ca-6. プログラムカウンタ

（コンピュータ・アーキテクチャ演習）

URL: https://www.kkaneko.jp/cc/ca/index.html

金子邦彦

アウトライン

6-1 プログラムカウンタの振る舞い

6-2 Visual Studio でプログラムカウンタの表示

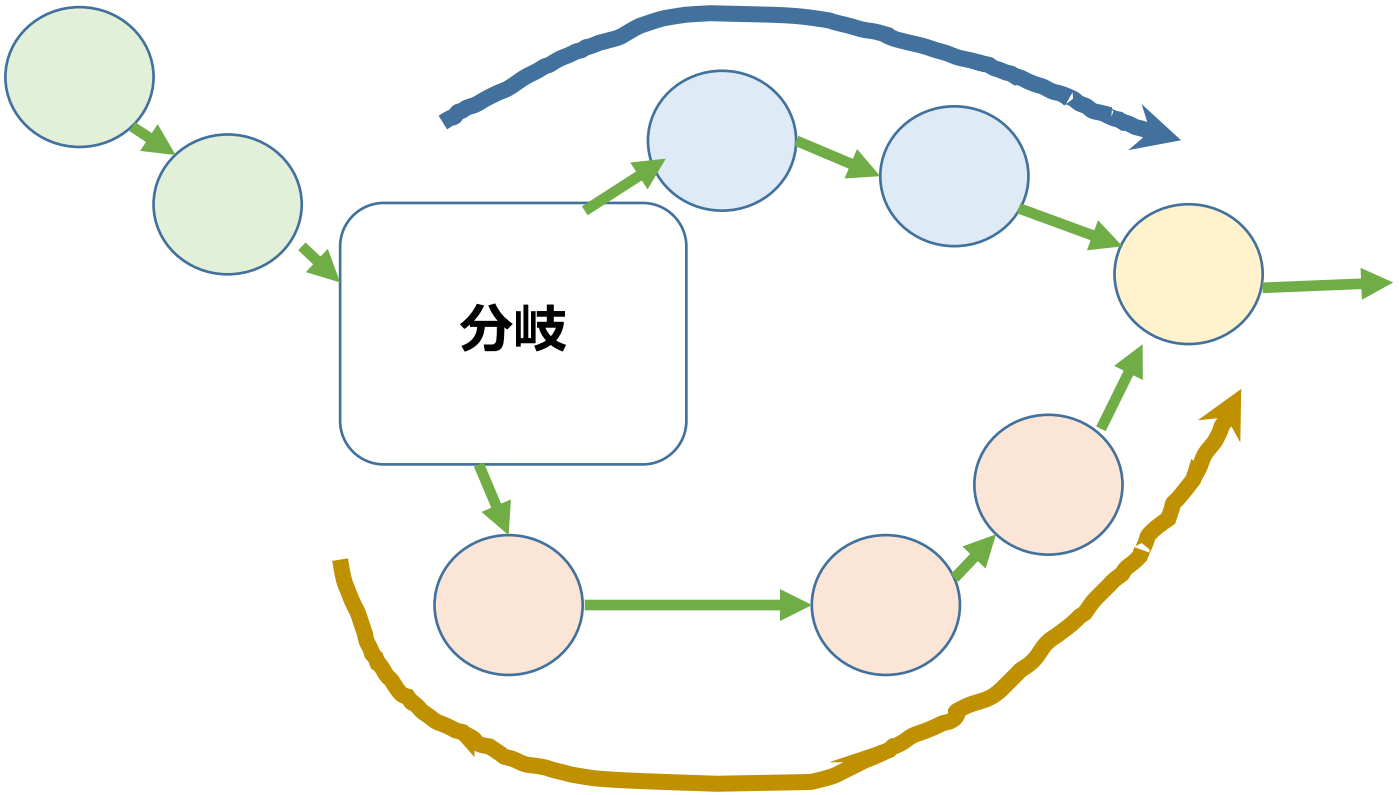
6-3 命令実行サイクル

6-1 プログラムカウンタの振る舞い

6-1 プログラムカウンタとは

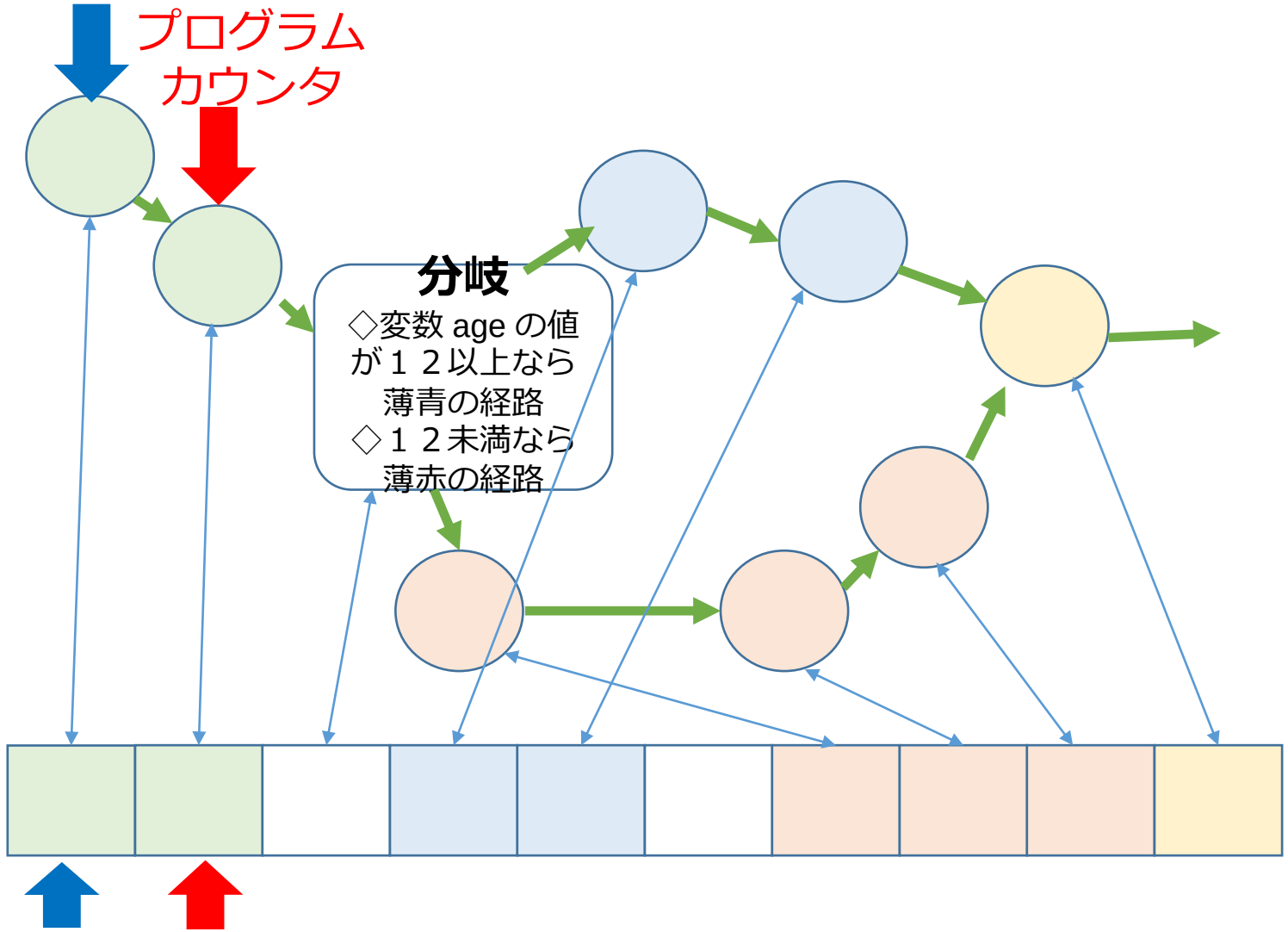
- プログラムカウンタには，次に実行すべき命令のアドレスが入っている

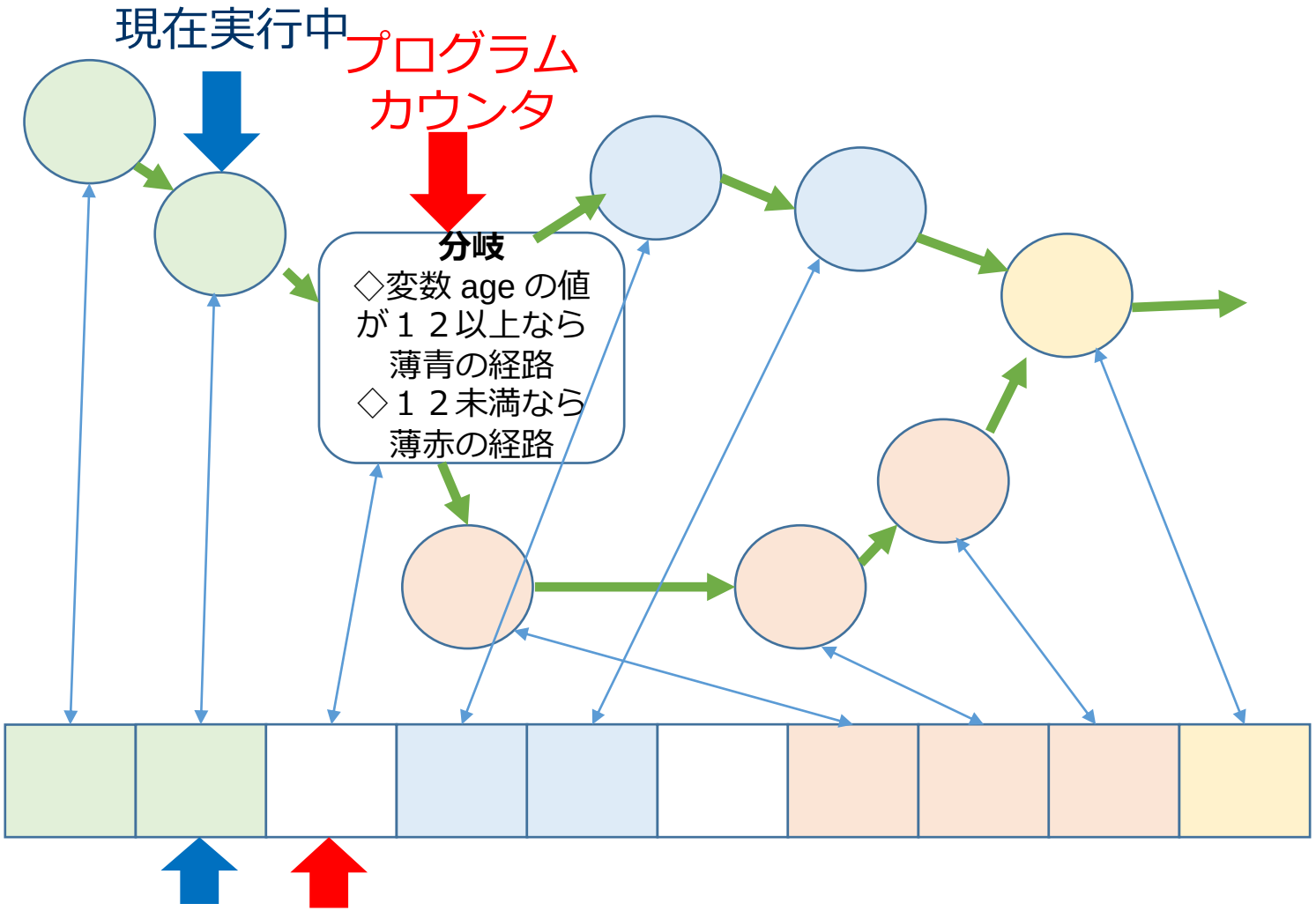
変数 age の値が 12 以上のときの経路

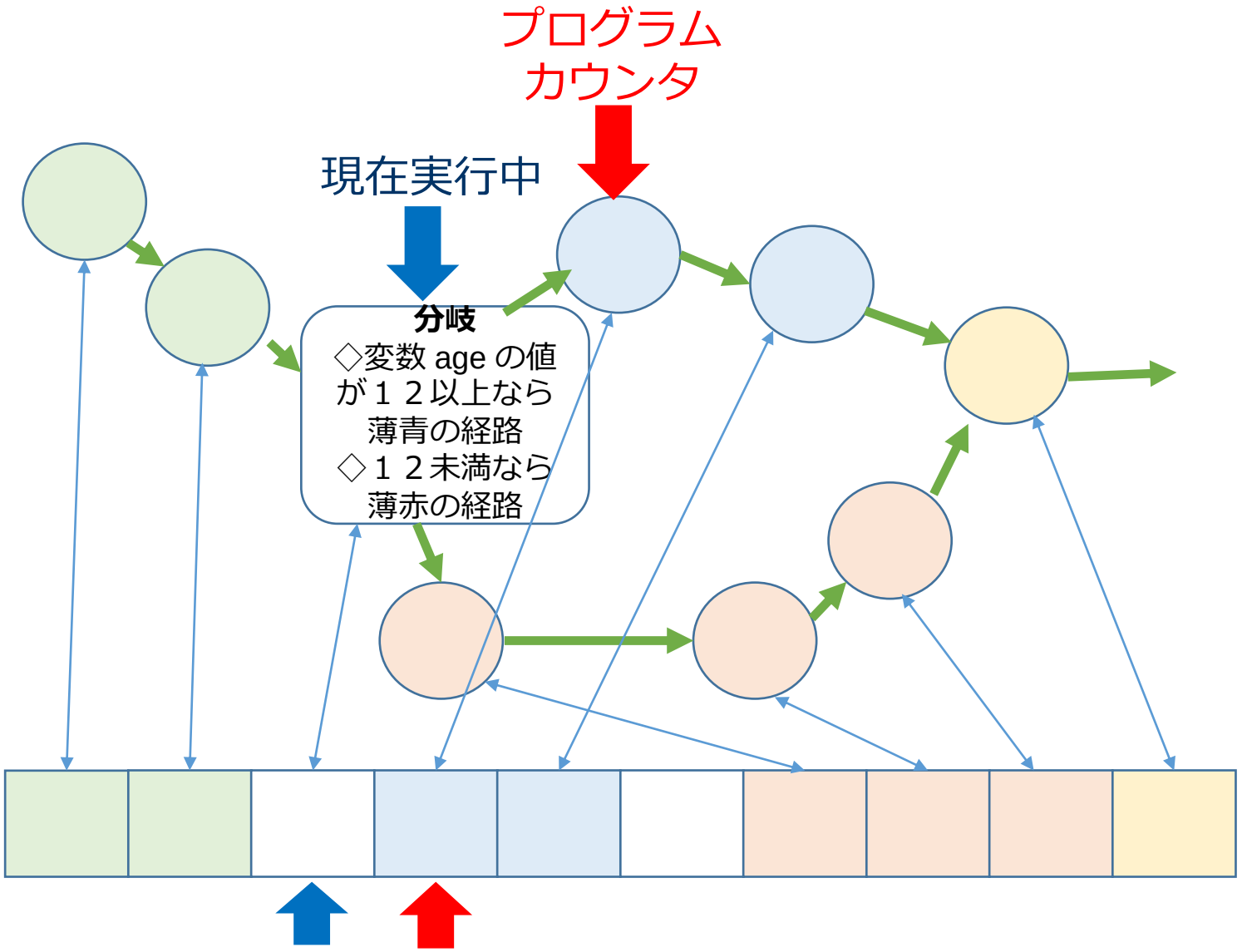


変数 age の値が 12 未満のときの経路

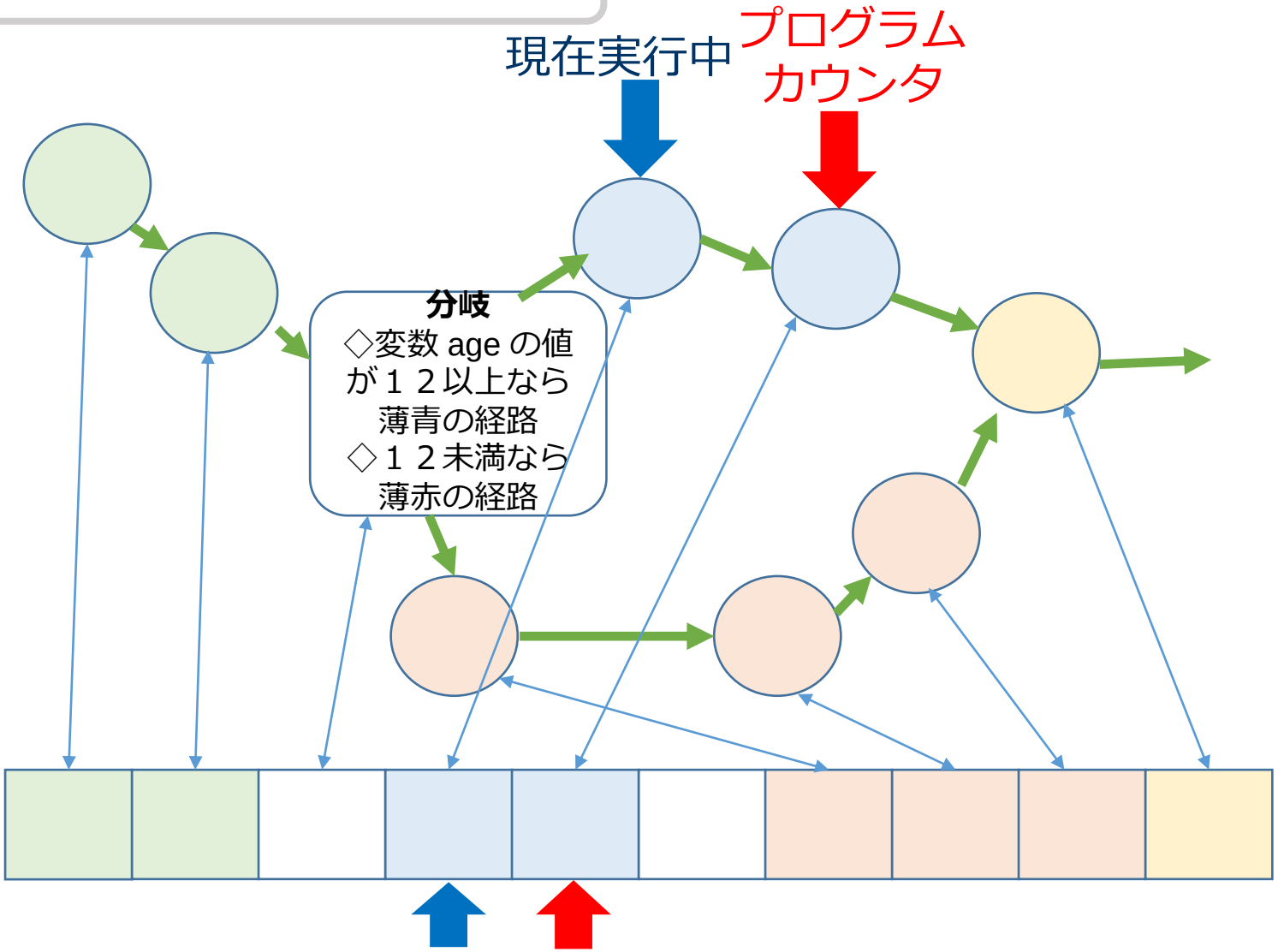
現在実行中



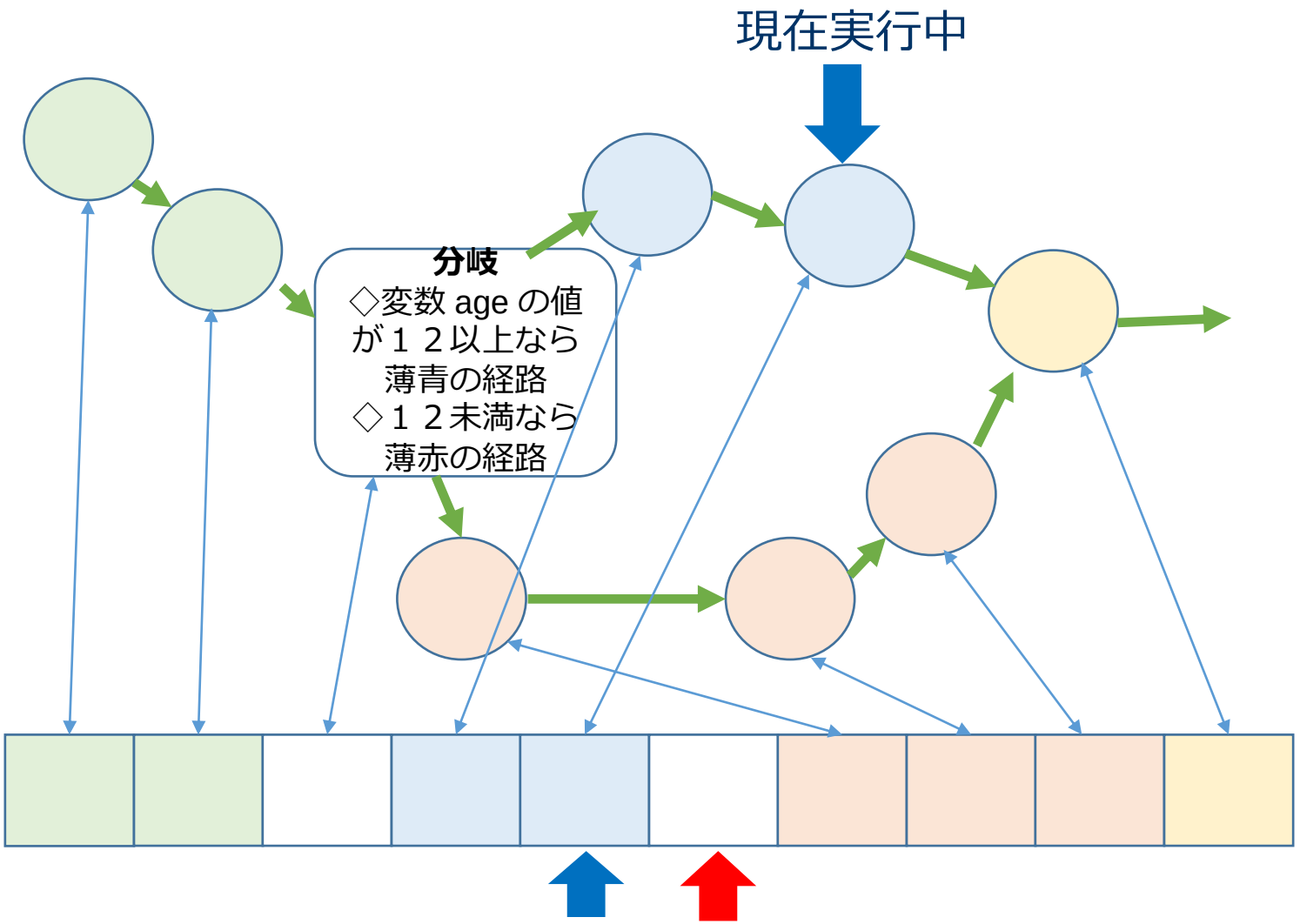

現在実行中
プログラム
カウンタ
分岐
◇変数 age の値
が１２以上なら
薄青の経路
◇１２未満なら
薄赤の経路


プログラム
カウンタ
現在実行中
分岐
◇変数 age の値
が１２以上なら
薄青の経路
◇１２未満なら
薄赤の経路

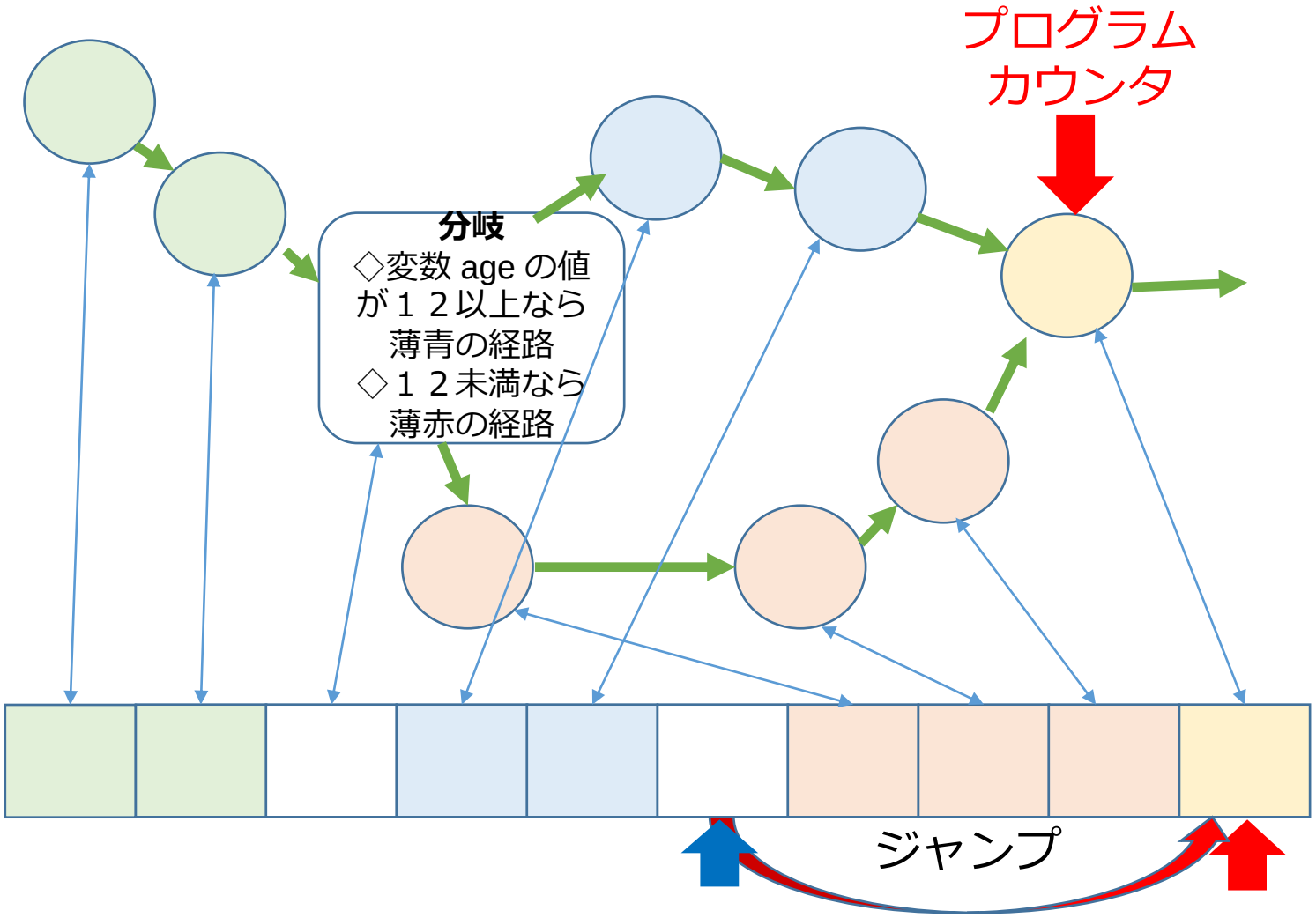
age の値が１２以上のとき



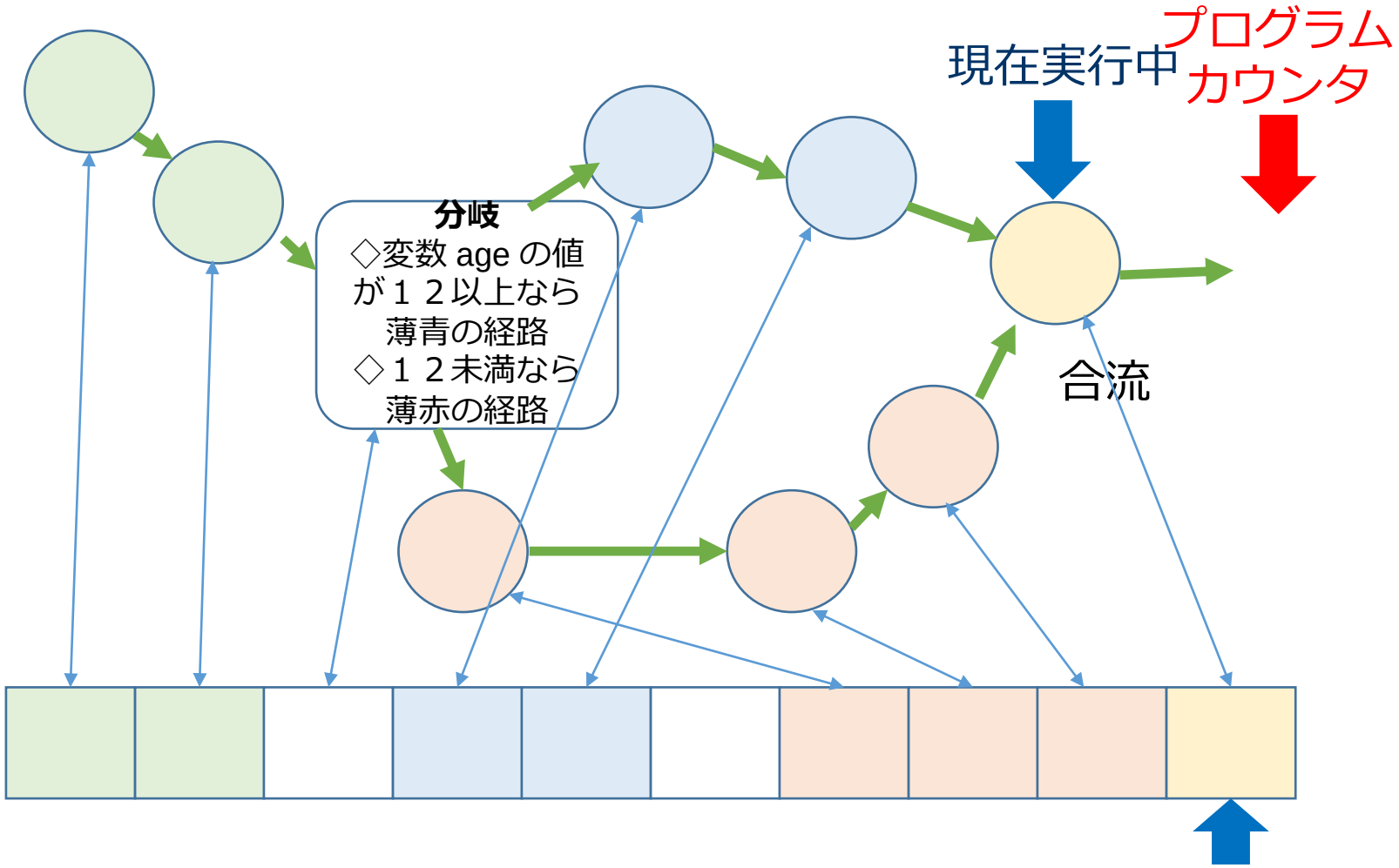
age の値が１２以上のとき



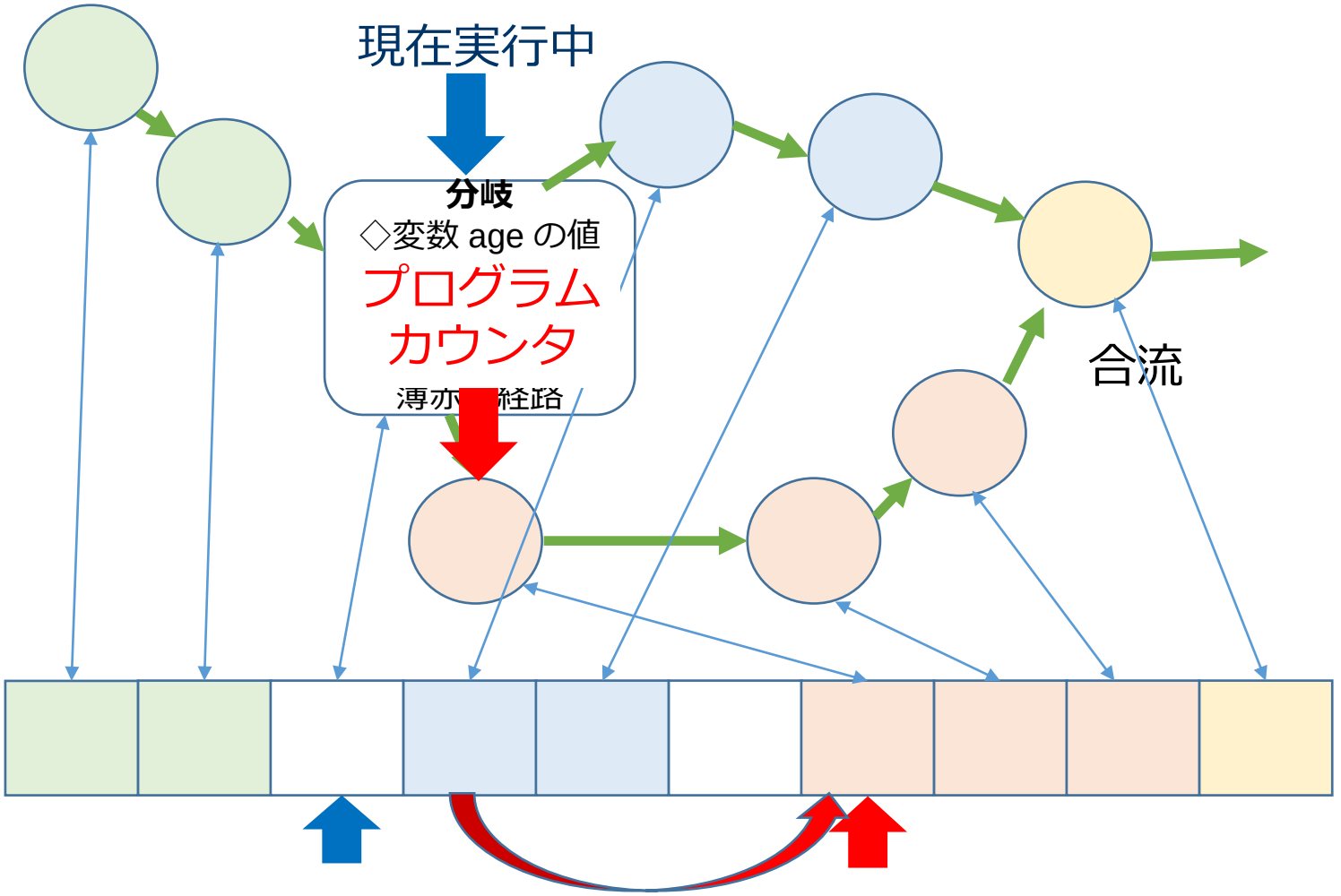
age の値が１２以上のとき



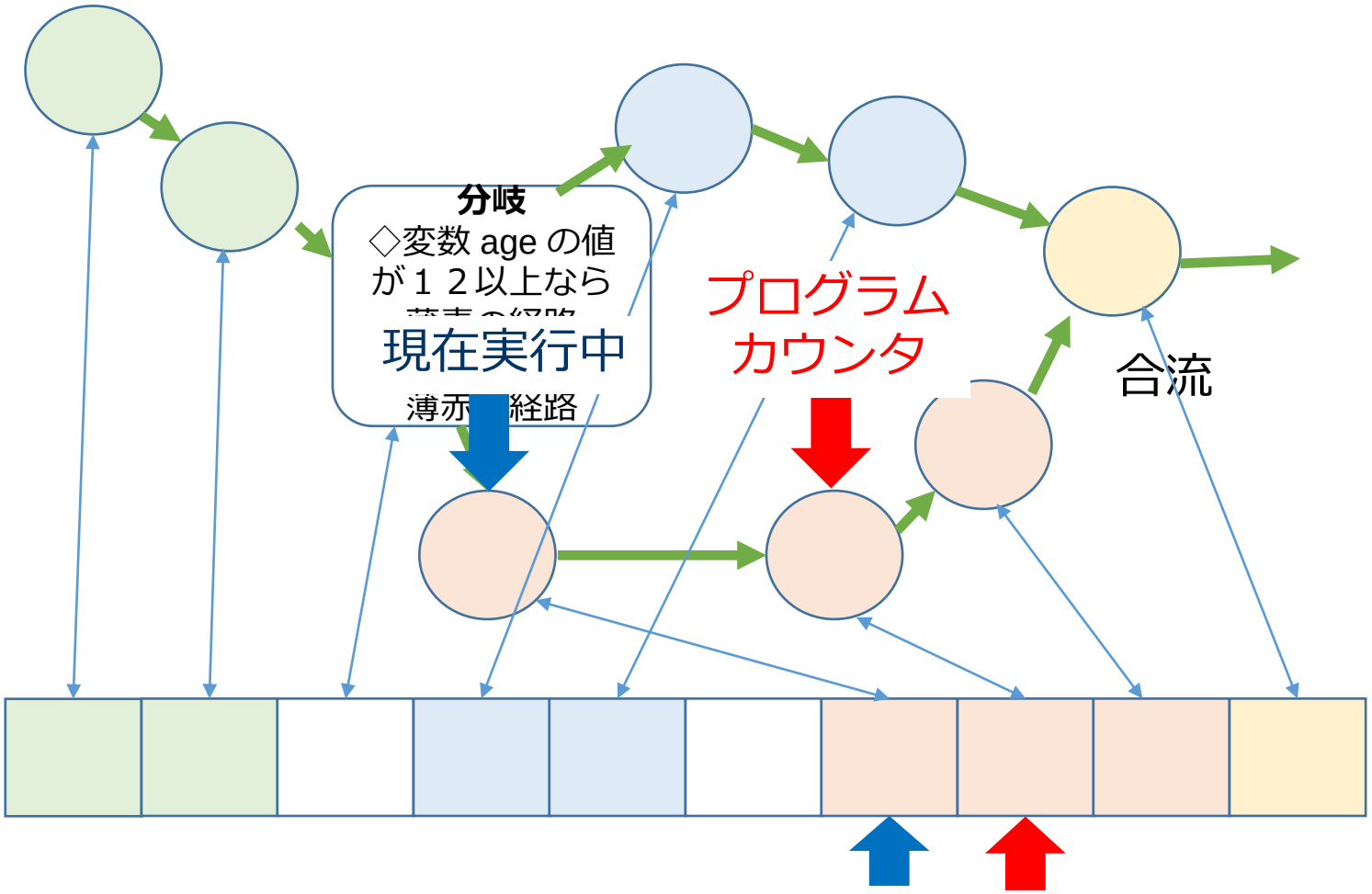
age の値が１２以上のとき



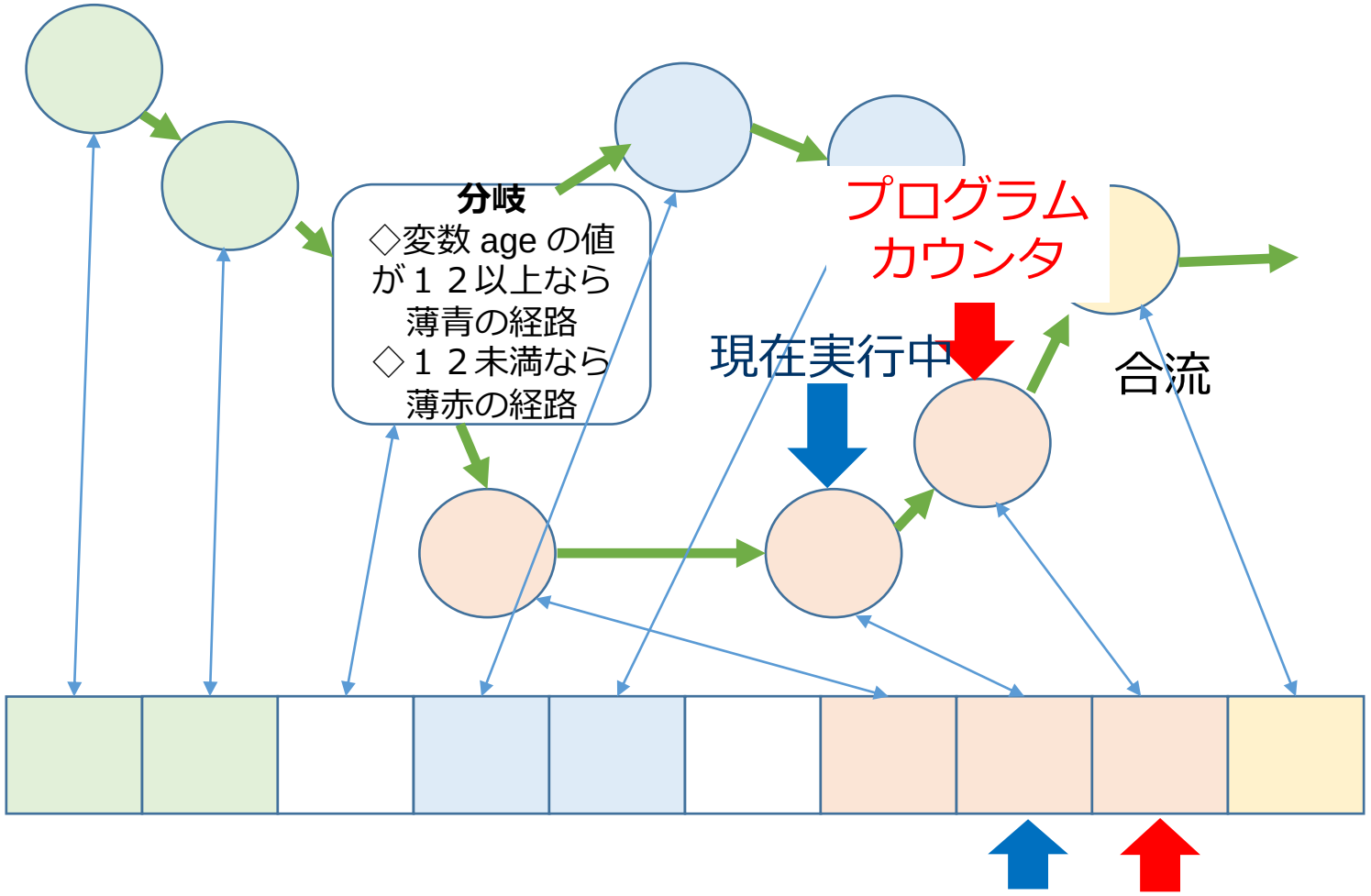
age の値が１２未満のとき



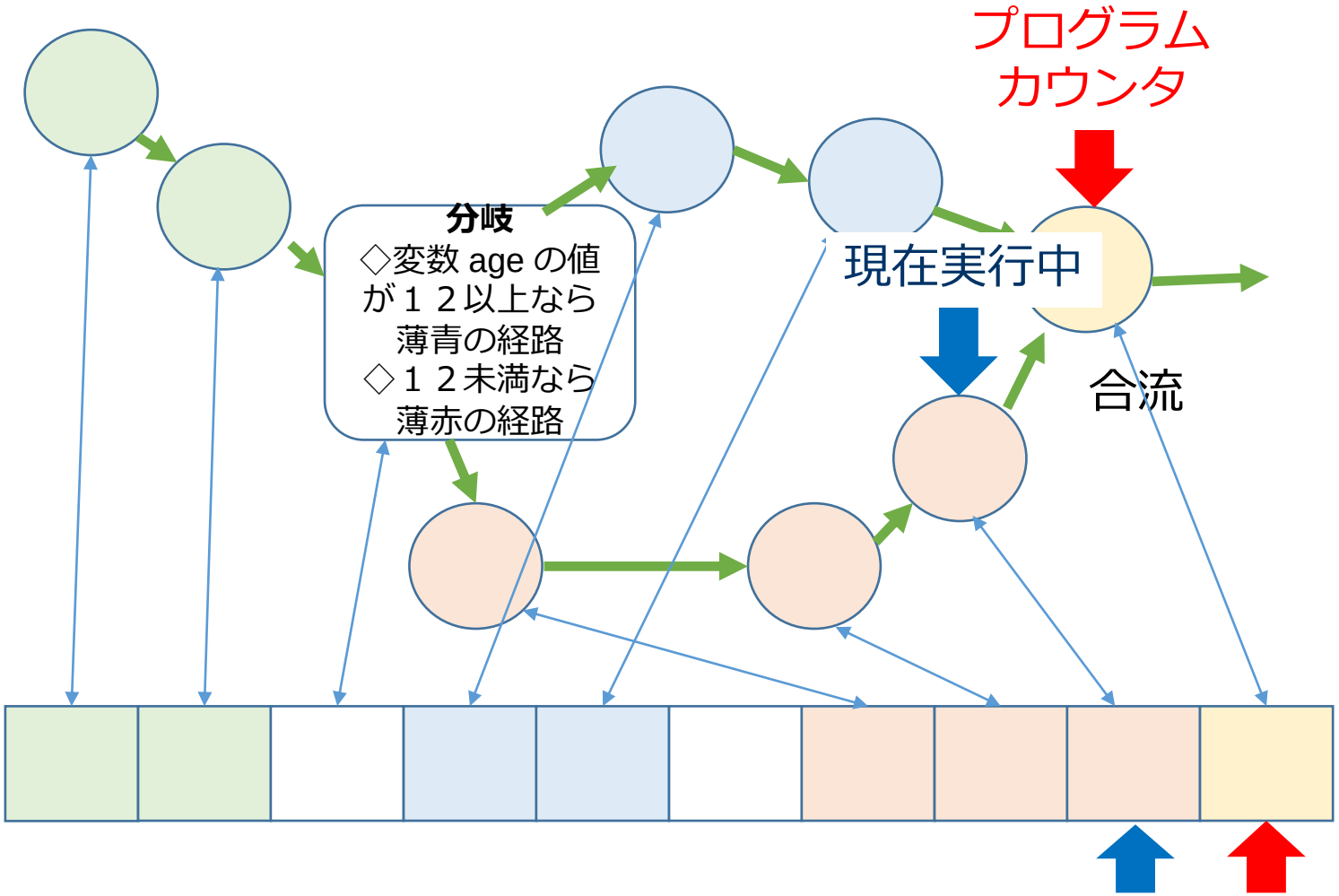
age の値が１２未満のとき



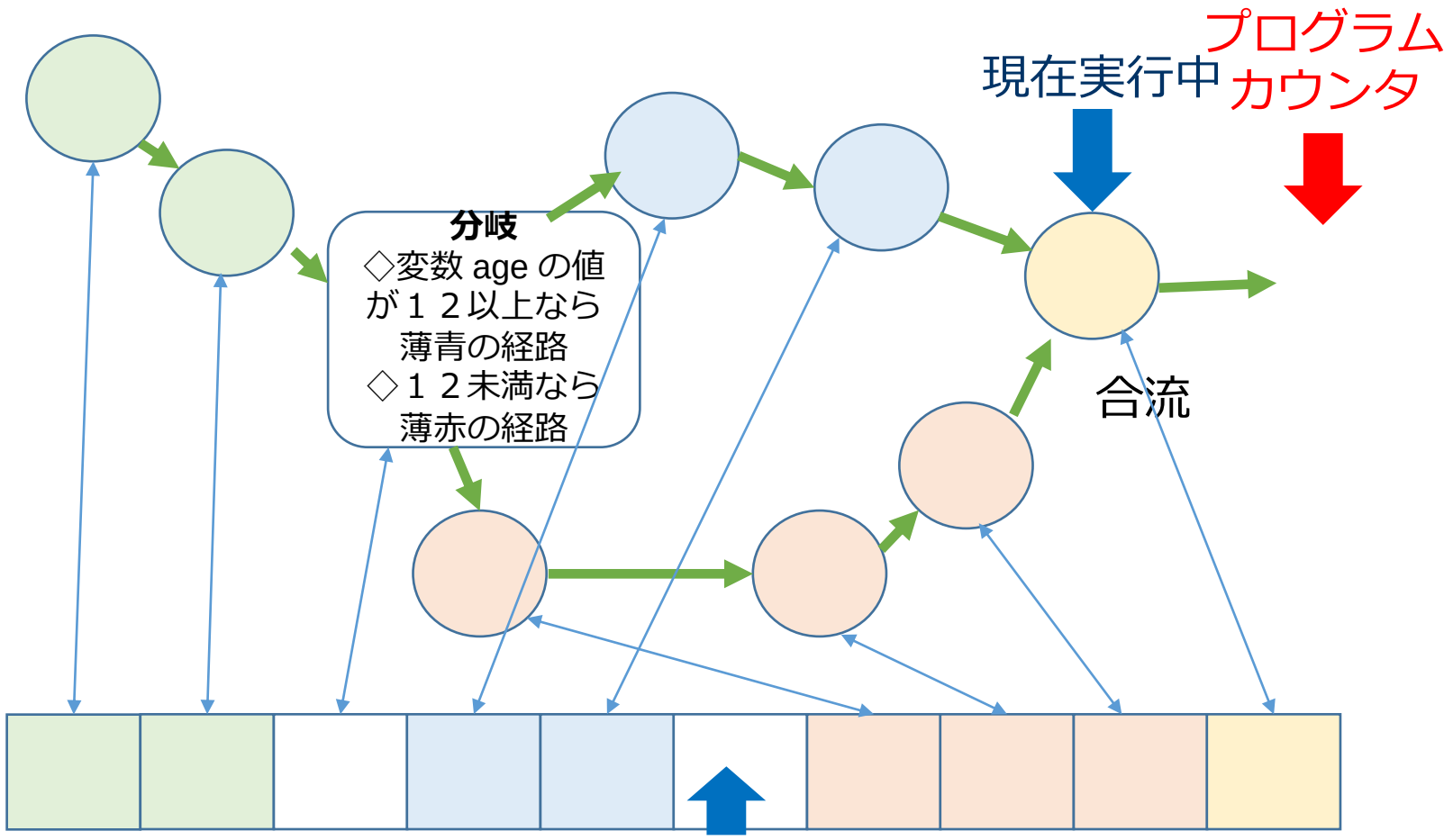
age の値が１２未満のとき



age の値が１２未満のとき

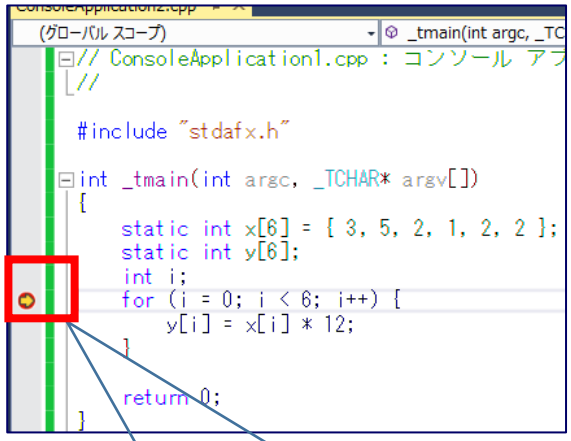


age の値が１２未満のとき

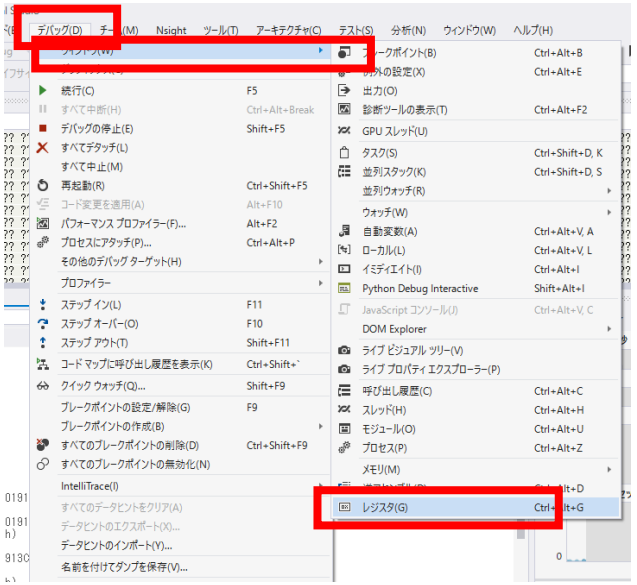


6-2 Visual Studio で プログラムカウンタの表示

プログラムカウンタの表示操作



デバッガーを起動済みで，プログラムの実行が中断しているときに・・・



① 「デバッグ」
→「ウインドウ」→「**レジスタ**」

```
EAX = CCCCCCCC EBX = 7EFDE000 ECX = 00000000 EDX = 00000001 ESI = 00000000 EDI = 001BFA24 EIP = 00C3139E
  ESP = 001BF958 EBP = 001BFA24 EFL = 00000200

0x00c38130 = 00000000
```

② レジスタが表示される

演習

- Visual Studio を起動しなさい
- Visual Studio で，Win32 コンソールアプリケーション用プロジェクトを新規作成しなさい

プロジェクトの「名前」は何でもよい

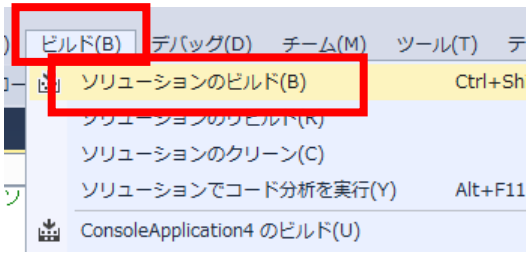
- Visual Studioのエディタを使って，ソースファイルを編集しなさい

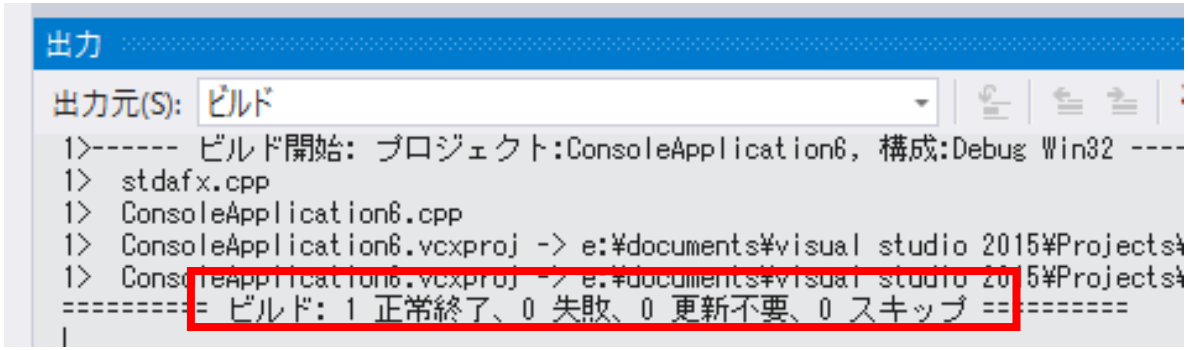
```
int main()
{
    static int age, p;
    age = 20;
    if (age > 12)
        p = 1800;
    else
        p = 500;
    return 0;
}
```

追加

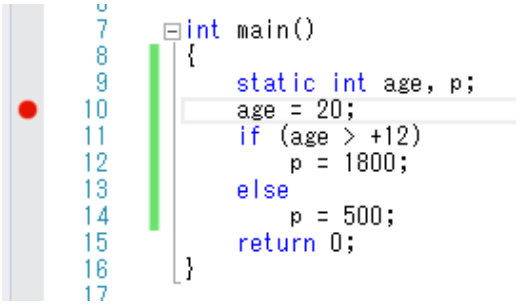
- ビルドしなさい．ビルドのあと「１　正常終了，０　失敗」の表示を確認しなさい

→　表示されなければ，プログラムのミスを自分で確認し，修正して，ビルドをやり直す



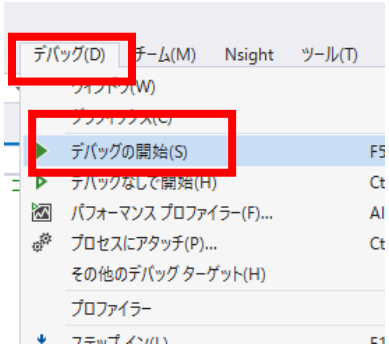


- Visual Studioで「**age = 20;**」の行に，**ブレークポイント**を設定しなさい



赤丸がブレークポイントの印

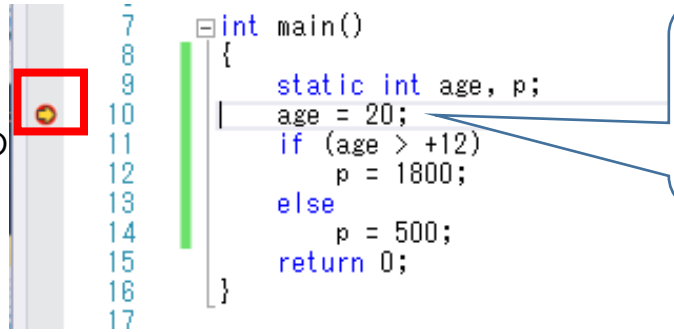
- Visual Studioで，デバッガーを起動しなさい.



「デバッグ」
→ 「**デバッグ開始**」

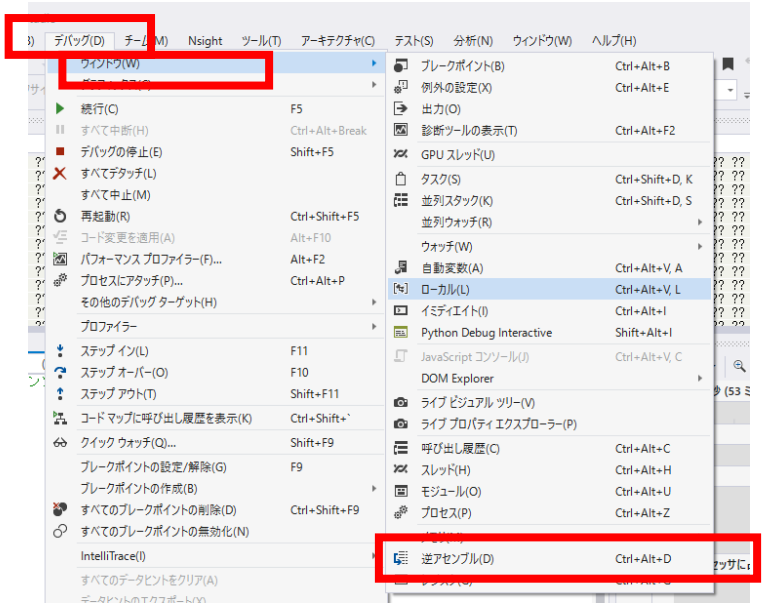
- 「age = 20;」の行で，実行が中断することを確認しなさい
- あとで使うので，中断したままにしておくこと

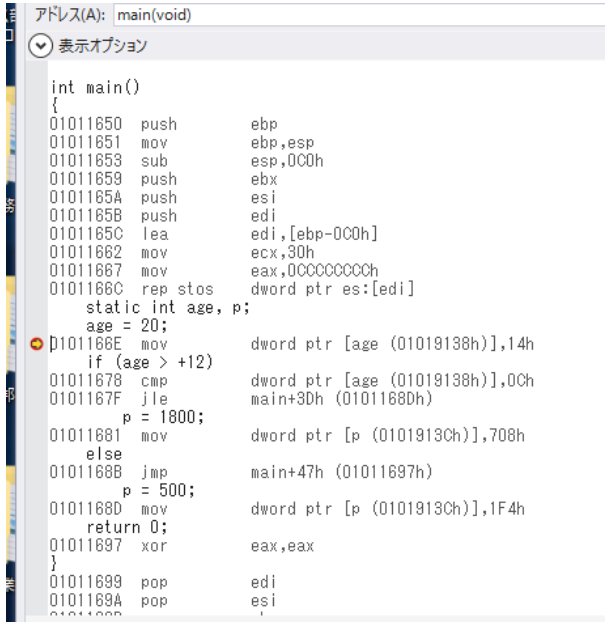
黄色の矢印



「age = 20;」の行で実行が**中断**している

- 「age = 20;」の行で，実行が中断した状態で，逆アセンブルを行いなさい．





① 「デバッグ」→「ウインドウ」→「**逆アセンブル**」

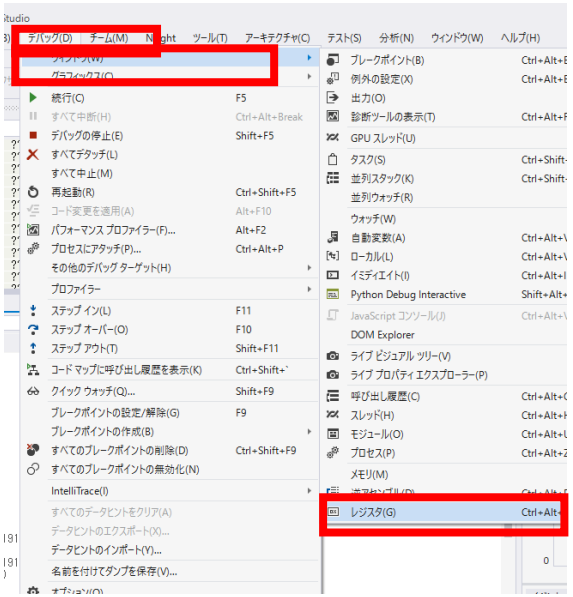
② 逆アセンブルの結果が表示される

- 「age = 20;」の行で，実行が中断した状態で，レジスタの中身を表示させなさい．手順は次の通り．

EIP は**プログラムカウンタ**

```
01011166E  mov         dword ptr [age (01019138h)],14h
    if (age > +12)
01011678  cmp         dword ptr [age (01019138h)],0Ch
0101167F  jle         main+3Dh (0101168Dh)
        p = 1800;
01011681  mov         dword ptr [p (0101913Ch)],708h
    else
01011688  jmp         main+47h (01011697h)
        p = 500;
0101168D  mov         dword ptr [p (0101913Ch)],1F4h
    return 0;
01011697  xor         eax,eax
```

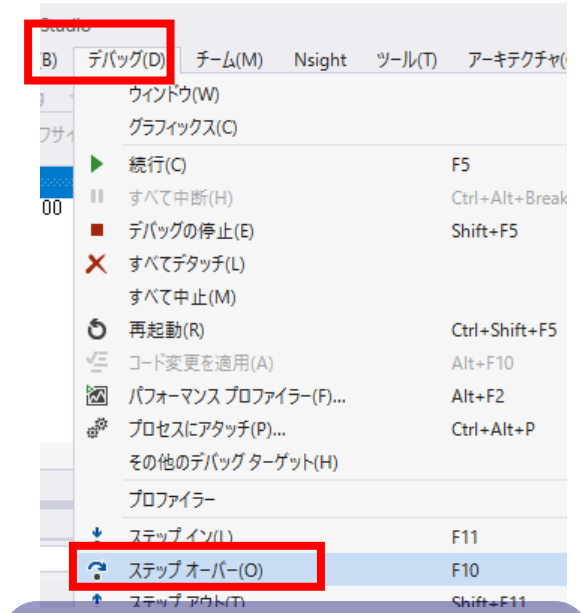
デバッガーを**起動済み**で，プログラムの実行が中断しているときに・・・



① 「デバッグ」
→「ウインドウ」→「**レジスタ**」

② レジスタが表示される．
EIP に注目

- ステップオーバーの操作を１回ずつ行いながら，レジスタウインドウの中の EIP の変化を確認しなさい．



「デバッグ」
→「ステップオーバー」
（あるいは F10 キー）

```
    age = 20;
01011660  mov         dword ptr [age (01019138h)],14h
    if (age > +12)
01011678  cmp         dword ptr [age (01019138h)],0Ch
0101167F  jle         main+3Dh (0101168Dh)
        p = 1800;
01011681  mov         dword ptr [p (0101913Ch)],708h
    else
0101168B  jmp         main+47h (01011697h)
        p = 500;
0101168D  mov         dword ptr [p (0101913Ch)],1F4h
    return 0;
01011697  xor         eax,eax
}
```

アセンブリ言語の中で：

赤丸：　**ブレークポイント**

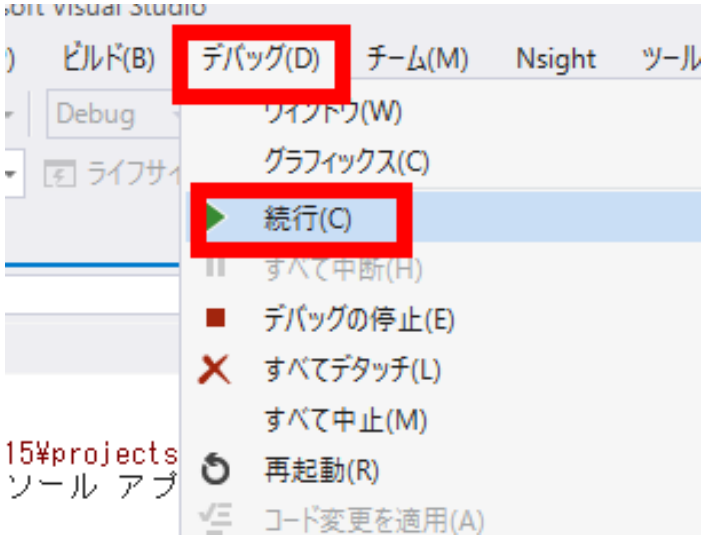
黄色矢印：　**プログラムカウンタ**

```
レジスタ
EAX = CCCCCCCC EBX = 013C5000 ECX = 00000000 EDX = 01019590 ESI = 01011041 EDI = 0111FAB4 EIP = 0101166E ESP = 0111F9E8 EBP = 0111FAB4 EFL = 00000200

0x01019138 = 00000000
```

EIP は
プログラムカウンタ

- 最後に，プログラム実行の再開の操作を行いなさい．これで，デバッガーが終了する．



「デバッグ」
→ 「**続行**」

演習

- 「age = 20;」の行を「age = 10;」に変えて，今の手順を繰り返しなさい.
- ジャンプの様子が変わるので確認しなさい

演習

- 次のプログラムでプログラムカウンタの値の変化の様子を確認しなさい.

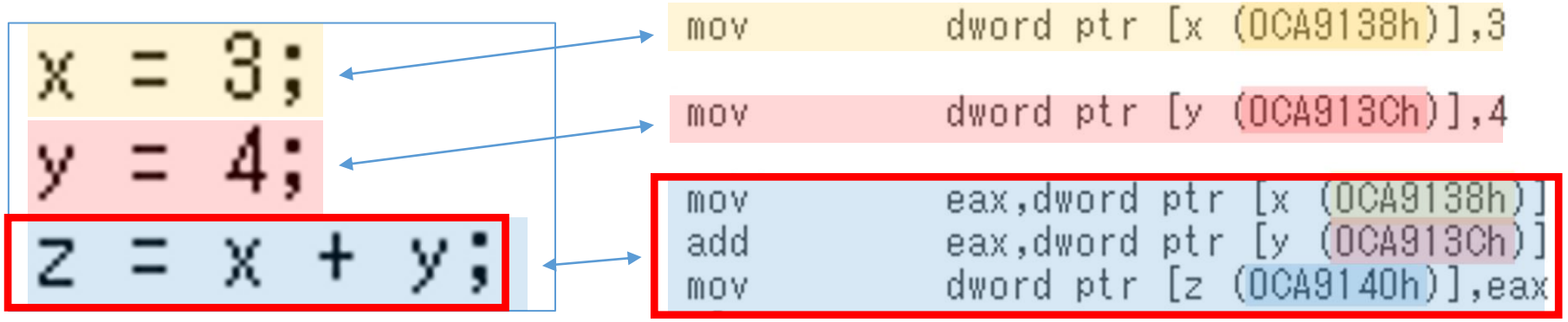
```
int main()
{
    static int i, s;
    s = 0;
    for (i = 1; i <= 3; i++)
        s = s + (i * i);
    return 0;
}
```

5-3 命令実行サイクル

Visual C++ 言語とアセンブリ言語

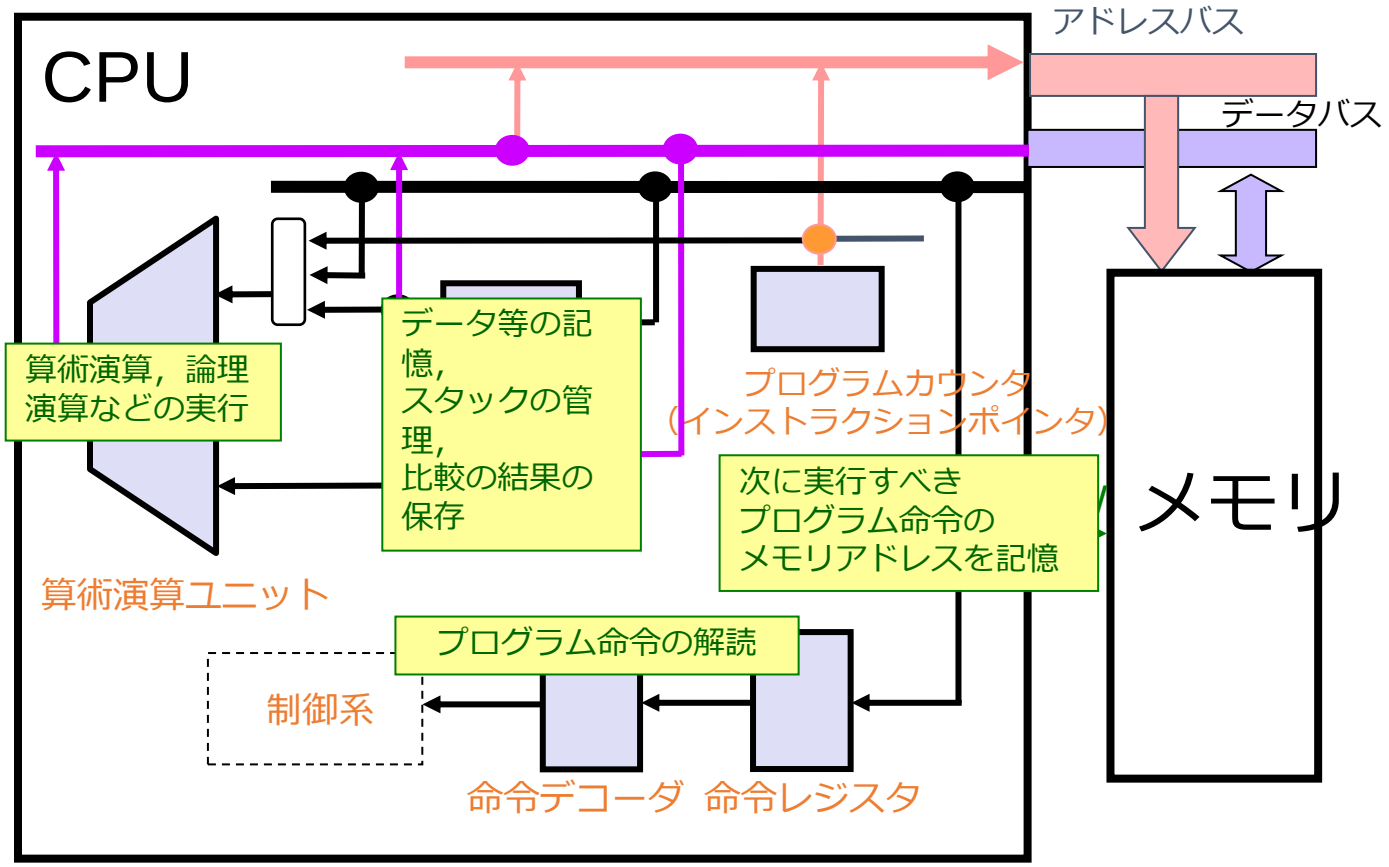
Visual C++ の
プログラム

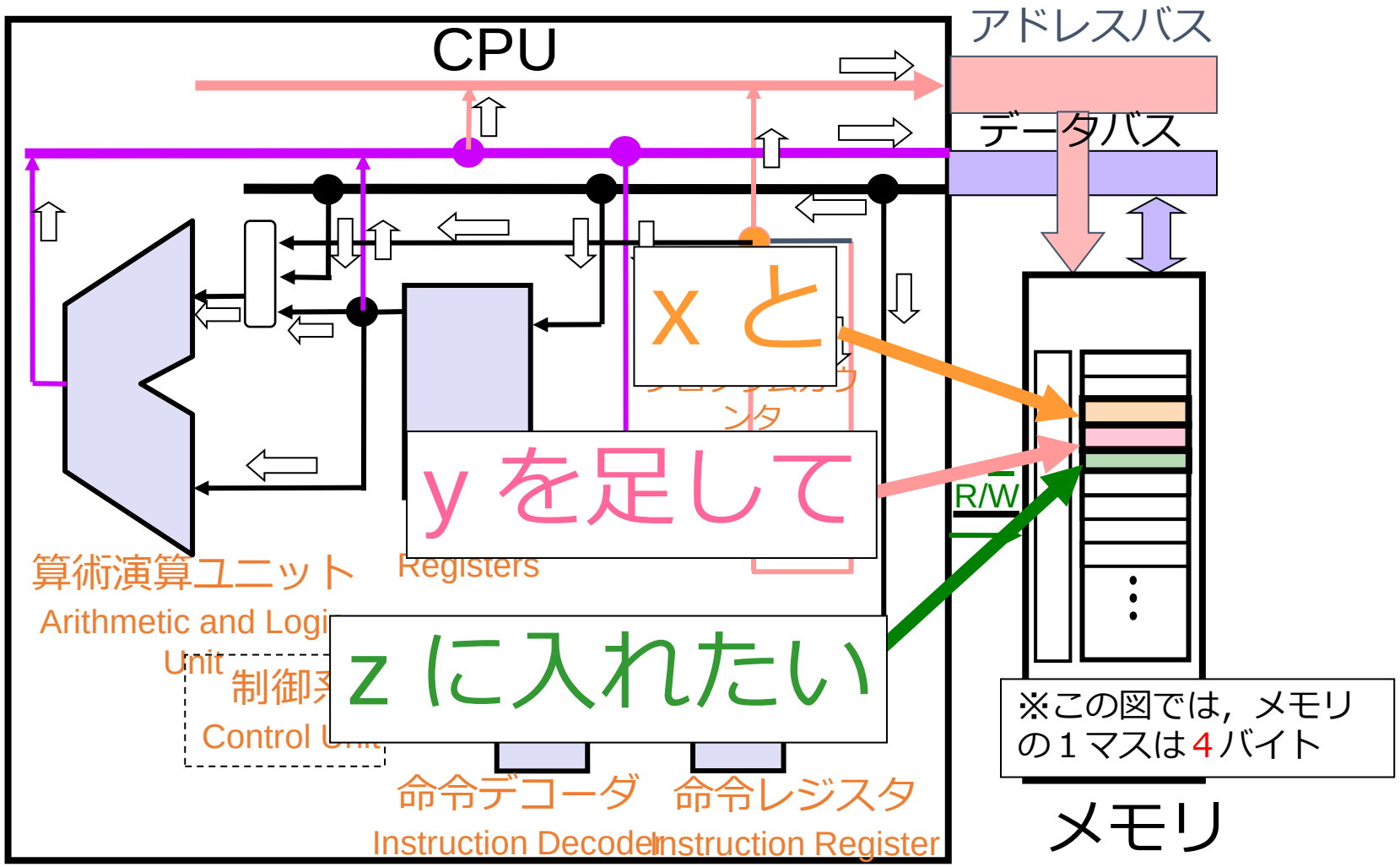
アセンブリ言語

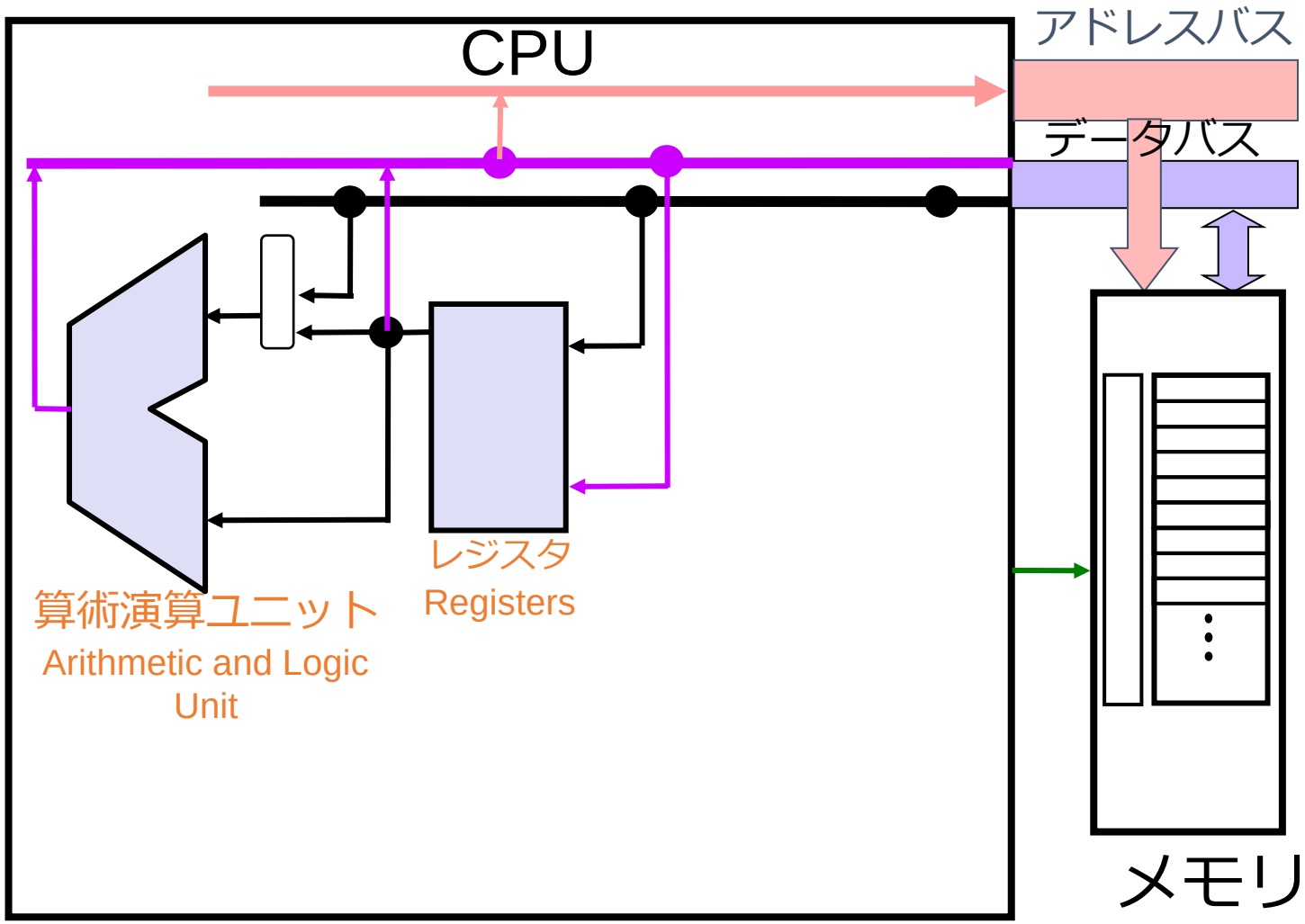


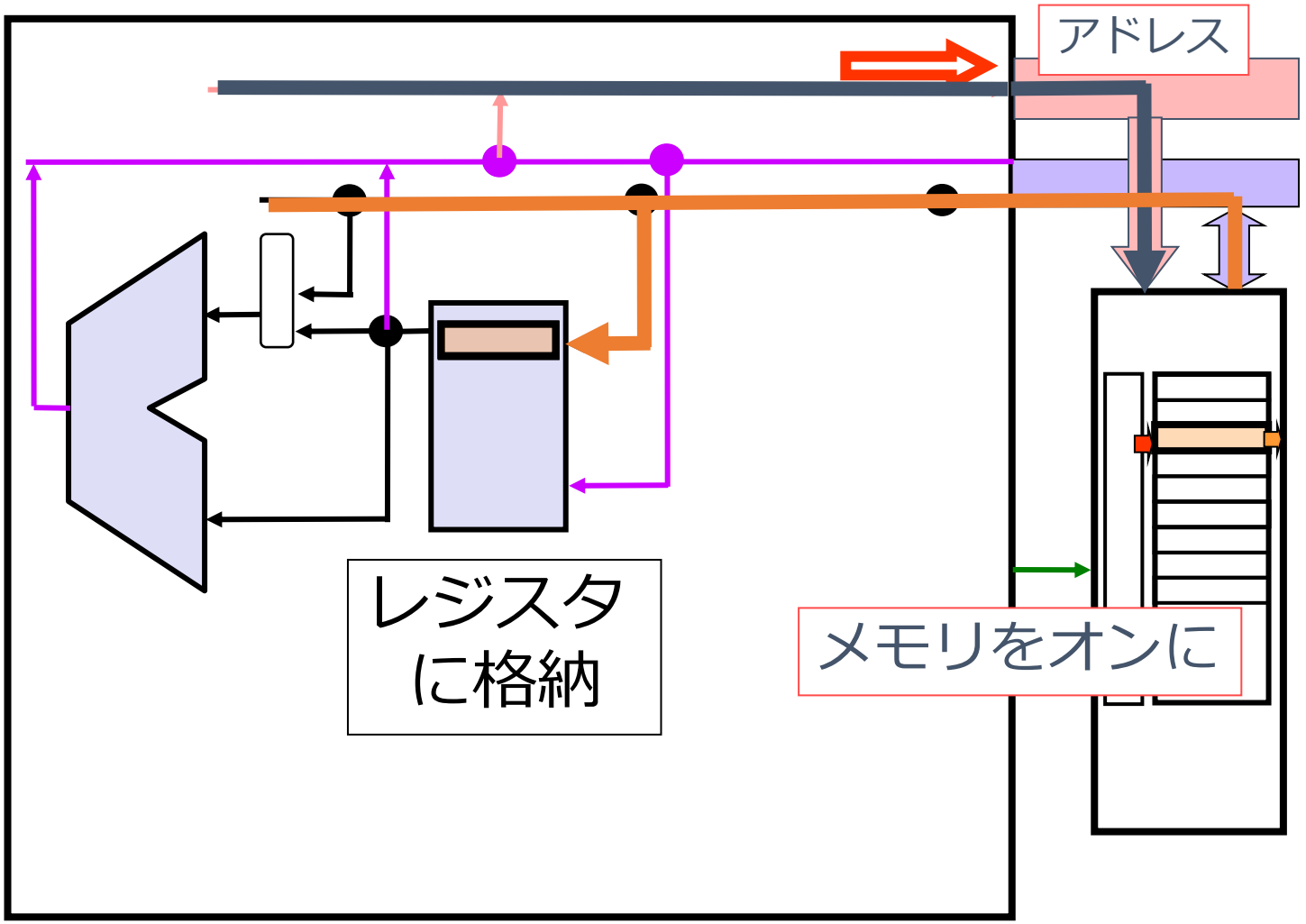
いまから，この３行を図解で説明

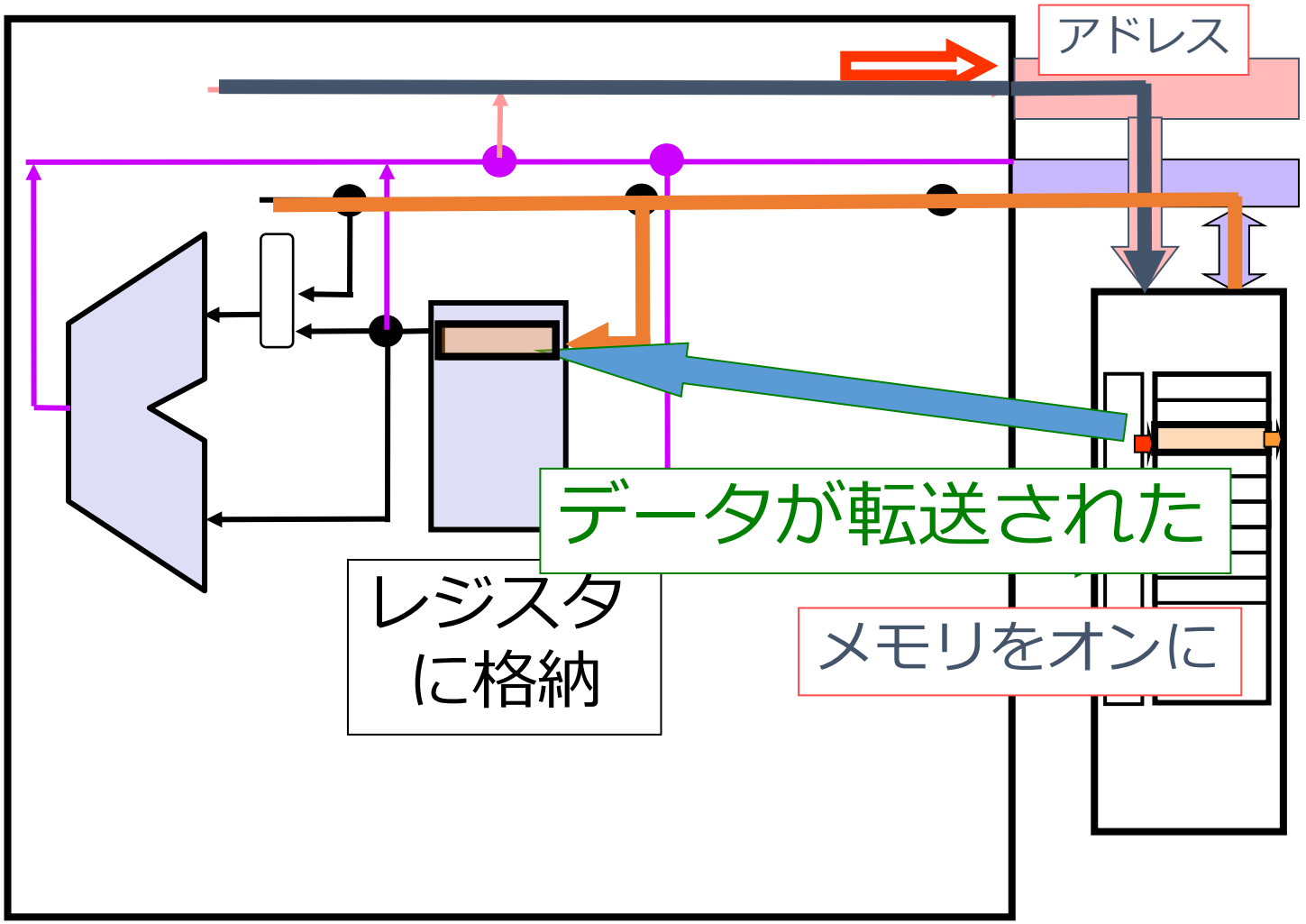
プロセッサの仕組みの概要

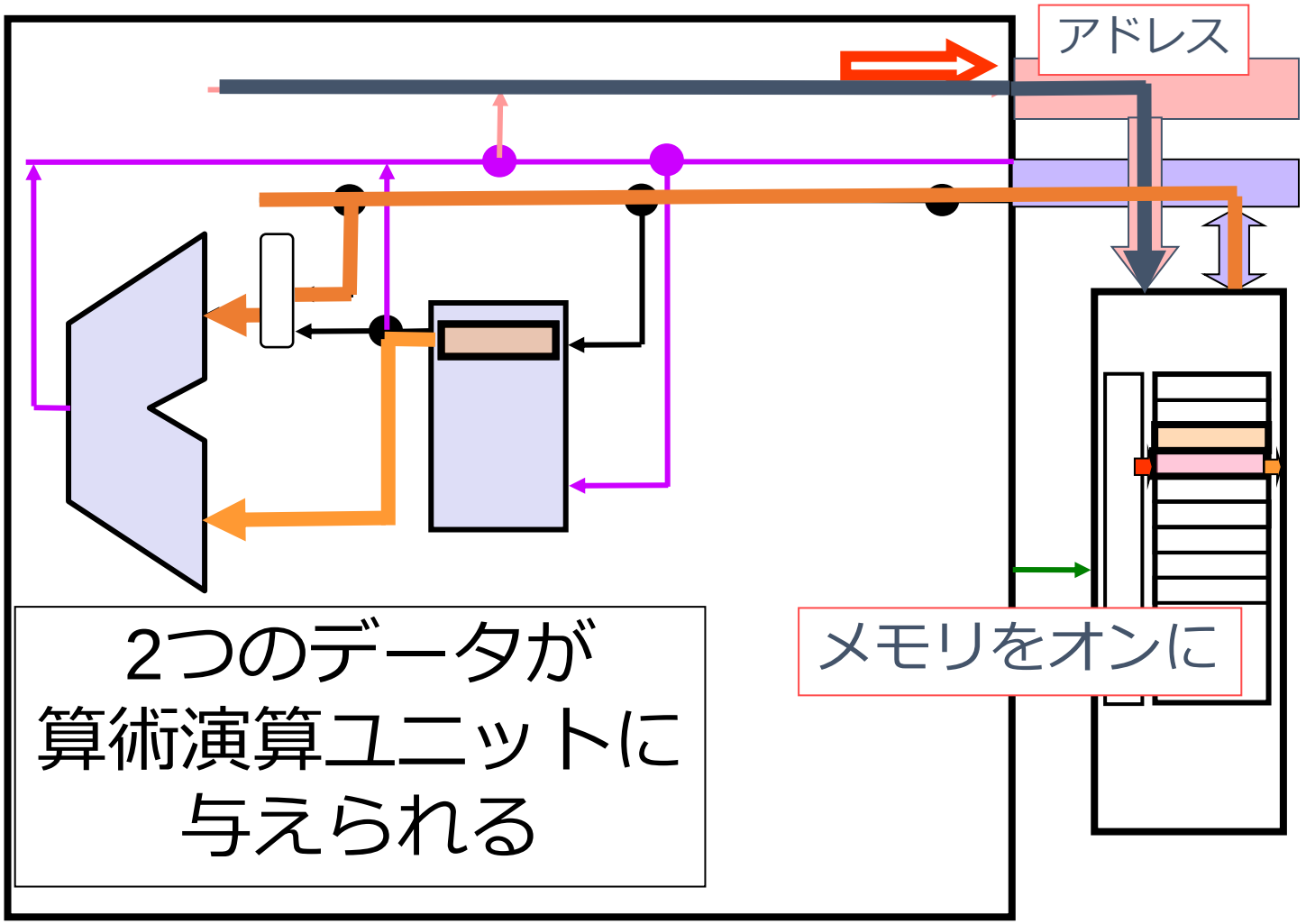


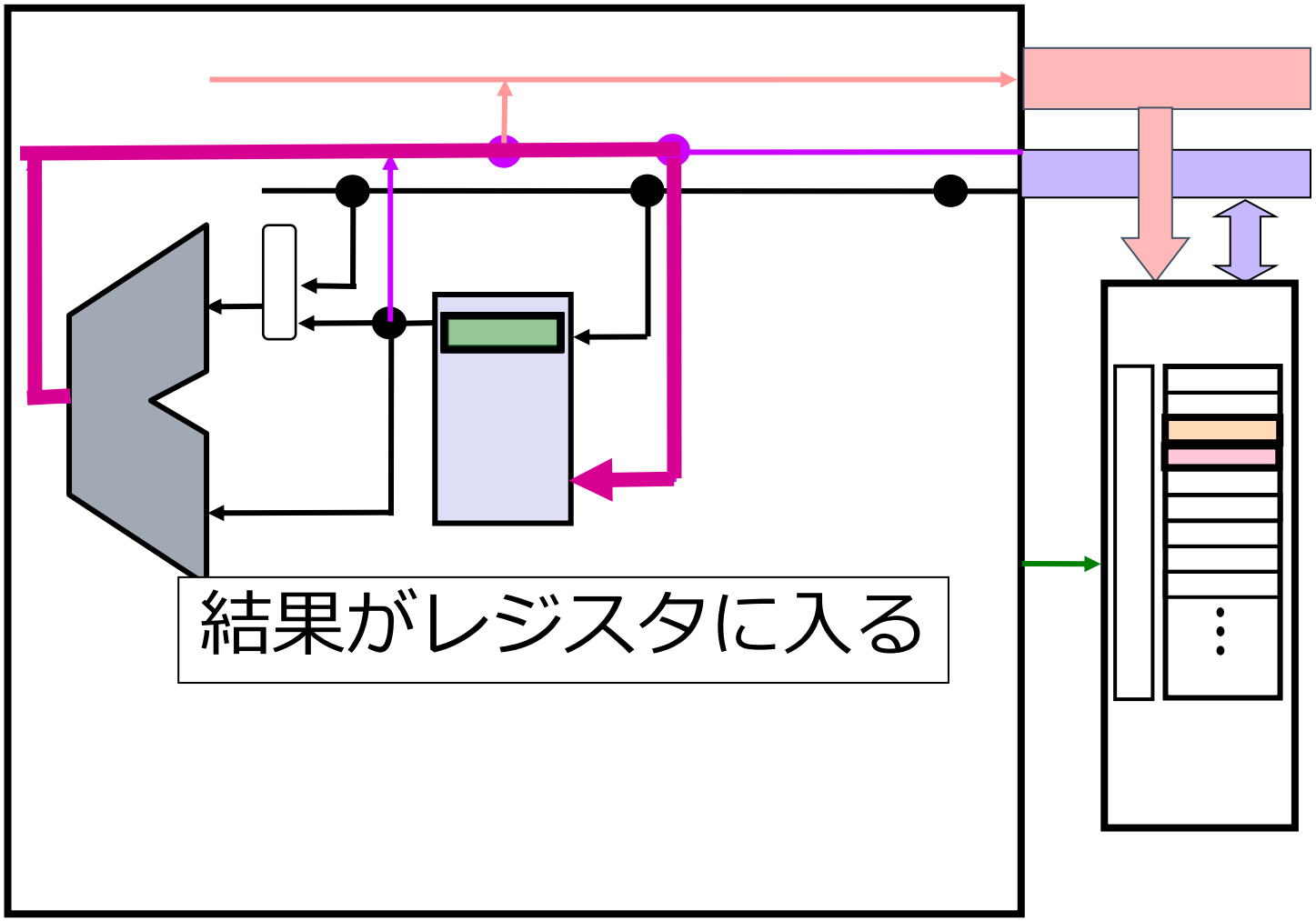

CPU
アドレスバス
データバス
x と
y を足して
z に入れたい
R/W
算術演算ユニット
Registers
命令デコーダ
命令レジスタ
Instruction Register
※この図では，メモリの１マスは４バイト
メモリ

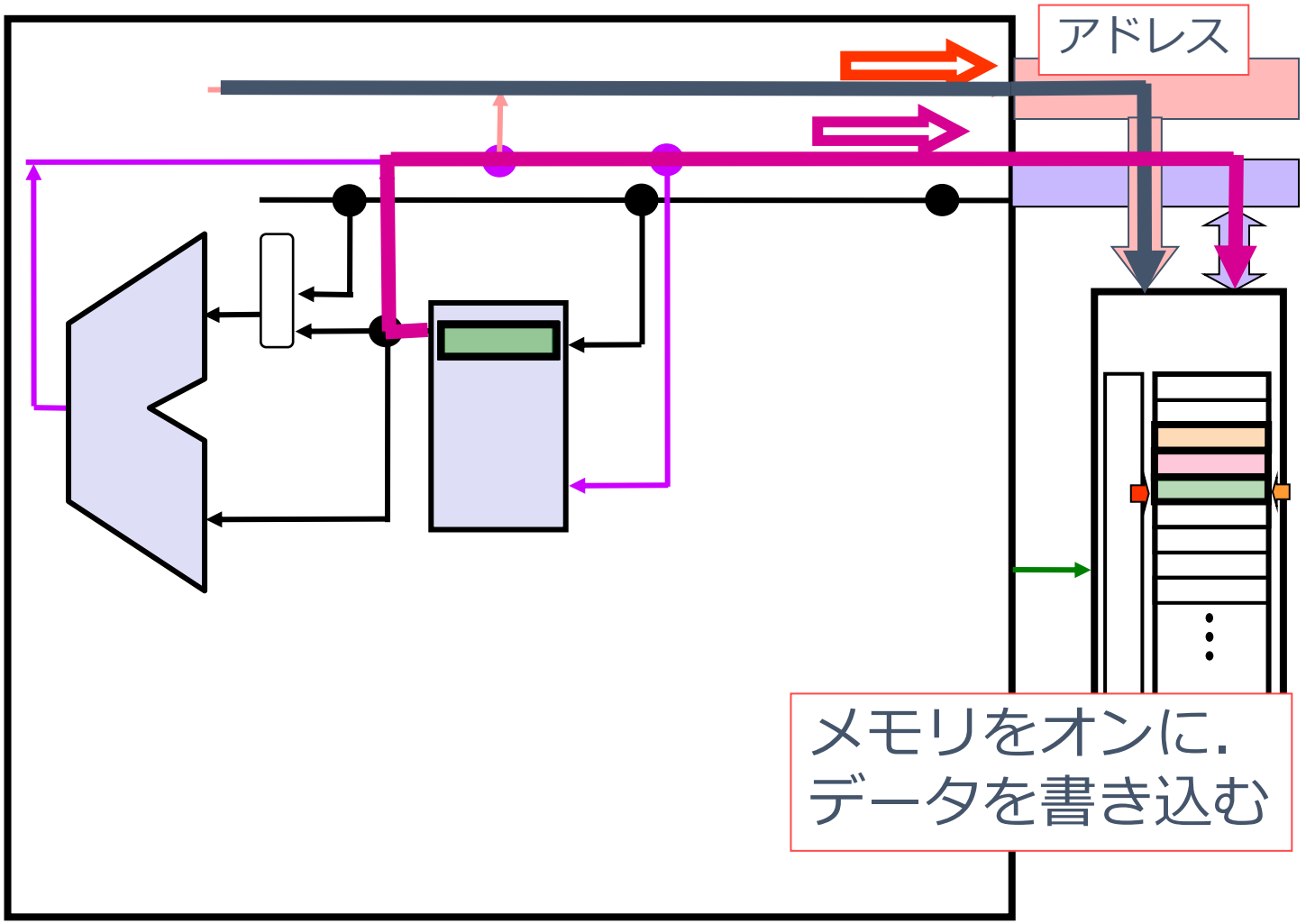

CPU
アドレスバス
データバス
レジスタ
Registers
算術演算ユニット
Arithmetic and Logic Unit
メモリ


アドレス
レジスタ
に格納
メモリをオンに


アドレス
データが転送された
レジスタ
に格納
メモリをオンに


アドレス
2つのデータが
算術演算ユニットに
与えられる
メモリをオンに


結果がレジスタに入る


アドレス
メモリをオンに.
データを書き込む