

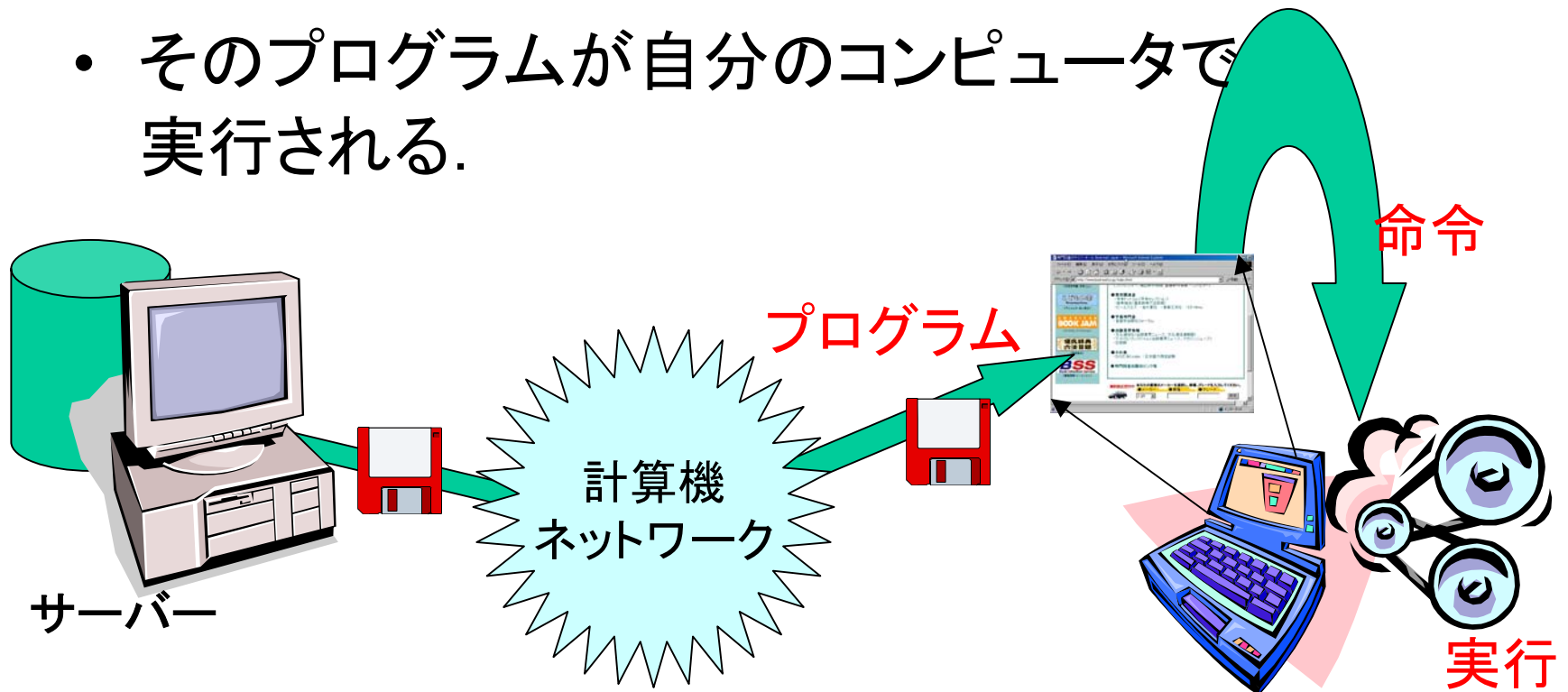
モバイルコード技術と セキュリティポリシー (1)

2005年12月14日

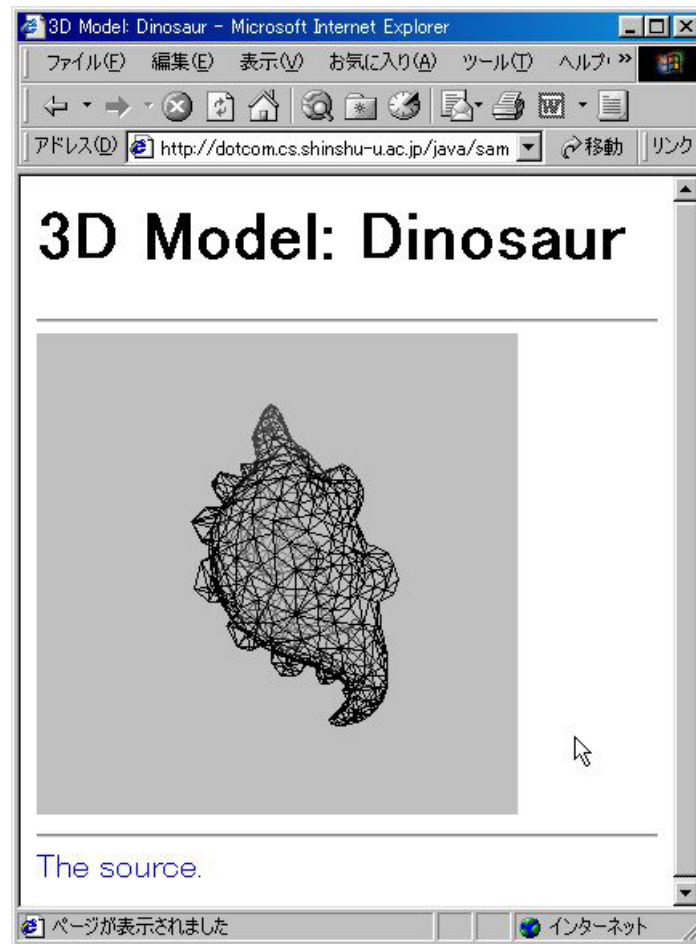
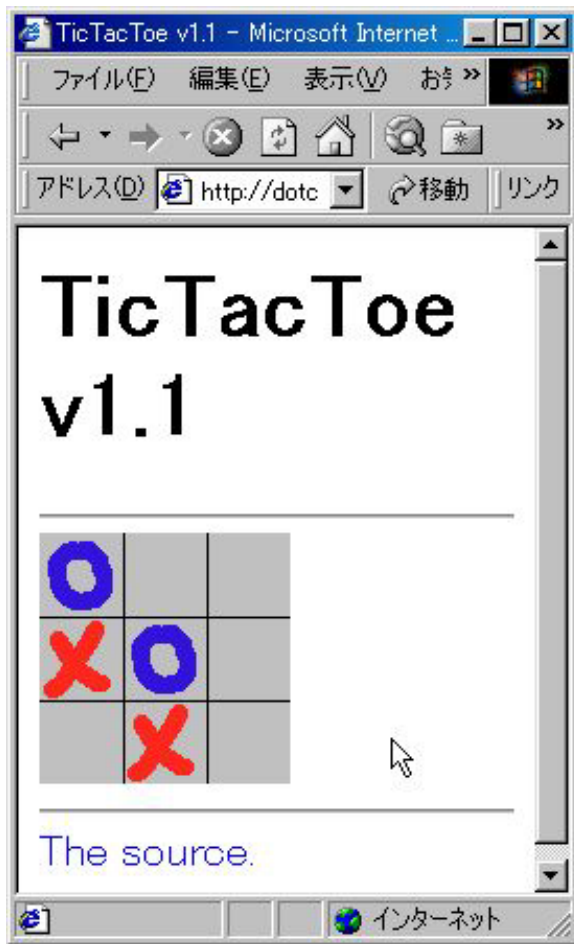
海谷 治彦

モバイル・コード技術

- ウェブページ(データ)と同様に,
プログラムがダウンロードされる.
- そのプログラムが自分のコンピュータで
実行される.

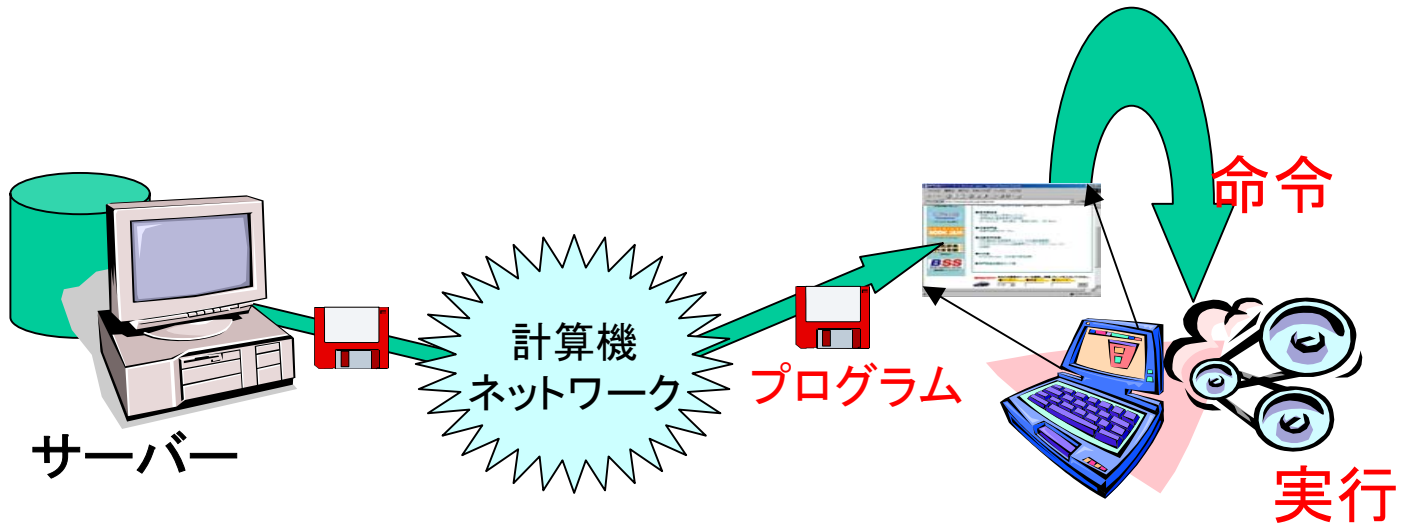


モバイルコードの例: Applet



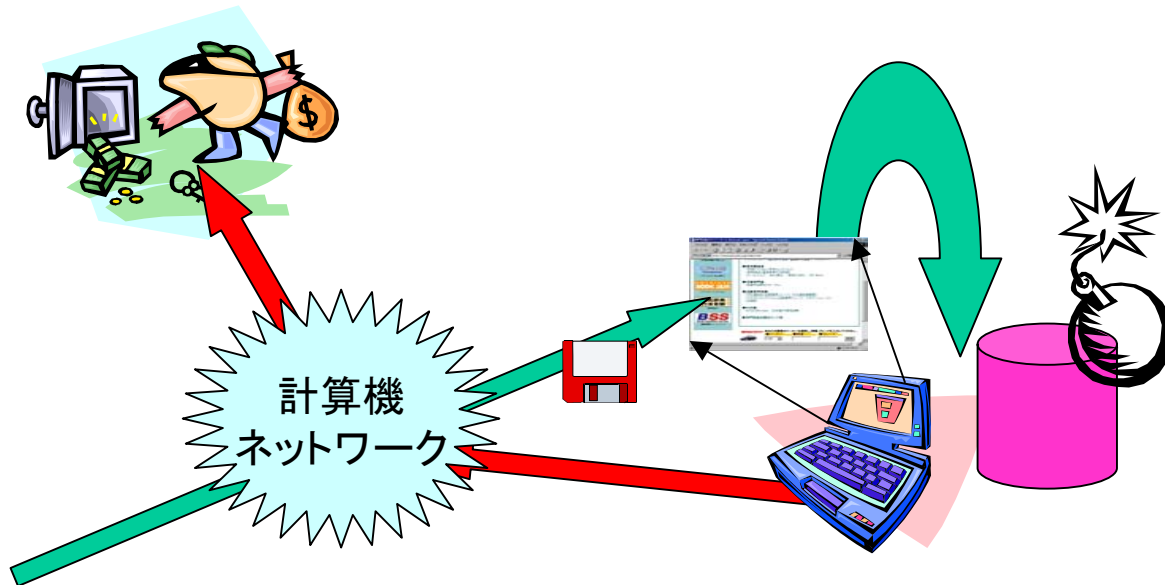
モバイルコードの利点

- ネットワーク上のデータの交通量の削減
- レスポンスの向上
- 一々、ソフトをインストールしないでも、他人の作ったソフトを利用できる.

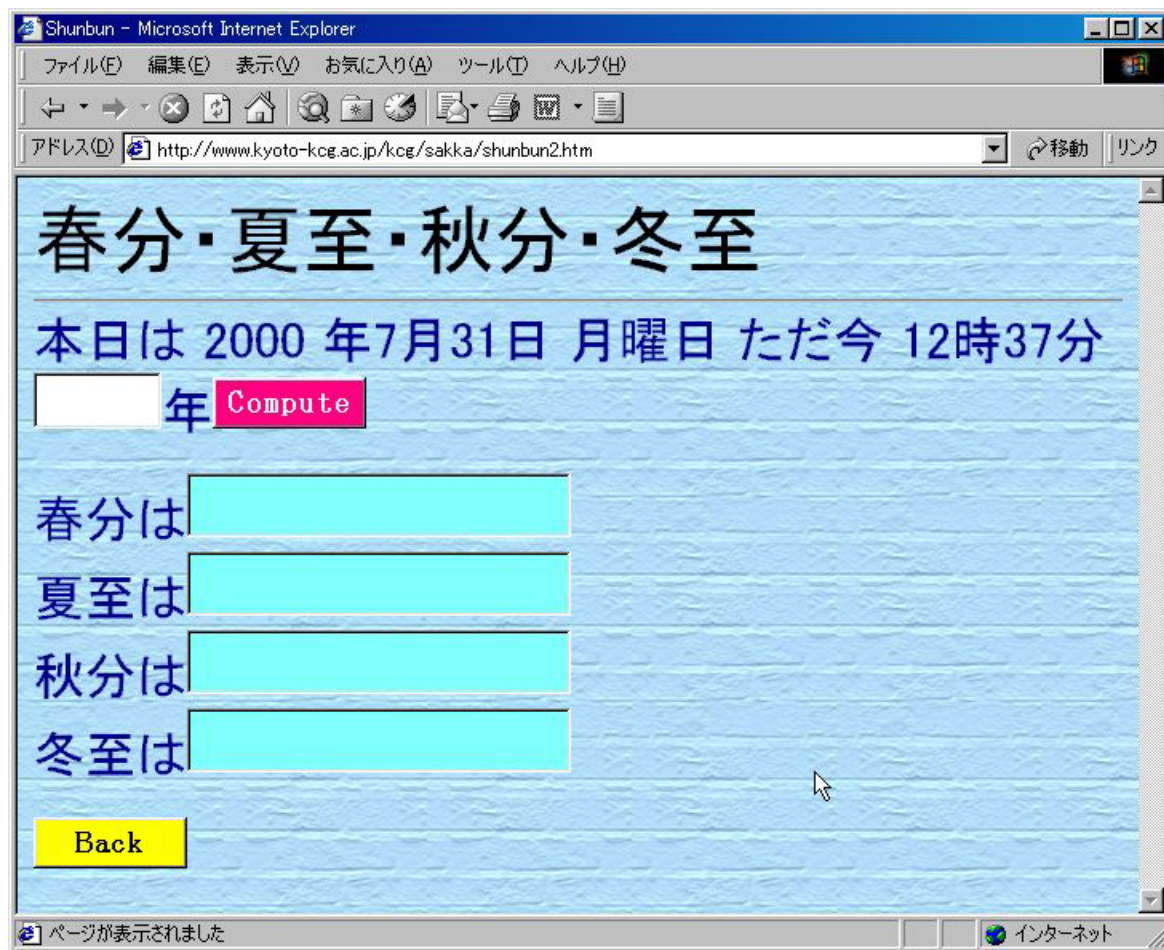


モバイルコードの欠点

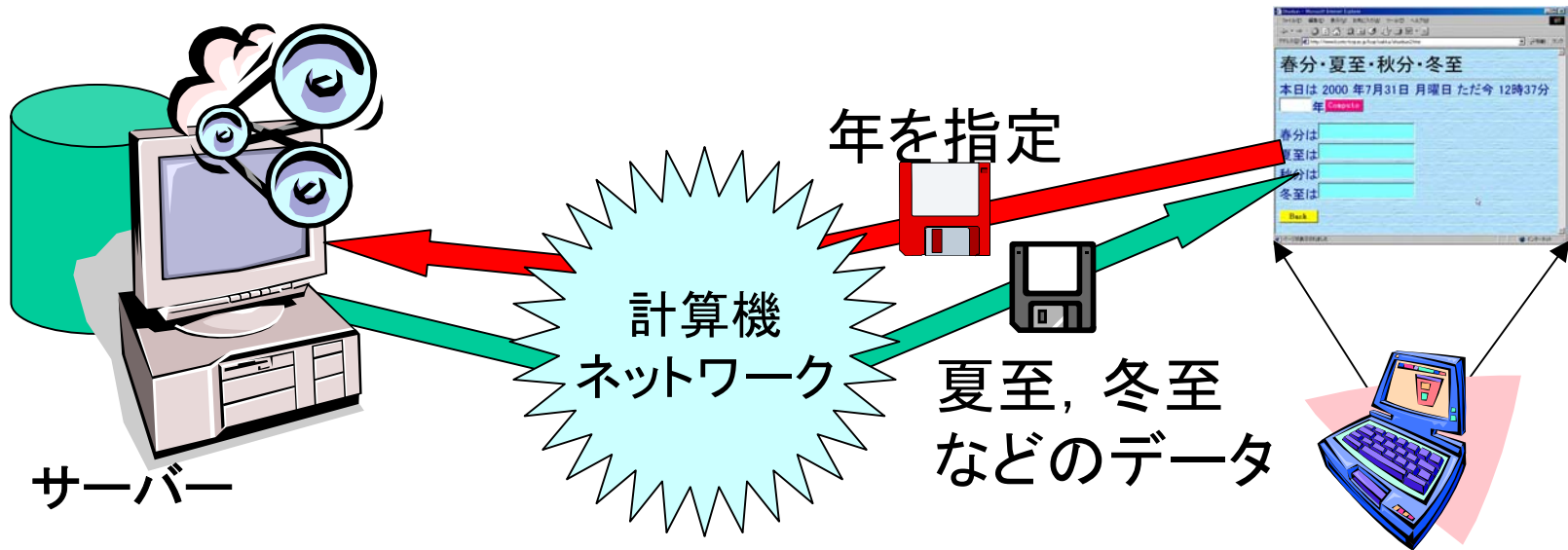
- 他人のプログラムに悪意があった場合,
 - 自分のデータが破壊されるかも？
 - 自分のデータが盗まれるかも？
 - 自分のパソコンが勝手に悪戯するかも？



CGI vs モバイルコード: 例題



CGIやウェブサービスの場合



- 計算自体はサーバーで行われる
- 手元で繰り返し計算する場合, 不利.

モバイルコードの場合



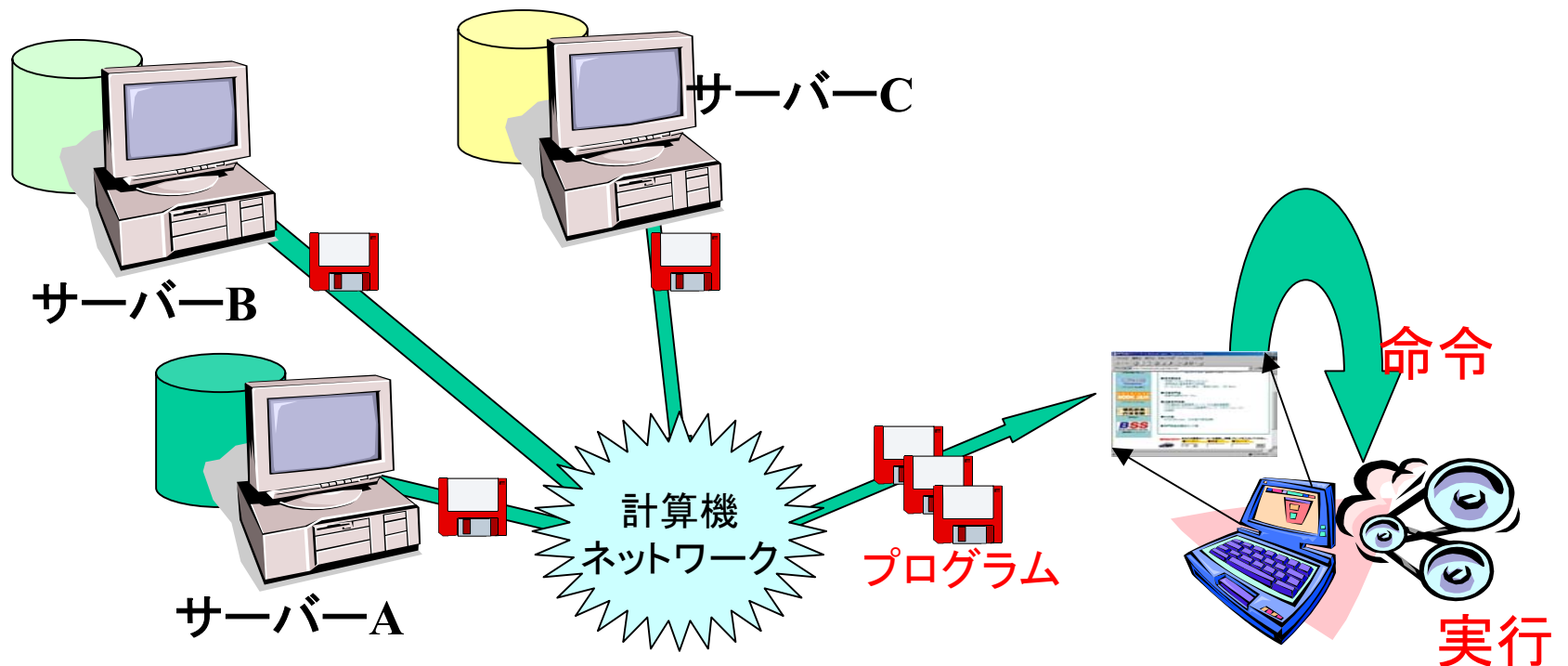
- 計算は自分の手元のパソコンで行われる
- 大規模なデータを参照する場合には不利かも？

CGI vs モバイルコード: 結論

- CGI: 計算はサーバーで (**一極集中**)
 - 大規模データを参照する場合便利
 - サーバーが混みだすとヤバイ
 - 一般にネットワーク上のデータやりとりが多い
- モバイルコード: 計算は自分のPCで (**分散**)
 - 計算の度にサーバーにアクセスしないで済む
 - セキュリティの問題
 - 大規模もしくは更新の多いデータに依存する場合不向き

複数サーバーからのモバイルコード

- 個々のコードが別の組織から配布され、
- それがクライアント側で連結・実行される。



複数モバイルコードの利点・欠点

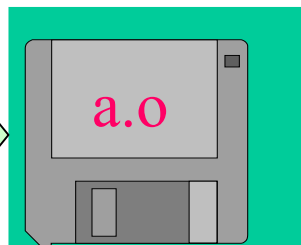
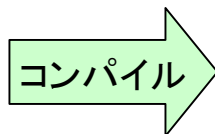
- アプリより細かい粒度(コンポーネント)でのソフトウェアサービスを「比較可能な商品」として流通させられる。
 - 同じインタフェースを持ち、性能や効果、値段が違うコンポーネントを取捨選択して利用可能.
- セキュリティポリシーの決定がより複雑になる.

Javaのロード・メカニズム

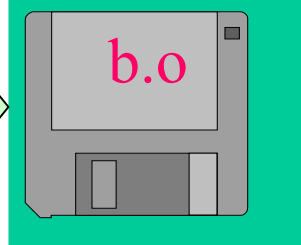
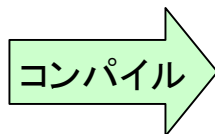
- ロード:
 - マシン語をマシンに読み込む作業.
- ローダー:
 - ロードを補助するためのプログラム.
- モバイルコードとのかかわり:
 - モバイルコードを受け付けるローダーでなければ, モバイルコードは実行できない.

Cのロード

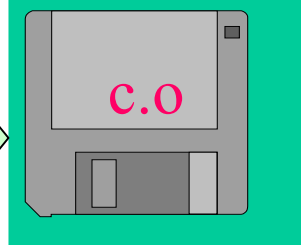
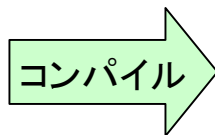
```
// a.c  
int foo(){  
    ....  
}
```



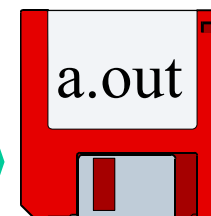
```
// b.c  
int bar(){  
    .....  
}
```



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



リンク



ロード

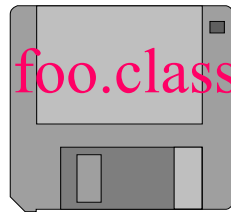
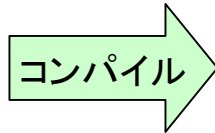


foo()
bar()
main()
.
.
.
.

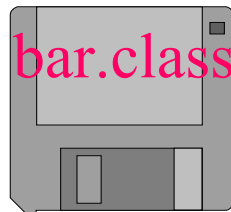
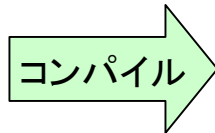
基本的にリンクされた関数は全てマシン内に読み込まれる。

Javaのロード

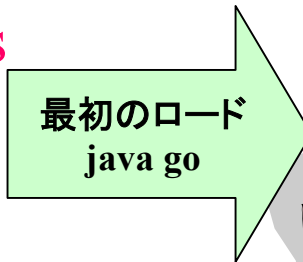
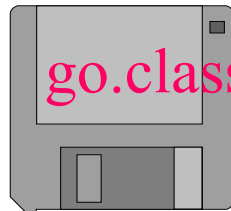
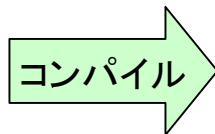
```
// foo.java  
class foo{  
  ....  
}
```



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



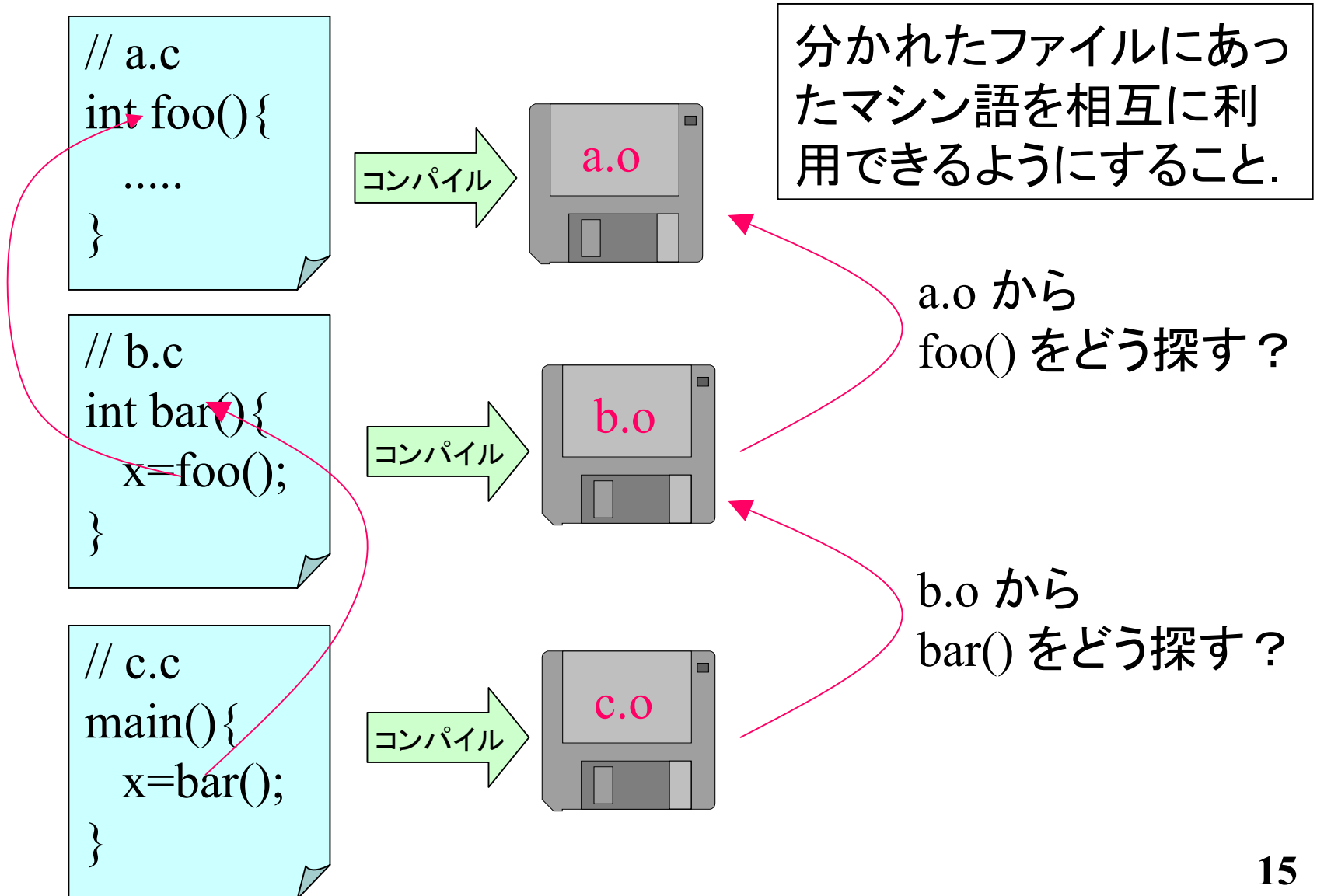
```
// go.java  
class go{  
  ... main(...){  
    if(x>10)  
      new foo();  
    else  
      new bar();  
  } ....  
}
```



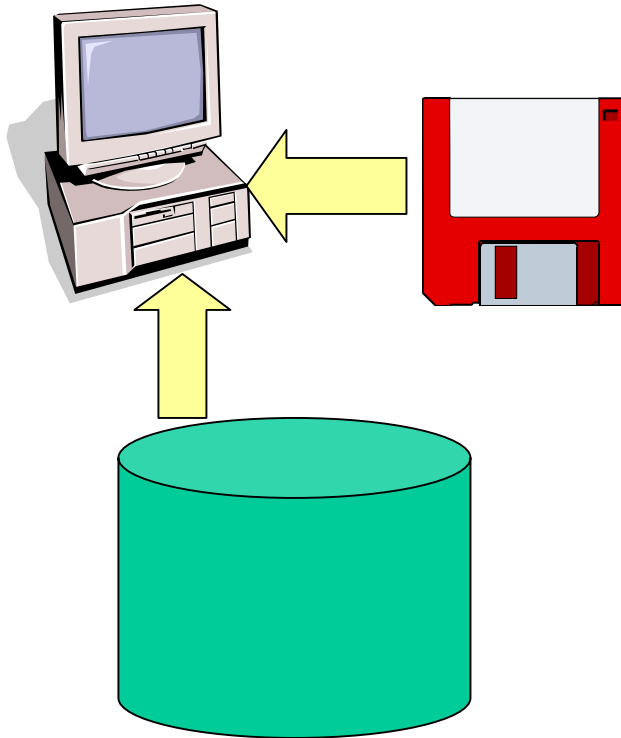
例えば,
go.class
foo.class
.
.
.

- 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.)

参考: リンク, リンケージ

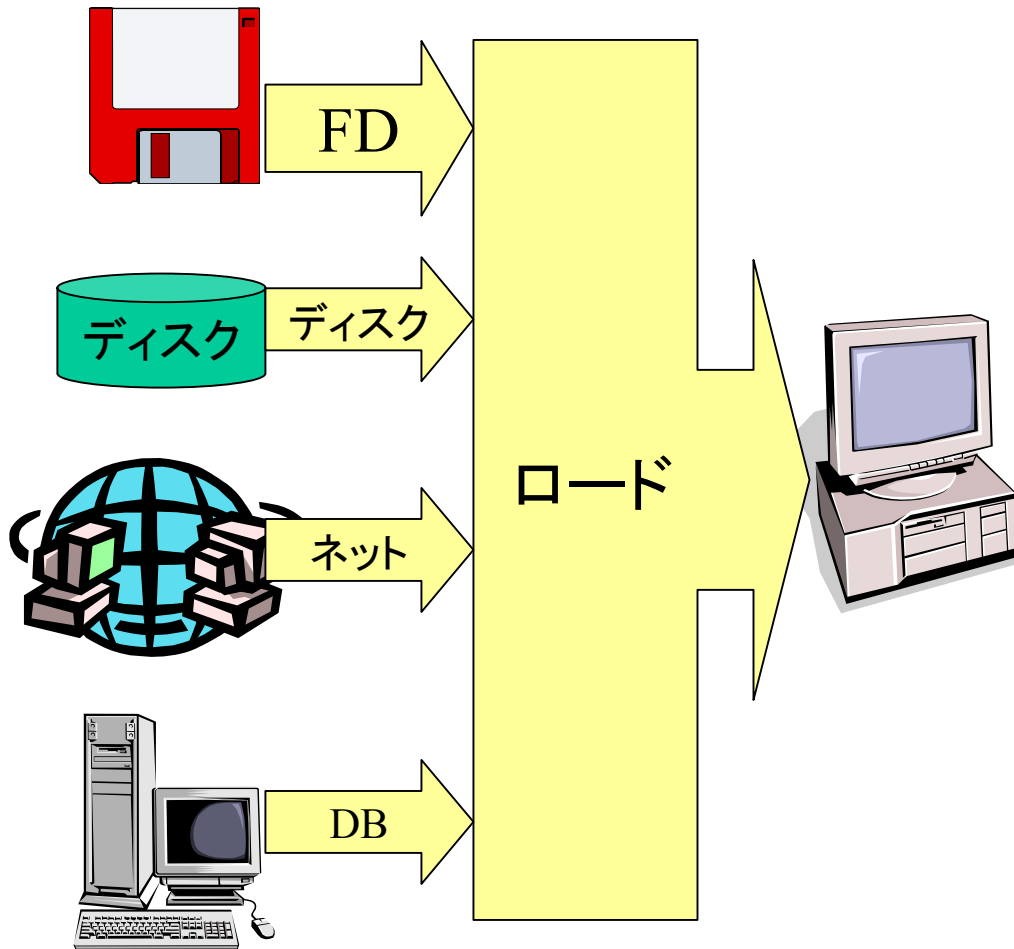


Cの場合のロード元



- 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。

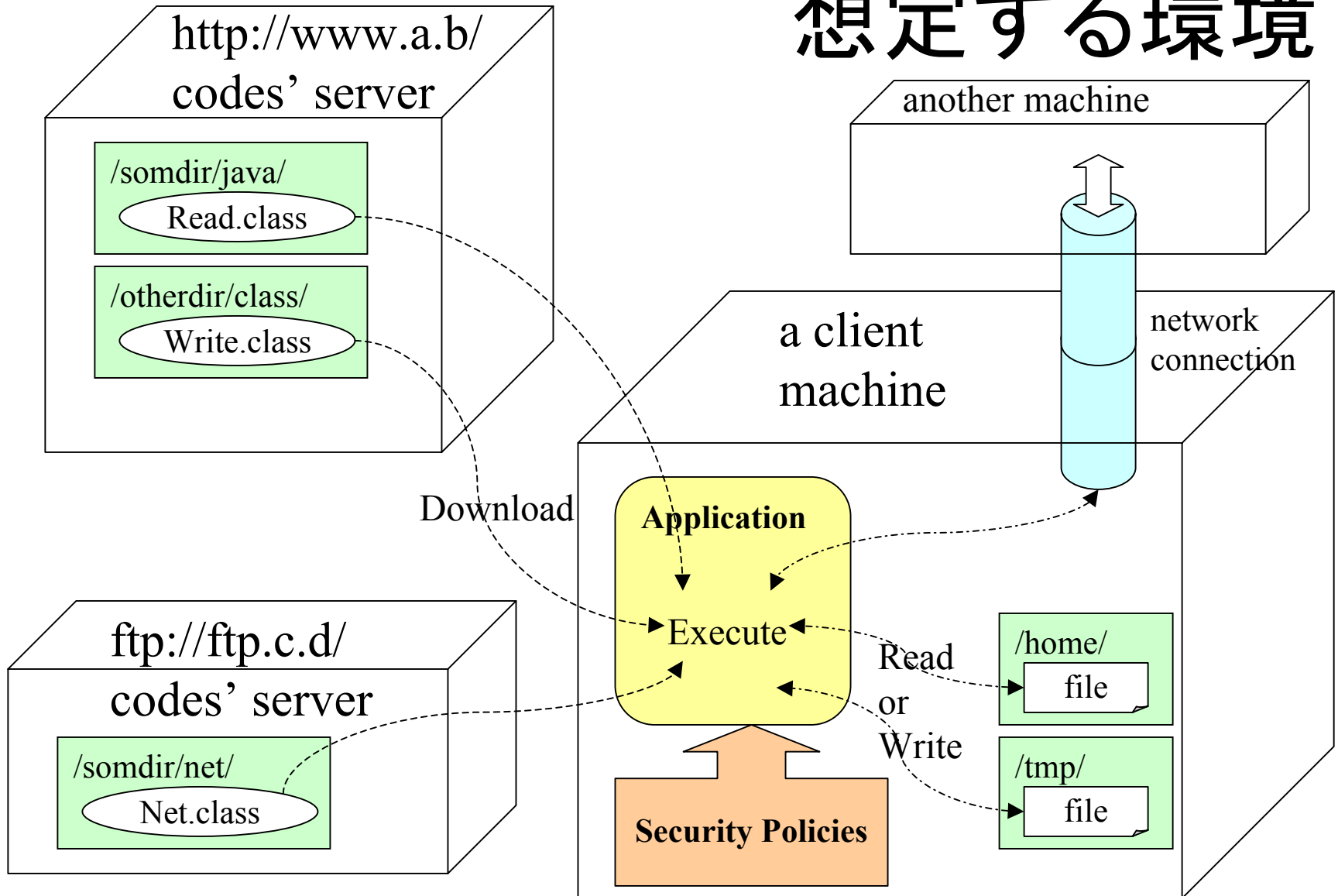
- 1つのプログラムが複数のソースを持つ。

- モバイルコードを受け付けるローダーを持つ。

本演習で扱う モバイルコードアプリケーション

- コードを動的にロード, アンロードして動作中でさえ機能変更が可能.
- CGI/ASP等, サーバーサイド技術よりスケーラブル.
- クライアント側のマシンに多数の外部コードを動的に読み込み・実行するので危険.
- Javaモバイルコードが一般的, かつ手軽に開発・テストできる状況にある.

想定する環境



NetMultiLoader.class

- 実際に前述のような複数モバイルコードの実行を起動するためのローダー.
- 起動の仕方は以下の通り

```
java -Djava.security.manager -Djava.security.policy==設定ファイル  
NetMultiLoader クラス名 引数
```

実際は一行で書きます

ココの=は二つです

- 「クラス名」で指定されたクラスのmainメソッドが、ネットワーク上からダウンロードされ、起動される.
- 続く「引数」は、そのmainメソッドの引数として扱われる.
- 指定されたクラスが必要は他のクラスも同様にダウンロード、実行される.
- ネットワークの検索パスは loadpath.xml というファイルに記載する.

実行準備

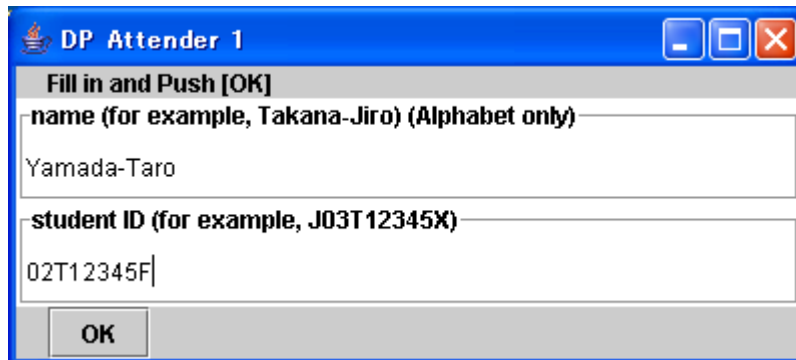
1. 以下のアーカイブをダウンロードし展開
<http://kaiya.cs.shinshu-u.ac.jp/dp/sec/loader1/kit.zip>
2. バッチファイルがあるので、それを実行
 - run.bat ポリシーファイル名 起動クラス名
3. ポリシーファイルはサンプルを提供済
 - all.txt 全てを許すポリシー
 - file.txt ファイル読み書きの許すポリシー
 - net.txt ネットワーク接続のみゆるすポリシー他
4. ソース等も以下のURLに公開済
<http://kaiya.cs.shinshu-u.ac.jp/dp/sec/loader1>

例題 出席確認

- 以下を実行

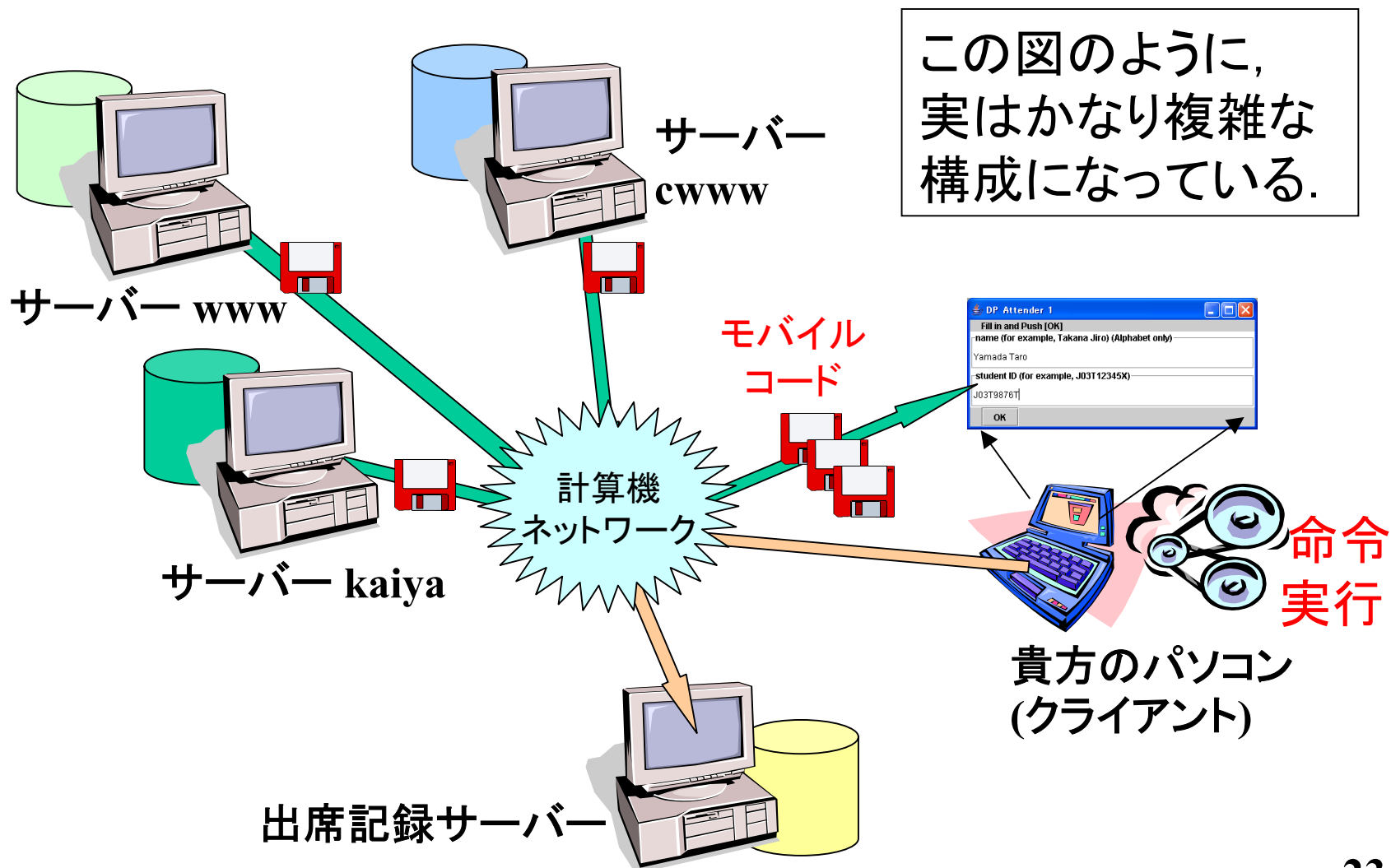
```
run.bat all.txt Attend1
```

- 以降の五回の実験では必ず以上を実行してください。(出席確認になります)
- 以下のようなフォームに名前と学籍番号を入れて。(英数文字しか入りません)



The screenshot shows a Windows dialog box titled "DP Attender 1". The dialog has a blue title bar with standard minimize, maximize, and close buttons. The main area is white with a grey border. At the top, it says "Fill in and Push [OK]". Below that, there are two input fields. The first is labeled "name (for example, Takana-Jiro) (Alphabet only)" and contains the text "Yamada-Taro". The second is labeled "student ID (for example, J03T12345X)" and contains the text "02T12345F". At the bottom left, there is an "OK" button.

出席確認コードの構造



Javaセキュリティポリシー

- 実行しようとするJavaモバイルコードについて,
 - 配布元の場所
 - サイン(著名)等の有無をもとに,
- そのコードの振る舞いに許可を与える(grant).
 - permission 操作の型 (ファイル操作等)
 - target 操作対象 (ディレクトリ指定等)
 - action 許可されるアクション (読み書き等)
- 禁止(revoke)指定はできない.
- アプリケーション毎に異なるポリシーを与えることができる.
- 詳細は別ファイル参照.

ポリシー設定の実例

```
grand codeBase "http://www.a.b/staff/" {  
    permission java.io.FilePermission "/answer/*", "read";  
    permission java.io.FilePermission "/score/*", "read,write";  
    permission java.io.FilePermission "/remark/*", "read,write";  
};
```

```
grand codeBase "http://www.a.b/learner/" {  
    permission java.io.FilePermission "/answer/*", "read,write";  
    permission java.io.FilePermission "/score/*", "read";  
    permission java.io.FilePermission "/remark/*", "read,write";  
};
```

```
grand codeBase "http://www.a.b/colearner/" {  
    permission java.io.FilePermission "**", "read, write";  
};
```

ファイルのみでなく、いくつかの既存permissionがあり、
加えて、独自のpermissionを作成することもできる。

簡単な例題

- ファイルの読み書きのアクセス権
- ネットワーク接続のアクセス権
- プロパティのアクセス権
 - コレは次週

課題S1

- 例題の実行確認
- ポリシーを変えると機能しないことを確認.