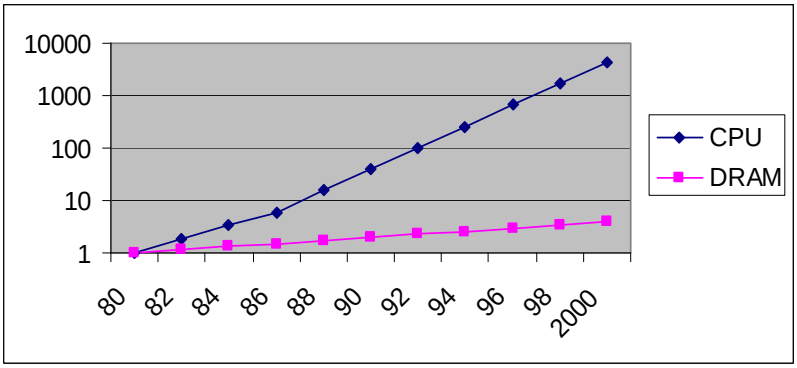
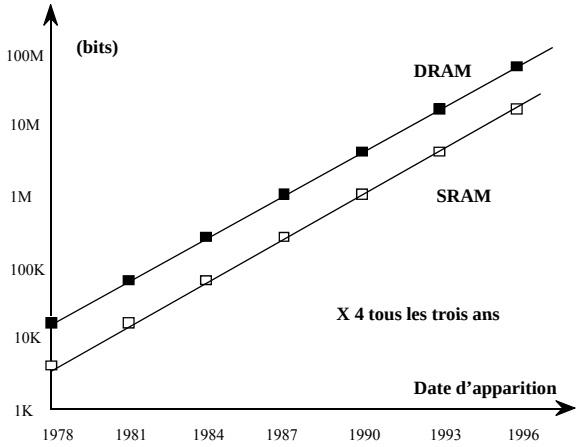
# Accès aux données
# Caches et alternatives

Daniel Etiemble
de@lri.fr

## Un différentiel croissant

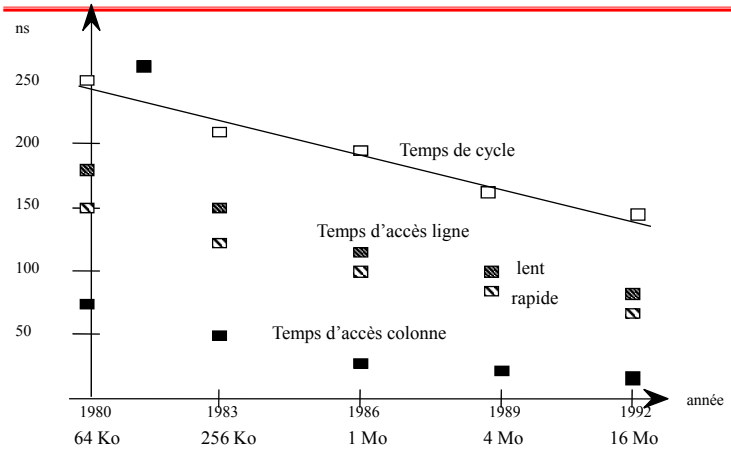


# Capacité mémoire



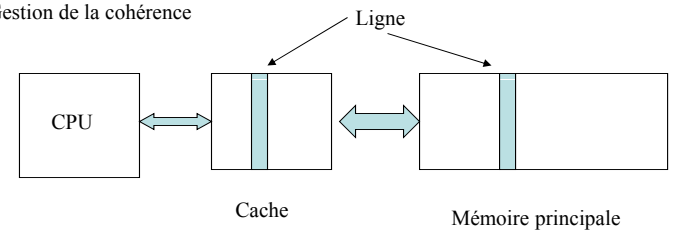



# Caractéristiques temporelles des DRAM



# Principe des caches

- Fondés sur le principe de localité
- Mémoires de taille et vitesse différentes
- Organisation
  - Découpage en lignes (blocs)
  - Mécanisme de correspondance
    - Placement des lignes dans le cache
    - Détection succès ou échec
  - Gestion de la cohérence



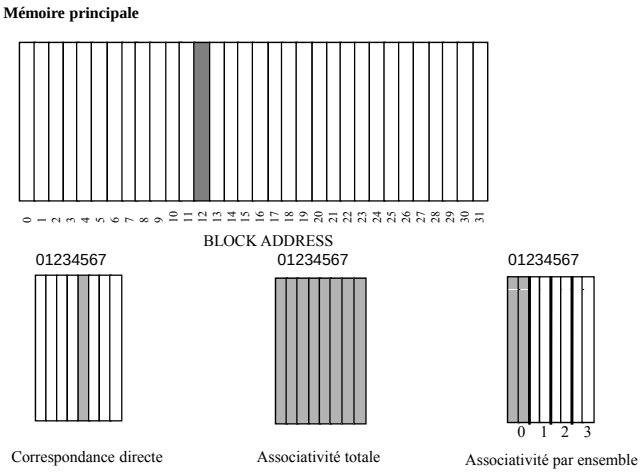



# Principe de localité

- Localité spatiale
  - Si on accède à une case mémoire, on accédera à une case proche
- Localité temporelle
  - si on accède à une case mémoire, on y accédera probablement très bientôt (ou dans très longtemps !)

# Mécanismes de correspondance

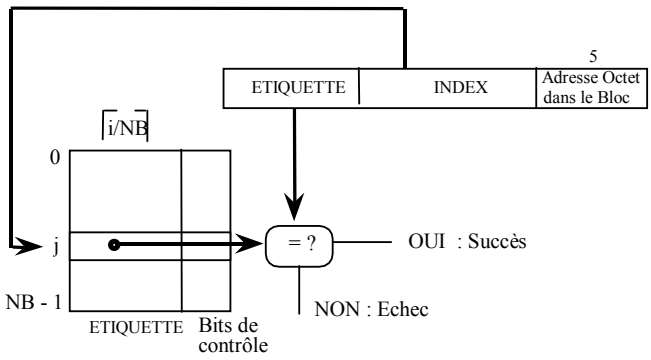
OU PLACER UNE LIGNE DE LA MP DANS LE CACHE ?





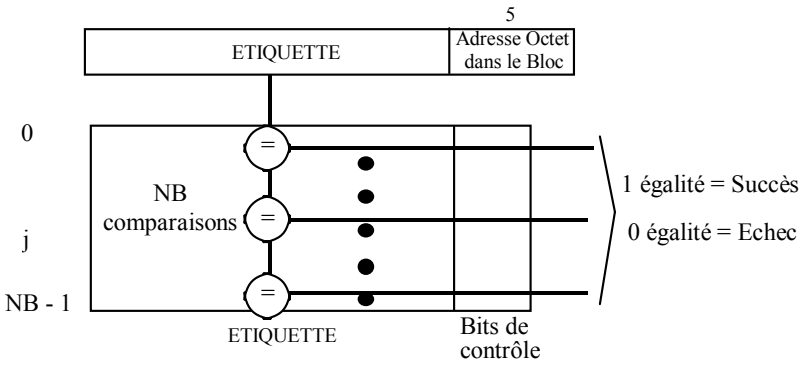

# Correspondance directe

j (cache) = i (MP) mod NB, avec NB = nombre de lignes du cache



# Associativité totale

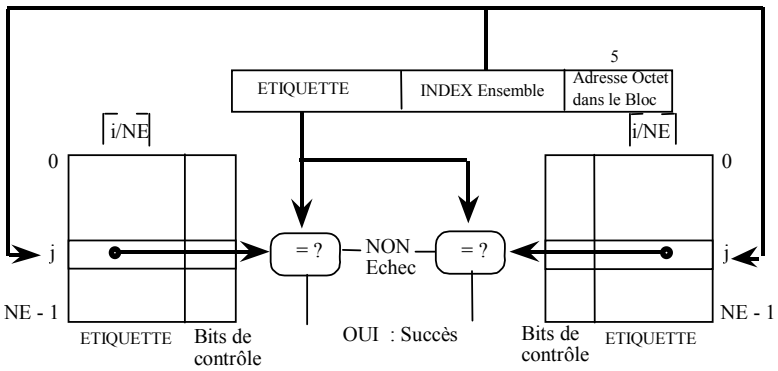
Ligne i(MP) dans n'importe quelle ligne j du cache.
Etiquette = i, et NB comparateurs pour détecter succès ou échec





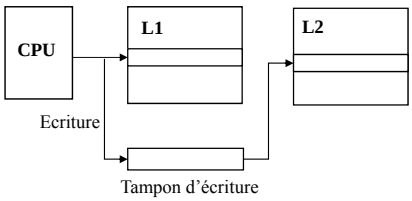

# Associativité par ensemble

N (2, 4, 8) lignes par ensemble. Correspondance directe pour les ensembles et associativité à l'intérieur d'un ensemble.
N comparateurs.

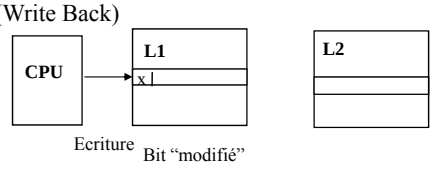


# Politique en écriture

Ecriture simultanée
(Write Through)



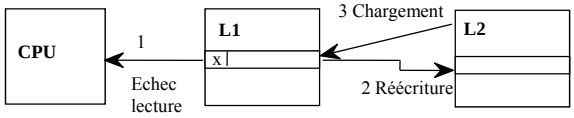

Réécriture
(Write Back)




Cohérence au plus tard

Cohérence
au plus tôt

Réécriture **Réécriture quand une ligne modifiée est remplacée**

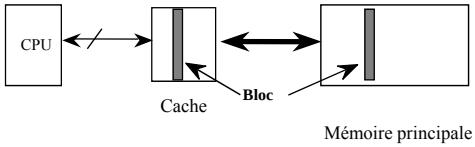





# Politiques de remplacement

- Ligne à remplacer ?
  - Least Recently Used (LRU)
  - Random
  - Least Frequently Used (LFU)

# CPI Mémoire




$CPI_{mémoire}$ = ma x m x p

ma: accès mémoire/instruction

m: taux d'échec

p : pénalité d'échec (cycles d'horloge)



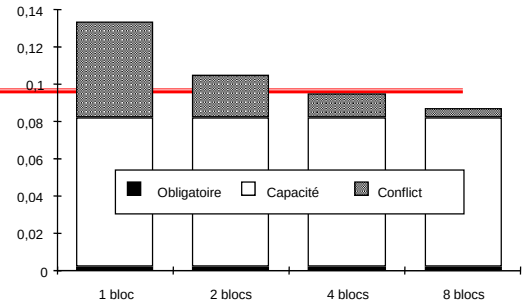
# Améliorer les performances des caches

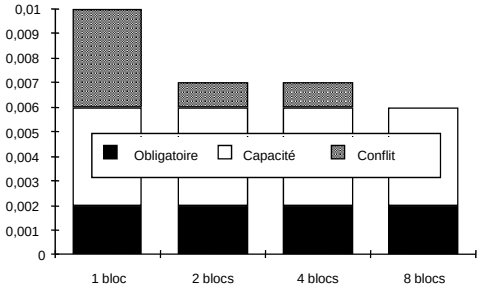
- Diminuer le taux d'échec
  - Taille (cache, ligne) et associativité
  - Optimisations du compilateur
  - Préchargement
- Diminuer le temps d'accès réussi (hit)
  - Petits caches
  - Eviter les traductions d'adresse
- Diminuer la pénalité d'échec
  - Caches non bloquants
  - Caches de second niveau

# Les 3C

- Echecs de démarrage
- Echecs de capacité
- Echecs de conflit

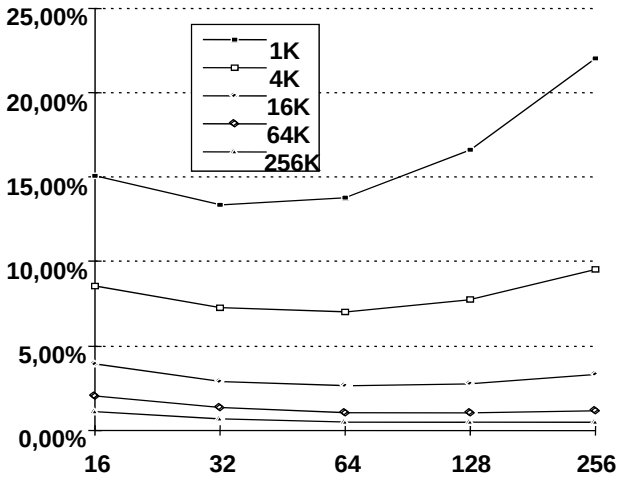
SPEC92
Lignes de 32 octets - LRU
DECStation5000



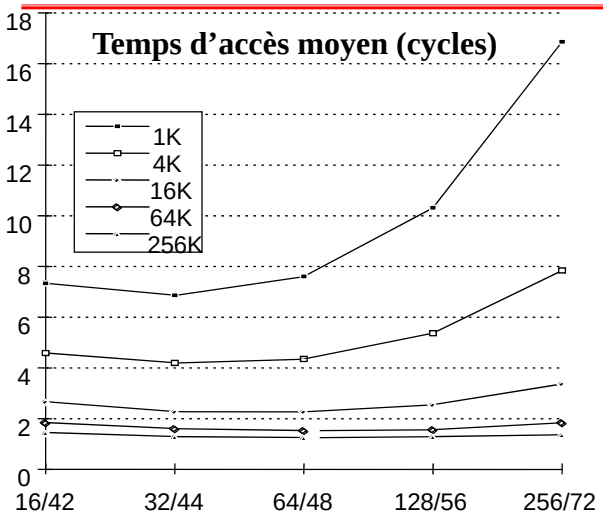






# Taux d'échec (Taille de ligne)



# Temps d'accès (taille de ligne)




Hypothèse

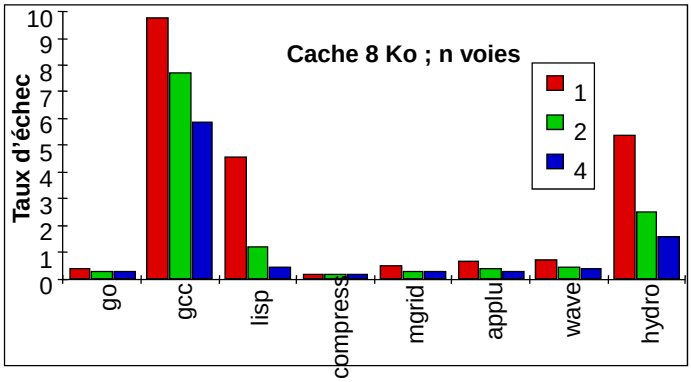
40 cycles de démarrage, puis 16 octets tous les 2 cycles

Taille de ligne / Pénalité d'échec



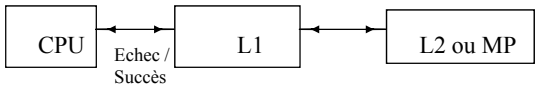
# Taux d'échecs (associativité)

Le taux d'échec décroît quand l'associativité croit, mais les lectures "optimistes" ne peuvent plus être utilisées (nécessité de prédicteurs d'ensemble)

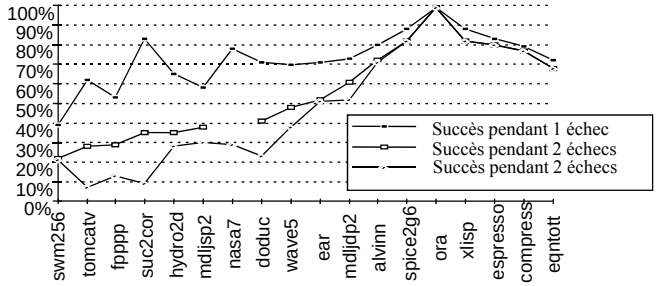


# Caches non bloquants

Succès pendant échecs : le cache continue à fournir les données des accès réussis pendant qu'il traite un échec.



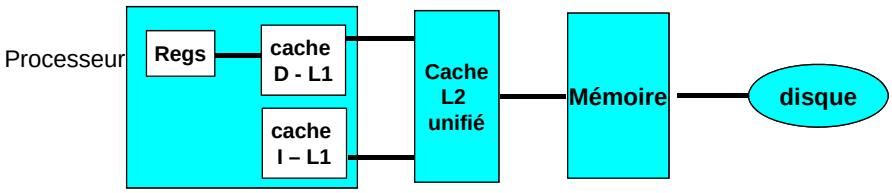

Rapport moyen entre les temps de suspension d'un cache non bloquant/cache bloquant






# Plusieurs niveaux de cache

- Options: caches instructions et données séparés ou cache unifié

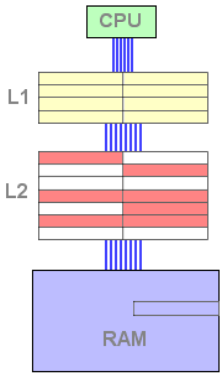



| | | | | | |
|---|---|---|---|---|---|
| Taille : | 200 o | 8-64 Ko | SRAM 1-4Mo | DRAM 128 Mo | 30 Go |
| Vitesse | <1 ns | <1 ns | 6 ns | 60 ns | 8 ms |
| €/Moctet : | | | 100 €/Mo | 1.50€/Mo | 0.05€/Mo |
| Taille ligne : | 8 o | 32 o | 32 o | 8 Ko | |

Plus gros, plus lent, moins cher

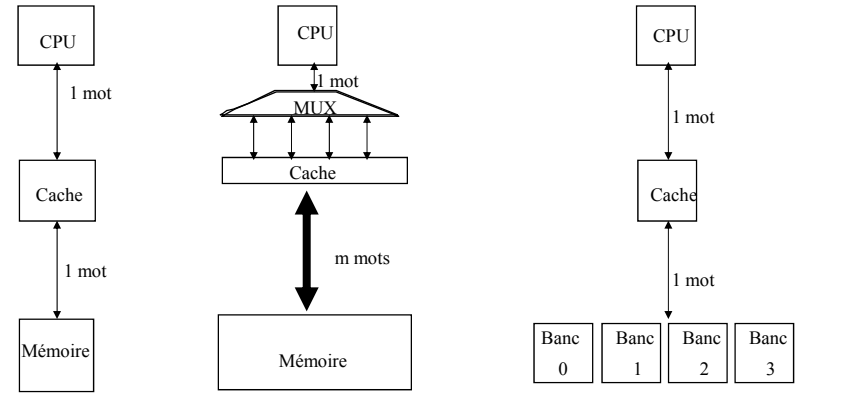
# Caches inclusifs ou exclusifs

- Caches inclusifs
  - L1⊂ L2 ⊂ MP
- Caches exclusifs
  - L1 ⊄ L2
  - Fonctionnement
    - Echec L1 : bloc de MP dans L1
      - Si remplacement, bloc remplacé dans L2 et bloc demandé dans L1
    - Echec L1 et Succès L2
      - Remplacement d'un bloc dans L1 (copié dans L2) et bloc demandé de L2 dans L1
    - Nécessité d'un tampon des victimes
      - Lors d'un remplacement, le bloc remplacé est placé dans un tampon pour ne pas retarder le transfert L2 vers L1



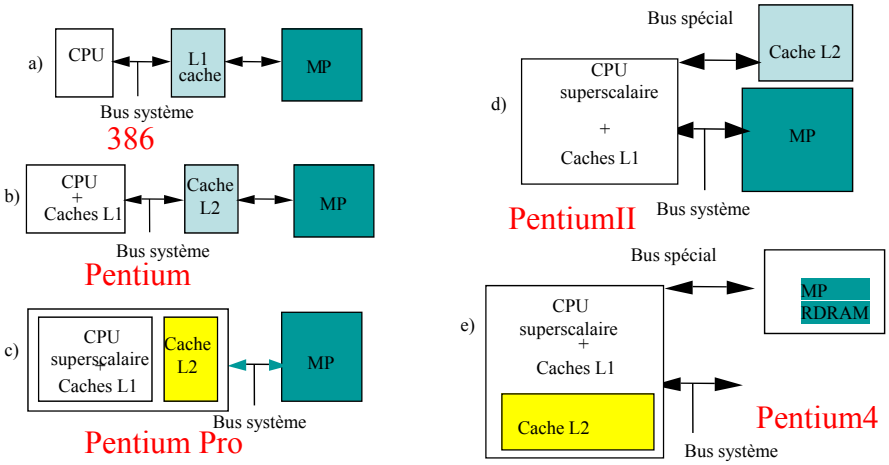



# Liaison cache-mémoire principale




a) Mémoire 1 mot b) Mémoire large b) Mémoire entrelacée m bancs

# Evolution des hiérarchies mémoire (1985-2000)






# Caches données des processeurs Intel

**CACHE PRIMAIRE**

| | Taille | Assoc. | Lignes | Latence | Ecriture |
|---|---|---|---|---|---|
| Pentium Pro | 8 Ko | 2 voies | 32 oct | 3 | Réécriture |
| Pentium III | 16 Ko | 4 voies | 32 oct | 3 | Réécriture |
| Pentium 4 | 8 Ko | 4 voies | 64 oct | 2/6 | Simultanée |

**CACHE SECONDAIRE (UNIFIE)**

| | Taille | Assoc. | Lignes | Latence | Ecriture |
|---|---|---|---|---|---|
| Pentium Pro | 512 Ko | 4 voies | 32 oct | | Réécriture |
| Pentium III* | 256 Ko | 8 voies | 32 oct | 6 | Réécriture |
| Pentium 4 | 256 Ko | 8 voies | 128 oct | 7/7 | Réécriture |

## Amélioration des performances cache

- Réduire le taux d'échec
  - Taille de cache, de ligne et degré d'associativité — ***HW***
  - Optimisations du compilateur — ***SW***
  - Préchargement — ***HW*** ***SW***
- Réduire le temps de l'accès réussi
  - Prédiction d'ensemble — ***HW***
  - Eviter les traductions d'adresse — ***HW***
- Réduire la pénalité d'échec
  - Caches non bloquants — ***HW***
  - Caches de second niveau — ***HW***



## Code en fonction du cache

- Accès répété à des variables est favorable (localité temporelle)
- Accès avec un pas de 1 est favorable (localité spatiale)
- Exemples:
  - Cache à froid, mots de 4 octets, lignes de cache de 4 mots

```
int sumarrayrows(int a[M][N])
{
    int i, j, sum = 0;

    for (i = 0; i < M; i++)
        for (j = 0; j < N; j++)
            sum += a[i][j];
    return sum;
}
```

**Taux d'échec = 1/4 = 25%**

```
int sumarraycols(int a[M][N])
{
    int i, j, sum = 0;

    for (j = 0; j < N; j++)
        for (i = 0; i < M; i++)
            sum += a[i][j];
    return sum;
}
```

**Taux d'échec = 100%**

# Exemple : multiplication de matrices

- Aspects cache à considérer
  - Taille totale du cache
    - Exploiter la localité temporelle et conserver l'ensemble de travail petit (utilisation du blocage)
  - Taille de ligne
    - Exploiter la localité spatiale
- Description:
  - Multiplication de matrice N x N
  - $O(N^3)$ opérations
  - Accès
    - N lecture par élément
    - N valeurs sommées par destination
      - Mais peuvent être conservées dans des registres

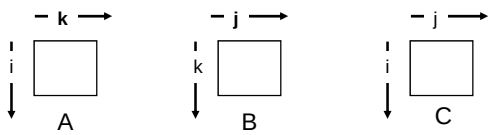
```
/* ijk */
for (i=0; i<n; i++)  {
  for (j=0; j<n; j++) {
    sum = 0.0;
    for (k=0; k<n; k++)
      sum += a[i][k] * b[k][j];
    c[i][j] = sum;
  }
}
```

*sum dans un registre*



# Analyse du taux d'échecs pour la multiplication de matrices

- Hypothèses:
  - Taille de ligne = 32 octets (assez grand pour 4 mots de 64 bits)
  - La taille des matrices (N) est très grande
    - $1/N \rightarrow 0.0$
  - La taille du cache est insuffisante pour contenir plusieurs lignes
- Méthode d'analyse:
  - Examiner les accès dans la boucle interne



# L'allocation mémoire des tableaux C

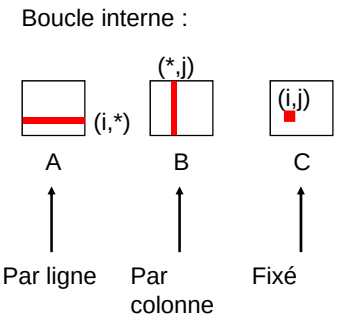
- Les tableaux C sont alloués dans l'ordre ligne d'abord
  - Ligne dans des cases mémoires contigues
- Balayage à travers les colonnes dans une ligne :
  - `for (i = 0; i < N; i++)`
    `sum += a[0][i];`
  - Accède aux éléments successifs
  - Si la taille de bloc (B) > 4 octets, exploite la localité spatiale
    - Taux d'échecs obligatoires = 4 octets / B
- Balayage à travers les lignes dans une colonne :
  - `for (i = 0; i < n; i++)`
    `sum += a[i][0];`
  - Accèdent à des éléments distants
  - Aucune localité spatiale !
    - Taux d'échecs obligatoires = 1 (i.e. 100%)



# Multiplication de matrices (ijk)

```
/* ijk */
for (i=0; i<n; i++)  {
  for (j=0; j<n; j++) {
    sum = 0.0;
    for (k=0; k<n; k++)
      sum += a[i][k] * b[k][j];
    c[i][j] = sum;
  }
}
```

PRODUIT SCALAIRE




- Echecs par itération de la boucle interne:

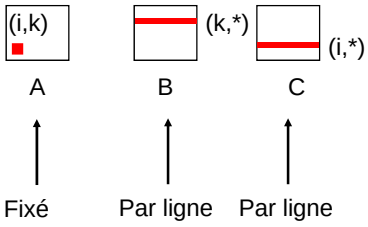
| A | B | C |
|---|---|---|
| 0,25 | 1,0 | 0,0 |

# Multiplication de matrices (ikj)

```
/* ikj */
for (i=0; i<n; i++) {
   c[i][0]=0.0;
  for (k=0; k<n; k++) {
    r = a[i][k];
    for (j=0; j<n; j++)
      c[i][j] += r * b[k][j];
  }
}
```

SAXPY/DAXPY

Boucle interne :




- Echecs par itération de la boucle interne:

| A | B | C |
|---|---|---|
| 0,0 | 0,25 | 0,25 |



# Multiplication ijk après transposition

```
 for (i=0; i<n; i++)
   for (j=0; j<n; j++)
        bt[j][i] = b[i][j]


/* ijk */
for (i=0; i<n; i++)  {
  for (j=0; j<n; j++) {
    sum = 0.0;
    for (k=0; k<n; k++)
      sum += a[i][k] * bt[j][k];
    c[i][j] = sum;
  }
}
```

Défauts de cache
~ $N^2$

Défauts de cache
~ $N^3$

## Améliorer la localité temporelle par blocage

- Exemple : multiplication de matrices avec blocage
  - "bloc" (dans ce contexte) ne signifie pas "bloc de cache".
  - C'est un sous bloc de la matrice.
  - Exemple: N = 8; taille de sous bloc = 4

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \times \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix} = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix}$$

Idée de base : Un sous bloc (i.e., $\mathbf{A}_{xy}$) peut être traité comme un scalaire.

$$C_{11} = A_{11}B_{11} + A_{12}B_{21} \qquad C_{12} = A_{11}B_{12} + A_{12}B_{22}$$

$$C_{21} = A_{21}B_{11} + A_{22}B_{21} \qquad C_{22} = A_{21}B_{12} + A_{22}B_{22}$$



## Multiplication par bloc (bijk)

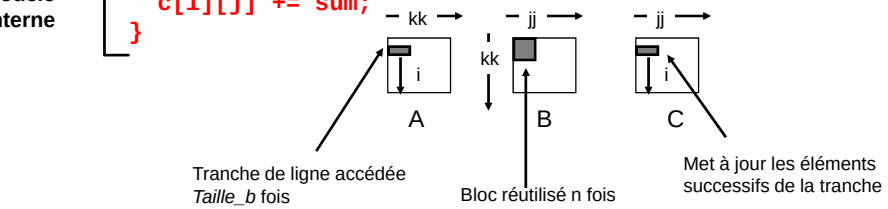
```
for (jj=0; jj<n; jj+=bsize) {
  for (i=0; i<n; i++)
    for (j=jj; j < min(jj+bsize,n); j++)
      c[i][j] = 0.0;
  for (kk=0; kk<n; kk+=bsize) {
    for (i=0; i<n; i++) {
      for (j=jj; j < min(jj+bsize,n); j++) {
        sum = 0.0
        for (k=kk; k < min(kk+bsize,n); k++) {
          sum += a[i][k] * b[k][j];
        }
        c[i][j] += sum;
      }
    }
  }
}
```

## Analyse de la multiplication de matrices par blocs

- La boucle interne multiplie une tranche de *A* (*1 X tailleb)* par un bloc de *B* *(taille_b X taille_b)* et accumule dans une tranche de C (*1 X taille_b*)
- *i* étapes à travers *n* tranches de lignes of *A* & *C*, en utilisant le même *B*

```
for (i=0; i<n; i++) {
  for (j=jj; j < min(jj+bsize,n); j++) {
    sum = 0.0
    for (k=kk; k < min(kk+bsize,n); k++) {
      sum += a[i][k] * b[k][j];
    }
    c[i][j] += sum;
  }
```

Boucle interne




Tranche de ligne accédée *Taille_b* fois

Bloc réutilisé n fois

Met à jour les éléments successifs de la tranche



## Optimisations du compilateur

*Améliore la localité spatiale*

### ECHANGE DE BOUCLES

```
for (j=0; j<100; j++)
   for (i=0; i<5000; i++)
      x[i][j] = 2*x[i][j];
```

```
for (i=0; i<5000; i++)
   for (j=0; j<100; j++)
      x[i][j] = 2*x[i][j];
```

### FUSION DE TABLEAUX

```
int val[SIZE];
int key[SIZE];
```

```
struc merge {
        int val;
;       int key; }
struc merge merge-array [SIZE];
```

# Optimisations du compilateur

*Améliore la localité temporelle*

**FUSION DE BOUCLES**

```
for (i=0; i<N; i++)
    for (j=0; j<N; j++)
       a[i][j] = 1/b[i][j]*c[i][j];

for (i=0; i<N; i++)
    for (j=0; j<N; j++)
       d[i][j] = a[i][j]+c[i][j];
```

```
for (i=0; i<N; i++)
    for (j=0; j<N; j++)
      {
       a[i][j] = 1/b[i][j]*c[i][j];
       d[i][j] = a[i][j]+c[i][j];
      }
```



# Optimisations du compilateur

**BLOCAGE**

*AMELIORE LA LOCALITE TEMPORELLE*

```
for (i=0; i<N; i++)
    for (j=0; j<N; j++)
        {r=0;
        for (k=0; k<N; k++)
            r+=y[i][k]*z[k][j];
        x[i][j]=r;}
```

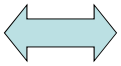
```
for (jj=0; jj<N; jj+=B)
for (kk=0; kk<N; kk+=B)
for (i=0; i<N; i++)
    for (j=jj; j<(min(jj+B-1,N); j++)
        {r=0;
        for (k=kk; j<(min(kk+B-1,N); k++)
            r+=y[i][k]*z[k][j];
        x[i][j]+=r;}
```

B : facteur de blocage

# Compatibilités entre optimisations

- Pas unitaire est efficace pour les caches et pour la vectorisation
- Mais contre-exemples

```
for (j=0; j<2; j++)
    for (i=0; i<1000; i++)
        x[i][j] = 0.0;
```



```
for (i=0; i<1000; i++)
    for (j=0; j<2; j++)
        x[i][j] = 0.0;
```

Le tableau tient dans le cache
Réutilisation totale

Boucle interne trop courte

```
for (j=1; j<m; j++)
    for (i=0; i<n; i++)
        x[i][j] = x[i][j] + x[i][j-1];
```

```
for (i=0; i<n; i++ )
    for (j=1; j<m; j++)
        x[i][j] = x[i][j] + x[i][j-1];
```

Bon comportement du cache, mais récursion sur j



# Préchargement matériel

- Précharger les instructions ou les données avant qu'elles ne soient nécessaires (charger une ligne avant un échec cache)
- Quelle ligne ?
  - La suivante (préchargement séquentiel selon la localité spatiale)
  - Une ligne prédite
- Quand ?
  - Toujours
  - Sur un échec
  - Sur un échec et lorsqu'on accède une donnée préchargée
- Où ?
  - Dans le cache (pollution potentielle)
  - Dans un tampon

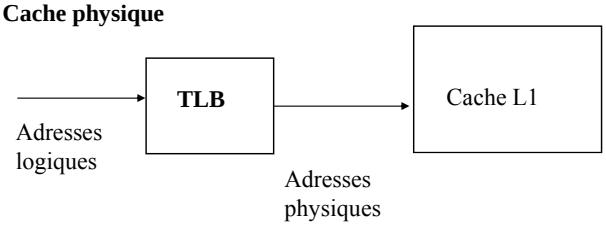
# Préchargement logiciel

- Instructions de préchargement (IA-32)
  - PREFETCHT0 m8 : tous les niveaux de cache
  - PREFETCHT1 m8: tous les niveaux sauf L0
  - PREFETCHT2 m8: tous les niveaux sauf L0 et L1
  - PREFETCHNTA m8: préchargement dans une structure non temporelle
- Indications de préchargement
  - Dépendent des implémentations

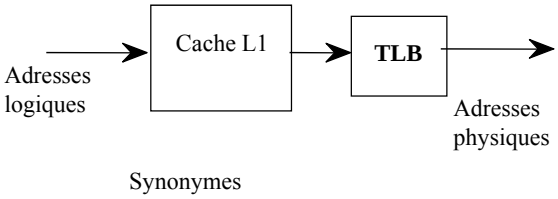


# Cache et TLB

**Cache physique**



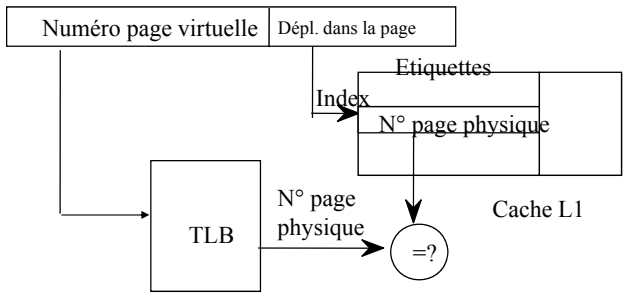

**Cache virtuel**




Synonymes

# Cache et TLB

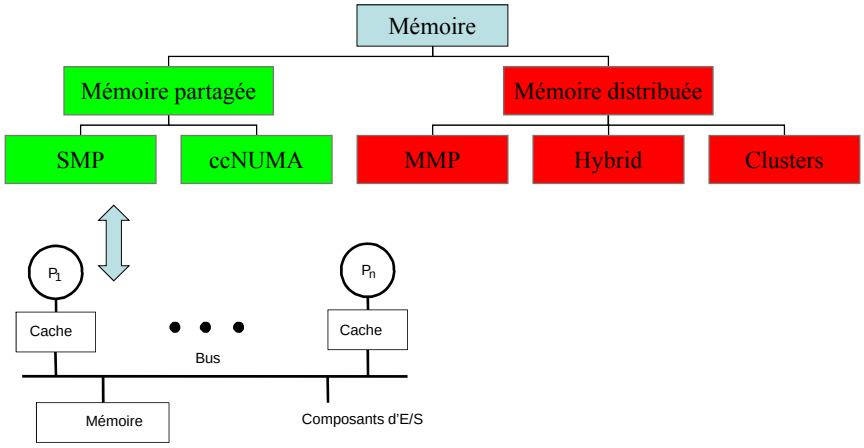
**Caches avec index virtuels et adresses physiques**




TLB et cache sont accédés en parallèle
Avec la correspondance directe, taille du cache = taille de page



# Mémoire des machines parallèles




**Modèle mémoire SMP**

## Les mémoires partagées à bus

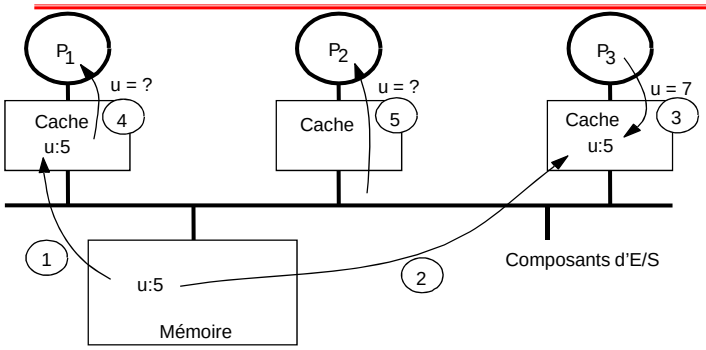
- Les multiprocesseurs symétriques (SMP)
  - Pour tout processeur, accès symétrique à l'ensemble de la mémoire principale
- Dominent le marché des serveurs
  - Briques de base pour des systèmes plus gros,
  - arrivent sur le marché des ordinateurs de bureau
- Intéressant pour les serveurs orientés débit et les programmes parallèles
  - Partage des ressources à grain fin
  - Accès uniforme via chargements/rangements (load/store)
  - Transferts automatiques de données et duplication cohérente dans les caches
  - Utile aussi pour les systèmes d'exploitation
- Mécanismes normaux (monoprocesseur) pour accéder aux données (lectures et écritures)
  - Le point clé est l'extension de la hiérarchie mémoire pour plusieurs processeurs.



## Caches et cohérence des caches

- Rôle clé des caches
  - Réduisent le temps d'accès moyen aux données
  - Réduisent les besoins en bande passante sur l'interconnexion partagée.
- Les caches privés des processeurs posent problème
  - Les copies d'une variable peuvent être présentes dans plusieurs caches
  - Une écriture par un processeur peut ne pas être visible aux autres
  - Problème de la cohérence des caches
  - Actions nécessaires pour assurer la visibilité

# Le problème de cohérence des caches




1 P1 lit u
2 P3 lit u
3 P3 écrit dans u
4 P1 lit u
5 P2 lit u

Résultat des lectures 4 et 5
avec écriture simultanée ?
avec la réécriture ?



# Cohérence des caches avec un bus

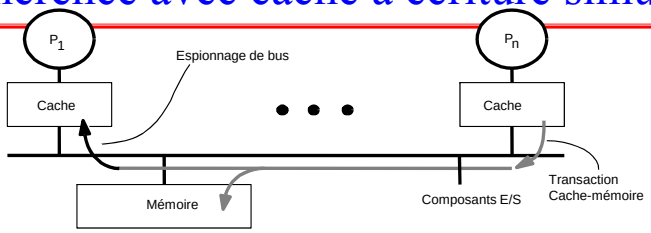
- Construite au dessus de deux fondements des systèmes monoprocesseurs
  - Les transactions de bus
  - Le diagramme de transitions d'états des caches
- La transaction de bus monoprocesseur
  - Trois phases : arbitrage, commande/adresse, transfert de données
  - Tous les composants observent les adresses ; un seul maître du bus
- Les états du cache monoprocesseur
  - Ecriture simultanée sans allocation d'écriture
    - Deux états : valide et invalide
  - Réécriture
    - Trois états : valide, invalide, modifié
- Extension aux multiprocesseurs pour implémenter la cohérence

# La cohérence par espionnage de bus

- *Idée de base*
  - Les transactions bus sont visibles par tous les processeurs.
  - Les processeurs peuvent observer le bus et effectuer les actions nécessaires sur les évènements importants (changer l'état)
- *Implémenter un protocole*
  - Le contrôleur de cache reçoit des entrées de deux côtés :
    - Requêtes venant du processeur, requêtes/réponses bus depuis l'espion
  - Dans chaque cas, effectue les actions nécessaires
    - Mise à jour des états, fourniture des données, génération de nouvelles transactions bus
  - Le protocole est un algorithme distribué : machines d'états coopérantes.
    - Ensemble d'états, diagramme de transitions, actions
  - La granularité de la cohérence est typiquement le bloc de cache



# Cohérence avec cache à écriture simultanée



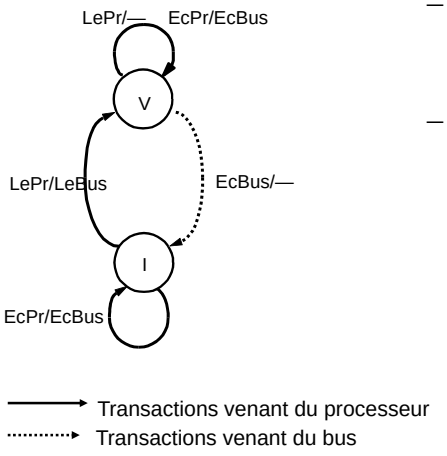

- Extension simple du monoprocesseur : les caches à espionnage avec invalidation ou propagation d'écriture
  - Pas de nouveaux états ou de transactions bus
  - Protocoles à diffusion d'écriture
- Propagation des écritures
  - la mémoire est à jour (écriture simultanée)
- Bande passante élevée nécessaire

# Les actions du cache à écriture simultanée (monoprocesseur)

| *Etat présent* | *Succès lecture* | *Succès écriture* | *Echec lecture* | *Echec écriture* | |
|---|---|---|---|---|---|
| Invalide | ----- | ----- | Valide | Valide | *Etat futur* |
| | ----- | ----- | Echec lecture sur Bus | Echec écriture sur Bus | *Action* |
| Valide | Valide | Valide | Valide | Valide | *Etat futur* |
| | CPU lit le cache | • Ecriture bus<br>• CPU écrit dans le cache | Echec lecture sur Bus | Echec écriture sur Bus | *Action* |



# Le diagramme de transition (écriture simultanée non allouée)




- Deux états par bloc de cache
  - L'état d'un bloc peut être considéré comme un vecteur de p éléments.
- Les bits d'état sont associés aux seuls blocs présents dans le cache
  - Les autres blocs sont considérés invalides (non présents)
  - L'écriture va invalider tous les autres caches
  - Il peut y avoir plusieurs blocs lus simultanément, mais l'écriture les invalide

# Les actions du cache à réécriture (monoprocesseur)

| *Etat présent* | *Succès lecture* | *Succès écriture* | *Echec lecture* | *Echec écriture* | |
|---|---|---|---|---|---|
| Invalide | ----- | ----- | Valide | Modifié | ***Etat futur*** |
| | | | Echec lecture sur Bus | Echec écriture sur bus | ***Action CPU*** |
| Valide | Valide | Modifié | Valide | Modifié | ***Etat futur*** |
| | CPU lit le cache | CPU écrit dans le cache | Echec lecture sur Bus | Echec écriture sur bus | ***Action CPU*** |
| Modifié | Modifié | Modifié | Valide | Modifié | ***Etat futur*** |
| | CPU lit le cache | CPU écrit dans le cache | Bloc éjecté et copié en MP Echec lecture sur Bus | Bloc éjecté et copié en MP Echec écriture sur bus | ***Action CPU*** |



# Le protocole de base MSI : réécriture et invalidation

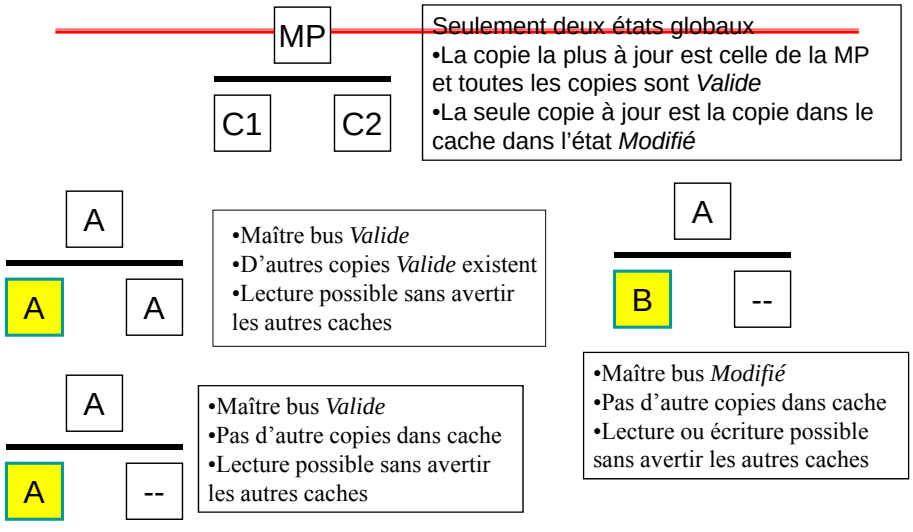
- Etats
  - Invalide (I)
  - Partagé (S): un ou plusieurs
  - Modifié (M): un seul
- Les évènements processeurs :
  - LePr (lecture)
  - EcPr (écriture)
- Les transactions bus
  - LeBus: demande une copie sans vouloir la modifier
  - LeExBus: demande une copie pour la modifier
  - RéBus : Mise à jour de la mémoire
- Actions
  - Modifie l'état, effectue une transaction bus, envoie les données sur le bus

# Protocole à invalidation 3 états

- Modification légère du protocole à réécriture
- Hypothèse
  - Un seul bus et une seule MP
  - Deux CPU ou plus, avec chacun un cache à réécriture (WB)
  - Chaque bloc de cache est dans un état parmi trois : *Invalide, Valide et Modifié*
  - Les copies de bloc en MP n'ont pas d'état
  - A tout instant, un seul cache est maître du bus
  - Le maître du bus ne répond pas aux commandes bus
  - Tous les blocs provoquant des échecs sont fournis
    - Par la MP si toutes les copies du cache sont *Valide*
    - Seulement par la copie du bloc *Modifié* (qui n'est alors plus modifié) et on écrit le bloc en MP au lieu de le lire en MP



# Le protocole intuitivement



# Protocole 3-états : diagramme états d'un bloc de cache (CPU)

| *Etat présent* | *Succès lecture* | *Succès écriture* | *Echec lecture* | *Echec écriture* | |
|---|---|---|---|---|---|
| Invalide | ----- | ----- | Valide | Modifié | *Etat futur* |
| | | CPU écrit dans le cache | Echec lecture sur Bus | Echec écriture sur bus | *Action CPU* |
| Valide | Valide | Modifié | Valide | Modifié | *Etat futur* |
| | CPU lit le cache | Echec écriture sur bus<br>CPU écrit dans le cache | Echec lecture sur Bus | Echec écriture sur bus | *Action CPU* |
| Modifié | Modifié | Modifié | Valide | Modifié | *Etat futur* |
| | CPU lit le cache | CPU écrit dans le cache | Bloc éjecté et copié en MP<br>Echec lecture sur Bus | Bloc éjecté et copié en MP<br>Echec écriture sur bus | *Action CPU* |

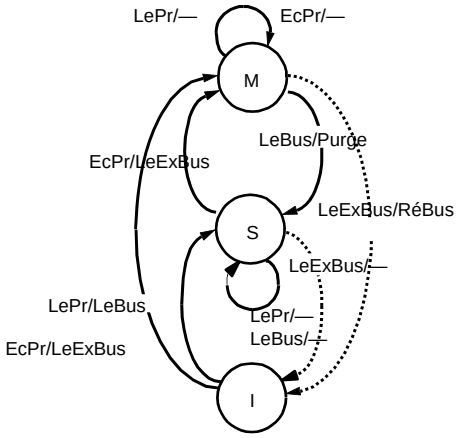


# Protocole 3-états : diagramme états d'un bloc de cache (bus)

| *Etat présent* | *Echec lecture bus* | *Echec écriture bus* | |
|---|---|---|---|
| Invalide | Valide | Modifié | *Etat futur* |
| | rien | rien | *Action bus* |
| Valide | Valide | Modifié | *Etat futur* |
| | rien | rien | *Action bus* |
| Modifié | Valide | Modifié | *Etat futur* |
| | Mettre le bloc sur le bus<br>Interdire à la MP de faire de même | Mettre le bloc sur le bus<br>Interdire à la MP de faire de même | *Action bus* |

## Le protocole MSI à invalidation : Diagramme de transitions

- Ecriture à un bloc modifié
  - A déjà la valeur la plus récente ; peut utiliser la mise à jour (BusUpgr) au lieu de la LeExBus
- Un remplacement change l'état de deux blocs : le bloc remplacé et le bloc arrivant






## Comparaison avec un seul cache à réécriture

- Similarités
  - Succès lecture invisible sur le bus
  - Tous les échecs sont visibles sur le bus
- Différences
  - Avec un seul cache WB, tous les blocs provoquant un échec sont fournis par la MP. Dans le protocole à trois états, les blocs sont fournis soit par la MP ou par le seul bloc de cache contenant la seule copie *Modifié*
  - Avec un seul cache WB, un échec écriture est invisible sur le bus. Dans le protocole à trois états, un succès en écriture sur un bloc *Valide* invalide tous les autres blocs Valide par un Echec écriture Bus (action nécessaire)

## Validation du protocole à trois états

- ***Problème :*** la transition d'état d'un automate est supposée être atomique, mais ce n'est pas le cas dans ce protocole à cause du bus
- Exemple : Echec lecture CPU sur un bloc Modifié
  - L'accès CPU au cache détecte l'échec
  - Requête bus
  - Acquisition du bus, et changement de l'état du bloc de cache
  - Eviction du bloc du cache et copie en MP
  - Positionnement de l'Echec lecture bus sur le bus
  - Réception du bloc demandé depuis la MP ou un autre cache
  - Libération du bus, et lecture à partir du bloc de cache que l'on vient de recevoir
- L'arbitrage du bus peut provoquer un écart entre les étapes 2 et 3
  - Toute la séquence n'est plus atomique.
  - Le protocole fonctionne correctement si les étapes 3 à 7 sont atomiques, c'est à dire si l'on n'a pas un bus à transactions éclatées

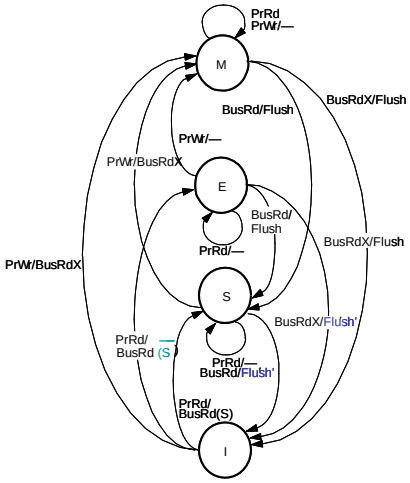


## Le protocole MESI

- Quatre états, à invalidation d'écriture
- Version améliorée du protocole à 3 états
  - L'état Valide est séparé en Exclusif et Modifié
    - Exclusif : seule copie – identique au bloc en MP
    - Partagé : plusieurs copies –identique au bloc en MP
- Différentes versions légèrement différentes du protocole MESI
  - Le protocole MESI du PowerPC 601 ne supporte pas les transferts de bloc de cache à cache.

# Le protocole MESI

- 4 états
  - Exclusif
    - Seule copie du bloc
    - Non modifié (= Mémoire)
  - Modifié
    - Modifié (!= Mémoire)
  - Partagé
    - Copie dans plusieurs blocs
    - Identique en mémoire
  - Invalide
- Actions
  - Processeurs
  - Bus






# Diagramme d'états du protocole MESI (côté CPU)

| *Etat* | *Succès lecture* | *Succès écriture* | *Echec lecture* | *Echec écriture* | *CPU* |
|---|---|---|---|---|---|
| Invalide | ------- | ------- | Partagé | Exclusif | Etat futur |
| | ------- | ------- | Echec lecture Bus | Echec écriture Bus | Action |
| Exclusif | Exclusif | Modifié | Partagé | Exclusif | Etat futur |
| | CPU lit le cache | CPU écrit dans le cache | Echec lecture Bus | Echec écriture Bus | Action |
| Partagé | Partagé | Modifié | Partagé | Exclusif | Etat futur |
| | CPU lit le cache | • Invalidation<br>• CPU écrit dans le cache | Echec lecture Bus | Echec écriture Bus | Action |
| Modifié | Modifié | Modifié | Partagé | Exclusif | Etat futur |
| | CPU lit le cache | CPU écrit dans le cache | • Bloc éjecté et copié en MP<br>• Echec lecture Bus | • Bloc éjecté et copié en MP<br>• Echec écriture Bus | Action |

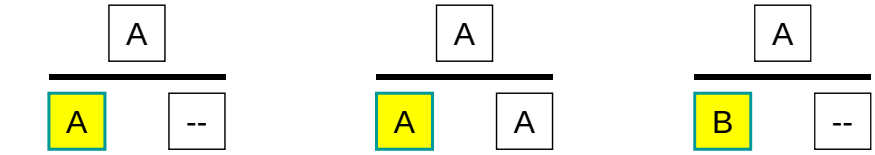
# Diagramme d'états du protocole MESI (côté bus)

| *Etat présent* | *Invalidation* | *Echec lecture Bus* | *Echec écriture Bus* | |
|---|---|---|---|---|
| Invalide | Invalide | Invalide | Invalide | Etat futur |
| | Rien | Rien | Rien | Action bus |
| Exclusif | Invalide | Partagé | Invalide | Etat futur |
| | Rien | Rien | Rien | Action bus |
| Partagé | Invalide | Partagé | Invalide | Etat futur |
| | Rien | Rien | Rien | Action bus |
| Modifié | Invalide | Partagé | Invalide | Etat futur |
| | Rien | Fournir le bloc et mettre à jour la mémoire | Fournir le bloc et mettre à jour la mémoire | Action bus |



# Comparaison avec le protocole à 3 états

- Similarités
  - Succès lecture invisible sur le bus
  - Tous les échecs sont traités de la même manière
- Différences
  - Amélioration importante pour traiter les échecs écriture
    - Succès écriture sur état *Exclusif* invisible sur le bus
    - Succès écriture sur état *Partagé* n'implique pas un transfert de bloc, seulement un signal de contrôle




•Etat *Exclusif*
•Peut être lu ou écrit

•Etat *Partagé*
•Peut être seulement lu

•Etat *modifié*
•Peut être lu ou écrit

## Commentaire sur les protocoles à invalidation d'écriture

- Performance
  - Un processeur peut perdre un bloc de cache à cause de l'invalidation par un autre processeur
  - Le temps d'accès mémoire moyen s'accroît, puisque toutes les écritures dans des blocs partagés prennent plus de temps (les autres copies doivent être invalidées)
- Implémentation
  - Bus et CPU veulent accéder simultanément au même cache
    - Soit pour le même bloc, soit pour des blocs différents, mais conflit dans les deux cas
  - Trois solutions possibles
    - Utiliser une seule table des étiquettes et accepter les aléas structurels
    - Utiliser deux tables des étiquettes séparées pour bus et CPU, mais elles doivent être cohérentes en permanence
    - Utiliser une table des étiquettes multi-port (c'est ce que font les Pentium et le PowerPC 601)



## Le problème du faux partage

- Un processeur écrit dans une partie d'une ligne de cache.
- Un autre processeur écrit dans une autre partie d'une ligne de cache
- Même si chaque processeur ne modifie que sa « partie » de la ligne de cache, toute écriture dans un cache provoque une invalidation de « toute » la ligne de cache dans les autres caches.

# Alternatives aux caches

- Approche des processeurs vectoriels
  - Chargement/rangement de vecteurs dans des registres vectoriels
  - Utilisation de mémoires SRAM multi-bancs
- Approche « mémoire scratch pad » (processeurs embarqués)
  - Zone mémoire contrôlée par logiciel
  - Préchargement par logiciel des données nécessaires
    - Instructions de manipulation de blocs



# "Caches logiciels"

- Utilisation de mémoires locales
- Transferts gérés par logiciel (avec DMA)