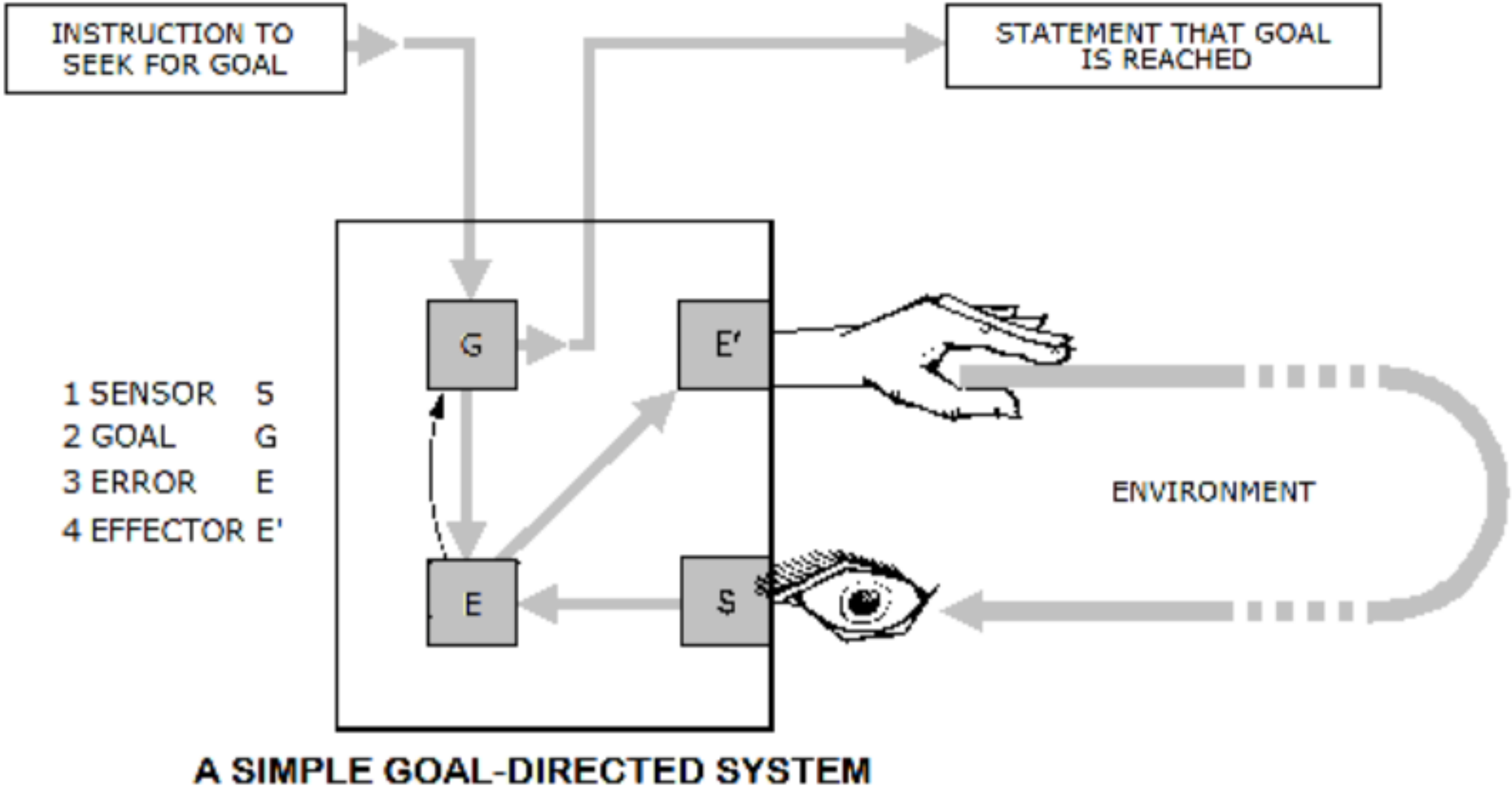
Quest to design intelligent machine Search, Decisions, Games, Learning, …

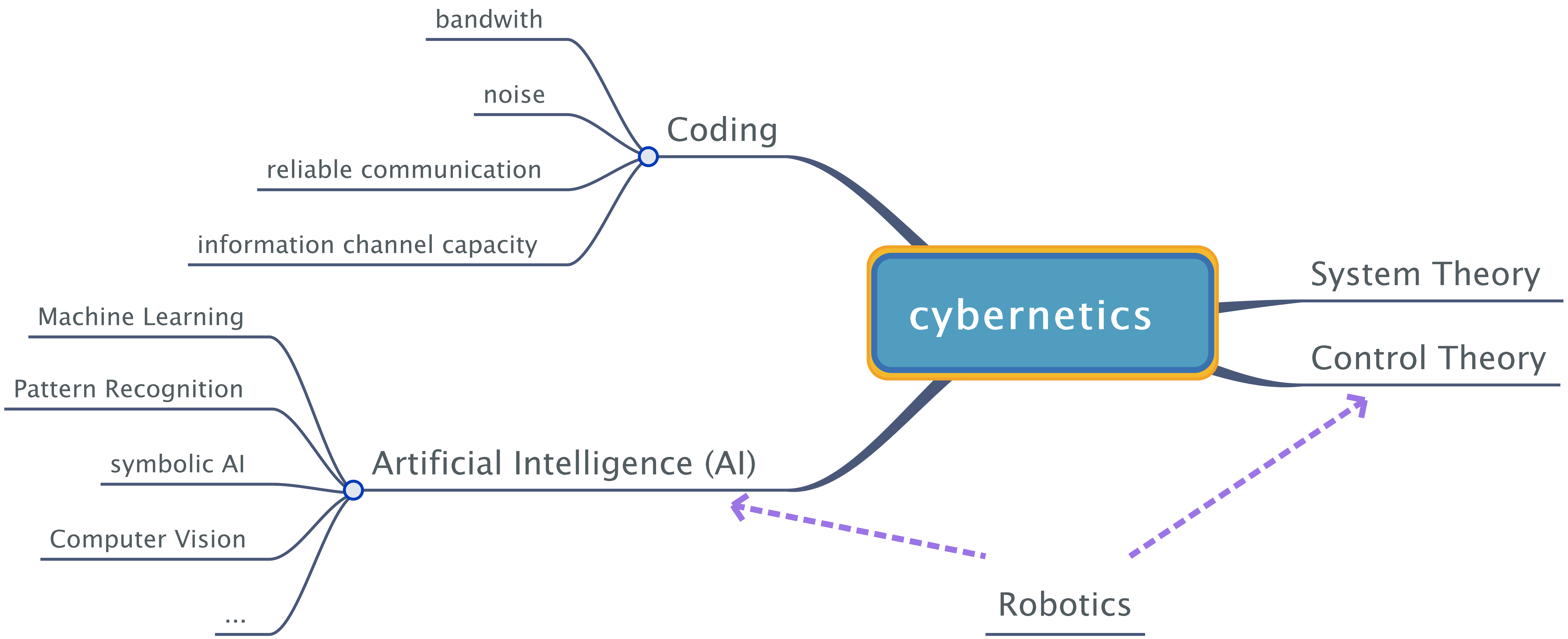
Tomáš Svoboda, Petr Pošík, Jana Kostlivá, Kateřina Poláková, Jan Černý, **B3B33KUI 2022/2023**

Course target: goal-directed system

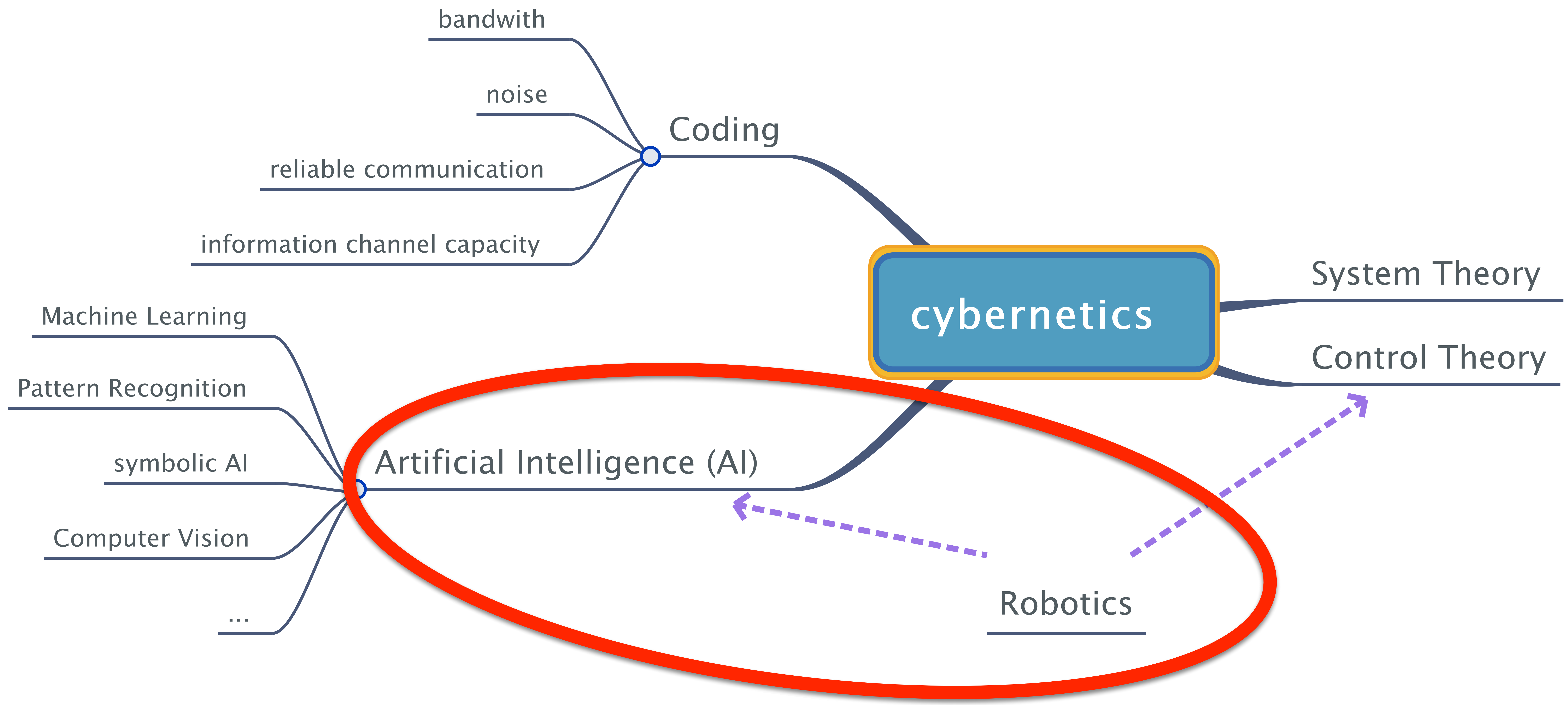


Pask, Gordon (1972). "Cybernetics". Encyclopædia Britannica.

cybernetics now

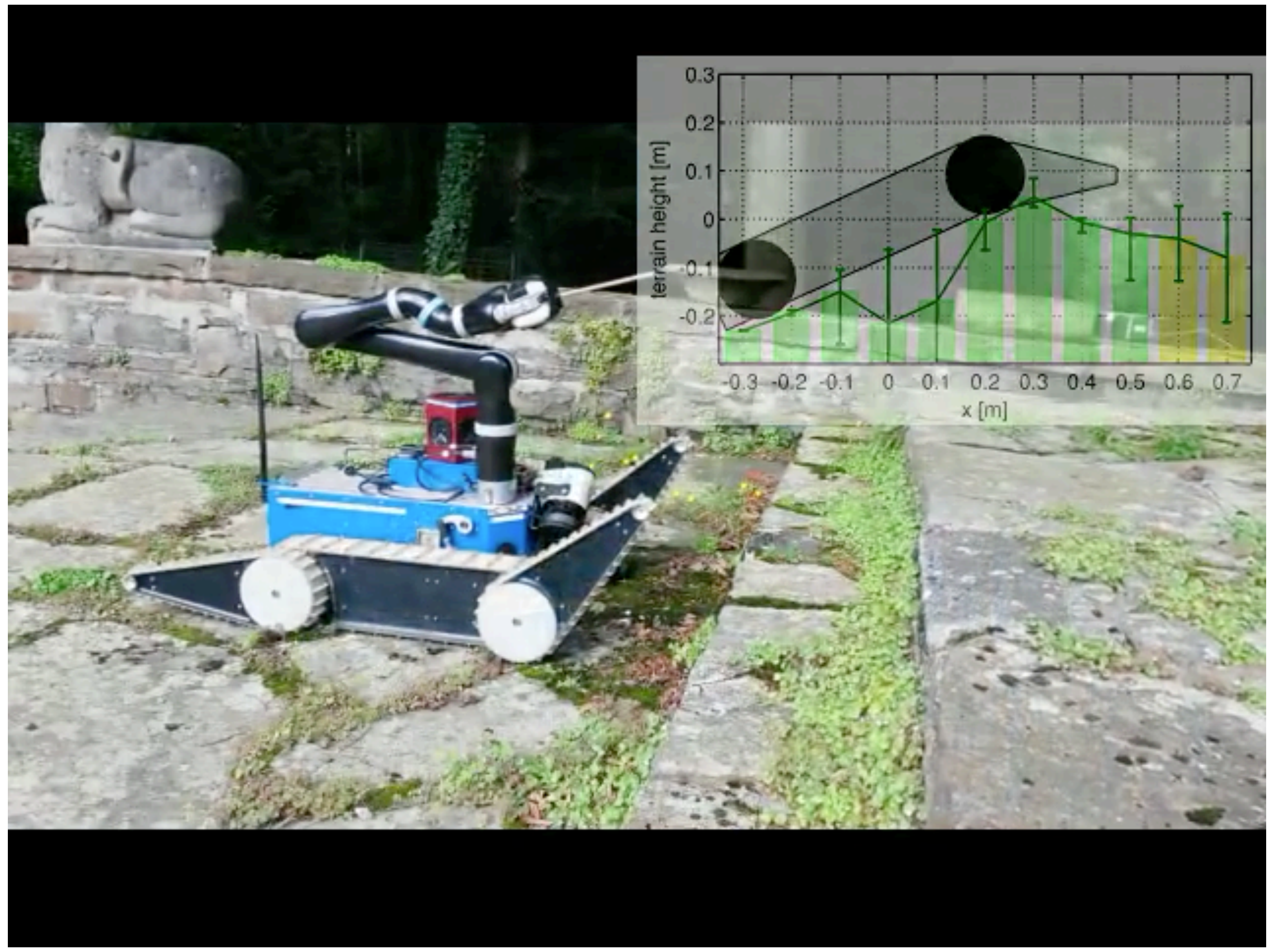


cybernetics now



- our motivation from (intelligent) robotics
- yet basic concepts from cybernetics
- modern terminology will be used

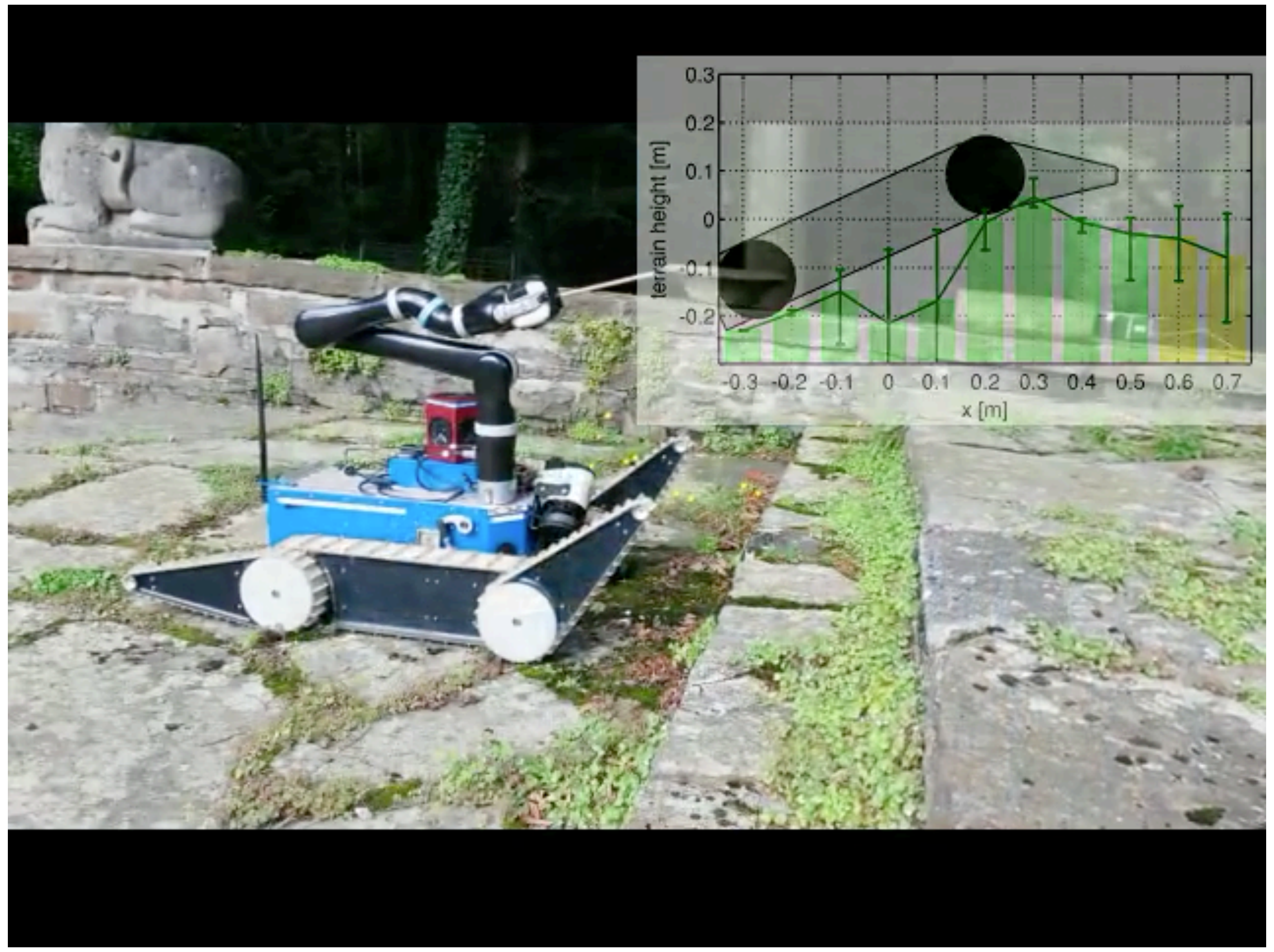
where we stand 50 years later:
machine control in unstructured environment



V. Salansky, K. Zimmermann, T. Petricek, T. Svoboda. Pose consistency KKT-loss for weakly supervised learning of robot-terrain interaction model. IEEE Robotics and Automation Letters, 2021, Volume 6, Issue 3.
M. Pecka, K. Zimmermann, M. Reinstein, and T. Svoboda. Controlling Robot Morphology from Incomplete Measurements. In *IEEE Transactions on Industrial Electronics*, Feb 2017, Vol 64, Issue: 2
V. Šalanský, V. Kubelka, K. Zimmermann, M. Reinstein, T. Svoboda. Touching without vision: terrain perception in sensory deprived environments. CVWW 2016

http://www.tradr-project.eu, https://robotics.fel.cvut.cz/cras/darpa-subt/, https://cyber.felk.cvut.cz/category/department/cmp/vras/

where we stand 50 years later: machine control in unstructured environment



V. Salansky, K. Zimmermann, T. Petricek, T. Svoboda. Pose consistency KKT-loss for weakly supervised learning of robot-terrain interaction model. IEEE Robotics and Automation Letters, 2021, Volume 6, Issue 3.
M. Pecka, K. Zimmermann, M. Reinstein, and T. Svoboda. Controlling Robot Morphology from Incomplete Measurements. In *IEEE Transactions on Industrial Electronics*, Feb 2017, Vol 64, Issue: 2
V. Šalanský, V. Kubelka, K. Zimmermann, M. Reinstein, T. Svoboda. Touching without vision: terrain perception in sensory deprived environments. CVWW 2016

http://www.tradr-project.eu, https://robotics.fel.cvut.cz/cras/darpa-subt/, https://cyber.felk.cvut.cz/category/department/cmp/vras/

Amatrice 2016

https://youtu.be/rTP64z52JFE

DARPA SubTerranean Challenge - Urban Circuit, 2020/02

CTU-CRAS-NORLAB

@DARPA Subterranean Challenge
URBAN CIRCUIT

CTU
CZECH TECHNICAL UNIVERSITY IN PRAGUE

VRAS

AI CENTER

MRS

https://youtu.be/rTP64z52JFE

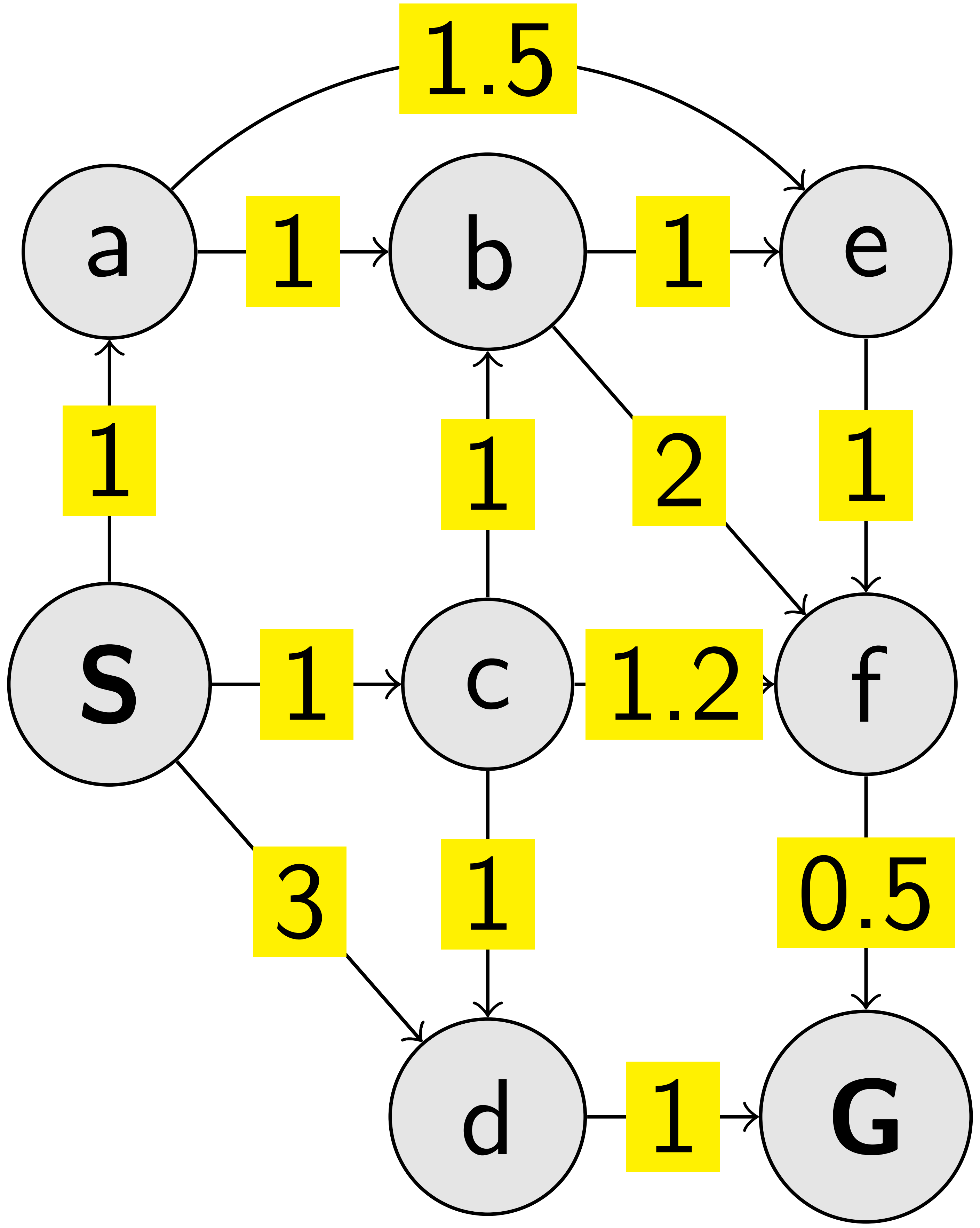
http://robotics.fel.cvut.cz/cras/darpa-subt/

DARPA SubTerranean Challenge - Urban Circuit, 2020/02

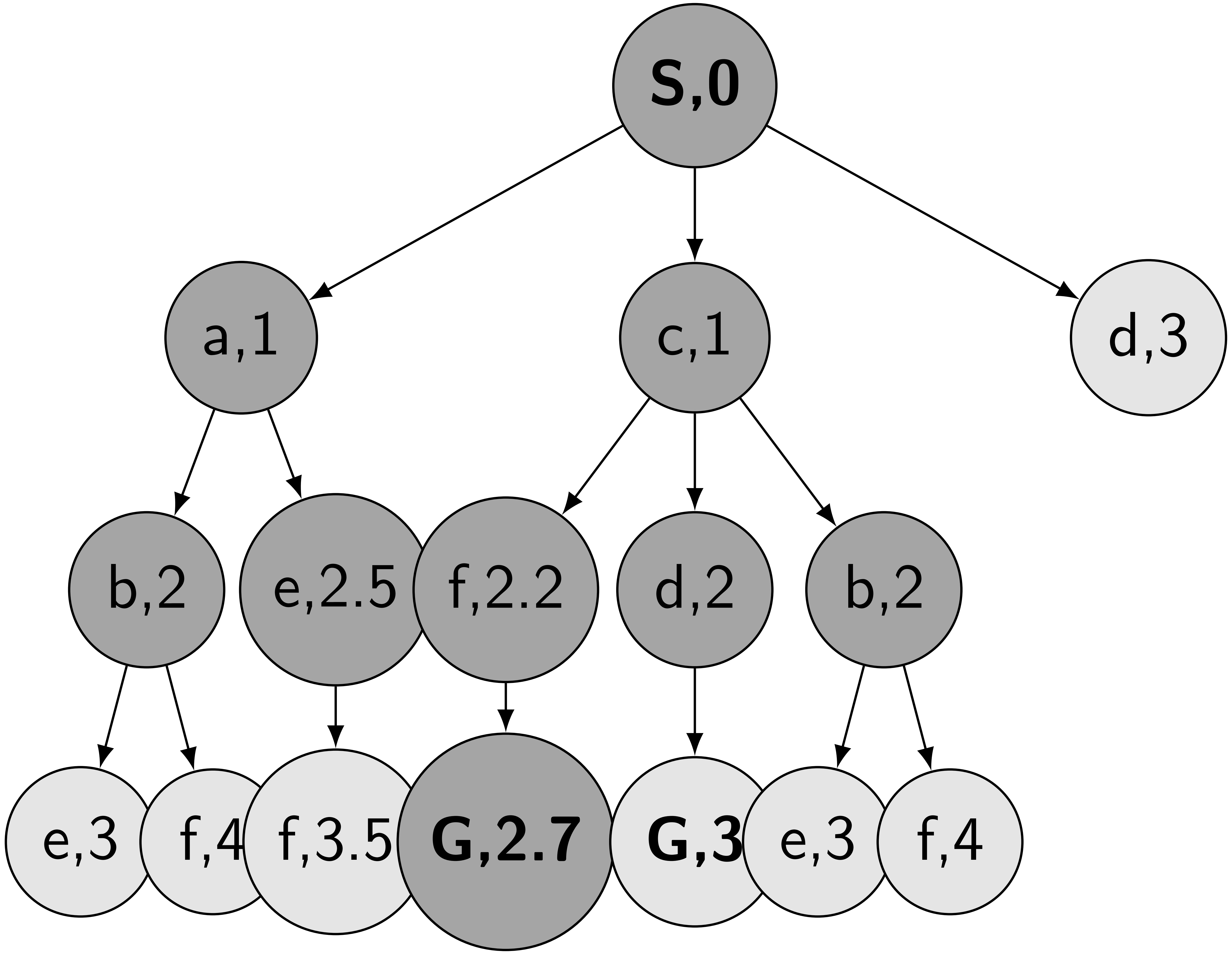
https://youtu.be/HzBh6QdySDI
https://robotics.fel.cvut.cz/cras/darpa-subt/

https://youtu.be/HzBh6QdySDI
https://robotics.fel.cvut.cz/cras/darpa-subt/

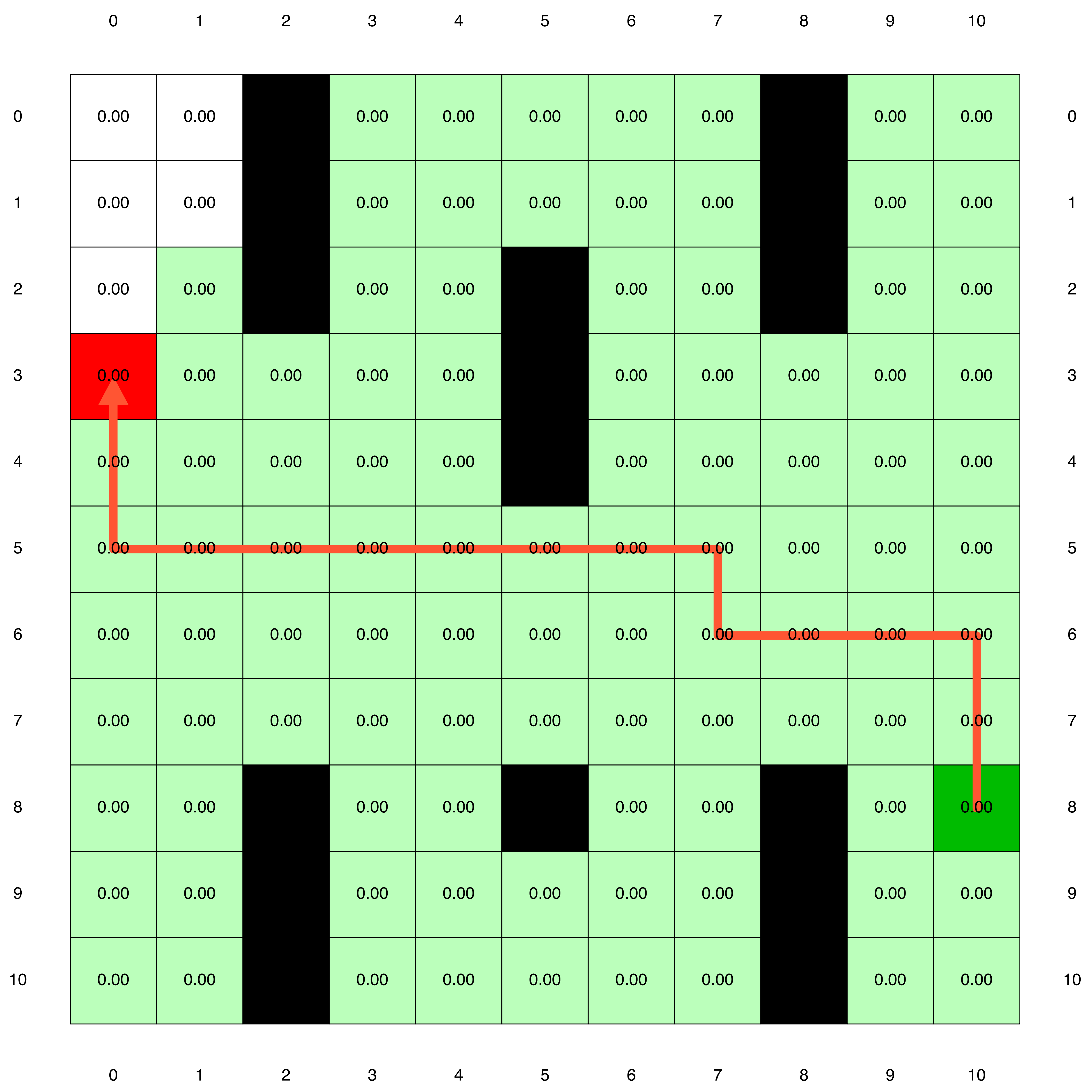
Problem: graph with costs

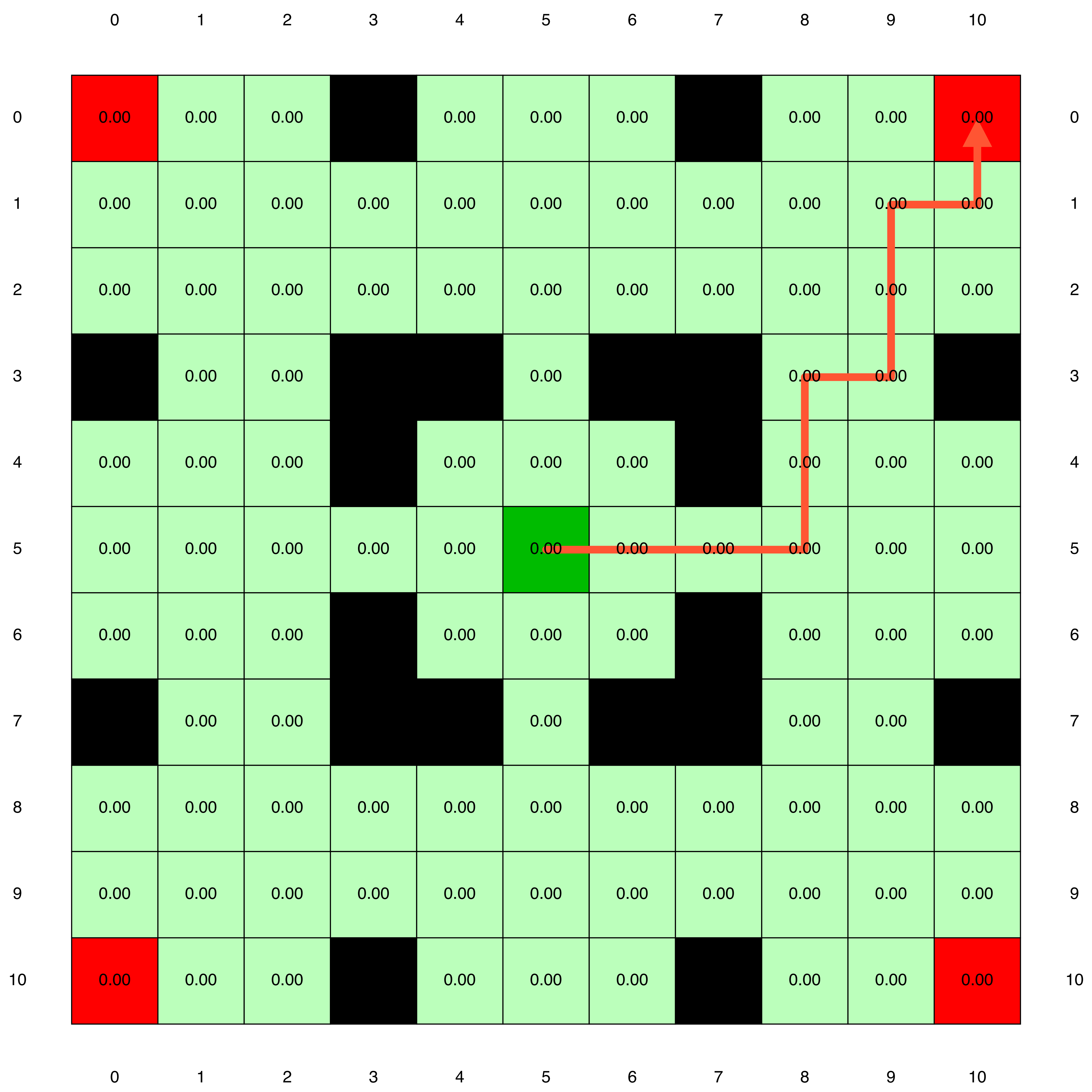


Complete, optimal **search (plan)**



Solution: Path (shortest, chapest, …)





State cost/value: $f(S_t) = g(S_t) + h(S_t)$

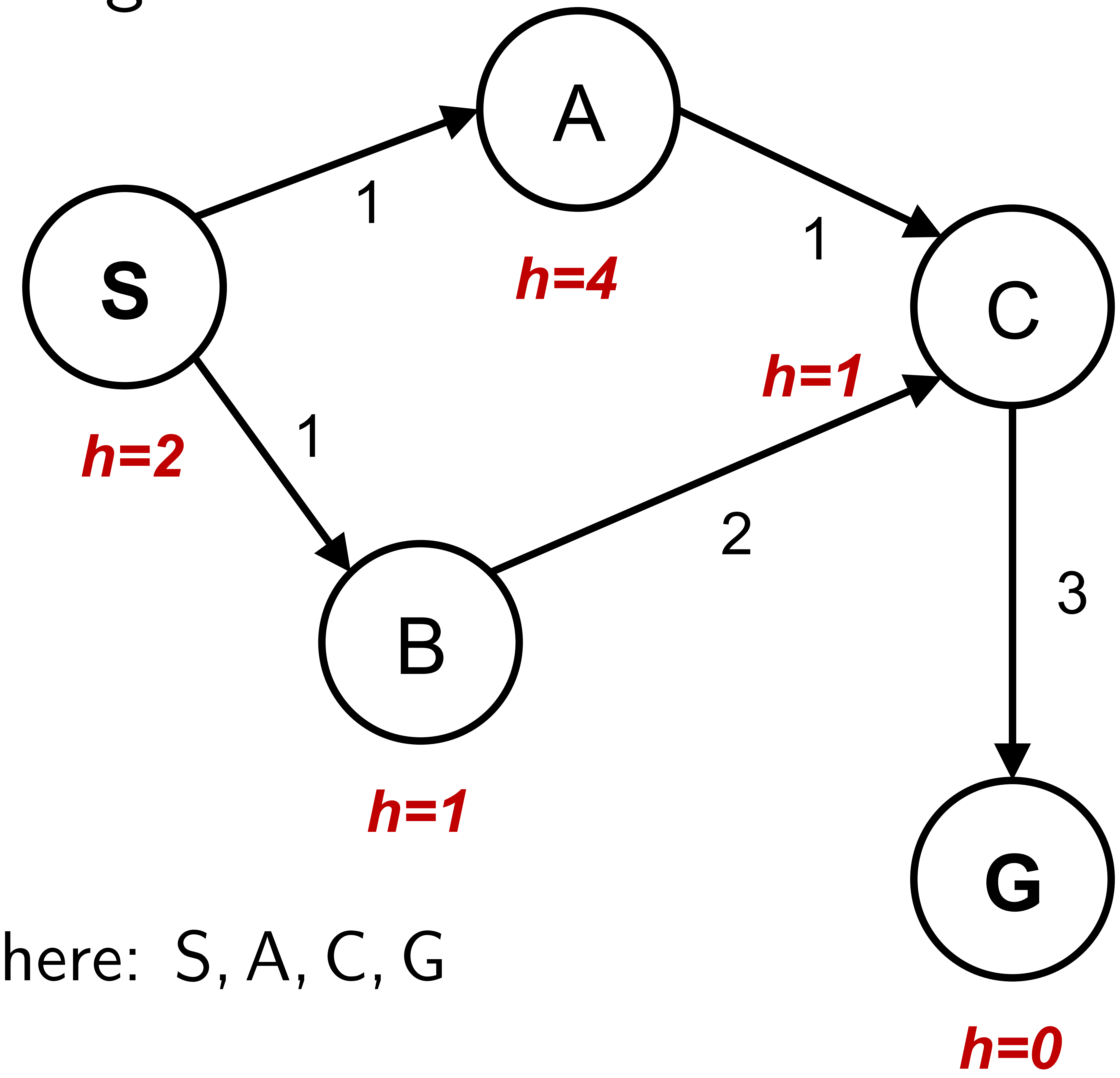
Backward value/cost, accumulates as it goes

$g(S_t) = g(S_{t-1}) + c(S_{t-1}, S_t)$

$g(C) = g(A) + c(A, C)$

Forward cost, guess of

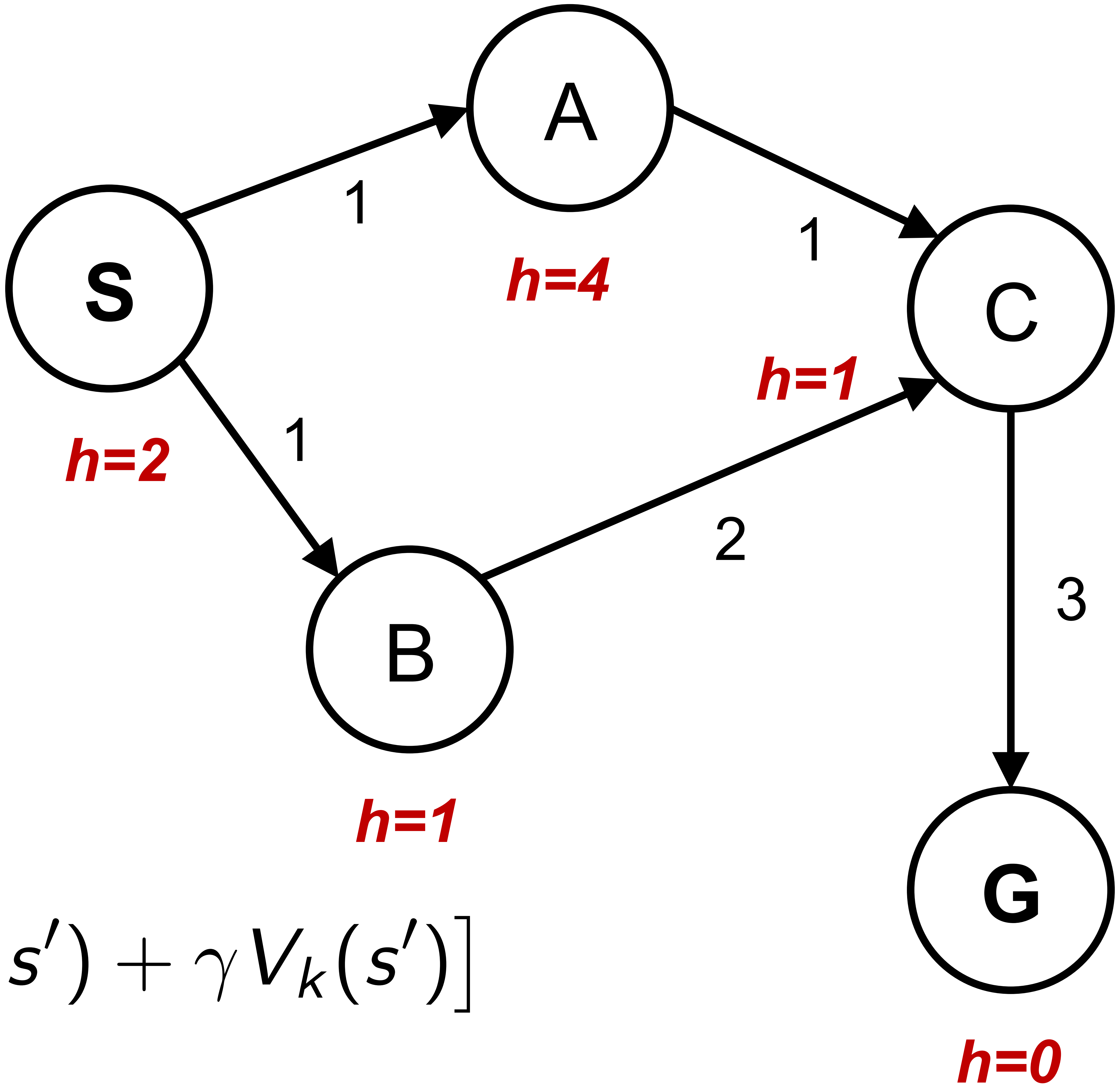
$h(S_t) \approx c(S_t, G)$



Solution minimizes overall cost.

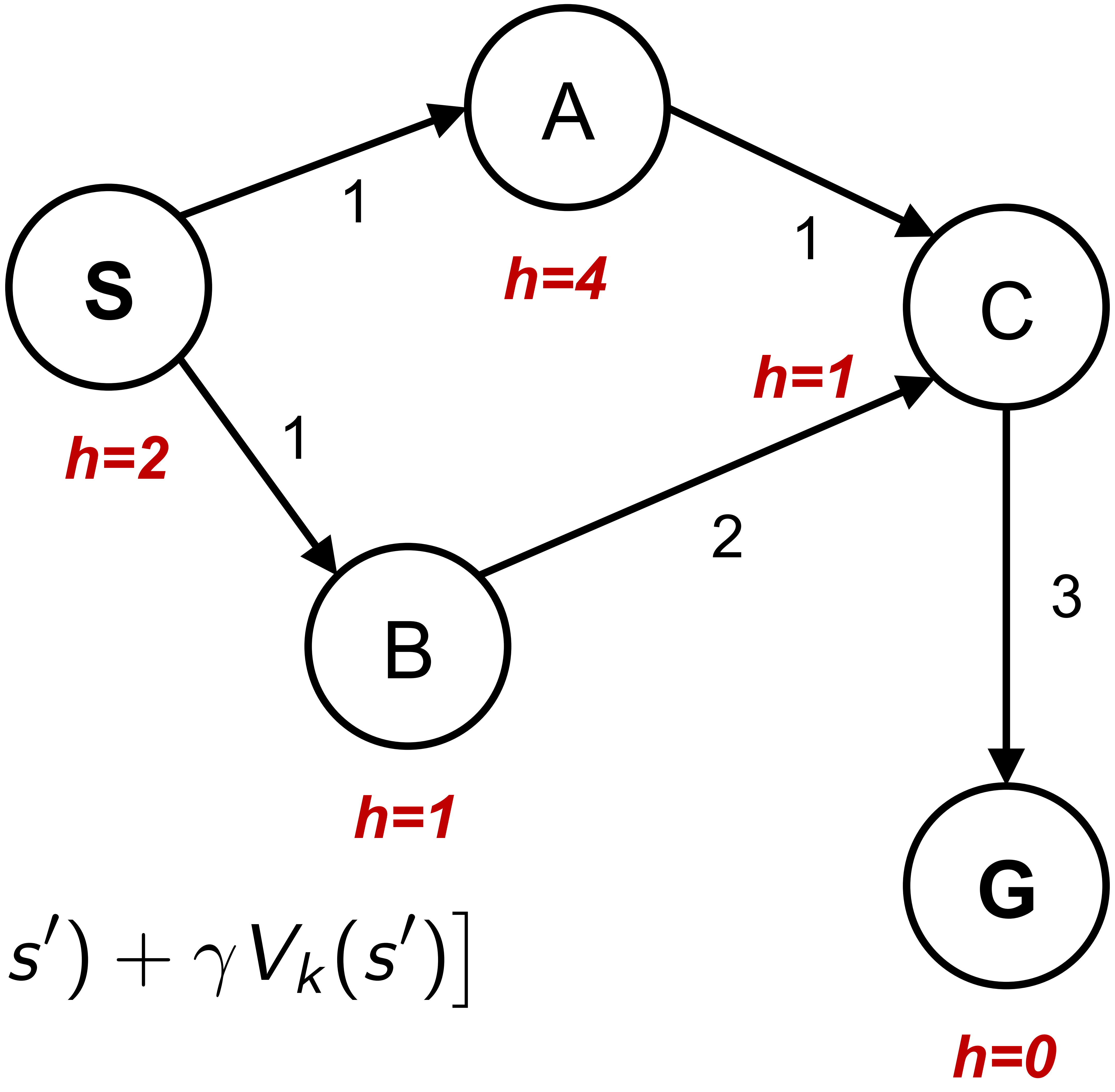
From Start to Goal (terminal):

$\sum_S^G f(S_t)$ Solution: $S_{t=0}, S_1, S_2, \ldots, G$; here: S, A, C, G



$$V_{k+1}(s) \leftarrow \max_{a \in A(s)} \sum_{s'} p(s'|s,a)\left[r(s,a,s') + \gamma V_k(s')\right]$$

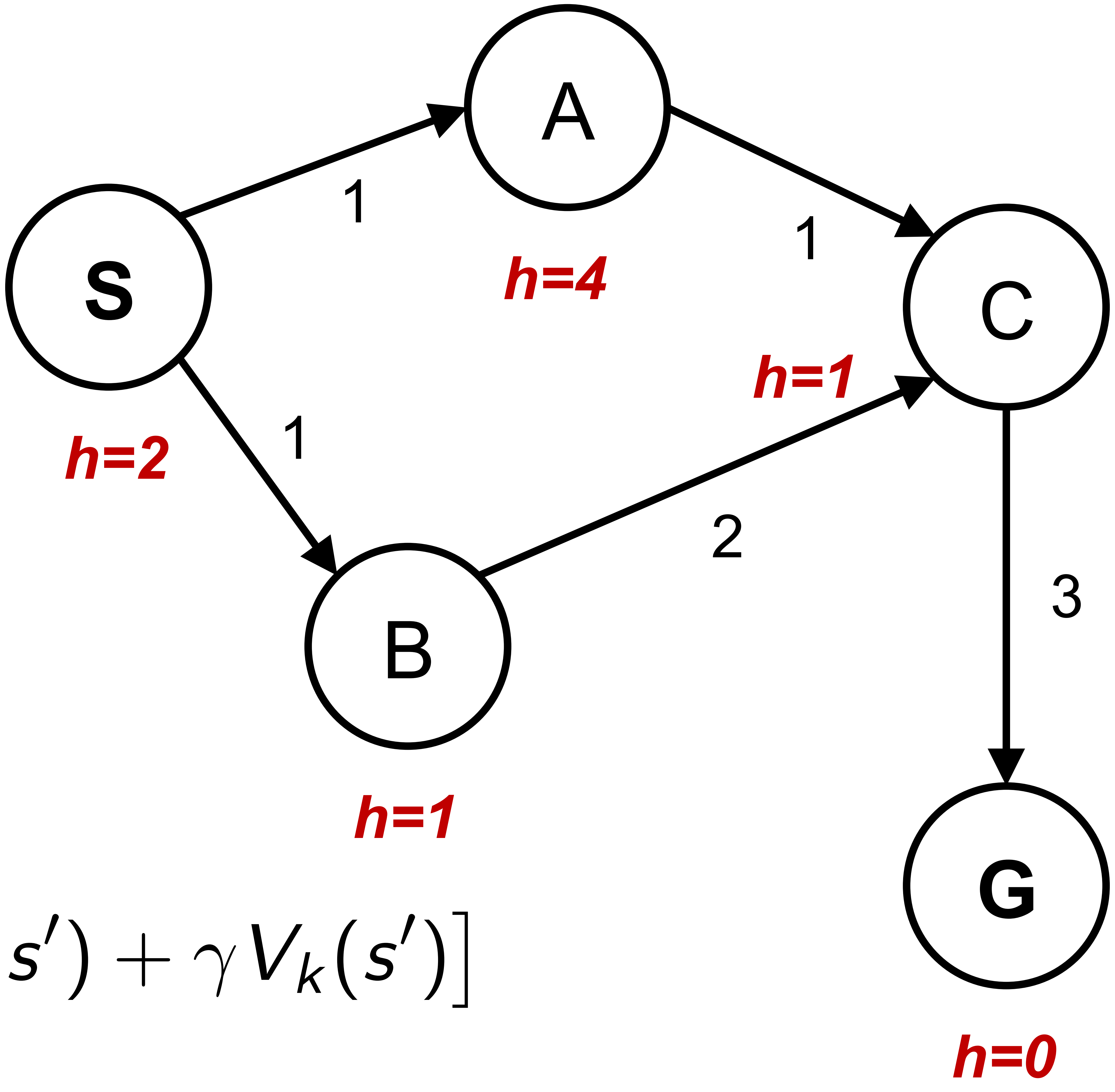
Maximize sum of (expected) rewards, *(state) Value iteration*:



$$V_{k+1}(s) \leftarrow \max_{a \in A(s)} \sum_{s'} p(s'|s,a)\left[r(s,a,s') + \gamma V_k(s')\right]$$

Maximize sum of (expected) rewards, *(state) Value iteration*:

assume deterministic robot, no discounting

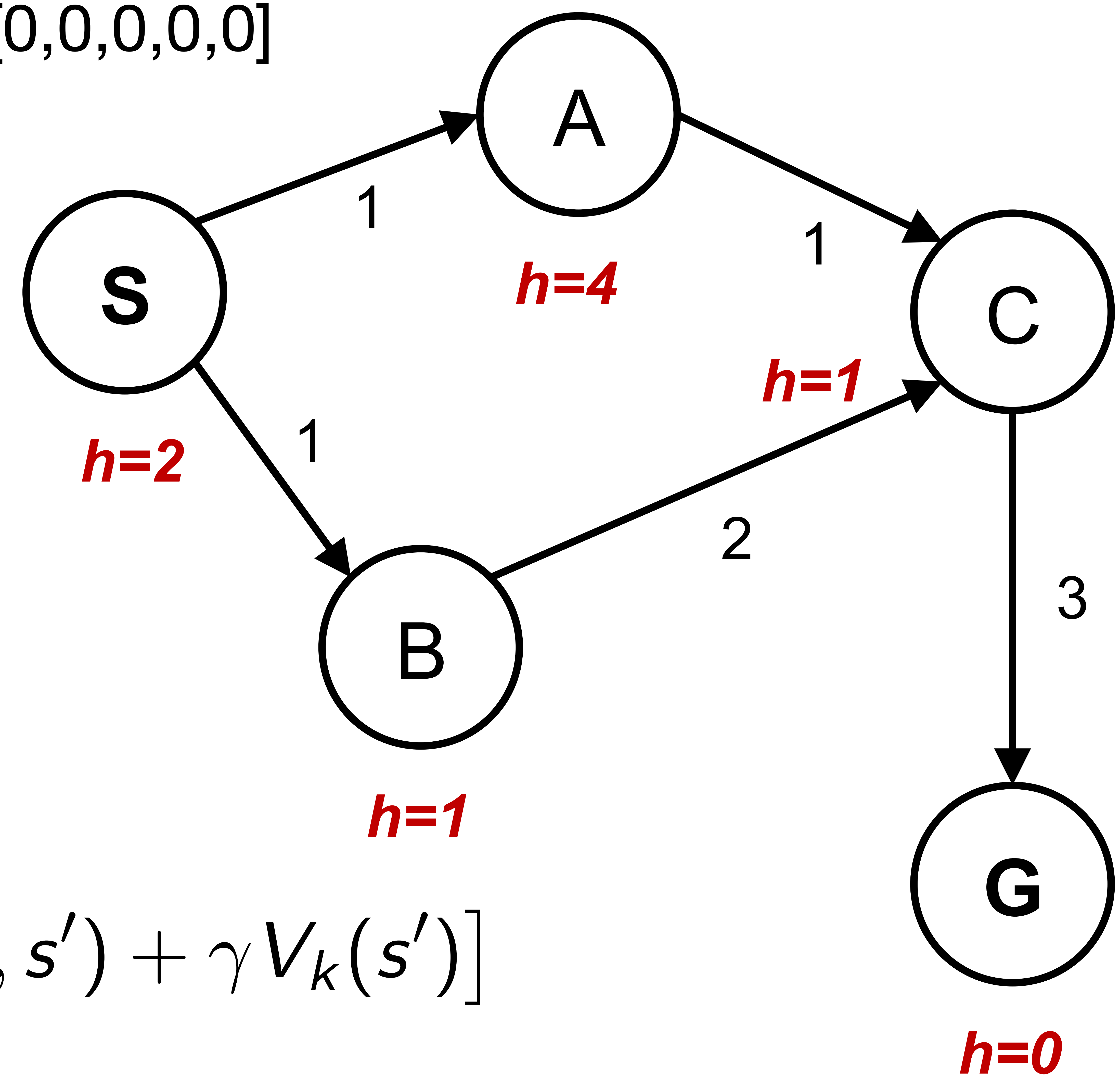


$$V_{k+1}(s) \leftarrow \max_{a \in A(s)} \sum_{s'} p(s'|s,a) \left[r(s,a,s') + \gamma V_k(s') \right]$$

Maximize sum of (expected) rewards, *(state) Value iteration*:

assume deterministic robot, no discounting

- init all V(s)=0, [V(S),V(A),V(B),V(C),V(G)] = [0,0,0,0,0]

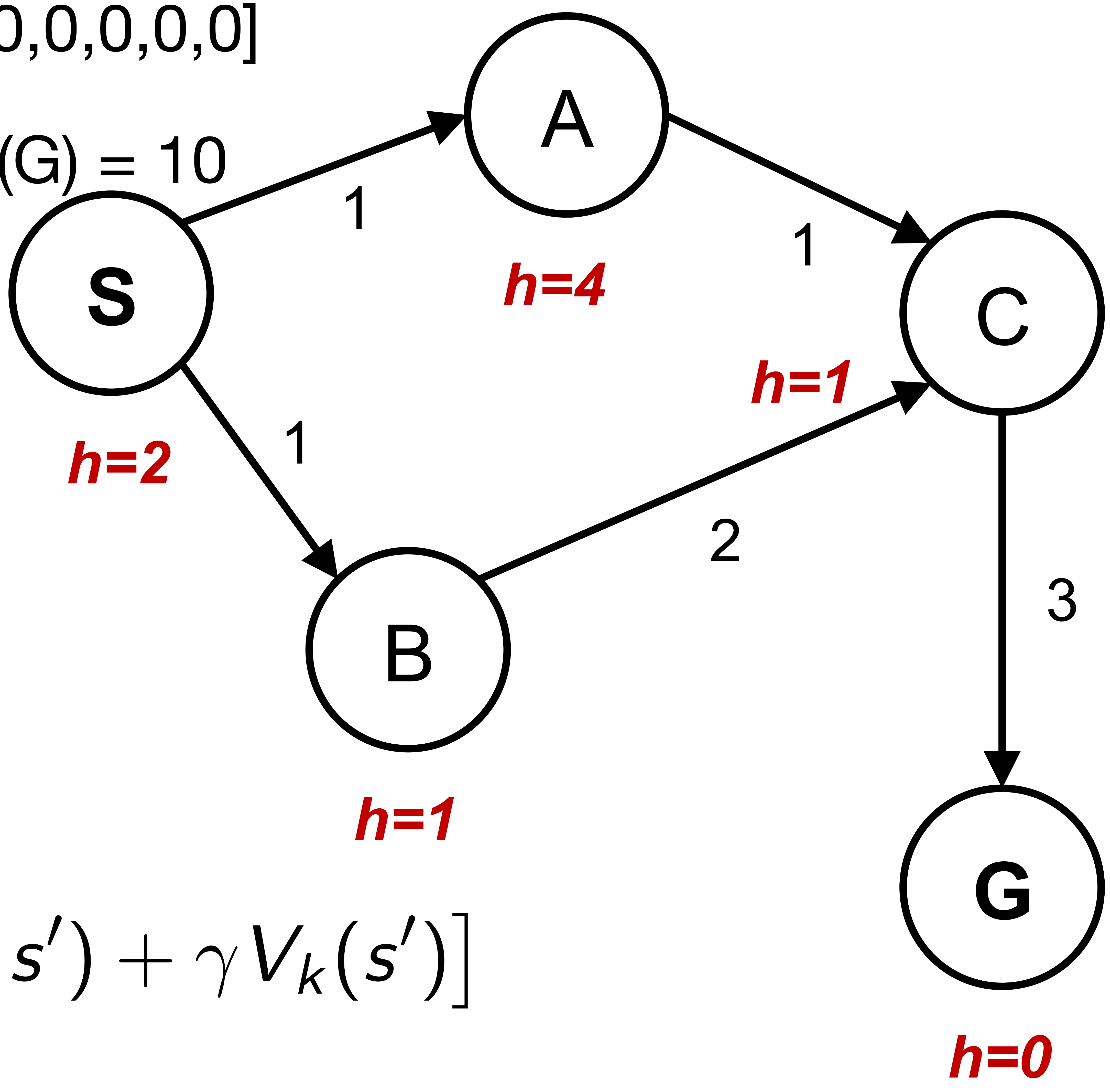


$$V_{k+1}(s) \leftarrow \max_{a \in A(s)} \sum_{s'} p(s'|s,a)\left[r(s,a,s') + \gamma V_k(s')\right]$$

Maximize sum of (expected) rewards, *(state) Value iteration*:

assume deterministic robot, no discounting

- init all V(s)=0, [V(S),V(A),V(B),V(C),V(G)] = [0,0,0,0,0]
- V(S) = -1, V(A) = -1, V(B) = -2, V(C) = -3, V(G) = 10

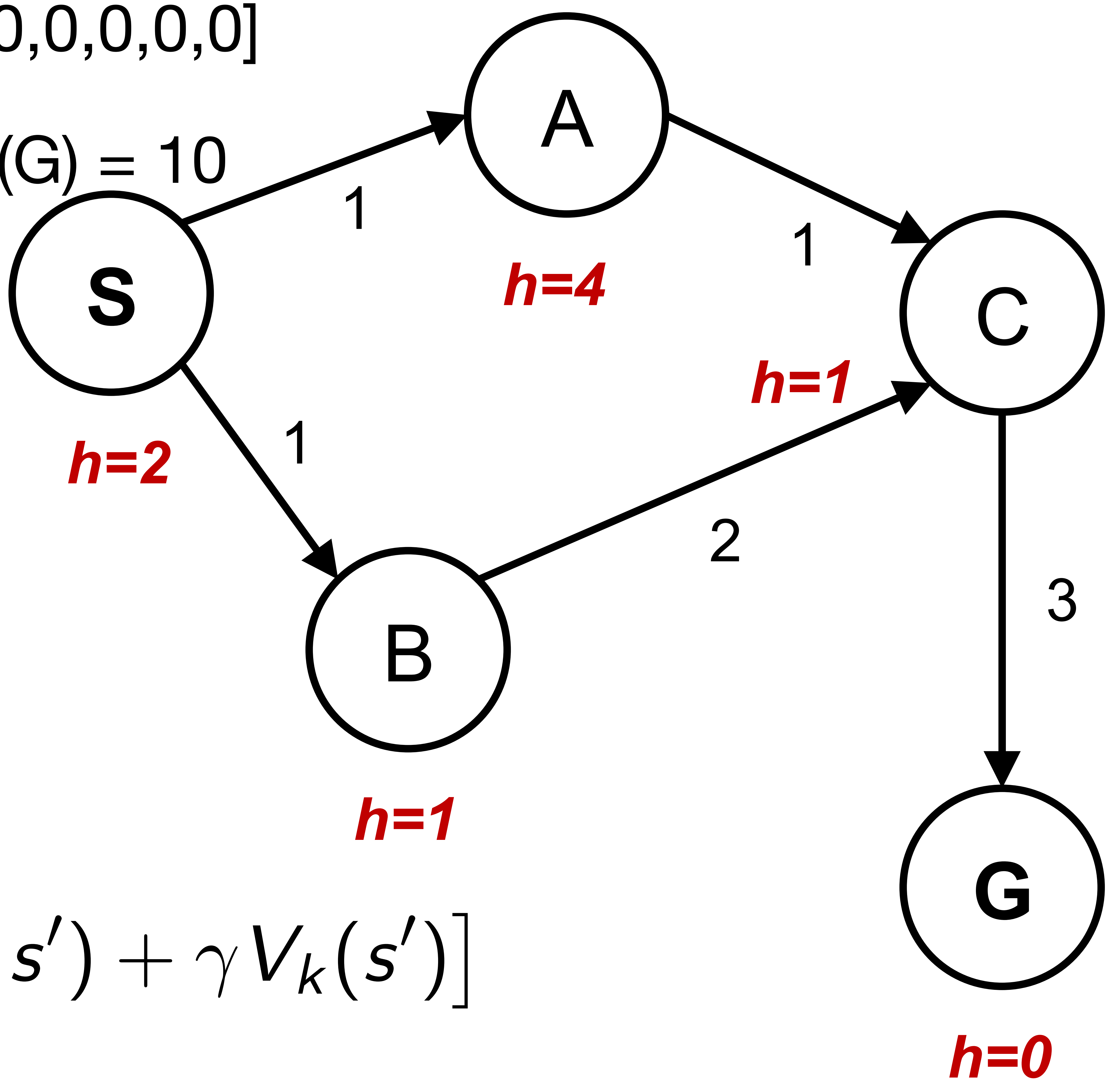


$$V_{k+1}(s) \leftarrow \max_{a \in A(s)} \sum_{s'} p(s'|s,a)\left[r(s,a,s') + \gamma V_k(s')\right]$$

Maximize sum of (expected) rewards, *(state) Value iteration*:

assume deterministic robot, no discounting

- init all V(s)=0, [V(S),V(A),V(B),V(C),V(G)] = [0,0,0,0,0]
- V(S) = -1, V(A) = -1, V(B) = -2, V(C) = -3, V(G) = 10
- [-2, -4, -5, 7, 10]

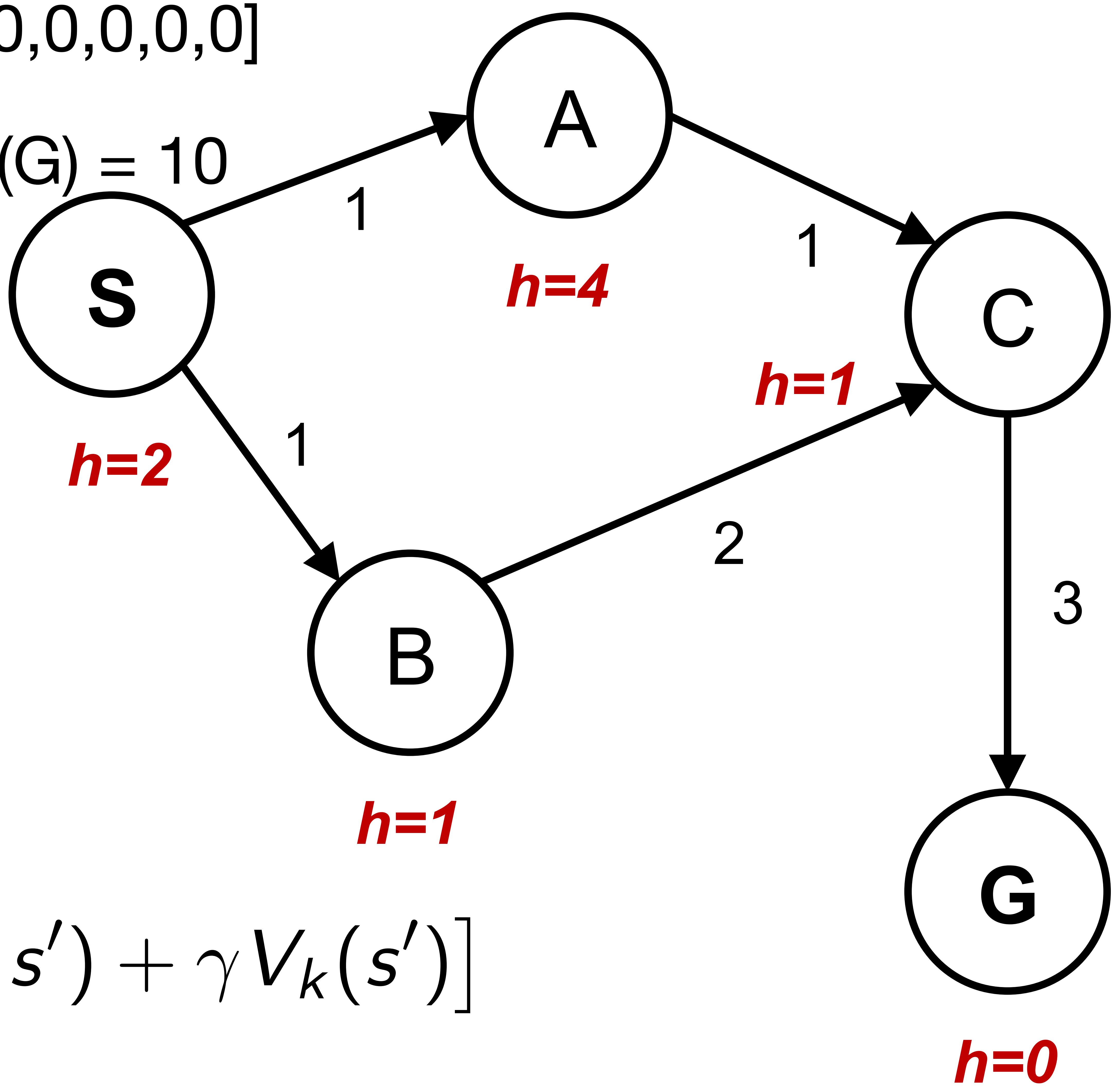


$$V_{k+1}(s) \leftarrow \max_{a \in A(s)} \sum_{s'} p(s'|s,a) \left[r(s,a,s') + \gamma V_k(s') \right]$$

Maximize sum of (expected) rewards, *(state) Value iteration*:

assume deterministic robot, no discounting

- init all V(s)=0, [V(S),V(A),V(B),V(C),V(G)] = [0,0,0,0,0]
- V(S) = -1, V(A) = -1, V(B) = -2, V(C) = -3, V(G) = 10
- [-2, -4, -5, 7, 10]
- [-5, 6, 5, 7, 10]

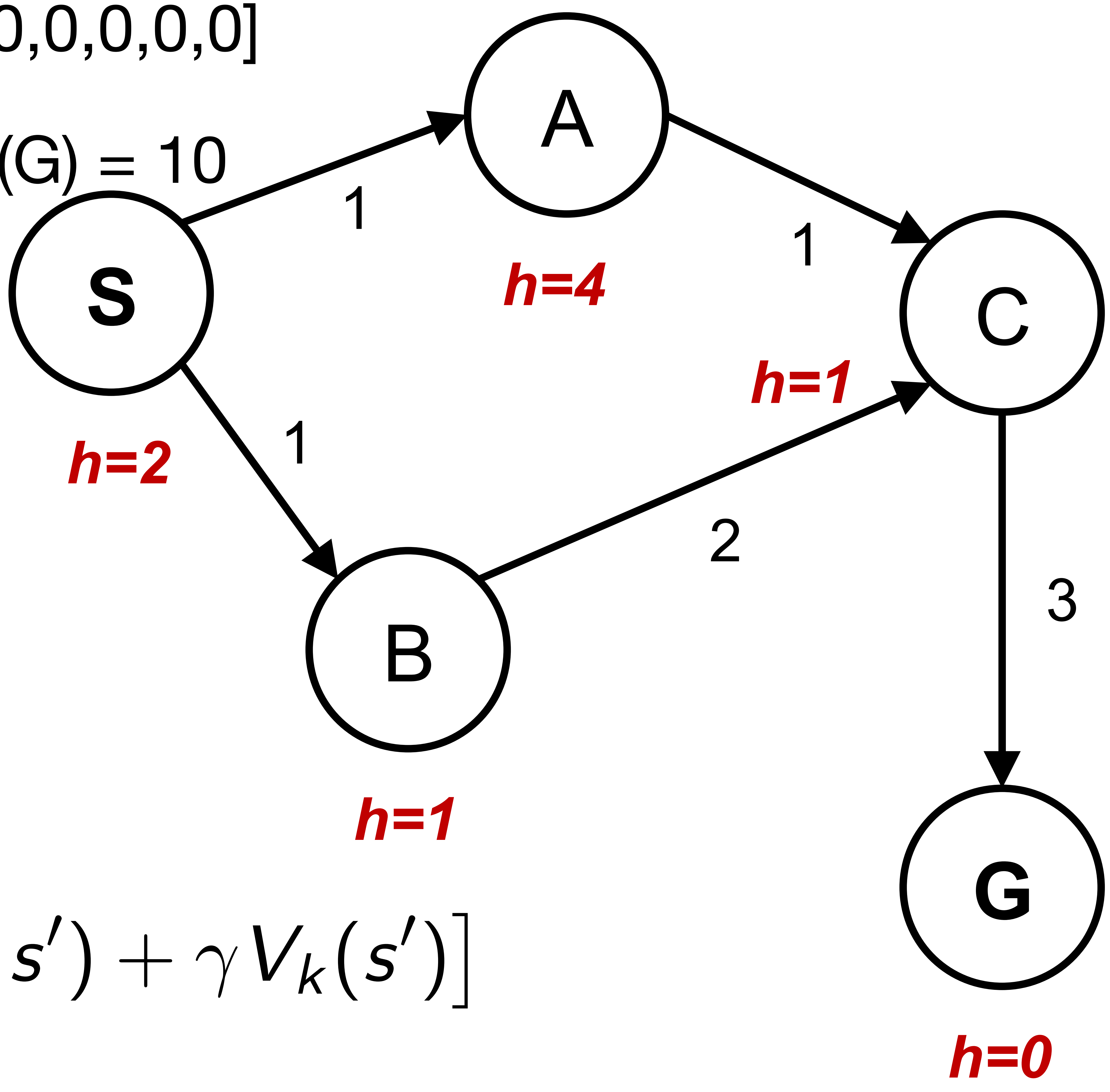


$$V_{k+1}(s) \leftarrow \max_{a \in A(s)} \sum_{s'} p(s'|s,a)\left[r(s,a,s') + \gamma V_k(s')\right]$$

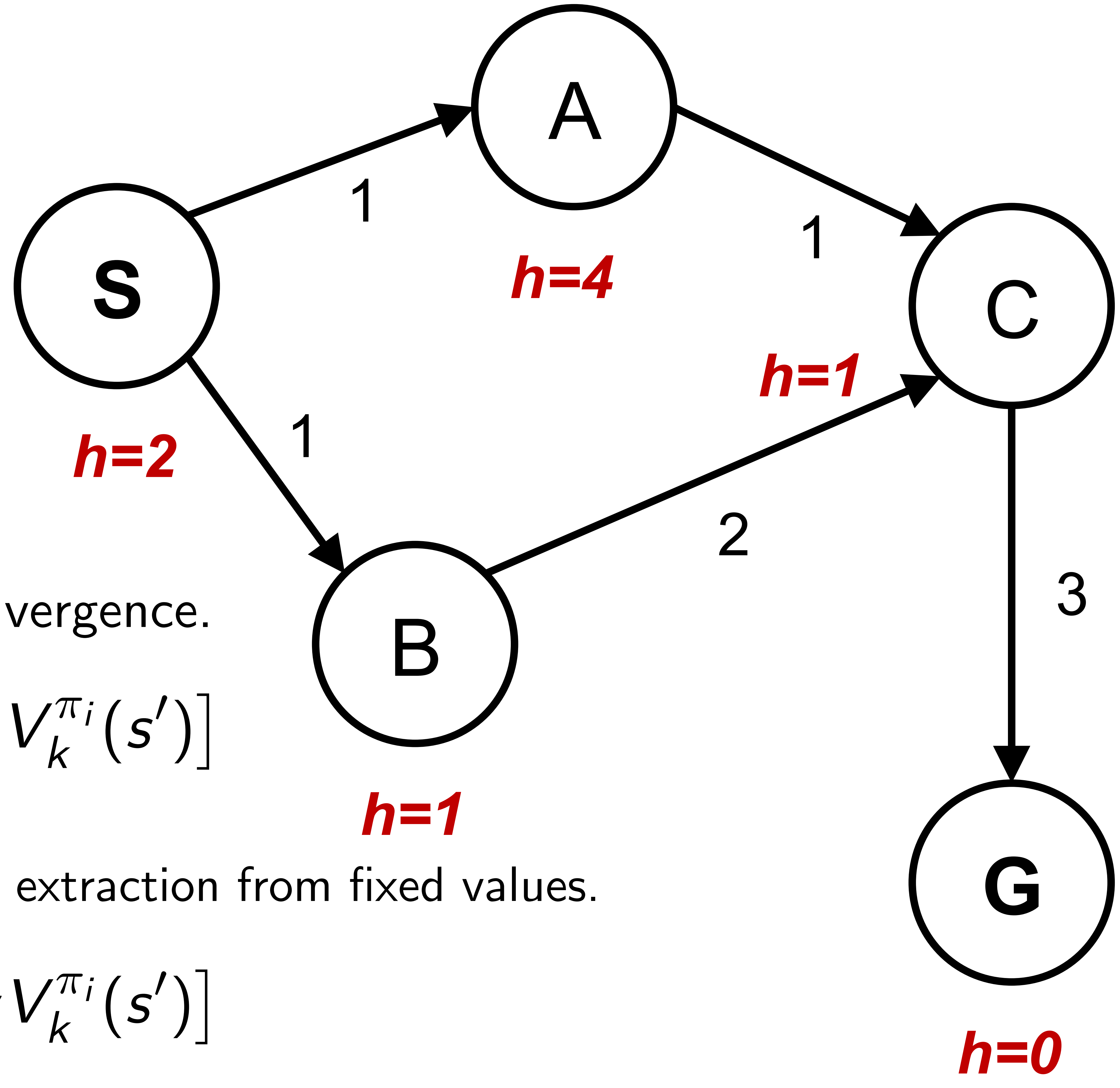
Maximize sum of (expected) rewards, *(state) Value iteration*:

assume deterministic robot, no discounting

- init all V(s)=0, [V(S),V(A),V(B),V(C),V(G)] = [0,0,0,0,0]
- V(S) = -1, V(A) = -1, V(B) = -2, V(C) = -3, V(G) = 10
- [-2, -4, -5, 7, 10]
- [-5, 6, 5, 7, 10]
- [5, 6, 5, 7, 10]



$$V_{k+1}(s) \leftarrow \max_{a \in A(s)} \sum_{s'} p(s'|s,a)\left[r(s,a,s') + \gamma V_k(s')\right]$$



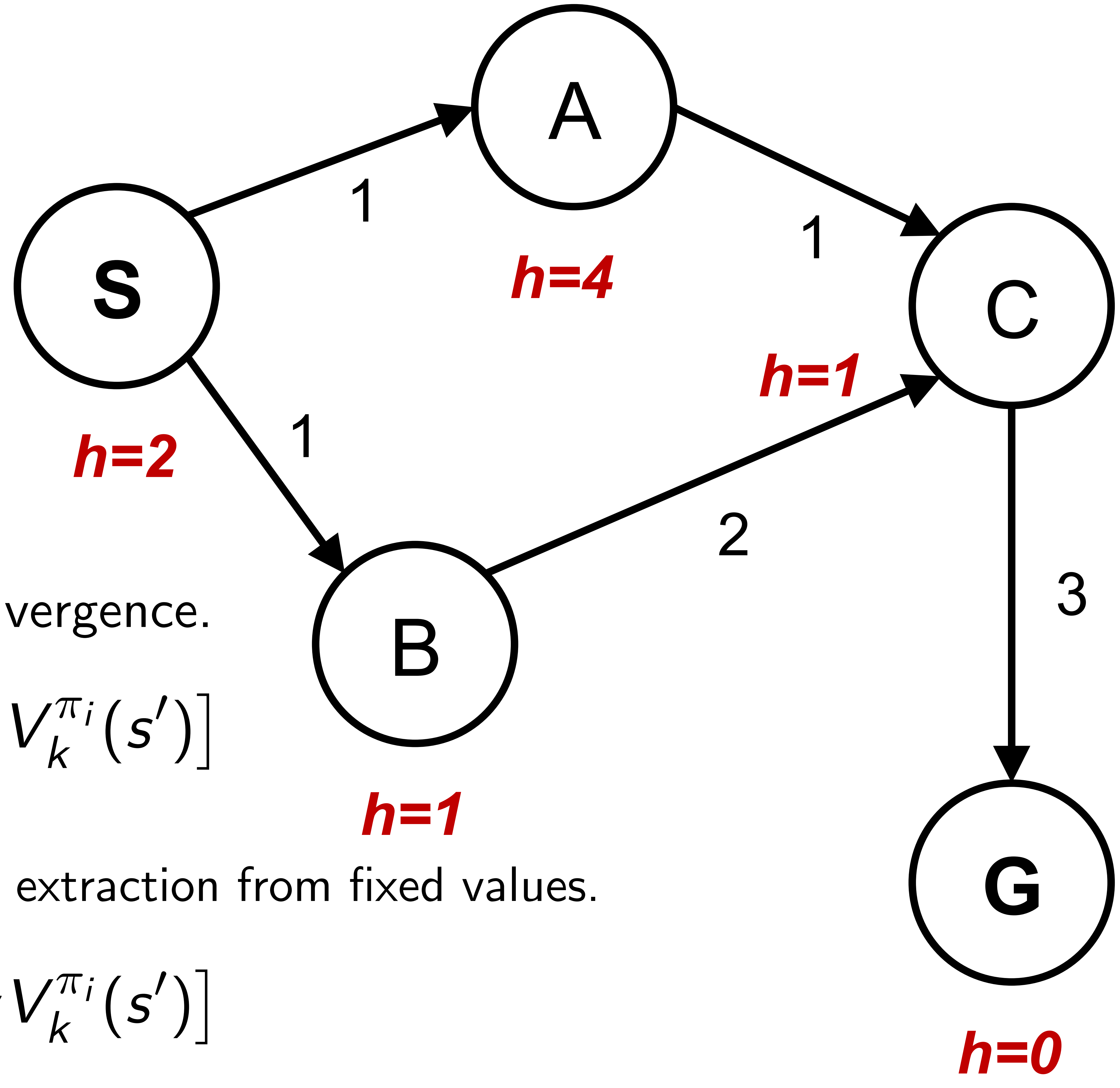
Policy π evaluation. Solve equations or iterate until convergence.

$$V_{k+1}^{\pi_i}(s) \leftarrow \sum_{s'} p(s' \mid s, \pi(s)) \left[r(s, \pi(s), s') + \gamma V_k^{\pi_i}(s')\right]$$

Policy improvement. Look-ahead and keep optimality. Policy extraction from fixed values.

$$\pi_{i+1}(s) = \arg\max_{a \in \mathcal{A}(s)} \sum_{s'} p(s' \mid s, a) \left[r(s, a, s') + \gamma V_k^{\pi_i}(s')\right]$$

Maximize sum of (expected) rewards, *Policy iteration*:



Policy π evaluation. Solve equations or iterate until convergence.

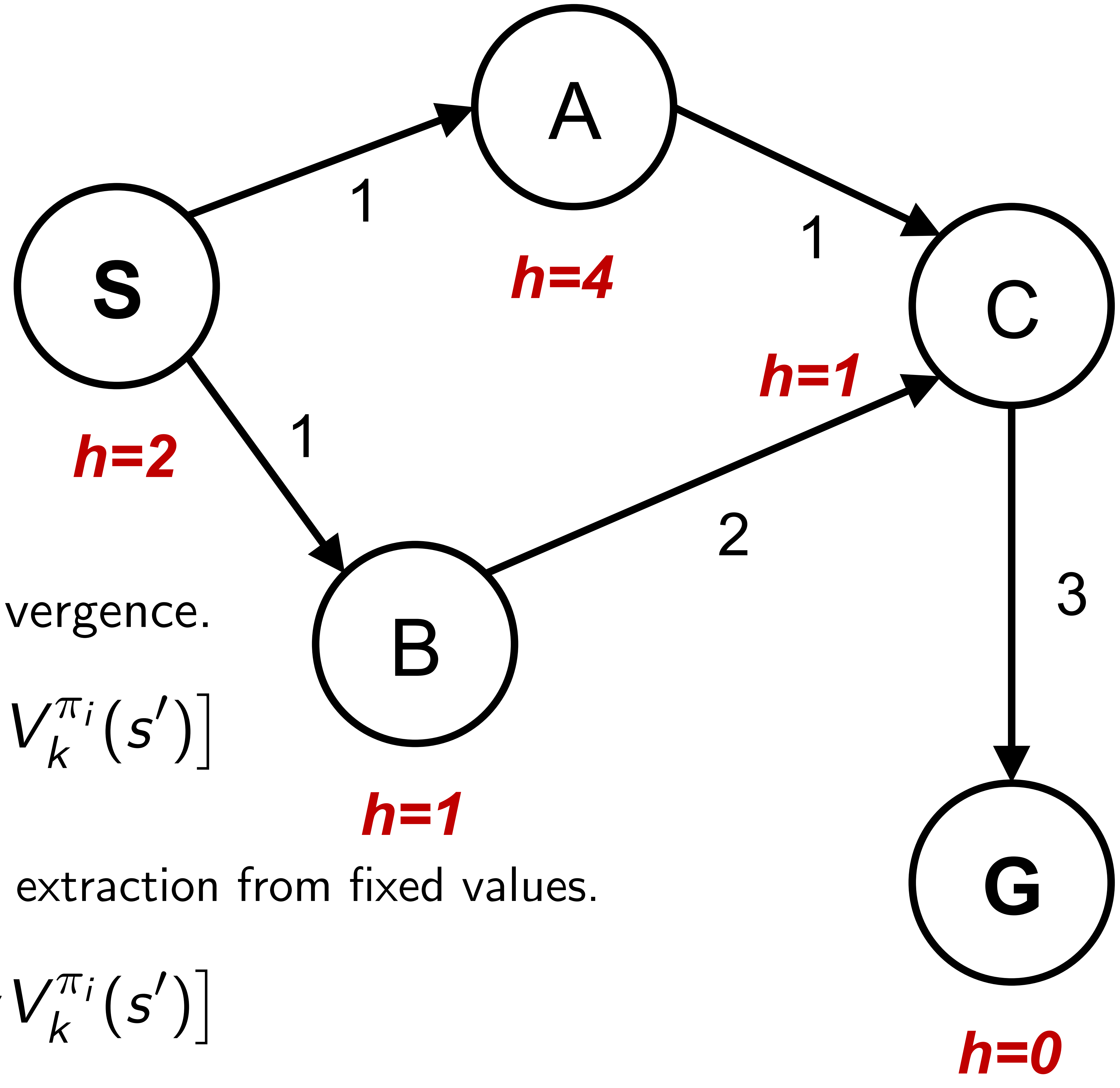
$$V_{k+1}^{\pi_i}(s) \leftarrow \sum_{s'} p(s' \mid s, \pi(s)) \left[r(s, \pi(s), s') + \gamma V_k^{\pi_i}(s')\right]$$

Policy improvement. Look-ahead and keep optimality. Policy extraction from fixed values.

$$\pi_{i+1}(s) = \arg\max_{a \in \mathcal{A}(s)} \sum_{s'} p(s' \mid s, a) \left[r(s, a, s') + \gamma V_k^{\pi_i}(s')\right]$$

Maximize sum of (expected) rewards, *Policy iteration*:

assume deterministic robot, no discounting



Policy π evaluation. Solve equations or iterate until convergence.

$$V_{k+1}^{\pi_i}(s) \leftarrow \sum_{s'} p(s' \mid s, \pi(s)) \left[r(s, \pi(s), s') + \gamma V_k^{\pi_i}(s') \right]$$

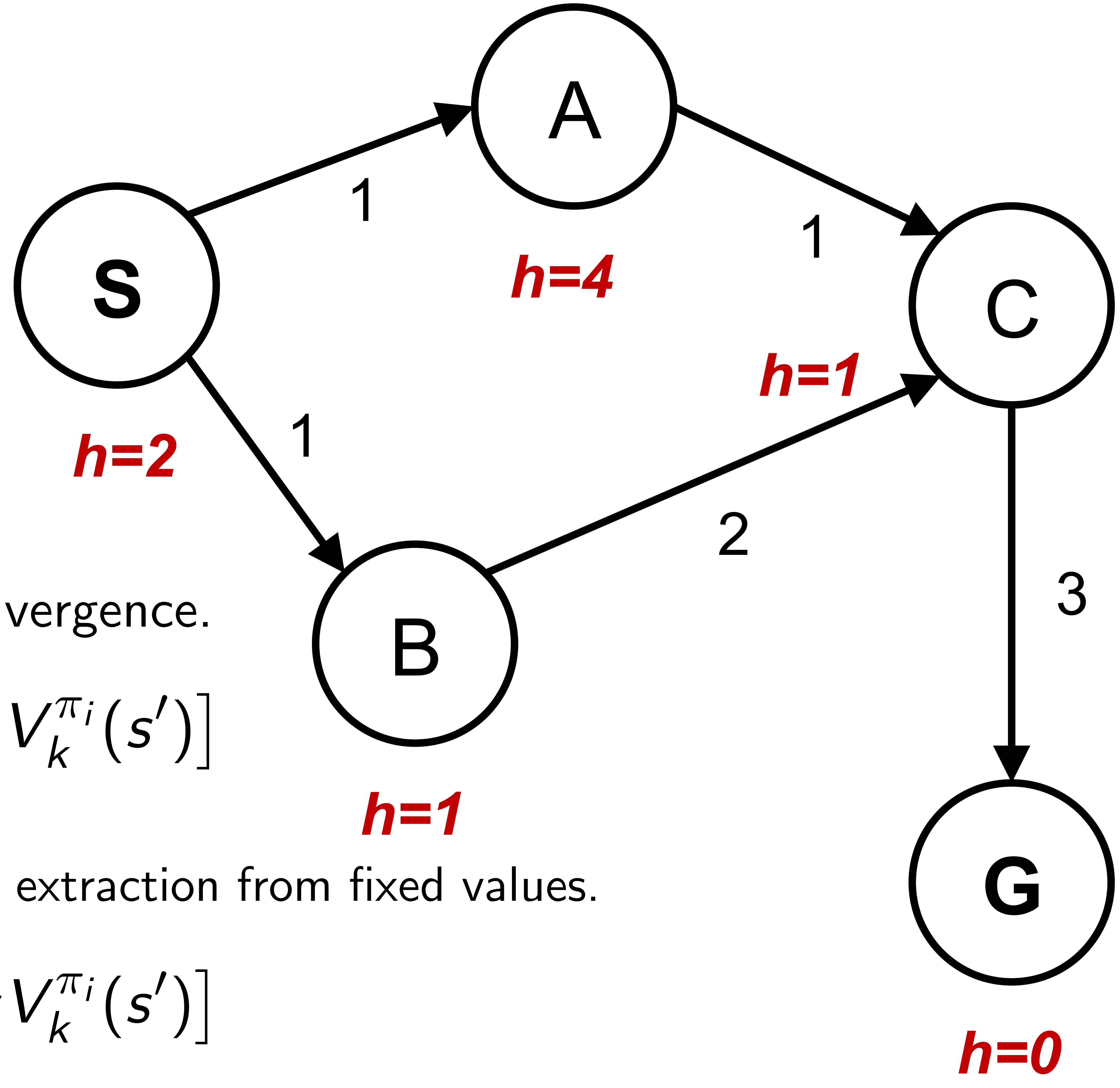
Policy improvement. Look-ahead and keep optimality. Policy extraction from fixed values.

$$\pi_{i+1}(s) = \arg\max_{a \in \mathcal{A}(s)} \sum_{s'} p(s' \mid s, a) \left[r(s, a, s') + \gamma V_k^{\pi_i}(s') \right]$$

Maximize sum of (expected) rewards, *Policy iteration*:

assume deterministic robot, no discounting

init: p([S,A,B,C,G]) = [right,go,go,go,exit]



Policy π evaluation. Solve equations or iterate until convergence.

$$V_{k+1}^{\pi_i}(s) \leftarrow \sum_{s'} p(s' \mid s, \pi(s)) \left[r(s, \pi(s), s') + \gamma V_k^{\pi_i}(s') \right]$$

Policy improvement. Look-ahead and keep optimality. Policy extraction from fixed values.

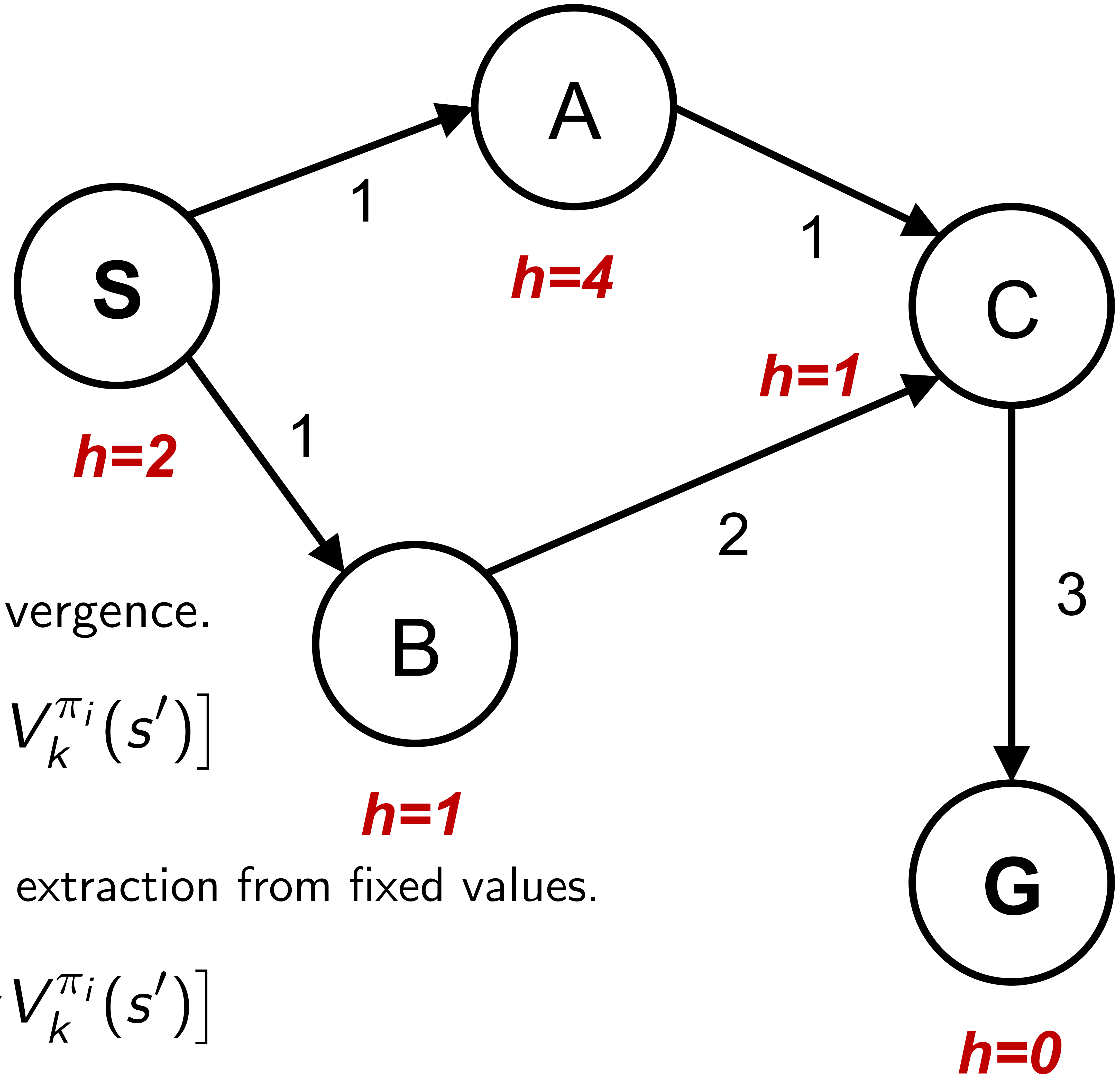
$$\pi_{i+1}(s) = \arg\max_{a \in \mathcal{A}(s)} \sum_{s'} p(s' \mid s, a) \left[r(s, a, s') + \gamma V_k^{\pi_i}(s') \right]$$

Maximize sum of (expected) rewards, *Policy iteration*:

assume deterministic robot, no discounting

init: p([S,A,B,C,G]) = [right,go,go,go,exit]

- policy eval => V([]) = [4,6,5,7,10]



Policy π evaluation. Solve equations or iterate until convergence.

$$V^{\pi_i}_{k+1}(s) \leftarrow \sum_{s'} p(s' \mid s, \pi(s)) \left[r(s, \pi(s), s') + \gamma V^{\pi_i}_k(s') \right]$$

Policy improvement. Look-ahead and keep optimality. Policy extraction from fixed values.

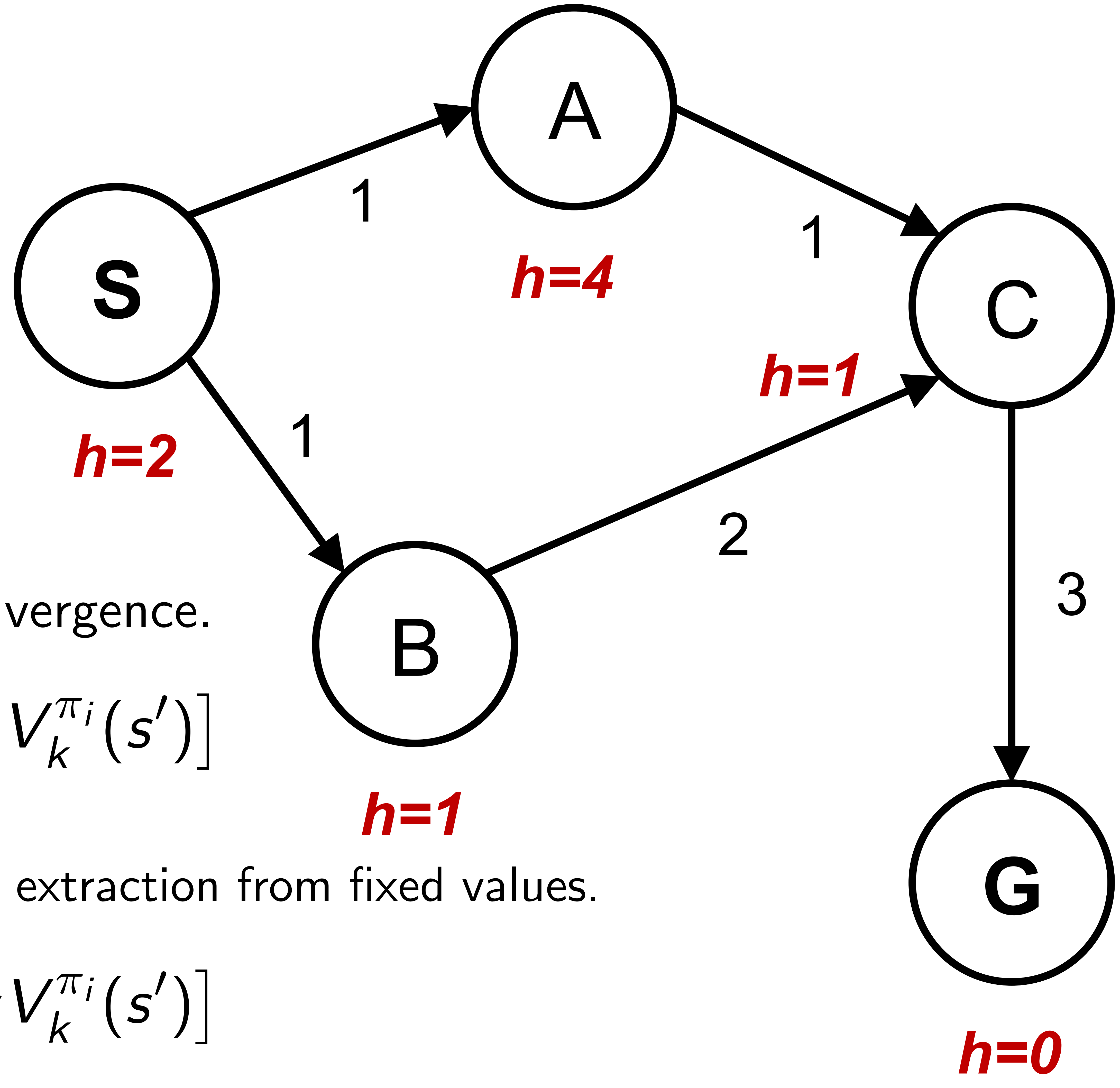
$$\pi_{i+1}(s) = \arg\max_{a \in \mathcal{A}(s)} \sum_{s'} p(s' \mid s, a) \left[r(s, a, s') + \gamma V^{\pi_i}_k(s') \right]$$

Maximize sum of (expected) rewards, *Policy iteration*:

assume deterministic robot, no discounting

init: p([S,A,B,C,G]) = [right,go,go,go,exit]

- policy eval => V([]) = [4,6,5,7,10]
- policy update p = [left,go,go,go,exit]



Policy π evaluation. Solve equations or iterate until convergence.

$$V_{k+1}^{\pi_i}(s) \leftarrow \sum_{s'} p(s' \mid s, \pi(s)) \left[r(s, \pi(s), s') + \gamma V_k^{\pi_i}(s') \right]$$

Policy improvement. Look-ahead and keep optimality. Policy extraction from fixed values.

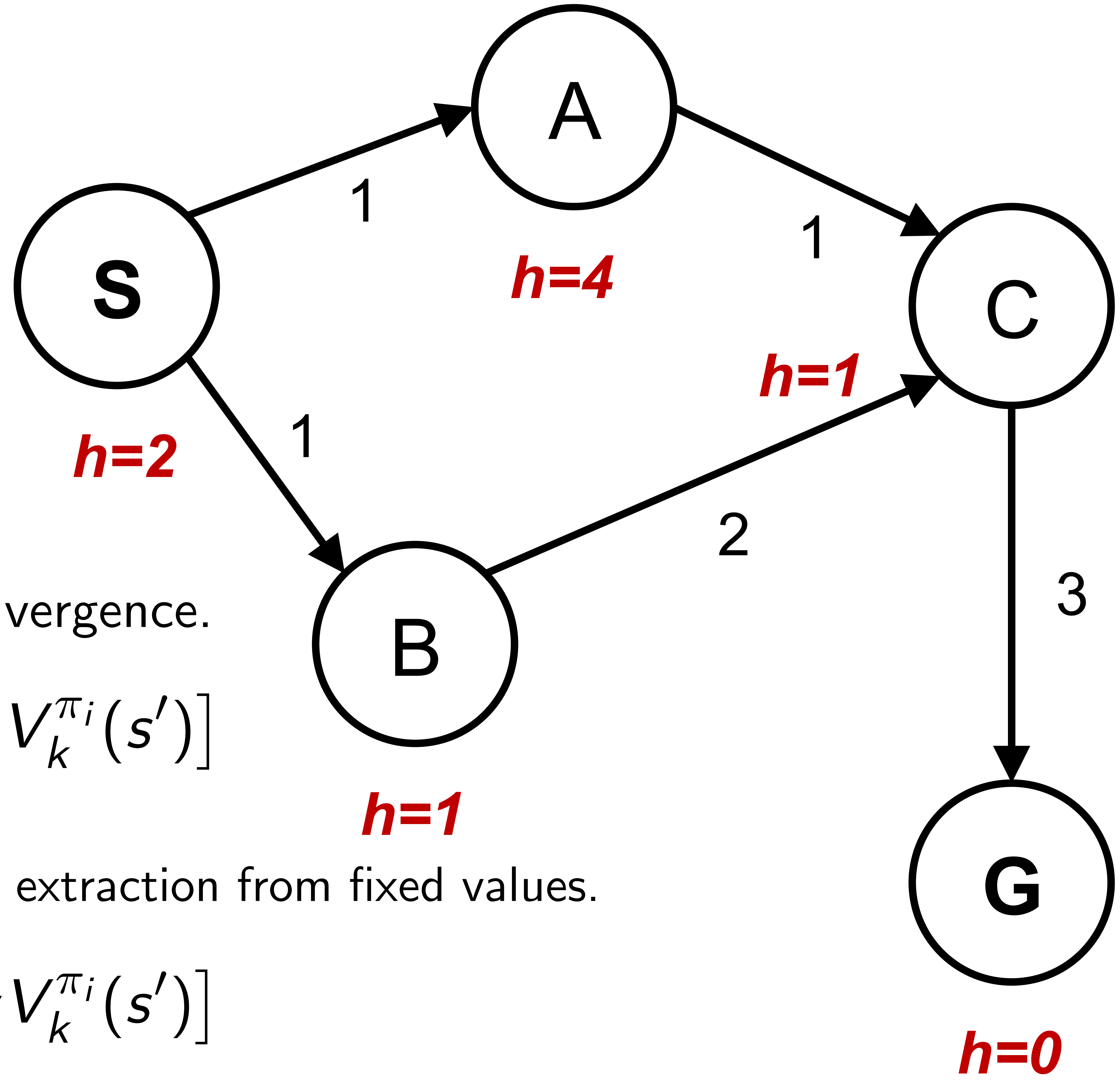
$$\pi_{i+1}(s) = \arg\max_{a \in \mathcal{A}(s)} \sum_{s'} p(s' \mid s, a) \left[r(s, a, s') + \gamma V_k^{\pi_i}(s') \right]$$

Maximize sum of (expected) rewards, *Policy iteration*:

assume deterministic robot, no discounting

init: p([S,A,B,C,G]) = [right,go,go,go,exit]

- policy eval => V([]) = [4,6,5,7,10]
- policy update p = [left,go,go,go,exit]
- eval V([]) = [5,6,5,7,10]



Policy π evaluation. Solve equations or iterate until convergence.

$$V_{k+1}^{\pi_i}(s) \leftarrow \sum_{s'} p(s' \mid s, \pi(s)) \left[r(s, \pi(s), s') + \gamma V_k^{\pi_i}(s') \right]$$

Policy improvement. Look-ahead and keep optimality. Policy extraction from fixed values.

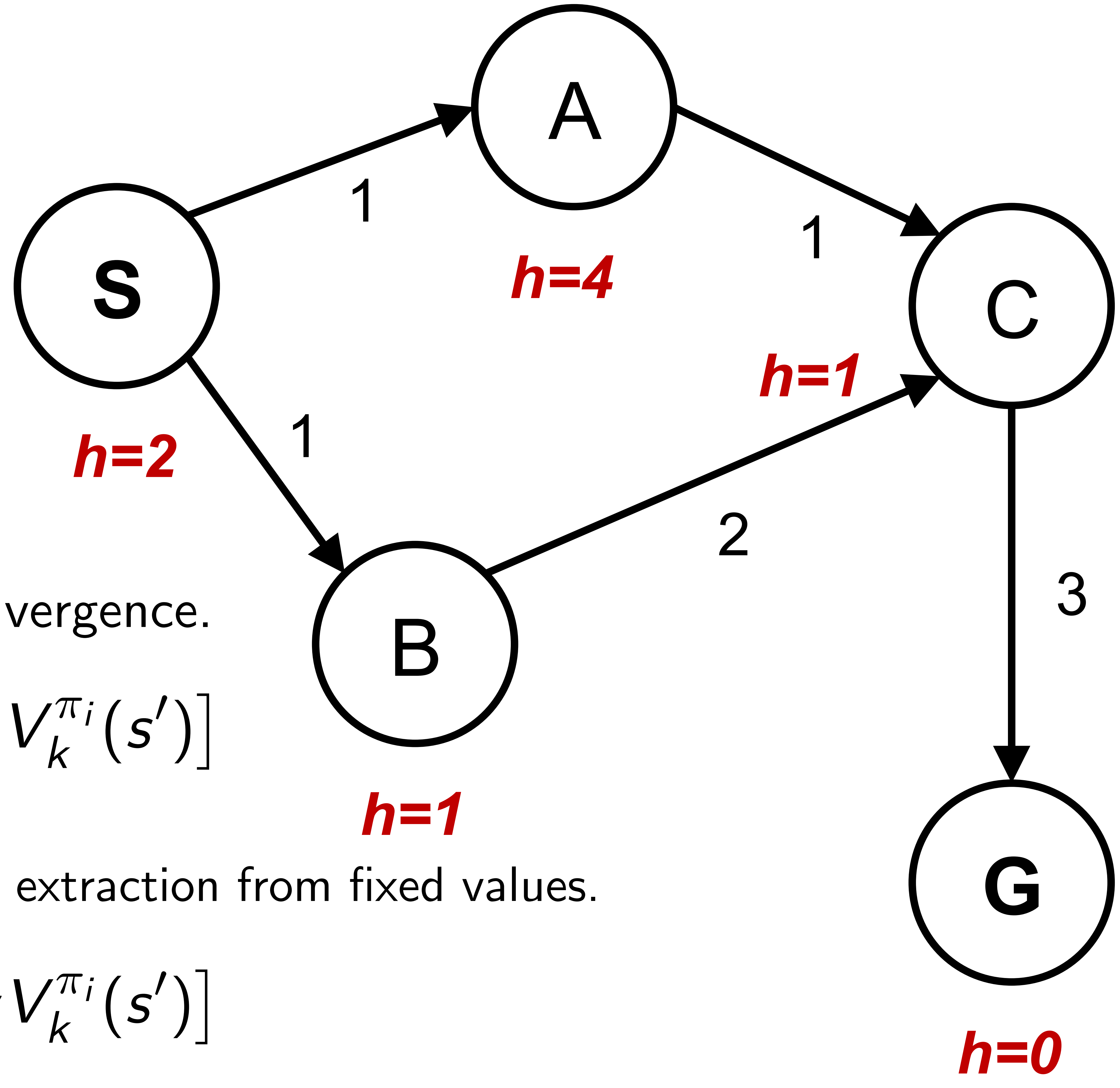
$$\pi_{i+1}(s) = \arg\max_{a \in \mathcal{A}(s)} \sum_{s'} p(s' \mid s, a) \left[r(s, a, s') + \gamma V_k^{\pi_i}(s') \right]$$

Maximize sum of (expected) rewards, *Policy iteration*:

assume deterministic robot, no discounting

init: p([S,A,B,C,G]) = [right,go,go,go,exit]

- policy eval => V([]) = [4,6,5,7,10]
- policy update p = [left,go,go,go,exit]
- eval V([]) = [5,6,5,7,10]
- update p = [left,go,go,go,exit]



Policy π evaluation. Solve equations or iterate until convergence.

$$V_{k+1}^{\pi_i}(s) \leftarrow \sum_{s'} p(s' \mid s, \pi(s)) \left[r(s, \pi(s), s') + \gamma V_k^{\pi_i}(s') \right]$$

Policy improvement. Look-ahead and keep optimality. Policy extraction from fixed values.

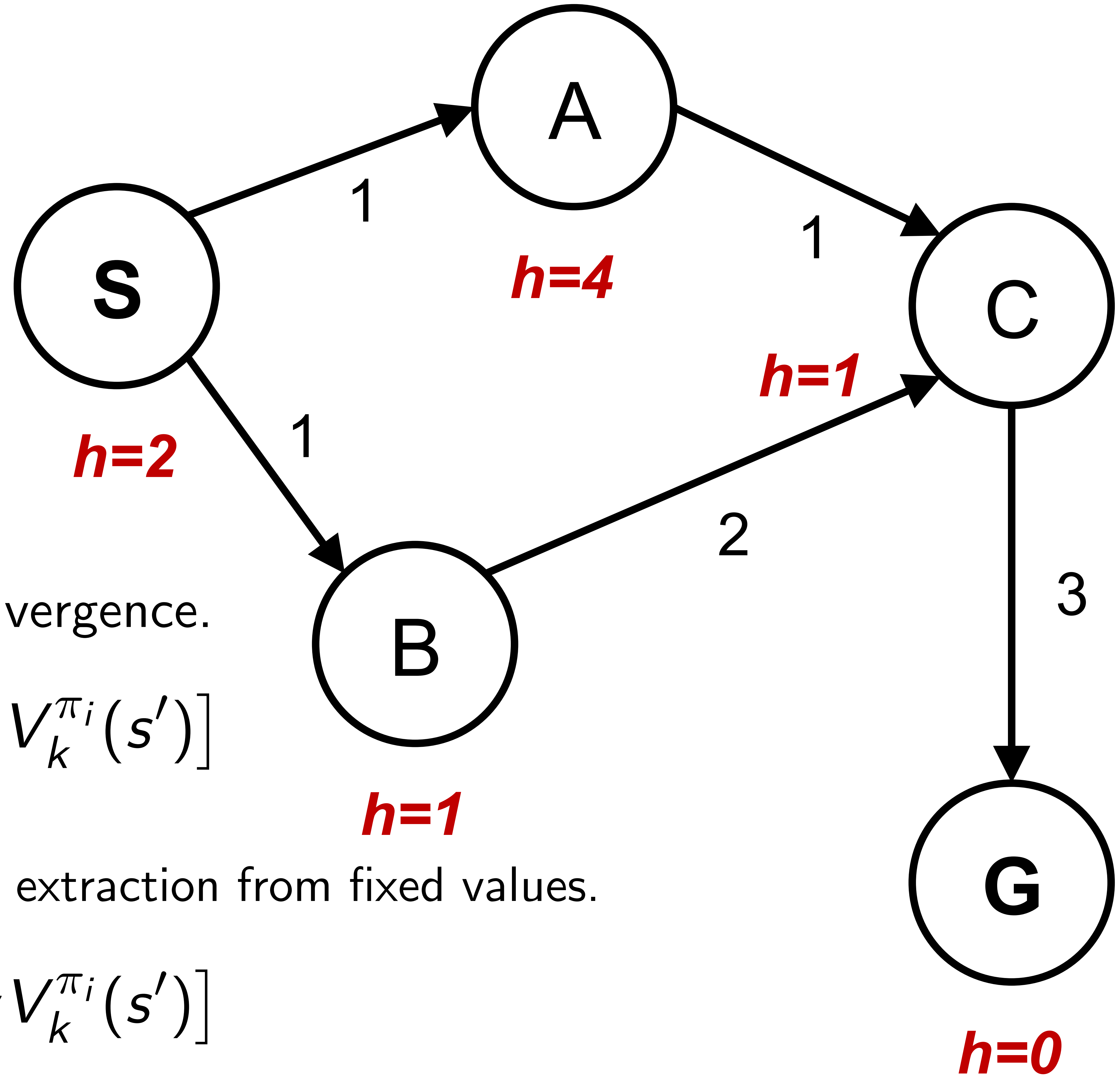
$$\pi_{i+1}(s) = \arg\max_{a \in \mathcal{A}(s)} \sum_{s'} p(s' \mid s, a) \left[r(s, a, s') + \gamma V_k^{\pi_i}(s') \right]$$

Maximize sum of (expected) rewards, *Policy iteration*:

assume deterministic robot, no discounting

init: p([S,A,B,C,G]) = [right,go,go,go,exit]

- policy eval => V([]) = [4,6,5,7,10]
- policy update p = [left,go,go,go,exit]
- eval V([]) = [5,6,5,7,10]
- update p = [left,go,go,go,exit]
- no change, stops

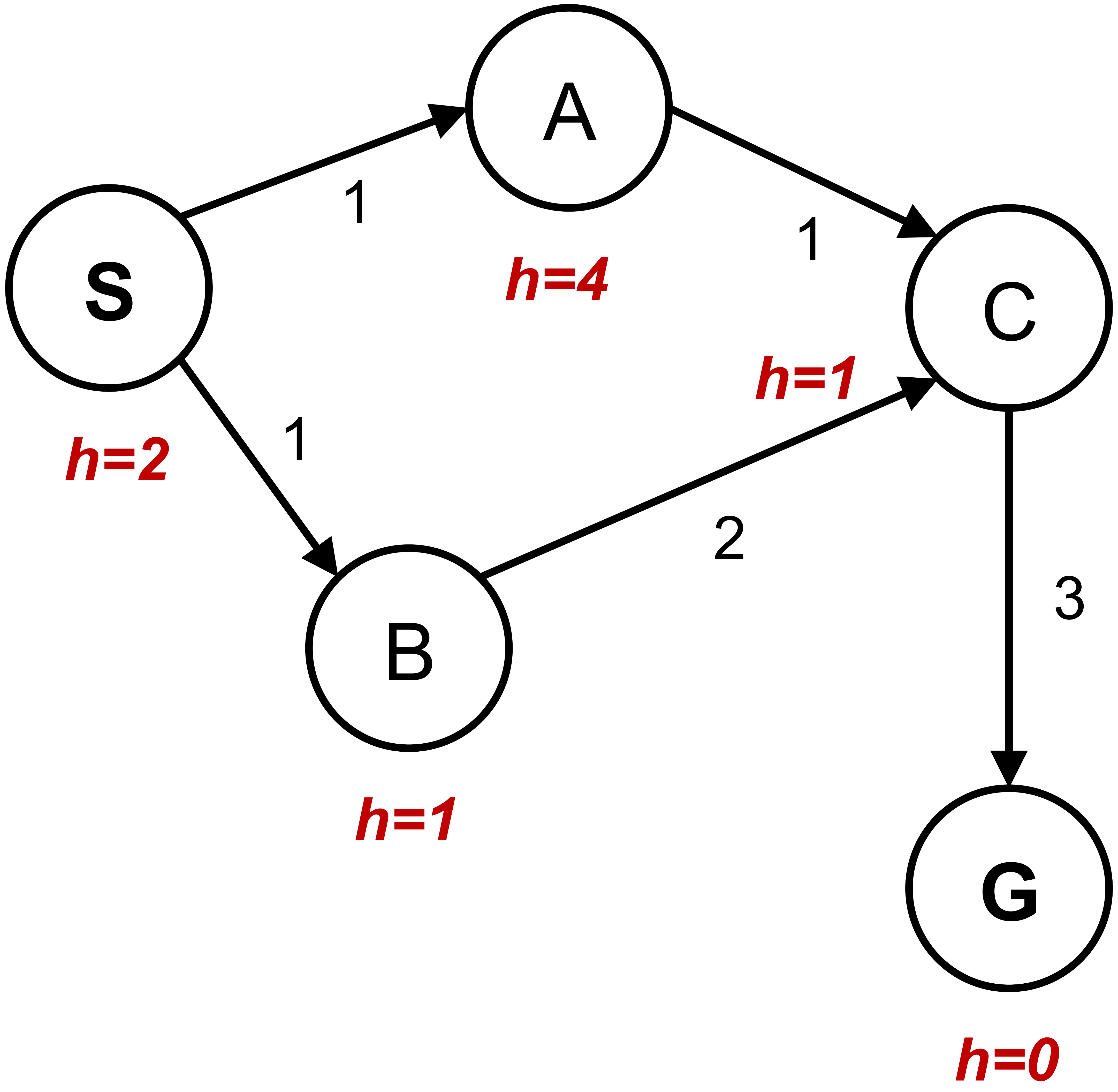


Policy π evaluation. Solve equations or iterate until convergence.

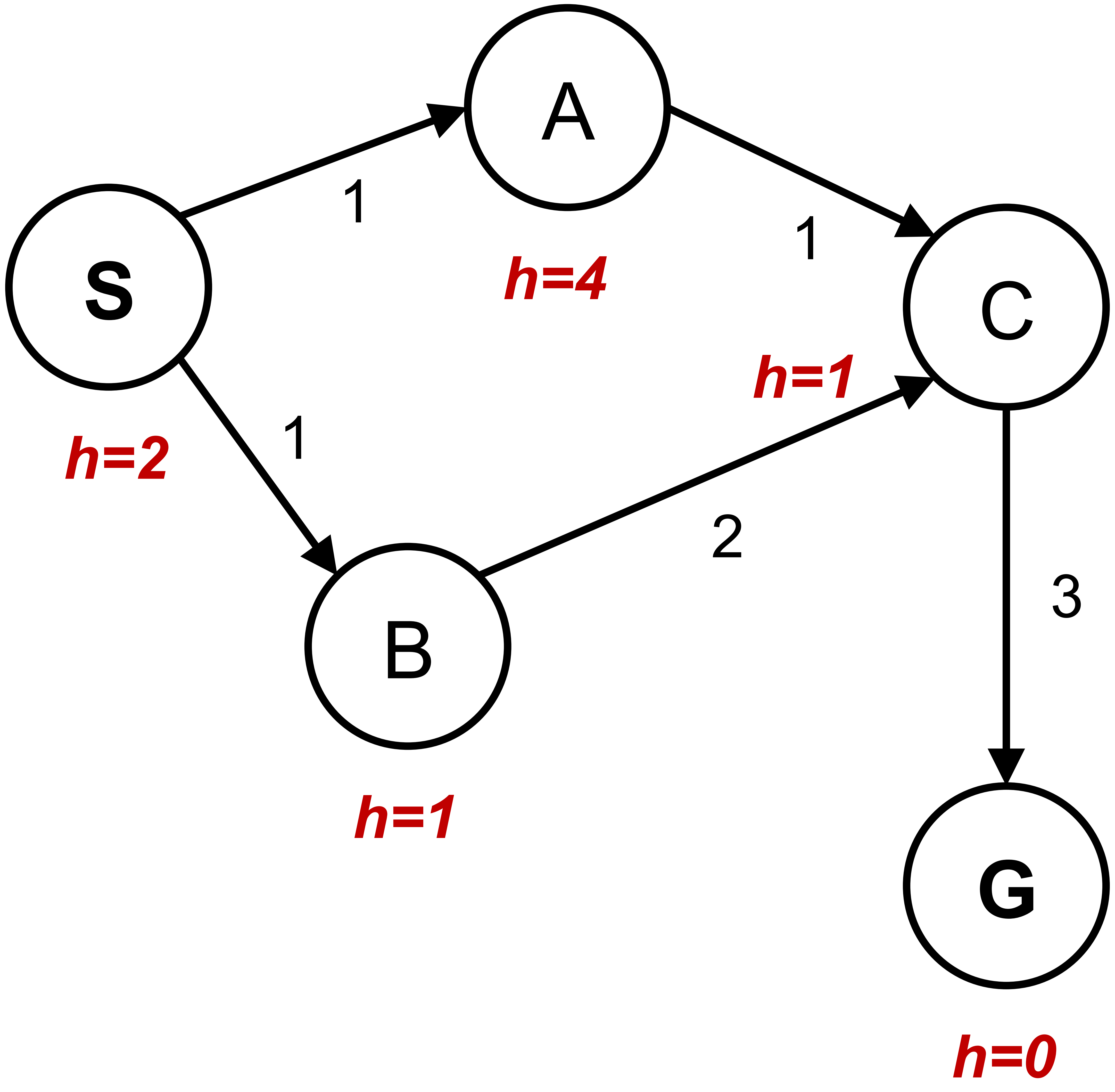
$$V_{k+1}^{\pi_i}(s) \leftarrow \sum_{s'} p(s' \mid s, \pi(s)) \left[r(s, \pi(s), s') + \gamma V_k^{\pi_i}(s') \right]$$

Policy improvement. Look-ahead and keep optimality. Policy extraction from fixed values.

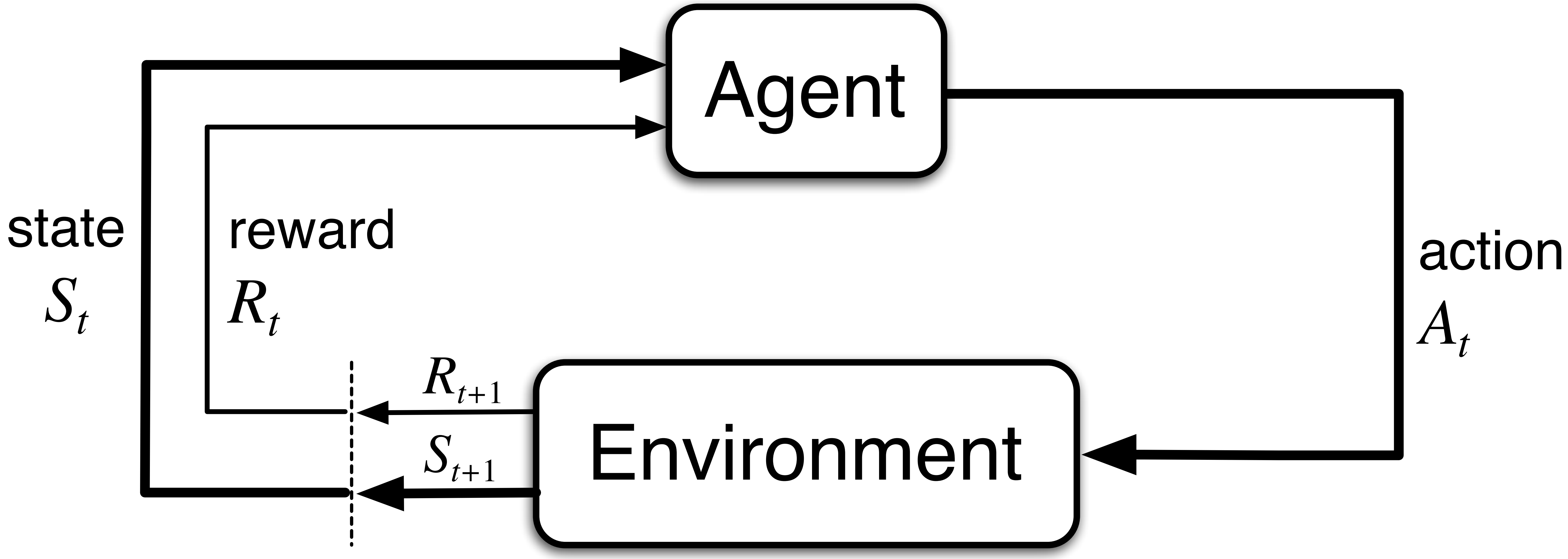
$$\pi_{i+1}(s) = \arg\max_{a \in \mathcal{A}(s)} \sum_{s'} p(s' \mid s, a) \left[r(s, a, s') + \gamma V_k^{\pi_i}(s') \right]$$

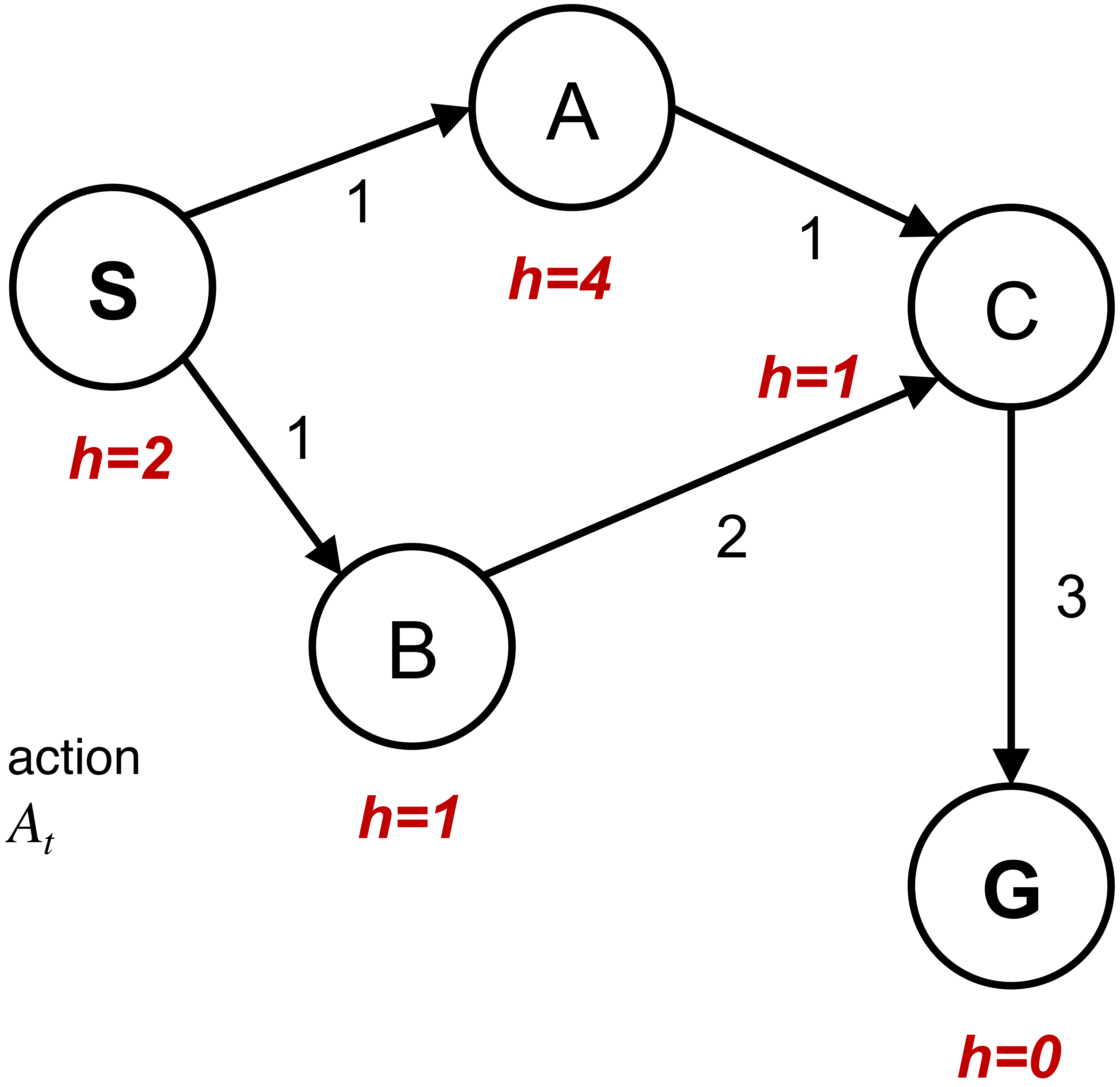

A
S
1
1
h=4
C
h=1
h=2
1
2
3
B
h=1
G
h=0

Let robot/agent walk at random and learn from experience (episodes):



Let robot/agent walk at random and learn from experience (episodes):





Let robot/agent walk at random and learn from experience (episodes):

Let robot/agent walk at random and learn from experience (episodes):

S,left,-1,A, go,-1,C, go,-3,G, exit,10 => Return = 6

Let robot/agent walk at random and learn from experience (episodes):

S,left,-1,A, go,-1,C, go,-3,G, exit,10 => Return = 6

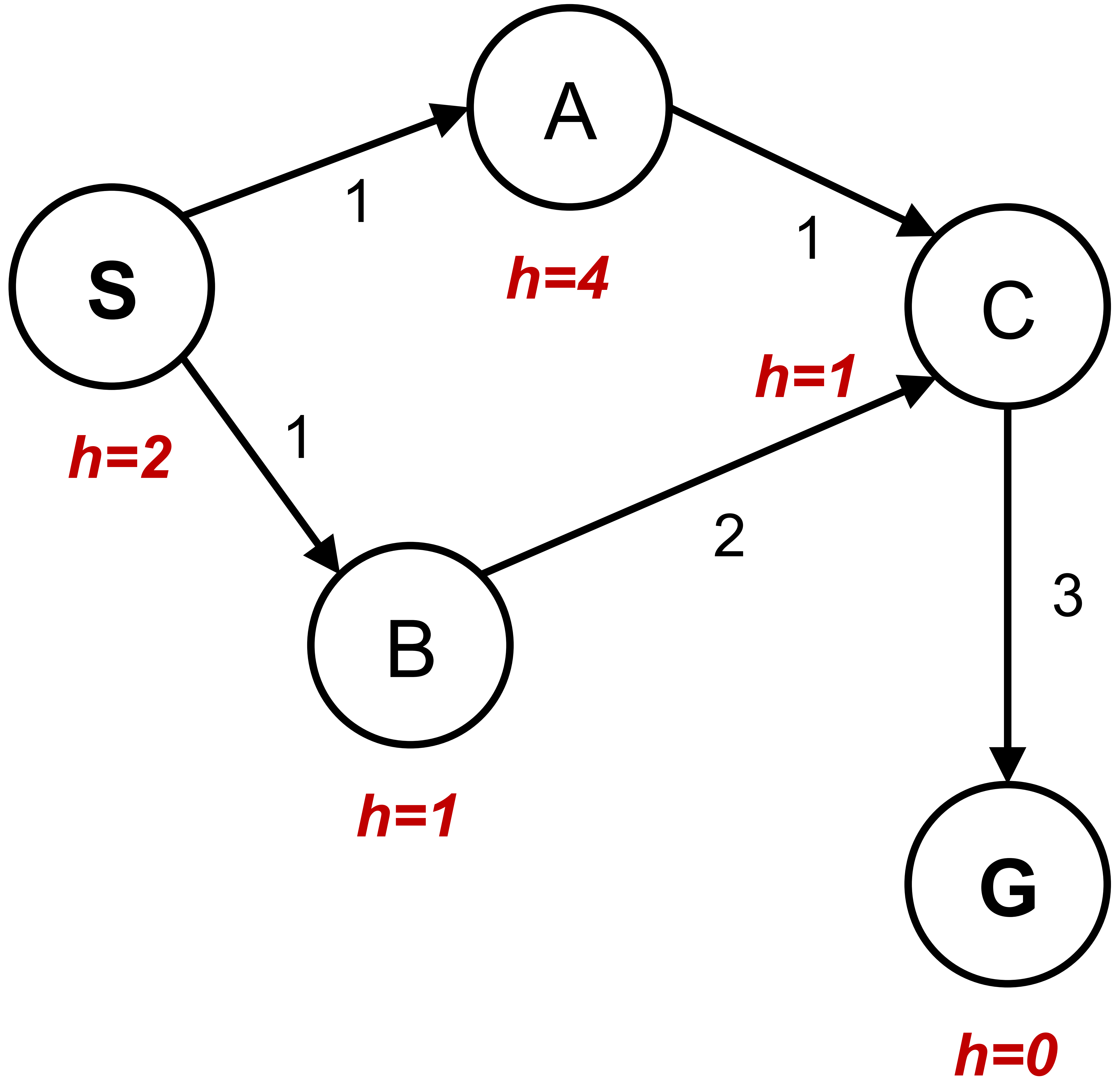
S,right,-1,B, go,-2,C, go,-3,G, exit,10 => Return = 5

Let robot/agent walk at random and learn from experience (episodes):

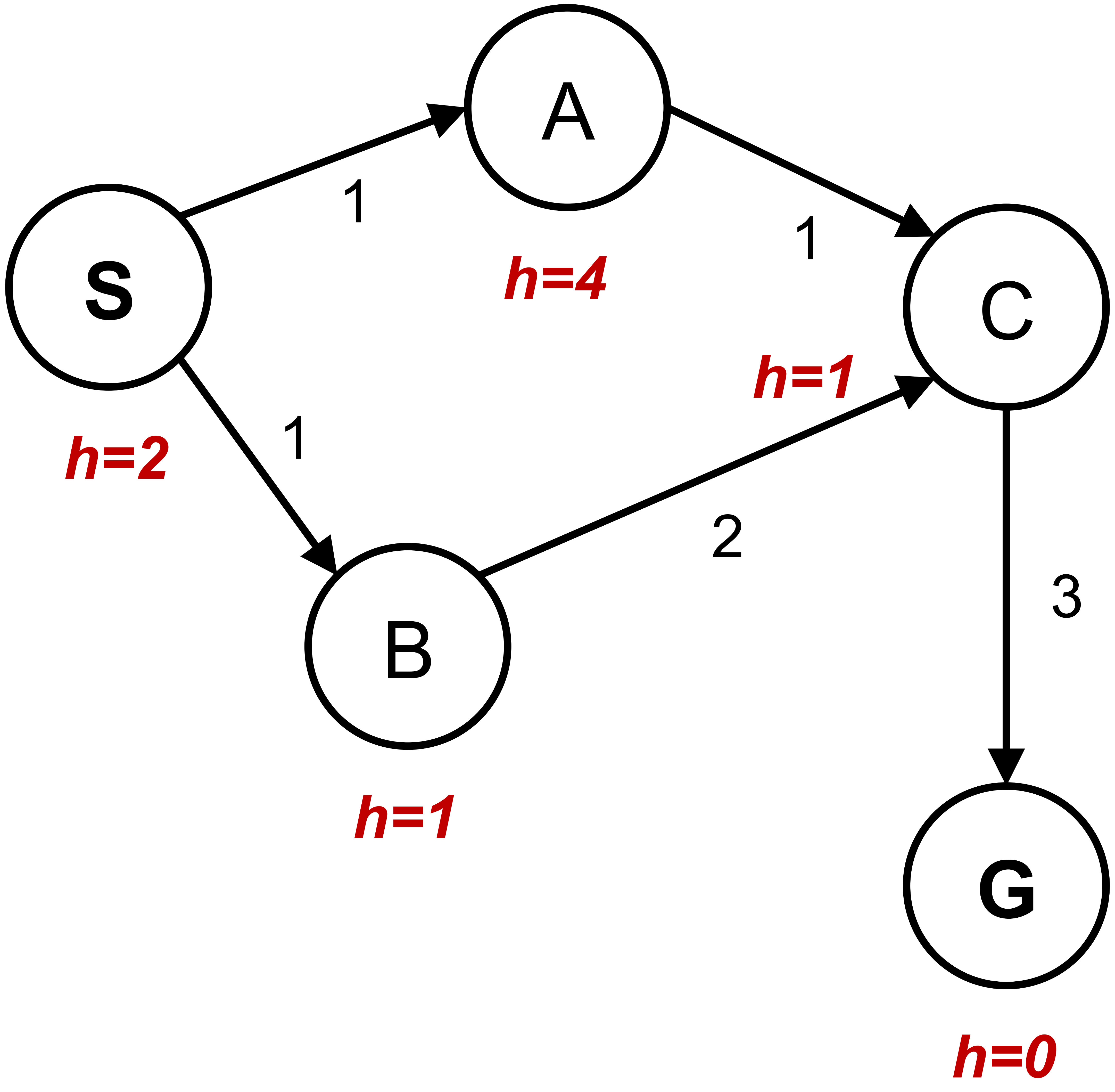
S,left,-1,A, go,-1,C, go,-3,G, exit,10 => Return = 6

S,right,-1,B, go,-2,C, go,-3,G, exit,10 => Return = 5

…

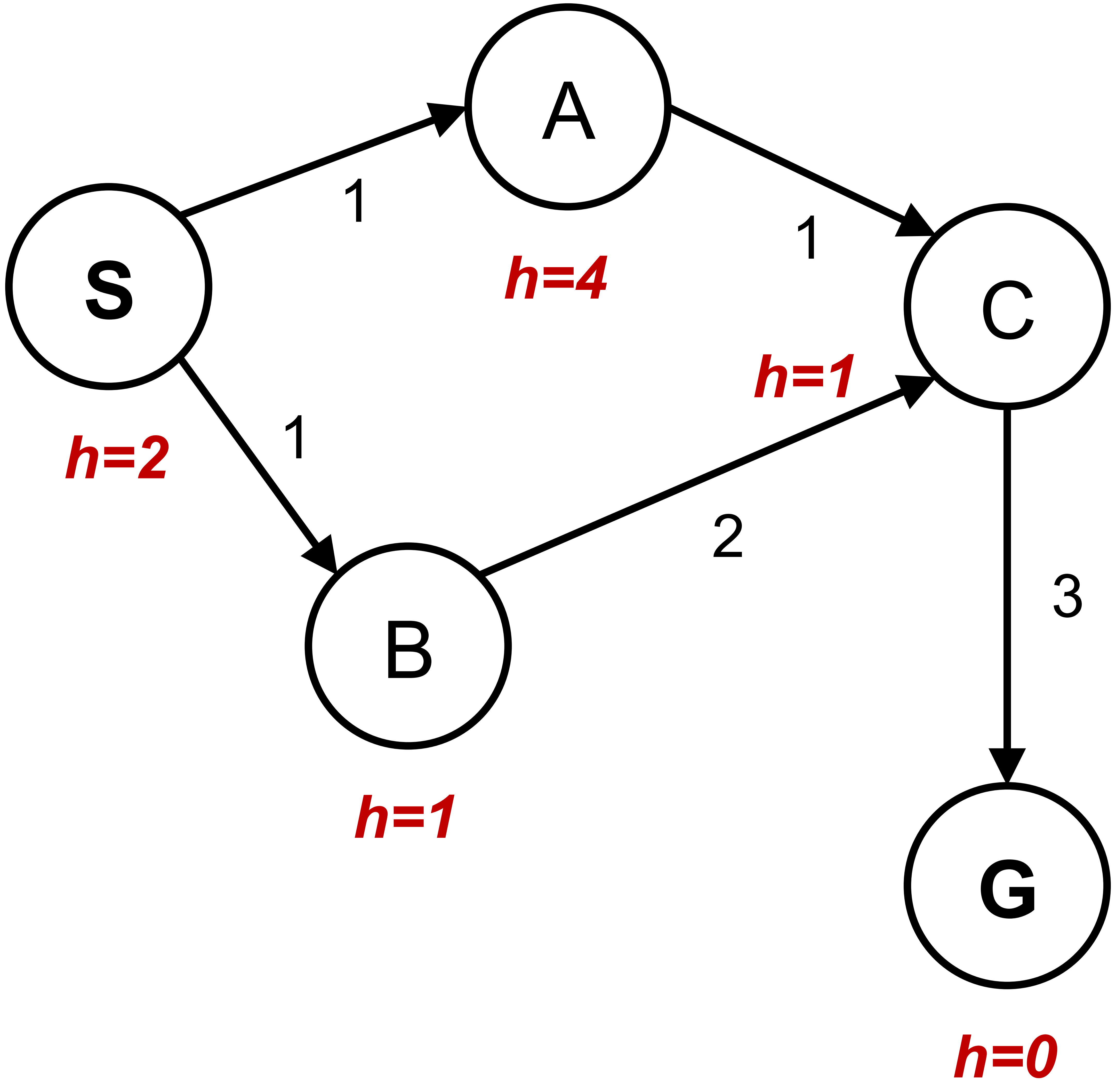

S
h=2
A
h=4
B
h=1
C
h=1
G
h=0
1
1
1
2
3

Let robot/agent walk at random and learn from experience (episodes)



Let robot/agent walk at random and learn from experience (episodes)

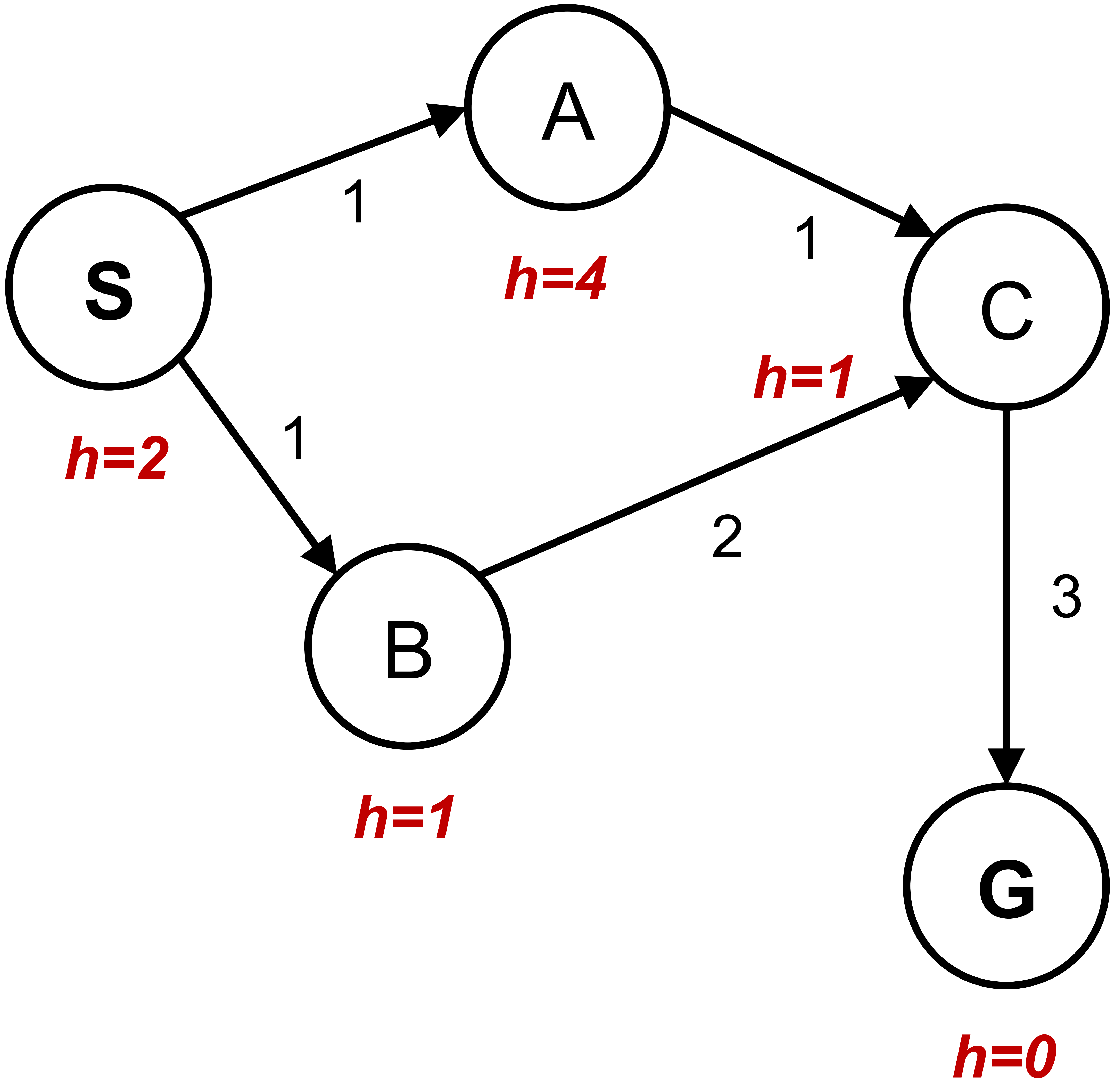
Direct evaluation, init all Q(state, action) = 0



Let robot/agent walk at random and learn from experience (episodes)

Direct evaluation, init all Q(state, action) = 0

S,left,-1,A, go,-1,C, go,-3,G, exit,10

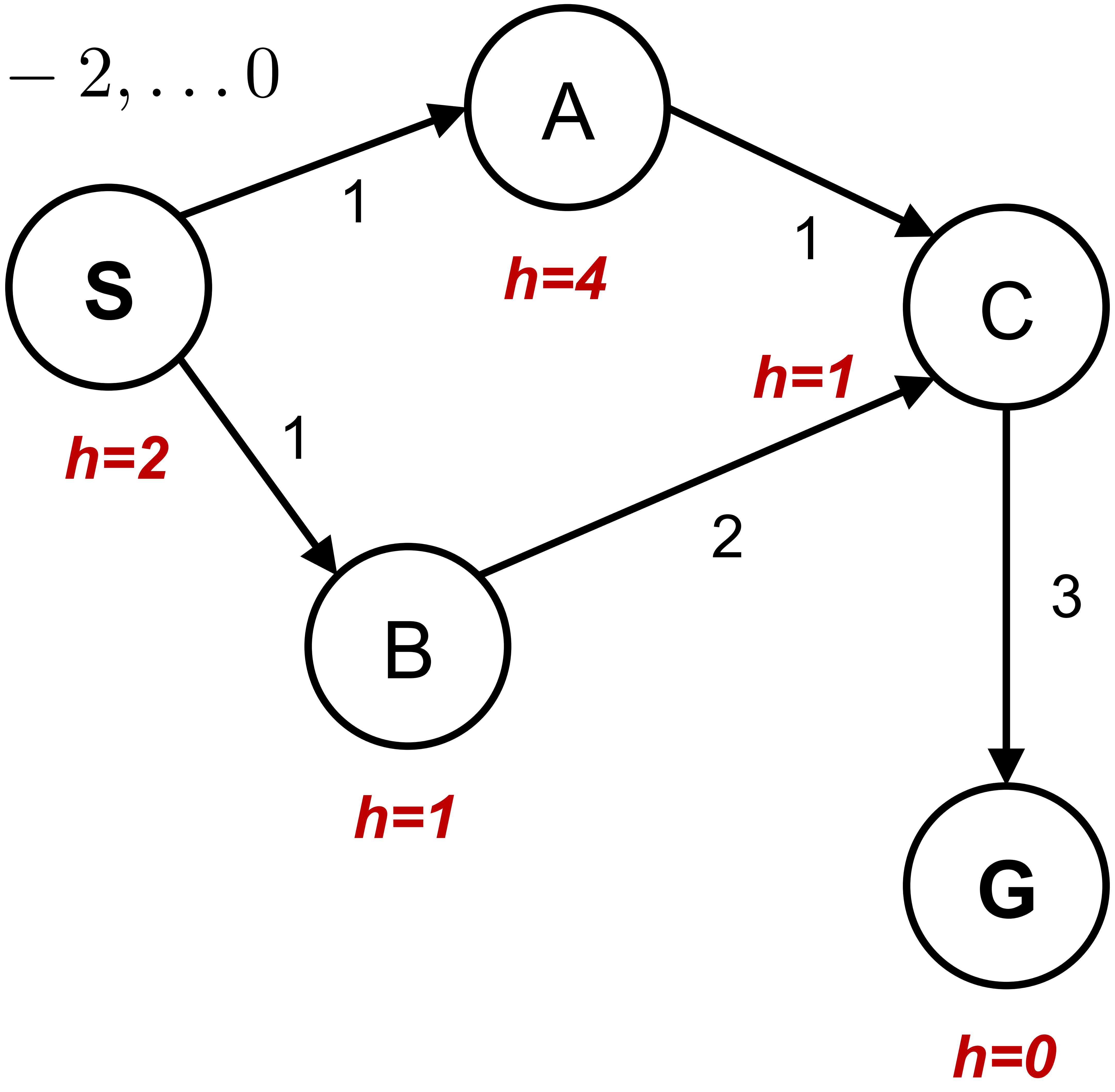


Let robot/agent walk at random and learn from experience (episodes)

Direct evaluation, init all Q(state, action) = 0

S,left,-1,A, go,-1,C, go,-3,G, exit,10

$G \leftarrow 0$ and loop backwards, $t = T-1, T-2, \ldots 0$
$G \leftarrow R_{t+1} + \gamma G$
Append G to Returns$(Q(S_t, A_t))$
$Q(S_t, A_t) \leftarrow$ average[Returns$(Q(S_t, A_t))$]

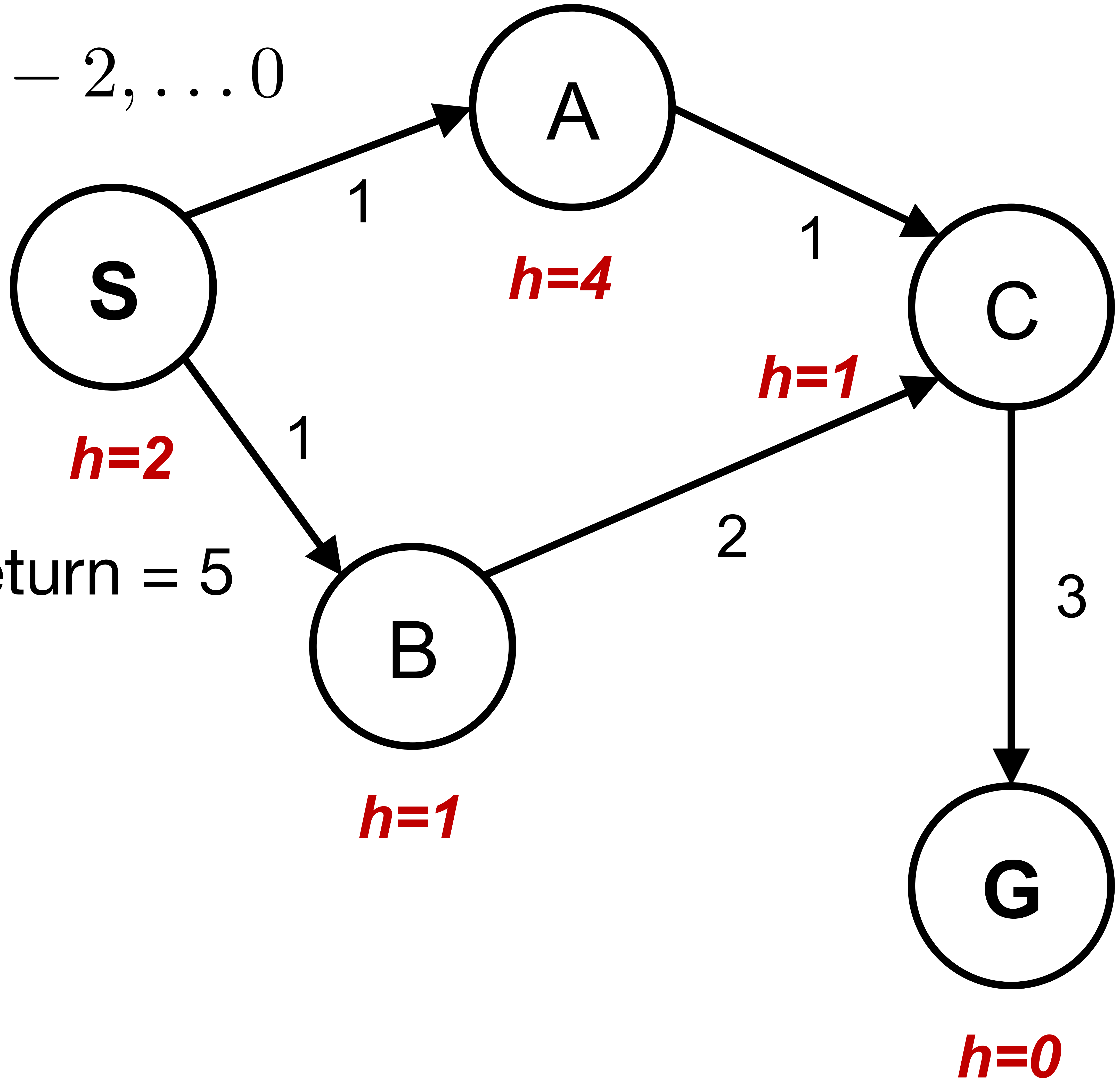


Let robot/agent walk at random and learn from experience (episodes)

Direct evaluation, init all Q(state, action) = 0

S,left,-1,A, go,-1,C, go,-3,G, exit,10

$G \leftarrow 0$ and loop backwards, $t = T-1, T-2, \ldots 0$
$G \leftarrow R_{t+1} + \gamma G$
Append G to Returns$(Q(S_t, A_t))$
$Q(S_t, A_t) \leftarrow$ average[Returns$(Q(S_t, A_t))$]



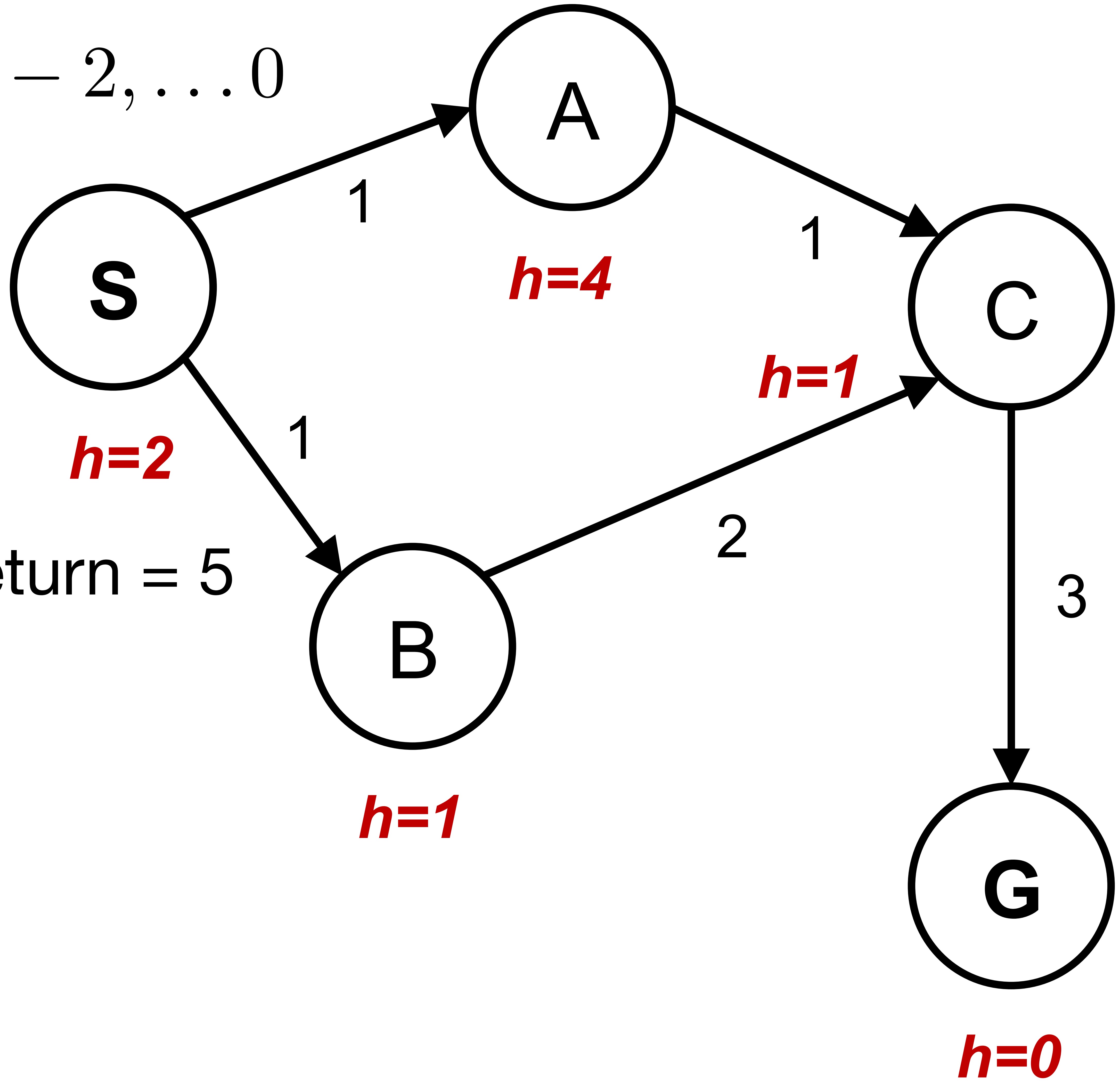
S,right,-1,B, go,-2,C, go,-3,G, exit,10 => Return = 5

Let robot/agent walk at random and learn from experience (episodes)

Direct evaluation, init all Q(state, action) = 0

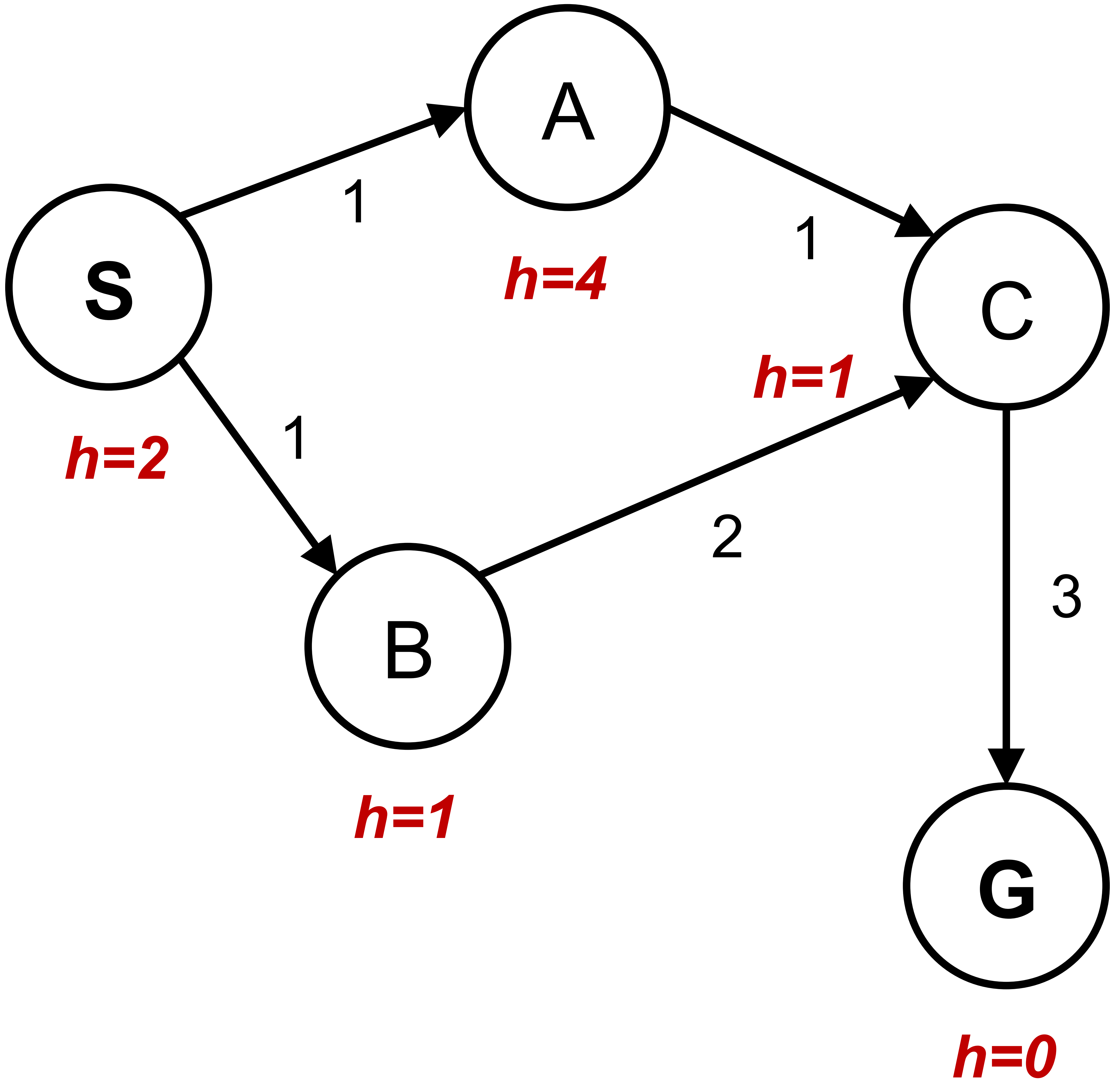
S,left,-1,A, go,-1,C, go,-3,G, exit,10

$G \leftarrow 0$ and loop backwards, $t = T-1, T-2, \ldots 0$
$G \leftarrow R_{t+1} + \gamma G$
Append G to Returns$(Q(S_t, A_t))$
$Q(S_t, A_t) \leftarrow$ average[Returns$(Q(S_t, A_t))$]

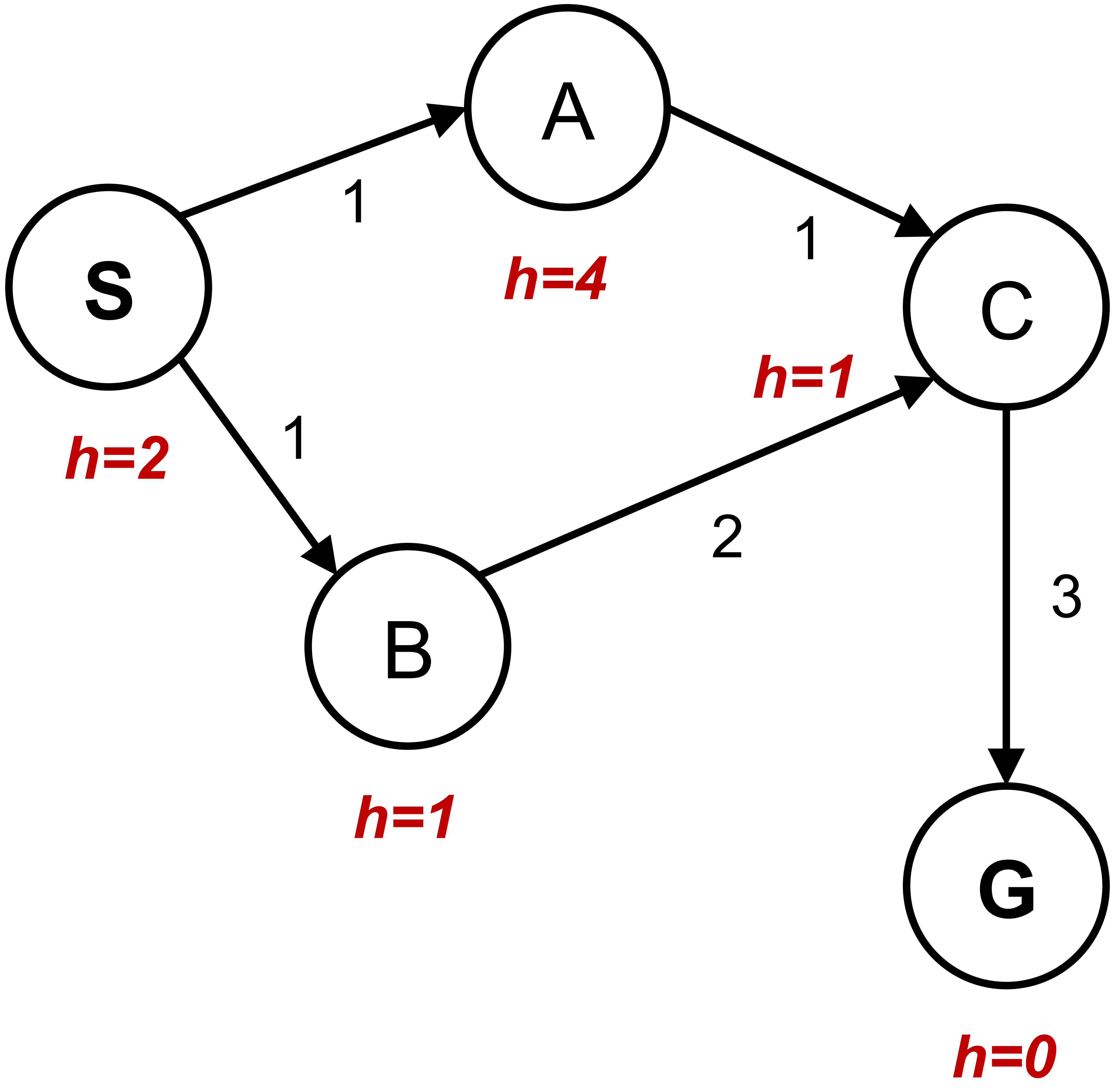


S,right,-1,B, go,-2,C, go,-3,G, exit,10 => Return = 5

…

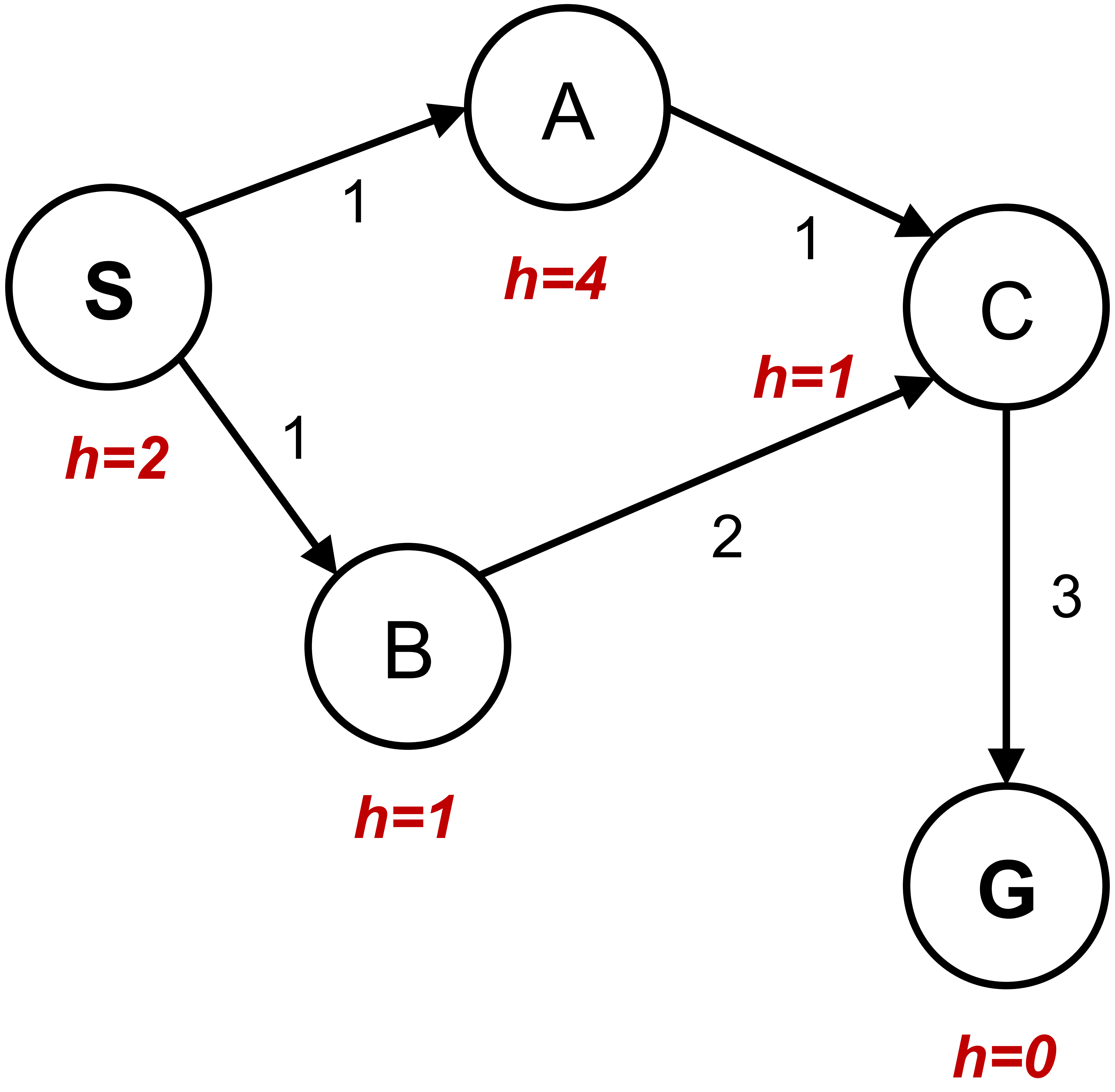

S
A
C
B
G
1
1
1
2
3
h=4
h=2
h=1
h=1
h=0

Let robot/agent walk at random and learn from experience (episodes)



Let robot/agent walk at random and learn from experience (episodes)

Learn from every visit - Temporal differences, init all Q(state, action) = 0



Let robot/agent walk at random and learn from experience (episodes)

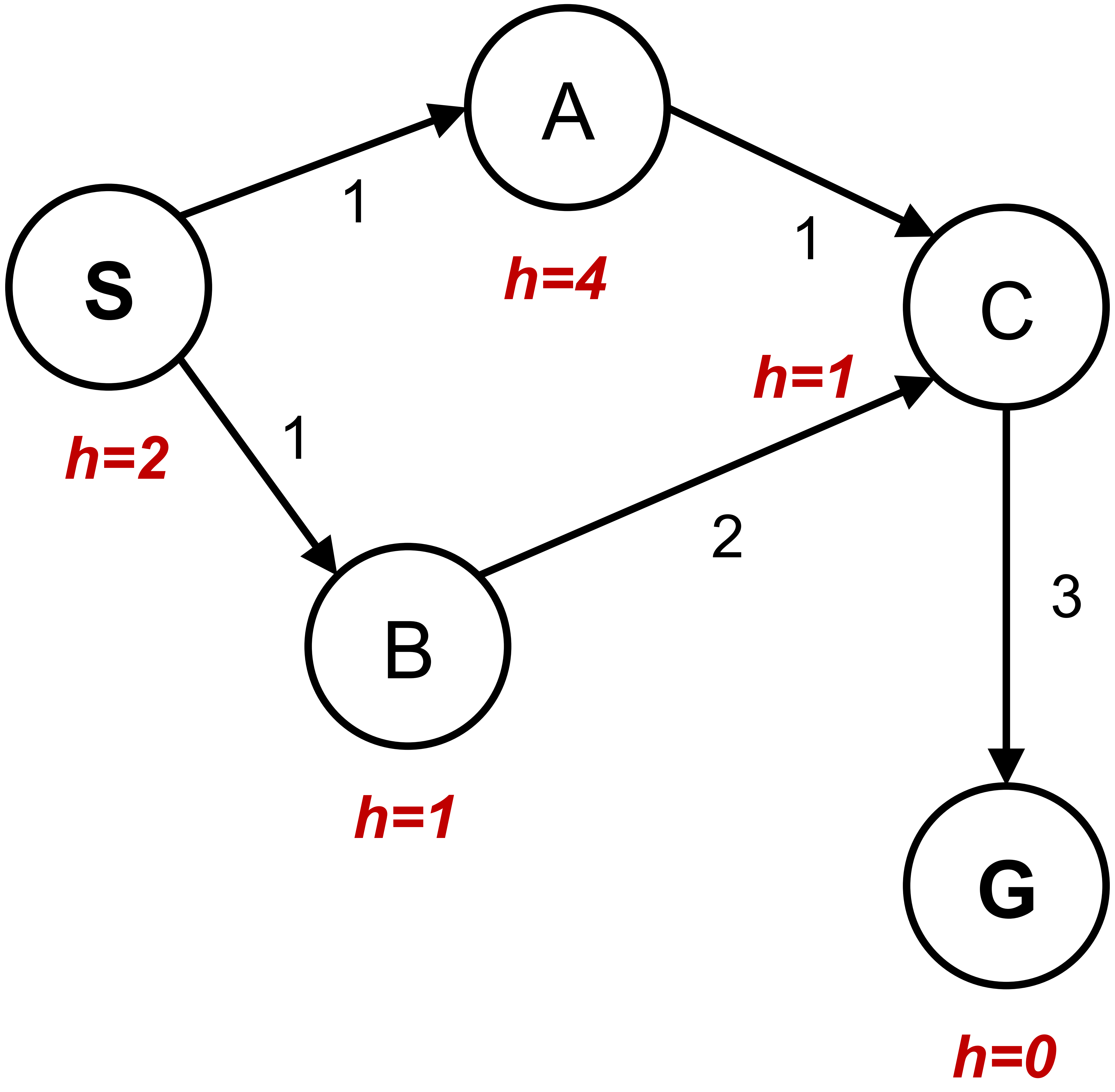
Learn from every visit - Temporal differences, init all Q(state, action) = 0

A new trial/sample estimate at time t

$$\text{trial} = R_{t+1} + \gamma \max_a Q(S_{t+1}, a)$$

α update

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(\text{trial} - Q(S_t, A_t))$$



Let robot/agent walk at random and learn from experience (episodes)

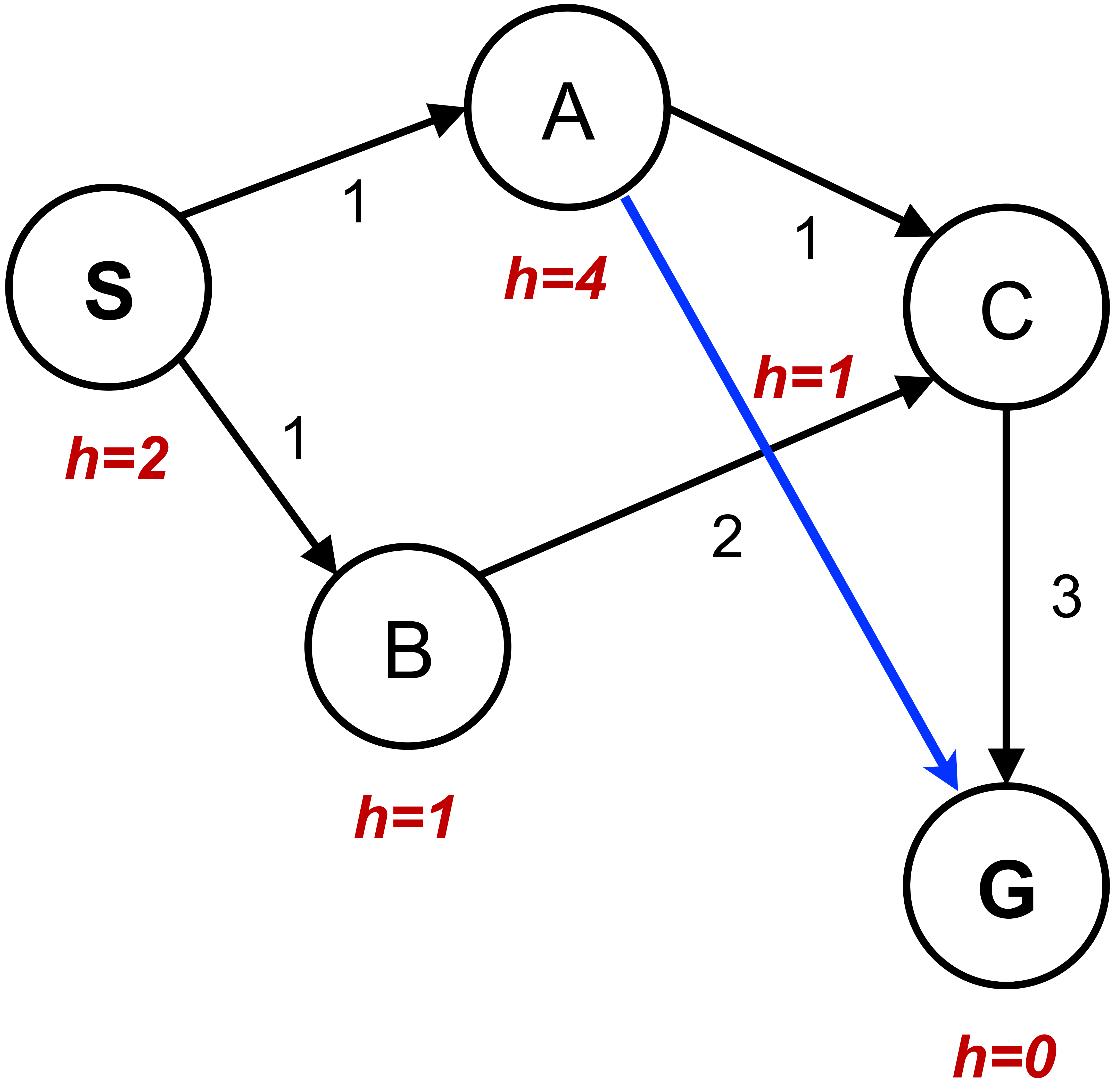
Learn from every visit - Temporal differences, init all Q(state, action) = 0

A new trial/sample estimate at time t

$$\text{trial} = R_{t+1} + \gamma \max_a Q(S_{t+1}, a)$$

α update

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(\text{trial} - Q(S_t, A_t))$$



Let robot/agent walk at random and learn from experience (episodes)

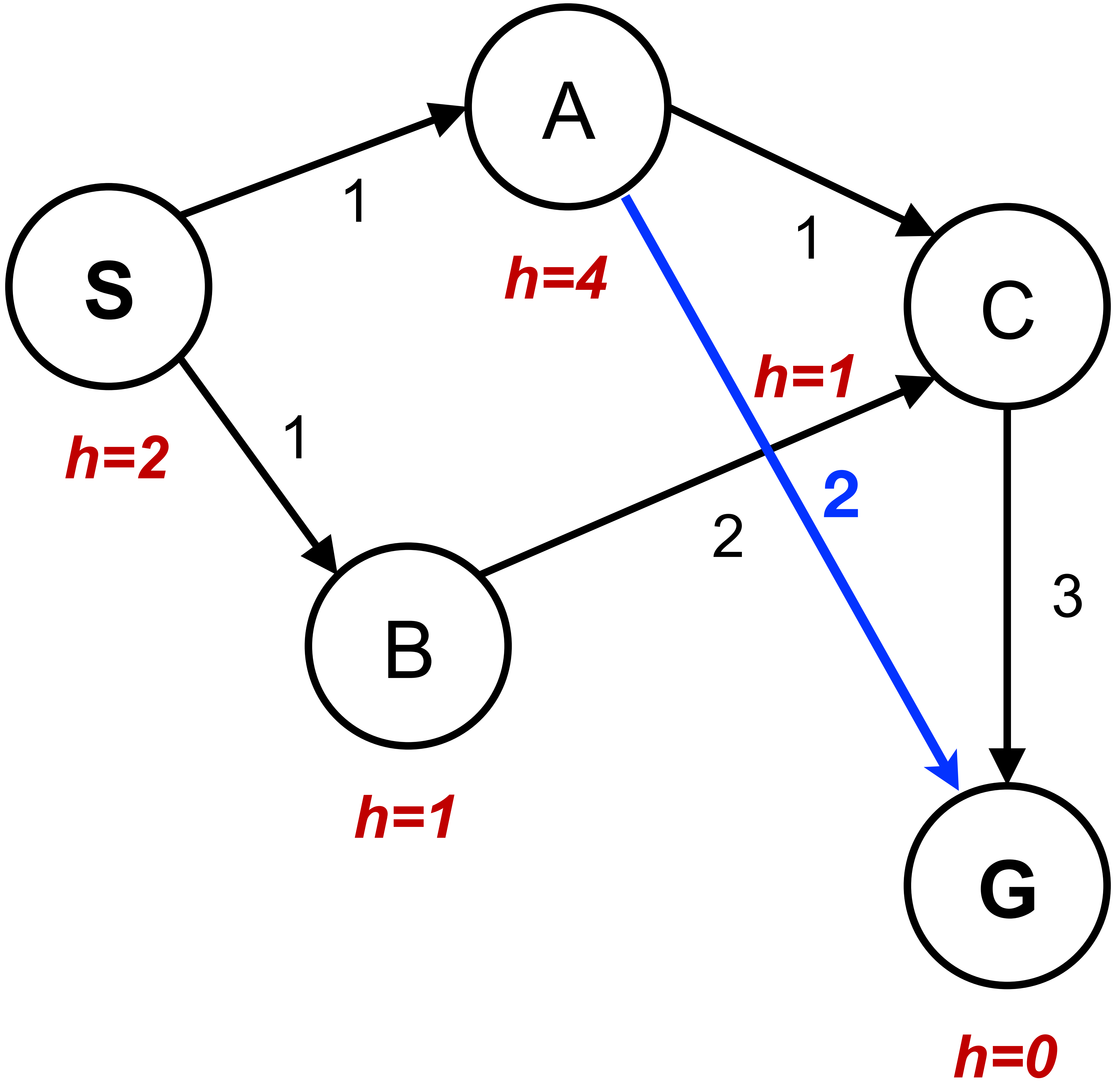
Learn from every visit - Temporal differences, init all Q(state, action) = 0

A new trial/sample estimate at time t

$$\text{trial} = R_{t+1} + \gamma \max_a Q(S_{t+1}, a)$$

α update

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(\text{trial} - Q(S_t, A_t))$$



Let robot/agent walk at random and learn from experience (episodes)

Learn from every visit - Temporal differences, init all Q(state, action) = 0

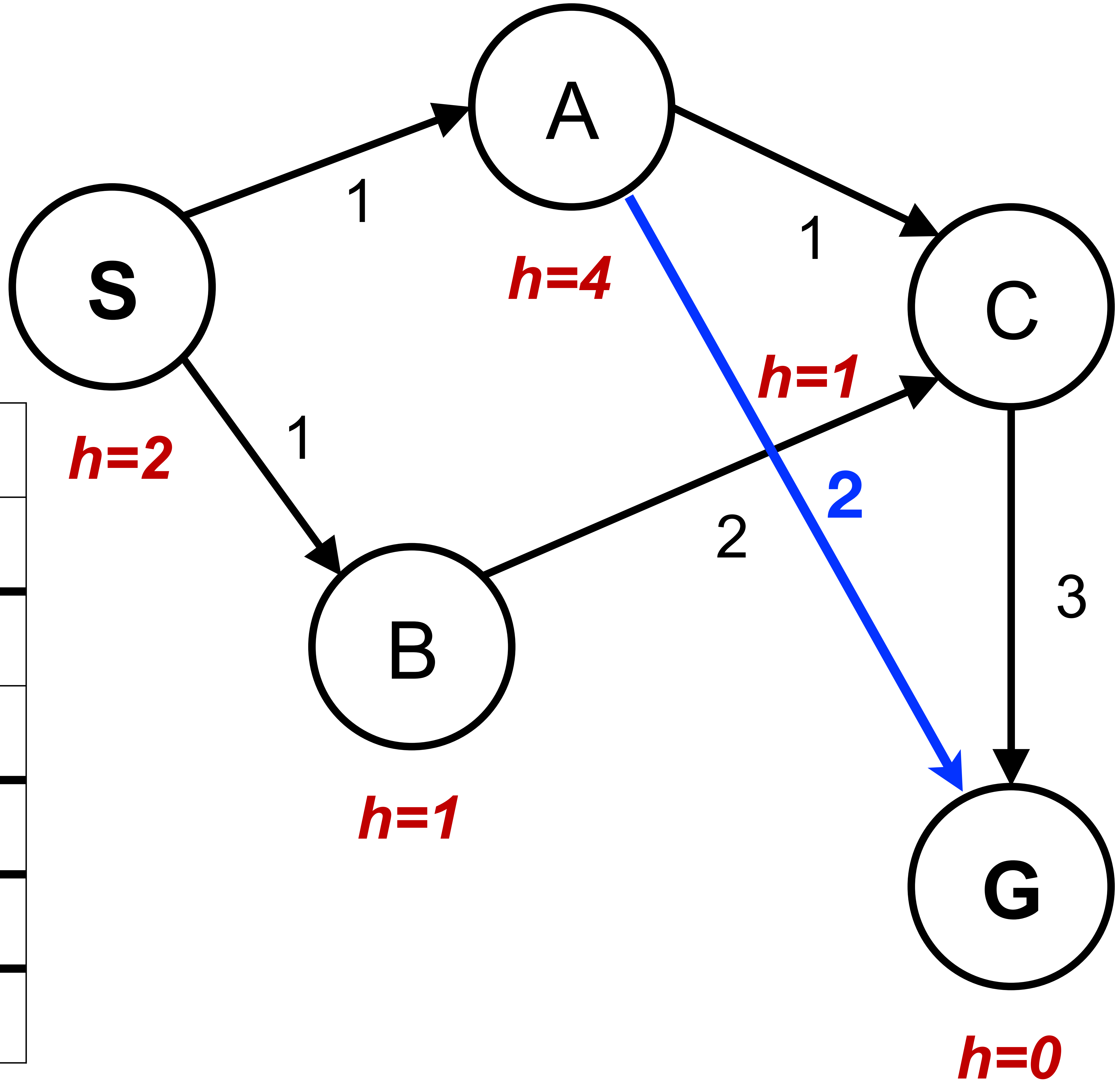
A new trial/sample estimate at time t

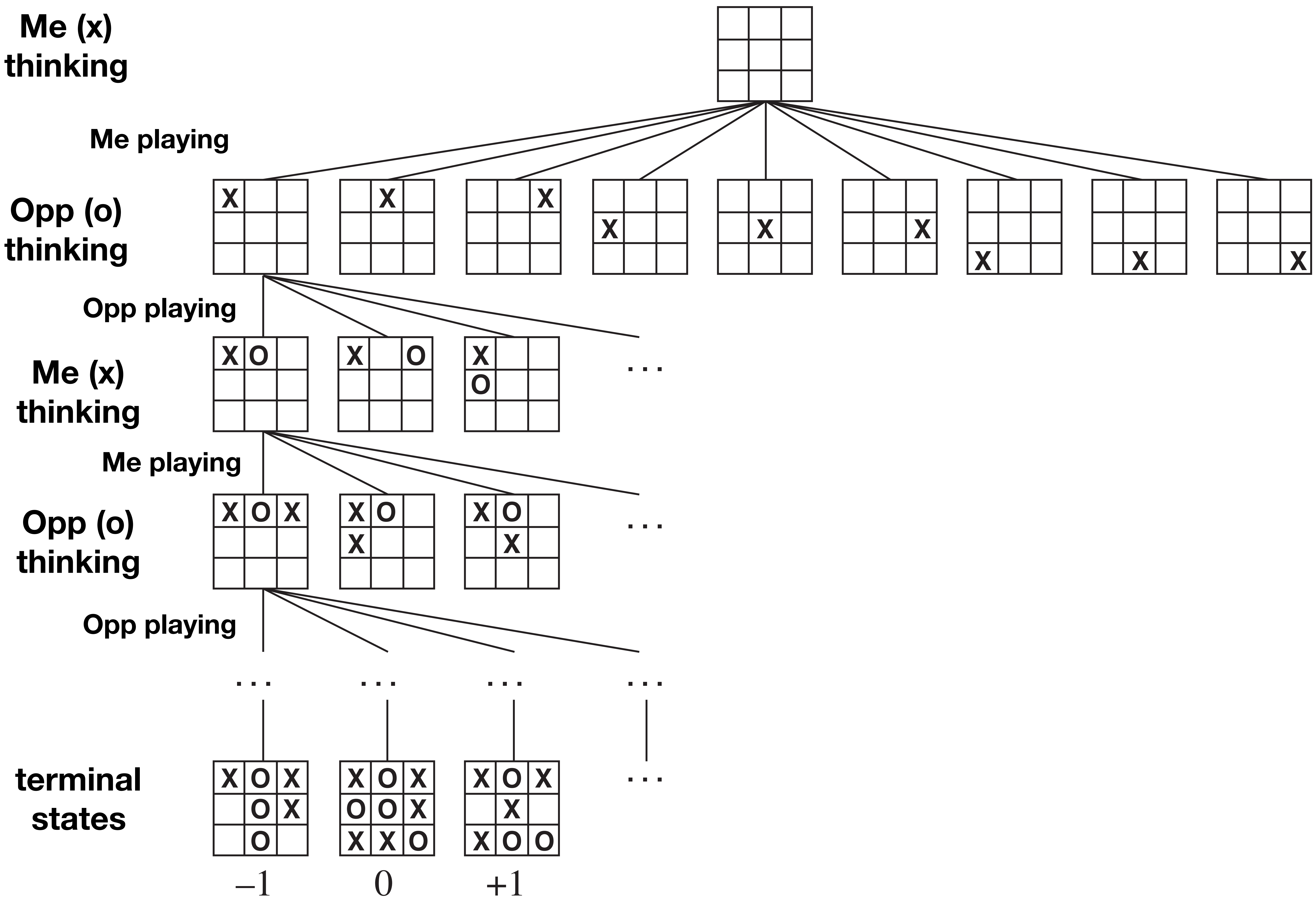
$$\text{trial} = R_{t+1} + \gamma \max_a Q(S_{t+1}, a)$$

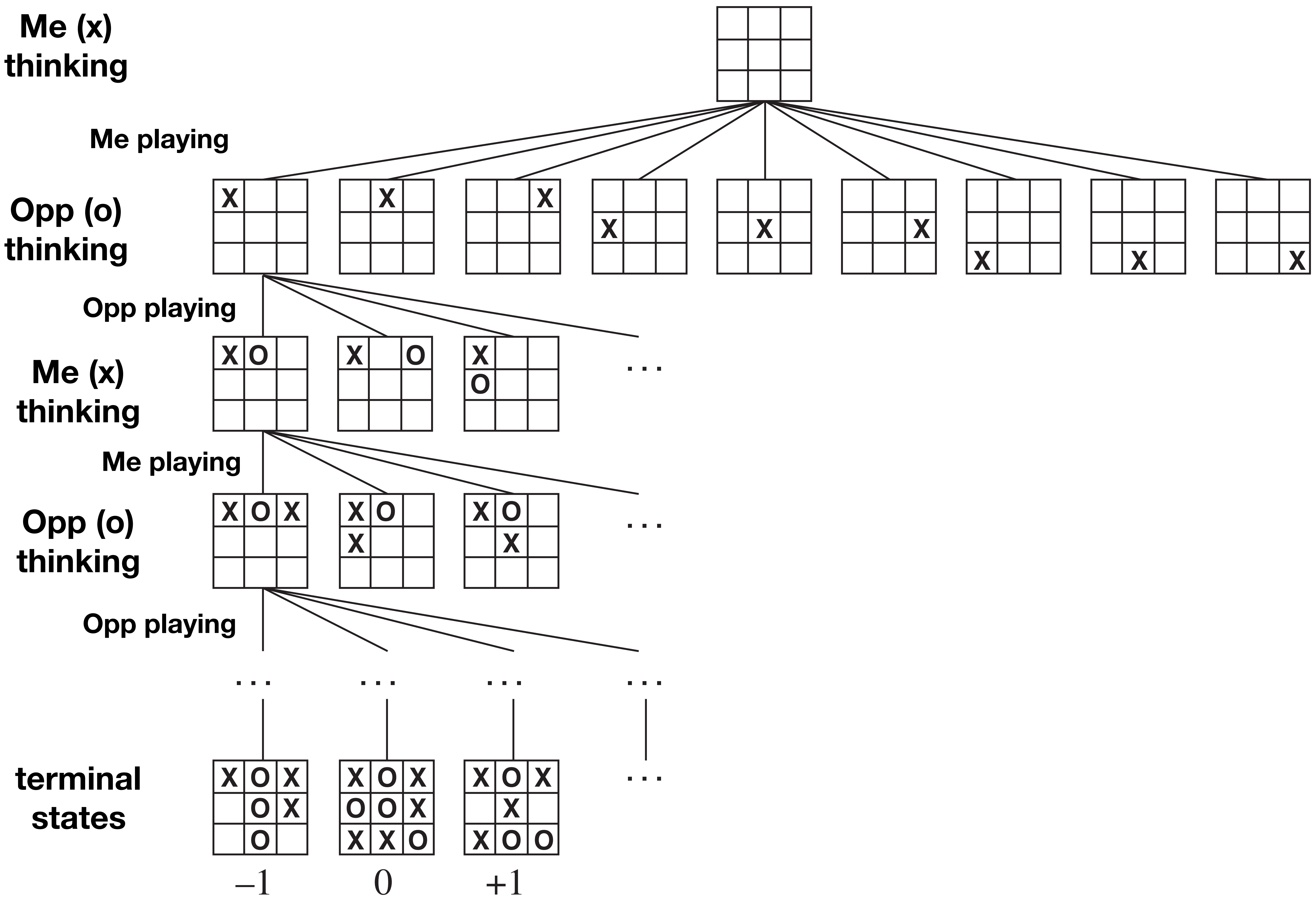
α update

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(\text{trial} - Q(S_t, A_t))$$

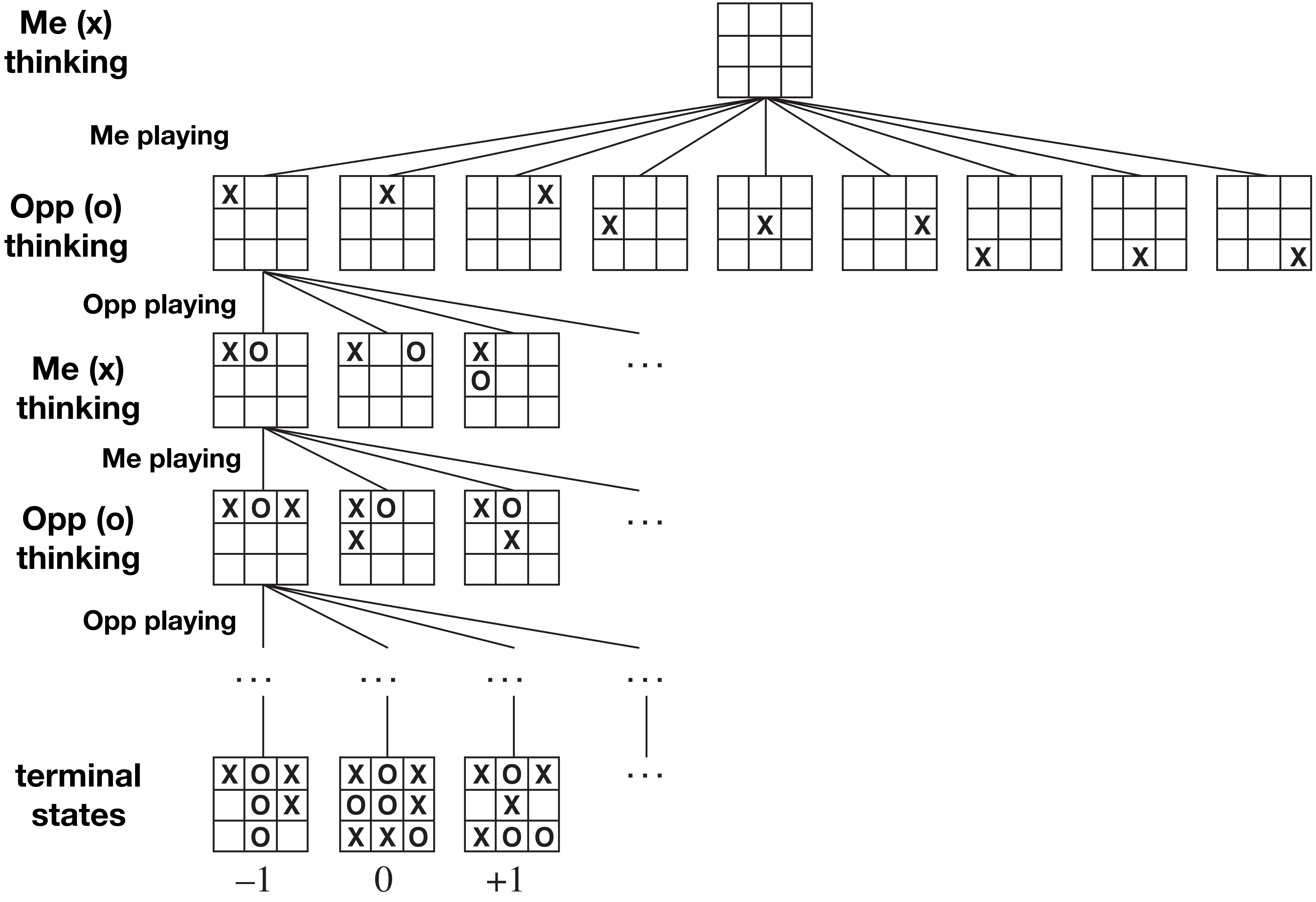
S	left	0	-1		-1							
	right	0		-1								
A	left	0			6							
	right	0	-2									
B	go	0		-2								
C	go	0		7								
G	exit	0	10	10								




Me (x) thinking
Me playing
Opp (o) thinking
Opp playing
Me (x) thinking
Me playing
Opp (o) thinking
Opp playing
terminal states
−1
0
+1

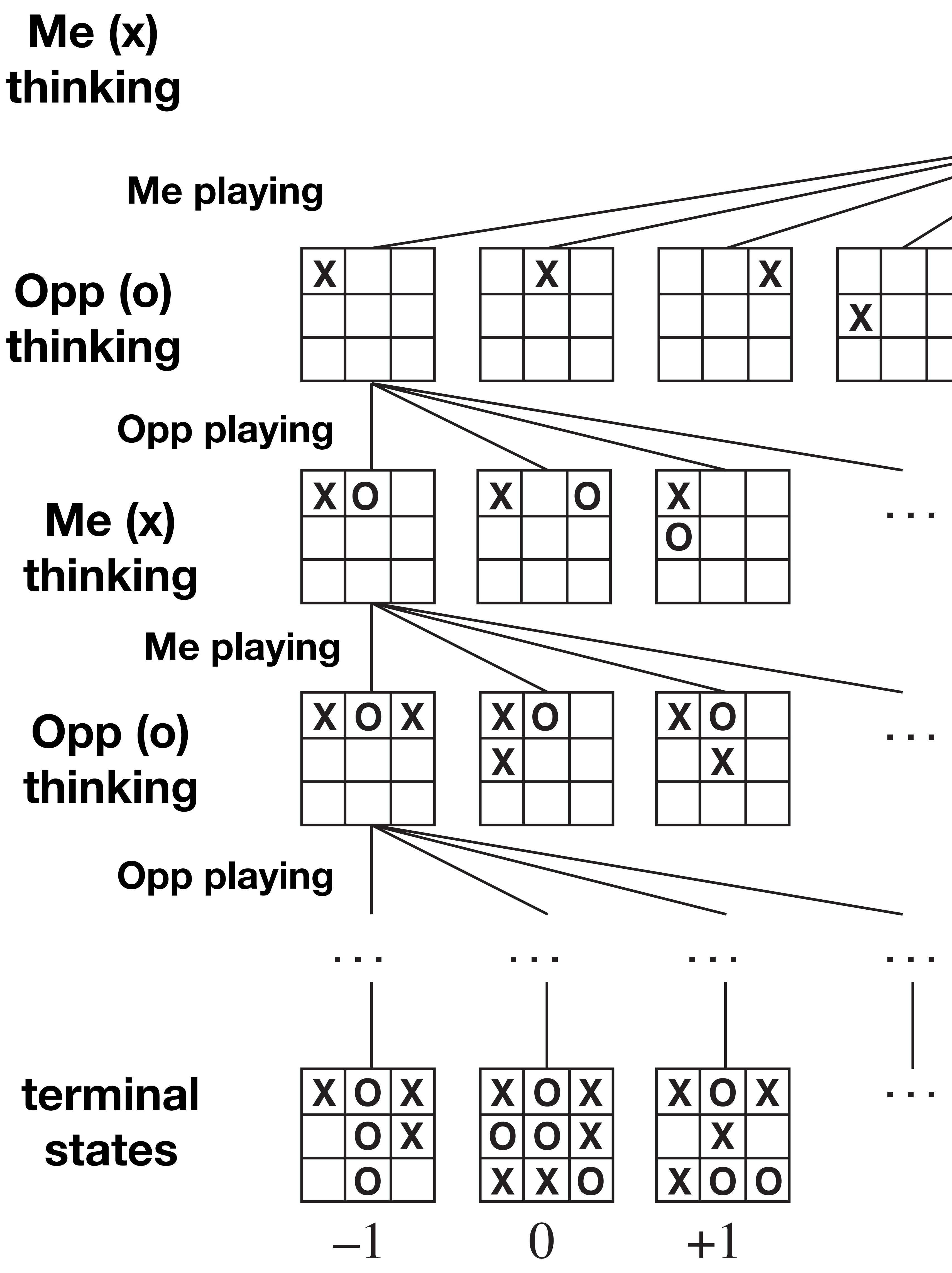

Me (x) thinking
Me playing
Opp (o) thinking
Opp playing
Me (x) thinking
Me playing
Opp (o) thinking
Opp playing
terminal states
–1
0
+1

Player 1: Me



Player 1: Me

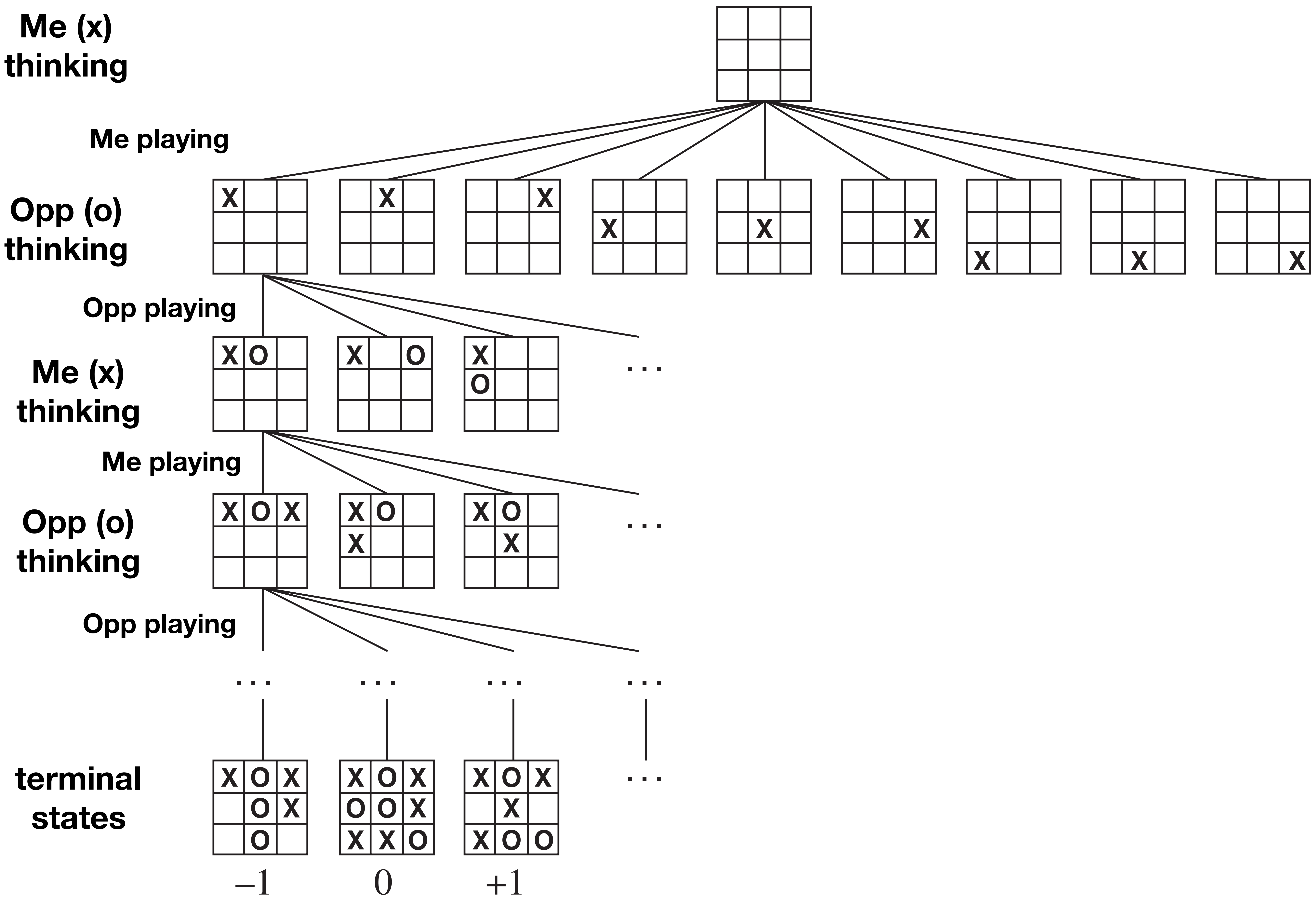
Game Environment



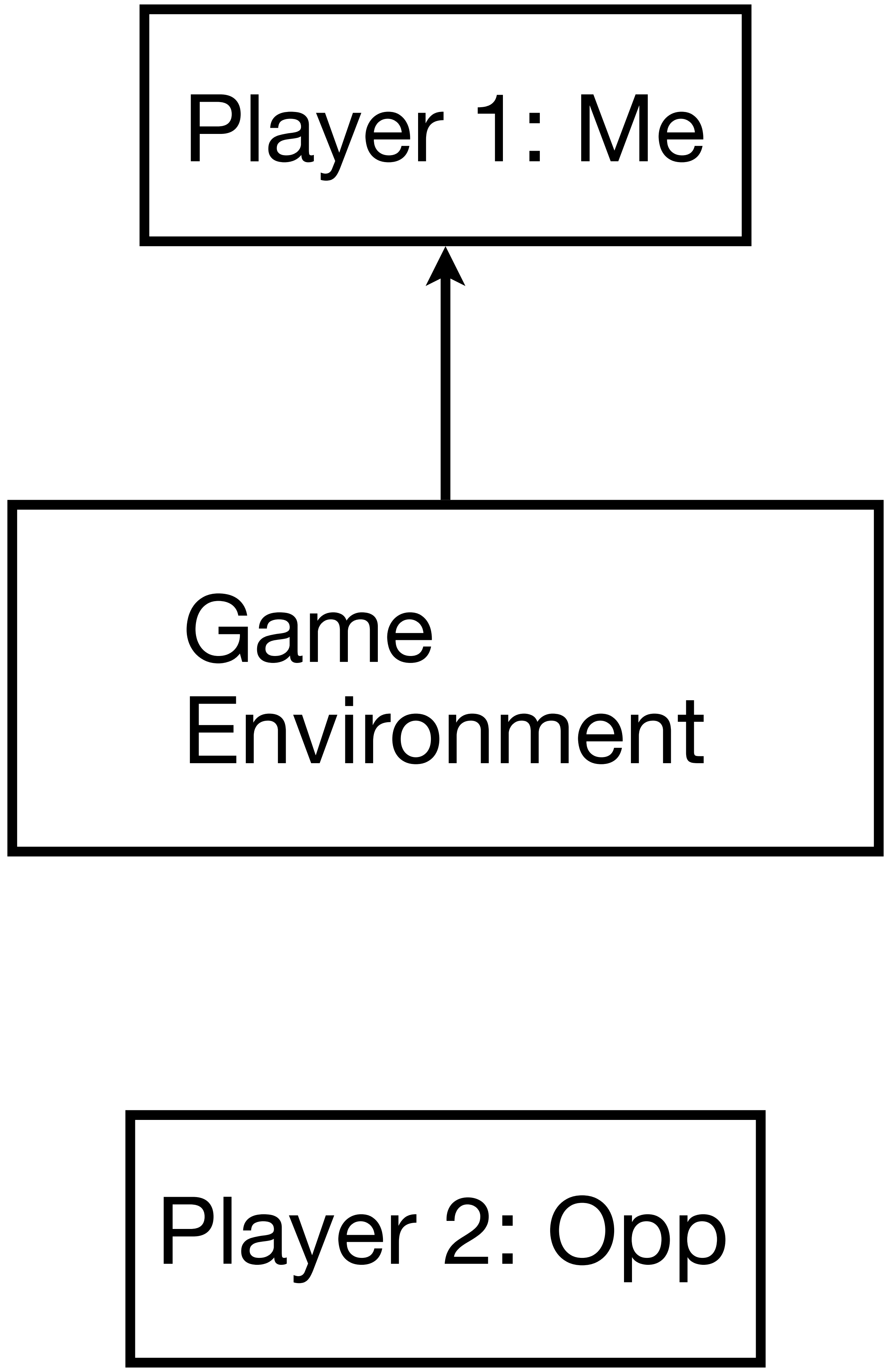
Player 1: Me

Game Environment

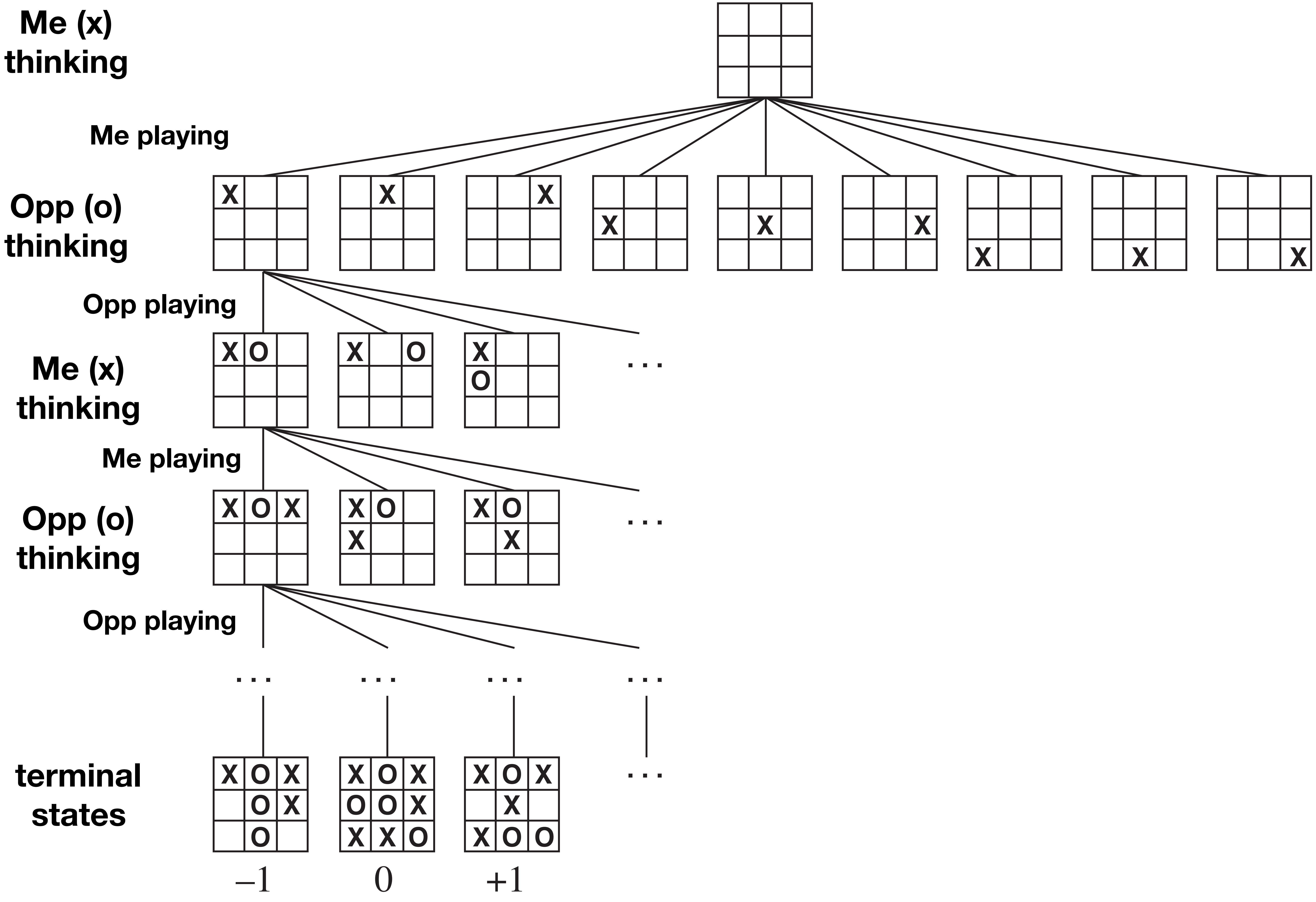
Player 2: Opp

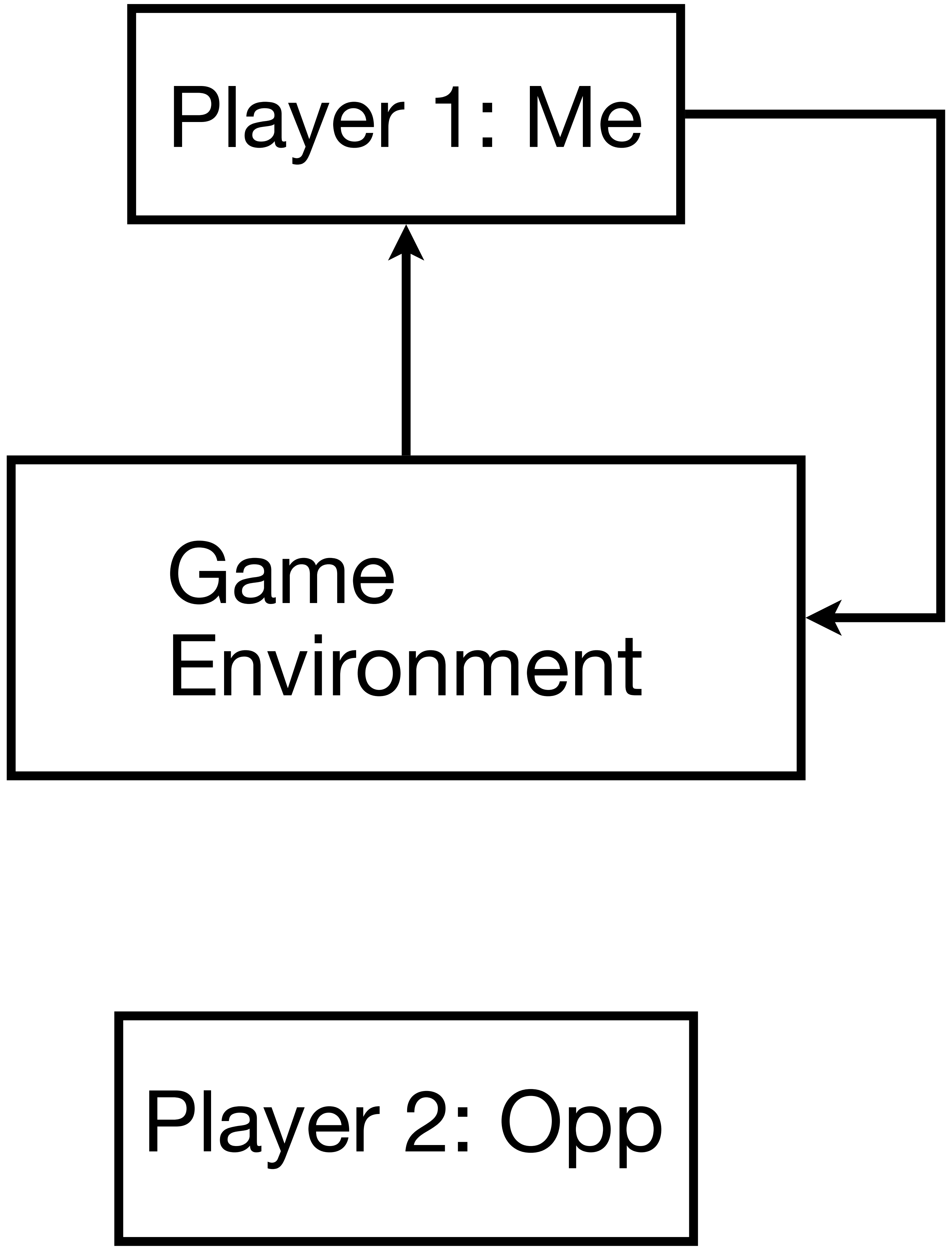


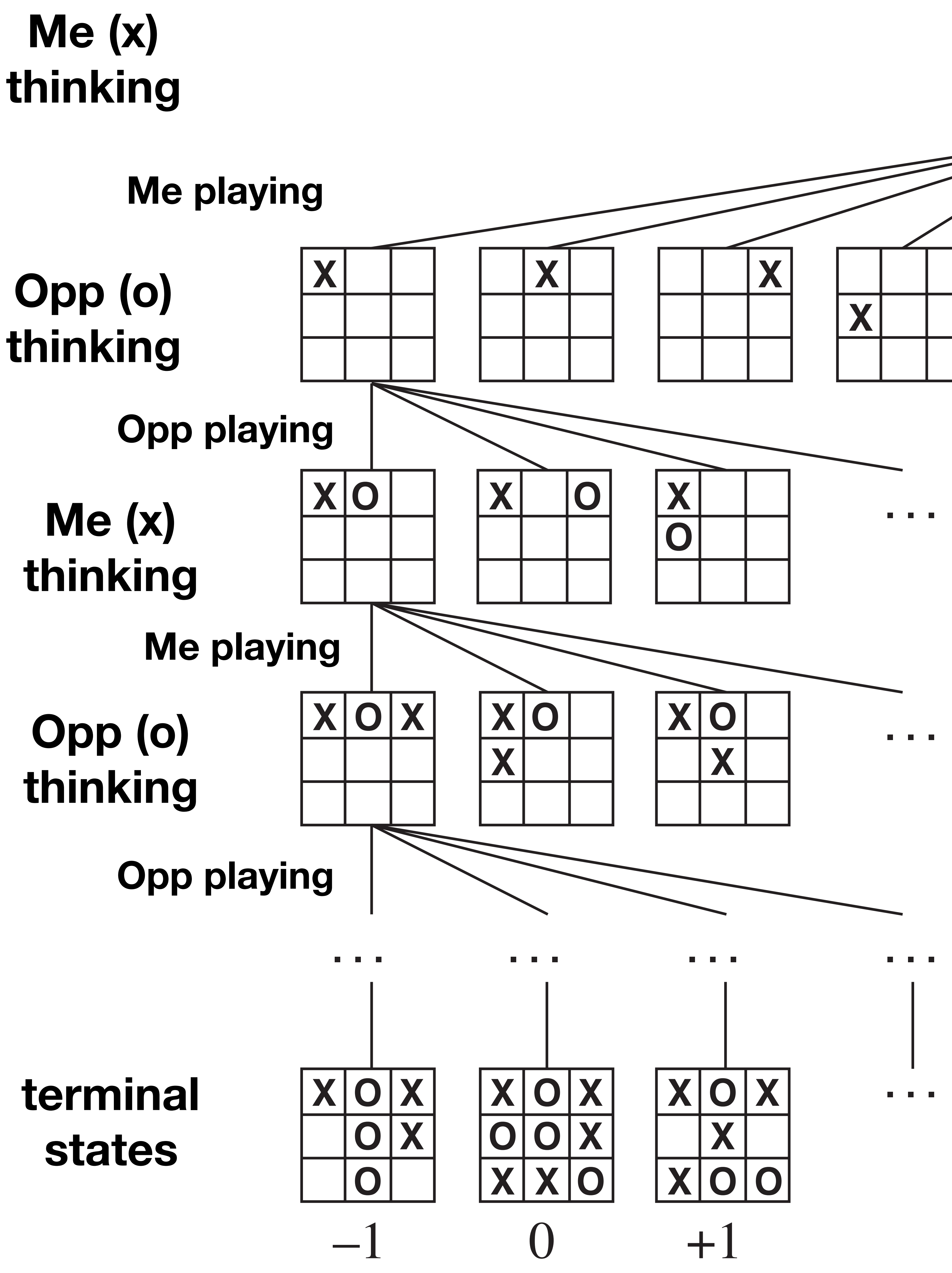
Me (x) thinking
Me playing
Opp (o) thinking
Opp playing
Me (x) thinking
Me playing
Opp (o) thinking
Opp playing
terminal states
–1
0
+1



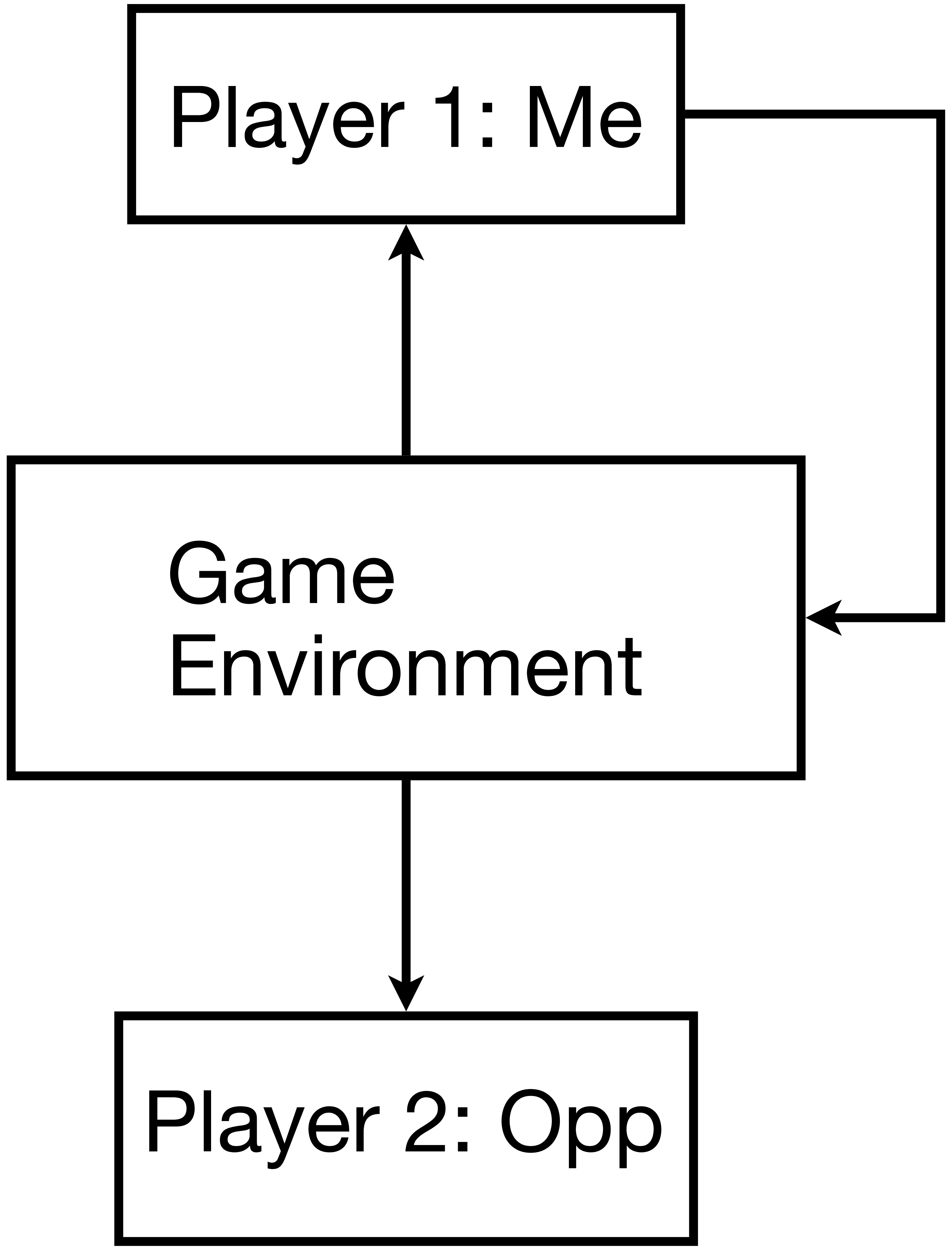
Player 1: Me
Game Environment
Player 2: Opp


Me (x)
thinking
Me playing
Opp (o)
thinking
Opp playing
Me (x)
thinking
Me playing
Opp (o)
thinking
Opp playing
terminal
states
−1
0
+1

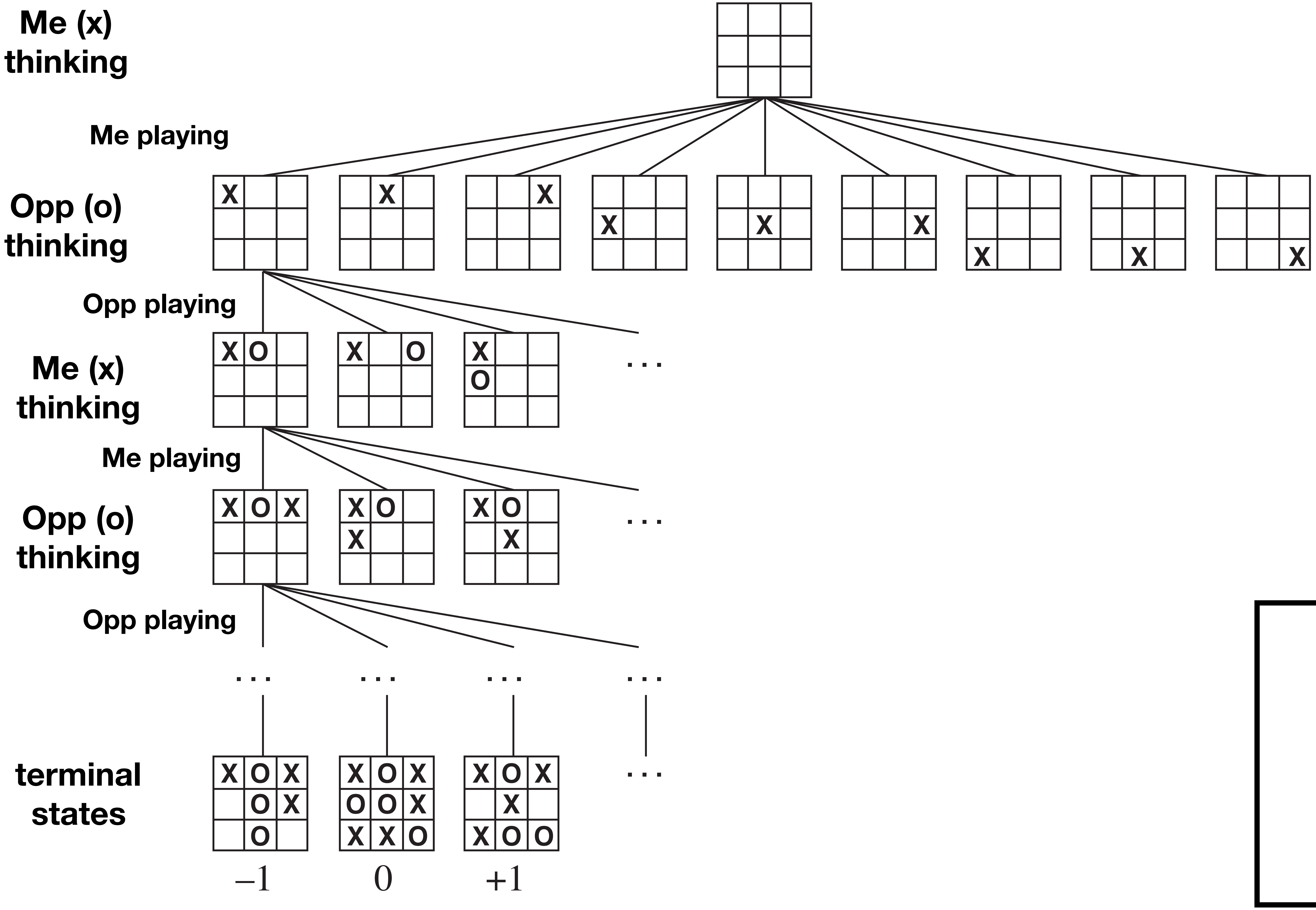

Player 1: Me
Game
Environment
Player 2: Opp

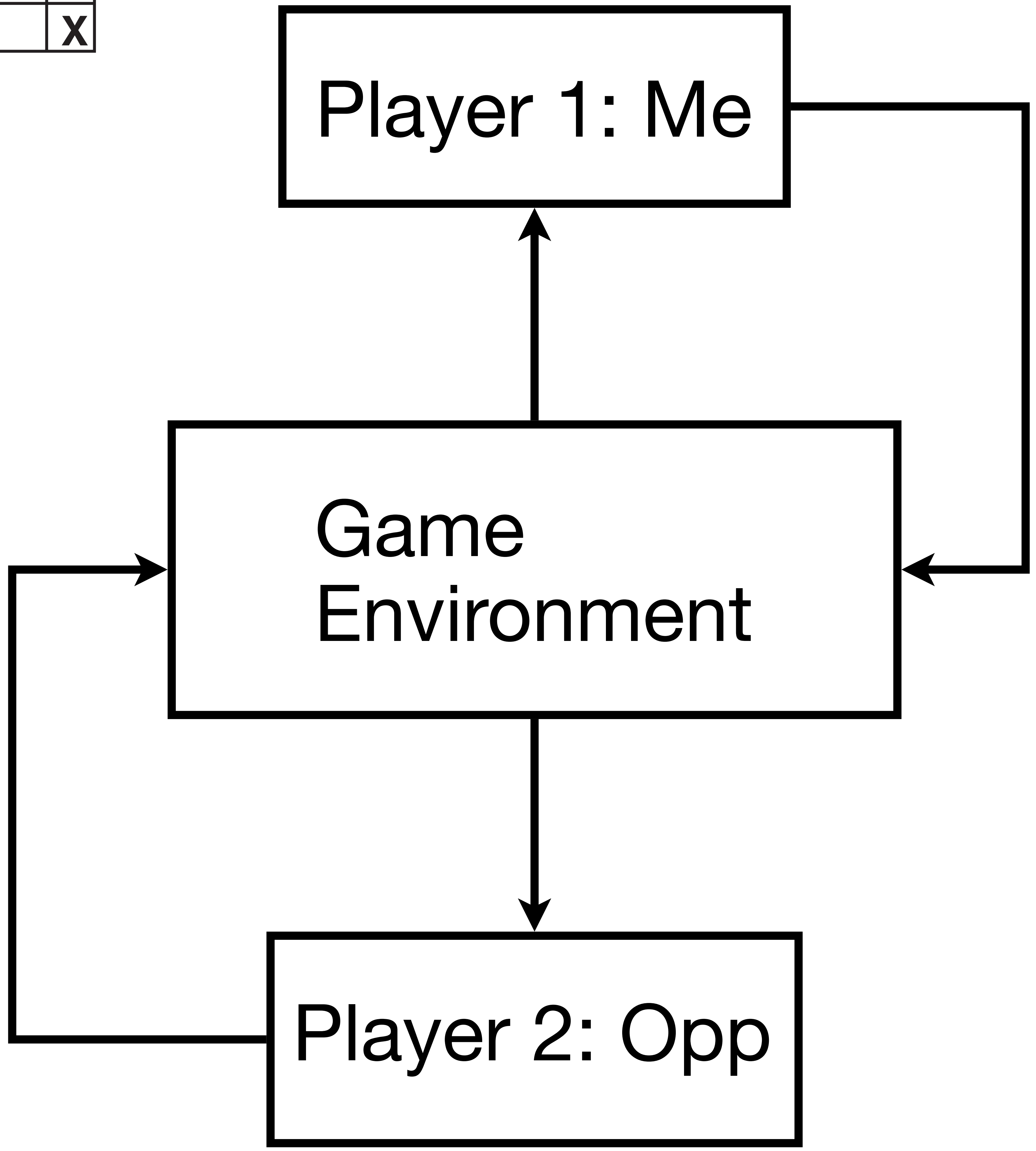


Me (x) thinking
Me playing
Opp (o) thinking
Opp playing
Me (x) thinking
Me playing
Opp (o) thinking
Opp playing
terminal states
–1
0
+1



Player 1: Me
Game Environment
Player 2: Opp

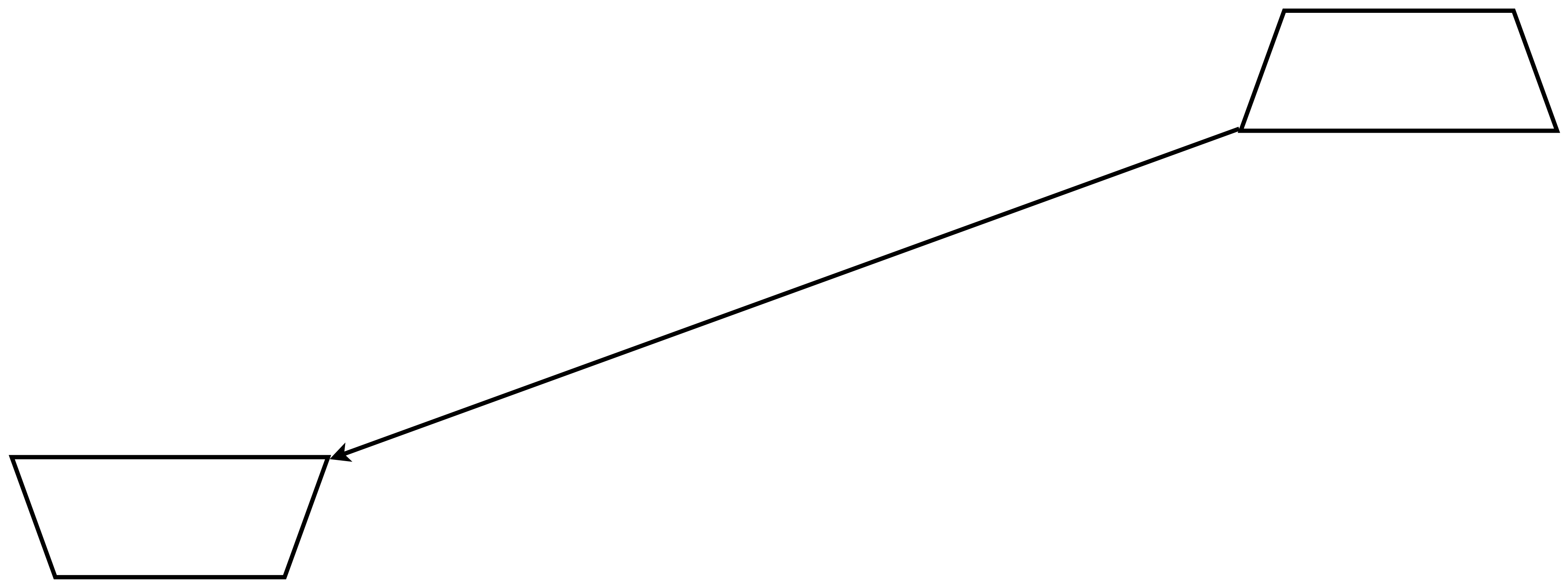

Me (x) thinking
Me playing
Opp (o) thinking
Opp playing
Me (x) thinking
Me playing
Opp (o) thinking
Opp playing
terminal states
−1
0
+1


Player 1: Me
Game Environment
Player 2: Opp

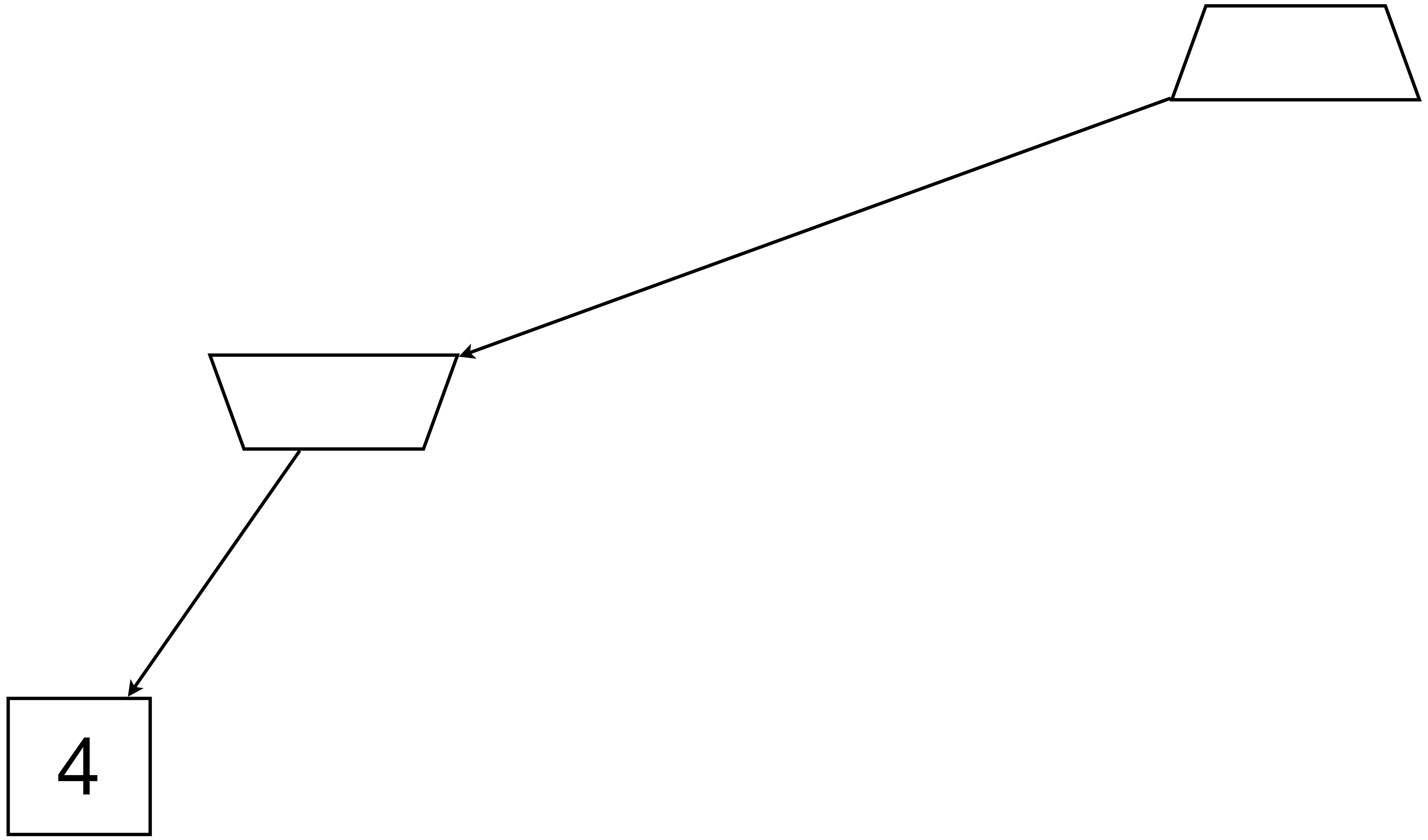
(recursive) thinking game: what if my/opp move is …

(recursive) thinking game: what if my/opp move is …

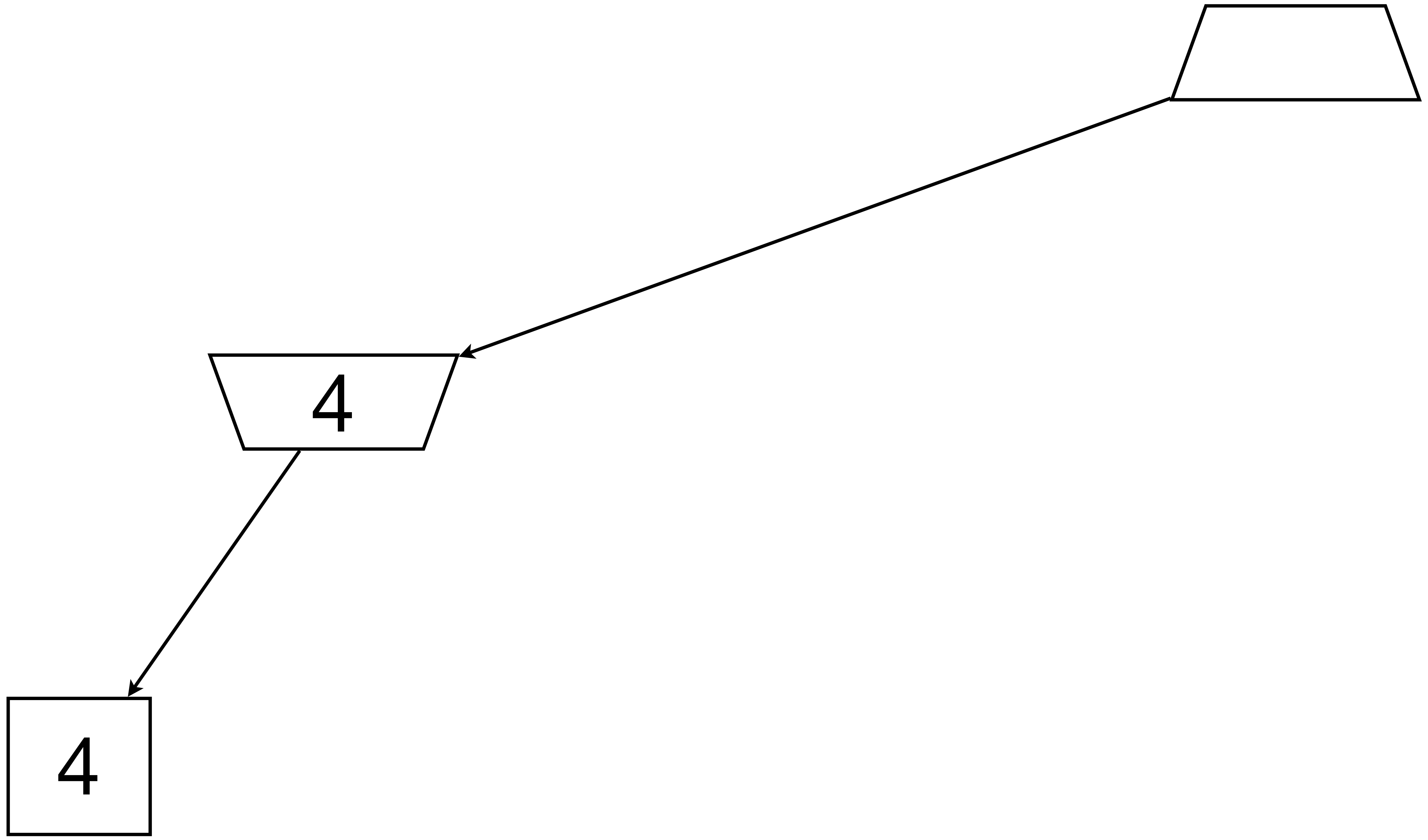
(recursive) thinking game: what if my/opp move is …



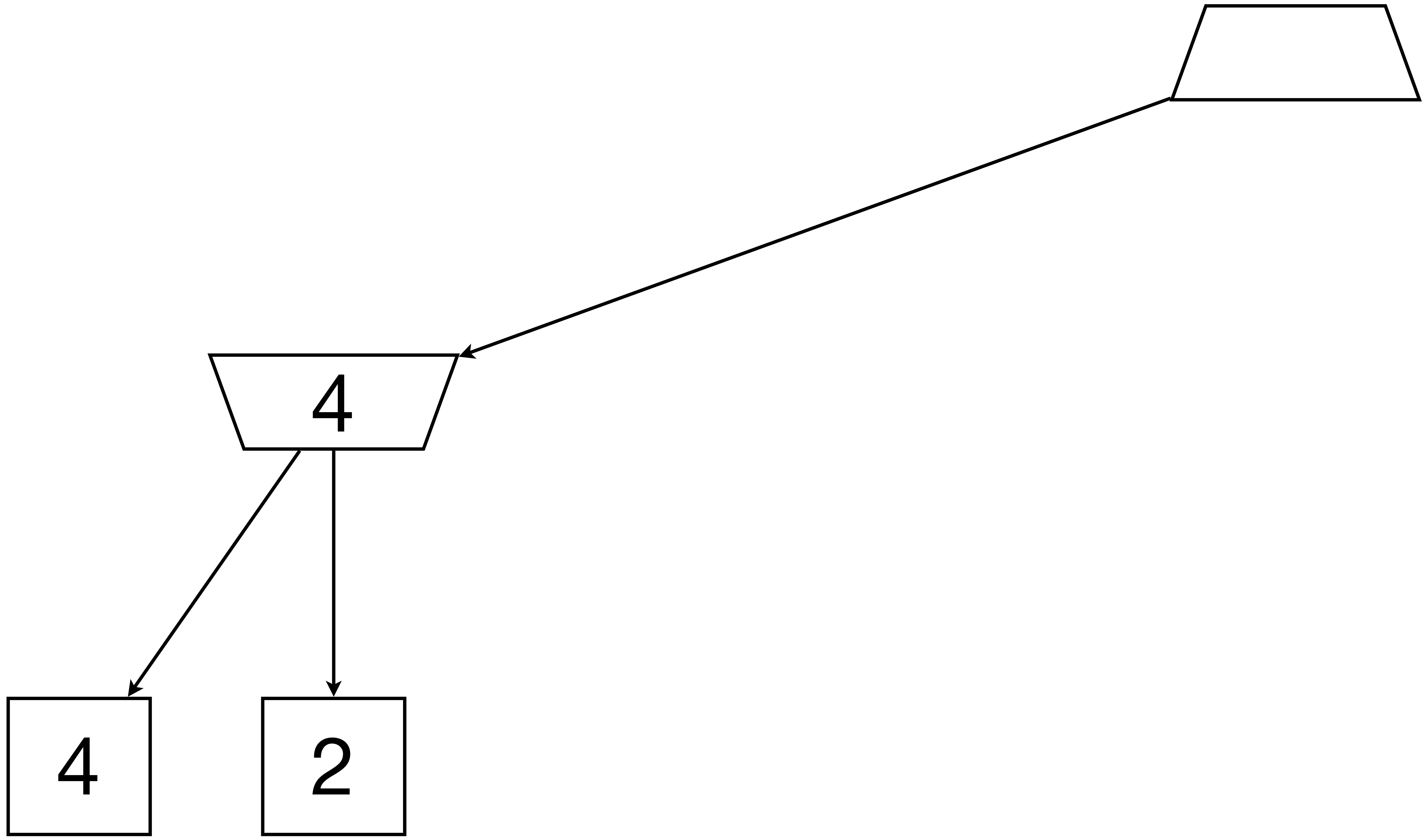
(recursive) thinking game: what if my/opp move is …



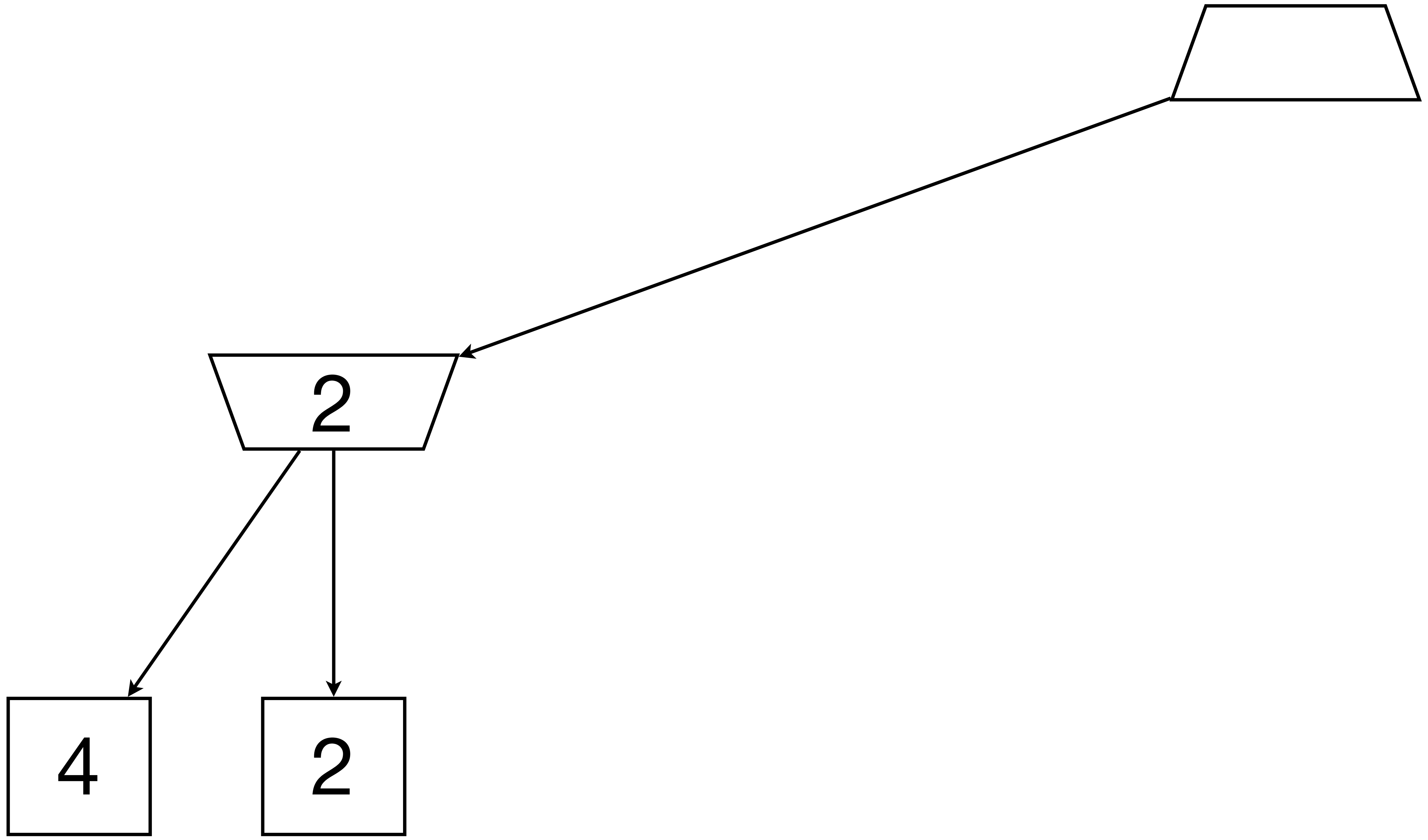
(recursive) thinking game: what if my/opp move is …



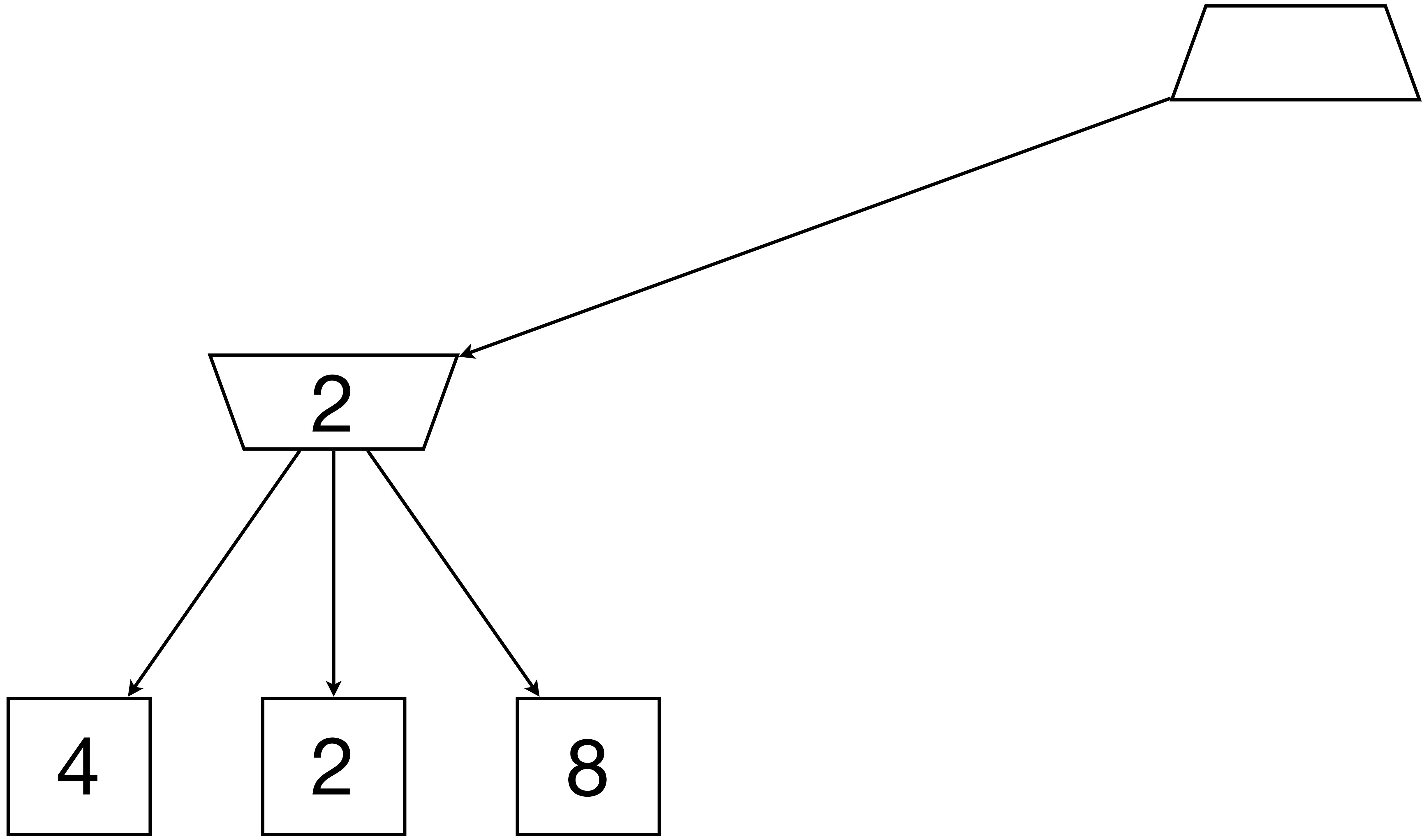
(recursive) thinking game: what if my/opp move is …



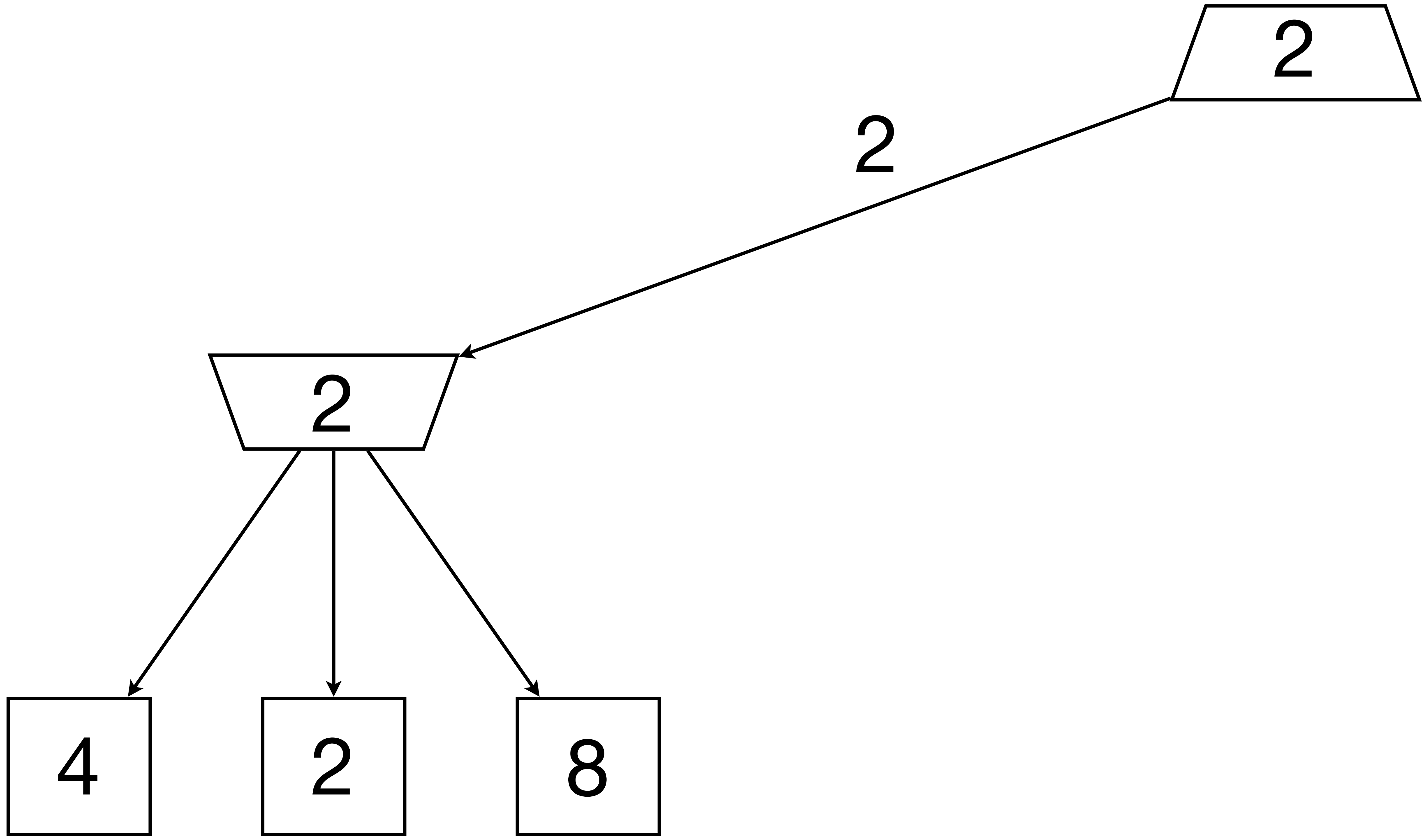
(recursive) thinking game: what if my/opp move is …



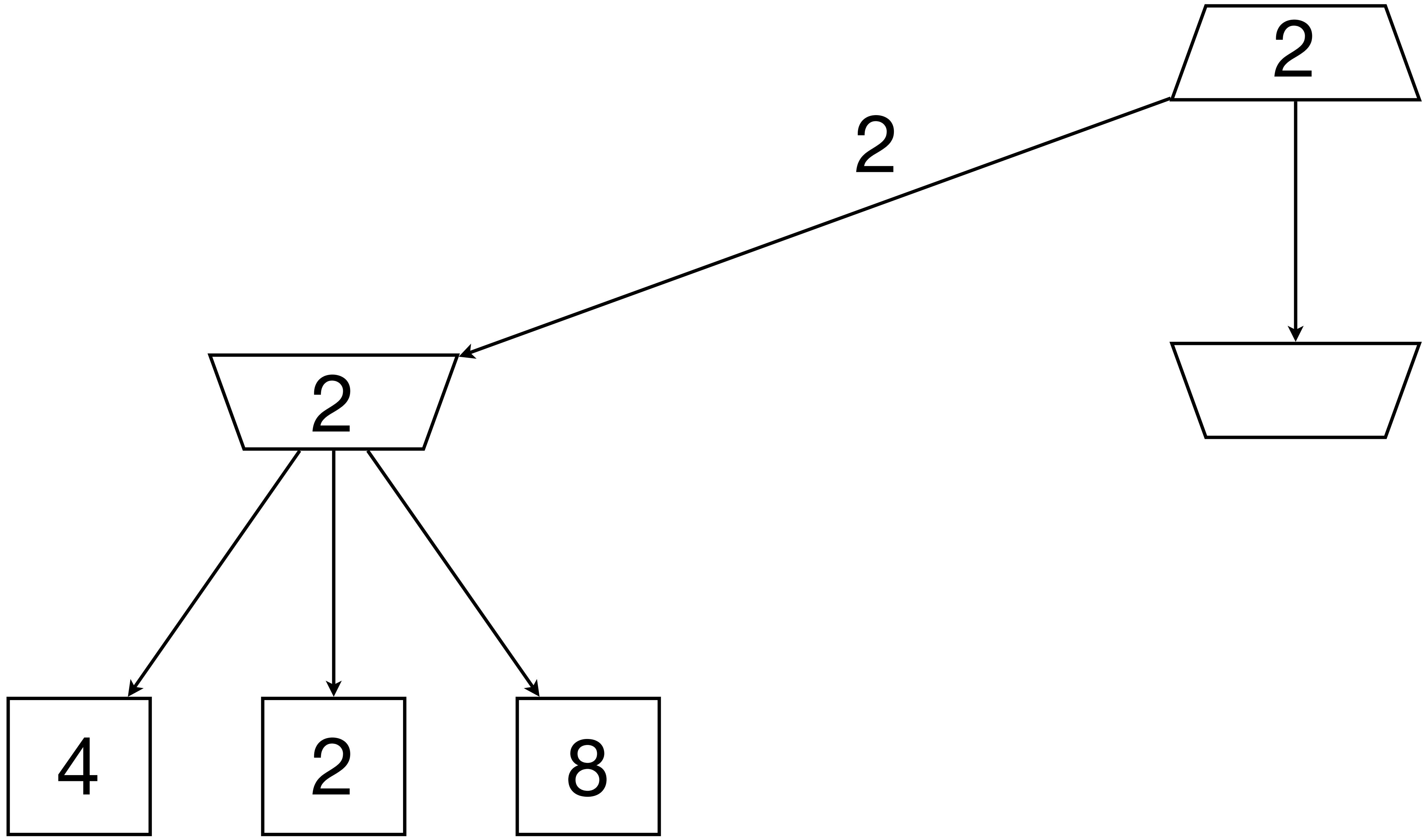
(recursive) thinking game: what if my/opp move is …



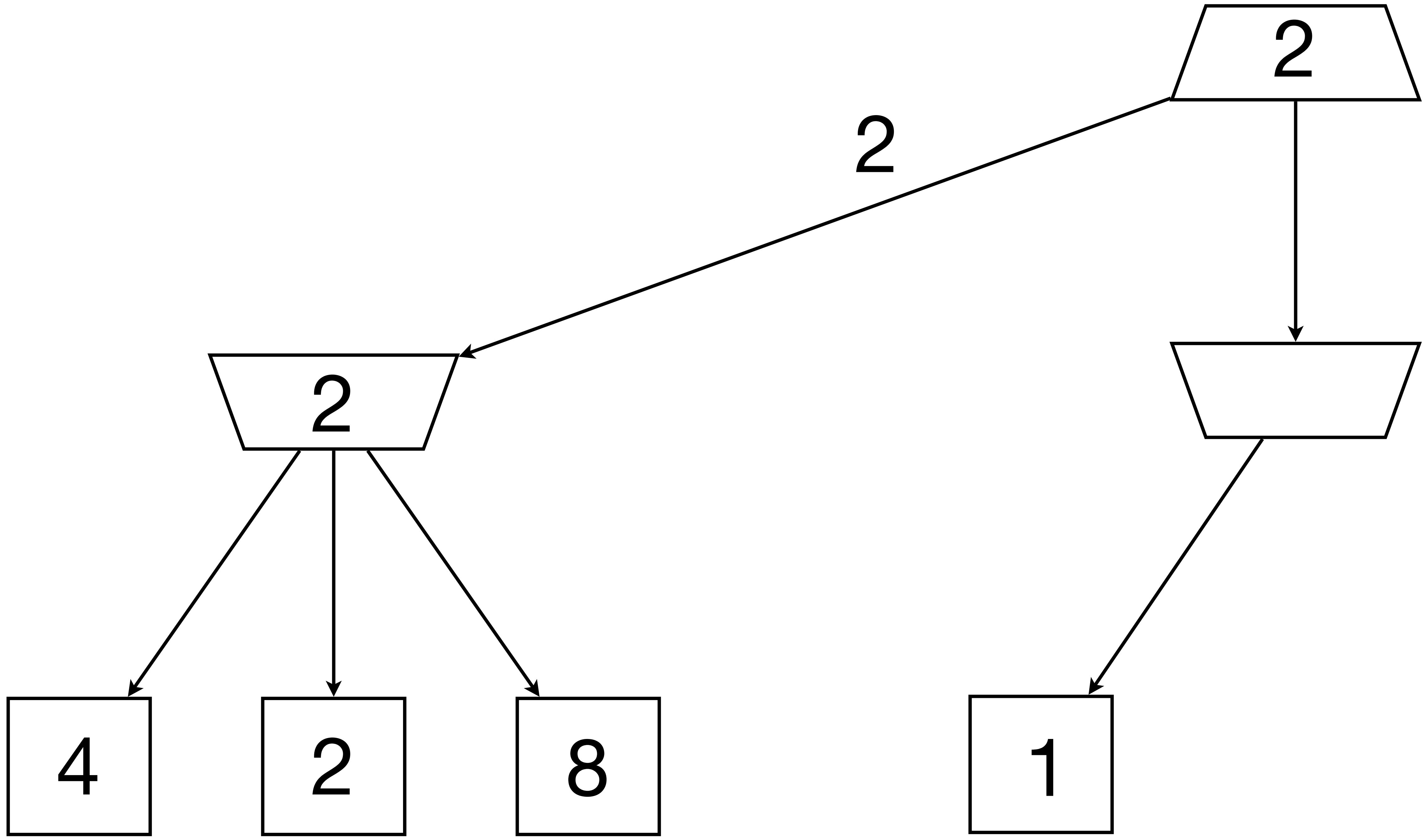
(recursive) thinking game: what if my/opp move is …



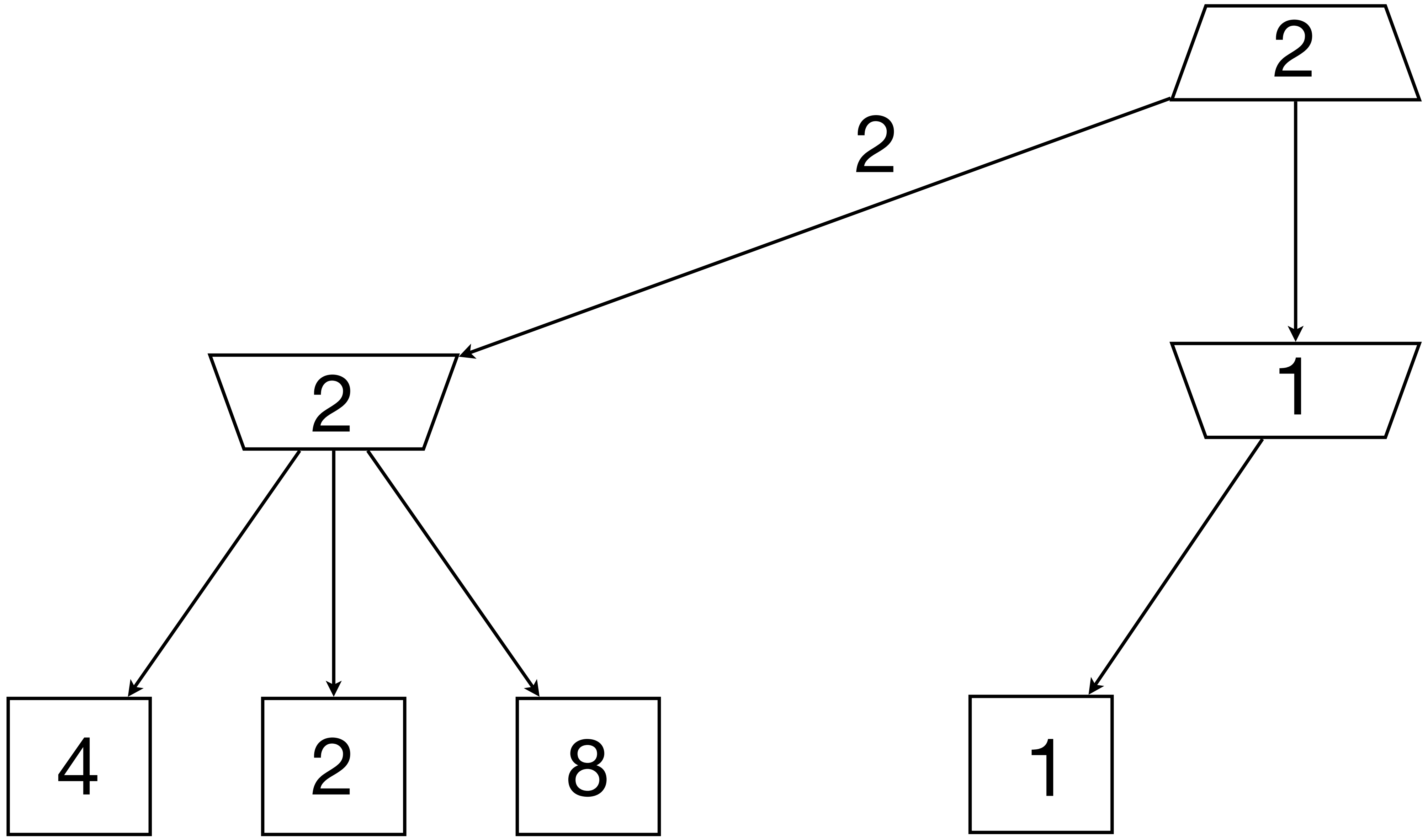
(recursive) thinking game: what if my/opp move is …



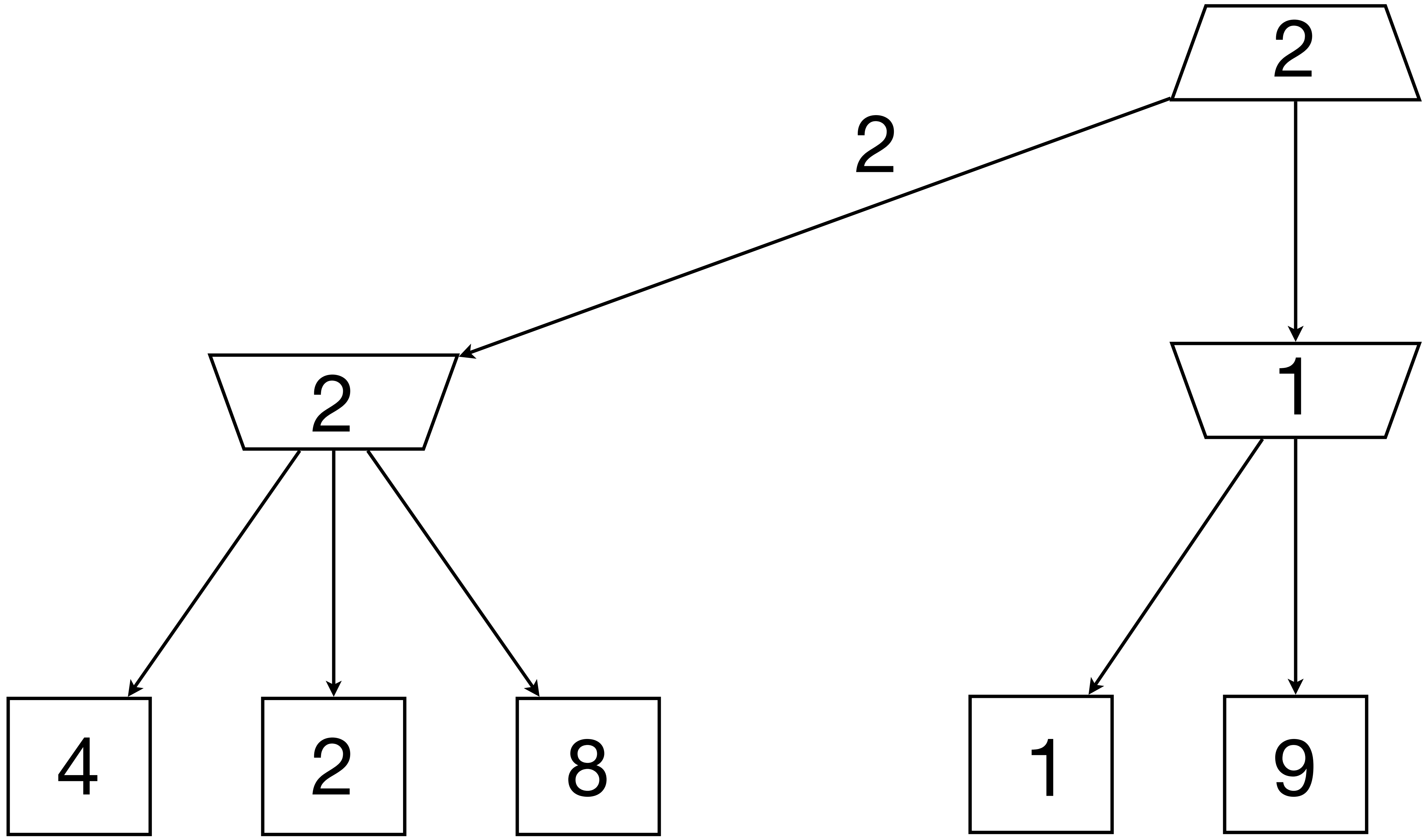
(recursive) thinking game: what if my/opp move is …



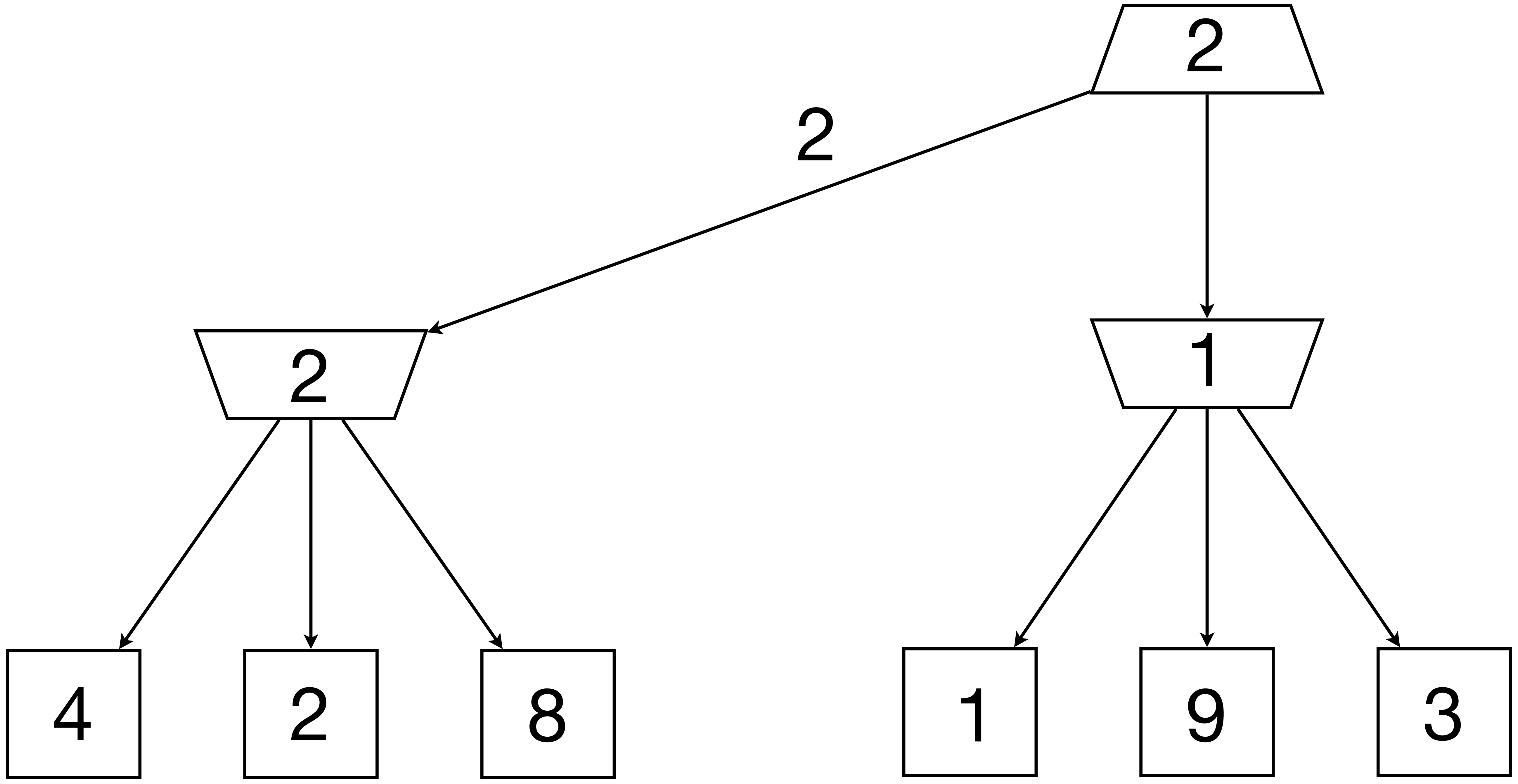
(recursive) thinking game: what if my/opp move is …



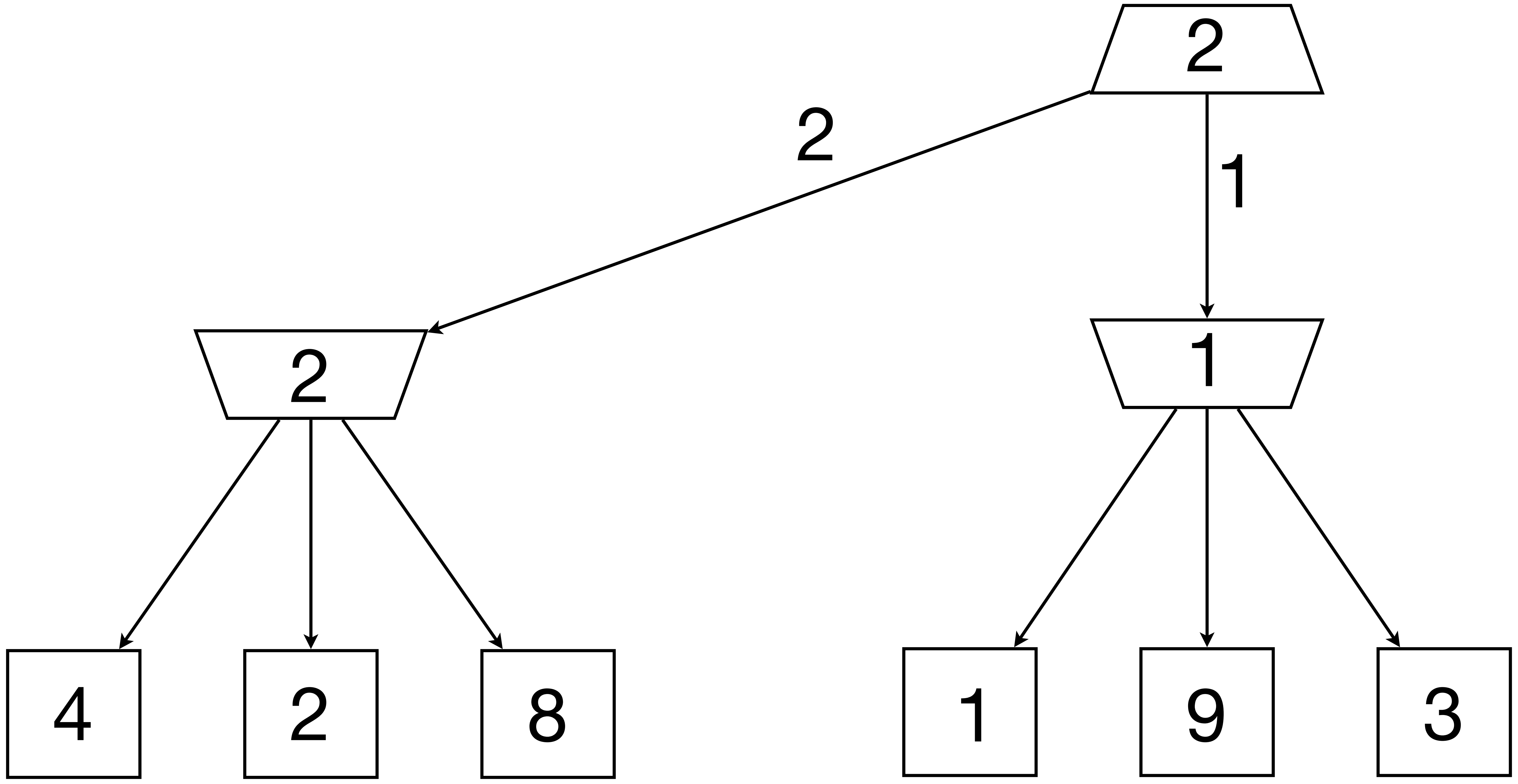
(recursive) thinking game: what if my/opp move is …



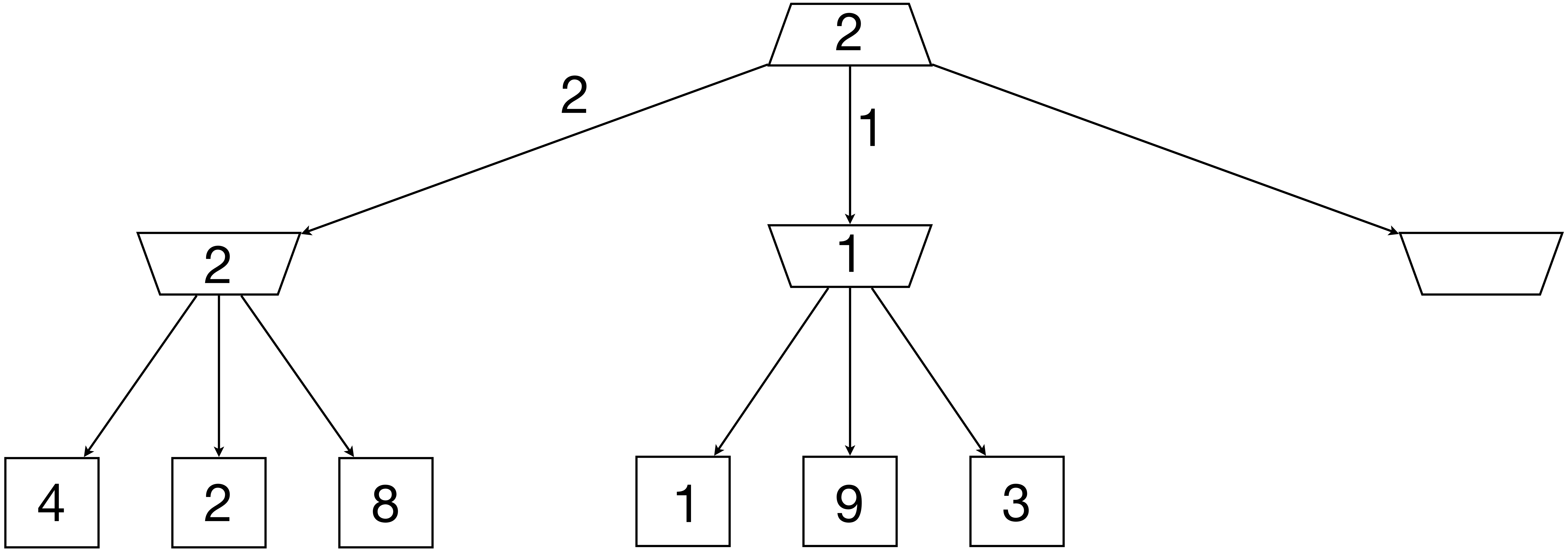
(recursive) thinking game: what if my/opp move is …



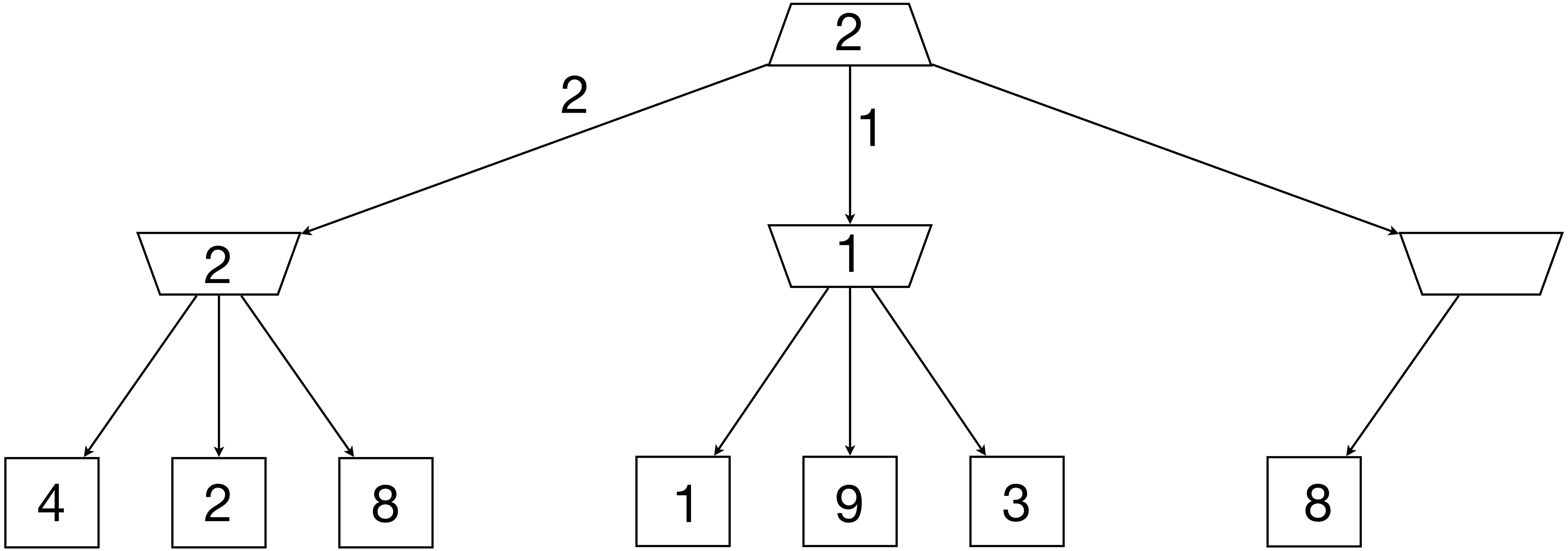
(recursive) thinking game: what if my/opp move is …



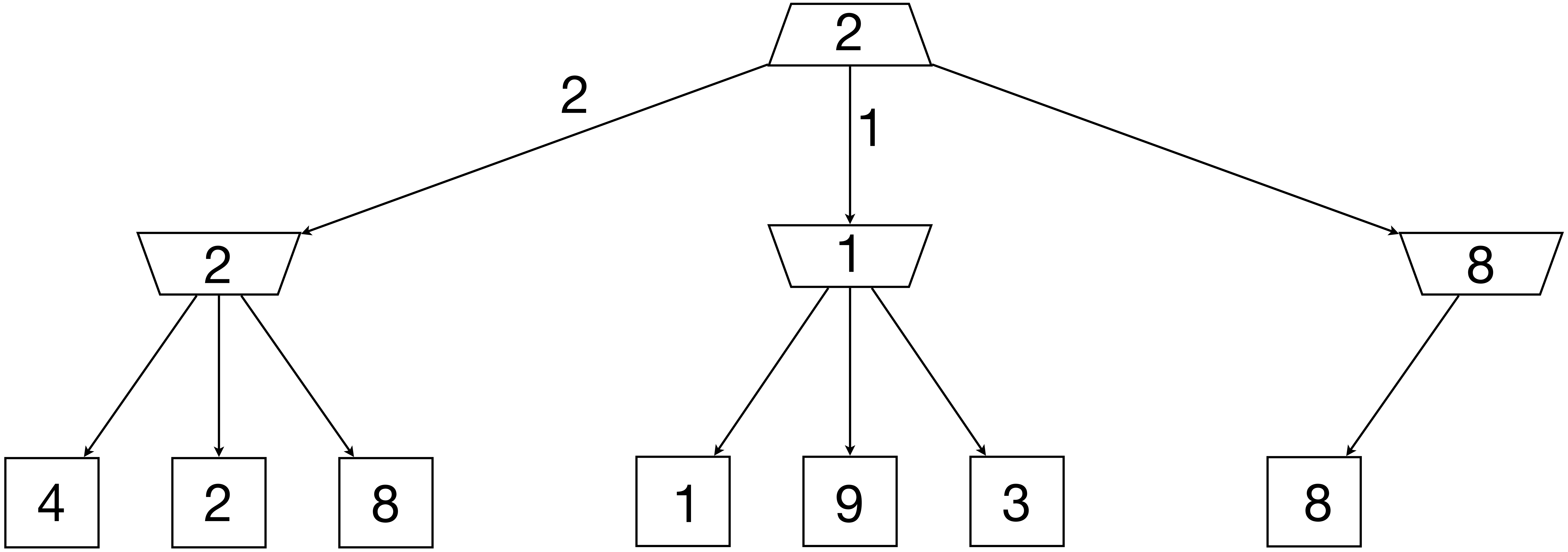
(recursive) thinking game: what if my/opp move is …



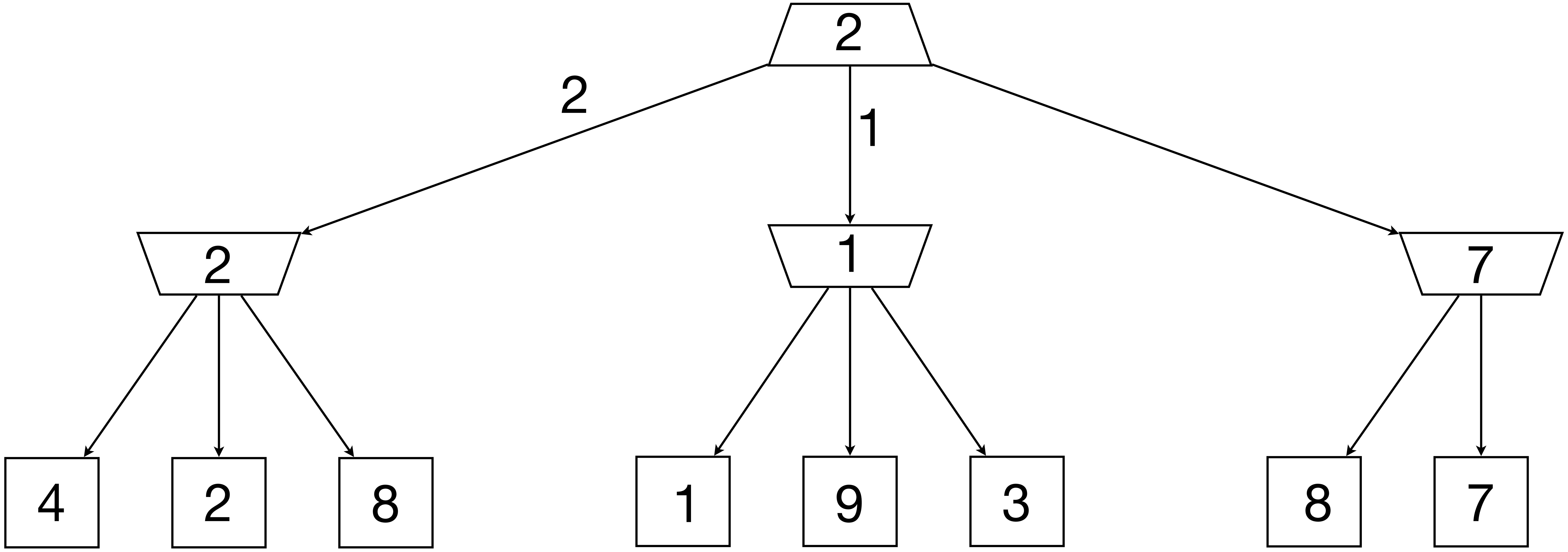
(recursive) thinking game: what if my/opp move is …



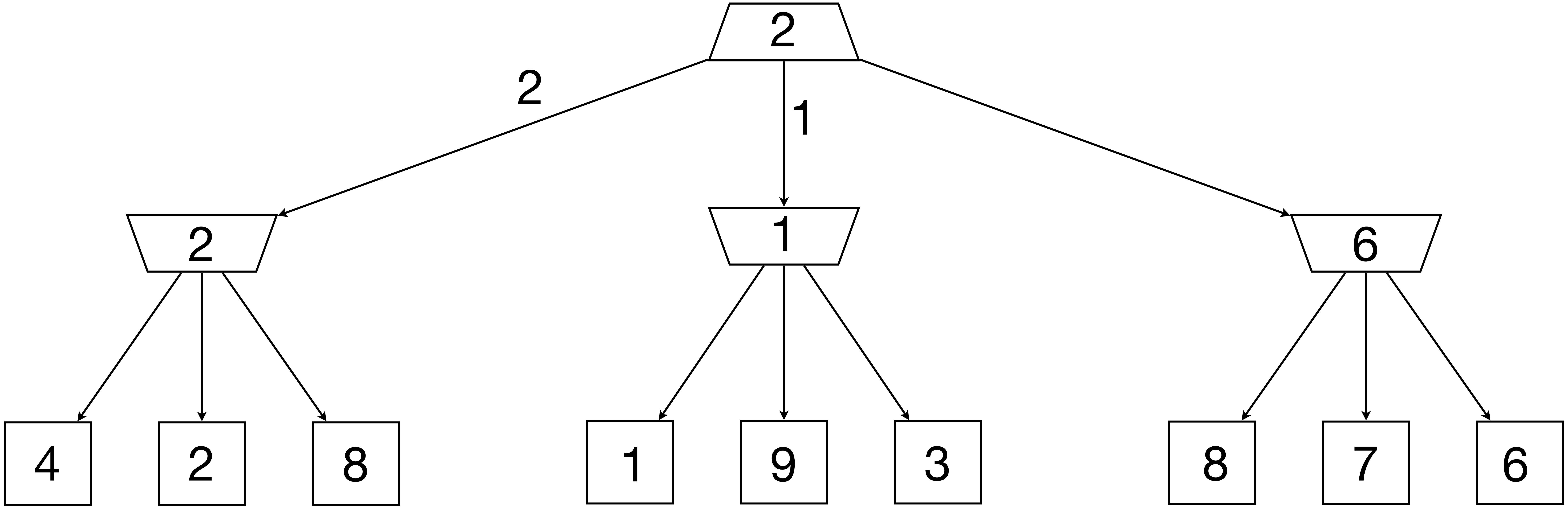
(recursive) thinking game: what if my/opp move is …



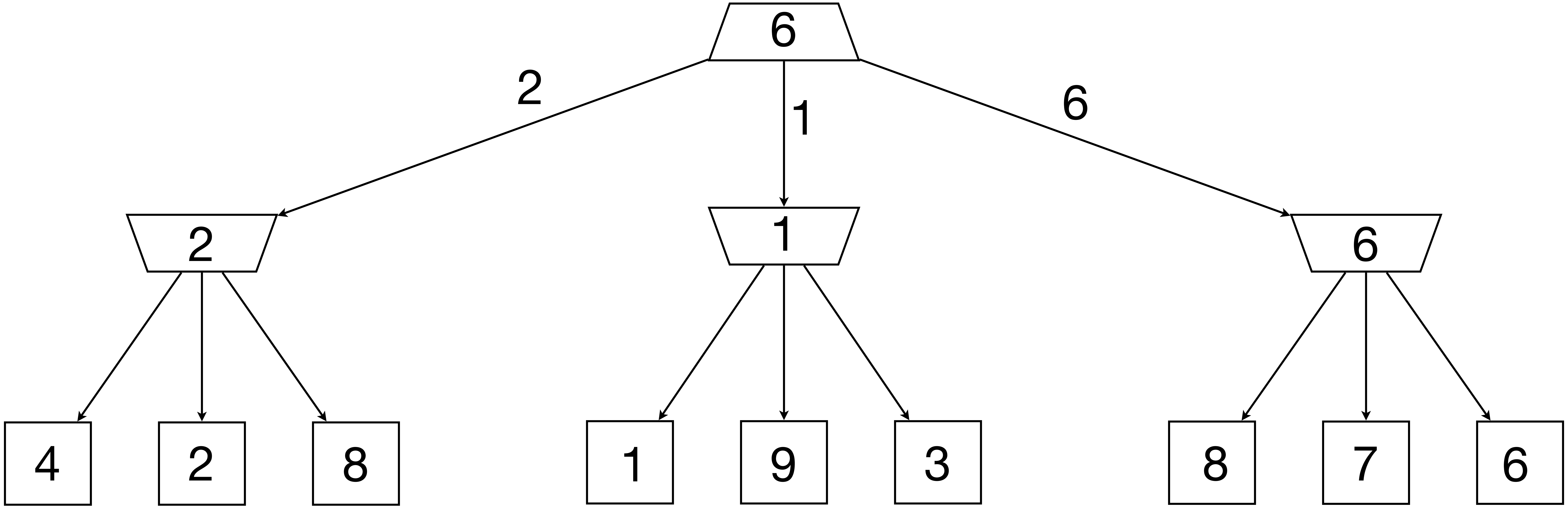
(recursive) thinking game: what if my/opp move is …



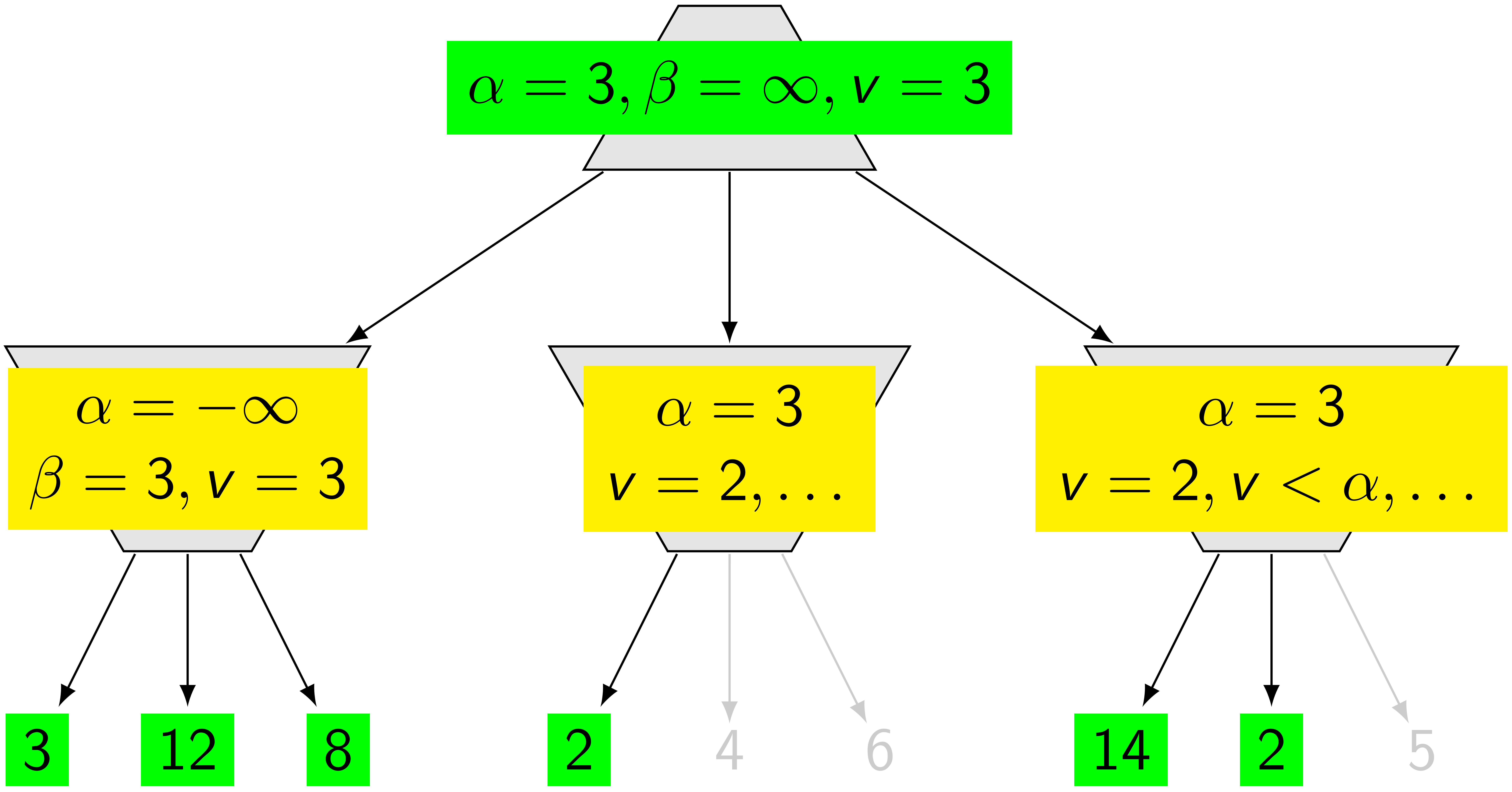
(recursive) thinking game: what if my/opp move is …



(recursive) thinking game: what if my/opp move is …



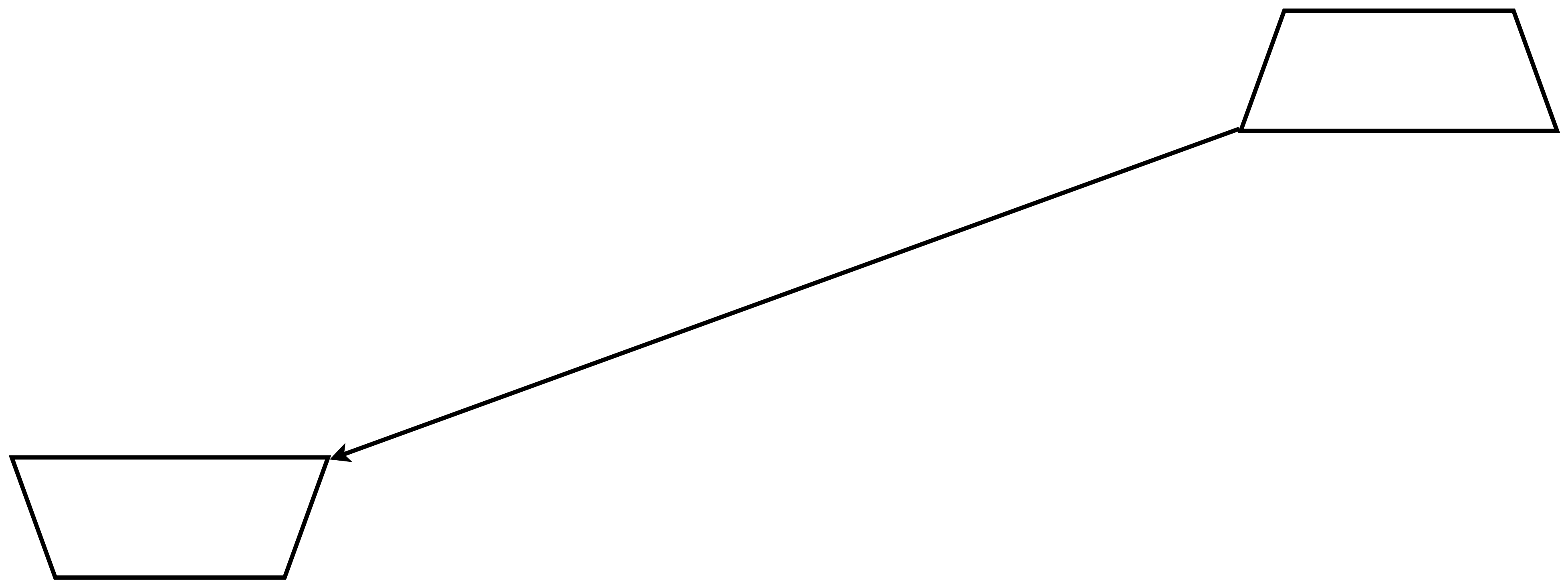
We can go (think) deeper if we prune …



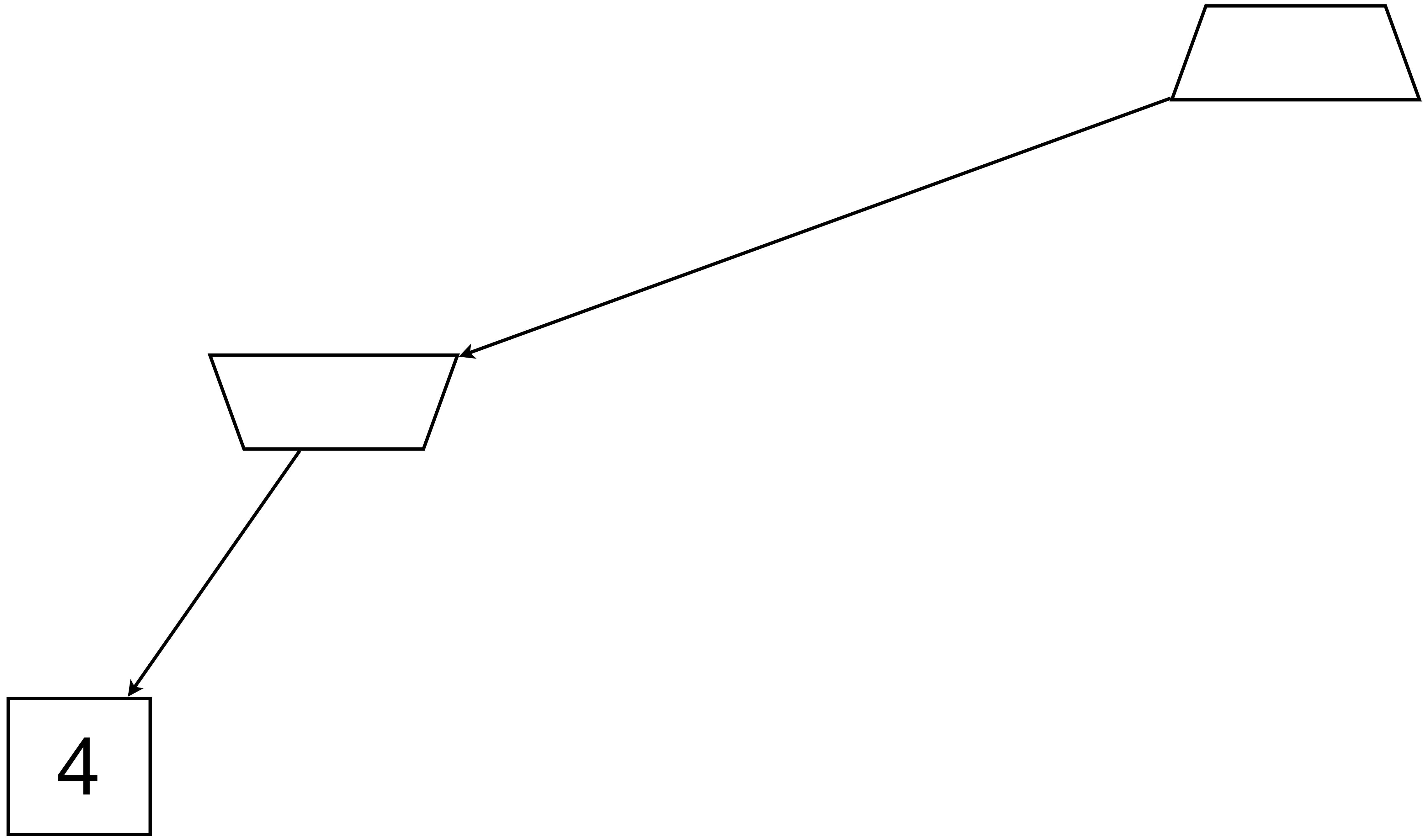
(recursive) thinking game: what if my/opp move is …

(recursive) thinking game: what if my/opp move is …

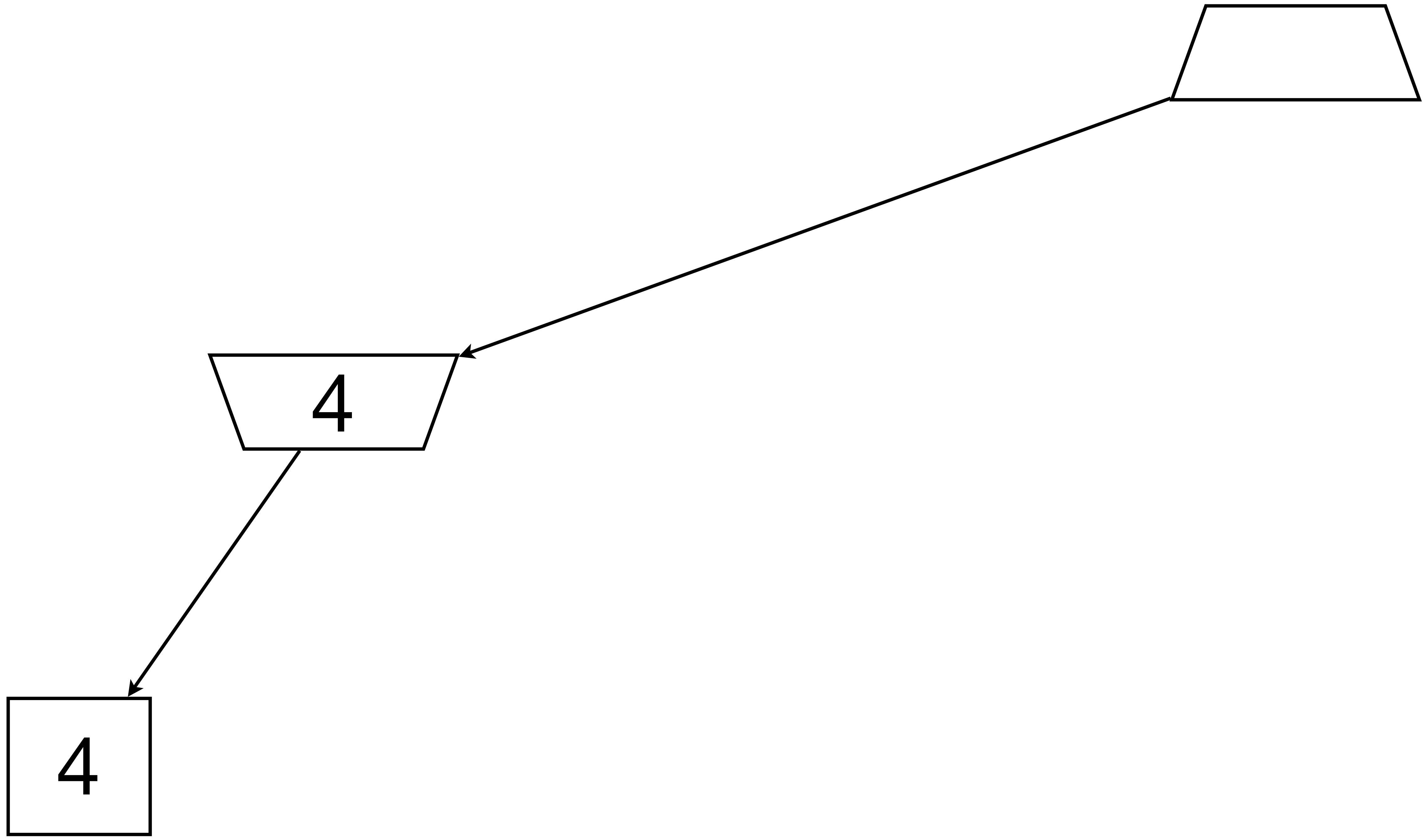
(recursive) thinking game: what if my/opp move is …



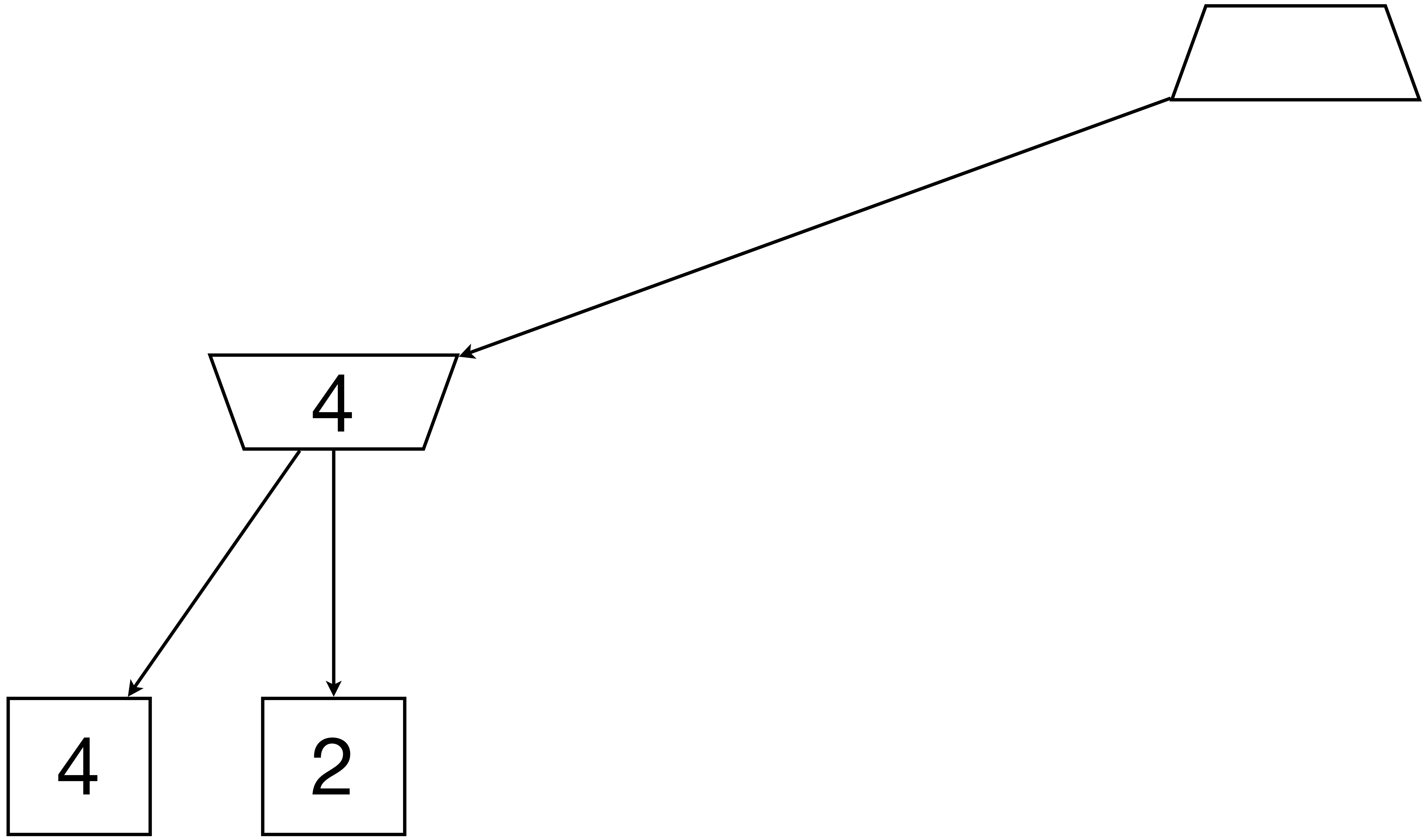
(recursive) thinking game: what if my/opp move is …



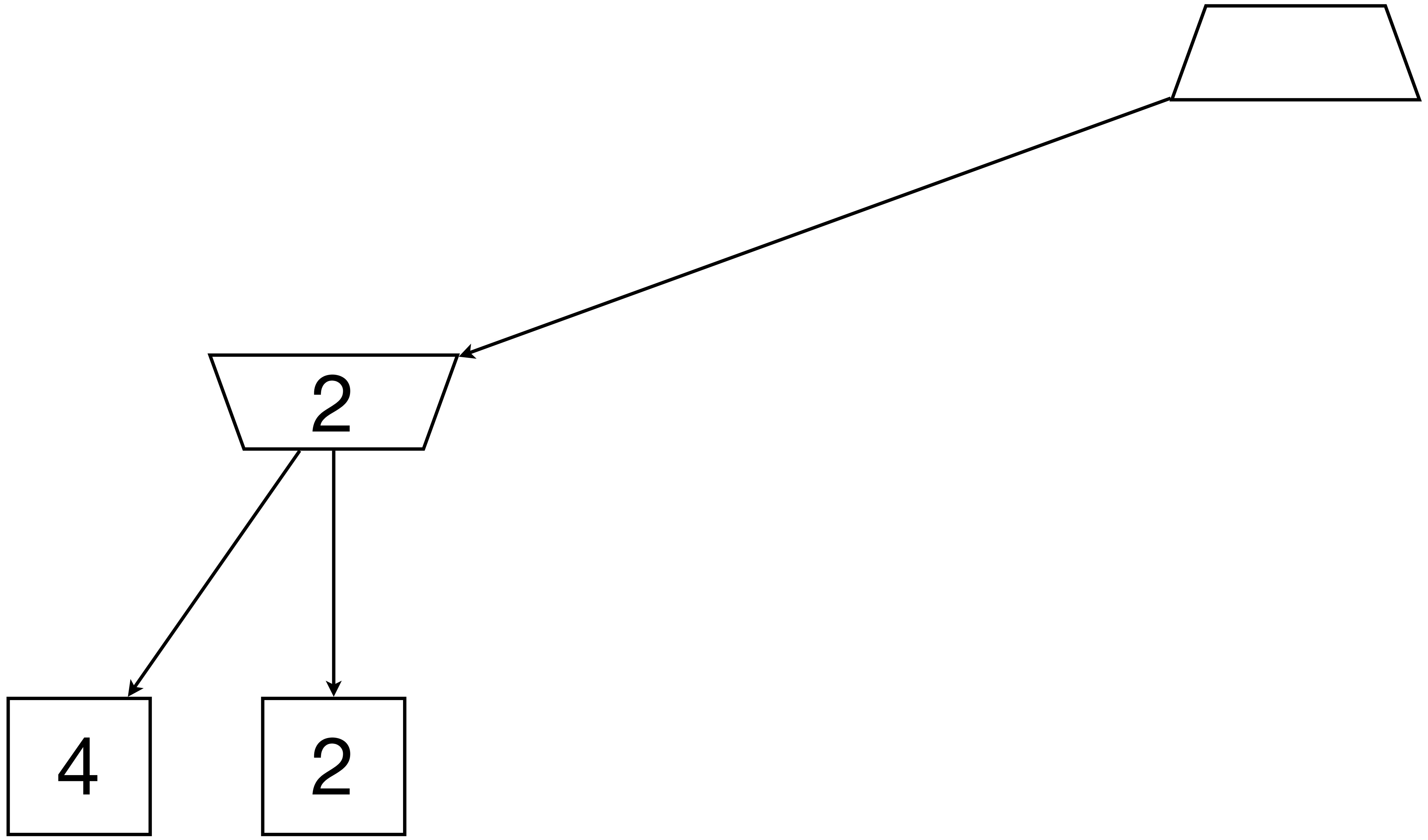
(recursive) thinking game: what if my/opp move is …



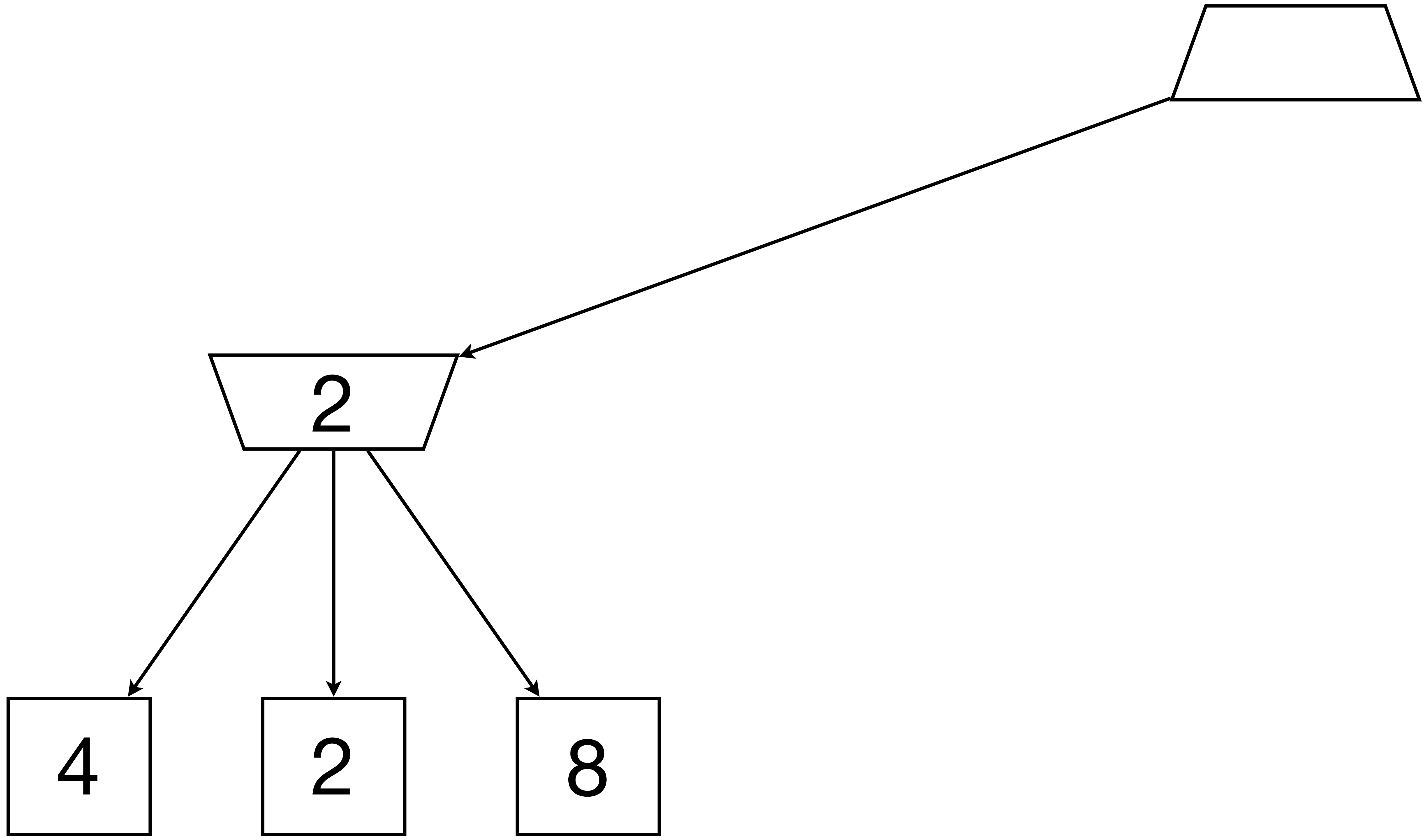
(recursive) thinking game: what if my/opp move is …



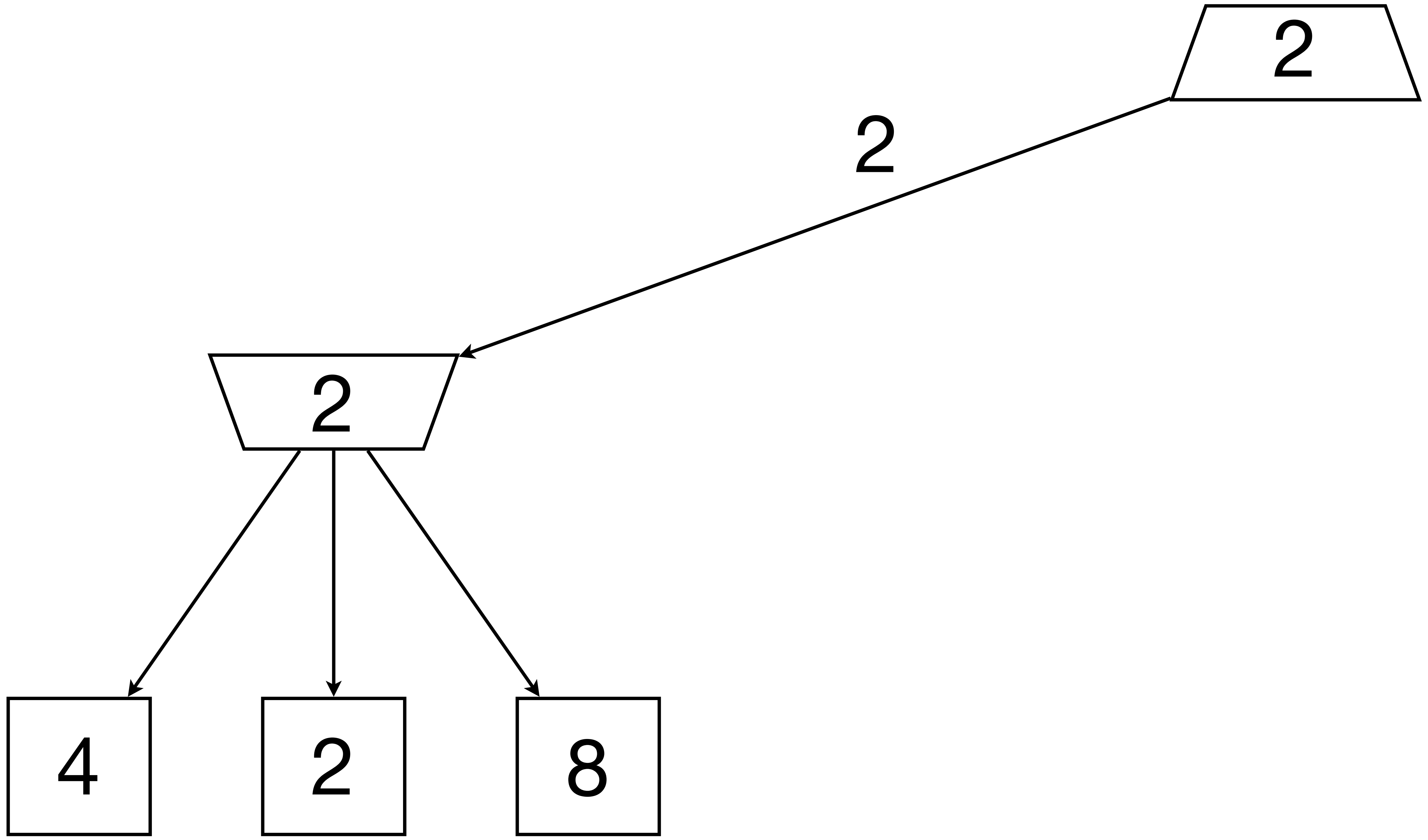
(recursive) thinking game: what if my/opp move is …



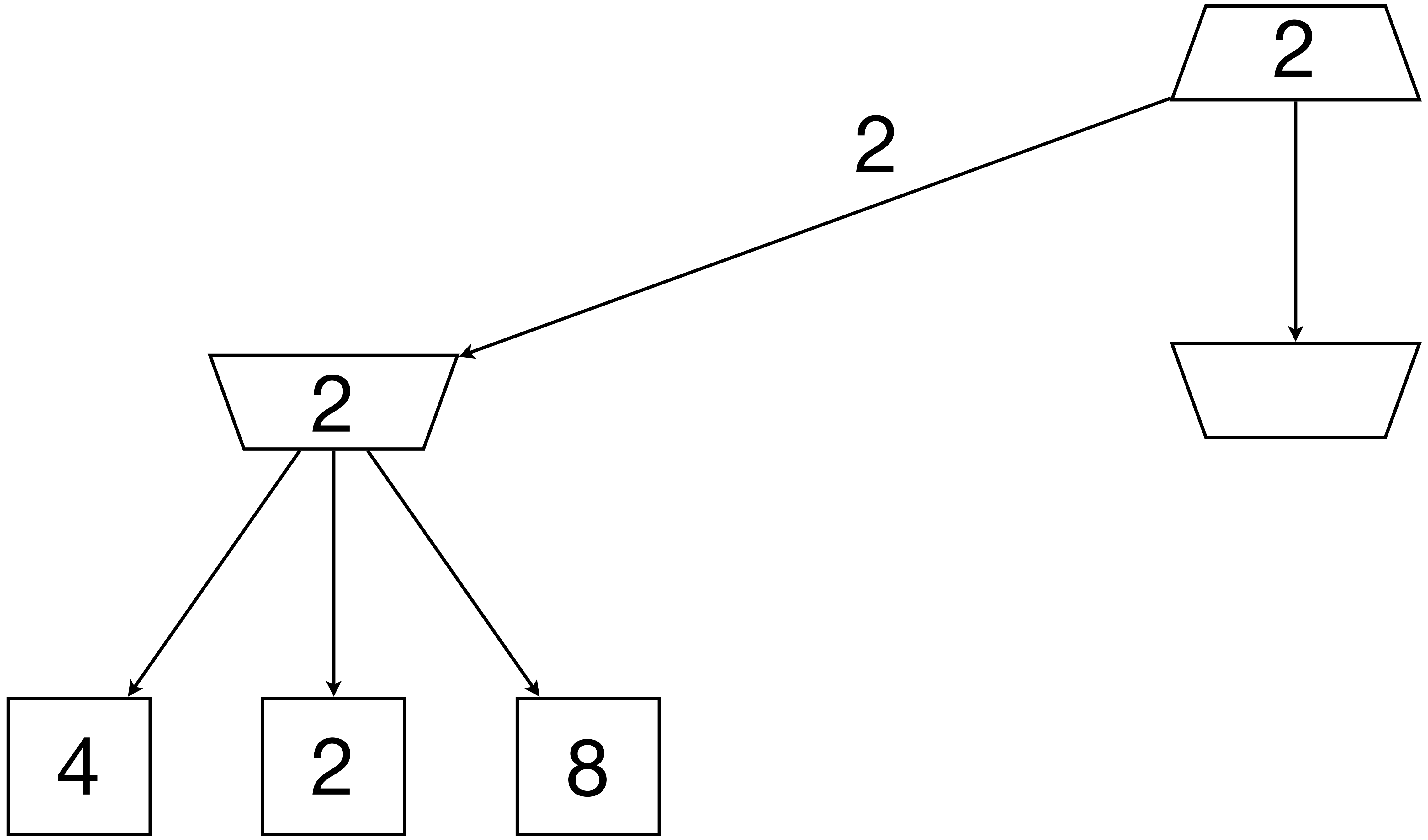
(recursive) thinking game: what if my/opp move is …



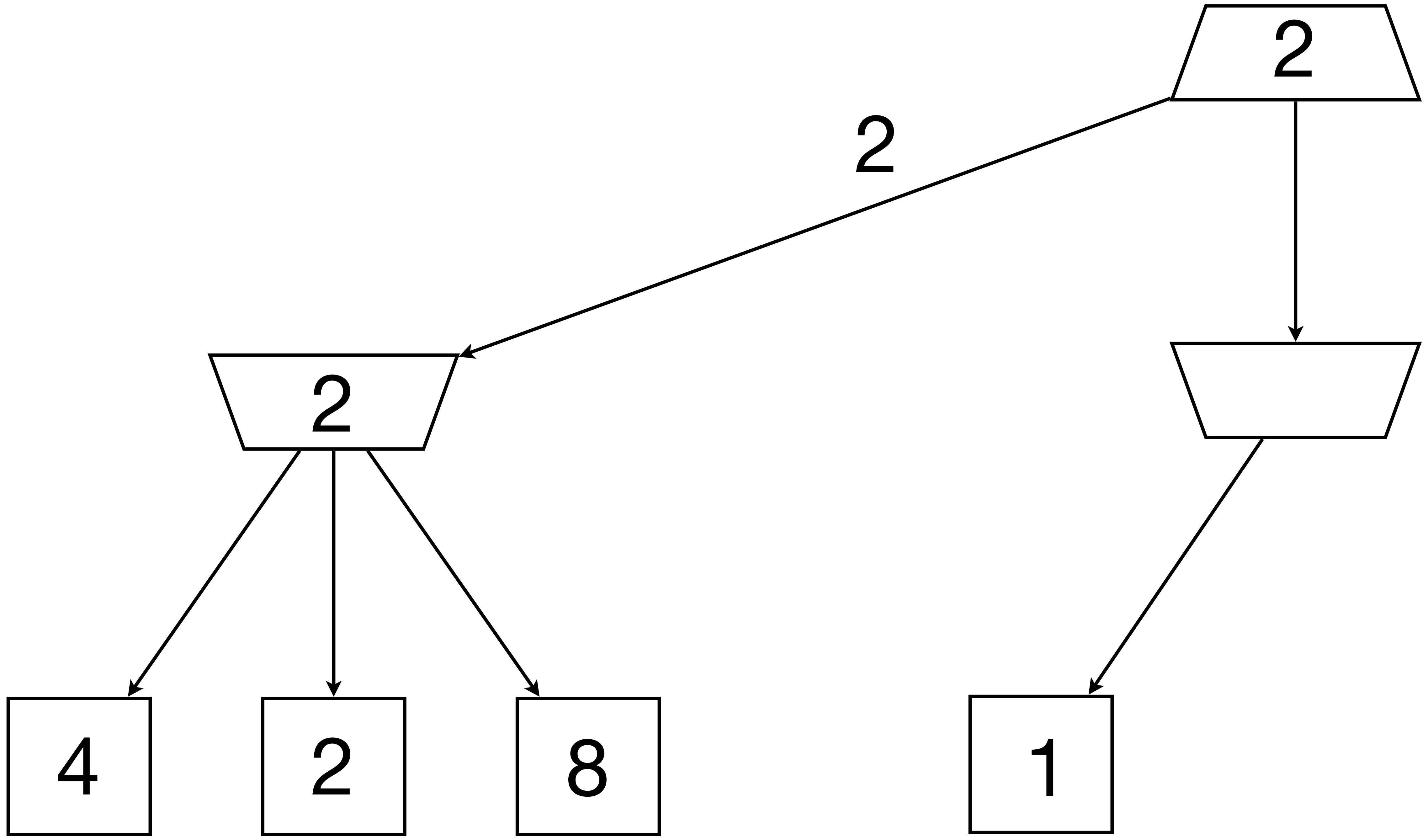
(recursive) thinking game: what if my/opp move is …



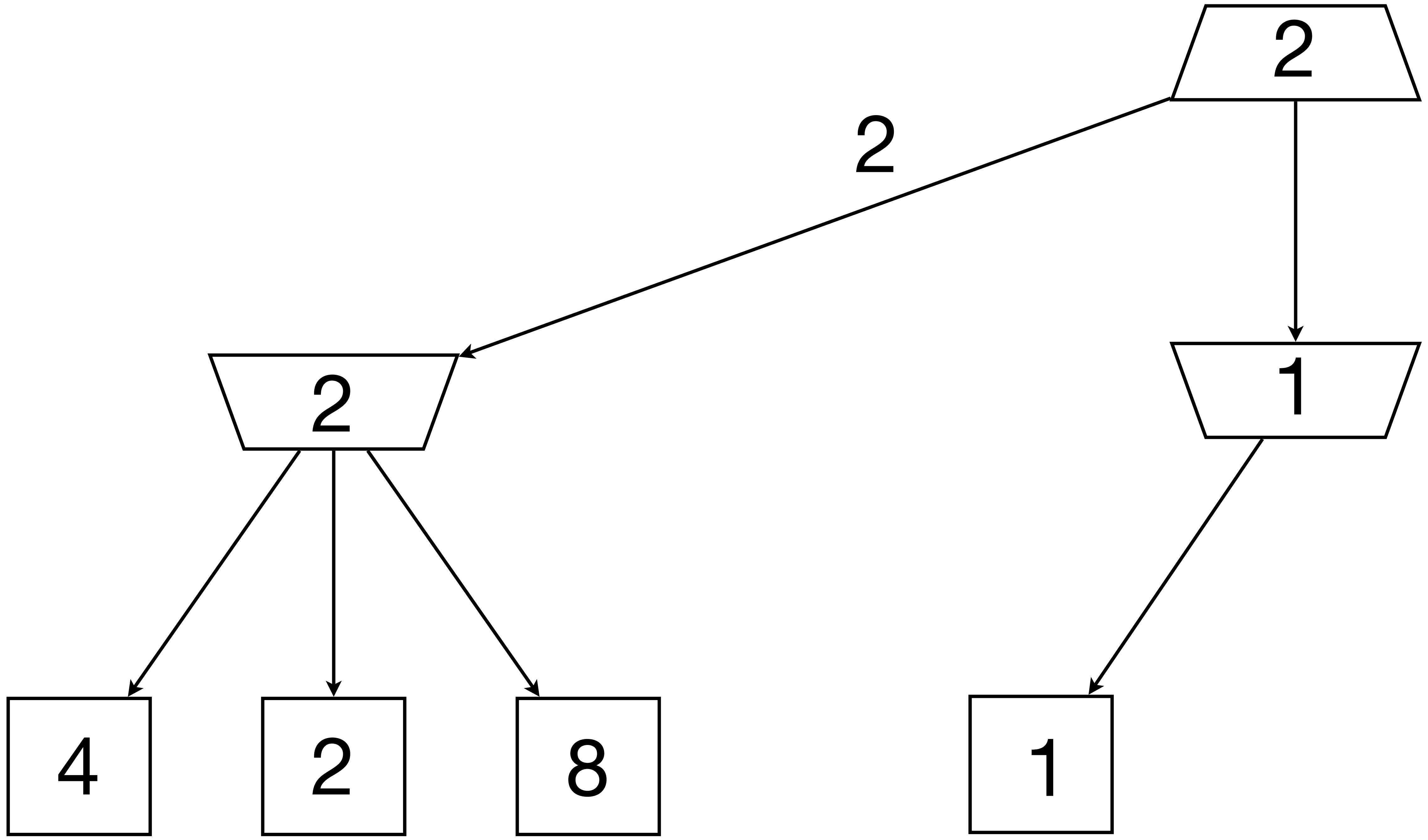
(recursive) thinking game: what if my/opp move is …



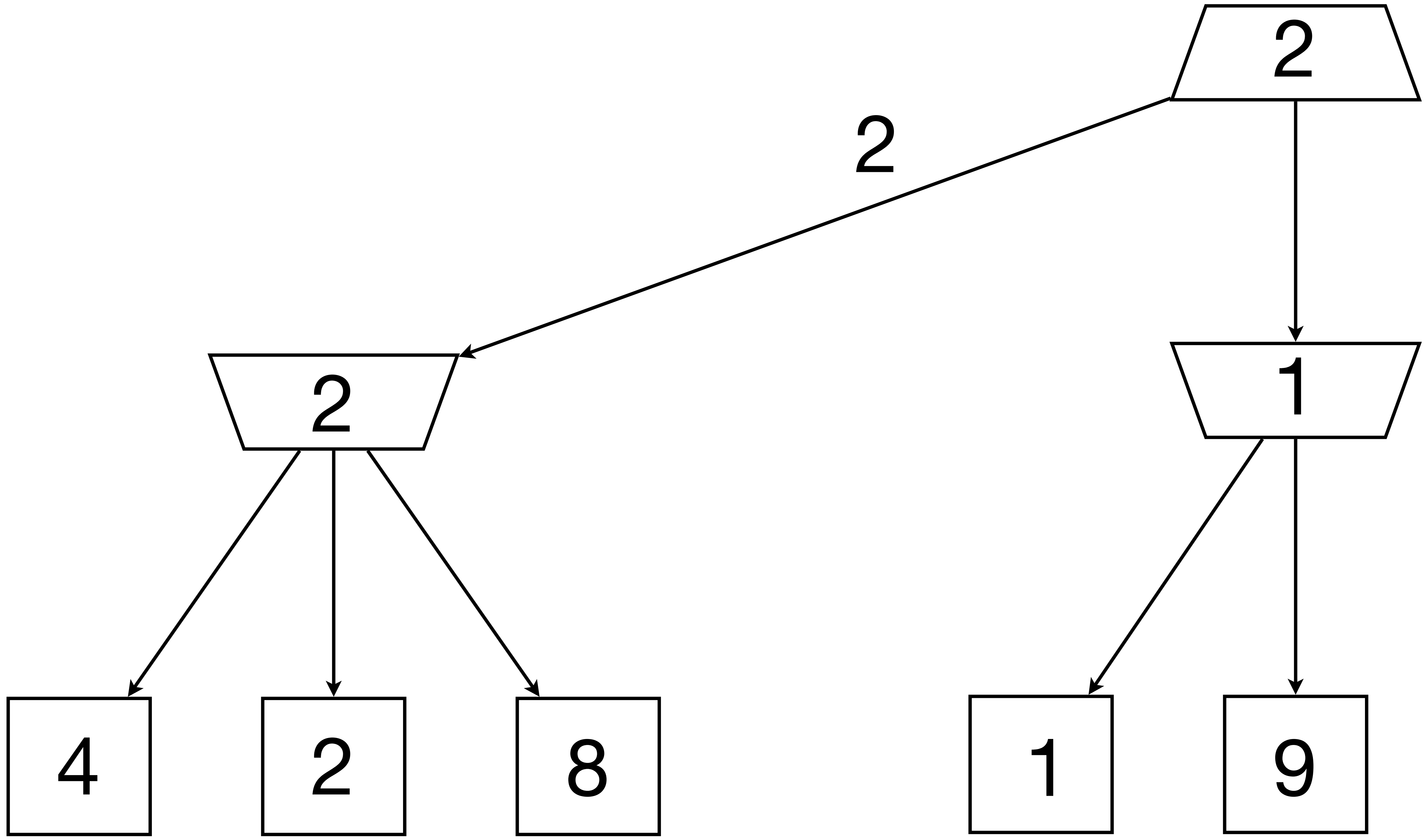
(recursive) thinking game: what if my/opp move is …



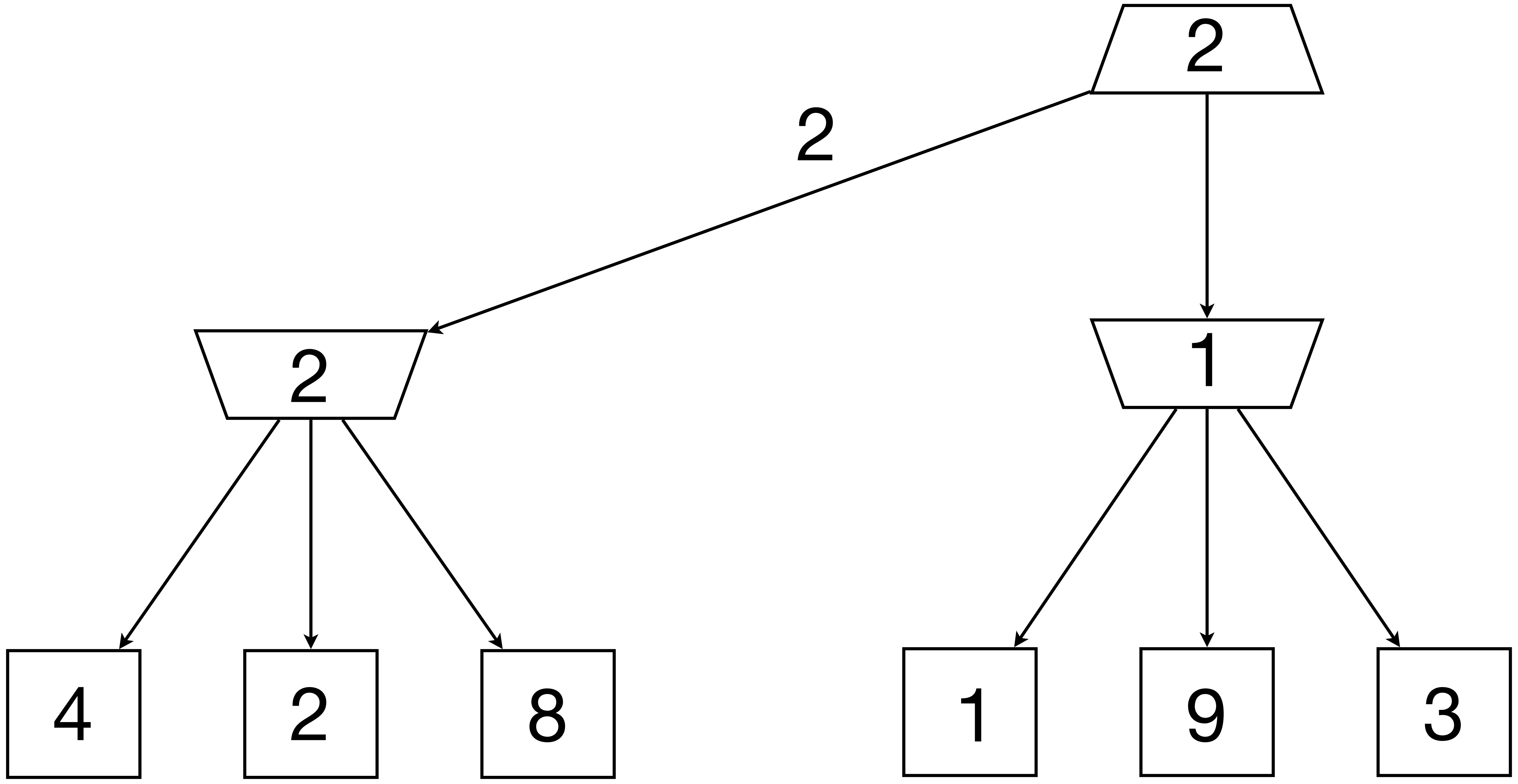
(recursive) thinking game: what if my/opp move is …



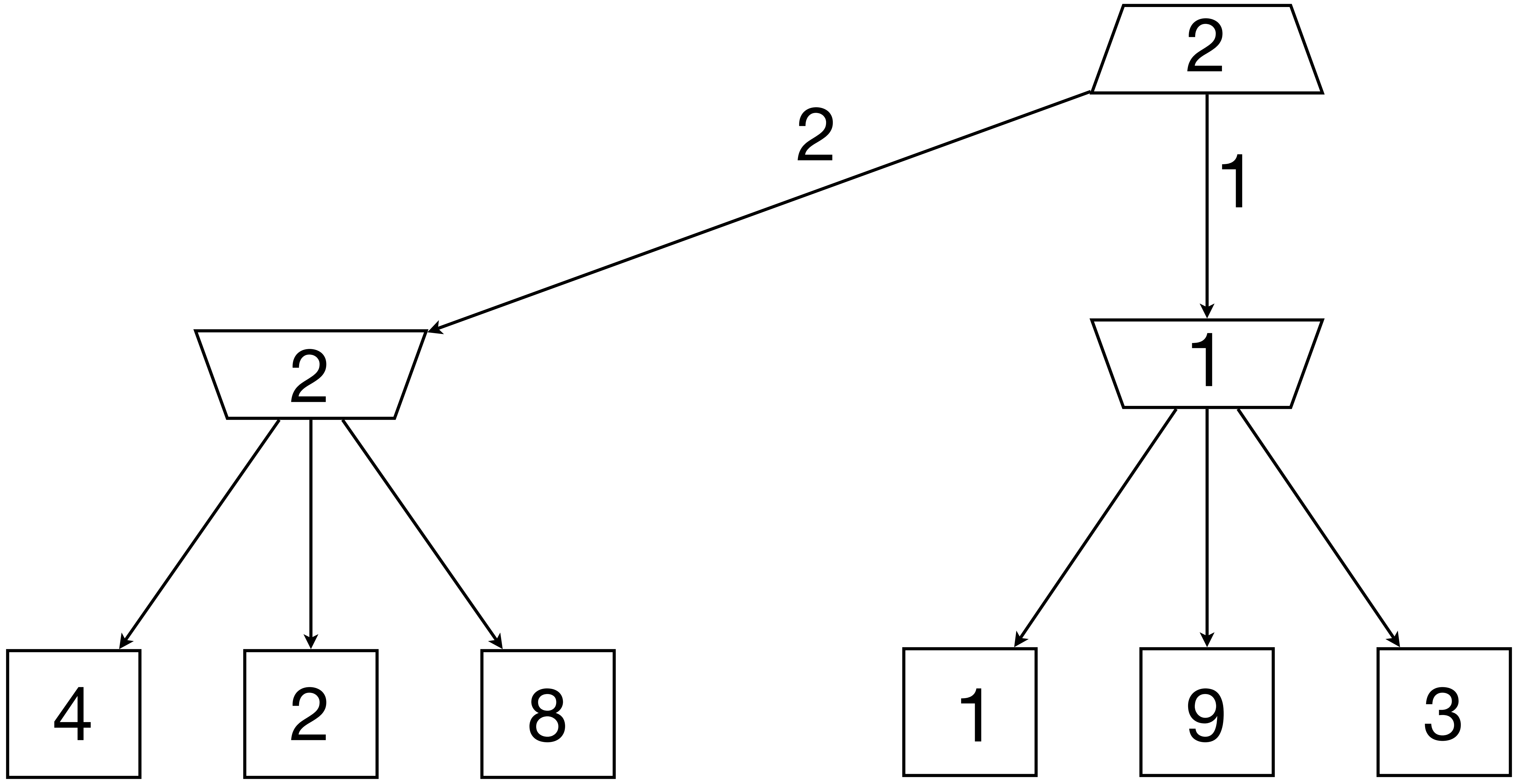
(recursive) thinking game: what if my/opp move is …



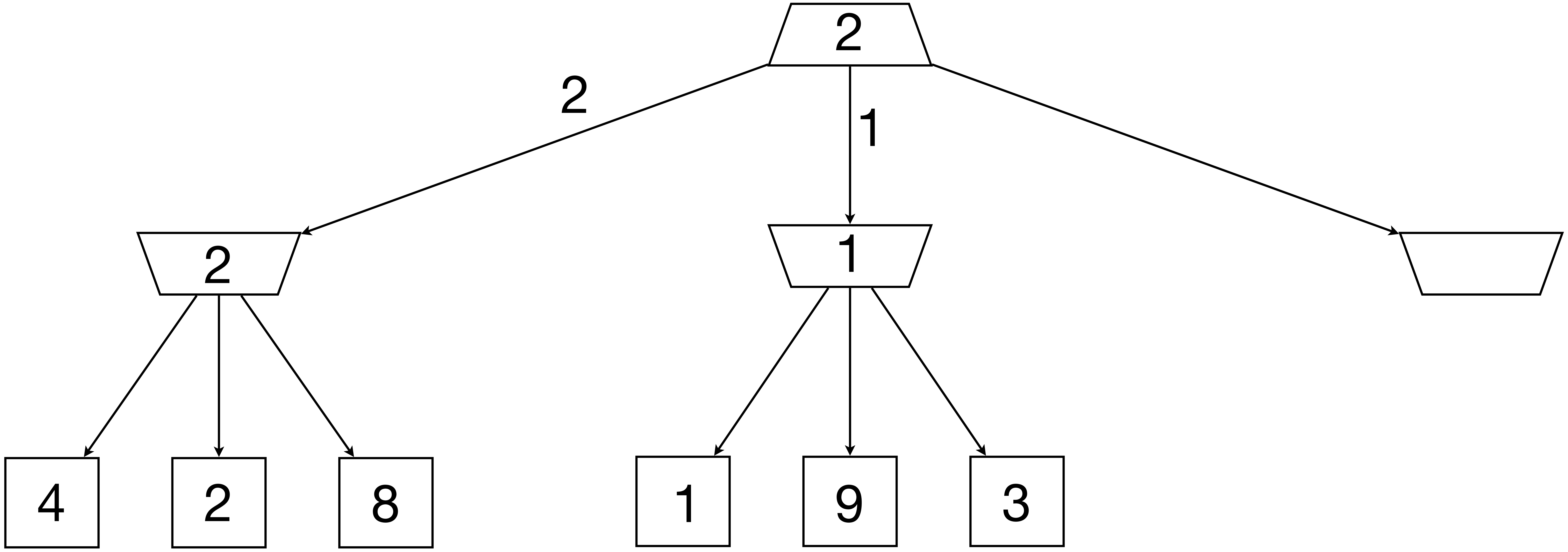
(recursive) thinking game: what if my/opp move is …



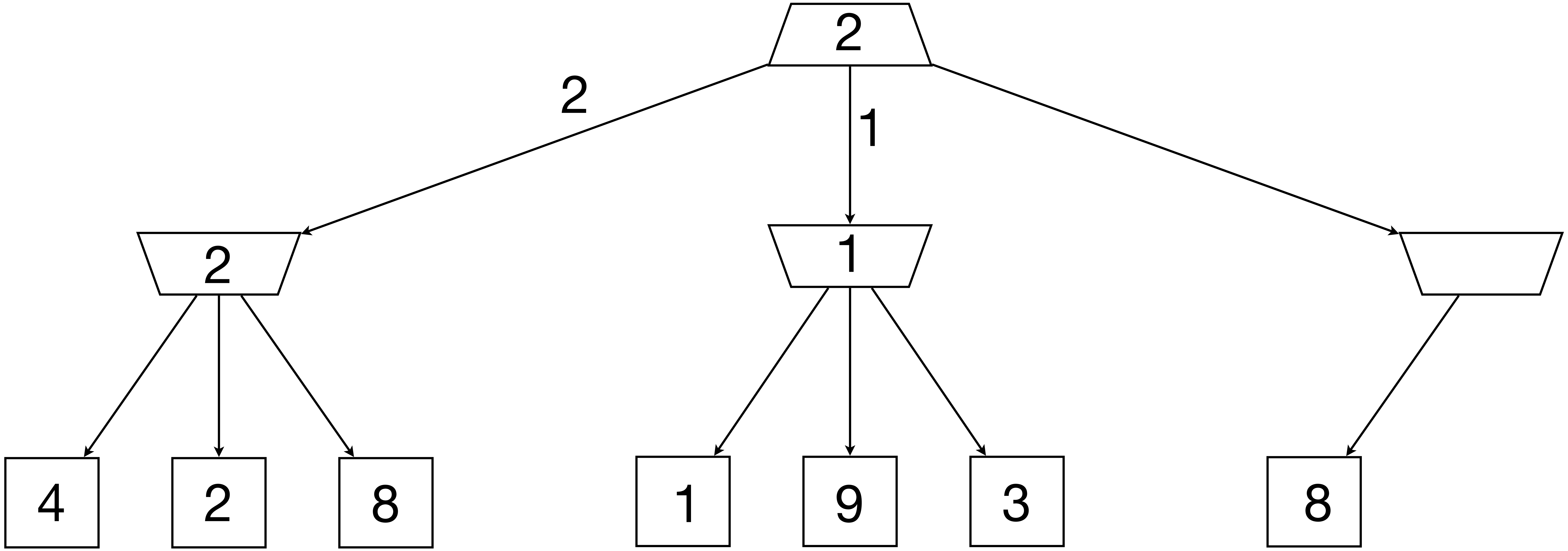
(recursive) thinking game: what if my/opp move is …



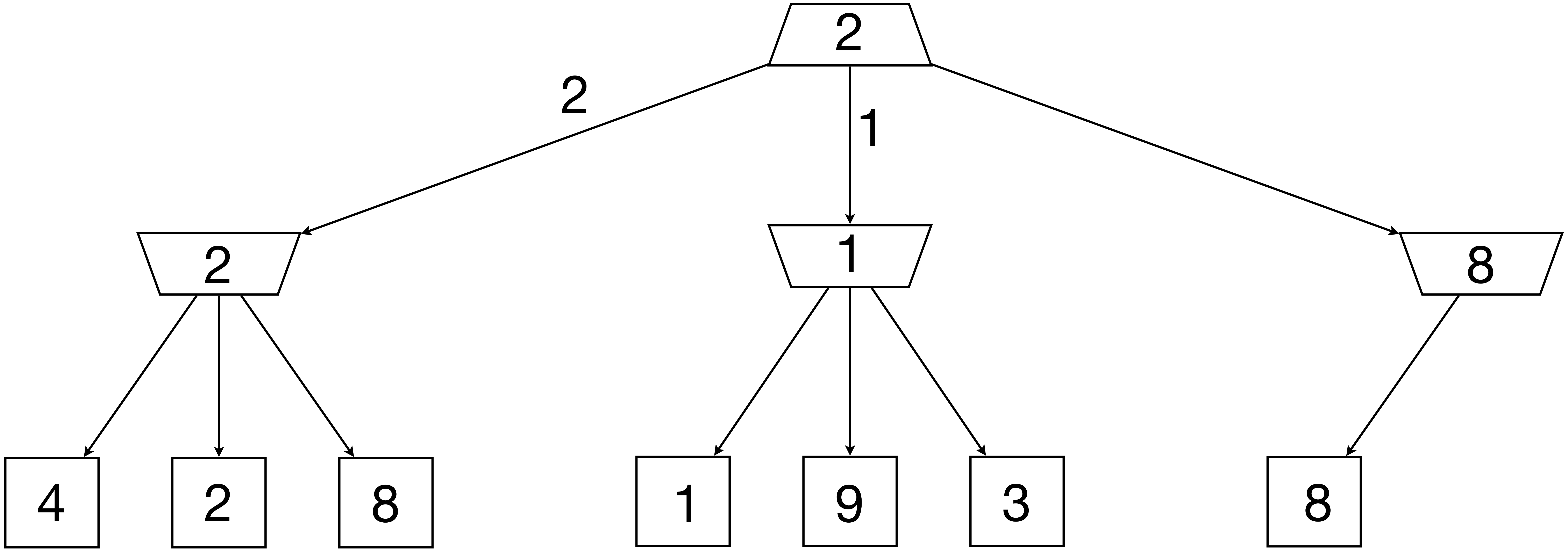
(recursive) thinking game: what if my/opp move is …



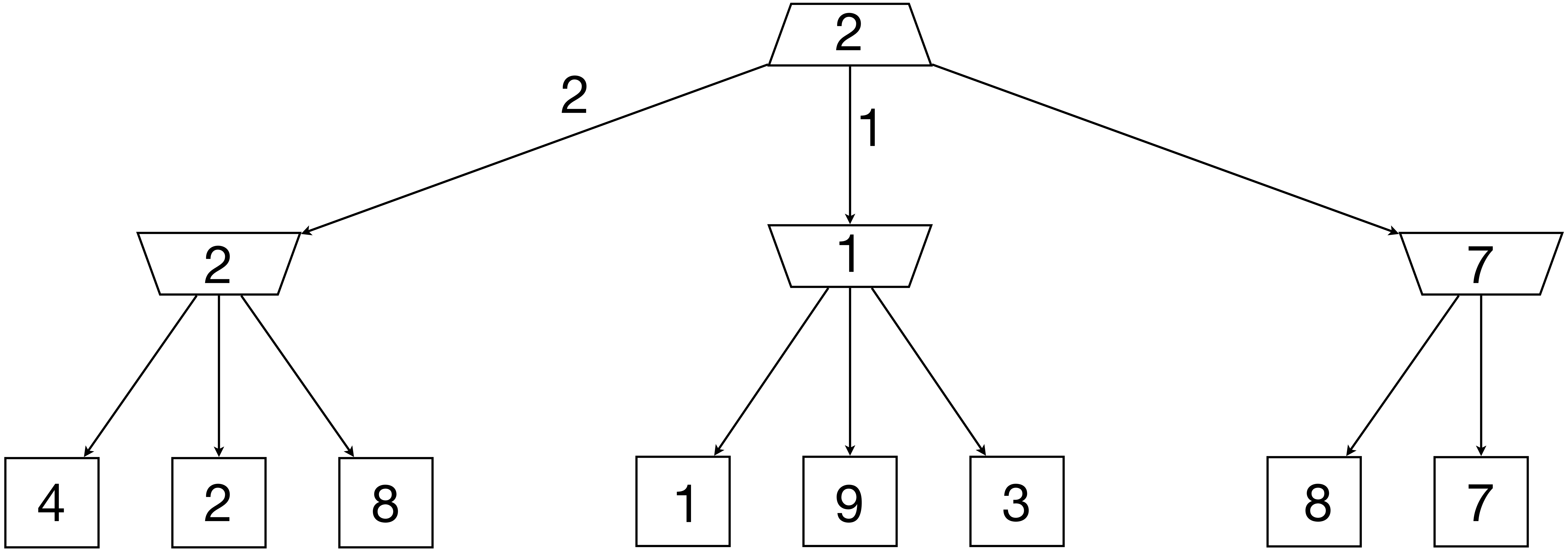
(recursive) thinking game: what if my/opp move is …



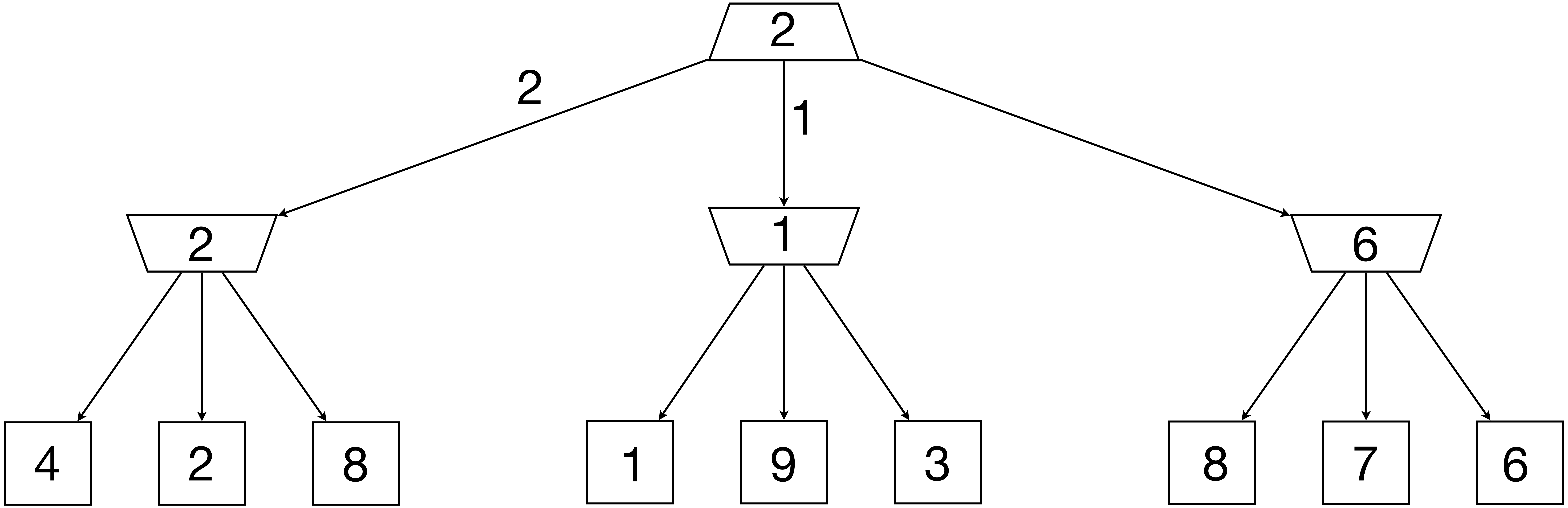
(recursive) thinking game: what if my/opp move is …



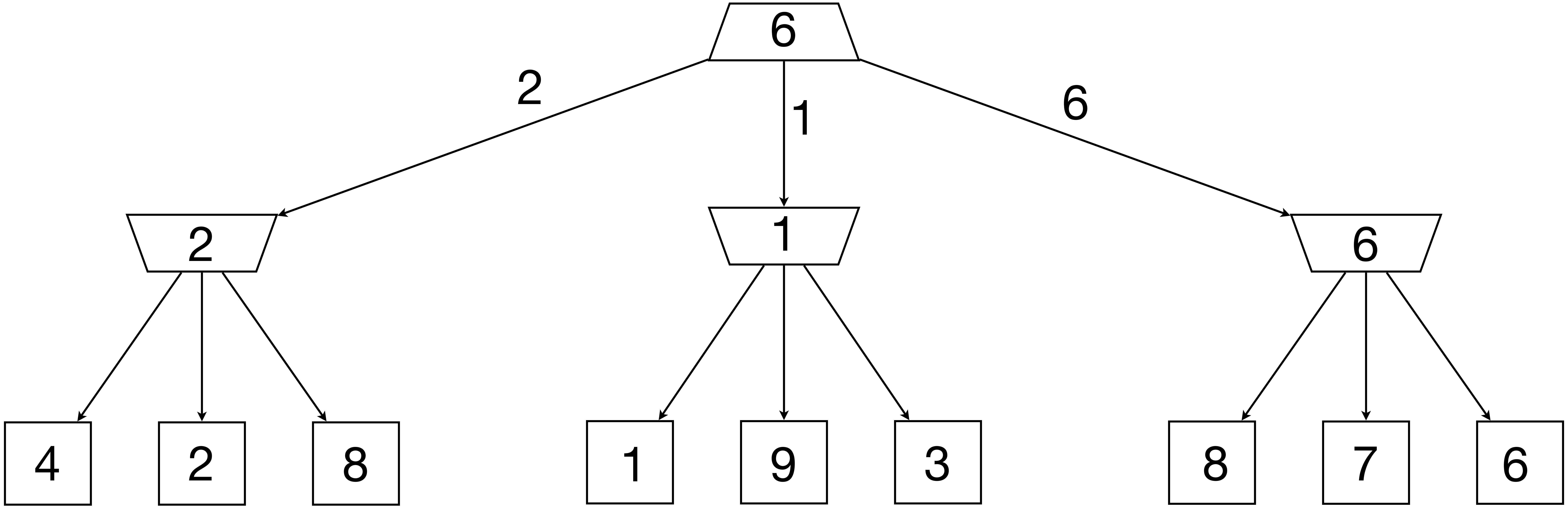
(recursive) thinking game: what if my/opp move is …



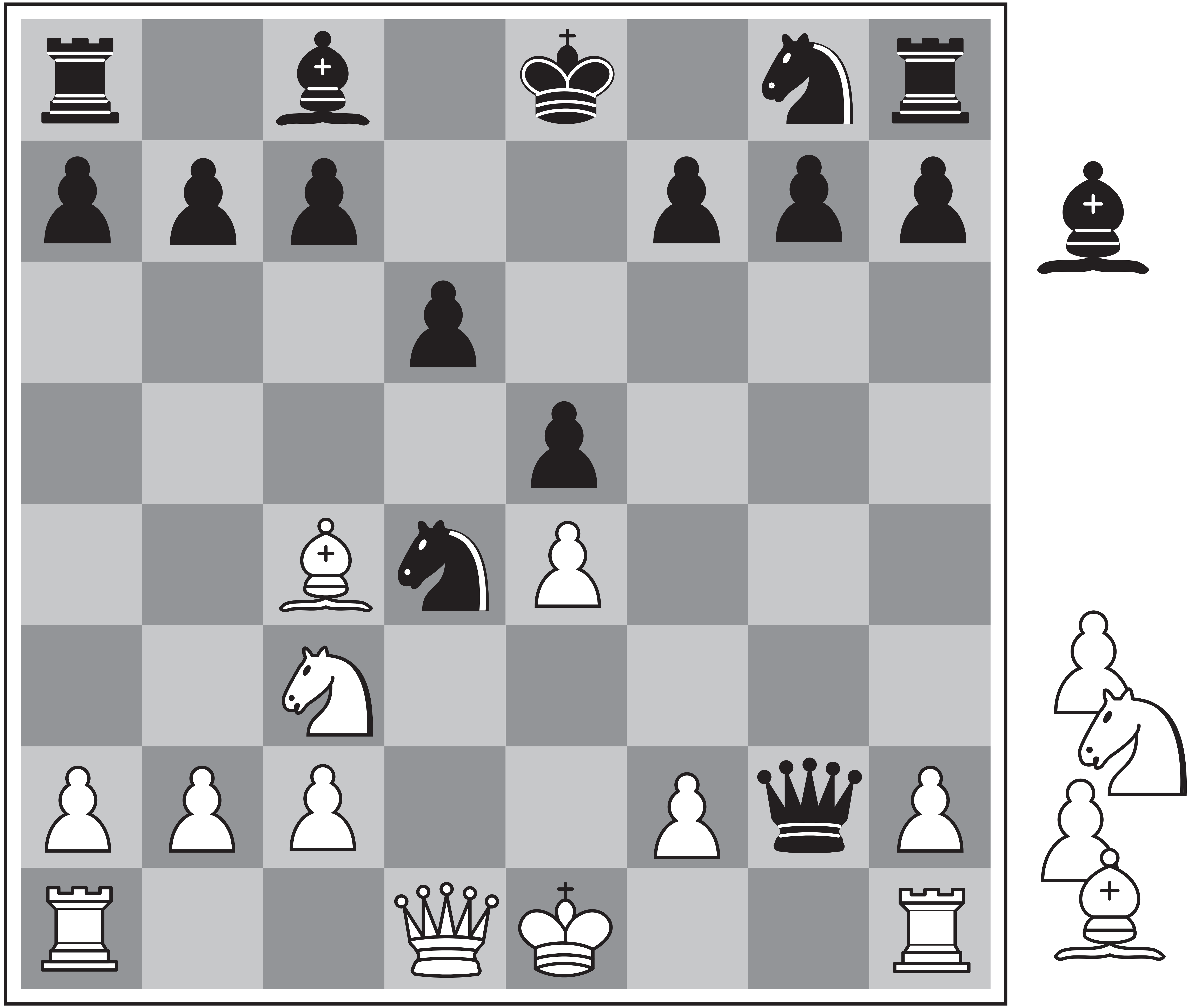
(recursive) thinking game: what if my/opp move is …



(recursive) thinking game: what if my/opp move is …



Eval(state)



- Uncertain outcome of an action.
- Robot/Agent may not know the current state!

What state (disease) given some observation (symptoms)?

$$P(disease|symptoms) = \frac{P(symptoms|disease) \times P(disease)}{P(symptoms)}$$

$$posterior = \frac{likelihood \times prior}{evidence}$$

- For each of the 9 possible situations (3 possible decisions $\times$ 3 possible states), the cost is quantified by a loss function $l(d, s)$:

$l(s, d)$	$d =$ *nothing*	$d =$ *pizza*	$d =$ *g.T.c.*
$s =$ *good*	0	2	4
$s =$ *average*	5	3	5
$s =$ *bad*	10	9	6

The wife's state of mind is an uncertain state.

$$P(x, s) = P(s|x)P(x)$$

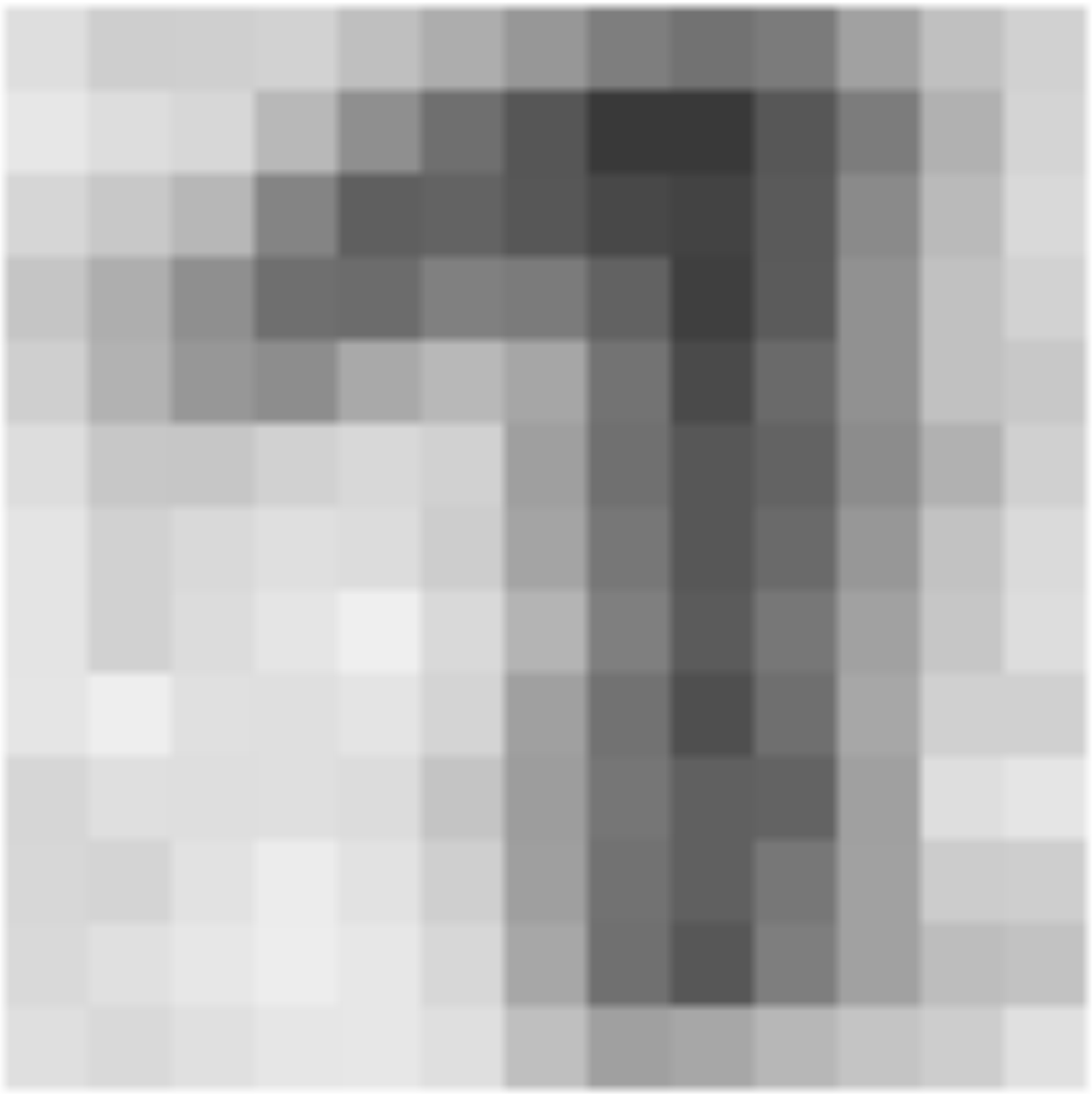
$P(x, s)$	$x =$ *mild*	$x =$*irritated*	$x =$ *upset*	$x =$ *alarming*
$s =$ *good*	0.35	0.28	0.07	0.00
$s =$ *average*	0.04	0.10	0.04	0.02
$s =$ *bad*	0.00	0.02	0.05	0.03

$$\delta^*(x) = \arg\min_d \sum_s l(s, d)P(s|x)$$

$P(x, s) = P(s|x)P(x)$

$\delta(x)$	$x =$ *mild*	$x =$ *irritated*	$x =$ *upset*	$x =$ *alarming*
$\delta_1(x) =$	*nothing*	*nothing*	*pizza*	*g.T.c.*
$\delta_2(x) =$	*nothing*	*pizza*	*g.T.c.*	*g.T.c.*
$\delta_3(x) =$	*g.T.c.*	*g.T.c.*	*g.T.c.*	*g.T.c.*
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$

Classification as a special case of statistical decision theory



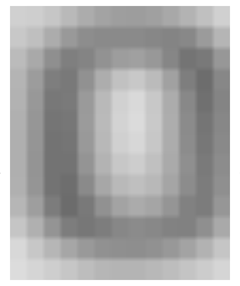
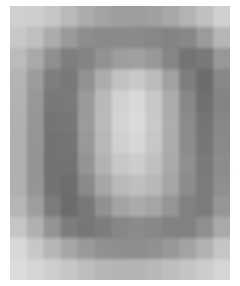
- Attribute vector $\vec{x} = [x_1, x_2, \dots]^\top$: pixels 1, 2, ...
- **State set $\mathcal{S}$ = decision set $\mathcal{D} = \{0, 1, \dots 9\}$.**
- State = actual class, Decision = recognized class
- Loss function: $l(s,d) = \begin{cases} 0, & d = s \\ 1, & d \neq s \end{cases}$

Optimal decision strategy:

$$\delta^*(\vec{x}) = \arg\min_d \sum_s \underbrace{l(s,d)}_{0 \text{ if } d=s} P(s|\vec{x}) = \arg\min_d \sum_{s \neq d} P(s|\vec{x})$$

Obviously $\sum_s P(s|\vec{x}) = 1$, then: $P(d|\vec{x}) + \sum_{s\neq d} P(s|\vec{x}) = 1$

Inserting into above:

$\delta^*($  $) = \arg\max_d P(d|$  $)$

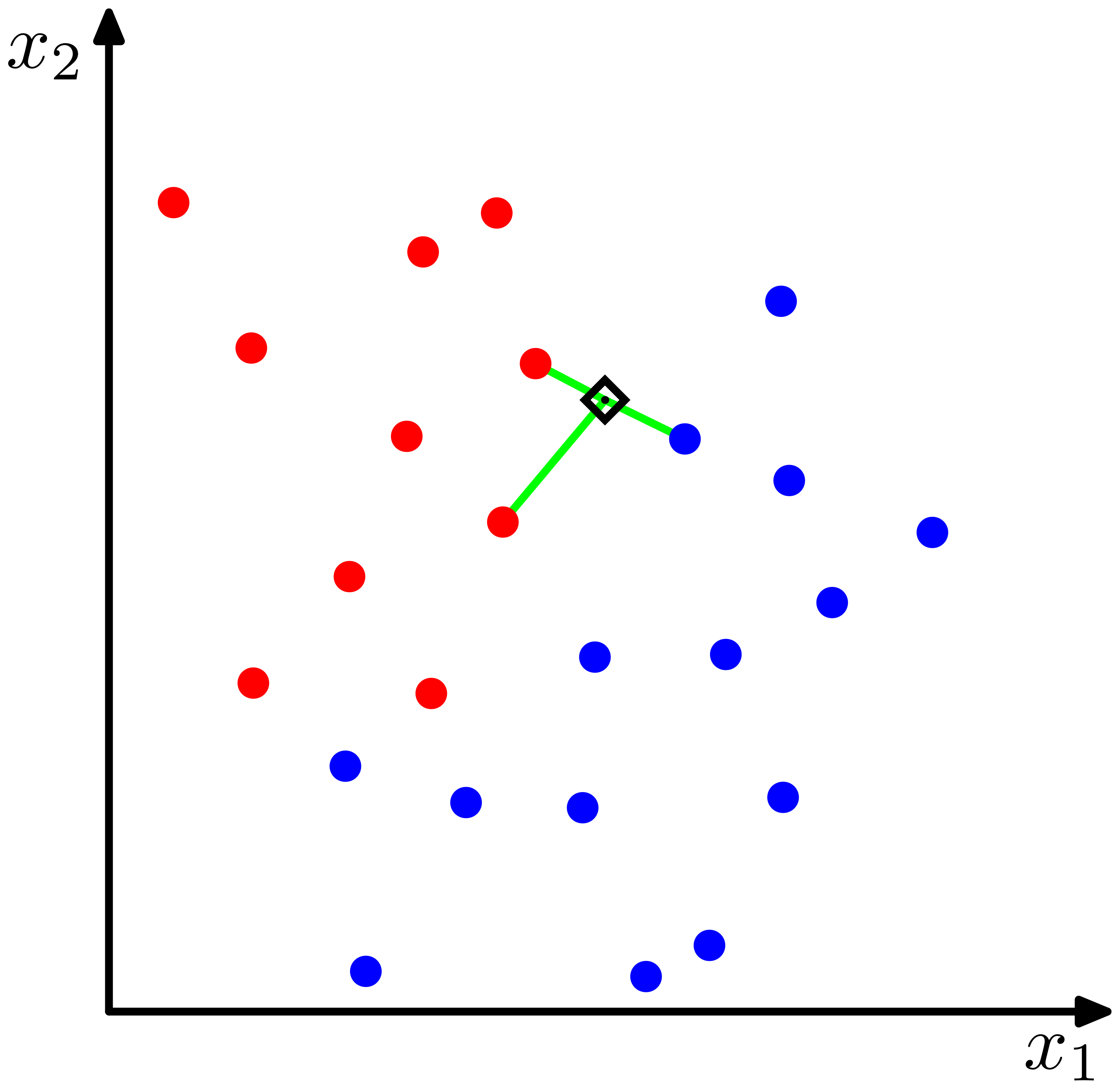
$$\delta^*(\vec{x}) = \arg\min_d \Big(1 - P(d|\vec{x})\Big) = \arg\max_d P(d|\vec{x})$$

$K-$ Nearest Neighbor and Bayes $j^* = \text{argmax}_j P(s_j|\mathbf{x})$

Assume data:

- N points $\mathbf{x}$ in total.
- N_j points in s_j class. Hence, $\sum_j N_j = N$.

We want to classify $\mathbf{x}$. Draw a sphere centered at $\mathbf{x}$ containing K points irrespective of class. V is the volume of this sphere. $P(s_j|\mathbf{x}) = ?$



$$P(s_j|\mathbf{x}) = \frac{P(\mathbf{x}|s_j)P(s_j)}{P(\mathbf{x})}$$

K_j is the number of points of class s_j among the K nearest neighbors.

$$P(s_j) = \frac{N_j}{N}$$

$$P(\mathbf{x}) = \frac{K}{NV}$$

$$P(\mathbf{x}|s_j) = \frac{K_j}{N_j V}$$

$$P(s_j|\mathbf{x}) = \frac{P(\mathbf{x}|s_j)P(s_j)}{P(\mathbf{x})} = \frac{K_j}{K}$$

- Usually, we are not given $P(s|\vec{x})$
- It has to be estimated from already classified examples – training data
- For discrete $\vec{x}$, training examples $(\vec{x}_1, s_1), (\vec{x}_2, s_2), \ldots (\vec{x}_l, s_l)$
  - every $(\vec{x}_i, s)$ is drawn independently from $P(\vec{x}, s)$, i.e. sample i does not depend on $1, \cdots, i-1$
  - so-called i.i.d (independent, identically distributed) multiset
- Without knowing anything about the distribution, a non-parametric estimate:

$$P(s|\vec{x}) = \frac{P(\vec{x}, s)}{P(\vec{x})} \approx \frac{\#\text{ examples where } \vec{x}_i = \vec{x} \textbf{ and } s_i = s}{\#\text{ examples where } \vec{x}_i = \vec{x}}$$

- In the exceptional case of statistical independence between components of $\vec{x}$ for each class s it holds

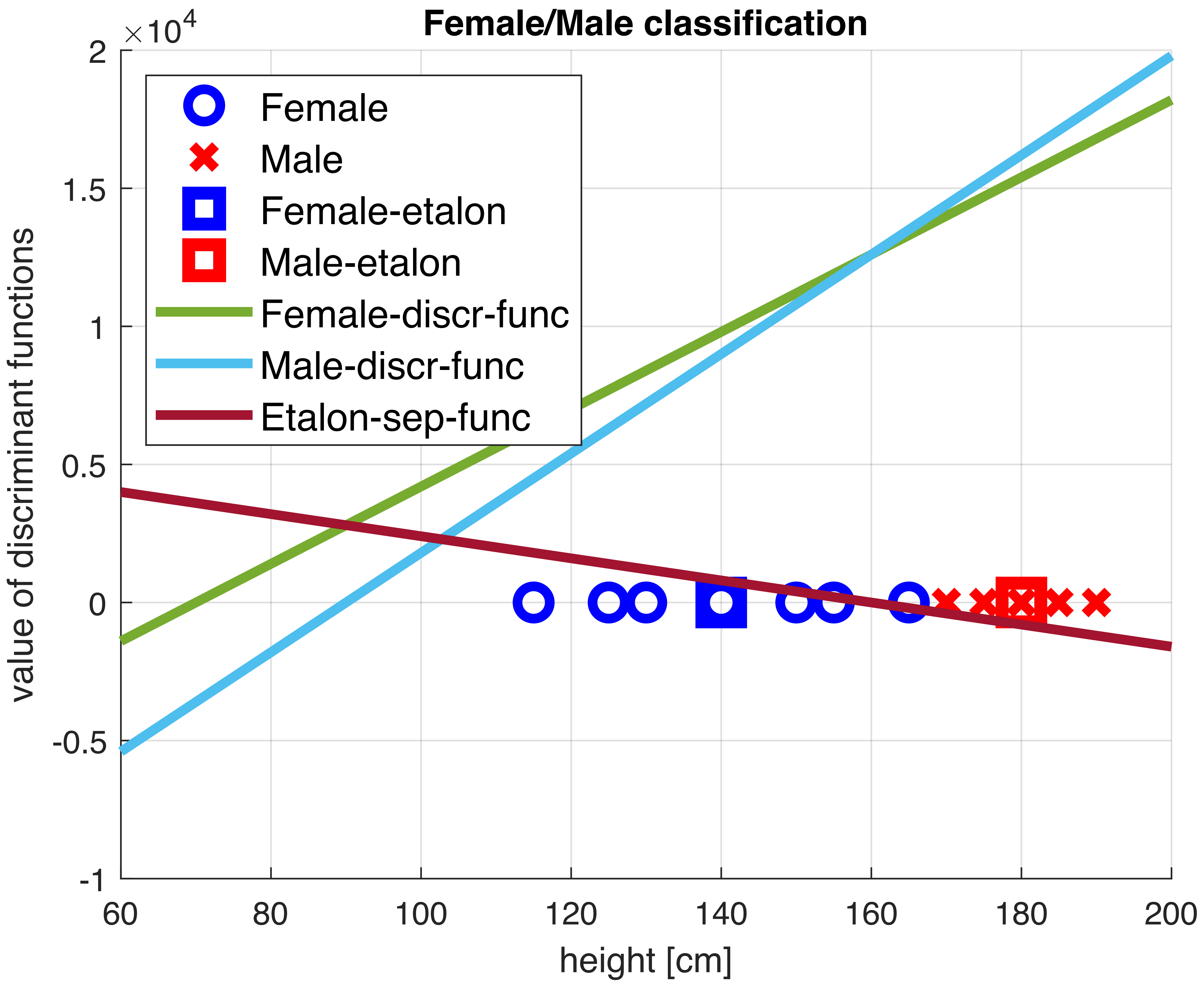
$$P(\vec{x}|s) = P(x[1]|s) \cdot P(x[2]|s) \cdot \ldots$$

- Use simple Bayes law and maximize:

$$P(s|\vec{x}) = \frac{P(\vec{x}|s)P(s)}{P(\vec{x})} = \frac{P(s)}{P(\vec{x})} P(x[1]|s) \cdot P(x[2]|s) \cdot \ldots =$$

Discriminant functions

$$\delta(\mathbf{x}) = \operatorname{argmax}_{s \in S} f_s(\mathbf{x})$$



Discriminant functions for 2 classes:

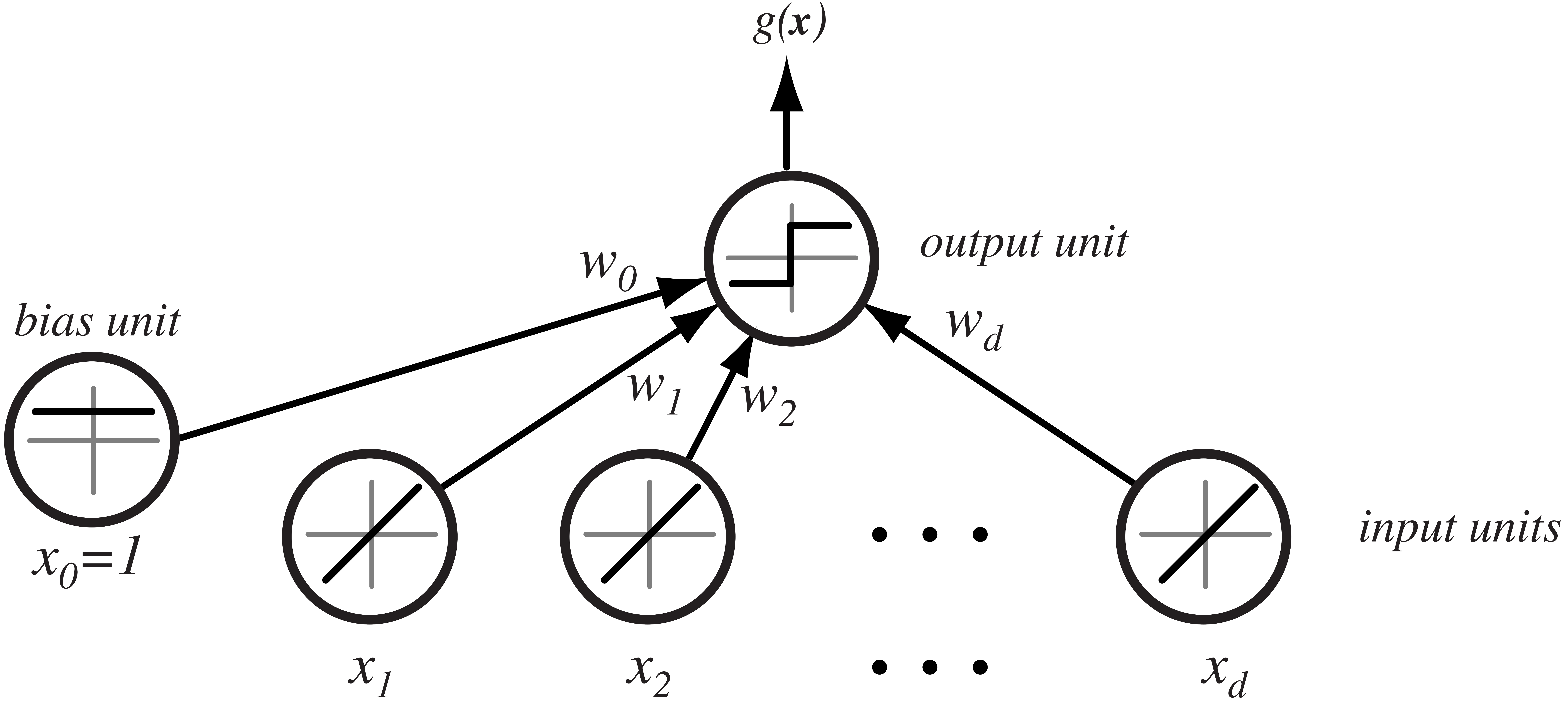
$$f_F(x) = a_F x + b_F = $$
$$= e_F x - \frac{1}{2} e_F^2 = 140x - 9800$$
$$f_M(x) = a_M x + b_M = $$
$$= e_M x - \frac{1}{2} e_M^2 = 180x - 16200$$

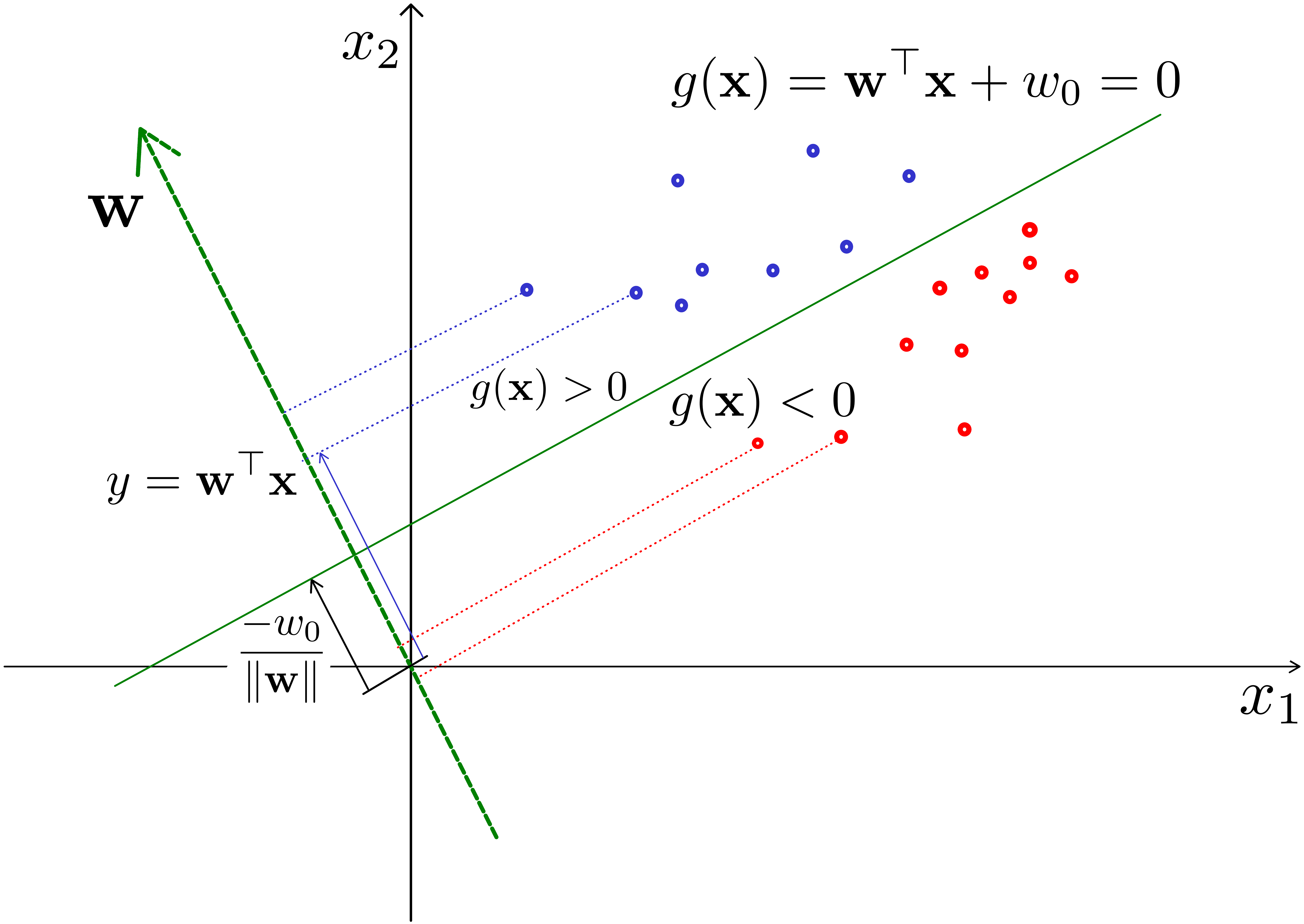
A single discriminant function separating 2 classes:

$$g(x) = f_F(x) - f_M(x) = $$
$$= -40x + 6400$$

$$g(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} + w_0$$

Decide s_1 if $g(\mathbf{x}) > 0$ and s_2 if $g(\mathbf{x}) < 0$





x_2
$g(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} + w_0 = 0$
$\mathbf{w}$
$g(\mathbf{x}) > 0$
$g(\mathbf{x}) < 0$
$y = \mathbf{w}^\top \mathbf{x}$
$\frac{-w_0}{\|\mathbf{w}\|}$
x_1

Gradient descent

Initialize $\mathbf{w}$, threshold θ, learning rate α

$k \leftarrow 0$

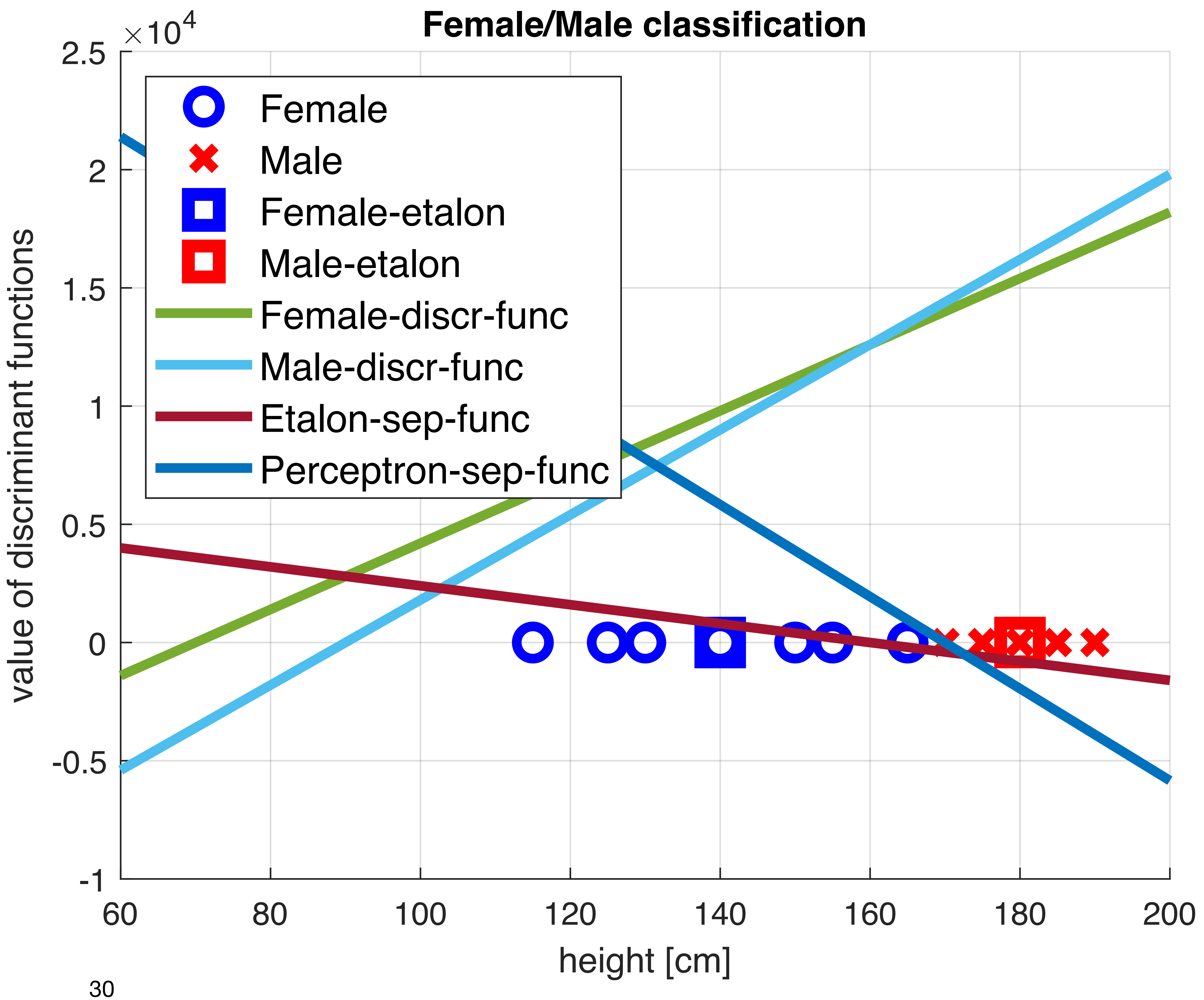
repeat

$$k \leftarrow k + 1$$

$$\mathbf{w} \leftarrow \mathbf{w} - \alpha(k)\nabla J(\mathbf{w})$$

until $|\alpha(k)\nabla J(\mathbf{w})| < \theta$

return $\mathbf{w}$



What next?

- gradient descent, linear programming, … Optimization, B0B33OPT
- machine learning, classifiers, Bayesian and non-Bayesian decisions, … Pattern Recognition and Machine Learning (B4B33RPZ), Statistical Machine Learning (BE4M33SSU)
- machine learning pragmatically, deep nets Robot Learning (B3B33UROB)
- deeper in deep nets, Deep Learning, BEV033DLE
- perception, Computer Vision Methods, B4M33MPV
- planning, Artificial Intelligence in Robotics, B4M36UIR