Static resource management & optimization

Xiaoliang "David" Wei
Facebook Inc.
www.facebook.com/DavidWei
DavidWei@acm.org

Velocity China, Dec 7th, 2010, Beijing

SR management & optimization

A major role in Web Performance

- 12 out of 14 rules in “High Performance Websites”

  - Few HTTP requests
  - Use Content Distribution Network
  - …
- Half of the rules in “Even Faster Web Sites”

  - Splitting Initial Payload
  - Loading scripts without blockings
  - …

Agenda

1. The challenges & the dreams
2. Interface and architecture
3. Scalability
4. Adaptability

Challenges & Dreams

Facebook: the largest social network

500+ million heterogeneous users around the world

- 100+ languages translated by grass-root users
- 10000+ browser varieties and changing (~8 browsers >1% market shares)
- IPs from Large spectrum of latency and connectivity
- Different usage patterns (40K+ patterns of usages in major full page end point)

Facebook: the largest social network

500+ million heterogeneous users around the world

- 100+ languages translated by grass-root users
- 10000+ browser varieties and changing (~8 browsers >1% market shares)
- IPs from Large spectrum of latency and connectivity
- Different usage patterns (40K+ patterns of usages in major full page end point)

- Summary: a large scale web site with high user variety

Performance optimization: Challenge

- Example:

"Best Practices for Speeding Up You Web Site" – YSlow

Rule 1: Minimize HTTP Request

- Concatenating files: Javscript and Stylesheets packaging
- Images spriting
- ...

Performance optimization: Challenge

- Day 1: Some smart engineers start a project!

‹Print CSS tag for feature A›

‹Print CSS tag for feature B›

‹Print CSS tag for feature C›

‹print HTML of feature A›

‹print HTML of feature B›

‹print HTML of feature C›

...

“Let’s write a new page with features A, B and C!”

Performance optimization: Challenge

- Day 2: One smart engineer read YUI blog and says...

‹Print CSS tag for feature A›

‹Print CSS tag for feature B›

‹Print CSS tag for feature C›

‹print HTML of feature A›

‹print HTML of feature B›

‹print HTML of feature C›

...

"A & B & C are always used; let's package them together!"

Performance optimization: Challenge

- Day 2: Awesome!

<Print CSS tag for feature A&B&C in a single package>

<print HTML of feature A>

<print HTML of feature B>

<print HTML of feature C>

...

Performance optimization: Challenge

- Day 3: feature C evolves…

‹Print CSS tag for feature A&B&C in a single package›

‹print HTML of feature A›

‹print HTML of feature B›

If (users_signup_for_C()) { ‹print HTML of feature C›}

…

Performance optimization: Challenge

- Day 3: feature C evolves…

‹Print CSS tag for feature A&B&C›

A&B are always used, while C is not. ..

‹print HTML of feature A›

‹print HTML of feature B›

If (users_signup_for_C()) { ‹print HTML of feature C›}

…

Performance optimization: Challenge

- Day 4: feature C is deprecated

<Print CSS tag for feature A&B&C>

<print HTML of feature A>

<print HTML of feature B>

// no one uses C { <print HTML of feature C>}

...

Performance optimization: Challenge

- Day 4: we start to send unused bits

‹Print CSS tag for feature A&B&C›

It is hard to remember we should remove C here.

‹print HTML of feature A›

‹print HTML of feature B›

// no one uses C { ‹print HTML of feature C›}

...

Performance optimization: Challenge

- One month later…

‹Print CSS tag for feature A&B&C&D&E&F&G…›

if (F is used) ‹print HTML of feature F›

‹print HTML of feature G›

if (F is not used) { ‹print HTML of feature E›}

…

Thousands of dead CSS bytes in the package.

The dreams

Fully customized user experience:

- Most suitable delivery mechanisms for each browser capability
- Best packaging / sprite strategy for a user's connectivity
- Accurate predictive loading based on usage patterns

Allow product engineers to move fast: 3D

- Simple **D**evelopment process
- Quick **D**eployment
- Easy **D**ebugging

The dreams

Fully customized user experience:

- Most suitable delivery mechanisms for each browser capability
- Best packaging / sprite strategy for a user's connectivity
- Accurate predictive loading based on usage patterns

Frontend Infrastructure: Static Resource Management System

Allow product engineers to move fast: 3D

- Simple **D**evelopment process
- Quick **D**eployment
- Easy **D**ebugging

Architecture

Interfaces

- Back to Day 1:

<Print CSS tag for feature A>

<Print CSS tag for feature B>

<Print CSS tag for feature C>

<print HTML of feature A>

<print HTML of feature B>

<print HTML of feature C>

Interfaces

<Print CSS tag for feature A>

<Print CSS tag for feature B>

<Print CSS tag for feature C>

<print HTML of feature A>

<print HTML of feature B>

<print HTML of feature C>

- Back to Day 1:

require_static(A_css); <render HTML of feature A>

require_static(B_css); <render HTML of feature B>

require_static(C_css);<render HTML of feature C>

render_page($htmls); // deliver all CSS and render HTMLs

Interfaces

<Print CSS tag for feature A>

<Print CSS tag for feature B>

<Print CSS tag for feature C>

<print HTML of feature A>

<print HTML of feature B>

<print HTML of feature C>

- Back to Day 1:

require_static(A_css); <render HTML of feature A>

require_static(B_css); <render HTML of feature B>

require_static(C_css);<render HTML of feature C>

Separate *Declaration* from actual *Delivery*

render_page($htmls); // deliver all CSS and render HTMLs

Interfaces

- Back to Day 1:

require_static(A_css); ‹render HTML of feature A›

require_static(B_css); ‹render HTML of feature B›

require_static(C_css);‹render HTML of feature C›

render_page($htmls);

Requirement Declaration lives with HTML rendering

Global Optimization on ***Delivery***

Interfaces

```
require_static(A_css); ‹render HTML of feature A›
require_static(B_css); ‹render HTML of feature B›
require_static(C_css);‹render HTML of feature C›
render_page($htmls);
```

A.css:

A component name separate from file name

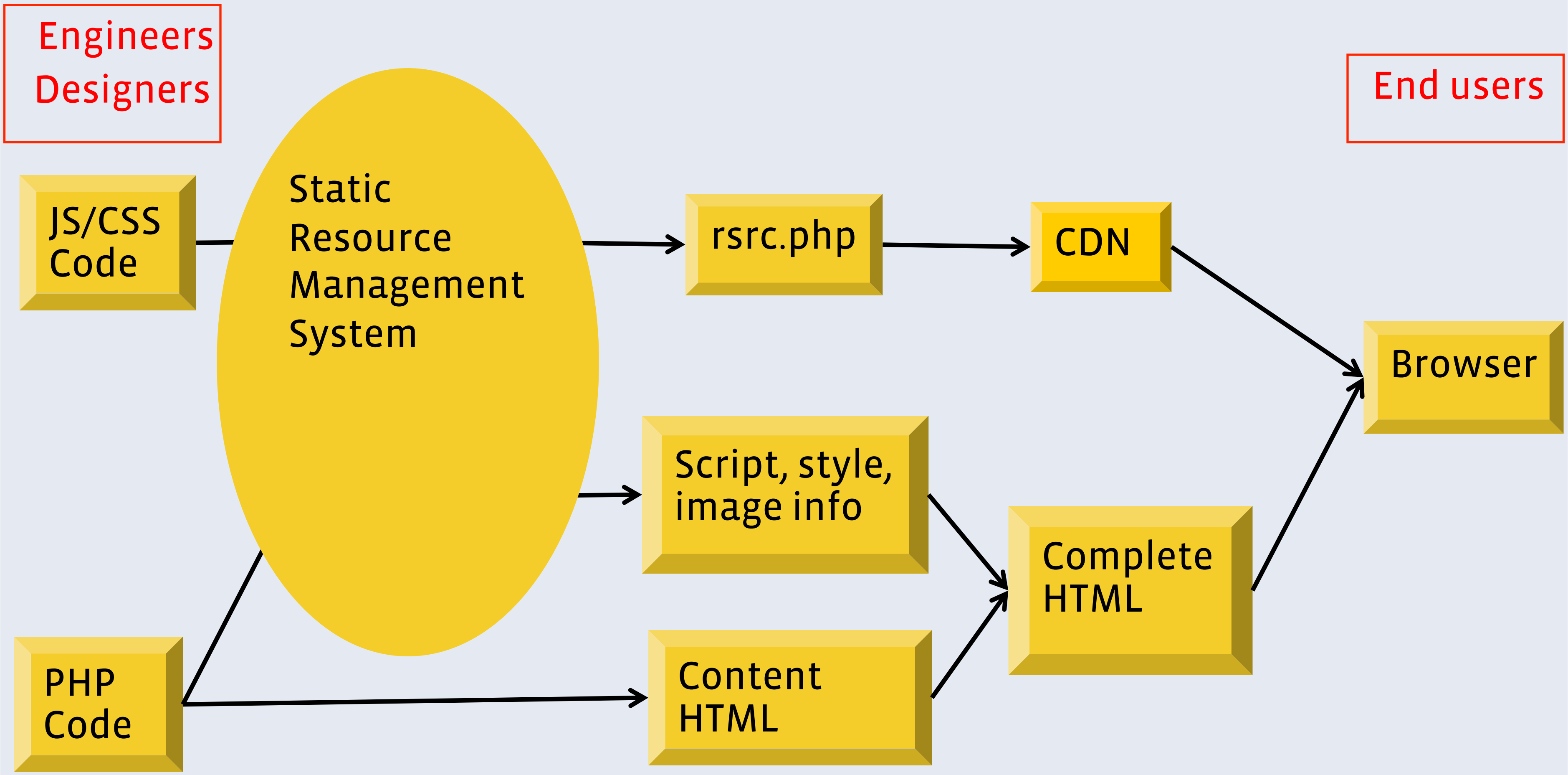
```
/**
 * @provides A_css
 * @requires Core_css
 *
 * @author dwei
 * @non-blocking
 */
```

Direct dependencies for this component

Optional delivery preference

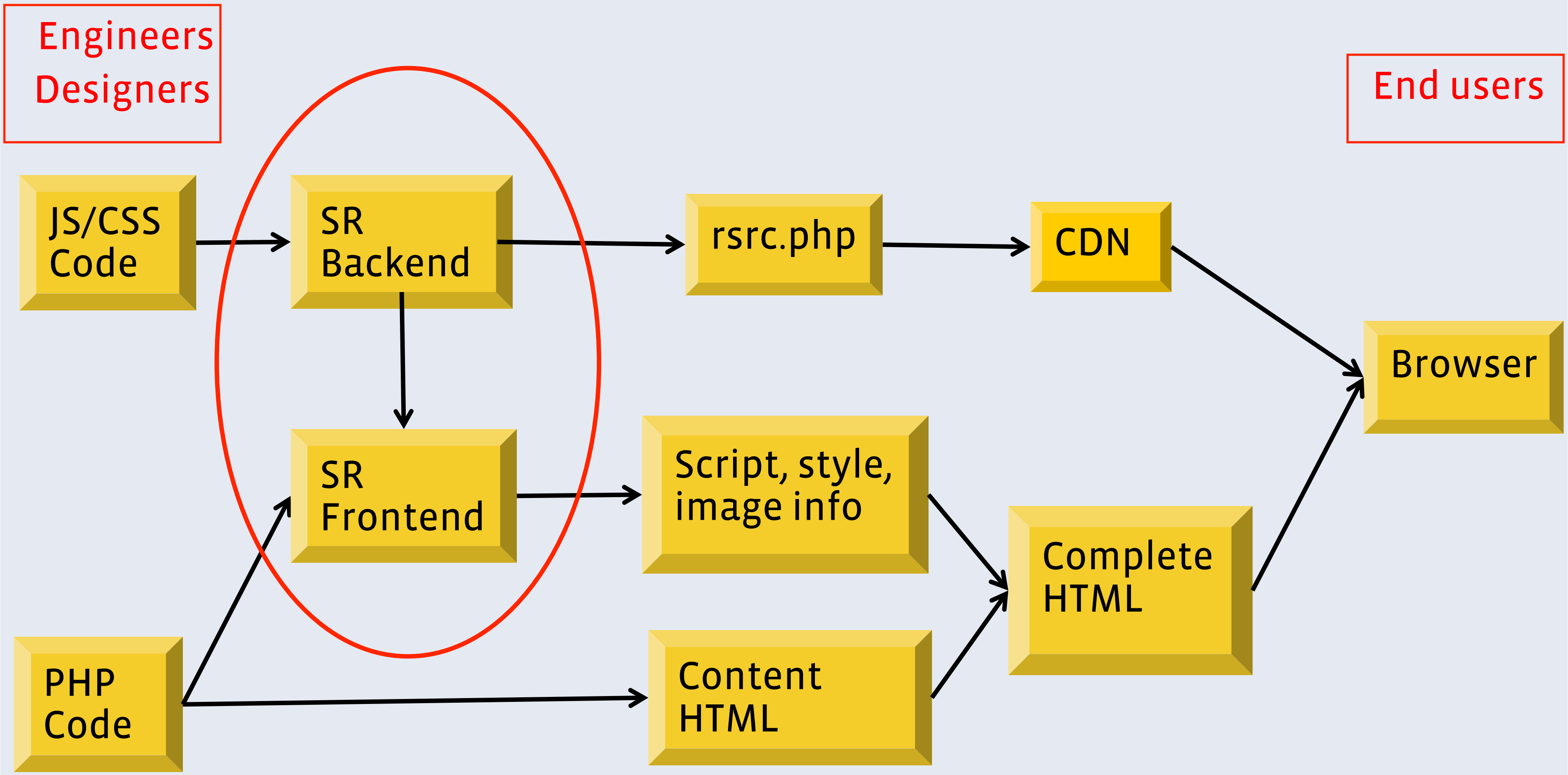
Development and deployment process

The Life of JS/CSS/Images



Development and deployment process

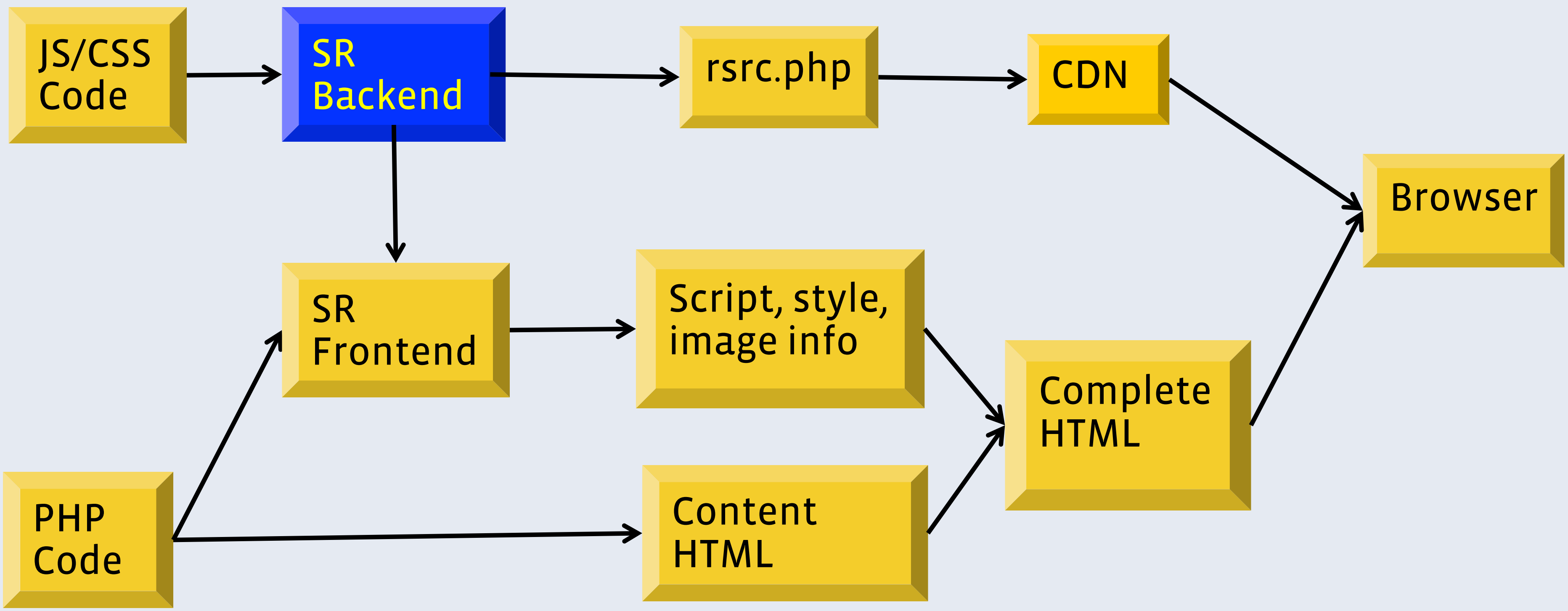
The Life of JS/CSS/Images



SR Management System

Backend:

- Dependency analysis
- Transforms: localization / minification
- Combinations: packaging / sprite



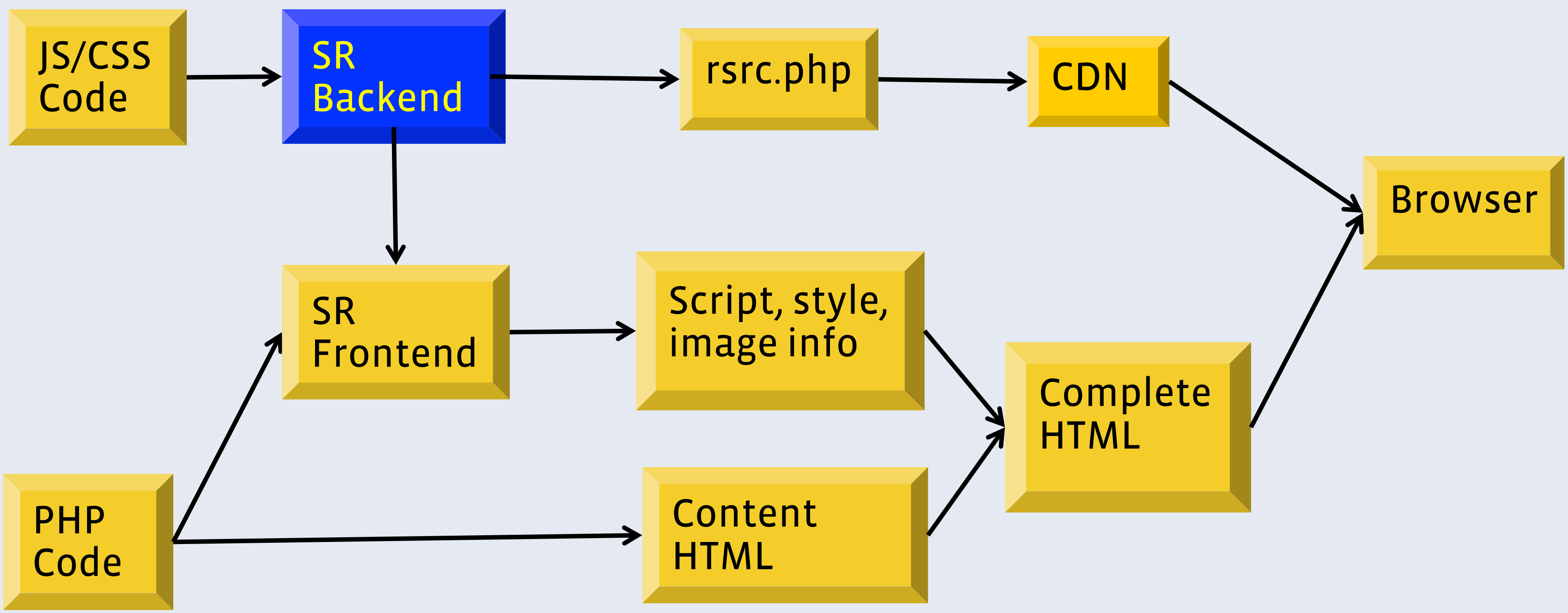
SR Management System

Backend:

- Dependency analysis
- Transforms: localization / minification
- Combinations: packaging / sprite

Component => URIs

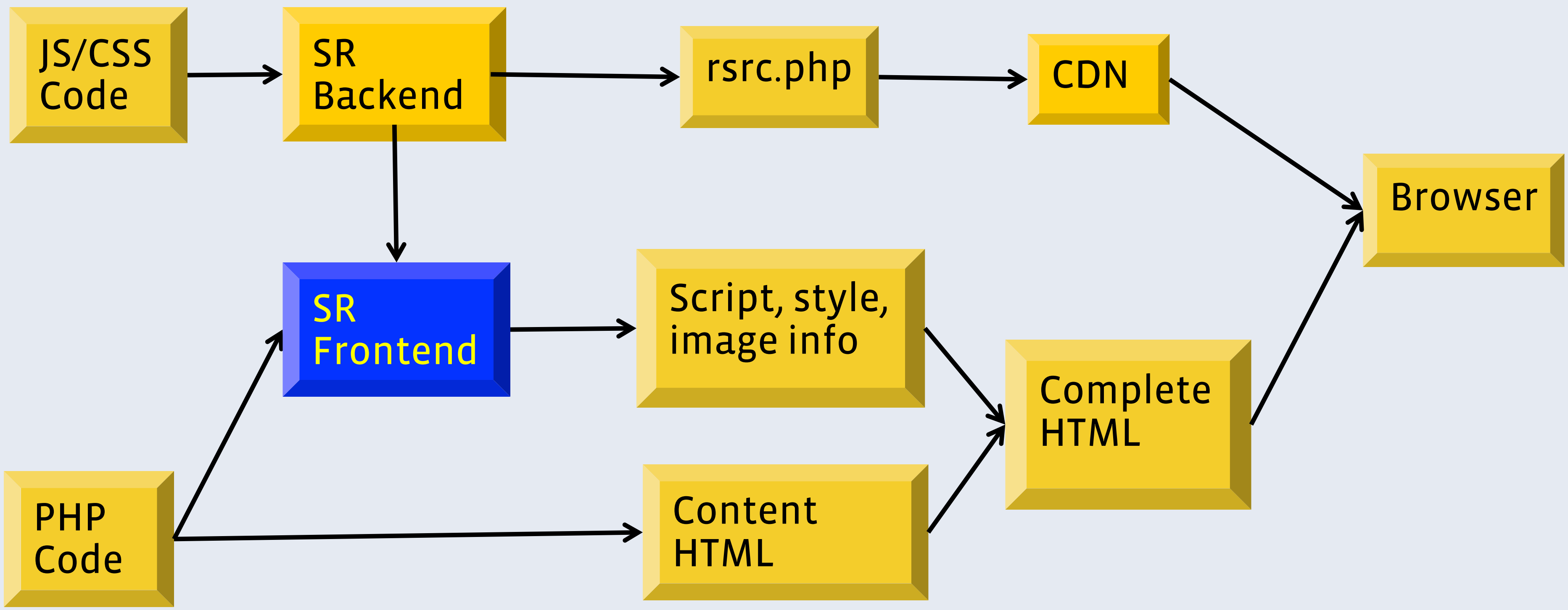
URI => Data



SR Management System

Frontend:

- Component => URIs
- URI + user profile => delivery mechanisms
  - static tags, async loading, packaging or not, ...



Scalability

A website with 500M active users

Dimensions of parameters

- 10000+ static resources
- x 100+ language translations
- x 3 browser setups
- x 5 packaging strategies
- x 3 user AB testing groups
- x 2 delivery strategies (iframe / raw)
- x 2 minification strategies

- = 3,000,000 + different static resources

A website with 500M active users

Dimensions of parameters

- 10000+ static resources
- x 100+ language translations
- x 3 browser setups
- x 5 packaging strategies
- x 3 user AB testing groups
- x 2 delivery strategies (iframe / raw)
- x 2 minification strategies

- = 3,000,000 + different static resources

To support

- Multiple revisions
- Released in 10 minutes
- Even more dimensions in the future

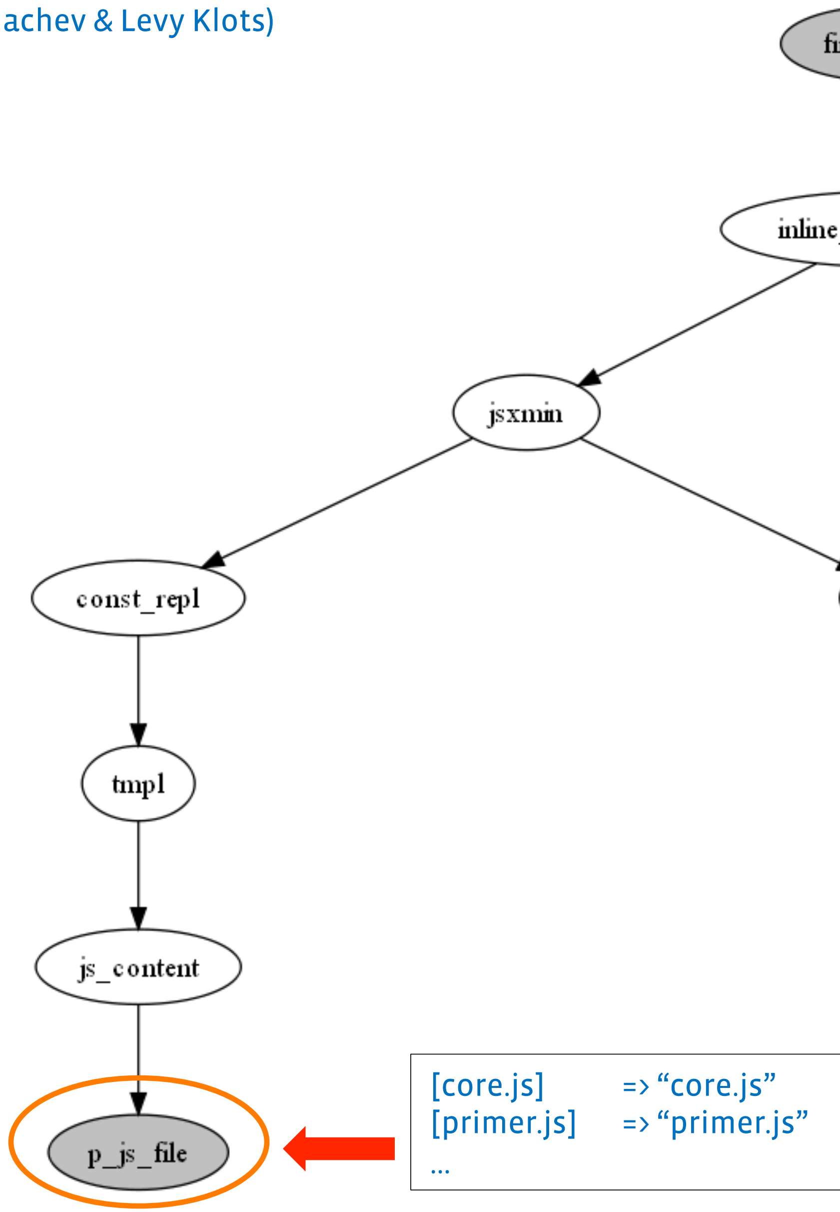
Scaling the static resource system

A “make” approach

- Describe a build process as a graph
- Node in the graph: data
  - Initial parameters / data (stubs)
  - Results of a processing step (transits)
- Edge in the graph: dependencies between the processing steps and the data

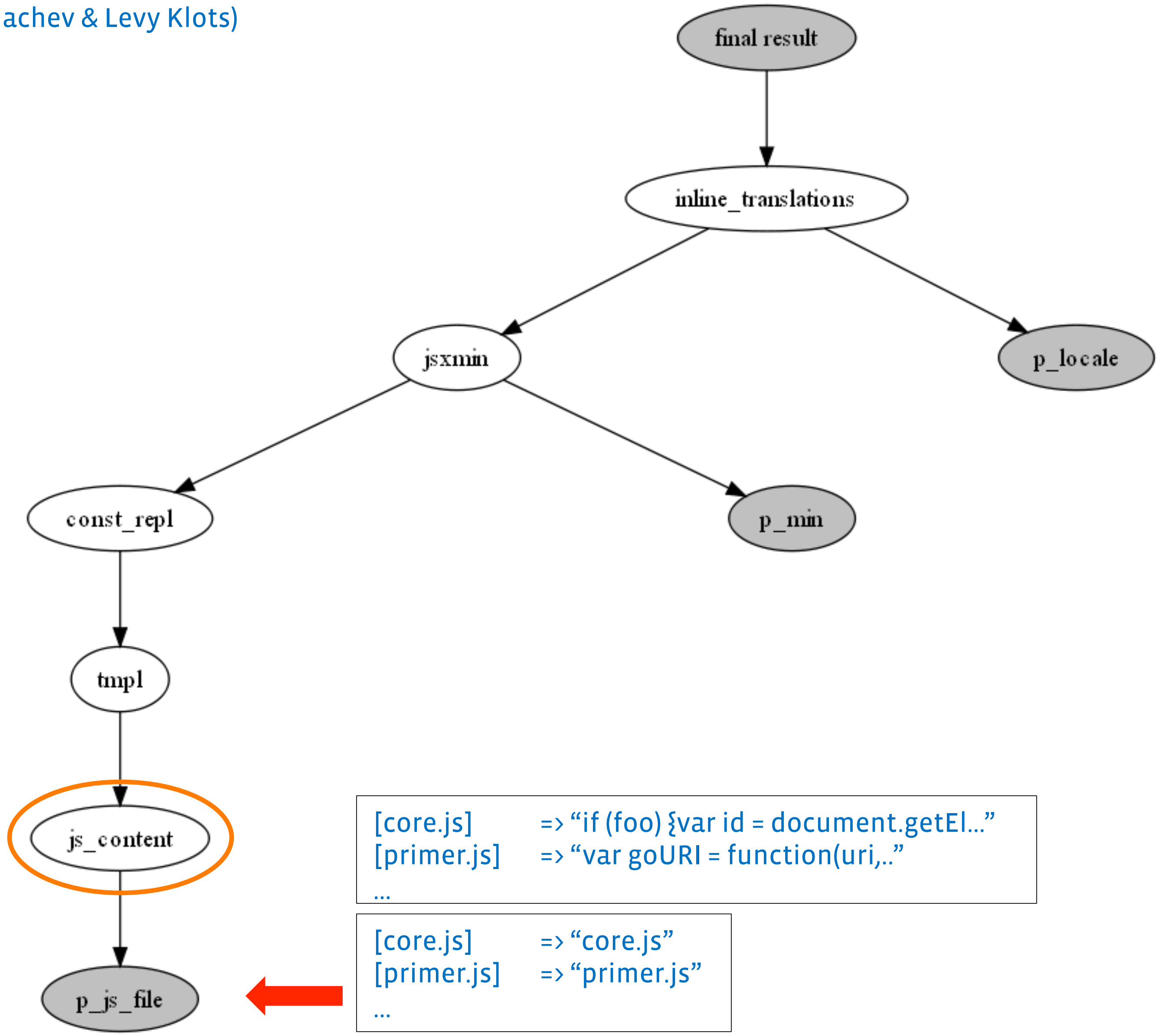
A “make” system for static resource: Example

(by Andrey Sukhachev & Levy Klots)



A “make” system for static resource: Example

(by Andrey Sukhachev & Levy Klots)

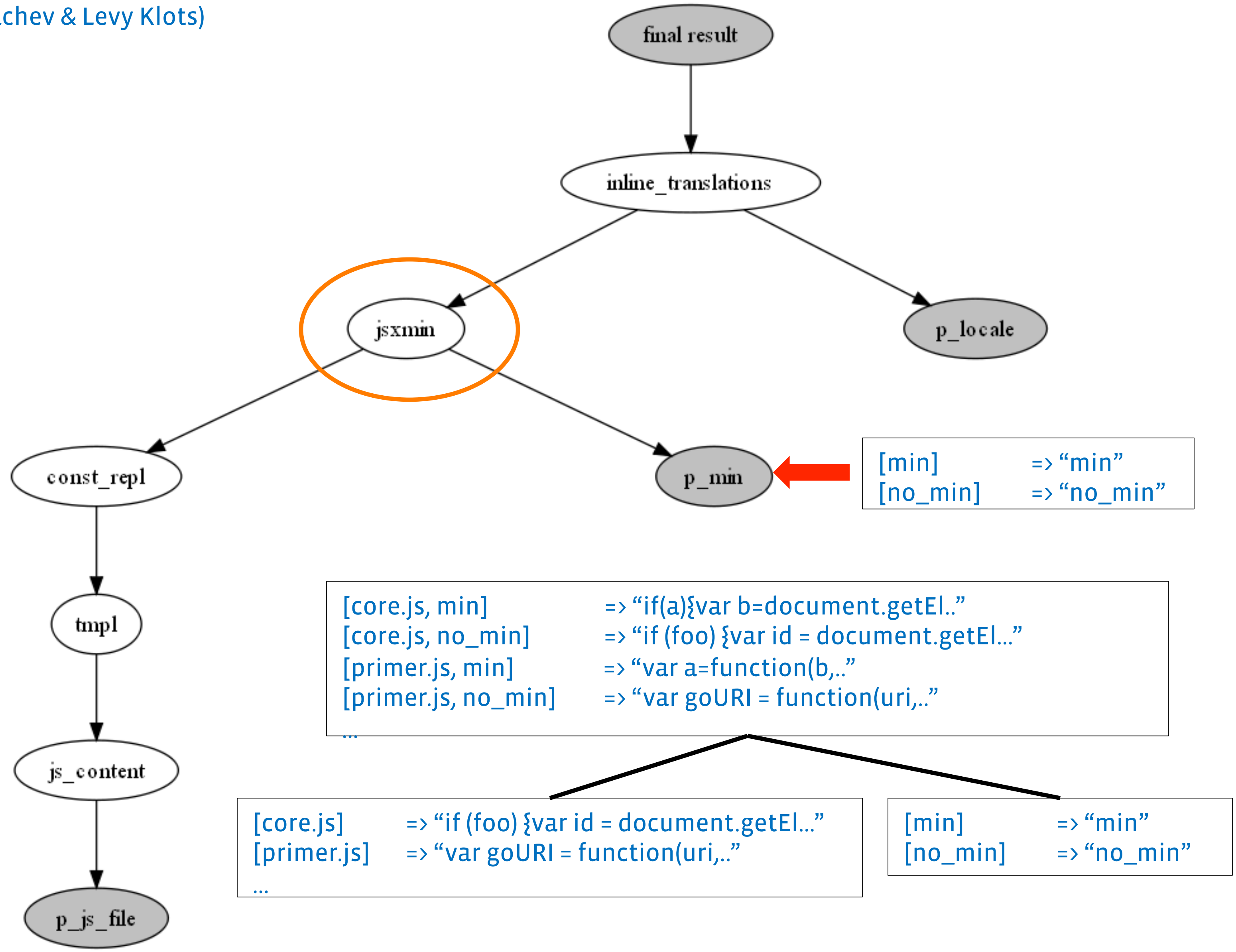


[core.js] => “if (foo) {var id = document.getEl…”
[primer.js] => “var goURI = function(uri,..”
…

[core.js] => “core.js”
[primer.js] => “primer.js”
…

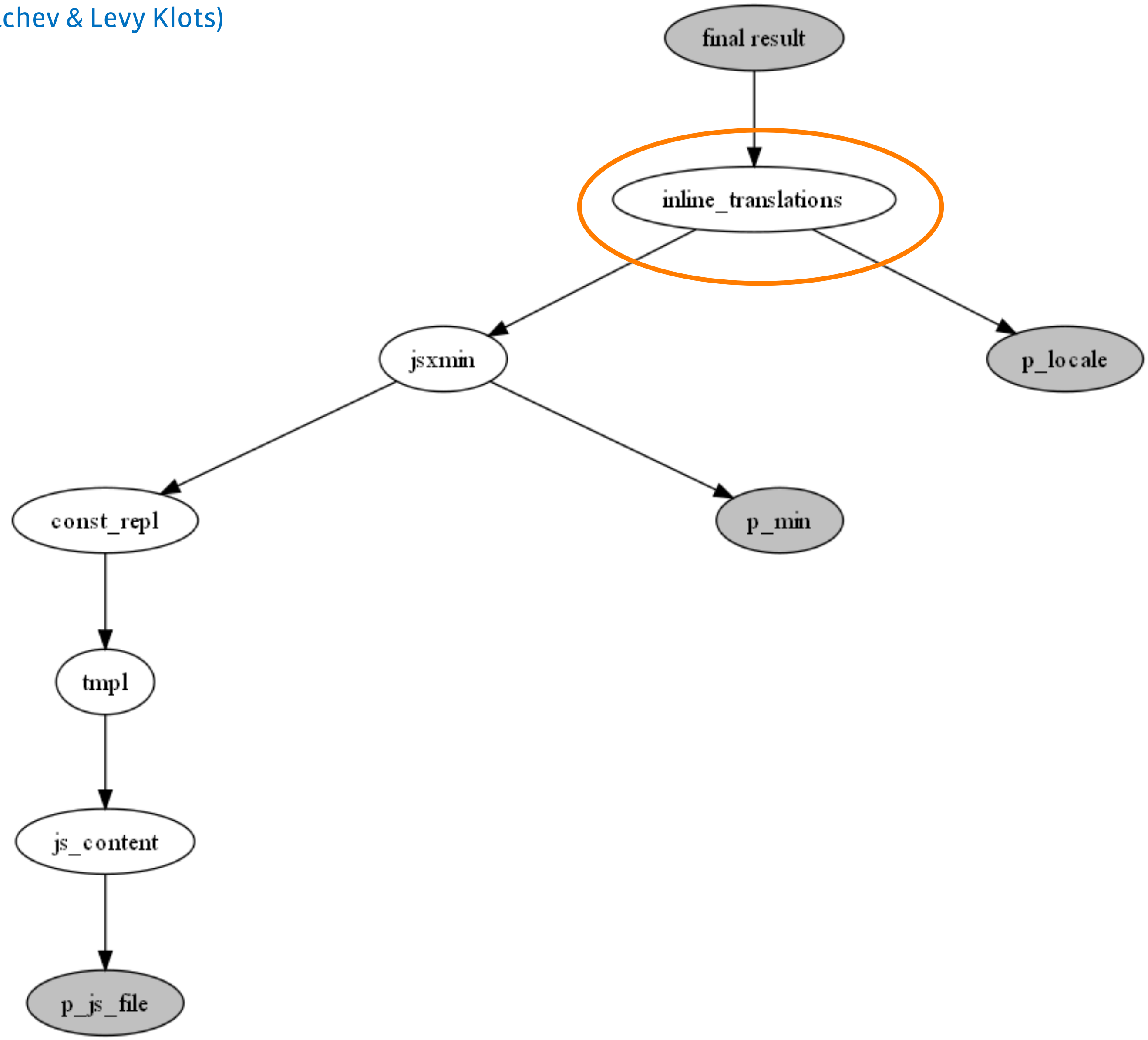
A “make” system for static resource: Example

(by Andrey Sukhachev & Levy Klots)



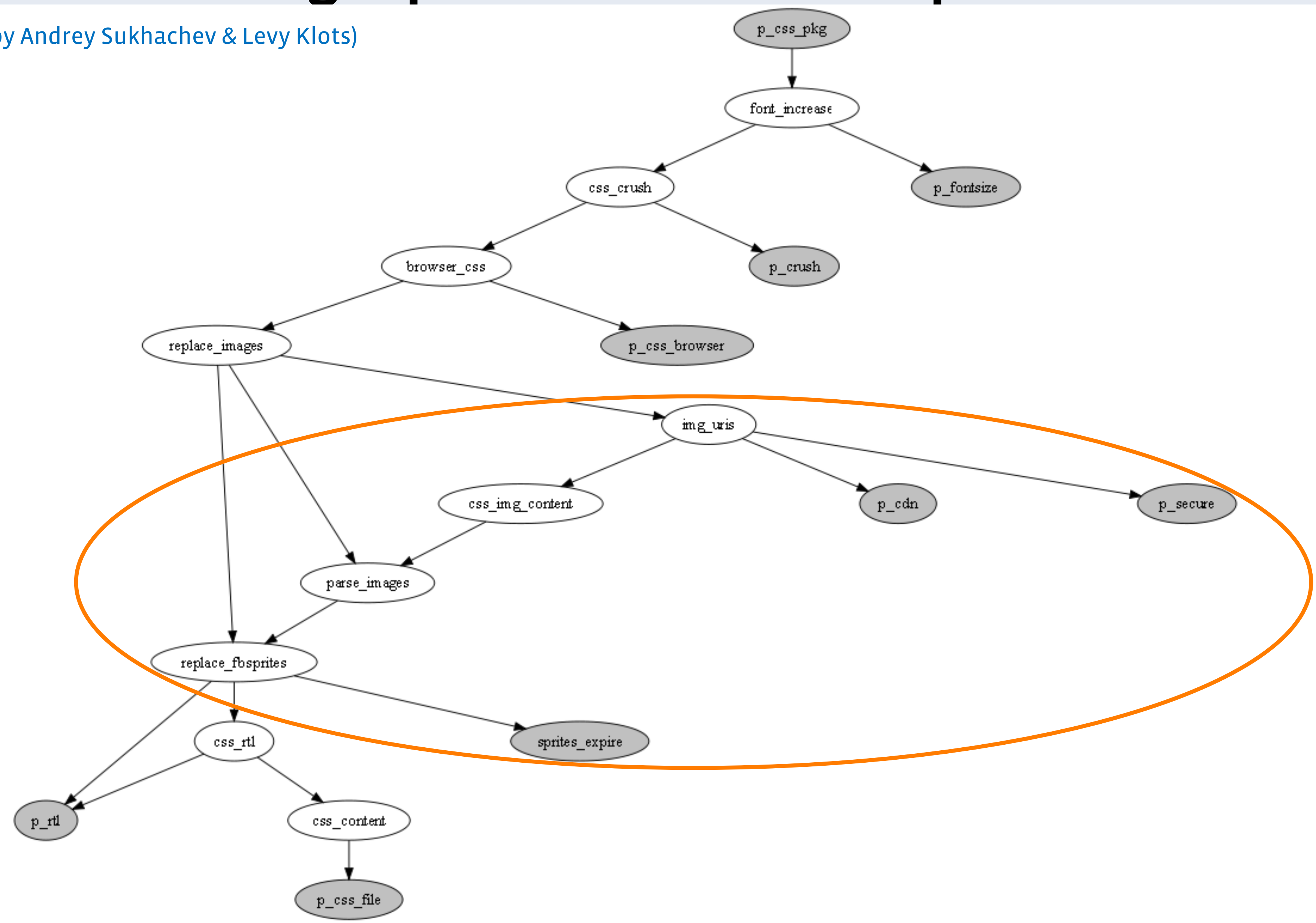
A “make” system for static resource: Example

(by Andrey Sukhachev & Levy Klots)



CSS build graph: the most complex build

(by Andrey Sukhachev & Levy Klots)



Static resource backend

Allow product engineers to move fast

- Simple development:
    - Engineers develop one version of static resource;
    - The build system builds 100+ versions of it for different users;
- Quick deployment:
    - Real time for sandboxes
    - Production deploy within 10 minutes
- Easy to add new dimensions

Adaptability

Case Study – Revisited

- One months later…

‹Print CSS tag for feature A&B&C&D&E&F&G…›

Thousands of dead CSS bytes in the package.

if (F is used) ‹print HTML of feature F›

‹print HTML of feature G›

if (F is not used) { ‹print HTML of feature E›}

…

Good Ideas, manual optimization not adaptable

- Packaging most often used files;
- Send these the references of these files ASAP

Case Study – Revisited

Optimization needs adaptability

- SR management system tracks change of usage patterns
- SR adapts its optimization strategies adaptively

Good Ideas, manual implementation not adaptable

- Packaging most often used files;
- Send these the references of these files ASAP

Packager: Global JS/CSS Optimization

Online API

require_static(A_css); ‹render HTML of A›

require_static(B_css); ‹render HTML of B›

require_static(C_css); ‹render HTML of C›

render_page($htmls);

Packager: Global JS/CSS Optimization

Online API

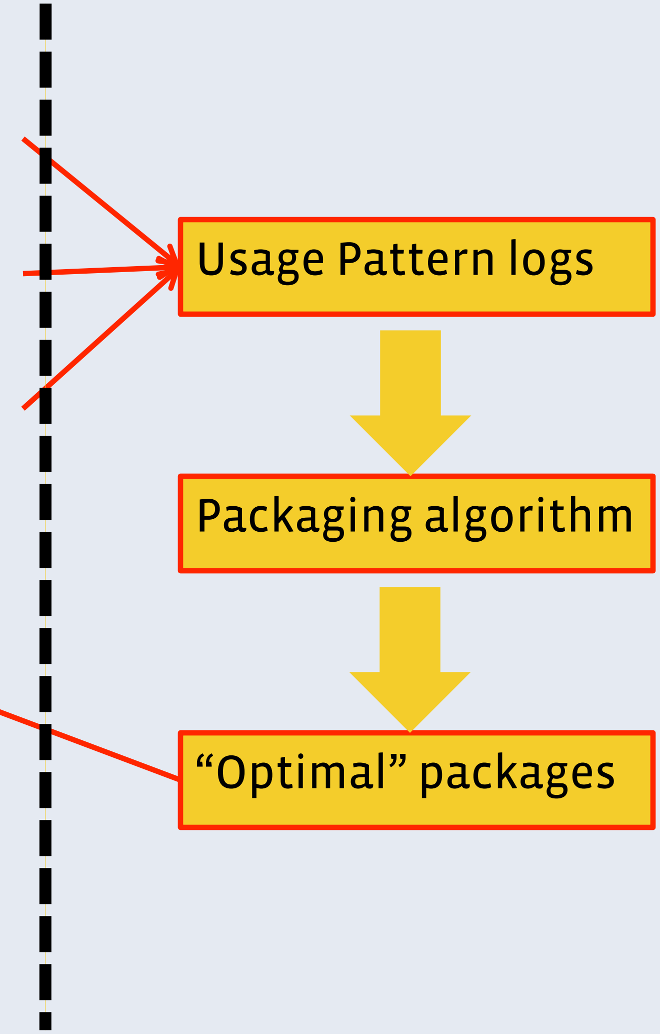
require_static(A_css); <render HTML of A>

require_static(B_css); <render HTML of B>

require_static(C_css); <render HTML of C>

render_page($htmls);

Offline analysis



Packager: Usage Pattern Logs

Page Load	1	2	3	4	5	...	100000
Page Count	10 M	1 M	100 K	20 K	10 K		1K
A.css (1 KB)	1	1	1	1	1		1
B.css (1 KB)	1	1	1	1	1		1
C.css (300B)	1	1	1				
D.css (2 KB)	1						1
E.css (700B)				1	1		1
F.css (400B)				1			1
G.css (600B)	1		1	1	1		1

Usage Pattern logs

Packager: Cost/Benefit Model

Page Load	1	2	3	4	5	…	100000
Page Count	10 M	1 M	100 K	20 K	10 K		1K
A.css (1 KB)	1	1	1	1	1		1
B.css (1 KB)	1	1	1	1	1		1
C.css (300B)	1	1	1				
D.css (2 KB)	1						1
E.css (700B)				1	1		1
F.css (400B)				1			1
G.css (600B)	1		1	1	1		1

- To package two files A & B:
  - "Cost": for page requests that only uses A, we waste the bytes of B, vice versa
  - "Benefit": for page requests that uses both A and B: we save one round trip
  - Bytes / Bandwidth ~ Latency
  - "Profit" to be maximized:
    Benefit – Cost

Packager: Cost/Benefit Model

Page Load	1	2	3	4	5	…	100000
Page Count	10 M	1 M	100 K	20 K	10 K		1K
A.css (1 KB)	1	1	1	1	1		1
B.css (1 KB)	1	1	1	1	1		1
C.css (300B)	1	1	1				
D.css (2 KB)	1						1
E.css (700B)				1	1		1
F.css (400B)				1			1
G.css (600B)	1		1	1	1		1

- Assume: latency = 40ms, and bandwidth = 1 Mbps
  - A+B: 40ms * 11.131M

No cost, pure gain.

Definitely package

Packager: Cost/Benefit Model

Page Load	1	2	3	4	5	…	100000
Page Count	10 M	1 M	100 K	20 K	10 K		1K
A.css (1 KB)	1	1	1	1	1		1
B.css (1 KB)	1	1	1	1	1		1
C.css (300B)	1	1	1				
D.css (2 KB)							1
E.css (700B)				1	1		1
F.css (400B)				1			1
G.css (600B)	1		1	1	1		1

- Assume: latency = 40ms, and bandwidth = 1 Mbps
  - B+C: 40ms * 11.1M
    - 300B / 1Mbps * 0.031M

Benefit larger than cost

OK to package

Packager: Cost/Benefit Model

Page Load	1	2	3	4	5	…	100000
Page Count	10 M	1 M	100 K	20 K	10 K		1K
A.css (1 KB)	1	1	1	1	1		1
B.css (1 KB)	1	1	1	1	1		1
C.css (300B)	1	1	1				
D.css (2 KB)							1
E.css (700B)				1	1		1
F.css (400B)				1			1
G.css (600B)	1		1	1	1		1

- Assume: latency = 40ms, and bandwidth = 1 Mbps
  - B+D: 40ms * 1K
    - – 2K / 1Mbps * 11.13M

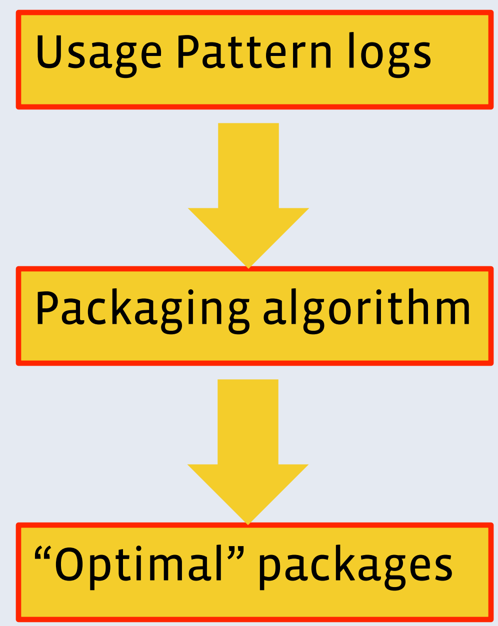
Cost larger than benefit

Don't package

Packager: Optimal packages

Page Load	1	2	3	4	5	…	100000
Page Count	10 M	1 M	100 K	20 K	10 K		1K
A.css (1 KB)	1	1	1	1	1		1
B.css (1 KB)	1	1	1	1	1		1
C.css (300B)	1	1	1				
D.css (2 KB)							1
E.css (700B)				1	1		1
F.css (400B)				1			1
G.css (50B)	1		1	1	1		1

- Pkg 1: A, B, C
- Pkg 2: E, F, G



Adaptive Static Resource Optimization

Adaptive Packaging / Spriting

- Cross-feature optimizations (e.g. search + ads)
- Adaptive to change of user behaviors and code developments
- Similar technology works for image spriting (different cost function for the extra sprite CSS)
- Models can be improved for different TTI goals

Experiment: Adaptive Image Spriting

The puzzle of image spriting:

- Thousands of virtual gifts with static images, which to sprite?

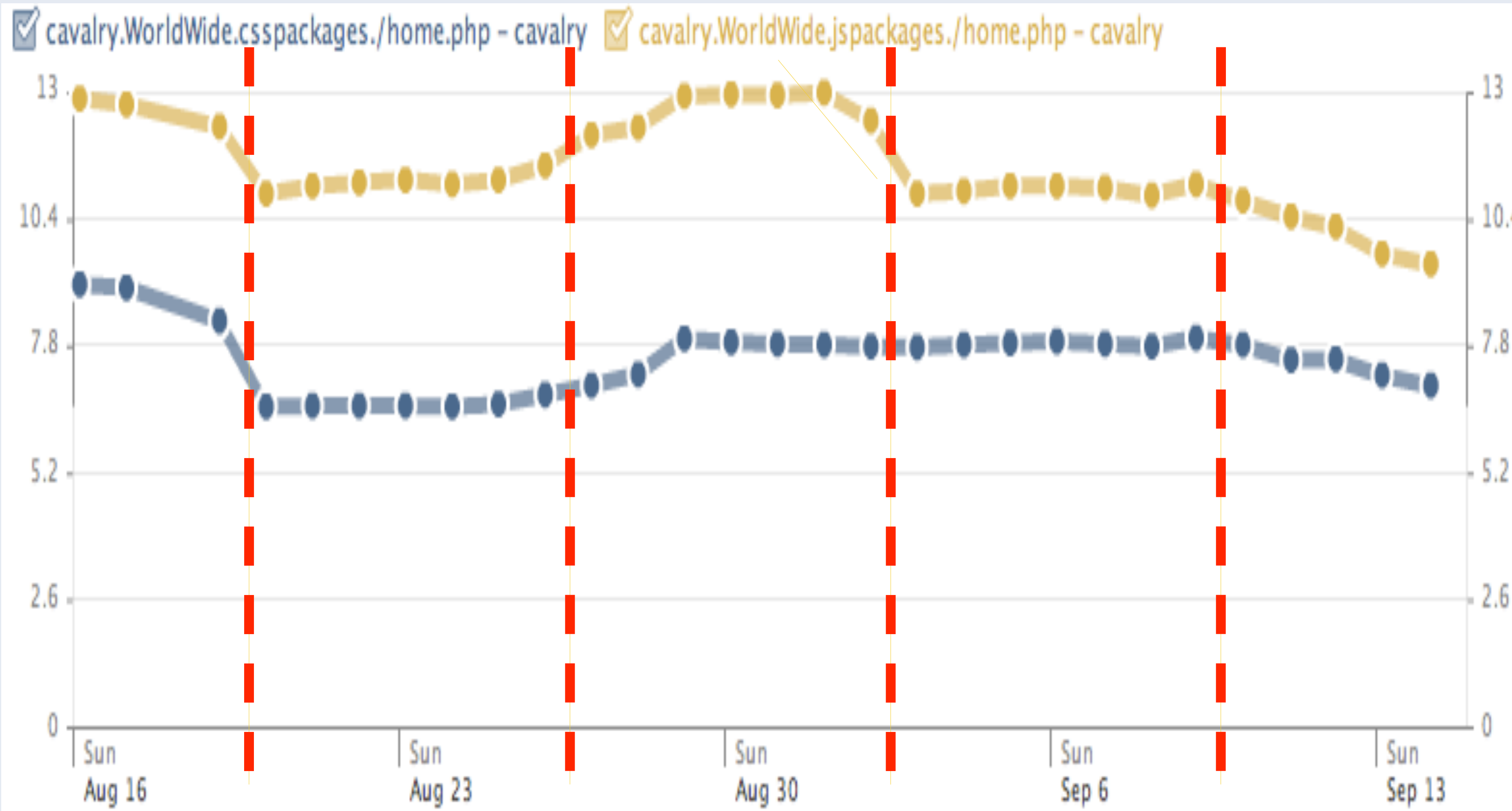
Experiment: Adaptive Image Spriting

The puzzle of image spriting:

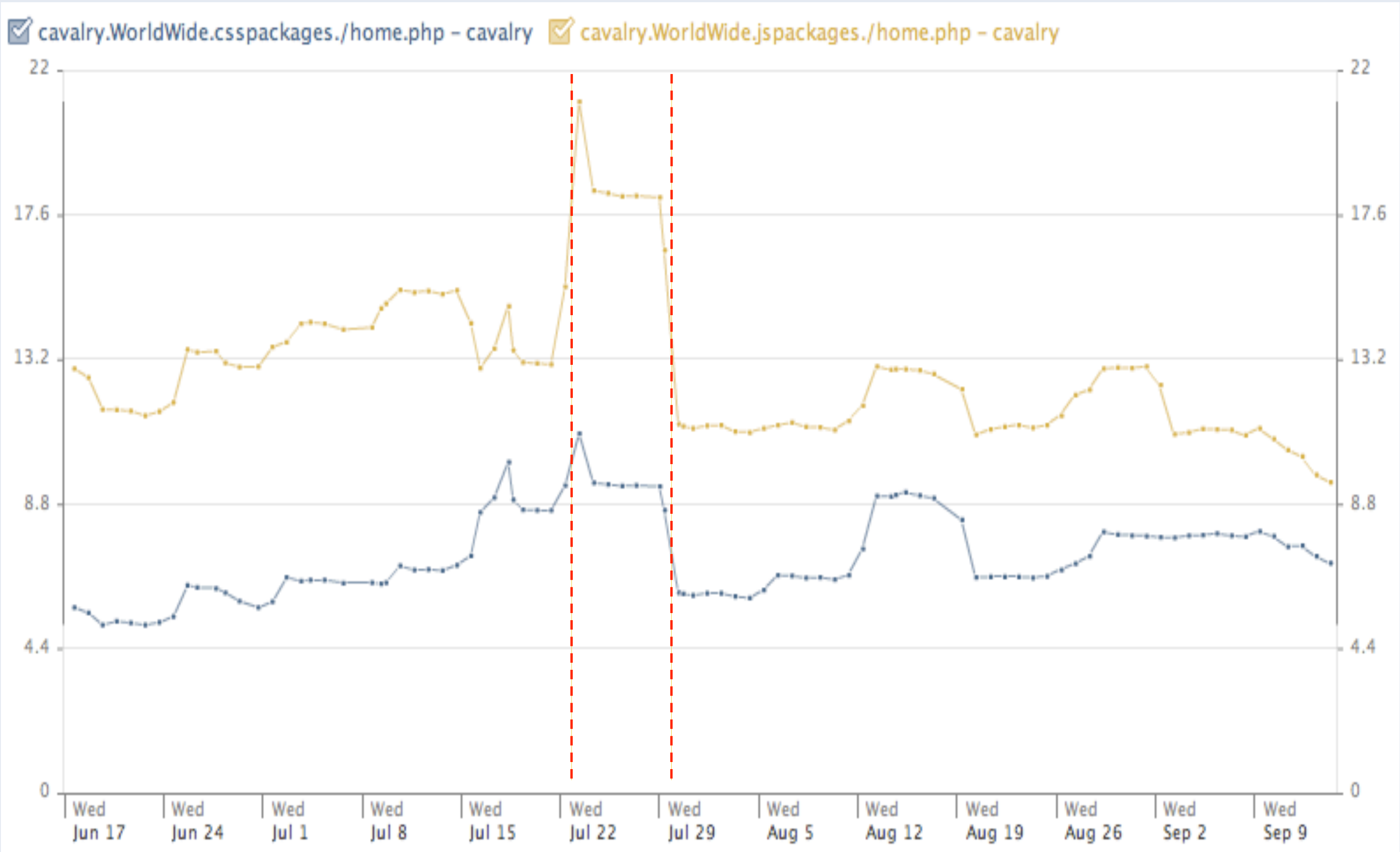
- The answer is…

Adaptive to new usages

of JS/CSS Packages served in one month of 2009



When data go wrong



Conclusions

Static Resource Management

A major component in Web performance

- **Challenges:**
  - Easy to start, hard to make right;
  - Particularly challenging for large scale web sites with heterogeneous users.
- **Experience:**
  - Focus on interface: good interface frees the engineers and provides high leverage opportunities for global optimizations;
  - Adaptability is important to ensure the web site is fast by default;
  - Scalability is a must for large sites.

Thank you!

Xiaoliang "David" Wei
Facebook Inc.
www.facebook.com/DavidWei
DavidWei@acm.org