計算機システム

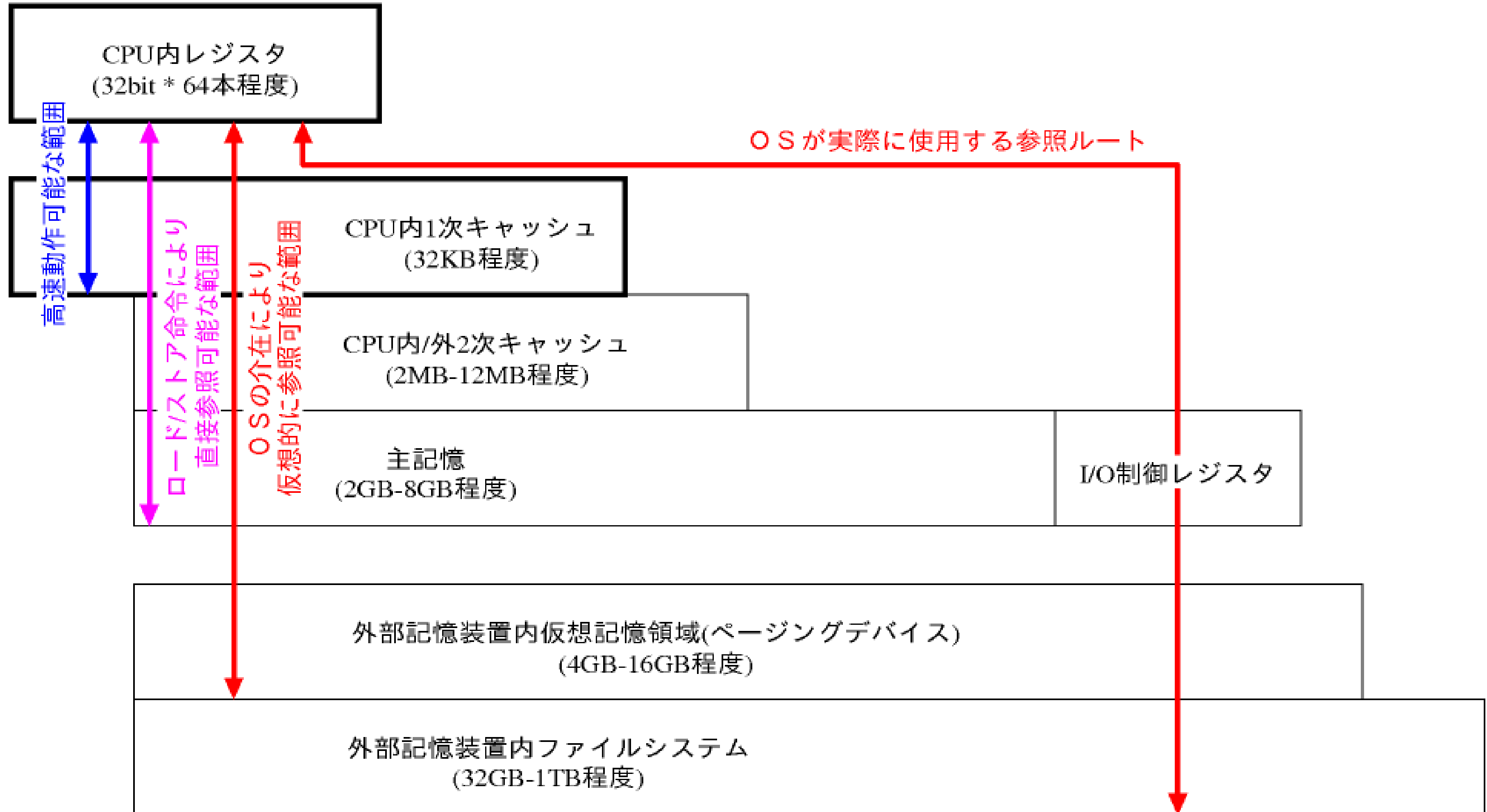
第４回 CA0702:キャッシュメモリとプログラムの実行速度

http://arch.naist.jp/Lectures/ARCH/ca0702/ca0702j.pdf

Copyright © 2021 奈良先端大 中島康彦

記憶階層

キャッシュに当たらなければ，極めて低速



キャッシュは万能ではない

なぜキャッシュが効くのか

- 一般に，プログラムが参照するデータには空間的局所性があると期待
  - ある主記憶アドレスを参照したら，以後，近隣のアドレスを参照する確率が高い
- 同様に，時間的局所性があると期待
  - ある主記憶アドレスを参照したら，暫くは，近隣のアドレスを参照する確率が高い

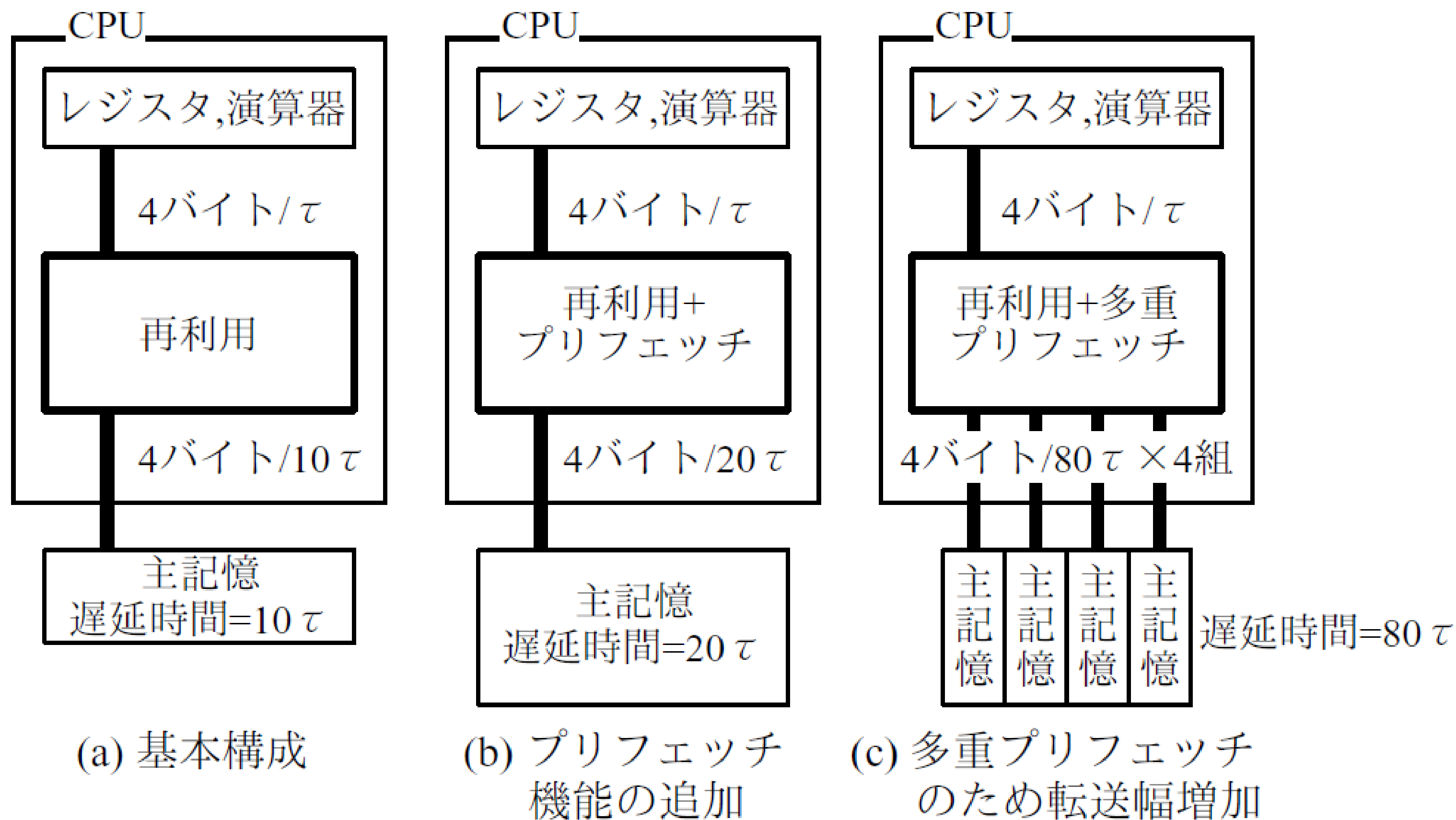
ということは，以下のプログラムにはキャッシュは効かない

- キャッシュ容量よりもワーキングセットが大（ランダム参照を含む）
- アドレスが単調変化（プリロードが効くケースを除く）

アドレスが連続する64byte程度の主記憶内容を256組程度，保存しておく

- キャッシュの大きさと速度は反比例関係
  - 中速中量と高速小量のいずれがよいかはプログラムの挙動次第
- ウェイ数やラインサイズもハードの複雑さ,動作速度,ヒット率に関係
  - どのパラメタが最適であるかはプログラムの挙動次第

遠くなる主記憶とキャッシュの高機能化



NAIST

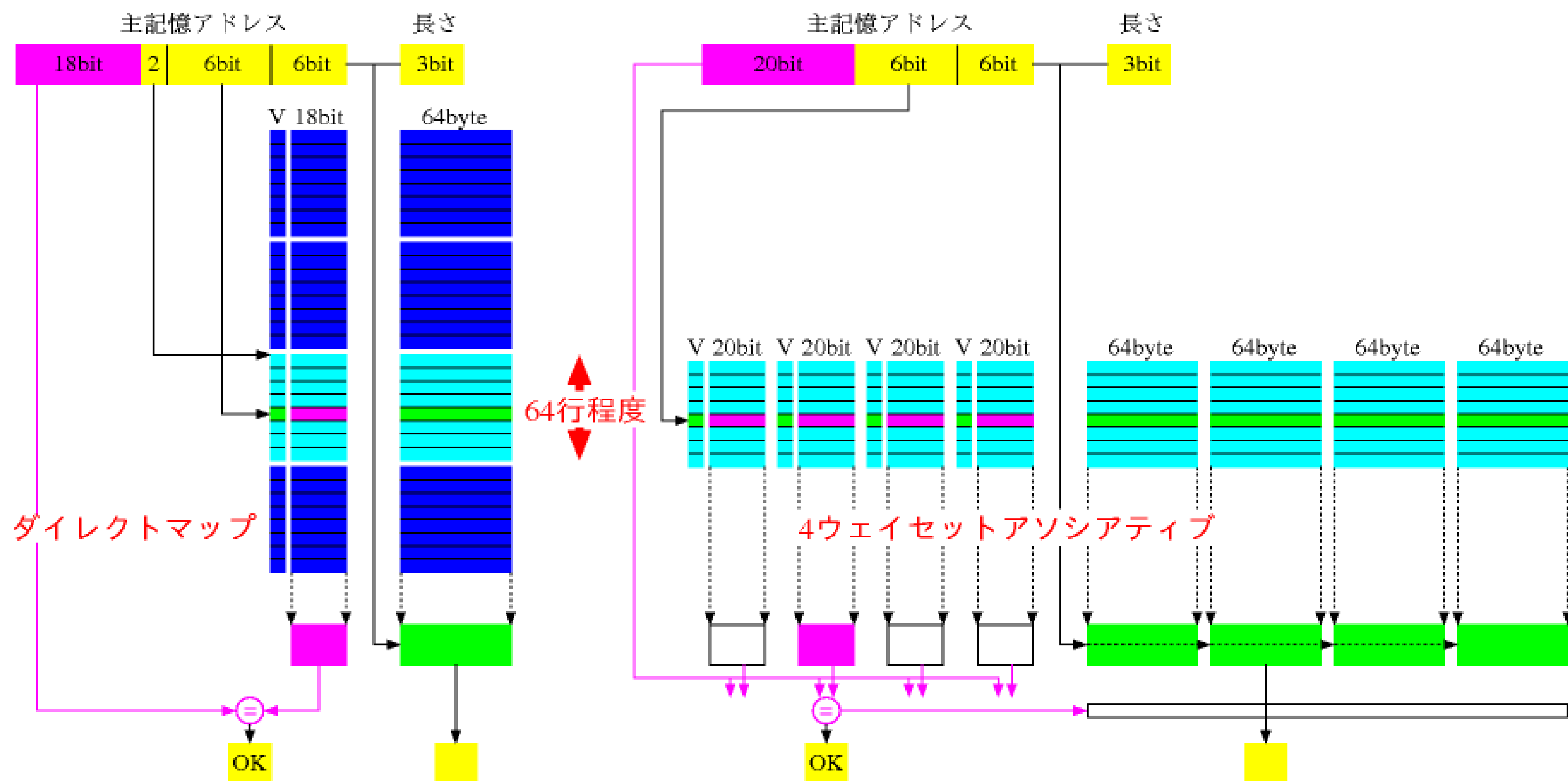
キャッシュの構成

参照アドレスを３つのフィールドに分割

　上位アドレス　　… アドレス長から以下を除いた残り（キャッシュ内タグと比較）
　ライン番号　　　… 全部で256ラインなら，中間の8bit
　ライン内アドレス… ラインサイズが64byteなら，最下位の6bit

ウェイ数とハードの複雑さ/消費電力との関係

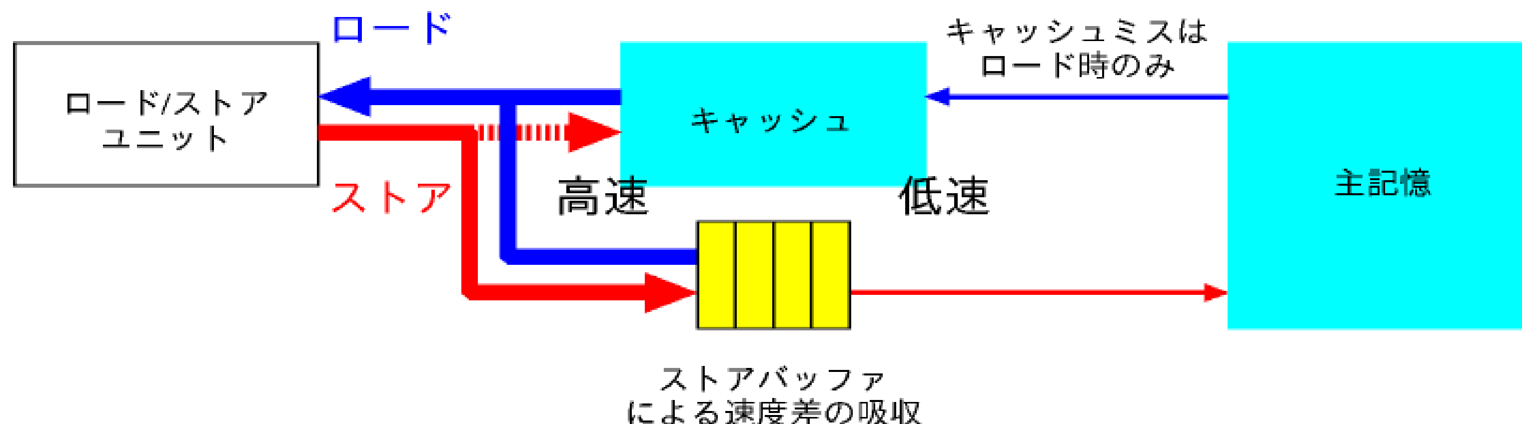
　総量が同じなら，ウェイ数増加がヒット率向上に貢献する．ただし消費電力増加



NAIST

キャッシュの種類(ライトスルー)

ライトスルー方式
- キャッシュと下位階層に対して同時にストアする方法
  - ストアバッファ（未消化ストアを保留するキュー）が必要
  - 下位階層に，バイトやワードなどの単位で書き込むインタフェースが必要
  - キャッシュから下位階層への書き戻しが不要
  - キャッシュミスは，ロード時のみ発生する
  - ロード時にストアバッファも参照する必要がある
- ストア命令がない期間を利用して，下位階層へのストアを消化可能な場合に有効

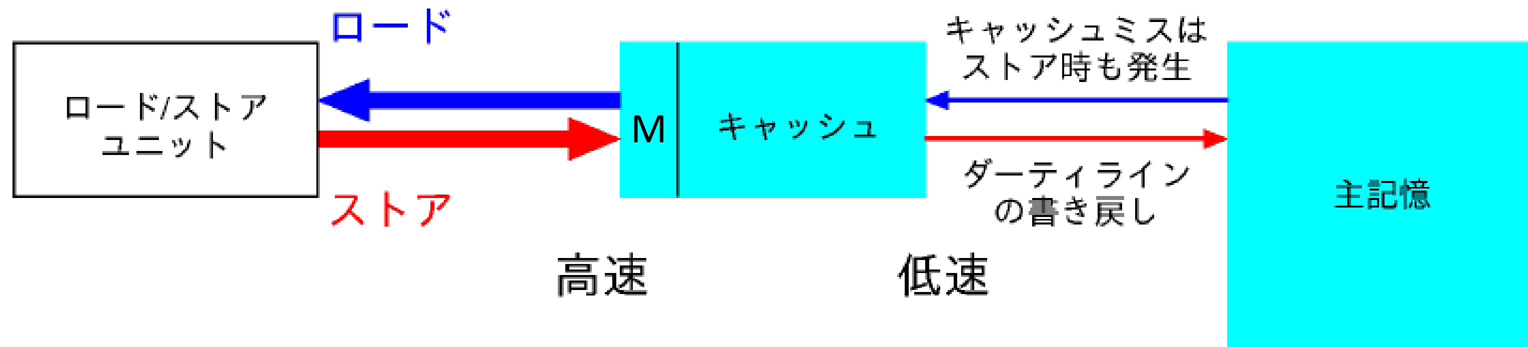


NAIST

キャッシュの種類(ライトバック)

ライトバック方式

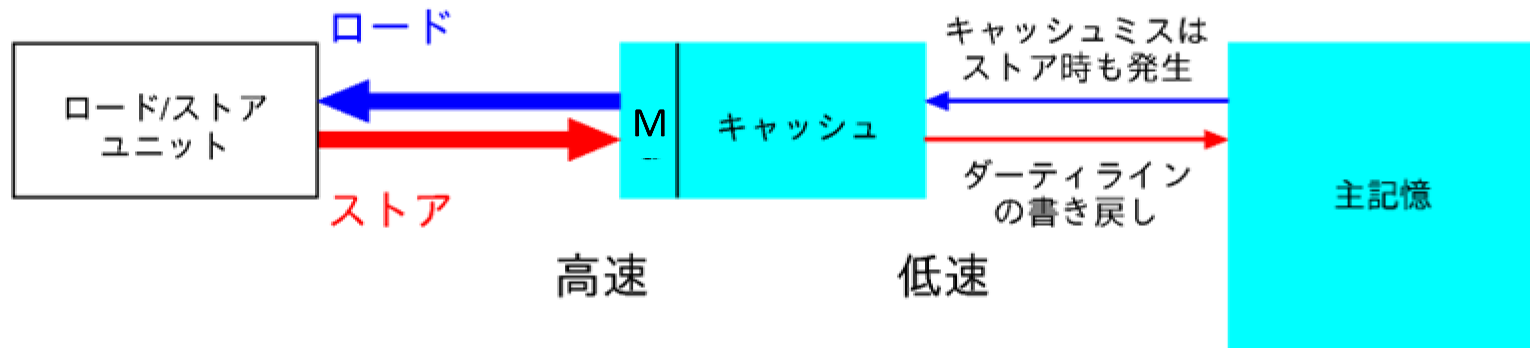
- キャッシュのみにストアし下位階層へはライン入換時に反映する方法
  - ロード/ストアいずれの場合もキャッシュミスにより主記憶から転送
  - 下位階層に，ラインサイズ単位のデータ転送インタフェースがあればよい
  - 各ラインに，下位階層と内容が異なることを示す「ダーティービット」が必要
  - キャッシュラインの入れ換え時，追い出すラインのダーティビットがONならば，下位階層からの読み込みに先立ち，下位階層への書き戻しが必要
- ストア命令の出現頻度が高いか下位階層が極めて遅い場合に有効
  - 最近ではDDRメモリの消費電力も問題（ストアスルーは不利）



NAIST

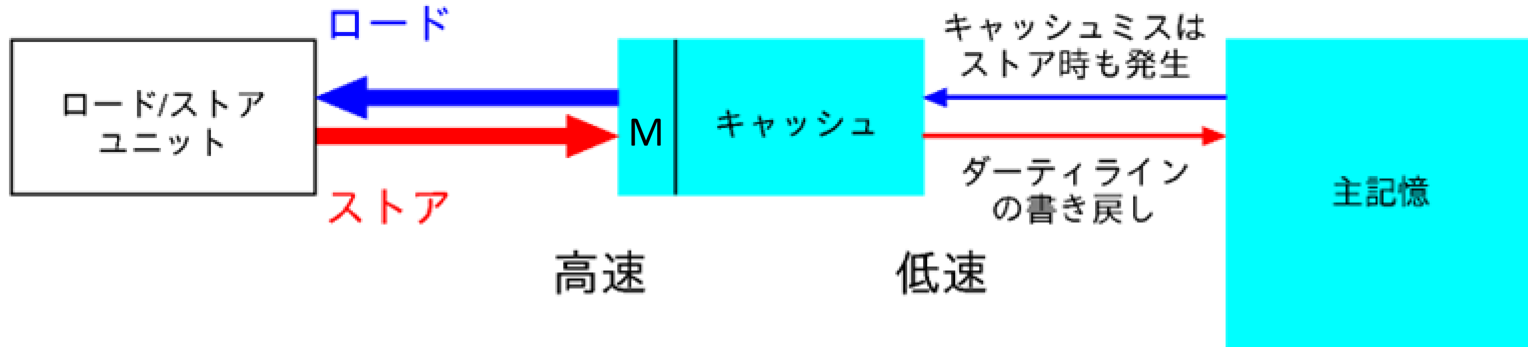
演習問題

●図7.3(b) において，ストアがキャッシュミスした場合のデータの流れを順に示せ（入れ換え対象ラインのMビットが0 と1 の場合を考えること）。



演習問題

●図7.3(b) において，ストアがキャッシュミスした場合のデータの流れを順に示せ（入れ換え対象ラインのMビットが0 と1 の場合を考えること）。



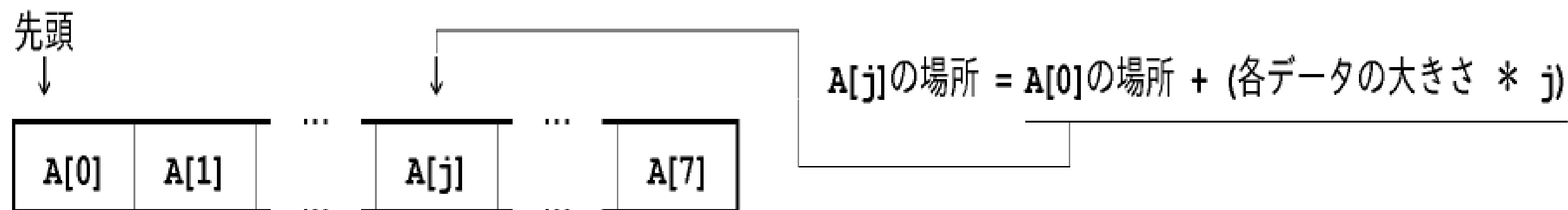
アドレスAがキャッシュミス，当該キャッシュラインにアドレスBの内容が入っている。

- Mが0であれば、当該ラインは主記憶内容と同じであるので破棄する。
  主記憶に対してアドレスAに対応する内容の読み出し要求を発行
  キャッシュに到着後，キャッシュにストアして完了
- Mが1であれは，当該ラインは主記憶に書き戻す必要がある。
  主記憶に対してアドレスBに対応する内容の書き込み要求を発行
  書き込み完了後，引き続いてアドレスAに対する内容の読み出し要求を発行
  キャッシュに到着後，キャッシュにストアして完了
- いずれの場合も，最終的にMは1となる。

プログラミング言語とデータキャッシュ

データ構造とキャッシュの相性（一次元配列）

アーキテクチャ上，特に留意すべき話 … 配列



Cの場合

```
int i;
int A[1000];                    /* 宣言時に総数が決まる */
for (i=0; i<1000; i++) {        /* 使える要素は A[0]~A[999] */
  A[i] = xxxx;
}
```

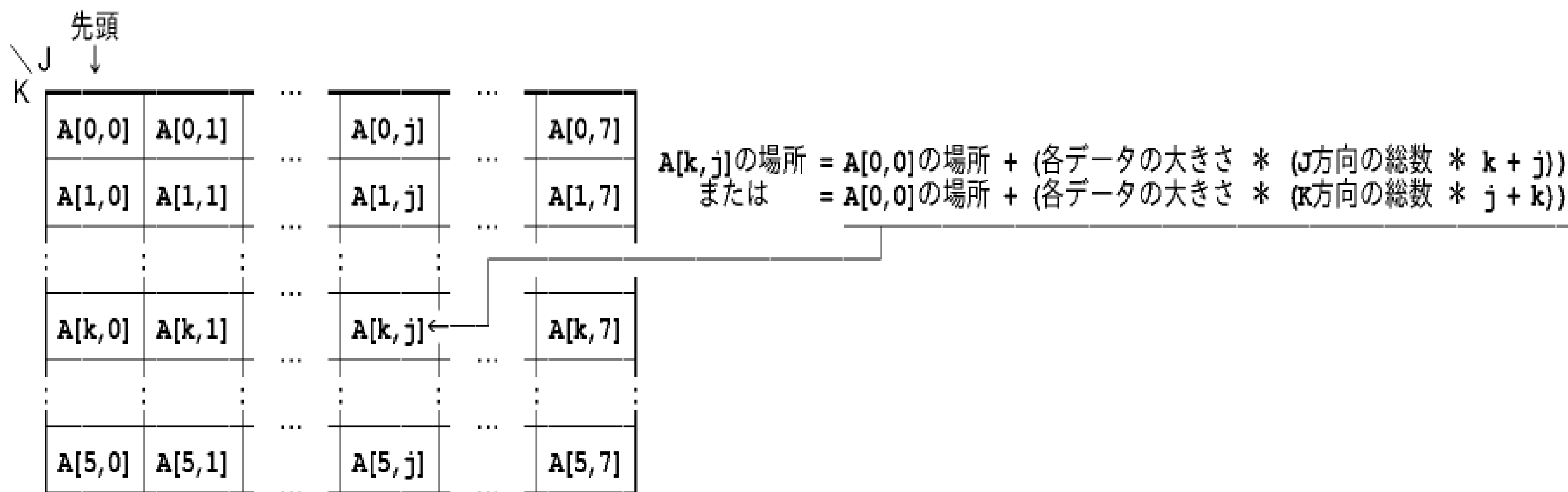
FORTRANの場合

```
INTEGER I
INTEGER A(1000)
DO 10 I=1,1000        … 先頭要素番号が０ではなく１であることに注意
  A(I) = xxxx         … Ａ（Ｉ）の位置 ＝ 先頭位置＋要素サイズ＊（Ｉ－１）
10 CONTINUE
```


NAIST

データ構造とキャッシュの相性（二次元配列）

高速化には，参照順に要素が並ぶよう次元を選ぶ（言語により異なる）



Cの場合

```
for (i=0; i<1000; i++)   /* 使える要素は A[0][0]~A[999][999] */
  for (j=0; j<1000; j++) /* 最右の添字を最内ループで変化させると高速 */
    A[i][j] = xxxx;  /* A[0][0] A[0][1]…A[999][998] A[999][999] */
```

FORTRANの場合

```
DO 10 I=1,1000
  DO 10 J=1,1000    … 最左の添字を最内ループで変化させると高速
    A(J,I) = xxxx   … A(1,1) A(2,1)…A(999,1000) A(1000,1000)
10 CONTINUE
```


NAIST

単調参照によるキャッシュ容量の推定

```
double d0[ 512][1024/8]; /* 0.5MB */
double d1[1024][1024/8]; /* 1.0MB */
double d2[2048][1024/8]; /* 2.0MB */
double d3[4096][1024/8]; /* 4.0MB */
main() { int i, j;
    restme();
    for (i=0; i<4096*N; i++)
        for (j=0; j<512; j++) d0[j][0] += 1.0;
    printf(" .5M:%d", gettme());

    restme();
    for (i=0; i<2048*N; i++)
        for (j=0; j<1024; j++) d1[j][0] += 1.0;
    printf(" 1M:%d", gettme());

    restme();
    for (i=0; i<1024*N; i++)
    for (j=0; j<2048; j++) d2[j][0] += 1.0;
    printf(" 2M:%d", gettme());

    restme();
    for (i=0; i<512*N; i++)
    for (j=0; j<4096; j++) d3[j][0] += 1.0;
    printf(" 4M:%d\n", gettme());
}
```

参照範囲の広さとプログラム実行時間

CPU	周波数-L2容量	測定結果（秒）
SPARC	650M-0.5M	.5M:67 1M:616 2M:733 4M:742
Pentium-M	1.5G-1.0M	.5M:29 1M:34 2M:265 4M:301
XEON	2.8G-0.5M	.5M:25 1M:312 2M:305 4M:311 ヒット率が高いプログラム:26sec ヒット率が低いプログラム:70sec
Pentium-4	3.2G-2.0M	.5M:22 1M:23 2M:25 4M:233 ヒット率が高いプログラム:25sec ヒット率が低いプログラム:51sec
Pentium-4	3.4G-1.0M	.5M:18 1M:22 2M:122 4M:121 ヒット率が高いプログラム:24sec ヒット率が低いプログラム:48sec
XEON	3.6G-1.0M	.5M:19 1M:19 2M:207 4M:205 ヒット率が高いプログラム:22sec ヒット率が低いプログラム:45sec

NAIST

今日はここまで