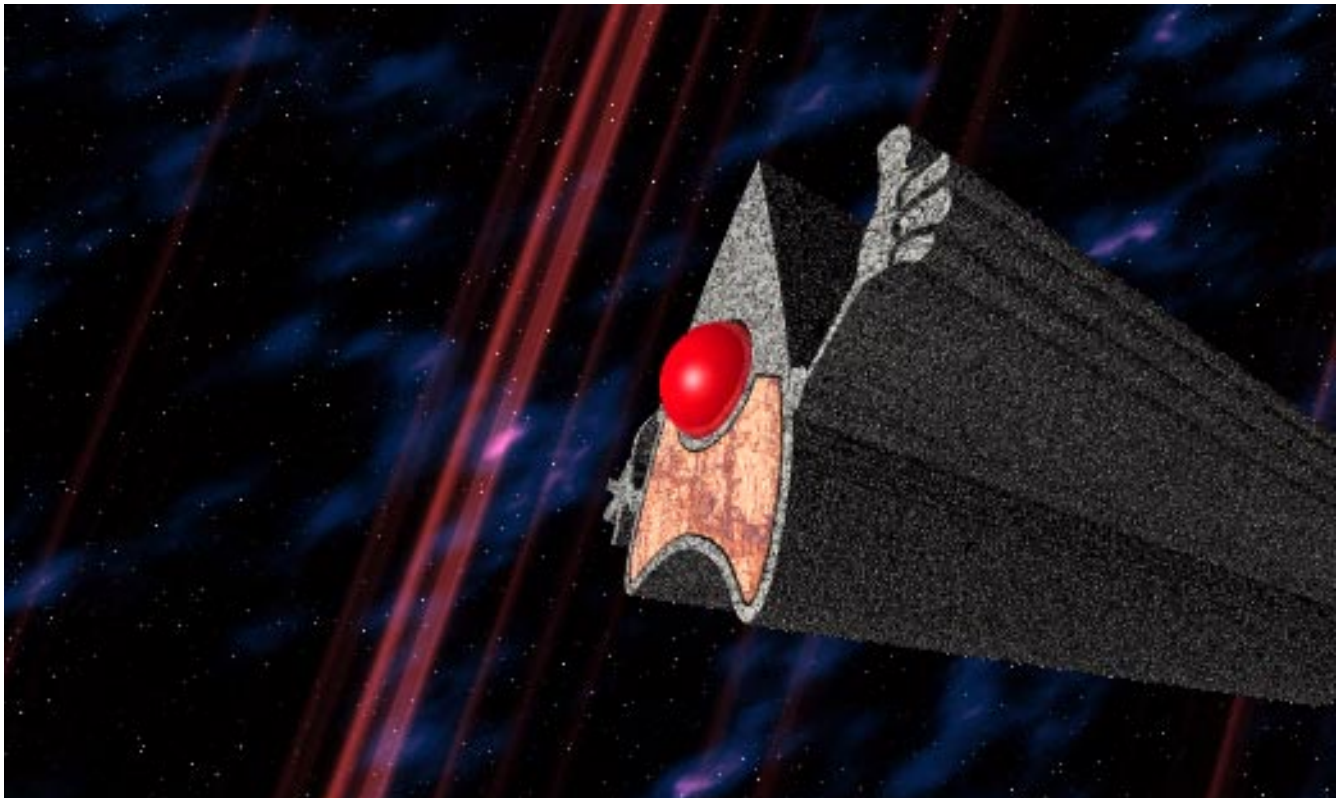
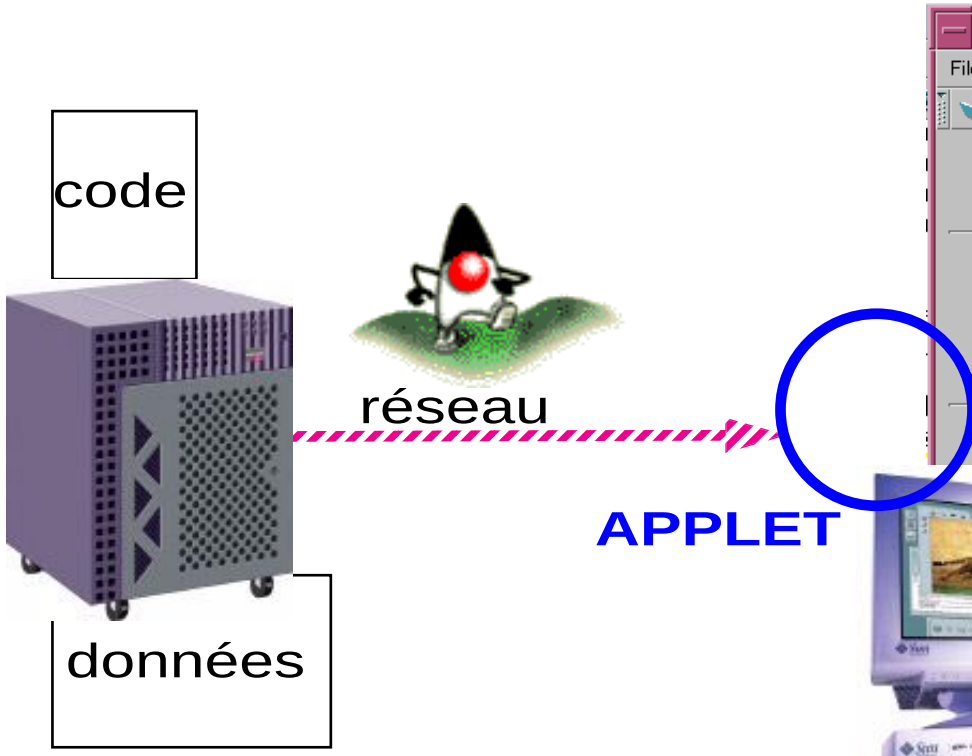
LA GALAXIE JAVA

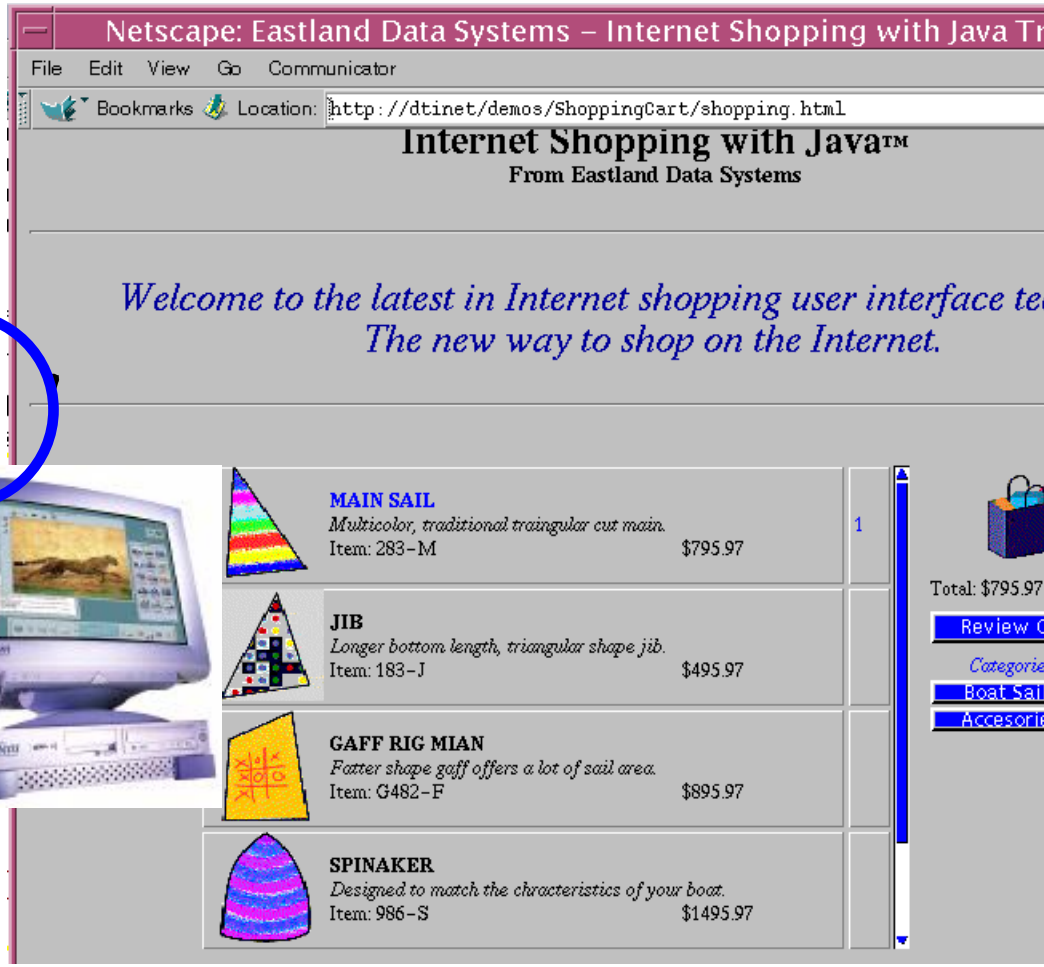


La galaxie JAVA

- Les lignes de force
- Les raisons d'un succès
- JAVA dans le système d'information
- Comment ça marche? (et est-ce que ça marche?)
- L'Objet selon Java

Les lignes de force





Les lignes de force

- code multi-plateformes
- code dynamique
- organisation/modularité
- sécurité
- multi-tâches

programmer en Java?

- formulations élégantes
- souci constant de génie logiciel
- dispositifs de sécurité
- compromis bien dosés

techniques complémentaires

- composants “beans”
- e.j.b.
- servlets
- distribution (RMI, JINI)

librairies

utilitaires programmation

calculs

E/S, E/S objets

internationalisation

réseau

I.H.M. portables, graphique

objets distants

composants dynamiques

accès B.D.D.

sécurité

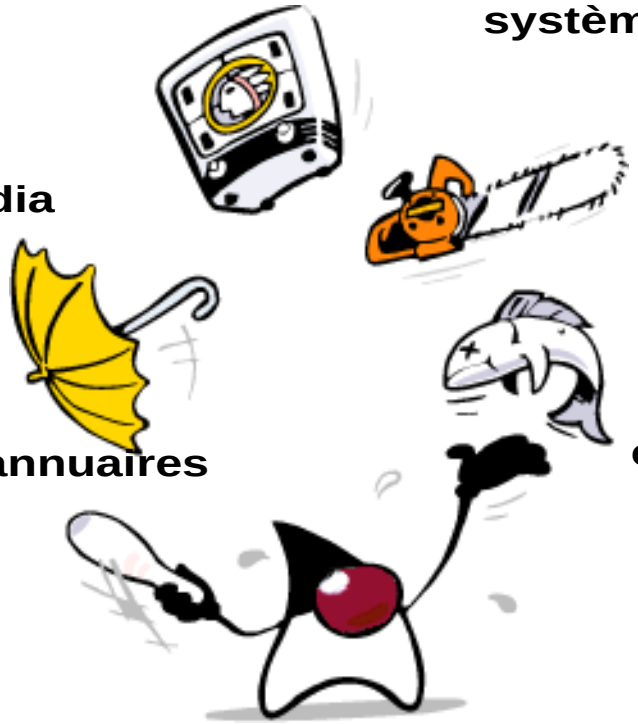
extensions

utilitaires devt.

système, deploiement

graphique, 3D, multimedia

b.d.d, transactions



réseau, services répartis, annuaires

composants d’entreprise

embarqué

échanges sécurisés

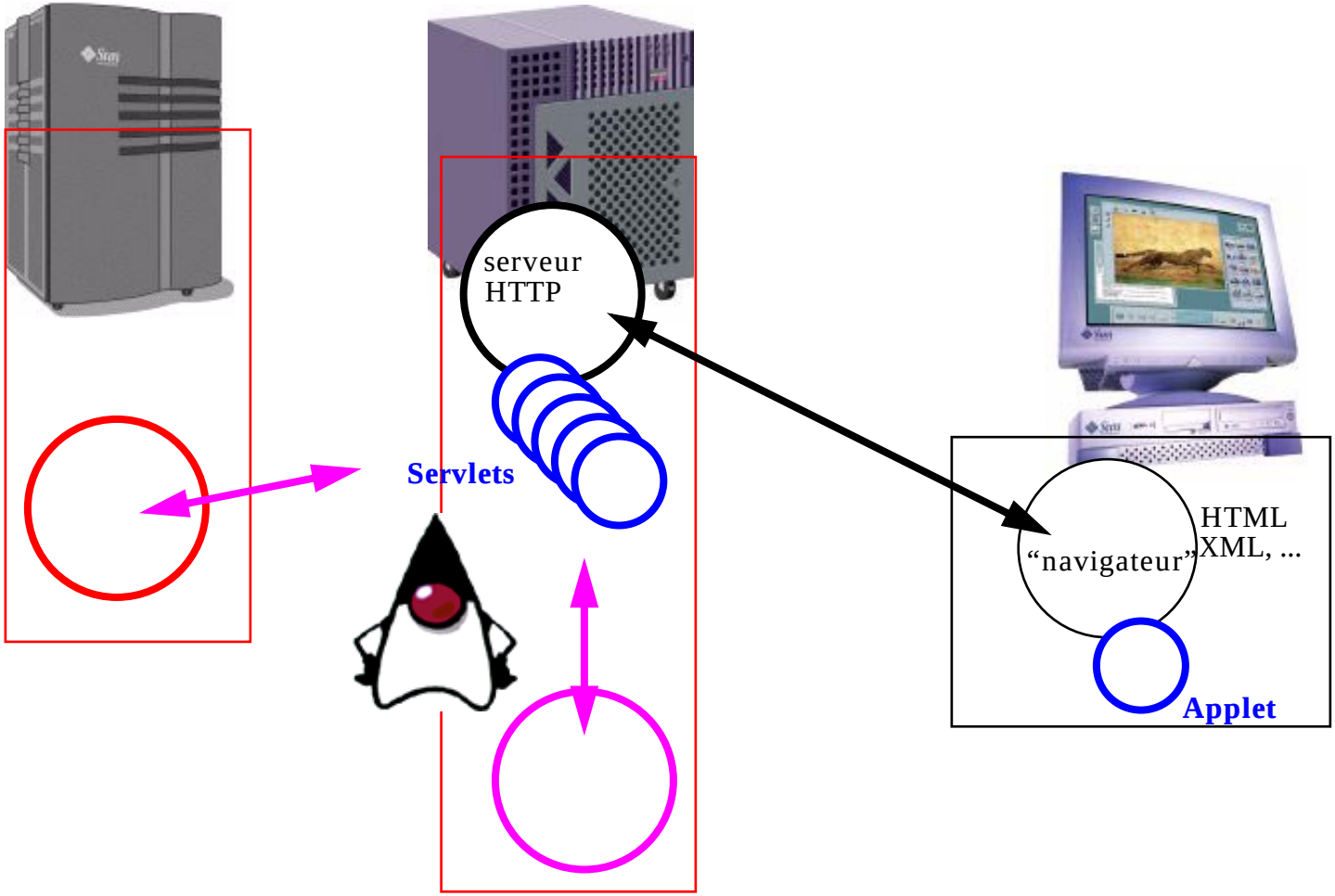
JavaCard,....

Java dans le système d'information

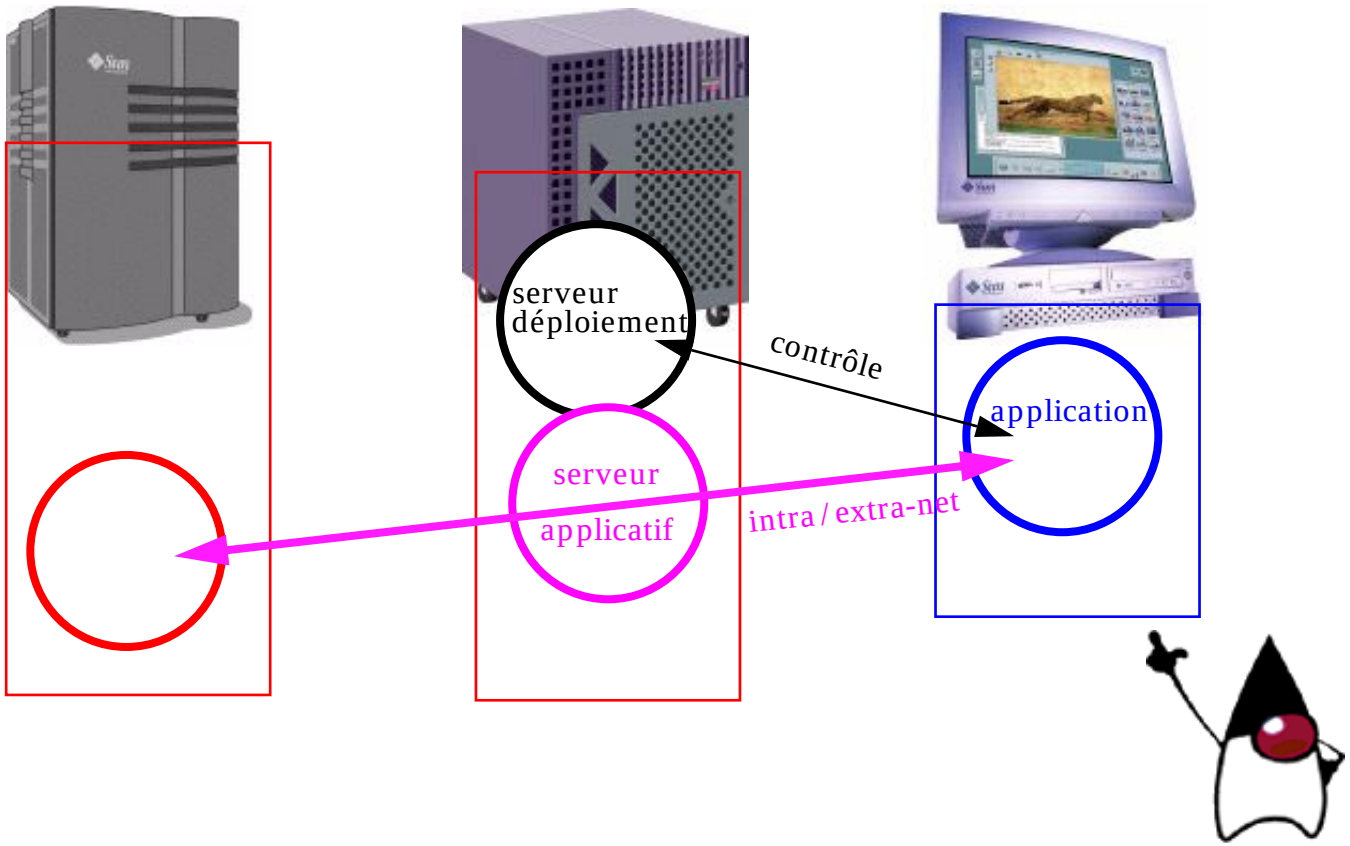
A.Faucillon

de nouvelles applications pour de nouvelles architectures

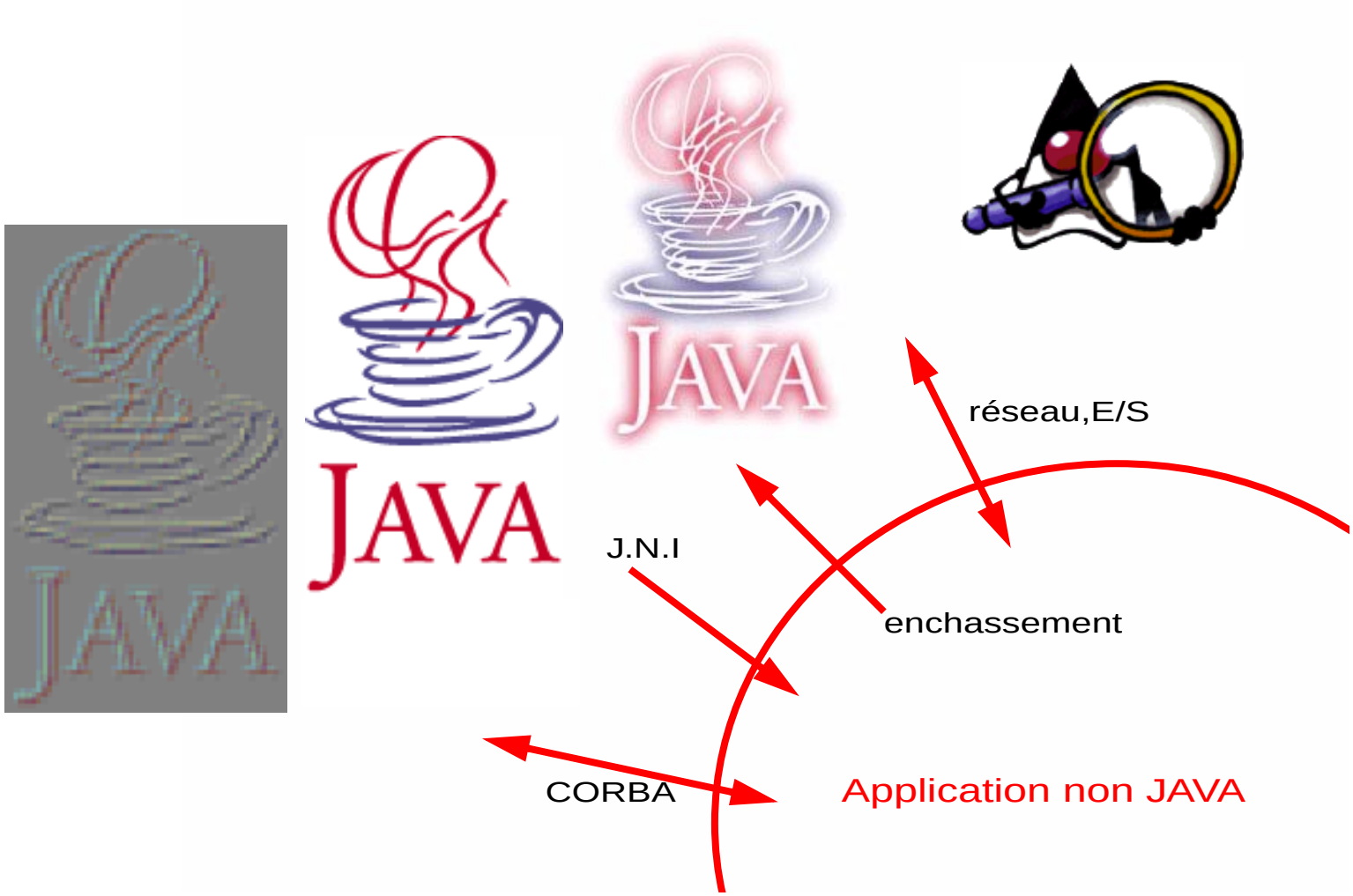
client léger sur Web



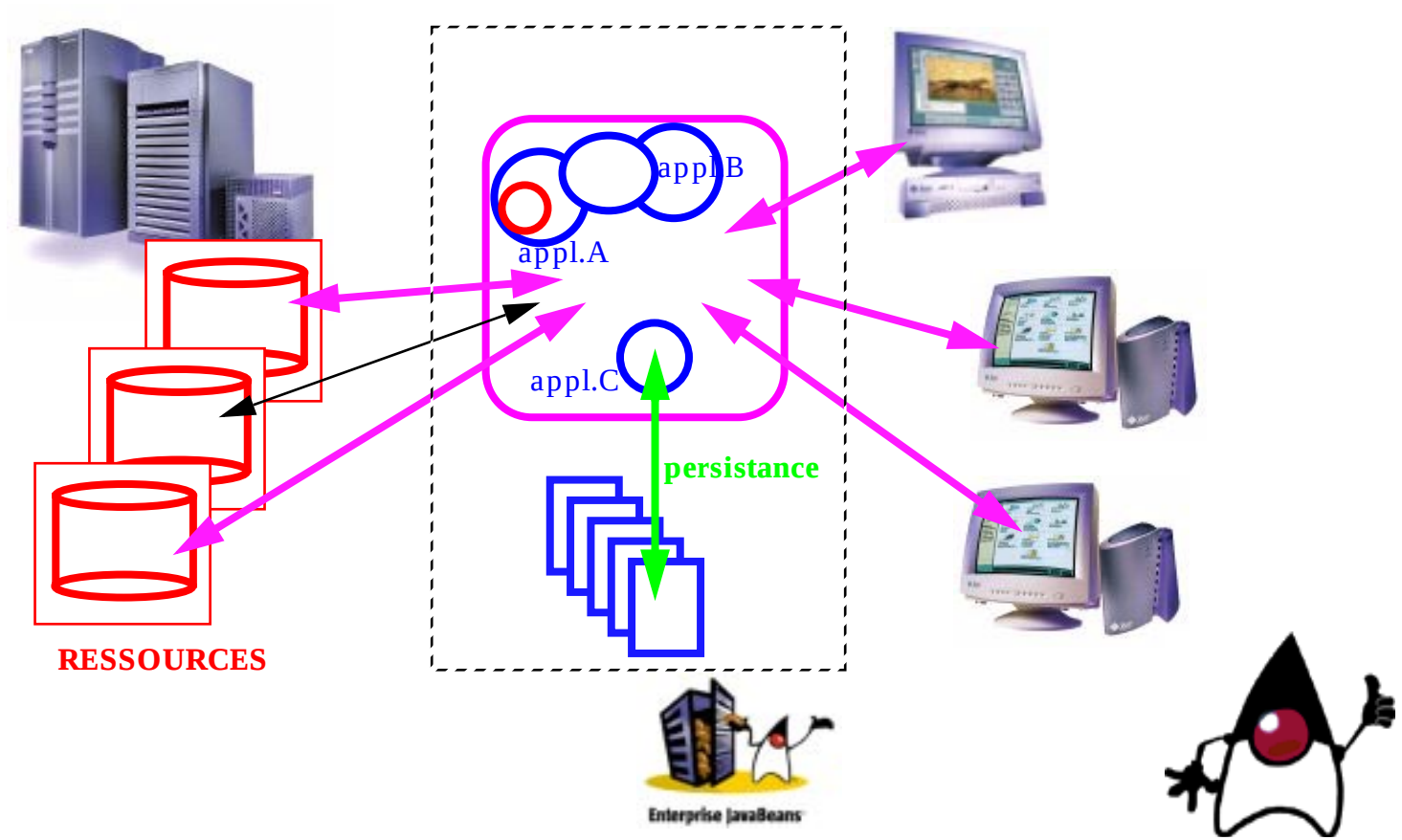
client “lourd”



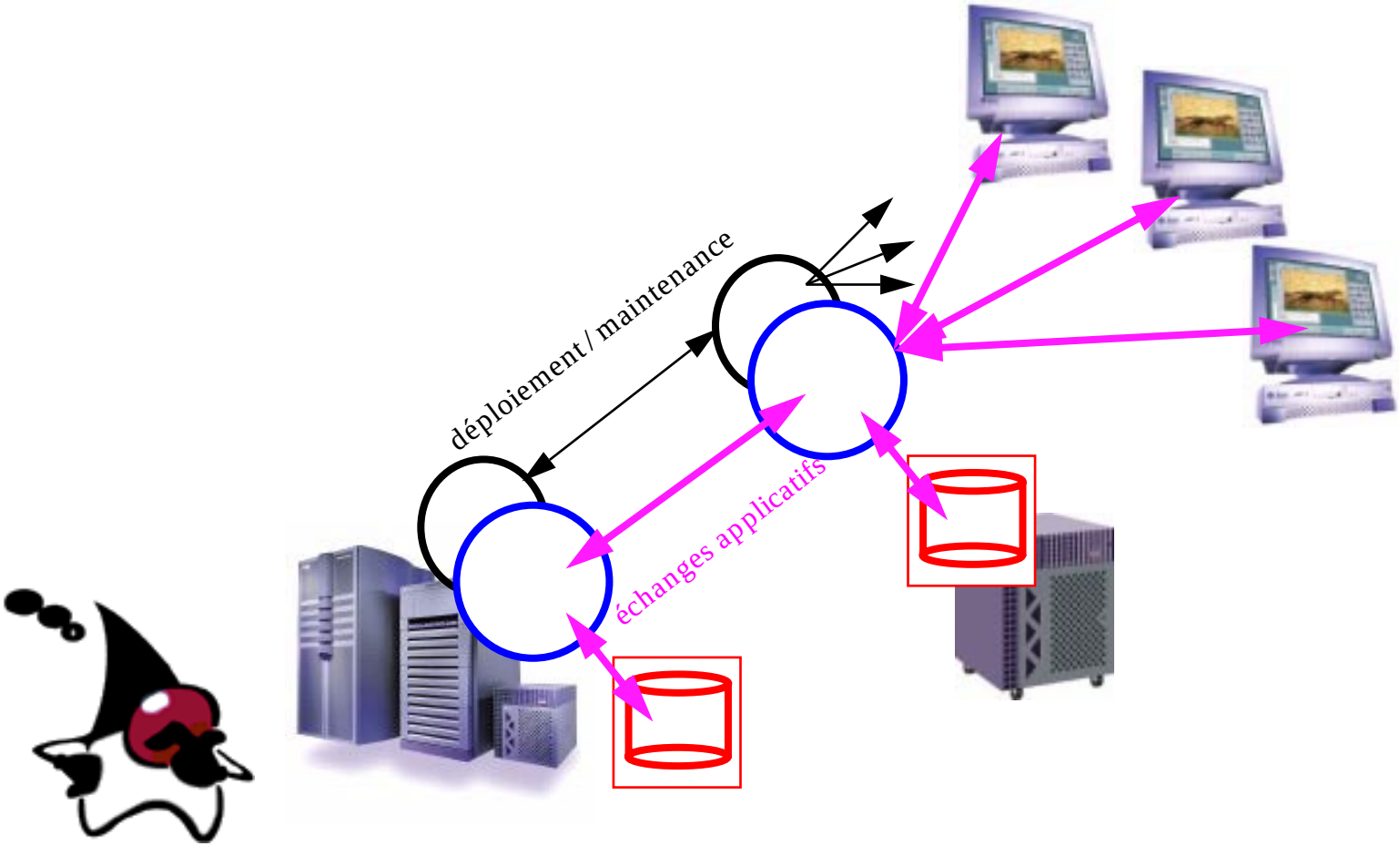
nota: Java comme langage d'extension



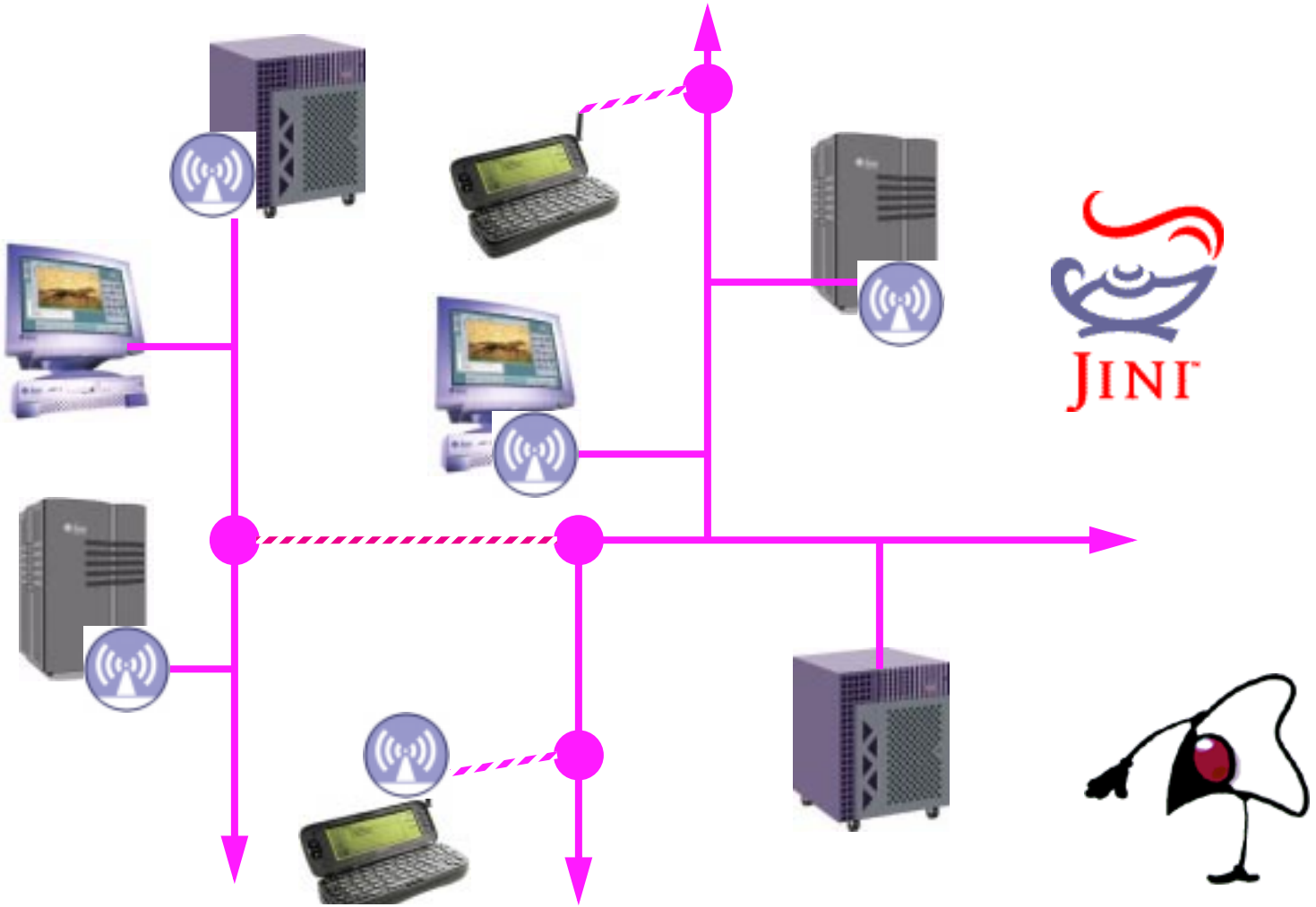
serveurs d'application



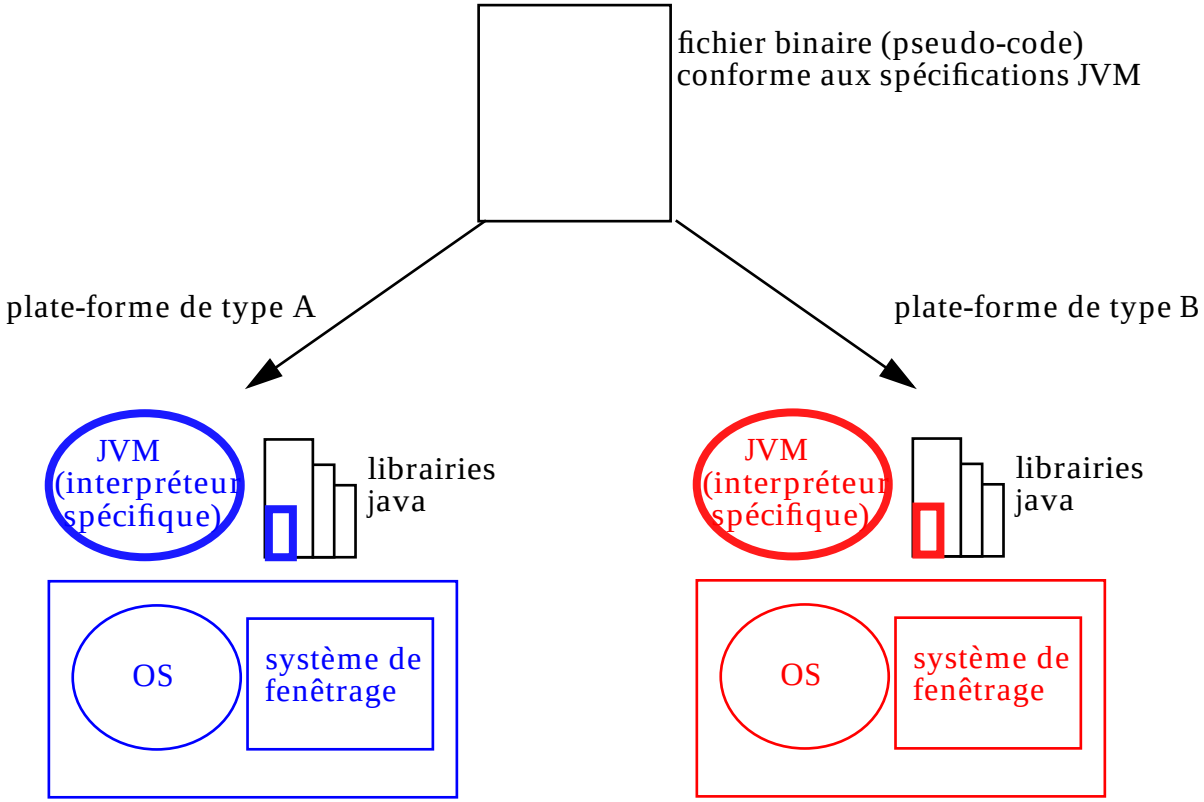
serveurs hiérarchisés



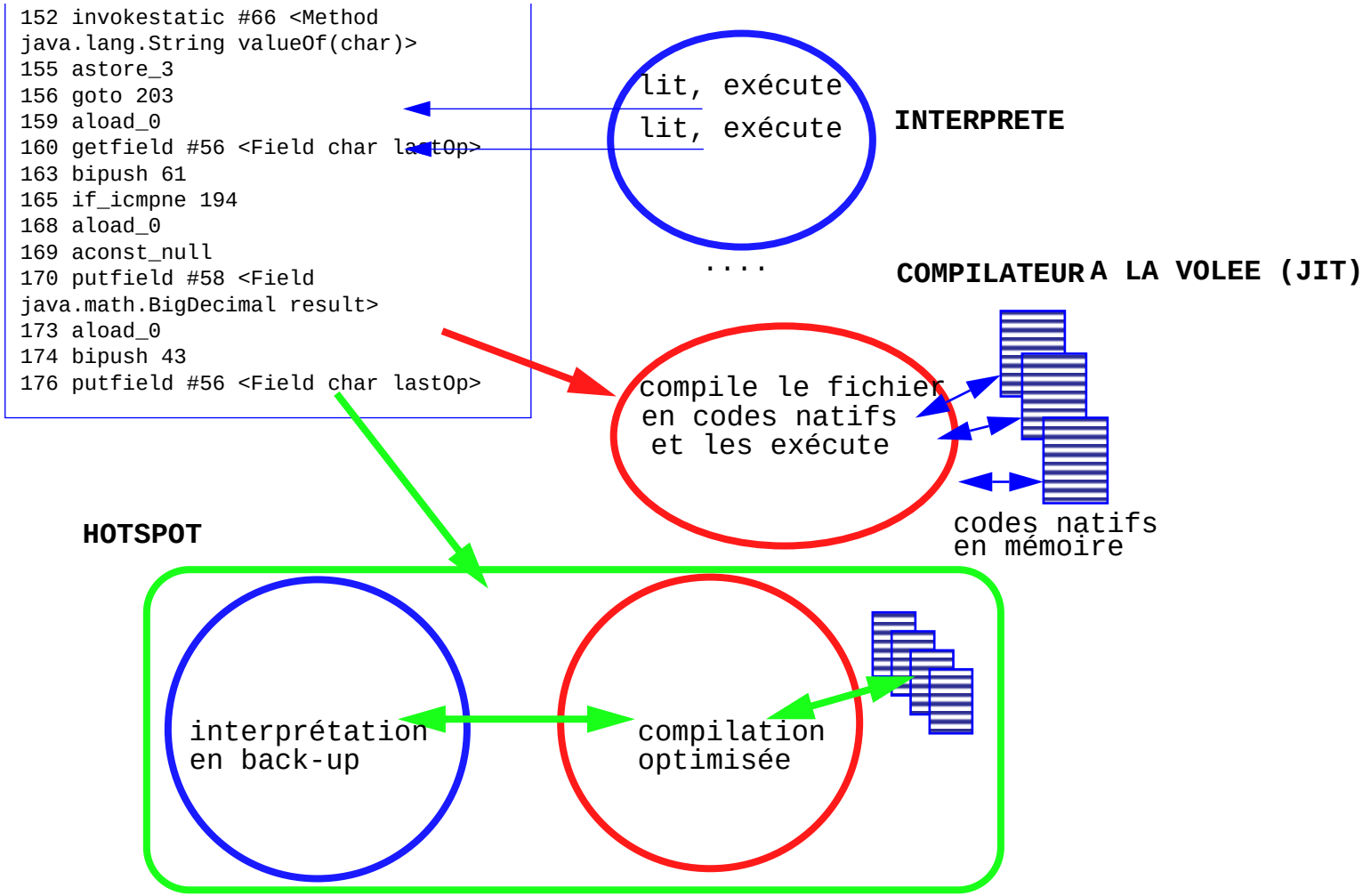
services “spontanés”



Comment ça marche ?



techniques d'exécution



sécurité

- chargeurs de classes
- vérificateur de pseudo-code
- espaces d'exécution sécurisés :
  * politique par défaut ("sandbox")
  * politiques personnalisées
    (codes authentifiés
    + autorisations spécifiques)

dans la soute ....

- gestion automatique de la mémoire
  (glaneur de mémoire
  + réorganisation dynamique du tas)
- gestion de processus légers
  (ordonnanceur intégré
  ou *threads* natifs du système)

L’Objet selon JAVA

- ça ressemble à du C++
- ça n’a pas le “goût” du C++
- ce n’est absolument pas du C++ !

La définition de classe

```
import fr.emse.medialab.util.* ;

public class UneClasse extends SuperClasse {
        //membres
        public final String oid ;
        private Chose obj ;.....
        public void setChose(Chose arg) {// mutateur
           obj=arg ;
        }
        public static class Comparator extends
                        Chose.Comparator {// classe membre!
            int compare(Object o1, Object o2){
              return super.compare( ((Unclasse)o1).obj,
                          ((UneClasse)o2).obj) ;
            }
        } // pas de surcharge d'opérateur
        // non membres
       public UneClasse(Truc truc, Chose obj){//constructeur
           super(truc);
          oid = truc.toString() ;......
       }//pas de destructeur (ou presque)
}
// RIEN EN DEHORS DE LA CLASSE!
```

rien hors de la classe

- pas de variables ou de fonctions “globales”

```
double ix = Math.sin( iy * Math.PI) ;
double iz = StrictMath.sin(valeur) ;
```

- pas de fichier “include”

  les contrôles de types fonctionnent par introspection du binaire des classes

encapsulation

Les classes sont regroupées dans des “packages”

`fr.emse.medialab.graphics`

`edu.mit.medialab.graphics`

L’encapsulation se considère:

- au niveau de la classe (`private`),
- au niveau du package (par défaut),
- au niveau d’une relation d’héritage (`protected`)

types

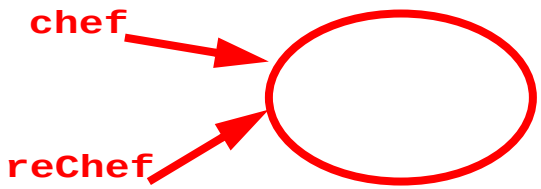
type variable != classe de l' instance!

- aspect conceptuel

```
Manager leSousChef; // extends Salarie ....
Manager lePatron = new PDG(...); // extends Manager
Salarie[] employés = { leLampiste, leSousChef, lePatron}
```

- aspect "physique"

```
PDG chef = new PDG(..);
Manager reChef = chef ;
// tout est "référence"
```



- aspect évaluation (méthodes d'instance virtuelles)

polymorphisme: héritage

- on n'hérite que d'une seule classe
- on ne peut aggraver le "contrat"

  * pas d'encapsulation aggravée

  * pas "plus" d'exception

  le type des exceptions propagées fait partie de la spécification des méthodes

  * par contre possibilité de surcharger des méthodes de la super-classe
- on peut bloquer toute tentative de redéfinition (modificateur `final`)

types abstraits purs

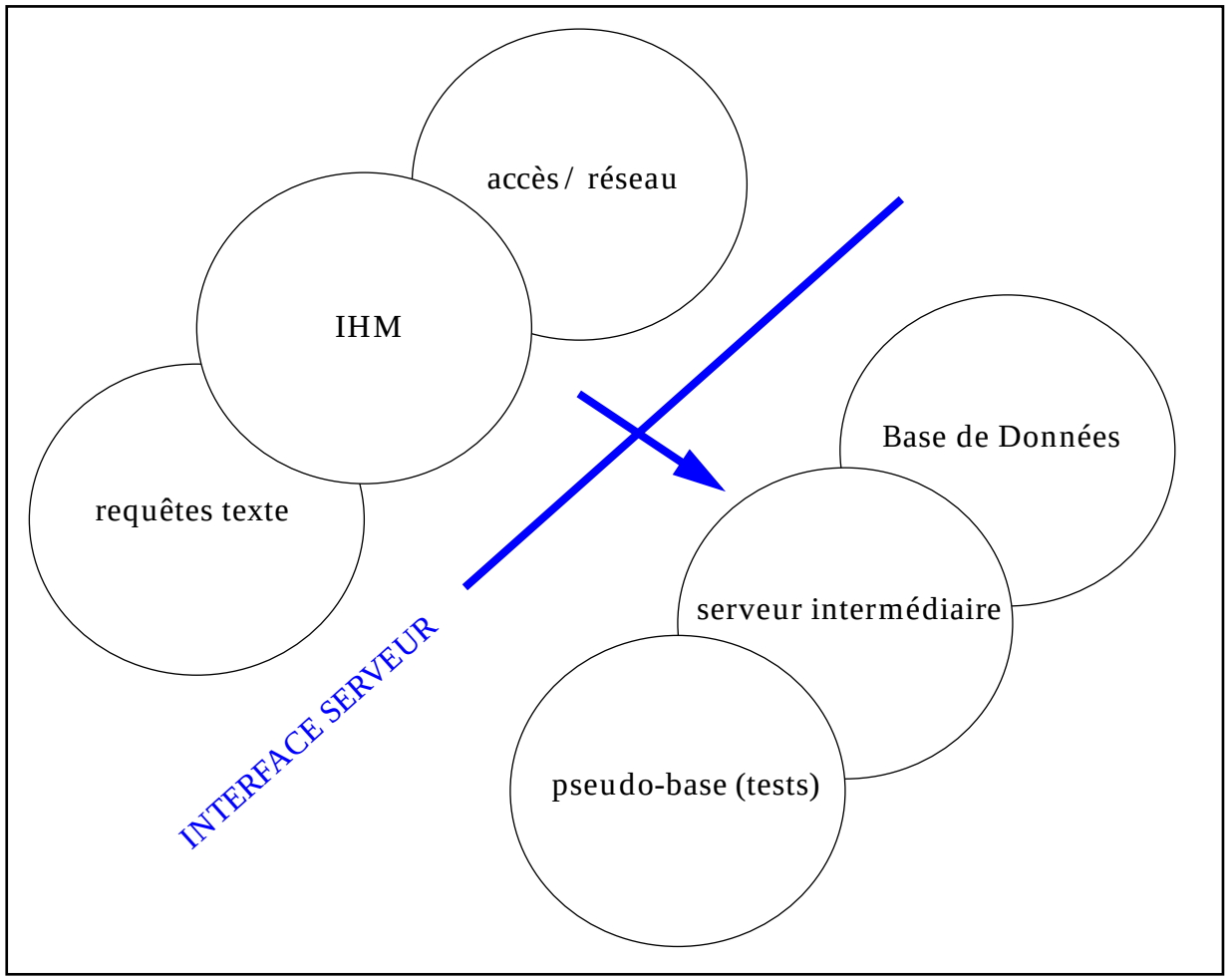
```
public class Client {
          .....
          Messager messager ;
          .....
```

```
public interface Messager {
          public void envoiMessage(String message);
}
```

```
public class Fax extends Telephone implements Messager {
          public void envoiMessage(String message) {
          ...
```

```
public class EMail extends AgentReseau implements Messager {
          public void envoiMessage(String message) {
          ...
```

```
          Client dupond, durand ;
          ....
          dupond.messager = new Fax("0141331733") ;
          durand.messager = new EMail("durand@schtroumpf.fr") ;
          ....
          for (int ct= 0 ; ct < tableauClient.length; ct++){
                    tableauClient[ct].messager.envoiMessage(
                              "Tout va bien!");
          }
```


accès / réseau
IHM
requêtes texte
Base de Données
serveur intermédiaire
pseudo-base (tests)
INTERFACE SERVEUR

Classes membres

```
public class Pile {
        private Object[] tableExtensible ;
        private int sommetDePile ;
        ...
        private class ParcoursPile
                  implements java.util.Enumeration {
                  int index = sommetDePile ;
                  public boolean hasMoreElements(){
                   return index >= 0;
                  }
                  public Object nextElement(){
                   return tableExtensible[index--];
                  }
        }// fin parcoursPile

        public java.util.Enumeration elements() {
         return new ParcoursPile();
        }
        ...
}
```

Autres caractéristiques

- gestion exceptions
- processus et concurrence d’accès
- prise en compte de la répartition sur réseau
- documentation intégrée

des savoir-faires en devenir

- idiomes, “patterns”, bonnes pratiques
- relations conception/réalisation