

Relationships between communication models based on registers for fault-tolerant distributed computing on networks

Colette Johnen, Lisa Higham
Univ. Bordeaux, CNRS, LaBRI, UMR 5800

Semantics of a register

[Lamport 86]

Single-Writer Register (SR)- def

- A register is a memory cell on which two types of operations are possible **READ** and **WRITE**
- On a Single-Writer register, only one process can do the **WRITE** operation
- On a register R , **READ** and **WRITE** operation are not atomic, they take some time

A **READ** operation on a register R may overlap several **WRITE** operations on R

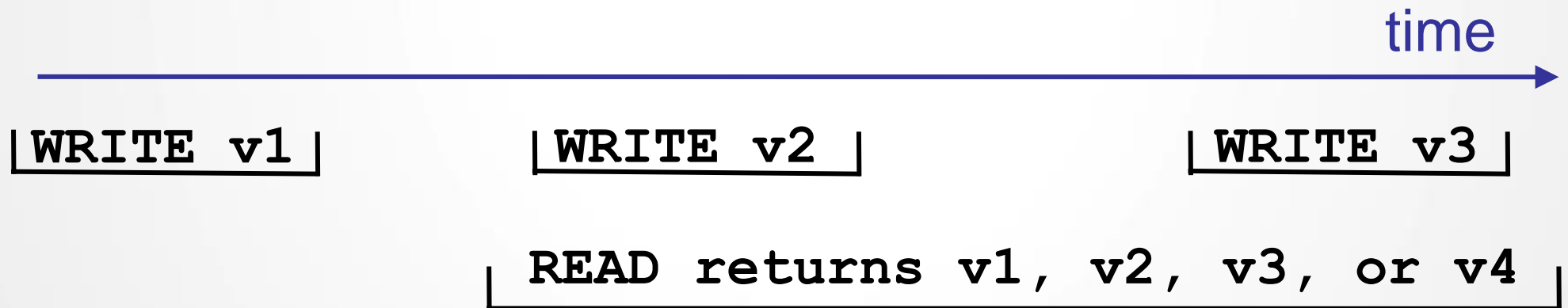
Single-Writer register - semantic

- On R , a **READ** operation that does not overlap any **WRITE** operation returns the most recent preceding written value ($v1$) in R



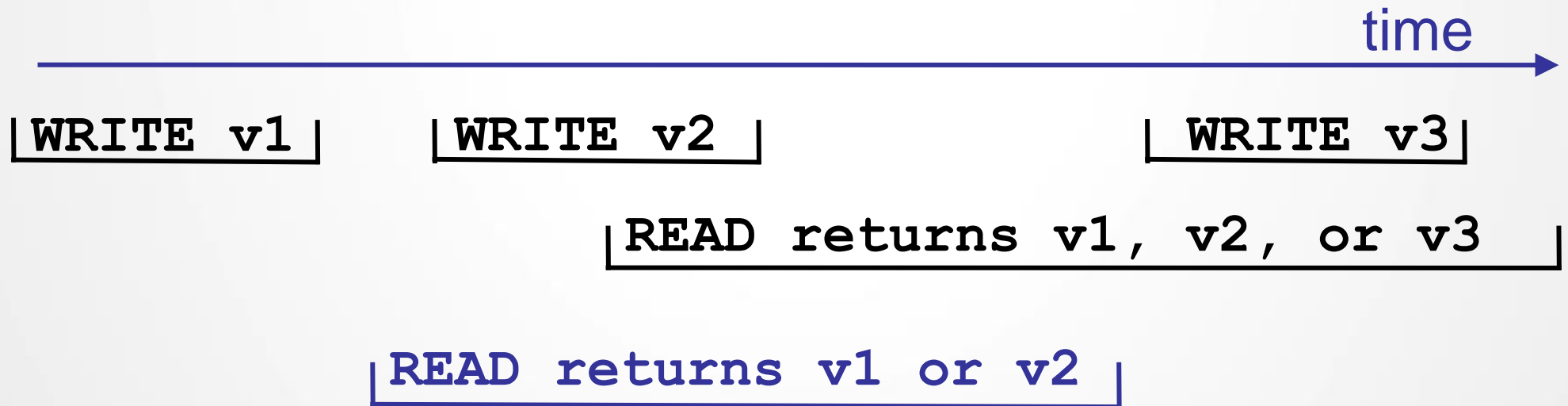
Safe register [Lamport 86]

- On R , a **READ** operation that does overlap a **WRITE** operation may return any value



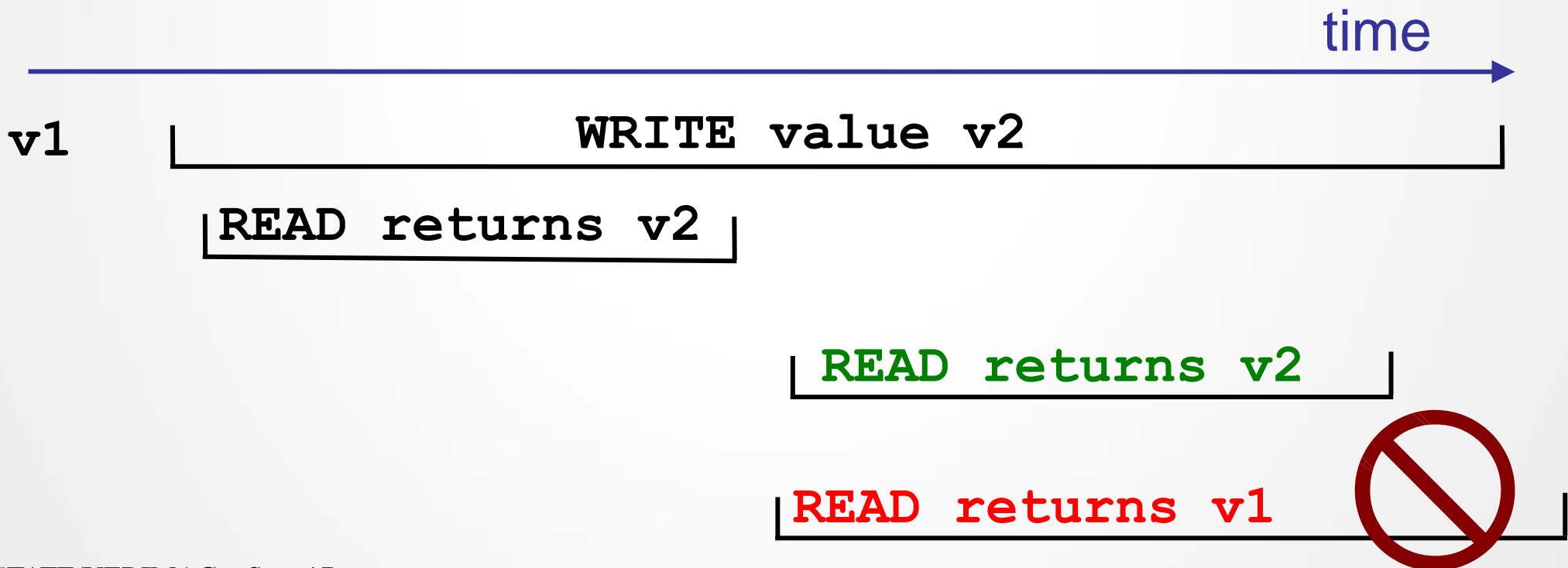
Regular and Atomic register [Lamport 86]

- On R , a **READ** operation that does overlap a **WRITE** operation returns the most recent preceding written value or any value written during overlapping **WRITE** operations



Atomic register [Lamport 86]

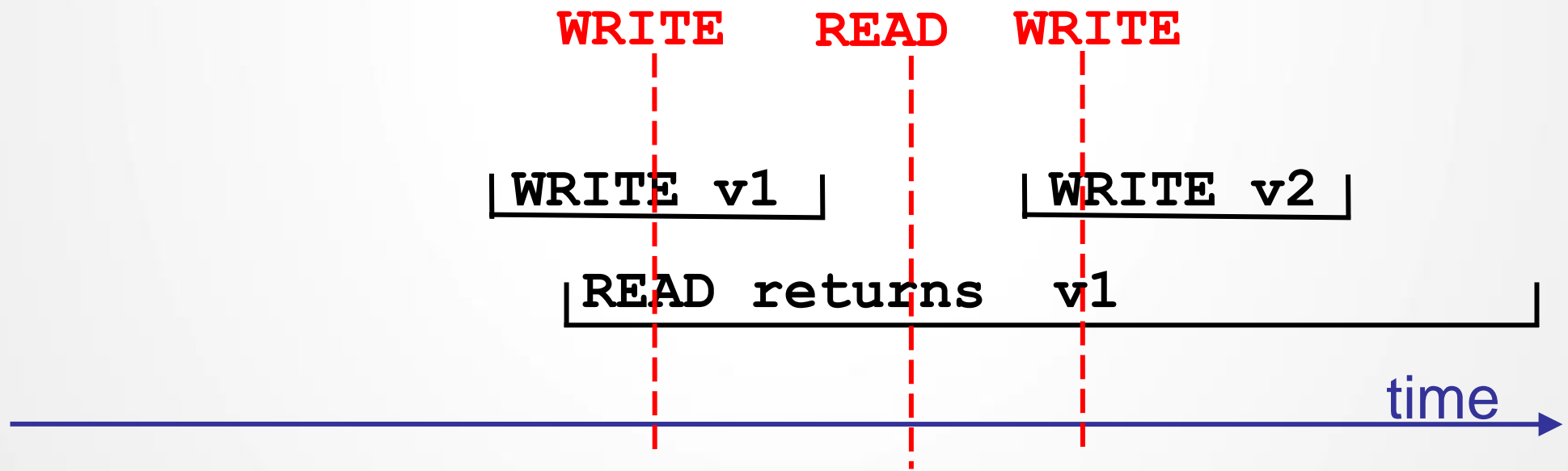
- On R , if a **READ** operation returns the value written during the overlapping **WRITE** operation then any subsequence **READ** cannot return the most recent preceding written value ($v1$)



Property of atomic registers

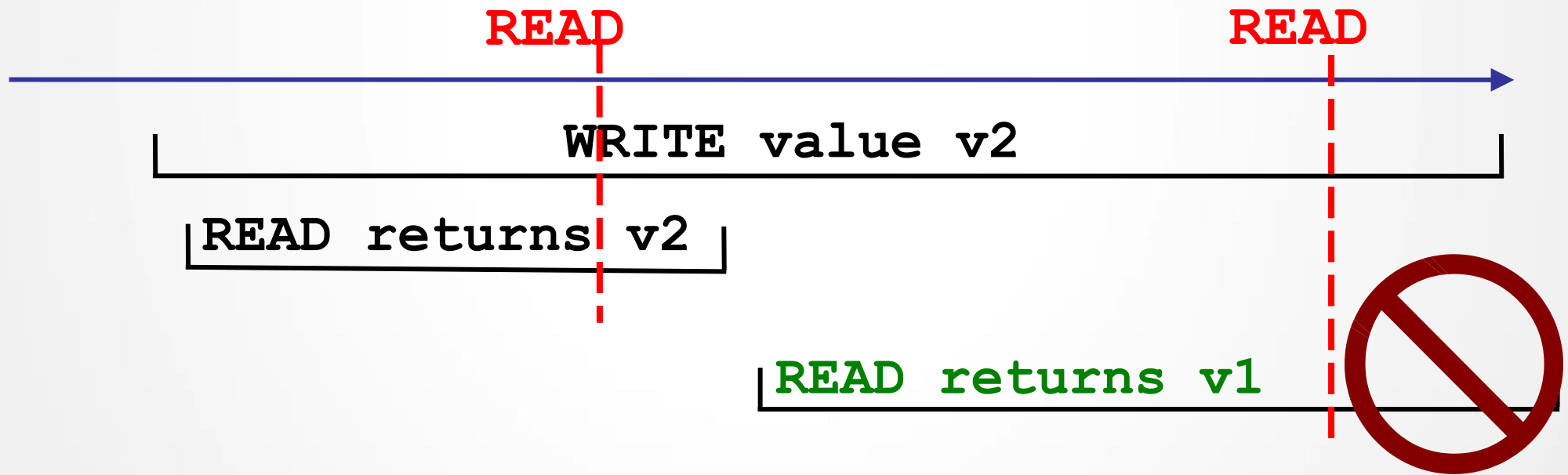
- A sequence of operations on an atomic register is [linearizable](#) [Herlihy, Wing 90]

Each operation appears to happen instantaneously at some point during its execution



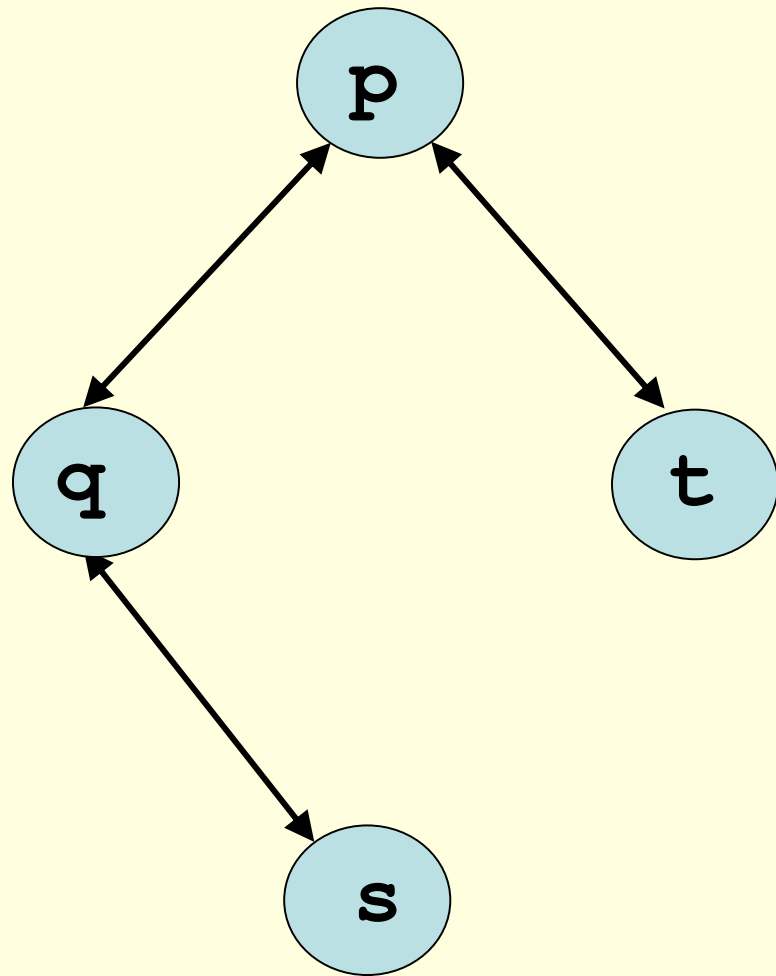
Regular register

- A sequence of operations on an regular register may not linearizable



Communication Model on networks

Distributed computing on networks

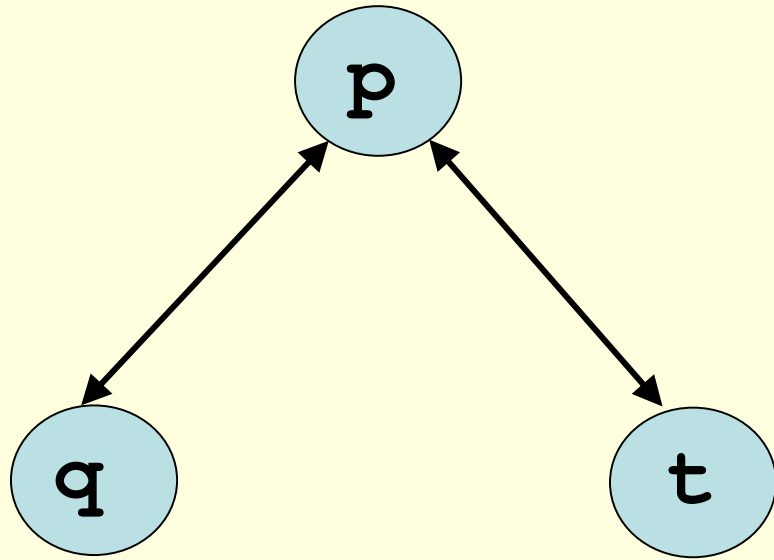


Topology G

A processor can only communicate with its neighbors

Ex: p can only communicate with q and t

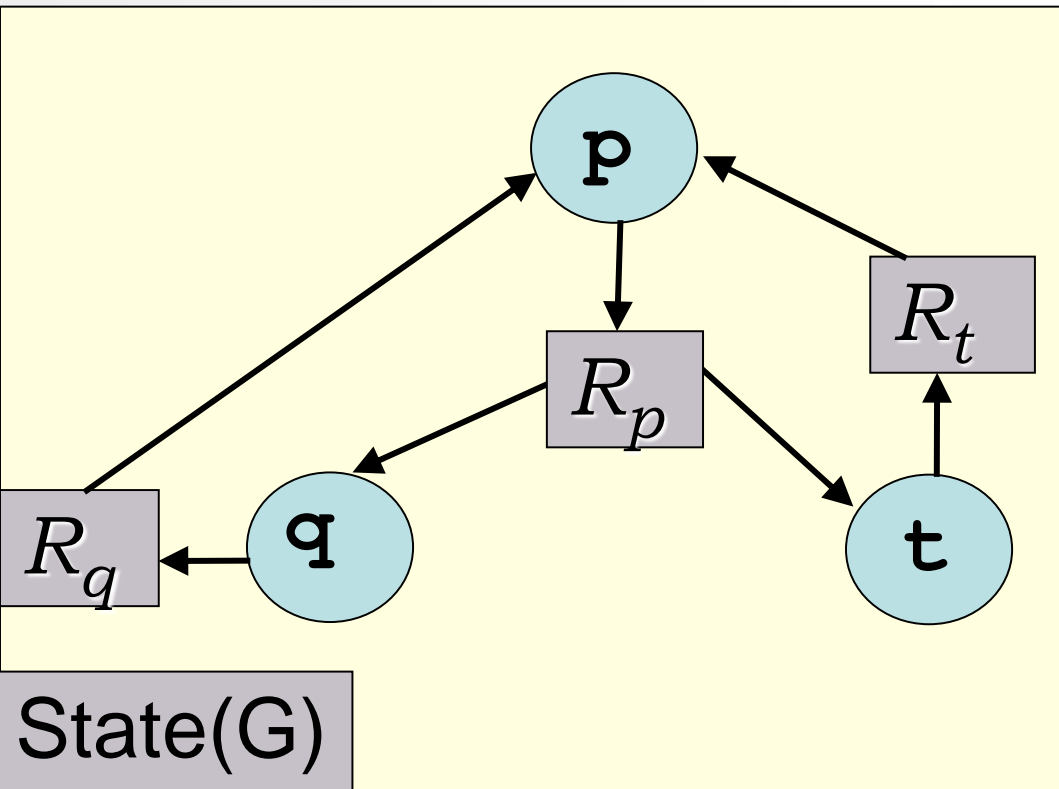
Communication models based on Single-Writer registers



Topology G

- a single register per process : multi-reader register
[MN98], [AS99], [H00], [NA02]
- a register is associated to a link : single-reader register
[DIM93], [Dolev 02]

a single register per process : state network model

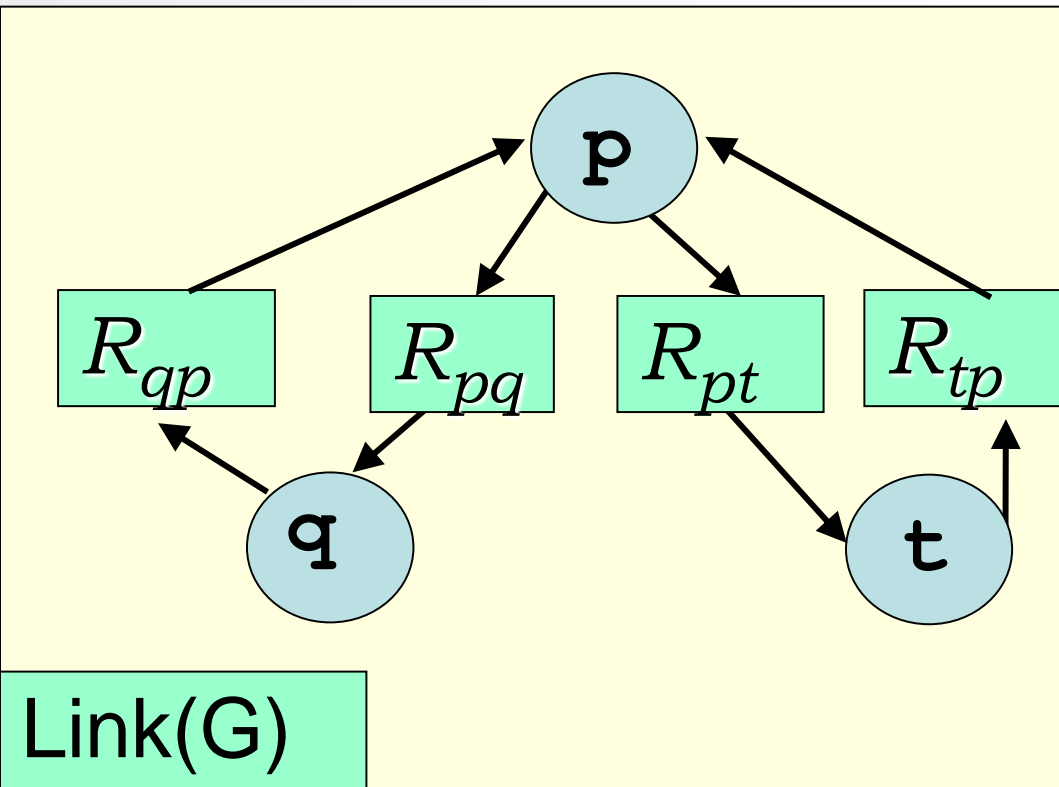


A processor p has a single atomic single-writer multi-reader register R_p

R_p is readable by p 's neighbors : q and t

R_p is writable by p

Communication model for network: link network model



A processor p has several atomic single-writer single-reader registers (one per neighbor)

R_{pt} is readable by t

R_{pt} is writable by p

Communication models based on registers on network G

Semantic location	Atomic	Regular	Safe
State multi-reader	atomic-state(G) [MN98], [AS99], [H00], [NA02]	regular-state(G)	safe-state(G)
Link singler-reader	atomic-link(G) Read-Write atomicity model [DIM93], [Dolev 02]	regular-link(G)	safe-link(G)

Distributed System

$S = (G : \text{Graph}, MC : \text{Communication Model}, A : \text{Algo})$

Computation of S is the set of computations of A on $MC(G)$

Goal : to implement every algorithm A for $MC1(G)$ on $MC2(G)$

Transformation $\tau : \text{MC1}(G) \rightarrow \text{MC2}(G)$

Let R be a register of $\text{MC1}(G)$ writable by p and readable by q

Transformation τ is two programs

$\tau(\text{READ}(R))$ returns a value

$\tau(\text{WRITE}(R, v))$

These two programs are a series of **valid READ** and **WRITE** operations on registers of $\text{MC2}(G)$

$\tau(\text{WRITE}(R, v))$ invocation by p contains only **READ** and **WRITE** operations on registers respectively readable or writable by p

$\tau(\text{READ}(R))$ invocation by q contains only **READ** and **WRITE** operations on registers respectively readable or writable by q

Simple Transformation State \rightarrow Link

On p ,

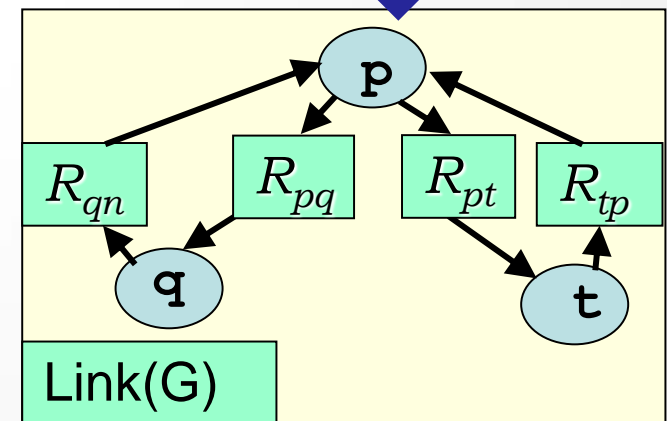
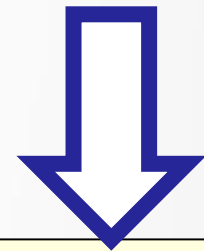
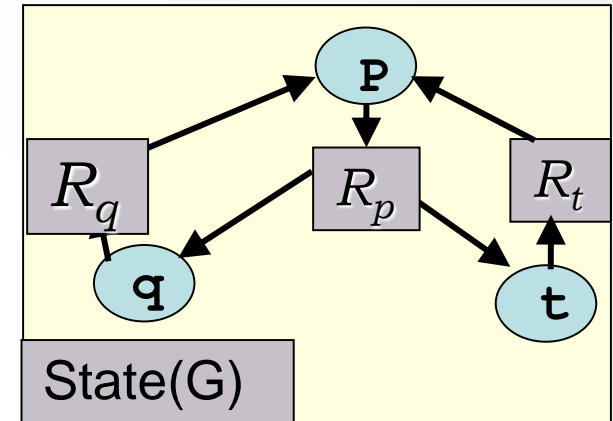
$\tau(\text{STATE-WRITE}(R_p, v)) :$

for every q in neighborhood of p do
 LINK-WRITE(R_{pq}, v)

$\tau(\text{STATE-READ}(R_q))$

$v \leftarrow \text{LINK-READ}(R_{qp})$

return v



Transformation $\tau : MC1(G) \rightarrow MC2(G)$

Transformation τ is two programs

$\tau(\text{READ}(R))$ returns a value

$\tau(\text{WRITE}(R, v))$

These two programs are a series of **valid READ** and **WRITE** operations on registers of MC2(G)

- Let A be an algorithm on $MC1(G)$
 $\tau(A) : \text{READ}(R)$ and $\text{WRITE}(R, v)$ operation invocations in A are respectively replaced by two program executions $\tau(\text{READ}(R))$ and $\tau(\text{WRITE}(R, v))$

$\tau(A)$ is an algorithm on $MC2(G)$

Compiler $\tau : MC1(G) \rightarrow MC2(G)$

τ be a transformation of $MC1(G)$ to $MC2(G)$

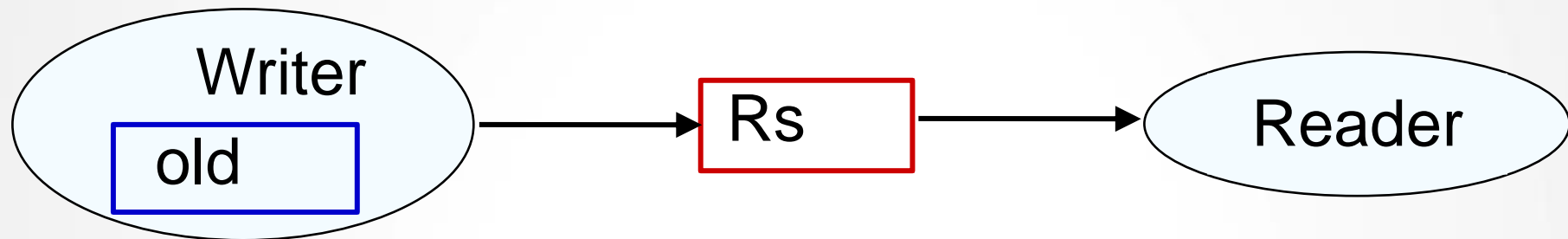
$$\tau \quad S1=(G, MC1, A) \rightarrow S2=(G, MC2, \tau(A))$$

$\tau(A) : \text{READ}(R)$ and $\text{WRITE}(R, v)$ operation invocations in A are respectively replaced by two program executions $\tau(\text{READ}(R))$ and $\tau(\text{WRITE}(R, v))$

$\tau(A)$ is an algorithm on $MC2(G)$

The transformation τ is a **compiler** iff $S2=(G, MC2, \tau(A))$ is a **syntactically and semantically valid** transformation of $S1 = (G, MC1, A)$

Wait-free implementation of binary regular SWSR register by a binary safe SWSR register [Lamport 86]



$\tau(\text{REG-WRITE}(R, \text{new}))$

```
if old  $\neq$  new then
  SAFE-WRITE(Rs, new)
  old  $\leftarrow$  new;
fi
```

$\tau(\text{REG-READ}(R))$

SAFE-READ(Rs)

Fault tolerance : Wait-Freedom

Wait-free operation: a processor can complete the operation in a finite number of steps, regardless of the actions of other processors

(i.e. a **READ** and a **WRITE** operation is done in finite number of steps)

A wait-free operation is tolerant of processor crashes

a **compiler is wait-free** if it preserves the wait-freedom property

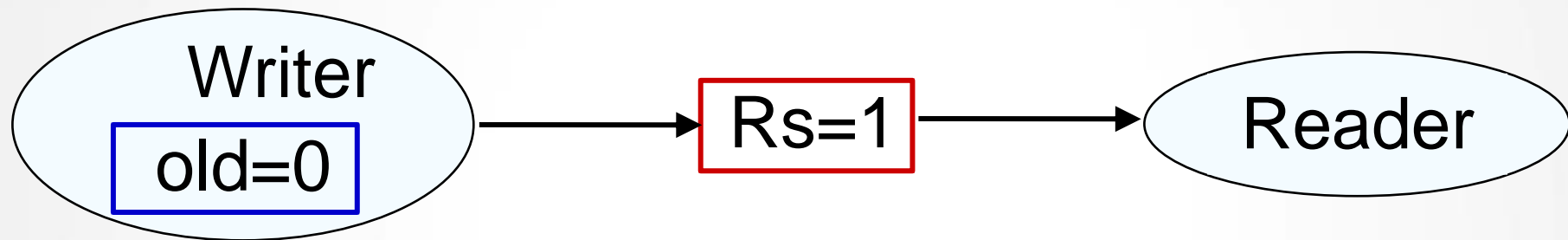
Fault-tolerance : Self-Stabilization

- Self-stabilization system automatically convergence to a legitimate configuration from any arbitrary configuration. From a legitimate configuration, the system behaves correctly (i.e. semantic of **READ** and **WRITE** operations is provided).

A self-stabilizing system is tolerant of transient failures that corrupt the processor state.

a **compiler is self-stabilizing** if it preserves the self-stabilizing property

Wait-free implementation of binary regular SWSR register by a binary safe SWSR register [Lamport 86]



All reads of the following execution return 1:
 $[\text{regular-write}(R, 0), \text{regular-read}(R)]^*$

Lamport construction is not self-stabilizing

Self-stabilizing, Wait-free implementation of SWSR regular binary register by a dual-reader safe binary register

[Hoepman, Papatriantafilou, Tsigas 02]



$\tau(\text{REG-WRITE}(R, \text{new}))$

```
if SAFE-READ( $R_s$ )  $\neq$  new
then
    SAFE-WRITE( $R_s$ , new)
fi
```

$\tau(\text{REG-READ}(R))$

SAFE-READ(R_s)

Simple Transformation State \rightarrow Link

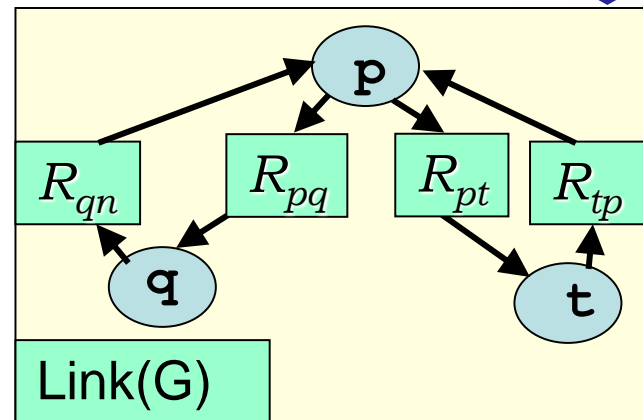
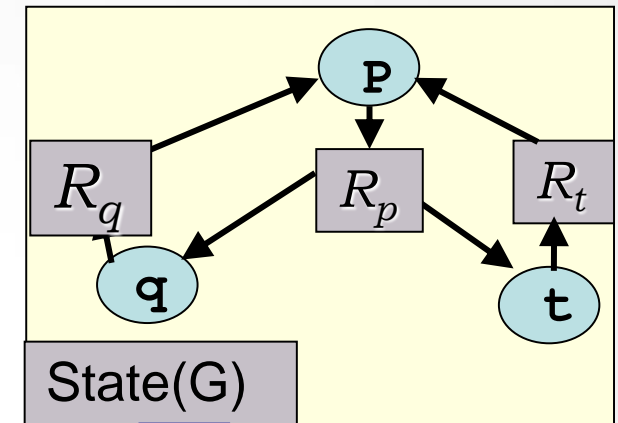
On p ,

$\tau(\text{STATE-WRITE}(R_p, v)) :$

for every q in neighborhood of p do
 $\text{LINK-WRITE}(R_{pq}, v)$

$\tau(\text{STATE-READ}(R_q))$

$v \leftarrow \text{LINK-READ}(R_{qp})$
 return v



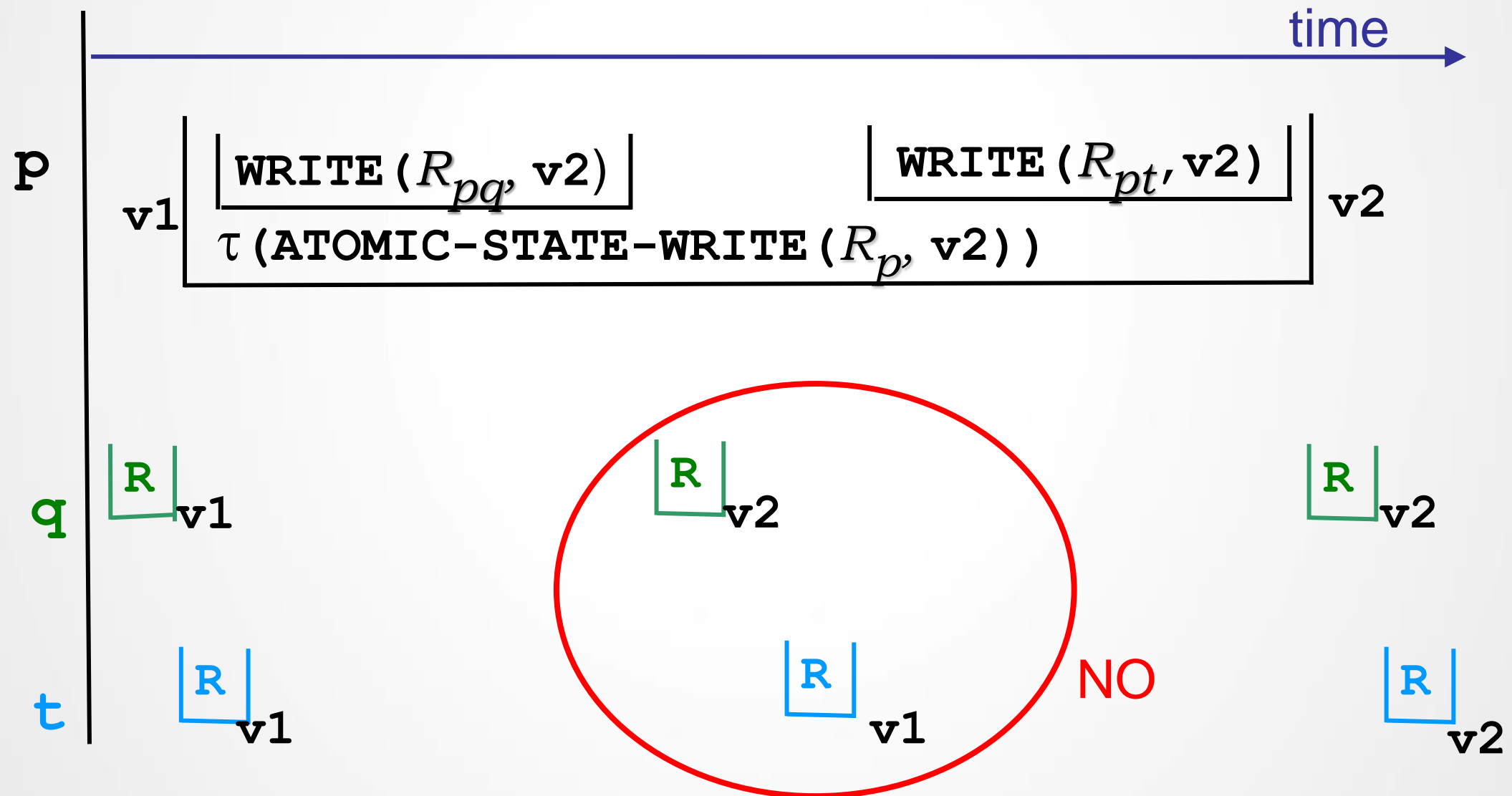
$\text{ATOMIC-LINK-WRITE}(R_{pq})$

$\text{ATOMIC-LINK-WRITE}(R_{pt})$

$\tau(\text{ATOMIC-STATE-WRITE}(R_p, v_2))$

Linearization of an execution of simple transformation on atomic registers ?

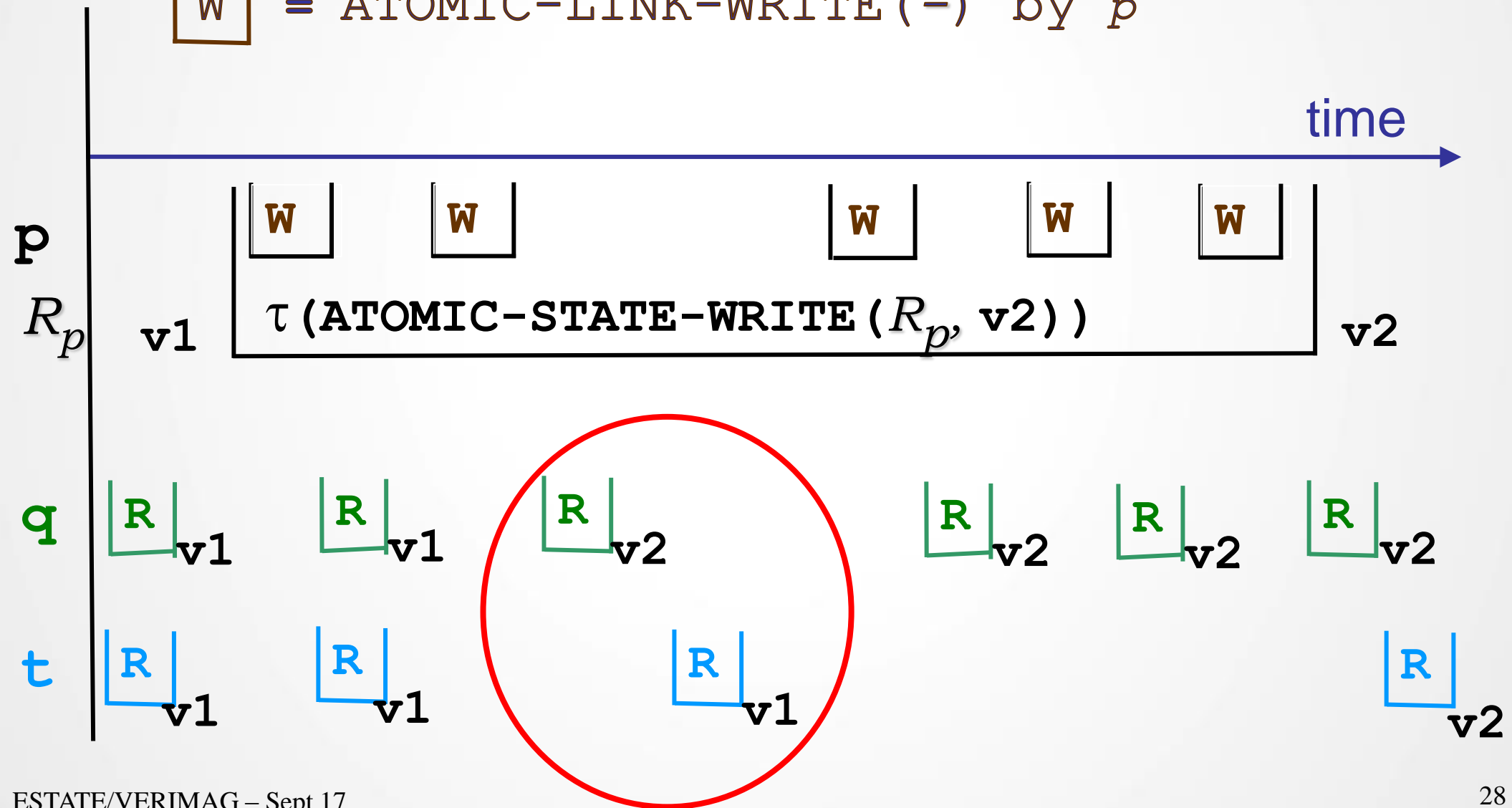
Operations on R_p $\boxed{R} = \tau(\text{ATOMIC-STATE-READ}(R_p))$




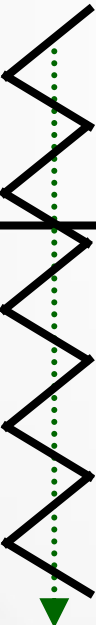
Wait-Free compiler Atomic-State(G) \rightarrow Atomic-Link(G) ? **NO**

$\boxed{R} = \tau(\text{ATOMIC-STATE-READ}(R_p))$

$\boxed{W} = \text{ATOMIC-LINK-WRITE}(-) \text{ by } p$



Wait-free compilers

	Atomic	Regular	Safe
State	Higham, Johnen 07		
multi-reader			
Link	Higham, Johnen 06		
Single reader			
		Lamport 1986	Lamport 1986

Self-stabilizing compilers

	Atomic	Regular	Safe
State			
Multi-reader	C. Johnen, L. Higham 07 ----->		
Link	L. Higham, C. Johnen 06 		
Single reader	↓		

Self-Stabilizing compiler Atomic-State(G) → Atomic-Link(G) [IPDPS06]

Drawbacks/Features of Self-Stabilizing compiler from Atomic-State(G) to Atomic-Link(G) [IPDPS06] :

- $\tau(\text{ATOMIC-STATE-WRITE}(R_p))$ is not wait-free : during an execution of $\tau(\text{ATOMIC-STATE-WRITE}(R_p))$ any p 's neighbor, q , has to do the operation $\text{ATOMIC-LINK-READ}(R_{pq})$ two times
- Each process performs infinitely often ATOMIC-STATE-READ operations
- $\tau(\text{ATOMIC-STATE-READ}(R_p))$ is not wait-free
- Each process performs two $\text{ATOMIC-STATE-WRITE}$

Self-Stabilizing compiler Atomic-State(G) → Regular-State(G)

Drawbacks/Features of compiler from Atomic-State(G) of Regular-State(G) :

- $\tau(\text{ATOMIC-STATE-WRITE}(R_p))$ is wait-free
- $\tau(\text{ATOMIC-STATE-READ}(R_p))$ is not wait-free in case there is an overlay $\tau(\text{ATOMIC-STATE-WRITE}(R_p))$
- Each process p performs one $\tau(\text{ATOMIC-STATE-WRITE}(R_p))$ operation

Wait-free and Self-stabilizing compilers

	Atomic	Regular	Safe
State multi- reader		Simple transformation	
Link			Johnen, Higham 09
Single reader	«Lamport 86 » Conjecture		



Bibliography

- Self-stabilizing algorithms in **ATOMIC-LINK** network model :

[DIM93], [Dolev 02]

called R/W atomicity model

- Self-stabilizing algorithms in **ATOMIC-STATE** network model :

[MN98], [AS99], [H00], [NA02]

compilerAtomic-state to Atomic-link

A register has 4 fields :

- written value and written flag
- copied value and copied flag

Definition of atomic-Link procedures

Read R_{xy} (wv, wf, cv, cf) : an atomic-link-read(R_{xy}) operation
retuning: w_v, w_f, c_v, c_f

Write R_{xy} (wv, wf, cv, cf) : an atomic-link-write(R_{xy})
operation to write w_v, w_f, c_v, c_f

Acknowledged-writing in R_{xy} of (v, f) - by x :

Write R_{xy} $(v, f, -, -)$	Read R_{xy} $(-, -, v, f)$
---	--

Acknowledged-reading in R_{xy} of (v, f) - by x :

Read R_{xy} $(v, f, -, -)$	Write R_{xy} $(-, -, v, f)$
--	---

$\tau(\text{ATOMIC-STATE-WRITE}(-,-))$

Procedures on R_{tp} during a $\tau(\text{ATOMIC-STATE-WRITE}(R_t, v2))$

By t:

acknowledged-writing
in R_{tp} of $(v2, 0)$

acknowledged-writing
in R_{tp} of $(v2, 1)$

$\tau(\text{ATOMIC-STATE-WRITE}(R_t, v2))$

By p:

acknowledged-
reading in
 R_{pt} of $(v2, 0)$

acknowledged-
reading in
 R_{pt} of $(v2, 1)$

$\tau(\text{ATOMIC-STATE-WRITE}(-, -))$

Procedures on R_{pt} or R_{pq} during a $\tau(\text{ATOMIC-STATE-WRITE}(R_p, -))$

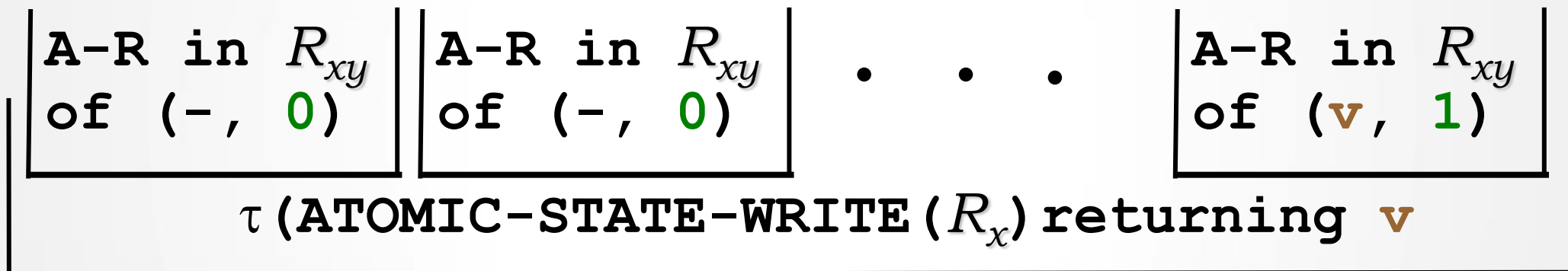
A-W in R_{pq} of $(v, 0)$	A-W in R_{pt} of $(v, 0)$	A-W in R_{pt} of $(v, 1)$	A-W in R_{pt} of $(v, 1)$
$\tau(\text{ATOMIC-STATE-WRITE}(R_p, v))$			

A-W in R_{px} of (v, f) : an acknowledged-writing in R_{px} of (v, f)
- by p

$\tau(\text{ATOMIC-STATE-READ}(R_x))$ by y

Procedures on R_{xy} during a $\tau(\text{ATOMIC-STATE-READ}(R_x))$ by y

By y :



A-R in R_{xy} of (v, f) : an acknowledged-reading(R_{xy}) returning (v, f)



No wait-free compiler on network

Let G be a network topology that is not complete

Theorem: there is not wait-free compiler from $AS(G)$
to $AL(G)$

IPDPS'06



1-Regular

[Abraham, Chockler, Keidar, Malkhi 07]

On R , a **READ** operation that does overlap a single **WRITE** operation returns the most recent preceding written value ($v1$) or the value written during the overlapping **WRITE** operations ($v2$)

