To Reach or not to Reach? Efficient Algorithms for Total-Payoff Games

Thomas Brihaye (UMONS), Gilles Geeraerts (ULB), Axel Haddad (UMONS),
and **Benjamin Monmege (ULB – Aix-Marseille University)**

Madrid meet 2015 — CONCUR

Game theory for synthesis

Game theory for synthesis

- More and more complex systems: difficult to design

Game theory for synthesis

- More and more complex systems: difficult to design
- Rather than spending energy on verifying handmade code…
  Synthesise some code correct-by-construction!

Game theory for synthesis

- More and more complex systems: difficult to design
- Rather than spending energy on verifying handmade code…
  Synthesise some code correct-by-construction!
- Well-establish model for synthesis: games on graphs
  - 2 antagonistic players: controller and environment
  - objective: reachability, repeated reachability, LTL…

Game theory for synthesis

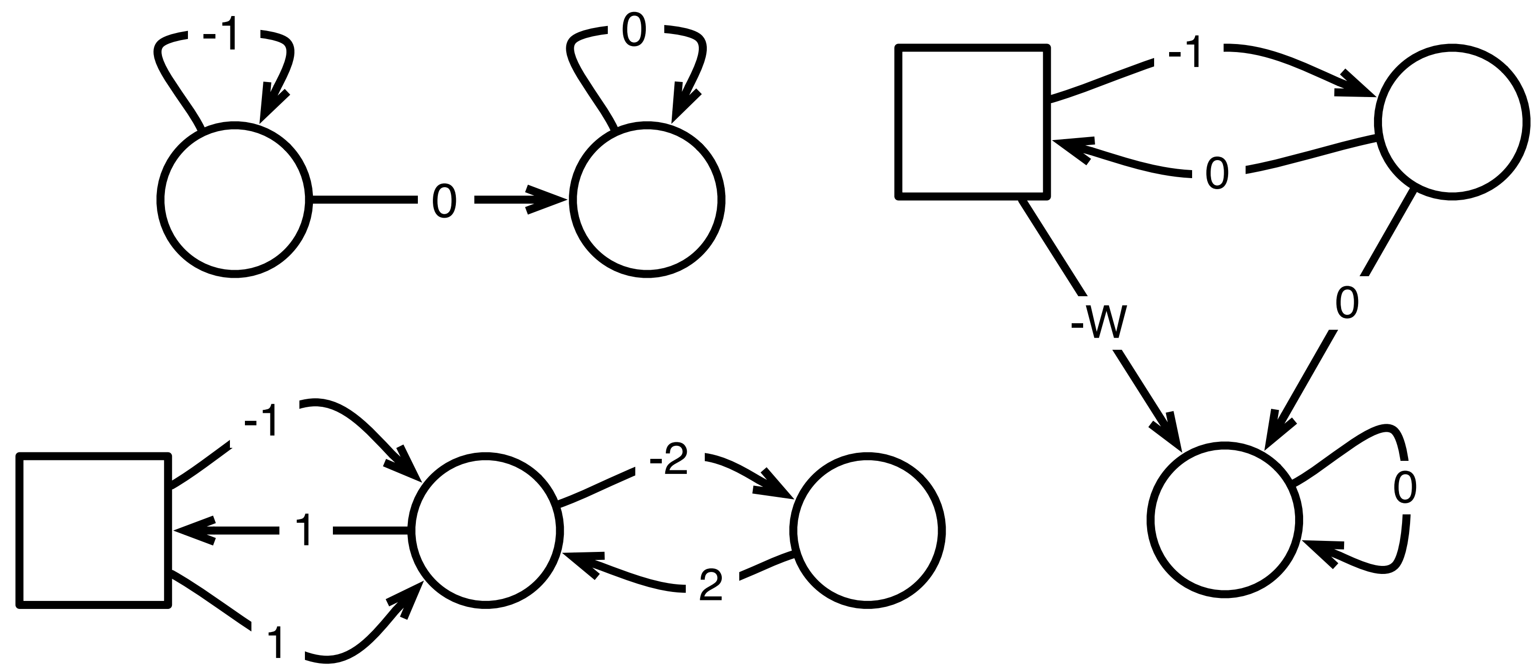
- More and more complex systems: difficult to design
- Rather than spending energy on verifying handmade code…
  Synthesise some code correct-by-construction!
- Well-establish model for synthesis: games on graphs
  - 2 antagonistic players: controller and environment
  - objective: reachability, repeated reachability, LTL…
- Interested with energy consumption, reliability, lifetime…
  Quantitative synthesis with games on weighted graphs

Games on weighted graphs

$$(V, E, w)$$

$$V = V_{\min} \uplus V_{\max}$$

$$w : E \to \mathbf{Z}$$

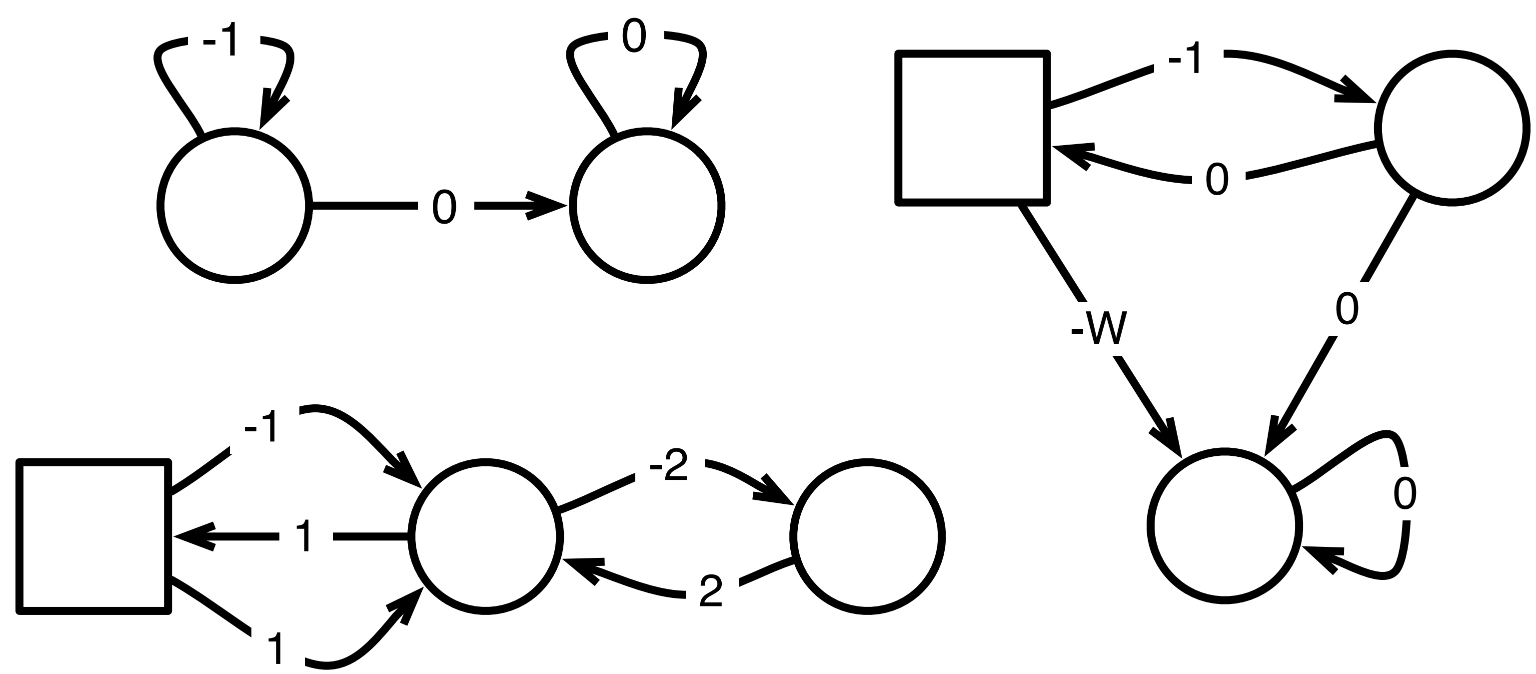


Games on weighted graphs

$$(V, E, w)$$

$$V = V_{\min} \uplus V_{\max}$$

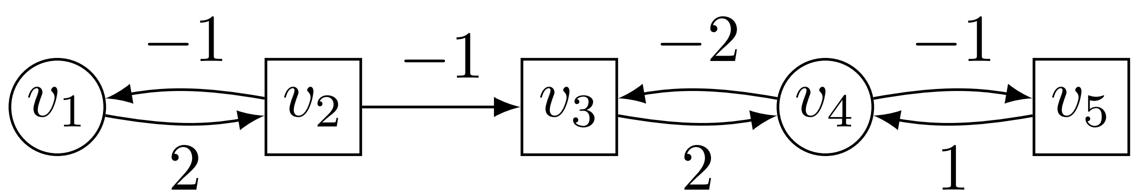
$$w : E \to \mathbf{Z}$$



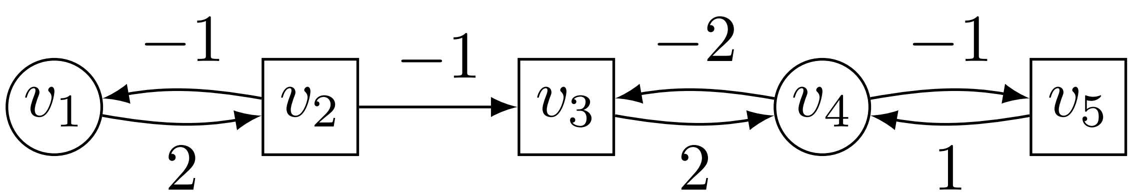
- Quantitative objective of the controller: maximising his payoff, accumulated along the computation of the system
    - Mean-payoff: *good in average*.

      Abundantly studied, NP∩co-NP, pseudo-polynomial time algorithm by Zwick & Paterson…
    - Total-payoff: *good in total*. Refinement of mean-payoff
    - Discounted-payoff…

Total-payoff games



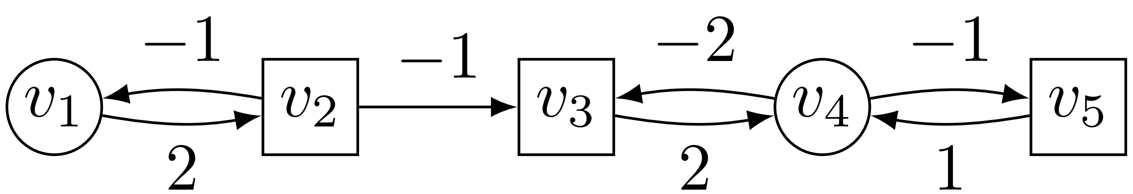
Total-payoff games



$$\pi = v_1 v_2 v_3 \ v_4 v_5 \ v_4 v_3 \ (v_4 v_5)^\omega$$

- Play: $\pi = v_0 v_1 v_2 \cdots$

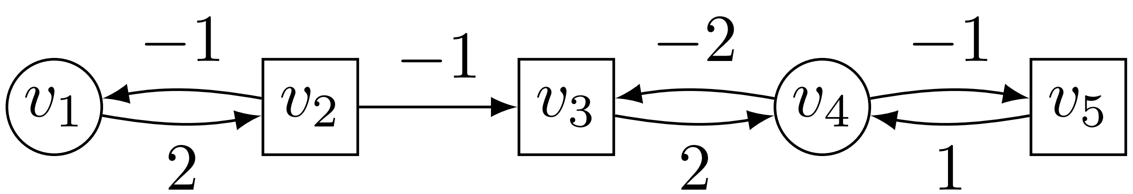
Total-payoff games



$$\pi = v_1 v_2 v_3 \; v_4 v_5 \; v_4 v_3 \; (v_4 v_5)^\omega$$

- Play: $\pi = v_0 v_1 v_2 \cdots$
- Prefix of length *k*: $\pi[k]$

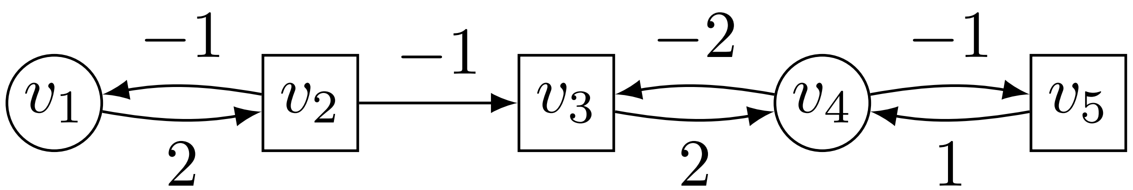
Total-payoff games



$$\pi = v_1 v_2 v_3 \; v_4 v_5 \; v_4 v_3 \; (v_4 v_5)^\omega$$

- Play: $\pi = v_0 v_1 v_2 \cdots$
- Prefix of length *k*: $\pi[k]$
- Accumulated cost: $\mathbf{TP}(\pi[k]) = \sum_{i=0}^{k-1} w(v_i, v_{i+1})$

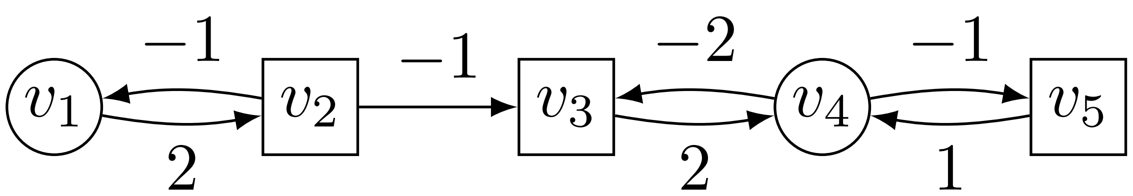
Total-payoff games



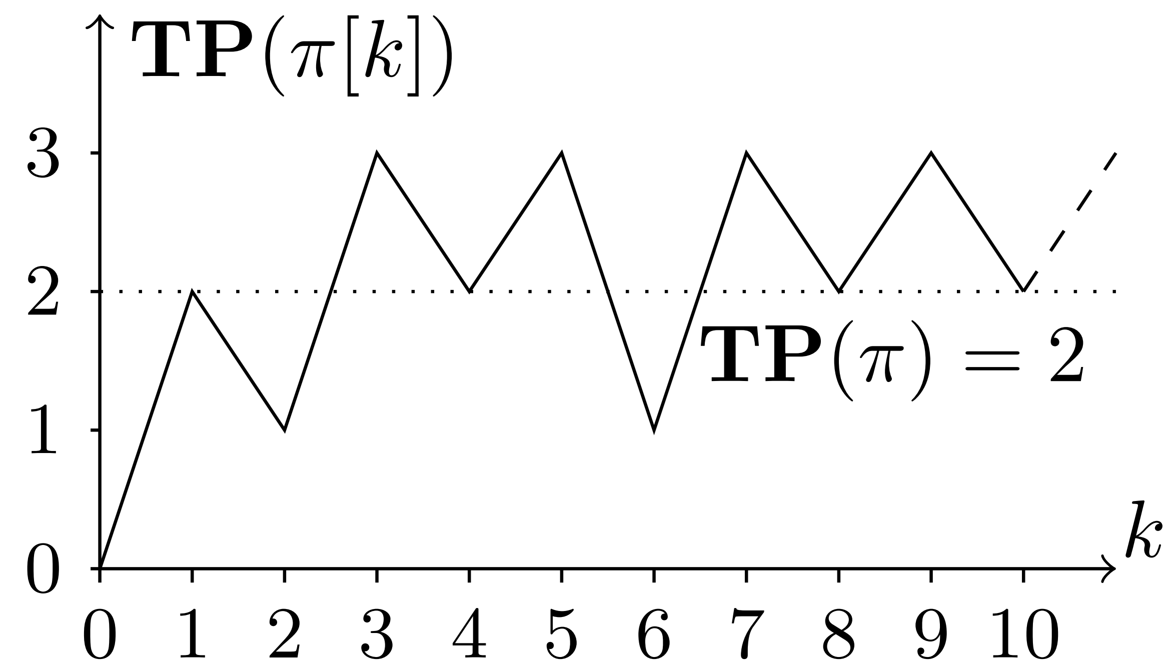
$\pi = v_1 v_2 v_3 \; v_4 v_5 \; v_4 v_3 \; (v_4 v_5)^\omega$

- Play: $\pi = v_0 v_1 v_2 \cdots$
- Prefix of length *k*: $\pi[k]$
- Accumulated cost: $\mathbf{TP}(\pi[k]) = \sum_{i=0}^{k-1} w(v_i, v_{i+1})$
- Total-payoff: $\mathbf{TP}(\pi) = \liminf_{k \to \infty} \mathbf{TP}(\pi[k])$

Total-payoff games



$$\pi = v_1 v_2 v_3 \; v_4 v_5 \; v_4 v_3 \; (v_4 v_5)^\omega$$



- Play: $\pi = v_0 v_1 v_2 \cdots$
- Prefix of length *k*: $\pi[k]$
- Accumulated cost: $\mathbf{TP}(\pi[k]) = \sum_{i=0}^{k-1} w(v_i, v_{i+1})$
- Total-payoff: $\mathbf{TP}(\pi) = \liminf_{k\to\infty} \mathbf{TP}(\pi[k])$

Total-payoff games

Known results:

- Gimbert & Zielonka 2004: optimal memoryless strategies always exist for both players
- Gawlitza & Seidl 2009: UP∩co-UP, best known algorithm runs in exponential time (policy iteration)
- No value iteration scheme known to work…

- Play: $\pi = v_0 v_1 v_2 \cdots$
- Prefix of length *k*: $\pi[k]$
- Accumulated cost: $\mathbf{TP}(\pi[k]) = \sum_{i=0}^{k-1} w(v_i, v_{i+1})$
- Total-payoff: $\mathbf{TP}(\pi) = \liminf_{k\to\infty} \mathbf{TP}(\pi[k])$

Total-payoff games

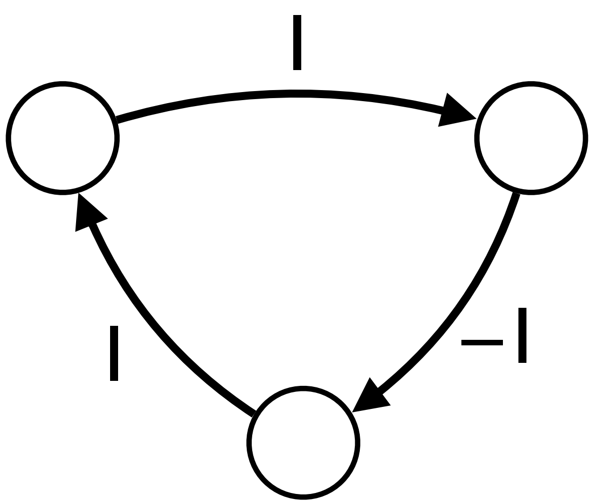
Known results:

- Gimbert & Zielonka 2004: optimal memoryless strategies always exist for both players
- Gawlitza & Seidl 2009: UP∩co-UP, best known algorithm runs in exponential time (policy iteration)
- No value iteration scheme known to work…

Our contribution:

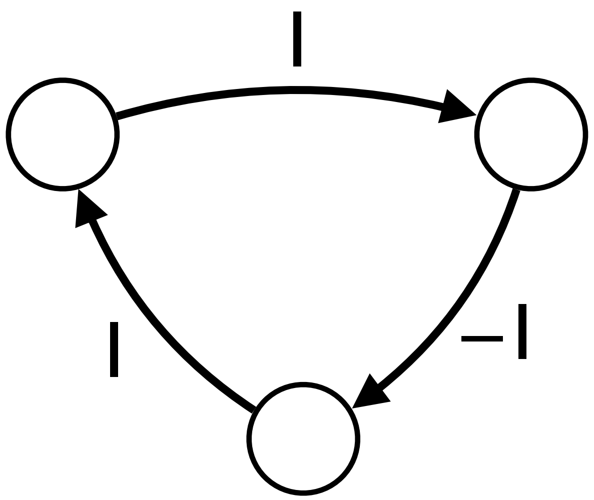
- First pseudo-polynomial time algorithm for total-payoff games + heuristics
- Requires the study of a variant with reachability

Mean-payoff vs Total-payoff

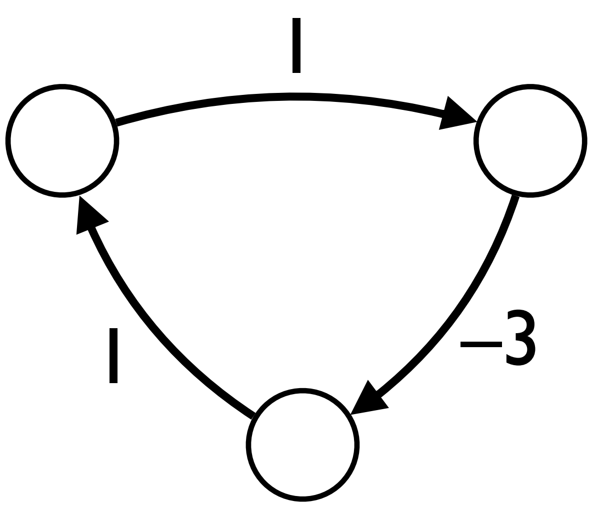


Mean-payoff value = 1/3 > 0 ⇒ Total-payoff value = +∞

Mean-payoff vs Total-payoff

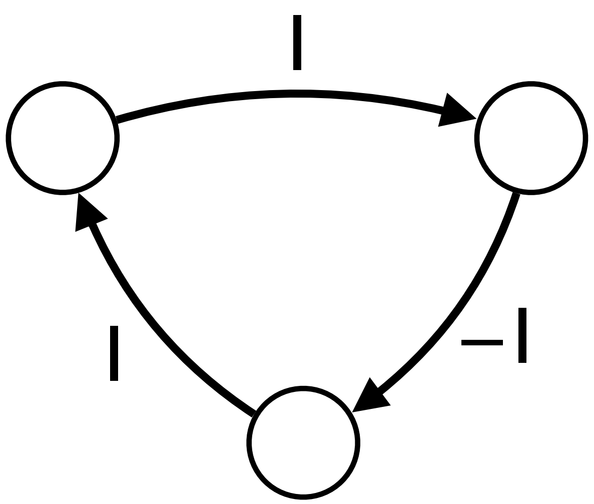


Mean-payoff value = 1/3 > 0 ⇒ Total-payoff value = $+\infty$

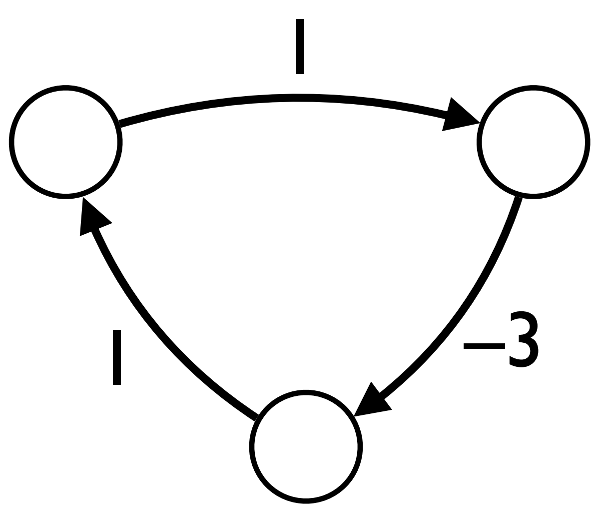


Mean-payoff value = –1/3 < 0 ⇒ Total-payoff value = $-\infty$

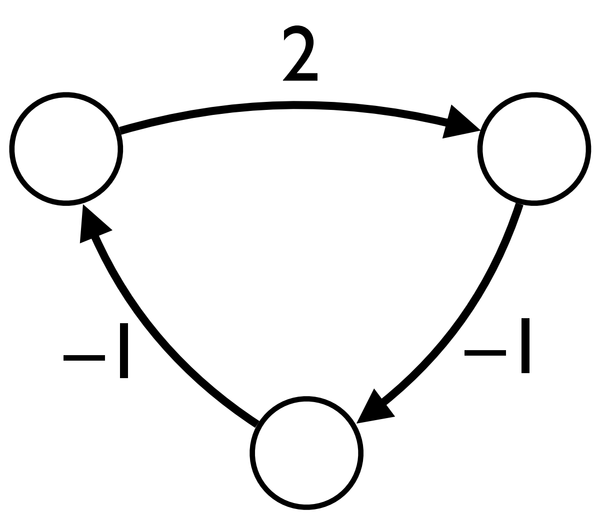
Mean-payoff vs Total-payoff



Mean-payoff value $= 1/3 > 0 \quad \Rightarrow \quad$ Total-payoff value $= +\infty$

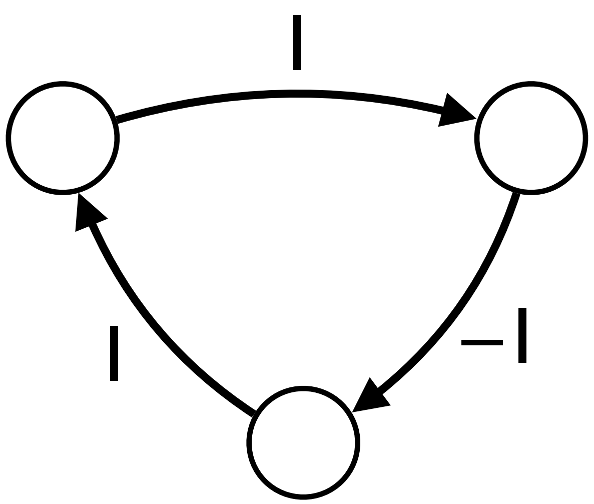


Mean-payoff value $= -1/3 < 0 \quad \Rightarrow \quad$ Total-payoff value $= -\infty$

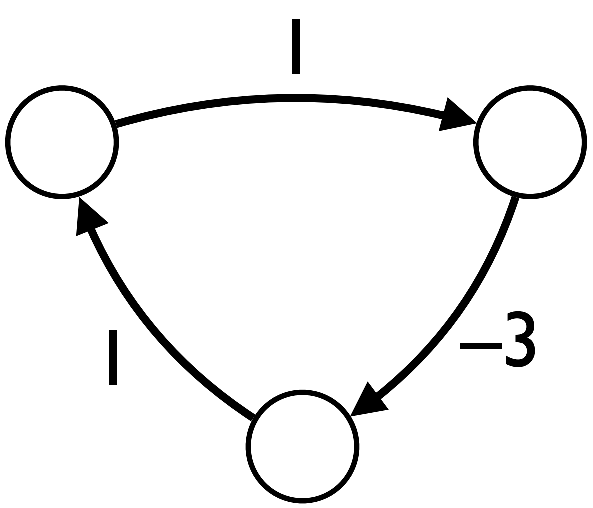


Mean-payoff value $= 0 \quad \Rightarrow \quad$ Total-payoff value finite

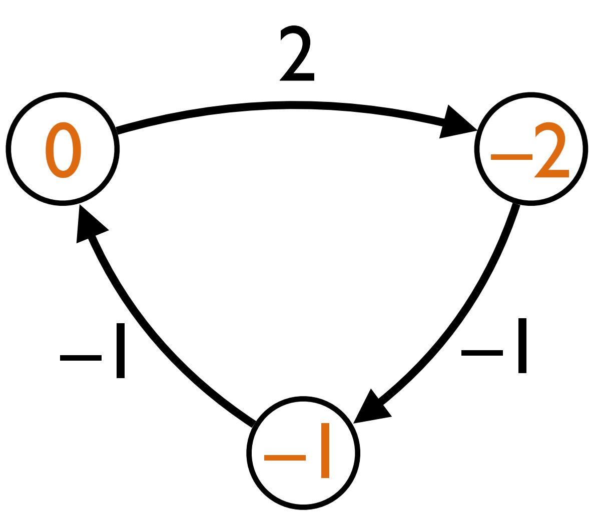
Mean-payoff vs Total-payoff



Mean-payoff value = 1/3 > 0 ⇒ Total-payoff value = +∞

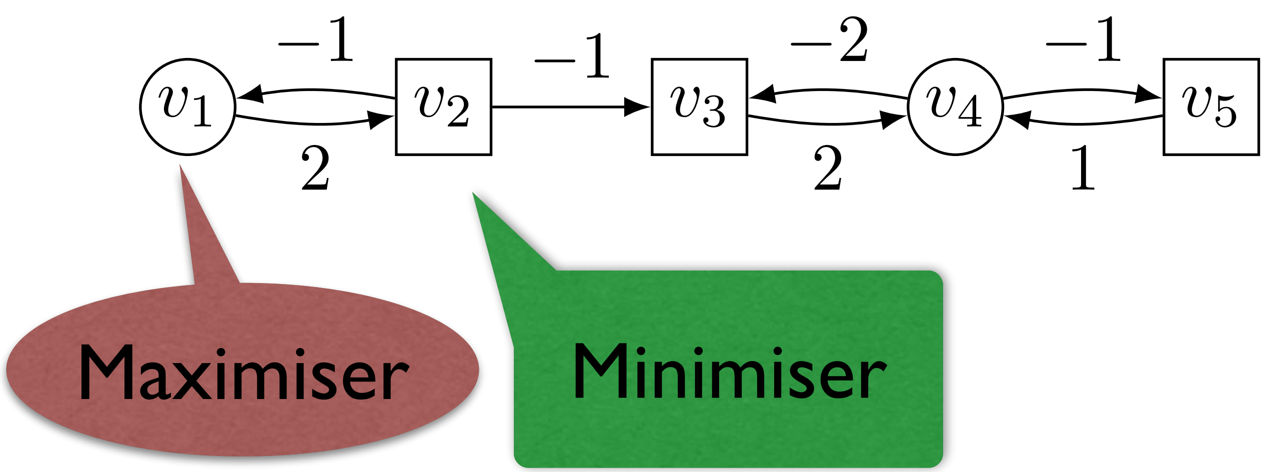


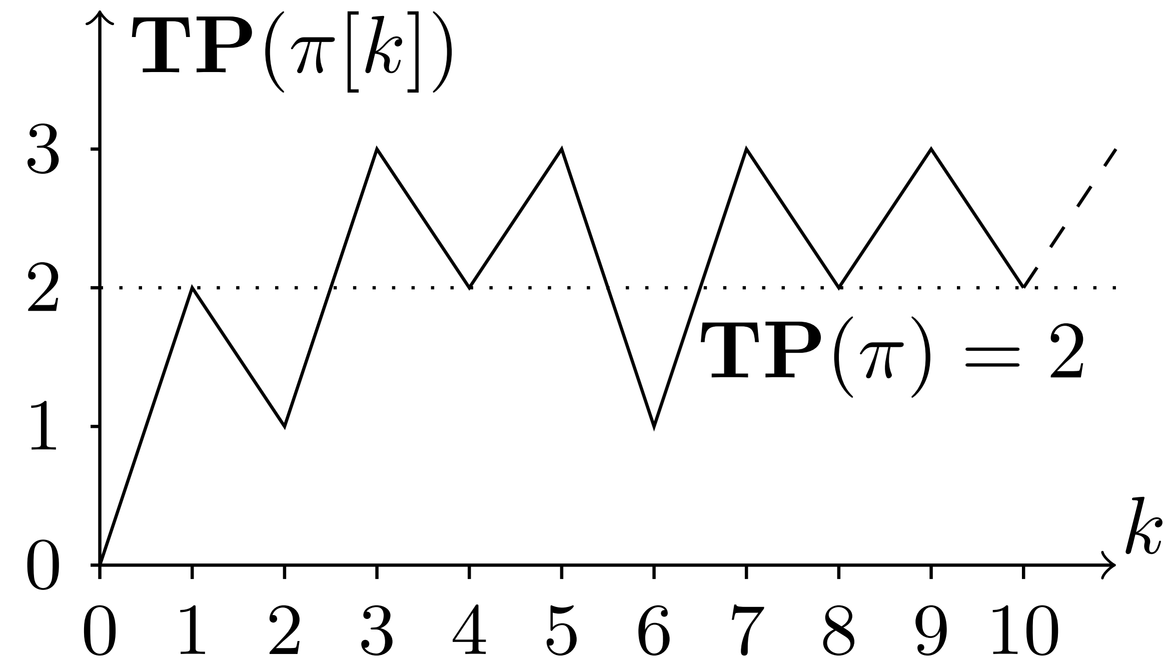
Mean-payoff value = −1/3 < 0 ⇒ Total-payoff value = −∞



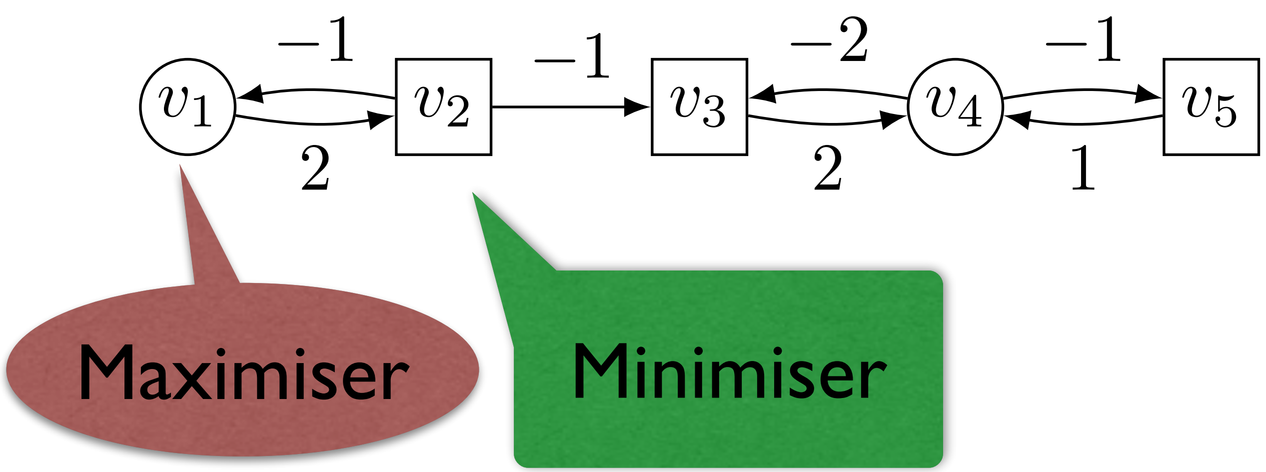
Mean-payoff value = 0 ⇒ Total-payoff value finite

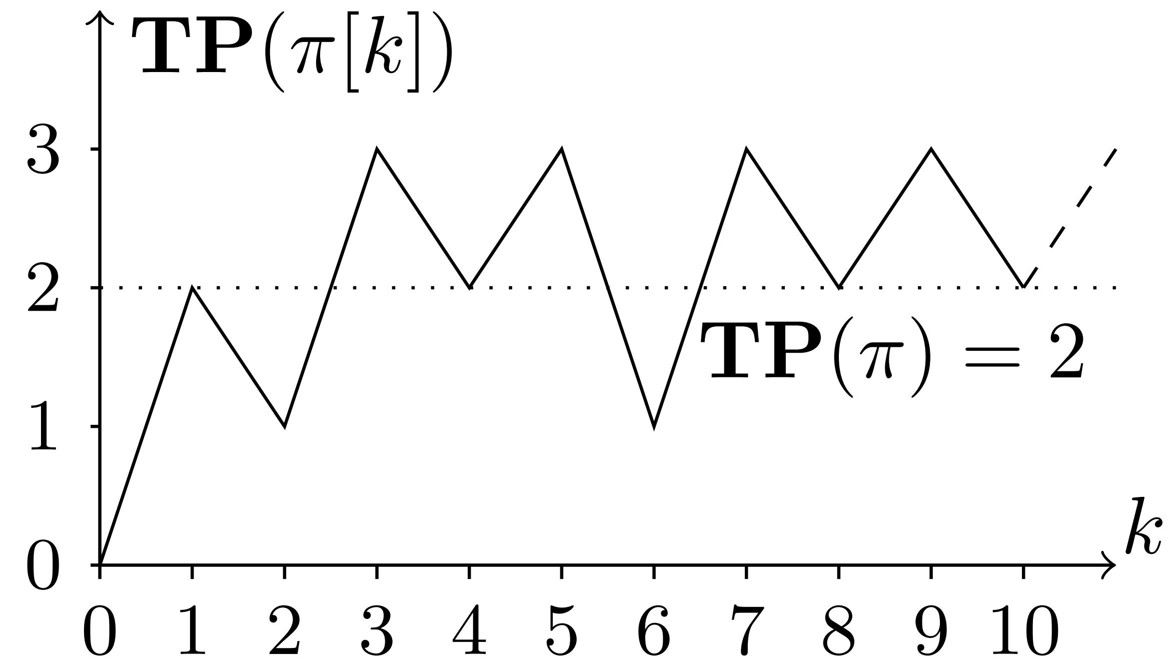
Our result in a nutshell





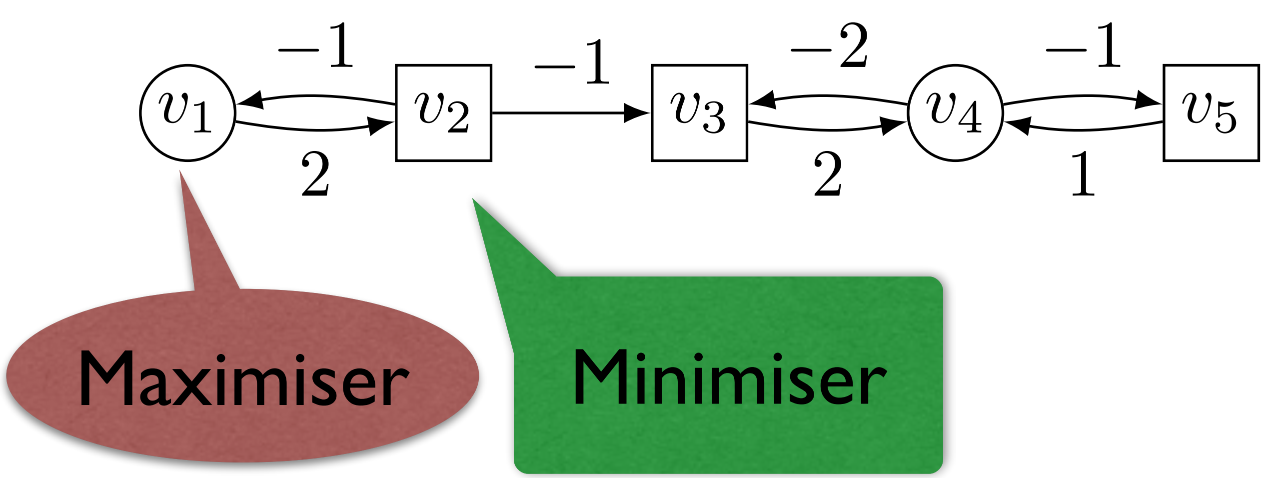
Our result in a nutshell

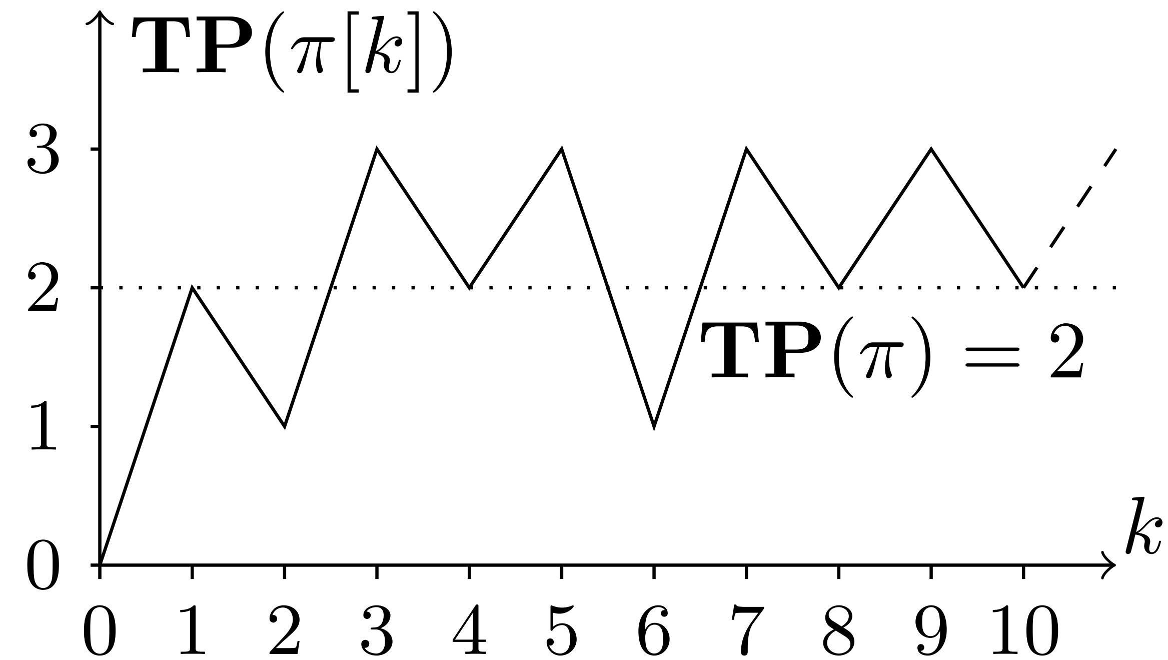




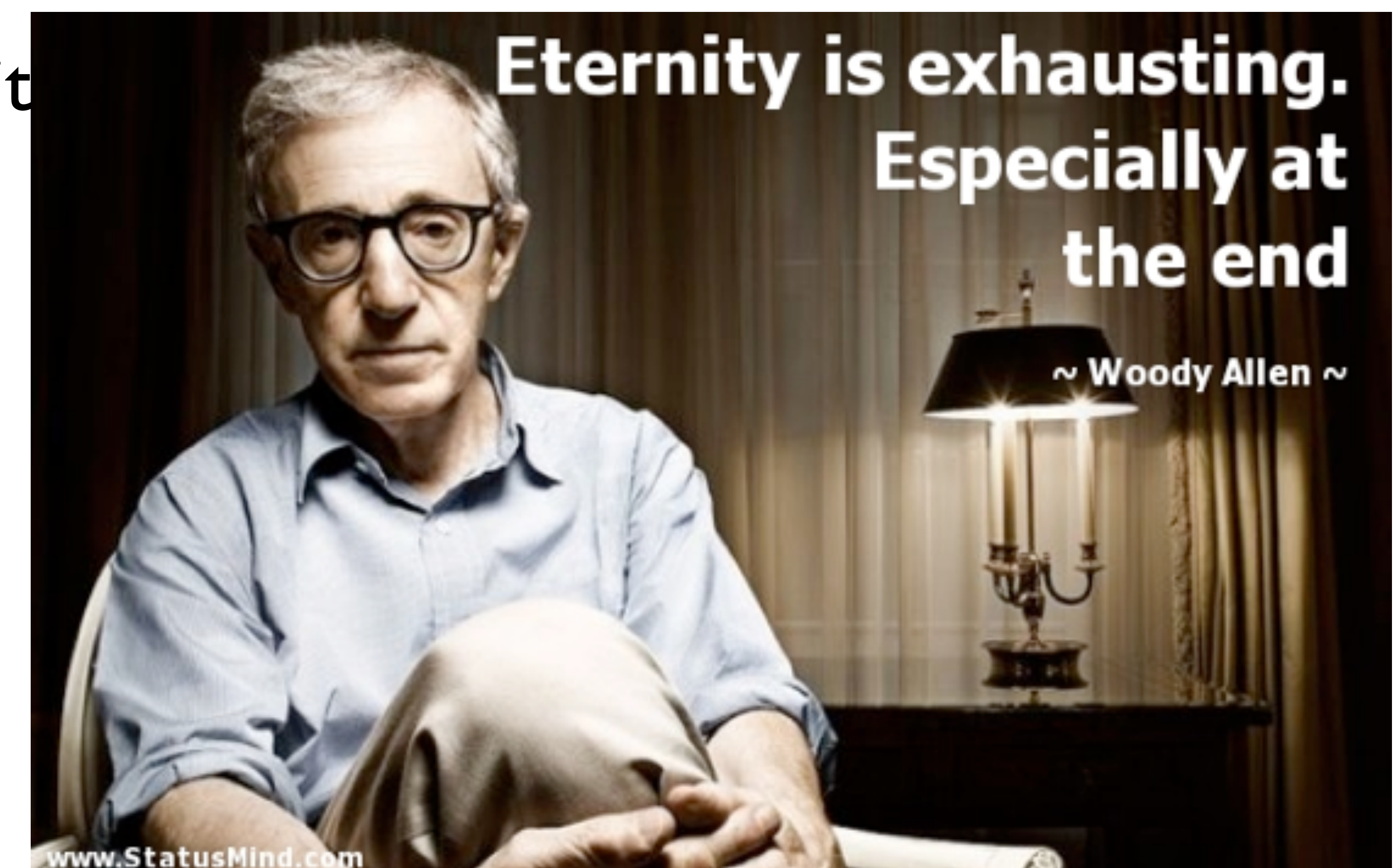
- Total-payoff games are infinite by essence…

Our result in a nutshell

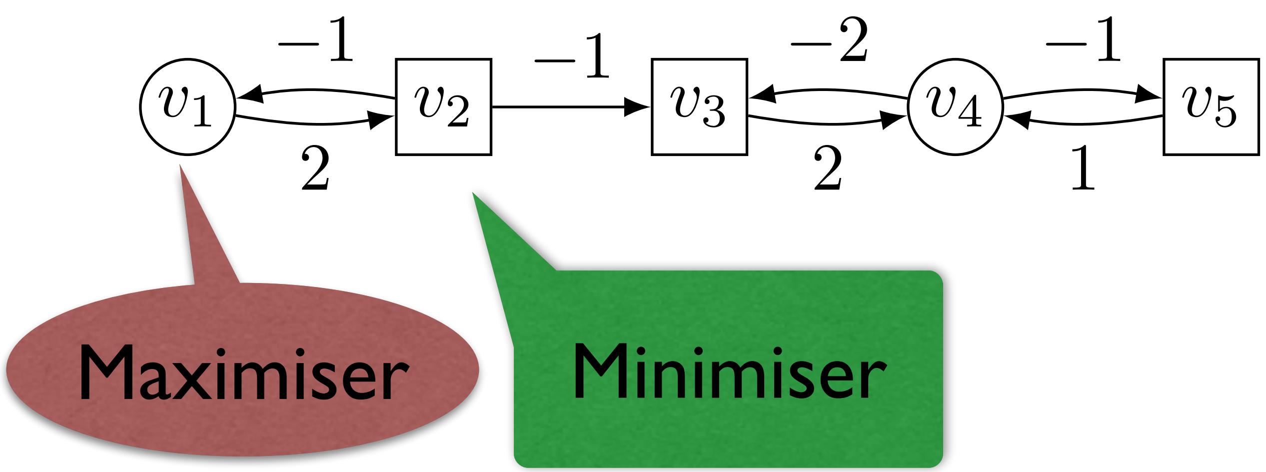


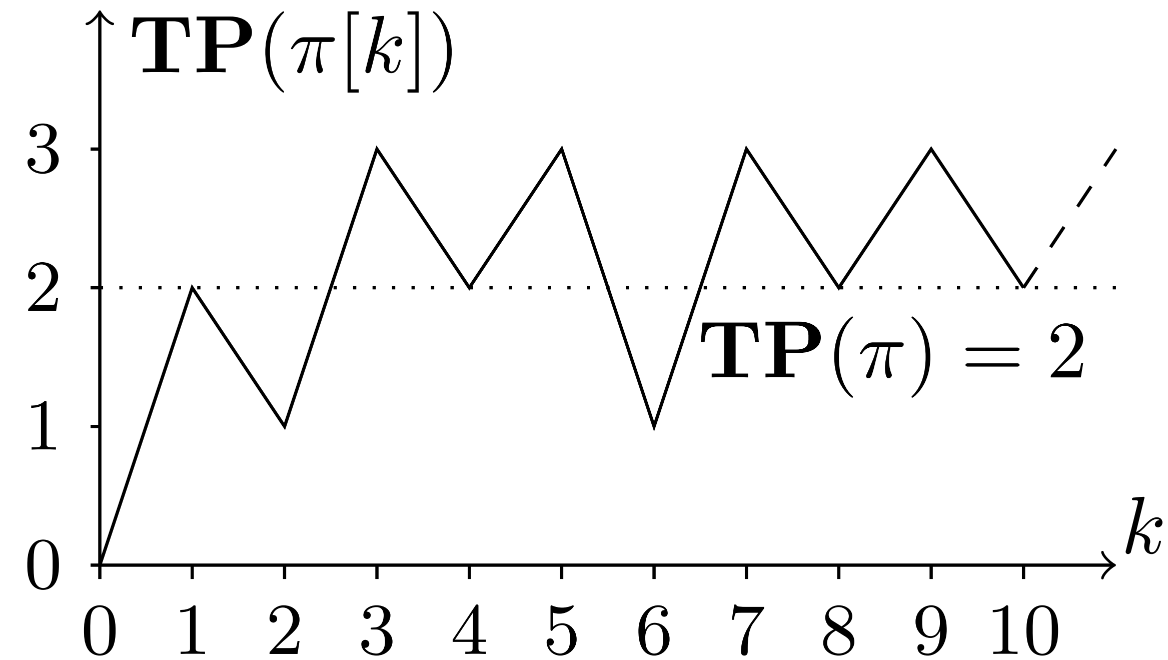


- Total-payoff games are infinit



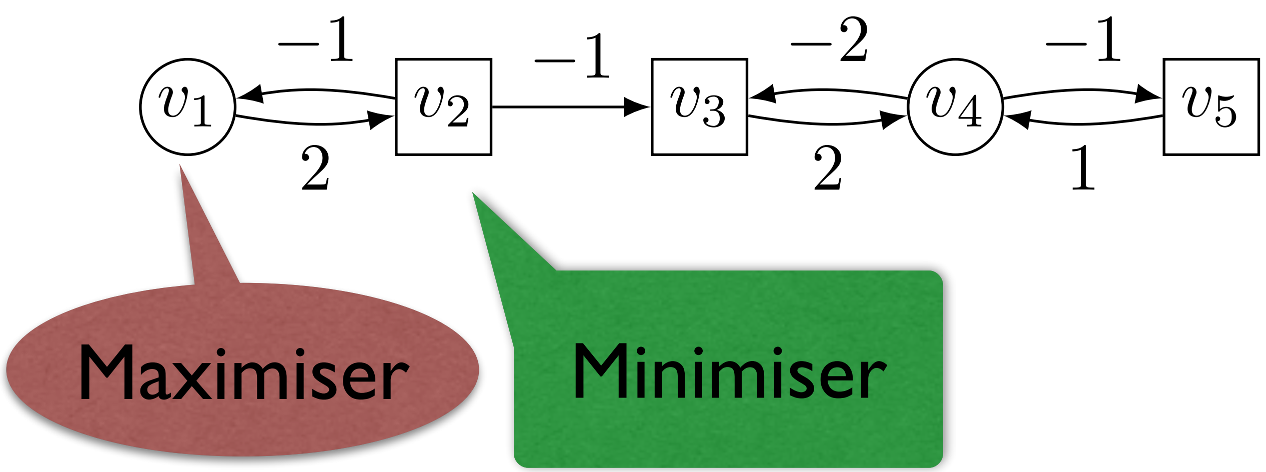
Our result in a nutshell

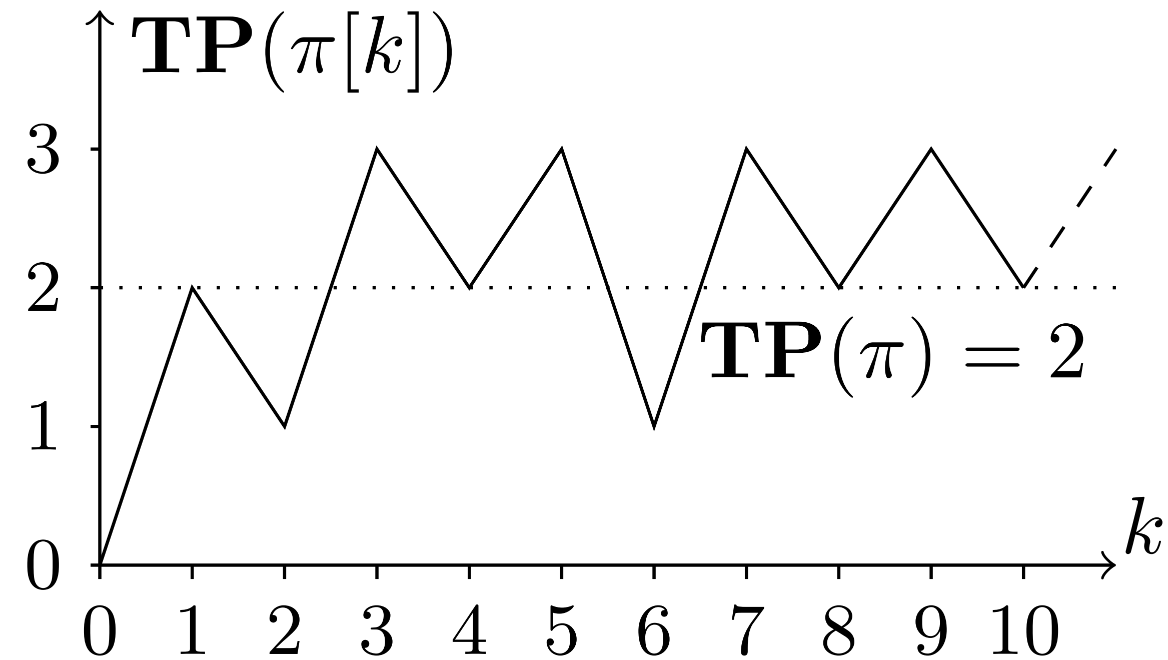




- Total-payoff games are infinite by essence…

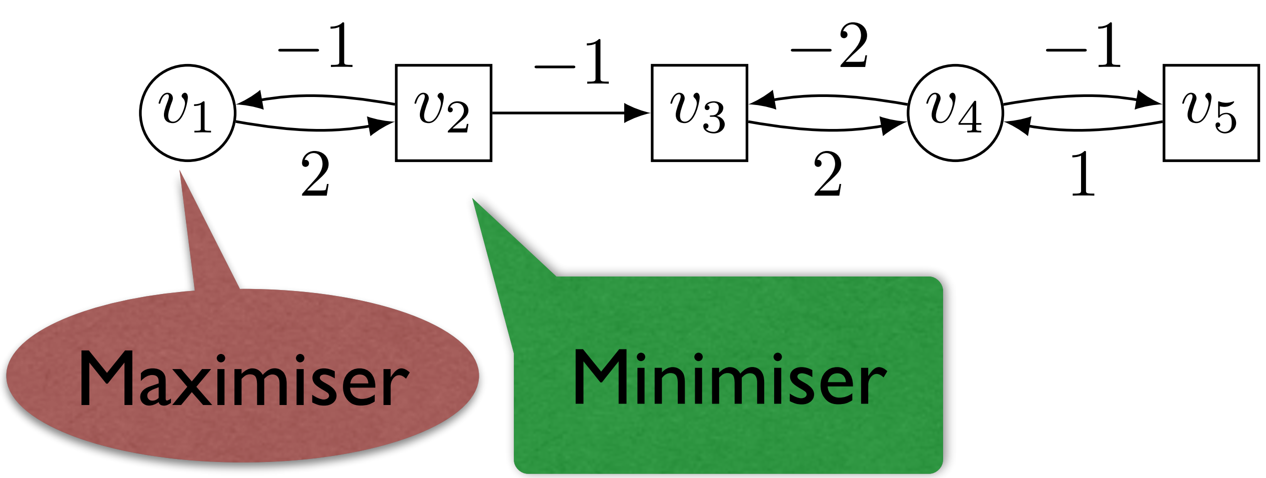
Our result in a nutshell

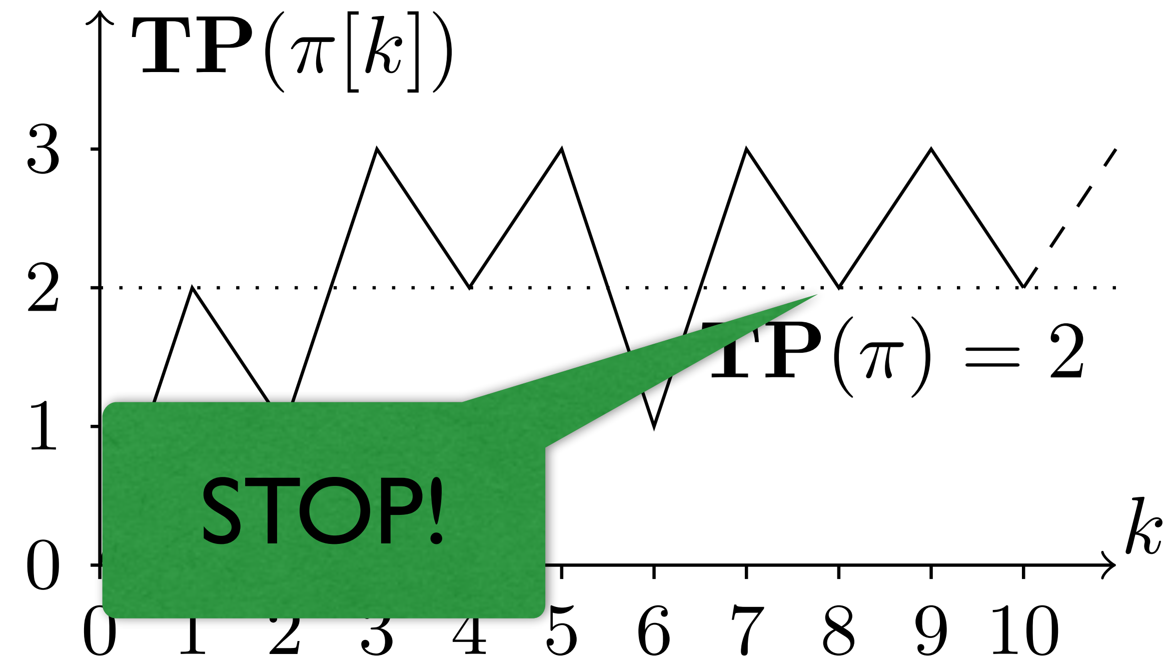




- Total-payoff games are infinite by essence…
- Easier if finite-horizon? Add a stopping possibility for the Minimiser!

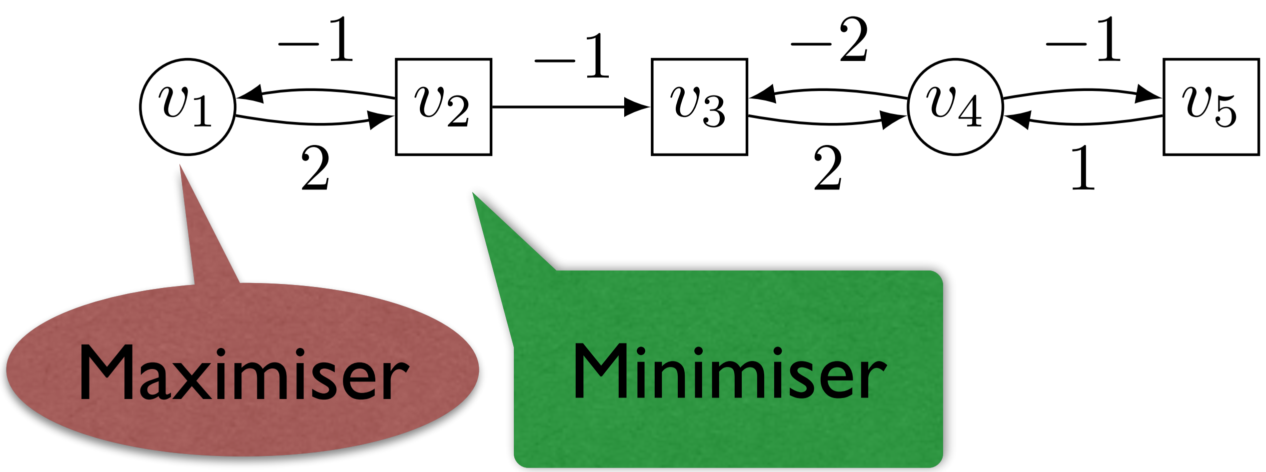
Our result in a nutshell

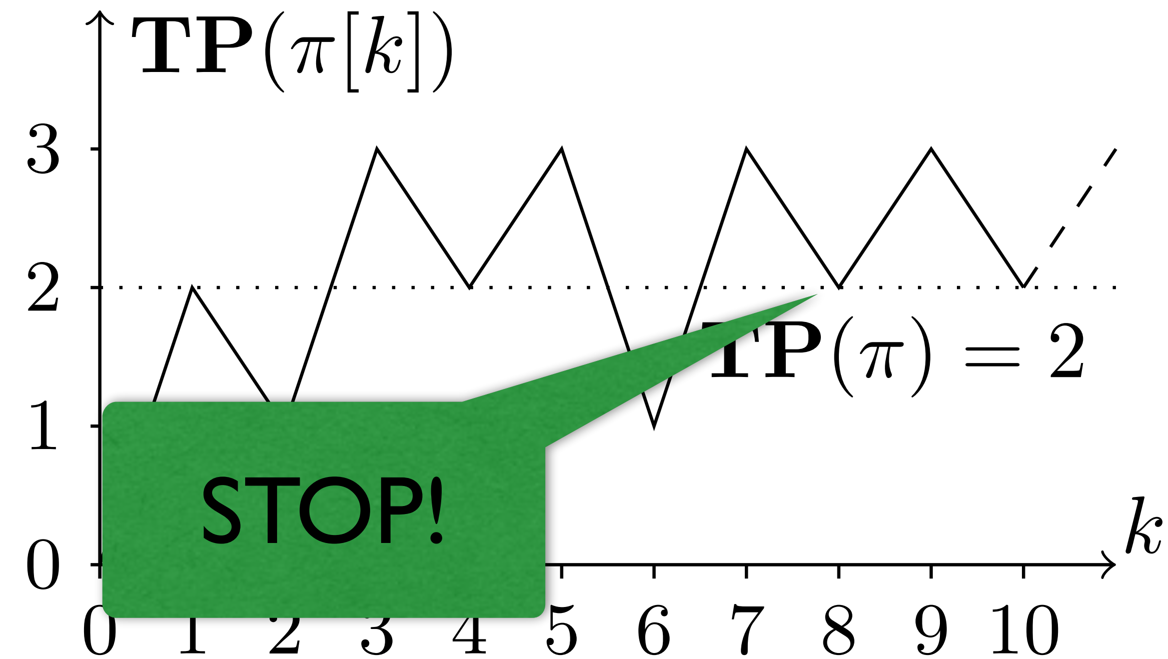




- Total-payoff games are infinite by essence…
- Easier if finite-horizon? Add a stopping possibility for the Minimiser!

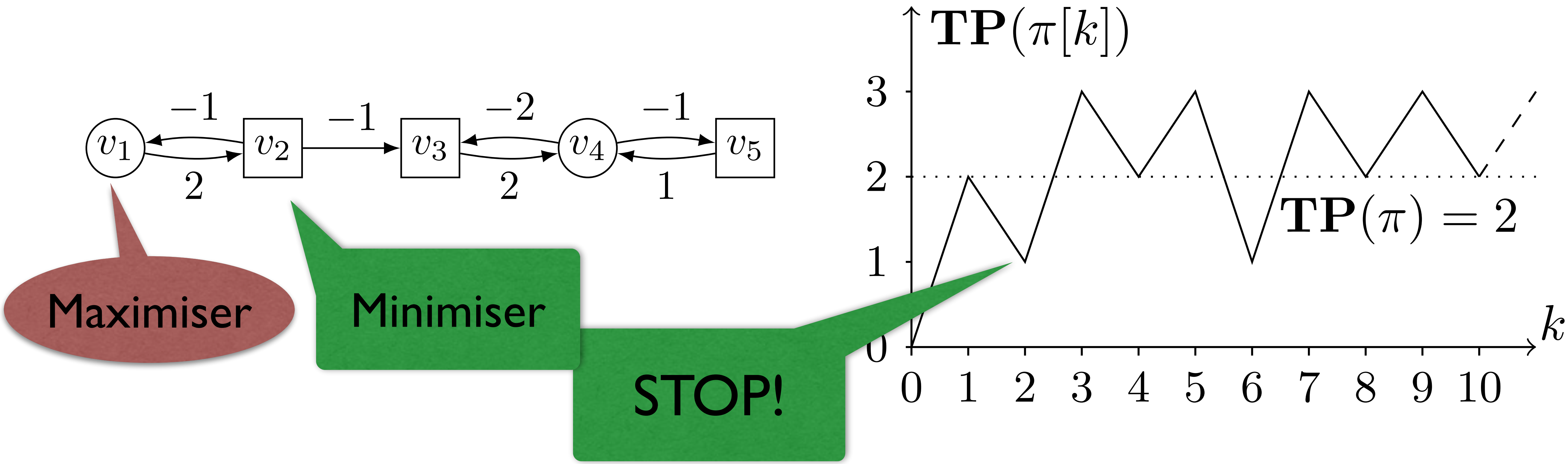
Our result in a nutshell





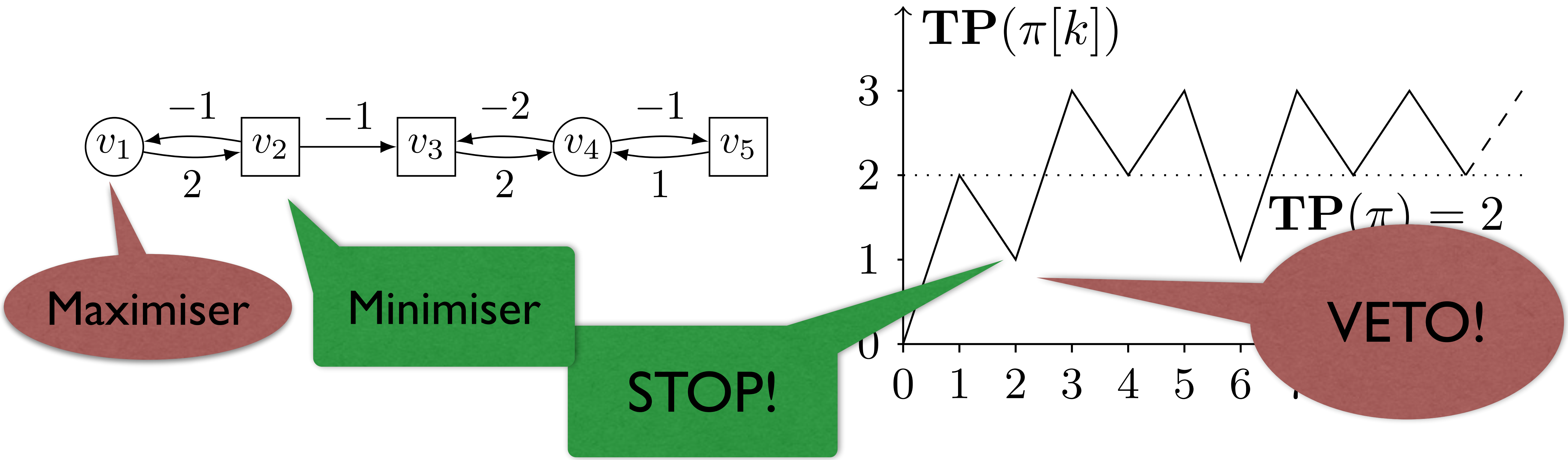
- Total-payoff games are infinite by essence…
- Easier if finite-horizon? Add a stopping possibility for the Minimiser!
- Too easy for him! Let the Maximiser have the right to veto, for a limited number *K* of times… Here *K*=3 suffices.

Our result in a nutshell



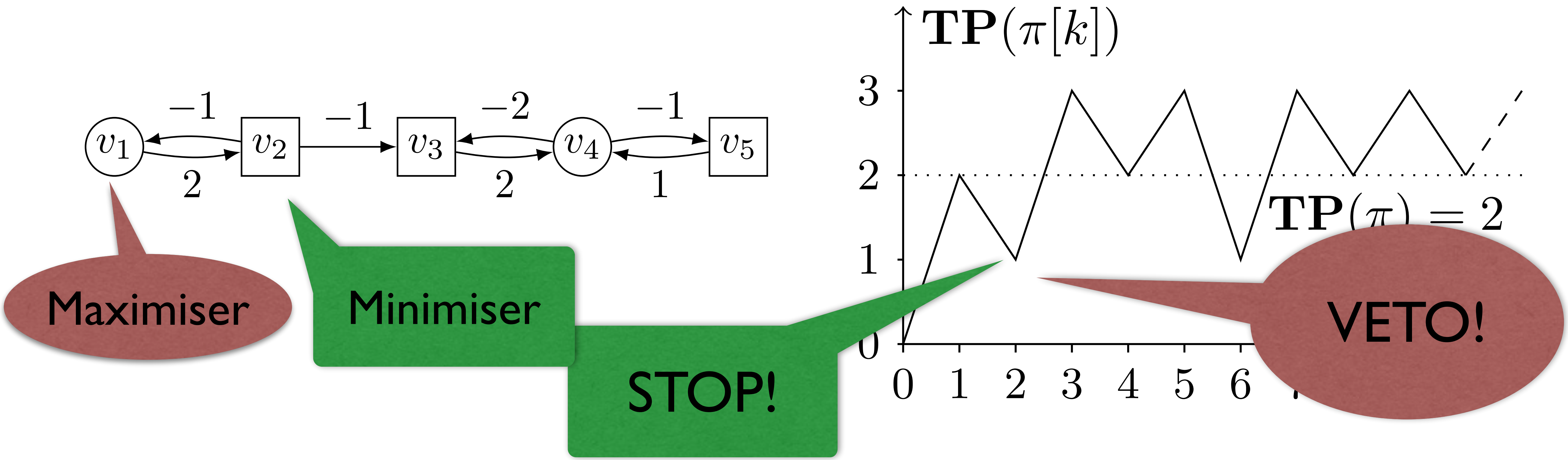
- Total-payoff games are infinite by essence…
- Easier if finite-horizon? Add a stopping possibility for the Minimiser!
- Too easy for him! Let the Maximiser have the right to veto, for a limited number *K* of times… Here *K*=3 suffices.

Our result in a nutshell



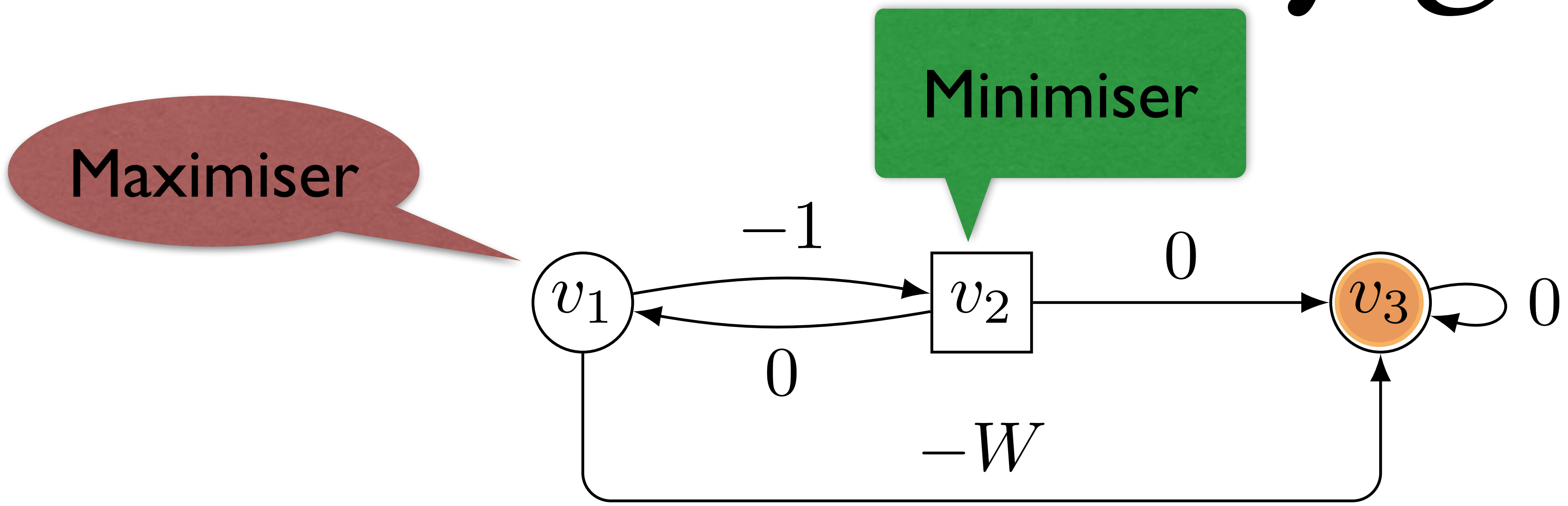
- Total-payoff games are infinite by essence…
- Easier if finite-horizon? Add a stopping possibility for the Minimiser!
- Too easy for him! Let the Maximiser have the right to veto, for a limited number *K* of times… Here *K*=3 suffices.

Our result in a nutshell

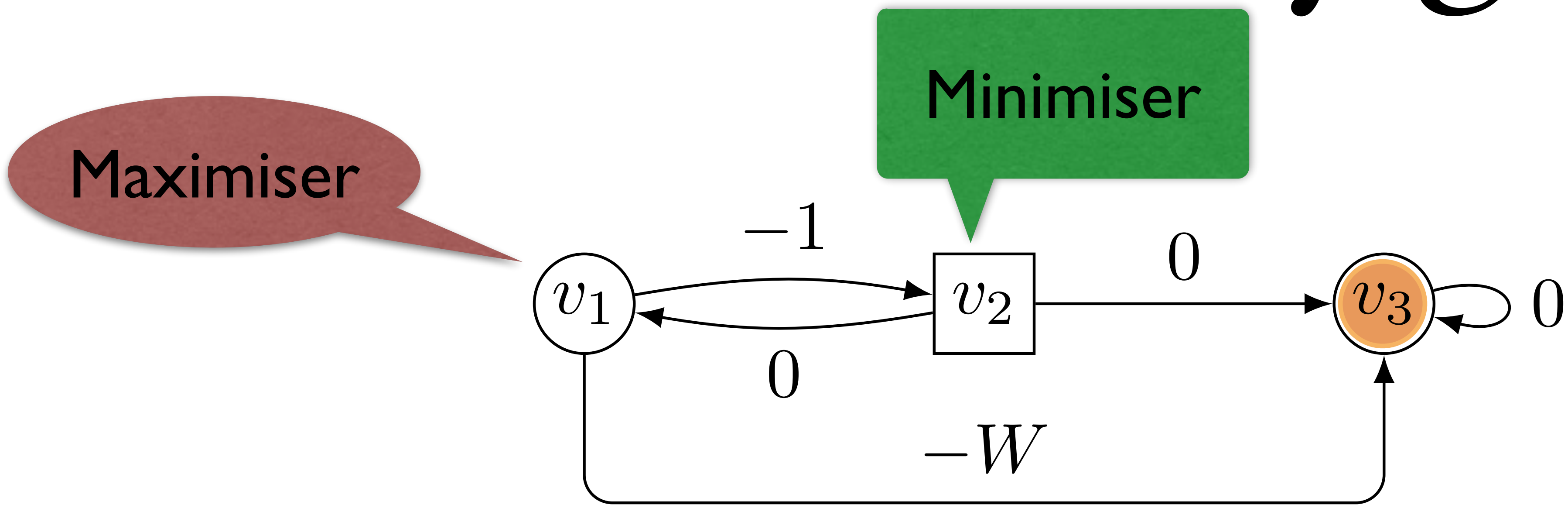


- Total-payoff games are infinite by essence…
- Easier if finite-horizon? Add a stopping possibility for the Minimiser!
- Too easy for him! Let the Maximiser have the right to veto, for a limited number *K* of times… Here *K*=3 suffices.
- Is there always a good value of *K* so that both games are equivalent?

Min-cost reachability games

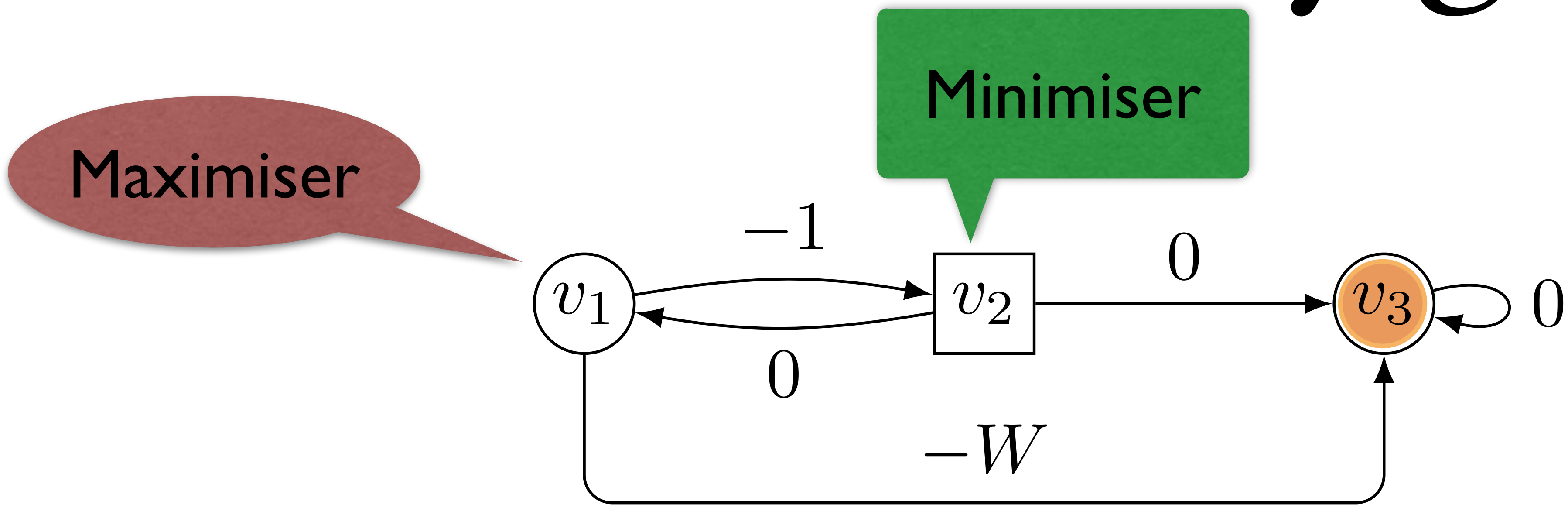


Min-cost reachability games



- Target set of states T

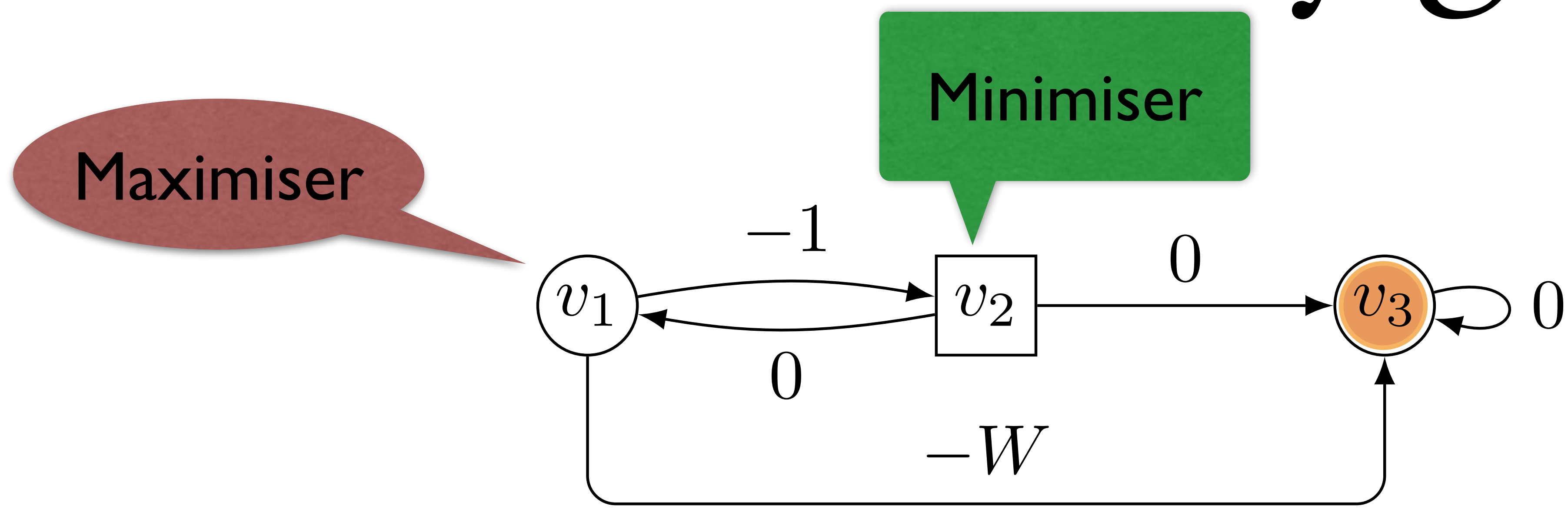
Min-cost reachability games



- Target set of states *T*

- Minimiser wants to reach:
  - if target is not reached, payoff $+\infty$,
  - if target is reached, accumulated sum until first occurrence of a target state

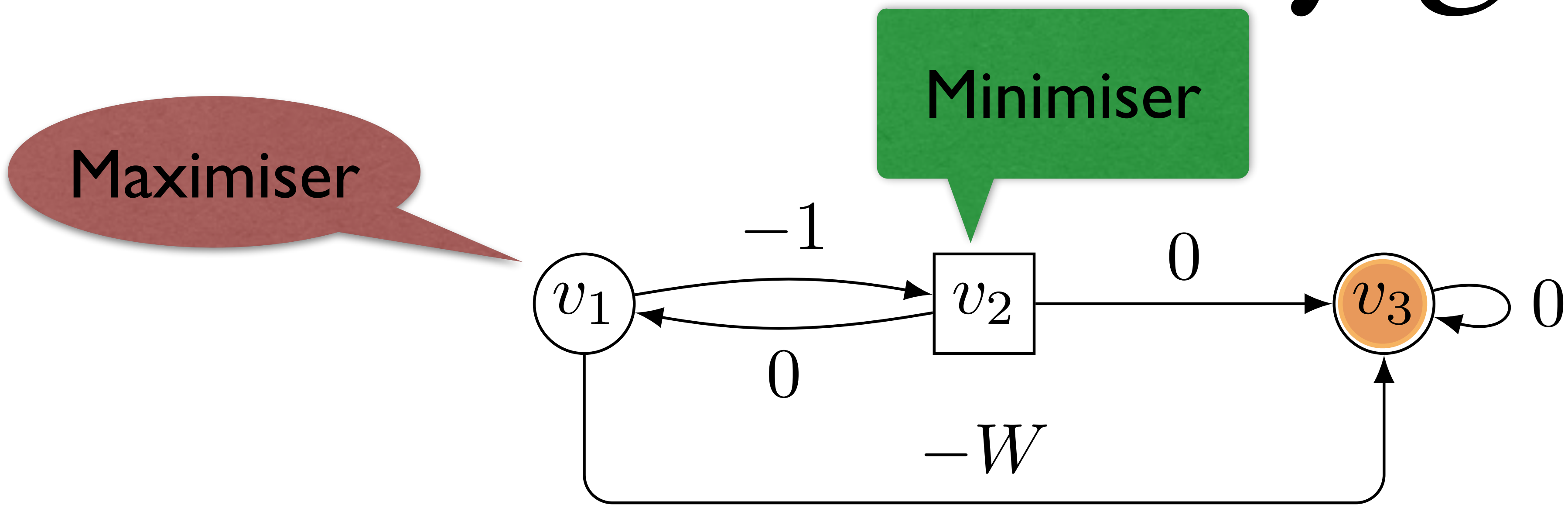
Min-cost reachability games



- Target set of states T
- Minimiser wants to reach:
  - if target is not reached, payoff $+\infty$,
  - if target is reached, accumulated sum until first occurrence of a target state
- Value of the game:

$$\inf_{\sigma_{\min}} \sup_{\sigma_{\max}} T\text{-MCR}(v, \sigma_{\min}, \sigma_{\max}) = \sup_{\sigma_{\max}} \inf_{\sigma_{\min}} T\text{-MCR}(v, \sigma_{\min}, \sigma_{\max})$$

Min-cost reachability games



- Target set of states T
- Minimiser wants to reach:
  - if target is not reached, payoff $+\infty$,
  - if target is reached, accumulated sum until first occurrence of a target state
- Value of the game:

$$\inf_{\sigma_{\min}} \sup_{\sigma_{\max}} T\text{-MCR}(v, \sigma_{\min}, \sigma_{\max}) = \sup_{\sigma_{\max}} \inf_{\sigma_{\min}} T\text{-MCR}(v, \sigma_{\min}, \sigma_{\max})$$

- Example: value of v_1 and v_2 is $-W$… and Minimiser needs memory to ensure it!

Solving MCR games (1)

Solving MCR games (1)

- Case one player: shortest path in a weighted graph…
  ➡ Polynomial time

Solving MCR games (1)

- Case one player: shortest path in a weighted graph…
  ➡ Polynomial time
- Case non-negative weights: Dijsktra's algorithm adapted by Khachiyan *et al* 2008
  ➡ Polynomial time

Solving MCR games (1)

- Case one player: shortest path in a weighted graph…
  ➡ Polynomial time
- Case non-negative weights: Dijsktra's algorithm adapted by Khachiyan *et al* 2008
  ➡ Polynomial time
- General case? Not known…
  ➡ Our contribution: pseudo-polynomial time and as hard as solving mean-payoff games

Solving MCR games (2)

Solving MCR games (2)

- Detecting vertices with value $+\infty$
  - ➡ Attractor computation: polynomial time

Solving MCR games (2)

- Detecting vertices with value $+\infty$
  ➡ Attractor computation: polynomial time

- Detecting vertices with value $-\infty$
  ➡ Equivalent to checking if the value of a mean-payoff game is negative: decidable in NP∩co-NP, pseudo-polynomial time

Solving MCR games (2)

- Detecting vertices with value $+\infty$
  - ➡ Attractor computation: polynomial time
- Detecting vertices with value $-\infty$
  - ➡ Equivalent to checking if the value of a mean-payoff game is negative: decidable in NP∩co-NP, pseudo-polynomial time
- Computing finite values
  - ➡ Value iteration running in pseudo-polynomial time

Solving MCR games (2)

- Detecting vertices with value $+\infty$
  - ➡ Attractor computation: polynomial time
- Detecting vertices with value $-\infty$
  - ➡ Equivalent to checking if the value of a mean-payoff game is negative: decidable in NP∩co-NP, pseudo-polynomial time
- Computing finite values
  - ➡ Value iteration running in pseudo-polynomial time
- Computing optimal strategies when all values are finite
  - ➡ May be done simultaneously, with same complexity

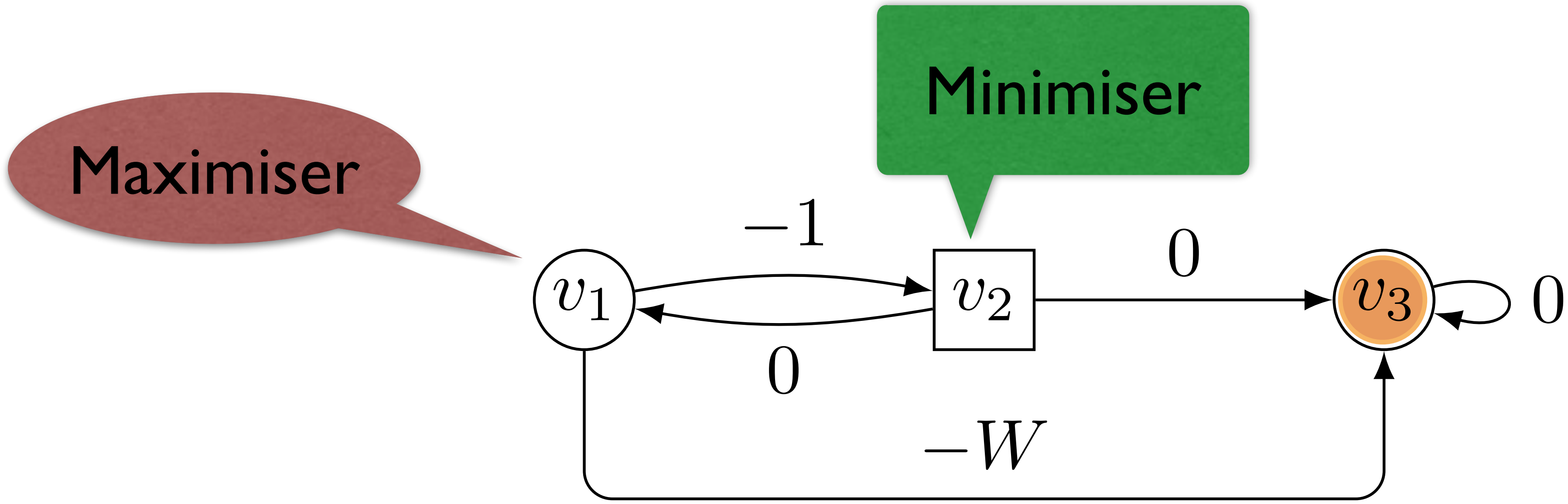
Solving MCR games (2)

- Detecting vertices with value $+\infty$
  - ➡ Attractor computation: polynomial time
- Detecting vertices with value $-\infty$
  - ➡ Equivalent to checking if the value of a mean-payoff game is negative: decidable in NP∩co-NP, pseudo-polynomial time
- Computing finite values
  - ➡ Value iteration running in pseudo-polynomial time
- Computing optimal strategies when all values are finite
  - ➡ May be done simultaneously, with same complexity
  - ➡ Maximiser: memoryless optimal strategy

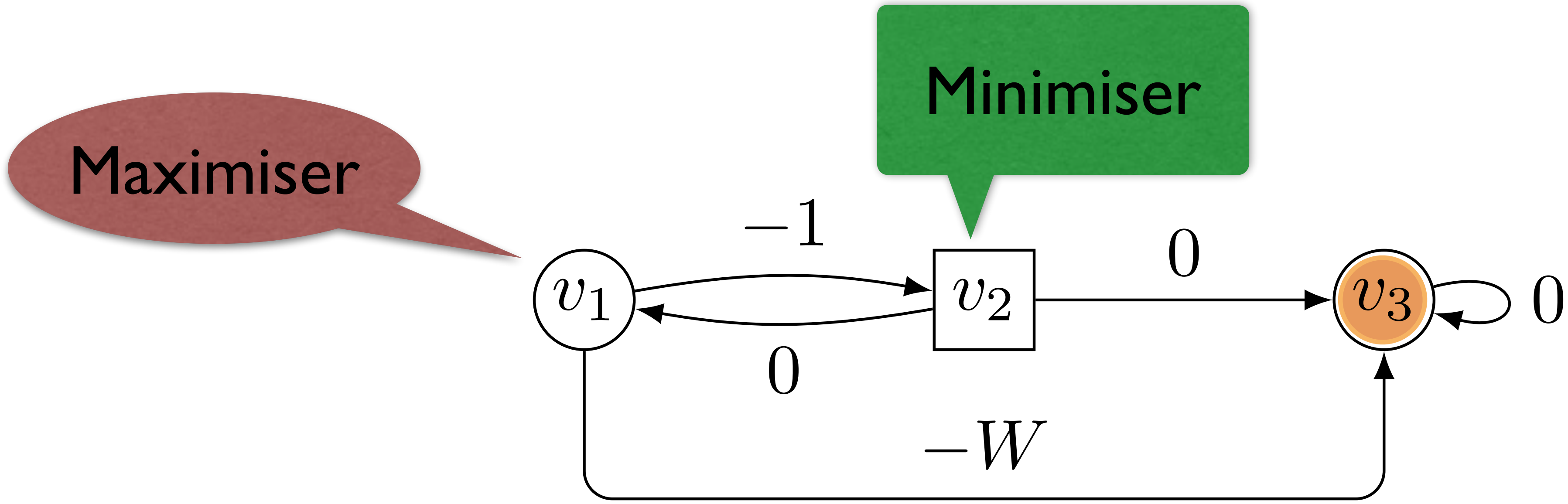
Solving MCR games (2)

- Detecting vertices with value $+\infty$
  - ➡ Attractor computation: polynomial time
- Detecting vertices with value $-\infty$
  - ➡ Equivalent to checking if the value of a mean-payoff game is negative: decidable in NP∩co-NP, pseudo-polynomial time
- Computing finite values
  - ➡ Value iteration running in pseudo-polynomial time
- Computing optimal strategies when all values are finite
  - ➡ May be done simultaneously, with same complexity
  - ➡ Maximiser: memoryless optimal strategy
  - ➡ Minimiser: finite memory suffices, and may be required

Value iteration on an example

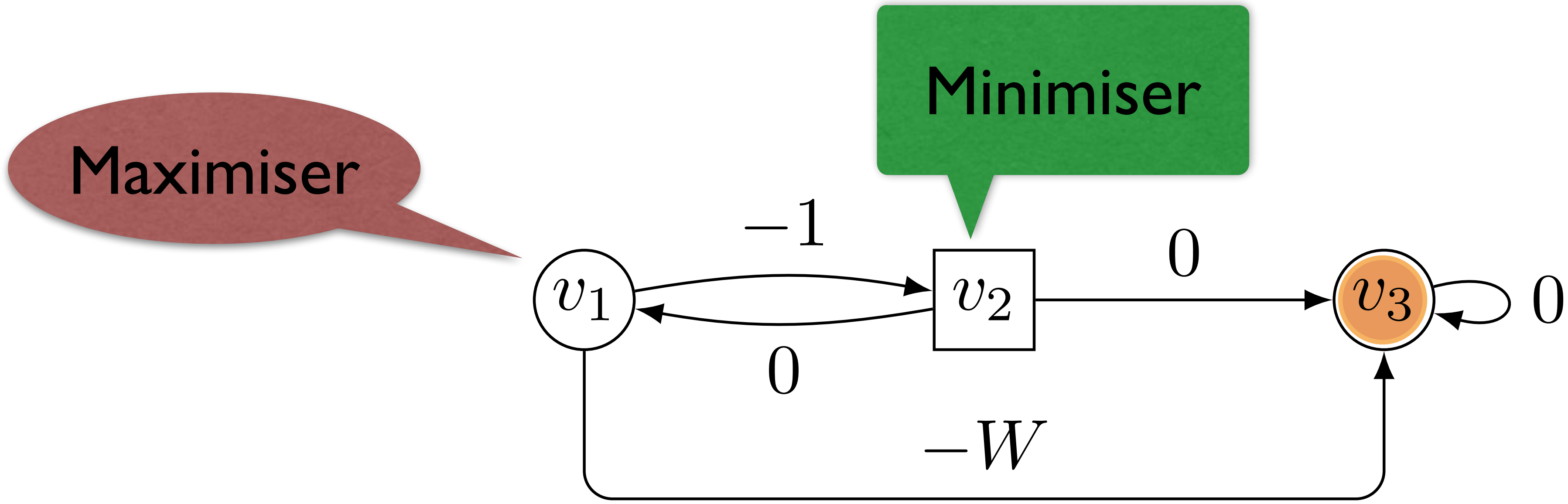


Value iteration on an example



$+\infty$	$+\infty$	0

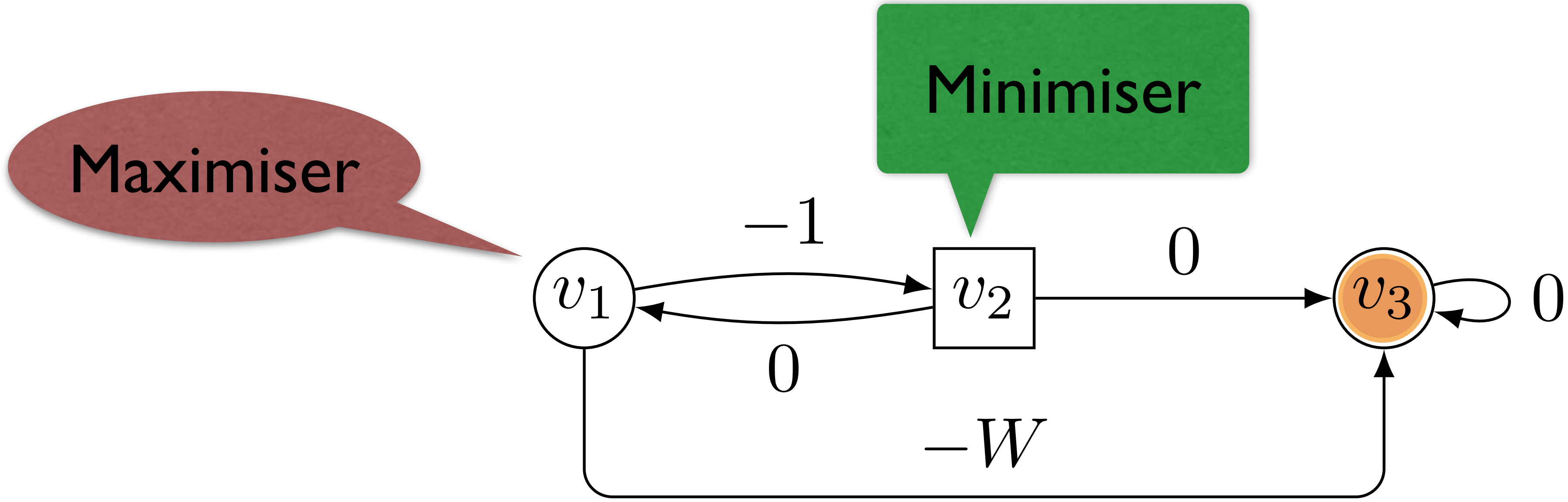
Value iteration on an example



what may both players achieve in 1 step

$+\infty$	$+\infty$	0
$+\infty$	0	0

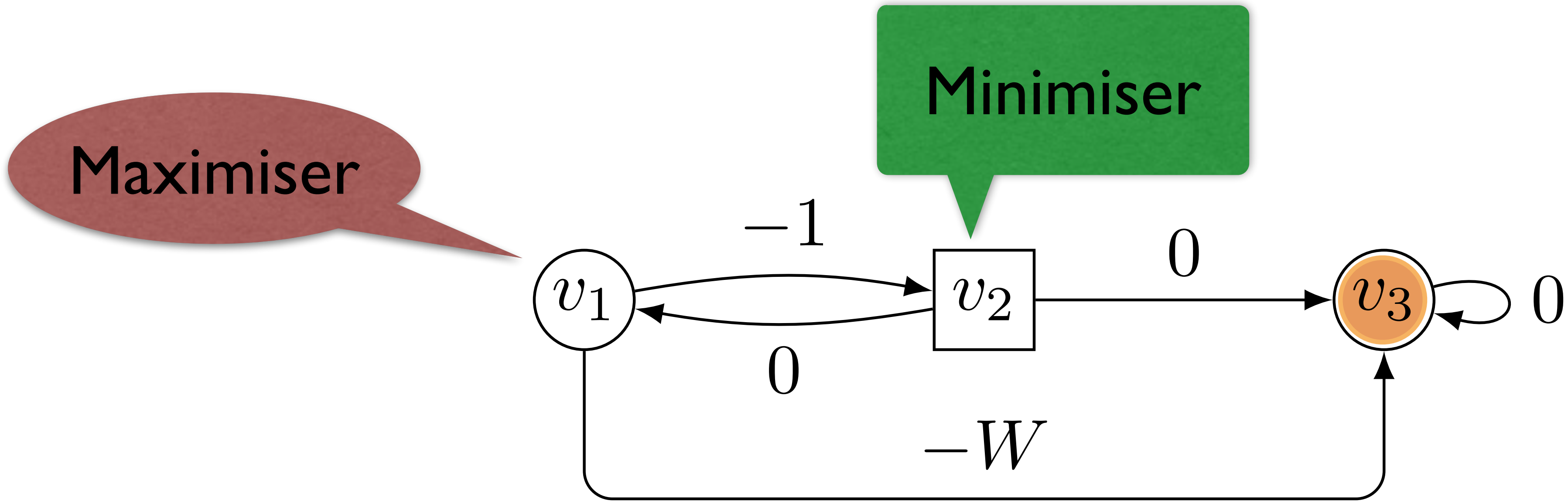
Value iteration on an example



what may both players achieve in 1 step

$+\infty$	$+\infty$	0
$+\infty$	0	0
-1	0	0

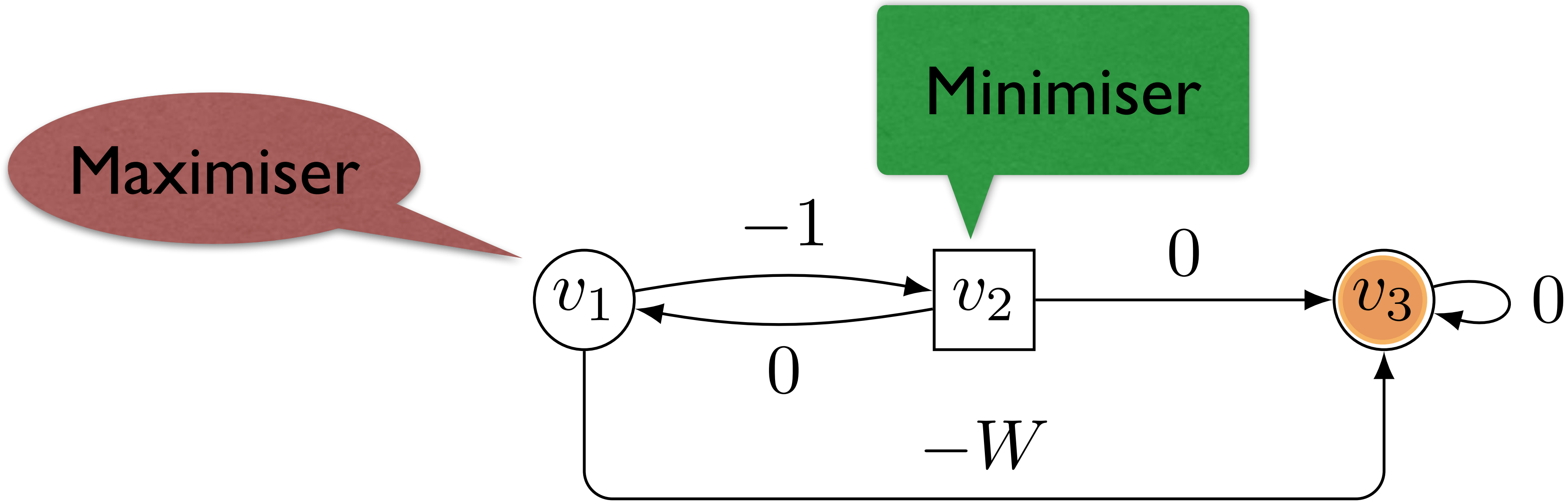
Value iteration on an example



what may both players achieve in 1 step

$+\infty$	$+\infty$	0
$+\infty$	0	0
-1	0	0
-1	-1	0

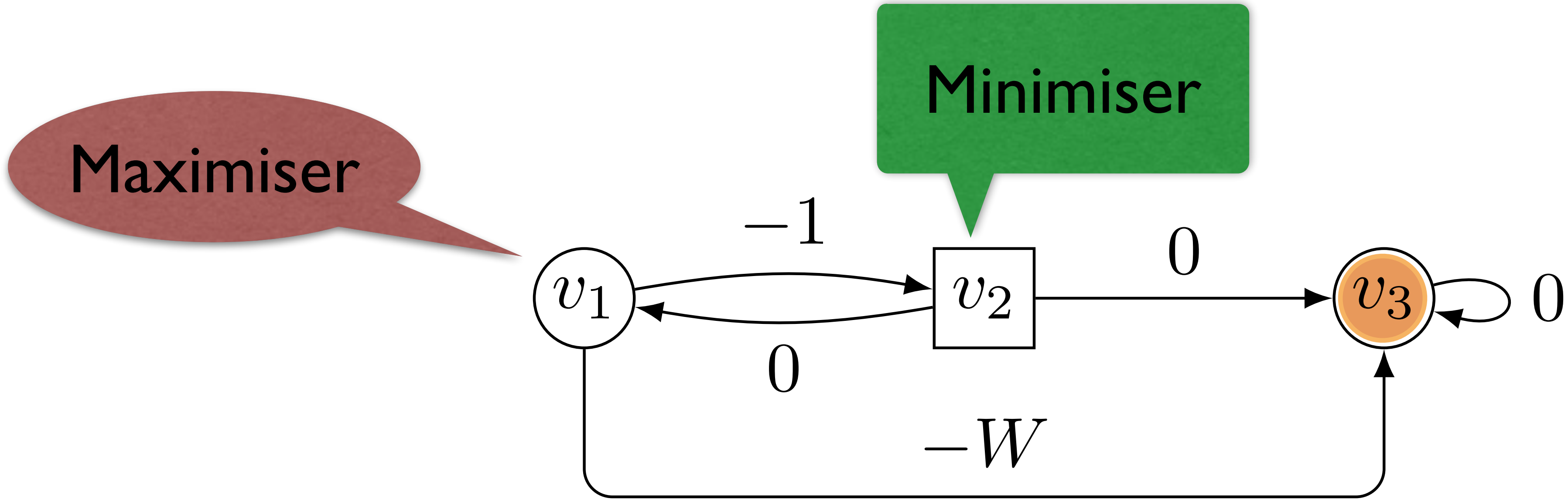
Value iteration on an example



what may both players achieve in 1 step

$+\infty$	$+\infty$	0
$+\infty$	0	0
-1	0	0
-1	-1	0
-2	-1	0

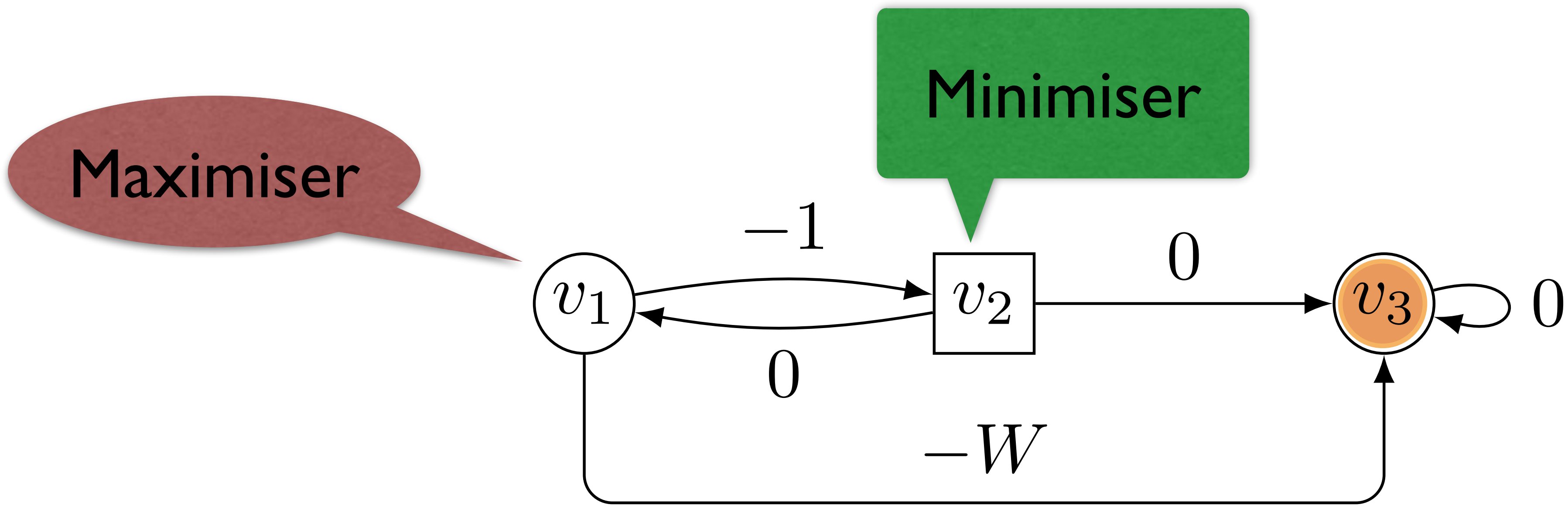
Value iteration on an example



what may both players achieve in 1 step

$+\infty$	$+\infty$	0
$+\infty$	0	0
–1	0	0
–1	–1	0
–2	–1	0
…	…	…
$-W$	$-W$	0

Value iteration on an example

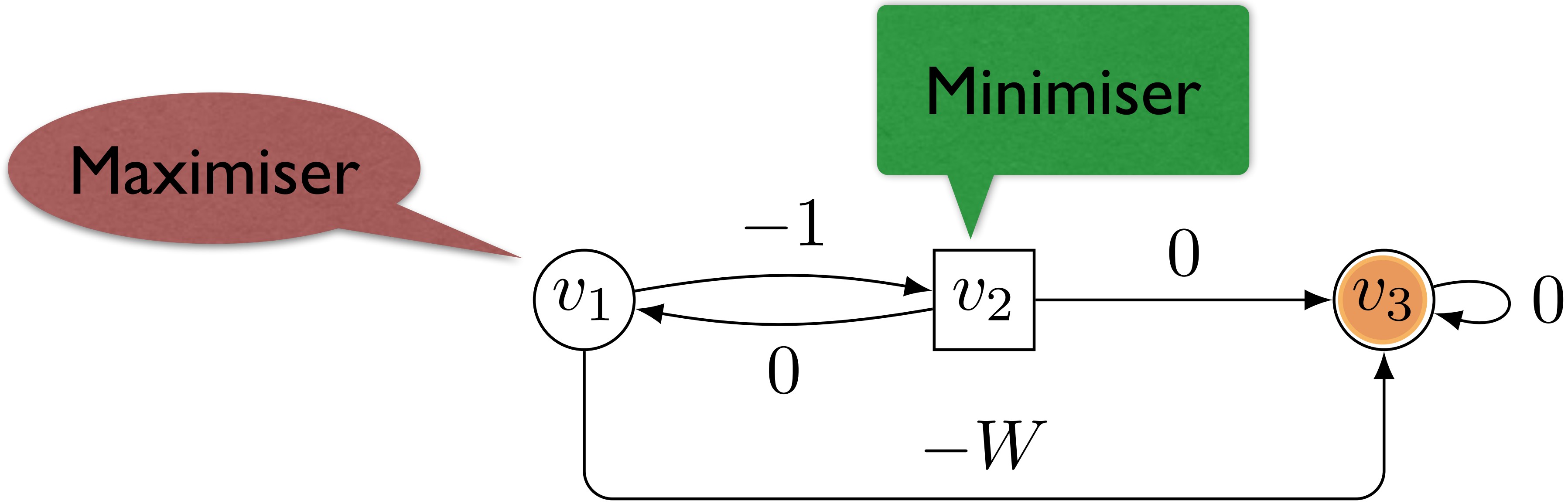


what may both players achieve in 1 step

$+\infty$	$+\infty$	0
$+\infty$	0	0
–1	0	0
–1	–1	0
–2	–1	0
...	...	...
–W	–W	0
–W	–W	0

stabilisation is proved always to happen in pseudo-polynomial time and the result is the value

Value iteration on an example



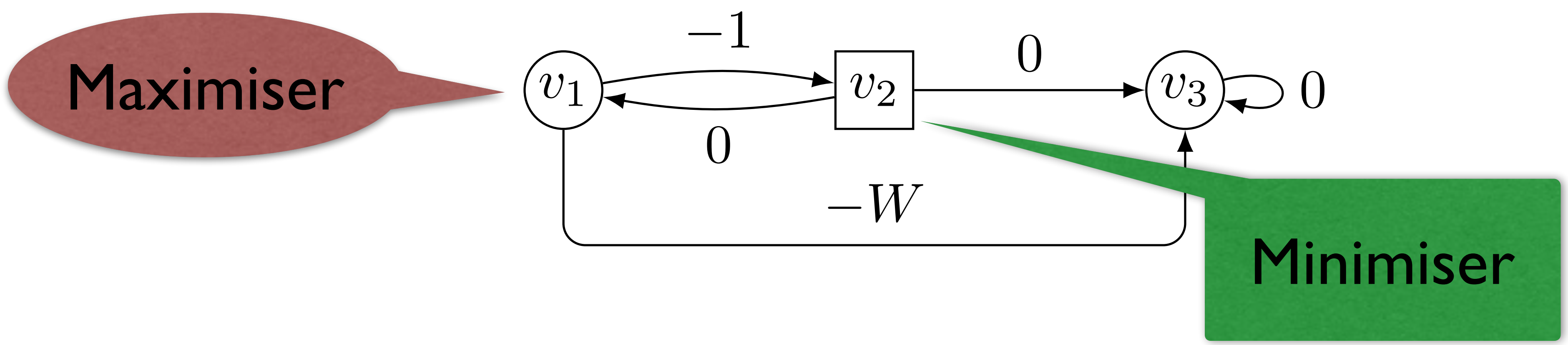
what may both players achieve in 1 step

stabilisation is proved always to happen in pseudo-polynomial time and the result is the value

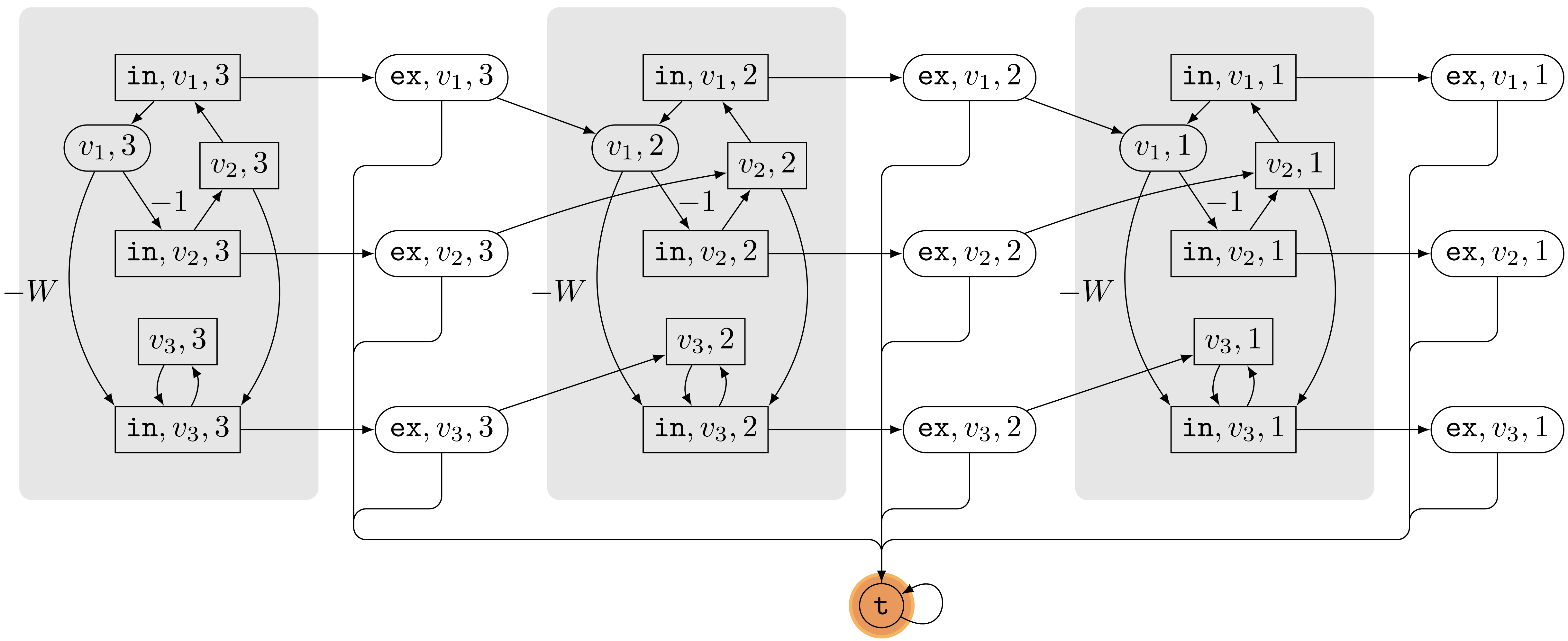
$+\infty$	$+\infty$	0
$+\infty$	0	0
-1	0	0
-1	-1	0
-2	-1	0
...	...	...
$-W$	$-W$	0
$-W$	$-W$	0

Strategy of Minimiser

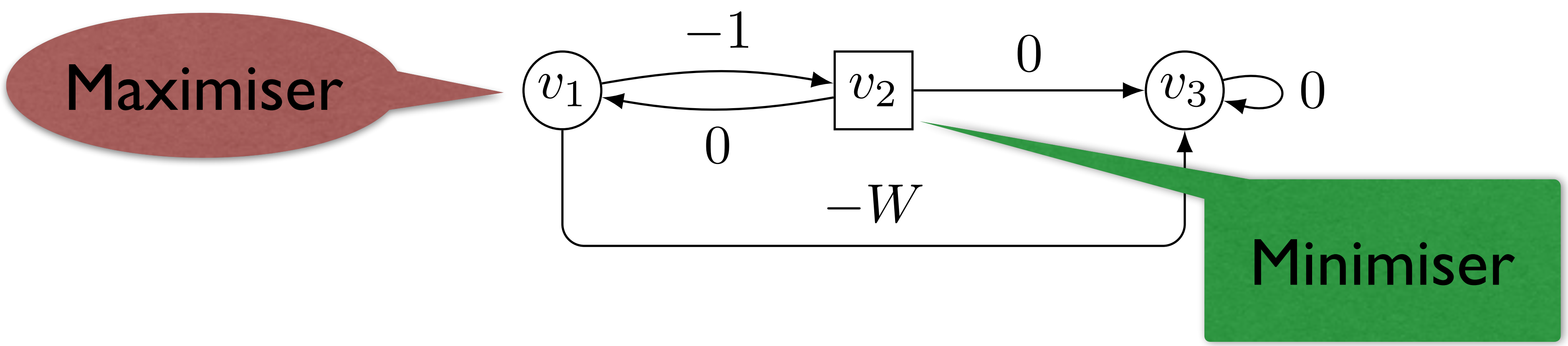
From total-payoff to MCR games



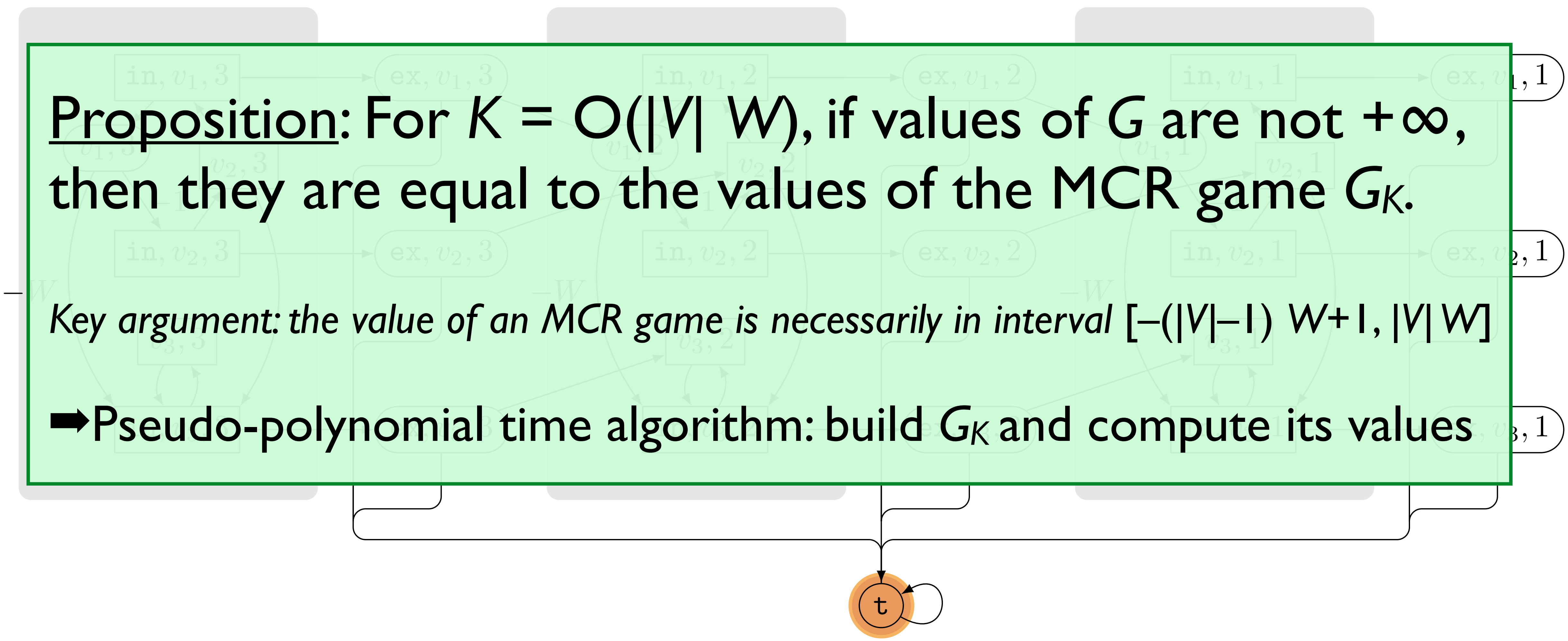
- For all *K*, unfold *K* times the arena, allowing Minimiser to ask to go to target t



From total-payoff to MCR games



- For all *K*, unfold *K* times the arena, allowing Minimiser to ask to go to target t

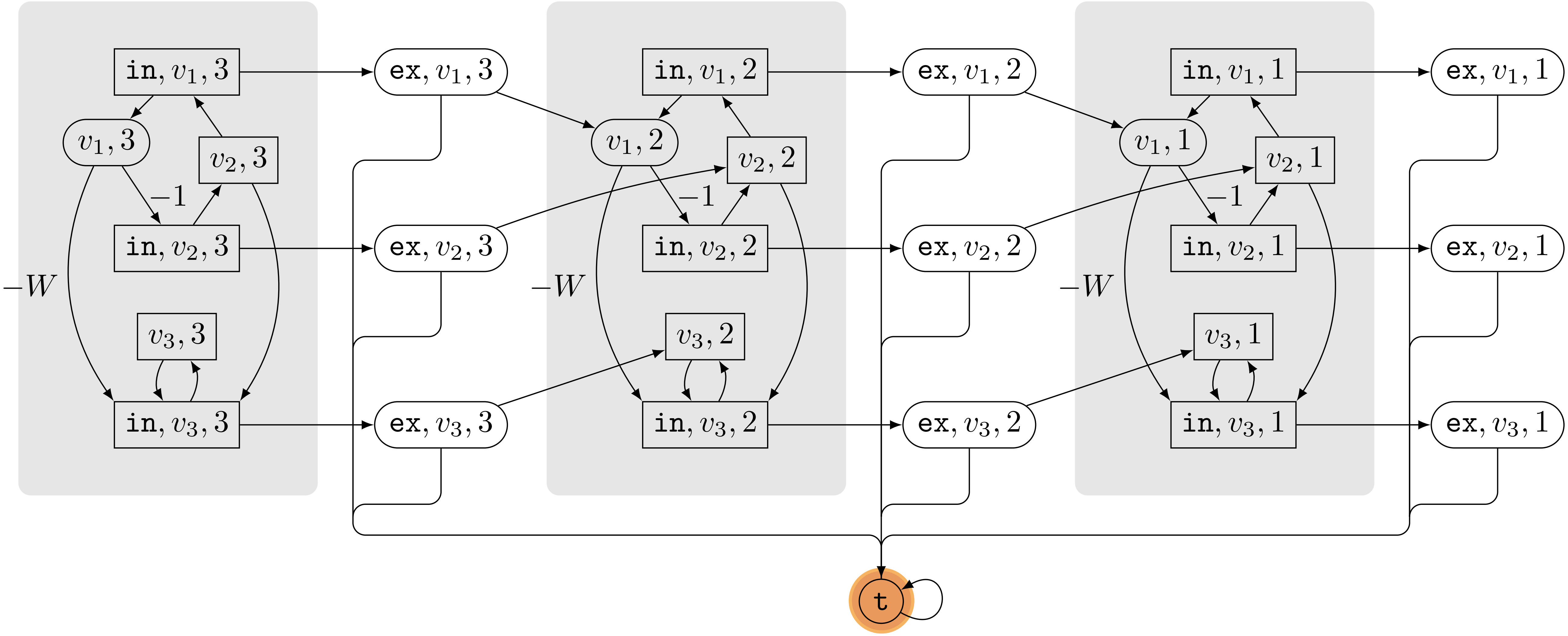


<u>Proposition</u>: For *K* = O(|*V*| *W*), if values of *G* are not +∞, then they are equal to the values of the MCR game G_K.

Key argument: the value of an MCR game is necessarily in interval [–(|*V*|–1) *W*+1, |*V*| *W*]

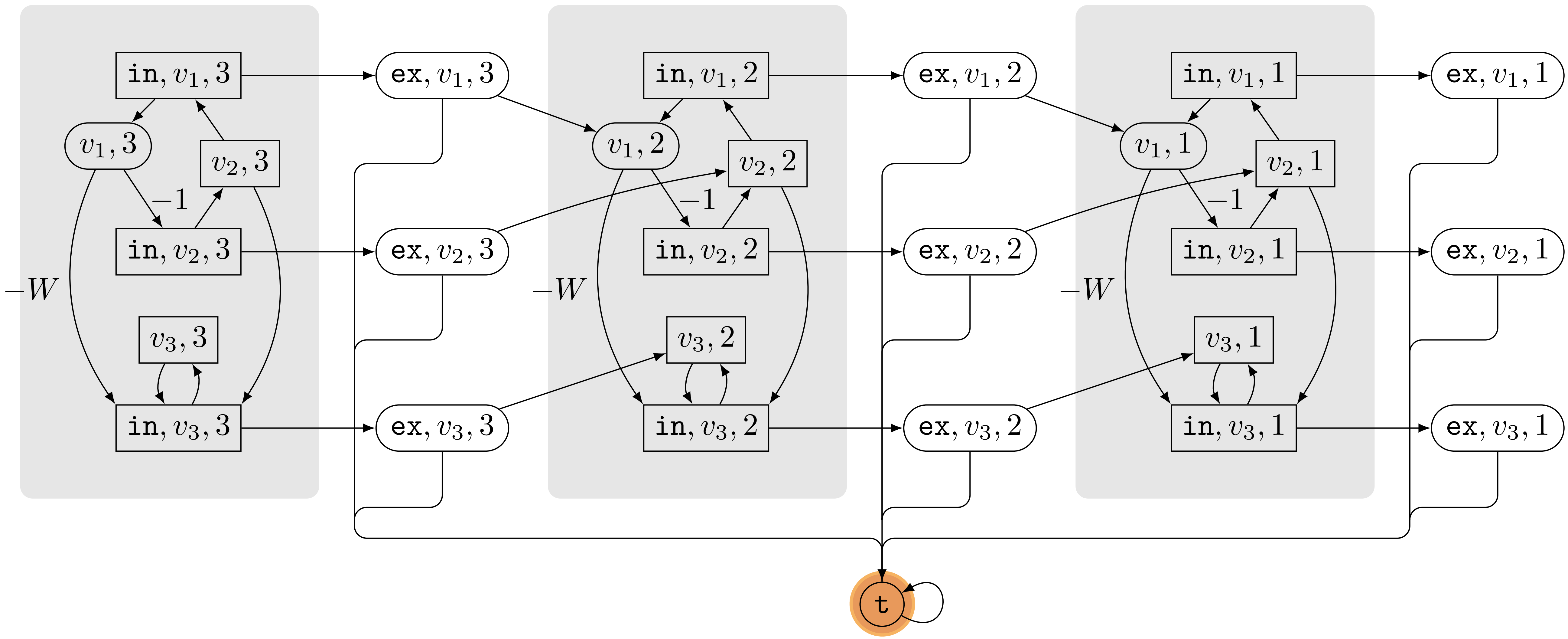
➡Pseudo-polynomial time algorithm: build G_K and compute its values

How not to build G_K?...



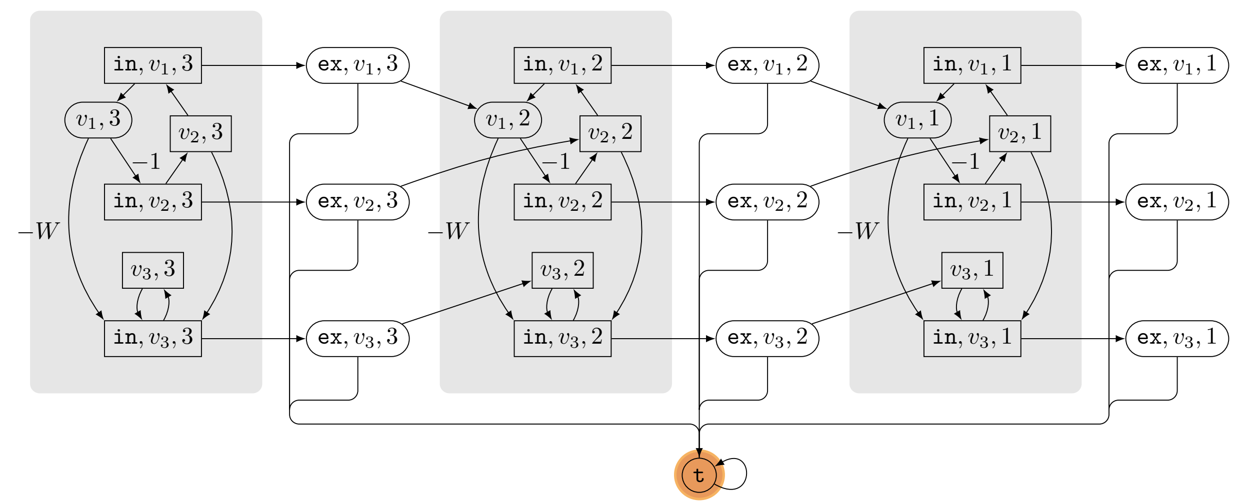
How not to build G_K?...

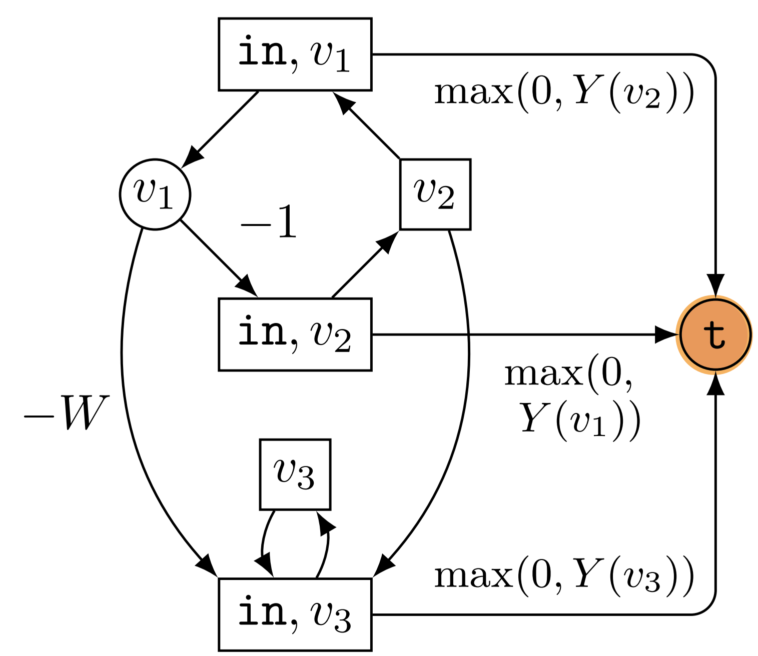
- In the value iteration for MCR games, we may compute the value from the last copy of the game to the first one (outer loop)



How not to build G_K?…

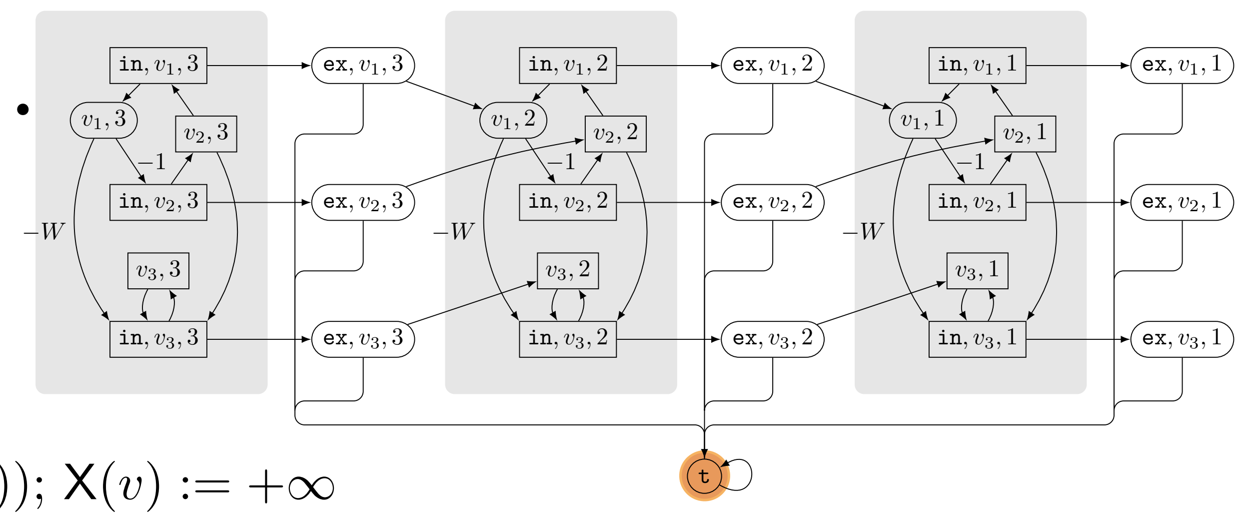
- In the value iteration for MCR games, we may compute the value from the last copy of the game to the first one (outer loop)
- Each time, it is the same arena: only the *exit* values evolve… Compute the values of a linear size MCR game (inner loop)



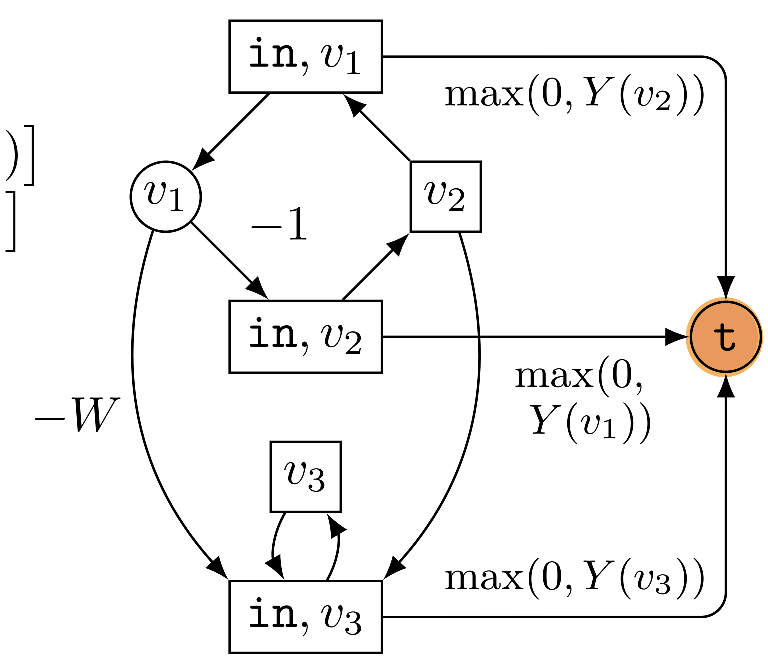


How not to build G_K?…

- In the value iteration for MCR games, we may compute the value from the last copy of the game to the first one (outer loop)
- Each time, it is the same arena: only the *exit* values evolve… Compute the values of a linear size MCR game (inner loop)
- Stop early inner and outer loops…



foreach $v \in V$ **do** $\mathsf{Y}(v) := -\infty$
repeat
  foreach $v \in V$ **do** $\mathsf{Y}_{pre}(v) := \mathsf{Y}(v)$; $\mathsf{Y}(v) := \max(0, \mathsf{Y}(v))$; $\mathsf{X}(v) := +\infty$
  repeat
    $\mathsf{X}_{pre} := \mathsf{X}$
    foreach $v \in V_{\mathsf{Max}}$ **do** $\mathsf{X}(v) := \max_{v' \in E(v)} \left[\omega(v, v') + \min(\mathsf{X}_{pre}(v'), \mathsf{Y}(v'))\right]$
    foreach $v \in V_{\mathsf{Min}}$ **do** $\mathsf{X}(v) := \min_{v' \in E(v)} \left[\omega(v, v') + \min(\mathsf{X}_{pre}(v'), \mathsf{Y}(v'))\right]$
    foreach $v \in V$ such that $\mathsf{X}(v) < -(|V| - 1)W$ **do** $\mathsf{X}(v) := -\infty$
  until $\mathsf{X} = \mathsf{X}_{pre}$
  $\mathsf{Y} := \mathsf{X}$
  foreach $v \in V$ such that $\mathsf{Y}(v) > (|V| - 1)W$ **do** $\mathsf{Y}(v) := +\infty$
until $\mathsf{Y} = \mathsf{Y}_{pre}$
return Y



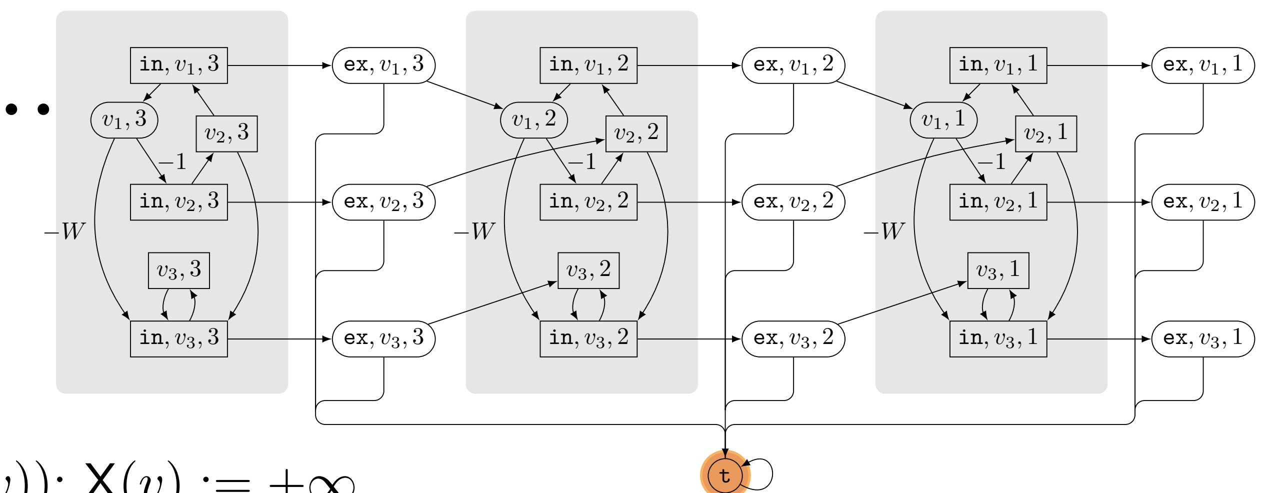
How not to build G_K?...

- In the value iteration for MCR games, we may compute the value from the last copy of the game to the first one (outer loop)
- Each time, it is the same arena: only the *exit* values evolve… Compute the values of a linear size MCR game (inner loop)

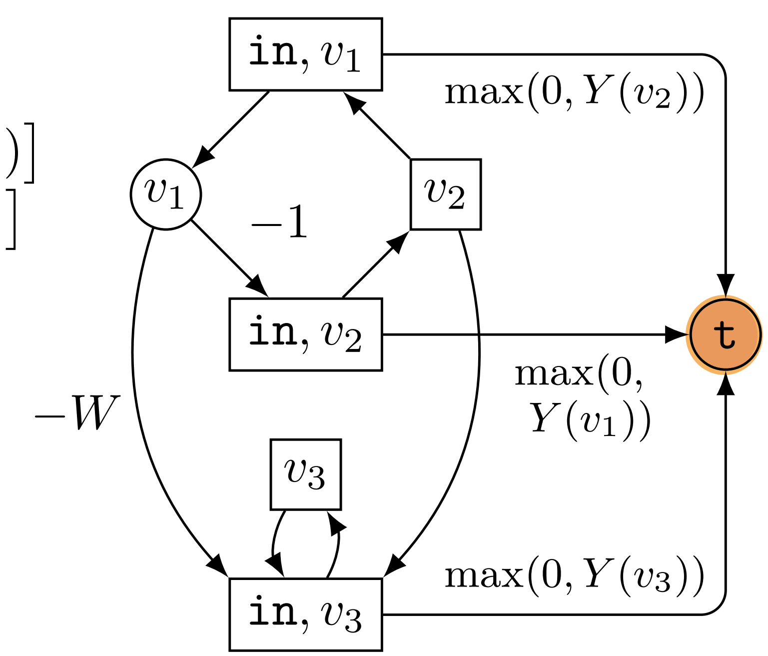
Requires very few memory (no need to construct G_K)

➡Pseudo-polynomial time: $O(|V|^4\ |E|\ W^2)$

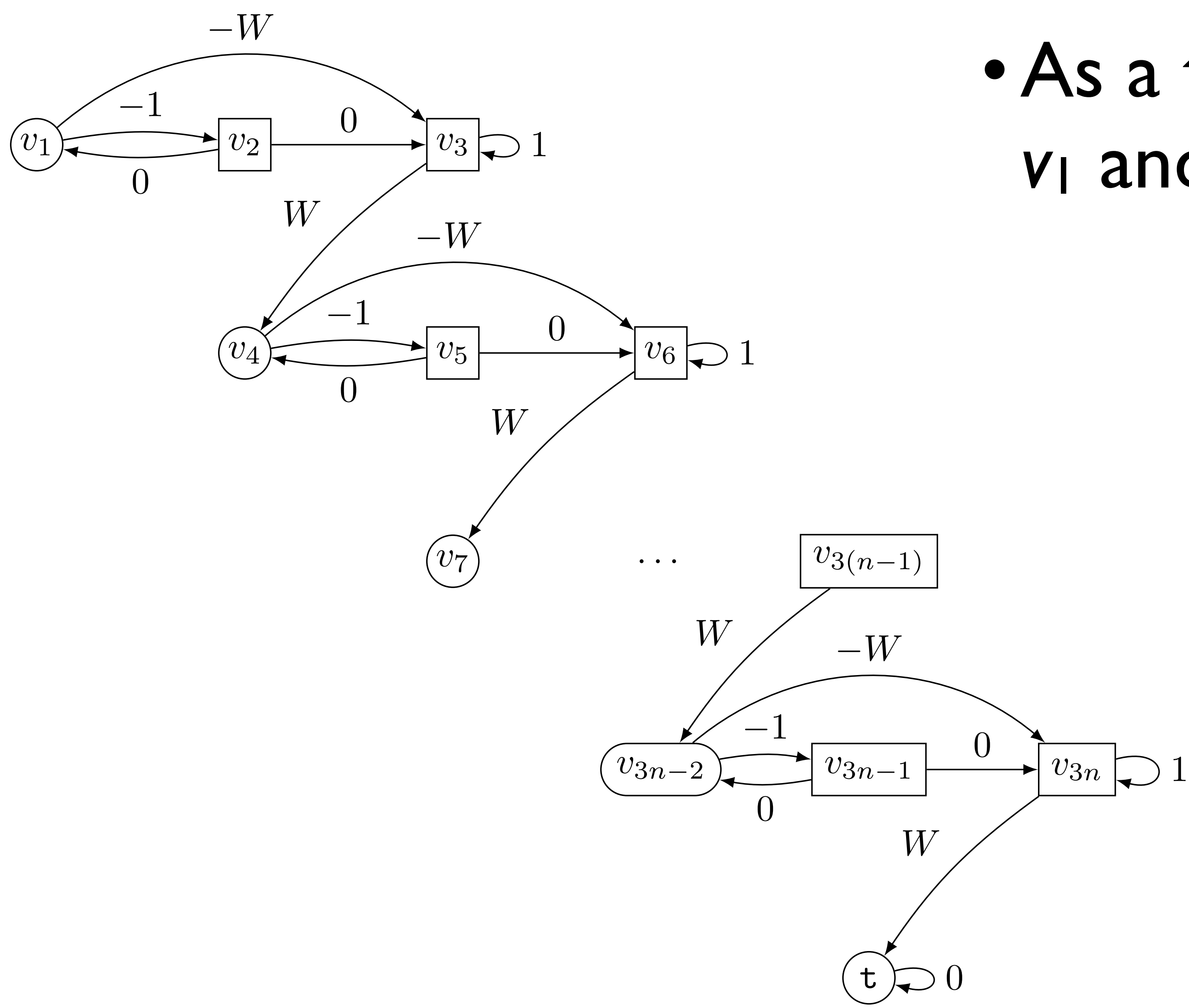
- Stop early inner and outer loops…



foreach $v \in V$ **do** $\mathsf{Y}(v) := -\infty$
repeat
  foreach $v \in V$ **do** $\mathsf{Y}_{pre}(v) := \mathsf{Y}(v)$; $\mathsf{Y}(v) := \max(0, \mathsf{Y}(v))$; $\mathsf{X}(v) := +\infty$
  repeat
    $\mathsf{X}_{pre} := \mathsf{X}$
    foreach $v \in V_{\mathsf{Max}}$ **do** $\mathsf{X}(v) := \max_{v' \in E(v)} \left[\omega(v, v') + \min(\mathsf{X}_{pre}(v'), \mathsf{Y}(v'))\right]$
    foreach $v \in V_{\mathsf{Min}}$ **do** $\mathsf{X}(v) := \min_{v' \in E(v)} \left[\omega(v, v') + \min(\mathsf{X}_{pre}(v'), \mathsf{Y}(v'))\right]$
    foreach $v \in V$ such that $\mathsf{X}(v) < -(|V| - 1)W$ **do** $\mathsf{X}(v) := -\infty$
  until $\mathsf{X} = \mathsf{X}_{pre}$
  $\mathsf{Y} := \mathsf{X}$
  foreach $v \in V$ such that $\mathsf{Y}(v) > (|V| - 1)W$ **do** $\mathsf{Y}(v) := +\infty$
until $\mathsf{Y} = \mathsf{Y}_{pre}$
return Y

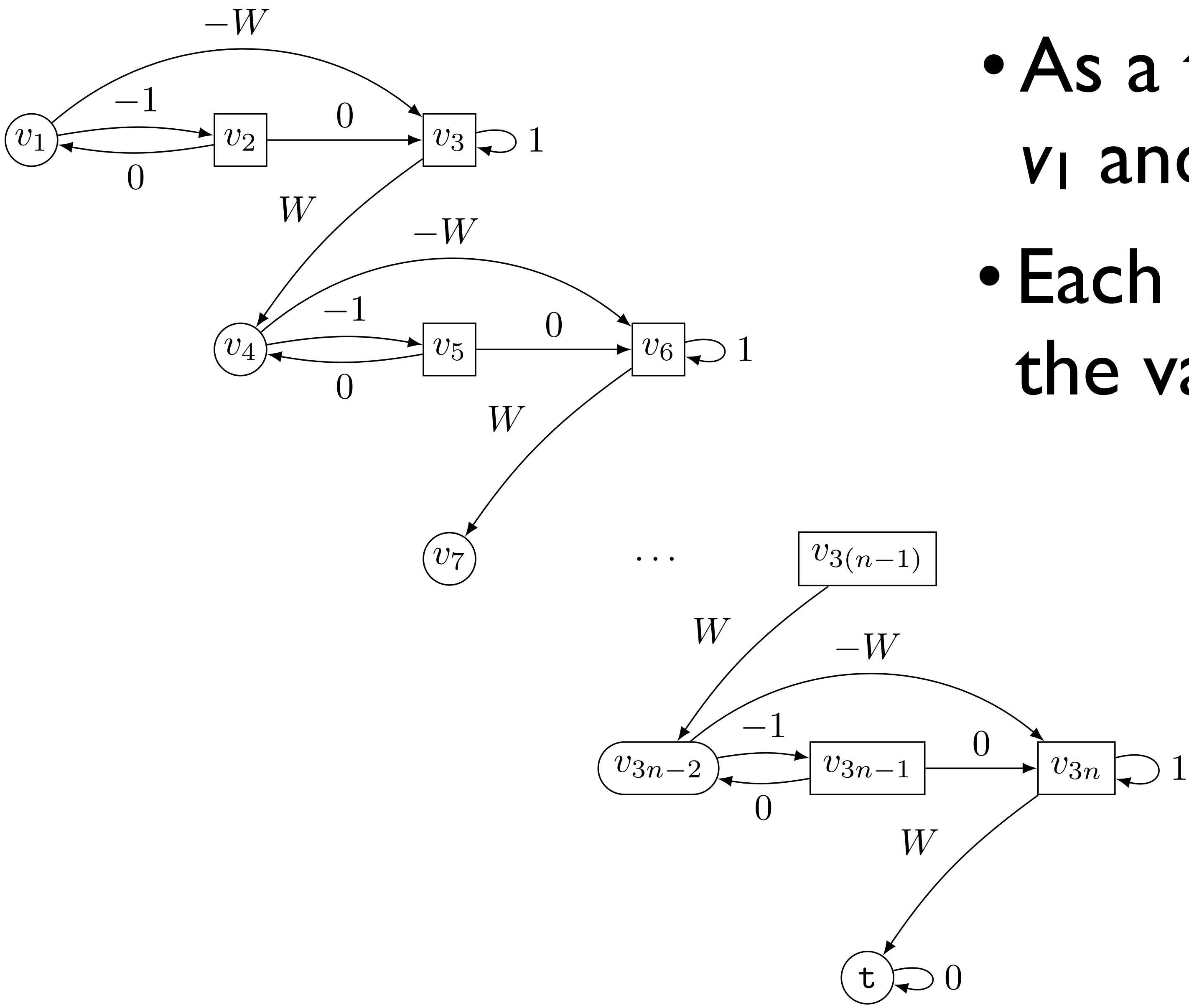


On an example



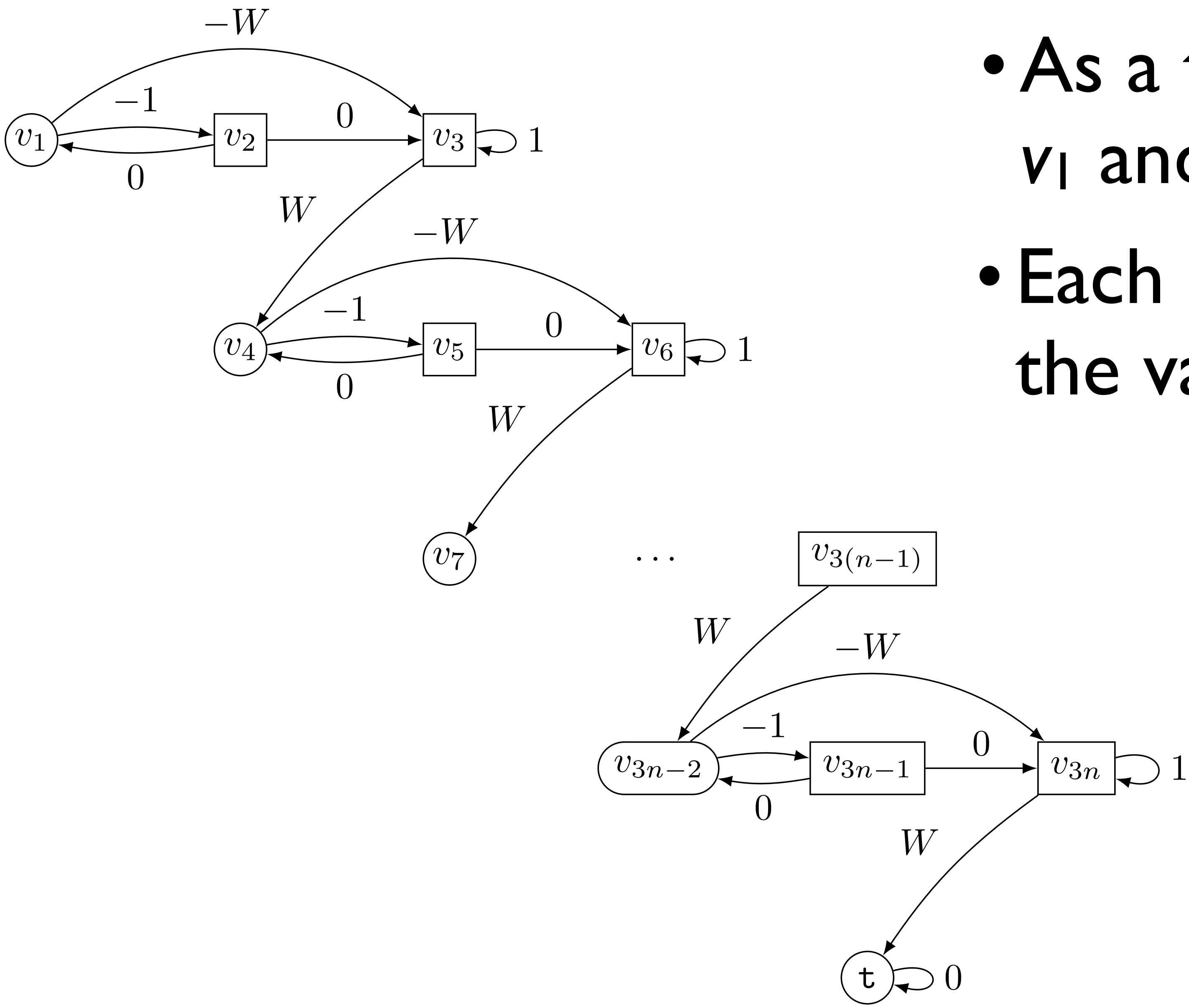
- As a total-payoff game, values of v_1 and v_2 are 0, value of v_3 is W…

On an example



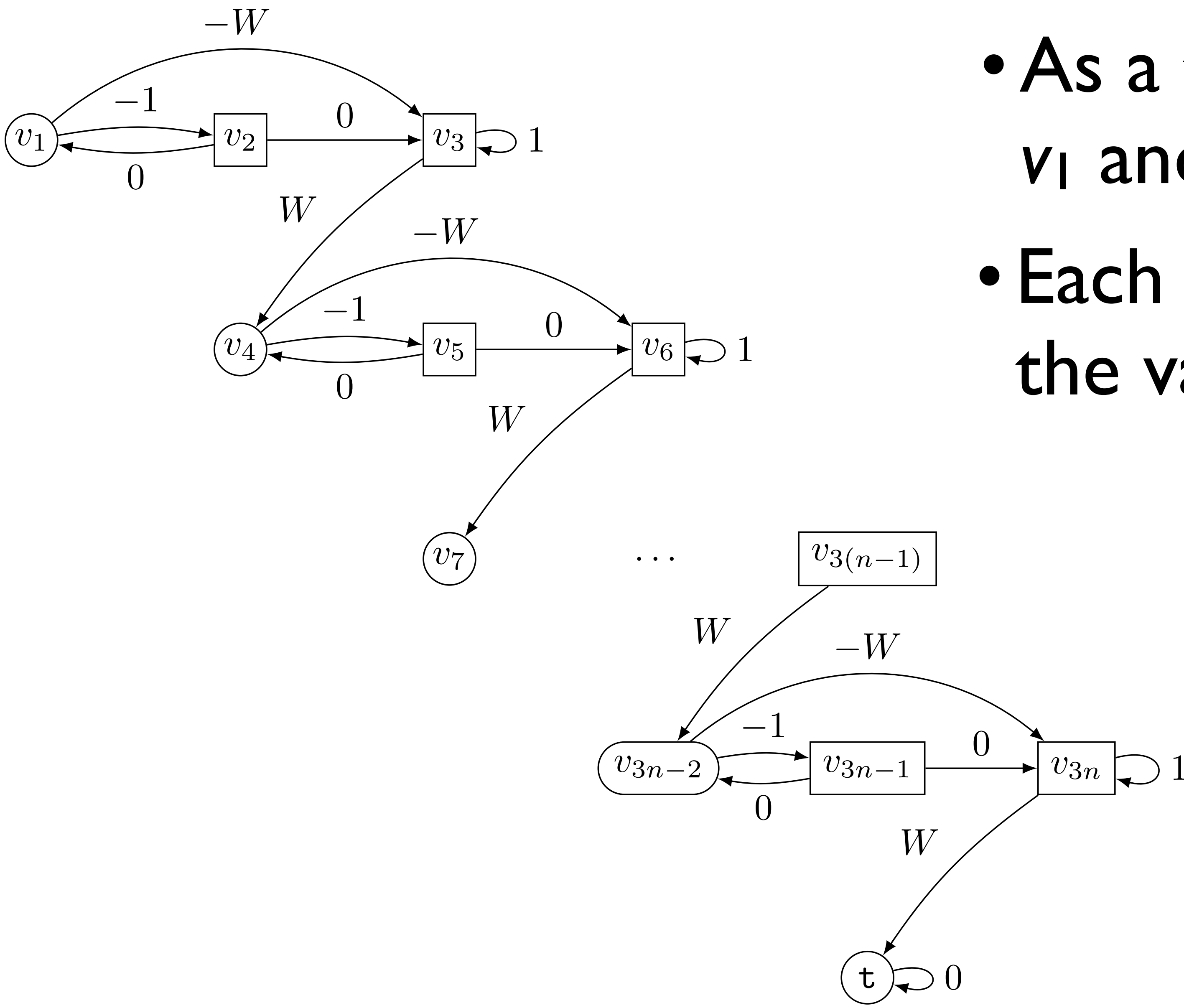
- As a total-payoff game, values of v_1 and v_2 are 0, value of v_3 is W…
- Each inner loop computes all the values (needs $O(nW)$ steps)

On an example



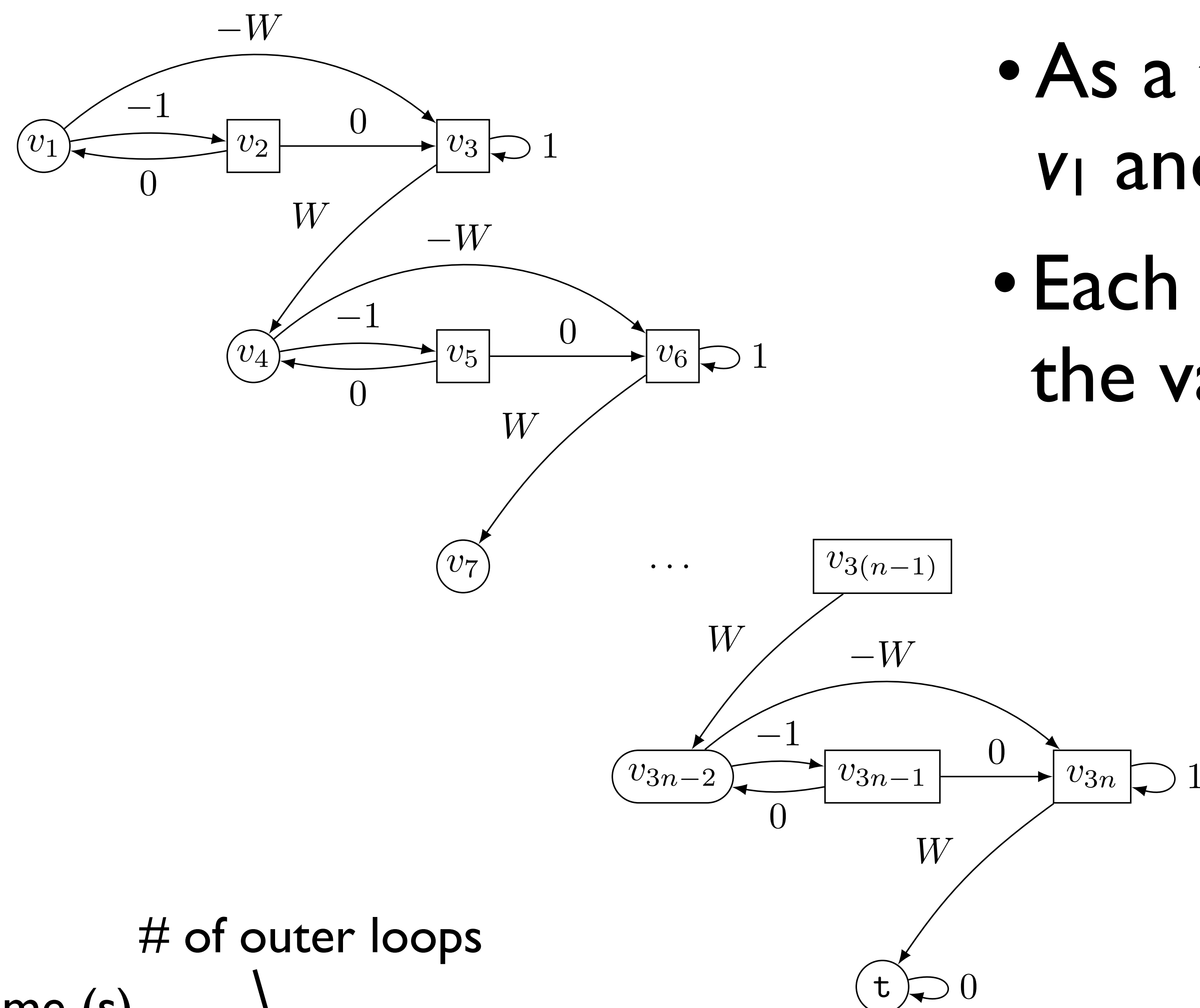
- As a total-payoff game, values of v_1 and v_2 are 0, value of v_3 is W…
- Each inner loop computes all the values (needs $O(n\,W)$ steps)
  - After 1 outer loop, only the values of the last component are correct…

On an example



- As a total-payoff game, values of v_1 and v_2 are 0, value of v_3 is W…
- Each inner loop computes all the values (needs $O(n\,W)$ steps)
  - After 1 outer loop, only the values of the last component are correct…
  - Requires n outer loops to converge

On an example



- As a total-payoff game, values of v_1 and v_2 are 0, value of v_3 is W…
- Each inner loop computes all the values (needs $O(n\,W)$ steps)
- After 1 outer loop, only the values of the last component are correct…
- Requires n outer loops to converge

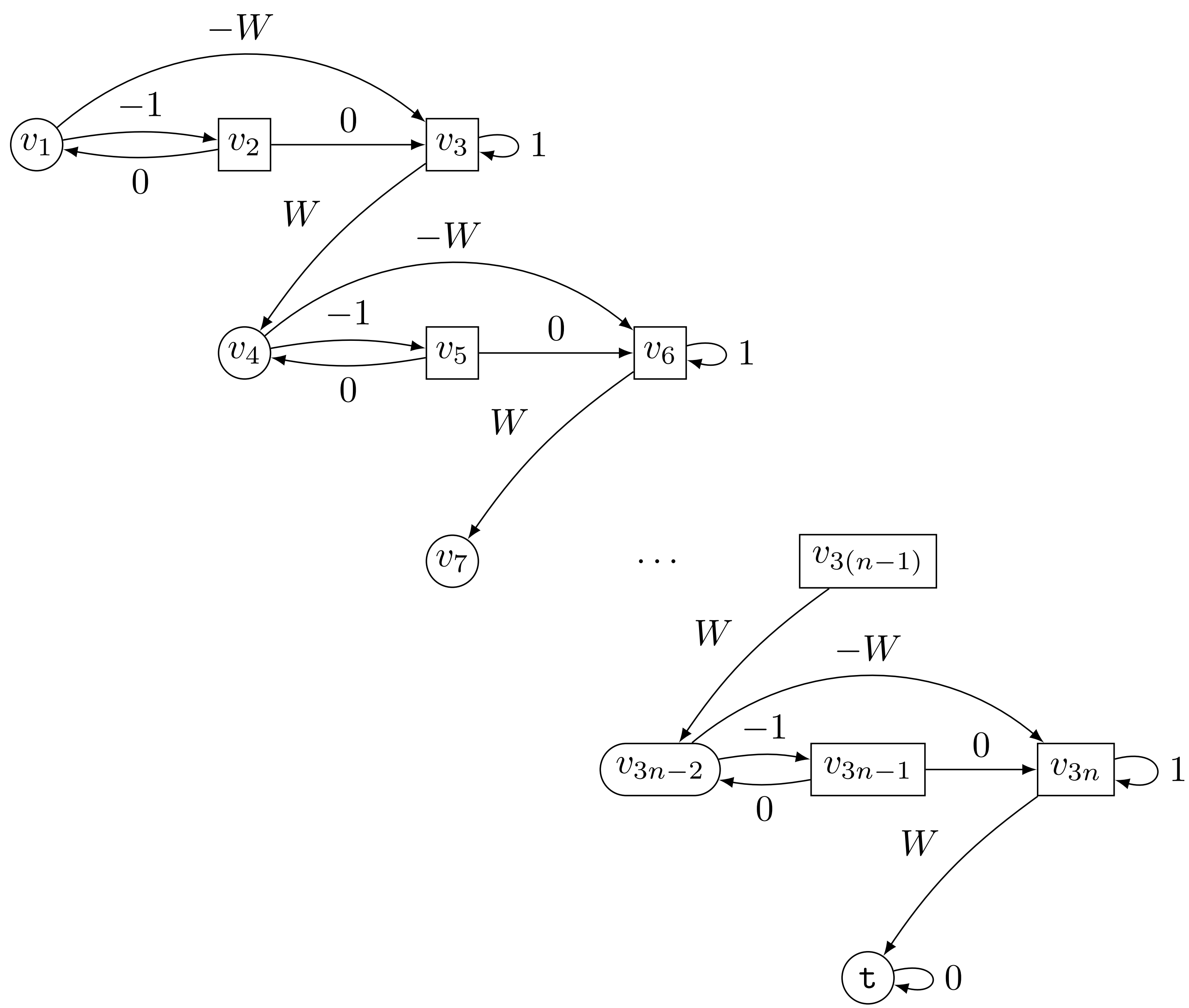
# of outer loops

time (s)

total # of inner loops

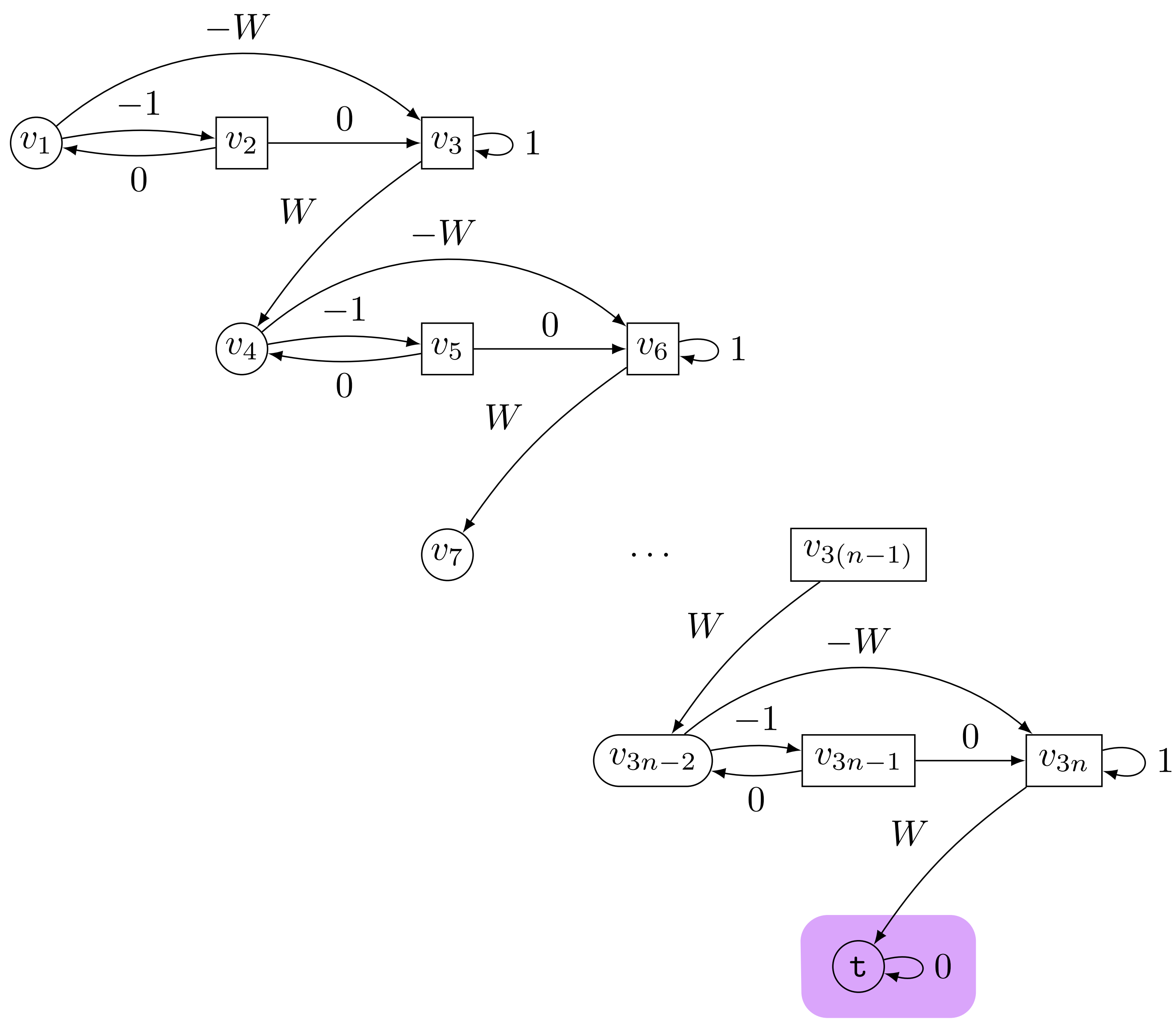
$W \backslash n$	100	200	300	400	500
50	0.52 / 151 / 12603	1.90 / 251 / 22703	3.84 / 351 / 32803	6.05 / 451 / 42903	9.83 / 551 / 53003
100	1.00 / 201 / 30103	3.48 / 301 / 50203	8.64 / 401 / 70303	13.53 / 501 / 90403	22.64 / 601 / 110503
150	1.89 / 251 / 52603	6.02 / 351 / 82703	12.88 / 451 / 112803	22.13 / 551 / 142903	34.16 / 651 / 173003
200	2.96 / 301 / 80103	9.62 / 401 / 120203	18.33 / 501 / 160303	30.42 / 601 / 200403	45.64 / 701 / 240503
250	3.92 / 351 / 112603	13.28 / 451 / 162703	25.18 / 551 / 212803	46.23 / 651 / 262903	71.51 / 751 / 313003

Heuristics: compute as little as possible!



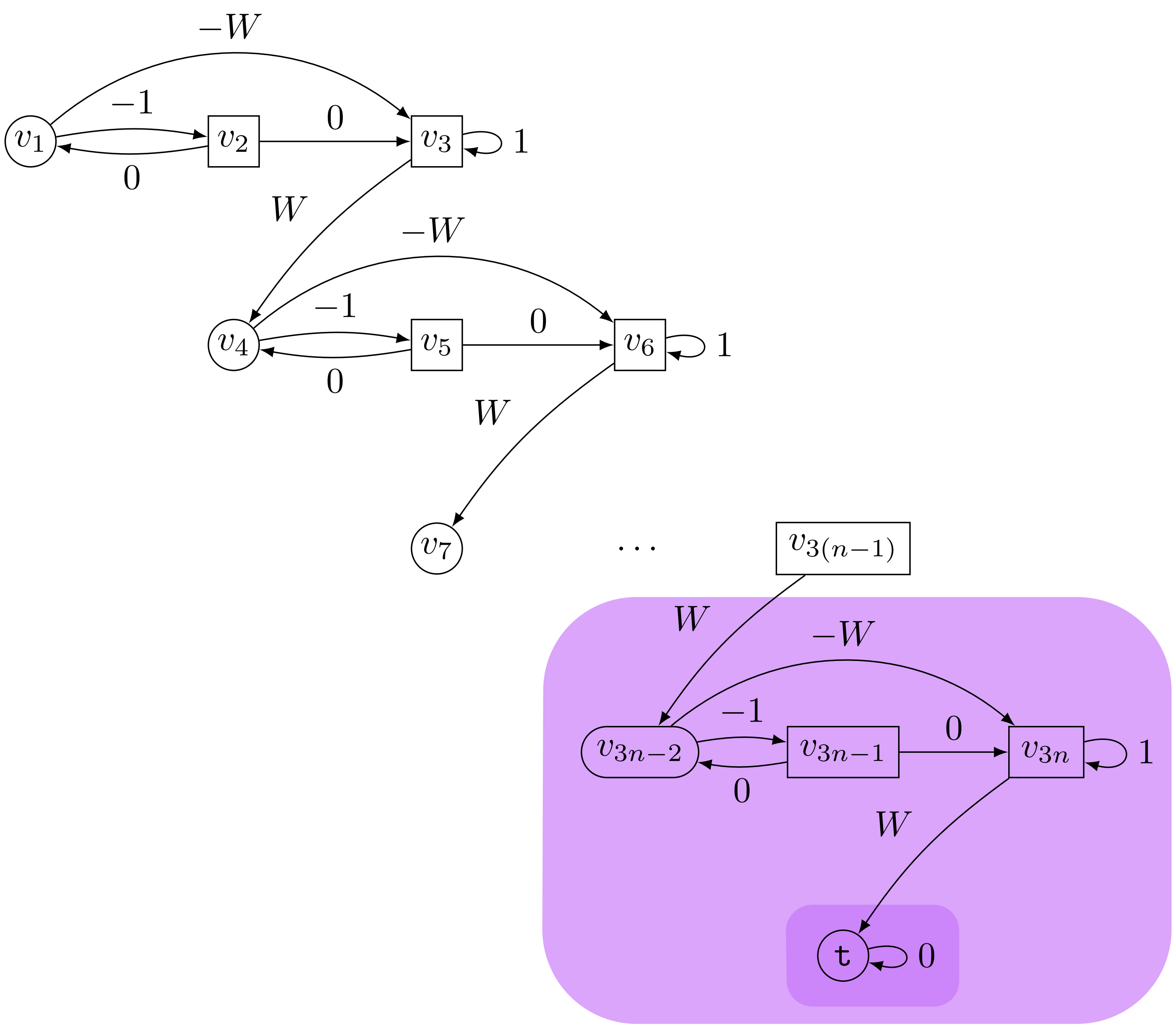
- In the outer loop, compute SCC by SCC

Heuristics: compute as little as possible!



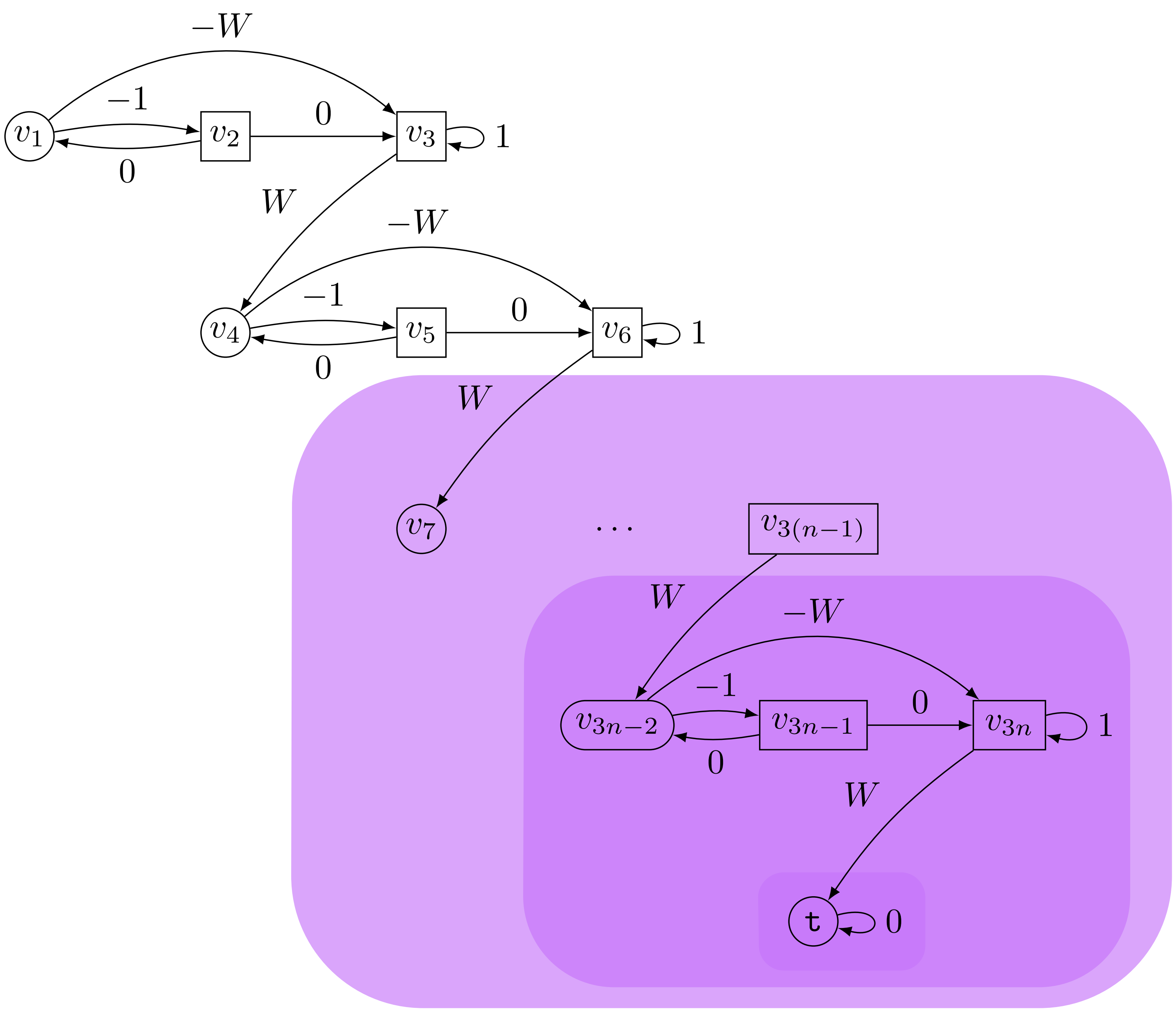
- In the outer loop, compute SCC by SCC

Heuristics: compute as little as possible!



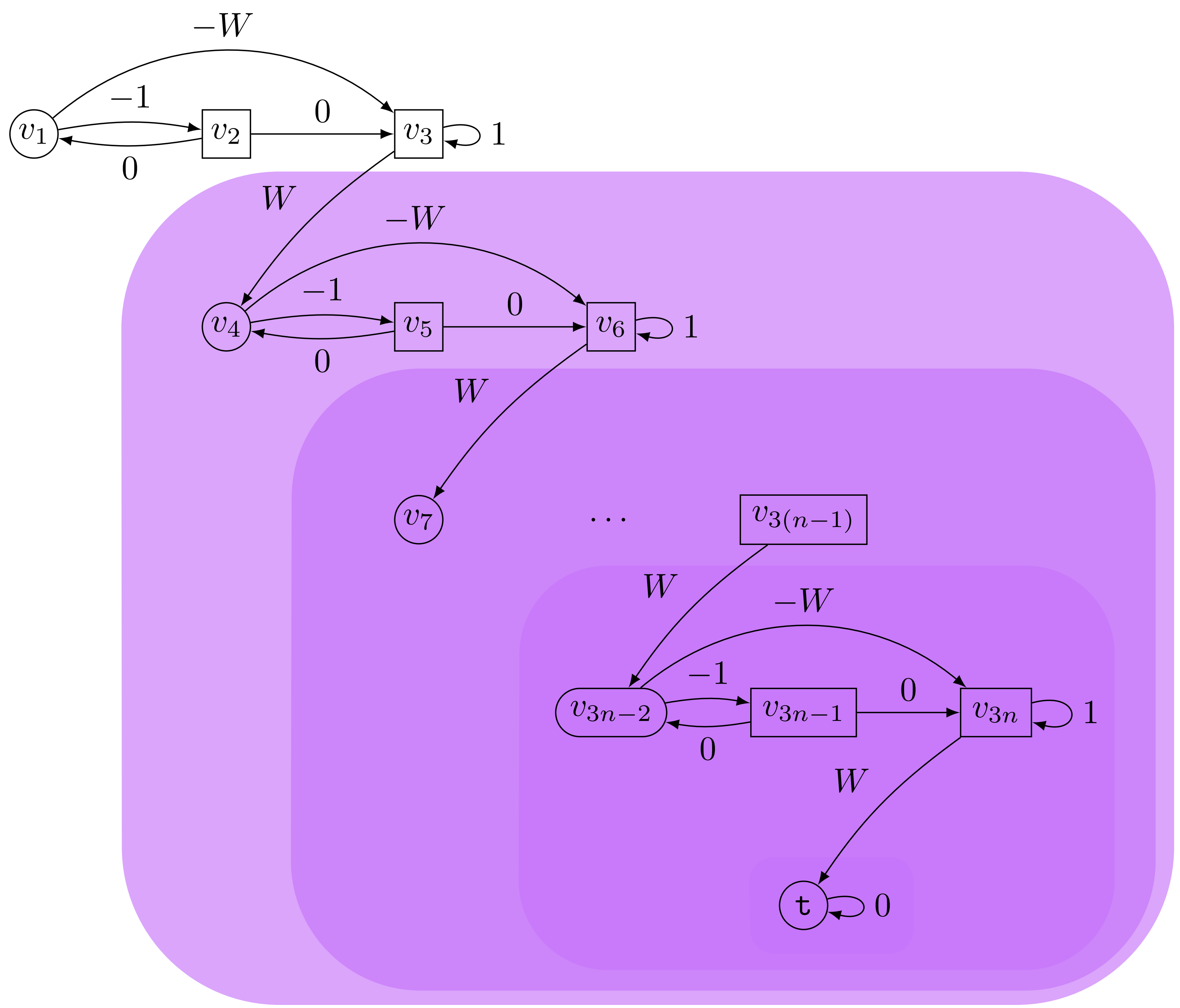
- In the outer loop, compute SCC by SCC

Heuristics: compute as little as possible!



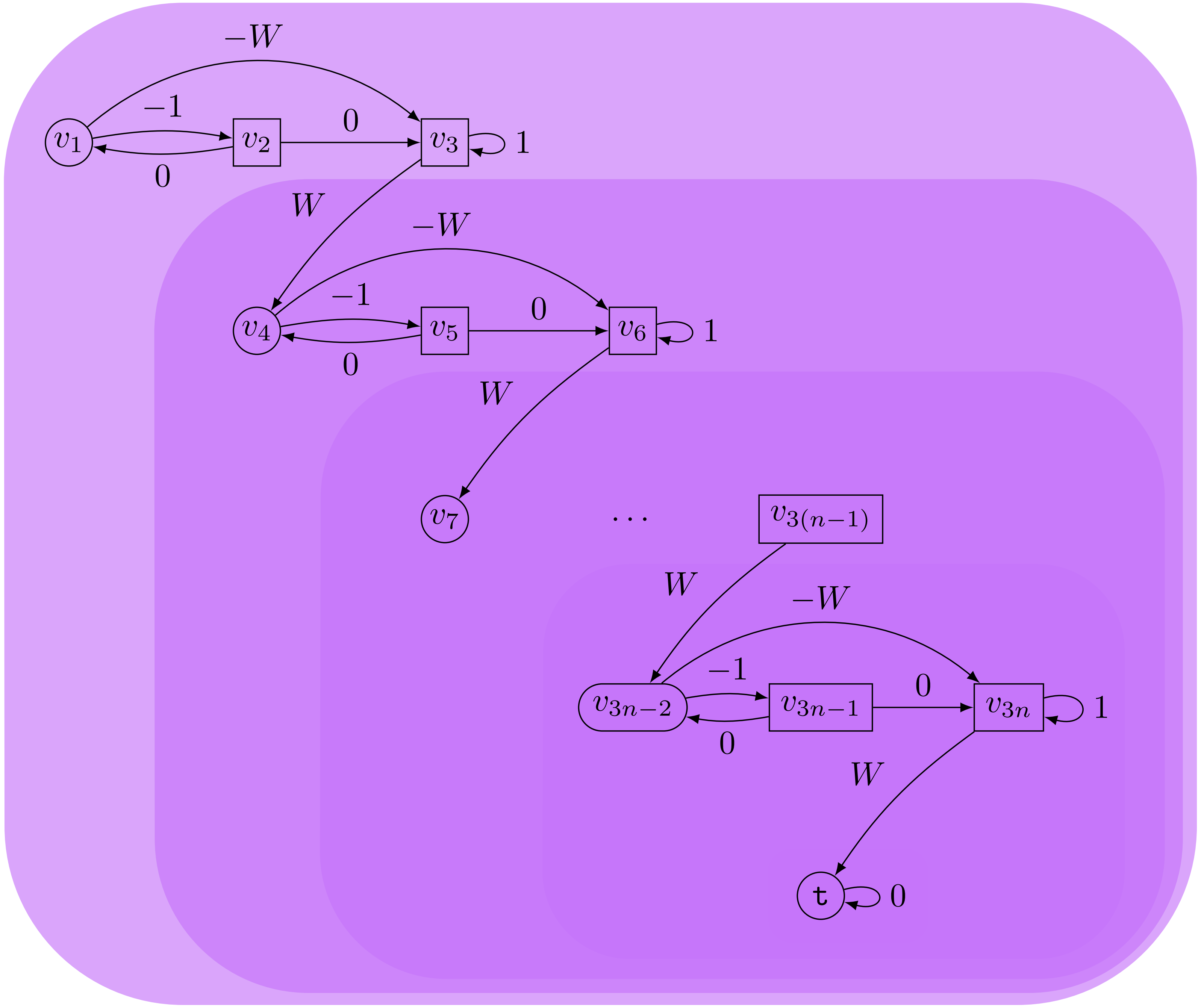
- In the outer loop, compute SCC by SCC

Heuristics: compute as little as possible!



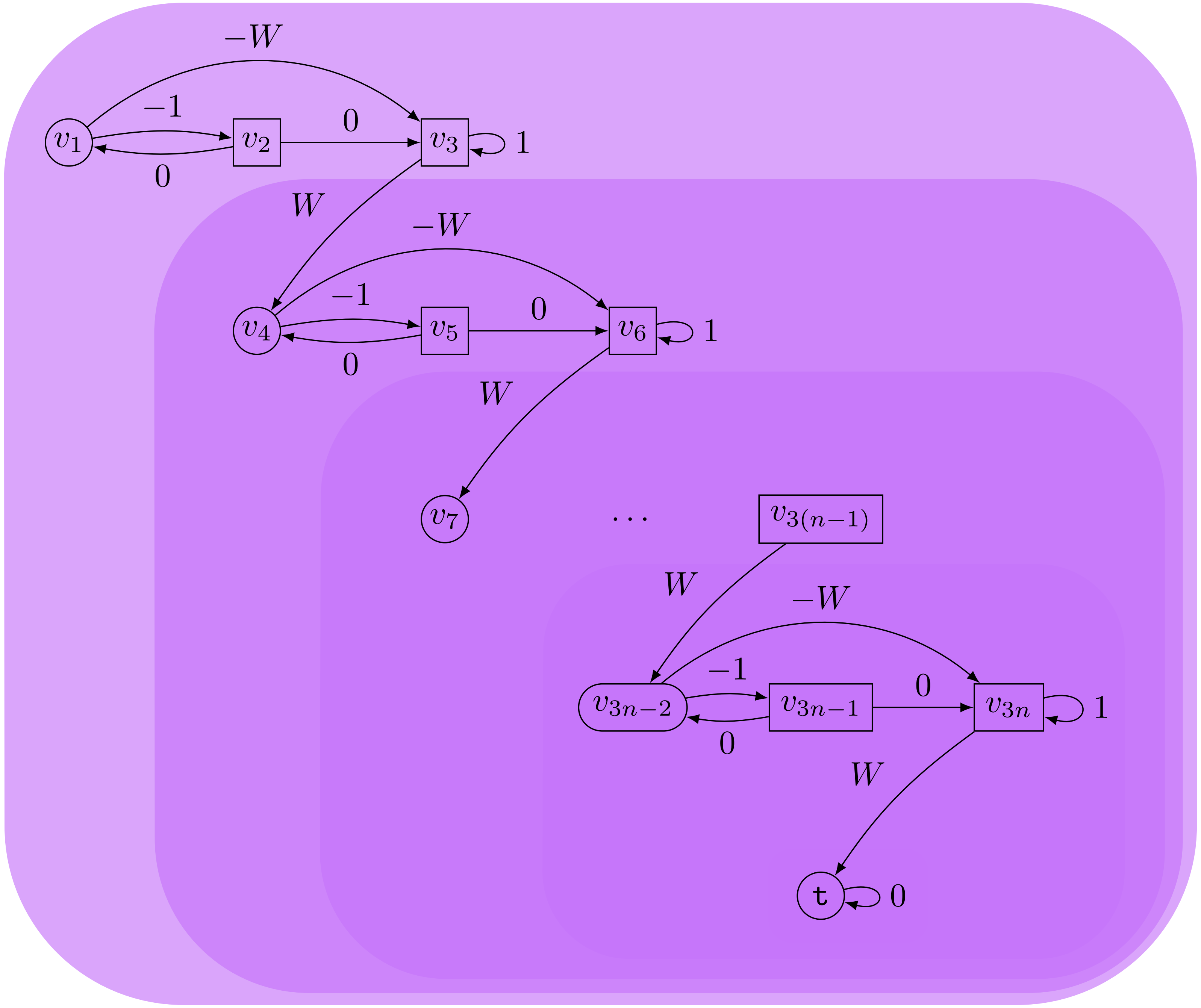
- In the outer loop, compute SCC by SCC

Heuristics: compute as little as possible!



- In the outer loop, compute SCC by SCC

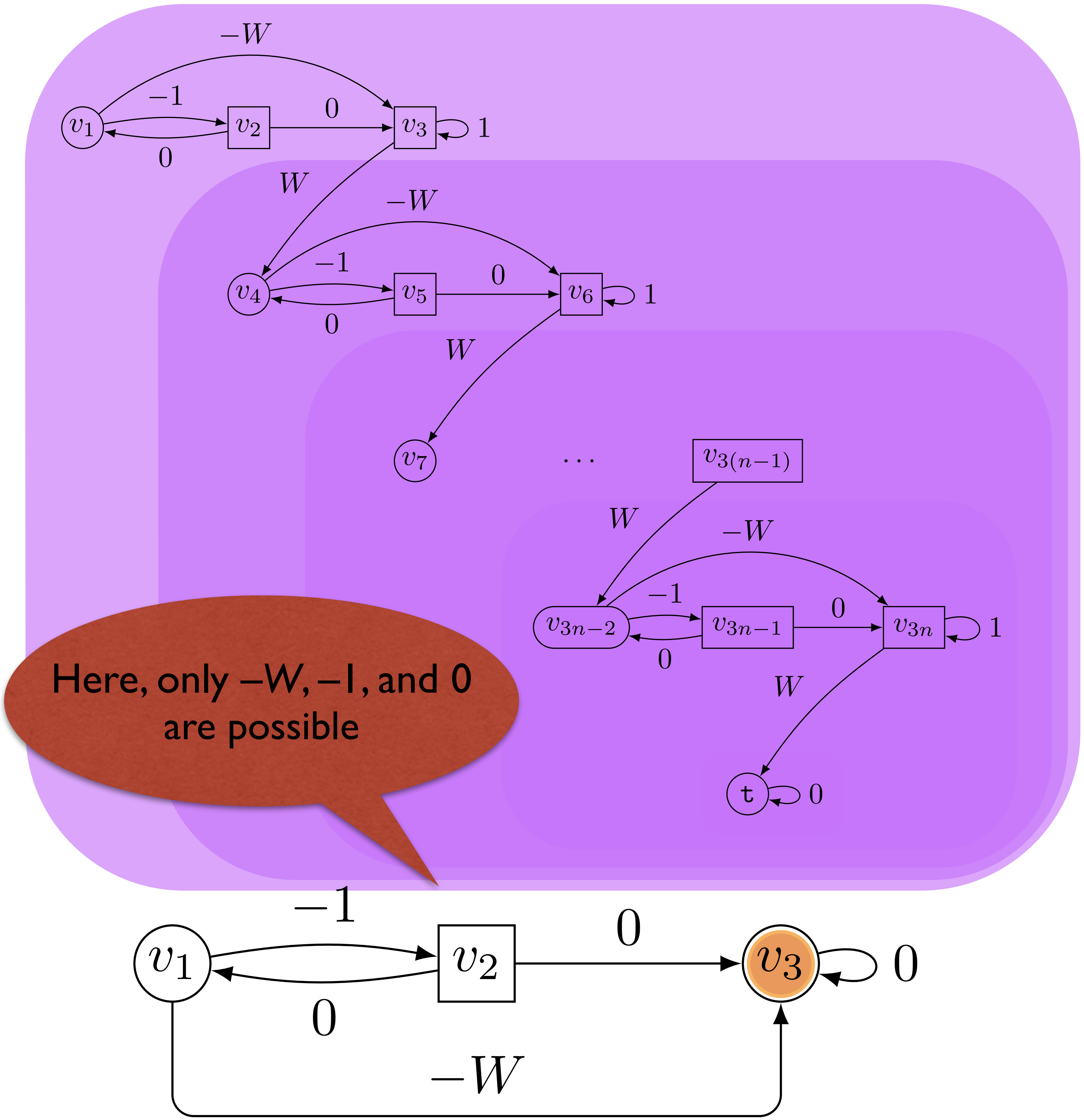
Heuristics: compute as little as possible!



- In the outer loop, compute SCC by SCC
- For each inner loop, we solve an MCR game: optimal memoryless strategies, so value is weight of a simple path…

$+\infty$	$+\infty$	0
$+\infty$	0	0
-1	0	0
-1	-1	0
-2	-1	0
-2	-2	0
-3	-2	0
-3	-3	0
…	…	…
$-W$	$-W$	0
$-W$	$-W$	0

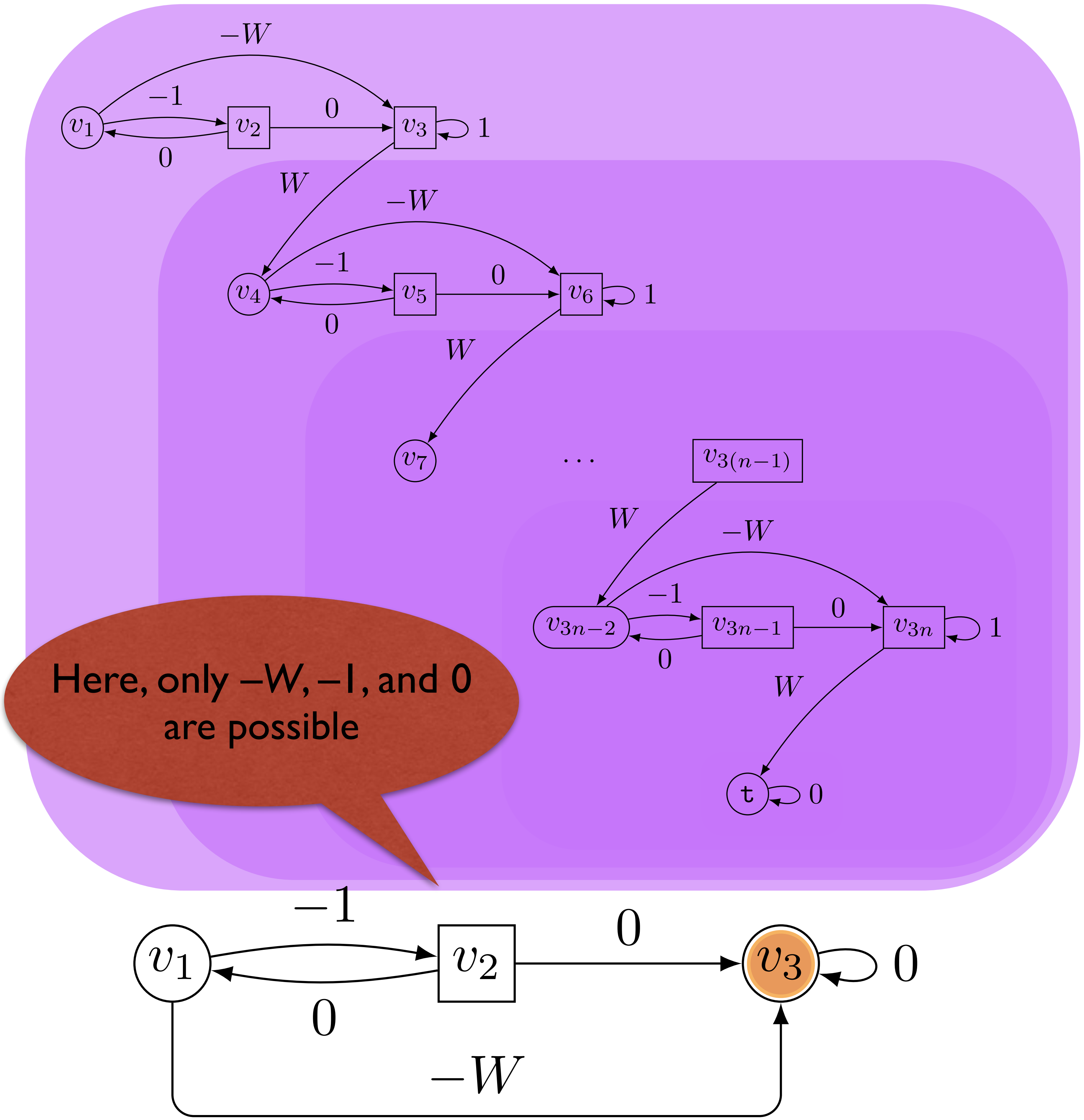
Heuristics: compute as little as possible!



- In the outer loop, compute SCC by SCC
- For each inner loop, we solve an MCR game: optimal memoryless strategies, so value is weight of a simple path…

$+\infty$	$+\infty$	0
$+\infty$	0	0
–1	0	0
–1	–1	0
–2	–1	0
–2	–2	0
–3	–2	0
–3	–3	0
…	…	…
–W	–W	0
–W	–W	0

Heuristics: compute as little as possible!

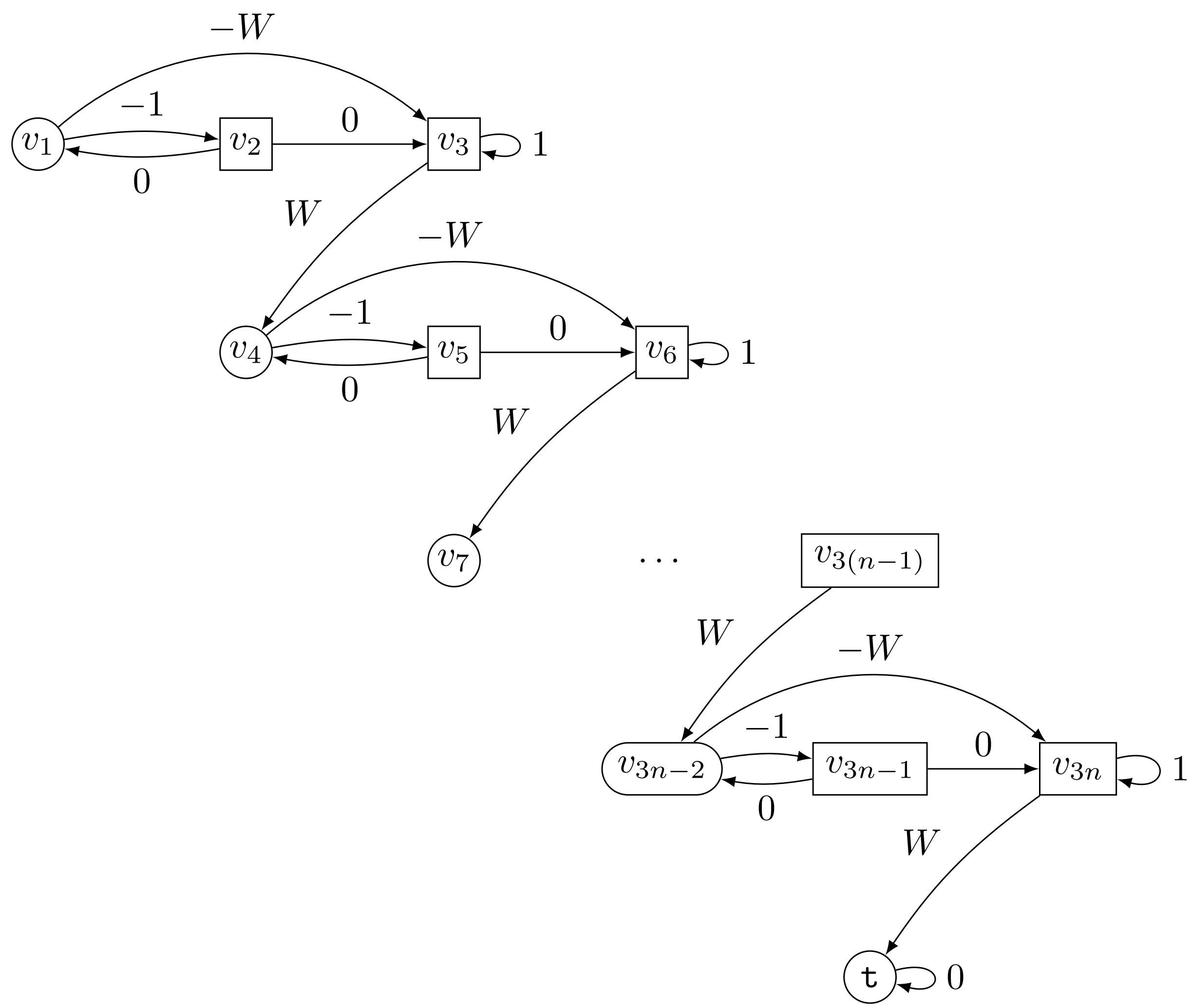


- In the outer loop, compute SCC by SCC
- For each inner loop, we solve an MCR game: optimal memoryless strategies, so value is weight of a simple path…

+∞	+∞	0
+∞	0	0
–1	0	0
–1	–1	0
–2	–1	0
–2	–2	0
–3	–2	0
–3	–3	0
…	…	…
–W	–W	0
–W	–W	0

skip!

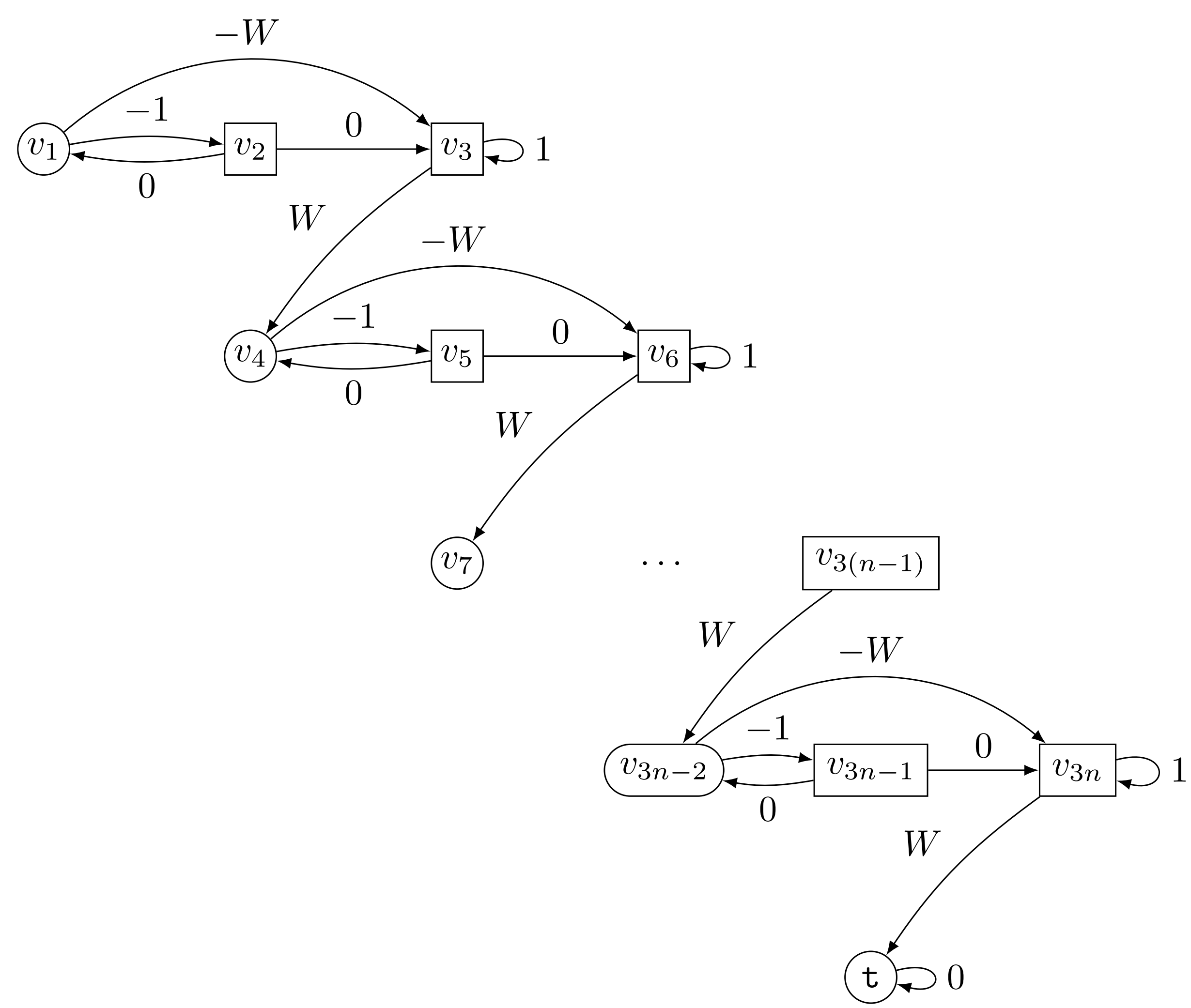
Some total-payoff games in polynomial time



- Combination of both heuristics

- If all SCC uses at most *L* distincts weights (that can be arbitrarily large in absolute values), algorithm with heuristics runs in polynomial time.

Some total-payoff games in polynomial time



- Combination of both heuristics
- If all SCC uses at most *L* distincts weights (that can be arbitrarily large in absolute values), algorithm with heuristics runs in polynomial time.
- Implementation as an add-on to PRISM games available at *http://www.ulb.ac.be/di/verif/monmege/tool/TP- MCR/*

		without heuristics			with heuristics		
W	n	t	k_e	k_i	t	k_e	k_i
50	100	0.52s	151	12,603	0.01s	402	1,404
50	500	9.83s	551	53,003	0.42s	2,002	7,004
200	100	2.96s	301	80,103	0.02s	402	1,404
200	500	45.64s	701	240,503	0.47s	2,002	7,004
500	1,000	536s	1,501	1,251,003	2.37s	4,002	14,004

Conclusion and future works

- First pseudo-polynomial time algorithm to solve total-payoff games, by nested fixed point computation with value iteration
- By means of a reachability variant (MCR games), interesting on their own
- Large subclasses with polynomial time complexity
- Tool: add-on of PRISM games

Conclusion and future works

- First pseudo-polynomial time algorithm to solve total-payoff games, by nested fixed point computation with value iteration
- By means of a reachability variant (MCR games), interesting on their own
- Large subclasses with polynomial time complexity
- Tool: add-on of PRISM games
- **Perspectives:** test the tool over larger benchmarks, enable logic-like specification of the payoff function (like PRISM games)

Conclusion and future works

- First pseudo-polynomial time algorithm to solve total-payoff games, by nested fixed point computation with value iteration
- By means of a reachability variant (MCR games), interesting on their own
- Large subclasses with polynomial time complexity
- Tool: add-on of PRISM games
- **Perspectives:** test the tool over larger benchmarks, enable logic-like specification of the payoff function (like PRISM games)

Thank you for your attention!