- 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
- Roan: Robot
- Trevor: Human
- Lots of crossover
- Understand the problem of client-rich development
- Know how to make things scale well
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
- 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
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!
- Where do these resources come from?
- Core software (filesystem)
- Extension software (filesystem)
- User scripts and styles (database)
- 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

- 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

- 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

- 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
- 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...
- Scripts, styles, and messages can be packaged together
- Or components of modules can be accessed individually
- Any number of modules can be bundled together in a single request
Scripts

Wrapping
  delayed execution

Minification
  whitespace removal

Conditions
  debug, skin or language
- 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)
- Sent together with JS
- Whitespace is gone
- Skin-specific styles is good
- 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

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

- 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

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

Ported from python

$humans--; seriously, go robots!

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

CSSJanus: [http://tinyurl.com/CSSJanus](http://tinyurl.com/CSSJanus)
Balance is important

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

- You recall this chart?
It Works!

- 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%

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

- 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

- Download our presentation
- Download our code
- Build on our techniques
- Make the web a better place
Work @ Wikimedia

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