Flywheel: Google's Data Compression Proxy for the Mobile Web

Victor Agababov, Michael Buettner, Victor Chudnovsky,
Mark Cogan, Ben Greenstein, Shane McDaniel,
Michael Piatek, Colin Scott*, Matt Welsh, Bolian Yin

Google

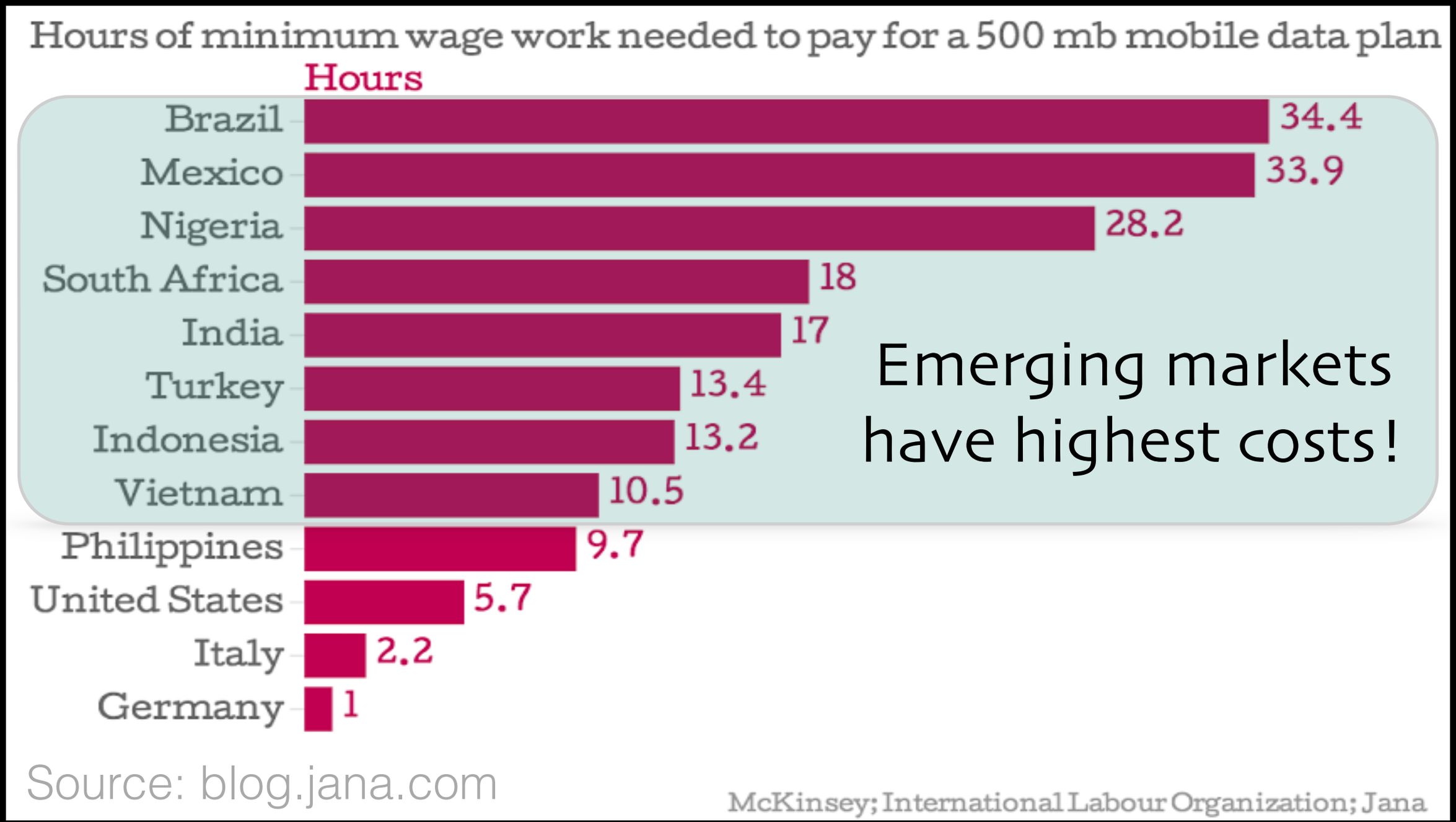
cs@cs.berkeley.edu mdw@google.com

*Currently a graduate student at UC Berkeley

Dominant Access Tech: Mobile

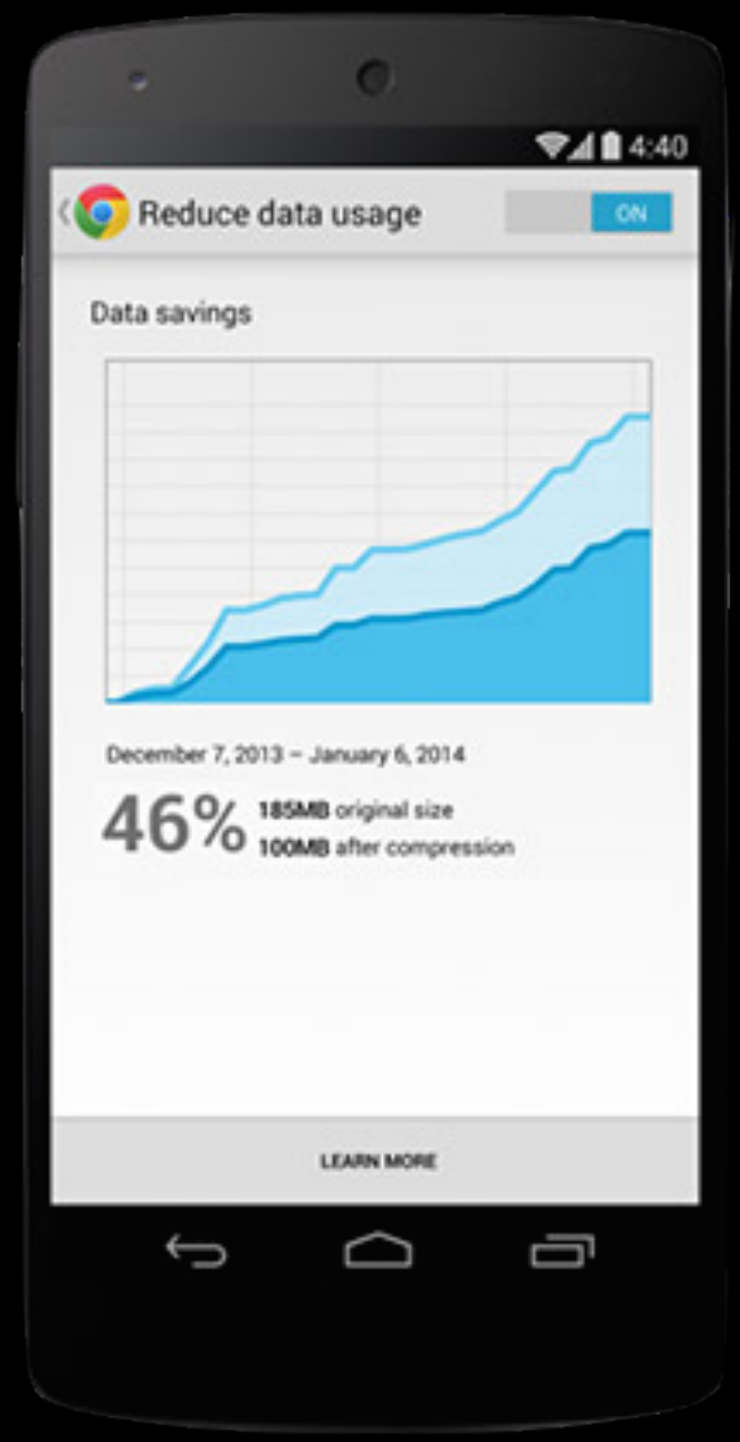
- Mobile devices are increasingly dominant
- Growth is greatest in emerging markets

Mobile Data is Expensive

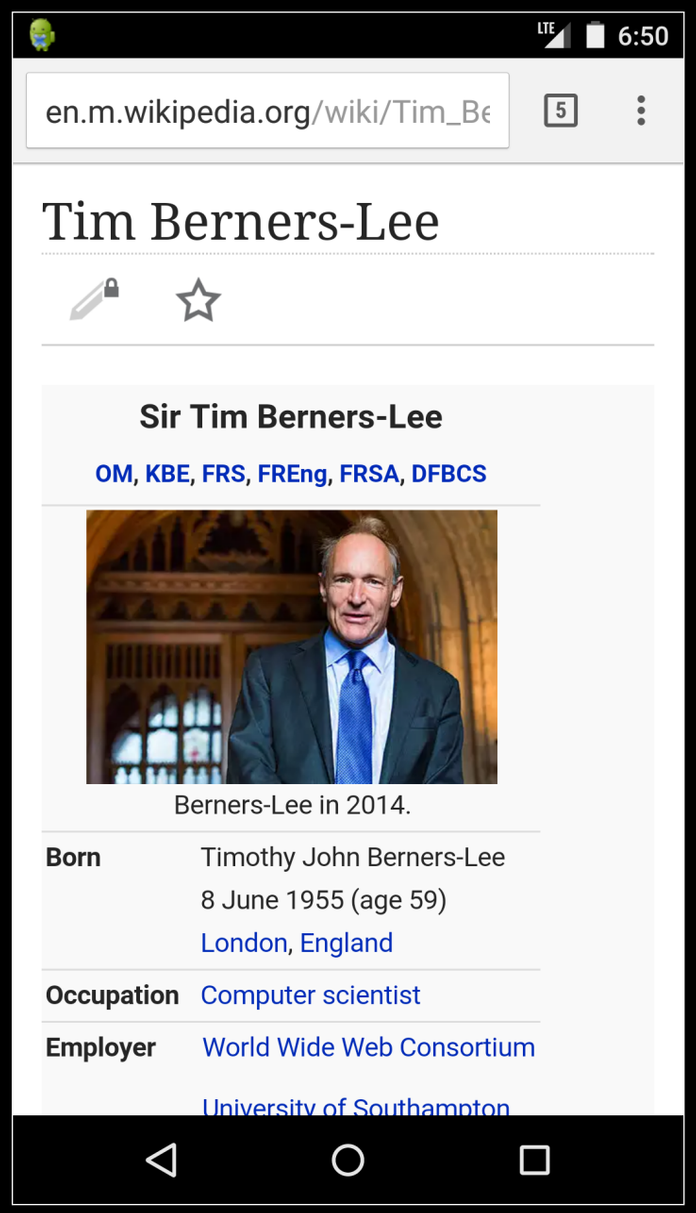


Flywheel

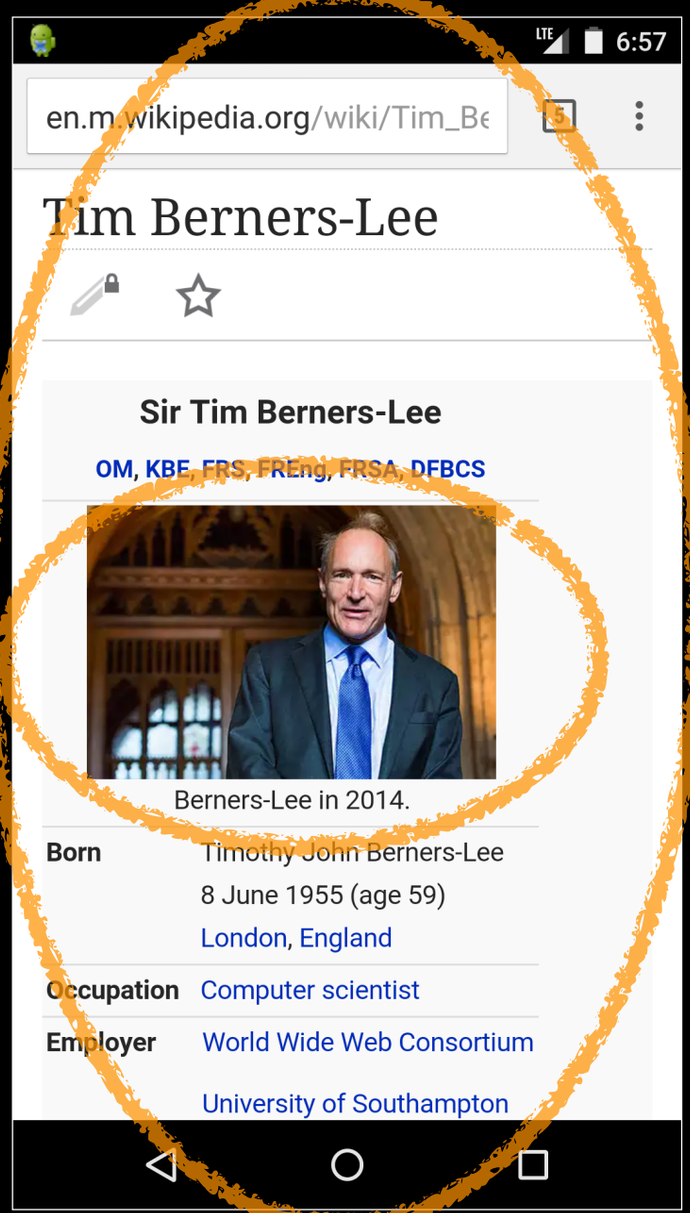
- Flywheel: proxy service that optimizes HTTP response size
  - Three years of deployment experience, part of Chrome for Android, iOS, Desktop
  - Currently serving millions of users & billions of requests per day



What Flywheel Does



Total bytes: 9764



Total bytes: 5565

Transcode to WebP

Pick quality level

Minify CSS, JS

GZip text objects

None of our optimizations are novel

What lessons did we learn from building and operating Flywheel?

Key lessons:

- Highly challenging to maintain good performance
- Tussles are pervasive, ongoing, & time-consuming

Outline

- Is a proxy really needed?
- What's hard about engineering Flywheel?
- Does Flywheel meet our goals?
- What can be learned?

The web isn't well optimized for mobile

- Case in point:
  - 42% of HTML response bytes not compressed

Hard to keep up with best practices

- New optimizations: WebP, SDCH, HTTP/2,... rolled out as often as every 6 weeks
- Heterogeneity of mobile devices increasing

Need: Optimizing service for the mobile web

Outline

- Is a proxy really needed?
- What's hard about engineering Flywheel?
- Does Flywheel meet our goals?
- What can be learned?

Design Constraints

- Opt-in deployment model
- Transparent to users
- HTTP only (No HTTPS)

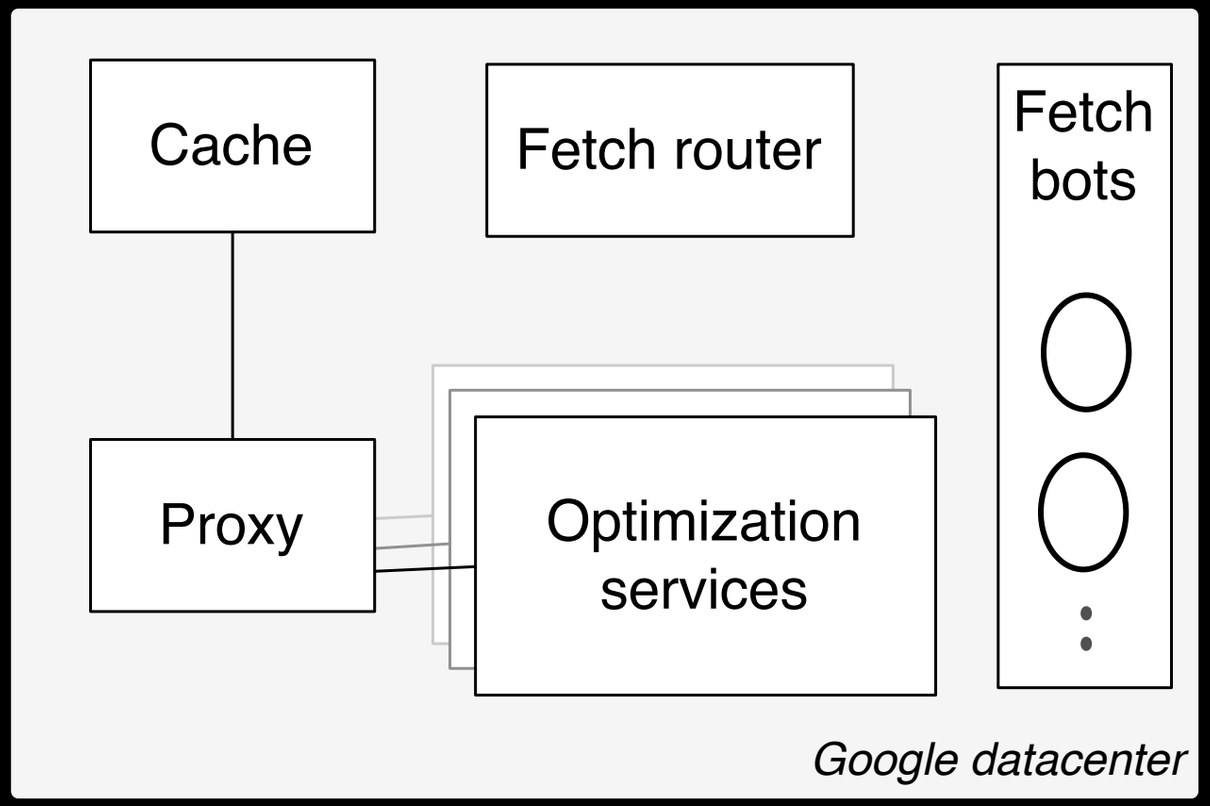
Challenge: trading off latency vs compression

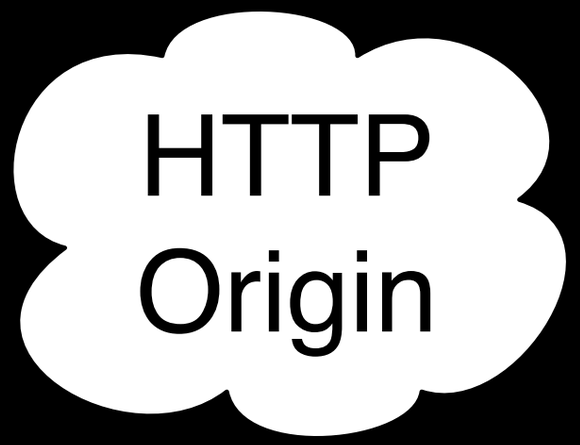
Indirection through Flywheel often increases RTT

Network latency is dominant performance factor
(Page size is not a dominant factor!)

Flywheel Design

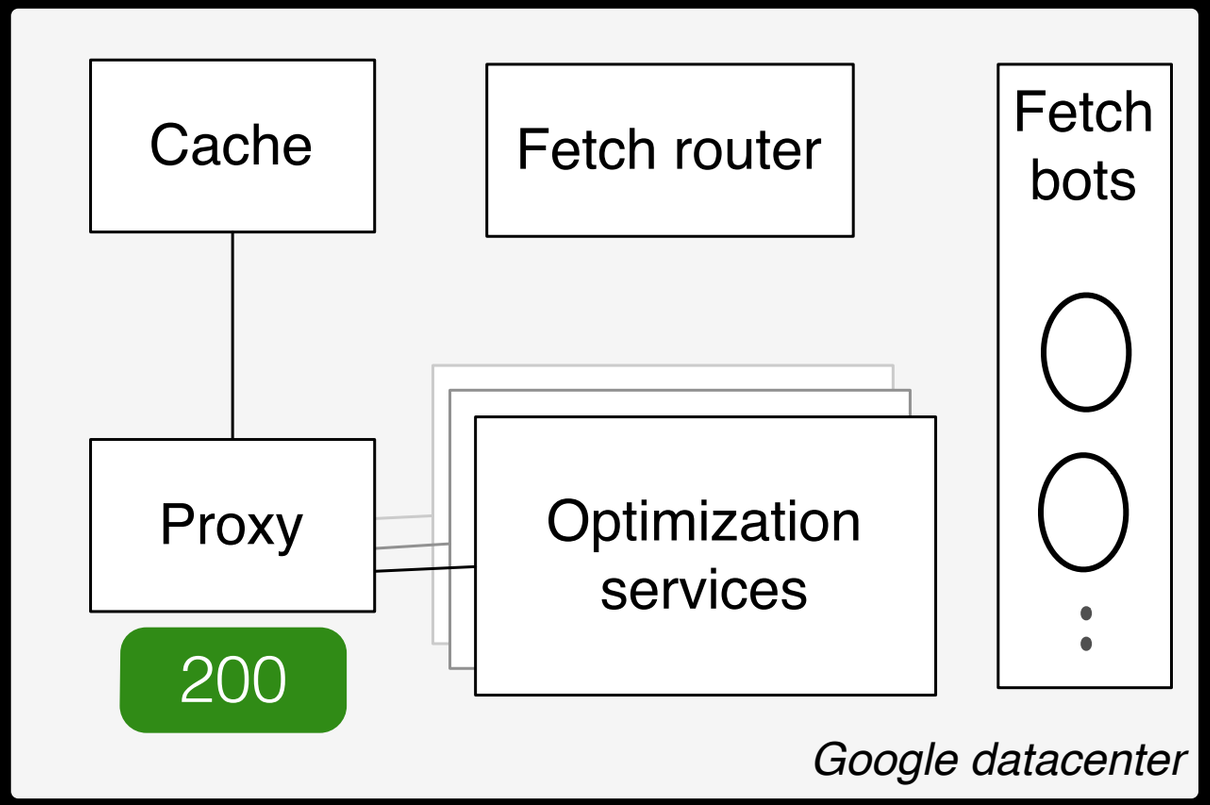
GET /





Fetch router maintains connection affinity

Flywheel Design



Optimizations: image transcoding, GZip, minification ...
Separate optimization services for isolation, provisioning

Selective Proxying

Indirection through Flywheel often increases RTT

GET /

HTTP Origin

Fetch objects on critical path from origin

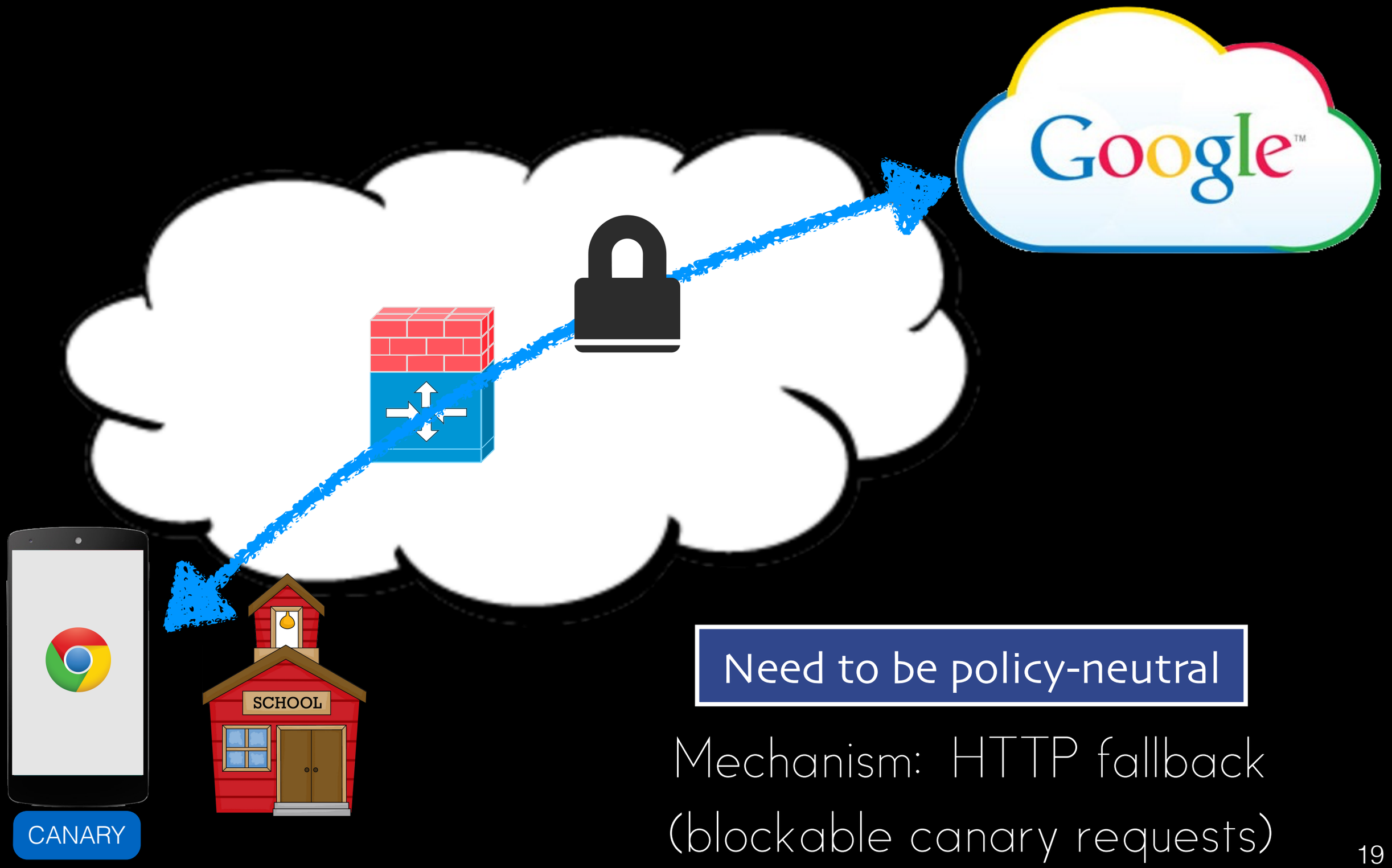
Selective Proxying

GET /i.jpg

HTTP Origin

Fetch objects on critical path from origin
Proxy objects that yield high data reduction

Challenge: Accommodating Tussles



Need to be policy-neutral

Mechanism: HTTP fallback (blockable canary requests)

Outline

- Is a proxy really needed?
- What's hard about engineering Flywheel?
- Does Flywheel meet our goals?
- What can be learned?

Evaluation

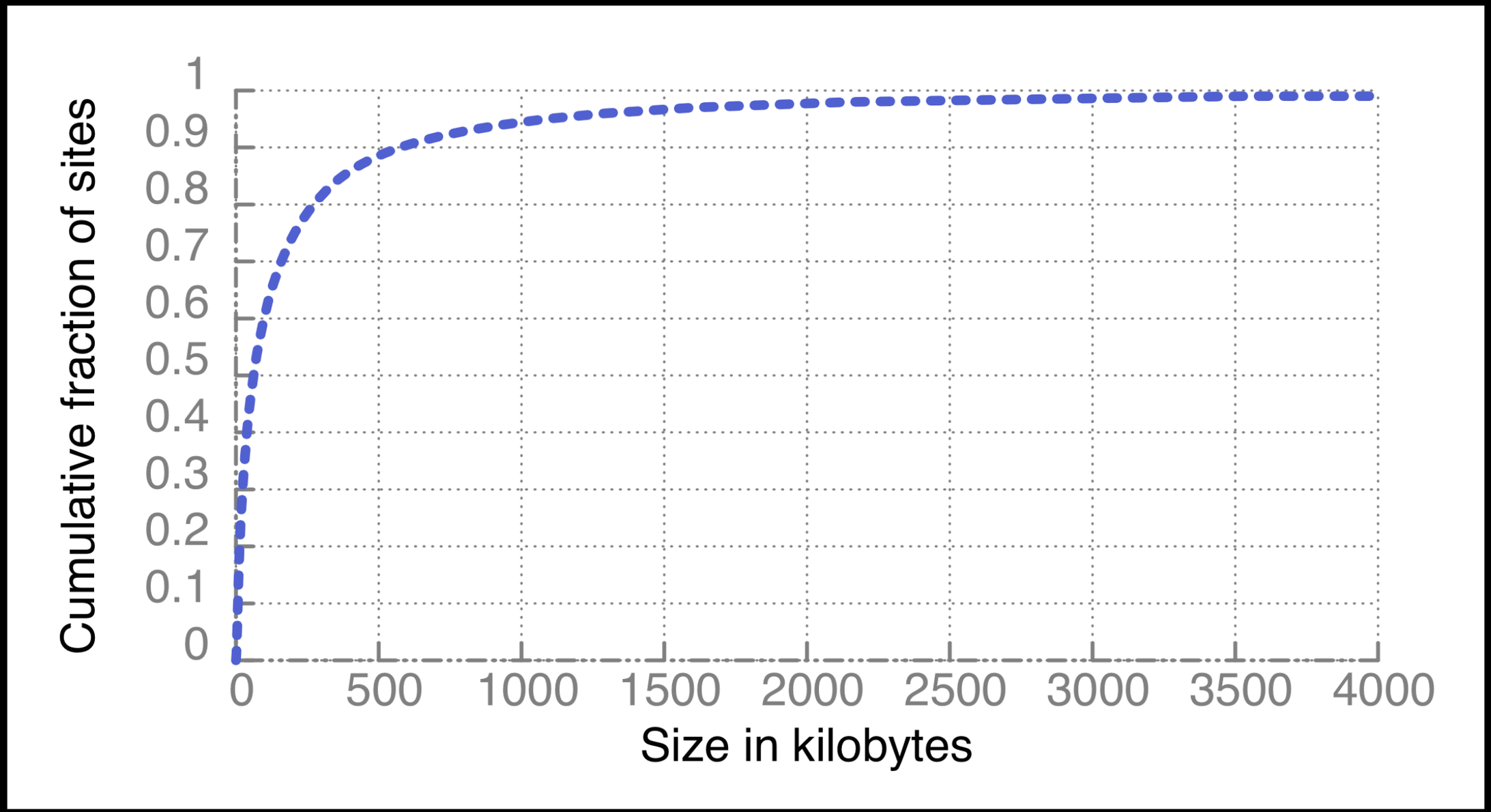
- This talk:
  - Primary goal: reduce web page size
  - Secondary goal: maintain good performance
- Paper:
  - Fault tolerance

Workload: Geographic Adoption

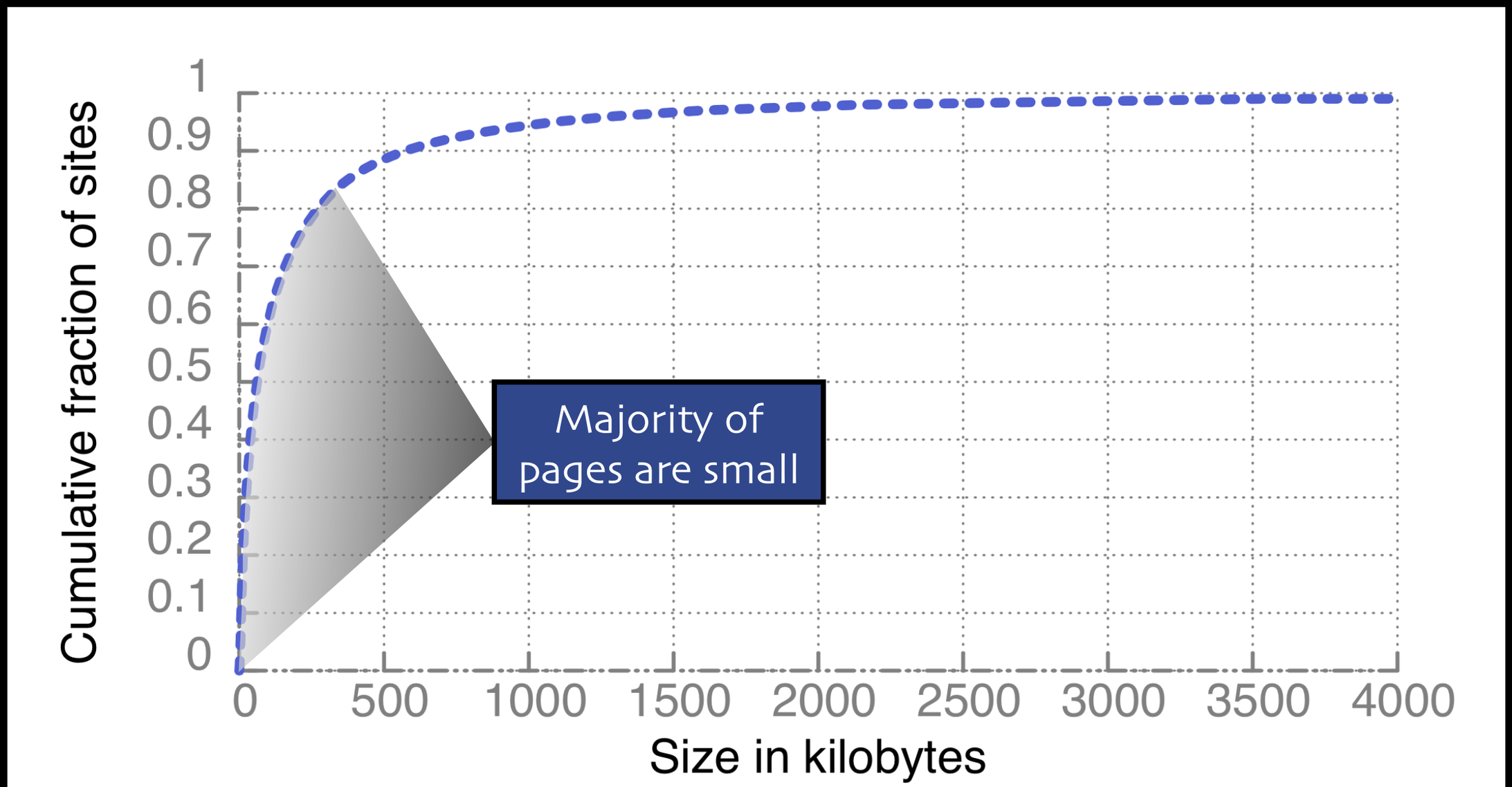
Country	Adoption
Worldwide	**10.5%**
Brazil	17%
Russia	16.5%
Indonesia	16.3%
Mexico	15.5%
USA	9.5%

Adoption highest in developing markets

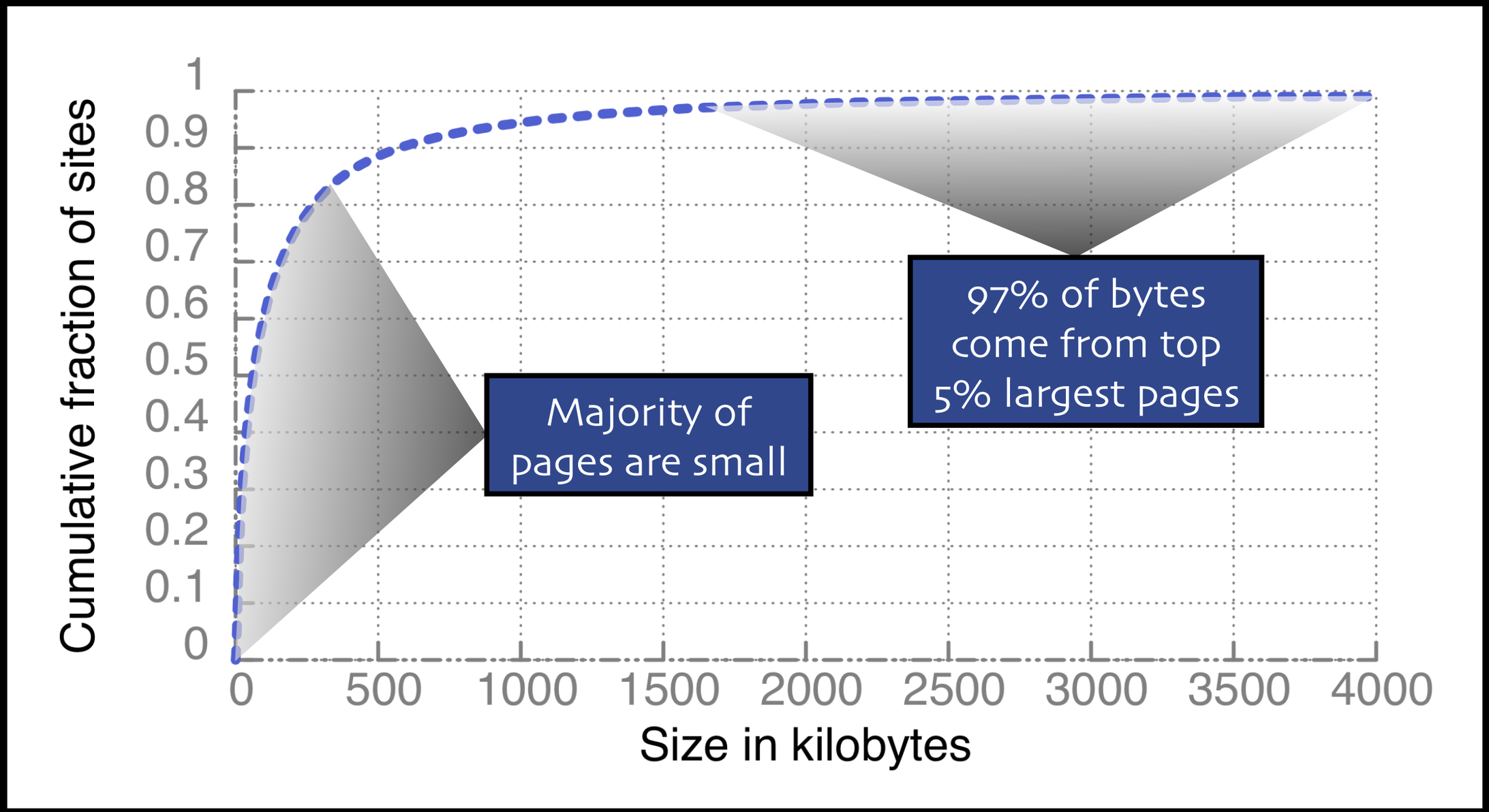
Workload: Page Footprints



Workload: Page Footprints



Workload: Page Footprints



Data Reduction

Savings = 1 - (outgoing bytes / incoming bytes) * 100

Type	% of Bytes	Savings	Share of Benefit
Total	**100%**	**58%**	**-**
Images	74.12%	66.40%	85%
HTML	9.64%	38.43%	6%
JavaScript	9.10%	41.09%	6%
CSS	1.81%	52.10%	2%
Other	5.33%	9.23%	1%

Data Reduction

Savings = 1 - (outgoing bytes / incoming bytes) * 100

Type	% of Bytes	Savings	Share of Benefit
Total	**100%**	**58%**	**-**
Images	74.12%	66.40%	85%
HTML	9.64%	38.43%	6%
JavaScript	9.10%	41.09%	6%
CSS	1.81%	52.10%	2%
Other	5.33%	9.23%	1%

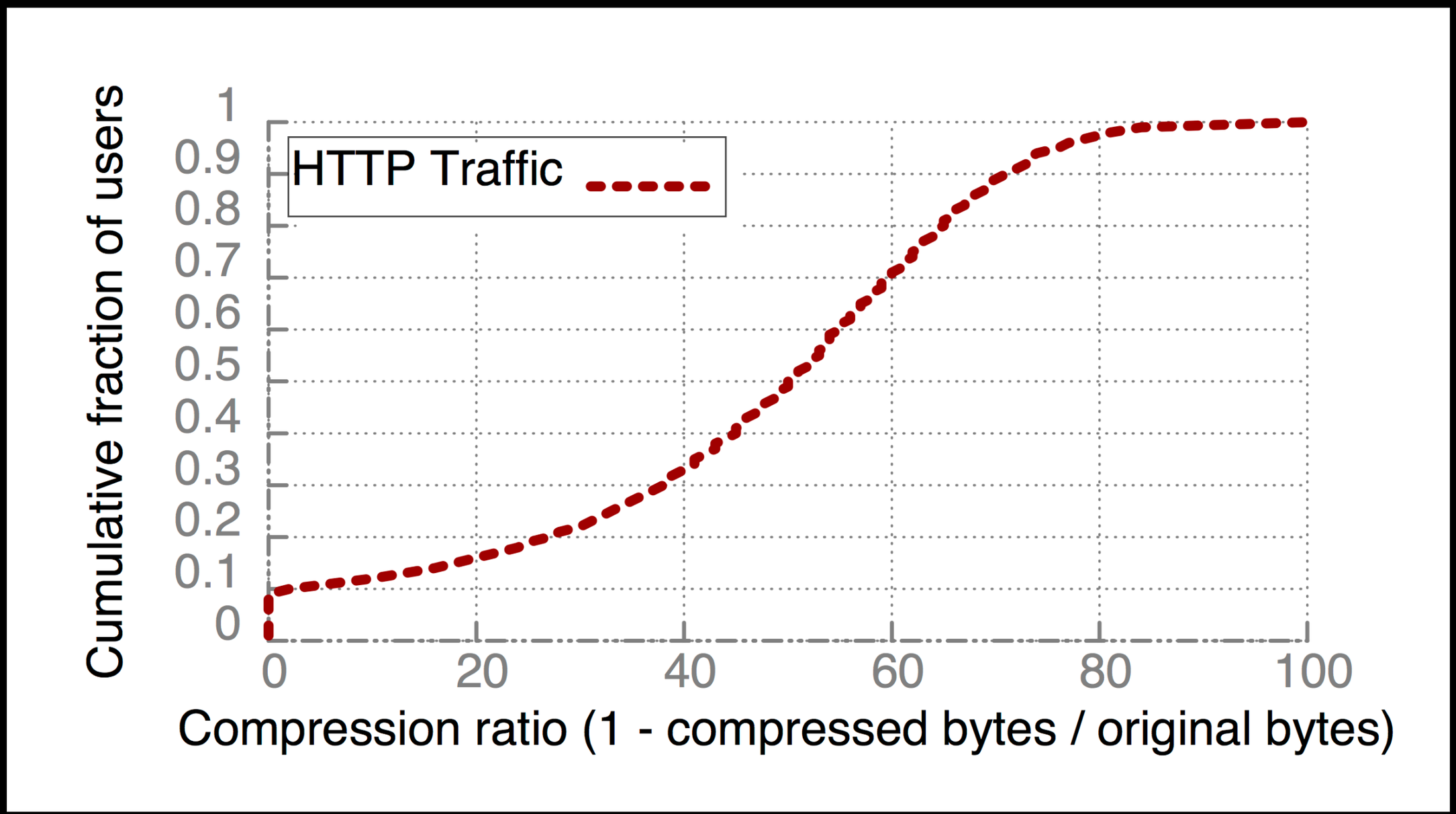
Data Reduction

Savings = 1 - (outgoing bytes / incoming bytes) * 100

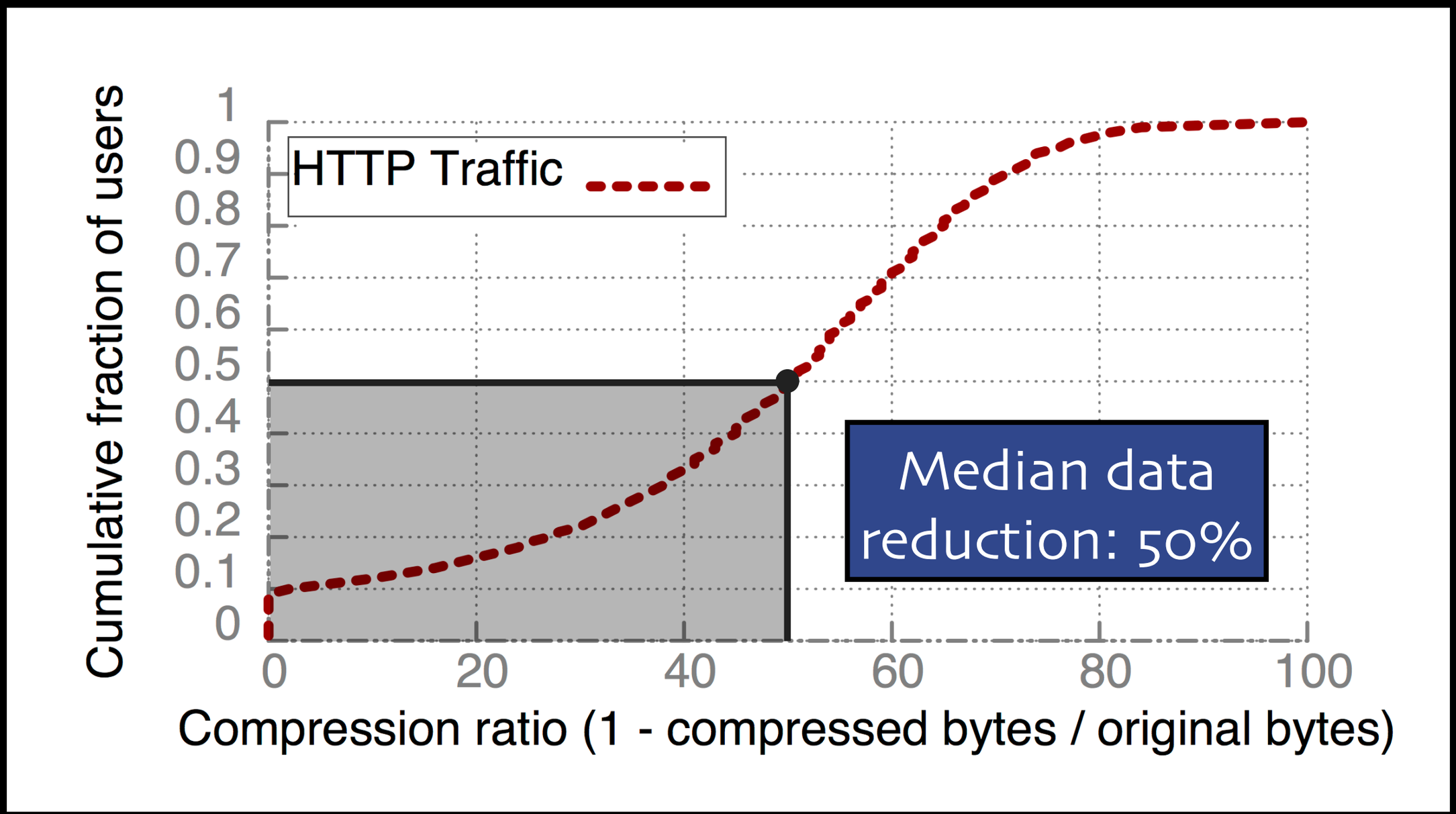
Type	% of Bytes	Savings	Share of Benefit
Total	**100%**	**58%**	**-**
Images	74.12%	66.40%	85%
HTML	9.64%	38.43%	6%
JavaScript	9.10%	41.09%	6%
CSS	1.81%	52.10%	2%
Other	5.33%	9.23%	1%

Images are bulk of bytes & savings

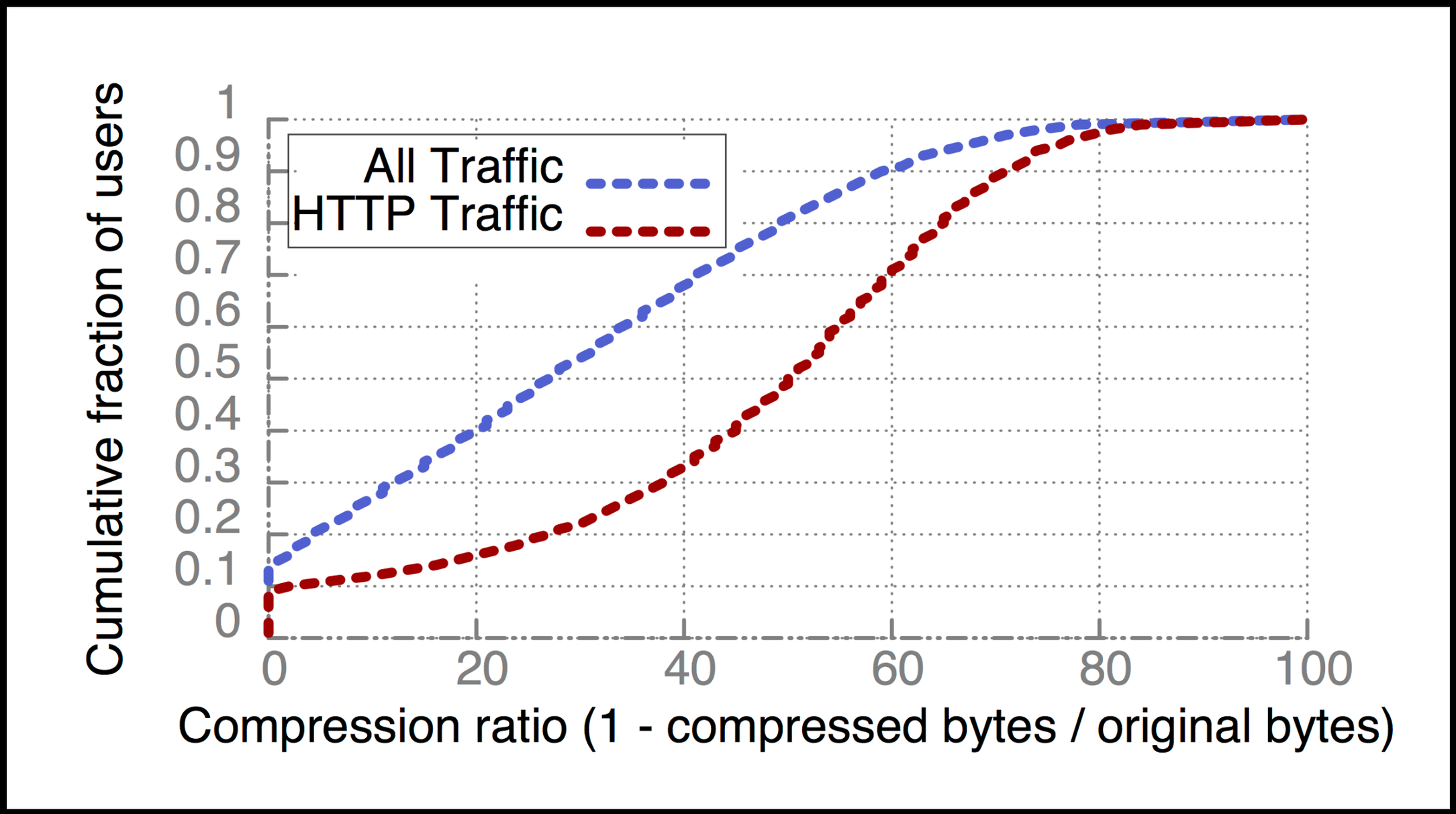
Reduction Across Users



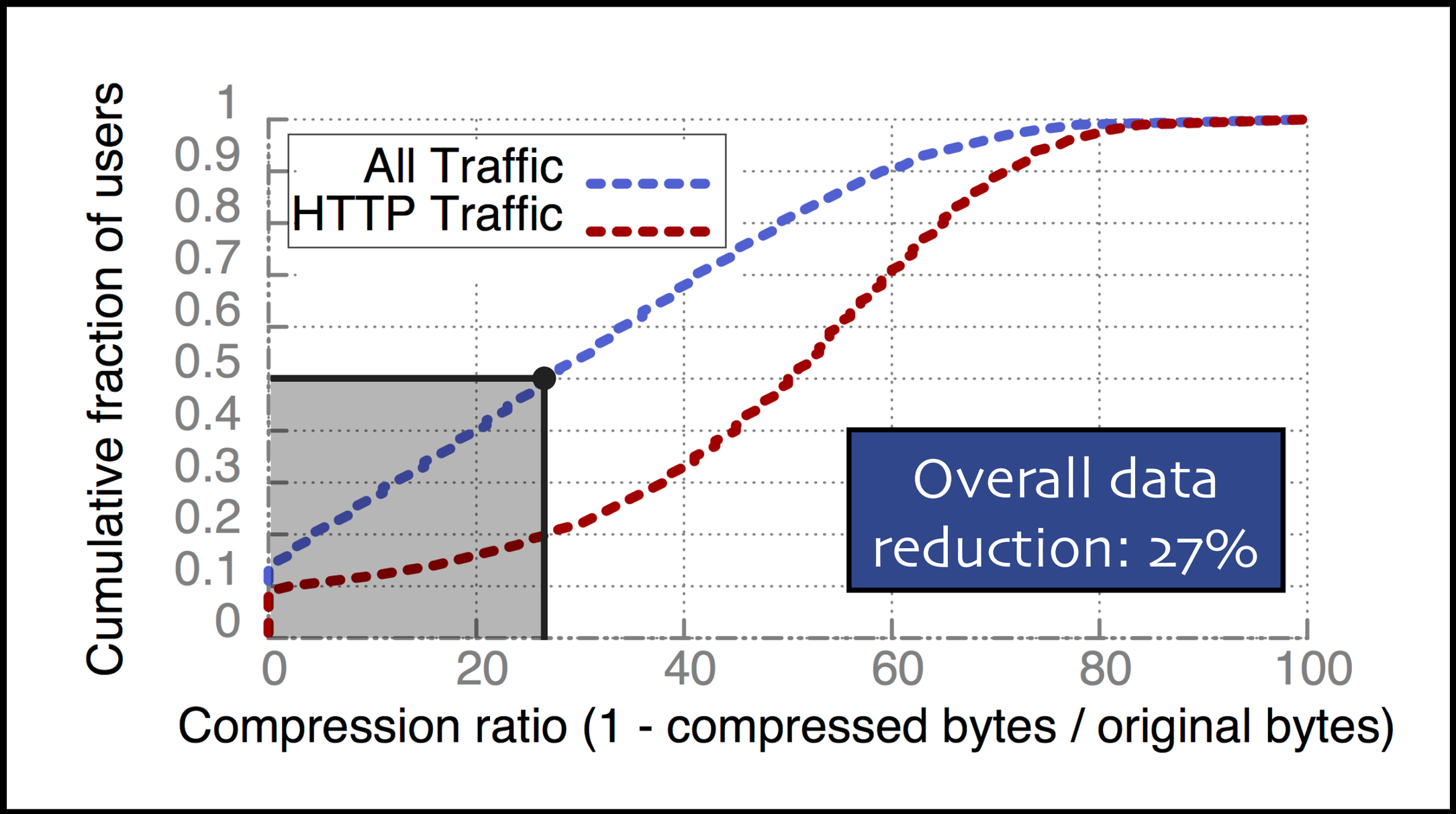
Reduction Across Users



Reduction Across Users



Reduction Across Users



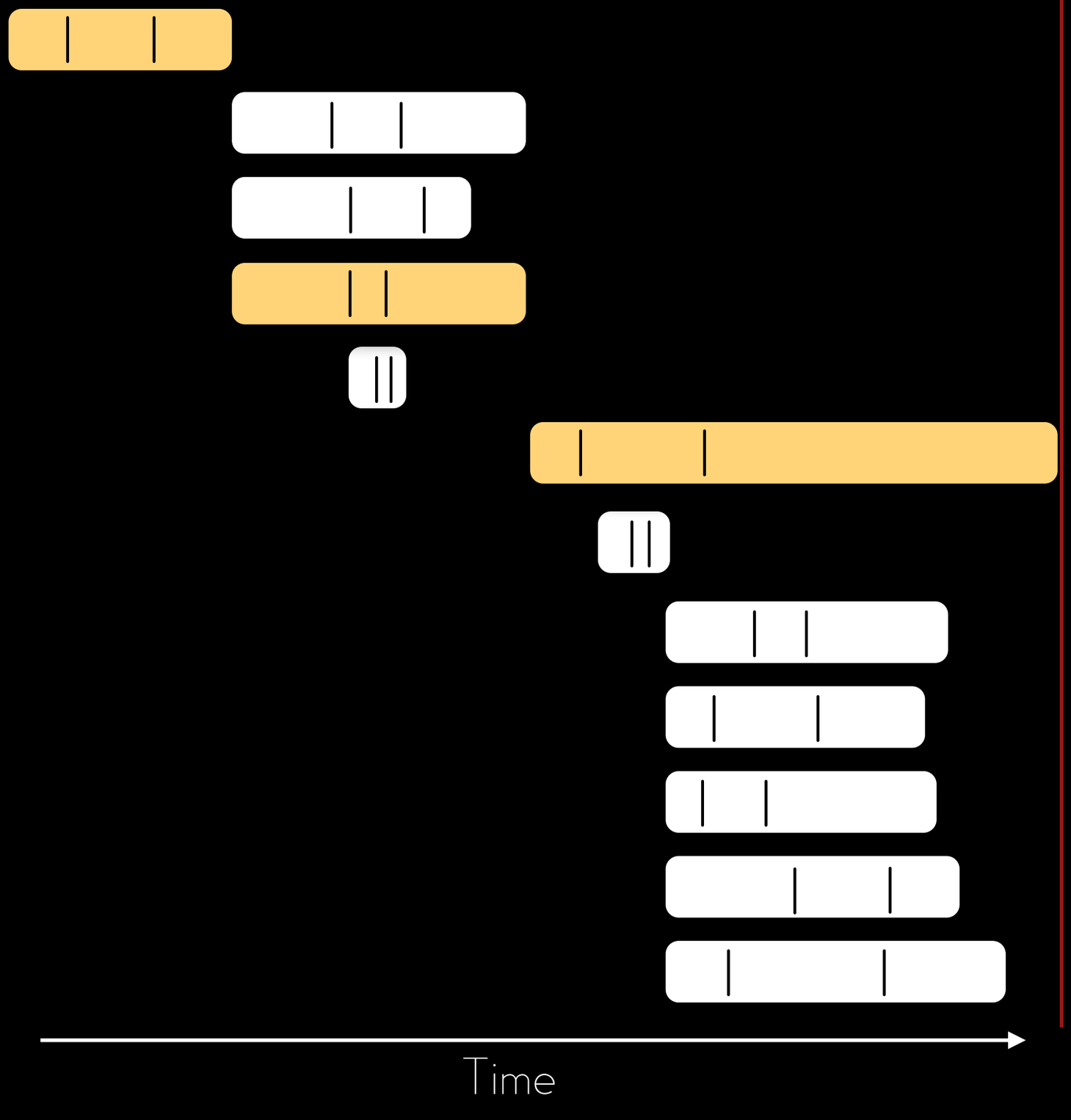
Performance: Methodology

- Goal: don't degrade performance

Compare:

- Random sampling of Flywheel users
- Holdback experimental group

Simple Model of Page Load Time

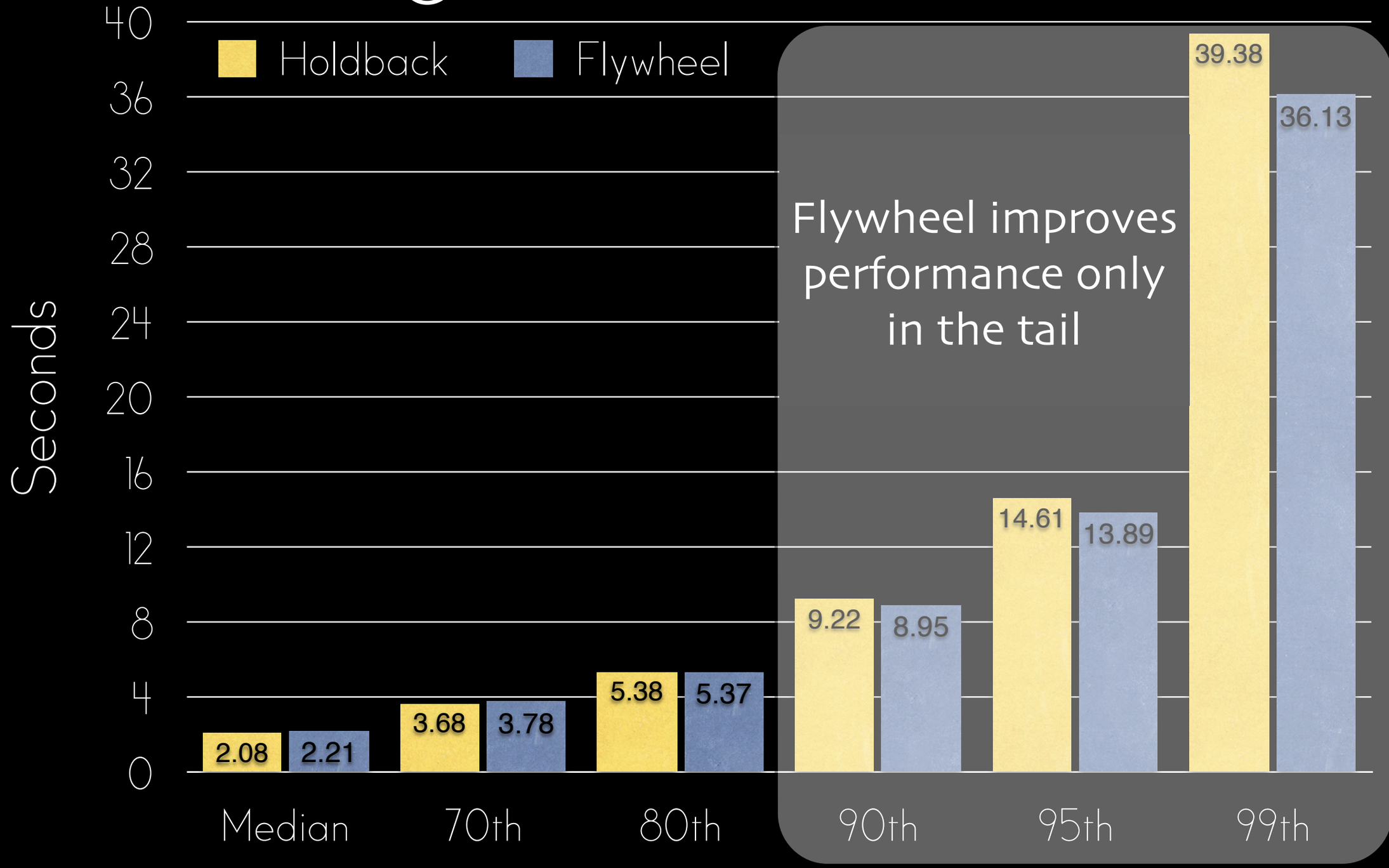


Load time of subresource s_i = propagation delay + transmission delay + computation time

Critical path = longest chain of dependent subresources

Page load time = Σ s_i on critical path

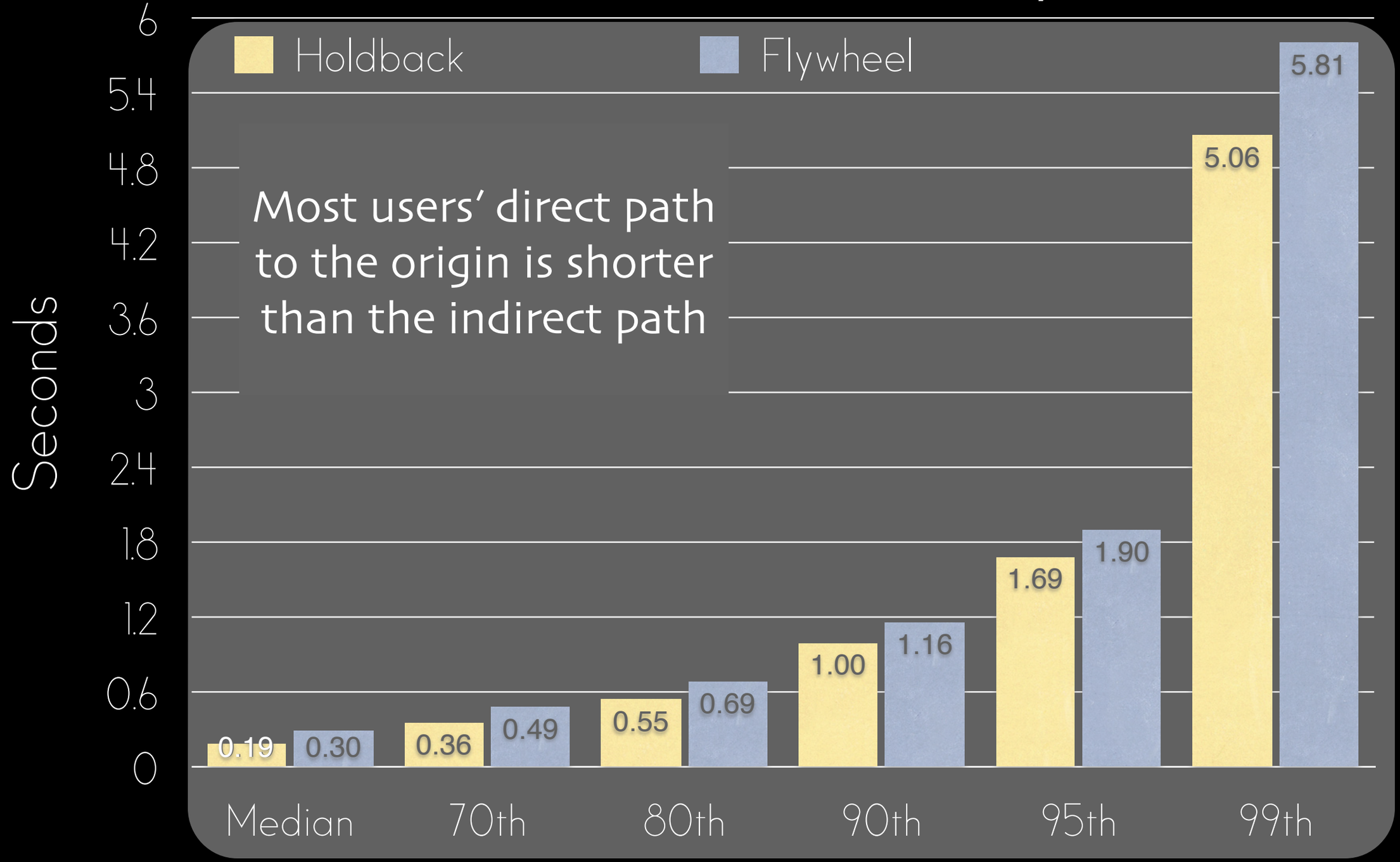
Page Load Time



Why is this?

- Recall our workload:
  - Long tail of large pages → Transmission delay dominant factor
  - Most pages are small → Propagation delay dominant factor
- Good way to understand propagation delay: time to first byte (TTFB)

Time to First Byte



Outline

- Is a proxy really needed?
- What's hard about engineering Flywheel?
- Does Flywheel meet our goals?
- What can be learned?

Lessons Learned

Many performance optimizations
we expected to have impact
did not at Web scale

Disappointing Optimizations

- Preconnect: open TCP connection with origin early

- Increases reused connections from 73% to 80%

- Yields less than 2% decrease in median PLT

Already a strong tendency for connection affinity

Disappointing Optimizations

- Prefetch: request cacheable subresources early

- Increases cache hit ratio from 22% to 32%

- Yields less than 2% decrease in median PLT

Cacheable items often aren't on the critical path

Lessons Learned

Improving PLT is highly challenging

- Compression doesn't help (much)
- Difficult to target critical path

Lessons Learned

If you want widespread adoption, you must accommodate policy issues!

Lessons Learned

Many more measurement findings and lessons in the paper!

Conclusion

- Flywheel shows it's possible to provide 58% average HTTP data reduction at web scale
- Data reduction is the easy part
- Maintaining performance and accommodating tussles are the hard part

lmgtfy.com/?q=Chrome+Data+Saver

cs@cs.berkeley.edu mdw@google.com