Registers

Prof R. Guerraoui

Distributed Programming Laboratory

Register

- A ***register*** has two operations: ***read()*** and ***write()***

- Sequential specification

  - ***read()***

    - return(x)

  - ***write(v)***

    - x <- v; return(ok)

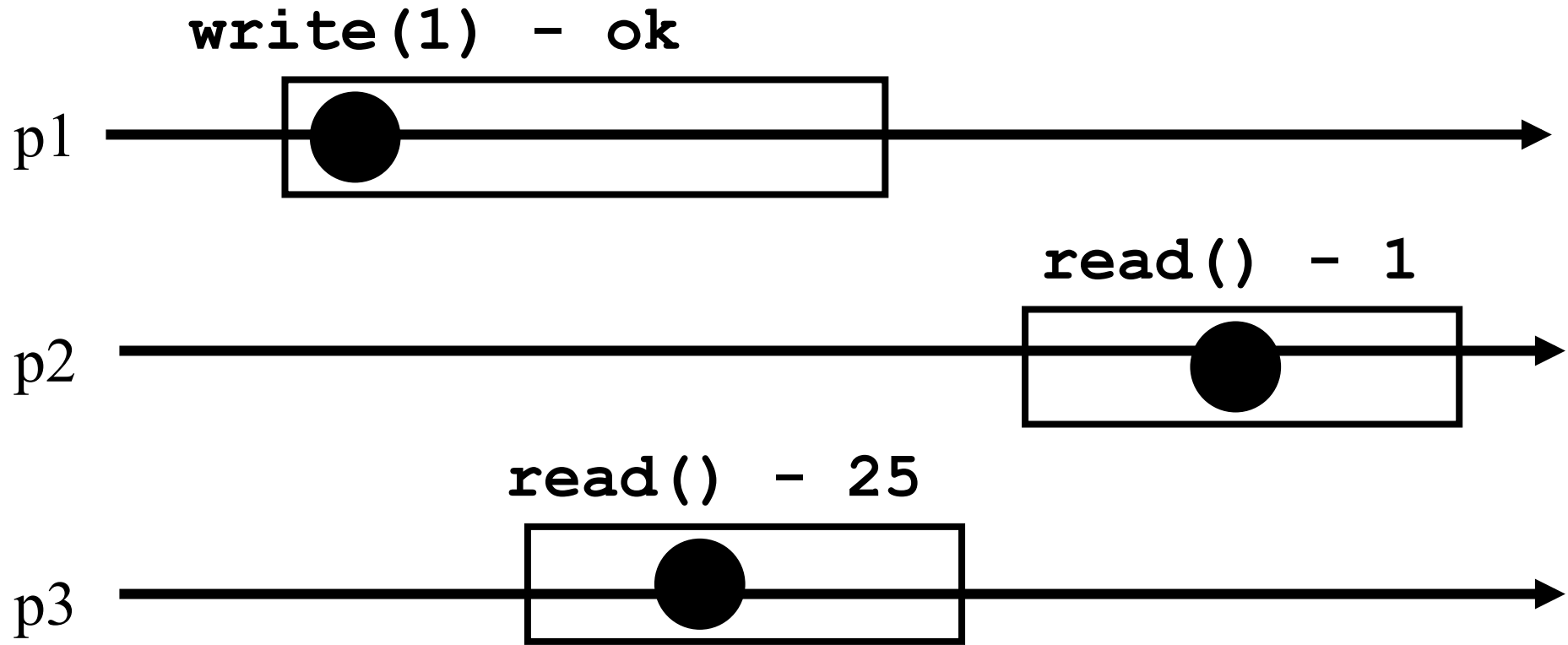# Simplifications

- We assume that **registers** contain only integers

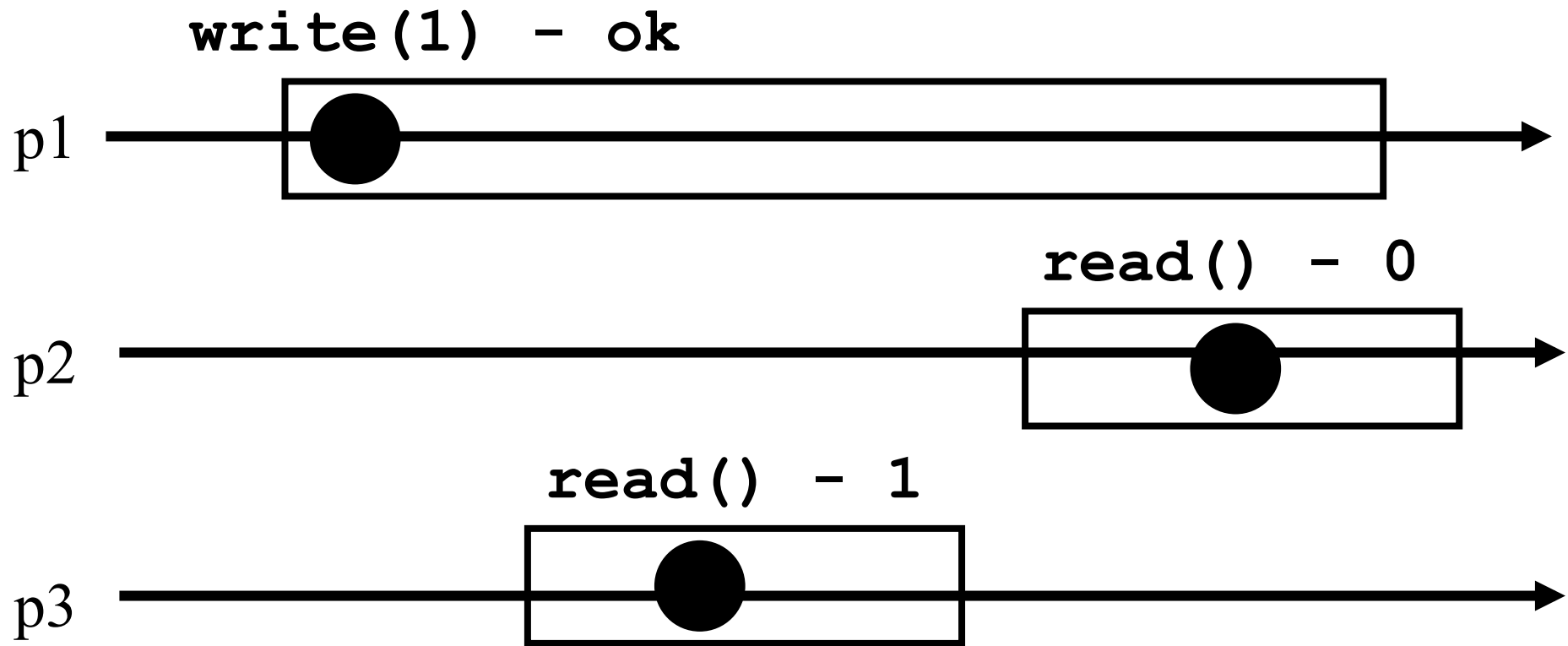- Unless explicitly stated otherwise, **registers** are initially supposed to contain 0

Space of registers

🔹 Dimension 1: binary (boolean) – multivalued

🔹 Dimension 2:

　　🔹 SRSW (single reader, single writer)
　　🔹 MRSW (multiple reader, single writer)
　　🔹 MRMW (multiple reader, multiple writer)

🔹 Dimension 3: safe – regular – atomic

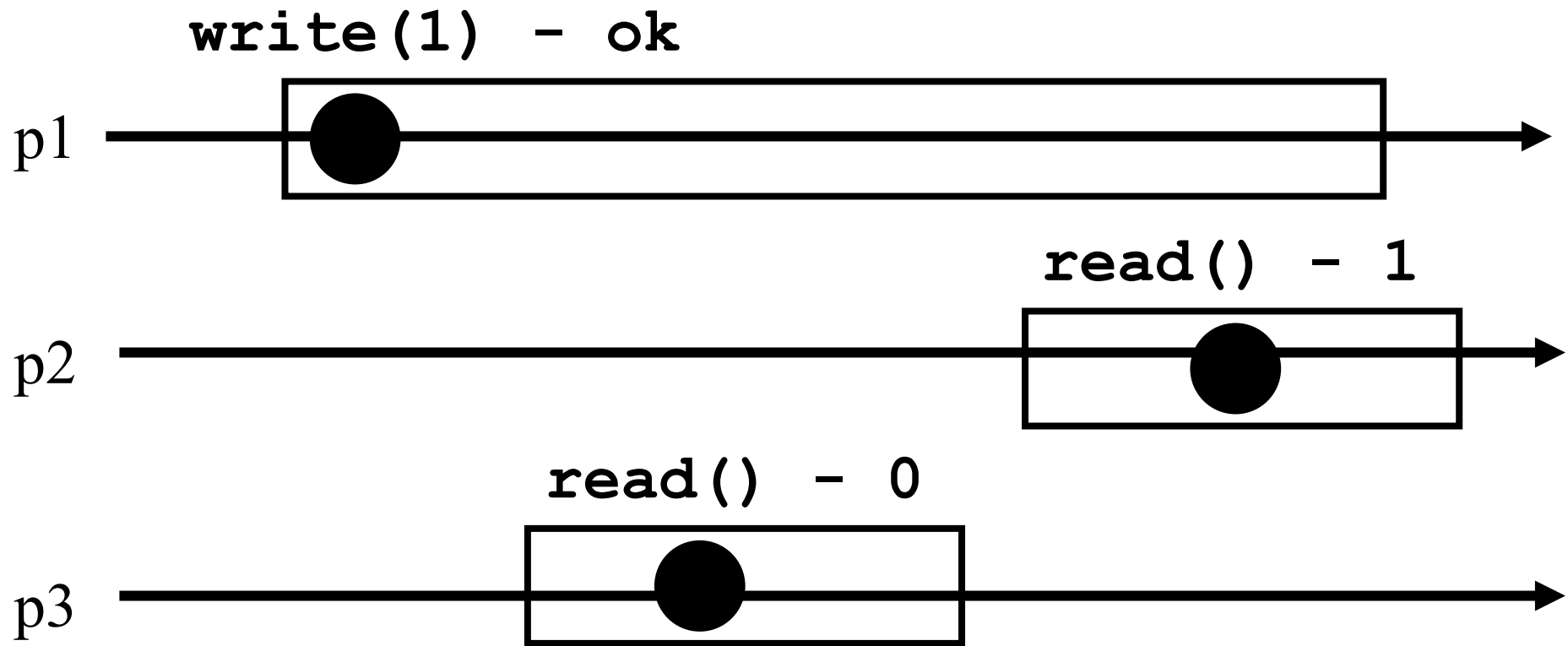# Safe execution

write(1) - ok

p1

read() - 1

p2

read() - 25

p3

Regular execution

write(1) - ok

p1

read() - 0

p2

read() - 1

p3

Atomic execution

write(1) - ok

p1

read() - 1

p2

read() - 0

p3

2 decades of hard work

Theorem: A multivalued MRMW atomic *register* can be implemented with binary SRSW safe *register*

Algorithms

- The process executing the code is implicitely assumed to be pi

- We assume a system of N processes

- NB. We distinguish base and high-level registers

Conventions

- The operations to be implemented are denoted *Read()* and *Write()*

- Those of the base registers are denoted *read()* and *write()*

- We omit the *return(ok)* instruction at the end of *Write()* implementations

(1) From (binary) SRSW safe to (binary) MRSW safe

- We use an array of SRSW *registers*
  Reg[1,..,N]
- **Read()**
  - return (Reg[i].read());

- **Write(v)**
  - for j = 1 to N
    - Reg[j].write(v);

From (binary) SRSW safe to (binary) MRSW safe

- The transformation works also for multi-valued registers and regular ones

- It does not however work for atomic registers

(2) From binary MRSW safe to binary MRSW regular

🐾 We use one MRSW safe register

🐾 **Read()**

🐾 return(Reg.read());

- **Write(v)**

🐾 if old ≠ v then

🐾 Reg.write(v);

🐾 old := v;

From binary MRSW safe to binary MRSW regular

- The transformation works for single reader **registers**

- It does not work for multi-valued **registers**

- It does not work for atomic **registers**

(3) From *binary* to *M-Valued* MRSW regular

- We use an array of MRSW registers Reg[0,1,..,M] init to [1,0,..,0]

- **Read()**
  - for j = 0 to M
    - if Reg[j].read() = 1 then return(j)

- **Write(v)**
  - Reg[v].write(1);
  - for j=v-1 downto 0
    - Reg[j].write(0);

From *binary* to *M-Valued* MRSW regular

- The transformation would not work if the Write() would first write 0s and then 1

- The transformation works for regular but NOT for atomic registers

(4) From SRSW *regular* to SRSW *atomic*

- We use one SRSW register Reg and two local variables t and x

- **Read()**
  - (t',x') = Reg.read();
  - if t' > t then t:=t'; x:=x';
  - return(x)

- **Write(v)**
  - t := t+1;
  - Reg.write(v,t);

From SRSW regular to SRSW atomic

- The transformation would not work for multiple readers

- The transformation would not work without timestamps

(variable t represents logical time)

(5) From SRSW atomic to MRSW atomic

- We use N*N SRSW atomic registers RReg[(1,1),(1,2),..,(k,j),..(N,N)] to communicate among the readers

  - In RReg[(k,j)] the reader is pk and the writer is pj

- We also use n SRSW atomic **registers** WReg[1,..,N] to store new values

  - the writer in all these is p1

  - the reader in WReg[k] is pk

(5) From SRSW atomic to MRSW atomic (cont'd)

Write(v)

- t1 := t1+1;
- for j = 1 to N
  - WReg.write(v,t1);

(5) From SRSW atomic to MRSW atomic (cont'd)

Read()

- for j = 1 to N do
    - (t[j],x[j]) = RReg[i,j].read();
- (t[0],x[0]) = WReg[i].read();
- (t,x) := highest(t[..],x[..]); ◁ **Value with highest timestamp**
- for j = 1 to N do
    - RReg[j,i].write(t,x);
- return(x)

From SRSW atomic to MRSW atomic

- The transformation would not work for multiple writers

- The transformation would not work if the readers do not communicate (i.e., if a reader does not write)

(6) From *MRSW* atomic to *MRMW* atomic

- We use N MRSW atomic registers Reg[1,..,N]; the writer of Reg[j] is pj

- **Write(v)**
  - for j = 1 to N do
    - (t[j],x[j]) = Reg[j].read();
  - (t,x) := highest(t[..],x[..]);
  - t := t+1;
  - Reg[i].write(t,v);

(6) From MRSW atomic to MRMW atomic (cont'd)

- **Read()**
    - for j = 1 to N do
        - (t[j],x[j]) = Reg[j].read();
    - (t,x) := highest(t[..],x[..])**;**
    - return(x)