Wait-free Robot Gathering Problems on Graphs with Termination

Sergio Rajsbaum

Based on results in *Distributed Computing* 2019

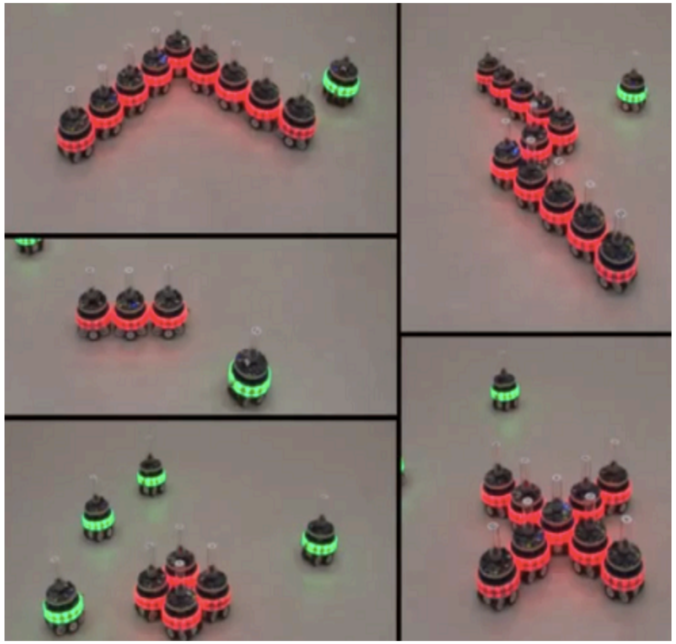
MANUEL ALCANTARA
ARMANDO CASTAÑEDA
DAVID FLORES
SERGIO RAJSBAUM

Instituto de Matemáticas
UNAM, Mexico

October 12, 2021
Ecole Polytechnique

Introduction

- Since 1999 much interest in studying distributed algorithms for mobile robots in Look-Compute-Move models



Introduction

- n robots located in some space
- operate in Look-Compute-Move cycles:
  - **Looks** at its surroundings and obtains a snapshot containing the positions of all robots;
  - **Computes** a destination, and then
  - **Moves** to an nearby destination

Introduction

- Robots located in various spaces have been considered:
  - Plane
  - Sphere, torus, …
  - Graphs

Introduction

- What problems can the robots solve as a function of their capabilities?

Introduction

- What problems can the robots solve as a function of their capabilities?

- many variants have been *extensively* studied for over two decades (for many problems)

Introduction

- "… basic coordination problems: pattern formation, gathering, scattering, leader election,… we analyze the impact of the different assumptions on the robots' computability power."

Introduction

- ¿What kind of problems do the robots can solve under restricted capabilities?

  - Orientation sense
  - Memory
  - Environment visibility
  - Communication

Introduction

- ¿What kind of problems do the robots can solve under and under different models of timing?

➢ Timing:

Asynchronous, partially synchronous, synchronous

Introduction

- ¿What kind of problems do the robots can solve under and under different models of failures and timing?

  - Reliability:

    Often failure-free (not always)

  - Timing:

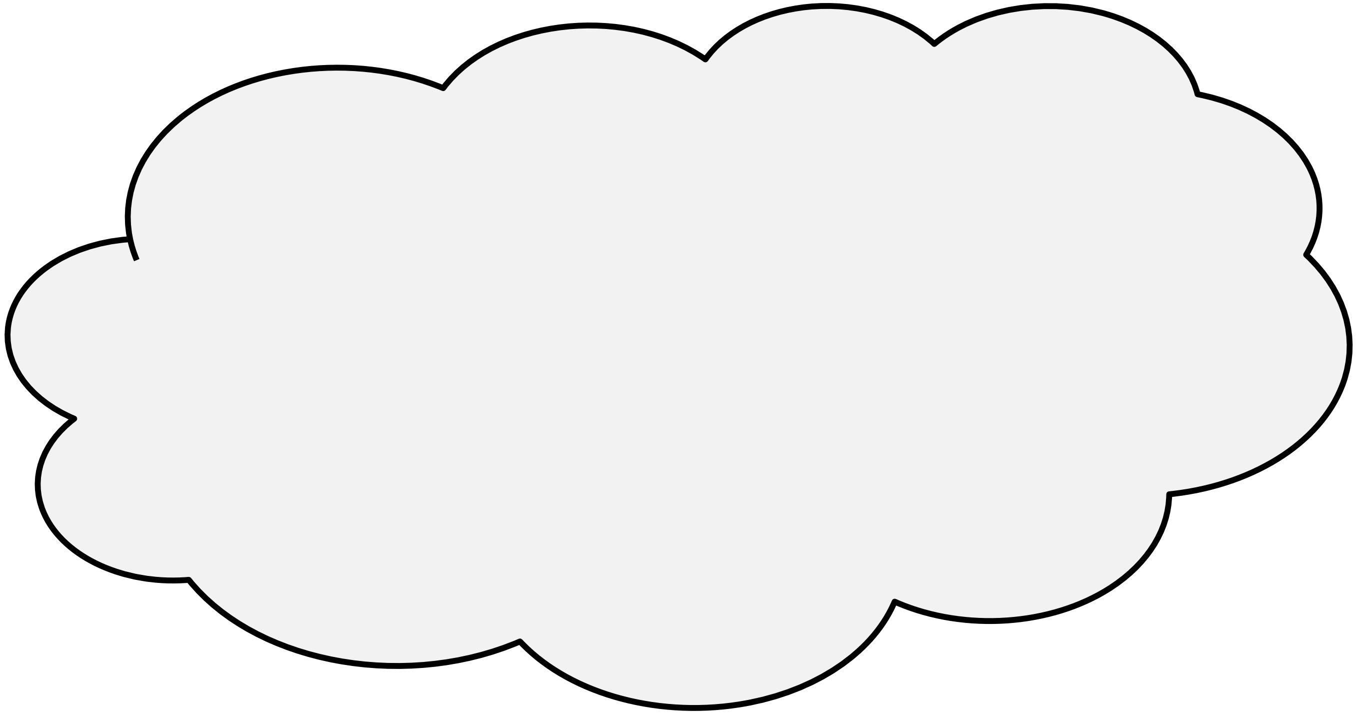
    Asynchronous, partially synchronous, synchronous

This work

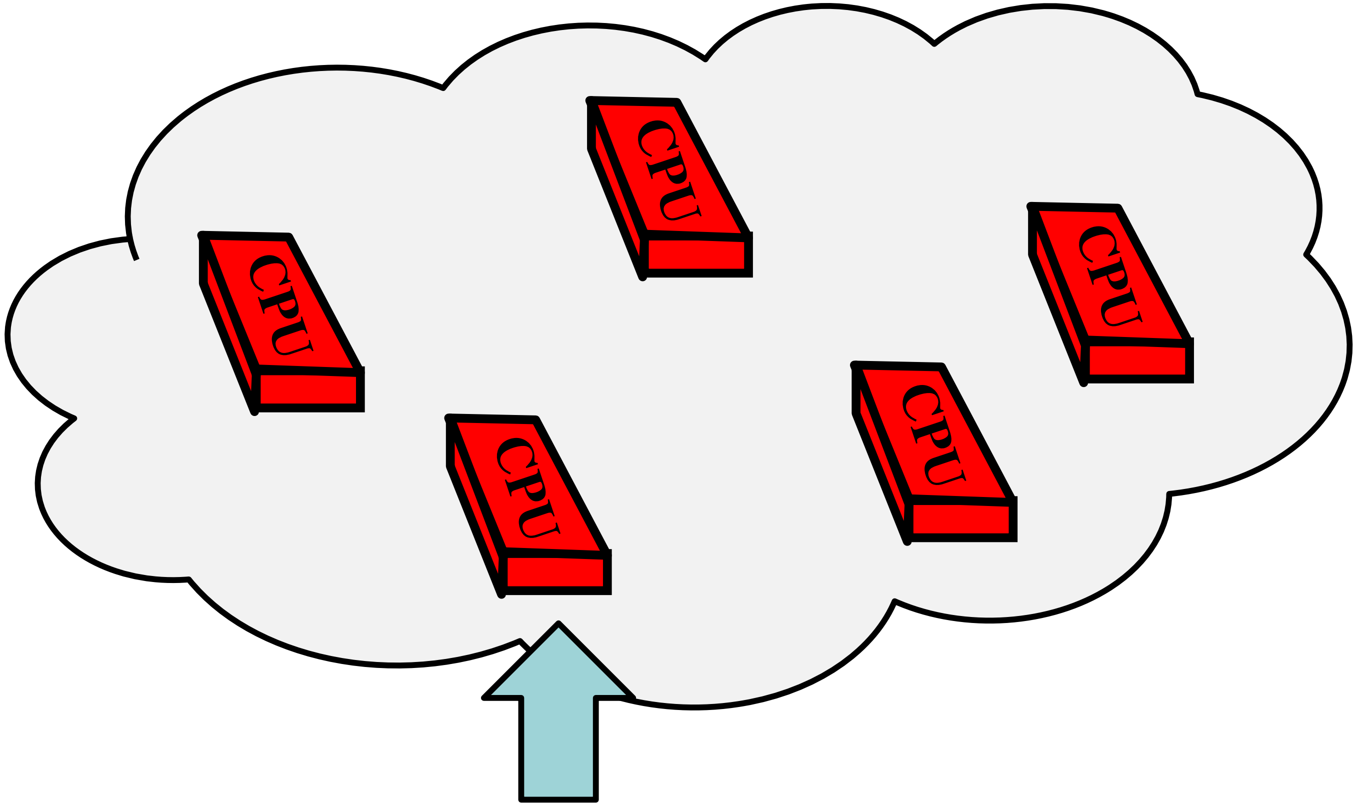
- ¿What kind of **problems** do the robots can solve under restricted capabilities, in some space, given a failure/timing model?

  - ➢ This work interested in a central concern: bringing the robots close to each other

A basic usual model

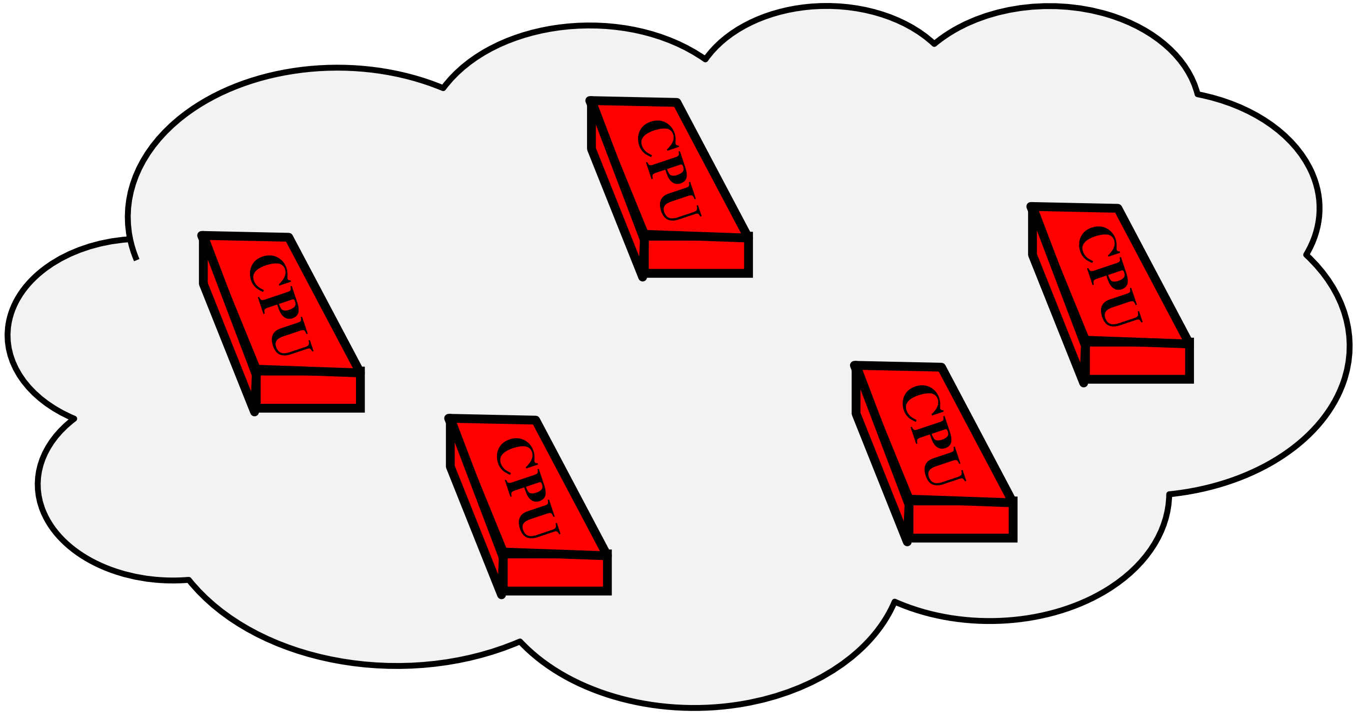


A basic usual model



***n* robots**

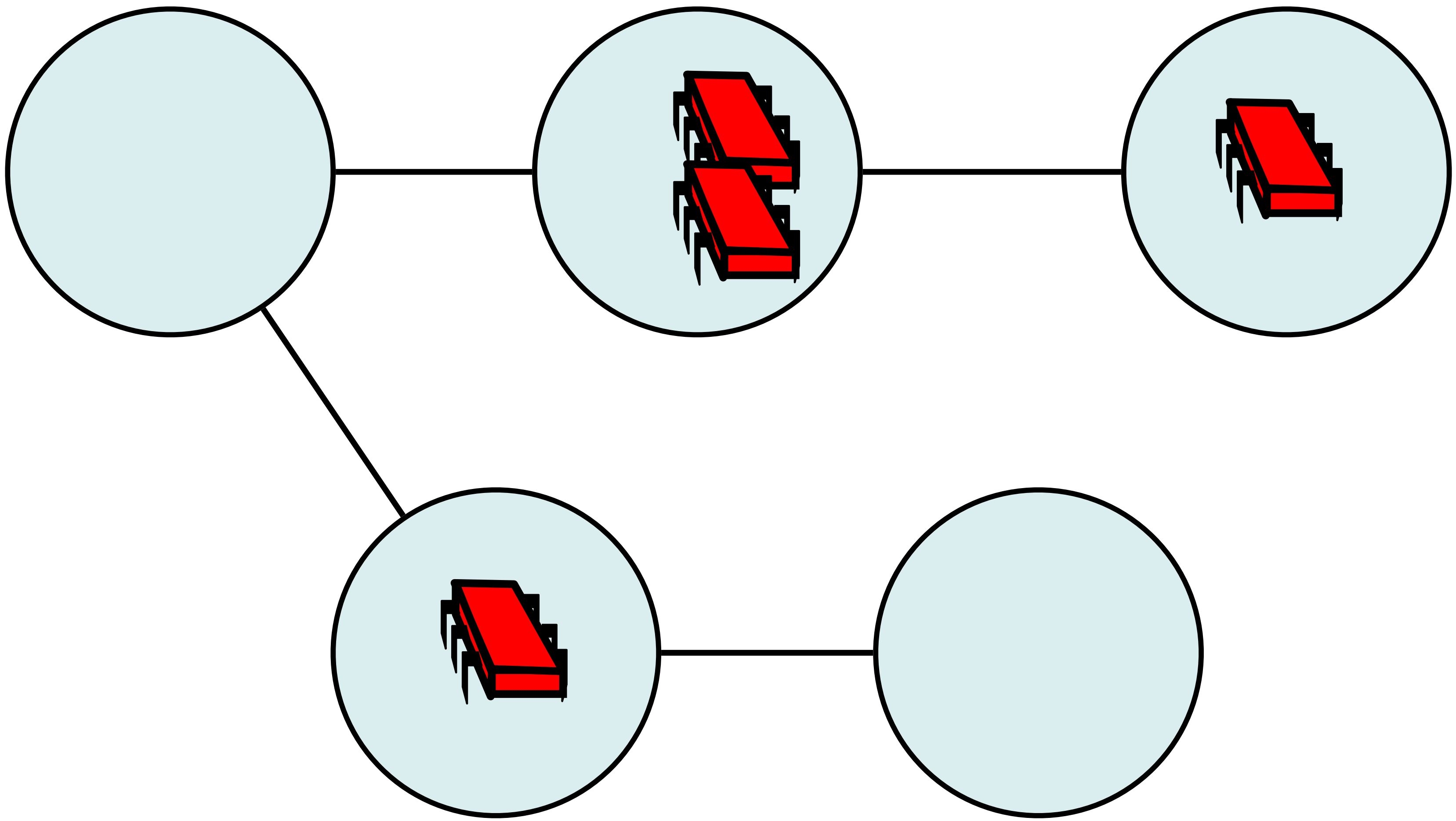
A basic usual model



- Asynchronous (There is no concept of global time)

A basic usual model

- The n robots are located on the vertices of a graph G

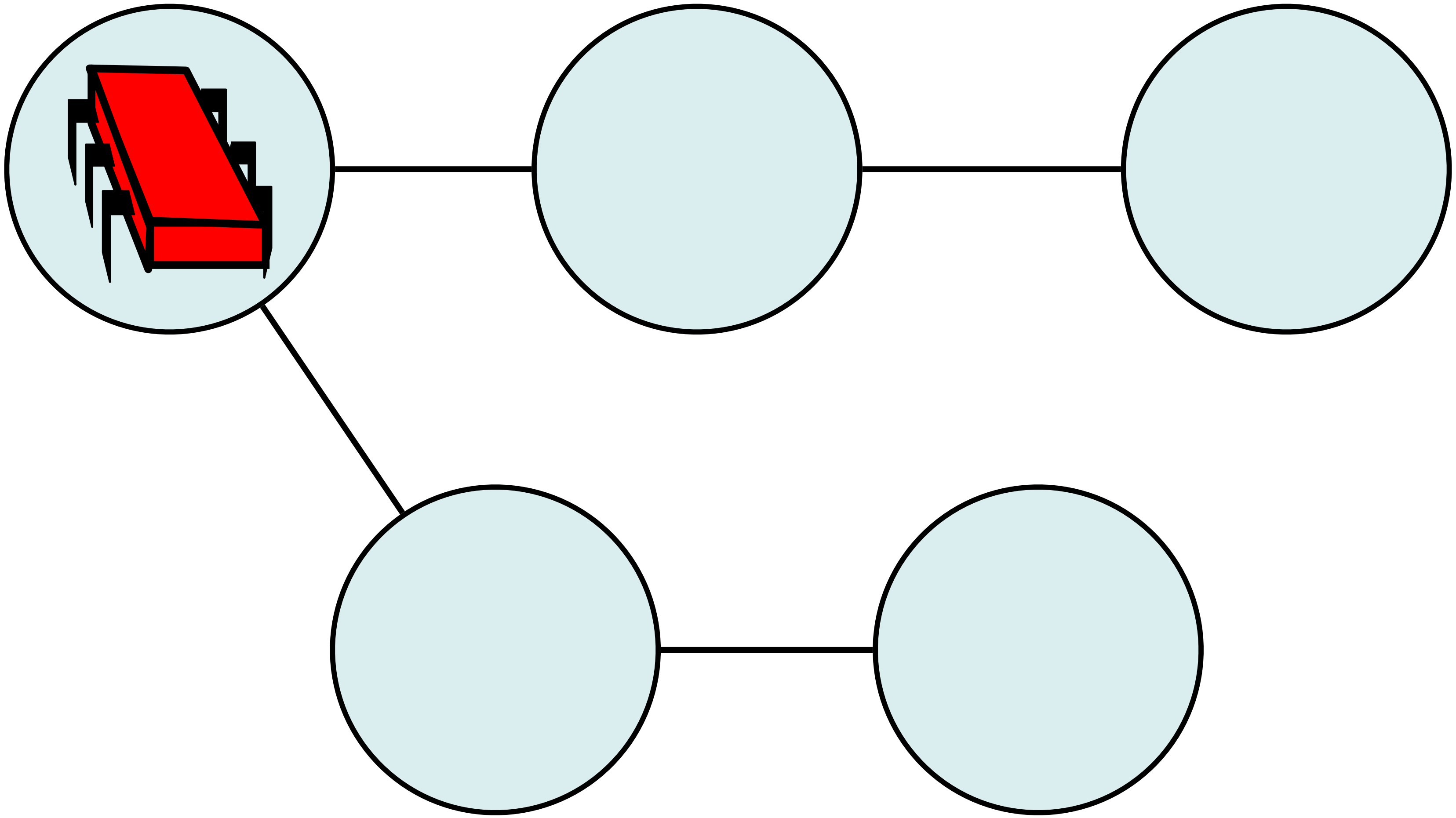


A basic usual model

- Robots move over edges on the vertices of a graph

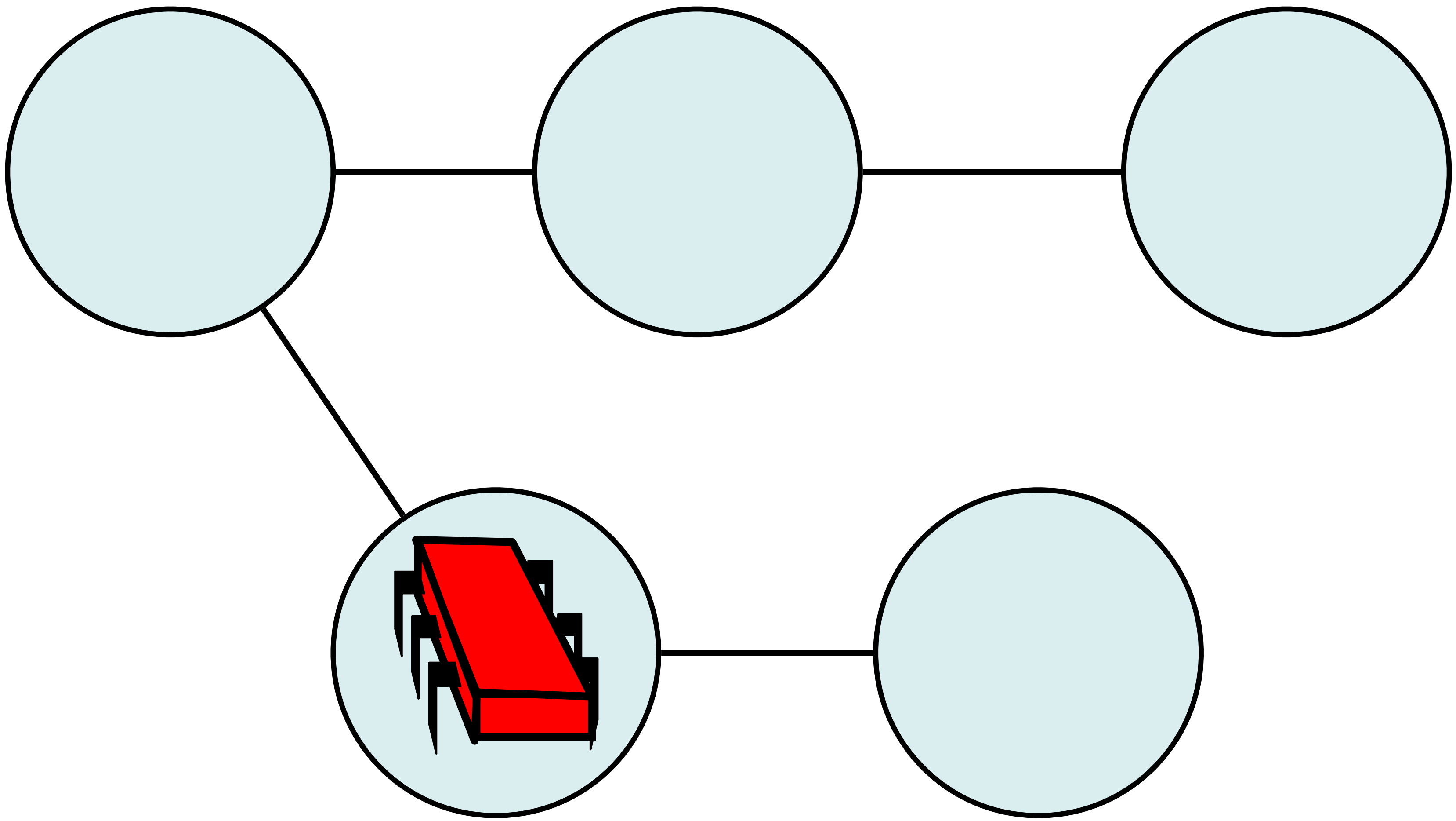
A basic usual model

- Robots move over edges on the vertices of a graph



A basic usual model

- Robots move over the edges on the vertices of a graph

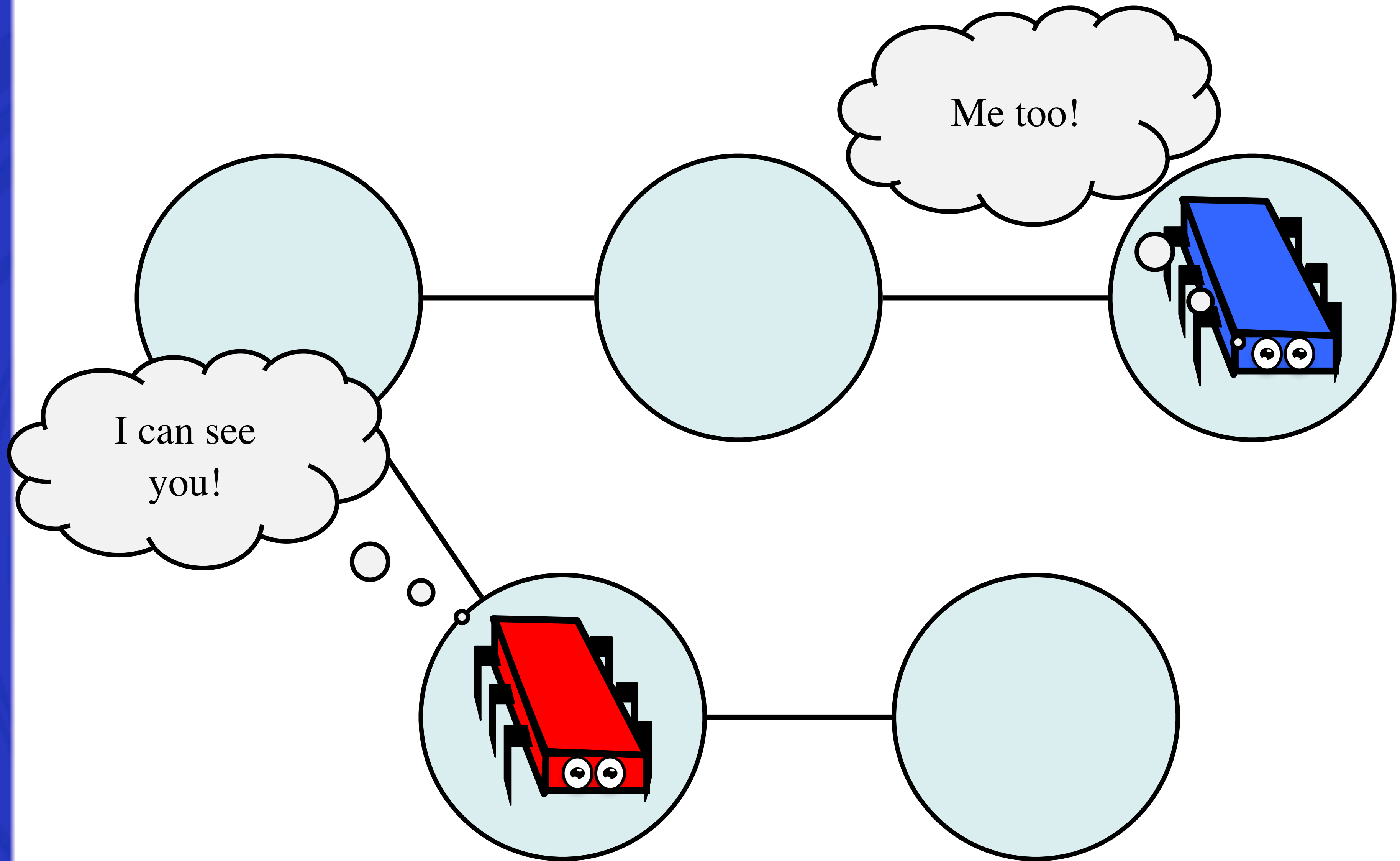


A basic usual model

- Use of cameras to see where other robots are located

A basic usual model

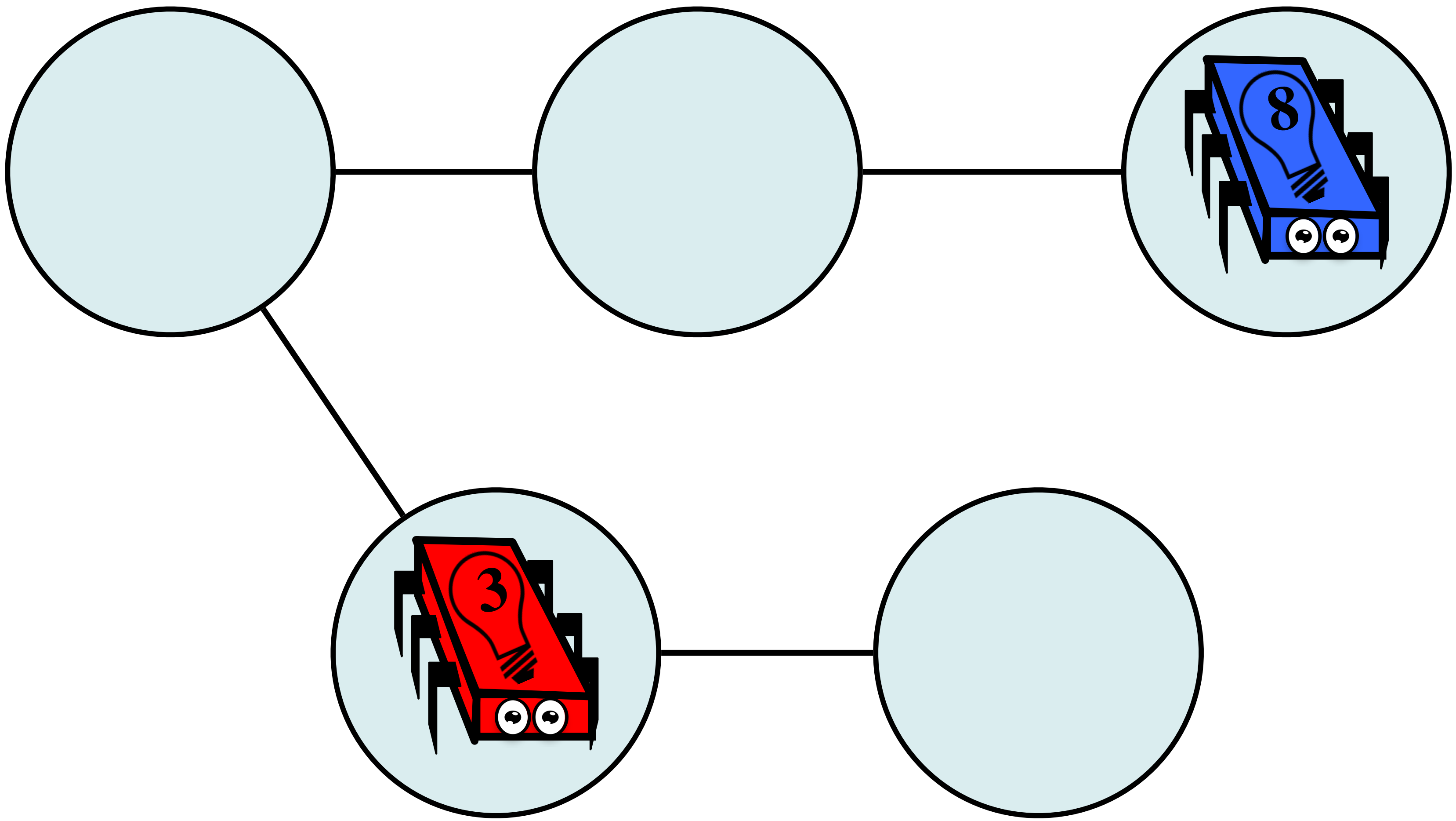
- Look: see the locations of all robots in a single observation



Asynchronous Robots with Lights (ARL)

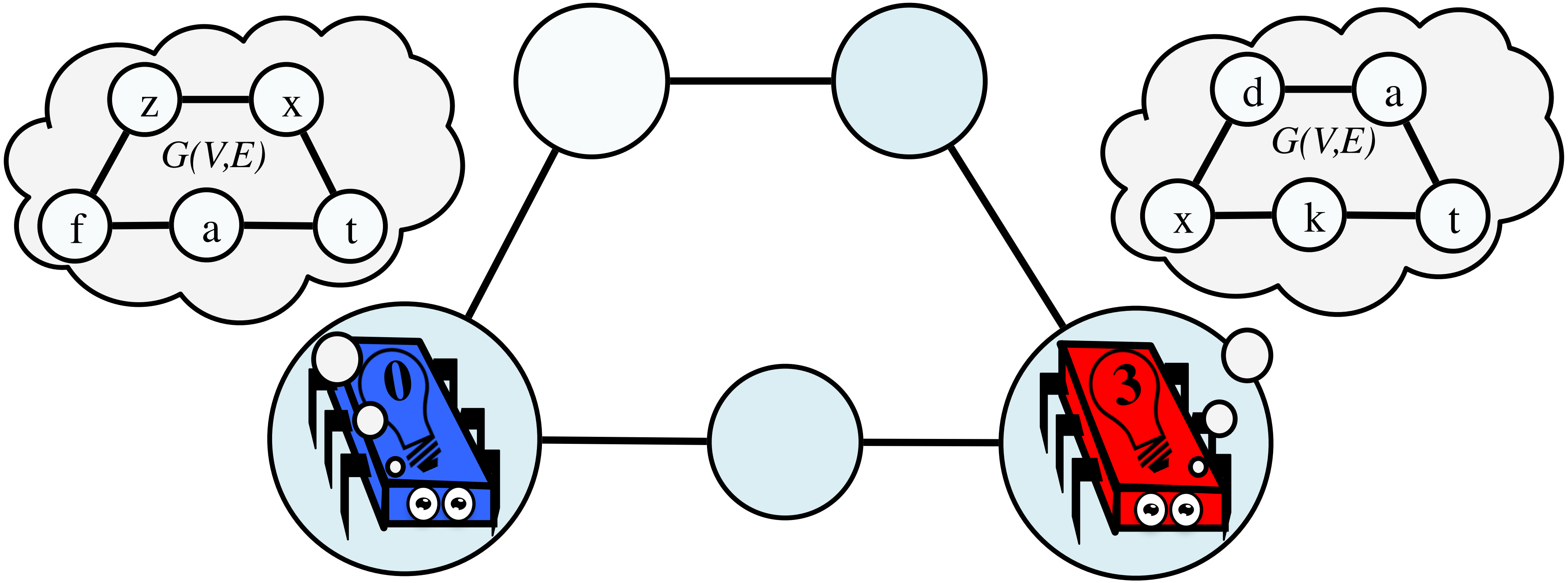
- External lights to transmit information

e.g. Das, Flocchini, Prencipe, Santoro, Yamashita, TCS 2016



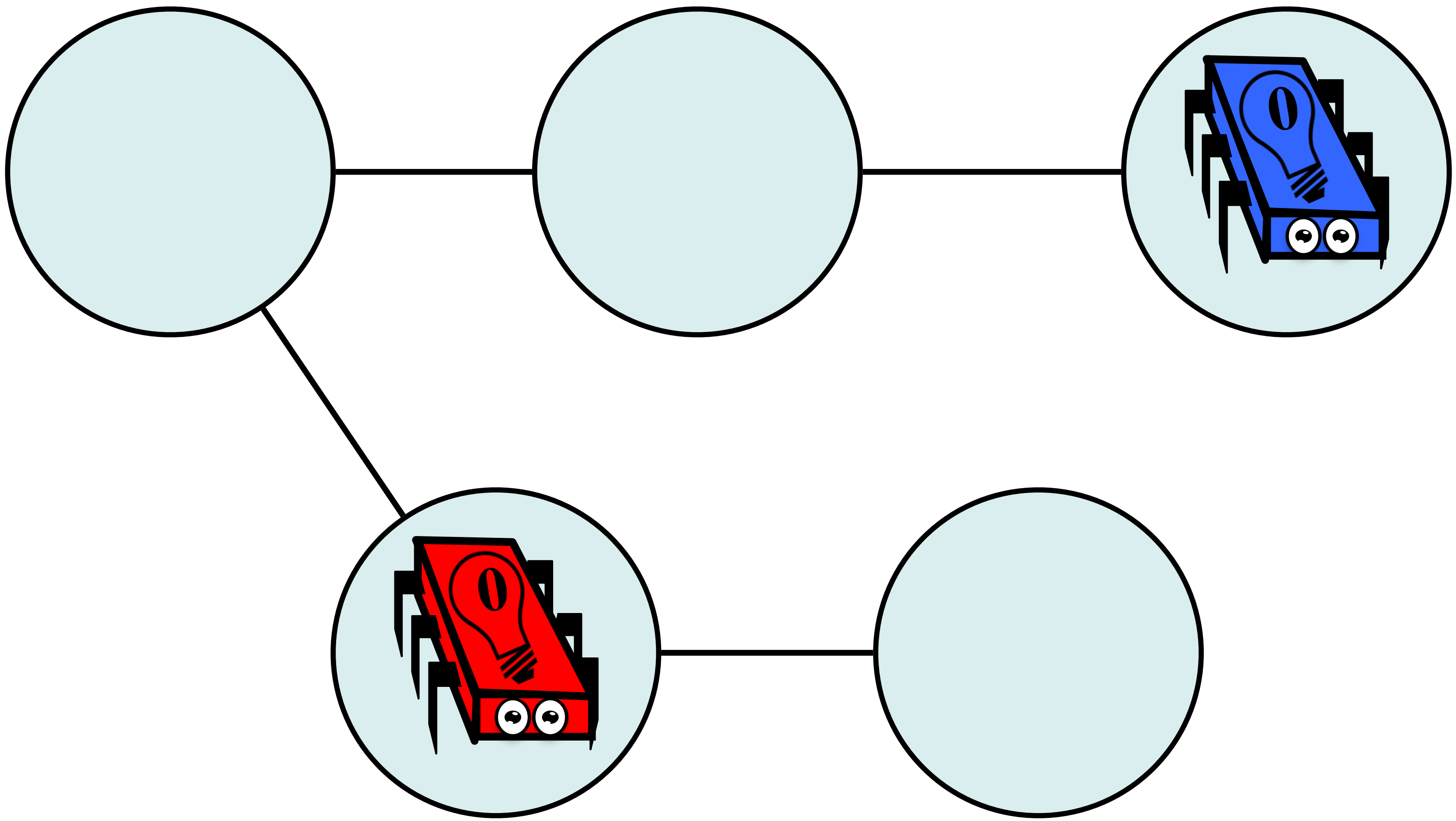
Asynchronous Robots with Lights (ARL)

- They know G (but may not have the same labeling)



Usual models: Waking Times

- All robots are present initially. Hence, they are visible during all the execution.

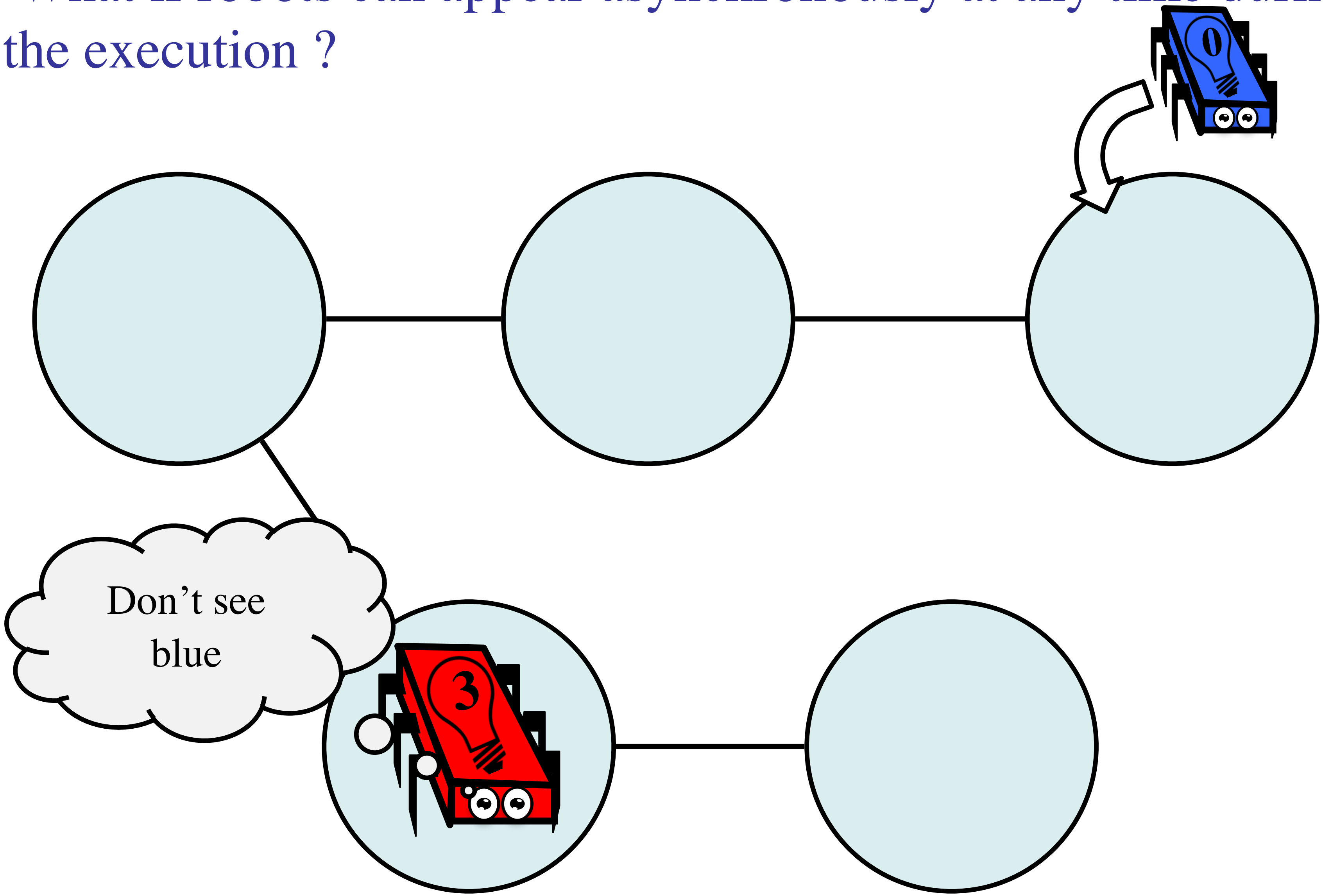


New: Arbitrary Waking Times

- What if robots can appear asynchronously at any time during the execution ?

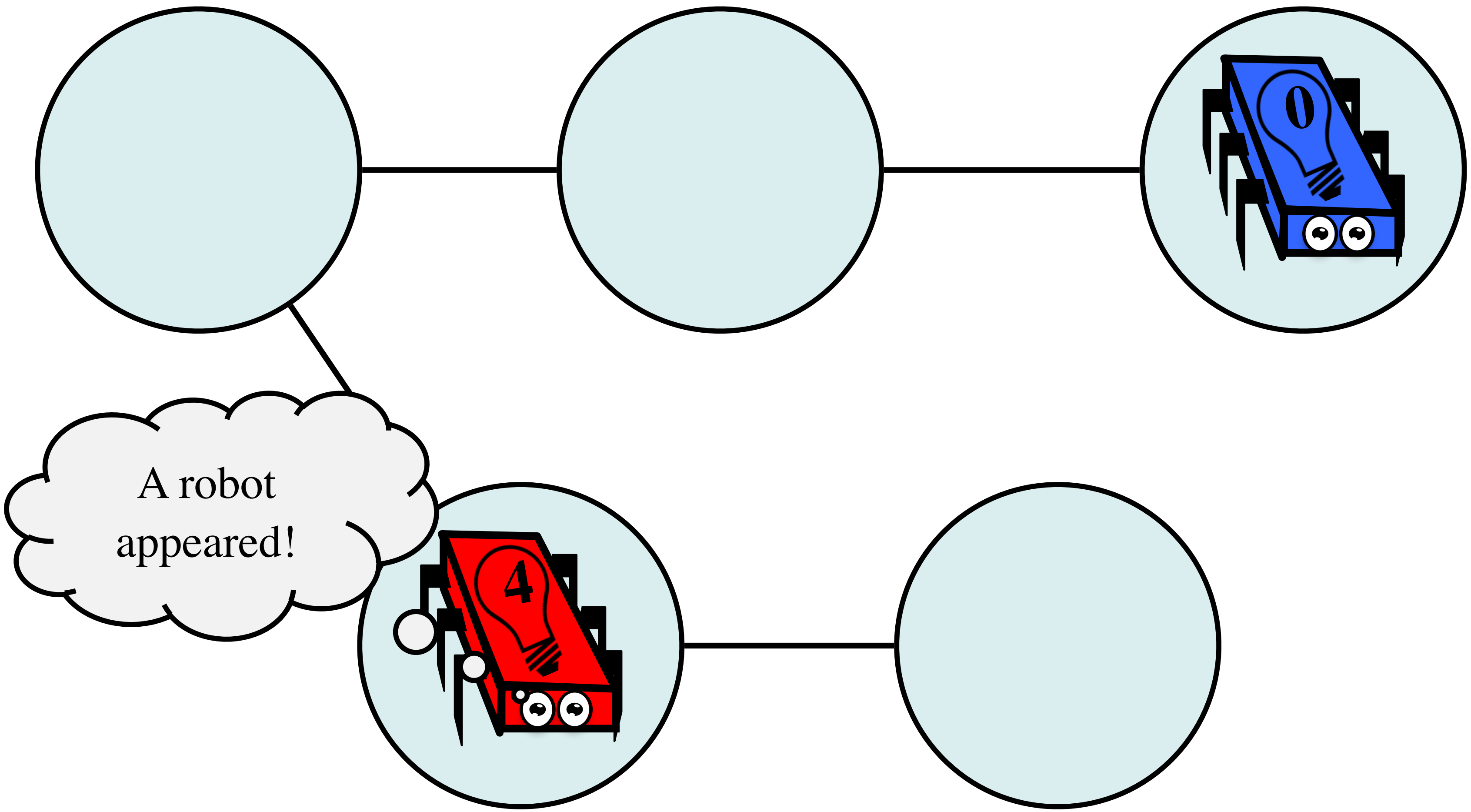
New: Arbitrary Waking Times

- What if robots can appear asynchronously at any time during the execution ?



Waking Times + failures

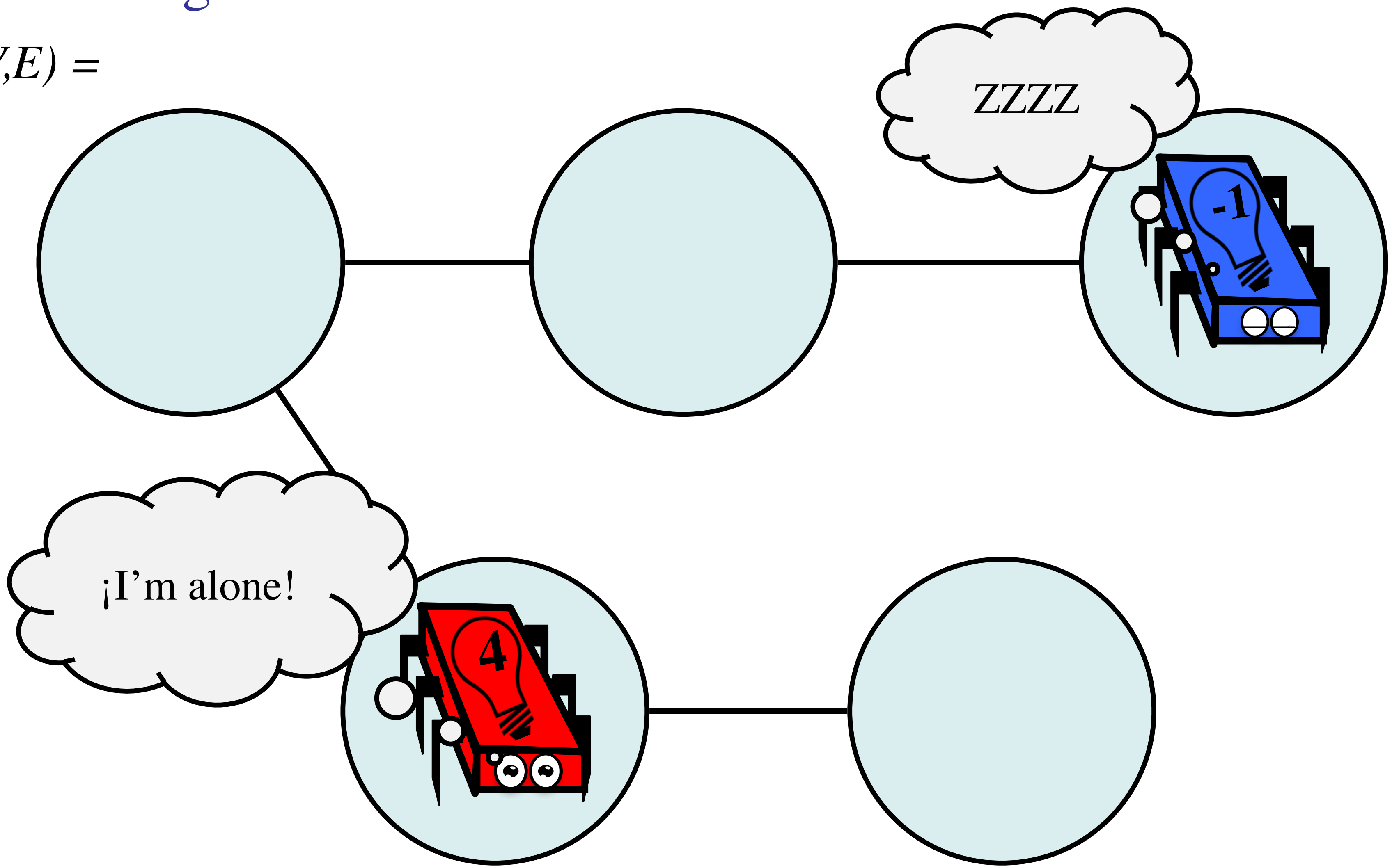
- An interesting combination!



Waking Time

- We modeled the arbitrary waking times with a negative number in the lights.

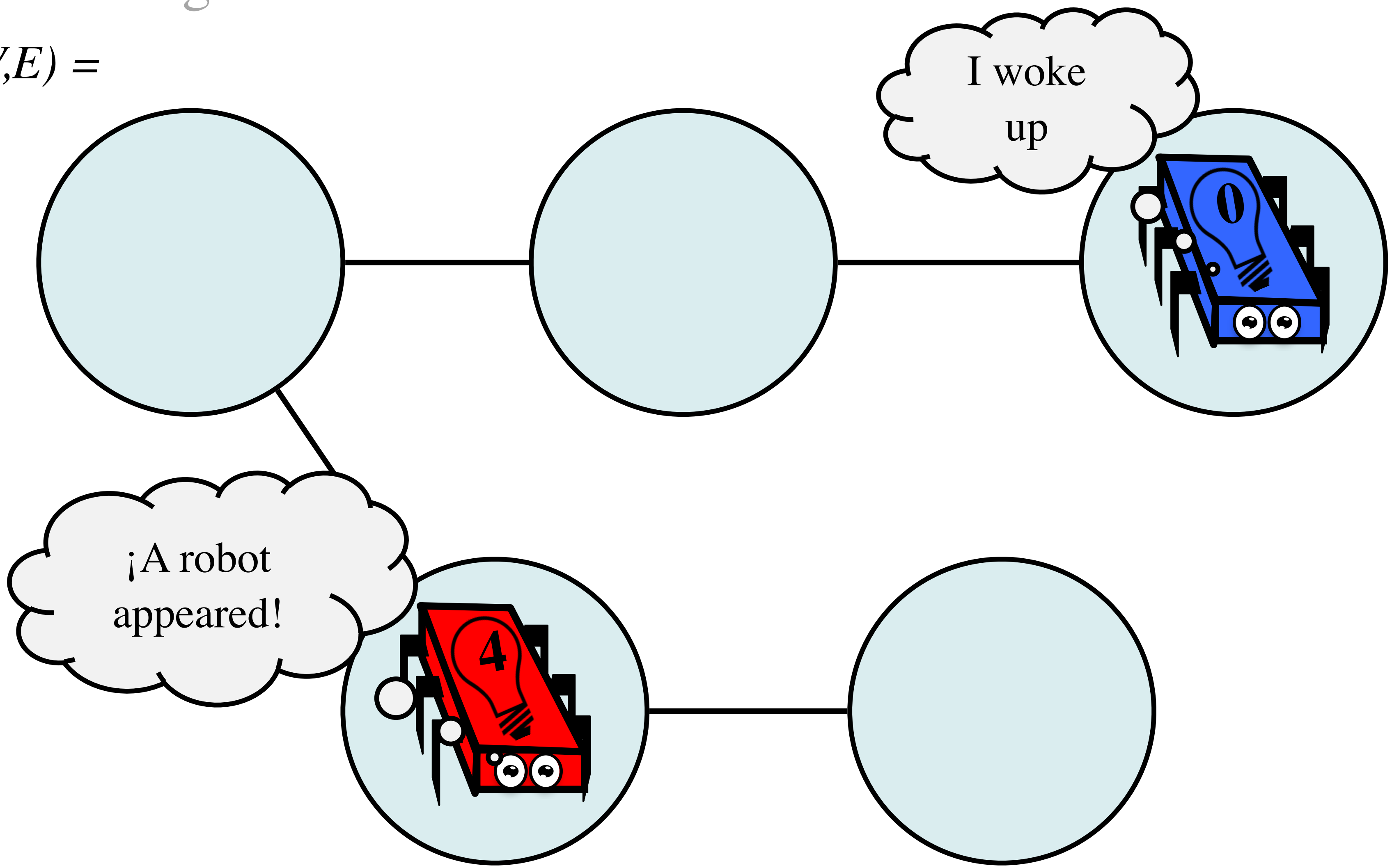
$G(V,E) =$



Waking Time

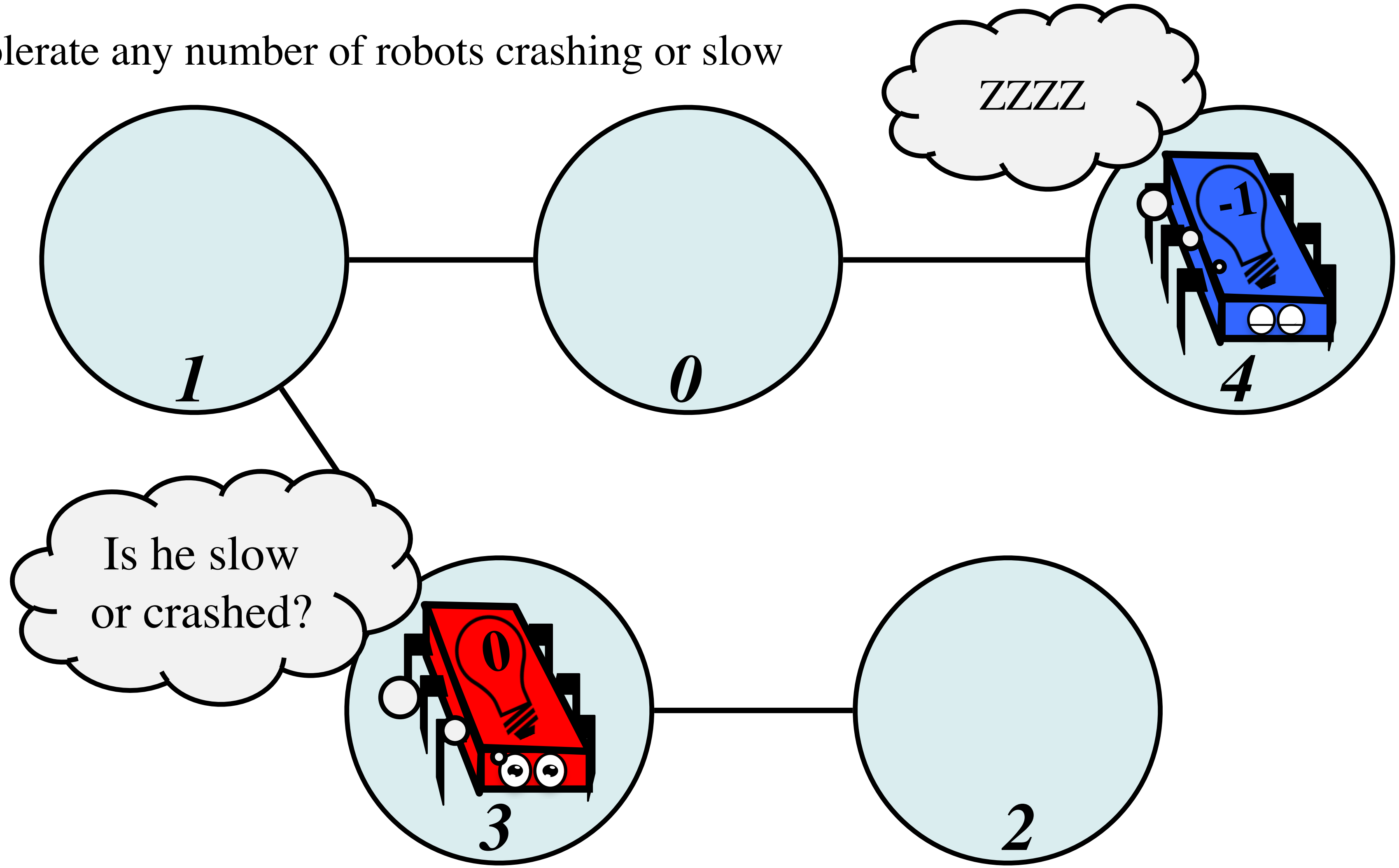
- We modeled the arbitrary waking times with a negative number in the lights.

$G(V,E) =$



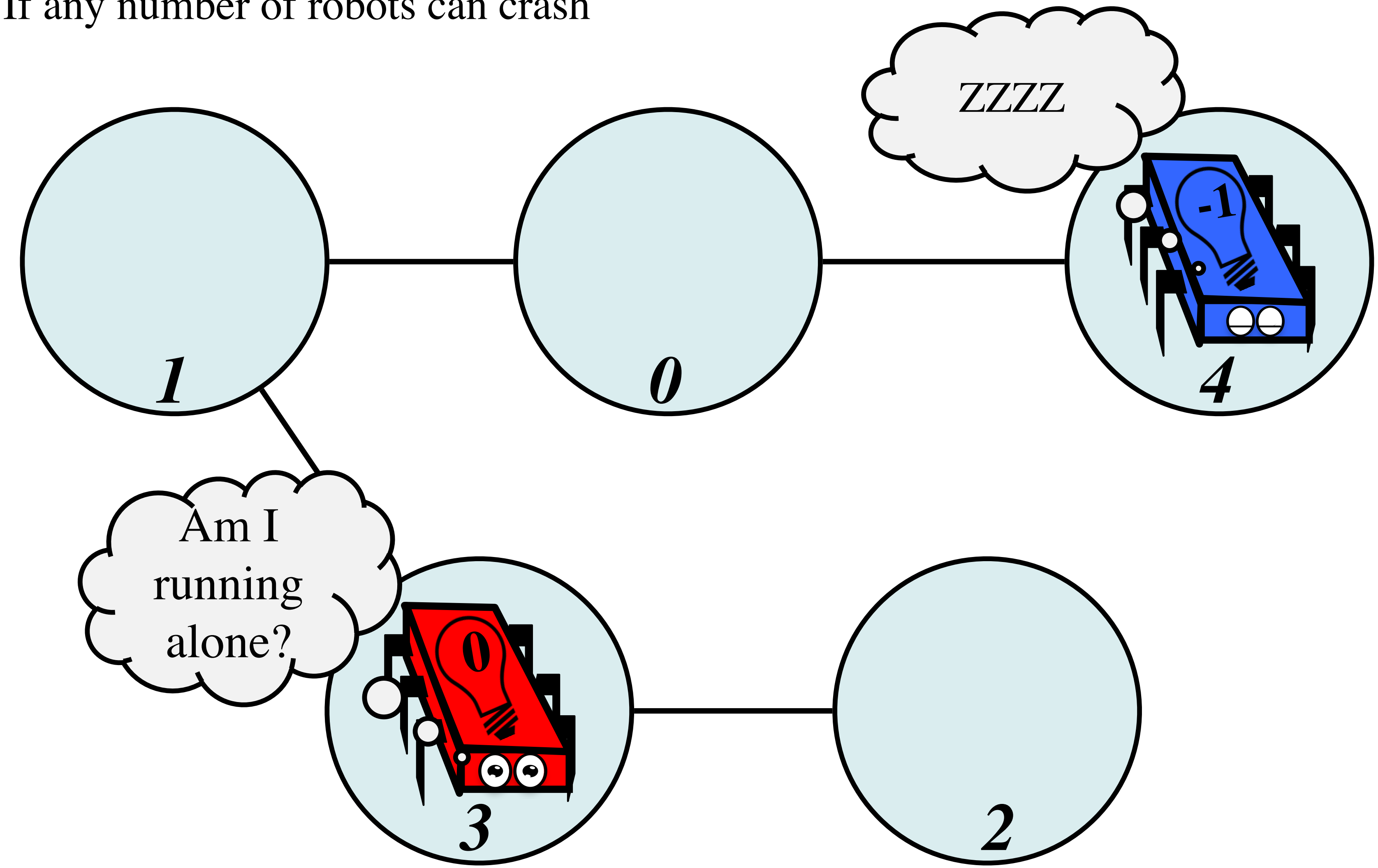
Wait-free algorithms

Tolerate any number of robots crashing or slow



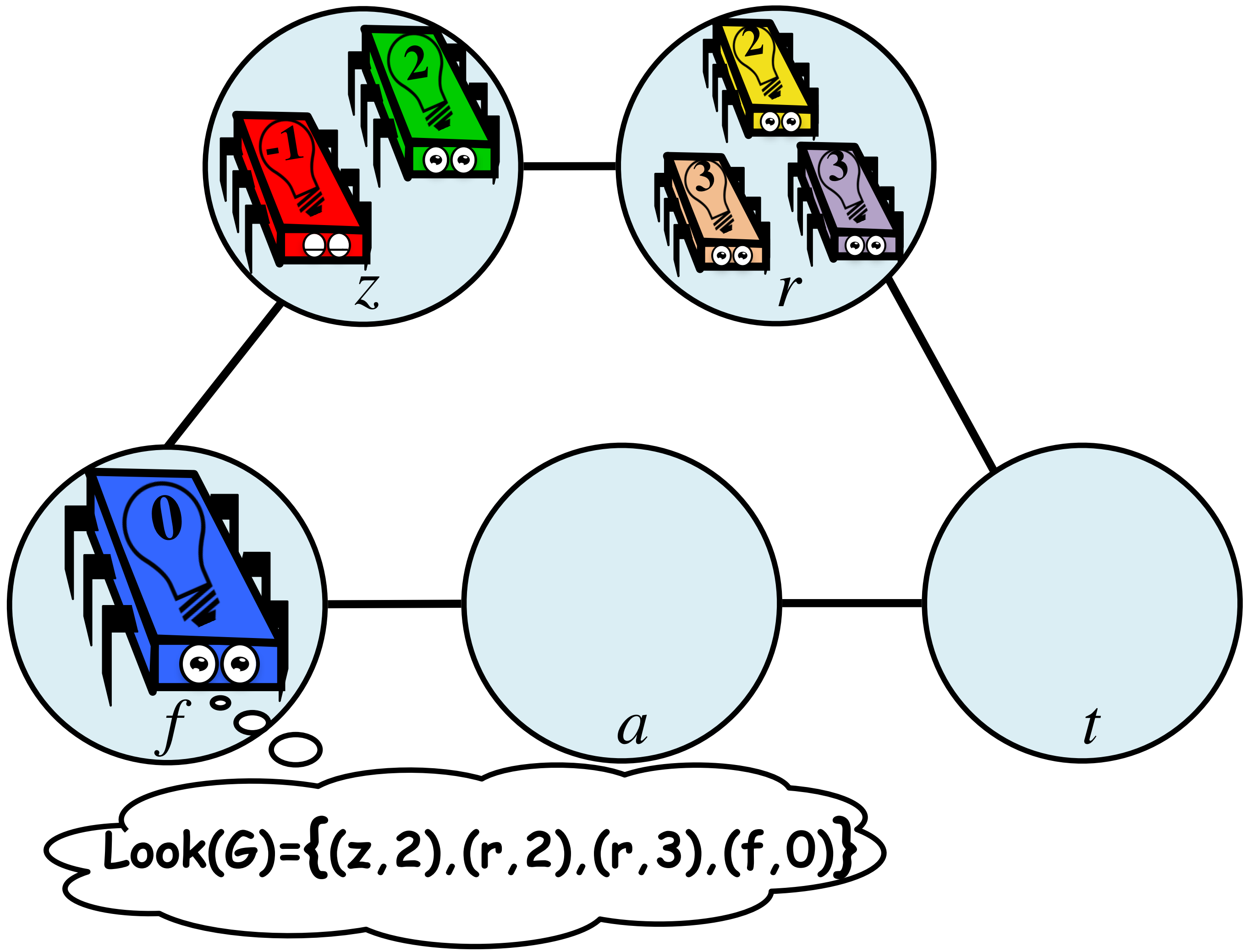
Solo executions

If any number of robots can crash



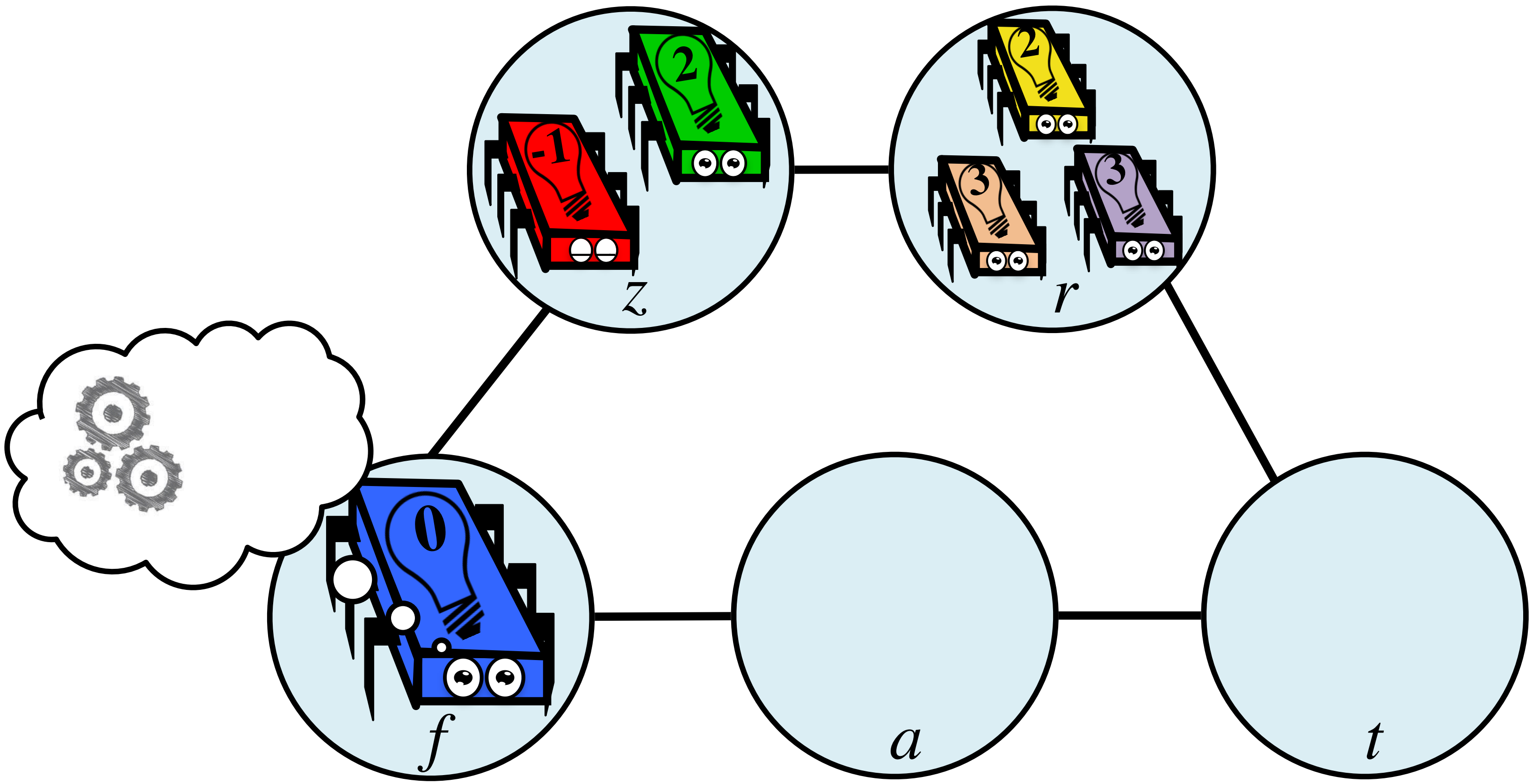
Look-Compute-Move Cycles

- **Look** :
- light=-1 means invisible



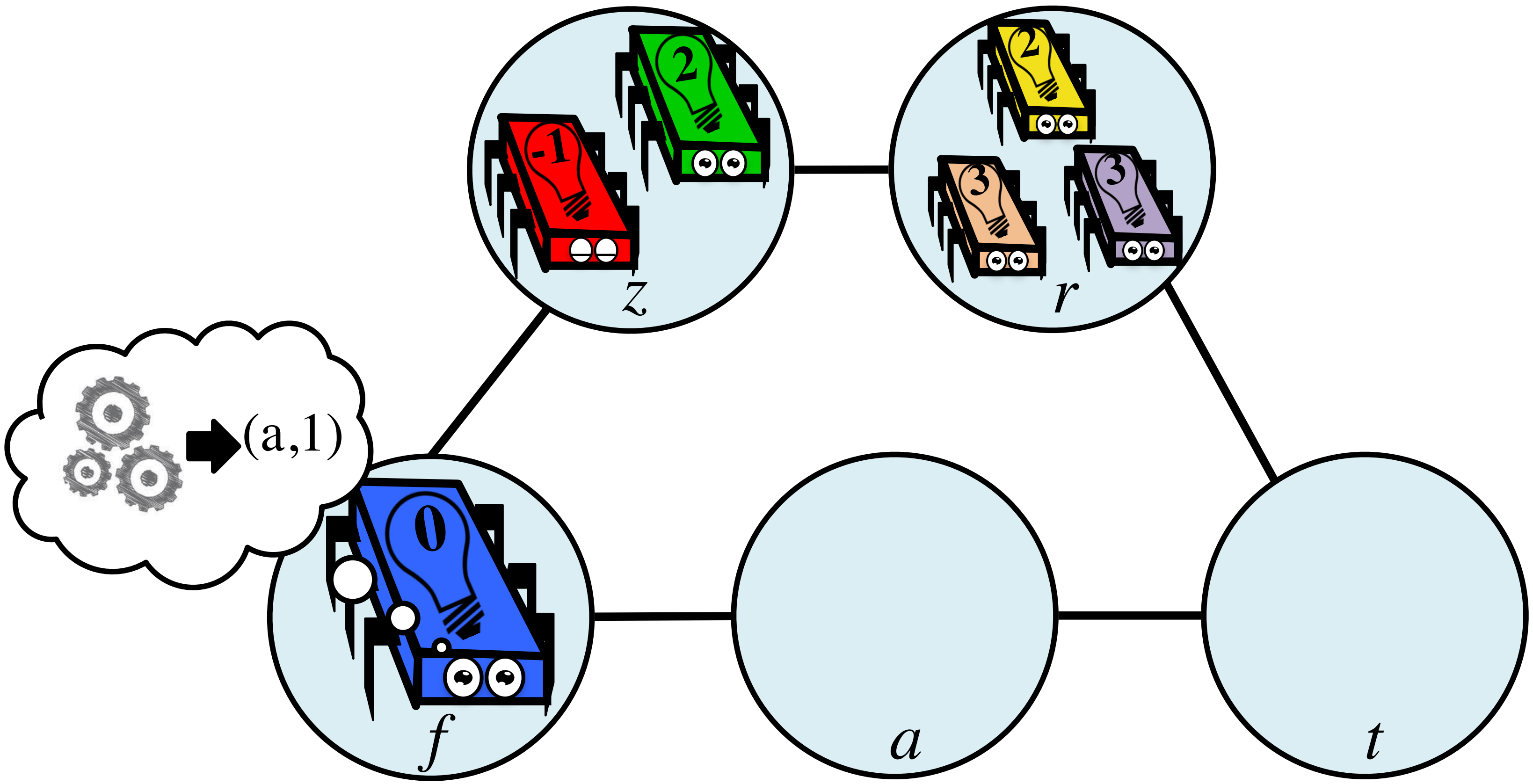
Look-Compute-Move Cycles

- **Compute :**



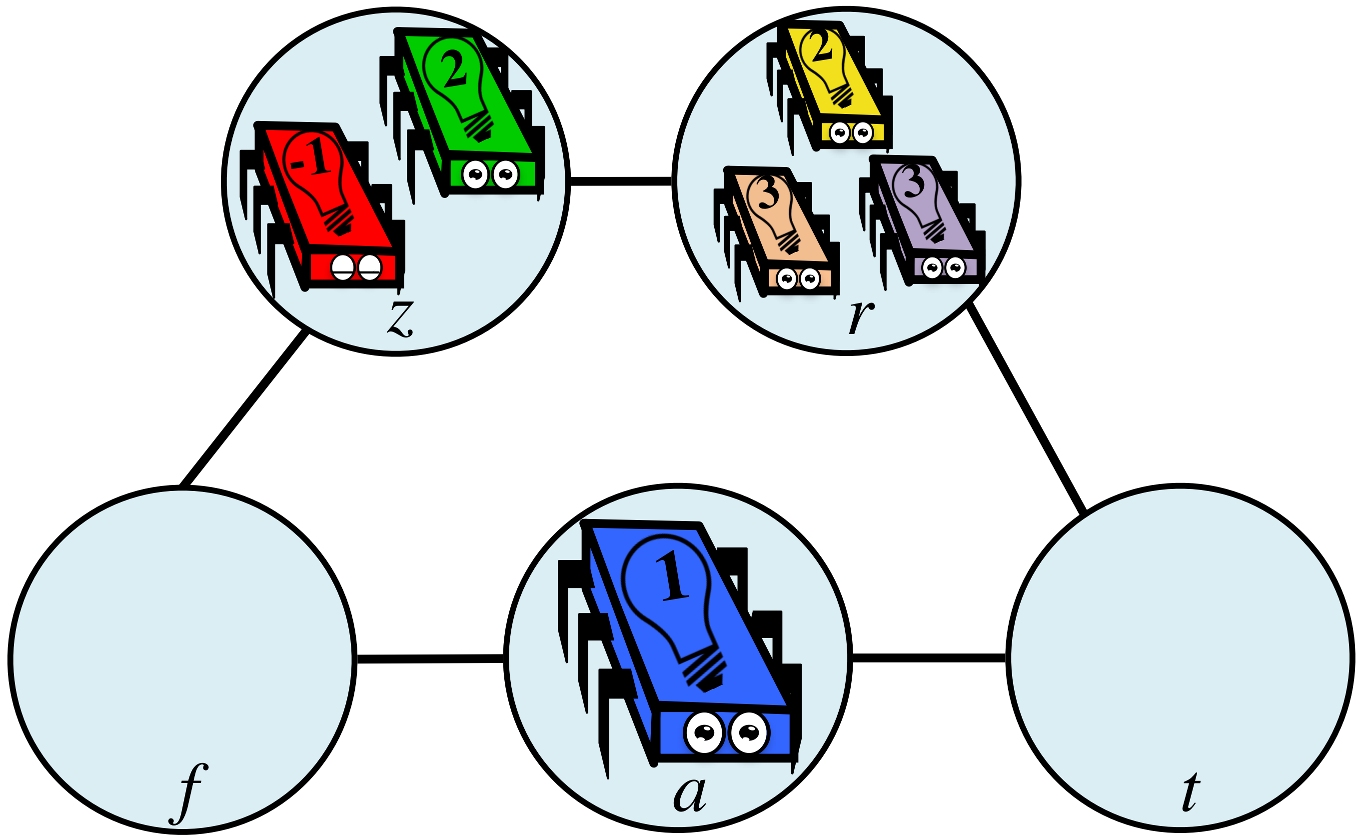
Look-Compute-Move Cycles

- **Compute** : neighbor vertex and new light



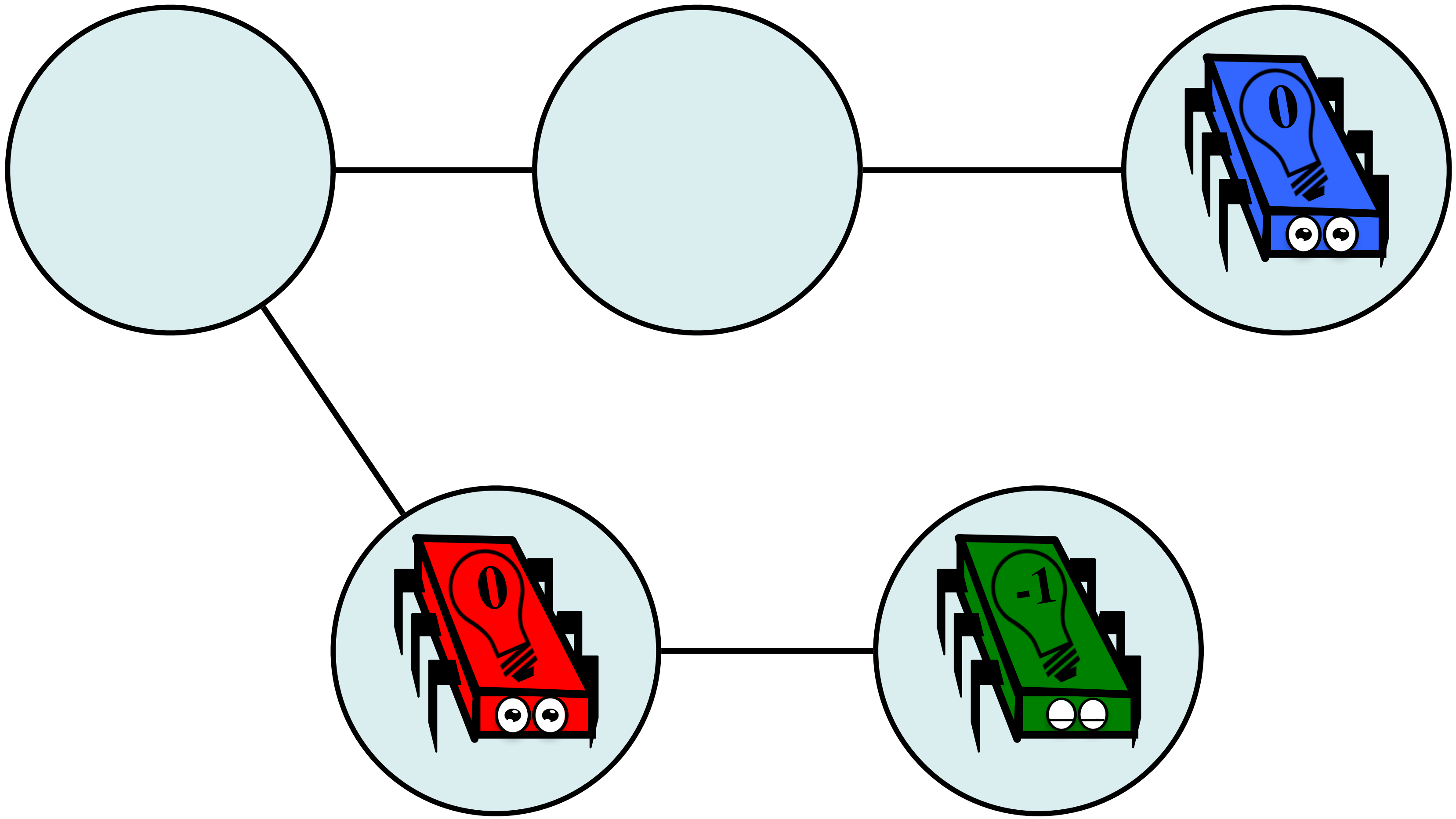
Look-Compute-Move Cycles

- Move(a,1) :

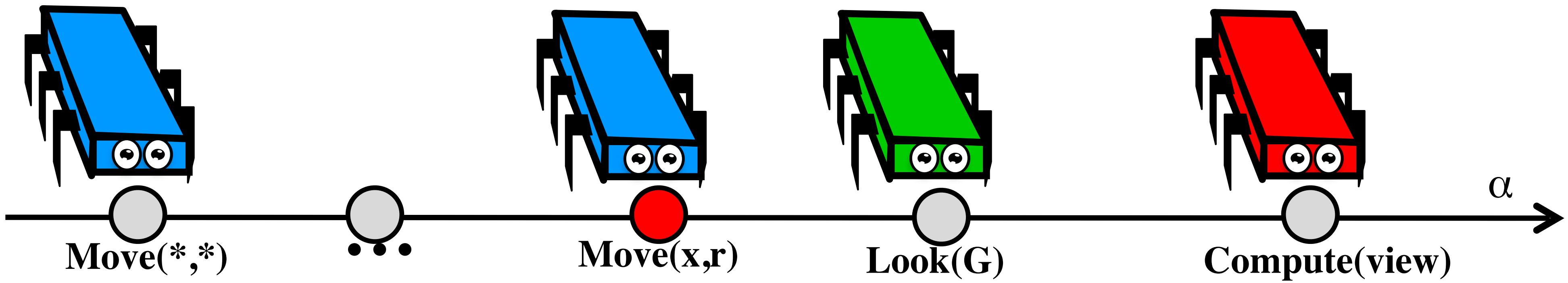


Execution

Arbitrary interleaving of *Look*, *Compute*, *Move* operations
by different robots



Execution



General Algorithm in ARL

Initial vertex

procedure ALGORITHM(G, v_i)
  $Move(v_i, 0)$
  $view_i \leftarrow \emptyset$
  while undecided **do**
    $view_i \leftarrow view_i \cdot \{\, Look(G) \,\}$
    $(v_i, r_i) \leftarrow Compute(view_i)$
    $Move(v_i, r_i)$
  end while
  return v_i
end procedure

General Algorithm in ARL

Initial vertex

Light, initially -1

procedure ALGORITHM(G, v_i)
    $Move(v_i, 0)$
    $view_i \leftarrow \emptyset$
    while undecided do
        $view_i \leftarrow view_i \cdot \{\, Look(G) \,\}$
        $(v_i, r_i) \leftarrow Compute(view_i)$
        $Move(v_i, r_i)$
    end while
    return v_i
end procedure

General Algorithm in ARL

Initial vertex

Appear!

```
procedure ALGORITHM(G, v_i)
    Move(v_i, 0)
    view_i ← ∅
    while undecided do
        view_i ← view_i · { Look(G) }
        (v_i, r_i) ← Compute(view_i)
        Move(v_i, r_i)
    end while
    return v_i
end procedure
```

General Algorithm in ARL

procedure ALGORITHM(G, v_i)
    $Move(v_i, 0)$
    $view_i \leftarrow \emptyset$
    while undecided **do**
        $view_i \leftarrow view_i \cdot \{ \, Look(G) \, \}$
        $(v_i, r_i) \leftarrow Compute(view_i)$
        $Move(v_i, r_i)$
    end while
    return v_i
end procedure

May remember the past

General Algorithm in ARL

procedure ALGORITHM(G, v_i)
  $Move(v_i, 0)$
  $view_i \leftarrow \emptyset$
  while undecided **do**
    $view_i \leftarrow view_i \cdot \{ Look(G) \}$
    $(v_i, r_i) \leftarrow Compute(view_i)$
    $Move(v_i, r_i)$
  end while
  return v_i
end procedure

May remember the past

Where to move and which light to emit

General Algorithm in ARL

procedure $\textsc{Algorithm}(G, v_i)$
    $Move(v_i, 0)$
    $view_i \leftarrow \emptyset$
    while undecided **do**
        $view_i \leftarrow view_i \cdot \{ Look(G) \}$
        $(v_i, r_i) \leftarrow Compute(view_i)$
        $Move(v_i, r_i)$
    end while
    return v_i
end procedure

Where to move and which light to emit

Do it!

General Algorithm in ARL

```
procedure ALGORITHM(G, v_i)
    Move(v_i, 0)
    view_i ← ∅
    while undecided do
        view_i ← view_i · { Look(G) }
        (v_i, r_i) ← Compute(view_i)
        Move(v_i, r_i)
    end while
    return v_i
end procedure
```

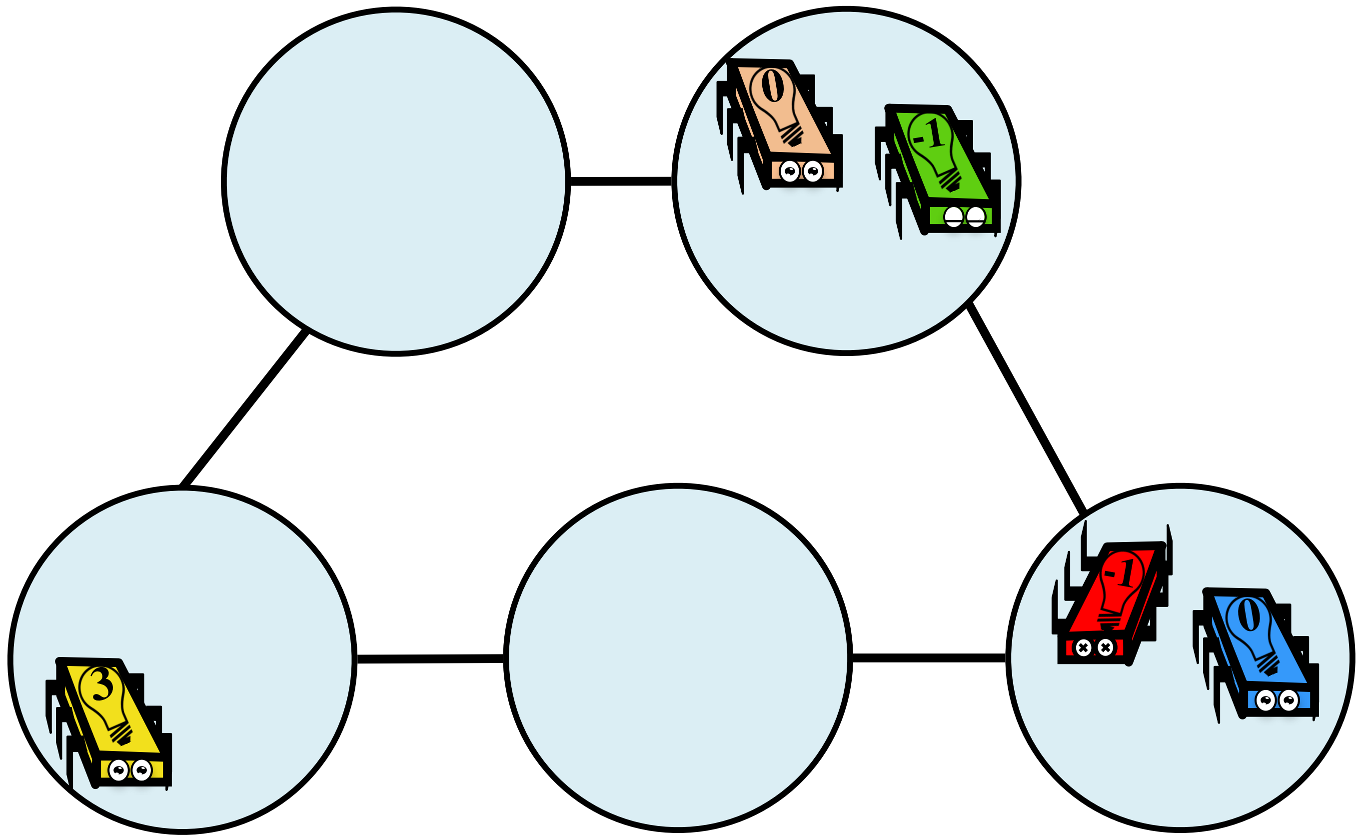
Decide where to stay

General Algorithm in ARL

```
procedure ALGORITHM(G, v_i)
    Move(v_i, 0)
    view_i ← ∅
    while undecided do
        view_i ← ~~view_i~~ · { Look(G) }
        (v_i, r_i) ← Compute(view_i)
        Move(v_i, r_i)
    end while
    return v_i
end procedure
```

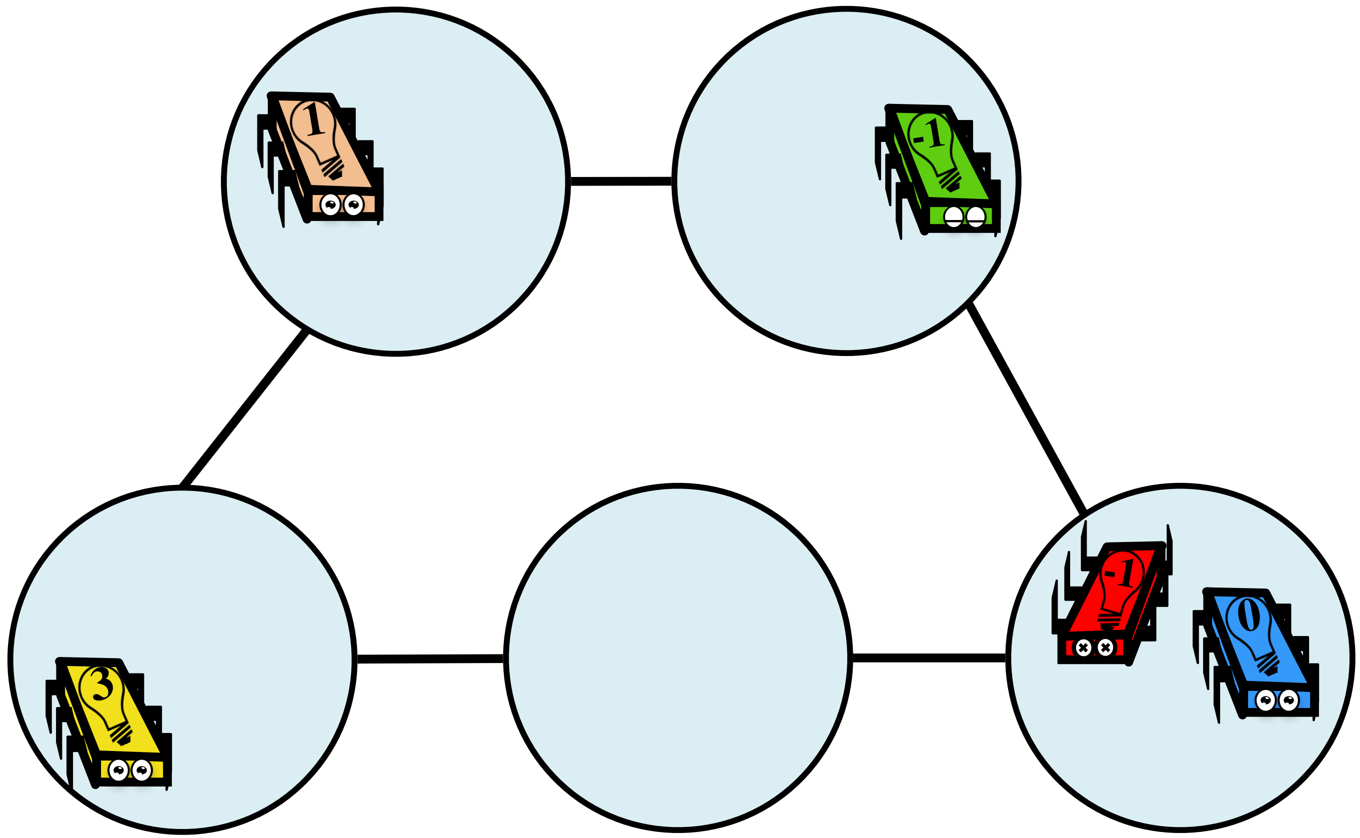
Gathering

- Move all (correct) robots to the same vertex



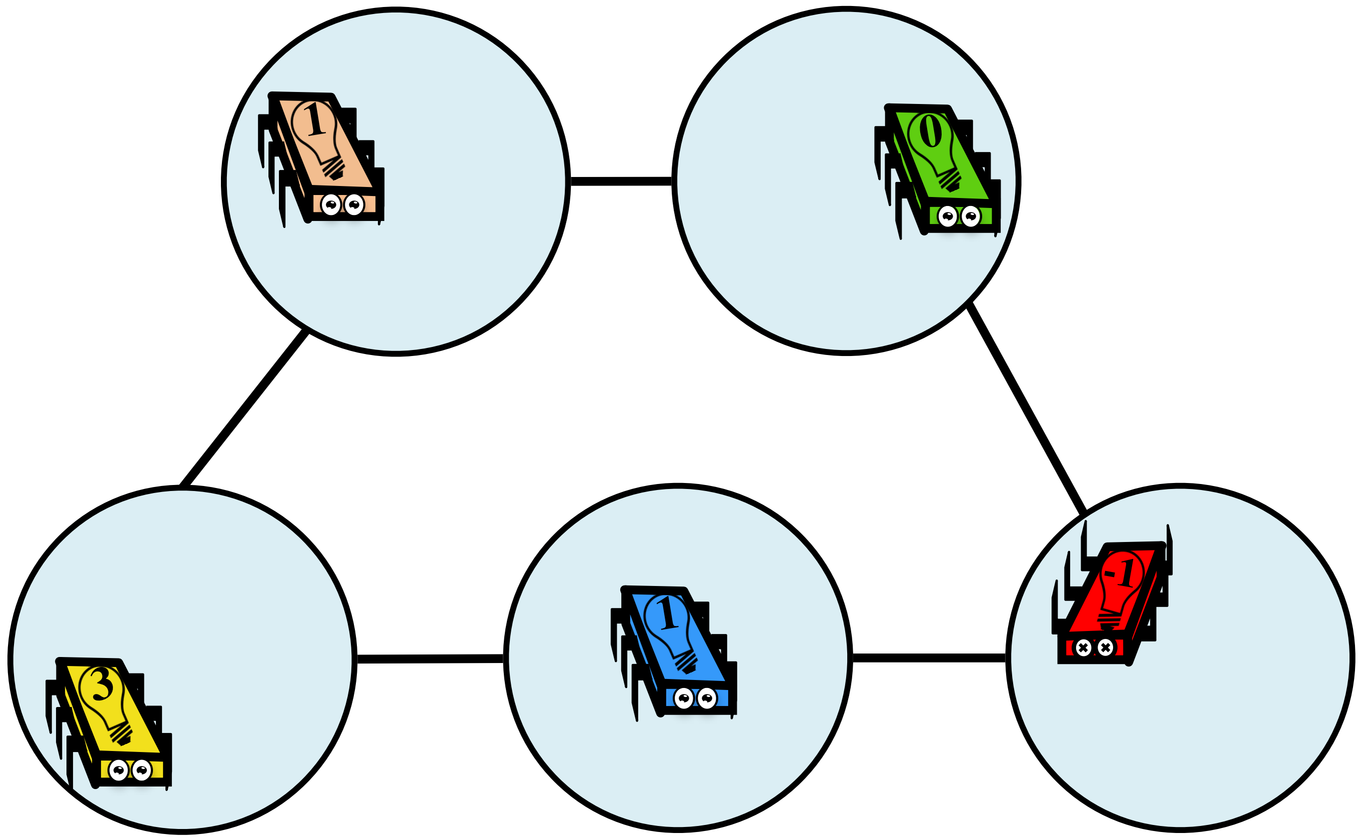
Gathering

- Group all correct Robots on the same vertex



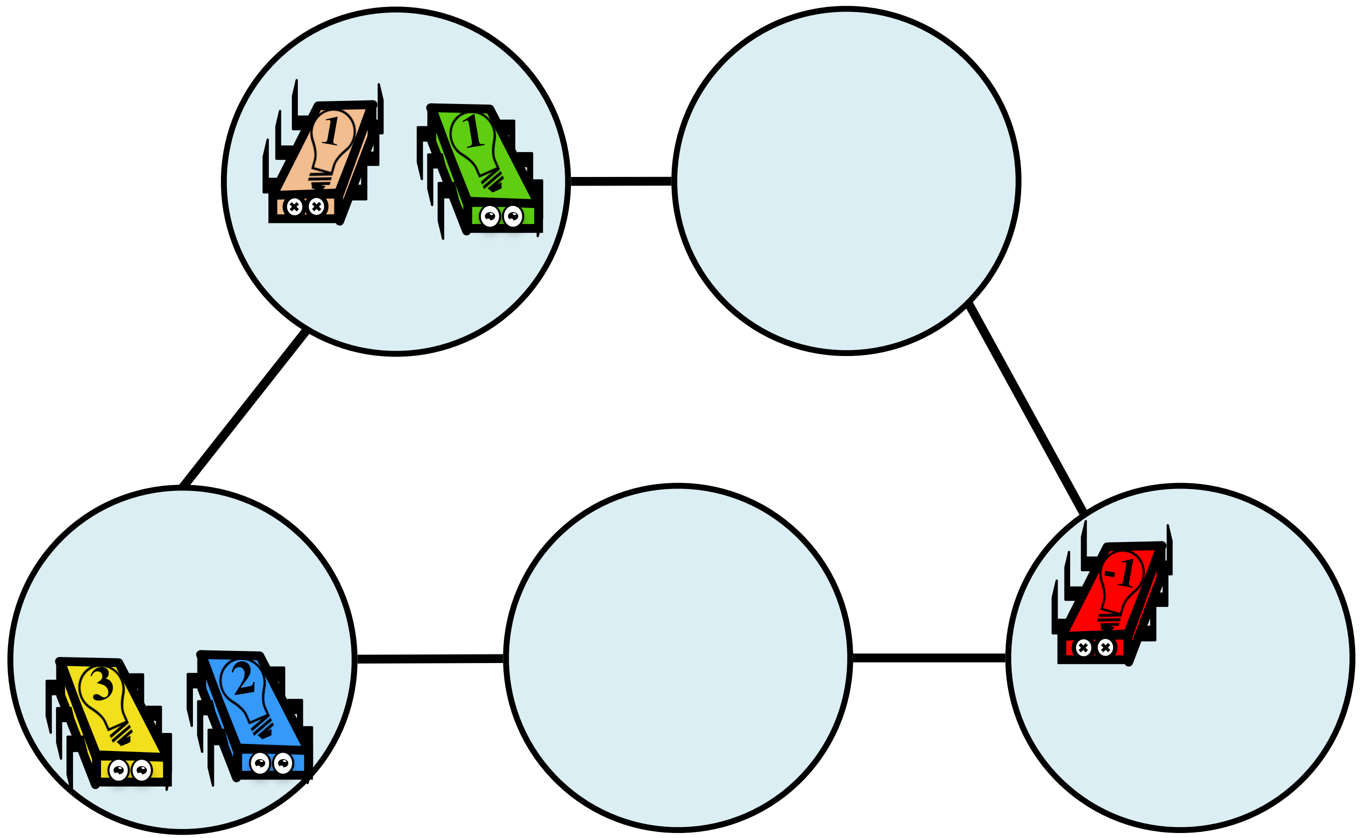
Gathering

- Group all correct Robots on the same vertex



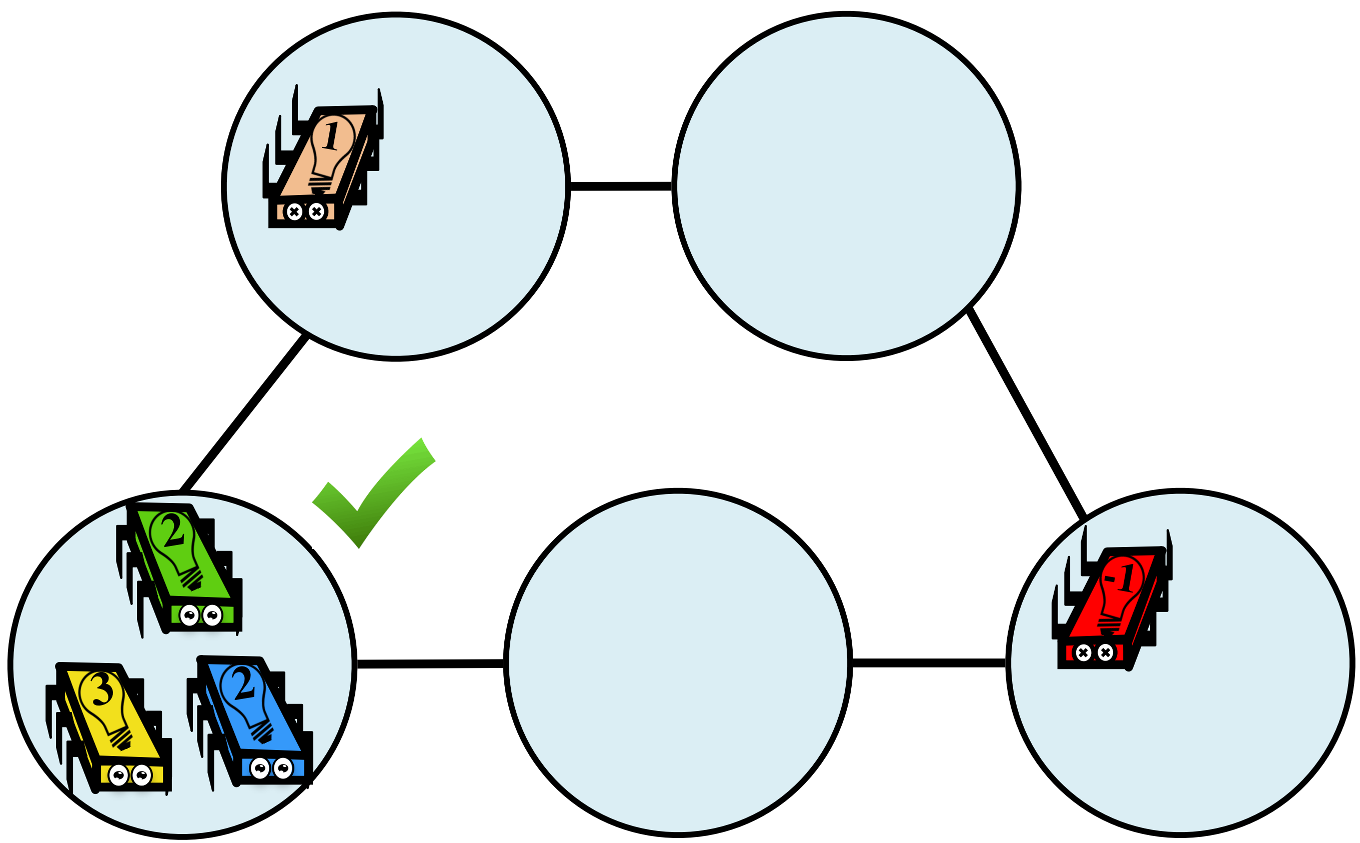
Gathering

- Group all correct Robots on the same vertex



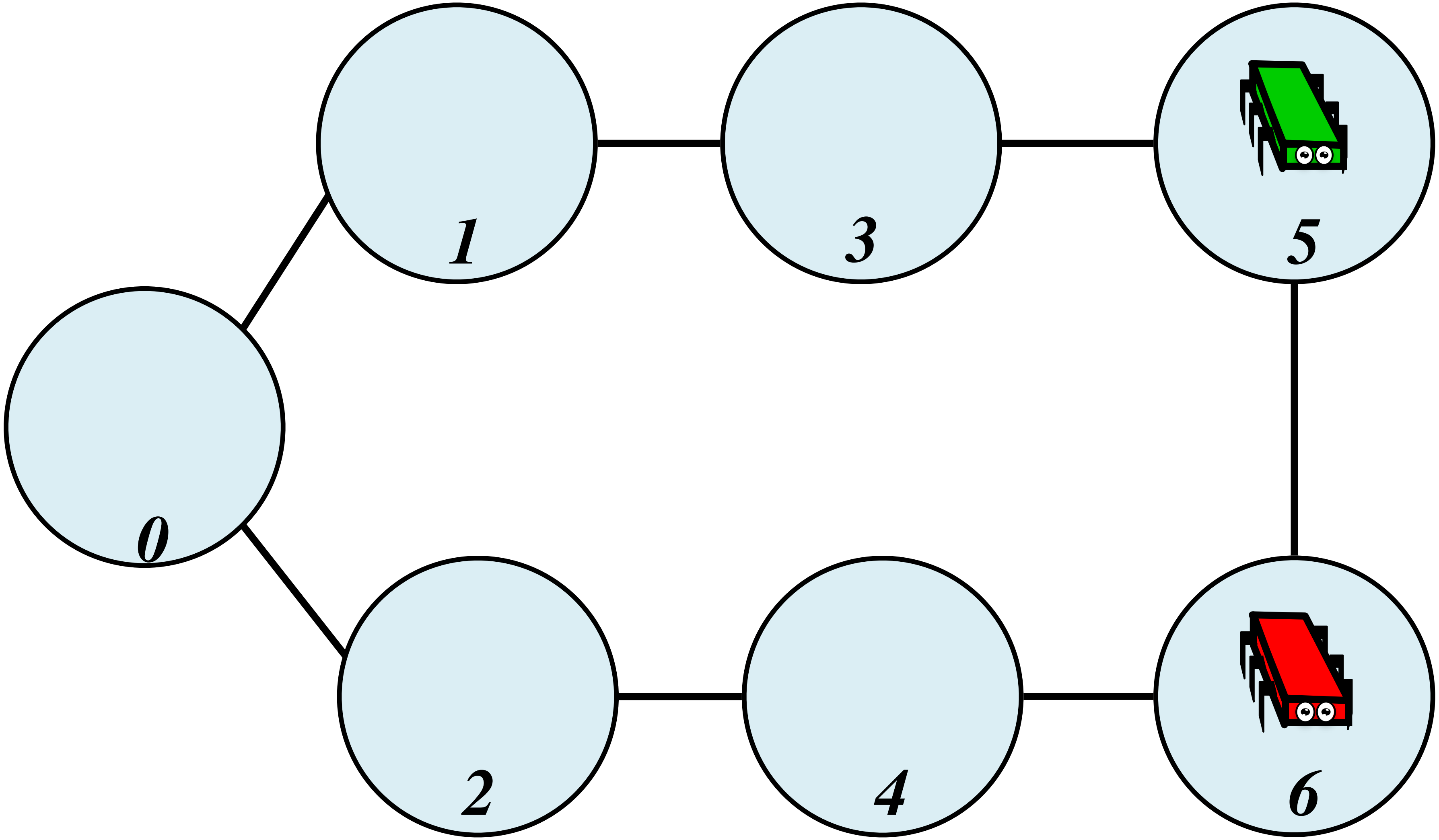
Gathering

- Group all correct Robots on the same vertex



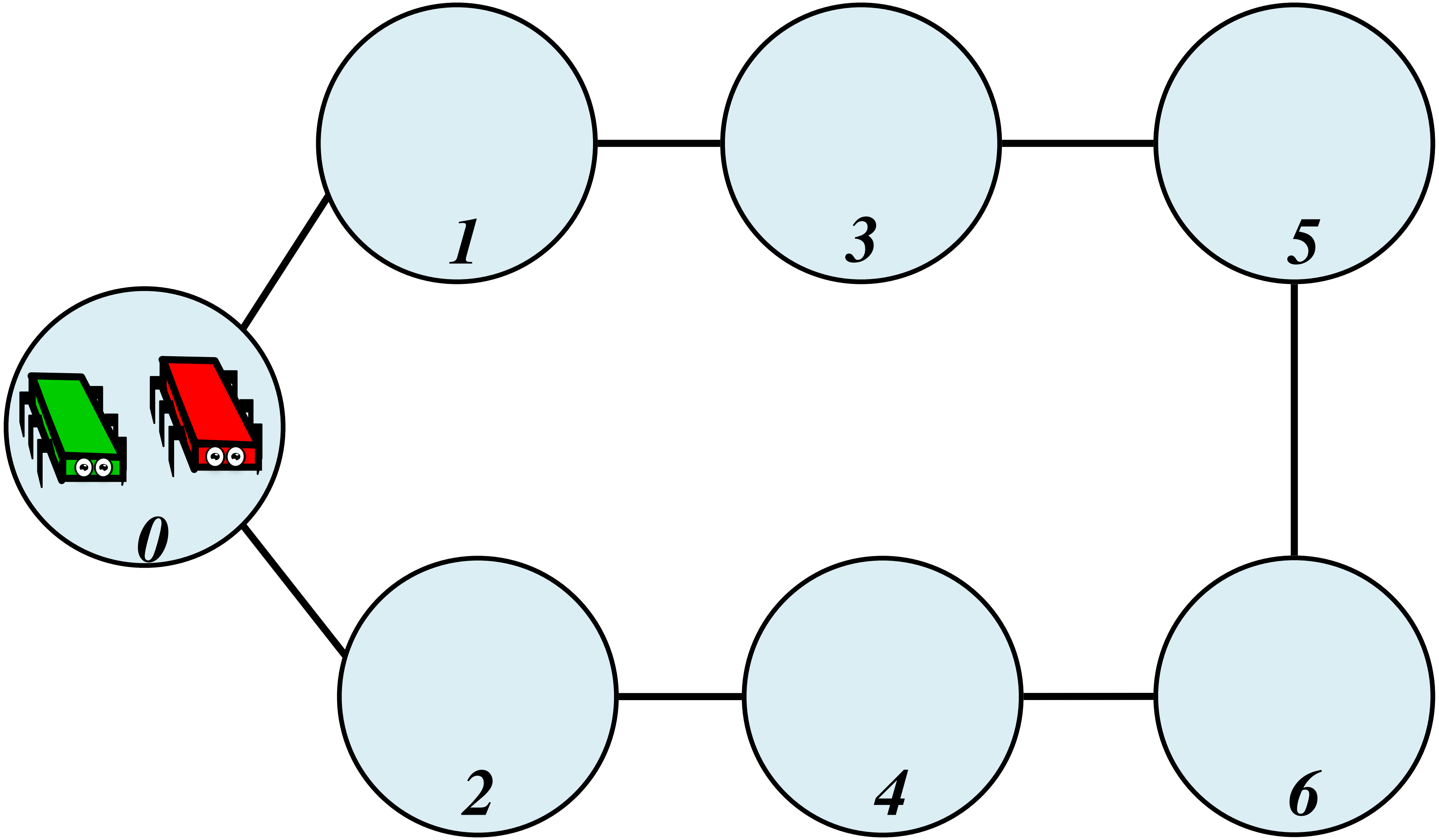
Gathering

- Gathering is trivial if OK to move to a fixed vertex



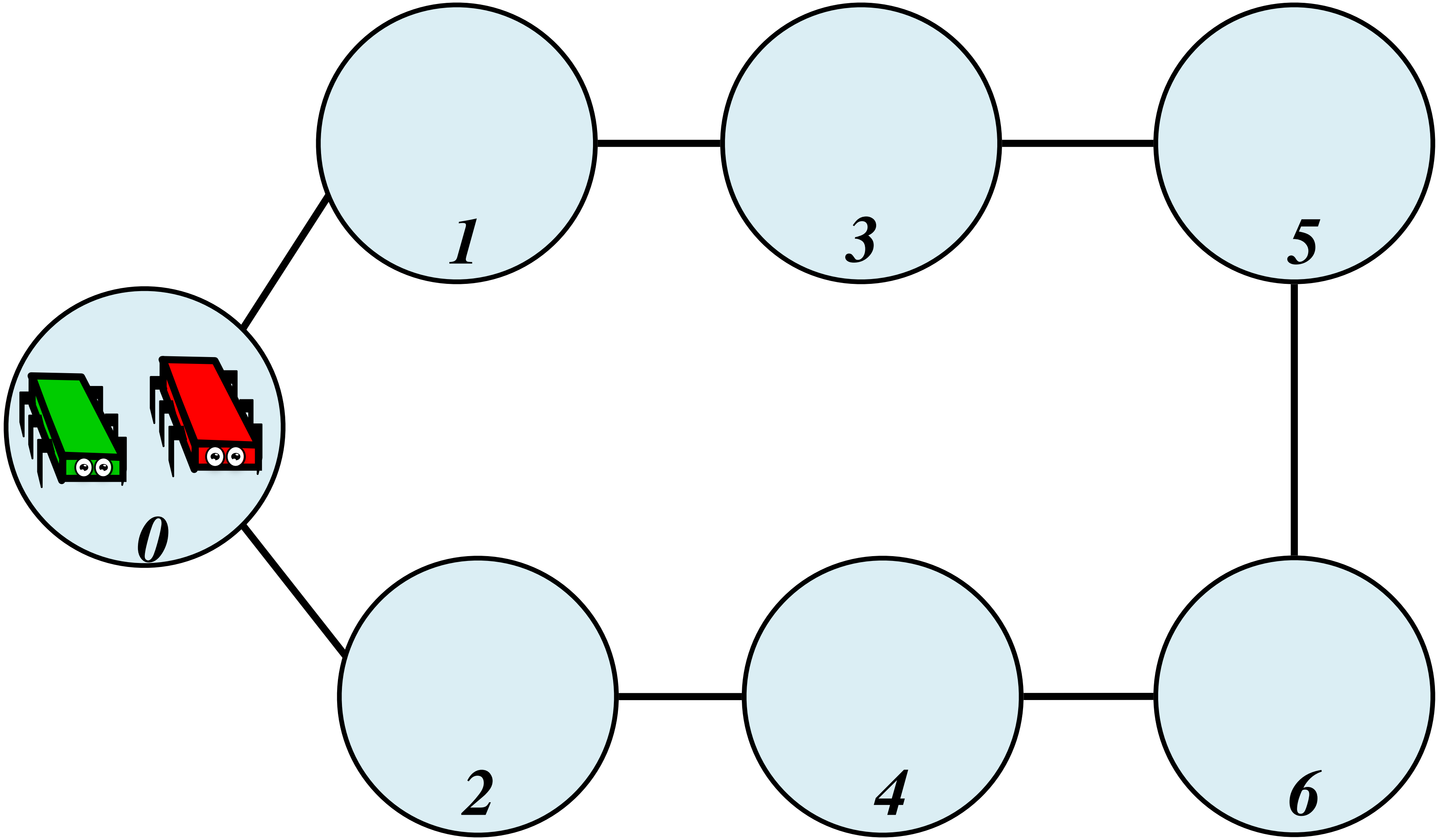
Gathering

- if there is a common labeling and there are no restrictions.



Gathering

➢Require that final location not fixed in advance.



Gathering definition

- ***Termination:*** Every correct robot decides a vertex of G.

Gathering definition

- ***Termination:*** Every correct robot decides a vertex of G.
- ***Validity:*** If all robots start on the same vertex v, then every decided vertex is v.

Gathering definition

- ***Termination:*** Every correct robot decides a vertex of G.
- ***Validity:*** If all robots start on the same vertex v, then every decided vertex is v. [Prevents trivial]
- ***Agreement:*** All decided vertices are the same.

Gathering vs convergence

- ***Termination:*** Every correct robot decides a vertex of G.
- If we remove the termination requirement -> convergence problem

gathering vs edge-gathering

- *Agreement: All decided vertices are the same.*
- A different way of weakening -> get close to each other, instead of to the same vertex

Edge-Gathering

- ***Termination:*** Every correct robot decides a vertex of G.

Edge-Gathering

- ***Termination:*** Every correct robot decides a vertex of G.

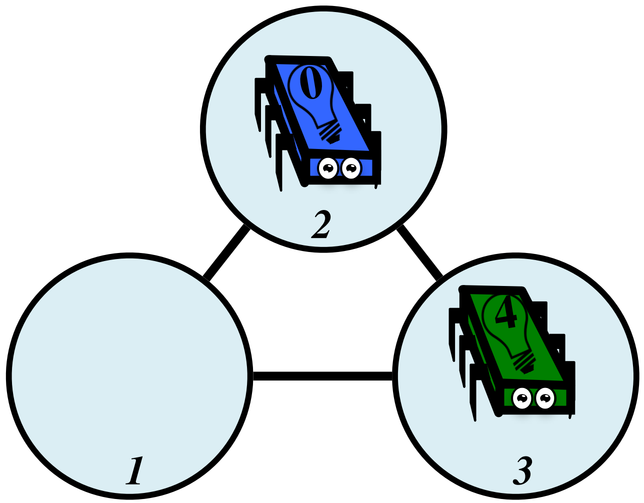
- ***Validity:*** If all robots start on the same vertex v then every decided vertex is v. If the initial vertex of robots cover an edge e, then every decided vertex is a vertex of e.

Edge-Gathering

- ***Termination:*** Every correct robot decides a vertex of G.
- ***Validity:*** If all robots start on the same vertex v then every decided vertex is v. If the initial vertex of robots cover an edge e, then every decided vertex is a vertex of e
- ***Edge-Agreement:*** All decided vertices belong to the same edge.

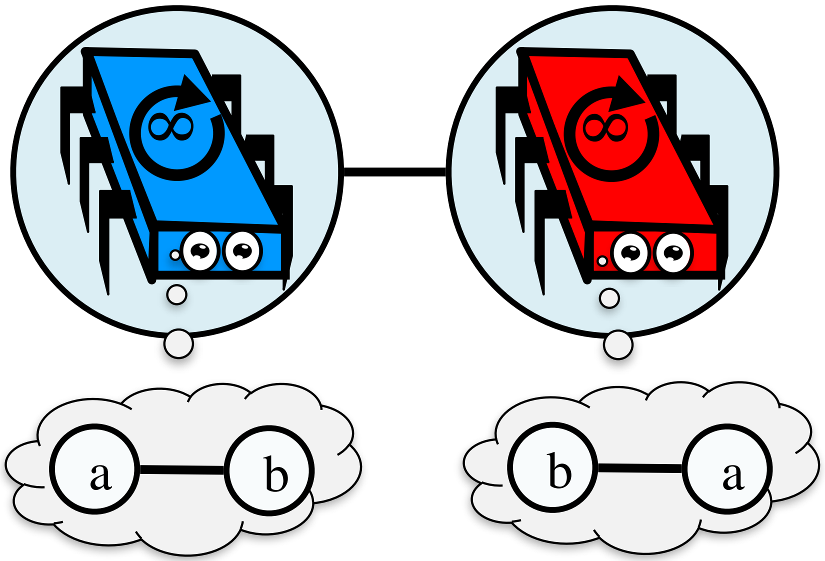
Gathering Problems Summary

- Convergence (No termination)
- Gathering
    - Edge-Gathering
- Also 1-Gathering (see paper)



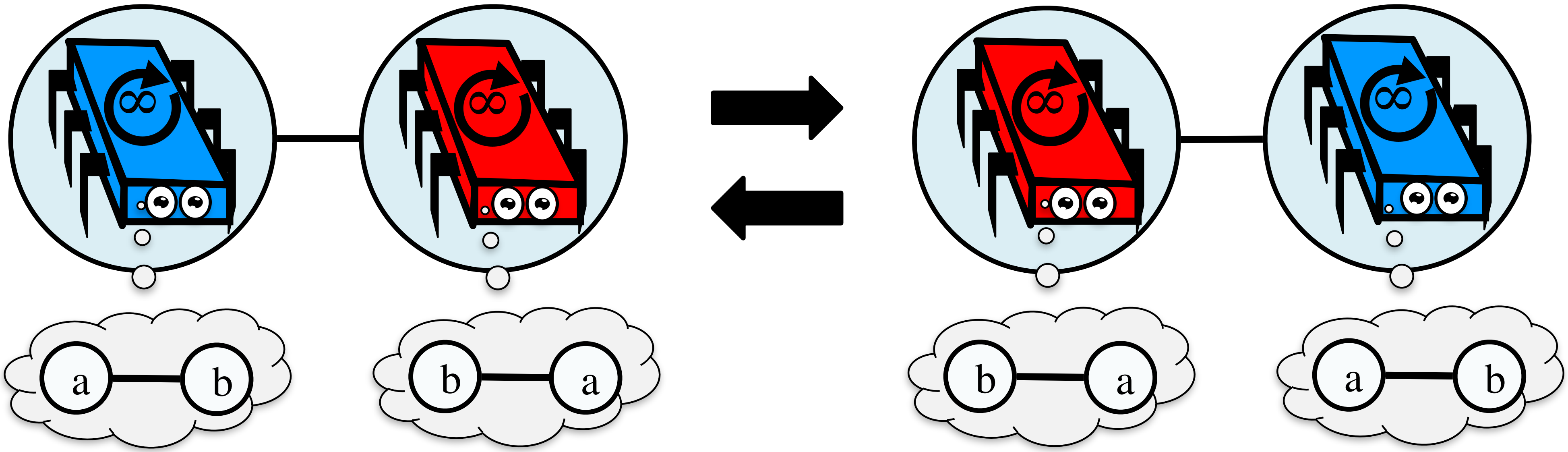
Convergence without labeling

- It is impossible to achieve convergence in a general Graph without a common labeling, if the robots are identical



Convergence without labeling

- It's impossible to achieve convergence in a general Graph without a common labeling, if the robots are identical



Convergence with labeling (no lights)

- There are various algorithms to solve convergence with a common labeling of G (with identical robots)
- Many interesting questions, we don't focus on them here

Function TrivialConvergence(v_i, G)

```
Move(v_i)
loop
    view_i ← Look(G)
    S_i ← { v_j ∈ view_i | v_j ≠ v_i }
    if S_i ≠ ∅ then
        canonical_i ← the vertex with minimum label in G.
        v_i ← some closest vertex to canonical_i.
    end if
    Move(v_i)
end loop
```

Convergence with labeling, no lights

- There are various algorithms to solve convergence with a common labeling of G
- Many interesting questions, but we don't consider them here

Function TrivialConvergence(v_i, G)

```
Move(v_i)
loop
    view_i ← Look(G)
    S_i ← { v_j ∈ view_i | v_j ≠ v_i }
    if S_i ≠ ∅ then
        canonical_i ← the vertex with minimum label in G.
        v_i ← some closest vertex to canonical_i.
    end if
    Move(v_i)
end loop
```

Never decides where to stay!

Algorithm: Convergence

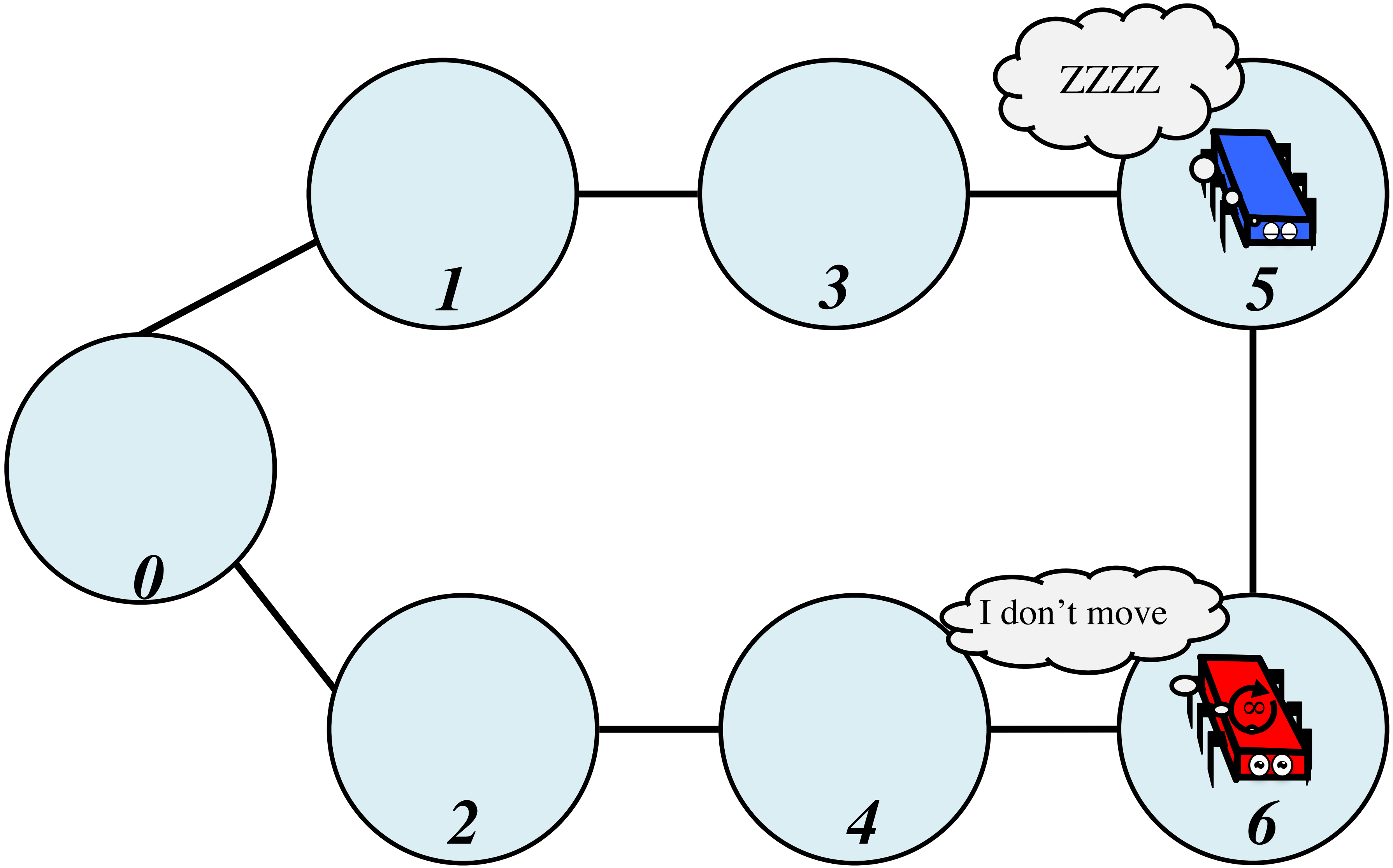
- Trivial algorithm to solve convergence problem

Function TrivialConvergence(v_i, G)

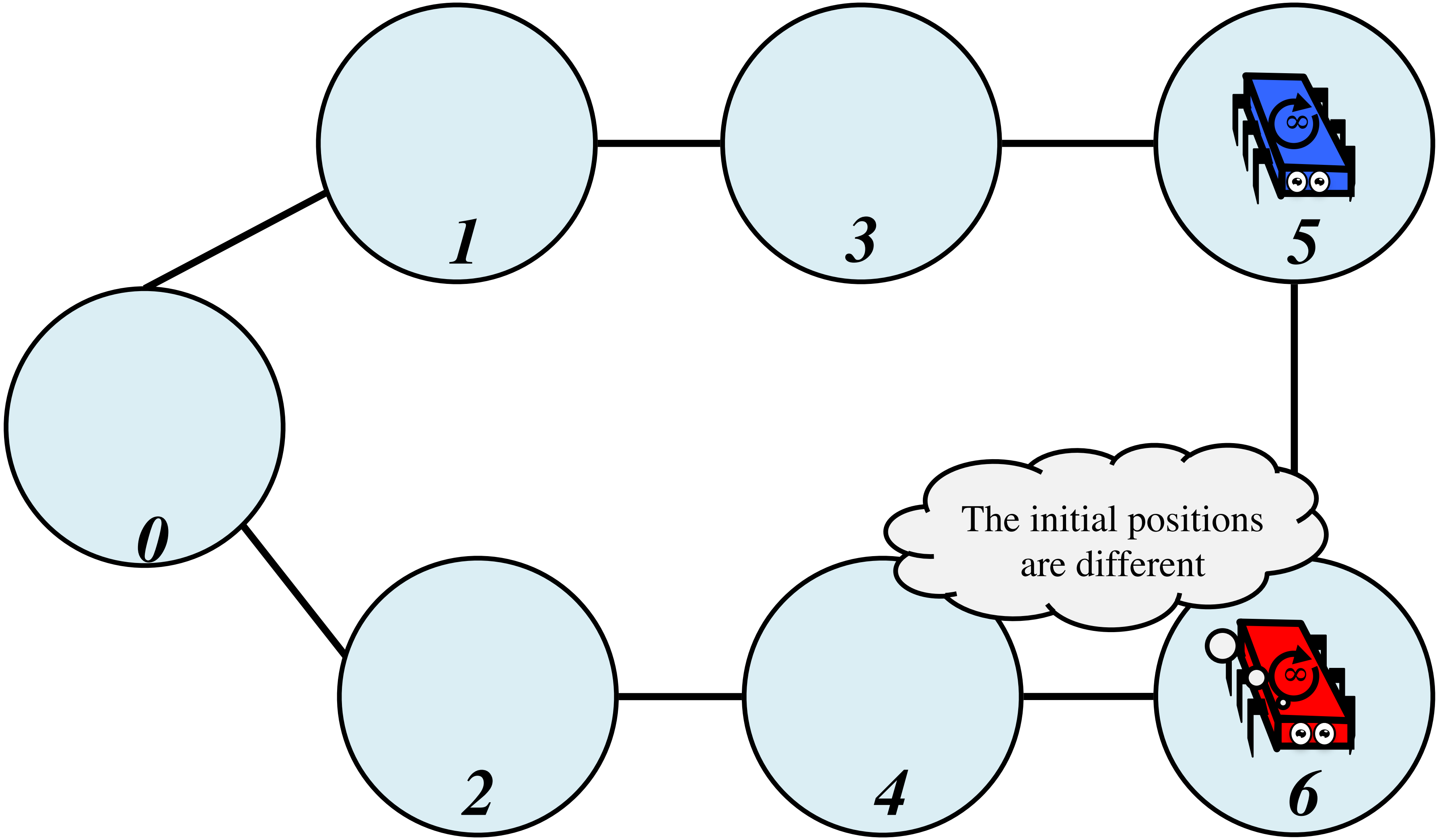
Move(v_i)
loop
    $view_i \leftarrow$ **Look**(G)
    $S_i \leftarrow \{ v_j \in view_i \mid v_j \neq v_i \}$
    if $S_i \neq \varnothing$ **then**
        $canonical_i \leftarrow$ the vertex with minimum label in G.
        $v_i \leftarrow$ some closest vertex to $canonical_i$.
    end if
    $Move(v_i)$
end loop

This is one of many possibilities to solve convergence.

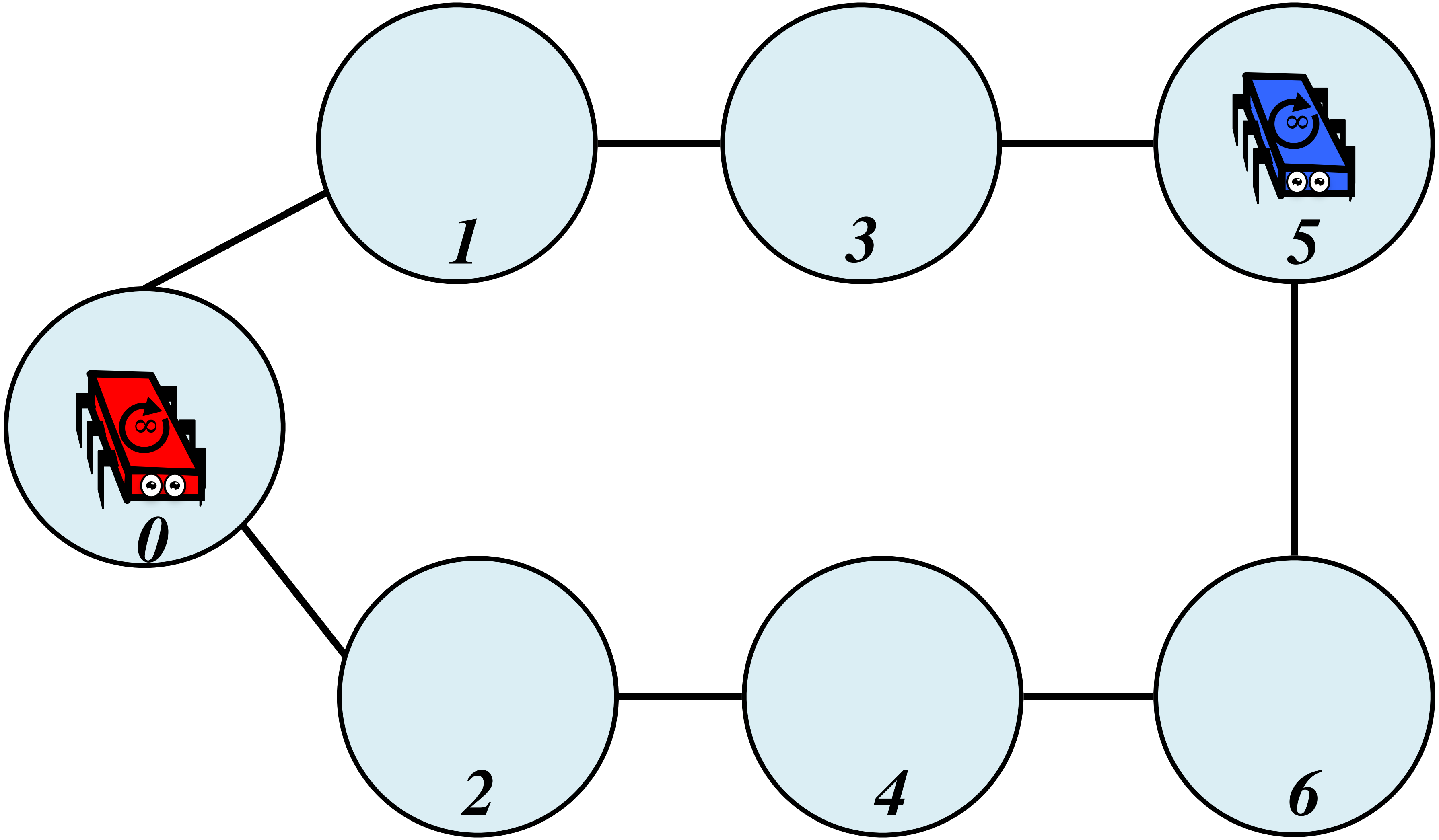
Idea of Convergence Algorithm



Idea of Convergence Algorithm



Idea of Convergence Algorithm

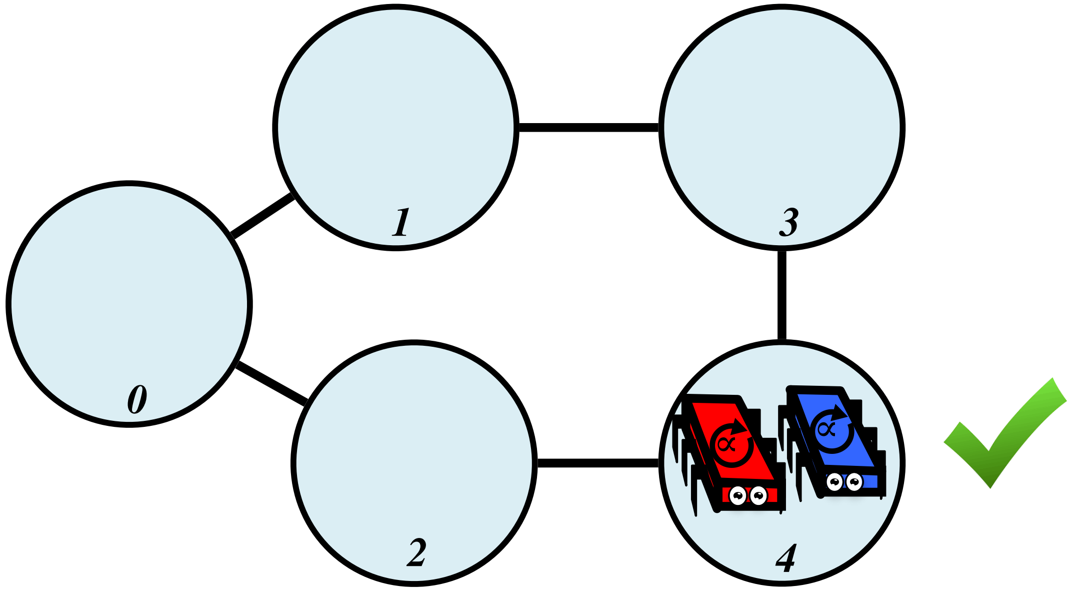


- As the initial positions were different the red robot can go to a predefined vertex

Proof of Convergence Algorithm

- *Stabilization:*

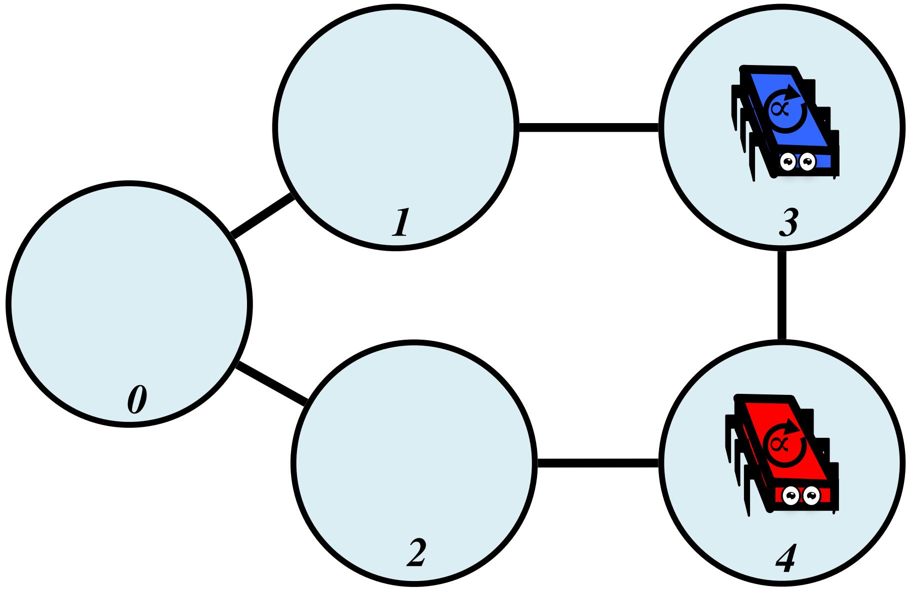
  Case 1: The initial positions are the same



Proof of Convergence Algorithm

- ***Stabilization:***

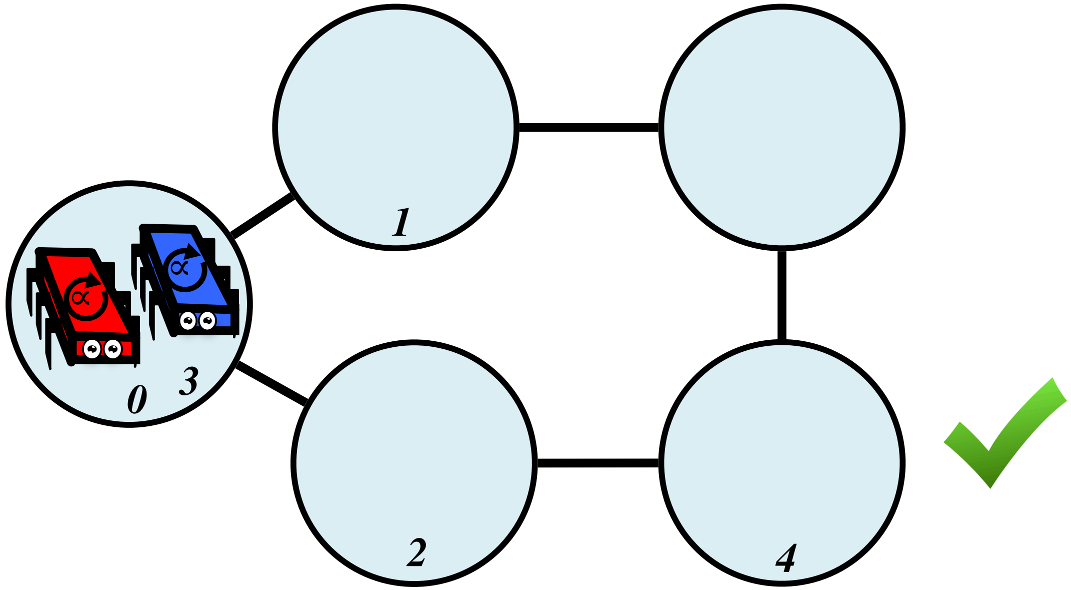
  Case 2: There are at least two different initial positions.



Proof of Convergence Algorithm

- ***Stabilization:***

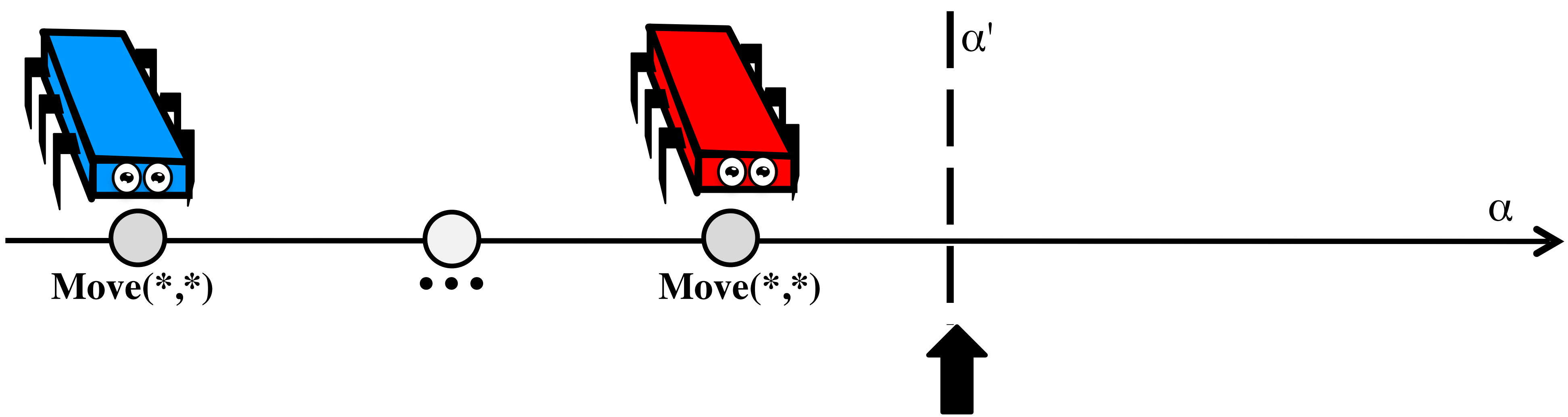
  Case 2:



Proof of Convergence Algorithm

- ***Agreement:***

By contradiction: Suppose the final position of both robots is not the same

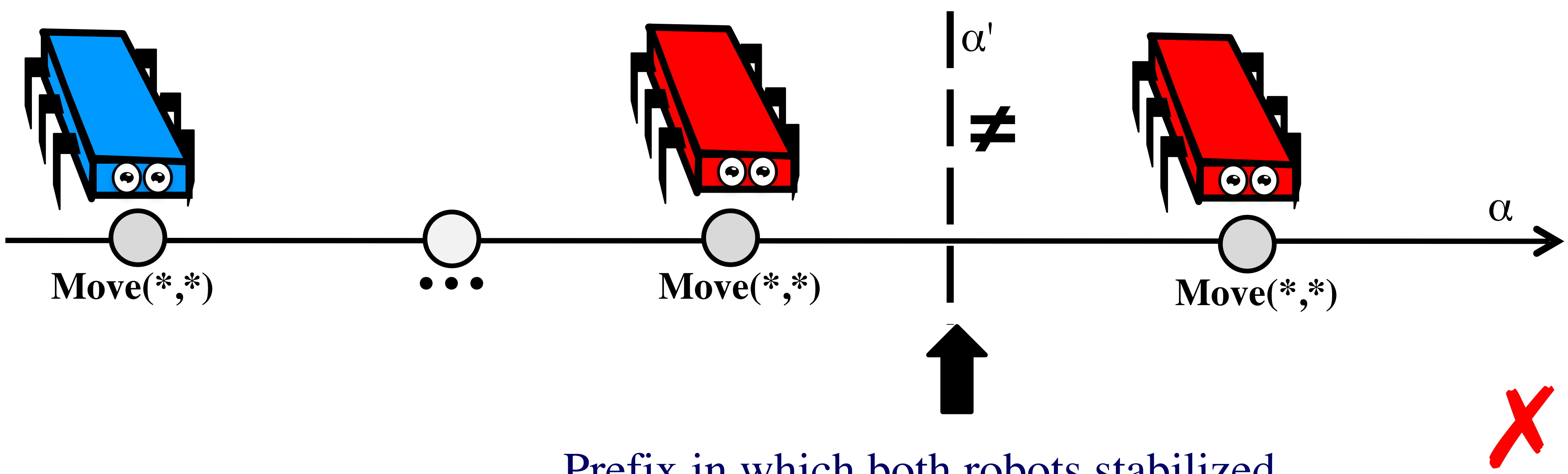


Prefix in which both robots stabilized

Proof of Convergence Algorithm

- ***Agreement:***

By contradiction: Suppose the final position of both robots is not the same



Convergence is possible, what about gathering?

Can a robot decide when to stop ?

Gathering is Impossible in ARL

First result:

For every every G with at least 2 vertices, there is no algorithm that solves gathering on G for *n* robots in *strong* ARL, and even if only one robot can fail.

Gathering is Impossible in ARL

First result:

For every every G with at least 2 vertices, there is no algorithm that solves gathering on G for *n* robots in *strong* ARL, and even if only one robot can fail.

Strong version of the ARL, robots are non-oblivious, non-anonymous, use an unbounded number of light colors and share a labelling (or orientation) of G

Gathering in ARL

Notice: gathering is POSSIBLE if all robots are present initially

Gathering is Impossible in ARL

Proof by reduction to consensus in the read/write shared-memory model of FLP

Gathering is Impossible in ARL

Any ARL algorithm can be simulated in the WFSM model in a wait-free manner

Proof by reduction to consensus in the read/write shared-memory model of FLP

The simulation

ARL Model

```
procedure GATHERING(G, v_i)
    Move(v_i, 0)
    view_i ← ∅
    while undecided do
        view_i ← view_i · { Look(G) }
        (v_i, r_i) ← Compute(view_i)
        Move(v_i, r_i)
    end while
    return v_i
end procedure
```

The simulation

ARL Model

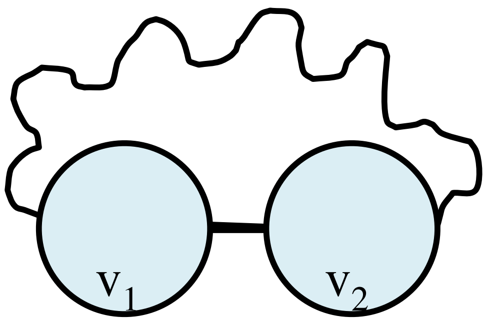
```
procedure GATHERING(G, v_i)
    Move(v_i, 0)
    view_i ← ∅
    while undecided do
        view_i ← view_i · { Look(G) }
        (v_i, r_i) ← Compute(view_i)
        Move(v_i, r_i)
    end while
    return v_i
end procedure
```

The simulation

ARL Model

```
procedure GATHERING(G, v_i)
    Move(v_i, 0)
    view_i ← ∅
    while undecided do
        view_i ← view_i · { Look(G) }
        (v_i, r_i) ← Compute(view_i)
        Move(v_i, r_i)
    end while
    return v_i
end procedure
```

G(V,E) =

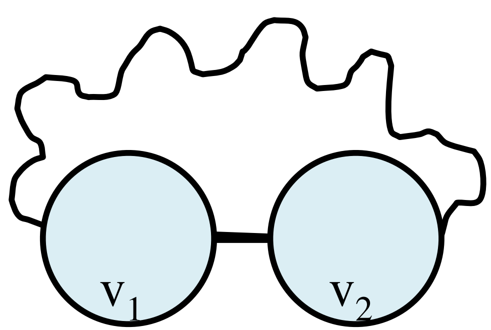


The simulation

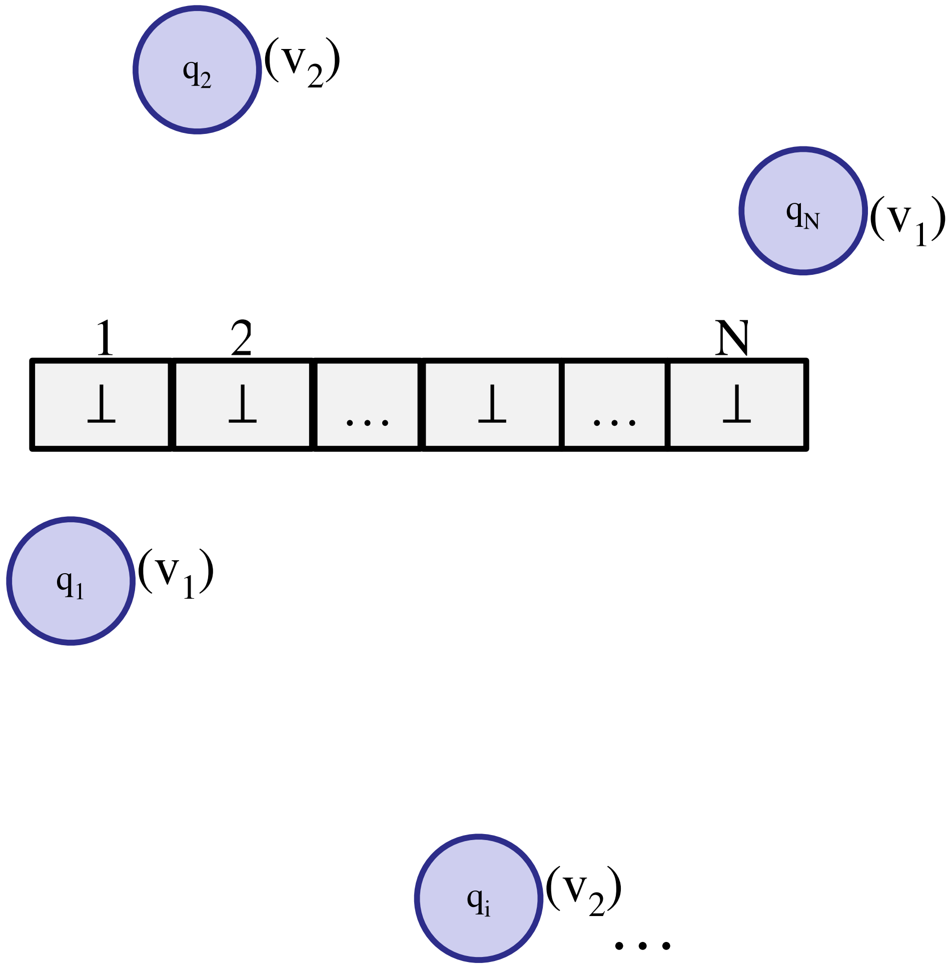
ARL Model

```
procedure GATHERING(G, v_i)
    Move(v_i, 0)
    view_i ← ∅
    while undecided do
        view_i ← view_i · { Look(G) }
        (v_i, r_i) ← Compute(view_i)
        Move(v_i, r_i)
    end while
    return v_i
end procedure
```

G(V,E) =



WFSM Model

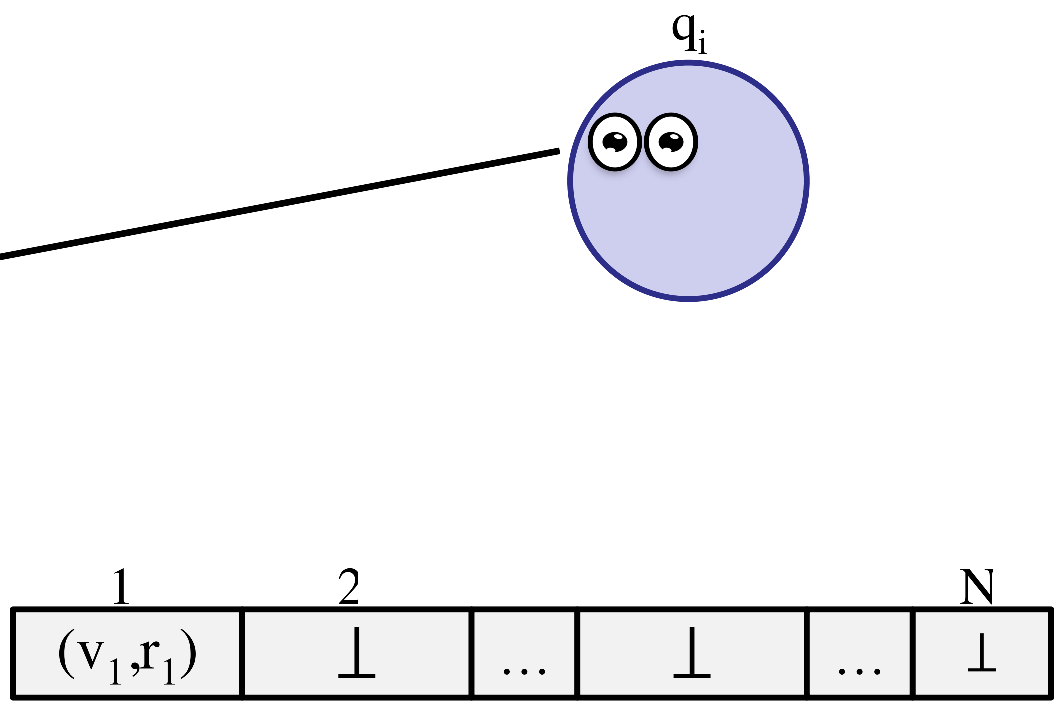


Gathering is Impossible in ARL

ARL Model

```
procedure GATHERING(G, v_i)
    Move(v_i, 0)
    view_i ← ∅
    while undecided do
        view_i ← view_i · { Look(G) }
        (v_i, r_i) ← Compute(view_i)
        Move(v_i, r_i)
    end while
    return v_i
end procedure
```

WFSM Model

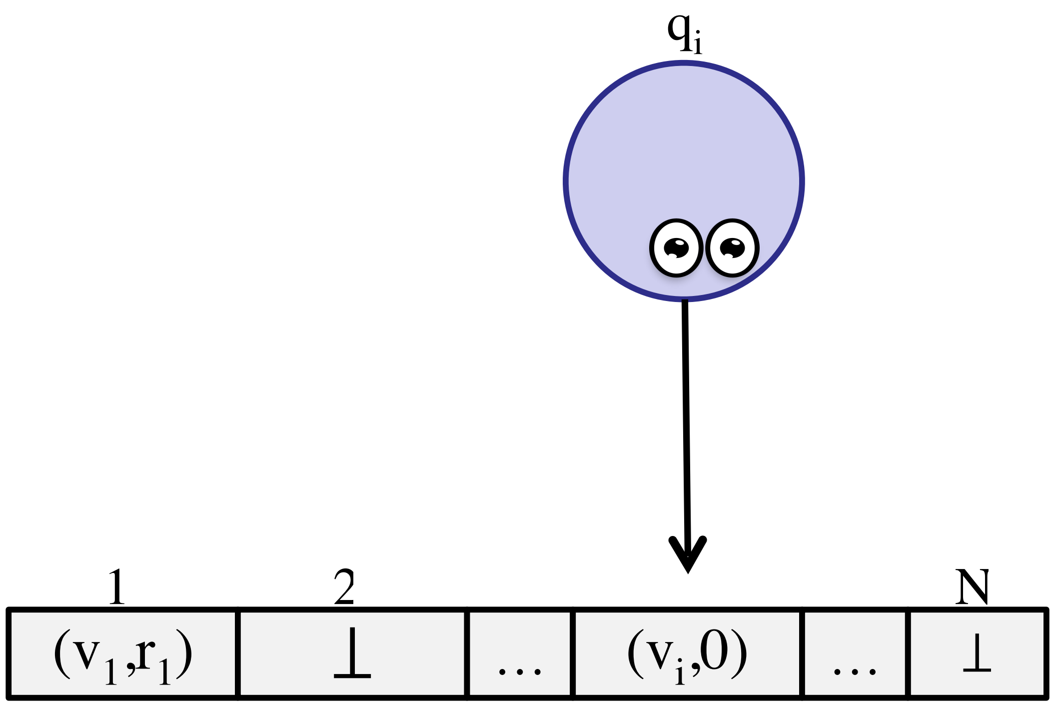


Gathering is Impossible in ARL

ARL Model

```
procedure GATHERING(G, v_i)
    Move(v_i, 0)
    view_i ← ∅
    while undecided do
        view_i ← view_i · { Look(G) }
        (v_i, r_i) ← Compute(view_i)
        Move(v_i, r_i)
    end while
    return v_i
end procedure
```

WFSM Model

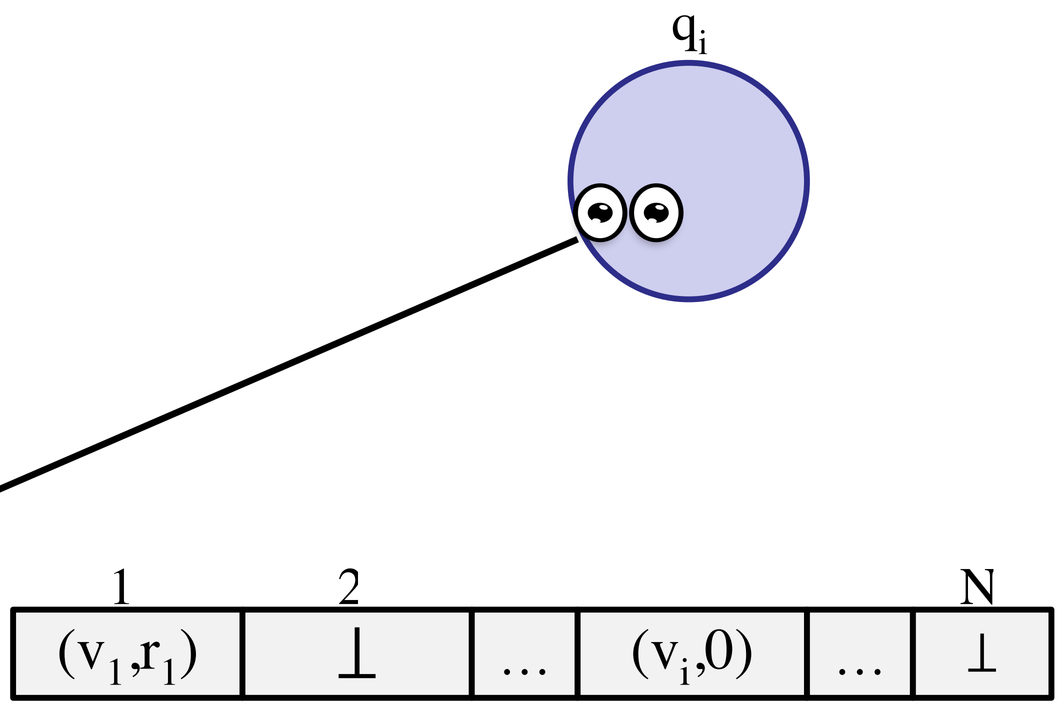


The simulation

ARL Model

```
procedure GATHERING(G, v_i)
    Move(v_i, 0)
    view_i ← ∅
    while undecided do
        view_i ← view_i · { Look(G) }
        (v_i, r_i) ← Compute(view_i)
        Move(v_i, r_i)
    end while
    return v_i
end procedure
```

WFSM Model

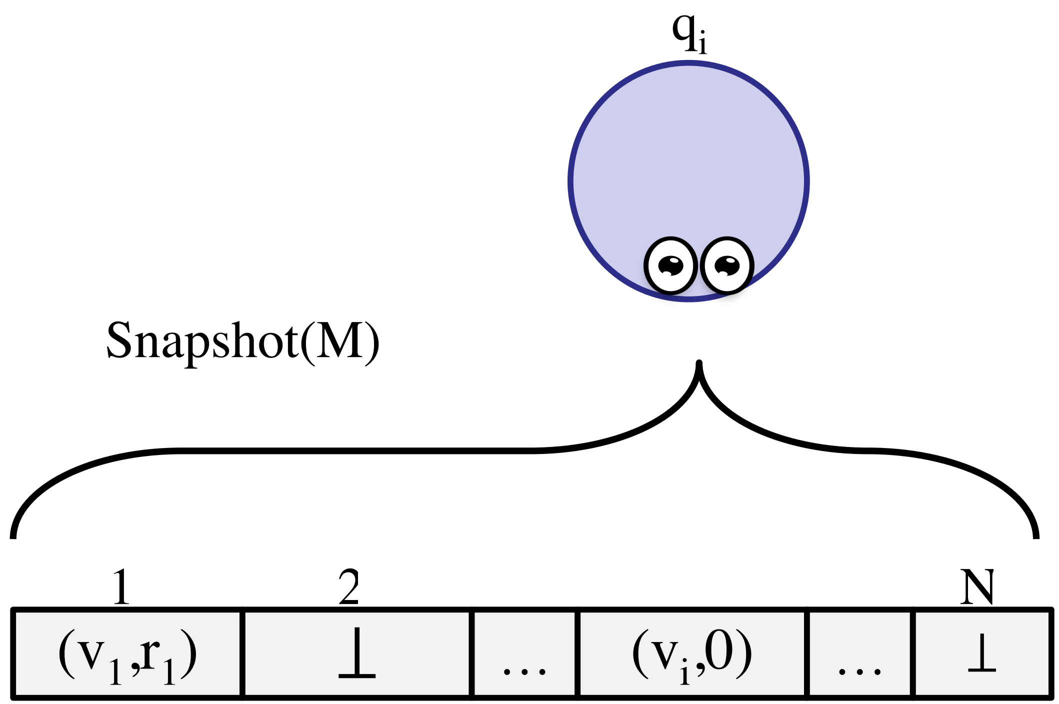


V

ARL Model

```
procedure GATHERING(G, v_i)
    Move(v_i, 0)
    view_i ← ∅
    while undecided do
        view_i ← view_i · { Look(G) }
        (v_i, r_i) ← Compute(view_i)
        Move(v_i, r_i)
    end while
    return v_i
end procedure
```

WFSM Model

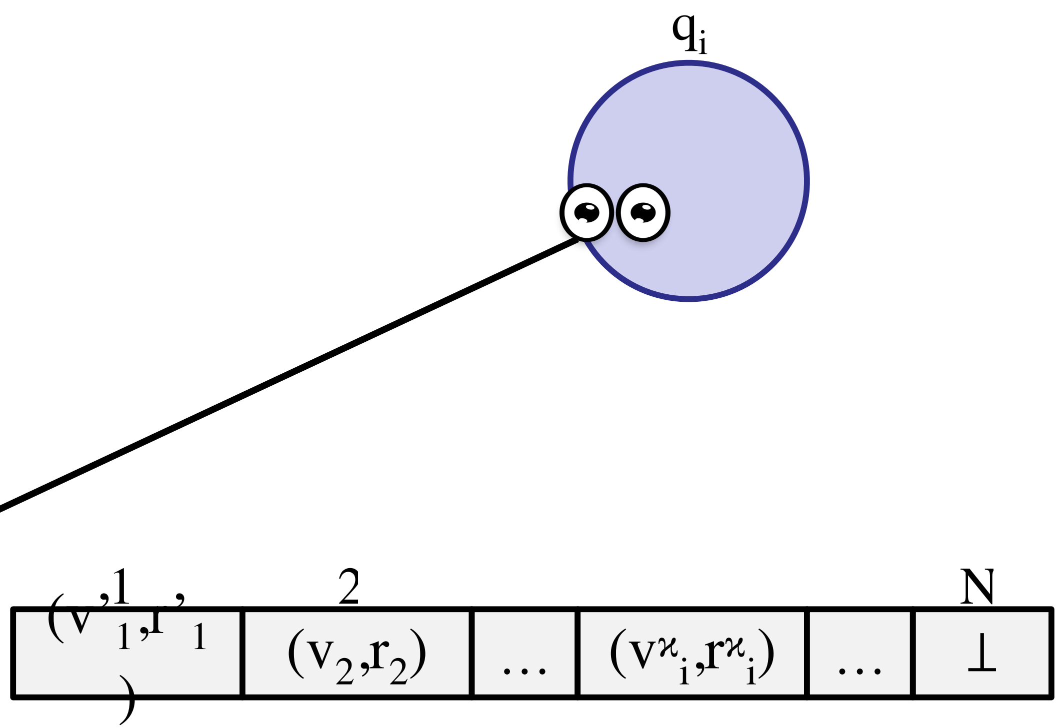


Gathering is Impossible in ARL

ARL Model

```
procedure GATHERING(G, v_i)
    Move(v_i, 0)
    view_i ← ∅
    while undecided do
        view_i ← view_i · { Look(G) }
        (v_i, r_i) ← Compute(view_i)
        Move(v_i, r_i)
    end while
    return v_i
end procedure
```

WFSM Model

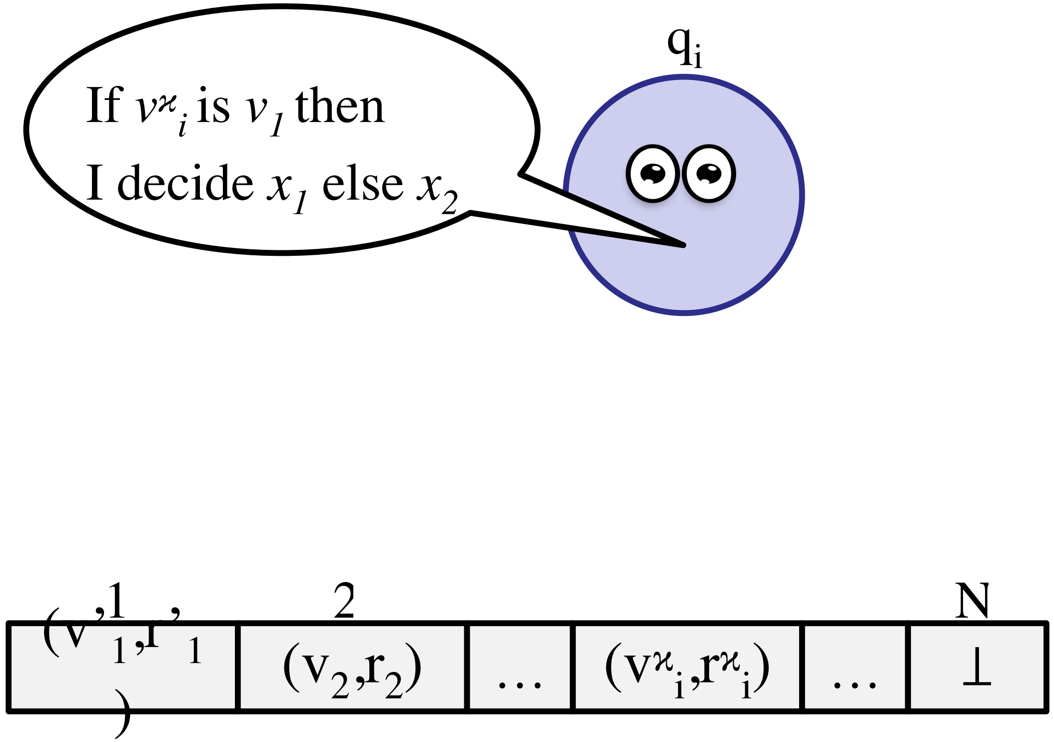


Gathering is Impossible in ARL

ARL Model

```
procedure GATHERING(G, v_i)
    Move(v_i, 0)
    view_i ← ∅
    while undecided do
        view_i ← view_i · { Look(G) }
        (v_i, r_i) ← Compute(view_i)
        Move(v_i, r_i)
    end while
    return v_i
end procedure
```

WFSM Model

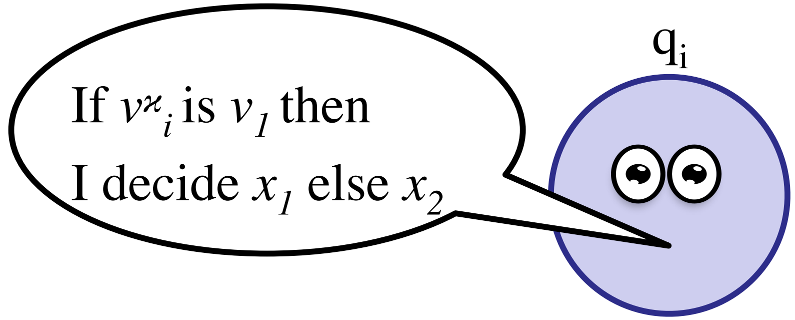


Gathering is Impossible in ARL

ARL Model

```
procedure GATHERING(G, v_i)
    Move(v_i, 0)
    view_i ← ∅
    while undecided do
        view_i ← view_i · { Look(G) }
        (v_i, r_i) ← Compute(view_i)
        Move(v_i, r_i)
    end while
    return v_i
end procedure
```

WFSM Model



1	2				N
(v'_1, r'_1)	(v_2, r_2)	…	$(v^{\varkappa}_i, r^{\varkappa}_i)$	…	$\perp$

- All the processes must to decide v^k_i ➡ We solve Binary Consensus

Weakening of gathering in ARL

Given that gathering is impossible, even in *strong* ARL, require only to get close to each other

Edge-Gathering for 2 Robots

- The following algorithm solves Edge-Gathering problem for 2 robots in any connected graph without lights.

```
Function EdgeGatheringTwoRobots(v_i, G)
  Move(v_i)
  for round_i ← 1 to diam(T) − 1 do
     view_i ← Look(G)
     S_i ← { v_j ∈ view_i : v_j ≠ v_i }
     if |S_i| = 1 then
        v_j ← the only vertex in S_i
        if (v_i, v_j) ∉ E(G) then
           v_i ← vertex of Path_T(v_i, v_j) that is adjacent to v_i
        end if
     end if
     Move(v_i)
  end for
  return v_i
```

Edge-Gathering

Results:

- So, 2 robots can solve edge gathering without lights
- What about $N \geq 3$ robots ?

Impossibility of Edge-Gathering

Theorem

- If G has a cycle, then there is no edge-gathering algorithm on G for $N \geq 3$ robots even if at most 2 robots fail, in *strong* ALR

Main impossibility of Edge-Gathering

- If G has a cycle, then there is no edge-gathering algorithm on G for $N \geq 3$ robots even if at most 2 robots fail, in *strong* ALR

By reduction to shared memory:
Suppose there is an algorithm ***A*** and let's prove that we can solve 2-set agreement for 3 robots.

Main impossibility of Edge-Gathering

- If G has a cycle, then there is no edge-gathering algorithm on G for $N \geq 3$ robots even if at most 2 robots fail, in *strong* ALR

By reduction to shared memory:
Suppose there is an algorithm ***A*** and let's prove that we can solve 2-set agreement for 3 robots.

Topological reason !!!

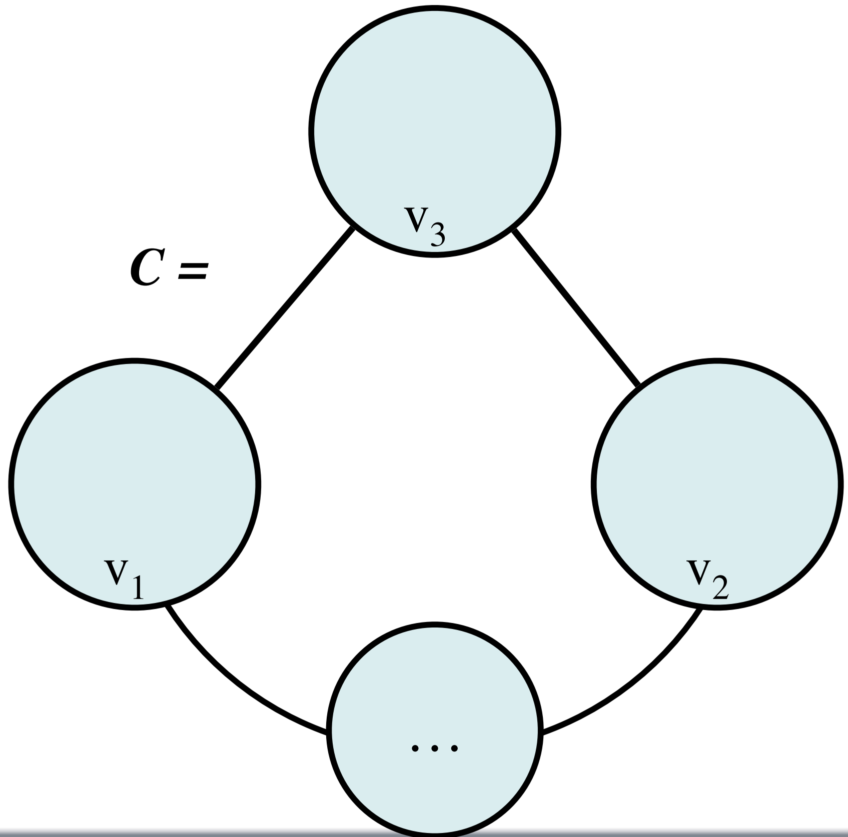
(Cycle contractibility->Sperner's lemma)

Impossibility of Edge-Gathering with Cycles

- If G has a cycle C, then there is no deterministic algorithm that solves edge gathering on G for $N \geq 3$ robots even if at most 2 robots fail.

  By contradiction : Suppose there is an algorithm A and let's prove that we can solve 2-set agreement for 3 robots.

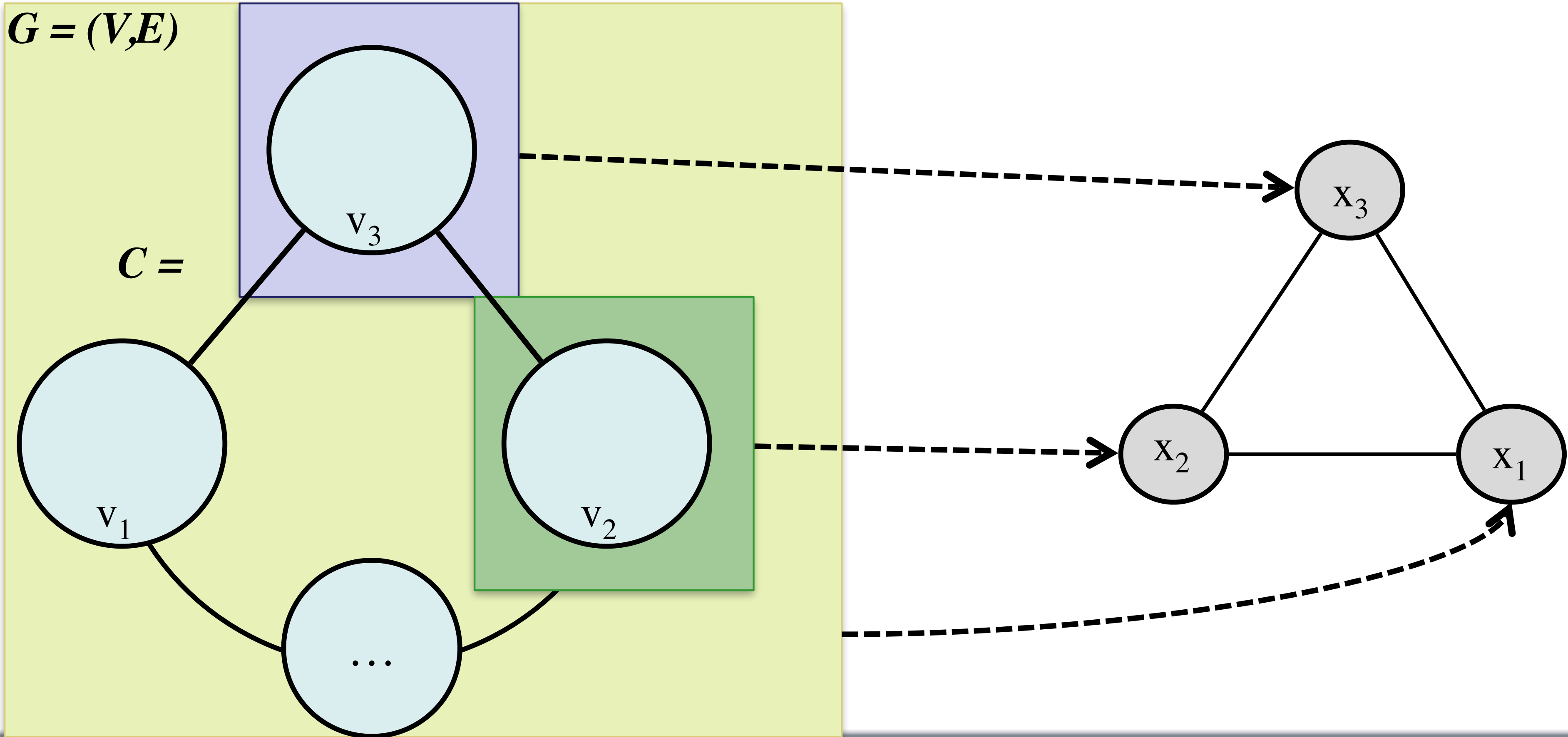
C =



Impossibility of Edge-Gathering with Cycles

- If G has a cycle C, then there is no deterministic algorithm that solves edge gathering on G for $N \geq 3$ robots even if at most 2 robots fail.

By contradiction : Suppose there is an algorithm *A* and let's prove that we can solve 2-set agreement for 3 robots.



Edge-Gathering without Lights

- Edge-Gathering without Lights its impossible in a graph G with $diam(G) \geq 3$

  (Proof: An indistinguishability argument)

Edge-Gathering $N \geq 3$

Main Results:

- If G has a cycle, then there is no edge-gathering algorithm on G for $N \geq 3$ robots even if at most 2 robots fail, in *strong* ALR

- If G is a tree, then there is an edge-gathering algorithm on G for $N \geq 3$ robots even if at most 2 robots fail, in *strong* ALR

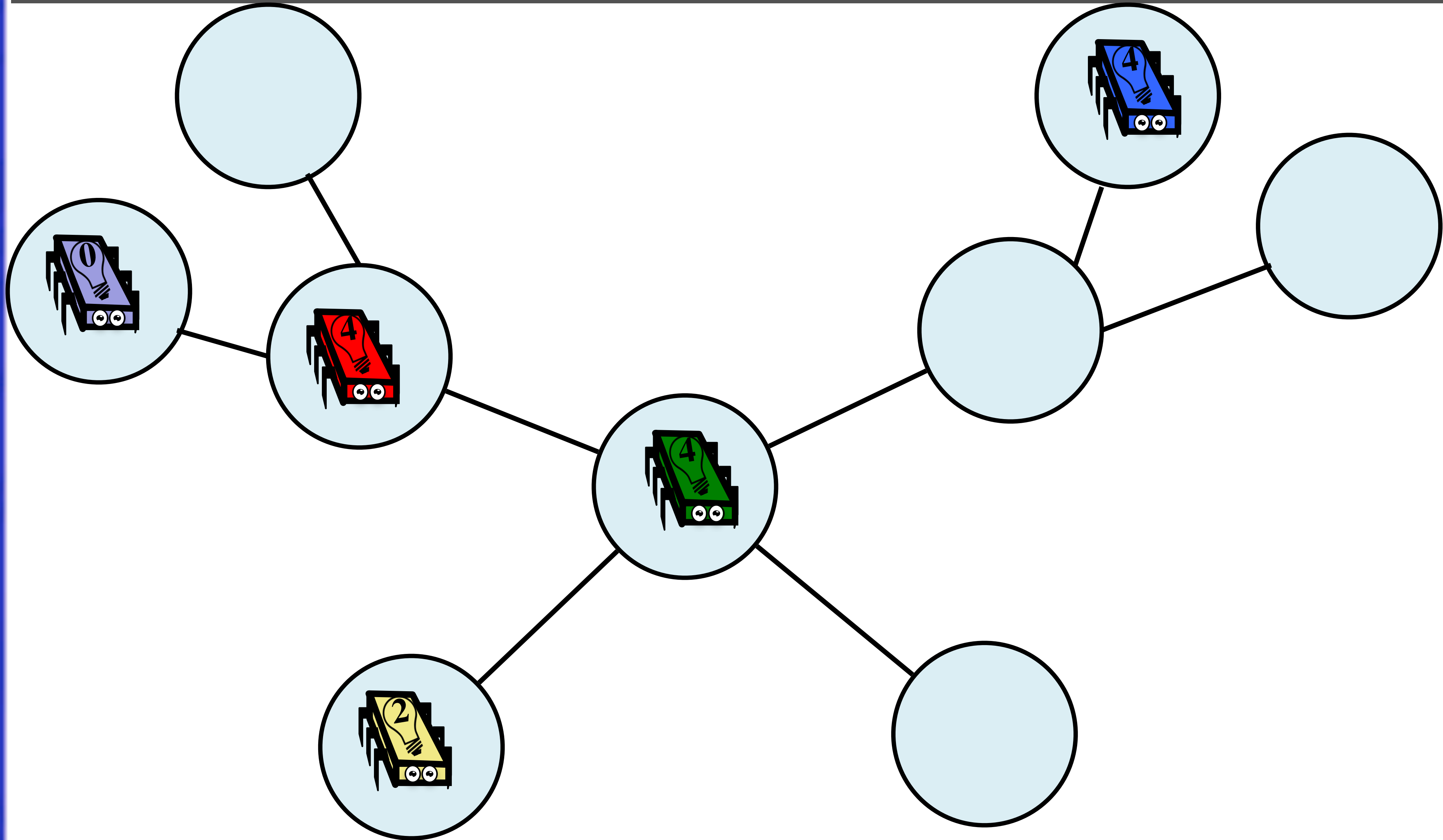
Edge-Gathering on Trees

- for $N \geq 3$ robots

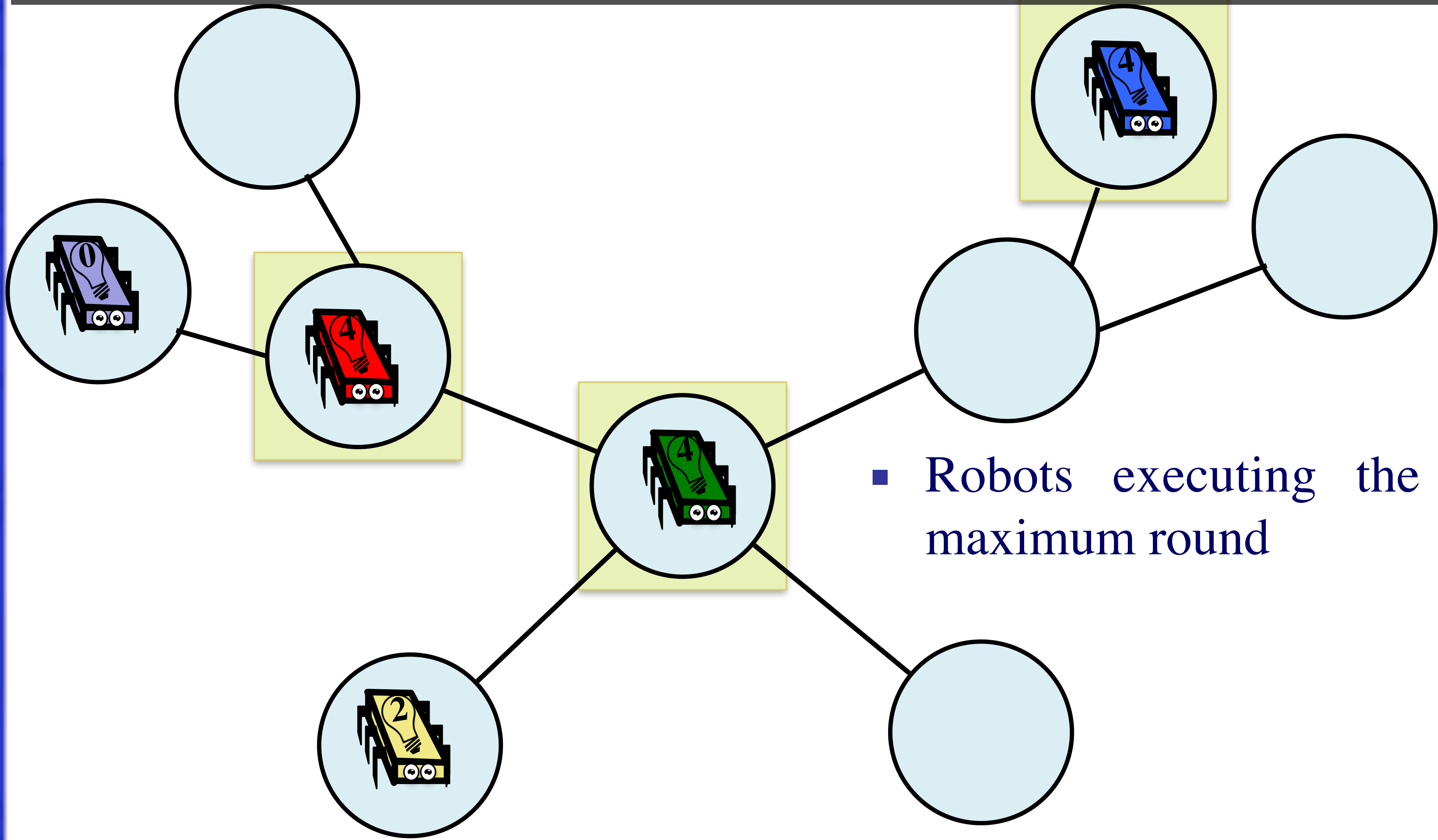
Need lights

```
Function EdgeGatheringTree(v_i, T)
Move(v_i, 0)
for r_i ← 1 to diam(T) − 1 do
    view_i ← Look(T)
    max_round_i ← máx{r_j : (*, r_j) ∈ view_i}
    S_i ← {v_j : (v_j, max_round_i) ∈ view_i ∨ v_j = v_i}
    T_i ← smallest subtree of T spanning all vertices in S_i
    if diam(T_i) > 0 ∧ v_i is leaf of T_i then
        v_i ← vertex of T_i that is adjacent to v_i
    end if
    Move(v_i, r_i)
end for
return v_i
```

Edge-Gathering on Trees



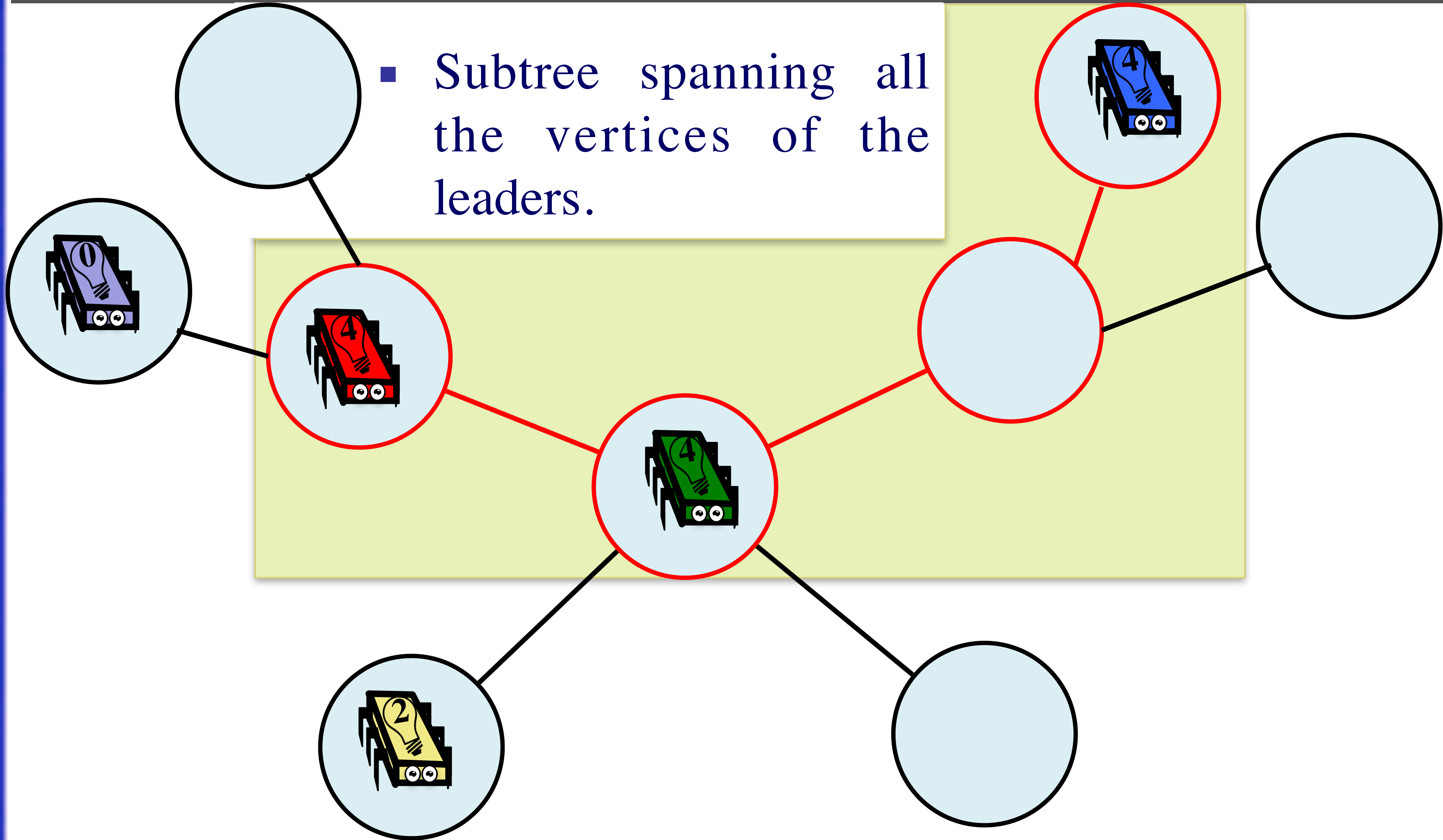
Edge-Gathering on Trees



- Robots executing the maximum round

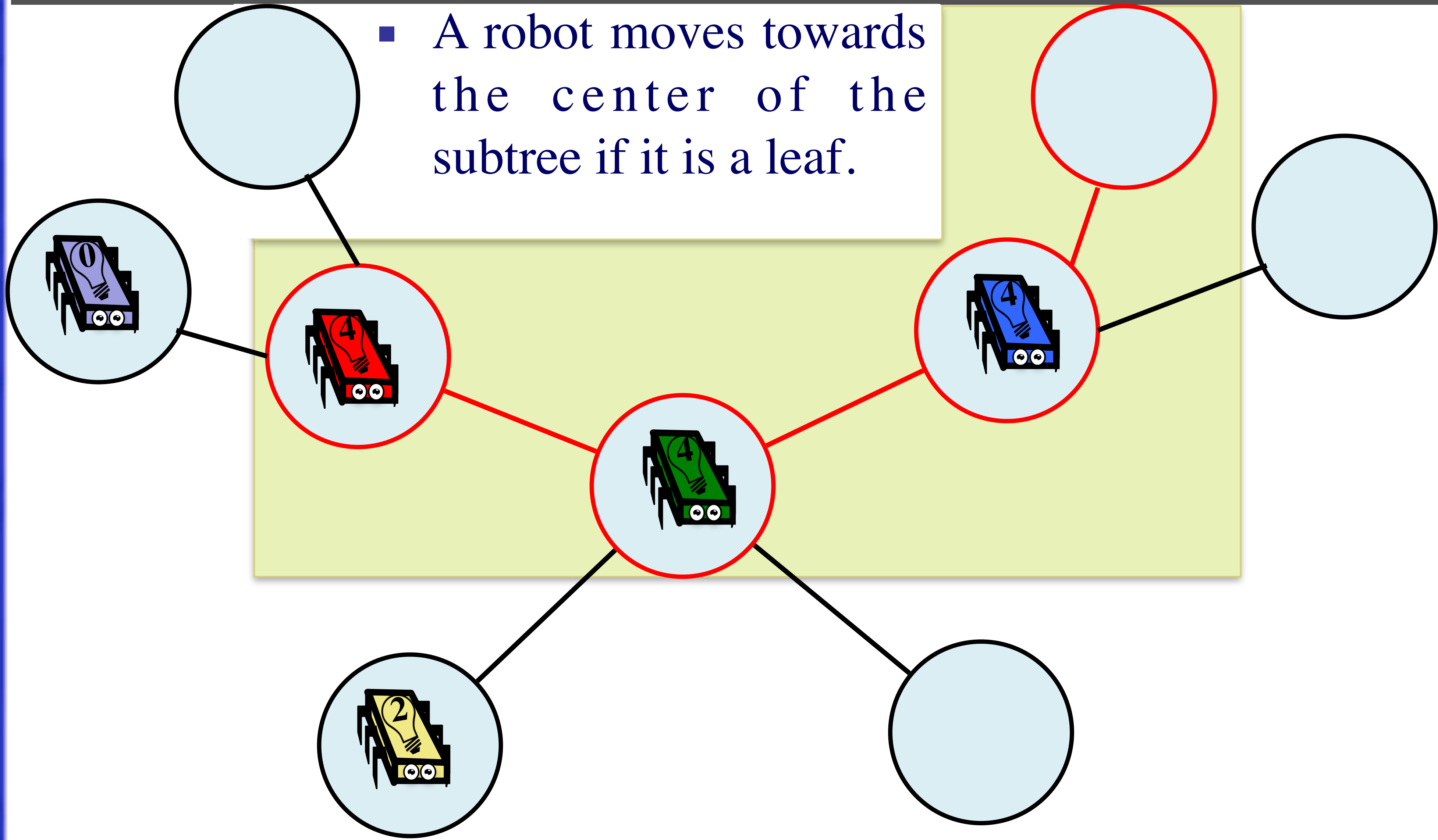
Edge-Gathering on Trees

- Subtree spanning all the vertices of the leaders.



Edge-Gathering on Trees

- A robot moves towards the center of the subtree if it is a leaf.



Edge-Gathering on Trees

- The algorithm is simple, but the analysis is not due to the combination of asynchrony, distinct waking times and failures

For example, in a given execution, two robots might compute their next vertices using the same maximal round but with very different sets of positions;

moreover, a robot might compute its next position considering only crashed robots.

Such difficulties make difficult to find and prove invariants

Main result

- We provide a characterization of the solvable robot tasks in graphs, in the strong ALR

A robot task on G, $\langle I, O, \Delta \rangle$ is solvable for N robots if and only if there is a subdivision Subd(I) and a simplicial map δ from Subd(I) to O, such that for every input simplex σ, $\delta(\mathrm{Subd}(\sigma)) \subseteq \Delta(\sigma)$.

Corollary

- The characterization implies undecidability:

  There is no (sequential) algorithm that *decides* if a given robot task on G for three processes tolerating two failures is solvable in the ALR model.

Conclusions (1)

- Investigated the basic asynchronous LOOK-COMPUTE-MOVE model, considering the possibility that not all robots are present initially.
- Robots on a graph, and may use lights
- Gathering-type of problems with stopping
- Characterized the solvable cases

Conclusions (2)

- Exposed a close relationship with fault-tolerant shared-memory computing,
- and hence with topology
- Providing a framework to unify the many Look/Compute/Move different models
- and to study them, eg., synchronous vs asynchronous allow to solve different tasks, but the underlying topological setting is similar

Open problems

- Non-gathering type of problems
- Other spaces represented by a simplicial complex, and continuous versions
- Other models: e.g. synchronous, semi-synchronous, Byzantine failures…
- We focused on computability; study complexity (time, memory, lights, etc)
- Dynamic networks

Thanks you for your attention