Self-Stabilizing Leader Election in Highly Dynamic Networks

Karine Altisen[1] Stéphane Devismes[2] Anaïs Durand [3]
Colette Johnen[4] Franck Petit[5]

[1] VERIMAG, UGA, Grenoble
[2] MIS, UPJV, Amiens
[3] LIMOS, Univ. Clermont, Clermont Ferrand
[4] LaBRI, Univ. Bordeaux
[5] LIP6, Sorbonne Université, Paris

November 9^{th} 2021, Fontainebleau

Challenge

Self-stabilization in ***Highly Dynamic Networks*** ?

where **topological changes** are not :

{ transient / an anomaly } but { intermittent / inherent }

To tolerate both **transient faults** and **high dynamics**

Case Study: Leader Election

Objectives

Self-stabilizing leader election in highly dynamic message-passing systems

- Finding **conditions** under which **self-stabilizing leader election** can be achieved.

We look for
self-stabilizing algorithm for
general classes of dynamic networks

(*e.g.*, we do not enforce the network to be in a particular topology at a given time)

- Finding the **limits** where self-stabilizing leader election becomes impossible?

- Studying **lower bounds** on the convergence time

Leader Election

n identified processes: $\forall p \in V$, $id(p)$ is the unique identifier of p

Leader Election

n identified processes: $\forall p \in V$, $id(p)$ is the unique identifier of p

Let $IDSET$ be the definition domain of identifiers ($|IDSET| > n$)

$\forall v \in IDSET$,

- v is a **real ID** if $\exists p \in V$, $id(p) = v$
- v is a **fake ID** otherwise

Leader Election

n identified processes: $\forall p \in V$, $id(p)$ is the unique identifier of p

Let $IDSET$ be the definition domain of identifiers ($|IDSET| > n$)

$\forall v \in IDSET$,

- v is a **real ID** if $\exists p \in V$, $id(p) = v$
- v is a **fake ID** otherwise

Every process p computes the identifier of the leader in $lid(p)$

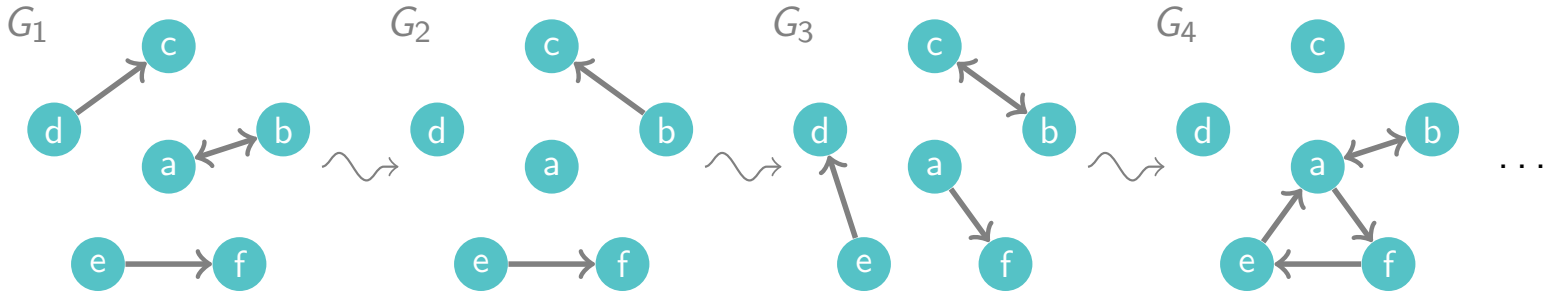
Initially, $lid(p)$ may contain a fake ID

GOAL: converge to a configuration from which all lid variables constantly designates **the same real ID**

Computation Model

- ***Synchronous Rounds:***

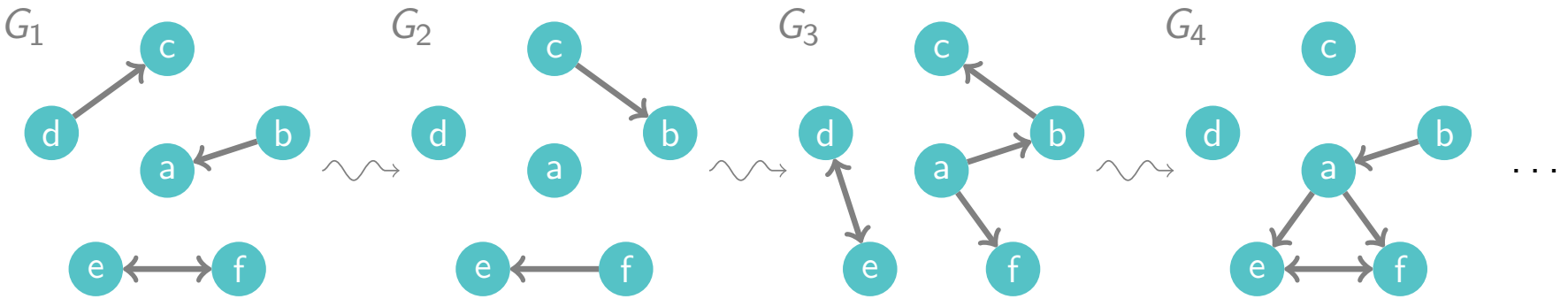
- ***Dynamics*** modeled with a **Dynamic Graph (DG)**



[Xuan *et. al.*, 03], [Casteigts *et. al.*, 13]

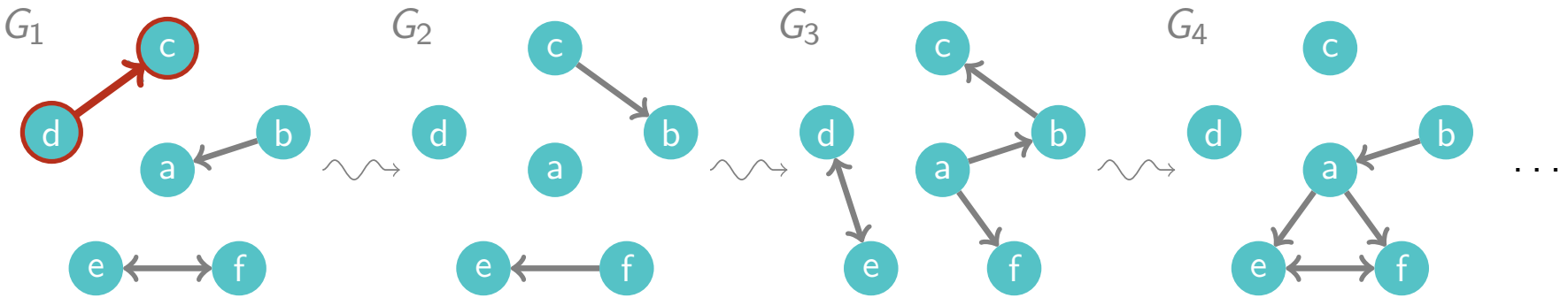
Journey

Can d transmit information to a ?



Journey

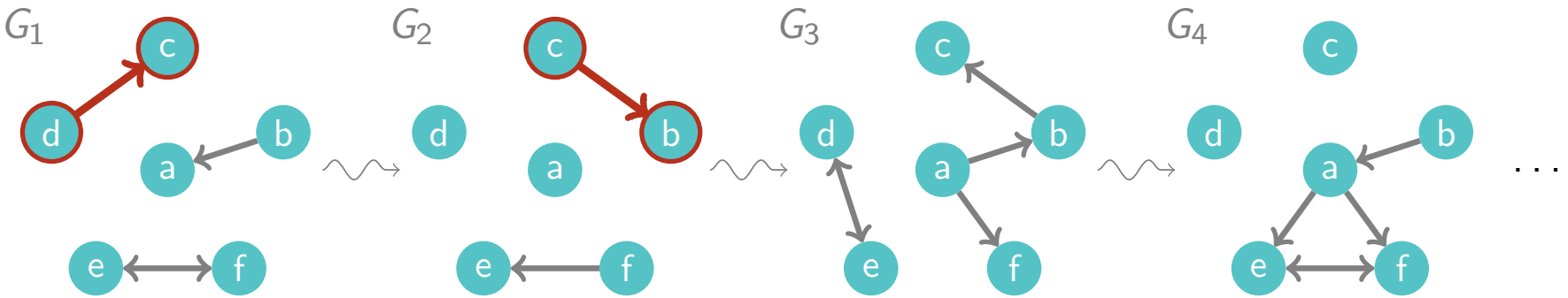
Can d transmit information to a ?



$1, (d, c)$

Journey

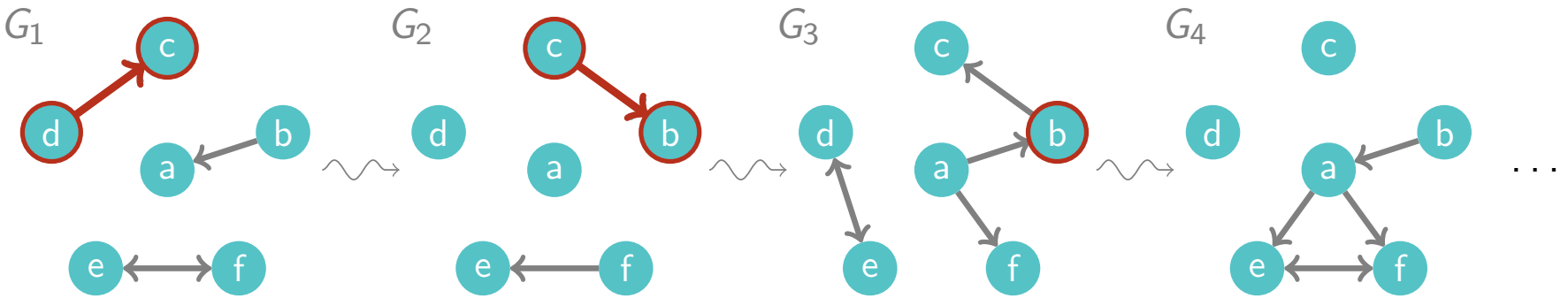
Can d transmit information to a ?



$1, (d, c);\ 2, (c, b)$

Journey

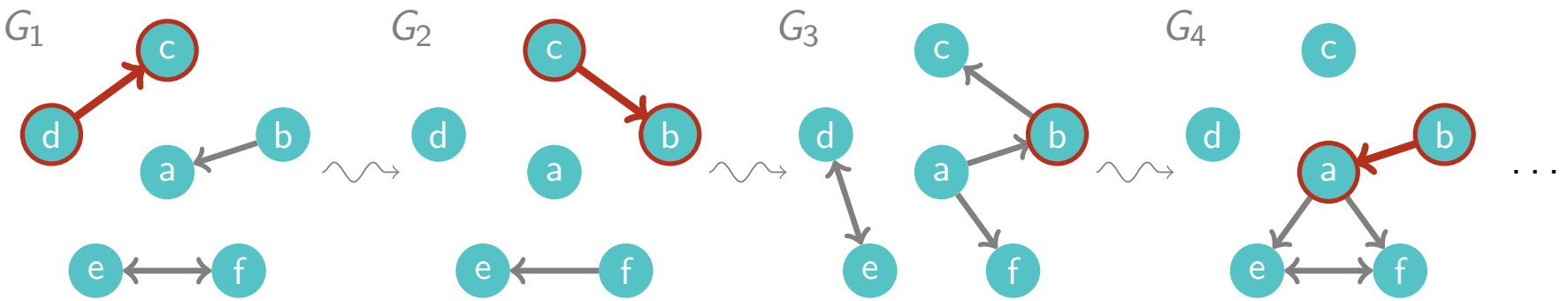
Can d transmit information to a ?



$1, (d, c);\ 2, (c, b)$

Journey

Can d transmit information to a ?



$1, (d, c);\ 2, (c, b);\ 4, (b, a) =$ Journey from d to a of length 4.

Sources and Sinks

Source: **can infinitely often reach** any other
through a journey

Sources and Sinks

Source: **can infinitely often reach** any other through a journey

Quasi-Timely Source: **can infinitely often reach** any other through a journey of length $\leq \Delta$

Sources and Sinks

Source: **can infinitely often reach** any other through a journey

Quasi-Timely Source: **can infinitely often reach** any other through a journey of length $\leq \Delta$

Timely Source: **can always reach** any other through a journey of length $\leq \Delta$

Sources and Sinks

Source: **can infinitely often reach** any other through a journey

Quasi-Timely Source: **can infinitely often reach** any other through a journey of length $\leq \Delta$

Timely Source: **can always reach** any other through a journey of length $\leq \Delta$

Sink: **can infinitely often be reached by** any other through a journey

Quasi-Timely Sink: **can infinitely often be reached by** any other through a journey of length $\leq \Delta$

Timely Sink: **can always be reached by** any other through a journey of length $\leq \Delta$

Classes where All processes are Sources (and so Sinks)

[ICDCN'21]

$\mathcal{J}_{*,*}$: All processes are **sources**

$\mathcal{J}^{\mathcal{Q}}_{*,*}(\Delta)$: All processes are **quasi-timely sources**

$\mathcal{J}^{\mathcal{B}}_{*,*}(\Delta)$: All processes are **timely sources**

Generalization: Classes with at least One Source or One Sink

[PODC'21]

$\mathcal{J}_{1,*}$: **At least one** Source

$\mathcal{J}_{1,*}^{\mathcal{Q}}(\Delta)$: **At least one** Quasi-Timely Source

$\mathcal{J}_{1,*}^{\mathcal{B}}(\Delta)$: **At least one** Timely Source

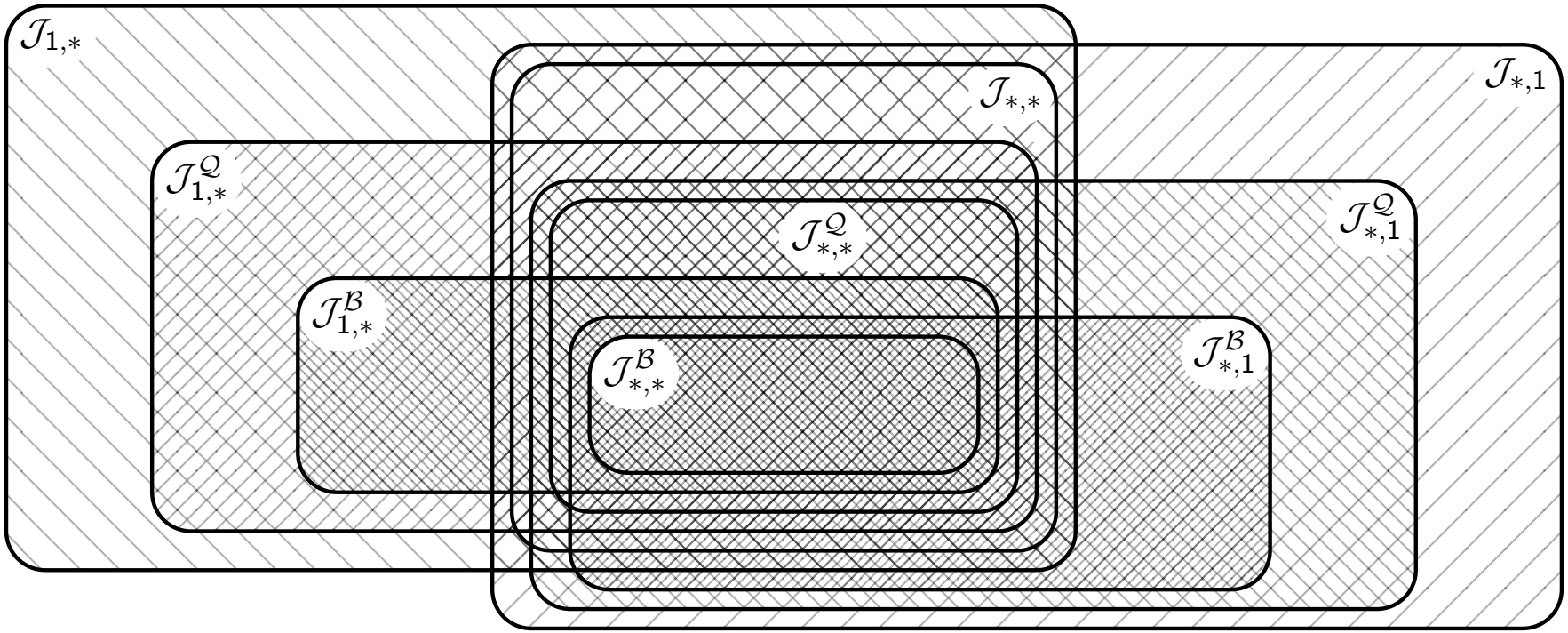
---

$\mathcal{J}_{*,1}$: **At least one** Sink

$\mathcal{J}_{*,1}^{\mathcal{Q}}(\Delta)$: **At least one** Quasi-Timely Sink

$\mathcal{J}_{*,1}^{\mathcal{B}}(\Delta)$: **At least one** Timely Sink

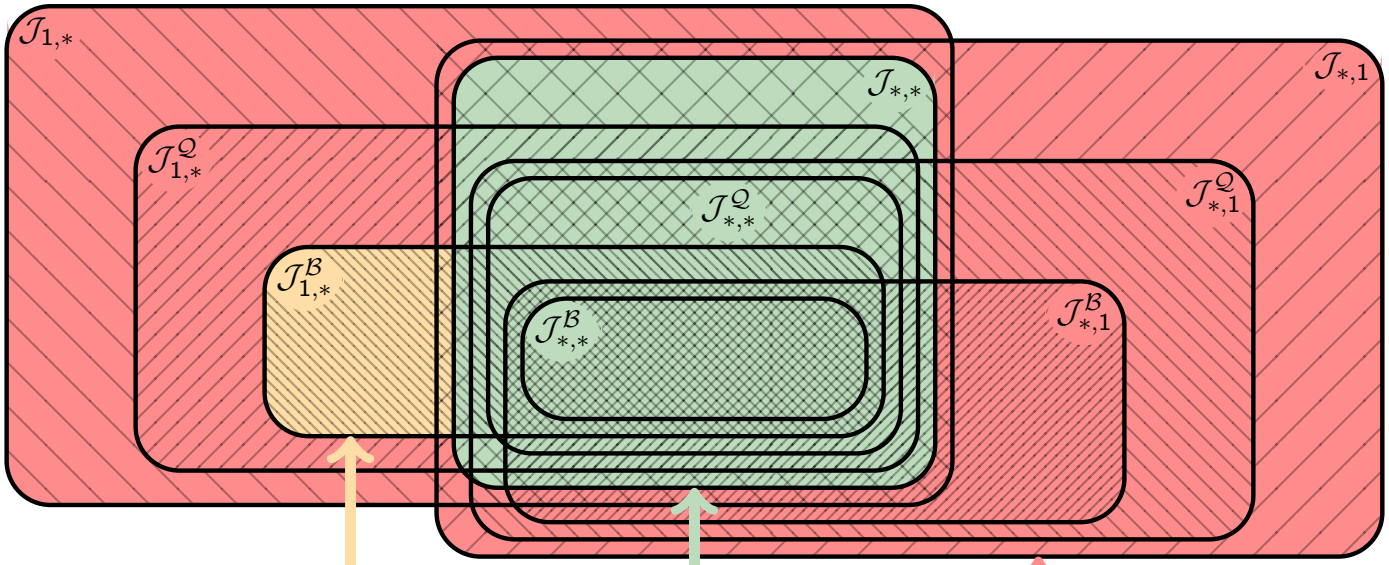
Hierarchy



$$\begin{array}{ccccc}
\mathcal{J}^{\mathcal{B}}_{1,*} & \longrightarrow & \mathcal{J}^{\mathcal{Q}}_{1,*} & \longrightarrow & \mathcal{J}_{1,*} \\
\uparrow & & \uparrow & & \uparrow \\
\mathcal{J}^{\mathcal{B}}_{*,*} & \longrightarrow & \mathcal{J}^{\mathcal{Q}}_{*,*} & \longrightarrow & \mathcal{J}_{*,*} \\
\downarrow & & \downarrow & & \downarrow \\
\mathcal{J}^{\mathcal{B}}_{*,1} & \longrightarrow & \mathcal{J}^{\mathcal{Q}}_{*,1} & \longrightarrow & \mathcal{J}_{*,1}
\end{array}$$

$A \to B$ means that $A \subset B$

Main Results



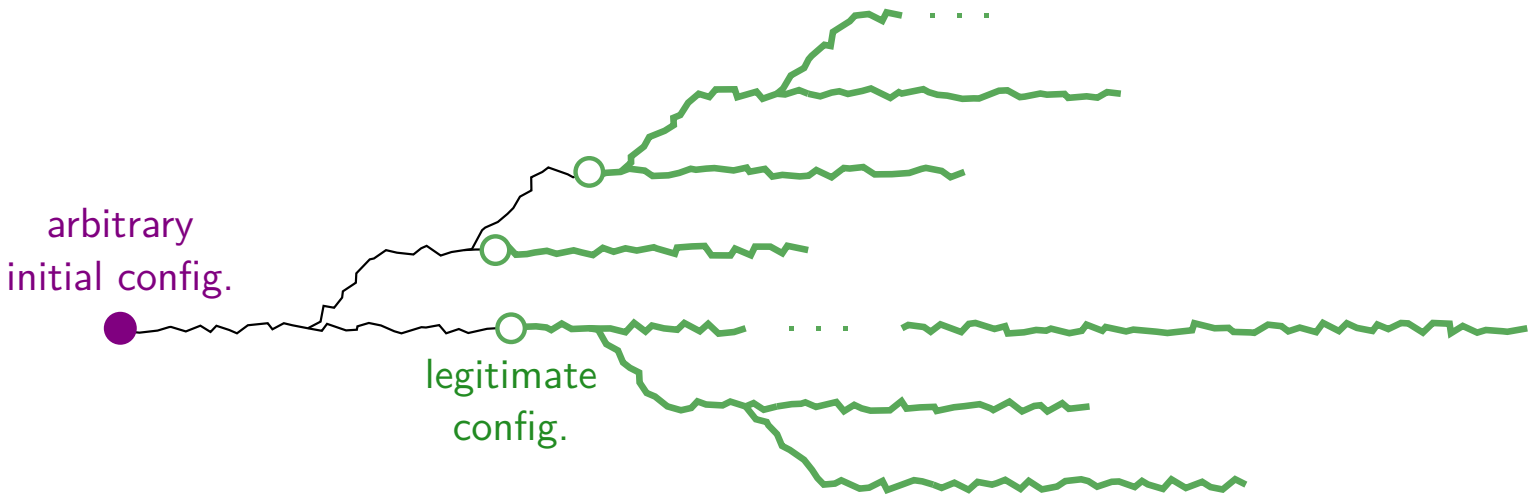
Self-stabilization ✗
Pseudo-stabilization ✓
Convergence Time Unboundable ✗

Self-stabilization ✓
Convergence Time Boundable only in $\mathcal{J}^{\mathcal{B}}_{*,*}$ 😐

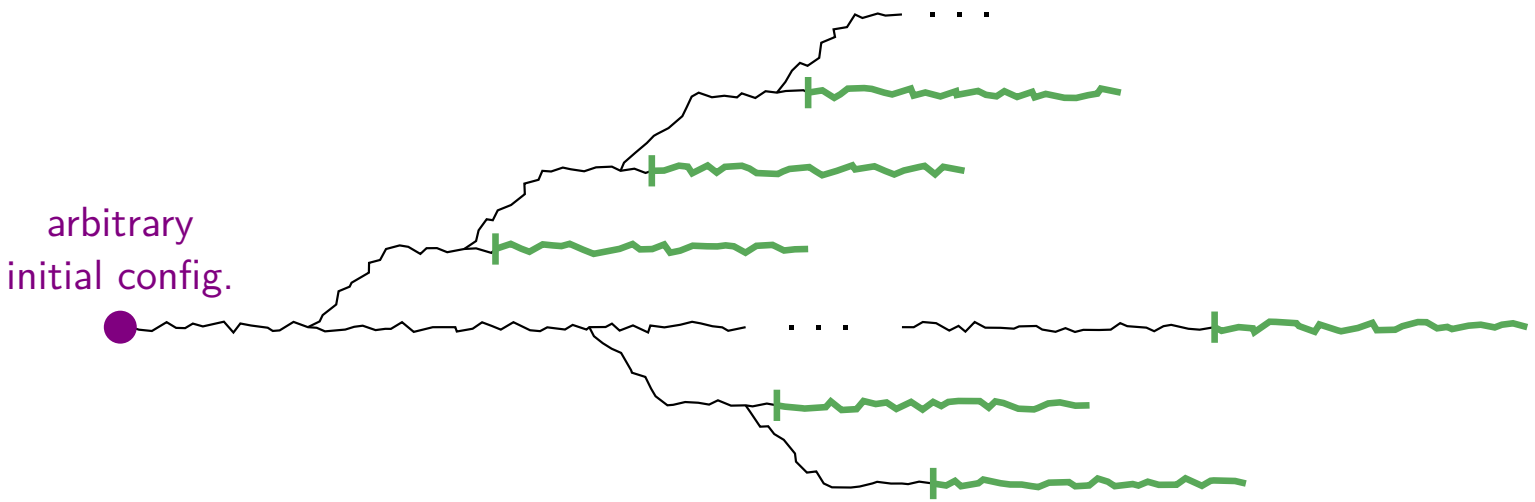
Self-stabilization ✗
Pseudo-stabilization ✗

Self- vs. Pseudo-stabilization: "cannot" vs. "does not"

- ***Self-stabilization:***



- ***Pseudo-stabilization:***



Classes where All Processes are Sources

$$\mathcal{TC}^{\mathcal{B}}(\Delta) \subseteq \mathcal{TC}^{\mathcal{Q}}(\Delta) \subseteq \mathcal{TC}^{\mathcal{R}}$$

Self-stabilizing Leader Election in $\mathcal{TC}^{\mathcal{B}}(\Delta)$, $\mathcal{TC}^{\mathcal{Q}}(\Delta)$, and $\mathcal{TC}^{\mathcal{R}}$

Class $\mathcal{TC}^{\mathcal{B}}(\Delta)$ with $\Delta \in \mathbb{N}^*$ (Bounded Temporal Diameter):

- Δ known
- Stabilization Time: at most 3Δ rounds
- Memory Requirement: $O(\log n + \log \Delta)$ bits per node

Class $\mathcal{TC}^{\mathcal{Q}}(\Delta)$ with $\Delta \in \mathbb{N}^*$ (Quasi Bounded Temporal Diameter):

- Δ and n known
- Memory Requirement: $O(n(\log n + \log \Delta))$ bits per node

Class $\mathcal{TC}^{\mathcal{R}}$(Recurrent Temporal Connectivity):

- n known
- Memory Requirement: infinite

For $\mathcal{TC}^{\mathcal{Q}}(\Delta)$and $\mathcal{TC}^{\mathcal{R}}$, the convergence time cannot be bounded and knowledge of n is mandatory !

n required in $\mathcal{TC}^{\mathcal{Q}}(\Delta)$ and $\mathcal{TC}^{\mathcal{R}}$: *size-ambiguity* (1/3)

Let p_1, p_2, p_3, p_4, p_5, and p_6 be a set of processes.
The identifier of p_i is i.

n required in $\mathcal{TC}^{\mathcal{Q}}(\Delta)$ and $\mathcal{TC}^{\mathcal{R}}$: *size-ambiguity* (1/3)

Let p_1, p_2, p_3, p_4, p_5, and p_6 be a set of processes.
The identifier of p_i is i.

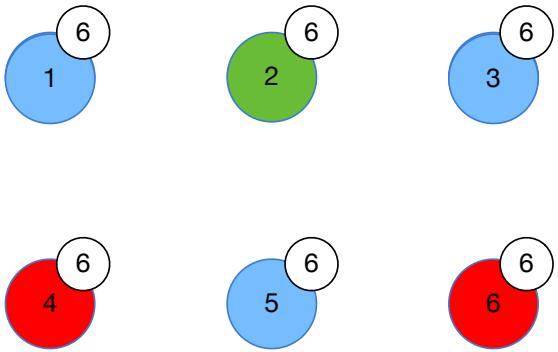
Assume an algorithm $\mathcal{A}$ that "knows" **the number of processes** n in the system.

n required in $\mathcal{TC}^{\mathcal{Q}}(\Delta)$ and $\mathcal{TC}^{\mathcal{R}}$: *size-ambiguity* (1/3)

Let p_1, p_2, p_3, p_4, p_5, and p_6 be a set of processes.
The identifier of p_i is i.

Assume an algorithm $\mathcal{A}$ that "knows" **the number of processes** n in the system.

If $\mathcal{A}$ runs on p_1, p_2, p_3, p_4, p_5, and p_6:

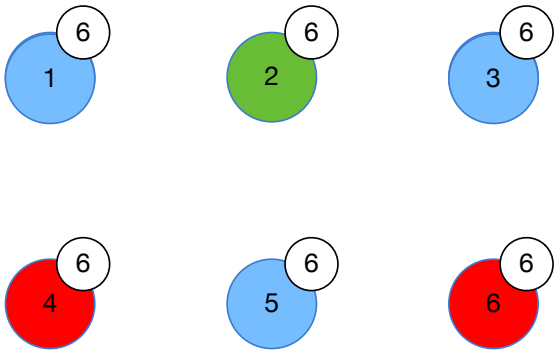


n required in $\mathcal{TC}^{\mathcal{Q}}(\Delta)$ and $\mathcal{TC}^{\mathcal{R}}$: *size-ambiguity* (1/3)

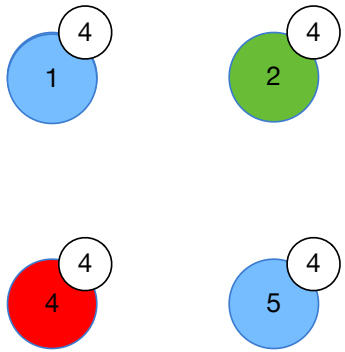
Let p_1, p_2, p_3, p_4, p_5, and p_6 be a set of processes.
The identifier of p_i is i.

Assume an algorithm $\mathcal{A}$ that "knows" **the number of processes** n in the system.

If $\mathcal{A}$ runs on p_1, p_2, p_3, p_4, p_5, and p_6:



If $\mathcal{A}$ runs on p_1, p_2, p_3, and p_4:

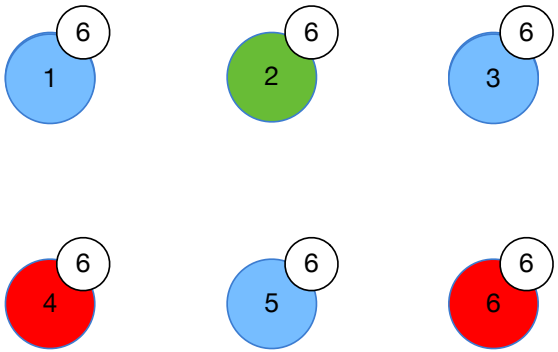


n required in $\mathcal{TC}^{\mathcal{Q}}(\Delta)$ and $\mathcal{TC}^{\mathcal{R}}$: *size-ambiguity* (1/3)

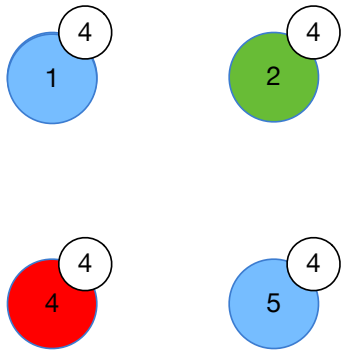
Let p_1, p_2, p_3, p_4, p_5, and p_6 be a set of processes.
The identifier of p_i is i.

Assume an algorithm $\mathcal{A}$ that "knows" **the number of processes** n in the system.

If $\mathcal{A}$ runs on p_1, p_2, p_3, p_4, p_5, and p_6:



If $\mathcal{A}$ runs on p_1, p_2, p_3, and p_4:



It is not size-ambiguous!

n required in $\mathcal{TC}^{\mathcal{Q}}(\Delta)$ and $\mathcal{TC}^{\mathcal{R}}$: *size-ambiguity* (2/3)

Let p_1, p_2, p_3, p_4, p_5, and p_6 be a set of processes.
The identifier of p_i is i.

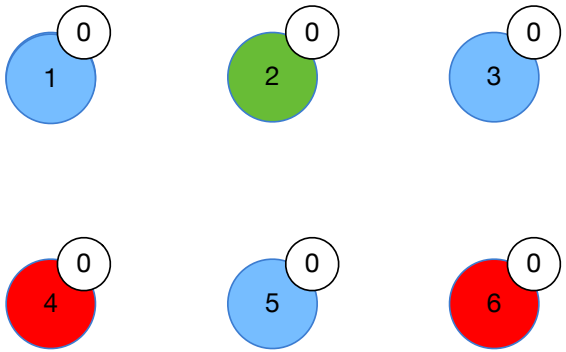
Assume an algorithm $\mathcal{A}$ that "knows" **the parity of number of processes** n in the system.

n required in $\mathcal{TC}^{\mathcal{Q}}(\Delta)$ and $\mathcal{TC}^{\mathcal{R}}$: *size-ambiguity* (2/3)

Let p_1, p_2, p_3, p_4, p_5, and p_6 be a set of processes.
The identifier of p_i is i.

Assume an algorithm $\mathcal{A}$ that "knows" **the parity of number of processes** n in the system.

If $\mathcal{A}$ runs on p_1, p_2, p_3, p_4, p_5, and p_6:

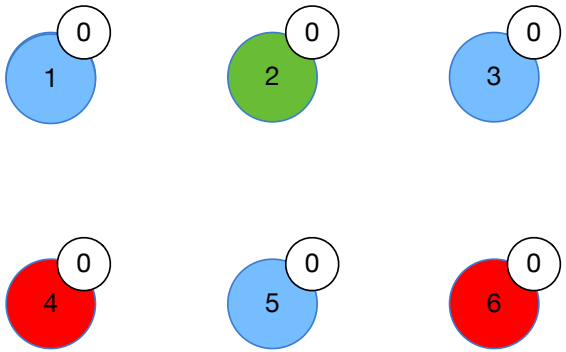


n required in $\mathcal{TC}^{\mathcal{Q}}(\Delta)$ and $\mathcal{TC}^{\mathcal{R}}$: *size-ambiguity* (2/3)

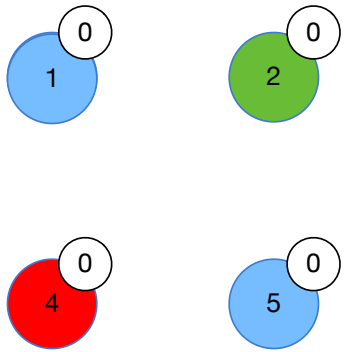
Let p_1, p_2, p_3, p_4, p_5, and p_6 be a set of processes.
The identifier of p_i is i.

Assume an algorithm $\mathcal{A}$ that "knows" **the parity of number of processes** n in the system.

If $\mathcal{A}$ runs on p_1, p_2, p_3, p_4, p_5, and p_6:



If $\mathcal{A}$ runs on p_1, p_2, p_3, and p_4:

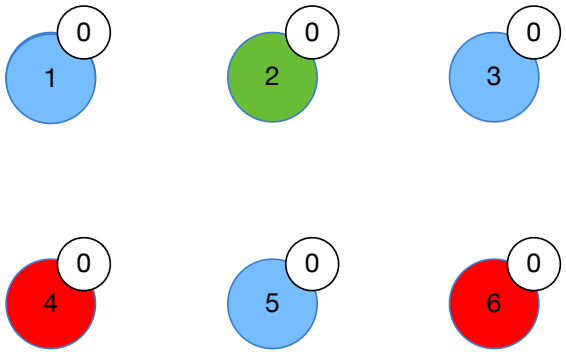


n required in $\mathcal{TC}^{\mathcal{Q}}(\Delta)$ and $\mathcal{TC}^{\mathcal{R}}$: *size-ambiguity* (2/3)

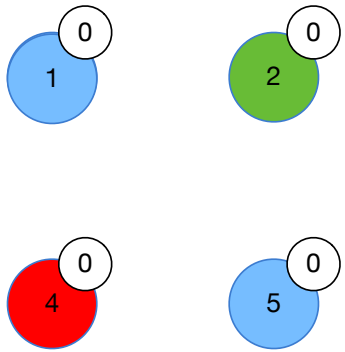
Let p_1, p_2, p_3, p_4, p_5, and p_6 be a set of processes.
The identifier of p_i is i.

Assume an algorithm $\mathcal{A}$ that "knows" **the parity of number of processes** n in the system.

If $\mathcal{A}$ runs on p_1, p_2, p_3, p_4, p_5, and p_6:



If $\mathcal{A}$ runs on p_1, p_2, p_3, and p_4:



It is size-ambiguous!

n required in $\mathcal{TC}^{\mathcal{Q}}(\Delta)$ and $\mathcal{TC}^{\mathcal{R}}$: *size-ambiguity* (3/3)

Let p_1, p_2, p_3, p_4, p_5, and p_6 be a set of processes.
The identifier of p_i is i.

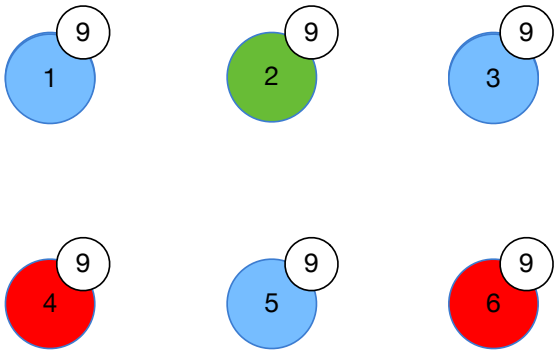
Assume an algorithm $\mathcal{A}$ that "knows" **the bound $K = 9$ of number of processes** n in the system.

n required in $\mathcal{TC}^{\mathcal{Q}}(\Delta)$ and $\mathcal{TC}^{\mathcal{R}}$: *size-ambiguity* (3/3)

Let p_1, p_2, p_3, p_4, p_5, and p_6 be a set of processes.
The identifier of p_i is i.

Assume an algorithm $\mathcal{A}$ that "knows" **the bound $K = 9$ of number of processes** n in the system.

If $\mathcal{A}$ runs on p_1, p_2, p_3, p_4, p_5, and p_6:

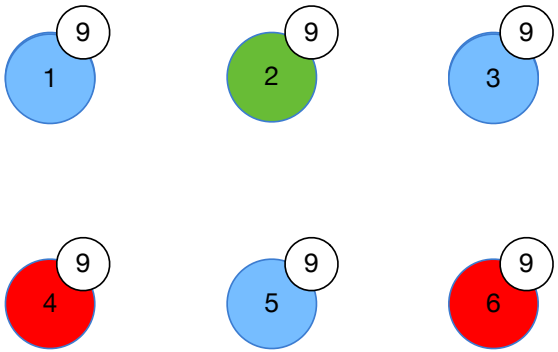


n required in $\mathcal{TC}^{\mathcal{Q}}(\Delta)$ and $\mathcal{TC}^{\mathcal{R}}$: *size-ambiguity* (3/3)

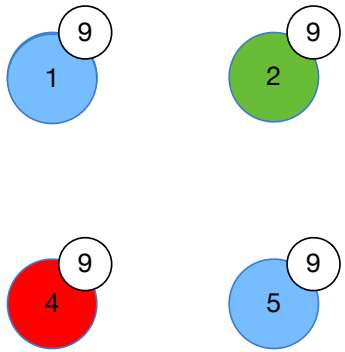
Let p_1, p_2, p_3, p_4, p_5, and p_6 be a set of processes.
The identifier of p_i is i.

Assume an algorithm $\mathcal{A}$ that "knows" **the bound $K = 9$ of number of processes** n in the system.

If $\mathcal{A}$ runs on p_1, p_2, p_3, p_4, p_5, and p_6:



If $\mathcal{A}$ runs on p_1, p_2, p_3, and p_4:

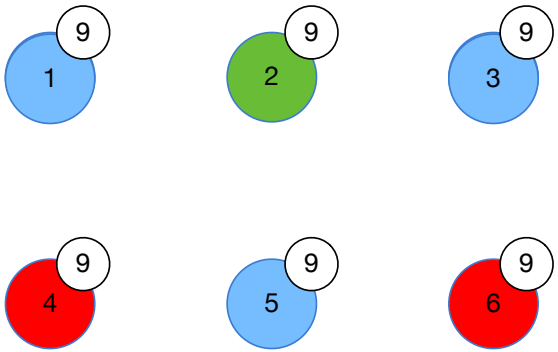


n required in $\mathcal{TC}^{\mathcal{Q}}(\Delta)$ and $\mathcal{TC}^{\mathcal{R}}$: *size-ambiguity* (3/3)

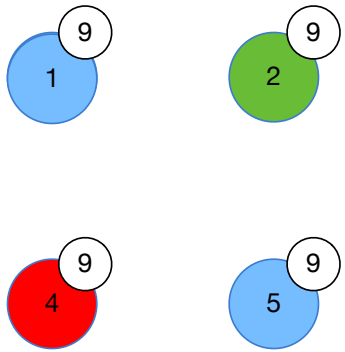
Let p_1, p_2, p_3, p_4, p_5, and p_6 be a set of processes.
The identifier of p_i is i.

Assume an algorithm $\mathcal{A}$ that "knows" **the bound $K = 9$ of number of processes** n in the system.

If $\mathcal{A}$ runs on p_1, p_2, p_3, p_4, p_5, and p_6:



If $\mathcal{A}$ runs on p_1, p_2, p_3, and p_4:



It is size-ambiguous!

Size-ambiguity (definition)

Let V be a set of processes and $k \in \mathbb{N}$.

$\mathcal{A}$ is (k, V)-ambiguous if $0 < k < |V|$ and for every $U \subset V$ such that $|U| = k$, $\mathcal{A}$ can be run on U and for every $p \in U$, p has the same set of states whether $\mathcal{A}$ runs on U or V.

$\mathcal{A}$ is size-ambiguous if there exists V and k such that $\mathcal{A}$ is (k, V)-ambiguous.

Size-ambiguity (definition)

Let V be a set of processes and $k \in \mathbb{N}$.

$\mathcal{A}$ is (k, V)-ambiguous if $0 < k < |V|$ and for every $U \subset V$ such that $|U| = k$, $\mathcal{A}$ can be run on U and for every $p \in U$, p has the same set of states whether $\mathcal{A}$ runs on U or V.

$\mathcal{A}$ is size-ambiguous if there exists V and k such that $\mathcal{A}$ is (k, V)-ambiguous.

$\mathcal{A}$ is size-ambiguous $\approx$ "$\mathcal{A}$ has a partial knowledge of n"

n required in $\mathcal{TC}^{\mathcal{Q}}(\Delta)$ and $\mathcal{TC}^{\mathcal{R}}$: the result

Let $\mathcal{A}$ be any self-stabilizing leader election algorithm for $\mathcal{TC}^{\mathcal{Q}}(\Delta)$ ($\Delta \geq 2$), V be a set of processes, $\mathcal{L}$ be a set of legitimate configurations of $\mathcal{A}$ for V, and $k \in \mathbb{N}$.

$\mathcal{L}$ is **closed** in $\mathcal{TC}^{\mathcal{Q}}(\Delta)$: if γ' reachable from $\gamma \in \mathcal{L}$ by $\mathcal{A}$ in $\mathcal{TC}^{\mathcal{Q}}(\Delta)$, $\gamma' \in \mathcal{L}$ too.

- If $\mathcal{A}$ is (k,V)-ambiguous, then $\mathcal{L}$ is **not** closed in $\mathcal{TC}^{\mathcal{Q}}(\Delta)$. (also holds for $\mathcal{TC}^{\mathcal{B}}(\Delta)$)
- $\exists$ a set of legitimate configurations of $\mathcal{A}$ for V which is closed in $\mathcal{TC}^{\mathcal{Q}}(\Delta)$.

n required in $\mathcal{TC}^{\mathcal{Q}}(\Delta)$ and $\mathcal{TC}^{\mathcal{R}}$: the result

Let $\mathcal{A}$ be any self-stabilizing leader election algorithm for $\mathcal{TC}^{\mathcal{Q}}(\Delta)$ ($\Delta \geq 2$), V be a set of processes, $\mathcal{L}$ be a set of legitimate configurations of $\mathcal{A}$ for V, and $k \in \mathbb{N}$.

$\mathcal{L}$ is **closed** in $\mathcal{TC}^{\mathcal{Q}}(\Delta)$: if γ' reachable from $\gamma \in \mathcal{L}$ by $\mathcal{A}$ in $\mathcal{TC}^{\mathcal{Q}}(\Delta)$, $\gamma' \in \mathcal{L}$ too.

- If $\mathcal{A}$ is (k,V)-ambiguous, then $\mathcal{L}$ is **not** closed in $\mathcal{TC}^{\mathcal{Q}}(\Delta)$. (also holds for $\mathcal{TC}^{\mathcal{B}}(\Delta)$)
- $\exists$ a set of legitimate configurations of $\mathcal{A}$ for V which is closed in $\mathcal{TC}^{\mathcal{Q}}(\Delta)$.

Theorem 1

No self-stabilizing leader election algorithm for $\mathcal{TC}^{\mathcal{Q}}(\Delta)$, with $\Delta \geq 2$, can be size-ambiguous.

Corollary 2

No self-stabilizing leader election algorithm for $\mathcal{TC}^{\mathcal{R}}$ can be size-ambiguous.

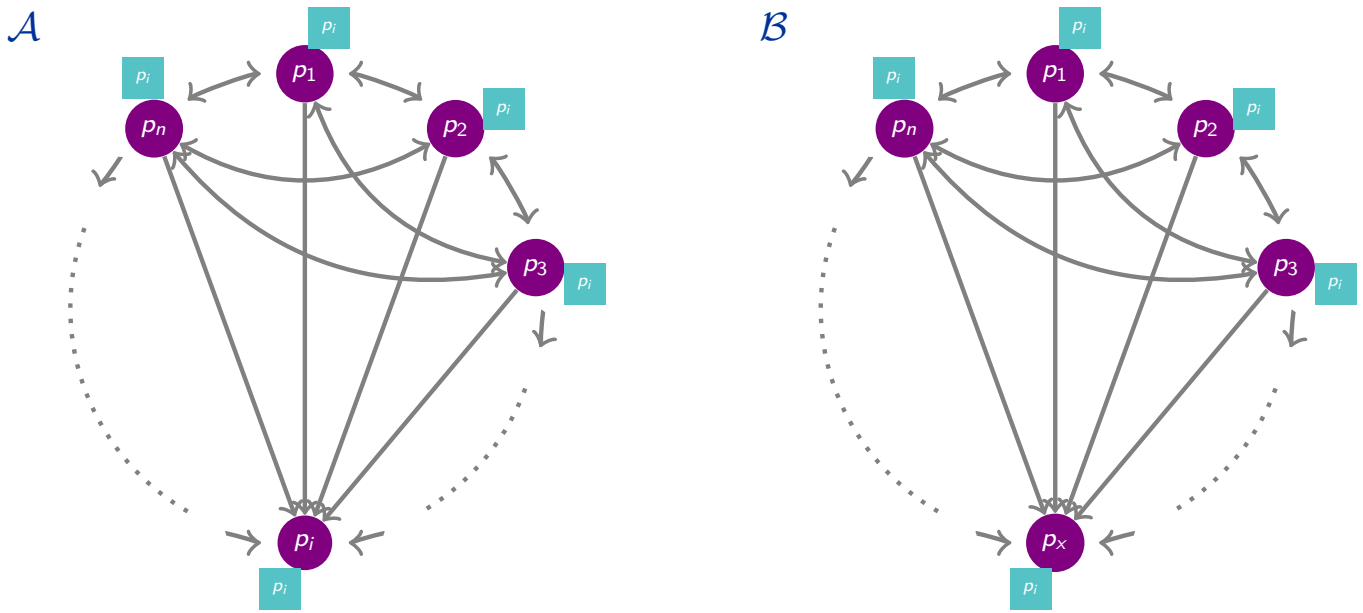
Other Classes: Focus on $\mathcal{J}^{\mathcal{B}}_{1,*}(\Delta)$

At least one a priori unknown process (a timely source)
can always reach any other
through a journey of length $\leq \Delta$

Impossibility of Self-stabilizing Leader Election in $\mathcal{J}^{\mathcal{B}}_{1,*}(\Delta)$

Preliminary Result

In situation $\mathcal{A}$, one process eventually changes its leader output.



Proof:

- All processes except p_i (resp. p_x) are connected to any other at any time
$$\Rightarrow \in \mathcal{J}^{\mathcal{B}}_{1,*}(\Delta)$$
- $\forall j \notin \{i, x\}$, the executions of p_j in $\mathcal{A}$ and $\mathcal{B}$ are ***indistinguishable***
$$\Rightarrow \text{if } p_j \text{ elects } p_i \text{ in } \mathcal{A}, \text{ then } p_j \text{ elects } p_i \text{ in } \mathcal{B} \Rightarrow \otimes$$
$$\Rightarrow p_j \text{ eventually changes its leader}$$

Impossibility of Self-stabilizing Leader Election in $\mathcal{J}^{\mathcal{B}}_{1,*}(\Delta)$

Assume a self-stabilizing algorithm exists

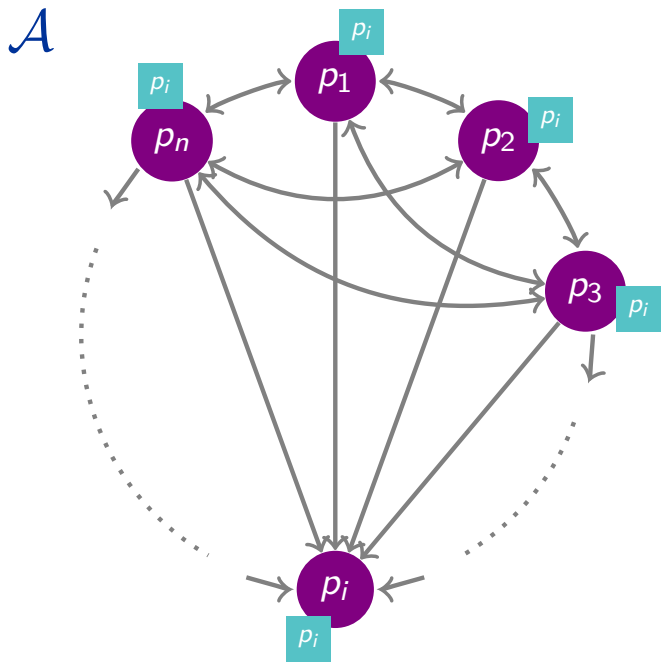
1. Starting from any legitimate configuration, *lid* variables should be constant
2. Now, from any legitimate configuration, situation $\mathcal{A}$ is possible

   Preliminary result $\Rightarrow$ one process eventually changes its leader

Contradiction

Impossibility of Self-stabilizing Leader Election in $\mathcal{J}^{\mathcal{B}}_{1,*}(\Delta)$

Assume a self-stabilizing algorithm exists



1. Starting from any legitimate configuration, *lid* variables should be constant
2. Now, from any legitimate configuration, situation $\mathcal{A}$ is possible

   Preliminary result $\Rightarrow$ one process eventually changes its leader

Contradiction

Pseudo-stabilizing Leader Election Algorithm for $\mathcal{J}^{\mathcal{B}}_{1,*}(\Delta)$

Goal: Electing a "***stable***" process

Stable Process: eventually, **all other processes** receive (maybe indirectly) information about it at least every Δ rounds

Pseudo-stabilizing Leader Election Algorithm for $\mathcal{J}^{\mathcal{B}}_{1,*}(\Delta)$

Goal: Electing a "***stable***" process

Stable Process: eventually, **all other processes** receive (maybe indirectly) information about it at least every Δ rounds

At each round, each process initiates a ***flooding*** (relayed Δ times)

$\rightarrow \exists$ **stable processes**: each source is stable!

Pseudo-stabilizing Leader Election Algorithm for $\mathcal{J}^{\mathcal{B}}_{1,*}(\Delta)$

Goal: Electing a "***stable***" process

Stable Process: eventually, **all other processes** receive (maybe indirectly) information about it at least every Δ rounds

At each round, each process initiates a ***flooding*** (relayed Δ times)

$\rightarrow \exists$ **stable processes**: each source is stable!

How to evaluate stability?

Pseudo-stabilizing Leader Election Algorithm for $\mathcal{J}^{\mathcal{B}}_{1,*}(\Delta)$

Goal: Electing a "***stable***" process

Stable Process: eventually, **all other processes** receive (maybe indirectly) information about it at least every Δ rounds

At each round, each process initiates a ***flooding*** (relayed Δ times)

$\rightarrow \exists$ **stable processes**: each source is stable!

How to evaluate stability? ***suspicion counter***

A process increments its suspicion counter each time it is accused to be NOT stable by some process

After the 1^{st} round, suspicion counters are **monotically non-decreasing**

(a counter may be reset during the first round due to initial inconsistency)

Pseudo-stabilizing Leader Election Algorithm for $\mathcal{J}^{\mathcal{B}}_{1,*}(\Delta)$

Goal: Electing a "***stable***" process

Stable Process: eventually, **all other processes** receive (maybe indirectly) information about it at least every Δ rounds

At each round, each process initiates a ***flooding*** (relayed Δ times)

$\rightarrow \exists$ **stable processes**: each source is stable!

How to evaluate stability? ***suspicion counter***

A process increments its suspicion counter each time it is accused to be NOT stable by some process

After the 1^{st} round, suspicion counters are **monotically non-decreasing**

(a counter may be reset during the first round due to initial inconsistency)

Elected Leader: a process with the minimum suspicion counter value

(we use identifiers to break ties)

Pseudo-stabilizing Leader Election Algorithm for $\mathcal{J}^{\mathcal{B}}_{1,*}(\Delta)$

Locally and Globally Stable Processes

Each process p maintains two maps:

- *LStable*(p): Map of ***locally stable*** processes at p
  $\Rightarrow$ p itself and processes from which p (directly) receives information at most Δ rounds ago.

- *GStable*(p): Map of ***globally stable*** processes
  $=$ locally stable at any process (p included)
  $\Rightarrow$ must eventually contain at least every stable process

Pseudo-stabilizing Leader Election Algorithm for $\mathcal{J}^{\mathcal{B}}_{1,*}(\Delta)$

Locally and Globally Stable Processes

Each process p maintains two maps:

- *LStable*(p): Map of ***locally stable*** processes at p
  $\Rightarrow$ p itself and processes from which p (directly) receives information at most Δ rounds ago.

- *GStable*(p): Map of ***globally stable*** processes
  $=$ locally stable at any process (p included)
  $\Rightarrow$ must eventually contain at least every stable process

p always considers **itself** locally & globally stable

Pseudo-stabilizing Leader Election Algorithm for $\mathcal{J}^{\mathcal{B}}_{1,*}(\Delta)$

Locally and Globally Stable Processes

Values inside $LStable(p)$ and $GStable(p)$: triplet $< id, susp, ttl >$

- id: identifier
- $susp$: the suspicion value of id
- ttl: time to live

Pseudo-stabilizing Leader Election Algorithm for $\mathcal{J}^{\mathcal{B}}_{1,*}(\Delta)$

Locally and Globally Stable Processes

Values inside $LStable(p)$ and $GStable(p)$: triplet $< id, susp, ttl >$

- id: identifier
- $susp$: the suspicion value of id
- ttl: time to live

$LStable(p)$ and $GStable(p)$ are **appended/updated with received information**

- update in $LStable(p)$: information with the highest ttl is considered as the freshest one
- update in $GStable(p)$: received information is considered as fresher and inserted with $ttl = \Delta$

Pseudo-stabilizing Leader Election Algorithm for $\mathcal{J}^{\mathcal{B}}_{1,*}(\Delta)$

Locally and Globally Stable Processes

Values inside $LStable(p)$ and $GStable(p)$: triplet $< id, susp, ttl >$

- id: identifier
- $susp$: the suspicion value of id
- ttl: time to live

$LStable(p)$ and $GStable(p)$ are **appended/updated with received information**

- update in $LStable(p)$: information with the highest ttl is considered as the freshest one
- update in $GStable(p)$: received information is considered as fresher and inserted with $ttl = \Delta$

A triplet is **removed** from a map when its ttl reaches 0

Pseudo-stabilizing Leader Election Algorithm for $\mathcal{J}^{\mathcal{B}}_{1,*}(\Delta)$

At Every Round

1. p initiates a flooding of the triplet $< id(p), LSP = LStable(p), ttl = \Delta >$

Pseudo-stabilizing Leader Election Algorithm for $\mathcal{J}^{\mathcal{B}}_{1,*}(\Delta)$

At Every Round

1. p initiates a flooding of the triplet $< id(p), LSP = LStable(p), ttl = \Delta >$
2. p updates $LStable(p)$ and $GStable(p)$ according to received triplets

Pseudo-stabilizing Leader Election Algorithm for $\mathcal{J}^{\mathcal{B}}_{1,*}(\Delta)$

At Every Round

1. p initiates a flooding of the triplet $< id(p), LSP = LStable(p), ttl = \Delta >$
2. p updates $LStable(p)$ and $GStable(p)$ according to received triplets
3. ttl variables (except those associated to p) are decremented and expired triplets are deleted

Pseudo-stabilizing Leader Election Algorithm for $\mathcal{J}^{\mathcal{B}}_{1,*}(\Delta)$

At Every Round

1. p initiates a flooding of the triplet $< id(p), LSP = LStable(p), ttl = \Delta >$
2. p updates $LStable(p)$ and $GStable(p)$ according to received triplets
3. ttl variables (except those associated to p) are decremented and expired triplets are deleted
4. For each received LSP map, if $id(p)$ is absent from LSP, p increments its ***suspicion counter***

Pseudo-stabilizing Leader Election Algorithm for $\mathcal{J}^{\mathcal{B}}_{1,*}(\Delta)$

At Every Round

1. p initiates a flooding of the triplet $< id(p), LSP = LStable(p), ttl = \Delta >$
2. p updates $LStable(p)$ and $GStable(p)$ according to received triplets
3. ttl variables (except those associated to p) are decremented and expired triplets are deleted
4. For each received LSP map, if $id(p)$ is absent from LSP, p increments its ***suspicion counter***
5. p elects $q \in GStable(p)$ with ***lowest suspicion counter***

Pseudo-stabilizing Leader Election Algorithm for $\mathcal{J}_{1,*}^{\mathcal{B}}(\Delta)$

Pseudo-stabilization

1. **"Time To Live"** allow to delete fake IDs.

Pseudo-stabilizing Leader Election Algorithm for $\mathcal{J}^{\mathcal{B}}_{1,*}(\Delta)$

Pseudo-stabilization

1. **"Time To Live"** allow to delete fake IDs.

2. Let s be a **source**.
   $p \neq s$ receives $< id(s), LSP, ttl >$ at least every Δ rounds

   $\rightarrow$ **eventually** $id(s) \in LStable(p)$ **forever,** $\forall p \in V$
   $\Rightarrow$ **the suspicion counter of s is eventually forever constant**

   $\rightarrow$ $id(s) \in LSP$
   $\Rightarrow$ **eventually** $id(s) \in GStable(p)$ **forever,** $\forall p \in V$

Pseudo-stabilizing Leader Election Algorithm for $\mathcal{J}^{\mathcal{B}}_{1,*}(\Delta)$

Pseudo-stabilization

1. **"Time To Live"** allow to delete fake IDs.

2. Let s be a **source**.
   $p \neq s$ receives $< id(s), LSP, ttl >$ at least every Δ rounds

   $\rightarrow$ **eventually** $id(s) \in LStable(p)$ **forever,** $\forall p \in V$
   $\Rightarrow$ **the suspicion counter of s is eventually forever constant**

   $\rightarrow$ $id(s) \in LSP$
   $\Rightarrow$ **eventually** $id(s) \in GStable(p)$ **forever,** $\forall p \in V$

3. Let x be a process whose suspicion counter is eventually constant
   Eventually $id(x) \in LStable(s)$ forever, for every source s

   $\rightarrow$ $id(x) \in GStable(p)$, $\forall p \in V$

Pseudo-stabilizing Leader Election Algorithm for $\mathcal{J}^{\mathcal{B}}_{1,*}(\Delta)$

Pseudo-stabilization

4 x infinitely often absent of $GStable(p)$
$\Longrightarrow$ infinitely often, during Δ consecutive rounds,
p only receives $< -, LSP, - >$ with $id(x) \notin LSP$

Some of those triplets were initiated by sources
$\rightarrow$ x also receives these latter
$\rightarrow$ x **increments its counter infinitely often**

Pseudo-stabilizing Leader Election Algorithm for $\mathcal{J}^{\mathcal{B}}_{1,*}(\Delta)$

Pseudo-stabilization

4 x infinitely often absent of $GStable(p)$
$\Longrightarrow$ infinitely often, during Δ consecutive rounds,
p only receives $< -, LSP, - >$ with $id(x) \notin LSP$

Some of those triplets were initiated by sources
$\rightarrow$ x also receives these latter
$\rightarrow$ x **increments its counter infinitely often**

5 Eventually:

Processes with **eventually const.** susp. counter (at least 1) $\in GStable(p)$ forever

Suspicion counter of other processes $>$ constant suspicion counters

Pseudo-stabilizing Leader Election Algorithm for $\mathcal{J}^{\mathcal{B}}_{1,*}(\Delta)$

Pseudo-stabilization

4 x infinitely often absent of $GStable(p)$
$\Longrightarrow$ infinitely often, during Δ consecutive rounds,
p only receives $< -, LSP, - >$ with $id(x) \notin LSP$

Some of those triplets were initiated by sources
$\rightarrow$ x also receives these latter
$\rightarrow$ x **increments its counter infinitely often**

5 Eventually:

Processes with **eventually const.** susp. counter (at least 1) $\in GStable(p)$ forever

Suspicion counter of other processes $>$ constant suspicion counters

6 Eventually, $\ell \in GStable(p)$ with **lowest suspicion counter** is the same at every p
$\Longrightarrow$ **every process elects** ℓ, a stable process.

Conclusion

Conclusion

Time Complexity

When stabilization is possible, ***convergence time*** is, most of the time, unboundable ...

Notable exception: $\mathcal{J}^{\mathcal{B}}_{*,*}(\Delta)$

Conclusion

Time Complexity

When stabilization is possible, ***convergence time*** is, most of the time, unboundable ...

Notable exception: $\mathcal{J}^{\mathcal{B}}_{*,*}(\Delta)$

However, to mitigate this issue, we have speculation

Conclusion

Speculation

Speculation [Kotla *et al.*, ACM Trans. Comput. Syst., 2009]:

- the system satisfies its requirements for all executions,
- but also exhibits significantly better performances in a subset of more probable executions.

Idea: worst possible scenarios are often rare in practice.

When the stabilization time cannot be bounded in a class, we exhibit **an important subclass where it can be bounded**.

Conclusion

Speculation

Speculation [Kotla *et al.*, ACM Trans. Comput. Syst., 2009]:

- the system satisfies its requirements for all executions,
- but also exhibits significantly better performances in a subset of more probable executions.

Idea: worst possible scenarios are often rare in practice.

When the stabilization time cannot be bounded in a class, we exhibit **an important subclass where it can be bounded**.

In all cases where stabilization is possible but the convergence time is unboundable, our algorithms are speculative: when deployed in ***the subclass*** $\mathcal{J}^{\mathcal{B}}_{*,*}(\Delta)$, the convergence time is in $O(\Delta)$ ***rounds***.

Conclusion

Space Complexity

Important open questions:

- Can we solve pseudo-stabilizing leader election in $\mathcal{J}^{\mathcal{B}}_{1,*}$ with **a bounded memory**?
- Can we solve self-stabilizing leader election in $\mathcal{J}_{*,*}$ with **a bounded memory**?

Thank You for Your Attention

Publications:

- Karine Altisen, Stéphane Devismes, and Anaïs Durand, Colette Johnen, and Franck Petit. Self-stabilizing Systems in Spite of High Dynamics. *ICDCN'21*.
- Karine Altisen, Stéphane Devismes, and Anaïs Durand, Colette Johnen, and Franck Petit. On Implementing Stabilizing Leader Election with Weak Assumptions on Network Dynamics. *PODC'21*.