

コンピュータ概論B - ソフトウェアを中心に -

---

#8 コンパイル・プログラムの実行

Yutaka Yasuda

# プログラミング

---

- 「1から10までの数を足した結果を出せ」  
これはコンピュータにとって「難しすぎる」指示
- 手順の明確化（アルゴリズムを得る）  
「Xを1から10まで変化させ、毎回Yに繰り返し込め」
- プログラムとは何か  
目的に対して「何をどう処理するか」を詳述したものの  
アルゴリズムをプログラミング言語で書き下したものの

# 手順をどのように書くか

---

- コンピュータは日本語を理解できない

「Xを1から10まで変化させ、それを毎回Yに繰り返す」のもコンピュータには複雑すぎる

コンピュータが理解できる言語で書き直す必要がある

C言語での例

```
Y=0;  
for (X=1; X<=10; X++) {  
    Y=Y+X;  
}
```

Yははじめ0だと設定している

1から10まで変化させるということを、「1からはじめて10以下の場合は終わりまでの処理を行い、1加算してもう一度繰り返し」という表現で明記している

Yに増え続けるXを足したものを再びYに代入

## 手順をどのように書くか

---

- しかしC言語でもハードウェアには直接理解できない  
さらに単純な処理に分解しなければ  
CPU(例は Motorola 68000)向けのアセンブリ言語で記述

```
    st 0,$1004
    mov 1,%o0
    st %o0,$1000
.LL2:
    ld $1000,%o0
    cmp %o0,11
    bgt .LL3
    ld $1004,%o0
    ld $1000,%o1
    add %o0,%o1,%o0
    st %o0,$1004
    ld $1000,%o1
    add %o1,1,%o0
    mov %o0,%o1
```

### C言語での例

```
Y=0;
for (X=1;X<=10;X++) {
    Y=Y+X;
}
```

## 手順をどのように書くか

---

- アセンブリ言語でもまだ直接は理解できない。  
更に機械向けの言語＝機械語に直さなくては

(以下は機械語の一部)

```
7400 5f5f 4354 4f52 5f4c 4953 545f 5f00
5f65 6e76 6972 6f6e 005f 656e 6400 5f47
4c4f 4241 4c5f 4f46 4653 4554 5f54 4142
4c45 5f00 6174 6578 6974 0065 7869 7400
....
```

# 二つの変換

---

人間側

1から10までの数を  
足した結果を得る

人間が変換  
(プログラミング)

```
Y=0;  
for (X=1;X<=10;X++) {  
    Y=Y+X;  
}
```

機械的に変換できる  
(コンパイラ)

```
02af93e8f  
37de76e0a  
4e3a2...
```

コンピュータ側

# コンパイラとインタプリタ

---

- プログラミング言語から機械語へ  
CPUは機械語しか実行できない  
機械語によって等価な振舞いをさせなければ
- どうやって?  
二つの方法
- コンパイラ  
機械語に変換する
- インタプリタ  
変換せず真似て動作させる

# コンパイラ

---

- Compiler = 編集、編纂
- 機械語に変換して実行

変換前：原始プログラム (Source program)

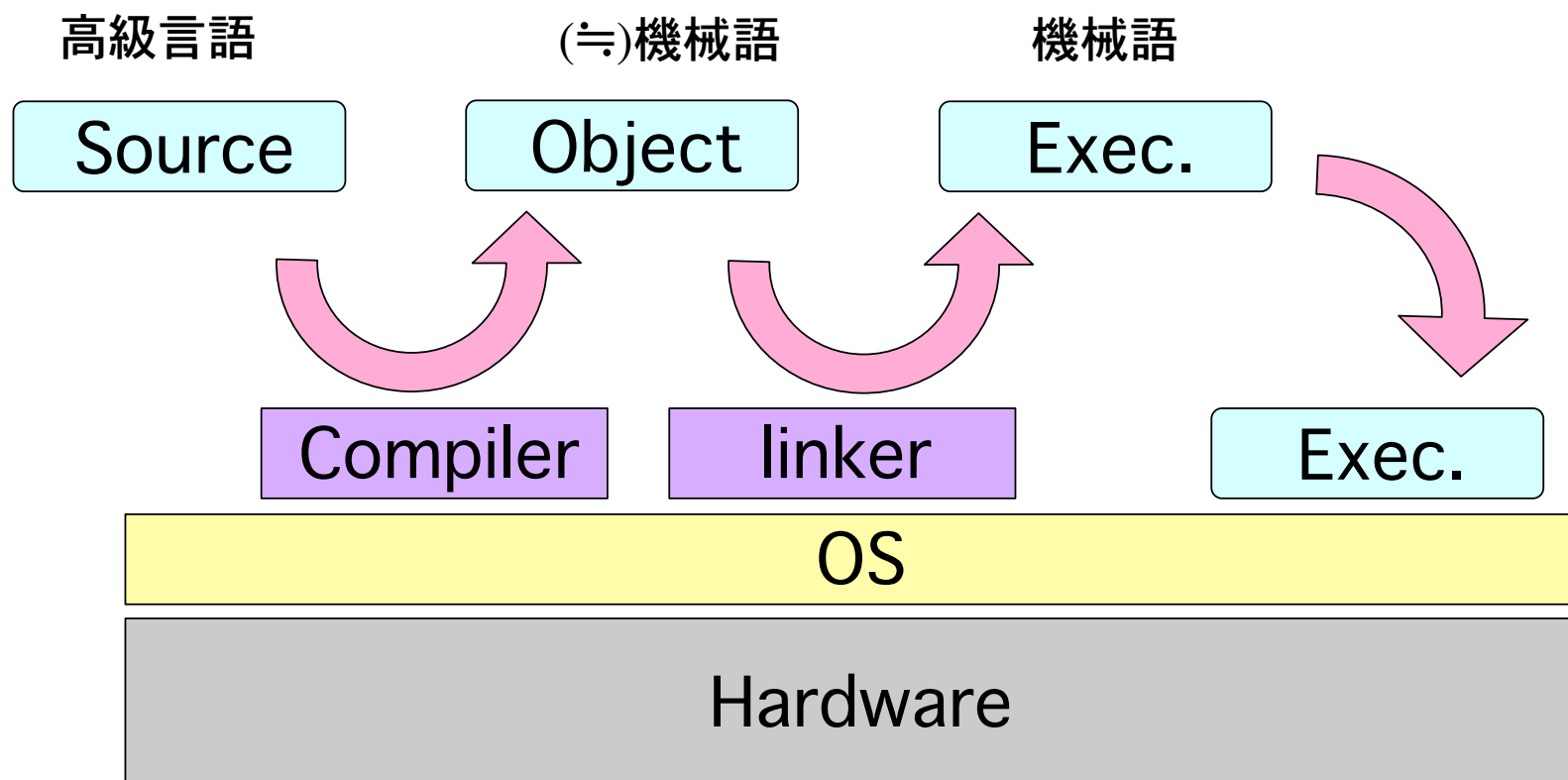
変換後：目的プログラム (Object program)

- 多くの OS では目的プログラムをリンク (link) と呼ばれる処理を通してライブラリと結合し、実行可能プログラム (executable program) とし、これを実行する



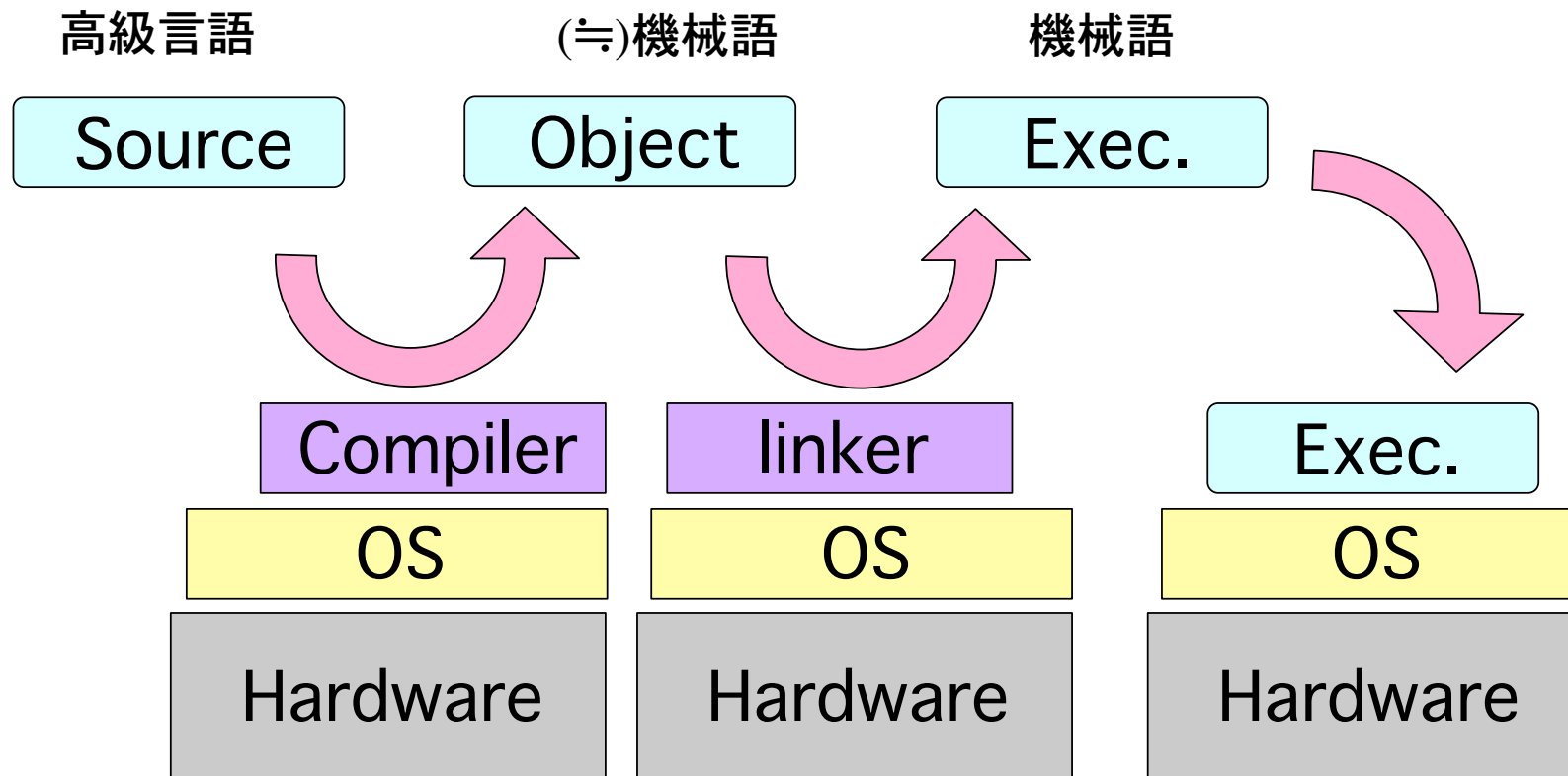
# コンパイラ

---



# コンパイラ

---



別々であっても構わない

# コンパイラ

---

- 最適化の可能性
- 変換一回、実行多数回、では高効率
- 機密保持

ソースが得られない場合が多い（逆変換不可）

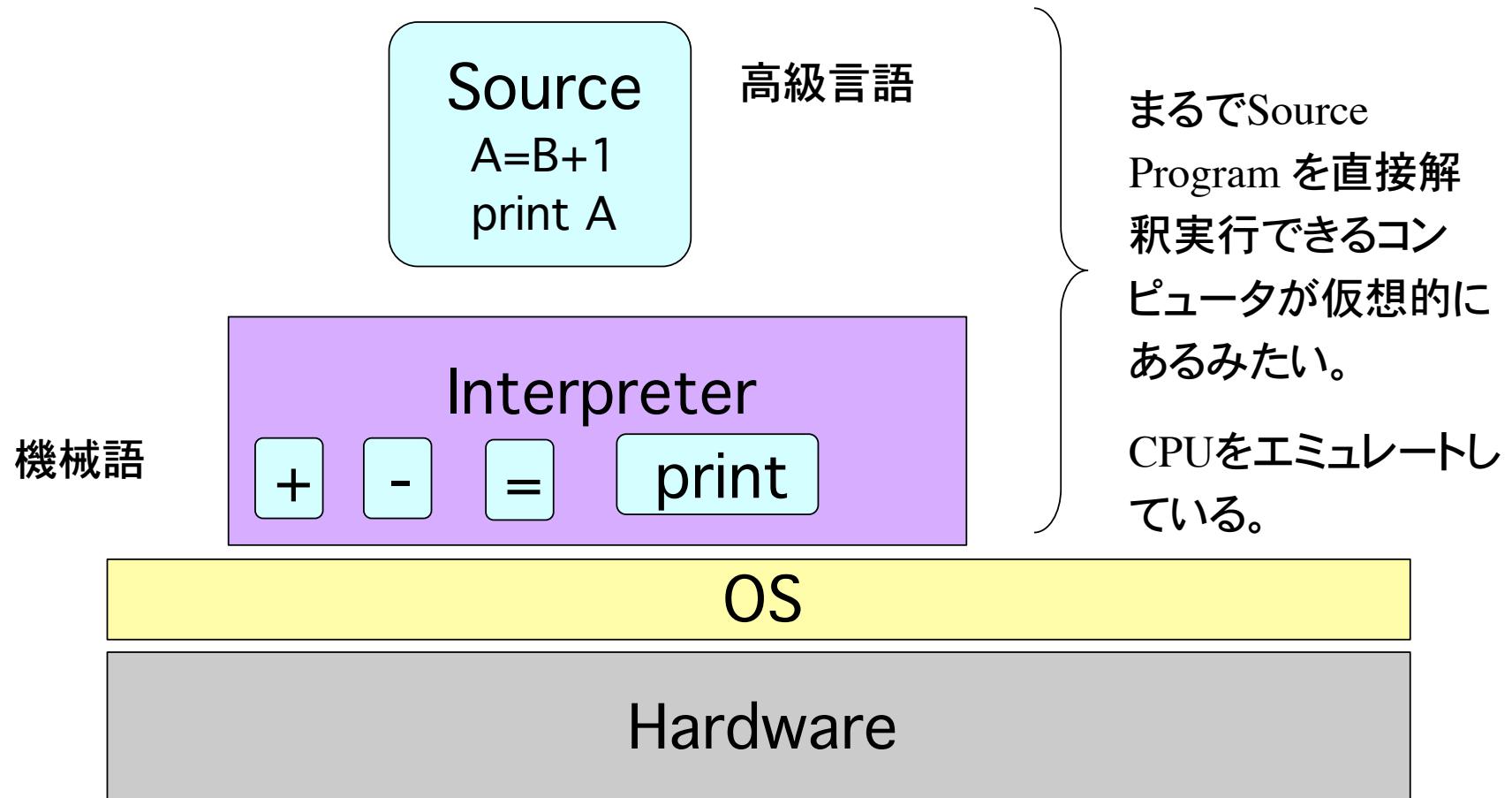
- 商用ソフトウェアでこうした性質に合う場合多し

# インタプリタ

---

- プログラムの機能ごとに対応し、部分的に実行可能な機械語コードを予め用意しておく
- 実行したいプログラムを部分的に読み出し、対応する機械語コードを選択して実行
- Interpreter = 通訳、ではあるが変換はしていない  
逐語的に処理しているため、と理解するべき  
まるでシミュレーション  
(simulate : 装う / emulation : 真似る)
- その言語を直接実行できる CPU をソフトウェアで作ったようなもの

# インタプリタ



変換はされていない!

# インタプリタ

---

- 部分的実行が可能

一行ずつテストしながら開発するような作業が可能

- 実行速度が遅い

真似るために必要な機械語は等価変換した機械語より処理量が多い

- CPU/機械語が異なる環境でも動作する

- Java / iアプリ

## Java : インタプリタによる言語の例

---

- Sun Microsystems が 1995 年に開発
  - C に似た文法
- まず中間プログラムにコンパイル
  - 実行速度を向上させるため
  - バイトコード (Byte Code) と呼ばれる
- インタプリタで実行
  - CPU やハードウェアの差を吸収して実行
  - Java VM (Virtual Machine 仮想計算機) の存在
  - iAPPLや Web で使われる

# Java

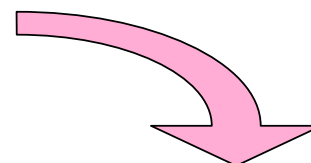
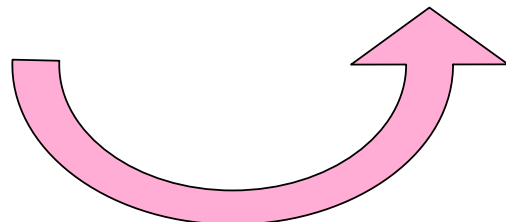
---

Java 言語

中間コード

Source

Object



Compiler

Java VM (Interpreter)

+

-

=

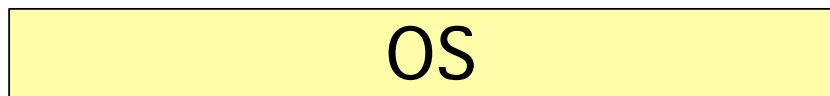
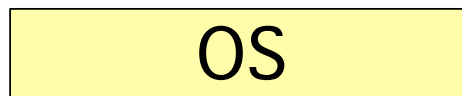
print

OS

OS

Hardware

Hardware





# コンパイラとインタプリタ

---

- コンパイラ：一括変換して実行
- インタプリタ：逐次真似して実行
- 様々な方式がある
  - Java のような組み合わせ
- Java 自身、VM 高速化のためにいろいろやっている
  - Just In Time コンパイラ：バイトコード実行前に一括して機械語に変換、実行
  - Hot Spot：バイトコードの一部（ループの中など）をコンパイルしながら実行
- 工夫の産物である
  - 無意味な分類にとらわれないように