low hanging fruit   vs.   micro-optimization

Creative Techniques for Loading Web Pages Faster

Trevor Parscal - @trevorparscal
Roan Kattouw - @catrope

OSCON 2011

WIKIMEDIA

– We work for Wikimedia Foundation
– We created a JS/CSS loading system for Wikipedia
– We want to share what we did, and what we learned
– Questions throughout or at the end are OK
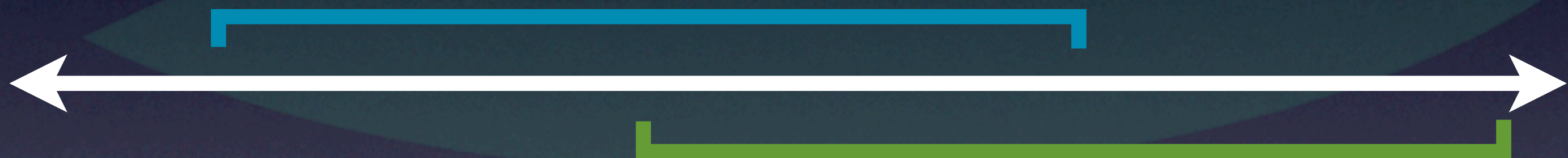
People

Roan

robot

Trevor

human

WIKIMEDIA

– Roan: Robot
– Trevor: Human

People

Back-end   Front-end

WIKIMEDIA

– Lots of crossover
– Understand the problem of client–rich development
– Know how to make things scale well

Software

MediaWiki

platform

Wikipedia

project

OSCON 2011

WIKIMEDIA

- MediaWiki: wiki application that powers Wikimedia's sites
- Open source!
- Linux, Apache, MySQL, PHP, Memcached, Squid, Varnish
- Wikimedia sites are the 5th most visited sites in the world
- The largest Wikimedia site is Wikipedia

– This is a Wikipedia page
– It's got lots of resources that make it load slowly

Resources
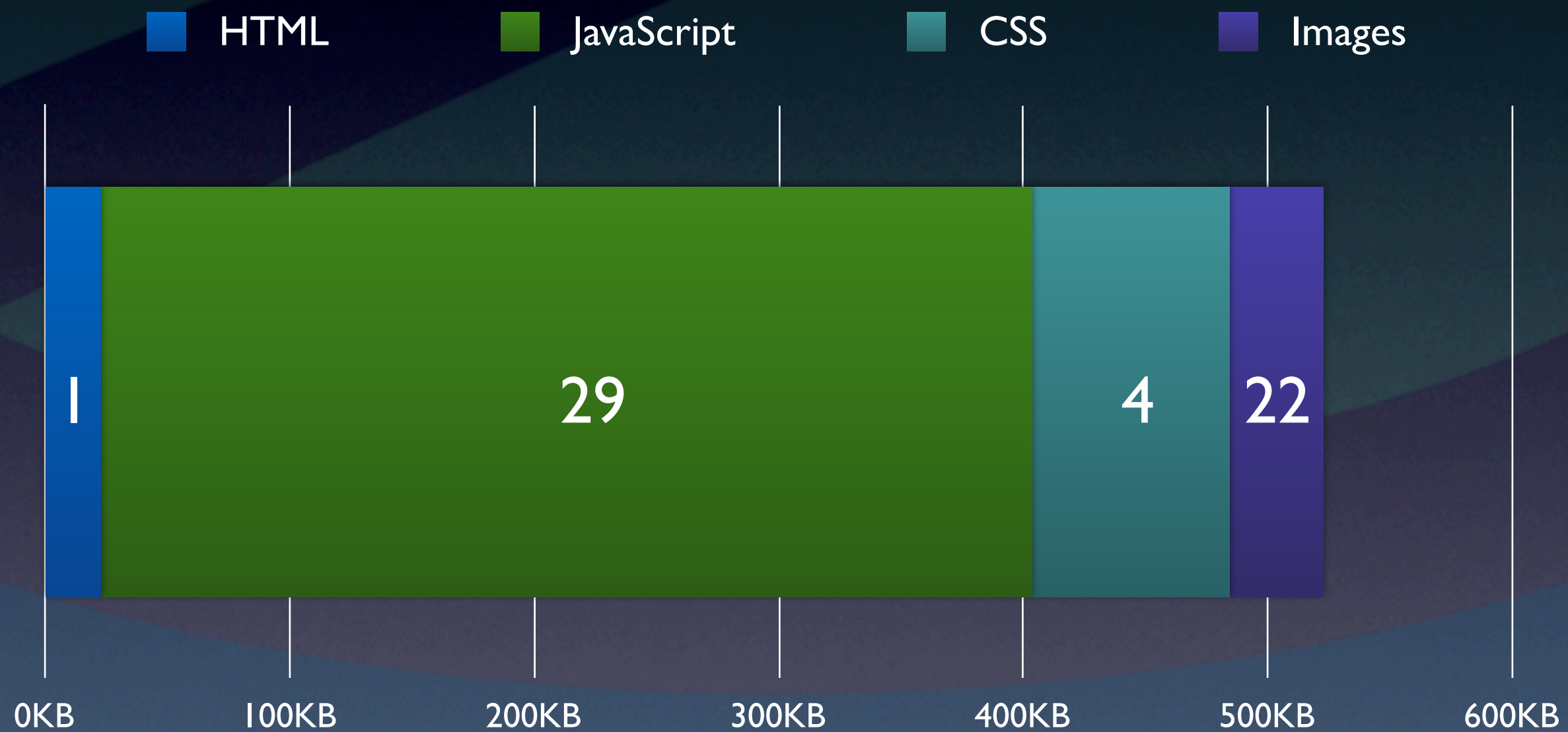
- HTML is a tiny part of the problem
- JS is where most of the gains are
- CSS is large-ish, but at least only a few requests
- Images are small, but lots of requests are costly

Resources

Scripts
javascript

Styles
css

Translations
json

– What are all these things doing?
–– Scripts power rich user interfaces – we are doing more of this now
–– Styles make things look right – this is getting complex too
– We also need translated messages on the client!

Origins

Core
skins & common stuff

Extensions
most everything else

Users
Math.random();

WIKIMEDIA

– Where do these resources come from?
– Core software (filesystem)
– Extension software (filesystem)
– User scripts and styles (database)

Optimizations

Minification
whitespace removal

Combination
file concatenation
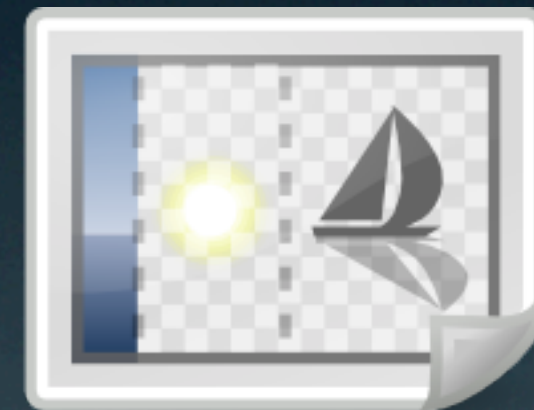
Sprites
multi-graphic images

WIKIMEDIA

– Minification wasn't used commonly
– Nothing was combined – no way to know what to combine
– Sprites were not used much, limited functionality & laziness
– We were using gzip at least

Caching

Versioning
one version for everything

Query String
versions in the html

Purging
wait for cache expiry

WIKIMEDIA

- Global style version increment on update -- global, so all or nothing
- Must be global because resources are not always known in advance
- Embedded in HTML
- Sucks for purging, version number stuck in Squid-cached HTML

Being a developer sucked

Creating
good luck!

Maintaining
messy and boring

Using
clunky and slow

WIKIMEDIA

– Just getting scripts style and messages to the client was a chore
– RTL had to be done manually
– Localization was awkward and clunky
– Optimization was a pain to maintain
– All of this distracted us from making software better
– User experience was bad, things were slow

Traffic
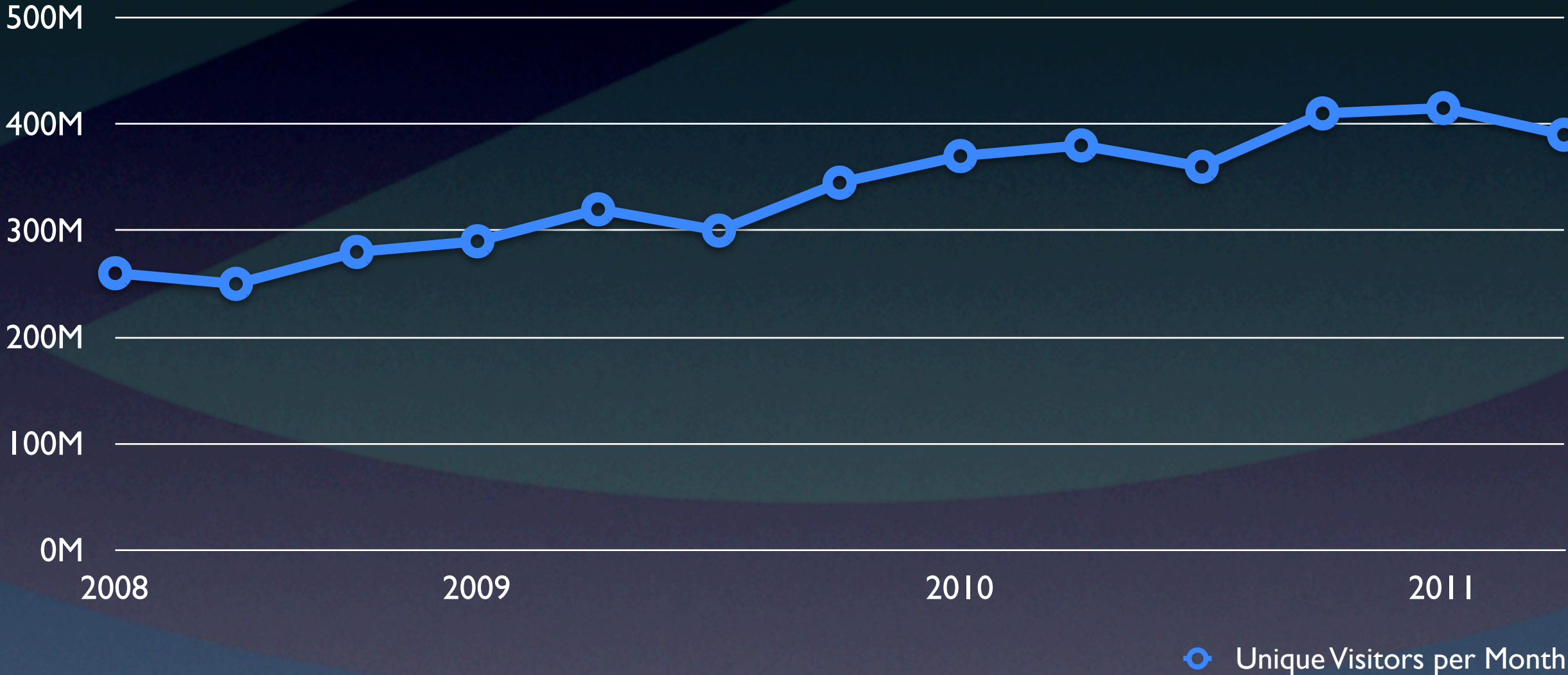
– These are in MILLIONS
– Per month

Cache Invalidation

Users
user-specific resources

Admins
site-wide resources

Translators
site-wide messages

WIKIMEDIA

- User scripts and styles can be changed at any time
-- They only affect that user
- Site scripts and styles can be changed at any time
-- They affect everyone
- Localizations are updated every day

Approach

Easy Gains
no micro-optimization

Client Focus
it's where the magic is

Ease of Use
developers use this stuff

WIKIMEDIA

- Easy 80%, not chasing down the final 20%
- Requests and client-side caching are more important than server load
- Making it harder to make software makes software worse
- NEXT: This is how it works...

Modules

Scripts, styles
& messages

WIKIMEDIA

– Scripts styles and messages can be packaged together
– Or components of modules can be access individually

Modules
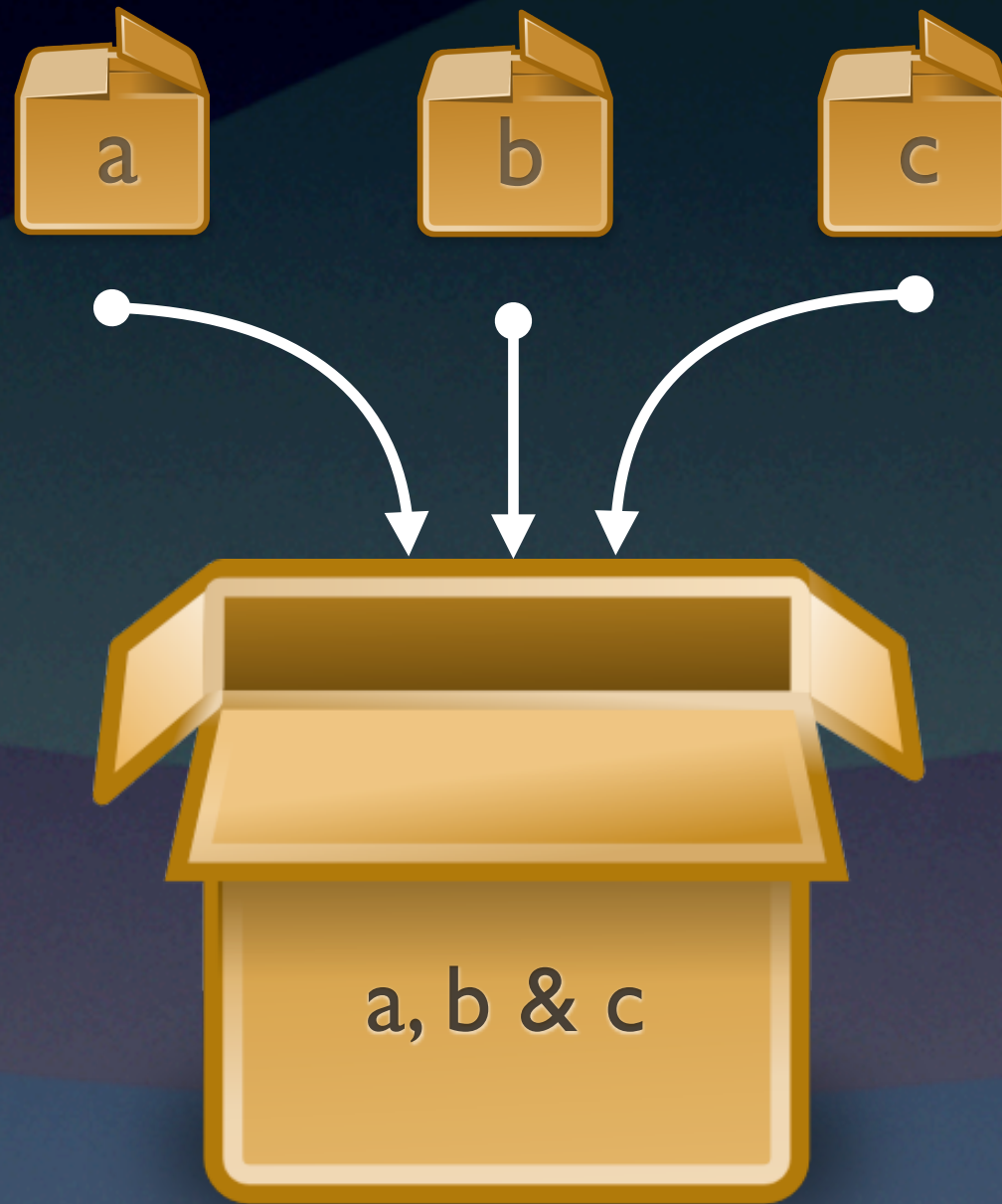
– Any number of modules can be bundled together in a single request

Scripts

Wrapping
delayed execution

Minification
whitespace removal

Conditions
debug, skin or language

WIKIMEDIA

Styles

Embedding
data uri sweetness

Remapping
relative urls still work

Flipping
free rtl support

– Who's used CSS sprites before?
– Explains sprites (good and bad)
– It's all magical and transparent!
– Remapping makes injected styles still work
– Flipping (more about that later)

Styles

Bundling
one request

Minification
whitespace removal

Conditions
skin specific styles

WIKIMEDIA

– Sent together with JS
– Whitespace is gone
– Skin-specific styles is good

Messages

Bundling

one request

Conditions

language

Startup Module

Sanity check
if (IE5) { giveUp(); }
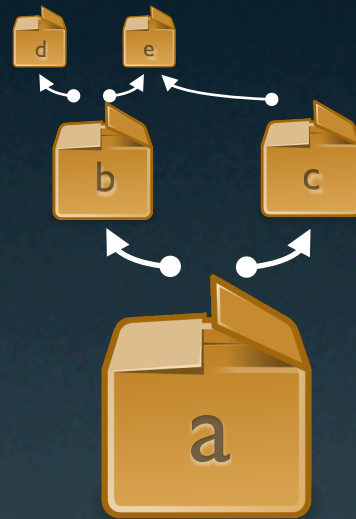
Dependencies
module manifest

Configuration
site-wide settings

WIKIMEDIA

– Bail on old–school browsers
– Provide module manifest for dependency resolution
– Tack on site–wide configuration

Client-side Loader

Resolution
calculate dependencies

Batching
give it to me all at once

Execution
run in correct order

WIKIMEDIA
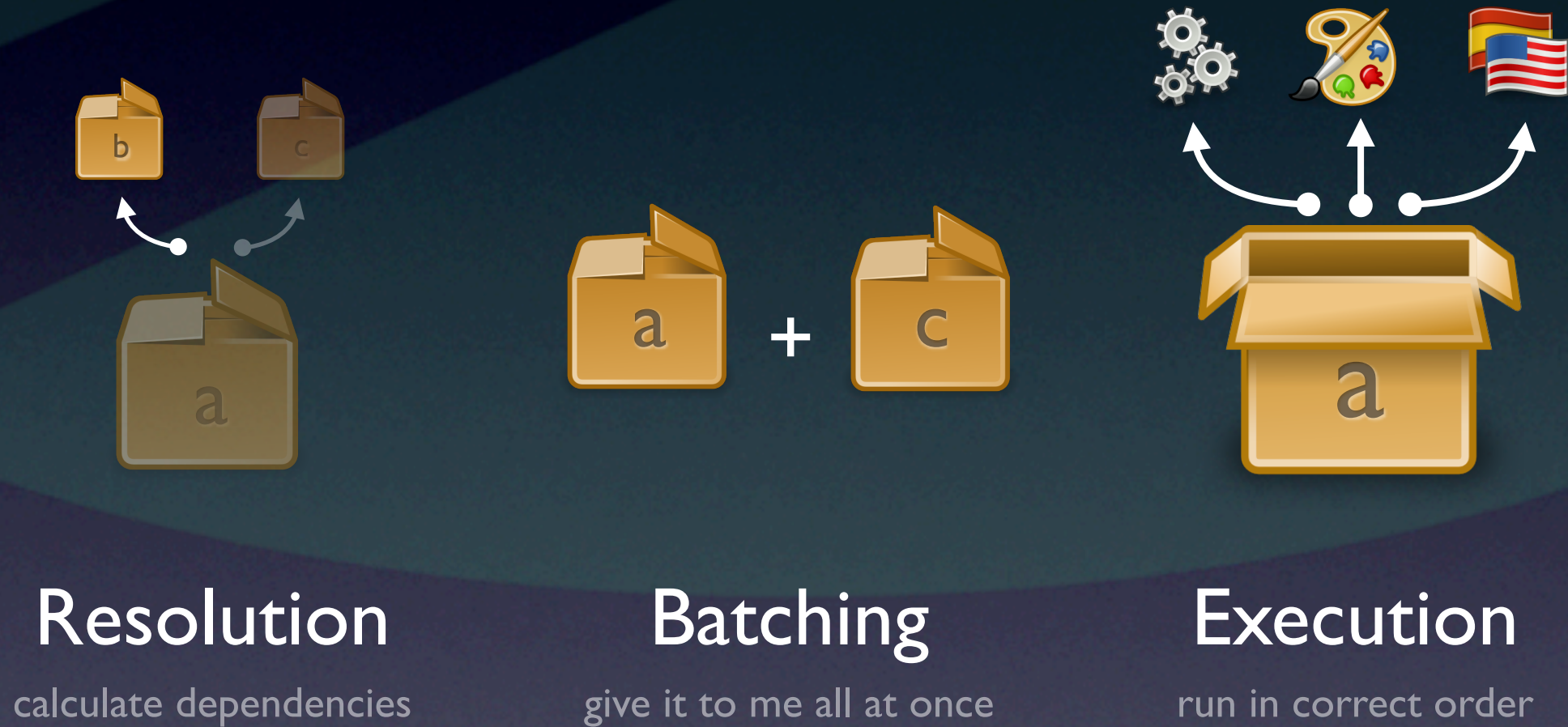
– Automatic resolution, never loads something twice
– Module are requested in batches
– Modules (scripts, styles, messages) are executed in proper order

Caching
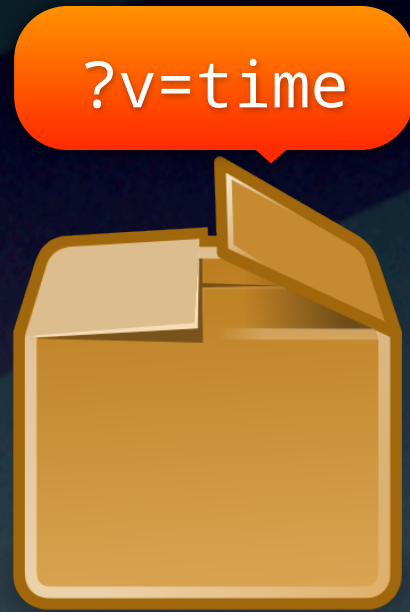
Versioning
per-module

?v=time

Startup Module
5 minutes

Resources
30 days

WIKIMEDIA

– We use timestamps, based on latest modified time of resources in module
– Startup module is cached for 5 min (talk about caching)
– Resources are cached for a month, could be forever

So, it turns out...

Embedding is sweet

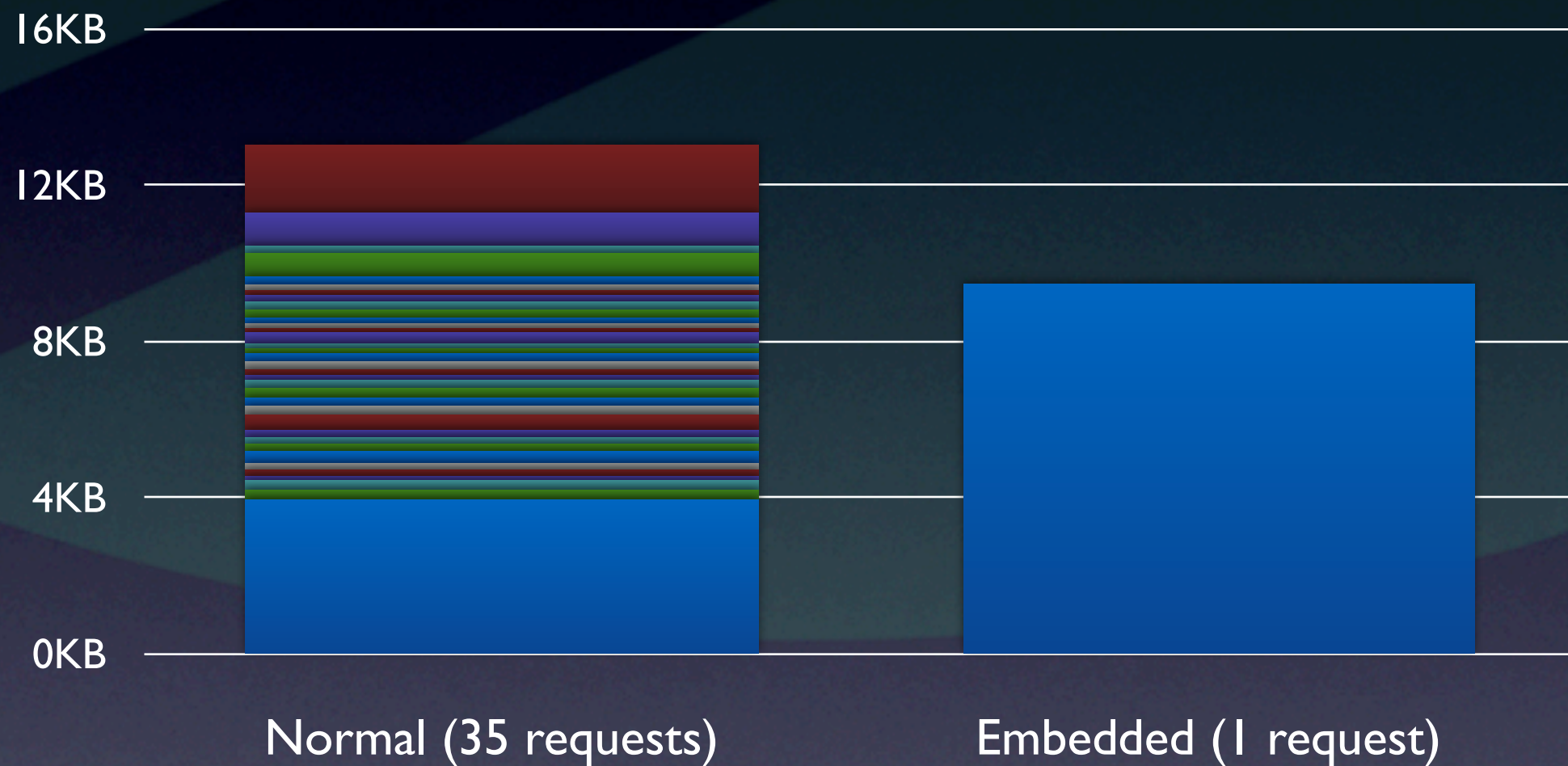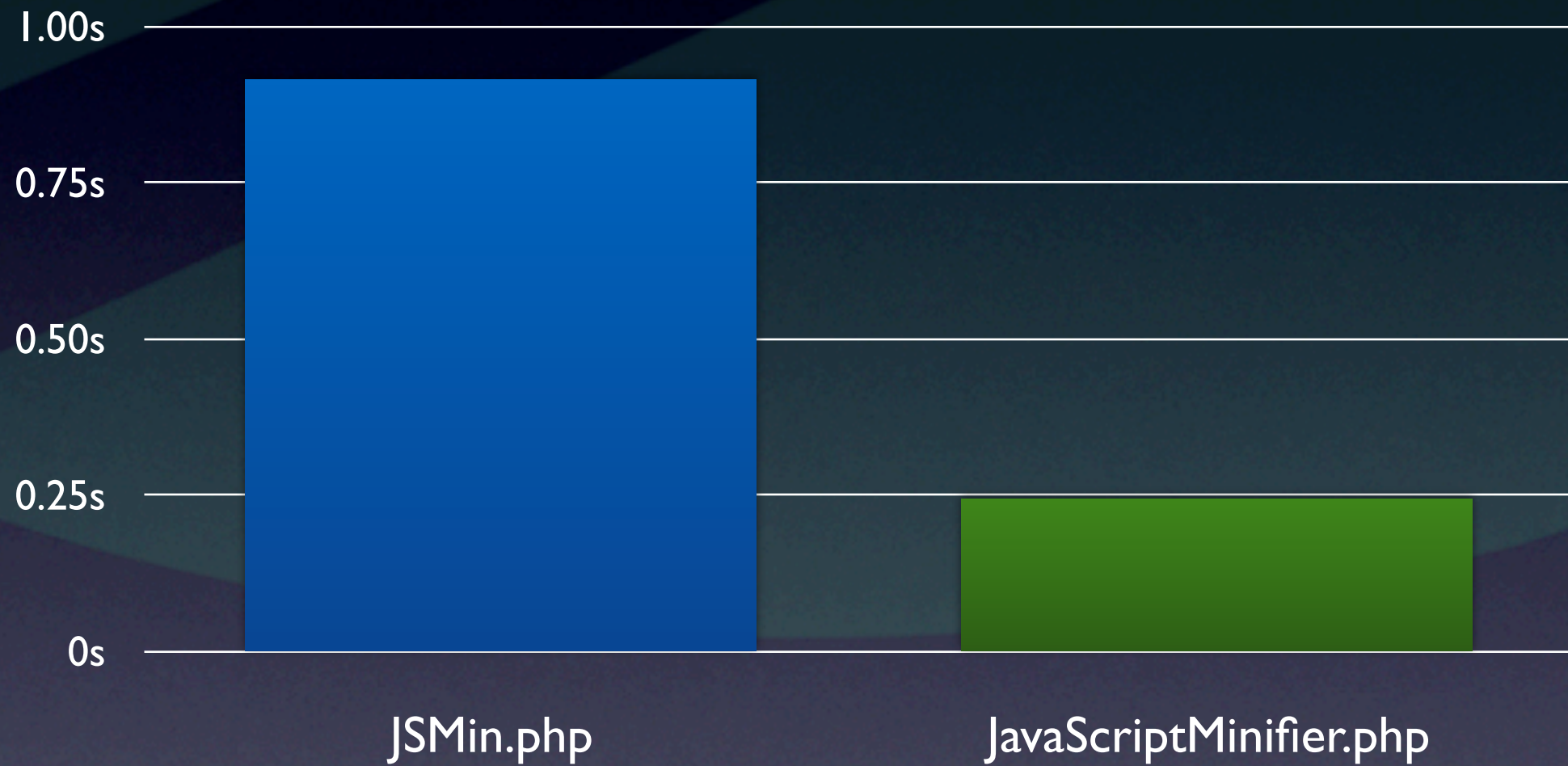Chart showing two bars: "Normal (35 requests)" with a stacked multi-colored bar reaching about 12.5KB, and "Embedded (1 request)" with a solid blue bar reaching about 9.5KB. Y-axis marked 0KB, 4KB, 8KB, 12KB, 16KB.

Test: embedding images in the Vector skin, 27.3% reduction in total size after compression, 97.2% reduction in requests

CSSMin: http://tinyurl.com/CSSMin-php

- Vector normally would have 35 requests, mostly images
- Despite base64 inflation, size is smaller after gzip
- Gzip gets a chance to combine common PNG headers and pixel data
- Plus, sprites are a pain to get right, to maintain, can't always be used
- Embedding is idiot-proof: just type /* @embed */

JSMin.php is slow

Test: minifying jQuery development distribution, 160KB of uncompressed code; 0.5% larger size than JSMin output after compression

JavaScriptMinifier: http://tinyurl.com/JavaScriptMinifier

WIKIMEDIA

- Written by Paul Copperman (volunteer)
- 0.5% difference
- More correct than JSMin!
- Stand-alone code, download and use!

CSS Janus is Awesome

php

Ported

from python

$humans--;

seriously, go robots!

CSSJanus: http://tinyurl.com/CSSJanus

WIKIMEDIA

– RTL for free!
– Can use @noflip when needed
– Even just blindly flipping styles, results are amazing

Balance is important
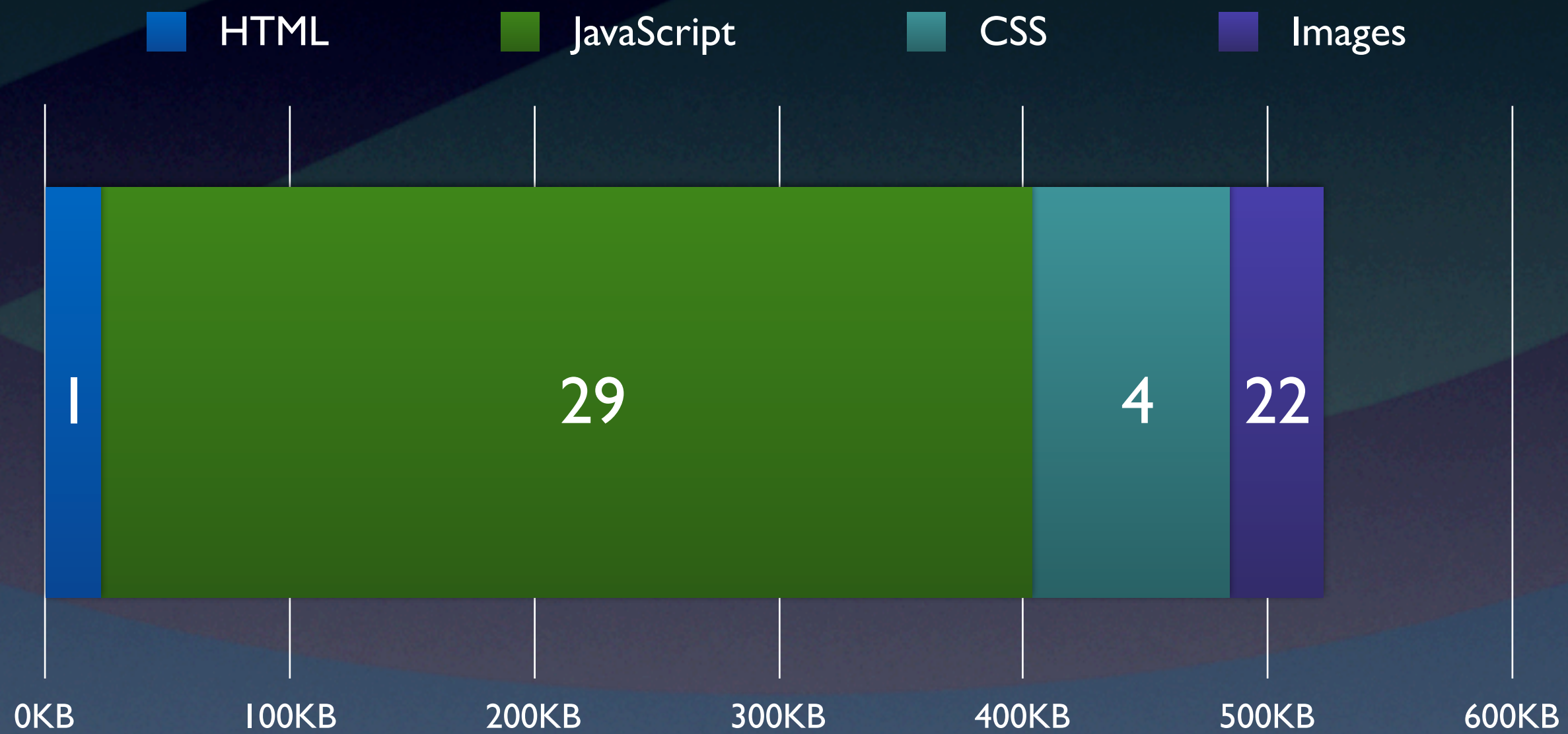

Batching
duplicate data may be sent

Groups
controlled fragmentation

WIKIMEDIA

– Overzealous batching causes cache fragmentation
– Added grouping to intentionally split requests
– Groups are defined in module manifest
– Typical value for b: JUI

– You recall this chart?

It Works!

HTML | JavaScript | CSS | Images

1 | 7 | 2 | 4

0KB | 100KB | 200KB | 300KB | 400KB | 500KB | 600KB

WIKIMEDIA
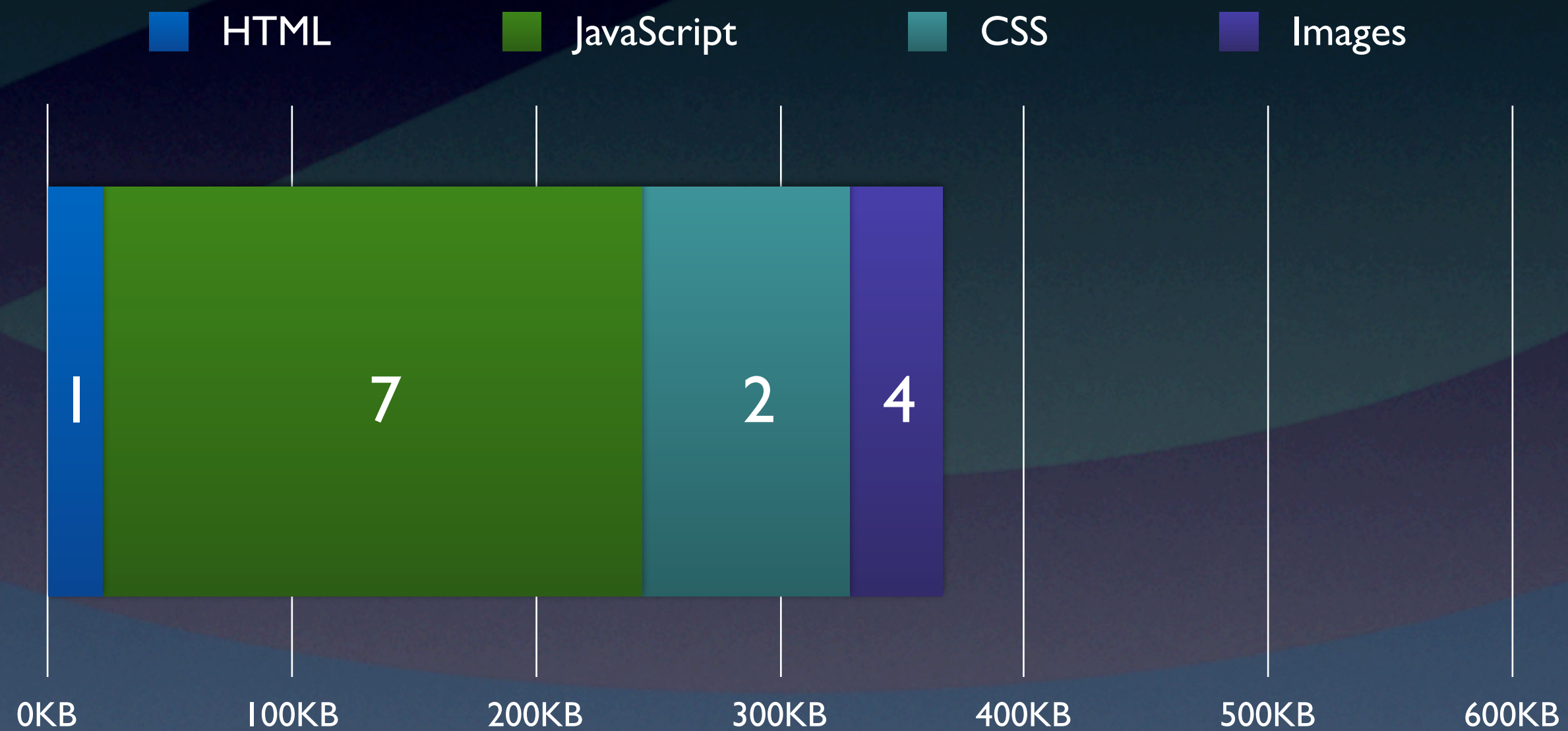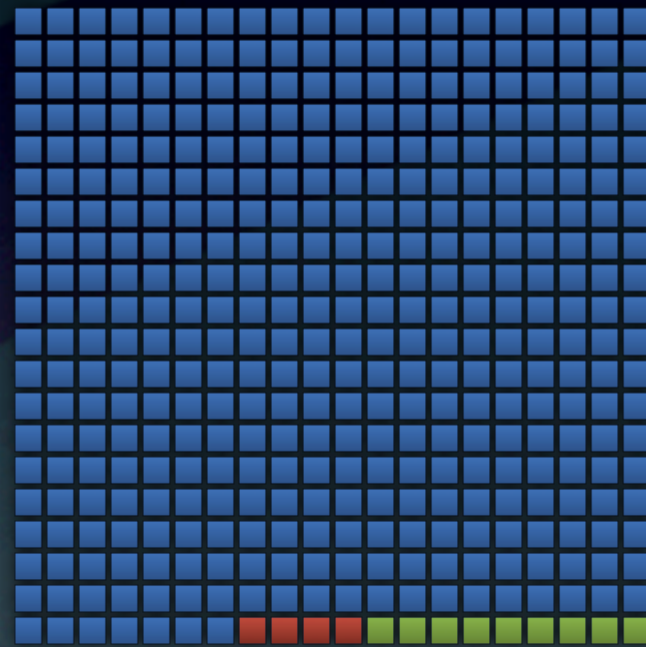
– Dramatic reduction in number of requests
– Less total data transfer, gzip does it's thing

It's Efficient!

Servers

~400 servers

For resources: 4 app, 9 cache

Requests

90k req/s peak load

of which 40k are for resources

and 73 are cache misses

cache hit rate: 98.2%

WIKIMEDIA

– Guess how many servers?
– 13 Resource servers (4 app, 9 cache)
– **Not even fully optimized**

ResourceLoader

Easy Gains
no micro-optimization

Client Focus
it's where the magic is

Ease of Use
developers use this stuff

WIKIMEDIA

– To recap, we employed these techniques, focusing on:
–– Easy gains
–– The client
–– Developer friendliness

ResourceLoader

Easy to Make
happy developers

Lightweight
happy servers

Snappy Pages
happy people

WIKIMEDIA

– Development is easier
– Servers are happier
– Users are happier
– Everyone wins

Thanks!

http://wikitech.wikimedia.org/view/Presentations

http://www.mediawiki.org/wiki/ResourceLoader

Trevor Parscal   -   @trevorparscal
Roan Kattouw   -   @catrope

WIKIMEDIA

– Download our presentation
– Download our code
– Build on our techniques
– Make the web a better place

Work @ Wikimedia

http://jobs.wikimedia.org

WIKIMEDIA

- Point to people they can talk to
- Tech-related and non-tech jobs too
- GO BACK TO PREVIOUS SLIDE NOW PLZ!