ADOBE CUSTOMER BROWN BAG SERIES

TODAY'S TOPIC: Optimizing the CQ Dispatcher Cache

Andrew Khoury
Customer Support Engineer, Adobe

✓ For the best listening experience, we recommend using a headset

What's Covered

1. Review of dispatcher basics
2. Dispatcher cache flushing
3. Tips and tricks for improving performance
4. How to design your site so you get the most out of your Dispatcher

Adobe

Prerequisite Knowledge

You should know:

- What a URL is
- What a Web server is
- What Adobe CQ is and have a basic understanding of a typical CQ architecture
- Basic understanding of HTTP protocol including what HTTP methods and headers are

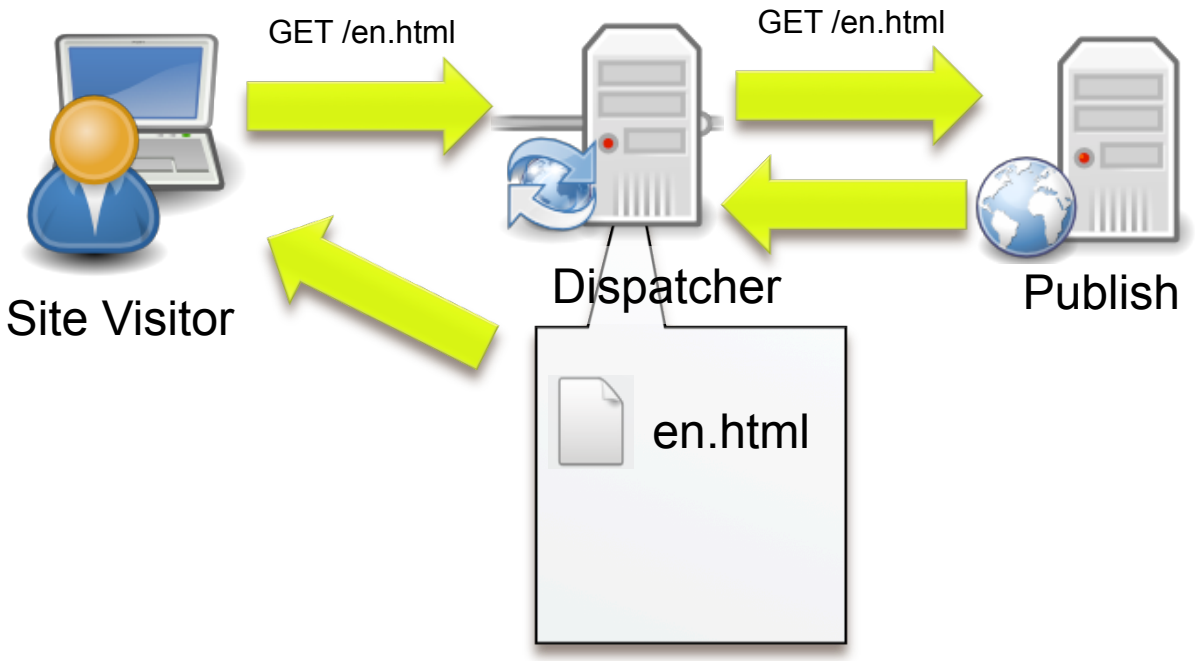
Note:

- This presentation focuses on dispatchers using Apache Web Server.

What is the Dispatcher?

- A web caching proxy and load balancing tool that works as a layer in front of your CQ instances.
- It can either be used as a load balancer or you can put it behind your hardware load balancer.
- Works as a module installed to a web server such as Apache.
- Works with these web servers:
  - Apache HTTP server 2.x
  - Microsoft IIS 6 and IIS 7.x
  - Oracle iPlanet Web Server

What is Dispatcher?

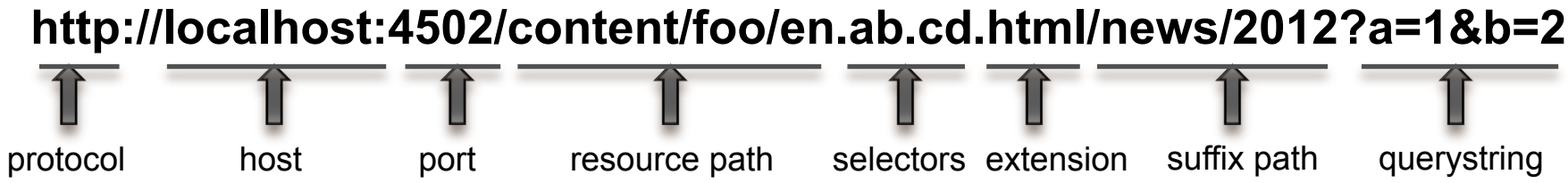


Why do I need a dispatcher?

- Why?
  - Reduce load on publish instances
  - Improve site performance
  - Implement load balancing
  - Filter out unwanted traffic
- Can’t I just use a CDN instead?
  - Yes, however…
- You should also use a dispatcher
  - Out of the box, the Dispatcher gives you more fine grained control over how you delete (“flush”) files from your cache.
  - Since the dispatcher is installed to a web server it adds the ability to rewrite URLs, and use SSI (Server Side Includes) before the requests hit the publish instance.
  - Dispatcher gives you added security because it allows you to block certain client HTTP headers, URL patterns, and to serve error pages from a cache.

URL Decomposition

- **Resource Path** – The path before the first '.' in the URL.
- **Selectors** – Any items separated by '.' characters between the resource path and the file extension.
- **Extension** – Comes after the resource path and selectors.
- **Suffix Path** – A path that comes after the file extension.



The dispatcher will only cache files that meet the following criteria…

What does dispatcher cache?

#1

The **URL** must be **allowed**

by the **/cache => /rules** and **/filter**

sections of **dispatcher.any**

What does dispatcher cache?

#2

The **URL** must have a **file extension.**

Example:

/content/foo**.html** ✔**cached**

/content/foo ✖ **not cached**

What does dispatcher cache?

#3

The URL must **not contain** any **querystring parameters** (no “?” in the URL).

Example:

/content/foo.html**?queryparam=1** ✖ **not cached**

/content/foo.html ✔**cached**

What does dispatcher cache?

#4

If the URL has a suffix path then the **suffix path must have a file extension**.

Example:

/content/foo.html/suffix/path**.html** ✔**cached**

/content/foo.html/suffix/path ✖ **not cached**

What does dispatcher cache?

#5

The HTTP method must be **GET** or **HEAD**

Example:

GET /foo.html HTTP/1.1	✔**cached**
HEAD /foo.html HTTP/1.1	✔**cached**
POST /foo.html HTTP/1.1	✖ **not cached**

What does dispatcher cache?

#6

HTTP response status (from CQ)

must be **200 OK**

Example:

HTTP/1.1 200 OK	✔cached
HTTP/1.1 500 Internal Server Error	✖ not cached
HTTP/1.1 404 Not Found	✖ not cached

What does dispatcher cache?

#7

HTTP response (from CQ)

must **not contain** the **response header "Dispatcher: no-cache"**

Example:

HTTP/1.1 200 OK
Content-Type: text/plain
Content-Length: 42

✔ **cached**

HTTP/1.1 200 OK
Content-Type: text/plain
Content-Length: 42
Dispatcher: no-cache

✖ **not cached**

Suffix Exception #1

If the URL is a cacheable resource path

with an extension and a URL with a suffix path is requested then it will **not** be cached

Example:

/content/foo.html is already cached

/content/foo.html/suffix/path**.html** ✖ **not cached**

Suffix Exception #2

In the opposite case, when the suffix file already exists in the cache and the resource file is requested then the resource file is cached instead

Example:

~~/content/foo.html/suffix/path.html~~	**deleted**
/content/foo.html	✔**cached**

How does caching work?

Demonstration

?

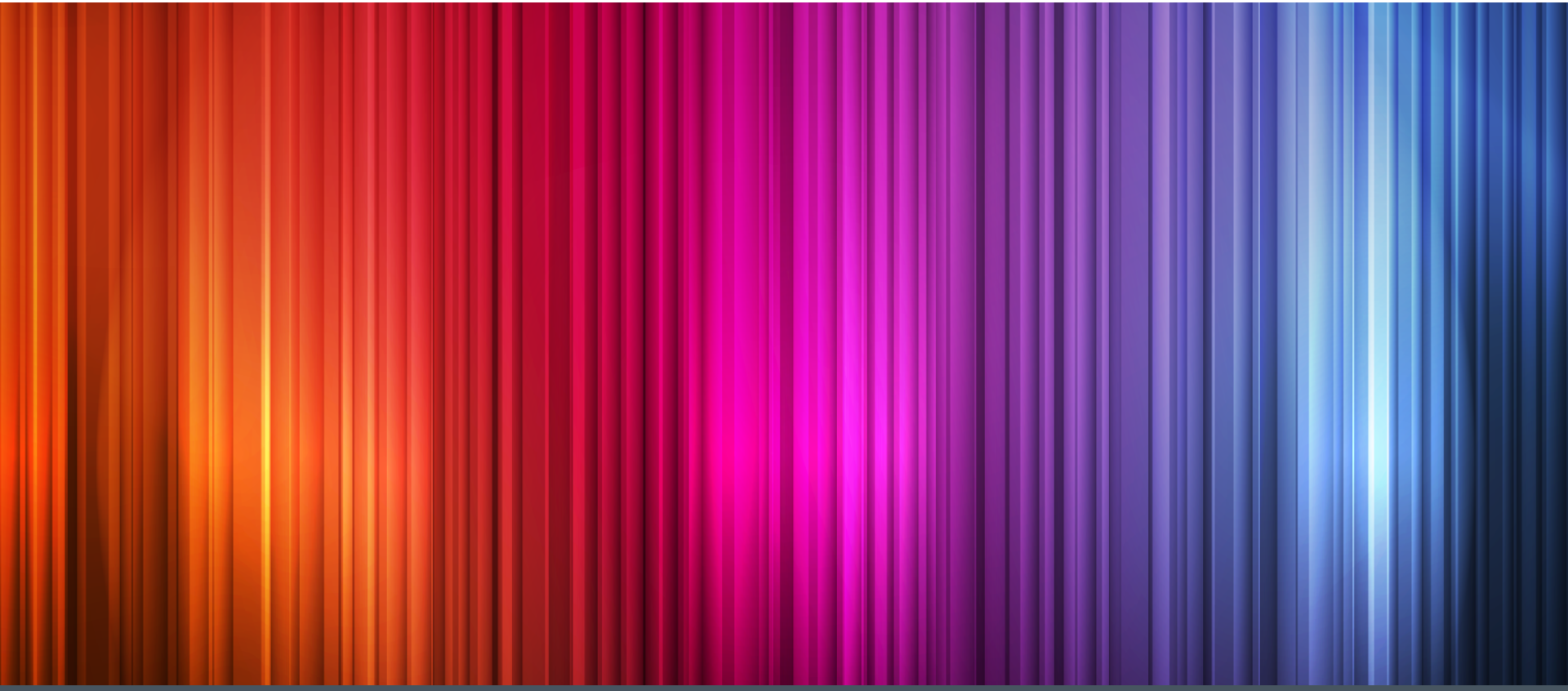
What should I cache?

What should I cache?

- Cache everything if possible.
- What requests should I not cache?
  - You shouldn't cache files that are expected to change on every request
  - Personalized content (User's name, location, preferences, etc.)
  - Do not cache files that will only be requested once
  - Do not cache URLs that use selectors or suffixes where an **infinite number of values are allowed**
    - For example: /content/foo.selector.html
      - /content/foo.1.html
      - /content/foo.2.html
      - Etc…
  - Do not allow caching on unwanted requests. Block caching of unwanted requests by returning 403 or 404 for requests that use an invalid selector or suffix.

Dispatcher Cache Flushing

Deleting and invalidating files in your dispatcher cache.



Flushing your Cache

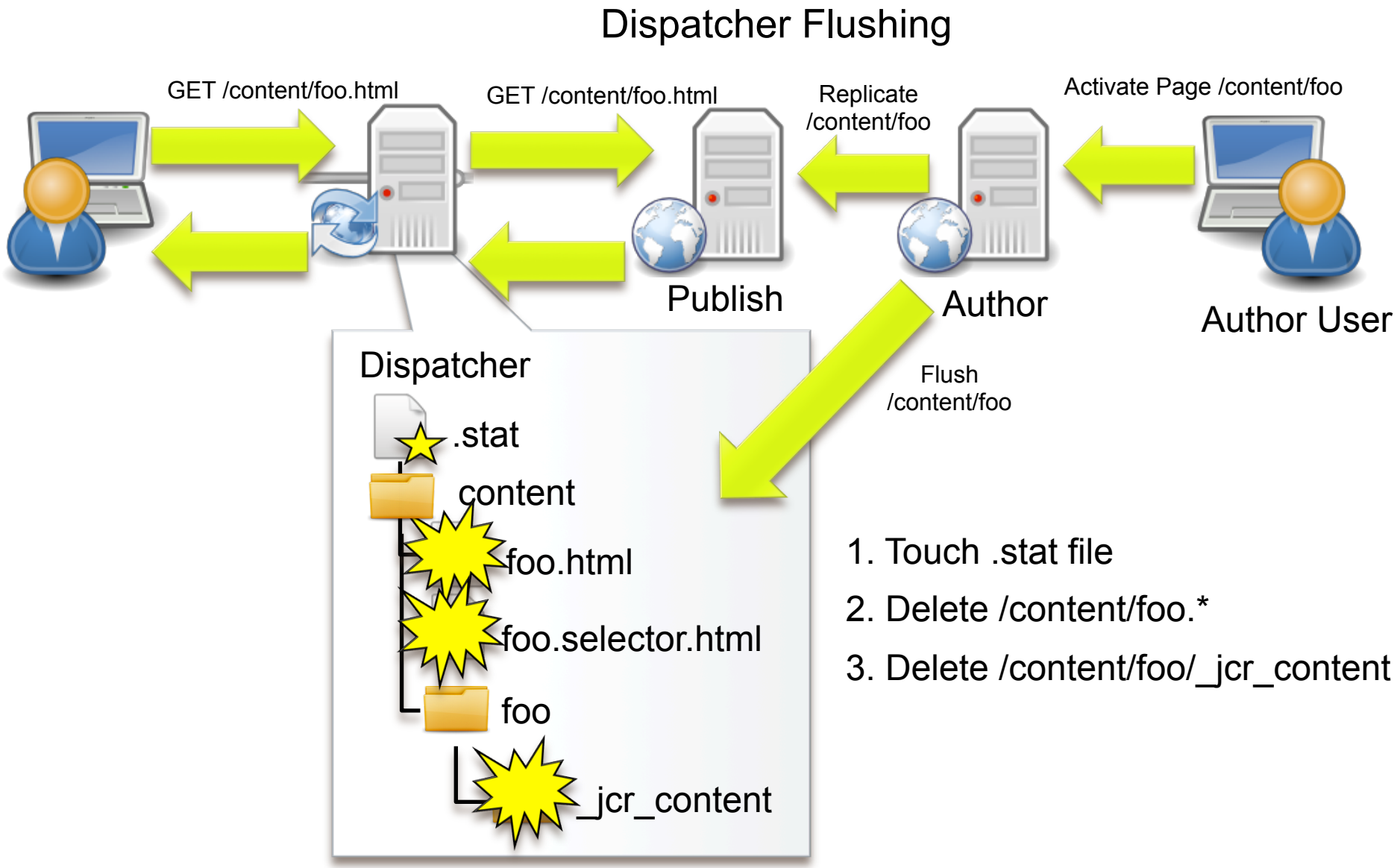
What is a dispatcher cache flush?

- It is a mechanism that allows you to selectively delete files from your dispatcher cache via an HTTP request.
- Flushing allows you to bring your cache files up-to-date.
- Here is a sample flush request for path /content/foo:

```
GET /dispatcher/invalidate.cache HTTP/1.1
CQ-Action: Activate
CQ-Handle: /content/foo
CQ-Path: /content/foo
Content-Length: 0
Content-Type: application/octet-stream
```

- If you configure a flush agent in CQ then CQ will automatically flush that file from the cache on activation.

Tip #1: Do cache flushing from publish



Dispatcher Flush

- How does dispatcher cache flushing work?

1. User activates a page in CQ
2. CQ sends a flush request like this to the dispatcher:

```
GET /dispatcher/invalidate.cache HTTP/1.1
CQ-Action: Flush
CQ-Handle: /content/foo
CQ-Path: /content/foo
Content-Length: 0
Content-Type: application/octet-stream
```

3. This request tells dispatcher to do the following:

   i. Touch the .stat file to indicate the last time a cache flush occurred. (only touches the nearest ancestor .stat file)

   ii. Delete all files under {CQ-Path}.* . For example if the flushed path was /content/foo then /content/foo.* would be deleted

   iii. Delete directory {CQ-Path}/_jcr_content

   iv. Dispatcher responds to CQ's flush request with 200 OK status

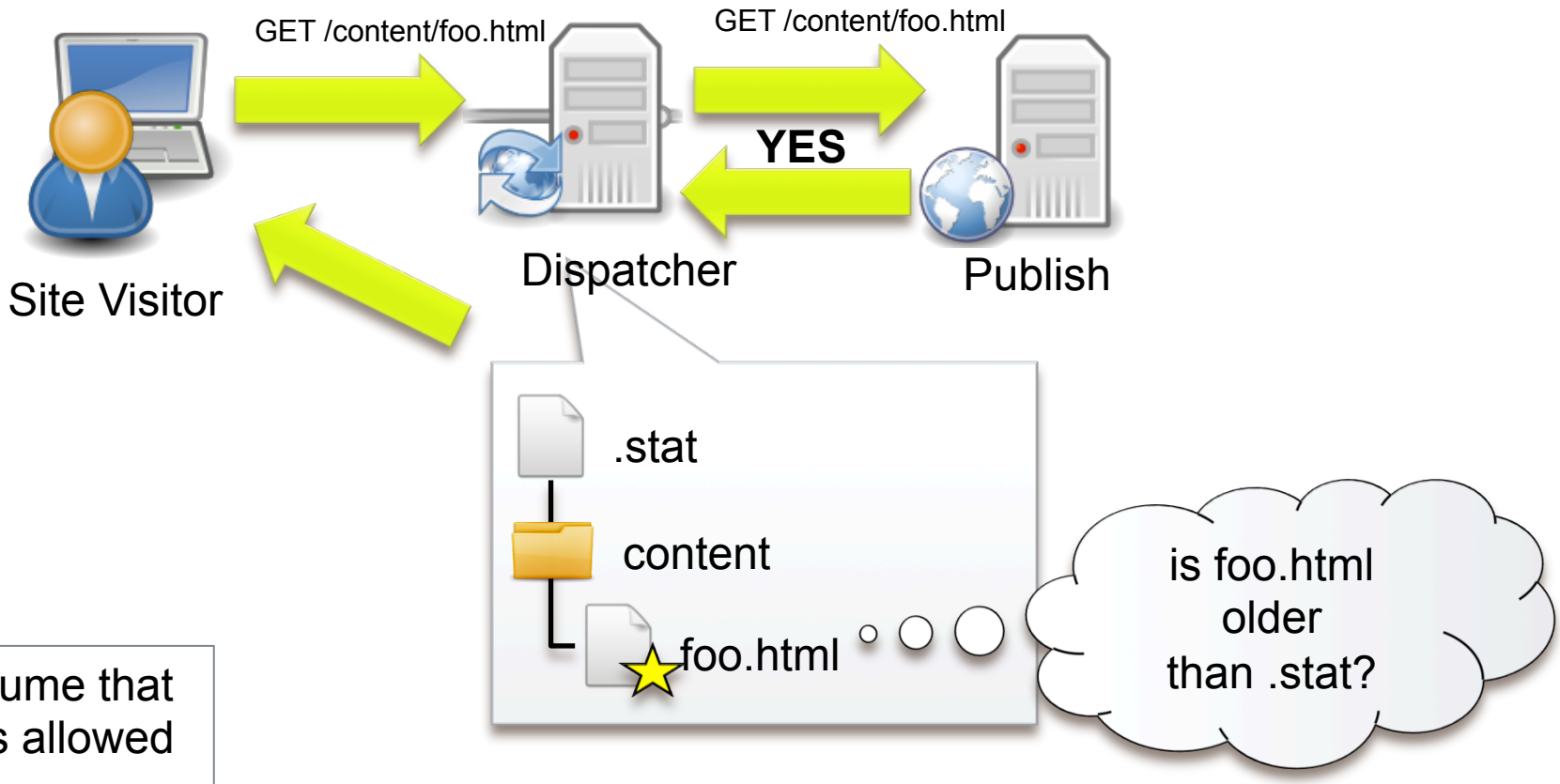
Cache Invalidation

What is cache invalidation?

- Cache invalidation is the mechanism by which the dispatcher considers certain cache files to be “outdated” or “stale”

1. Dispatcher checks if the file is allowed by the rules in the **/cache => invalidate** section of dispatcher.any
2. The file’s last modified time is compared to that of its nearest .stat file
3. If the file is older then than .stat it is considered “outdated” and will be re-requested from publish

What is Dispatcher?



We assume that *.html is allowed in the /invalidate rules in dispatcher.any

Cache Invalidation

- dispatcher.any **/cache => /invalidate** section
  - Defines rules telling the dispatcher which files should be auto-invalidated on flush requests
  - For example, this is the default configuration, it is configured to allow auto-invalidation on all .html files:

```
/farms
 {
 # first farm entry (label is not important, just for your convenience)
 /website
  {
      ….

      /cache
      {
        …

        /invalidate
        {
          /0000 { /glob "*" /type "deny" }
          /0001 { /glob "*.html" /type "allow" }
        }
       }
       …
   }
 }
```

Cache Invalidation

- How do .stat files work?
  1. If you set the /statfileslevel in dispatcher.any to 0 then only a single .stat file will be created in the root of the cache
  2. A flush request will cause the .stat file to be touched
  3. All files allowed by the /invalidate rules will be re-cached next time they are requested

Stat Files Level

- /statfileslevel
  - 0 – single stat file at the root of the cache
    /.stat
  - 1 – stat file at root of cache and under each direct child directory. For example:
    /.stat
    /content/.stat
    /etc/.stat
  - 2 – stat file at root of cache and under each direct child directory and each of their direct child directories:

    /.stat
    /content/.stat
    /etc/.stat
    /content/site1/.stat
    /content/site2/.stat
    /etc/designs/.stat
  - And so on…

Stat Files Level Example

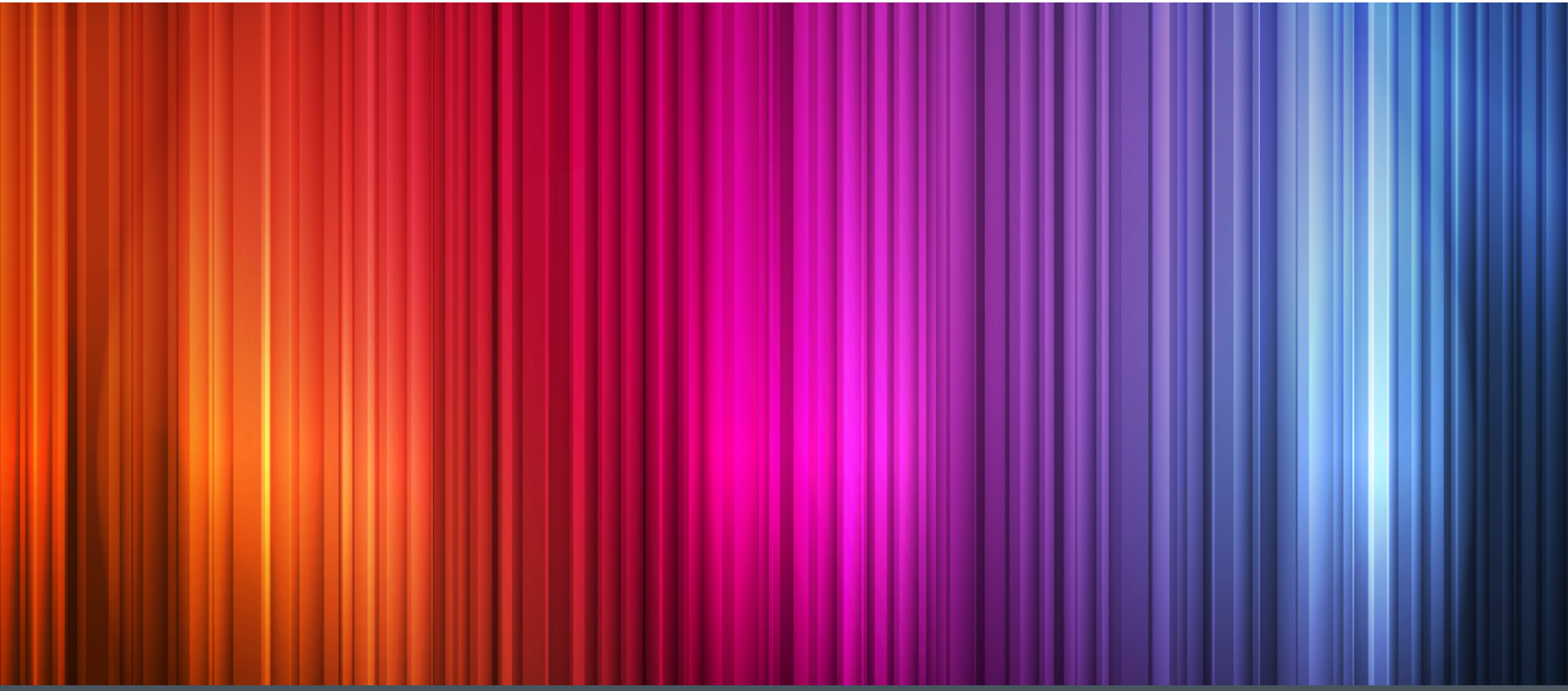
- If we have statfileslevel set to 2 and our cache looks like this
  /.stat
  /content/.stat
  /content/foo/.stat
  /content/bar/.stat
- Then if the path **/content/foo/en** is flushed then the following files are touched:
  /.stat
  /content/.stat
  /content/foo/.stat
- Only files directly under /, /content, and /content/foo and its subdirectories would be affected by this.

Summary of Cache Invalidation

- The /statfileslevel allows you to isolate the invalidation to the branch where the flush occurs.
- When the dispatcher decides if a file is “outdated” it will only look at the nearest ancestor .stat file in the file system tree.
- You can leverage this feature if you know that your cache in one section will be unaffected by changes in another section.

Tips, Tricks and Best Practices

Getting the most out of your dispatcher.



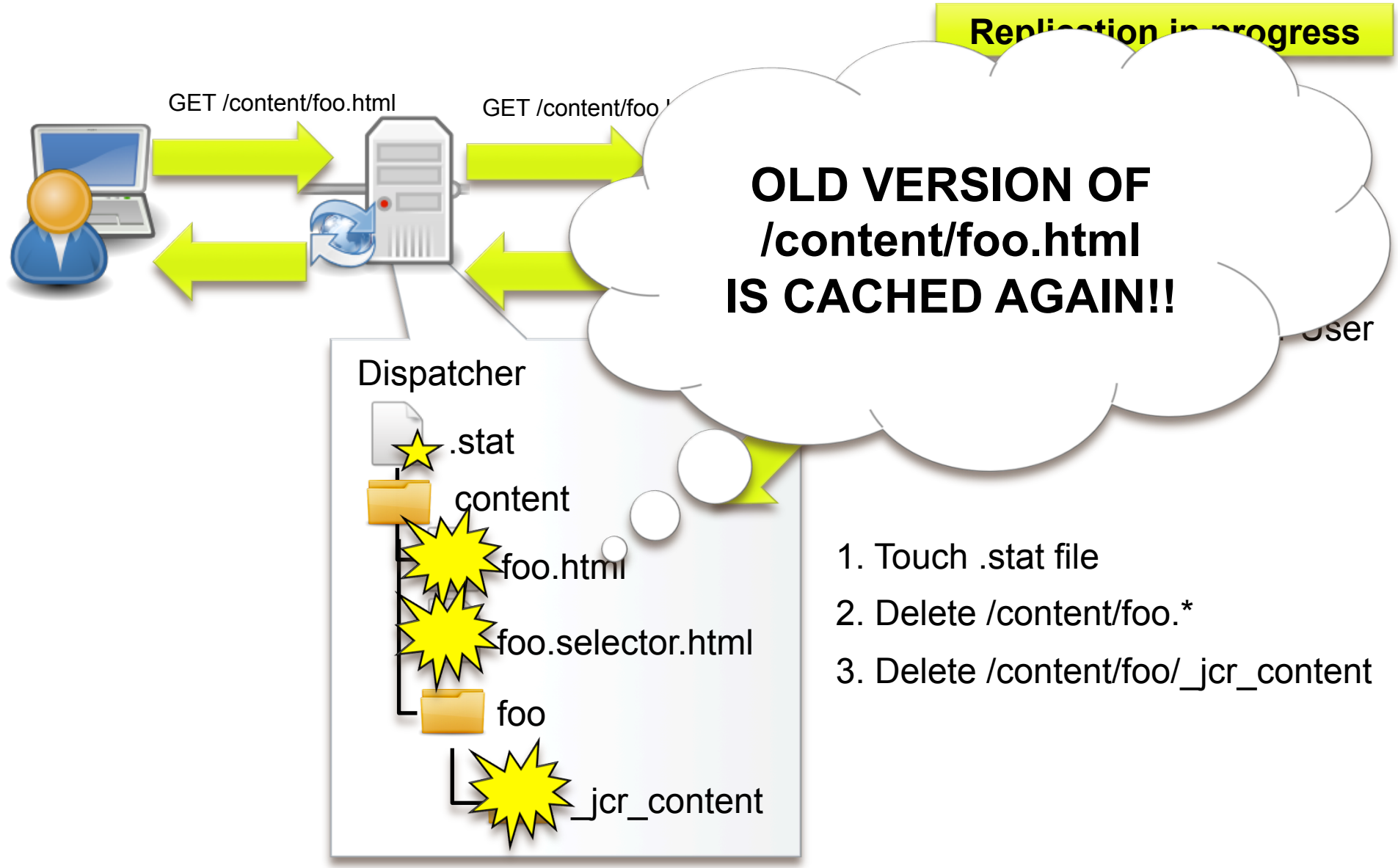
Tip #1
Configure cache flushing on the publish instance

Tip #1: Do cache flushing from publish

- Problem
  - When flushing is done from author the old version of file could get re-cached due to a race condition.

- Solution
  - Configure dispatcher flushing on the publish instances instead.
  - See here for how to configure this:
    - http://dev.day.com/docs/en/cq/current/deploying/dispatcher/page_invalidate.html#Invalidating%20Dispatcher%20Cache%20from%20a%20Publishing%20Instance
    - http://helpx.adobe.com/cq/kb/HowToFlushAssetsPublish.html

Tip #1: Do cache flushing from publish



Stat Files Level

Demonstration:

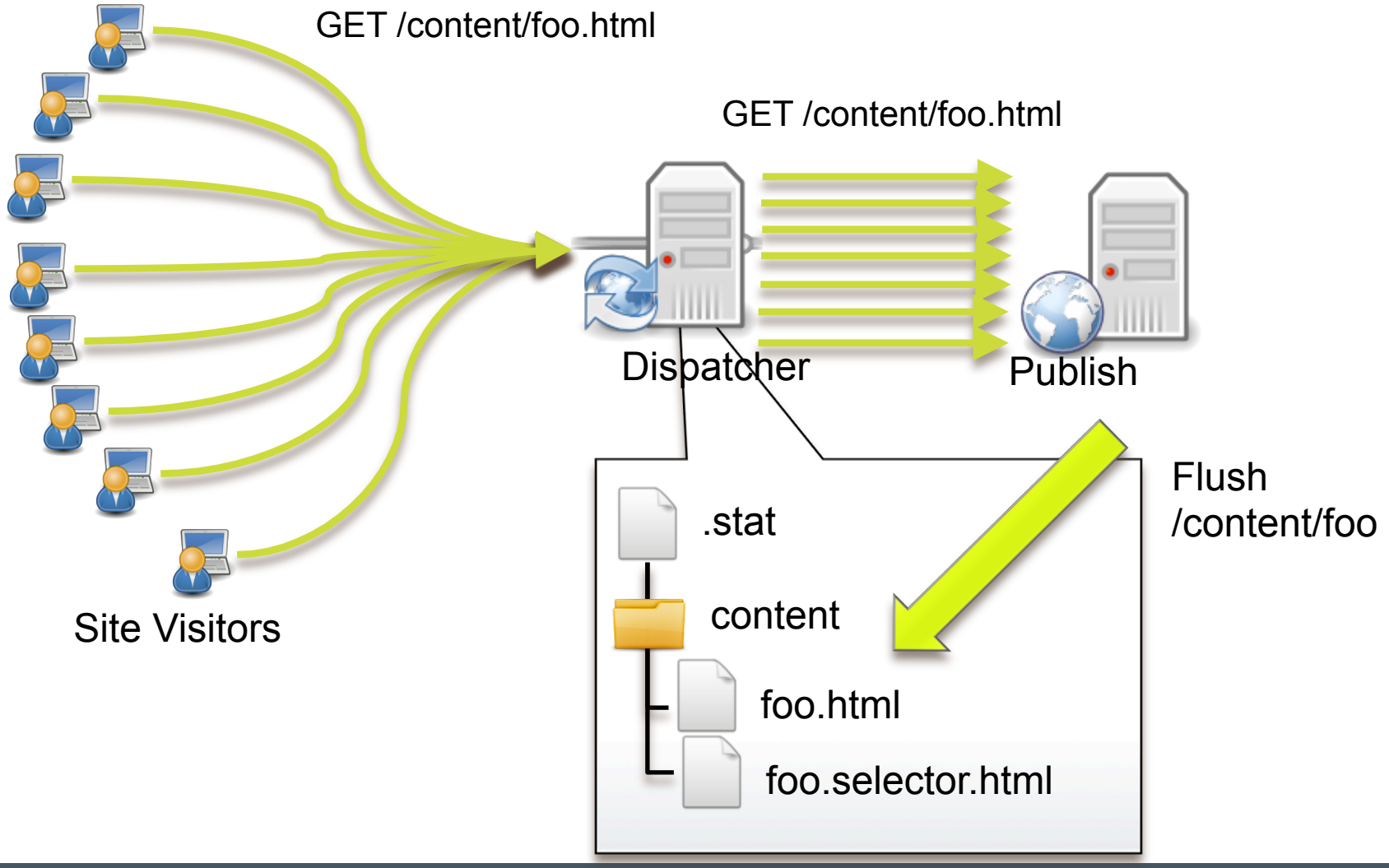
How to configure flushing from publish

Tip #2
Use a re-fetching dispatcher flush agent

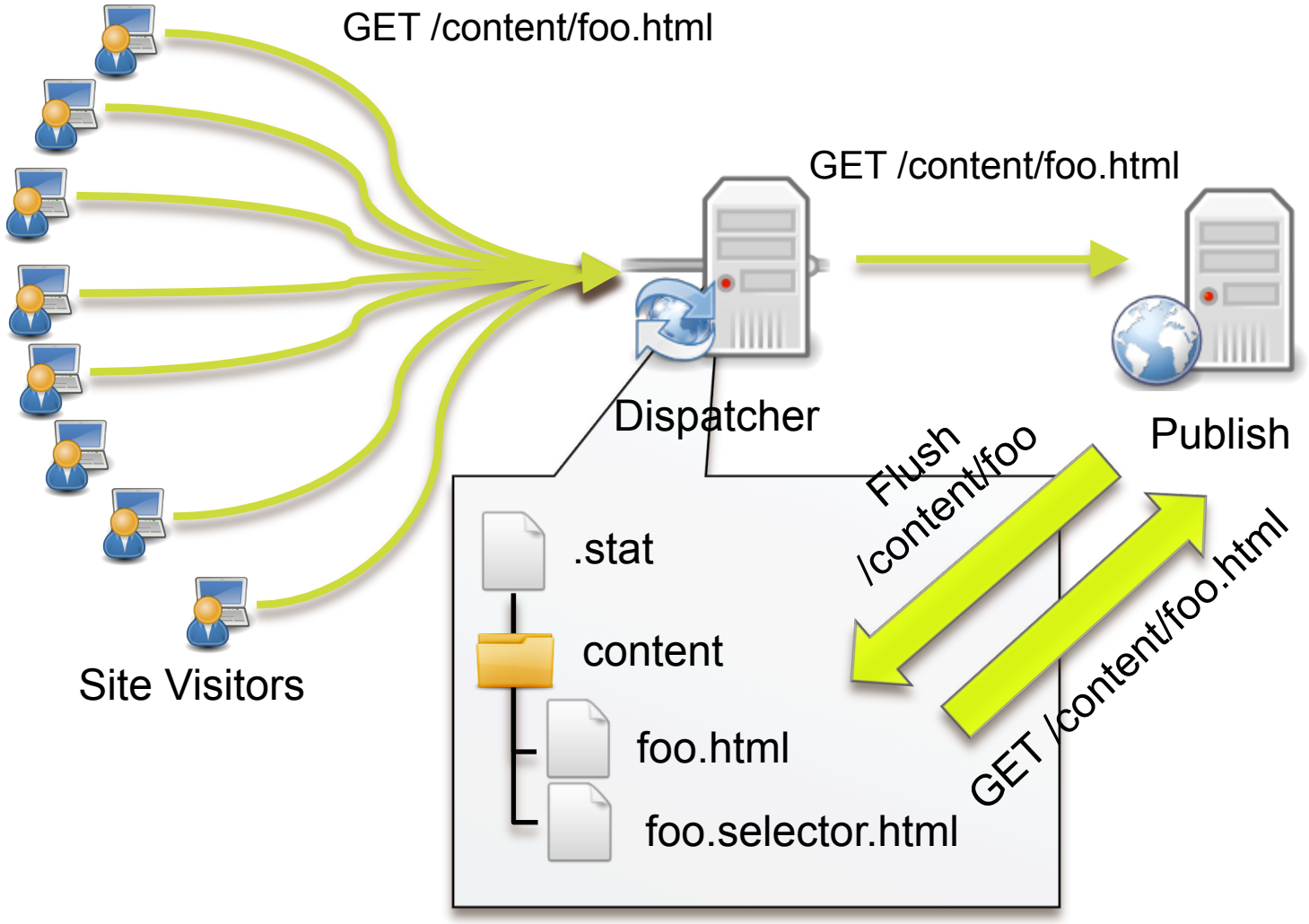
Tip #2: Implement a Re-fetching Flush Agent

- Problem
  - If a file is doesn't exist in the cache yet then simultaneous requests for the file will get requested from the publish instances until it is cached.
- Solution
  - A feature called re-fetching exists in the dispatcher since version 4.0.9 which helps avoid this issue.
  - To install the re-fetching agent, do the following:
    - Install the sample flushing agent package found [here] to the CRX package manager
    - Go to flushing agent config page, for example http://localhost:4502/etc/replication/agents.publish/flush.html
    - Edit the configuration like this:
      General => Serialization Type = Custom Dispatcher Flush
      Extended => HTTP Method = POST

Problem with deleting files from the cache



Same diagram with a re-fetching agent



Tip #2: Implement a Re-fetching Flush Agent

Dispatcher Flush with Re-Fetching

1. Touch nearest .stat file
2. Delete all files under {flush-path}.* **excluding those in the re-fetch list**. For example if the flushed path was /content/foo then /content/foo.* would be deleted.
3. Delete directory {flush-path}/jcr:content
4. **Refetch the files in the refetch list**

Tip #2: Implement a Re-fetching Flush Agent

Demonstration:

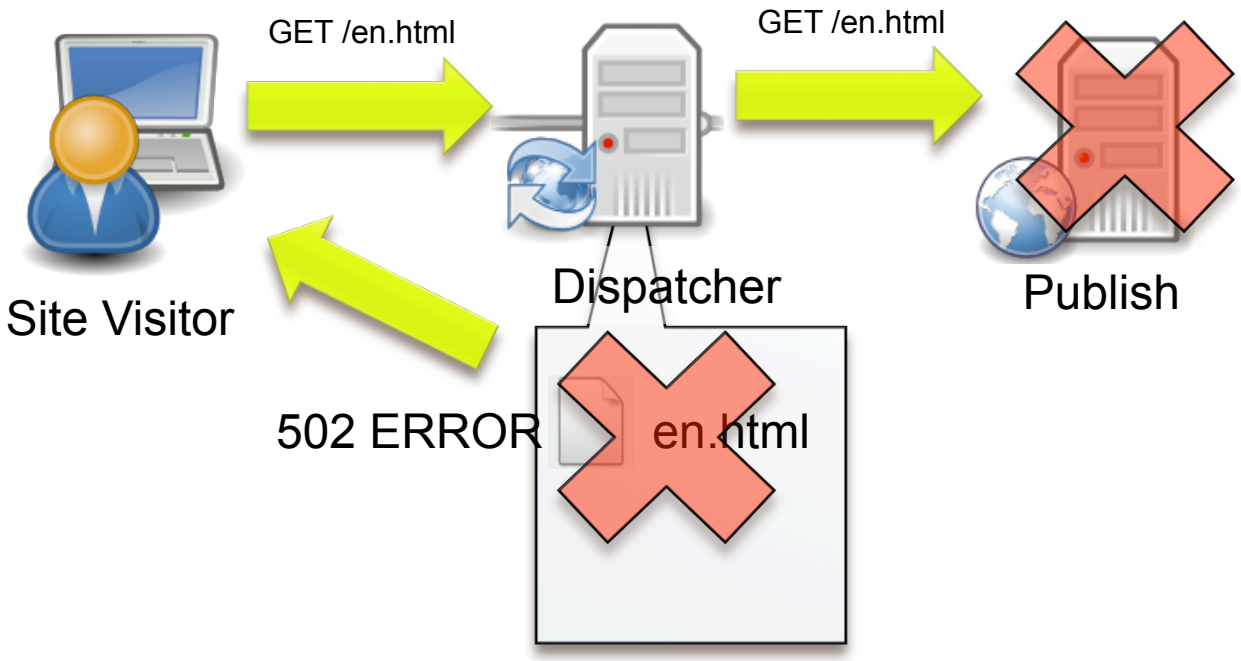
How to install the re-fetching flush agent

Tip #3
Use /serveStaleOnError

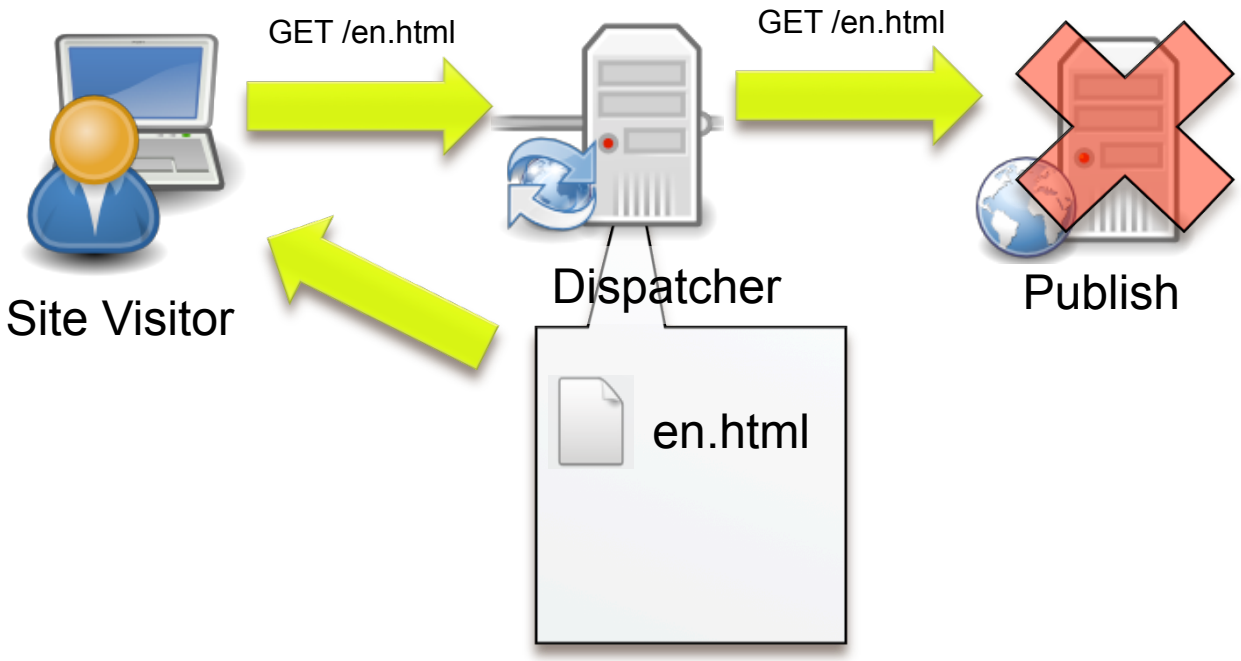
Tip #3: Set ServeStaleOnError

- Problem
  - If all publish instances are unavailable, you will get "502 Bad response code". It might possible that all your pages are already cached before publishers went down. In this case it makes sense to instead serve cached content.
- Solution
  - Set **/serveStaleOnError “1”** in your dispatcher.any

What is Dispatcher?



What is Dispatcher?



Tip #3: Set ServeStaleOnError

Demonstration:

How to configure /serveStaleOnError

Tip #4
Cache custom error pages

Tip #4: Implement Proper Error Handling

- Problem:
  - By default, the content served for custom error pages such as the 404 and 500 are not cached
- Solution:
  - Have Apache Web server handle serving the error page content instead

Tip #4: Implement Proper Error Handling

- Do the following to configure this:

1. In the Apache configuration, set **DispatcherPassError 0**
2. Configure error pages for each error code:
   For example, here is a snippet from an Apache httpd.conf file:

```
<LocationMatch \.html$>
  ErrorDocument 404 /errorpages/404.html
  ErrorDocument 500 /errorpages/500.html
</LocationMatch>
ErrorDocument * /errorpages/blank.html
```

3. In CQ, overlay the error handler script so that it doesn't do an authentication check for a non-existent page. Modify **/libs/sling/servlet/errorhandler/404.jsp** and **/libs/sling/servlet/errorhandler/Throwable.jsp**

- See here for related Apache documentation
  http://httpd.apache.org/docs/2.2/mod/core.html#errordocument

Tip #4: Implement Proper Error Handling

Demonstration:

How to implement error handling

Tip #5
Block unwanted requests at the dispatcher level

Tip #5: Filter Out Invalid Requests

- Fine tune your dispatcher.any **/filter** section so that you block as much as possible.
  - Use a “whitelist” configuration
  - Deny everything and only allow what you need

Tip #5: Filter Out Invalid Requests

Demonstration:

How to configure /filter section

Tip #6
Block unwanted requests in publish

Tip #6: Only Allow Valid Selectors, Suffixes, and Querystrings

- Implement a javax.servlet.Filter to return 404 or 403 when an invalid selector, querystring params or suffix path is used against your publish instance.
  - This will prevent caching of the unwanted requests.
  - Once again use a “whitelist” approach, block everything and only allow only the selectors, querystring params, and suffix paths you want.
  - A sample implementation is provided [here]

Tip #6: Filter Out Invalid Requests

Demonstration:

How to implement a blocking javax.servlet.Filter

Tip #7
Configure dispatcher to ignore invalid querystrings

Tip #6: Ignore Unused Querystring Params

- If the URL has a querystring then page will not be cached.
- However, you can define exclusions using the /ignoreUrlParams
- Allow all invalid querystrings to be ignored and “deny” all valid query strings from being ignored.
- This will cause invalid querystrings to be removed from the url before reaching publish.
- Here is an example configuration that ignores all querystrings except for the param “q”:

```
/ignoreUrlParams
{
   /0001 { /glob "*" /type "allow" }
   /0002 { /glob "q" /type "deny" }
}
```

- For example, with this configuration, when **/content/foo.html?test=1** is requested the page would get cached and the publish instance would only see the request **/content/foo.html**
  However, for request **/content/foo.html?q=1** the request would not get cached and the publish instance would see **/content/foo.html?q=1**

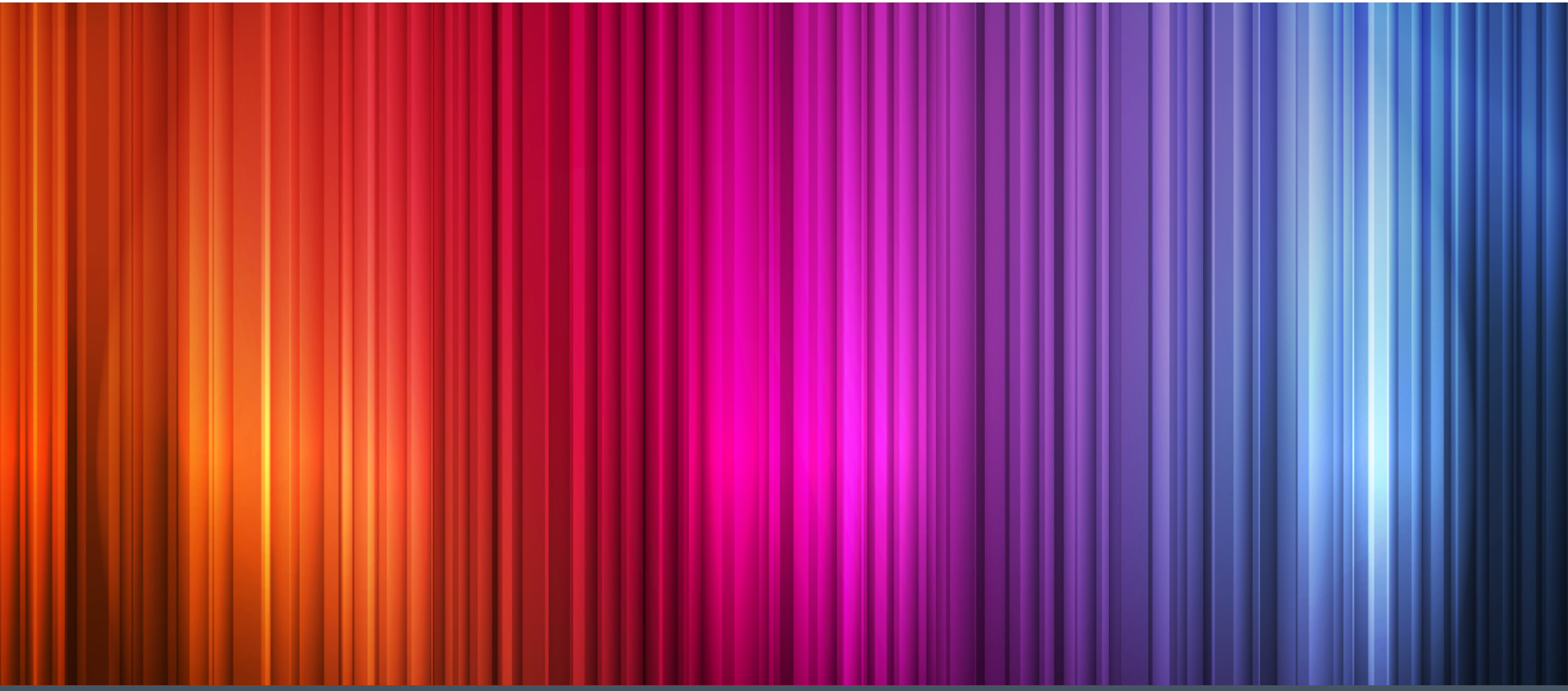
Tip #7: Ignore invalid querystring parameters

Demonstration:

How to conifgure /ignoreUrlParams

How to Design your Site with Caching in Mind

Building an effective cache strategy.



Developing a Cache Strategy

- Caching is something that should be considered during the design and development phases of your project.
- In order to leverage the dispatcher cache to its full extent, your application must be designed to be “cacheable”.

Basic Cache Strategy

- Avoid use of URL querystring parameters
- Use selectors in URLs where there is cacheable content and a limited set of possibilities
  - Ex.
    - Img.thumb.jpg
    - img.small.jpg
    - Img.large.jpg
- If the site is personalized, use sticky sessions on your load balancer
- Maximize caching by using Ajax and SSI...

Using Ajax and SSI

- If you call the component's content path directly then you can request it with a .html extension.
- For example: http://localhost:4502/content/foo/_jcr_content/par/textimage1.html
- We can use this for flexible caching:
  - Using SSI (Server Side Includes) or Ajax
- This allows us to load the non-changing parts of the page from the cache and load dynamic sections separately. This will help improve page load times.
- This is especially useful in sites that use a lot of personalization.

Server Side Includes

- **How to use SSI in Apache**
- http://httpd.apache.org/docs/2.2/howto/ssi.html
- Set this configuration in your Apache httpd configuration within the VirtualHost for your CQ site:

  Options +IncludesNOEXEC
  AddOutputFilter INCLUDES .html

  Note: We use IncludesNOEXEC and not just Includes for security reasons. Using the option +Includes may open up your web server to remote execution vulnerabilities.
- In your application code, only allow ssi selector on components where you allow ssi calls. For example, on components where you would like to request via SSI, you could rename html.jsp to ssi.html.jsp and create an html.jsp file like this:

  <!--#include virtual="<%= resource.getPath() %>.ssi.html" -->

SSI

- **How to use Ajax to Load Specific Components**
- Many different ajax libraries will work for this. However, I will demonstrate using jquery:
  - Rename your component's html.jsp file to html.ssi.jsp
  - Create a new html.jsp script so that it only does the ajax call. For example:

```
<div id="<%= divId %>”></div>
<script>
    $(document).ready(function(){
        $("#<%= divId %>").load("<%= resource.getPath() %>.ssi.html");
    });
</script>
```

Combining Ajax and SSI

- Let's say you are using SSI for serving content through the webserver, but you still want to be able to view the page if you access CQ directly. We can handle this case by looking at the X-Forwarded-For header to verify whether or not the request is coming from the dispatcher and handle it appropriately. Following our SSI and Ajax examples above, here is what your component's html.jsp would look like:

```
<% if(request.getHeader("X-Forwarded-For") != null) { %>
   <!--#include virtual="<%= resource.getPath() %>.ssi.html" -->
<% } else {
   <div id="<%= divId %>”></div>
   <script>
        $(document).ready(function(){
           $("#<%= divId %>").load("<%= resource.getPath() %>.ssi.html");
        });
   </script>
<% } %>
```

Ajax and SSI Example

Demonstration

Q& A + References

- Dispatcher Documentation
  - http://dev.day.com/docs/en/cq/current/deploying/dispatcher.html
  - /cache section of dispatcher.any - http://dev.day.com/docs/en/cq/current/deploying/dispatcher/disp_config.html#par_146_44_0010
- Apache HTTP Server
  - Error Document Directive - http://httpd.apache.org/docs/2.2/mod/core.html#errordocument
  - SSI - http://httpd.apache.org/docs/2.2/howto/ssi.html
- Microsoft IIS
  - How to implement SSI in IIS
    - http://msdn.microsoft.com/en-us/library/ms525185%28v=vs.90%29.aspx
    - http://tech.mikeal.com/blog1.php/server-side-includes-for-html-in-iis7


Adobe