

[www.Mcours.com](http://www.Mcours.com)  
Site N°1 des Cours et Exercices Email: [contact@mcours.com](mailto:contact@mcours.com)

# Développement PHP

Scripting (pour tous)

Objet (un peu)

[jd@ecomms.fr](mailto:jd@ecomms.fr)

Version du 07/11/2007

# Rapide historique du langage

- Créé en 1995 par Rasmus Lerdorf
  - Au départ, simples outils pour dynamiser sa page personnelle
- Rapide évolution du langage devenu très populaire
  - PHP3 : première version “universelle”
  - PHP4 et PHP5 : concepts objets dans PHP

# Bases du langage

- Langage interprété par un module à inclure dans Apache (ou IIS)
- Code entre balises `<?php et ?>`
- Syntaxe proche du langage C
- Langage non typé : pas de int, String...
- Philosophie générale : aller au plus simple

# Les balises PHP

- Balise standard : `<?php .....?>`
- Peut se décliner sur plusieurs lignes
- Version raccourcie (non conseillée) :  
`<? ....?>`
- Affichage de contenu :  
`<?=....?>`
  - Affichage d'une variable, ou d'un retour de fonction

# Variables en PHP

- Les déclarations se font au fur et à mesure de l'utilisation des variables

```
$nom="Steve";
```

```
$age=47;
```

```
$ttc=$ht+$tva;
```

- Toute concaténation s'effectue avec le symbole .

```
$date=$jour."/".$mois."/".$an;
```

# Affichage

- Toutes les instructions PHP ne provoqueront aucun affichage sur la page HTML finale
- Sauf :
  - La balise affichant le contenu d'une variable  
`<?=$i ?>`
  - La commande echo  
`<?php echo $i; ?>`

# Traitement d'une page PHP par le serveur

# Traitement d'une page PHP par le serveur

Fichier sur  
le serveur

```
<?php $nom="Durand"; ?>  
<html>  
<body>  
<p>  
Bonjour, Monsieur <?=$nom ?>  
</p>  
</body>  
</html>
```

# Traitement d'une page PHP par le serveur

Affichage  
sur le browser

```
<html>
<body>
<p>
Bonjour, Monsieur Durand
</p>
</body>
</html>
```

Fichier sur  
le serveur

Serveur

```
<?php $nom="Durand";?>
<html>
<body>
<p>
Bonjour, Monsieur <?=$nom ?>
</p>
</body>
</html>
```

# Traitement d'une page PHP par le serveur

Affichage sur le browser

Le PHP a été interprété et ne laisse pas de traces

```
<html>
<body>
<p>
Bonjour, Monsieur Durand
</p>
</body>
</html>
```

Fichier sur le serveur

Serveur

```
<?php $nom="Durand";?>
<html>
<body>
<p>
Bonjour, Monsieur <?=$nom ?>
</p>
</body>
</html>
```

# Traitement d'une page PHP par le serveur

Affichage sur le browser

Le PHP a été interprété et ne laisse pas de traces

```
<html>
<body>
<p>
Bonjour, Monsieur Durand
</p>
</body>
</html>
```

Fichier sur le serveur

```
<?php $nom="Durand";?>
<html>
<body>
<p>
Bonjour, Monsieur <?=$nom ?>
</p>
</body>
</html>
```

Serveur

La variable a été affichée

# Mélange HTML / PHP

```
<html>
```

```
<body>
```

```
<? (for $i=0;$i<10;$i++) { ?>
```

```
La valeur courante est : <?=$i ?><br />
```

```
<? } ?>
```

```
</body>
```

```
</html>
```

# Affichage mixte

- Précéder toutes les variables d'un '\$' permet à l'interpréteur de les repérer

- La concaténation n'est donc pas obligatoire

*echo "Bonjour \$prenom \$nom, comment vas tu?";*

*echo "select \* from User where id='\$id'";*

- L'utilisation de ' est réservée à l'écriture de balises HTML, gourmandes en " :

*echo '<input type="text" name="prenom" />';*

# Tests

- L'indémoudable if...else est bien entendu au programme

```
if($age >= 18)
    { echo "Hi there!"; }
else
    { echo "Bye"; }
```

# Boucles

[www.Mcours.com](http://www.Mcours.com)

Site N°1 des Cours et Exercices Email: [contact@mcours.com](mailto:contact@mcours.com)

```
for($i=0;$i<10;$i++)  
{...}
```

```
while($nom=="Steve")  
{...}
```

# Fonctions

```
function calcul TVA($ht)  
{  
    return $ht*0.196;  
}
```

...

```
$taxe=calcul TVA(123.45);
```

# Fragmentation des applications

- Il est fortement conseillé de stocker ses fonctions dans des “bibliothèques”
  - De simples fichiers .php qui ne feront pas d’affichage
- Ces bibliothèques seront incluses lors de l’exécutions

```
include(“mesfonctions.php”);
```

```
require_once(“mesfonctions.php”);
```

require  
génère une erreur  
fatale, pas include

# PHP et objet

- Concepts de bases présents dès PHP3
- Implémentation solide à partir de PHP5
  - Mais peu d'hébergeurs offrent cette possibilité
  - La plupart des sites PHP "objet" sont donc en PHP4

# Carences de l'objet en PHP4

- Nombreux concepts manquants
  - Polymorphisme fragile
  - Pas d'encapsulation (private, public...)
- Mais permet toutefois une meilleure structuration que les simples fonctions
- PHP5 n'est pas parfait
  - Encapsulation médiocre
  - Pas de surcharge de méthodes

# Ecriture d'une classe

```
<?php
class Client
{
    private $nom; //attribut
    private $prenom;
    public function __construct($nom,$prenom) // constructeur
    {
        $this->nom=$nom;
        $this->prenom=$prenom;
    }
    public function arrivee() { ... } // méthode
    public function depart($heure) { ... }
}
?>
```

# Appel d'une classe

<?

```
require_once("class_Client.php");
```

```
$cli=new Client("Capone","Al");
```

```
$cli->arrivee();
```

```
$cli->depart("11h");
```

?>

Instanciacion

Appel de  
méthode

# Affichage d'une instance

`echo $instance;`

- provoquera l'affichage d'une adresse mémoire

`print_r($instance)`

- affichage de la structure complète
- si la classe contient une méthode...

`function __toString()`

- ...cette méthode est appelée

# Autochargement

- Mécanisme standard permettant de palier aux oublis d'include ou require

```
<?php
function __autoload($class_name)
{
    require_once 'class_'. $class_name. '.php';
}

$obj = new MaClasse1();
$obj2 = new MaClasse2();
?>
```

# Structuration avec classes

Permet de reprendre le schéma 3-tiers :

1. Frontal Web avec une page HTML et un minimum de code PHP
  - Idéal pour les Web designers
2. Couche métier avec des classes PHP
  - Issu du travail des analystes programmeurs
3. Couche persistance avec la base de données

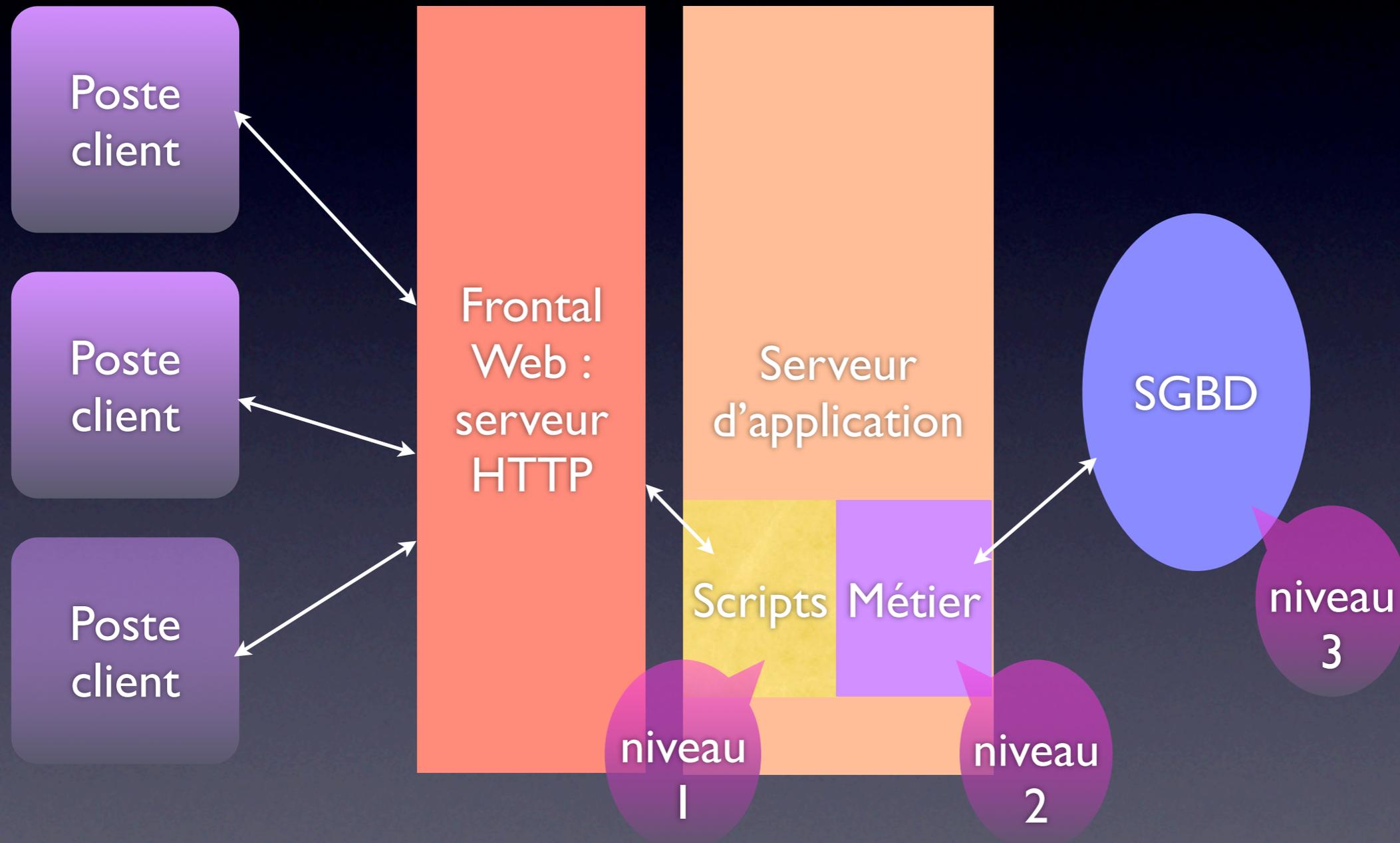
# Architecture 3-tiers

[www.Mcours.com](http://www.Mcours.com)

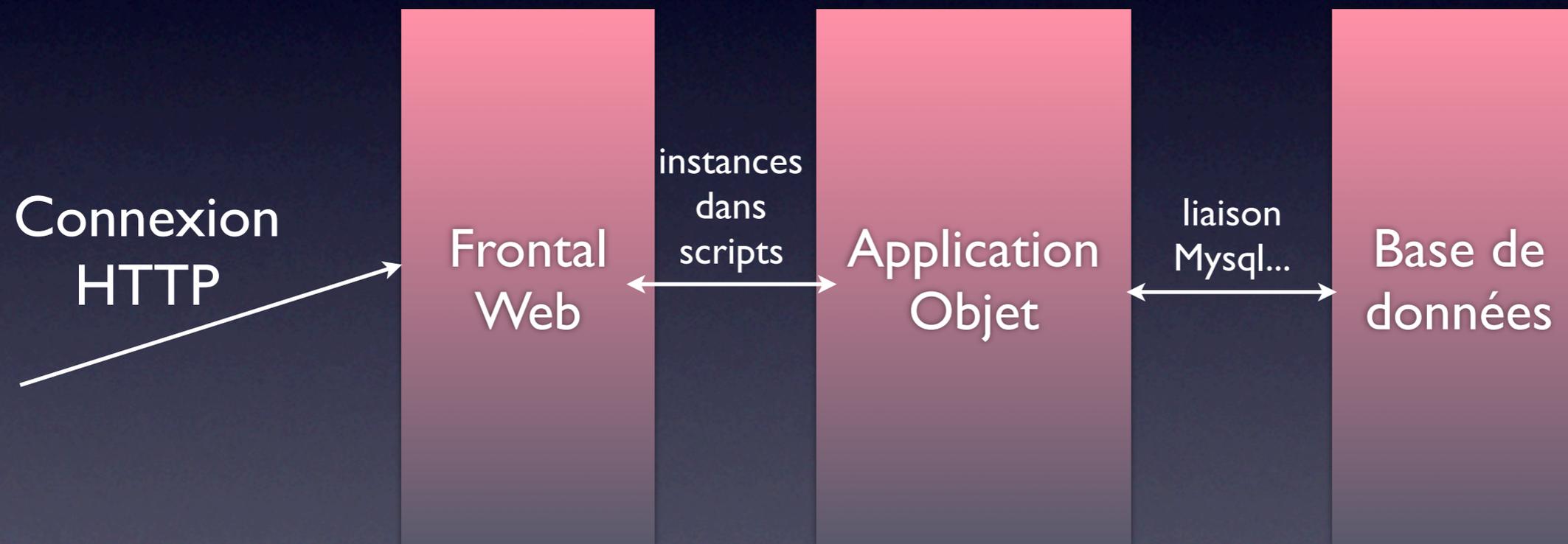
Site N°1 des Cours et Exercices Email: [contact@mcours.com](mailto:contact@mcours.com)

- Un frontal : la page HTML façon scripting
- Une partie métier : l'application elle même, objet ou autre
- Une couche de persistance : la base de données et son interfaçage applicatif

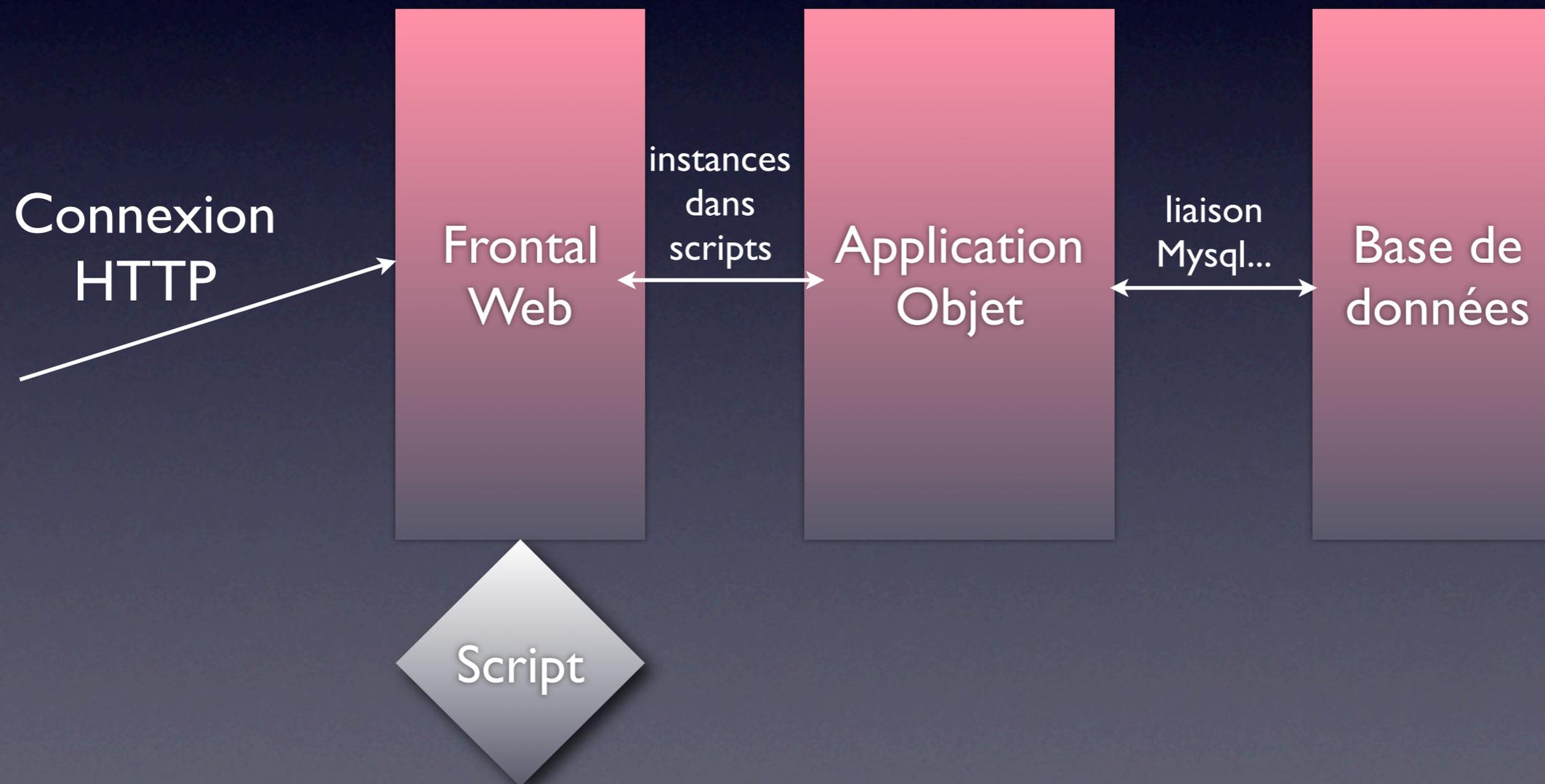
# Structure 3-tiers



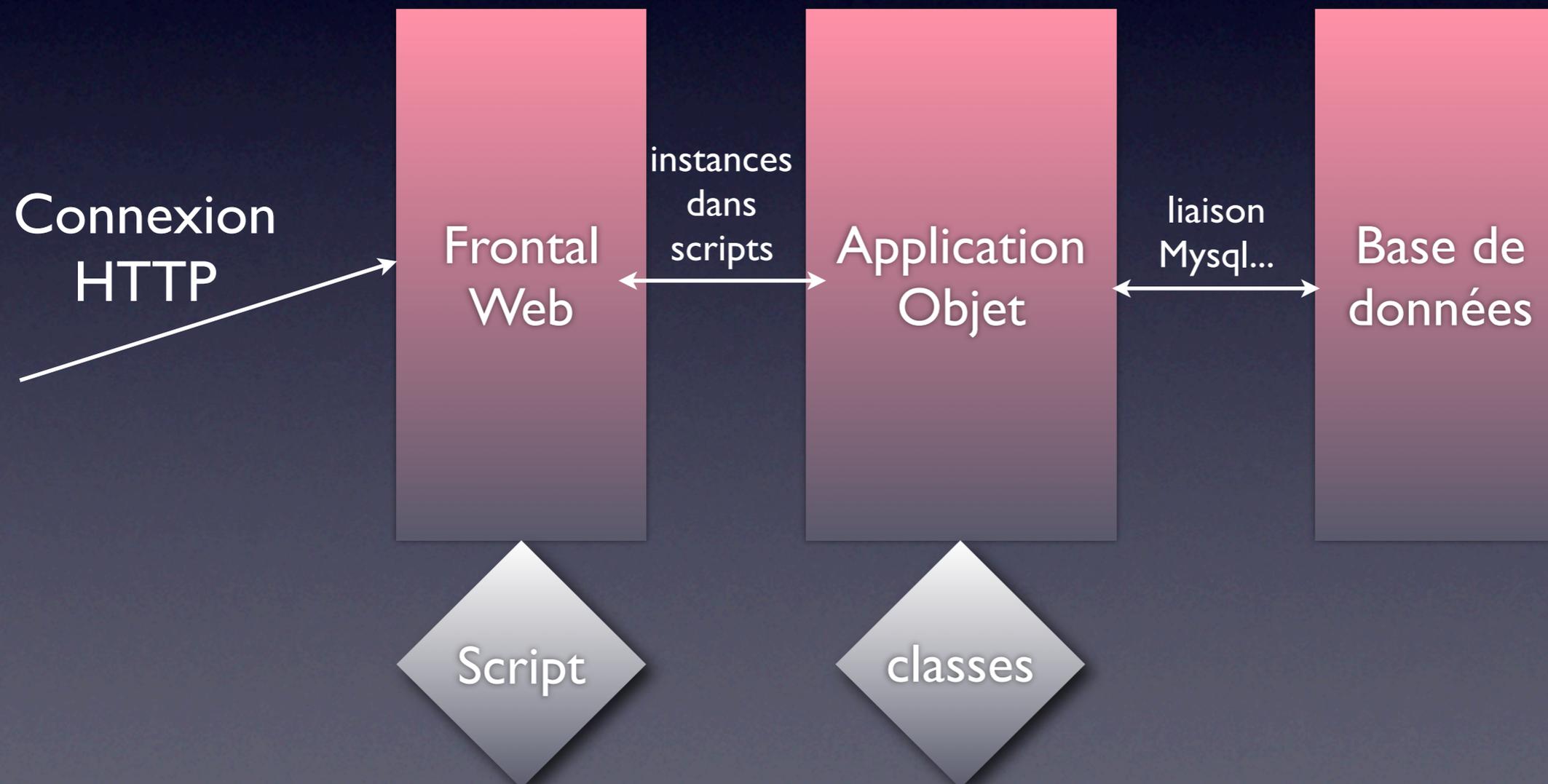
# Conception 3-tiers



# Conception 3-tiers



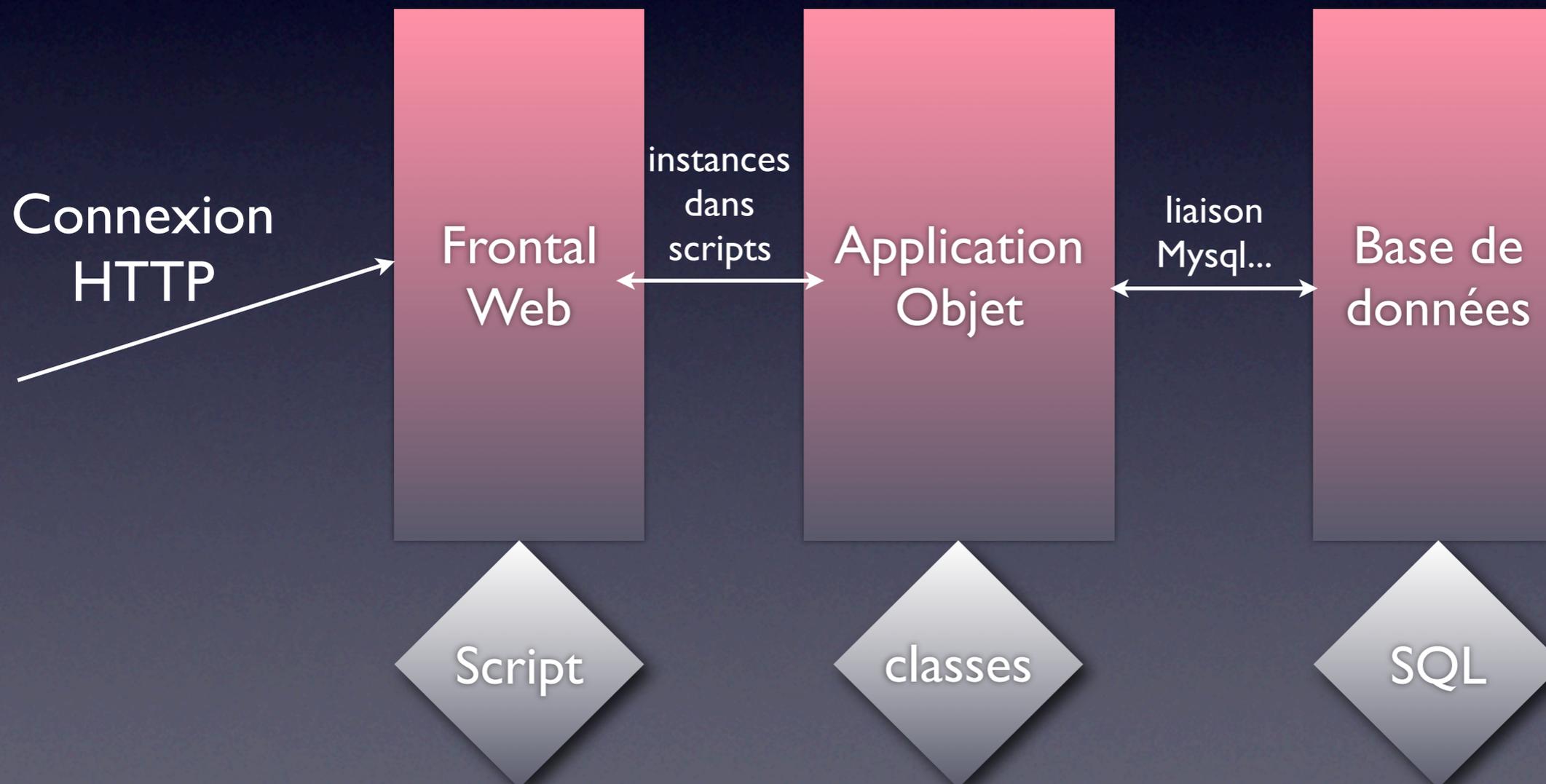
# Conception 3-tiers



# Conception 3-tiers

[www.Mcours.com](http://www.Mcours.com)

Site N°1 des Cours et Exercices Email: [contact@mcours.com](mailto:contact@mcours.com)



# Conception 3-tiers

A partir  
d'une maquette  
HTML

Connexion  
HTTP

Frontal  
Web

instances  
dans  
scripts

Application  
Objet

liaison  
Mysql...

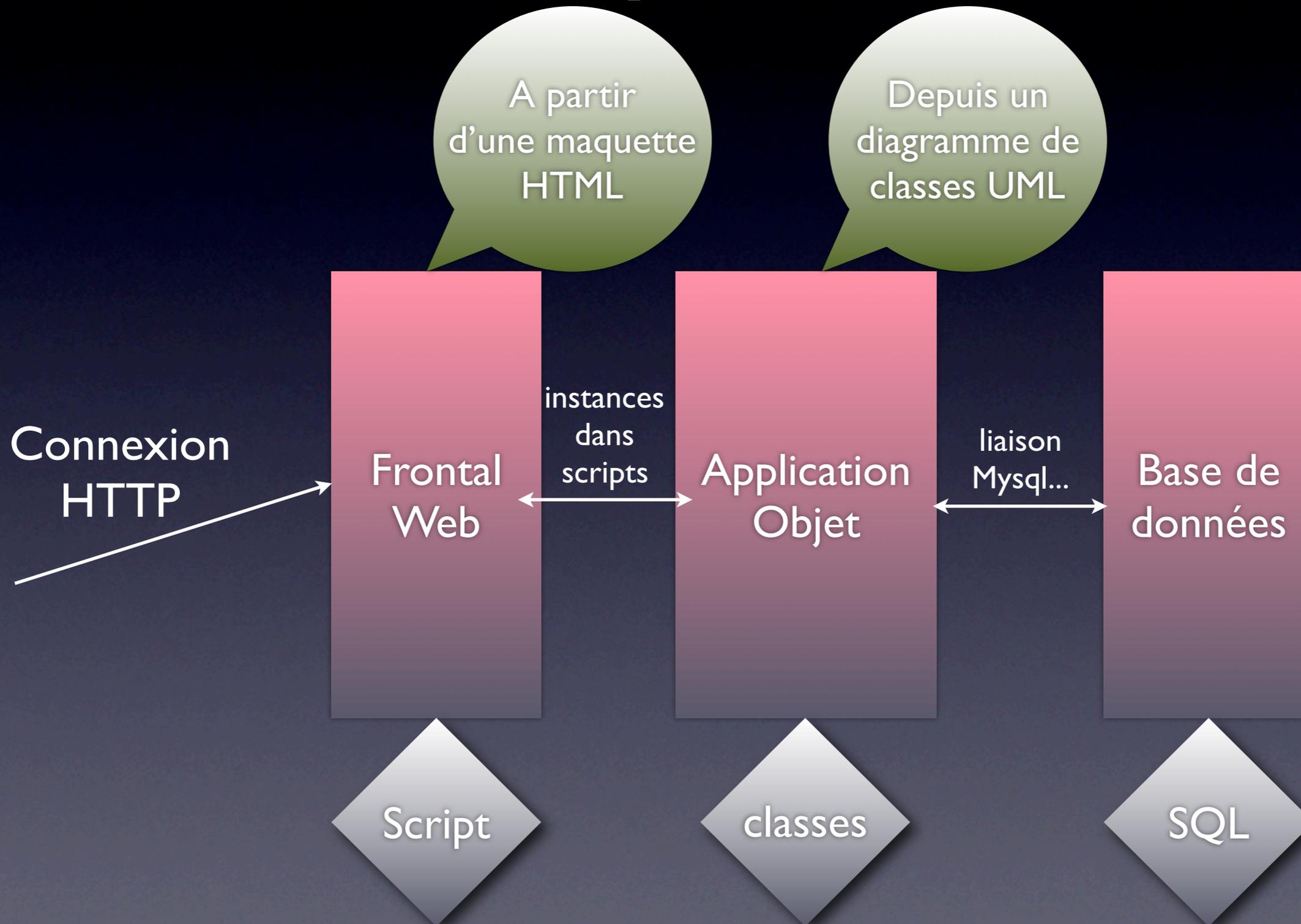
Base de  
données

Script

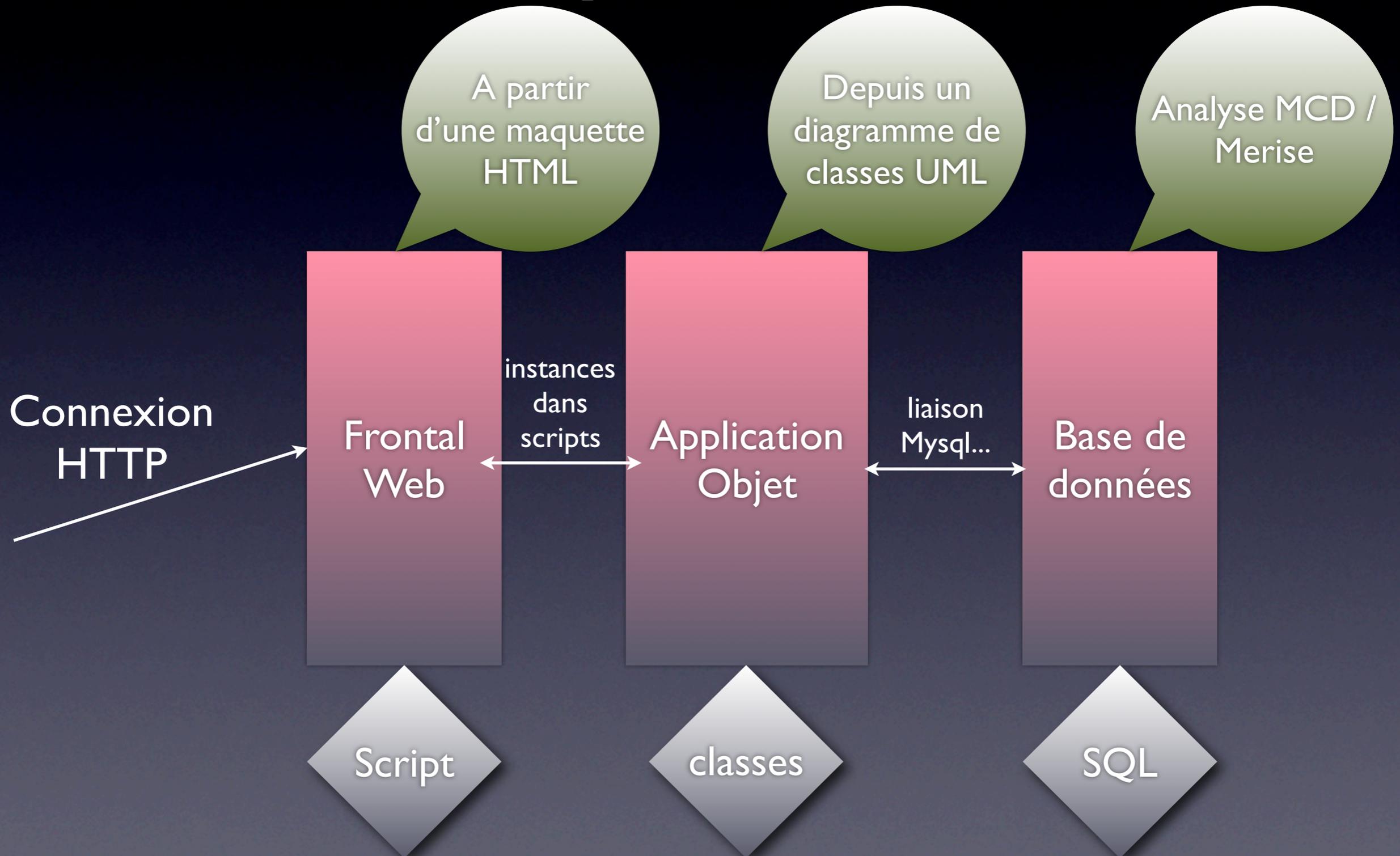
classes

SQL

# Conception 3-tiers



# Conception 3-tiers



# Portée des membres

- Les visibilités classiques existent :
  - *public* : membre accessible depuis l'extérieur
  - *private* : membre accessible uniquement à l'intérieur de la classe
  - *protected* : extension de la visibilité 'private' aux classes filles

# Constructeurs

- En PHP5, on utilise la syntaxe :

```
function __construct()
```

- La syntaxe de PHP4 avec le nom de classe subsiste
-  La surcharge n'existe pas en PHP : pas de constructeurs multiples à paramètres différents
- Il existe aussi une notion de destructeur

```
function __destruct()
```

# Staticité

- Certains membres peuvent être statiques
  - Ils deviennent accessibles sans instance
  - Ils sont communs à toutes les instances
  - Accessibles via l'opérateur ::
- Précieux en PHP, car fournissent un espace mémoire indépendant des pages

# Le retour des fonctions ?

- Une méthode statique est finalement très proche d'une fonction
  - En effet, une méthode est une fonction interagissant avec ses attributs de l'instance
  - Or, une méthode statique ne peut pas interagir avec les attributs (sauf attributs statiques)
- Seule 'petite' différence : n'est pas globale et peut être camouflée (méthode statique privée)

# Exemple de staticité

```
class UneClasse
{
    public static $elem_static = 'coucou';

    public function getElemStatic()
    {
        return self::$elem_static;
    }
}

echo UneClasse::$elem_static . "\n";
```

**self::**

est utilisé pour mentionner un membre statique de sa propre classe

Syntaxe

*Classe::elemStatique*

On ne peut pas pointer sur un élément statique à partir d'un objet

# Classe parente et ::

- parent:: sert à mentionner un des éléments de la classe parente
- Ex :

```
public __construct()  
{  
    parent::__construct();  
}
```

# Constantes de classe

[www.Mcours.com](http://www.Mcours.com)

Site N°1 des Cours et Exercices Email: [contact@mcours.com](mailto:contact@mcours.com)

- Dans certains cas, il peut être utile de stocker des constantes au sein d'une classe
- On n'utilise pas le symbole '\$'
- Attention, ces constantes sont uniquement accessibles à l'intérieur même de la classe

```
const tva=0.196;
```

```
...
```

```
$prixttc=$prixht*(1+self::tva);
```

# Implémentation d'une liaison 0..1 par composition

```
class Client
{
    private $panier; // instance de la classe Panier
    function __construct()
    {
        // instantiation dans le constructeur :
        // il s'agit d'une composition entre Client et Panier
        $this->panier=new Panier();
    }
}
```

# Implémentation d'une liaison 0..1 par agrégation

```
class Client
{
    private $panier; // instance de la classe Panier
    function __construct($panier)
    {
        // affectation d'une instance externe :
        // il s'agit d'une agrégation entre Client et Panier
        $this->panier=$panier;
    }
}
```

# Implémentation d'une association

```
class Peage
{
  fonction passeVoiture($voiture)
  {
    // a un moment donné, on fournit à l'instance
    // de péage une instance de voiture
    ...
  }
}
```

# Implémentation d'une liaison 0..n

```
class Groupe
{
    private $etudiants; // futur tableau dynamique
    function ajoutEtudiant($etudiant)
    {
        $this->etudiants[]=$etudiant; // ajout instance à liste
    }
    function getEtudiants() // accesseur du tableau
    {
        return $this->etudiants;
    }
}
```

# Parcours de la liste

```
$etudiants=$classe->getEtudiants();  
foreach($etudiants as $etudiant)  
{  
    $sum+=$etudiant->getNote();  
}  
$moy=$sum/sizeof($etudiants);
```

# Héritage

```
class Forme
{
  private $trucprive;
  protected $ref;
  public $titre;
  public final
  function method1()
  {
    ...
  }
  public
  function method2()
  {
    ...
  }
}
```

Seul l'héritage simple est autorisé

```
class Cercle extends Forme
```

```
{
  public function method1() {...} //
  public function method2() {...}
  public function method3()
  {
    parent::method1();
  }
}
```

Surcharge de méthode **interdit**

Référence à la classe mère

# Notion d'abstraction

- Créer une méthode abstraite implique :
  - La classe sera également abstraite
    - On ne pourra l'instancier
  - obligation d'implémentation de la méthode dans les classes filles
  - On pourra mettre en place un traitement polymorphe

# Classes abstraites

```
abstract class Forme
```

```
{
```

```
  abstract protected function calculeAire();
```

```
  public function affiche()
```

```
{
```

```
  ...
```

```
}
```

```
}
```

```
class Cercle extends Forme
```

```
{
```

```
  public function calculeAire()
```

```
  {...}
```

```
  public function autreMethode()
```

```
  {...}
```

```
}
```

Obligation  
d'implémentation

# Interface

- Une interface est une classe abstraite qui ne contient aucune implémentation

interface Mesurable

{

public function uniteMesure();

}

class Fruit **implements** Mesurable

{

private \$poids;

public function uniteMesure()

{

return \$this->poids;

}

}

Obligation  
d'implémentation

# Traitement polymorphe

[www.Mcours.com](http://www.Mcours.com)

Site N°1 des Cours et Exercices Email: [contact@mcours.com](mailto:contact@mcours.com)

- Un traitement polymorphe va permettre de traiter une liste d'éléments avec un traitement générique

```
foreach($listeMesurables as $mesurable)
{
    $somme+=$mesurable->uniteMesure();
}
```

# Attributs et méthodes inexistantes

- Un mécanisme en PHP permet “d’attraper” les appels à des méthodes ou attributs n’existant pas
  - Afin d’appliquer un traitement alternatif
  - Peut éventuellement servir à simuler la vraie surcharge de méthode objet

# Attribut inexistant

- On utilise les méthodes `__get()` et `__set()`

```
class UneClasse
```

```
{  
    public function __get($nomattr)  
    {  
        if($nomattr=="bla")  
            return "bli";  
        else  
            return "rien";  
    }  
    public function __set($nomattr)  
    { ... }  
}
```

```
$inst=new UneClasse();  
echo $inst->bla;  
echo $inst->toto;
```

# Méthode inexistante

- On utilise la méthode `__call()`

```
class UneClasse
```

```
{
```

```
public function __call($meth,$p)
```

```
{
```

```
if($meth=="setColor")
```

```
if(sizeof($p)==1
```

```
$this->setColor1($p[0])
```

```
if(sizeof($p)==3
```

```
$this->setColor3($p[0],$p[1],$p[2])
```

```
}
```

```
}
```

```
$inst=new UneClasse();
```

```
$inst->setColor("rouge");
```

```
$inst->setColor(0,0,0);
```

# Clonage d'instance de classe

- Une affectation fonctionne par référence  
`$inst2=$inst1;`
- `$inst1` et `$inst2` “pointent” sur la même instance
- Il est possible de recopier une instance, afin qu'elle gagne sa totale indépendance :  
`$inst2= clone $inst1;`
- La méthode `__clone()` de l'instance est appelée

# Comparaison entre instances

- La comparaison classique permet de tester l'égalité attribut par attribut

`if($inst1 == $inst2) ...`

- Il est possible de comparer si deux variables pointent vers la même instance

`if($inst1 === $inst2) ...`

# Exceptions

- Une exception représente soit :
  - Une erreur (système)
  - Une erreur (de votre application)
  - Un cas alternatif

# Section de code critique

- Le code potentiellement source d'erreur sera encadré par un bloc try :

```
try
{
    ...
    ... // instruction pouvant poser problème
    ...
}
```

# Capture des erreurs

- Chaque bloc try est suivi d'un ou plusieurs blocs catch
- Si une erreur apparaît, l'exécution de try est stoppée, et le bloc catch apparaît à la place

# Cinématique

.....

```
try
```

```
{
```

```
...
```

```
instruction à problème
```

```
.....
```

```
}
```

```
catch(Exception $e)
```

```
{
```

```
.... gestion de l'erreur
```

```
}
```

```
.....
```

# Cinématique



```
.....  
try  
{  
    ...  
    instruction à problème  
    ....  
}  
catch(Exception $e)  
{  
    .... gestion de l'erreur  
}  
.....
```

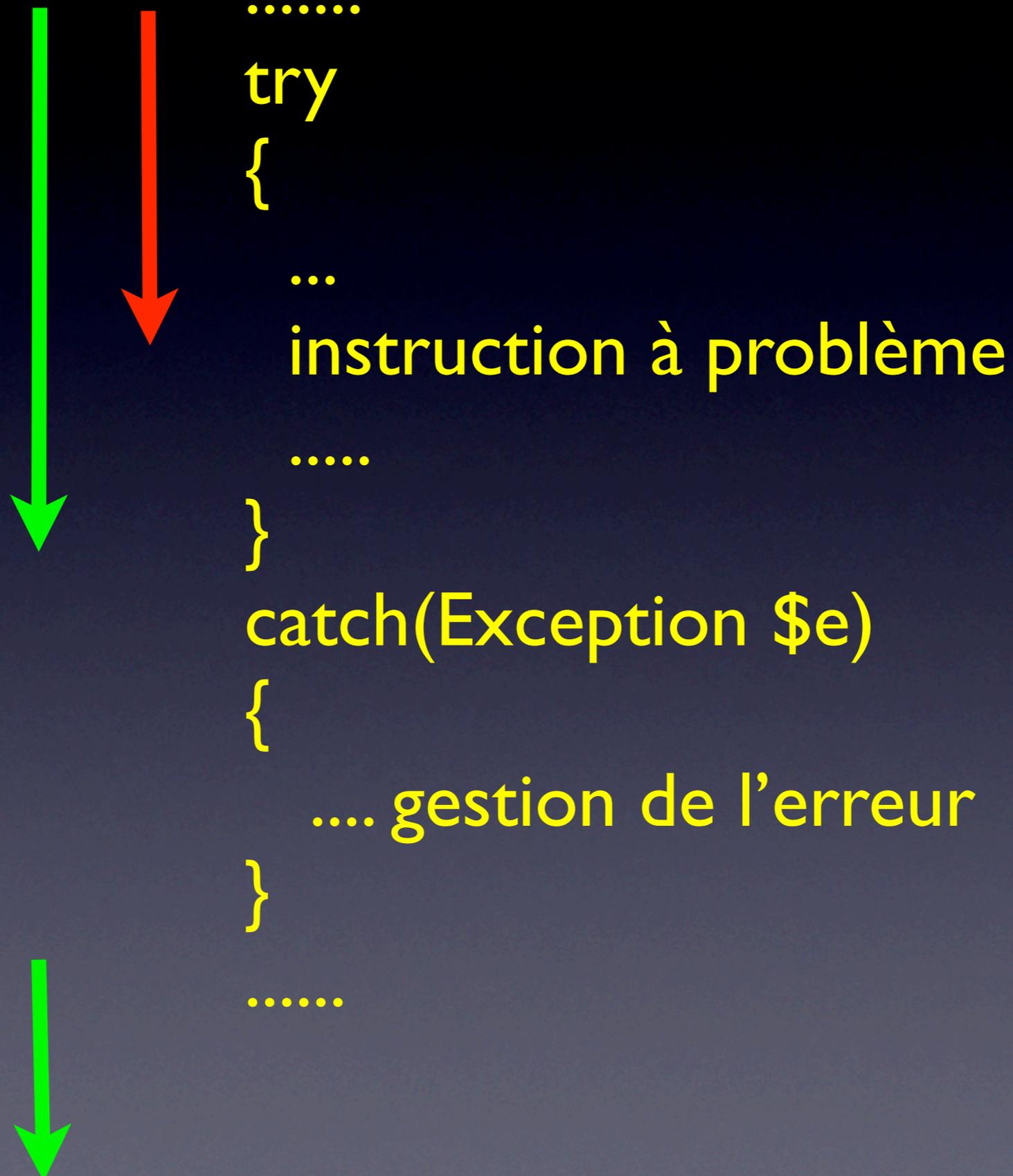
# Cinématique



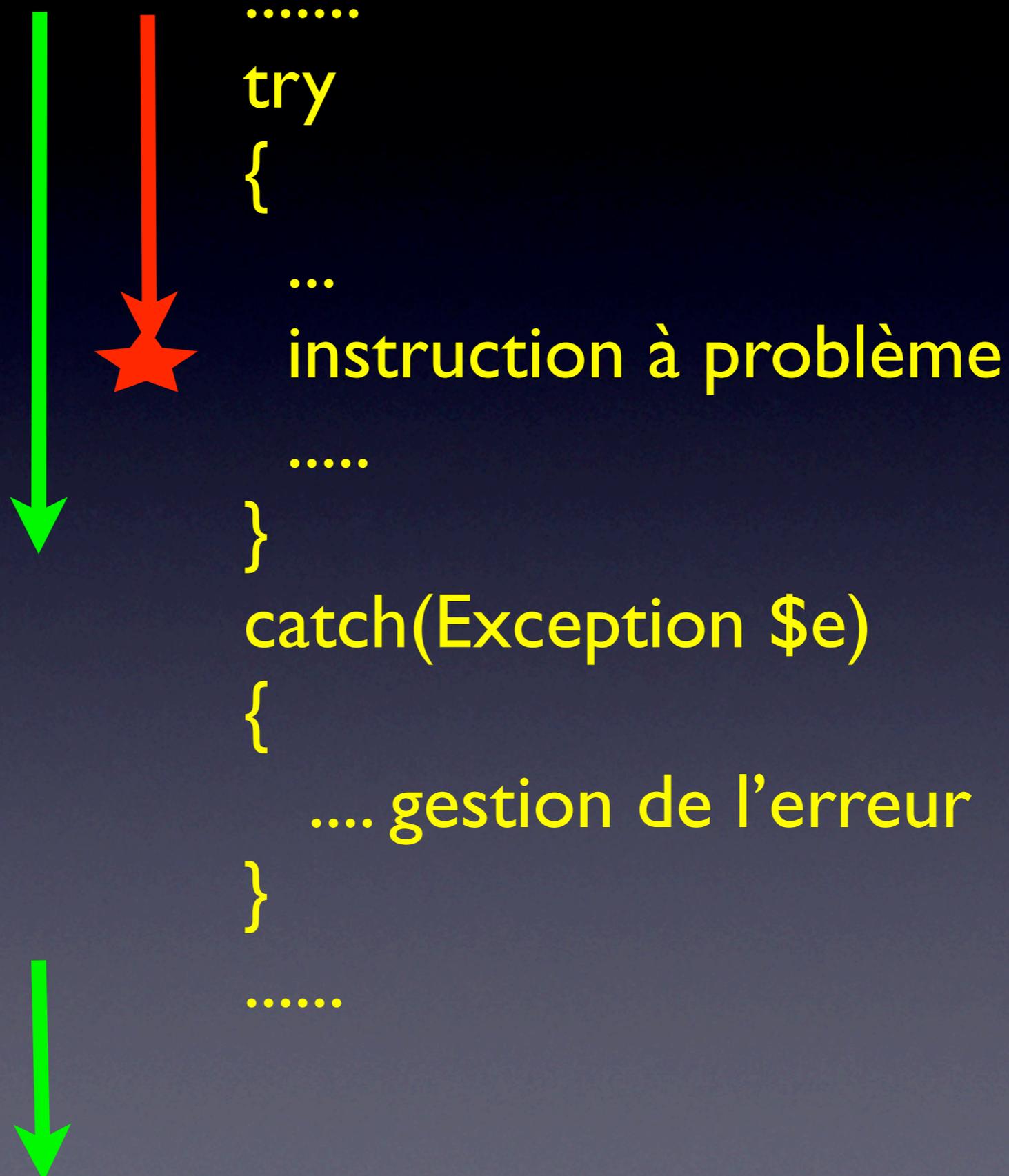
```
.....  
try  
{  
  ...  
  instruction à problème  
  ....  
}  
catch(Exception $e)  
{  
  .... gestion de l'erreur  
}  
.....
```



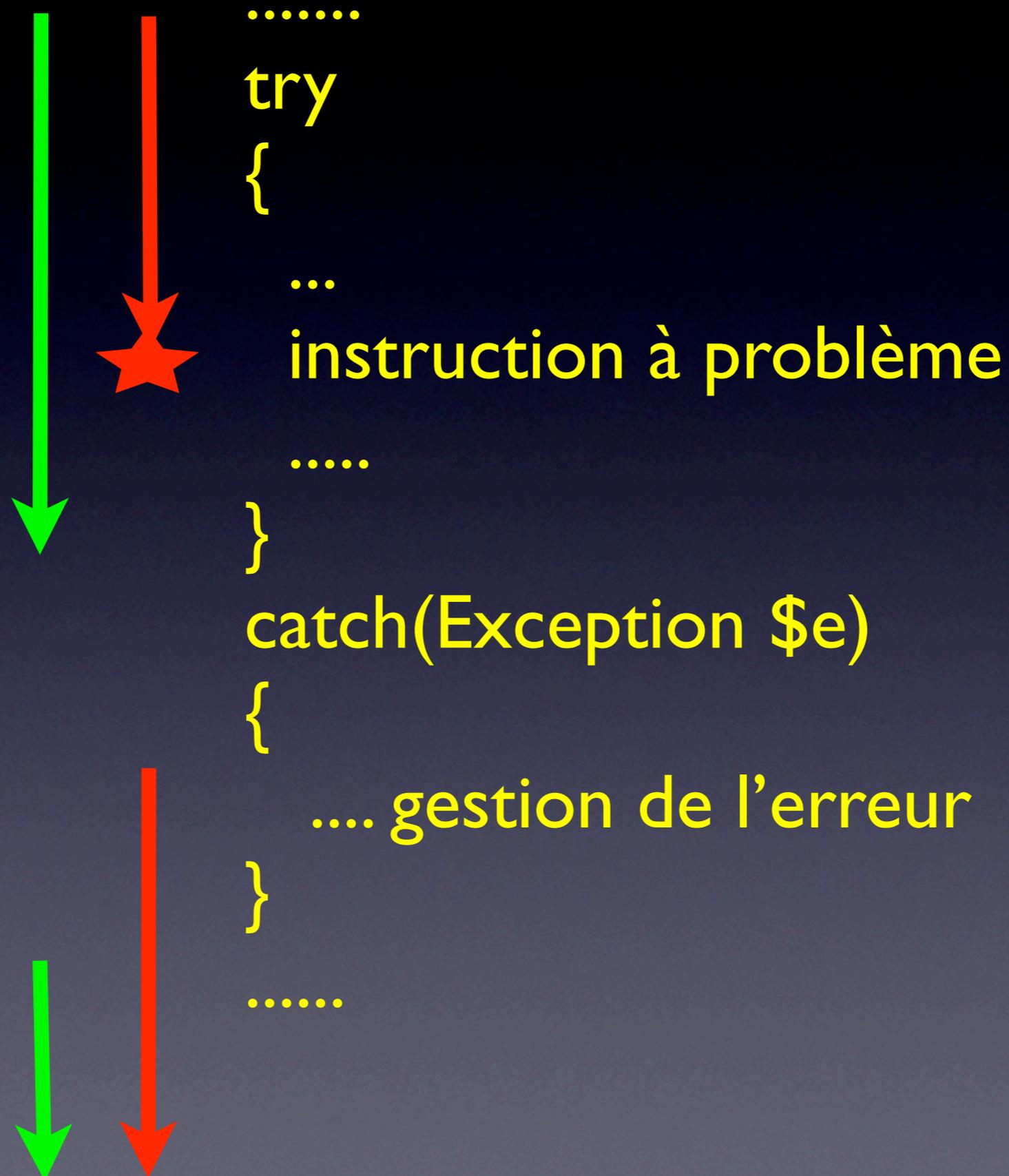
# Cinématique



# Cinématique



# Cinématique



# Que faire en cas d'erreur ?

- L'erreur peut se réparer localement ?
  - On fait un try/catch
- L'erreur est grave, il faut sortir de la méthode
  - On relance l'exception via un throw

# Avant la remontée d'erreur

```
function uneMethode()  
{  
  ...  
  instructionAProbleme  
  ...  
}
```

```
{  
  ...  
  $inst->uneMethode();  
  ...  
}
```

# Remontée d'erreur

```
function uneMethode()
{
    ....
    if(probleme)
        throw new Exception("..");
    ...
}
```

```
{
    ....
    try
    {
        $inst->uneMethode();
    }
    catch(Exception $e)
    { ... }
    ...
}
```



Cet appel  
est devenu section  
de code critique

# Persistance d'information

- Problématique : 1 page PHP = un programme
  - On perd tout une fois la page affichée
- Solutions : rendre “persistante” l'information
  - Par passage de paramètres
  - Par base de données
  - Par éléments statiques
  - Par session utilisateur
  - Par cookie

# Liaison entre client et page PHP

- HTTP prévoit le passage de paramètres du client vers le serveur
  - Ex : visualisation du produit numéro XXX
  - Ex : enregistrement du client dont le nom est XXX, le prénom YYY...

# Logique d'enchaînement de pages

[www.Mcours.com](http://www.Mcours.com)  
Site N°1 des Cours et Exercices Email: [contact@mcours.com](mailto:contact@mcours.com)

- HTTP impose une cinématique lourde :
  1. Saisie sur le client en mode **autonome** (pas d'échange avec le serveur)
  2. Après validation client, transfert des paramètres sur le serveur
  3. Traitement des informations sur le serveur
  4. Transfert d'une nouvelle **page** sur le client

# Passage de paramètre via l'URL

- Syntaxe limitée mais suffisante

<http://localhost/fiche.php?idProduit=12>

<http://localhost/enregcli.php?nom=Capone&prenom=Al>

- Nom de cette méthode : GET
- Limité à la taille de l'URL
  - Souvent, moins de 1000 caractères

# Récupération des paramètres en PHP

```
$id=$_GET['id'];
```

- ou

```
$id=$_POST['id'];
```

- Nb : l'affectation directe des paramètres dans des variables est maintenant désactivé pour des raisons de sécurité

# Passage de paramètres via formulaire

- Permet une interactivité côté client
- Choix de la méthode :
  - GET : via l'URL
  - POST : via un flux de données

# Écriture d'un formulaire

Nom page  
suivante

GET  
ou  
POST

```
<form action="suivante.php" method="GET">  
<input type="text" name="id" />  
<input type="submit" value="Valider" />  
</form>
```

Bouton de  
validation

<http://sitecourant/suivante.php?id=XXX>

# POST ou GET ?

## POST

- **GET**
- On voit les paramètres
- Navigation fluide
- Taille limitée
- Problèmes d'encodage

Volume de données  
"illimité"

URL plus "propres"

Problèmes d'utilisation du  
bouton "Retour" du  
browser

Debug moins facile

Formulaire obligatoire

# PHP et MySQL

- PHP peut interagir avec différents SGBD
- MySQL est toutefois le plus utilisé :
  - Conçu pour le Web (optimisé pour des requêtes simples)
  - OpenSource
  - Bonne coordination entre développeurs

# Connexion à MySQL

```
<?php

if(mysql_connect('host','login','pass')>0)
    echo 'Connexion réussie !' ;
else
    echo 'Connexion impossible !';

if(mysql_select_db('nombase')== True)
    echo 'Sélection de la base réussie' ;
else
    echo 'Sélection de la base impossible' ;

?>
```

# Requête SQL

```
<?php
```

```
$requete="SELECT * FROM users WHERE nom='olek'";  
$resultat = mysql_query( $requete );  
echo 'nb résultats:'.mysql_num_rows( $resultat );  
while($enreg=mysql_fetch_array( $resultat ))  
{  
    echo $enreg['nom'].','.$enreg['prenom'];  
}  
  
?>
```

# Insertion de données

```
<?php  
  
$requete="INSERT INTO user(nom,prenom)  
VALUES( '$nom' , '$prenom' )";  
mysql_query( $requete );  
echo 'clé générée : ' .mysql_insert_id();  
  
?>
```

# Cloisonnage en pages

[www.Mcours.com](http://www.Mcours.com)  
Site N°1 des Cours et Exercices Email: [contact@mcours.com](mailto:contact@mcours.com)

- Une page PHP=une application lancée puis terminée
  - Liaison SGBD close en fin de page
  - Toutes les variables sont perdues

# Modèle objet vs modèle relationnel

- On peut partir de l'un comme de l'autre
  - Le modèle relationnel va permettre de structurer les données
  - Le modèle objet va définir la façon de les exploiter

# Passage d'un modèle à l'autre

- Une table -> une classe
  - Quasiment toujours vrai
- Une liaison 0..N relationnel -> un tableau en attribut
  - Suivant les besoins
  - En prenant soin de limiter la consommation mémoire
- Une structure d'héritage
  - Avec quelques adaptations

# Liaison simple

- On intègre les primitives de lecture/écriture dans chacun des objets métiers

```
class Client
```

```
{
```

```
  public function select($cle) { .... }
```

```
  public function insert() { ... }
```

```
  public function update() { ... }
```

```
}
```

Méthode = les données  
seront issues des attributs de  
la classe

# Liste de résultats

- En objet : c'est un attribut qui contient des instances d'autres objets
- En SGBD : c'est le résultat d'une requête
  - Il va donc falloir faire une requête qui va remplir la liste
  - Problème : comment instancier chacun des éléments de la liste ?

# Exemple de liste en objet

```
class Catalogue
```

```
{
```

```
    private $liste;
```

```
    public function rech($texte)
```

```
    {
```

```
        // remplissage de la liste
```

```
    }
```

```
}
```

```
class Produit
```

```
{
```

```
    private $cle;
```

```
    private $nom;
```

```
    ...
```

```
    public function load($cle)
```

```
    {
```

```
        // lecture d'un enregistrement
```

```
    }
```

```
}
```

# Remplissage “procédural”

```
public function rech($texte)
{
    $sql="SELECT * FROM produit WHERE ...";
    while(...)
    {
        $p=new Produit();
        $p->nom=rs['nom'];
        ...
        $liste[]=$p;
    }
}
```

Problème :

on gère la lecture d'un produit  
en dehors de la classe Produit

# Remplissage “objet”

```
public function rech($texte)
{
    $sql="SELECT cle FROM produit WHERE ...";
    while(...)
    {
        $cle=rs['cle'];
        $p=new Produit();
        $p->load($cle);
        $liste[]=$p;
    }
}
```

Problème :

On génère un grand nombre de requête (N+1 requêtes, N étant le nombre de produits)

# Solution mixte

```
public function rech($texte)
{
    $sql="SELECT * FROM produit WHERE ...";
    while(...)
    {
        $cle=rs['cle'];
        $p=new Produit();
        $p->load($rs);
        $liste[]=$p;
    }
}
```

Dans cette solution, on passe à Produit directement l'objet "rs" afin de traiter la requête à la source

# L'héritage dans un modèle relationnel

- Problématique : il n'est pas possible de définir directement une structure d'héritage dans un système de tables
- Plusieurs solutions sont possibles :
  - 1 table regroupant tout
  - 1 table par classe fille + 1 table pour la classe mère
  - 1 table par classe fille

# Modèle à une table

- A partir d'une hiérarchie de classe Produit -> Cd ou Livre :

## Table produit :

-refprod  
-type  
-nom  
-prix  
-dureecd  
-nbpageslivres

Le type est encodé dans un champ

Cette solution est valable si CD et Livre ont peu de données divergentes

La table contient à la fois les infos du CD, et celles du livre

# Une table par classe fille sans factorisation

- A partir d'une hiérarchie de classe Produit ->

Cd ou Livre :

Cette solution facilite les requêtes mais complique les recherches inter-catégories

Table CD :

- refprod
- type
- nom
- prix
- dureecd

Table Livre :

- refprod
- type
- nom
- prix
- nbpages

La table Livre contient toutes les données

# 1 table par classe fille + 1 table pour classe mère

- A partir d'une hiérarchie de classe Produit ->  
Cd ou Livre :

Table produit :

-refprod

-type

-nom

-prix

Table CD :

-refprod

-dureecd

Table Livre :

-refprod

-nbpages

Cette solution concilie factorisation  
et particularité des types de produits

La table Livre contient  
uniquement les données  
propres au livre

# Lecture base d'une classe polymorphe

- On passe par une Factory qui va délivrer suivant les cas une instance de CD, de Livre...

```
$p=ProduitFactory::getProduit($cle);
```

# Factory modèle à 1 table

```
public static getProduit($cle)
{
    $p=null;
    $sql="SELECT * FROM produit WHERE ...";
    if(...)
    {
        $type=$rs['cle'];
        if($type=="cd")
        {
            $p=new CD();
            $p->load($rs);
        }
    }
    return $p;
}
```

# Factory modèle à 2 tables

```
public static getProduit($cle)
{
    $p=null;
    $p=new CD();
    if(!$p.load($rs))
    {
        $p=new Livre();
        if(!$p.load($rs),
        {
            ...
        }
        else $p=null;
    }
    return $p;
}
```

On ne peut que faire des tests en cascade

# Factory modèle à 3 tables

```
public static getProduit($cle)
{
    $p=null;
    $sql="SELECT * FROM produit WHERE ...";
    if(...)
    {
        $type=rs['type'];
        if($type=="cd")
        {
            $p=new CD();
            $p->load($rs);
        }
    }
    return $p;
}
```

C'est dans chacune des méthodes "load()" que l'on va faire une jointure entre les tables 'mère' et 'fille'

# Données d'une page à l'autre

- Via l'URL
  - Clair mais lourd
- Via la session utilisateur
  - Espace mémoire lié à l'existence d'une session
- Via cookie
  - Espace disque stocké côté client

# Session utilisateur

- Chaque connexion d'un utilisateur sur un site est délimitée
- Permet de faire transiter des informations tout au long de sa connexion
  - Ex : panier d'achat
- Session liée à la fenêtre du navigateur
- Fermeture session : par timeout
  - Fermeture fenêtre non signalée au serveur

# Liaison avec la session

- Doit se faire avant tout affichage

```
session_start();  
if(isset($_SESSION['id'])) { // si id existe dans la session  
    $id= $_SESSION['id'];  
else { // il faut créer et stocker id  
    $id=.....;  
    $_SESSION['id']=$id;  
}
```

# Sérialisation et session

- Permet de 'figer' une instance de classe en mémoire et la transformer en chaîne de caractères
- Avant d'être stockés dans la session, les objets sont sérialisés implicitement

```
$_SESSION['user']=$user;
```

- Ils sont désérialisés avant d'être récupérés

```
$user=$_SESSION['user'];
```

➔ La sérialisation s'effectue en cascade

# Autres sérialisations

- La sérialisation peut être demandée explicitement  
`$serial=serialize($unelInstance);`
- Utile par ex pour stocker dans un fichier, ou transférer en réseau
- Ne pas oublier de désérialiser :  
`$inst=unserialize($serial);`
- Les méthodes `__sleep()` et `__wakeup()` sont appelées avant et après sérialisation

# Liaison avec les cookies

- Un cookie est stocké sur le client
  - Nominatif
  - Limité à 1Ko
  - Avec date de péremption

```
$cookieTimeout=time()+60;  
if(empty($_COOKIE['id'])  
{  
    setcookie('id','1234',$cookieTimeout);  
}
```

# Exemple récapitulatif

- Page suivante, on effectue une saisie/contrôle de login :
  - une classe représente la notion de User
  - son constructeur permet la lecture en base du User, ainsi que le test de validité du User/password
  - une méthode `isOK()` permet de vérifier cette validité

# Vers le pattern MVC

- Une page nommée 'Contrôleur' effectue les tâches suivantes :
  - Récupération des données saisies sur l'interface utilisateur (Vue)
  - Instanciación / appel des objets métiers (Modèles)
  - Reroutage en fonction des retours de méthode
  - Aucun affichage effectué sur cette page

# Schéma 3-tiers avec MVC

Vue

login.php

login  
incorrect!

login :  
pass :

OK

ctrllogin.php

```
session_start();  
...  
$login=$_GET['login'];  
$pass=$_GET['pass'];  
$user=new User($login, $pass);  
if($user->isOK() {  
  
    $_SESSION['user']=serialize($user  
);  
    header("Location: suite.php");  
}  
else  
    header("Location: login.php?
```

Ctrlleur

suite.php

....

.....

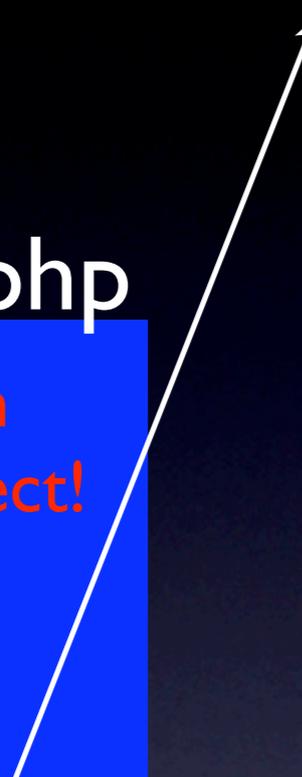
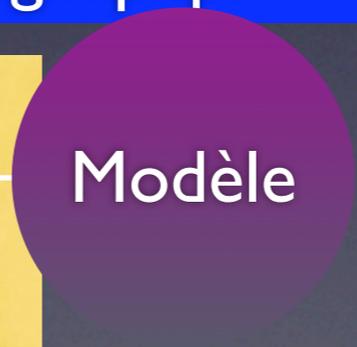
USER

login  
pass

User(login,pass)  
isOK() : boolean

Modèle

SGBD



# Ajax

- Casse le schéma  
“une interaction = un changement de page”
- Passe par des requêtes RPC

# Ajax côté client

- Certains éléments (formulaires...) provoquent un appel RPC sur le serveur
  - Chaque page intègre donc un mini-client RPC
- Les résultats sont récupérés et transformés en HTML
- Le HTML est intégré dynamiquement à la page
  - Via du Javascript

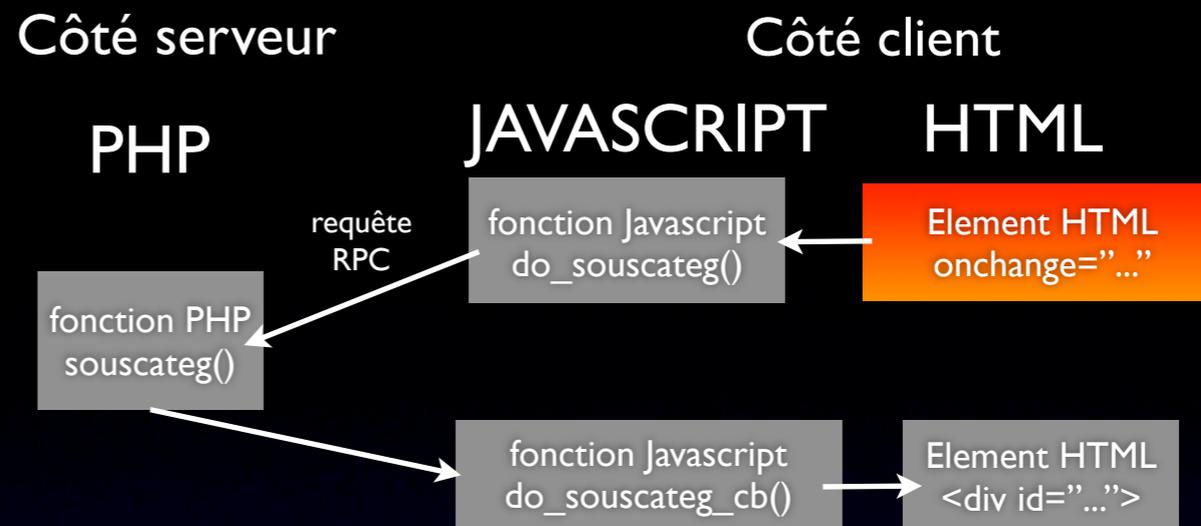
# Exemple avec Simple Ajax (SAjax)

- Ajax n'est en effet qu'un ensemble de recommandation
- Diverses implémentations en différents langages
- Certaines implémentations sont de véritables Framework

# Exemple

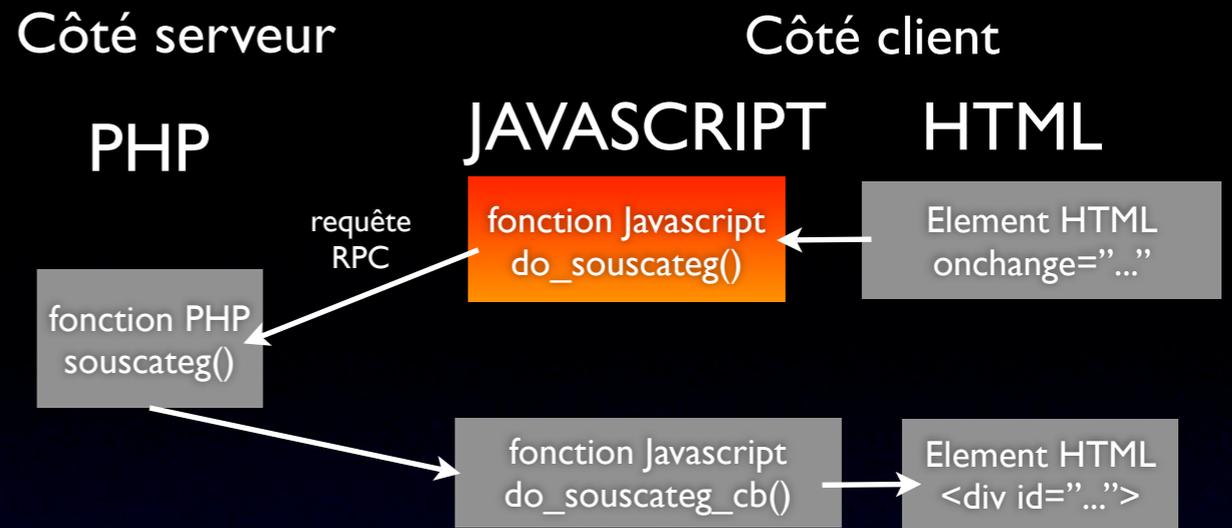
- Problème classique : catégorie/sous catégorie
  - Adapter la liste des sous catég en fonction du choix de la catégorie
- Un premier combo (liste) de sélection provoque un appel sur le serveur
- Le serveur retourne un second combo qui sera affiché dynamiquement

# HTML appelant



```
Catégorie : <select name="categ" id="categ"
onchange="do_souscateg(); return false;">
  <option>Veuillez sélectionner..</option>
  <option>XX</option>
  <option>XXXX</option>
</select>
<div id="souscateg"></div>
```

# Partie Javascript : appel

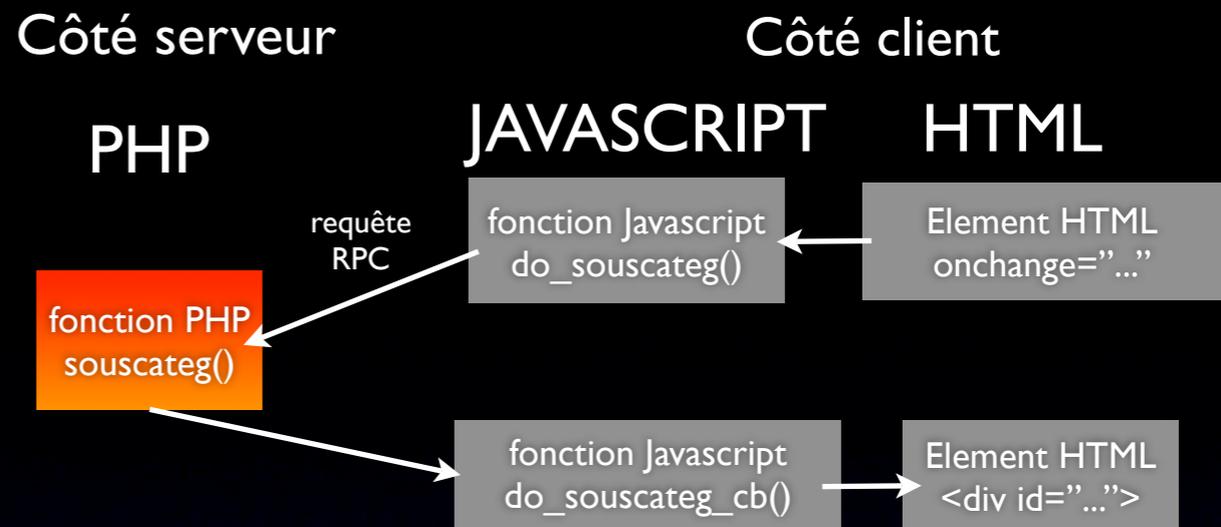


```
function do_souscateg() {  
    var categ =  
    document.getElementById("categ").value;  
    x_souscateg(categ,do_souscateg_cb);  
}
```

Callback

# Partie PHP

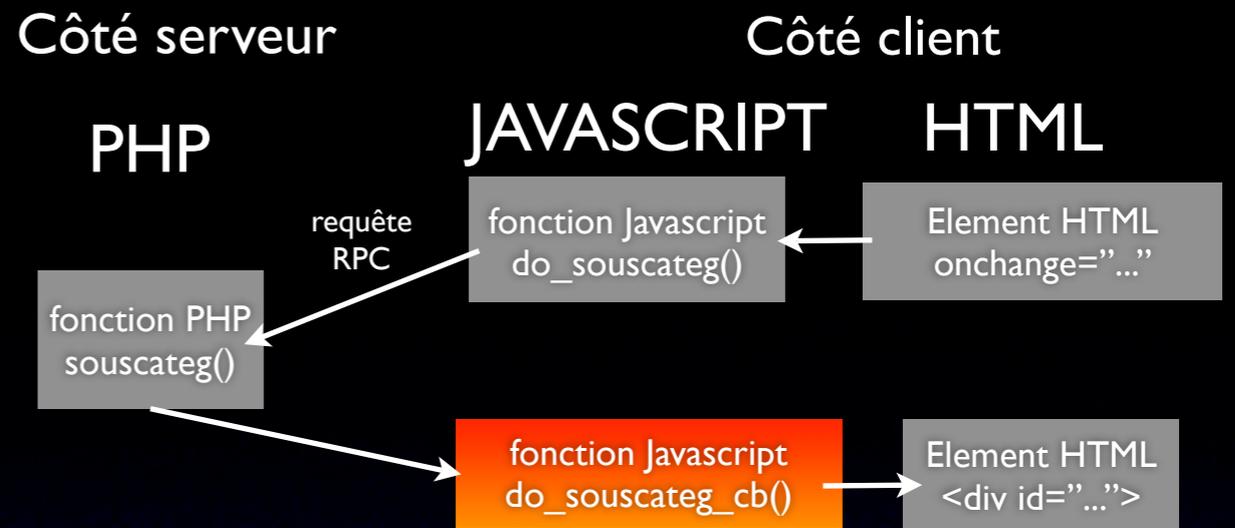
<?



```
require("Sajax.php"); // import du Framework Sajax
```

```
function souscateg($categ) { // génération sous catég
    // requête permettant de récup liste sous catég
    // (appel objets métiers)
    // génération liste
    $ret='<select name="souscateg">';
    $ret=$ret.'<option>XXXX</option>';
    $ret=$ret.'</select>';
    return $ret;
}
```

# Partie Javascript : Callback

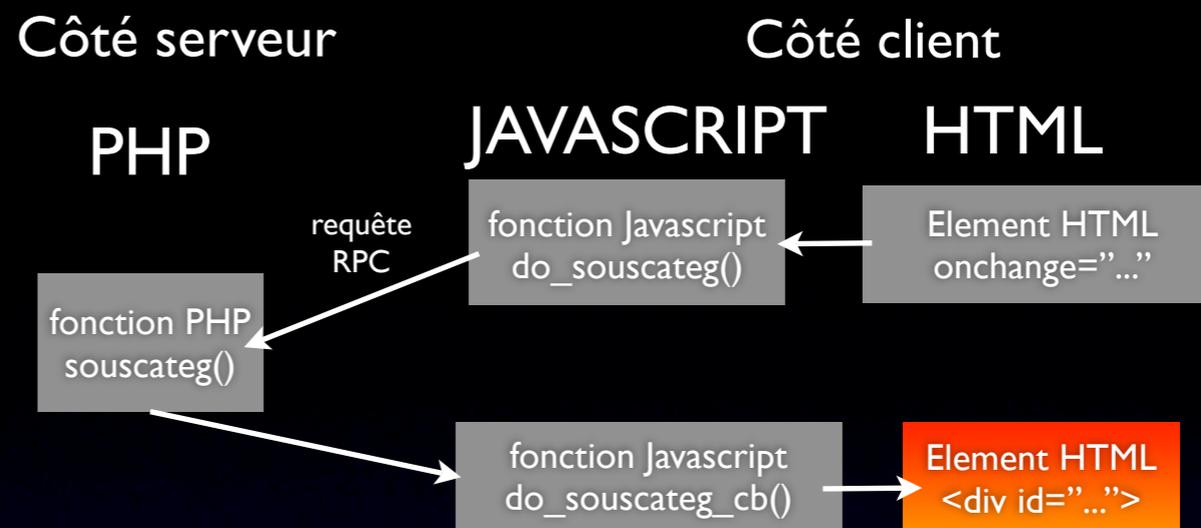


```
<script>
```

```
<? // insertion du client RPC en Javascript  
sajax_show_javascript();  
?>
```

```
function do_souscateg_cb(valretour) {  
    document.getElementById("souscateg").  
        innerHTML = valretour;  
}
```

# HTML génééré



```
Catégorie : <select name="categ" id="categ"
onchange="do_souscateg(); return false;">
  <option>Veuillez sélectionner..</option>
  <option>XX</option>
  <option>XXXX</option>
</select>
```

```
<div id="souscateg"></div>
```

# Structure du fichier complet

```
<?  
    require("Sajax.php"); // import du Framework Sajax  
?>
```

```
    <script>  
    <? // insertion du client RPC en Javascript  
    sajax_show_javascript();  
    ?>  
  
    </script>
```

```
    <html>  
    </html>
```

# Schéma récapitulatif

Côté serveur

PHP

fonction  
PHP  
souscateg()

Côté client

JAVASCRIPT

fonction  
Javascript  
do\_souscateg  
()

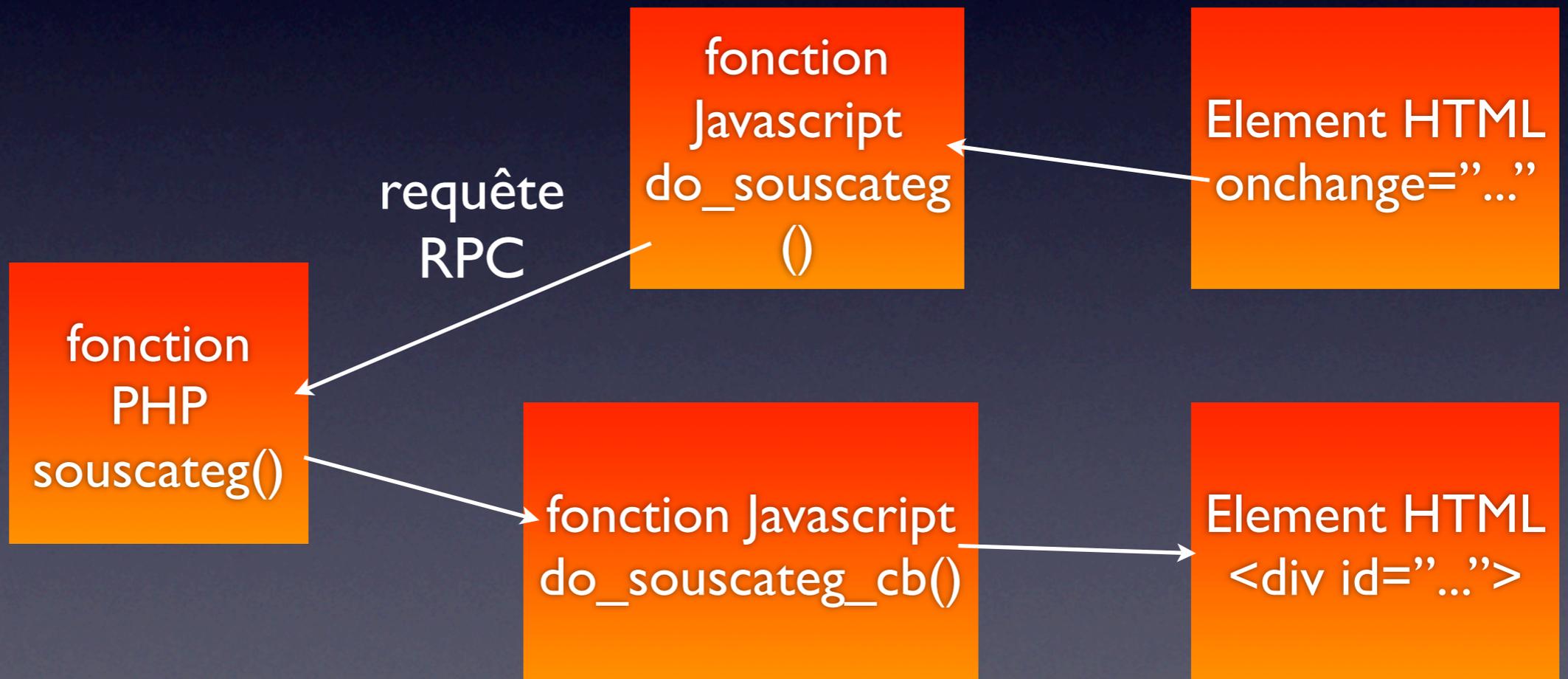
fonction Javascript  
do\_souscateg\_cb()

HTML

Element HTML  
onchange="..."

Element HTML  
<div id="...">

requête  
RPC



# Drag'n'Drop avec scriptaculous

- On va utiliser 3 éléments principaux du framework :
  - Les éléments Draggable
  - Les éléments Droppable
  - La fonction Ajax.Updater

# Élément que l'on peut “tirer” : draggable

- Il s'agit de textes ou d'images

```
<div>
```

```
  
```

```
    <script type="text/javascript">
      new Draggable('product_1', {revert:true})
    </script>
```



```
</div>
```

# Emplacement où poser l'élément : Droppable

```
<div id="panier">  
  Glissez vos produits ici  
</div>  
<script type="text/javascript">  
  Droppables.add('panier',  
    {accept:'products',  
      onDrop:function(element)  
      {  
        // ajax.updater  
      }  
    })  
</script>
```

l'emplacement qui  
recevra les éléments

Fonction  
appelée lors du  
"drop"

# Appel Ajax : Updater

Emplacement  
où sera appelé le  
résultat

Fonction  
appelée lors du  
“drop”

```
new Ajax.Updater('panier', 'ajout.php',  
{  
  parameters:'id=' + encodeURIComponent(element.id)  
})
```

Paramètres  
envoyés à ajout.php

[www.Mcours.com](http://www.Mcours.com)

Site N°1 des Cours et Exercices Email: [contact@mcours.com](mailto:contact@mcours.com)