

Self-Stabilizing Atomicity Refinement Allowing Neighborhood Concurrency

Franck Petit

LME Problem

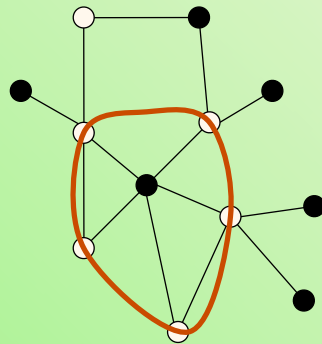
- ❖ Safety

If a process is executing its critical section, then none of its neighbors is executing its critical section simultaneously.

- ❖ Liveness

If a process requests to enter its critical section, then it will eventually execute its critical section.

LME Algorithm

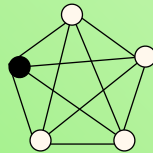


Concurrency

Best (expected) case



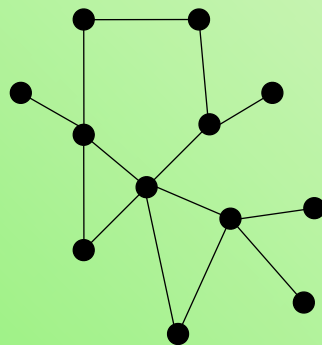
Worst case



i.e., Mutual Exclusion

Question

Could we increase concurrency?



Answer

Yes!

Local Resource Allocation !

Local Resource Allocation

❖ Statement

Neighboring Processes share m resources with compatibility criteria

❖ Compatibility

Two resources X and Y are said to be compatible, denoted $X \leftrightarrow Y$, if two neighboring processes can access X and Y concurrently

Local Resource Allocation

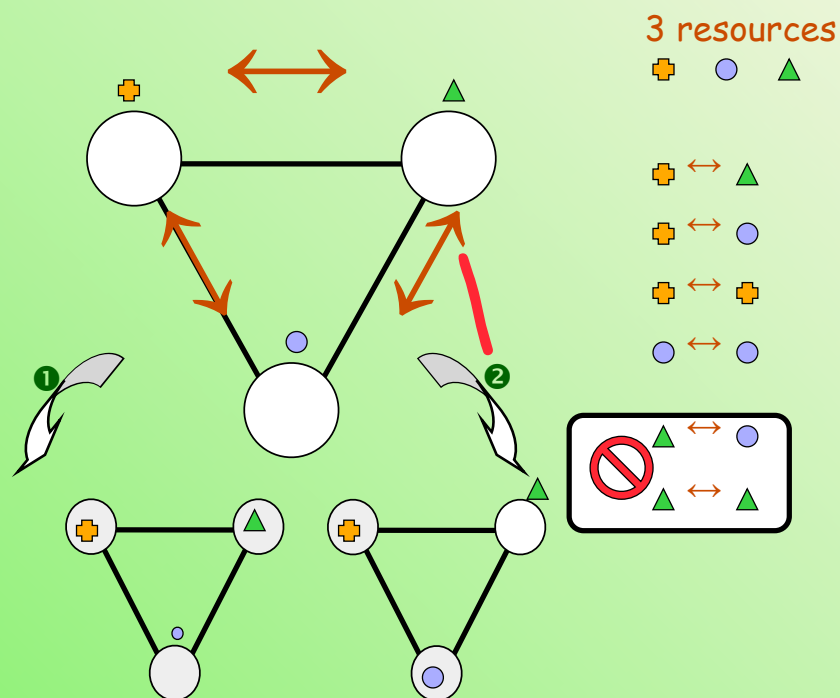
❖ Safety

If two neighboring processes p and q are executing their critical section concurrently using X and Y , respectively, then $X \leftrightarrow Y$.

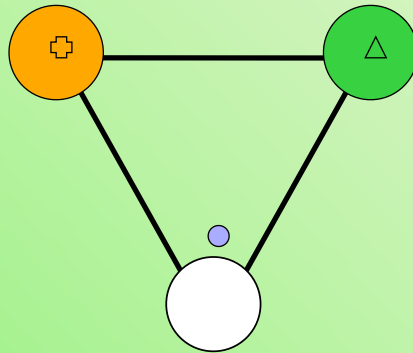
❖ Liveness

If a process requests to enter its critical section, then it will eventually execute its critical section.

Local Resource Allocation



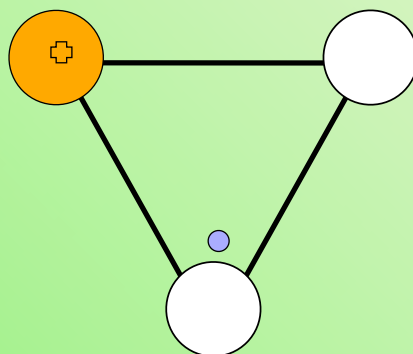
Local Resource Allocation



3 resources



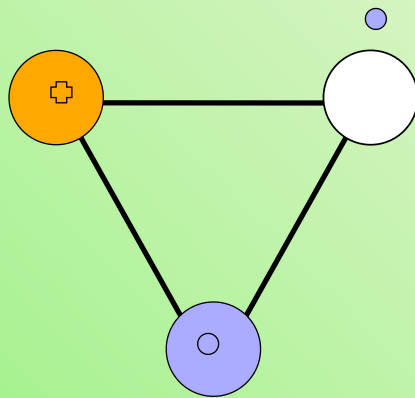
Local Resource Allocation



3 resources



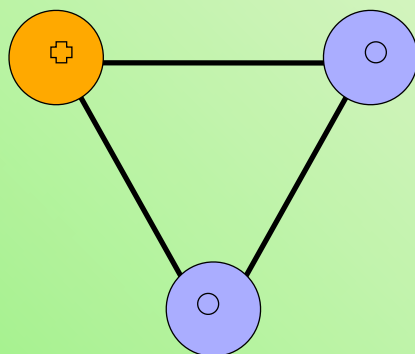
Local Resource Allocation



3 resources



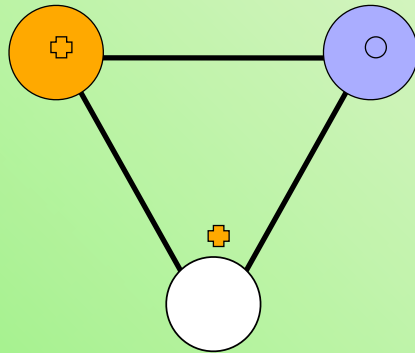
Local Resource Allocation



3 resources



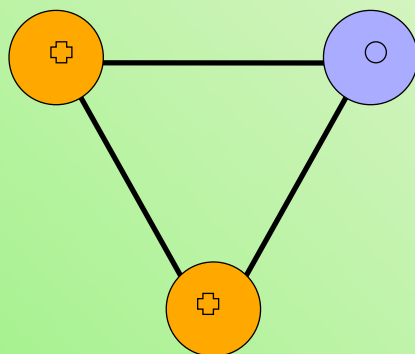
Local Resource Allocation



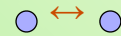
3 resources



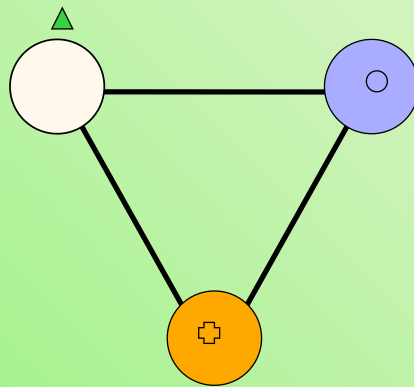
Local Resource Allocation



3 resources



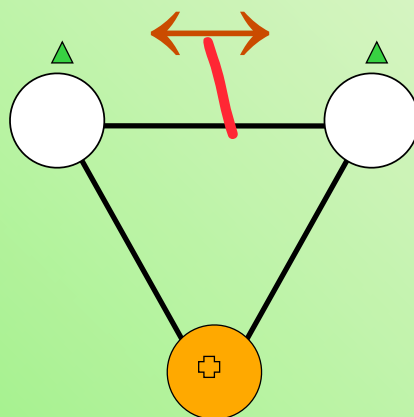
Local Resource Allocation



3 resources



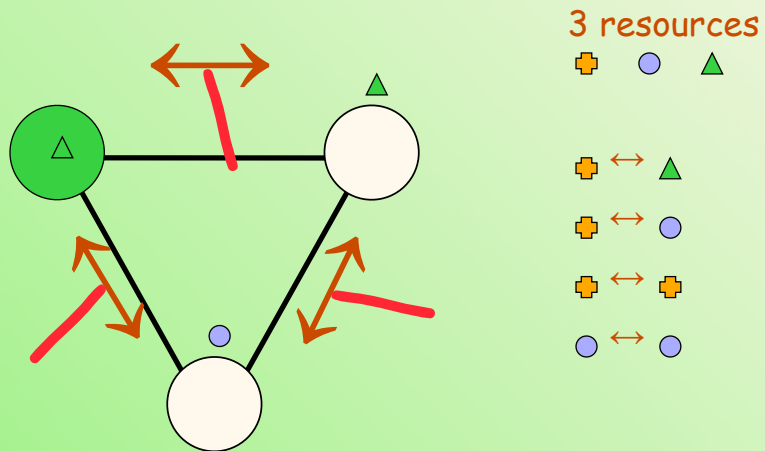
Local Resource Allocation



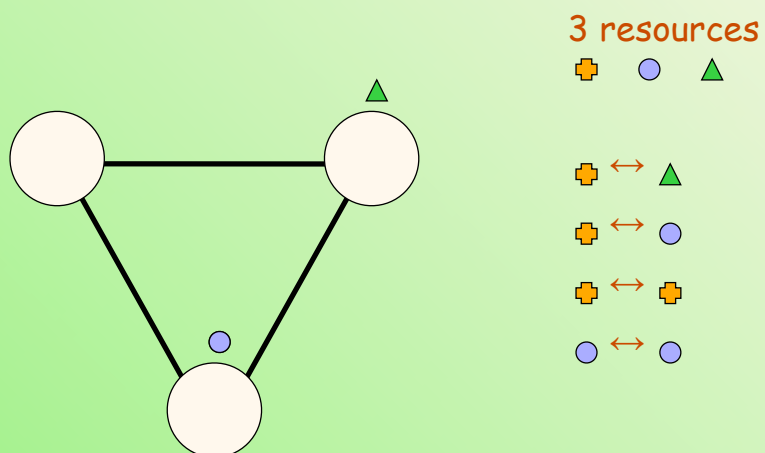
3 resources



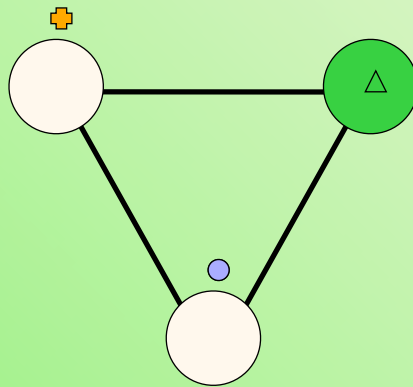
Local Resource Allocation



Local Resource Allocation



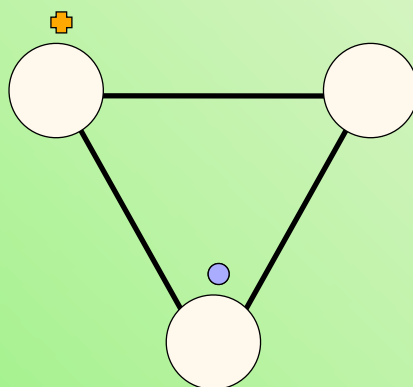
Local Resource Allocation



3 resources



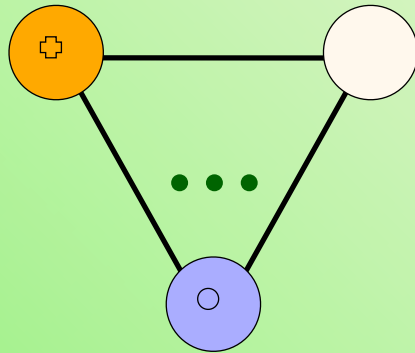
Local Resource Allocation



3 resources



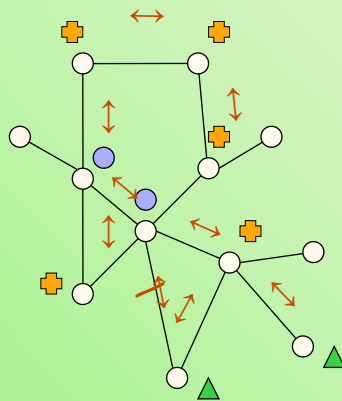
Local Resource Allocation



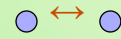
3 resources



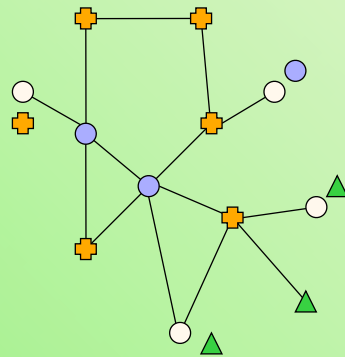
Local Resource Allocation



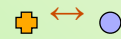
3 resources



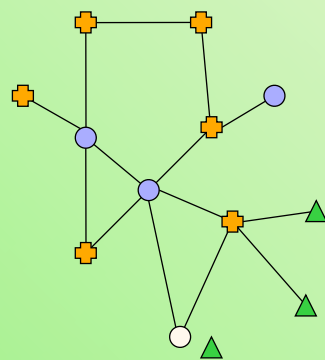
Local Resource Allocation



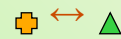
3 resources



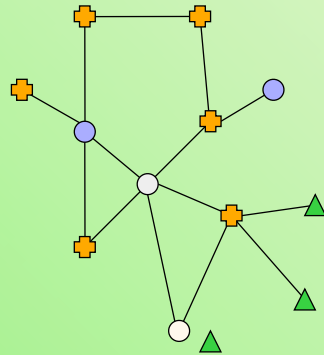
Local Resource Allocation



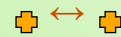
3 resources



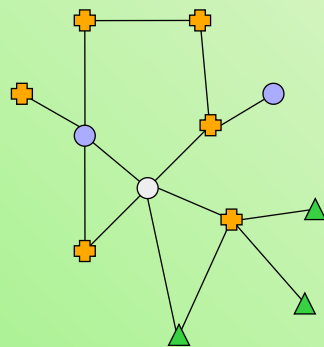
Local Resource Allocation



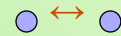
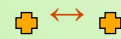
3 resources



Local Resource Allocation



3 resources



Relation With Other Resource Allocation Problems?

Related Resource Sharing Problem

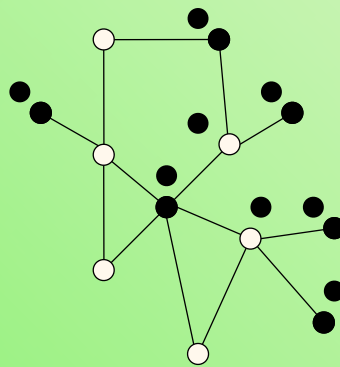
- ❖ Dining Philosophers [Dijkstra, EWD 625, 1978]

(Local Mutual Exclusion)

One Resource Shared in Exclusive Access by Neighbor Processes

Concurrency = 1 process in the neighborhood

LRA solves LME



1 resource only



Related Resource Sharing Problem

- ❖ Drinking Philosophers [Chandy and Misra, TOPLAS 1984]

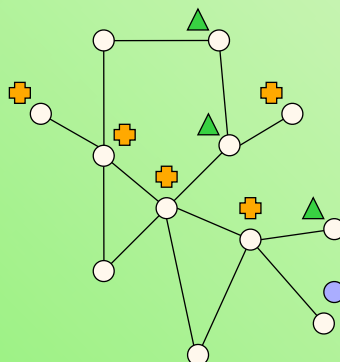
Several Resources Shared in Exclusive Access by Neighbor Processes

Concurrency = 1 process in the neighborhood for each resource

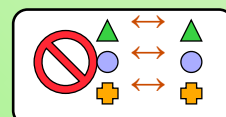
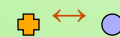
Neighbors are allowed to use resources simultaneously provided that

they are using different resources

LRA solves Drinking Philosophers Problem



3 resources



Related Resource Sharing Problem

❖ Drinking Philosophers [Chandy and Misra, TOPLAS 1984]

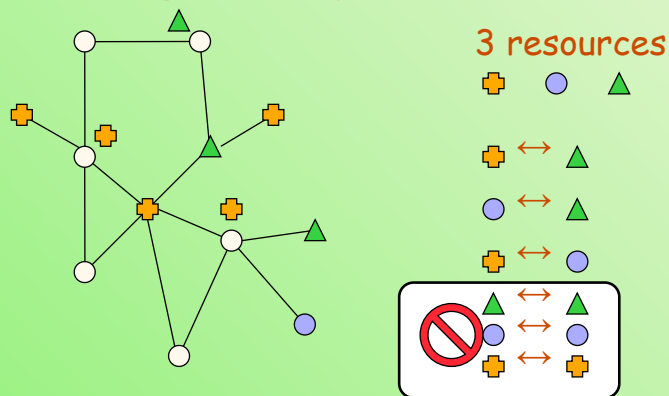
Several Resources Shared in Exclusive Access by Neighbor Processes

Concurrency = 1 process in the neighborhood for each resource

Neighbors are allowed to use resources simultaneously provided that

they are using different resources

LRA solves Drinking Philosophers Problem



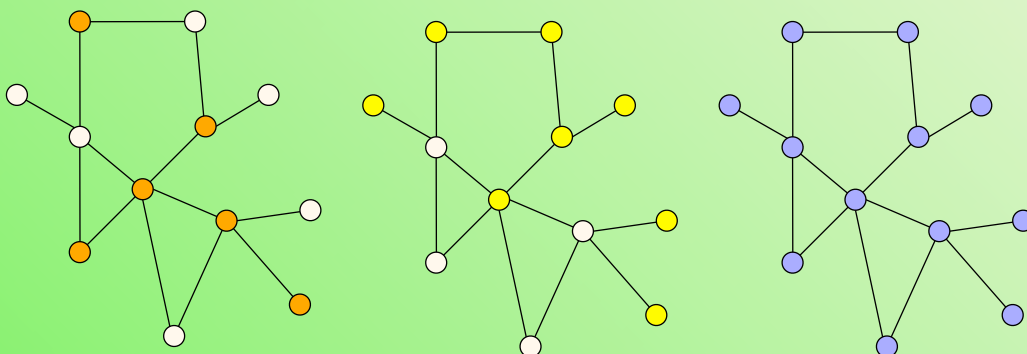
Related Resource Sharing Problem

❖ Group Mutual Exclusion [Joung, DISC 1998]

M Exclusive Resources Shared in Concurrent Access

Concurrency = n processes

... but exclusive resource access



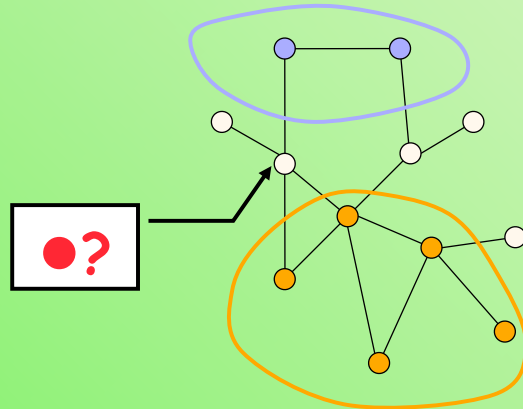
Related Resource Sharing Problem

❖ Local Group Mutual Exclusion

M Local Exclusive Resources Shared in Concurrent Access

Concurrency = $\Delta+1$ processes in the neighborhood

... but exclusive resource access



Related Resource Sharing Problem

❖ Local Group Mutual Exclusion

M Local Exclusive Resources Shared in Concurrent Access

Concurrency = $\Delta+1$ processes in the neighborhood

... but exclusive resource access

Does LRA solves the (Local) GME Problem?

Yes!

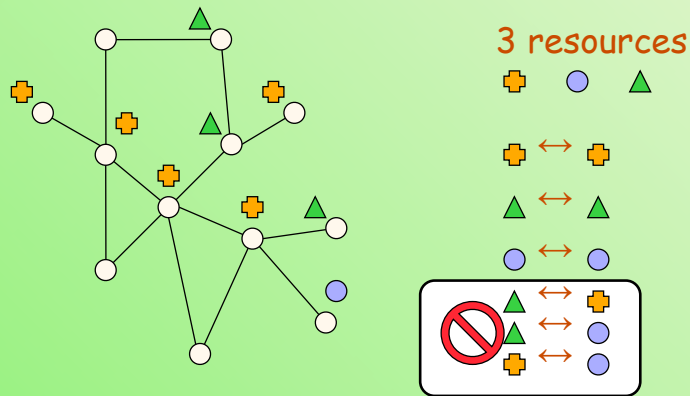
Related Resource Sharing Problem

❖ Local Group Mutual Exclusion

M Local Exclusive Resources Shared in Concurrent Access

Concurrency = $\Delta+1$ processes in the neighborhood

... but exclusive resource access



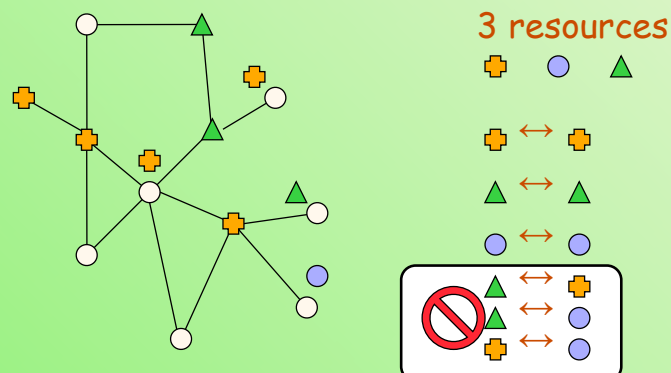
Related Resource Sharing Problem

❖ Local Group Mutual Exclusion

M Local Exclusive Resources Shared in Concurrent Access

Concurrency = $\Delta+1$ processes in the neighborhood

... but exclusive resource access



Related Resource Sharing Problem

LRA is a generalisation of

- ❖ Local Mutual Exclusion
- ❖ Dining and Drinking Philosophers
- ❖ Local Group Mutual Exclusion
(and then Local Readers/Writers)

An SS LRA Algorithm

Each process p maintains

- ❖ A compatibility graph $CG = (R, C)$
 - R = Resources accessed by p
 - C = Compatibility Edges
- ❖ Request $\in R \cup \{\perp\}$ (upper layer)
- ❖ **Grant**: Boolean (LRA layer)
- ❖ c : integer, a Lamport's timestamp (LRA layer)

An SS LRA Algorithm

Entry Section

- ❖ Each time a process p requests to access the critical section with Resource X (Upper Layer):
 - p sets Request to X (Upper Layer)
 - p updates its timestamp c (LRA Layer)
- ❖ Then, p waits for one of the two following conditions to become true (Grant to true by the LRA Layer):
 - p has the maximum timestamp in its neighborhood, or
 - Request (X) is compatible with its neighbors

Exit Section

- ❖ p resets Request to \perp , Grant to false, and c to 0 (LRA Layer)

The Upper Layer Algorithm

•
•

REQ : $(\text{Request} = \perp) \wedge \neg \text{Grant} \wedge (X \in R \text{ requested}) \rightarrow \text{Request} := X$

CS : $(\text{Request} \neq \perp) \wedge \text{Grant} \rightarrow \langle\langle \text{CS} \rangle\rangle; \text{Request} := \perp$

•
•

No request \Rightarrow No action \Rightarrow Silent Algorithm

The LRA Layer Algorithm

$$p \prec q \equiv (c < c_q) \vee ((c = c_q) \wedge (p < q))$$

$$\text{Ready} \equiv (\forall q \in N: (c_q > 0) \Rightarrow ((p \prec q) \vee ((\text{Request}_q \neq \perp) \Rightarrow (\text{Request} \leftrightarrow \text{Request}_q))))$$

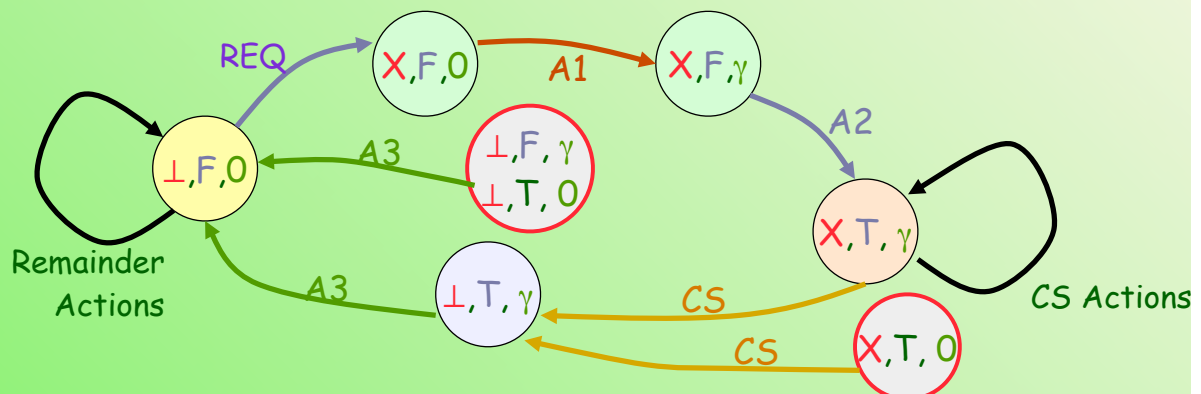
$$A1: (\text{Request} \neq \perp) \wedge \neg \text{Grant} \wedge (c = 0) \rightarrow c := \max \{ c_q \in N \} + 1$$

$$A2: (\text{Request} \neq \perp) \wedge \neg \text{Grant} \wedge (c > 0) \wedge \text{Ready} \rightarrow \text{Grant} := \text{true}$$

$$A3: (\text{Request} = \perp) \wedge (\text{Grant} \vee (c > 0)) \rightarrow \text{Grant} := \text{false}; c := 0$$

Self-Stabilizing LRA Scheme

α, β, γ $\alpha = \text{Request}, \beta = \text{Grant}, \gamma = c$



No request \Rightarrow No action \Rightarrow Silent Algorithm

Conclusion

- ❖ Local Resource Allocation (LRA)
- ❖ Self-stabilizing LRA Algorithm
- ❖ Self-Stabilizing Atomicity Transformer
- ❖ R/W Atomicity Model
- ❖ Silent-Preserving