Lélia Blin

Allocation de ressources

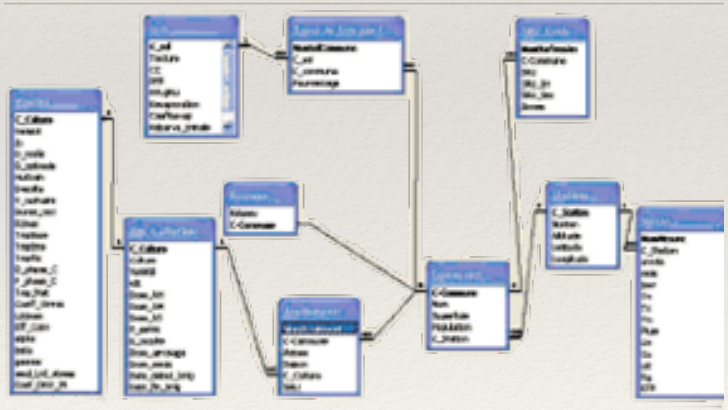
Algorithmique répartie
Master 1
Evry

Partage de ressources

- On aborde ici un des premiers problèmes qui s'est posé lors de l'élaboration des systèmes répartis
- A savoir le partage de ressources communes par plusieurs sites.

Partage de ressources

- Ces ressources peuvent être de plusieurs types:
  - matériels:
    - imprimantes
    - scanners
    - ....
  - logiciels
    - Base de données
    - mémoire commune
    - ....



Propriétés

- Ces ressources partagées ont des propriétés et doivent être gérées en respectant certaines règles d'accès.
- Ici et en TD on va s'intéresser à une ressource qui aura les propriétés suivantes:
  - Elle est complètement partagée (tous les sites peuvent y accéder via le réseau).
  - Au plus M sites peuvent y accéder simultanément.

Remarque

- Si à un moment donné, plus de M sites y accèdent simultanément nous serons dans une **situation d'erreur**
- Il faut donc construire un protocole d'accès à cette ressource qui respecte la condition qu'il n'y ait pas d'erreur

Propriétés

- De manière plus précise, notre protocole devra vérifier les propriétés suivantes pour une ressource à M entrées:
  - **Sûreté**: A chaque instant au plus M sites utilisent la ressource.
  - **Vivacité**: Chaque site demandant un accès à la ressource l'obtiendra au bout d'un temps fini (en l'absence de pannes)
    - On parle d'*absence de famine*.
- Ces deux propriétés sont indispensables pour avoir système:
  - Sûr et «équitable»

Protocole schématique

- Les demandes d'accès sont totalement asynchrones.
- Pour encadrer la bonne utilisation d'une ressource, le protocole est exécuté schématiquement de la façon suivante:

  <Acquisition>

  <Section critique>

  <Libération>

Phase d'acquisition

- La phase d'acquisition consiste pour le site à demander la ressource (nous verrons comment) jusqu'à l'obtenir.

Section critique

- L'utilisation d'une ressource peut être totalement quelconque.
- Nous décrirons pas cette phase qui est propre à chaque ressource.
- Nous supposerons malgré tout que la *phase d'utilisation a une durée finie*.

Phase de Libération

- Après l'utilisation de la ressource, le site qui l'avait la libère pour que d'autre sites puissent l'utiliser à leur tour.

Protocole

- Les protocoles que nous aurons à décrire sont donc constitués des deux phases:
  - Acquisition et Libération

Protocole

- Nous supposerons (sauf précision contraire) que:
  - Chaque site qui est dans sa phase d'acquisition **ne rentre pas** dans une nouvelle phases d'acquisition **avant** d'avoir exécuté sa phase de libération
  - Un site ne demande qu'une ressource à la fois.
  - Il n'y pas de panne.

Ressource à une entrée

- Dans cette partie nous étudions des protocoles pour gérer une ressource avec ***une seule entrée*** (M=1)
- Comment représenter le fait d'utiliser cette ressource?

Droit à la parole

- Comment fait un professeur des écoles qui veut que ses élèves parlent les uns après les autres?
- Approche centralisée:
  - Il donne la parole à ceux qui lèvent la main.
- Approche distribuée:
  - La parole est représentée par un baton.
  - Seul l'élève qui a le baton à le droit de parler.

Billet

- Que garanti un billet pour un organisateur de concert?
  - Que la place a été payé.
  - Qu'il y a autant de personnes que de places.

Lélia Blin

Protocole à jeton

Algorithmique Répartie
Partage de ressources

Jeton

- La ressource est représenté par un jeton
- Le fait qu'il n'y ai qu'un seul jeton qui circule permet de garantir la propriété de **sûreté**
  - Remarque:
    - Il faut être sûr que notre protocole ne duplique pas le jeton

Topologies fixes

- Nous allons voir des protocoles de partages de ressource utilisant un jeton pour des topologie fixes
  - Cycle
  - Arbre

Dans un cycle

- On va supposer le réseau en forme de cycle unidirectionnel.
- L'idée de l'algorithme est la suivante.
  - "Au départ" le jeton est placé de façon arbitraire sur un site.
  - Si un site n'a pas besoin de la ressource il la fait passer à son voisin.
  - Lorsqu'un site a besoin de la ressource il attend le passage du jeton.

Dans un cycle

- Lorsqu'un site demandeur (en phase d'acquisition) reçoit le jeton
  - Il le garde
  - Il l'utilise pour accéder à la ressource
  - Une fois qui la fini d'utiliser la ressource
    - Il a passe à son voisin dans le cycle

Variables du protocole

- Les variables et constante à chaque site p sont les suivantes:
  - **Avoir_jeton**$_p$ $\in$ {Vrai, Faux}
  - **Demandeur**$_p$ $\in$ {Vrai, Faux}
  - **Suivant**$_p$ désigne le suivant dans le cycle
- Initialement les variables sont à Faux
- Sauf pour un site qui a **Avoir_jeton**$_q$ = Vraie

Protocole

- Procédure acquisition

  Demandeur$_p$=Vrai

  Attendre(Avoir_jeton$_p$)

- Procédure libération

  Demandeur$_p$=Faux

  Avoir_jeton$_p$=Faux

  Envoyer (<JETON>) à Suivant$_p$

- Lors de la réception de <JETON>
  - Si (non **Demandeur$_p$**) alors
    - Envoyer (<JETON>) à **Suivant$_p$**
  - Sinon **Avoir_jeton$_p$**=Vrai

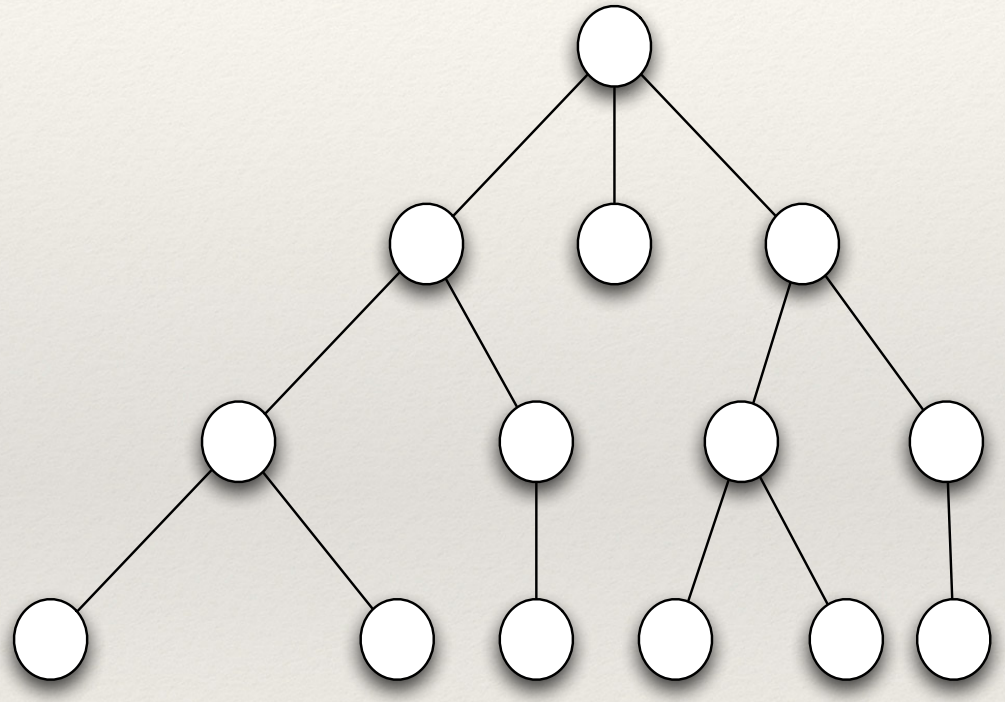
Avantages de ce protocole

- Simplicité
- Il peut être étendu automatiquement si on rajoute des sites au système
- Aucune connaissance globale
- Taille du message O(1)

Inconvénients du protocole

- Longue attente, même sans autre demande
- Le jeton circule constamment
  - parfois pour rien
- Il n'y a pas de priorité temporelle
  - les sites ne sont pas forcément servi dans l'ordre des demandes

Dans un arbre



Variables du protocole

- Les variables et constante à chaque site p sont les suivantes:
  - **Avoir_jeton**$_p$ $\in$ {Vrai, Faux}
  - **En_SC**$_p$ $\in$ {Vrai, Faux}
  - **Racine**$_p$ identifiant de site qui indique à p
    - vers quel voisin dans l'arbre il doit demander la ressource
  - **Request**$_p$ est une file d'identificateurs de site, initialement vide
- Initialement les variables sont à Faux
- sauf pour un site qui a **Avoir_jeton**$_q$ = Vraie

Protocole

- Procédure d'acquisition

```
Si (non Avoir_jeton_p)
    Si (File_vide(Request_p))
        Envoyer (<DEMANDE>) à Racine_p
    Ajouter (Request_p,p)
    Attendre (Avoir_jeton_p)
Sinon En_SC_p:=vrai
```

- Procédure de Libération

```
En_SC_p=Faux
si (non File_Vide(Request_p))
    Racine_p:=Defiler(Request_p)
    Envoyer <JETON> à Racine_p
    Avoir_jeton_p:=Faux
    si (non File_Vide(Request_p))
        Envoyer <DEMANDE> à Racine_p
```

- Reception de <DEMANDE > envoyé par q

```
Si(Avoir_jeton_p)
    si (En_SC_p=vrai)
        Ajouter(Request_p,q)
    Sinon
        Racine_p=Defiler(Request_p)
        Envoyer <JETON> à Racine_p
        Avoir_jeton_p=Faux
Sinon
    Si (FileVide(Request_p)) alors
        Envoyer <DEMANDE> à Racine_p
        Ajouter(Request_p,q)
```

- Reception de <JETON> envoyé par q

```
Racine_p=Defiler(Request_p)
Si (Racine_p=p) alors
    Avoir_jeton_p=Vrai
Sinon
        Envoyer <JETON> à Racine_p
        Si (FileVide(Request_p)) alors
            Envoyer <DEMANDE> à Racine_p
```

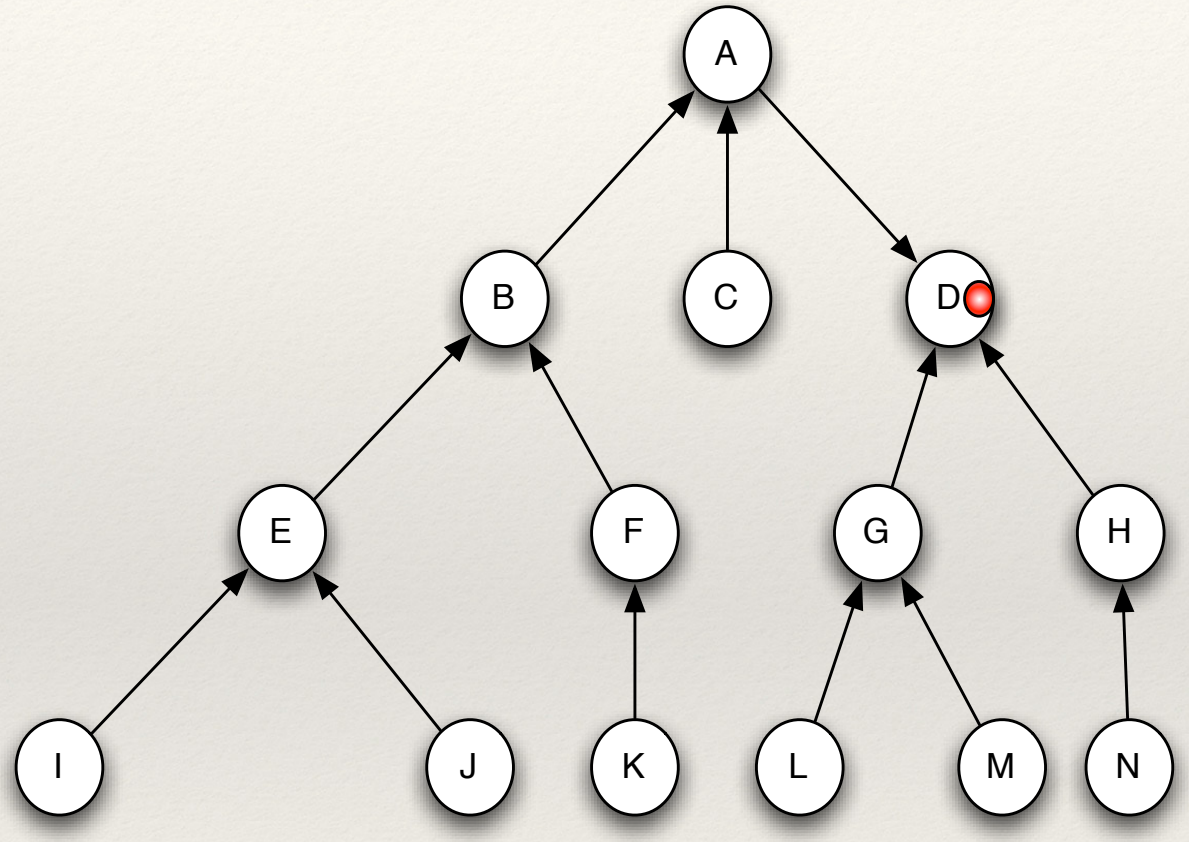
Remarques

- La file $\texttt{Request}_p$ sert à ordonner les demandes de chaque sous-arbre
- Notons:
  - Si un site reçoit un message <DEMANDE>
    - Si sa file est vide
      - le site le transmet dans son sous-arbre dans lequel il y a le jeton
    - Si sa file n'est pas vide
      - Le site bloque la demande à son niveau
      - Le site débloquera la demande
        - que lorsque les autres demande qui sont avant dans la file
          - auront été servies.

Remarques

- Lors de la phase de libération
  - le site possédant le jeton le transmet
    - au premier qui lui a demandé
    - autrement dit le premier de sa file
  - Si après l'envoie du jeton sa file reste non vide:
    - il transmet une nouvelle demande
    - pour que le suivant dans la file puisse être servi.
- La file d'attente permet d'assurer la vivacité du mécanisme

Exemple



E fait une demande de jeton, suivit par une demande de N et avant que E soit servi, K fait une demande.

Avantage

- Le jeton circule que si il y a des demandes
- L'ordre des demandes est plus respecté que dans l'algorithme sur le cycle
- Taille des messages: O(1) bits
- Nombre de messages par demande est proportionnel au diamètre de l'arbre en moyenne O(log n) messages

Lélia Blin

Protocole à base de permissions

Algorithmique Répartie
Partage de ressources

Estampillage

- Pour ces protocoles nous avons besoin d'estampiller les message avec des marques d'horloge
- Aucune horloge globale n'est disponible
- On va donc utiliser un mécanisme à base d'estampillage basé sur les *horloges logiques* locale.
- Ce mécanisme peut être vu comme un sous-protocole

Estampillage protocole

- variable locale: $horloge_i$
- Initialisation:
  - $horloge_i$:=0
- Lors de la **réception** d'un message <M,*h*>
  - $horloge_i$=max{$horloge_i$,*h*}+1
  - Délivrer(M)
- Lors de l'**envoie** des données M
  - $horloge_i$=$horloge_i$+1
  - Envoyer(< M,$horloge_i$ >)

Remarques

- Les messages sont donc transportés avec l'heure logique à laquelle ils ont été émis.
- Remarque:
  - On peut se retrouver avec deux message ayant la même estampille
  - Le moyen usuel dans ce cas et de prendre en compte l'identifiant de l'envoyeur

Estampillage

- A la place de considérer uniquement l'horloge logique
  - On prendre en compte le **couple** (*h,i*) où i est l'identifiant du site émetteur

Priorité

- L'ordre sur le couple (h_i,i) et (h_j,j) ou i et j sont les identifiant des émetteurs des messages est le suivant
- $(h_i,i)<(hj,j) \Leftrightarrow (h_i<h_j$ ou $(h_i=h_j$ et i<j))

Priorité

- Ce qui se traduit par i est plus prioritaire que j
- Car:
  - Soit la demande de i a été faite avant celle de j en prenant en compte les horloges h_i et h_j
  - Soit les horloges sont les mêmes mais l'identité de i est plus petite que celle de j
  - Cet ordre est donc total
    - Toutes les paires sont comparables

Lélia Blin

L'Algorithme de Ricart et Agrawala

Algorithmique Répartie
Partage de ressources

Principe de l'algorithme

- Dans cet algorithme
  - Chaque site désirant la ressource va demander le permission à tous les autres

Conflits

- On départage les conflits en utilisant les étiquettes
- Chaque étiquette donnant l'heure à laquelle la demande a été faite.
- Rappel: les demandes les plus anciennes sont les plus prioritaires.

Variables locales

- Chaque site i maintient les variables locales suivantes:
  - $Heure\_demande_i$ l'heure à laquelle le site i a fait sa dernière demande
  - $Rep\_attendues_i$ est un entier qui compte le nombre de réponses attendu par i suite à sa demande
  - $Demandeur_i$ booléen qui est à vrai si le site a demandé la ressource faux sinon (initialisé à faux)
  - $differe_i[j]$ tableau de n cases de booléens, $differe_i[j]$=vrai ssi i a différé sa réponses à la demande de j; initialisé à faux

Protocole

- Procédure d'acquisition

  $Heure\_demande_i := horloge_i$

  $Demandeur_i := vrai$

  $Rep\_attendues_i := n-1$

  pour tout ($x_i \in V-\{i\}$) faire

  Envoyer(<DEMANDE, ($Heure\_demande_i$,i)>) à x_i

  Attendre($Rep\_attendues_i = 0$)

- Procédure de Libération

  $Demandeur_i := faux$

  pour tout ($x_i \in V-\{i\}$) faire

  si ($differe_i[x_i]$) alors

  Envoyer(<REPONSE >) à x_i

  $differe_i[x_i]$=faux

- Réception de <DEMANDE,h,j> envoyé par j

  Si (non $Demandeur_i$ *ou* ($Heure\_demande_i$,i)>(h,j))

  Envoyer(<REPONSE >) à *j*

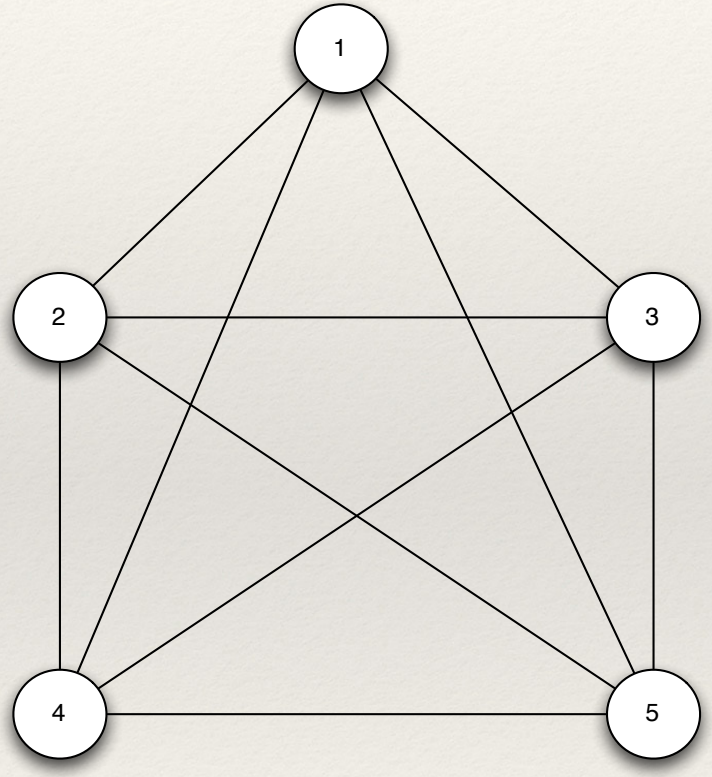
  Sinon

  $differe_i[j]$:=vrai

- Réception de <REPONSE> envoyé par j

  $Rep\_attendues_i := Rep\_attendues_i - 1$

Exemple



- 2 et 3 sont demandeurs avec un h égal à 1
- 4 est demandeur avec un h égal à 2

Remarques

- La complexité est de 2(n-1) messages par demande
- Inconvénient
  - Une demande est faite à tous les autres sites ce qui entraîne un grand nombre de messages

Lélia Blin

L'algorithme de Maekawa et variantes

Algorithmique Répartie
Partage de ressources

Améliorations

- Pour améliorer le nombre de message
  - Au lieu de demander un si grand nombre de permissions
  - On peut imaginer que chaque site i a un sous ensemble S_i à qui il va demander la permission d'utiliser le ressource

Quorum

- Un tel ensemble S_i sera appelé quorum
- Pour être sûr que la propriété de sûreté est vérifiée
  - il faut que les quorums vérifient la propriété suivante
    - Règle d'intersection
      - si $i \neq j$ lors $S_i \cap S_j \neq \emptyset$

Règle d'intersection

- C'est propriété est indispensable
  - pour s'assurer que lorsque deux sites i et j demandent les permissions pour entrer en section critique
  - les sites qui sont à la fois dans S_i et S_j ne peuvent pas accorder la permission aux deux
  - Ainsi en cas de demandes simultanées
    - au moins un des sites n'aura pas une permission
- La présence de l'estampillage reste obligatoire pour départager les demandes multiples

Diminuer la charge

- On ne veut pas que la charge de travail relatif au demande soit supporter que par un seul site
- On veut équilibrer la charge de travail que doit faire chaque site pour la communauté
  - même si il a pas besoin de la ressource
- En même temps on veut réduire la complexité

Formellement

- On veut que chaque site i fasse partie de D quorums S
  - et que pour tout j $|S_j|=k$
- Il faut donc minimiser k et D
- Rm: Dans ce cas la complexité est en O(k) messages.

Quorum

- On prend un quorum contant k membres
- qui font partie d'au plus D-1 autre ensemble
- Ainsi le maximum de quorum qui peut être construits est de
  - n=k(D-1)+1

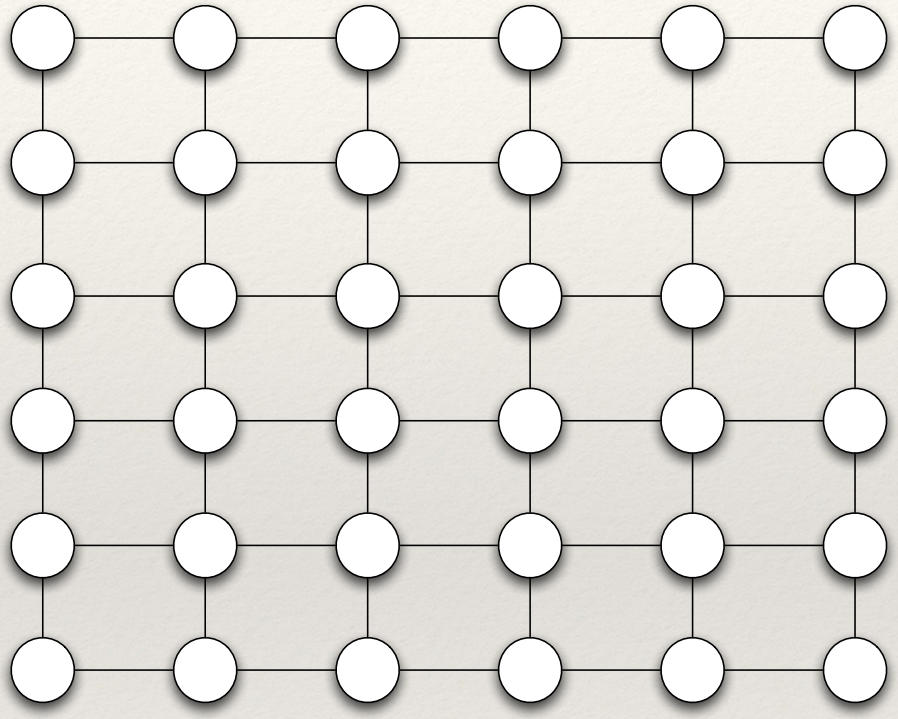
Quorums

- De plus Dn/k est le nombre maximum de quorums
- car chaque site peut être dans au plus D quorums
  - contenant chacun au plus k sites
- Si il y autant de quorum que de sites on a
  - n=Dn/k c'est à dire D=k
- En combinant ses égalités on obtient
  - n=k(k-1)+1 et k=O(√n)

Construction

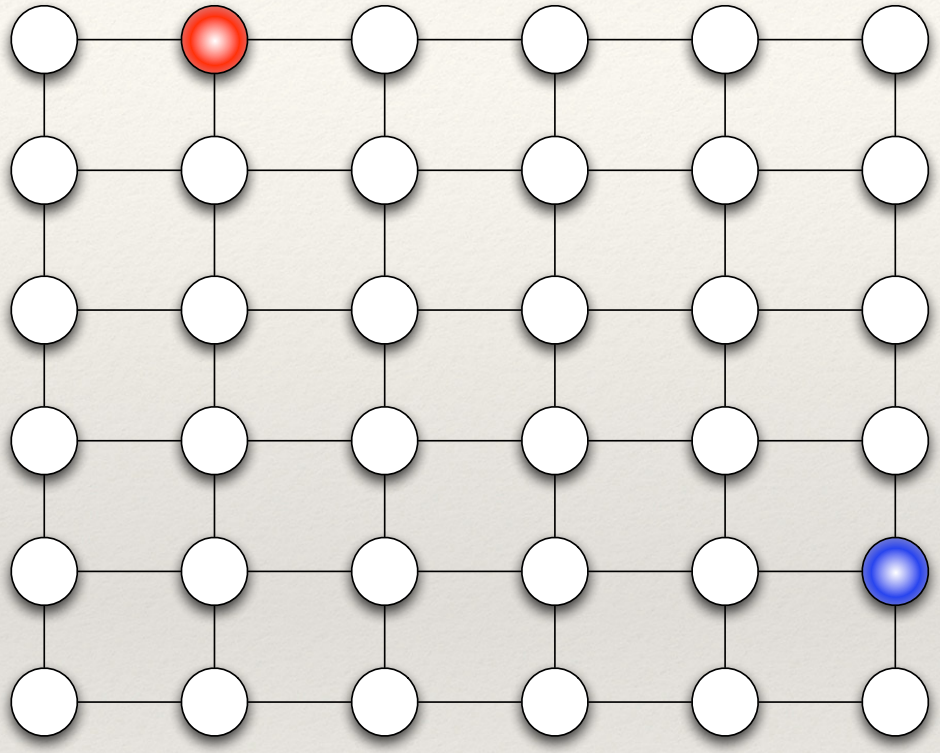
- Construire effectivement un quorums avec ses propriétés est difficile
- Par contre si $n=p^2$
  - il est facile de construire n quorums
  - avec $\sqrt{n}=p$ sites dans chacun d'eux

Grille 6x6



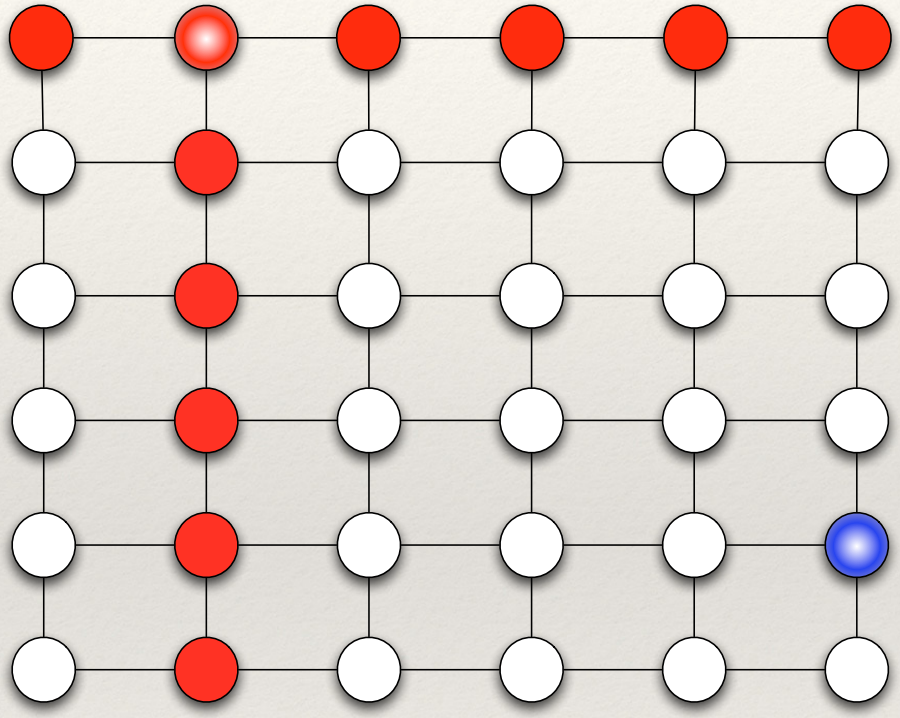
Avec cette construction toute les propriétés sont vérifiées

Grille 6x6



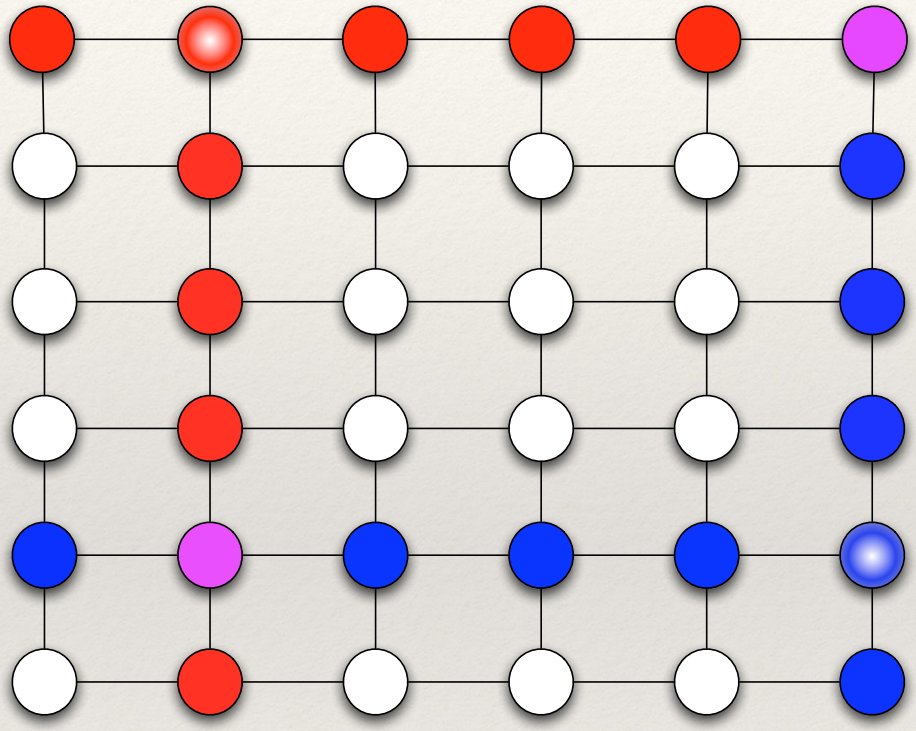
Avec cette construction toute les propriétés sont vérifiées

Grille 6x6



Avec cette construction toute les propriétés sont vérifiées

Grille 6x6



Avec cette construction toute les propriétés sont vérifiées

Algorithme pour grille pxp

- Avec ce système chaque site i connaît son quorum S_i
- Chaque fois que i demande la ressource
  - il demande la permission
  - estampillée par son horloge
  - à tous les sites de S_i

Algorithme pour grille pxp

- Lorsqu'un site i reçoit une demande de la part de j
  - on peut être dans trois cas
  - Premier cas:
    - i ne demande pas la ressource il envoie sa permission à j

Algorithme pour grille pxp

- Deuxième cas:
  - i a demandé la ressource
  - si sa demande a une estampille plus ancienne que celle de j
    - il lui envoie un message pour dire non
  - sinon il lui donne la permission

Algorithme pour grille pxp

- Troisième cas:
  - i a déjà donné sa permission à un site u
    - dont l'heure de demande avait une estampille plus récente que celle de j.
  - Dans ce cas i va essayer de récupérer la permission qu'il a donné à u pour la donner à j.
  - Si u a reçu un message de refus de permission
    - il redonne la permission à i qui va la donner à j.