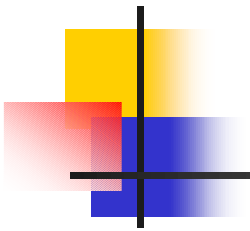
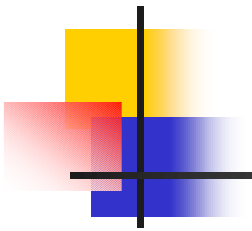
オブジェクト指向AtoZセミナー
分散オブジェクト環境HORB入門

萩本順三　（株）豆蔵
hagimoto@mamezou.com
http://www.mamezou.com



今日のお話

- HORBとは
- HORB Openの活動
- HORBの設計コンセプト
- ベンチマーク
- プログラミング入門



HORBとは

- 日本で生まれたJava分散オブジェクト技術
  - ・電総研平野博士により95年にリリース
  - ・世界中にHORB利用者がいる
- オープンソース
  - ・商用研究にて無償利用可
  - ・HORB本体のソースが公開されている
- Javaを知っていれば誰でも使える

HORBの歴史
http://www.horb.org/

95年

世界初Javaベース分散OO技術としてデビュー

工業技術院電子総合研究所・平野博士が開発

99年

ライセンスがオープンソースに変更される

電総研とNJKの共同研究＆委託開発開始

2000年

HORB Openによってオープンソース運営開始

HORB ver2.0リリース

HORBの主要な機能と特徴

100%

SUNのJavaと100%互換性があり、あらゆるJava処理系で動作します。

のサポート

CORBA　IIOP（Ver2.0）をサポートしています。

レス

IDLを書く必要はありません。あなたのクラス定義がそのままリモートオブジェクトになります。CORBAプロトコルを使用する場合でもIDLを書く必要はありません。CORBA IDLはクラス定義から自動生成されます。もちろんIDLコンパイラも装備しています。

超高速・軽量

RMIや他のORBの2倍以上高速です。またランタイム環境が小さいため組み込み用途にも向いています。

動的オブジェクト生成

HORB特有のリモートオブジェクトの動的生成と接続が簡単に行えます。

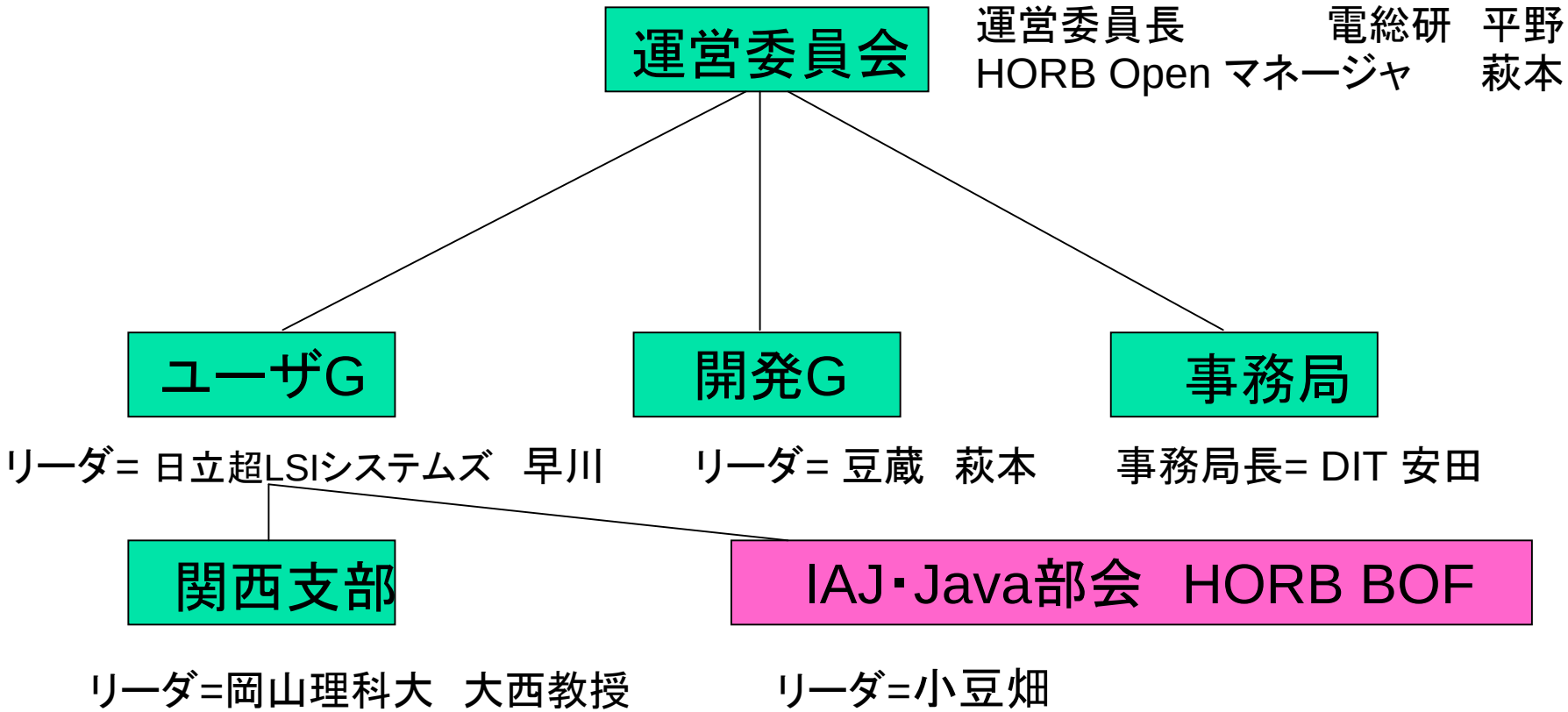
非同期メソッド

非同期メソッド呼び出しとFutureをサポートしています。組み込み用途に向いています。

オープンソース

Javaで書かれたフルソースコードがパッケージについてます。商用でも無償でも使えます。

HORB Open運営組織
http://www.horbopen.org/



学習教材・記事

萩本順三： JavaPress １１月発売号から連載。

HomePage関連

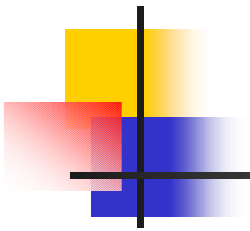
萩本順三： @IT技術情報（Javaソリューション）
http://www.atmarkit.co.jp
HORBの連載　１０月中旬より開始

セミナー（月１度を目標）

HORB入門、アーキテクチャ概説、デザインパターン

第４回HORBシンポジューム（発表者募集開始！！）

日時：平成１３年３月２日（予定）

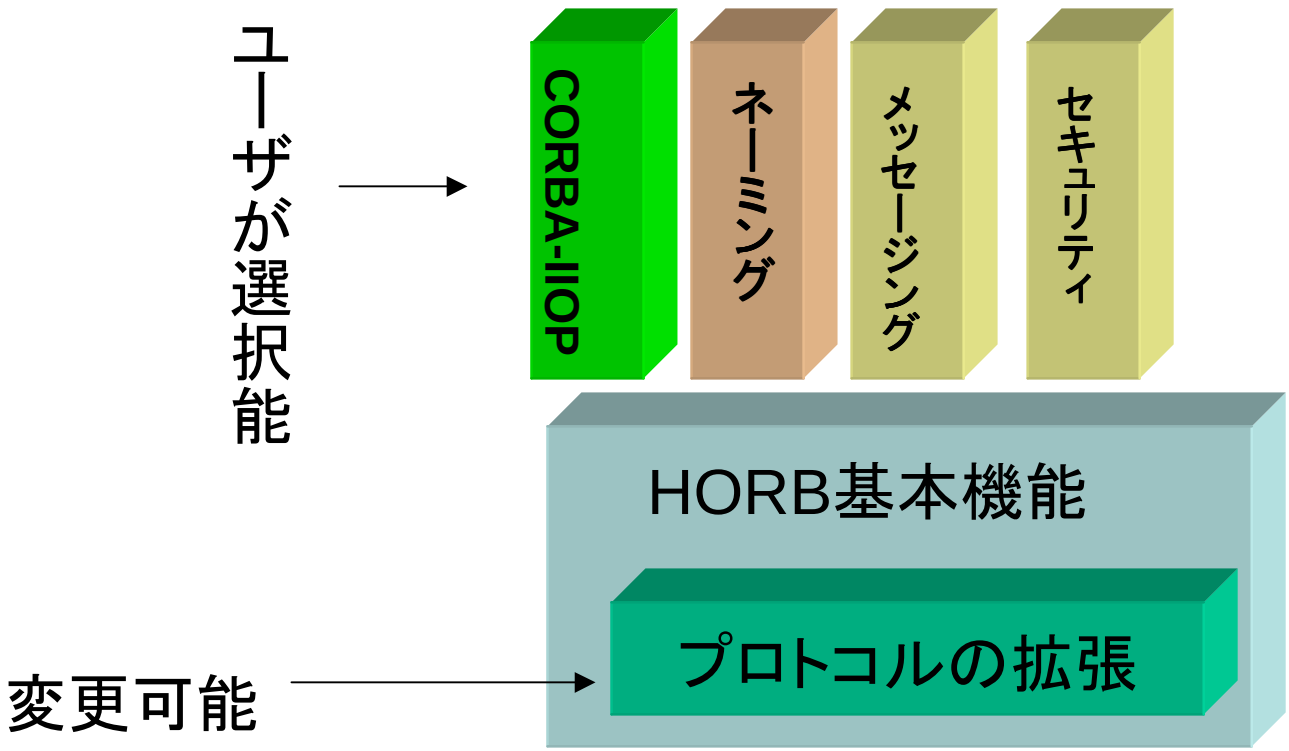


HORBが提供する世界

- Javaを知っていれば小・中学学生でも使える分散オブジェクト環境
  - ・やさしいプログラミングインタフェース
  - Java言語にシームレスなプログラミング環境
- 拡張可能なORBプロトコル
  - 自社製ORBプロトコルを開発可能
- 高速・軽量
  - 組み込み系にも向いている

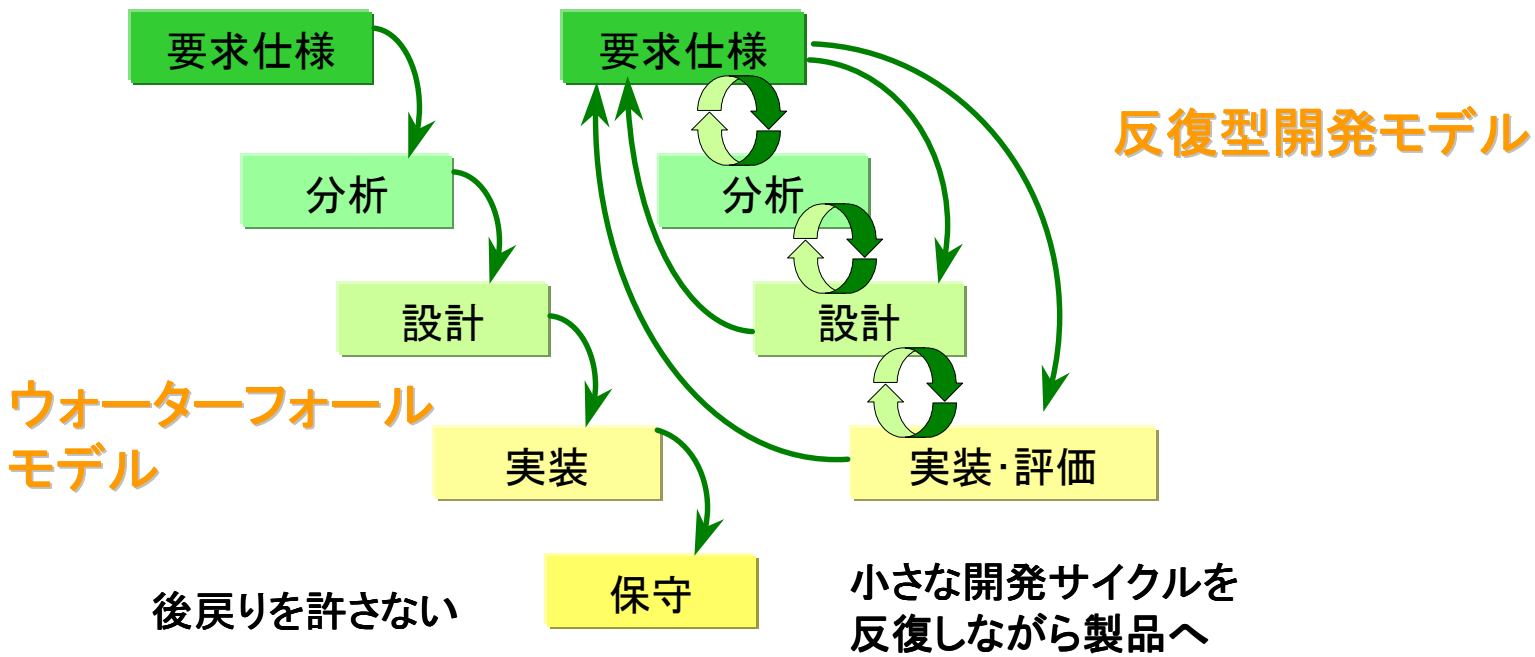
HORBの設計コンセプト

- 利用者にやさしいプログラムインタフェースを維持しながら拡張可能なORBを追求

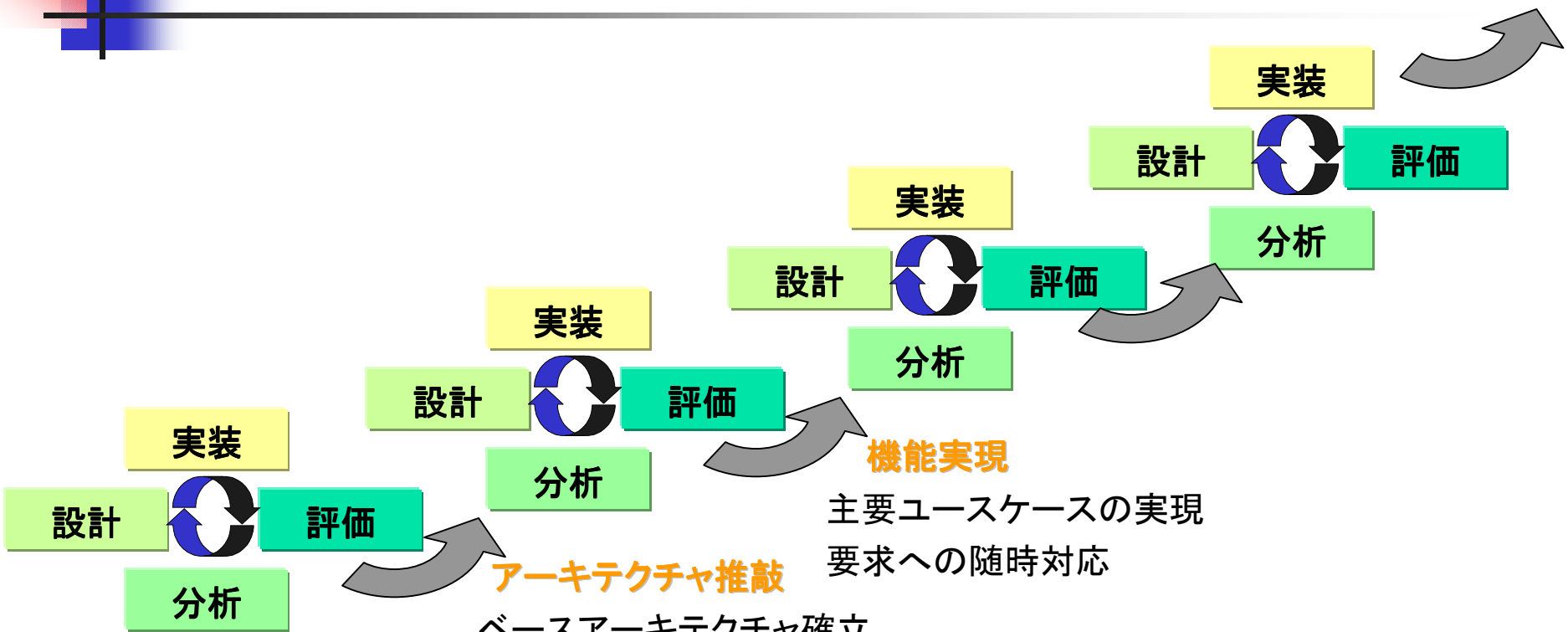


やさしいプログラムインタフェース（なぜHORBにとって重要か？）

- それは初心者にとって入りやすいということだけではない。
- 現在のソフトウェア開発プロセス（反復型）



反復型開発



方向付け

問題領域の大局的把握
実現可能性の判断
第1次見積もり

アーキテクチャ推敲

ベースアーキテクチャ確立
コア機能の実現
技術リスクの早期解決

機能実現

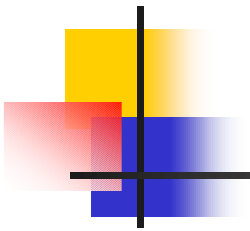
主要ユースケースの実現
要求への随時対応

反復開発

反復開発における重要事項

- 設計は初めからよいものはできない。
- アイデアをそのままの形で実現する仕組みが必要
- 上記がHORBの目指す分散オブジェクト環境
  - アイデアをオブジェクト・イメージに！
  - オブジェクトイメージを実際のクラスに！
  - クラスをそのまま分散化へ
  - 反復開発する間に、リスクとメリットを天秤にかけながら最良のデザインへ成長させていく。

- 技術リスクの早期解決
  - 技術的にリスクが大きいところを早めに解決



HORBの挑戦(反復開発)

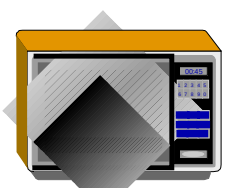
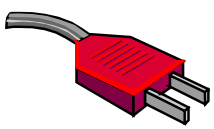
- インタフェースを必要としない
  - 思いついたクラスをそのままORB化が可能
  - 継承可能なリモートオブジェクト
- サーバアプリケーションを必要としない
  - 代理オブジェクトを生成するとサーバ側にリモートオブジェクトが自動的に生成される。
- 非同期メソッド
  - 通常の呼び出しと同じように簡単に使用可能
- ネーミングサービスなどは拡張機能として用意
  - 必要に応じてダウンロード利用可能

仮想世界における人にわかりやすい単位は何？？

- 手がかり
  - 目を瞑って実世界をイメージするときに、もっとも自然な最小の単位とは何でしょうか？
- ものを仮想空間に作り出すのがオブジェクト指向の考え。
- 分散オブジェクト環境においても、オブジェクトの自然な表現が重要。



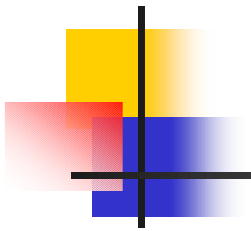


「もの」の認知

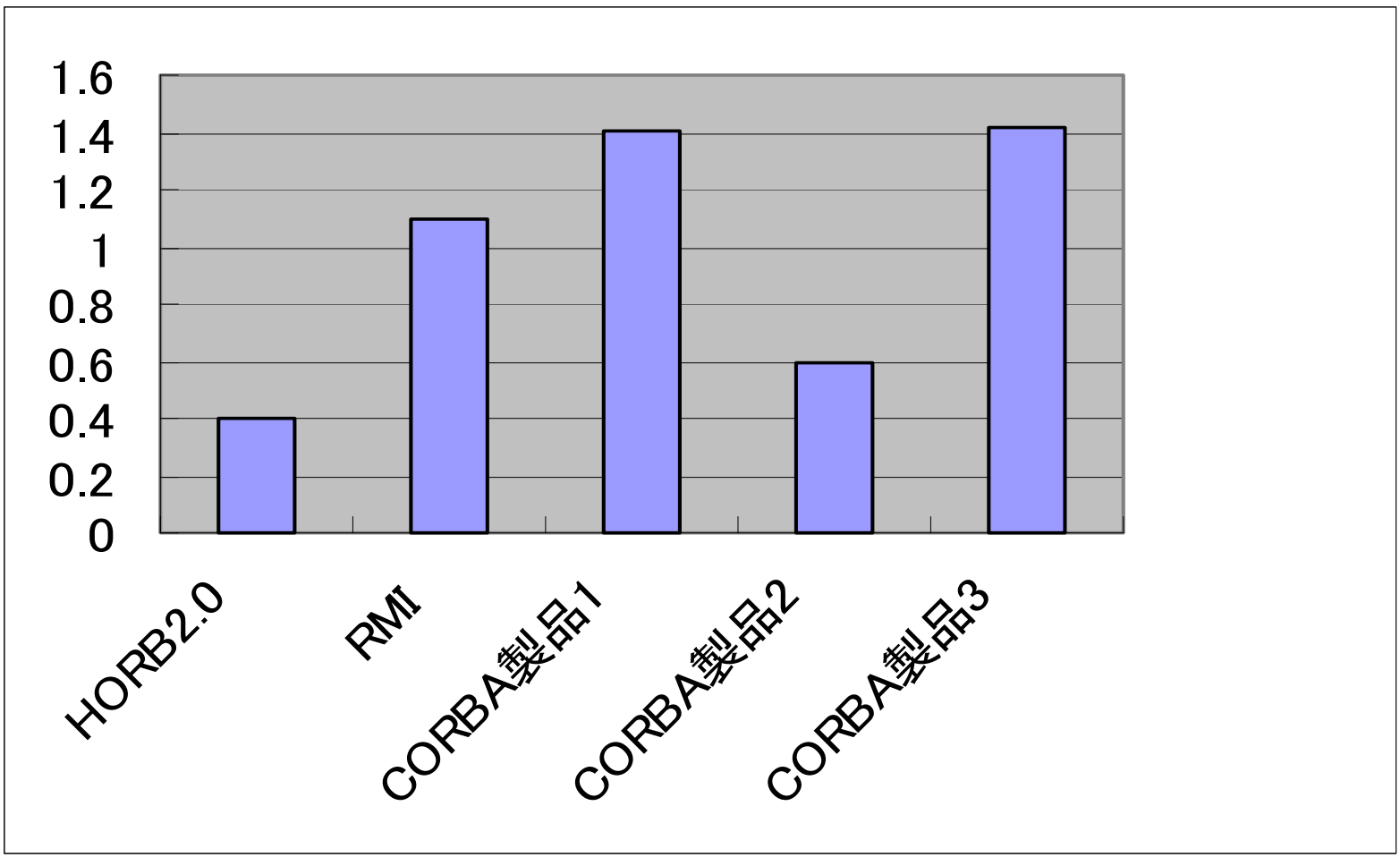
人は「もの」を認知するとき、「もの」が存在する意味や役割を捉える。

HORBの目指す分散オブジェクト環境



- イメージしたアイデアをオブジェクトとして分散環境に容易に配置できる。
- HORBのターゲット
  - 組み込む機器
  - アプリケーション開発
  - 学校教育
  - 研究部門
  - 専門分野
    - 医療
    - アート
    - ゲーム

ベンチマーク
(メソッド呼び出し　JDK1.1.8)



HORBにチャレンジ

- 通常のJavaプログラム

```
public class  Test{
   public String greeting(String s){
      System.out.println(s);
      return "こんにちは、Testです。";
   }
}
```

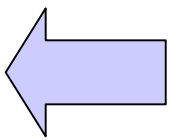
```
>java Client
こんにちは、Testです。
こんにちは、Clientです。
```

```
public class  Client{
            public static void main( String argv[] ){
              Test  test = new Test();
              String result = test.greeting("こんにちは、Clientです。");
              System.out.println(result);
            }
}
```

HORBよるネットワークプログラミング

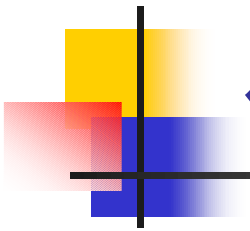
- HORBのプログラム

```
public class  Test{
   public String greeting(String s){
      System.out.println(s);
      return "こんにちは、Testです。";
   }
}
```



変更なし

```
public class  Client2{
        public static void main( String argv[] ){
            String host = (argv.length == 1) ? argv[0] : "localhost";
            Test_Proxy  test = new Test_Proxy("horb://"+host);
            String result = test.greeting("こんにちは、Clientです。");
            System.out.println(result);
        }
}
```



代理オブジェクトの利用

```
String host = (argv.length == 1) ? argv[0] : "localhost";
Test_Proxy  test = new Test_Proxy("horb://"+host);
```

- Test_Proxyは代理オブジェクト。
- 代理オブジェクトをTestオブジェクトと思えばよい
- 引数の horb は、プロトコル名
- hostは、接続するホスト名

HORBの実行

- HORB実行

(1)HORBサーバを起動する

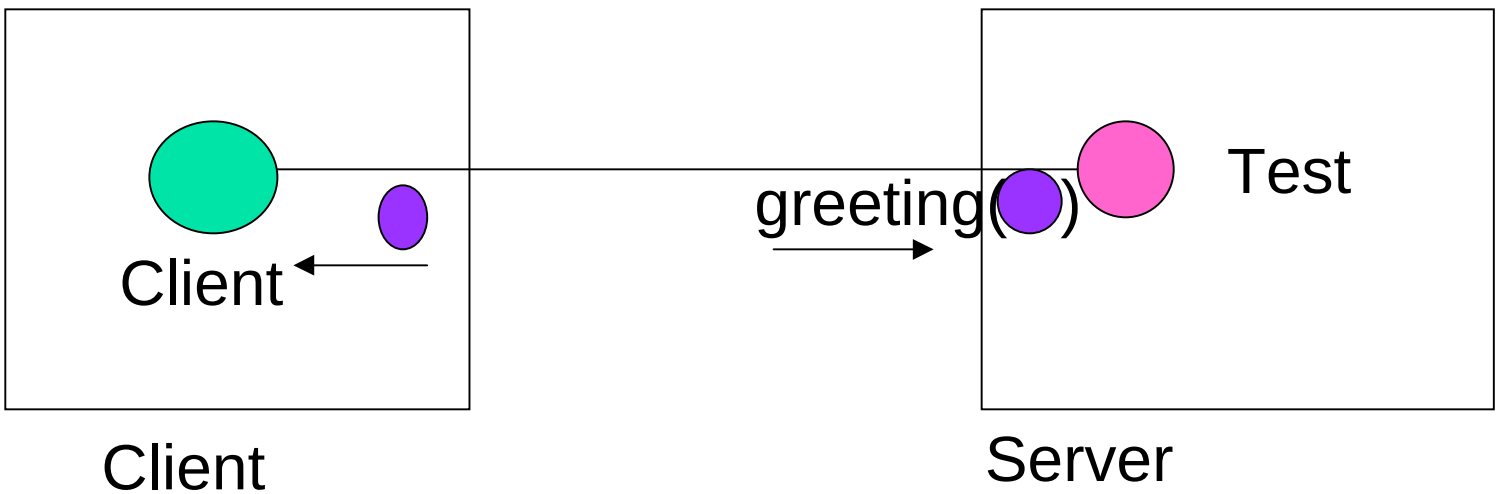
```
＞HORB
こんにちは、Clientです。
```

(2)クライアントを起動する。

```
> java Client
こんにちは、Testです。
```

HORBの動作イメージ

- サーバにあるTestオブジェクトにメッセージが飛ぶ
- ● はStringオブジェクト



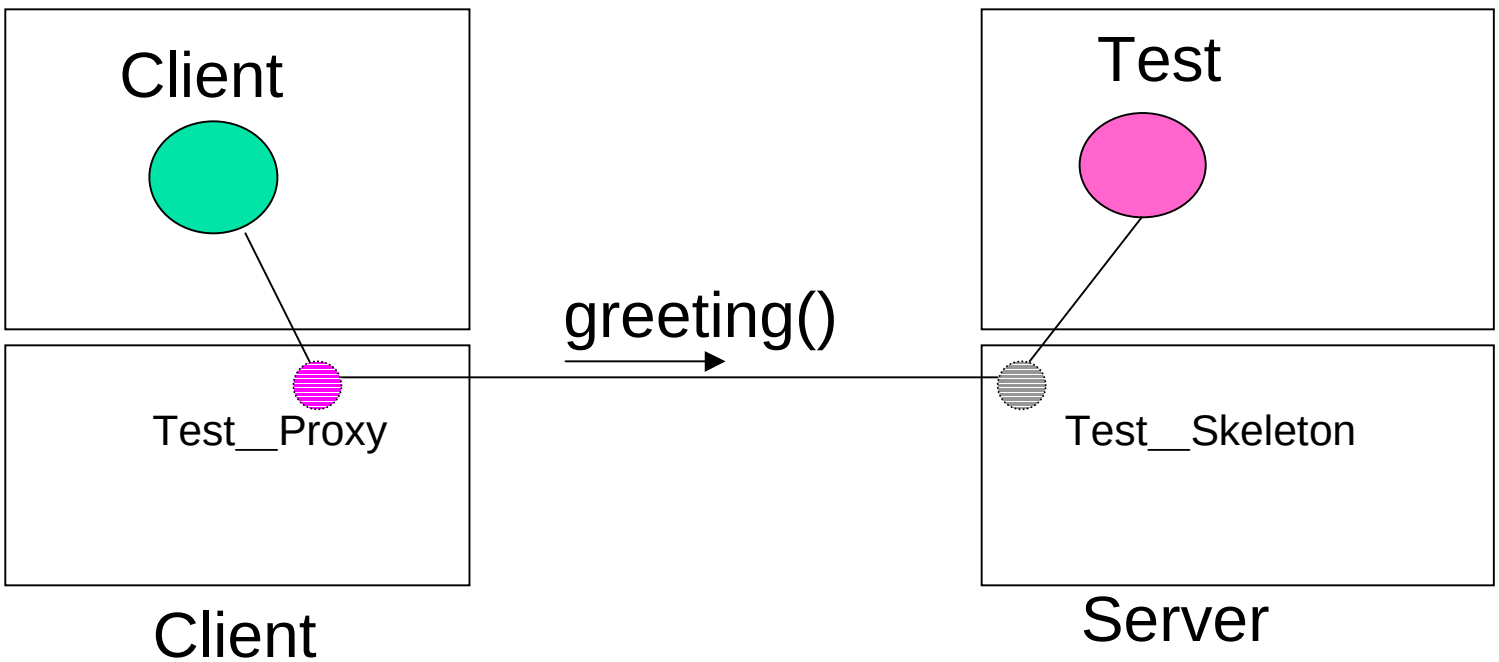
HORBの実行まで

- HORBコンパイル

```
>Horbc  Test.java
compiling Test.java
generating Test_Proxy.java
compiling Test_Proxy.java
generating Test_Skeleton.java
compiling Test_Skeleton.java
>java Client. Java
```

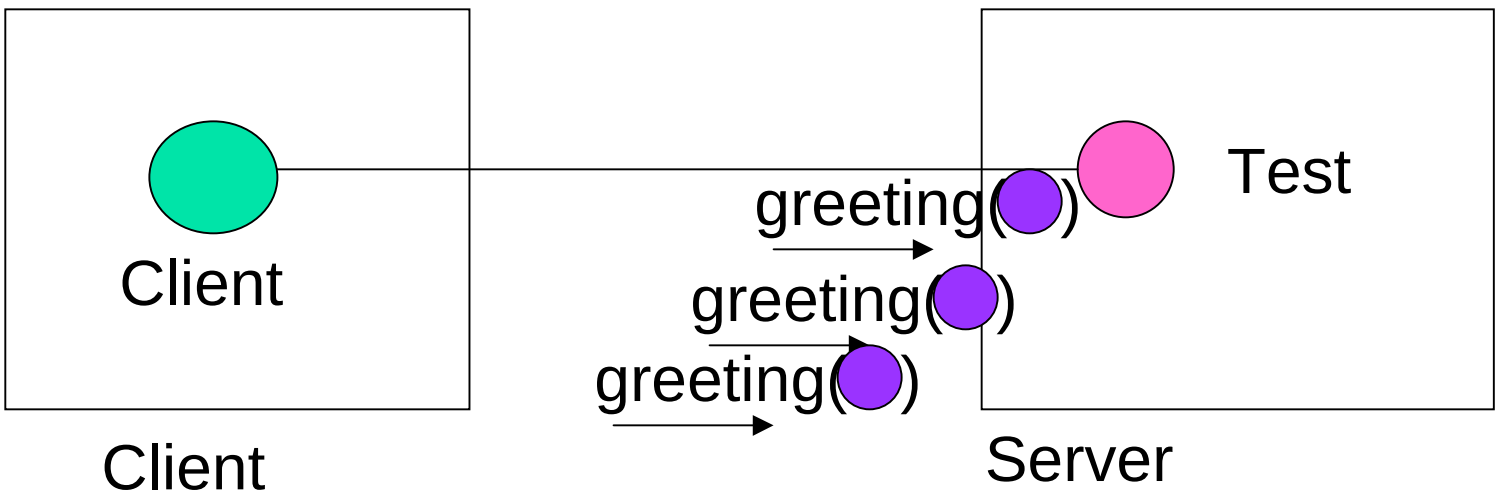
分散OOアーキテクチャ

- HORBコンパラで作成した代理オブジェクトとスケルトンを使ってメソッド呼び出しとオブジェクト転送を実現
- java.io.SerializableをimplementsしていればObject転送可能



非同期メソッドに挑戦

- サーバのメソッドを非同期的に呼び出し可能

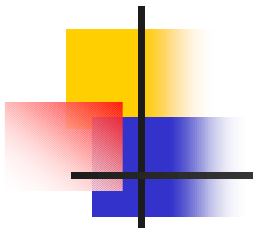


非同期メソッドに挑戦

- 通常の同期メソッド呼び出し

```
public class  Test2{
   public void greeting(String s,int time){
      try{ Thread.sleep(time*1000);}catch(Exception e){}
      System.out.println(s+" waiting"+time+"sec.");
   }
}
```

```
public class  Client2{
        public static void main( String argv[] ){
            String host = (argv.length == 1) ? argv[0] : "localhost";
            Test2_Proxy  test = new Test2_Proxy(“horb://”+host);
            for(int i=4;i>0;i--)
                  test.greeting("message"+(5-i),i);
        }
}
```



- 実行結果
  - 前の呼び出しに待たされる

```
>HORB
 message1 waiting4sec.
 message2 waiting3sec.
 message3 waiting2sec.
 message4 waiting1sec.
```

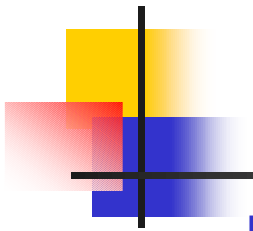
```
> java Client
```

非同期メソッドに挑戦

- 非期メソッド呼び出し (OneWay)

```
public class  Test3{
   public void greeting_OneWay(String s,int time){
      try{ Thread.sleep(time*1000);}catch(Exception e){}
      System.out.println(s+" waiting"+time+"sec.");
   }
}
```

```
public class  Client3{
            public static void main( String argv[] ){
                String host = (argv.length == 1) ? argv[0] : "localhost";
              Test3_Proxy  test = new Test3_Proxy("horb://"+host);
              for(int i=4;i>0;i--)
                  test.greeting_OneWay("message"+(5-i),i);
             try{ Thread.sleep(6000);}catch(Exception e){}
            }
}
```



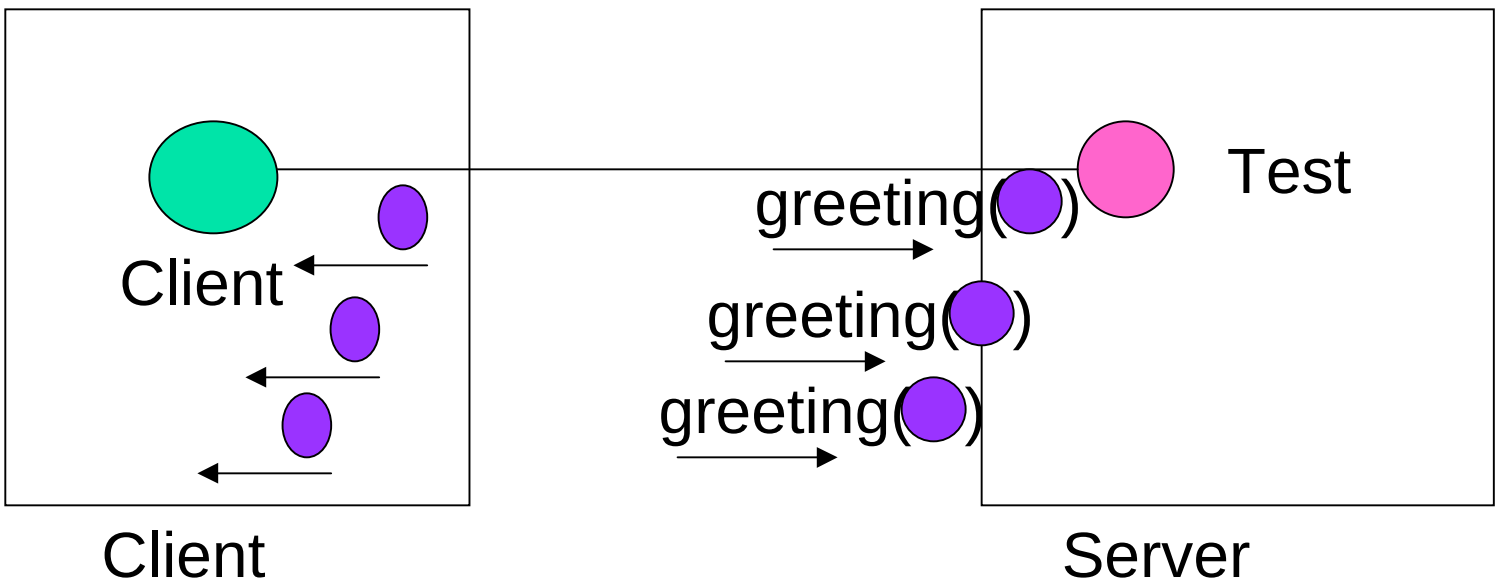
- 実行結果
  - 早く終了した順に表示されている

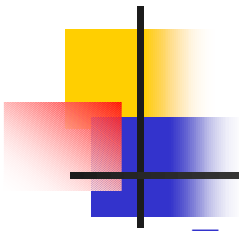
```
>HORB
 message4 waiting1sec.
 message3 waiting2sec.
 message2 waiting3sec.
 message1 waiting4sec.
```

```
> java Client
```

非同期メソッド(Async)に挑戦

- サーバのメソッドを非同期的に呼び出し可能
- 戻り値も取得可能





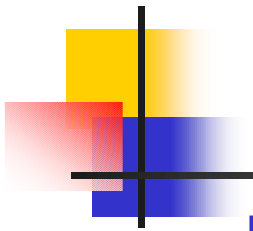
非同期メソッド(Async)に挑戦

- 戻り値あり非期メソッド呼び出し(Async)
  - 引数と戻り値にVector
  - 引数に渡されたVectorに文字列追加・リターン
  - 下記をHORBコンパイルすると
    - greeting_Requestメソッド
    - greeting__Receiveメソッドが生成される。

```
import java.util.*;
public class  Test4{
   public Vector greeting_Async(Vector v,int time){
      try{ Thread.sleep(time*1000);}catch(Exception e){}
      v.add(" waiting"+time+"sec.");
      return v;
   }
}
```

非同期メソッド(Async)に挑戦

```
import java.util.*;
import horb.orb.*;
public class  Client4{
            public static void main( String argv[] ){
                String host = (argv.length == 1) ? argv[0] : "localhost";
                  Test4_Proxy  test = new Test4_Proxy("horb://"+host);
                ResultAsync[] ra = new ResultAsync[5];
                for(int i=4;i>0;i--){
                     Vector v = new Vector();
                     v.add("message"+(5-i));
                     ra[i-1] = test.greeting_Request(v,i);
                  }
                for(int i=3;i>=0;i--){
                     Vector v = test.greeting_Receive(ra[i]);
                     System.out.println(v);
                  }
            }
}
```



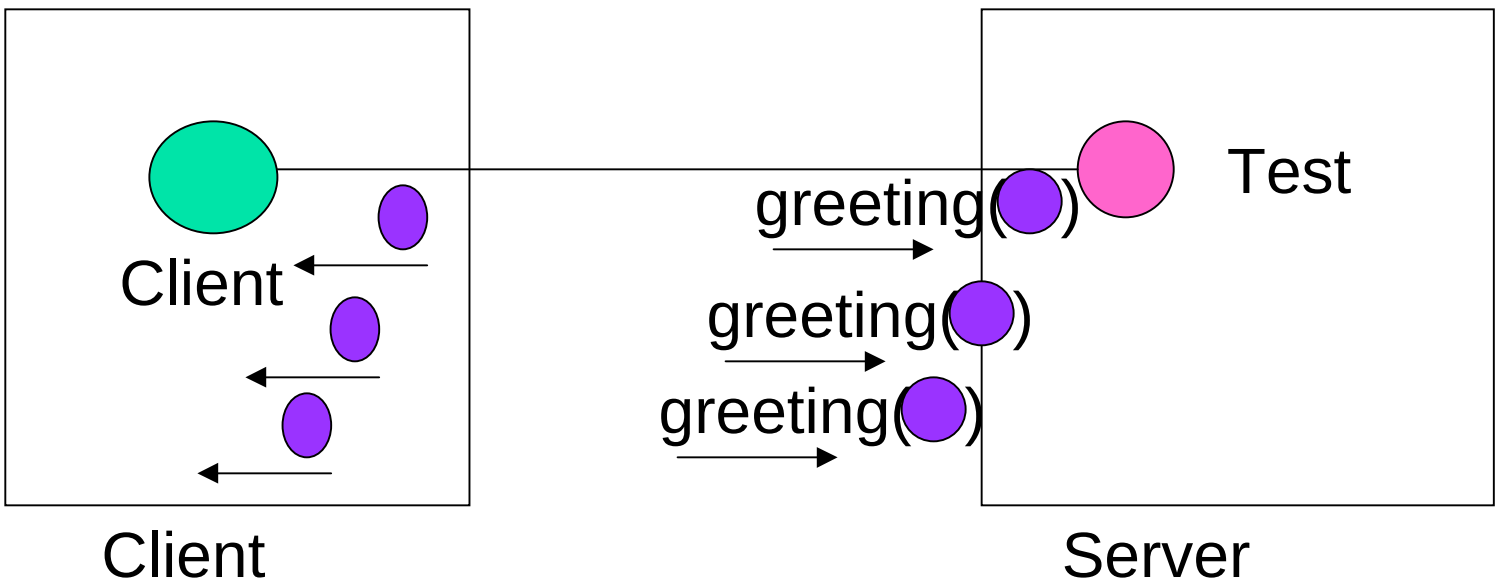
- 実行結果
  - 非同期メソッドの戻り値がgreeting_Receiveによって取得できる

```
>HORB
```

```
> java Client
[message1,   waiting4sec.]
[message2,   waiting3sec.]
[message3,   waiting2sec.]
[message4,   waiting1sec.]
```

Asyncコールバックに挑戦

- サーバのメソッド処理が完了したら、クライアントに通知される。
- 完了通知は、runメソッドが別スレッドで呼ばれる。



Asyncコールバックソース no1

- 完了通知を受けたいクラスは、AsyncMethodHandlerのrunメソッドを実装する。

```
import java.util.*;
import horb.orb.*;
public class  Client5 implements horb.orb.AsyncMethodHandler{
    static String host;
    static Test4_Proxy  test;
        public static void main( String argv[] ){
                host = (argv.length == 1) ? argv[0] : "localhost";
                new Client5().test();
        }
//続く
```

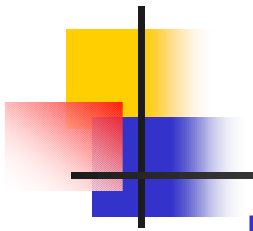
Asyncコールバックソース no2

- setHandlerの第2引数は、runの第2引数に。

```
protected void test(){
        test = new Test4_Proxy("horb://"+host);
        ResultAsync[] ra = new ResultAsync[5];
        for(int i=4;i>0;i--){
            test._setHandler(this, 5-i);
            Vector v = new Vector();
            v.add("message"+(5-i));
            ra[i-1] = test.greeting_Request(v,i);
        }
        try{ Thread.sleep(5000);}catch(Exception e){}
    }
public void run(ResultAsync ra, int tag) {
        Vector v = test.greeting_Receive(ra);
        System.out.print(tag+"番目の呼び出し..");
        System.out.println(v);
}
}
```

Thisは、完了通知を受けたいObject。

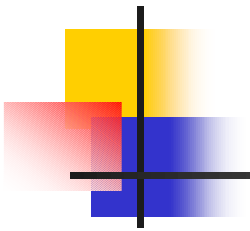
raは完了した戻り値を持っているThread
たtagは、setHandlerで渡したタグ。



- 実行結果
  - 非同期メソッド処理が終わり次第、クライアントのrunが呼ばれResultAsyncが渡される。

```
＞HORB
```

```
> java Client
4番目の呼び出し..[message4,  waiting1sec.]
3番目の呼び出し..[message3,  waiting2sec.]
2番目の呼び出し..[message2,  waiting3sec.]
1番目の呼び出し..[message1,  waiting4sec.]
```



まとめ

- HORBは簡単
- オープンソースなので無償で利用
- ソースが公開されているので安心
- ダウンロード方法・インストール方法は
  - ＠IT　Javaソリューションページの HORBで遊ぼう参照
  - http://www.atmarkit.co.jp/fjava/index.html
- 君もHORB Openの仲間に
  - まずは、horb-jへ入会
    - http://www.horbopen.org/
- 開発者を募集中