Impossibility of Gathering, a Certification

Pierre Courtieu Lionel Rieg Sébastien Tixeuil Xavier Urbain

Collège de France, LIP6, CÉDRIC

FRIDA, June 5, 2015

Formal proofs

Why specification & formal proofs?

- specification
  $\rightsquigarrow$ clear statement of results
- show implicit hypotheses
  $\rightsquigarrow$ slight difference in hypotheses $\Rightarrow$ huge difference in results
- ensure correctness of proofs
  $\rightsquigarrow$ some definitions have dark corners

Formal proofs

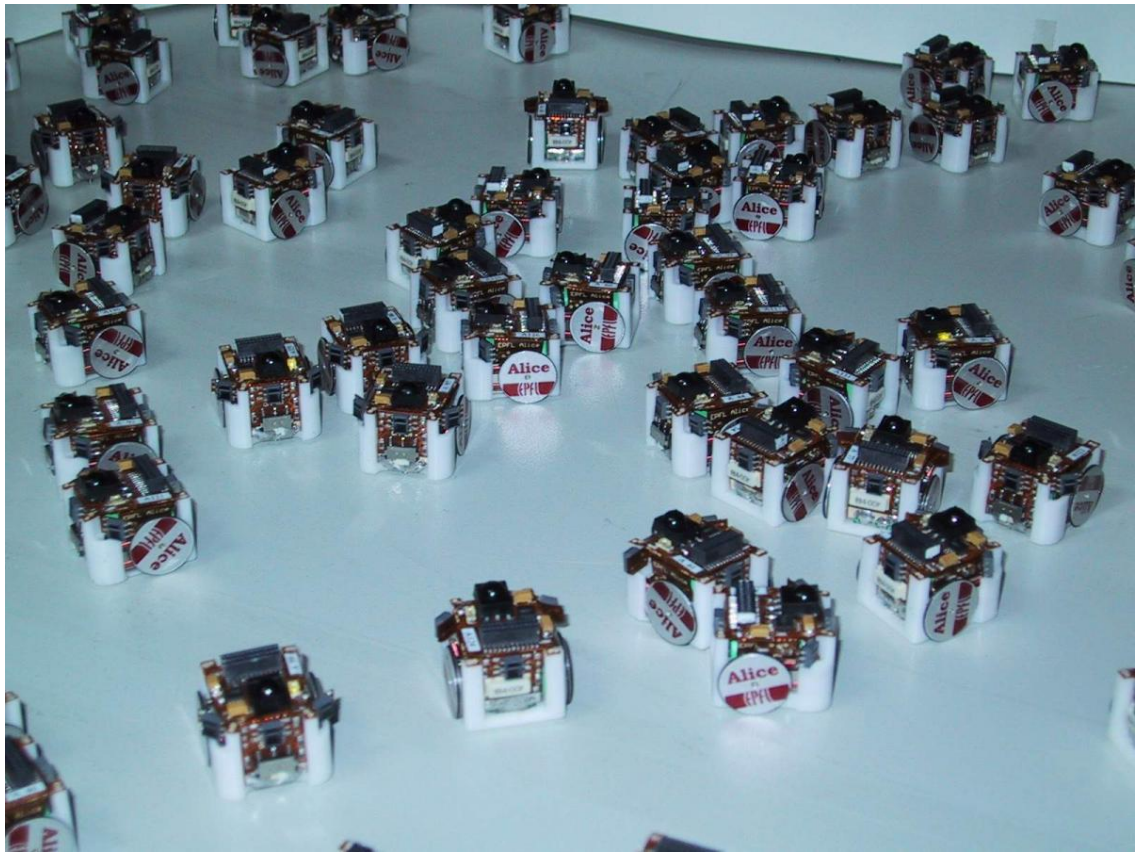
Why specification & formal proofs?

- specification
  $\rightsquigarrow$ clear statement of results
- show implicit hypotheses
  $\rightsquigarrow$ slight difference in hypotheses $\Rightarrow$ huge difference in results
- ensure correctness of proofs
  $\rightsquigarrow$ some definitions have dark corners

Coq:

- functional programming language + proof assistant
- based on the Curry-Howard correspondence (proof = program)
- higher-order logic, induction/coinduction
- interactive, requires expertise
- complementary to model checking/TLA+

Why robot swarms



- rescue/exploration (hostile env.)
- cooperative construction
- drone formation flying
- "swarmboats"
- low cost

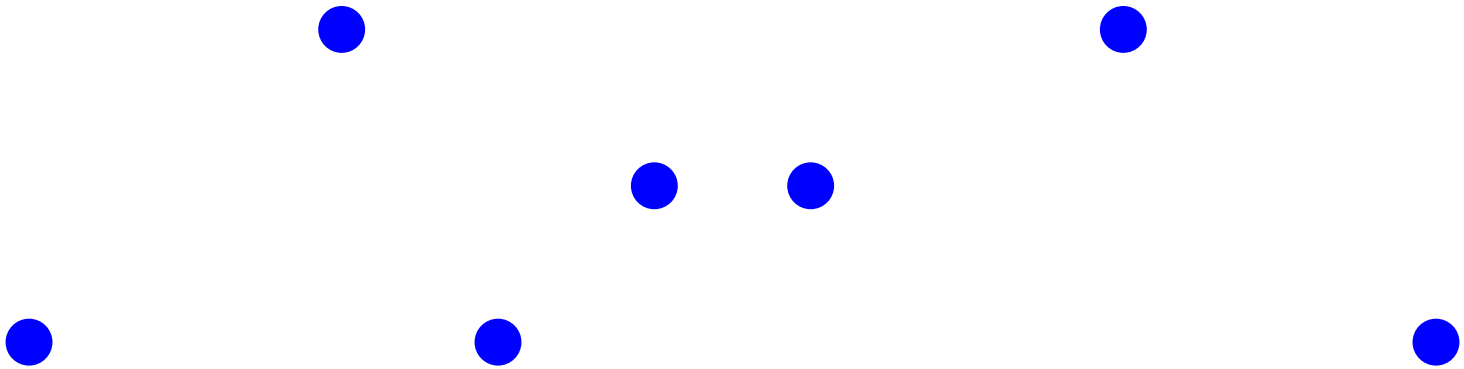
Robots Model

Robots: [SY99, FPD13]

Robots Model

Robots: [SY99, FPD13]

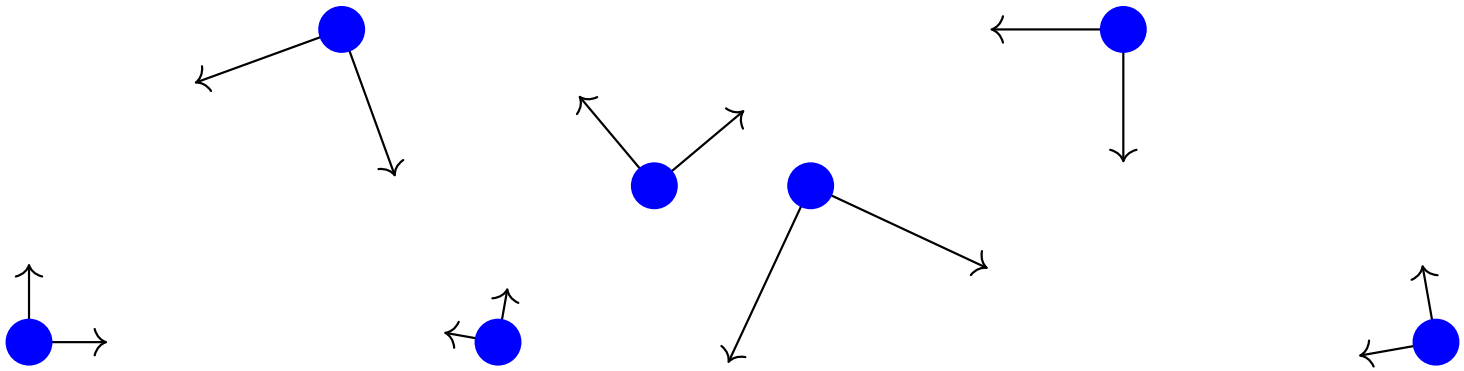
- points



Robots Model

Robots: [SY99, FPD13]

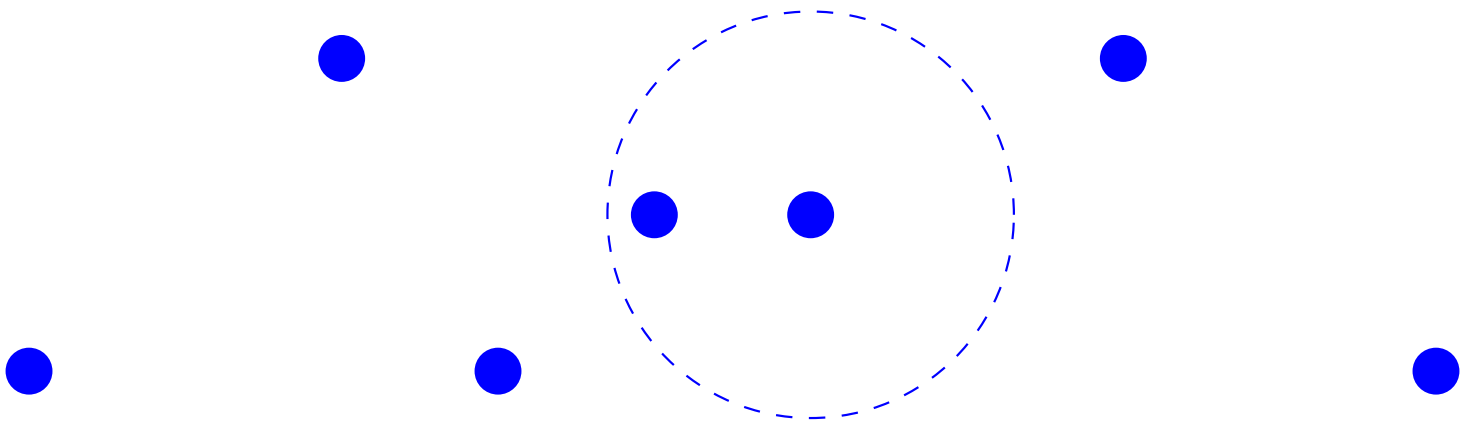
- points
- not necessarily common scale/direction/chirality
  $\rightsquigarrow$ local referential



Robots Model

Robots: [SY99, FPD13]

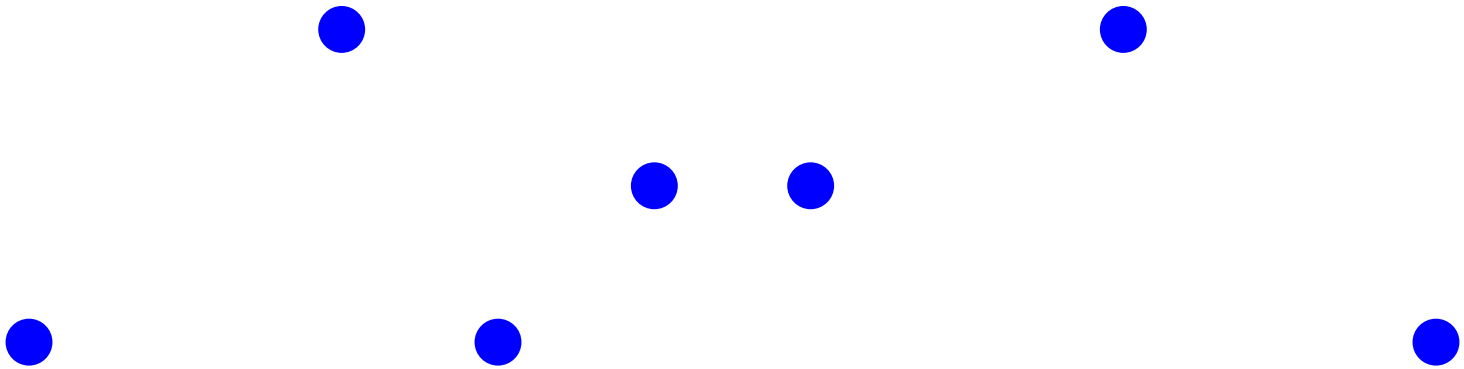
- points
- not necessarily common scale/direction/chirality
  $\rightsquigarrow$ local referential
- limited vision? multiplicity?



Robots Model

Robots: [SY99, FPD13]

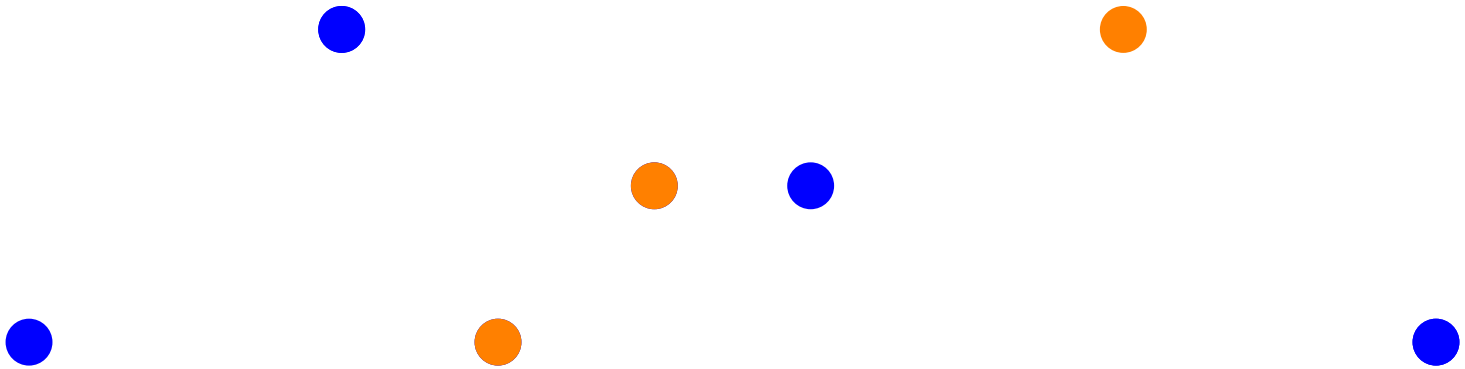
- points
- not necessarily common scale/direction/chirality
  $\rightsquigarrow$ local referential
- limited vision? multiplicity?
- same deterministic program (robogram)



Robots Model

Robots: [SY99, FPD13]

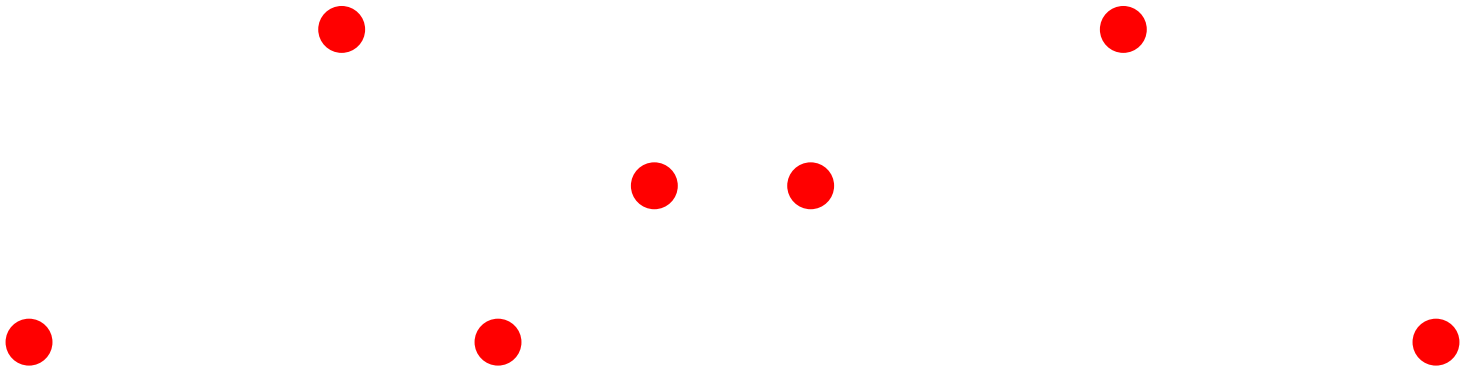
- points
- not necessarily common scale/direction/chirality
  $\rightsquigarrow$ local referential
- limited vision? multiplicity?
- same deterministic program (robogram)
- with byzantine faults



Robots Model

Robots: [SY99, FPD13]

- points
- not necessarily common scale/direction/chirality
  $\rightsquigarrow$ local referential
- limited vision? multiplicity?
- same deterministic program (robogram)
- with byzantine faults
- anonymous



Execution Model

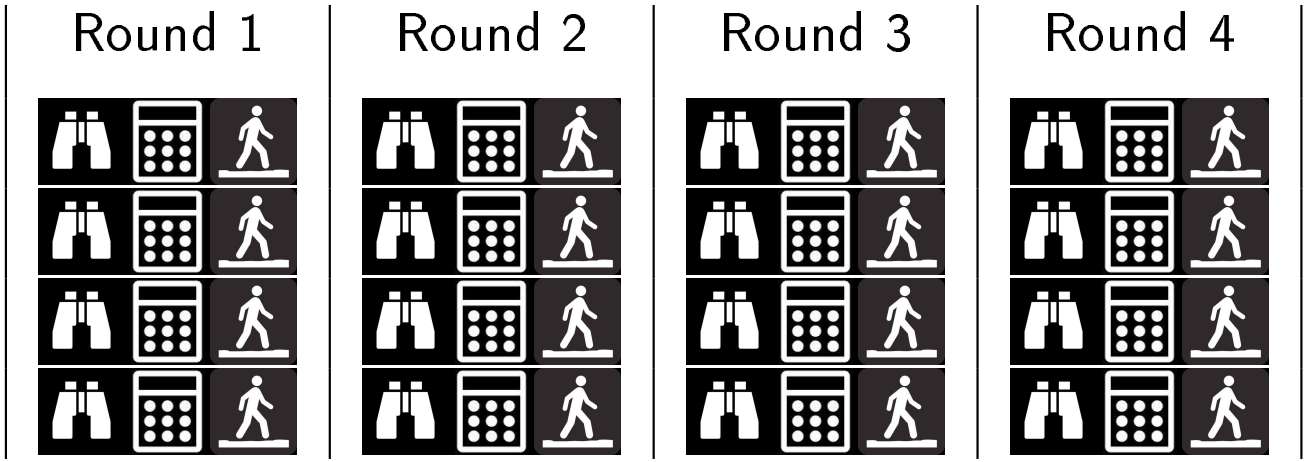
Execution:

- 3 phases: Look, Compute, Move
  - rigid move: teleport, infinite range
  - grouped or not, atomic or not

Execution Model

Execution:

- 3 phases: Look, Compute, Move
  - rigid move: teleport, infinite range
  - grouped or not, atomic or not
- 1^{st} kind: synchronous phases (rounds)
  - FSYNC: all robots are activated at each round

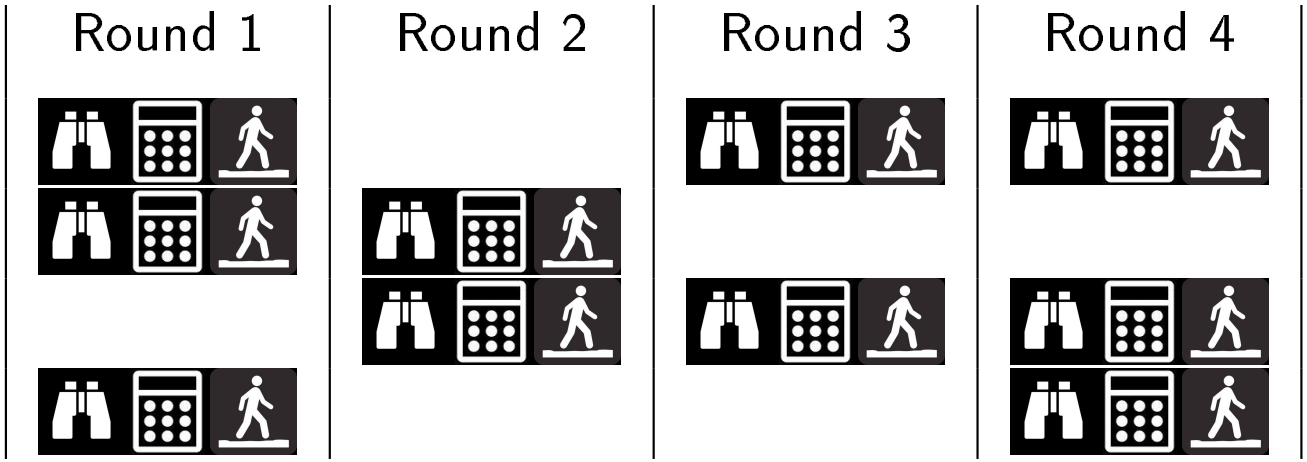


Execution Model

Execution:

- 3 phases: Look, Compute, Move
  - rigid move: teleport, infinite range
  - grouped or not, atomic or not
- 1^{st} kind: synchronous phases (rounds)
  - FSYNC: all robots are activated at each round
  - SSYNC: only a subset is activated

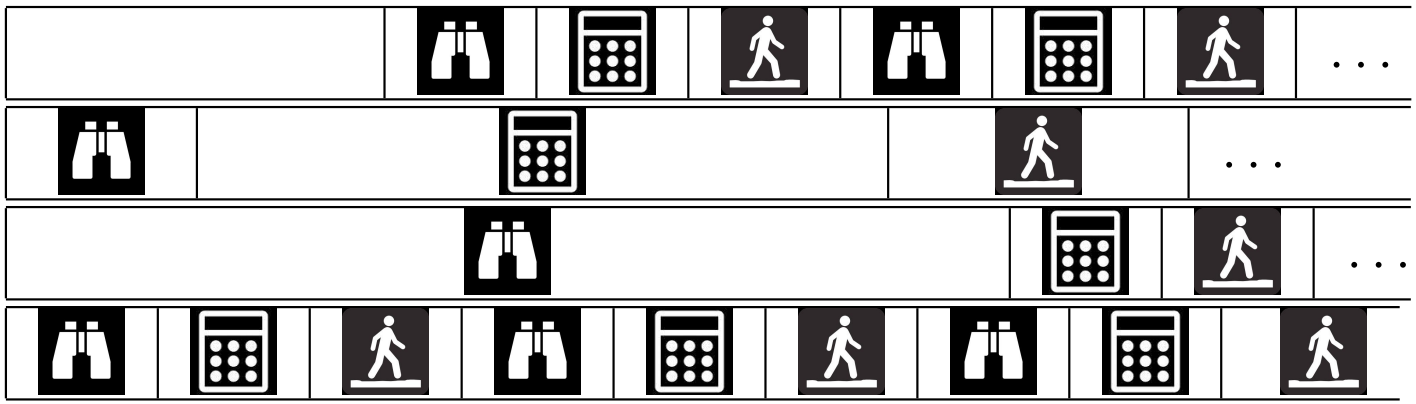
  $\rightsquigarrow$ fairness of the demon (scheduling)



Execution Model

Execution:

- 3 phases: Look, Compute, Move
  - rigid move: teleport, infinite range
  - grouped or not, atomic or not
- 1^{st} kind: synchronous phases (rounds)
  - FSYNC: all robots are activated at each round
  - SSYNC: only a subset is activated
    $\rightsquigarrow$ fairness of the demon (scheduling)
- 2^{nd} kind: ASYNC, no synchrony

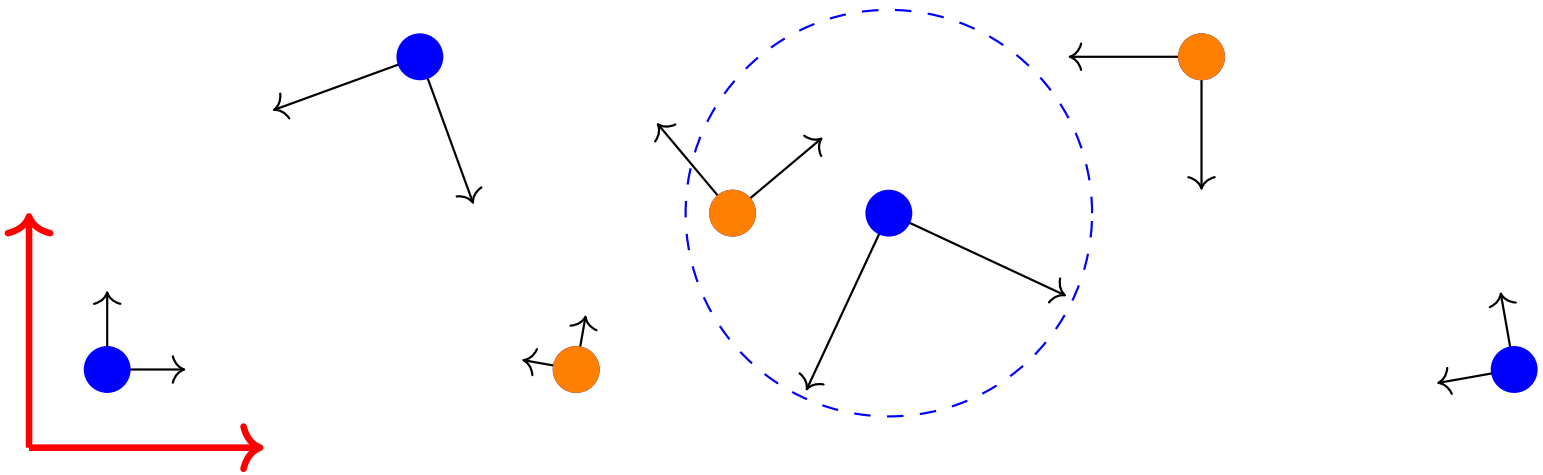


Local *vs* Global

Global (demon)	**Local** (robots)
absolute locations (abstract type)	local referential
byzantine/good robots: B / G	
identifiers ident $= G + B$	
configuration : ident $\rightarrow$ location	local configuration

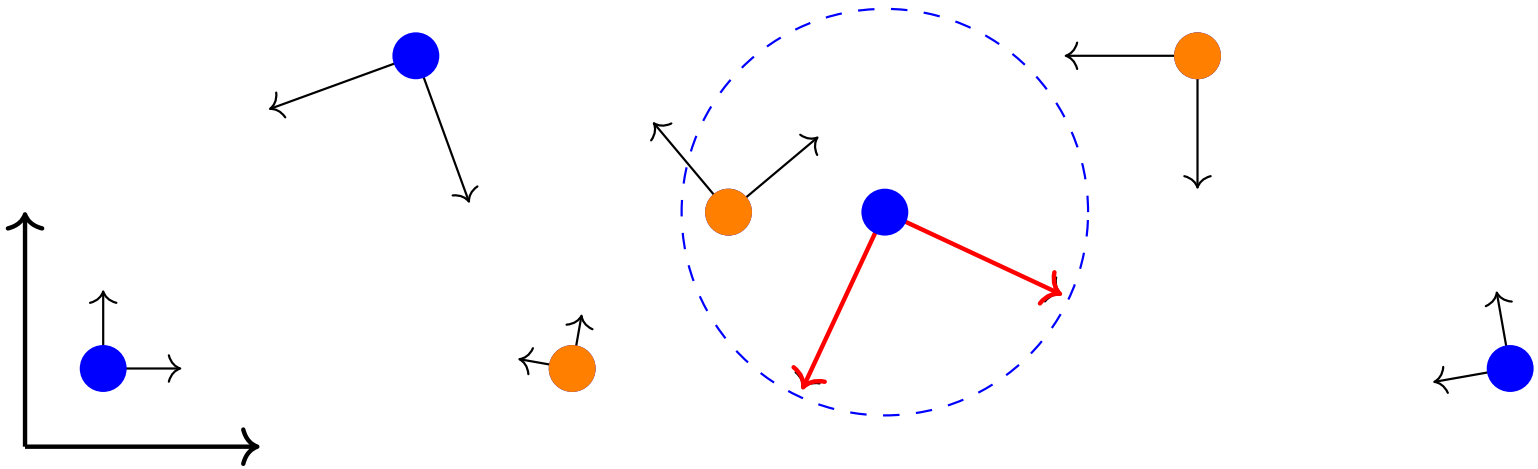
Local *vs* Global

Global (demon)	**Local** (robots)
absolute locations (abstract type)	local referential
byzantine/good robots: B / G	
identifiers ident $= G + B$	
configuration : ident $\rightarrow$ location	local configuration



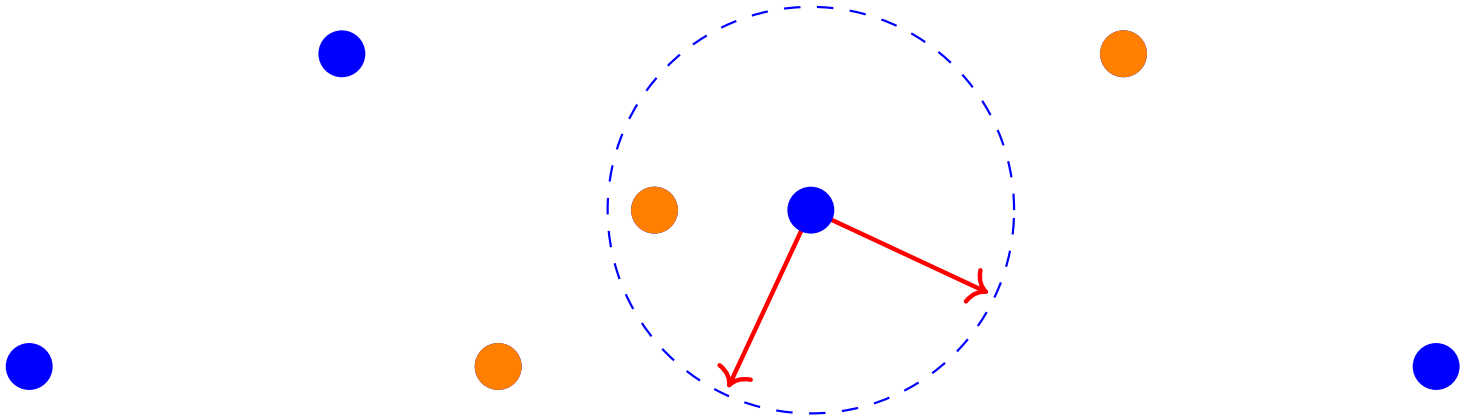
Local *vs* Global

Global (demon)	**Local** (robots)
absolute locations (abstract type)	local referential
byzantine/good robots: B / G	
identifiers ident $= G + B$	
configuration : ident $\rightarrow$ location	local configuration



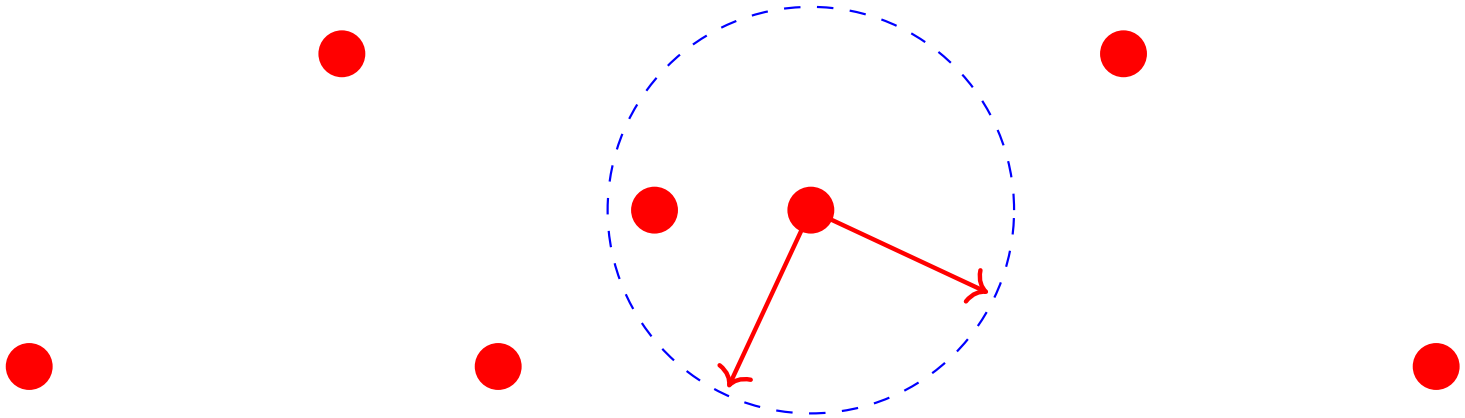
Local *vs* Global

Global (demon)	**Local** (robots)
absolute locations (abstract type)	local referential
byzantine/good robots: B / G	
identifiers ident $= G + B$	
configuration : ident $\rightarrow$ location	local configuration



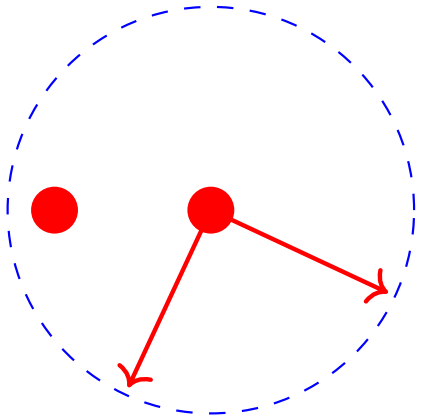
Local *vs* Global

Global (demon)	**Local** (robots)
absolute locations (abstract type)	local referential
byzantine/good robots: B / G	
identifiers ident $= G + B$	
configuration : ident $\rightarrow$ location	local configuration



Local *vs* Global

Global (demon)	**Local** (robots)
absolute locations (abstract type)	local referential
byzantine/good robots: B / G	
identifiers ident $= G + B$	
configuration : ident $\rightarrow$ location	local configuration



spectrum = local and weakened view of a configuration

Local *vs* Global

Global (demon)	**Local** (robots)
absolute locations (abstract type)	local referential
byzantine/good robots: B / G	
identifiers ident $= G + B$	
configuration : ident $\rightarrow$ location	local configuration
global spectrum	spectrum (abstract type)
	robogram
	: spectrum $\rightarrow$ location
round r d : config $\rightarrow$ config	
execution : stream of configs	

Local *vs* Global

Global (demon)	**Local** (robots)
absolute locations (abstract type)	local referential
byzantine/good robots: B / G	
identifiers ident $= G + B$	
configuration : ident $\rightarrow$ location	local configuration
global spectrum	spectrum (abstract type)
	robogram
	: spectrum $\rightarrow$ location
round r d : config $\rightarrow$ config	
execution : stream of configs	

We get exactly these definitions in Coq!

SSYNC round in Coq

```
Definition round (r : robogram) (da : demonic_action) (config : Pos.t)
  : Pos.t :=
  fun id ⇒
    let t := config id in                                  (* global pos of id *)
    match da.(step) id with
      | Off ⇒ t                                              (* not activated *)
      | On f _ ⇒                                  (* activated, new frame = f *)
        match id with
        | Byz b ⇒ da.(relocate_byz) b                         (* relocate byz *)
        | Good g ⇒
          let local_config := Pos.map (f t) config in
          let spectrum := da.(spectrum_of) g local_config in
          (f t)⁻¹ (r spectrum)                        (* apply r + back to global *)
        end
    end.
```

Gathering (on a line)

Task

Gather all robots at the exact same place and make them stay there

Gathering (on a line)

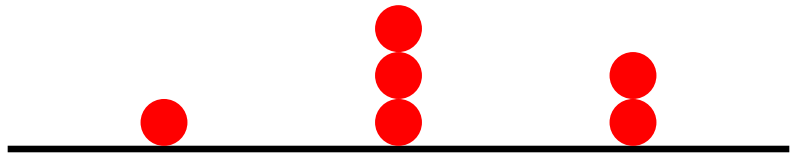
Task

Gather all robots at the exact same place and make them stay there

Gathering (on a line)

Task

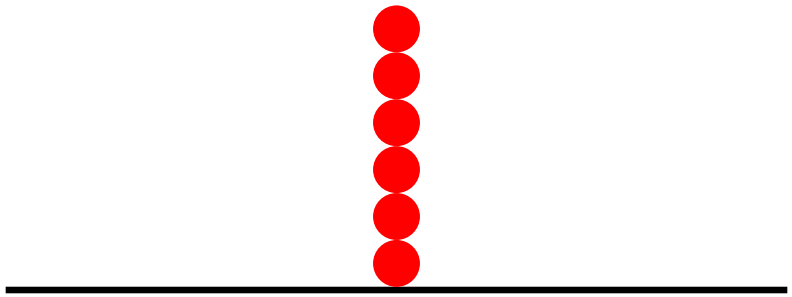
Gather all robots at the exact same place and make them stay there



Gathering (on a line)

Task

Gather all robots at the exact same place and make them stay there



Gathering (on a line)

Task

Gather all robots at the exact same place and make them stay there

Models:

- Robots: in $\mathbb{R}$, no byzantine, strong global multiplicity
- Execution: SSYNC (easy with FSYNC)

Gathering (on a line)

Task

Gather all robots at the exact same place and make them stay there

Models:

- Robots: in $\mathbb{R}$, no byzantine, strong global multiplicity
- Execution: SSYNC (easy with FSYNC)

Theorem (Suzuki & Yamashita 99)

Initial situation: robots at pairwise distinct locations
Gathering impossible for 2 robots

Theorem (Our result)

Gathering impossible for an even number of robots *(no initial cond)*

Proof Sketch (same as Suzuki & Yamashita)

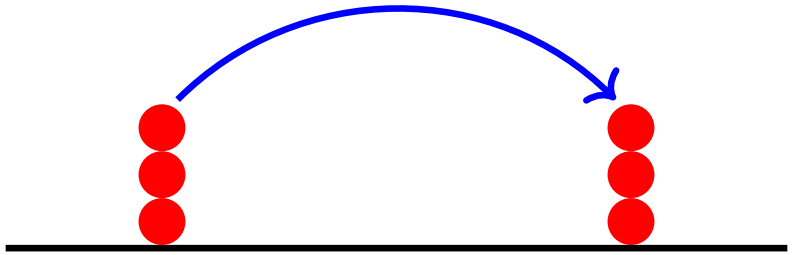
Keeping symmetry, same behavior

Two cases:

1.

2.

Proof Sketch (same as Suzuki & Yamashita)



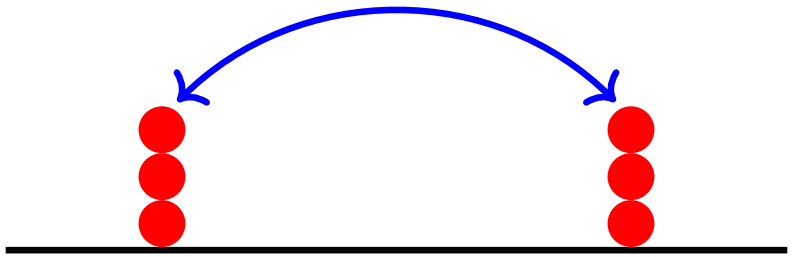
Keeping symmetry, same behavior

Two cases:

1. tower goes on the other one

2.

Proof Sketch (same as Suzuki & Yamashita)



Keeping symmetry, same behavior

Two cases:

1. tower goes on the other one
   activate both: swap positions
2.

Proof Sketch (same as Suzuki & Yamashita)

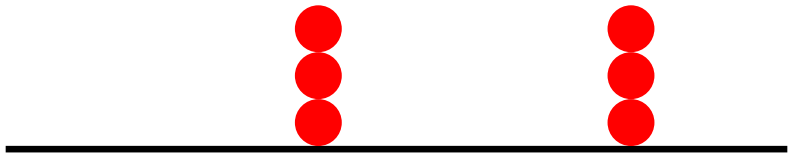
Keeping symmetry, same behavior

Two cases:

1.

2. tower goes anywhere else

Proof Sketch (same as Suzuki & Yamashita)



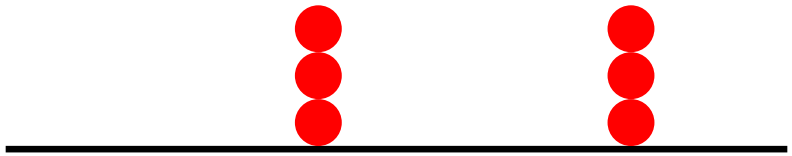
Keeping symmetry, same behavior

Two cases:

1.

2. tower goes anywhere else
   activate only one: same configuration up to the scale

Proof Sketch (same as Suzuki & Yamashita)



Keeping symmetry, same behavior

Two cases:

1. tower goes on the other one
   activate both: swap positions
2. tower goes anywhere else
   activate only one: same configuration up to the scale

In both cases, local configuration identical at next round

Formalisation in Coq: Inductive/Coinductive

- Coinductive: forever true properties
  - fairness of demon
  - robots stay gathered

Gathering = All robots forever at the exact same place

```
Definition gathered_at (pt : Loc.t) (config : Config.t) :=
  ∀ g : G, config (Good g) = pt.

CoInductive Gather (pt : Loc.t) (e : execution) : Prop :=
  Gathering : gathered_at pt (execution_head e) →
              Gather pt (execution_tail e) → Gather pt e.
```

Formalisation in Coq: Inductive/Coinductive

- Coinductive: forever true properties
  - fairness of demon
  - robots stay gathered
- Inductive: eventually true properties
  - robots will gather

```
Definition gathered_at (pt : Loc.t) (config : Config.t) :=
  ∀ g : G, config (Good g) = pt.

CoInductive Gather (pt : Loc.t) (e : execution) : Prop :=
  Gathering : gathered_at pt (execution_head e) →
              Gather pt (execution_tail e) → Gather pt e.

Inductive WillGather (pt : Loc.t) (e : execution) : Prop :=
  | Now : Gather pt e → WillGather pt e
  | Later : WillGather pt (execution_tail e) → WillGather pt e.
```

Formalisation in Coq: Final Theorem

```
Definition gathered_at (pt : Loc.t) (config : Config.t) :=
  ∀ g : G, config (Good g) = pt.

CoInductive Gather (pt : Loc.t) (e : execution) : Prop :=
  Gathering : gathered_at pt (execution_head e) →
              Gather pt (execution_tail e) → Gather pt e.

Inductive WillGather (pt : Loc.t) (e : execution) : Prop :=
  | Now : Gather pt e → WillGather pt e
  | Later : WillGather pt (execution_tail e) → WillGather pt e.
```

Formalisation in Coq: Final Theorem

```
Definition gathered_at (pt : Loc.t) (config : Config.t) :=
  ∀ g : G, config (Good g) = pt.

CoInductive Gather (pt : Loc.t) (e : execution) : Prop :=
  Gathering : gathered_at pt (execution_head e) →
              Gather pt (execution_tail e) → Gather pt e.

Inductive WillGather (pt : Loc.t) (e : execution) : Prop :=
  | Now : Gather pt e → WillGather pt e
  | Later : WillGather pt (execution_tail e) → WillGather pt e.

Definition solGathering (r : robogram) (d : demon) :=
  ∀ config : Config.t, ∃ pt : Loc.t, WillGather pt (execute r d pos).

Theorem noGathering : even |G| → inhabited G →
  ∀ k r, 1≤k → ∃d, kFair k d ∧ ¬solGathering r d.
```

Formalisation in Coq: Final Theorem

```
Definition gathered_at (pt : Loc.t) (config : Config.t) :=
  ∀ g : G, config (Good g) = pt.

CoInductive Gather (pt : Loc.t) (e : execution) : Prop :=
  Gathering : gathered_at pt (execution_head e) →
              Gather pt (execution_tail e) → Gather pt e.

Inductive WillGather (pt : Loc.t) (e : execution) : Prop :=
  | Now : Gather pt e → WillGather pt e
  | Later : WillGather pt (execution_tail e) → WillGather pt e.

Definition solGathering (r : robogram) (d : demon) :=
  ∀ config : Config.t, ∃ pt : Loc.t, WillGather pt (execute r d pos).

Theorem noGathering : even |G| → inhabited G →
  ∀ k r, 1≤k → ∃d, kFair k d ∧ ¬solGathering r d.
```

Higher-order important here!

Formalisation in Coq: Main Steps

Robots in two halves: `left` and `right`

One reference configuration: pos1: `left` at 0, `right` at 1.

pos2 := `left` at 1, `right` at 0
pos3 d := `left` at 0, `right` at d
move := r (Spect.from_config pos1)

- If $move = 1$
  - define demon1
  - by demon1, local config = pos1 — FSYNC demon
    $\rightsquigarrow$ identity or symmetry
  - pos1 $\xrightarrow{round}$ pos2 $\xrightarrow{round}$ pos1
- If $move \neq 1$
  - same thing but every other round — 1-fair SSYNC demon
    $\rightsquigarrow$ scaling or scaling + symmetry
  - pos3 d $\xrightarrow{round}\xrightarrow{round}$ pos3 d/ρ^2

Conclusion

Pactole framework:

- `http://pactole.lri.fr`
- various models (sensors)
- higher-order + inductive/coinductive very useful
- impossibility results (convergence, gathering)
- correctness algo, relaxed initial cond
- complementary to model-checking/TLA+