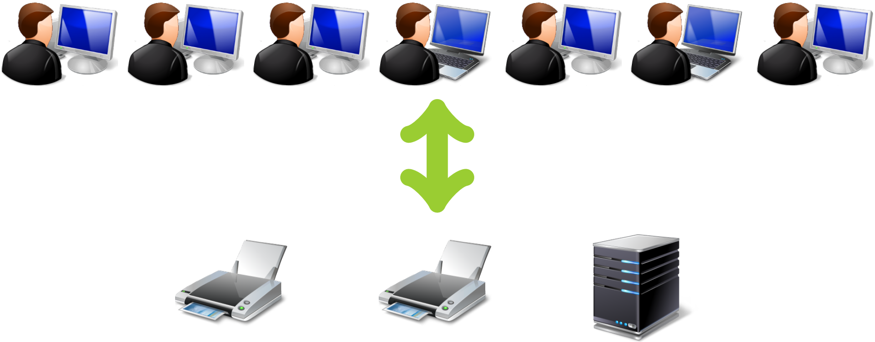
Concurrency in Snap-Stabilizing Local Resource Allocation

Karine Altisen Stéphane Devismes **Anaïs Durand**

May ??, 2015

Resource Allocation Problems

n processes, k resources, $n \gg k$



Resource Allocation Problems

Critical Section (CS)

- Code to access a resource
- **Finite** but unbounded (*i.e.* unpredictable)

Resource Allocation Problems

Critical Section (CS)

- Code to access a resource
- **Finite** but unbounded (*i.e.* unpredictable)

With Several Resources ($k > 1$)

Resource Allocation Problems

Critical Section (CS)

- Code to access a resource
- **Finite** but unbounded (*i.e.* unpredictable)

With Several Resources ($k > 1$)

- **Concurrency:** Maximize the utilization of the resources

ℓ-Exclusion [Fischer *et al*, 79]

ℓ **identical** copies of a non-shareable reusable resource

Properties

ℓ-Exclusion [Fischer *et al*, 79]

ℓ **identical** copies of a non-shareable reusable resource

Properties

- **Safety:** $|\{\text{processes concurrently in CS}\}| \leq \ell$

ℓ-Exclusion [Fischer *et al*, 79]

ℓ **identical** copies of a non-shareable reusable resource

Properties

- **Safety:** $|\{\text{processes concurrently in CS}\}| \leq \ell$
- **Fairness:** Requesting process eventually enters CS

ℓ-Exclusion [Fischer *et al*, 79]

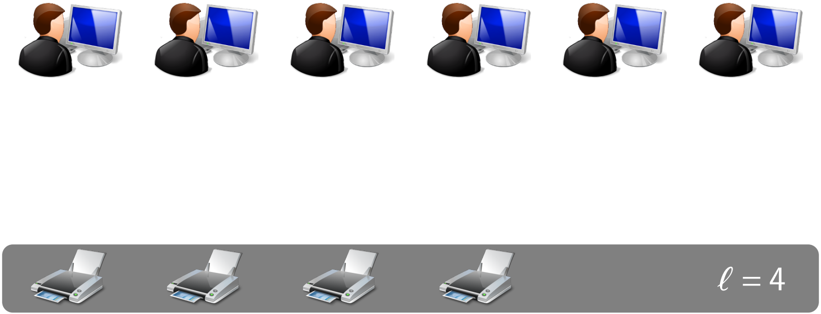
ℓ **identical** copies of a non-shareable reusable resource

Properties

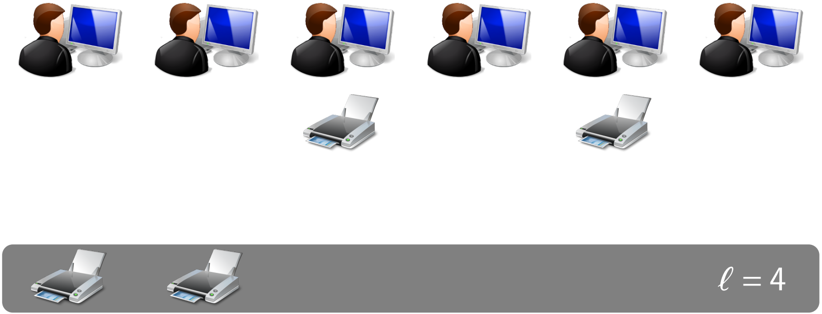
- **Safety:** $|\{\text{processes concurrently in CS}\}| \leq \ell$
- **Fairness:** Requesting process eventually enters CS
- **Avoiding ℓ-Deadlock:**
  If $|\{\text{processes concurrently in CS}\}| < \ell$

  then
  - a requesting process can obtain CS
  - **even if no process leaves CS meanwhile**

Avoiding ℓ-Deadlock [Fischer *et al*, 79]

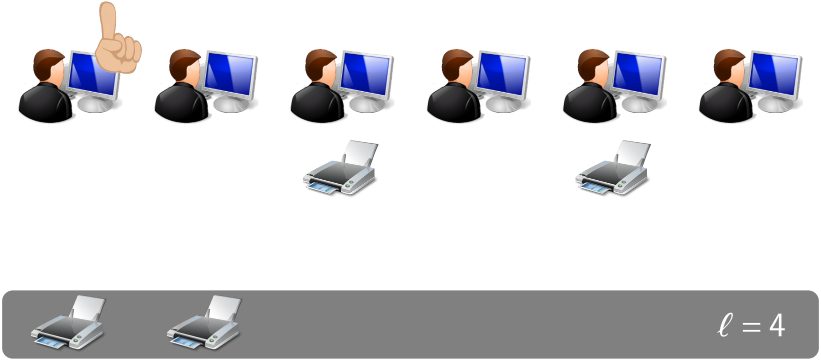


Avoiding ℓ-Deadlock [Fischer *et al*, 79]

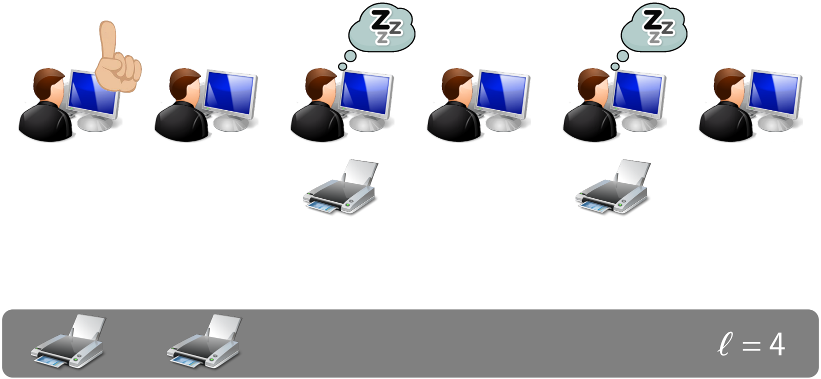


$\ell = 4$

Avoiding ℓ-Deadlock [Fischer *et al*, 79]

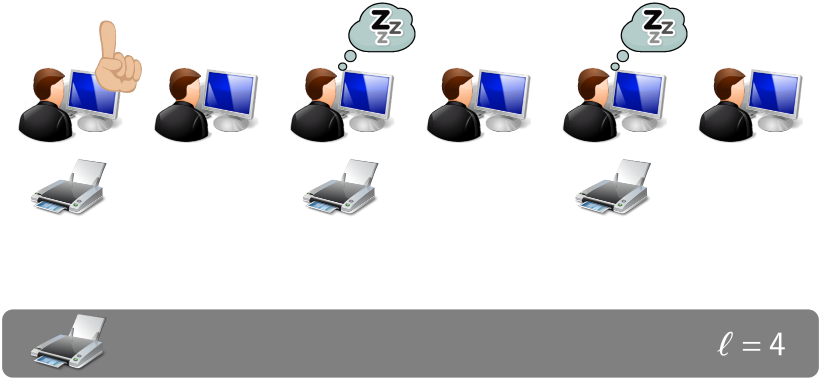


Avoiding ℓ-Deadlock [Fischer *et al*, 79]



$\ell = 4$

Avoiding ℓ-Deadlock [Fischer *et al*, 79]



$\ell = 4$

Property Handling Concurrency

Avoiding ℓ-deadlock = property handling **concurrency**

Property Handling Concurrency

Avoiding ℓ-deadlock = property handling **concurrency**

- **Necessary** to prevent **degenerated solutions**:
  A mutual exclusion algorithm satisfies the safety and fairness of ℓ-exclusion problem.

Property Handling Concurrency

Avoiding ℓ-deadlock = property handling **concurrency**

- **Necessary** to prevent **degenerated solutions**:
  A mutual exclusion algorithm satisfies the safety and fairness of ℓ-exclusion problem.
- But, often **not considered** in correctness proofs of resource allocation algorithms.

Several Properties Handling Concurrency

- **Avoiding ℓ-Deadlock:**

  ℓ-exclusion problem [Fischer *et al*, 79]

- **(k,ℓ)-Liveness:**

  k-out-of-ℓ-exclusion problem [Datta *et al*, 03]

- **Maximal-Concurrency:**

  Committee coordination problem [Bonakdarpour *et al*, 11]

Several Properties Handling Concurrency

- **Avoiding ℓ-Deadlock:**

  ℓ-exclusion problem [Fischer *et al*, 79]

- **(k,ℓ)-Liveness:**

  k-out-of-ℓ-exclusion problem [Datta *et al*, 03]

- **Maximal-Concurrency:**

  Committee coordination problem [Bonakdarpour *et al*, 11]

Drawback : **dedicated** to a specific problem

Maximal-Concurrency

Generalization of the previous properties

where $P_{FREE} = \{$ requesting processes can obtain CS without violating safety $\}$

Maximal-Concurrency

Generalization of the previous properties

Maximal-Concurrency

If $P_{FREE} \neq \emptyset$

then
- a requesting process can obtain CS
- **even if no process leaves CS meanwhile**

where $P_{FREE} = \{$ requesting processes can obtain CS without violating safety $\}$

Maximal-Concurrency

Generalization of the previous properties

Maximal-Concurrency

If $P_{FREE} \neq \emptyset$

then

- a requesting process can obtain CS
- **even if no process leaves CS meanwhile**

Equivalent Definition of Maximal-Concurrency

If CSs last a **long enough time**

then eventually $P_{FREE} = \emptyset$

where $P_{FREE} = \{$ requesting processes can obtain CS without violating safety $\}$

Local Resource Allocation (LRA) [Cantarell *et al*, 03]

Generalization of Many Classical Problems

- Dining Philosophers
- Local Mutual Exclusion
- Drinking Philosophers
- Local Reader/Writer
- Local Group Mutual Exclusion
- ...

Local Resource Allocation (LRA) [Cantarell *et al*, 03]

LRA

Local Resource Allocation (LRA) [Cantarell *et al*, 03]

LRA

- **Safety:** Two neighbors p and q are concurrently executing their CS using X and Y, respectively, then $X \rightleftharpoons Y$.

Local Resource Allocation (LRA) [Cantarell *et al*, 03]

LRA

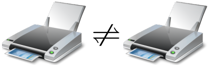
- **Safety:** Two neighbors p and q are concurrently executing their CS using X and Y, respectively, then $X \rightleftharpoons Y$.
- **Fairness:** A requesting process eventually enters its CS.

Local Resource Allocation (LRA) [Cantarell *et al*, 03]

LRA

- **Safety:** Two neighbors p and q are concurrently executing their CS using X and Y, respectively, then $X \rightleftharpoons Y$.
- **Fairness:** A requesting process eventually enters its CS.

Example: Local Mutual Exclusion



Local Resource Allocation (LRA) [Cantarell *et al*, 03]

LRA

- **Safety:** Two neighbors p and q are concurrently executing their CS using X and Y, respectively, then $X \rightleftharpoons Y$.
- **Fairness:** A requesting process eventually enters its CS.

Example: Local Mutual Exclusion

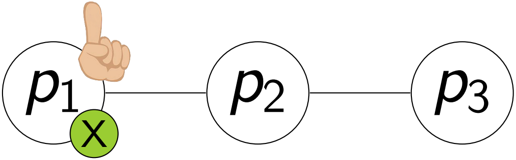
🖨 $\neq$ 🖨

Example: Local Reader-Writer Problem

👁 $\rightleftharpoons$ 👁   👁 $\neq$ ✏   ✏ $\neq$ ✏

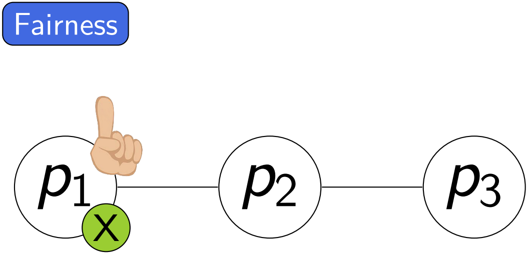
Impossibility Result: Maximal-Concurrency in LRA

Two resources: X $\neq$ Y



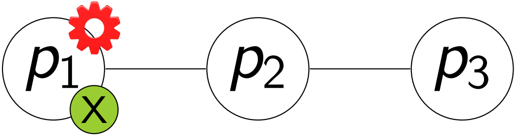
Impossibility Result: Maximal-Concurrency in LRA

Two resources: X $\neq$ Y



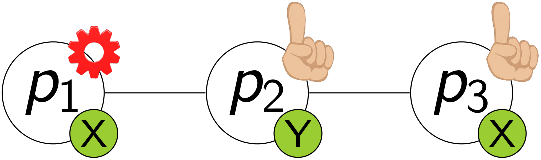
Impossibility Result: Maximal-Concurrency in LRA

Two resources: X $\neq$ Y



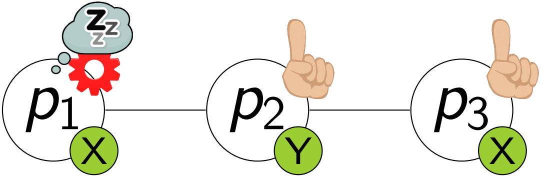
Impossibility Result: Maximal-Concurrency in LRA

Two resources: X $\neq$ Y



Impossibility Result: Maximal-Concurrency in LRA

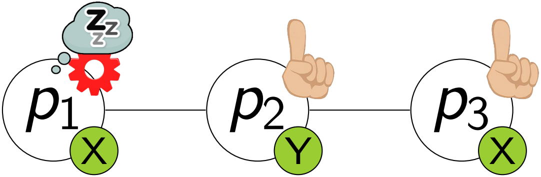
Two resources: X ≠ Y



Impossibility Result: Maximal-Concurrency in LRA

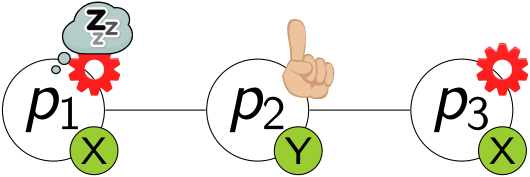
Two resources: X $\neq$ Y

Maximal-Concurrency



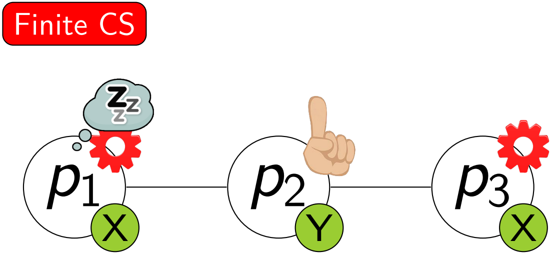
Impossibility Result: Maximal-Concurrency in LRA

Two resources: X $\neq$ Y



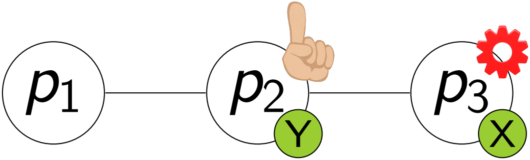
Impossibility Result: Maximal-Concurrency in LRA

Two resources: $\mathsf{X} \not\rightleftharpoons \mathsf{Y}$



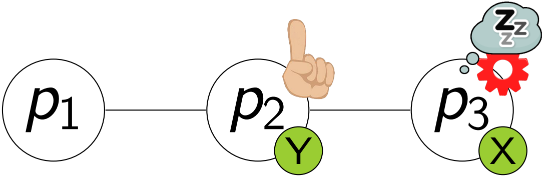
Impossibility Result: Maximal-Concurrency in LRA

Two resources: $X \neq Y$



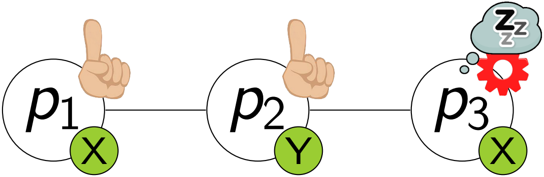
Impossibility Result: Maximal-Concurrency in LRA

Two resources: $X \neq Y$



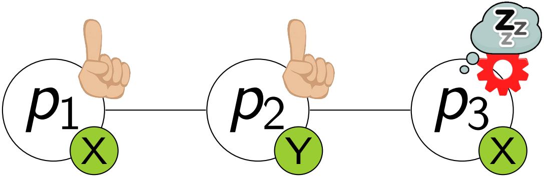
Impossibility Result: Maximal-Concurrency in LRA

Two resources: X $\neq$ Y



Impossibility Result: Maximal-Concurrency in LRA

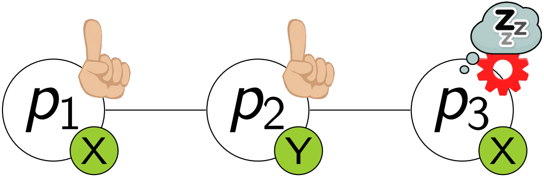
Two resources: $X \neq Y$



p_2 continuously requests but never enters its critical section.

Impossibility Result: Maximal-Concurrency in LRA

Two resources: $X \neq Y$



p_2 continuously requests but never enters its critical section.

Fairness property violated

(Strong) Partial Maximal-Concurrency

Weaker version of the maximal-concurrency

(Strong) Partial Maximal-Concurrency

Weaker version of the maximal-concurrency

Partial Maximal-Concurrency, Parameter: X

If CSs last a **long enough time**
then eventually $P_{FREE} \subseteq X$

$P_{FREE} = \{$ requesting processes can obtain CS without violating safety $\}$

(Strong) Partial Maximal-Concurrency

Weaker version of the maximal-concurrency

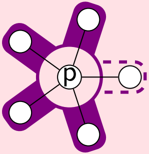
Partial Maximal-Concurrency, Parameter: X

If CSs last a **long enough time**
then eventually $P_{FREE} \subseteq X$

$P_{FREE} = \{$ requesting processes can obtain CS without violating safety $\}$

Strong Partial Maximal-Concurrency

Partial Maximal-Concurrency with
$X =$ neighbors of **a unique** process **minus one**.



Snap-Stabilizing LRA Algorithm

Requirements

Locally Shared Memory Model

- Locally shared variables
- Read/write atomicity
- Distributed weakly fair daemon

Snap-Stabilizing LRA Algorithm

Requirements

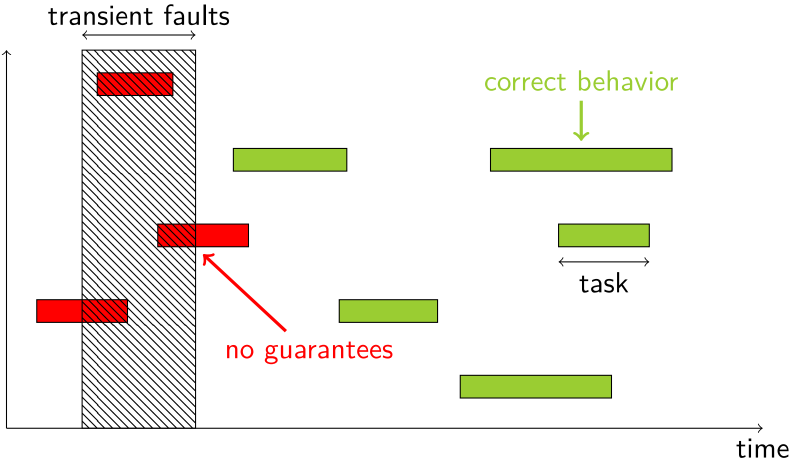
Locally Shared Memory Model

- Locally shared variables
- Read/write atomicity
- Distributed weakly fair daemon

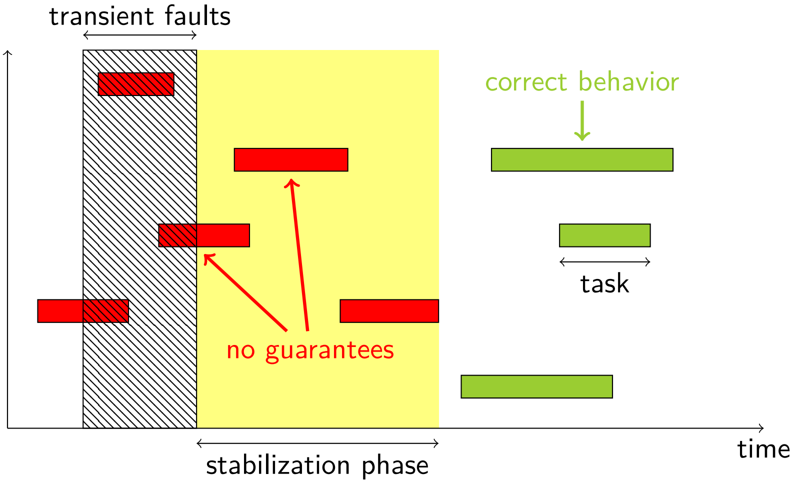
Network

- Connected
- Bidirectional
- Identified

Snap-Stabilization [Bui *et al*, 07]



Self-Stabilization [Dijkstra, 74]



Snap-Stabilizing LRA Algorithm

Guarantees

- Snap-stabilizing
- Strongly partially maximal-concurrent

Snap-Stabilizing LRA Algorithm

Guarantees

- Snap-stabilizing
- Strongly partially maximal-concurrent

Ideas

Snap-Stabilizing LRA Algorithm

Guarantees

- Snap-stabilizing
- Strongly partially maximal-concurrent

Ideas

- ID-based priority

Snap-Stabilizing LRA Algorithm

Guarantees

- Snap-stabilizing
- Strongly partially maximal-concurrent

Ideas

- ID-based priority
- Locked state : 🔒

Snap-Stabilizing LRA Algorithm

Guarantees

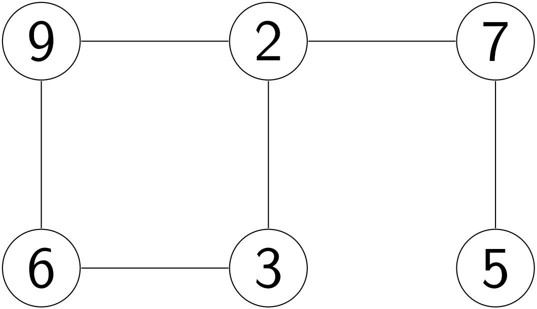
- Snap-stabilizing
- Strongly partially maximal-concurrent

Ideas

- ID-based priority
- Locked state : 🔒
- (Self-stabilizing) Token : ★

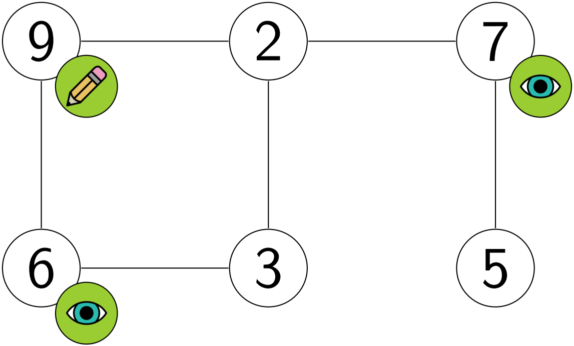
Snap-Stabilizing LRA Algorithm

Example on the Local Reader-Writer Problem



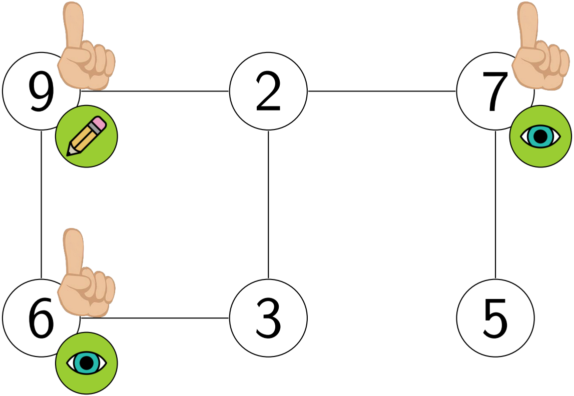
Snap-Stabilizing LRA Algorithm

Example on the Local Reader-Writer Problem



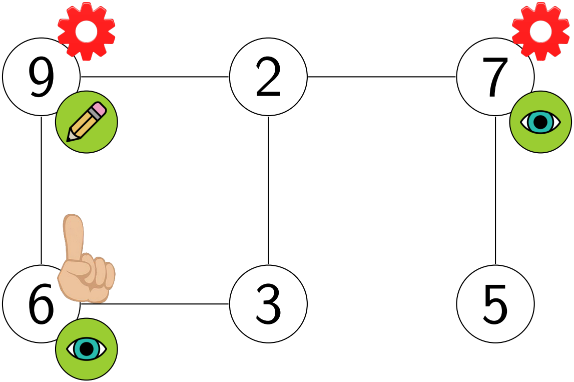
Snap-Stabilizing LRA Algorithm

Example on the Local Reader-Writer Problem



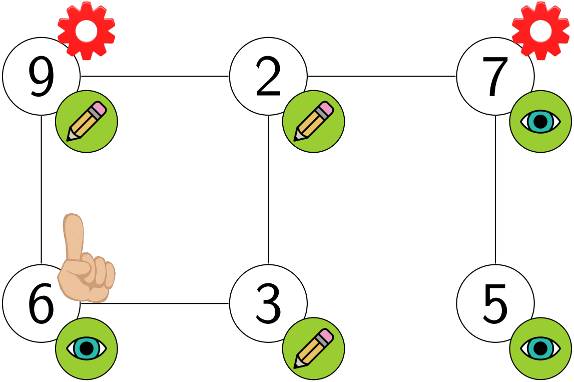
Snap-Stabilizing LRA Algorithm

Example on the Local Reader-Writer Problem



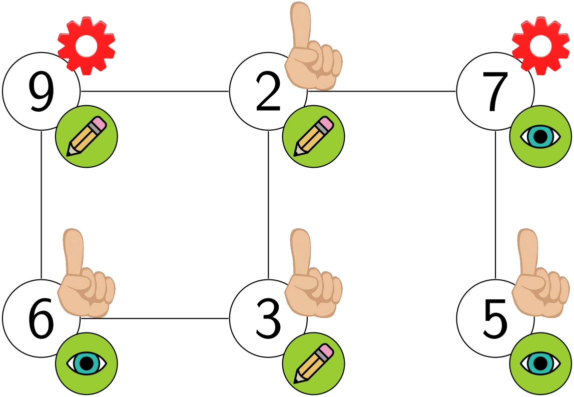
Snap-Stabilizing LRA Algorithm

Example on the Local Reader-Writer Problem



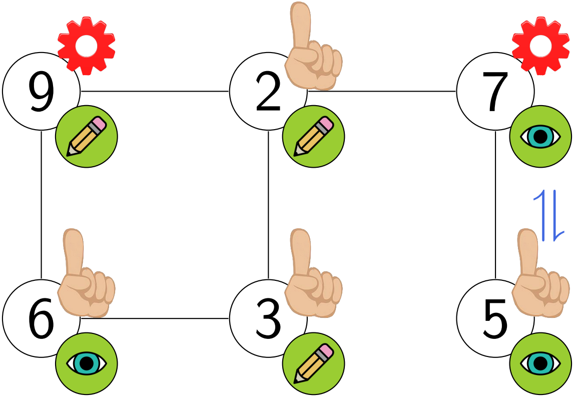
Snap-Stabilizing LRA Algorithm

Example on the Local Reader-Writer Problem



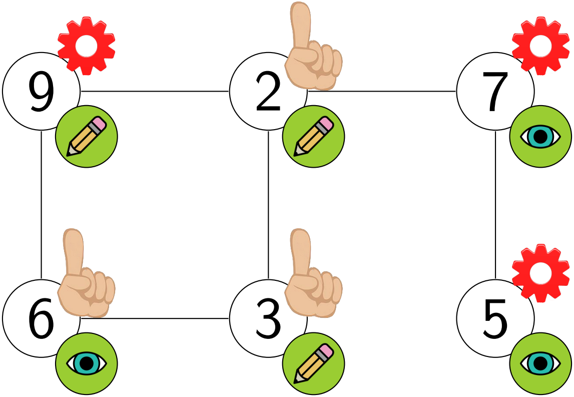
Snap-Stabilizing LRA Algorithm

Example on the Local Reader-Writer Problem



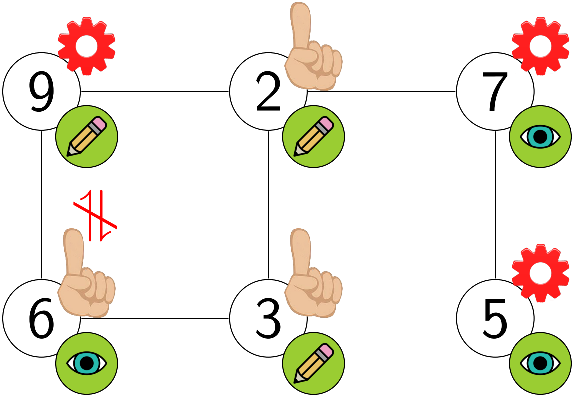
Snap-Stabilizing LRA Algorithm

Example on the Local Reader-Writer Problem



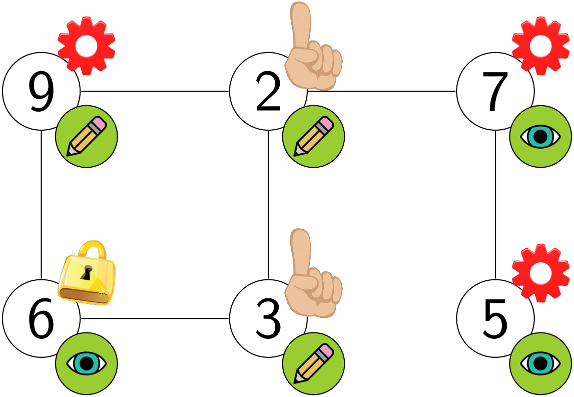
Snap-Stabilizing LRA Algorithm

Example on the Local Reader-Writer Problem



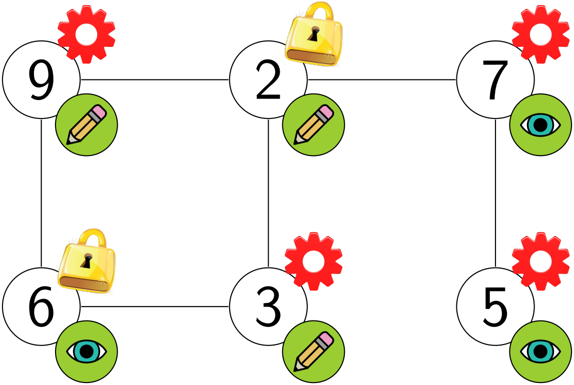
Snap-Stabilizing LRA Algorithm

Example on the Local Reader-Writer Problem



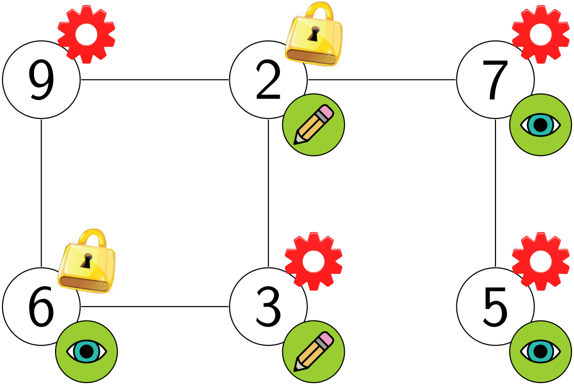
Snap-Stabilizing LRA Algorithm

Example on the Local Reader-Writer Problem



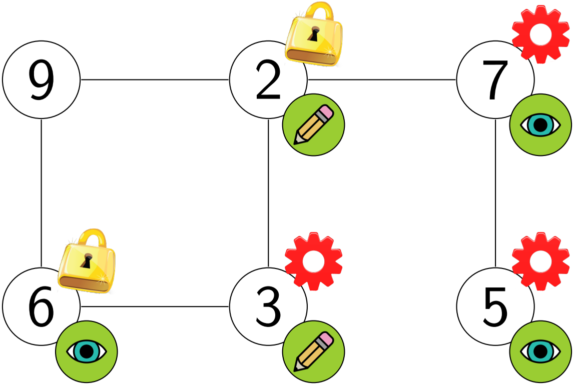
Snap-Stabilizing LRA Algorithm

Example on the Local Reader-Writer Problem



Snap-Stabilizing LRA Algorithm

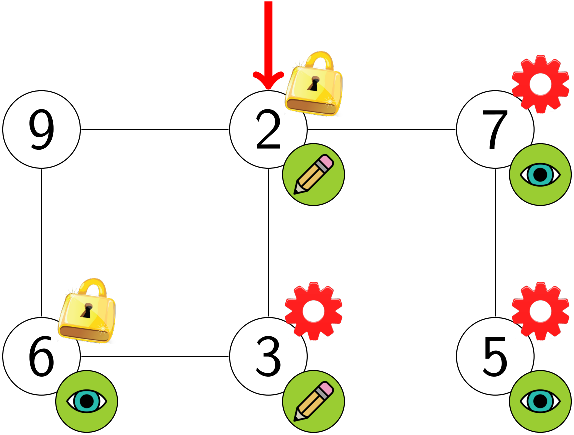
Example on the Local Reader-Writer Problem



Snap-Stabilizing LRA Algorithm

Example on the Local Reader-Writer Problem

Local minimum ⇒ may never enter its CS



Snap-Stabilizing LRA Algorithm

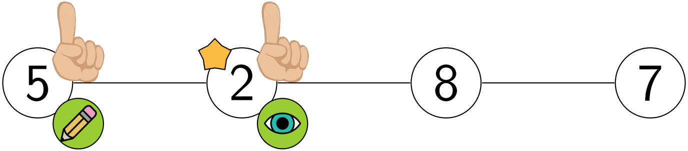
Self-Stabilizing Token Circulation

- **Safety:** There eventually is a unique token holder.
- **Liveness:** A process p holds a token infinitely often.

Snap-Stabilizing LRA Algorithm

Self-Stabilizing Token Circulation

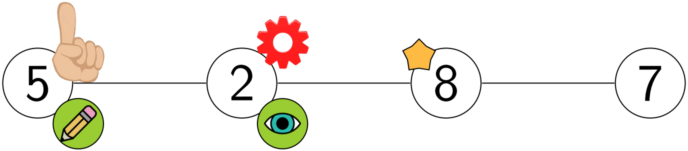
- **Safety:** There eventually is a unique token holder.
- **Liveness:** A process *p* holds a token infinitely often.



Snap-Stabilizing LRA Algorithm

Self-Stabilizing Token Circulation

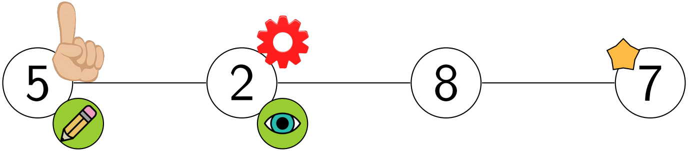
- **Safety:** There eventually is a unique token holder.
- **Liveness:** A process p holds a token infinitely often.



Snap-Stabilizing LRA Algorithm

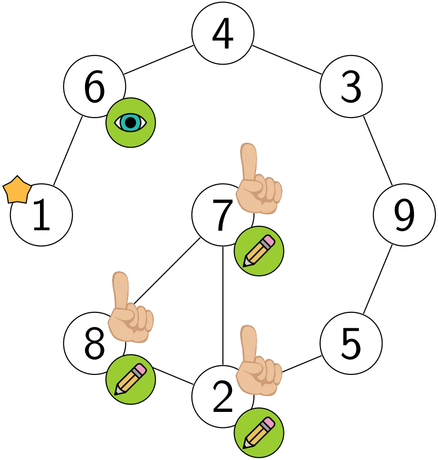
Self-Stabilizing Token Circulation

- **Safety:** There eventually is a unique token holder.
- **Liveness:** A process *p* holds a token infinitely often.



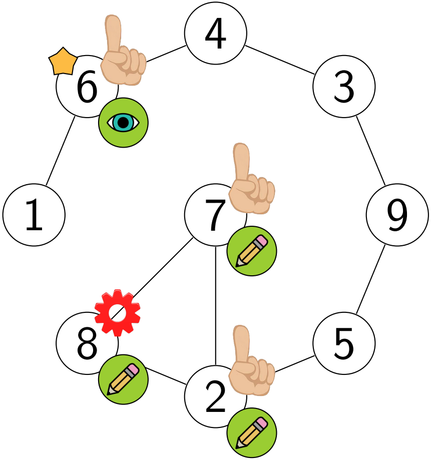
Snap-Stabilizing LRA Algorithm: Fairness

Example on the Local Reader-Writer Problem



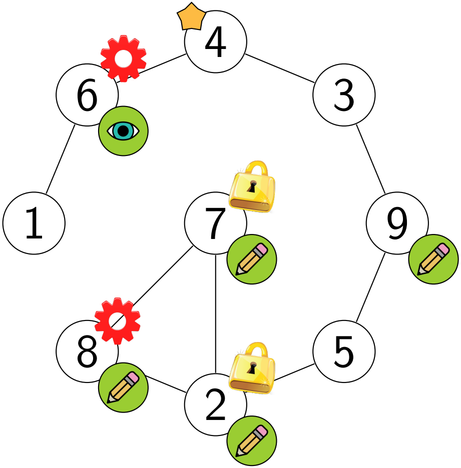
Snap-Stabilizing LRA Algorithm: Fairness

Example on the Local Reader-Writer Problem



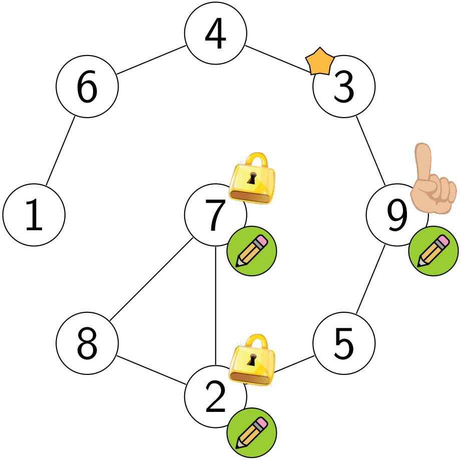
Snap-Stabilizing LRA Algorithm: Fairness

Example on the Local Reader-Writer Problem



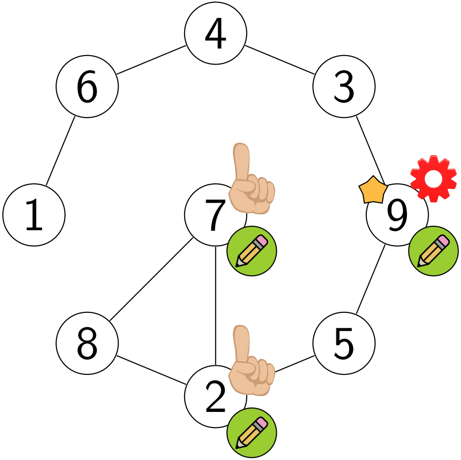
Snap-Stabilizing LRA Algorithm: Fairness

Example on the Local Reader-Writer Problem



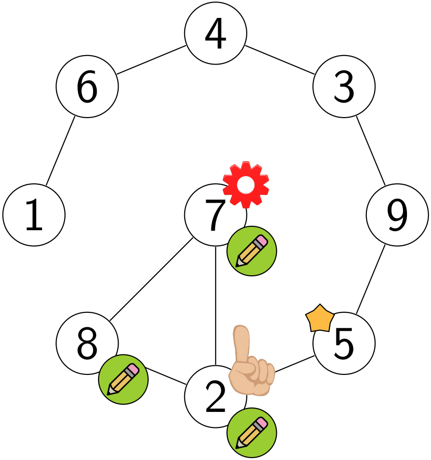
Snap-Stabilizing LRA Algorithm: Fairness

Example on the Local Reader-Writer Problem



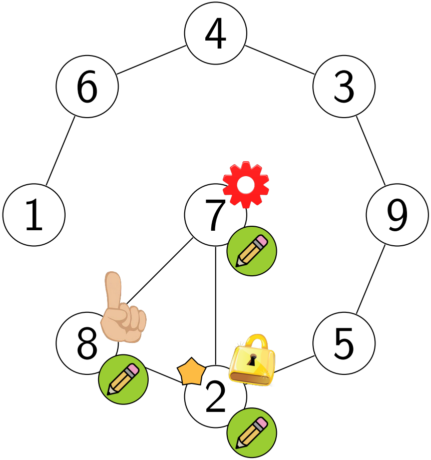
Snap-Stabilizing LRA Algorithm: Fairness

Example on the Local Reader-Writer Problem



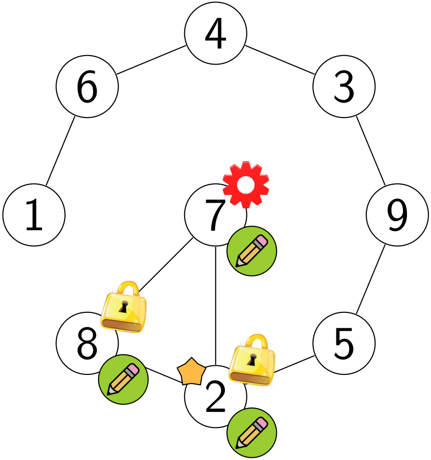
Snap-Stabilizing LRA Algorithm: Fairness

Example on the Local Reader-Writer Problem



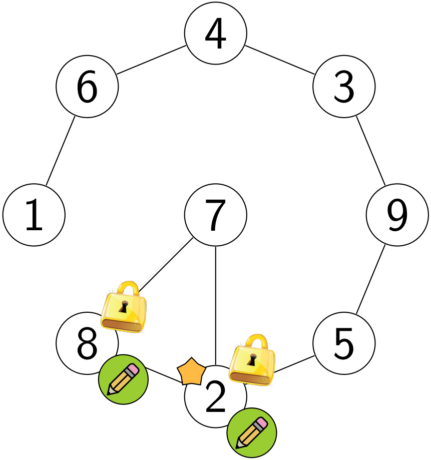
Snap-Stabilizing LRA Algorithm: Fairness

Example on the Local Reader-Writer Problem



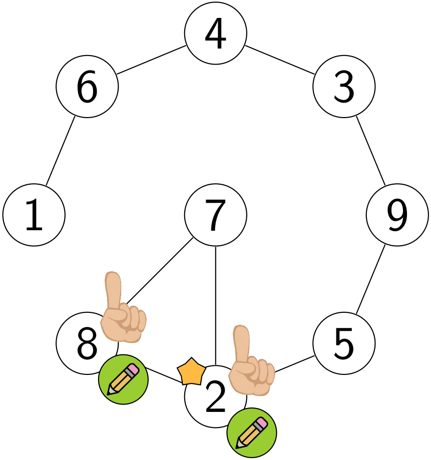
Snap-Stabilizing LRA Algorithm: Fairness

Example on the Local Reader-Writer Problem



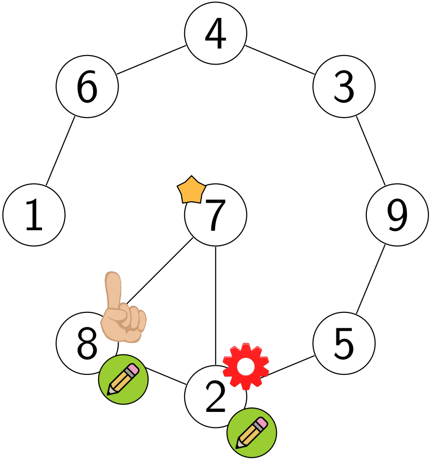
Snap-Stabilizing LRA Algorithm: Fairness

Example on the Local Reader-Writer Problem



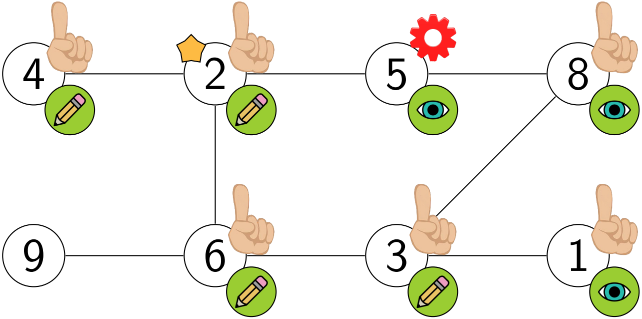
Snap-Stabilizing LRA Algorithm: Fairness

Example on the Local Reader-Writer Problem



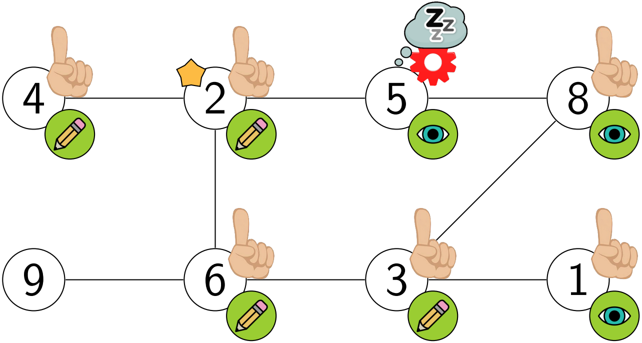
Snap-Stabilizing LRA Algorithm: Strong Partial Maximal-Concurrency

A **worst case** in the Local Reader-Writer Problem



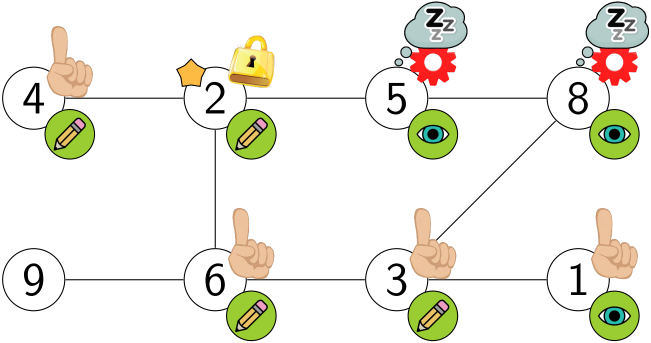
Snap-Stabilizing LRA Algorithm: Strong Partial Maximal-Concurrency

A **worst case** in the Local Reader-Writer Problem



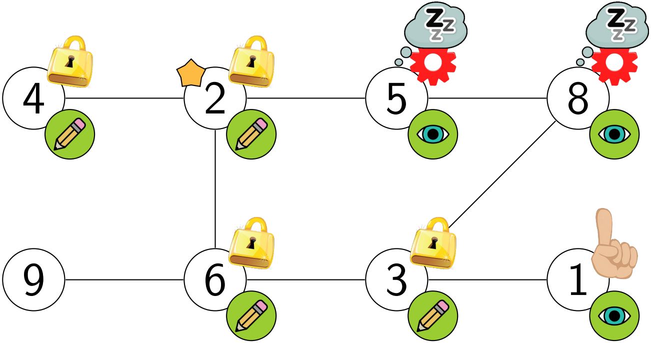
Snap-Stabilizing LRA Algorithm: Strong Partial Maximal-Concurrency

A **worst case** in the Local Reader-Writer Problem



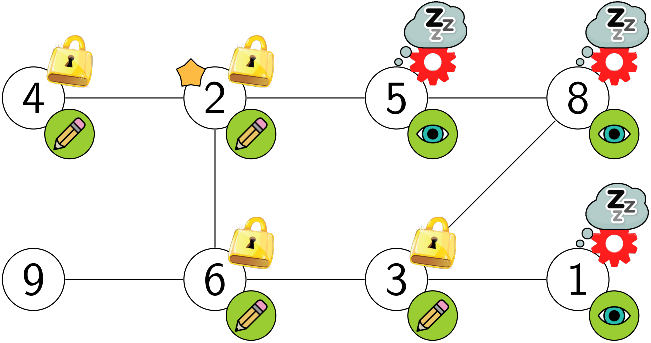
Snap-Stabilizing LRA Algorithm: Strong Partial Maximal-Concurrency

A **worst case** in the Local Reader-Writer Problem



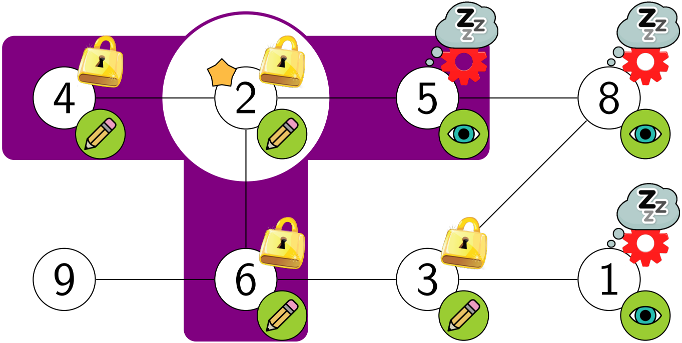
Snap-Stabilizing LRA Algorithm: Strong Partial Maximal-Concurrency

A **worst case** in the Local Reader-Writer Problem



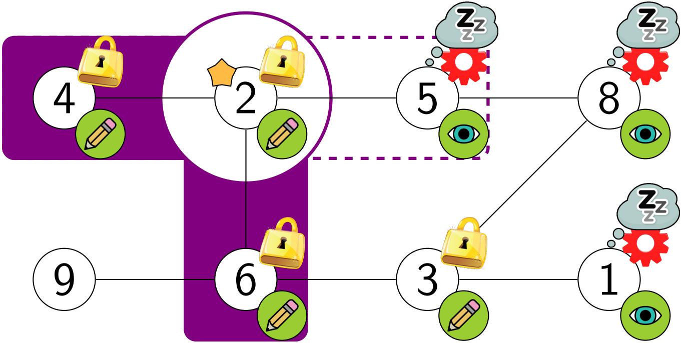
Snap-Stabilizing LRA Algorithm: Strong Partial Maximal-Concurrency

A **worst case** in the Local Reader-Writer Problem



Snap-Stabilizing LRA Algorithm: Strong Partial Maximal-Concurrency

A **worst case** in the Local Reader-Writer Problem



Conclusion

Contributions

- Definition of the **maximal-concurrency**
- Proof of **impossibility** of maximal-concurrency in LRA
- Definition of the **(strong) partial maximal-concurrency**
- Design and proof of a snap-stabilizing strongly partially maximal-concurrent LRA algorithm

Conclusion

Contributions

- Definition of the **maximal-concurrency**
- Proof of **impossibility** of maximal-concurrency in LRA
- Definition of the **(strong) partial maximal-concurrency**
- Design and proof of a snap-stabilizing strongly partially maximal-concurrent LRA algorithm

Perspectives

Define the class of resource allocation problems where maximal-concurrency/strong partial-maximal concurrency can be achieved.

Thank you for your attention.

Do you have any questions ?