メンバ関数の実装方法 (基本)

```
class foo {
    void lightfunc();
    void heavyfunc();
};
```

クラス宣言内にはプロトタイプのみ

```
void foo::lightfunc()
{
    return hoge;
}

Void foo::heavyfunc()
{
 ...
 ...
 ...
}
```

宣言部と実装部は別ファイルにする

スコープ解決演算子を使って、メンバ関数がどのクラスに属するものかを明確に。

メンバ関数の実装をヘッダのクラス宣言内に書くと…

```
class foo {
    void lightfunc() { return hoge; }
    void heavyfunc() {
                ....
                ....
                ....
    }
};
```

セミコロンはいらない

- クラス宣言内に記述された関数に対して、コンパイラはインライン展開を試みる
- インライン展開できないような大きな関数は、通常の関数が生成される

見比べると

クラス宣言内にはプロトタイプのみ

```
class foo {
    void lightfunc();
    void heavyfunc();
};
```

```
void foo::lightfunc()
{
    return hoge;
}

void foo::heavyfunc()
{
 ...
 ...
 ...
}
```

宣言部と
実装部は
別ファイル

スコープ解決演算子を使って、メンバ関数がどのクラスに属するものかを明確に。

```
class foo {
    void lightfunc(){ return hoge;}
    void heavyfunc(){
                    ....
                    ....
                    ....
    }
};
```

インライン展開の例

```
int add(int a, int b)
    {  return a + b;    }

inline int il_add(int a, int b)
    {   return a + b;    }

void foo()
{
     int result;
     //通常の関数呼び出し
     result = add(1, 2);
     //インライン展開
     result = il_add(3, 4);
}
```

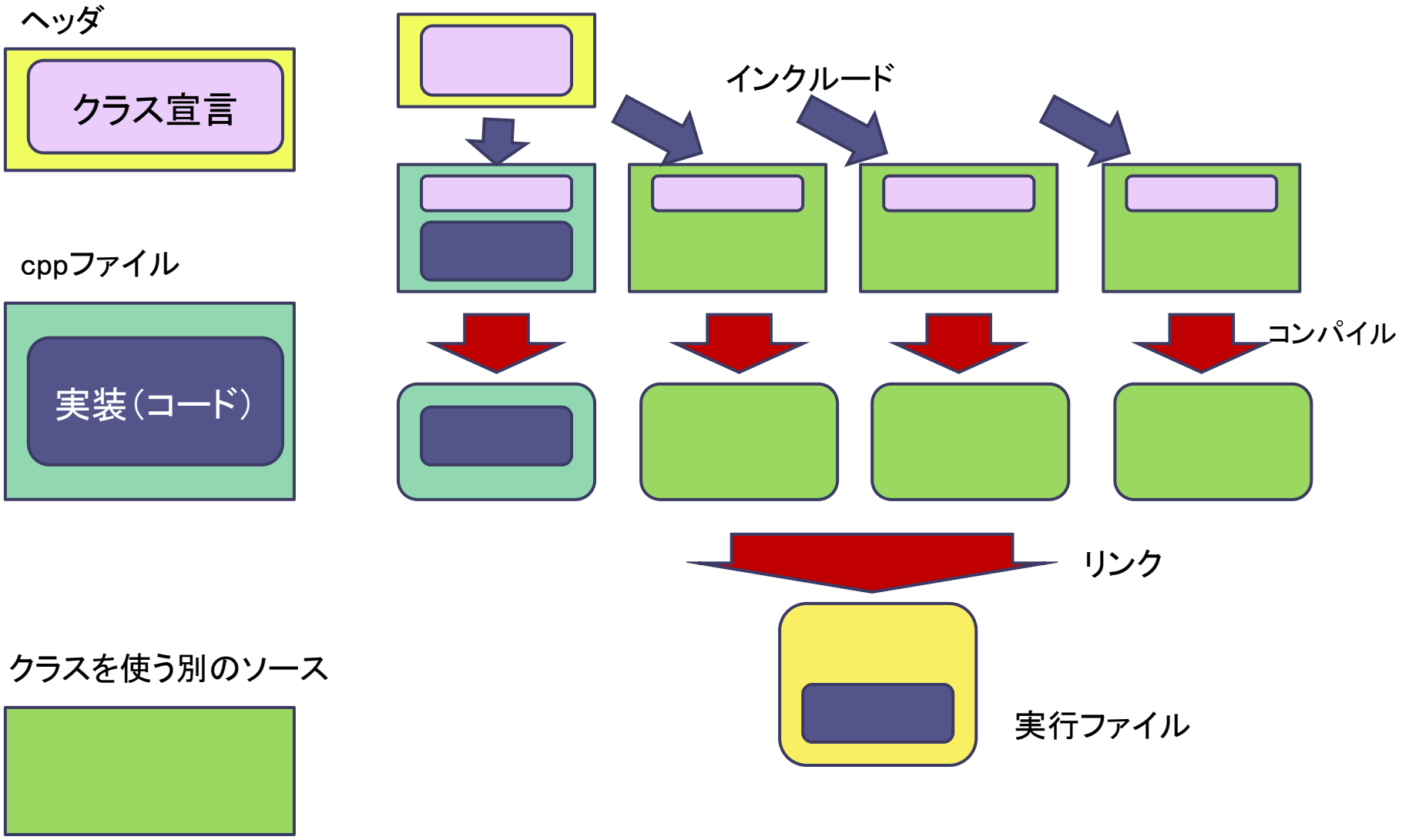
```
push    2
push    1
call    ?add@@YAHHH@Z
add     esp,8
mov     dword ptr [ebp-4],eax
```

```
mov     eax,3
add     eax,4
mov     dword ptr [ebp-4],eax
```

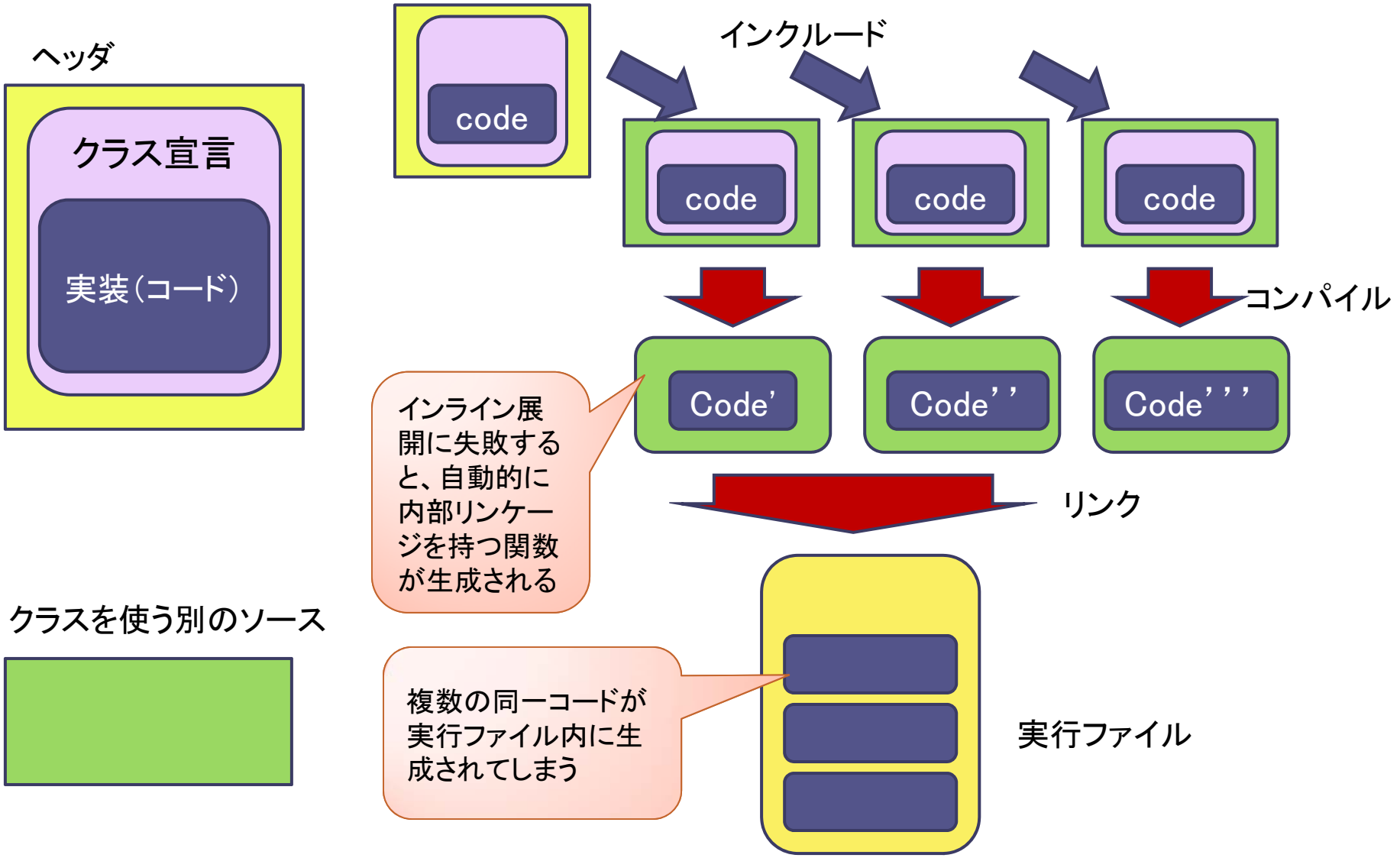
- この例では、インライン展開したほうが命令数も少なく、実行サイクル数も低減している
- インライン展開することにより、コード最適化の可能性が高まる
- 命令数の多い関数をインライン展開すると、全体のコードサイズが増加し、実行速度の低下をまねく可能性がある

では、インライン展開不能な関数の場合、この2つは見た目以外になにが違うのか？

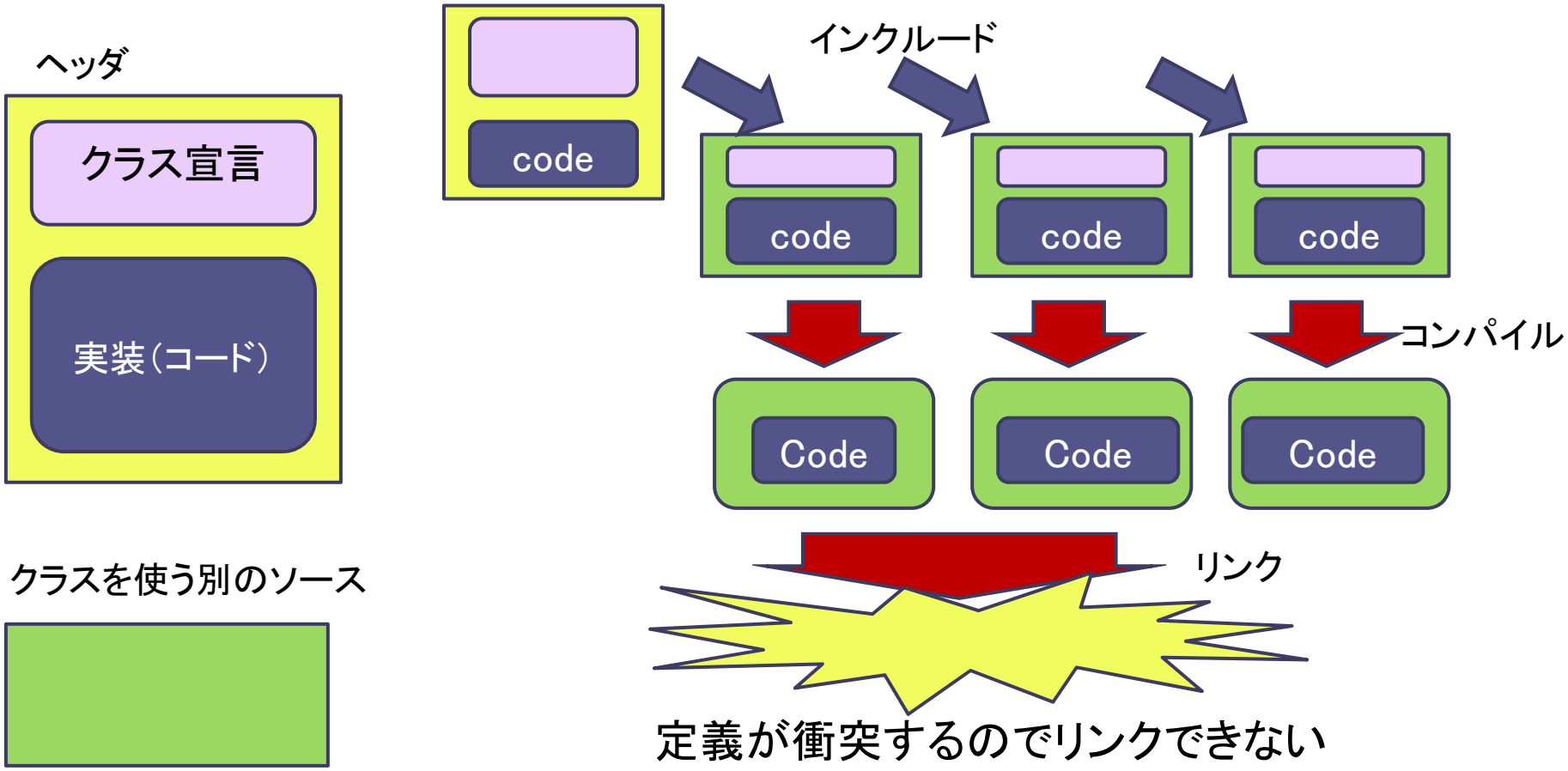
実装部を正しく分けた場合



実装部をクラス内に書くと



実装部をヘッダ内のクラス外に書くと…（もっとひどい失敗）



- 結論：インライン可能な関数はクラス内に、そうでない関数は別ファイルにという原則を守ろう！