eZ Unconf #2

eZ Publish 5.1 Introduction

eZ UnConf
Montpellier
2013

Florent Huck – florent.huck@ez.no
Philippe Vincent-Royol – philippe.vincent-royol@ez.no

Agenda

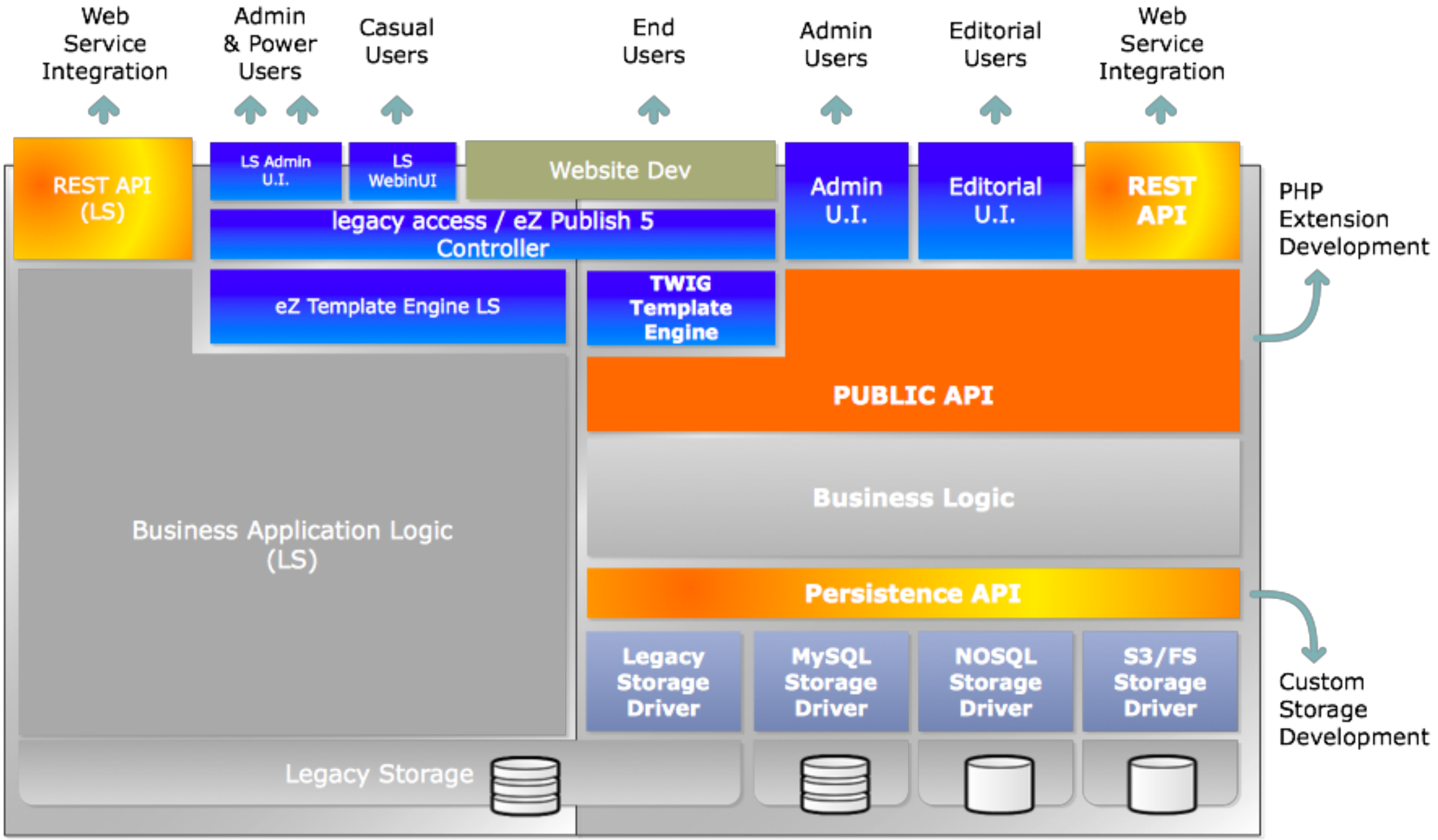
- **Welcome to eZ Publish**
  - Dual Core
  - What's change between eZ 4, 5.0, 5.1 ?
- **That's one small step for a developper ...**
  - Bundle eZ 5 full stack
  - Multisite with one repository
  - Templates override
  - Deeper in source code
- **... one giant leap for production !**
  - Cache introduction
  - Http Cache
  - Varnish configuration

Welcome to eZ Publish

eZ Publish 5 – Wall of Fame

- **Simple Integration with our API**
- **HMVC** (Hierarchical Model View Controller) stack
- **Decoupled Composents**
- **Dependency Injection**
- **New Template Engine**
- Extensible, Open, Reliable
- **Backward Compatibility**

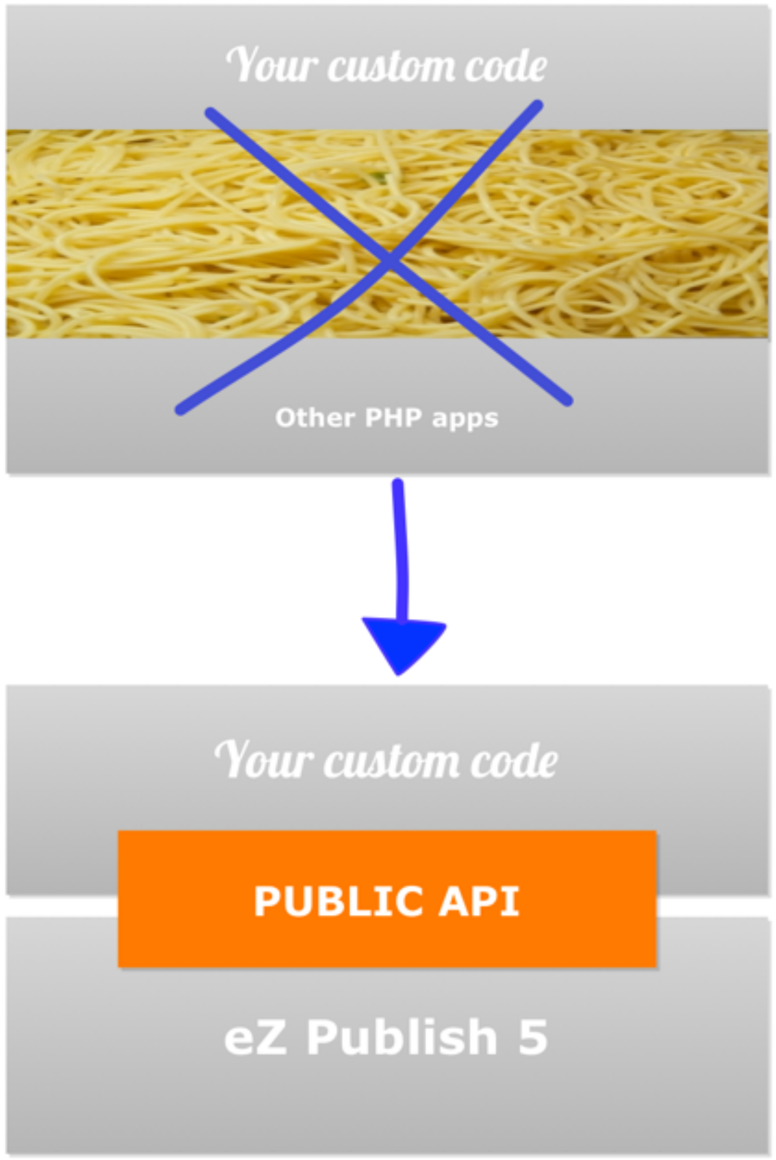
eZ Publish 5 - Architecture



eZ Publish Public API

For the first time ever, a Public API for eZ Publish: **a major step forward**!

- **All the CMS features** available from PHP code
  - easier to develop import/export scripts
  - easier to add new content mgmt interfaces
- A clear API definition, which means **better documentation** for developers
- Allows refactoring of core code for improved performance, scalability, etc… while having
- **long term backward compatibility** of your custom developments
- **Fully extensible**, extend it with your custom features relying on the same framework



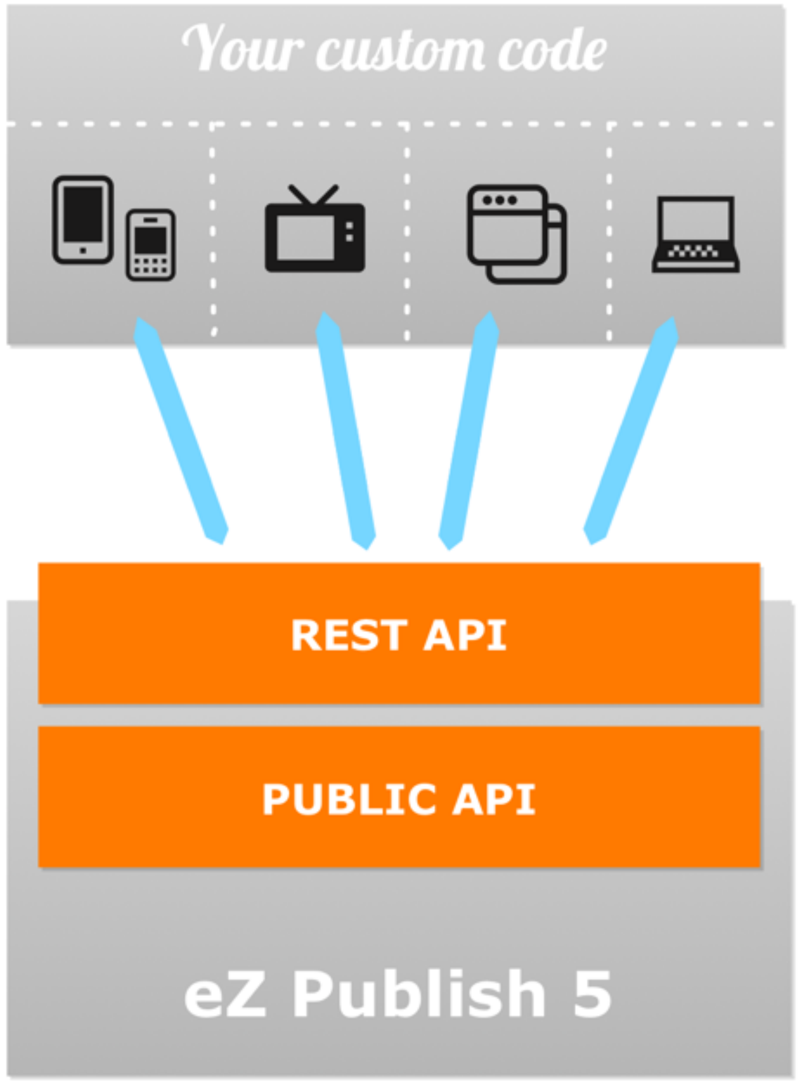
eZ Publish Rest API v. 2

With version 5, eZ Publish REST API covers **the whole scope of the content management engine** and not only a limited part (read only)

The REST API fully relies now on the Public API for **better maintainability** and **simplicity**

It is the reliable and maintainable for:

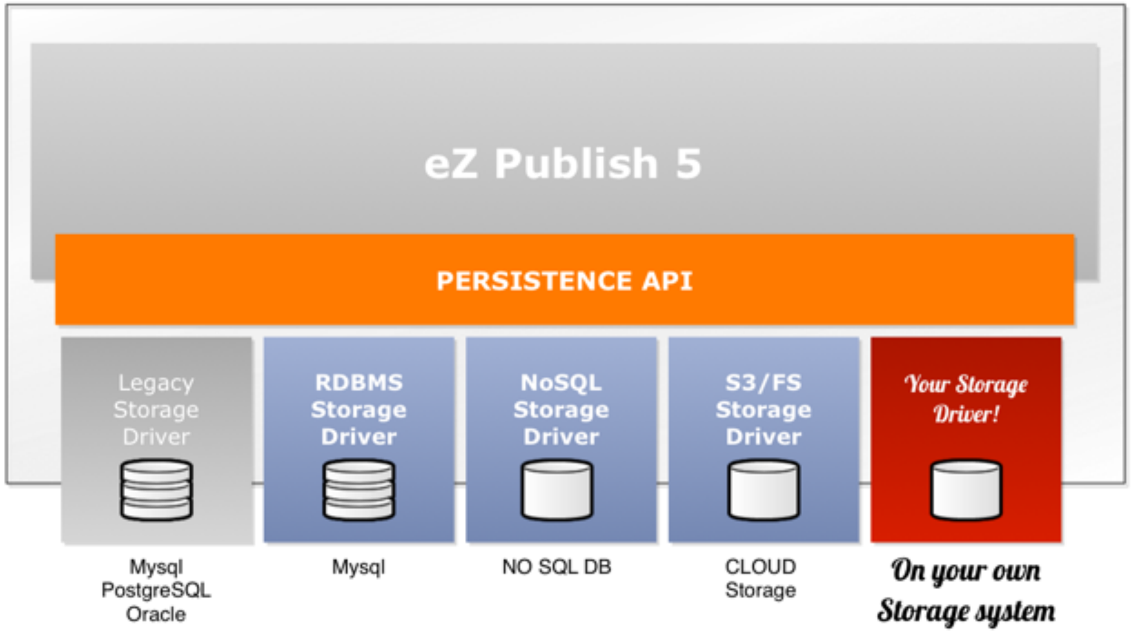
- **Native mobile applications**
- **Integration in existing business applications**
- **Social Media Integrations**
- Any other application (desktop application, internet TV apps…)



Handle big data content repositories

» eZ Publish 5 is the first content management platform designed to handle **Big Content & Big Data**!

- A clear goal to provide **Relational, NoSQL and Hybrid** storage system
- **A persistence API** enabling (eZ or 3rd party) to develop the support of any storage system
- **Long term backward compatibility** of your custom developments
- **Bridging** eZ Publish 4.x storage system for backward compatibility with legacy



Symfony2, a new Web framework for eZ Publish

Benefits & Opportunities

» eZ Publish 5 integrates the **Symfony 2** framework. What does this mean?

A whole new framework available for developers

- Extensible HMVC structure
- Clean and maintainable modularity via native dependency injection
- Easier to merge your custom apps with eZ Publish
- A wide range of existing resources on top of Symfony

A new template engine: Twig

- Enhanced rendering speed
- Easier and more extensible templating language
- Support of hierarchical requests
- Better IDE support

Symfony2, a new Web framework for eZ Publish

Advantages & Disaventages

Advantages

- **Knowledgeable Community**
- **Standardization**
  - Time saving
  - Re-usability of code
  - Re-usability of knowledge
- **Reduced maintenance costs** at the component level
- **Faster introduction** of new features
- **Faster development** of items such as alternate database options
- **Faster and more efficient kernel**

Disavantages

- New development APIs to learn
- Porting of legacy code
- Transition period
- No more excuses not to do your work

What’s change ?

eZ Publish Evolution

What Changes?

Legacy system :

is an old method, technology, computer system, or application program. The legacy system may or may not remain in use. Even if it is no longer used, it may continue to impact the organization due to its historical role.

(Wikipedia)

Changes

- Online resources
- File structure
- API
- Terminology
- Feature implementation

File structure

/ez5	
ezpublish	configurations, cache, logs
ezpublish_legacy	the full Legacy Stack
…	incl. legacy extensions
src	your bundles
vendor	libraries from 3rd parties
ezystems	Kernel and 3 bundles (core, legacy, rest)
web	all files which are accessible from the web

- Static assets are either copied or symlinked from legacy designs / extensions into the *web* directory via a cli script
- Different rewrite rules are needed
- Per-environment configuration is managed via different frontend controllers in the *web* directory

Terminology

Translation 4.x to 5.x – content model

4.x	5.X
Content Class	ContentType
Content Class Group	ContentTypeGroup
Datatype	FieldType
Node	Location
Content Object	Content (meta info in ContentInfo)
Content Object Version	VersionInfo
Content Object Attribute	Field + FieldValue
Content Class Attribute	FieldDefinition

Features

Translation 4.x to 5.x – feature implementation

4.x	5.X
eZ Publish Template	Twig Template
Extension	Bundle
Module	Controller class (based on eZ\Bundle\EzPublishCoreBundle \Controller)
View	Controller action
Fetch function	Replaced by controller+view, the template "fetch" function is replaced by "render"
Template operator	Filter + function
settings/	ezpublish/config and Resources/config on each bundles
site.ini	ezpublish.yml

What Changes

Feature translation 4.x to 5.x – all the rest

As of 5.0, some features are not implemented on the new stack, such as:

- Admin Interface; content editing
- Cluster Mode **(works now with eZ 5.1+)**
- Information Collectors
- Workflows
- Notifications
- eZ Find
- eZ Flow (display of «layout» attribute) (**5.1+ includes read part**)
- Multisite with a repository **(works now with eZ 5.1+)**

Documentation is an effort under way.

You can participate too!

Online resources

- Issue Tracker: http://jira.ez.no
- Developer documentation: https://confluence.ez.no/display/EZP/eZ+Publish+Documentation
- API docs:
  - http://ezsystems.github.com/ezpublish-kernel/phpDocumentor/
  - http://pubsvn.ez.no/
  - Also the *doc* folder inside *vendor/ezsystems/ezpublish*
- Source code:
  - https://github.com/ezsystems/ezpublish-legacy => the 4.x code, a.k.a. **Legacy Stack**
  - https://github.com/ezsystems/ezpublish-kernel => the new kernel (incl. Public API)
  - https://github.com/ezsystems/ezpublish-community => pulling it all together
- Tutorials: http://share.ez.no as always
- eZ 5 CheatSheet https://github.com/dspe/ez5_cheatsheet

That's one small step for a developper ...

eZ Publish 5 – Development mode

- Activated via a custom controller in the new kernel
  - Debug toolbar at bottom of the screen – always in view
  - Full featured profiler
  - Saves data of every profiler run
  - Exception handling gets stricter

- Easier to keep separate settings per-environment

- DebugOutput still available when running legacy modules

More customizable in 5.2+ and 2013.5. Have a look at https://github.com/ezsystems/ezpublish-community/pull/51

eZ Publish 5 configuration model

The eZ 5's configuration model uses Symfony2's configuration model : **YML files**

A YML file contains settings that control behavior of a specific part of eZ Publish

- **ezpublish.yml is** the most important YML file
- **config.yml** to configure some behavior with Symfony2 for example
- Etc …

Settings are managed **manually** (by editing the YML file with a text editor)

eZ Publish Dynamic Configuration

Dynamic configuration with the **ConfigResolver** can be resolved depending on a **scope**.

Available scopes are (in order of precedence)

• *Global:*	ezsettings.*global*.field_templates
• *Siteaccess:*	ezsettings.*ezdemo_site*.field_templates
• *Group:*	ezsettings.*my_siteaccess_group*.field_templates
• *Default:*	ezsettings.*default*.field_templates

Your dynamic settings must comply internally to the following name format:

<namespace>.<scope>.parameter.name

Dynamic configuration example

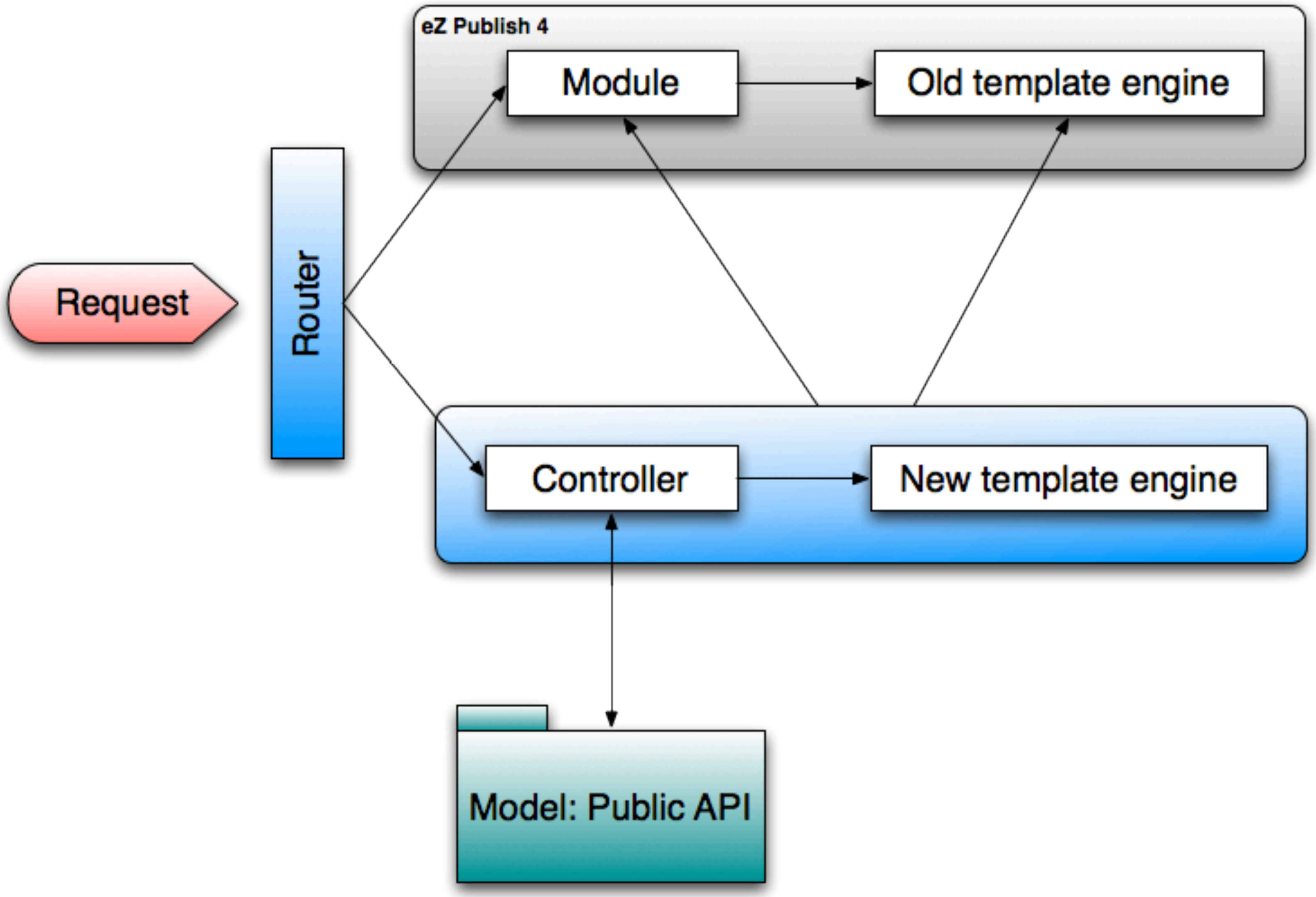
1. eZ Publish app configuration - YML configuration : **ezpublish.yml**

```yaml
ezpublish:
    system:
        ezspace:
            field_templates:
                - { template: EztTrainingBundle:FieldType:content_fields.html.twig, priority: 0 }
```

2. Bundle configuration - Syntactic configuration : **parameters.yml**

```yaml
parameters:
    ezsettings.ezspace.field_templates :
            - { template: EztTrainingBundle:FieldType:content_fields.html.twig, priority: 0 }
```

eZ Publish 5 – BC



Fallback Routing & Content view

- All URIs wich are not matched by Symfony controllers are sent to the Legacy Stack
  - All existing functionality is thus preserved
  - Including custom modules
  - Including url-aliases
  - For speed, this can be optimized via configuration (f.e. for Admin Interface)
  - Sf controller in use: LegacyKernelController::indexAction

- Content is rendered using the ViewController ::viewLocation controller
  - Class located in **Core\MVC\Symfony\Controller\Content**
  - It uses a twig template for pagelayout: pagelayout.html.twig in DemoBundle (set in parameters.yml)
  - And a ViewManager to find the appropriate template for the current Location. We will use the **template selection rules**.

Legacy template fallback

When falling back to the legacy kernel, the old **content/view** module is run to return the appropriate view for the given content.

However, the pagelayout is not rendered as it needs to be still rendered by Twig in the Symfony part, for consistency. In this regard, the system uses the **Decorator design pattern** to include the generated view in your layout.

```
parameters:
   # Base layout for legacy fallback
   ezpublish_legacy.my_siteaccess.view_default_layout: AcmeDemoBundle::pagelayout.html.twig

   # Block name
   ezpublish.content_view.content_block_name: content

   # Module Layout
   ezpublish_legacy.my_siteaccess.module_default_layout: AcmeDemoBundle::pagelayout_legacy.html.twig
```

Pagelayout example

Example of pagelayout.html.twig

```
<!DOCTYPE html>
<html>
  <head>
    <!-- Insert everything you need here that is common -->
  </head>
  <body>
    {% block content %}{# Regular content will be inserted here #}{% endblock %}
  </body>
</html>
```

Example of pagelayout_legacy.html.twig

```
{% extends "AcmeDemoBundle::pagelayout.html.twig" %}

{% block content %}
  {# module_result variable is received from the legacy controller. #}
  {{ module_result.content|raw }}
{% endblock %}
```

Some usefull tools

- **Display a location**

```
{{ render( controller("ez_content:viewLocation", { "locationId": 123, "viewType": "full" }  )) }}
```

- **Display a field**

```
{{ ez_render_field( content, "my_field" ) }}
```

- **Include a legacy template**

```
{% include "design:my_folder/my_template.tpl" with {"my_key" : "my_value" } %}
```

Some usefull twig helper

Property	Description
ezpublish.siteaccess	Returns the current siteaccess
ezpublish.requestUriString	Returns the request URI string
ezpublish.systemUriString	Return the systems URI string
ezpublish.viewParameters	Return the view parameters as a hash
ezpublish.viewParametersString	Return the view parameters as a string
ezpublish.siteName	Return the name of the current siteaccess
ezpublish.legacy	Returns legacy information

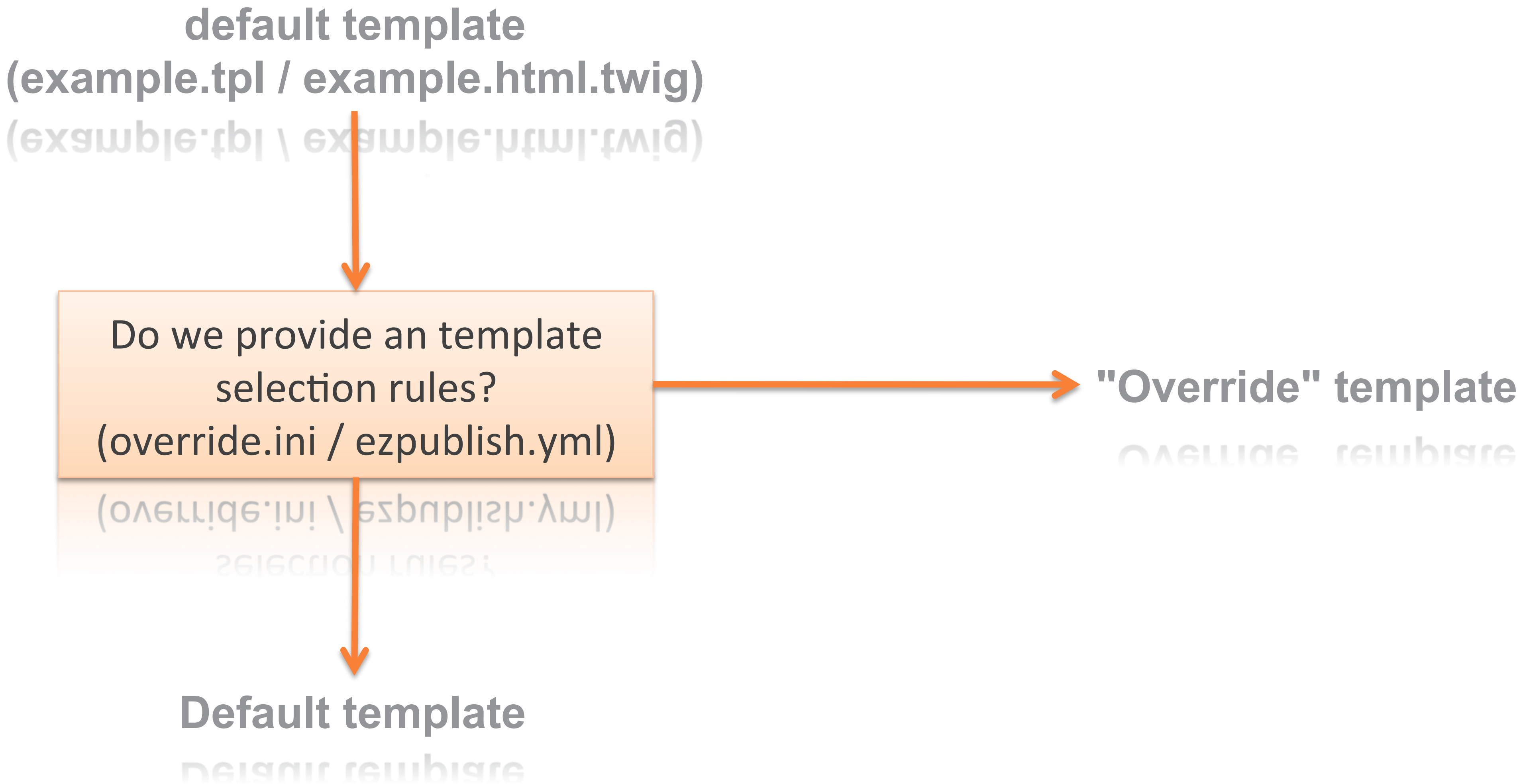
Legacy

Property	Description
ezpublish.legacy.all	Return all parameters
ezpublish.legacy.keys	Returns the parameter keys only
ezpublish.legacy.get	Returns a parameter by name
ezpublish.legacy.has	Returns true if the parameter is defined

Template selection rules system

Simplified view

Of course, eZ Publish is a (little) bit more complex but you could remember that



default template
(example.tpl / example.html.twig)
Do we provide an template selection rules?
(override.ini / ezpublish.yml)
"Override" template
Default template

Template selection rules - Configuration

Example from ezpublish.yml

```
ezpublish:
    system:
        my_siteaccess:
            location_view:
                full:
                    welcome_page:
                        template: AcmeDemoBundle:full:welcome.html.twig
                        match:
                            Id\Location: 2
                    article_full:
                        template: AcmeDemoBundle:full:article.html.twig
                        match:
                            Identifier\ContentType: article
                preview:
                    article_preview:
                        template: AcmeDemoBundle:preview:article.html.twig
                        match:
                            Identifier\ContentType: [article, news]
```

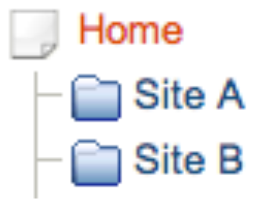
location_view and **content_view** are available in override system

Multisite – One repository

eZ Publish 5.1 provide a way to build **multiple website** with a single **repository**, in different subtrees of content. This system will be extends on next releases.

```
ezpublish:
    system:
        my_siteaccess:
            content:
                tree_root:
                    # Root locationId. Default is top locationId
                    location_id: 123
                    # Every URL aliases starting with those prefixes will be
                    # considered being outside of the subtree starting at
                    # root_location.
                    # Default value is an empty array.
                    # Prefixes are not case sensitive.
                    excluded_uri_prefixes:
                        - media
                        - users
```

Content structure



Bonus

Coding Standard !

Coding Standard

Be awesome, use coding standards

When contributing code to eZ Publish, you must follow **its coding standards**.

Remember that the main advantage of standards is that every piece of code **looks and feels familiar**, it's not about this or that being more readable.

- Use **PHP Code Sniffer** !
- Install **eZ v2 PHPCodeSniffer standard** (http://github.com/ezsystems/ezcs)

https://github.com/ezsystems/ezpublish-kernel/wiki/codingstandards

Exercise

Installation & our first bundle !

... one giant leap for production !

Cache - Reminder

Cache me if you can !

The goal of caching in HTTP/1.1 is to eliminate the need to send requests in many cases, and to eliminate the need to send full responses in many other cases

Types of caches

- **Browser caches**: the browser cache is a private cache as cached resources **aren't shared with anyone else**
- **Proxy caches**: a proxy is a shared cache as many people can be behind a single one
- **Gateway caches**: (a.k.a. Reverse Proxy) like a proxy, it's also a shared cache but **on the server side**

More informations about HTTP Rfc: http://tools.ietf.org/html/draft-ietf-httpbis-p6-cache-21

Reminder: HTTP caching

Caching headers

HTTP 1.1 specifies multiple response cache headers that can be used to control caching behavior:

• Cache-Control	=> can control public/private as well as TTL
• Expires, Max-age	=> sets a TTL for the given resource
• Etag	=> used to revalidate a resource which is already in a cache (note: E-tag is removed on eZ 5.1)
• Last-Modified	=> used to calculate a default TTL if no expiration date is set or as weak validator

Side note: compression of content is also a good idea to make web pages load faster

- But take care about gateway caches having to store *N* versions of each resource

eZ Systems AS - Florent Huck & Philippe Vincent-Royol

Reminder: HTTP caching
Validation vs. Expiration

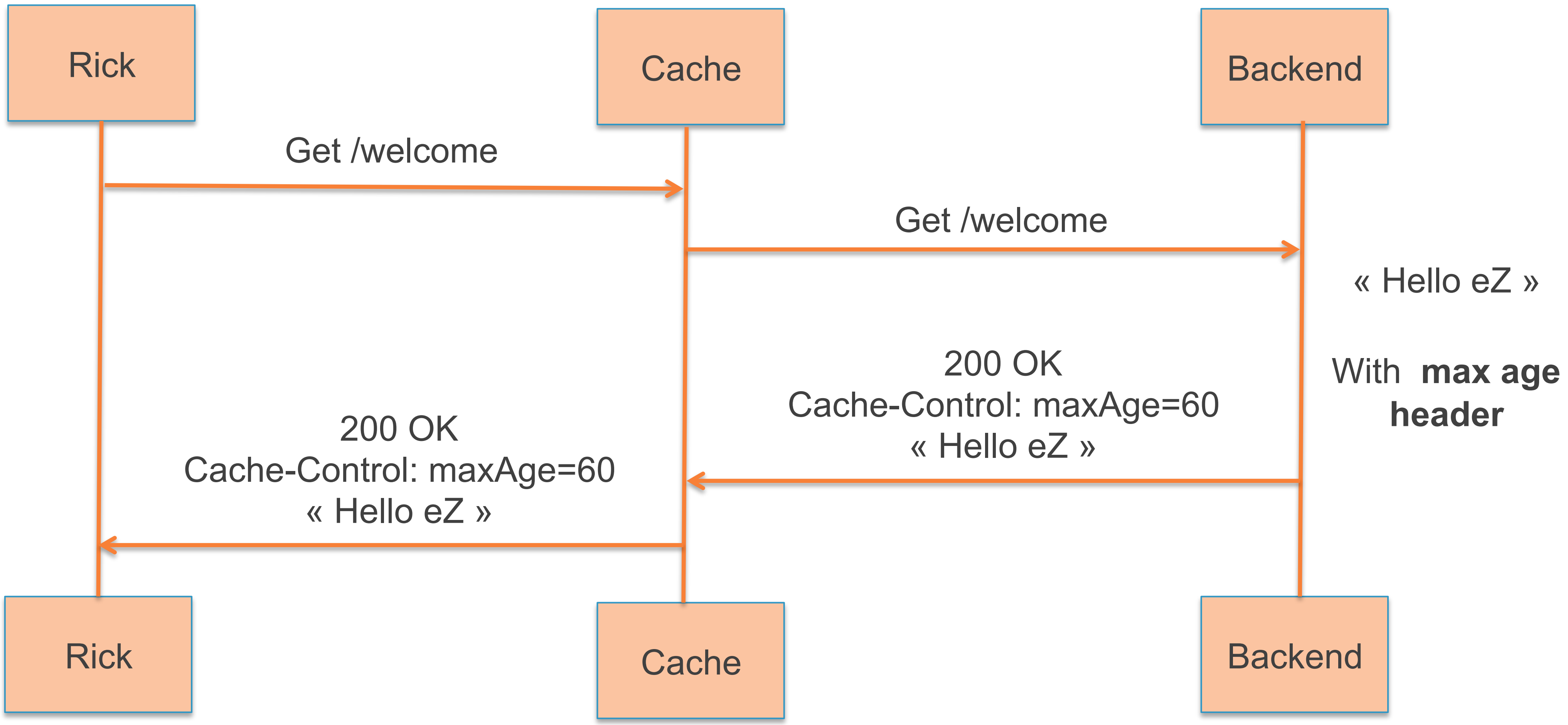
Two caching models are available:

- **Validation**
  - the client always checks if the resource it has in its cache is valid (1 network roundtrip)
  - It gets back a “fast” answer (HTTP 304 with no body) if resource dis not change: good

- **Expiration**
  - Along with the 1st response, the server sends to the client a TTL
  - As long as the TTL is not expired, the client does not request the resource again
  - This saves one network roundtrip per resource: better
  - But it makes it hard to have a TTL if resources can change on random schedule
  - Workaround: use different URL for each version of resource (e.g. ezjscore, content images)
  - If there is a gateway cache, a PURGE command can be used to overcome this, forcing a refresh of the resource before the TTL is expired

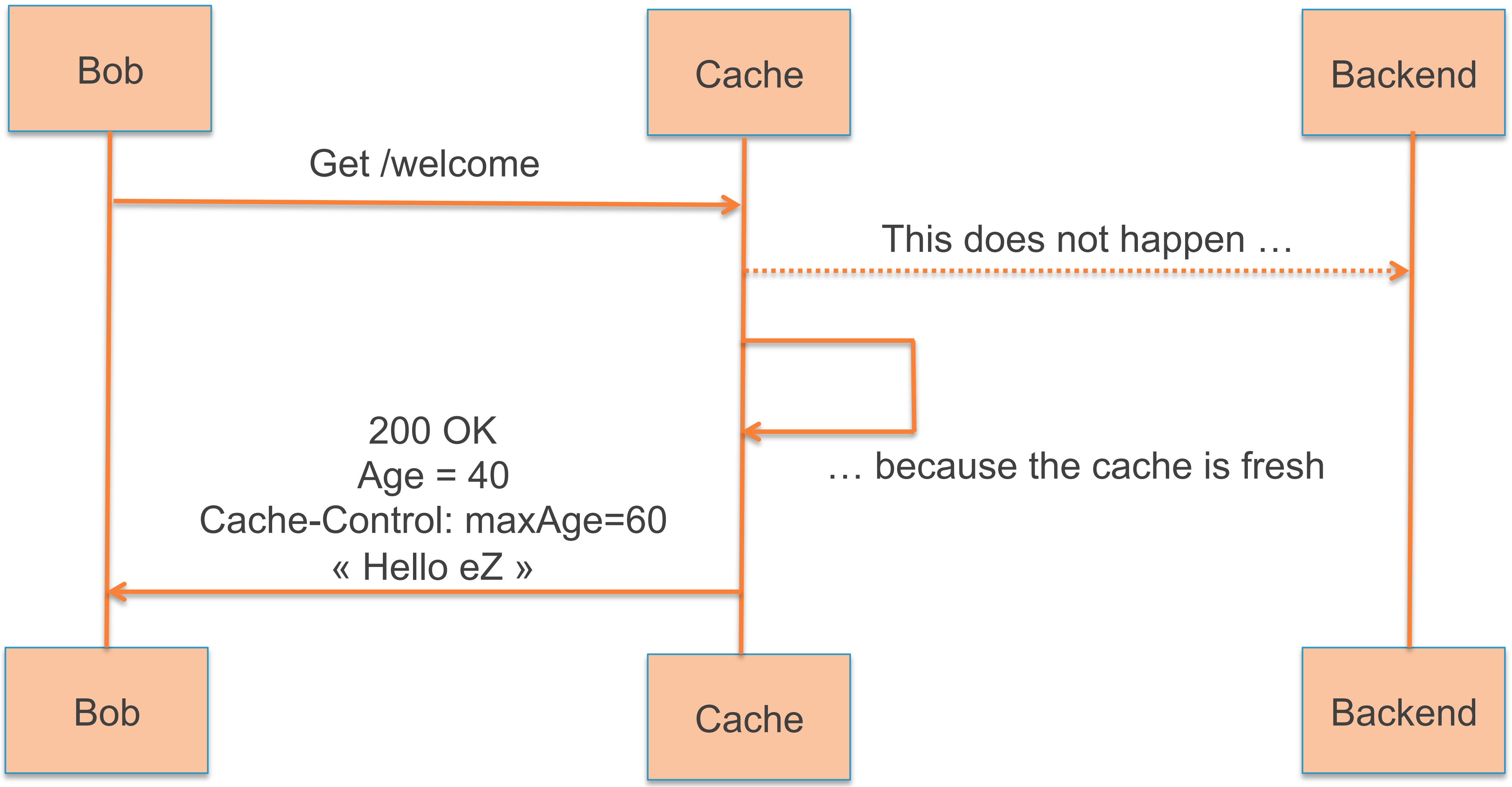
Expiration Model 1/2

Including either or both of the **Cache-Control: max-age=N** or **Expires** headers

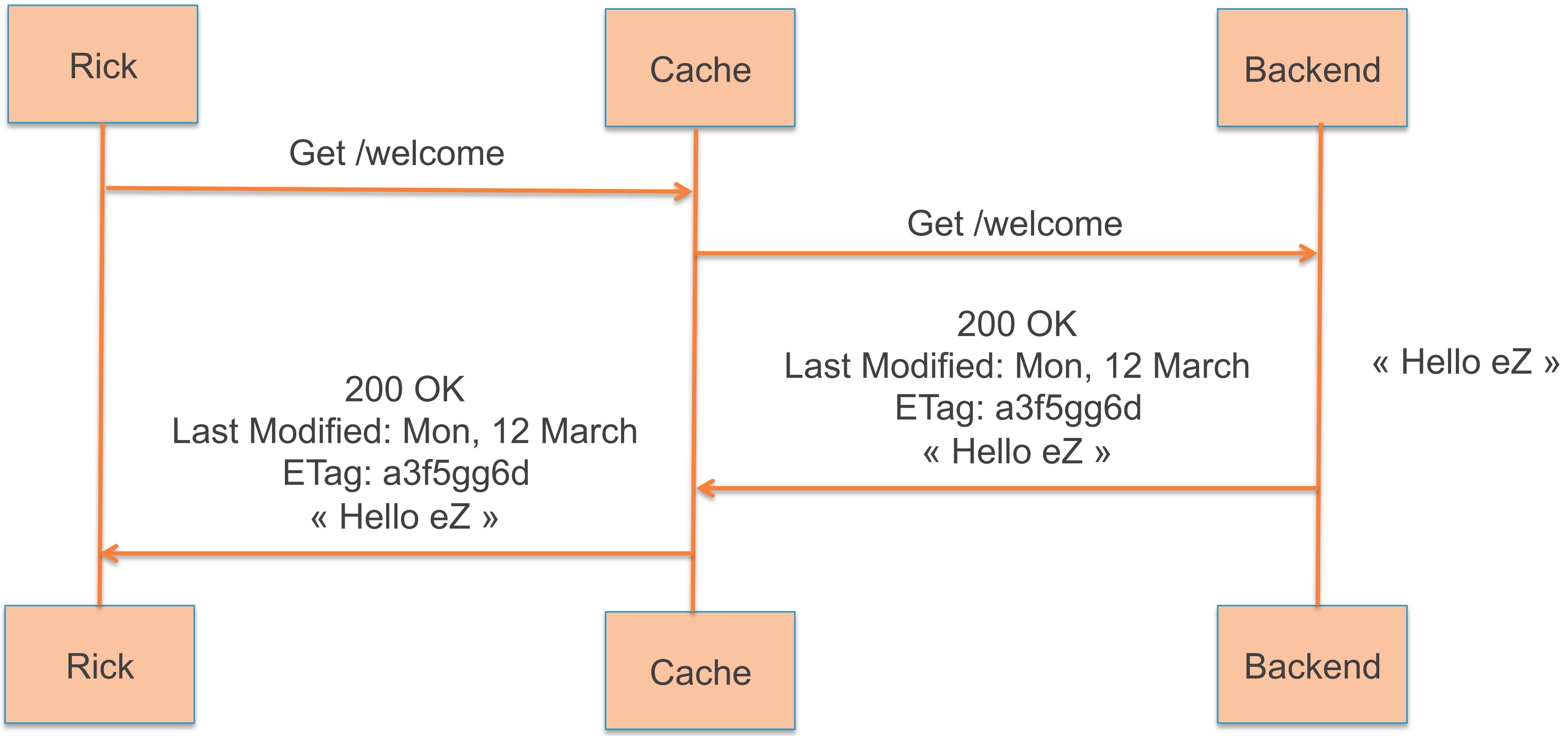


Expiration Model 2/2

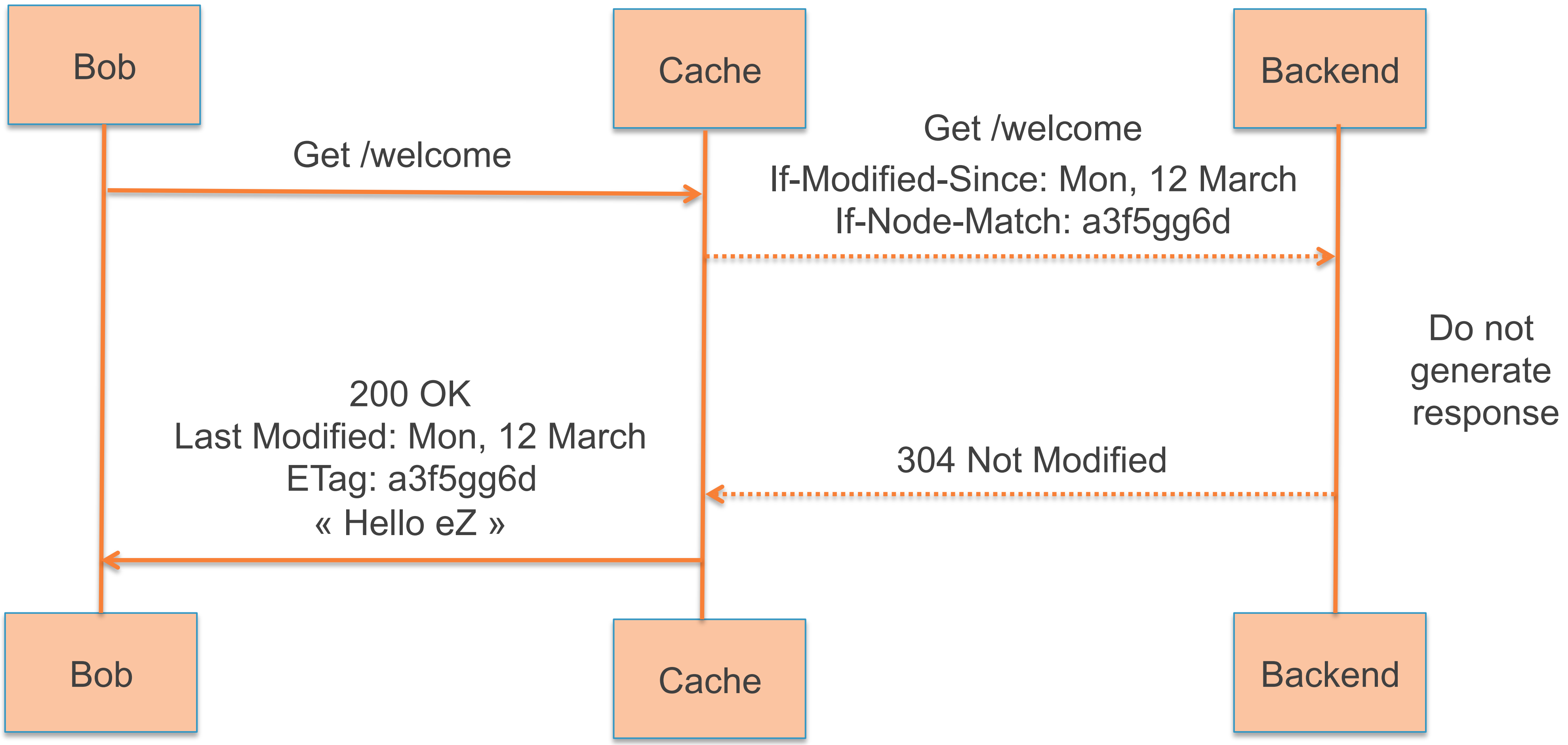
Including either or both of the **Cache-Control: max-age=N** or **Expires** headers



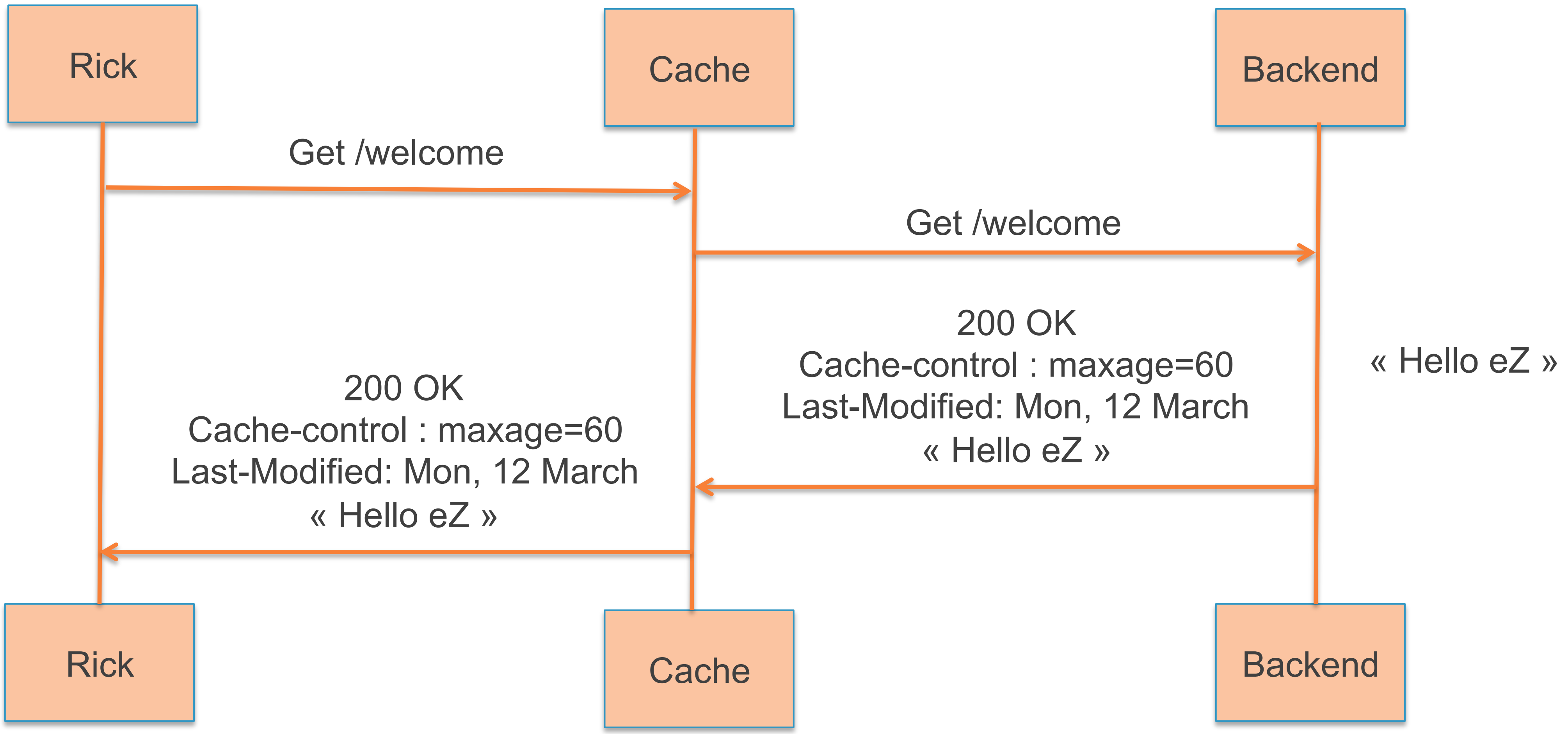
Validation Model 1/2



Validation Model 2/2

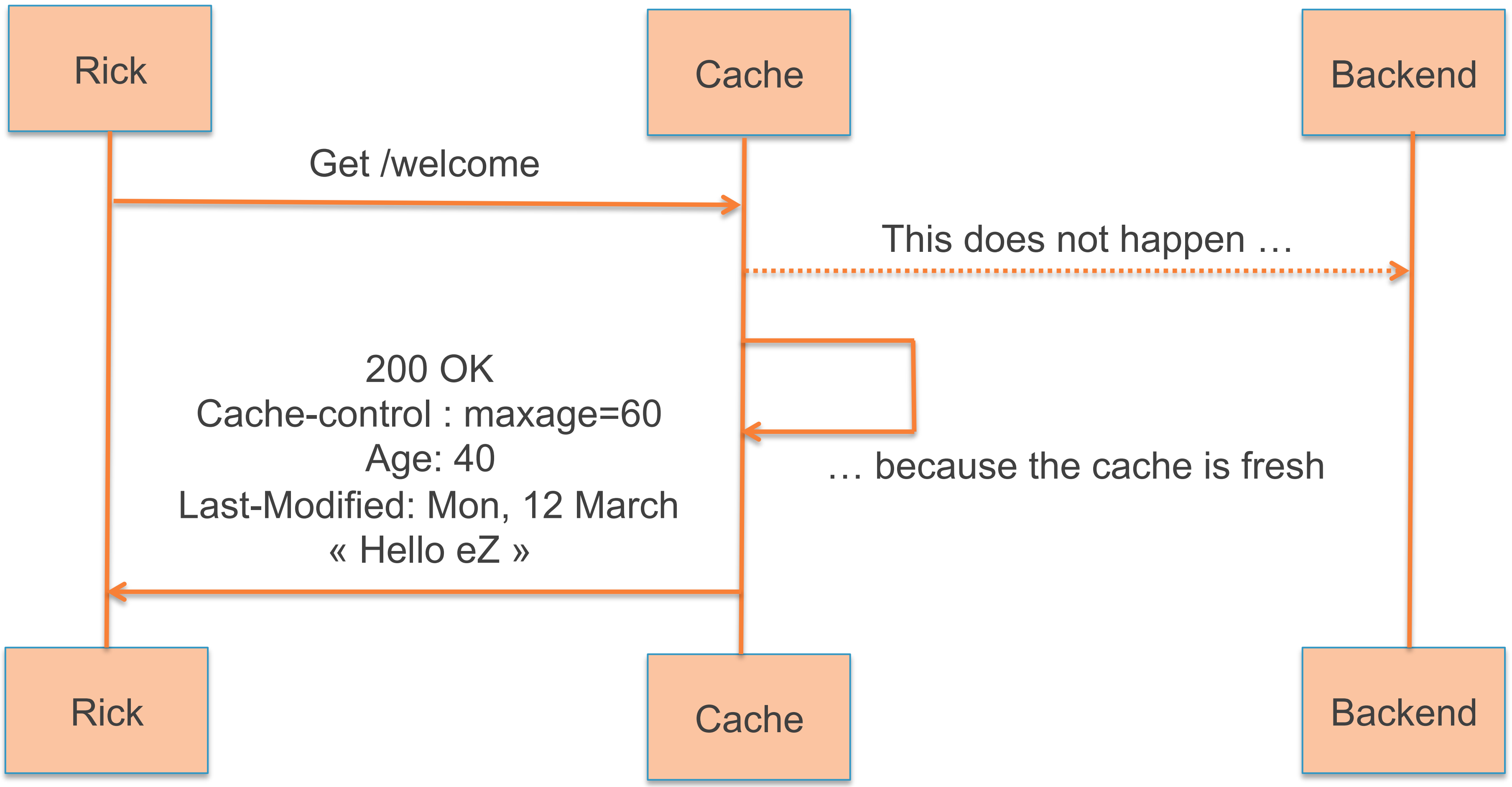


Combining Expiration & Validation models 1/3


Rick
Cache
Backend
Get /welcome
Get /welcome
200 OK
Cache-control : maxage=60
Last-Modified: Mon, 12 March
« Hello eZ »
« Hello eZ »
200 OK
Cache-control : maxage=60
Last-Modified: Mon, 12 March
« Hello eZ »
Rick
Cache
Backend

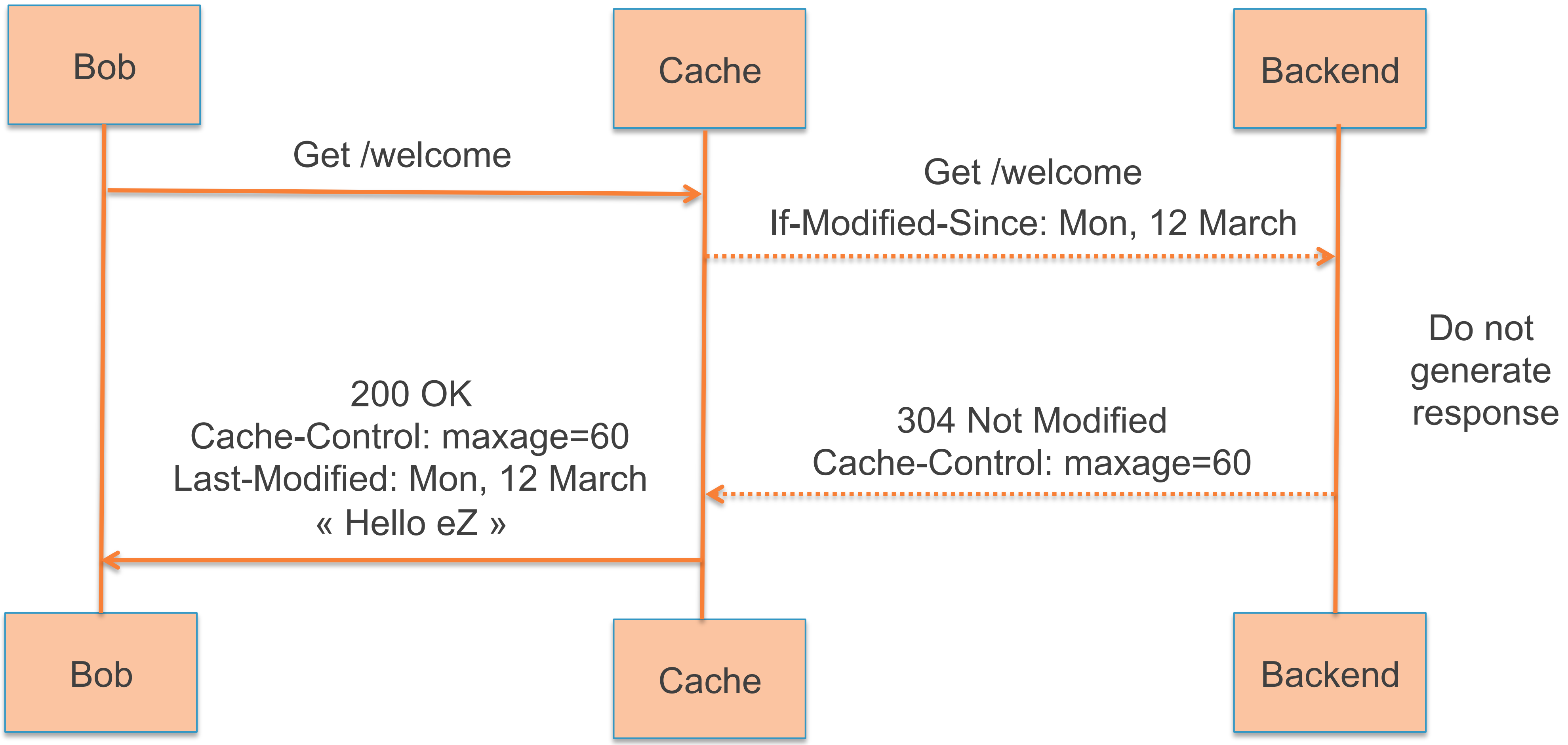
Combining Expiration & Validation models 2/3

Rick comes back after 40s …



Combining Expiration & Validation models 3/3

Bob make the same request, 50 seconds after Rick …



Reminder: Symfony2 cache I/II

Concepts

- **Gateway Cache**, or reverse proxy, is an independent layer that sits in front of your application (it can be within Symfony or external). The reverse proxy caches responses as they're returned from your application and answers requests with cached responses **before they hit your application code**

- **HTTP cache headers** are used to communicate with the gateway cache and any other caches between your application and the client. Symfony2 provides a powerful interface for managing the cache headers

- **HTTP expiration and validation** are the two models used for determining whether cached content is fresh (can be reused from the cache) or stale (should be regenerated by the application)

- **Edge Side Includes (ESI)** allow HTTP cache to be used to cache page fragments (even nested fragments) independently. With ESI, you can f.e. cache an entire page for 60 minutes, but an embedded sidebar for only 5 minutes

More info: http://symfony.com/doc/current/book/http_cache.html

eZ Publish 5 : HttpCache

Content cache

- **eZ Publish** uses the Symfony HttpCache to manage content cache, with both expiration and validation model
- An ETag is computed for every content/version and sent in the response
- It is also possible to use the expiration model to get lightning fast responses

Default configuration

```
ezpublish:
    system:
        my_siteaccess:
            content:
                view_cache: true      # Activates HttpCache for content
                ttl_cache: true       # Activates expiration-based HttpCache for content (very fast)
                default_ttl: 60       # Number of seconds an Http response is valid in cache (if ttl_cache is true)
```

- An additional X-Location-Id header is added in the response for identification

- The HttpCache is disabled for the dev environment

eZ Publish 5 : HttpCache

Making your controller content-cache aware

- You can tie the response of a custom controller to a particular Location id
- It will be cached by default in the Gateway Cache
- It will be automatically expired when that content is published again

```
use Symfony\Component\HttpFoundation\Response;

// In a controller
//  "Connects" the response to location #123 and sets a max age (TTL) of 1 hour.
$response = new Response();
$response->headers->set( ‘X-Location-Id’, 123 );
$response->setSharedMaxAge( 3600 );
```

eZ Publish 5 : Purge

The PURGE mechanism complements the standard validation+expiration model

- It is only useful in presence of a Gateway Cache
- It gives the best of both worlds (large TTL and fresh content)

On publish, one or several Http PURGE requests are sent to the Gateway Cache

This request will have a specific header:

- X-Location-Id (in the case of 1 request per location to purge), or
- X-Group-Location-Id (in the case of 1 request for all locations to purge)

Emulated purge request

- Default mode
- No Http requests will be sent to an external Gateway Cache when publishing
- Works with the internal Httpcache

```
ezpublish:
    http_cache:
        purge_type: local
```

eZ Publish 5 : Purge

Managing external gateways

- One purge request per location, aka «MultipleHttp»

  ```
  purge_type: multiple_http
  ```

- One purge request for all locations, aka «SingleHttp» (X-Group-Location-id header)

  ```
  purge_type: single_http
  ```

  NB: needs specific code for varnish to work with (C language plugin)

- Gateway cache server address

  ```
  purge_servers: [ "http://varnish.server1/", "http://varnish.server2/" ]
  ```

- *Purge all contents*

  On response, you will have an header like this:

  ```
  X-location-Id: *
  ```

eZ Publish 5 : Purge

Manual purging

You can expire the content cache for any location you desire

```
$locationIds = array( 123, 456 );
$container->get( ‘ezpublish.http_cache.purger’ )->purge( $locationIds );

$container->get( ‘ezpublish.http_cache.purger’ )->purgeAll();
```

eZ Publish 5 : ESI support

To use ESI blocks with a reverse proxy :

- VirtualHost is pointing on index.php
- Don’t load **EzPublishCache()** in index.php (only if you are using a reverse proxy)

=> otherwise Symfony2 will use internal reverse proxy and send to the real reverse proxy the HTML result of ESI blocks

Varnish Cache

Varnish Cache is a web application accelerator also known as a caching HTTP **reverse proxy**.

You install it in front of any server that speaks HTTP and configure it to cache the contents. Varnish Cache is really, really fast. It typically speeds up delivery with a factor of **300 - 1000x**, depending on your architecture.
A high level overview of what Varnish does can be seen in the video attached to this web page.

https://www.varnish-cache.org/about

Varnish Cache – How to ?

eZ Publish 5.x was tested with last major release of Varnish Cache (3.x).

All informations about installation, configuration are located here:
https://www.varnish-cache.org/docs/3.0/installation/index.html

How to start Varnish ? Two ways:

Manually

```
varnishd -f /usr/local/etc/varnish/default.vcl -s malloc,1G -T
127.0.0.1:2000 -a 0.0.0.0:8080
```

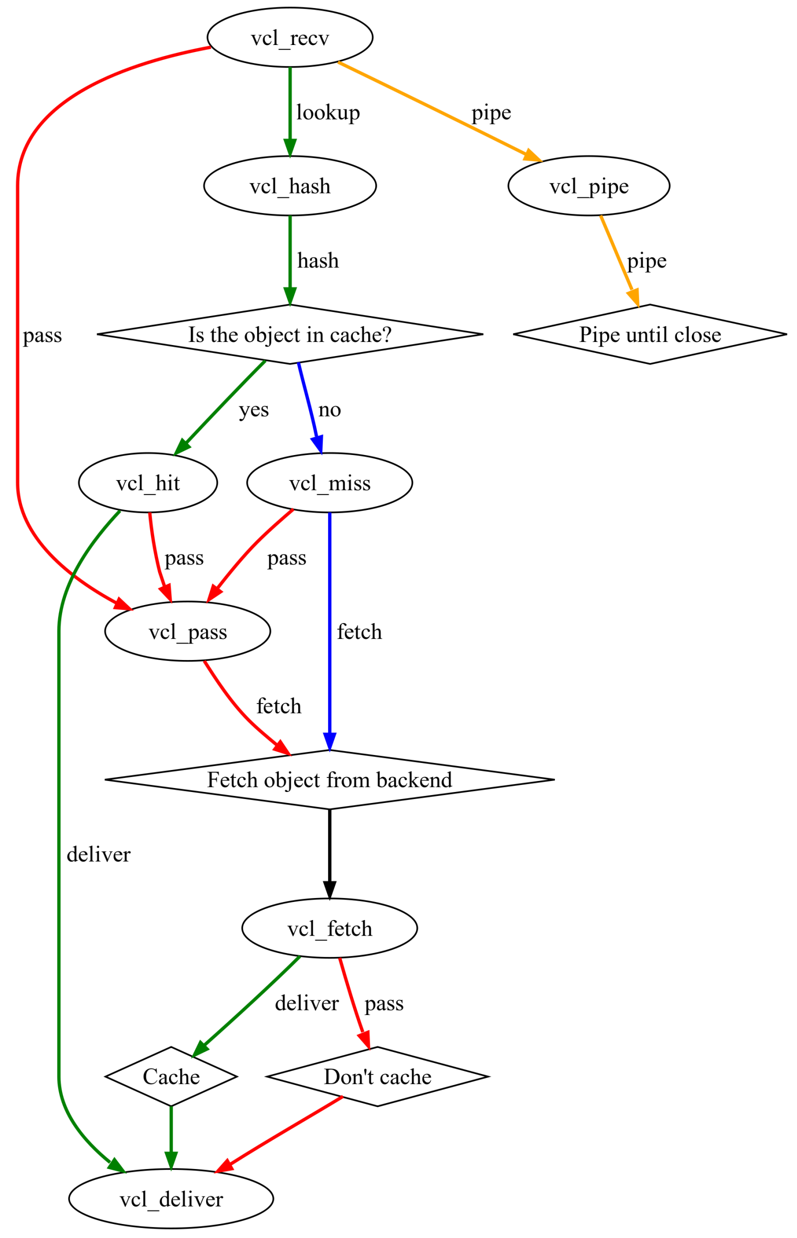
Init Script

About varnish configuration (*On Debian/Ubuntu OS*):

- You have to modify your */etc/default/varnish* file (configuration used by init.d script)

For all Operating Systems, modify or create a new vcl file (**/etc/varnish/default.vcl**)

Varnish - Vcl



- **vcl_recv**: is the first VCL function executed, right after Varnish has decoded the request into its basic data structure
- **vcl_hash**: defines what is unique about a request
- **vcl_hit**: right after an object has been found in the cache
- **vcl_miss**: right after an object was looked up and not found in cache
- **vcl_fetch**: is the backend-counterpart to vcl_recv. Return deliver, tells Varnish to cache, returning hit_for_pass tells it not to cache, but does not run the vcl_pass function of VCL for this specific client.

Varnish VCL and eZ Publish 5

A dedicated VCL could be found on confluence
https://confluence.ez.no/display/EZP/Using+Varnish

Be careful: only multiple_http is implemented on this example. If you would like to switch to single_http, you have to create a Vmod or add some C code.

Some tools to debug with Varnish:

- varnishlog
- varnishadm

Exercise

eZ Publish 5 and Varnish