Algorithme de Maekawa (quorum)

- **Chaque site ne peut donner sa permission qu'à un seul à la fois**
  - Arbitrer un certain nombre de conflits
- **Message REQUEST n'est pas diffusé à tous les sites :**
  - Chaque site S_i appartient à un ensemble (quorum) RS_i (Request Set) dont il doit obtenir l'accord (msg LOCKED) de tous les membres pour pouvoir entrer en SC.
  - Il doit y avoir au moins un site commun entre deux ensembles RS_i et RS_j. Ce site arbitre les conflits.

$$\forall i, j \in \{1.., N\} \text{ tels que } i!=j, RS_i \cap RS_j != \varnothing \qquad (1)$$

Algorithme de Maekawa (quorum)

N= nombre de sites
K_i = nombre de sites dans RS_i
D= nombre d'ensembles auquel chaque site appartient

- **Afin de minimiser le trafic des messages et de demander le même effort à tous les sites:**

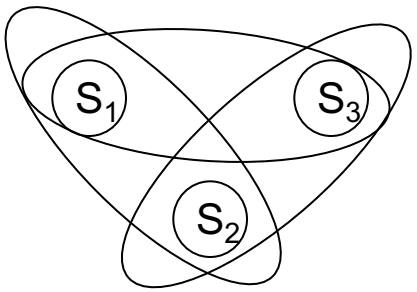
- $|RS_1| = |RS_2| = |RS_3| \ldots = |RS_N| = K$
- $\forall\ S_i \in \{S_1.., S_N\}, S_i \in RS_i$
- $\forall\ i, j \in \{1.., N\}$ tels que i != j,
  S_i et S_j appartiennent à *D RS*
  /* même nombre d'ensembles */
- D = K est une possibilité

Algorithme de Maekawa (quorum)

- **Exemples de quorum**

$RS_1 = \{S_1, S_2\}$
$RS_2 = \{S_2, S_3\}$
$RS_3 = \{S_3, S_1\}$

N=3, K=2



$RS_1 = \{S_1, S_2, S_3\}$
$RS_2 = \{S_2, S_4, S_6\}$
$RS_3 = \{S_3, S_5, S_6\}$
$RS_4 = \{S_4, S_1, S_5\}$
$RS_5 = \{S_5, S_2, S_7\}$
$RS_6 = \{S_6, S_1, S_7\}$
$RS_7 = \{S_7, S_3, S_4\}$

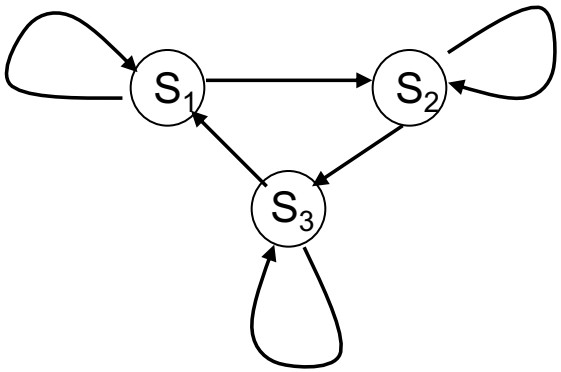
N=7, K=3

Algorithme de Maekawa

- **Pour entrer en SC, le site S_i doit verrouiller tous les membres de son ensemble RS_i en leur envoyant un message du type REQUEST.**
  - En recevant un msg REQUEST de S_j, si S_i ne se trouve pas déjà verrouillé, S_i envoie son accord (msg *LOCKED*) à S_j et se verrouille au profit de S_j.
    - S_i ne peut se verrouiller qu'au profit d'un seul site.
    - Si S_i arrive à verrouiller tous les membres de RS_i, aucun autre site ne pourra faire la même chose à cause de la propriété (1) – *intersection des ensembles*.
      - S_i rentre alors en SC. En sortant, S_i envoie un msg *RELEASE* à tous les membres de RS_i.

Algorithme de Maekawa

- **Risque d'interblocage :**
  - Le fait qu'un arbitre ne donne sa permission qu'à un seul demandeur (ne se verrouille qu'au profit d'un seul site) conduit à des situations d'interblocage.



$RS_1 = \{S_1, S_2\}$
$RS_2 = \{S_2, S_3\}$
$RS_3 = \{S_3, S_1\}$

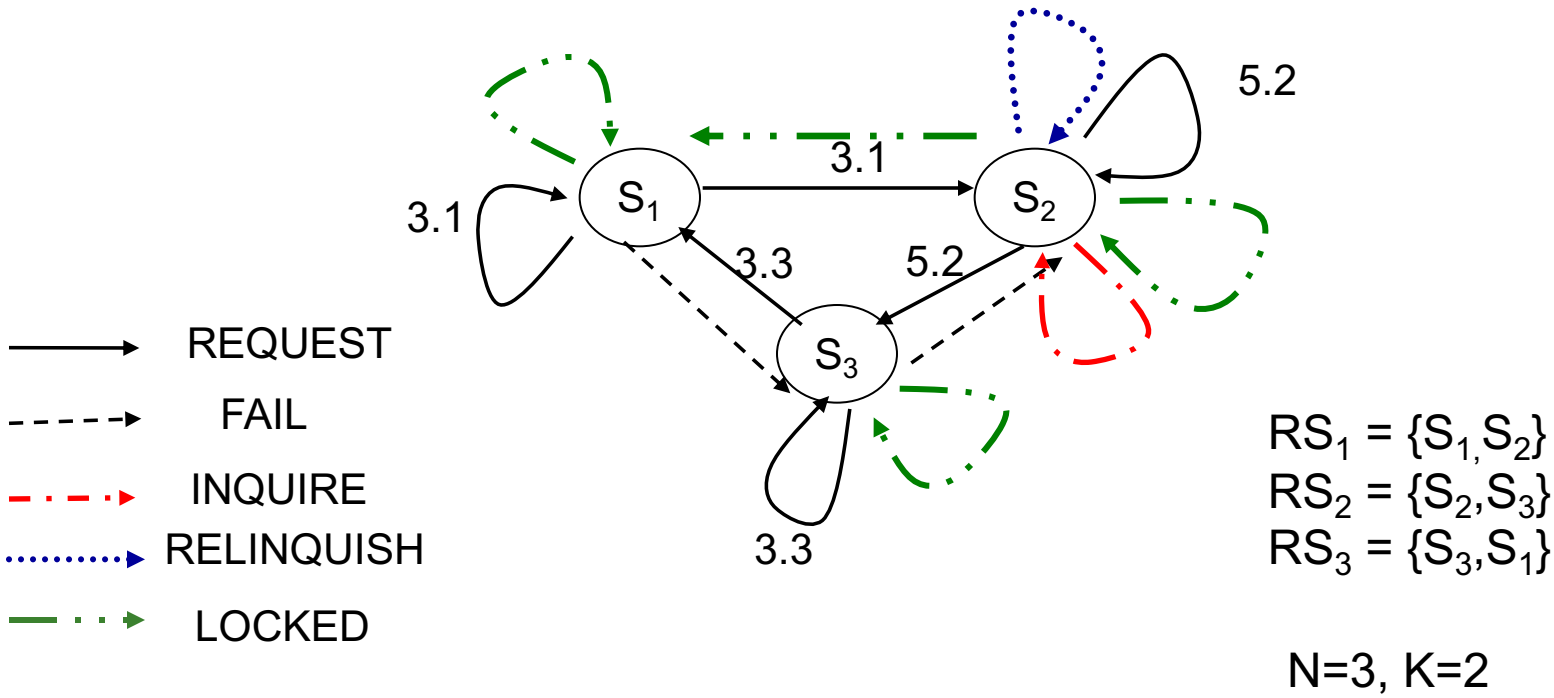
N=3, K=2

Algorithme de Maekawa

- **Solution pour le problème d'interblocage**
  - Dater les messages : ordre total
    - Horloge de Lamport + identifiant
  - Reprendre la permission accordée si la nouvelle demande est antérieure à celle déjà satisfaite
    - Si le site qui possède la permission sait qu'il n'est pas en mesure de recevoir tous les accords de son ensemble, il libère la permission obtenue.
      - Deux nouveaux types de messages :
        - *INQUIRE* : demande de la possibilité de reprendre la permission.
        - *RELINQUISH* : libération de la permission (verrou).

Algorithme de Maekawa

- **Solution pour le problème d'interblocage**



Algorithme de Maekawa

- **Contenu des messages :**
  - (type, (H_i, S_i)) : messages estampillés

- **Types de messages :**
  - ***REQUEST***
    - Demande d'entrée en SC. S_i envoie un tel message à tous les membres de son ensemble RS_i.
  - ***RELEASE***
    - Envoyé par un site S_i à tous les membres de son ensemble RS_i lorsqu'il sort de la SC.
  - ***LOCKED***
    - Envoyé par un site S_i en réponse à un message REQUEST de S_j, s'il ne l'a pas encore envoyé à un autre site. S_i se trouve alors verrouillé au profit de S_j

Algorithme de Maekawa

- **Types de messages (cont) :**
  - ***FAIL***
    - Envoyé par un site S_i en réponse à un message REQUEST de S_j , s'il ne peut pas donner son accord (S_i se trouve déjà verrouillé). Le message de S_j est moins prioritaire et sera mis dans la file d'attente.
  - ***INQUIRE***
    - Envoyé par un site S_i à S_j pour tenter de récupérer la permission accordée à S_j (S_i était verrouillé au profit de S_j).
  - ***RELINQUISH***
    - Réponse à un message du type INQUIRE afin de rendre une permission non utilisable.

Algorithme de Maekawa

Grandes Lignes

Variables Locales:

$FA_i = \emptyset$;

$H_i = 0$;

$At_i = \emptyset$;

Pour tous les processus S_i :

Request_CS() :

- H_i ++;
- $\forall$ j $\in$ RS, envoyer un message REQUEST à j ;
- $At_i = RS_i$;
- $\forall$ j $\in$ RS_i, attendre la réception d'un msg. LOCKED : ($At_i = \emptyset$);

Release_CS() :

- Hi ++;
- $\forall$ j $\in$ RS_i Envoyer un message RELEASE à j ;

Algorithme de Maekawa

Reception (msg de S_j) :

• *REQUEST* :

- Mettre à jour Hi *(H_i = max (H_i,H_j)+1);*
- Si S_i n'est pas verrouillé {
  - Verrouiller S_i au profit de S_j ;
  - Envoyer à S_j un message *LOCKED* ;

}

sinon /* S_i verrouillé au profit de S_k */ {

- (*FAi U {msg S_j}*); /* insérer la demande dans la file d'attente dans l'ordre*/
- Si la demande de S_k ou une autre dans la file FA_i est antérieure à celle de S_j
  - envoyer un message *FAIL* à S_j

sinon

- si un message de *INQUIRE* n'a pas encore été envoyé à S_k
  - envoyer un message *INQUIRE* à S_k.

}

• *LOCKED* :

- (*At_i=At_i -{S_j}); /*** comptabiliser la réception d'une permission en plus */

Algorithme de Maekawa

INQUIRE :

Si un message du type FAIL a été reçu

Envoyer message *RELINQUISH* à S_j; ($At_i = At_i \cup \{S_j\}$);

RELINQUISH :

libérer le verrou ;

$FA_i \cup \{S_j\}$; /* ajouter la requête de S_j dans la file dans l'ordre*/

se verrouiller au profit de S_k, le site qui se trouve en tête de la file;

$FA_i - \{S_k\}$; / *retirer la requête S_k de la file d'attente */

envoyer un message *LOCKED* à S_k;

RELEASE:

libérer le verrou;

se verrouiller au profit de S_k, le site qui se trouve en tête de la file;

$FA_i - \{S_k\}$; / *retirer la requête S_k de la file d'attente */

envoyer un message *LOCKED* à S_k;

FAIL :

enregistrer la réception d'un échec d'accord de la part de S_j;

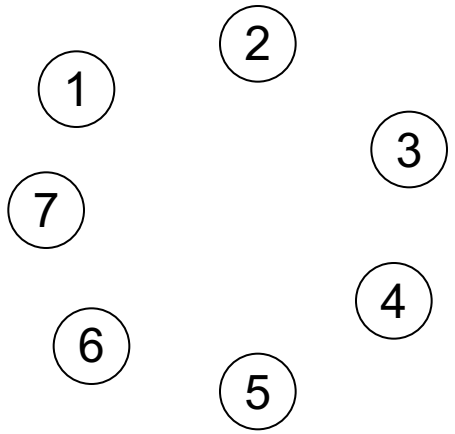
Si *INQUIRE* de S_k pendant

envoyer msg *RELINQUISH* à S_k ($At_i = At_i \cup \{S_k\}$);

}

Algorithme de Maekawa

- **Exemple**



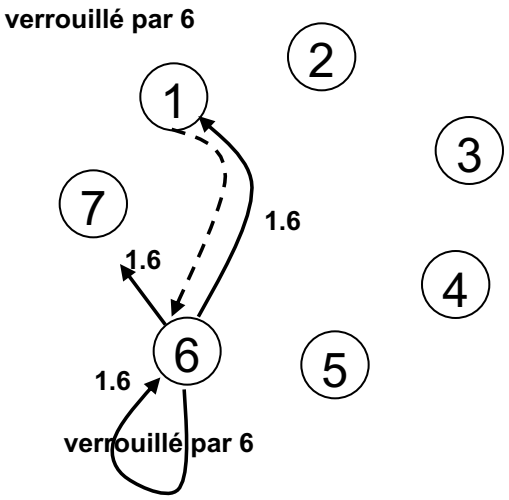
$RS_1 = \{S_1, S_2, S_3\}$
$RS_2 = \{S_2, S_4, S_6\}$
$RS_3 = \{S_3, S_5, S_6\}$
$RS_4 = \{S_4, S_1, S_5\}$
$RS_5 = \{S_5, S_2, S_7\}$
$RS_6 = \{S_6, S_1, S_7\}$
$RS_7 = \{S_7, S_3, S_4\}$

N=7, K=3

Sites 2,5 et 6 exécutent Request_CS
H_2, H_5 et H_6 =1

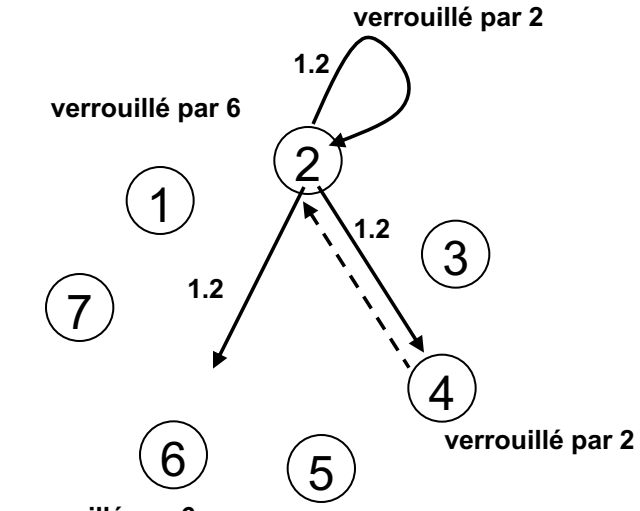
Algorithme de Maekawa

- **Exemple**



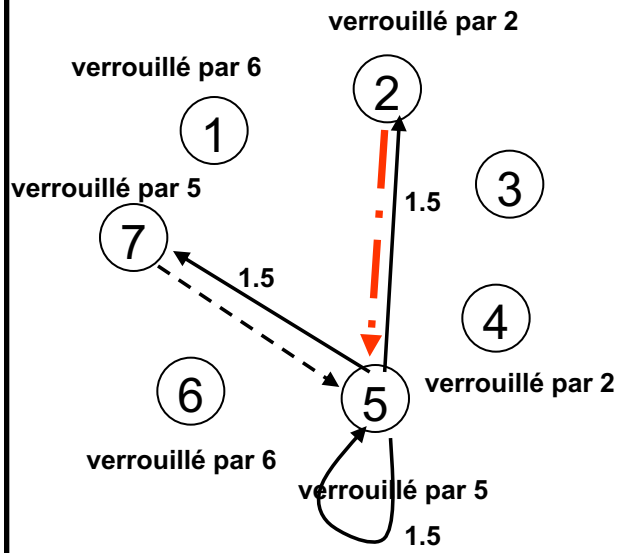
$RS_6 = \{S_6, S_1, S_7\}$

Site 6 verrou site 1 et 6;
Msg REQUEST pour site 7 en route



$RS_2 = \{S_2, S_4, S_6\}$

Site 2 verrou site 2 et 4;
Msg REQUEST pour site 6 en route



$RS_5 = \{S_5, S_2, S_7\}$

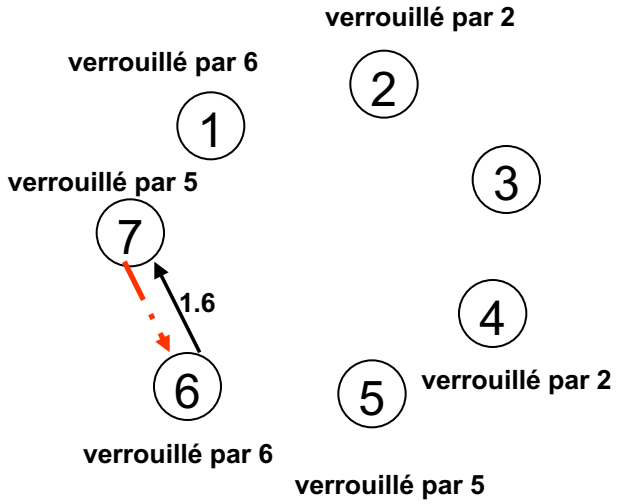
Site 5 verrou site 5 et 7;
Reçoit msg FAIL du site 2

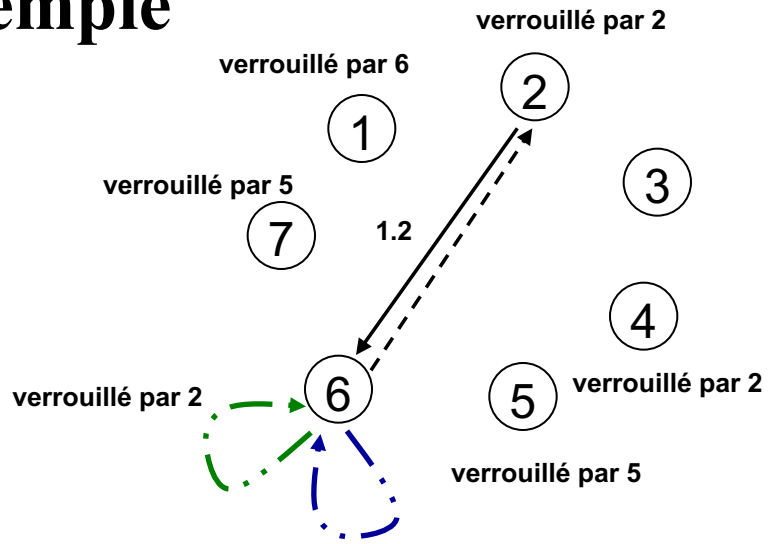
REQUEST
LOCKED
FAIL

$RS_2 = \{S_2, S_4, S_6\}$
$RS_5 = \{S_5, S_2, S_7\}$
$RS_6 = \{S_6, S_1, S_7\}$

Algorithme de Maekawa

- **Exemple**





Site 7 reçoit msg REQUEST du site 6

REQUEST
LOCKED
FAIL
INQUIRE
RELINQUISH

Site 6 reçoit msg REQUEST du site 2
Site 6 envoie Msg INQUIRE au site 6
Site 6 libère le verrou – msg RELINQUISH au site 6
Site 6 envoie msg LOCKED au site 2
Site 2 rentre en SC

Algorithme de Maekawa

- **Nombre de Messages par exécution de SC : O(sqrt(N))**
  - Faible demande : 3*(K-1)
    - (K-1) msg REQUEST + (K-1)msg LOCKED + (K-1)msg RELEASE
  - Forte demande : 5*(K-1)
    - (K-1) msg REQUEST + (K-1)msg LOCKED + (K-1)msg RELEASE + (K-1) *msg INQUIRE + (K-1)*RELINQUISH
  - La valeur de K est approximativement égale à sqrt (N)
    - Nombre de message entre **3*sqrt(N)** et **5*sqrt(N)**
- **Avantages:**
  - Si pas de conflit, moins de messages envoyés par rapport à Lamport et Ricart-Agrawala.
- **Inconvénients**
  - Possibilité d'interblocage
  - Construction des ensembles