

*WME Interoperability and  
Customization*

## WME Interoperability

- Basic WME iop comes from the interoperability offered by the Web and Internet
- On top of that, we want to make WME components interoperable in many ways:
  - modular, package, object-oriented organization
  - portability of components
  - mutual reinforcement among components
  - plug-and-play as much as possible
  - easy configuration, customization, mix-and-match, import and export
  - compatible interfaces; reuse of programs
  - systematic extension and expansion of capabilities

## The WME system

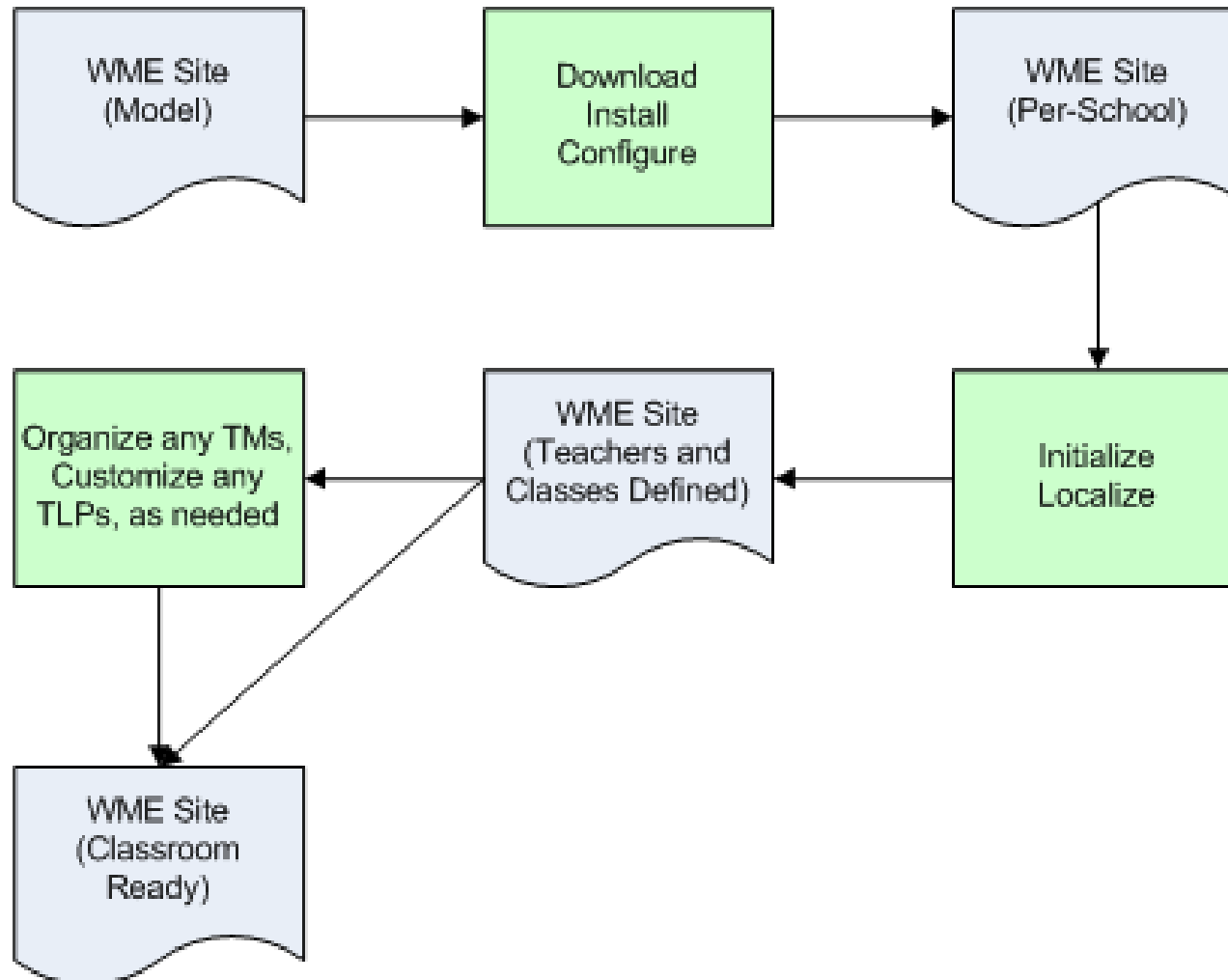
- WME sites (usually per institution)
- Site Architecture, organization, and navigation
- Site Contents: TMs, TLPs, Manipulatives, tools,
- Site Style: CSS, graphics
- Site Operations: generation of pages (of many types) (PHP)  
End-user customization support (PHP+MySQL) Program  
configuration support (PHP+MySQL) Site Admin and  
Configuration support (PHP+MySQL) Page Operations  
Support (server and client side scripts)
- WME server-side tools: assessment (DMAD), MathChat,  
MathBoard

- WME Server-side Plugins (need interface definition)
- Browser and browser plugins (Woodpecker, SVG viewer, Flash player, MathPlayer, Java-plugin, MathEditor, ...)
- WME services (within and without WME sites)

## Ways to Achieve Interoperability

- Context organization, architecture, and naming conventions
- Compatibility and standardization of interfaces
- Protocols and compliance to established standards
- Self-containment and configurability of components
- Modularization and Parameterization

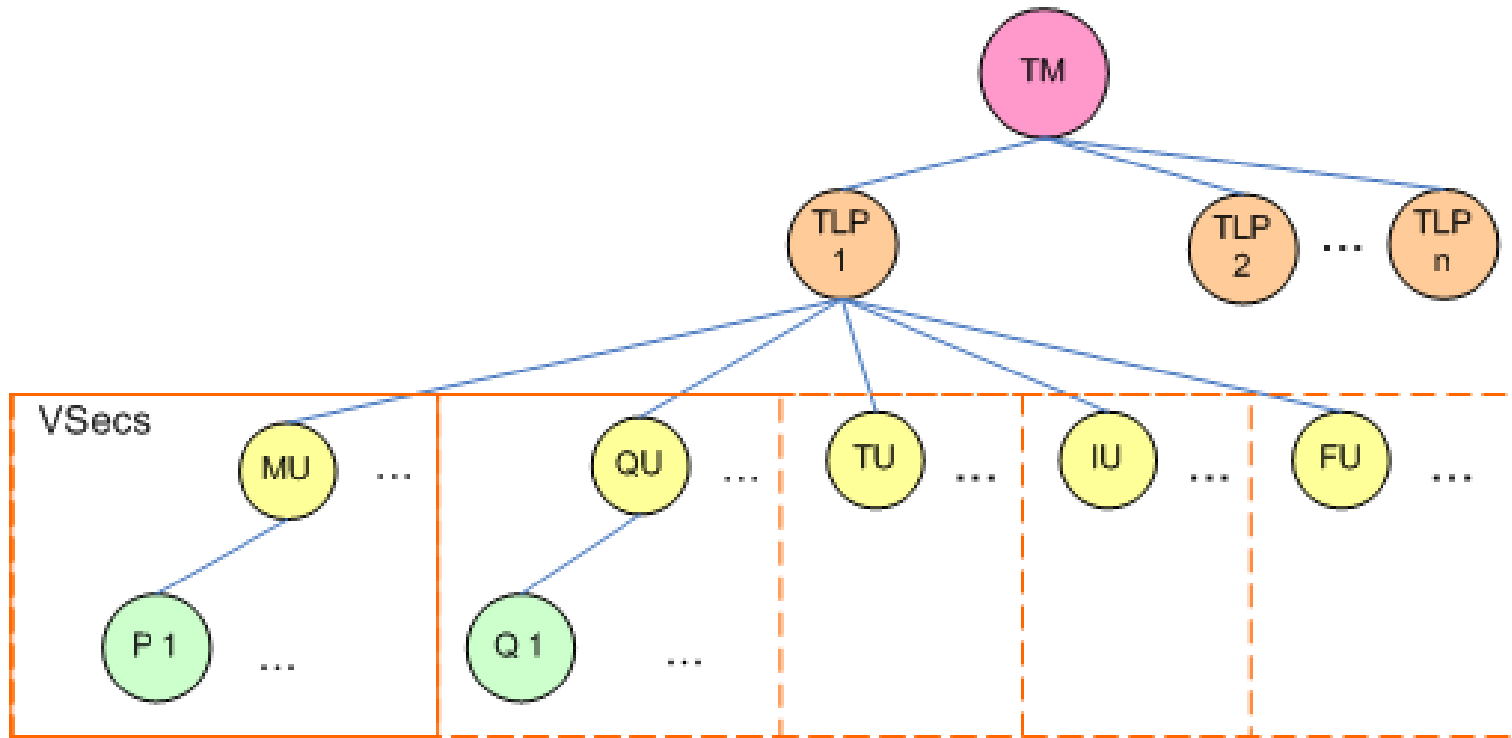
# WME Model Site Deployment



## Site Portability and Relocatability

- A model WME website makes setting up new schools (installations) easy.
- Model site can be downloaded and installed with one self-extracting (wme.sh) file (similar to NN, Mozilla and Firefox distribution on Linux?).
- Browser-based interactive installation and setup.
- Per school configuration and style at setup and other times.
- Automated database (MySQL) installation. (?)
- On-Web WME site admin.

# WME Topic Modules



## TM IOP and Customization

- Interoperability and customization of TMs and TLPs is most important for the *confluence of efforts* in the WME system.
- Initially, a TM lists a sequence of TLPs in (typical/default configuration) so it can be used as-is.
- When customizing a TM, a teacher can select from all supplied TLP in the module and sequence them.
- Available TMs can be selected or deselected at any time by teachers as part of a course.
- TMs are customizable per teacher and per class.
- TMs can be added into or removed from a working WME site by WME webmaster/administrator easily.

## WME Topic Lesson Pages

- Each TLP is a self-contained unit for easy plug-and-play.
- Each TLP contains a set of consecutive *visibility-control sections* (Vsec's). Each Vsec consists of editable and un-editable content. *Editable units* include TU, IU, FU, QU, and MU. Each of these may require a different editing tool.
- Customizations of TLPs are per-teacher and per-class.
- After customization, a TM (TLP) can be saved under a different name.

## Manipulative Interoperability

- Each manipulative is statically a self-contained program and dynamically an object instance so it is easily deployed in any TLP at any WME site.
- A TLP may contain one or more instances of the same manipulative and the same manipulative may be deployed in multiple TLPs.
- The containing Web page must be able to interact with any manipulative instance through a well-defined interface.
- Each manipulative is customizable by teachers on a per-instance, per-course, and per-teacher basis.

## Manipulative Architecture

- Each manipulative is self-contained and has its own directory for all of its files. A JavaScript source file, `MealOrder.js` for example, which defines, among other things, a constructor (`MealOrder`).
- To create a manipulative instance: `american = new MealOrder('american');`. An associative array of name-value pairs can be supplied as the 2nd arg.
- Deploying in a page:

```
<div class="manipulative" id="manip1">  
  <script type="text/javascript">  
    american.deploy("manip1"); </script></div>
```
- The manipulative CSS style class

## Manipulative Methods

Each manipulative must define these instance methods:

- `reset()`—Re-initializes the manipulative instance.
- `getArg(name)`—Returns the value of the named parameter.
- `getArgDescription(name)`—Returns the description of the parameter suitable for user consumption.
- `getArgs()`—Returns an associative array of the values of all parameters (in the same form as the array used to call the constructor).
- `setArg(name, value)`—Sets the named parameter to the given value.
- `getProperties()`—Returns an associative array of the values of

all instance properties made accessible.

- `getProperty(name)`—Returns the named instance property. For example, `total` (the total amount of the bill) is an instance property of a `MealOrder` instance.

## Questions and Manipulatives

```
<span class="pageparameter" id="labor_percent">50</span>
```

This editable percentage value and the menu order total are automatically passed to the answer checker via the code

```
american.getProperty('total')
```

```
document.getElementById('labor_percent')  
    .firstChild.nodeValue
```

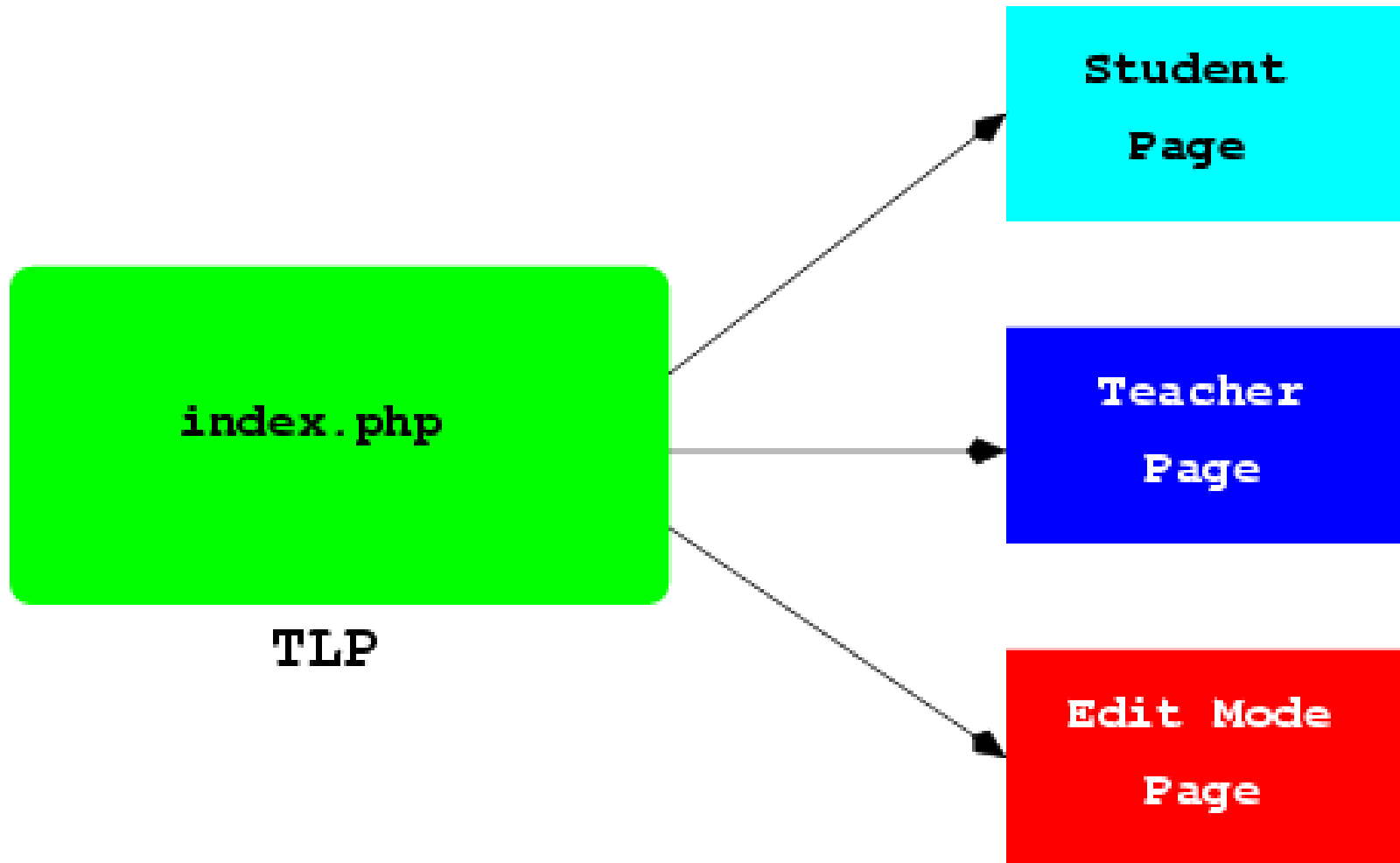
## Static Members of A Manipulative

Each manipulative provides these static (class-wide) properties to help usage and customization.

- instances—Is an array of ids (strings) for all existing instances.
- displayType—Is either "block" or "inline".
- location—Gives manipulative directory location relative to site root.
- document—Is a text string providing a brief API documentation.
- help()  
—Returns a string for end-user help.
- help("topic")  
—Returns a string for end-user help relating to the given topic.

- defaultArgs—Is an associative array for the default arguments when the constructor is called with no args.

# Dynamic Page Generation



## Editable Units

- Use PHP and MySQL to store per-teacher and per class customizations.
- Editable units are contained in index.php of a TLP directory, so they can be generated dynamically.
- Edited units are stored in the database or in files with automatically generated file names.

## Implementation Ideas

```
function htmlUnit(id)
{
  global $pageMode;
  if ( $pageMode == "customize" )
  {
    echo ' class="editable_html" ' .
        ' onclick="editHtml(\'' .
        id . '\')" '; }
}
```

```
function displayControl(id)
{
  if ( $pageMode == "teacher" )
    displays HTML toggle box
    to control visibility
}
```

```
<div class="VSec" id="introduction">
<? displayControl("introduction") ?>
    . . .
<anytag  <?php htmlUnit("foobar"); ?>
    id="foobar"    ...  >
    <?php $ct = getCustomContent("foobar");
        if { ( ct !== "" ) echo $ct; }
        else {
    ?>
        editable content
    <?php } ?>
</anytag>
    . . .    </div>
```

## Other Editable Units

```
<div class="editable_manipulative"  
    onclick="editManipulative(me)" >  
  
    <!--BEGIN_WME_EDIT  
        instanceof="MealOrder"  
        parameters="american.mpxml"  
    END_WME_EDIT-->  
  
    . . .  
</div>
```

```

<math class="editable_math"
      onclick="editMath(me)" ... > ... </math>

<div class="editable_questions"
      onclick="editQuestons(me)" >
<!--BEGIN_WME_EDIT
      question_set="percent_meal_1"
END_WME_EDIT-->

      . . .
</div>
```

## Browser Actions on Editables

- Editables have no special effect while displayed in normal mode.
- In normal mode, no special CSS rules are placed in the classes and event handlers are defined to do nothing.
- Editables take on distinct look (via CSS on the classes) and action (via properly define the event handlers in Javascript) when in customization mode.
- The mode switch is controlled by a GET or POST parameter sent to `index.php`.