Exclusive access

Support of sharing of resources - critical section

Requests

- Process, having resource assigned, has to unlock it before assignment to other process.
- Processes' requirements has to be performed in their logical time.
- If all processes unlock the critical section in a finite time, then every request is fullfiled in the finite time.

Exclusive access

Lamport's bakery algorithm

```
Entering: array [1..NUM_THREADS] of bool = {false};
Number: array [1..NUM_THREADS] of integer = {0};

lock(integer i)
{
  Entering[i] := true;
  Number[i] := 1 + max(Number[1], ..., Number[NUM_THREADS]);
  for (integer j := 1; j <= NUM_THREADS; j++) {
    while (Entering[j])
      while ((Number[j] != 0) && (Number[j] < Number[i]))
    }
  Entering[i] := false;
}

unlock(integer i) {
  Number[i] := 0;
}
```

Exclusive access

Lamport's bakery algorithm

```
Thread(integer i) {
  while (true) {
    lock(i);
      // The critical section goes here...
    unlock(i);
      // non-critical section...
  }
}
```

Exclusive access

Algorithms on the full graph

Lamport

- simple/basic algorithm, 3(n-1) messages/reguest

Ricart-Agarwala

- delayed acknowledgments, 2(n-1) messages/reguest

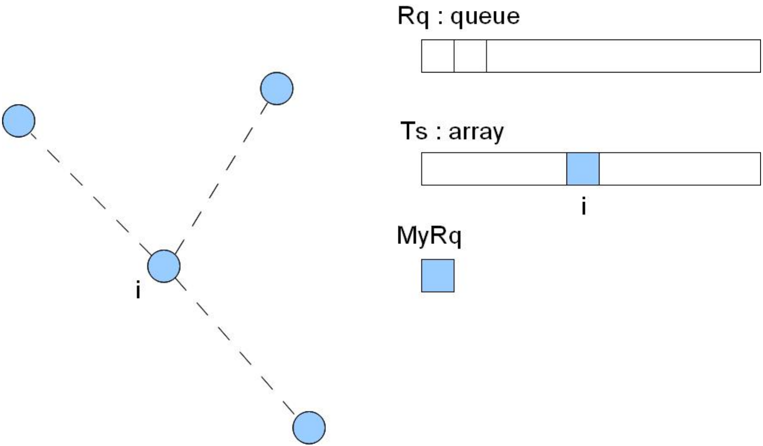
Carvalho-Roucairol

- access credits, 0 - 2(n-1) messages/reguest

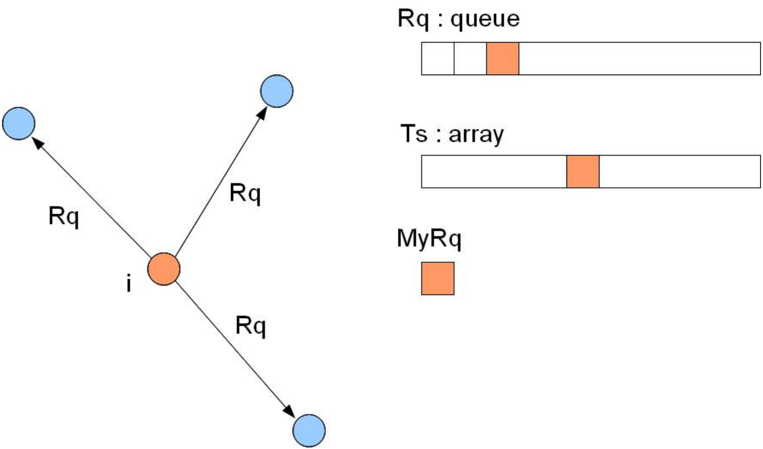
Ricart-Agarwala

- token passing, n messages/reguest

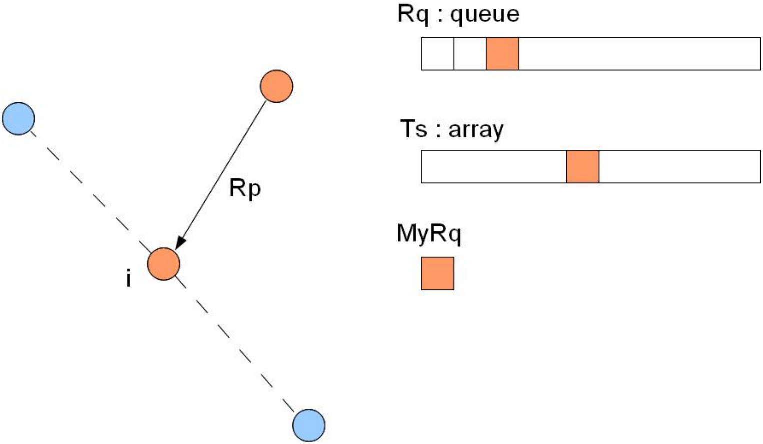
Lamport



Lamport



Lamport



Rp

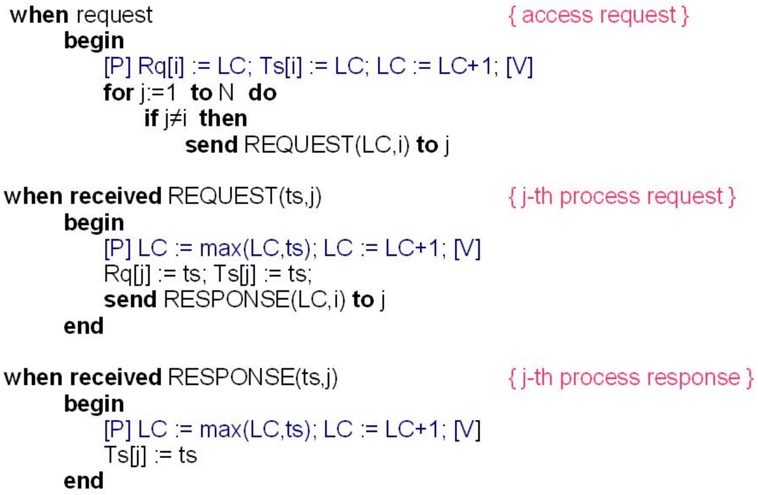
i

Rq : queue

Ts : array

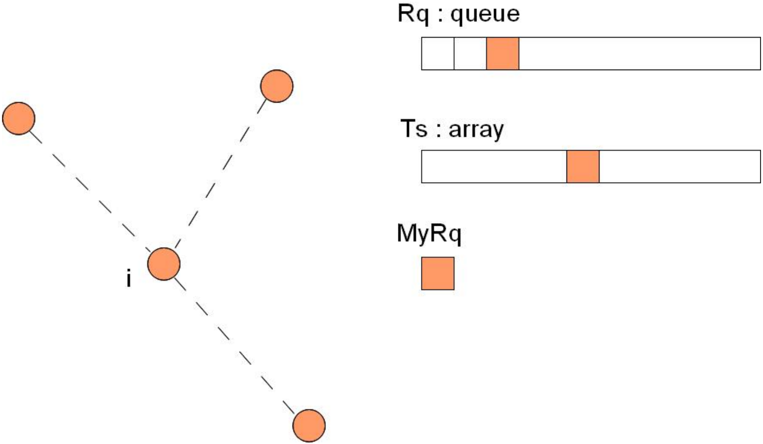
MyRq

Lamport



```
when request                                      { access request }
    begin
        [P] Rq[i] := LC; Ts[i] := LC; LC := LC+1; [V]
        for j:=1  to N  do
            if j≠i  then
                send REQUEST(LC,i) to j

when received REQUEST(ts,j)                       { j-th process request }
    begin
        [P] LC := max(LC,ts); LC := LC+1; [V]
        Rq[j] := ts; Ts[j] := ts;
        send RESPONSE(LC,i) to j
    end

when received RESPONSE(ts,j)                      { j-th process response }
    begin
        [P] LC := max(LC,ts); LC := LC+1; [V]
        Ts[j] := ts
    end
```

Lamport

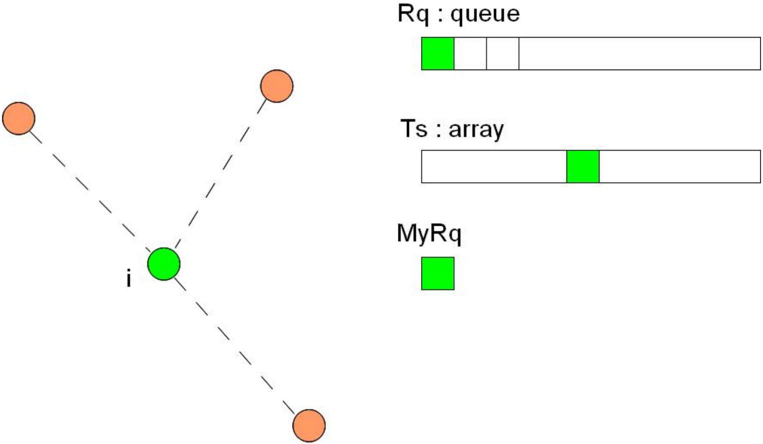


Rq : queue

Ts : array

MyRq

Lamport

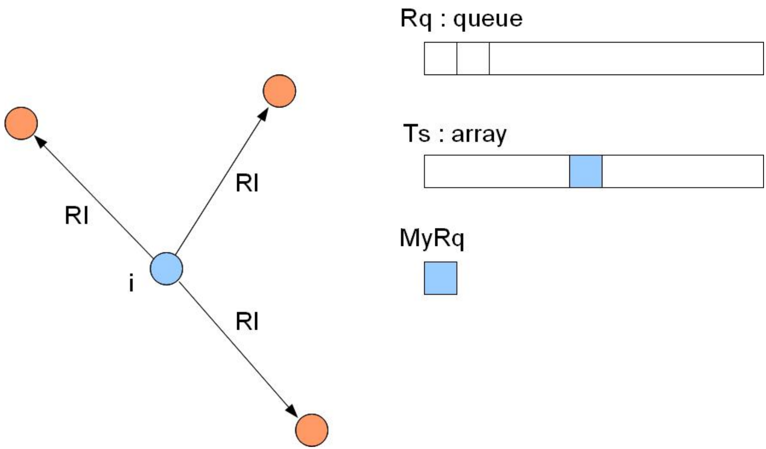


Rq : queue

Ts : array

MyRq

Lamport

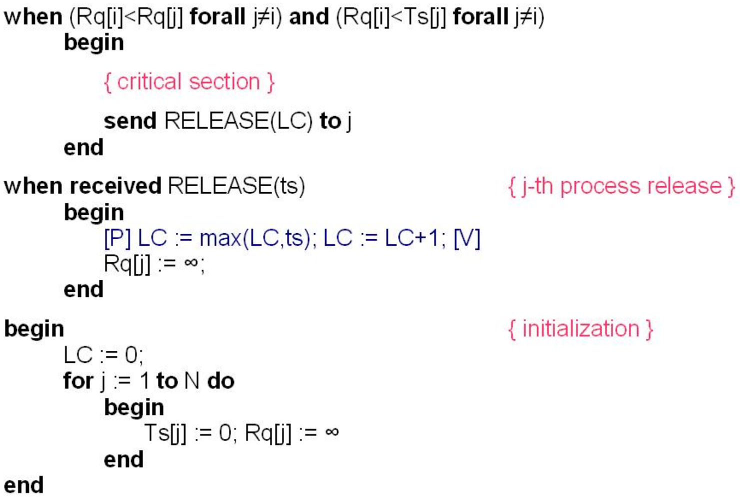


Rq : queue

Ts : array

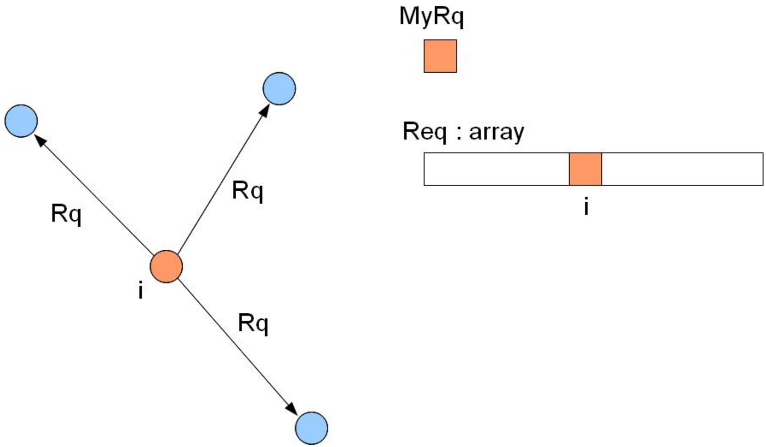
MyRq

Lamport

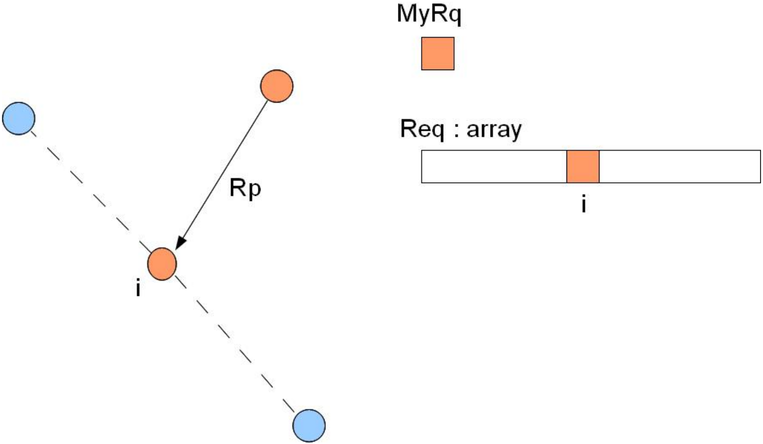


```
when (Rq[i]<Rq[j] forall j≠i) and (Rq[i]<Ts[j] forall j≠i)
      begin
          { critical section }

          send RELEASE(LC) to j
      end

when received RELEASE(ts)                         { j-th process release }
      begin
          [P] LC := max(LC,ts); LC := LC+1; [V]
          Rq[j] := ∞;
      end

begin                                             { initialization }
      LC := 0;
      for j := 1 to N do
          begin
              Ts[j] := 0; Rq[j] := ∞
          end
end
```

Ricart-Agarwala



Ricart-Agarwala



MyRq

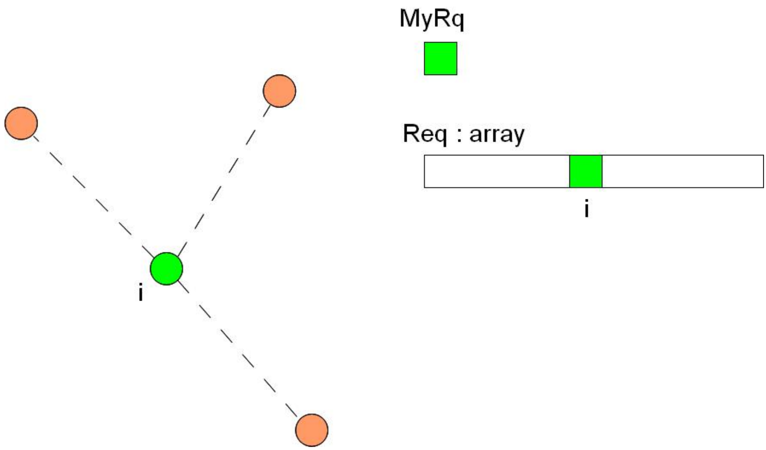
Req : array

i

Rp

i

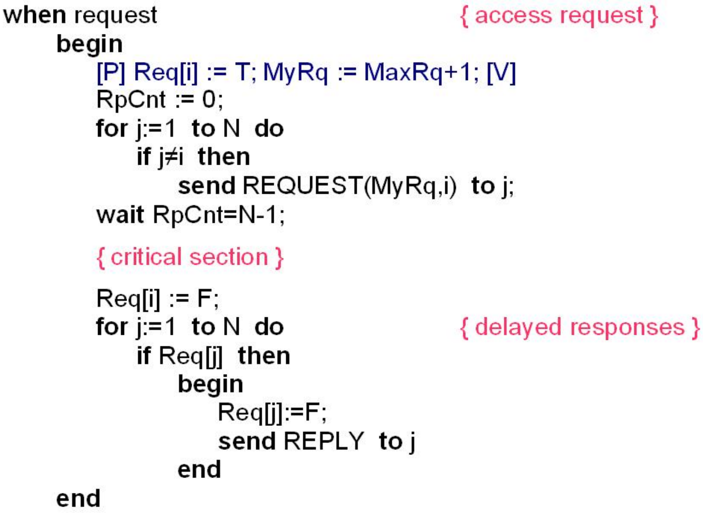
Ricart-Agarwala



MyRq

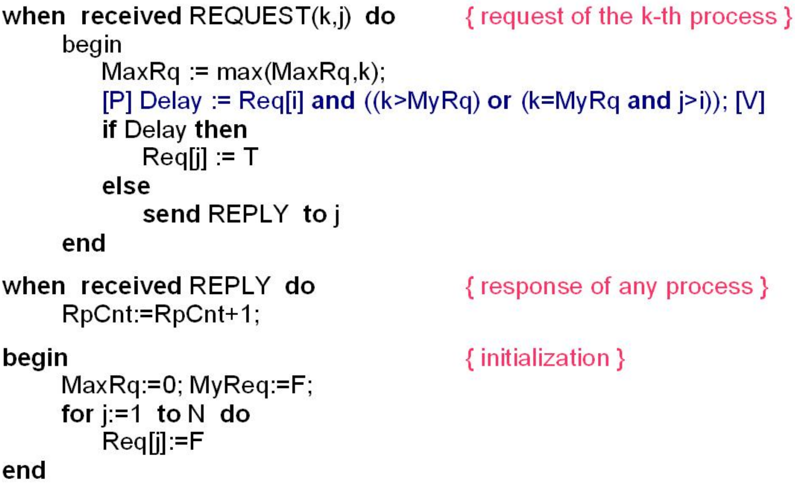
Req : array

Ricart-Agarwala



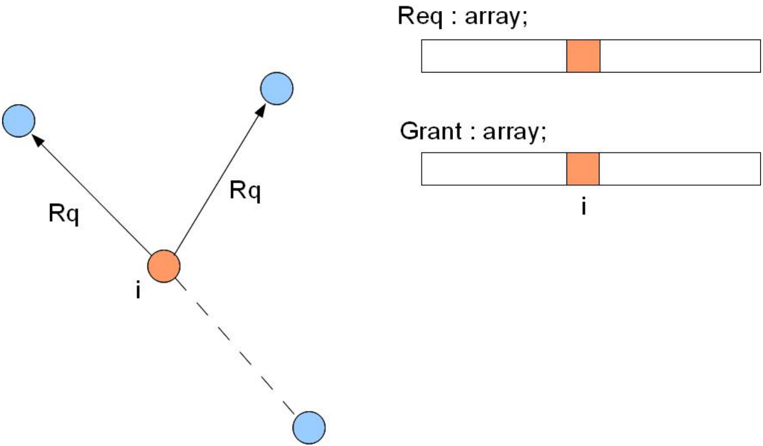
```
when request                                    { access request }
    begin
        [P] Req[i] := T; MyRq := MaxRq+1; [V]
        RpCnt := 0;
        for j:=1  to N  do
            if j≠i  then
                send REQUEST(MyRq,i)  to j;
        wait RpCnt=N-1;

        { critical section }

        Req[i] := F;
        for j:=1  to N  do                      { delayed responses }
            if Req[j]  then
                begin
                    Req[j]:=F;
                    send REPLY  to j
                end
    end
```

Ricart-Agarwala

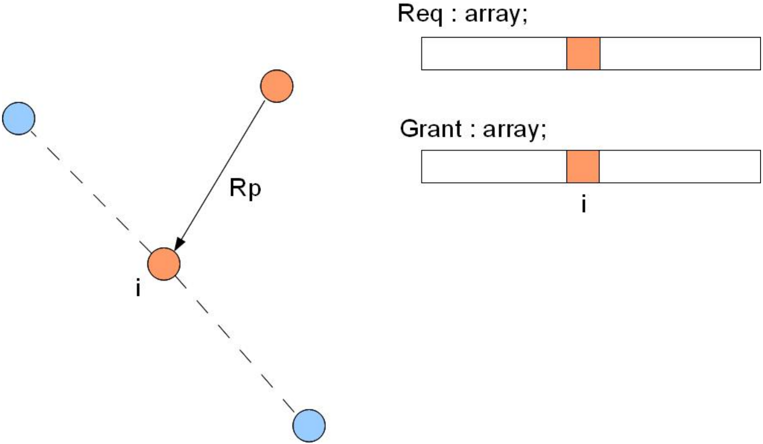


```
when  received REQUEST(k,j)  do          { request of the k-th process }
    begin
        MaxRq := max(MaxRq,k);
        [P] Delay := Req[i] and ((k>MyRq) or (k=MyRq and j>i)); [V]
        if Delay then
            Req[j] := T
        else
            send REPLY  to j
    end

when  received REPLY  do                 { response of any process }
    RpCnt:=RpCnt+1;

begin                                    { initialization }
    MaxRq:=0; MyReq:=F;
    for j:=1  to N  do
        Req[j]:=F
end
```

Carvalho-Roucairol



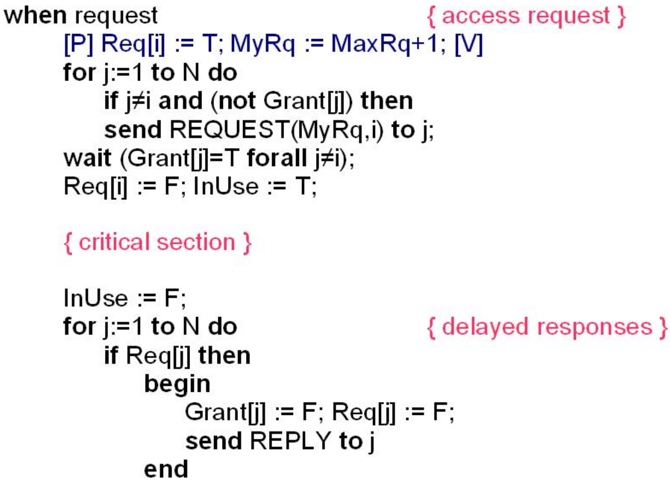
Carvalho-Roucairol



Req : array;

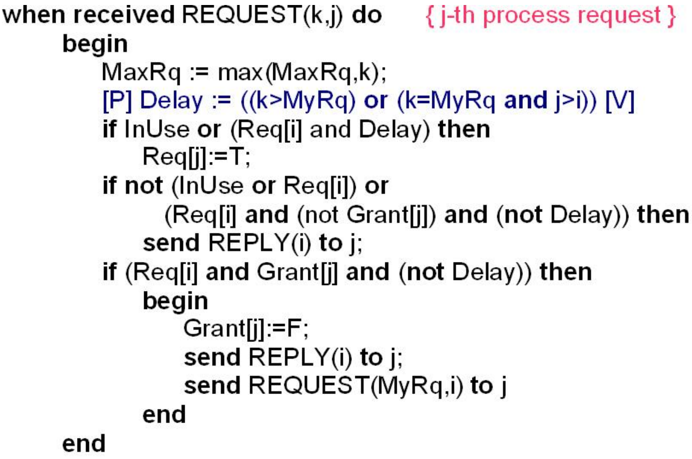
Grant : array;

Carvalho-Roucairol



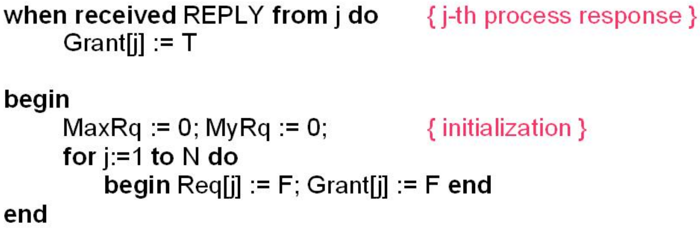
```
when request                                  { access request }
    [P] Req[i] := T; MyRq := MaxRq+1; [V]
    for j:=1 to N do
        if j≠i and (not Grant[j]) then
        send REQUEST(MyRq,i) to j;
    wait (Grant[j]=T forall j≠i);
    Req[i] := F; InUse := T;

    { critical section }

    InUse := F;
    for j:=1 to N do                          { delayed responses }
        if Req[j] then
            begin
                Grant[j] := F; Req[j] := F;
                send REPLY to j
            end
```

Carvalho-Roucairol



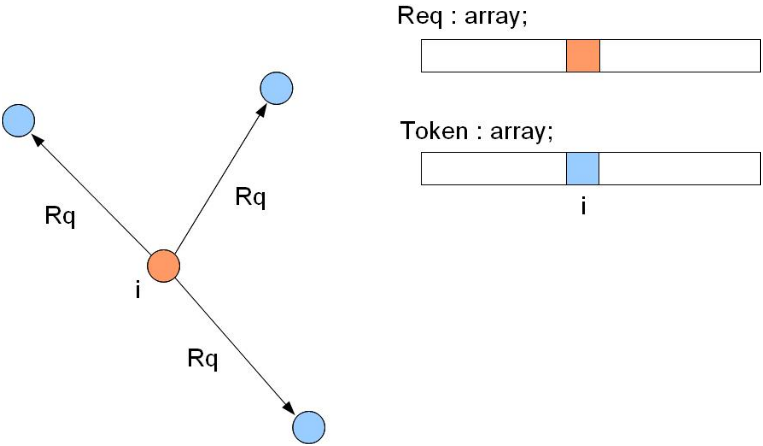
```
when received REQUEST(k,j) do      { j-th process request }
    begin
        MaxRq := max(MaxRq,k);
        [P] Delay := ((k>MyRq) or (k=MyRq and j>i)) [V]
        if InUse or (Req[i] and Delay) then
            Req[j]:=T;
        if not (InUse or Req[i]) or
              (Req[i] and (not Grant[j]) and (not Delay)) then
            send REPLY(i) to j;
        if (Req[i] and Grant[j] and (not Delay)) then
            begin
                Grant[j]:=F;
                send REPLY(i) to j;
                send REQUEST(MyRq,i) to j
            end
    end
```

Carvalho-Roucairol



```
when received REPLY from j do          { j-th process response }
      Grant[j] := T

begin
      MaxRq := 0; MyRq := 0;           { initialization }
      for j:=1 to N do
          begin Req[j] := F; Grant[j] := F end
end
```

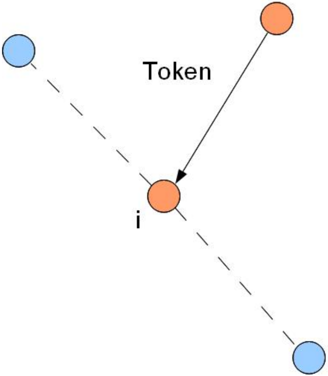
Ricart-Agarwala (token passing)

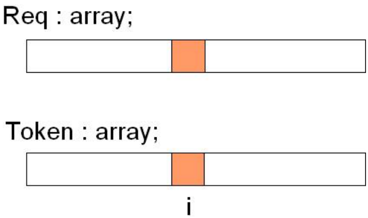


Req : array;

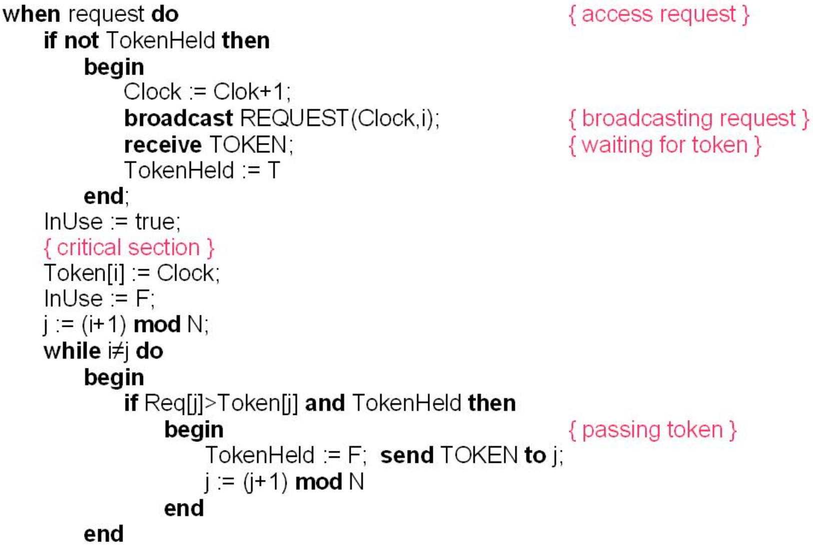
Token : array;

Ricart-Agarwala (token passing)



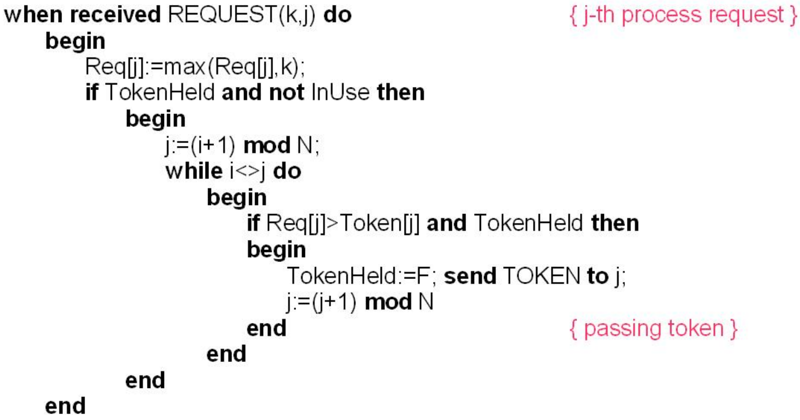


Ricart-Agarwala (token passing)



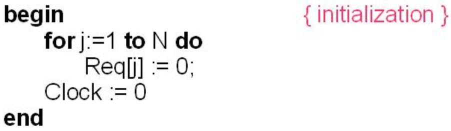
```
when request do                                          { access request }
    if not TokenHeld then
        begin
            Clock := Clok+1;
            broadcast REQUEST(Clock,i);                  { broadcasting request }
            receive TOKEN;                               { waiting for token }
            TokenHeld := T
        end;
    InUse := true;
    { critical section }
    Token[i] := Clock;
    InUse := F;
    j := (i+1) mod N;
    while i≠j do
        begin
            if Req[j]>Token[j] and TokenHeld then
                begin                                    { passing token }
                    TokenHeld := F;  send TOKEN to j;
                    j := (j+1) mod N
                end
        end
```

Ricart-Agarwala (token passing)



```
when received REQUEST(k,j) do                              { j-th process request }
    begin
        Req[j]:=max(Req[j],k);
        if TokenHeld and not InUse then
            begin
                j:=(i+1) mod N;
                while i<>j do
                    begin
                        if Req[j]>Token[j] and TokenHeld then
                        begin
                            TokenHeld:=F; send TOKEN to j;
                            j:=(j+1) mod N
                        end                                { passing token }
                    end
            end
    end
```

Ricart-Agarwala (token passing)



```
begin                          { initialization }
    for j:=1 to N do
        Req[j] := 0;
    Clock := 0
end
```

Cyclical assignment passing

simple algorithm

- group of sequentially identified tasks
- generally used in OS cores

assures security

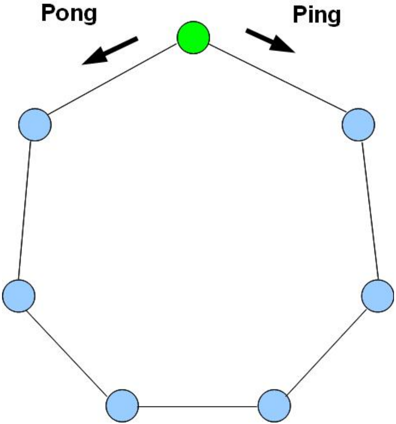
- protection to survive failures

regeneration of assignment

- Misra: Ping-Pong algorithm

Regeneration of assignment

Misra: Ping-Pong algorithm



Misra: Ping-Pong algorithm

```
eceived PING(NPing) do
NPing                                          { regeneration of the lost F

in
Ping:=NPing+1;
Pong:=-NPing


NPing

eceived PONG(NPong) do
NPong                                              { regeneration of lost I

in
Pong:=NPong+1;
Ping:=-NPong


NPong

neeting (PING,PONG) do                            { meeting of Ping and F
```