Responsive Design + Server Side Components (RESS)

Responsive Design Approaches

- There is no shortage of debate about the best way to develop Web sites that work well across many networked devices.
  - Some teams favor a client-side approach
  - Some lean towards server-side solutions (e.g., Adaptive Images)
  - Others are interested in solutions that try to bring together the best of both worlds.
- RESS (Responsive Web Design + Server Side Components) is one such proposal.

Responsive Design Approaches

- In Responsive Web Design implementations,
  - Web URLs are consistent across devices and
  - Adapt their content based on the capabilities of the browser in which they are displayed.
- This means the same hyperlink can deliver a great experience across a wide range of devices.
  - We know that this has a negative impact on the use of CDNs which in turn impacts performance

Responsive Design Approaches

- In order to adapt effectively, Web sites generally need
  - a single source order of markup,
  - a single URL structure, and
  - flexible media that can scale across screen sizes (which can become a performance challenge).

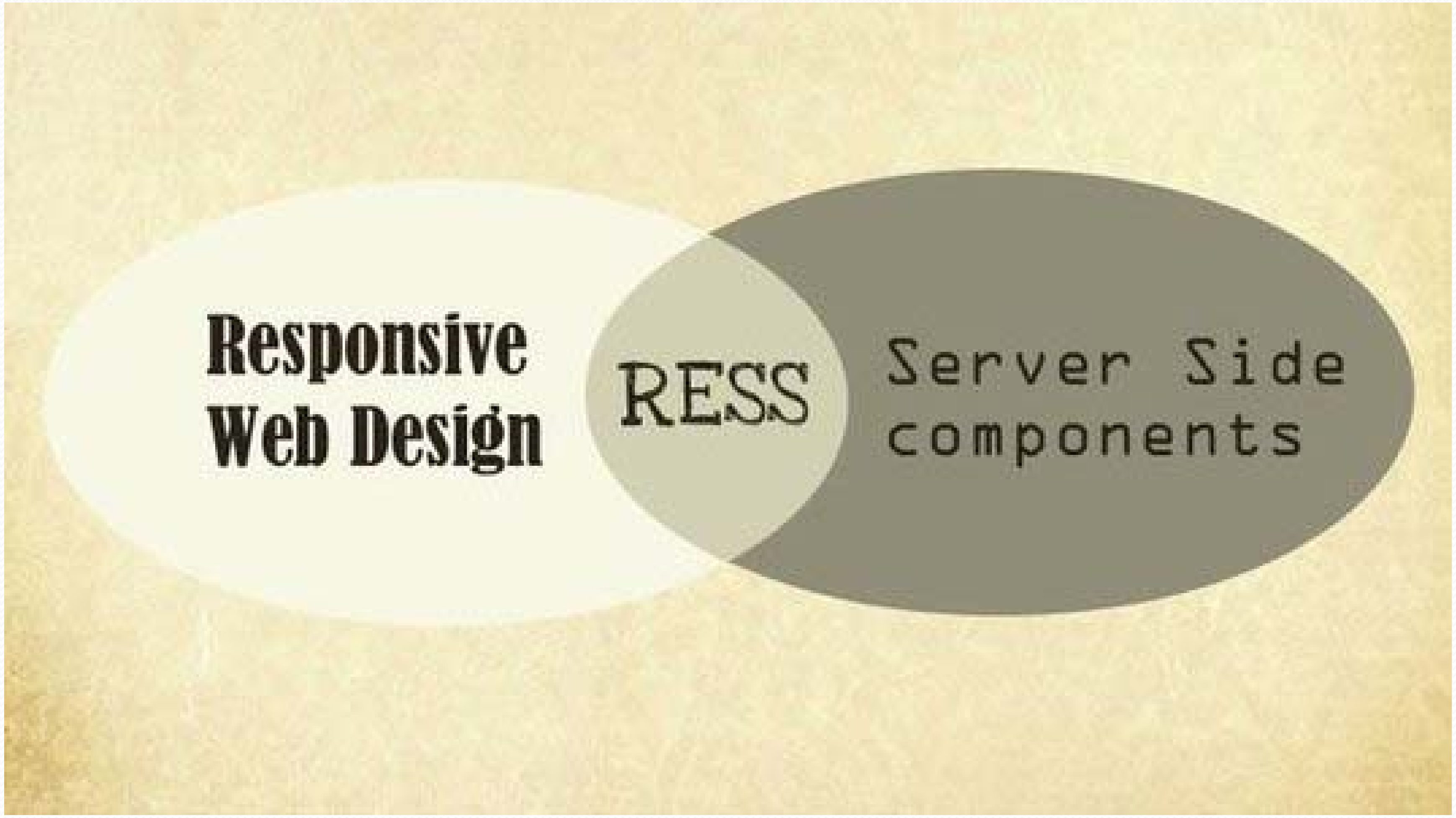
Responsive Design

- Server side solutions, on the other hand, only send what a client needs. This means that source order, URL structure, media, and application designs can be finely optimized for a specific device class before ever reaching its browser.
- But server-side solutions generally rely on user agent redirects to device-specific code templates. Each device class that warrants adaptation needs its own set of templates and these templates may ultimately contain duplicative code that actually applies to every class of device.

Responsive Design

- So both client and server side adaptations have benefits but a few limitations.
- How do we get the best of both worlds without the challenges that can hamper each?
  - RESS (Responsive Web Design + Server Side Components) is a concept that looks to address this.

RESS



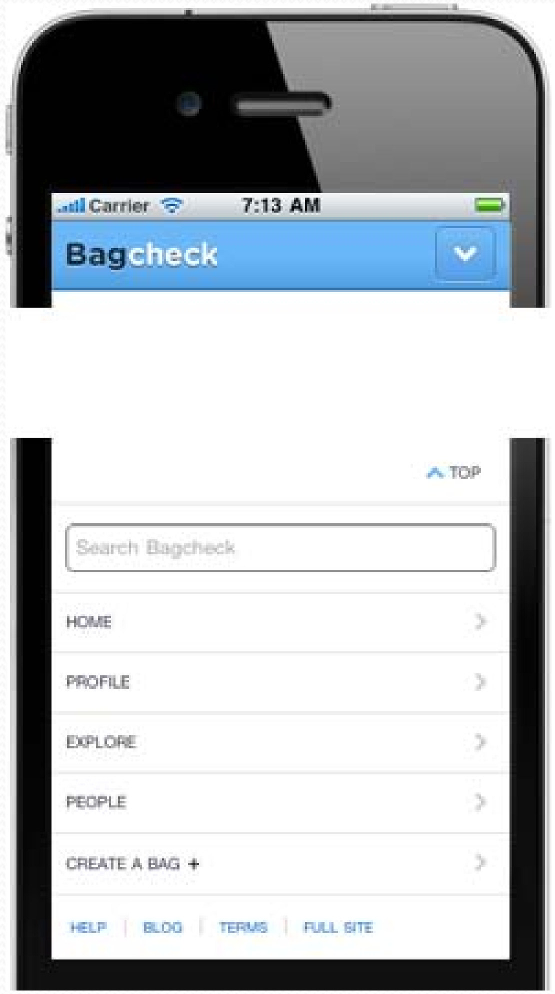
RESS

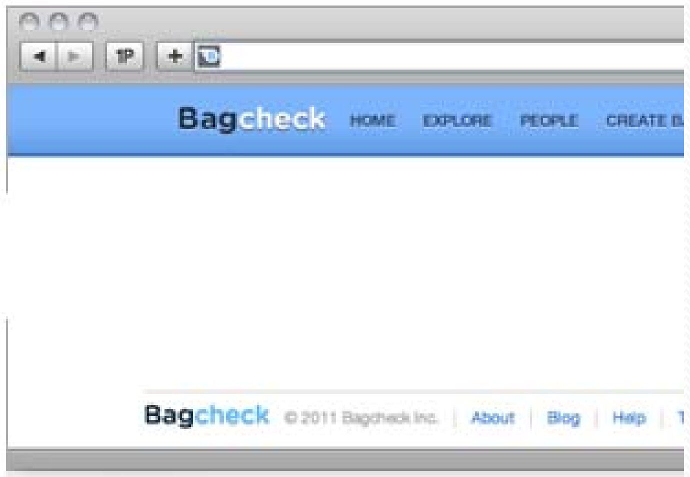
- Responsive Design + Server Side Components (RESS) is being called Next Generation Responsive Design
  - Term was coined in 2011
- In a nutshell, RESS combines adaptive layouts with server side component (not full page) optimization.
- A single set of page templates define an entire Web site for all devices but key components within that site have device-class specific implementations that are rendered server side.

RESS – How it Works

- Let us assume we want a different navigation solution for mobile and desktop devices.
    - Because screens are small on mobile, we want a minimal header that does not take space away from content.
    - But we need to allow people to navigate the site in a comfortable way so we will position the navigation links at the bottom of the page where accessing them tends to be easier with one-handed use.
    - On the desktop, however, we want the same navigation at the top of the page. There is plenty of room for the content and we can expose all our menu choices to give people a sense of what's on the site.

Example Navigation





RESS – How it Works

- To implement this with RESS I wll use two variables in our page template (defined here using Mustache) for the header and footer.

```
<body>   {{>header}}
    [...document content...]
  {{>footer}} </body>
```

RESS – How it Works

- Here the entire page is coded using flexible grids and media queries to manage multiple resolution breakpoints thereby taking advantage of what Responsive Web Design does well.
- But each of the variables is a component that has device class optimized implementations for it on the server. In this case:

```
header.html mobile_header.html footer.html
 mobile_footer.html
```

Feature Detection and Device Detection

- Modernizr (http://modernizr.com) is a feature detection framework that makes it easy to detect browser features.
- It simply runs a test in the browser to get a boolean answer as output: “does X work?” and the answer is mostly “true” or “false.”

Modernizer Example

```
Modernizr.load({
  test: Modernizr.geolocation,
  yep : 'geo.js',
  nope: 'geo-polyfill.js'
});
```

Modernizer Example

- In this example, we decided that we should load a different script depending on whether `geolocation` is supported in the host browser or not.
- By doing this, you save users from having to download code that their browser does not need.
- This increases page performance, and offers a clear place to build a healthy amount of abstraction to your polyfills (both '`geo.js`' and '`geo-polyfill.js`' would seem the same to the rest of the app, even if they are implemented differently)

Polyfills

- Definition
  - a ***polyfill*** (or ***polyfiller***) is downloadable code which provides facilities that are not built into a web browser.
  - It implements technology that a developer expects the browser to provide ***natively*** providing a more uniform API landscape.

- For example, many features of `HTML5` are not supported by versions of Internet Explorer older than version 8 or 9, but can be used by web pages if those pages install a polyfill

Device Detection

- Device detection on the other hand, is something different. It all happens on the server
  - Analyses the HTTP header of the device.
  - It then looks up in a database of known devices and return a set of capabilities for that device.

Device Detection

- The beauty of this is that it is a database of information that is collected and maintained by humans and it can hold incredibly detailed information about capabilities that is currently impossible to feature test.
  - Examples include device type (desktop, TV, mobile, tablet), device marketing name, video codec support and so on.

Device Detection

- Device Detection Frameworks
  - WURFL (http://scientiamobile.com/cloud)

- A free and for fee service

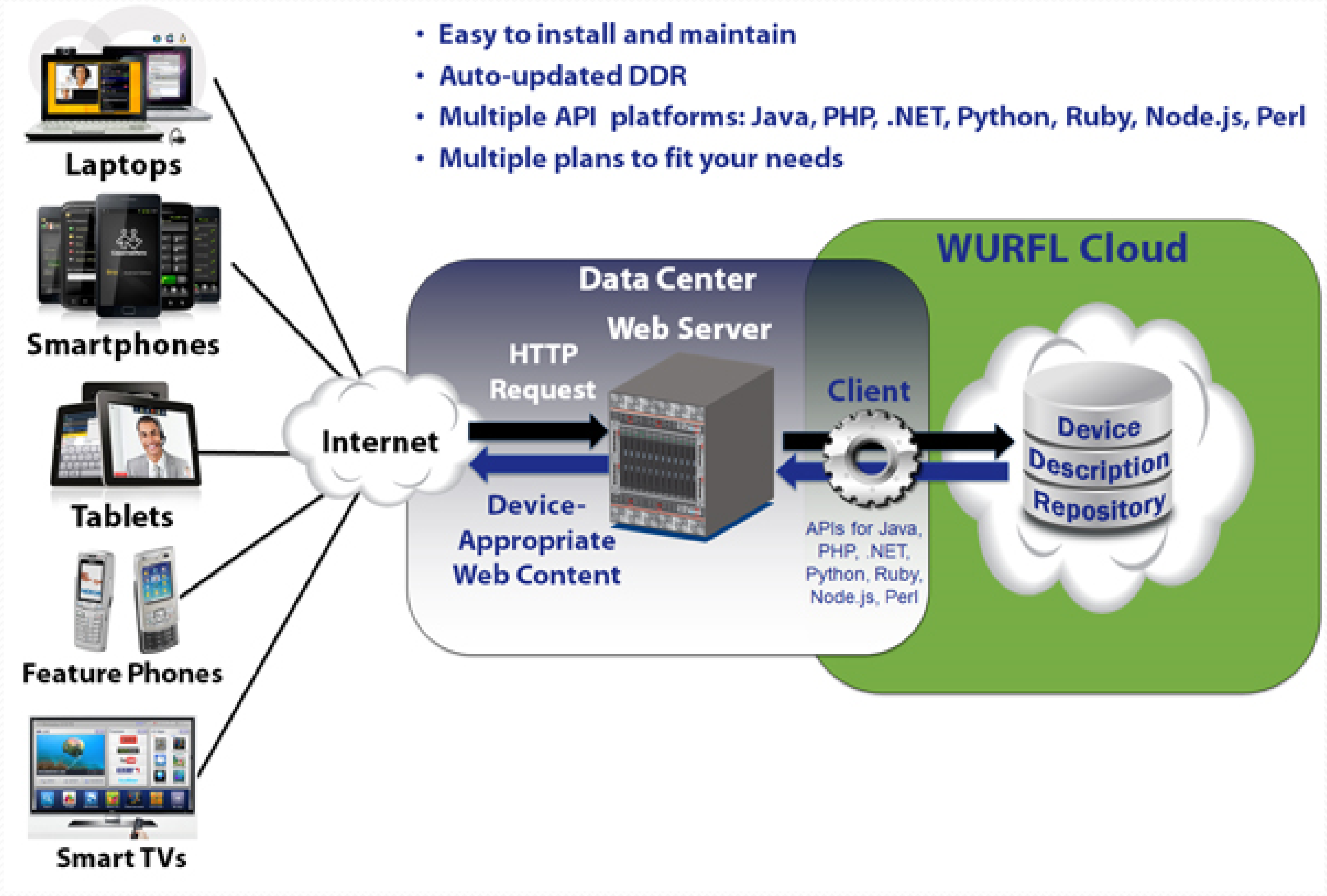
RESS – Device Detection

- Sample Site
  - http://andmag.se/ress/

- Visit this site using various devices
  - Desktop
  - Smart Phone – try several
  - Tablet

WURFL

- Easy to install and maintain
- Auto-updated DDR
- Multiple API platforms: Java, PHP, .NET, Python, Ruby, Node.js, Perl
- Multiple plans to fit your needs


Laptops
Smartphones
Tablets
Feature Phones
Smart TVs
Internet
Data Center
Web Server
HTTP Request
Device-Appropriate Web Content
Client
APIs for Java, PHP, .NET, Python, Ruby, Node.js, Perl
WURFL Cloud
Device Description Repository

Getting Started with WURFL

- http://scientiamobile.com/wurflCloud/gettingStarted

Device Detection with WURFL

- WURLF stands for stands for Wireless Universal Resource File
- WURFL is a Device Description Repository (DDR) which serves as a central source of device information for mobile web applications. It contains descriptions of thousands of mobile devices that can be used in any application, free or commercial

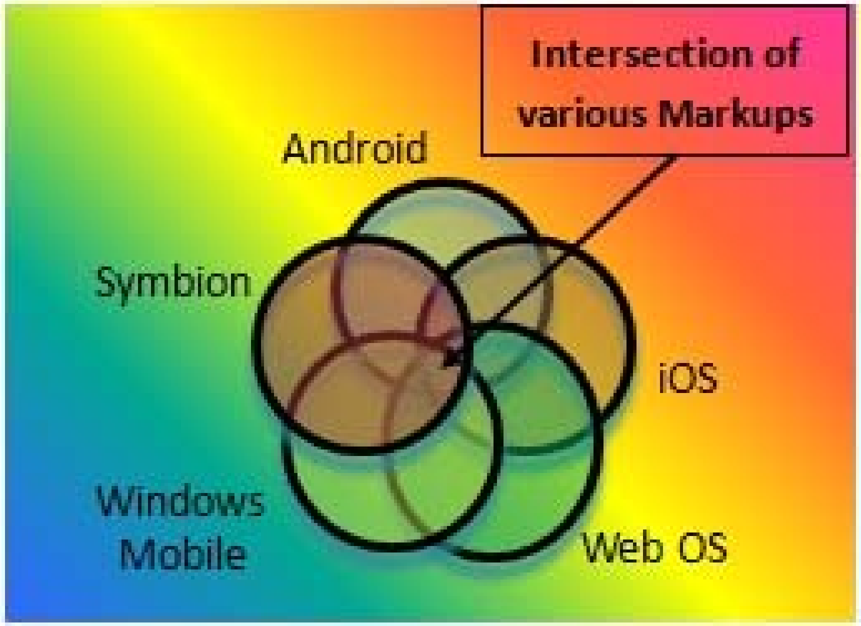
Device Detection with WURFL

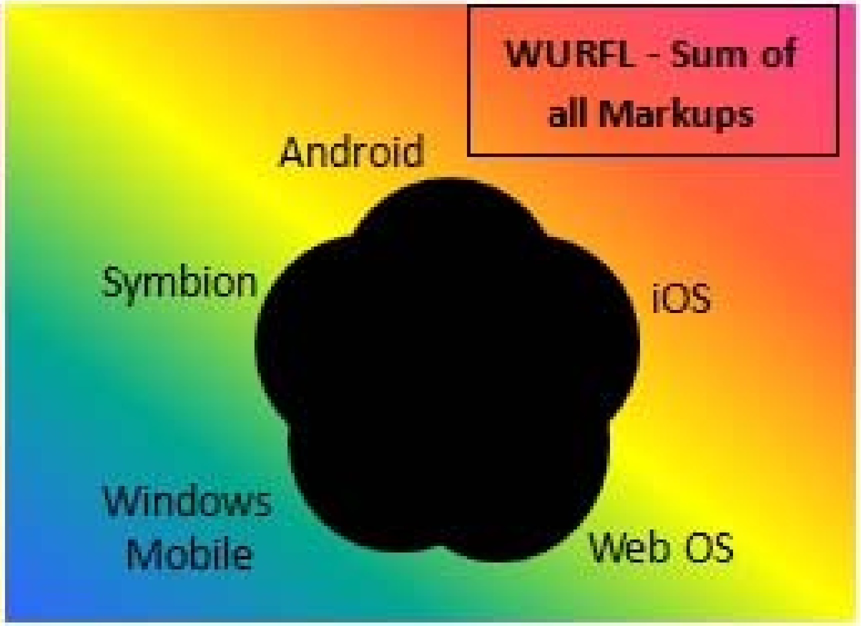
- WURFL is recognized as the de-facto standard in the area of Device Description Repositories.
- The WURFL project began in 2002, with the intention of tackling the issues of mobile device fragmentation. It seeks to do so by providing developers across the world with a common set of device information.

Device Detection with WURFL

- WURFL has the following editions:
  - Java Edition
  - PHP Edition
  - .Net Edition (Also called NuGet)
  - Database Edition
  - JavaScript
  - Perl

Device Detection with WURFL





Device Detection with WURFL

- The figure on the left shows the different markup languages.
- Since the coders design the application to support all the available devices, the available domain for coding is the area of intersection. Due to introduction of more and more devices and the difference in device support and features, this region generally becomes smaller and smaller.

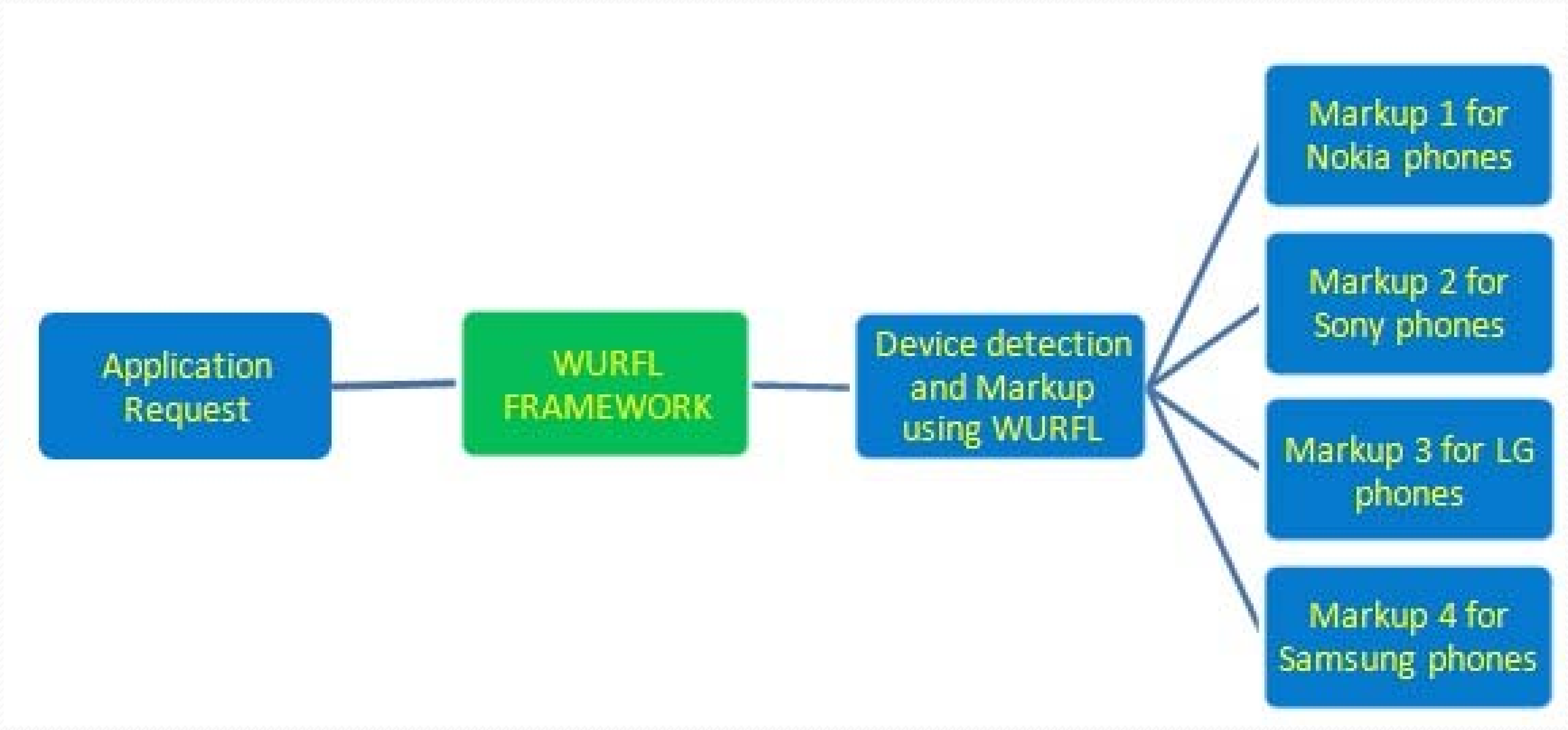
Device Detection with WURFL

- WURFL addresses this issue by detecting the device type and then delivering suitable markup. The figure on right demonstrates the available markup range for the coders to develop, which is the sum of all the markup regions.
- WURFL contains 7,000+ unique devices and thousands of firmware variations, representing nearly every mobile device on the market worldwide!

WURFL Flow

- Once an application request from a device reaches the web server, the WURFL framework detects the device type and produces markup generated specifically for the device.

WURFL Flow



Application Request
WURFL FRAMEWORK
Device detection and Markup using WURFL
Markup 1 for Nokia phones
Markup 2 for Sony phones
Markup 3 for LG phones
Markup 4 for Samsung phones

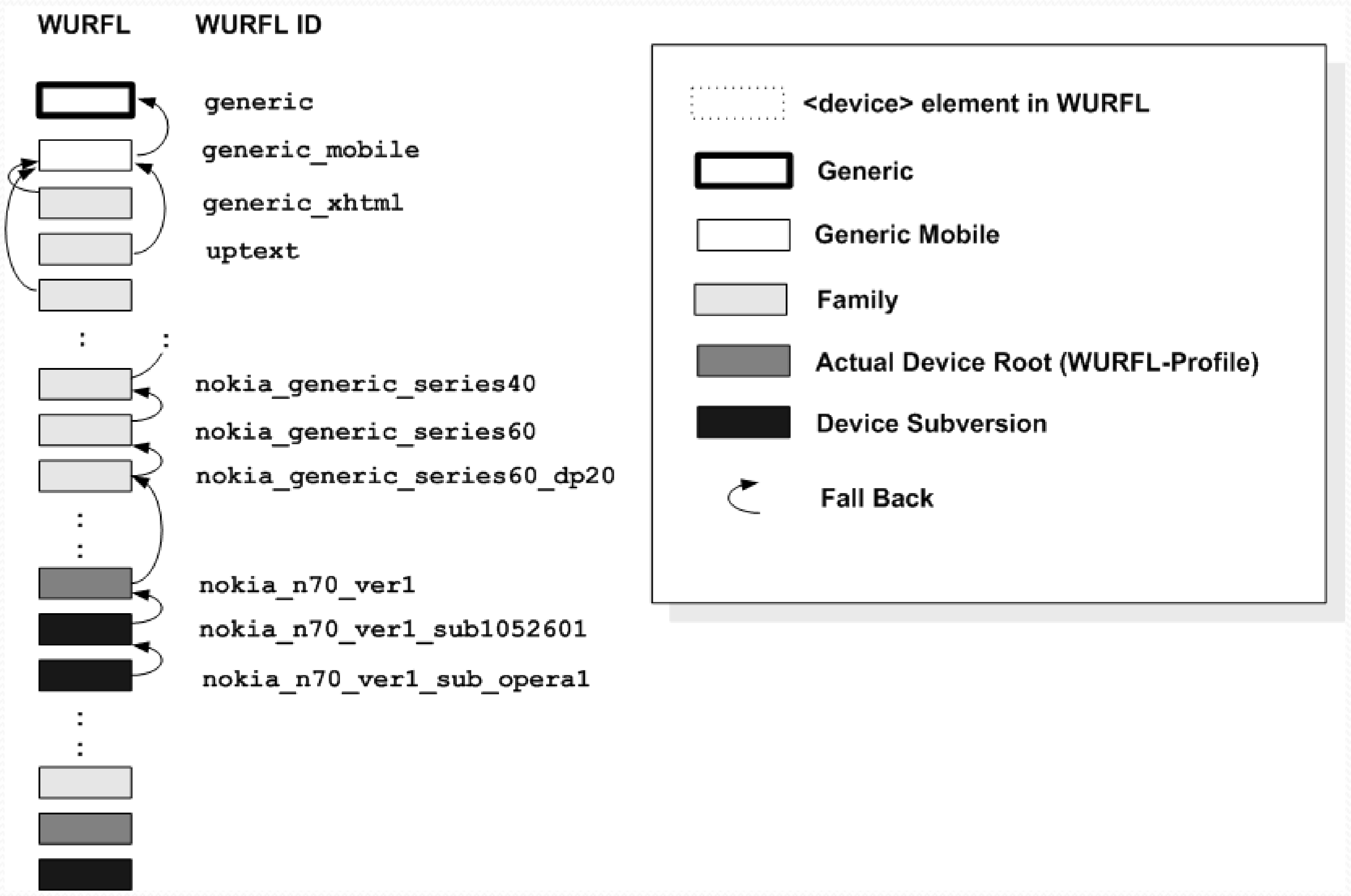
WURFL Delivery Formats

- WML – Wireless Markup Language
- XHTML - Extensible HypeText Markup Language
- C-HTML – Compact HTML

WURL Device & WURL Capabilities

- http://wurfl.sourceforge.net/help_doc.php

The WURFL Device Hierarchy



Java Code Example

- Download from
  - http://sourceforge.net/projects/wurfl/files/WURFL%20Java%20API/1.5/
- Unpack the package into your tomcat `webapps/` directory
- hit `http://yourserver[:port]/wurfl-helloworld-{servlet|spring}-{version}/` with your browser.

Code Dissected

```
import java.io.IOException;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import net.sourceforge.wurfl.core.Device;
import net.sourceforge.wurfl.core.MarkUp;
import net.sourceforge.wurfl.core.WURFLEngine;
import net.sourceforge.wurfl.core.exc.CapabilityNotDefinedException;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
```

Code Dissected

```
public class HelloWorld extends HttpServlet {
    /** Serial */
    private static final long serialVersionUID = 10L;

    private static final String XHTML_ADV = 
"xhtmladv.jsp";
    private static final String XHTML_SIMPLE = 
"xhtmlmp.jsp";
    private static final String CHTML = "chtml.jsp";
    private static final String WML = "wml.jsp";

    /** Logger */
    private final Logger log = 
LoggerFactory.getLogger(getClass());
```

Code Dissected

```
protected void doGet(HttpServletRequest request,
                     HttpServletResponse response)
              throws ServletException, IOException {
        processRequest(request, response);
}

protected void doPost(HttpServletRequest request,
                      HttpServletResponse response)
              throws ServletException, IOException {
        processRequest(request, response);
}
```

Code Dissected

```
protected void processRequest(HttpServletRequest  request,
                       HttpServletResponse response)
              throws ServletException, IOException {
        WURFLEngine engine = (WURFLEngine)
getServletContext().getAttribute(WURFLEngine.class.getName
());

        Device device =
engine.getDeviceForRequest(request);

        log.debug("Device: " + device.getId());
        log.debug("Capability: " +
device.getCapability("preferred_markup"));
```

Code Dissected

```
String jspView = null;

if (MarkUp.XHTML_ADVANCED.equals(markUp)) {
     jspView = XHTML_ADV;
} else if (MarkUp.XHTML_SIMPLE.equals(markUp)) {
     jspView = XHTML_SIMPLE;
} else if (MarkUp.CHTML.equals(markUp)) {
     jspView = CHTML;
} else if (MarkUp.WML.equals(markUp)) {
     jspView = WML;
}

log.info("View: " + jspView);
```

Code Dissected

```
 if (markUp == MarkUp.XHTML_ADVANCED ||
     markUp == MarkUp.XHTML_SIMPLE) {
     String contentType = "text/html";
     try {
        contentType =
device.getCapability("xhtmlmp_preferred_mime_type");
     } catch (CapabilityNotDefinedException e) {
       throw new RuntimeException("Somethingh is seriously
wrong with your WURFL:" + e.getLocalizedMessage(), e);
     }
     request.setAttribute("contentType", contentType);
     log.debug("ContentType: " + contentType);
}
```

xml.com - Listener

- The class is called
  `net.sourceforge.wurfl.core.web.WURFLServletContextListener`
  and it will take care of configuring
  `net.sourceforge.wurfl.core.WURFLEngine`
  as a wurflEngine.

```
<listener>
  <listener-class>
net.sourceforge.wurfl.core.web.WURFLServletC
ontextListener
  </listener-class>
</listener>
```

- In addition to this class, the `WURFLServletContextListener` will also load the `wurfl.zip` file by reading the "`wurfl`" parameter from the web.xml itself.

```
<context-param>
  <param-name>wurfl</param-name>
  <param-value>/WEB-INF/wurfl.zip</param-value>
</context-param>
```

web.xml - Snippet

- New capability filter feature can also be configured directly from the web.xml; capability names can be specified into a context parameter value, simply called capability-filter. Capabilities need to be specified one-per-line without punctuation marks, like this:

web.xml - Snippet

```
<context-param>
    <param-name>capability-filter</param-name>
    <param-value>
        device_os
        device_os_version
        is_tablet
        is_wireless_device
        mobile_browser
        mobile_browser_version
        pointing_method
        preferred_markup
        resolution_height
        resolution_width
        ux_full_desktop
        xhtml_support_level
    </param-value>
</context-param>
```

xhtmladv.jsp - snippet

```
<body>
        <h1>Hello From XHTML ADVANCED</h1>
        <p>ContentType: <%=request.getAttribute("contentType")
%></p>
        <p>Device:
<%=((Device)request.getAttribute("device")).getId() %></p>
        <p><b>Virtual capabilities:</b></p>
        <p>Is smartphone:
<%=((Device)request.getAttribute("device")).getVirtualCapability(
"is_smartphone") %></p>
        <p>Device OS:
<%=((Device)request.getAttribute("device")).getVirtualCapability(
"advertised_device_os") %></p>
        <p>Is Android:
<%=((Device)request.getAttribute("device")).getVirtualCapability(
"is_android") %></p>
        <img src="images/logo.gif" width="59" height="76"
alt="logo"/>
</body>
```