Some results with MWMR registers

C. Delporte-Gallet, H. Fauconnier,
and E. Gafni, L. Lamport, S. Rajsbaum

Context: shared memory

- Context : Shared memory
  - n asynchronous processes sharing r registers.
    - Tasks
- Space complexity
  - How many registers are needed to solve a given task?

Registers: Classical results

registers [Lamport86]:

binary, multi-values

concurrency:

safe

regular

atomic (linearizable)

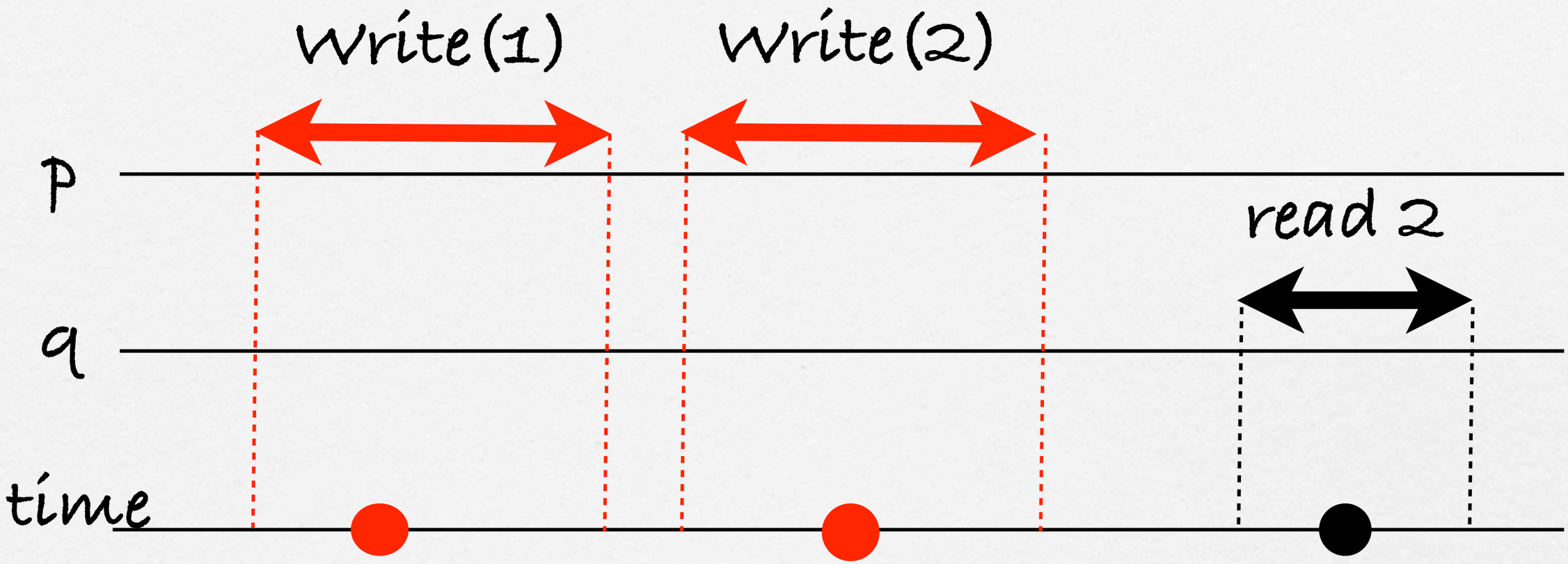
SWSR / SWMR / MWMR

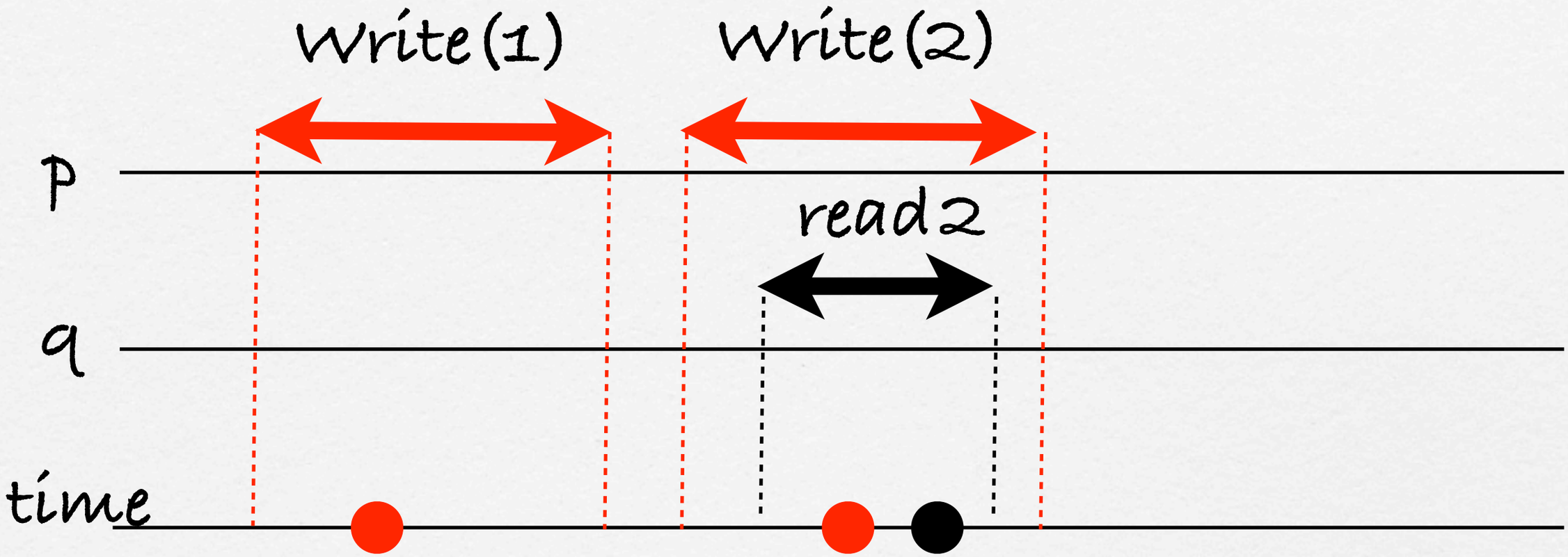
main result: [Anderson et al.94,Dolev-Shavit97...]

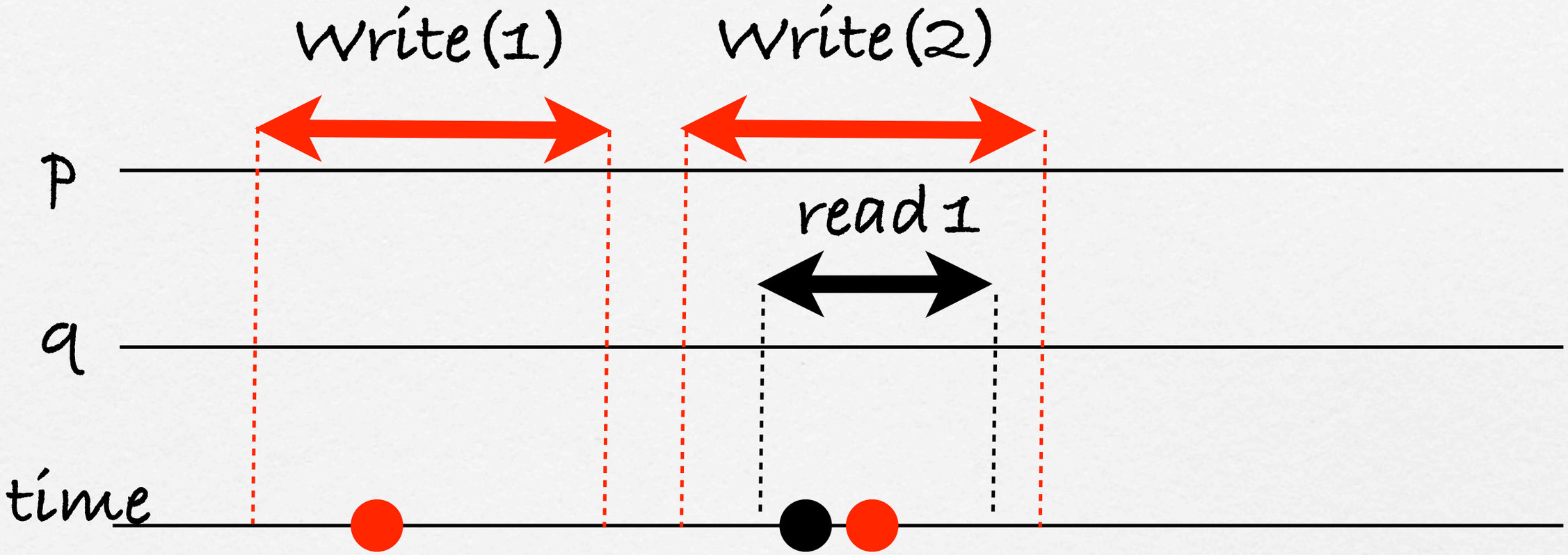
with weaker registers (e.g. binary, SWSR, safe) it is possible to (wait-free) implement the strongest ones (e.g. multi-valuate, MWMR, atomic)

registers

- sequential specification: a variable w
  - write(w,v): w:=v
  - read(w) returns the value of the variable
- concurrency: more than one process may access the registers:
  - begin_read end_read, begin_write end_write
  - linearizability (atomicity):
    - the history of the register corresponds to a sequential execution that satisfies the sequential specification.
    - for all read - write operation there is a time (linearization point) between the beginning and the end of the operation such that considering these times the sequential specification is ensured


Write(1)
Write(2)
p
read 2
q
time


Write(1)
Write(2)
p
read2
q
time


Write(1)
Write(2)
p
read 1
q
time

- multi valued, atomic registers
- SWMR, MWMR

Registers

- Processes $\{p_1,\ldots,p_n\}$ with names in some set N. Name(p_i) is the name of process p_i
- set of registers R.
  - for register r in R: some processes may perform write(r,v), read(r) ...
  - with SWMR register only process p may write register r_p and every process may read r_p

Registers...

- with MWMR
  - if p writes v in r and just after q writes v' in r, then the write of p is not visible
  - It is possible that:
    xp is always 0
    (the writes of q erase the writes of p)
    or xp always 1 or...
    same for xq

```
p: forever
        write(r,1); xp:=read(r)

q: forever
        write(r,0); xq:=read(r)
```

Space complexity

number of SWMR registers in shared memory:

- if each process has its own SWMR registers « everything » can be done.
- If a process does not have its SWMR register, it is not visible
- space complexity (the number of registers) doesn't make sense with SWMR registers

SPACE COMPLEXITY

number of MWMR registers in shared memory:

- if each process has its own MWMR registers « everything » can be done.
- But what can be done if each process does not have its own register (MWMR registers)? more registers are needed?
- What can be done with less than n registers:
  - lower bound n for mutual exclusion (without failure)
  - lower bound $\log(n)$ for leader election (without failure)
  - lower bound $\sqrt(n)$ for randomized consensus

Space complexity

- with MWMR registers, the number of registers may be less or more than the number of processes:
  - space complexity (the number of registers) makes sense with MWMR registers

space complexity for MWMR registers

- if each process has its own (SWMR) register « everything » is possible
- first step for space complexity for MWMR registers: implement or emulate SWMR with MWMR
  - if there is such an implementation/ emulation with f(n) MWMR registers, f(n) registers are enough to do « everything »

two ways...

A system of n-SWMR registers may be emulated by a system of MWMR registers

1. assign to each process one MWMR register only this process writes the register: SWMR
2. « direct » emulation of the SWMR by the MWMR registers (processes cooperates using MWMR registers and emulate standard RW model)

Main results (this talk)

- A system of n-SWMR registers may be (non-blocking) emulated by a system of n-MWMR registers (and n MWMR are needed)
- A system of n-SWMR registers may be (wait-free) emulated by a system of (2n-1)—MWMR registers
- Every wait-free solvable n-task is solvable With n MWMR registers
- Adaptive version: when n is not a known constant, a system of n-SWMR registers may be emulated by a system of (3n)-MWMR registers.

with O(n) MWMR registers
« everything » may be done

First approach:

assign MWMR registers to each process

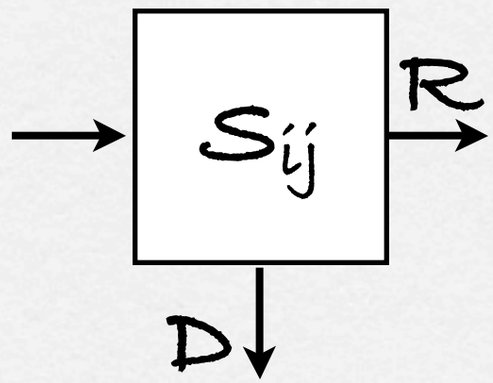
- each process has a name from a set M={1,..,m} assign to process with name x register R[x]
  - But M may be very large...
    number of registers= |M|
- renaming: give to each process a new name in a « small » set of names then assign registers to processes.
  number of registers = size of this « small » set of names

renaming!

renaming...

(here renaming only with MWMR registers)

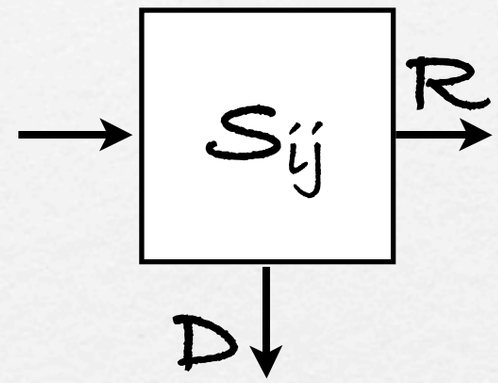
- «splitters» [Moir, Anderson'95]:
  - validity: output Right, Down or Stop
  - solo: if a single process invokes the splitter only Stop is possible
  - concurrency:
    - when k processes invoke the splitter:
      - at most k-1 processes obtain Right
      - at most k-1 processes obtain Down
      - at most one process obtains Stop
- when a process obtains Stop -> gets a new name



splitter

Code for the splitter with identity my_name:

```
Last:=my_name
if(Closed) then return Right
else Closed:=True;
   if(Last=my_name)
   then return(Stop)
   else return(Down)
```

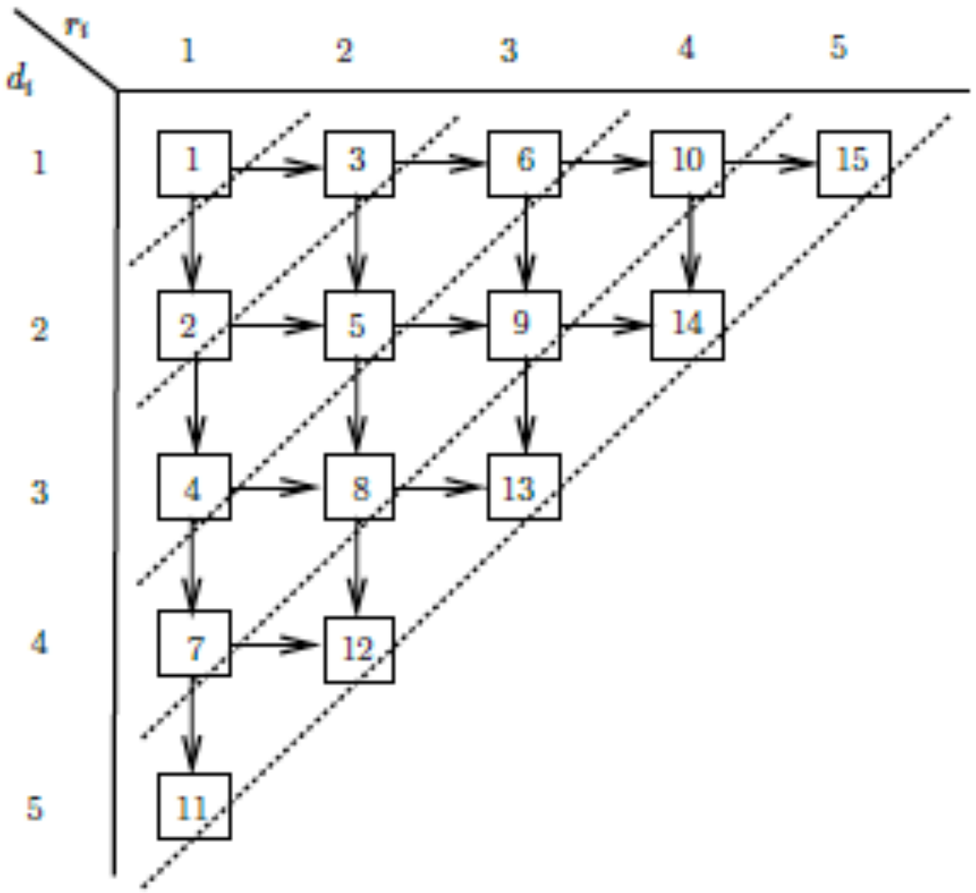


if k processes:

- R: closed=true -> at least one D or S
- S or D: the last one that writes Last before Closed=True goes to Stop or to R (if Closed changes)

Renaming

- Grid of n(n+1)/2 splitters (with an enumeration of the splitters on {1,..., n(n+1)/2})
- when a process obtains Stop it chooses as new name the name of the splitter
- No process stops on the same splitter
- Every process stops on some splitter



Renaming with splitters

- Complexity:
  - $\Theta(n^2)$ splitters
  - size set of new names $\Theta(n^2)$
  - two registers for each splitter (only MWMR registers): $\Theta(n^2)$
  - (improvement [Aspnes'10] $\Theta(n^{3/2})$ splitters)
- from MWMR to SWMR:
  - renaming uses $\Theta(n^2)$ (or $\Theta(n^{3/2})$) MWMR registers
  - new set of names of size $\Theta(n^2)$ (or $\Theta(n^{3/2})$):
    - R[new-name(p)] is the register for p
    - Size of R: $\Theta(n^2)$ (or $\Theta(n^{3/2})$)

$\Theta(n^2)$ (or with the improvement $\Theta(n^{3/2})$) MWMR registers

with renaming

- with splitters standard RW model may be implemented with $\Theta(n^{3/2})$ registers.

(not linear)

another approach: emulation

- (with known algorithms) renaming doesn't enable to have a solution using only a linear number of registers to ensure that each process has its own register.
- «Direct» solution: emulate n SWMR registers with MWMR registers without renaming (do not assign processes to registers)

emulate SWMR

Results:

- with less than n MWMR no emulation of n SWMR
- with n MWMR registers n SWMR may be emulated (but only non-blocking emulation)

Impossibility: covering

- If p is just about to write register reg (p covers reg), q writes reg, just after p writes, the value written by q is «lost»
- With m registers and a set of m processes covering each register, all values written in the m registers may be «lost».

Covering

it can be proven that in every emulation necessarily each process p has to write in n registers.

By contradiction, assume n-1 registers.

Write=high level write (emulated write)

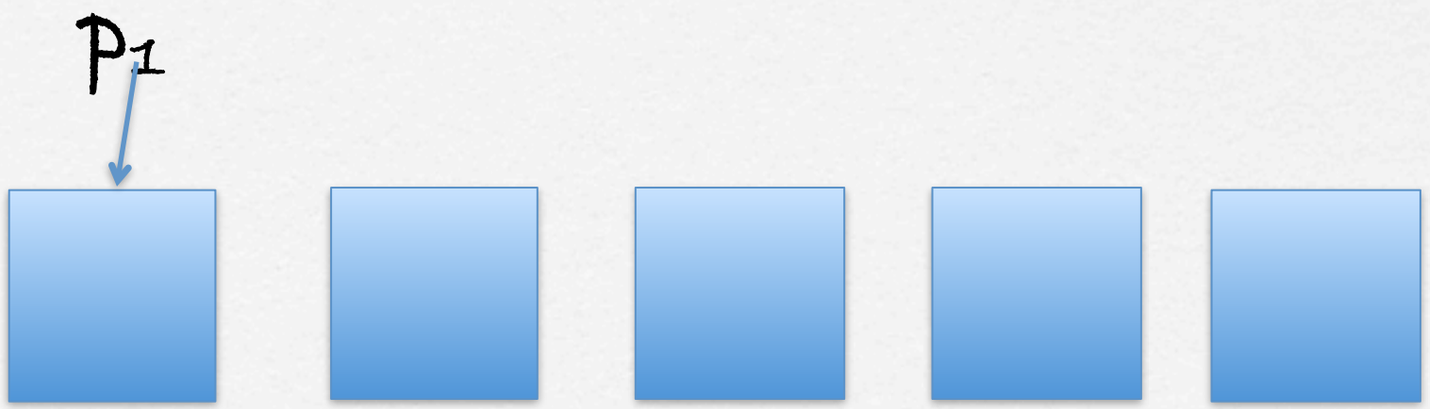
write=write in the basic registers

Covering

p_1 Write(1),

If p_1 does not write in some registers, it is impossible to Read its written value.

p_1 writes in some register. Stop it just before this write.

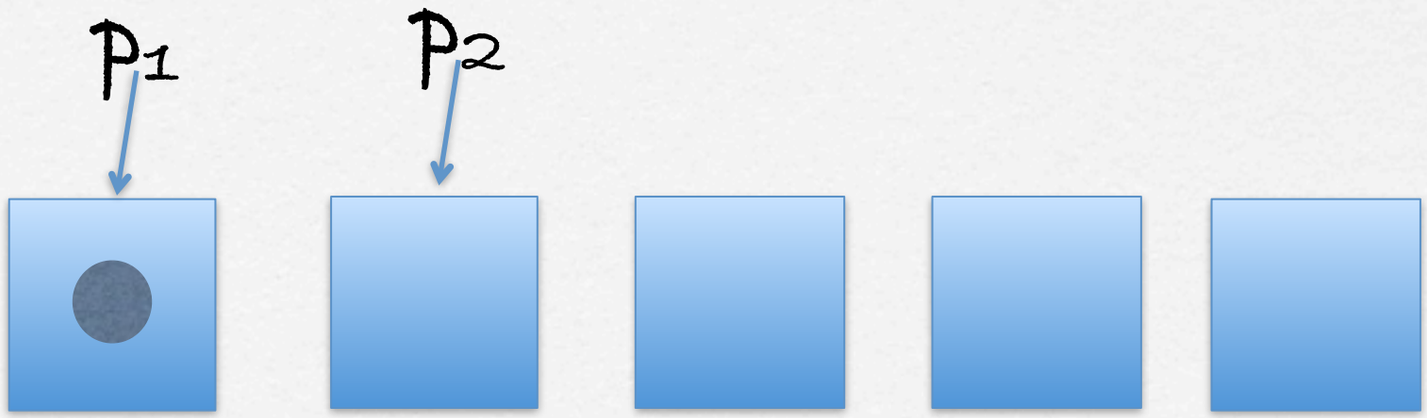


Covering

p2 Write(1) 1.

If p2 writes only in the first register, p1 covers —> p2 writes in another register.

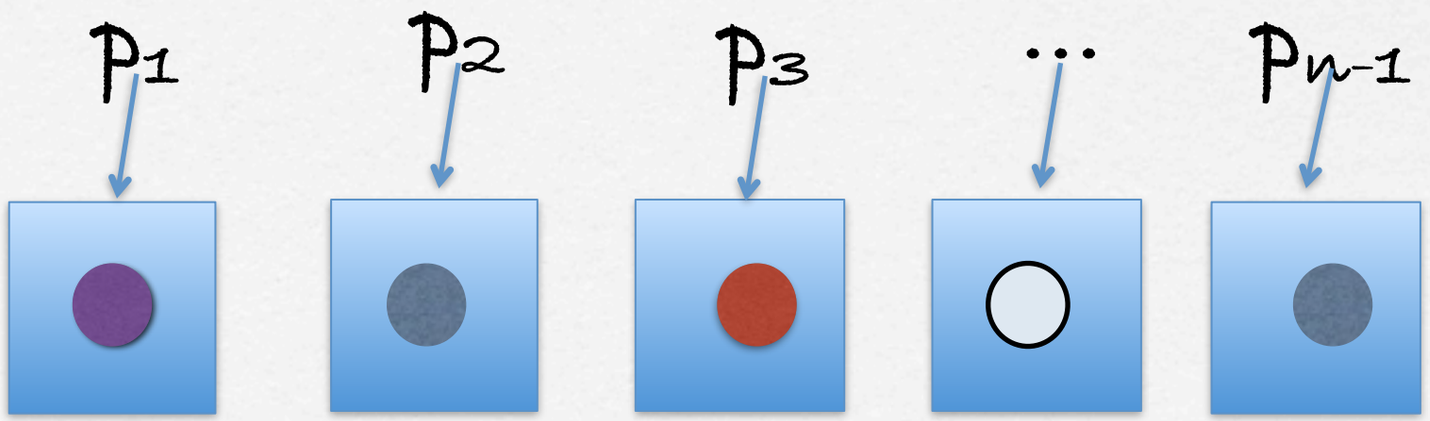
Stop p2 just before



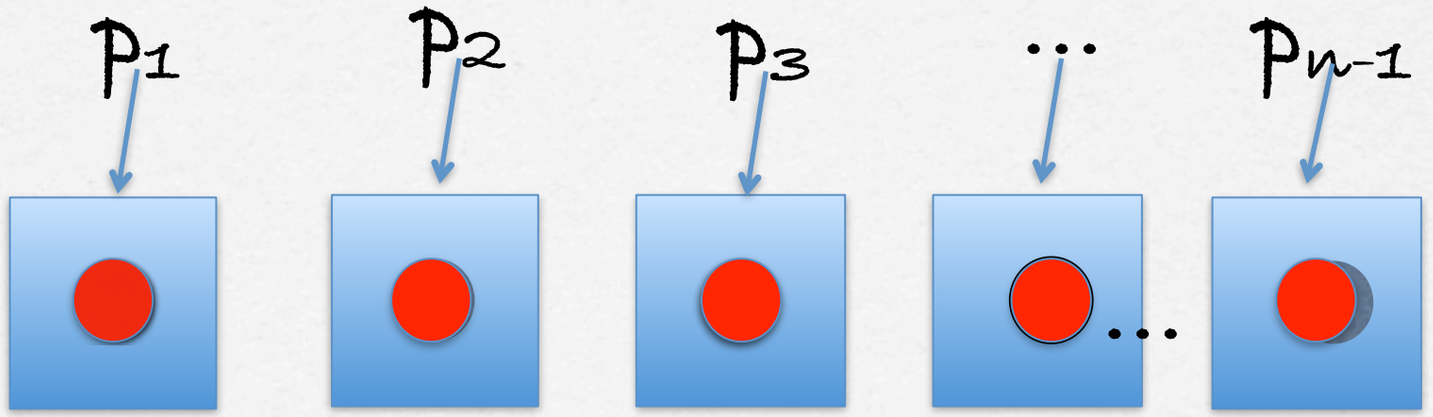
Covering

stop each process just before in this way

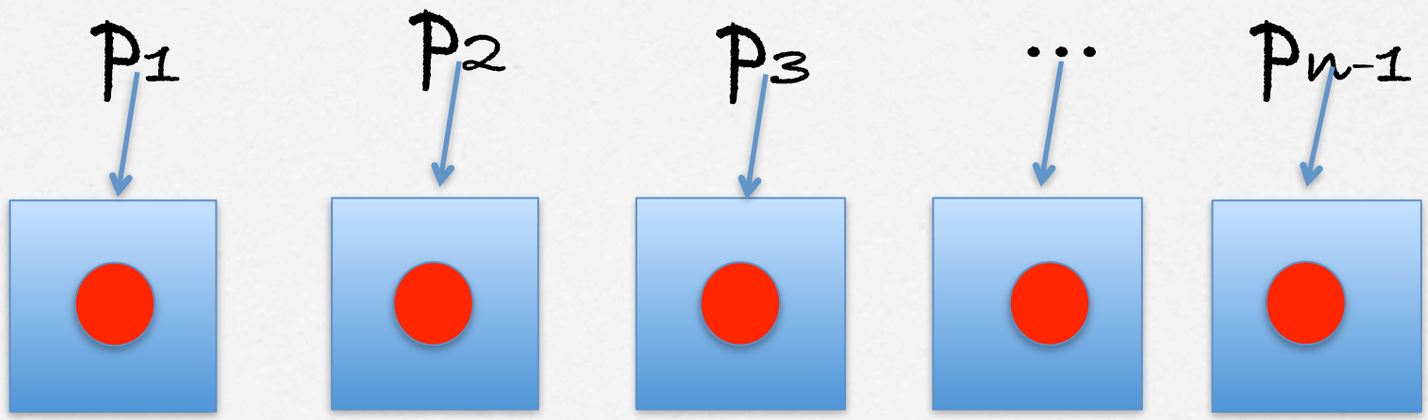
with n-1 processes it is possible to have:



process p_n executes Write(1) and runs alone: it writes in the registers

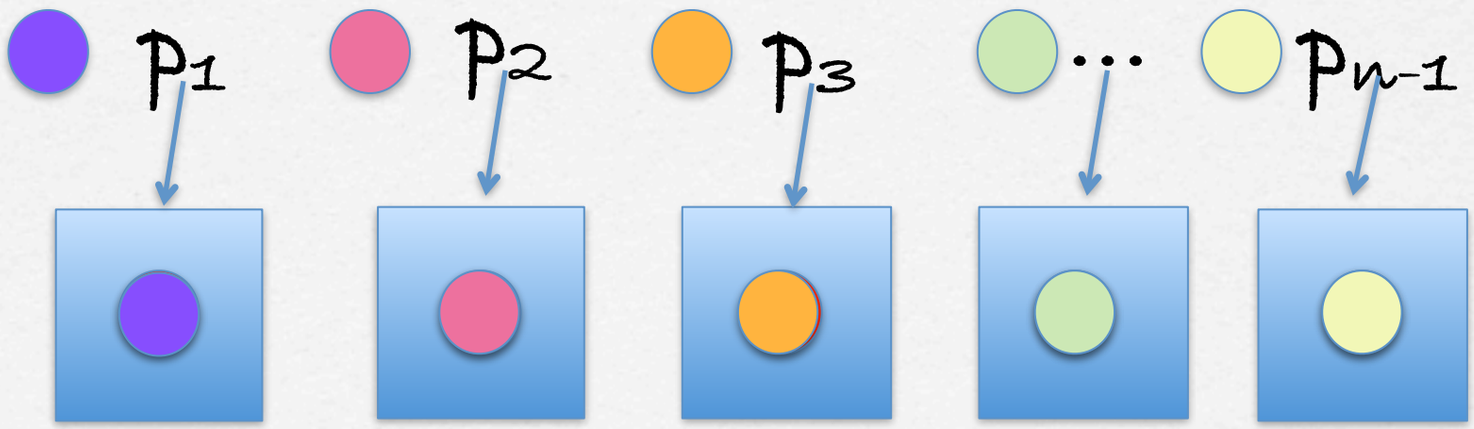


process p_n executes Write(1) and runs alone



until it finishes its Write

then $p_1, p_2, , p_{n-1}$ write



values written by p_n are lost!

Impossibility

- with less than n registers no solution.

Algorithm with n registers

- n processes,
- (array R of) n MWMR registers

to Write *a value x*

View: all the values already seen
add x in View
    read all the values in R , add them to View
    choose a register (e.g. round robin)
    write View in this register
until all registers contain x

Pigeonhole Principle!

then when the Write ends, x is in all registers
some registers may be crushed,
but at most n-1 registers may be covered,
then at least one register will be read before the next write
the next Write reads all registers hence reads x then writes x again in the registers
if x is in all registers even if each other process covers a register, x will be forever in some register (and eventually in all)

to Read a value

read all the values in R ,
choose the « freshest » one

(for this the k-th Write of v, p writes
(v, p, k) in the registers)

if a process terminates its Write(x)

- x will be forever in some register (and eventually in all)
- a read of all the registers contains x:
  - SAFETY
  - regular register

Only Non-blocking

non-blocking

a process may never succeed to Write:

each time it writes on a register
another process writes the same
register

n-1 processes may be prevented to write !

but at least one succeed

result: tasks

- n-task:
    - (I,O,D):
        - I input (n-vector: one value for each process),
        - O output (n-vector: one value for each process),
        - D specification of the task mapping every input to a set of possible outputs.
- liveness
    - wait-free: each process may terminate in a bounded number of its own steps
    - obstruction-free: if a process takes steps alone it terminates.

Consequence...

- [ ] tasks

Tasks

decision task: (example consensus renaming...)

in a (wait-free) task all correct processes have to terminate then:

non-blocking => wait-free

every task solvable with Registers is solvable with n MWMR registers

Wait free?

- In the previous algorithm: a process may never succeed to Write
  - each time it writes on a register another process writes the same register
- To obtain wait-freeness:
  - two arrays of shared MWMR registers
    - W (working registers, size n-1)
    - PR (personal registers, size n+1)

wait free..

- as before a view contains everything the process has seen and views are written in registers.
- (n-1) W working registers as before: each process writes its view with x in all these registers
- each process tries to get one of the PR registers and to be the only one writing only in this register
- then
  - value x is written in n-1 + 1 registers =>no covering
  - eventually each process « has » its own register in PR
    => wait free (no process may crush the value and every process will see x and add x in its view)

Result:

2n (in fact 2n-1) MWMR registers are sufficient to wait-free emulate n SWMR

lower-bound?

- Remark: here there is no renaming of the processes

Adaptive

- When the number of processes is not fixed
  - adaptive algorithms: the number of registers depends on the number of participating processes not on the total number of processes
- Result

adaptive algorithm with 3n registers for emulating n SWMR registers

- formal proof written in TLA+

adaptive algorithm

- more difficult... R1, R2, R3
- with k+1 registers for k participants, each process gets a set of processes such if p and q get Sp and Sq and |Sp|=|Sq| then Sp=Sq
- write Sp in R2[|Sp|], read R2, evaluate the number w of processes
- use w registers of R3 to write the value
- if no new processes in R2 the write is finish.

3k +1 registers

Extensions

π

Tasks

- we can solve 'classical' solvable task wait free (resp. obstruction free) with n MWMR registers, what can be done with less than n registers?

k-set agreement

- each process has an input value and has to decide (irrevocably)
  - at most k values are decided
  - if p decides v then v is the initial value of some process
  - if a process takes an infinite number of steps then it decides
- (when k=1 : consensus
  when k=n-1 : set agreement)

k-set agreement

- Recall: (in the standard model for which each process has its own register)
- there is no wait free solution for the k-set agreement
- but obstruction free (one process is alone for enough time) solutions for the k-set agreement

Results and conjecture

- (obstruction free) algorithm to solve k-set agreement with n-k+1 MWMR registers
- with 2 registers set-agreement (lower bound)
  - Conjecture: k-set agreement solvable if and only if r≥n-k+1 registers
  - (lower bound for consensus n?)

Conclusion

- with more than n registers, many results and many things are (now) well understood
- with less than registers, with failures ????
  - Consensus $\sqrt{n}$ registers versus n
  - timestamp...
- conjecture:
  - k-set agreement solvable if and only if r≥n-k +1 registers
    - consensus n registers --- set-agreement 2 registers

- Thank you for your attention!!