Structural Information in Distributed Computing

David Ilcinkas

HDR defense
March 15, 2019

Distributed systems

Several computing entities cooperating in the absence of a central controller

Various examples

- Computer networks (over the Internet typically)
  Routers / P2P systems / GAFA
- Multi-core processors
- Internet of Things / Sensor networks
- Software agents
- Robots in environments not accessible by humans
- ...

Heterogeneity in size, power, communication mechanisms...

Distributed Computing

What tasks / problems can be solved by several cooperating computing entities despite some constraints:

- Different and partial perceptions of the environment
- Constrained communication
- Asynchrony
- Failures
- Dynamicity
- ...

Differences with parallelism

As usually expressed by Michel Raynal

- Parallelism: provides Efficiency
- Distributed Computing: masters Uncertainty

The multiplicity of computing entities is rather

- a choice in Parallelism
- a constraint in Distributed Computing

Differences with parallelism

As usually expressed by Michel Raynal

- Parallelism: provides Efficiency
- Distributed Computing: masters Uncertainty

The multiplicity of computing entities is rather

- a choice in Parallelism
- a constraint in Distributed Computing

Models

Huge variety of possible assumptions

- Shared memory / Message passing
- Synchronous / Asynchronous
- Anonymous / with IDs
- Reliable nodes / Crashes / Malicious nodes
- Messages: reliable / omission / duplication / reordering / alteration
- ...

Two main points of view

From Pierre Fraigniaud's PODC 2010 invited talk

Time-oriented Dist. Comp.

- Asynchrony
- Failures (typically crashes)
- Direct communication (usually Shared memory)
- Usual goal: Computability

Space-oriented Dist. Comp.

- Communication network (usually Message passing)
- No failures
- Usual goal: efficiency

Distributed computing landscape

Message adversaries

Dynamic networks

Asynchronous
Fault-prone
Direct comm

Comm. network
Fault-free

Self-stabilization

Mobile agents

Distributed computing landscape

Message adversaries

Dynamic networks

Asynchronous
Fault-prone
Direct comm

Comm. network
Fault-free

My work

Self-stabilization

Mobile agents

Outline

My contributions

Usual focus

Impact of the information available to the computing entities

- A priori global knowledge
- Knowledge about the future in dynamic networks
- Information from various sensors
- Space complexity

Supervision

- Evangelos Bampas (postdoc) on mobile agents
- Christian Glacet (PhD) on routing
- Ahmed Wade (PhD) on dynamic networks
- Antonin Lentz (PhD) on multi-criteria shortest-paths

Moving and Computing

The computing entities are mobile (controlled mobility)

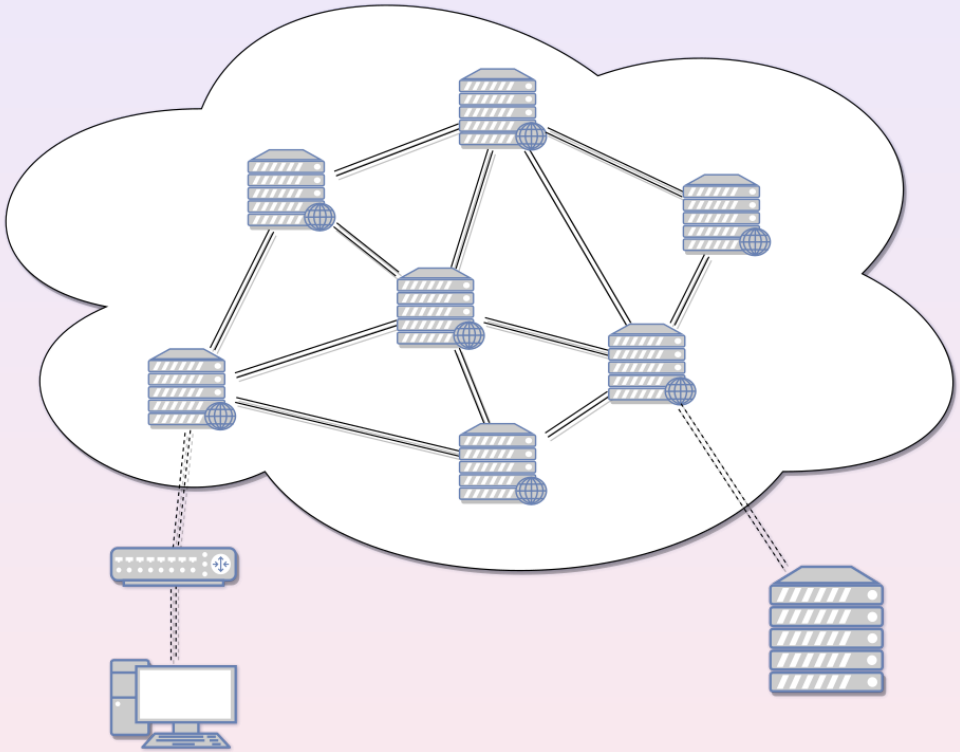
Agents

- Operating in a network
- Can move from node to node through the edges
- Local view (just the current node)

Robots

- Global view of the environment
- Operating in a generally continuous environment

On routing



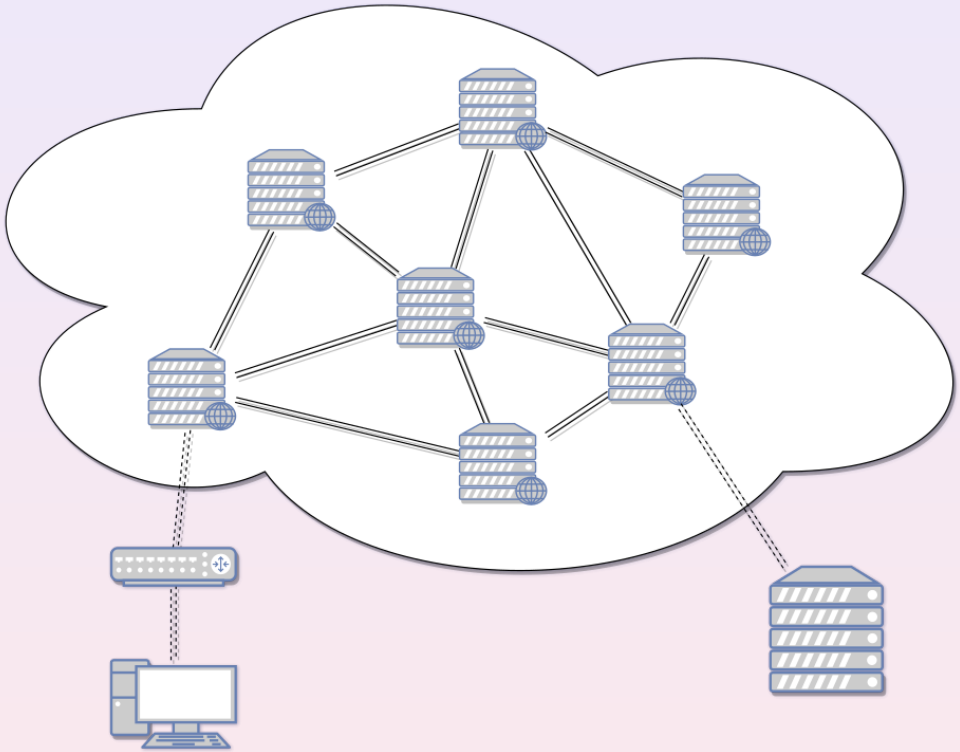
CC-BY-SA-4.0 Wikipedia Tomybrz

Routing scheme

- Routing tables
- Message/packet headers
- Routing algorithm

Issue: routing tables grow faster than the memory

On routing



CC-BY-SA-4.0 Wikipedia Tomybrz

Routing scheme

- Routing tables
- Message/packet headers
- Routing algorithm

Issue: routing tables grow faster than the memory

One result about routing
(cf. Christian Glacet's PhD thesis)

[Gavoille, Glacet, Hanusse, I. DISC 2013]

Distributed routing scheme for weighted n-node networks

- Asynchronous algorithm (and optimal synchronous time)
- Compact routing tables: $\tilde{O}(\sqrt{n})$ bits
- In "small-world" networks: $\tilde{O}(n^{3/2})$ messages
- Stretch ≤ 5 (multiplicative overhead on the path length)

Although shortest-path routing implies

- Routing tables of $\tilde{\Omega}(n)$ bits
- $\tilde{\Omega}(n^2)$ messages (even in "small-world" networks)

Context on dynamic networks

Distributed computing systems are becoming more and more dynamic.

Causes

- Frequent connections / disconnections to/in the network
- Fault or crash of communicating objects and links
- Uncontrolled mobility of the computing entities

Classical models: Static fault-tolerant networks

- Assume that the frequency of fault occurrences is small
- Assume that the number of fault occurrences is small

In fact, these models become insufficient for very dynamic networks.

Context on dynamic networks

Distributed computing systems are becoming more and more dynamic.

Causes

- Frequent connections / disconnections to/in the network
- Fault or crash of communicating objects and links
- Uncontrolled mobility of the computing entities

Classical models: Static fault-tolerant networks

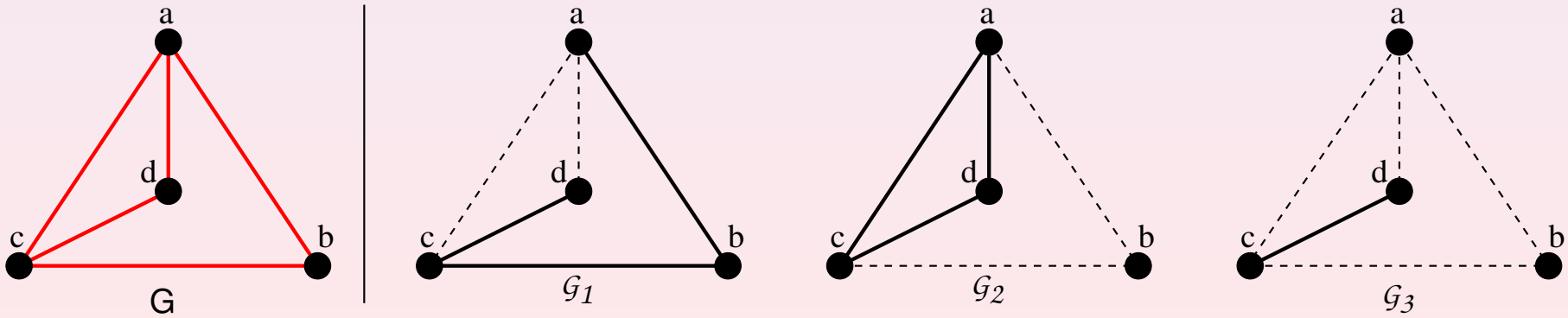
- Assume that the frequency of fault occurrences is small
- Assume that the number of fault occurrences is small

In fact, these models become insufficient for very dynamic networks.

The model we consider

Evolving graphs

- One of the first developed models, and also one of the most standard.
- An evolving graph is a (possibly infinite) sequence $\mathcal{G}_1$, $\mathcal{G}_2$, $\mathcal{G}_3$, ... of static graphs based on the same set of nodes.
- We focus here on constantly connected evolving graphs



Some results about dynamic networks
(cf. Ahmed Wade's PhD thesis)

Exploration of constantly connected evolving graphs with known dynamics

Known results

- Temporal diameter is at most $n - 1$
- $\Longrightarrow$ Exploration time is at most $(n-1)^2$
- Some m-edge base graphs require exploration time $\Omega(m)$

[I., Wade, ToCS, 2018] & [I., Klasing, Wade, SIROCCO 2014]

- Ring: optimal worst-case time $2n - 3$
- Ring: exploration time decreases when stability increases
- Cactus (tree of rings): algorithm with sub-polynomial overhead

Some results about dynamic networks
(cf. Ahmed Wade's PhD thesis)

Exploration of constantly connected evolving graphs with known dynamics

Known results

- Temporal diameter is at most $n-1$
- $\Longrightarrow$ Exploration time is at most $(n-1)^2$
- Some m-edge base graphs require exploration time $\Omega(m)$

[I., Wade. ToCS, 2018] & [I., Klasing, Wade. SIROCCO 2014]

- Ring: optimal worst-case time $2n-3$
- Ring: exploration time decreases when stability increases
- Cactus (tree of rings): algorithm with sub-polynomial overhead

Outline

Problem

Black hole in a network

Node blocking and destroying any mobile agent entering it

Motivations:

- Site which is destroyed or dangerous for a physical robot
- Node infected by a "killer" virus
- Mute or crashed node

Black Hole Search

From a safe node (homebase), a team of mobile agents has to locate the black hole(s)

Performance: nb of agents, nb of moves

Problem

Black hole in a network

Node blocking and destroying any mobile agent entering it

Motivations:

- Site which is destroyed or dangerous for a physical robot
- Node infected by a "killer" virus
- Mute or crashed node

Black Hole Search

From a safe node (homebase), a team of mobile agents has to locate the black hole(s)

Performance: nb of agents, nb of moves

(A)synchronous

Synchronous

Agents act simultaneously at every pulse of a global clock

$\Longrightarrow$ Possible to wait for an agent gone exploring a potentially dangerous edge

Asynchronous (our case)

Every action takes a finite but unbounded time

$\Longrightarrow$ A priori impossible to distinguish a very slow link from a link leading to the black hole

$\Longrightarrow$ Useless to wait for an agent

$\Longrightarrow$ Pb equivalent to the exploration of dangerous graphs

(A)synchronous

Synchronous

Agents act simultaneously at every pulse of a global clock

$\Longrightarrow$ Possible to wait for an agent gone exploring a potentially dangerous edge

Asynchronous (our case)

Every action takes a finite but unbounded time

$\Longrightarrow$ A priori impossible to distinguish a very slow link from a link leading to the black hole

$\Longrightarrow$ Useless to wait for an agent

$\Longrightarrow$ Pb equivalent to the exploration of dangerous graphs

Indirect communication models

Communication/coordination

No direct communication but

Whiteboards (memory space on each node).

or Pointers (mark a specific port).

or Pebbles (node markers, which can be carried by agents).

Related work

Seminal results (asynchronous, whiteboards)

[Dobrev, Flocchini, Prencipe, Santoro. Dist. Comp. 2006]

	nb of agents	nb of moves
No information (besides n)	$\Delta + 1$	$\Theta(n^2)$
Sense of direction	2	$\Theta(n^2)$
Known map	2	$\Theta(n \log n)$

Pointers, ring, FIFO links

[Dobrev, Santoro, Shi. CIAC 2006]

- $O(1)$ pointers in total, up to 3 per node
- Nb of moves : $\Theta(n \log n)$

Pb : Is coordination through pebbles sufficient ?

Related work

Seminal results (asynchronous, whiteboards)

[Dobrev, Flocchini, Prencipe, Santoro. Dist. Comp. 2006]

	nb of agents	nb of moves
No information (besides n)	$\Delta + 1$	$\Theta(n^2)$
Sense of direction	2	$\Theta(n^2)$
Known map	2	$\Theta(n \log n)$

Pointers, ring, FIFO links

[Dobrev, Santoro, Shi. CIAC 2006]

- $O(1)$ pointers in total, up to 3 per node
- Nb of moves : $\Theta(n \log n)$

Pb : Is coordination through pebbles sufficient ?

Related work

Seminal results (asynchronous, whiteboards)

[Dobrev, Flocchini, Prencipe, Santoro. Dist. Comp. 2006]

	nb of agents	nb of moves
No information (besides n)	$\Delta + 1$	$\Theta(n^2)$
Sense of direction	2	$\Theta(n^2)$
Known map	2	$\Theta(n \log n)$

Pointers, ring, FIFO links

[Dobrev, Santoro, Shi. CIAC 2006]

- $O(1)$ pointers in total, up to 3 per node
- Nb of moves : $\Theta(n \log n)$

Pb : Is coordination through pebbles sufficient ?

Our results

Model

- Arbitrary graphs (known map)
- Links do not need to be FIFO

Optimal result

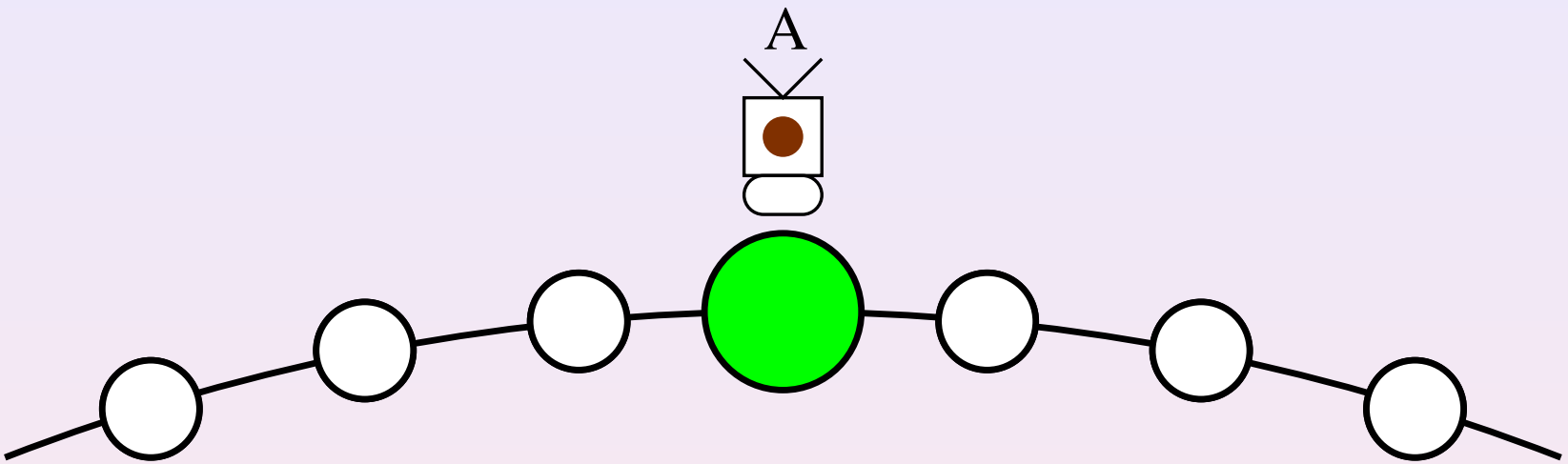
- 1 pebble per agent
- 2 agents
- $\Theta(n \log n)$ moves

Case of the ring

Model

- Two identical agents equipped with one pebble each
- Indistinguishable pebbles
- At most one pebble per node / carried by an agent
- Asynchronous ring with exactly one black hole

First steps

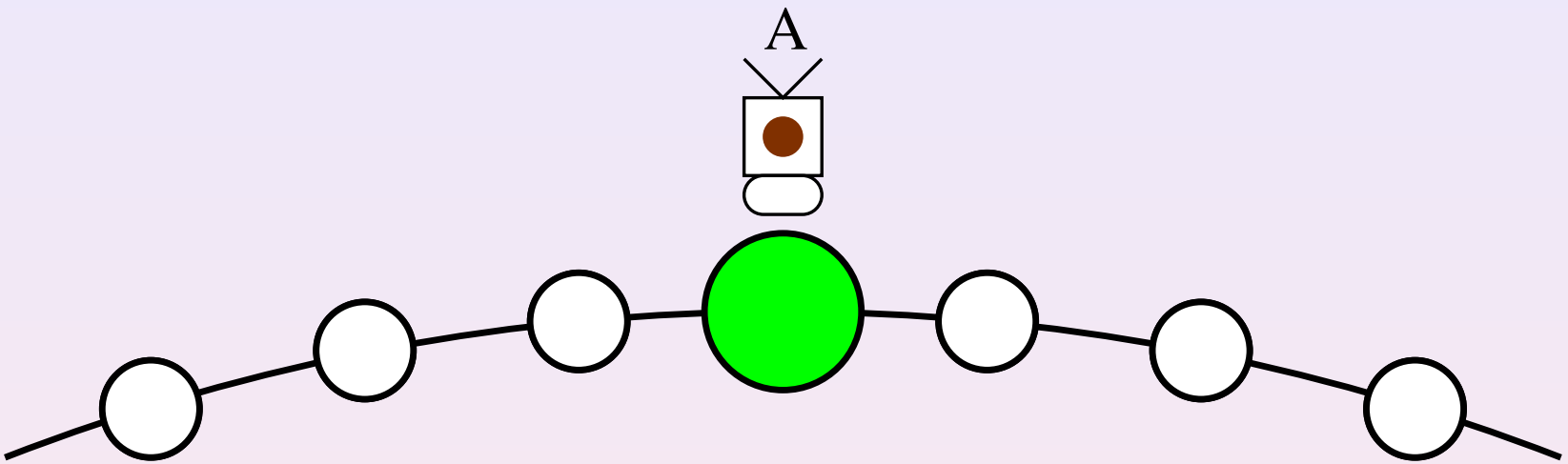


Beginning of the algorithm

If node is empty **then**

Go right

First steps



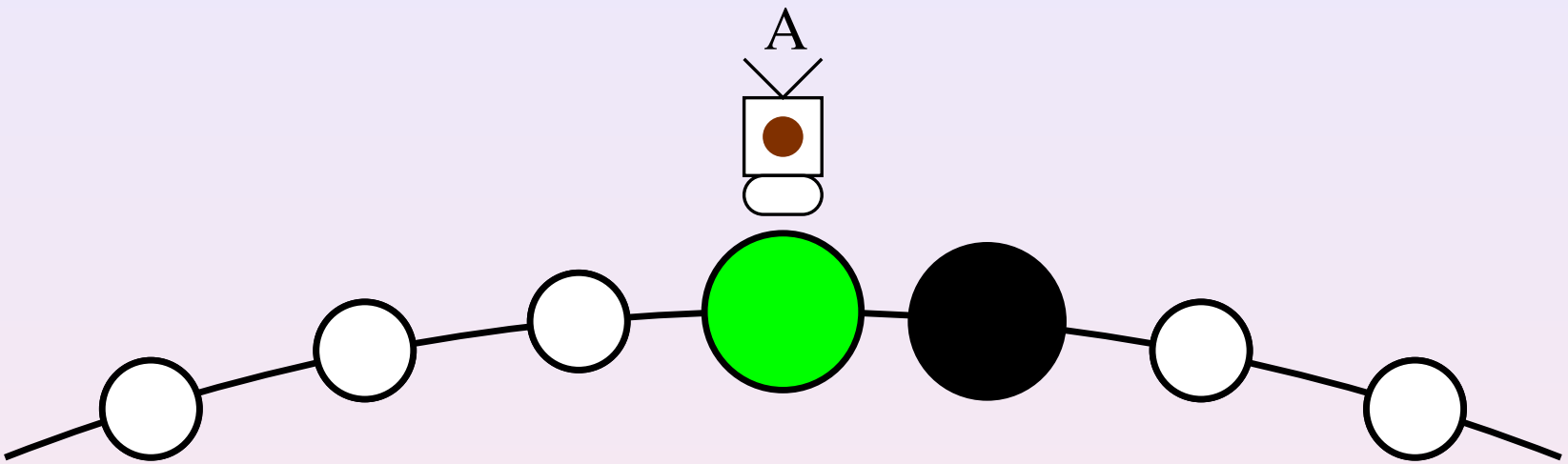
Beginning of the algorithm

If node is empty **then**

Do not put down the pebble

Go right

First steps



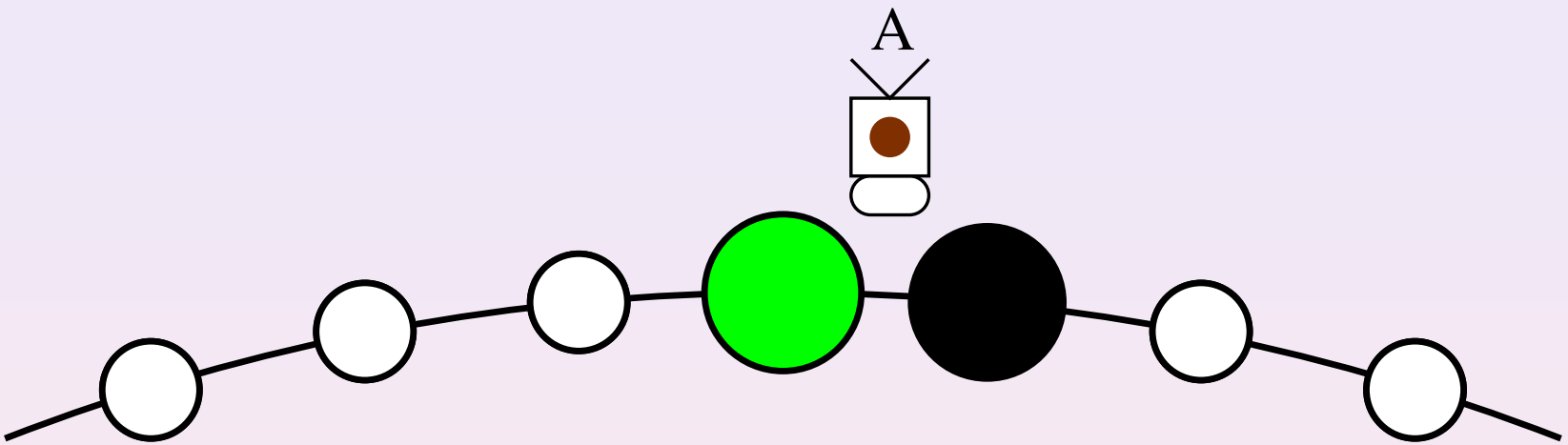
Beginning of the algorithm

If node is empty **then**

Do not put down the pebble

Go right

First steps



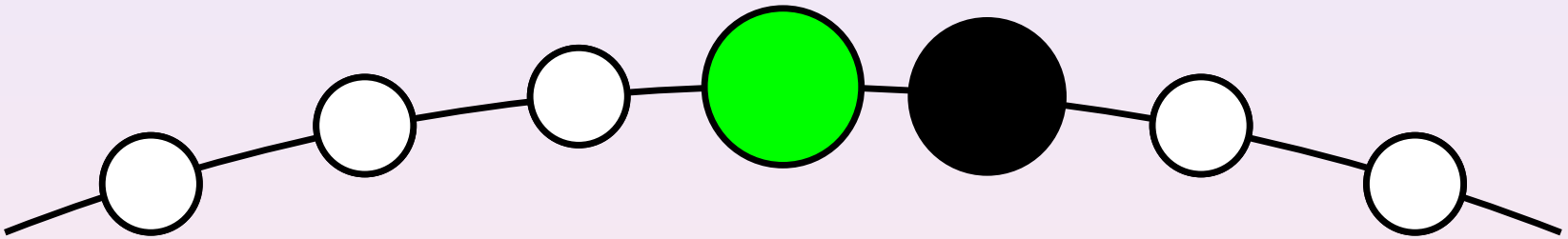
Beginning of the algorithm

If node is empty **then**

Do not put down the pebble

Go right

First steps



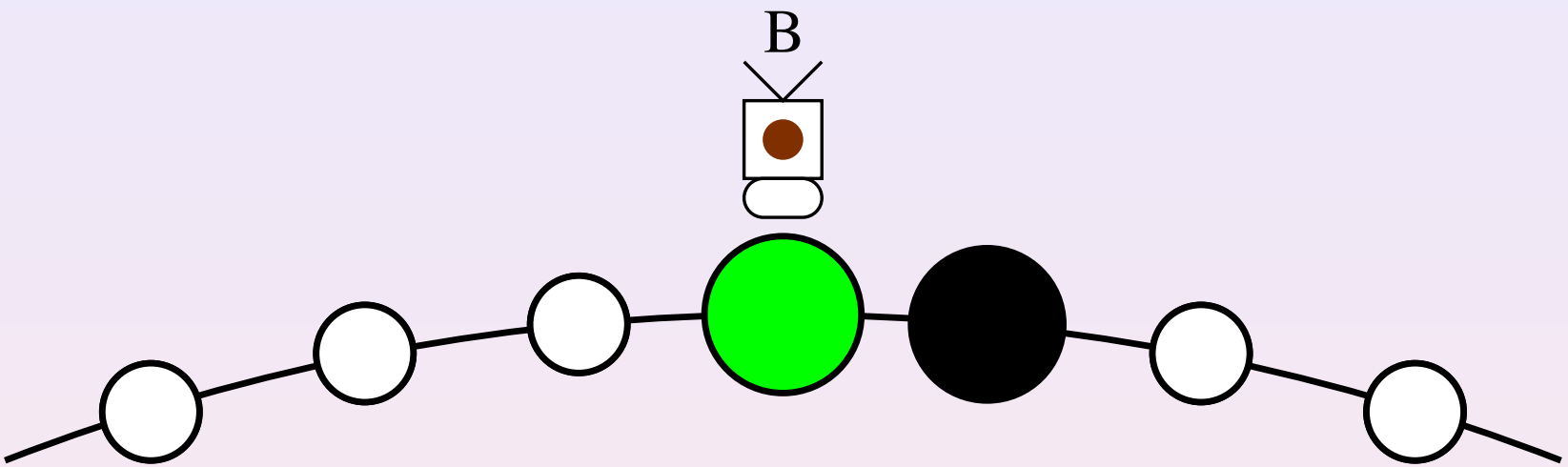
Beginning of the algorithm

If node is empty **then**

Do not put down the pebble

Go right

First steps



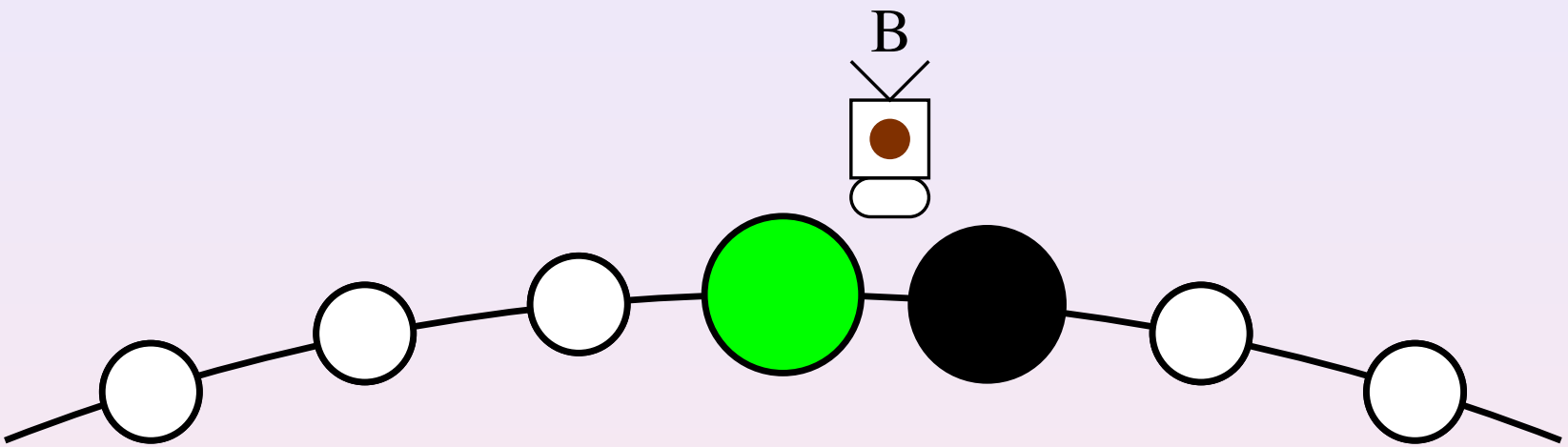
Beginning of the algorithm

If node is empty **then**

Do not put down the pebble

Go right

First steps



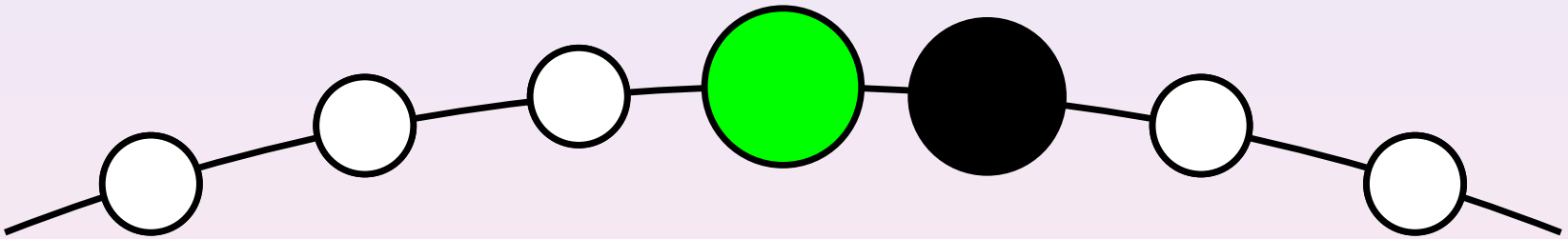
Beginning of the algorithm

If node is empty **then**

Do not put down the pebble

Go right

First steps



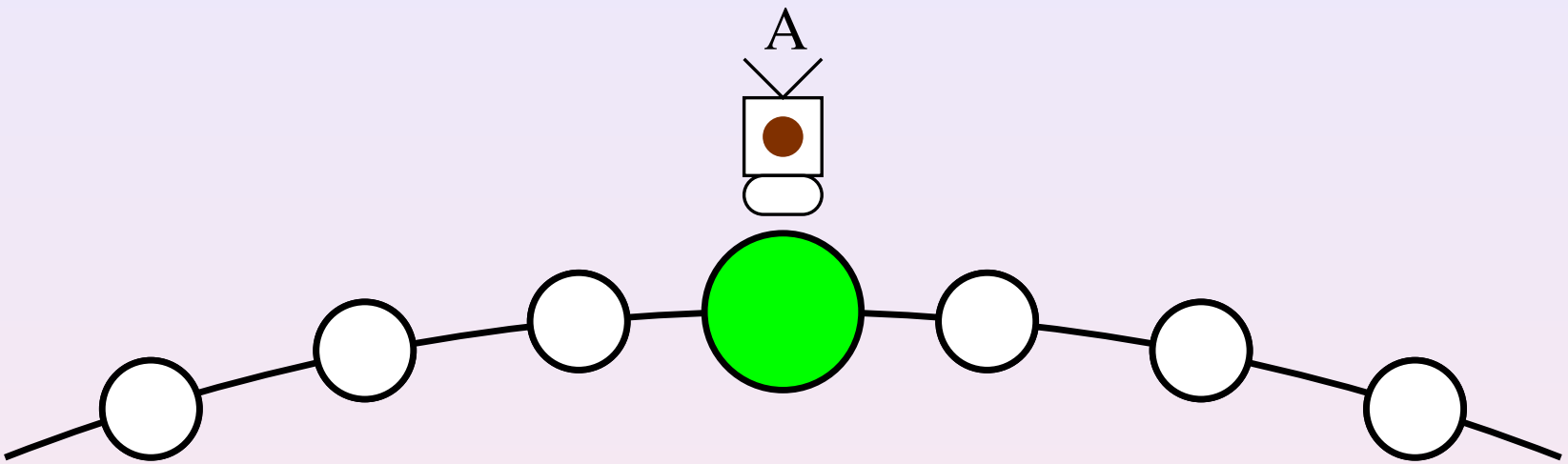
Beginning of the algorithm

If node is empty **then**

Do not put down the pebble

Go right

First steps



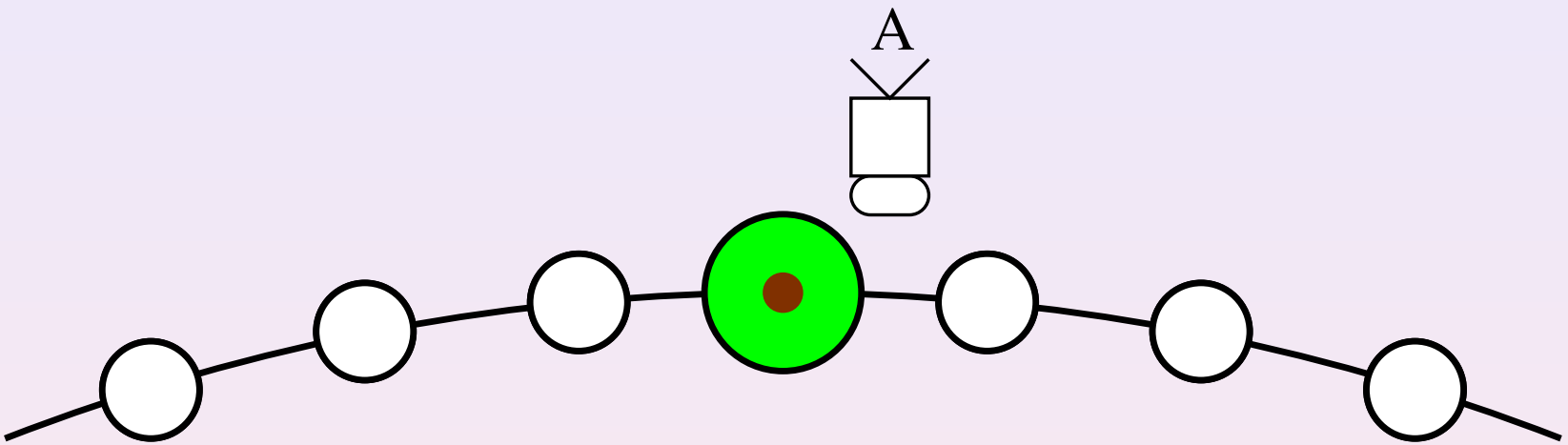
Beginning of the algorithm

If node is empty **then**

Put down the pebble

Go right

First steps



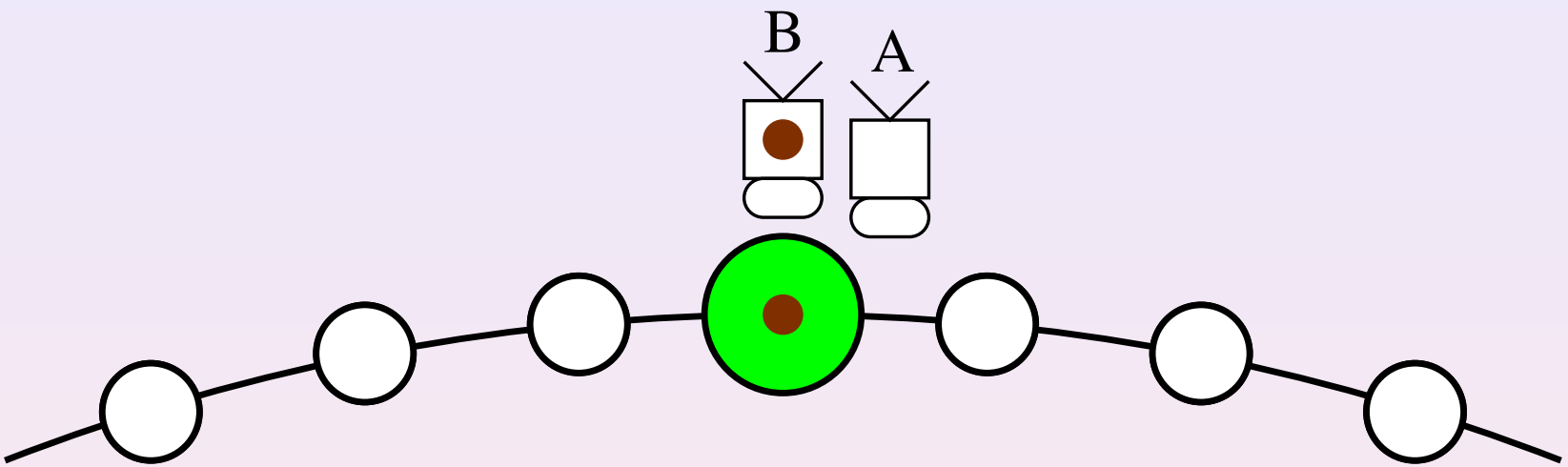
Beginning of the algorithm

If node is empty **then**

Put down the pebble

Go right

First steps



Beginning of the algorithm

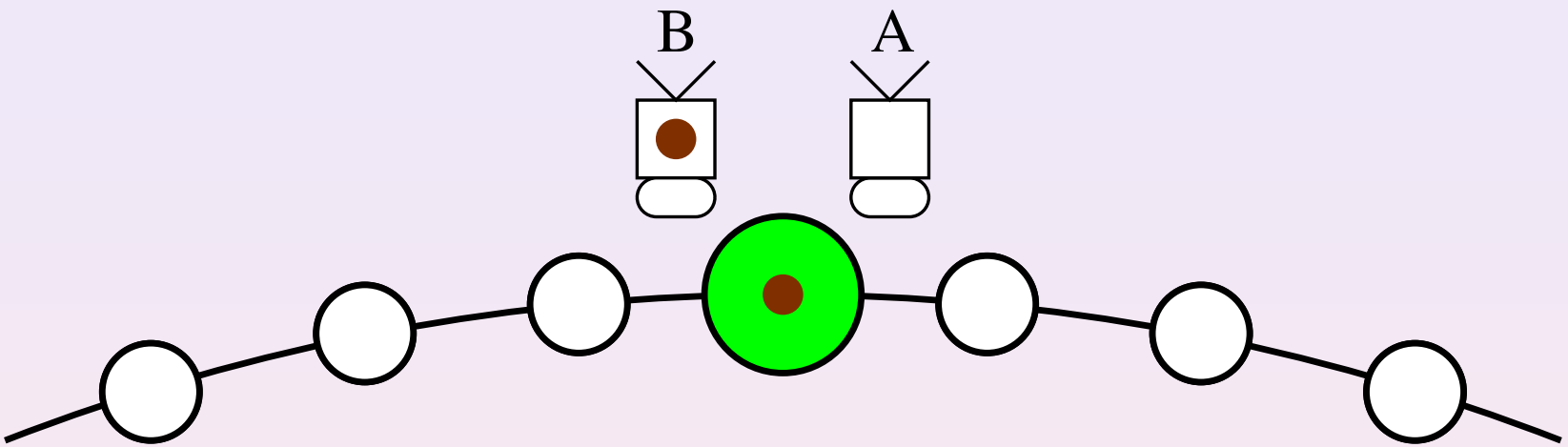
If node is empty **then**

    Put down the pebble

    Go right

else

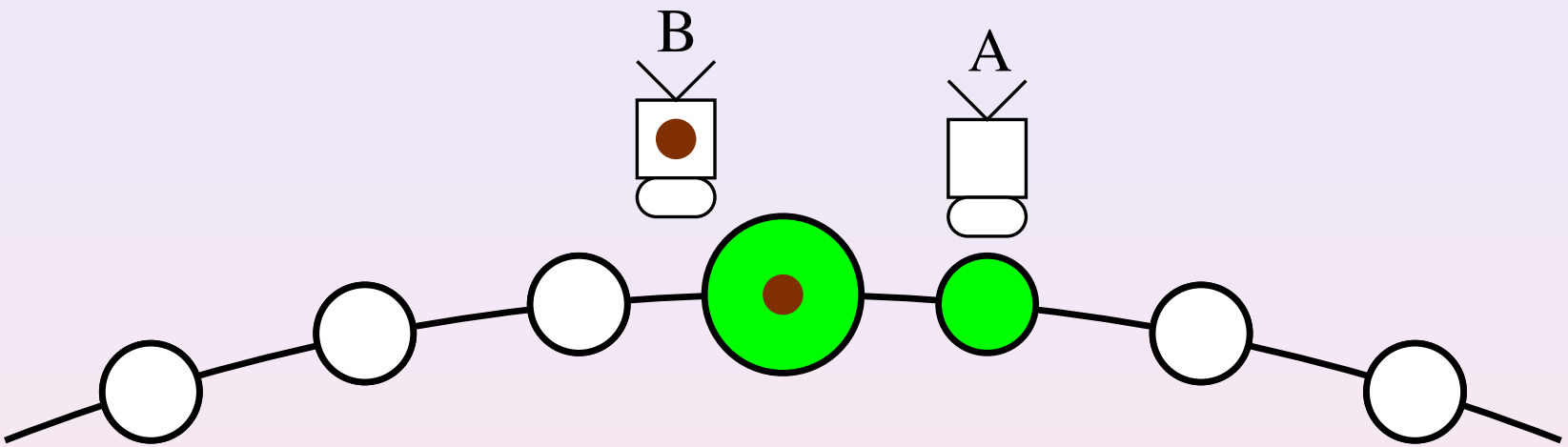
First steps



Beginning of the algorithm

If node is empty **then**
    Put down the pebble
    Go right
else Go left **endif**

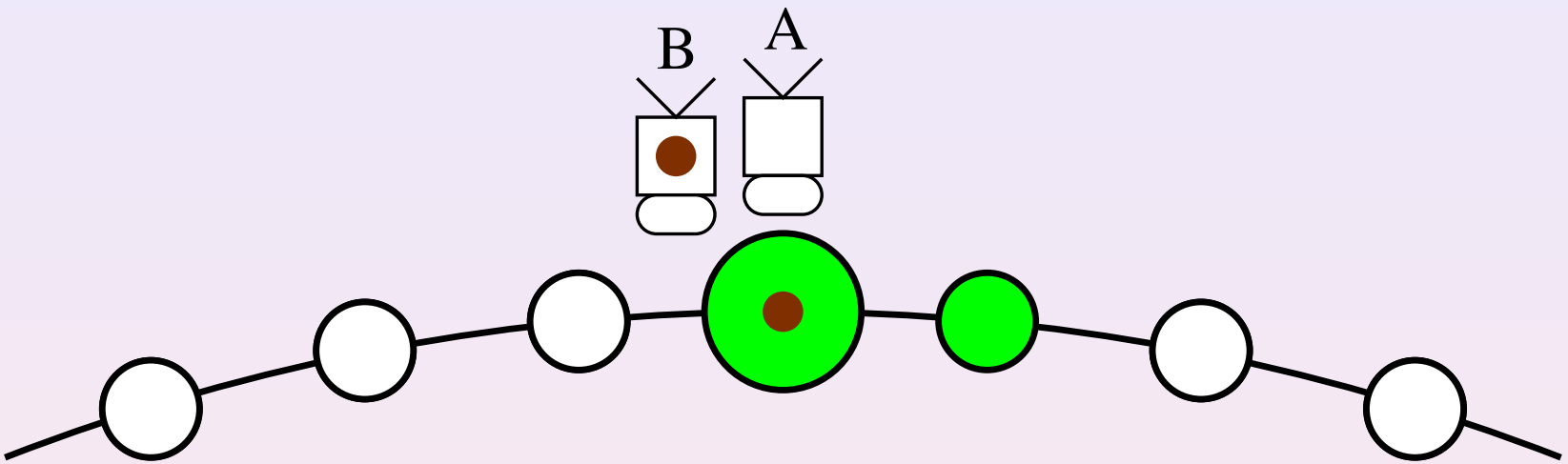
First steps



Beginning of the algorithm

If node is empty **then**
    Put down the pebble
    Go right
else Go left **endif**

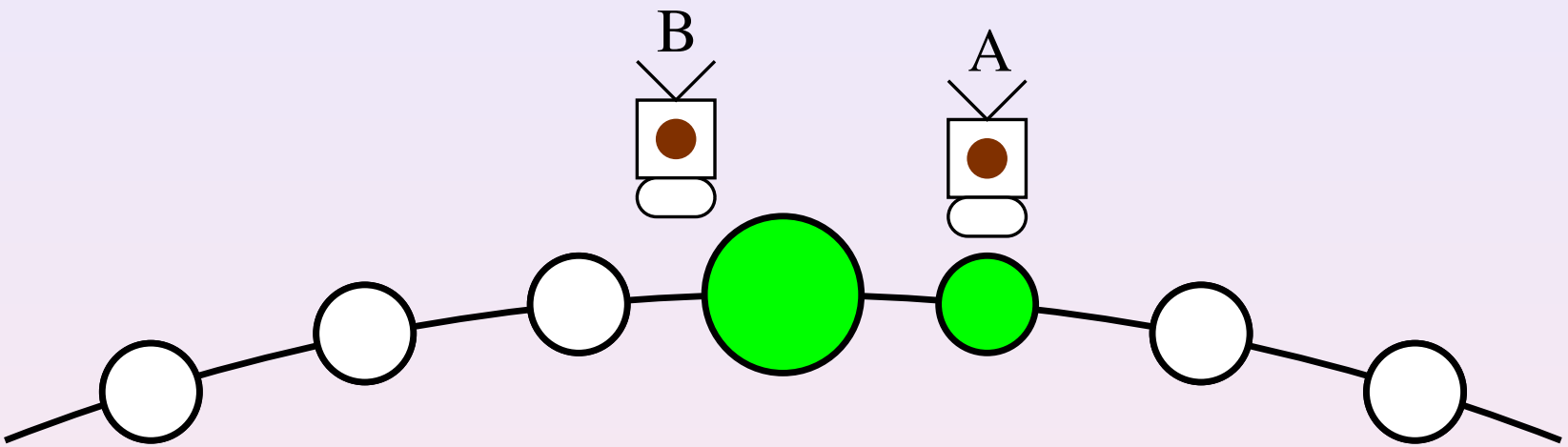
First steps



Beginning of the algorithm

If node is empty **then**
    Put down the pebble
    Go right
else Go left **endif**

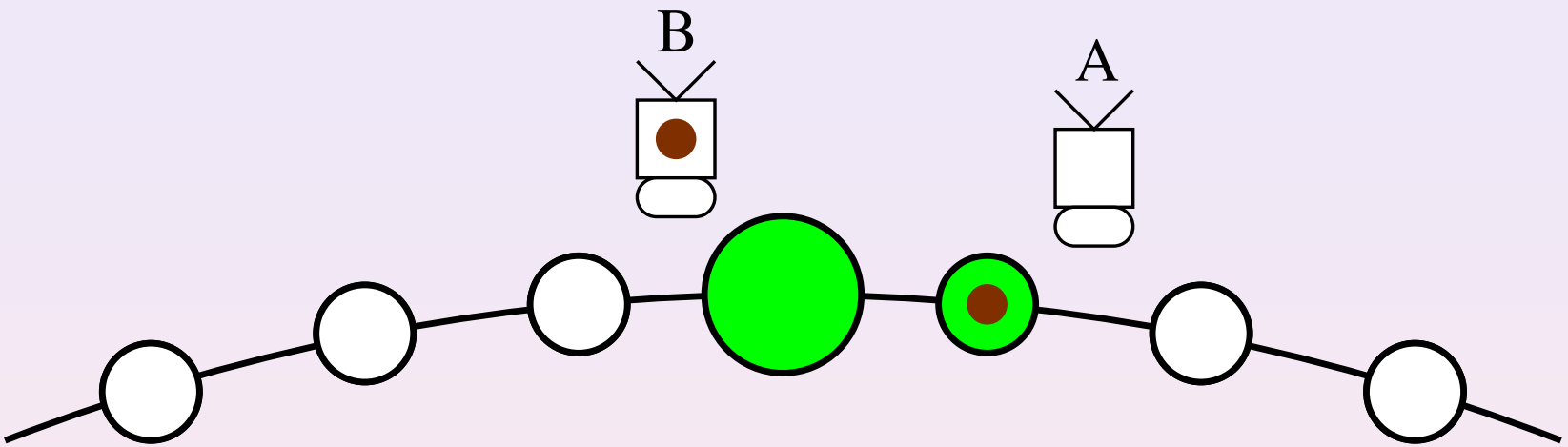
First steps



Beginning of the algorithm

If node is empty **then**
    Put down the pebble
    Go right
else Go left **endif**

First steps



Beginning of the algorithm

If node is empty **then**
    Put down the pebble
    Go right
else Go left **endif**

Cautious walk

Exploration of $e = \{u, v\}$ from u to v

Simple cautious walk

Initially: agent at u with a pebble, node without a pebble

- Put down the pebble (warn the other agent) and go to v
- Come back to u
- Retrieve the pebble and return at v

Double cautious walk

Initially: agent at u with a pebble, node u with a pebble

- Go to v
- Put down the pebble and go back to u
- Retrieve the other pebble and return at v

Cautious walk

Exploration of $e = \{u, v\}$ from u to v

Simple cautious walk

Initially: agent at u with a pebble, node without a pebble

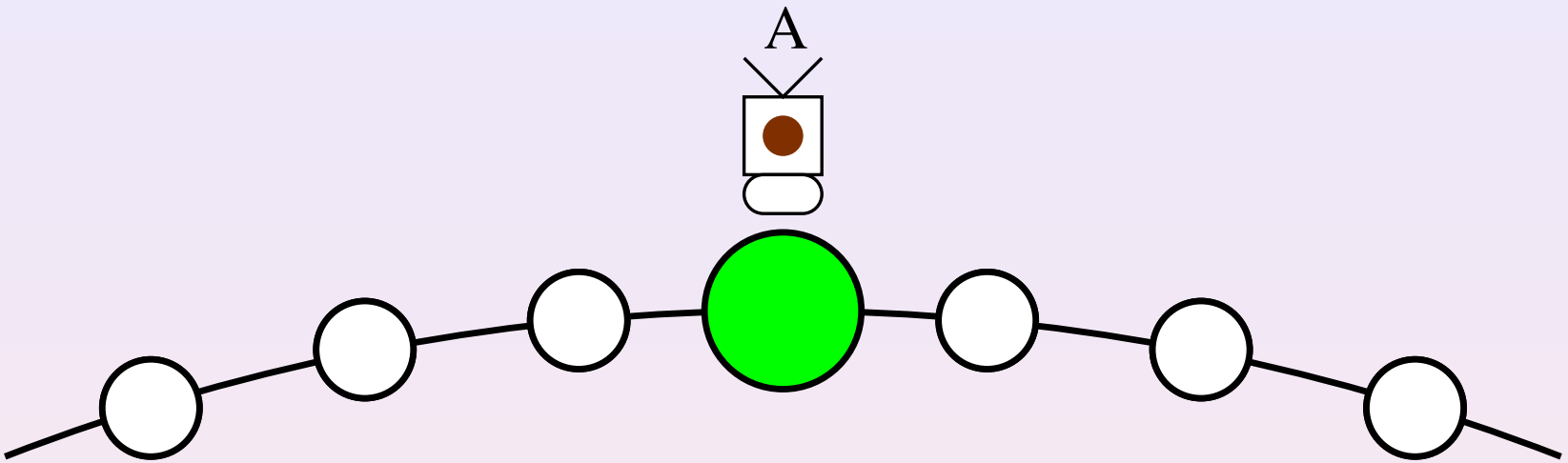
- Put down the pebble (warn the other agent) and go to v
- Come back to u
- Retrieve the pebble and return at v

Double cautious walk

Initially: agent at u with a pebble, node u with a pebble

- Go to v
- Put down the pebble and go back to u
- Retrieve the other pebble and return at v

Ping Pong algorithm

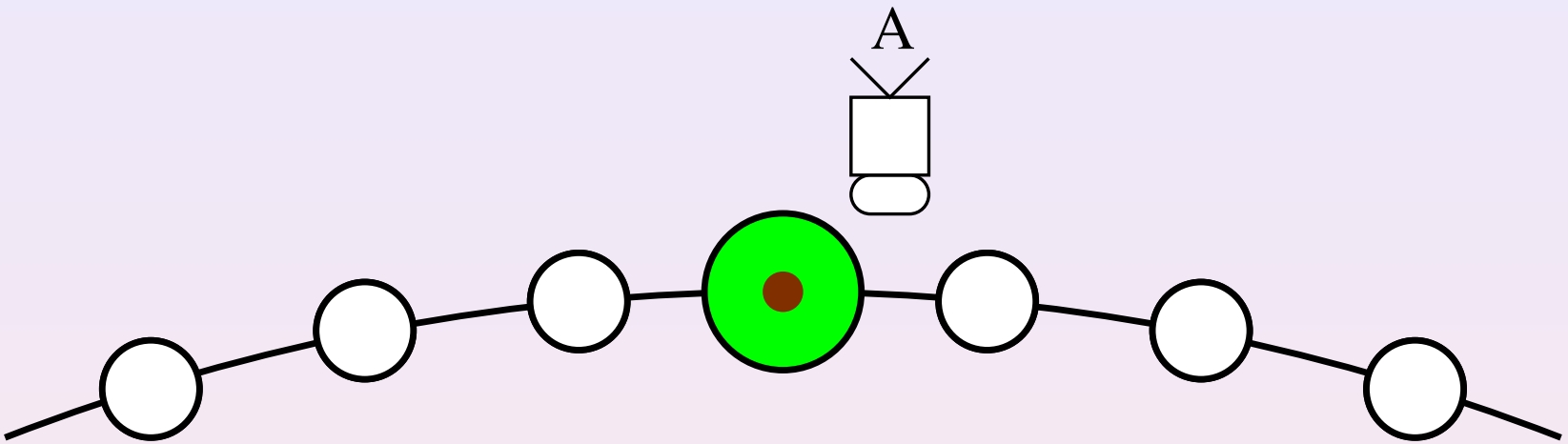


Quick description

If the agent "controls"

- 1 pebble → Simple cautious walk to the right
- 2 pebbles → Double cautious walk to the left
- 0 pebbles → Non-cautious walk to the left

Ping Pong algorithm

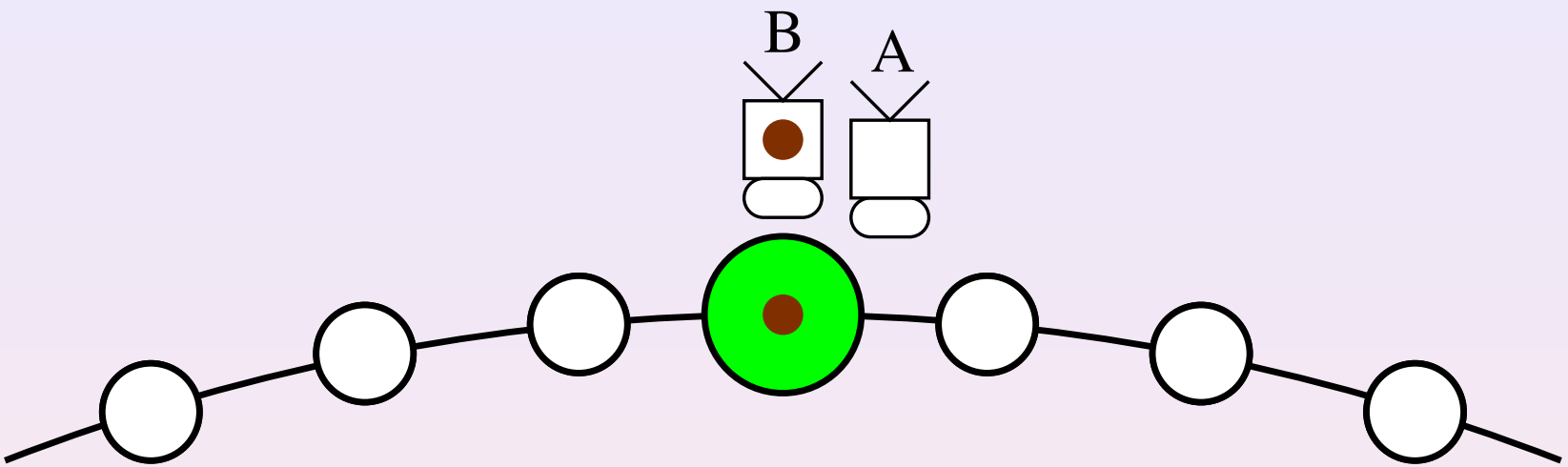


Quick description

If the agent "controls"

- 1 pebble → Simple cautious walk to the right
- 2 pebbles → Double cautious walk to the left
- 0 pebbles → Non-cautious walk to the left

Ping Pong algorithm

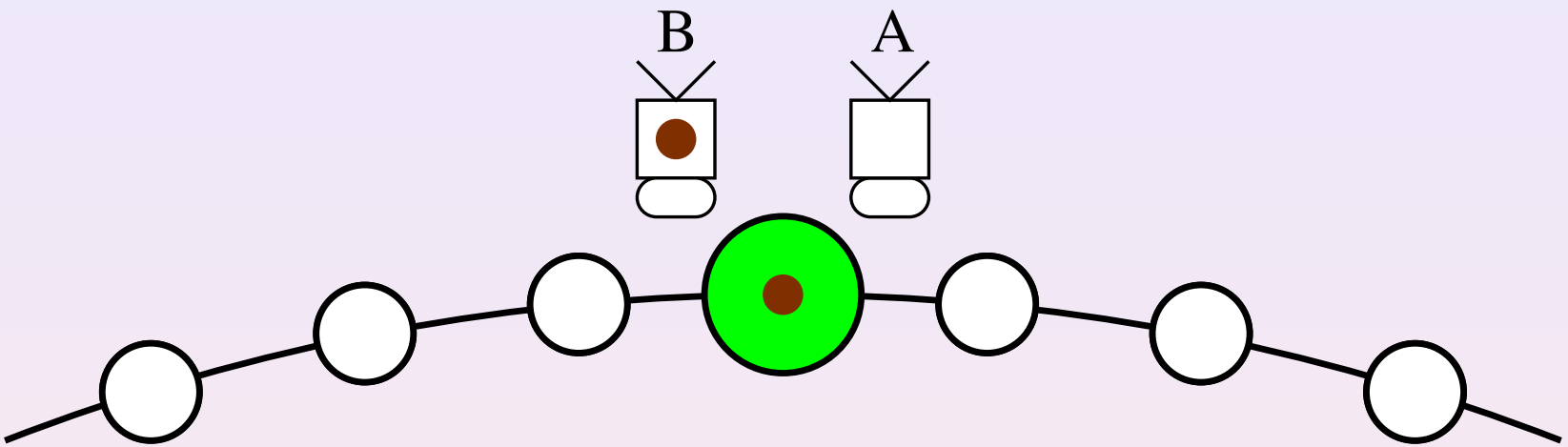


Quick description

If the agent "controls"

- 1 pebble → Simple cautious walk to the right
- 2 pebbles → Double cautious walk to the left
- 0 pebbles → Non-cautious walk to the left

Ping Pong algorithm

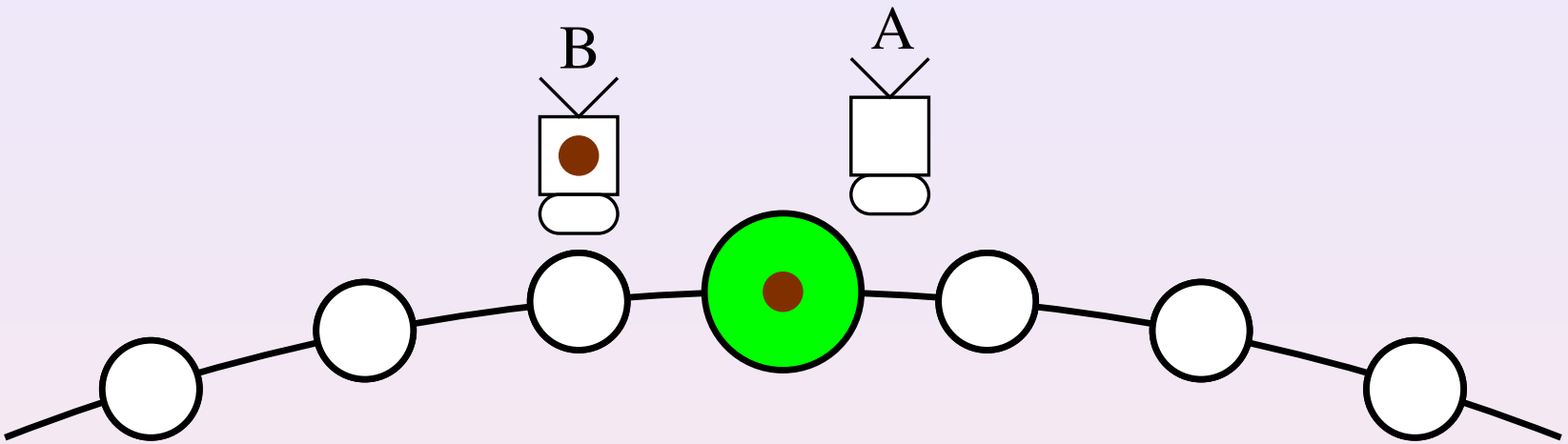


Quick description

If the agent "controls"

- 1 pebble → Simple cautious walk to the right
- 2 pebbles → Double cautious walk to the left
- 0 pebbles → Non-cautious walk to the left

Ping Pong algorithm

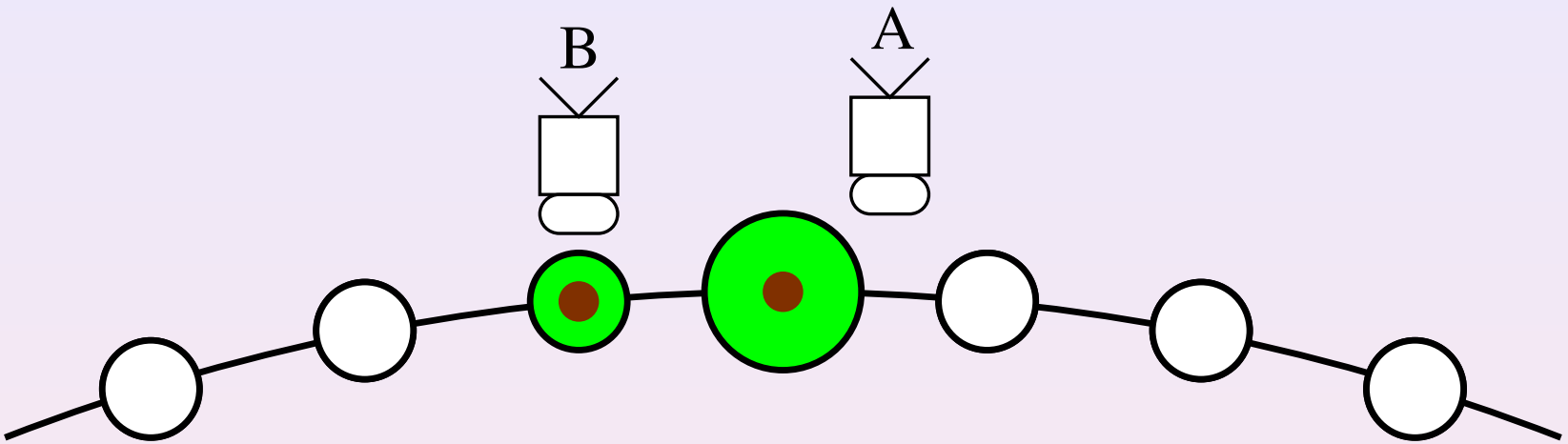


Quick description

If the agent "controls"

- 1 pebble → Simple cautious walk to the right
- 2 pebbles → Double cautious walk to the left
- 0 pebbles → Non-cautious walk to the left

Ping Pong algorithm

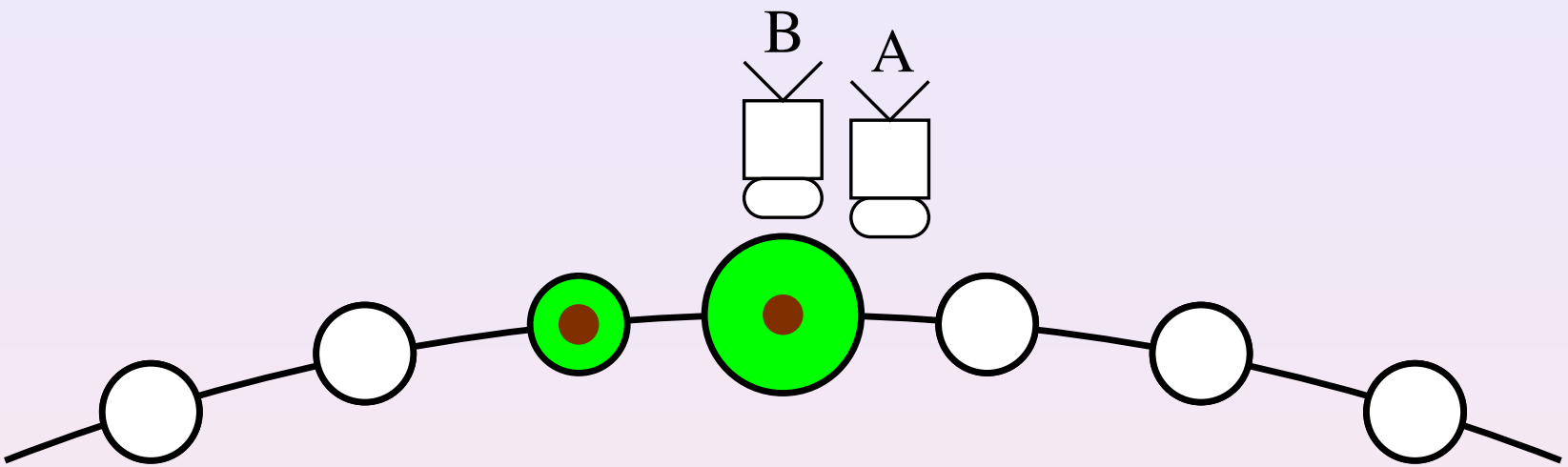


Quick description

If the agent "controls"

- 1 pebble → Simple cautious walk to the right
- 2 pebbles → Double cautious walk to the left
- 0 pebbles → Non-cautious walk to the left

Ping Pong algorithm

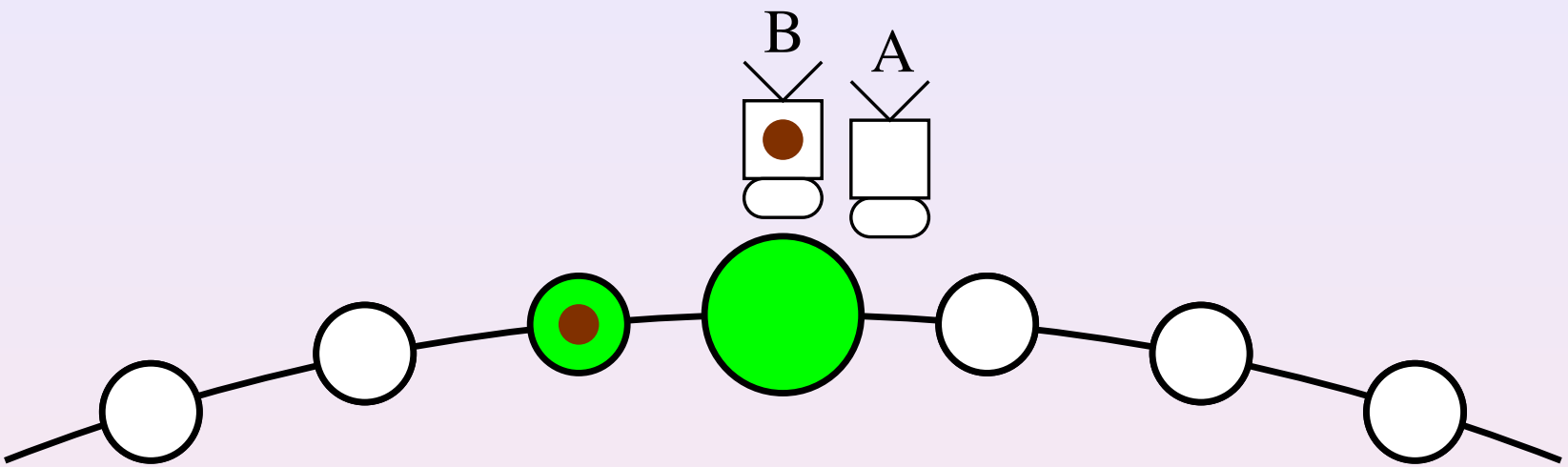


Quick description

If the agent "controls"

- 1 pebble → Simple cautious walk to the right
- 2 pebbles → Double cautious walk to the left
- 0 pebbles → Non-cautious walk to the left

Ping Pong algorithm

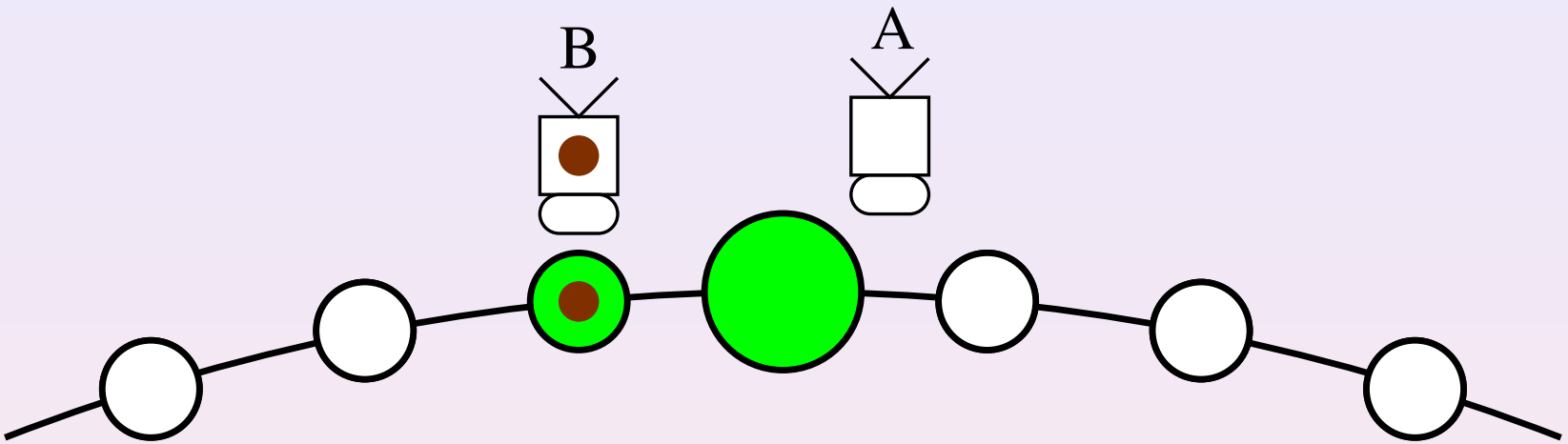


Quick description

If the agent "controls"

- 1 pebble → Simple cautious walk to the right
- 2 pebbles → Double cautious walk to the left
- 0 pebbles → Non-cautious walk to the left

Ping Pong algorithm

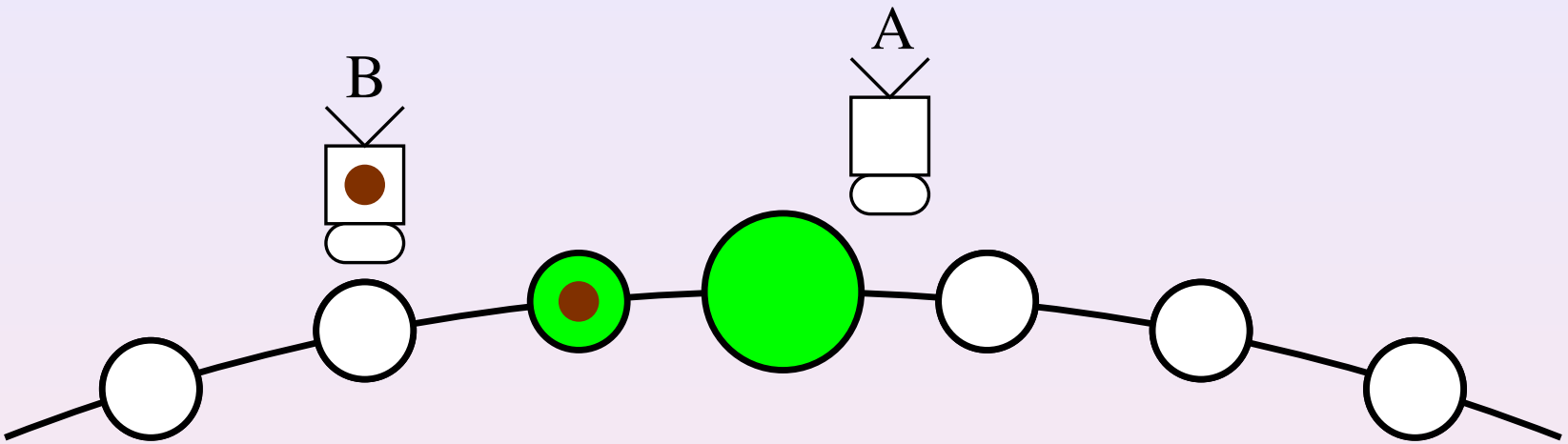


Quick description

If the agent "controls"

- 1 pebble → Simple cautious walk to the right
- 2 pebbles → Double cautious walk to the left
- 0 pebbles → Non-cautious walk to the left

Ping Pong algorithm

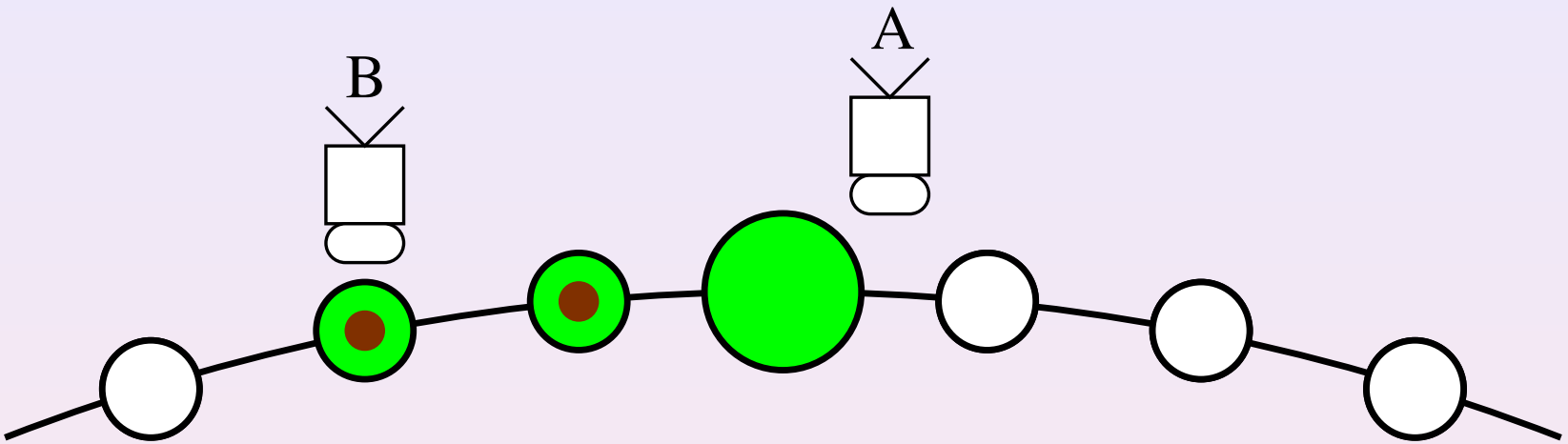


Quick description

If the agent "controls"

- 1 pebble → Simple cautious walk to the right
- 2 pebbles → Double cautious walk to the left
- 0 pebbles → Non-cautious walk to the left

Ping Pong algorithm

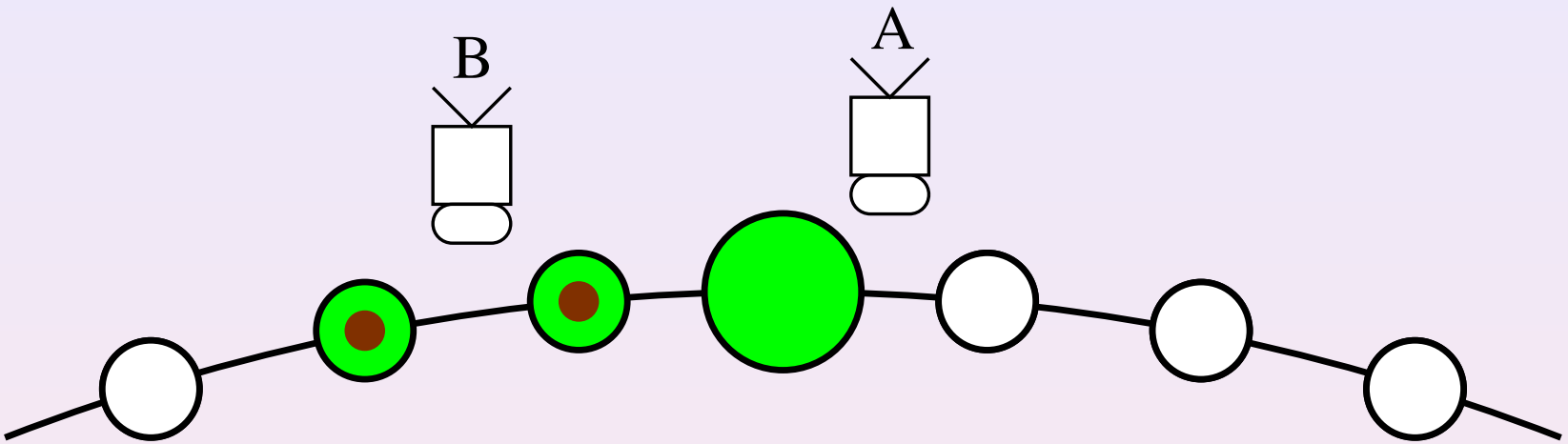


Quick description

If the agent "controls"

- 1 pebble → Simple cautious walk to the right
- 2 pebbles → Double cautious walk to the left
- 0 pebbles → Non-cautious walk to the left

Ping Pong algorithm

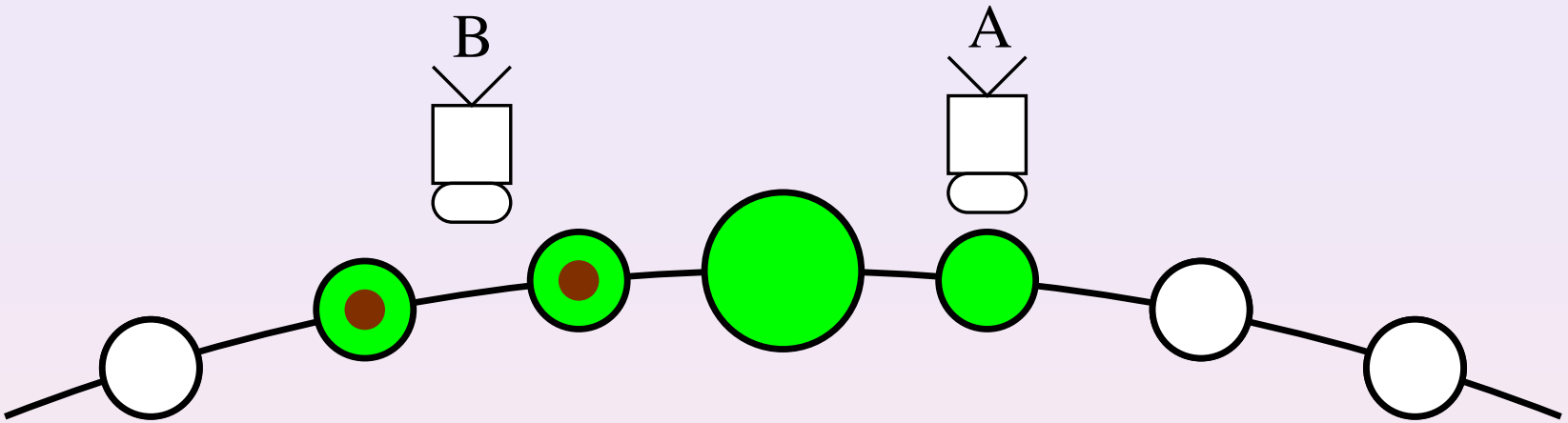


Quick description

If the agent "controls"

- 1 pebble → Simple cautious walk to the right
- 2 pebbles → Double cautious walk to the left
- 0 pebbles → Non-cautious walk to the left

Ping Pong algorithm

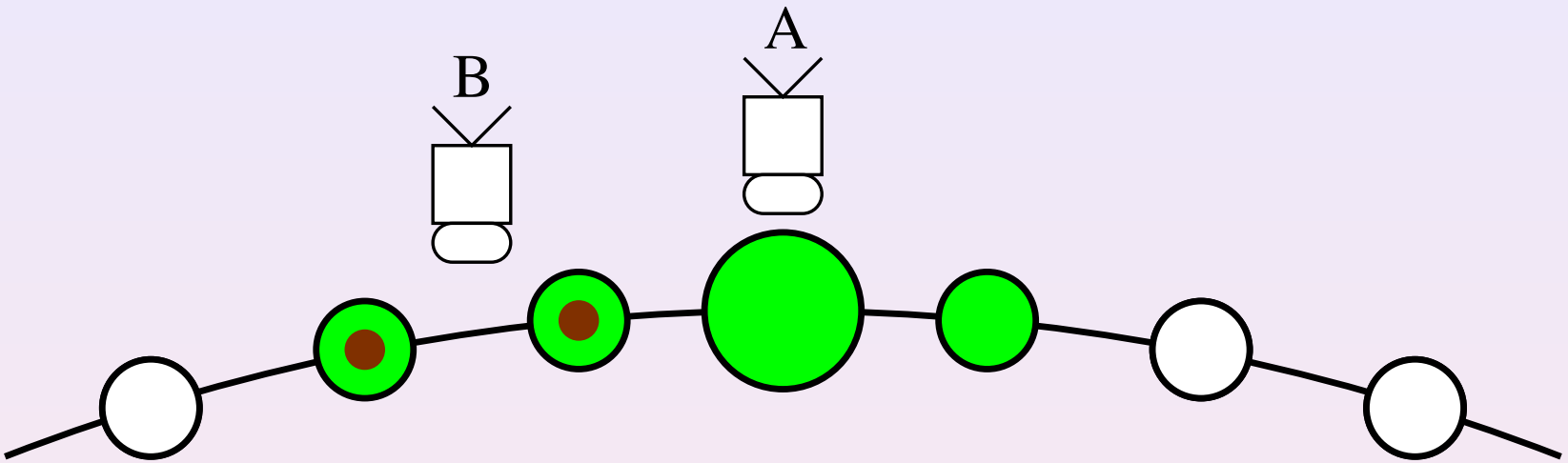


Quick description

If the agent "controls"

- 1 pebble → Simple cautious walk to the right
- 2 pebbles → Double cautious walk to the left
- 0 pebbles → Non-cautious walk to the left

Ping Pong algorithm

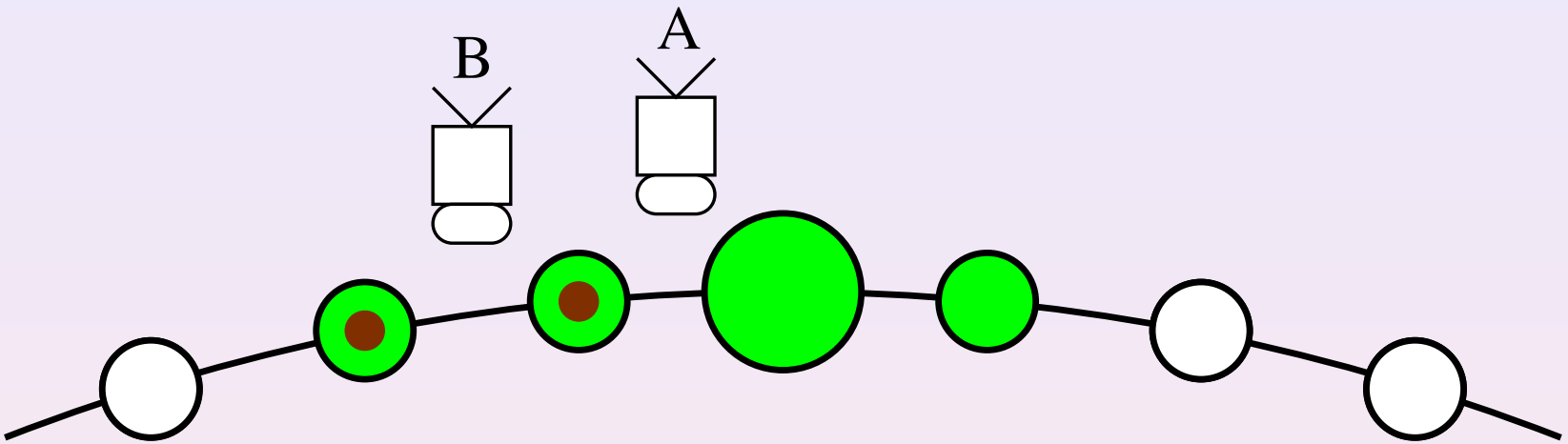


Quick description

If the agent "controls"

- 1 pebble → Simple cautious walk to the right
- 2 pebbles → Double cautious walk to the left
- 0 pebbles → Non-cautious walk to the left

Ping Pong algorithm

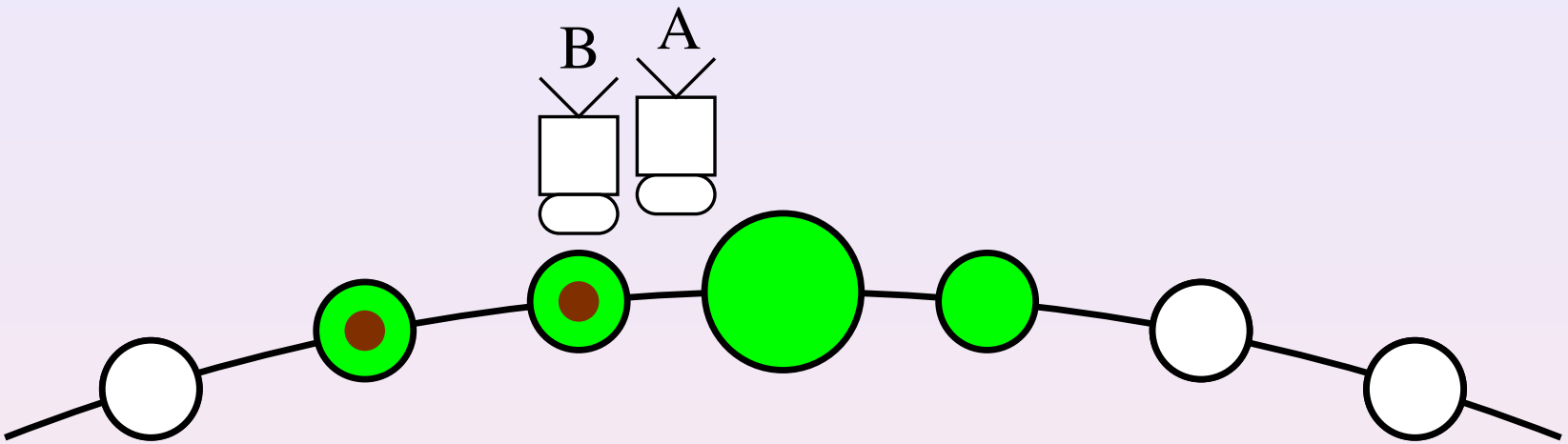


Quick description

If the agent "controls"

- 1 pebble → Simple cautious walk to the right
- 2 pebbles → Double cautious walk to the left
- 0 pebbles → Non-cautious walk to the left

Ping Pong algorithm

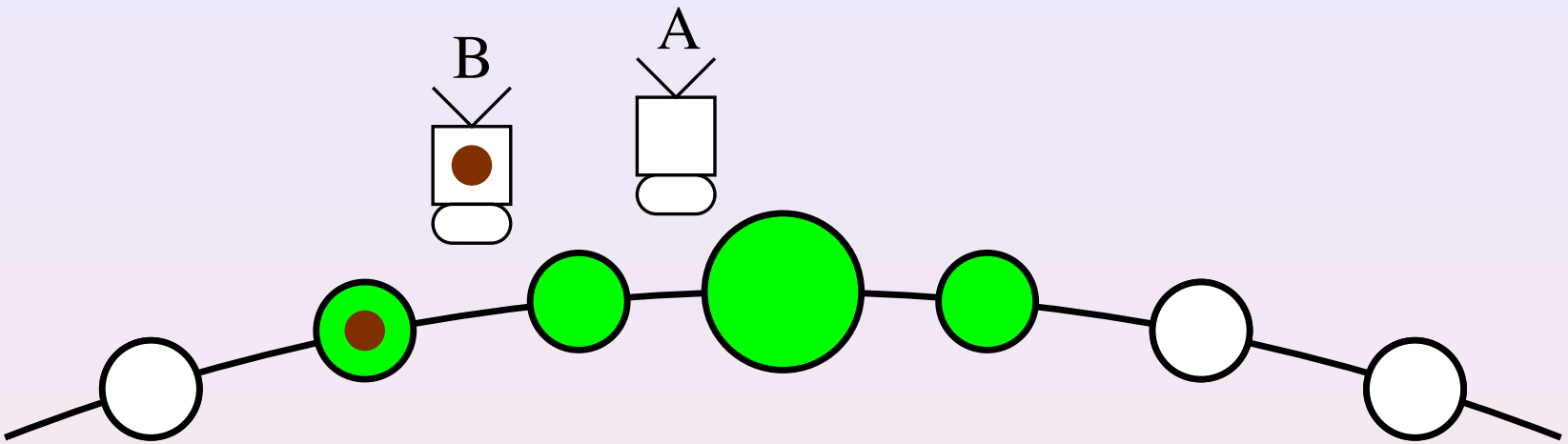


Quick description

If the agent "controls"

- 1 pebble → Simple cautious walk to the right
- 2 pebbles → Double cautious walk to the left
- 0 pebbles → Non-cautious walk to the left

Ping Pong algorithm

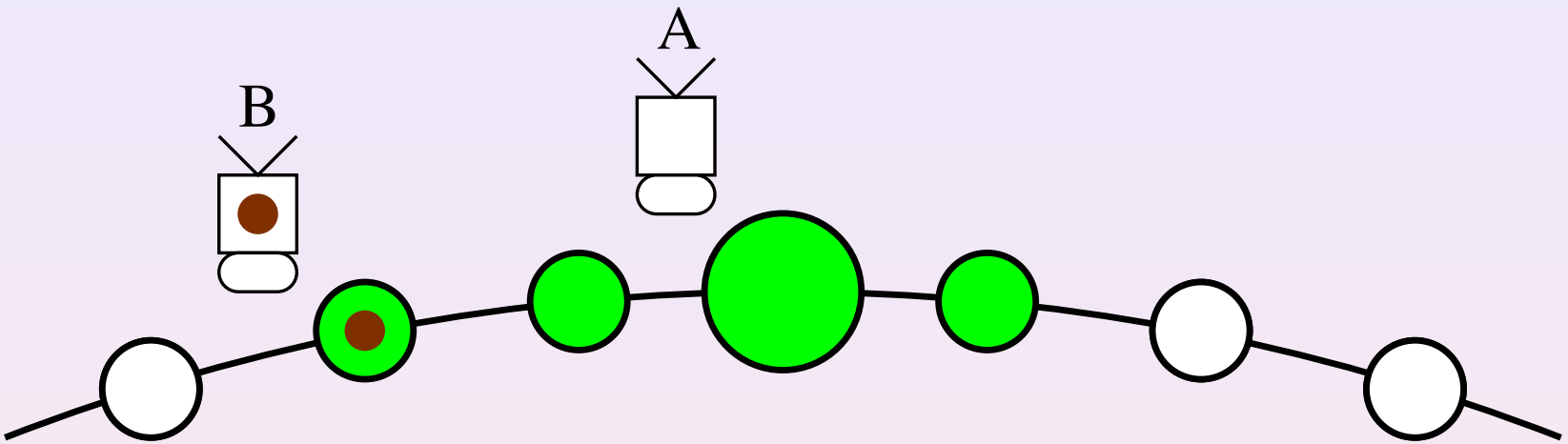


Quick description

If the agent "controls"

- 1 pebble → Simple cautious walk to the right
- 2 pebbles → Double cautious walk to the left
- 0 pebbles → Non-cautious walk to the left

Ping Pong algorithm

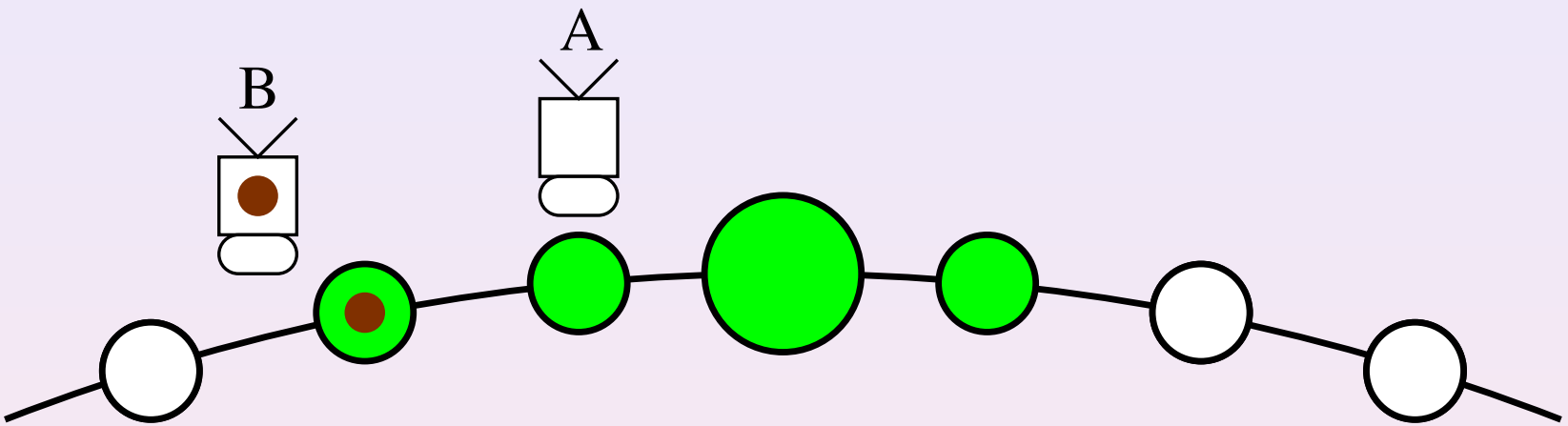


Quick description

If the agent "controls"

- 1 pebble → Simple cautious walk to the right
- 2 pebbles → Double cautious walk to the left
- 0 pebbles → Non-cautious walk to the left

Ping Pong algorithm

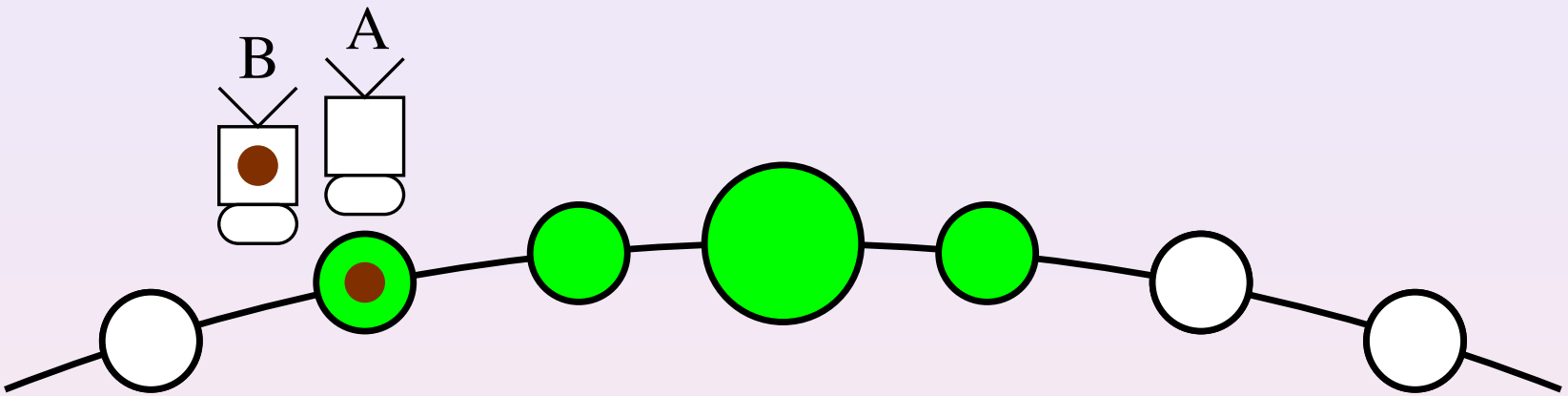


Quick description

If the agent "controls"

- 1 pebble → Simple cautious walk to the right
- 2 pebbles → Double cautious walk to the left
- 0 pebbles → Non-cautious walk to the left

Ping Pong algorithm

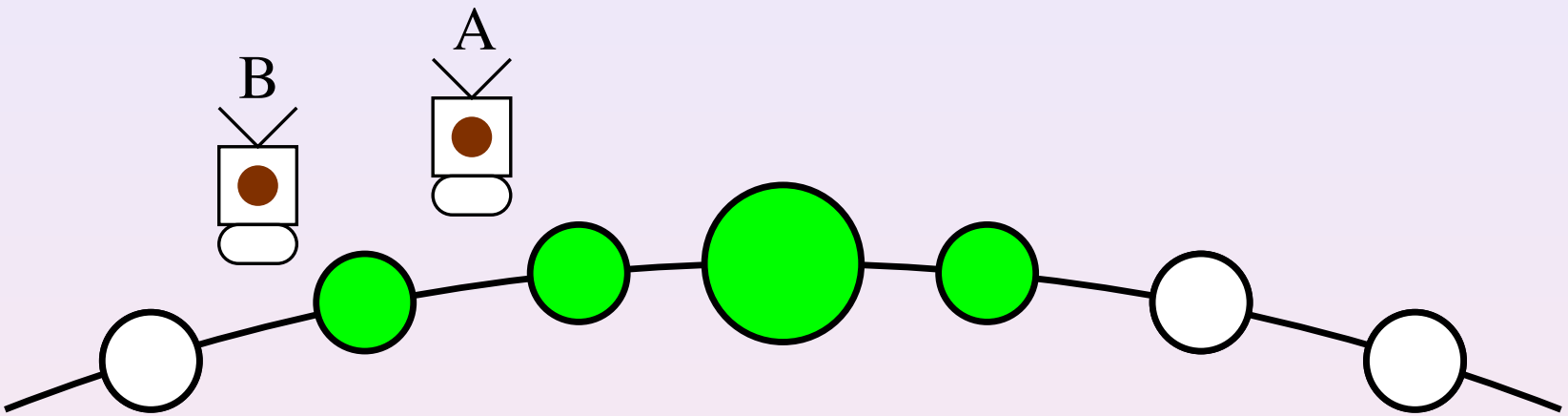


Quick description

If the agent "controls"

- 1 pebble → Simple cautious walk to the right
- 2 pebbles → Double cautious walk to the left
- 0 pebbles → Non-cautious walk to the left

Ping Pong algorithm

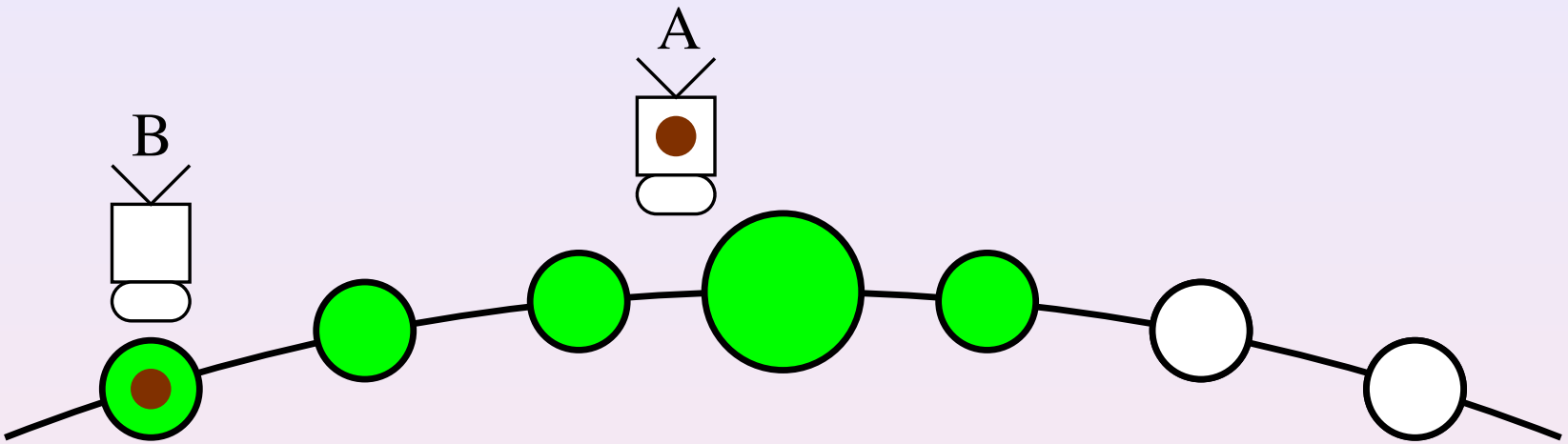


Quick description

If the agent "controls"

- 1 pebble → Simple cautious walk to the right
- 2 pebbles → Double cautious walk to the left
- 0 pebbles → Non-cautious walk to the left

Ping Pong algorithm

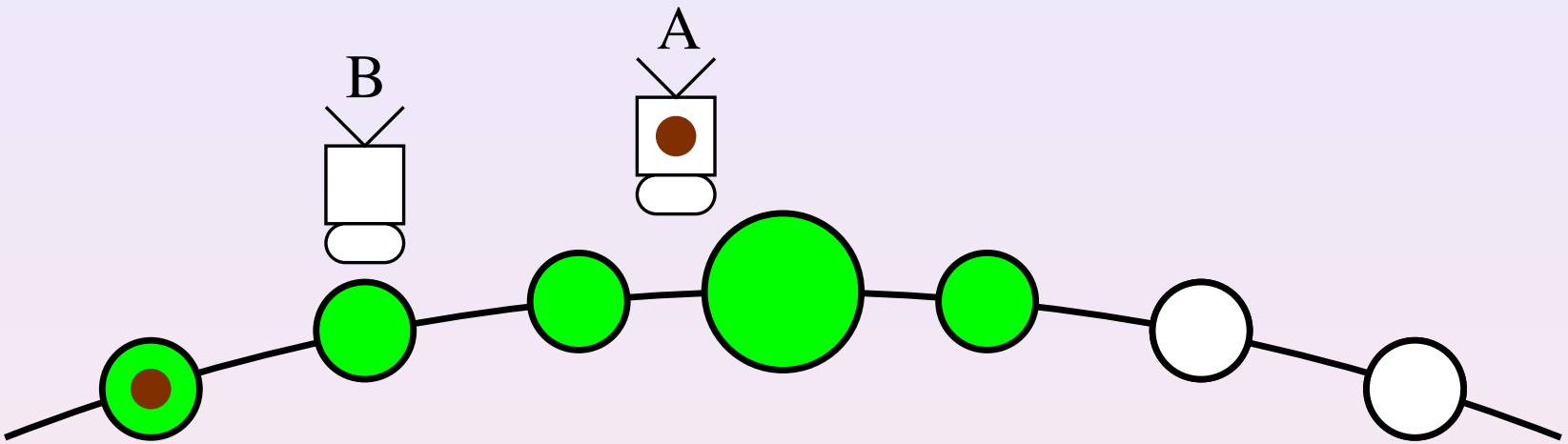


Quick description

If the agent "controls"

- 1 pebble → Simple cautious walk to the right
- 2 pebbles → Double cautious walk to the left
- 0 pebbles → Non-cautious walk to the left

Ping Pong algorithm

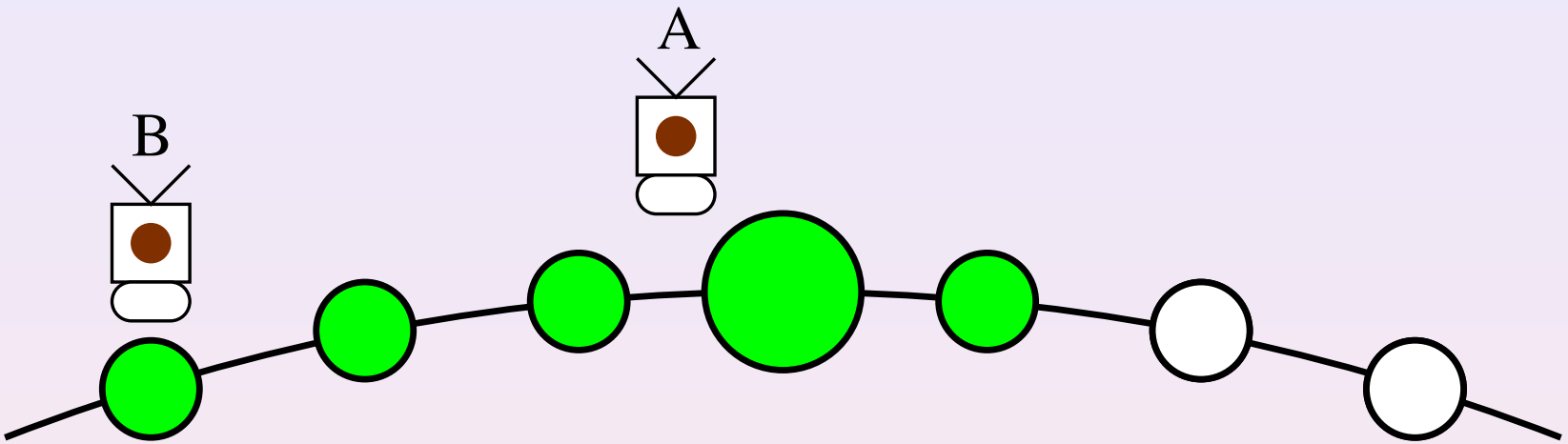


Quick description

If the agent "controls"

- 1 pebble → Simple cautious walk to the right
- 2 pebbles → Double cautious walk to the left
- 0 pebbles → Non-cautious walk to the left

Ping Pong algorithm

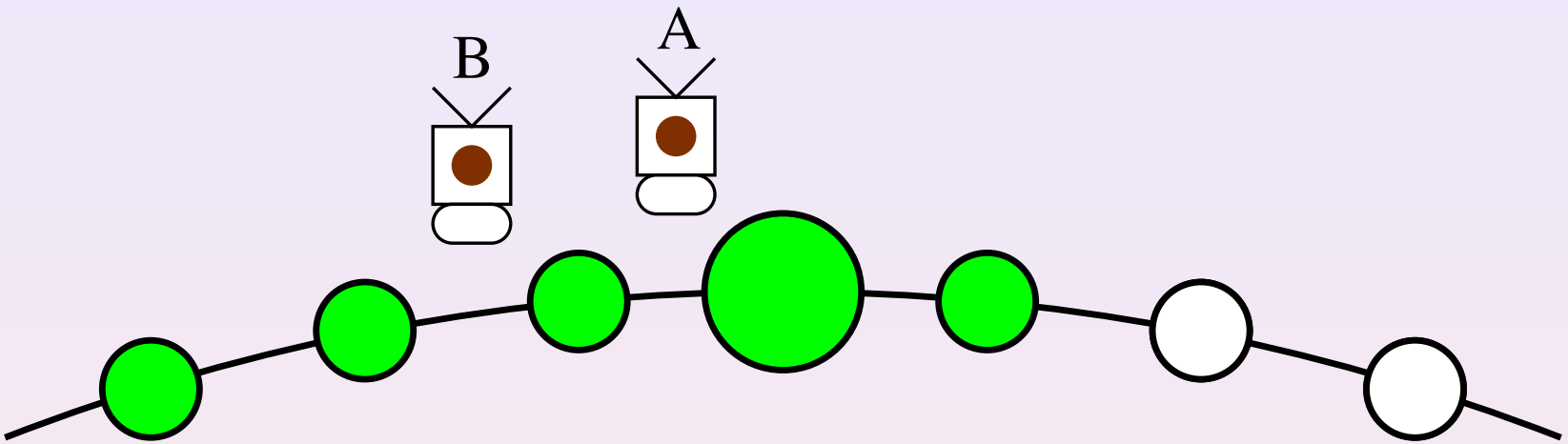


Quick description

If the agent "controls"

- 1 pebble → Simple cautious walk to the right
- 2 pebbles → Double cautious walk to the left
- 0 pebbles → Non-cautious walk to the left

Ping Pong algorithm

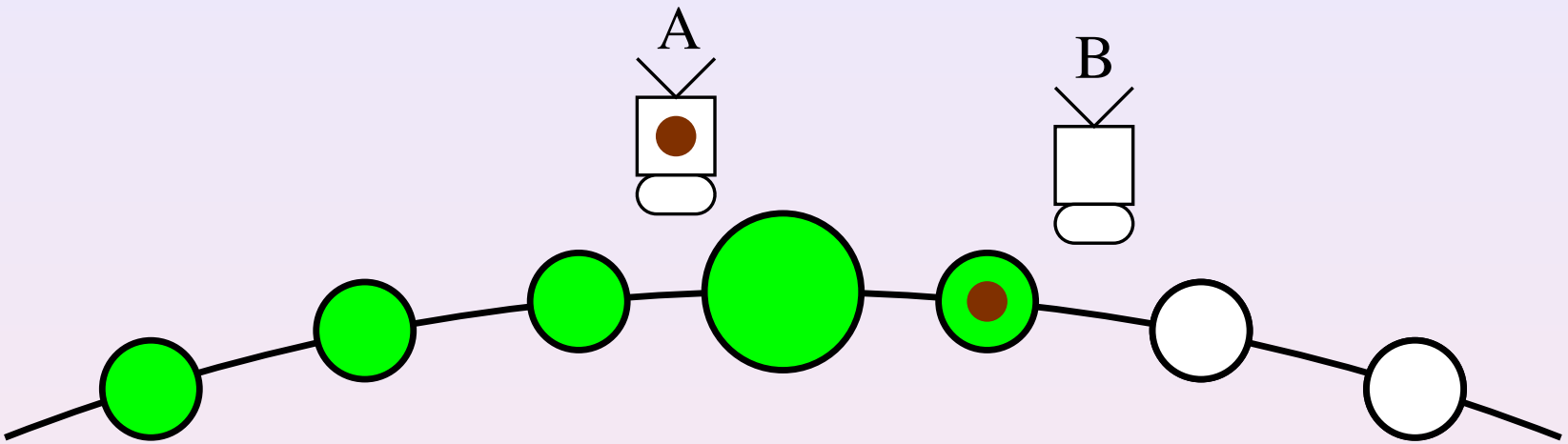


Quick description

If the agent "controls"

- 1 pebble → Simple cautious walk to the right
- 2 pebbles → Double cautious walk to the left
- 0 pebbles → Non-cautious walk to the left

Ping Pong algorithm

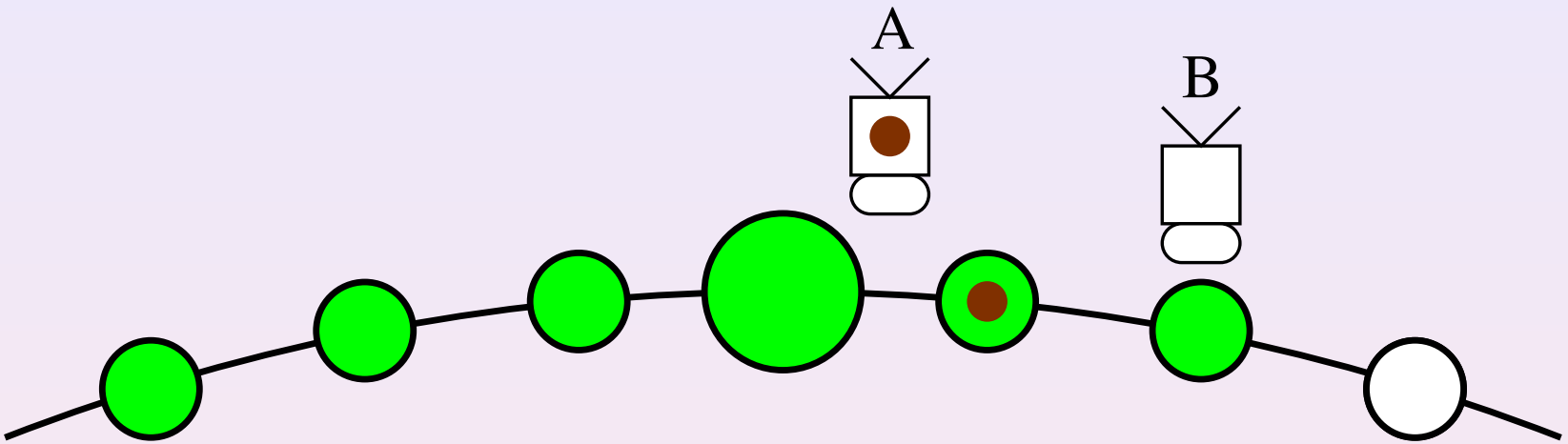


Quick description

If the agent "controls"

- 1 pebble → Simple cautious walk to the right
- 2 pebbles → Double cautious walk to the left
- 0 pebbles → Non-cautious walk to the left

Ping Pong algorithm

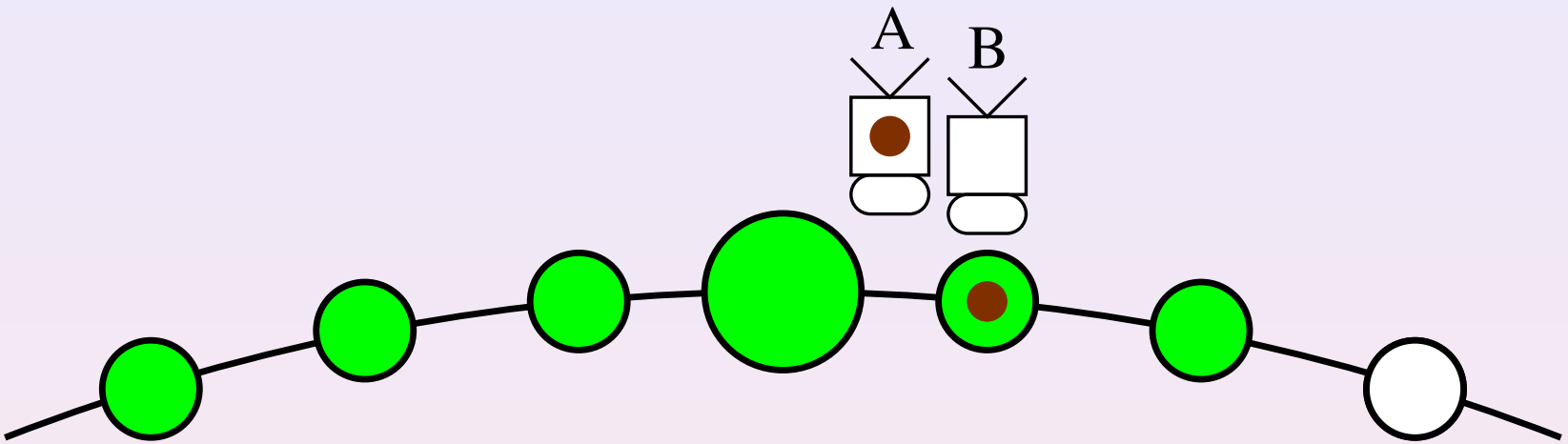


Quick description

If the agent "controls"

- 1 pebble → Simple cautious walk to the right
- 2 pebbles → Double cautious walk to the left
- 0 pebbles → Non-cautious walk to the left

Ping Pong algorithm

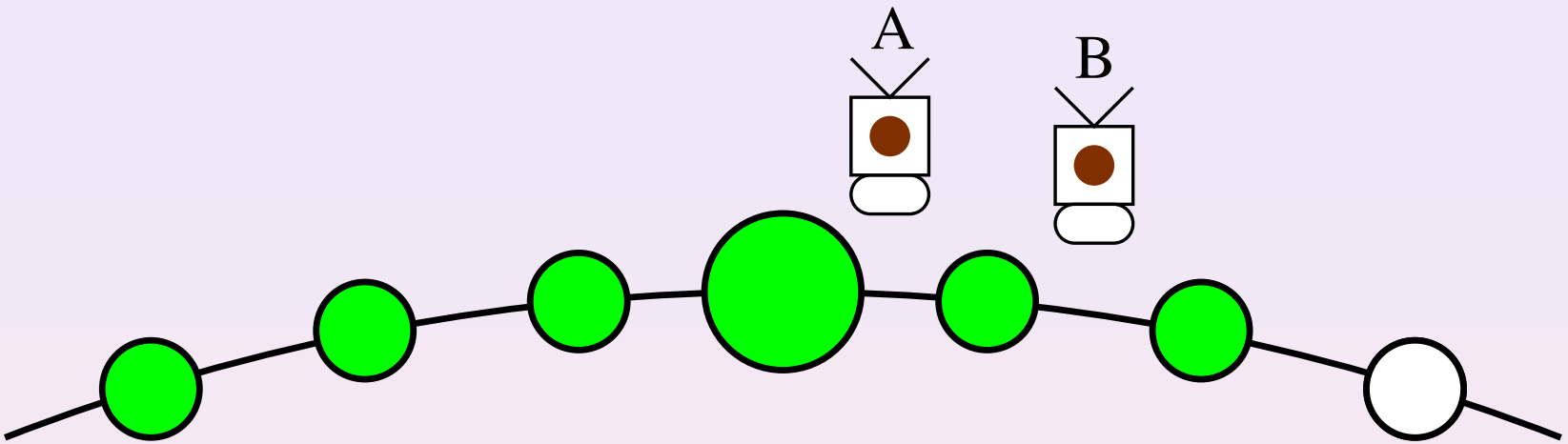


Quick description

If the agent "controls"

- 1 pebble → Simple cautious walk to the right
- 2 pebbles → Double cautious walk to the left
- 0 pebbles → Non-cautious walk to the left

Ping Pong algorithm

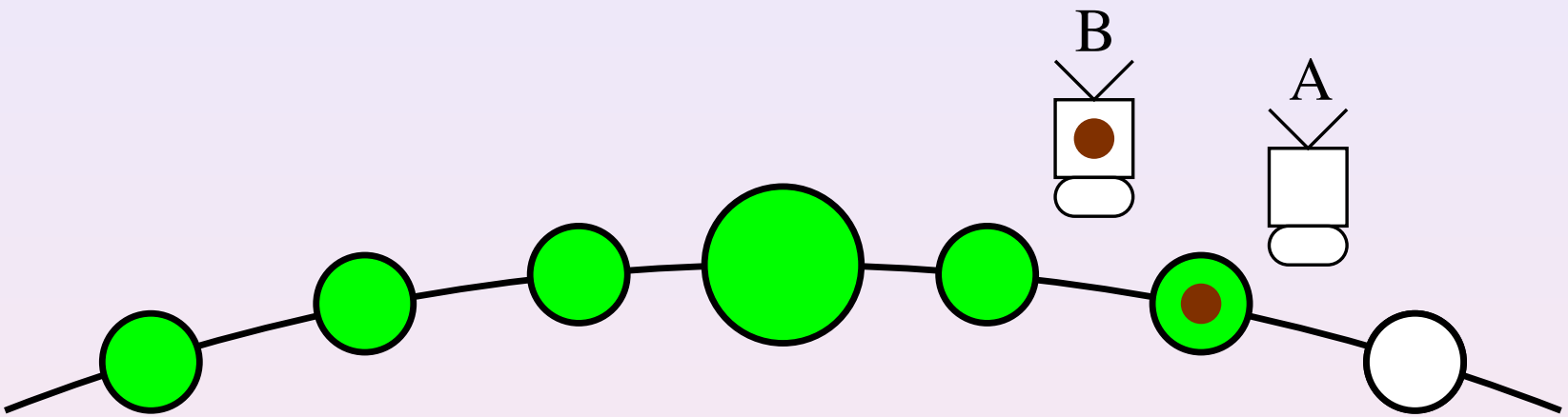


Quick description

If the agent "controls"

- 1 pebble → Simple cautious walk to the right
- 2 pebbles → Double cautious walk to the left
- 0 pebbles → Non-cautious walk to the left

Ping Pong algorithm



Quick description

If the agent "controls"

- 1 pebble → Simple cautious walk to the right
- 2 pebbles → Double cautious walk to the left
- 0 pebbles → Non-cautious walk to the left

Ping Pong algorithm

$\Rightarrow$ The algorithm is correct and uses $O(n^2)$ moves

Quick description

If the agent "controls"

- 1 pebble $\rightarrow$ Simple cautious walk to the right
- 2 pebbles $\rightarrow$ Double cautious walk to the left
- 0 pebbles $\rightarrow$ Non-cautious walk to the left

Enhanced Ping Pong algorithm

Quick description

- Execute Ping Pong **until** agents "meet" at least two/three nodes from the homebase (until the safe zone becomes sufficiently large)
- **While** still an unexplored node **do**
- Divide the unexplored part in two
- Each agent explores an half
- The first agent to finish its part goes and helps the other agent to explore the other part
- **When** an agent finds the pebble of the other agent
  - it moves the pebble by one node toward the center
  - and restart the loop

Analysis of Enhanced Ping Pong

Analysis

Between two halving

- Roughly half of the unexplored nodes are explored
- Agents do at most $O(n)$ moves

Result

- 1 pebble per agent
- 2 agents
- $\Theta(n \log n)$ moves

Analysis of Enhanced Ping Pong

Analysis

Between two halving

- Roughly half of the unexplored nodes are explored
- Agents do at most $O(n)$ moves

Result

- 1 pebble per agent
- 2 agents
- $\Theta(n \log n)$ moves

Outline

Recent trends in distributed computing

Formalization / Generalization

- Simulation / equivalence between models
- Computational complexity theories for distributed computing

Towards more realistic scenarios

- More performance measures (incl. space complexity)
- Dynamic settings
- Distributed Computing inspired by nature
- Heterogeneous computing entities
- More complex tasks

The confidence issue in distributed computing

Quote from Lamport

[Concurrent (multiprocess)] algorithms can be quite subtle and hard to get right; their correctness proofs require a degree of precision and rigor unknown to most mathematicians (and many computer scientists).

Main reasons

- Multiple computing entities
- Non-determinism due to asynchrony and faults
- Importance of knowledge in impossibility results

Possible solutions

More structured and detailed proofs

Already advocated by Lamport in the 90's

Verification (model checking)

- Useful for finding bugs
- Restricted: combinatorial explosion or undecidability

Certification (proof assistants)

- Not fully automated anymore
- Huge expressiveness
- Can handle both algorithms and impossibility results

Distributed computing projects using Coq

Loco (not currently active)

- Formal framework for Local computations
- Developed at Bordeaux (LaBRI)

Verdi

- Framework for implementing and verifying dist. systems
- Developed in the University of Washington

Pactole

- Framework for robots on continuous and discrete spaces
- Developed in France (mainly Paris, Lyon, Grenoble)

PADEC

- A framework for certified self-stabilization
- Developed at Grenoble (Verimag)

Thank you to my co-authors

PhD students: Christian Glacet, Ahmed Wade, Antonin Lentz

Postdoc: Evangelos Bampas

My other co-authors (after my PhD):

H. Arfaoui, L. Blin, N. Bonichon, C. Cooper, J. Czyzowicz, S. Devismes, S. Dobrev, P. Flocchini, P. Fraigniaud, L. Gasieniec, C. Gavoille, N. Hanusse, J. Jansson, C. Johnen, R. Klasing, T. Kociumaka, A. Kosowski, D. Kowalski, A. Labourel, I. Lignos, R. Martin, F. Mathieu, N. Nisse, D. Pajak, A. Pelc, M. Potop-Butucaru, T. Radzik, K. Sadakane, N. Santoro, D. Soguet, W.-K. Sung, S. Tixeuil