プログラムが実行されるまで

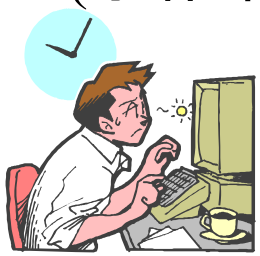
2002年4月14日

海谷 治彦

Cというか一般的なUNIX, Linuxのバイナリ

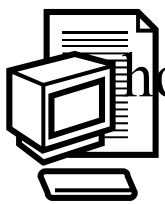
Cプログラムの開発の流れ

プログラミング
(手作業)



コンパイル

実行



ソースコード
(原始プログラム
hoge.c 等)

ロードモジュール
(実行可能プログラム
a.out 等)

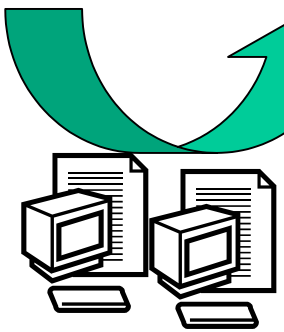
分割コンパイル

プログラミング
(手作業)

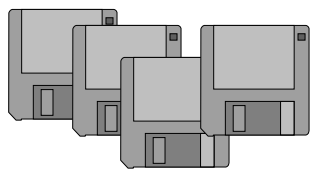
コンパイル

リンケージ
・エディット

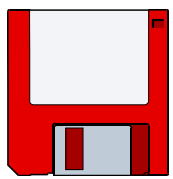
実行



ソースコード
(原始プログラム
hoge.c 等)



オブジェクトコード
(目的ファイル,
マシン語,
hoge.o など)

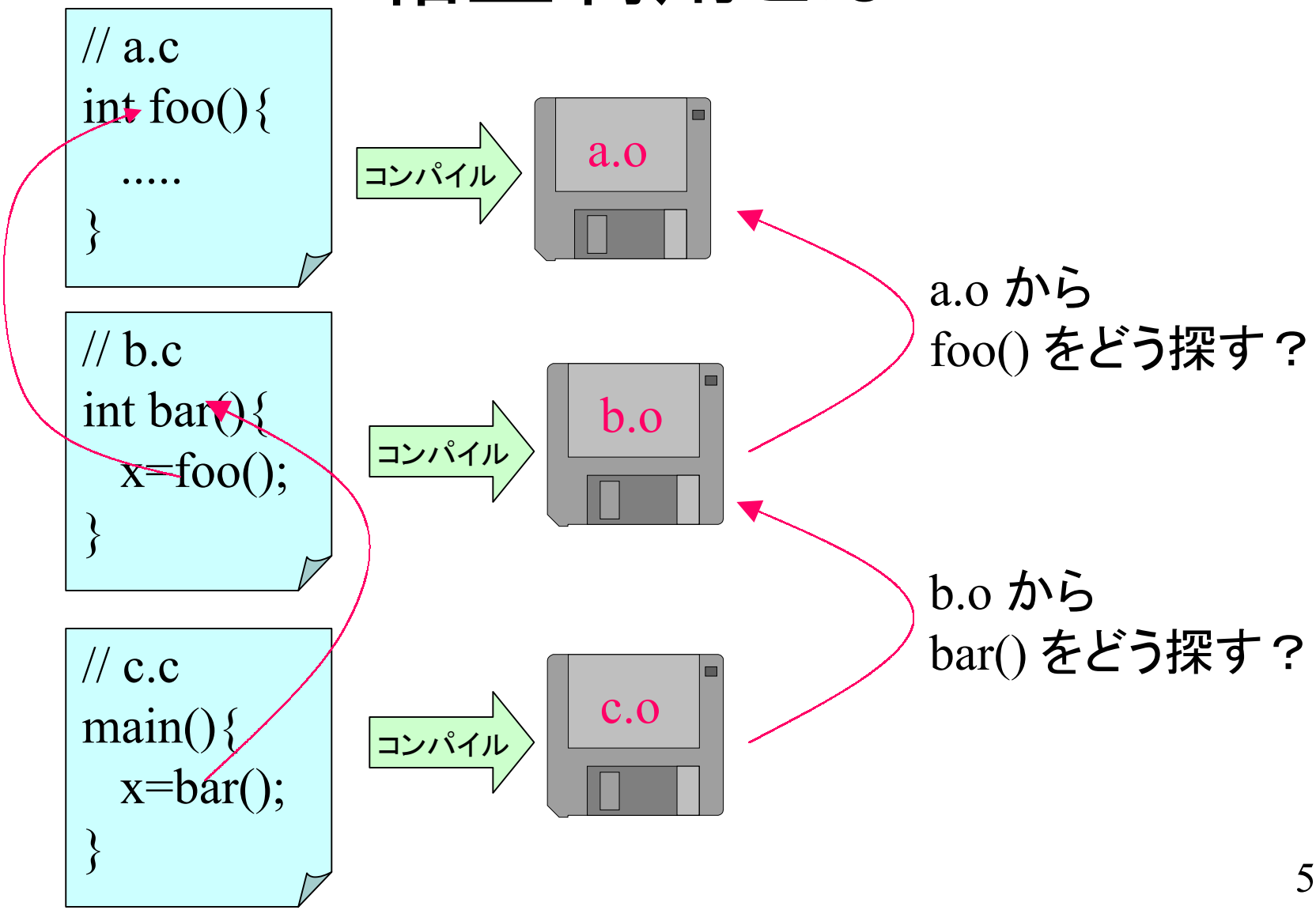


ロードモジュール
(実行可能プログラム
a.out 等)

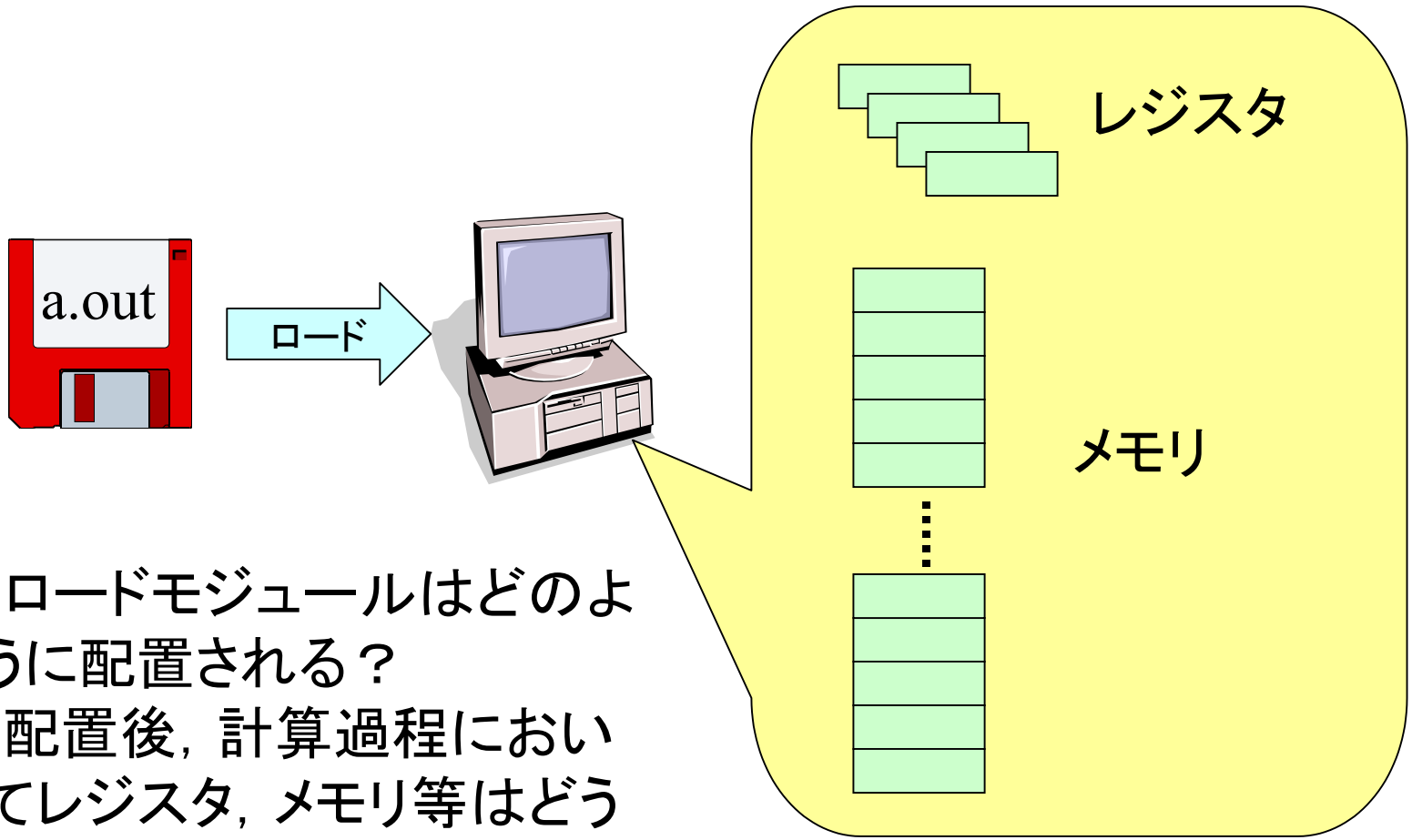
本授業で注目する点

- オブジェクトコード(マシン語)の中身の構造
- ばらばらのファイルに分かれたマシン語が相互利用できる仕組み(リンケージ)
- マシン語がマシンに読み込まれる仕組み(ロード)
- 読み込まれてからの振る舞い

相互利用とは？


// a.c
int foo(){
.....
}
// b.c
int bar(){
x=foo();
}
// c.c
main(){
x=bar();
}
コンパイル
コンパイル
コンパイル
a.o
b.o
c.o
a.o から
foo() をどう探す？
b.o から
bar() をどう探す？

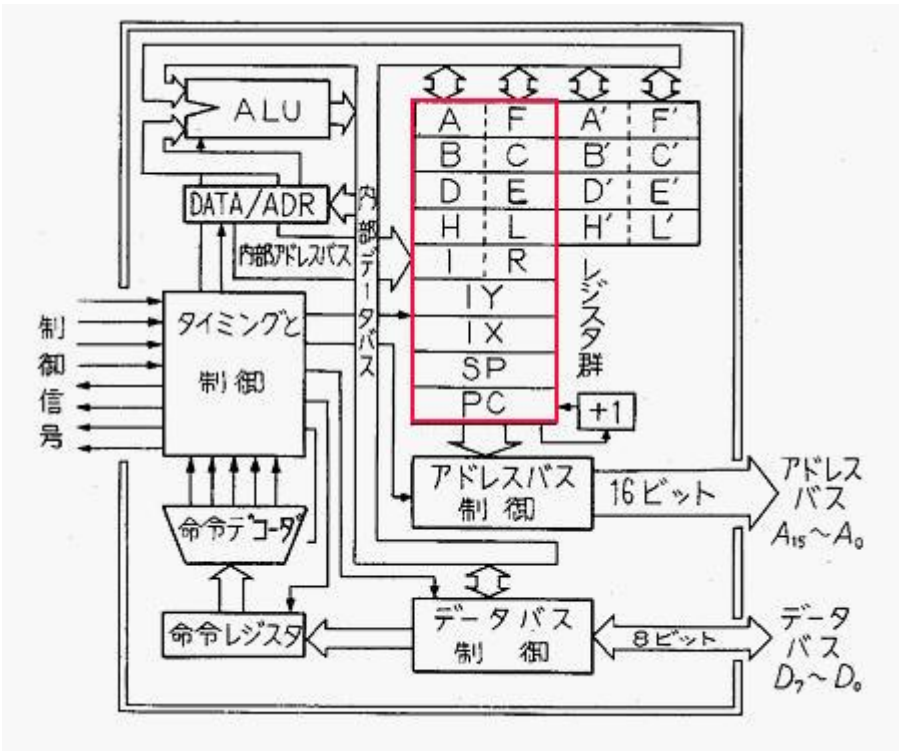
マシン語に読み込まれる

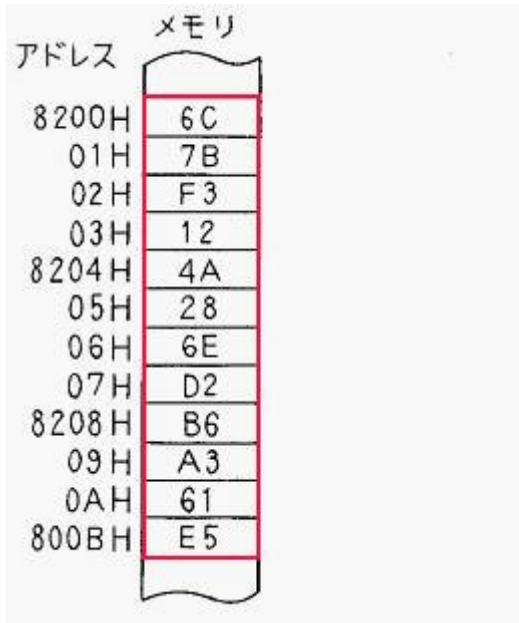


- ロードモジュールはどのように配置される？
- 配置後，計算過程においてレジスタ，メモリ等はどう変化する？

ノイマンマシンの基本構造

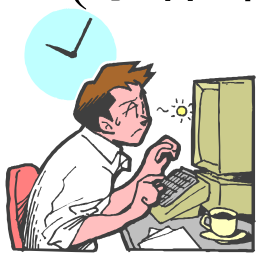
レジスタと呼ばれる数個の変数と，メモリと呼ばれるたった1つの配列しか使えないプログラミング環境．





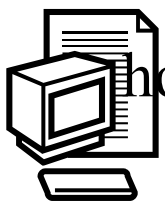
Javaプログラム開発の流れ

プログラミング
(手作業)



コンパイル
javac hoge.java

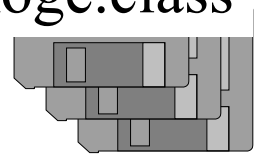
実行
java hoge



hoge.java

ソースコード
(原始プログラム
hoge.java 等)

hoge.class



クラスファイル
(**hoge.class** 等)

JavaとC(等)との違い

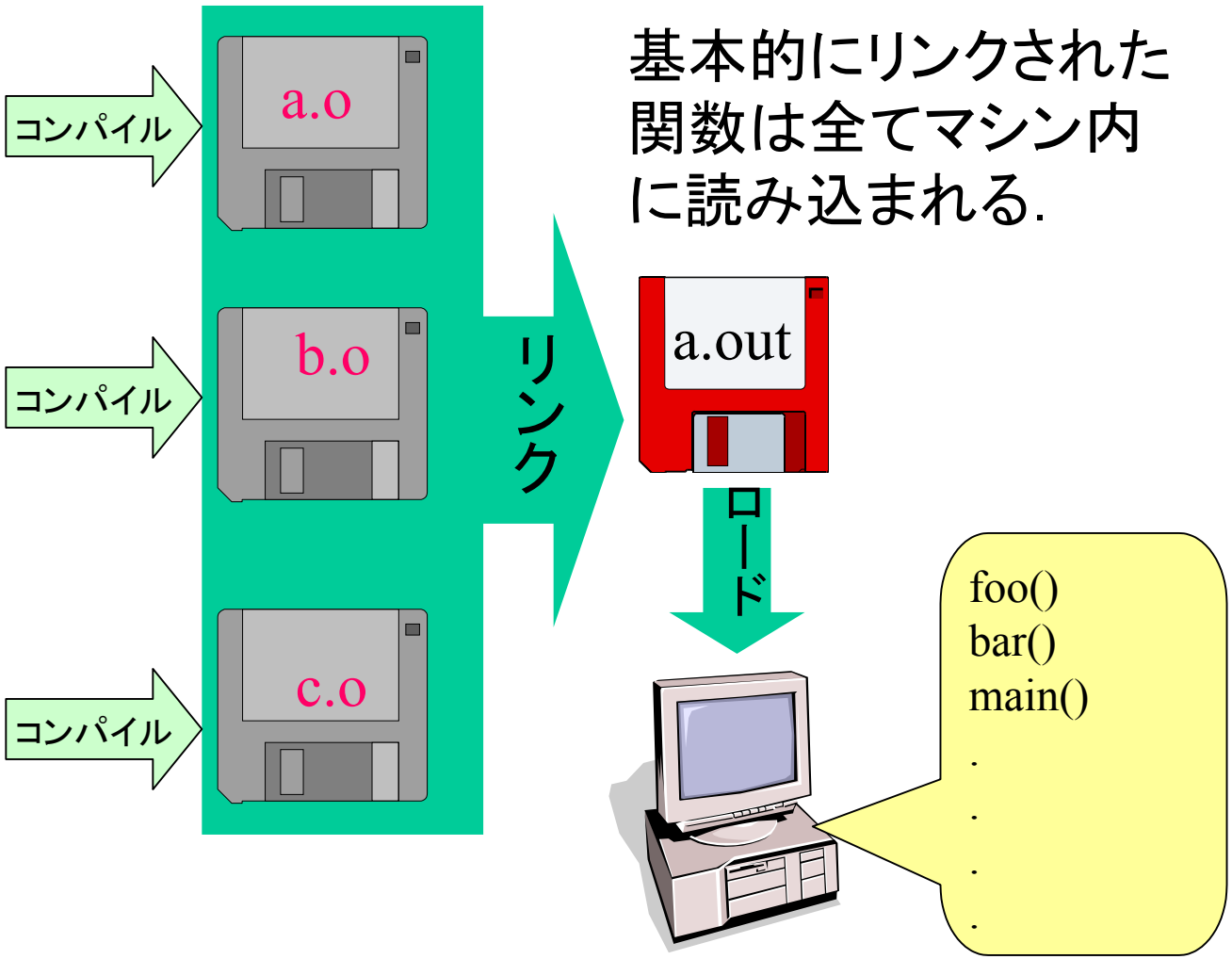
- リンケージはどこでやっているのか？
  - ロードした後，マシン内でやっている.
- ロードはどうやってやっているのか？
  - 必要なクラス(マシン語)のみロードする.

Cのロード

```
// a.c
int foo(){
   .....
}
```

```
// b.c
int bar(){
   ........
}
```

```
// c.c
main(){
 if(x>10)
   foo();
 else
   bar();
}
```



基本的にリンクされた関数は全てマシン内に読み込まれる.

Javaのロード

```
// foo.java
class foo{
   .....
}
```

コンパイル → foo.class

```
// bar.java
class bar{
   ........
}
```

コンパイル → bar.class

```
// go.java
class go{
... main(...){
 if(x>10)
   new foo();
 else
   new bar();
} ....
```

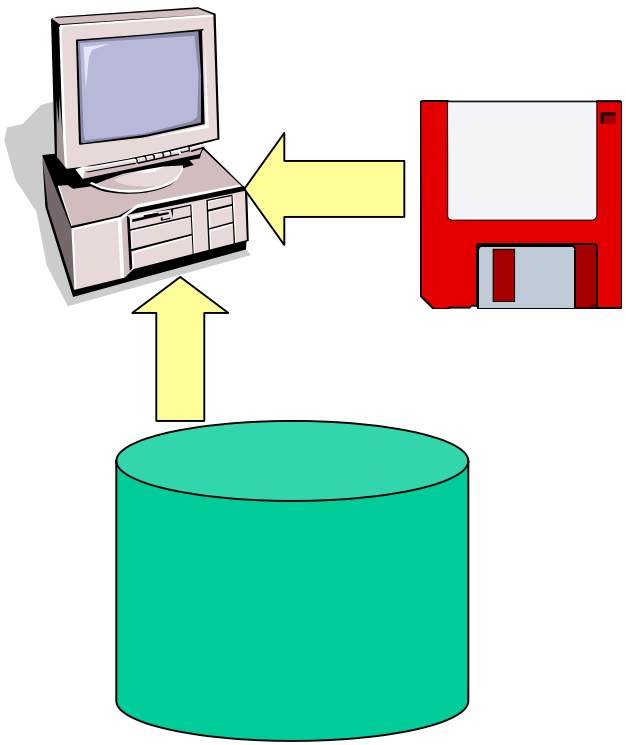
コンパイル → go.class

最初のロード
java go

例えば，
go.class
foo.class
.
.
.

- 事前にリンクしない．
- 必要に応じて必要なクラスを追加ロード．
- (分岐等の場合，一方しかロードされない．)

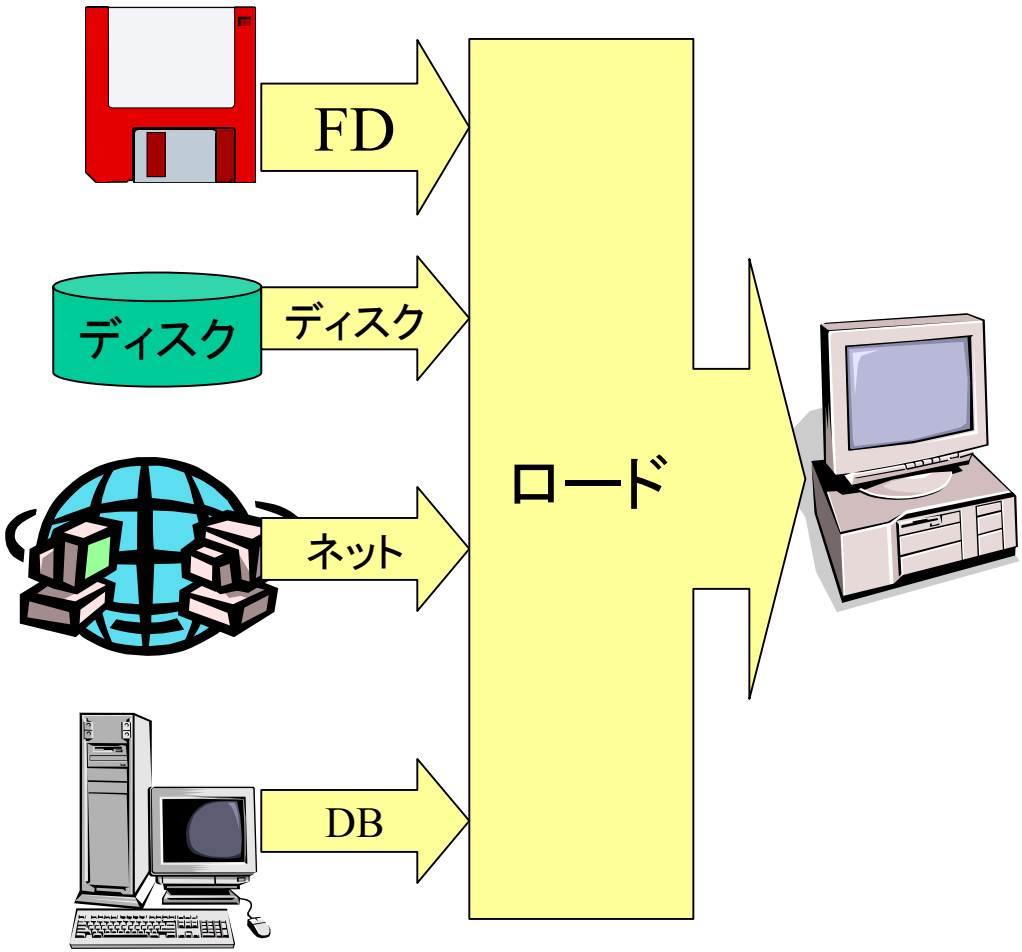
Cの場合のロード元



•基本的には自身の記憶デバイス(ディスク，FD，CDROM等)からしかロードできない．

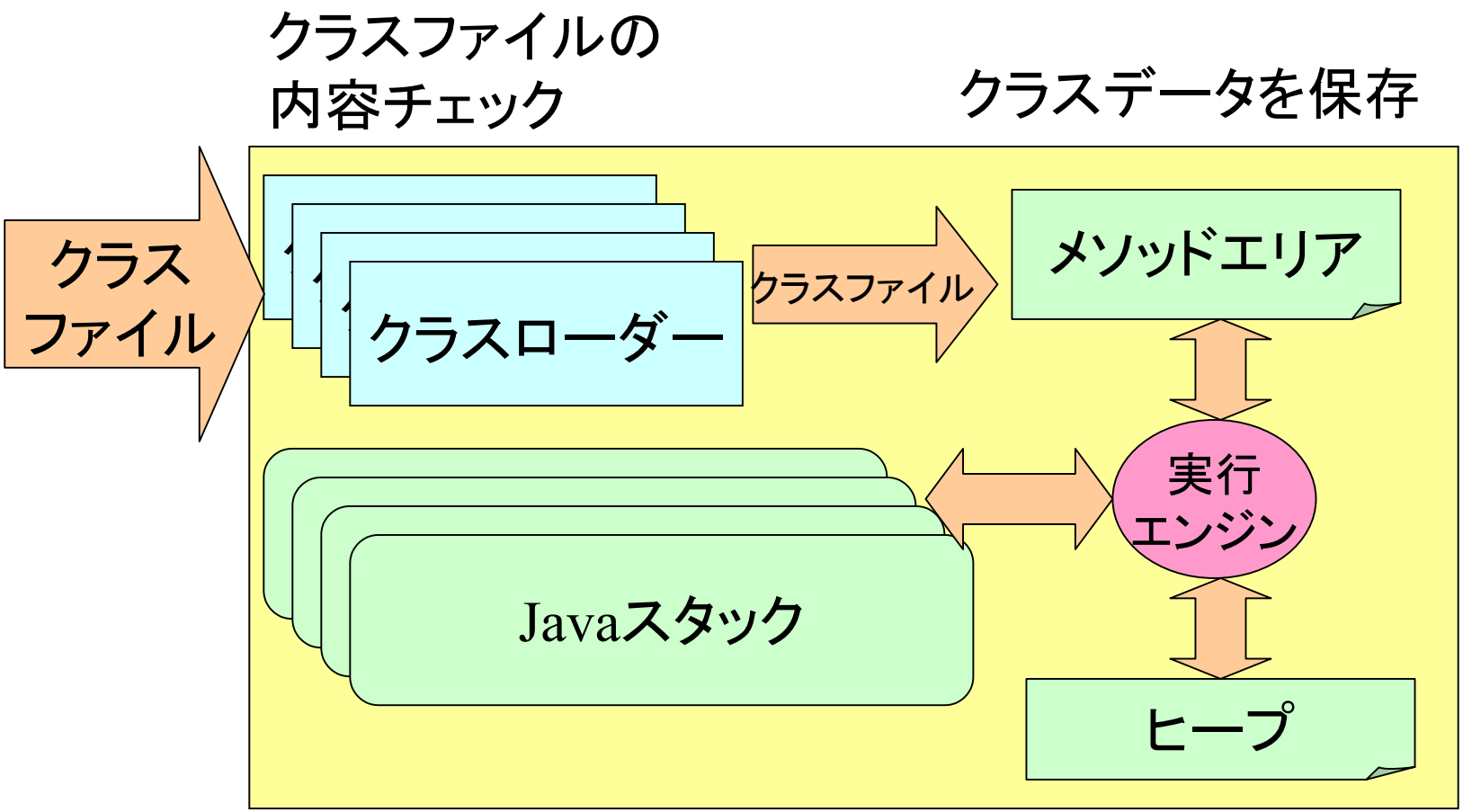
•1つのプログラムにおいて，多様なロード先を持つことは難しい．(というか原則，ロードは1回)

Javaの場合のロード



- 多様なソースからクラスをロードできる.
- 1つのプログラムが複数のソースを持てる.

JVMの基本構造(大雑把)



各実行スレッドのローカルデータ(実行経過)を保存

インスタンスデータを保存