XC 8th Dec, 2007

This release's purpose

- This is "snapshot" of a sandbox branch.
- Review Task Mechanism and Rendering Sequence.
- Includes a light-weight sample that uses new mechanisms.
- Other mechanisms (Cube 0.9 features) are not implemented.
- CVS: "minahito_sandbox" branch of "XCube_PHP4".

Roadmap

- 2008 Jun ... next snapshot
- 2008 Feb ... “Alpha”
  - Fix Virtual Service Mechanism.
  - Fix Manifesto File Format.
- 2008 Mar ... “Beta”
- 2008 May ... “Stable”
  - Finish Document Work

- If we need sample BASE to prove the spec of XOOPS Cube, I need more time.

- And, Legacy may steal much time from a developer.

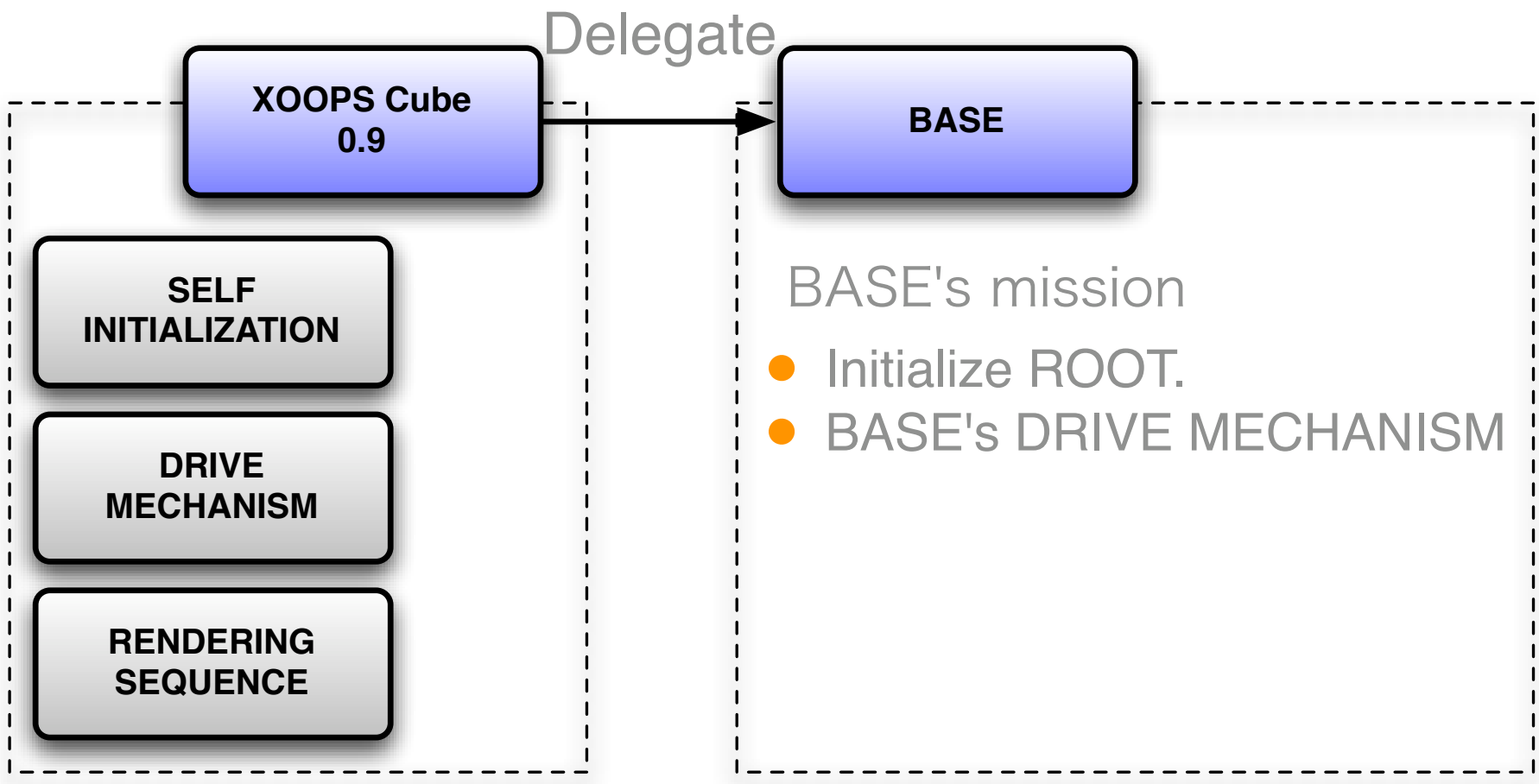
XOOPS Cube's MISSION

A Part Of Goals

- Exchangable CMS layer (aka BASE)
- Site Owners can use free combination subsystems.
- Site Owners can use free combination modules.
- Site Owners can use free combination themes.
- Site Owners can tweak their site easily.
- Developers work each mechanism in combination free.
- Needs a mechanism so that modules may run on multi BASEs easily.

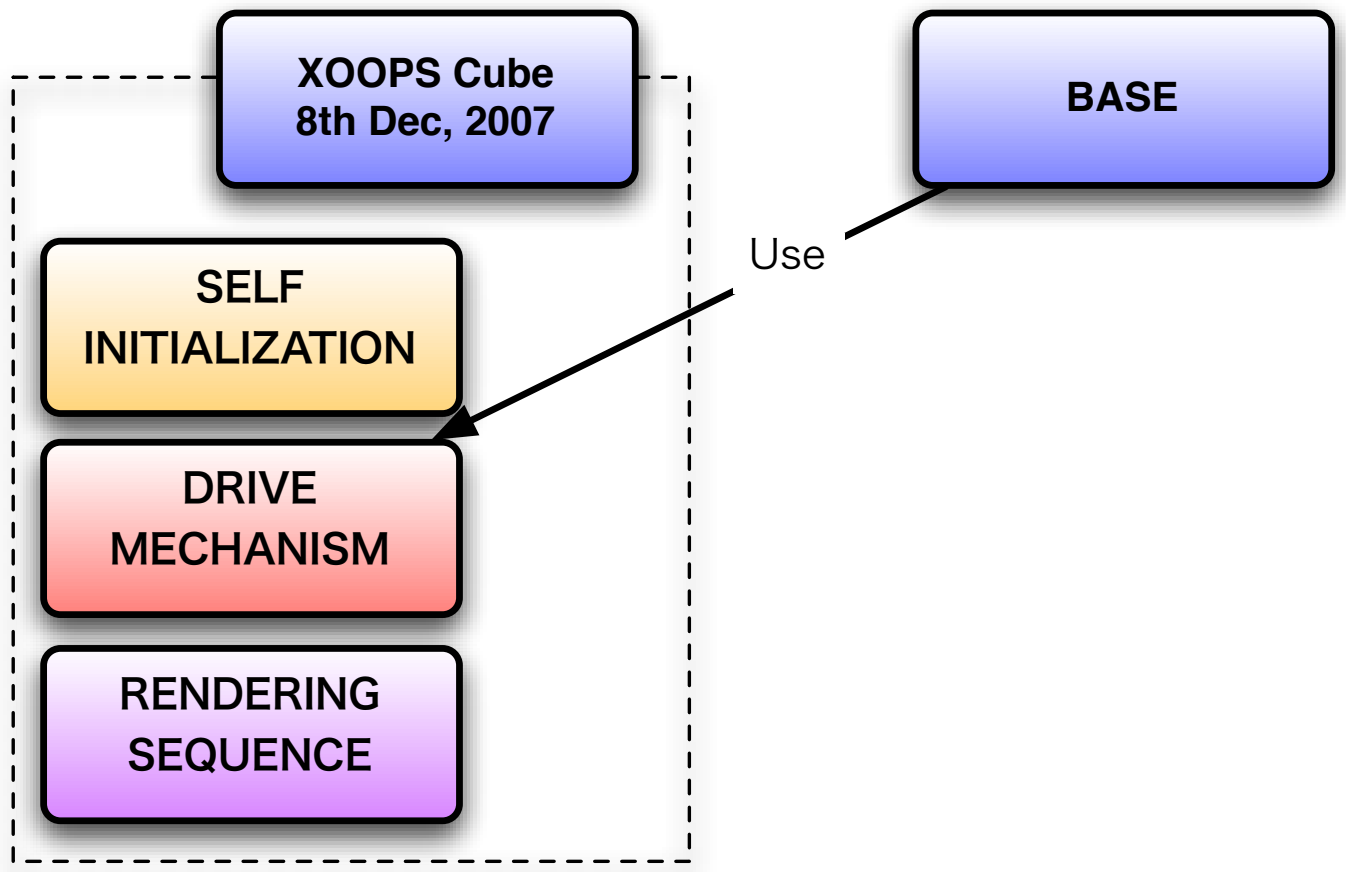
- To realize those, the core has to provide Common Process Mechanism to the BASE side.
  - “Detachable & Insert Free” Sequential Process.
  - Rendering Sequence for “Undefined Final Output”.

0.9 Implementation



- BASE has to implement each Drive Mechanism.

This version



- Implemented “Initialization”, “Drive Mechanism” and “Rendering Sequence”.

- BASE's index.php is 3 line.
- XCube_Root::execute() is 12 line for 3 processes.
- This is easy.
- You need to understand "Task" technic & Render Sequence.

XOOPS Cube Main Routine

Typical index.php

```php
<?php

require_once "../../core/XCube_Root.class.php";
require_once "../common/Sample_RenderSystem.class.php";
require_once "./classes.php";

$g_root =& XCube_Root::getSingleton();
$g_root->loadSiteConfig("./config/site_default.ini.php");

$g_root->execute();

?>
```

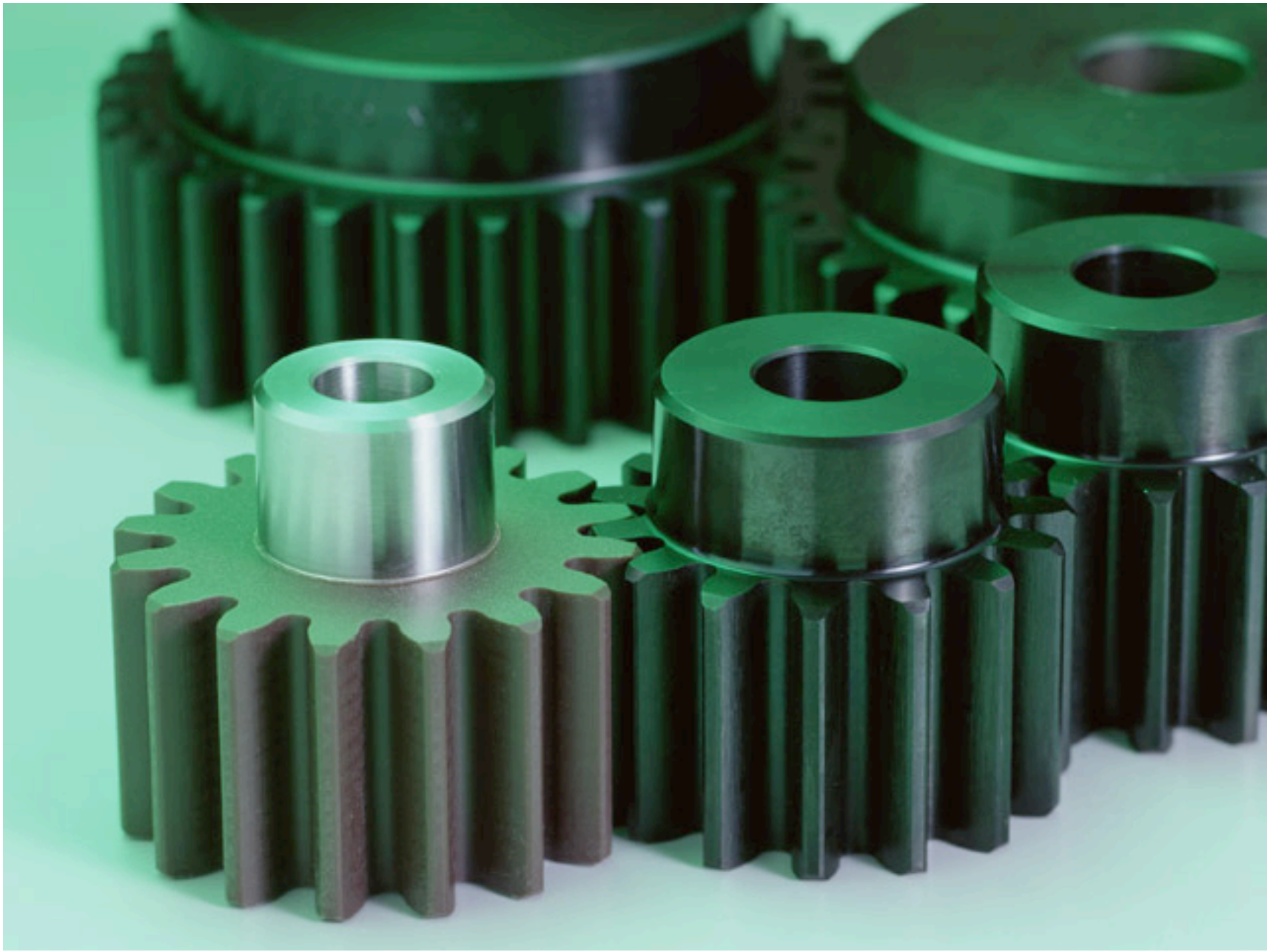
- Base's index.php is just 3 lines.

```
function execute()
{
        $this->initialize();

        $dmy = null;
        $rootTask =& new XCube_Task("root", $dmy);

        $this->mController->buildTask($rootTask);

        $rootTask->initializeAll();
        $rootTask->updateAll();
        $rootTask->drawAll();

        //----------------------
        // Rendering Sequences
        //----------------------
        $collector =& new XCube_RenderOpCollection();
        $rootTask->acceptCollectorAll($collector);

        $visitor =& new XCube_RenderableVisitor();
        $collector->acceptVisitor($visitor);

        print $this->mRenderTargetScreen->getResult();
}
```

- XCube_Root::execute() is 12 lines.

```
function execute()
{
    $this->initialize();

    $dmy = null;
    $rootTask =& new XCube_Task("root", $dmy);

    $this->mController->buildTask($rootTask);

    $rootTask->initializeAll();
    $rootTask->updateAll();
    $rootTask->drawAll();

    //----------------------
    // Rendering Sequences
    //----------------------
    $collector =& new XCube_RenderOpCollection();
    $rootTask->acceptCollectorAll($collector);

    $visitor =& new XCube_RenderableVisitor();
    $collector->acceptVisitor($visitor);

    print $this->mRenderTargetScreen->getResult();
}
```

- Initialization, Drive Mechanism and Rendering Sequence.



- Core defines the type of gears, and SWITCH ON!

Initialization

Confirm Missions

- Exchangable CMS layer (aka BASE)
- Site Owners can use free combination subsystems.
- Site Owners can use free combination modules.
- Site Owners can use free combination themes.
- Site Owners can tweak their site easily.
- Developers work each mechanism in combination free.
- Needs a mechanism so that modules may run on multi BASEs easily.

```
function execute()
{
        $this->initialize();

        $dmy = null;
        $rootTask =& new XCube_Task("root", $dmy);

        $this->mController->buildTask($rootTask);

        $rootTask->initializeAll();
        $rootTask->updateAll();
        $rootTask->drawAll();

        //----------------------
        // Rendering Sequences
        //----------------------
        $collector =& new XCube_RenderOpCollection();
        $rootTask->acceptCollectorAll($collector);

        $visitor =& new XCube_RenderableVisitor();
        $collector->acceptVisitor($visitor);

        print $this->mRenderTargetScreen->getResult();
}
```

- XOOPS Cube got to initialize itself, without BASE.

```
[Cube]
Controller=Sample01_Base

#
# You can register plural render systems.
#
[RenderSystems]
Sample_RenderSystem=Sample_RenderSystem


#             #
# components  #
#             #
[Sample01_Base]
class=Sample01_Base

[Sample_RenderSystem]
class=Sample_RenderSystem
```

- Writes all configuration on the setting ini file.
- The core uses inner default setting for empty setting item.

Drive Mechanism

Confirm Missions

- Exchangable CMS layer (aka BASE)
- Site Owners can use free combination subsystems.
- Site Owners can use free combination modules.
- Site Owners can use free combination themes.
- Site Owners can tweak their site easily.
- Developers work each mechanism in combination free.
- Needs a mechanism so that modules may run on multi BASEs easily.

- The traditional programing technic “Task” is availableness.

What is Task!?

- A kind of the traditional programing technic for modulable sequential process.
- It's good at CHANGE, EXCHANGE and EXTEND very much.
- Many programers know this popular technic.
- You can search "Process Control Block" or "Task Control Block" in Wikipedia, to know this technic.
- This is also known as the traditional technic of Japanese video game development.

- Task is a Common Socket, so it's easy that modules become able to run on multi BASEs.
- This is not original way but the traditional & popular way of the computer world. So it's easy to understand. Or many programers know already.
- This technic has a good experience for missions like XOOPS Cube.
- It is like ActionFilter. But, it's possible that the form of the task list changes and receive new tasks in runtime.

```
function execute()
{
        $this->initialize();

        $dmy = null;
        $rootTask =& new XCube_Task("root", $dmy);

        $this->mController->buildTask($rootTask);

        $rootTask->initializeAll();
        $rootTask->updateAll();
        $rootTask->drawAll();

        //----------------------
        // Rendering Sequences
        //----------------------
        $collector =& new XCube_RenderOpCollection();
        $rootTask->acceptCollectorAll($collector);

        $visitor =& new XCube_RenderableVisitor();
        $collector->acceptVisitor($visitor);

        print $this->mRenderTargetScreen->getResult();
}
```

- I'm going to explain that part:

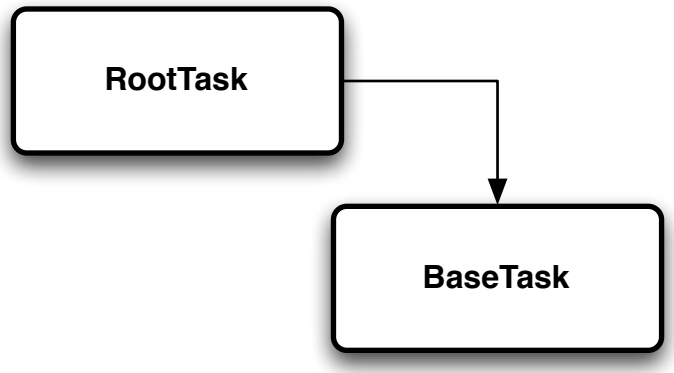
Overview

1. The Root generates the root task.
2. The BASE joins its tasks as child tasks to the root task.
3. At the base task's initialization, the task joins requirement tasks of active modules.
4. It handles the list of tasks as the sequential process.

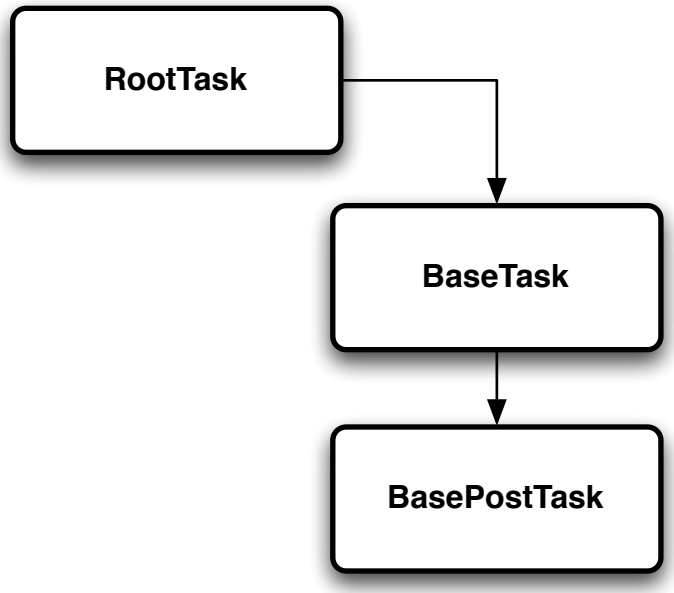
```
$dmy = null;
$rootTask =& new XCube_Task("root", $dmy);
```

- At the beginning, generates a root task.
- This task does nothing.

```
$this->mController->buildTask($rootTask);
```

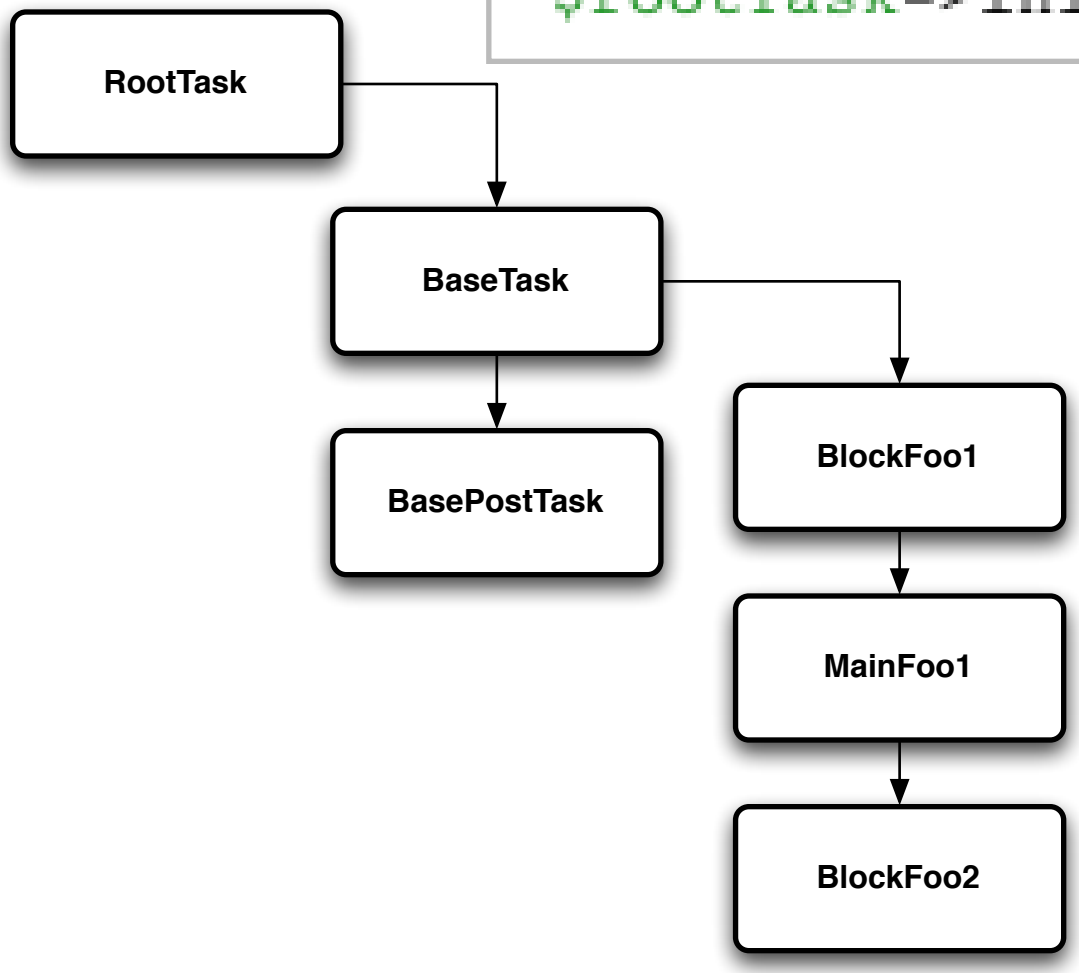


- Orders BASE to generate BASE tasks.

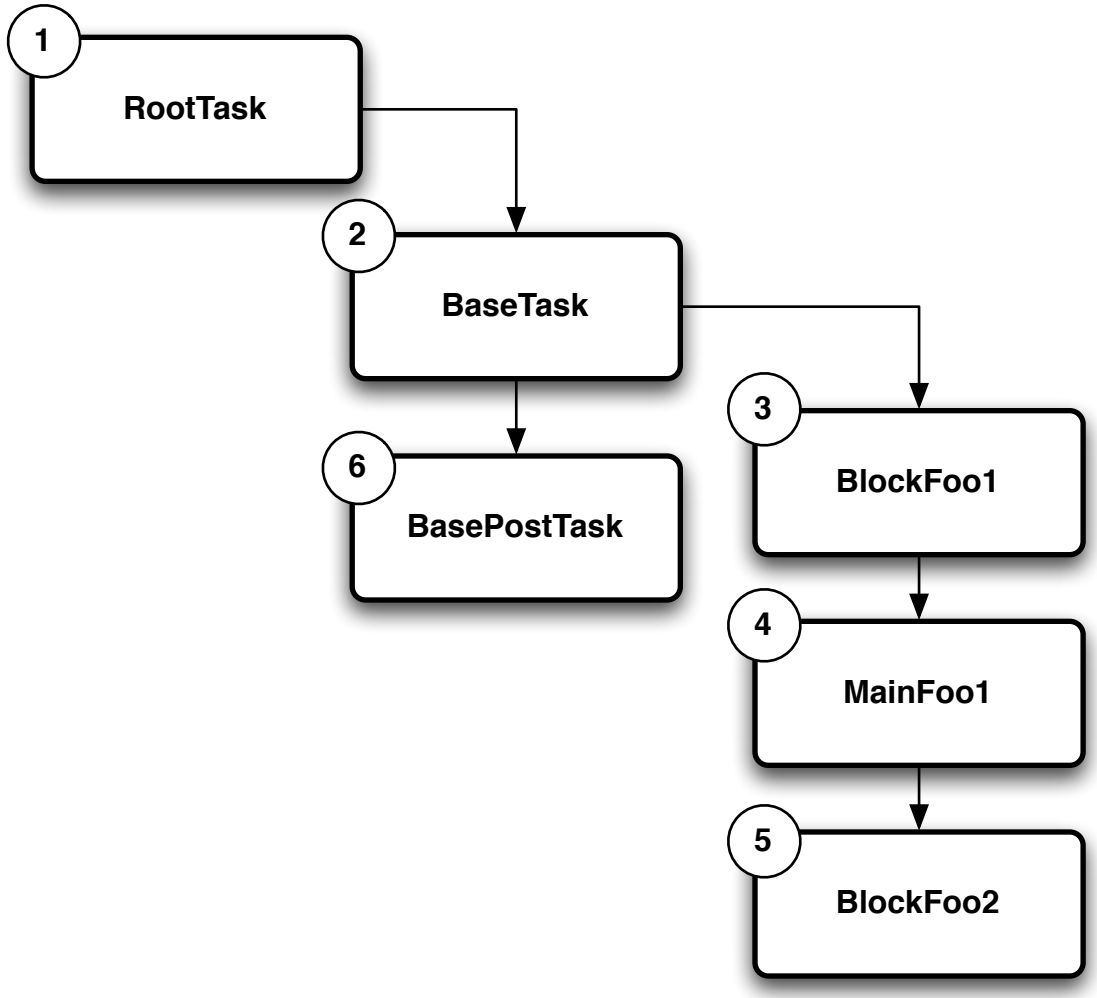


- BASE may join the plural tasks, if it needs those.

```
$rootTask->initializeAll();
```

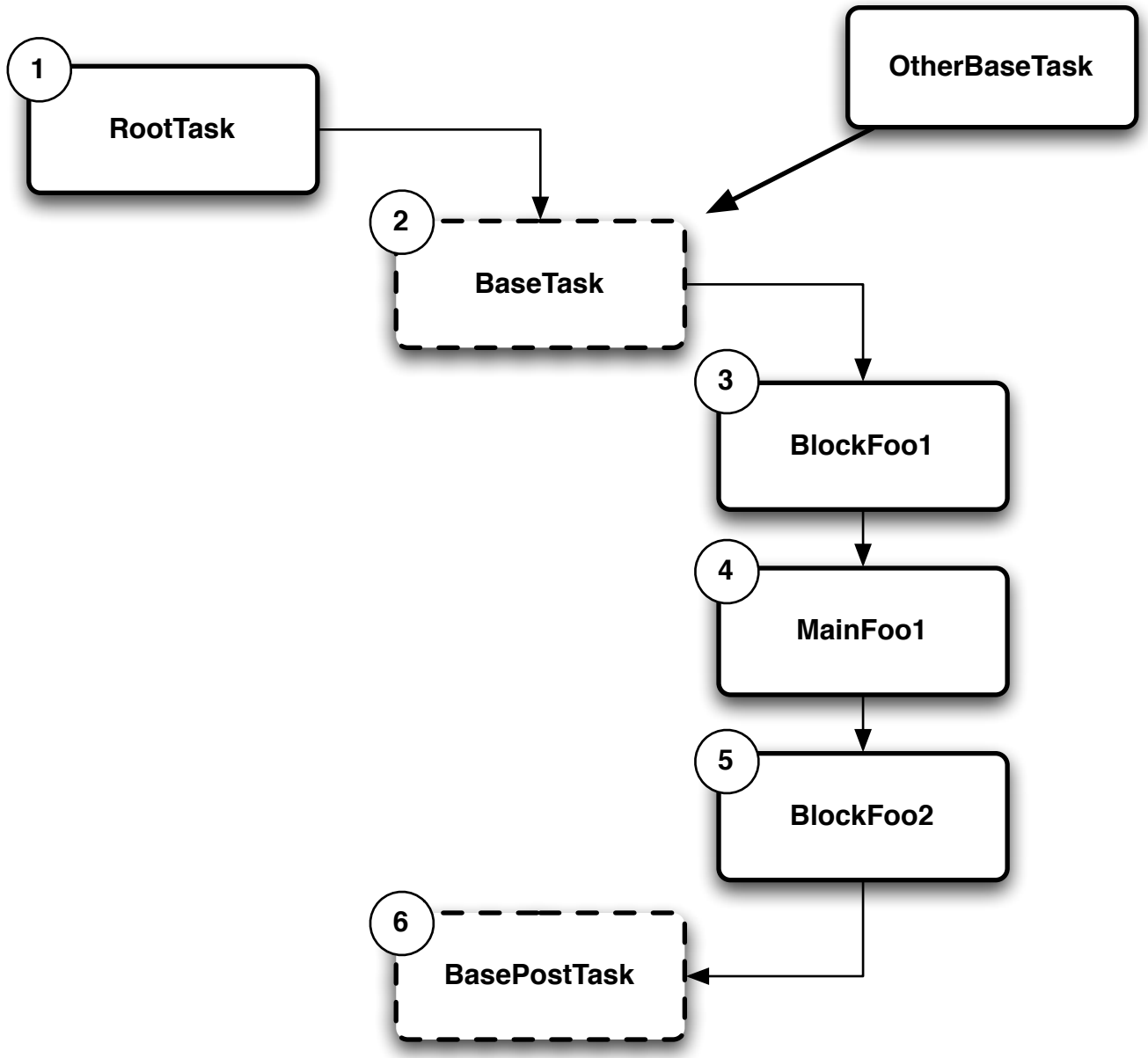


- The Root begins initialization for the task list.
- At the base task's initialization, the task joins requirement tasks of active modules.

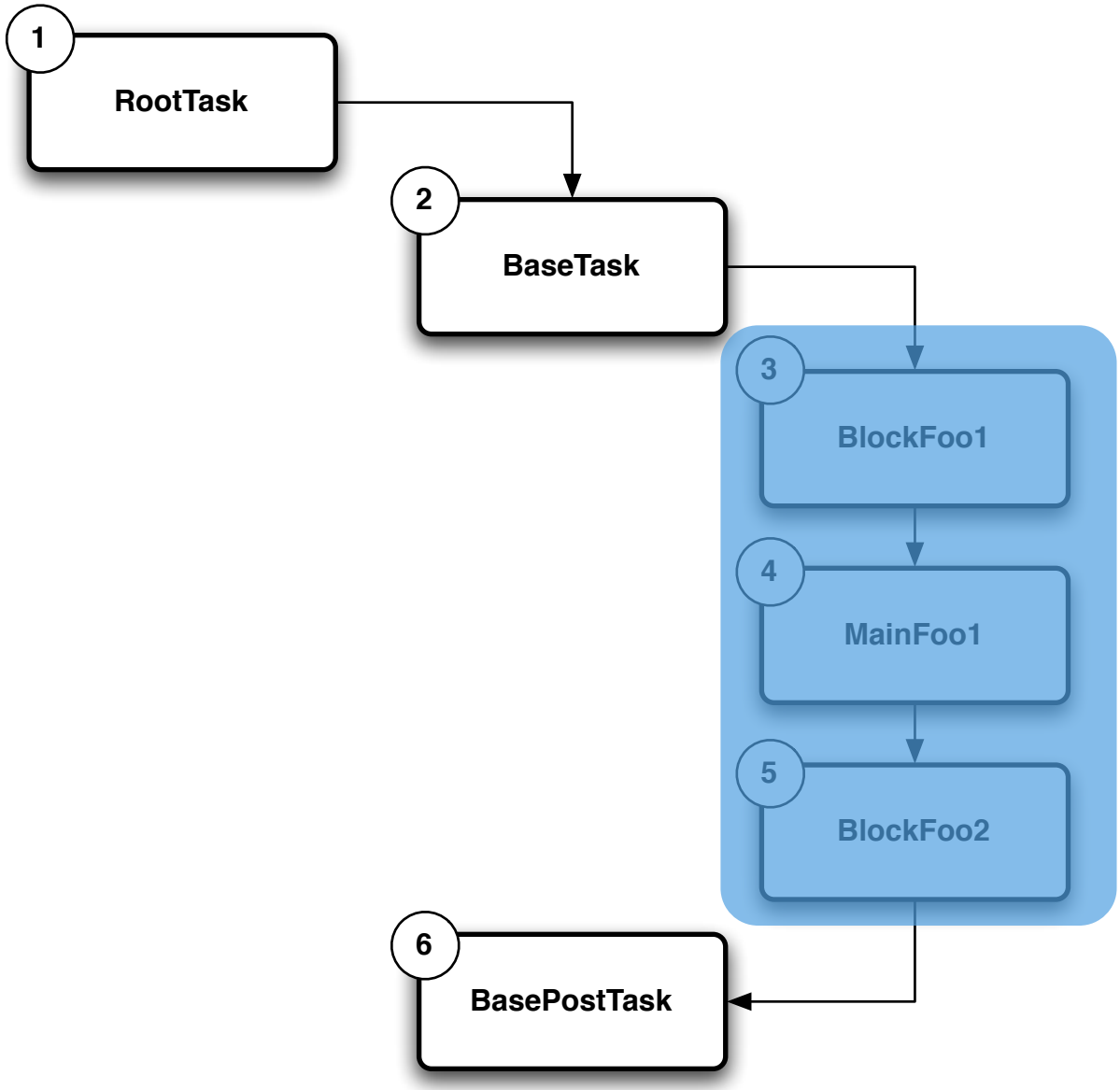


- The sequential process has been just decided.
- Go the sequential process:

```
$rootTask->updateAll();
$rootTask->drawAll();
```



- Site Owners can exchange BASE.



- Modules' tasks has possibility to run on other BASEs.

Rendering Sequence

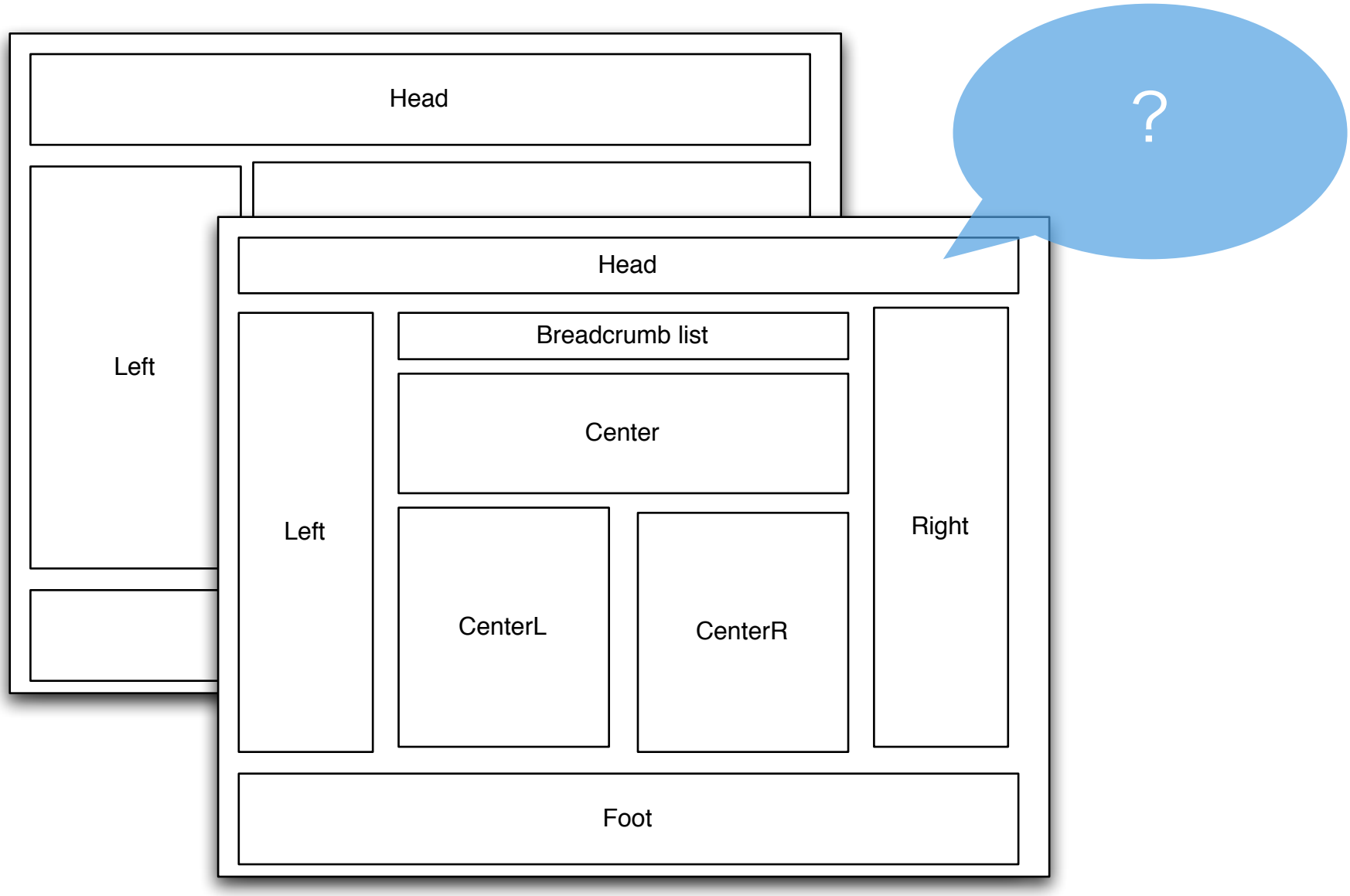
Confirm Missions

- Exchangable CMS layer (aka BASE)
- Site Owners can use free combination subsystems.
- Site Owners can use free combination modules.
- Site Owners can use free combination themes.
- Site Owners can tweak their site easily.
- Developers work each mechanism in combination free.
- Needs a mechanism so that modules may run on multi BASEs easily.

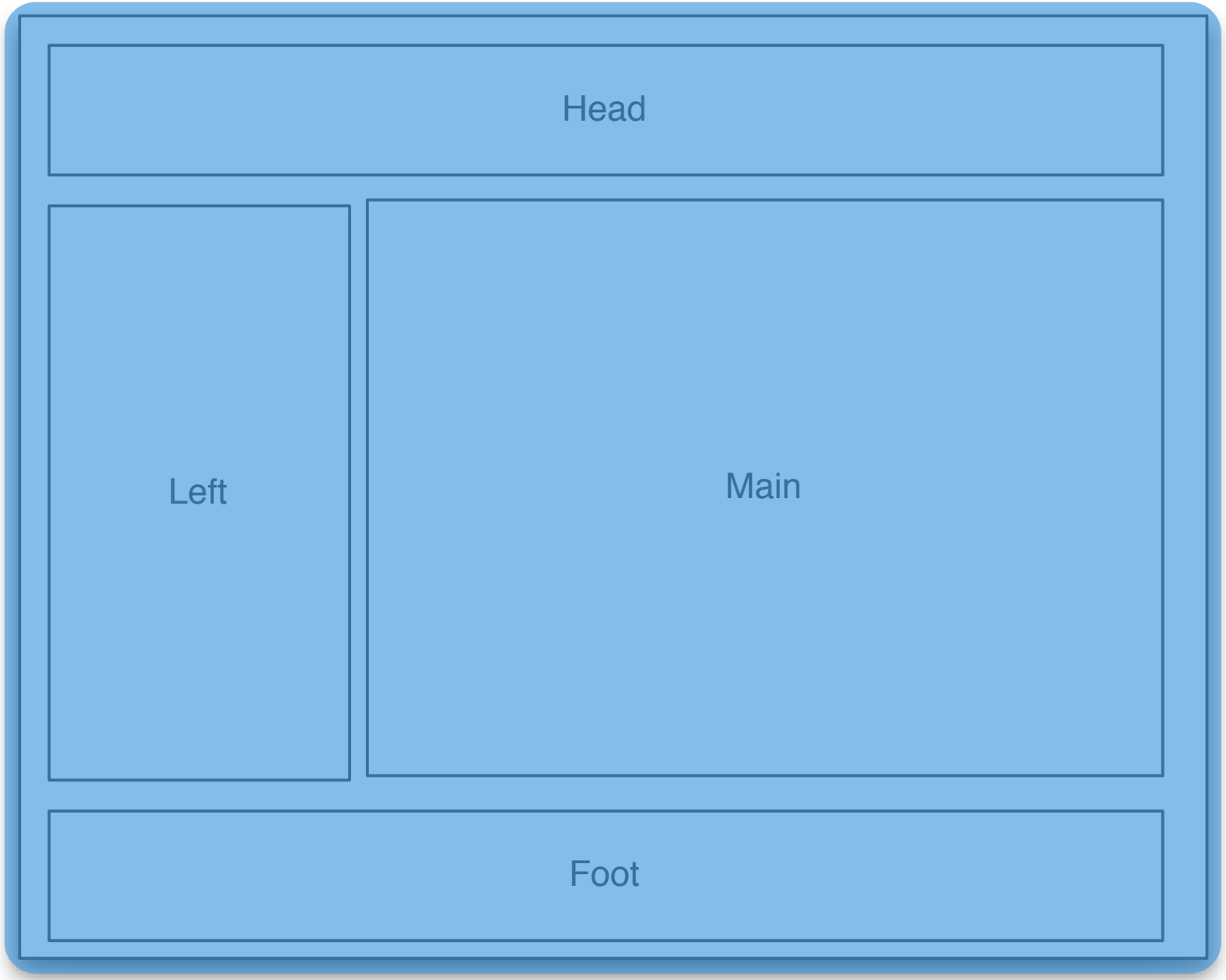
The core has many uncertainties

- Don’t force the specific theme layout.
- The core doesn’t know future’s extensions: breadcrumb list, additional javascript and so on.
- The core doesn’t know that the core should render for how many times, how many groups and how many categories.
- The core has to provide exchangable render-system. It doesn’t force the specific render-system.

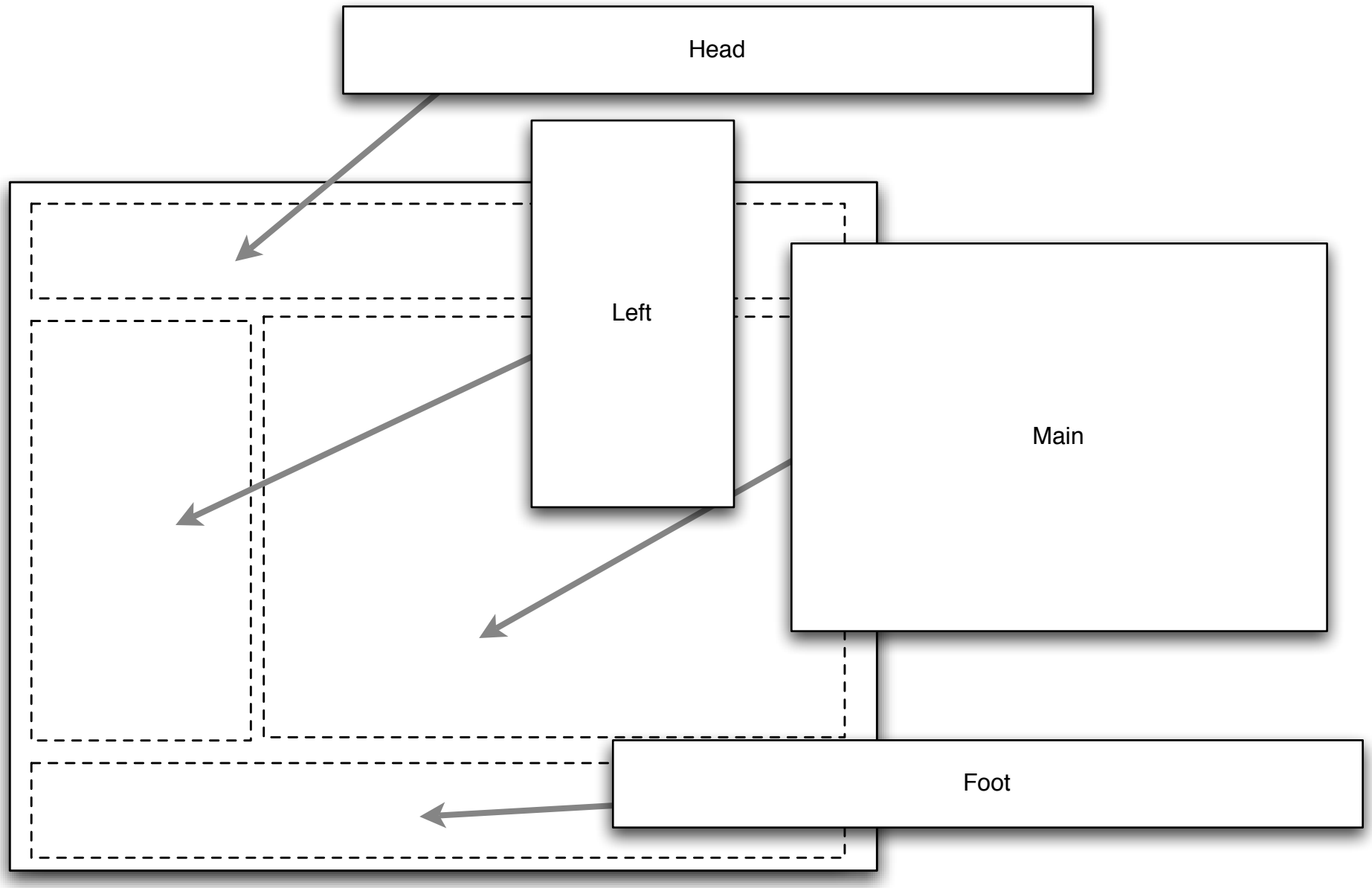
- XOOPS Cube got hints from real-time 3D rendering. And, XC 0.9 implemented the basic of those.
- This version implemented more better unity rendering sequence.
- I’m going to describe step by step:



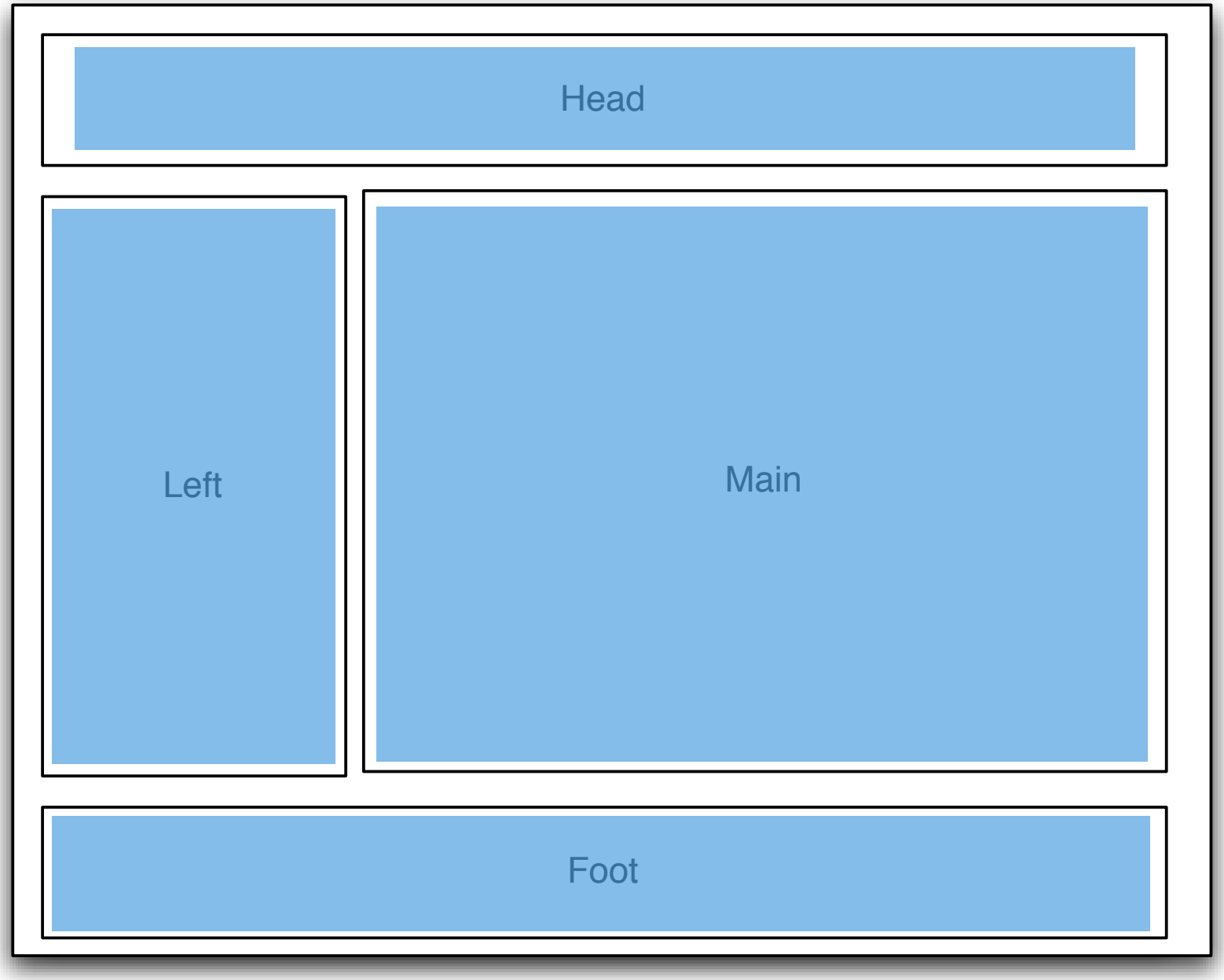
- XOOPS Cube doesn't force the specific theme layout.
- In other words, XOOPS Cube doesn't know the layout.



- But, it can trust that the final output is just one.

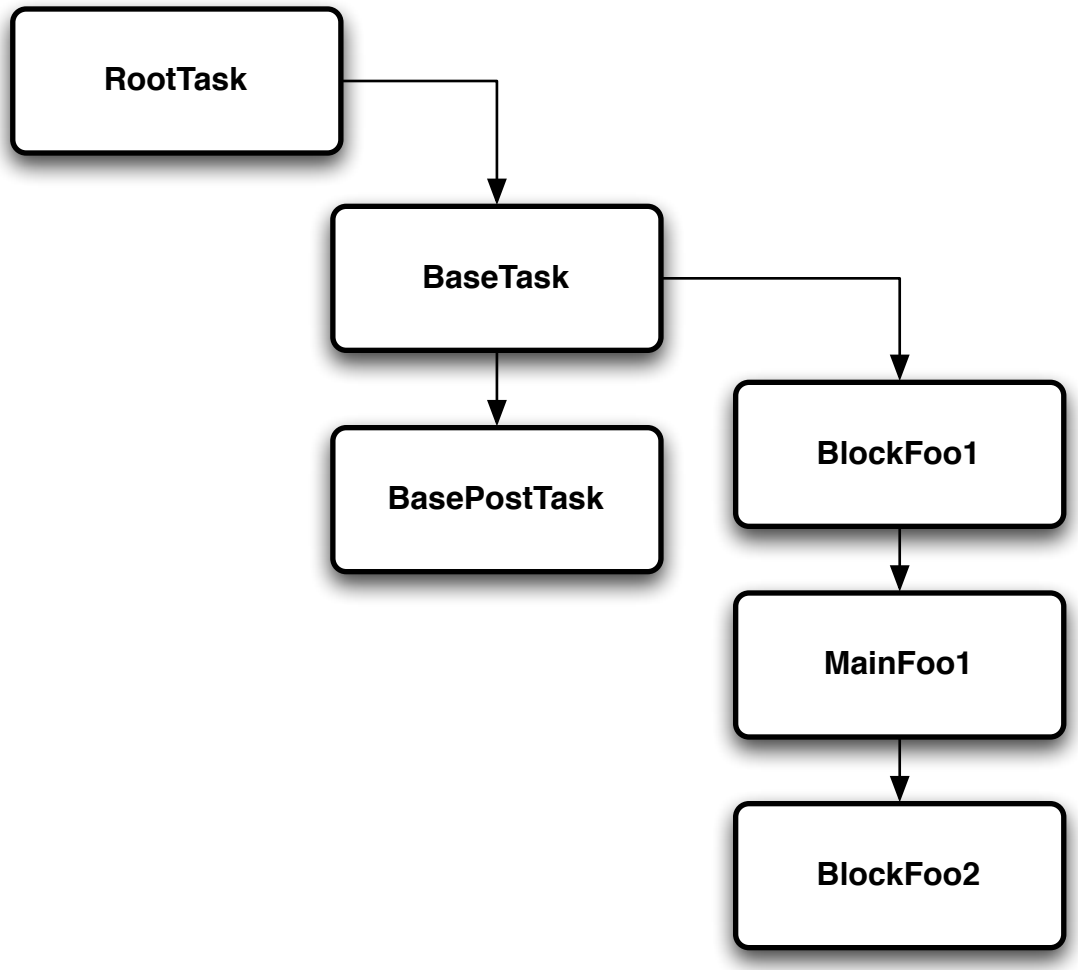


- The final output consists of sub output of sub parts.



- Fixedness: XOOPS Cube has to put “sub parts rendering” before final rendering.

- Render System is exchangable.
- Rendering Process is exchangable.
- But, tasks need the unity rendering sequence.
- Anyway, the final output is just 1. The core has to render sub parts as “material” before the final output.
- The final output is composite picture.



- But, the order of “Detachable & Combination Free” sequential process is **out of order** for rendering.

- Video games address the same challenge, but video games make custom-built rendering sequence for each title.

- XOOPS Cube applies the generic-purpose rendering-engine idea to this challenge.

```
function execute()
{
        $this->initialize();

        $dmy = null;
        $rootTask =& new XCube_Task("root", $dmy);

        $this->mController->buildTask($rootTask);

        $rootTask->initializeAll();
        $rootTask->updateAll();
        $rootTask->drawAll();

        //----------------------
        // Rendering Sequences
        //----------------------
        $collector =& new XCube_RenderOpCollection();
        $rootTask->acceptCollectorAll($collector);

        $visitor =& new XCube_RenderableVisitor();
        $collector->acceptVisitor($visitor);

        print $this->mRenderTargetScreen->getResult();
}
```

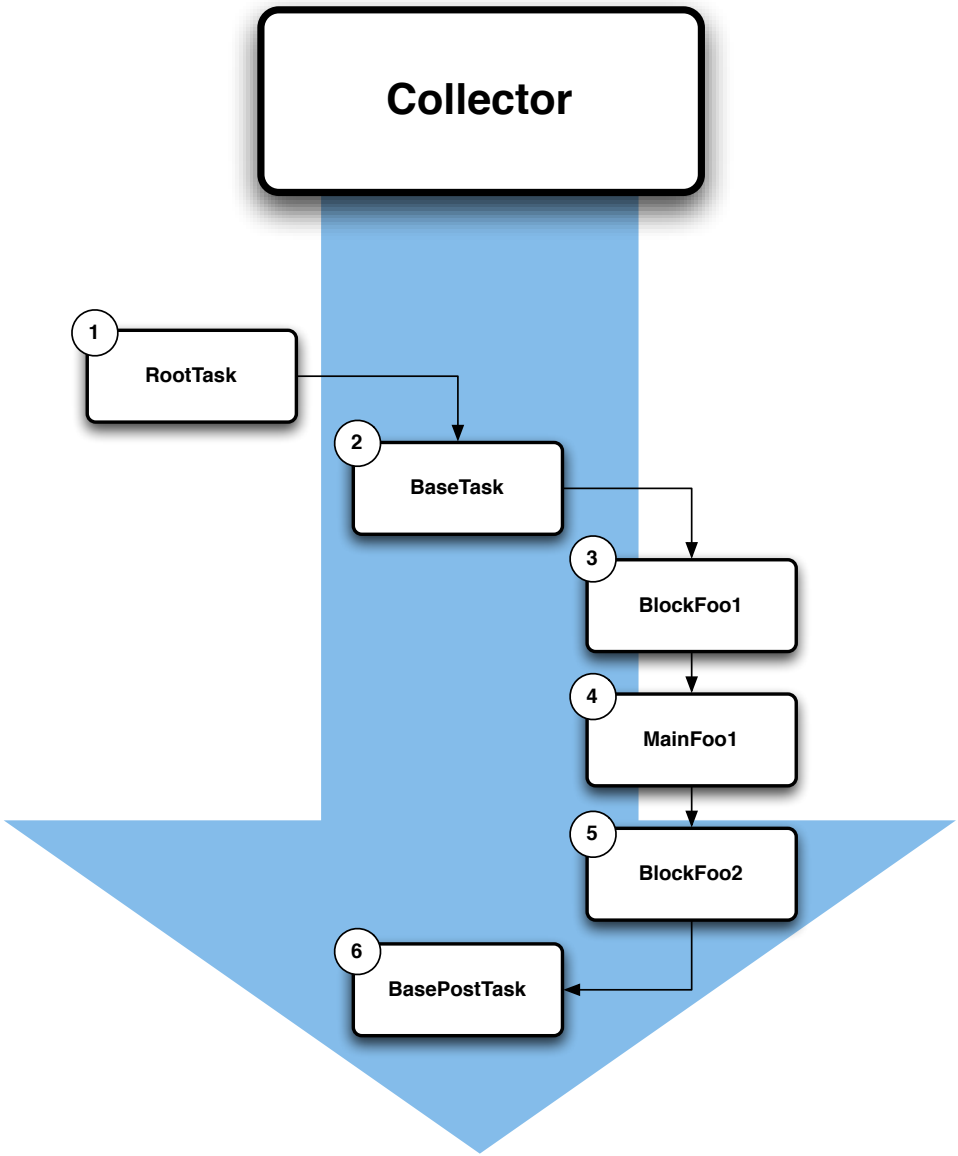
- I'm going to explain that part:

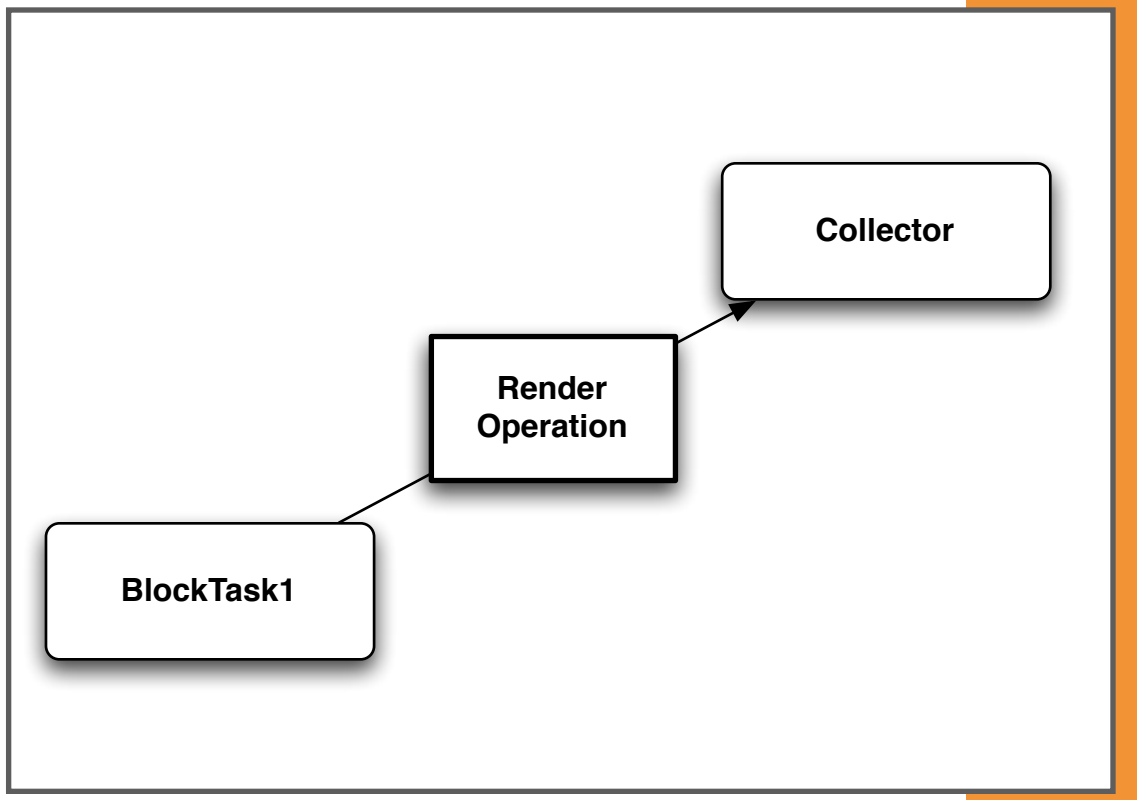
Overview

1. **Collector** collects **Render Operation** from tasks on the list.
2. **Collector** makes operations in-order.
3. Execute rendering. Because Collector class is final class, the core uses **Visitor** class that is extendable.

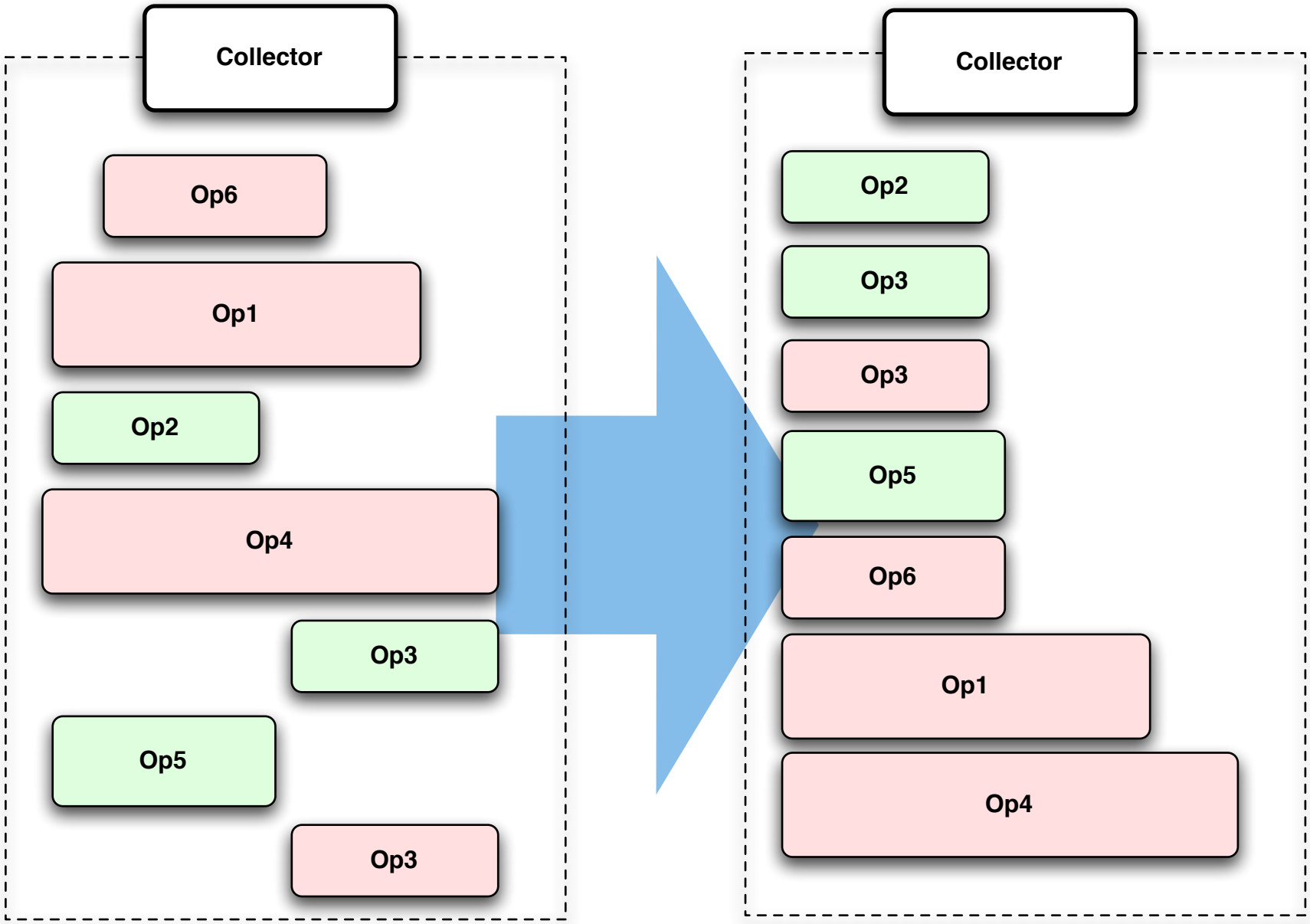
```
$collector =& new XCube_RenderOpCollection();
$rootTask->acceptCollectorAll($collector);
```

- Generates a collector. It makes a tour of tasks on the list.
- Collector class is final class. It's impossible to extend.
- In the future, Collector class will be optimized for performance. So you should not touch it.





- Tasks having a drawable element pass **Render Operation** to the **Collector**.



- **Corrector** sets right for **in-order**.

```php
$visitor =& new XCube_RenderableVisitor();
$collector->acceptVisitor($visitor);
```

- The collector prepared materials by fixed data type and fixed logic.
- You can not exchange those, but rendering is executed by exchangable **Visitor**.

- Collector/Visitor is good for variable amounts and unknown contents.
- BASE does not need to prepare rendering, cache preparations and recognize amounts.

```
$visitor =& new XCube_RenderableVisitor();
$collector->acceptVisitor($visitor);
```

P.S.

- Exchangable Visitor is fixed? A developer will modify this line by next version.

Attentions

- This version has only simple code for logic-test.
- Many lines are empty. So, review with this document.
- Better snapshot will be released next month.

- XOOPS Cube try to define re-use unit as a task of unity DRIVE MECHANISM.

- XOOPS Cube need re-use modules, not re-use libraries. It's really difficult, so I don't have other idea.

- I applied the new approach "A module (engine) for the unity control system that manges workload of hardware thread" to clear XOOPS Cube's difficult requirements.

http://research.cesa.or.jp/pdf/shiryo6-2-3.pdf

See you

Let's discuss at forums!