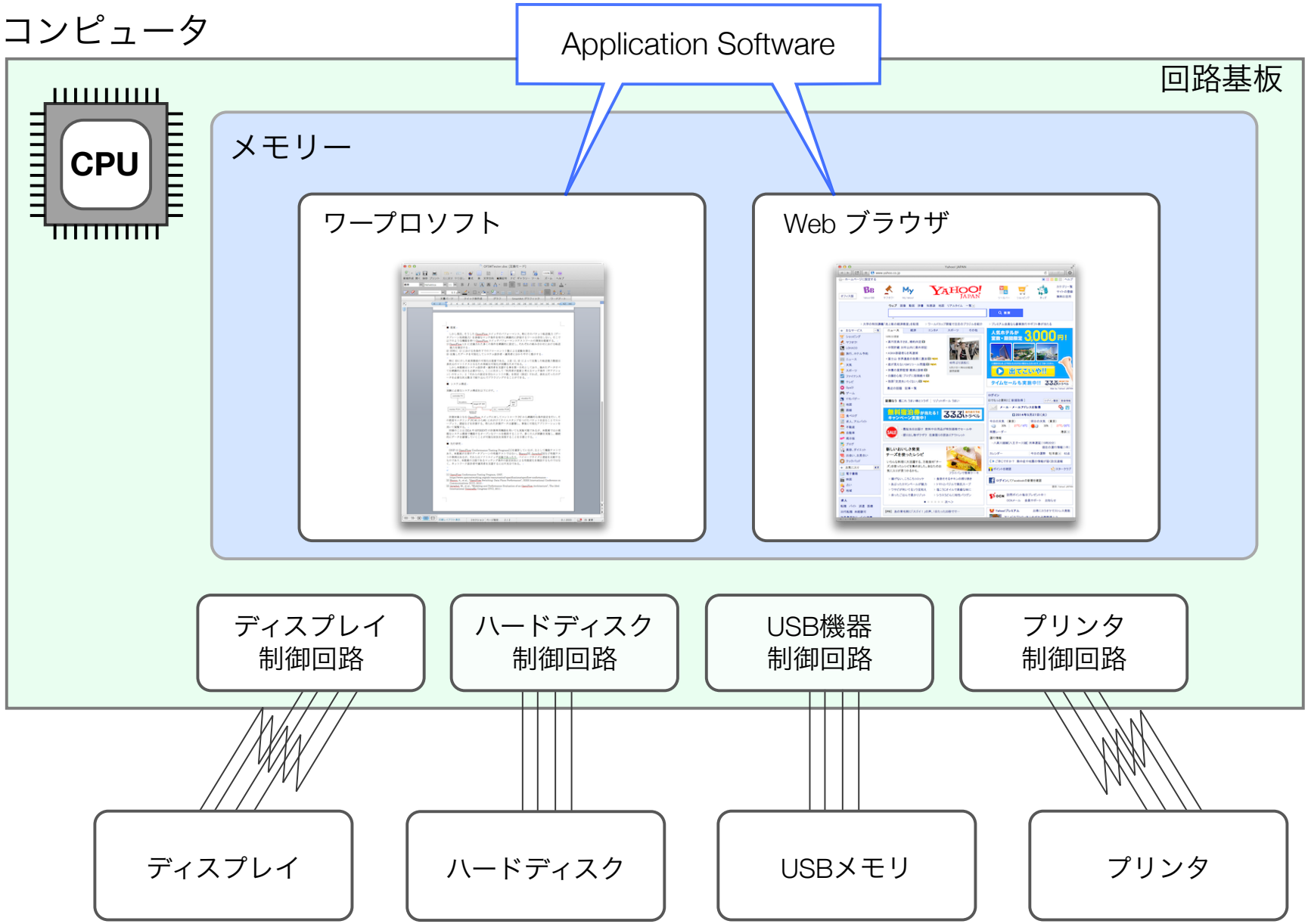
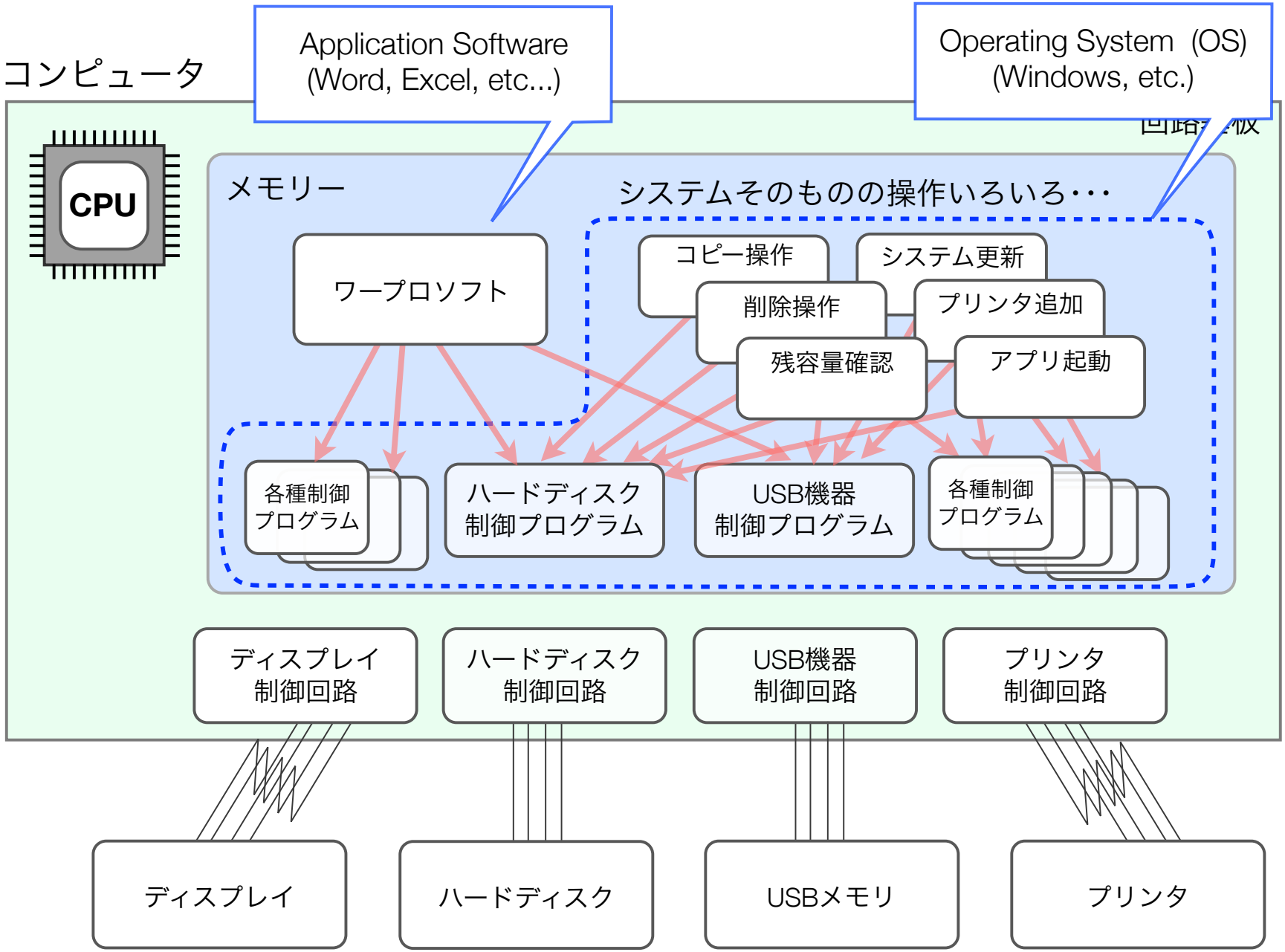
情報科学入門

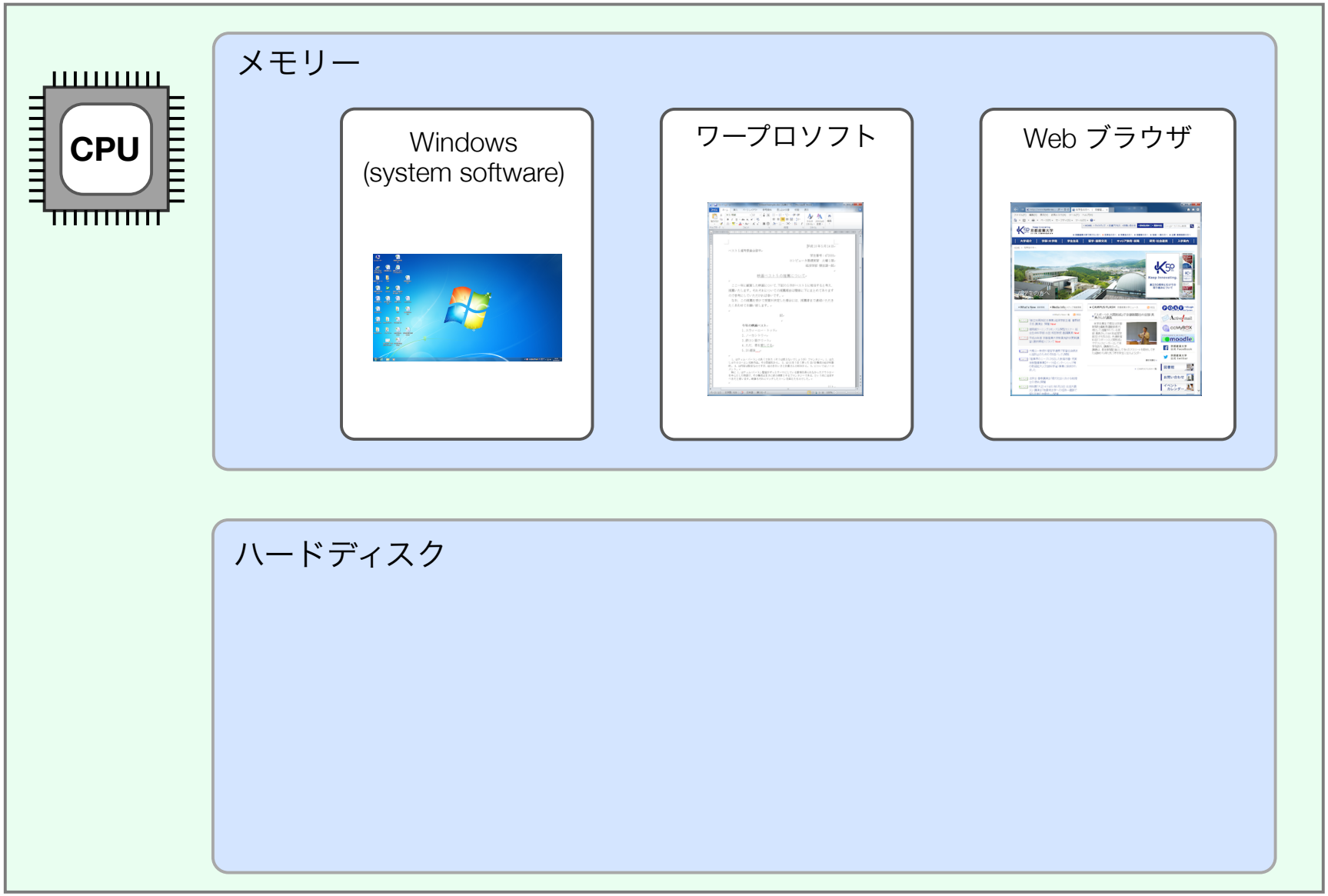
#11 機械語・高級言語・プログラムの実行

Yutaka Yasuda


コンピュータ
Application Software
回路基板
CPU
メモリー
ワープロソフト
Web ブラウザ
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


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

アプリケーションの起動



アプリケーションの起動

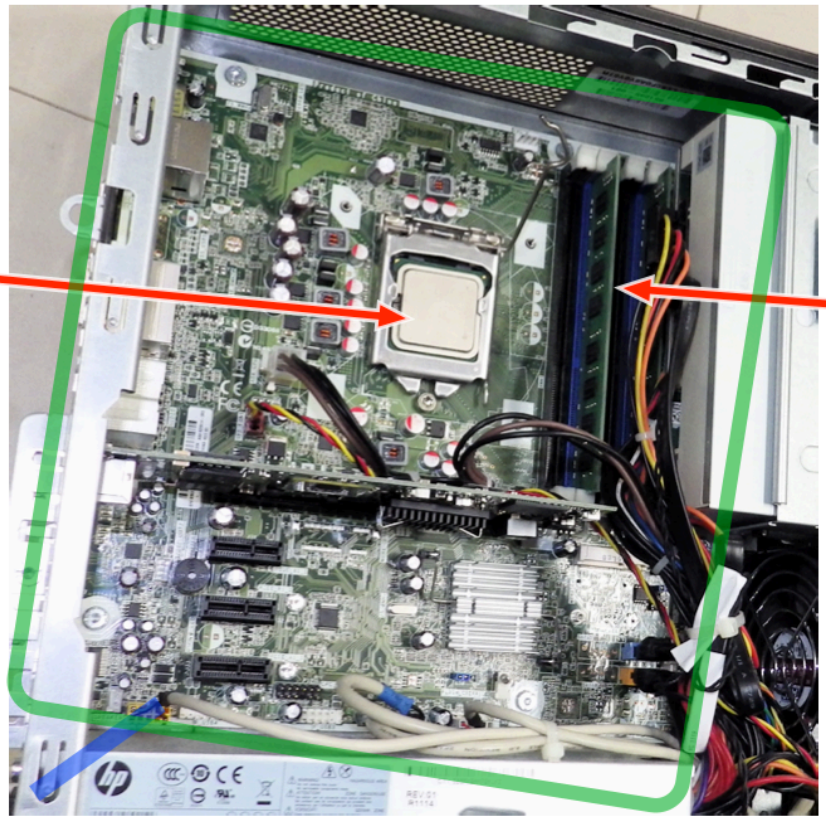
アプリケーションの起動

（#3 から再掲）

PC内部(クローズアップ)

(CPU 冷却ファンを外した状態)

CPU



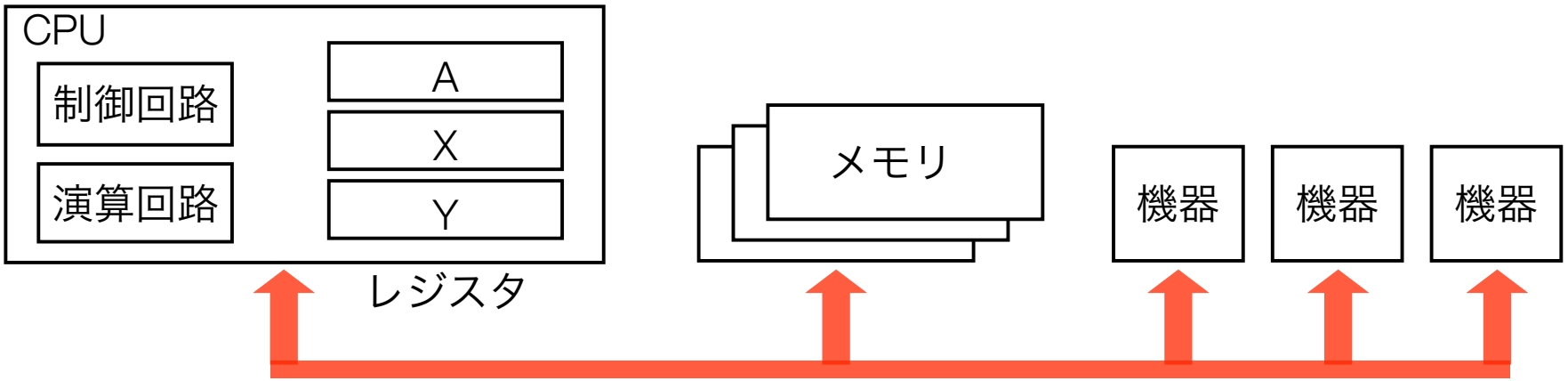
メモリ

緑色の部分は
マザーボード
（回路基板）

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

機械語の例

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
```

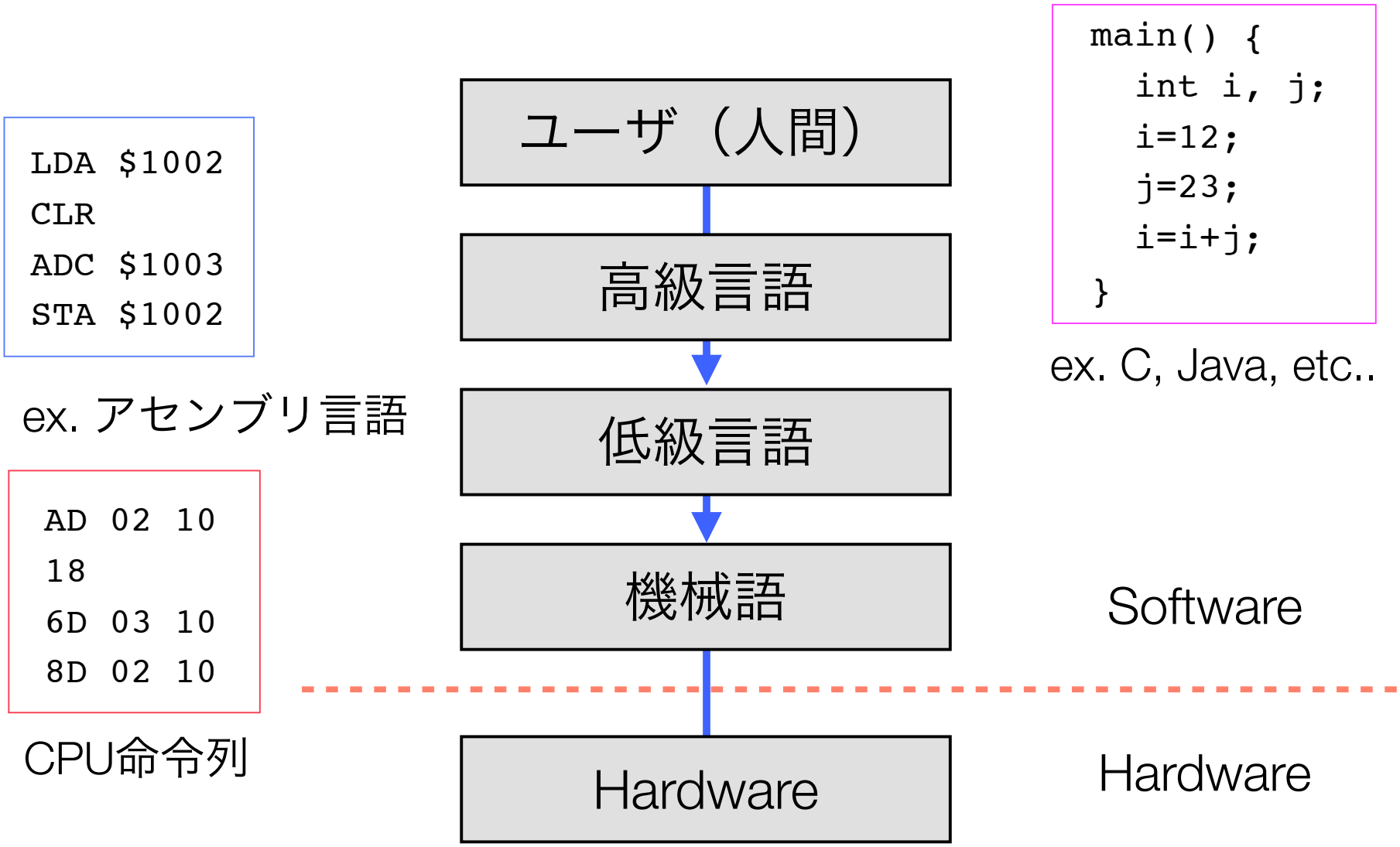
これが機械語

メモリの中ではこんな感じ

	AD
	02
	10
	18
	6D
	03
	10
	8D
	02
	10
	:
	:
1002	12
1003	23

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

プログラムの実行

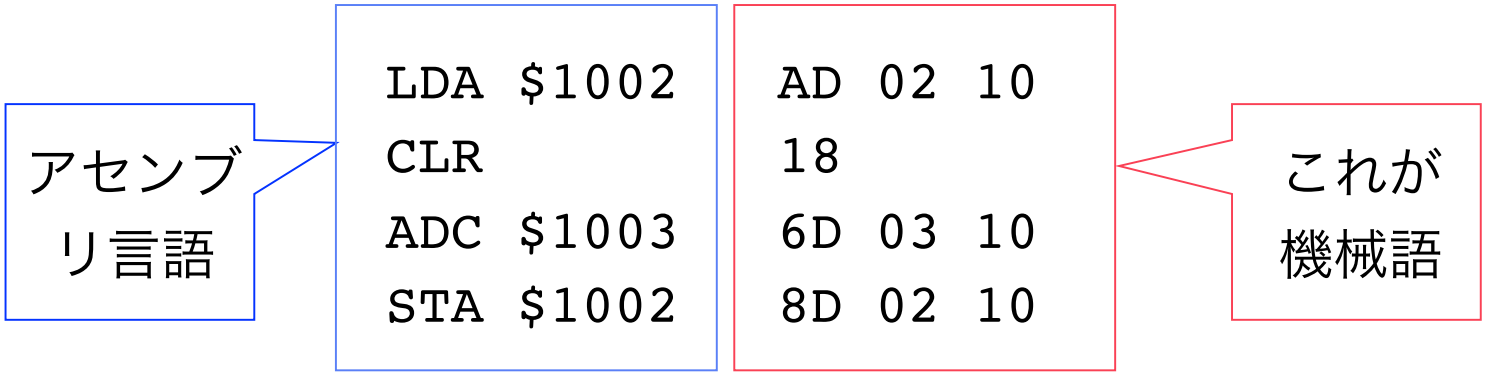

LDA $1002
CLR
ADC $1003
STA $1002
ex. アセンブリ言語
AD 02 10
18
6D 03 10
8D 02 10
CPU命令列
ユーザ（人間）
高級言語
低級言語
機械語
Hardware
main() {
int i, j;
i=12;
j=23;
i=i+j;
}
ex. C, Java, etc..
Software
Hardware

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：PowerPC 750

- 1998, IBM
- 32bit CPU
- iMac (original), PowerMac G3
- 233 - 700MHz clock
- 1GB memory
- レジスタ整数32, 実数32 本
- 典型的な32bit CPU

アセンブリ言語のCPU依存性

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

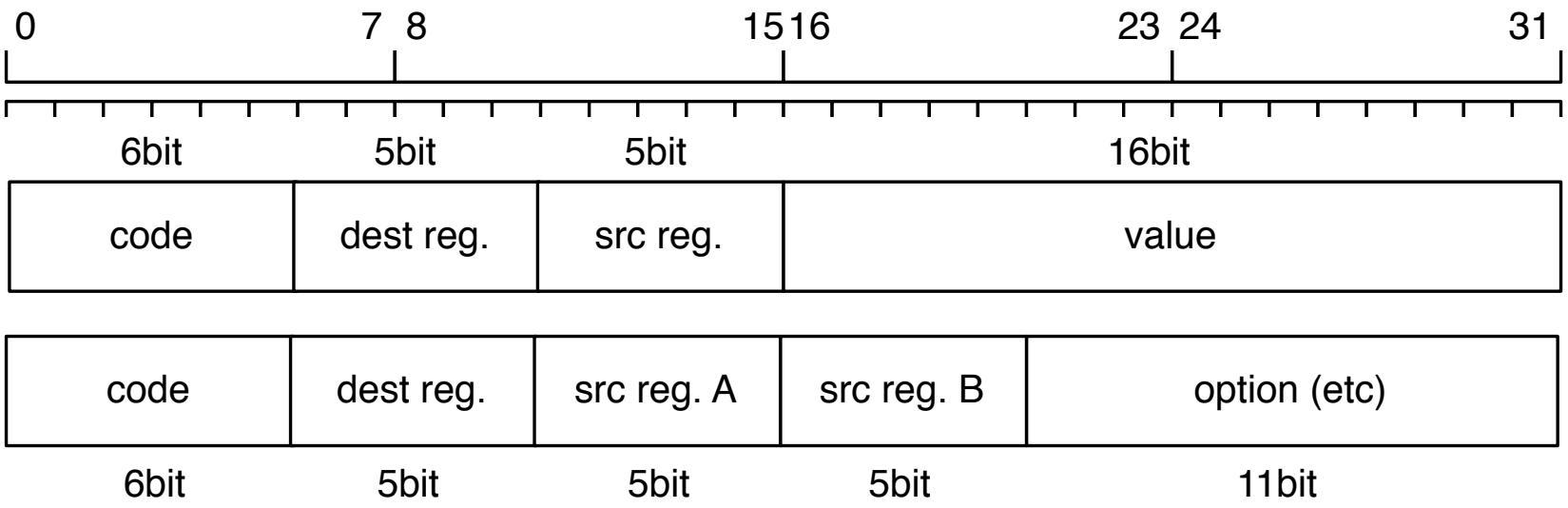
PowerPC 750

```
lwz r9,$1002    1002 番地の値(word)を Gr9 に
lwz r0,$1006    1006 番地の値(word)を Gr0 に
add r0,r9,r0     r0+r9 を r0 に
stw r0,$1002    結果 r0 を 1002 番地に
```

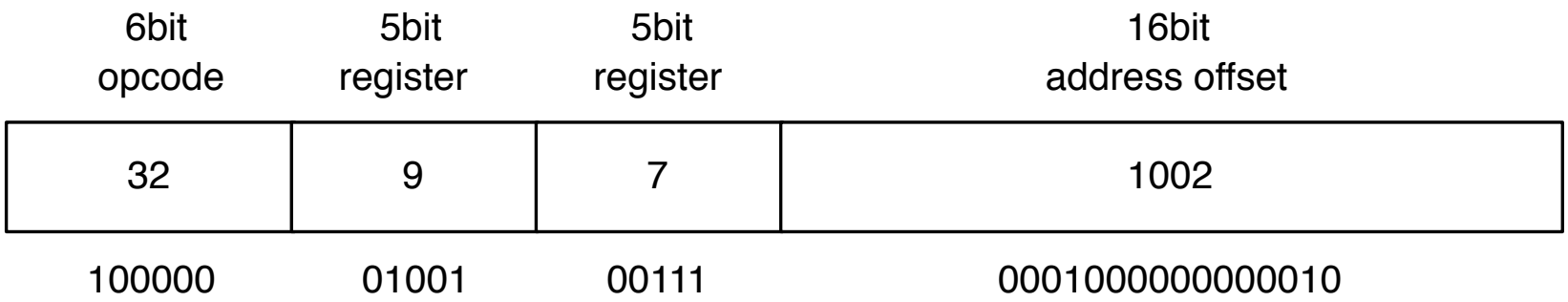
MOS Technology 6502

```
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
```

参考：PowerPC 750 のマシン語 (opcode)



lwz r9,$1002(r7)



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

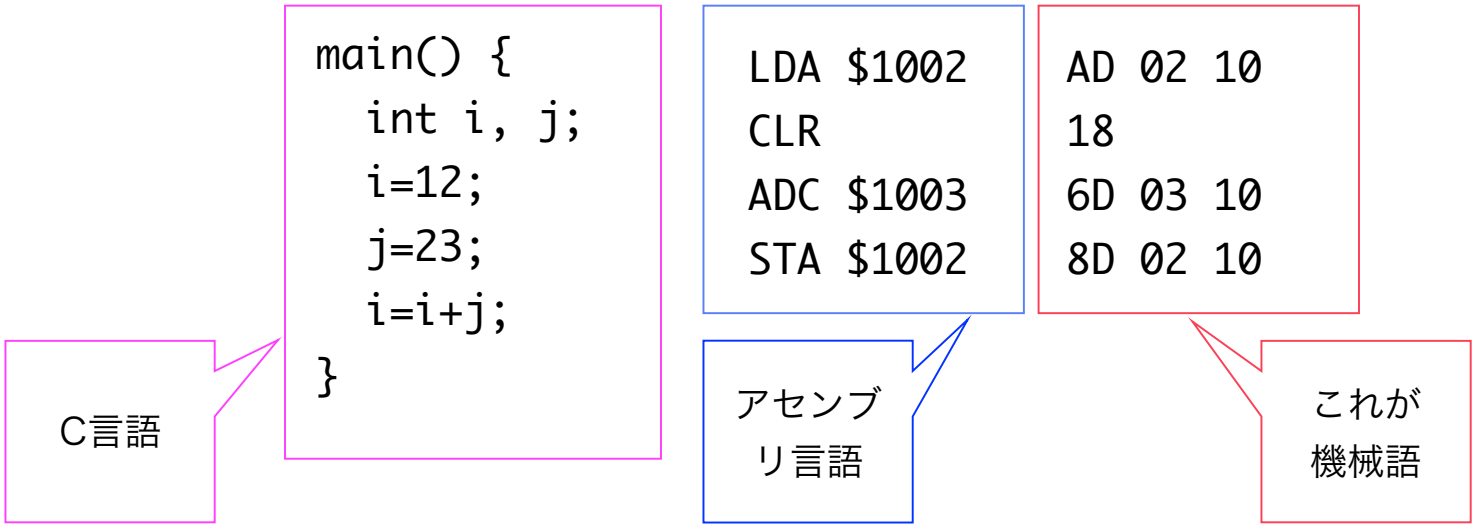
  長持ちして欲しい

  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

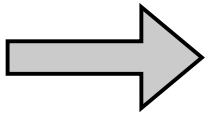
  機械語によって C 言語プログラムと等価な振舞いをさせなければ

  →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;
}
```



PowerPC 750 アセンブリ言語

```
.text
        .align 2
        .globl _main
_main:
        stmw r30,-8(r1)
        stwu r1,-64(r1)
        mr r30,r1
        li r0,1954          Gr0に 1954
        stw r0,32(r30)      Gr0 を Gr30 から 32 バイト先(i)に格納
        li r0,1959          Gr0に 1959
        stw r0,36(r30)      こんどは 36 バイト先に格納
        lwz r9,32(r30)      i を Gr9 に
        lwz r0,36(r30)      j を Gr0 に
        add r0,r9,r0        Gr0+Gr9 を Gr0 に
        stw r0,32(r30)      結果 Gr0 を i に
        mr r3,r0
        lwz r1,0(r1)
        lmw r30,-8(r1)
        blr
```

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

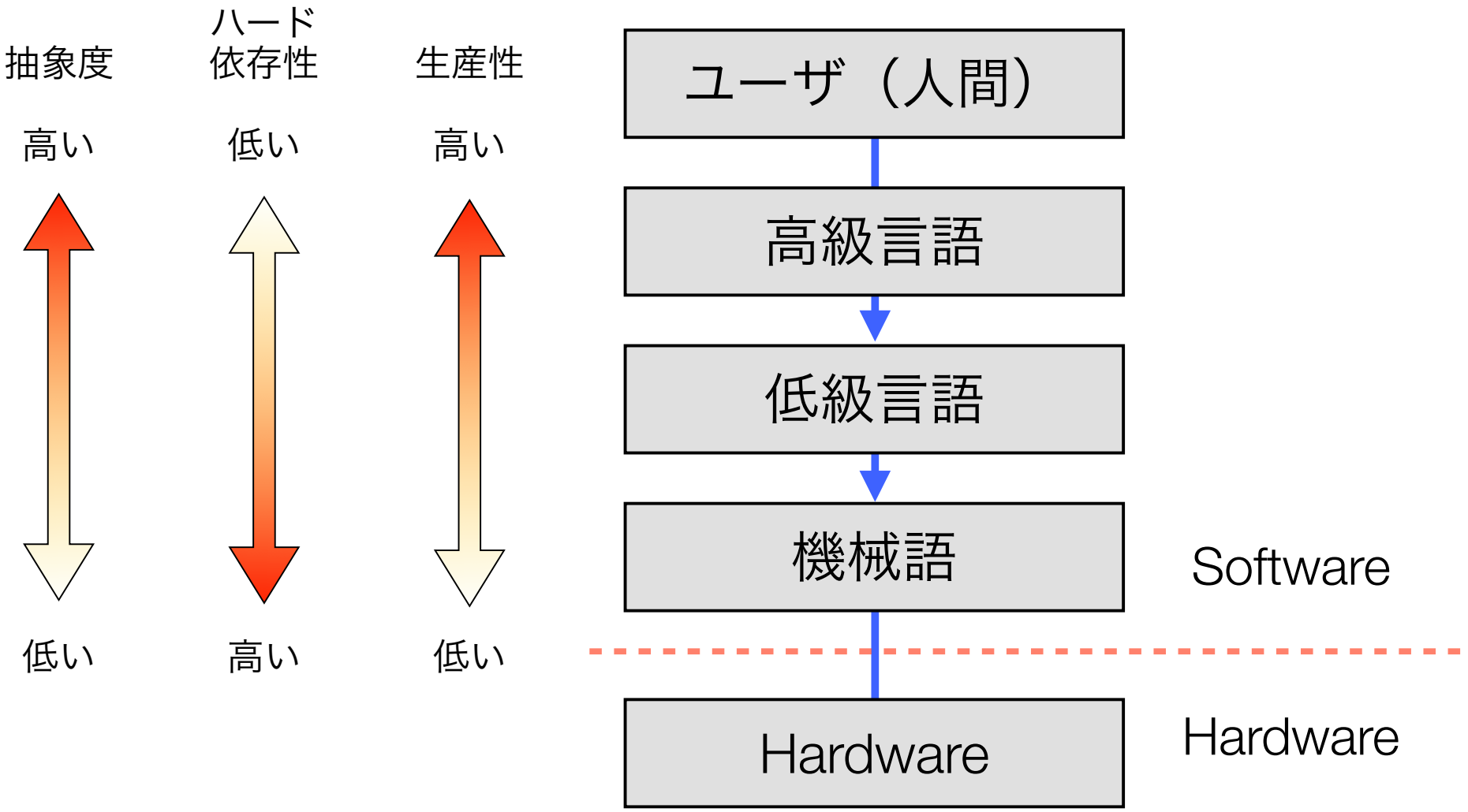
```
a=b+c
```

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）


抽象度
ハード
依存性
生産性
高い
低い
高い
低い
高い
低い
ユーザ（人間）
高級言語
低級言語
機械語
Hardware
Software
Hardware

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

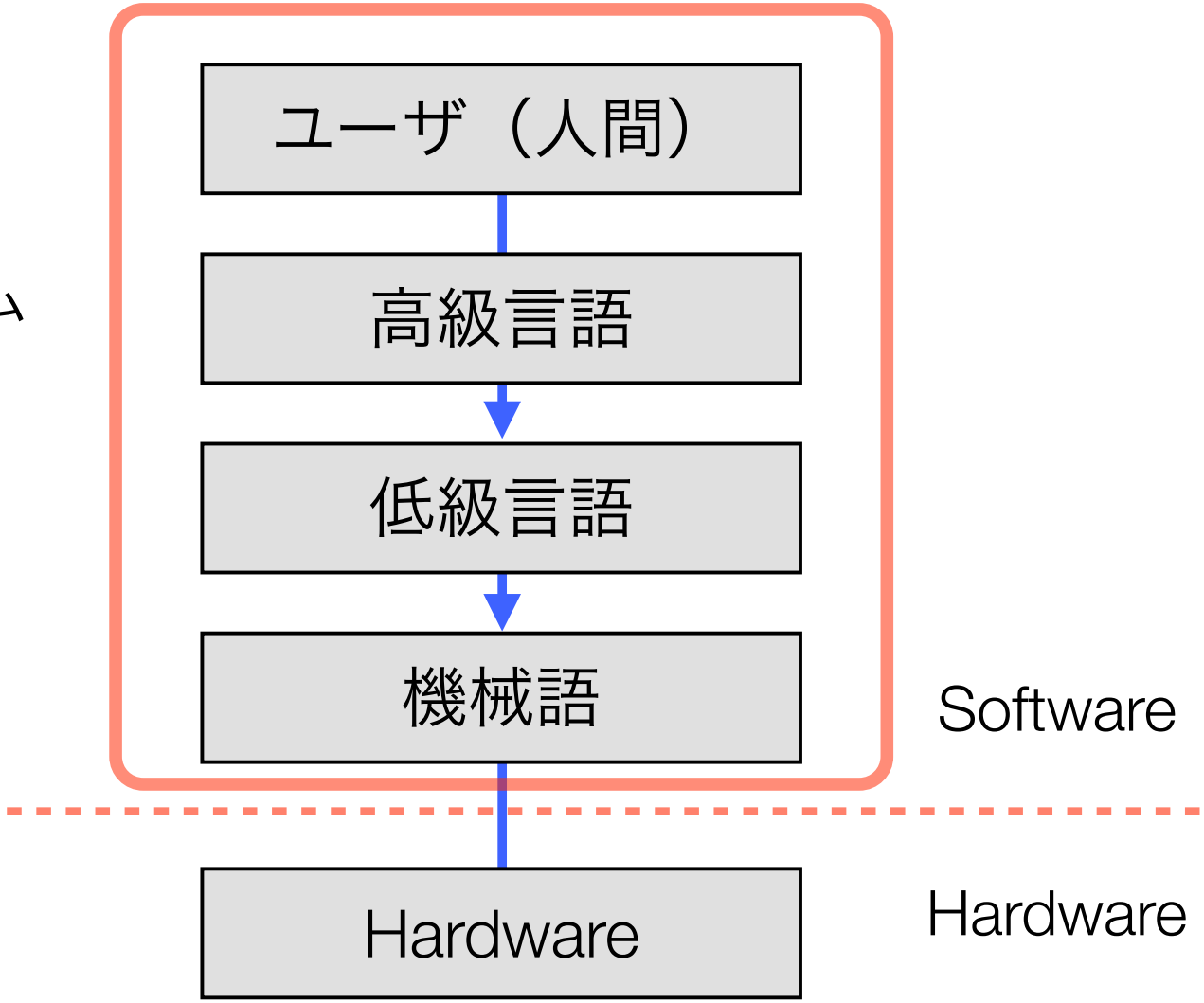
  ソート

  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
  アルゴリズムをプログラミング言語で書き下したもの
- 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 に繰り込む」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

アルゴリズムから機械語へ

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

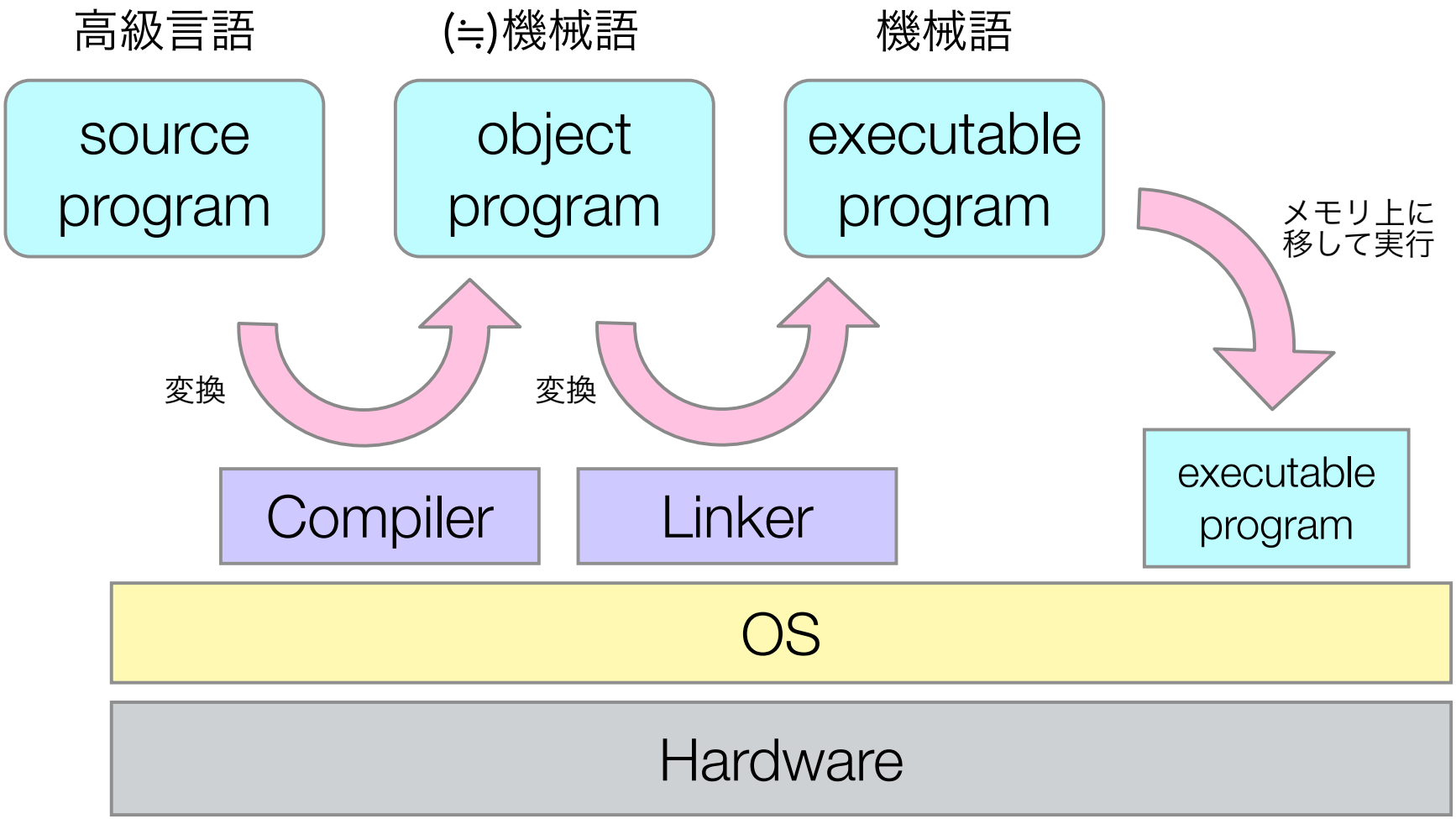
- 機械語に変換して実行

  変換前：原始プログラム (Source program)

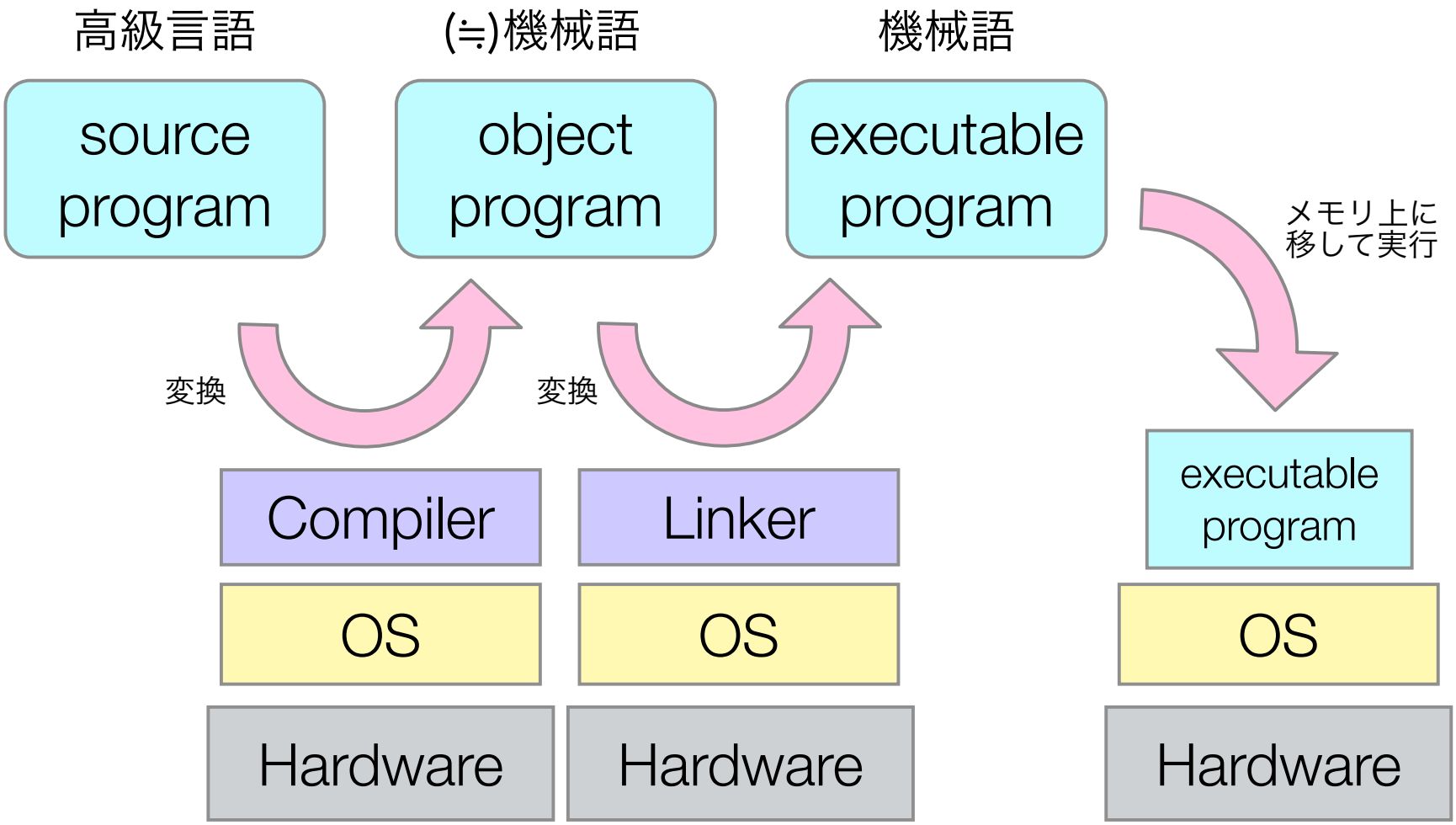
  変換後：目的プログラム (Object program)

- 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


高級言語
(≒)機械語
機械語
source program
object program
executable program
メモリ上に移して実行
変換
変換
Compiler
Linker
executable program
OS
Hardware

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