全体ミーティング(2011/1/19)

M1 池尻　拓朗

今日紹介する論文

- Compact Dispatch Tables for Dynamically Typed Object Oriented Languages, Jan Vitek et al. *CC'96*

- dynamic binding: メソッド呼び出しに実行時にどの実装が呼ばれるかを決定する手法

- dynamic bindingのためのspace-efficientで実行時間へのコストも低いdispatch tableの提案

背景

- 動的オブジェクト指向言語ではほとんどのフィールドアクセスやメソッド呼び出しにdynamic bindingを必要とする

- しかしdynamic bindingは遅い
  - 毎回オブジェクト名、メソッド名とメソッドの実装へのリファレンスの対応表をみて探さなければならない

背景(2)

- 様々なdynamic bindingの手法がある
  - Dispatch Table Search
  - Selector Indexed Dispatch Tables
    - Virtual Function Tables
    - Dispatch Table Compression
  - Dynamic Techniques
    - Inline Caches
- 従来の方法ではテーブルサイズが大きい、または小さくても実行時間へのペナルティが大きい

注意

- セレクタ=メソッド名のこと
- ベンチマークにはSelfのOBJECTWORKSというフレームワークを使用
- single inheritanceを仮定

Dispatch Table Search (DTS)

- 継承関係に従ってメソッドが見つかるまでクラスを一つずつ探していく方法
  - それぞれのクラスが<セレクタ、メソッドアドレス>のハッシュテーブルを持っておく

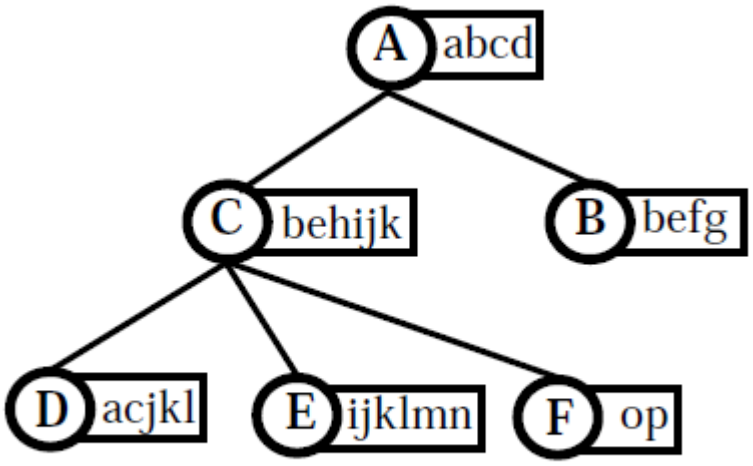
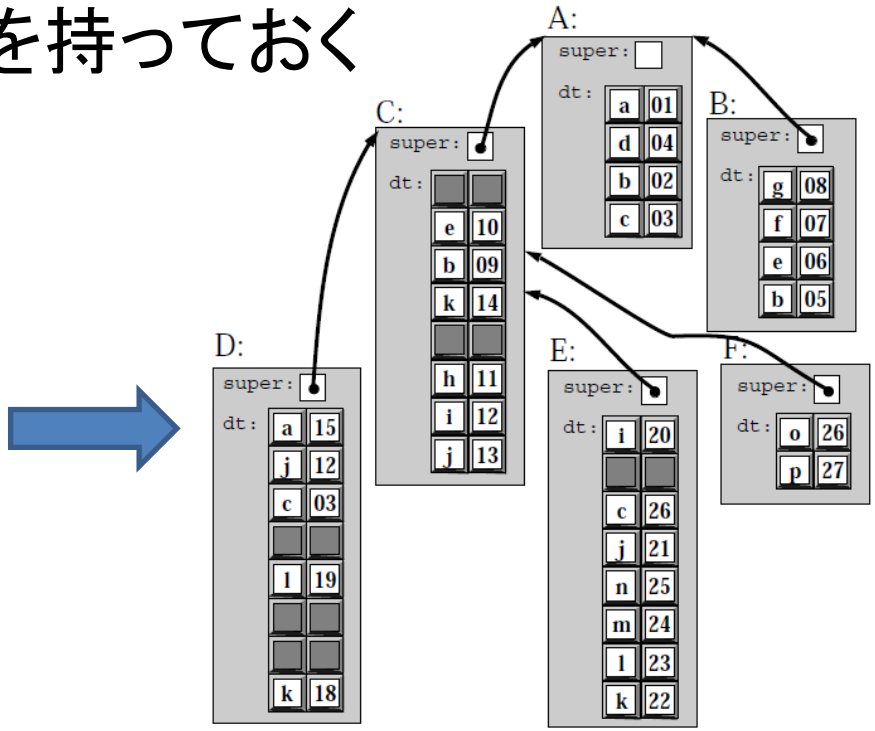


Figure 7 Sample class hierarchy.



Dispatch Table Search（２）

- dispatchの方法
  1. メソッドセレクタをハッシュしてインデックスを取得する
  2. テーブルにインデックスを使ってアクセスしてセレクタが一致していれば指しているアドレスに飛ぶ
  3. 一致していなければハッシュの衝突とみなし、親のハッシュテーブルを見に行く
  4. 継承木のルートまで行って見つからなければメソッドが見つからない旨のエラーを出す

Selector Indexed Dispatch Tables (STI)

- C個のクラスとS個のセレクタがある場合に C×Sの二次元配列のテーブルを作る
- 配列の中身はコンパイル時にあらかじめルックアップして埋めておく

	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p
A	01	02	03	04	0	0	0	0	0	0	0	0	0	0	0	0
B	01	05	03	04	06	07	08	0	0	0	0	0	0	0	0	0
C	01	09	03	04	10	0	0	11	12	13	14	0	0	0	0	0
D	15	09	26	04	10	0	0	11	12	17	18	19	0	0	0	0
E	01	09	03	04	10	0	0	11	20	21	22	23	24	25	0	0
F	01	09	03	04	10	0	0	11	12	13	14	0	0	0	26	27

※0の部分は対応するメソッドが存在しないことを表す

Selector Indexed Dispatch Tables（2）

- dispatchの方法
  - 作ったテーブルに、クラス名とセレクタ名をインデックスとして使ってアクセスする

- DTSに比べて高速化できる（250->6 cycles)が
- テーブルサイズは非常に大きくなる (93KB->16.5MB)

	STI	Value
speed	T_{STI}	6 cycles
data size	$S*C$	16.5 MB
code size	$4c$	811 KB

Table 7 Characteristics of STI

STIの問題点

1. 無駄な情報（0）が多くテーブルサイズが極端に大きくなってしまう
   - 普通、90%以上が0で埋まってしまう

2. クラスの階層構造の変化に大きな影響を受けてしまう
   - クラスがひとつ動的に追加されただけでも表全体を作り直す必要がある

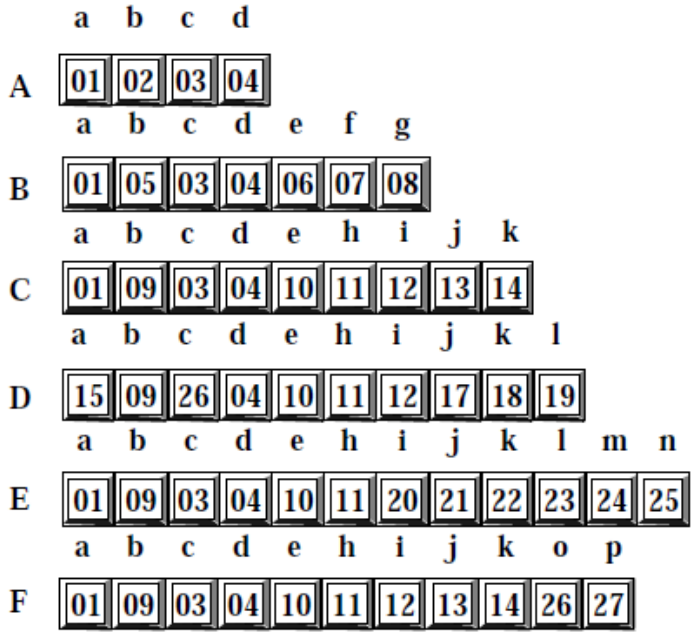
- 1.の問題点を改良することが本論文の目的

STIの種々の改良方法

- Virtual Function Table
  - 静的な型情報を使って空きがないテーブルを作る
- Dispatch Table Compression Techniques
  - Selector Coloring
    - テーブルの空きの部分を詰めて圧縮する
  - Row Displacement
    - 二次元配列を一次元にならして圧縮する
- Compact Dispatch Tables revisited: CT-95
  - 継承関係を利用して圧縮：提案手法

Virtual Function Tables (VTBL)

- 静的型チェックされたオブジェクトは、所属するクラスに呼び出されうる全てのセレクタが実装されていることが保証される（継承除く）
  - つまり、メソッドが見つからないということは起こらないはず
- それぞれのクラスについてスーパークラスのセレクタを全て含むように注意してDispatch Tableを作る



Virtual Function Tables (VTBL)

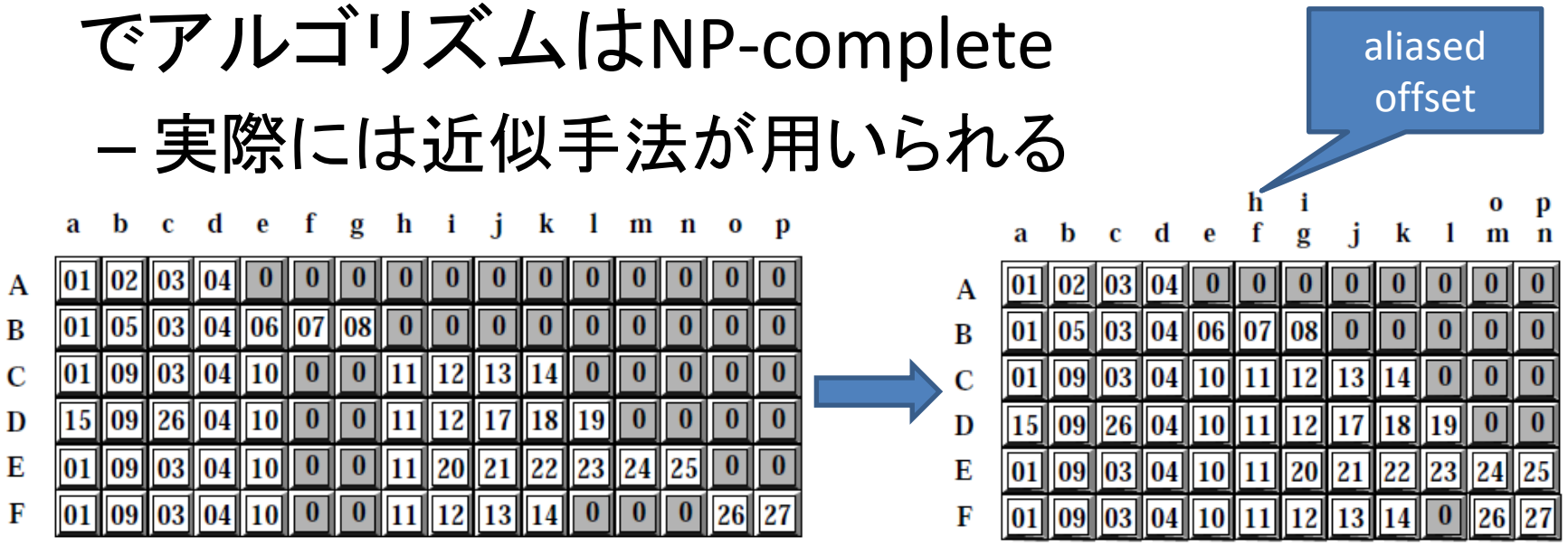
- dispatchの方法
  1. 対象のオブジェクトのクラスごとにあるテーブルを取ってくる
  2. 対象のセレクタをインデックスとしてテーブルの先のアドレスにジャンプする
- 早くてテーブルサイズもまずまずだが、動的言語に適用できない

	VTBL	Value
speed	T_{VTBL}	6 cycles
data size	$O_{VTBL} = \sum_{c=1}^{C} methods(c)$	868 KB
code size	$4c$	811 KB

Table 10

Selector Coloring (SC)

- 異なる参照が入っていない二つの列を共有することで空のセルを詰める
- これはグラフの最小彩色問題に帰着できるのでアルゴリズムはNP-complete
  - 実際には近似手法が用いられる



Selector Coloring

- dispatchの方法
  1. テーブルでaliased offsetとなっているメソッドは、先頭にどのセレクタ名で呼び出されたのかチェックするコードを挿入する
  2. メソッド呼び出しの際にはセレクタ名を一緒に渡してチェックできるようにする　あとはSTIと同じ
- テーブルサイズは小さいがコードサイズが大きくなり、スピードもやや落ちる

call site:

```
table = object->dt;
method = table[ #color_offset ];
method(object, #selector_code, arguments);
```

method prologue:

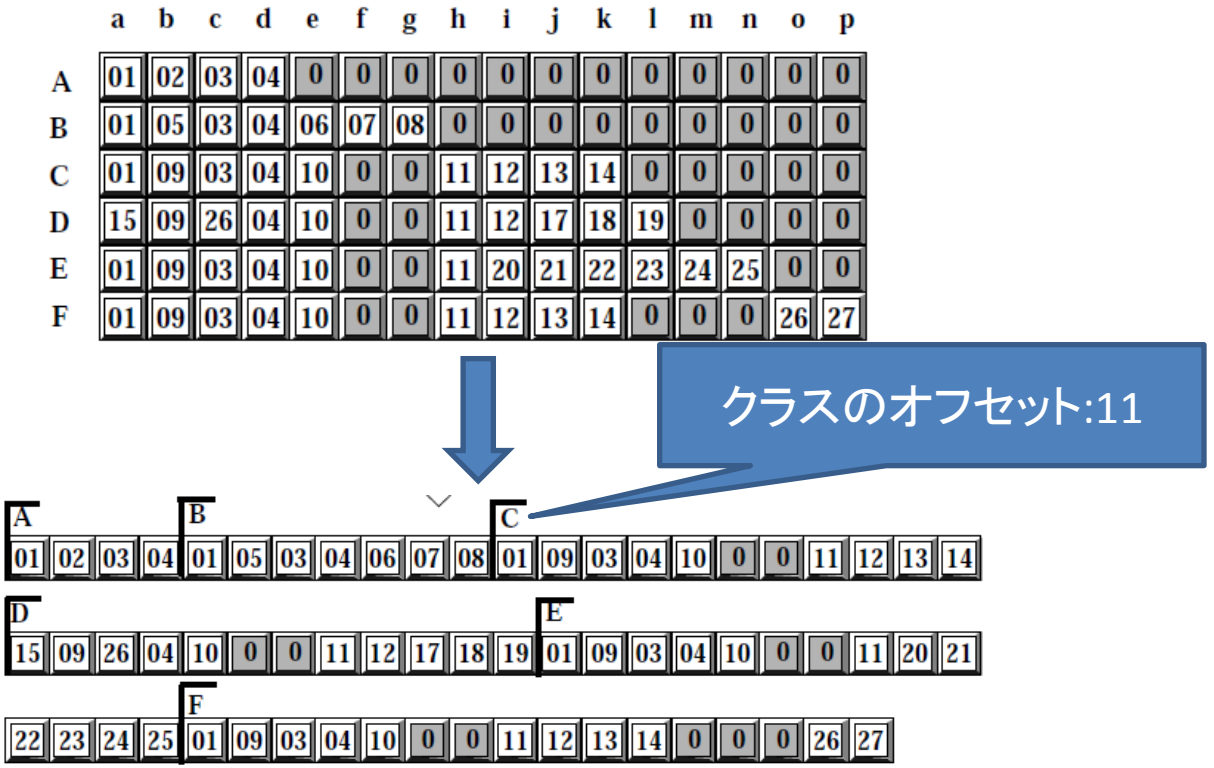
```
if (s != #The_selector_number)
    messageNotUnderstood();
```

	SC	Value
speed	T_{SC}	9 cycles
data size	$O_{SC} = O_{VTBL} * 133\%$	1,154 KB
code size	$4c + 3M$	916 KB

Table 11

Row Displacement

- 二次元配列を空のセルを重ねるようにして一次元配列にまとめる



Row Displacement

- dispatchの方法
  - SCと殆ど同じで、セレクタ名の代わりにクラスのオフセットを渡す

- SCよりも簡単にテーブルのサイズを小さくすることができる

	RD	Value
speed	T_{RD}	9 cycles
data size	$O_{RD} = O_{VTBL} * 101\%$	819 KB
code size	$4c + 3M$	916 KB

Table 14

Compact Dispatch Tables revisited: CT-95

- テーブルの作り方がやや複雑
  - 前準備としてSTIに以下の操作を適用
  1. factoring conflict selectors
  2. trimming
  3. selector aliasing
  - さらにpartitioningを適用

Factoring conflict selectors

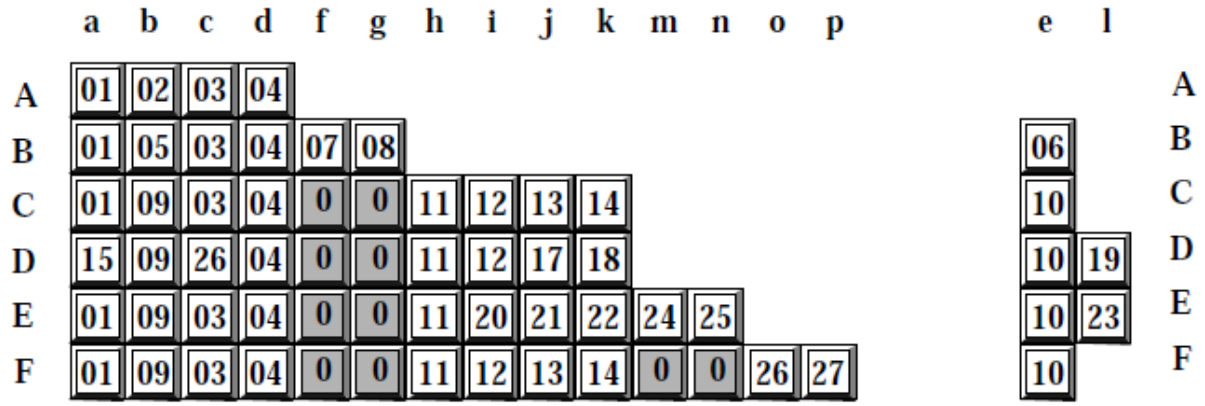
- 二つのクラスに実装されているが、そのクラスがお互いに継承関係にないセレクタをconflict selectorとして分離する
  - eはBとCのクラスに実装されていて、継承関係にない
  - これによりSelector aliasingで圧縮しやすくなる

	a	b	c	d	f	g	h	i	j	k	m	n	o	p
A	01	02	03	04	0	0	0	0	0	0	0	0	0	0
B	01	05	03	04	07	08	0	0	0	0	0	0	0	0
C	01	09	03	04	0	0	11	12	13	14	0	0	0	0
D	15	09	26	04	0	0	11	12	17	18	0	0	0	0
E	01	09	03	04	0	0	11	20	21	22	24	25	0	0
F	01	09	03	04	0	0	11	12	13	14	0	0	26	27

	e	l
A	0	0
B	06	0
C	10	0
D	10	19
E	10	23
F	10	0

Trimming

- 後ろの空のセルを削る



Selector aliasing

- クラスが互いに継承関係において関係がないなら、違うセレクタに同じオフセットを割り当てることで圧縮できる

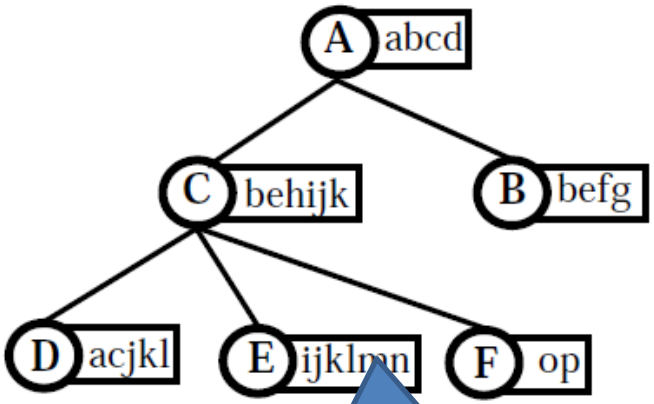


Figure 7 Sample [illegible]erarchy.

m,nとo,pは継承関係において関係がない

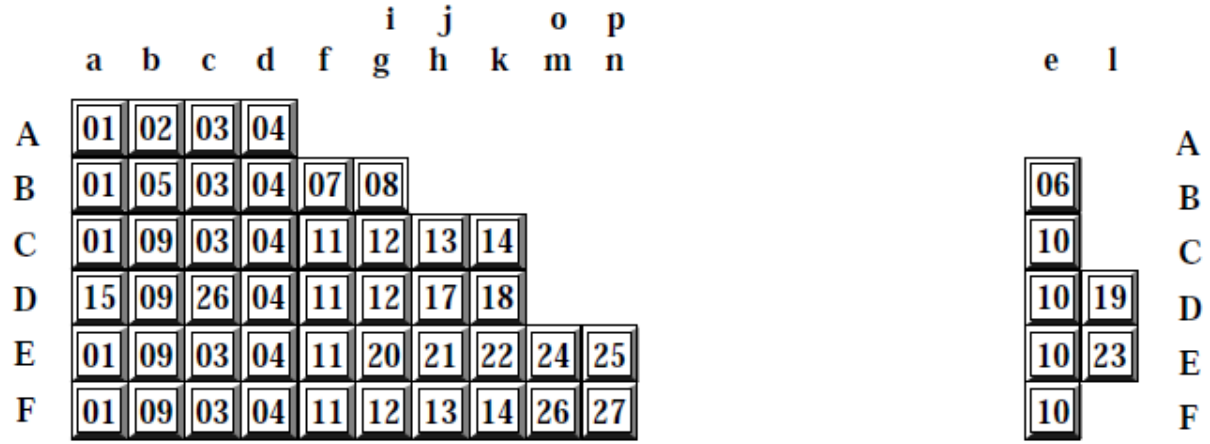


Figure 30 Aliasing the dispatch tables.

partitioning

- 基本的なアイディアは、テーブルの一部分を共有してテーブルサイズを小さくすること
  - １００個のメソッドを親から継承していてそのうち５つしか再定義していないなら、95個のメソッドをもつ共通テーブルと5つのメソッドからなる内容の異なる2つのテーブルに分けられる

partitioning

- それぞれのクラスは同じ数のパーティションテーブルをもつように制限
- さらに同じオフセットからアクセスされたパーティションは同じサイズに制限

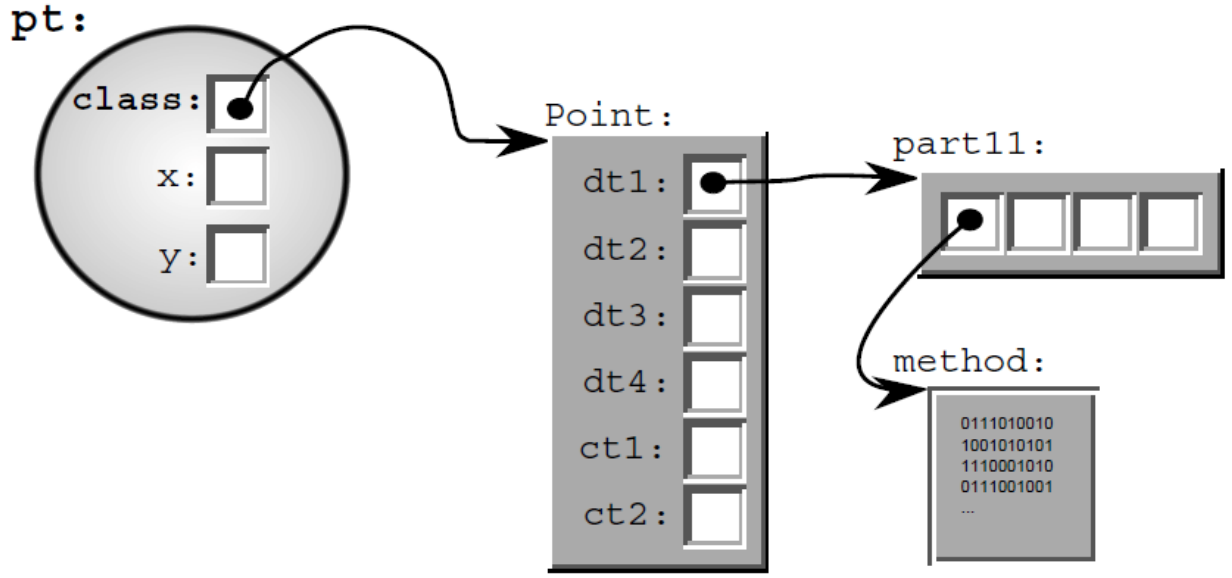
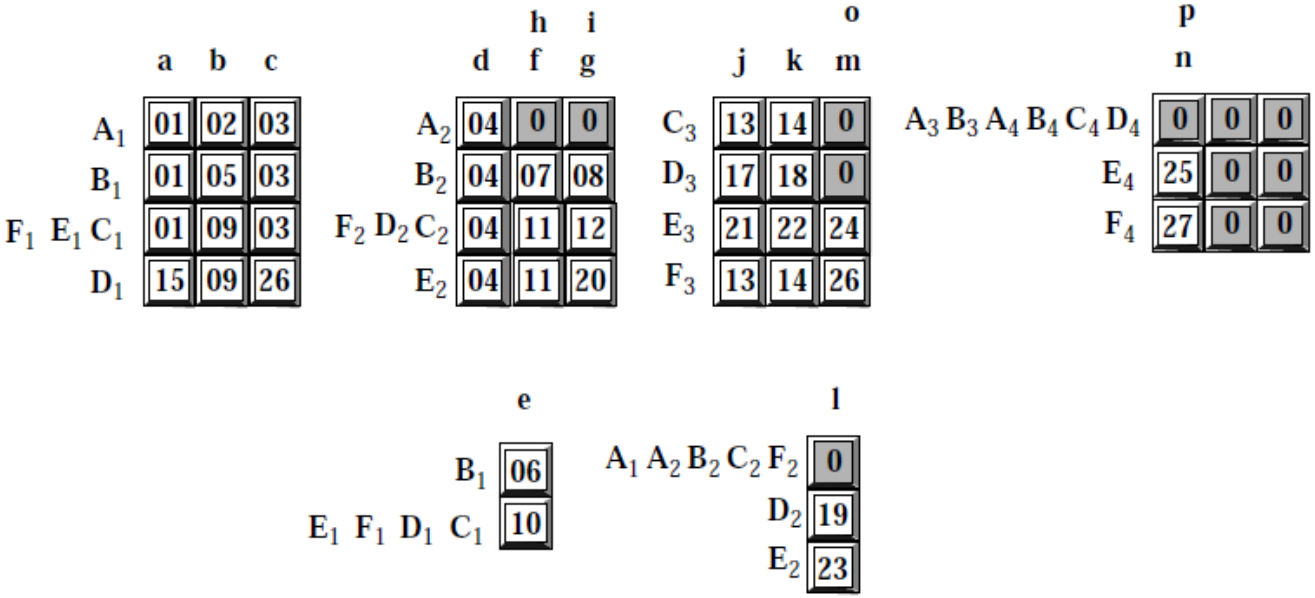


Figure 39 Objects, classes, partitions and methods.

Table partitioning

- 先程のテーブルをパーティションのサイズが同じになるように分割
  - 同じ中身を持つエントリは共有することができる
    - もちろんセレクタ名をメソッドでチェックする必要がある
    - conflict selector tableは性質上特に共有しやすい



Compact Dispatch Tables revisited: CT-95

- できたテーブルを以下のようにして使う

call site:

```
1. load   [object + #class_offset], class
2. load   [class + #table_offset],  table
3. load   [table + #selector_offset], method
4. call   method
5. setlo  #selector_color, color
```

method prologue:

```
5. cmp    #expected_color, color
6. bne    #message_not_understood
7. nop
8. <first instruction of target method>
```

- 実行時間をそれほど増やさずにテーブルサイズを著しく小さくできた

	CT-95	Value
speed	$T_{CT\text{-}95}$	11 cycles
data size	$O_{CT\text{-}95}$	221 KB
code size	$5c$	1 MB

Table 26 Compact Dispatch Tables (CT-95).

おまけ

- inline cachingと組み合わせてみた
  - キャッシュミス時にCT-95を呼び出す

	IC	Value
speed	$T_{IC} = hitTime_{IC} * hitRate + (1\text{-}hitRate) * (82 + T_{LC})$	12.3 cycles
data size	$O_{IC} = O_{LC}$	94 KB
code size	$4c + 3M$	916 KB

	IC + CT-95	Value
speed	$T_{IC+CT\text{-}95} = hitTime_{IC} * hitRate + (1\text{-}hitRate) * (82 + T_{CT\text{-}95})$	11.3 cycles
data size	$O_{IC+CT\text{-}95} = O_{CT\text{-}95}$	221 KB
code size	$4c + 3M$	916 KB

Table 27 IC + CT-95.

結果

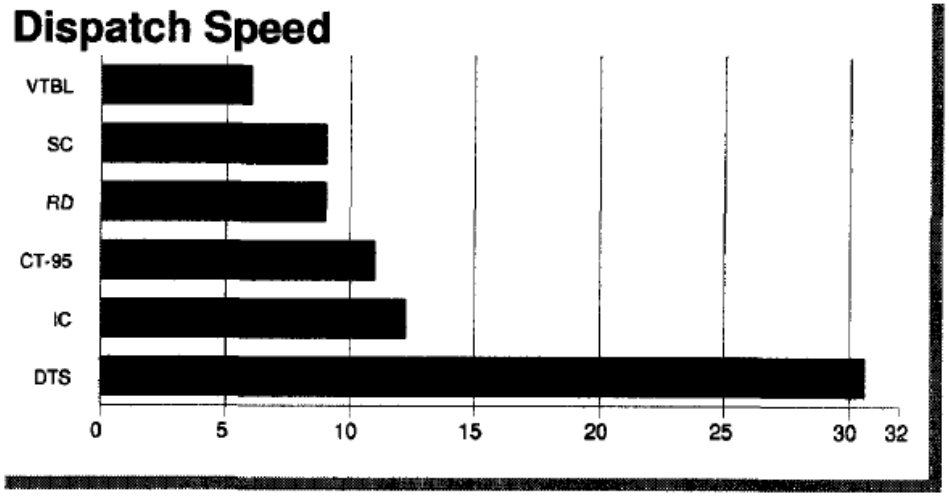


Figure 20 Comparing dispatching speeds.

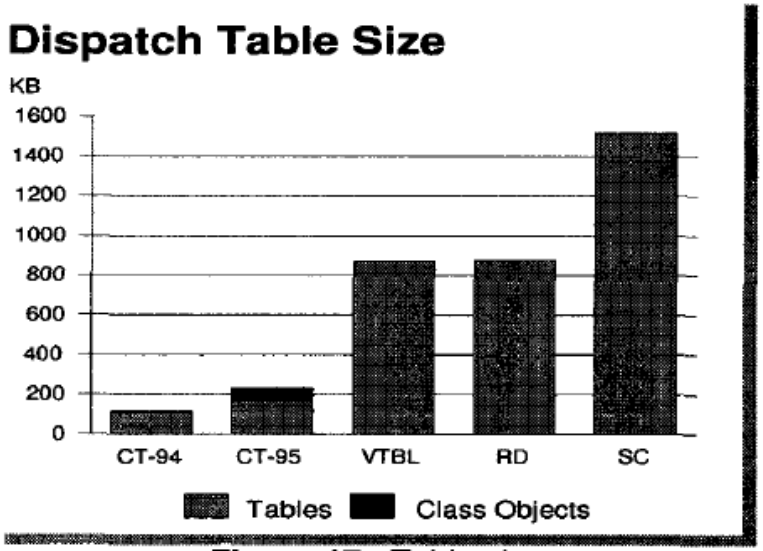


Figure 17 Table sizes.

まとめ

- 動的な言語ではdinamic bindingもそれほど簡単なことではない
  - dispatch tableのサイズが大きくなる
  - 小さくするとどうしても実行時間がかかる

- テーブルサイズが小さく、実行時間へのペナルティも少ないdinamic bindingの実装方法を考えた