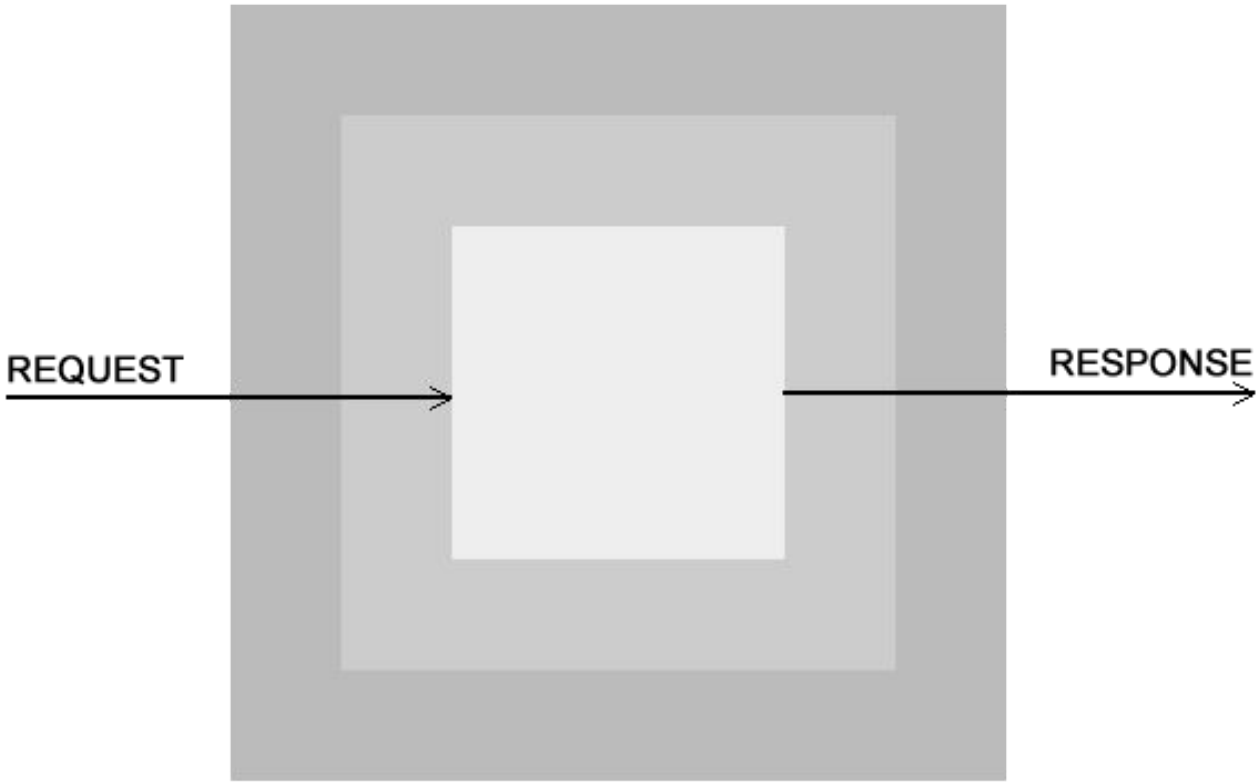
FROM REQUEST TO RESPONSE

How Drupal 8 Works

Deji Akala

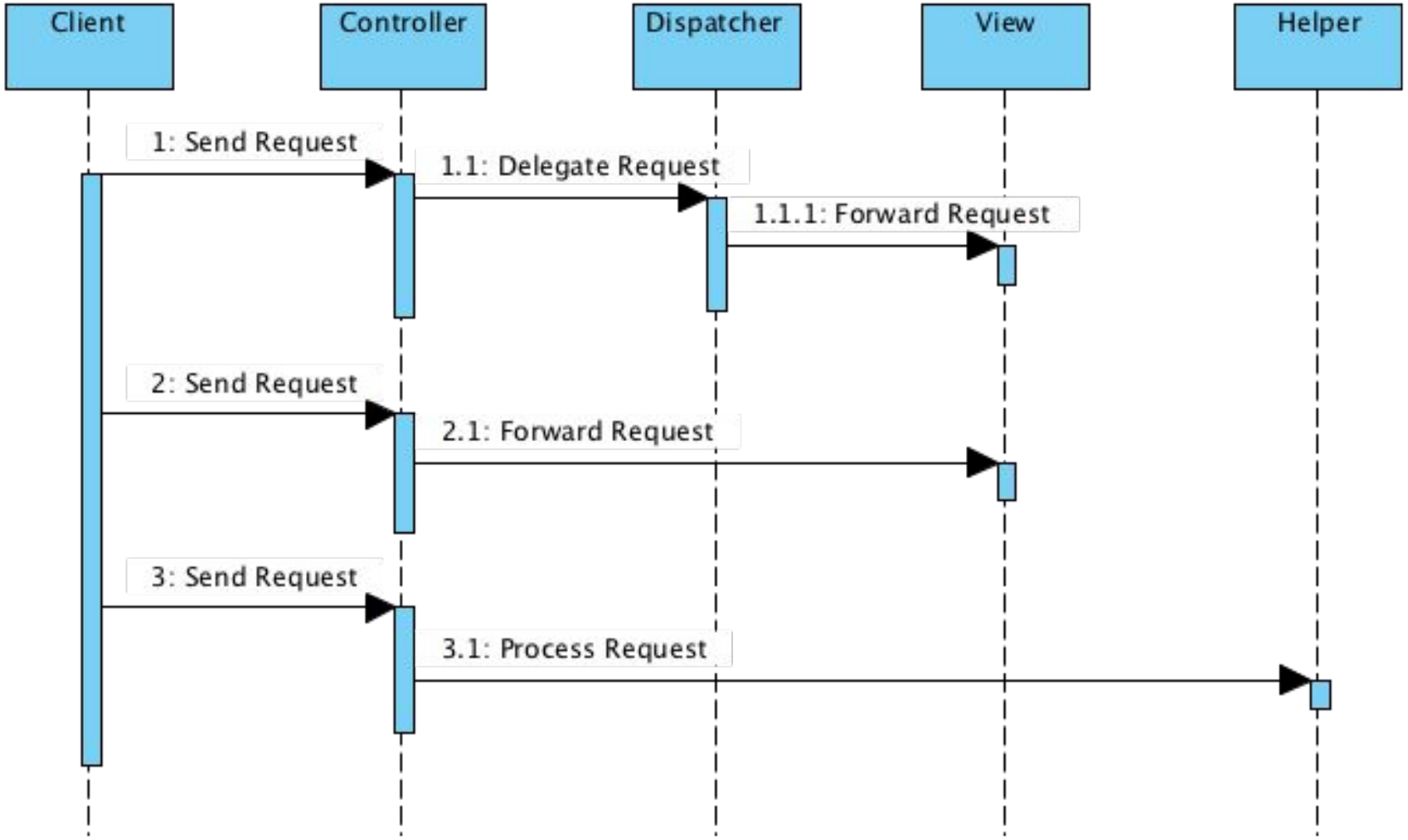
British Council, London

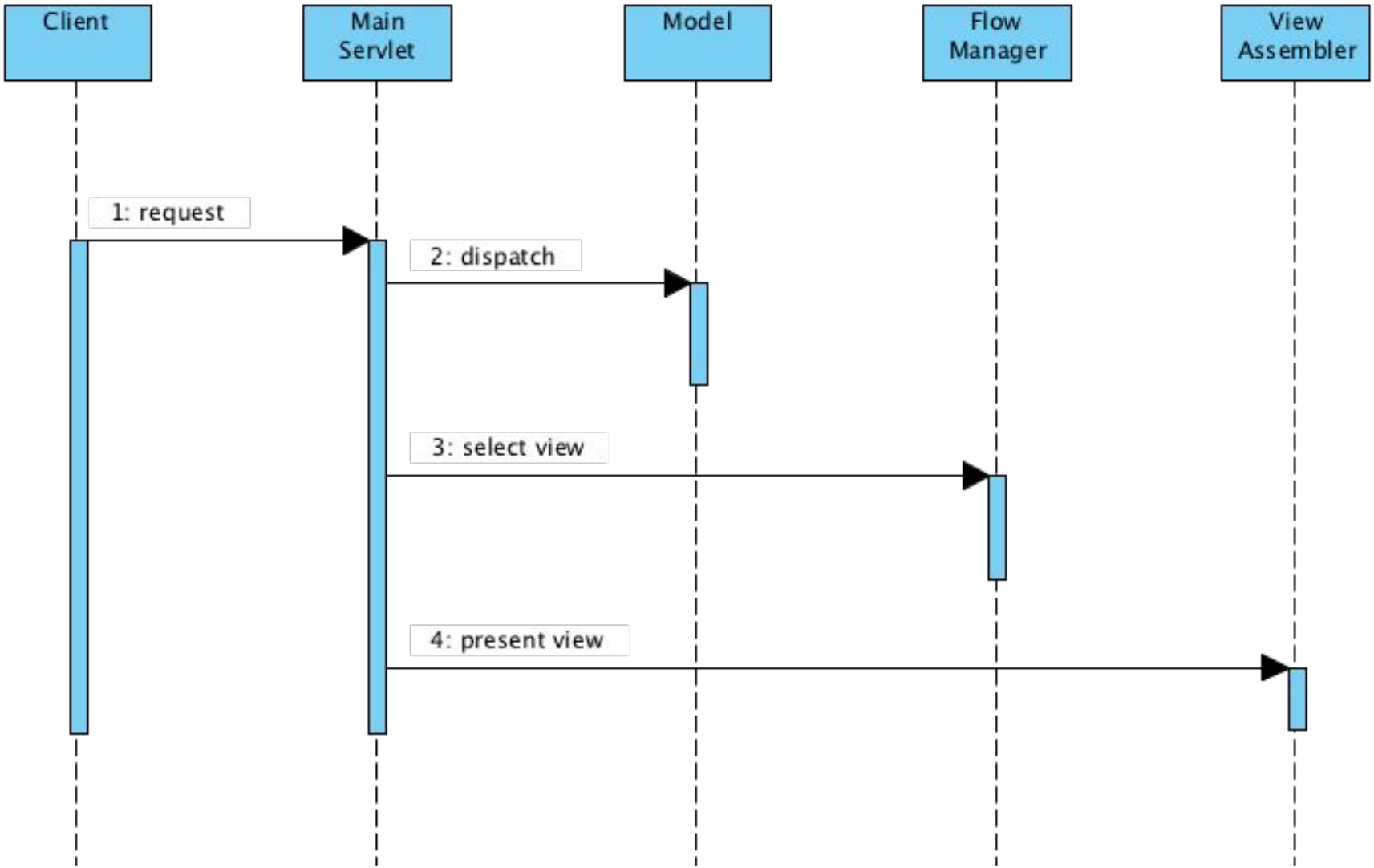

REQUEST
RESPONSE

FRONT CONTROLLER

A controller that handles all requests for a web site.

- *Martin Fowler*


Client
Controller
Dispatcher
View
Helper
1: Send Request
1.1: Delegate Request
1.1.1: Forward Request
2: Send Request
2.1: Forward Request
3: Send Request
3.1: Process Request


Client
Main Servlet
Model
Flow Manager
View Assembler
1: request
2: dispatch
3: select view
4: present view

```php
<?php

/**
 * @file
 * The PHP page that serves all page requests on a Drupal installation.
 *
 * All Drupal code is released under the GNU General Public License.
 * See COPYRIGHT.txt and LICENSE.txt files in the "core" directory.
 */

use Drupal\Core\DrupalKernel;
use Symfony\Component\HttpFoundation\Request;

$autoloader = require_once 'autoload.php';

$kernel = new DrupalKernel('prod', $autoloader);

$request = Request::createFromGlobals();
$response = $kernel->handle($request);
$response->send();

$kernel->terminate($request, $response);
```

DRUPAL + SYMFONY

```
use Drupal\Core\DrupalKernel;
use Symfony\Component\HttpFoundation\Request;
```

Namespaces allow unambiguous references to objects, functions or variables

```php
<?php

  $autoloader = require_once 'autoload.php';

  $kernel = new Drupal\Core\DrupalKernel('prod', $autoloader);

  $request = Symfony\Component\HttpFoundation\Request::createFromGlobals();
  $response = $kernel->handle($request);
  $response->send();

  $kernel->terminate($request, $response);
```

AUTOLOADING

```
$autoloader = require_once 'autoload.php';
```

```
require_once '/path/to/file1.php';
require_once '/path/to/file2.php';
require_once '/path/to/file3.php';
require_once '/path/to/file4.php';
require_once '/path/to/file5.php';
```

automatic + loading = autoloading

```
module_load_include('inc', 'system', 'system.admin');
```

PSR-4

A specification for autoloading classes from file paths

src\module\Entity

src\module\Controller

src\module\Plugin

src\module\Field

AUTOLOADING

```
$autoloader = require_once 'autoload.php';
```

```php
<?php

/**
 * @file
 * Includes the autoloader created by Composer.
 *
 * @see composer.json
 * @see index.php
 * @see core/install.php
 * @see core/rebuild.php
 * @see core/modules/statistics/statistics.php
 */

return require __DIR__ . '/vendor/autoload.php';
```

```php
<?php

// autoload.php @generated by Composer

require_once __DIR__ . '/composer/autoload_real.php';

return ComposerAutoloaderInitDrupal8::getLoader();
```

HTTPKERNEL COMPONENT

```
$kernel = new DrupalKernel('prod', $autoloader);
```

HTTPKERNEL COMPONENT

```
class DrupalKernel implements DrupalKernelInterface, TerminableInterface {

}


interface DrupalKernelInterface extends HttpKernelInterface, ContainerAwareInterface {

}
```

The HttpKernel component provides a structured process for converting a Request into a Response by making use of the EventDispatcher component.

- *Symfony documentation*

```php
<?php
namespace Symfony\Component\HttpKernel;

use Symfony\Component\HttpFoundation\Request;
use Symfony\Component\HttpFoundation\Response;

/**
 * HttpKernelInterface handles a Request to convert it to a Response.
 *
 * @author Fabien Potencier fabien@symfony.com
 */
interface HttpKernelInterface
{
  const MASTER_REQUEST = 1;
  const SUB_REQUEST = 2;

  /**
   * Handles a Request to convert it to a Response.
   *
   * When $catch is true, the implementation must catch all exceptions
   * and do its best to convert them to a Response instance.
   *
   * @param Request $request A Request instance
   * @param int     $type    The type of the request
   *    (one of HttpKernelInterface::MASTER_REQUEST or HttpKernelInterface::SUB_REQUEST)
   * @param bool    $catch   Whether to catch exceptions or not
   *
   * @return Response A Response instance
   *
   * @throws \Exception When an Exception occurs during processing
   */
  public function handle(Request $request, $type = self::MASTER_REQUEST, $catch = true);

 }
```

REQUEST

```
$request = Request::createFromGlobals();
```

$_SERVER

$_GET

$_POST

$_FILES

$_COOKIE

$_SESSION

$_REQUEST

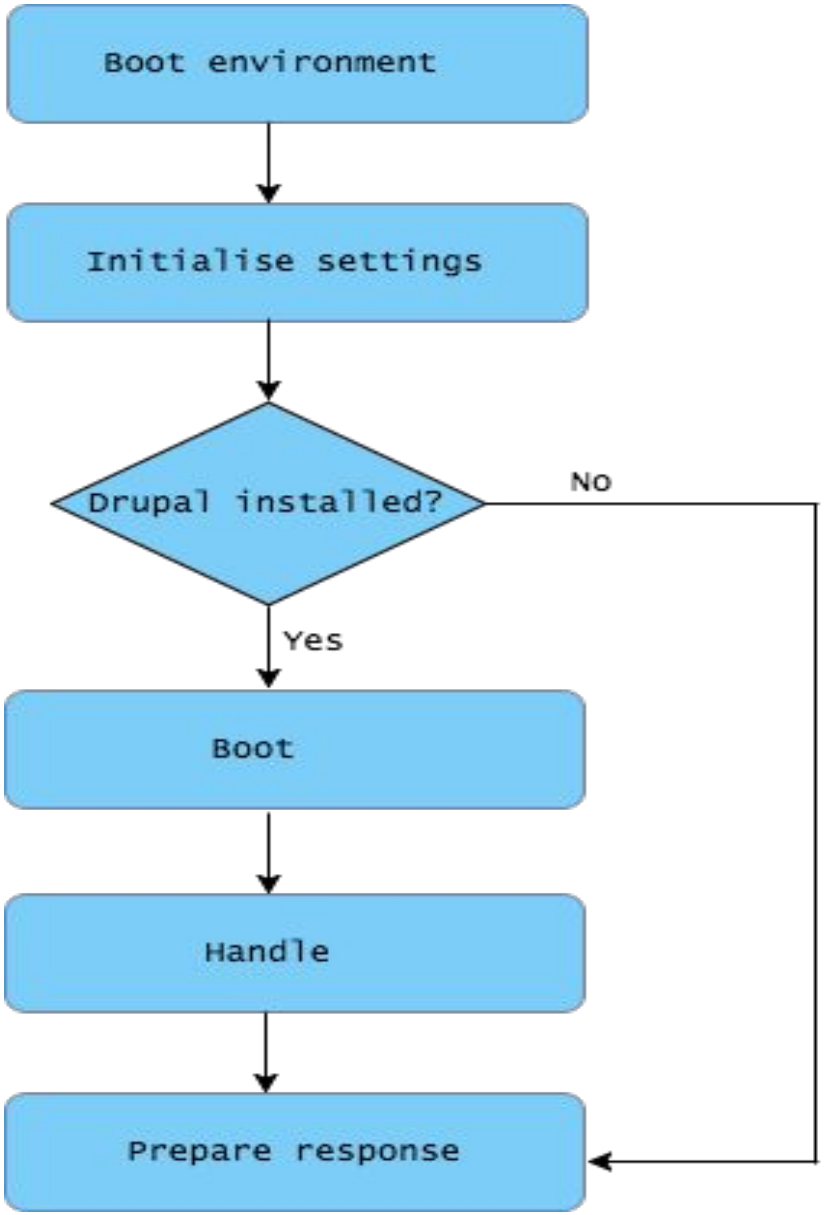
$_ENV

RESPONSE

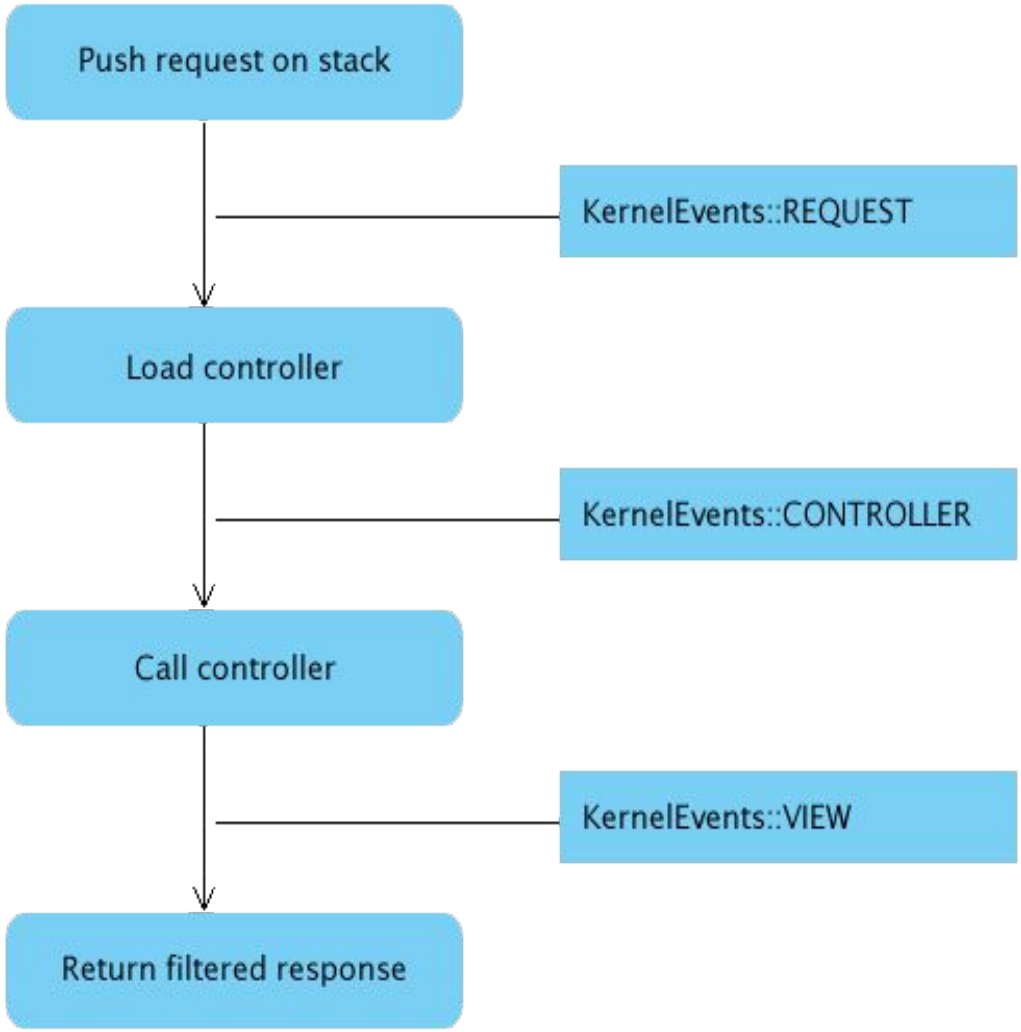
```
$response = $kernel->handle($request);
```

```php
<?php

namespace Symfony\Component\HttpKernel;

use Symfony\Component\HttpFoundation\Request;
use Symfony\Component\HttpFoundation\Response;

/**
 * HttpKernelInterface handles a Request to convert it to a Response.
 *
 * @author Fabien Potencier fabien@symfony.com
 */
interface HttpKernelInterface
{
  const MASTER_REQUEST = 1;
  const SUB_REQUEST = 2;

  /**
   * Handles a Request to convert it to a Response.
   *
   * When $catch is true, the implementation must catch all exceptions
   * and do its best to convert them to a Response instance.
   *
   * @param Request $request A Request instance
   * @param int     $type    The type of the request
   *    (one of HttpKernelInterface::MASTER_REQUEST or HttpKernelInterface::SUB_REQUEST)
   * @param bool    $catch   Whether to catch exceptions or not
   *
   * @return Response A Response instance
   *
   * @throws \Exception When an Exception occurs during processing
   */
  public function handle(Request $request, $type = self::MASTER_REQUEST, $catch = true);

}
```

DRUPAL HANDLE() METHOD



SYMFONY HANDLE() METHOD



OUTPUT

```
$response->send();
```

sendHeaders()

– headers, status, code, cookies

sendContent()

- response of valid type
- string, number, null or object that implement a __toString() method and format
- HTML, JSON, XML, TXT etc

POST RESPONSE

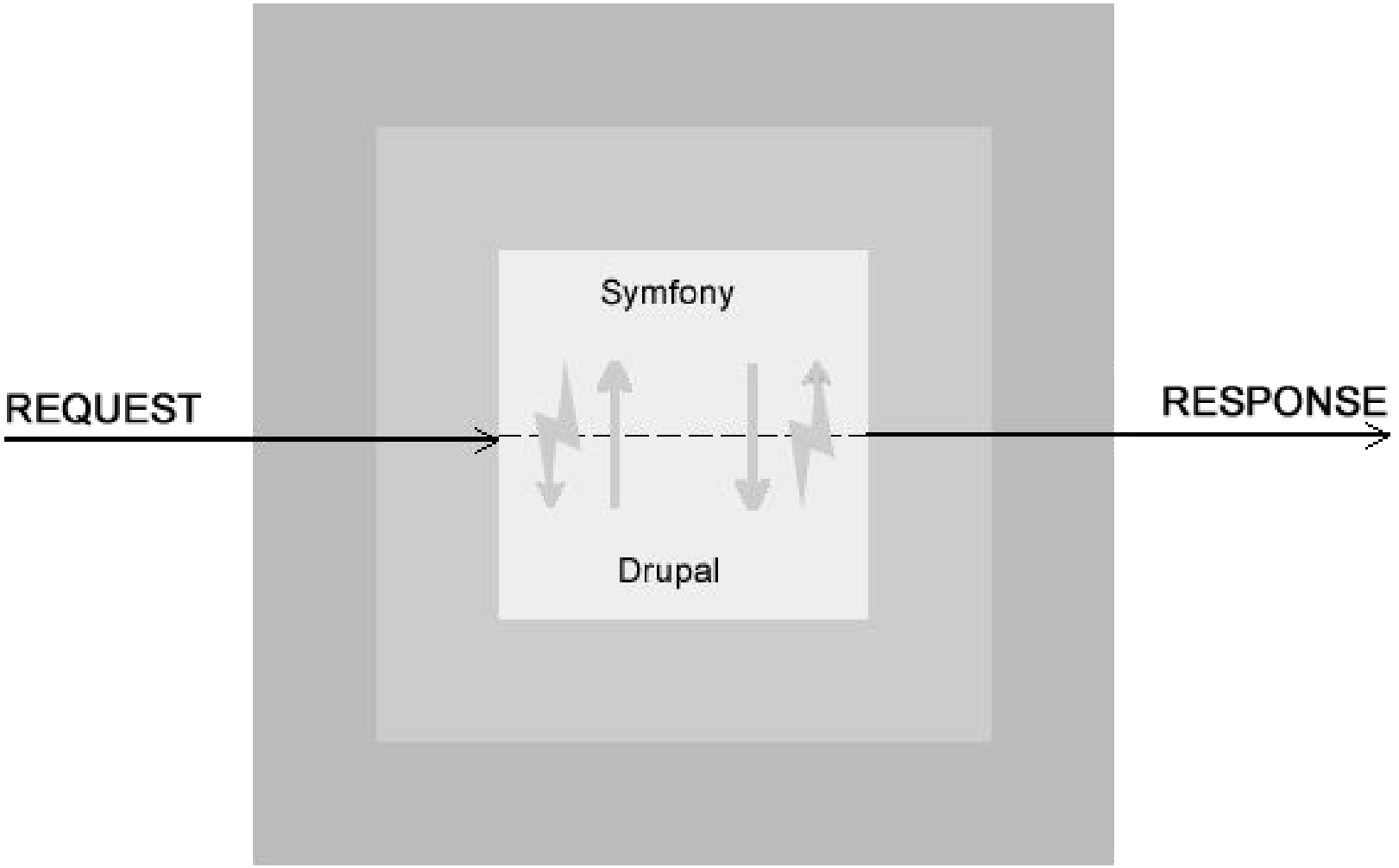
```
$kernel->terminate($request, $response);
```

```
public function terminate(Request $request, Response $response)
{
    $this->dispatcher->dispatch(
                KernelEvents::TERMINATE,
                new PostResponseEvent($this, $request, $response)
            );
 }
```

```php
// Performs end-of-request tasks.
function drupal_page_footer() {
  global $user;

  module_invoke_all('exit');

  // Commit the user session, if needed.
  drupal_session_commit();

  if (variable_get('cache', 0) && ($cache = drupal_page_set_cache())) {
    drupal_serve_page_from_cache($cache);
  } else {
    ob_flush();
  }

  _registry_check_code(REGISTRY_WRITE_LOOKUP_CACHE);
  drupal_cache_system_paths();
  module_implements_write_cache();
  drupal_file_scan_write_cache();
  system_run_automated_cron();
}
```

KernelEvents::TERMINATE

UserRequestSubscriber::onKernelTerminate() - update the current user's last access time (300)
PathSubscriber::onKernelTerminate() – cache system paths (200)
AutomatedCron::onTerminate() – run automated cron if enabled (100)
RequestCloseSubscriber::onTerminate() – write caches (100) – drupal_page_footer()
KernelDestructionSubscriber::onKernelTerminate() – destroy initialized services (-100)


Symfony
REQUEST
RESPONSE
Drupal

@dejiakala
http://www.github.com/dakala
dejiakala@gmail.com

https://joind.in/talk/6e65c