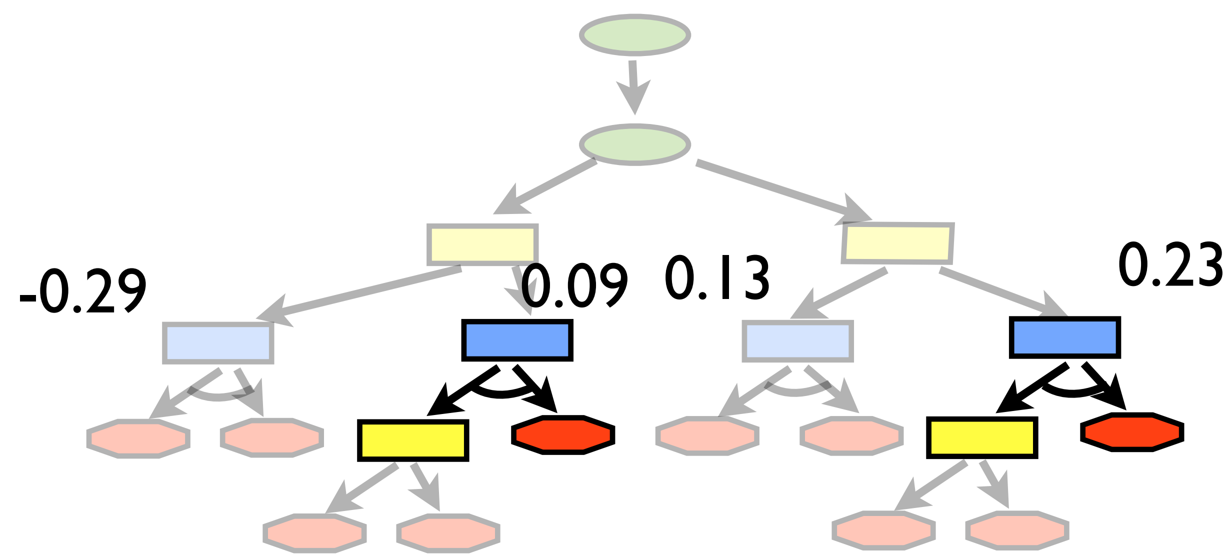


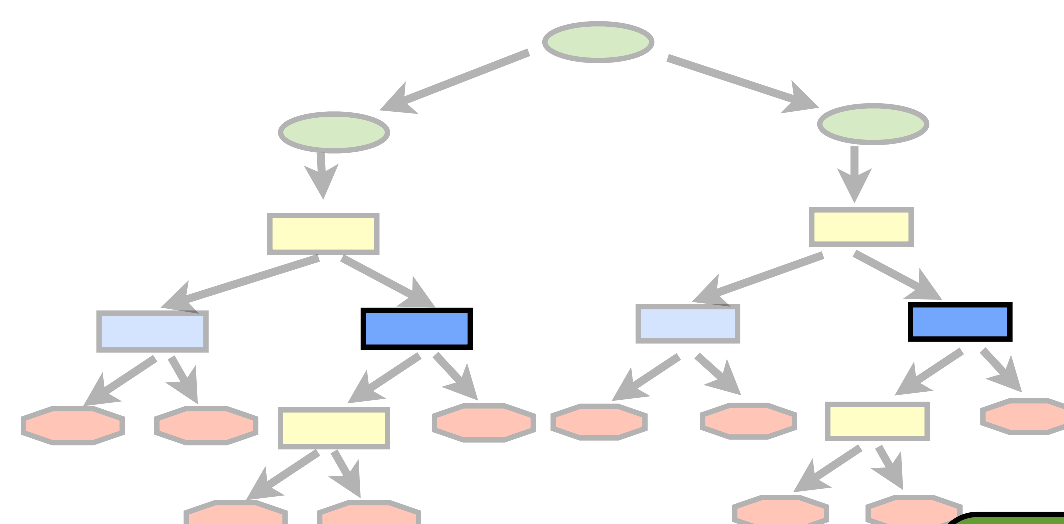
Their Reach Prob:

2: 0.5*0.75
K: 0.5*0.1

My Value:

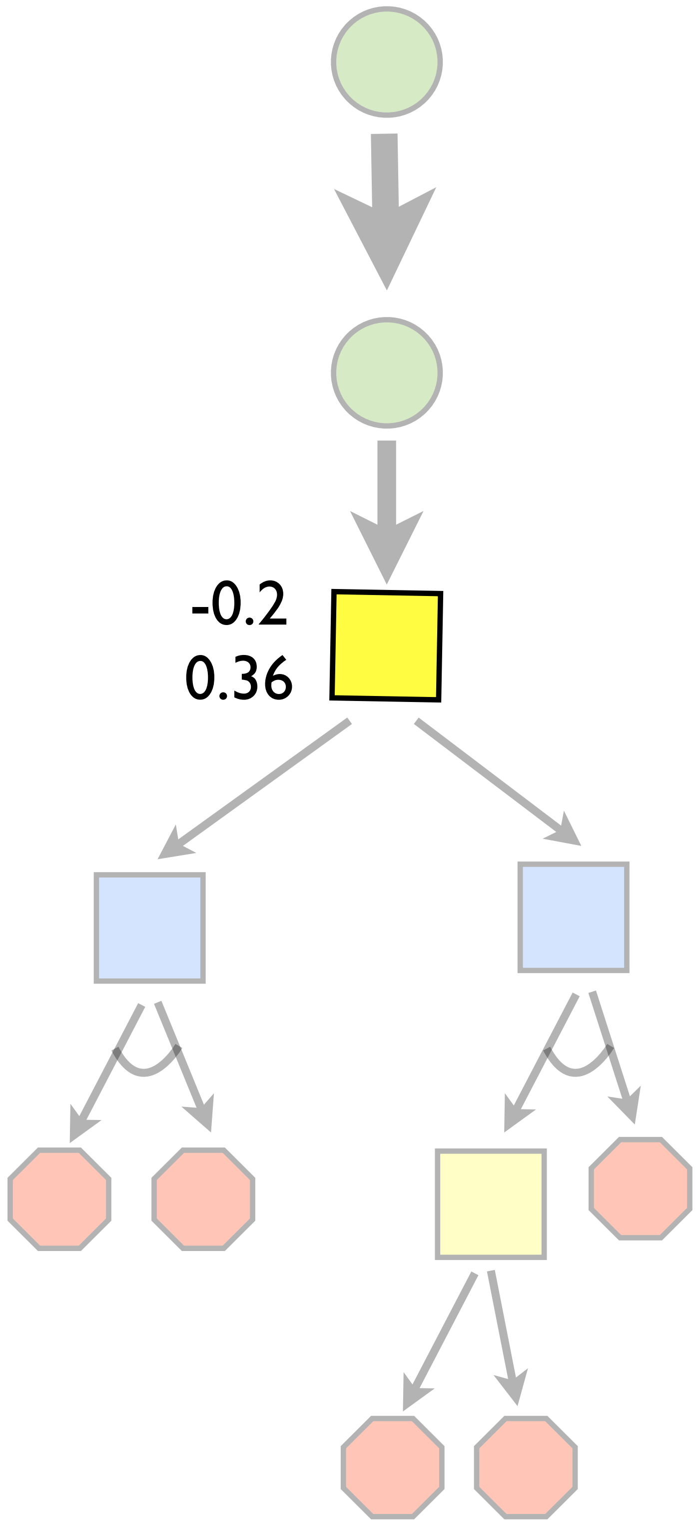
2: 0.09
K: 0.23





Home

1: Walking the Public Tree



Their Reach Prob:

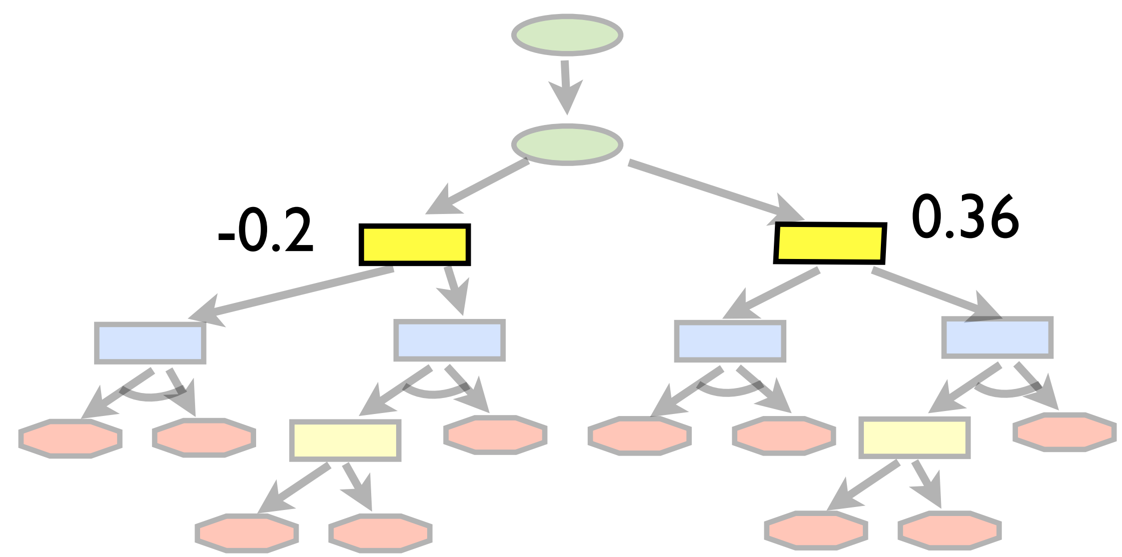
2: 0.5*0.75

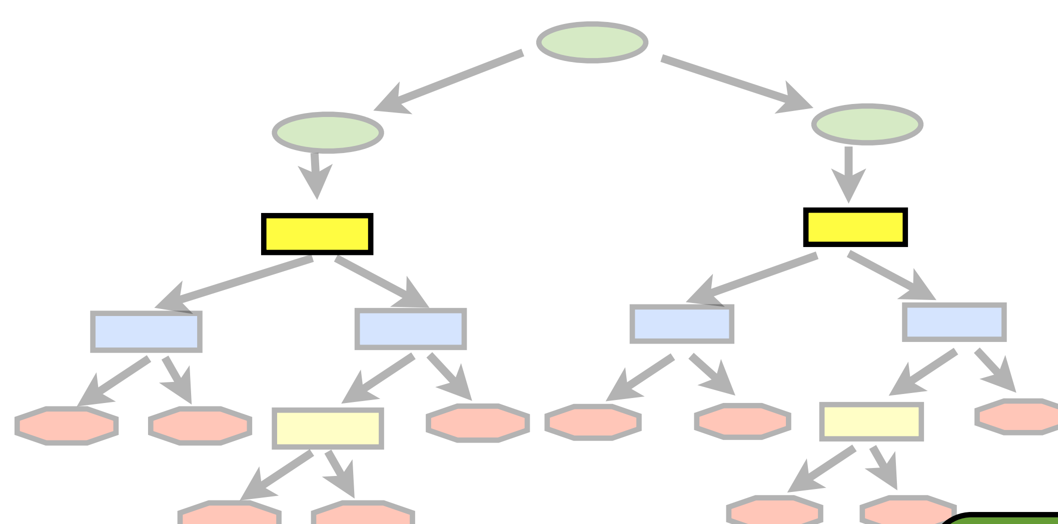
K: 0.5*0.1

My Value:

2: -0.2

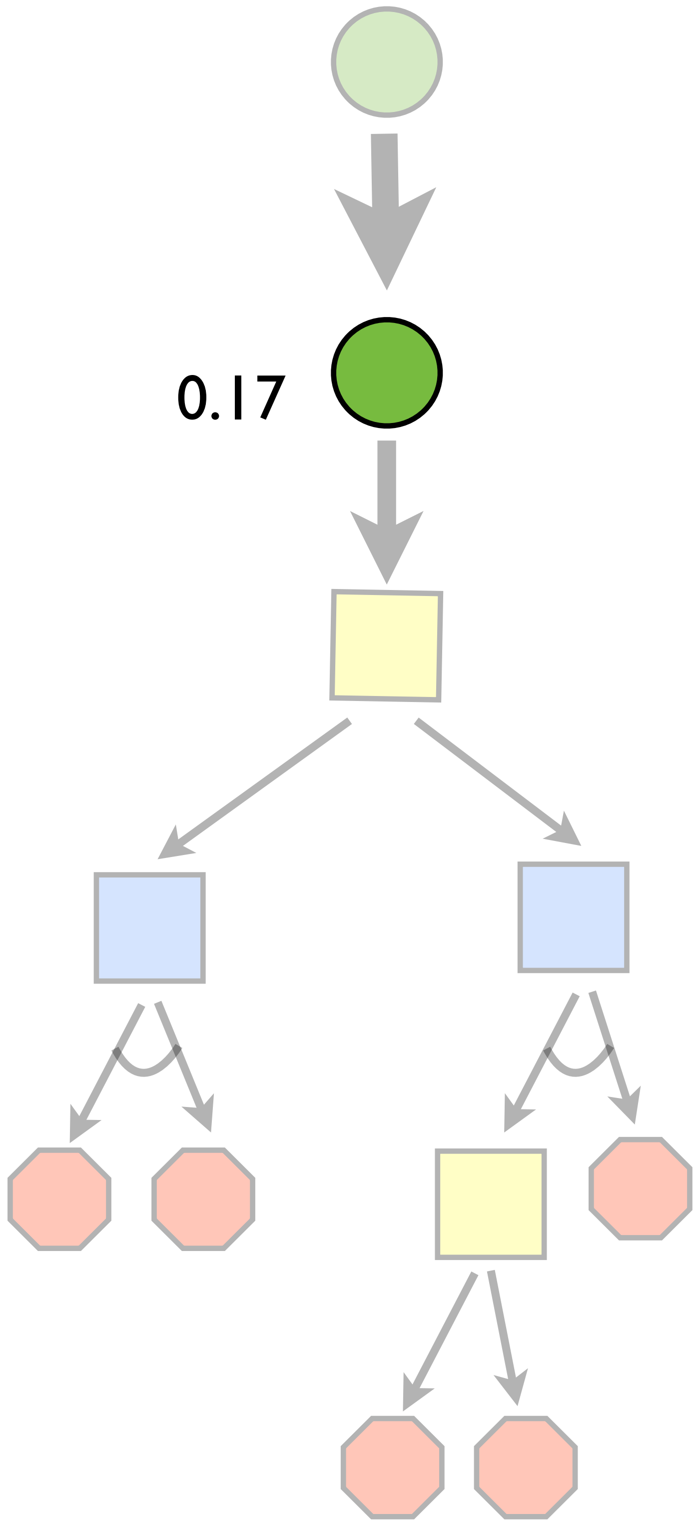
K: 0.36





Home

1: Walking the Public Tree



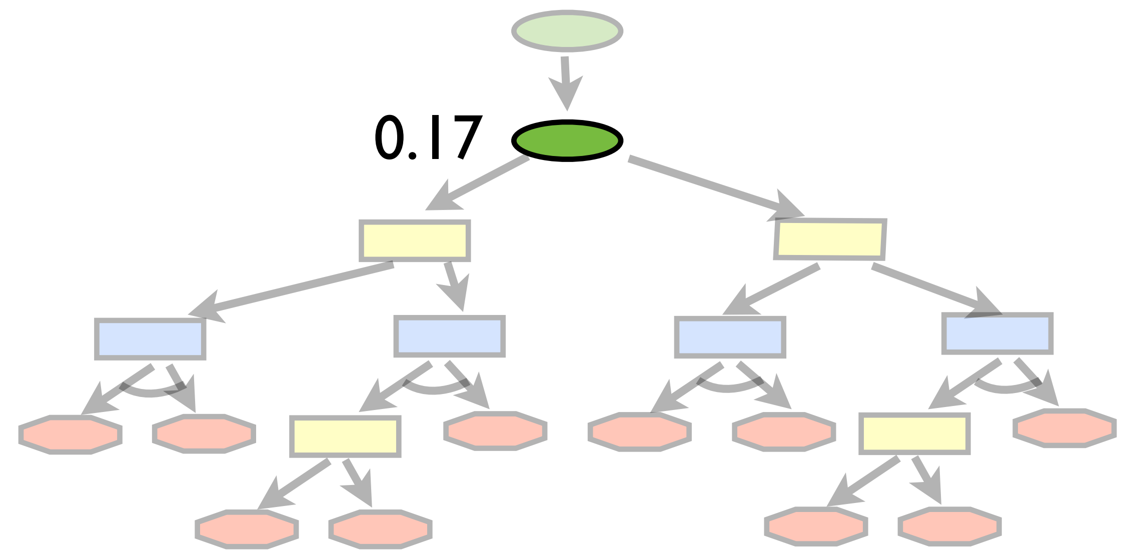
Their Reach Prob:

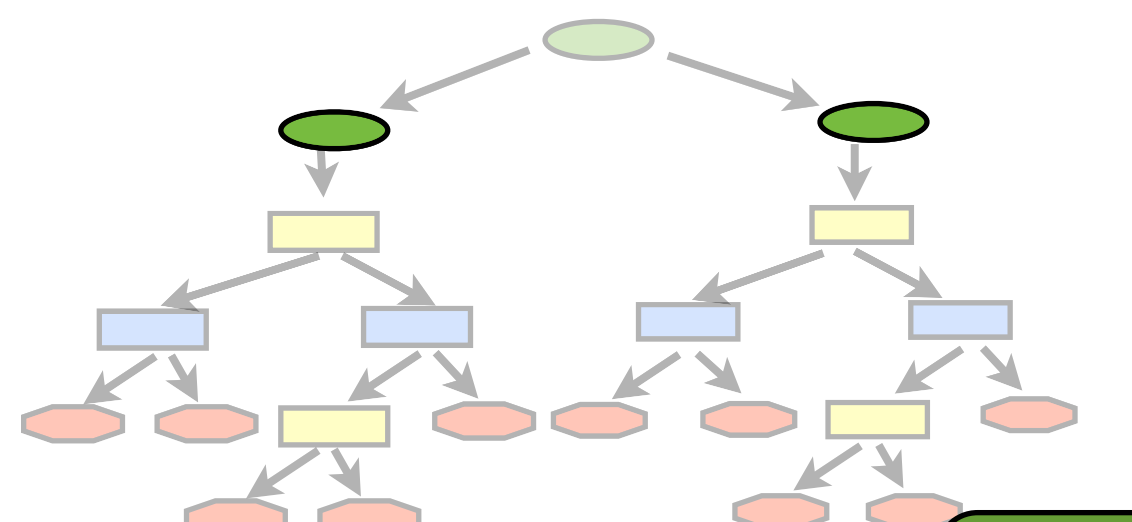
2: 0.5*0.75

K: 0.5*0.1

My Value:

0.18





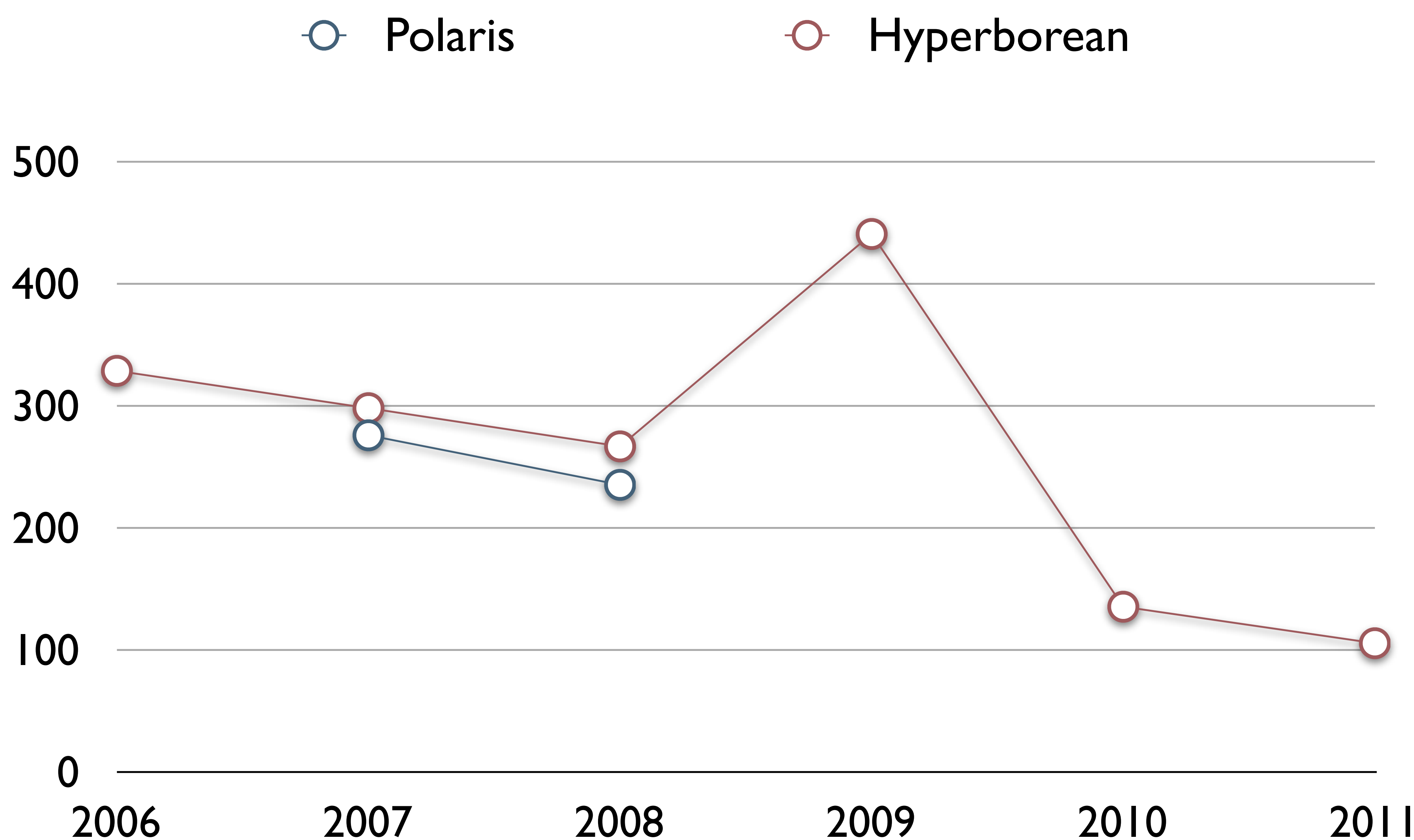
Home

Polaris 2008

Home

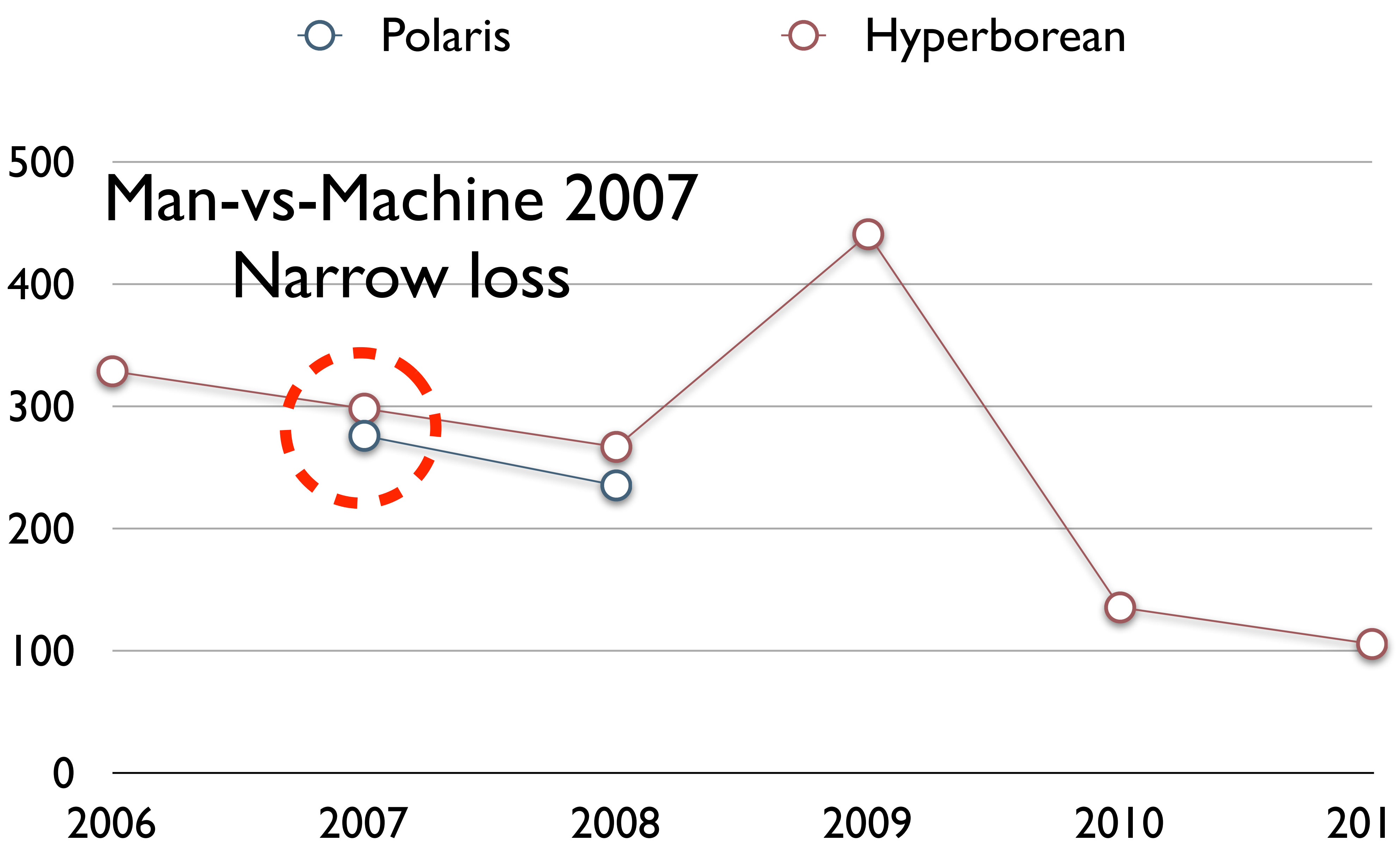
Agent	Size	Tilt	Best Response
Pink	266m	0, 0, 0, 0	235.294
Orange	266m	7, 0, 0, 7	227.457
Peach	266m	0, 0, 0, 7	228.325
Red	115m	0, -7, 0, 0	257.231
Green	115m	0, -7, 0, -7	263.702
(Reference)	115m	0, 0, 0, 0	266.797

Polaris



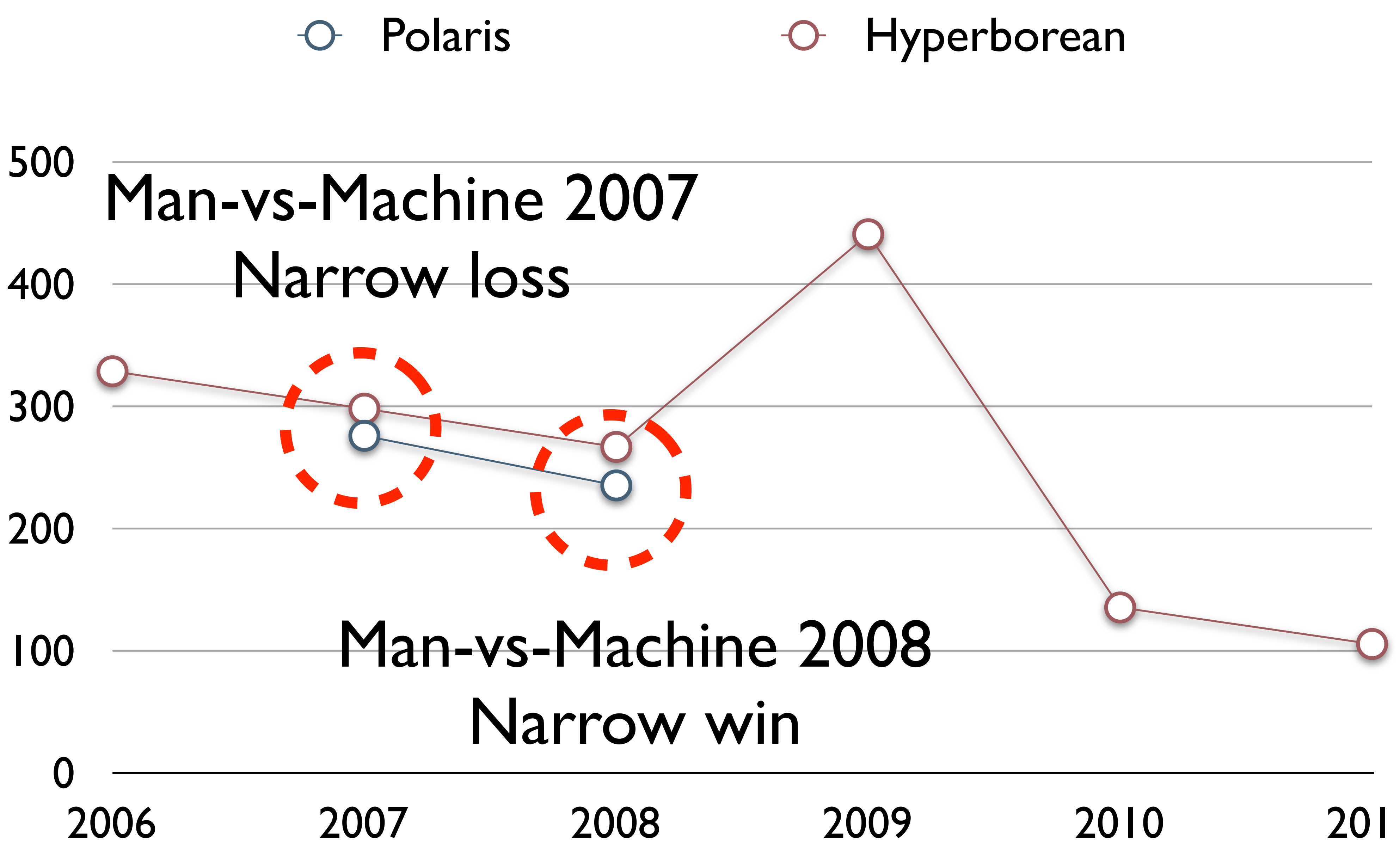
Home

Polaris



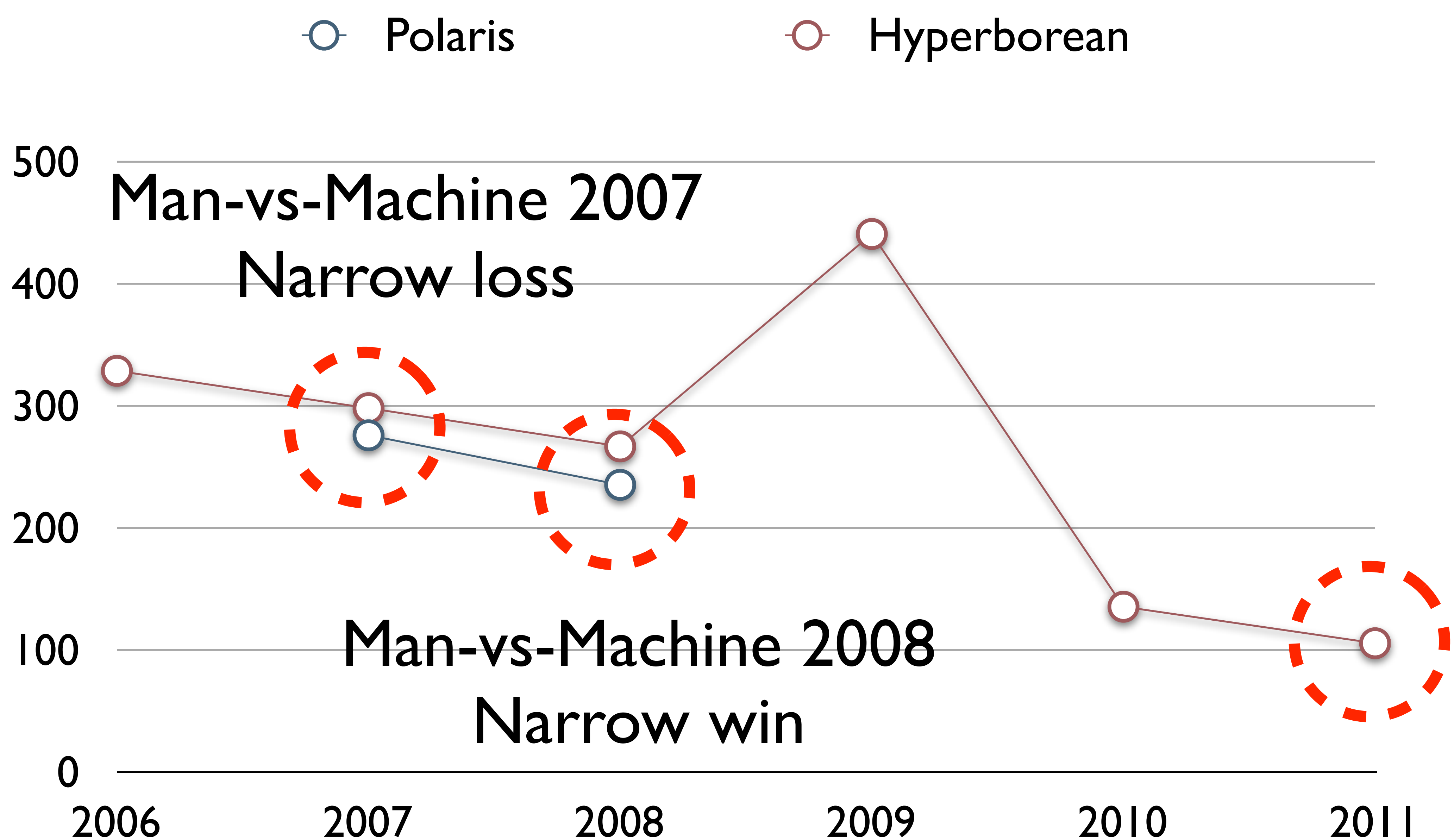
Home

Polaris



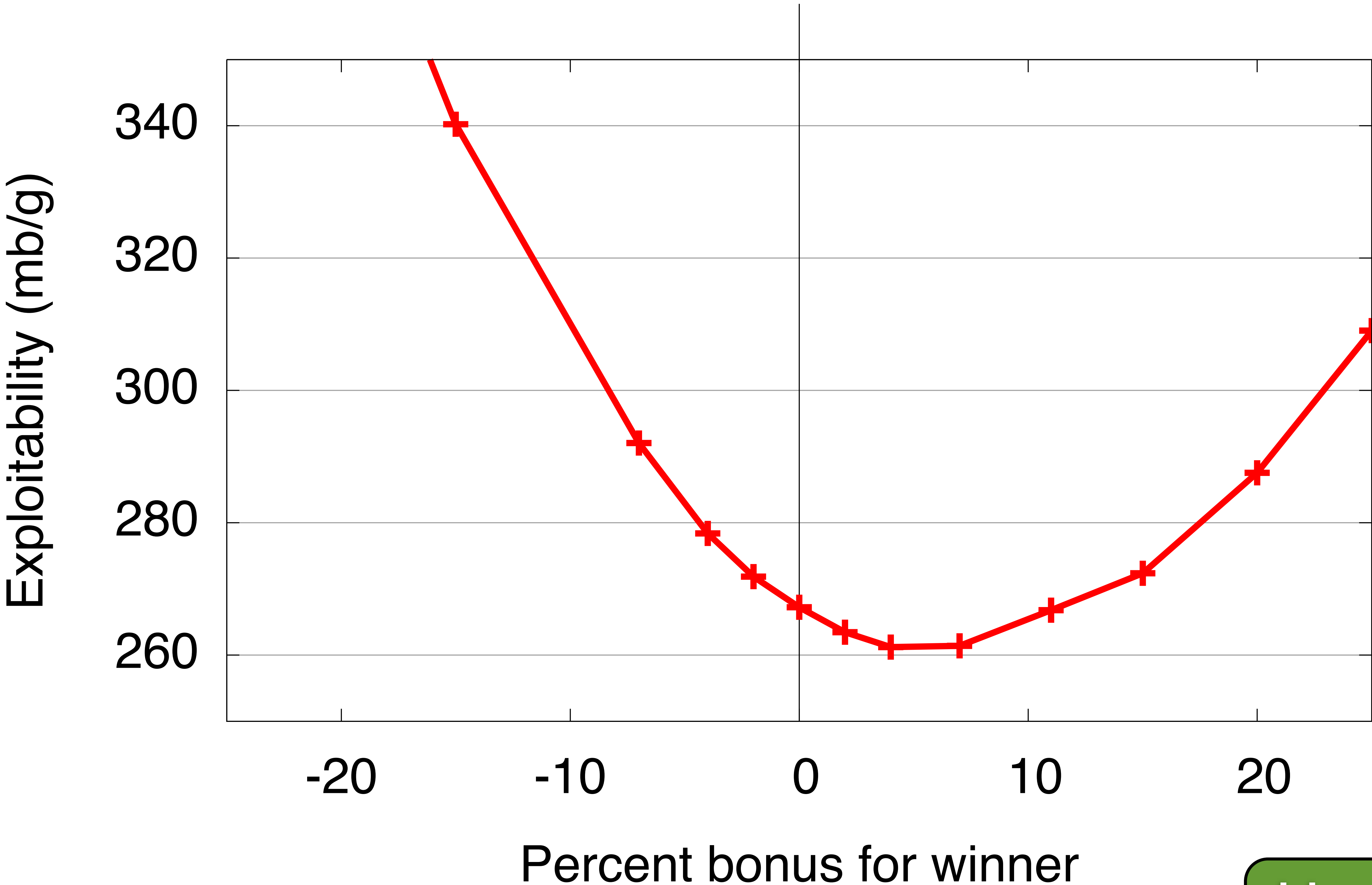
Home

Polaris



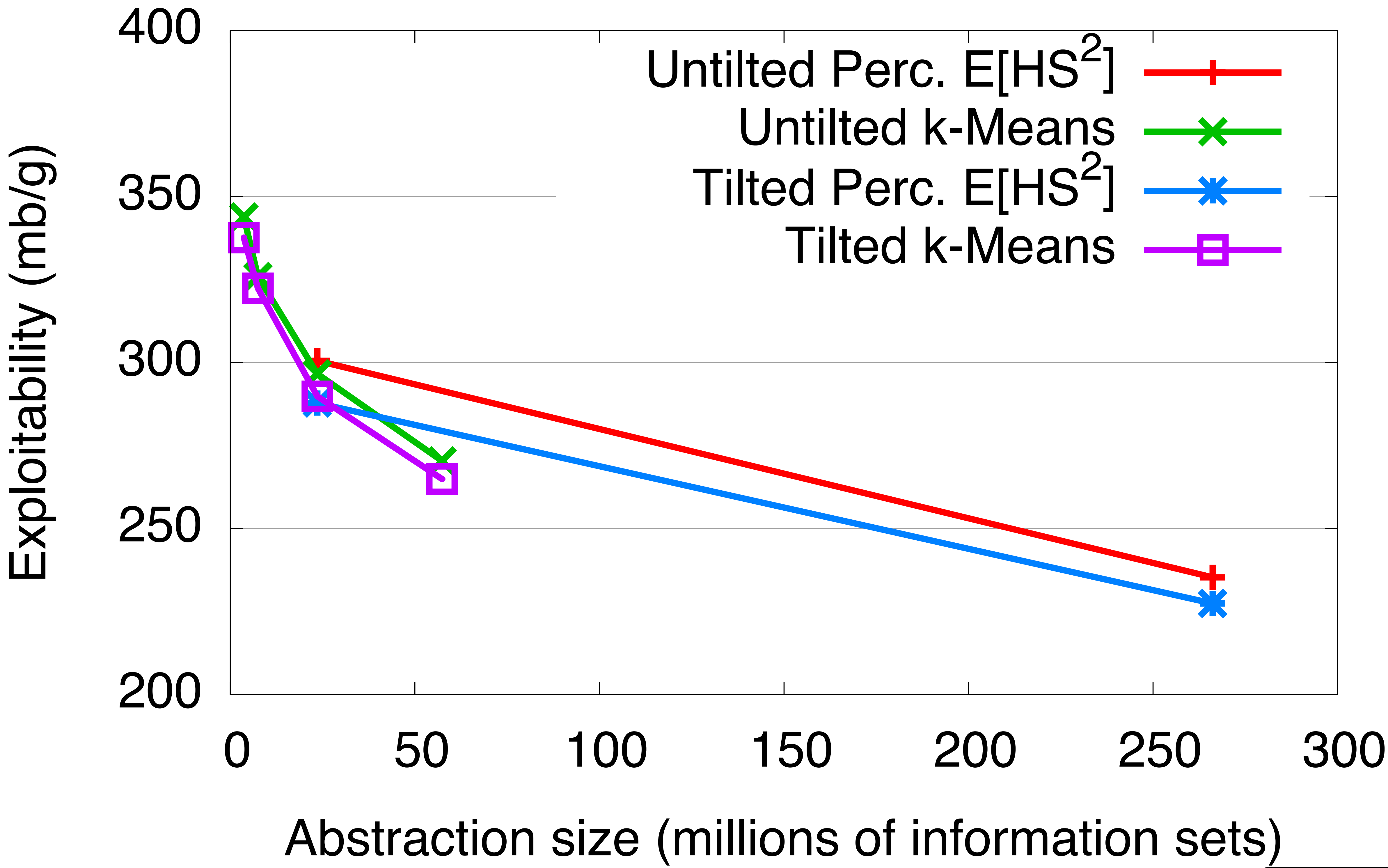
Home

Tilting



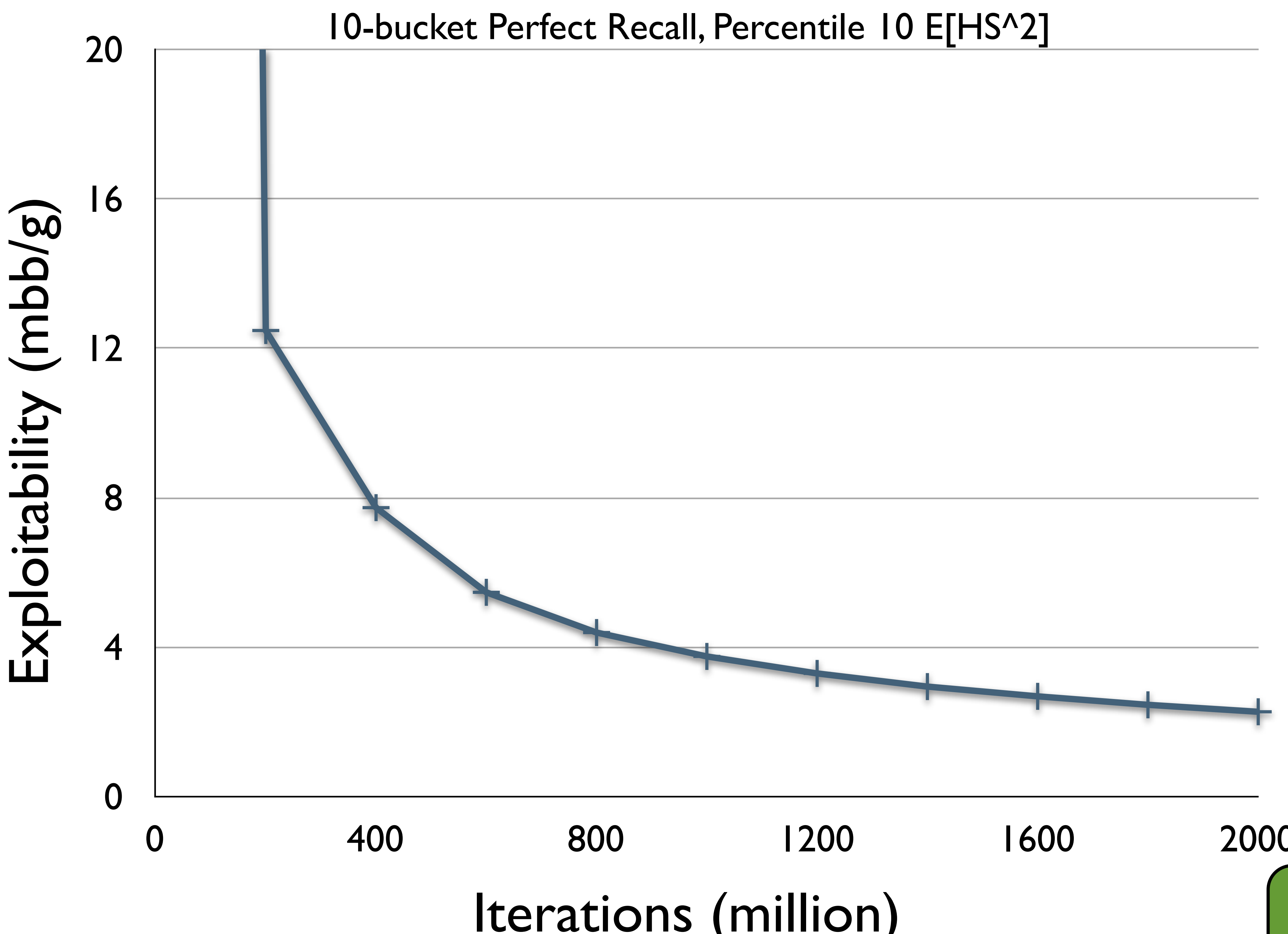
Home

Tilting: 7%



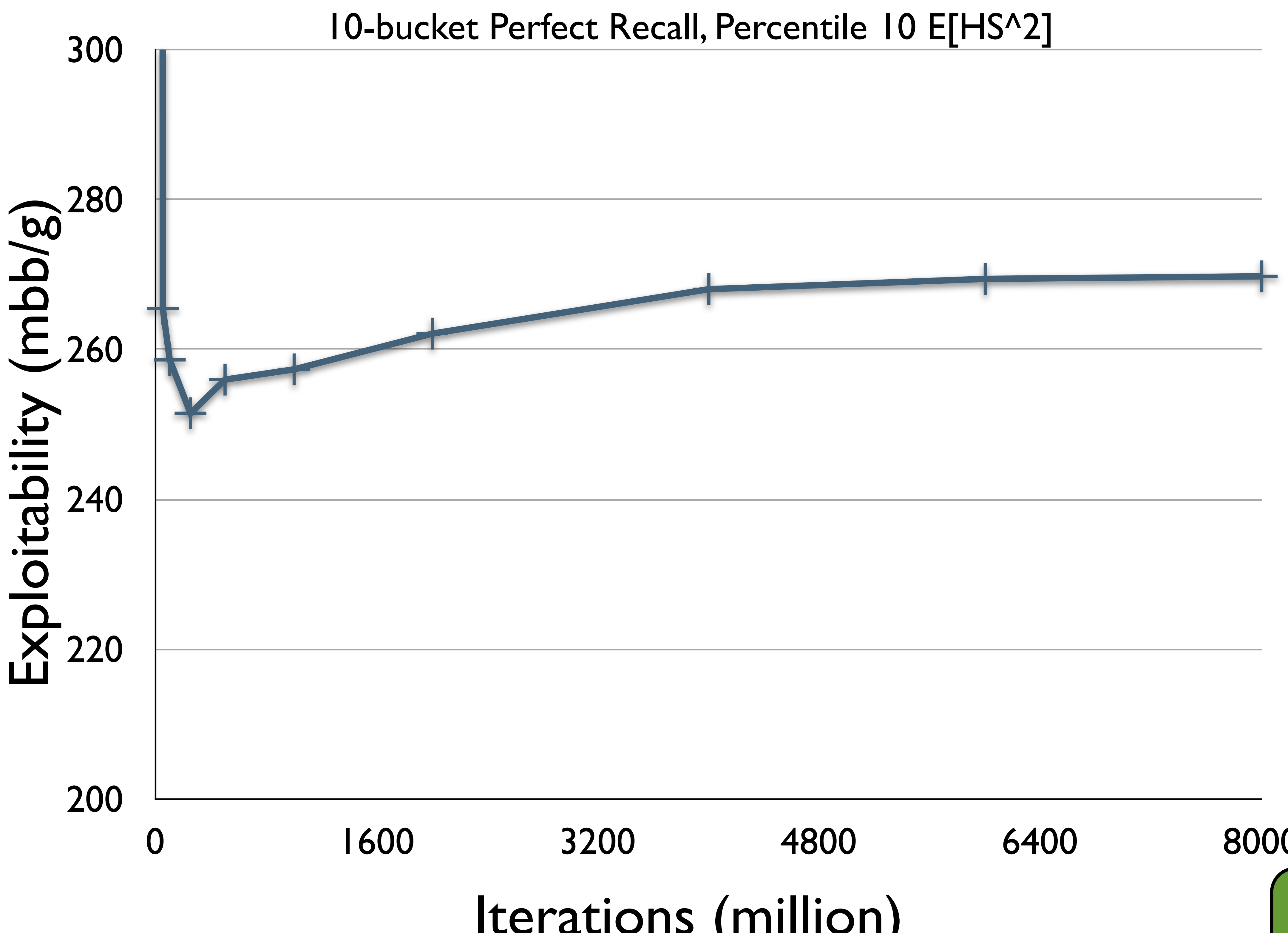
Home

Counterfactual Regret Minimization: Abstract-Game Best Response



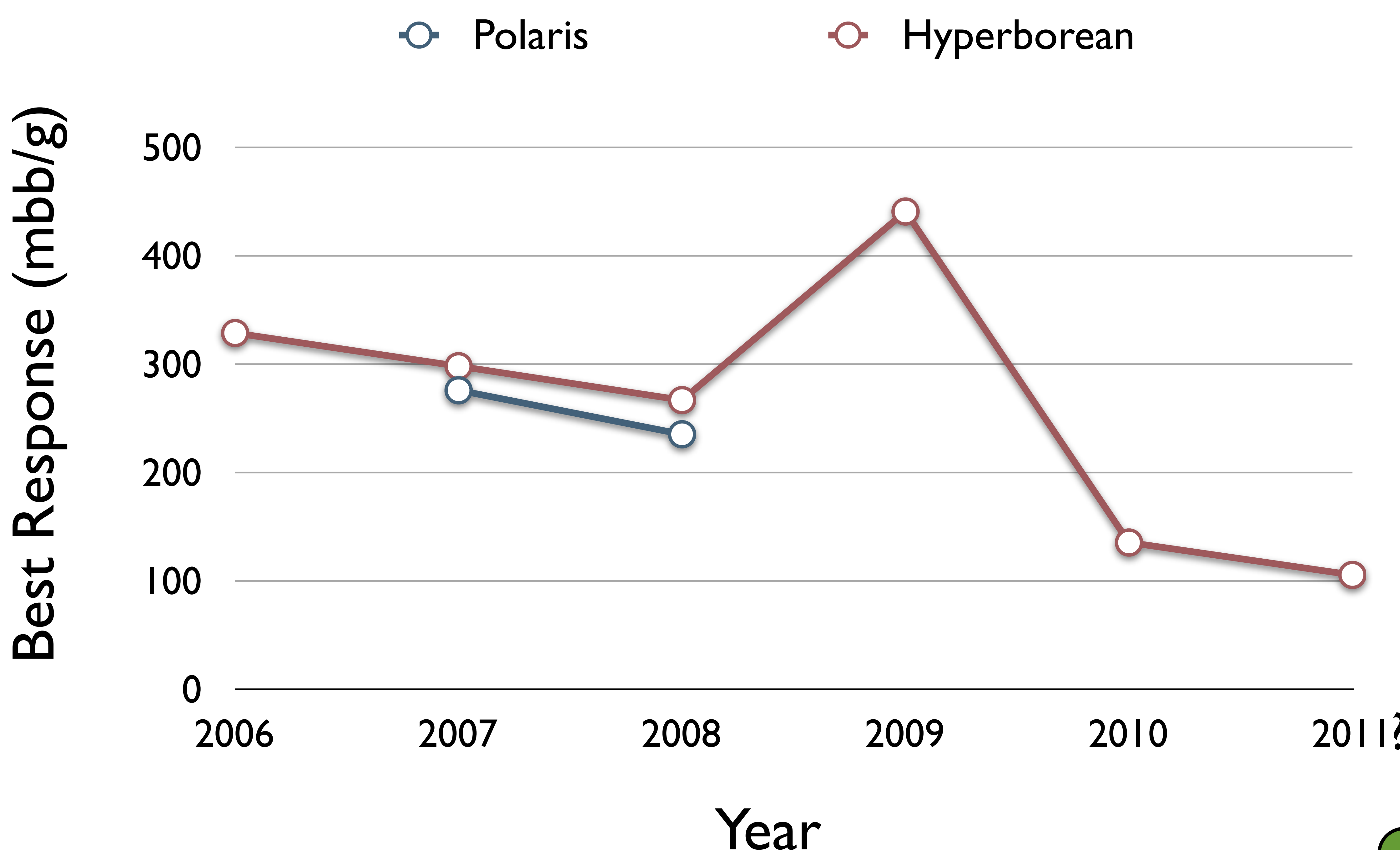
Home

Counterfactual Regret Minimization: Real Game Best Response



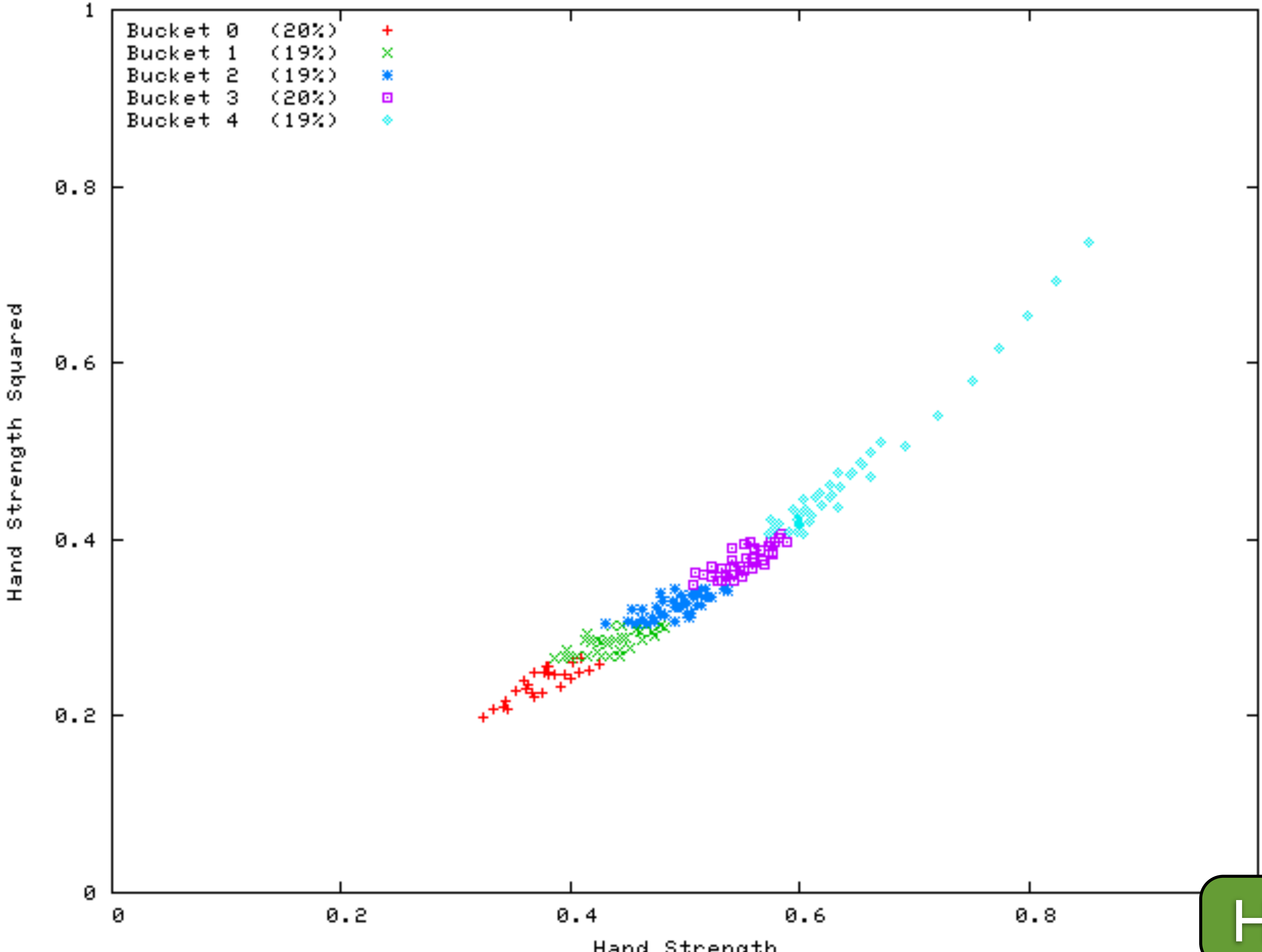
Home

Hyperborean 2009



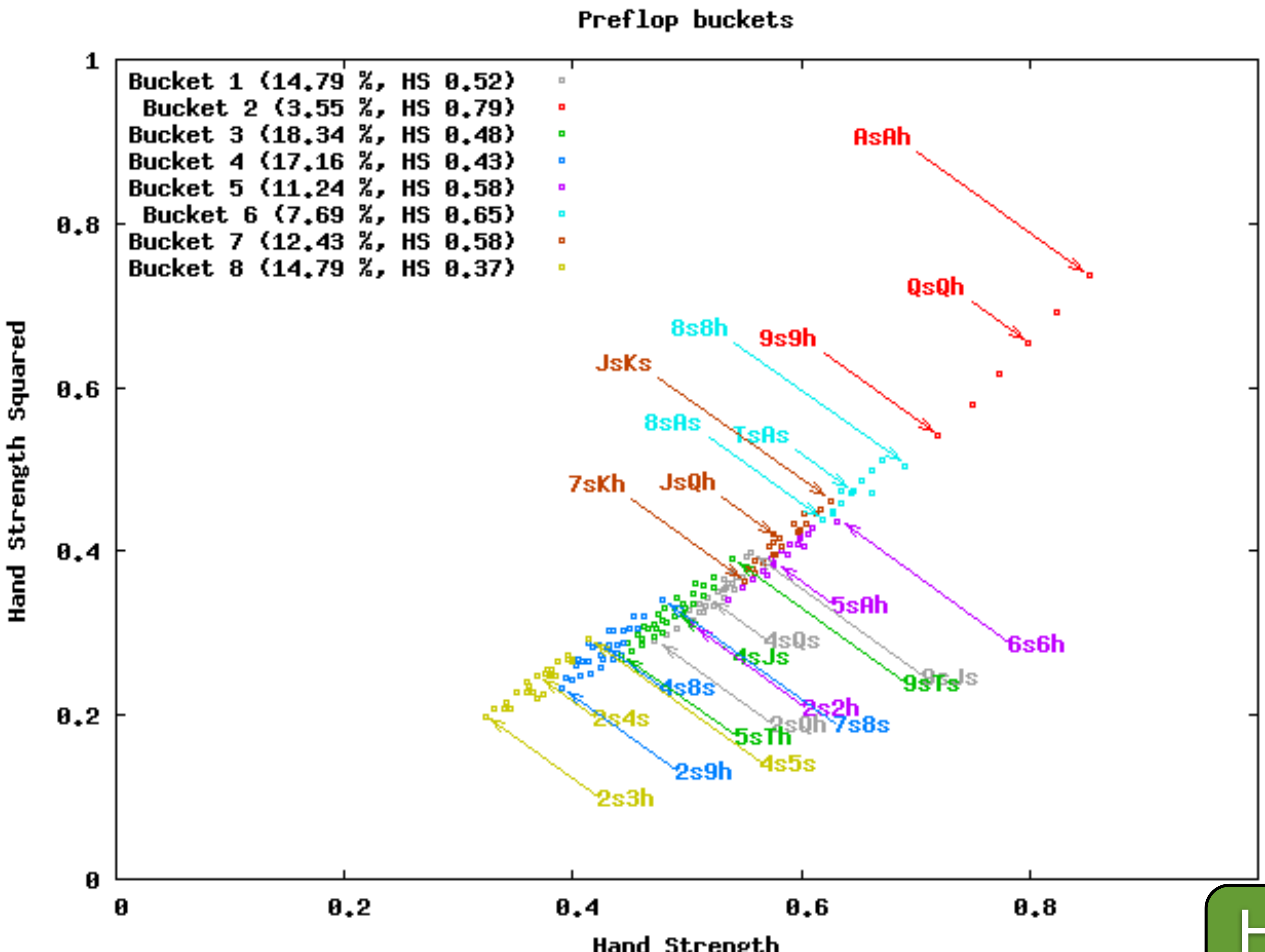
Home

Abstraction: Perc HS^2



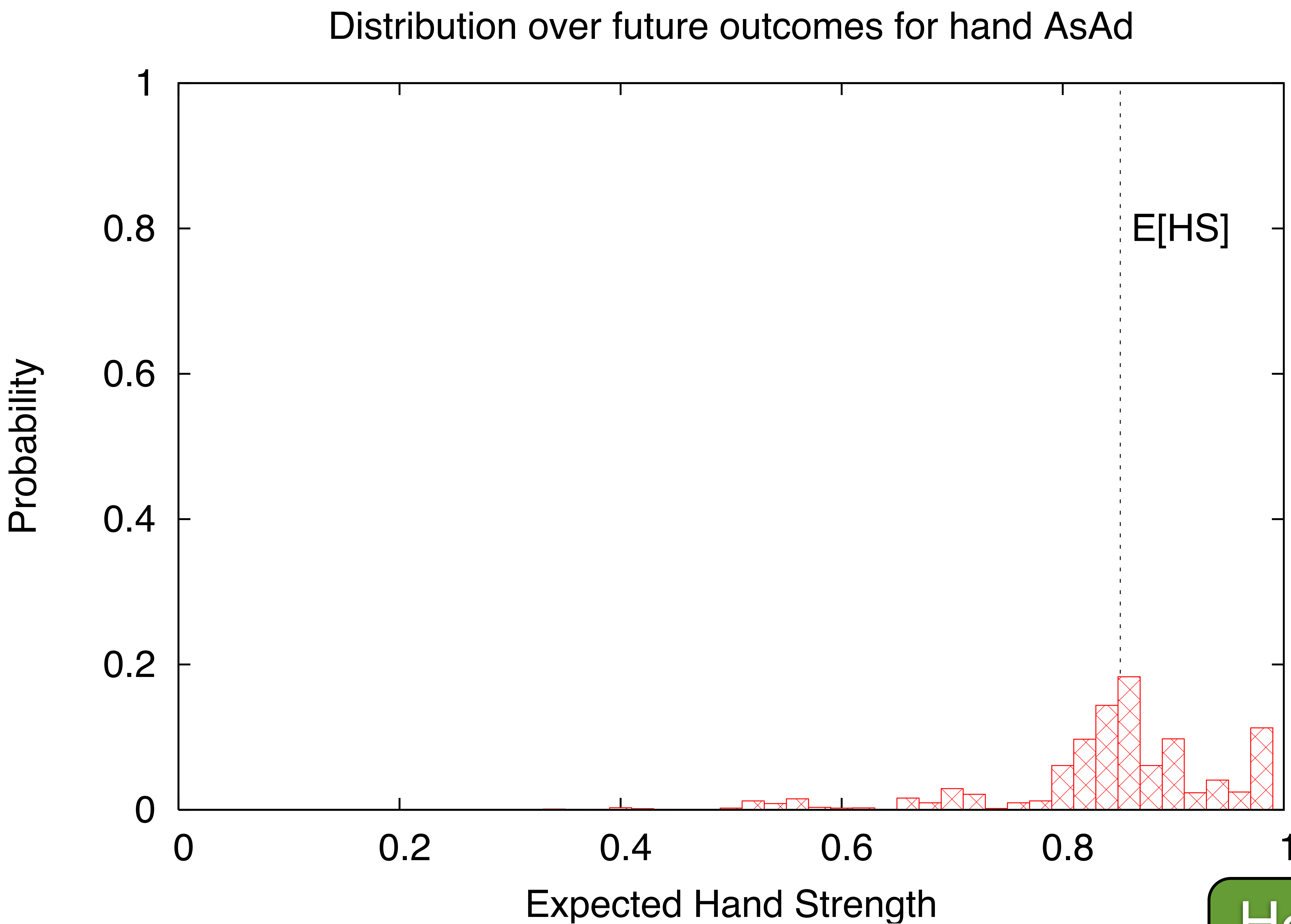
Home

Abstraction: k-Means



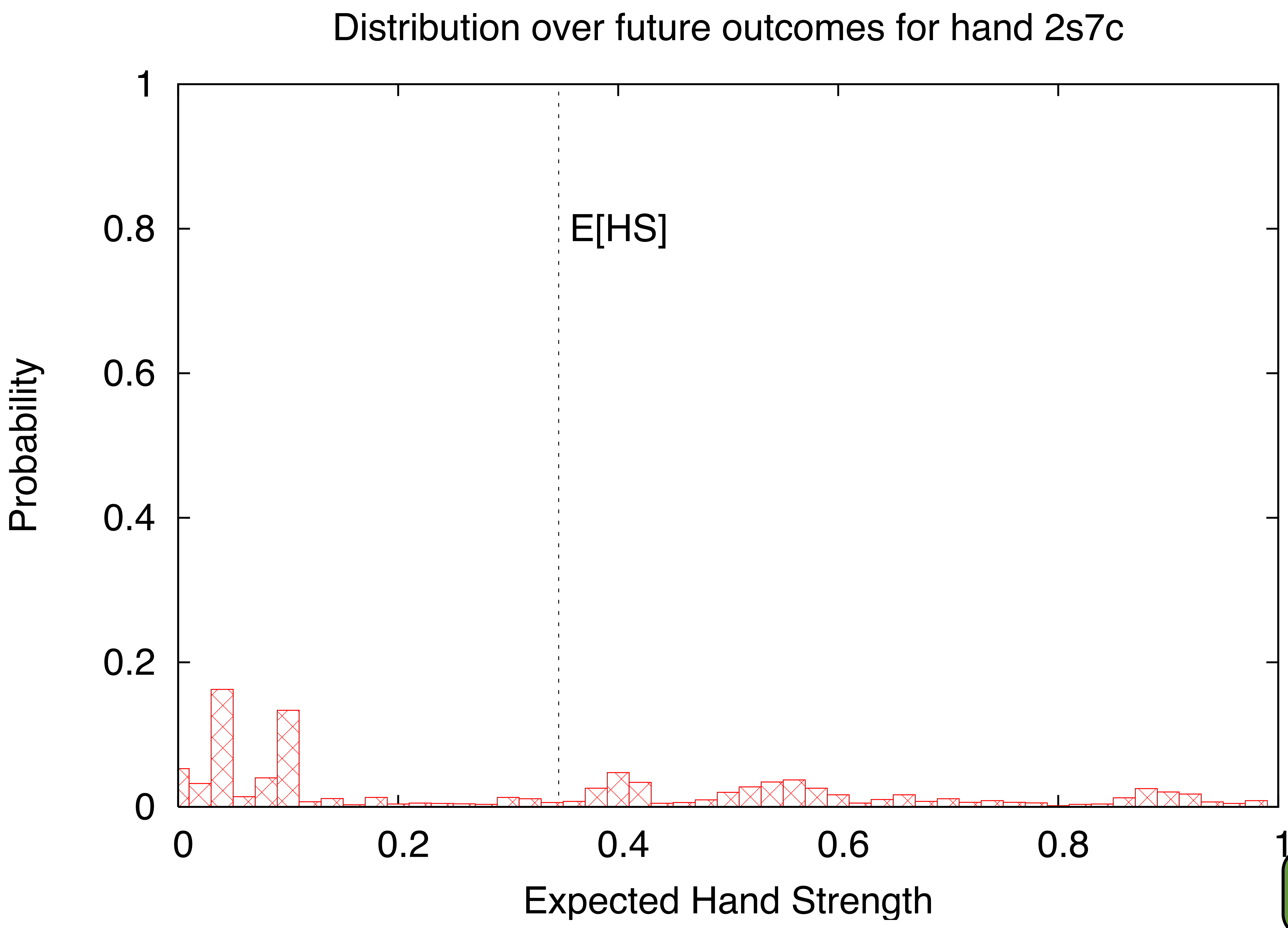
Home

Abstraction: HS Distributions



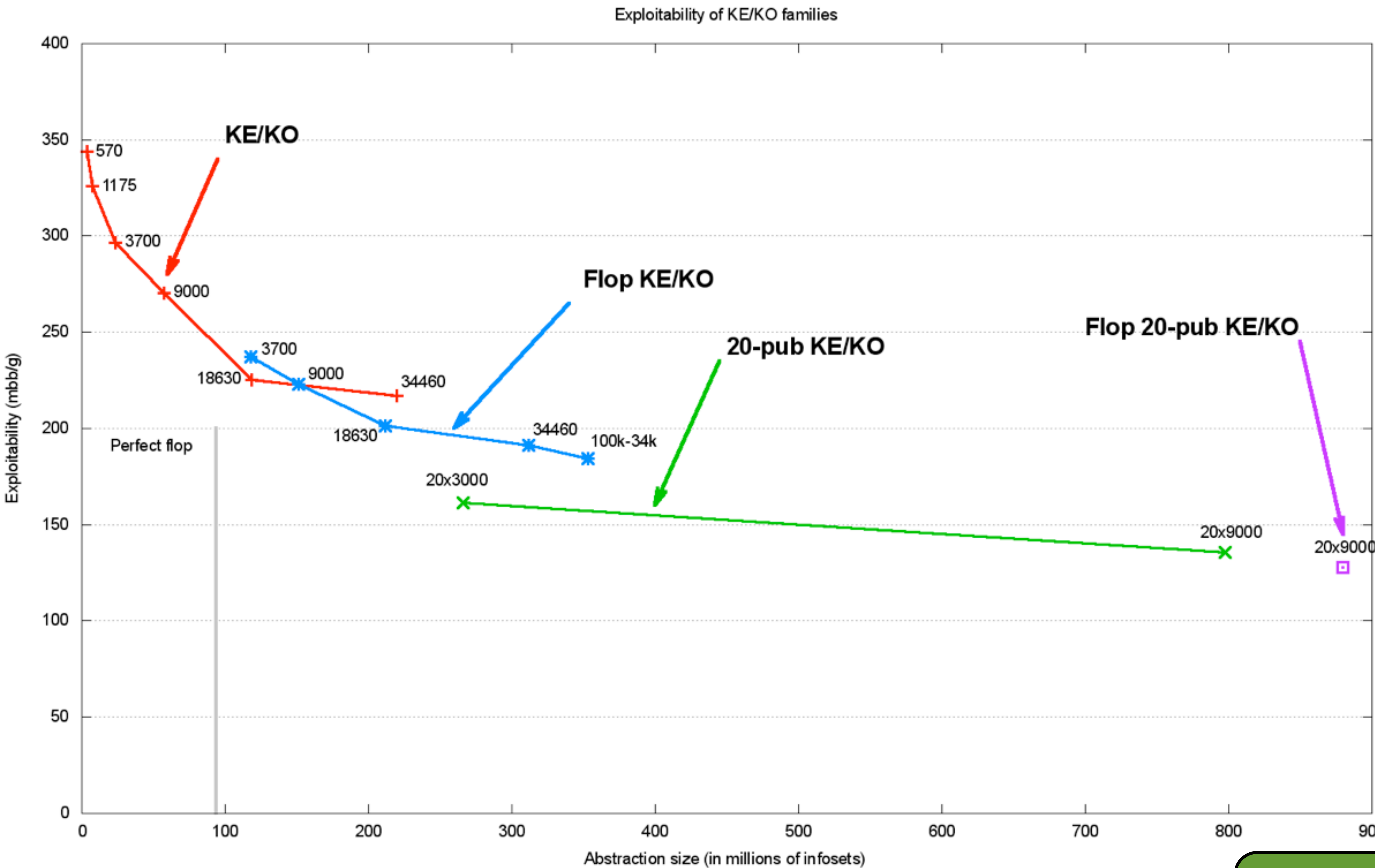
Home

Abstraction: HS Distributions



Home

k-Means Earthmover Abstraction



Home

3: Fast Terminal Node Evaluation

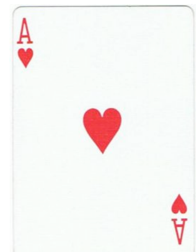
His Reach Probs:


0.1


0.05


0.02

My Values:


= ?


= ?


= ?

3: Fast Terminal Node Evaluation

His Reach Probs:

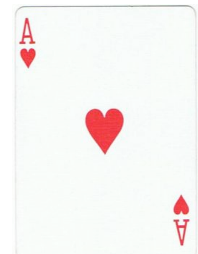
0.1

0.05

0.02

...

My Values: u = utility for winner



$= 0*0.1 + u*0.05 + u*0.02 + \ldots$

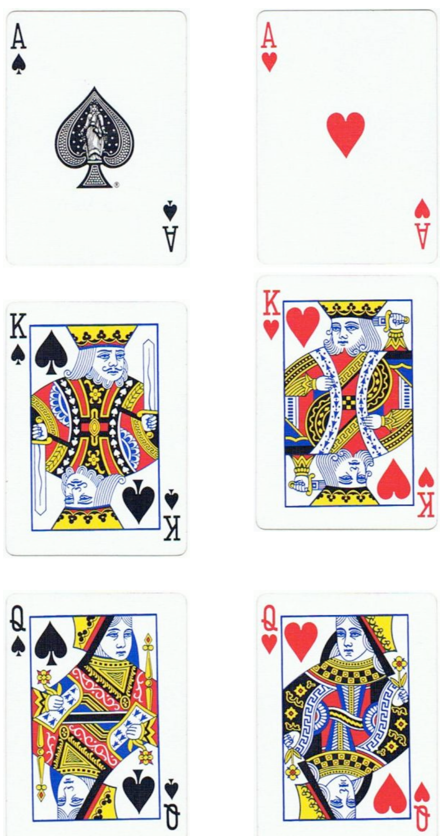
$= -u*0.1 + 0*0.05 + u*0.02 + \ldots$

$= -u*0.1 + -u*0.05 + 0*0.02 + \ldots$

...

3: Fast Terminal Node Evaluation

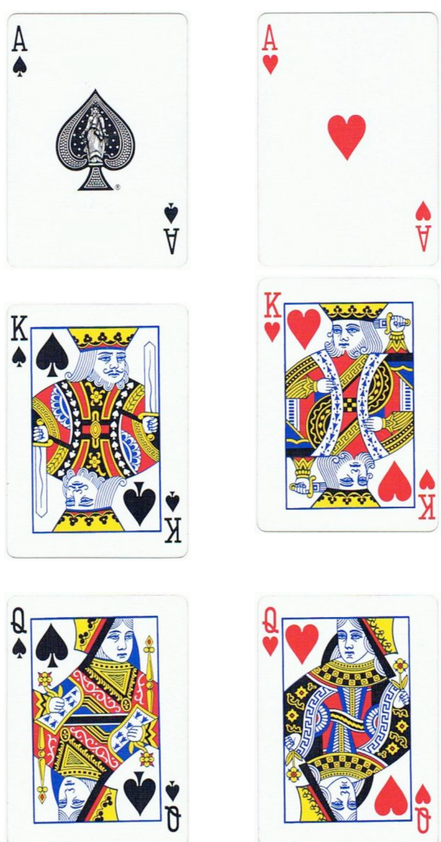
The obvious O(n^2) algorithm:



```
r[i] = his reach probs
v[i] = my values
u = utility for the winner

for( a = each of my hands )
   for( b = each of his hands )
      if( a > b )
         v[a] += u*r[b]
      else if( a < b )
         v[a] -= u*r[b]
```

3: Fast Terminal Node Evaluation



But games are fun because they have structure in determining the payoffs, and we can take advantage of that.

This Vector-vs-Vector evaluation can often be done in O(n) time, and not just in poker.

Home

3: Fast Terminal Node Evaluation

Reach:	0.05	0.1	0.1	0.05	0.1	0.1
Value:						

```
sum_win_prob = 0;
sum_lose_prob = 0;
for( i = each of his hands )
  sum_lose_prob += r[i]
```

Home

3: Fast Terminal Node Evaluation

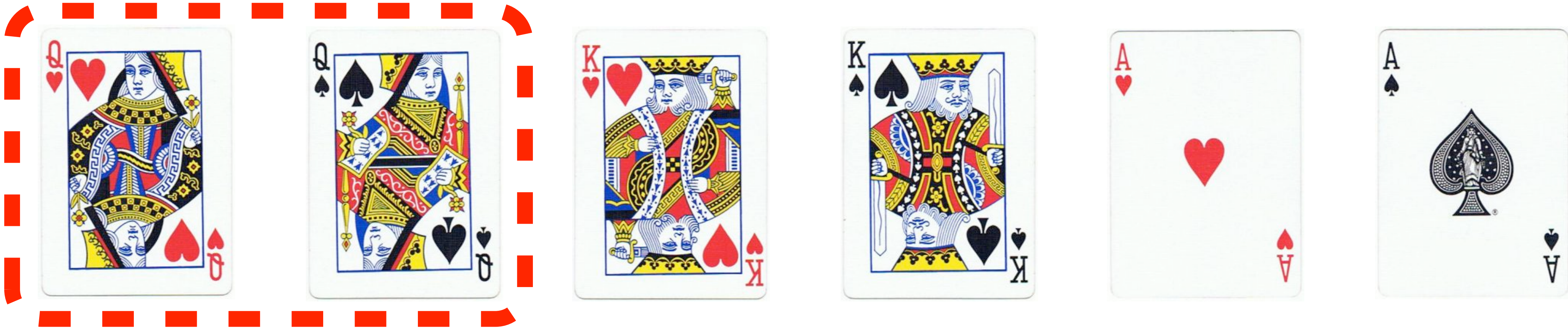
Reach:	0.05	0.1	0.1	0.05	0.1	0.1
Value:						

```
sum_win_prob = 0        sum_lose_prob = 0.5
```

```
for( s = each set of equal-strength hands )
  for( i = each tied hand in s )
    sum_lose_prob -= r[i];
  for( i = each tied hand in s )
    v[i] = -u*sum_lose_prob + u*sum_win_prob
  for( i = each tied hand in s )
     sum_win_prob += r[i];
```

Home

3: Fast Terminal Node Evaluation



Reach:	0.05	0.1	0.1	0.05	0.1	0.1
Value:						

```
sum_win_prob = 0        sum_lose_prob = 0.5
```

```
for( s = each set of equal-strength hands )
  for( i = each tied hand in s )
    sum_lose_prob -= r[i];
  for( i = each tied hand in s )
    v[i] = -u*sum_lose_prob + u*sum_win_prob
  for( i = each tied hand in s )
    sum_win_prob += r[i];
```

Home

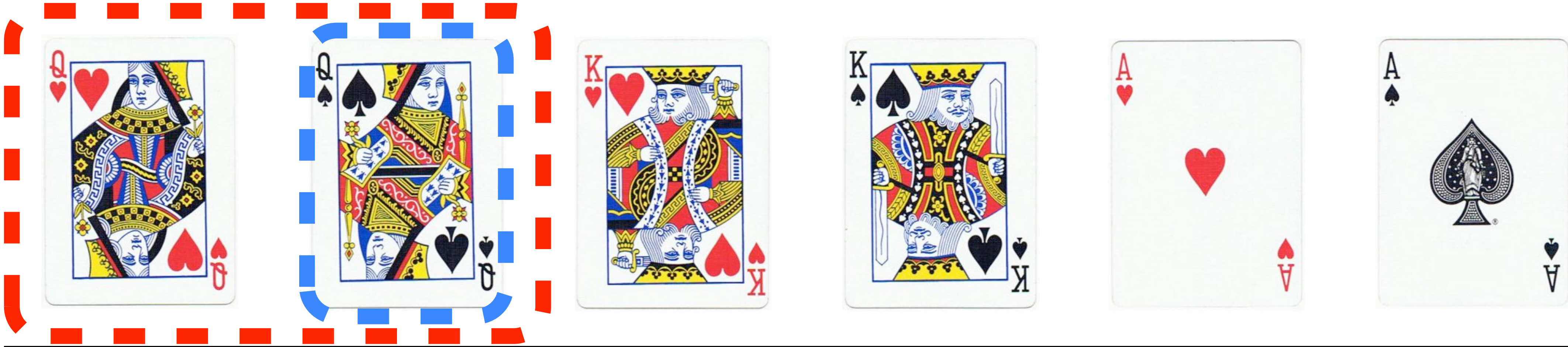
3: Fast Terminal Node Evaluation

Reach:	0.05	0.1	0.1	0.05	0.1	0.1
Value:						

```
sum_win_prob = 0        sum_lose_prob = 0.45

for( s = each set of equal-strength hands )
  for( i = each tied hand in s )
    sum_lose_prob -= r[i];
  for( i = each tied hand in s )
    v[i] = -u*sum_lose_prob + u*sum_win_prob
  for( i = each tied hand in s )
     sum_win_prob += r[i];
```

3: Fast Terminal Node Evaluation



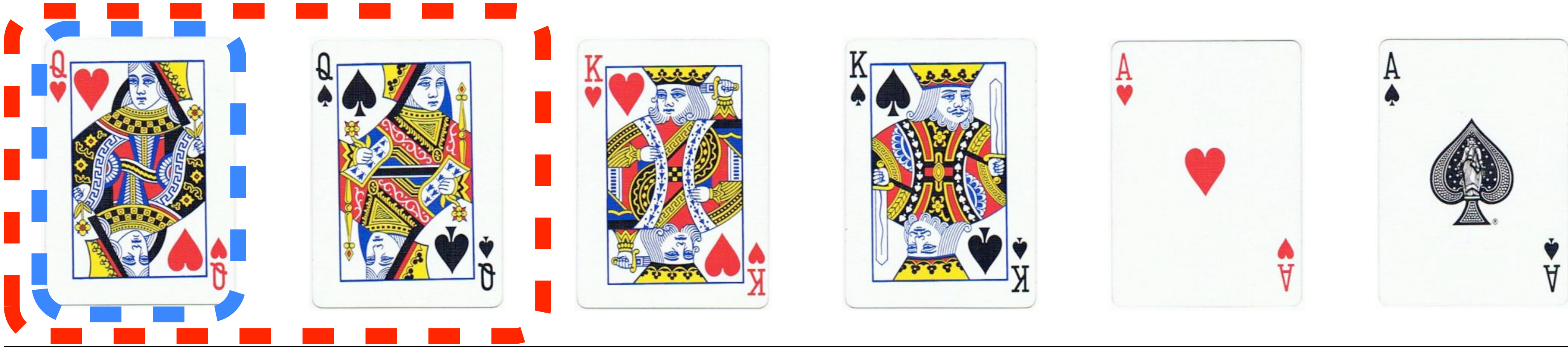
Reach:	0.05	0.1	0.1	0.05	0.1	0.1
Value:						

sum_win_prob = 0   sum_lose_prob = 0.35

```
for( s = each set of equal-strength hands )
  for( i = each tied hand in s )
    sum_lose_prob -= r[i];
  for( i = each tied hand in s )
    v[i] = -u*sum_lose_prob + u*sum_win_prob
  for( i = each tied hand in s )
     sum_win_prob += r[i];
```

Home

3: Fast Terminal Node Evaluation



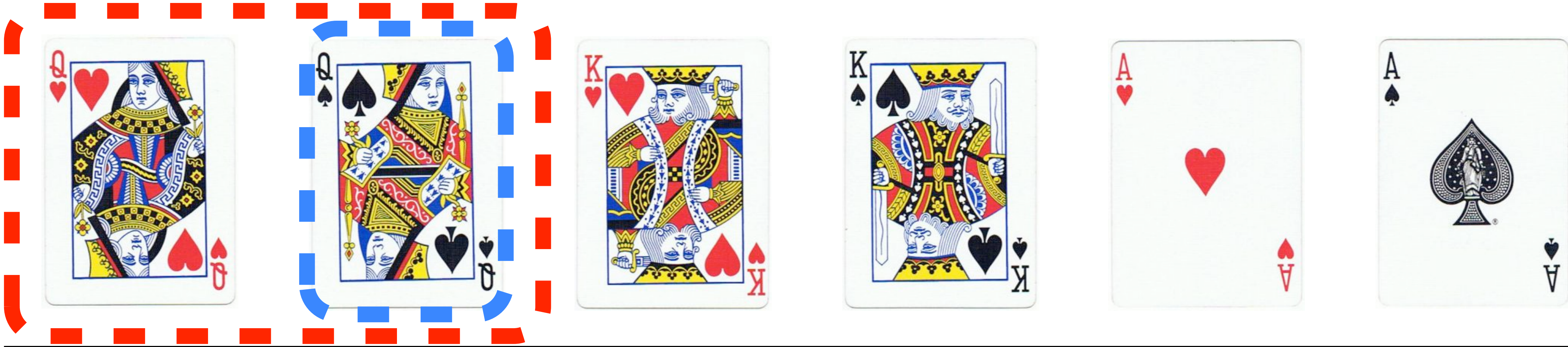
Reach:	0.05	0.1	0.1	0.05	0.1	0.1
Value:	-0.35u					

```
sum_win_prob = 0        sum_lose_prob = 0.35

for( s = each set of equal-strength hands )
  for( i = each tied hand in s )
    sum_lose_prob -= r[i];
  for( i = each tied hand in s )
    v[i] = -u*sum_lose_prob + u*sum_win_prob
  for( i = each tied hand in s )
     sum_win_prob += r[i];
```

Home

3: Fast Terminal Node Evaluation



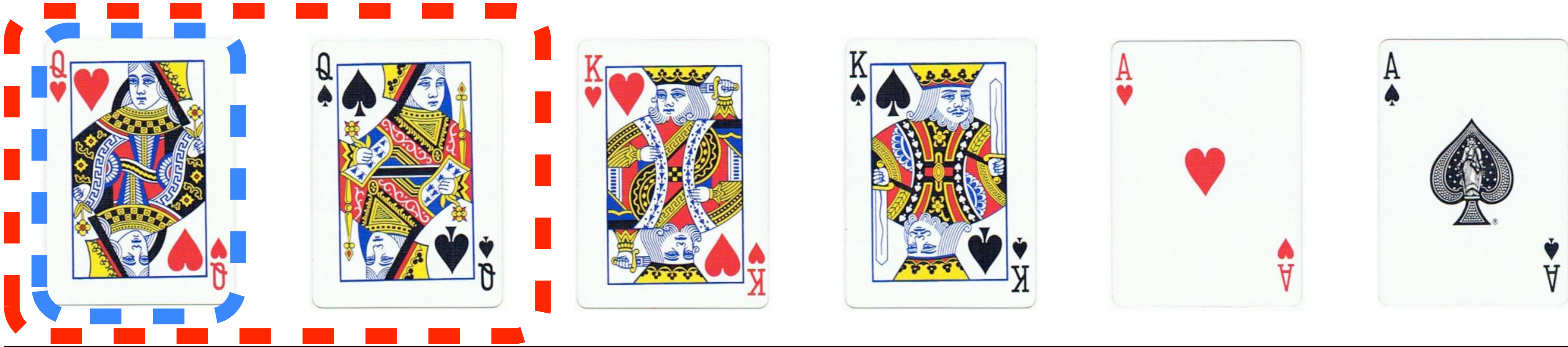
Reach:	0.05	0.1	0.1	0.05	0.1	0.1
Value:	-0.35u	-0.35u				

`sum_win_prob = 0`   `sum_lose_prob = 0.35`

```
for( s = each set of equal-strength hands )
  for( i = each tied hand in s )
    sum_lose_prob -= r[i];
  for( i = each tied hand in s )
    v[i] = -u*sum_lose_prob + u*sum_win_prob
  for( i = each tied hand in s )
    sum_win_prob += r[i];
```

Home

3: Fast Terminal Node Evaluation



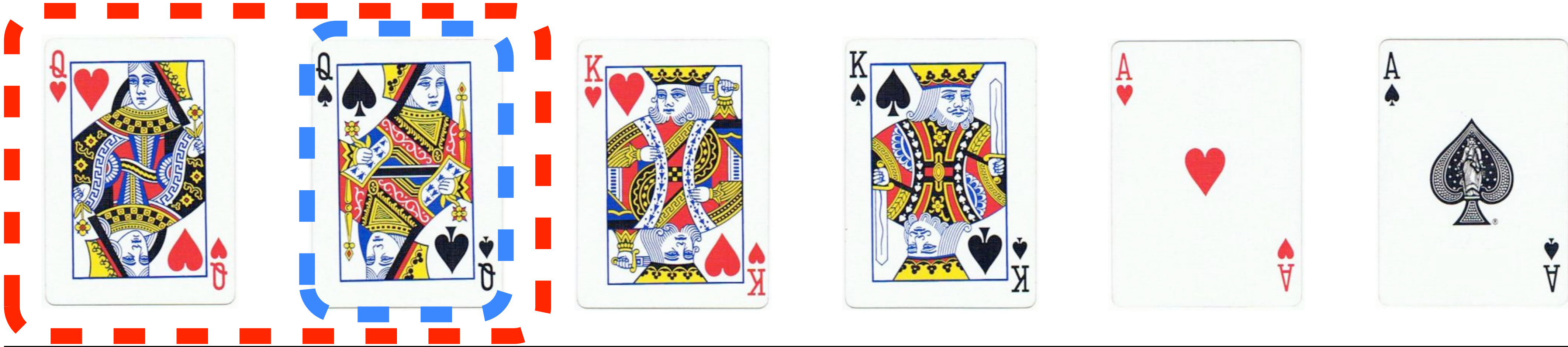
Reach:	0.05	0.1	0.1	0.05	0.1	0.1
Value:	-0.35u	-0.35u				

```
sum_win_prob = 0.05  sum_lose_prob = 0.35

for( s = each set of equal-strength hands )
  for( i = each tied hand in s )
    sum_lose_prob -= r[i];
  for( i = each tied hand in s )
    v[i] = -u*sum_lose_prob + u*sum_win_prob
  for( i = each tied hand in s )
    sum_win_prob += r[i];
```

Home

3: Fast Terminal Node Evaluation



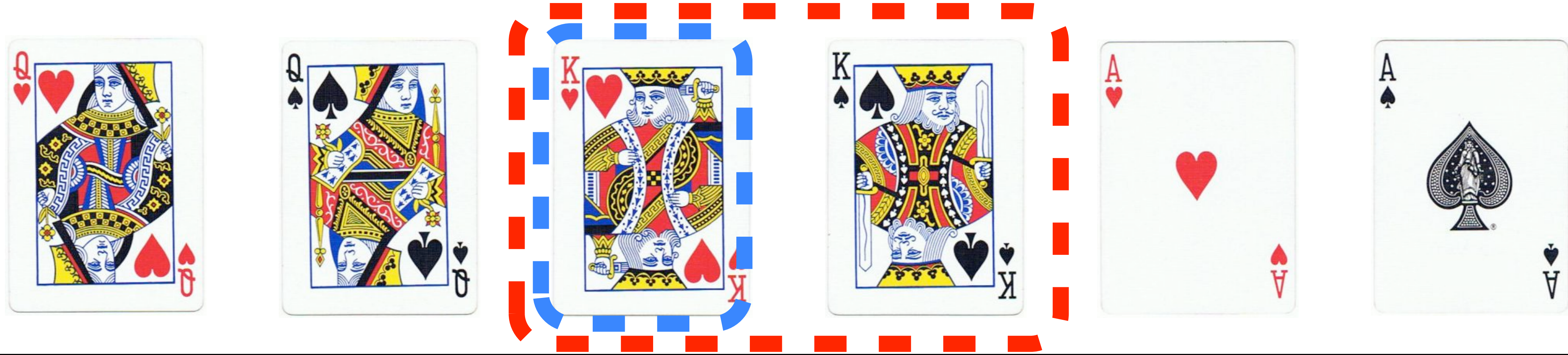
Reach:	0.05	0.1	0.1	0.05	0.1	0.1
Value:	-0.35u	-0.35u				

```
sum_win_prob = 0.15  sum_lose_prob = 0.35

for( s = each set of equal-strength hands )
  for( i = each tied hand in s )
    sum_lose_prob -= r[i];
  for( i = each tied hand in s )
    v[i] = -u*sum_lose_prob + u*sum_win_prob
  for( i = each tied hand in s )
    sum_win_prob += r[i];
```

Home

3: Fast Terminal Node Evaluation



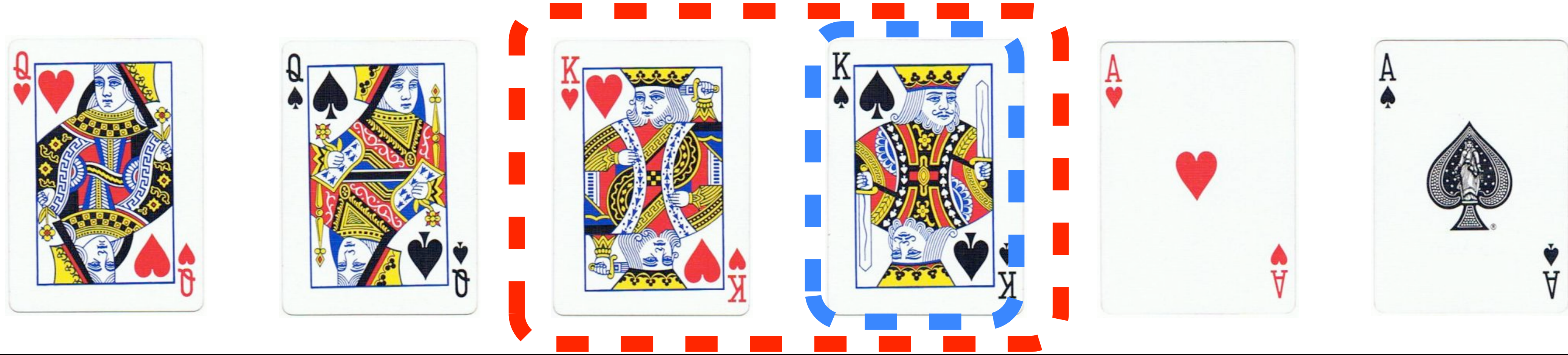
Reach:	0.05	0.1	0.1	0.05	0.1	0.1
Value:	-0.35u	-0.35u				

```
sum_win_prob = 0.15   sum_lose_prob = 0.25

for( s = each set of equal-strength hands )
  for( i = each tied hand in s )
    sum_lose_prob -= r[i];
  for( i = each tied hand in s )
    v[i] = -u*sum_lose_prob + u*sum_win_prob
  for( i = each tied hand in s )
     sum_win_prob += r[i];
```

Home

3: Fast Terminal Node Evaluation



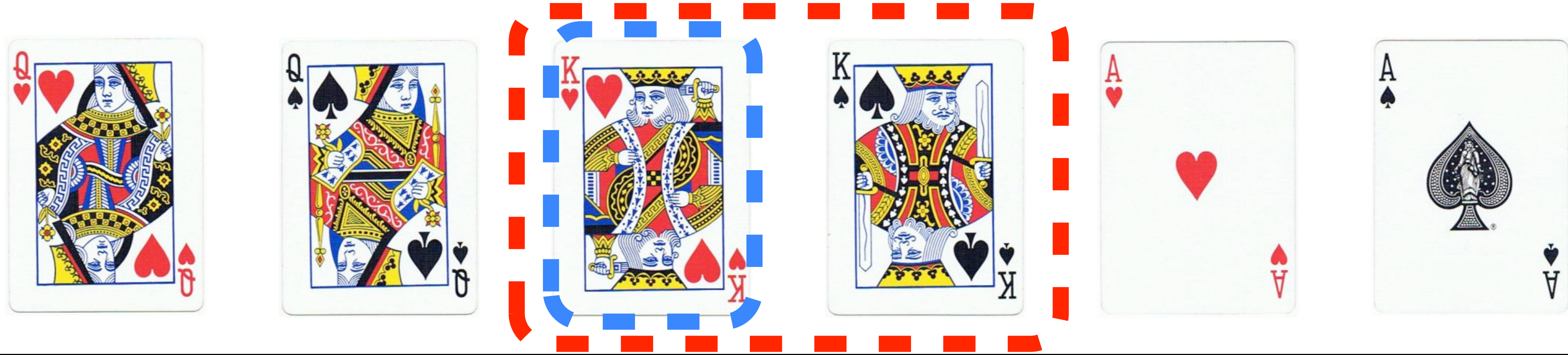
Reach:	0.05	0.1	0.1	0.05	0.1	0.1
Value:	-0.35u	-0.35u				

```
sum_win_prob = 0.15  sum_lose_prob = 0.20
```

```
for( s = each set of equal-strength hands )
  for( i = each tied hand in s )
    sum_lose_prob -= r[i];
  for( i = each tied hand in s )
    v[i] = -u*sum_lose_prob + u*sum_win_prob
  for( i = each tied hand in s )
    sum_win_prob += r[i];
```

Home

3: Fast Terminal Node Evaluation

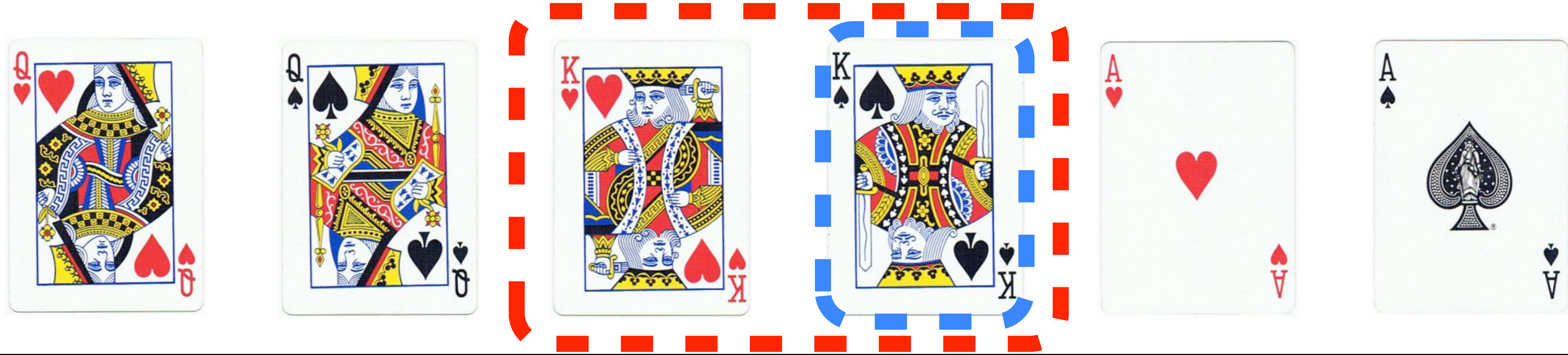


Reach:	0.05	0.1	0.1	0.05	0.1	0.1
Value:	-0.35u	-0.35u	0.05u			

```
sum_win_prob = 0.15   sum_lose_prob = 0.20
```

```
for( s = each set of equal-strength hands )
  for( i = each tied hand in s )
    sum_lose_prob -= r[i];
  for( i = each tied hand in s )
    v[i] = -u*sum_lose_prob + u*sum_win_prob
  for( i = each tied hand in s )
     sum_win_prob += r[i];
```

3: Fast Terminal Node Evaluation



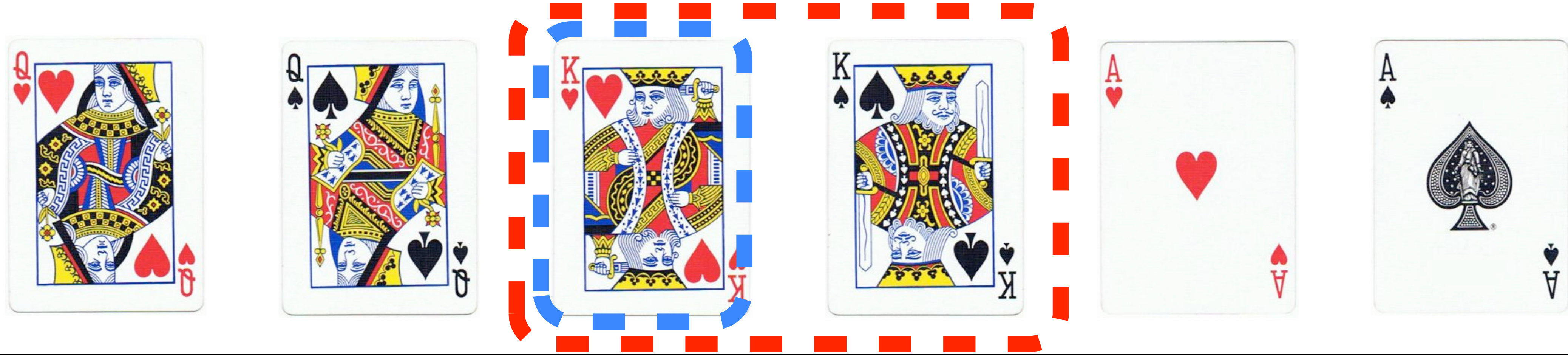
Reach:	0.05	0.1	0.1	0.05	0.1	0.1
Value:	-0.35u	-0.35u	0.05u	0.05u		

```
sum_win_prob = 0.15   sum_lose_prob = 0.20

for( s = each set of equal-strength hands )
  for( i = each tied hand in s )
    sum_lose_prob -= r[i];
  for( i = each tied hand in s )
    v[i] = -u*sum_lose_prob + u*sum_win_prob
  for( i = each tied hand in s )
    sum_win_prob += r[i];
```

Home

3: Fast Terminal Node Evaluation



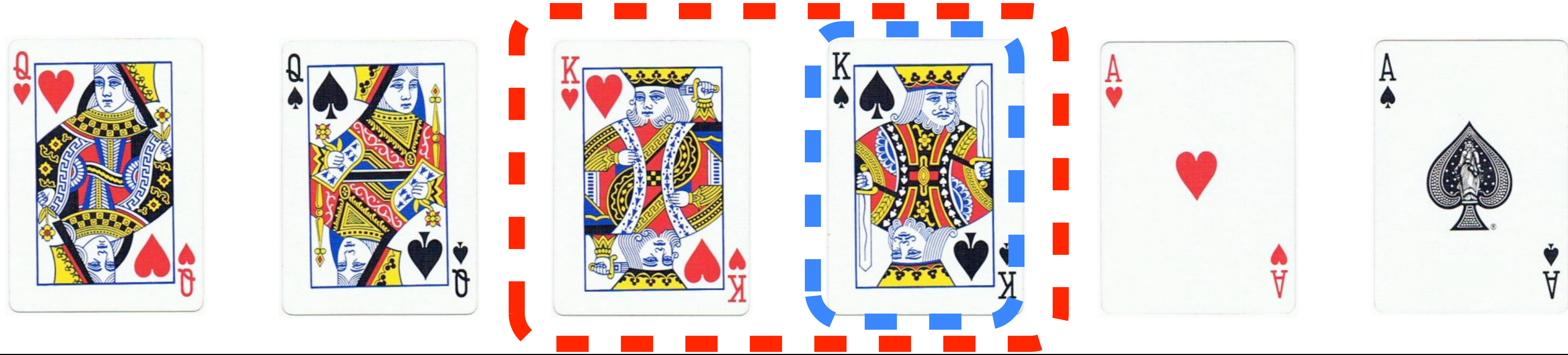
Reach:	0.05	0.1	0.1	0.05	0.1	0.1
Value:	-0.35u	-0.35u	0.05u	0.05u		

```
sum_win_prob = 0.25  sum_lose_prob = 0.20

for( s = each set of equal-strength hands )
  for( i = each tied hand in s )
    sum_lose_prob -= r[i];
  for( i = each tied hand in s )
    v[i] = -u*sum_lose_prob + u*sum_win_prob
  for( i = each tied hand in s )
     sum_win_prob += r[i];
```

Home

3: Fast Terminal Node Evaluation



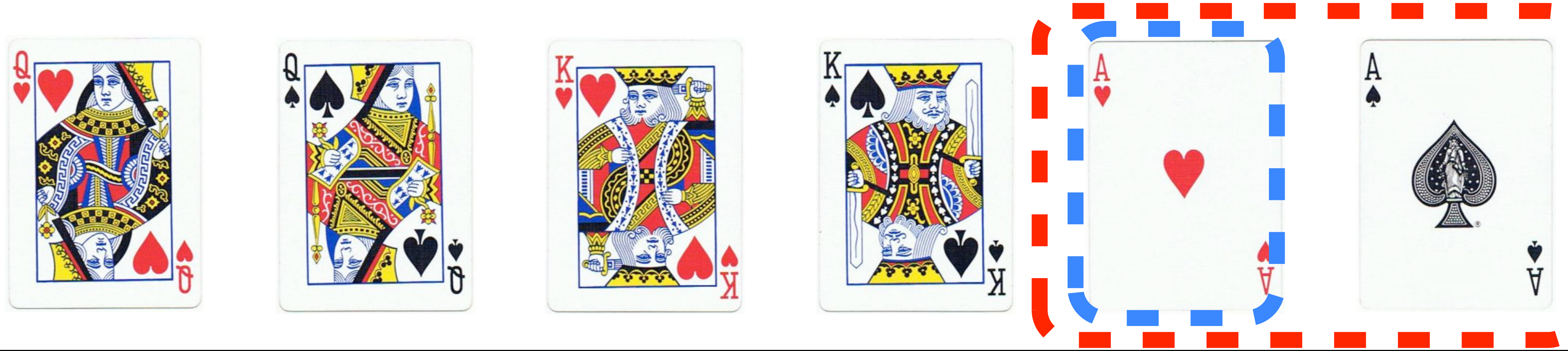
Reach:	0.05	0.1	0.1	0.05	0.1	0.1
Value:	-0.35u	-0.35u	0.05u	0.05u		

```
sum_win_prob = 0.3    sum_lose_prob = 0.20

for( s = each set of equal-strength hands )
  for( i = each tied hand in s )
    sum_lose_prob -= r[i];
  for( i = each tied hand in s )
    v[i] = -u*sum_lose_prob + u*sum_win_prob
  for( i = each tied hand in s )
    sum_win_prob += r[i];
```

Home

3: Fast Terminal Node Evaluation



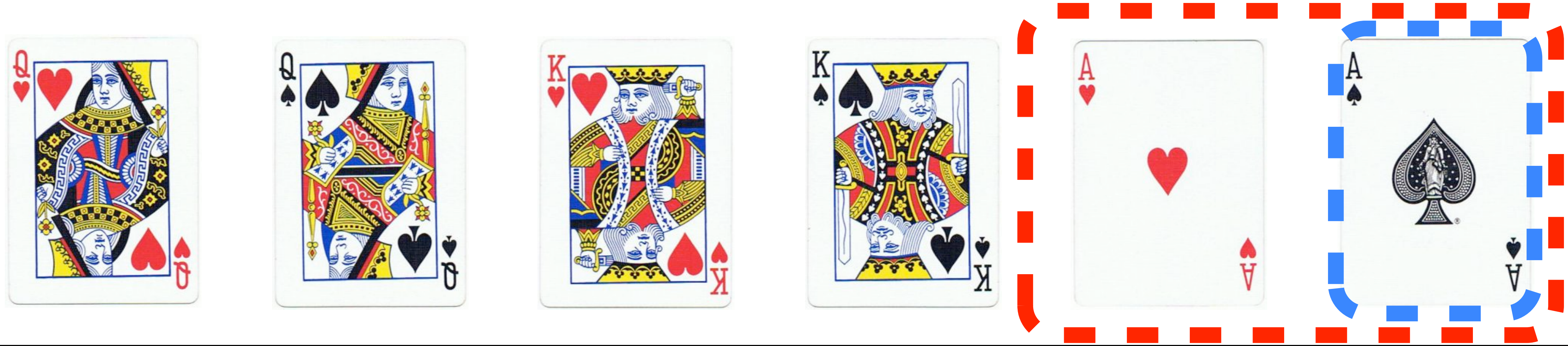
Reach:	0.05	0.1	0.1	0.05	0.1	0.1
Value:	-0.35u	-0.35u	0.05u	0.05u		

```
sum_win_prob = 0.3    sum_lose_prob = 0.10

for( s = each set of equal-strength hands )
  for( i = each tied hand in s )
    sum_lose_prob -= r[i];
  for( i = each tied hand in s )
    v[i] = -u*sum_lose_prob + u*sum_win_prob
  for( i = each tied hand in s )
    sum_win_prob += r[i];
```

Home

3: Fast Terminal Node Evaluation



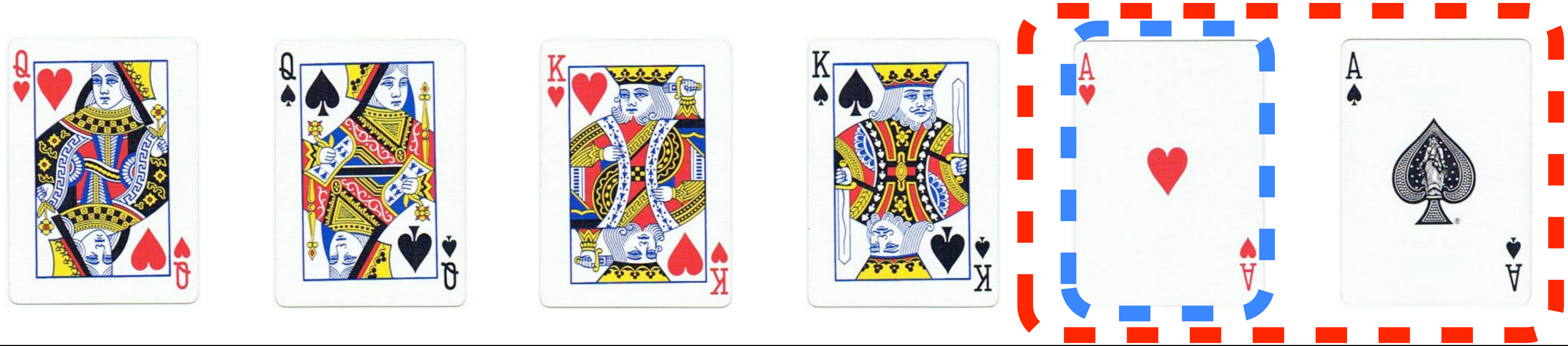
Reach:	0.05	0.1	0.1	0.05	0.1	0.1
Value:	-0.35u	-0.35u	0.05u	0.05u		

```
sum_win_prob = 0.3    sum_lose_prob = 0.0
```

```
for( s = each set of equal-strength hands )
  for( i = each tied hand in s )
    sum_lose_prob -= r[i];
  for( i = each tied hand in s )
    v[i] = -u*sum_lose_prob + u*sum_win_prob
  for( i = each tied hand in s )
     sum_win_prob += r[i];
```

Home

3: Fast Terminal Node Evaluation



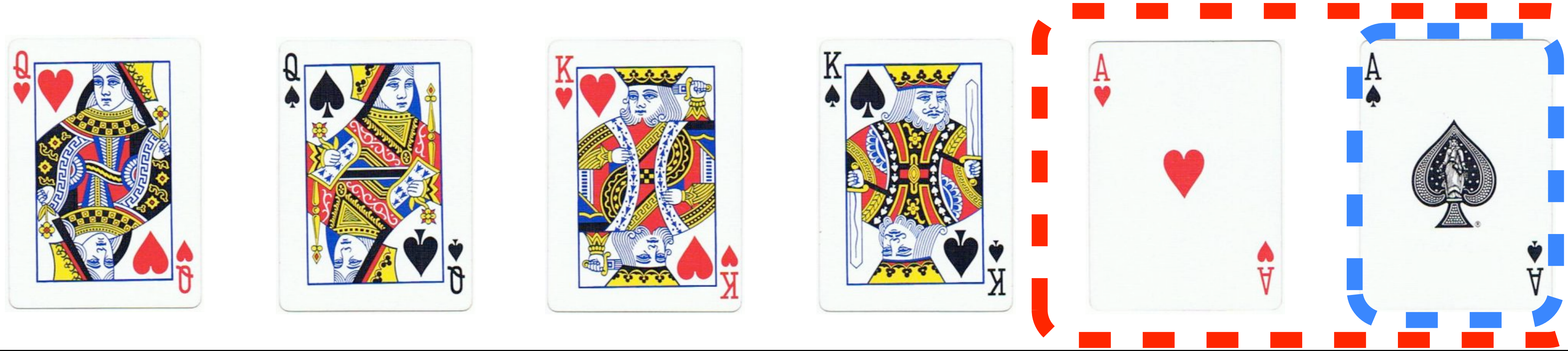
Reach:	0.05	0.1	0.1	0.05	0.1	0.1
Value:	-0.35u	-0.35u	0.05u	0.05u	0.3u	

```
sum_win_prob = 0.3    sum_lose_prob = 0.0

for( s = each set of equal-strength hands )
  for( i = each tied hand in s )
    sum_lose_prob -= r[i];
  for( i = each tied hand in s )
    v[i] = -u*sum_lose_prob + u*sum_win_prob
  for( i = each tied hand in s )
    sum_win_prob += r[i];
```

Home

3: Fast Terminal Node Evaluation



Reach:	0.05	0.1	0.1	0.05	0.1	0.1
Value:	-0.35u	-0.35u	0.05u	0.05u	0.3u	0.3u

```
sum_win_prob = 0.3    sum_lose_prob = 0.0
```

```
for( s = each set of equal-strength hands )
  for( i = each tied hand in s )
    sum_lose_prob -= r[i];
  for( i = each tied hand in s )
    v[i] = -u*sum_lose_prob + u*sum_win_prob
  for( i = each tied hand in s )
    sum_win_prob += r[i];
```

Home