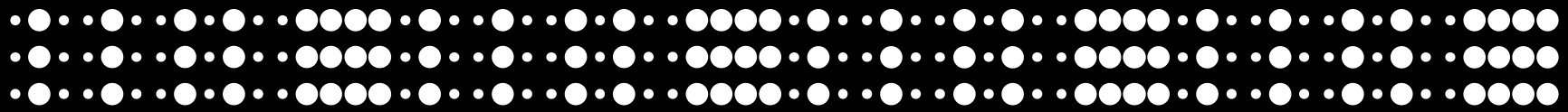


Faster-than-possible Code:

Deferred Binding in GWT

Bruce Johnson
May 2008

“GWT's mission is to radically improve the web experience for users by enabling developers to use existing Java tools to build no-compromise AJAX for any modern browser.”

“Making GWT Better”

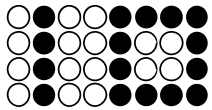
“We definitely do not view development in GWT as a form of compromise. We firmly believe that GWT should generate better JavaScript code than you would write by hand, and will generally choose to avoid making concessions to convenience if they hurt the performance of the resulting AJAX code. ”

“Making GWT Better”

“We want great results, where *great* is defined by how much it benefits end users.”

“It's much better to provide a solid platform so that other great libraries can be built on top of GWT rather than trying to be all things to all people out of the box.”

“Making GWT Better”



No-compromise Ajax

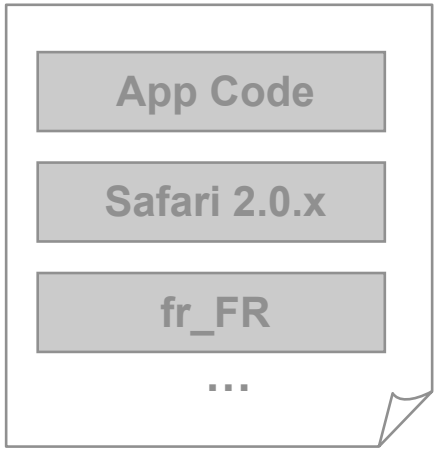
The Premise of Deferred Binding

- A modest goal: the best attainable performance
  - As determined by the end user
  - Fastest possible startup
  - Fastest possible execution
- Idea: send exactly the right code for a user's circumstances
  - User agent
  - Locale
  - Debug vs. production
  - Network characteristics
  - <anything that might make a difference>
- One single script cannot satisfy all these, so...

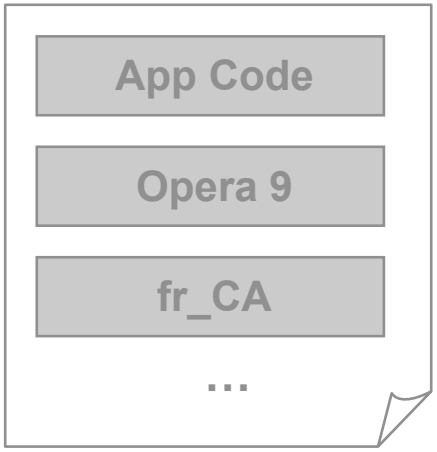
Modularize, Right?

User 1

User 2



User 3

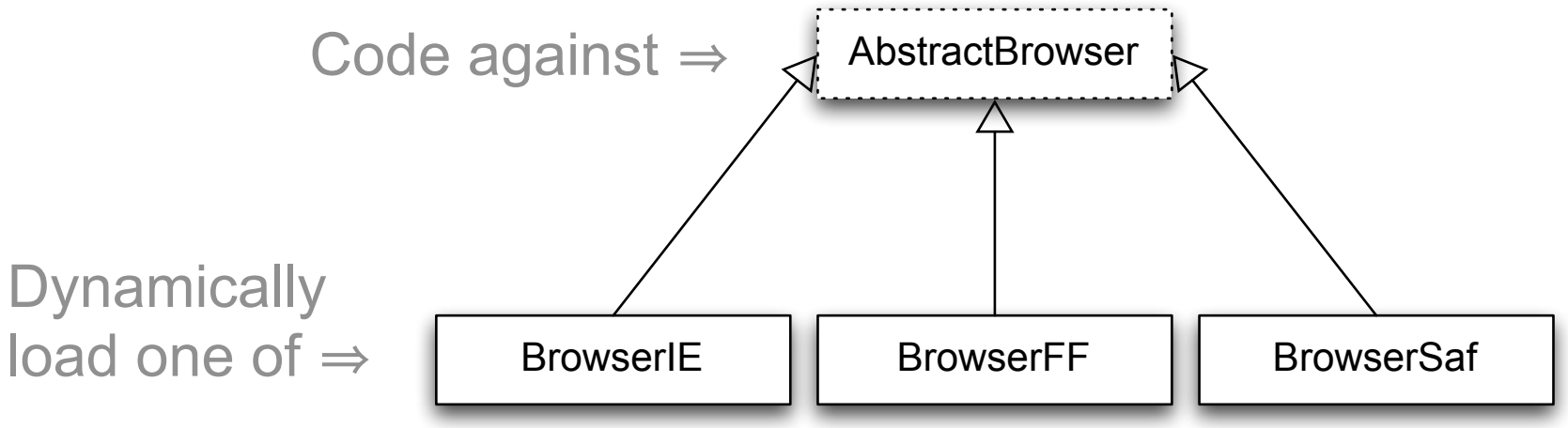


User 4

The Obvious Way to Modularize

...that sucks

- Wait! I know this pattern! Service Provider Interface (SPI)!
- Create a nice common interface
- Load the right implementation dynamically



The First Rule of GWT Club is...

- Rule #1: Make Fewer HTTP Requests
  - Steve Souders, “High Performance Web Sites”
  - Look at GWT Mail sample
- Dynamic modules over HTTP is calamity
- Slow to start
  - Awful network utilization
  - Particularly horrible for mobile
- Slow to execute
  - Requires polymorphic dispatch at runtime
  - Impossible to eliminate dead code
- Hard to maintain
  - “DLL Hell” for the web
  - Synchronous XHR is evil

Another Obvious Solution

...that also sucks

- Wait! I know have a better idea: capability detection!
- Don’t test for specific user agents at all...how elegant!

```
function setInnerText(elem, text) {
  if (hasInnerText) {
    // The fast way
    elem.innerText = text;
  } else {
    // The slow way
    while (elem.firstChild) {
      elem.removeChild(elem.firstChild);
    }
    if (text != null) {
      elem.appendChild(document.createTextNode(text));
    }
  }
}
```

The Second Rule of GWT Club is...

- Make Fewer HTTP Requests
- But I digress

This Path is Fraught With Peril

- Naive capability detection causes repeated if/then checks

```
function crossBrowserMethod() {
  if (hasDesiredCapability)
  { /* do it the good way */ }
  else
  { /* do it the Max Power way */ }
}
```

- But you're smarter than that, so:

```
function oneTimeSetup() {
  if (hasDesiredCapability) {
    browser.prototype.crossBrowserMethod = theRightWay;
  } else {
    browser.prototype.crossBrowserMethod = theMaxPowerWay;
  }
}
```

The Third Rule of GWT Club Is...

- “The fastest code is that which does not run.”
  - Joel Webber, GWT co-creator and performance zealot
- Back to innerText...when possible, you really *want* this:

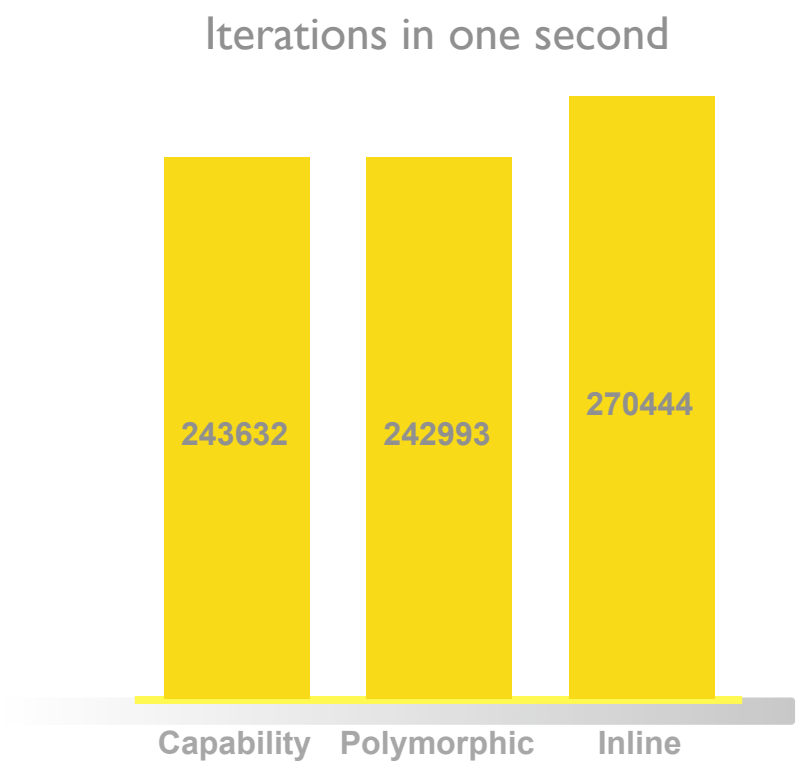
```
// ...lots of code...
elem.innerText = str;
// ...lots more code...
```

- And when that’s not possible, you want this:

```
// ...lots of code...
setInnerTextTheMaxPowerWay(elem, str);
// ...lots more code...
```

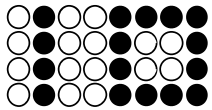
- You want *inlining*. Don’t pay for abstractions at runtime.
- Does it really matter?

When Scripting, Abstractions Cost Ya



- Equivalent setInnerText() behavior in all three implementations
- Negligible difference between polymorphism trick and capability detection
- Inlined code is ~10% faster in this case
- Other browsers have different behavior, which is exactly the point

Source: my MacBook Pro running Safari 3.1



Permutations

Help Wanted: Sane Programming Model

- You won't achieve speed in practice if it's too complicated
- What if we use good ol' polymorphism as a permutation axis?
- Example: DOMImpl
  - DOMImpl_InternetExplorer
  - DOMImpl_Mozilla
  - DOMImpl_WebKit
- That's exactly what GWT.create() does

GWT.create(T.class)

- Gets special treatment from the compiler
- Acts as a pivot point for permutations
- Gets rewritten in each permutation as new $T_{repl}()$
- `Browser b = GWT.create(Browser.class)` becomes
  - `Browser b = new Mozilla();`
  - `Browser b = new InternetExplorer();`
  - `Browser b = new WebKit();`
- This is why `T.class` must be a class literal

Digression: Monolithic Compilation

- Eschew dynamic loading! Just say no to runtime reflection!
- Ideal approach for compiler optimizations

```
Shape shape = new Square(4); // each side has length 4
int area = shape.getArea();  // polymorphic call
```

```
Square shape = new Square(4); // Square is known-final
int area = Square_getArea(shape); // static-ified call
```

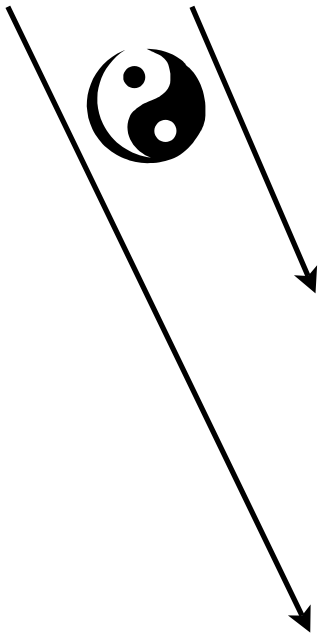
```
Square shape = new Square(4);
int area = shape.len * shape.len; // inlined!
```

And maybe in GWT 2.0...

```
int area = 16; // object existence elided completely
```

Branching Reality

```
abstract class DOMImpl {
  abstract void setInnerText(Element e, String s);
}
```



```
class DOMImplMozilla extends DOMImpl {
  native void setInnerText(Element e, String s)
  /*-{ e.textContent = s; }-*/;
}
```

```
class DOMImplInternetExplorer extends DOMImpl {
  native void setInnerText(Element e, String s)
  /*-{ e.innerText = s; }-*/;
}
```

Beauty! Independent Monolithic Compiles

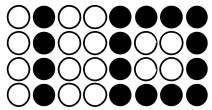
```
DOMImpl impl = GWT.create(DOMImpl.class);
// ... later ...
impl.setInnerText(x, “foo”);
```

<Mozilla version>.js

```
x.textContent = “foo”;
```

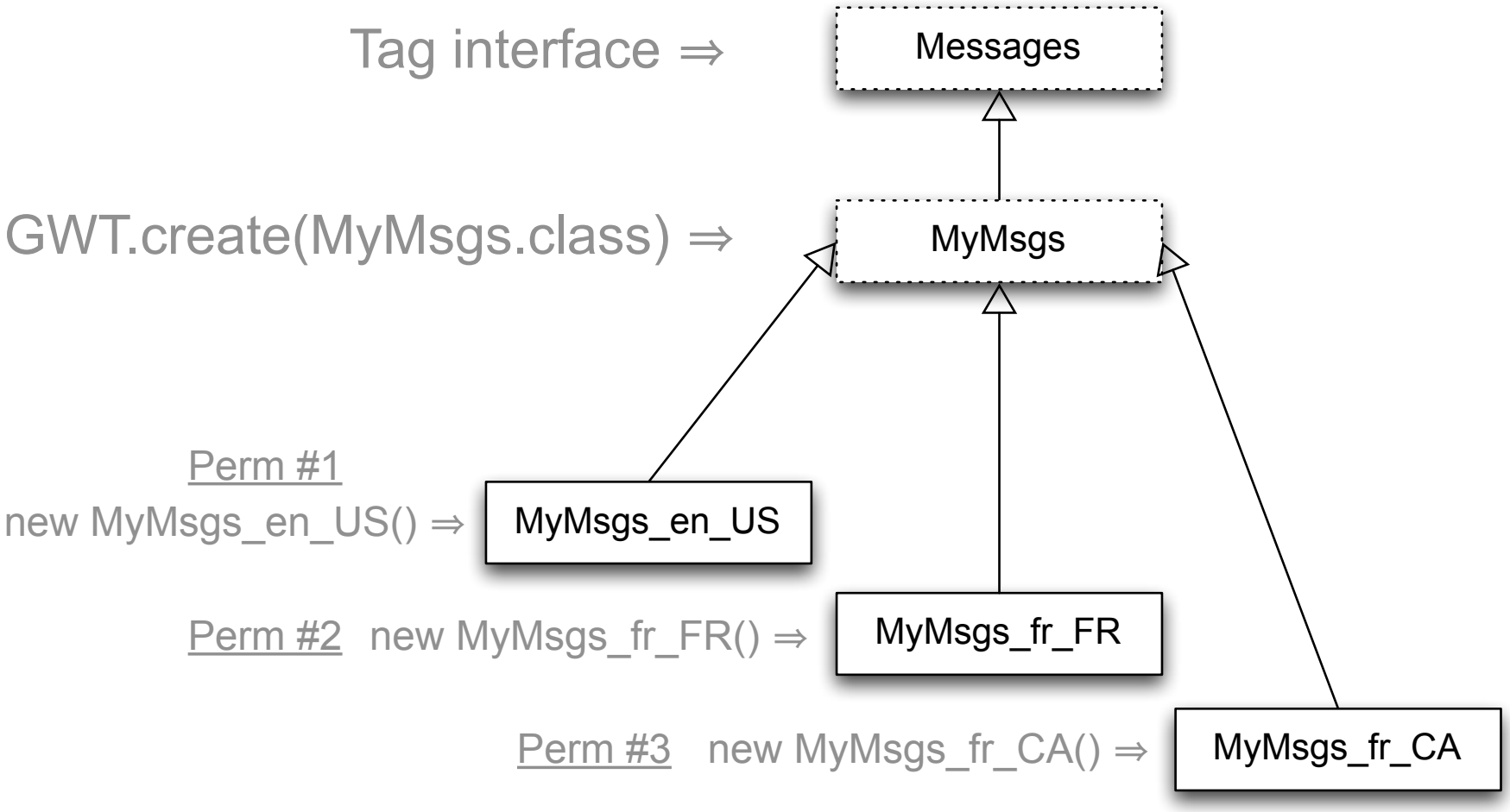
<Internet Explorer version>.js

```
x.innerText = “foo”;
```



Beyond User Agent

Locale, Also Nothing Special



Client Properties

- <define-property>

```
<define-property
    name="locale"
    values="en_US, en_UK"/>
```

- <extend-property>

```
<extend-property
    name="locale"
    values="fr_FR, fr_CA"/>
```

- <set-property>

```
<set-property
    name="locale"
    values="fr_CA"/>
```

Property Providers

- <property-provider> computes a value by startup sniffing...

```
<property-provider name="locale"><![CDATA[
  var locale;
  // Look for the locale as a url argument
  if (locale == null) {
    var args = location.search;
    // Look at query params...
  }
  if (locale == null) {
    // Look for the locale on the web page
    locale = __gwt_getMetaProperty("locale")
  }
  if (locale == null) {
    return "default";
  }
  // Iteratively rip off suffixes to find a close match.
  while (!__gwt_isKnownPropertyValue("locale",  locale)) {
    // Rip off everything after underscore...
  }
  return locale;
}]]></property-provider>
```

Resolving `GWT.create(T.class)`

- Rules defined in module XML consulted in reverse lexical order; first match wins
- Conditions
  - <when-type-is>, <when-type-assignable>, <when-property-is>
  - <all>, <any>, <none>
- Static substitution: <replace-with>
- Code generation: <generate-with>

```
<replace-with
    class="com.google.gwt.user.client.impl.DOMImplOpera">
  <when-type-is class="com.google.gwt.user.client.impl.DOMImpl"/>
  <when-property-is name="user.agent" value="opera"/>
</replace-with>
```

Compile-time Code Generation

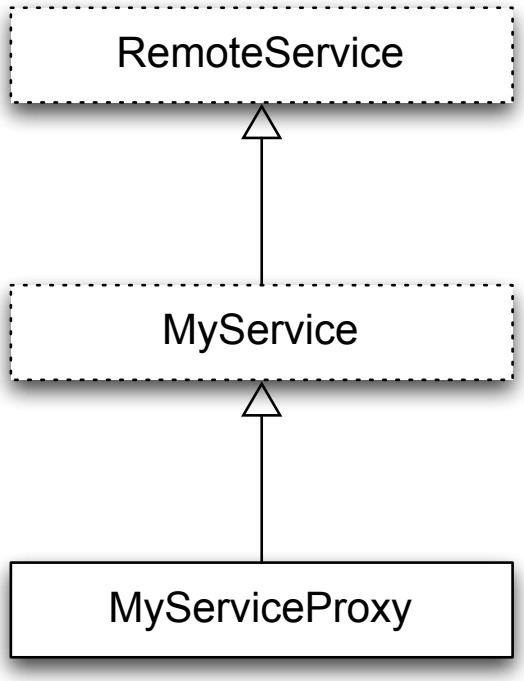
```
<generate-with class="ServiceInterfaceProxyGenerator">
  <when-type-assignable class="RemoteService"/>
</generate-with>
```

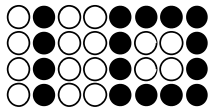
Used to trigger the rule ⇒

GWT.create(MyService.class) ⇒

⇓

new MyServiceProxy() ⇒





Maximum Startup Speed

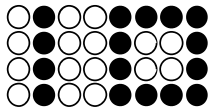
How a Permutation Gets Selected

- Two-phase startup
- Client requests the generated selection script
  - <module>.nocache.js
- Property providers run to decide property values
- Property values imply a permutation

```
map(['ar', 'ie6'], 'C423B4E263DBD2B411765BC573F4B95E.cache.html');
map(['en', 'ie6'], 'AFDD6408D742551B6E7A718388ACC024.cache.html');
map(['fr', 'ie6'], '856C7457EFD4634D93FF08D7B9D218B8.cache.html');
map(['zh', 'ie6'], 'E130C9E6D5962A222D7205C4517D0E9D.cache.html');
map(['ar', 'gecko'], '67F683D03A624E9EABC58780CEFB5F29.cache.html');
map(['en', 'gecko'], 'D903C36E26F86B1FA32E71ECF35524A4.cache.html');
map(['fr', 'gecko'], '3D1AB61674B879226E0479E77EDE9F77.cache.html');
map(['zh', 'gecko'], '63F25F00CB4F5589EEB846B399D12AB0.cache.html');
map(['ar', 'safari'], 'BAEDFF0347B3ED0A49DFF2865F0B6701.cache.html');
map(['en', 'safari'], 'ABD1C87060A7E92586938894CCC4DAC9.cache.html');
map(['fr', 'safari'], '01C1D7F45D4D431EF9049D83AA564583.cache.html');
map(['zh', 'safari'], '5059303563FF45A518434B71E68385BC.cache.html');
strongName = answers[computePropValue('locale')][computePropValue('user.agent')];
```

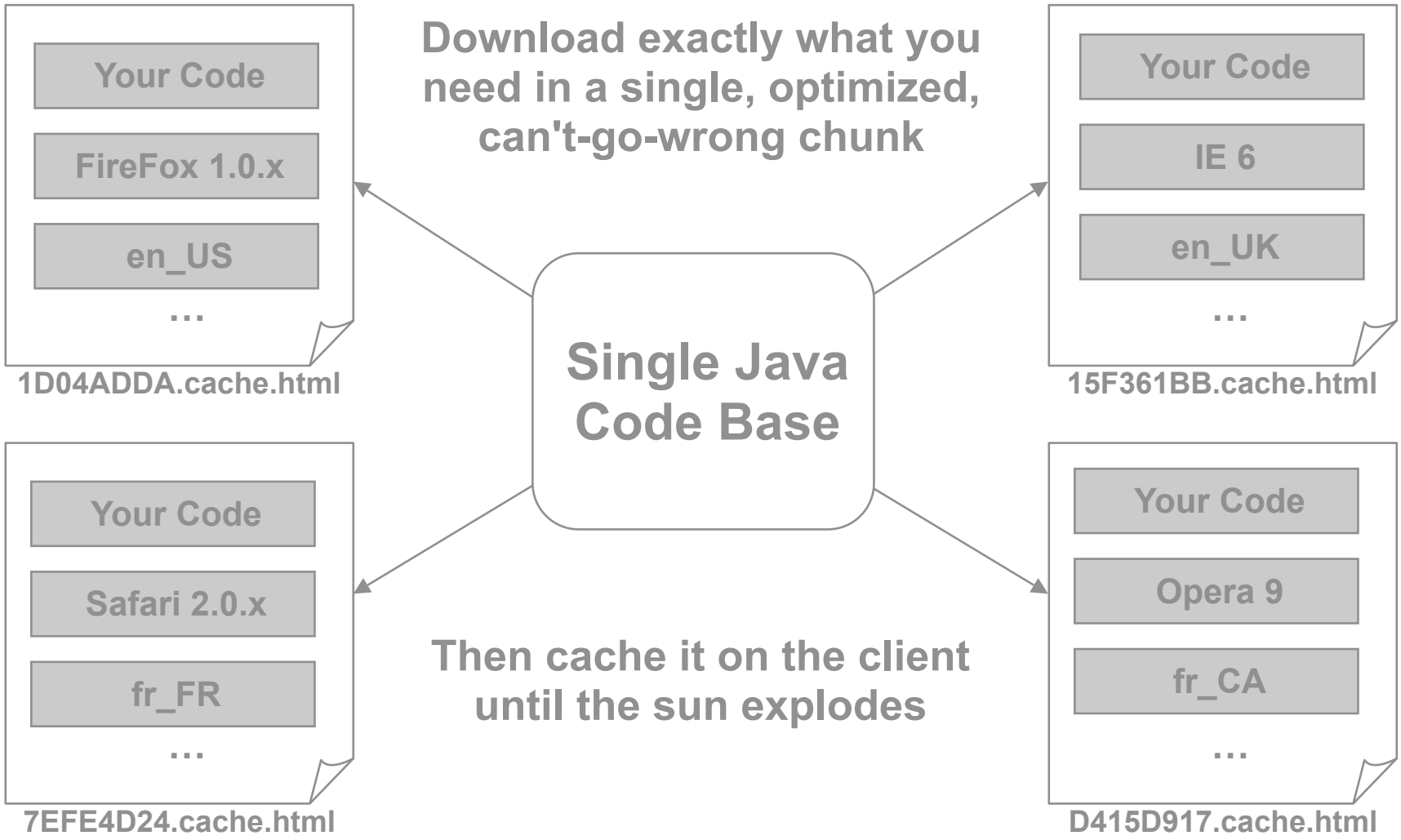
Perfect Caching

- The natural selection process provides a great opportunity for an efficient bootstrap mechanism
- <module>.nocache.js has must-revalidate semantics
  - But it's small
- <md5>.cache.js has cache-forever semantics
  - And it's big
- Perfect!
- Never fail to get the newest when it has been updated
- Never even ask if it hasn't been updated
  - That's right: not even an If-Modified-Since check
- For super-cool people...
  - Inline the selection script into the host page



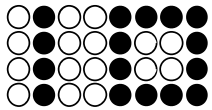
Summary

Putting it All Together

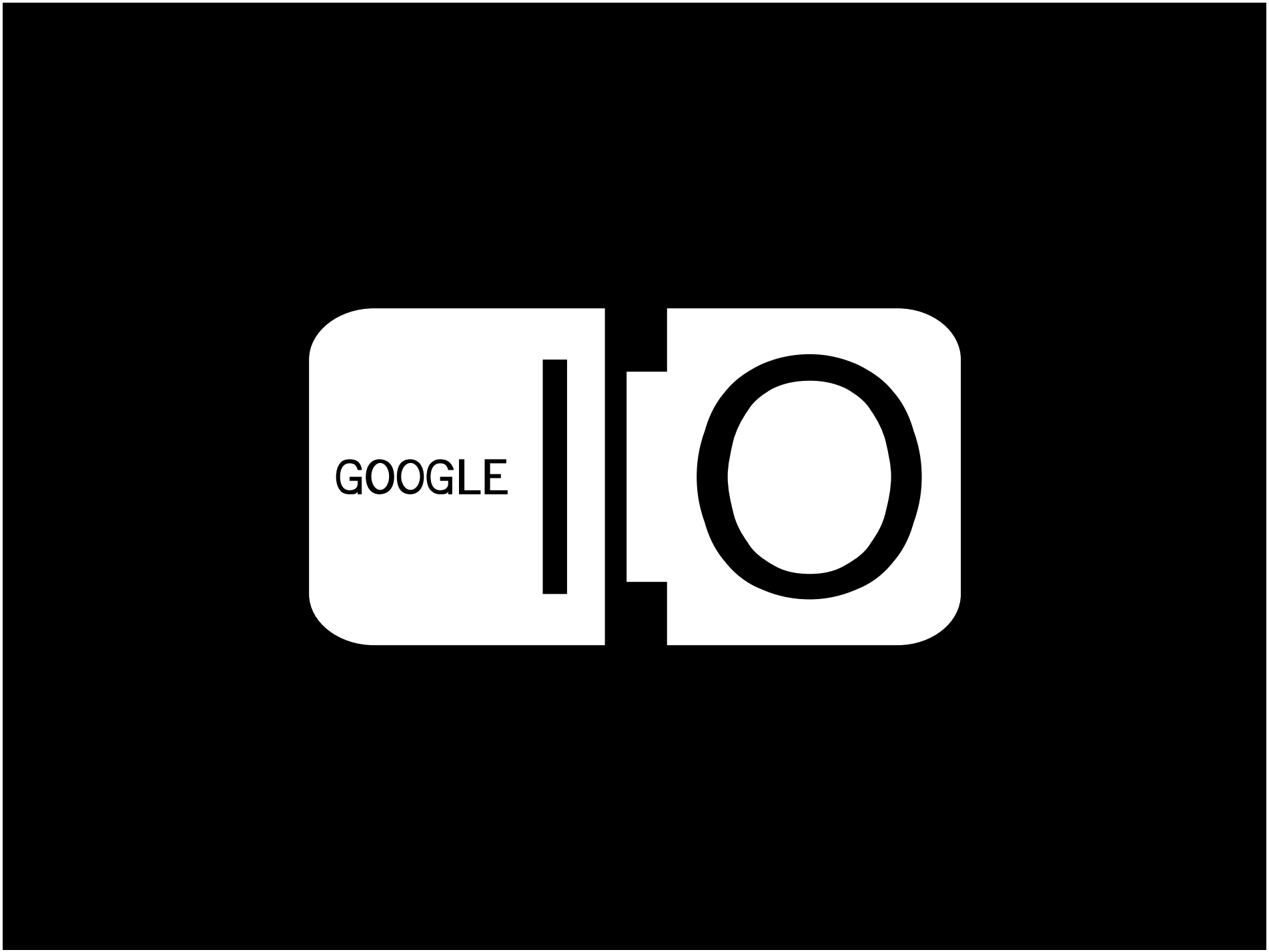


The Big Picture

- GWT.create() + <module>.gwt.xml
  - compile-time forking in permutations
  - compile-time code generation
- Each permutation is independently highly optimized
  - inlining
  - dead code elimination
- Two-phase startup
  - selects the right permutation
  - enables perfect caching
- The result: happy end users



Q & A


GOOGLE
I/O