アルゴリズムとプログラミング実践講座

http://akashi.ci.i.u-tokyo.ac.jp/mary/lectures/algorithm/

火曜　13:00 -- 14:30
I-REF 棟 ２階　会議室ab

稲葉真理　with 浅井大史・手塚宏史

この実践講座の背景

- 今年が初年度
- なぜ始めようと思ったか？
  - 研究室でも学生のバックグラウンドがまちまち
    - 研究室には、法学部・農学部出身者
  - アルゴリズムの基礎知識はあったほうが良い
    - 道具としてのアルゴリズム。 → 動作理解は楽しい
      - しめじソートとじゃがいもソート
    - 食わず嫌いの原因の一つは「証明」 → これは省く
  - 計算機実験で知っていてほしい基礎知識がある。
    - 実験の再現性（と効率）→ 自動化（手打ちは止めましょう）

この実践講座について

- ターゲット
  - 学部で情報系の授業を履修しなかった人
  - 学部で受けた情報系の授業の復習をしたい人

  かつ、アルゴリズムを使う（ ≒ プログラムを書く）予定がある人

- 目標
  - プログラムを書くとき、適材適所のツール（アルゴリズムを含む）が使えるように意識できること。

アルゴリズム（実践講座）について

- 適材適所
  - プログラムを書くとき、アルゴリズムとデータ構造を状況に応じて適切に選択できること
    - アルゴリズムを、その性質を含めて知っていること
      - ドラエモンのポケット、メアリーポピンズの絨毯バッグ
    - スケールするかどうかの判断ができること
      - テストプログラムと規模が変わったときに動くのか？
      - 解こうとしている問題の性質にあっているか？
- お金を稼いでる　or 世の中を変えたアルゴリズム
  - ある学生さんのリクエスト（社会に出る前に知っていたい♪）

プログラミング実践について

背景： 修士課程では演習の授業はあまりない。

- クラウド実践工房 （おすすめ!）
  → enPit プロジェクト、クラウドを作って使う実習
  浅井さんを中心に、グループワークの演習等

- プログラミング外伝
  → 真理研インターナル。手塚さんを中心に

アルゴリズムの授業やるときに、プログラミング演習もいれてみようかな。
→ 浅井さん、手塚さんに、お手伝いを依頼

おおまかな予定

- 概論（計算機システムと計算量など）
- データ構造（とアルゴリズム）の基礎
- グラフ理論と巨大グラフ
- モデル化・線形計画法・整数計画法
- データマイニングとクラスタリング
- 解探索とメタヒューリスティクス
- Library の使い方
- 「Can computers think?」 David Avis教授(7/8)

やりたかったけど、やらない事

- アルゴリズムに関する厳密な話
  - アルゴリズムデザインも若干範囲外。文献は紹介する

- データベース、SQL（卒業生のリクエストは多い）
- 統計（データをいかに読むか、大事）
- 数値計算と誤差
- 並列化
- ~~研究っぽい話~~ → 最後7/8（火）

  “ Can computers think?” Prof. David Avis@京大＆McGill

レポート・成績について

- ほぼ毎回、プログラミング課題を出題する予定
  - 効率の良い計算機実験のためのツールを使ってみる
  - アルゴリズムの実装
  - ライブラリの利用・・・　など

- ３回以上、レポートとプログラムのファイルの組をメールで提出のこと
  - E-mail: algorithm2014@edu.jar.jp
  - サブジェクト 「アルゴリズムとプログラム実践講座・レポート」
  - 学生証番号と名前は、 メールの本文に書いてください。
  - 〆切：次の週の日曜日深夜 （講評の都合上。〆切後も受付ます）
  - プログラムは（お手本として）公開することがあります。適宜、作者名や　コピーライトをいれておいてください。公開不可の場合は、プログラムの冒頭にその旨、コメントをいれておいてください。
  - 質問・作問提案も歓迎 （作問については採用の場合は別途加点）
  - サンプルプログラムは「初心者向け」です。 **上級者は無視してください。**

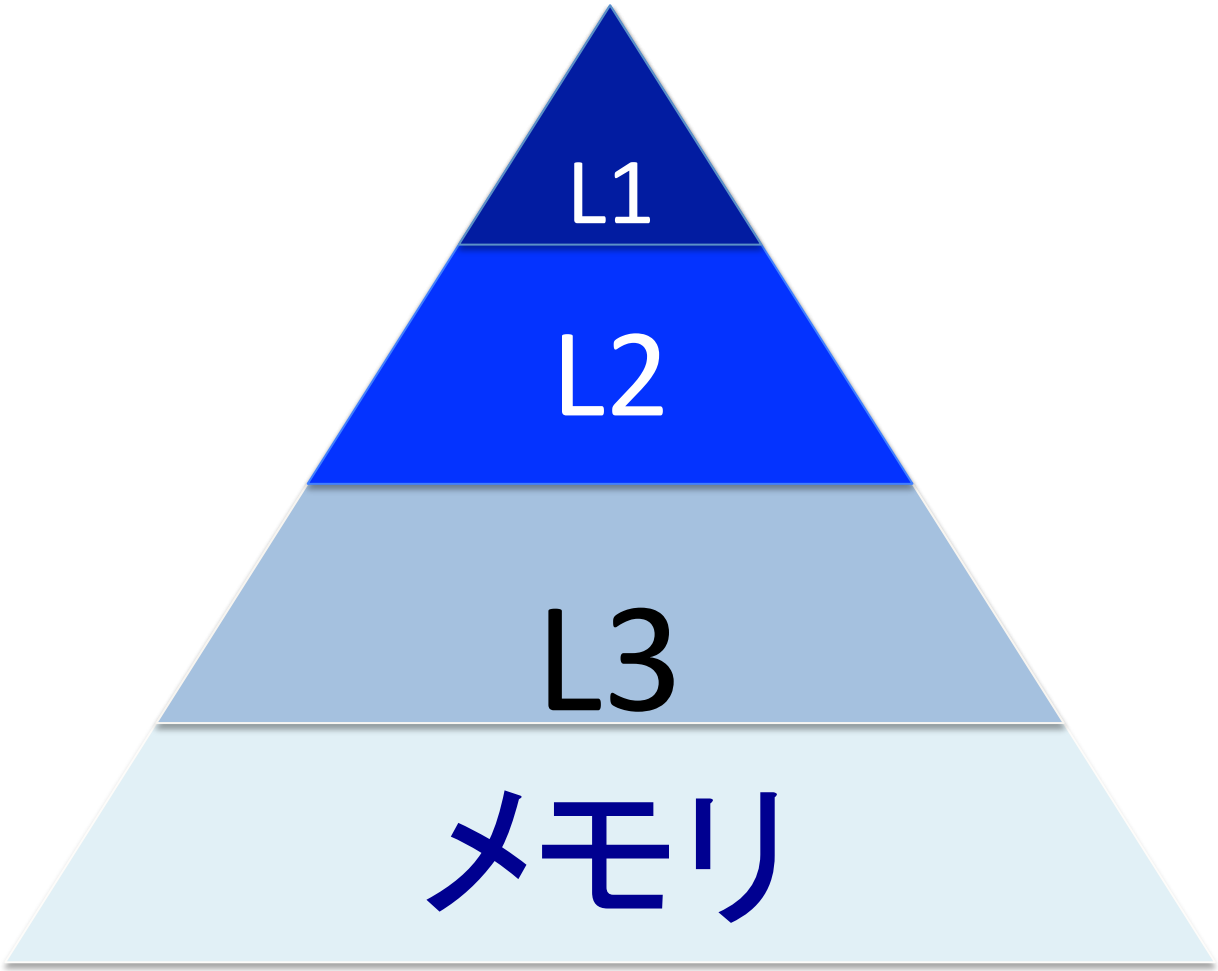
アルゴリズムの良さは
どうやってはかるのか？

その前に

計算機の仕組み

メモリ階層

メモリー階層



ワンポイントレッスン

速くて巨大なメモリは
お金をいくらかけても
作れない

速くて巨大なメモリは作れない

「速い」とは？

→　計算に間に合うように、データを供給

→　クロックスピード　(たとえば 3.3 GHz)

→　1サイクル　0.3 ns

0.3 ns ですすめる距離

- 光　10cm
- 電線の電気　5cm

速くて巨大なメモリは作れない

「速い」

→　計算に間に合うように、データを供給

→　クロックスピード　(たとえば 3.3 GHz)

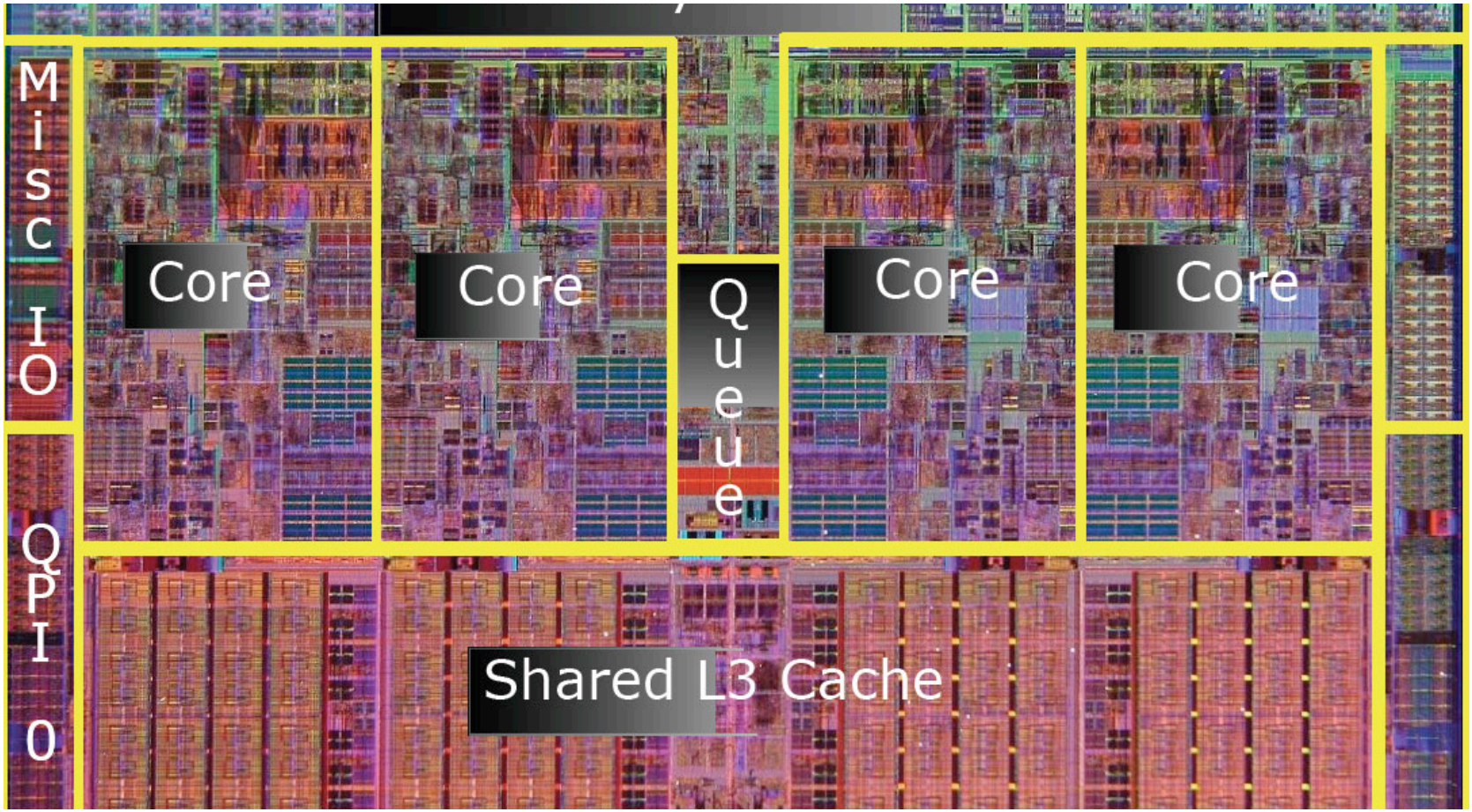
→　1サイクル　0.3 ns

0.3 ns ですすめる距離

- 光　10cm
- チップ内配線の電気　細いので　数ミリメートル

プロセッサ

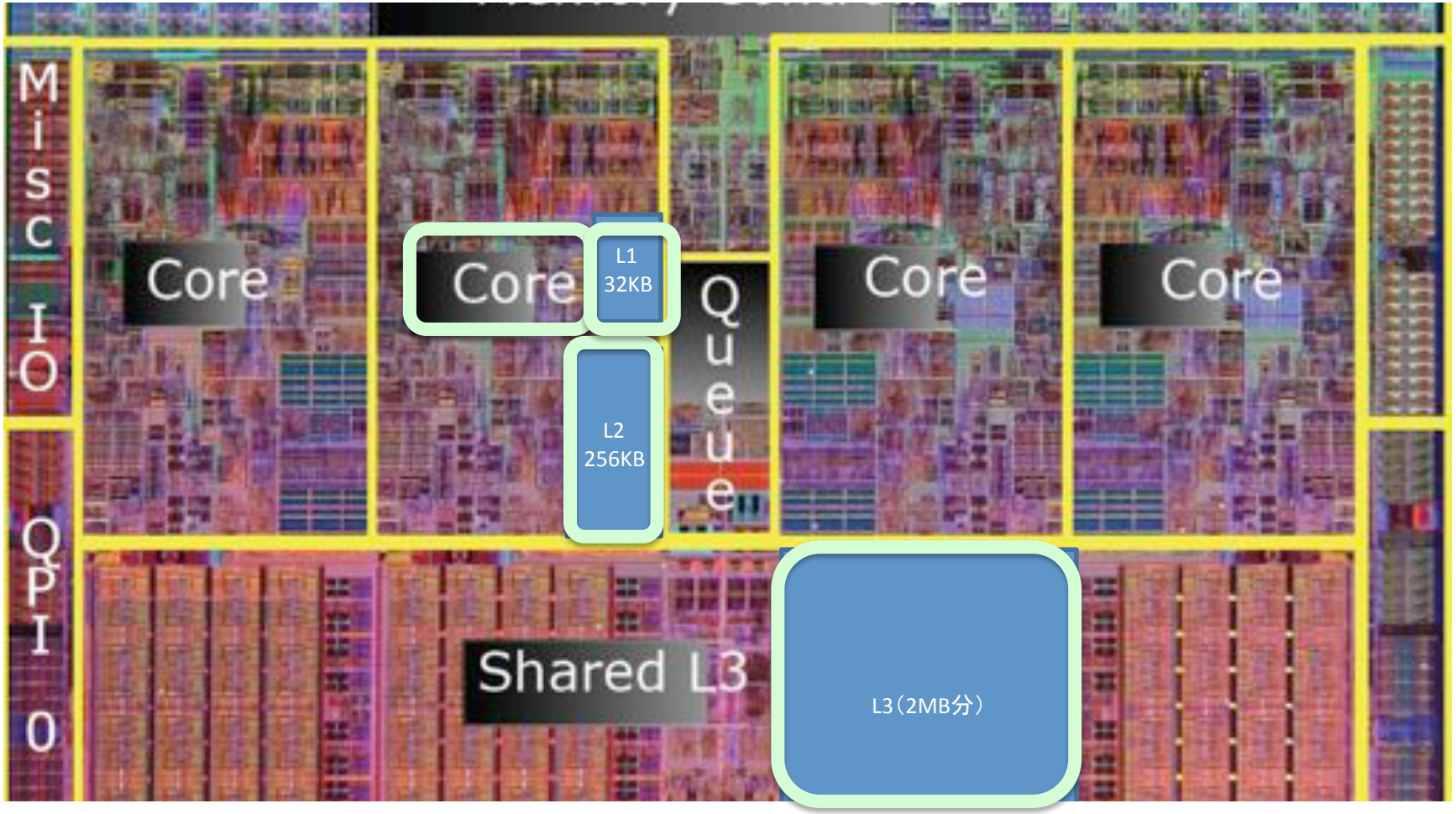
INTEL Nehalem アーキテクチャ
Bloomfield (ソケット 1366 プロセッサ)



http://pcjuornal.com/topics/Intel_Lynnfield_new-1.html より

キャッシュ

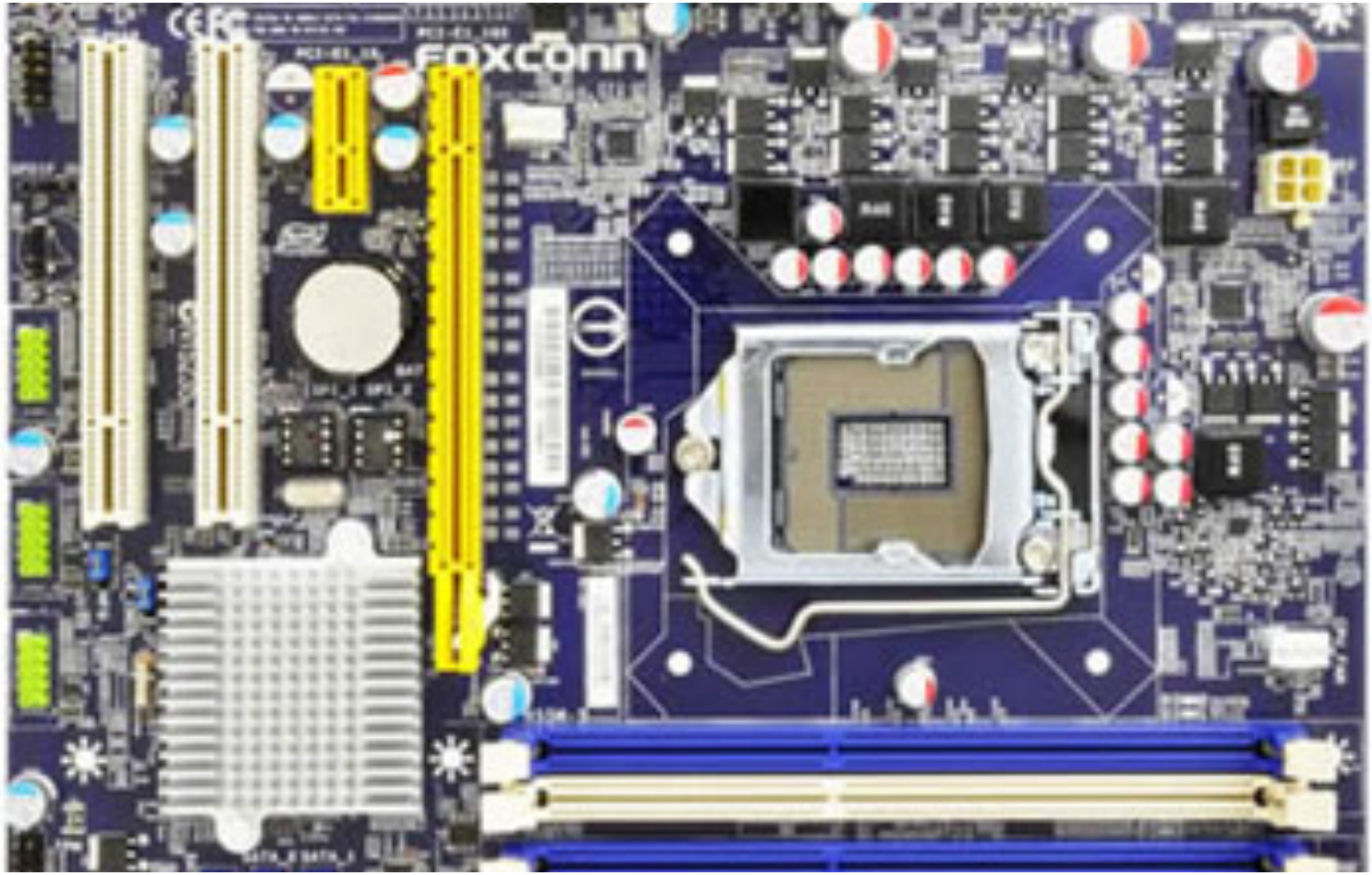
L1 - 32KB, L2 - 256KB , L3 - 2MB



L1 - 32KB, L2 - 256KB , L3 - 2MB

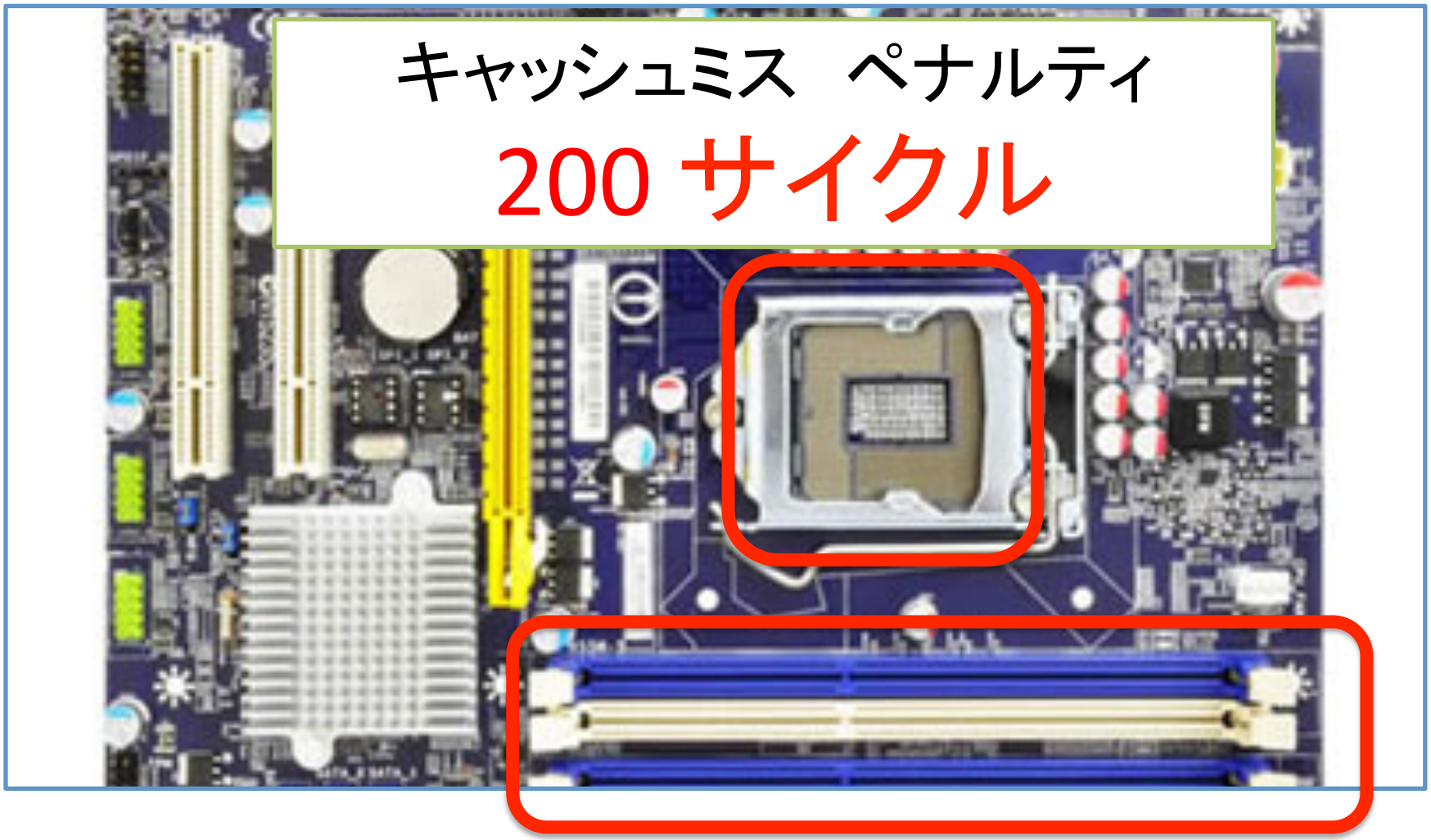
L1 キャッシュ： 1.4mm x 1.4mm
2～4 サイクル (0.5～1ns)
L2 キャッシュ ： 2mm x 2mm
8サイクル (2～3ns)
L3 キャッシュ： 4mm x 4mm
32サイクル (8～16ns)

メモリアクセス



http://www.sgame.jp/news/8/5138　より

メモリアクセス



レイテンシの隠蔽の工夫

- **キャッシュ　リプレースメント**
  再び使いそうなデータだけをとっておく
- **プリフェッチ**
  使いそうなデータを、あらかじめ持ってくる
- **投機実行**
  分岐命令は、片方を試して失敗なら巻戻す

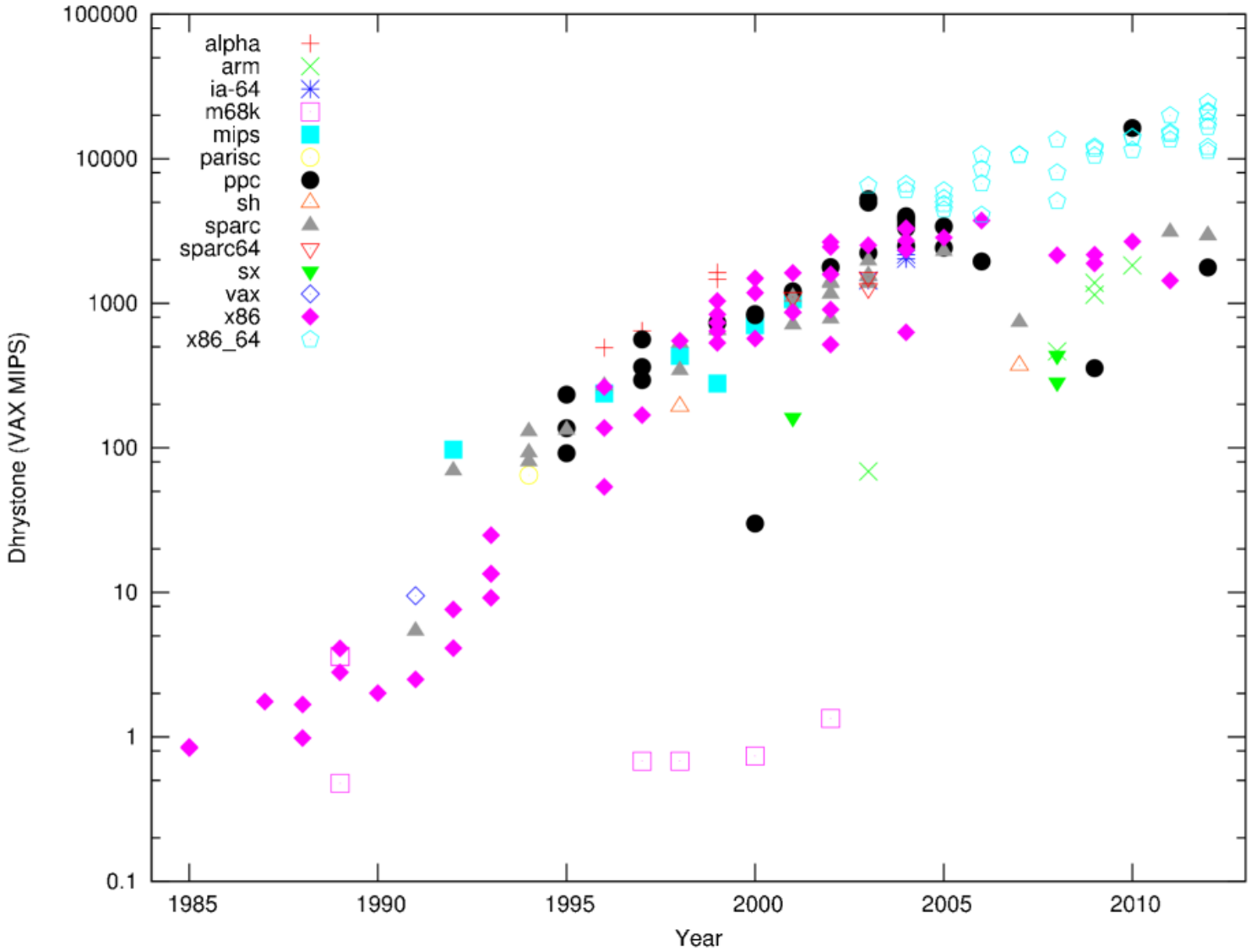
難しいところ

- 速さ勝負（200クロックを2クロックに）→できる事は限られる
- ハードウェアで実装
- 使えるハードウェアリソースの制限は、きつい

（おまけ）

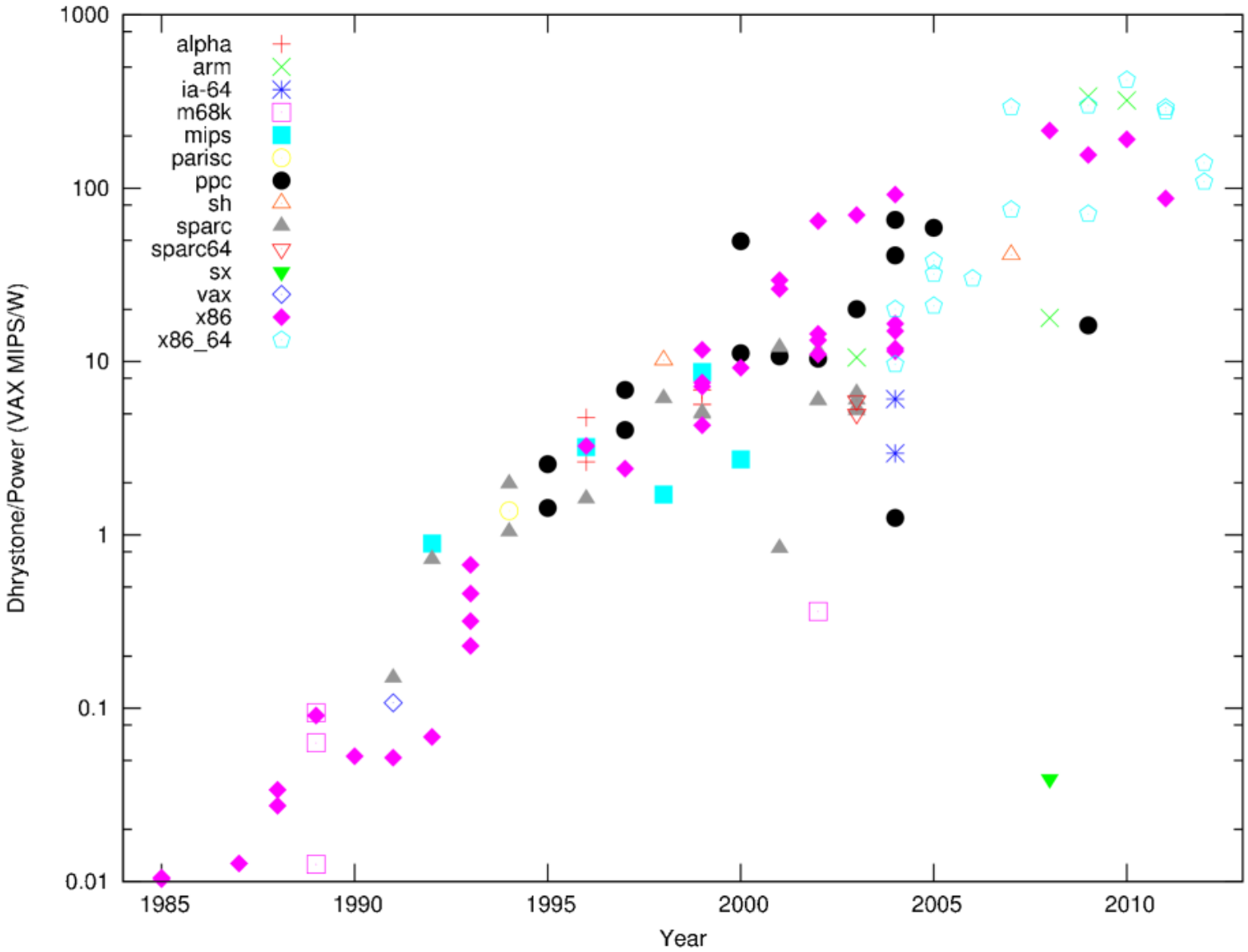
近年の計算機の性能向上

Performance (Dhrystone)



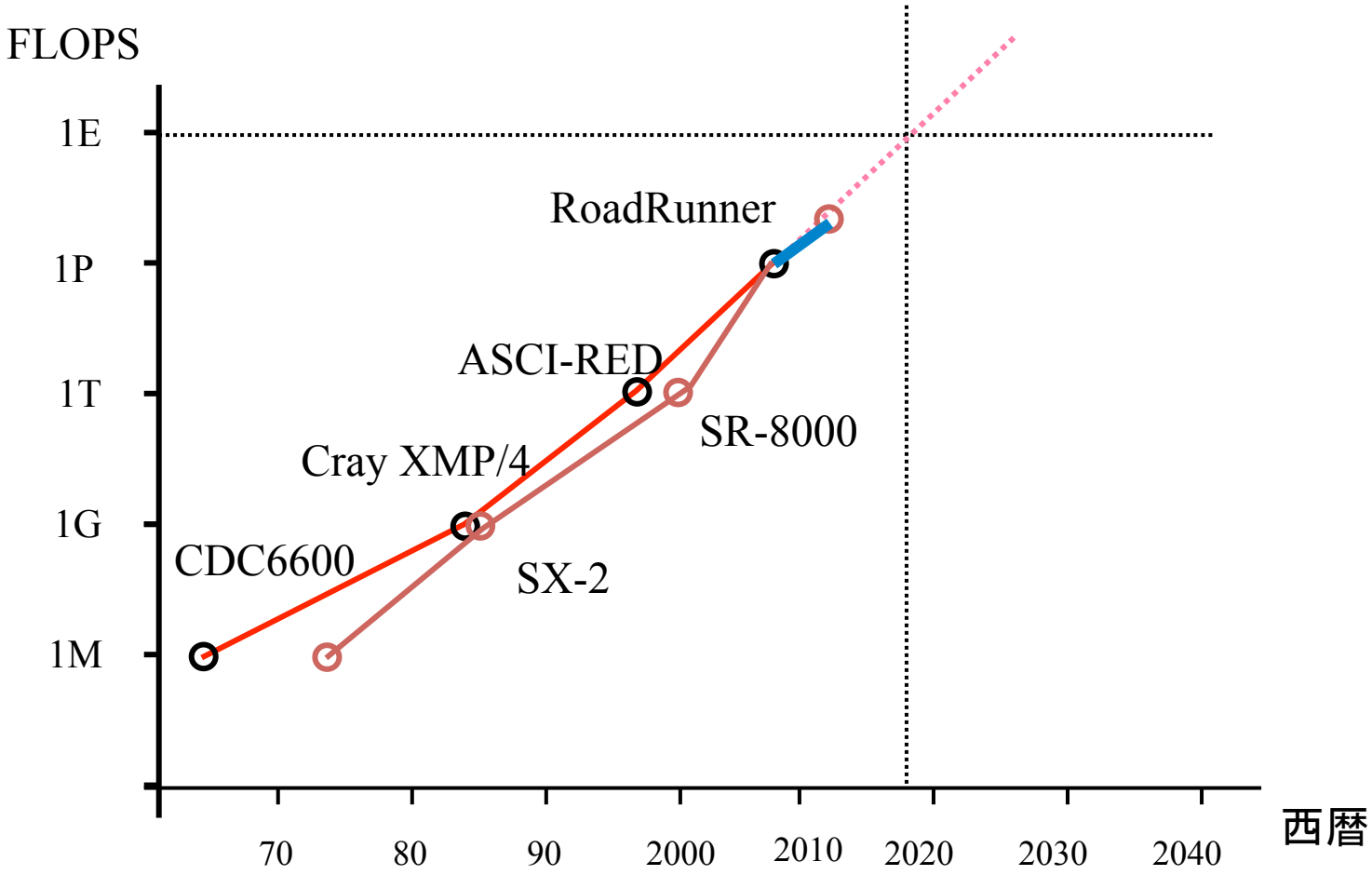
Computer Zooによる計測　[泊・平木]

Performance/Power



Computer Zoo による計測　[泊・平木]

高速化の歴史



平木先生作成の資料より

アルゴリズムの良さは
どうやってはかるのか？

アルゴリズムの良さはどうやって計る？

時間で語るのは（殆ど）意味がない

「このビルは構造をスパコンを◎時間使って計算したから安全（1974）」

→　今だったら、パソコンで・・・

したいこと

- 「どのハードを使ったか」考えなくて良いように　抽象化して考えたい。
- 問題がスケールするか判断したい

(キャッシュが当らなくても、たかだか300倍しか・・)

RAM モデル
Random Access Machine

- 記憶領域（メモリ）は無限　（レジスタは有限）
- メモリは番地指定で　単位時間でアクセス。
- 整数も実数もすべて　1ワード
- 算術演算も単位時間で実行

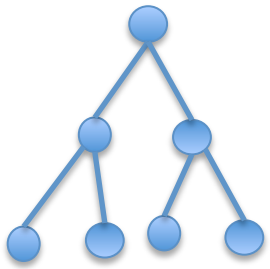
基本演算は、「ロード」「ストア」「演算」

入力はメモリに格納されている　→　**入力サイズ**

大雑把な話

繰り返しがなければプログラムはすぐ終る。
時間がかかるのは、「繰り返しの演算」の部分

- For 文の中身が何度実行されるか？
  - たとえば　1..n の　３重ループなら　n^3回
- Tree のサイズがどのくらいか？
  - たとえば、leaf がn 個なのか
  - たとえば、高さが n なのか



オーダー

「漸近記法（Asymptotic Notation)」「O- 記法」「ランダウ（Landau)の記法」

気分は「細かいことは気にしない」

良く使うもの　（同じものを、違う書き方をすることもある）

O(1)　定数時間
O(log n)　対数，　O((log n)^c)　対数多項式
O(n)　線形関数，O(n log n)　準線形，O(n^2)　二乗，
O(n^c)　とか O(poly(n))　多項式関数
O(C^n) とか　2$^{O(n)}$　指数関数　　・・・・・

組合せ爆発の怖さは、たとえば、「フカシギの数え方」の動画参照
http://akashi.ci.i.u-tokyo.ac.jp/mary/lectures/algorithm/

スケールするか？

たとえば、入力のサイズ n に対して

線形時間の時間計算量

データのサイズが10倍になったら時間も10倍 → 速い

準線形時間 O(n log n) の時間計算量 → 速い

多項式時間アルゴリズム → 計算できる

二乗時間アルゴリズム

データのサイズが 10倍になったら計算時間は 100倍

指数時間アルゴリズム → 難しい

データサイズが1増えても 厳しくなることも → 組み合わせ爆発

色々な計算量

計るルールは自分で決める（広まるかは多数決）

一般的なのは、worst case analysis、

時間計算量　(time complexity)

空間計算量　（space complexity)

- output sensitive なアルゴリズム

  入力に対して指数時間、出力に対して線形

- randomized アルゴリズム(乱択アルゴリズム）

  ランダム性をいれることで高い確率で速く動く

Randomized Algorithm
（かっこいいなと思ったアルゴリズム）

- Random Quick Sort Algorithm
- クイックソートは速く動くことが多い
  - 「じゃがいもソート」のパロディー動画
- ただし入力がすでにソートされていると$O(n^2)$

  ２つに分割する際、ピボットを端から取ると、全部が同じ側になって、分割が１：n　と偏ってしまうため
- なら、ピボットにする要素をランダムに選べばいいじゃない！　→　**高い確率でうまくいく**

アルゴリズムの記述

疑似コード

```
x は、集合X のメンバーか関数（2分探索）？

Member(x,X) ただし 集合X はソートして配列Aに格納してある
(1) i:=1; j:=n;
(2) i>j ならば 「x は X の要素でない」 と答えて終了
(3) k:＝(i+j)/2; （ただし k は 整数に丸める）
      A[k]=x なら 「x は X の要素」 と答えて終了
      A[k]<x なら i:=k+1 として (2) へ戻る
      A[k]>x なら j:=k-1 として (2) へ戻る
```

人間が読むことを想定したコード

相手が人間なので、色々察してもらえるから適当で大丈夫。
ただ、たとえば計算量が察せるように、大切な事をきちんと書く。
（PASCAL っぽく書くことが多い）

レポート・成績について

- ほぼ毎回、プログラミング課題を出題する予定
  - 効率の良い計算機実験のためのツールを使ってみる
  - アルゴリズムの実装
  - ライブラリの利用・・・　など

- ３回以上、レポートとプログラムのファイルの組をメールで提出のこと
  - E-mail: algorithm2014@edu.jar.jp
  - サブジェクト 「アルゴリズムとプログラム実践講座・レポート」
  - 学生証番号と名前は、 メールの本文に書いてください。
  - 〆切：次の週の日曜日深夜 （講評の都合上。〆切後も受付ます）
  - プログラムは（お手本として）公開することがあります。適宜、作者名や　コピーライトをいれておいてください。公開不可の場合は、プログラムの冒頭にその旨、コメントをいれておいてください。
  - 質問・作問提案も歓迎 （作問については採用の場合は別途加点）
  - サンプルプログラムは「初心者向け」です。 **上級者は無視してください。**

推奨環境など

- Linux, Mac,（Windows+Cygwin）
- 仮想マシン環境（VMware, VirtualBox, Parallels)
  - 余裕があれば、いろいろな組み合わせを試して比較してみると面白いと思います

- 言語
  - 自由。ただし、一般的でない言語については、上記いずれかのOS 上にインストール可能なもの

環境構築のための　TIPSなど

http://akashi.ci.i.u-tokyo.ac.jp/mary/lectures/algorithm/ にリンク

- 物理のかぎしっぽ（Linux, Cygwin, TeX など）
- VMware Player, VirtualBox

- パッケージ、ポートなど
  - Mac Ports
  - yum (Fedora Core, CentOS用)
  - apt-get (debian用)
  - Windows への　Ruby の installer

他に「是非お薦め！」のページがあったら教えてください

第一回　課題

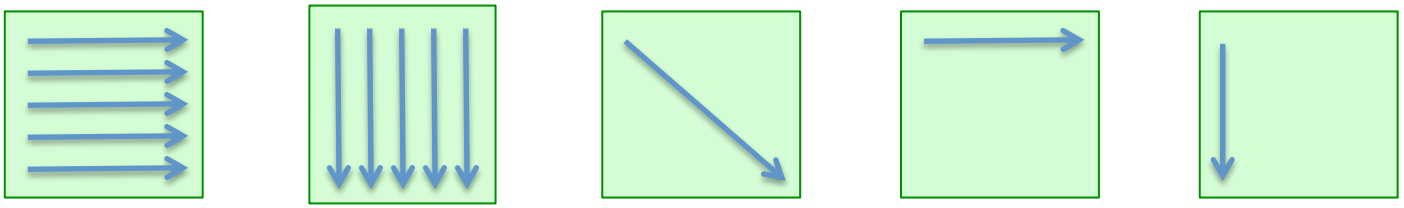
Sample programs

ijloop.c, diagonal.c + Makefile

array.rb, hash.rb

行列のアクセスの時間の計測

- 正方行列に、(a)行順全要素、(b)列順全要素、(c)対角要素のみ、(d)1行のみ　(e)1列のみ代入を行ったあと、代入した要素の読出しを行うプログラムを書き、かかった時間を計測、比較考察せよ。
- 実験環境のうち、ハードウェア、OS、言語、コンパイラオプションなど実験環境のうち、いくつかを変更して、比較考察せよ。
- **実験環境について、CPU のバージョン・動作周波数・メモリサイズ・カーネルのバージョンなど、他人に再現可能なように記述すること。**



(a)　(b)　(c)　(d)　(e)

この課題の狙うところ

- 簡単な時間の計測
- ハードウェア、OS、言語、コンパイラオプションなどによる速度等の違いを比較し実感する
- キャッシュの効きが実感できるとなお良い（でも、Ruby では見えないかも）

- 実験の自動化。（再現実験が楽に行えるように）
  - Makefile を使ってみる

サンプルコードに関する注意

- 授業の際に提示するコードは、必ずしも良いコードでないことも多い。お手本ではない。
- たとえば、第一回の　C　のサンプルソースは、コメントも日付も入ってない。