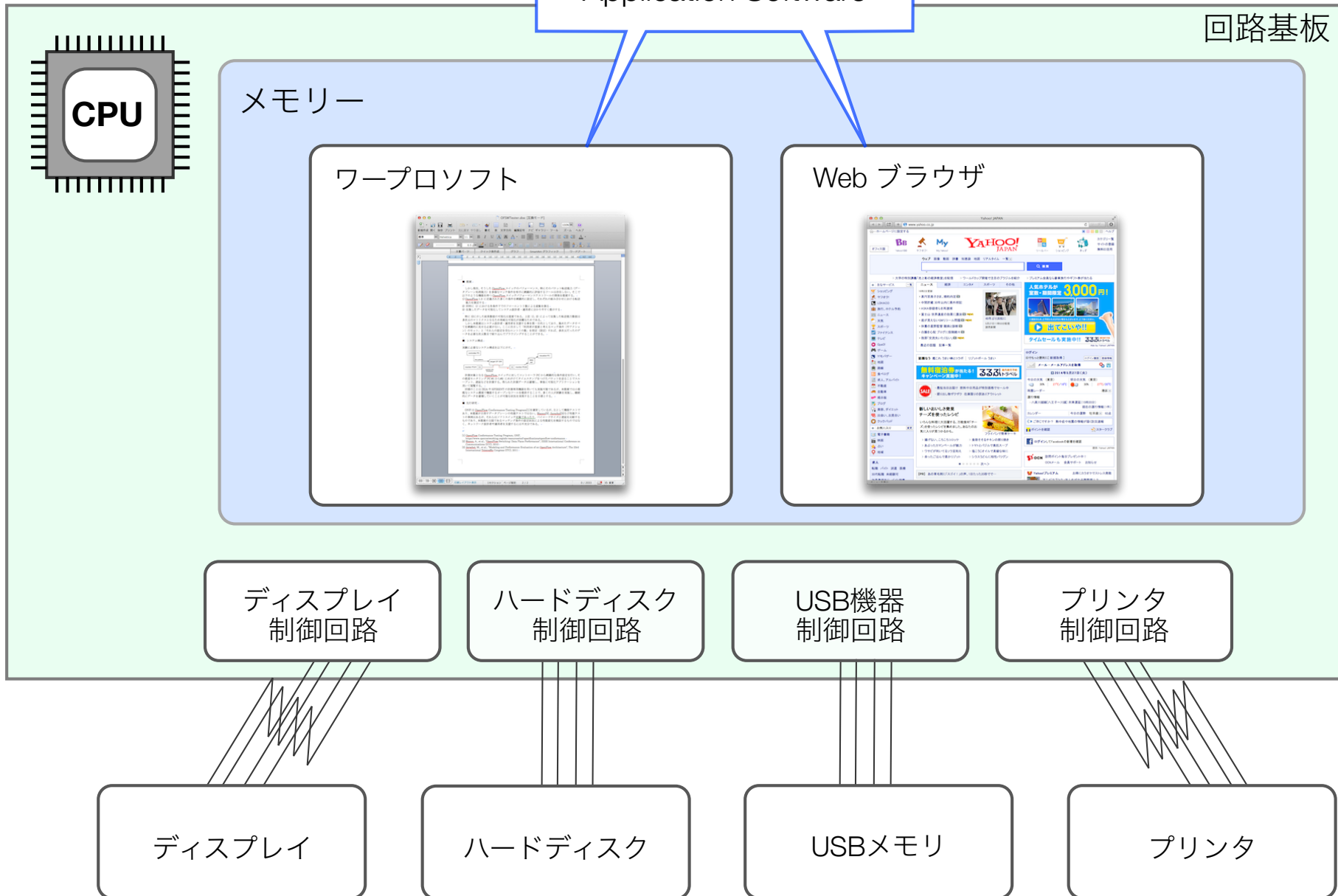


情報科学入門

#11 機械語・高級言語・プログラムの実行

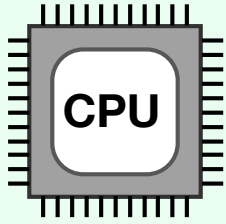
コンピュータ



コンピュータ

Application Software
(Word, Excel, etc...)

Operating System (OS)
(Windows, etc.)



CPU

メモリー

システムそのものの操作いろいろ...

ワープロソフト

コピー操作

システム更新

削除操作

プリンタ追加

残容量確認

アプリ起動

各種制御
プログラム

ハードディスク
制御プログラム

USB機器
制御プログラム

各種制御
プログラム

ディスプレイ
制御回路

ハードディスク
制御回路

USB機器
制御回路

プリンタ
制御回路

ディスプレイ

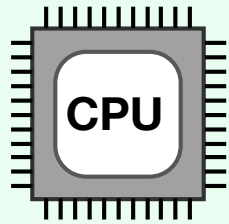
ハードディスク

USBメモリ

プリンタ

回路基板

アプリケーションの起動

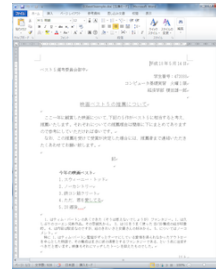


メモリー

Windows
(system software)



ワープロソフト



Web ブラウザ

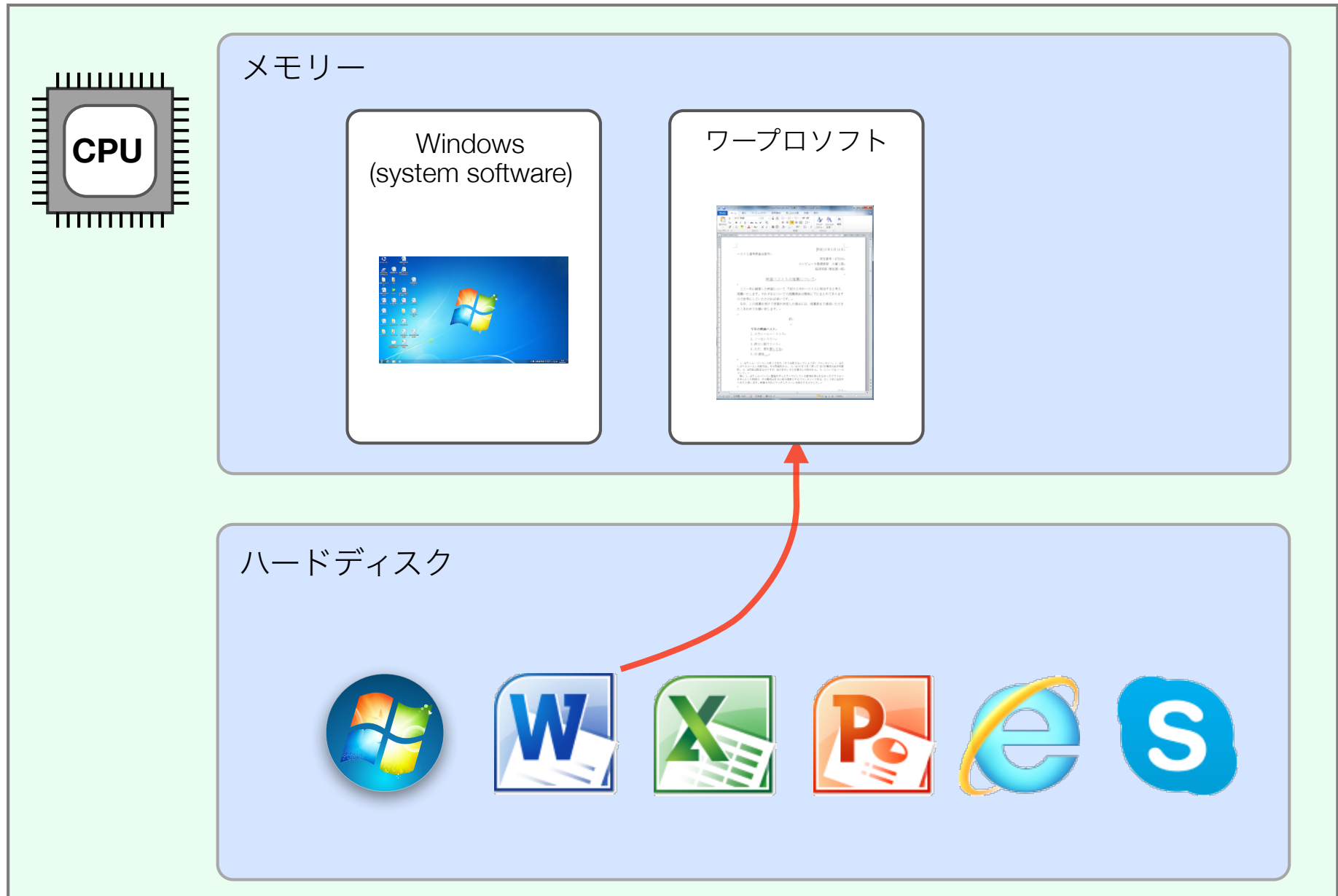


ハードディスク

アプリケーションの起動



アプリケーションの起動

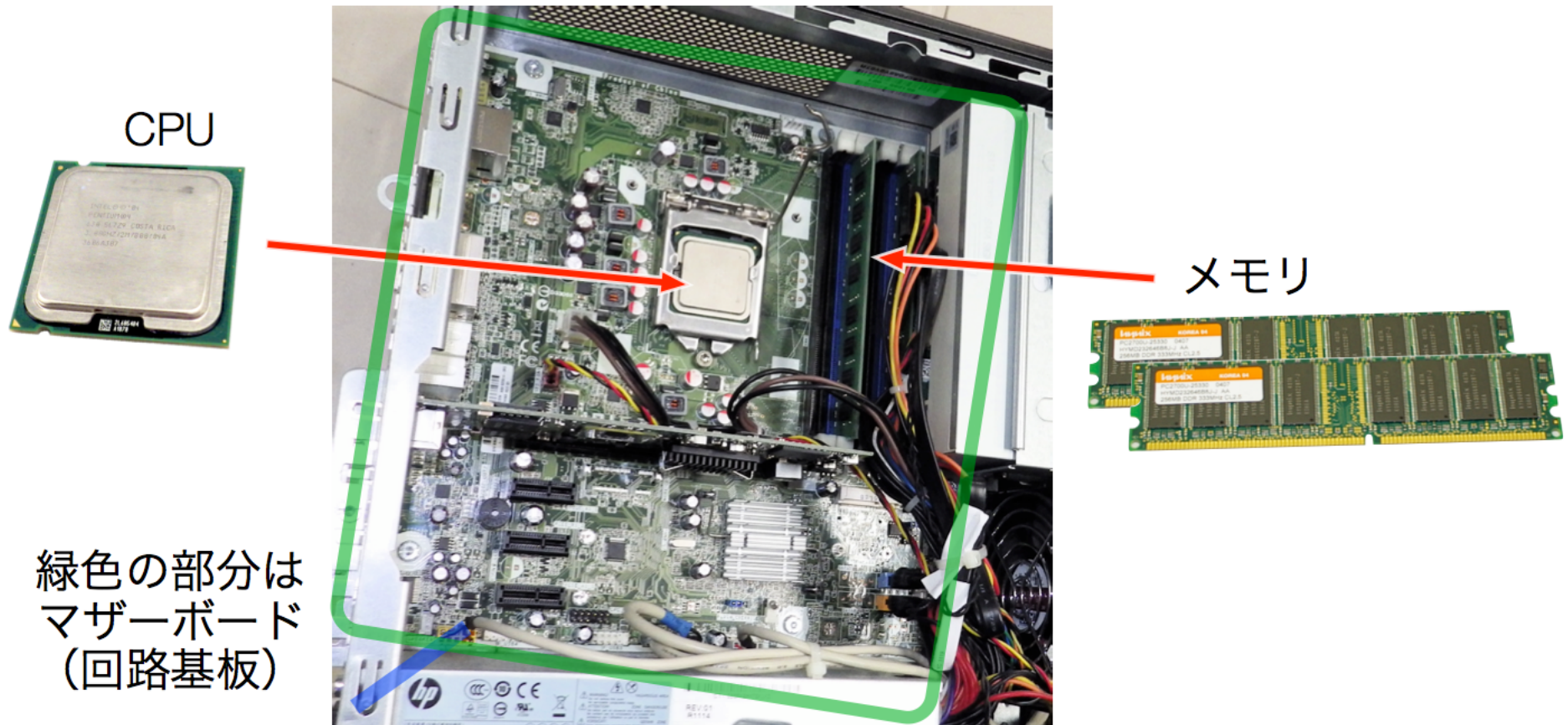


アプリケーションの起動



PC内部(クローズアップ)

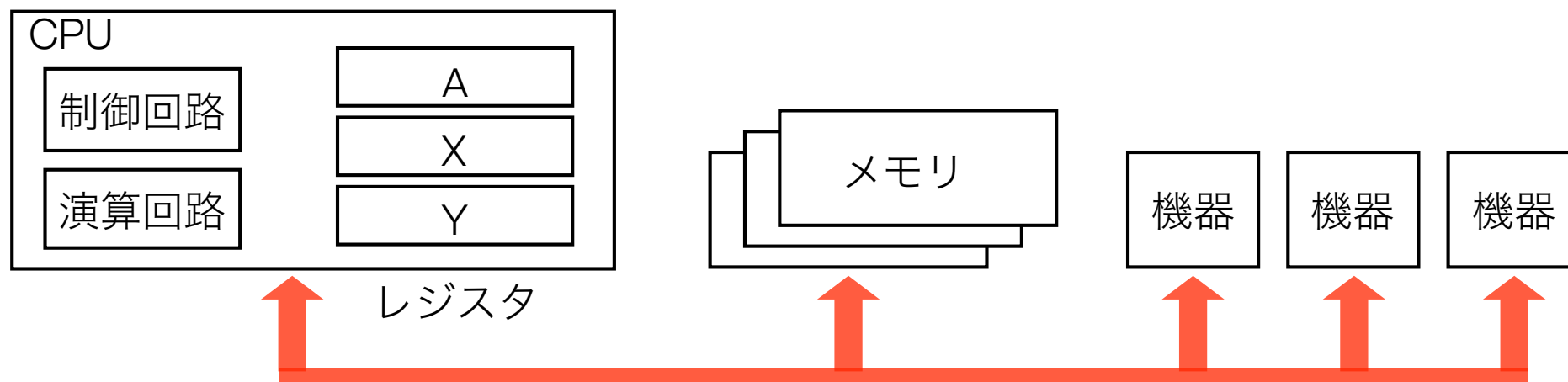
(CPU 冷却ファンを外した状態)



機械語：CPU向け作業指示

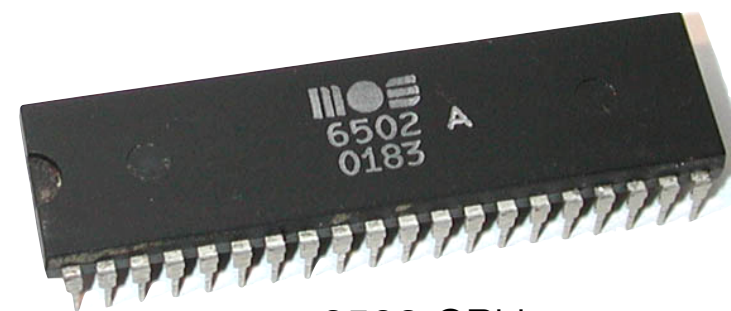
- CPUに対する命令
(CPUの機能を指定して実行)
- CPUの構造
レジスタの存在
メモリとレジスタを利用して
計算結果を保持しながら連続
処理

- 命令はメモリに連続的に配置
- CPUにできること
レジスタ・メモリ間のデータのやりとり
レジスタ・レジスタ間または
レジスタ・メモリ間の演算や
比較、条件分岐
機器回路の制御



CPU の例：6502

- 1975 MOS Technology
- 8bit CPU
- Apple II, Family Computer, etc.
- 1-2 MHz clock
- 64KB memory
- レジスタ 3 つ
- 非常にシンプル



6502 CPU



初代ファミコン (1983)



Apple II (1977)

機械語の例

メモリの中ではこんな感じ

MOS Technology の 6502 で 12 と 23 を足す例

アセンブリ言語

```
LDA $1002
CLR
ADC $1003
STA $1002
```

```
AD 02 10
18
6D 03 10
8D 02 10
```

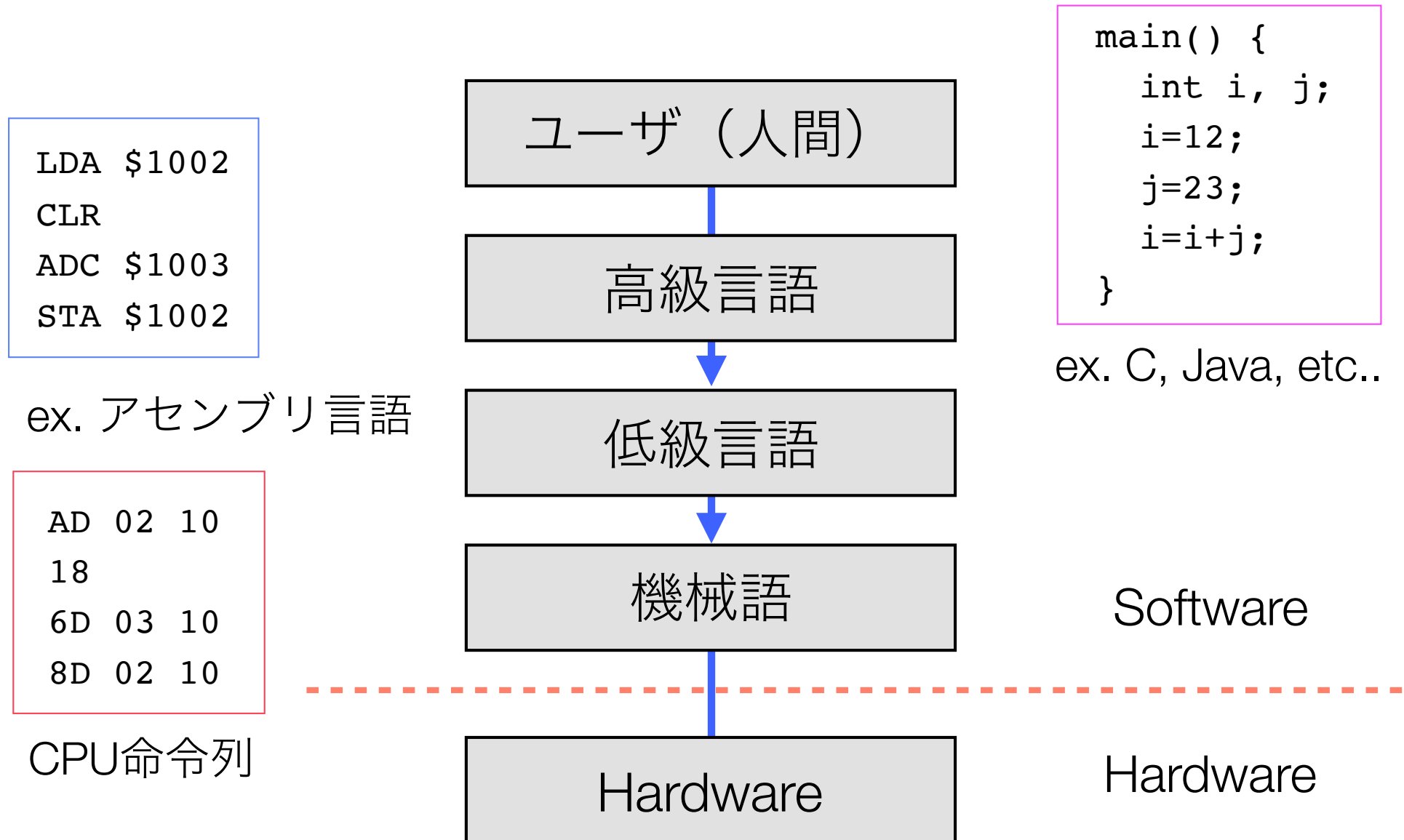
これが機械語

AD
02
10
18
6D
03
10
8D
02
10
:

1. 1002番地の値をアキュムレータに読み
2. 桁上がり無しで
3. 1003番地にある値を加え
4. 1002番地に書き込み

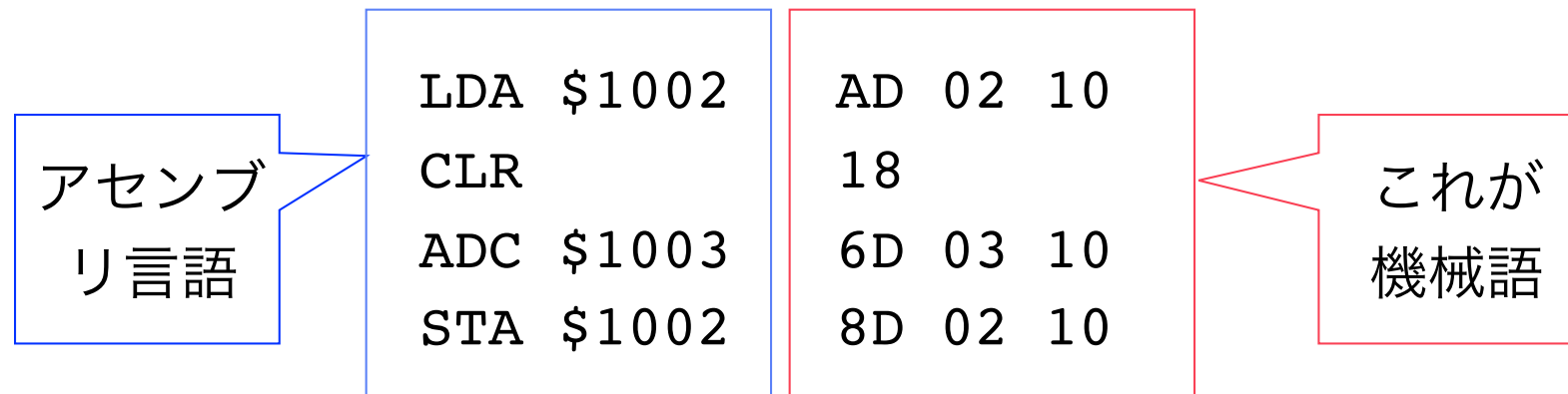
	:
1002	12
1003	23

プログラムの実行



アセンブリ言語

- 機械語を読みやすい形式（ニモニック）で表現したもの
このように表現するだけでかなり読みやすくなる。
- 実際には複雑な機能もあるがここでは紹介しない



マシン語とアセンブリ言語

- マシン語

ハードウェアにべったり依存

レジスタの名前や本数、CPU 命令 (=回路) の番号

CPU設計時に言語仕様が決定

- アセンブリ言語

低級言語

マシン語とほぼ一対一対応

レジスタ本数などハードウェア依存の性質が残る

実際にプログラミングする際には、どのCPU上で実行されるか意識してプログラミングする必要あり

もう一つCPUの例：ARM Cortex-A9

- 2011, ARM (v7)
- 32bit CPU
- iPhone 4S, Galaxy SII, etc.
- 800MHz - 1.2GHz clock
- 512MB - 1GB memory
- レジスタ整数16, 実数32 本
- 典型的なモバイル32bit CPU



Apple A5 (2011)



iPhone 4S (2011)



GALAXY S II (2011)

アセンブリ言語のCPU依存性

ARM v7

例の表記は一般的でなく、またレジスタ間接による実効アドレス計算などが省かれており正確ではありません

ldr r1, [sp, #4]	R1 に sp +4 番地の値を読み込み
ldr r2, [sp]	R2 に sp 番地の値を読み込み
add r1,r2	R2 を R1 に加算
str r1, [sp, #4]	結果の R1 を sp +4 番地に書き込み

MOS Technology 6502

LDA \$1002	1002番地の値をアキュムレータに
CLR	桁上がり無しで
ADC \$1003	1003番地にある値を加え
STA \$1002	1002番地に書き込み

高級言語の例

- 要求

CPU 等ハードウェア条件に依存しないプログラミング

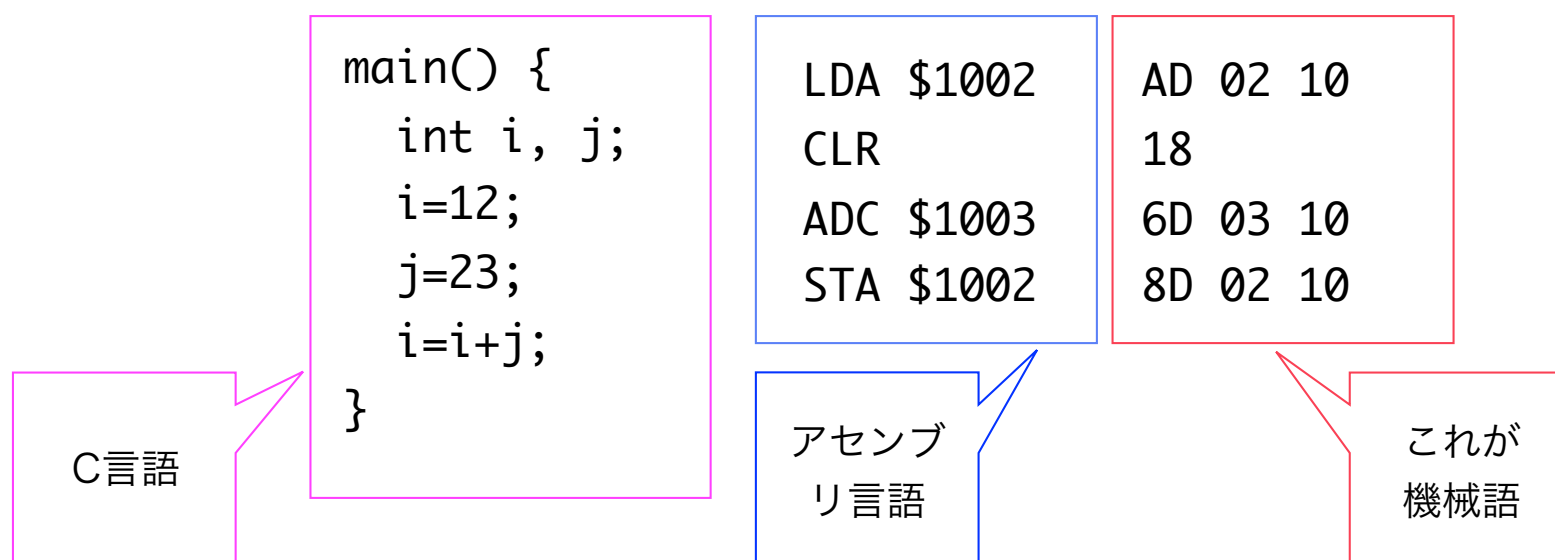
長持ちして欲しい

いろんなシステムで動いて欲しい

- アイディア

高水準言語で書いて、低水準言語（または機械語）に変換すれば良い

高級言語の例（C言語）



- メモリの構造やレジスタの名前を意識しないプログラムが書ける
- 問題：

CPUは機械語しか実行できない

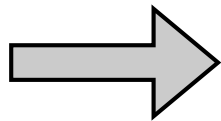
機械語によってC言語プログラムと等価な振舞いをさせなければ

→変換すれば良い

コンパイラ：高級言語から低級言語への変換

C 言語

```
main()  
{  
    int i,j;  
    i=1954;  
    j=1959;  
    i=i+j;  
}
```



ARM v7 アセンブリ言語

```
.section __TEXT,__text,regular,pure_instructions  
.globl _main  
.align 1  
.code 16 @ @main  
.thumb_func _main  
  
_main:  
@ BB#0:  
    sub sp, #8  
    movs r0, #0  
    movw r1, #1959 R1 に 1959 をセット  
    movw r2, #1954 R2 に 1954 をセット  
    str r2, [sp, #4] R2 を sp+4 番地に書き込み  
    str r1, [sp] R1 を sp 番地に書き込み  
    ldr r1, [sp, #4] sp+4 番地を R1 に読み出し  
    ldr r2, [sp] sp+4 番地を R2 に読み出し  
    add r1, r2 R1 に R2 を加算  
    str r1, [sp, #4] 結果の R1 を sp+4 番地に書き込み  
    add sp, #8  
    bx lr
```

この変換作業は機械的な作業なので人手
ではなくソフトウェアによって行える！

抽象度の違い：ハードウェア依存性からの脱却

二つの整数を加算するプログラム

```
47 F0 E0 57  
F0 E4 E7 F0  
E8
```

機械語

- レジスタ番号、メモリアドレスを直接指定

```
LD G1, F0E0  
ADD G1, F0E4  
ST G1, F0E8
```

低級言語

(例はアセンブリ言語)

- 機械語とほぼ一対一
- レジスタ番号などが記述に残っている

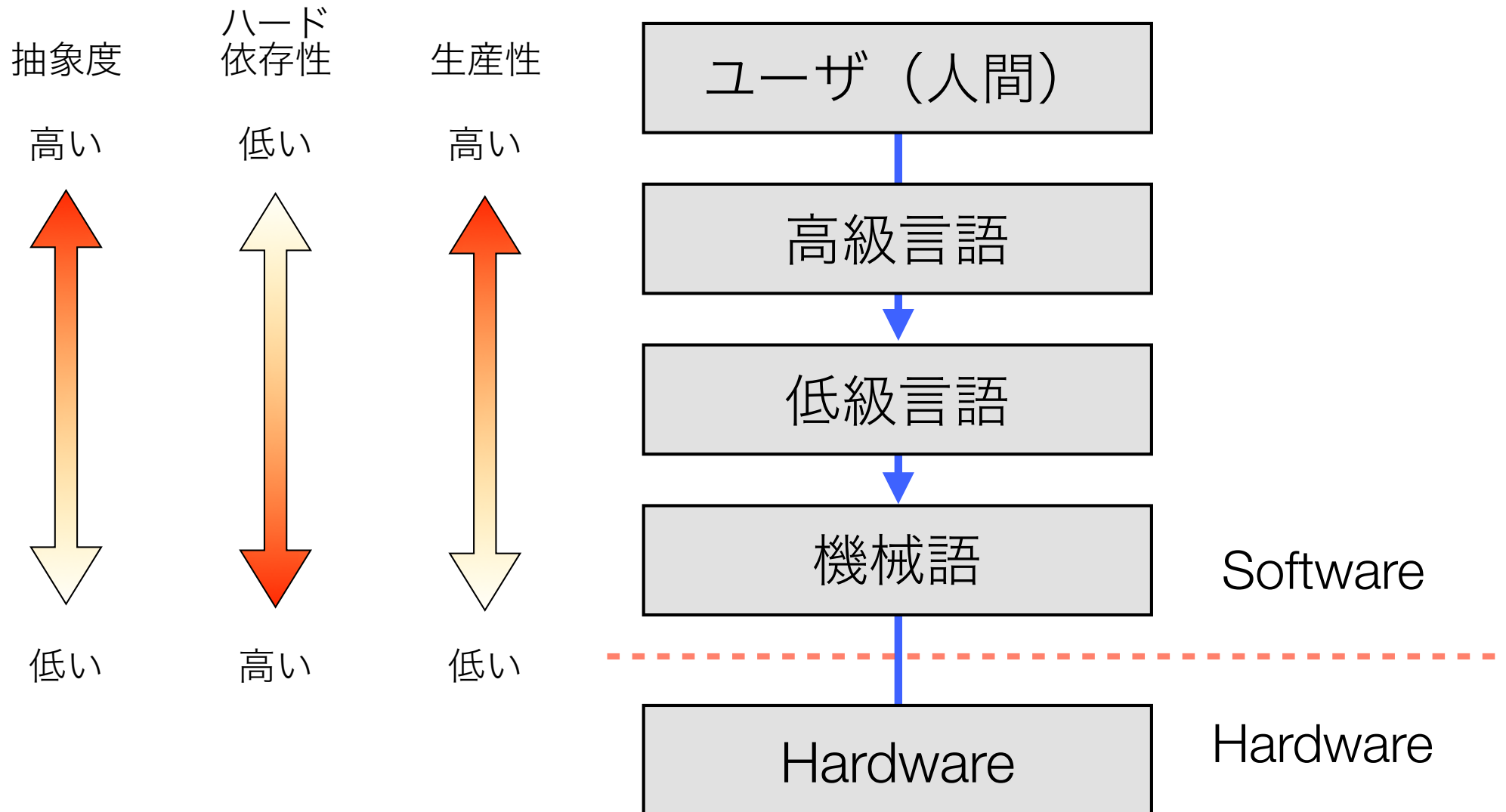
```
a=b+c
```

高級言語

(例はC言語)

- ハードウェアの中身から独立している
(移植が容易になった)
- 自然言語に近い (プログラマにやさしい) 語彙と文法
- 抽象度が高まった
(開発・保守が容易になった)

プログラムの実行 (再掲)



自然言語と人工言語

- 自然言語
 - 人間が日常生活で用いている言語 (日本語、英語、 etc..)
- 人工言語
 - 人間が意図的に作り出した言語
 - エスペラント (1887年 L.L. Zamenhof , ポーランド)
 - 全てのプログラミング言語
- プログラミング言語の特徴
 - 決定的な動作のための明確な手順指示書
 - 一点一字の間違いも許されない
 - 限定的な語彙 (C言語では予約語は 32)
 - 簡単な文法

アルゴリズム

- 目的とする結果を得るための処理手順

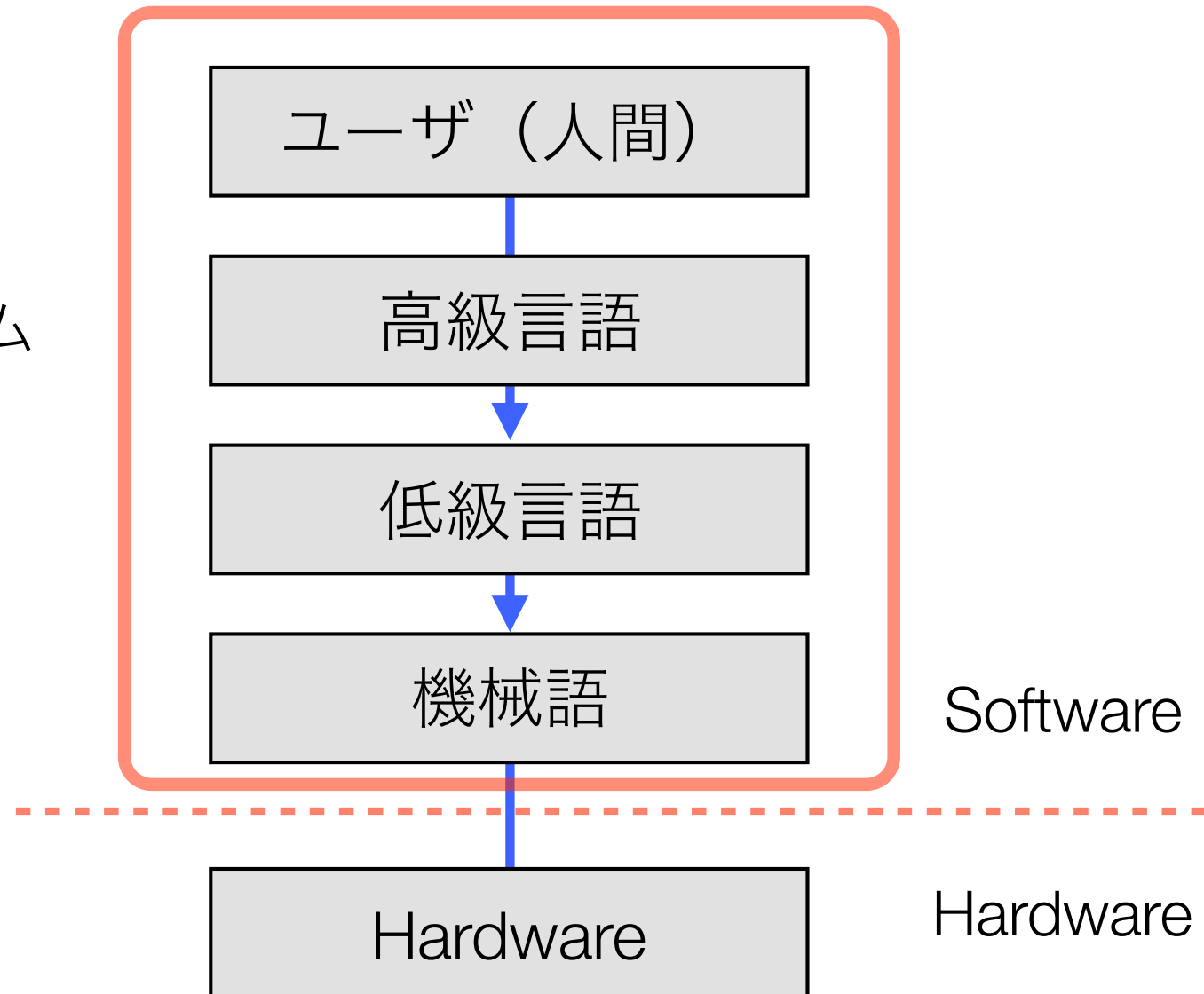
ソート

素数の列を求める

- アルゴリズムをプログラミング言語で書き下すことがプログラミングである。

プログラムの実行（再掲）

同じアルゴリズム
だが
異なる表現



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
- コンピュータには What ではなく How が必要

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くださる必要があ

C言語での例

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

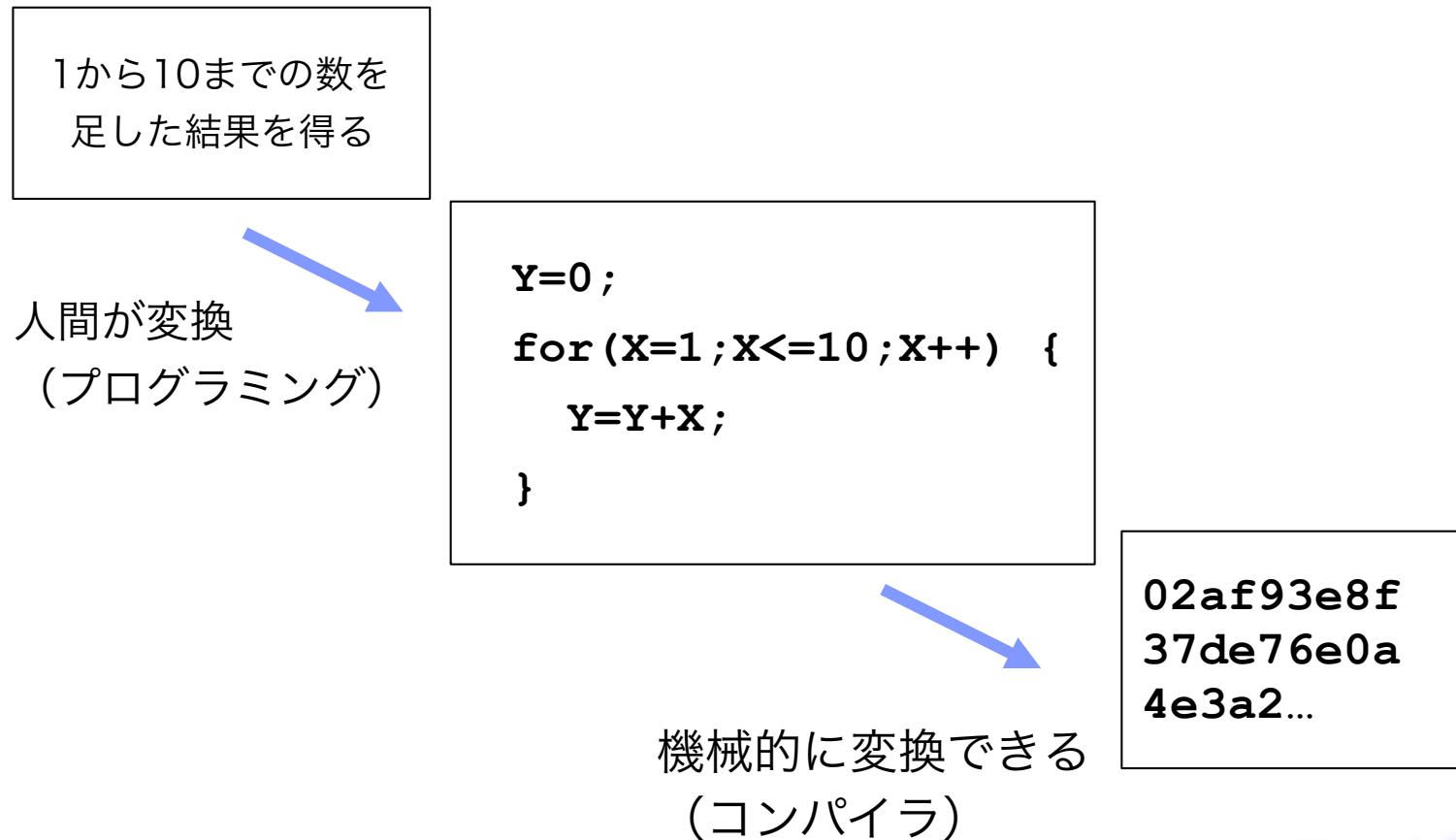
Yははじめ0だと設定している

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

アルゴリズムから機械語へ

人間側



コンピュータ側

コンパイラ

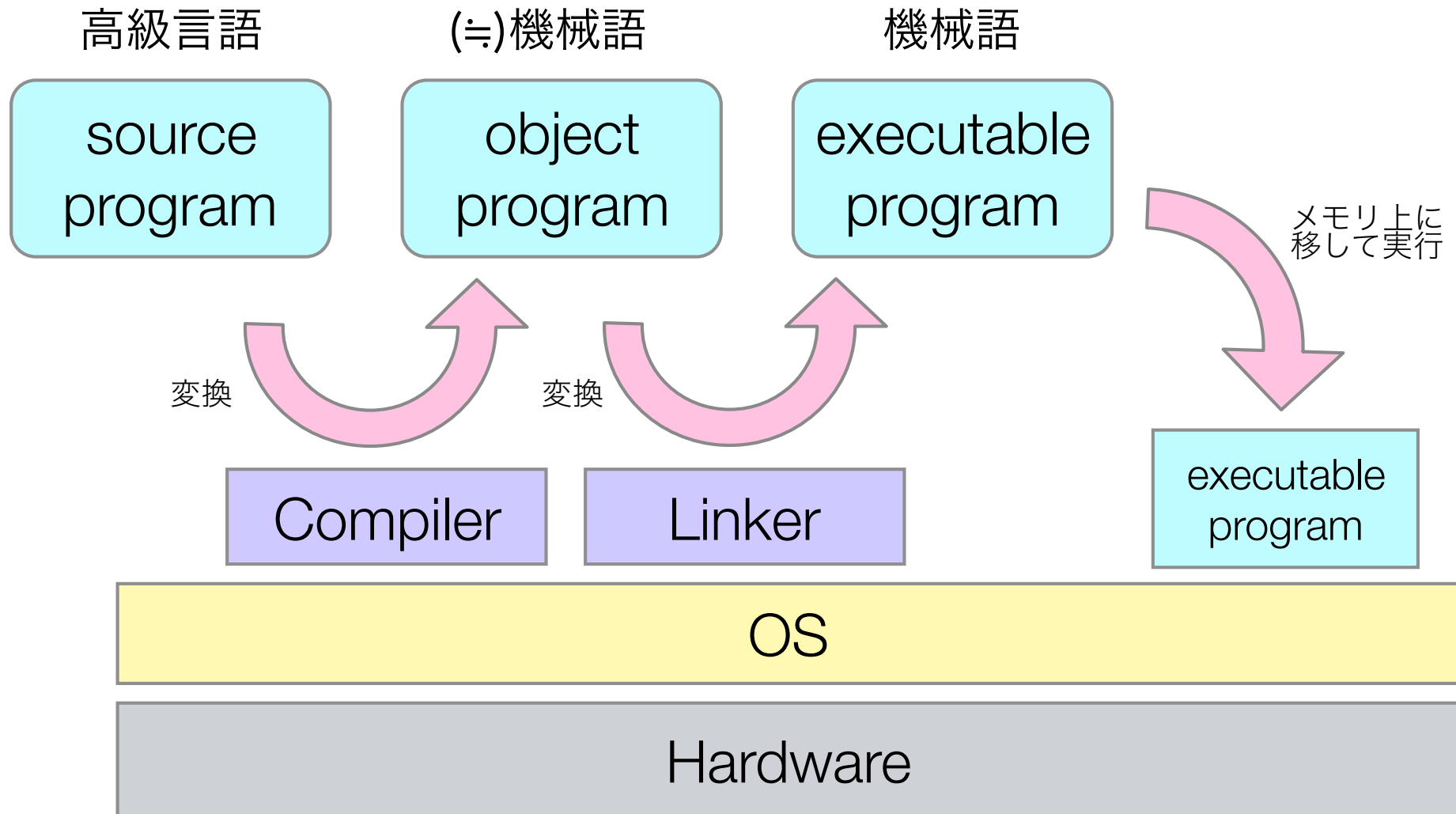
- 機械語に変換して実行

変換前：原始プログラム (Source program)

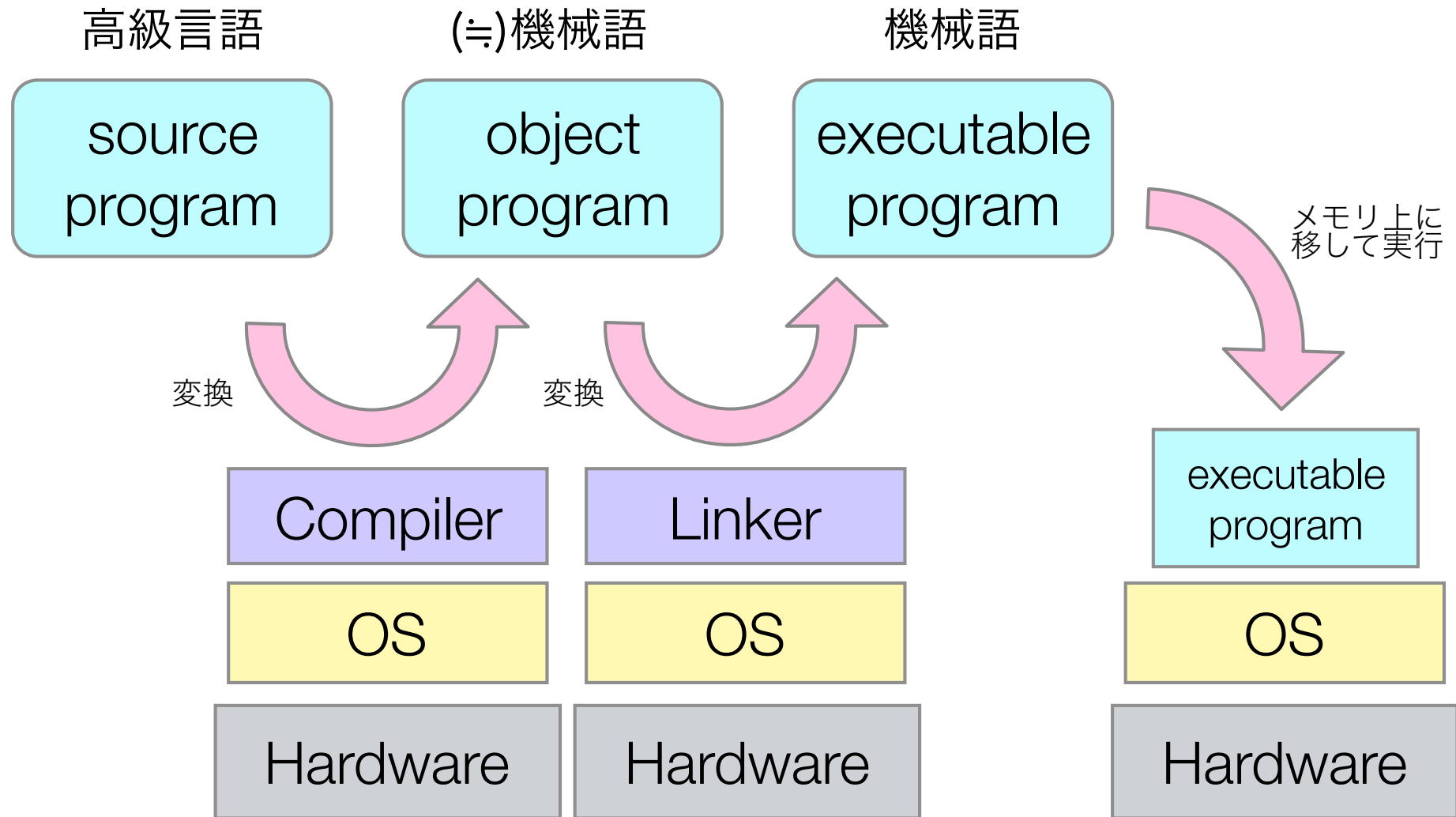
変換後：目的プログラム (Object program)

- 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