Locality and Wait-free: Coloring

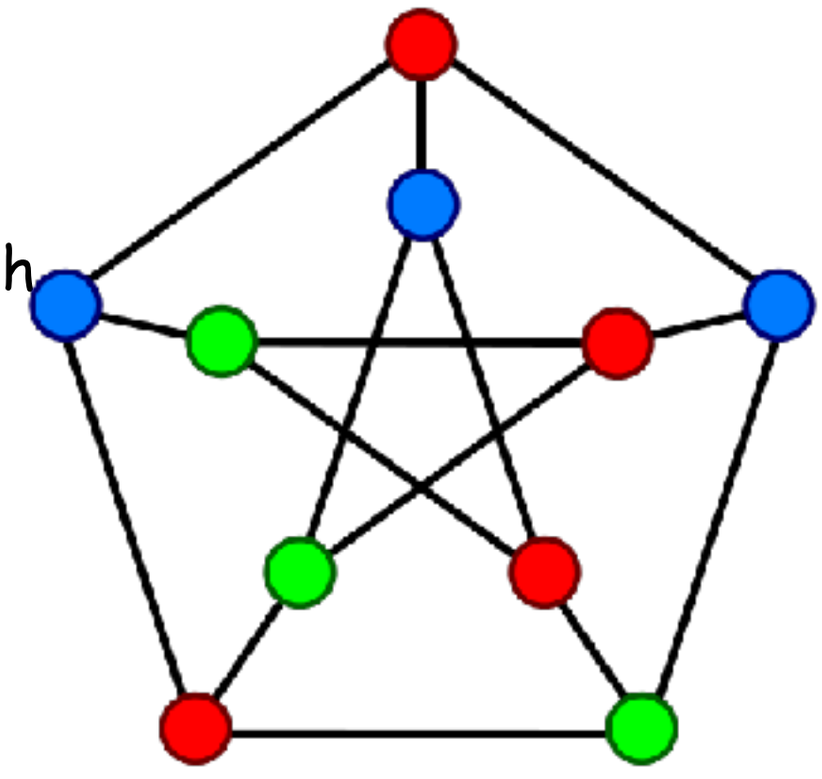
A. Castañeda, C. Delporte, H. Fauconnier, S. Rajsbaum,
M. Raynal

Motivations

- Red-Blue:
  - log* versus tasks
  - local versus Wait-free
  - ANR displexity
- wait-free and communication graphs?
- What could be local wait-free?

$\mathcal{LOCAL}$ and Graph coloring

- graphe G=(V,E)
  - coloring V->C s.t.:V(a)≠V(b) if (a,b)∈C
    - line: 2 colours, ring 3 colours (parity)
- Locality ($\mathcal{LOCAL}$ model)
  - at each round processes exchange messages with the neighbours
  - (d rounds : information from nodes at distance d)
  - coloring on ring and trees:
    - O(log*n rounds) (and Ω(log*n)) Cole-Vishkin

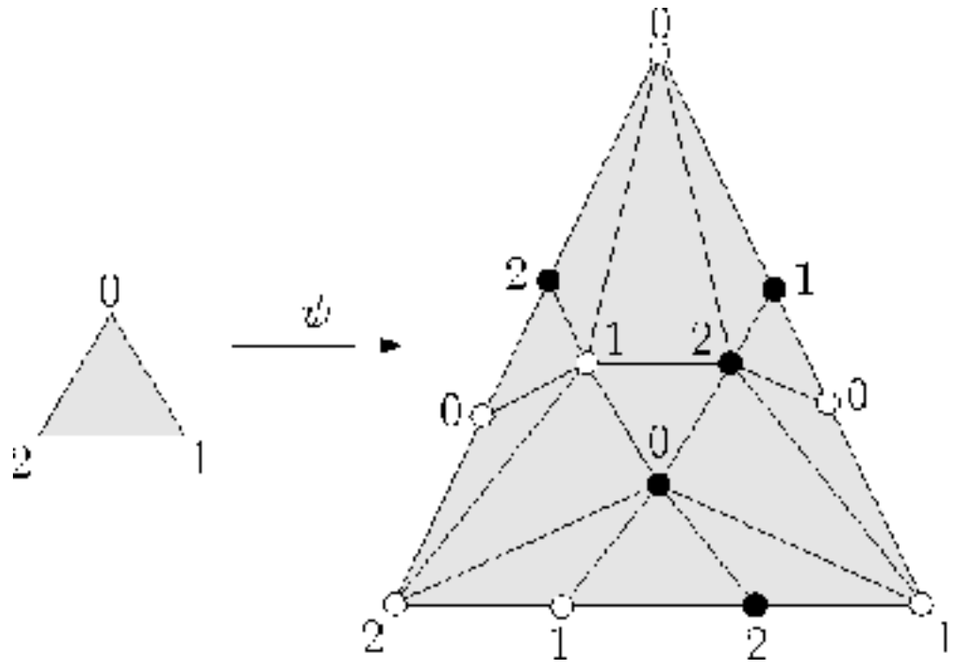


Wait-free

- asynchronous processes.
  (Atomic) Shared Memory

- no wait: a process cannot wait for another process
  - any set of processes are able to terminate alone
  - (any number of process crashes)

- Wait-Free computation:
  - Tasks Input->Output
  - Many results…



Cole-Vishkin

- for paths, rings and trees colouring in 3 colours in O(log*n)

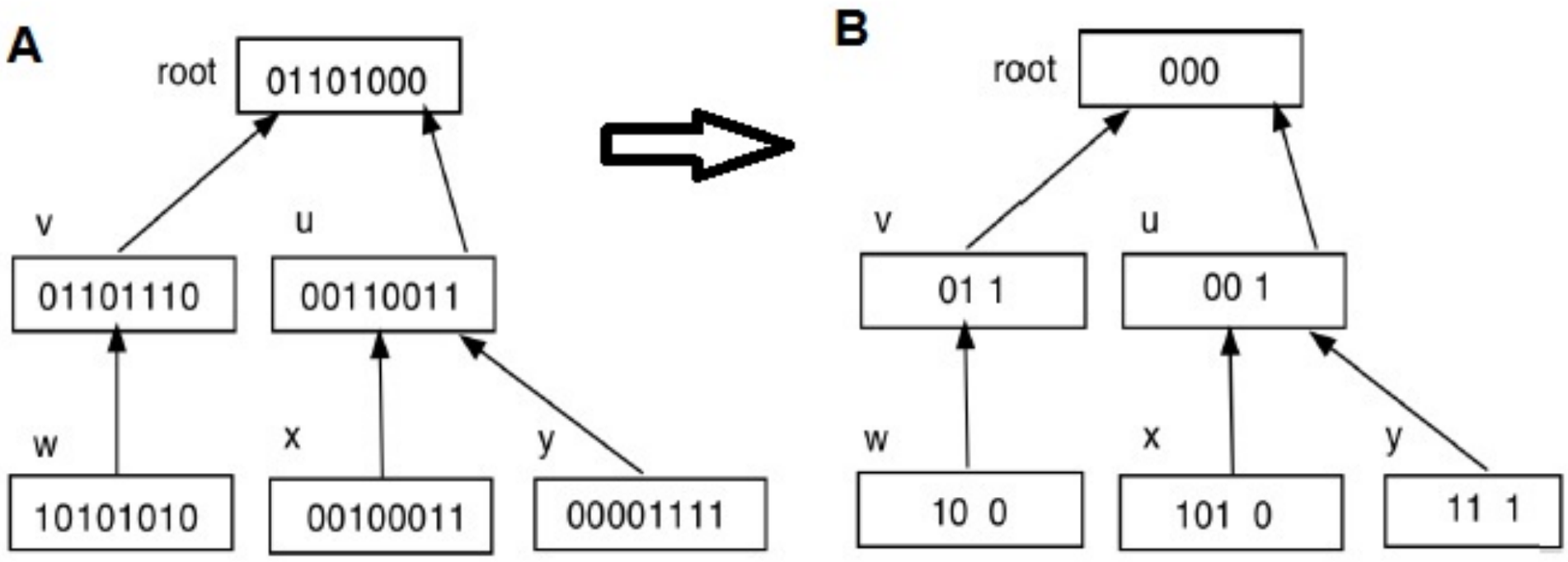
- 2 phases:
  - phase 1: reduce the number of colours to 6 (time : log* n)
  - phase 2: reduce the number of colours from 6 to 3 (time 3)

- $\log^* n$ = 0 si $n \leq 1$
  = 1+ $\log^*(\log n)$

x	$\lg^* x$
$(-\infty, 1]$	0
$(1, 2]$	1
$(2, 4]$	2
$(4, 16]$	3
$(16, 65536]$	4
$(65536, 2^{65536}]$	5

Algorithm

1: **Each node** v concurrently executes the following code:
2: Run Algorithm for $\log^* n$ rounds "6-Color" **Cole Vishkin Algorithm**
1: Assume that initially the vertices are legally colored.
each label only has $\log n$ bits
2: The root assigns itself the label 0.
3: **Each** other **node** v executes the following code (synchronously in parallel)
4: send c_v to all children
5: **repeat**
6: receive c_p from parent
7: interpret c_v and c_p as little-endian bit-strings: $c(k), \ldots, c(1), c(0)$
8: let i be the smallest index where c_v and c_p differ
9: the new label is i (as bitstring) followed by the bit $c_v(i)$ itself
10: send c_v to all children
11: **until** $c_w \in \{0, \ldots, 5\}$ for all nodes w



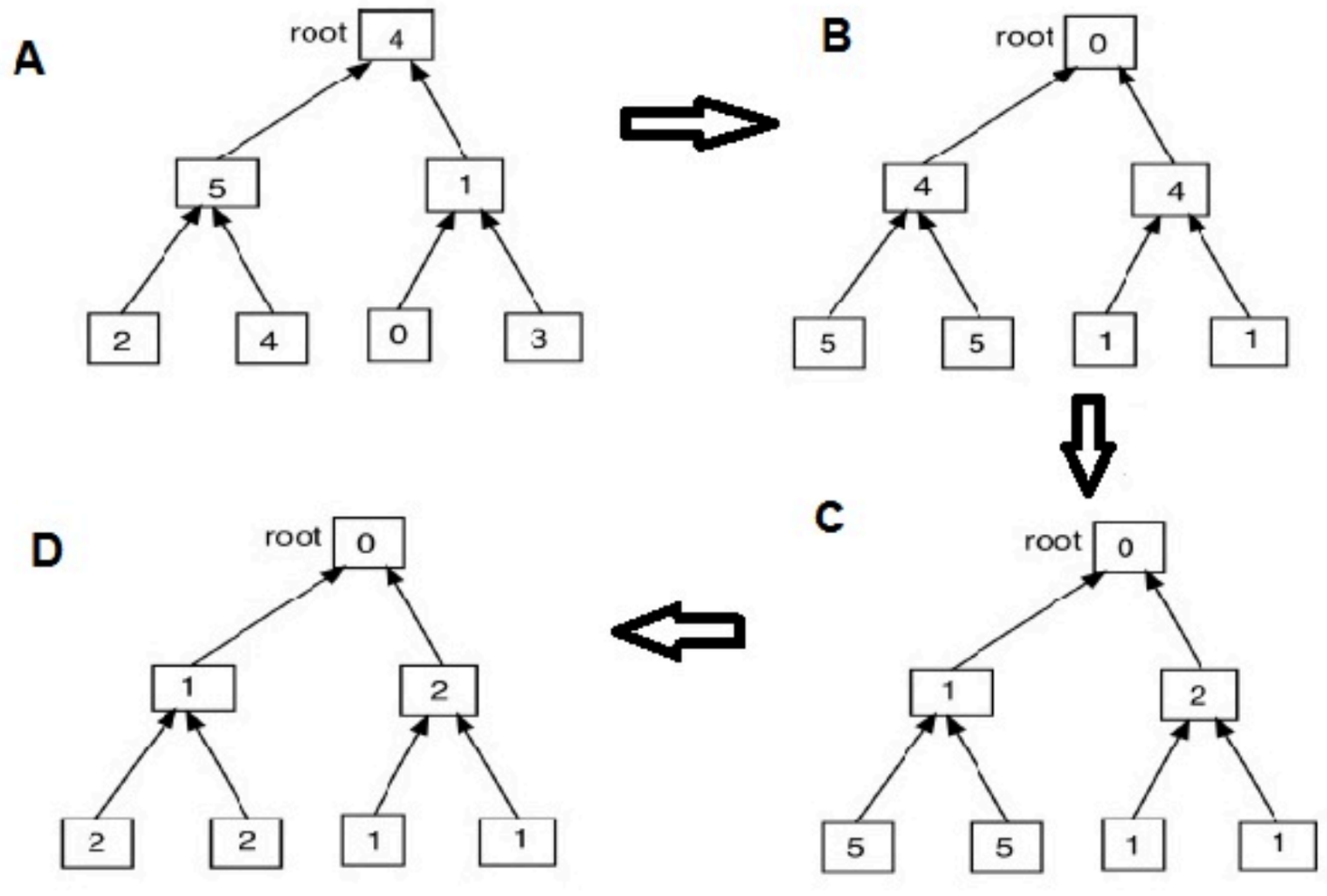
3: **for** $x = 5, 4, 3$ **do**
4: Perform subroutine Shift down

1: Root chooses a new (different) color from $\{0, 1, 2\}$
2: **Each** other **node** v concurrently executes the following code:
3: Recolor v with the color of parent

5: **if** $c_v = x$ **then**
6: choose new color $c_v \in \{0, 1, 2\}$ using subroutine **First Free**

Give v the smallest admissible color {i.e., the smallest node color not used by any neighbor}

7: **end if**
8: **end for**

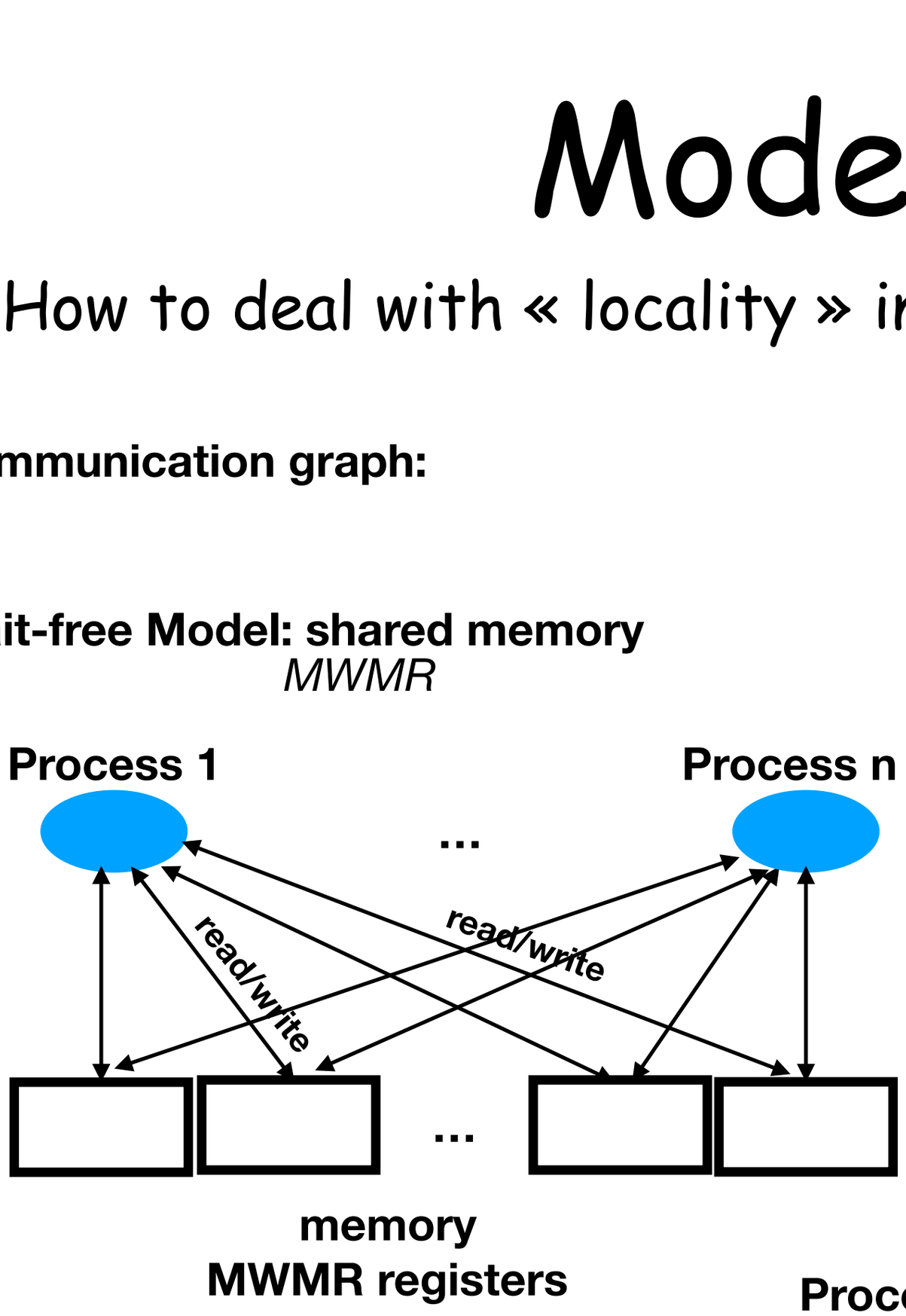


Model…

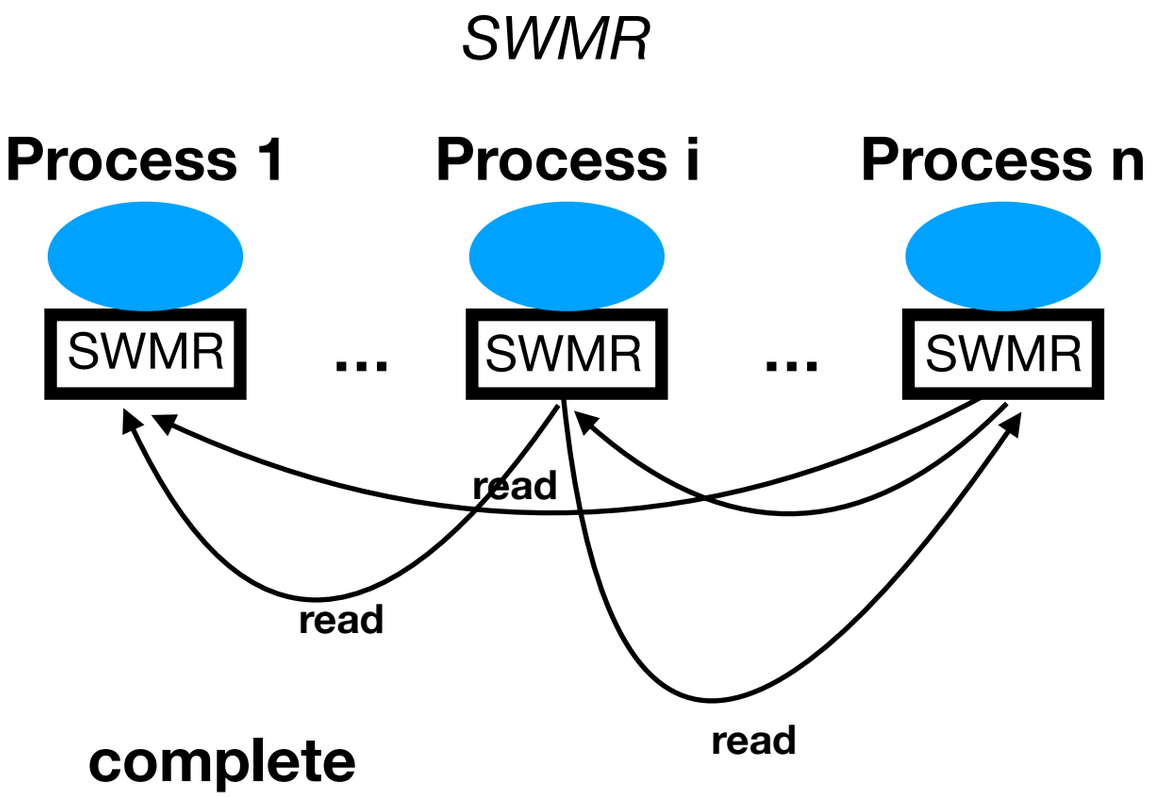
- How to deal with « locality » in shared memory?

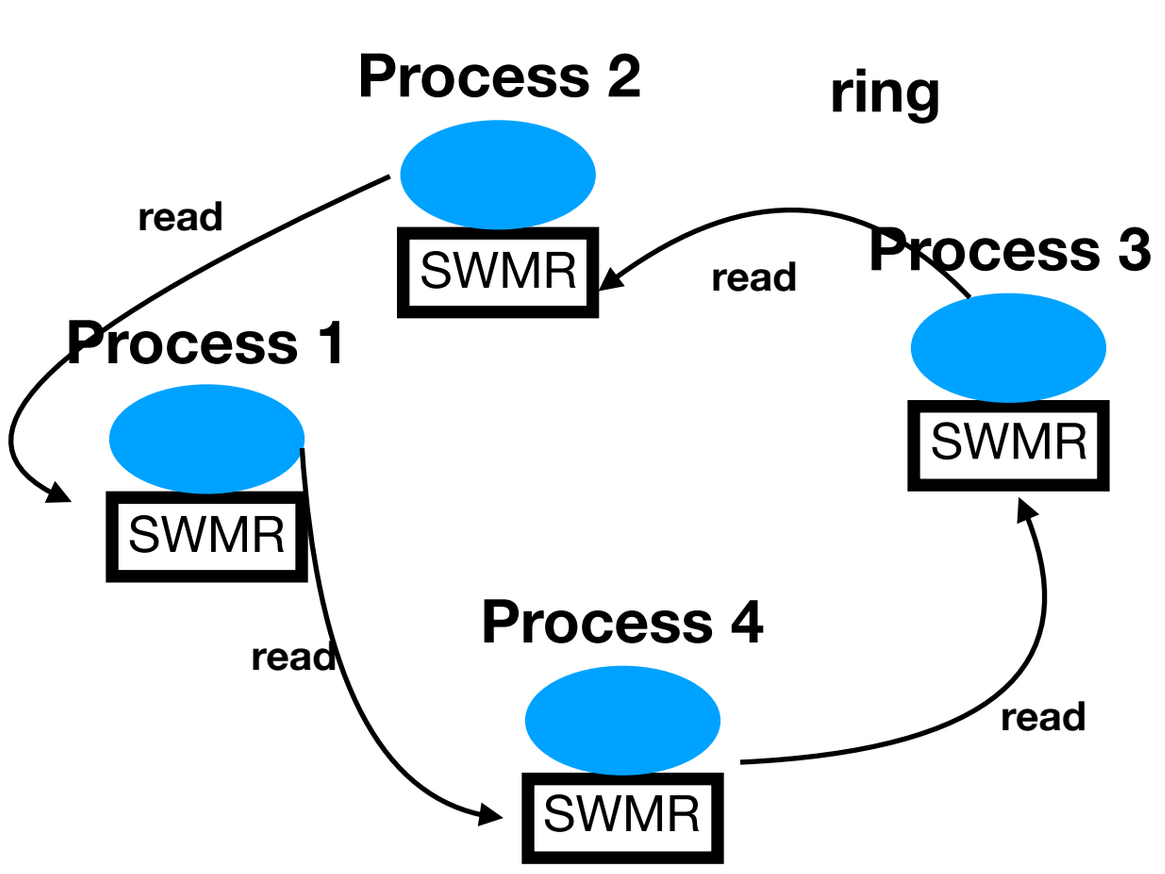
Communication graph:

Wait-free Model: shared memory
MWMR



communication

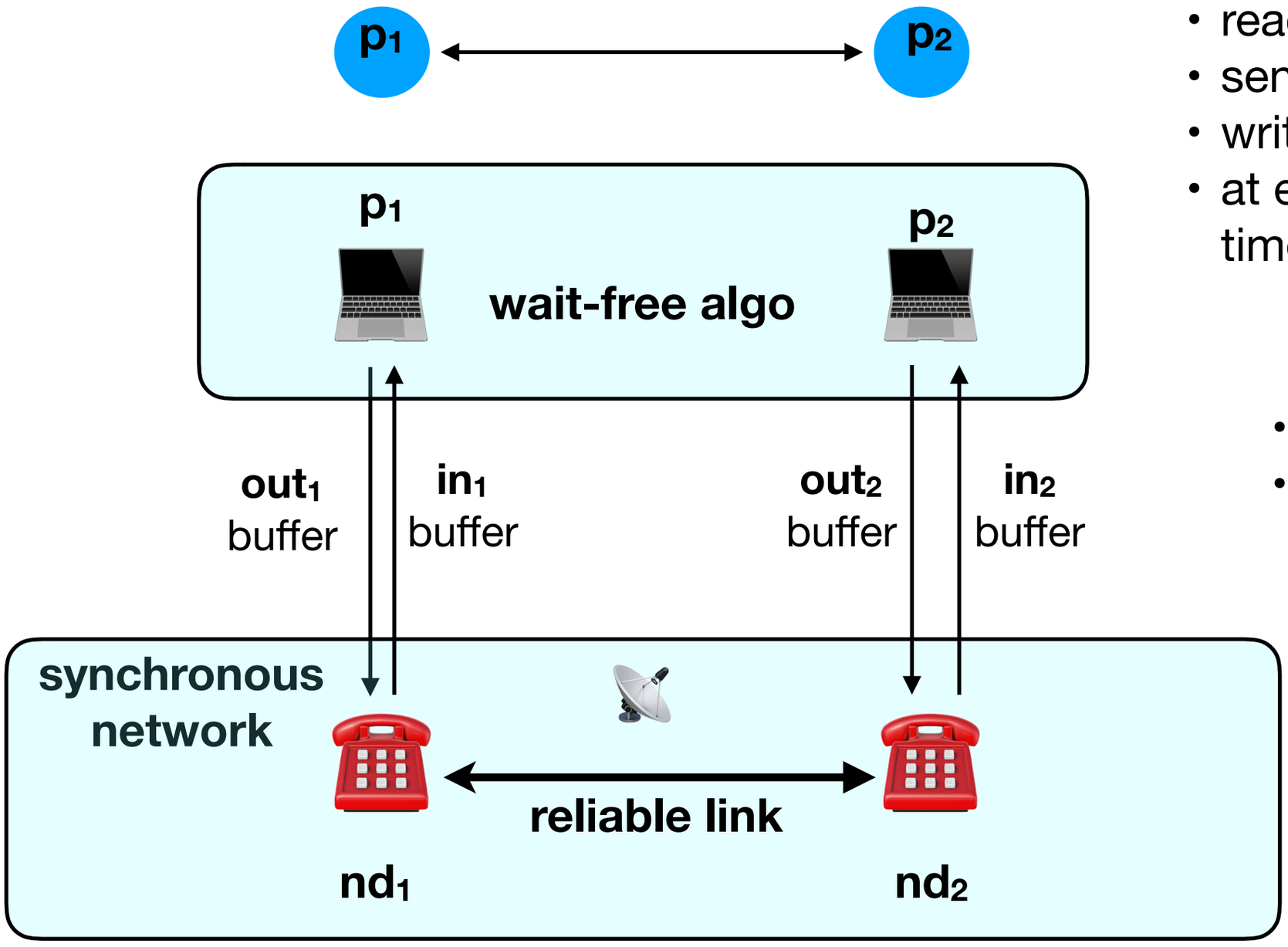




Model

- separate communication layer / processes
- Communication layer:
  - graph - synchronous communication (round by round)
  - Processes are asynchronous (may start at any time and be sleeping for any time) - communication by the way the communication layer (synchronous)
  - global clock

Model



At *each round*, nd_i :

- receives messages from its neighbours
- reads its buffer out_i
- sends all that to its neighbours
- writes that in in_i
- at each message is associated a timestamp *ts* from a global clock

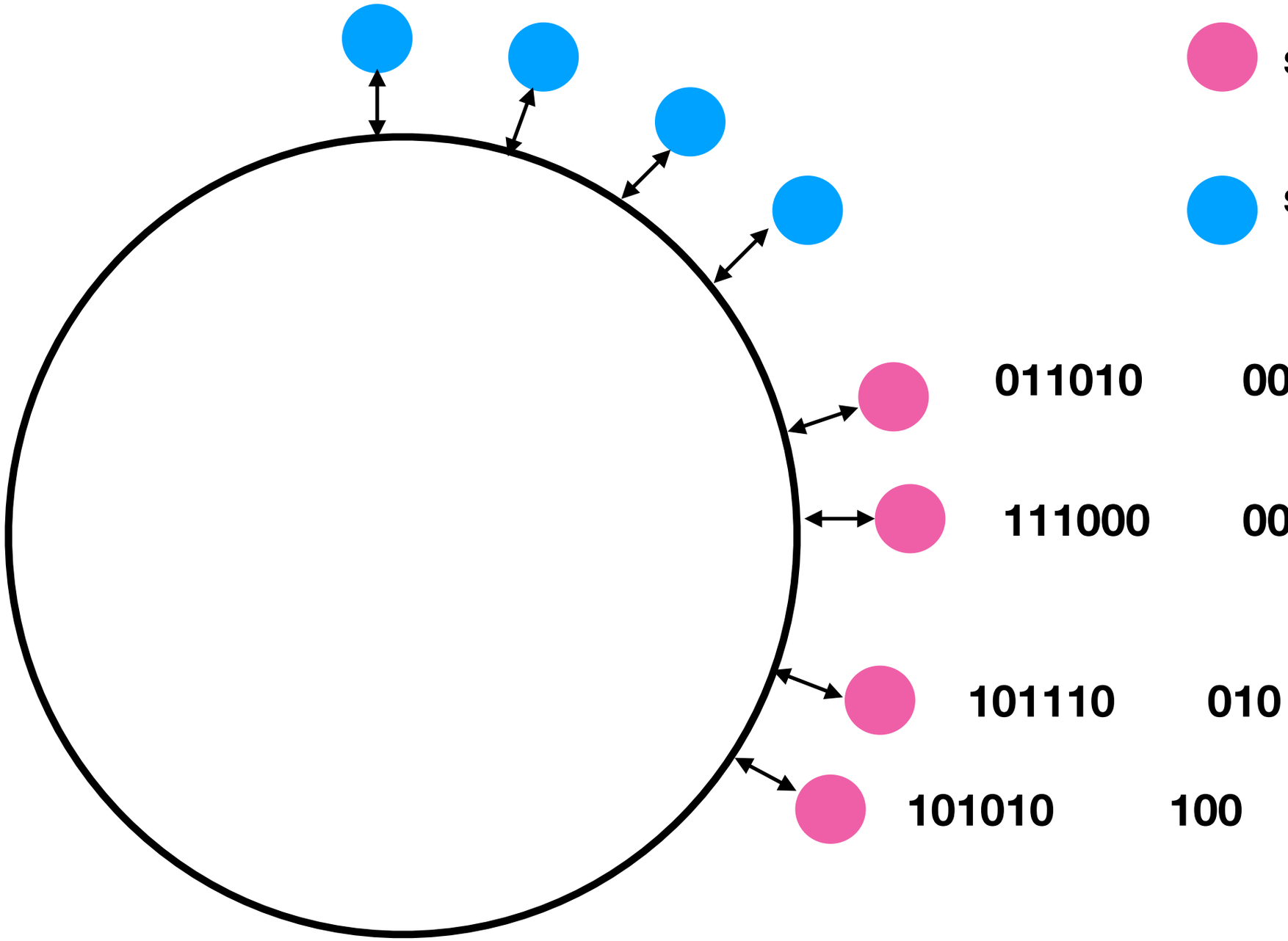
Process p_i is asynchronous

- reads in_i
- writes out_i

If p_i starts the algorithm at time *ts*, and p_j starts at time *ts'*,

if $d(p_j,p_i)<ts-ts'$ then p_i gets all messages sent by p_j from time *ts'* to $ts'-d(p_j,p_i)$

If pi starts at time t, after D units of time pi can have information from processes in the graph at distance D


start at the same time
start at the same time
011010
000
111000
001
101110
010
101010
100

6 colors…

========== Part 1: reduction from n colors to 6 colors ======

(01) **when** $r = st_i, st_i + 1, ..., (st_i - 1) + \log^* n$ **do**
(02) **begin synchronous round**
(03) send COLOR$(0, st_i, color_i)$ to $next_i$ and $pred_i$;
(04) receive msg_pred_i from $pred_i$;
(05) if $(msg\_pred_i = \text{COLOR}(0, st_i, col))$
(06) **then** x= first position (starting right at 0) where $color_i$ and col differ;
(07) $color_i \leftarrow$ bit string encoding the binary value of x followed at
(08) its right by b_x (first bit of $color_i$ where $color_i$ and col differ)
(09) **else** p_i has no predecessor (it is an end process of its unit segment) it
(10) considers $fict\_pred_i$ as its predecessor and executes lines 06-08
(11) **end if**;
(12) **end synchronous round**;
% Here $color_i \in \{0, 1, \cdots, 5\}$

3 colors

$==========$ Part 2: reduction from 6 to 3 colors $=======$

(13) **when** $r = (st_i - 1) + \log^* n + 1, (st_i - 1) + \log^* n + 2, (st_i - 1) + \log^* n + 3$ **do**
(14) **begin synchronous round**
(15) send COLOR$(0, st_i, color_i)$ to $pred_i$ and $next_i$;
(16) $color\_set \leftarrow \emptyset$;
(17) **if** COLOR$(0, st_i, color_p)$ received from $pred_i$
then $color\_set \leftarrow color\_set \cup color\_p$ **end if**;
(18) **if** COLOR$(0, st_i, color_n)$ received from $next_i$
then $color\_set \leftarrow color\_set \cup color\_n$ **end if**;
(19) **let** k **be** $r - (st_i - 1 + \log^* n) + 2$; % $k \in \{3, 4, 5\}$ %
(20) **if** $(color_i = k)$ **then** $color_i \leftarrow$ any color from $\{0, 1, 2\} \setminus color\_set$ **end if**
(21) **end synchronous round**;

$==========$

Change the left-end

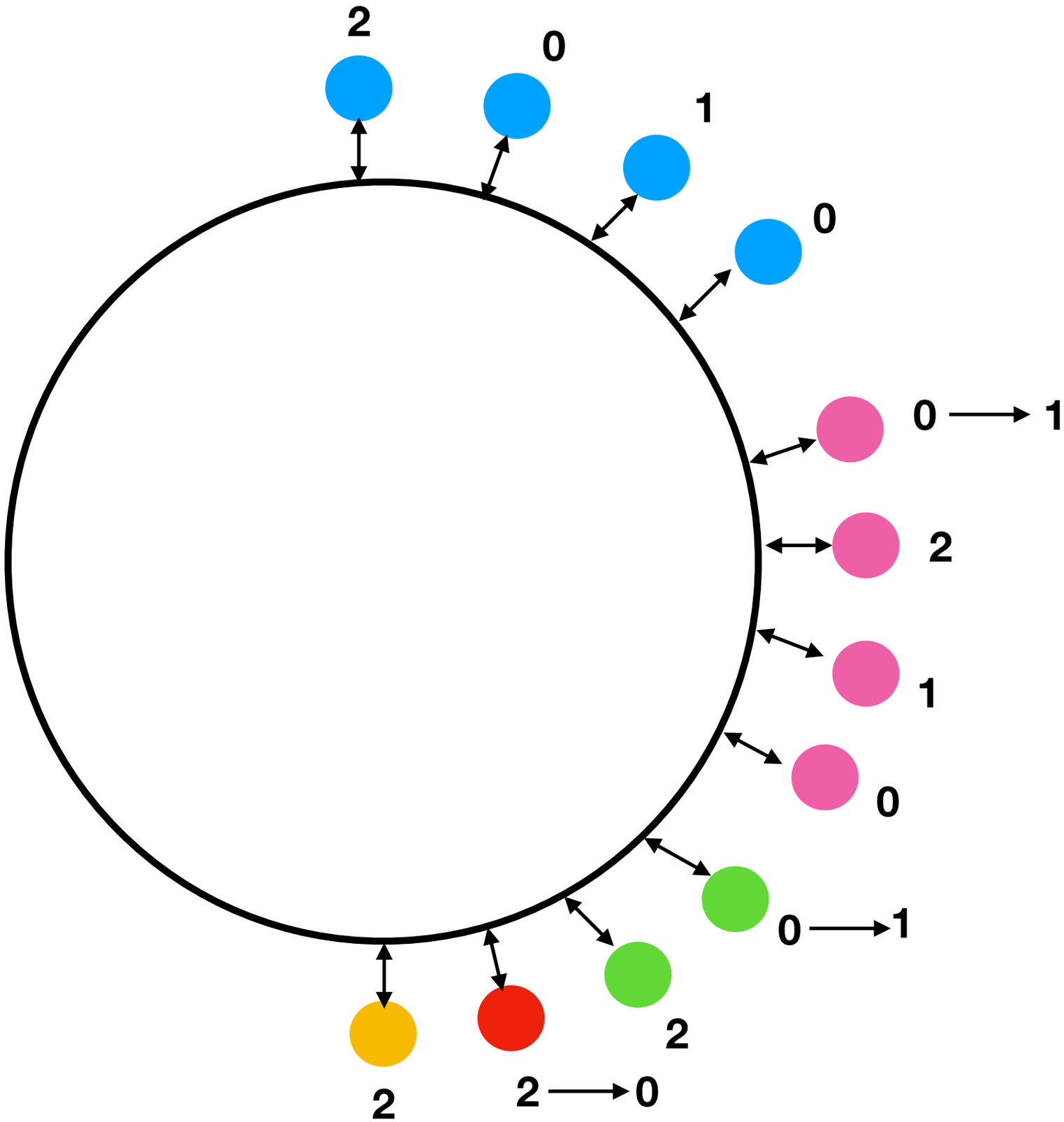
== Part 3 : $color_i$ can be changed only if p_i is the left end of its unit-segment

(22) **when** $r = (st_i - 1) + \log^* n + 4$ **do**
(23) **begin synchronous round**
(24) send COLOR$(1, color_i)$ **to** $pred_i$ and $next_i$;
(25) **for each** $j \in \{1, 2, 3\}$ **do**
(26) **if** (COLOR$(j, color)$ received from $pred_i$ in a round $\leq r$)
then $color_i[j, pred_i] \leftarrow color$ **end if**;
(27) **if** (COLOR$(j, color)$ received from $next_i$ in a round $\leq r$)
then $color_i[j, next_i] \leftarrow color$ **end if**
(28) **end for**;
(29) **if** $(st_i > st_i[pred_i])$ **then** % p_i has not priority
(30) **case** $(st_i = st_i[next_i])$ **then**
$color_i \leftarrow$ a color in $\{0, 1, 2\} \setminus \{color_i[2, pred_i], color_i[1, next_i]\}$
(31) $(sl_i > sl_i[nextl_i])$ **then**
$color_i \leftarrow$ a color in $\{0, 1, 2\} \setminus \{color_i[2, pred_i], color_i[2, next_i]\}$
(32) $(st_i < st_i[next_i])$ **then** $color_i \leftarrow$ a color in $\{0, 1, 2\} \setminus \{color_i[2, pred_i]\}$
(33) **end case**
(34) **end if**
(35) **end synchronous round**;

Change the right end

== Part 4: $color_i$ can be changed only if p_i is the right end of its unit-segment

(36) **when** $r = (sl_i - 1) + \log^* n + 5$ **do**
(37) **begin synchronous round**
(38) send COLOR$(2, color_i)$ to $pred_i$ and $next_i$;
(39) same statements as in lines 25-28;
(40) **if** $(st_i > st_i[next_i])$ **then** % p_i has not priority
(41) **case** $(st_i = st_i[pred_i])$ **then**
$color_i \leftarrow$ a color in $\{0,1,2\} \setminus \{color_i[2, pred_i], color_i[3, next_i]\}$
(42) $(st_i > st_i[pred_i])$ **then**
$color_i \leftarrow$ a color in $\{0,1,2\} \setminus \{color_i[3, pred_i], color_i[3, next_i]\}$
(43) $(st_i < st_i[pred_i])$ **then** $color_i \leftarrow$ a color in $\{0,1,2\} \setminus \{color_i[3, next_i]\}$
(44) **end case**
(45) **end if**
(46) **end synchronous round**;
== Additional round to inform the neighbors that will start later
(47) **when** $r = (st_i - 1) + \log^* n + 6$ **do send** COLOR$(3, color_i)$ to $pred_i$ and $next_i$;
(48) return$(color_i)$.


2
0
1
0
0 → 1
2
1
0
0 → 1
2
2
2 → 0

asynchronous

- only the first message is interesting
- communication only the starting message
- each process may simulate (alone) its part of the algorithm with only these starting messages
- (a process may sleep for a while)

Conclusion

- log* for blue people 😉
- and for a tree?
  - guess: impossible but…
- Wait-free on network?