プログラミング言語設計論

2014年度

第9回: 自己反映計算・メタプログラミング

担当: 増原英彦

復習: テンプレート

- 型名→定義 なる関数
- 目的: 色々な型で定義を作る
- 計算:
  - コードの穴に型名を埋める (ゆえにテンプレート)
  - 他のテンプレートを使う
  - プログラム実行前に実行
- 入力: 型名
  - 定数を取れるものもある
- 出力: クラスや関数の定義

マクロ: テンプレートの一般化

- 式→式 なる関数
- 目的: 記述の短縮・制御構造やデータ構造定義
  - 式の途中で使うこともある
- 計算: 簡単なものから強力なものまである
  - 穴埋め+α (CPP)
  - 一般的な関数 (LISP)
- 入力: 式
- 出力: 式や定義

マクロ記述言語の
計算能力

C preprocessor (CPP)

- 記述言語
  - テンプレート: 式の穴を埋める
  - +α：名前を結合する等
- 難しさ: 評価順序・回数

(例)

```
#define or(x,y) ((x)!=NULL?(x):(y))
… open(or(file, opt[opt_i++])) …
```

黄色はデータとしての式

何故括弧付? (x)

何故関数にしない?

LISPマクロ

- 記述言語: LISP自身 (強力!)
  - テンプレート的な記述のサポート
    - ◆S式表現: LISPのプログラムはLISPの基本データ構造と同じ表現
    - ◆逆引用符記法: データのとしての式に計算を埋め込む
- 例

```
(defmacro dotimes (spec body)
 `(let ((max ,(cadr spec))
        (,(car spec) 0))
   (while (< ,(car spec) max)
     ,body
     (setq ,(car spec) (+ ,(car spec) 1))))))
```

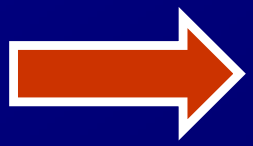
```
…(dotimes (i n)
     (setq sum (+ i sum))) …
                ↓
…(let ((max n) (i 0))
     (while (< i max)
       (setq sum (+ i sum))
       (setq i (+ i 1))))…
```

問題は?

マクロの問題: 変数の捕捉

- マクロが勝手に作る局所変数名とマクロを使っている場所にある変数名が重複する現象

```
(def max-element (a)
  (let ((max (- INFINITY)))
    (dotimes (i (length a))
      (if (< max (aref a i))
        (setq max (aref a i))))
    max)))
```



```
(def max-element (a)
  (let ((max (- INFINITY)))
    (let ((max (length a))
          (i 0))
      (while (< i max)
        (if (< max (aref a i))
          (setq max (aref a i)))
        (setq i (+ i 1)))
      max)))
```

Hygienic Macros [Kohlbecker+86]

- 変数名に関して安全なマクロ
  - テンプレート中で宣言された局所変数を自動的に付け替え

```
(define-syntax dotimes
  (syntax-rules ()
    ((dotimes (v n) body)
     (let ((max n) (v 0))
       (while (< v max)
         body
         (setq v (+ v 1))))))
```

パターン

テンプレート
(maxは付替)

```
(def max-element (a)
  (let ((max (- INFINITY)))
    (let ((max$1 (length a))
          (i 0))
      (while (< i max$1)
        (if (< max (aref a i))
            (setq max (aref a i)))
        (setq i (+ i 1)))
      max)))
```

eval: 動的なマクロ

- 式→計算 なる関数
  - 式の生成・計算は実行時
- 目的: 実行時情報を使ったプログラム生成

  例:
  - 対話的に入力された式を使って計算
  - ファイルに保存された関数定義を読み込み

evalの例: プログラム特化

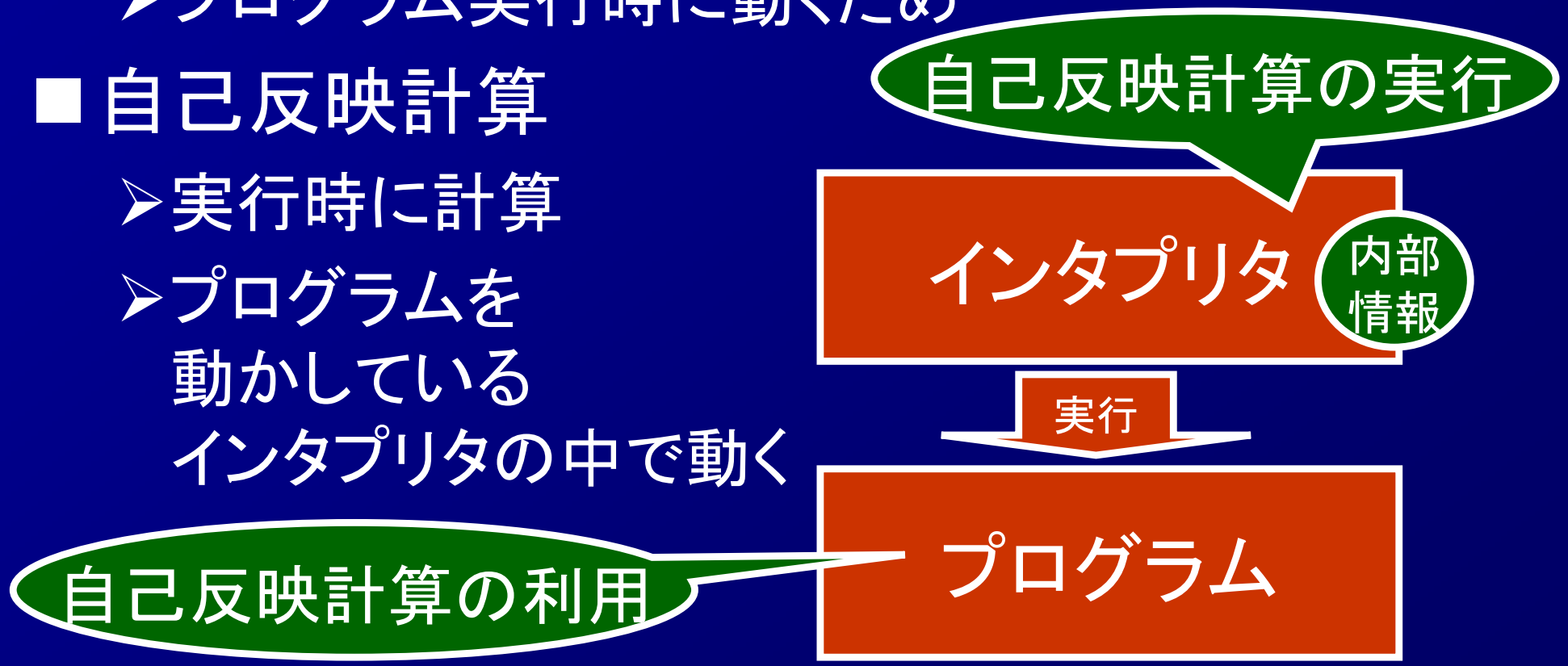
```
> def dp(n)
>  "def power"+n.to_s+"(x)"+("x*"*n)+"1; end"
> end
> dp(10)
"def power10(x)x*x*x*x*x*x*x*x*x*x*1; end"
> eval(dp(10))
> power10(2)
1024
```

evalの問題

- 遅い: インタプリタ実行
  - あるいは実行時コンパイル
  - 最適化目的のeval !?
- 危険
  - 実行前の保証(例: 型安全性)ができない
  - セキュリティ・ホールの源泉
  - 1ヶ所でもevalがあると全体が解析不能に

自己反映計算(reflection) [Smith84]
実行時情報を利用できるマクロ

- マクロの問題: 実行時の情報を使えない
  - プログラム実行時に動くため
- 自己反映計算
  - 実行時に計算
  - プログラムを動かしている インタプリタの中で動く

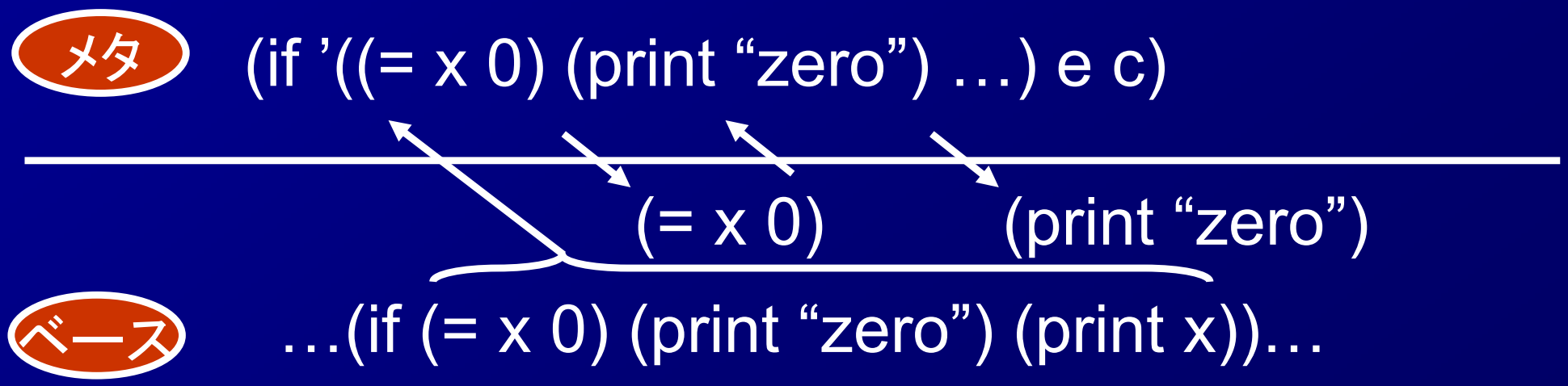


自己反映計算

- インタプリタ内部で動くユーザ定義の計算
  - 内部情報に触れる (例: 変数環境や継続)
- インタプリタの拡張を利用者プログラムが行える
  - 「この変数名は定義済みか?」
  - 「この変数定義を削除せよ」
  - 組み込み「メタ機能」を利用者定義関数として実現
    (例) boundp, throw/catch

自己反映計算の例

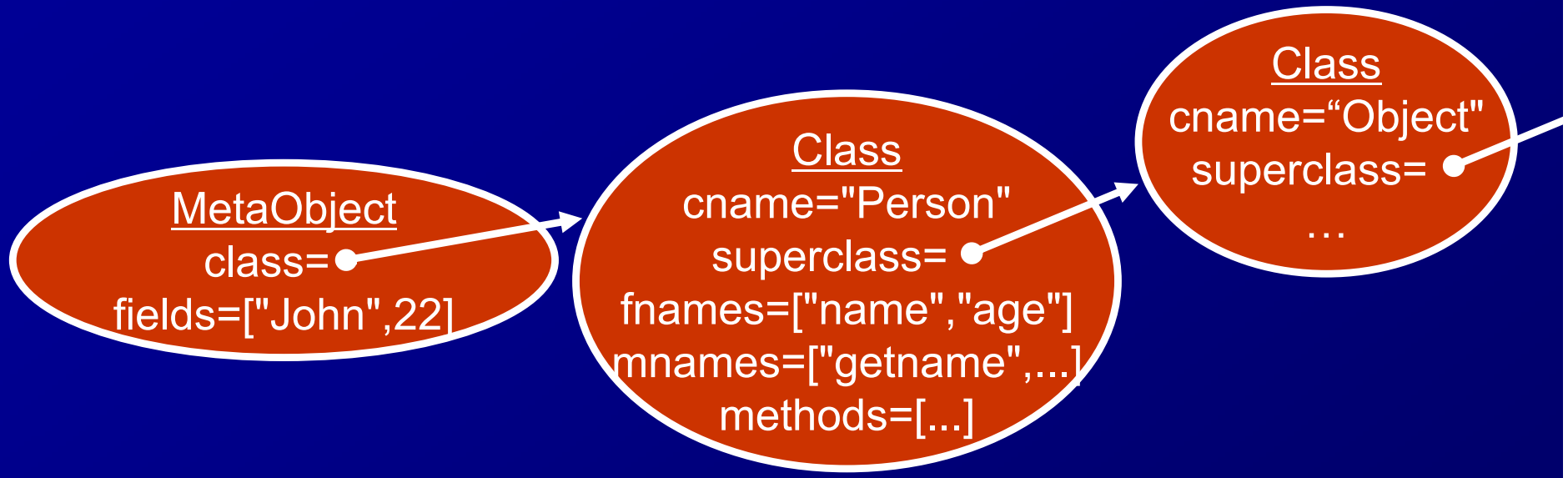
```
(define if
  (lambda reflect ((cond then else) env cont)
    (eval cond env
      (lambda (c) (eval (ef c then else) env cont))))))
;; efは全引数を評価する
```



オブジェクト指向言語のメタ機能

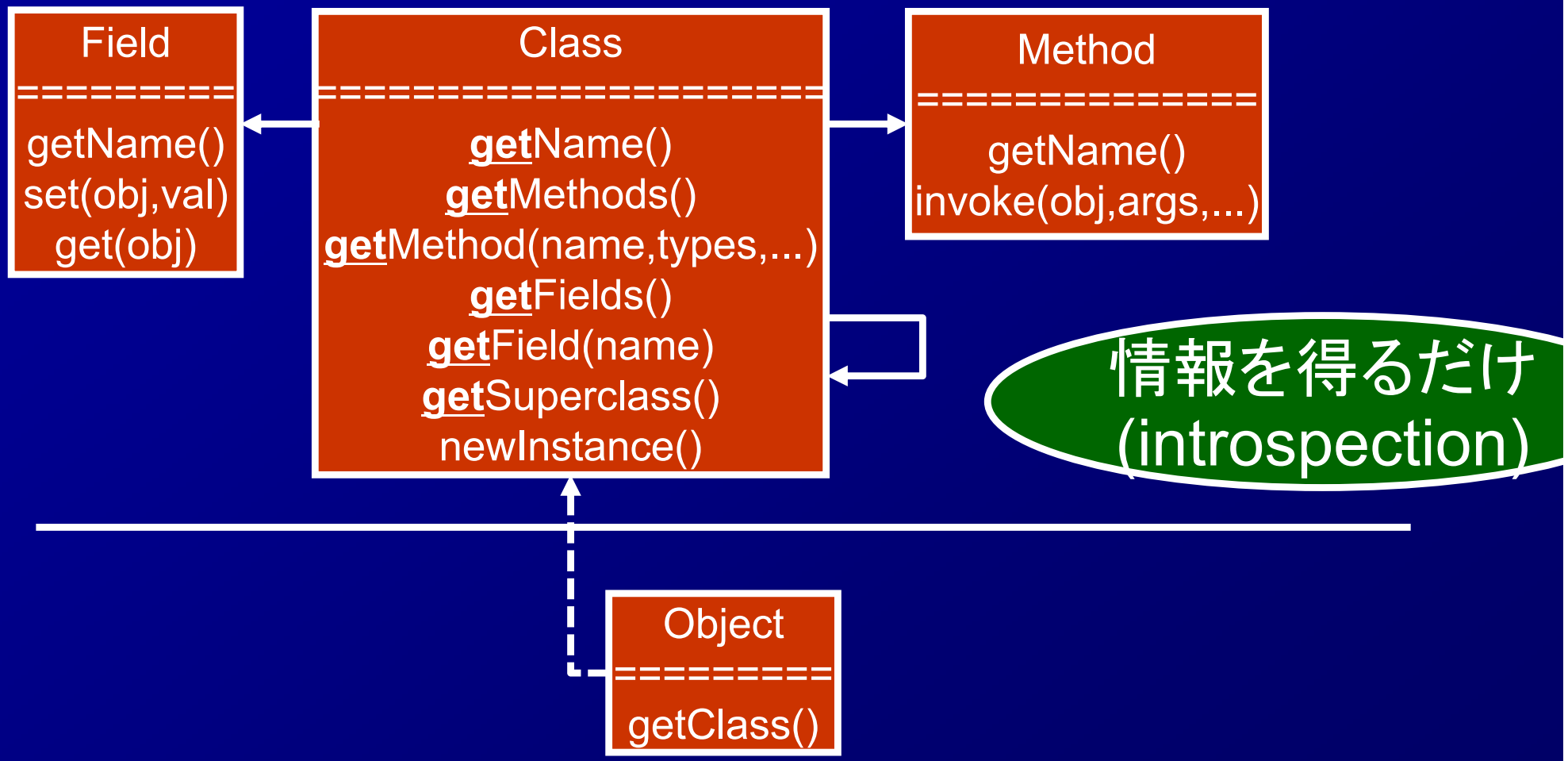
- 機能の例
  - あるオブジェクトが所属するクラスを調べる
  - あるクラスに定義されているメソッド・フィールドの一覧を得る
  - あるクラスの親クラスを調べる
  - ある名前のクラスのオブジェクトを作る
- 用途の例
  - オブジェクトの状態をファイルに保存・復元

メタオブジェクト:
メタ機能を提供するオブジェクト



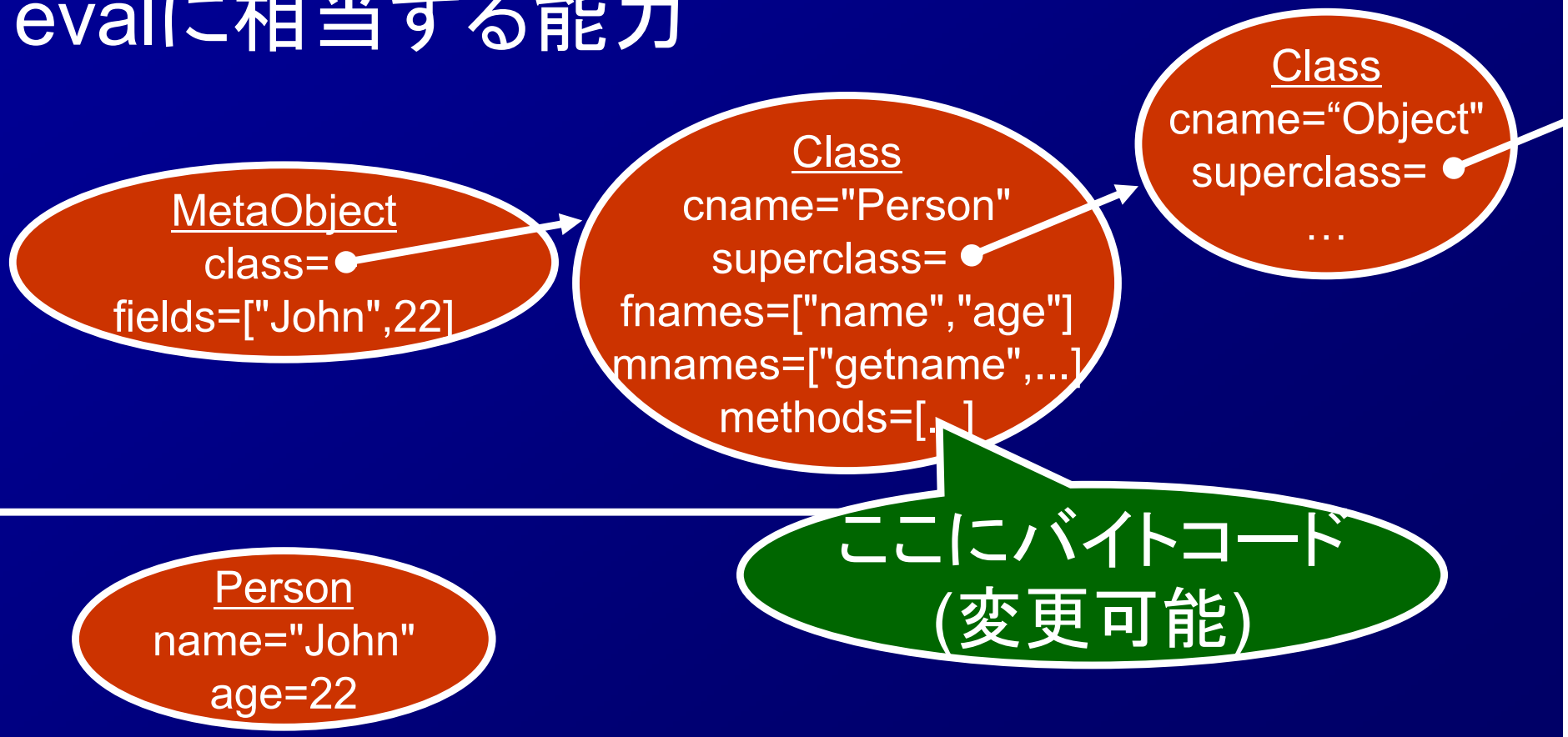
- あるオブジェクトが所属するクラスを調べる
- あるクラスに定義されているメソッド・フィールドの一覧を得る
- あるクラスの親クラスを調べる
- ある名前のクラスのオブジェクトを作る

Java Reflection



Smalltalkメタオブジェクト

- 処理系の内部情報を直接提供
  evalに相当する能力



CLOS Metaobject Protocol [Kiczales91]

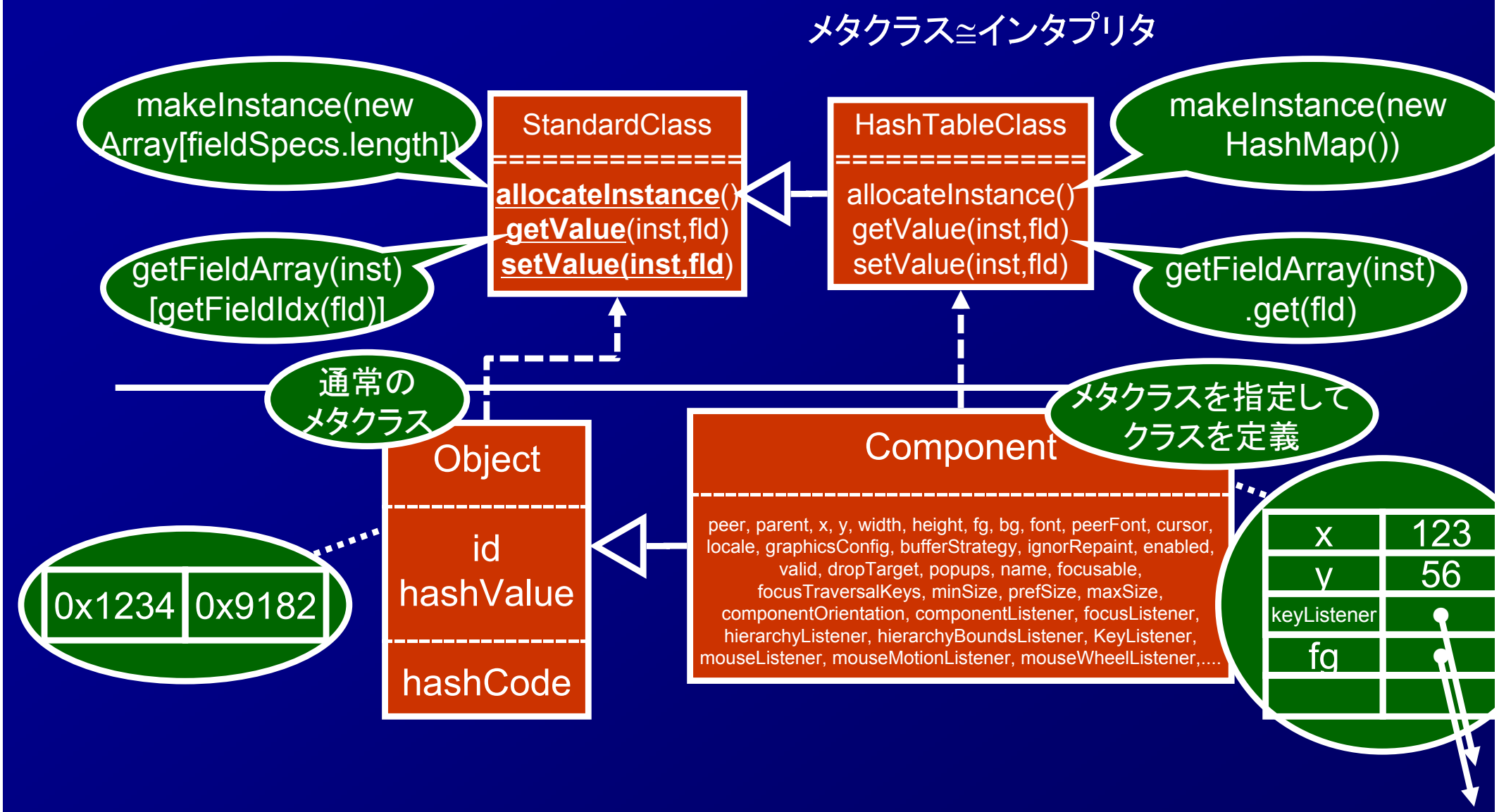
CLOS= Common Lisp Object System

利用者定義可能なメタオブジェクト

利用例:

- オブジェクトのフィールドの内部表現 (cf. スカスカのオブジェクト)
  - 通常: 配列 (読み書き O(1))
  - 変更: ハッシュ表・データベース連動・・・
- メソッド選択アルゴリズムの変更 (多重継承)
  - 通常: 親クラスを順序付け
  - 変更: 実行時の条件で順位変更

CLOS MOPによる自己反映計算



参考文献

[Kohlbecker+86] Kohlbecker, Eugene, et al. "Hygienic macro expansion." Proceedings of the 1986 ACM conference on LISP and functional programming. ACM, 1986.

[Smith84] Smith, Brian Cantwell. "Reflection and semantics in Lisp." Proceedings of the 11th ACM SIGACT-SIGPLAN symposium on Principles of programming languages. ACM, 1984.

[Kiczales91] Kiczales, Gregor. The art of the metaobject protocol. MIT press, 1991.