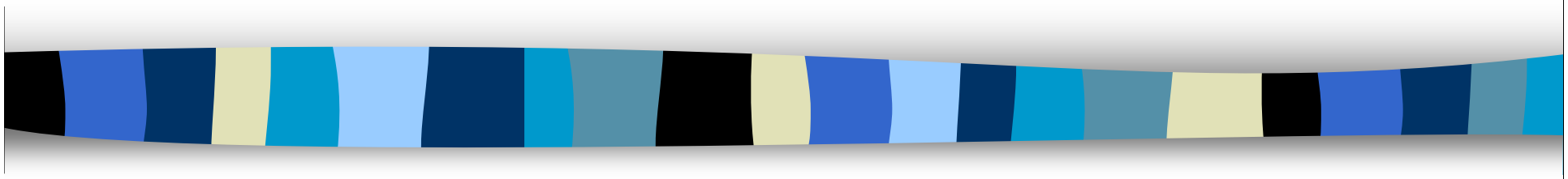
LispのISO標準規格 ISLISP処理系の研究開発



1999年 10月 13日

湯浅　太一　京都大学
梅村　恭司　豊橋技術科学大学
長坂　篤　　沖電気工業株式会社

目次

- ISLISP言語
- ISLISP処理系
- 高速かつ高い移植性を両立させる処理系方式
- 処理系の構成
- Lispオブジェクトの表現
- オブジェクトシステムの実装
- 他言語インタフェース
- 処理系の評価
- ISLISP検証システム

1. ISLISP処理系の設計目標

- 開発目的
  - 日本案(KL)をベースとするISLISPを普及を目的として，普及促進のために移植性に優れ，かつ高速な ISLISP処理系を開発する
- 処理系設計方針
  - 移植性（Portability）
    - Windows PC，Unix 等の種々のマシンへの移植の容易さ
  - 高速性
    - Common Lispに対する優位性の実証
    - 実用アプリケーションの開発に使用可能な十分な高速性
  - IS規格への完全な準拠
    - 規格普及のための，IS規格の解釈の規定・補強
    - ISLISP処理系がIS規格準拠であることを検証するシステム

ISLISP処理系依存仕様の規定

- Lisp基本機能
  - Bignumのサポート
  - 日本語文字のサポート
- オブジェクトシステム
  - クラスの再定義
    - 会話型プログラミング環境は Lispの特徴であり、クラス/メソッドの再定義機能は必須
  - メタクラスの実装:　実装しない
- 拡張機能
  - コンパイラ、インタプリタ
    - ISLISPでは、言語仕様から処理系に関する仕様は分離
    - Lispの特徴を生かすためにインタプリタを開発
  - Debugger その他ツール

2. 処理系の方式

Portabilityと高速性を両立させた Lisp処理系

Portable Standard Lisp(PSL)
- Utah大学
- PCL(SYSLSIPで記述)が生成する抽象アセンブリプログラムをマクロ(c-macro)によって目的マシンコードに変換

Kyoto Common Lisp(KCL)

京都大学
等価なCプログラムへ変換
C処理系が必要

Tachyon Common Lisp

沖電気工業
仮想32bit RISCプロセッサ＋抽象アセンブラの上に開発
Pentium対応では設計変更が必要

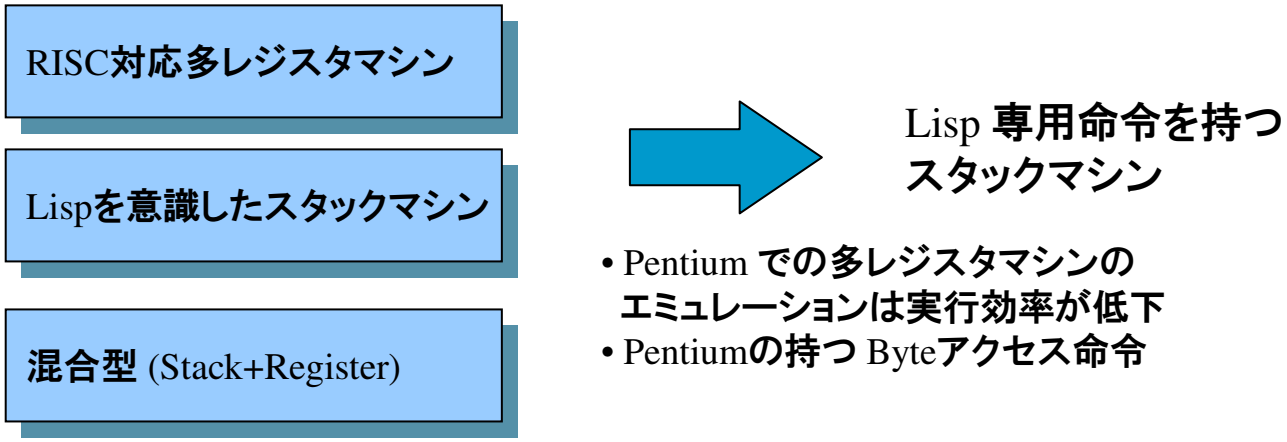
仮想マシン(Byte Code)方式を採用

高速性

Lisp Objectの設計
高速な Object System

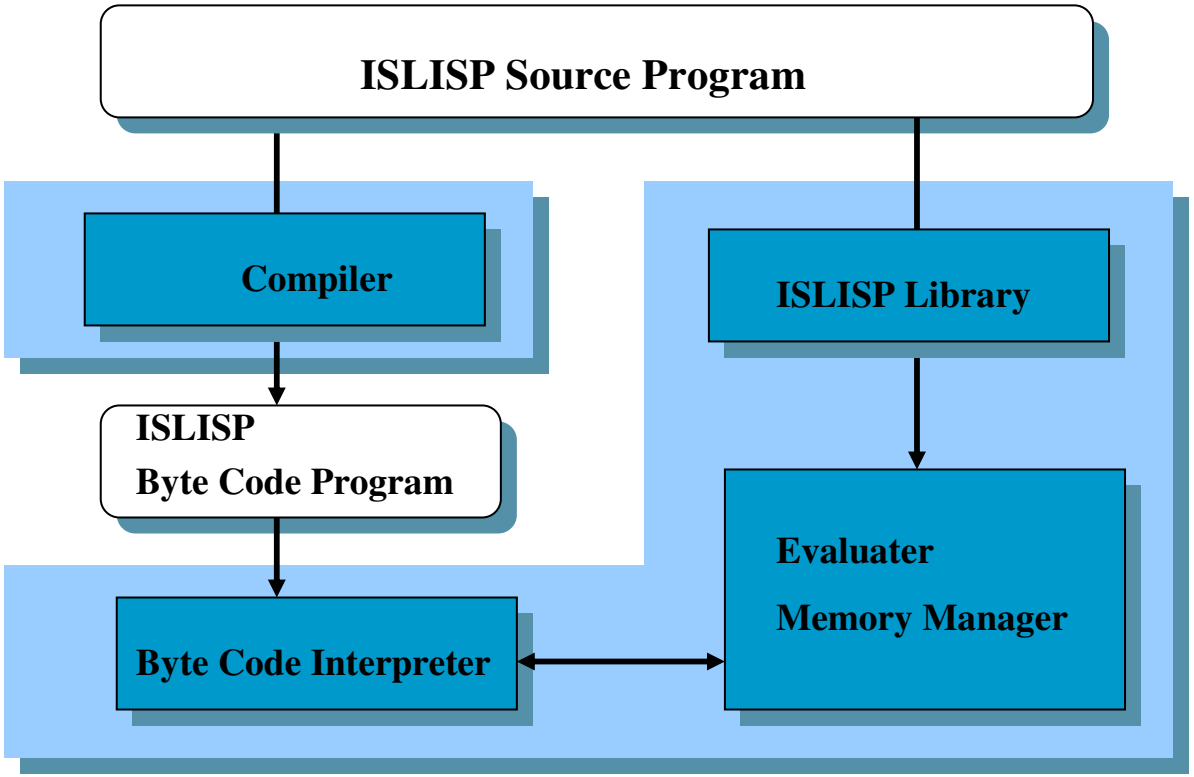
3. 仮想マシンの(Byte Codeマシン)構成

- **仮想マシンのアーキテクチャ**



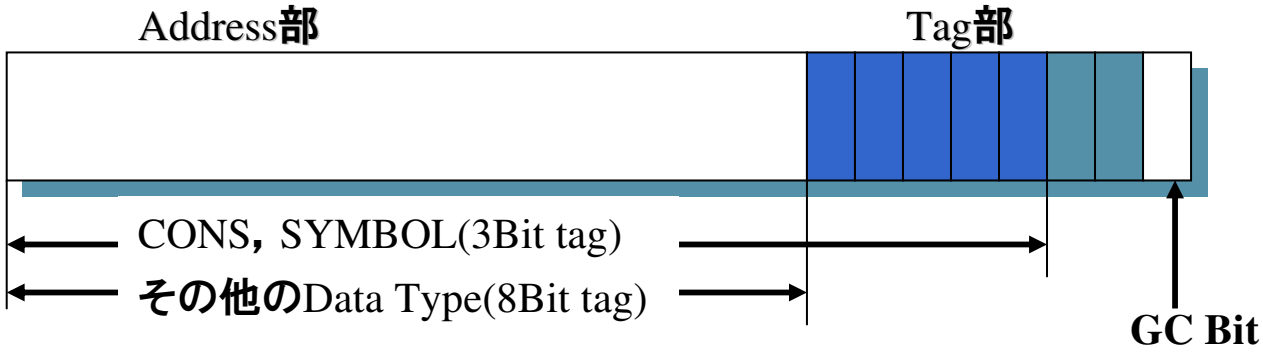
- Byte Code **仕様**
  - Opcode：1～2 Byte，Operand：1，2，4 Byte
  - **スタックマシン上でLispプログラムを効率よく実行可能な命令体系**
  - Byte Code
    - **スタック操作，変数アクセス，Lisp オブジェクト即値**
    - **関数呼出し**
    - **分岐**
    - **特殊関数**

処理系の構成


ISLISP Source Program
Compiler
ISLISP
Byte Code Program
Byte Code Interpreter
ISLISP Library
Evaluater
Memory Manager

4. Lispオブジェクト

- Lispオブジェクトの設計方針
  - 高速な実行性能
  - アドレス空間の制限がない
- Lisp オブジェクト方式
  - Pointer Tag
  - 3bit-8bit 可変長タグ



CONS，SYMBOL

3Bit tag(CONS:000, SYMBOL:100) と実体領域を8Byte境界で格納

→ オブジェクトの実体に直接アクセス可能

即値(Fixnum，character)

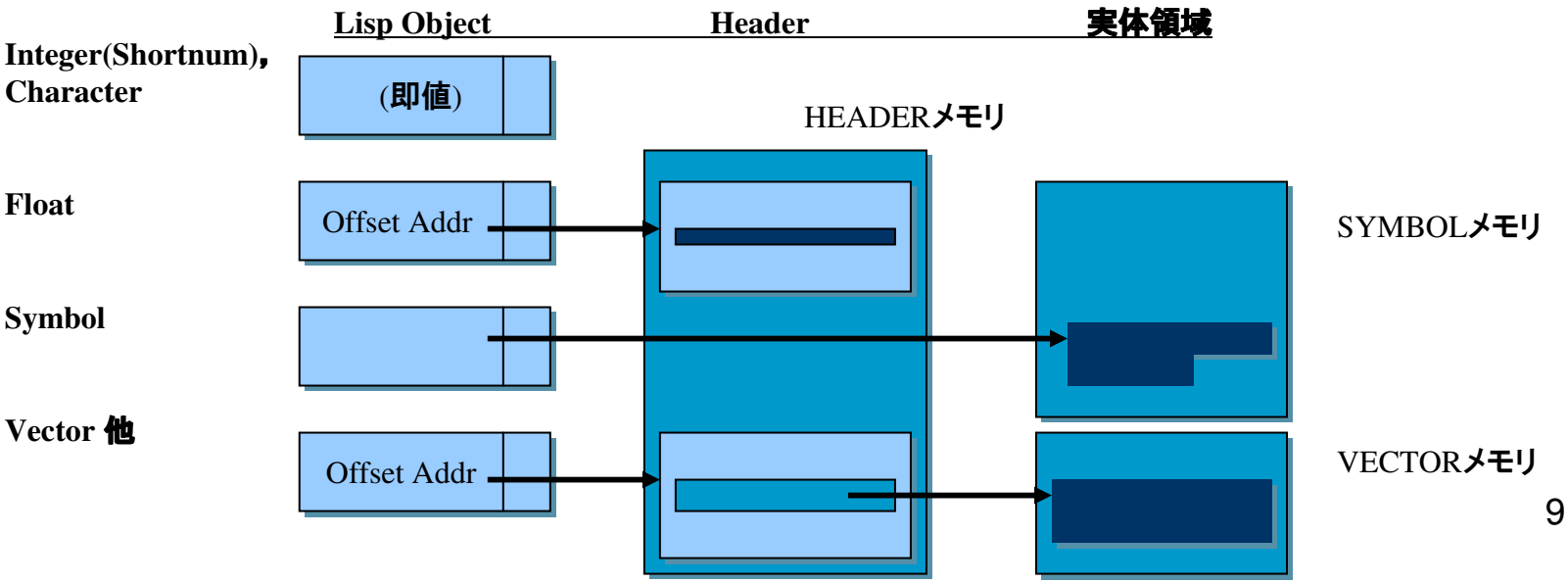
Fixnum は24Bit整数型とし，アドレス部に値を格納

その他のタイプ

アドレス部に，HEADERメモリのオフセットを格納し，間接アドレス

5. メモリ管理ーLispメモリ

- タイプ毎に次の5種類のメモリブロック(Lispメモリ)を用意
  - CONSメモリ
  - SYMBOLメモリ
  - HEADERメモリ　　Header
  - VECTORメモリ　　Vector，Array他
  - STACKメモリ　　Lispスタックとして使用



メモリ管理– GC(Garbage Collection)

- GC方式
  - 一括型 GCであるMark&Sweep GCを採用
- メモリ圧縮
  - VECTORメモリ:　　圧縮
  - CONS、SYMBOL、HEADERメモリ:　　圧縮しない
- CONS、SYMBOL、HEADERメモリでは非圧縮のため、アドレスが移動せずGCが高速となる

6. オブジェクトシステム

- ISLISP オブジェクトシステムの仕様
  - CLOSのサブセット
    - 包括関数(総称関数)ベース
    - 多重継承の制限
    - MOP(メタオブジェクトプロトコル)の制限
- 処理系依存機能
  - メタクラスのサポート
- 拡張機能
  - クラス再定義、メソッド再定義機能
    - インタラクティブなプログラミング環境はLispの特徴であり、クラス及びメソッドの再定義機能は必要

オブジェクトシステムークラス再定義の課題

- クラス再定義における課題
  - 実行時に高速になるようにする
  - 再定義におけるインスタンス，サブクラスの効率的な変更
  - 以前のクラス定義で生成されたインスタンスの効率的な扱い
- インスタンス，サブクラスの変更方法

一括型
- 再定義時に全インスタンス，サブクラスを変更
- 再定義は遅いが，インスタンス操作は速い
- 生成したインスタンス，サブクラスの管理が必要

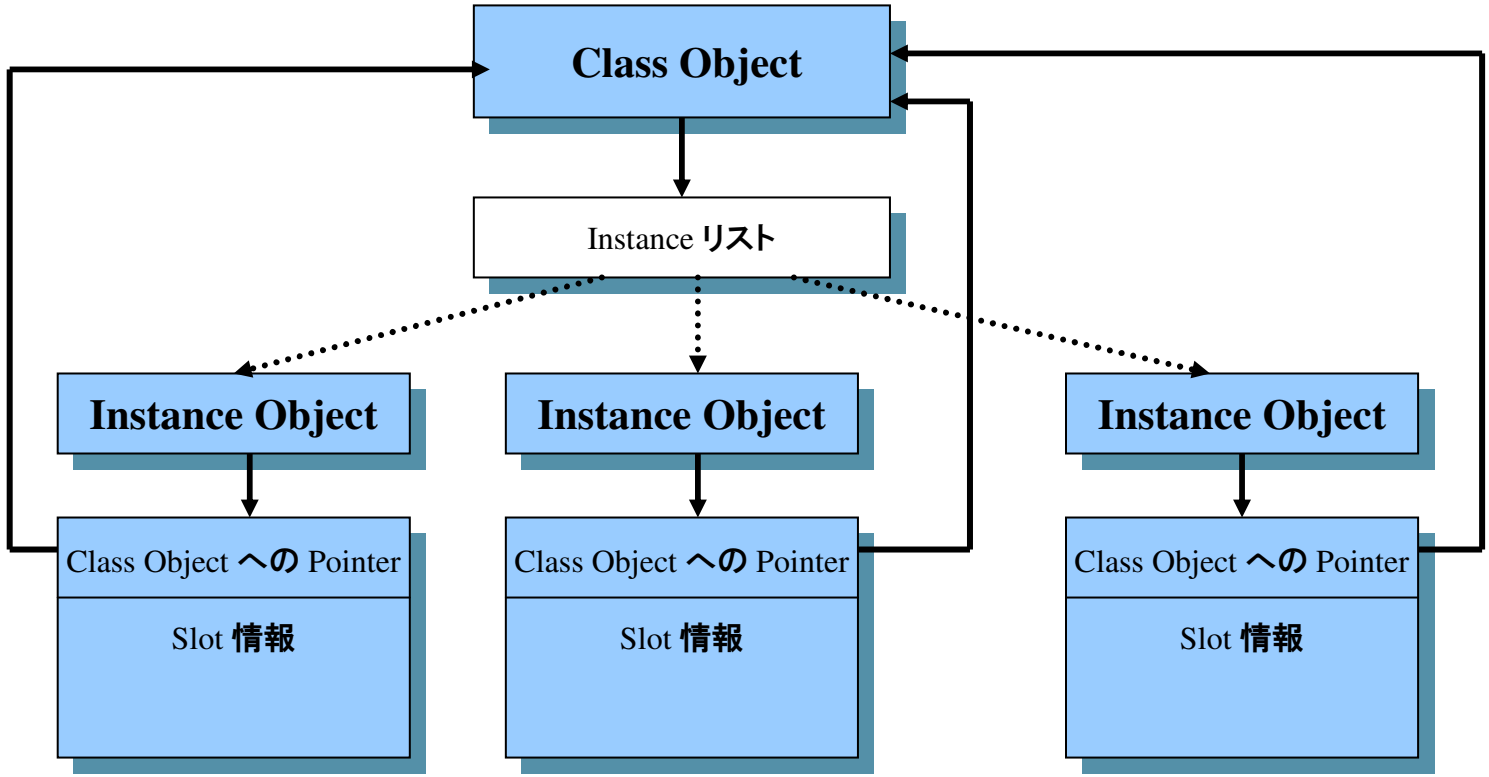
逐次型
- 変更されたクラスやそのインスタンスに対する操作が行われたときに変更
- 再定義は速いが，インスタンス操作は遅い
- 生成したインスタンス，サブクラスの管理が不要

実行時(インスタンス操作時)の高速化のため一括型を採用

インスタンスの管理に Weak Pointer を使用

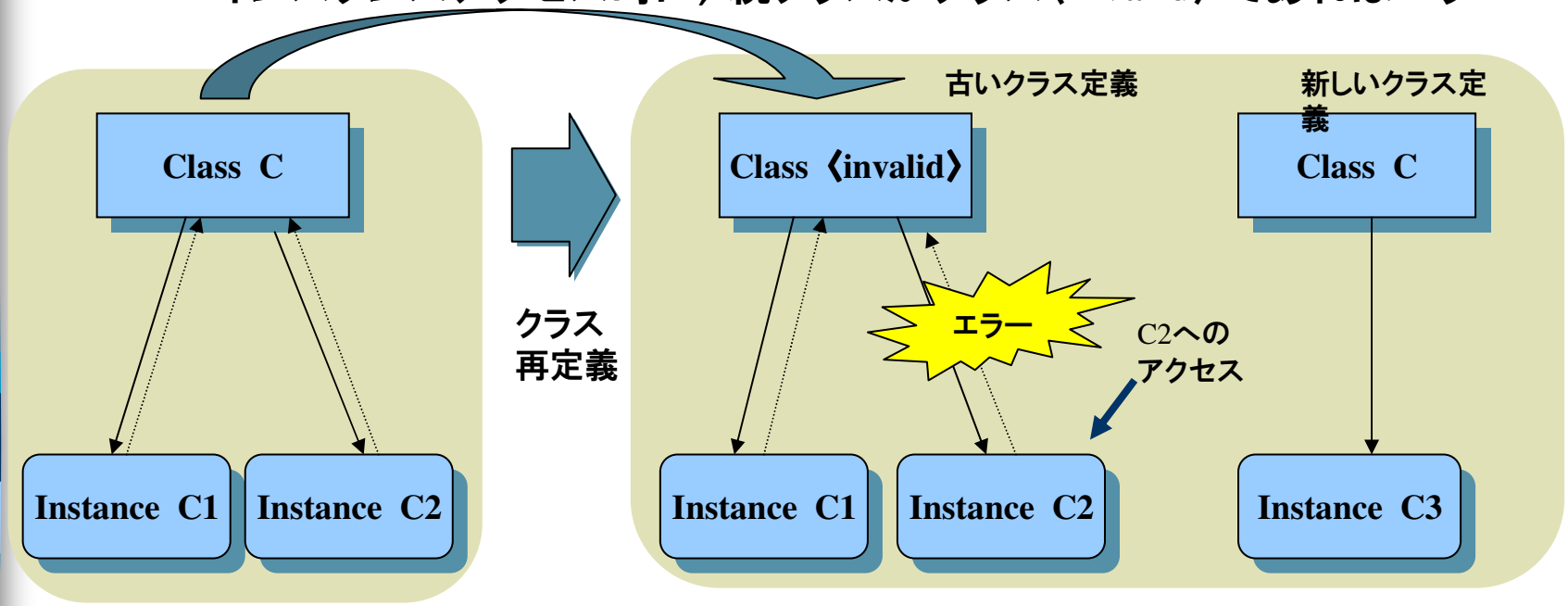
オブジェクトシステム－実装方式

- クラスのインスタンスを一括管理
- インスタンスは Weak Pointer で管理
- インスタンスは親クラスへの Back Pointerを持つ



クラス再定義と〈Invalid〉クラス

- インスタンスをWeak Pointerで管理
- クラス〈invalid〉を導入
  - 古いクラス定義のクラスを〈invalid〉に変更
  - インスタンスアクセス時に，親クラスがクラス〈invalid〉であればエラー

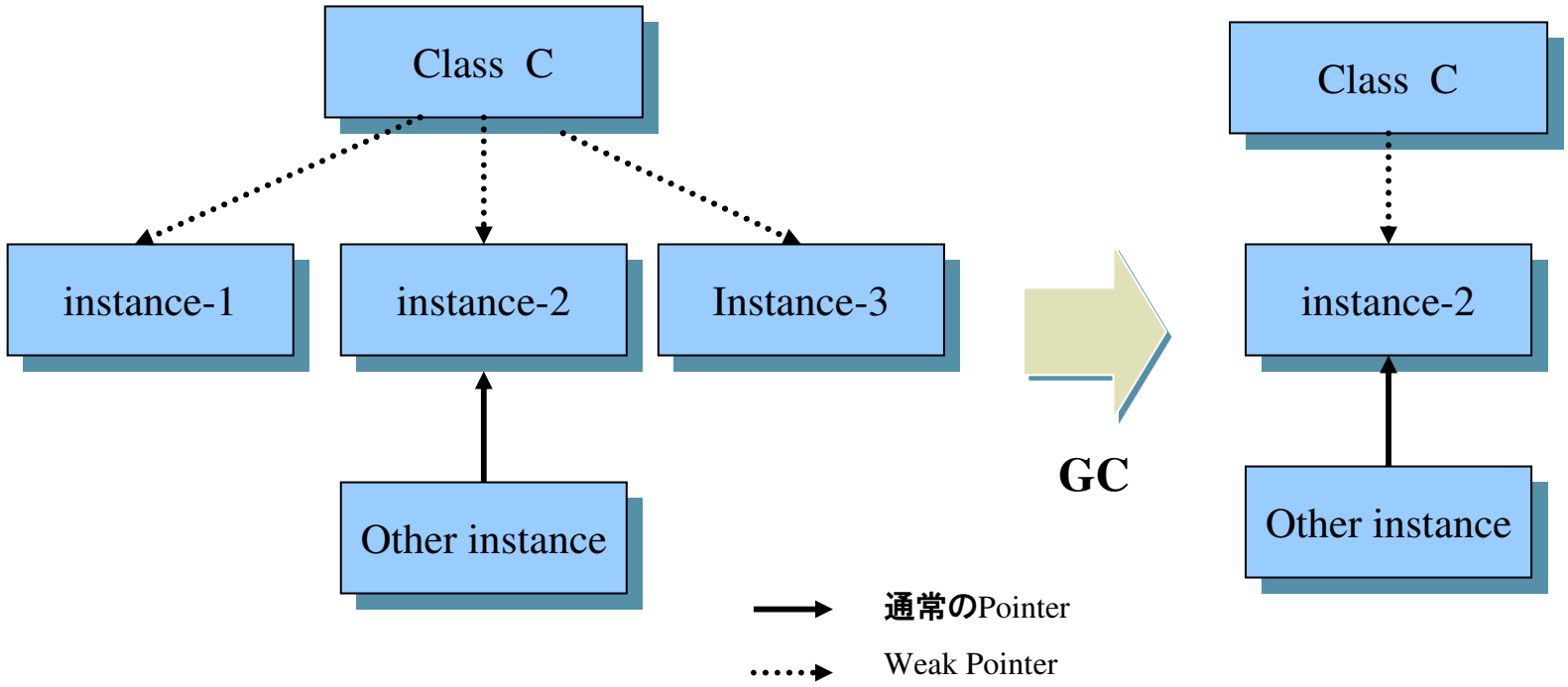


クラス〈invalid〉の使用例

```
ISLisp> (defcalss  circle (figure)                    ; クラス〈circle〉を定義
                ((radius  :accessor circle-radius
                          :initarg  raidus)))
CIRCLE
ISLisp> (defglobal  fig1 (create  (class circle ) ‘radius 1)) ; 〈circle〉のインスタンス
FIG1                                                  ; fig1生成
ISLisp> (circle-radius fig1)
1
ISLisp> (defclass circle (figure)                     ;クラス〈circle〉を再定義
                :initarg radius
                :initform  1)))
CIRCLE
ISLisp> (circle-radius fig1)                          ;インスタンス fig1へのアクセスは
> Error at CIRCLE-RADIUS                              ; エラーとなる
> No applicable method
```

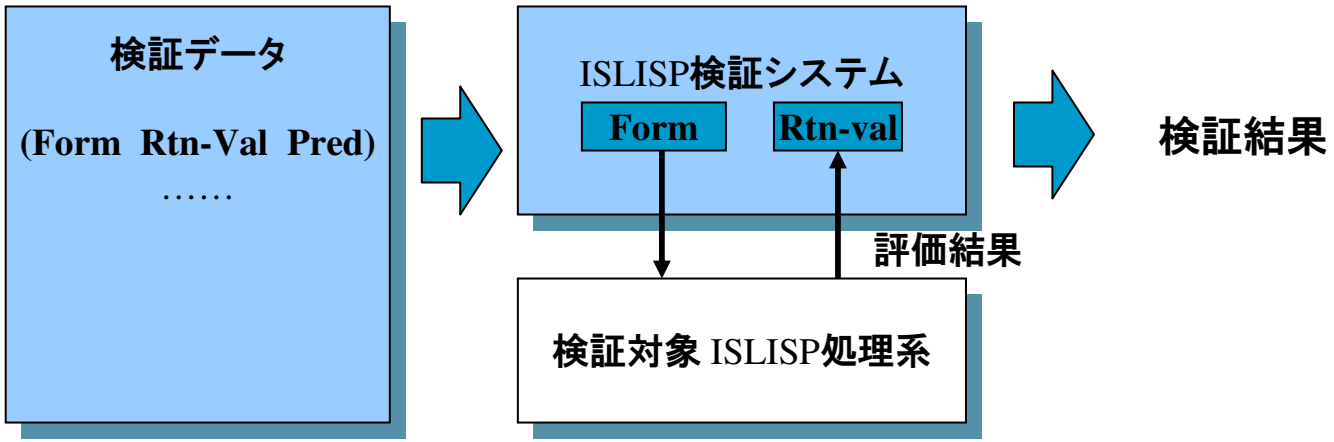
Weak Pointer

- ウィークポインタのみで参照されているLISPオブジェクトはゴミ(GCで回収)
- クラスから全インスタンスをウィークポインタで管理



7. 検証システム

- 与えられたISLISP処理系が ISLISP規格にCompliantであることを検証するシステム
- 仕様の検証に十分な検証データ(入力 ISLISPテキストとそのとるべき値)を与え、処理系の評価結果が正しいか否かを検証
- 検証システムの特徴
  - 正常な実行に加えて、エラー実行の記述が可能
  - 処理系が追加したエラーもサポート可能
    処理系定義のエラーは、<serious-error> のサブクラスか否かを判定



検証システム－記述機能

- **正常実行**

  (form return-value predicate)　　((cons 1 2) (1 ． 2）equal)

- **エラー**

  ($error form error-name)　　($error (car 3) $ERR_NotCons)

- **単なる実行(前準備他)**

  ($eval form)　　($eval (defglobal x 10))

- **述語関数のチェック**

  ($predicate func-name true-type*)　　($predicate consp $cons)

- **引数のデータ型チェック**

  ($type func-name (true-type*) error-name args*)

- **引数の個数チェック**

  ($argc func-name reqs opt rest)　　($argc eq 2 0 0)

- **メッセージ表示**

  ($echo message)

8. 他言語/システムとのインタフェース

(1)双方向他言語(C/C++)インタフェース機能
- OS毎にオブジェクトファイル形式が異なり、移植性に課題

(2) Java VM上のISLISP処理系
- 実用的な性能の達成が困難

(3) ISLISP処理系上のJava VM
- 実用的な性能の達成が困難

(4) ISLISP処理系の DLL化と，WWWブラウザのPlug-In化
- ユーザインタフェースとして WWWブラウザを使用可能

(5) ISLISP処理系から他DLLを呼び出し機能

(6) プロセス間通信による連携機能

(7) OMG CORBA準拠 ISLISP IDLコンパイラの開発

⇨ (4)，(5)，(6)，(7) をサポート予定

9. 評価ー性能評価

- Gabriel Benchmarkによる評価
  - マシン　　Pentium Ⅱ 133MHz/Windows95
  - Open Lisp　　Christien Jullien 開発の ISLISP処理系

（単位：秒）

	Open Lisp	Interpreter	Compiler
Tak	0.270	0.840	0.233
Stak	1.373	0.412	0.570
Ctak	0.975	0.316	0.370
Takl	5.507	1.373	1.930
Takr	0.865	0.226	0.370
derivative	1.415	0.617	1.040
dderivative	1.539	0.627	1.100
div2(iterative)	1.648	0.452	0.550
div2(recursive)	1.934	0.523	0.650
browse	19.446	4.147	5.460
destuctive	2.019	0.687	0.970
init-traverse	12.812	4.682	5.400
run-traverse	90.848	25.381	39.050
FFT	1.044	0.206	0.410
puzzle	17.276	4.312	6.630
triangle	195.798	70.753	80.190

性能評価

- ISLISP処理系性能比較
  - Open Lispより 約1.42倍高速
  - Open Lispでは以下の問題があり、処理系自体の性能としては本処理系がさらに高速
    - Bignum をサポートしていない
    - 一部特殊形式で引数個数のチェックを行っていない
- 言語仕様による性能評価
  - Common Lisp (Lucid Common Lisp)より約5倍高速

評価ー移植性

- **移植済みのマシン/システム**
  - Windows95,98/Pentium
  - WindowsNT4.0/Pentium
  - Solaris2.x/SPARC
  - SUN OS4.**x**/SPARC
  - HP-UX 10.x～/HP-PA
  - Linux/Pentium
  - Free BSD/Pentium
- **移植性**
  - Byte Code**マシン**
  - **全てを**ANSI C**で記述（浮動少数点数の**Overflow/Underflow**検出を除く）**
    - **1つの実行ファイルとなり，配布とインストールが容易**
    - **必要初期メモリが小さくコンパクト (処理系サイズ：** 205KB～400KB)
    - **将来の**Plug-In**化対応が可能**

まとめ

- ISLISPの普及を目的として，高い移植性と高速性を持つ ISLISP処理系を開発
  - 移植性と限られた時間ないでの開発からByte Codeマシン方式採用
  - 効率的な実行を可能にするLispオブジェクト，オブジェクトシステムによって高速性を達成
- ISLISP検証システムの開発
- 既存処理系（Open Lisp）よりも 1.42倍の高速性を達成
- Windows PC，種々のUnixマシンの移植
- 処理系の公開
  - http://www.okilab.com/islisp
- 今後の課題
  - 他言語・システムインタフェース機能の実装
  - 開発環境の充実