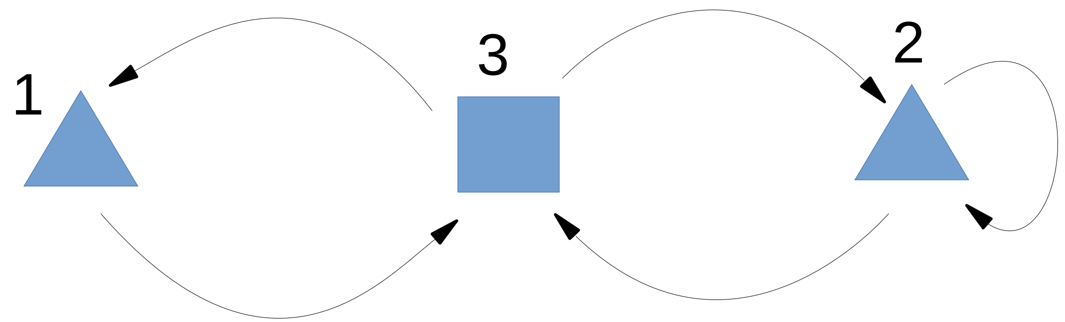
Parity Games: Zielonka's Algorithm in Quasi-Polynomial Time

Paweł Parys
University of Warsaw

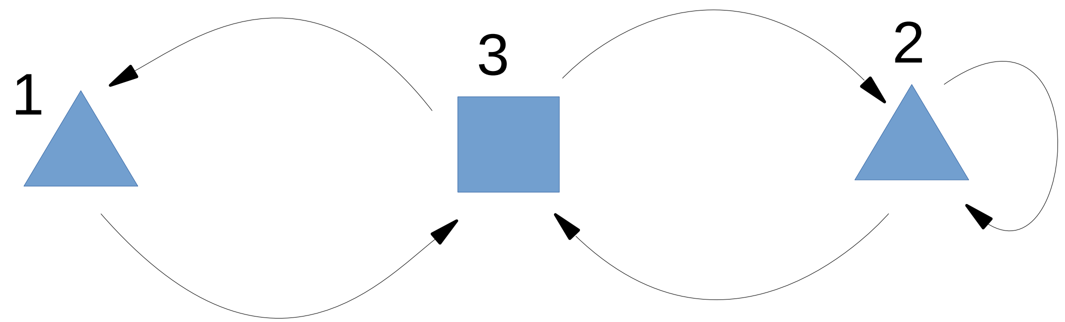
MFCS 2019

Parity games



- Priorities on vertices
- Player owning the current vertex choses the next vertex
- Player □ wins if the biggest priority seen infinitely often is even.

Parity games



- Priorities on vertices
- Player owning the current vertex choses the next vertex
- Player ☐ wins if the biggest priority seen infinitely often is even.

Long standing open problem:
Decide in PTIME which player has a winning strategy.

Recent results

Long standing open problem:
Decide in PTIME which player has a winning strategy.

Recent result:
This can be decided in quasi-polynomial time, i.e. $n^{O(\log n)}$

A few algorithms achieving this:

- Calude, Jain, Khoussainov, Li, Stephan 2017
- Fearnley, Jain, Schewe, Stephan, Wojtczak 2017
- Jurdziński, Lazić 2018
- Lehtinen 2018

Recent results

Long standing open problem:
Decide in PTIME which player has a winning strategy.

Recent result:
This can be decided in quasi-polynomial time, i.e. $n^{O(\log n)}$

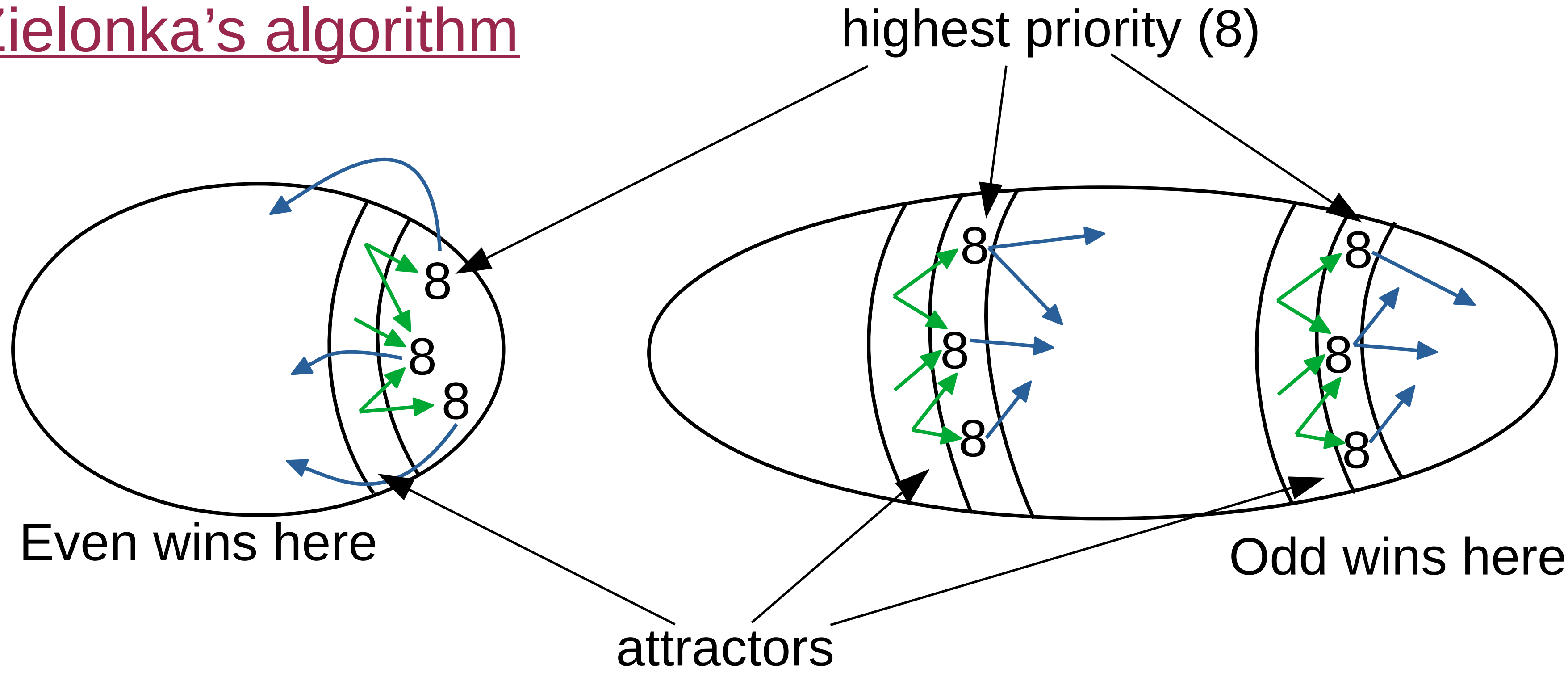
A few algorithms achieving this:
- Calude, Jain, Khoussainov, Li, Stephan 2017
- Fearnley, Jain, Schewe, Stephan, Wojtczak 2017
- Jurdziński, Lazić 2018
- Lehtinen 2018

Older results:
- multiple (sub-)exponential algorithms
- among them: **Zielonka's algorithm** 1998
  - ➔ very simple recursive algorithm
  - ➔ exponential in the worst case
  - ➔ behaves quite well in practice

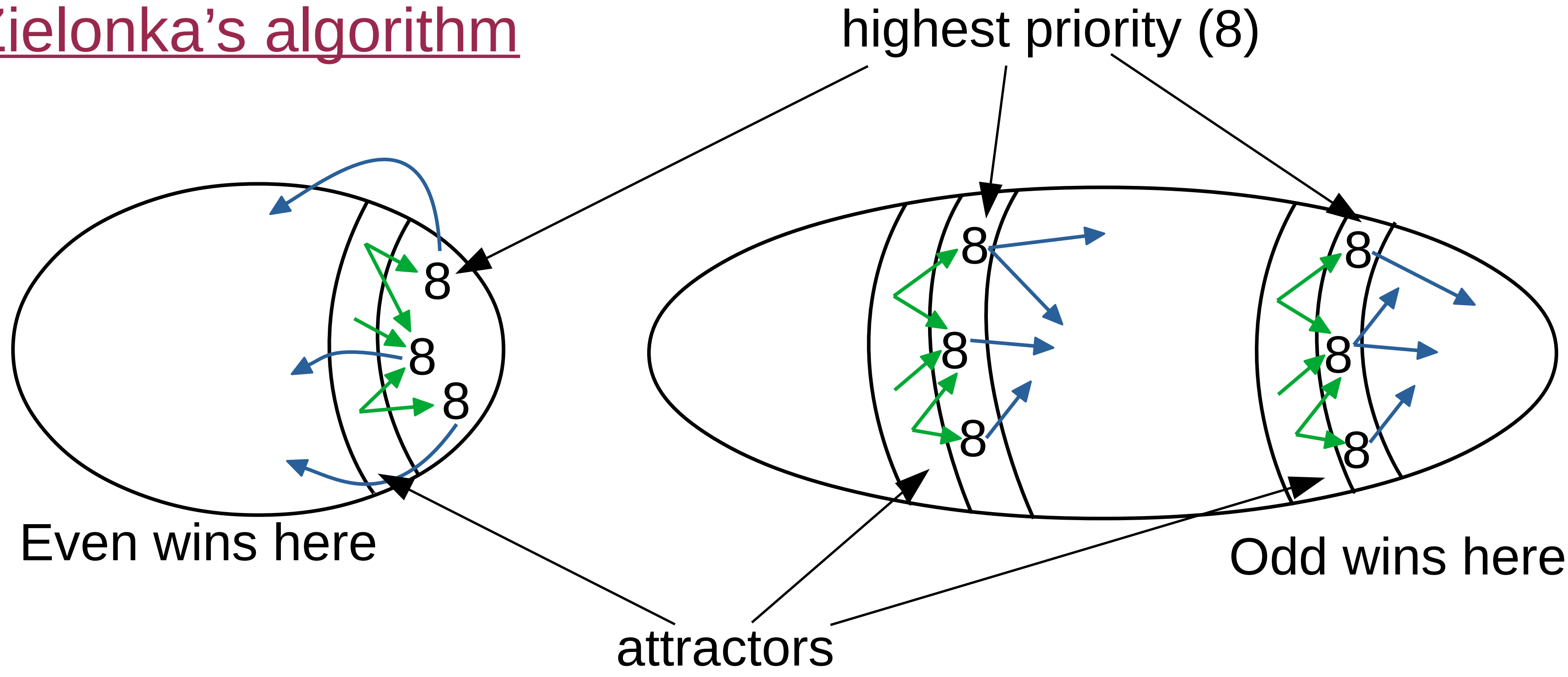
Our contribution

We present a small modification
of the simple, recursive Zielonka's algorithm,
so that it works in quasi-polynomial time, i.e. $n^{O(log(n))}$

Zielonka's algorithm



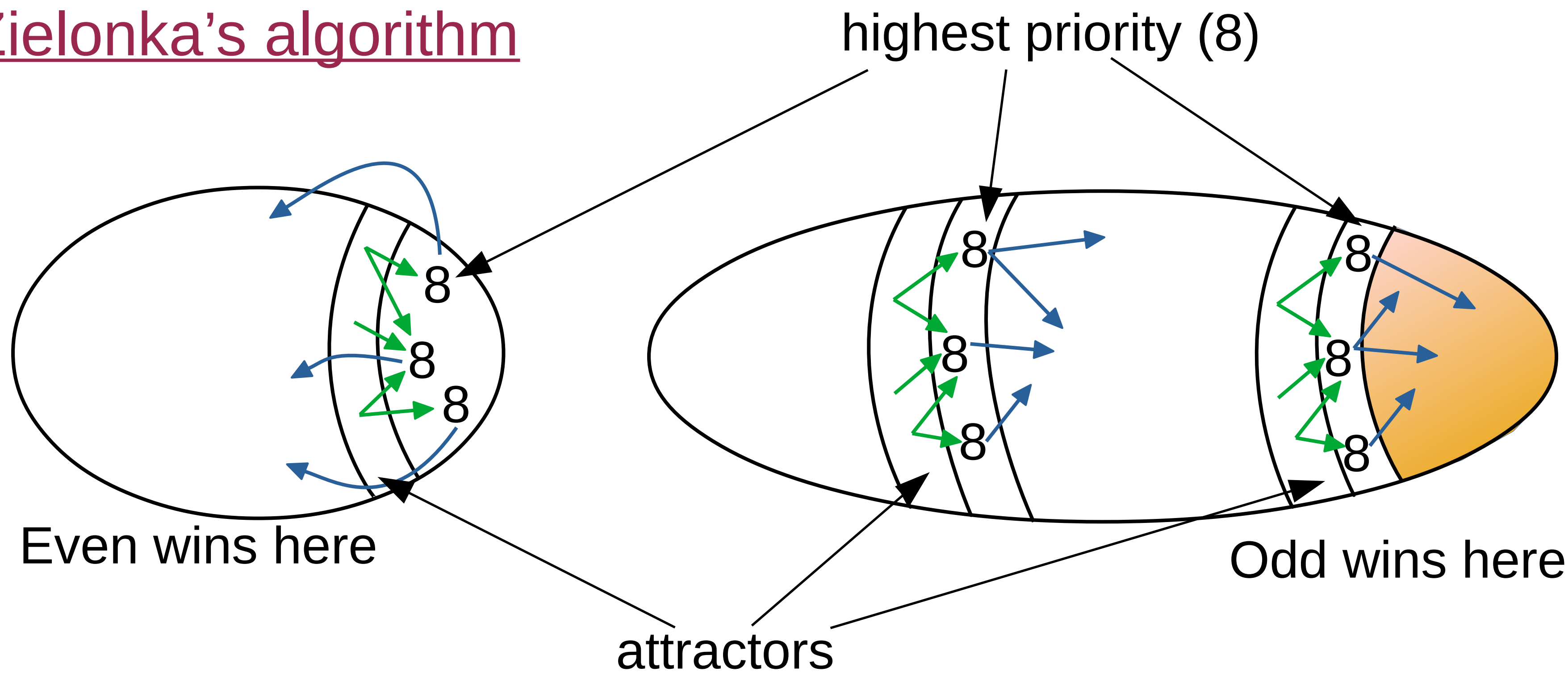
Zielonka’s algorithm



Idea of the recursion in the Zielonka's algorithm:

- Assume that reaching 8 is winning for Even (i.e., remove all 8, and their attractors), and solve the game recursively.

Zielonka's algorithm

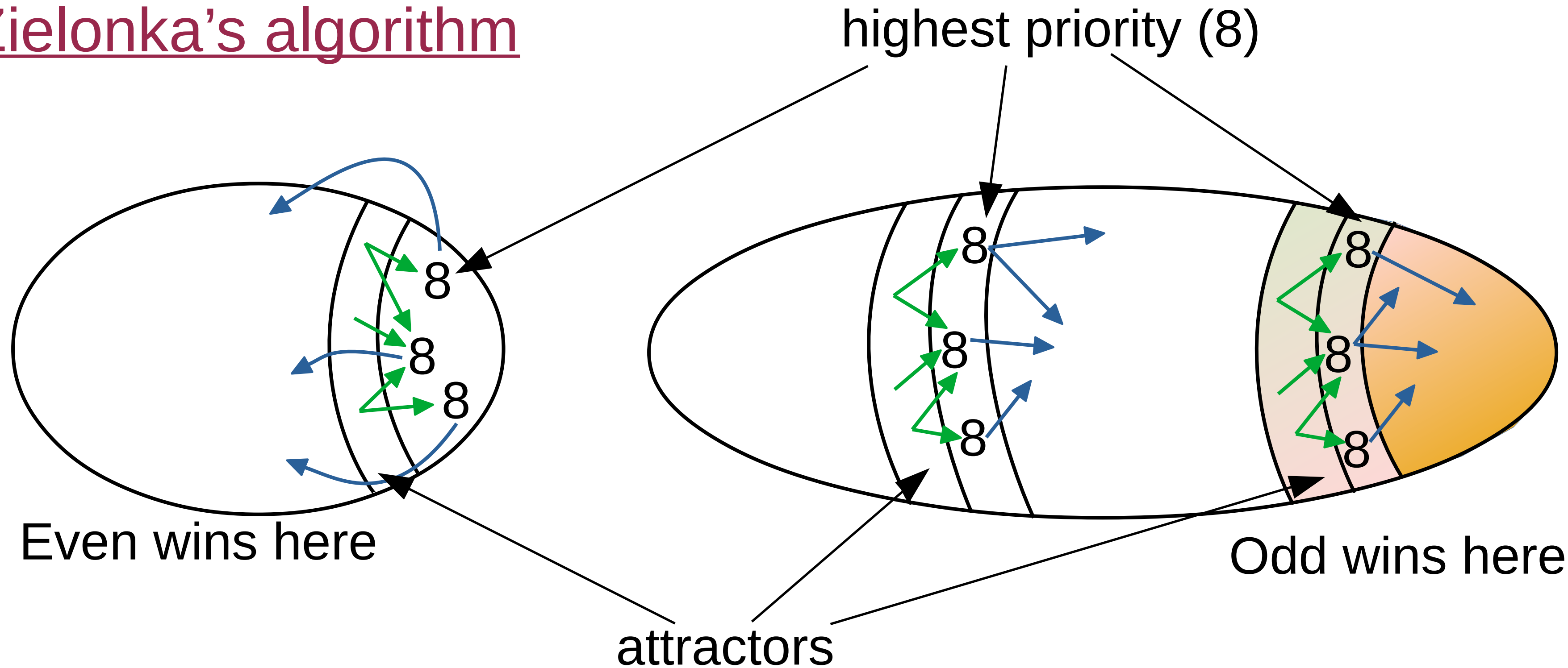


Idea of the recursion in the Zielonka's algorithm:

- Assume that reaching 8 is winning for Even (i.e., remove all 8, and their attractors), and solve the game recursively.

Odd wins the modified game ⇔ Odd wins the original game without seeing any 8

Zielonka's algorithm



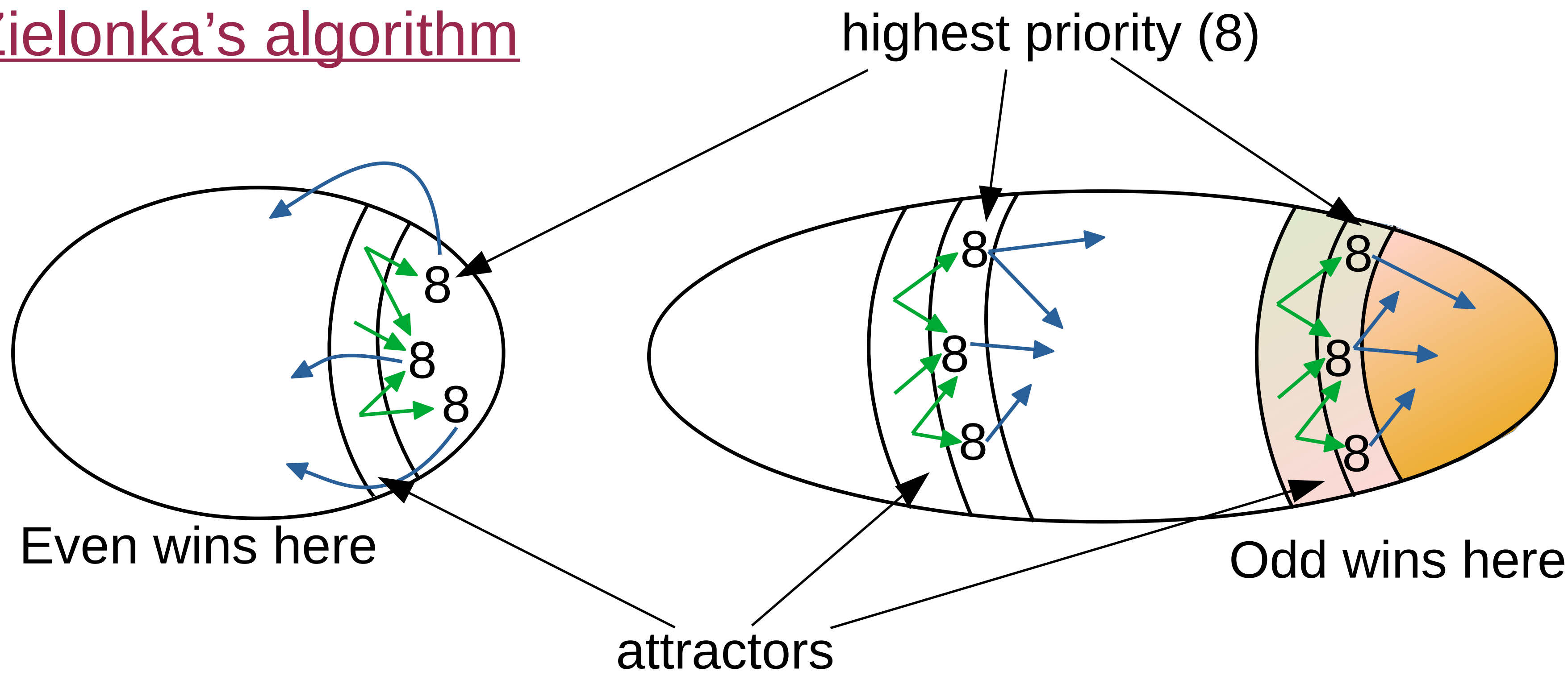
Idea of the recursion in the Zielonka's algorithm:

- Assume that reaching 8 is winning for Even (i.e., remove all 8, and their attractors), and solve the game recursively.

  Odd wins the modified game ⇔ Odd wins the original game without seeing any 8

- Remove the winning region of Odd, together with attractor; solve the remaining game recursively

Zielonka's algorithm

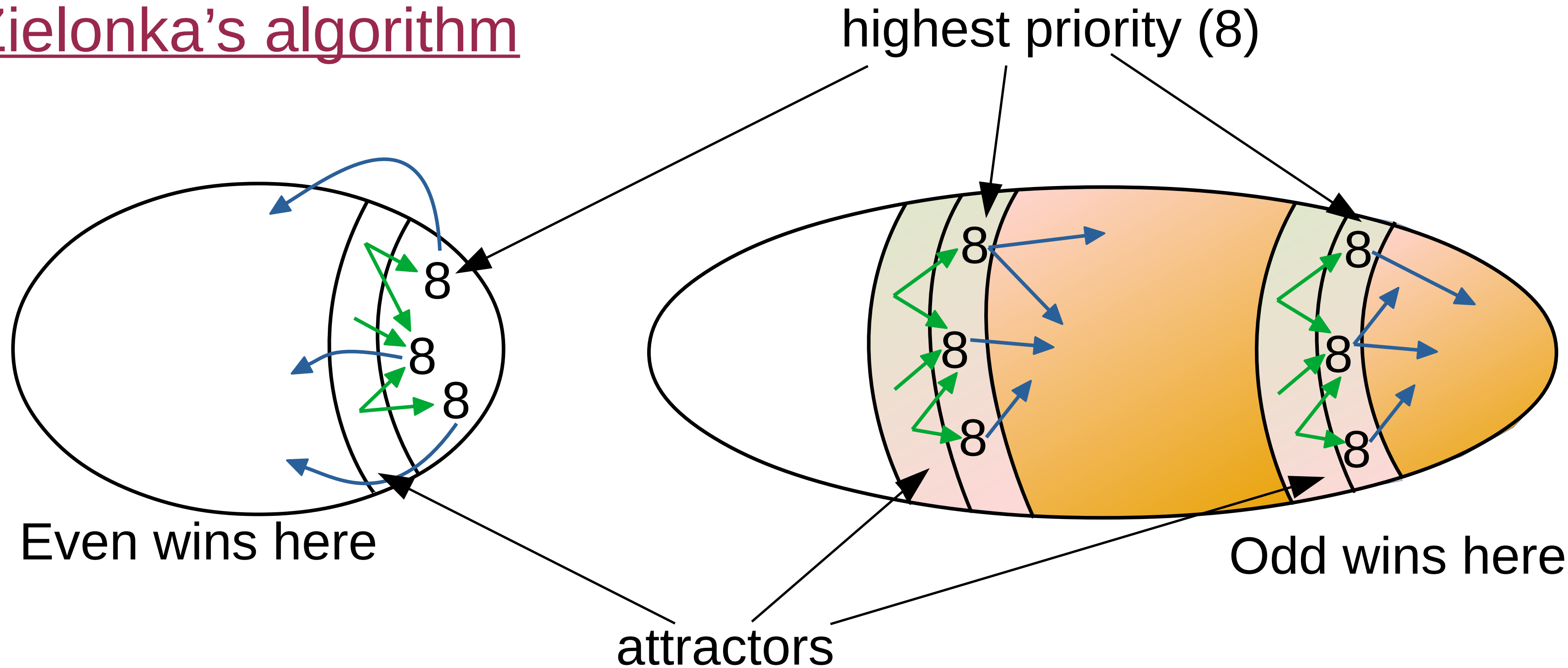


In other words:

- Assume that reaching 8 is winning for Even (i.e., remove all 8, and their attractors), and solve the game recursively; remove the winning region of Odd, together with attractor
- Assume that reaching 8 is winning for Even (i.e., remove all 8, and their attractors), and solve the game recursively; remove the winning region of Odd, together with attractor
- …

(repeat as long as anything changes)

Zielonka's algorithm

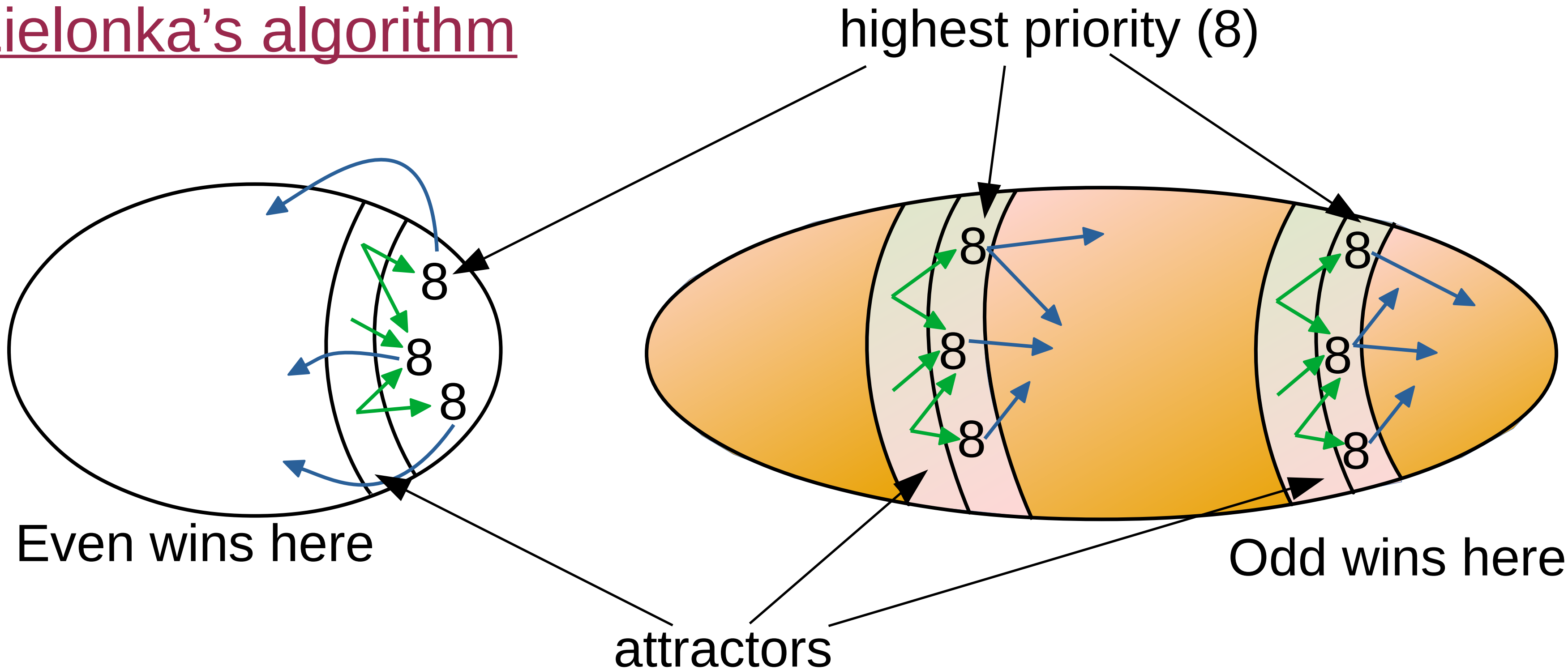


In other words:

- Assume that reaching 8 is winning for Even (i.e., remove all 8, and their attractors), and solve the game recursively; remove the winning region of Odd, together with attractor
- Assume that reaching 8 is winning for Even (i.e., remove all 8, and their attractors), and solve the game recursively; remove the winning region of Odd, together with attractor
- …

(repeat as long as anything changes)

Zielonka's algorithm

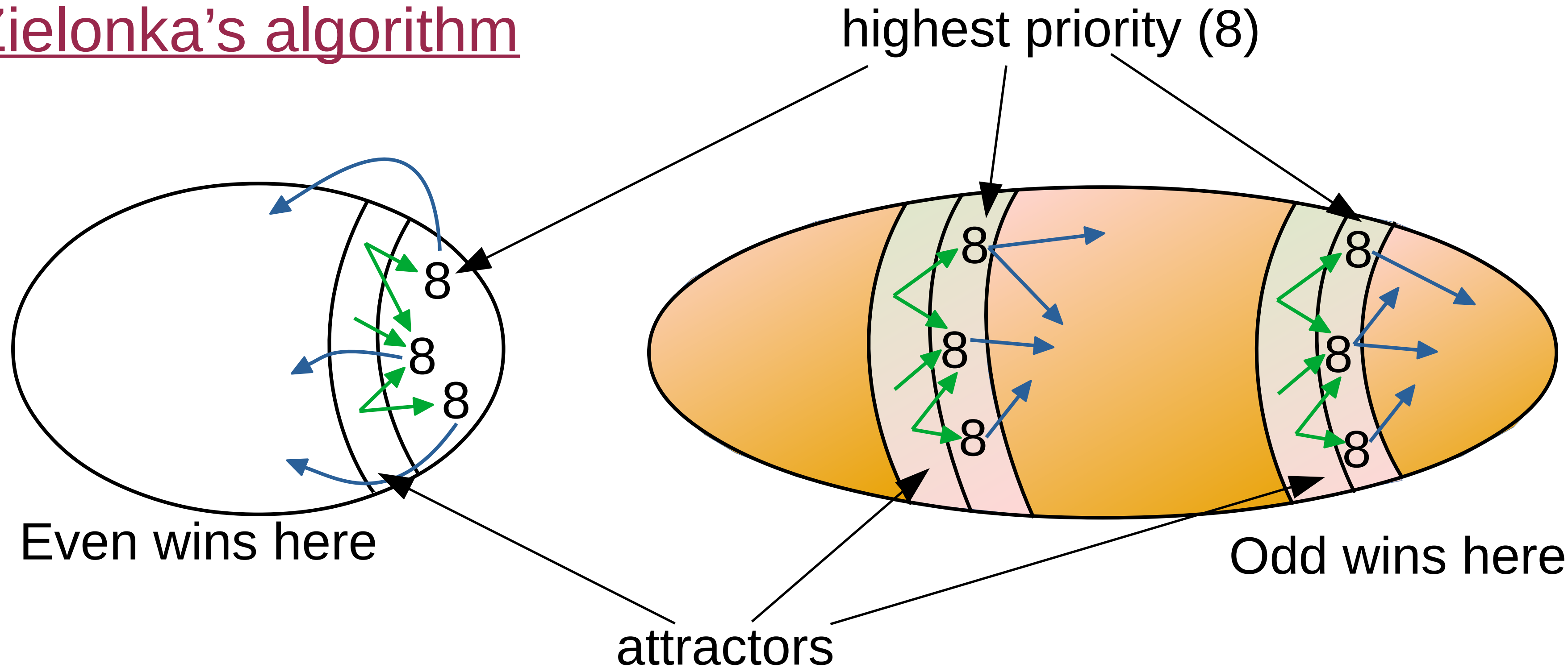


In other words:

- Assume that reaching 8 is winning for Even (i.e., remove all 8, and their attractors), and solve the game recursively; remove the winning region of Odd, together with attractor
- Assume that reaching 8 is winning for Even (i.e., remove all 8, and their attractors), and solve the game recursively; remove the winning region of Odd, together with attractor
- …

(repeat as long as anything changes)

Zielonka's algorithm



Formally:

procedure $\text{Solve}_{\text{E}}(G)$ // highest priority in G is even

do

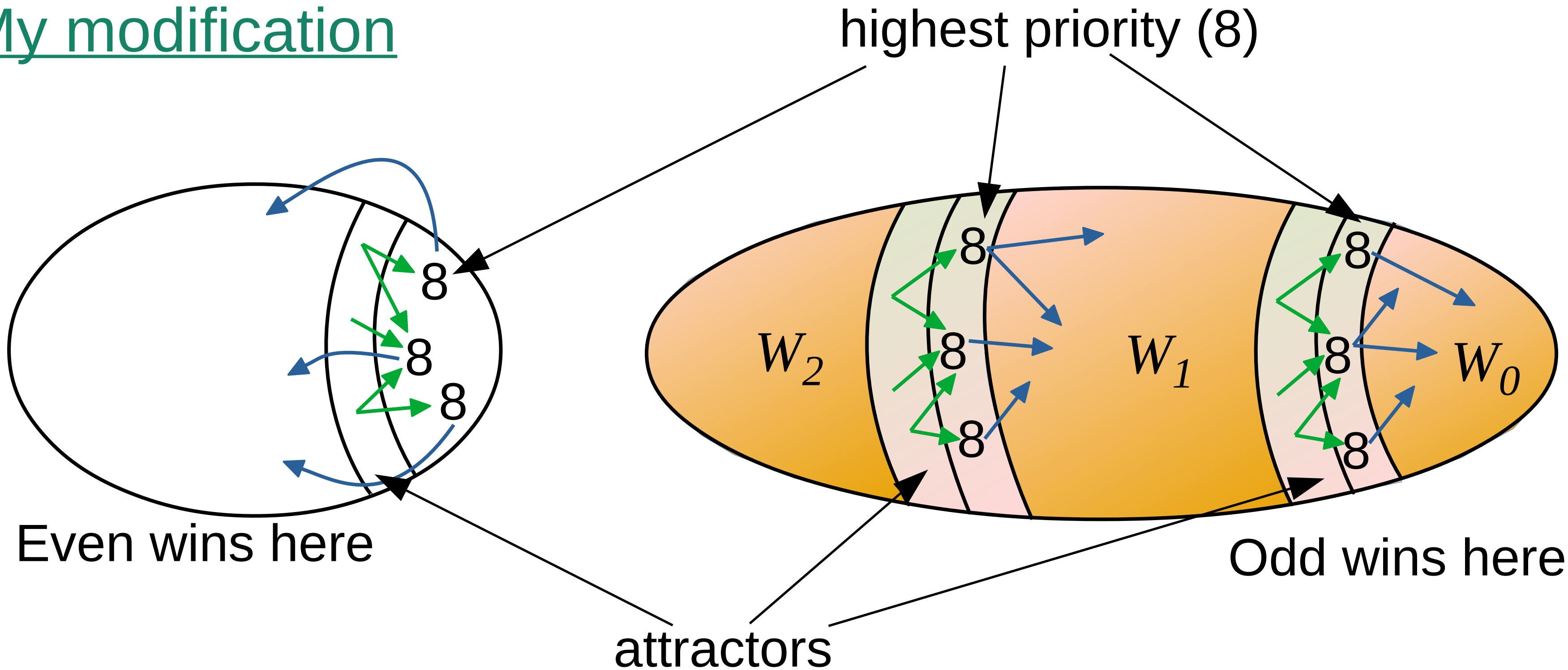
$H = G \setminus \text{Attr}_{\text{E}}(\text{nodes of highest priority})$

$W_{\text{O}} = \text{Solve}_{\text{O}}(H)$

$G = G \setminus \text{Attr}_{\text{O}}(W_{\text{O}})$

while $W_{\text{O}} \neq \emptyset$

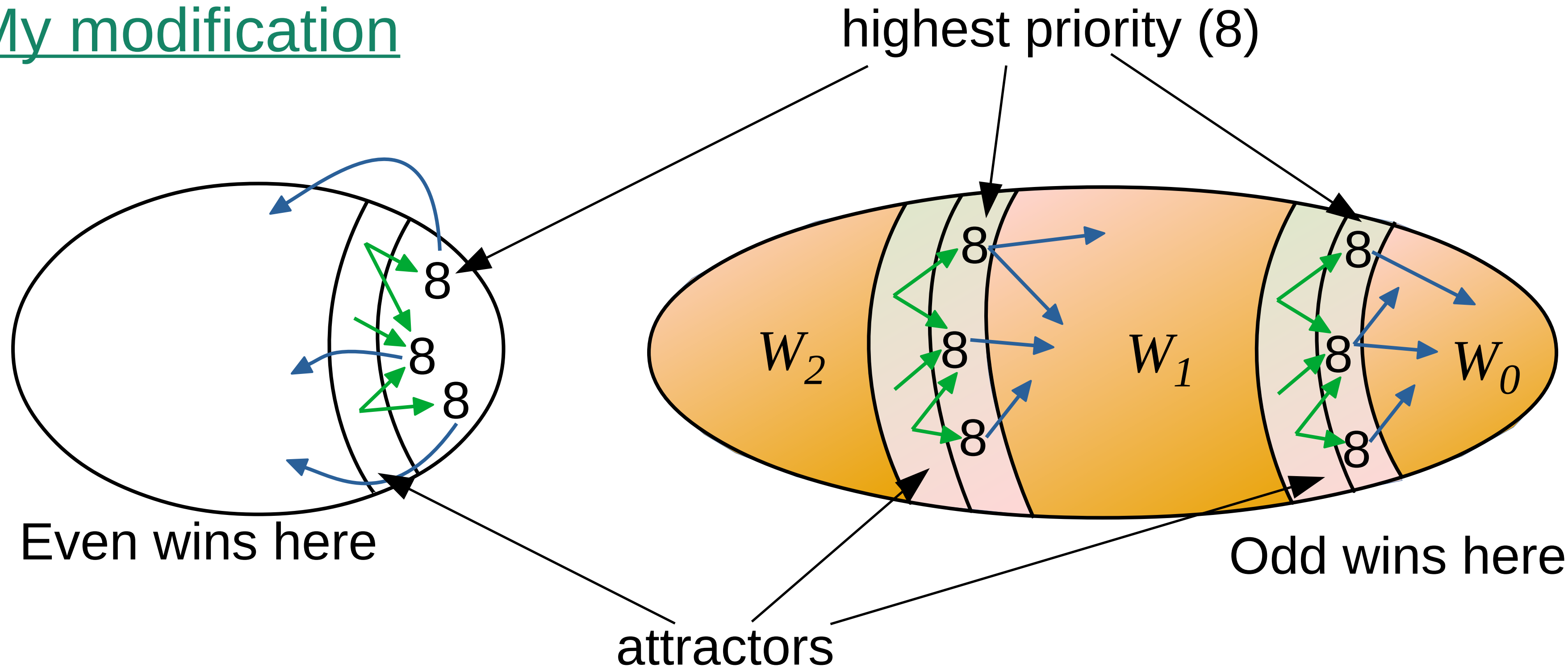
My modification



Observation:

- At most one of the regions W_0,W_1,W_2 has more than $n/2$ nodes (they are disjoint)

My modification



Observation:

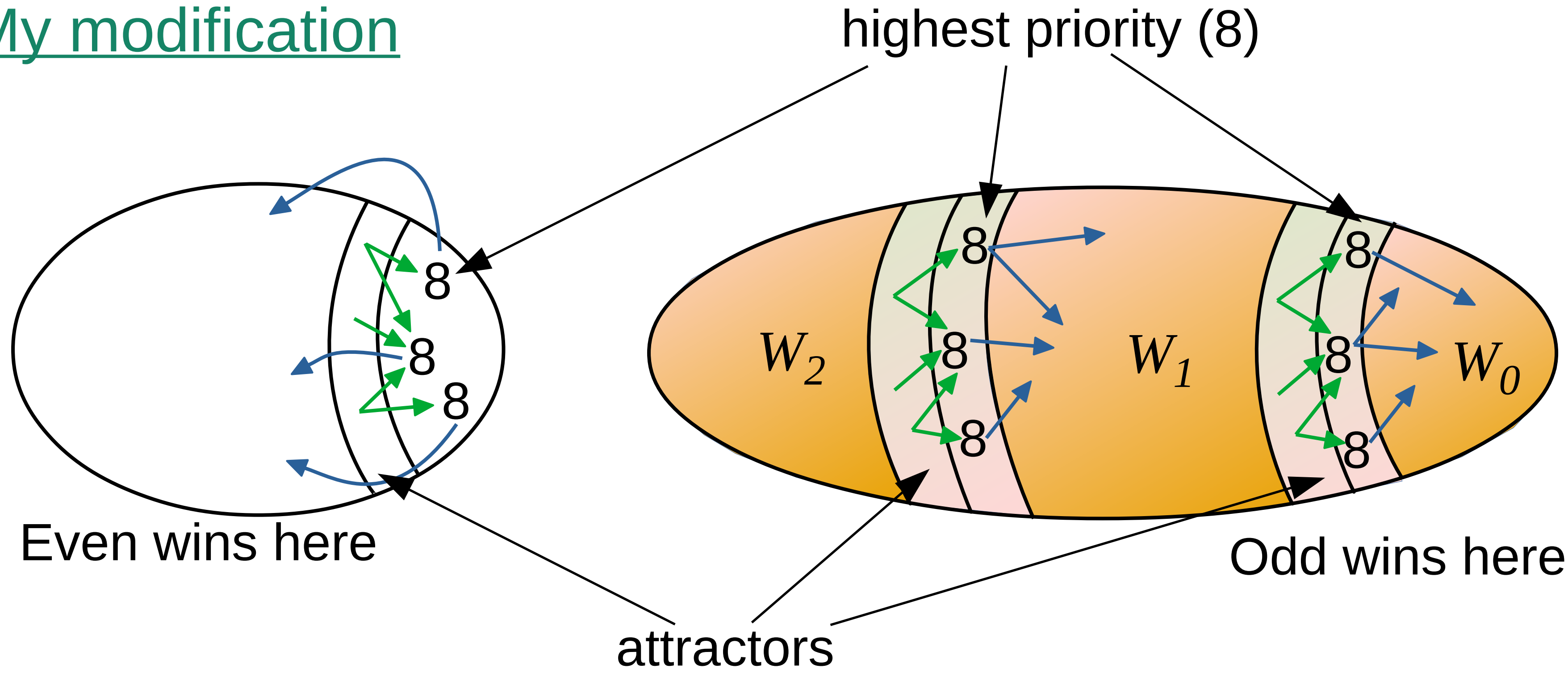
- At most one of the regions W_0, W_1, W_2 has more than $n/2$ nodes (they are disjoint)

Idea:

- Procedure that finds only small winning regions (dominions)

Def. *Dominion* = set of nodes W, such that the player wins from every node of W without leaving W

<u>My modification</u>

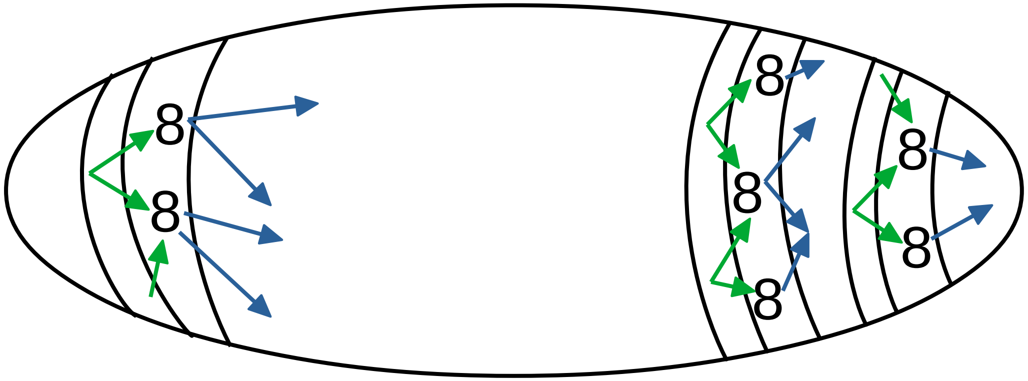


<u>Idea:</u> procedure that finds only small dominions

procedure $solve(G, n_E, n_O)$ returns a set W_E such that:

- if a node v belongs to Even's dominion of size $\leq n_E$ then $v \in W_E$
- if a node v belongs to Odd's dominion of size $\leq n_O$ then $v \notin W_E$
- other nodes v are classified arbitrarily

My modification



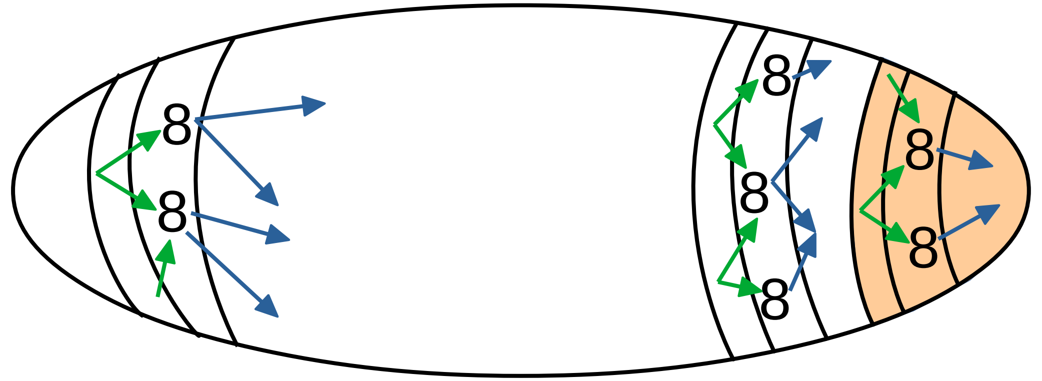
Odd's dominion of size $\leq n_O$

procedure $\text{Solve}_E(G, n_E, n_O)$
if $n_E < 1$ then return $\emptyset$
do
    $H = G \setminus \text{Attr}_E(\text{nodes of highest priority})$
    $W_O = \text{Solve}_O(H, n_E, n_O/2)$
    $G = G \setminus \text{Attr}_O(W_O)$
while $W_O \neq \emptyset$
$H = G \setminus \text{Attr}_E(\text{nodes of highest priority})$
$W_O = \text{Solve}_O(H, n_E, n_O)$
$G = G \setminus \text{Attr}_O(W_O)$
do
    $H = G \setminus \text{Attr}_E(\text{nodes of highest priority})$
    $W_O = \text{Solve}_O(H, n_E, n_O/2)$
    $G = G \setminus \text{Attr}_O(W_O)$
while $W_O \neq \emptyset$

only smaller dominions

size unchanged (once)

My modification



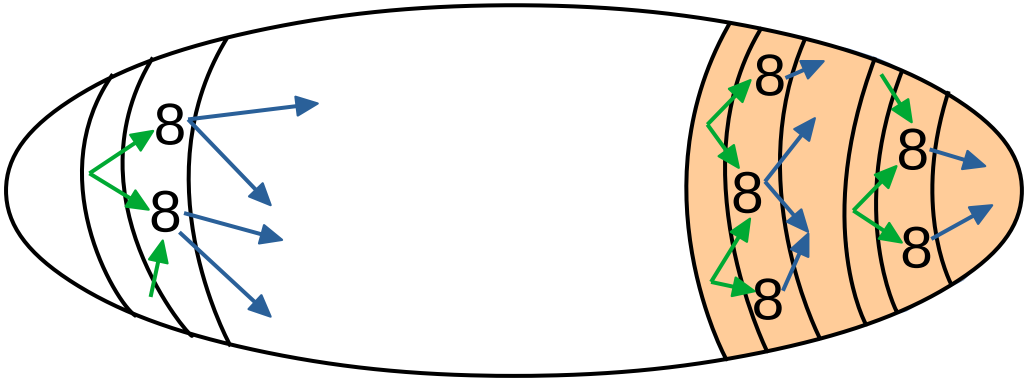
Odd's dominion of size $\leq n_O$

procedure Solve$_E$(G, n_E, n_O)
if $n_E < 1$ then return $\emptyset$
do
    $H = G \setminus \text{Attr}_E$(nodes of highest priority)
    $W_O = \text{Solve}_O(H, n_E, n_O/2)$
    $G = G \setminus \text{Attr}_O(W_O)$
while $W_O \neq \emptyset$
$H = G \setminus \text{Attr}_E$(nodes of highest priority)
$W_O = \text{Solve}_O(H, n_E, n_O)$
$G = G \setminus \text{Attr}_O(W_O)$
do
    $H = G \setminus \text{Attr}_E$(nodes of highest priority)
    $W_O = \text{Solve}_O(H, n_E, n_O/2)$
    $G = G \setminus \text{Attr}_O(W_O)$
while $W_O \neq \emptyset$

only smaller dominions

size unchanged (once)

My modification



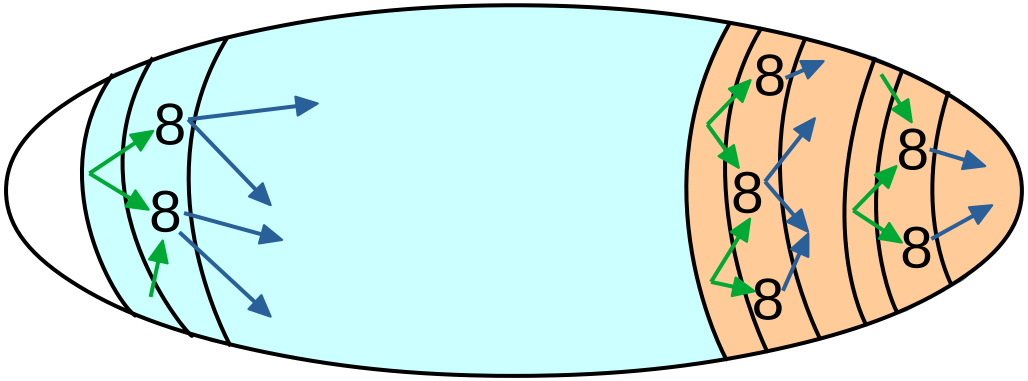
Odd's dominion of size $\leq n_O$

procedure $\text{Solve}_\text{E}(G, n_E, n_O)$
if $n_E < 1$ then return $\emptyset$
do
  $H = G \setminus \text{Attr}_\text{E}(\text{nodes of highest priority})$
  $W_\text{O} = \text{Solve}_\text{O}(H, n_E, n_O/2)$
  $G = G \setminus \text{Attr}_\text{O}(W_\text{O})$
while $W_\text{O} \neq \emptyset$
$H = G \setminus \text{Attr}_\text{E}(\text{nodes of highest priority})$
$W_\text{O} = \text{Solve}_\text{O}(H, n_E, n_O)$
$G = G \setminus \text{Attr}_\text{O}(W_\text{O})$
do
  $H = G \setminus \text{Attr}_\text{E}(\text{nodes of highest priority})$
  $W_\text{O} = \text{Solve}_\text{O}(H, n_E, n_O/2)$
  $G = G \setminus \text{Attr}_\text{O}(W_\text{O})$
while $W_\text{O} \neq \emptyset$

only smaller dominions

size unchanged (once)

My modification



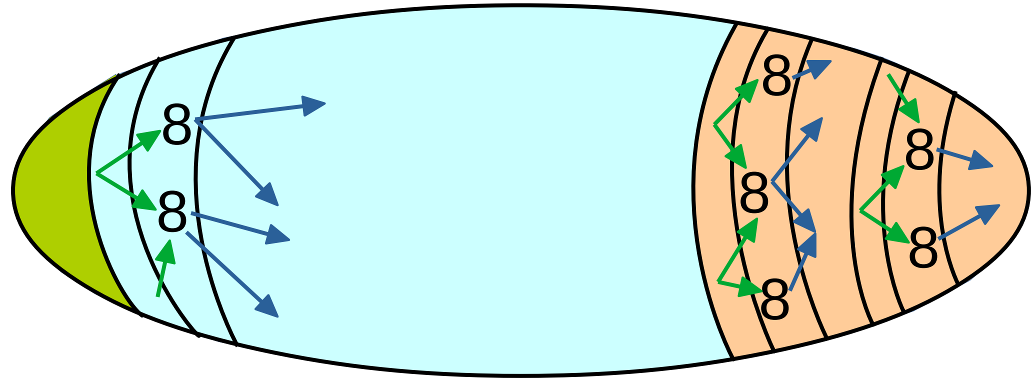
Odd's dominion of size $\leq n_O$

procedure $\text{Solve}_E(G, n_E, n_O)$
if $n_E < 1$ then return $\emptyset$
do
    $H = G \setminus \text{Attr}_E(\text{nodes of highest priority})$
    $W_O = \text{Solve}_O(H, n_E, n_O/2)$
    $G = G \setminus \text{Attr}_O(W_O)$
while $W_O \neq \emptyset$
$H = G \setminus \text{Attr}_E(\text{nodes of highest priority})$
$W_O = \text{Solve}_O(H, n_E, n_O)$
$G = G \setminus \text{Attr}_O(W_O)$
do
    $H = G \setminus \text{Attr}_E(\text{nodes of highest priority})$
    $W_O = \text{Solve}_O(H, n_E, n_O/2)$
    $G = G \setminus \text{Attr}_O(W_O)$
while $W_O \neq \emptyset$

only smaller dominions

size unchanged (once)

My modification



Odd's dominion of size $\leq n_O$

procedure Solve$_E$(G, n_E, n_O)
if $n_E < 1$ then return $\emptyset$
do
    $H = G \setminus \mathrm{Attr}_E$(nodes of highest priority)
    $W_O = \mathrm{Solve}_O(H, n_E, n_O/2)$
    $G = G \setminus \mathrm{Attr}_O(W_O)$
while $W_O \neq \emptyset$
$H = G \setminus \mathrm{Attr}_E$(nodes of highest priority)
$W_O = \mathrm{Solve}_O(H, n_E, n_O)$
$G = G \setminus \mathrm{Attr}_O(W_O)$
do
    $H = G \setminus \mathrm{Attr}_E$(nodes of highest priority)
    $W_O = \mathrm{Solve}_O(H, n_E, n_O/2)$
    $G = G \setminus \mathrm{Attr}_O(W_O)$
while $W_O \neq \emptyset$

only smaller dominions

size unchanged (once)

Running time

Let:

n = number of nodes

h = maximal priority

$l = log\ n_E + log\ n_O$

Then the running time (number of recursive calls) is:

$$R(h,l) \leq 1+n{\cdot}R(h-1,l-1)+R(h-1,l)$$

This gives us:

$$R(h,l) \leq n^{l}{\cdot}(h+l)^{l} = n^{O(log\ n)}$$

Running time

Let:

n = number of nodes

h = maximal priority

$l = log\ n_E + log\ n_O$

Then the running time (number of recursive calls) is:

$$R(h,l) \leq 1+n \cdot R(h-1,l-1)+R(h-1,l)$$

This gives us:

$$R(h,l) \leq n^{l} \cdot (h+l)^{l} = n^{O(log\ n)}$$

Follow up:

K. Lehtinen, S. Schewe, D. Wojtczak 2019:
the complexity can be improved to $n^{O(log\ h)}$

Running time

Let:

n = number of nodes

h = maximal priority

$l = log\ n_E + log\ n_O$

Then the running time (number of recursive calls) is:

$$R(h,l) \leq 1+n \cdot R(h-1,l-1)+R(h-1,l)$$

This gives us:

$$R(h,l) \leq n^l \cdot (h+l)^l = n^{O(log\ n)}$$

Follow up:

K. Lehtinen, S. Schewe, D. Wojtczak 2019:
the complexity can be improved to $n^{O(log\ h)}$

Implementation?

- Zielonka's algorithm – relatively fast in practice (usually)
- quasi-polynomial-time algorithms – much slower
- (a simple implementation of) my algorithm – also slow (similar to QPT)

Summary

We present a small modification
of the simple, recursive Zielonka's algorithm,
so that it works in quasi-polynomial time, i.e. $n^{O(log(n))}$

Why our algorithm is interesting?

- simplicity
- different approach (all the other quasi-polynomial-time algorithms follow so-called separation approach)

Thank you!