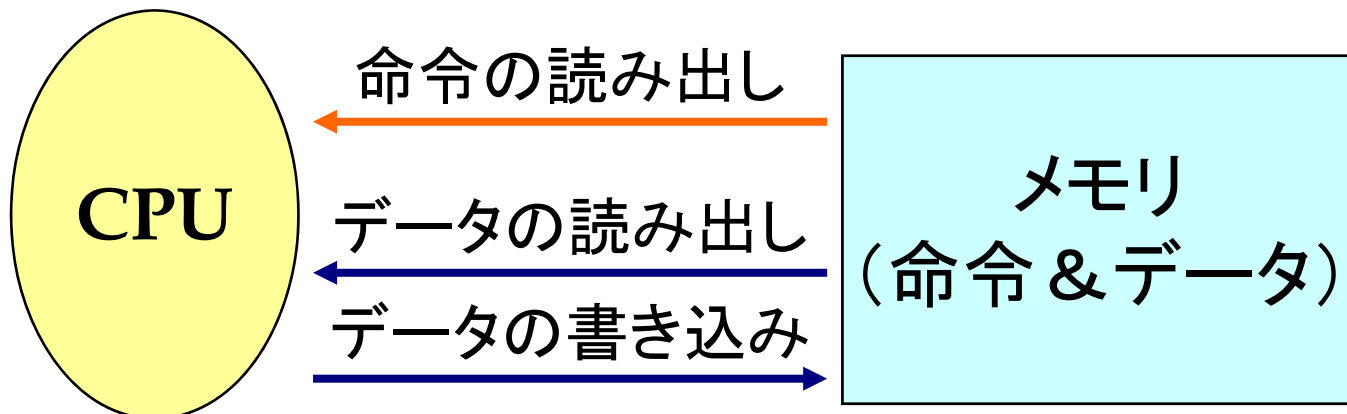


本日の話題: メモリ

- CPU とメモリ
- アドレス空間
- メモリアクセスに要する時間の測定
- キャッシュメモリ

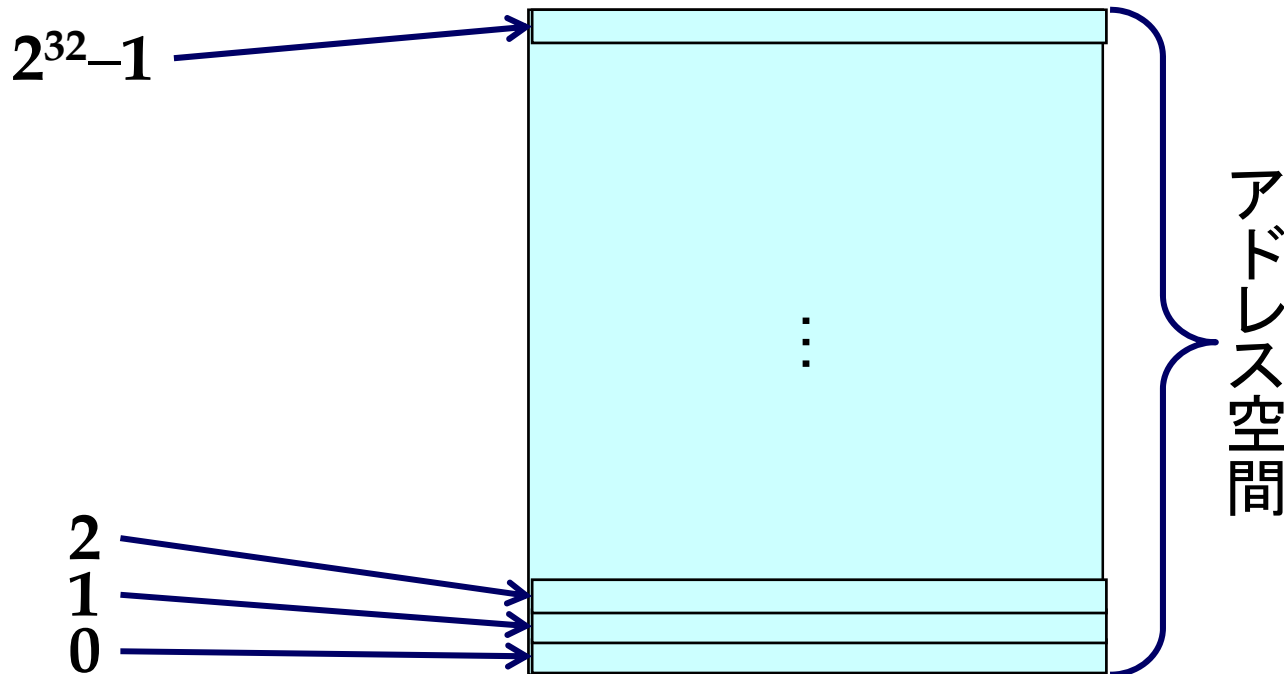
CPU とメモリ

- コンピュータが計算を行うために, CPU (Central Processing Unit) とメモリは必須の部品
 - 四則演算などの計算を行うのは CPU
 - 計算対象のデータや計算内容を指定する命令はメモリに格納



プログラムから見たメモリ (1/2)

- MacOS X の32bitアプリケーションは, 4GBの論理アドレス空間(メモリ空間)を持つ
 - メモリの先頭から, バイト単位で, $0 \sim 2^{32}-1$ の番号(アドレス)がつけられる



プログラムから見たメモリ (2/2)

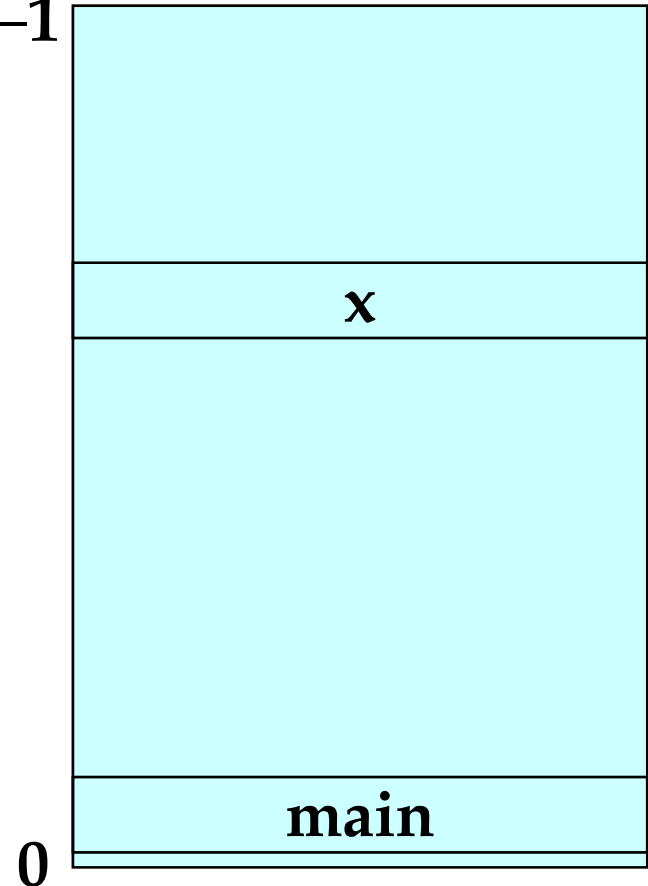
- 変数などのアドレスを C プログラムで扱うには `&` を使うとよい

```
#include <stdio.h>
```

```
main() {  
    int x;  
    printf("main at %x¥n", &main);  
    printf("x at %x¥n", &x);  
    return 0;  
}
```

```
main at 2abc  
x at bffff798
```

$2^{32}-1$



16進数

- アドレスは16進数で表記する場合が多い
 - 0~9, A~F の16種類の数字(digit)を用いる
- 16進1桁が2進4桁(4ビット)に相当
 - $16 = 2^4$
 - 32ビットの値なら16進8桁で表現できる

$$2abc = 2 \times 16^3 + a \times 16^2 + b \times 16^1 + c \times 16^0$$

A	10
B	11
C	12
D	13
E	14
F	15

配列のレイアウト

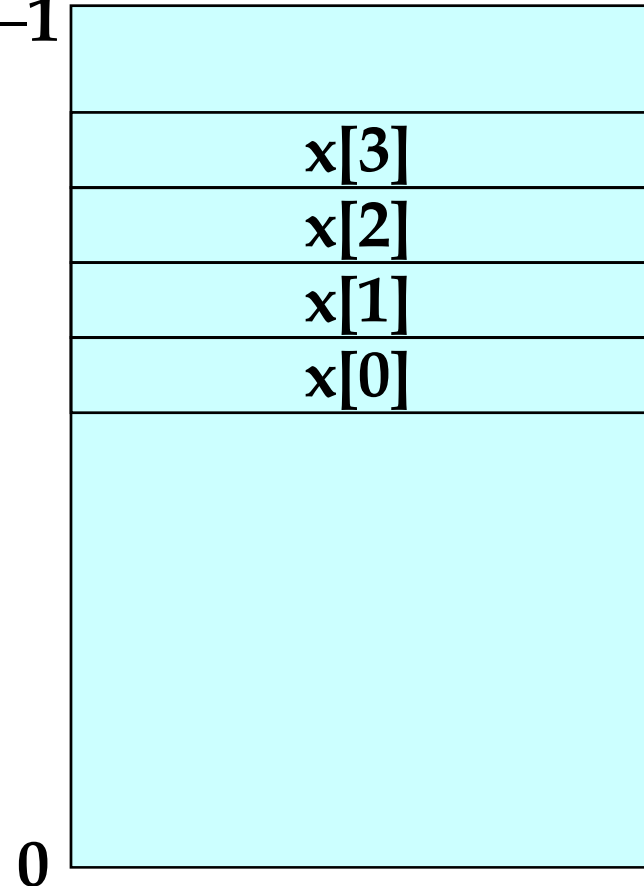
- C 言語の配列の各要素は、メモリ上の連続領域に割り当てられる

```
#include <stdio.h>
```

```
main() {  
    int x[4];  
    printf("x[0] at %x¥n", &x[0]);  
    printf("x[1] at %x¥n", &x[1]);  
    printf("x[2] at %x¥n", &x[2]);  
    printf("x[3] at %x¥n", &x[3]);  
    return 0;  
}
```

```
x[0] at bffff798  
x[1] at bffff79c  
x[2] at bffff7a0  
x[3] at bffff7a4
```

$2^{32}-1$



メモリアクセス時間を計る実験 (1/3)

- 探索アルゴリズムの理論的評価では、メモリアクセスに要する時間を一定と仮定した
 - i.e. メモリ上の場所によらず、同じサイズのデータは一定時間で読み書きできる
- この仮説を確かめるには、実際に計るしかない
 - 1回のメモリアクセス時間の正確な計測は困難
 - Mac OS のclock 関数で計れるのは10ms (0.01秒) 単位
 - mach_absolute_time 関数を使うと、約 30ns (= 3×10^{-8} 秒)単位で計れる
 - 1000回程度のメモリアクセスの合計時間ならほぼ正確に計測できる可能性が高い

メモリアクセス時間を計る実験 (2/3)

■ 実験 I

- 配列から連続する1024個の整数(4KB)を読み出す
- 読み出し開始位置を変えてみる

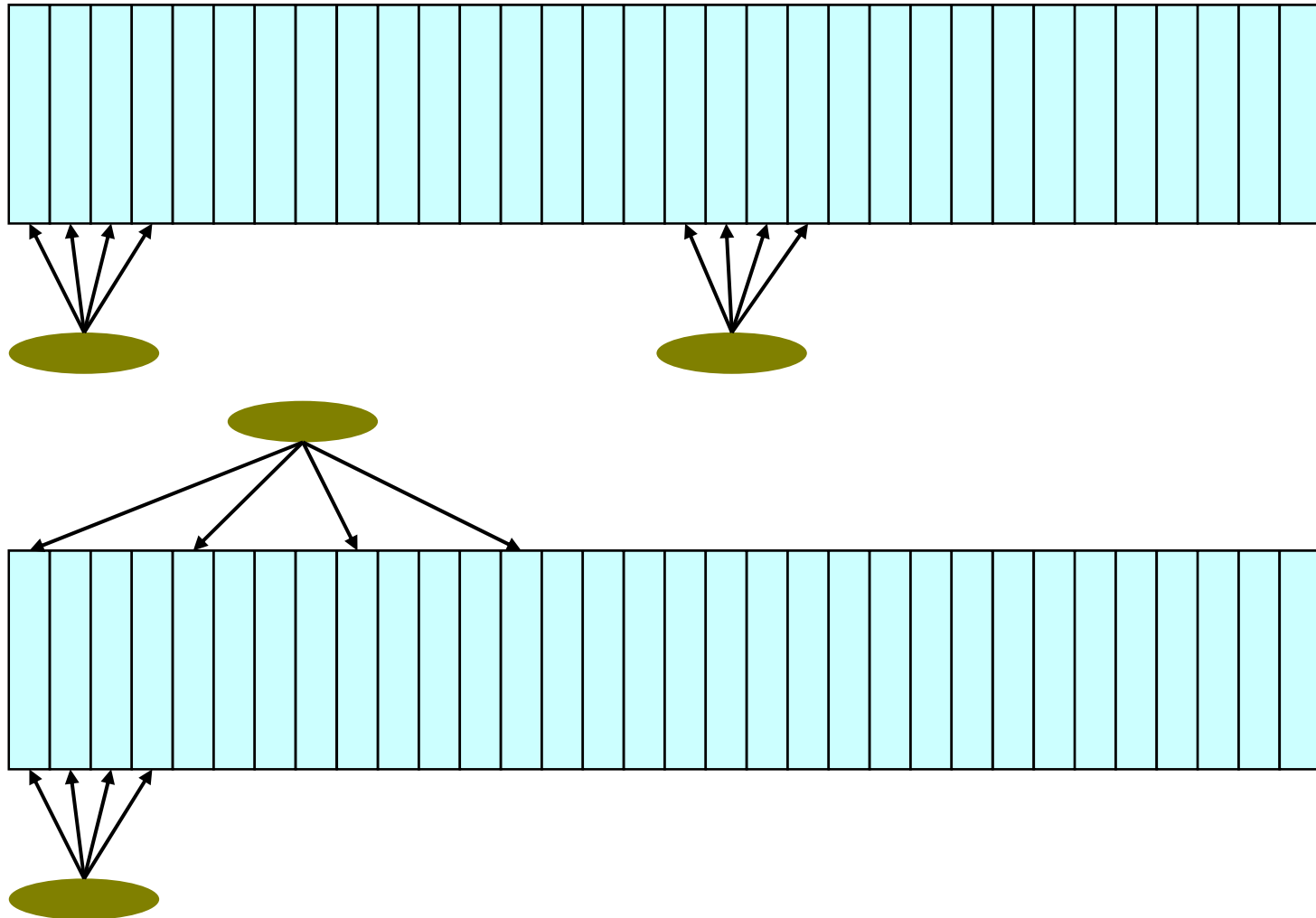
■ 実験 II

- 配列から一定間隔で1024個の整数(4KB)を読み出す
- 読み出し開始位置は固定し, 間隔を変えてみる

■ 実験に使ったコンピュータ

- CPU: 2.5GHz PowerPC 970FX, OS: MacOS 10.4, コンパイラ: gcc 4.0.1 (-fast で最適化)

メモリアクセス時間を計る実験 (3/3)



オフセットの影響 (1/4)

■ 実験条件

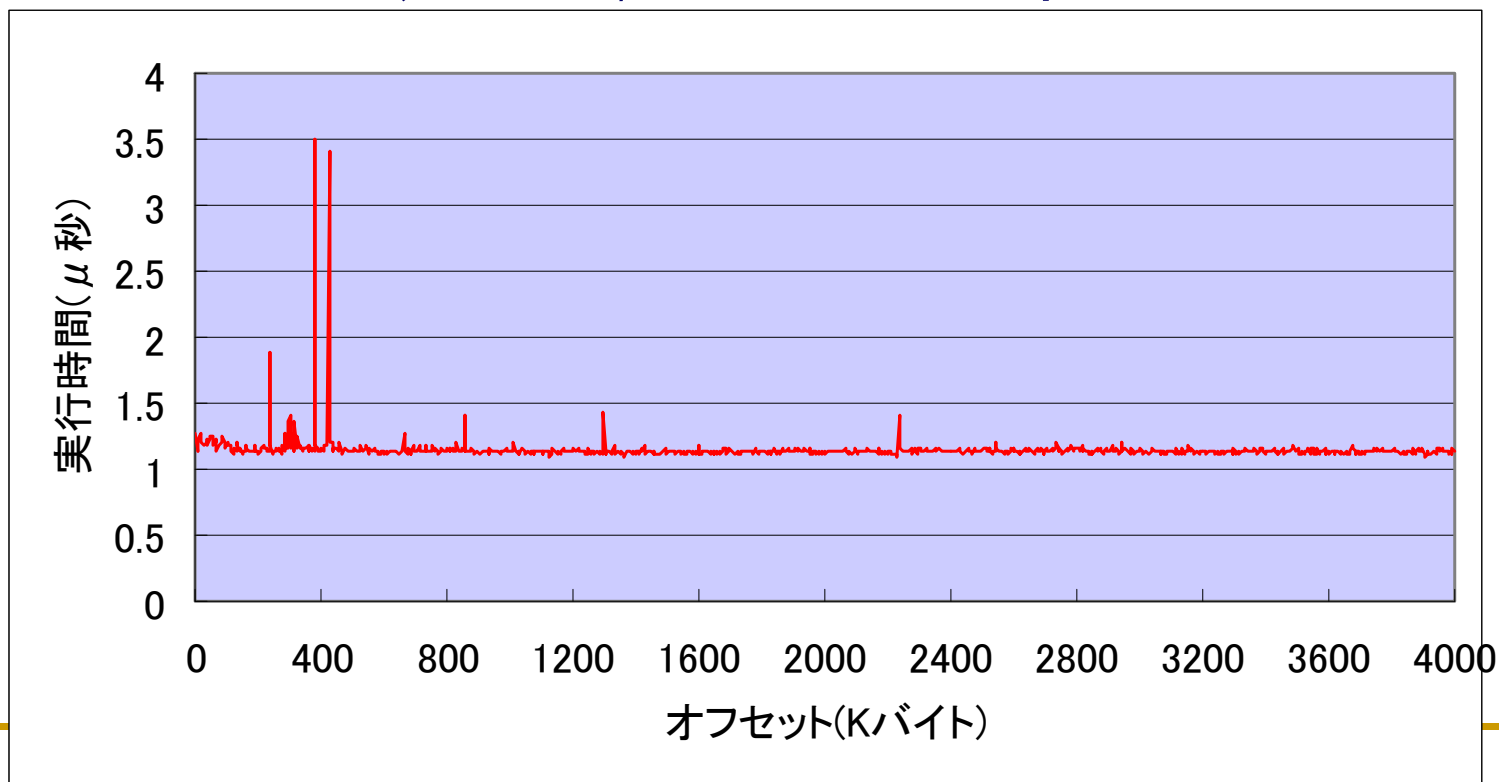
- 以下の for ループで `offset` の値をいろいろ変えて実行時間を計る
 - `SIZE` は十分に大きくとる (i.e. $\text{offset} + 1024 \leq \text{SIZE}$)
 - なお, `x += y` は `x = x + y` と同じ意味

```
int a[SIZE], i;  
int sum = 0;  
for(i = 0; i < 1024; i++) {  
    sum += a[offset + i];  
}
```

オフセットの影響 (2/4)

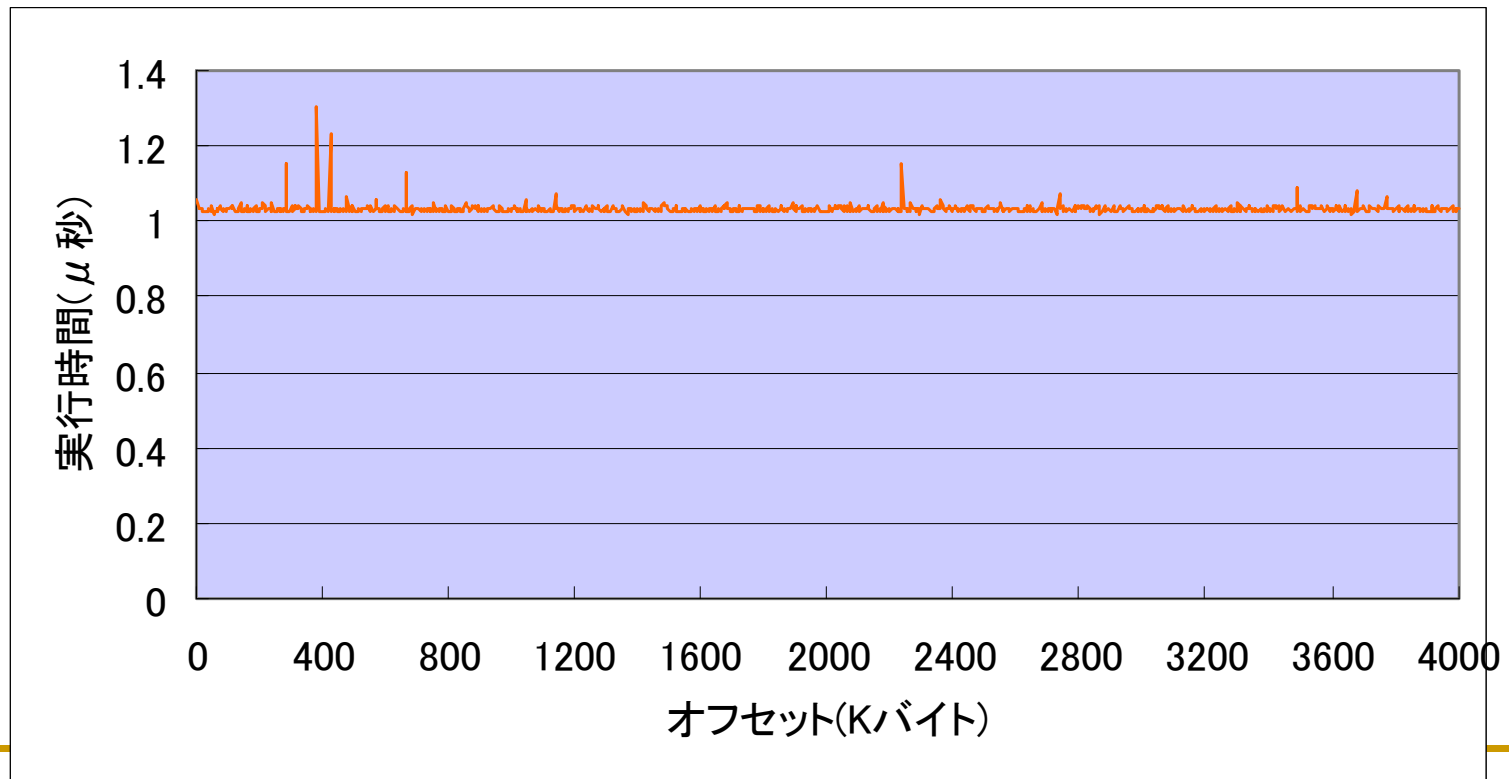
■ 実験結果

- 各オフセットから1024個の整数へのアクセスを10回連続して繰り返し、その10回の平均値を求める
- だいたい一定だが、ところどころ怪しい



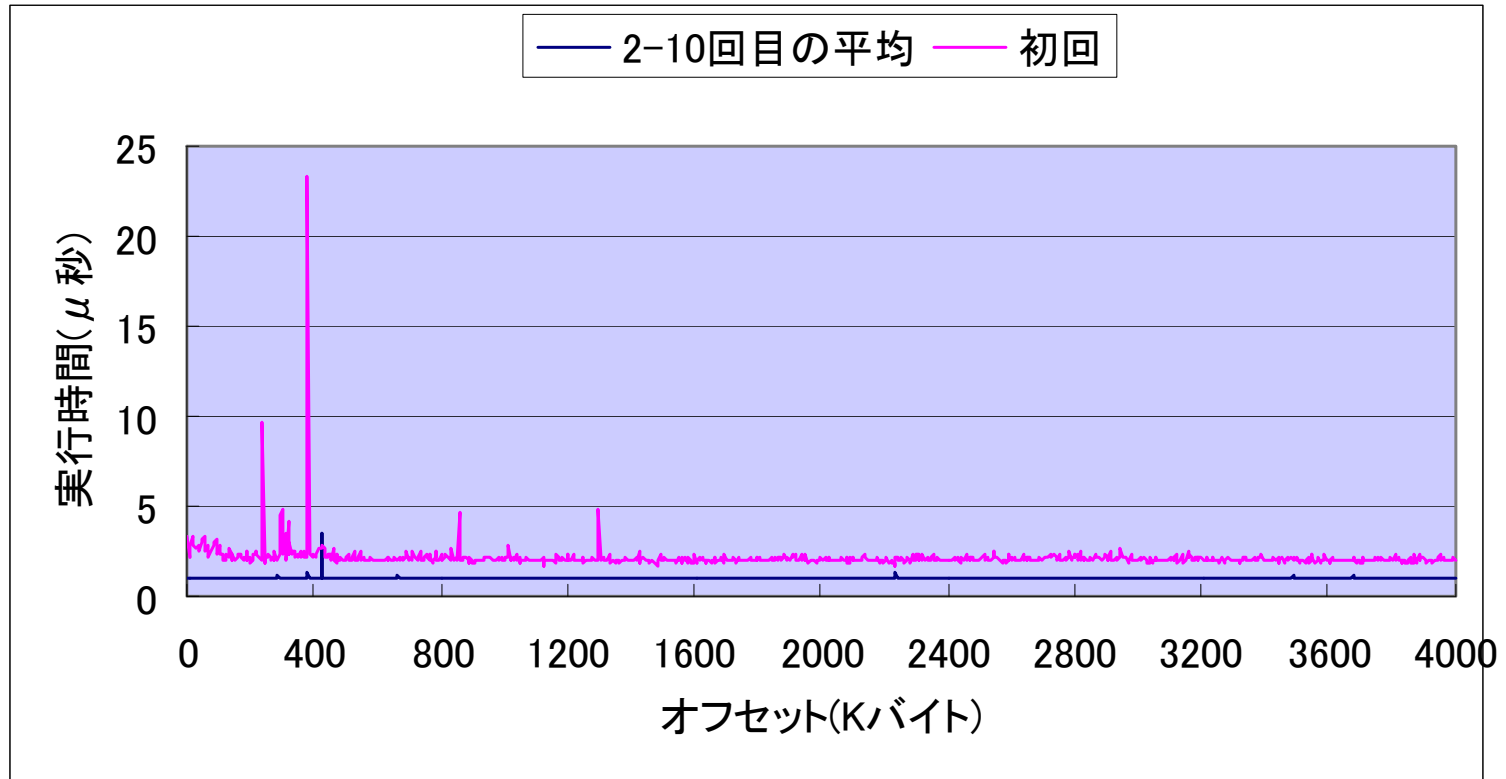
オフセットの影響 (3/4)

- 実行結果 (cont.)
 - 10回中最悪の1回を除く9回の平均なら, より安定
 - For 文の繰り返し1回あたり約1ナノ秒に相当



オフセットの影響 (4/4)

- 実験結果 (cont.)
 - 1回目は, 2回目以降より有意に遅い



ストライドの影響 (1/5)

■ 実験条件

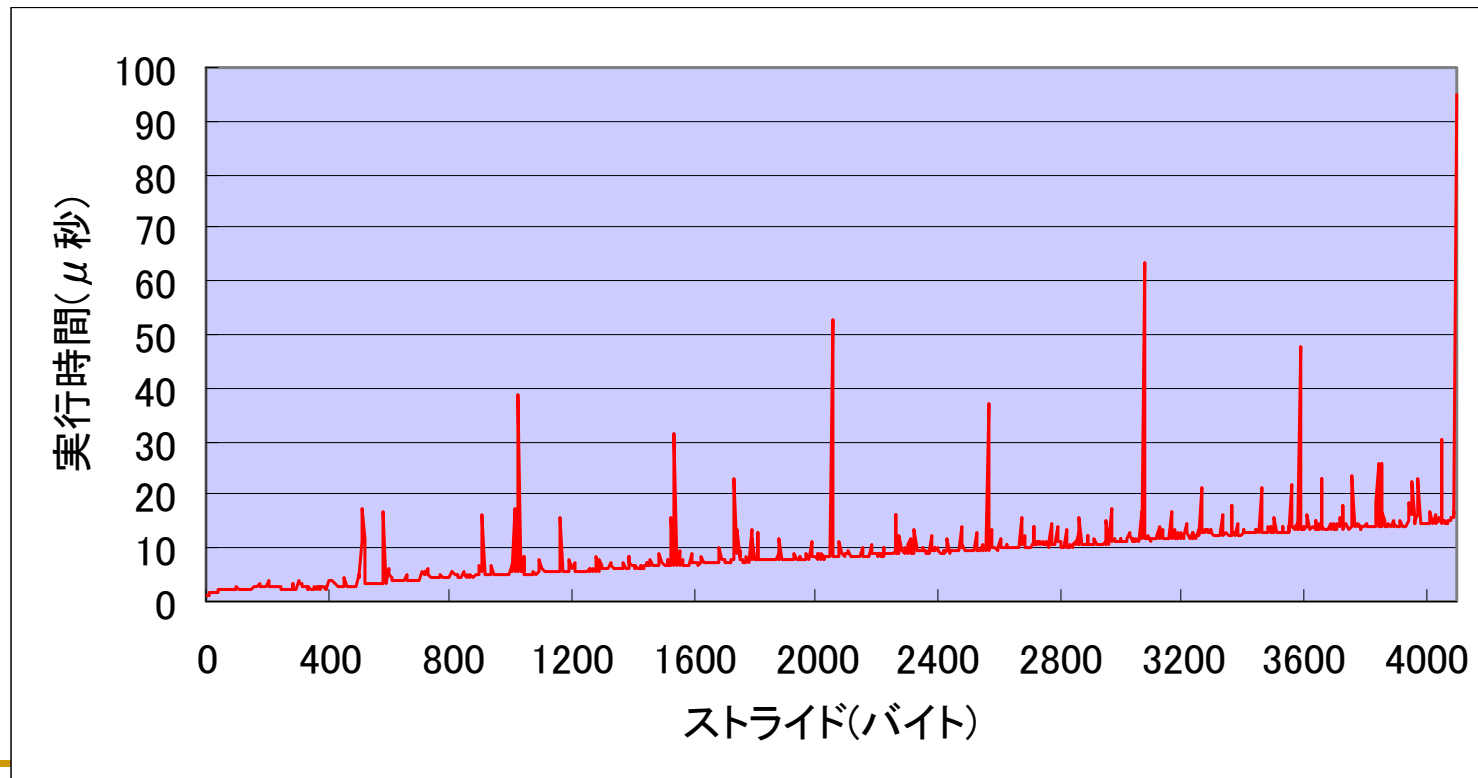
- stride を変えて, 以下の for ループの実行時間を計る
 - SIZE は十分に大きくとる (i.e. $\text{stride} * 1024 \leq \text{SIZE}$)
 - stride が 1 なら, 実験 I と大差ないはず

```
int a[SIZE], i = 0;
int sum = 0;
for(j = 0; i < 1024; i++, j += stride) {
    sum += a[j];
}
```

ストライドの影響 (2/5)

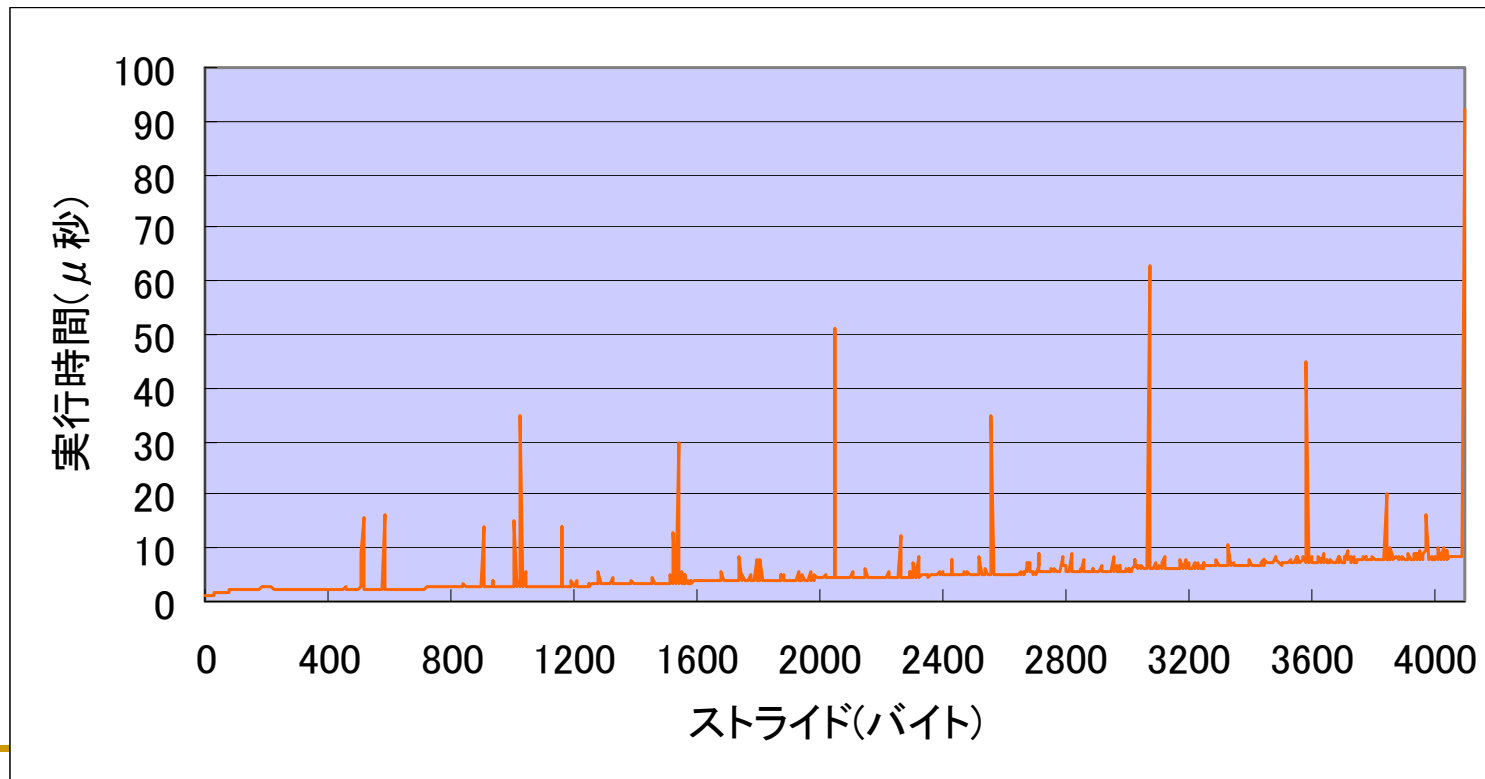
■ 実験結果

- 各ストライドに対し，連続した10回の平均を求める
- ストライドが大きくなると遅くなり，ばらつきも大きい



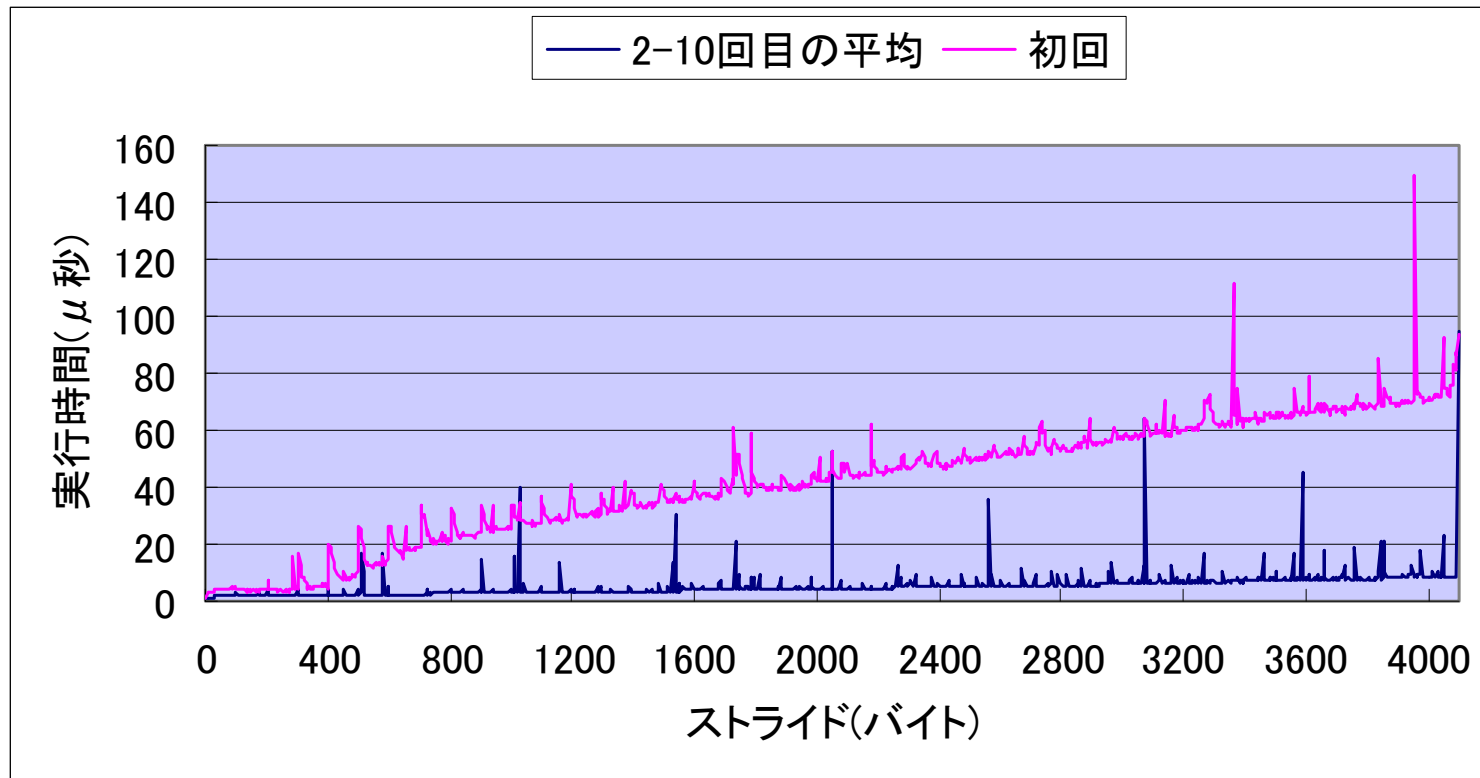
ストライドの影響 (3/5)

- 実験結果 (cont.)
 - 10回中の最良値を並べても傾向は変わらない
 - ただし、傾きは緩やかになる



ストライドの影響 (4/5)

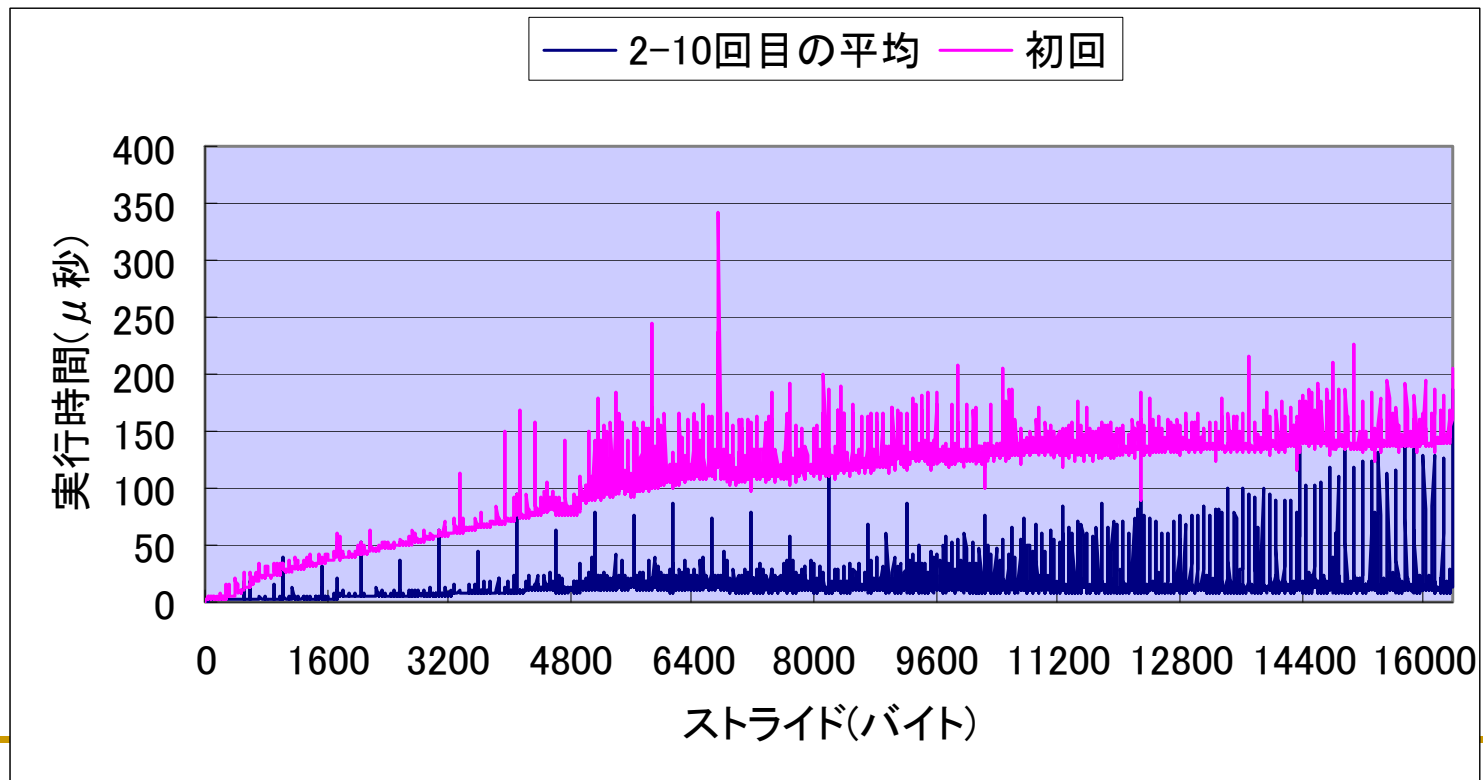
- 実験結果 (cont.)
 - 1回目と2回目以降の差はかなり大きい



ストライドの影響 (5/5)

■ 実験結果 (cont.)

- ストライドをさらに増やすと, 実行時間の増加傾向は鈍るが, ばらつきはやはり大きい

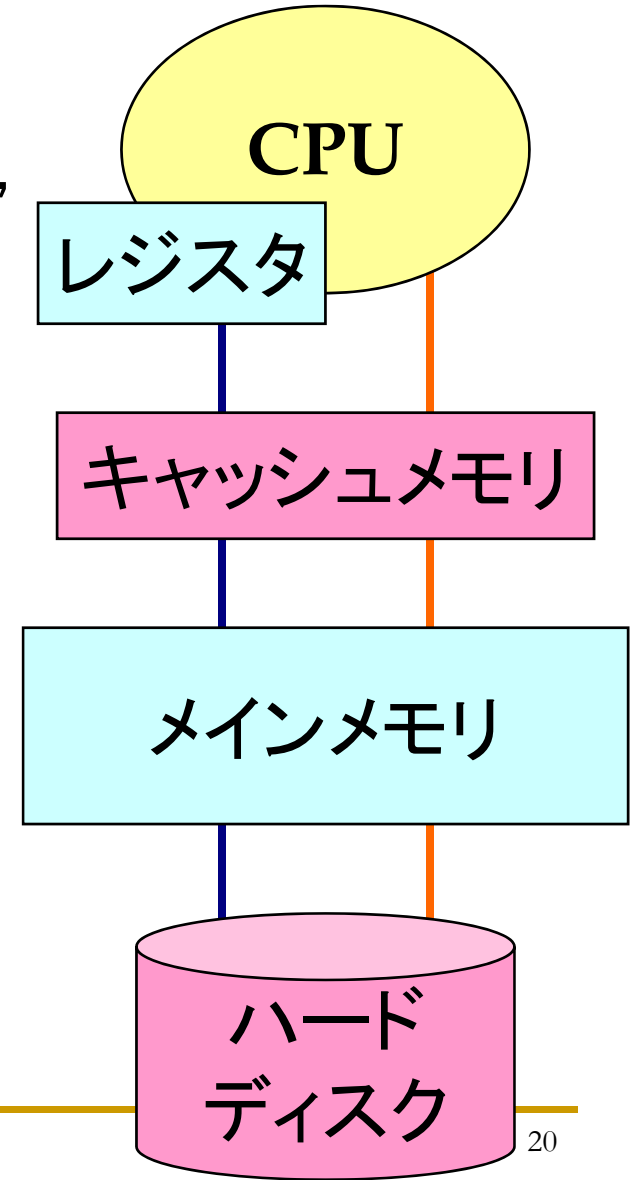


実験結果のまとめ

- メモリから読み出す時間に関する理論的仮定は、かなり怪しい
 - 連続アドレスからの読み出し時間は、場所によらずほぼ一定
 - 同じアドレスからの最初の読み出しは、2回目以降より遅い傾向
 - 飛び飛びのアドレスから読み出すと、連続アドレスから読み出すより、かなり遅くなる
 - 特定のストライドでアクセスすると異常に遅くなることもある
 - 書き込み時間についてこの結果からは何も言えない

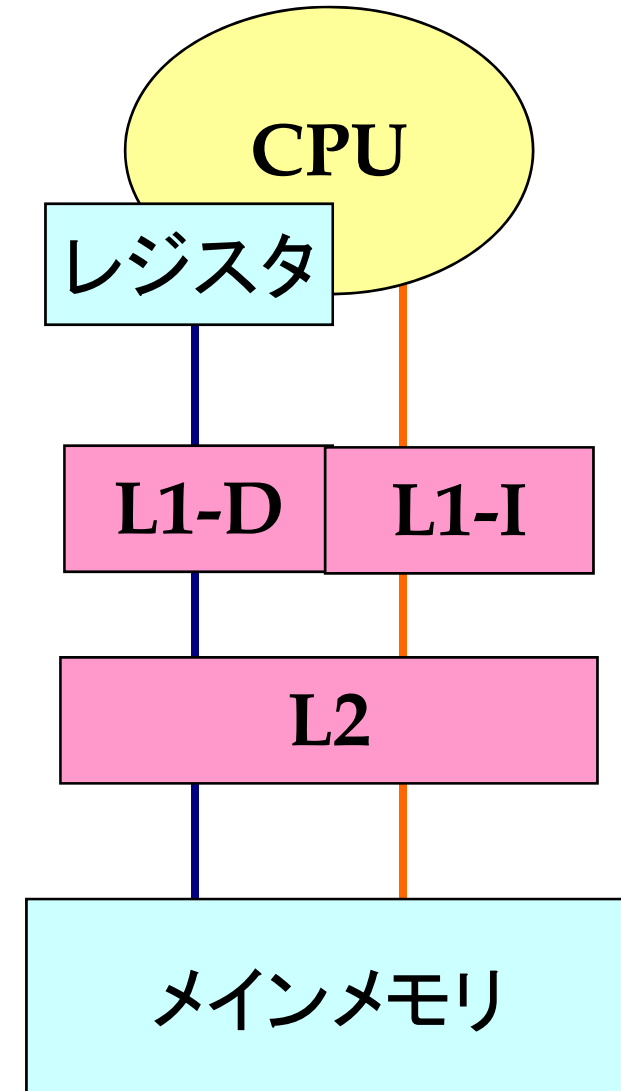
記憶階層

- 速くて、大きくて、安くて、消費電力の少ないメモリが理想だが、現実には難しい
 - 速いメモリは CPU の近くにおく必要があり、近くには少ししか置けない
 - 速いメモリは値段が高い
- 速くて小さいメモリと遅くて大きいメモリを適切に組み合わせる
 - 高い確率で速いメモリを使えば、平均的な性能は高く、コストは低くなる



キャッシュメモリ (1/2)

- ハードウェアで自動制御
 - (若干の例外はあるが)プログラムからは不可視
 - CPU が最近読み込んだデータや命令のコピーを自動的に格納
- PowerPC 970FX のキャッシュメモリは2レベル(L1 & L2)
 - L1キャッシュは, データ用32KBと命令用64KB
 - L2キャッシュは, データと命令で512KBを共有



キャッシュメモリ (2/2)

- 実験 I , II でメモリから読み込んだデータは4KB
 - L1データキャッシュのサイズより小さい
 - ➔ 初回と2回目以降で差があることを(定性的には)説明できる
- PowerPC 970FX のキャッシュメモリは, 128バイトのキャッシュライン単位で管理される
 - 読み込みは(アドレス空間内で連続した)128バイトを単位として行う
 - 飛び飛びのアドレスから読み込む実験では, 使わないデータもキャッシュに読み込まれる
 - ➔ ストライドが大きいと不利になることを(定性的には)説明できる

連絡事項

- 次回 12/13 は演習棟に集合してください