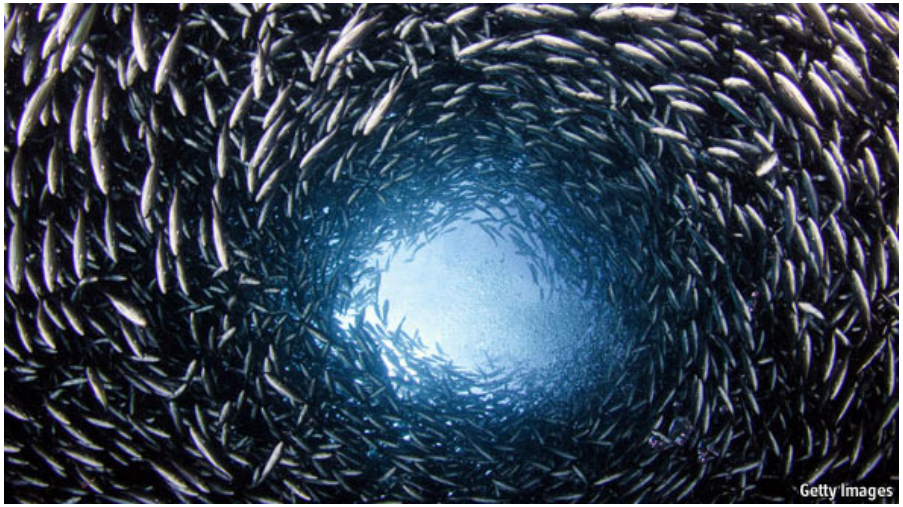
Combining SLiVER with CADP to Analyze Multi-agent Systems

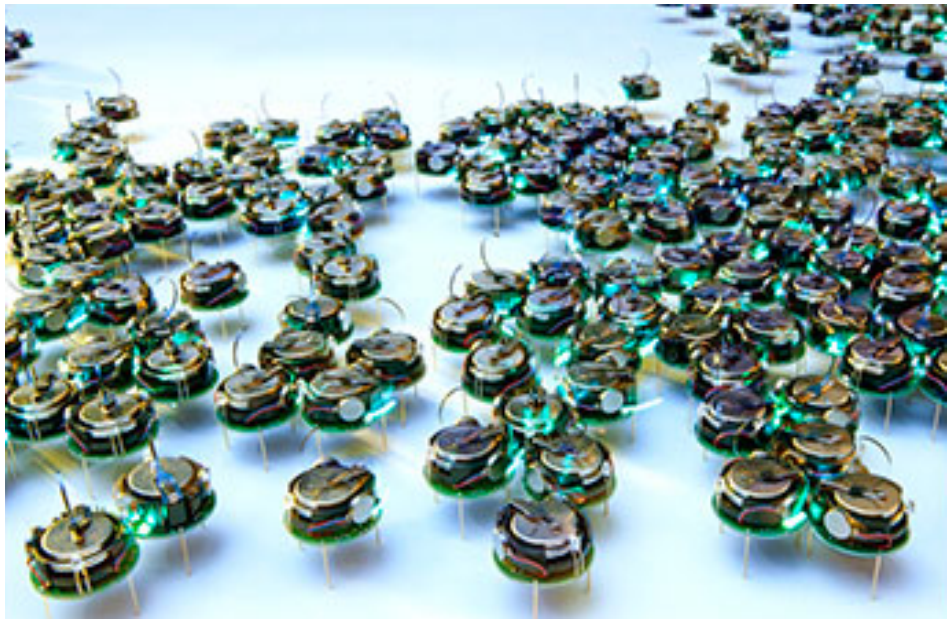
Luca Di Stefano[1,2] Frédéric Lang[3] Wendelin Serwe[3]

[1] Gran Sasso Science Institute (GSSI), L'Aquila, Italy
[2] IMT School of Advanced Studies, Lucca, Italy
[3] Univ. Grenoble Alpes, Inria, CNRS, Grenoble, France

18th June 2020
COORDINATION

Systems of agents





Challenges and desiderata

Challenges:

- Large number of agents
- Nondeterminism
- Asynchronous interaction
- Emergence of collective behavior

Desiderata:

- Languages to handle/hide the complexity
- Tools for automated analysis

How?

- Concepts from the MAS community
- Methodologies from the FM/verification communities

Challenges and desiderata

Challenges:

- Large number of agents
- Nondeterminism
- Asynchronous interaction
- Emergence of collective behavior

Desiderata:

- **Languages** to handle/hide the complexity
- **Tools** for automated analysis

How?

- **Concepts** from the MAS community
- **Methodologies** from the FM/verification communities

Challenges and desiderata

Challenges:

- Large number of agents
- Nondeterminism
- Asynchronous interaction
- Emergence of collective behavior

Desiderata:

- **Languages** to handle/hide the complexity
- **Tools** for automated analysis

How?

- **Concepts** from the MAS community
- **Methodologies** from the FM/verification communities

Language: LAbS

A Language with Attribute-based Stigmergies[1]

Small DSL to formally describe multi-agent systems

- Stimergic variables: indirect propagation of knowledge
  - Values of stigmergic variables are timestamped
  - Agents compare timestamps and agree on the most recent value
  - This happens transparently and asynchronously
- User can restrict stigmergic interaction with attribute-based predicates
  - *true*: no restriction
  - $var_1 = var_2$: sender and receiver must have the same value for *var*
- Also supports "classic" shared variables

[1] R. De Nicola, L. Di Stefano, and O. Inverso, *"Multi-agent systems with virtual stigmergy,"* Sci. Comput. Program. 187, 2020.

Language: LAbS

A Language with Attribute-based Stigmergies[1]

Small DSL to formally describe multi-agent systems

- Stimergic variables: indirect propagation of knowledge
  - Values of stigmergic variables are timestamped
  - Agents compare timestamps and agree on the most recent value
  - This happens transparently and asynchronously
- User can restrict stigmergic interaction with attribute-based predicates
  - *true*: no restriction
  - $var_1 = var_2$: sender and receiver must have the same value for *var*
- Also supports "classic" shared variables

[1] R. De Nicola, L. Di Stefano, and O. Inverso, *"Multi-agent systems with virtual stigmergy,"* Sci. Comput. Program. 187, 2020.

Language: LAbS

A Language with Attribute-based Stigmergies[1]

Small DSL to formally describe multi-agent systems

- Stimergic variables: indirect propagation of knowledge
  - Values of stigmergic variables are timestamped
  - Agents compare timestamps and agree on the most recent value
  - This happens transparently and asynchronously
- User can restrict stigmergic interaction with attribute-based predicates
  - *true*: no restriction
  - $var_1 = var_2$: sender and receiver must have the same value for *var*
- Also supports "classic" shared variables

[1] R. De Nicola, L. Di Stefano, and O. Inverso, *"Multi-agent systems with virtual stigmergy,"* Sci. Comput. Program. 187, 2020.

Language: LAbS

A Language with Attribute-based Stigmergies[1]

Small DSL to formally describe multi-agent systems

- Stimergic variables: indirect propagation of knowledge
  - Values of stigmergic variables are timestamped
  - Agents compare timestamps and agree on the most recent value
  - This happens transparently and asynchronously
- User can restrict stigmergic interaction with attribute-based predicates
  - *true*: no restriction
  - $var_1 = var_2$: sender and receiver must have the same value for *var*
- Also supports "classic" shared variables

[1] R. De Nicola, L. Di Stefano, and O. Inverso, *"Multi-agent systems with virtual stigmergy,"* Sci. Comput. Program. 187, 2020.

Stigmergy-based Leader Election (1)

N agents; each has a (unique) *id* between 0 and $N-1$
One stigmergic variable, *leader*, initially set to N
Each agent repeatedly acts as follows:

Is *leader* $>$ (my own *id*)?
YES $\Rightarrow$ update *leader* to my own *id*
NO $\Rightarrow$ do nothing

Agents exchange values of *leader* with each other
Eventually, *leader* should be 0 for all agents

Stigmergy-based Leader Election (1)

N agents; each has a (unique) *id* between 0 and $N - 1$
One stigmergic variable, *leader*, initially set to N
Each agent repeatedly acts as follows:

Is *leader* $>$ (my own *id*)?
YES $\Rightarrow$ update *leader* to my own *id*
NO $\Rightarrow$ do nothing

Agents exchange values of *leader* with each other
Eventually, *leader* should be 0 for all agents

Stigmergy-based Leader Election (1)

N agents; each has a (unique) *id* between 0 and $N-1$
One stigmergic variable, *leader*, initially set to N
Each agent repeatedly acts as follows:

Is *leader* $>$ (my own *id*)?
YES $\Rightarrow$ update *leader* to my own *id*
NO $\Rightarrow$ do nothing

Agents exchange values of *leader* with each other
Eventually, *leader* should be 0 for all agents

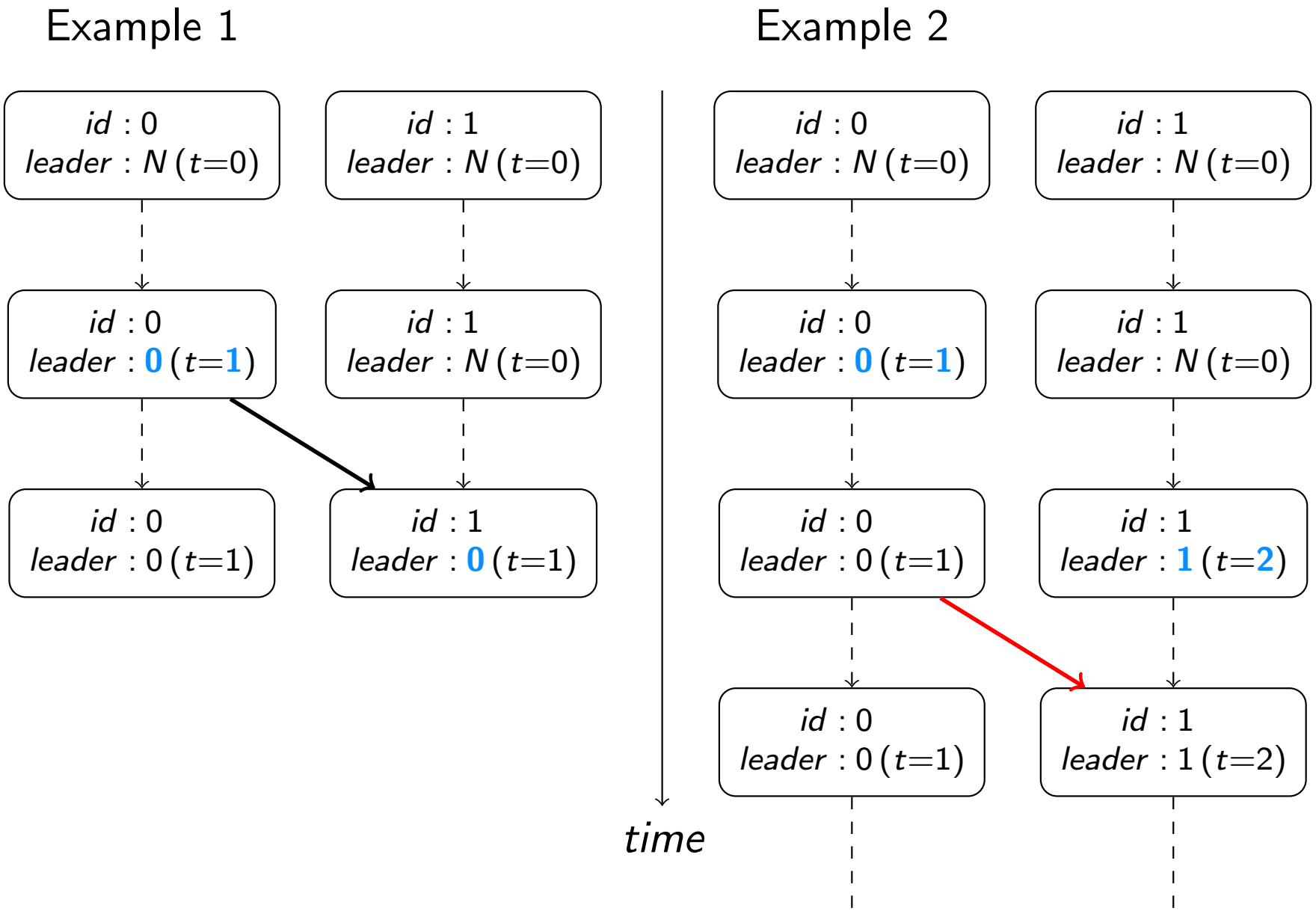
Stigmergy-based Leader Election (1)

N agents; each has a (unique) *id* between 0 and $N - 1$
One stigmergic variable, *leader*, initially set to N
Each agent repeatedly acts as follows:

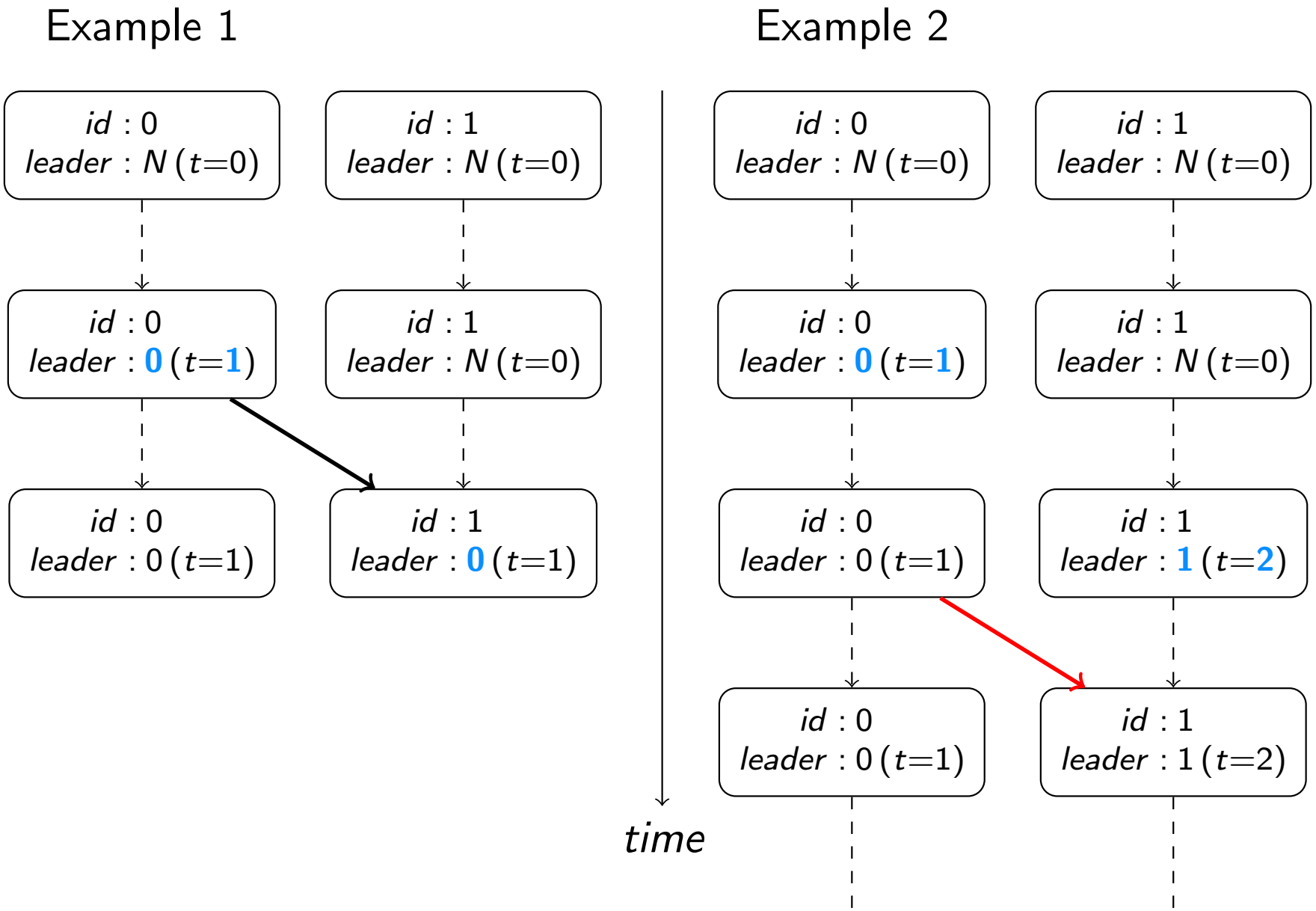
Is *leader* $>$ (my own *id*)?
YES $\Rightarrow$ update *leader* to my own *id*
NO $\Rightarrow$ do nothing

Agents exchange values of *leader* with each other
Eventually, *leader* should be 0 for all agents

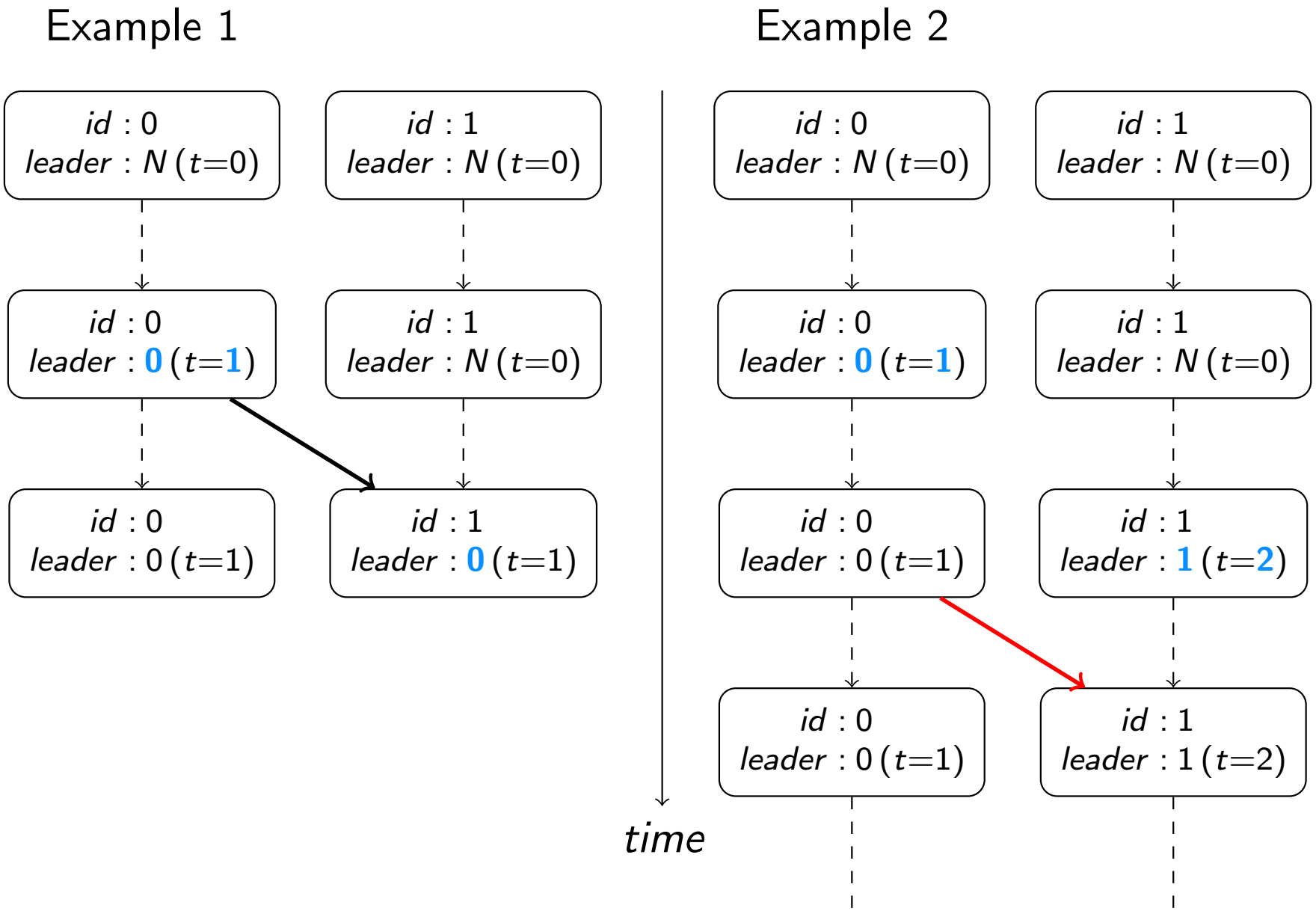
Stigmergy-based Leader Election (2)



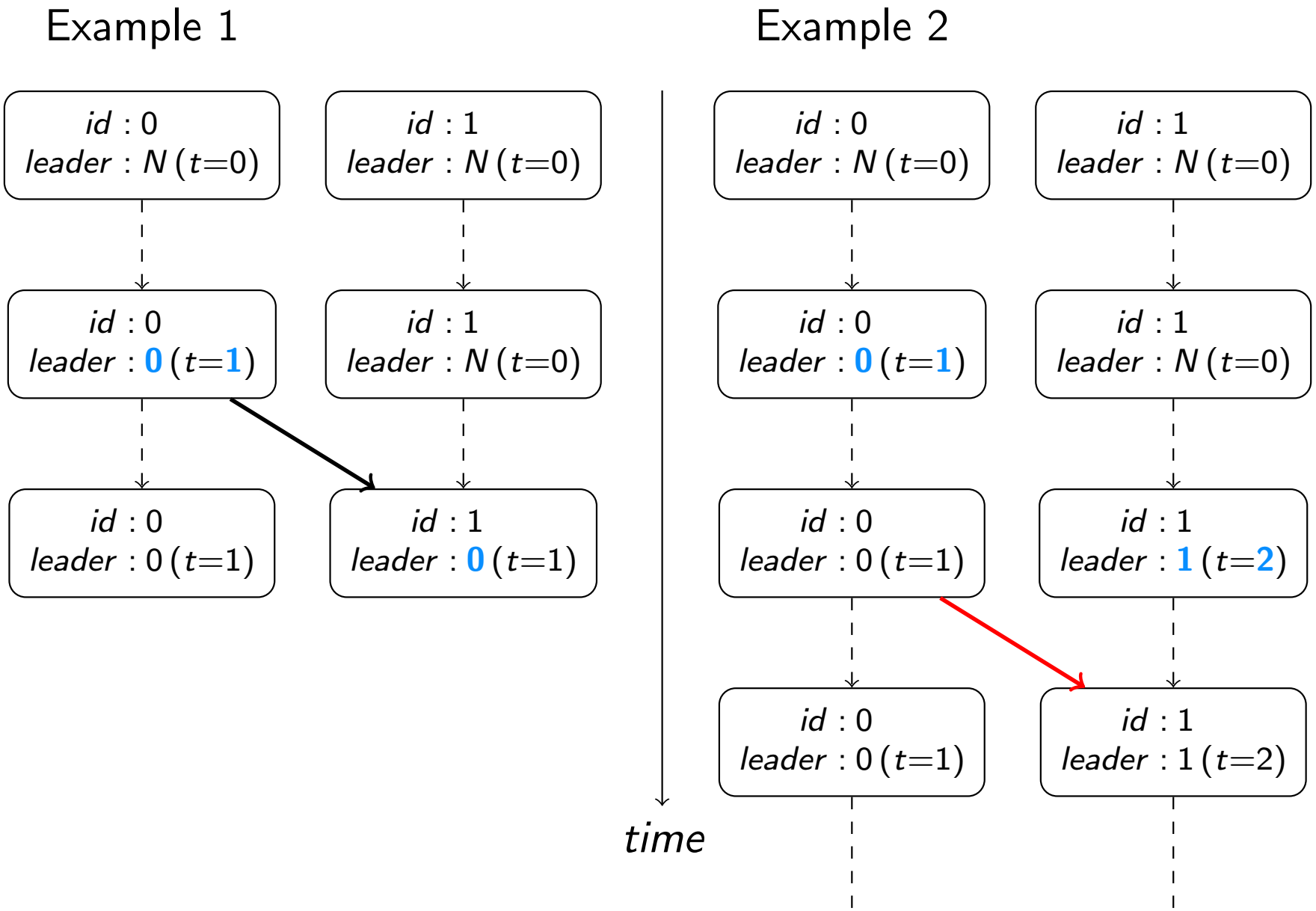
Stigmergy-based Leader Election (2)



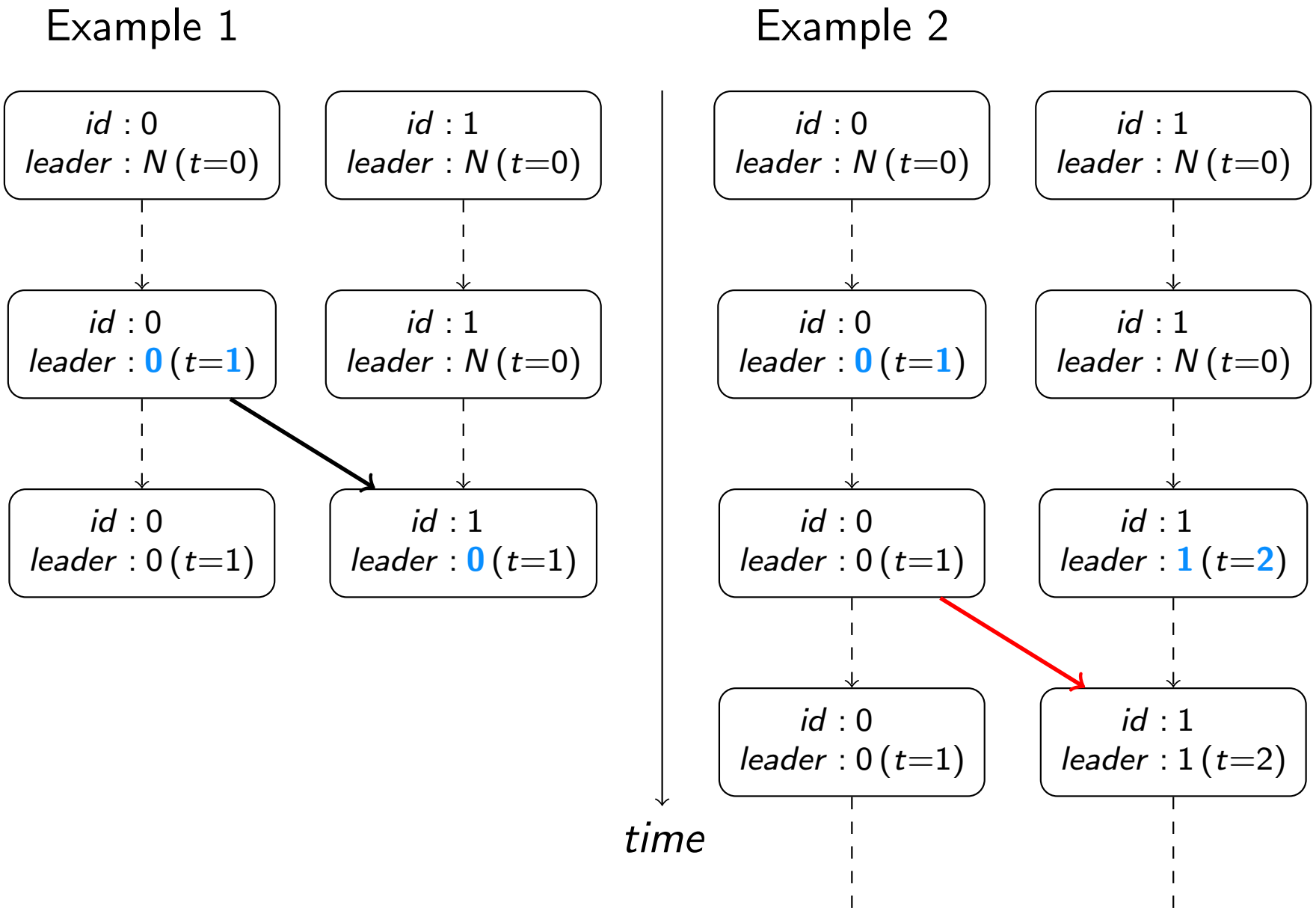
Stigmergy-based Leader Election (2)



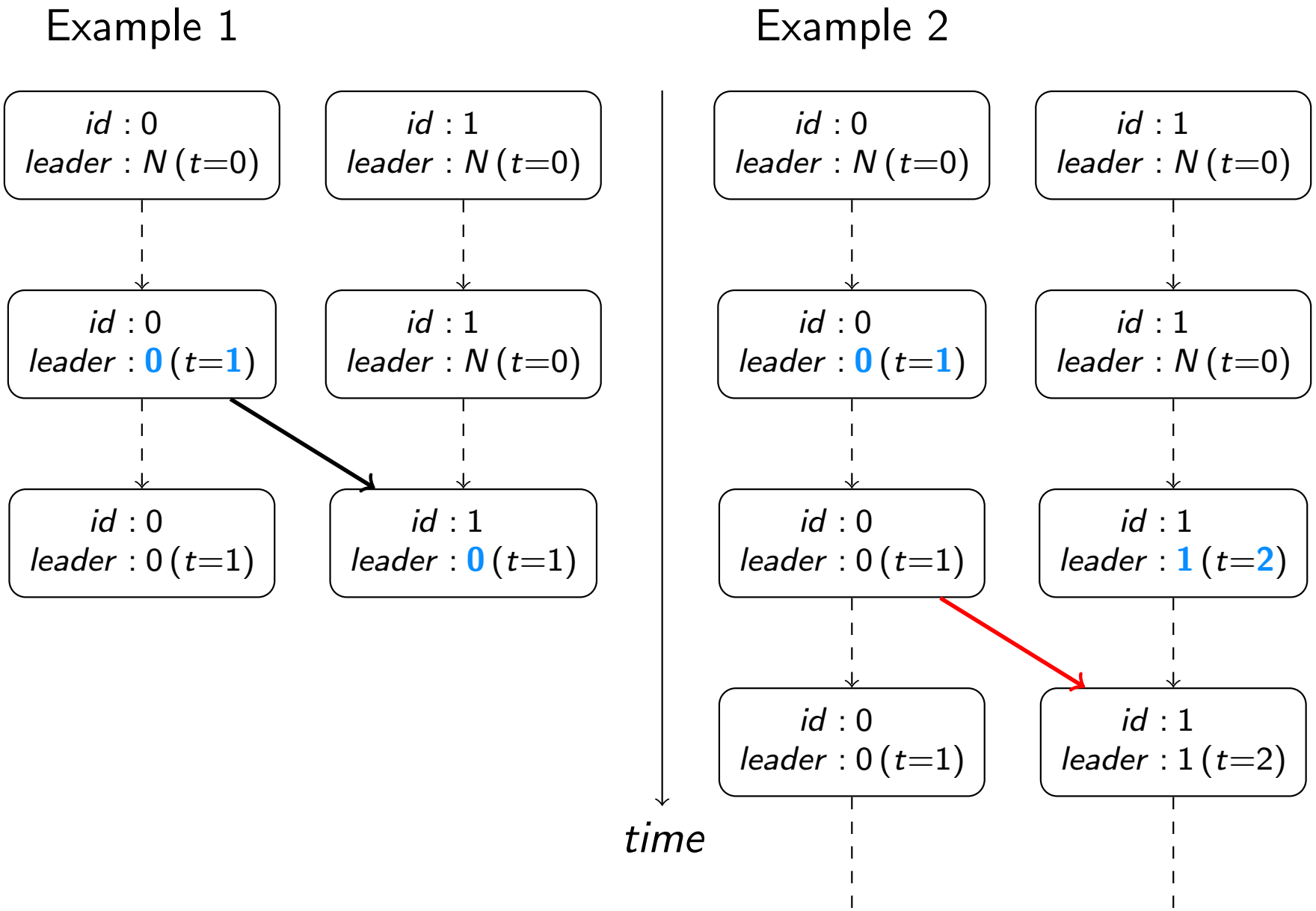
Stigmergy-based Leader Election (2)



Stigmergy-based Leader Election (2)



Stigmergy-based Leader Election (2)



Tools

Developing ad-hoc tools for a new language is a daunting task

- Evolve together with the language
- Keep up with the state of the art in formal verification/program analysis

SLiVER approach

- Encode LAbS specifications in some target language
- Exploit existing tools as analysis back ends
- Modular wrt. target language and analysis back end
- We can just focus on the correctness of the encoding procedure

Encoding from LAbS to the LNT process calculus; analysis with CADP[2]

- Explicit-state model checking
- Simulation

[2] https://cadp.inria.fr/

Tools

Developing ad-hoc tools for a new language is a daunting task

- Evolve together with the language
- Keep up with the state of the art in formal verification/program analysis

SLiVER approach

- Encode LAbS specifications in some target language
- Exploit existing tools as analysis back ends
- Modular wrt. target language and analysis back end
- We can just focus on the correctness of the encoding procedure

Encoding from LAbS to the LNT process calculus; analysis with CADP[2]

- Explicit-state model checking
- Simulation

[2] https://cadp.inria.fr/

Tools

Developing ad-hoc tools for a new language is a daunting task

- Evolve together with the language
- Keep up with the state of the art in formal verification/program analysis

SLiVER approach

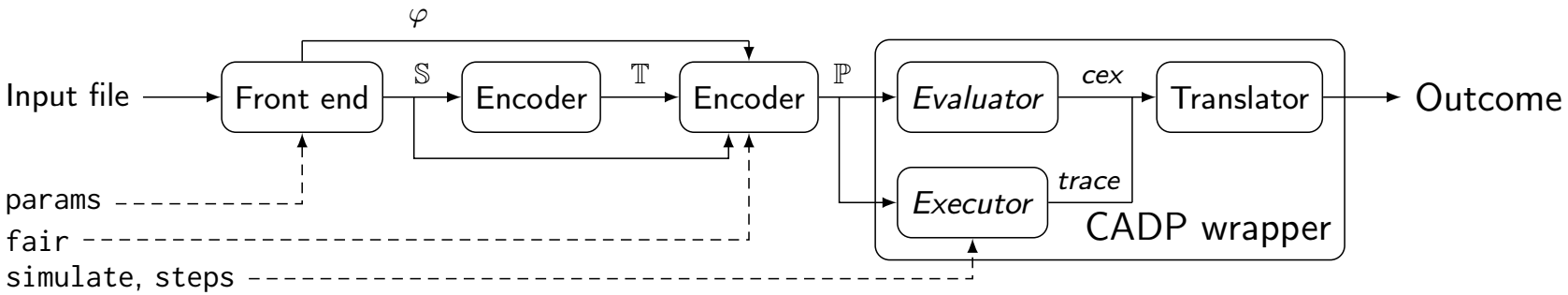
- Encode LAbS specifications in some target language
- Exploit existing tools as analysis back ends
- Modular wrt. target language and analysis back end
- We can just focus on the correctness of the encoding procedure

Encoding from LAbS to the LNT process calculus; analysis with CADP[2]

- Explicit-state model checking
- Simulation

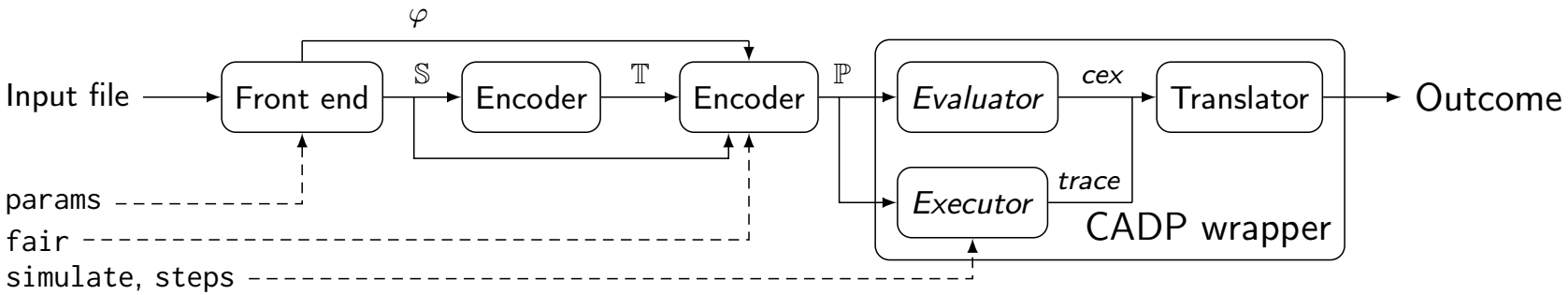
[2] https://cadp.inria.fr/

LNT workflow for SLiVER



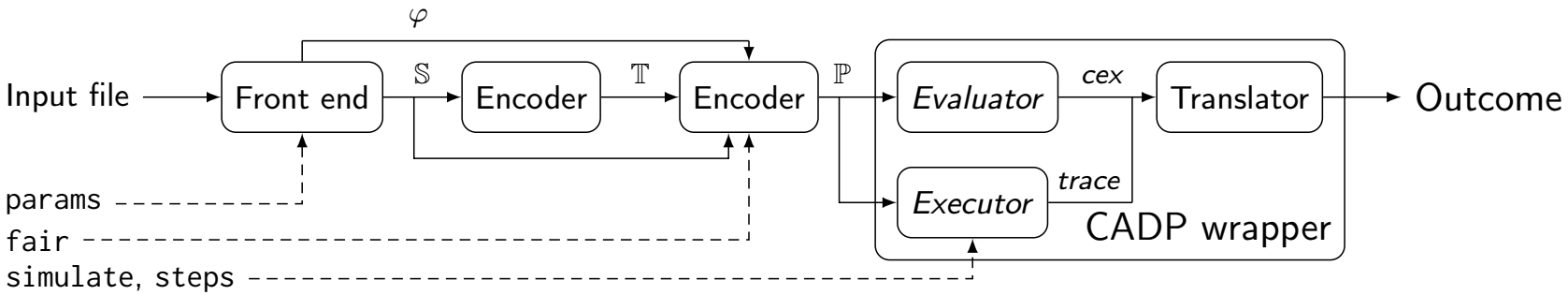
- $\mathbb{S}$: input specifications
- φ: property to verify
- $\mathbb{T}$: intermediate representation (independent of the target language)
- $\mathbb{P}$: LNT program that emulates $\mathbb{S}$

LNT workflow for SLiVER



- $\mathbb{S}$: input specifications
- φ: property to verify
- $\mathbb{T}$: intermediate representation (independent of the target language)
- $\mathbb{P}$: LNT program that emulates $\mathbb{S}$

LNT workflow for SLiVER



- $\mathbb{S}$: input specifications
- φ: property to verify
- $\mathbb{T}$: intermediate representation (independent of the target language)
- $\mathbb{P}$: LNT program that emulates $\mathbb{S}$

Usage: verification mode

Dining philosophers

```
1   system {
2     extern = _n
3     environment = fork[_n]: 0
4     spawn = Phil: _n
5   }
6
7   agent Phil {
8     interface = status: 0
9
10    Behavior =
11      fork[id] = 0 ->
12        fork[id] <-- 1;
13        status <- 1;
14        fork[(id+1) % _n] = 0 ->
15          fork[(id+1) % _n] <-- 1;
16          status <- 2;
17          fork[(id+1) % _n] <-- 0;
18          status <- 3;
19          fork[id] <-- 0;
20          status <- 0;
21          Behavior
22  }
23
24  check {
25    NoDeadlock =
26      always exists Phil p,
27        status of p != 1
28  }
```

```
sliver.py philosophers.labs n=5 --backend cadp
```

```
1   <initialization>
2           fork[0] <-- 0
3           fork[1] <-- 0
4           fork[2] <-- 0
5           fork[3] <-- 0
6           fork[4] <-- 0
7   Phil 0: status <- 0
8   Phil 1: status <- 0
9   Phil 2: status <- 0
10  Phil 3: status <- 0
11  Phil 4: status <- 0
12  <end initialization>
13  Phil 0: fork[0] <-- 1
14  Phil 4: fork[4] <-- 1
15  Phil 4: status <- 1
16  Phil 3: fork[3] <-- 1
17  Phil 3: status <- 1
18  Phil 2: fork[2] <-- 1
19  Phil 2: status <- 1
20  Phil 1: fork[1] <-- 1
21  Phil 1: status <- 1
22  Phil 0: status <- 1
23  <property violated>
```

Usage: verification mode

Dining philosophers

```
1   system {
2     extern = _n
3     environment = fork[_n]: 0
4     spawn = Phil: _n
5   }
6
7   agent Phil {
8     interface = status: 0
9
10    Behavior =
11      fork[id] = 0 ->
12        fork[id] <-- 1;
13        status <- 1;
14        fork[(id+1) % _n] = 0 ->
15          fork[(id+1) % _n] <-- 1;
16          status <- 2;
17          fork[(id+1) % _n] <-- 0;
18          status <- 3;
19          fork[id] <-- 0;
20          status <- 0;
21          Behavior
22  }
23
24  check {
25    NoDeadlock =
26      always exists Phil p,
27        status of p != 1
28  }
```

```
sliver.py philosophers.labs n=5 --backend cadp
```

```
1   <initialization>
2           fork[0] <-- 0
3           fork[1] <-- 0
4           fork[2] <-- 0
5           fork[3] <-- 0
6           fork[4] <-- 0
7   Phil 0: status <- 0
8   Phil 1: status <- 0
9   Phil 2: status <- 0
10  Phil 3: status <- 0
11  Phil 4: status <- 0
12  <end initialization>
13  Phil 0: fork[0] <-- 1
14  Phil 4: fork[4] <-- 1
15  Phil 4: status <- 1
16  Phil 3: fork[3] <-- 1
17  Phil 3: status <- 1
18  Phil 2: fork[2] <-- 1
19  Phil 2: status <- 1
20  Phil 1: fork[1] <-- 1
21  Phil 1: status <- 1
22  Phil 0: status <- 1
23  <property violated>
```

Usage: verification mode

Dining philosophers

```
1   system {
2     extern = _n
3     environment = fork[_n]: 0
4     spawn = Phil: _n
5   }
6
7   agent Phil {
8     interface = status: 0
9
10    Behavior =
11      fork[id] = 0 ->
12        fork[id] <-- 1;
13        status <- 1;
14        fork[(id+1) % _n] = 0 ->
15          fork[(id+1) % _n] <-- 1;
16          status <- 2;
17          fork[(id+1) % _n] <-- 0;
18          status <- 3;
19          fork[id] <-- 0;
20          status <- 0;
21          Behavior
22  }
23
24  check {
25    NoDeadlock =
26      always exists Phil p,
27        status of p != 1
28  }
```

```
sliver.py philosophers.labs n=5 --backend cadp
```

```
1   <initialization>
2           fork[0] <-- 0
3           fork[1] <-- 0
4           fork[2] <-- 0
5           fork[3] <-- 0
6           fork[4] <-- 0
7   Phil 0: status <- 0
8   Phil 1: status <- 0
9   Phil 2: status <- 0
10  Phil 3: status <- 0
11  Phil 4: status <- 0
12  <end initialization>
13  Phil 0: fork[0] <-- 1
14  Phil 4: fork[4] <-- 1
15  Phil 4: status <- 1
16  Phil 3: fork[3] <-- 1
17  Phil 3: status <- 1
18  Phil 2: fork[2] <-- 1
19  Phil 2: status <- 1
20  Phil 1: fork[1] <-- 1
21  Phil 1: status <- 1
22  Phil 0: status <- 1
23  <property violated>
```

Usage: simulation mode

Leader election

```
sliver.py leader.labs n=3 --backend cadp
    --simulate 1 --steps 100
```

```
1   system {
2     extern = _n
3     spawn = Node: _n
4   }
5
6   stigmergy Election {
7     link = true
8     leader: _n
9   }
10
11  agent Node {
12    stigmergies = Election
13    Behavior =
14      leader > id ->
15        leader <~ id;
16        Behavior
17  }
18
19  check {
20    LeaderIs0 =
21      eventually forall Node a,
22        leader of a = 0
23  }
```

```
1   <initialization>
2   Node 0: leader <~ 3,0
3   Node 1: leader <~ 3,1
4   Node 2: leader <~ 3,2
5   <end initialization>
6   Node 0: leader <~ 0,3
7   Node 2: leader <~ 2,4
8   <Node 0: confirm 'leader'>
9   Node 1: leader <~ 0,3
10  <Node 0: end confirm 'leader'>
11  <Node 1: propagate 'leader'>
12  <Node 1: end propagate 'leader'>
13  <Node 0: propagate 'leader'>
14  <Node 0: end propagate 'leader'>
15  <Node 2: confirm 'leader'>
16  Node 0: leader <~ 2,4
17  Node 1: leader <~ 2,4
18  <Node 2: end confirm 'leader'>
19  <Node 0: propagate 'leader'>
20  <Node 0: end propagate 'leader'>
21  Node 0: leader <~ 0,8
22  <Node 0: propagate 'leader'>
23  Node 1: leader <~ 0,8
24  Node 2: leader <~ 0,8
25  <Node 0: end propagate 'leader'>
26  <property satisfied>
```

Usage: simulation mode

Leader election

```
sliver.py leader.labs n=3 --backend cadp
    --simulate 1 --steps 100
```

```
1   system {
2     extern = _n
3     spawn = Node: _n
4   }
5
6   stigmergy Election {
7     link = true
8     leader: _n
9   }
10
11  agent Node {
12    stigmergies = Election
13    Behavior =
14      leader > id ->
15        leader <~ id;
16        Behavior
17  }
18
19  check {
20    LeaderIs0 =
21      eventually forall Node a,
22        leader of a = 0
23  }
```

```
1   <initialization>
2   Node 0: leader <~ 3,0
3   Node 1: leader <~ 3,1
4   Node 2: leader <~ 3,2
5   <end initialization>
6   Node 0: leader <~ 0,3
7   Node 2: leader <~ 2,4
8   <Node 0: confirm 'leader'>
9   Node 1: leader <~ 0,3
10  <Node 0: end confirm 'leader'>
11  <Node 1: propagate 'leader'>
12  <Node 1: end propagate 'leader'>
13  <Node 0: propagate 'leader'>
14  <Node 0: end propagate 'leader'>
15  <Node 2: confirm 'leader'>
16  Node 0: leader <~ 2,4
17  Node 1: leader <~ 2,4
18  <Node 2: end confirm 'leader'>
19  <Node 0: propagate 'leader'>
20  <Node 0: end propagate 'leader'>
21  Node 0: leader <~ 0,8
22  <Node 0: propagate 'leader'>
23  Node 1: leader <~ 0,8
24  Node 2: leader <~ 0,8
25  <Node 0: end propagate 'leader'>
26  <property satisfied>
```

Usage: simulation mode

Leader election

```
sliver.py leader.labs n=3 --backend cadp
    --simulate 1 --steps 100
```

```
1   system {
2     extern = _n
3     spawn = Node: _n
4   }
5
6   stigmergy Election {
7     link = true
8     leader: _n
9   }
10
11  agent Node {
12    stigmergies = Election
13    Behavior =
14      leader > id ->
15        leader <~ id;
16        Behavior
17  }
18
19  check {
20    LeaderIs0 =
21      eventually forall Node a,
22        leader of a = 0
23  }
```

```
1   <initialization>
2   Node 0: leader <~ 3,0
3   Node 1: leader <~ 3,1
4   Node 2: leader <~ 3,2
5   <end initialization>
6   Node 0: leader <~ 0,3
7   Node 2: leader <~ 2,4
8   <Node 0: confirm 'leader'>
9   Node 1: leader <~ 0,3
10  <Node 0: end confirm 'leader'>
11  <Node 1: propagate 'leader'>
12  <Node 1: end propagate 'leader'>
13  <Node 0: propagate 'leader'>
14  <Node 0: end propagate 'leader'>
15  <Node 2: confirm 'leader'>
16  Node 0: leader <~ 2,4
17  Node 1: leader <~ 2,4
18  <Node 2: end confirm 'leader'>
19  <Node 0: propagate 'leader'>
20  <Node 0: end propagate 'leader'>
21  Node 0: leader <~ 0,8
22  <Node 0: propagate 'leader'>
23  Node 1: leader <~ 0,8
24  Node 2: leader <~ 0,8
25  <Node 0: end propagate 'leader'>
26  <property satisfied>
```

Conclusions

SLiVER can use CADP to **verify** and **simulate** multi-agent systems
No knowledge of CADP/LNT is required to the user
Future work

- Formalize encoding correctness
- Compositional verification
- More expressive properties
- Statistical model checking

Artifacts:

- SLiVER releases (Linux)
  `https://github.com/labs-lang/sliver`
- LAbS example specifications
  `https://github.com/labs-lang/labs-examples`