プログラミング言語処理系論
Design and Implementation of Programming Language Processors

佐藤周行
（情報基盤センター/電気系専攻融合情報学コース）
schuko@satolab.itc.u-tokyo.ac.jp

講義の目的

アルゴリズムを表現するためのプログラミング言語でかかれたプログラムを具体的なアーキテクチャに向けて最適なコードに変換することは、従来からコンピュータサイエンスの中心的な話題であったが、プログラミング言語の抽象度とアーキテクチャの複雑度の両方が飛躍的に増している現在、その重要度と問題の複雑度は飛躍的に増加している。
本講義では、プログラミング言語をどう設計するか、そこで表現する概念をどう実装するか、その時に処理効率をあげるにはどうしたらよいかを理解することを目標とする。

今日の予定

- まぁ、1時間くらいかな
- この授業を取る（受講する）べきかどうかの決定をするための情報を与えるのが目的です
- 進行予定
  - 授業の進行予定
  - 背景の説明
  - プログラミングの変遷
  - プログラミング言語の歴史

講義予定

ここまで聞いて、少し手を動かすと、
自分でプログラミング言語を設計したくなります（たぶん）

1. 授業導入とプログラミング言
   4/14, 4/21
2. プログラミング言語の定義の手法(2回)
   4/28, 5/12, 5/19
3. 特別講義 NetFrontのソフトウェアアーキテクチャ
   5/19 特別講師　山上俊彦（アクセス）
4. VMと実行時環境(1回)
   5/26
5. プログラム解析と最適化(3回)
   6/2, 6/9, 6/23
6. 進んだ話題（Type, Effect, Verification）
   6/30, 7/7, 7/14

具体的にはどういう授業か

- プログラミング言語の処理系についての話です
- 処理系については、最近の話題は「最適化」に集中しています
- プログラミング言語については、最近出現してきた新しい計算環境でのプログラミング方法論と絡めての議論がさかんです
  - 分散環境→Cloud→…
  - XML →Web Services→…

具体的にはどういう授業か(II)

- 最適化は、プログラミング言語の中心的な話題です
  - (LO)新しいアーキテクチャの出現にともない、特定のパターンを高速に処理することができるように(特定のパターンしか高速に処理することができなく)なってきた
  - (HI)プログラミング言語の抽象度の高まりにつれて、「抽象的な」プログラムを具体的なコンピュータ上で実行可能なコードに変換する時の自由度が高まってきた

超高級言語の例

- Matlab
- Mathematica
- …

これとは別に

- データ処理量が爆発するにつれて、データ処理のための簡易言語を設計することがペイするようになりました
  - スクリプト言語の設計
  - PERL, PHP, …
- スクリプト言語は往々にして「いい加減なデザイン」に基づいています。
- スクリプト言語は、そのとっつきやすさから、ユーザが多くつくようになります。
- ユーザが多くなると、実装を再デザインして「まともな」プログラミング言語にすることがペイするようになってきます。

そこで

- プログラミング言語の設計の「作法」を覚えておくのは、悪くない
- 「よいデザイン」「よい実装」についての直観を養うことで、たとえばスクリプト言語のスタートアップを効率的にできる

具体的にはどういう授業か(III)

- クラシックなコンパイラとはどういうものか？

- 典型的な教科書
  Compilers Principles, Techniques, and Tools
  A. Aho et.al.
  ISBN 0321547985

クラシックなコンパイラの教科書では

- こっちのほうが年寄りに
  は有名

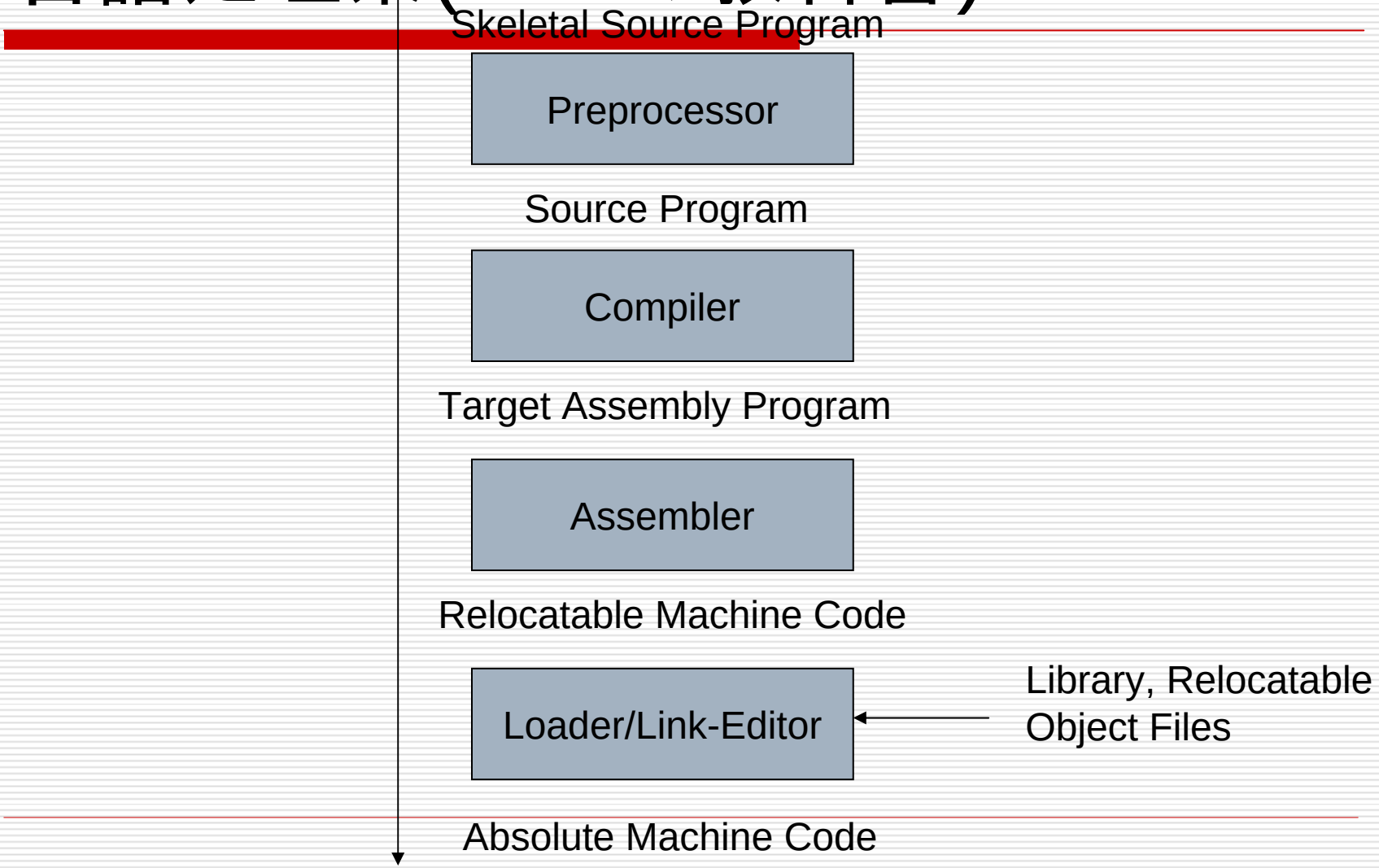
日本語の教科書

- 労作です

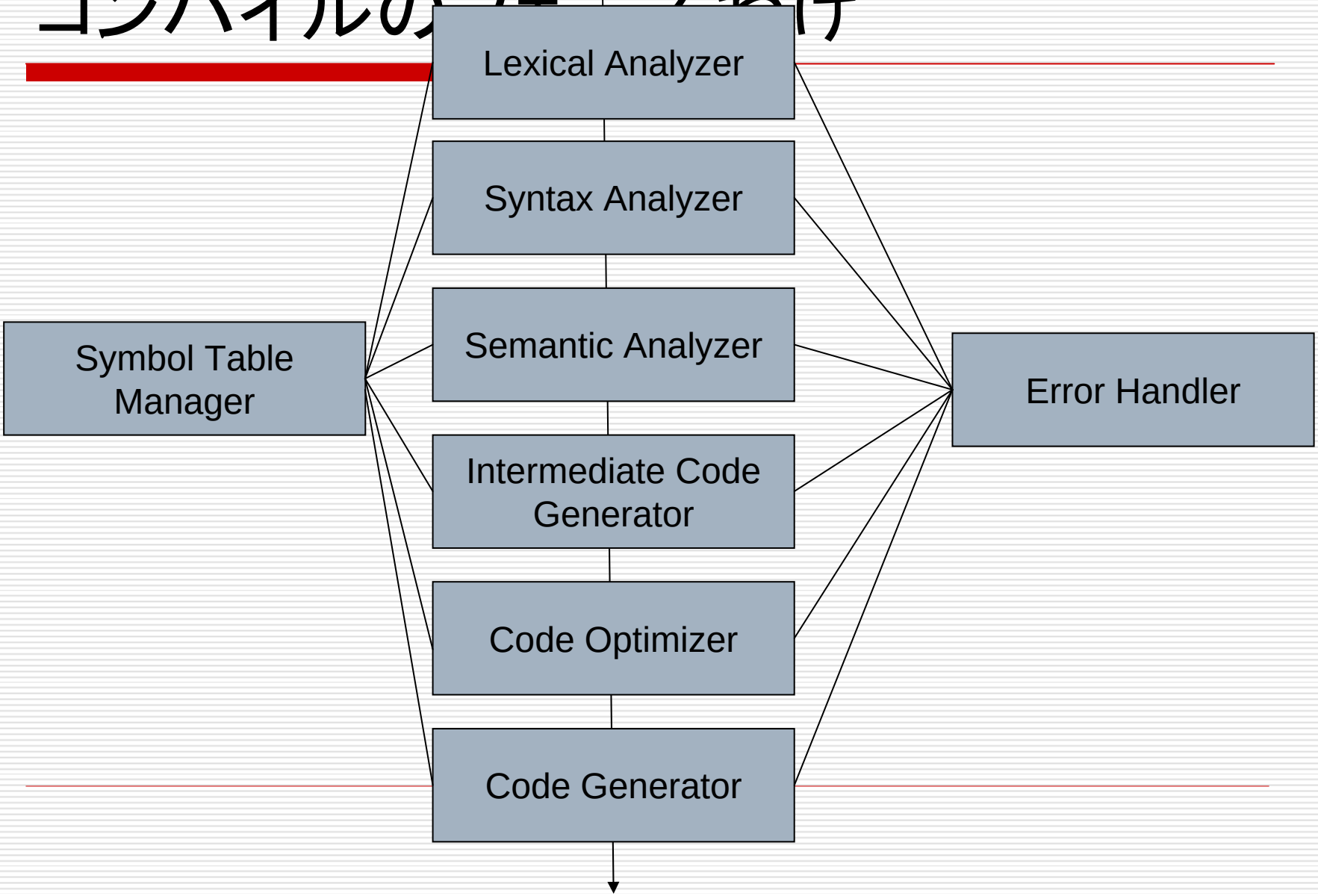
中田育夫
コンパイラの構成と最適化

出版社: 朝倉書店
(1999/09)
ISBN-10:
4254121393
ISBN-13: 978-
4254121391
発売日: 1999/09

言語処理系(Ahoの教科書)



コンパイルのフェーズわけ



Modern Compiler Construction

- 現在では、(前にも述べたとおり)研究のほとんど、コーディングのほとんどが「最適化」に集中しています。

- Syntax Analysis/Semantic Analysisについては自動化が進み、ここをまじめに論じる人はいません
  - 知らなくて良いわけではない
  - 情報系の学部なら3年くらいの教材のはず
  - 一昔前までは大学院の入試によくでた

最適化にフォーカスをおいた教科書

- 中田本
- クラシックなものはこれ

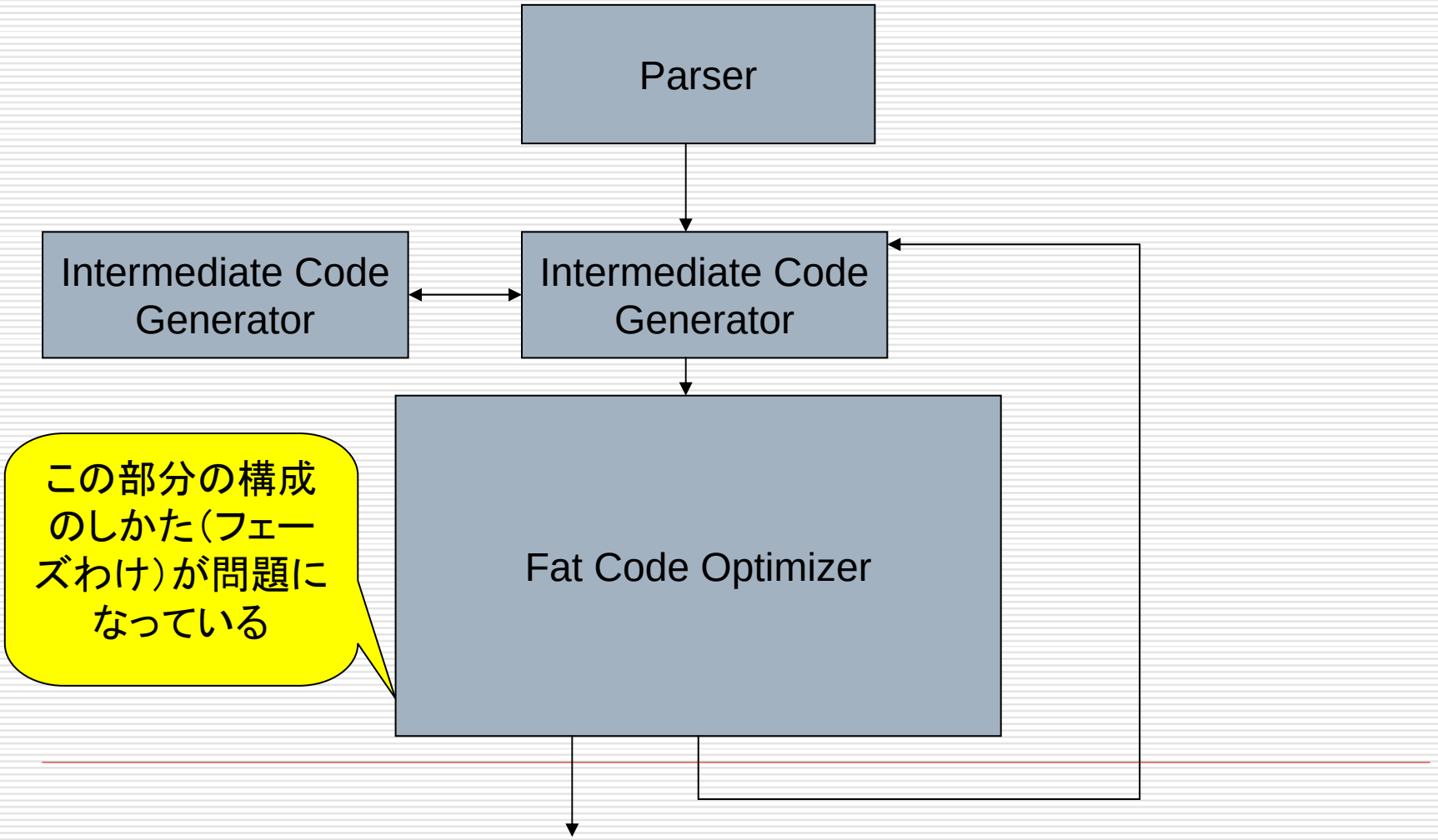
Muchnick, Steven S.

1997 Academic Press

Advanced Compiler Design and Implementation

ISBN 1-55860-320-4

Modern Compiler Construction



Parsing (I)

- ParsingをSyntax AnalysisとSemantic Analysisにわけて、それぞれに効率的な手法、自動化の手法を追求したのは1970年代の話です。
- LL(1)やLALRといった言語のクラスはParsingを楽にできる観点からCFGの部分クラスとして研究されました。(Chomskyもびっくり)

Parsing(II)

- プログラミング言語を定義するにはそれほど複雑な文法は必要ないというのが皆の共通の理解でした。
- C++がその常識に挑戦しました （文法のセンスを無視して機能を詰め込みすぎた結果のような気がします）
- 現在、C++は規格自体がメルトダウンしはじめているような感じがします
- C++の文法は(LALR＋アクション)ではかけないことがわかっています。Parsingは難しい
- Parser専門の会社があるほどです (http://www.edg.com/)

Parsing (III)

- でも、まぁ、おおまかな流れとしては、

- ParsingはCompilerでの中心的な話題からは引退しました。
- この講義ではLL(1)とかLALRについての具体的な説明をすることはしません
  - 繰り返しになりますが、情報系の学科の出身なら、すでに学んだはず（覚えているかどうかは…）

- でも、Parserをかけないと、自分で言語処理系を書けませんから、たとえば「打倒PHP」とか、「打倒Ruby」と考える人はParserの書き方は学習しましょう

Parsing (IV)

XMLのパースくらいはできるようになりましょう

- Regular Expressionとその受理オートマトンを含む (DTDの処理に必要)

- XMLの定義が読めるようになりましょう
  - BNF記法
  - 定義はここです
    http://www.w3.org/TR/2006/REC-xml11-20060816/

閑話休題

- 少し、個別の話題に足をつっこみすぎました。
- プログラミング言語とそれをとりまく環境の現状についての話に戻ります

プログラミングの変遷

- プログラミングの目的の変遷
- プログラミングの価値の変遷
- 目的と価値が決まれば、プログラムを何を使って表現するかの手段(プログラミング言語)の方向が決まる

プログラミングの目的の変遷

- 数値計算
  - 計算機の開発の driving force
  - 自然現象のシミュレーションは科学、工学、軍事において重要度は昔も今も変わっていない
  - HPCとスーパーコンピュータの隆盛（今は…）
- ビジネスでのプログラミング需要の増大
  - DB

プログラミングの目的の変遷

- システムプログラミングの需要
  - コンピュータのためのプログラミング
  - メモリシステムのモデル化などの成果
  - ネットワークプログラミングはここに入るだろう
- 分散環境とクラウド上でのプログラミングの隆盛
  - ネットワークプログラミングの上のレイヤ
  - Embedded Systemとともに有力な場所
  - ターゲットの激変
  - Web環境でのプログラミング
    - サービスの概念　→　SOA
    - コードの移動
    - HTML,XML（数値や文字列だけではなく、より大きい文書やサービスを扱うプログラム）

プログラミングの目的の変遷

- オブジェクトの発明
  - Alan Kay 1970ころから (Smalltalk 80)
  - 処理のパッケージ化とAPIの定義による抽象化
- より複雑なプログラミングへの対応
  - 複雑なオブジェクトへ
  - 複雑な継承と複雑なパターンへ

プログラミングの目的の変遷

- 今では、ほとんどのプログラミング言語が「オブジェクト」の概念を言語内にもっています(Cは別)

Fortran 2003

プログラミングの価値の変遷

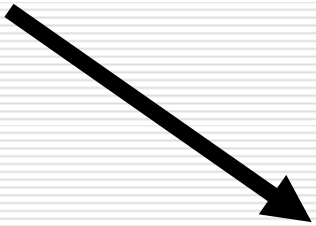
- 高速性から高速性＋正しさへ

- 高速性は常に「善」
  - 高速実行のためのコンピュータ
  - 高速実行のためのアルゴリズム
  - 高速実行のためのコンパイラ

  - 互いに影響しあう

高速実行のためのコンピュータ

Cluster System
(Dedicated)



Cluster System
From COTS

COTS: Commercial Off-the-Shelf

高速実行のためのコンパイラ

- プログラムの可読性、移植性を高めるには、「プログラムは素直に書く」ことが基本

（それ以前に速いアルゴリズムが必須）

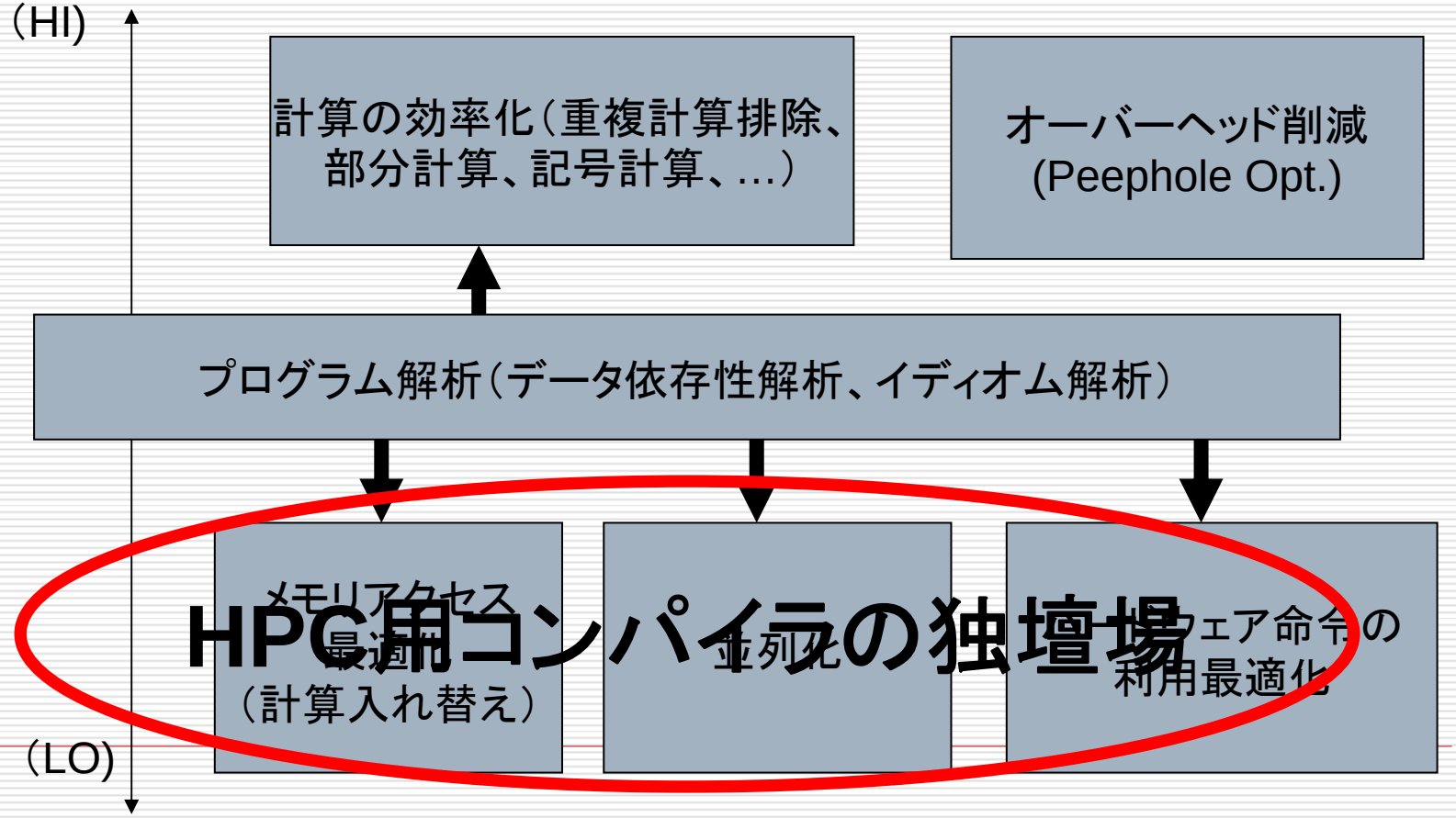
  - 「最適化コンパイラ」は、高速化のためにさまざまな最適化を行なう
    - 並列化
    - メモリ最適化
    - イディオム認識

コンパイラ最適化の機能と限界

- コンパイラ最適化は大きく進歩を見せ、現在ではソースコードからオブジェクトコードを類推することは困難になっているほどです。
  - どのような最適化が適用されたのか一目ではわからない
  - ソースコードをチューニングして、最適化と張り合おうとするのは愚の骨頂です
- でも、最適化は魔法ではありません
  - プログラム(静的)解析に基づくプログラム変換
  - アルゴリズムを解析してプログラム変換を行なうのは普通最適化といわない
  - 最適化　⊂　チューニング

高速化のための最適化

- 現在の最適化の流れ



高速化のための最適化

- HPC用の最適化は魅力的なテーマでした。
- 1980年代から2000年代前半にかけてデータ依存性解析に基づいたメモリアクセス最適化や並列性の抽出がさかんに研究されました。

- この講義では残念ながらこれを扱いません。興味のある人はちょっと古めのこの教科書がとりあえずのお勧め

HPCコンパイラの教科書

- High Performance Compiler for Parallel Computing

  Wolfe, M. (1995)

  ISBN 0805327304

プログラミングの価値の変遷

- 正しさへの要求の強まり
  - 分散環境では、データが外から飛んでくる（悪意を持ってデータを流し込んできたら…）
  - 分散環境では、コードが外から飛んでくる（悪意を持ってコードを流し込んできたら…）
  - そもそも、最適化ルーチンは正しく動作しているのか？
- プログラムの複雑さのアップ
  - 人間が人手でチェックするには複雑になりすぎた
  - 個々の最適化の正しさを人間が証明することは大変
- プログラムの生産性アップへの要求
  - 大規模なプログラムを効率よく開発するための言語でのサポート、ツールでのサポートの要求

正しさへの要求

- この講義ではこのトピックを最後にちょっとだけ掘り下げます。以下のことを考慮することの重要性はますます高まっています。

- 言語処理系の特に最適化が間違ったコードを生成しないことの保証はどこに？
- 悪意をプログラミングできないようなプログラミング言語とは？
- コードが「正しい」ことを証明しやすいプログラミング言語とは？
  - JAVAでのポインタの追放
  - 強力な型システムの導入
  - コンパイル時・実行時でのコードチェック

プログラミングの価値の変遷

- 次のような価値も認められています

- 消費電力最適化
- コードサイズ最適化
  - 特に組み込みシステムで

プログラミング言語の変遷

- プログラミングの概念の変遷（さっきやった）
  - 数値計算以外を対象に
- オブジェクトの登場（さっきやった）
- 「生産性」が評価軸に
  - プログラムのエラー（いろいろなレベル）をコンパイル時にチェックしてくれないか
  - 少量のコーディングですまないか
  - 「自然に」コーディングできないか
    ＝　High Level Programming
  - 一度書いたコードを使いまわしできないか
  - 一度書いたコードを他のマシンでも使えないか

プログラミング言語の変遷

- 現在の主流

- 高いレベルの概念を直接扱えるように
  - アプリケーション指向超高級言語
- 型の登場
  - データ型と関数プロトタイプ
- 再利用への関心
  - モジュール、ライブラリ、APIの言語による支援
- 分厚いライブラリとパターンで援護されたプログラミング （プログラム何行…ということが無意味になっている）

プログラミング言語の変遷

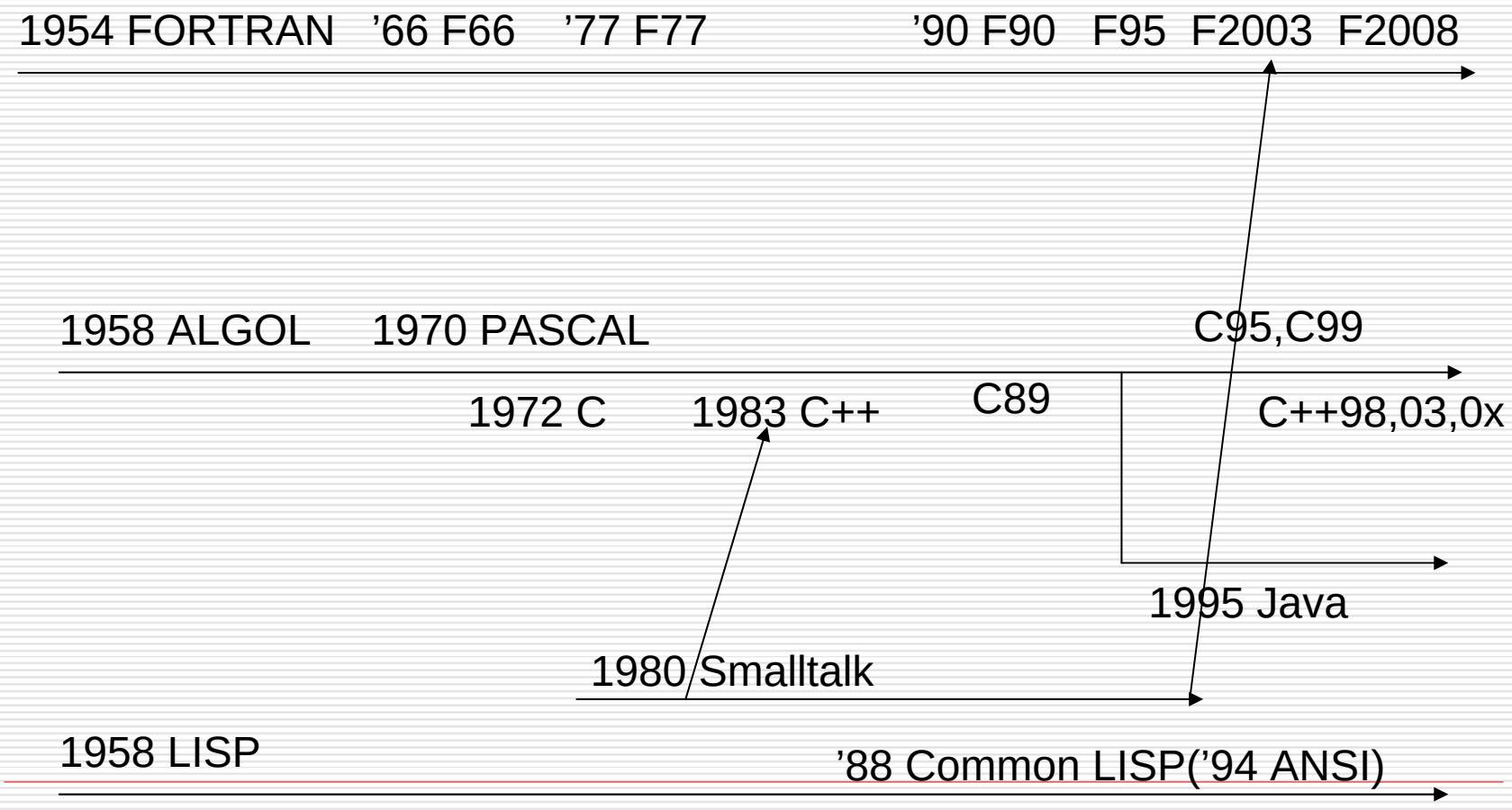
- でも、マシンが高級になるわけではない
- プログラムと物理の乖離が大きくなっている

コンパイラ（言語処理系）の出番

プログラミング言語の歴史

- Fortran, Algol系
- 規格、言語仕様の重要性の認識
- スクリプト言語
- JAVAの登場
- VMの登場

Programming Languages at a Glance



Fortranの登場

- 高級言語の概念の提示
- BNFを使った定義
- 数式（変数を使った数学の式）を直接表現することに主眼があった
- Assembly言語（GOTOが主たる実行制御方式）＋数式

- 言語規格を制定するときに大きな議論の場を提供した

ALGOL系言語の登場

- アルゴリズムの記述に主眼
- 制御構造その他で、「ALGOL系言語」にくくられるタイプをごく初期から決定した

- 子孫がたくさんいる
  - C, C++,…

オブジェクトの発明

- Smalltalk 72,76,80
- 多くの言語がオブジェクトを扱えるようになった（扱える ＝ first class object）
- CのstructとC++のclassを混同してはいけない

現在のプログラミング言語の流れ

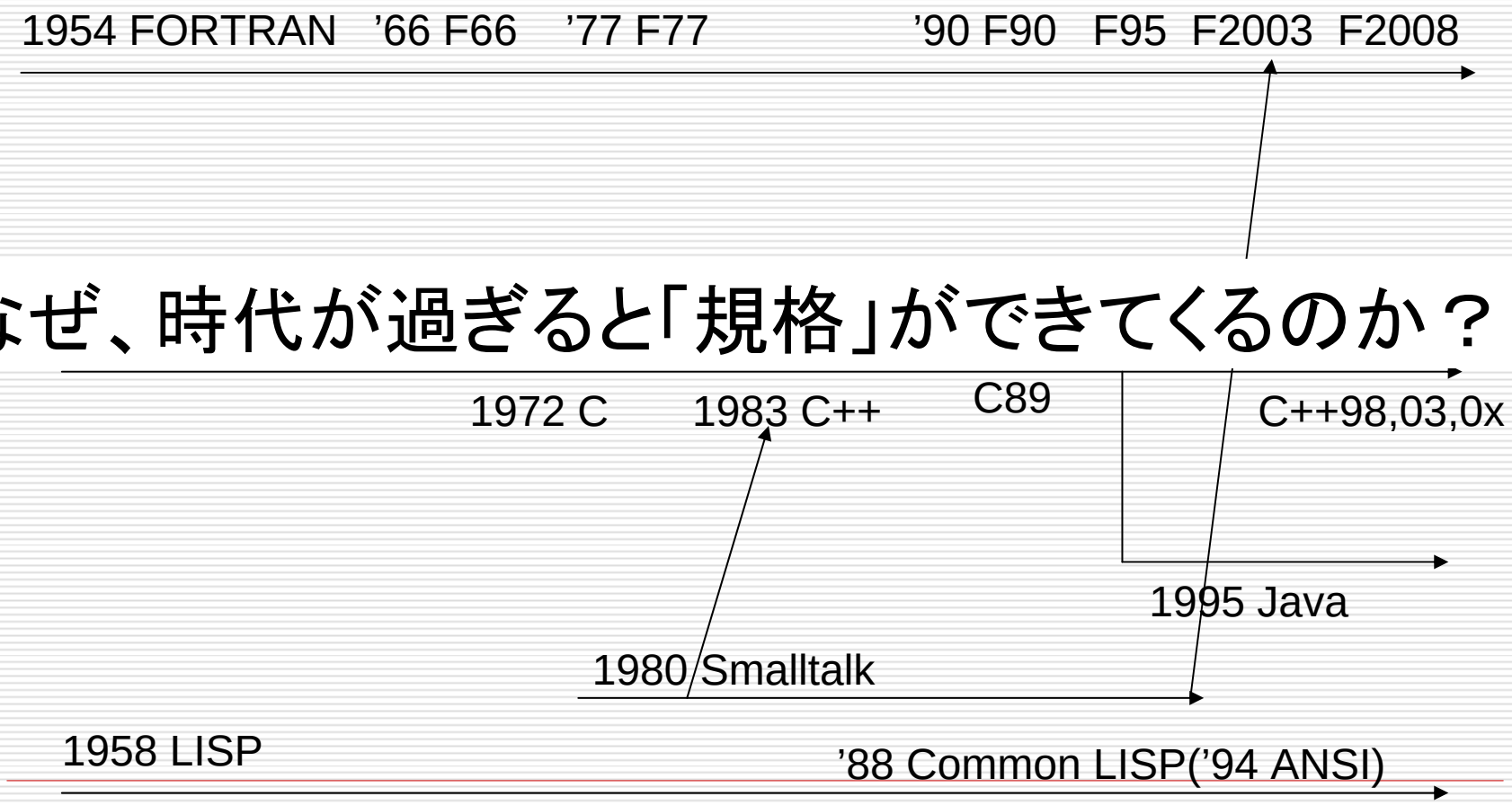
- オブジェクトはあたりまえ

- 総合的なプログラミング環境の提供
  - プログラミング方法論の拡張
    - テンプレート、継承、仮想関数
  - モジュールなどの提供
    - Module, namespace
  - ぶあついライブラリ群の提供

その他（落穂ひろい）

- スクリプト言語とインタープリタ
- システムとのインターフェイス、動的な環境変化への対応

最初の表に戻る


1954 FORTRAN
’66 F66
’77 F77
’90 F90
F95
F2003
F2008
なぜ、時代が過ぎると「規格」ができてくるのか？
1972 C
1983 C++
C89
C++98,03,0x
1995 Java
1980 Smalltalk
1958 LISP
’88 Common LISP(’94 ANSI)

プログラミング言語の規格(I)

- 初期のFortran
  - IBM Proprietary
  - Fortranが「使えるソフト」としてデファクトに
  - 各メーカがこぞってサポートを開始
  - 実装ごとに言語仕様を拡張 （そのうちのいくつかはよいアイデアとして他も採用）
  - 「Fortranプログラム」がコンパイルできるシステムとできないシステムがでてきた
- 規格の重要性の認識
  - ANSI (ISO)を主戦場とするか（C#）
  - 仲間を作って管理するか(各種コンソーシアム)
  - 一社（一者）で厳しく管理するか(Ada, Java)

プログラミング言語の規格(II)

- 規格の重要性の認識
- 規格を形式的に記述する技術の向上
  - SyntaxとSemanticsの分離
  - Syntaxは形式言語で(BNF)
  - Syntaxの足りないところは、BNFに対する注釈で

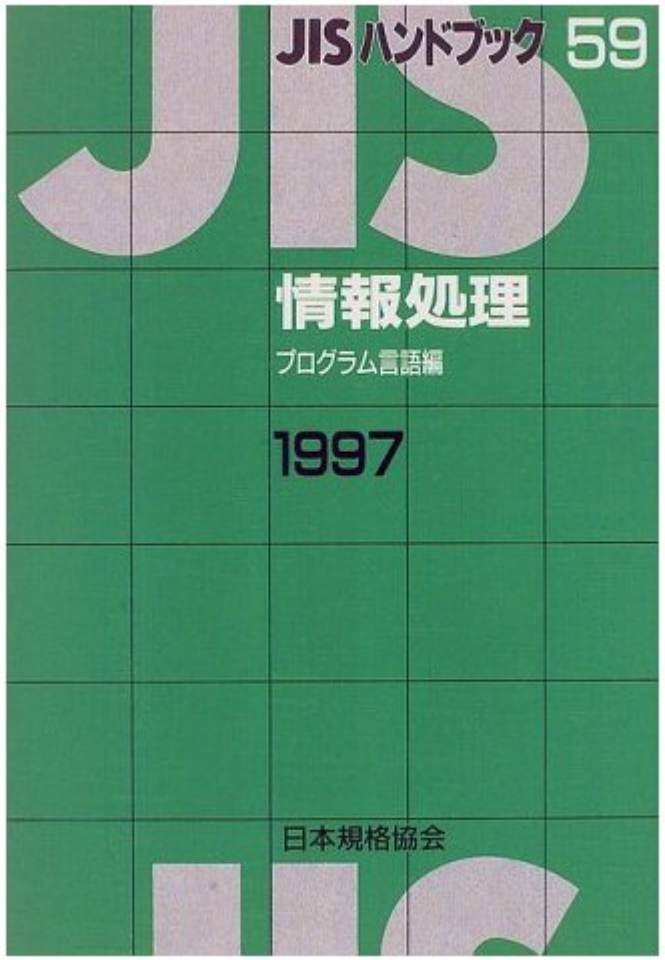
  - Semanticsはプログラムの実行の意味を決める
  - Semanticsは自然言語で記述

プログラミング言語の規格(III)

- Semanticsを記述する技術の向上が、言語の規格を厳格に定義することに大きく貢献した
  - 残念ながら、現在特定の形式主義に基づいたSemanticsの定義は行なわれていない（W3で無駄な試みがいくつか…）
  - Semanticsは自然言語で厳格に定義できる（数学が自然言語で展開されていることを考えればこれは驚くに足りない）
  - 必要だったのは、「形式主義」の理解と、それを遵守する能力（現在ではSemanticsを定める人間に大きな負担がかかっている）
- Fortranの規格を読んでみようか

プログラミング言語の規格

- 国際・国内機関
  - ISO
  - JIS
- コンソーシアム
  - IETF
  - W3
- その他
  - RSA他、ガリバーが保守する規格



- セマンティックスをもとにした最適化

この授業のテーマ(ブレークダウン)

- 「プログラミング環境」とは何か？新しい環境の考察
- XMLは、この授業でどういう役割を果たすのか？
- Webで何をプログラミングするのか？新しい概念の考察
- プログラミング言語はどう設計すればよいのか？型、名前空間、…
- 性能向上のためになにをすればよいのか？

授業のテーマ(ブレークダウン)

- われわれが注意すべき「オープンな規格」とはなにか
- われわれが使えるツールとしては何があるのか？