BAKERY PROTOCOL

Mutual Exclusion Protocols

Darshana Nafde
204050005

IITB CS 766
9th Feb 2021

Introduction

- The **Bakery algorithm** is one of the simplest known solutions to the mutual exclusion problem for the general case of N process
- Devised by Leslie Lamport
- Based on token system in bakery and banks. Preserves the first come first serve property
- Before the bakery algorithm, people believed that the mutual exclusion problem was unsolvable--that you could implement mutual exclusion only by using lower-level mutual exclusion constructs
- A New Solution of Dijkstra's Concurrent Programming Problem Leslie Lamport Massachusetts Computer Associates, Inc. Communications of the ACM August 1974 Volume 17 Number 8 http://lamport.azurewebsites.net/pubs/bakery.pdf
- Proving the Correctness of Multiprocess Programs

  IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. SE-3, NO. 2, MARCH 1977

  http://www.cis.umassd.edu/~hxu/courses/cis481/references/Lamport-1977.pdf

Algorithm

```
choosing[i] = 1;
number[i]= 1 + maximum(number[1],...number[N]);
choosing[i]=0;
for(j=1:N)
{
   while(choosing[j]){}
   while(number[j] != 0 && (number[j],j)<(number[i],i)){}
}
CRITICAL SECTION
number[i]=0;
```

Demo

Correctness Proof

- **Assertion 1.** If processors i and k are in the bakery and i entered the bakery before k entered the doorway, then number [i] < number [k].
- **Proof.**
  - *By hypothesis, number [i] had its current value while k was choosing the current value of number [k].*
  - *Hence, k must have chosen number [k] > =1 + number [i]*

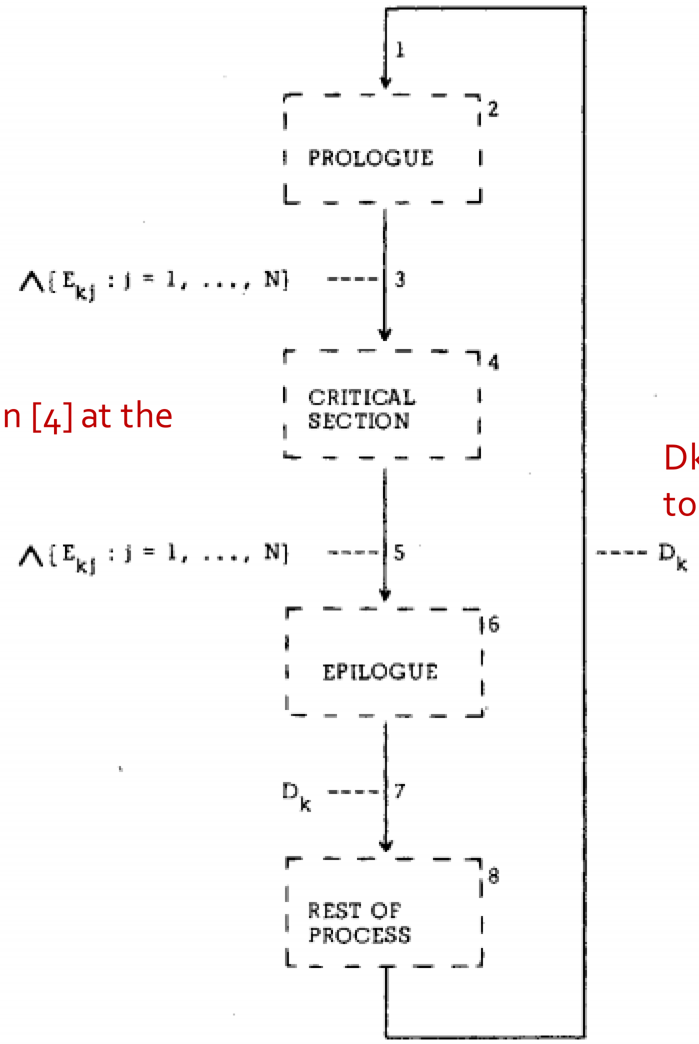
Correctness Proof

- **Assertion 2.** If processor i is in its critical section, processor k is in the bakery, and k <> i, then (number [i], i) < (number [k], k).
- **Proof:**
  - *k is done <u>choosing before</u> i starts its check, number[k] will have clear a value >, < or = number[i] and i will proceed accordingly*
  - *If k is <u>choosing while</u> i needs to decide if it can go ahead of k, it should wait till choosing is done, so that i is sure about its decision*
  - *If k <u>didn't start choosing</u>, when i is deciding if it can go ahead of k, even if k decides to start choosing now its number[k] < number[i]. From Assertion 1*
  - *If both k and i get the same number while choosing, then min(i, j) will proceed.*
  - *Thus, no two process can enter CS at same time*

Correctness Proof

- **Assertion 3.** Assume that only a bounded number of processor failures may occur. If no processor is in its critical section and there is a processor in the bakery which does not fail, then some processor must eventually enter its critical section.
- **Proof.**
  - *Assume that no processor ever enters its critical section.*
  - *Then there will be some time after which no more processors enter or leave the bakery.*
  - *At this time, assume that processor i has the minimum value of (number [i], i) among all processors in the bakery.*
  - *Then processor i must eventually complete the for loop and enter its critical section.*
  - *This is the required contradiction*

Two processes cannot be in [4] at the same time



Dk attached to all arcs of subroutine 8 and Ekj to all arcs of subroutine 4

Fig. 4. Stage 1 decomposition of Π_k.

Define Ekj such that if j ≠ k then Ekj ^ Ejk ^ πk € 4^ πj € 4 = False

The invariance of the interpretation containing the indicated assertions will imply that πk and πj cannot both be in their CS

Define $k \leq\leq j$: $(0 < n[k] < n[j]) \vee (0 = n[j] < n[k]) \vee (n[k] = n[j] \wedge k < j)$

πk can enter CS if $k \leq\leq j$

[2] = [2.1] U {2.2} U [2.3]

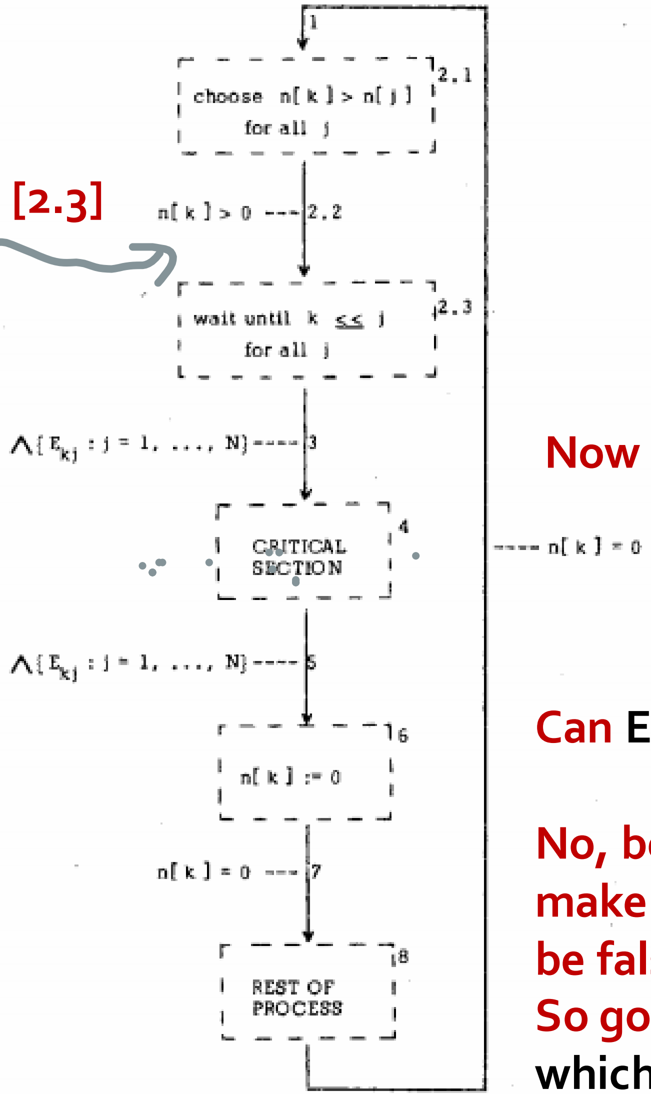


Fig. 5. Stage 2 decomposition of Π_k.

Now we have defined Dk : $n[k] = 0$

Can $E_{kj} = (n[k] > 0) \wedge k \ll j$?

No, because πj might be choosing a value which will make $j \ll k$ but not yet set, and once its set $k \ll j$ will be false.

So good $E_{kj} = (n[k] > 0) \wedge (k \ll j \vee \pi j$ is choosing $n[j]$ which will make $k \ll j)$

2.3 decomposed further

Rkj = n[k] > 0 and
if πj is not changing value of n[j],
then k <<j

Skj = n[k] > 0 and
if πj is choosing value of n[j],
then it will choose value > n[k]

Ekj = Rkj ^ Skj

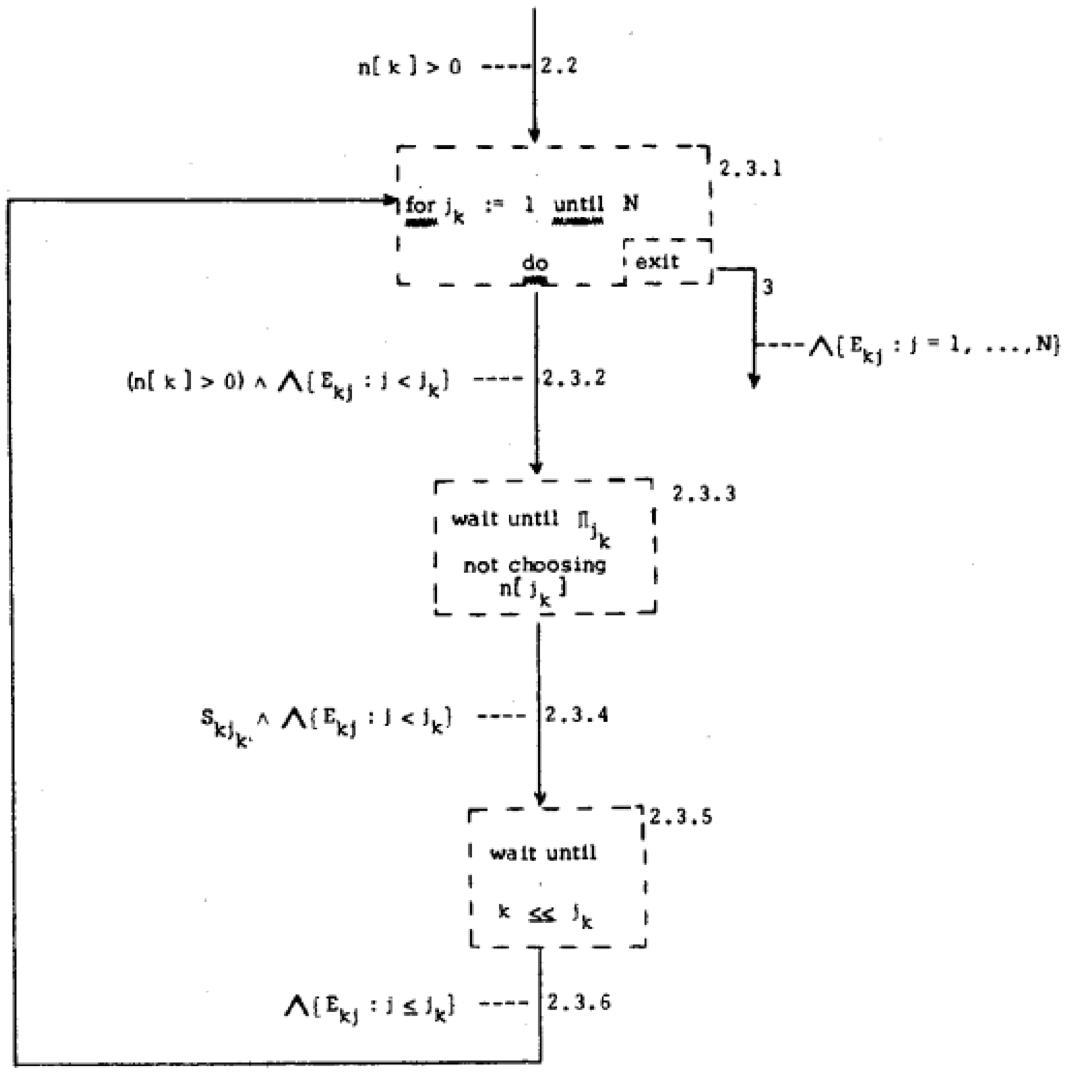


Fig. 6. Stage 3 decomposition of subroutine 2.3.

2.1 decomposed further, introducing cf array

- cf is initially false
- Modified only in 2.1.1 and 2.1.5

Rkj = (n[k] > 0) ^ [(πj not in [2.1.3] or [6]) => k << j]

Skj = (n[k]>0) ^ [(πj is in [2.1.3] => Tkj]

Tkj = function of πjs local variables
Tkj = true => either n[k] has not been read by [2.1.3] of πj or its current value was read

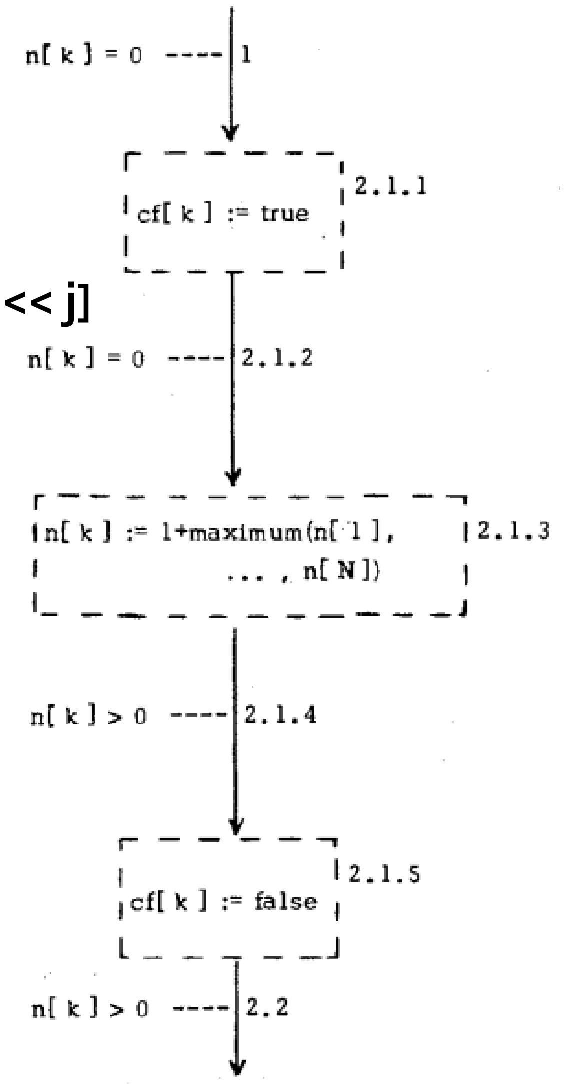


Fig. 7. Stage 4 decomposition of subroutine 2.1.

2.3.3 "wait until cf[i] = false" operation

Initial assertion:

^{πk is in [8] ^ n[k] = 0 ^ (cf[k]=0 :1≤k≤N)}

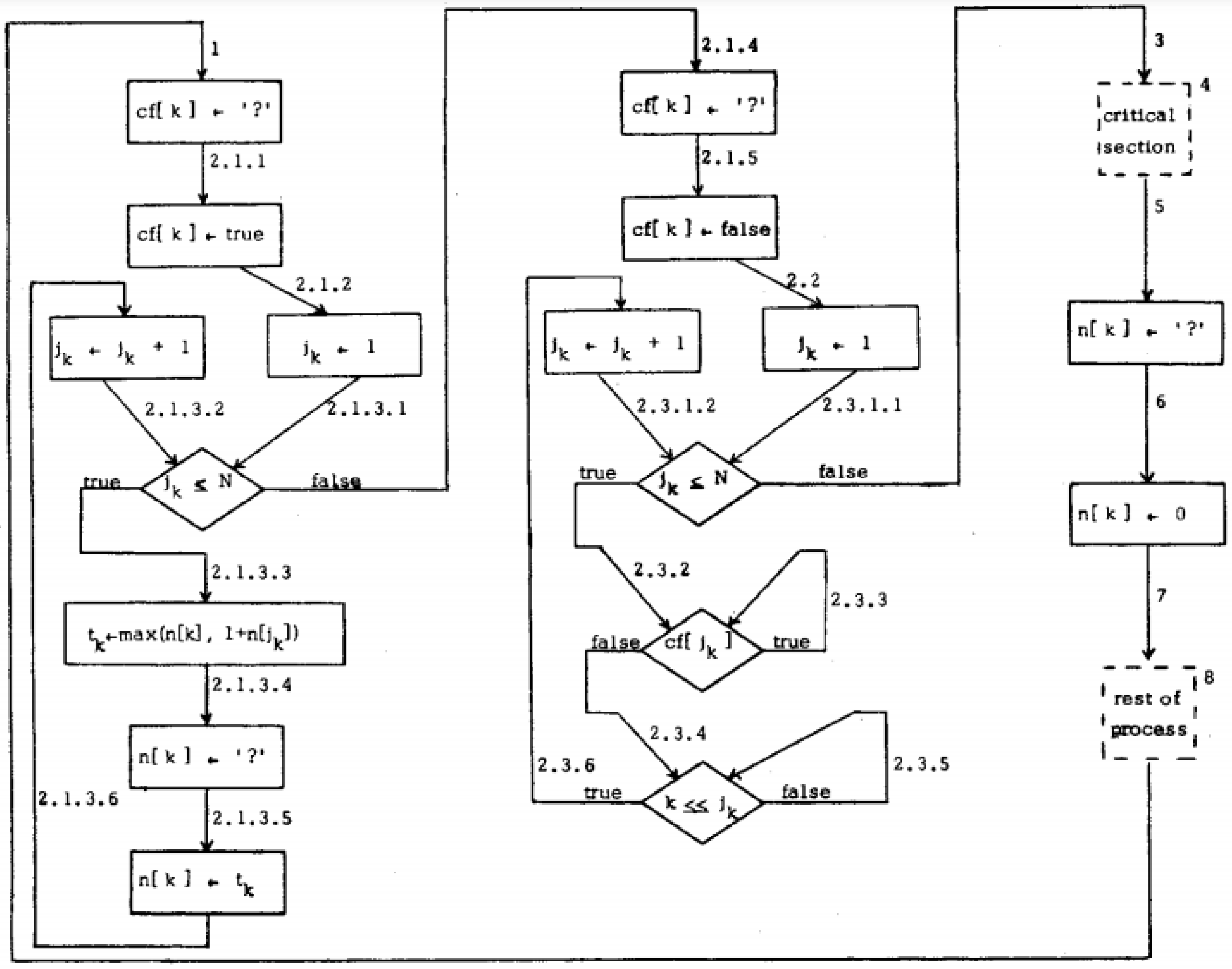
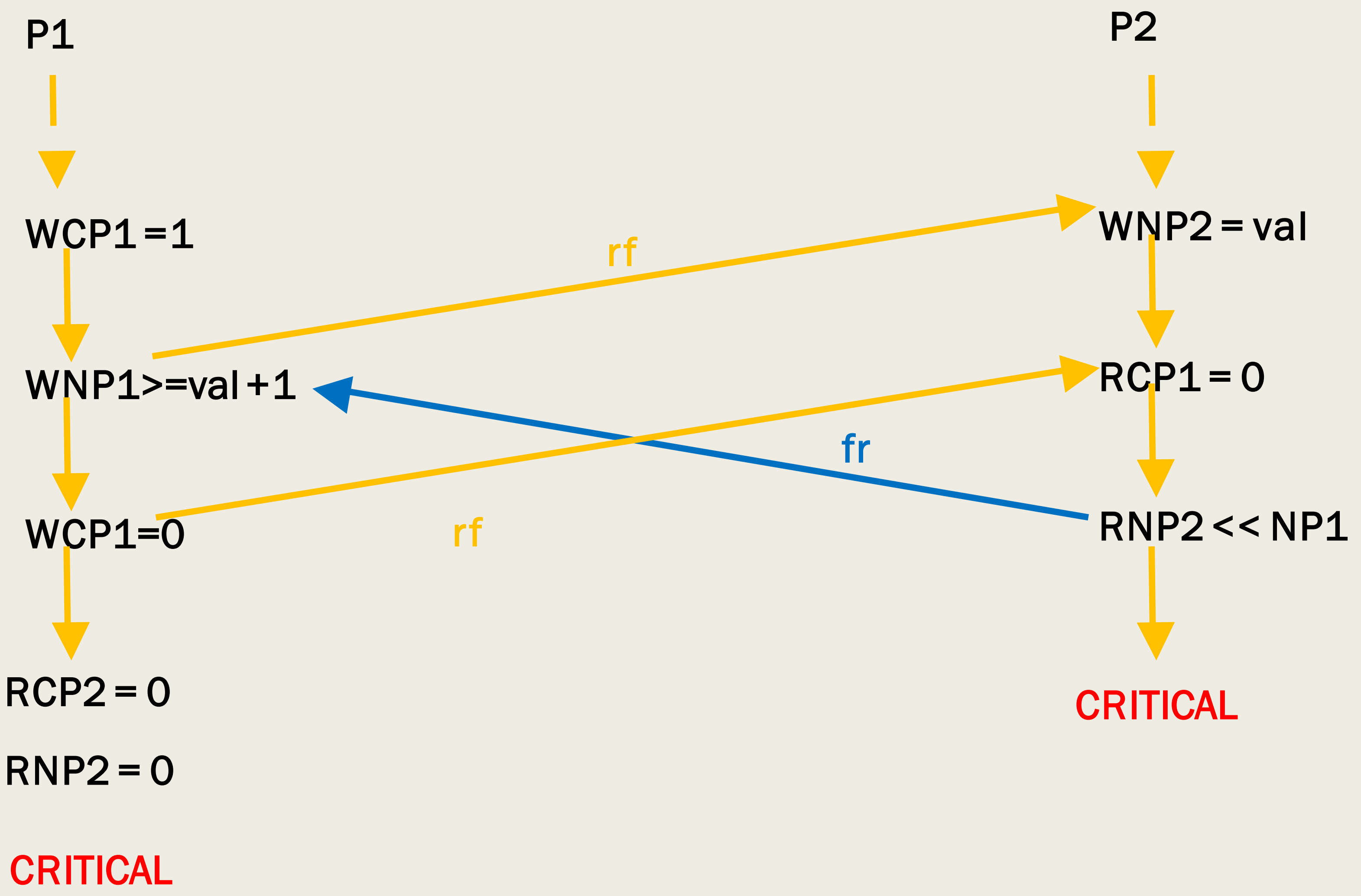


Fig. 8. The final program.

Tkj= ([(πk is in {2.1.3.2, 2.1.3.3}) ^(jj >k)] => [n[jj]]>n[k]) ^ ([(πj is in {2.1.3.4,2.1.3.5}) ^ (jj ≥ k)] => [tj] > n[k])^ ([(πj is in {2.1.3.6}) ^ (jj ≥ k)] => [n[jj] > n[k]])

Event Graph


P1
P2
WCP1 =1
WNP2 = val
rf
WNP1>=val +1
RCP1 = 0
fr
WCP1=0
rf
RNP2 << NP1
RCP2 = 0
RNP2 = 0
CRITICAL
CRITICAL

Deadlock Freedom

Deadlock situation on a resource can arise if and only if all of the following conditions hold simultaneously in a system:

- Mutual exclusion: Processes are using ME resources
- Hold and Wait
- No preemption
- Circular wait

In Bakery, concurrent reads are non interferring and concurrent writes are impossible

Thank You