Atomic register algorithms

R. Guerraoui

Distributed Programming Laboratory
lpdwww.epfl.ch

Laboratoire de Programmation Distribuée
Distributed Programming Laboratory

Overview of this lecture

- ***(1) From regular to atomic***
- ***(2) A 1-1 atomic fail-stop algorithm***
- ***(3) A 1-N atomic fail-stop algorithm***
- ***(4) A N-N atomic fail-stop algorithm***
- ***(5) From fail-stop to fail-silent***

Fail-stop algorithms

- We first assume a fail-stop model; more precisely:
  - any number of processes can fail by crashing (no recovery)
  - channels are reliable
  - failure detection is perfect

The simple algorithm

- Consider our fail-stop ***regular*** register algorithm
  - every process has a local copy of the register value
  - every process reads ***locally***
  - the writer writes ***globally,*** i.e., at all (non-crashed) processes

The simple algorithm

- Write(v) at pi
  - send [W,v] to all
  - for every pj, wait until either:
    - received [ack] or
    - suspected [pj]
  - Return ok

- At pi:

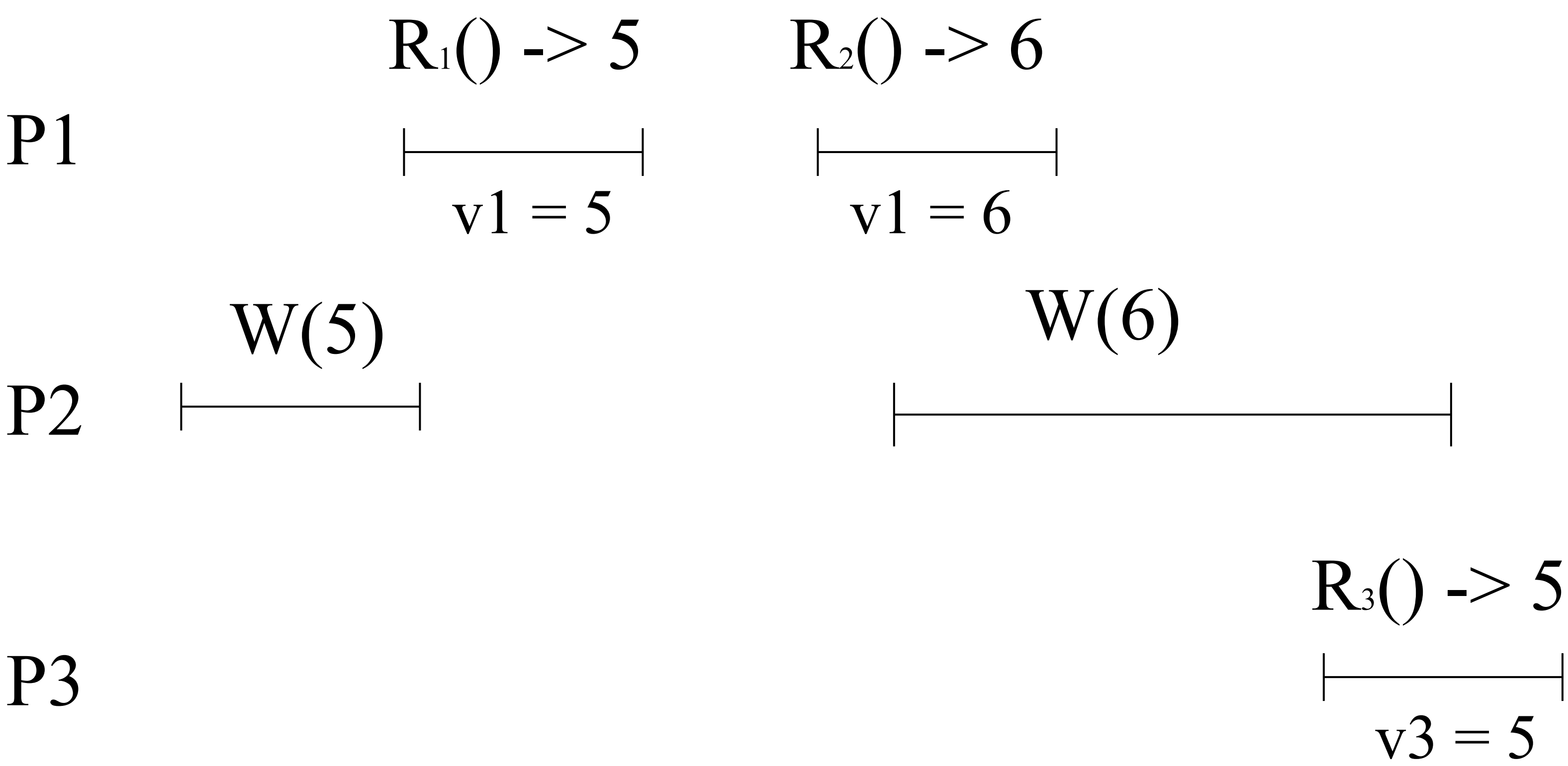
  when receive [W,v] from pj

  vi := v

  send [ack] to pj

- Read() at pi
  - Return vi

Atomicity?



Linearization?

$R_1() \to 5$ $R_2() \to 6$

P1

W(5) W(6)

P2

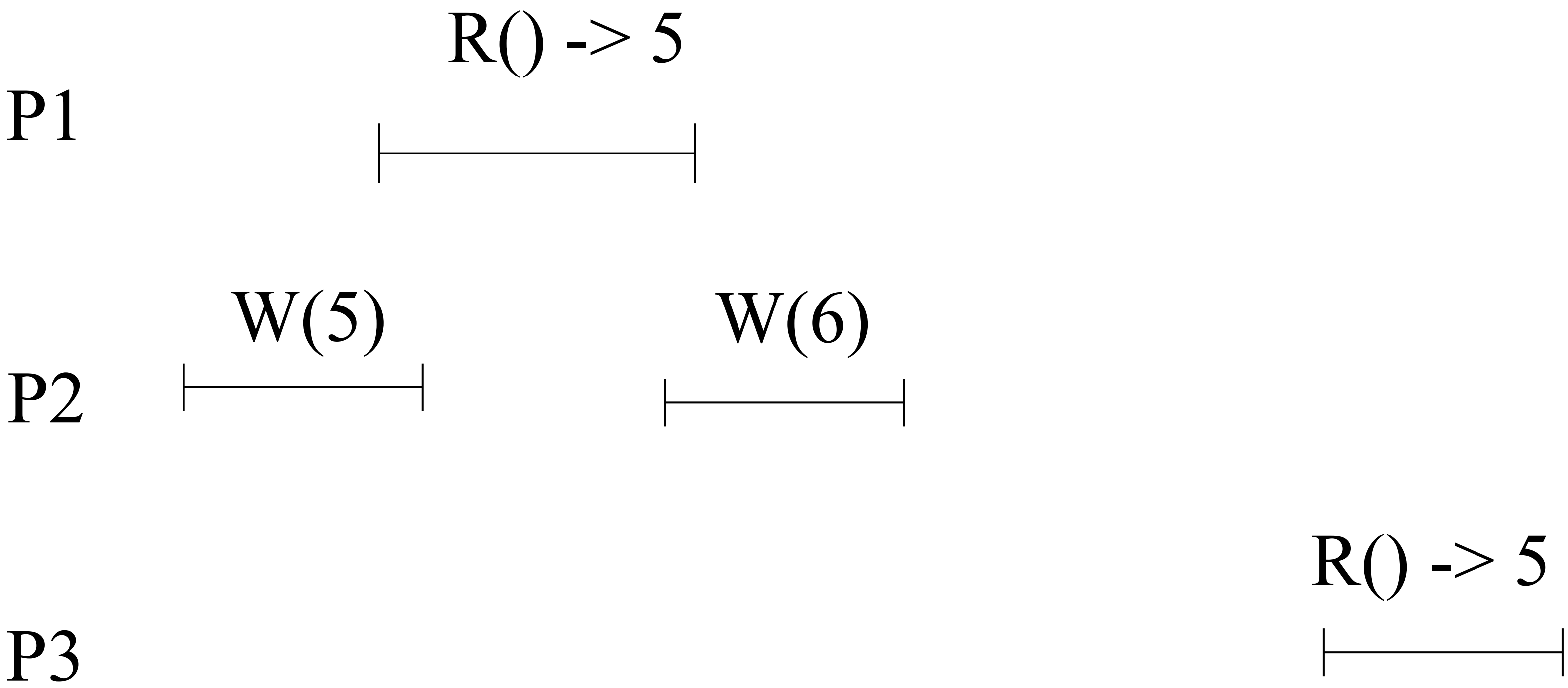
$R_3() \to 5$??

P3

Fixing the pb: read-globally

- Read() at pi
  - send [W,vi] to all
  - for every pj, wait until either:
    - receive [ack] or
    - suspect [pj]
  - Return vi

Still a problem



Linearization?

R_1() -> 5

P1

W(5) W(6)

P2

R_3() -> 5 ??

P3

Overview of this lecture

- ***(1) From regular to atomic***
- ***(2) A 1-1 atomic fail-stop algorithm***
- ***(3) A 1-N atomic fail-stop algorithm***
- ***(4) A N-N atomic fail-stop algorithm***
- ***(5) From fail-stop to fail-silent***

A fail-stop 1-1 atomic algorithm

- Write(v) at p1
  - send [W,v] to p2
  - Wait until either:
    - receive [ack] from p2 or
    - suspect [p2]
  - Return ok

- At p2:

  when receive [W,v] from p1

  v2 := v

  send [ack] to p2

- Read() at p2
  - Return v2

A fail-stop 1-N algorithm

- every process maintains a local value of the register as well as a sequence number
- the writer, p1, maintains, in addition a timestamp ts1
- any process can read in the register

A fail-stop 1-N algorithm

- Write(v) at p1
  - ts1++
  - send [W,ts1,v] to all
  - for every pi, wait until either:
    - receive [ack] or
    - suspect [pi]
  - Return ok

- Read() at pi
  - send [W,sni,vi] to all
  - for every pj, wait until either:
    - receive [ack] or
    - suspect [pj]
  - Return vi

A 1-N algorithm (cont'd)

- At pi
  - When pi receive [W,ts,v] from pj

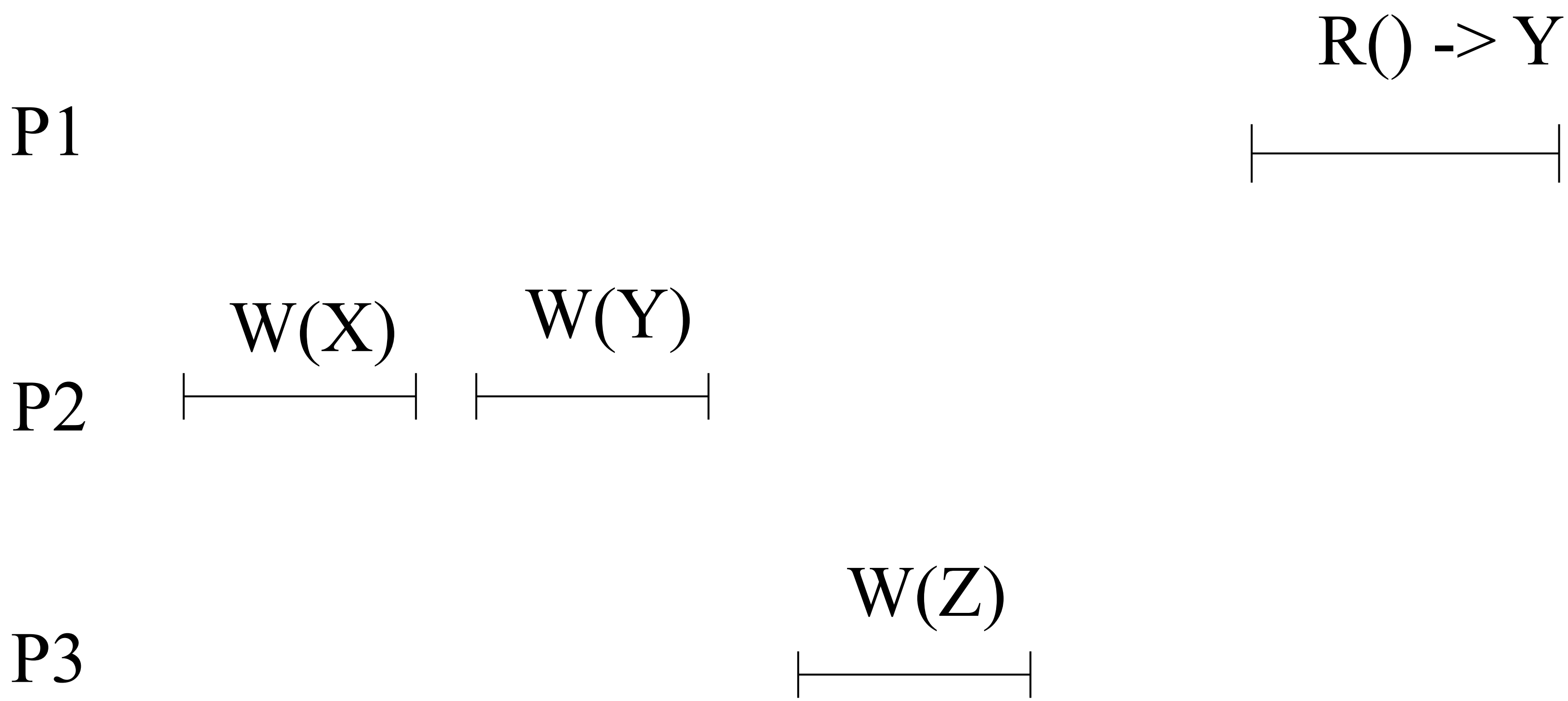
    if ts > sni then

        vi := v

        sni := ts

    send [ack] to pj

Why not N-N?



The Write() algorithm

- Write(v) at pi
  - ✓ send [W] to all
  - ✓ for every pj wait until
    - **receive [W,snj] or**
    - **suspect pj**
  - ✓ (sn,id) := (highest snj + 1,i)
  - ✓ send [W,(sn,id),v] to all
  - ✓ for every pj wait until
    - **receive [W,(sn,id),ack] or**
    - **suspect pj**
  - ✓ Return ok

- At pi

  T1:
  - ✓ when receive [W] from pj
    - send [W,sn] to pj

  T2:
  - ✓ when receive [W,(snj,idj),v] from pj
  - ✓ If (snj,idj) > (sn,id) then
    - vi := v
    - (sn,id) := (snj,idj)
  - ✓ send [W,(snj,idj),ack] to pj

The Read() algorithm

- Read() at pi
  - ✓ send [R] to all
  - ✓ for every pj wait until
    - **receive [R,(snj,idj),vj] or**
    - **suspect pj**
  - ✓ v = vj with the highest (snj,idj)
  - ✓ (sn,id) = highest (snj,idj)
  - ✓ send [W,(sn,id),v] to all
  - ✓ for every pj wait until
    - **receive [W,(sn,id),ack] or**
    - **suspect pj**
  - ✓ Return v

- At pi

T1:

- ✓ when receive [R] from pj
  - send [R,(sn,id),vi] to pj

T2:

- ✓ when receive [W,(snj,idj),v] from pj
- ✓ If (snj,idj) > (sn,id) then
  - vi := v
  - (sn,id) := (snj,idj)
- ✓ send [W,(snj,idj),ack] to pj

Overview of this lecture

- ***(1) From regular to atomic***
- ***(2) A 1-1 atomic fail-stop algorithm***
- ***(3) A 1-N atomic fail-stop algorithm***
- ***(4) A N-N atomic fail-stop algorithm***
- ***(5) From fail-stop to fail-silent***

From fail-stop to fail-silent

- We assume a majority of correct processes
- In the 1-N algorithm, the writer writes in a majority using a timestamp determined locally and the reader selects a value from a majority and then imposes this value on a majority
- In the N-N algorithm, the writers determines first the timestamp using a majority