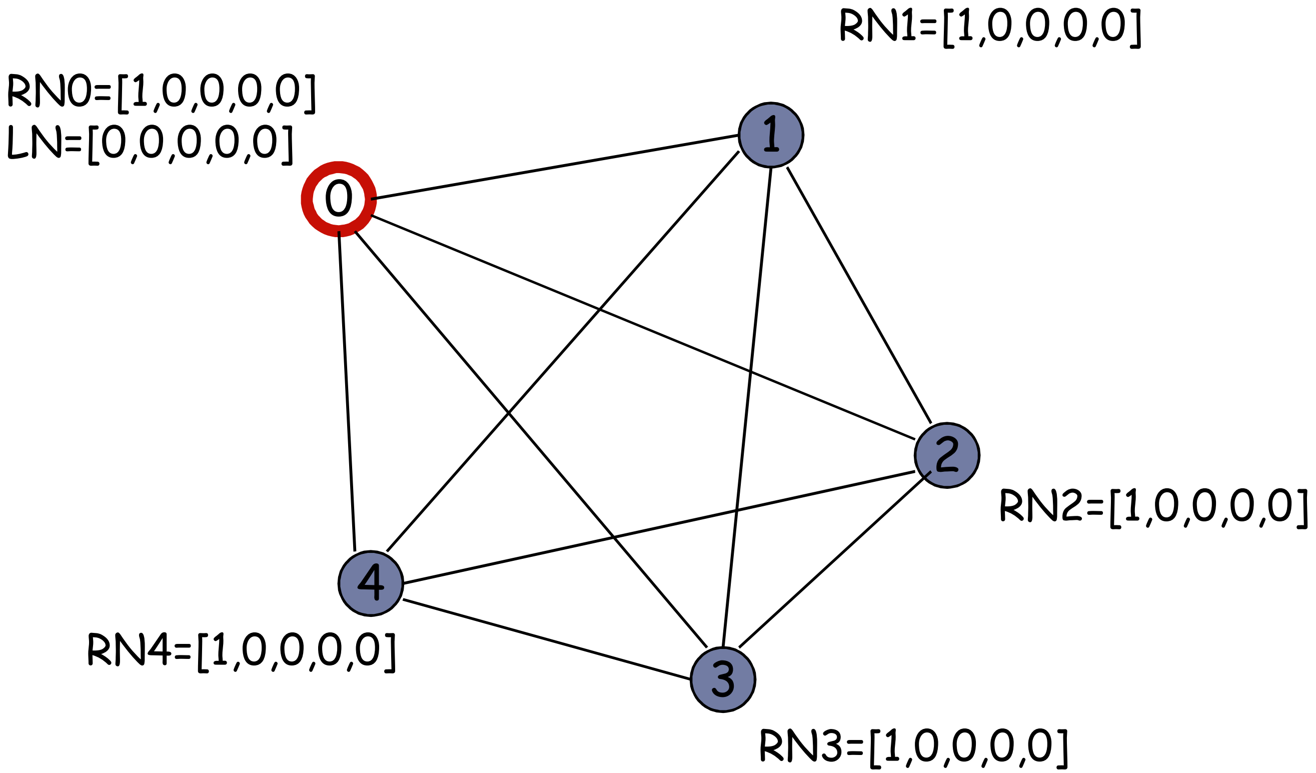
Suzuki Kasami's Distributed Mutual Exclusion Algorithm

Reference : Mukesh Singhal & N.G. Shivaratri, Advanced Concepts in Operating Systems, 5th Edition

Suzuki Kasami's Distributed Mutual Exclusion Algorithm



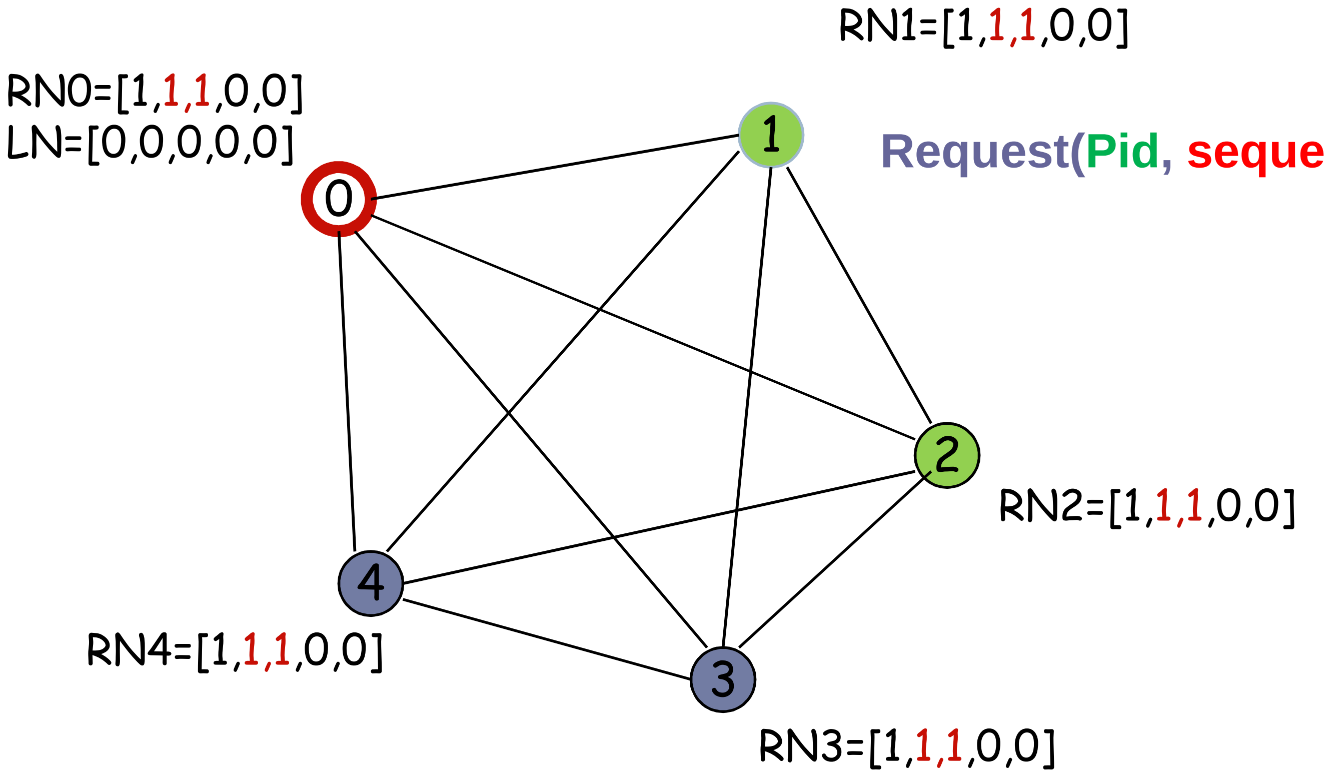
Requesting CS
RNi [i] = RNi [i] + 1
Send **REQUEST(i, sn)**
RNj [i] = max(RNj [i], sn).

Executing in CS
When process gets token,
it uses shared variable.

Exiting from CS
Remove process from
Token queue
Set **LN[i] = RNi [i].**
Give token to process
which satisfies
RNi [j]=LN[j]+1.

initial state: process 0 has sent a request to all, and grabbed the token

Suzuki Kasami's Distributed Mutual Exclusion Algorithm



Request(Pid, sequence no.)

Requesting CS

RNi [i] = RNi [i] + 1
Send **REQUEST(i, sn)**
RNj [i] = max(RNj [i], sn).

Executing in CS

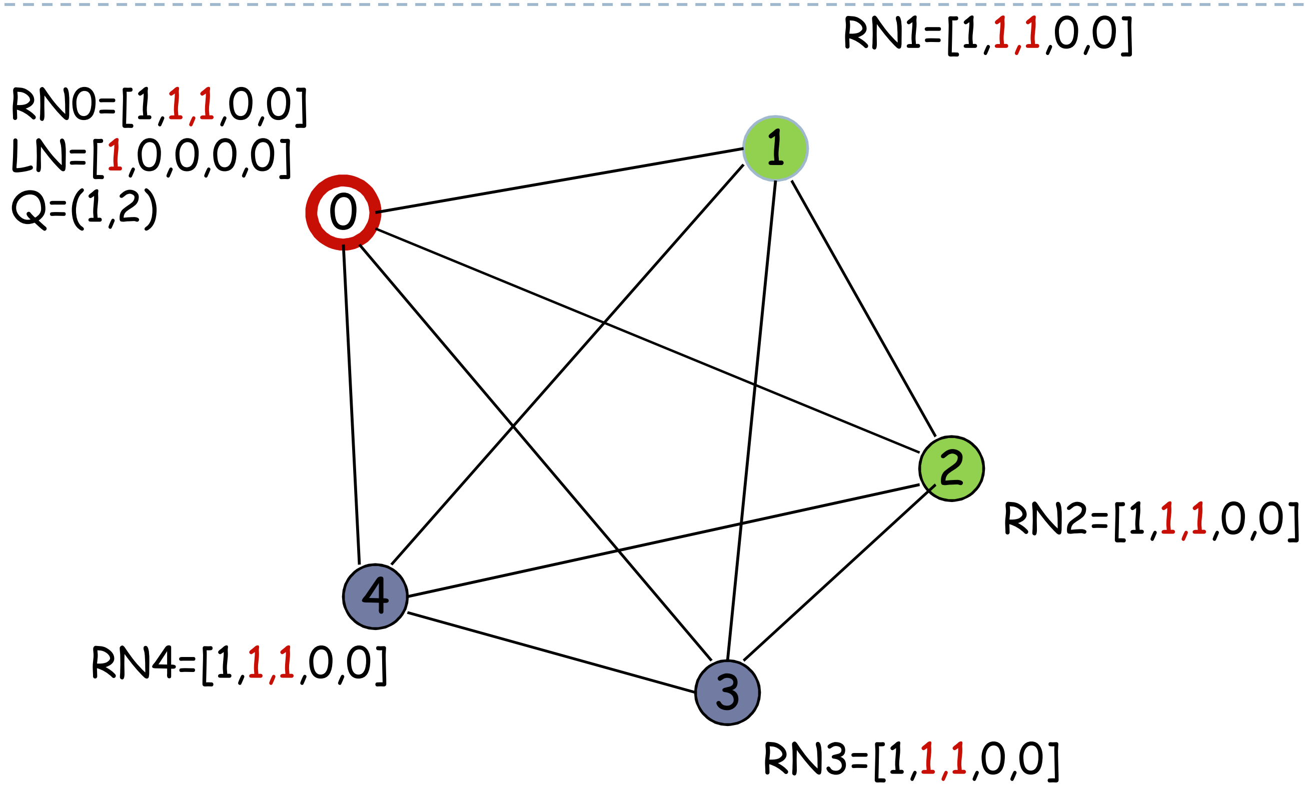
When process gets token, it uses shared variable.

Exiting from CS

Remove process from Token queue
Set **LN[i] = RNi [i].**
Give token to process which satisfies **RNi [j]=LN[j]+1.**

1 & 2 send requests to enter CS

Suzuki Kasami's Distributed Mutual Exclusion Algorithm



Requesting CS

RNi [i] = RNi [i] + 1

Send **REQUEST(i, sn)**

RNj [i] = max(RNj [i], sn).

Executing in CS

When process gets token, it uses shared variable.

Exiting from CS

Remove process from Token queue

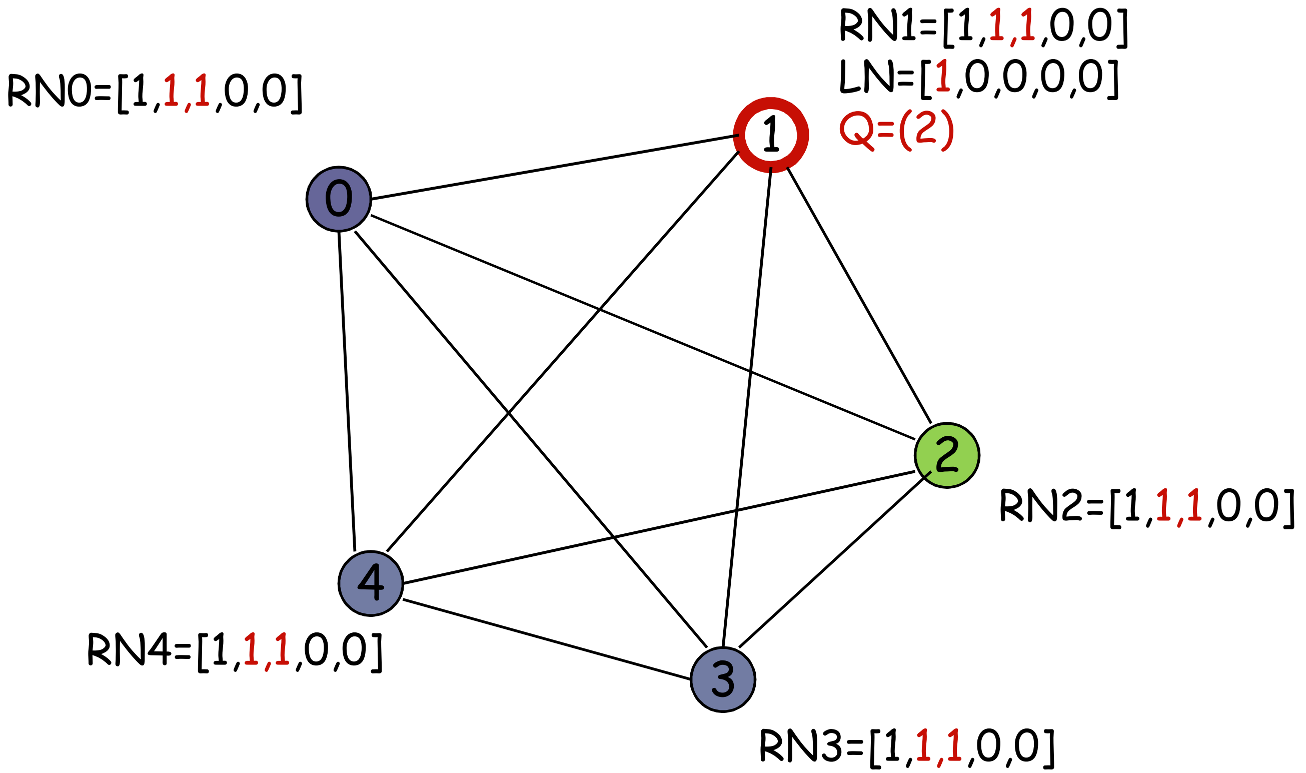
Set **LN[i] = RNi [i].**

Give token to process which satisfies

RNi [j]=LN[j]+1.

0 prepares to exit CS

Suzuki Kasami's Distributed Mutual Exclusion Algorithm



Requesting CS

RNi [i] = RNi [i] + 1

Send **REQUEST(i, sn)**

RNj [i] = max(RNj [i], sn).

Executing in CS

When process gets token, it uses shared variable.

Exiting from CS

Remove process from Token queue

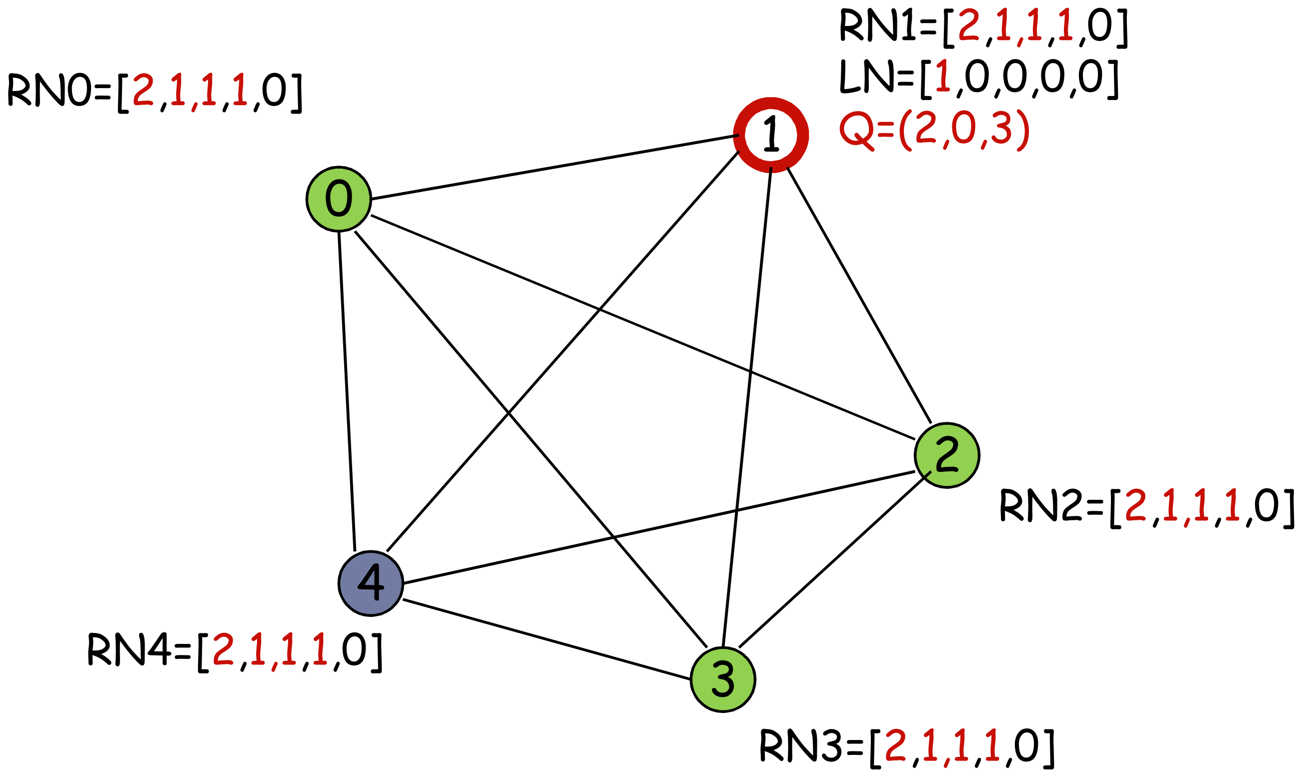
Set **LN[i] = RNi [i].**

Give token to process which satisfies

RNi [j]=LN[j]+1.

0 passes token (Q and last) to 1

Suzuki Kasami's Distributed Mutual Exclusion Algorithm



Requesting CS

RNi [i] = RNi [i] + 1

Send **REQUEST(i, sn)**

RNj [i] = max(RNj [i], sn).

Executing in CS

When process gets token, it uses shared variable.

Exiting from CS

Remove process from Token queue

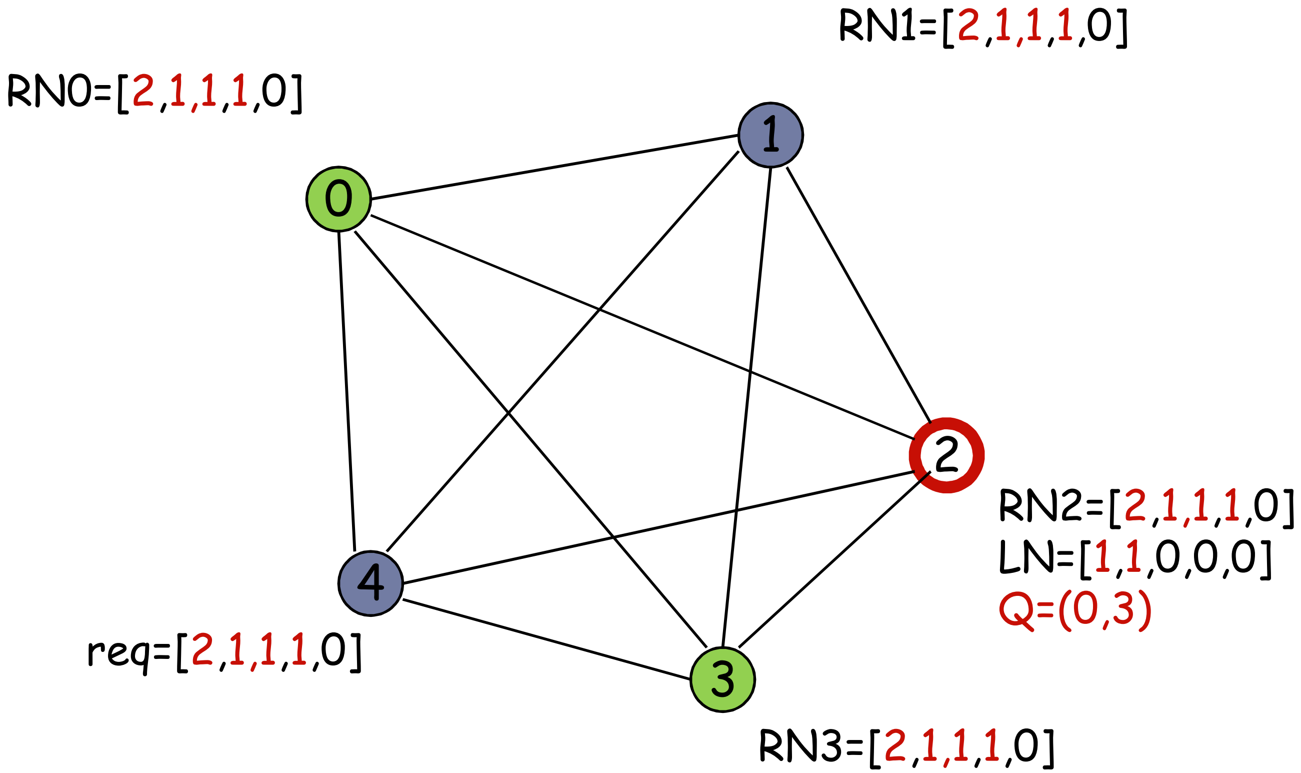
Set **LN[i] = RNi [i].**

Give token to process which satisfies

RNi [j]=LN[j]+1.

0 and 3 send requests

Suzuki Kasami's Distributed Mutual Exclusion Algorithm



Requesting CS

RNi [i] = RNi [i] + 1

Send **REQUEST(i, sn)**

RNj [i] = max(RNj [i], sn).

Executing in CS

When process gets token, it uses shared variable.

Exiting from CS

Remove process from Token queue

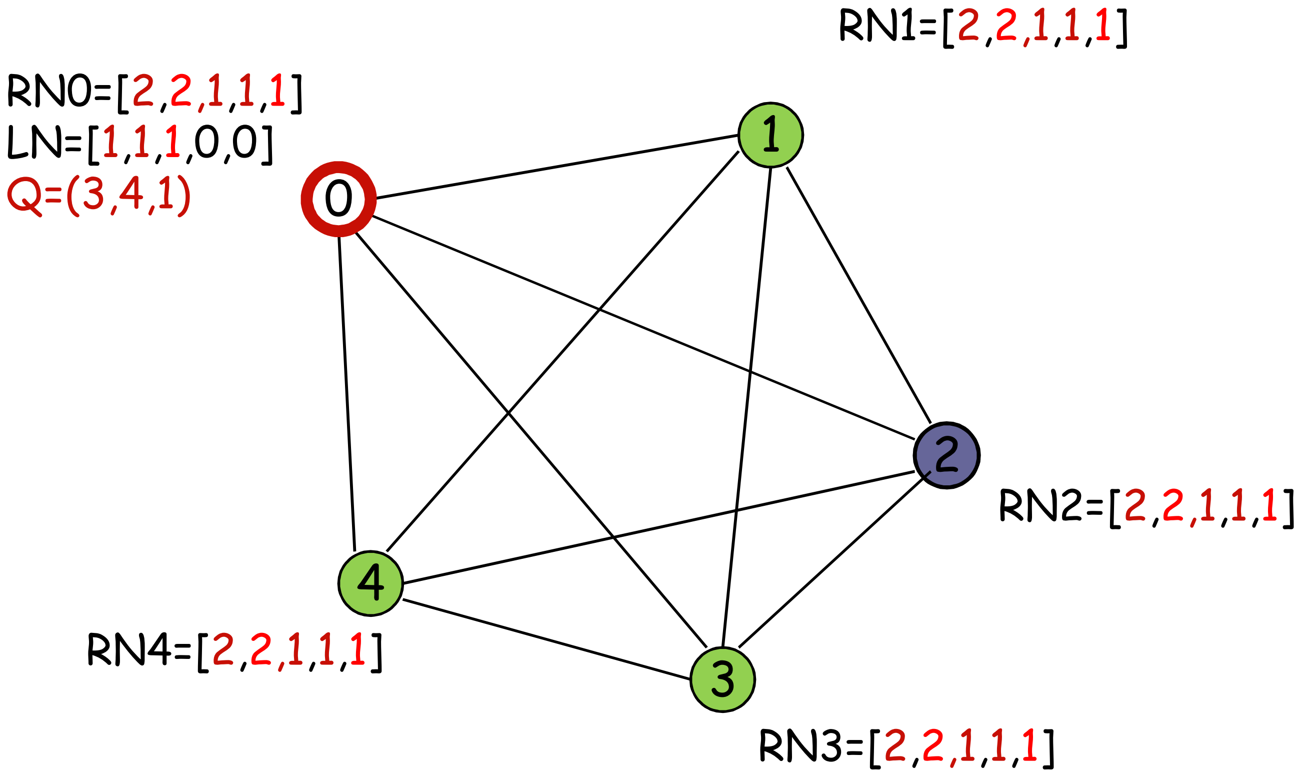
Set **LN[i] = RNi [i].**

Give token to process which satisfies

RNi [j]=LN[j]+1.

1 sends token to 2

Suzuki Kasami's Distributed Mutual Exclusion Algorithm



Requesting CS

RNi [i] = RNi [i] + 1

Send **REQUEST(i, sn)**

RNj [i] = max(RNj [i], sn).

Executing in CS

When process gets token, it uses shared variable.

Exiting from CS

Remove process from Token queue

Set **LN[i] = RNi [i].**

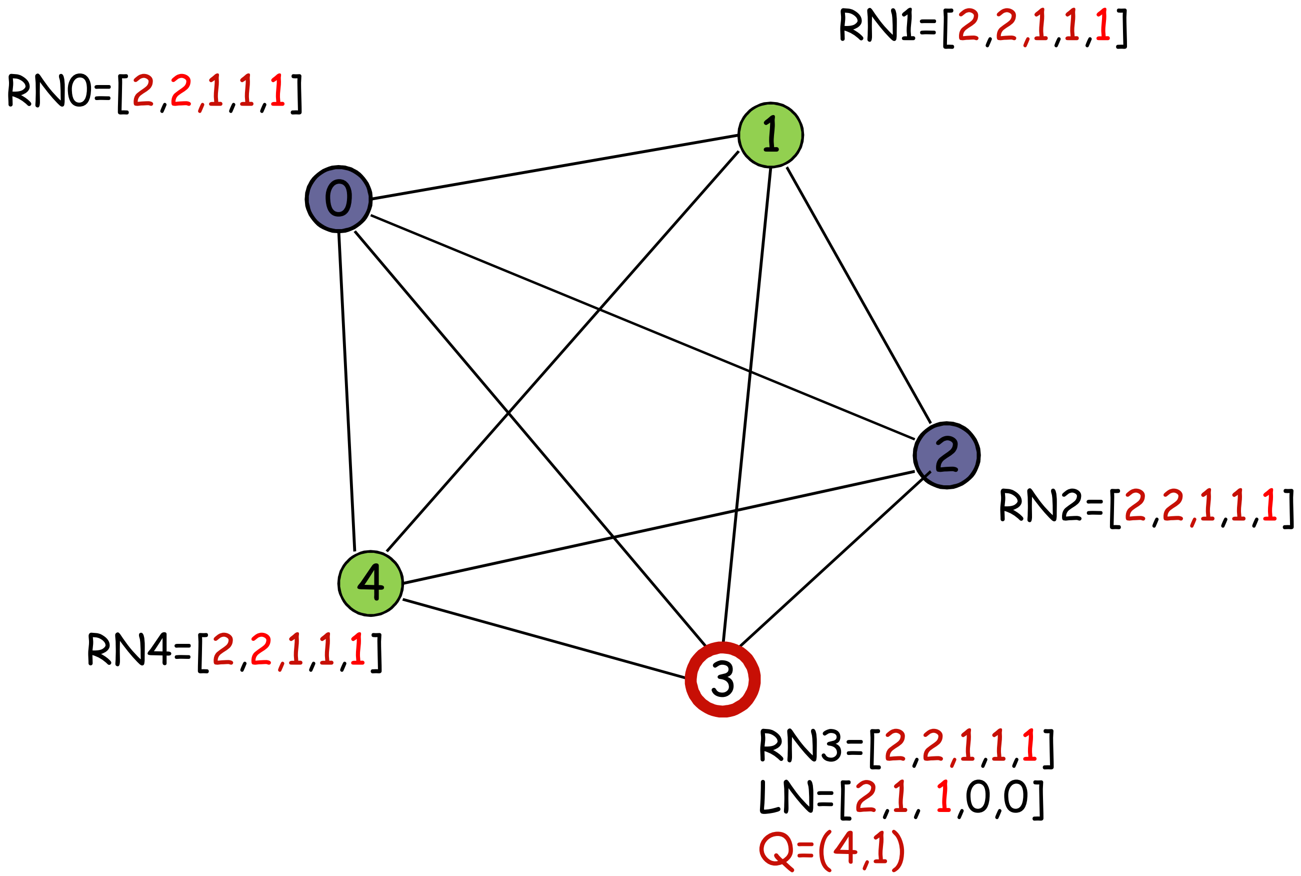
Give token to process which satisfies

RNi [j]=LN[j]+1.

2 prepares to exit & sends token to 0

1 & 4 sends request.

Suzuki Kasami's Distributed Mutual Exclusion Algorithm



RN1=[2,2,1,1,1]

RN0=[2,2,1,1,1]

RN2=[2,2,1,1,1]

RN4=[2,2,1,1,1]

RN3=[2,2,1,1,1]
LN=[2,1, 1,0,0]
Q=(4,1)

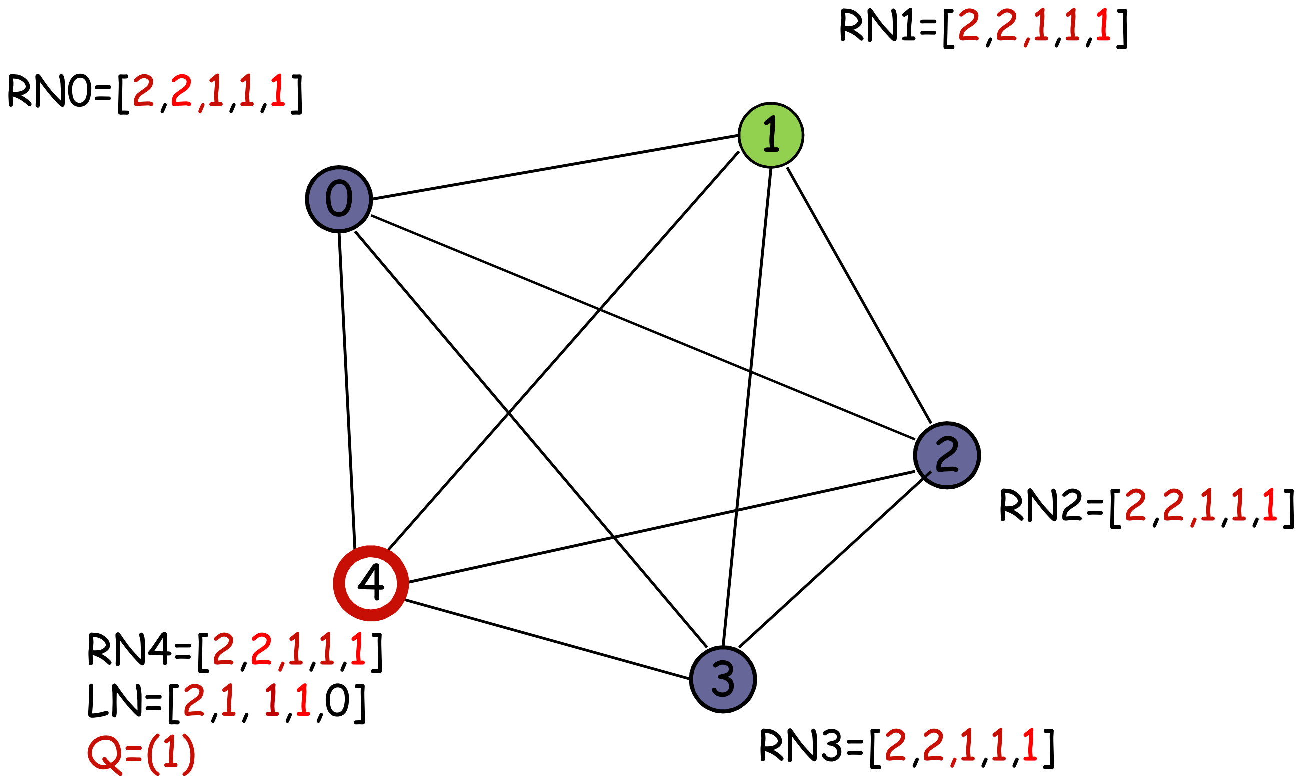
0 exits & sends token to 3

Requesting CS
RNi [i] = RNi [i] + 1
Send **REQUEST(i, sn)**
RNj [i] = max(RNj [i], sn).

Executing in CS
When process gets token,
it uses shared variable.

Exiting from CS
Remove process from
Token queue
Set **LN[i] = RNi [i].**
Give token to process
which satisfies
RNi [j]=LN[j]+1.

Suzuki Kasami's Distributed Mutual Exclusion Algorithm



Requesting CS

RNi [i] = RNi [i] + 1

Send **REQUEST(i, sn)**

RNj [i] = max(RNj [i], sn).

Executing in CS

When process gets token, it uses shared variable.

Exiting from CS

Remove process from Token queue

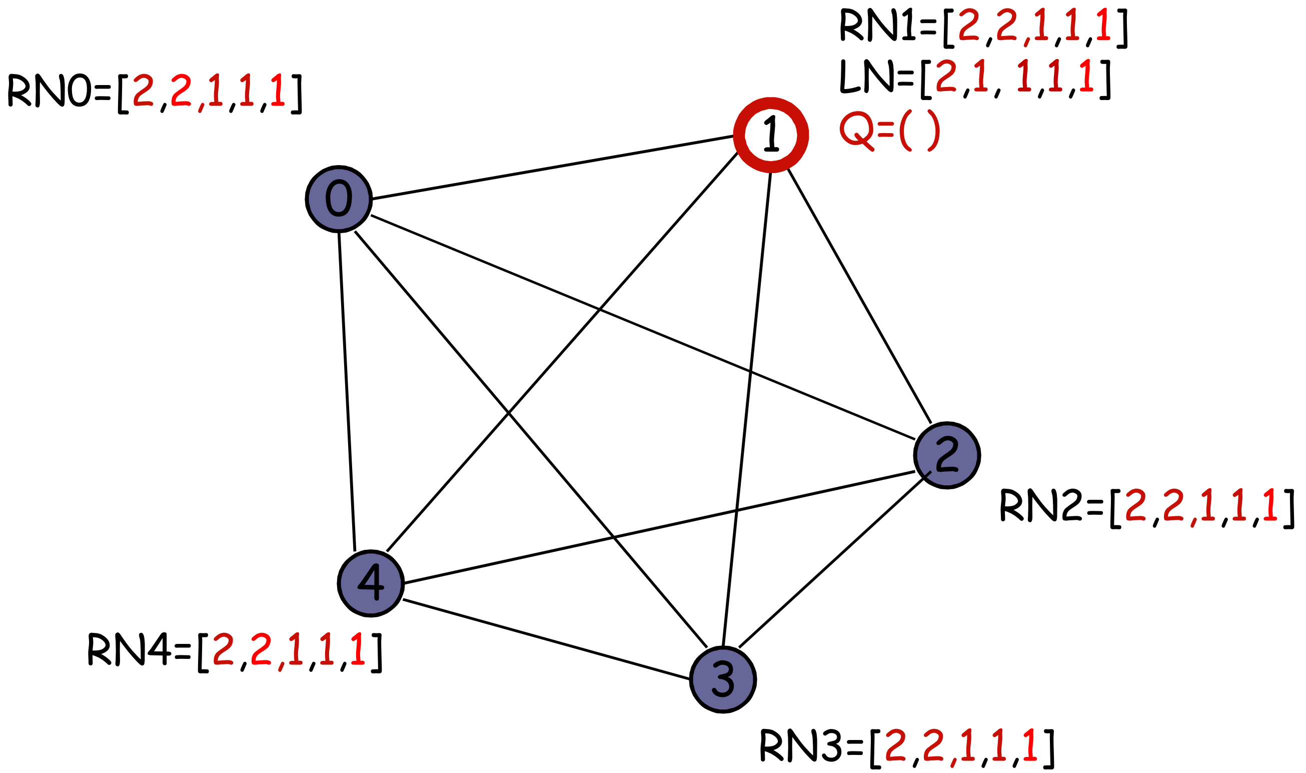
Set **LN[i] = RNi [i].**

Give token to process which satisfies

RNi [j]=LN[j]+1.

3 exits & sends token to 4

Suzuki Kasami’s Distributed Mutual Exclusion Algorithm



Requesting CS

RNi [i] = RNi [i] + 1

Send **REQUEST(i, sn)**

RNj [i] = max(RNj [i], sn).

Executing in CS

When process gets token, it uses shared variable.

Exiting from CS

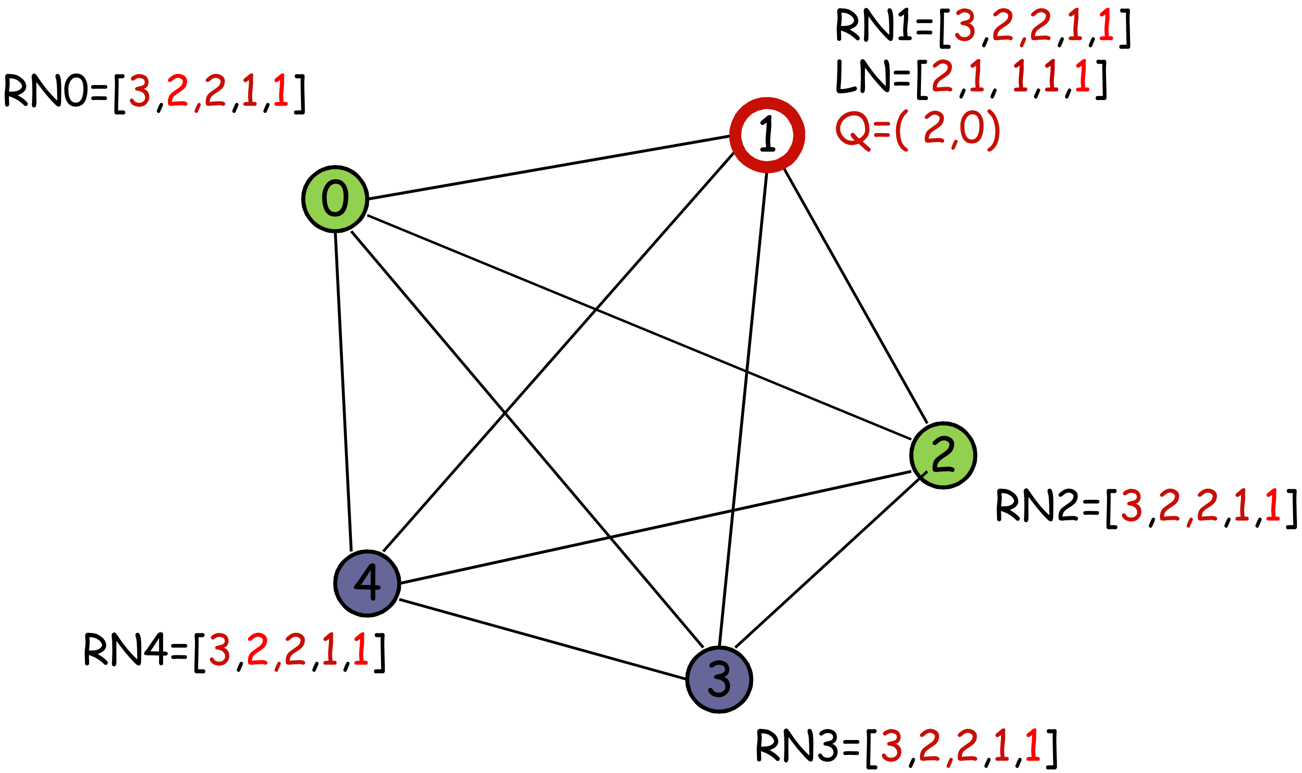
Remove process from Token queue

Set **LN[i] = RNi [i].**

Give token to process which satisfies **RNi [j]=LN[j]+1.**

4 exits & sends token to 1

Suzuki Kasami's Distributed Mutual Exclusion Algorithm



Requesting CS

RNi [i] = RNi [i] + 1

Send **REQUEST(i, sn)**

RNj [i] = max(RNj [i], sn).

Executing in CS

When process gets token, it uses shared variable.

Exiting from CS

Remove process from Token queue

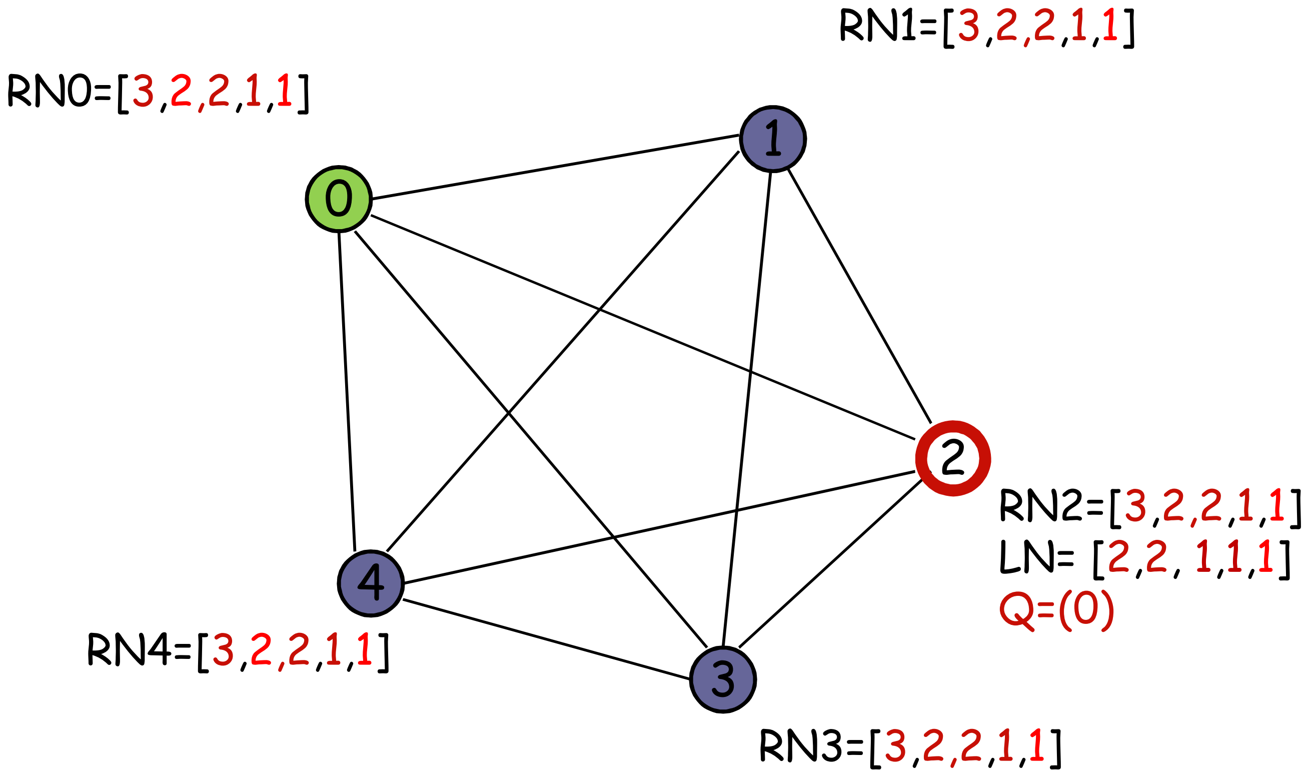
Set **LN[i] = RNi [i].**

Give token to process which satisfies

RNi [j]=LN[j]+1.

2 & 0 request for token

Suzuki Kasami's Distributed Mutual Exclusion Algorithm



RN1=[3,2,2,1,1]

RN0=[3,2,2,1,1]

RN2=[3,2,2,1,1]
LN= [2,2, 1,1,1]
Q=(0)

RN4=[3,2,2,1,1]

RN3=[3,2,2,1,1]

Requesting CS

RNi [i] = RNi [i] + 1
Send **REQUEST(i, sn)**
RNj [i] = max(RNj [i], sn).

Executing in CS

When process gets token, it uses shared variable.

Exiting from CS

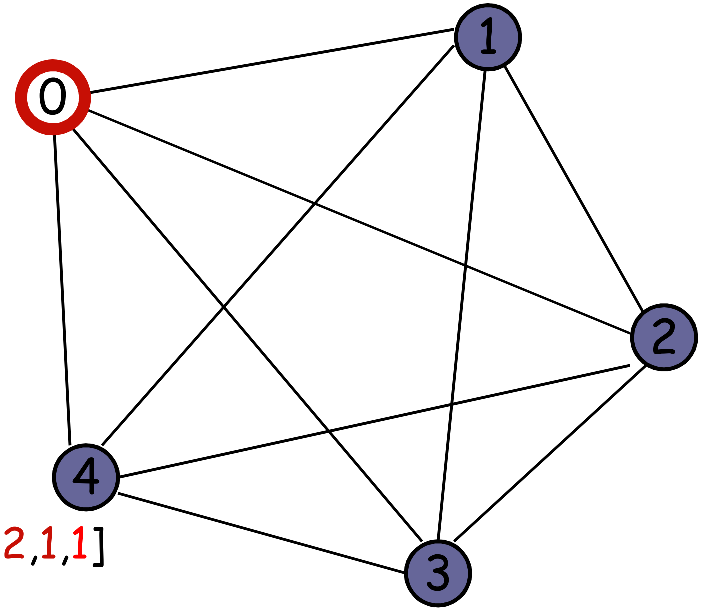
Remove process from Token queue
Set **LN[i] = RNi [i].**
Give token to process which satisfies
RNi [j]=LN[j]+1.

Token is given to 2
2 exits from CS and gives token to 0

Suzuki Kasami’s Distributed Mutual Exclusion Algorithm

RN1=[3,2,2,1,1]

RN0=[3,2,2,1,1]
LN= [2,2, 2,1,1]
Q=( )



RN2=[3,2,2,1,1]

RN4=[3,2,2,1,1]

RN3=[3,2,2,1,1]

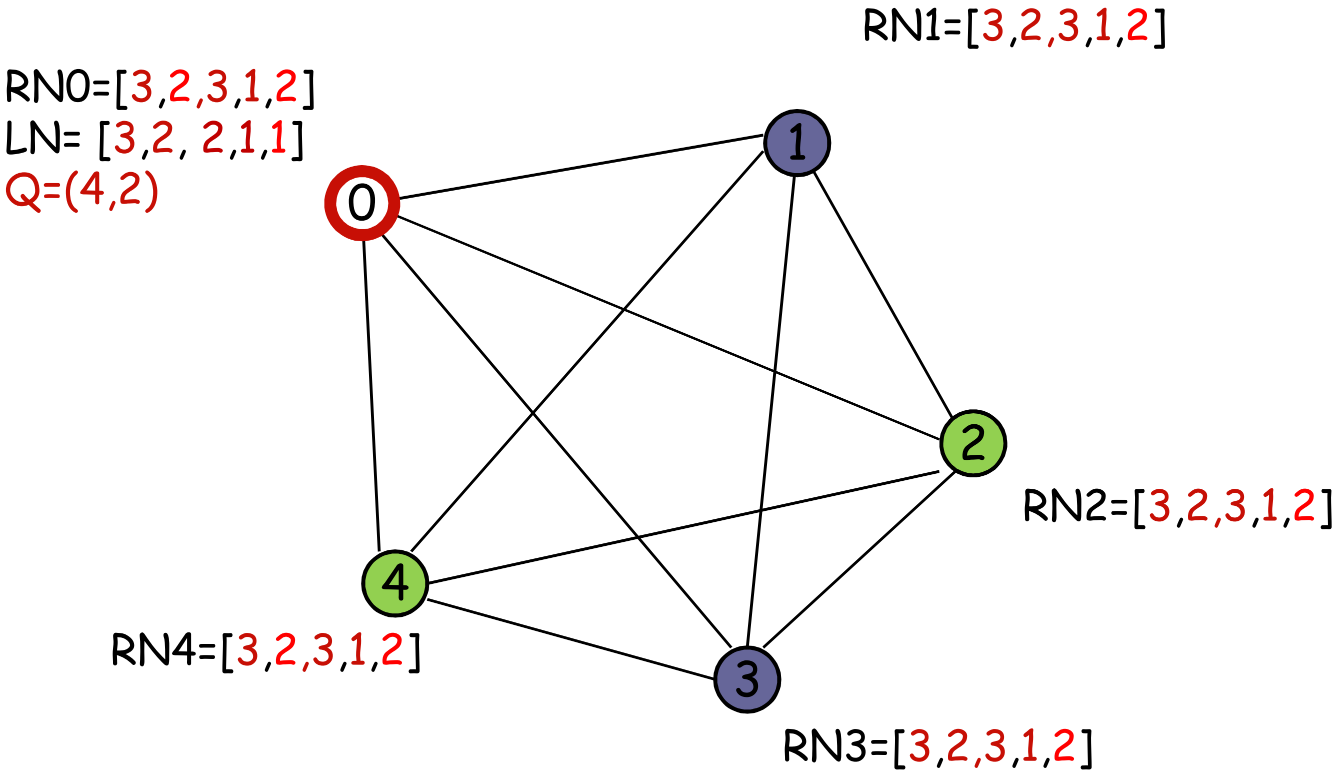
Requesting CS
RNi [i] = RNi [i] + 1
Send **REQUEST(i, sn)**
RNj [i] = max(RNj [i], sn).

Executing in CS
When process gets token,
it uses shared variable.

Exiting from CS
Remove process from
Token queue
Set **LN[i] = RNi [i].**
Give token to process
which satisfies
RNi [j]=LN[j]+1.

0 gets the token, 0 executes in CS
While process 0 exiting from CS, it sets
LN[0] = RN0[0] i.e.LN= [3,2, 2,1,1]

Suzuki Kasami's Distributed Mutual Exclusion Algorithm



RN1=[3,2,3,1,2]

RN0=[3,2,3,1,2]
LN= [3,2, 2,1,1]
Q=(4,2)

RN2=[3,2,3,1,2]

RN4=[3,2,3,1,2]

RN3=[3,2,3,1,2]

Requesting CS
RNi [i] = RNi [i] + 1
Send **REQUEST(i, sn)**
RNj [i] = max(RNj [i], sn).

Executing in CS
When process gets token,
it uses shared variable.

Exiting from CS
Remove process from
Token queue
Set **LN[i] = RNi [i].**
Give token to process
which satisfies
RNi [j]=LN[j]+1.

4 & 2 are making request for CS

Thank You