Formal Methods for the Verification of Distributed Algorithms

Paul Gastin

Laboratoire Spécification et Vérification
ENS Paris Saclay & CNRS

with C. Aiswarya (CMI) & Benedikt Bollig (LSV)

Motivations

- Distributed algorithms are extremely difficult to get right
- Correctness proofs are often involved
- Formal methods may help verifying the correctness of tricky algorithms

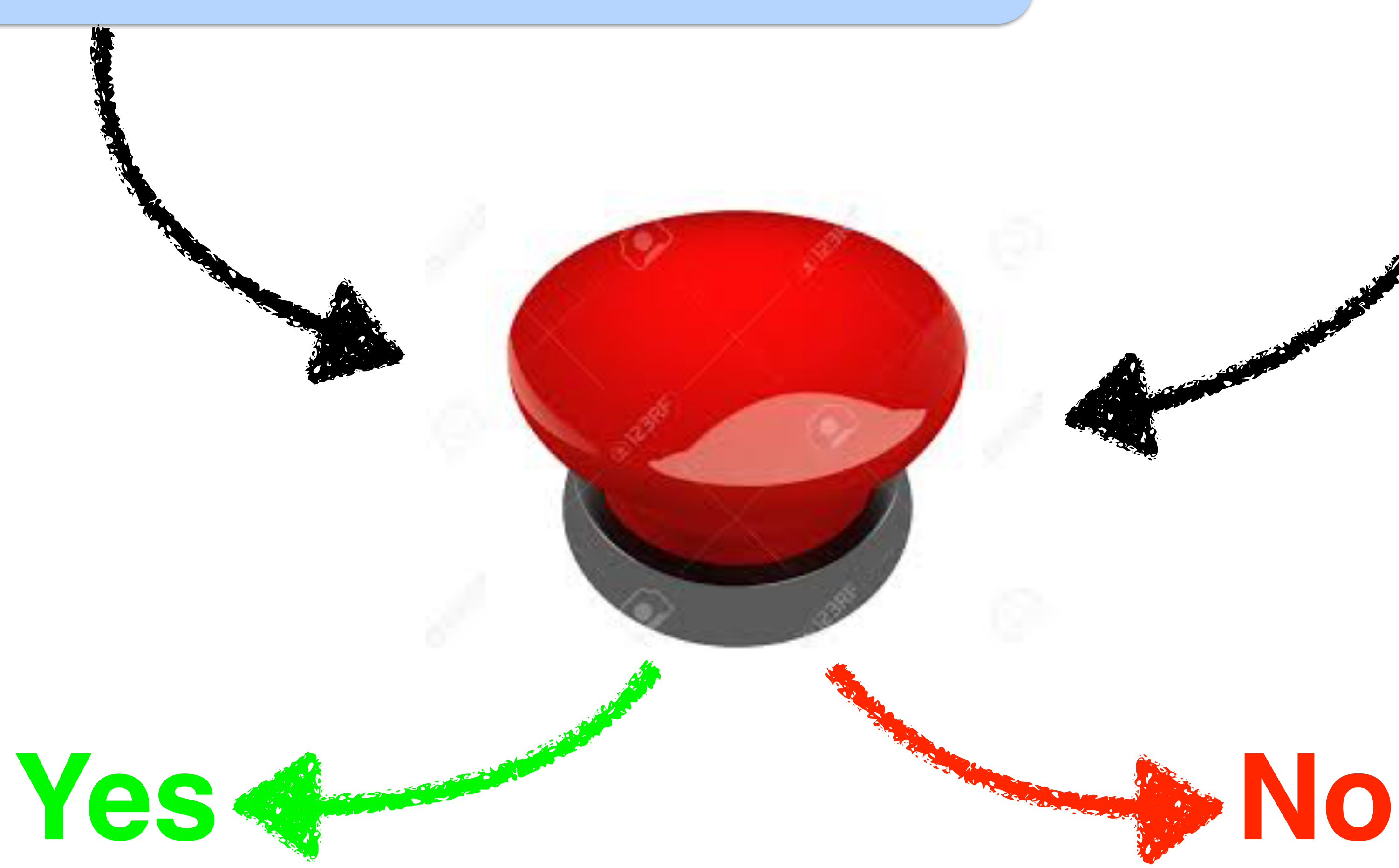
Formal methods: Model Checking

Peterson's algorithm

```
for n from 0 to N-1 exclusive
  level[i] ← n
  last_to_enter[n] ← i
  while last_to_enter[n] = i and there
        exists k ≠ i, such that level[k] ≥ n
    wait
```

Specification
Mutual exclusion

$$\bigwedge_{i \neq j} \neg(\mathsf{CS}_i \wedge \mathsf{CS}_j)$$



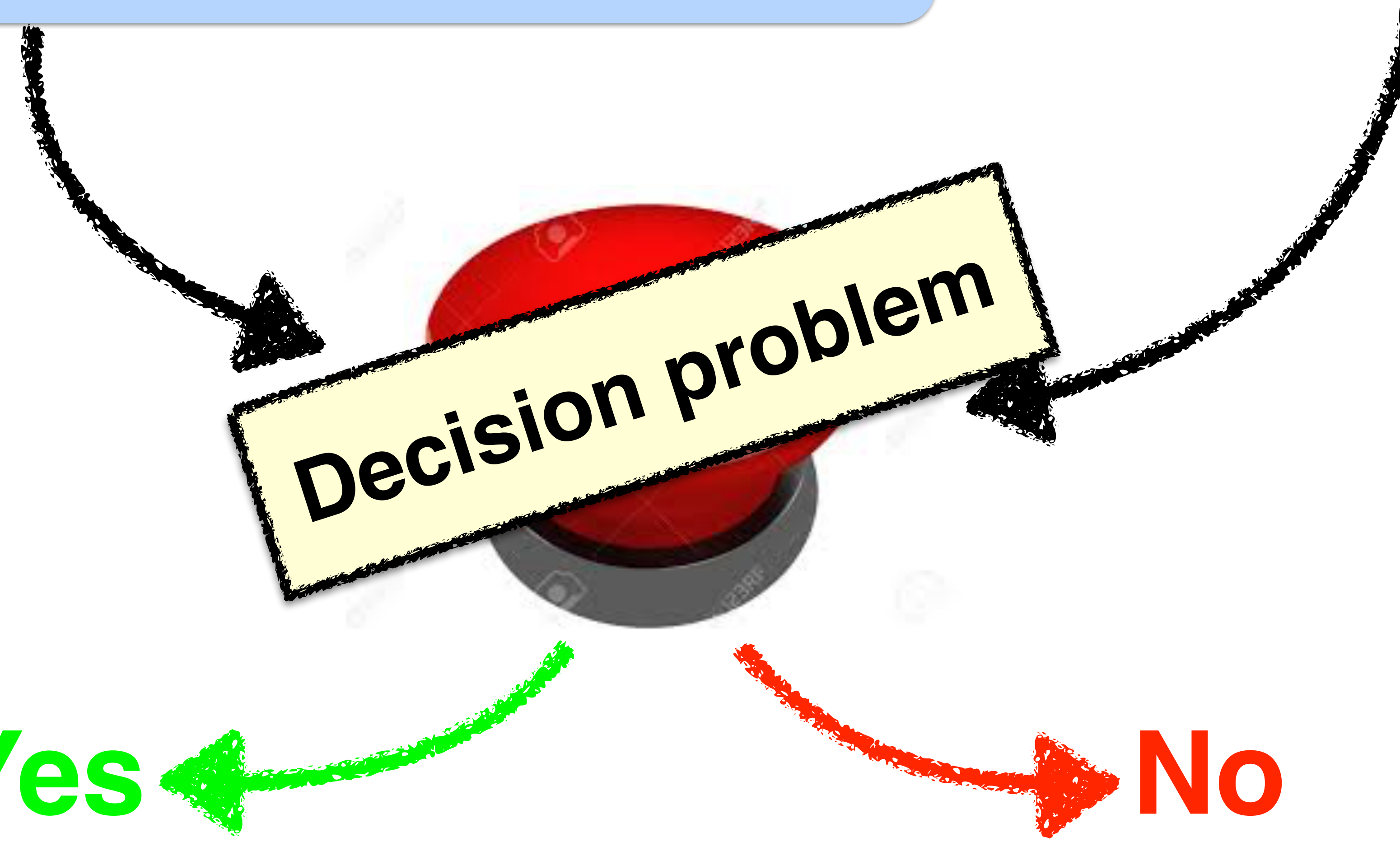
Formal methods: Model Checking

Peterson's algorithm

```
for n from 0 to N-1 exclusive
  level[i] ← n
  last_to_enter[n] ← i
  while last_to_enter[n] = i and there
       exists k ≠ i, such that level[k] ≥ n
    wait
```

Specification
Mutual exclusion

$$\bigwedge_{i \neq j} \neg(\mathsf{CS}_i \wedge \mathsf{CS}_j)$$



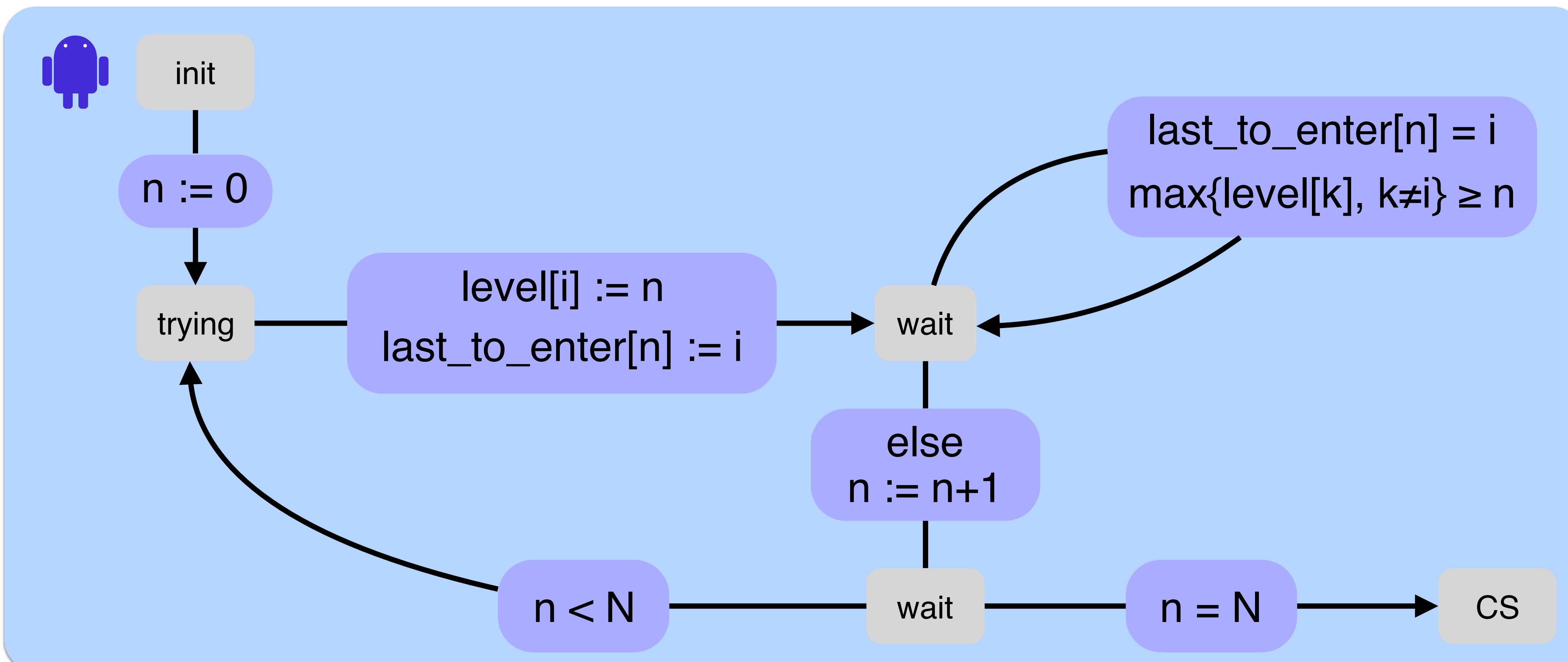
Models for programs/algorithms

- Finite state machine (control points)
- Data structures
  - Boolean variables
  - Integer variables
  - Stacks (recursivity)
  - Queues (asynchronous communication)

Models for programs/algorithms

Peterson's algorithm

```
for n from 0 to N-1 exclusive
  level[i] := n
  last_to_enter[n] := i
  while last_to_enter[n] = i and there
       exists k ≠ i, such that level[k] ≥ n
    wait
```

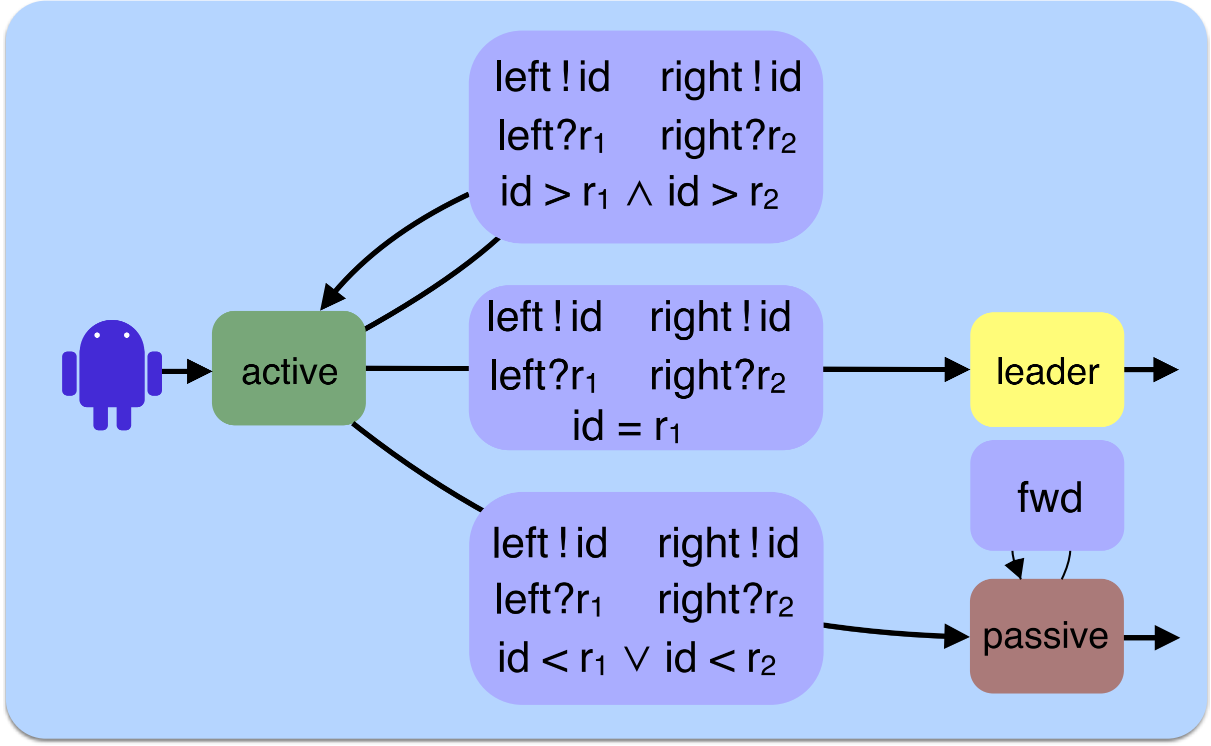


Models for programs/algorithms

Franklin's leader election algorithm

Processes are arranged in an undirected ring.
Each node has a unique identity.
Each node is either active or passive (relay mode) at a given time.

The algorithm executes as follows:
– Each active node sends its identity to its neighbors.
Let each active node p1 receive identities from p0 and p2. Where p0 and p2 are its either neighbors in the ring.
– If min(ID[p0], ID[p2]) > ID[p1], then p1 becomes passive
– If min(ID[p0], ID[p2]) < ID[p1], then p1 sends its ID to its neighbors again
– If min(ID[p0], ID[p2]) == ID[p1], then p1 declares itself as leader
– Passive nodes only pass on messages.
– The loop continues until a leader with highest unique ID has been elected.



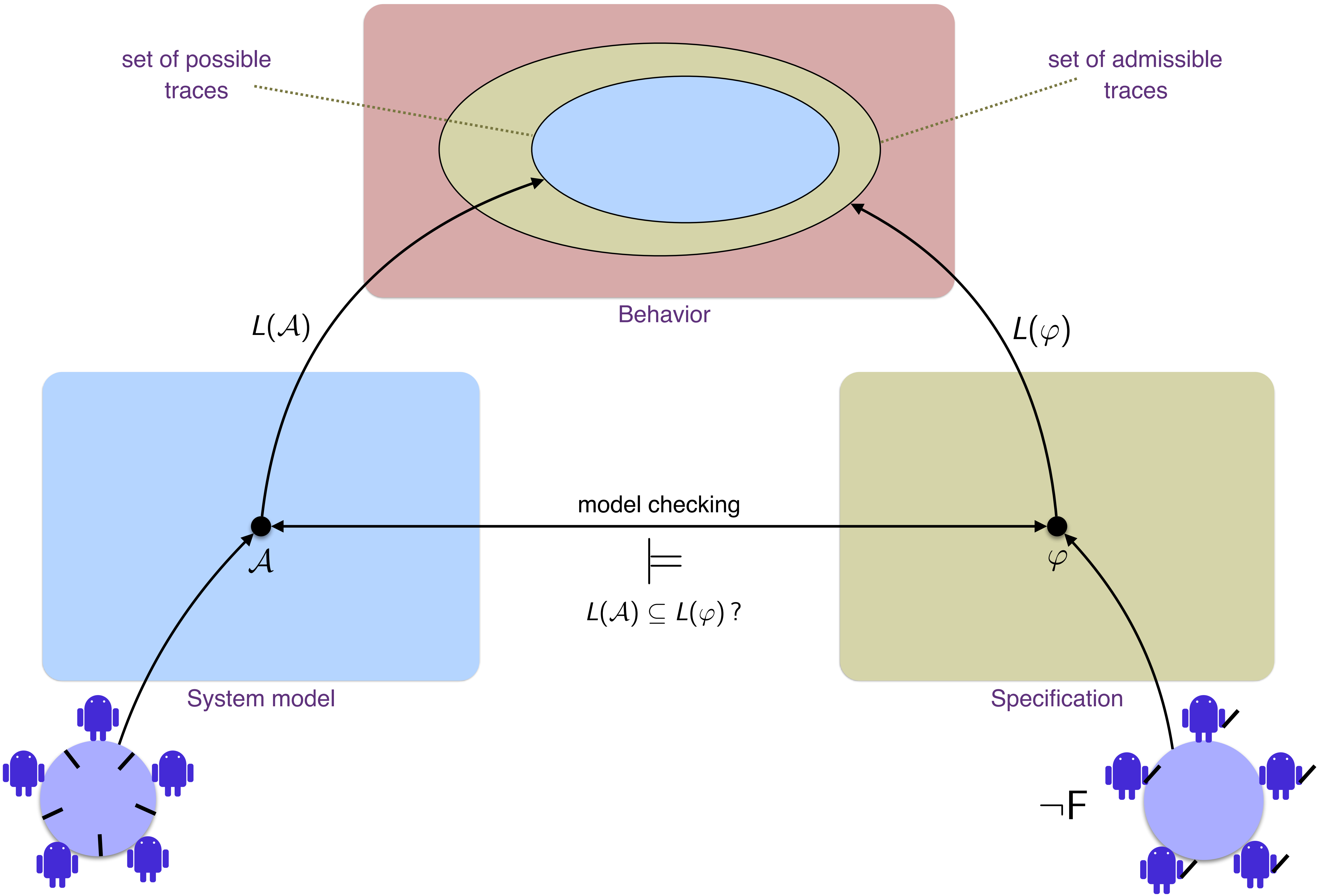
Languages for the specification

- Modal logics
- Temporal logics
- First-order logic
- Dynamic logics

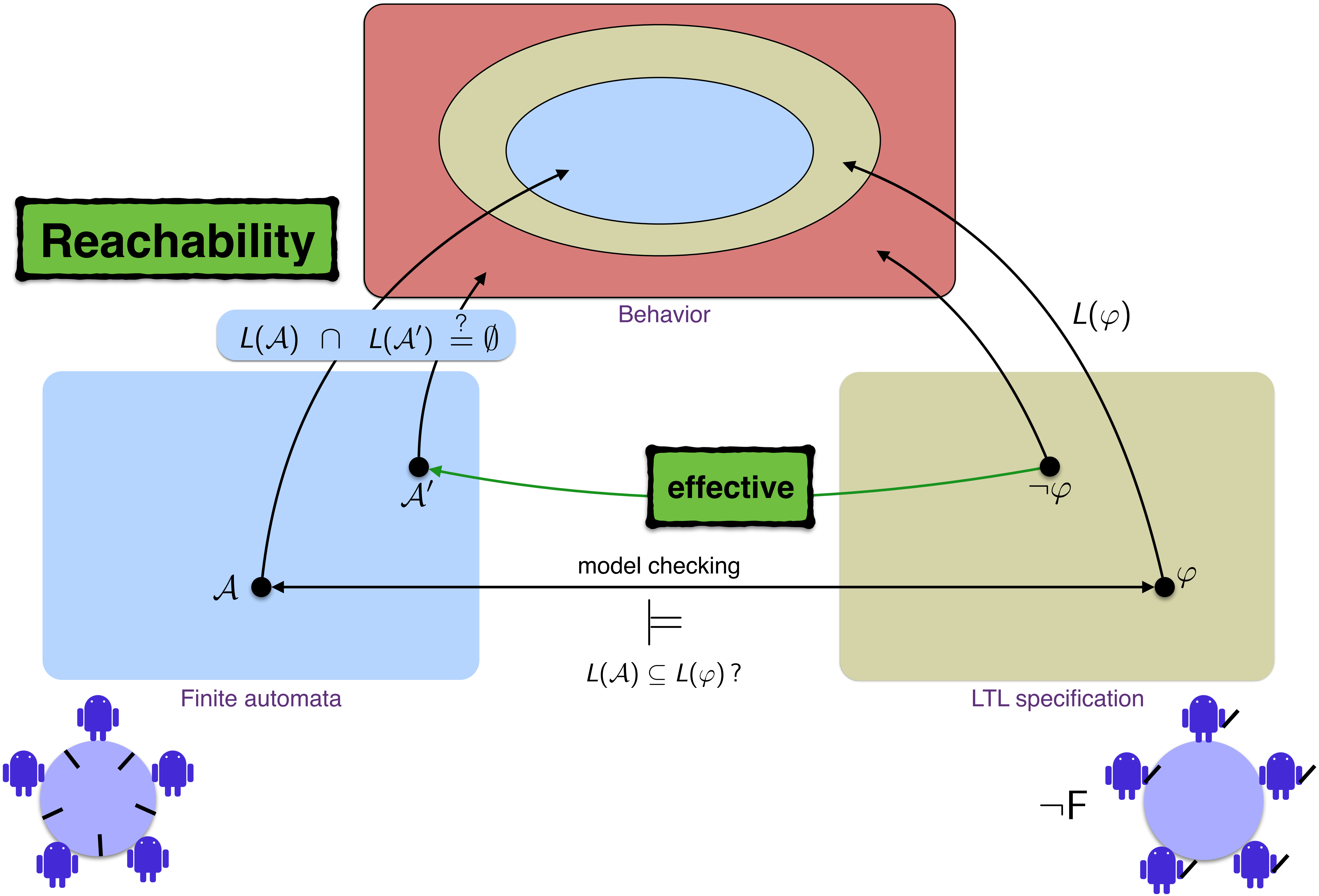
Model checking: sources of undecidability

- Each infinite/unbounded aspects
  - number of processes/agents
  - Integer variables (pids, timestamps, …)
  - FIFO channels (asynchronous communication)

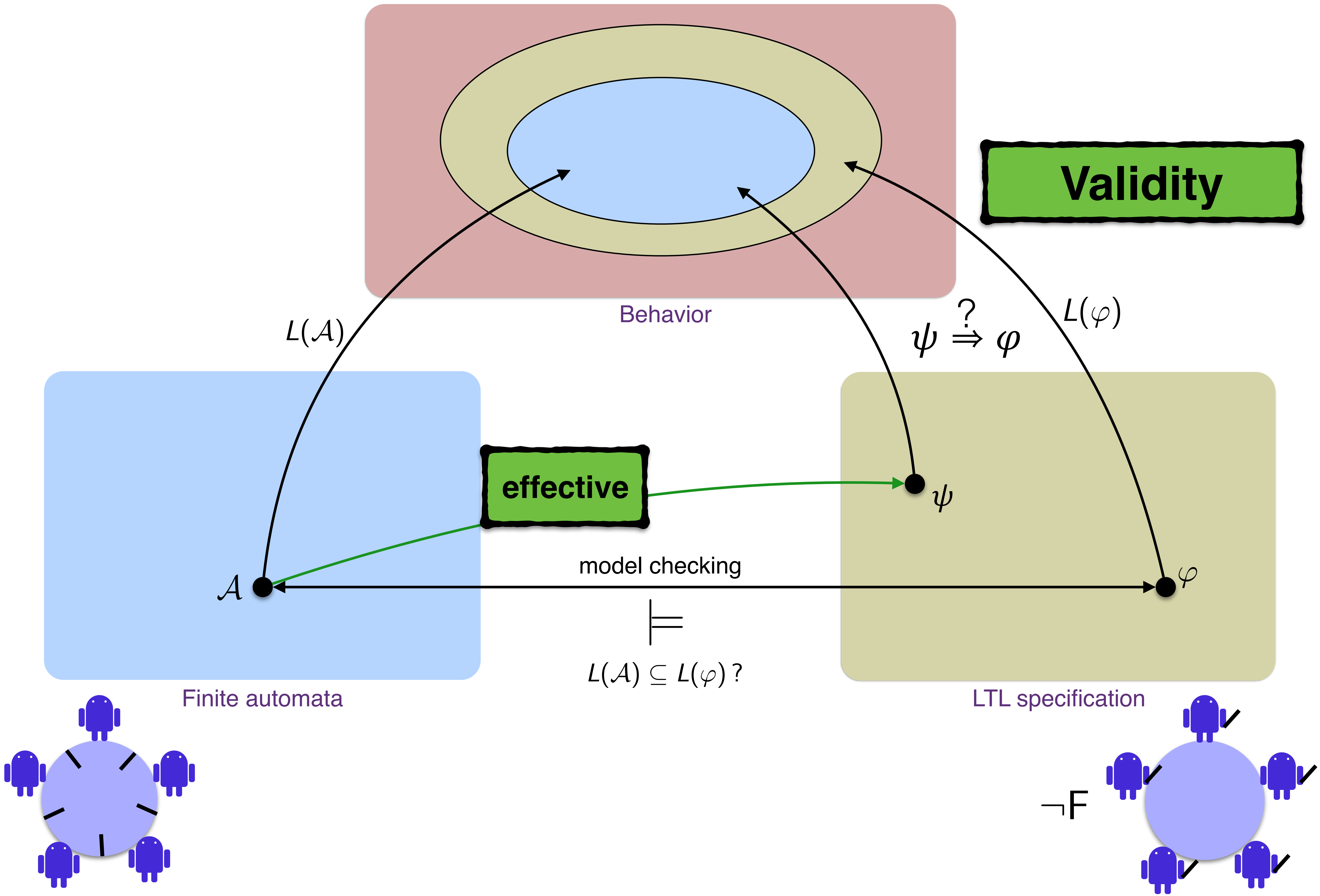
Model checking (Linear time)



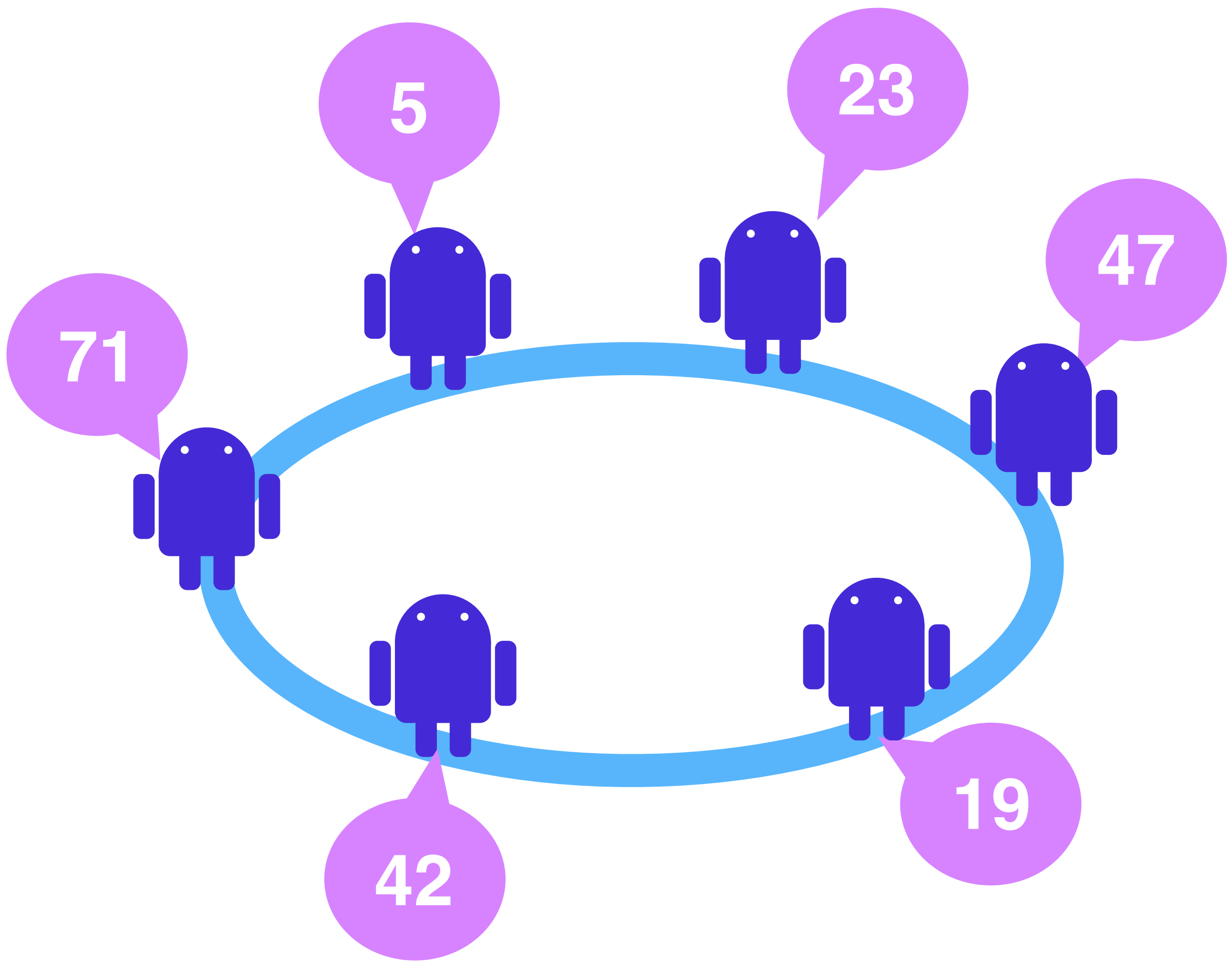
Model checking: First solution



Model checking: Second solution



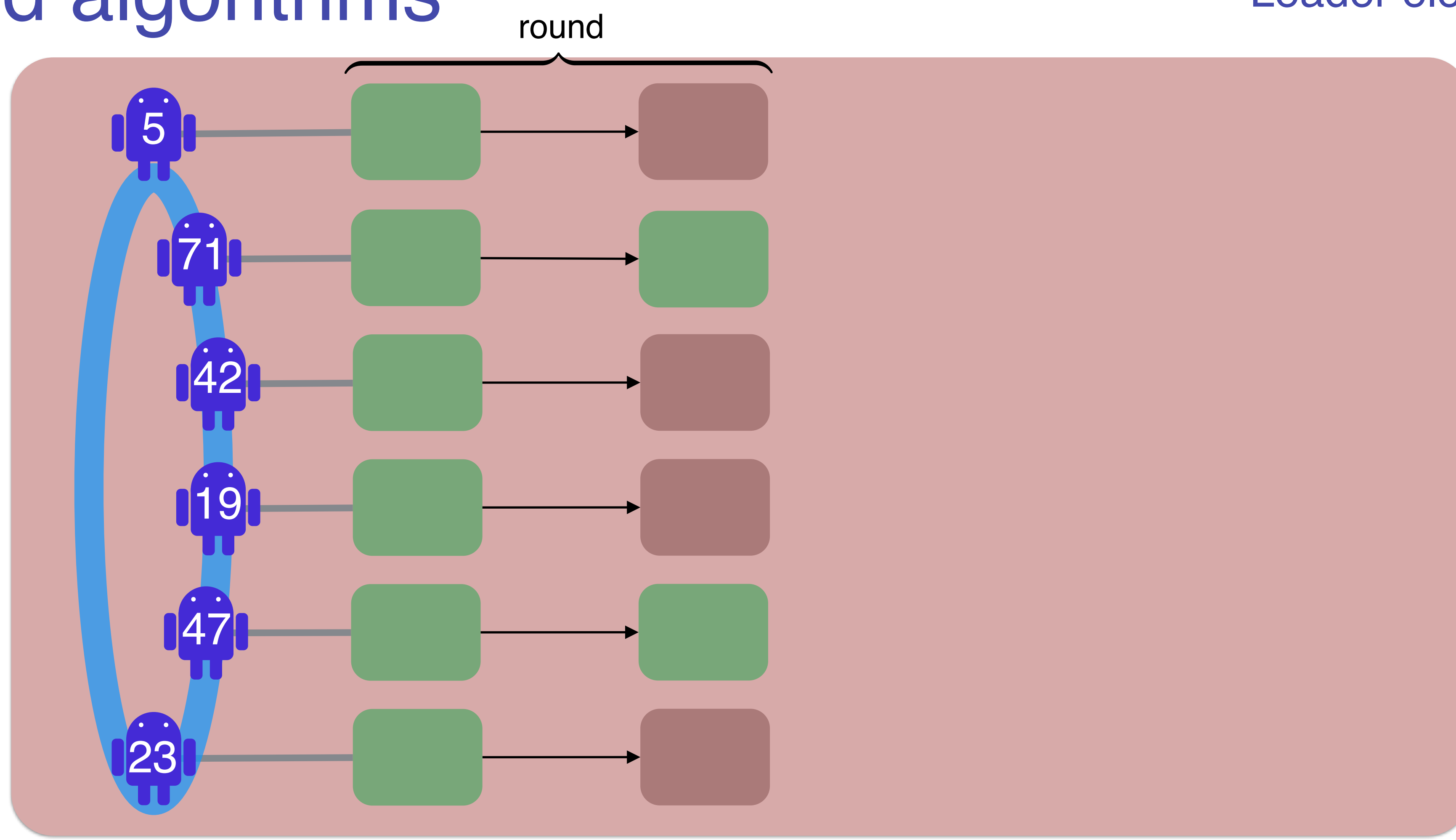
Models of Distributed Systems



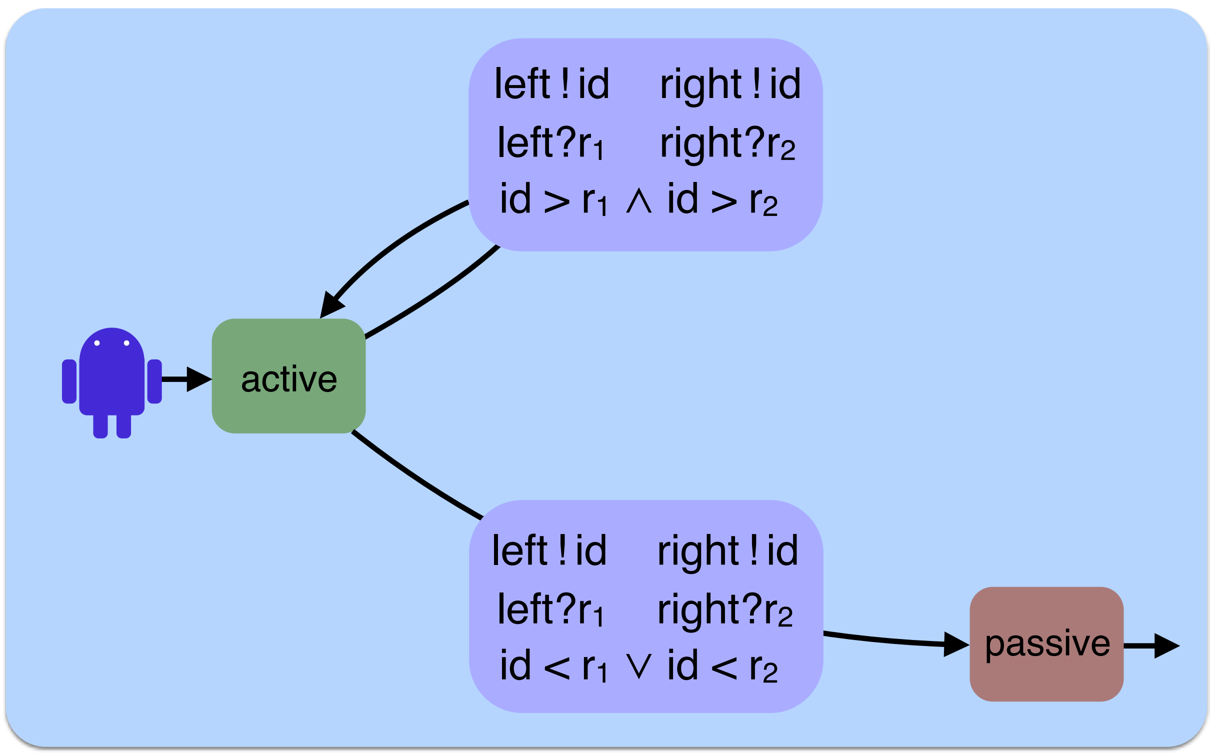
Distributed algorithms: our hypotheses

- Number of processes: arbitrary, unknown
- Unique process identification
  - Comparisons: <, =
  - No arithmetic
- Topology: fixed degree (ring, …)
- Communication: Synchronous in rounds
  - Round: send messages, receive messages, compute and update local registers

Distributed algorithms



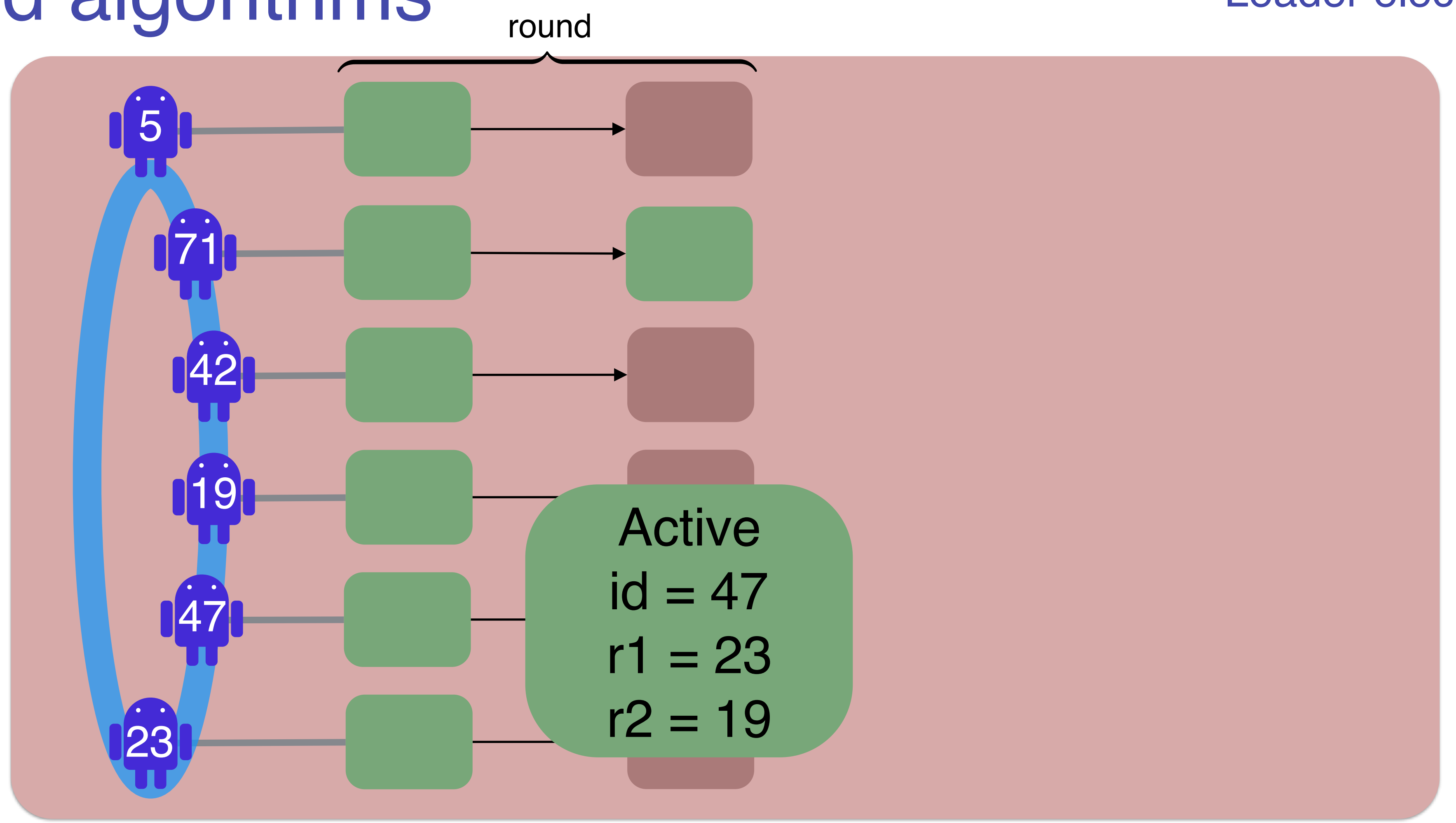
Behavior



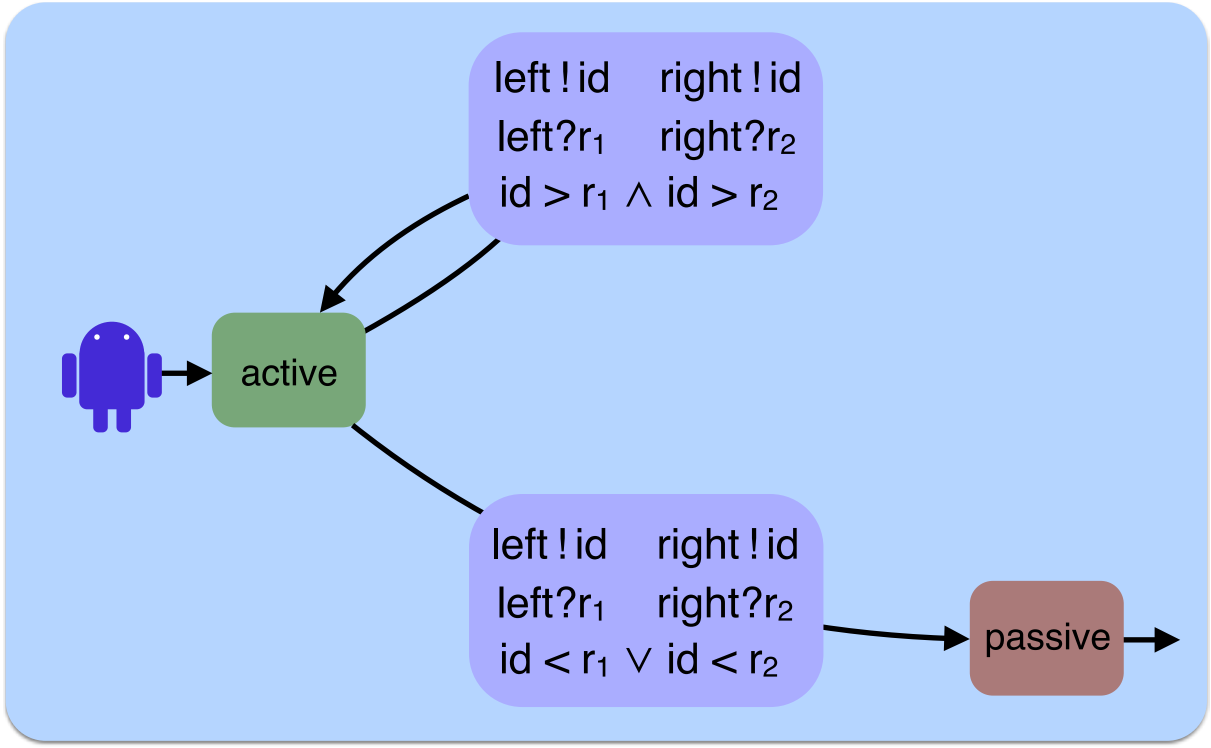
Distributed algorithm

Distributed algorithms

Leader election [Franklin ’82]

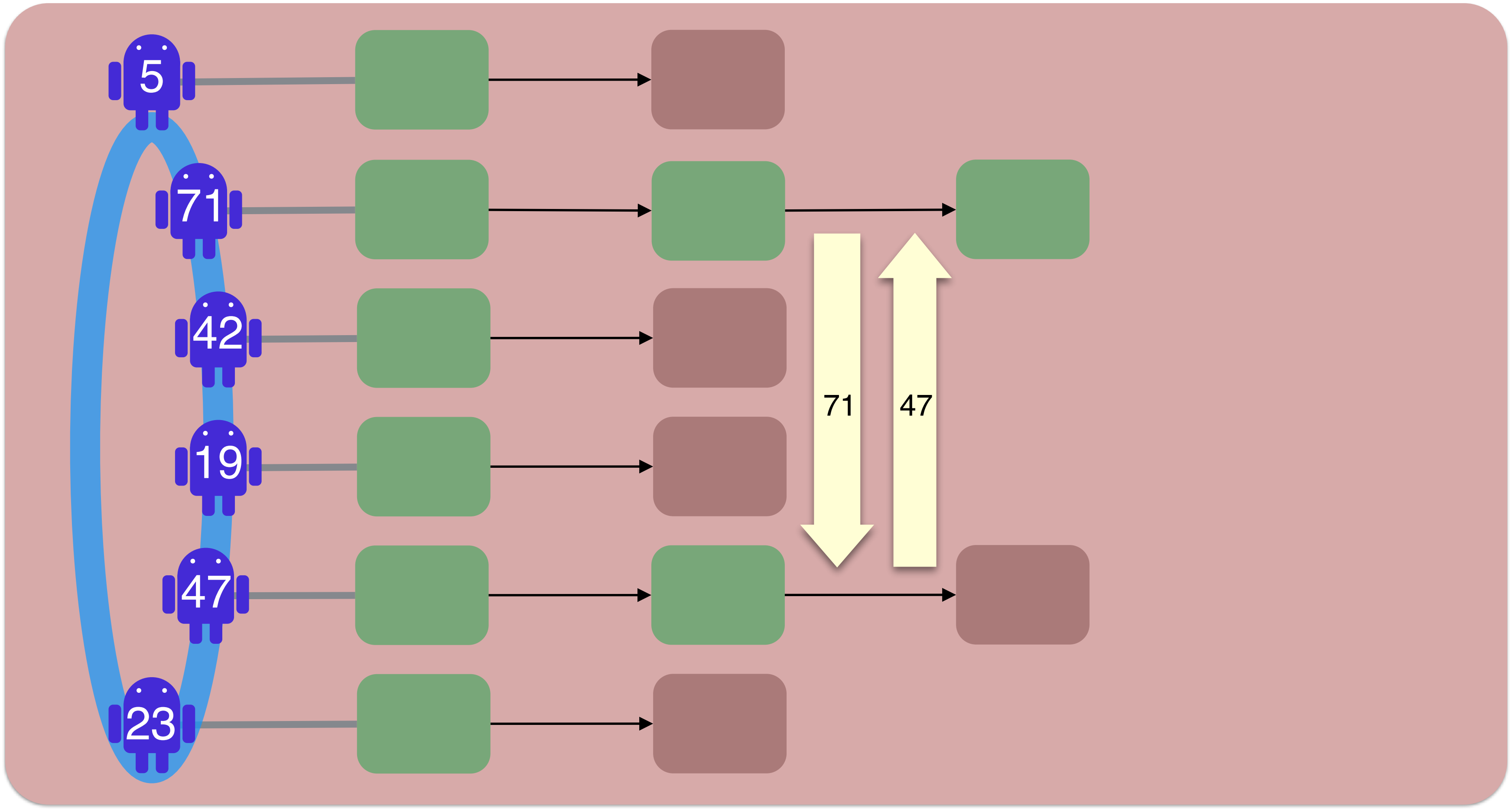


Behavior

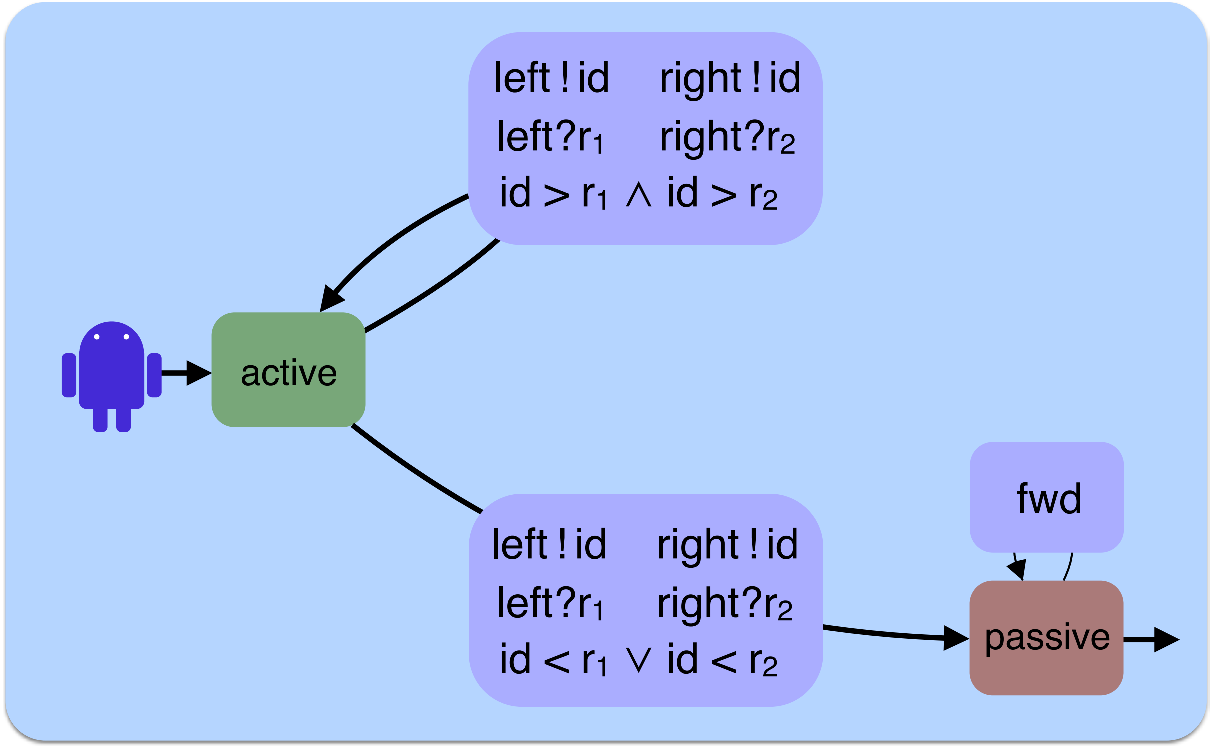


Distributed algorithm

Distributed algorithms

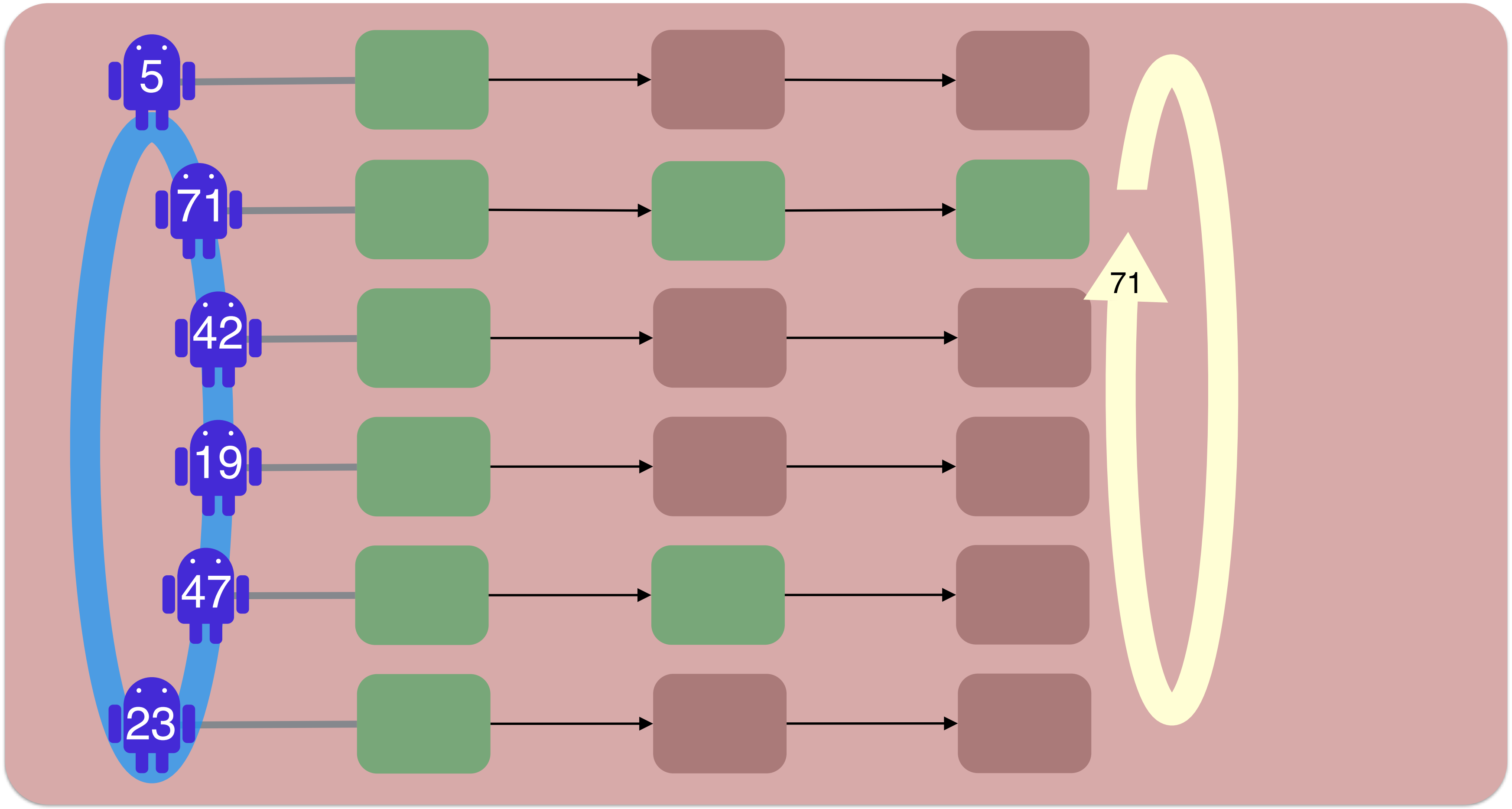


Behavior

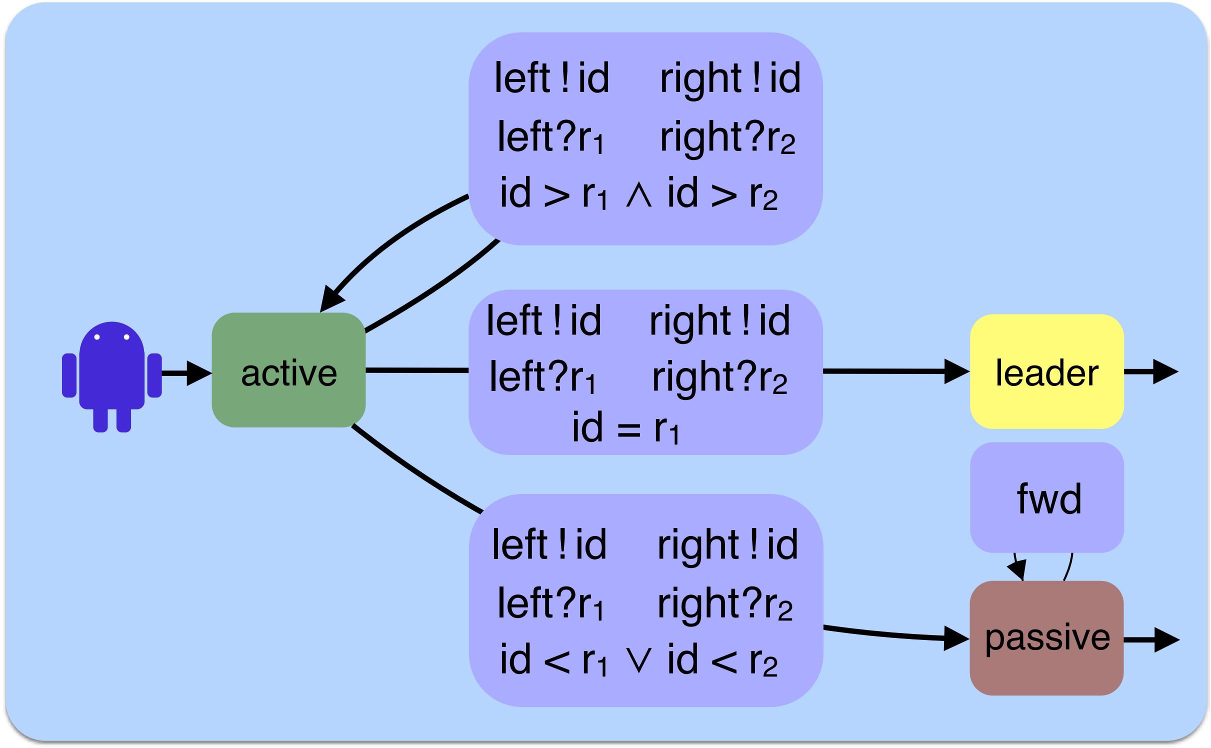


Distributed algorithm

Distributed algorithms



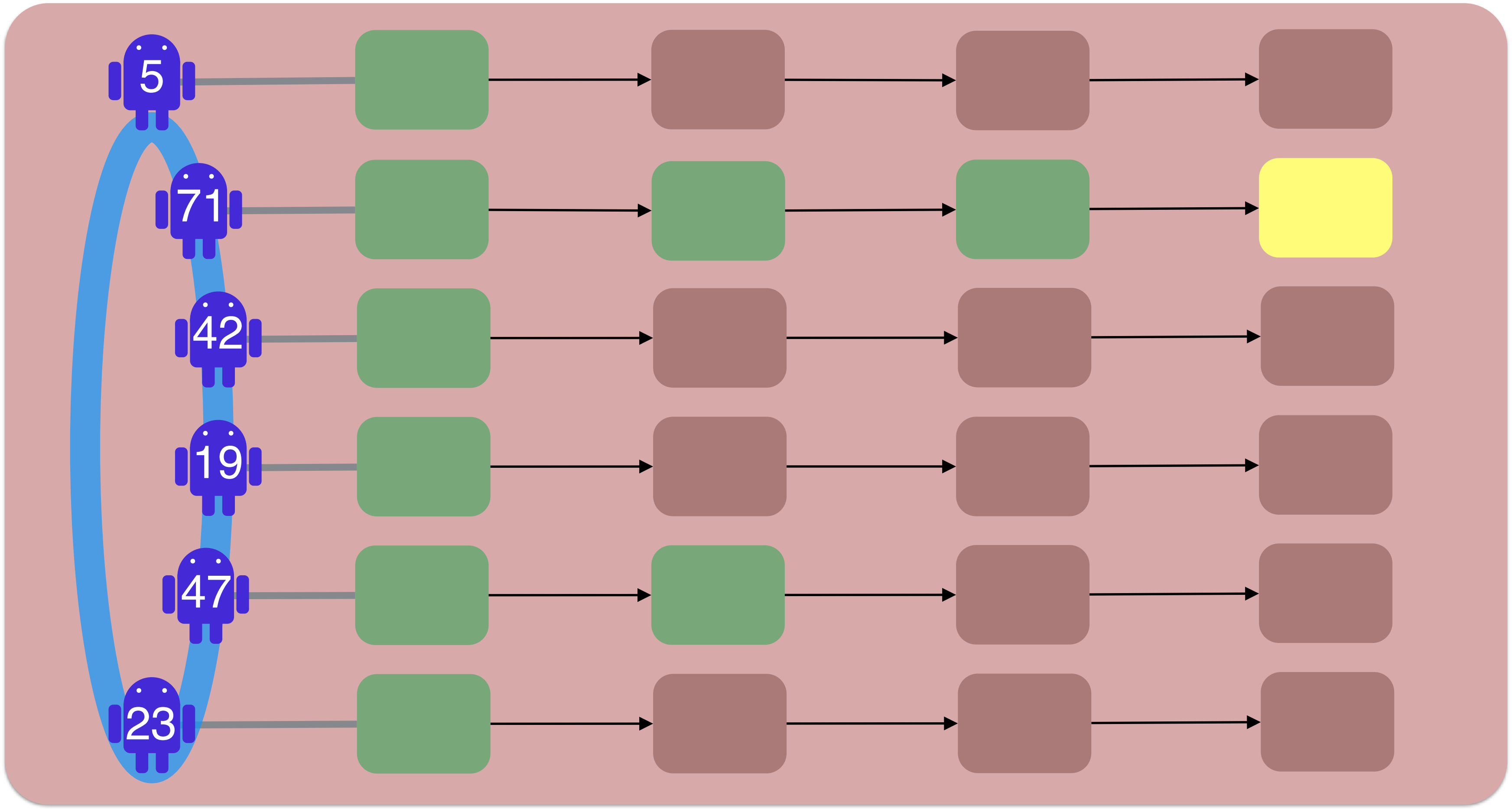
Behavior



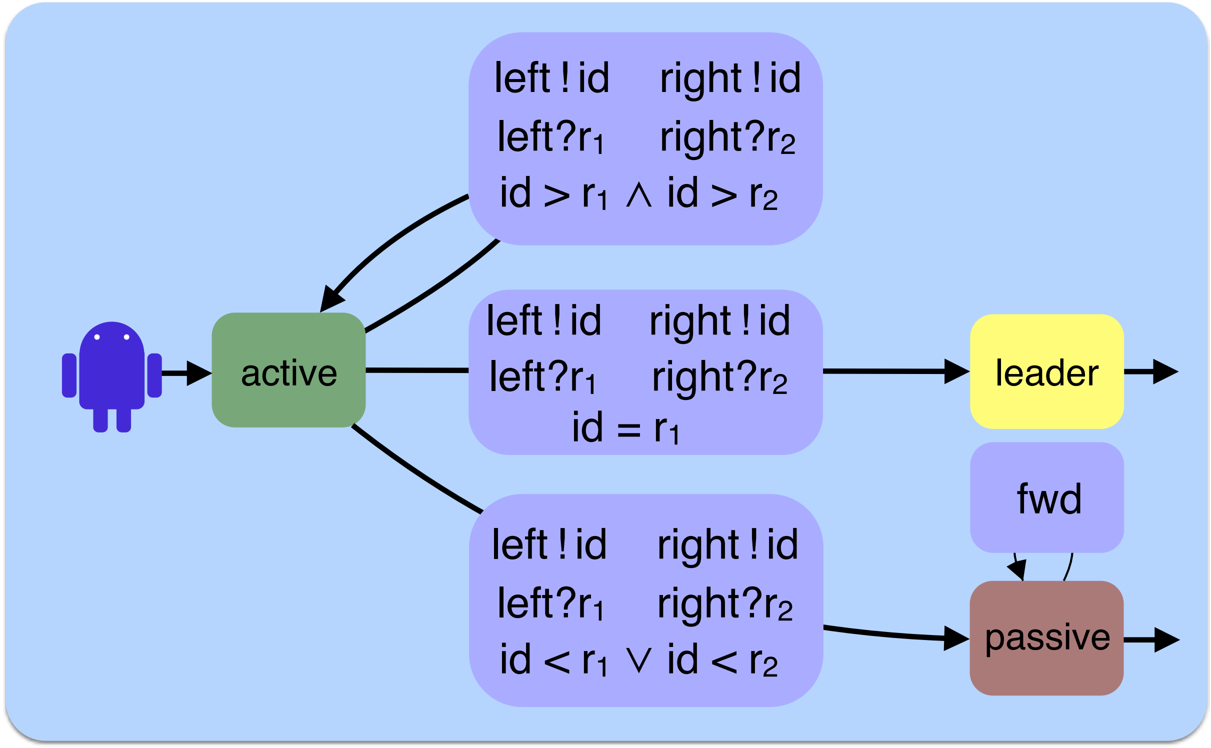
Distributed algorithm

Distributed algorithms

Leader election [Franklin '82]

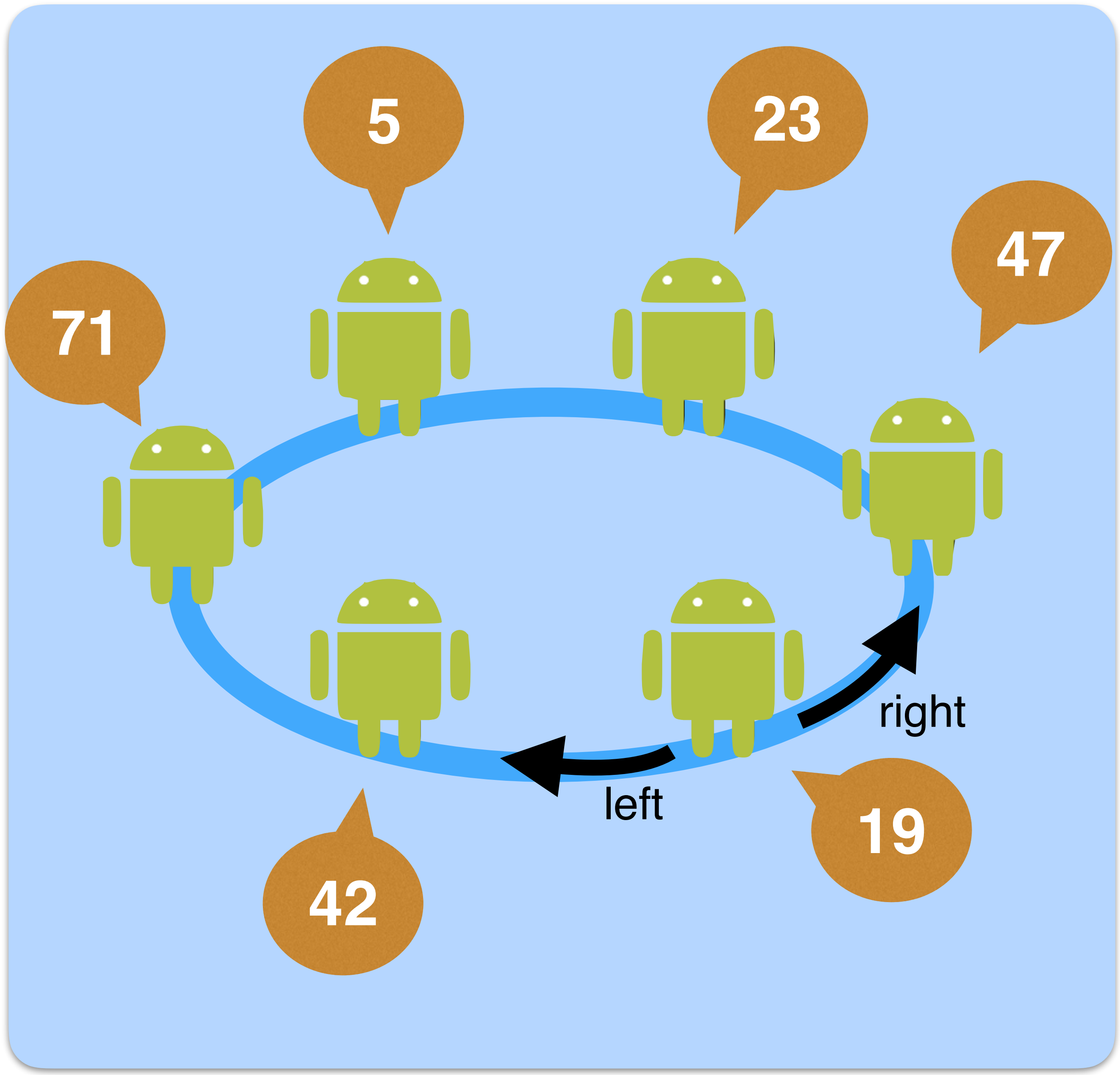


Behavior



Distributed algorithm

Distributed algorithms



- Identical finite-state processes
- Number of processes is unknown and unbounded
- Processes have unique pids (integers — unbounded data)

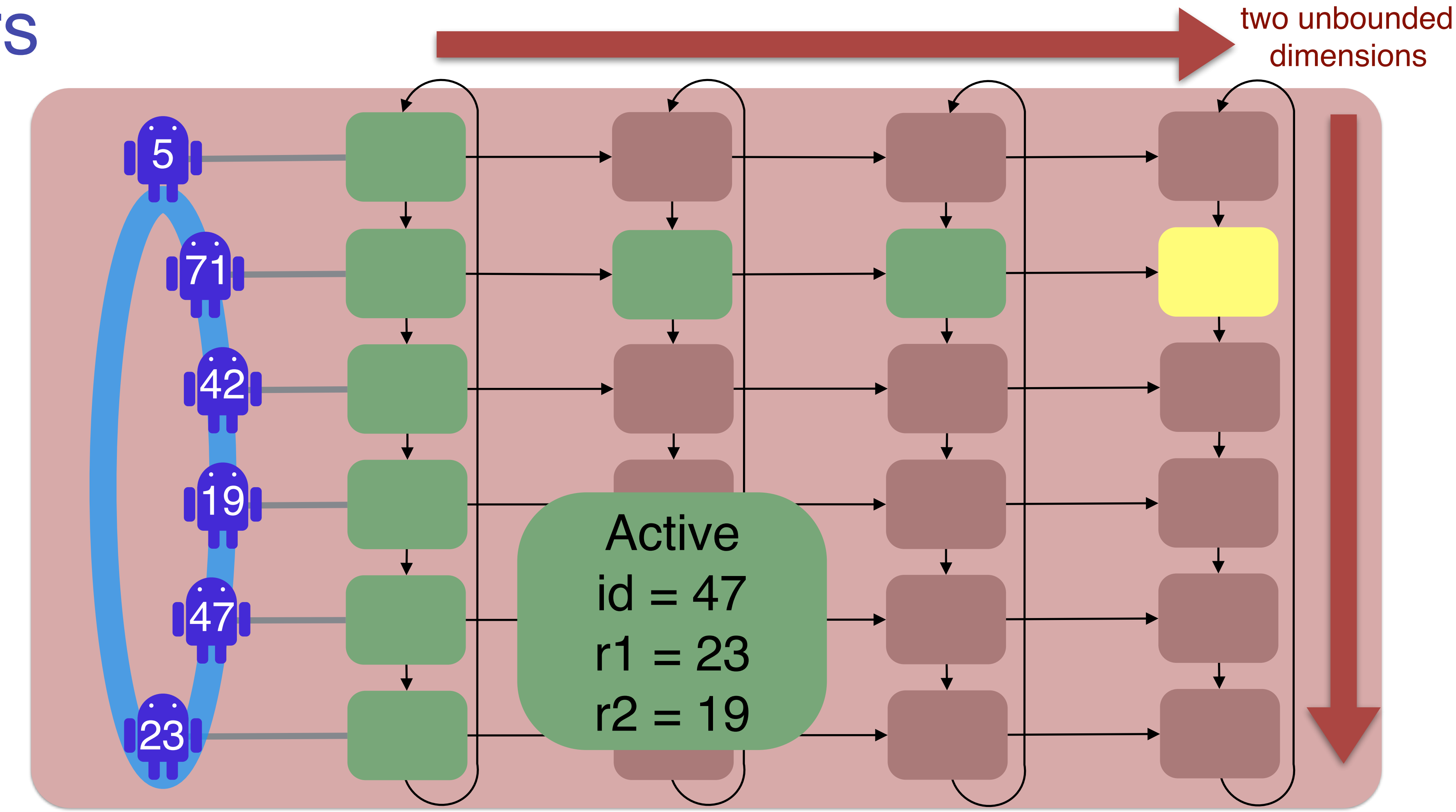
A formal model for distributed algorithms

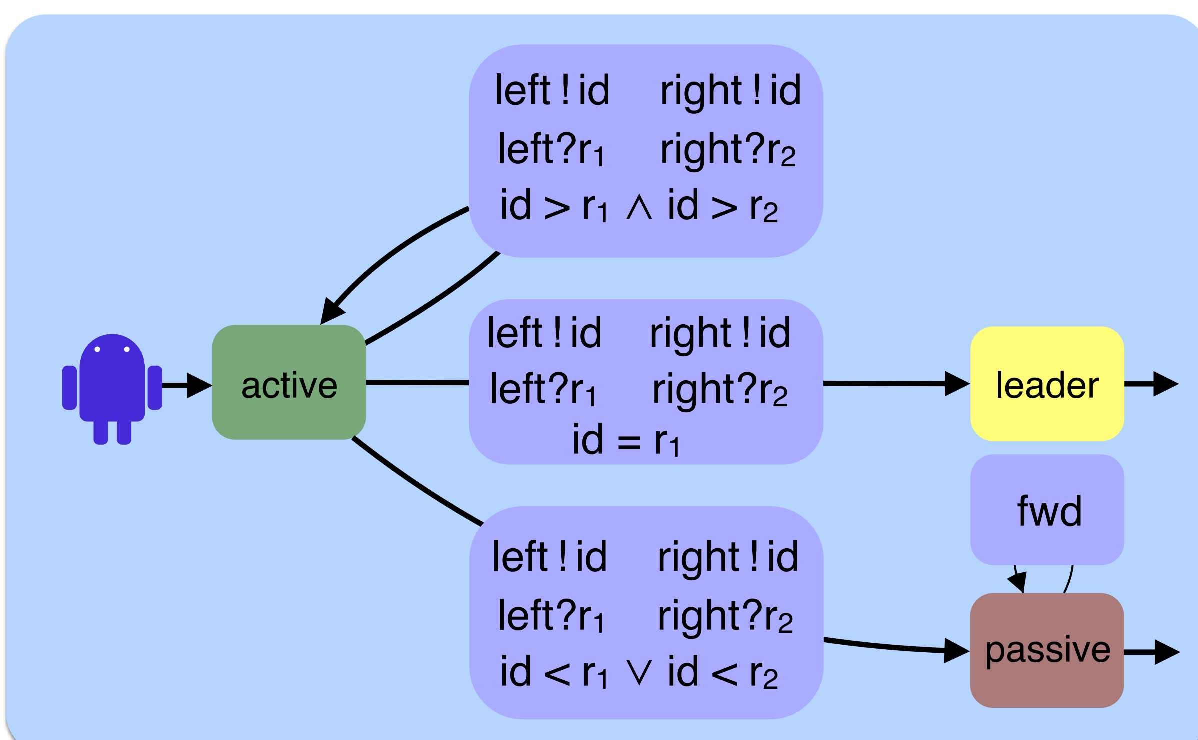
An automata-like way of writing DA

Every process can be described by:

- Set of states
- Initial state
- Set of registers
  - stores pid
- Set of transitions
  - send pids to neighbours
  - receive pids from neighbours, and store in registers
  - compare registers
  - update registers

Behaviors



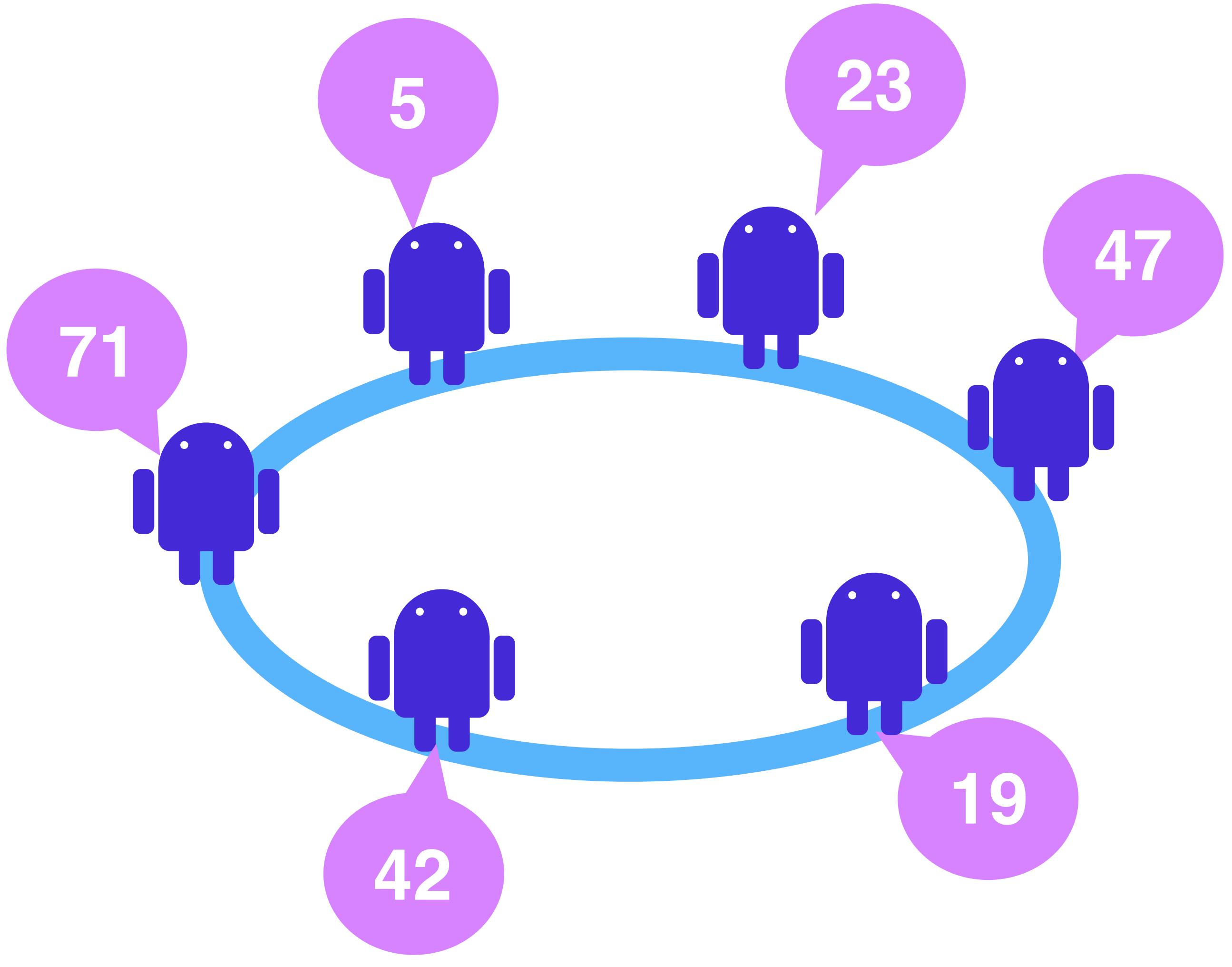


Distributed algorithm

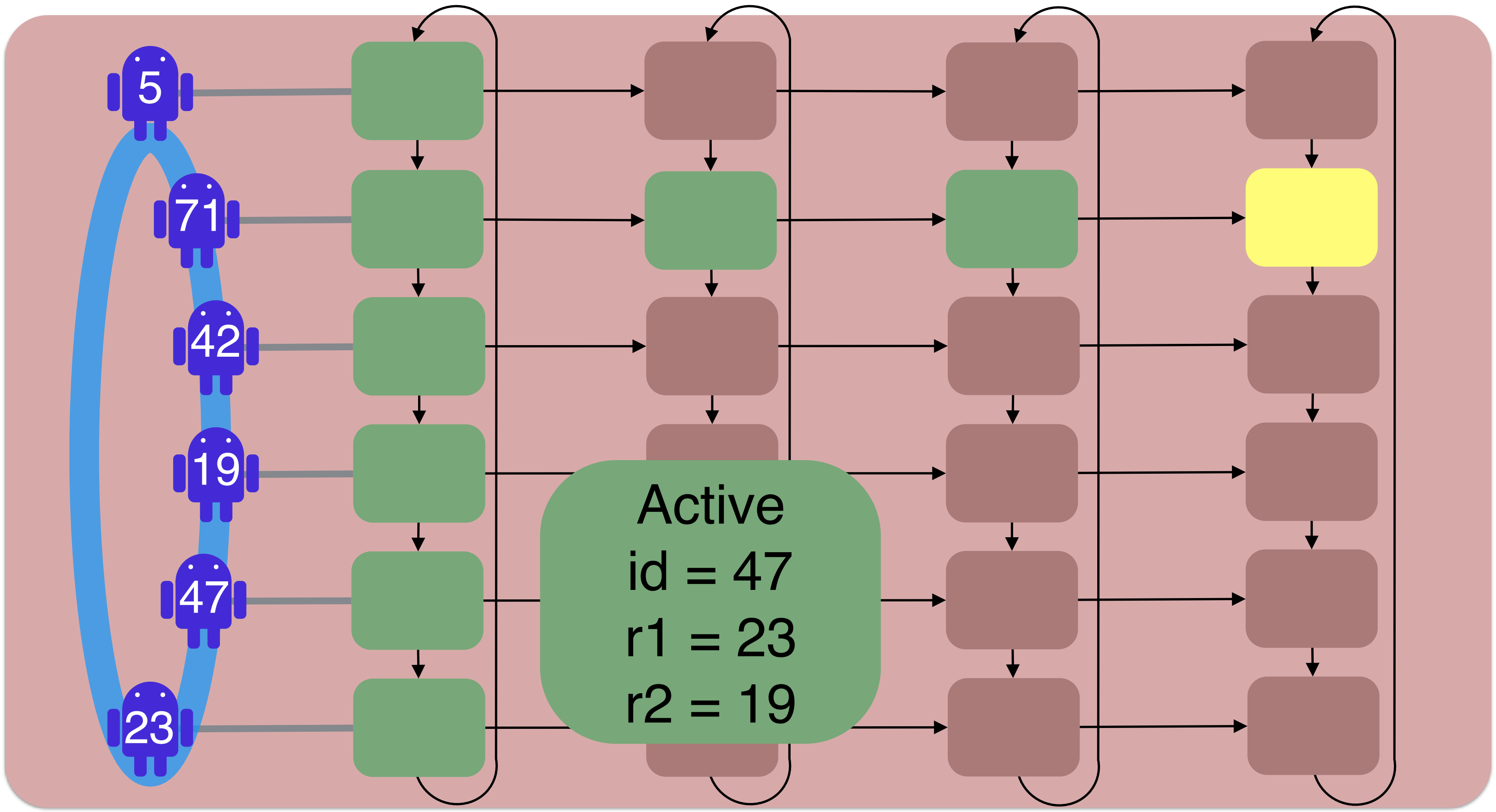
Cylinders
Arbitrary length and width
Labelled with data
from an infinite domain

3 sources of infinity

Abstraction of Data Values

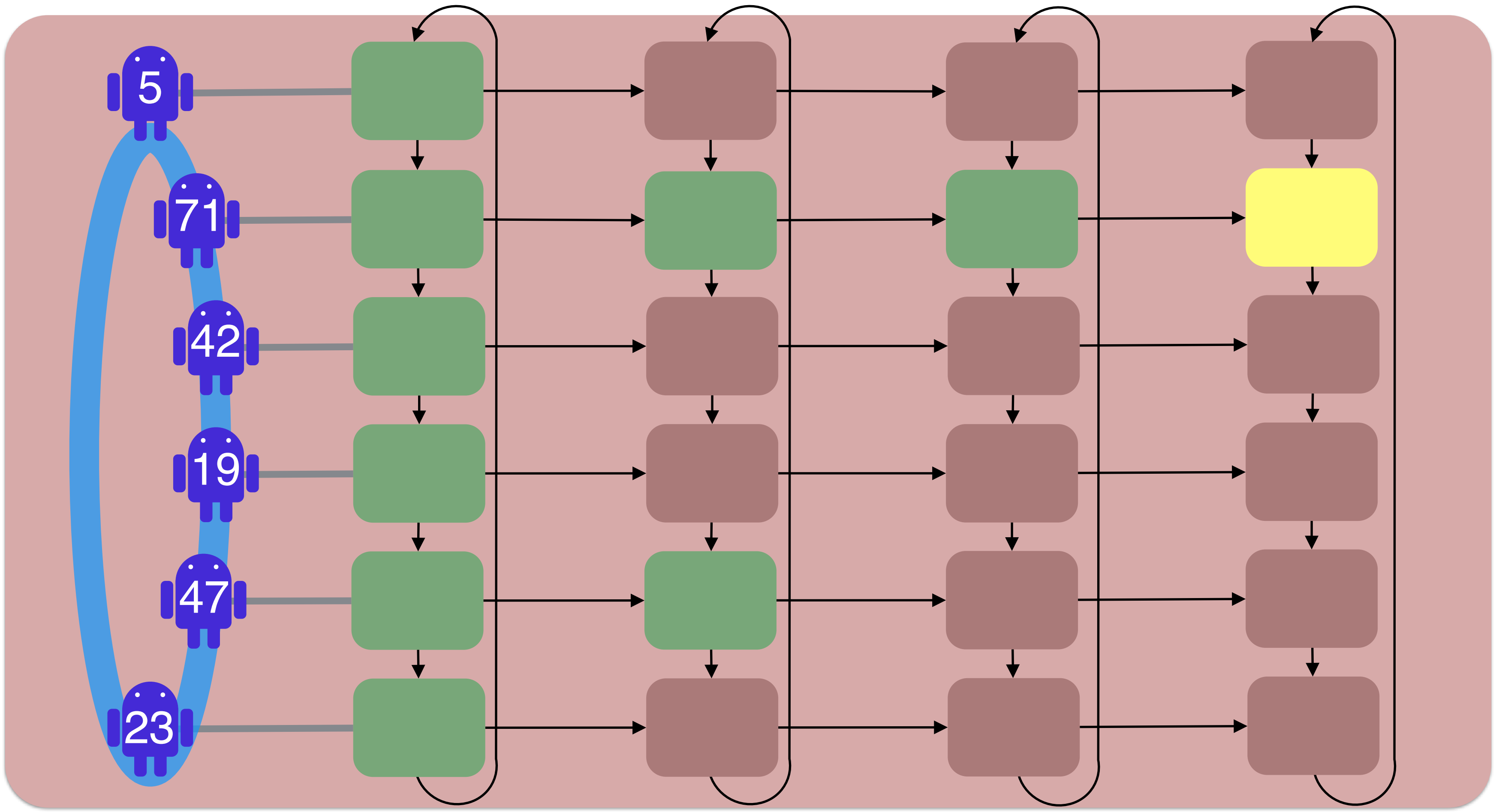


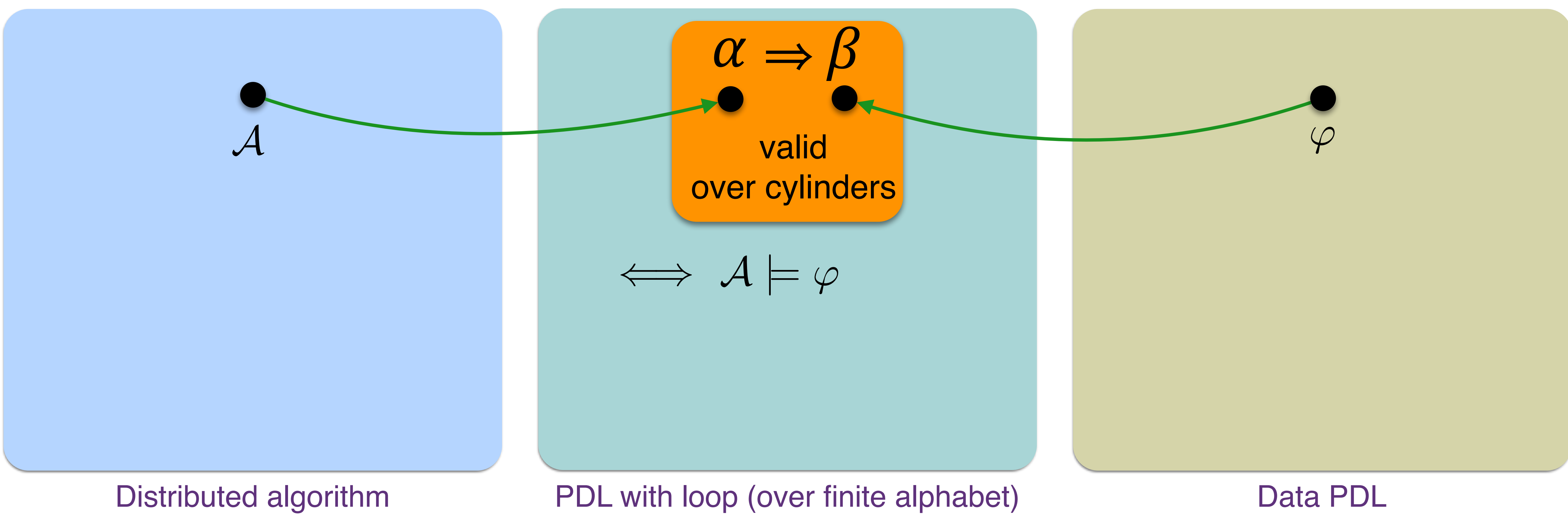
Model Checking Distributed algorithms



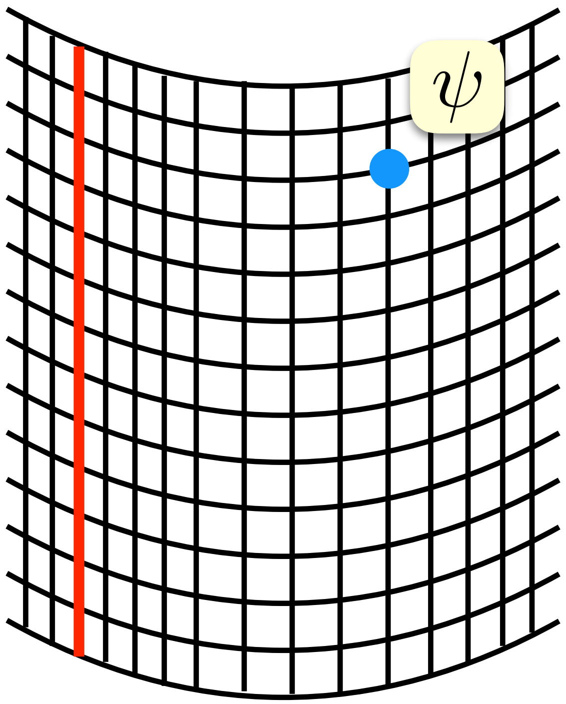
- Behaviors: Cylinders of arbitrary width and length
  Data from an infinite domain
- System: Register automata with data comparisons
- Specification: Data PDL with data comparisons

Reduction to Satisfiability of LCPDL: Data abstraction





LCPDL: Propositional Dynamic logic with Loop and Converse

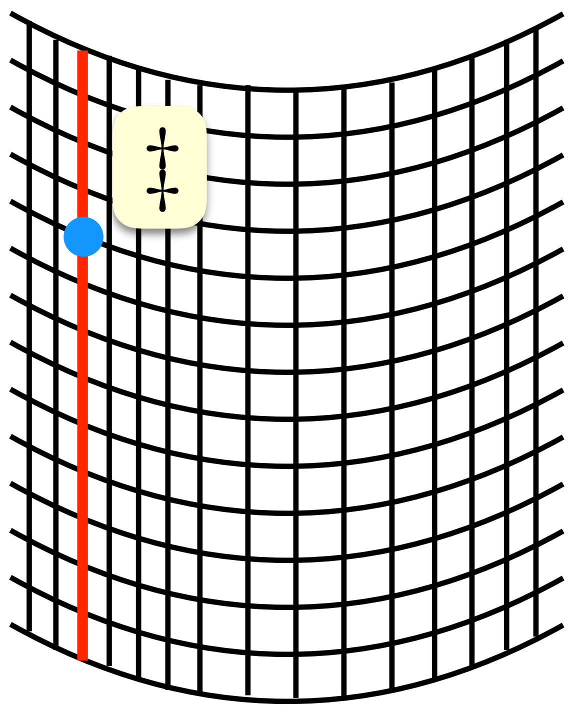


$$\Psi, \Psi' ::= \mathsf{E}\,\psi \mid \neg\Psi \mid \Psi \wedge \Psi'$$

$$\psi, \psi' ::= \ddagger \mid p \mid \neg\psi \mid \psi \wedge \psi' \mid \langle\pi\rangle\psi \mid \mathsf{loop}(\pi)$$

$$\pi, \pi' ::= \{\psi\}? \mid \rightarrow \mid \downarrow \mid \pi + \pi' \mid \pi \cdot \pi' \mid \pi^{*} \mid \pi^{-1}$$

LCPDL: Propositional Dynamic logic with Loop and Converse

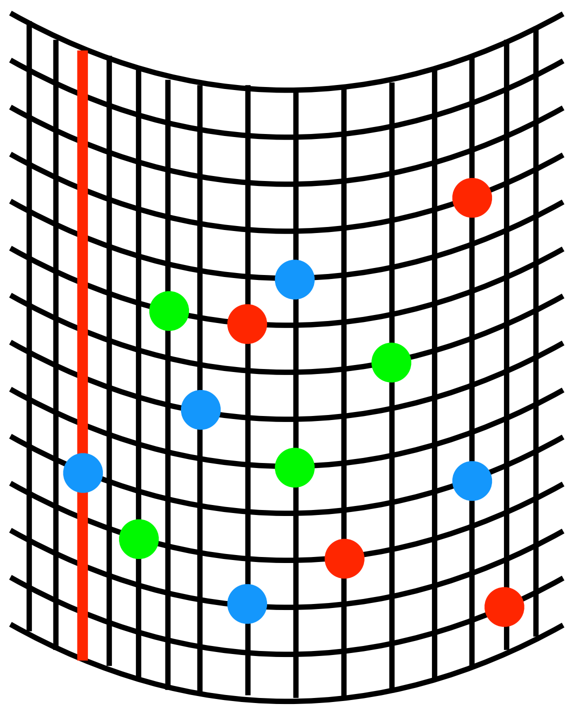


$$\Psi, \Psi' ::= \mathsf{E}\,\psi \mid \neg\Psi \mid \Psi \wedge \Psi'$$

$$\psi, \psi' ::= \ddagger \mid p \mid \neg\psi \mid \psi \wedge \psi' \mid \langle\pi\rangle\psi \mid \mathsf{loop}(\pi)$$

$$\pi, \pi' ::= \{\psi\}? \mid \rightarrow \mid \downarrow \mid \pi + \pi' \mid \pi \cdot \pi' \mid \pi^* \mid \pi^{-1}$$

LCPDL: Propositional Dynamic logic with Loop and Converse

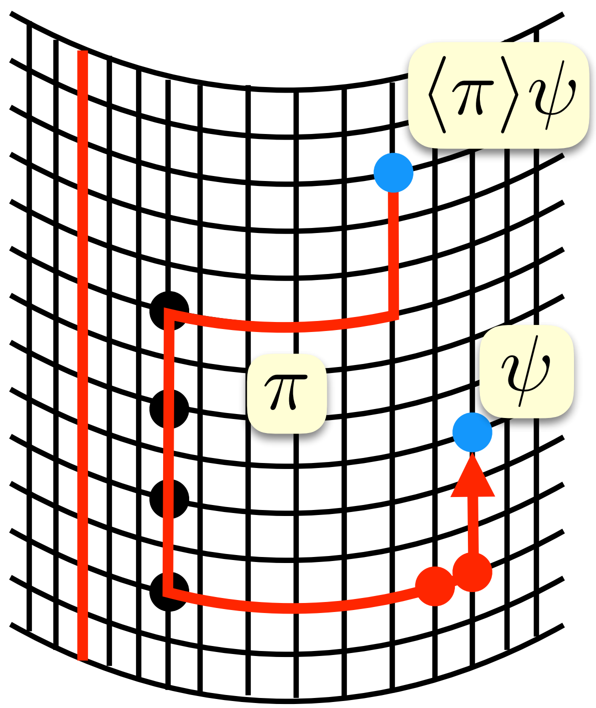


$$\Psi, \Psi' ::= \mathsf{E}\,\psi \mid \neg\Psi \mid \Psi \wedge \Psi'$$

$$\psi, \psi' ::= \ddagger \mid p \mid \neg\psi \mid \psi \wedge \psi' \mid \langle\pi\rangle\psi \mid \mathsf{loop}(\pi)$$

$$\pi, \pi' ::= \{\psi\}? \mid \rightarrow \mid \downarrow \mid \pi + \pi' \mid \pi \cdot \pi' \mid \pi^* \mid \pi^{-1}$$

LCPDL: Propositional Dynamic logic with Loop and Converse



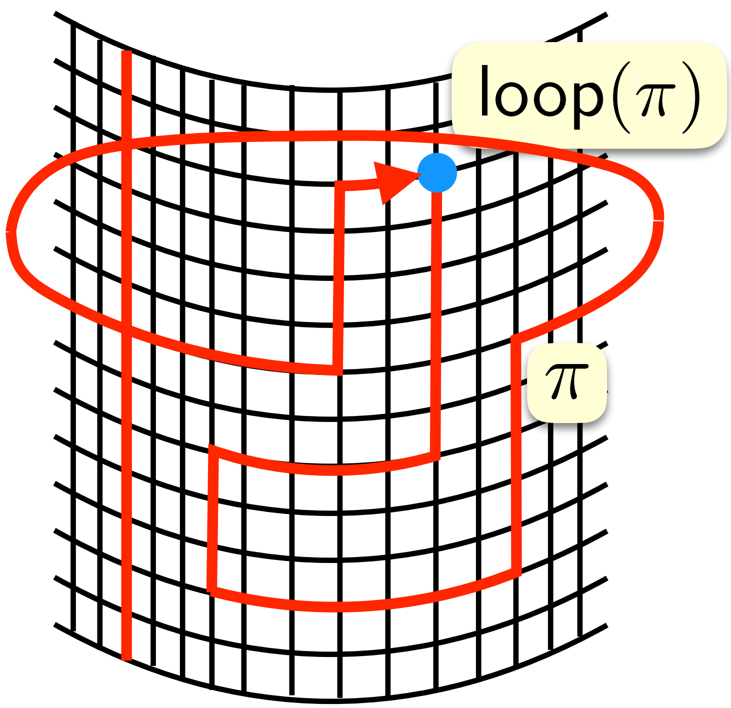
$$\langle \downarrow^* \leftarrow^* \{\bullet\}?(\downarrow\downarrow\{\bullet\}?)^* \rightarrow^* \{\bullet\}? \rightarrow \{\bullet\}? \uparrow^* \rangle \bullet$$

$$\Psi, \Psi' ::= \mathsf{E}\,\psi \mid \neg\Psi \mid \Psi \wedge \Psi'$$

$$\psi, \psi' ::= \ddagger \mid p \mid \neg\psi \mid \psi \wedge \psi' \mid \langle\pi\rangle\psi \mid \mathsf{loop}(\pi)$$

$$\pi, \pi' ::= \{\psi\}? \mid \rightarrow \mid \downarrow \mid \pi + \pi' \mid \pi \cdot \pi' \mid \pi^* \mid \pi^{-1}$$

LCPDL: Propositional Dynamic logic with Loop and Converse

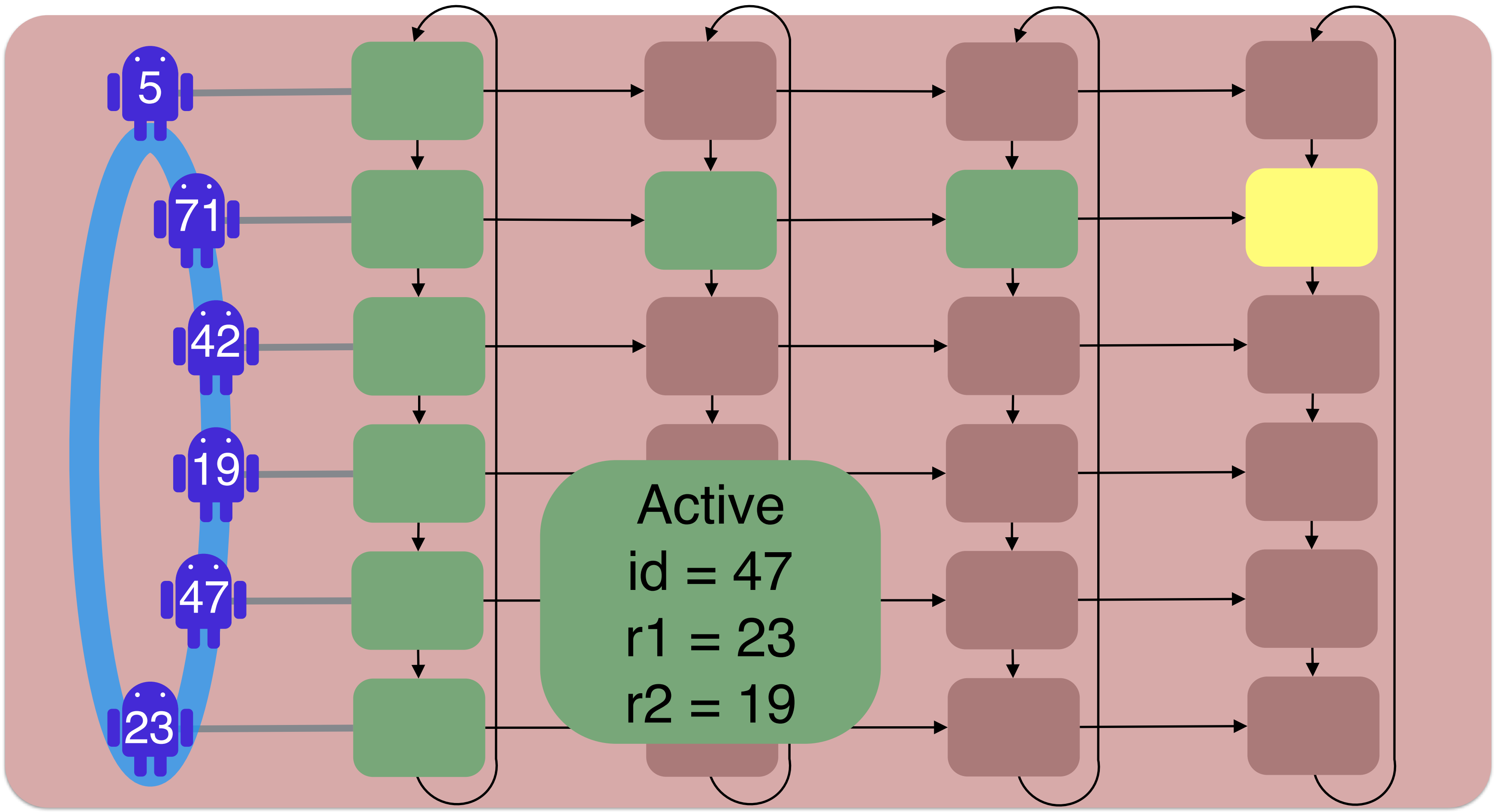


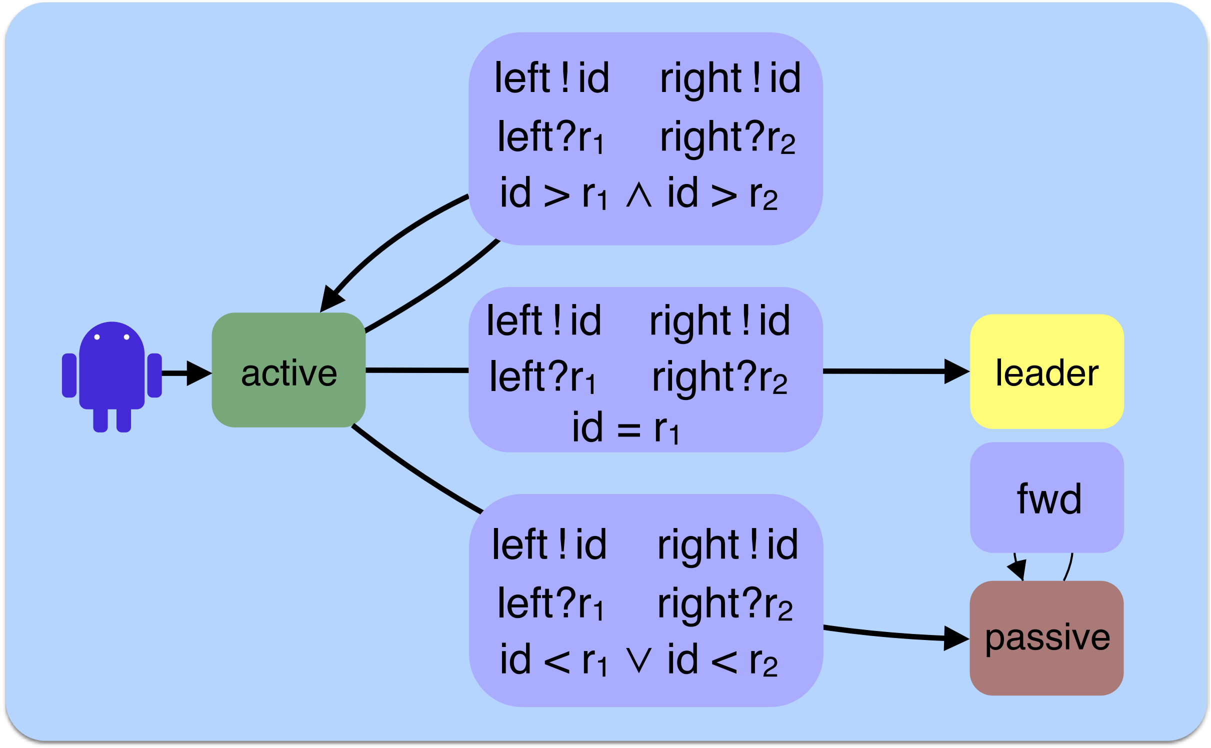
$$\Psi, \Psi' ::= \mathsf{E}\,\psi \mid \neg\Psi \mid \Psi \wedge \Psi'$$

$$\psi, \psi' ::= \ddagger \mid p \mid \neg\psi \mid \psi \wedge \psi' \mid \langle\pi\rangle\psi \mid \mathsf{loop}(\pi)$$

$$\pi, \pi' ::= \{\psi\}? \mid \rightarrow \mid \downarrow \mid \pi + \pi' \mid \pi \cdot \pi' \mid \pi^* \mid \pi^{-1}$$

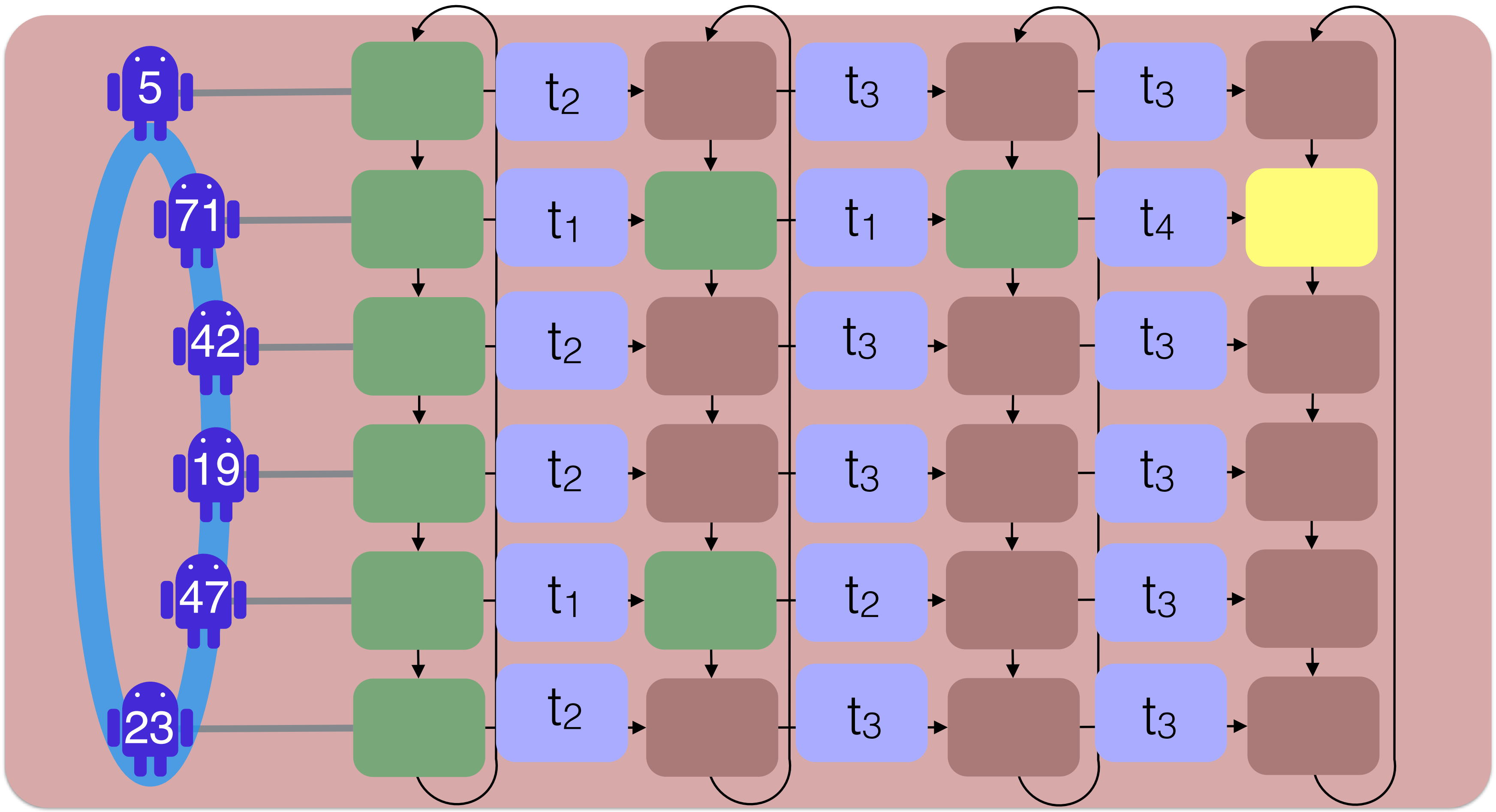
Data abstraction: symbolic runs + tracking data

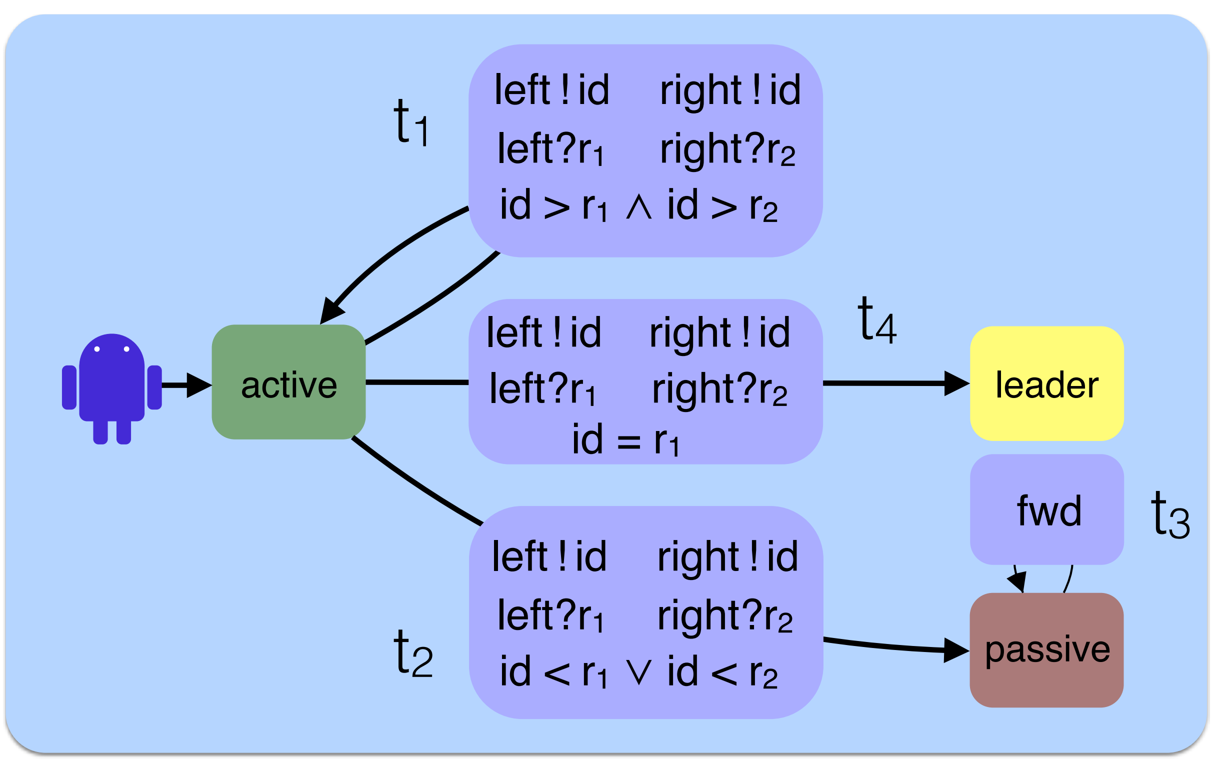




Distributed algorithm

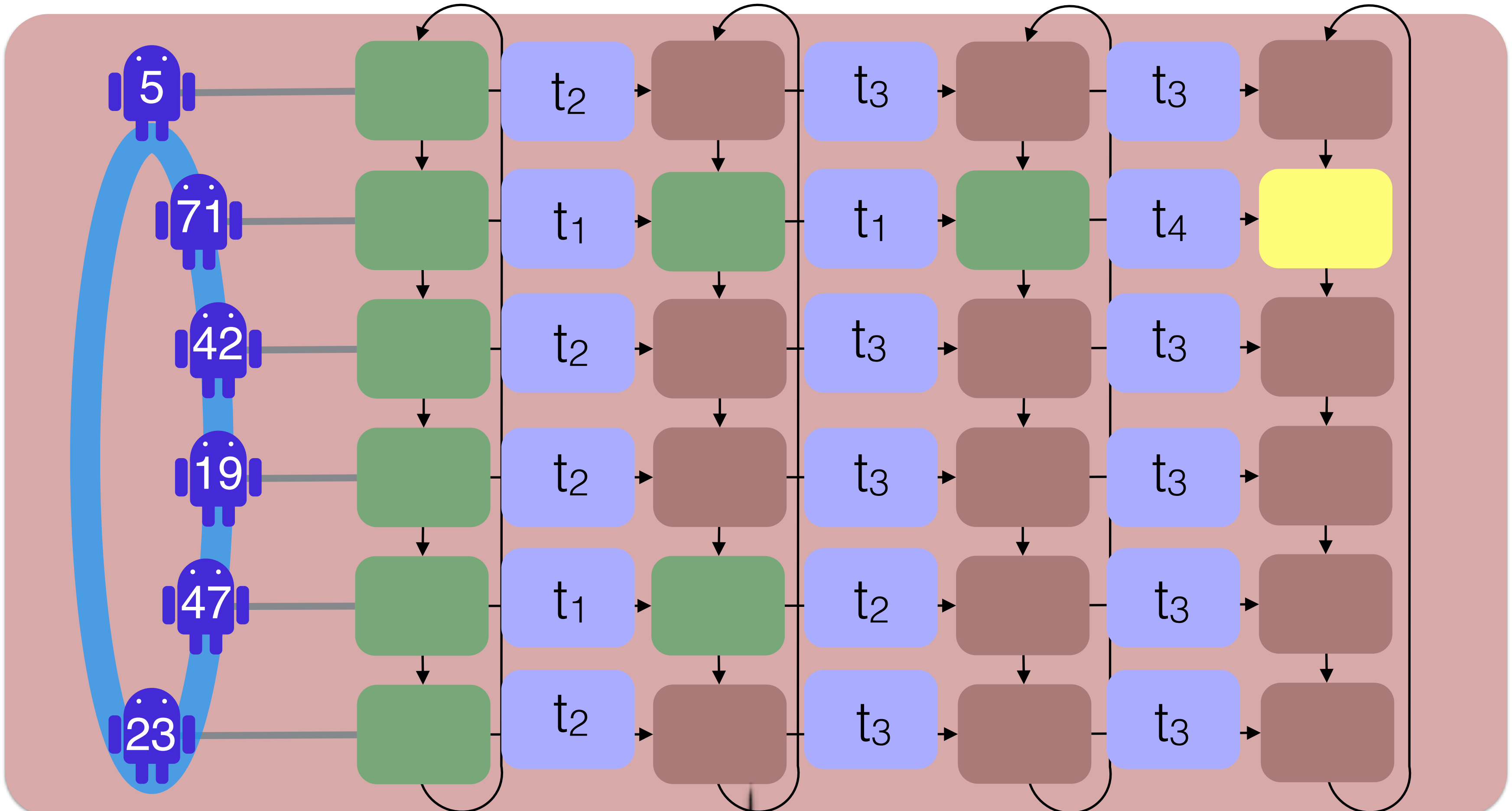
Data abstraction: symbolic runs + tracking data

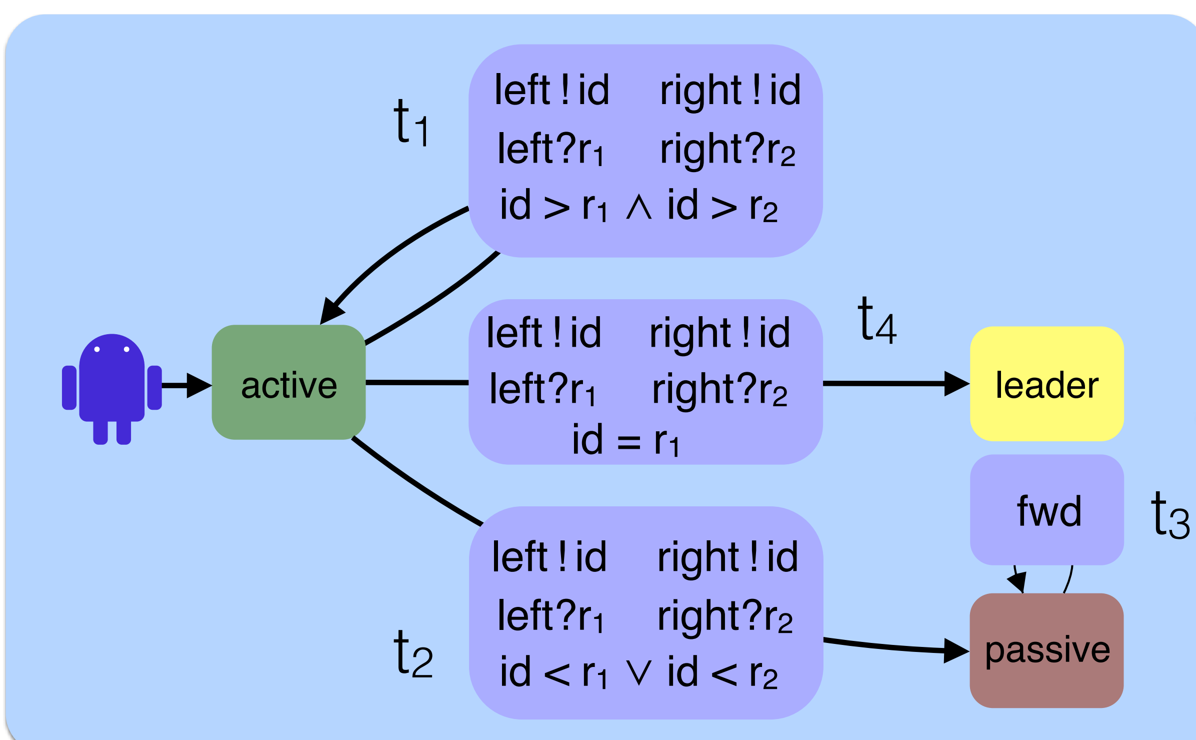




Distributed algorithm

Data abstraction: symbolic runs + tracking data



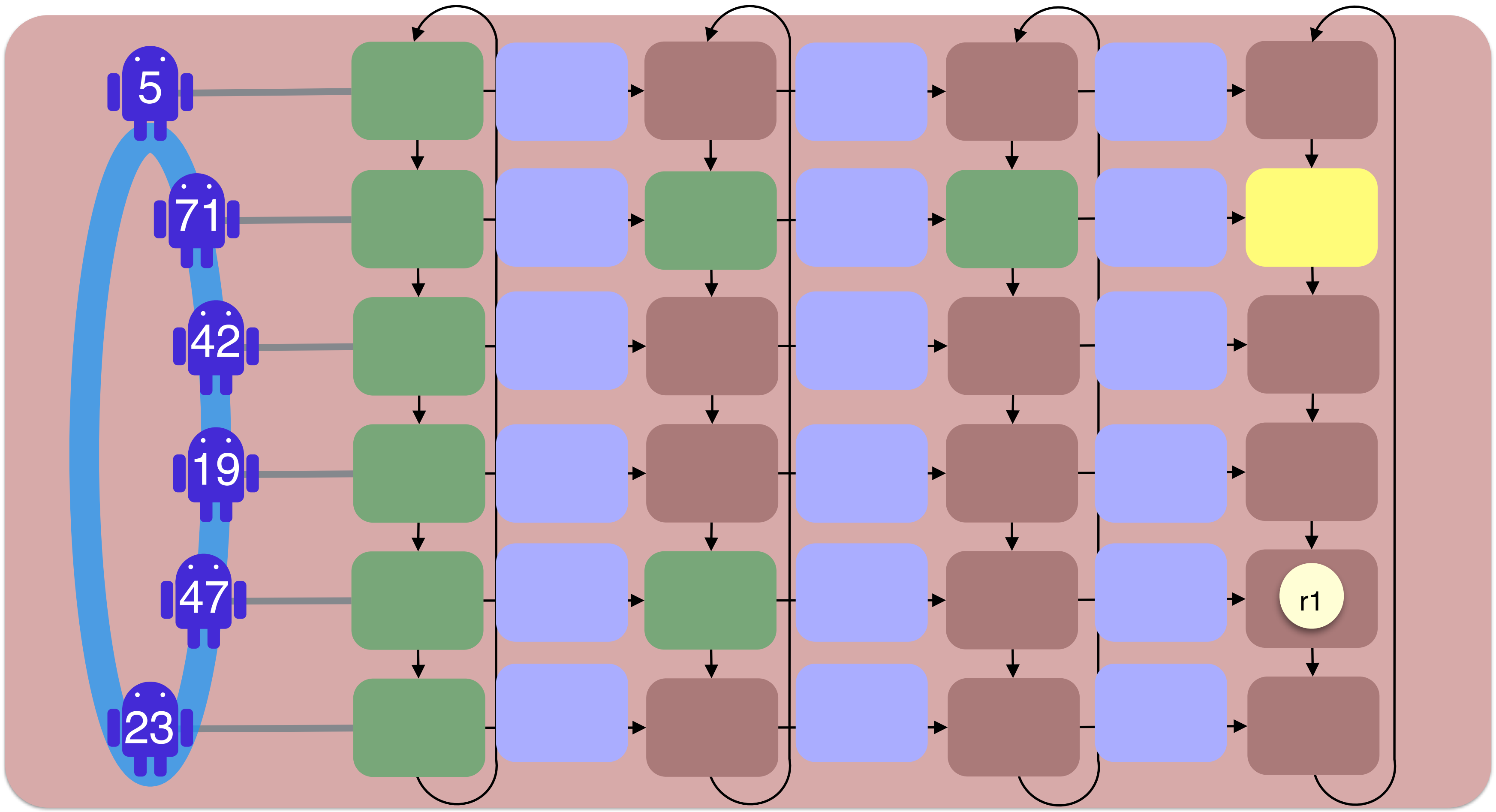


Distributed algorithm

A pid distribution realizes a symbolic run if all guards are satisfied.

Pb: Is there a pid distribution realizing a symbolic run?

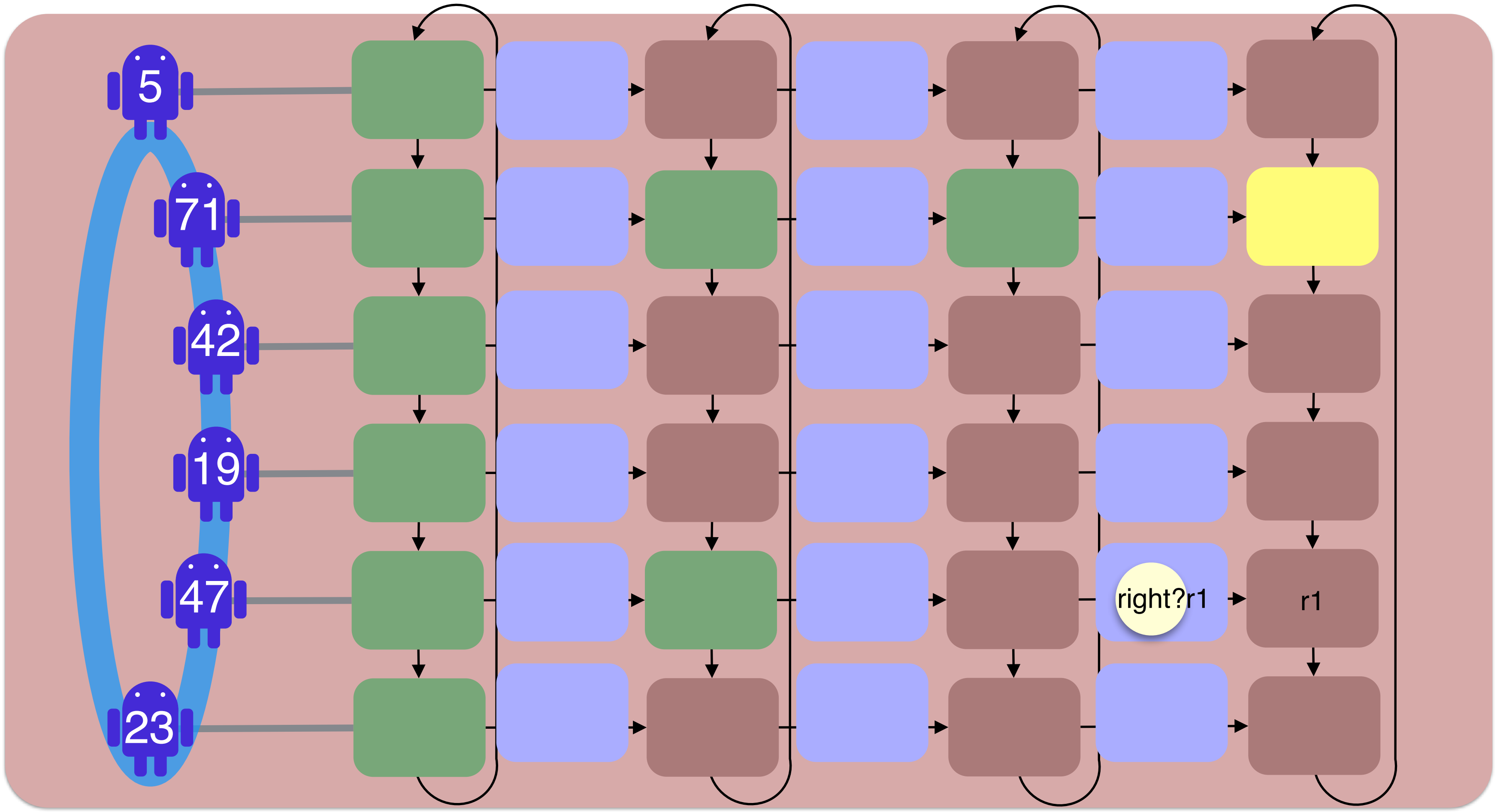
Data abstraction: symbolic runs + tracking data



- Register updates

Distributed algorithm

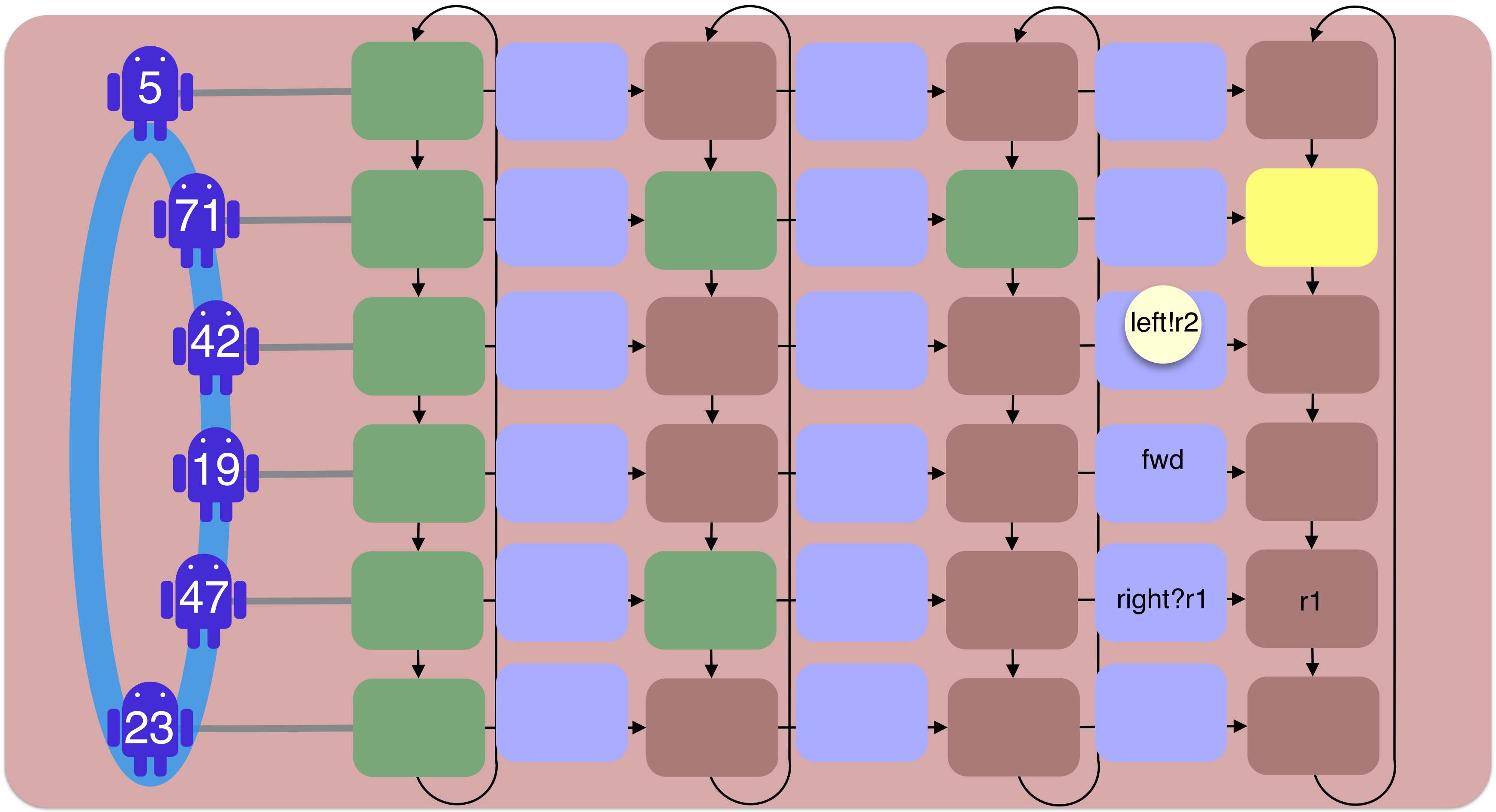
Data abstraction: symbolic runs + tracking data



- Register updates

Distributed algorithm

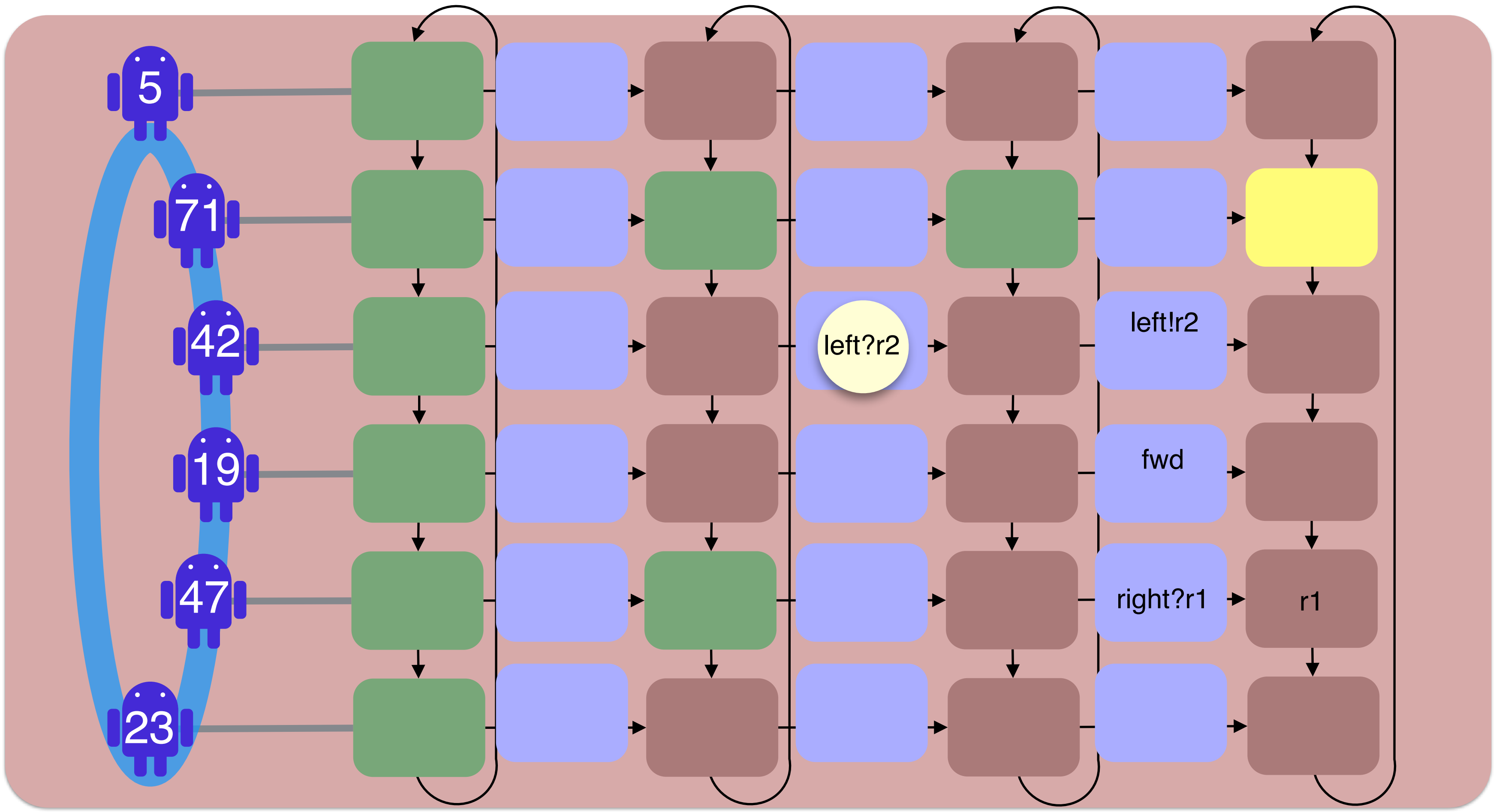
Data abstraction: symbolic runs + tracking data



- Register updates

Distributed algorithm

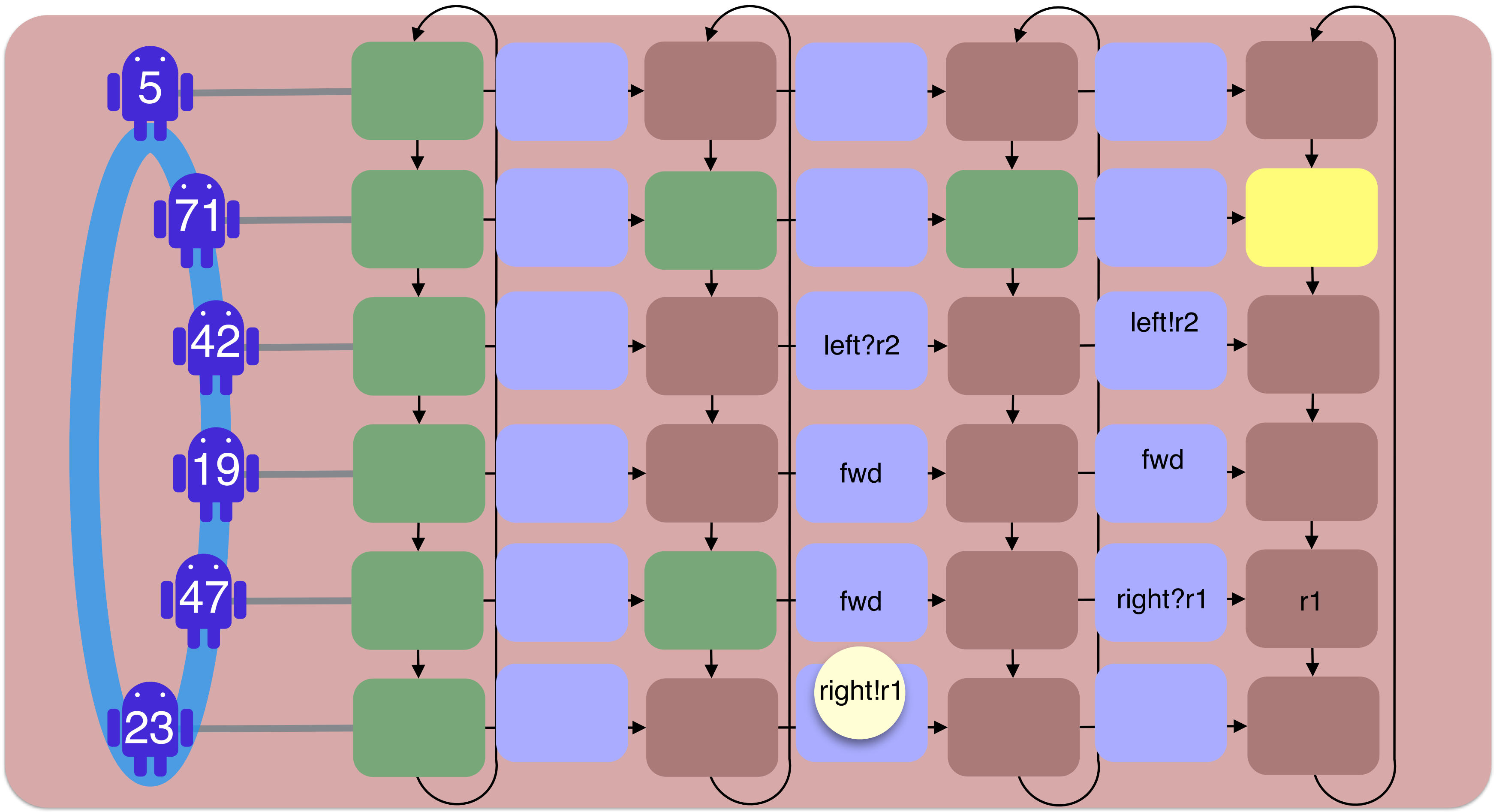
Data abstraction: symbolic runs + tracking data



- Register updates

Distributed algorithm

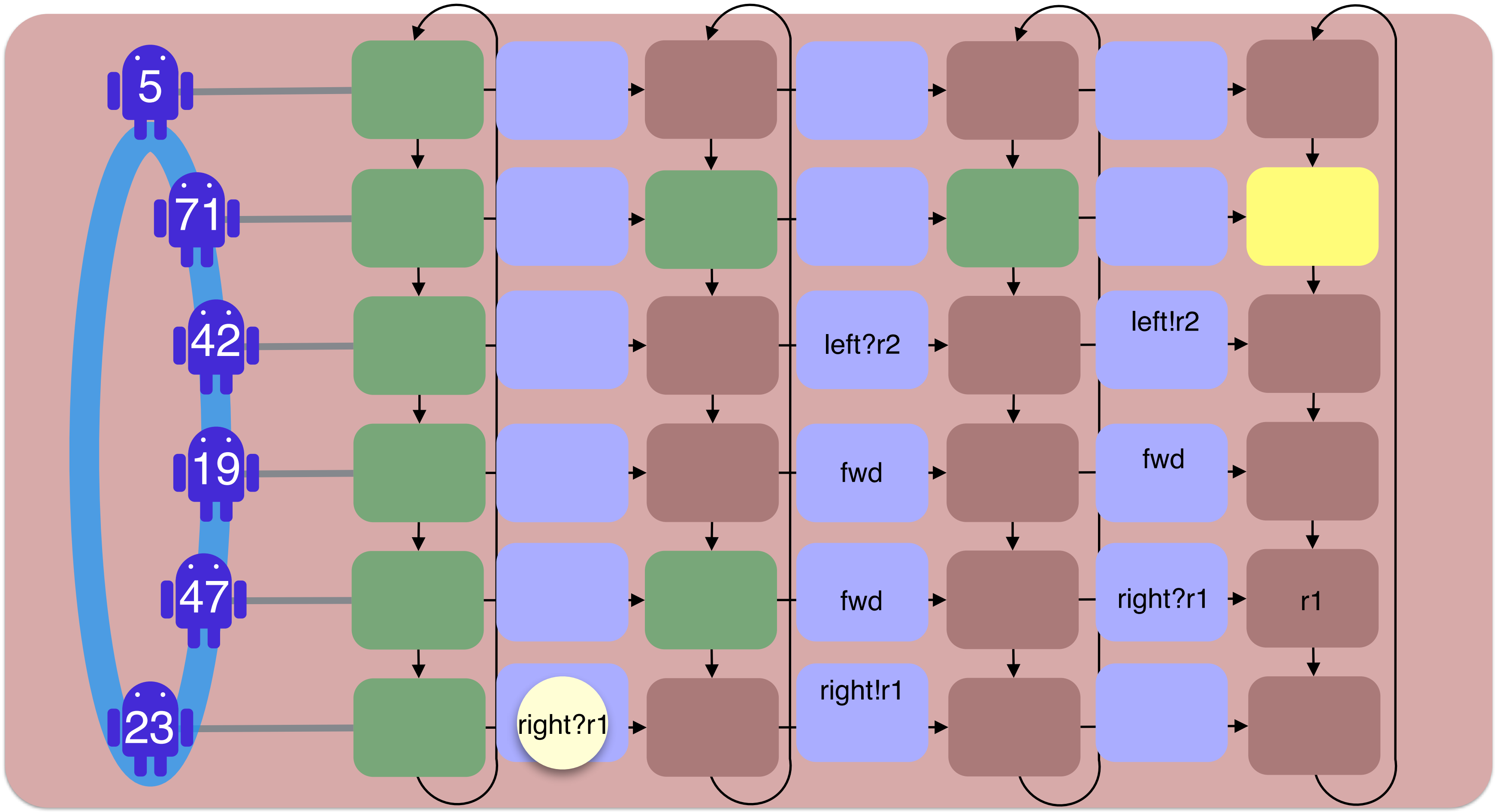
Data abstraction: symbolic runs + tracking data



- Register updates

Distributed algorithm

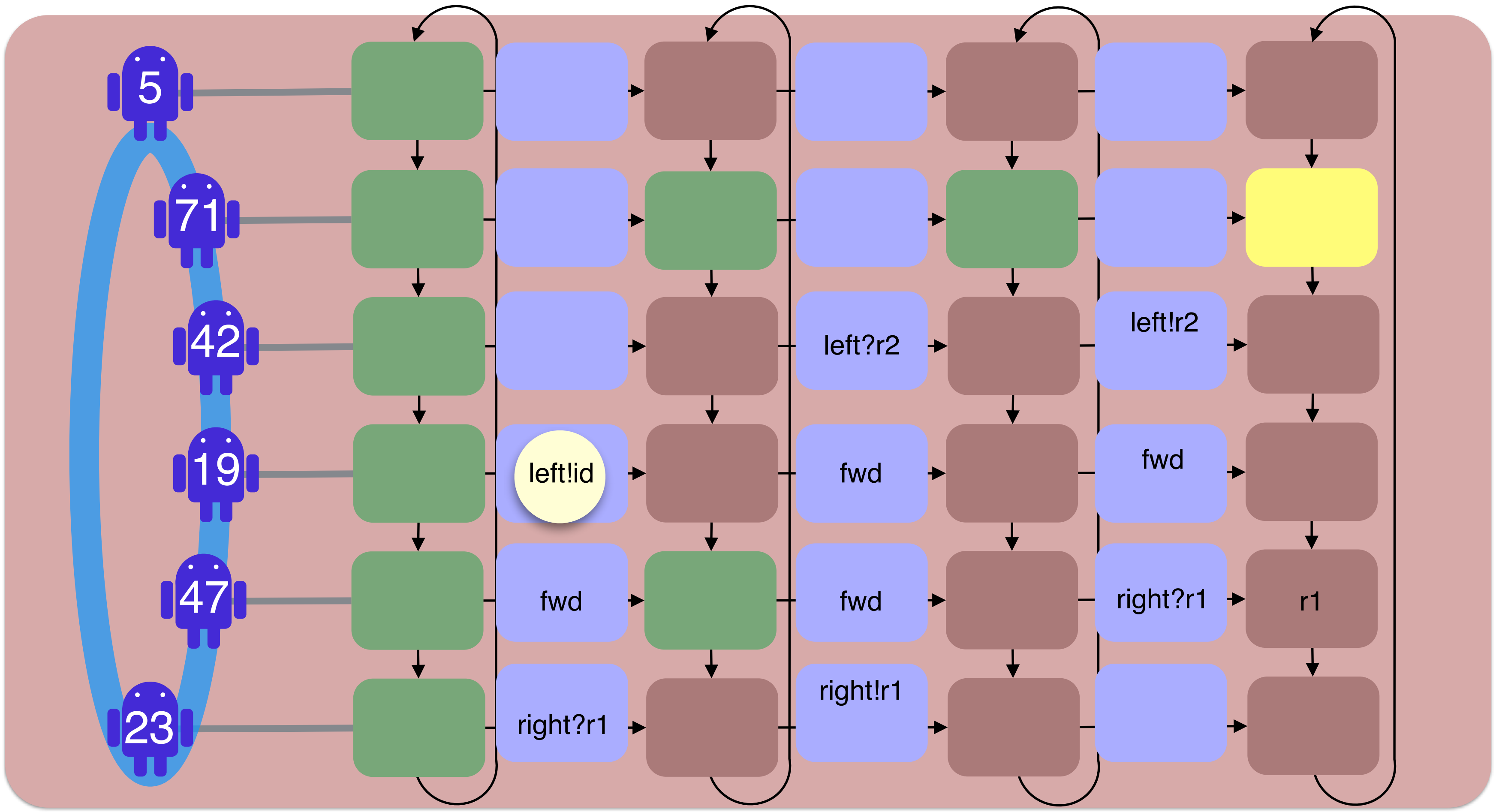
Data abstraction: symbolic runs + tracking data



- Register updates

Distributed algorithm

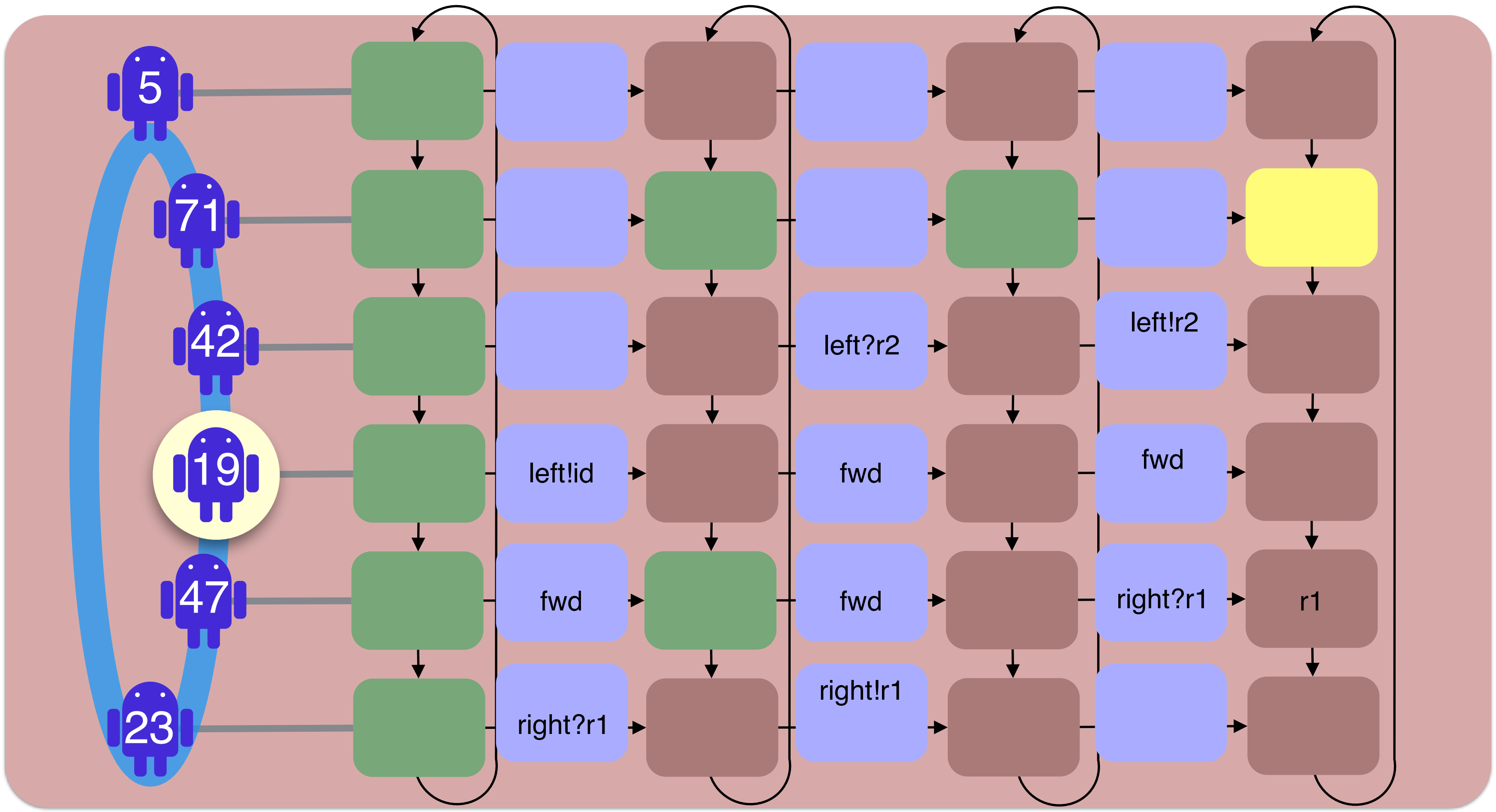
Data abstraction: symbolic runs + tracking data



- Register updates

Distributed algorithm

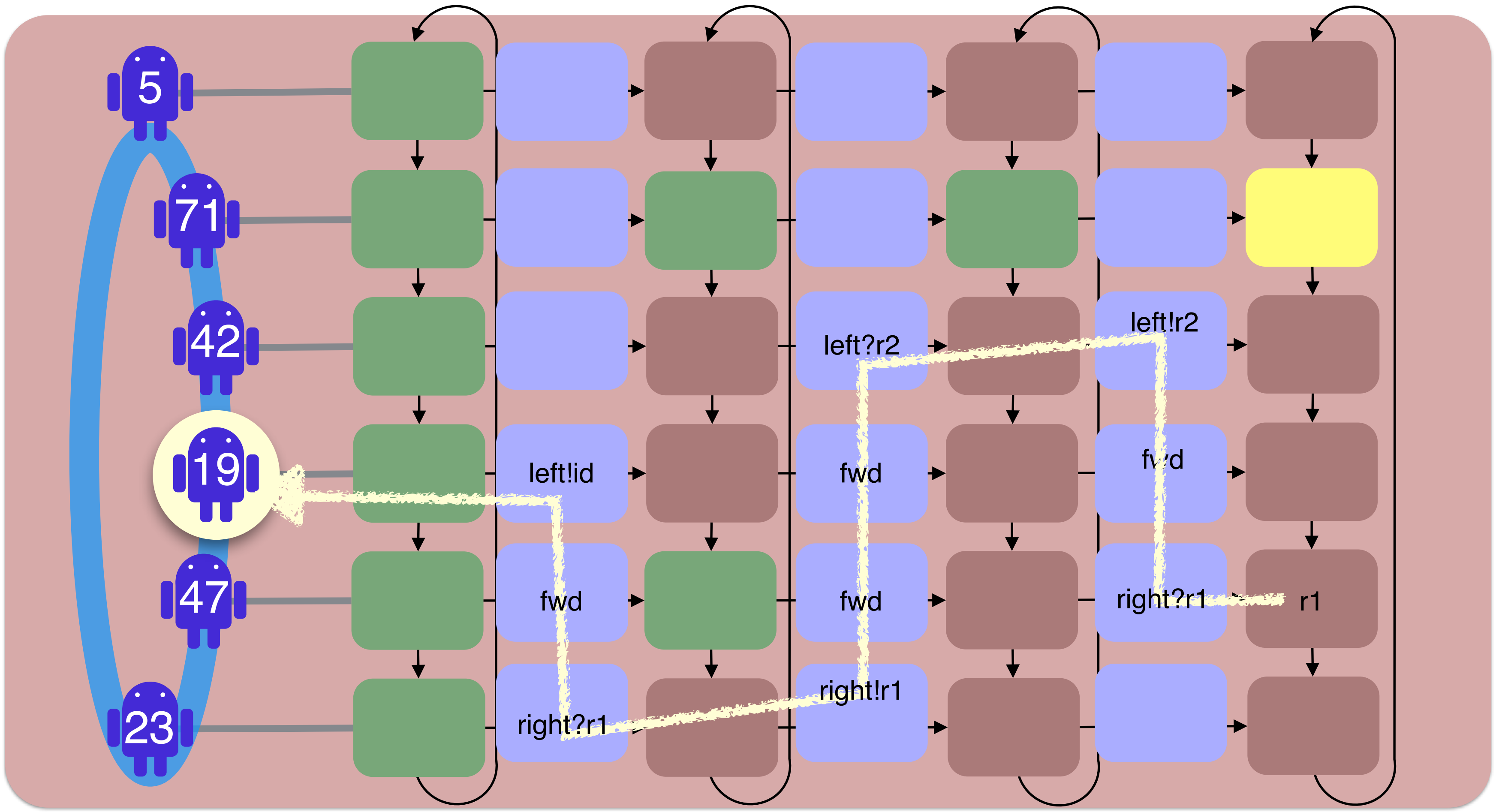
Data abstraction: symbolic runs + tracking data



- Register updates

Distributed algorithm

Data abstraction: symbolic runs + tracking data



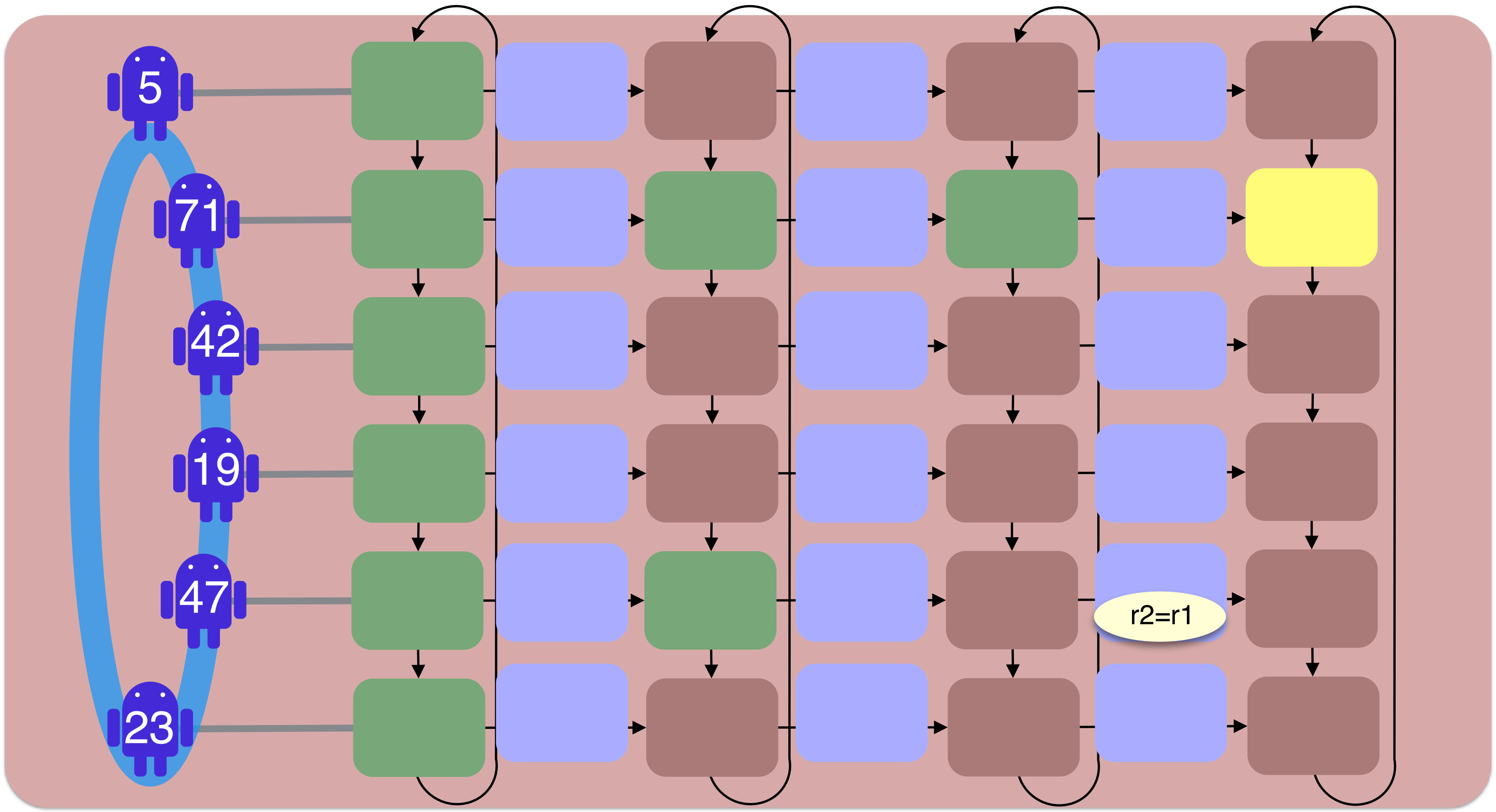
- Register updates

Distributed algorithm

(r$_1$,id)-path

can be expressed in CPDL
PDL with converse

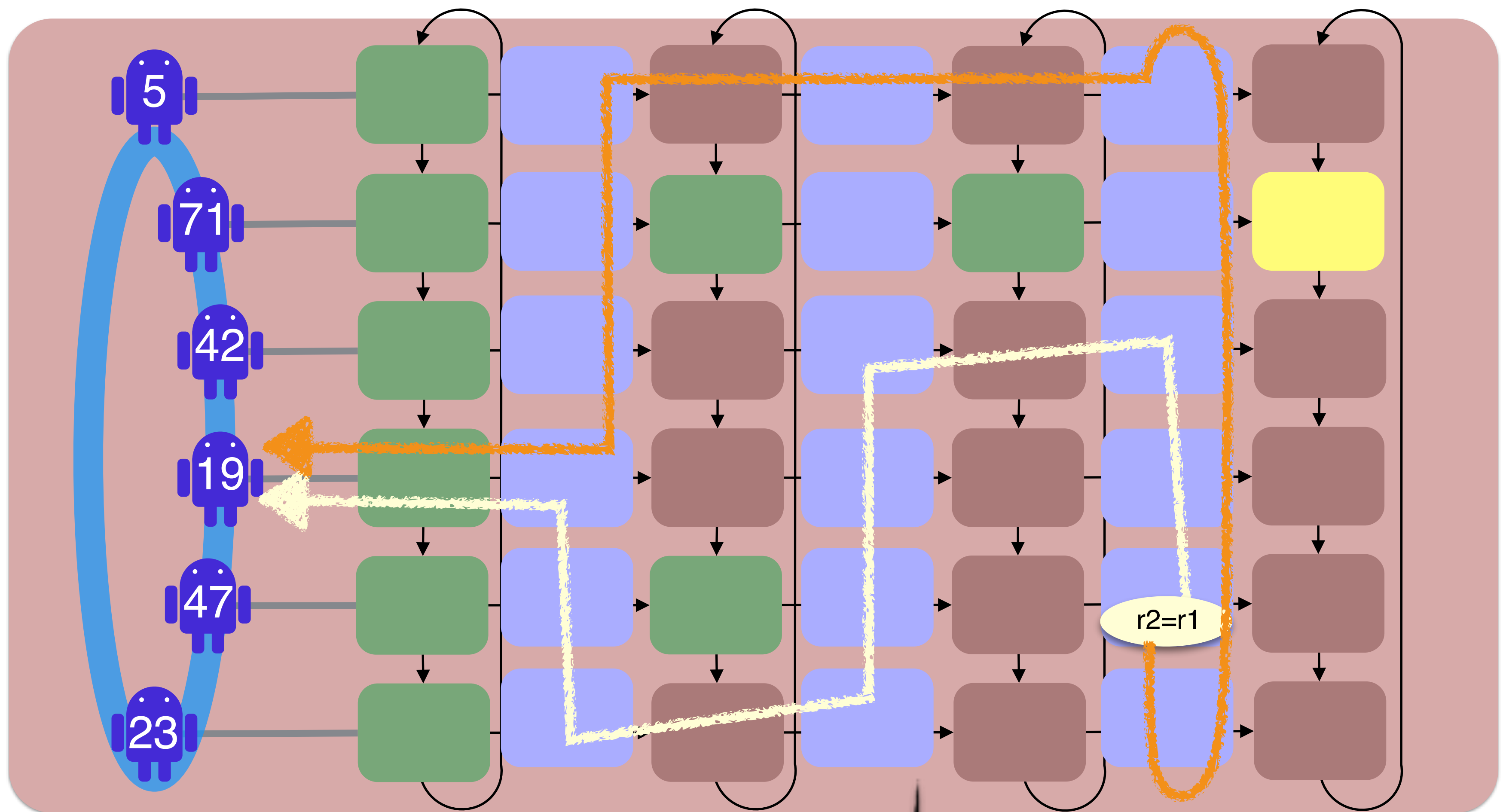
Data abstraction: symbolic runs + tracking data



- Register updates
- Register equality check

Distributed algorithm

Data abstraction: symbolic runs + tracking data

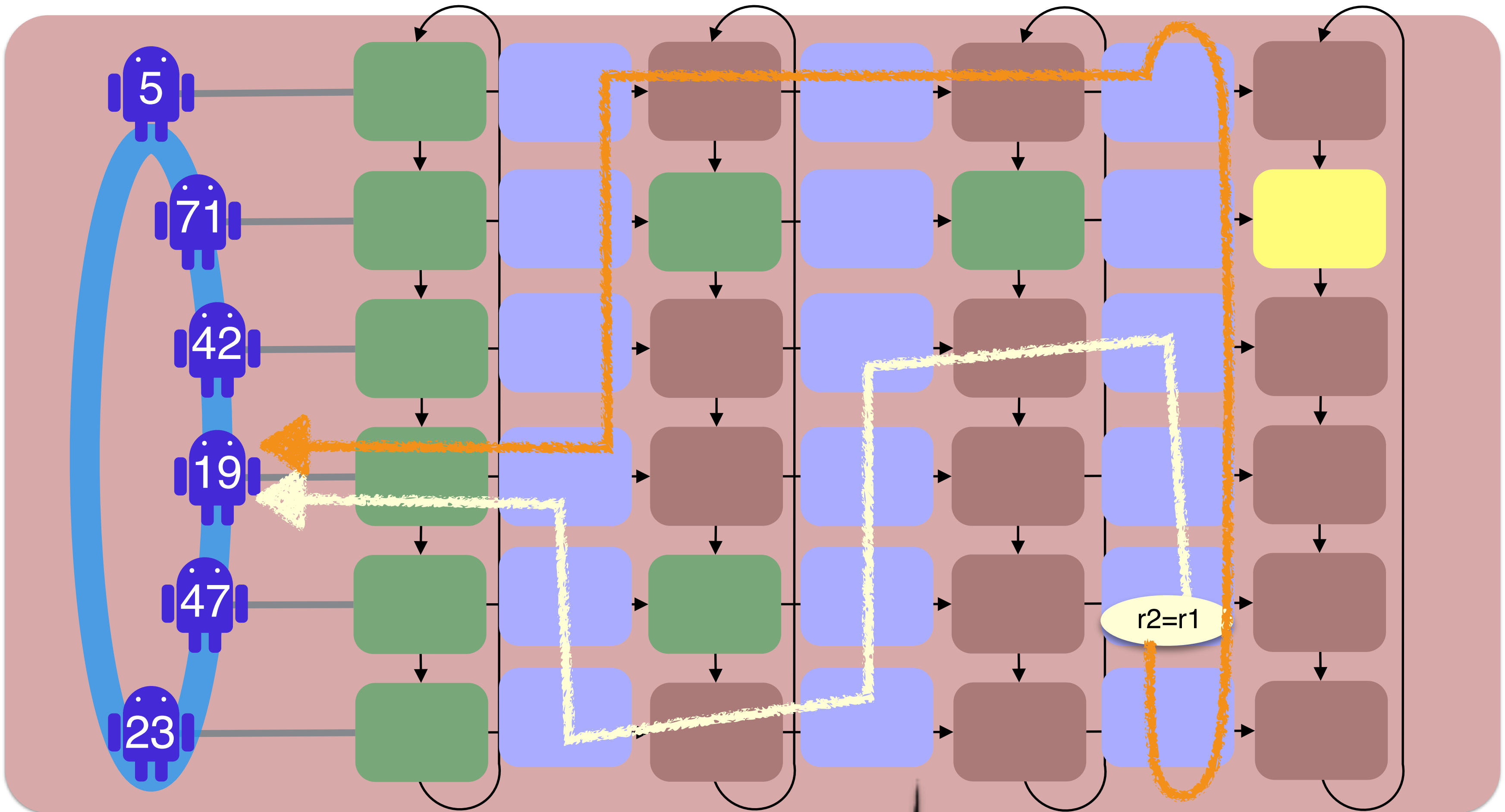


π_1:(r_1,id)-path
π_2:(r_2,id)-path

- Register updates
- Register equality check

Distributed algorithm

Data abstraction: symbolic runs + tracking data



- Register updates
- Register equality check

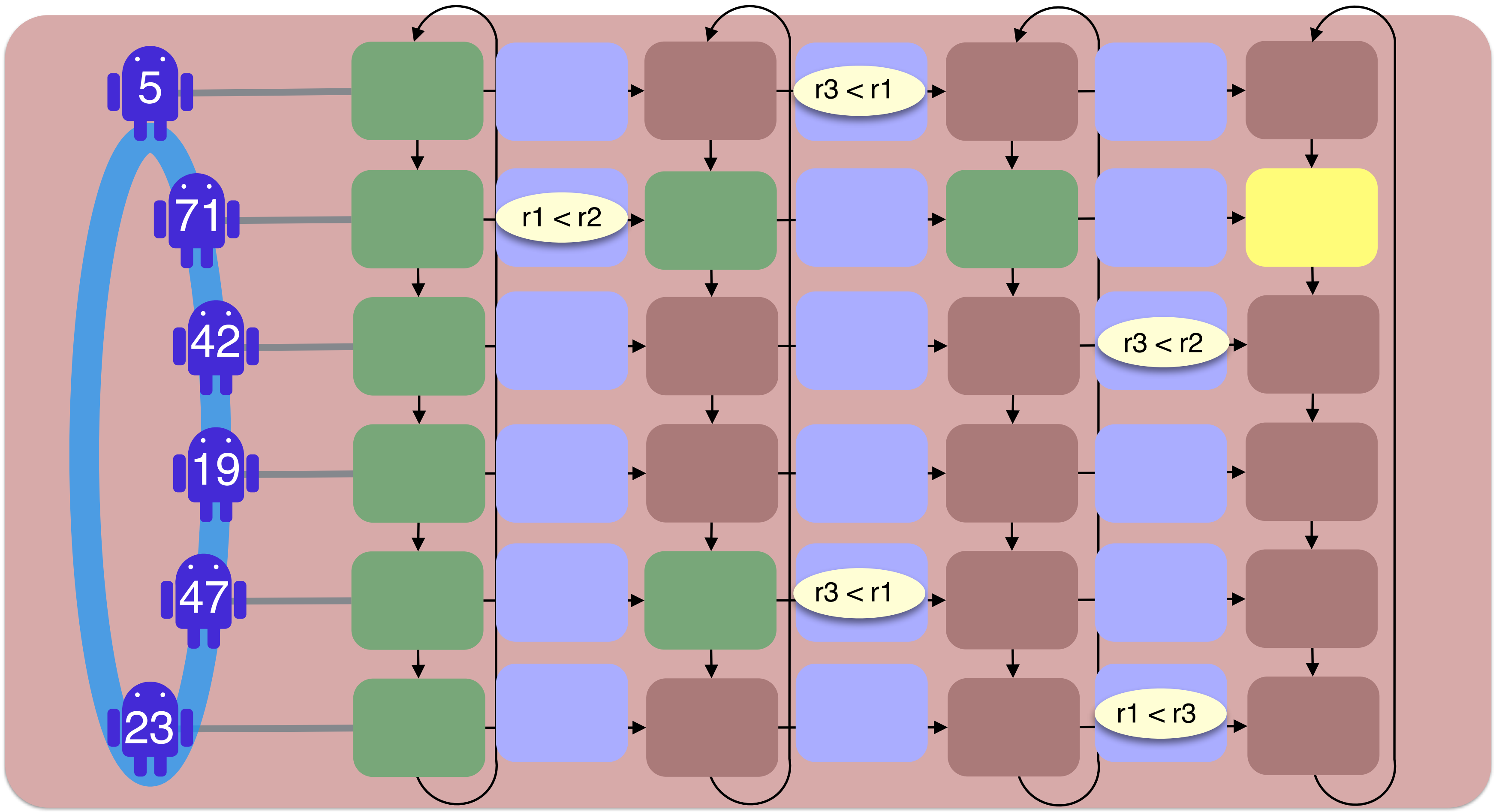
Distributed algorithm

π_1:(r_1,id)-path
π_2:(r_2,id)-path

$r_2 = r_1$ iff loop(π_1 ; π_2^{-1})

can be expressed in LCPDL
CPDL with loop

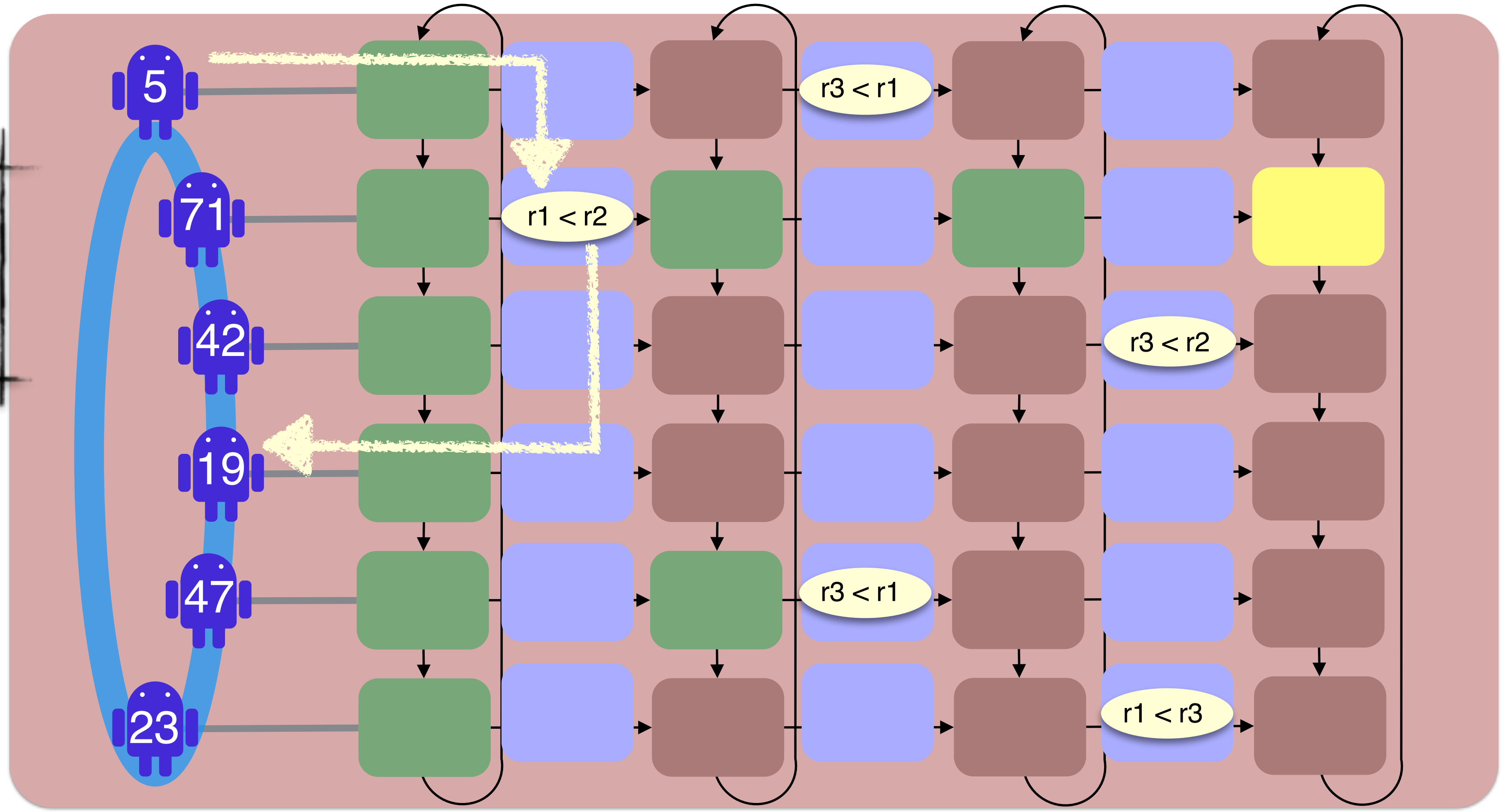
Data abstraction: symbolic runs + tracking data



- Register updates
- Register equality check
- Register comparison

Distributed algorithm

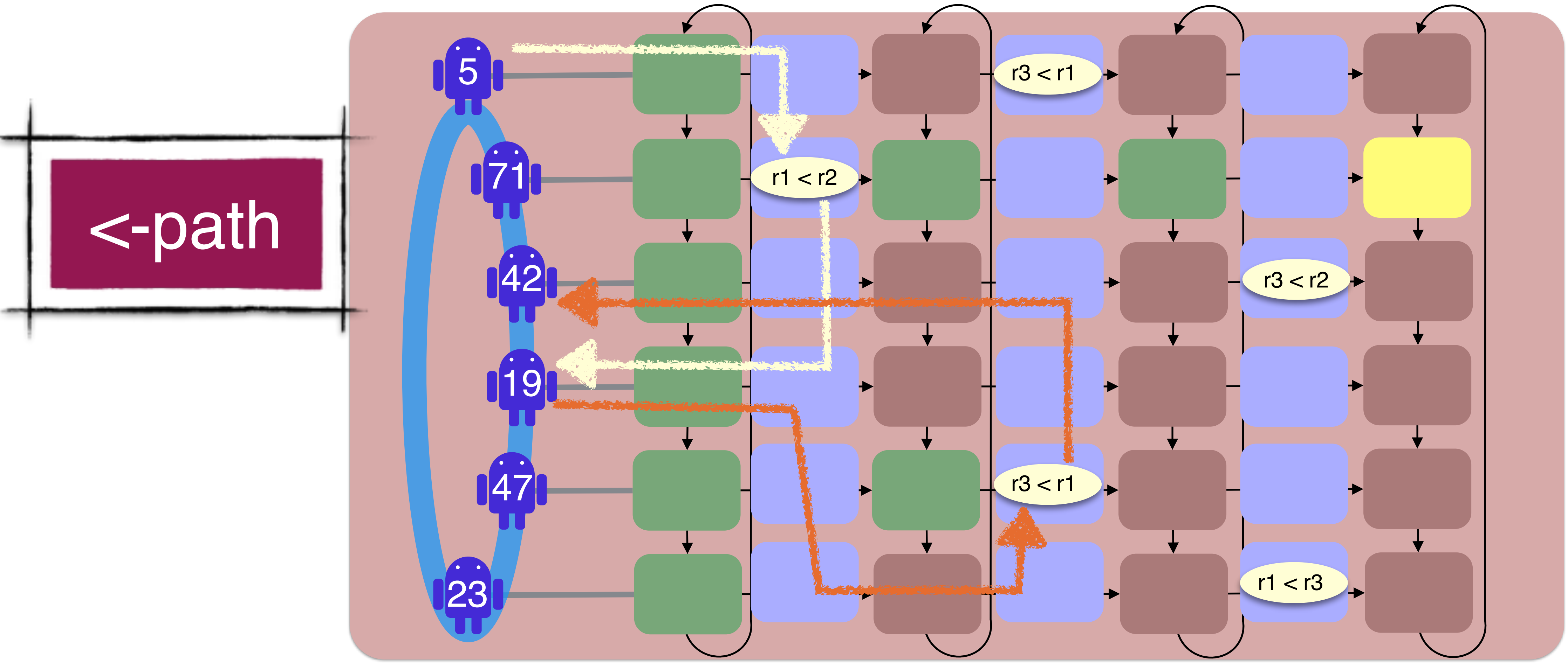
Data abstraction: symbolic runs + tracking data



- Register updates
- Register equality check
- Register comparison

Distributed algorithm

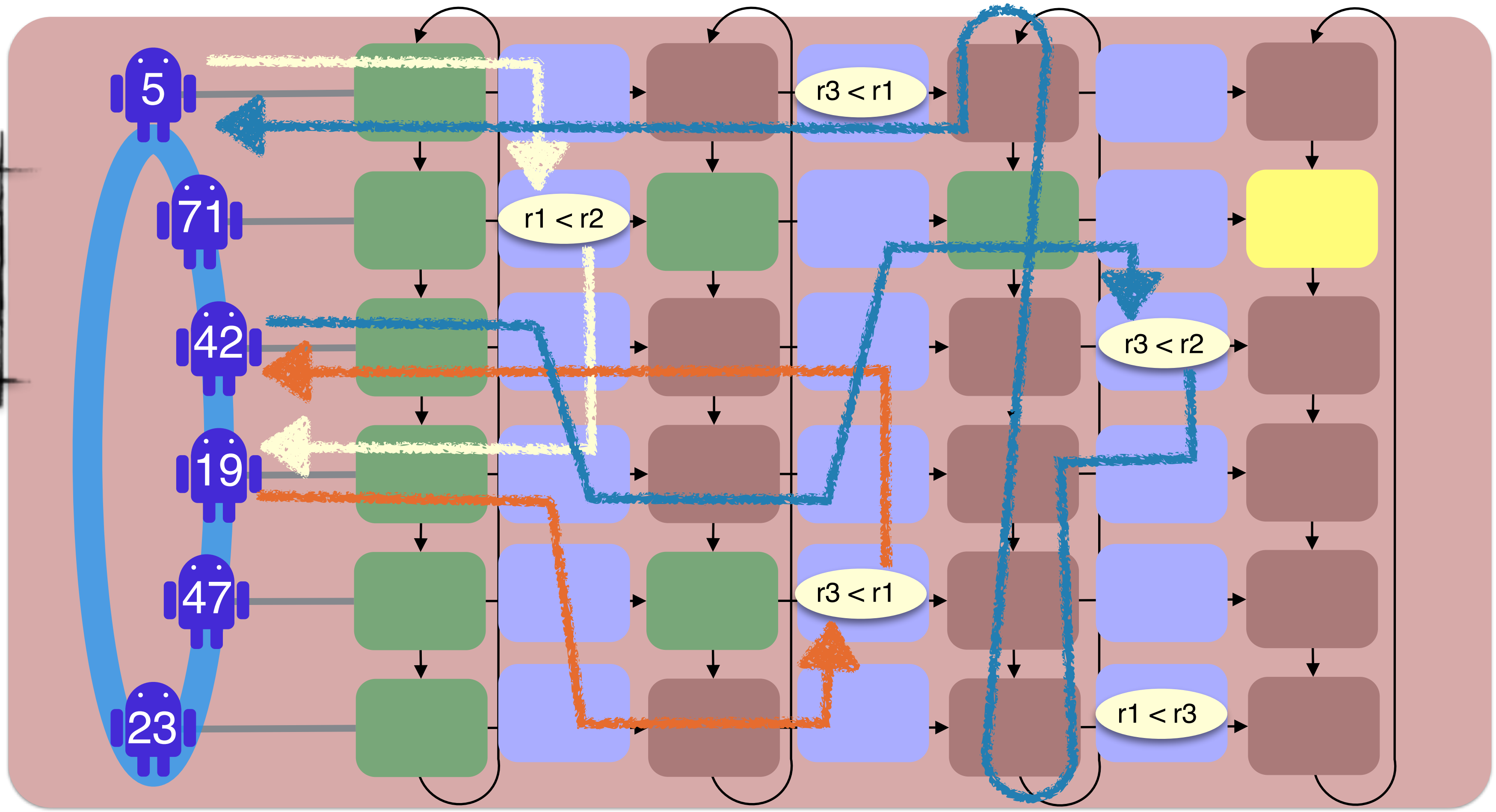
Data abstraction: symbolic runs + tracking data



- Register updates
- Register equality check
- Register comparison

Distributed algorithm

Data abstraction: symbolic runs + tracking data

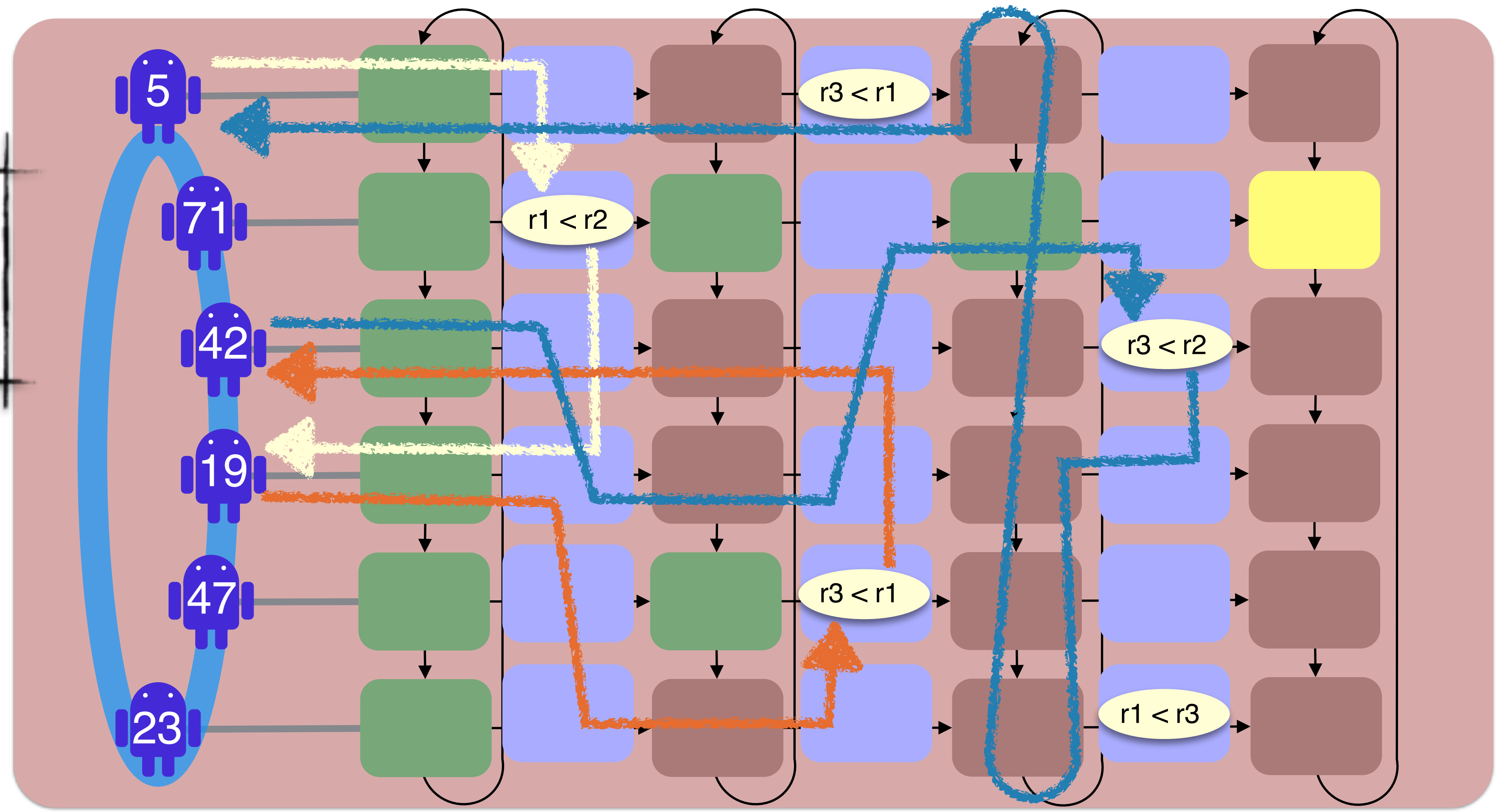


- Register updates
- Register equality check
- Register comparison

Distributed algorithm

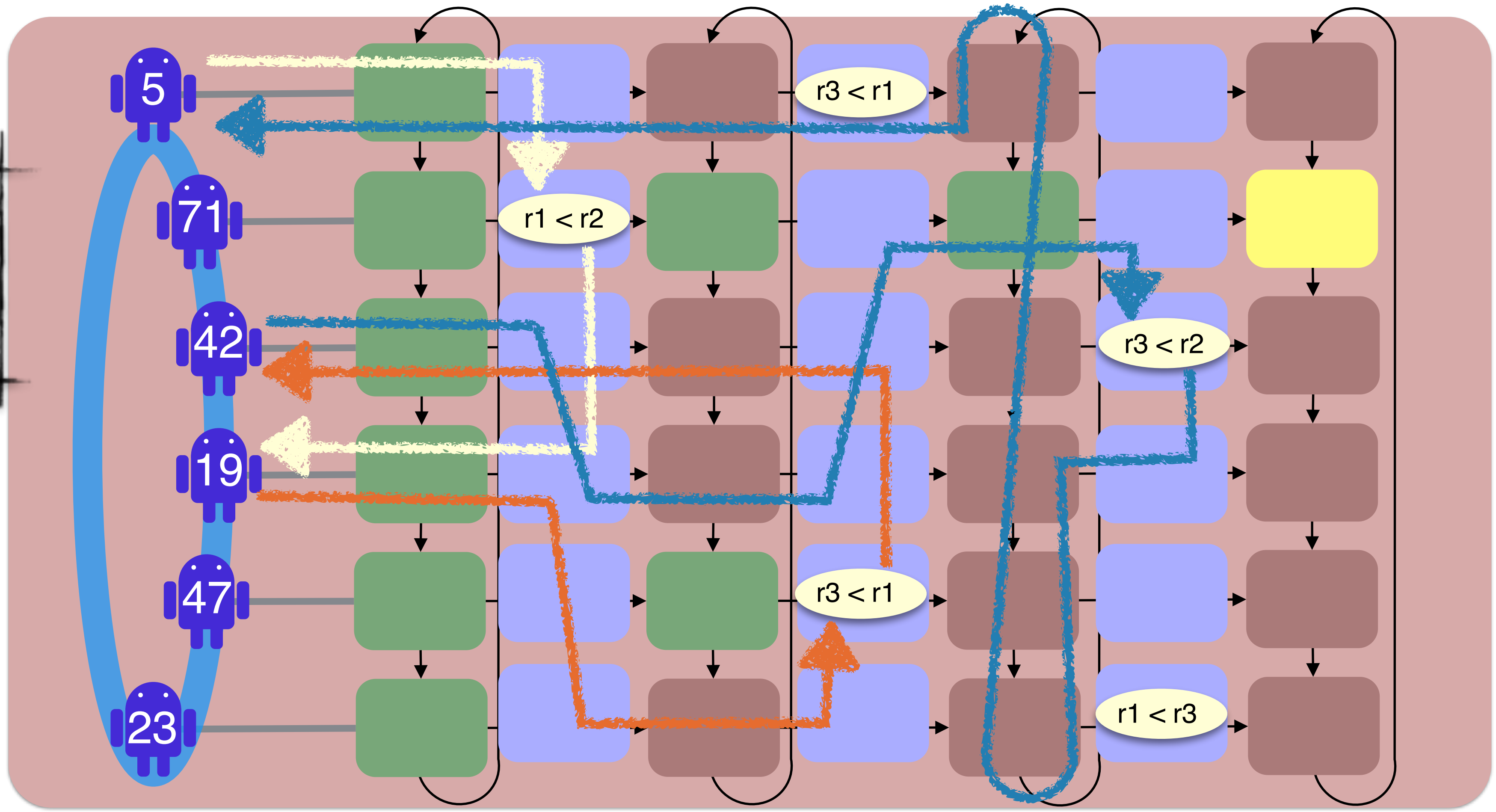
Data abstraction: symbolic runs + tracking data

<-path



- If there is a <-loop, no pid assignments can turn the symbolic cylinder into a valid run.
- If no such loops, then there are pids that allow a valid realization of the symbolic cylinder

Data abstraction: symbolic runs + tracking data



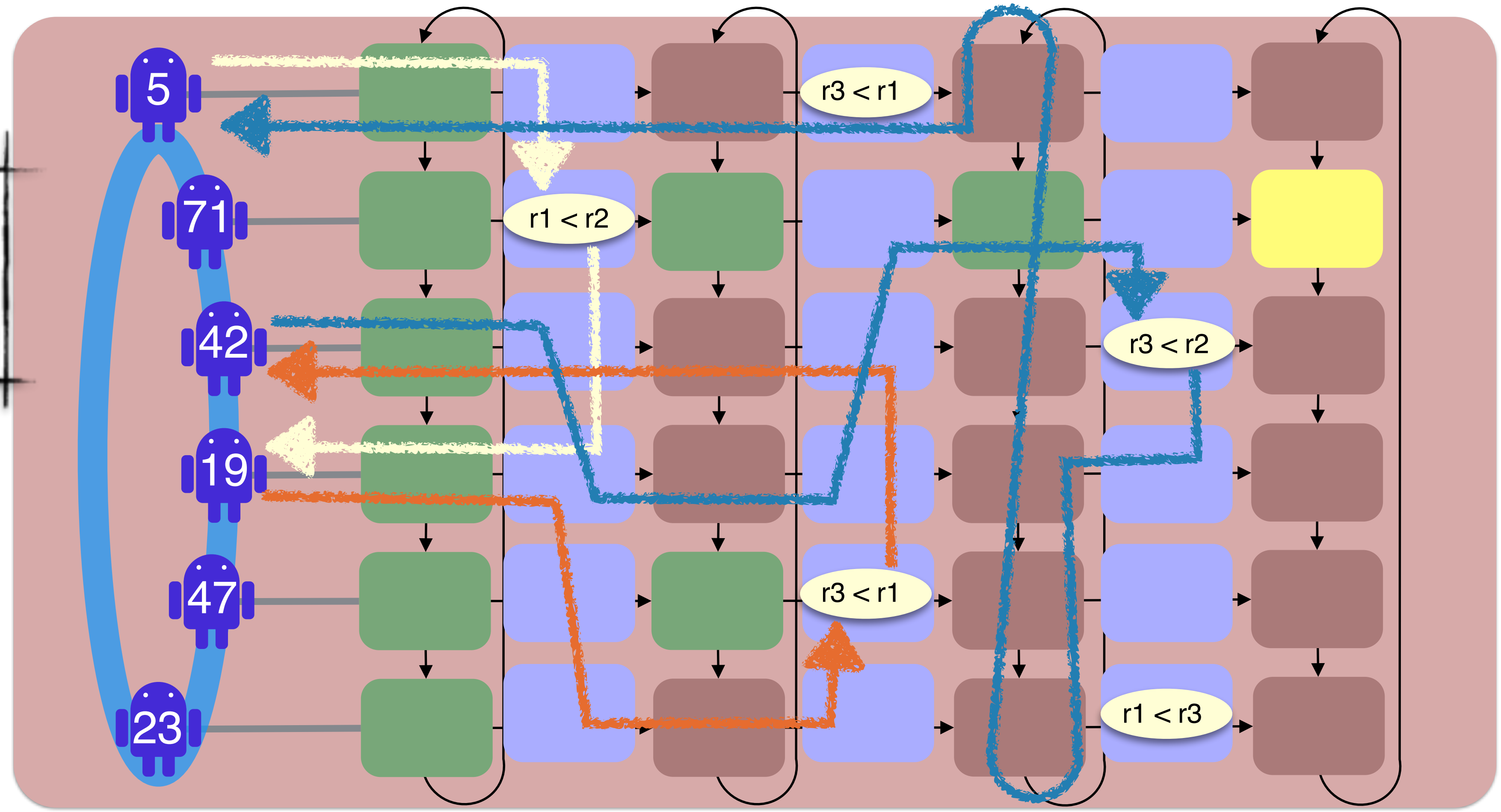
<-path

No loop of the form

$(\Sigma_{i,j}\ (r_i,id)\text{-path}^{-1};\ r_i<r_j;\ (r_j,id)\text{-path})^{+}$

- If there is a <-loop, no pid assignments can turn the symbolic cylinder into a valid run.
- If no such loops, then there are pids that allow a valid realization of the symbolic cylinder

Data abstraction: symbolic runs + tracking data



<-path

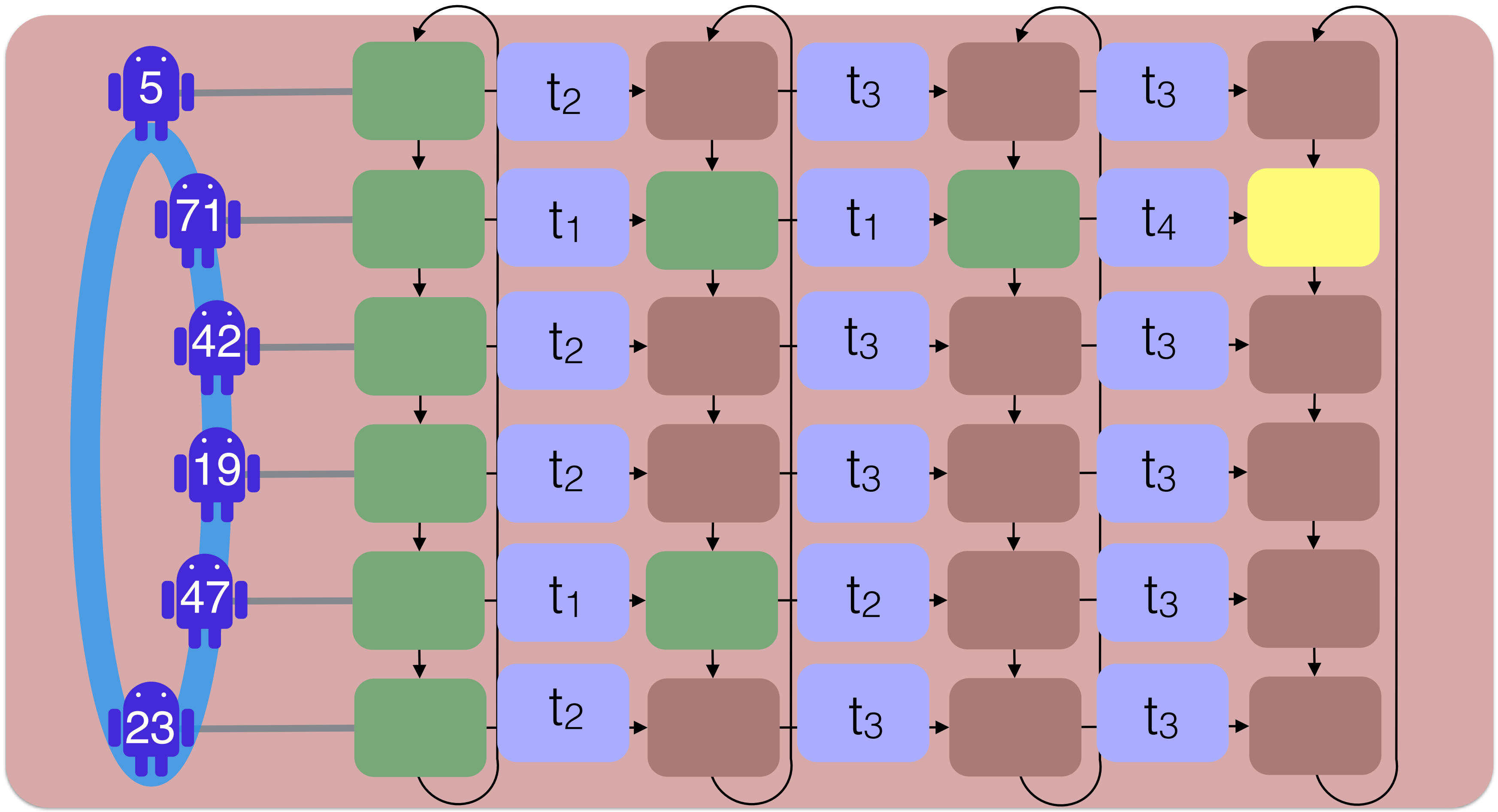
No loop of the form

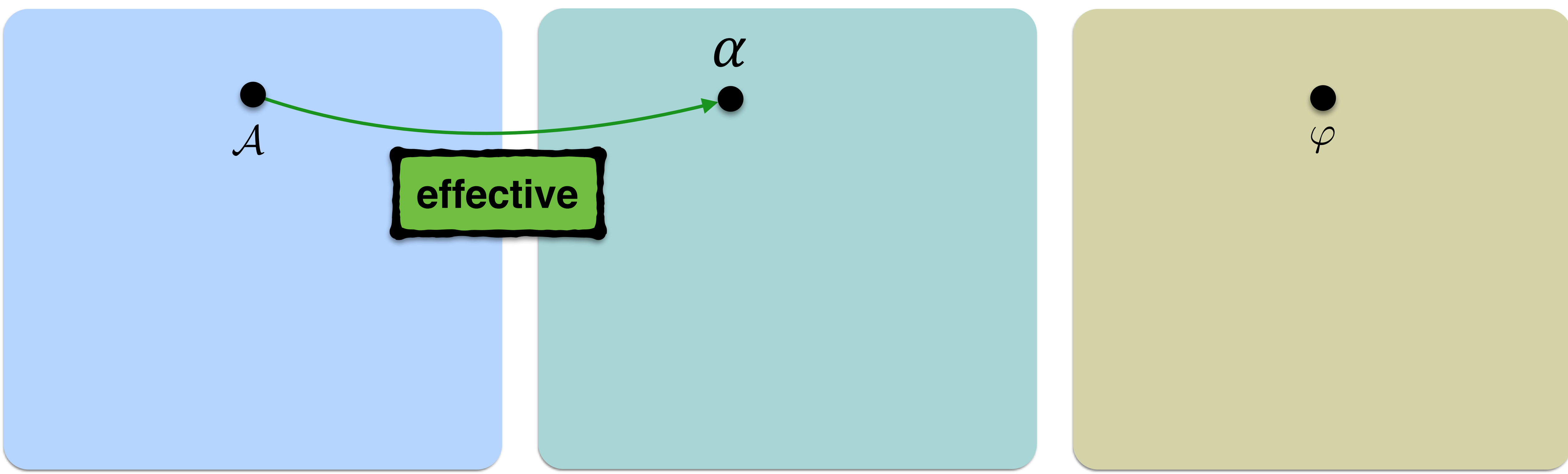
$(\Sigma_{i,j}\ (r_i,id)\text{-path}^{-1};\ r_i<r_j;\ (r_j,id)\text{-path})^+$

- If there is a <-loop, no pic symbolic cylinder into a v
- If no such loops, then the valid realization of the symbolic cylinder

can be expressed in LCPDL CPDL with loop

Data abstraction: symbolic runs + tracking data



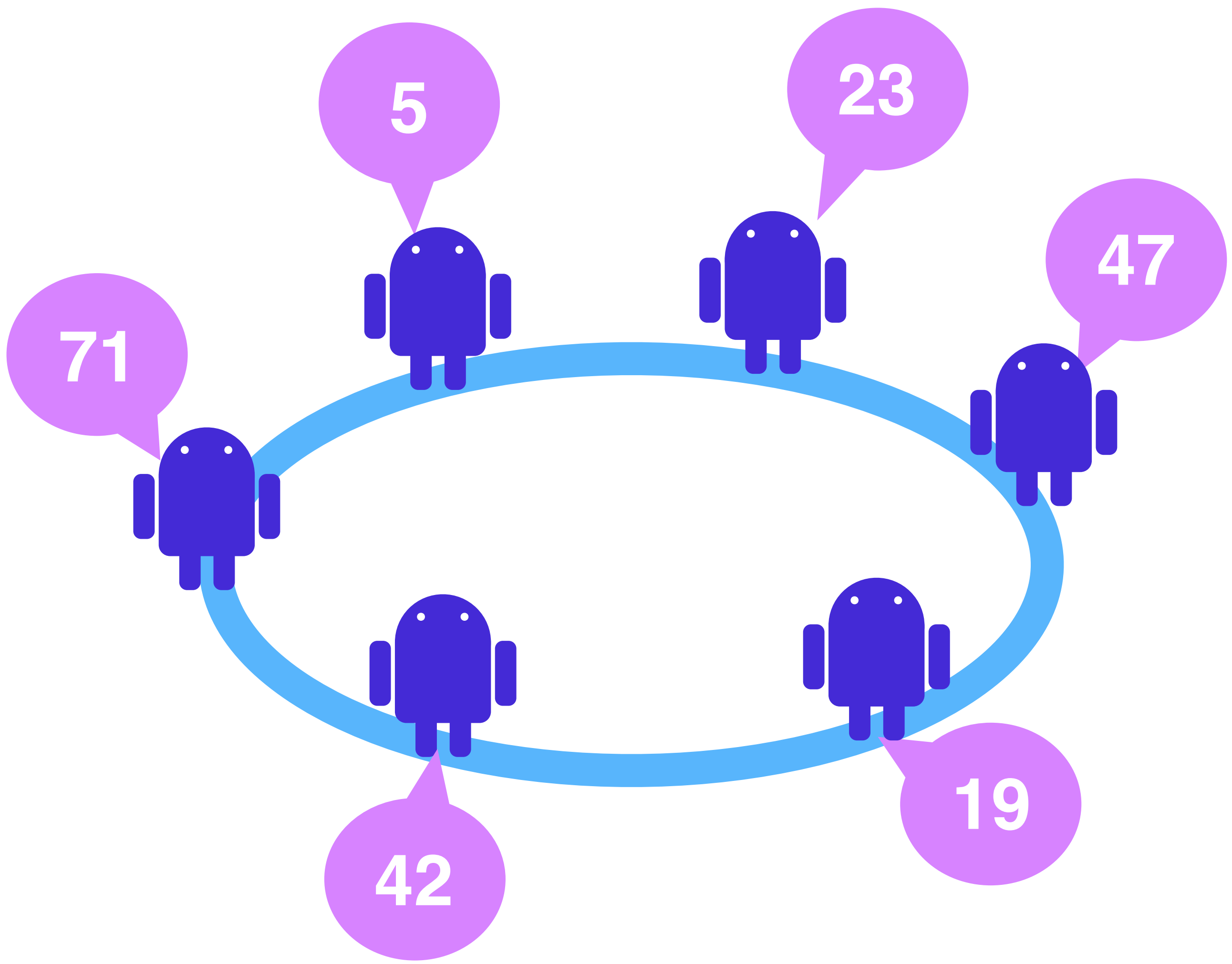


Distributed algorithm

PDL with loop (over finite alphabet)

Data PDL

Specification language



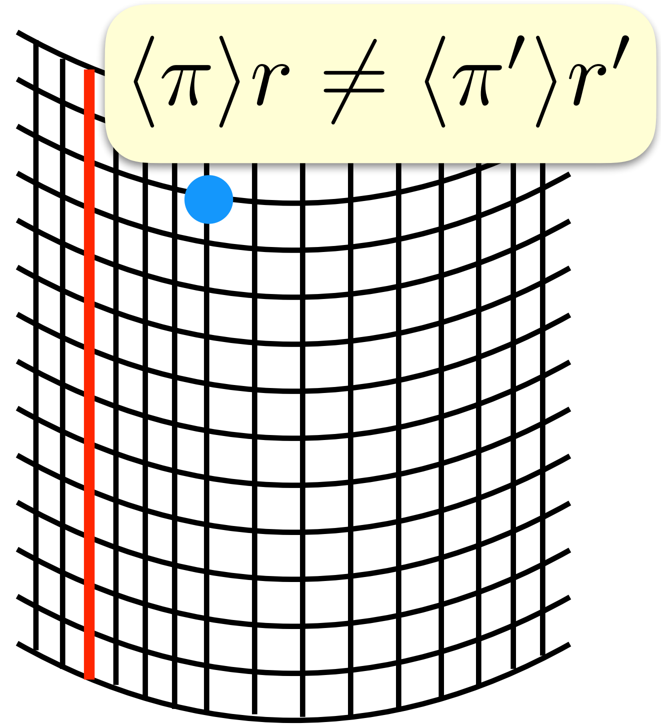
Distributed algorithms: typical properties

- Leader election:
  - At the end there is a unique leader
  - All other processes are passive
  - The leader has the maximal pid
- Distributed sorting algorithm
  - The output values form a permutation of the input values
  - If q is on the right of p, and q ≠ leader then p.v < q.v

Moves inside the behavior

compare values at different nodes

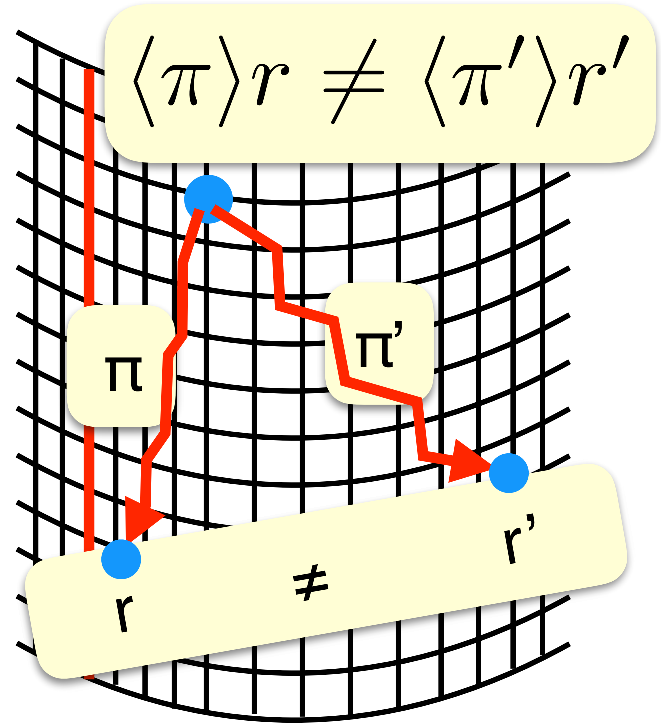
Specifications
Data PDL



$$
\begin{aligned}
\Phi, \Phi' &::= \mathsf{A}\,\phi \mid \Phi \wedge \Phi' \\
\phi, \phi' &::= \varphi \mid \phi \wedge \phi' \mid \varphi \vee \phi \mid [\pi]\phi \mid \langle \eta \rangle r < \langle \eta' \rangle r' \mid \langle \eta \rangle r \leq \langle \eta' \rangle r' \\
\varphi, \varphi' &::= \ddagger \mid p \mid \neg\varphi \mid \varphi \wedge \varphi' \mid \langle \pi \rangle \varphi \mid \langle \pi \rangle r = \langle \pi' \rangle r' \mid \langle \pi \rangle r \neq \langle \pi' \rangle r' \\
\pi, \pi' &::= \{\varphi\}? \mid \rightarrow \mid \downarrow \mid \pi^{-1} \mid \pi + \pi' \mid \pi \cdot \pi' \mid \pi^* \\
\eta, \eta' &::= \{\varphi\}? \mid \leftarrow \mid \rightarrow \mid \downarrow \mid \uparrow \mid \eta \cdot \eta' \mid \mathsf{F}^{\eta}_{\varphi}
\end{aligned}
$$

Inspired by [Bojanczyk et al. '09; Figueira-Segoufin '11]

Specifications
Data PDL



Moves inside the behavior

compare values at different nodes

$$\Phi, \Phi' ::= \mathsf{A}\,\phi \mid \Phi \wedge \Phi'$$

$$\phi, \phi' ::= \varphi \mid \phi \wedge \phi' \mid \varphi \vee \phi \mid [\pi]\phi \mid \langle\eta\rangle r < \langle\eta'\rangle r' \mid \langle\eta\rangle r \leq \langle\eta'\rangle r'$$

$$\varphi, \varphi' ::= \ddagger \mid p \mid \neg\varphi \mid \varphi \wedge \varphi' \mid \langle\pi\rangle\varphi \mid \langle\pi\rangle r = \langle\pi'\rangle r' \mid \langle\pi\rangle r \neq \langle\pi'\rangle r'$$

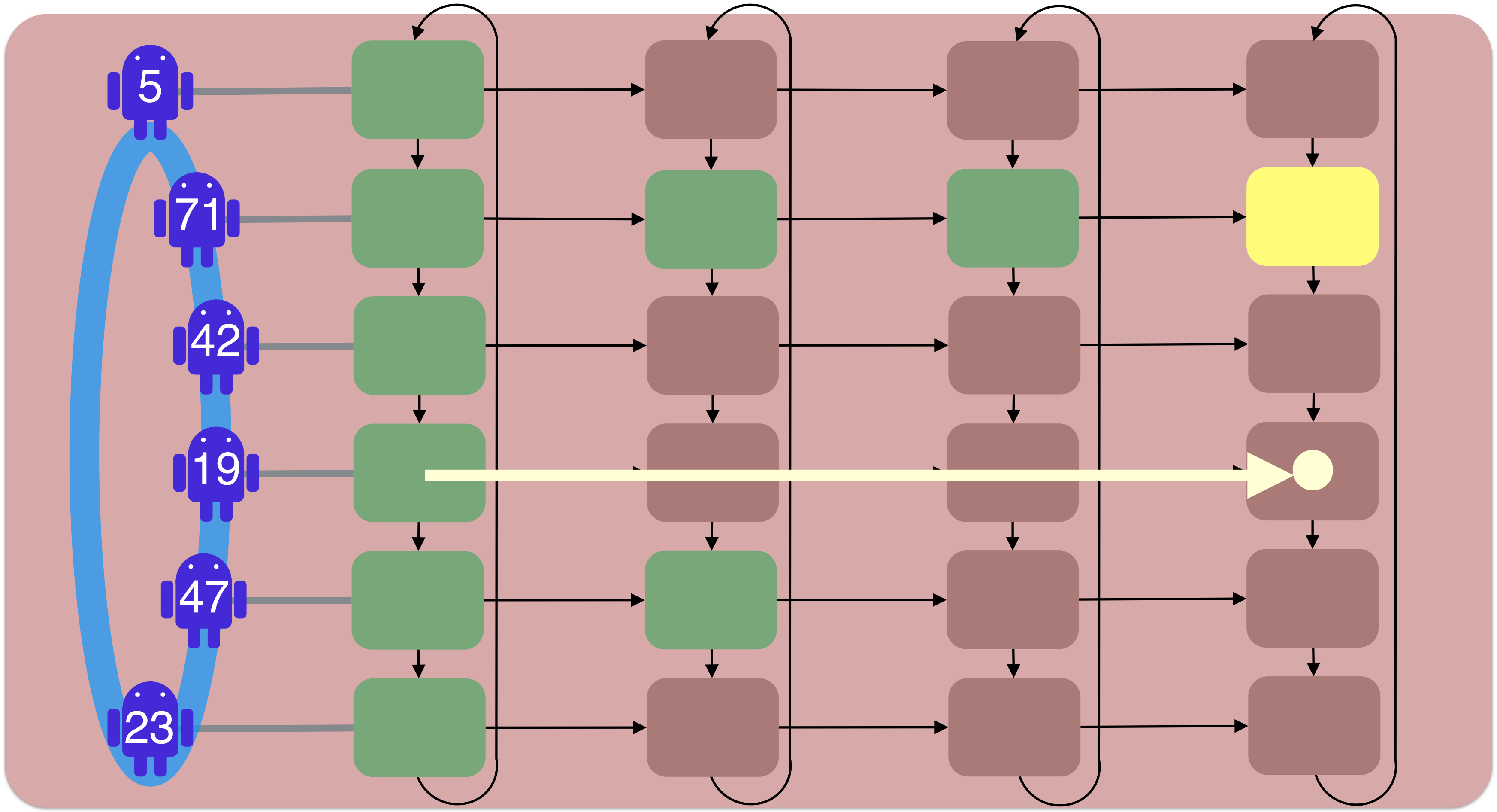
$$\pi, \pi' ::= \{\varphi\}? \mid \rightarrow \mid \downarrow \mid \pi^{-1} \mid \pi + \pi' \mid \pi \cdot \pi' \mid \pi^*$$

$$\eta, \eta' ::= \{\varphi\}? \mid \leftarrow \mid \rightarrow \mid \downarrow \mid \uparrow \mid \eta \cdot \eta' \mid \mathsf{F}^{\eta}_{\varphi}$$

Inspired by [Bojanczyk et al. ’09; Figueira-Segoufin ‘11]

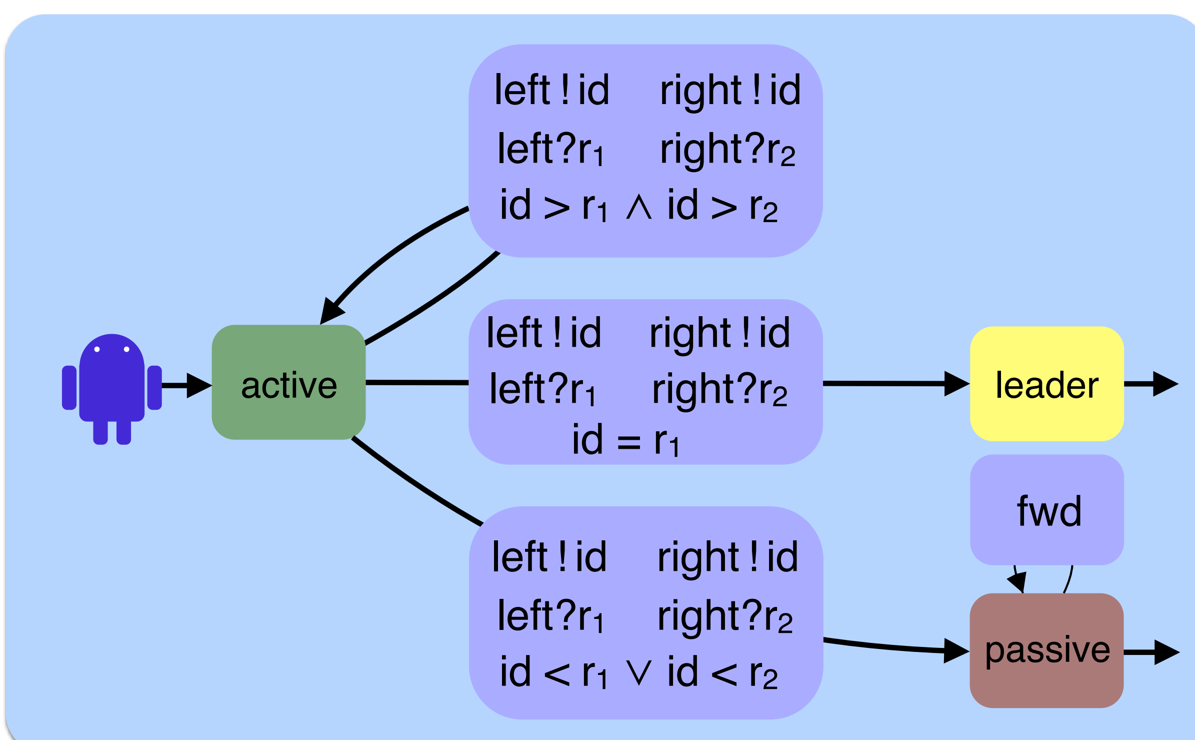
Distributed algorithms

Leader election [Franklin '82]



move in the cylinder

Behavior

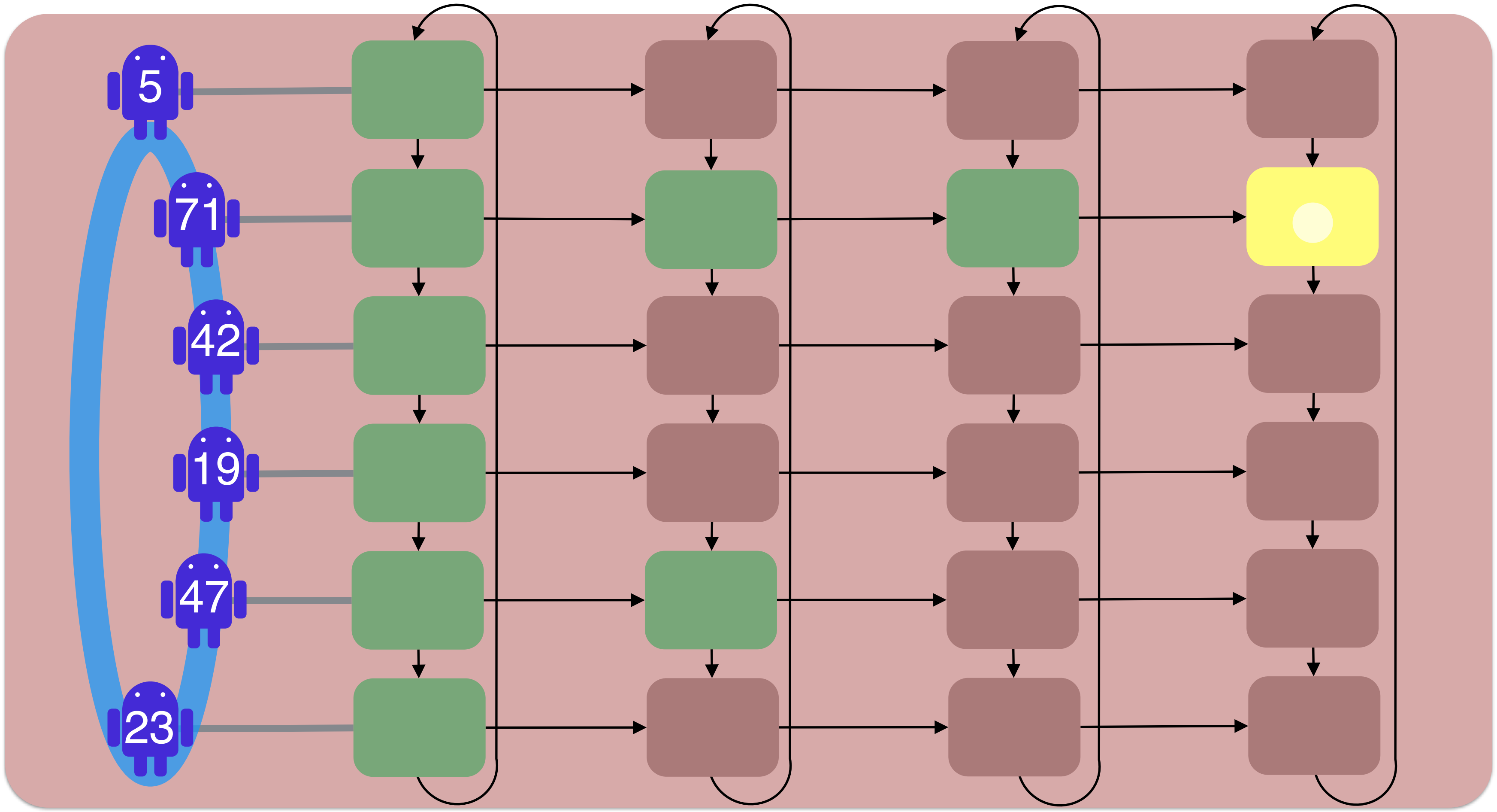


Distributed algorithm

«At the end, there is a leader, and
the leader is the process with the maximum id.»

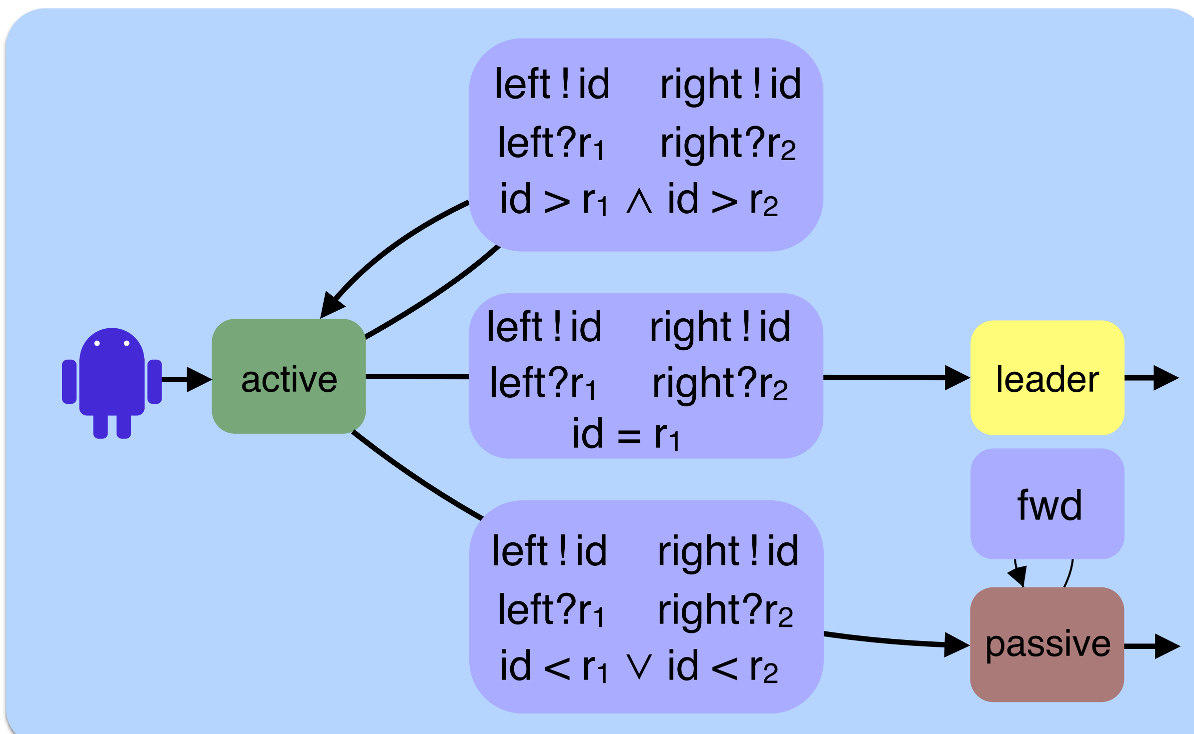
Specification

Distributed algorithms



Behavior

move in the cylinder



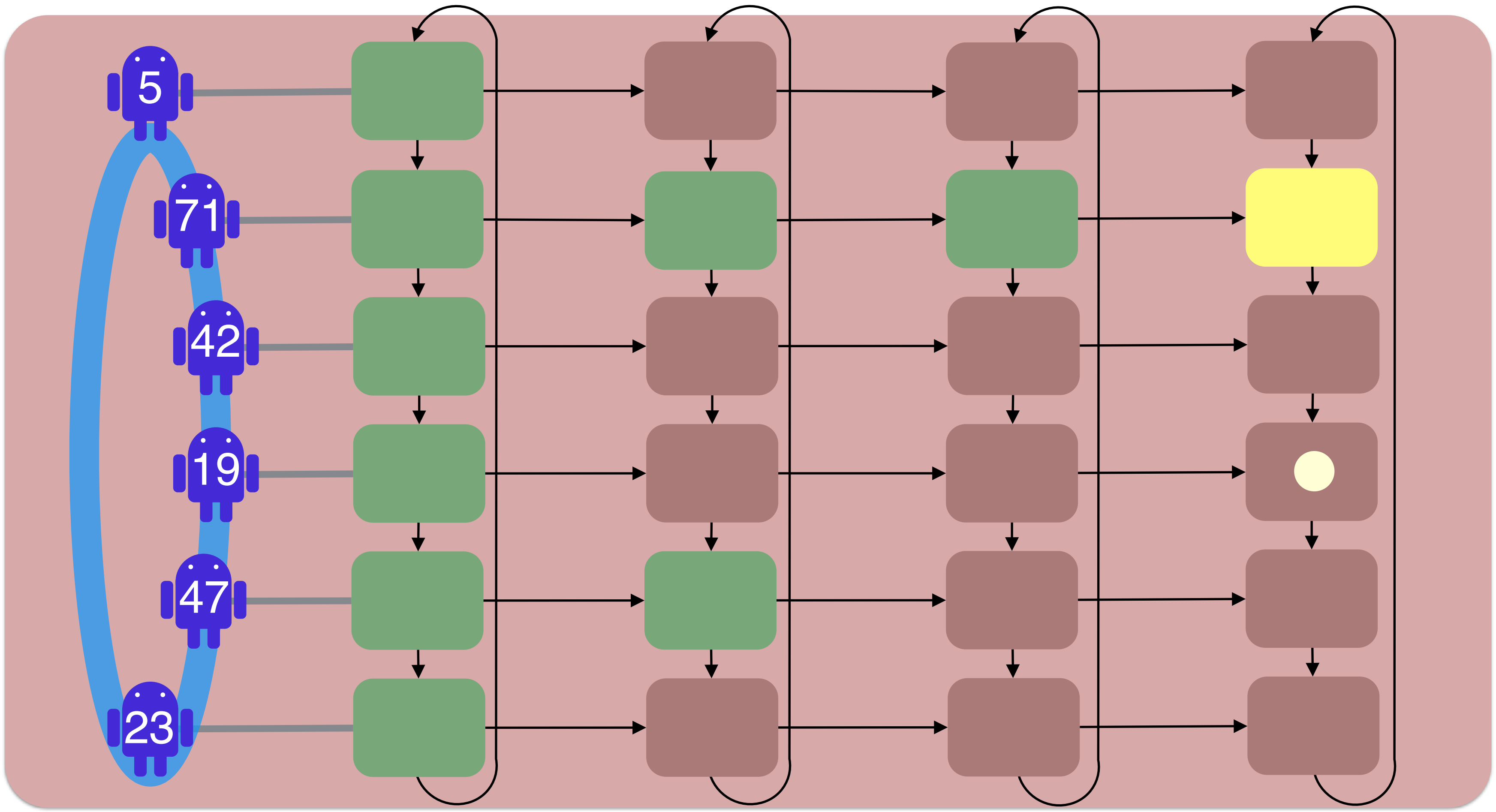
Distributed algorithm

«At the end, there is a leader, and
the leader is the process with the maximum id.»

Specification

Distributed algorithms

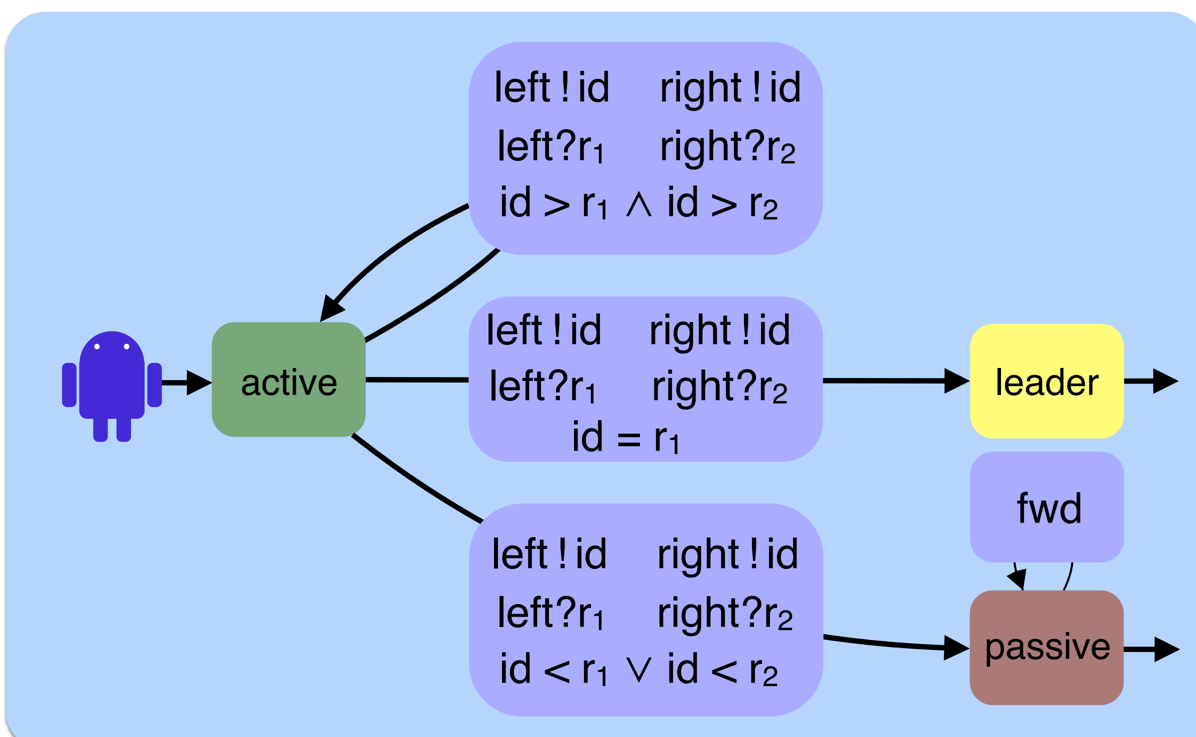
Leader election [Franklin ’82]



Behavior

move in the cylinder

compare values at different nodes



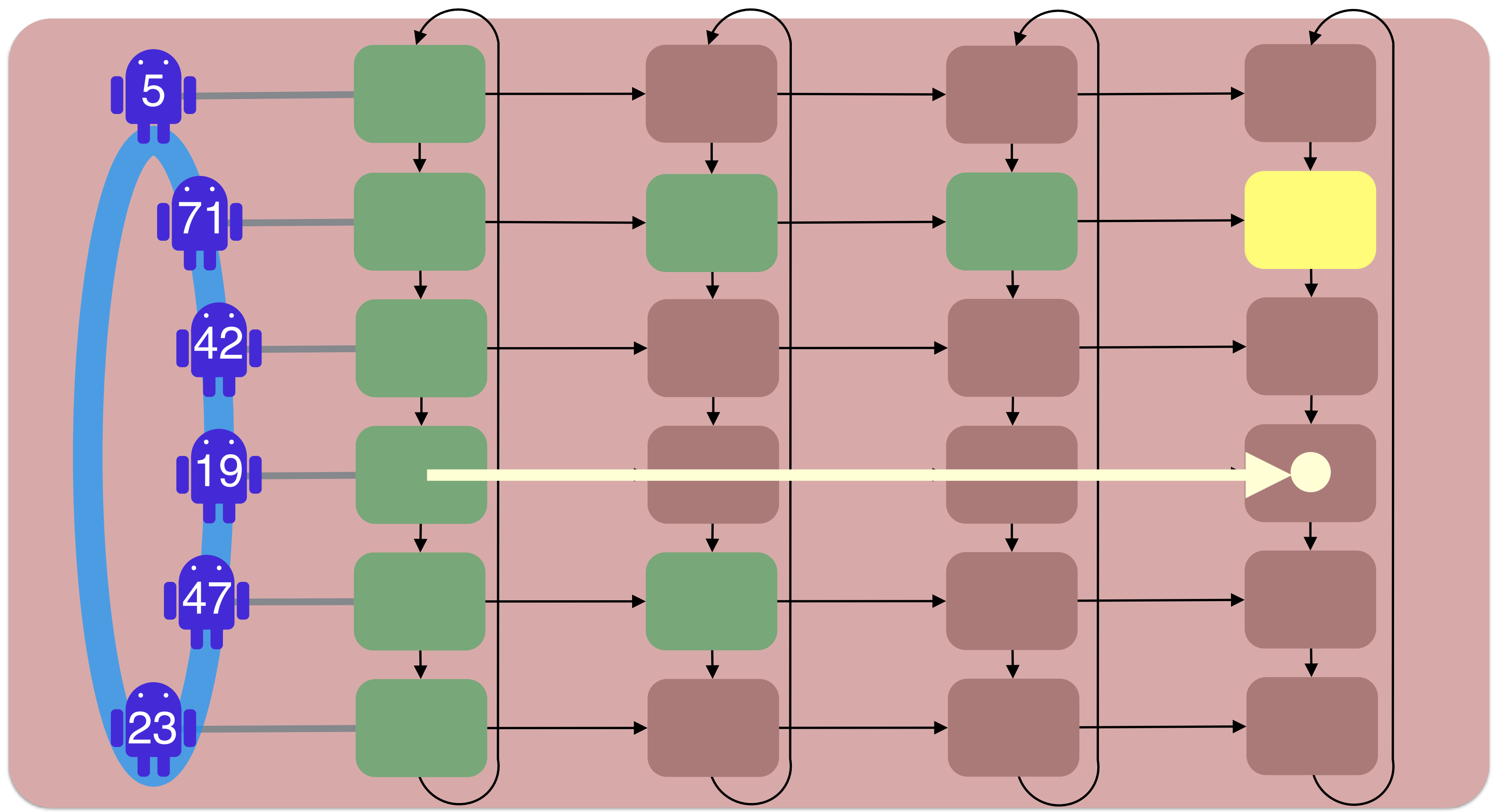
Distributed algorithm

«At the end, there is a leader, and
the leader is the process with the maximum id.»

Specification

Distributed algorithms

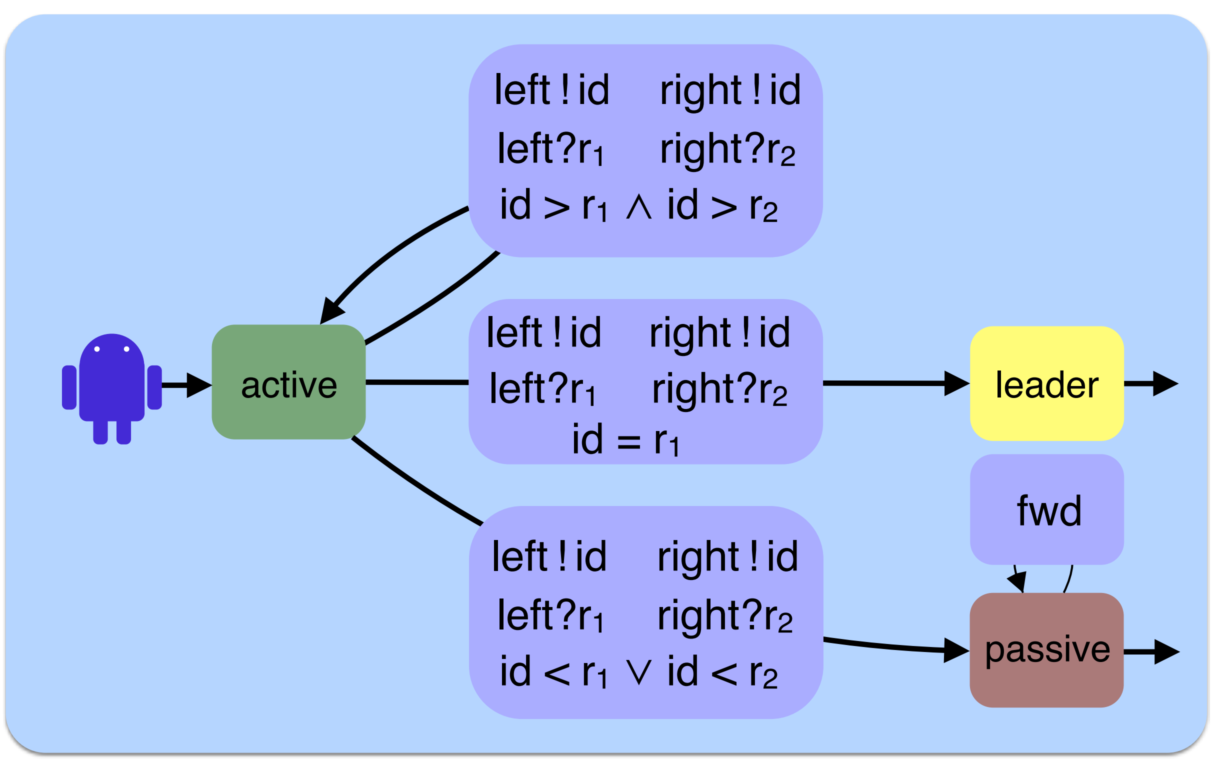
Leader election [Franklin ’82]



move in the cylinder

compare values at different nodes

Behavior



Distributed algorithm

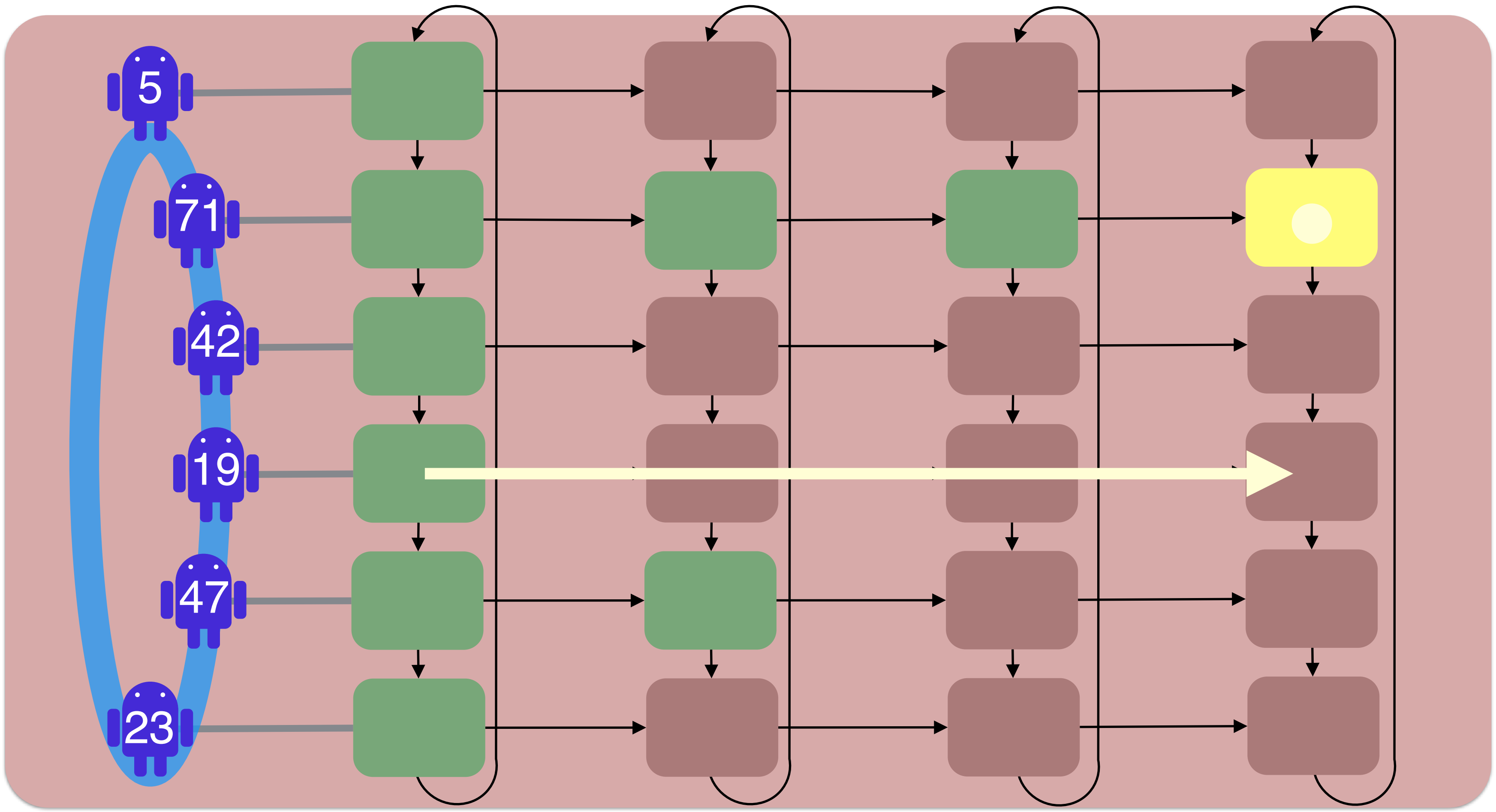
«At the end, there is a leader, and
the leader is the process with the maximum id.»

$$\langle \rightarrow^* \rangle \left(\neg\langle \rightarrow \rangle \wedge \langle \text{go-to-}\square \rangle \wedge [\downarrow^*] \left(id \leq \langle \text{go-to-}\square \rangle\, id \right) \right)$$

Data Propositional Dynamic Logic
[Bojanczyk et al. ’09; Figueira-Segoufin ‘11]

Distributed algorithms

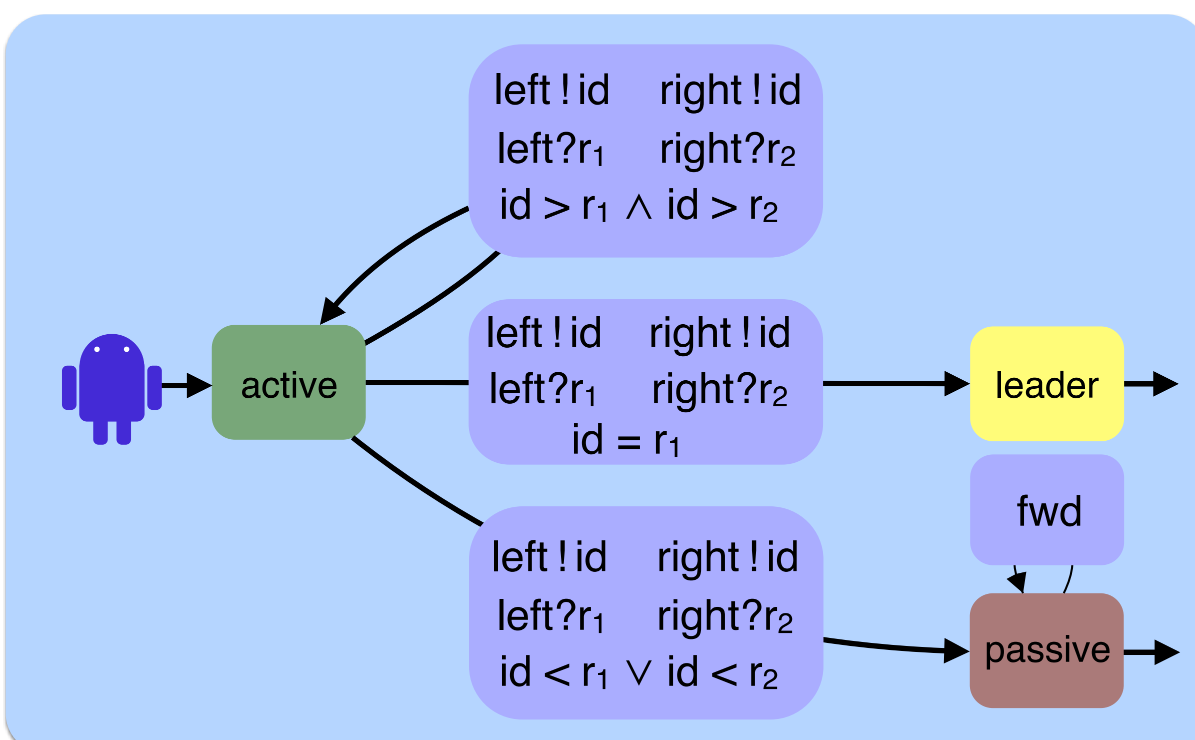
Leader election [Franklin '82]



move in the cylinder

compare values at different nodes

Behavior



Distributed algorithm

«At the end, there is a leader, and the leader is the process with the maximum id.»

$$\langle \rightarrow^* \rangle \left(\neg\langle \rightarrow \rangle \wedge \langle \text{go-to-}\square \rangle \wedge [\downarrow^*] \left(\mathbf{id} \leq \langle \text{go-to-}\square \rangle\, \mathbf{id} \right) \right)$$

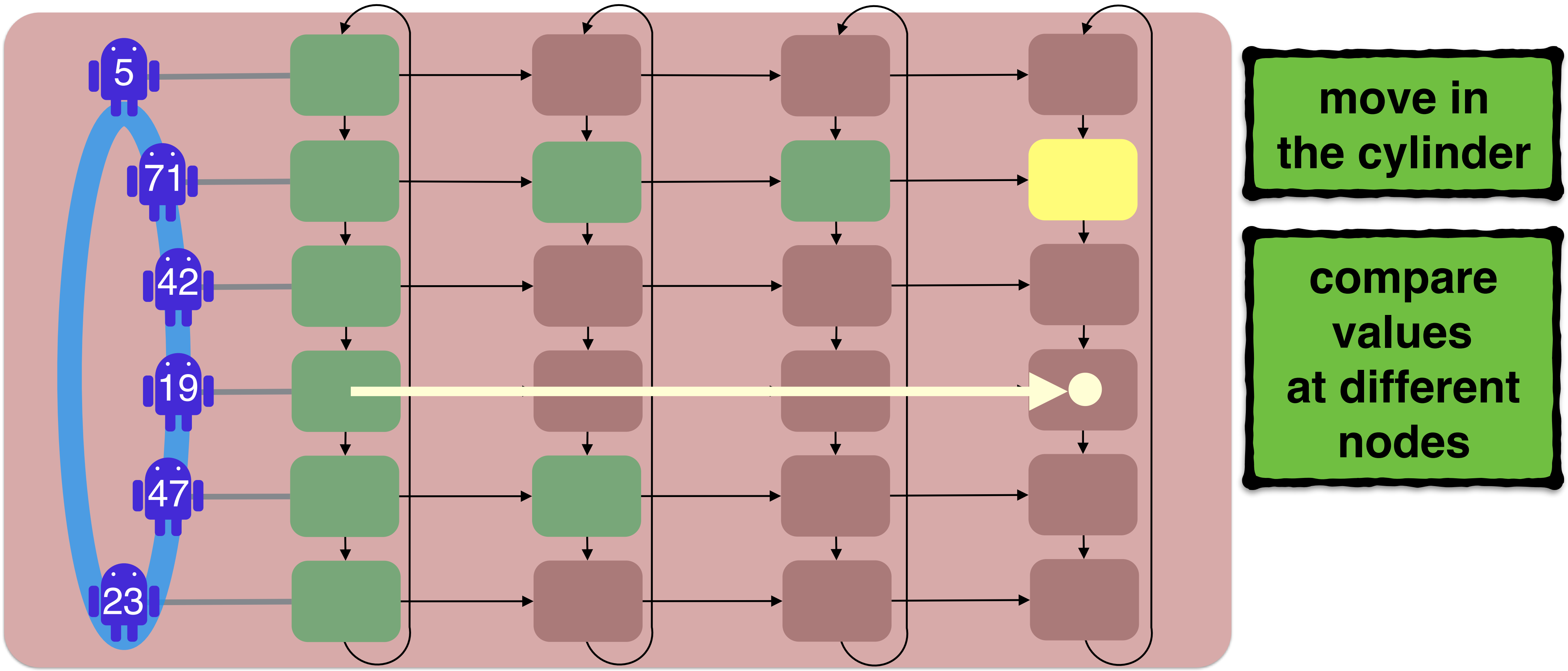
$$\text{go-to-}\square = (\neg \square \downarrow)^* \; \square$$

Data Propositional Dynamic Logic

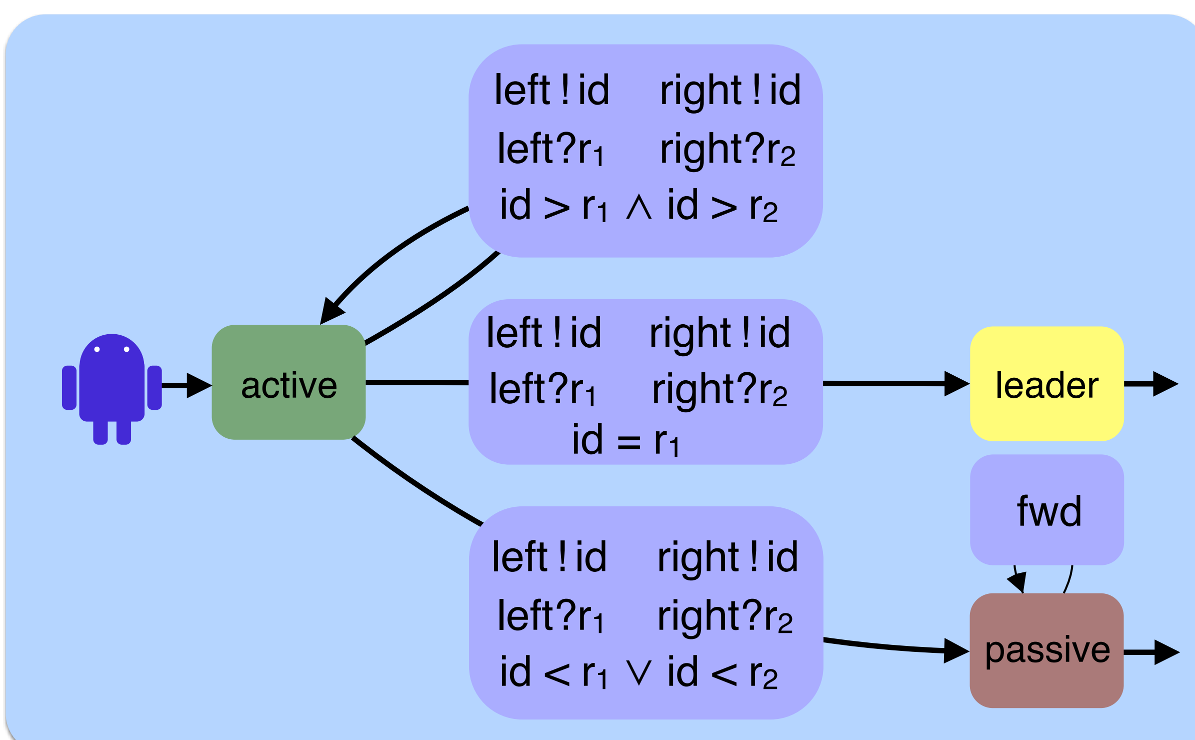
[Bojanczyk et al. '09; Figueira-Segoufin '11]

Distributed algorithms

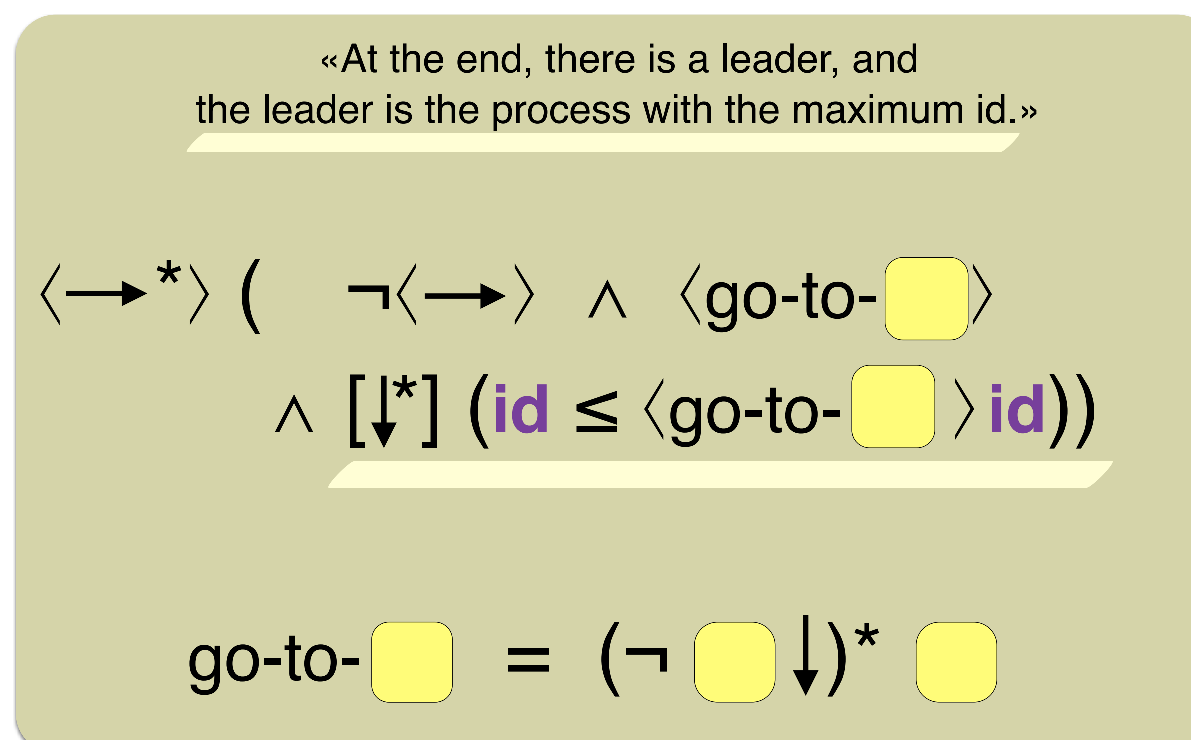
Leader election [Franklin '82]



Behavior



Distributed algorithm

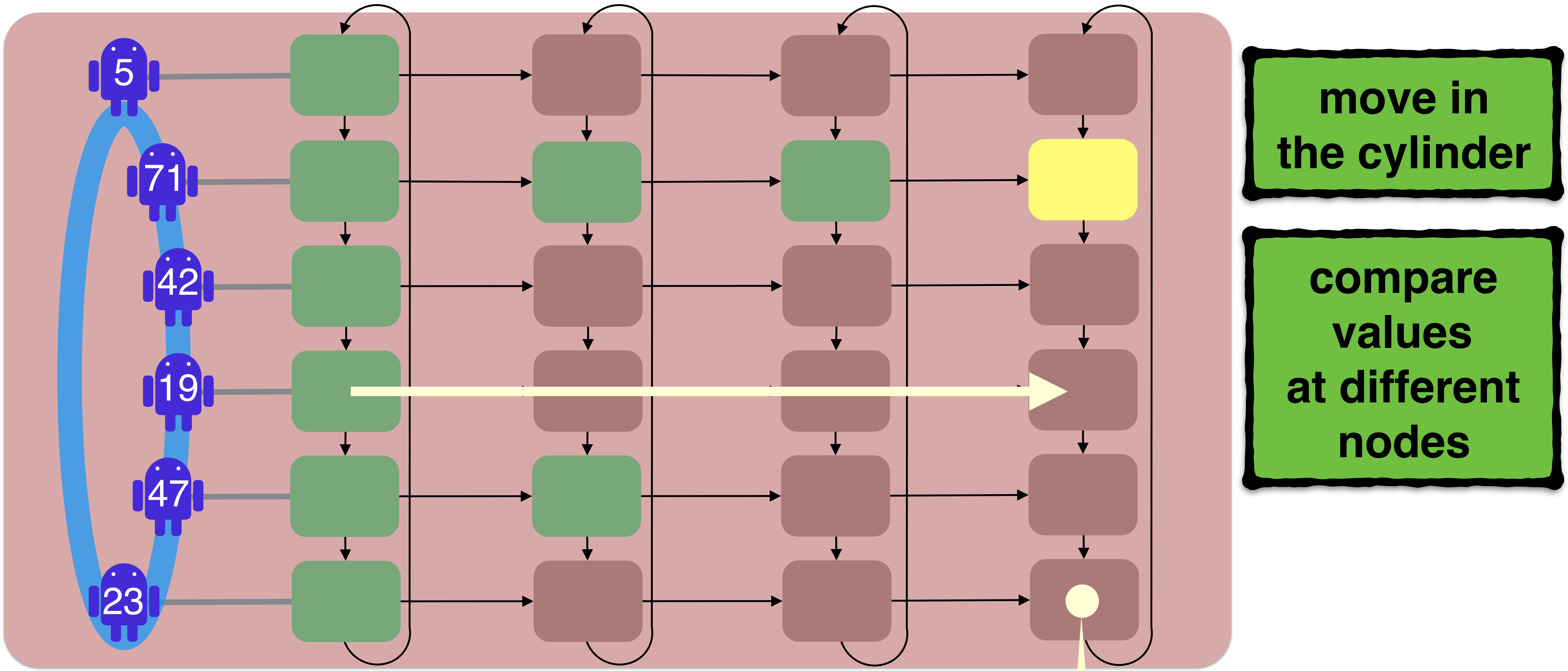


Data Propositional Dynamic Logic

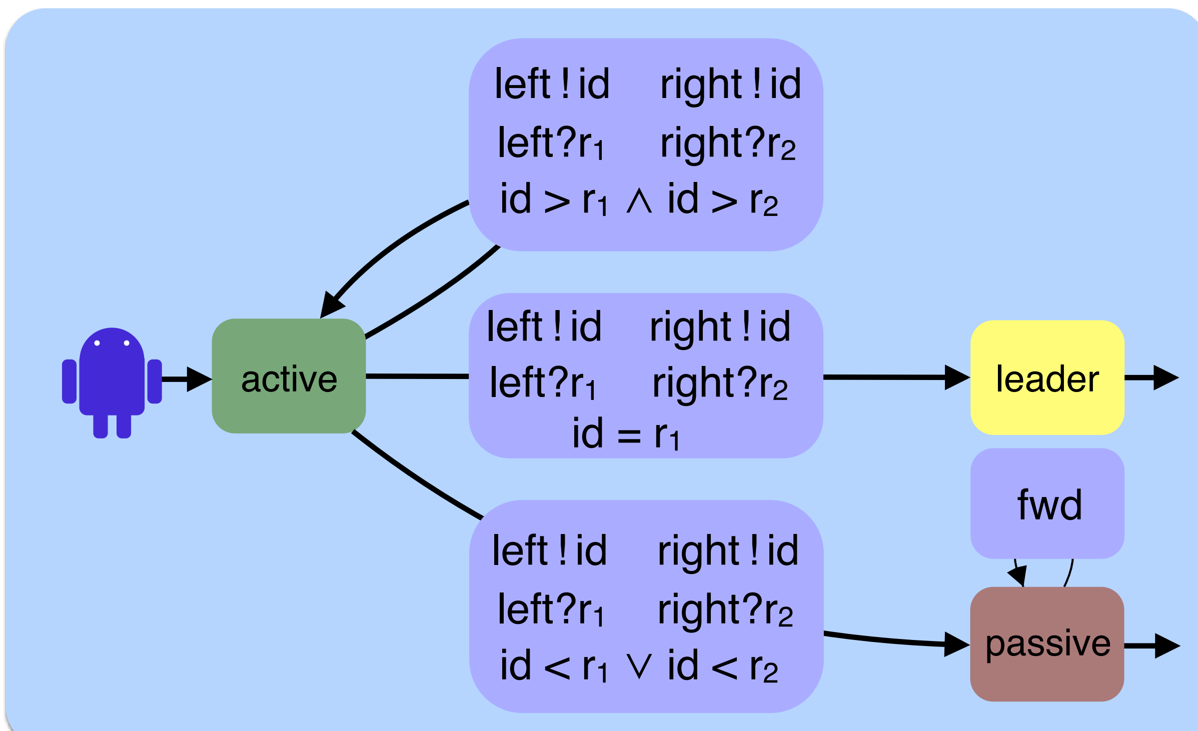
[Bojanczyk et al. '09; Figueira-Segoufin '11]

Distributed algorithms

Leader election [Franklin '82]



Behavior



Distributed algorithm

$\models$

«At the end, there is a leader, and
the leader is the process with the maximum id.»

For all *n*, pid distributions, and **accepting runs**:

$$\langle \longrightarrow^* \rangle \ (\quad \neg\langle \longrightarrow \rangle \quad \langle \text{go-to-}\square \rangle$$
$$\wedge\ [\downarrow^*]\ (\mathbf{id} \leq \langle \text{go-to-}\square \rangle\, \mathbf{id}))$$

$$\text{go-to-}\square \ = \ (\neg \square \downarrow)^* \ \square$$

Data Propositional Dynamic Logic
[Bojanczyk et al. '09; Figueira-Segoufin '11]

Specifications: distributed sorting

The output values form a permutation of the input values

- same set of values:

$$[\rightarrow^*](\langle\varepsilon\rangle r = \langle\uparrow^*\{\neg\langle\uparrow\rangle\}?\rightarrow^*\rangle r)$$

- pairwise distinct:

$$\neg\langle\rightarrow^*\rangle(\langle\varepsilon\rangle r = \langle(\rightarrow\{\neg\ddagger\}?)^+\rangle r)$$

$$\Phi, \Phi' ::= \mathsf{A}\,\phi \mid \Phi \wedge \Phi'$$

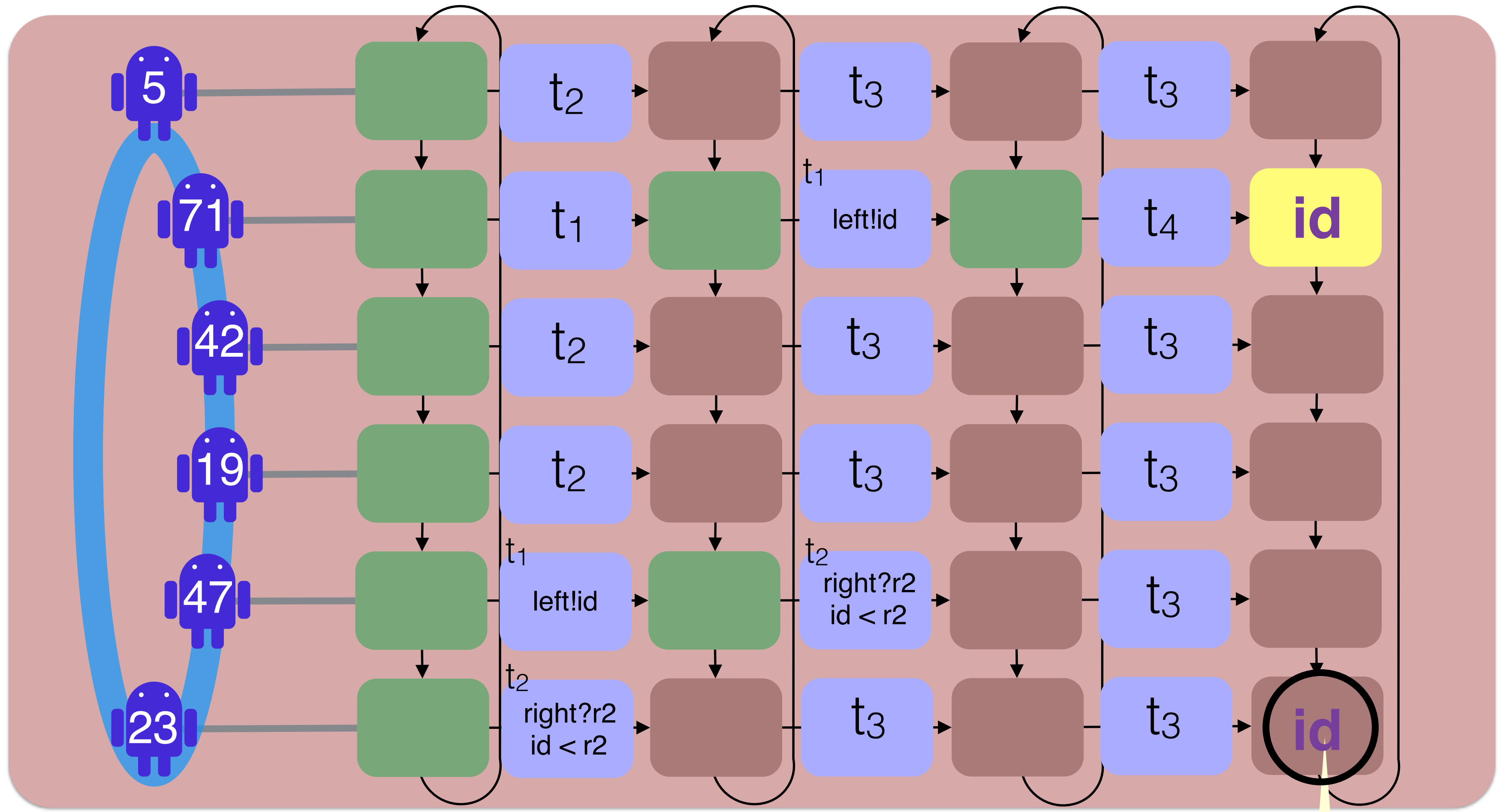
$$\phi, \phi' ::= \varphi \mid \phi \wedge \phi' \mid \varphi \vee \phi \mid [\pi]\phi \mid \langle\eta\rangle r < \langle\eta'\rangle r' \mid \langle\eta\rangle r \leq \langle\eta'\rangle r'$$

$$\varphi, \varphi' ::= \ddagger \mid p \mid \neg\varphi \mid \varphi \wedge \varphi' \mid \langle\pi\rangle\varphi \mid \langle\pi\rangle r = \langle\pi'\rangle r' \mid \langle\pi\rangle r \neq \langle\pi'\rangle r'$$

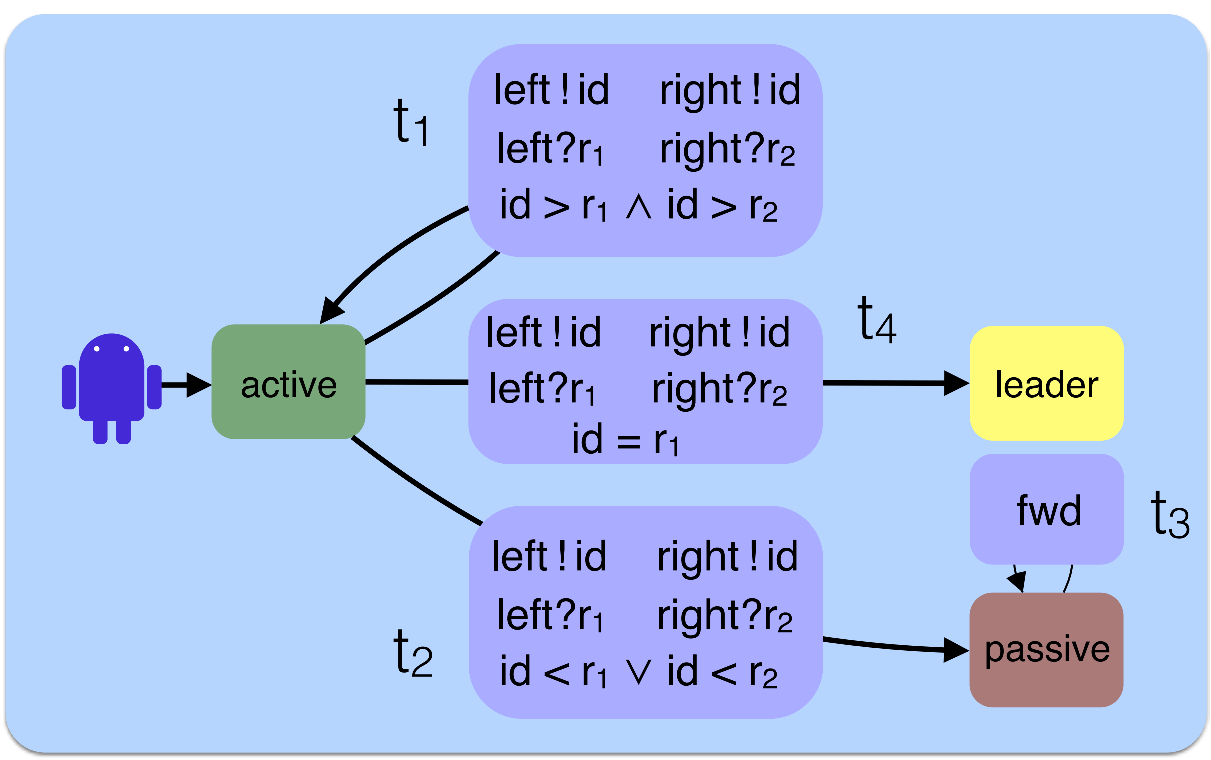
$$\pi, \pi' ::= \{\varphi\}? \mid \rightarrow \mid \downarrow \mid \pi^{-1} \mid \pi + \pi' \mid \pi \cdot \pi' \mid \pi^*$$

$$\eta, \eta' ::= \{\varphi\}? \mid \leftarrow \mid \rightarrow \mid \downarrow \mid \uparrow \mid \eta \cdot \eta' \mid \mathsf{F}^{\eta}_{\varphi}$$

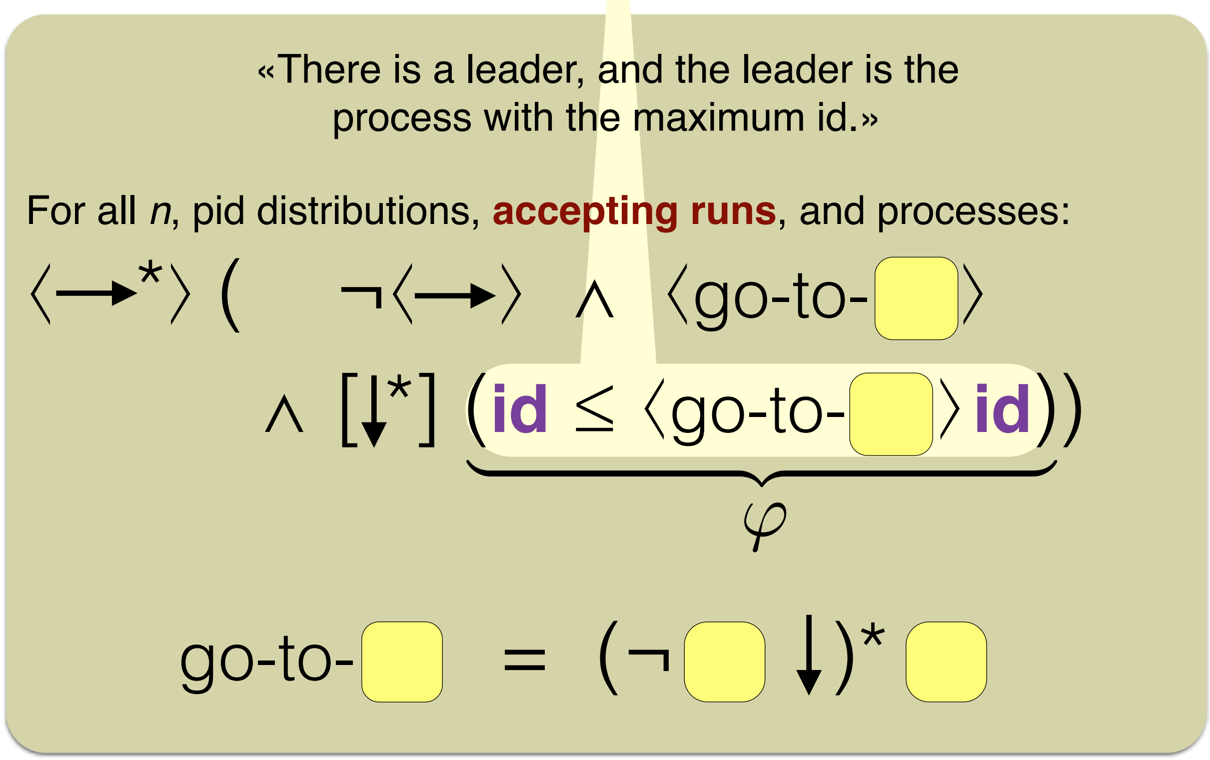
Distributed algorithms



Behavior

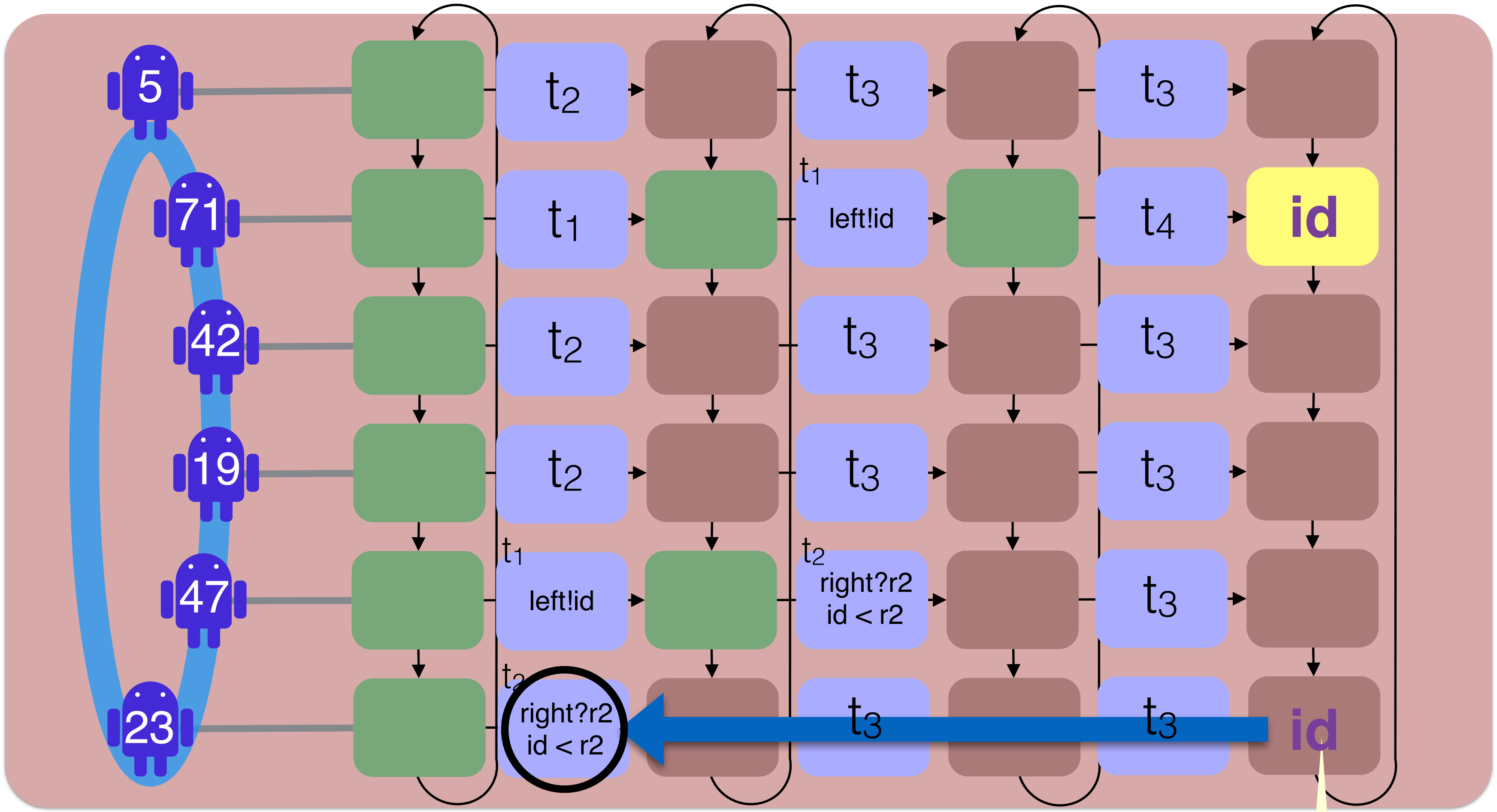


Distributed algorithm

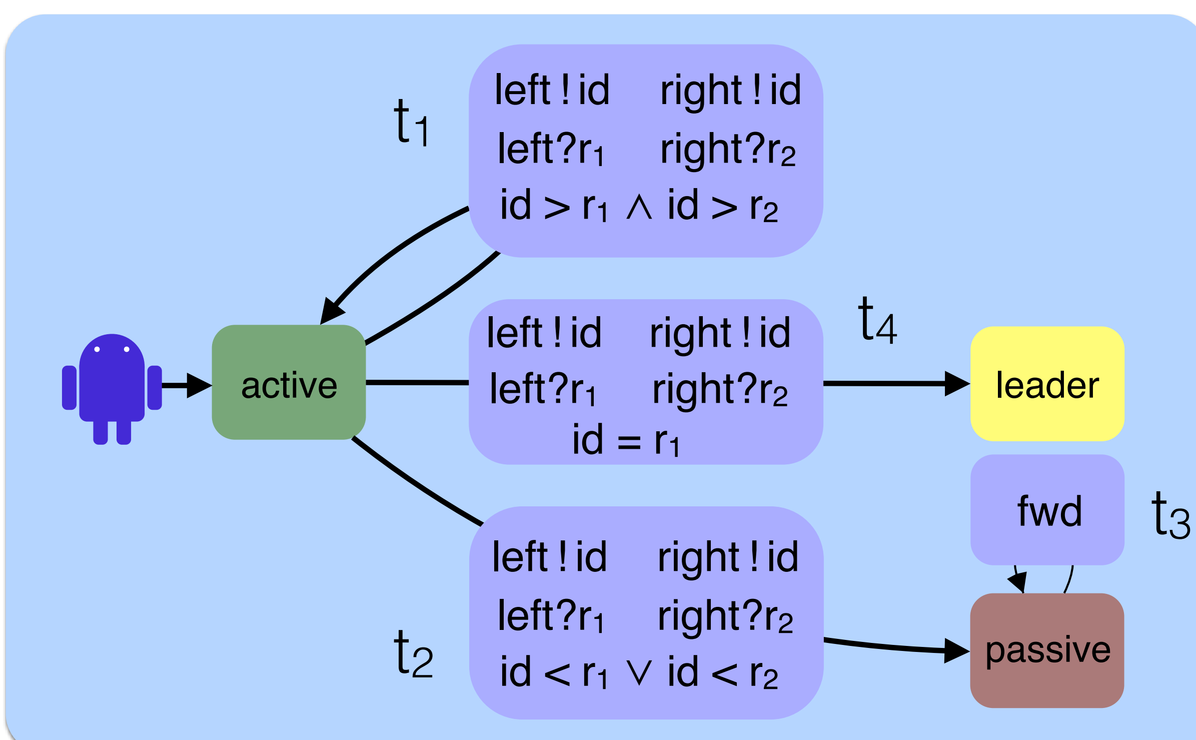


Data PDL

Distributed algorithms



Behavior



Distributed algorithm

«There is a leader, and the leader is the process with the maximum id.»

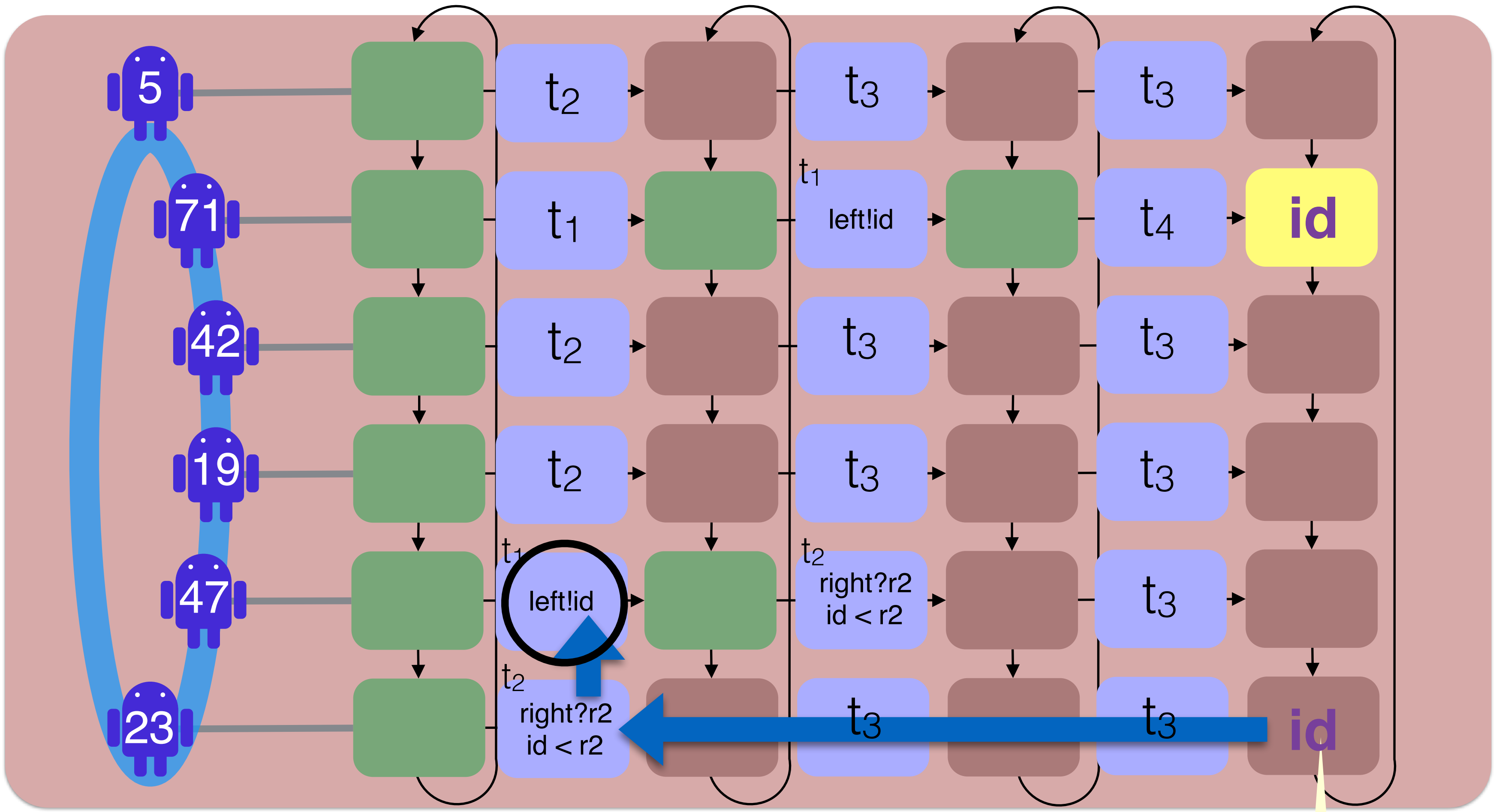
For all n, pid distributions, **accepting runs**, and processes:

$$\langle \rightarrow^* \rangle \left(\neg \langle \rightarrow \rangle \wedge \langle \text{go-to-}\square \rangle \wedge [\downarrow^*] \underbrace{(\mathbf{id} \leq \langle \text{go-to-}\square \rangle \mathbf{id})}_{\varphi} \right)$$

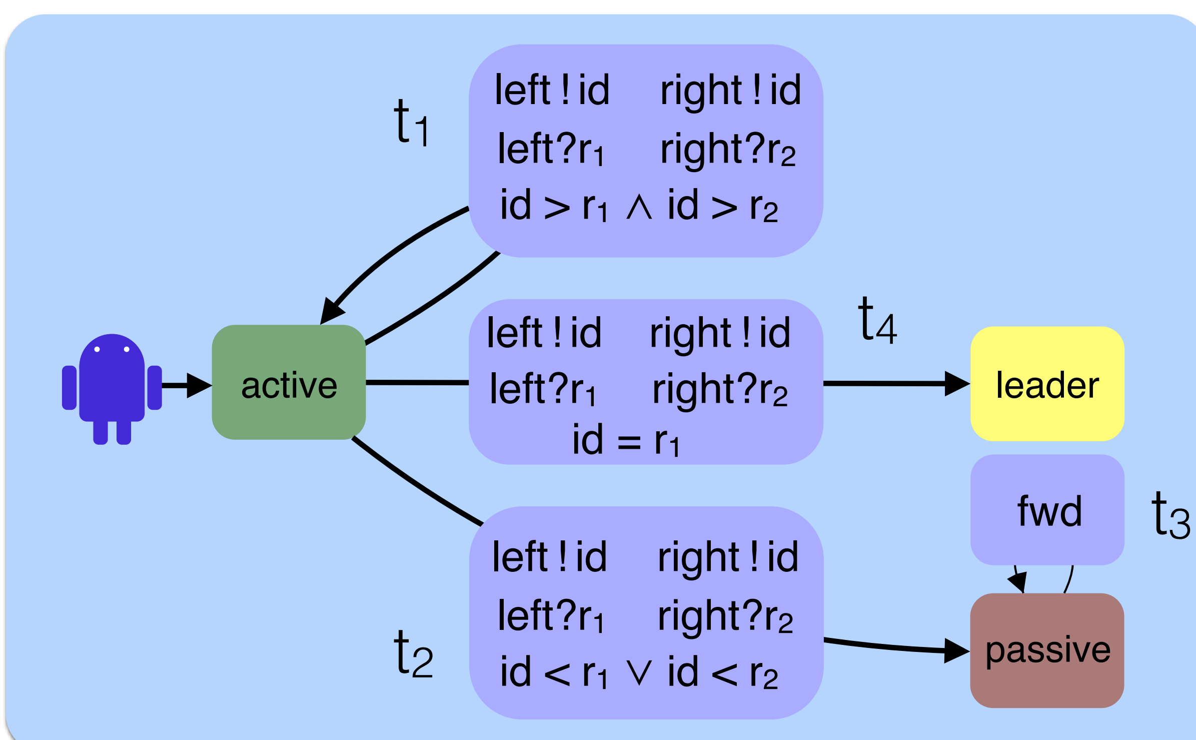
$$\text{go-to-}\square = (\neg \square \downarrow)^* \square$$

Data PDL

Distributed algorithms



Behavior



Distributed algorithm

«There is a leader, and the leader is the process with the maximum id.»

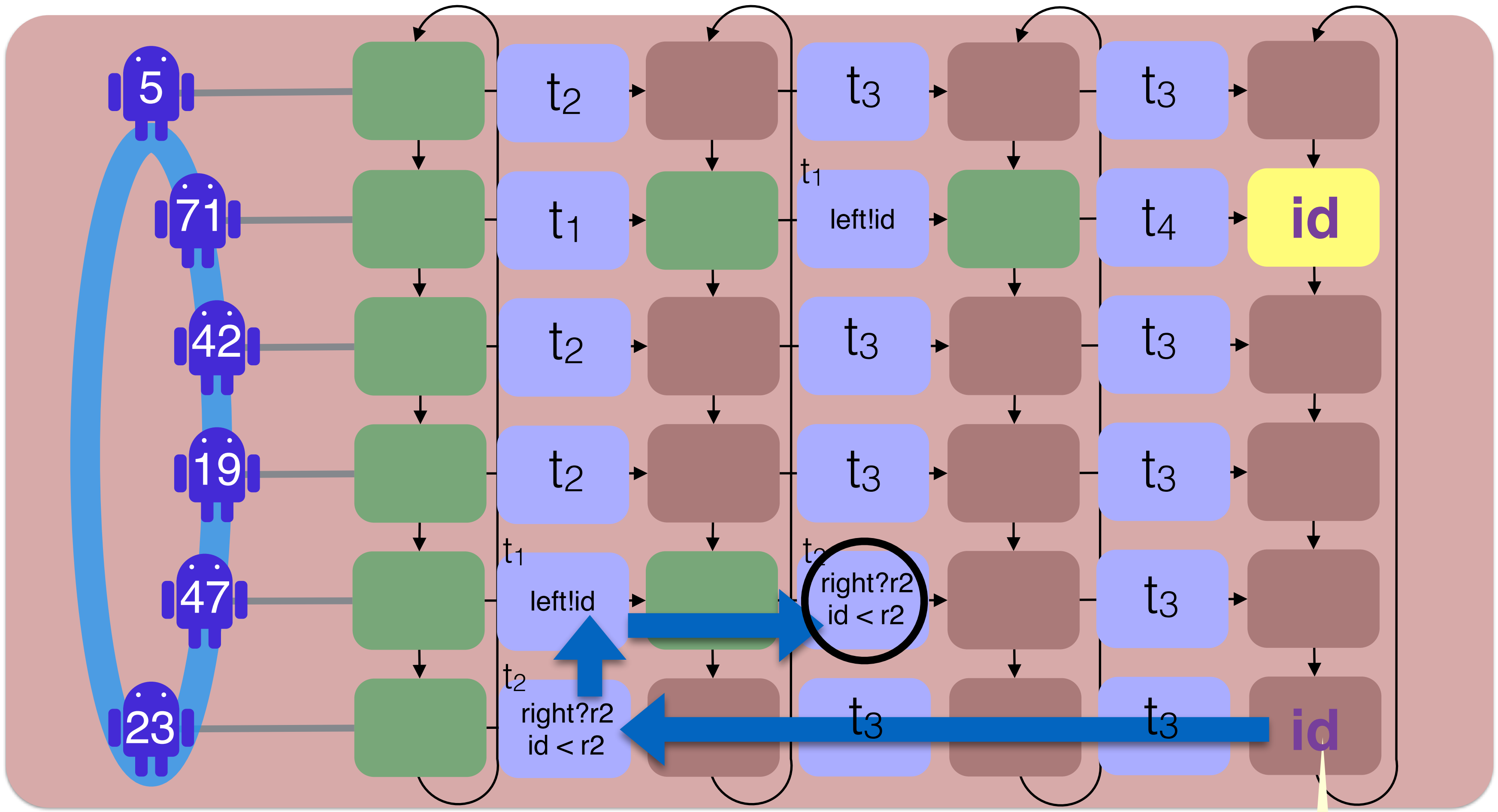
For all *n*, pid distributions, **accepting runs**, and processes:

$$\langle \rightarrow^* \rangle \left(\neg\langle \rightarrow \rangle \wedge \langle \text{go-to-}\square \rangle \wedge [\downarrow^*] \underbrace{\left(\mathbf{id} \leq \langle \text{go-to-}\square \rangle\, \mathbf{id}\right)}_{\varphi} \right)$$

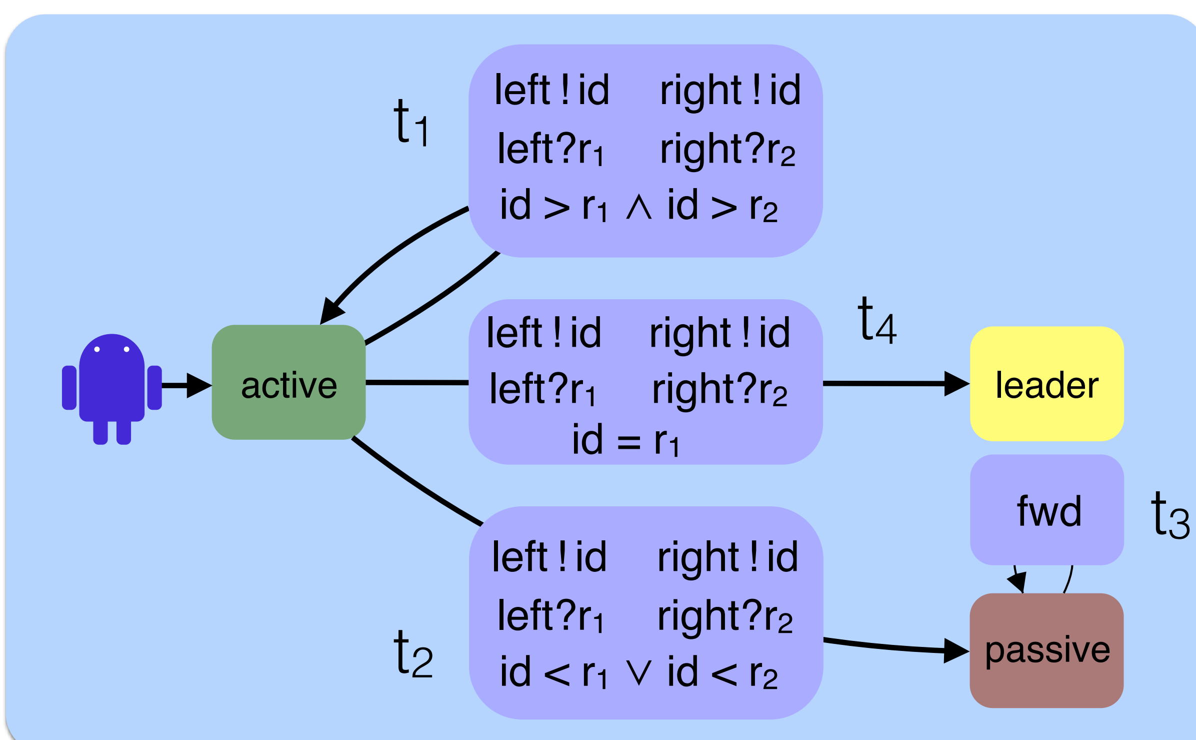
$$\text{go-to-}\square = (\neg \square \downarrow)^* \, \square$$

Data PDL

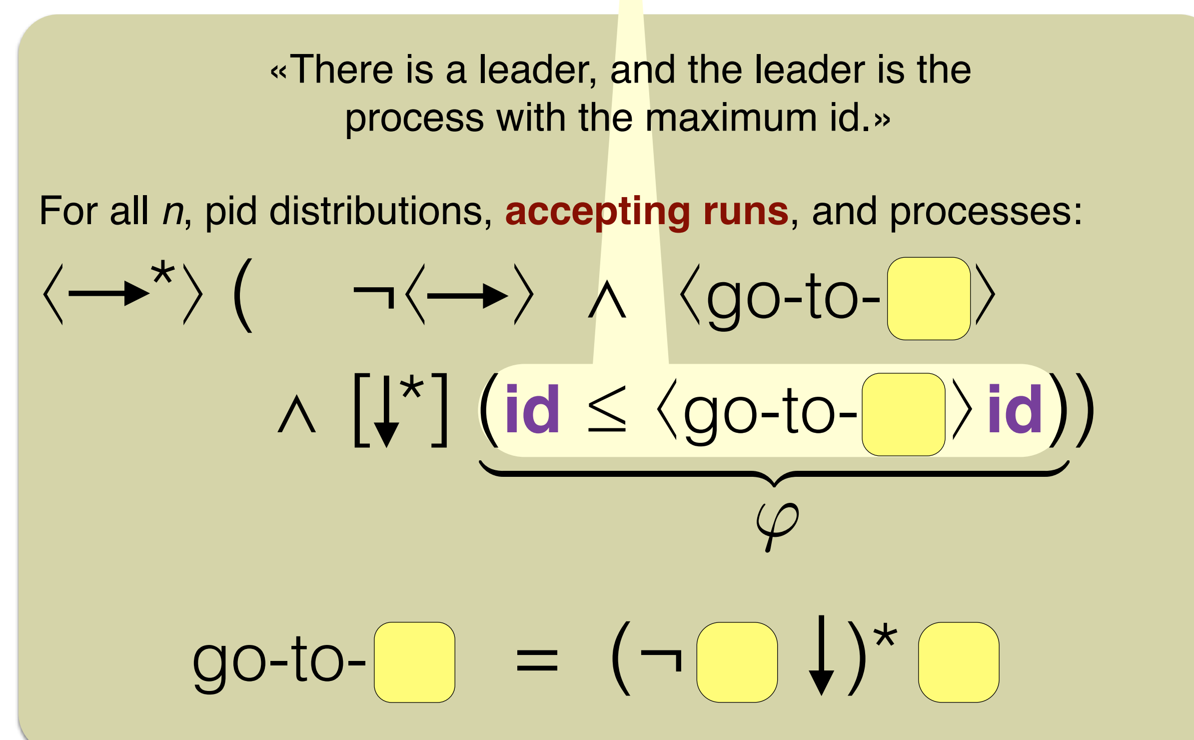
Distributed algorithms



Behavior

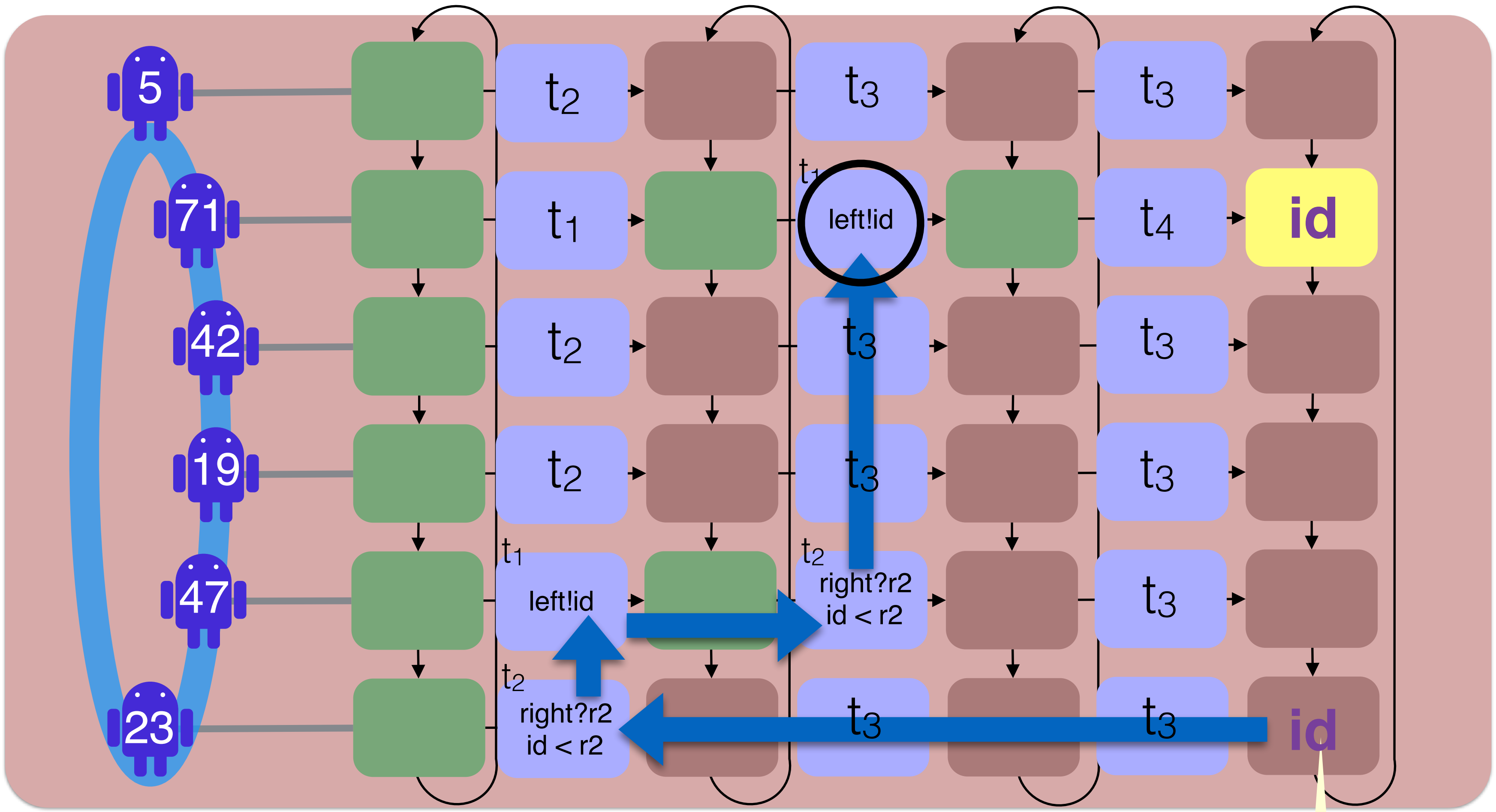


Distributed algorithm

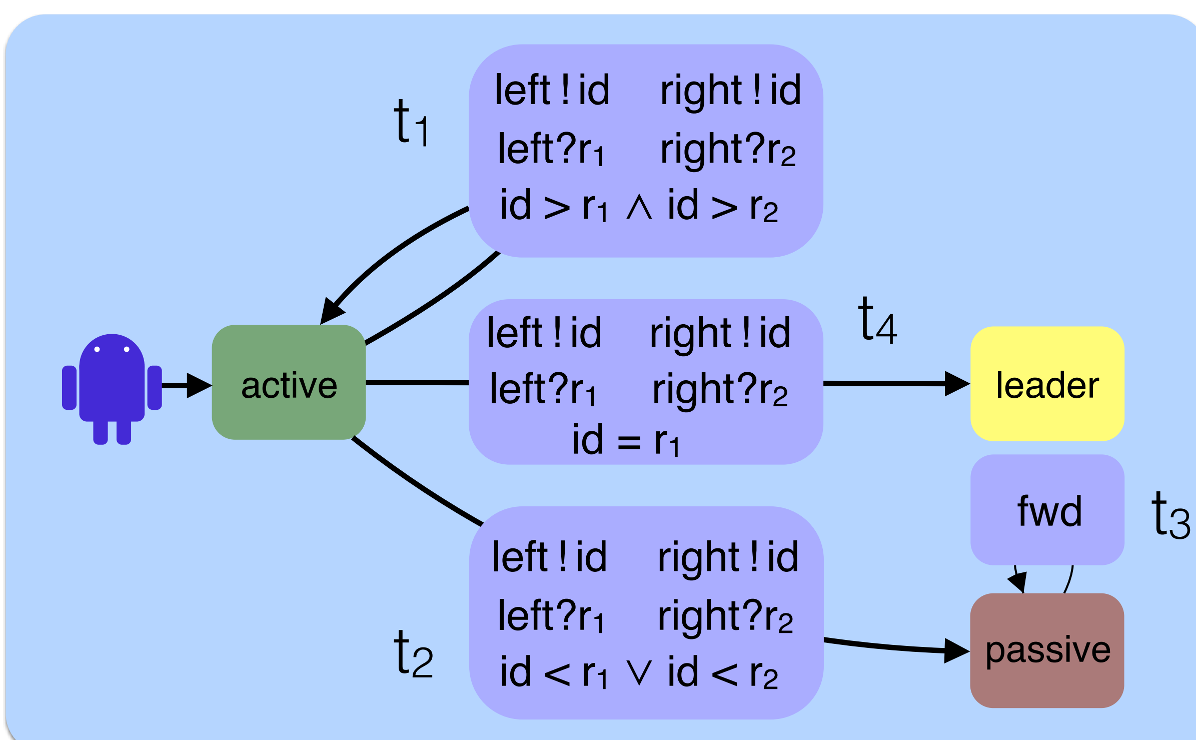


Data PDL

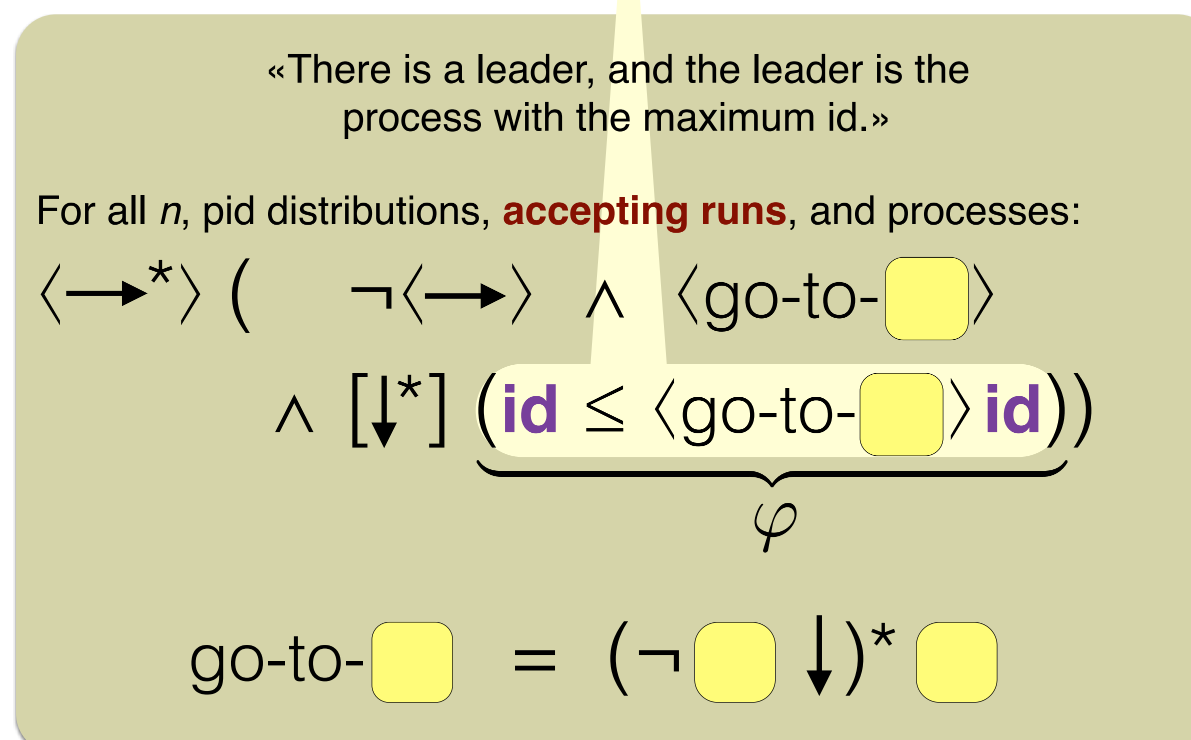
Distributed algorithms



Behavior

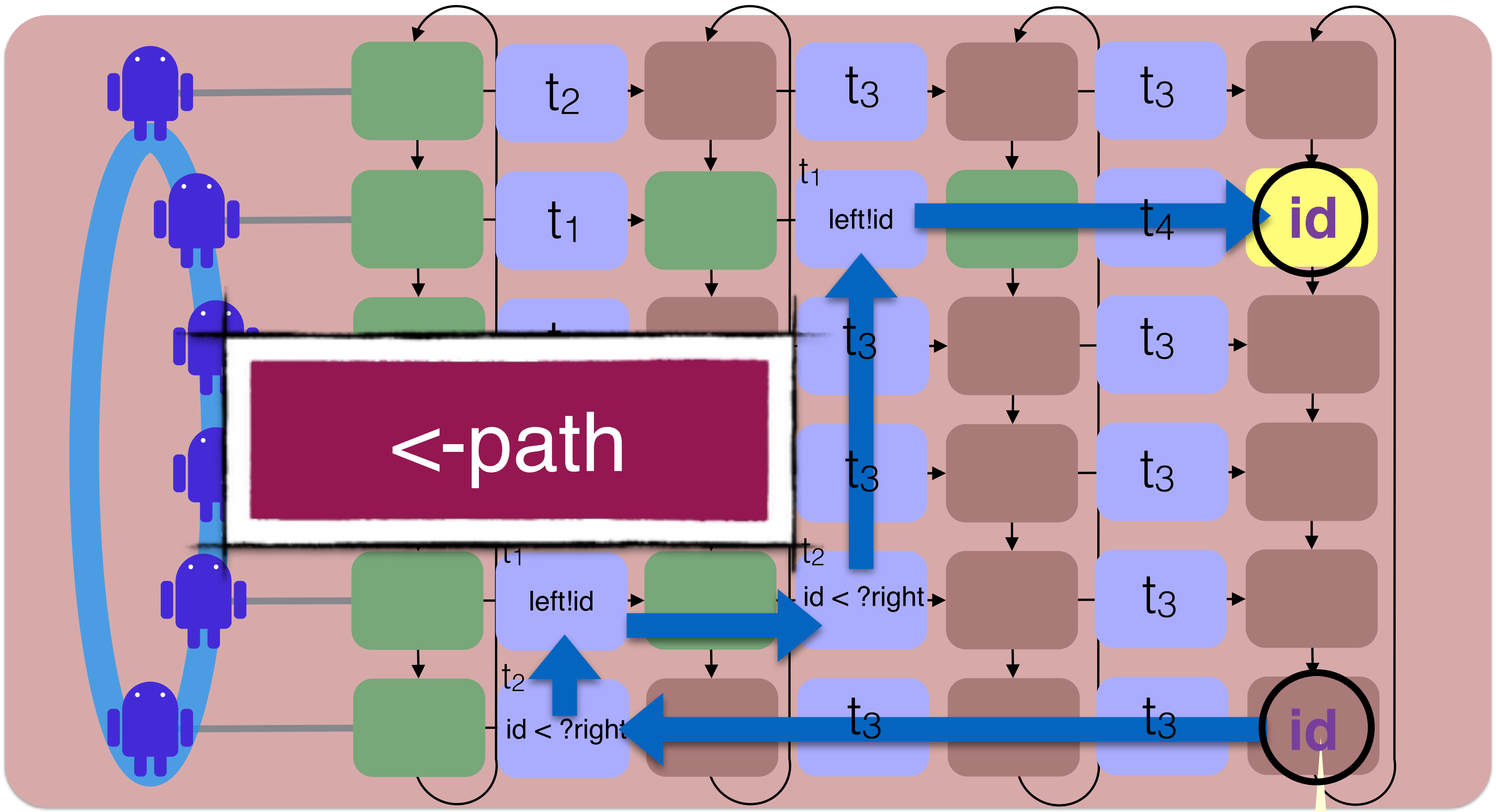


Distributed algorithm

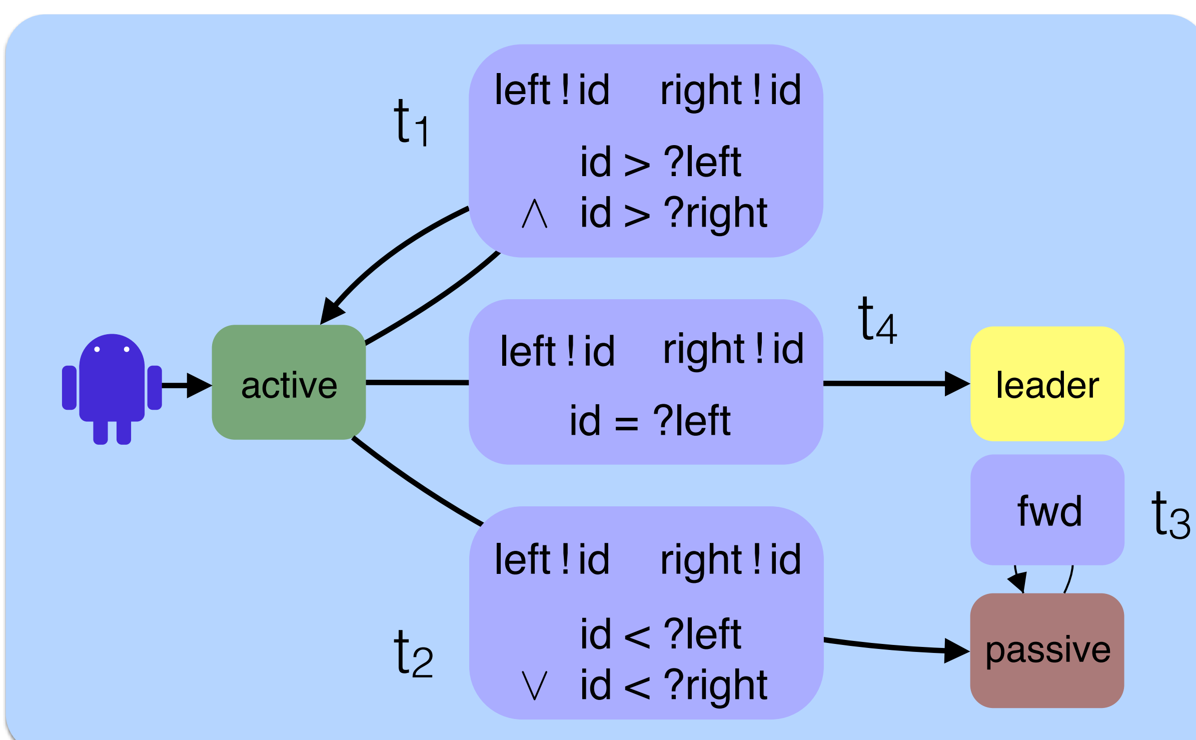


Data PDL

Distributed algorithms



Behavior



Distributed algorithm

«There is a leader, and the leader is the process with the maximum id.»

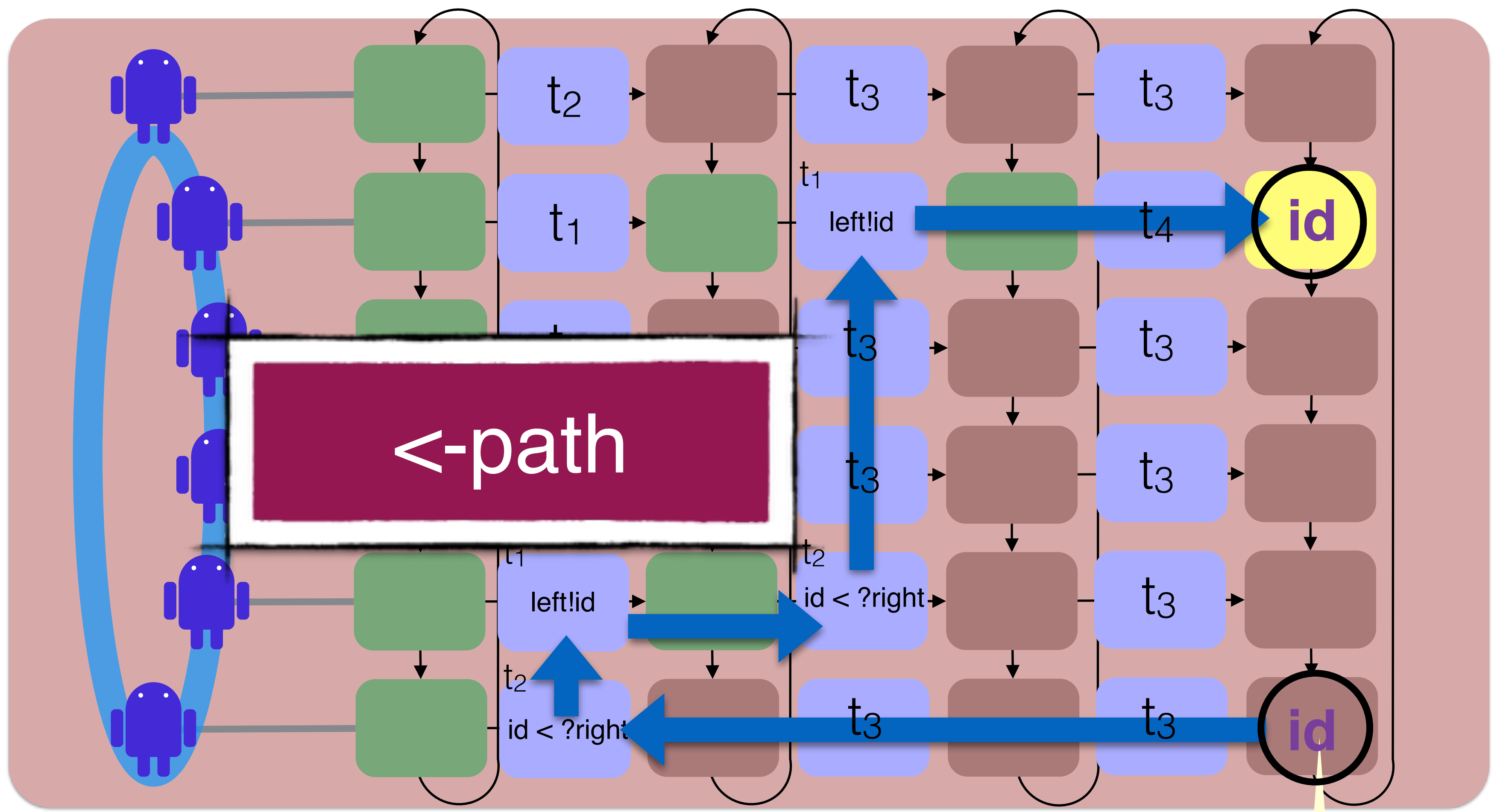
For all *n*, pid distributions, **accepting runs**, and processes:

$$\langle \rightarrow^* \rangle \left(\neg \langle \rightarrow \rangle \wedge \langle \text{go-to-}\square \rangle \wedge [\downarrow^*] \underbrace{(\mathbf{id} \leq \langle \text{go-to-}\square \rangle\, \mathbf{id})}_{\varphi} \right)$$

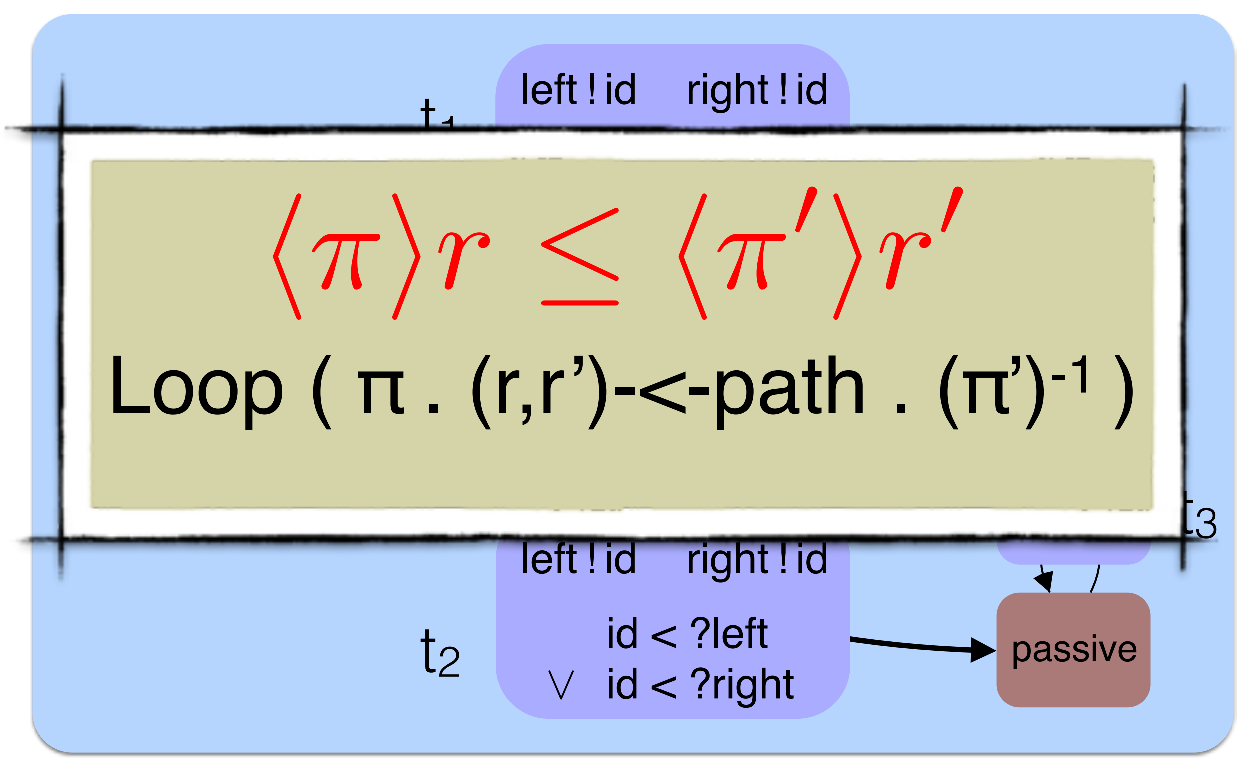
$$\text{go-to-}\square = (\neg \square \downarrow)^* \square$$

Data PDL

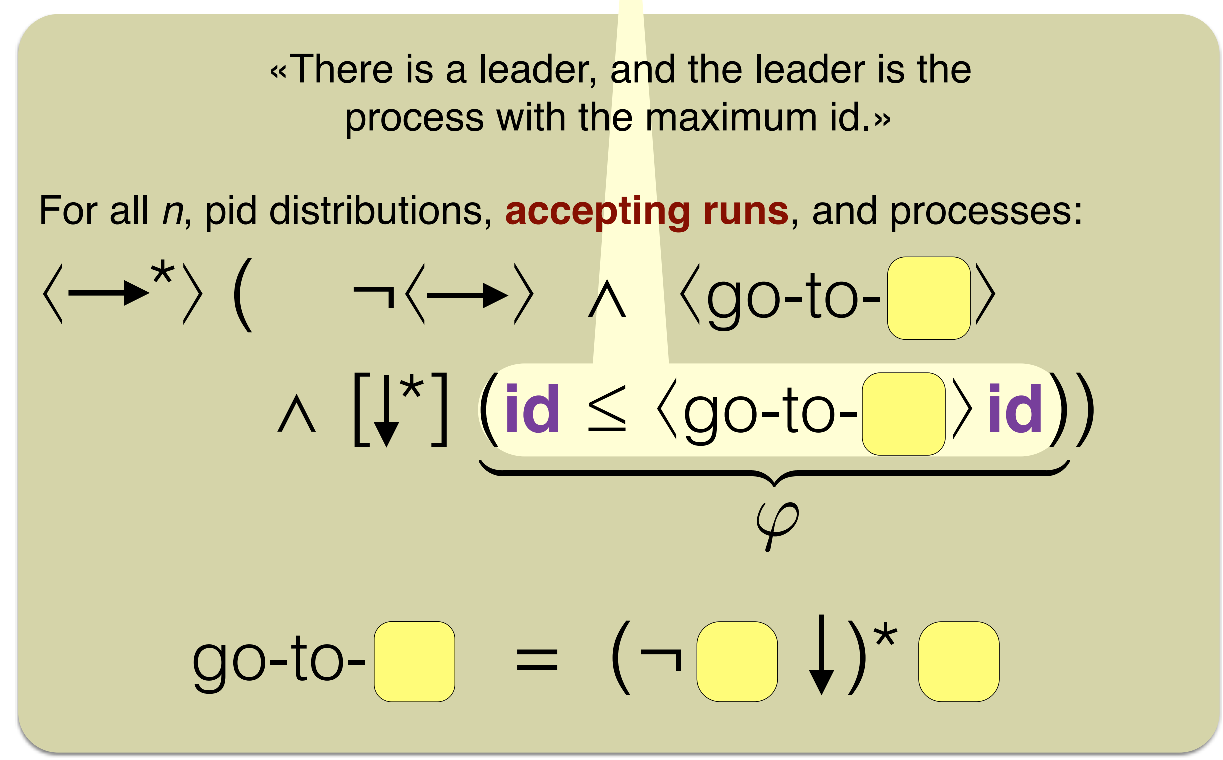
Distributed algorithms



Behavior

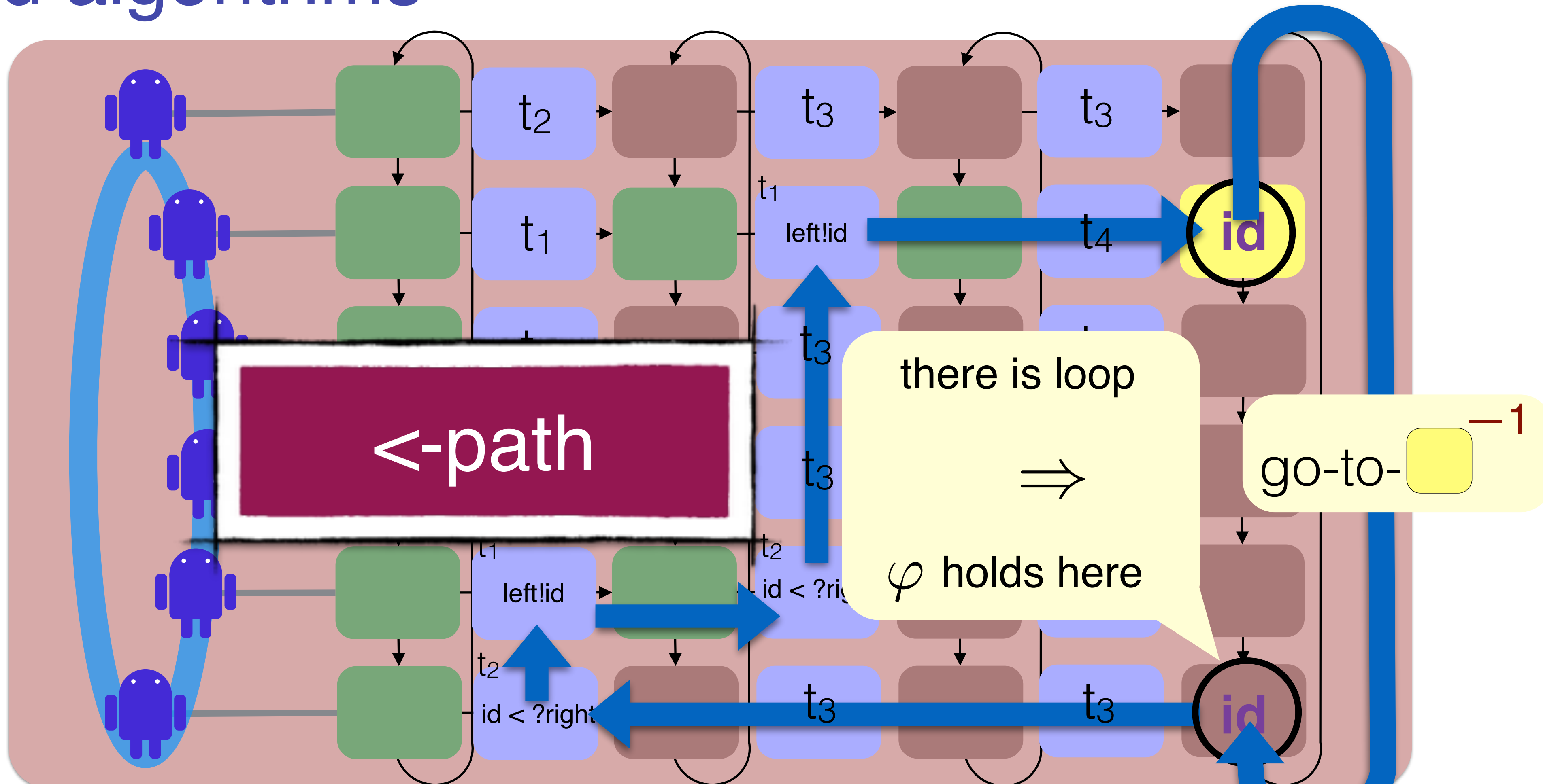


Distributed algorithm

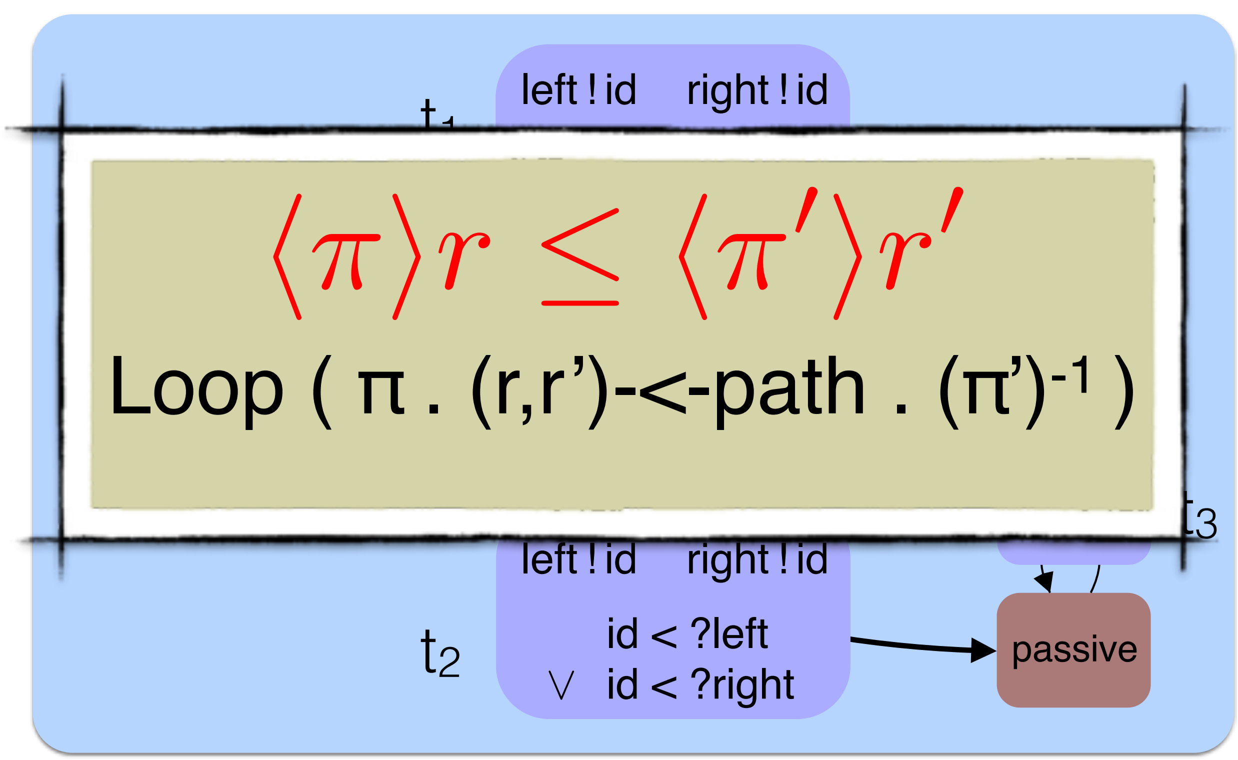


Data PDL

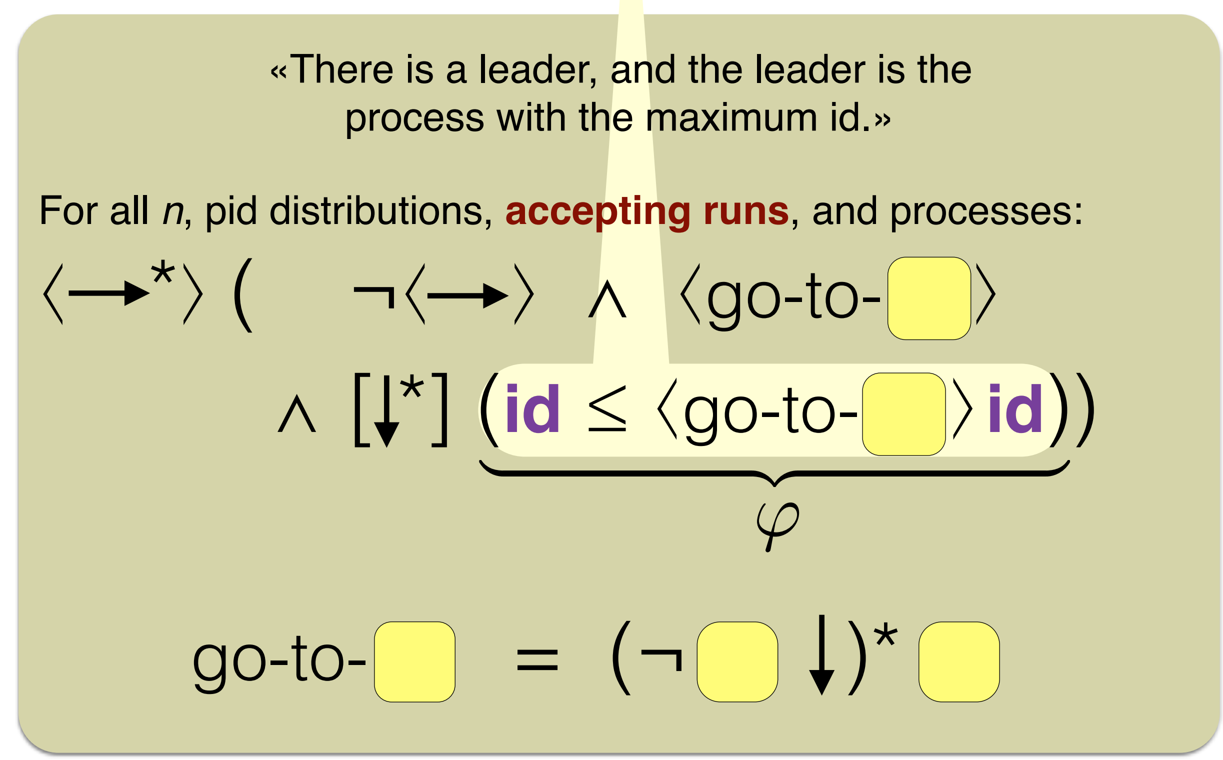
Distributed algorithms



Behavior

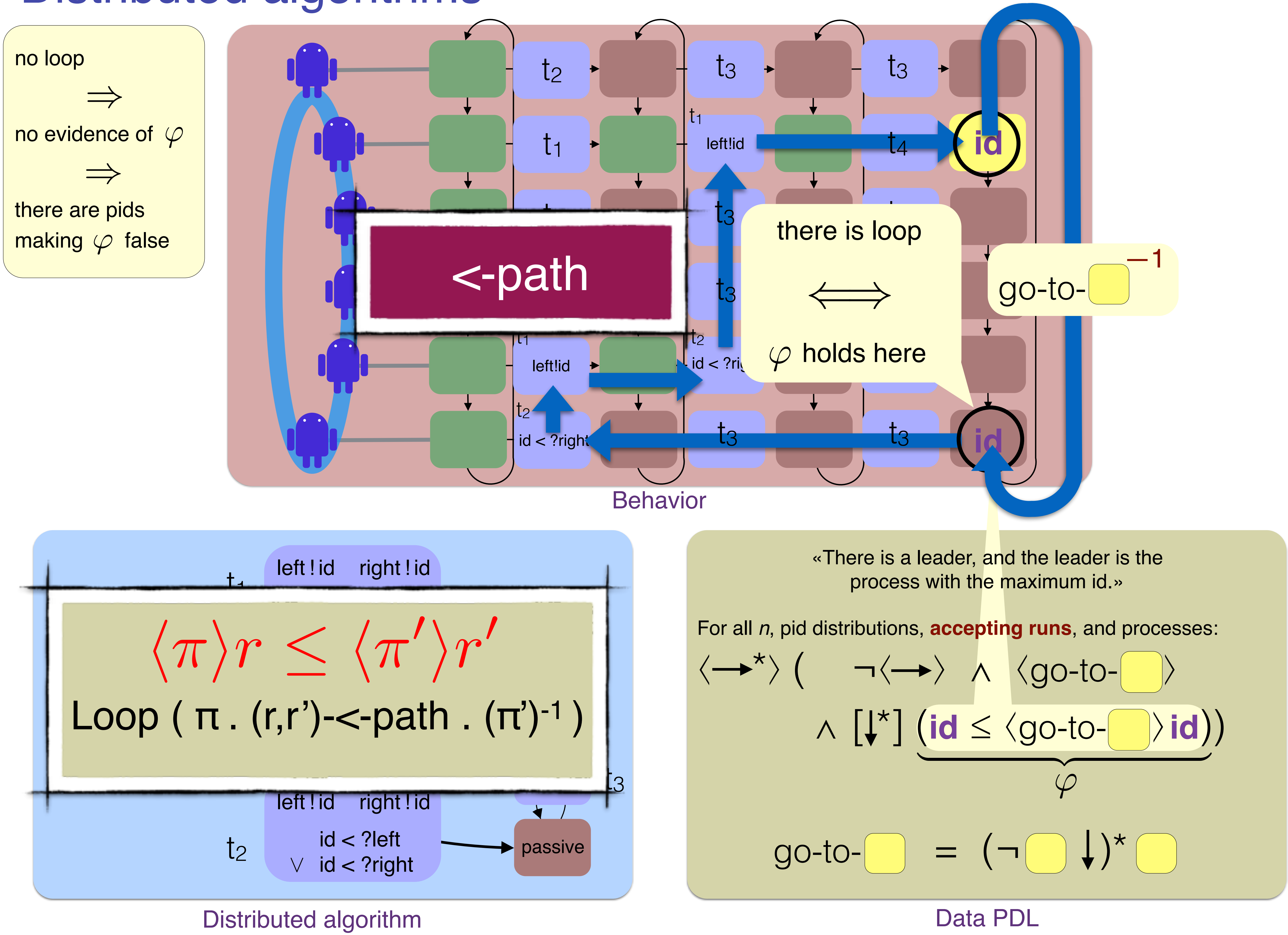


Distributed algorithm

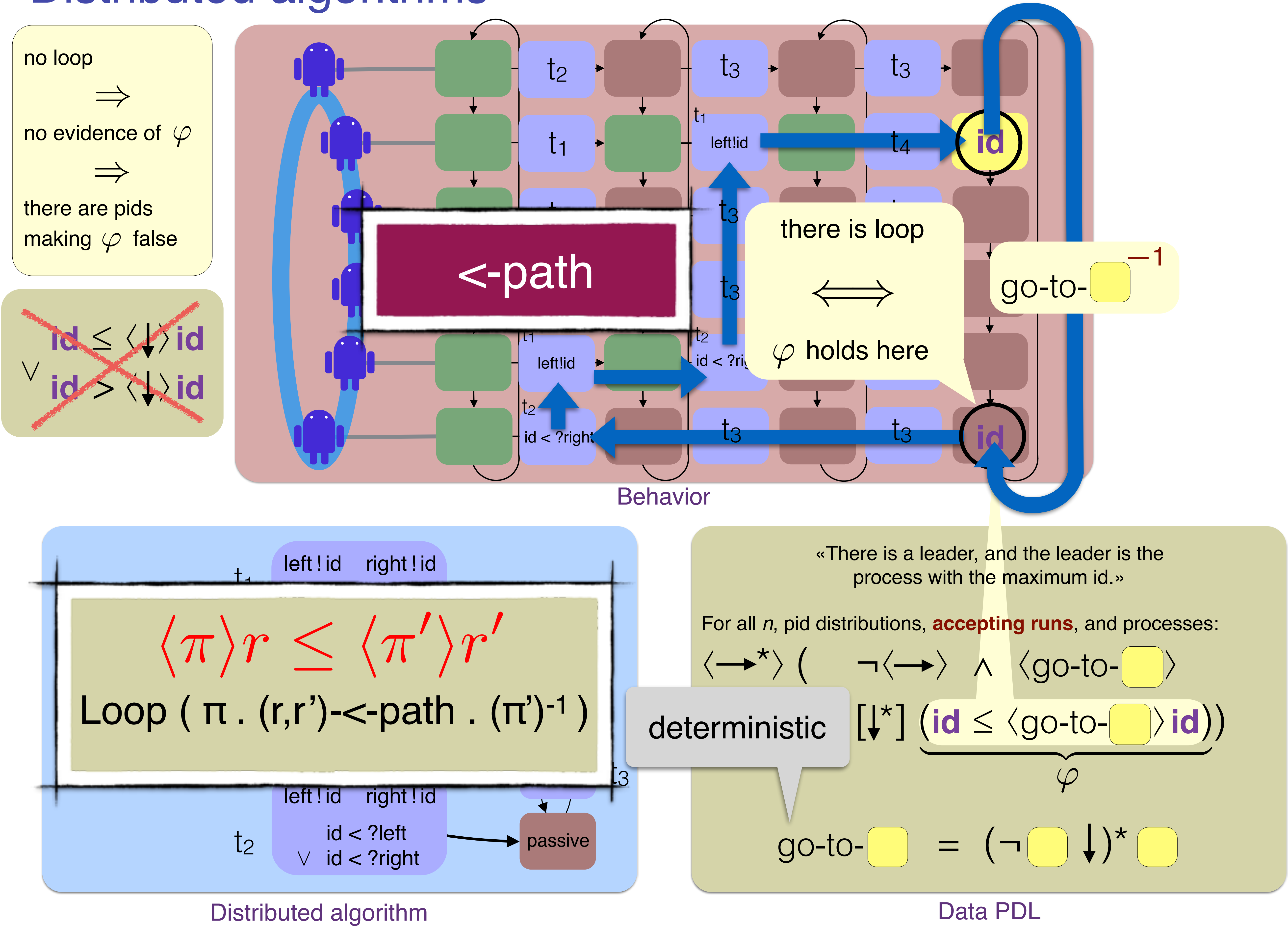


Data PDL

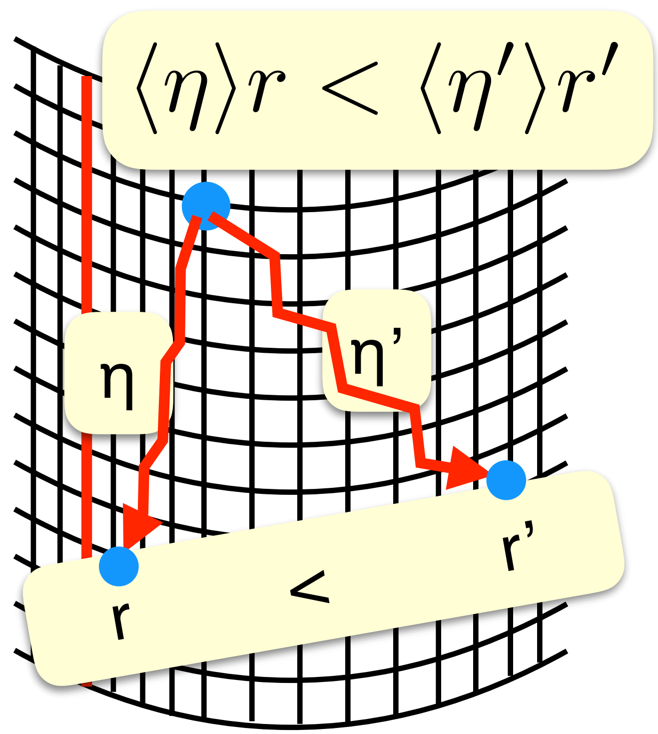
Distributed algorithms



Distributed algorithms



Specifications Data PDL



compare values at different nodes

$$\Phi, \Phi' ::= \mathsf{A}\,\phi \mid \Phi \wedge \Phi'$$

$$\phi, \phi' ::= \varphi \mid \phi \wedge \phi' \mid \varphi \vee \phi \mid [\pi]\phi \mid \langle\eta\rangle r < \langle\eta'\rangle r' \mid \langle\eta\rangle r \leq \langle\eta'\rangle r'$$

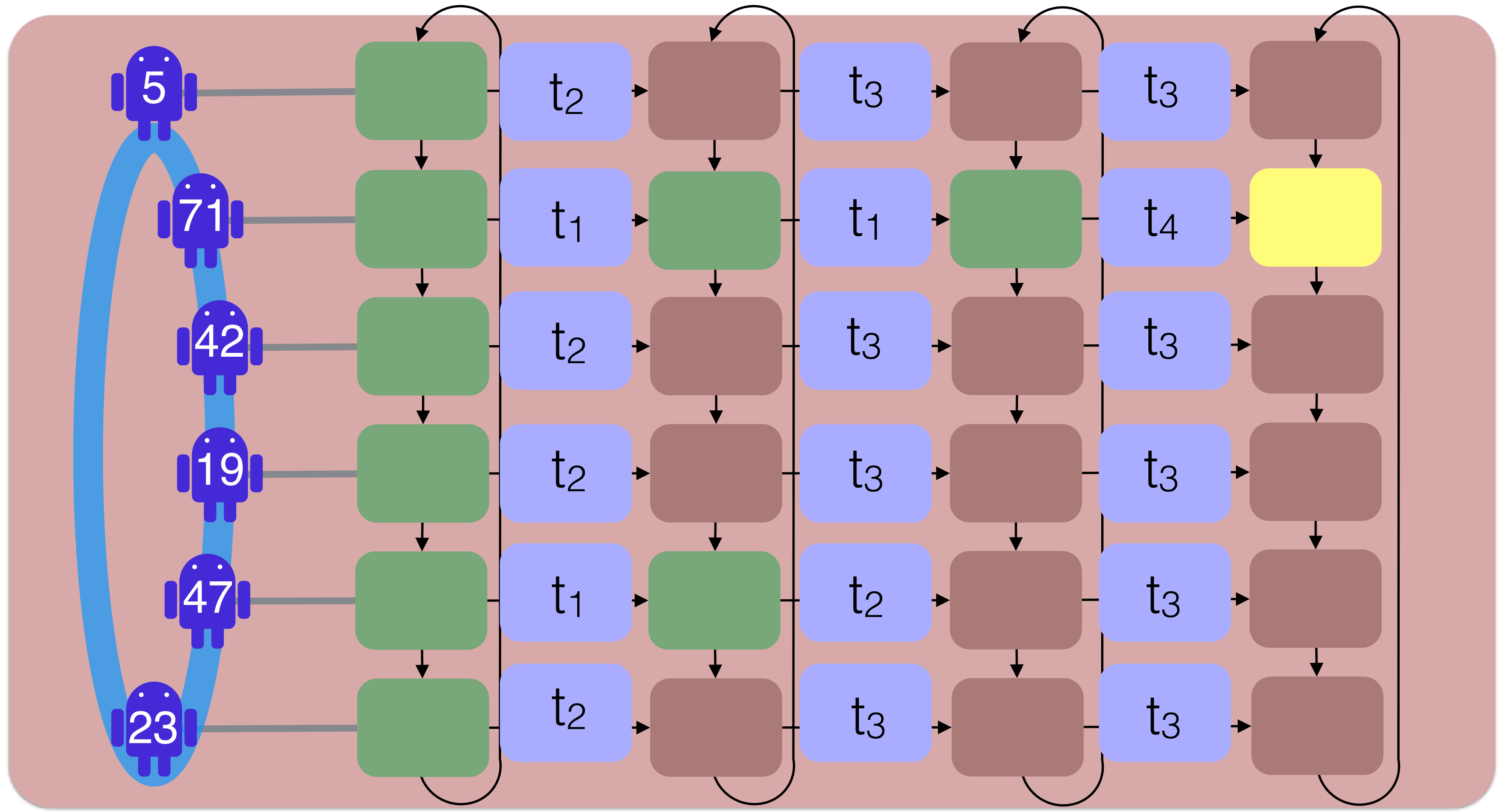
$$\varphi, \varphi' ::= \ddagger \mid p \mid \neg\varphi \mid \varphi \wedge \varphi' \mid \langle\pi\rangle\varphi \mid \langle\pi\rangle r = \langle\pi'\rangle r' \mid \langle\pi\rangle r \neq \langle\pi'\rangle r'$$

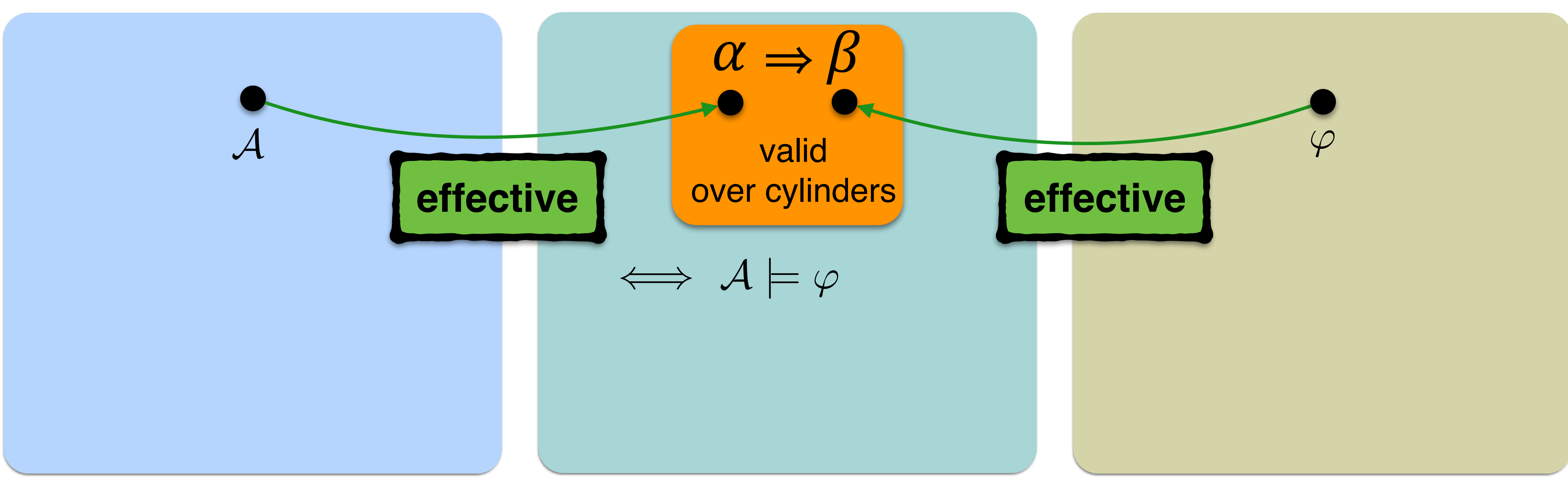
$$\pi, \pi' ::= \{\varphi\}? \mid \rightarrow \mid \downarrow \mid \pi^{-1} \mid \pi + \pi' \mid \pi \cdot \pi' \mid \pi^*$$

$$\eta, \eta' ::= \{\varphi\}? \mid \leftarrow \mid \rightarrow \mid \downarrow \mid \uparrow \mid \eta \cdot \eta' \mid \mathsf{F}^{\eta}_{\varphi}$$

deterministic paths

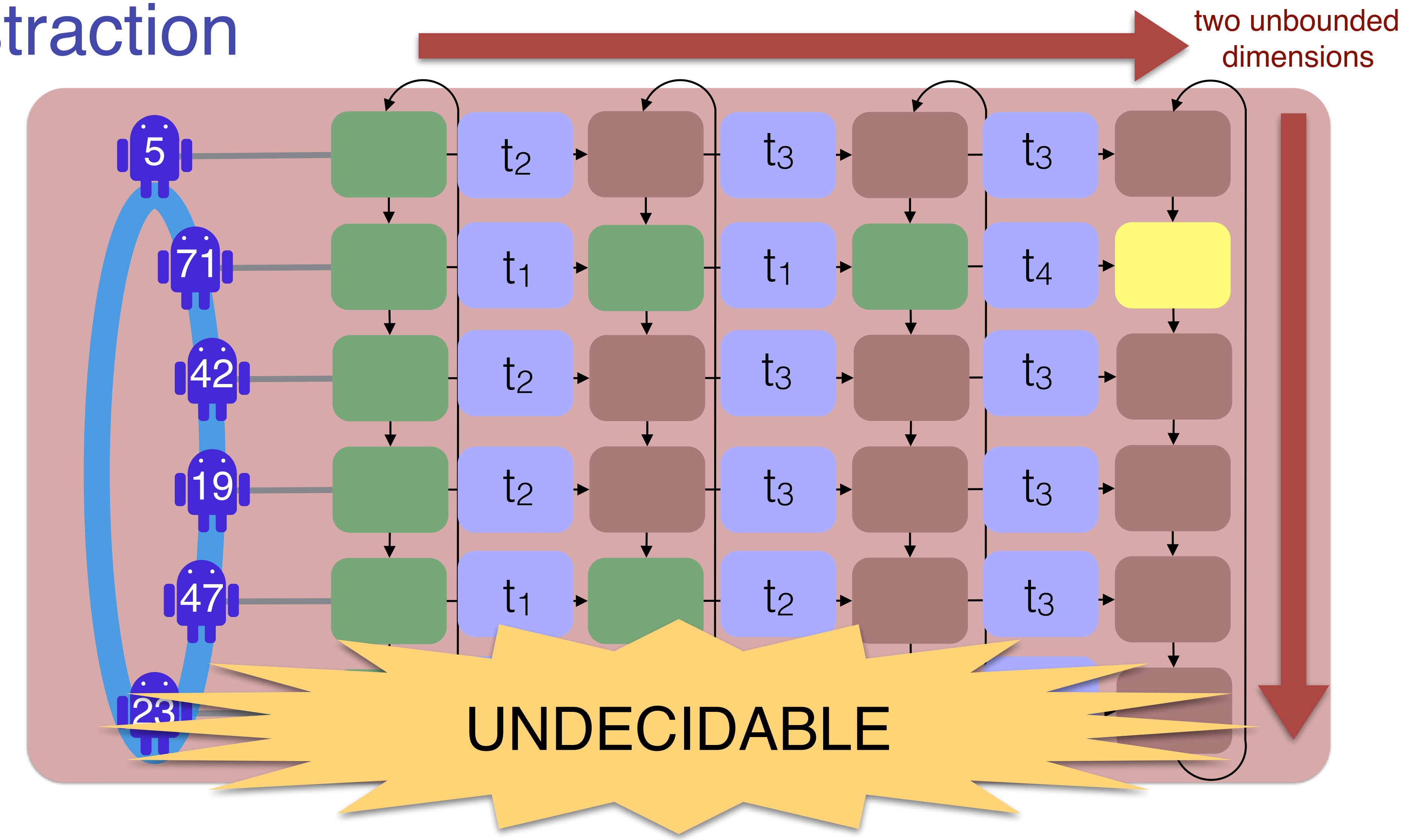
Data abstraction

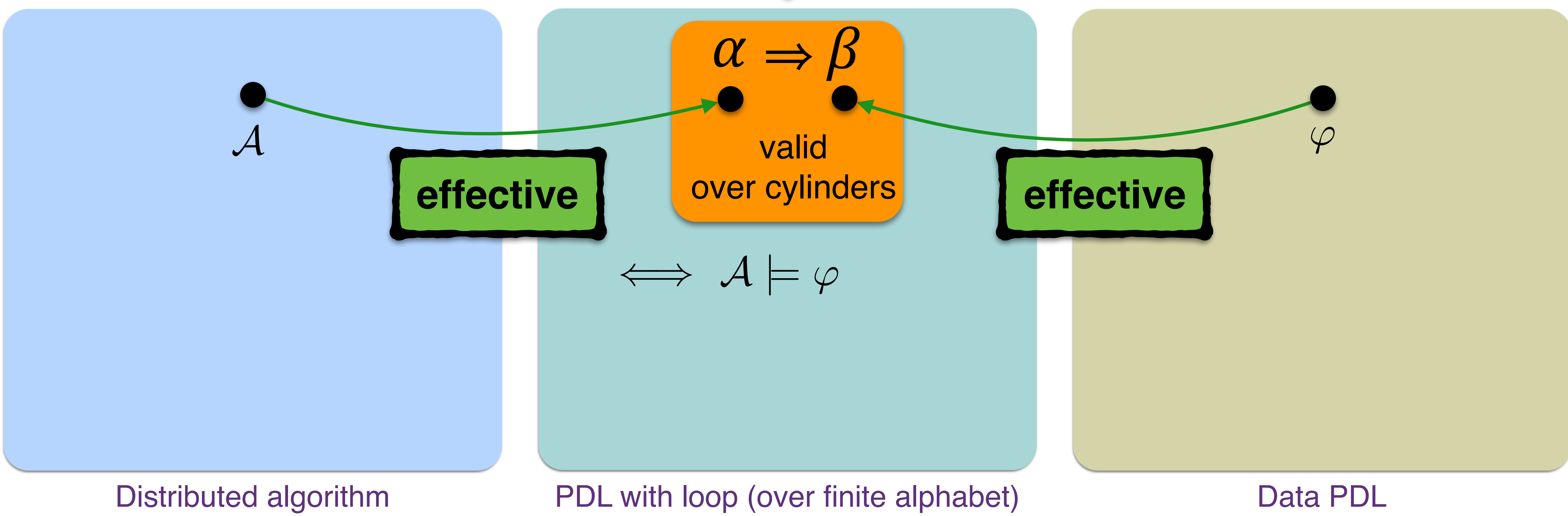




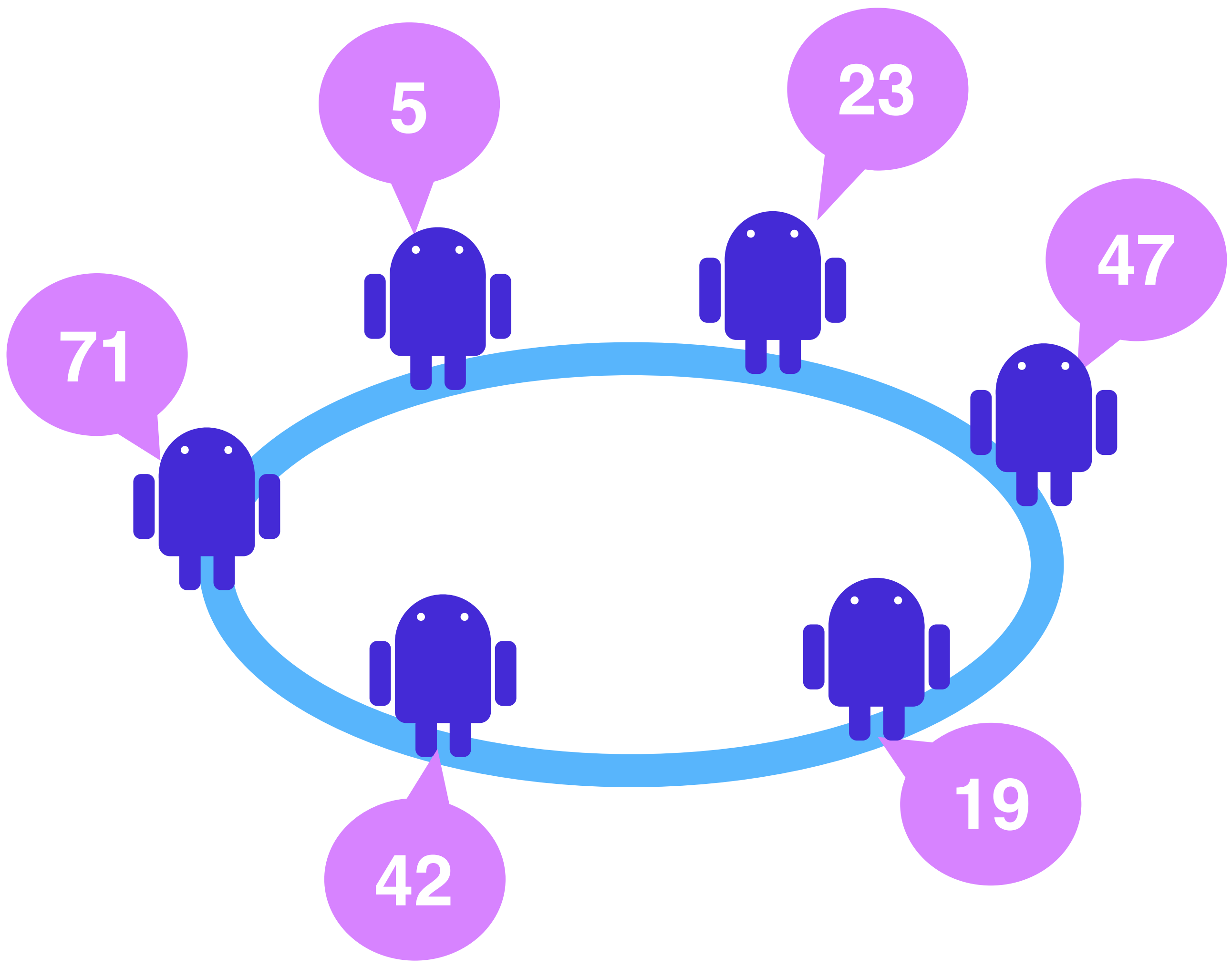
Distributed algorithm | PDL with loop (over finite alphabet) | Data PDL

Data abstraction

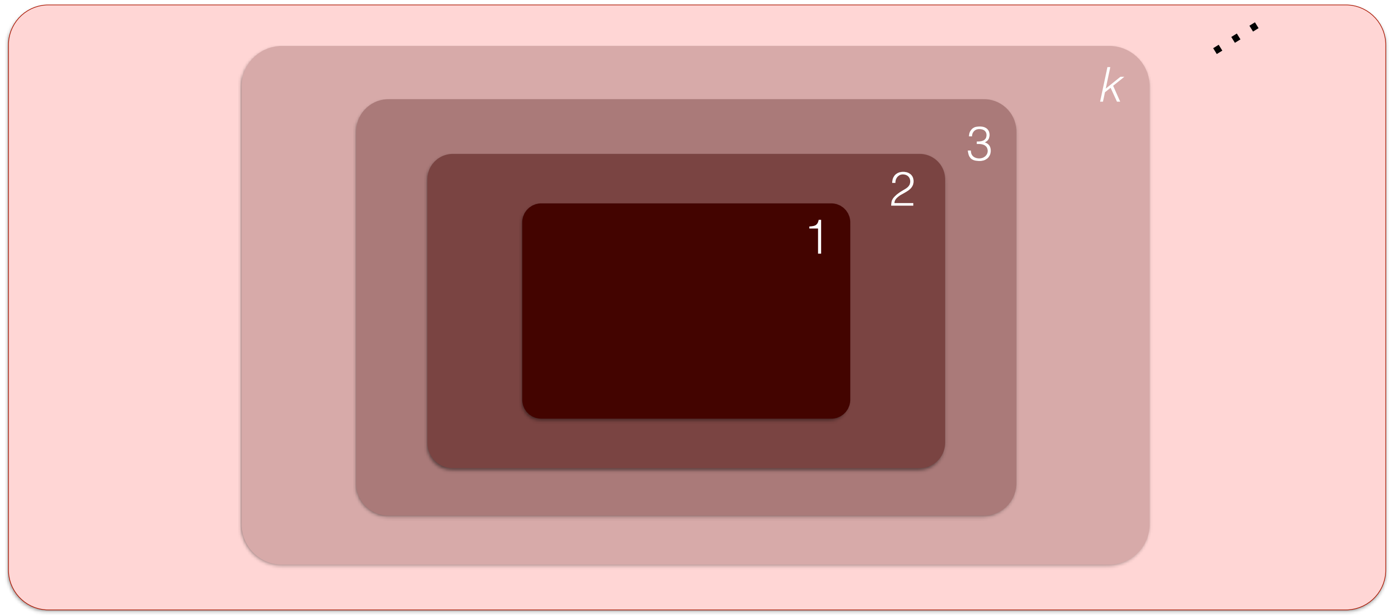




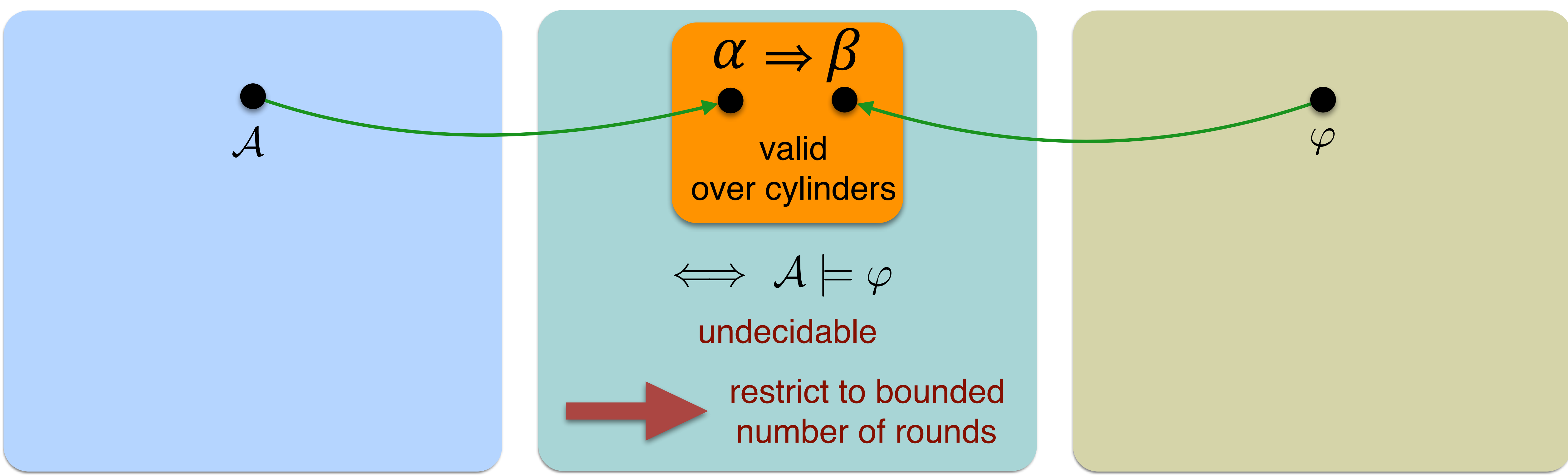
Model Checking 2



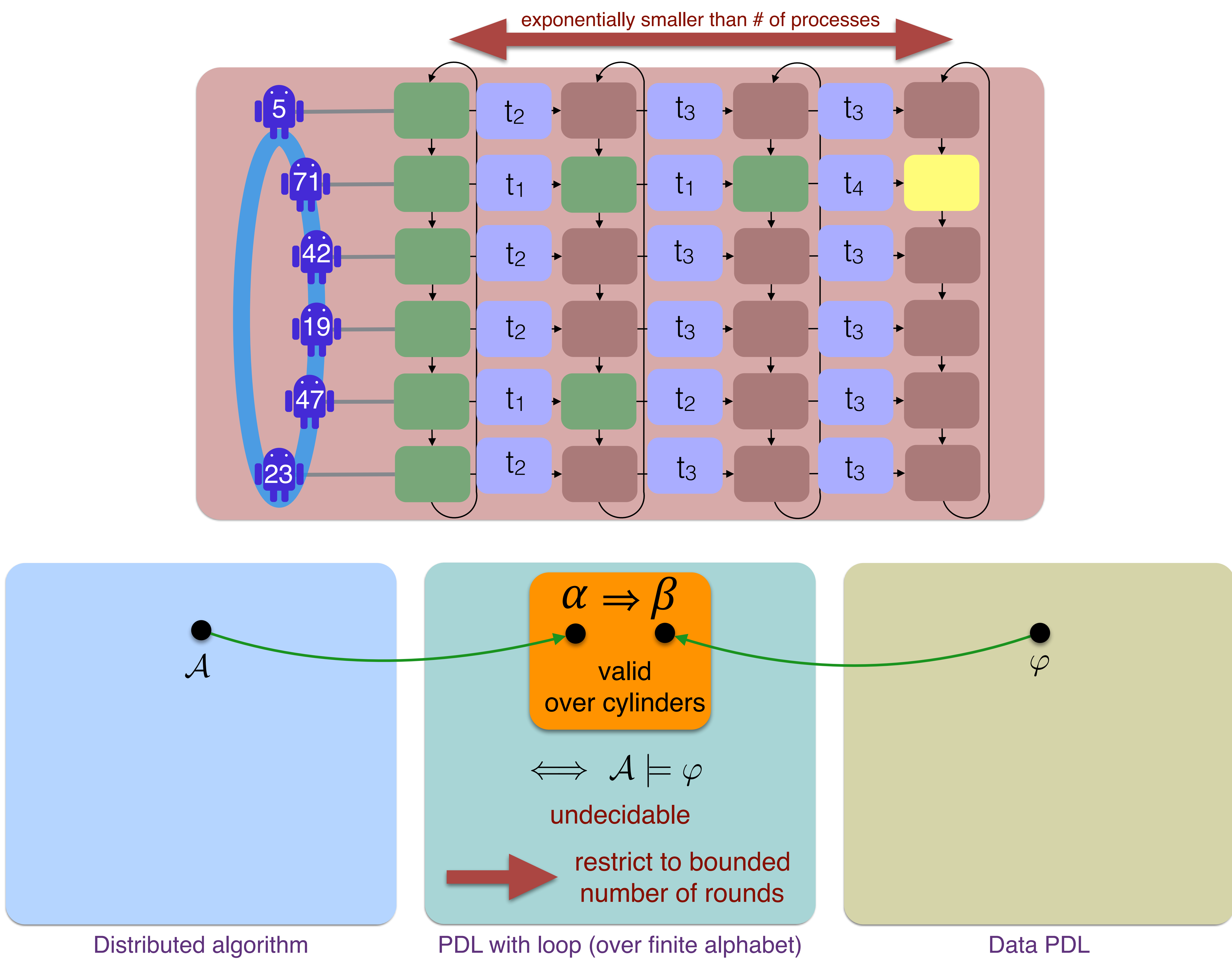
Under approximate verification

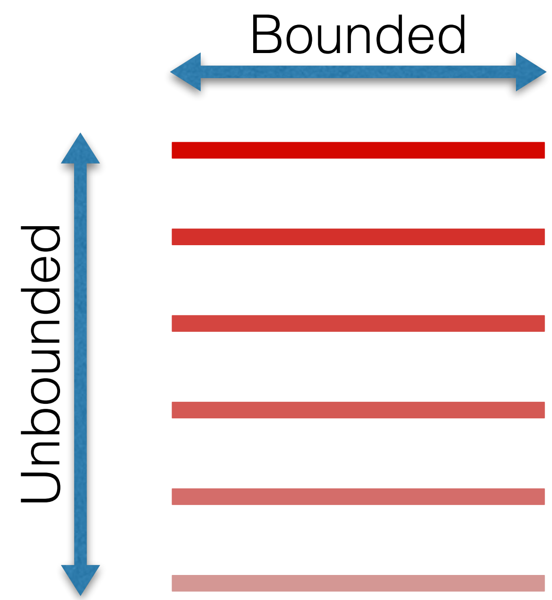


Behavior



Distributed algorithm | PDL with loop (over finite alphabet) | Data PDL

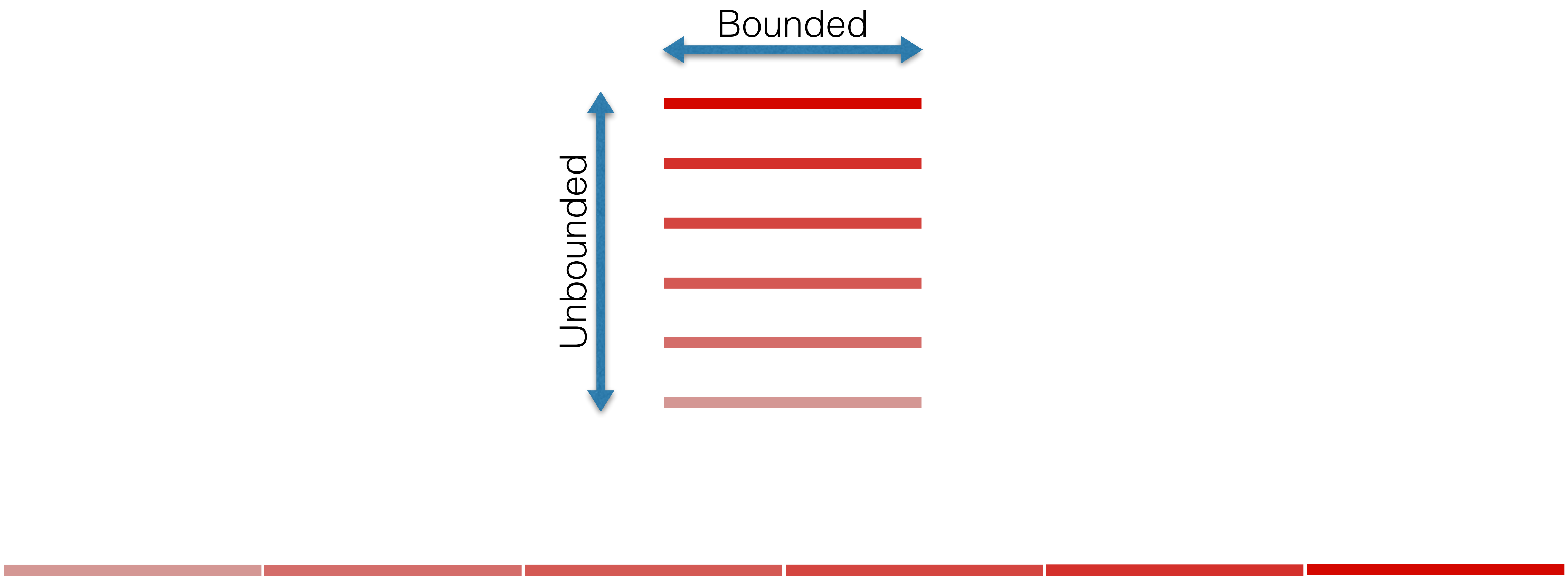

exponentially smaller than # of processes
5
71
42
19
47
23
$\alpha \Rightarrow \beta$
$\mathcal{A}$
φ
valid
over cylinders
$\iff \mathcal{A} \models \varphi$
undecidable
restrict to bounded
number of rounds
Distributed algorithm
PDL with loop (over finite alphabet)
Data PDL



PDL with loop over bounded cylinders

⇨

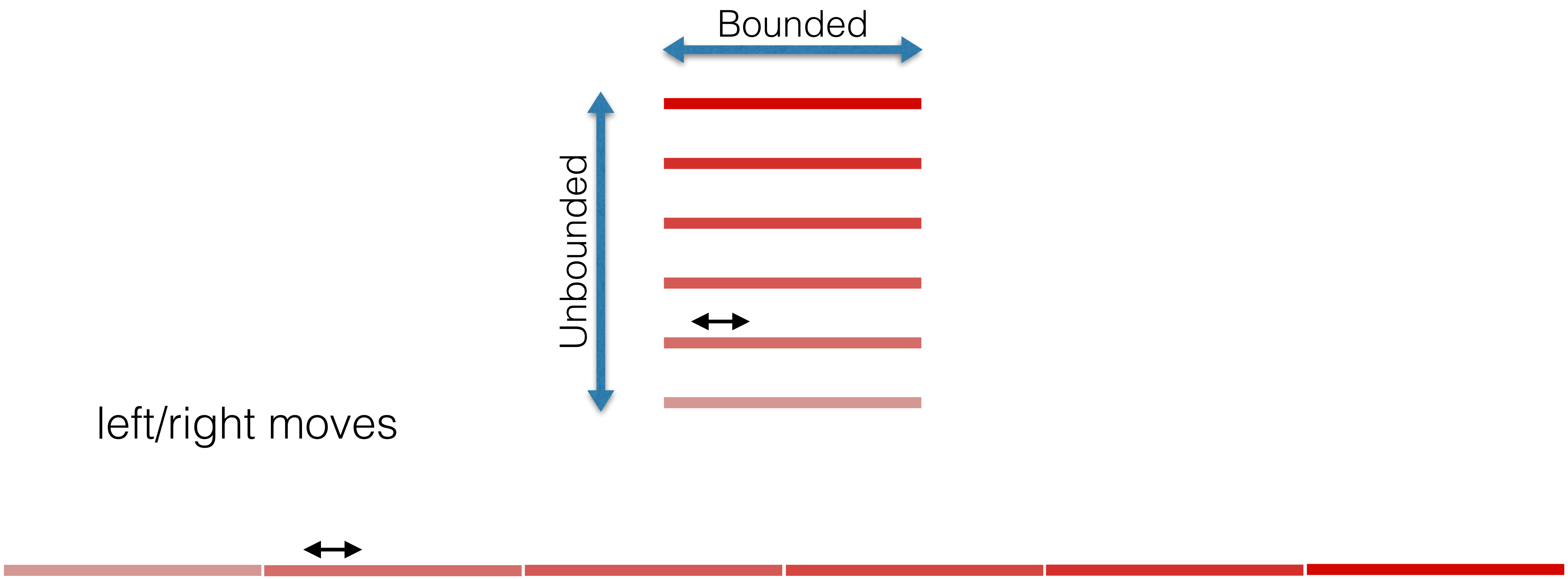
PDL with loop over words



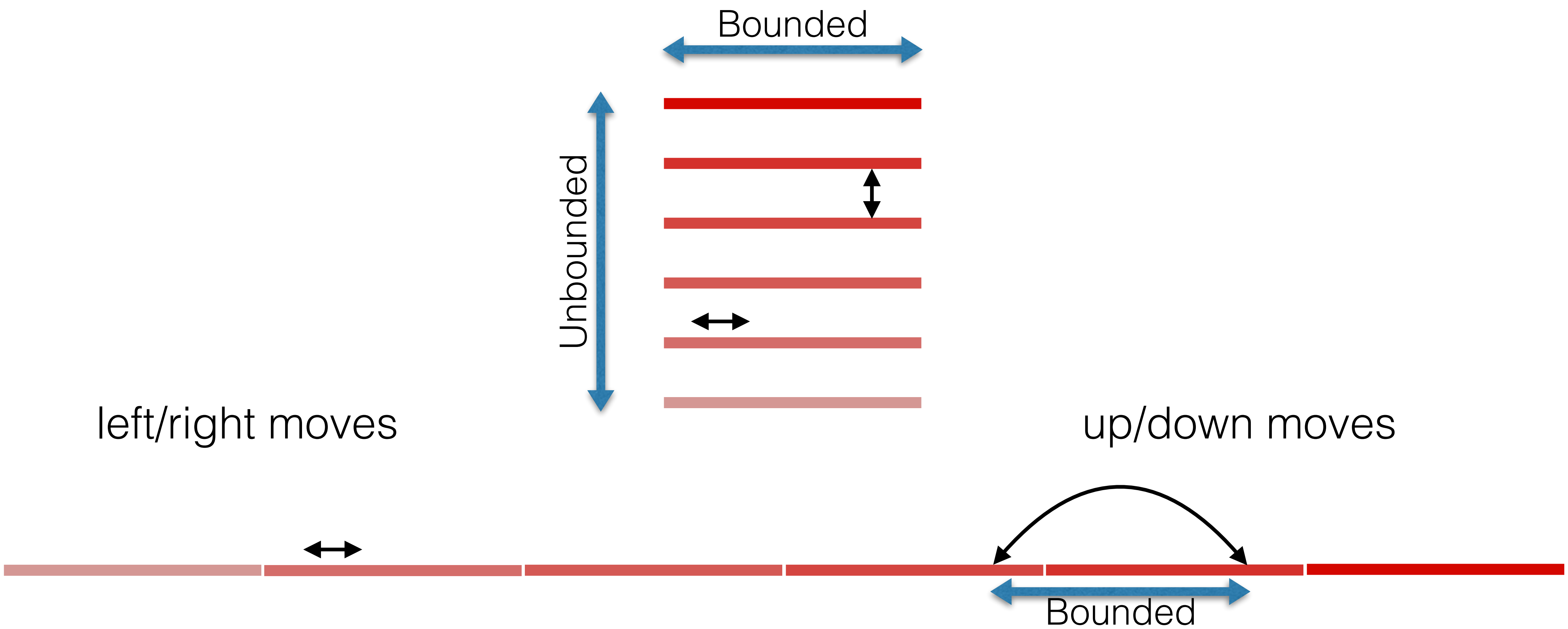
PDL with loop over bounded cylinders

⇨

PDL with loop over words



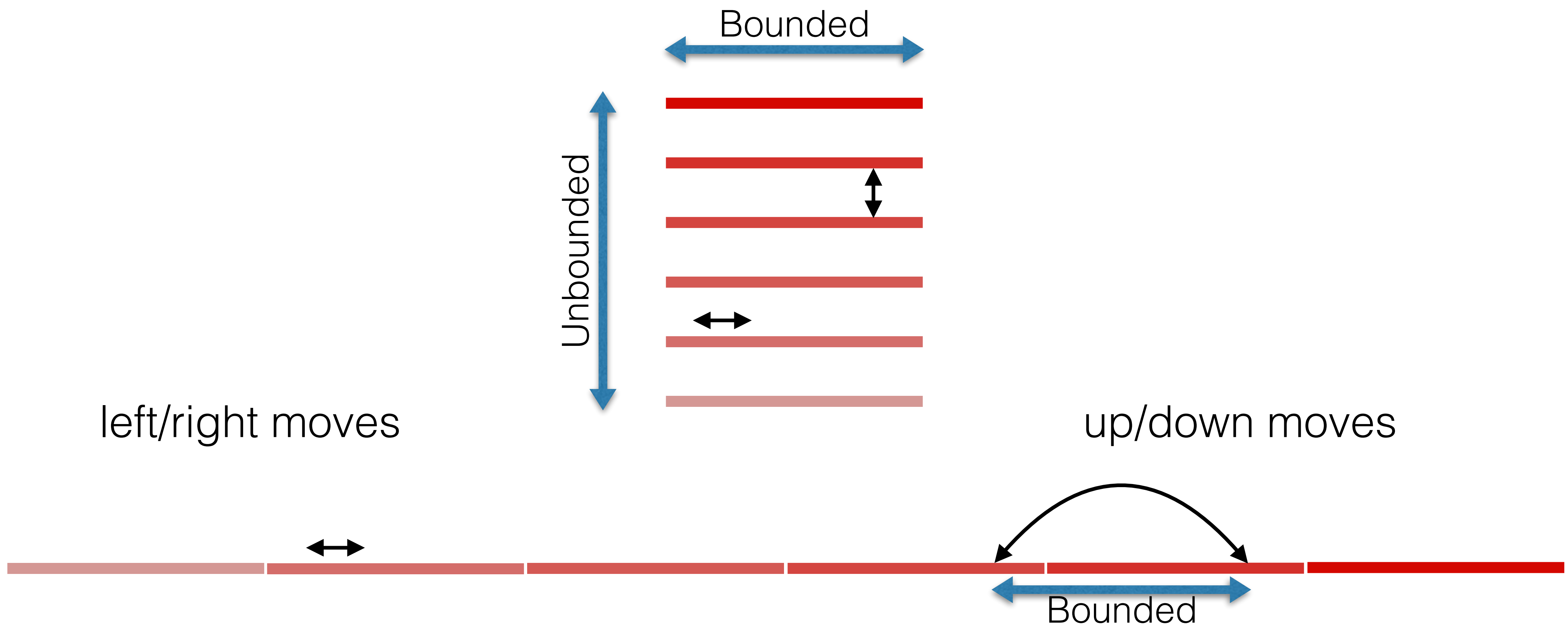
PDL with loop over bounded cylinders
⇨
PDL with loop over words



PDL with loop over bounded cylinders

⇨

PDL with loop over words



PDL with loop over bounded cylinders

⇨

PDL with loop over words

⇨

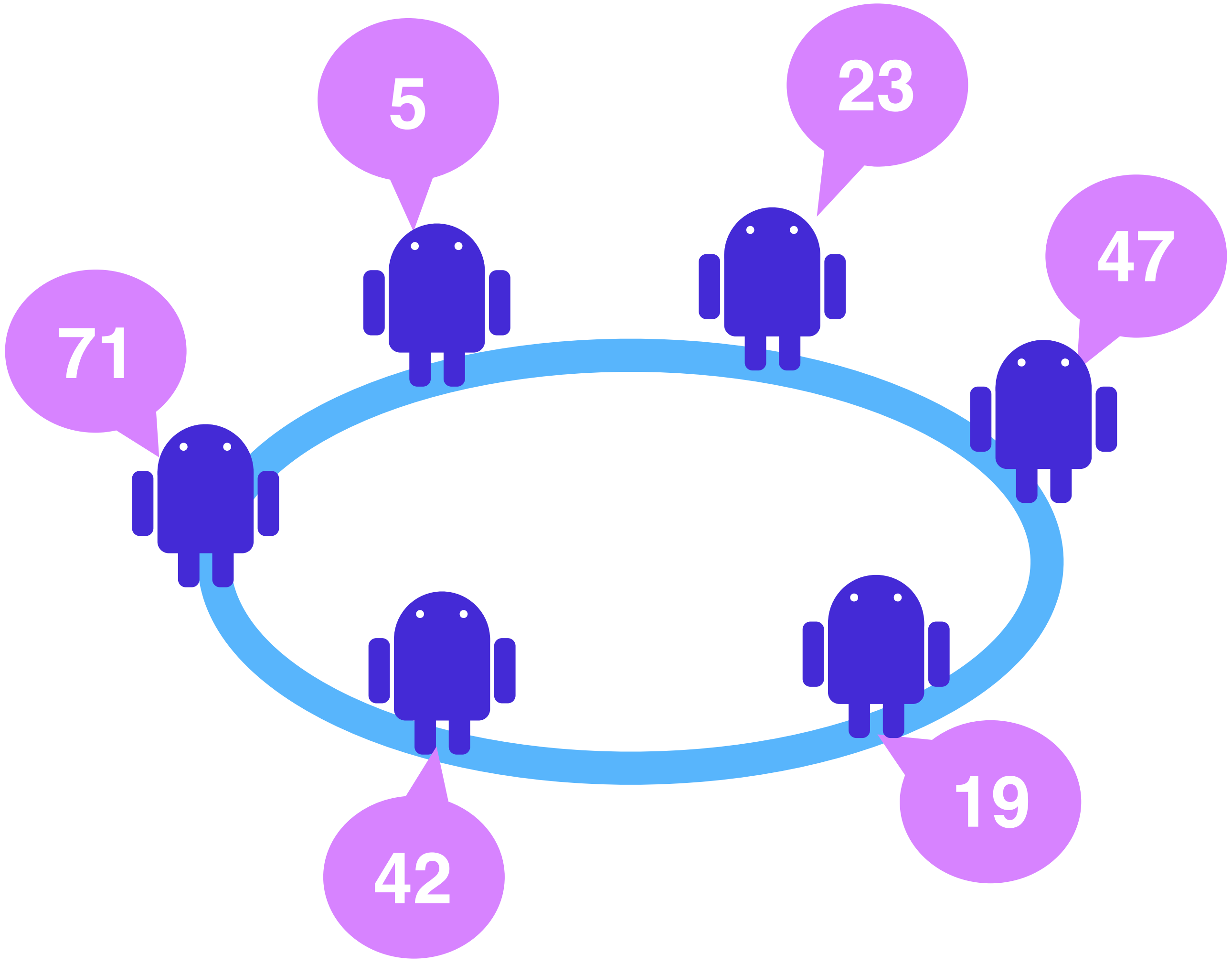
Alternating 2-way Automata

⇨

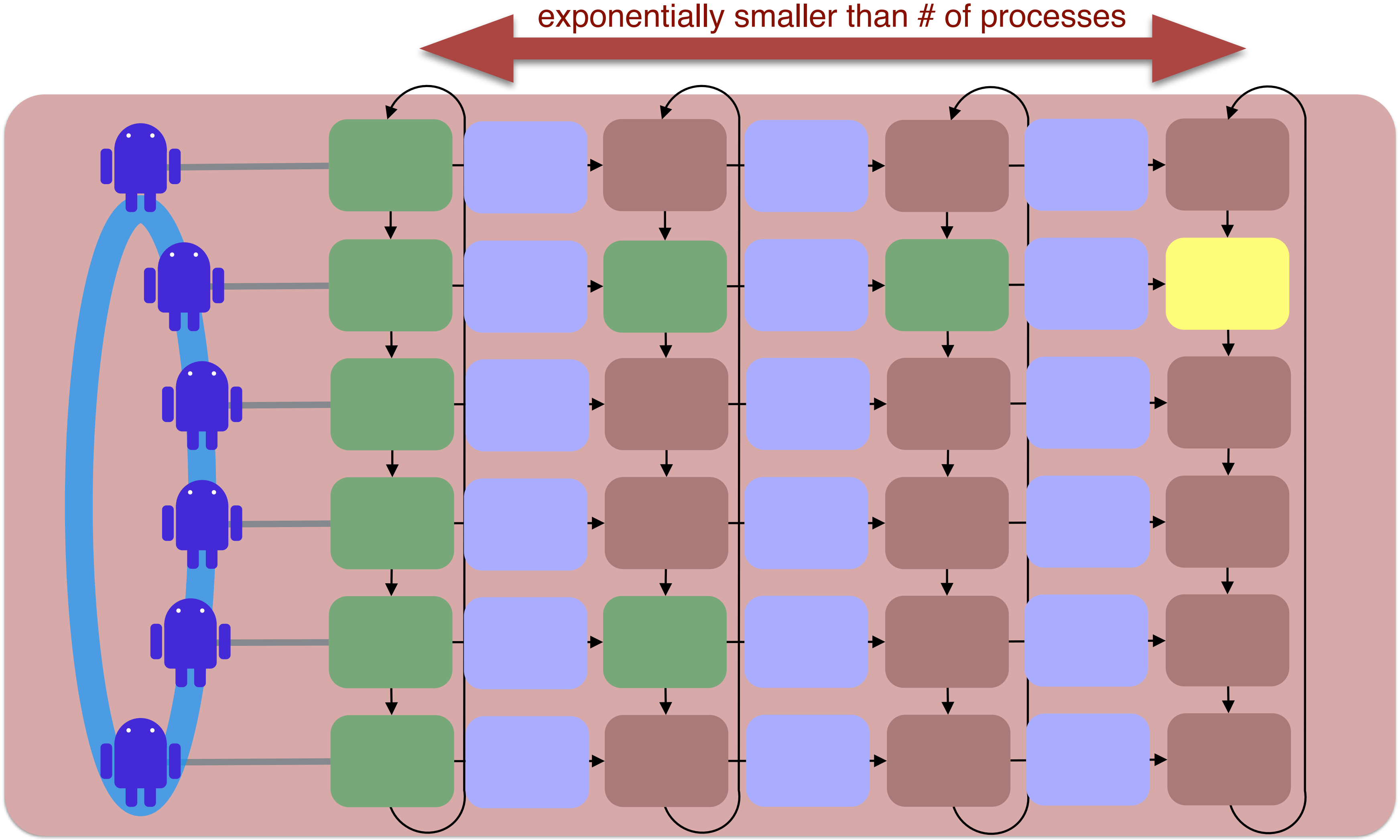
PSPACE

[Göller-Lohrey-Lutz ’08] [Serre ’08]

Summary & Conclusion



Summary



Theorem (Aiswarya-Bollig-Gastin; CONCUR '15).

Round-bounded model checking distributed algorithms* against Data PDL is PSPACE-complete**.

* with registers, register guards, and register updates (no arithmetic)

** unary encoding of # of rounds

Conclusion

- What is the right temporal logic? Use generic Data PDL.
- How to deal with data? Use symbolic technique.
- How to deal with undecidability? Under-approximation.

Future work …

- Other operations? (increment only, decrement only, …)
- Other topologies?
- Other restrictions? (bounded tree-width, …)
- Other hypotheses on DA?


Thank you!