Parity to Safety in Polynomial Time for Pushdown and Collapsible Pushdown Games

Matthew Hague
Royal Holloway, University of London

with Roland Meyer, Sebastian Muskalla, and Martin Zimmermann

Abstract

- Safety games for pushdown systems:
  n-EXPTIME-complete
- Parity games for pushdown systems:
  n-EXPTIME-complete

(for order-n collapsible pushdown)

Abstract

- Safety games for pushdown systems:
  n-EXPTIME-complete
- Parity games for pushdown systems:
  n-EXPTIME-complete

(for order-n collapsible pushdown)

We give a "natural" parity->safety reduction

Abstract

- Safety games for pushdown systems:
  n-EXPTIME-complete
- Parity games for pushdown systems:
  n-EXPTIME-complete

(for order-n collapsible pushdown)

We give a "natural" parity->safety reduction

(i.e. not parity algorithm -> TM -> safety)

Motivation

- Safety is easy to reason about (Parity is hard)
- Parity is more expressive
- To know the relationship

Ideas We Start With

- Finite-state parity to safety using counters (Bernet et al, 2002)

Ideas We Start With

- Finite-state parity to safety using counters (Bernet et al, 2002)
- Pushdown parity to safety with large counters (Fridman and Zimmermann, 2012)

Ideas We Start With

- Finite-state parity to safety using counters (Bernet et al, 2002)
- Pushdown parity to safety with large counters (Fridman and Zimmermann, 2012)
- Reduction order-n -> order-(n-1)
  - Rank awareness

  (H, Murawski, Ong, Serre, 2008)

Ideas We Start With

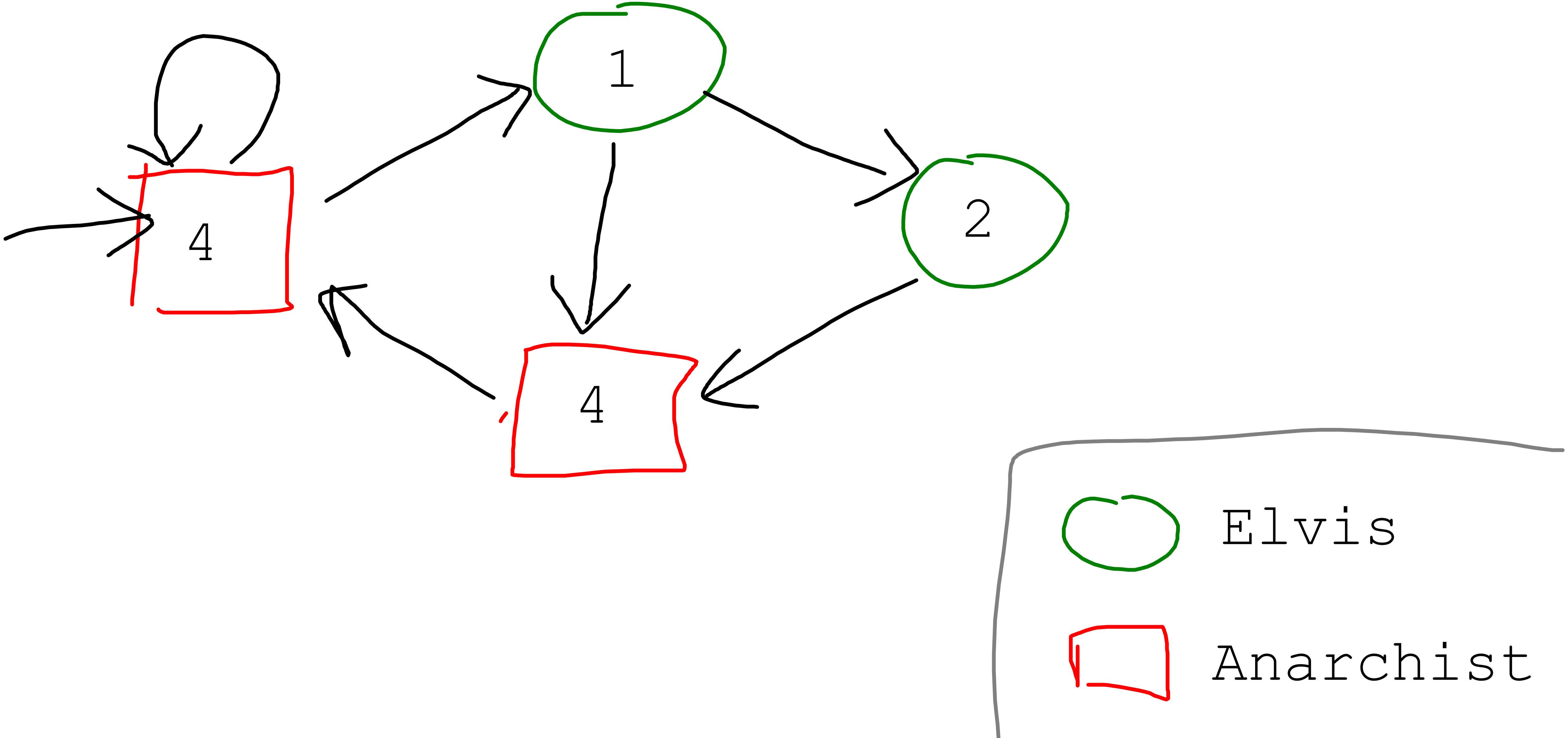
- Finite-state parity to safety using counters (Bernet et al, 2002)
- Pushdown parity to safety with large counters (Fridman and Zimmermann, 2012)
- Reduction order-n -> order-(n-1)
  - Rank awareness

  (H, Murawski, Ong, Serre, 2008)
- Encoding large counters in a pushdown stack (Cachat and Walukiewicz, 2007)

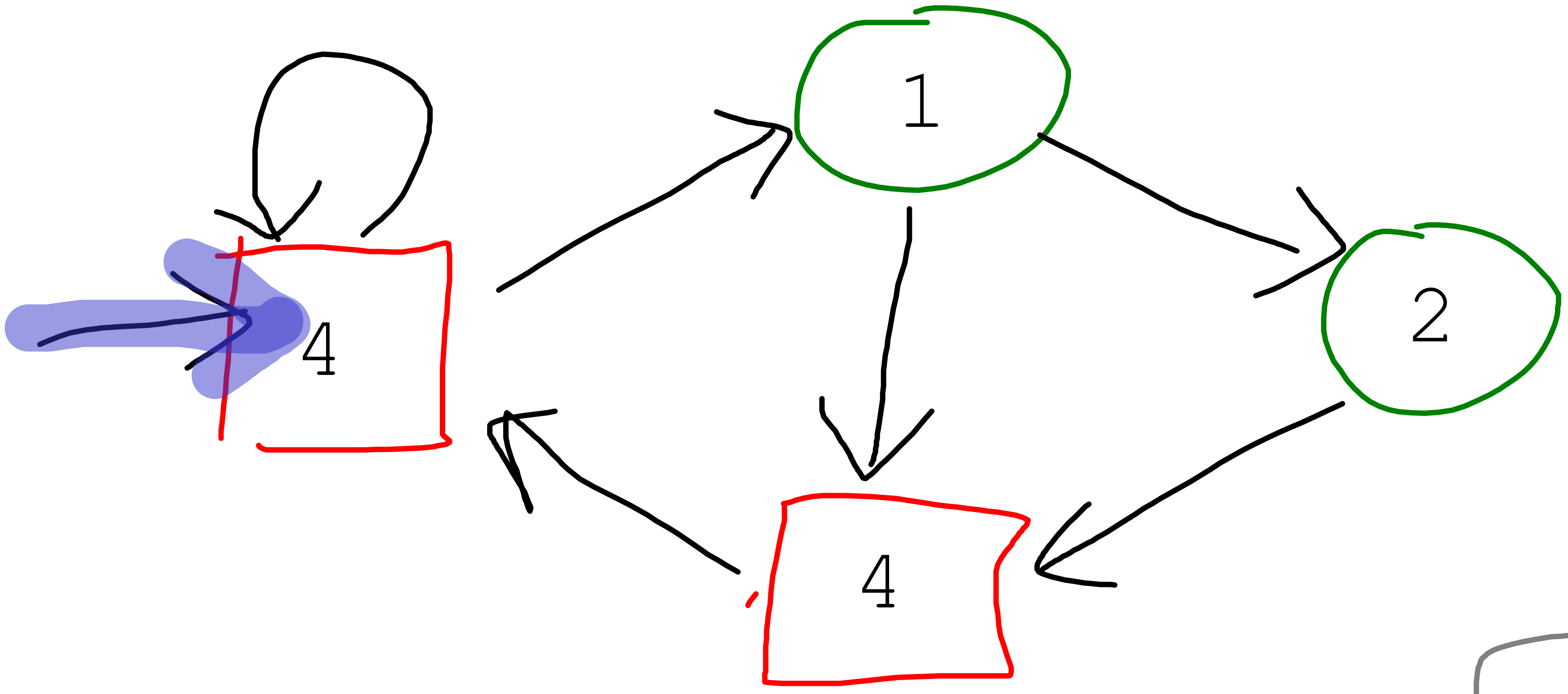
Contributions

- Generalise counter encoding to collapse
- Direct proof based on commutativity of
  - Counters encoding
  - Stack removal
- Counters behave like a stack

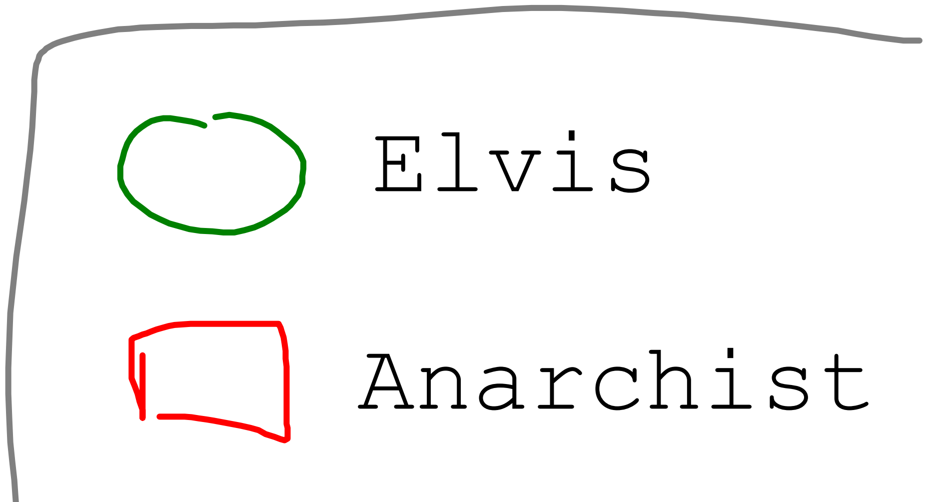
Parity Game



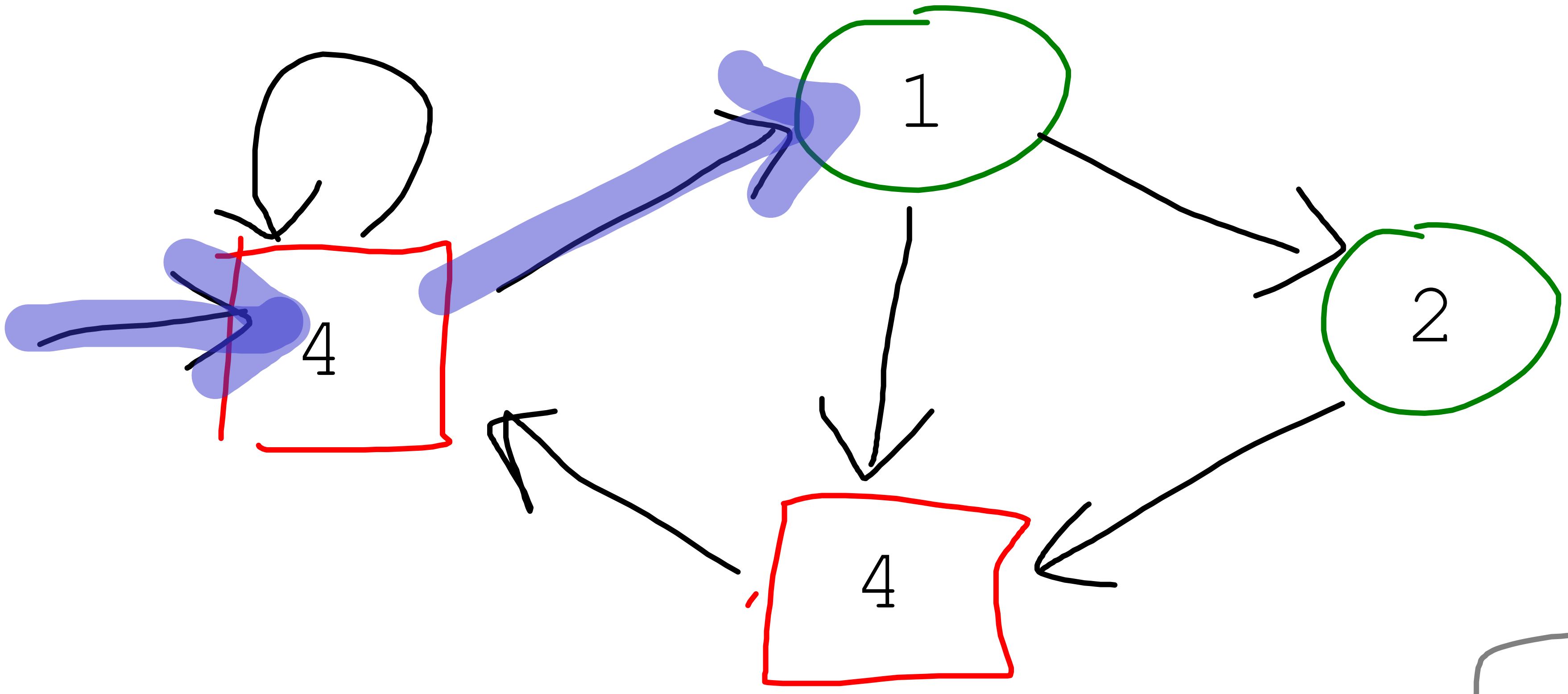
Parity Game



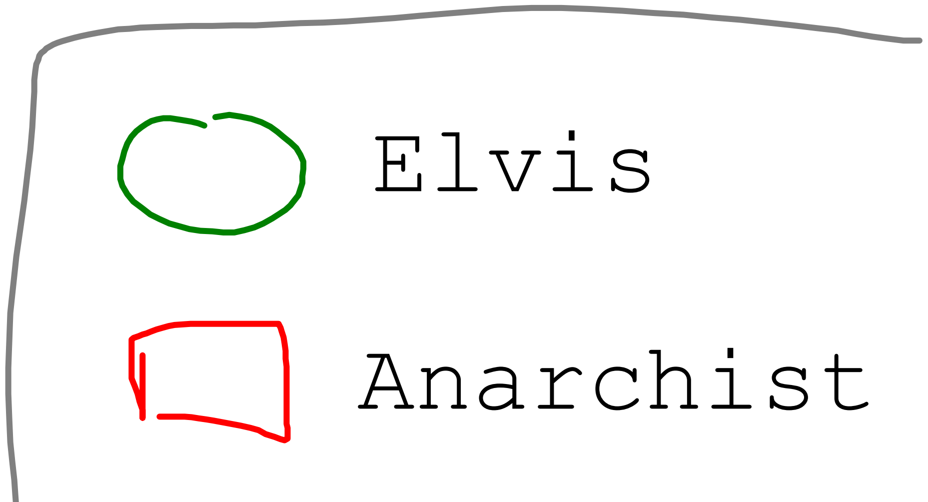
Play: 4



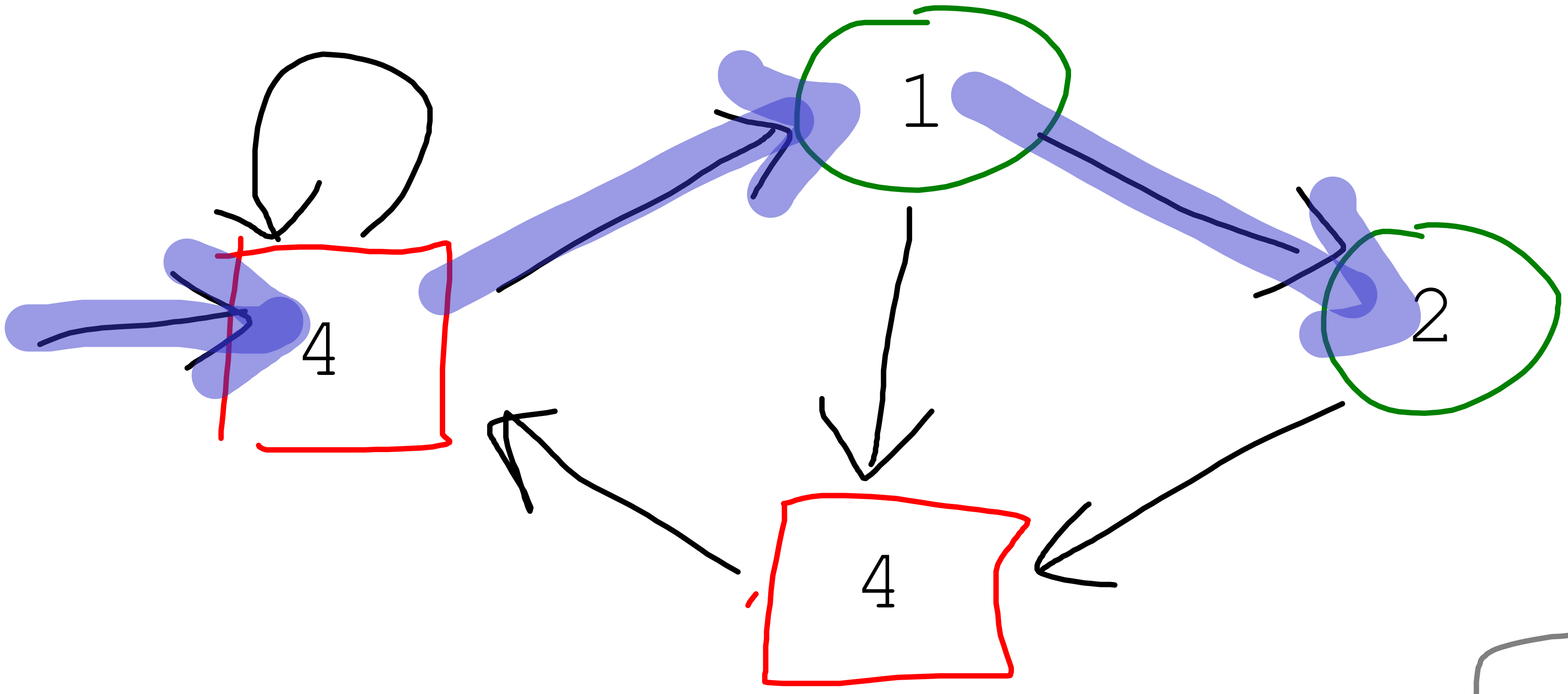
Parity Game



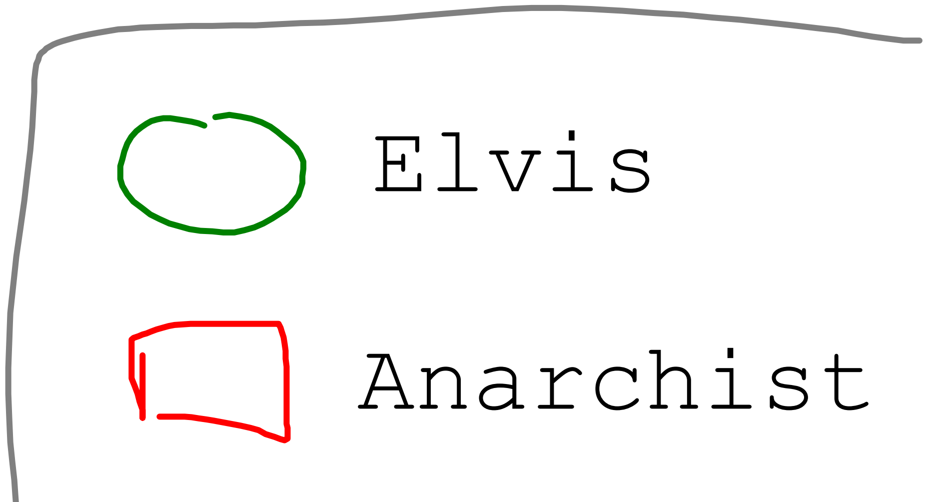
Play: 4 1



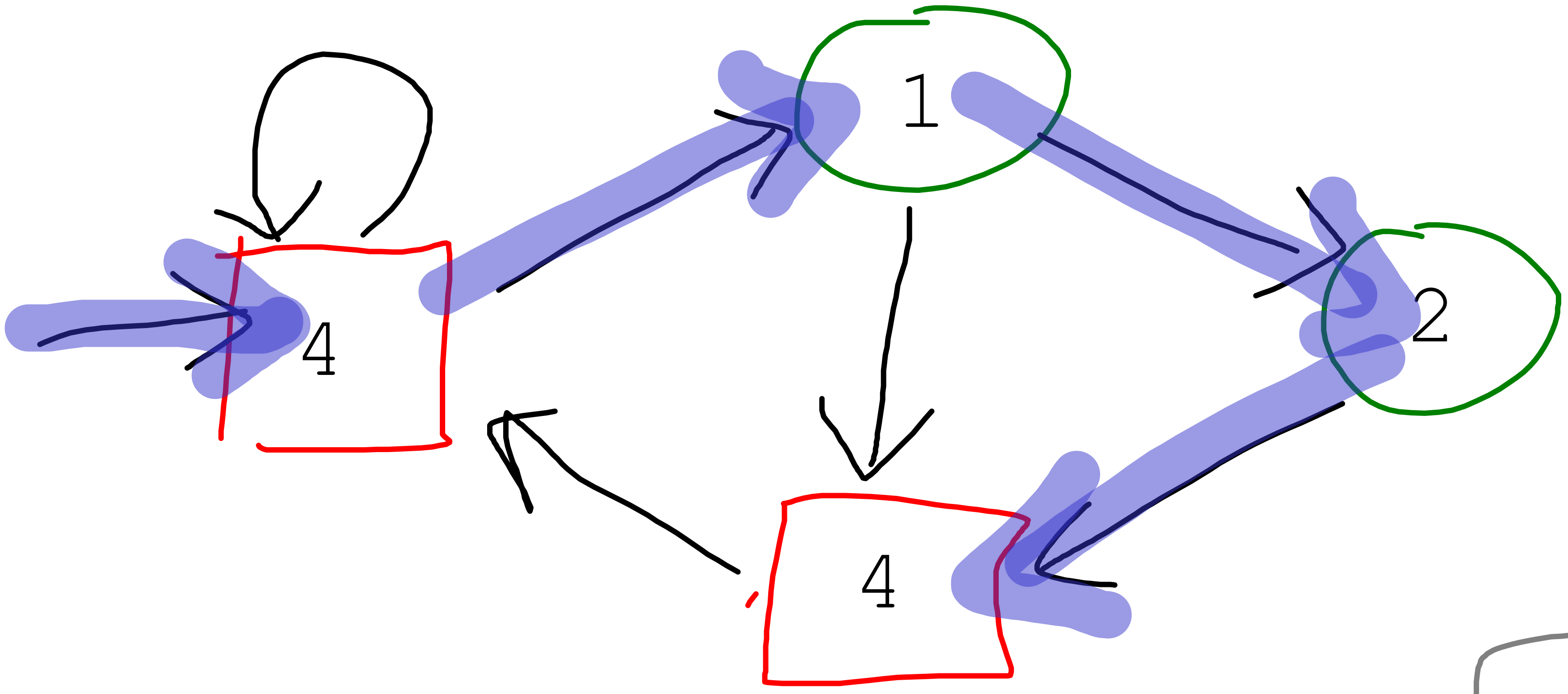
Parity Game



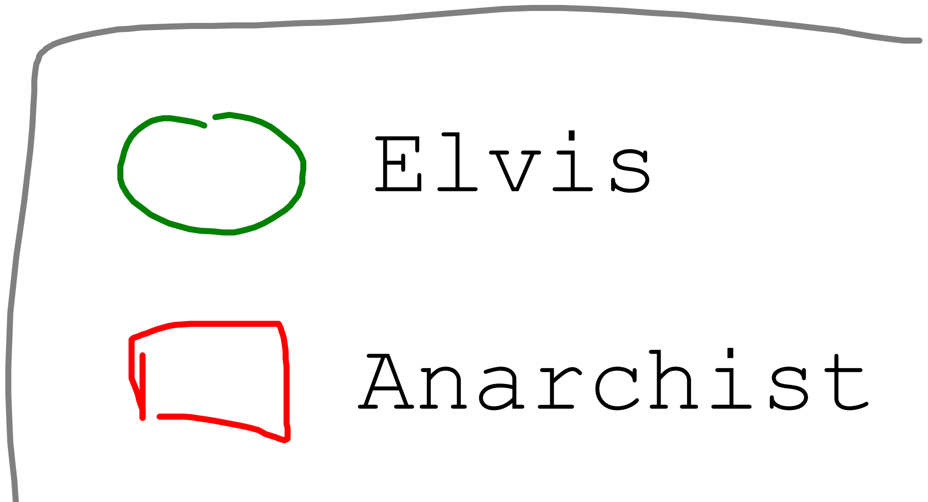
Play: 4 1 2



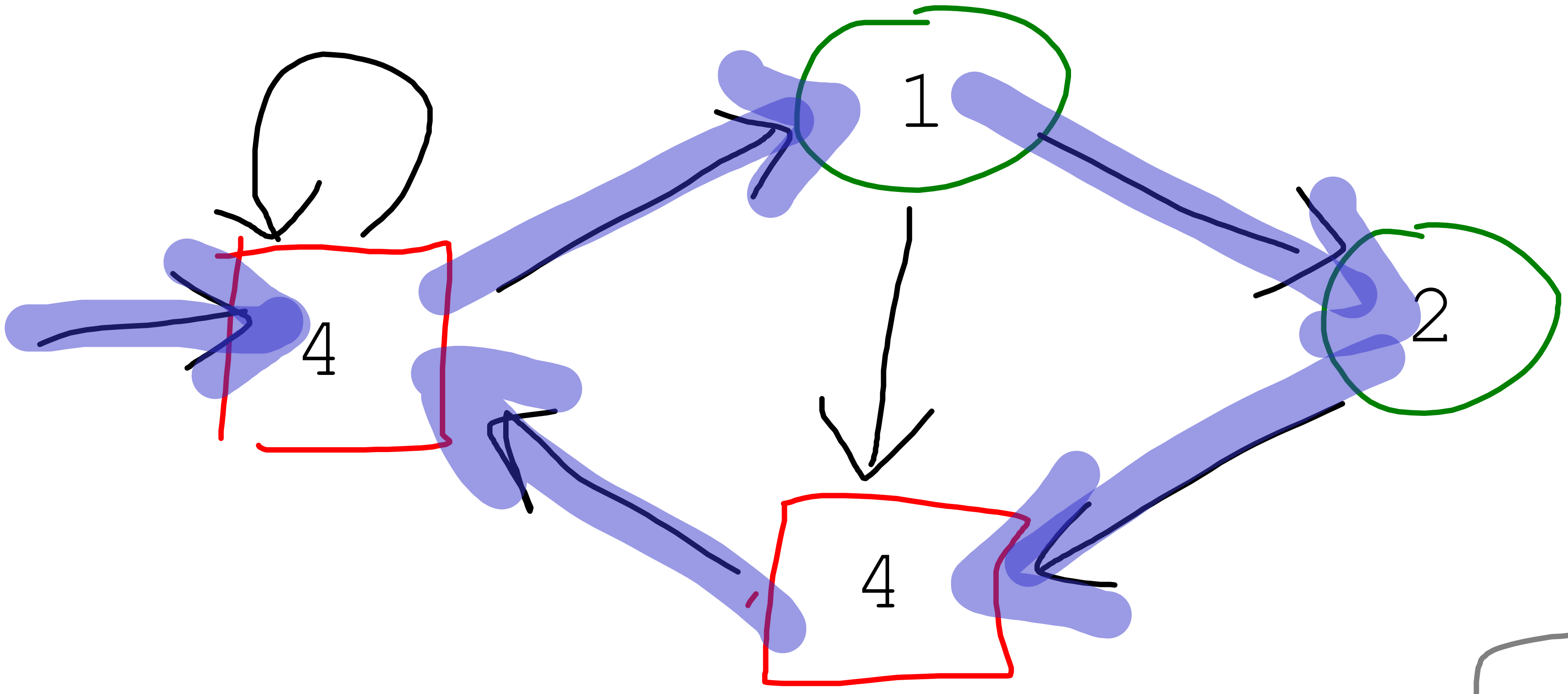
Parity Game



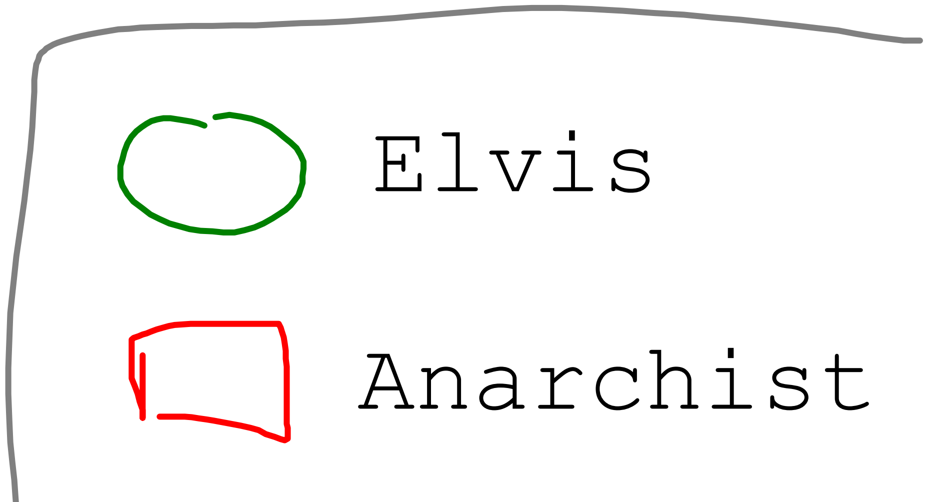
Play: 4 1 2 4



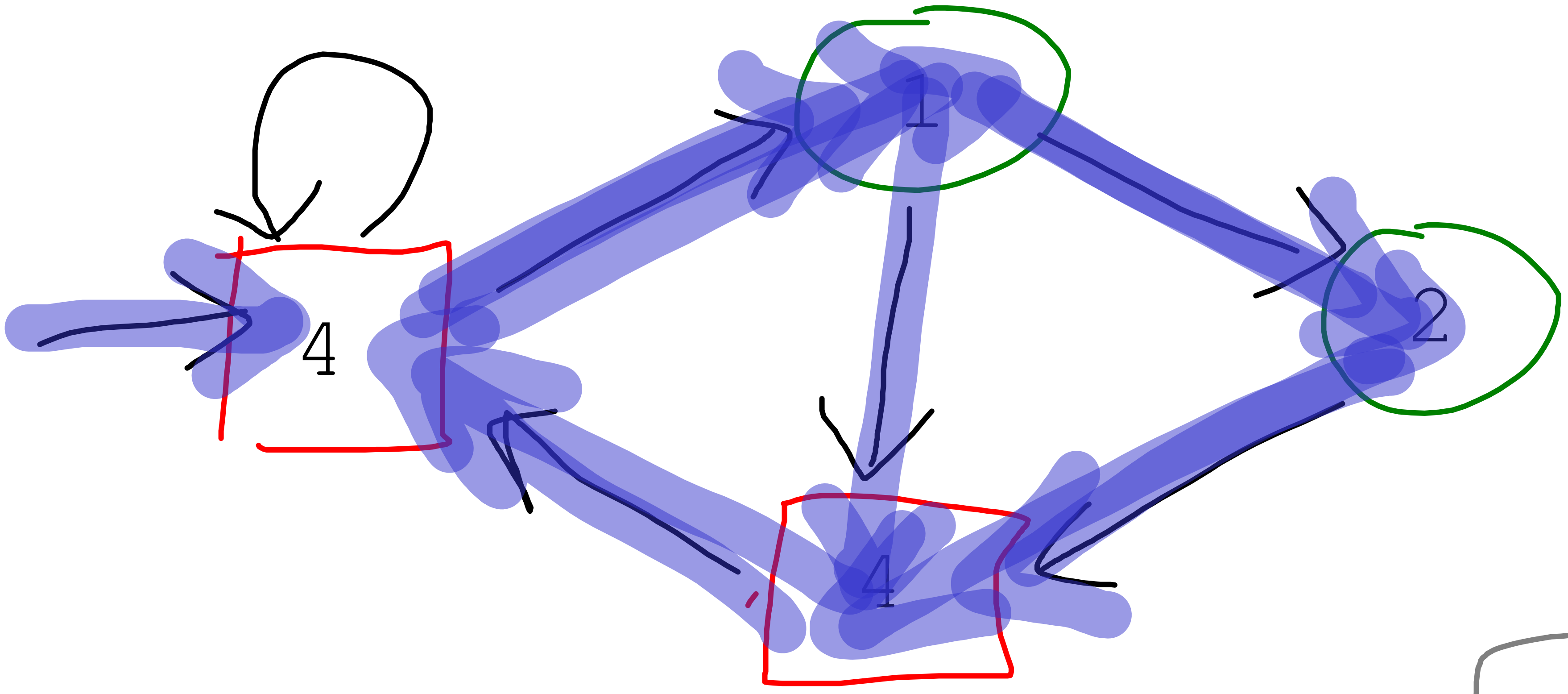
Parity Game



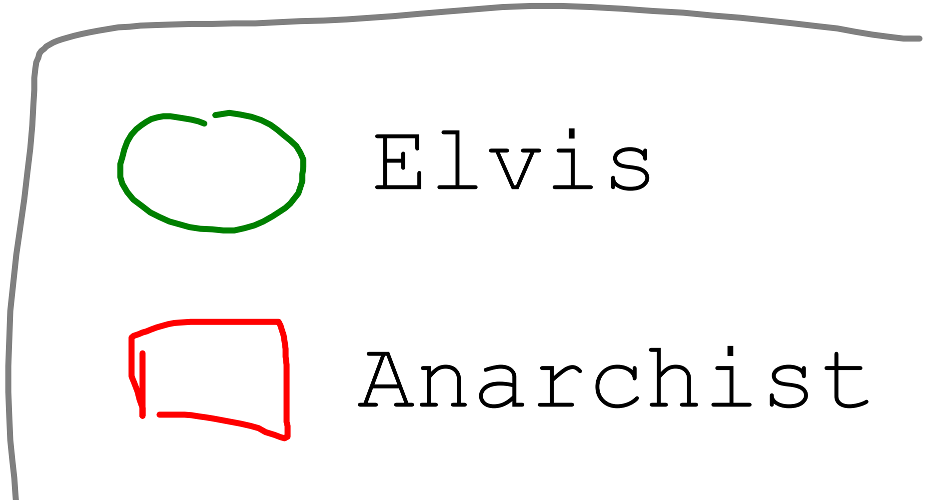
Play: 4 1 2 4 4



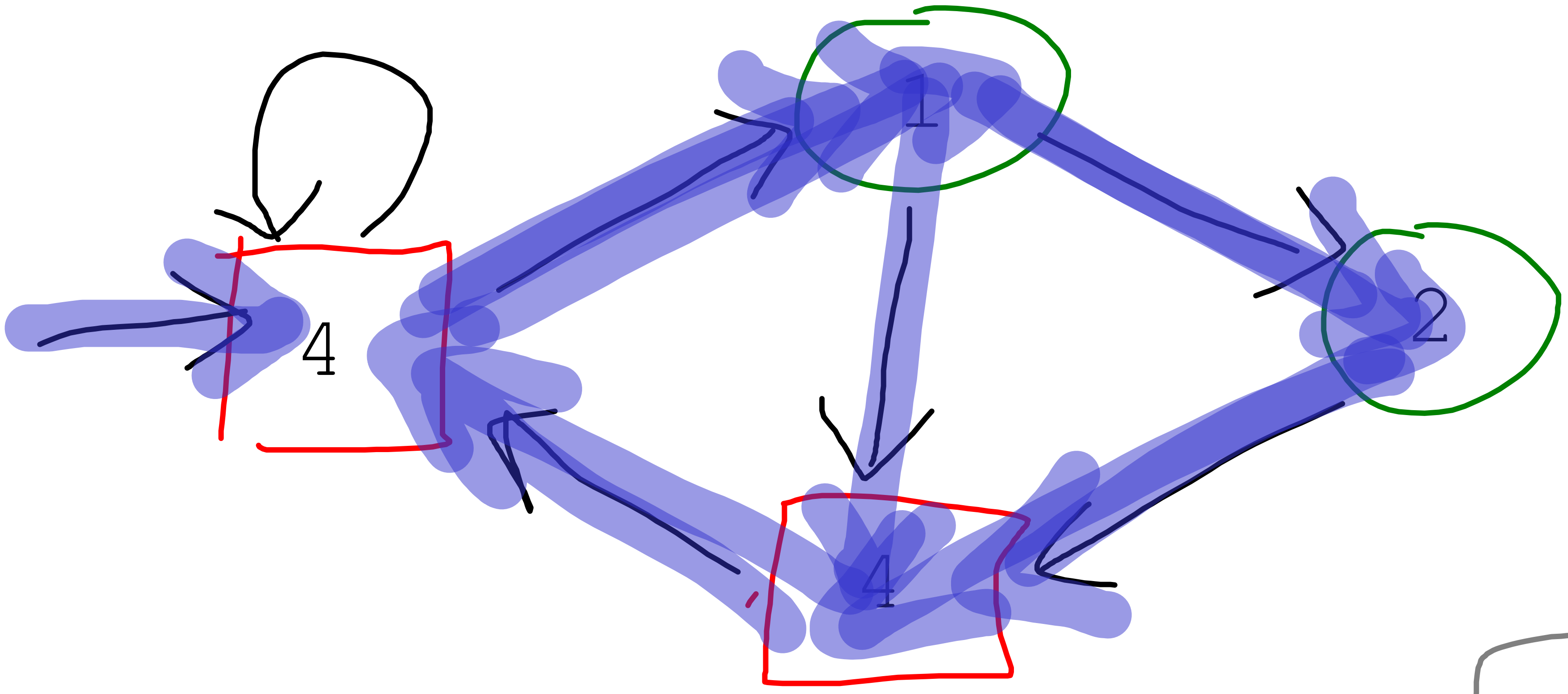
Parity Game



Play: 4 1 2 4 4 1 4 4 1 2 4...

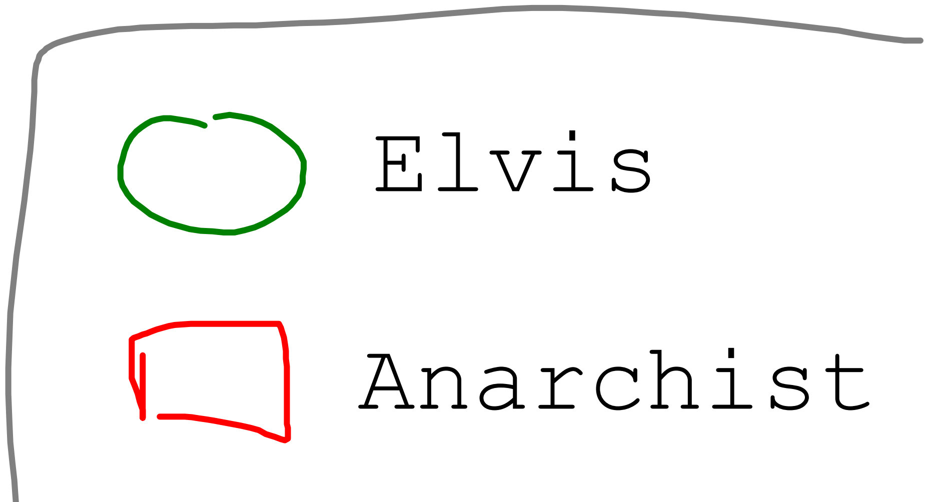


Parity Game

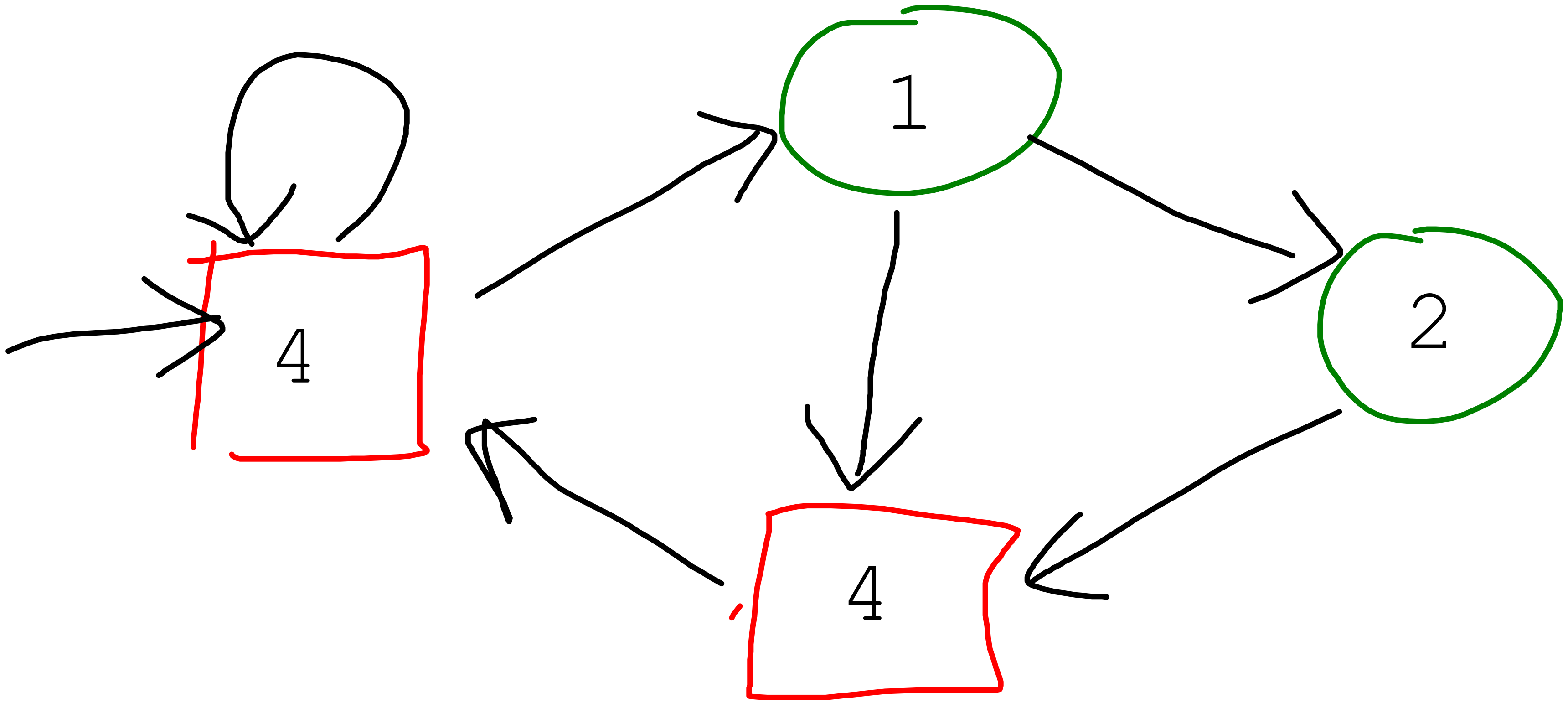


Play: 4 1 2 4 4 1 4 4 1 2 4...

Winner: Elvis if least infinitely occuring rank is even

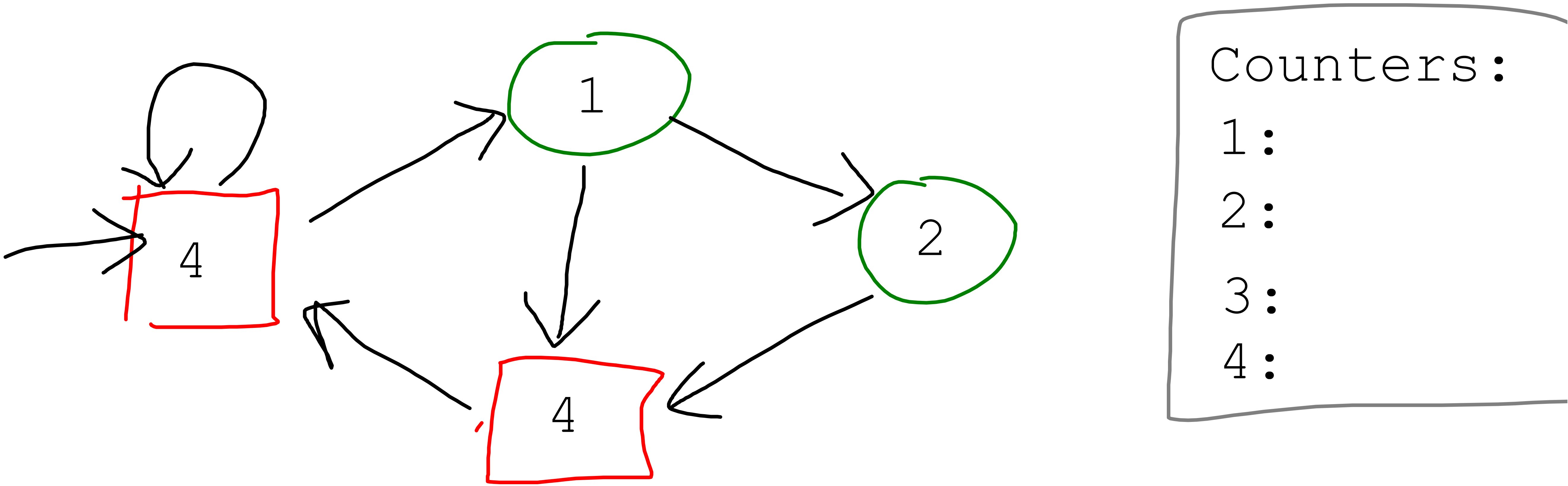


Parity with Counters



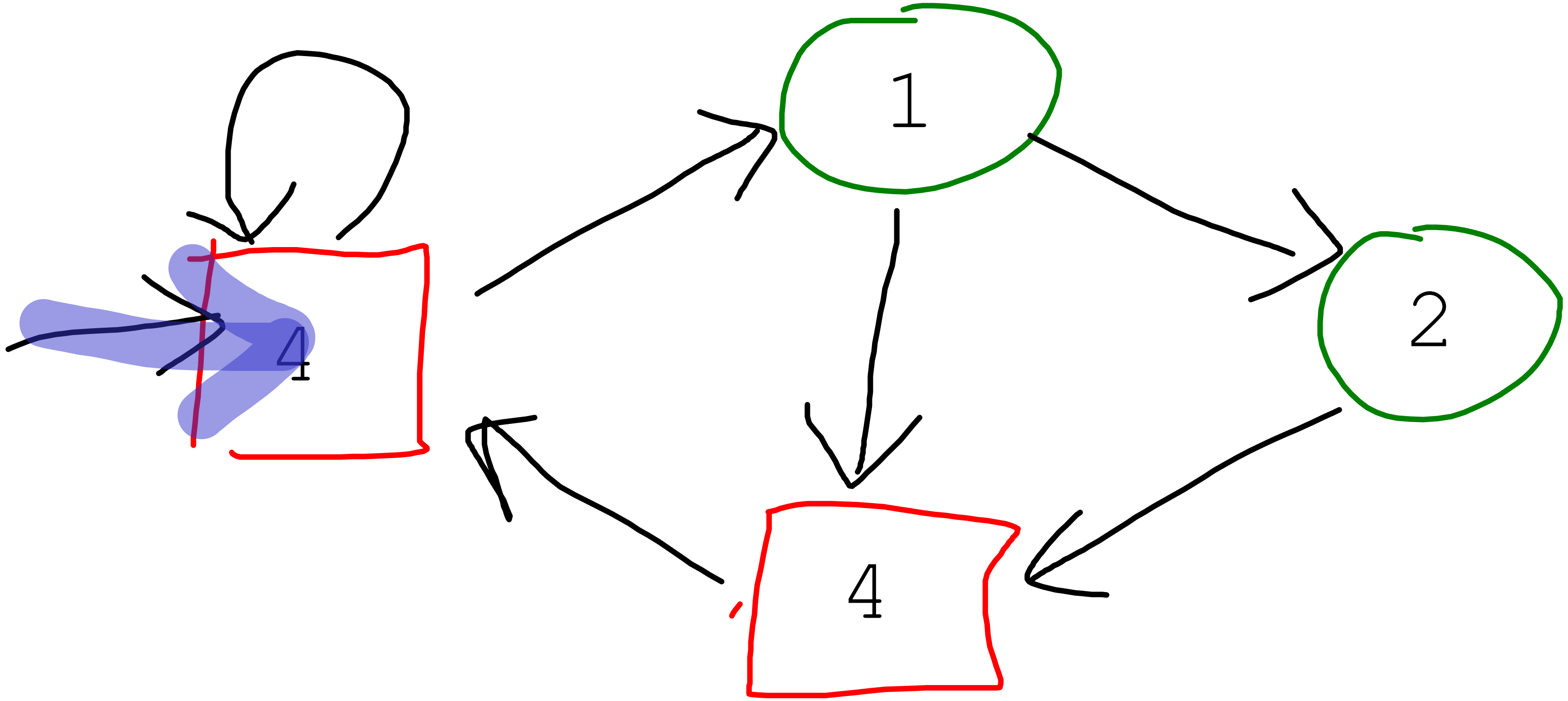
Count: number of each rank
(without seeing anything smaller)

Parity with Counters



Count: number of each rank
(without seeing anything smaller)

Parity with Counters



Counters:

1:

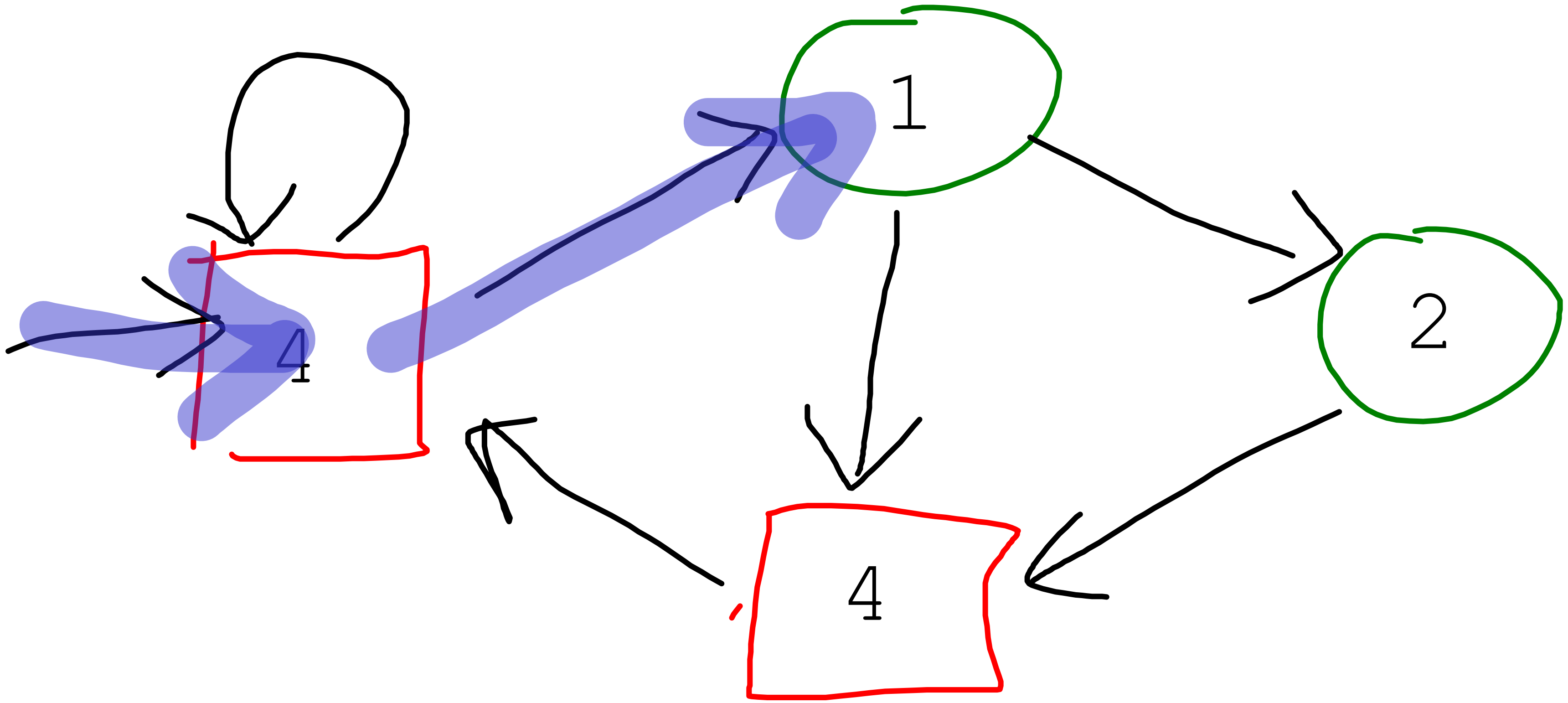
2:

3:

4: |

Count: number of each rank
(without seeing anything smaller)

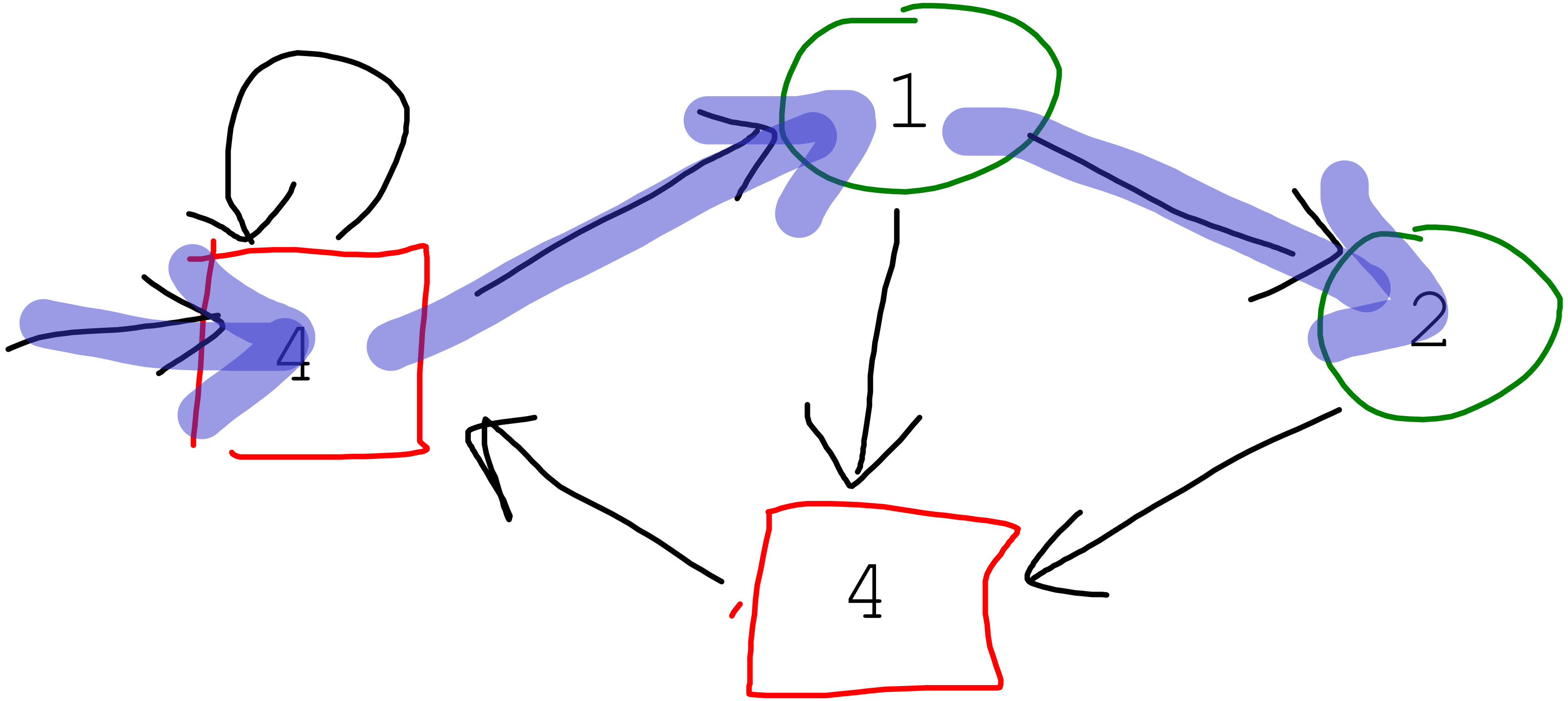
Parity with Counters



Counters:
1: |
2:
3:
4:

Count: number of each rank
(without seeing anything smaller)

Parity with Counters



Counters:

1: |

2: |

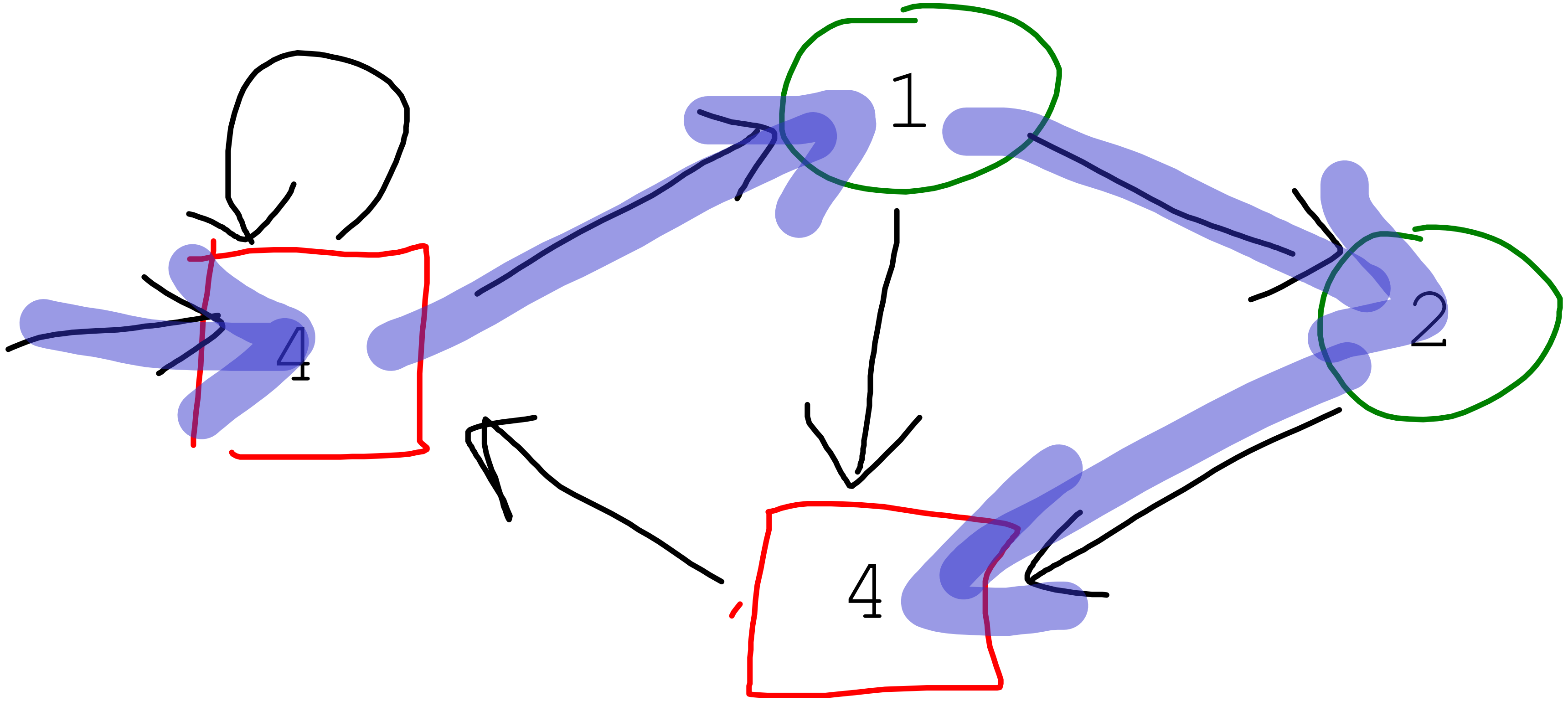
3:

4:

Count: number of each rank

(without seeing anything smaller)

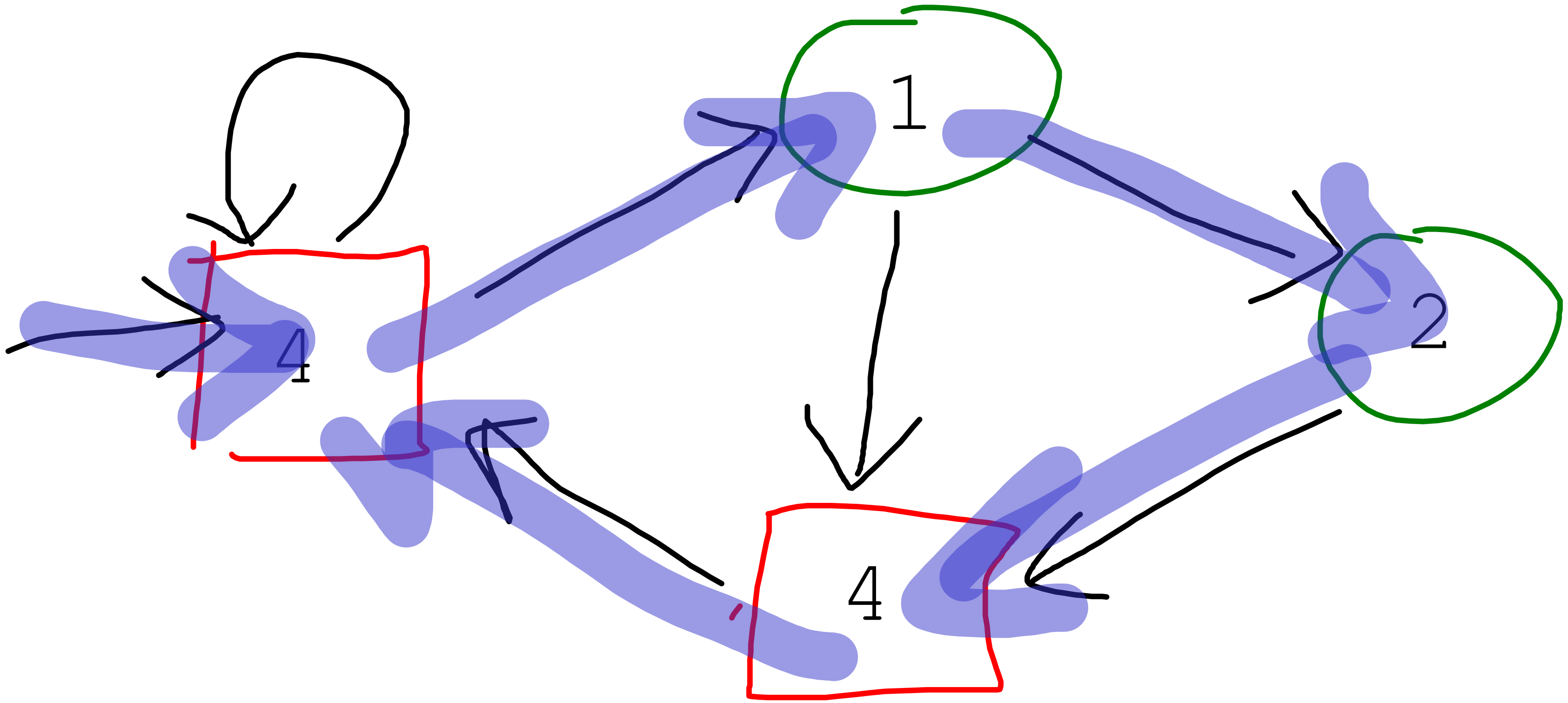
Parity with Counters



Counters:
1: |
2: |
3:
4: |

Count: number of each rank
(without seeing anything smaller)

Parity with Counters



Counters:

1: |

2: |

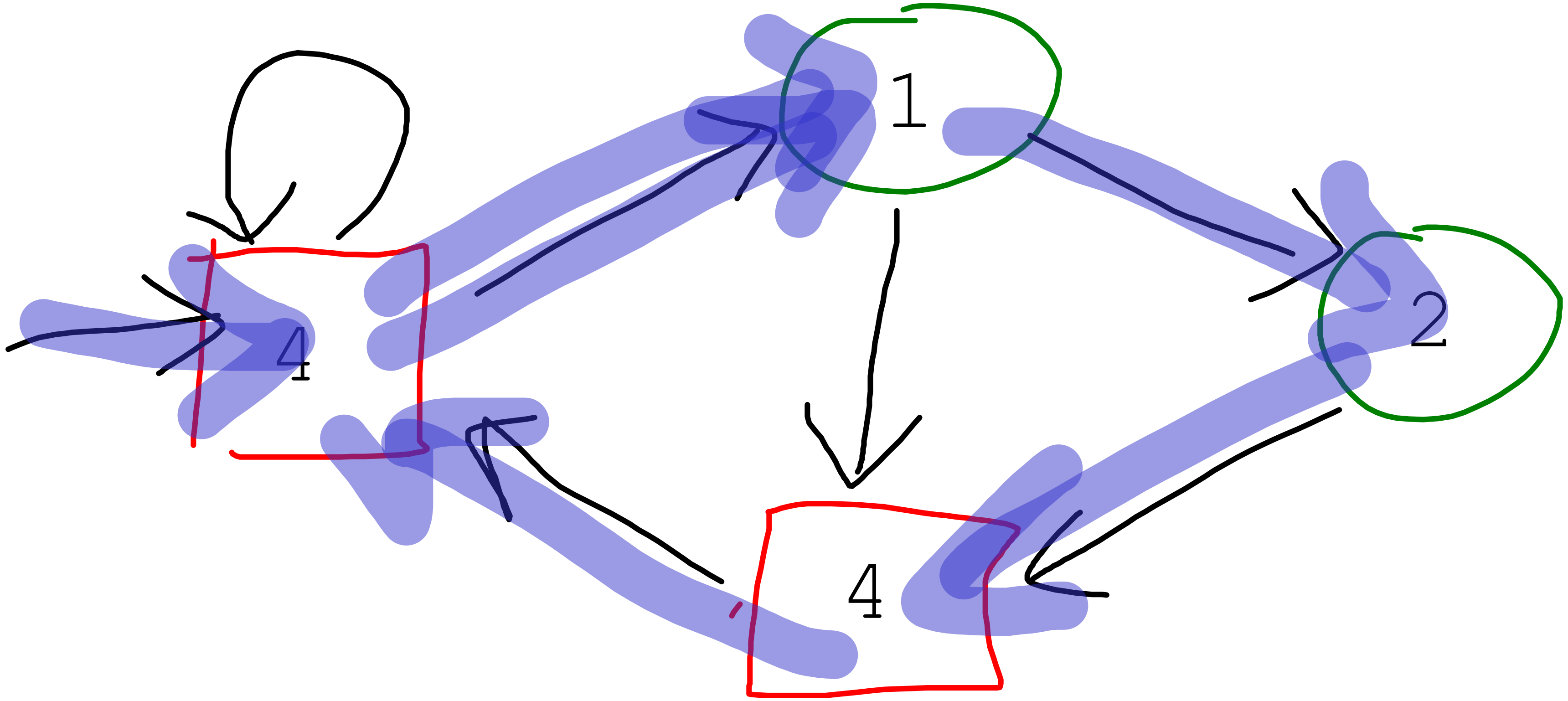
3:

4: ||

Count: number of each rank

(without seeing anything smaller)

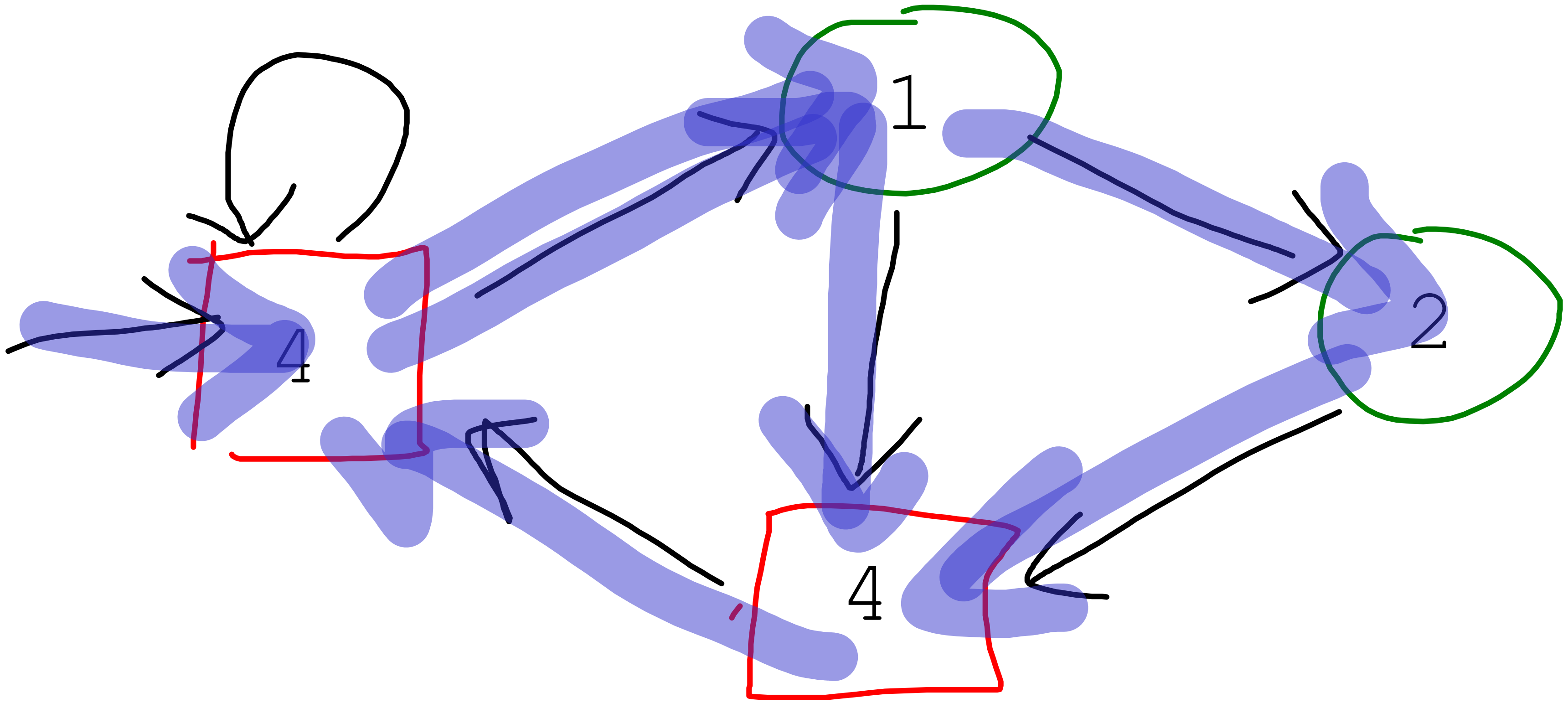
Parity with Counters



Counters:
1: ||
2:
3:
4:

Count: number of each rank
(without seeing anything smaller)

Parity with Counters



Counters:

1: ||

2:

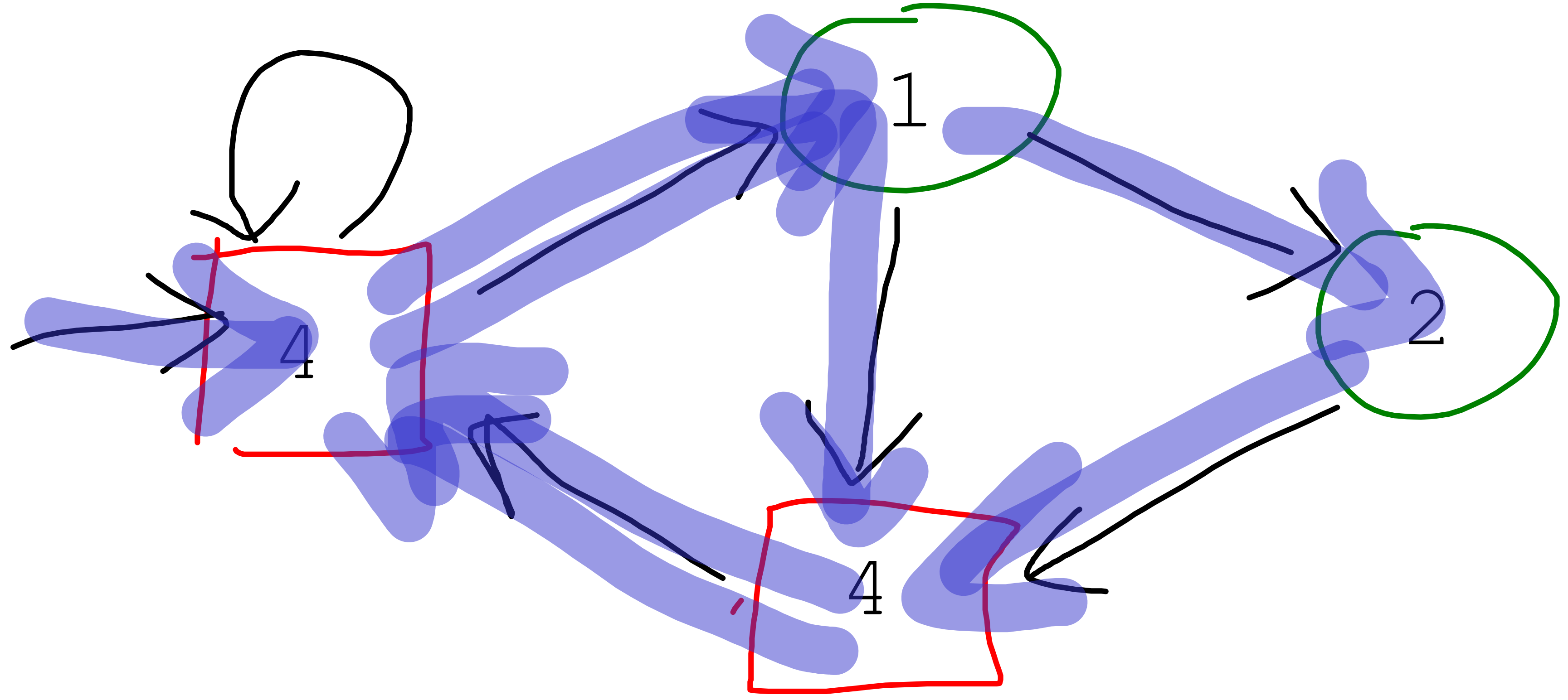
3:

4: |

Count: number of each rank

(without seeing anything smaller)

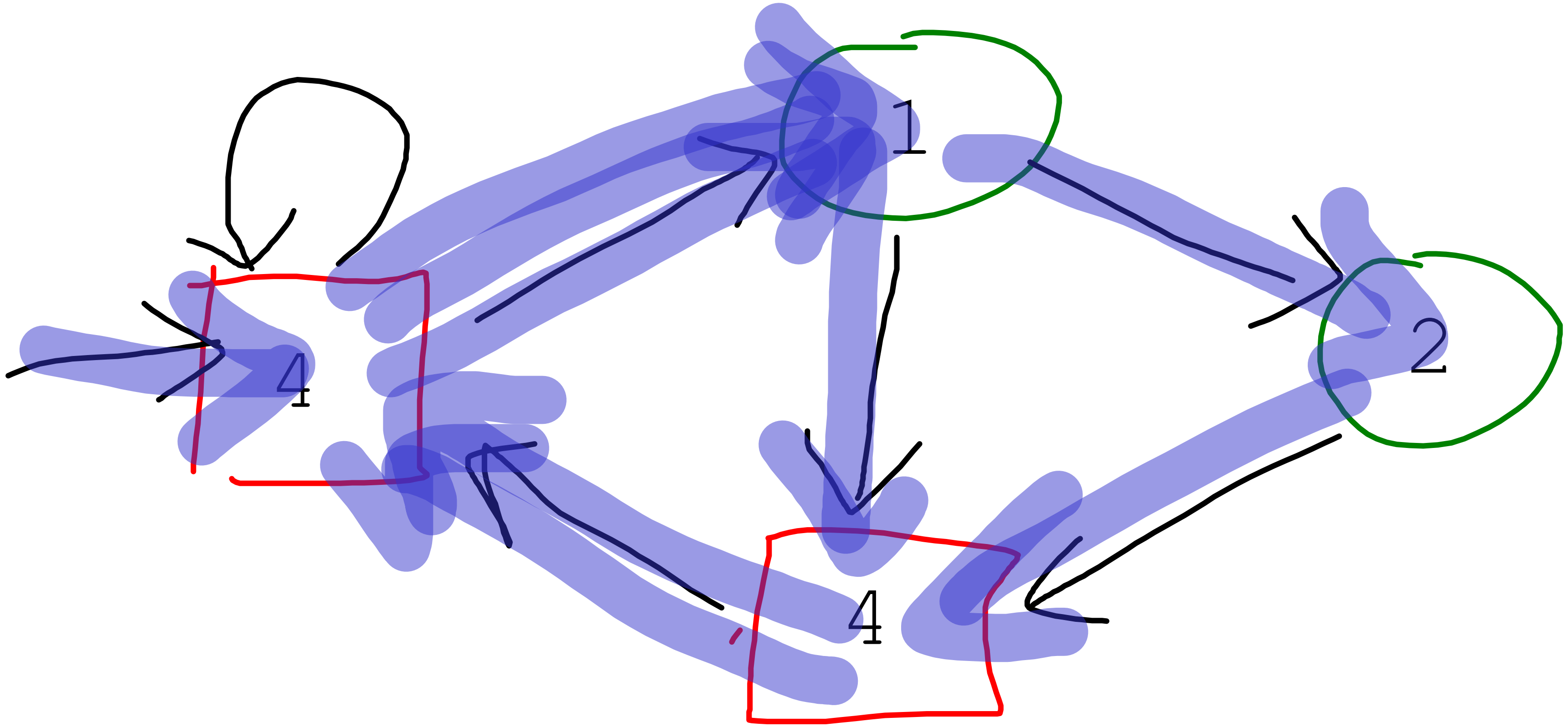
Parity with Counters



Counters:
1: ||
2:
3:
4: ||

Count: number of each rank
(without seeing anything smaller)

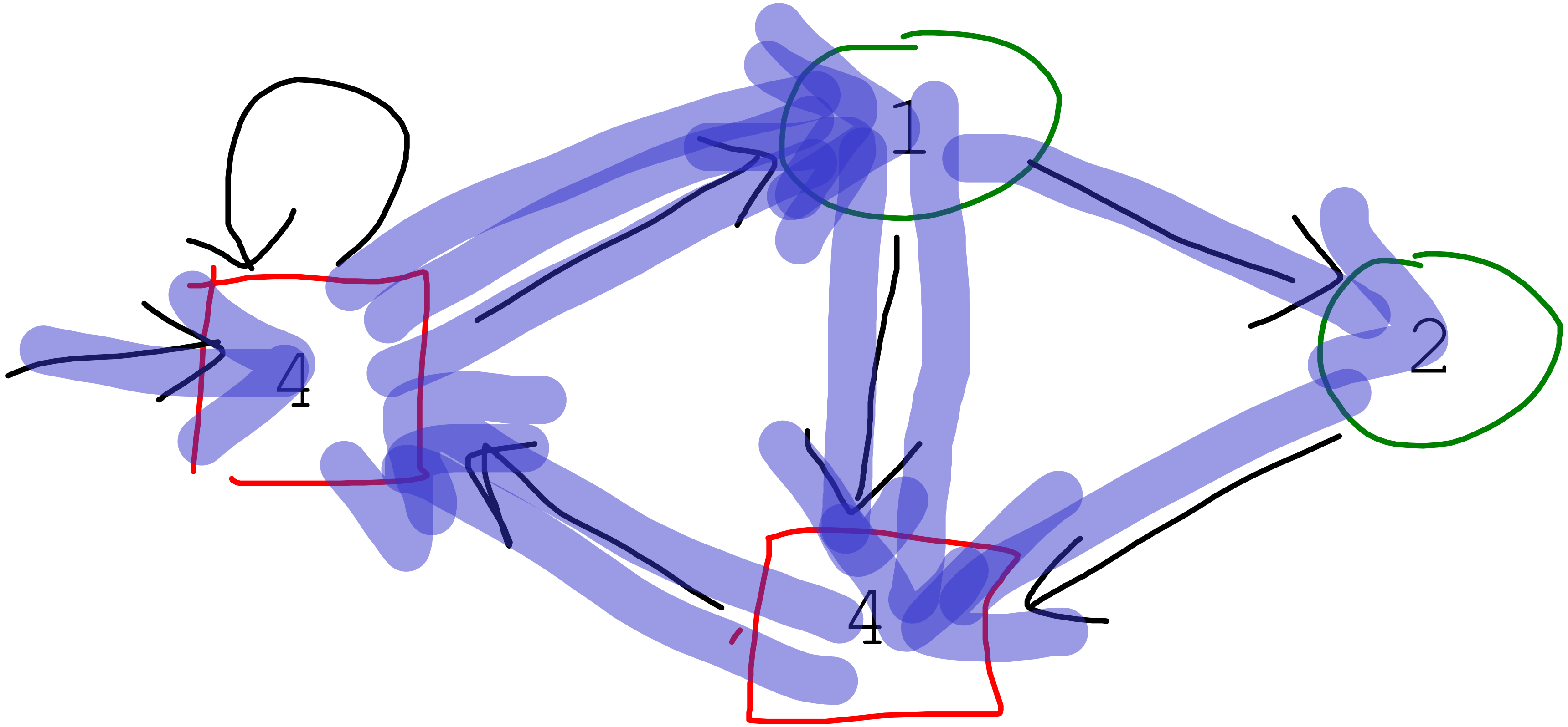
Parity with Counters



Counters:
1: |||
2:
3:
4:

Count: number of each rank
(without seeing anything smaller)

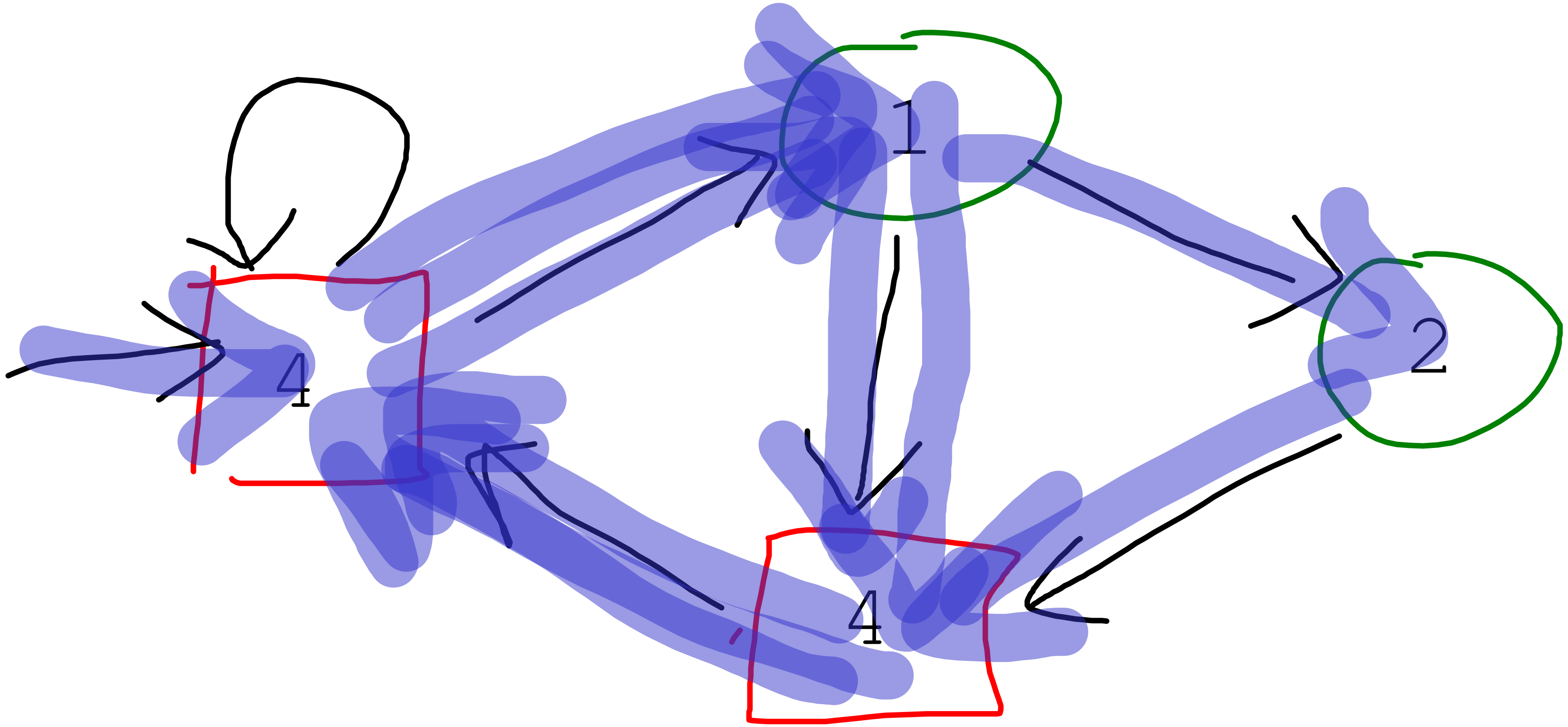
Parity with Counters



Counters:
1: |||
2:
3:
4: |

Count: number of each rank
(without seeing anything smaller)

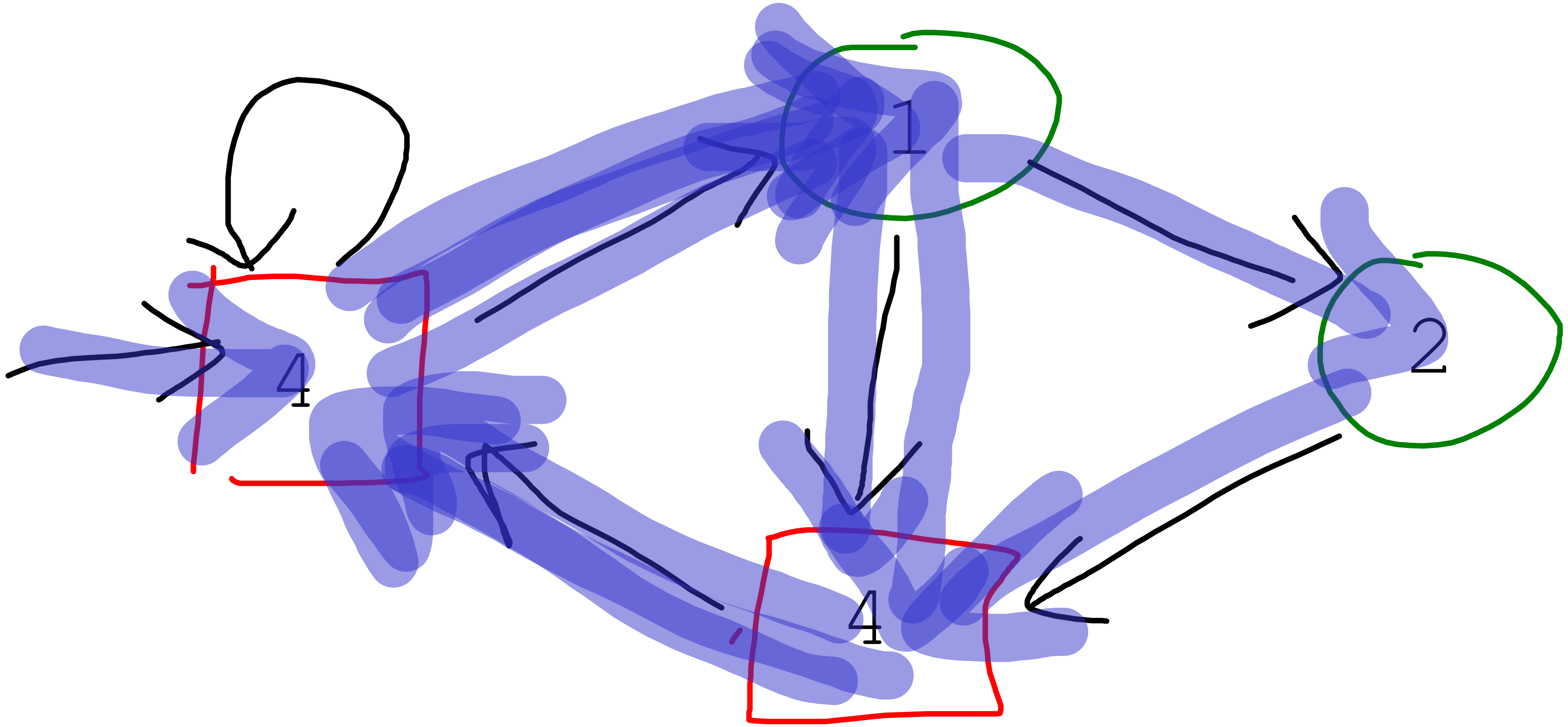
Parity with Counters



Counters:
1: |||
2:
3:
4: ||

Count: number of each rank
(without seeing anything smaller)

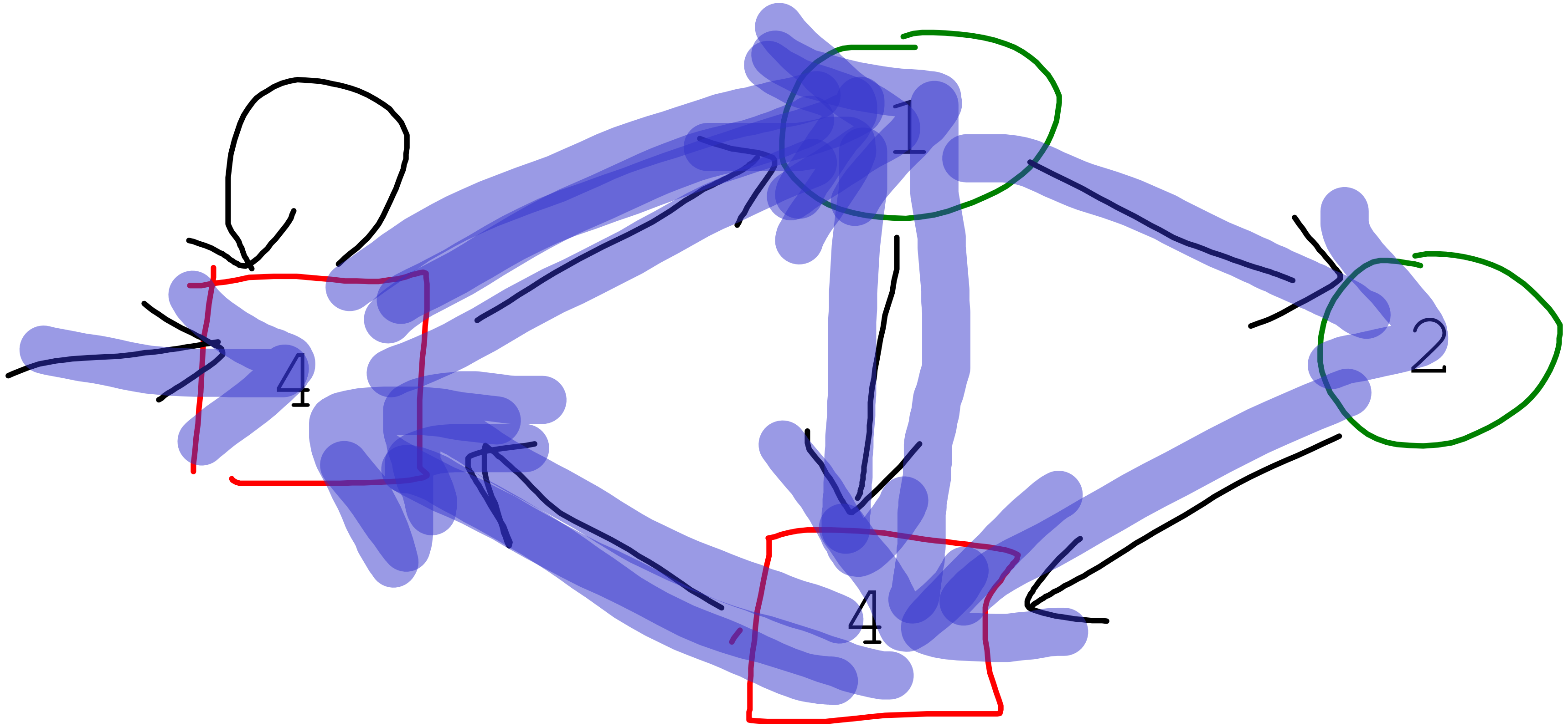
Parity with Counters



Counters:
1: |||||
2:
3:
4:

Count: number of each rank
(without seeing anything smaller)

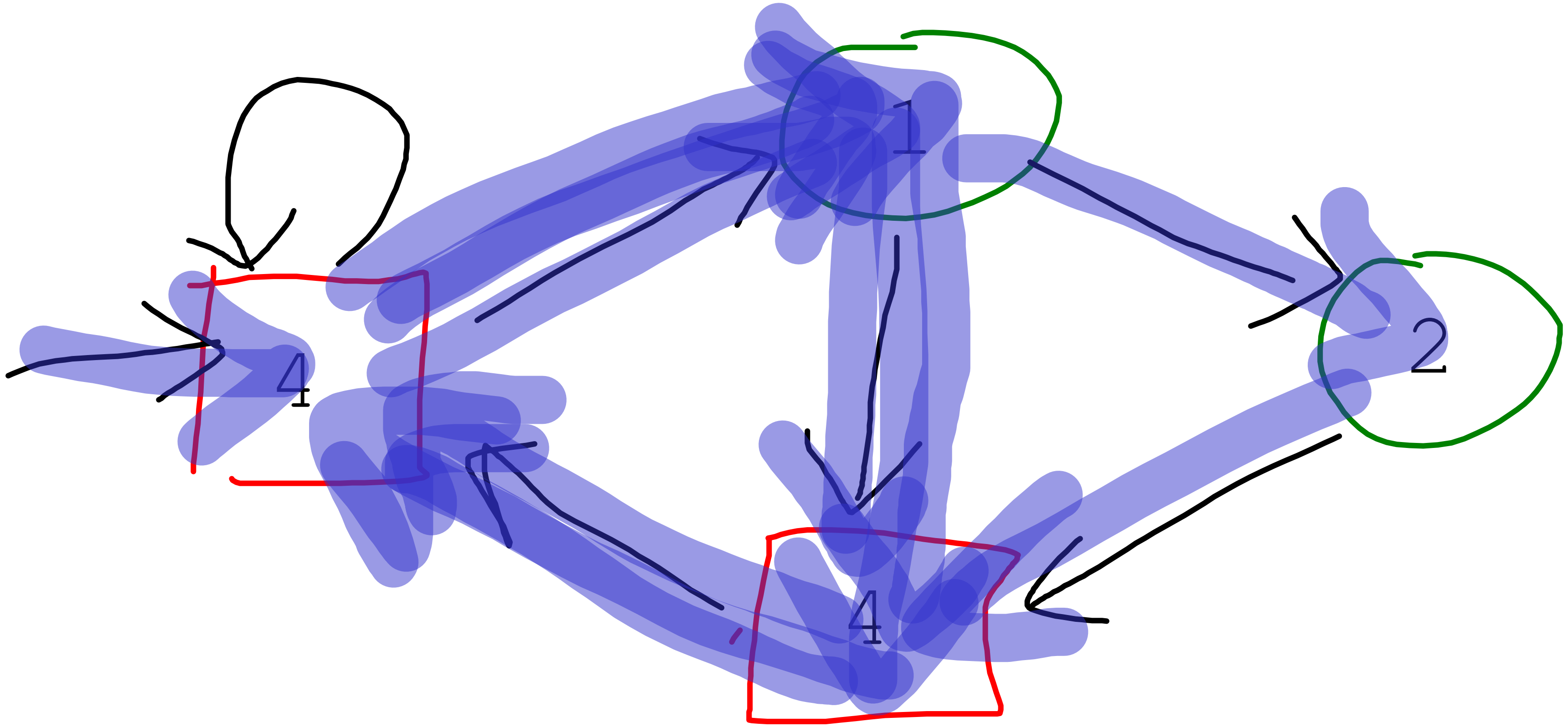
Parity with Counters



Counters:
1: |||||
2:
3:
4:

Count: number of each rank
(without seeing anything smaller)

Parity with Counters



Counters:

1: ||||

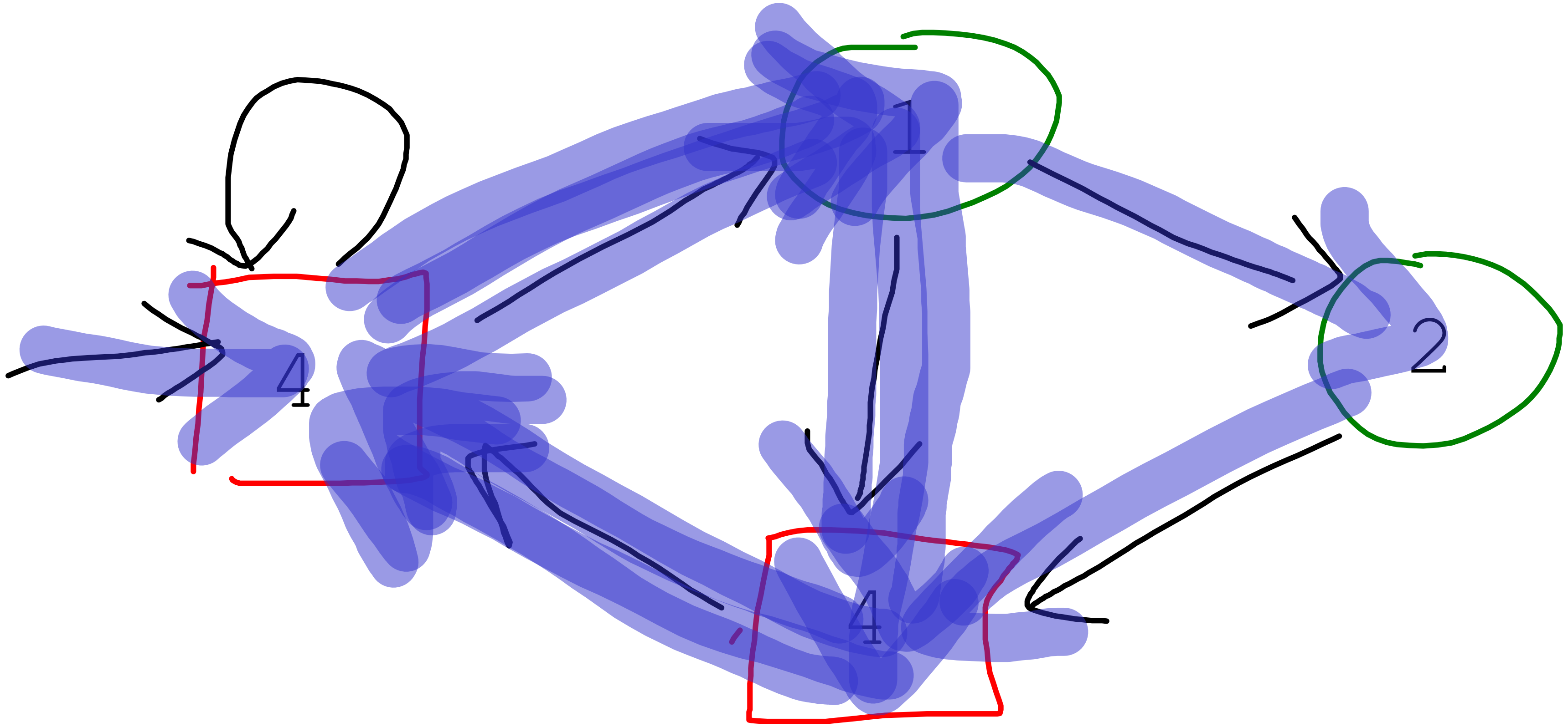
2:

3:

4: |

Count: number of each rank
(without seeing anything smaller)

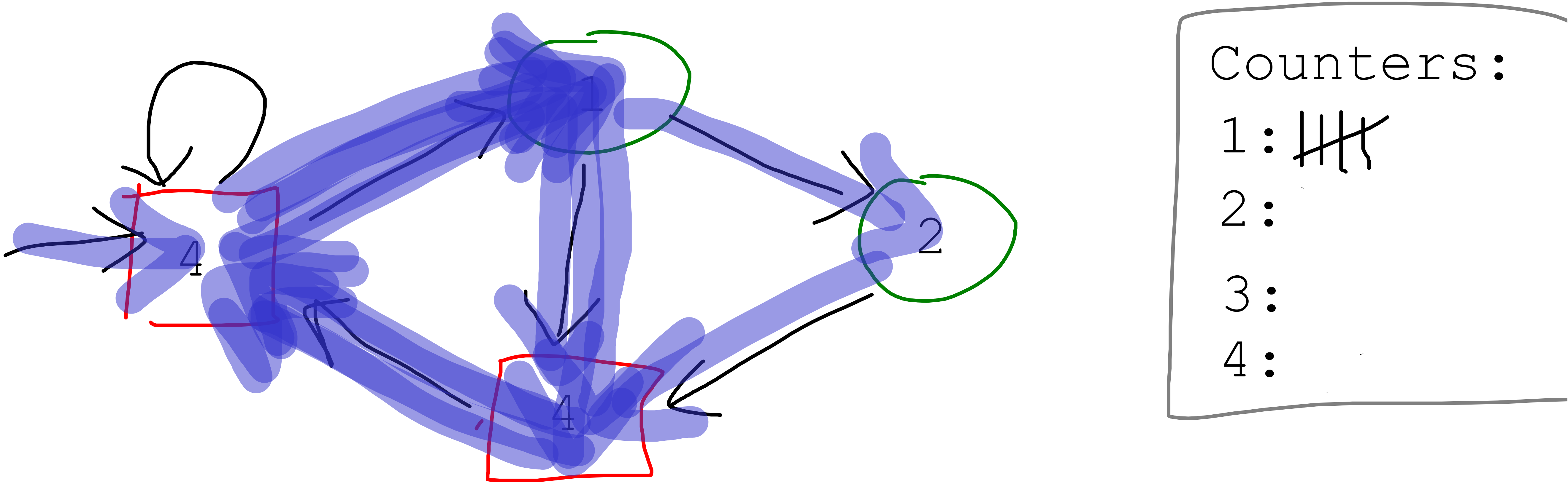
Parity with Counters



Counters:
1: ||||
2:
3:
4: ||

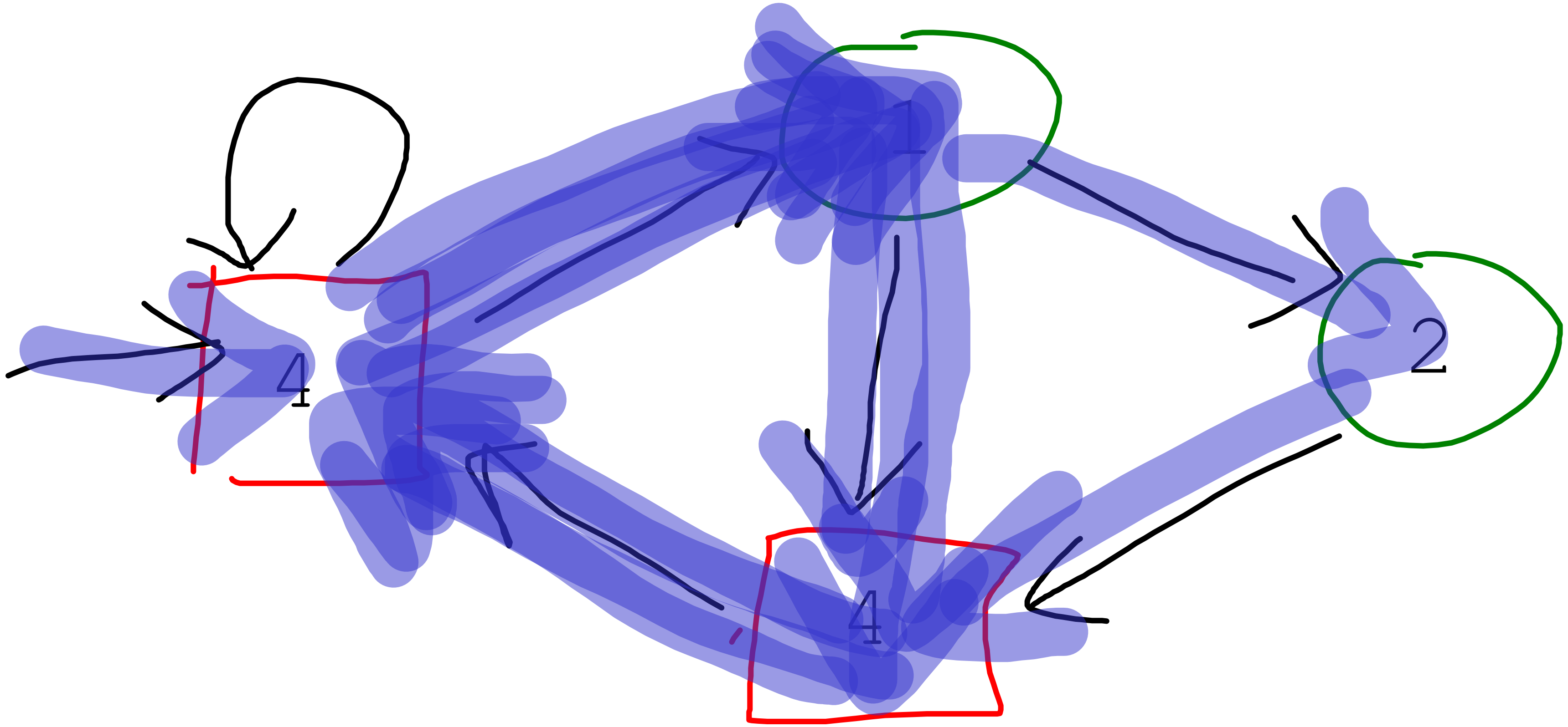
Count: number of each rank
(without seeing anything smaller)

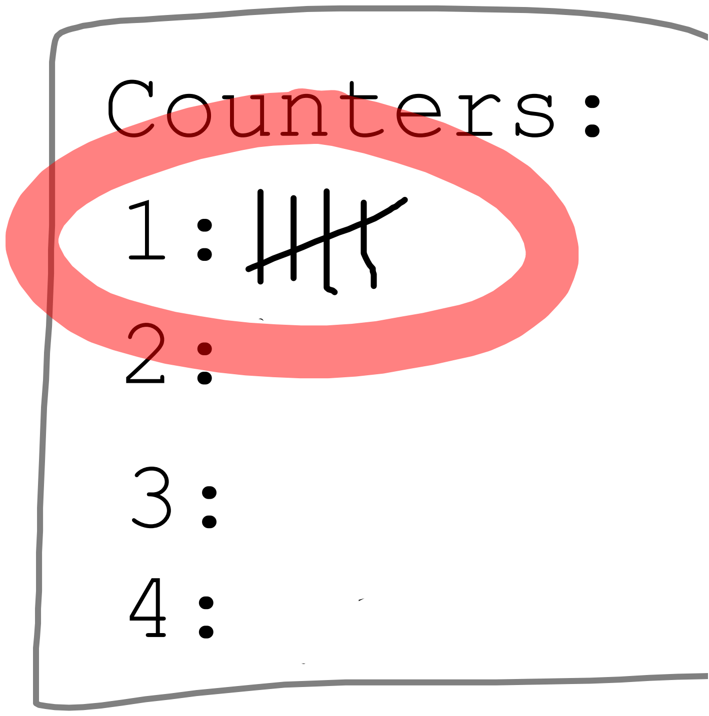
Parity with Counters



Count: number of each rank
(without seeing anything smaller)

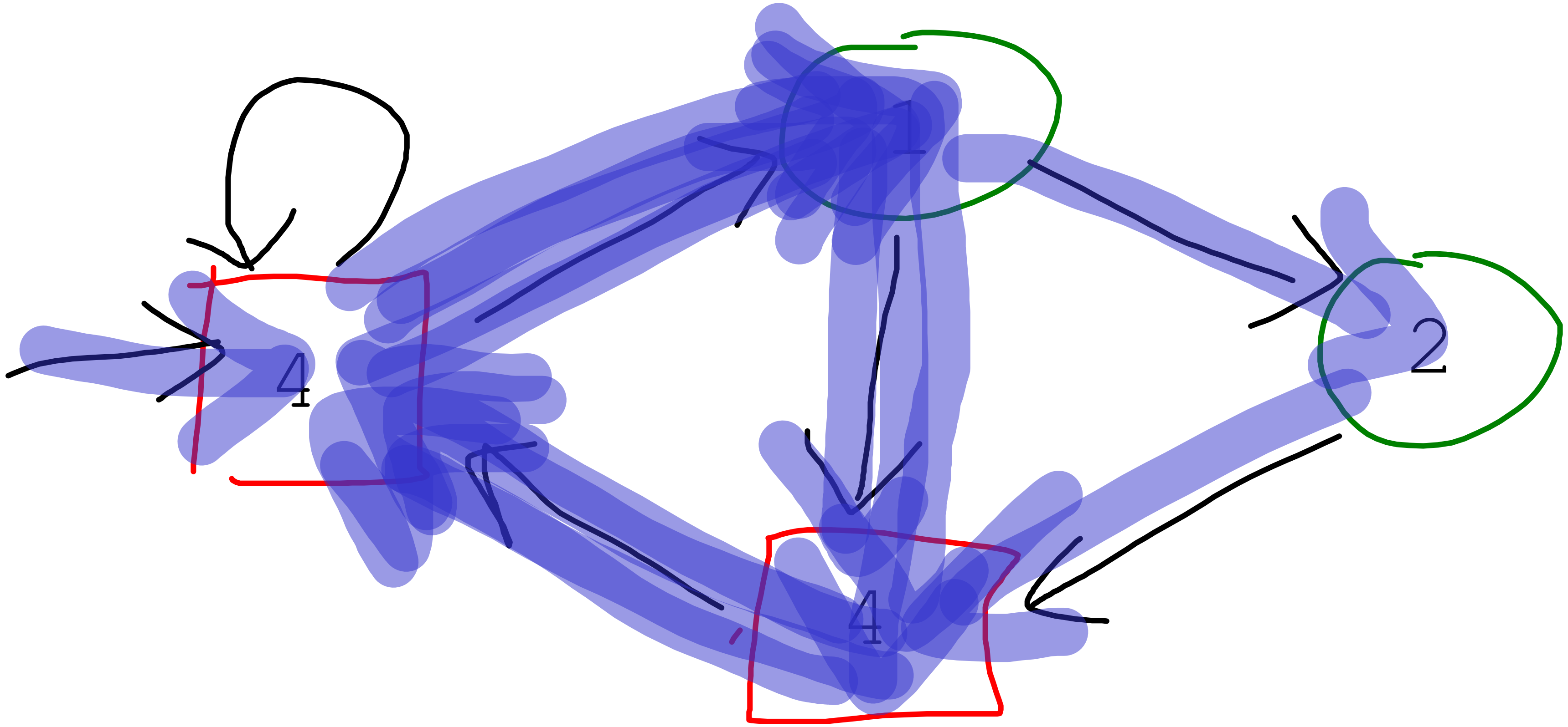
Parity with Counters

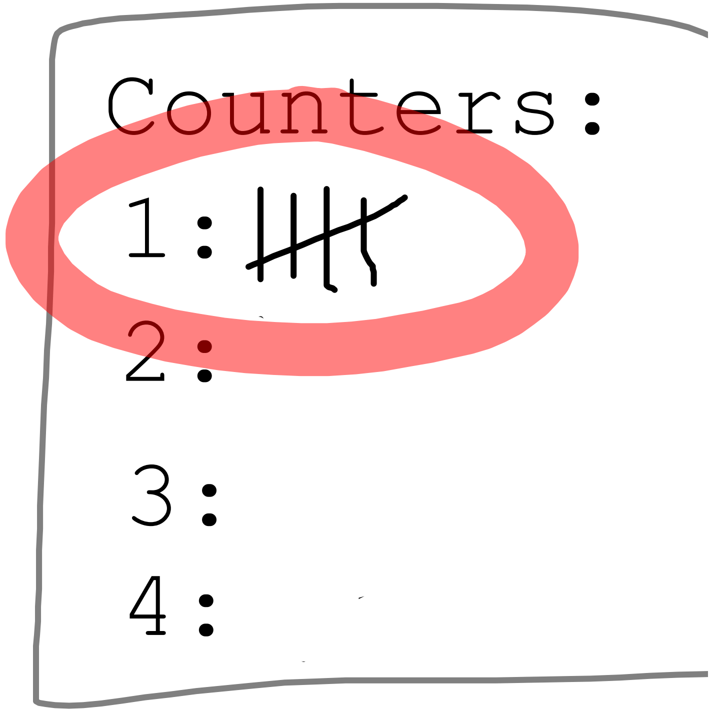




There is a loop!
(Hit 1 five times over 4 states)

Parity with Counters





There is a loop!
(Hit 1 five times over 4 states)

The smallest rank is odd: Elvis loses!

Parity to Safety

Reduce parity to safety game (Bernet et al):

- Keep counters in state (n * n^k states)
- Elvis loses if odd counter is high

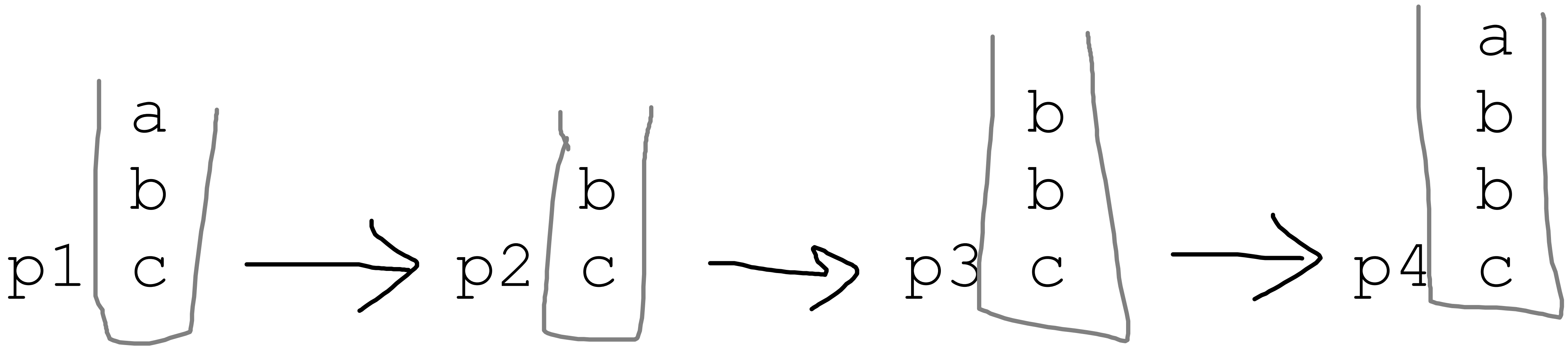
Parity to Safety

Reduce parity to safety game (Bernet et al):

- Keep counters in state (n * n^k states)
- Elvis loses if odd counter is high

Exponential blow up!

Pushdown Games



Each state has:

Control state (from finite set)

Stack of characters

Model recursive programs

Pushdown Parity to Safety

Pushdown Parity

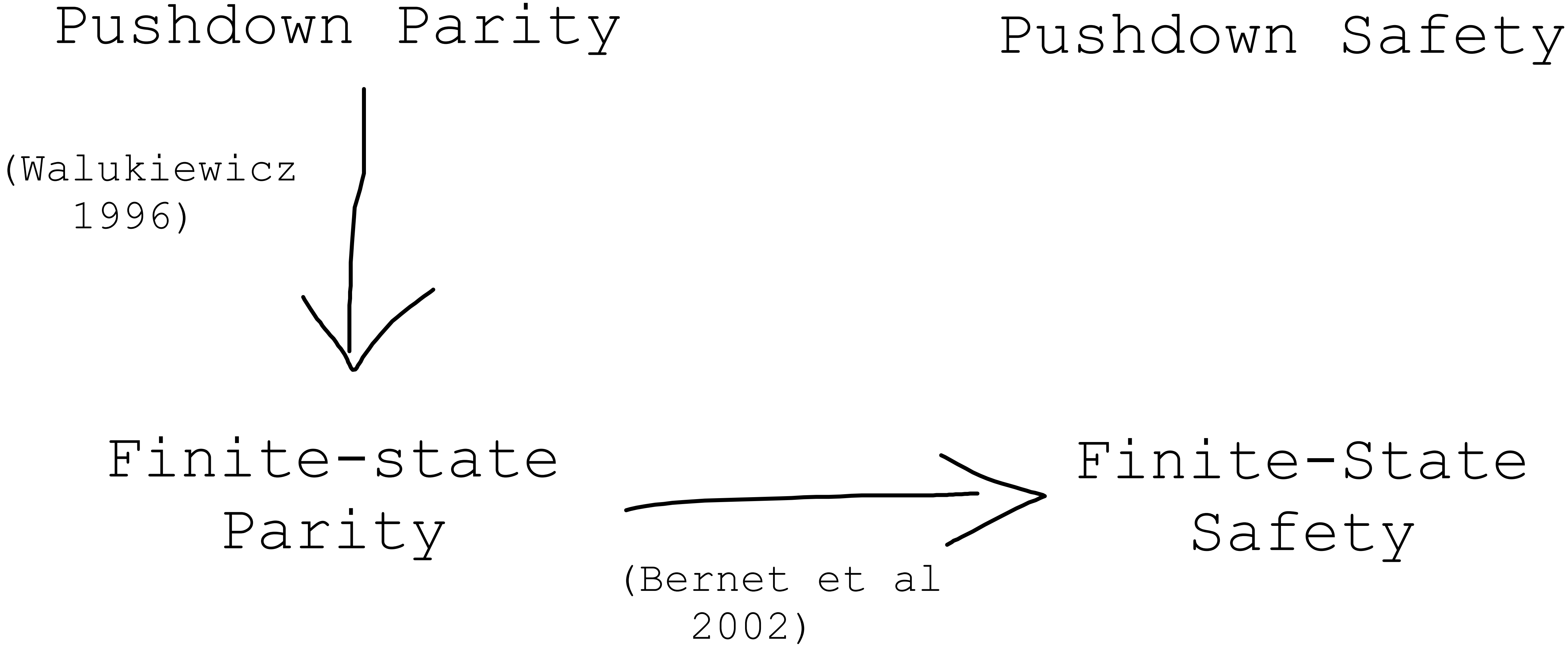
Pushdown Safety

(Walukiewicz 1996)

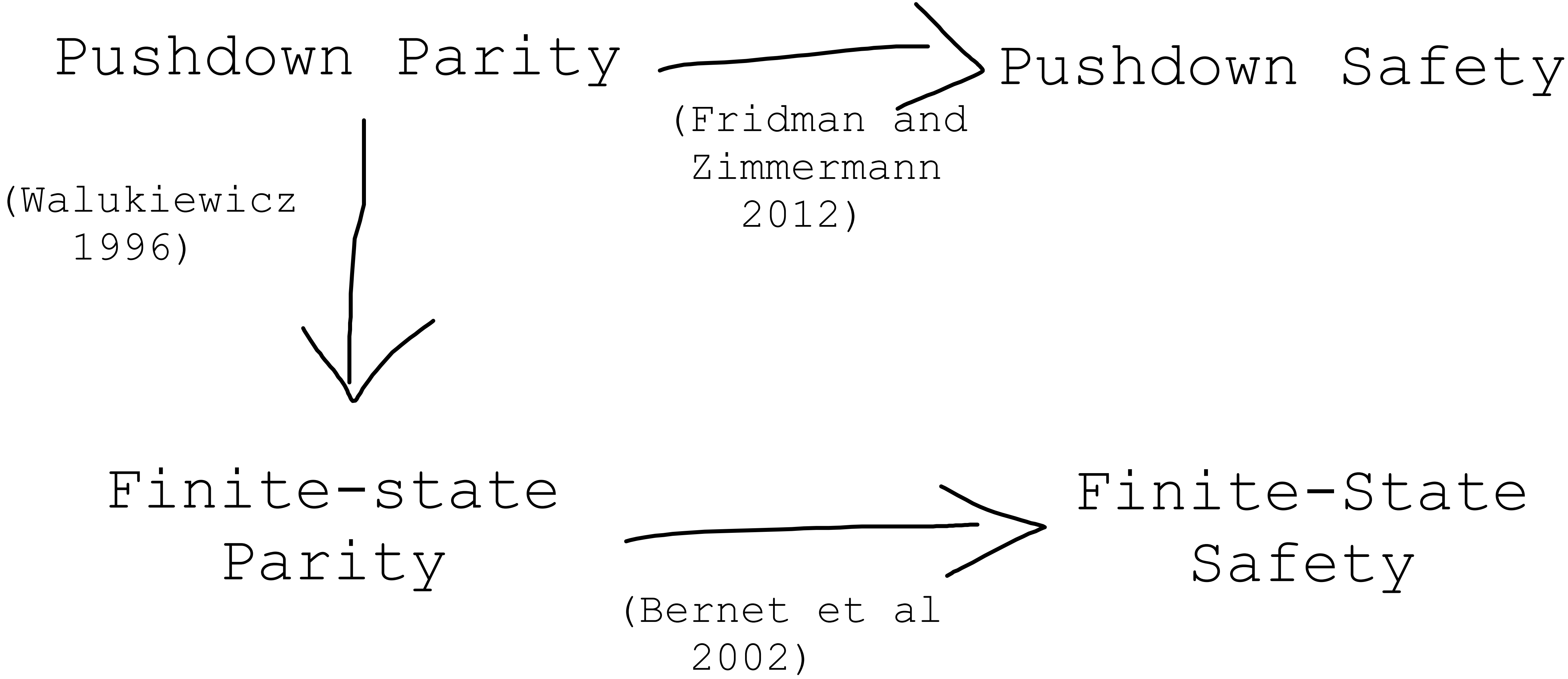
Finite-state Parity

Finite-State Safety

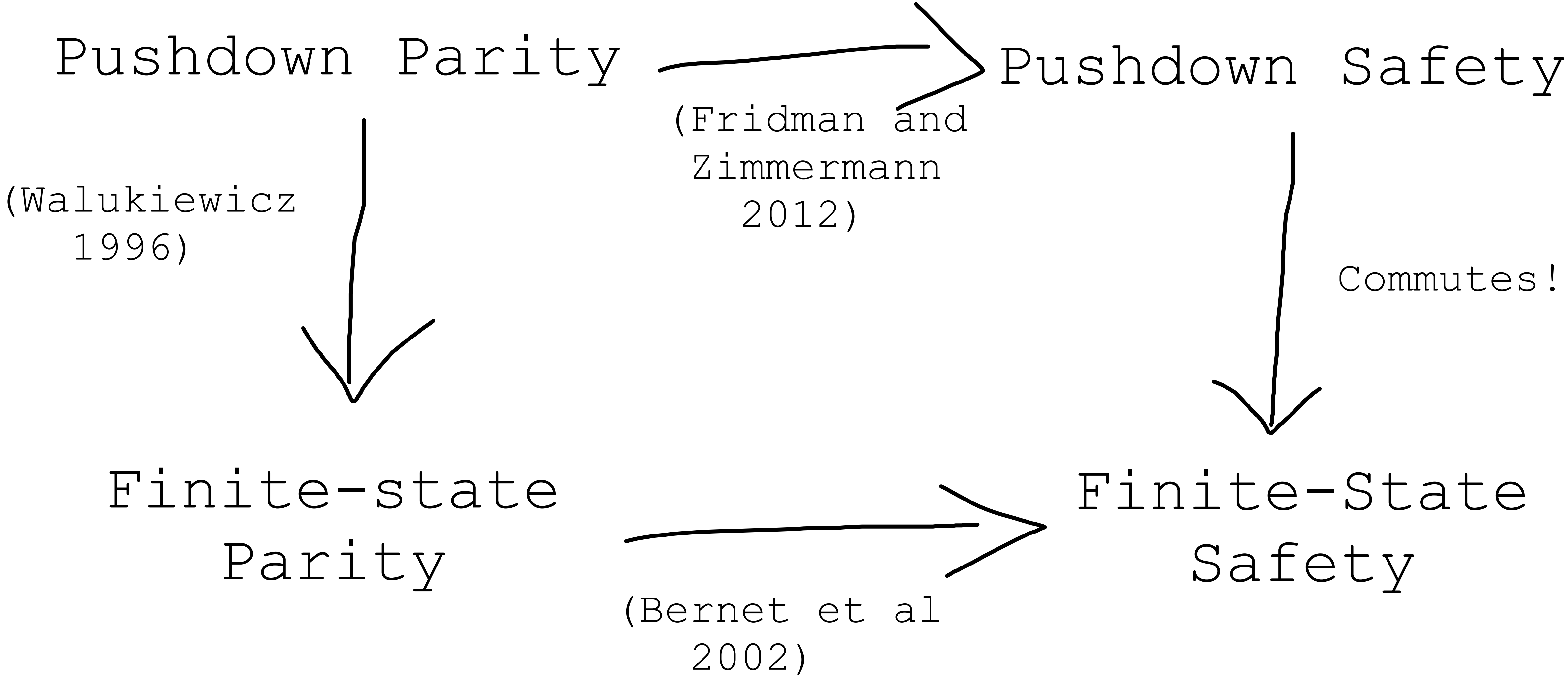
Pushdown Parity to Safety



Pushdown Parity to Safety

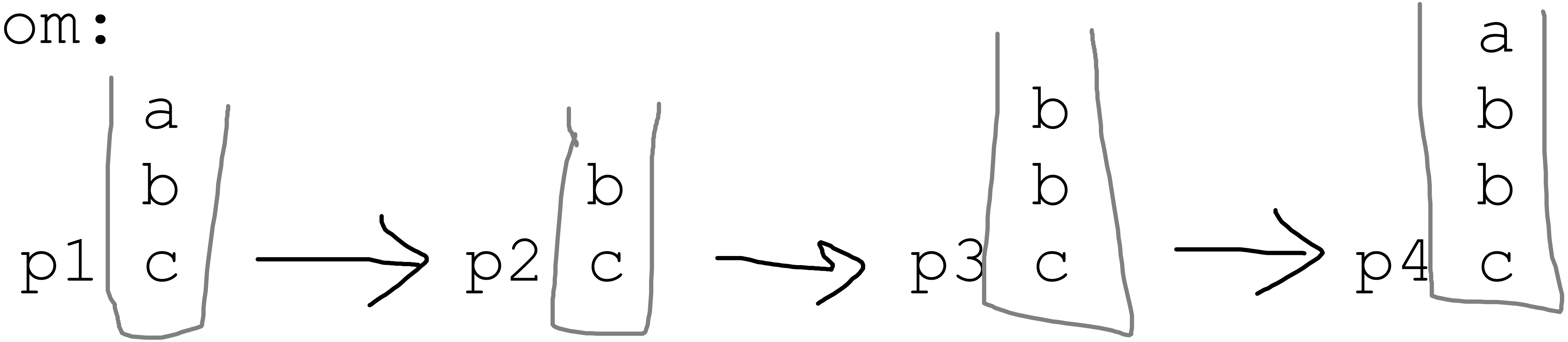


Pushdown Parity to Safety

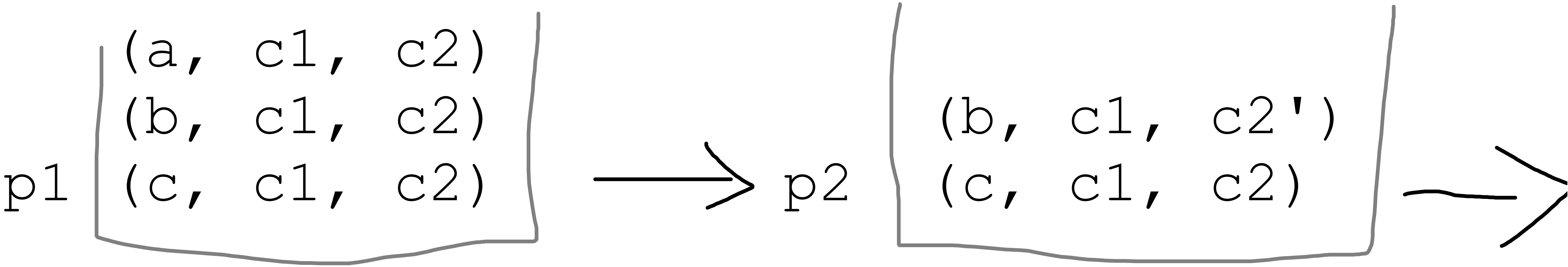


Pushdown Parity to Safety

From:

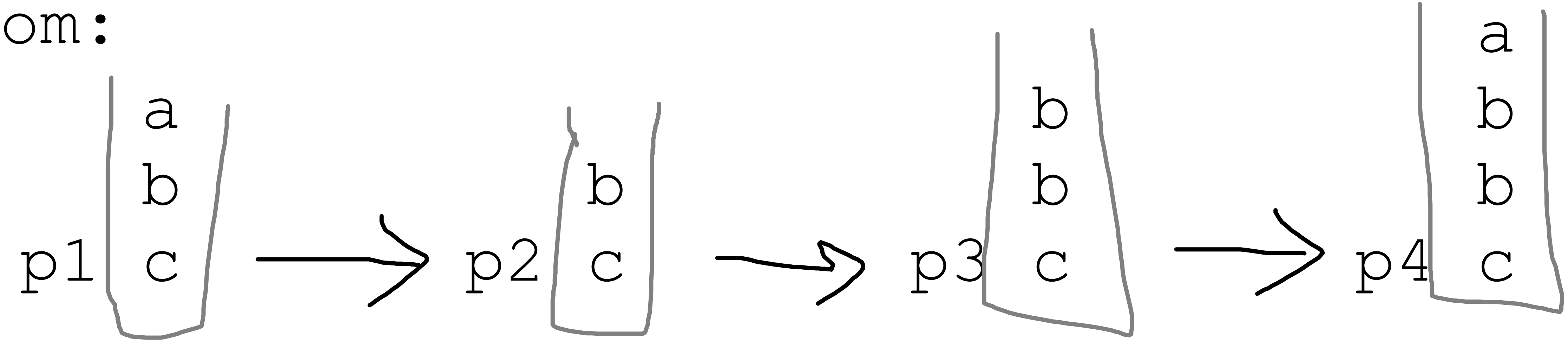


To:

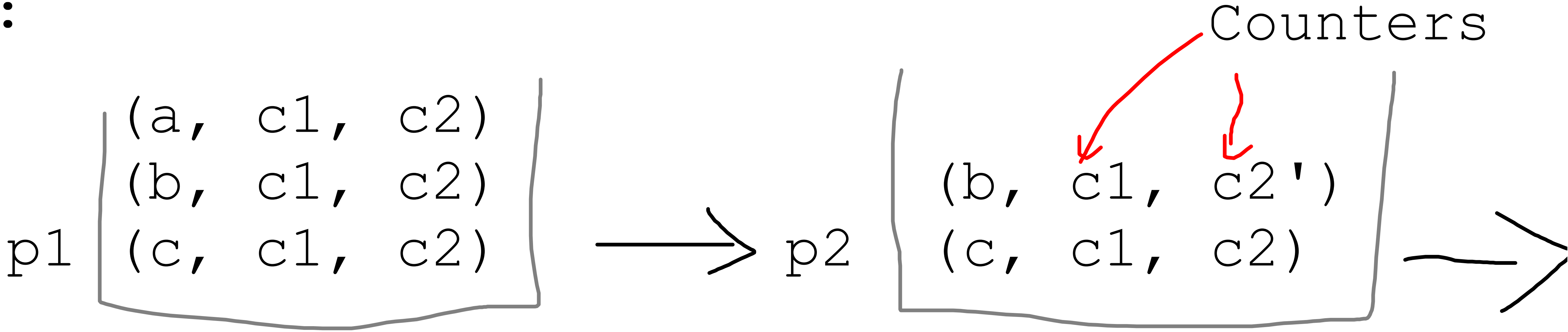


Pushdown Parity to Safety

From:



To:



Exponential Blow Up?

```
      (a, c1, c2')
p     (b, c1, c2)
```

- Counter values: can be exponential (Pushdown->finite-state has 2^n states)
- Number of stack characters: (2^n)^k (For k ranks)

Exponential Blow Up?

```
           (a, c1, c2')
p          (b, c1, c2)
```

- Counter values: can be exponential (Pushdown->finite-state has 2^n states)
- Number of stack characters: (2^n)^k (For k ranks)

Reducing Size

From:

```
p   (a, c1, c2')
    (b, c1, c2)
```

To:

```
    a
    c2'
    c1
    b
    c2
p   c1
```

Reducing Size

From:

```
p   (a, c1, c2')
    (b, c1, c2)
```

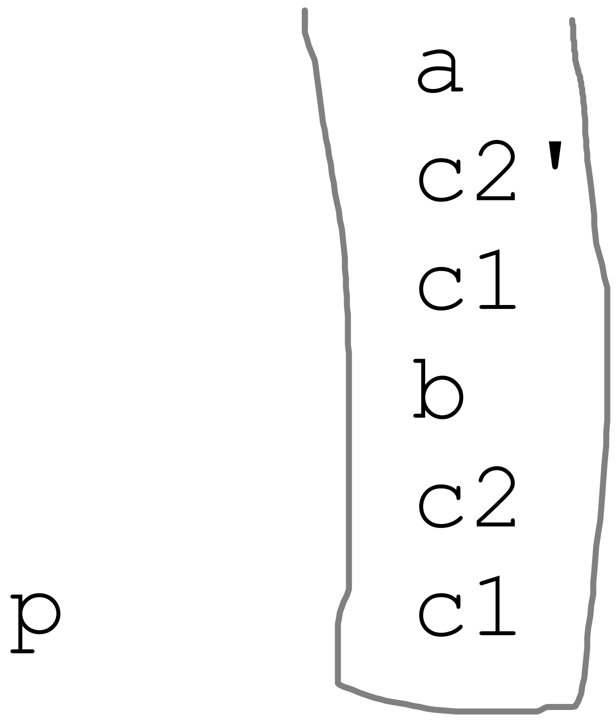
To:

```
    a
    c2'
    c1
    b
    c2
p   c1
```

Number of characters: 2^n

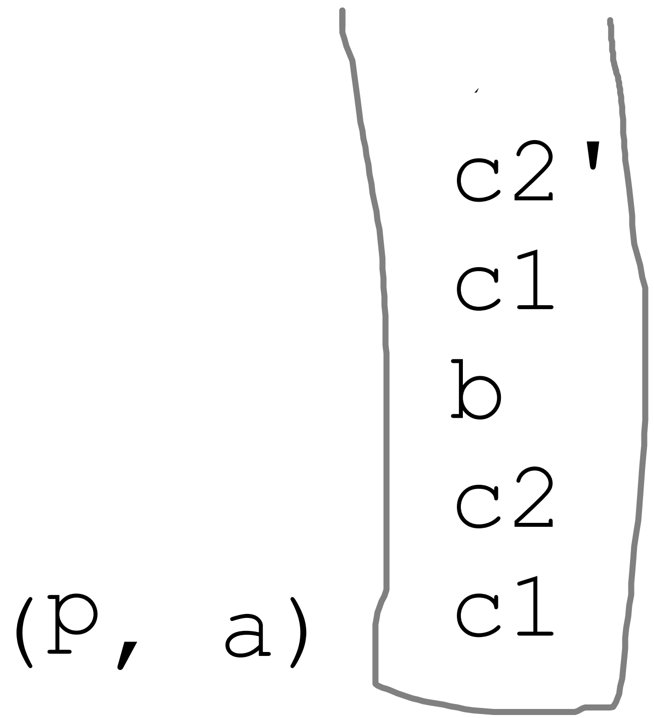
Updating Counters

Increment c1



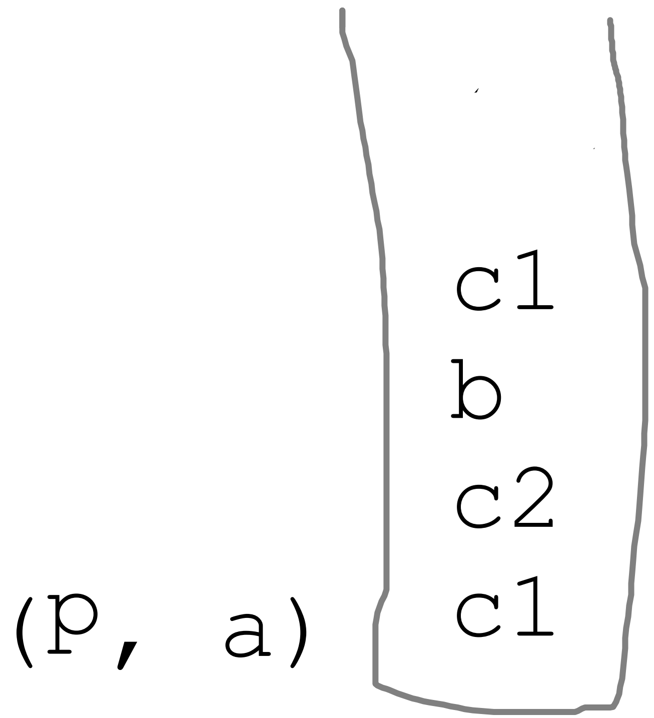
Updating Counters

Increment c1



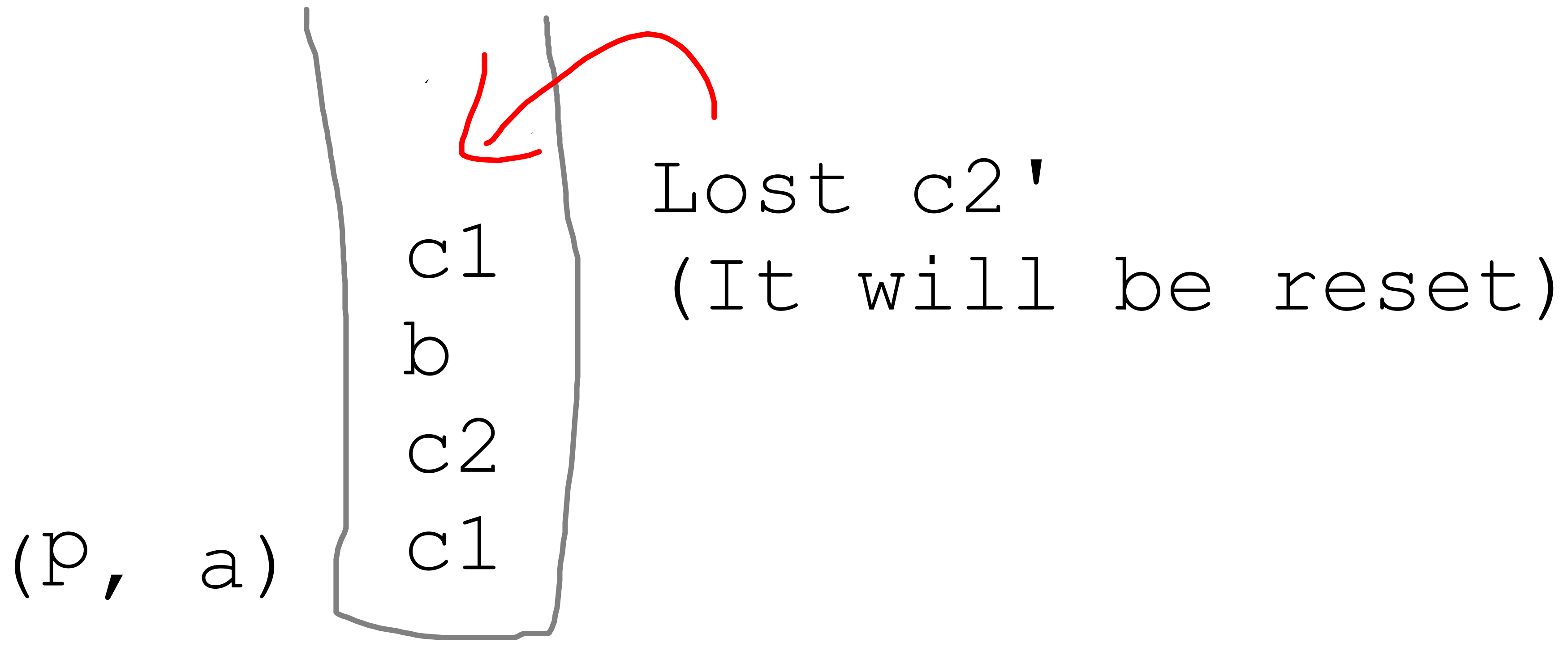
Updating Counters

Increment c1



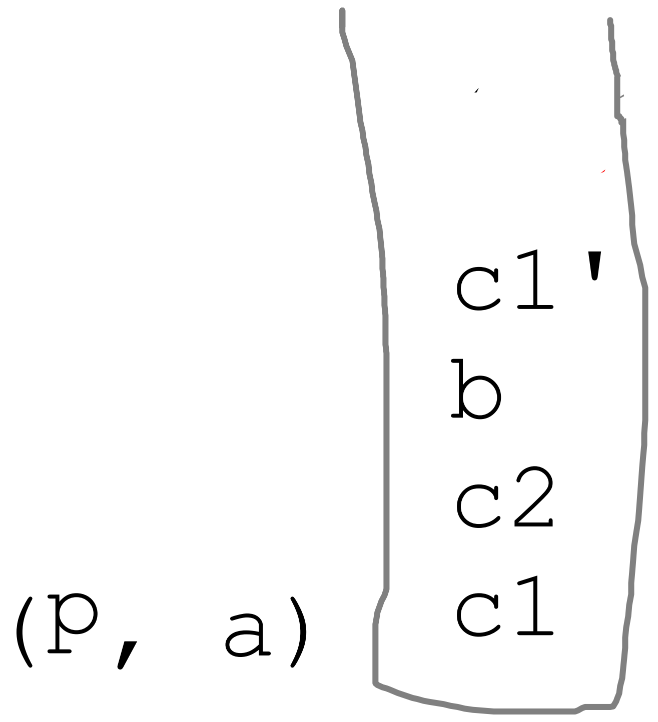
Updating Counters

Increment c1



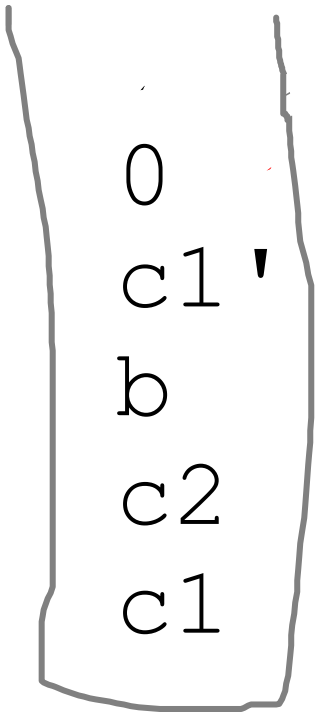
Updating Counters

Increment c1



Updating Counters

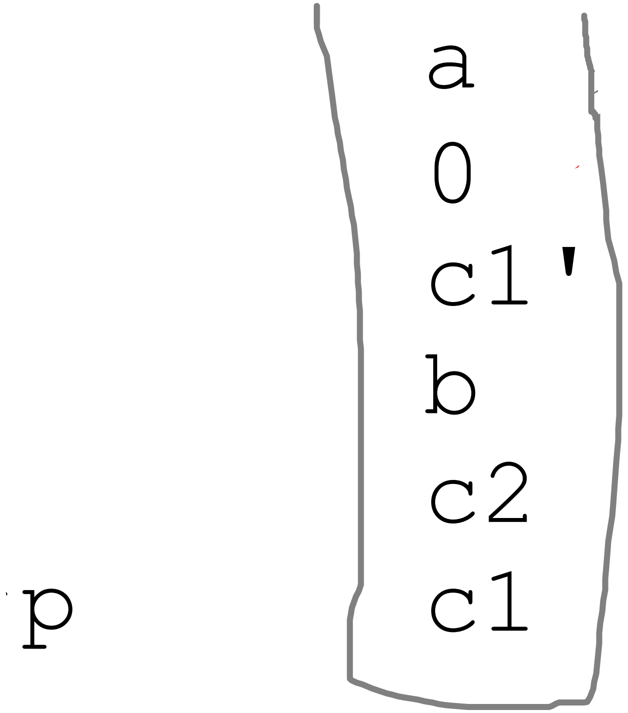
Increment c1



(p, a)

Updating Counters

Increment c1

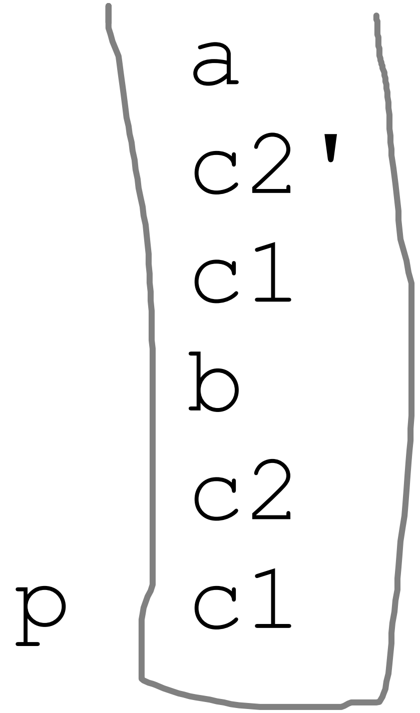


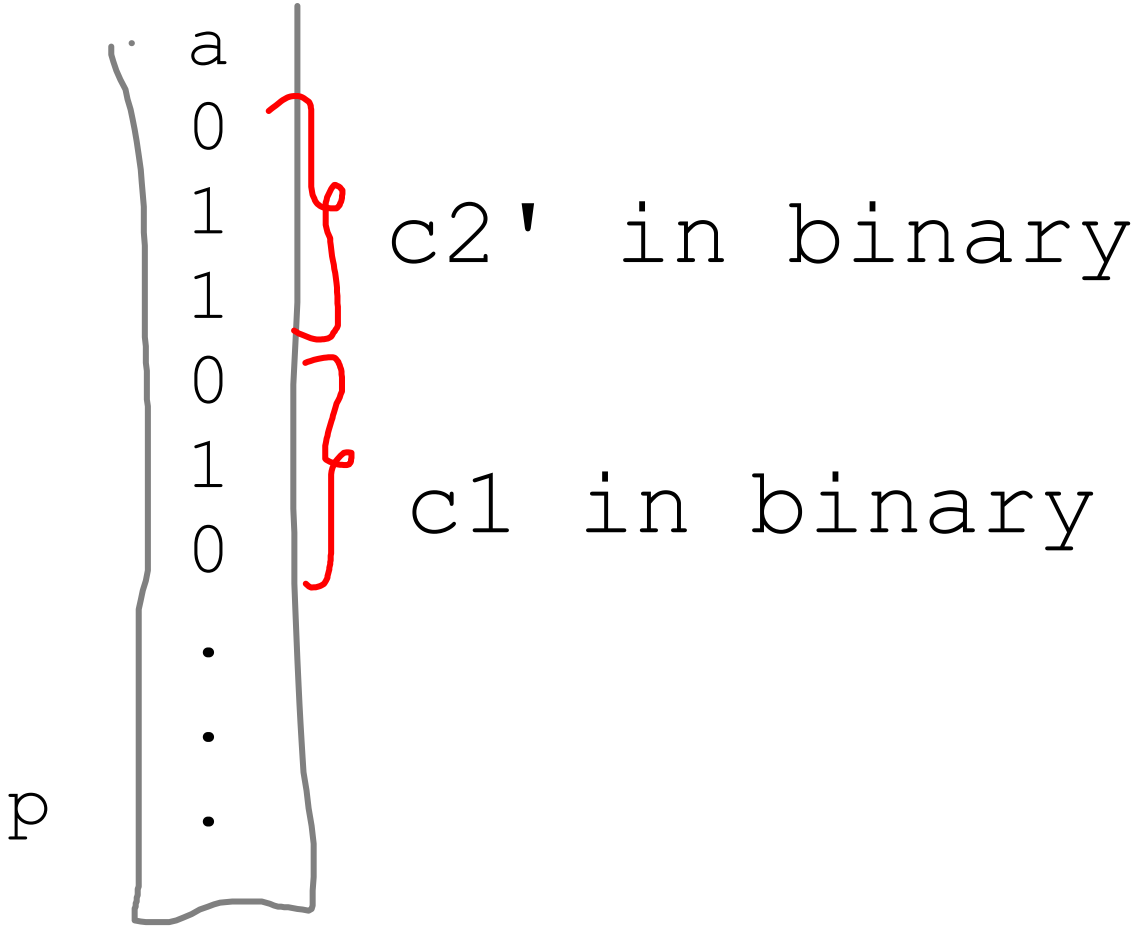
Polynomial No. of Characters

a
c2'
c1
b
c2
p c1

Counters up to 2^n
Alphabet exponential

Polynomial No. of Characters

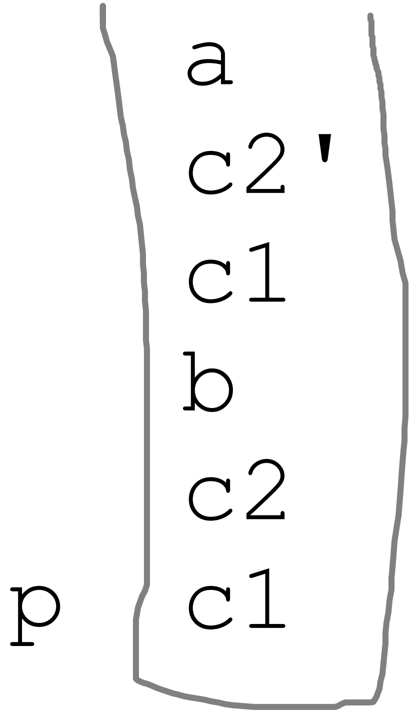




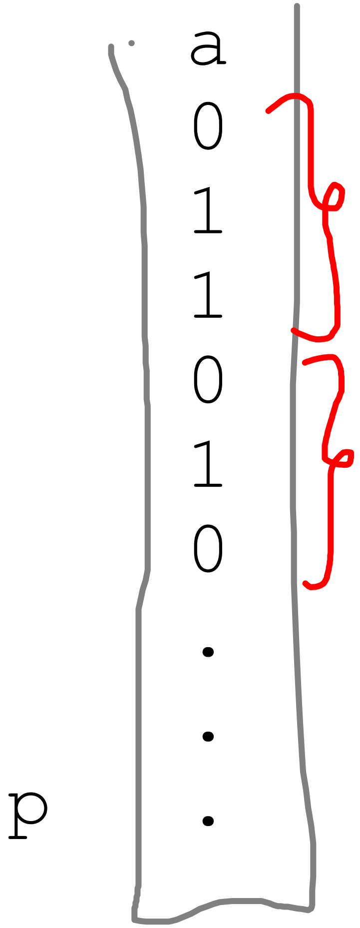
Counters up to 2^n

Alphabet exponential

Polynomial No. of Characters



Counters up to 2^n
Alphabet exponential



c2' in binary

c1 in binary

Pushdowns handle length-n binary

Alphabet polynomial!

Pushdown Parity to Safety

Pushdown parity game -> pushdown safety game

- Naively exponential
- Use stack discipline of counters
- Binary counter encoding of pushdowns
- Polynomial time reduction

Collapsible Pushdown Systems

Pushdown Systems = First-Order Recursion

Collapsible Pushdown Systems
=
Higher-Order Recursion

Higher-Order Programming: Niche?

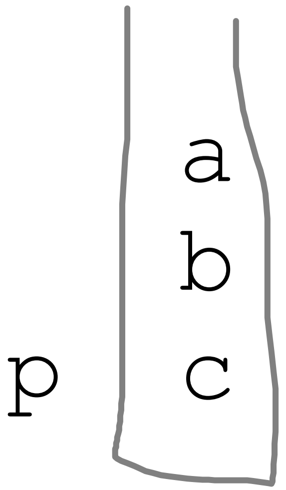
Almost all modern languages support it

- Scala, Go, JavaScript, Python, ...
- Retro-fitted to C++ and Java
- Asynchronous programs/callbacks
- Map/Reduce

Collapsible Pushdown Systems

Higher-order program: functions of functions

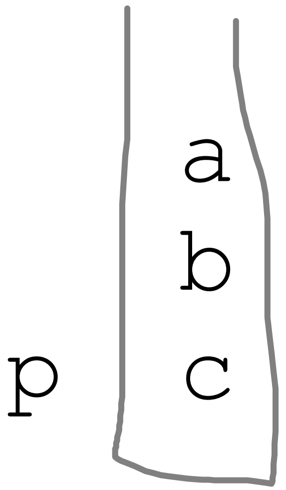
Higher-order pushdown: stack of stacks

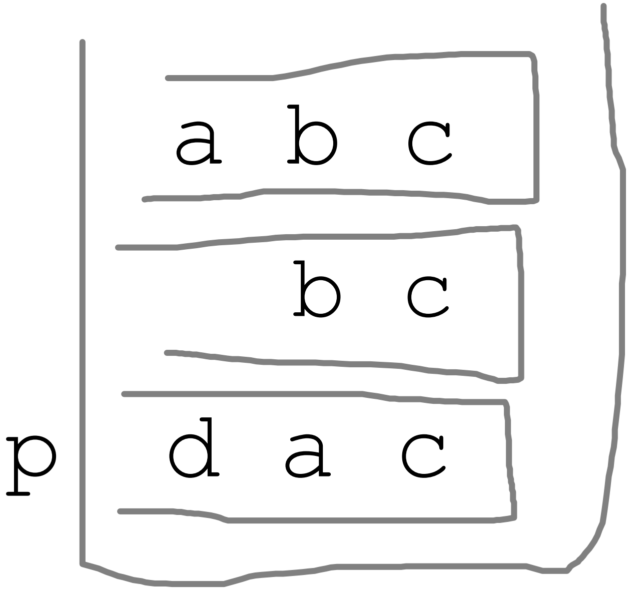


Collapsible Pushdown Systems

Higher-order program: functions of functions

Higher-order pushdown: stack of stacks

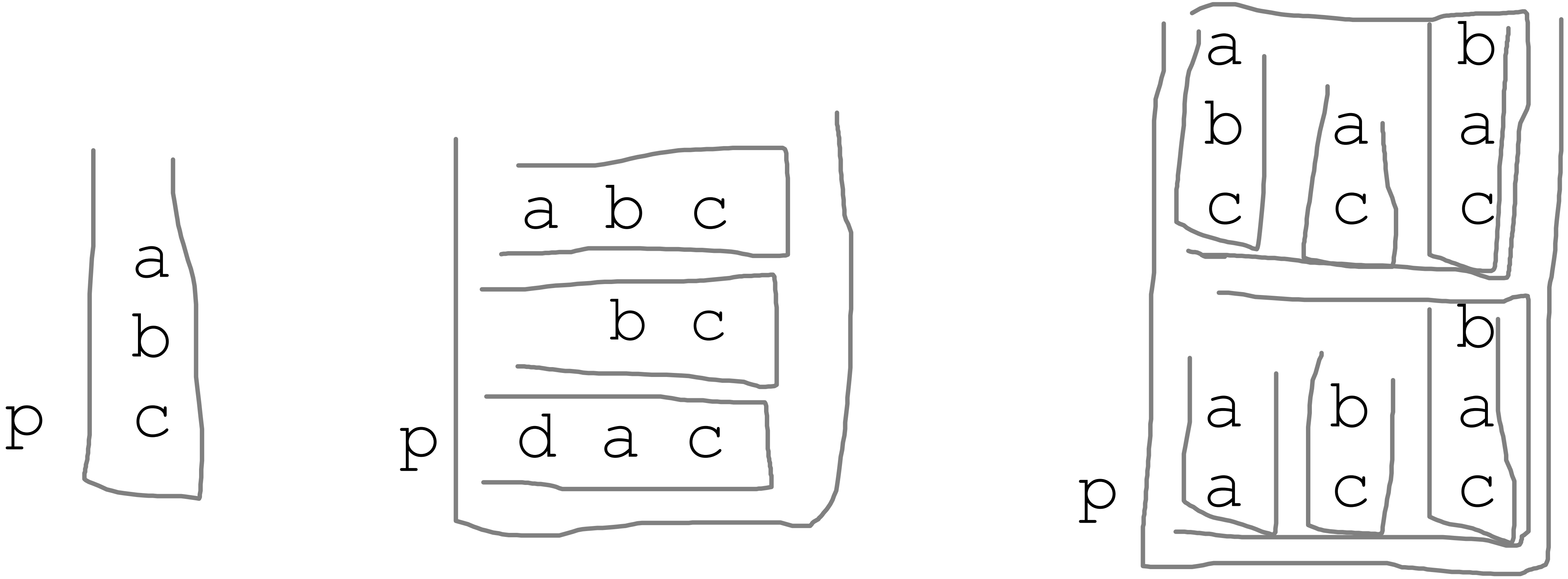




Collapsible Pushdown Systems

Higher-order program: functions of functions

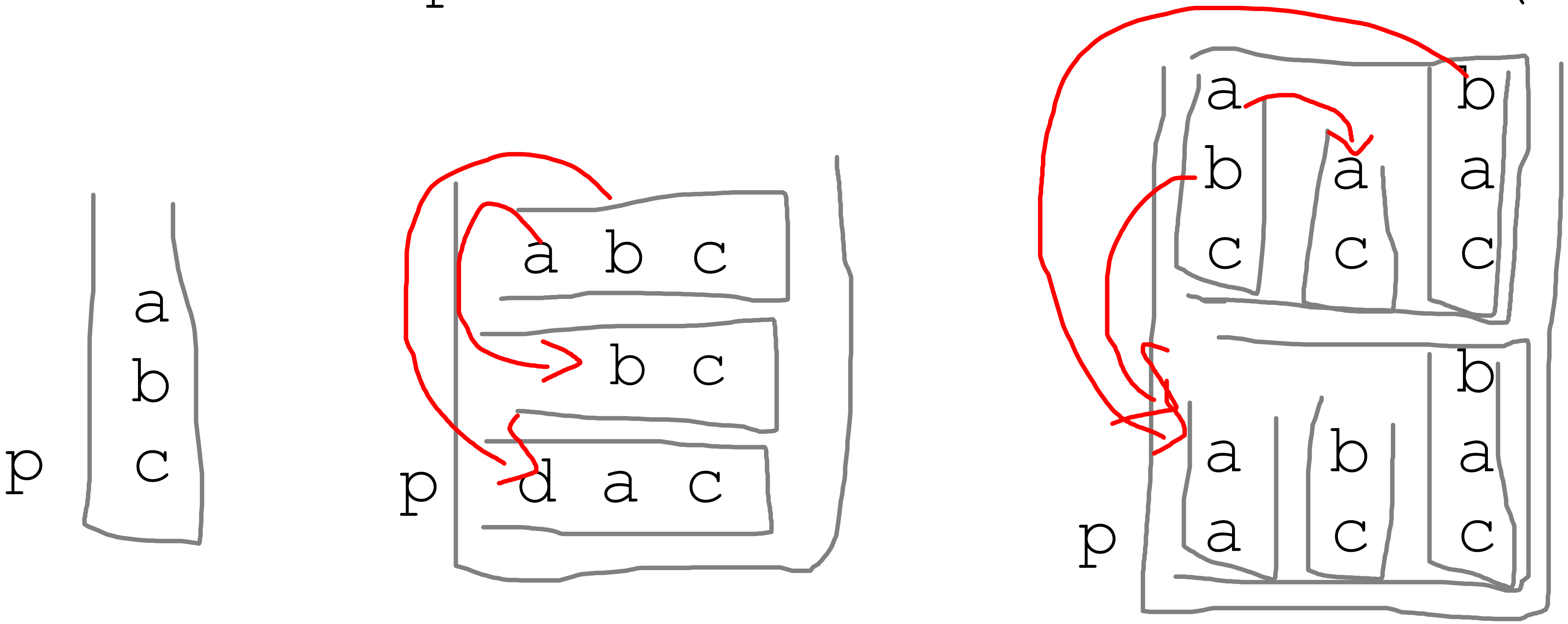
Higher-order pushdown: stack of stacks



Collapsible Pushdown Systems

Higher-order program: functions of functions

Higher-order pushdown: stack of stacks (+links)



Generalising Parity->Safety

- Collapsible Pushdown -> Finite-State (n-Exponential blow up)
- n-Exponential Counters
- Can encode in binary on order-n stack!

Summary

- Parity -> Safety
- Counters look out for loops
  (Even with infinite states)
- Polynomial-time encoding
  - Reduction to finite-state
  - Counters behave like stacks
  - Counter values match limits of system

Future and Related Work

- Can these ideas lead to implementations?
  - Direct implementation
  - Abstraction/refinement of counters
  - Counter size vs. structure of game
  - Backend: HorSat, Preface, &c.
- Parity->Safety:
  - Berwanger and Doyen, 2008
  - Sohail and Somenzi, 2009
  - Biere et al, 2002
  - Podelski and Rybalchenko, 2011
  - Konnov et al, 2017