ACM SIGMETRICS / IFIP PERFORMANCE 2022

Mumbai, India

June 6-10, 2022

Fusing Speed Index during Web Page Loading

Wei Liu, Xinlei Yang, Hao Lin, Zhenhua Li, Feng Qian

清華大學
Tsinghua University

University of Minnesota
Driven to Discover®

Outline

1. Background
2. Motivation
3. Design
4. Evaluation
5. Conclusion

1. Background

❑ Web page load performance is important

WordPress.com

Over **409 million people** view more than **20 billion pages** each month.

ContentKing for conductor

Amazon study: Every 100ms in Added Page Load Time Cost 1% in Revenue

Last updated: AUGUST 10, 2021

Forbes

LEADERSHIP

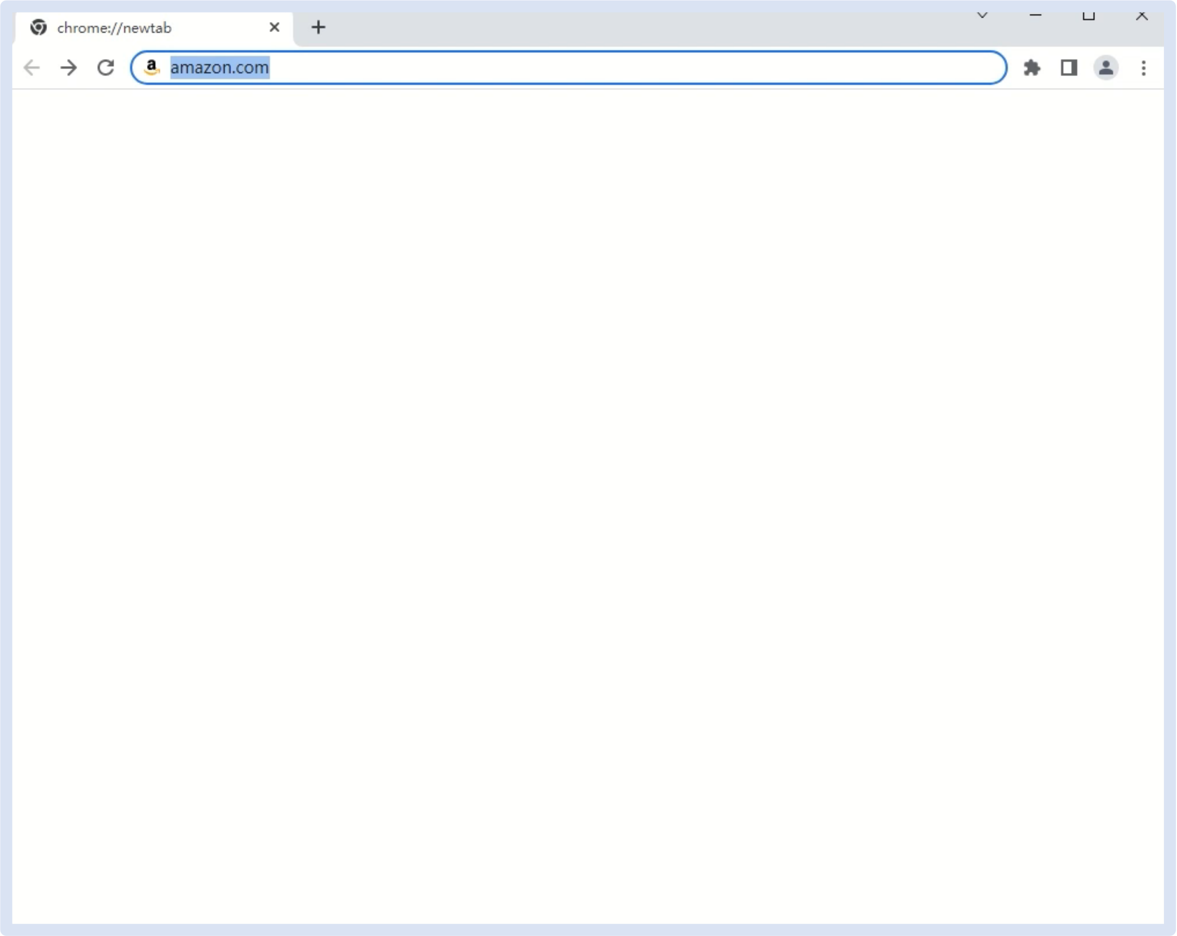
Why Site Speed Should Be An Optimization Priority

1. Background

Metrics to evaluate page load performance

Conventional metrics

- Page Load Time (PLT)
- Time to First/Largest Paint
- Time to Interactive



1. Background

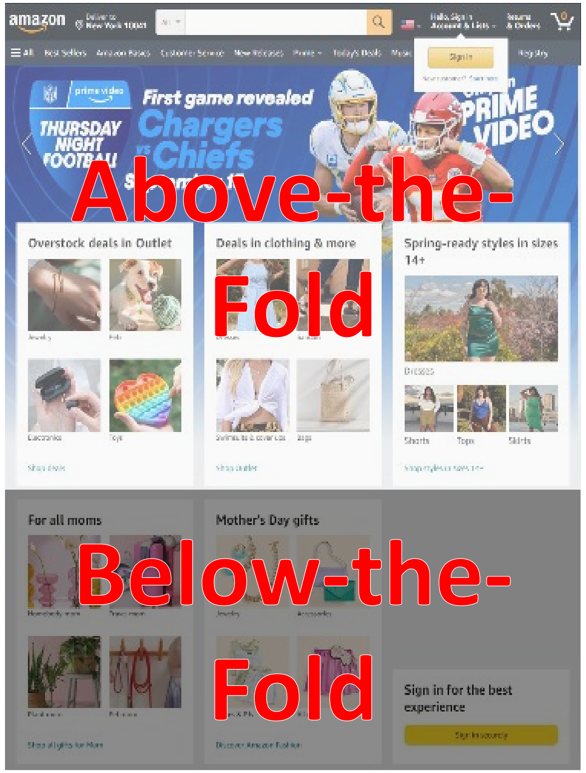
More advanced metrics

- Above-the-Fold Time (AFT)
- Object Index
- Byte Index
- **Speed Index (SI)**
- More...

1. Background

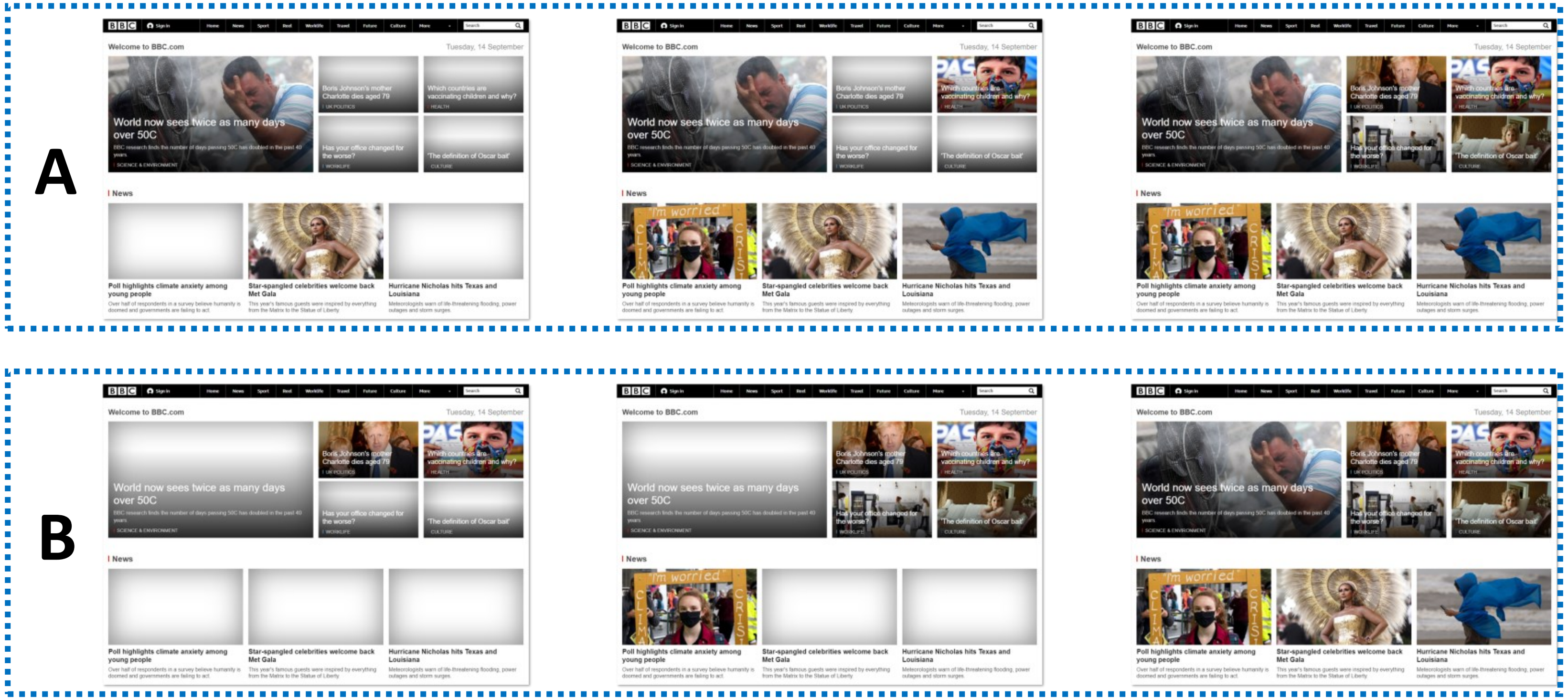
Speed Index (SI)

- How fast the page is filled up with the ***above-the-fold visible*** **elements** (i.e., **crucial elements**)



A

B



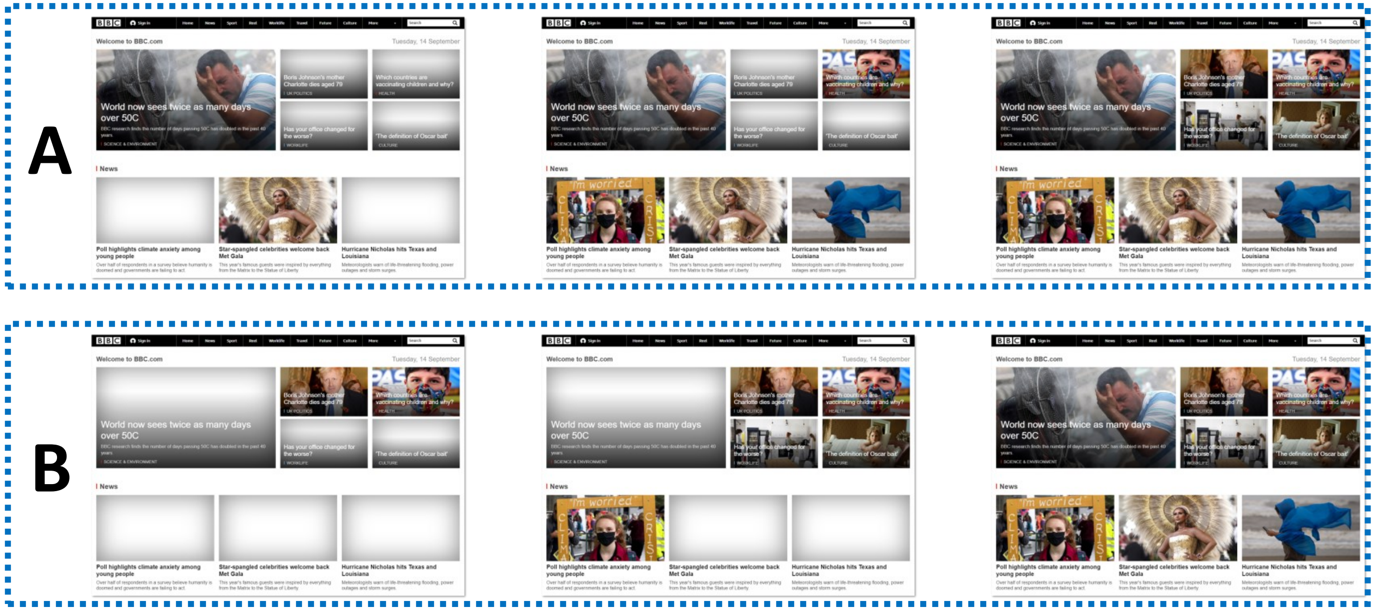
500 ms 1200 ms 3000 ms

1. Background

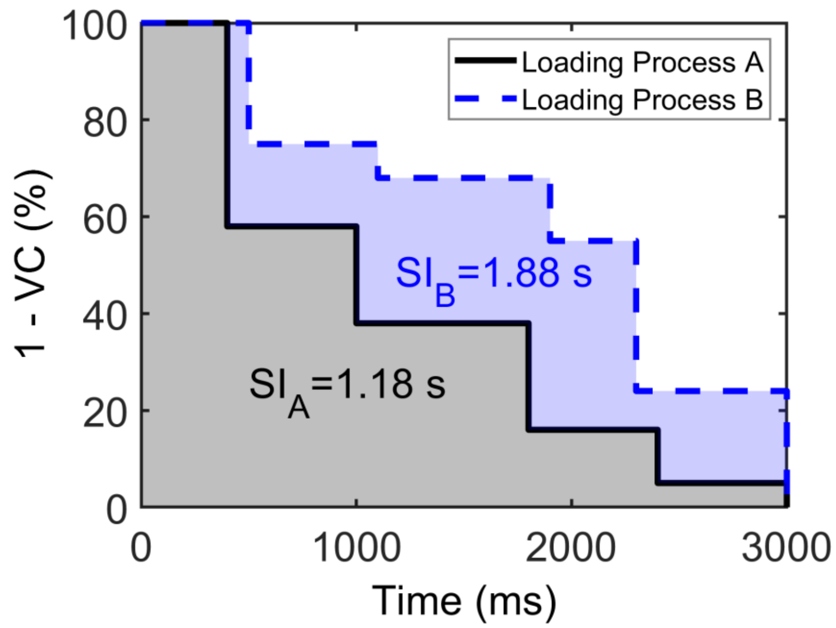
Formal definition of SI

$$SI = \int_0^{AFT} \left(1 - VC(t)\right) dt$$

AFT: Above-the-Fold Time
VC(t): **V**isual **C**ompleteness of the page's above-the-fold section at time ***t***



500 ms   1200 ms   3000 ms



2. Motivation

SI is being used retrospectively after page loading

Acting as a ***passive*** performance metric due to:

1. Integral calculation
2. Requiring the final rendered frame

"Fusing" SI into page loading

Proactively taking SI as an explicit heuristic to guide page loading ***in situ***.

In this way, we might be able to effectively improve SI of page loads.

2. Motivation

Measurement study

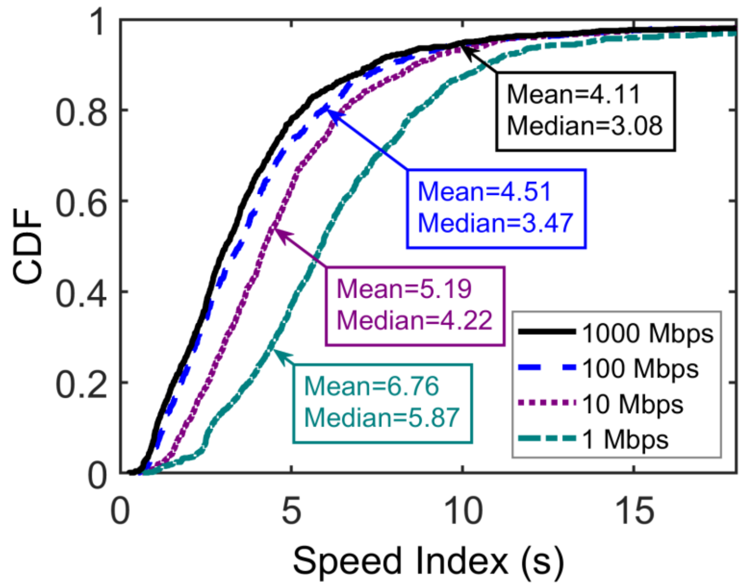
- Landing pages of the Alexa top 1,000 sites
- 3 PCs, 2 mobile phones
- Collecting network/rendering traces, snapshots, etc.

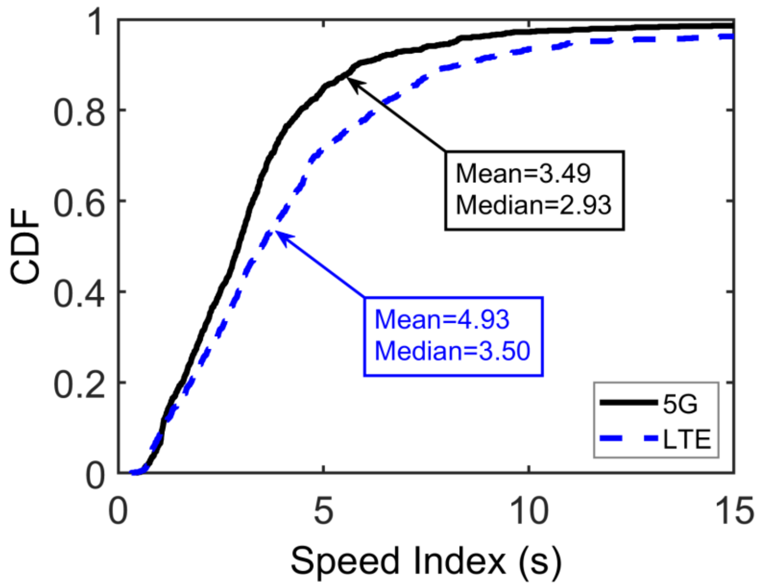
Device	CPU	RAM	Network	Viewport	OS
PC-1	Intel i7-10700F (2.90 GHz)	64 GB	Residential broadband	2560×1440	Windows 10
PC-2	Intel i7-10700F (2.90 GHz)	64 GB	Residential broadband	1920×1080	Windows 10
PC-3	Intel E5-2420 (1.90 GHz)	32 GB	Residential broadband	1920×1080	Windows 10
Xiaomi XM11	Snapdragon 888 (2.84 GHz)	12 GB	LTE/5G	3200×1440	Android 11
Huawei HV30	Kirin 990 (2.86 GHz)	6 GB	LTE/5G	2400×1080	Android 10

2. Measurement Findings

Network uncertainties

- Different network access methods
- Bandwidth/latency variation

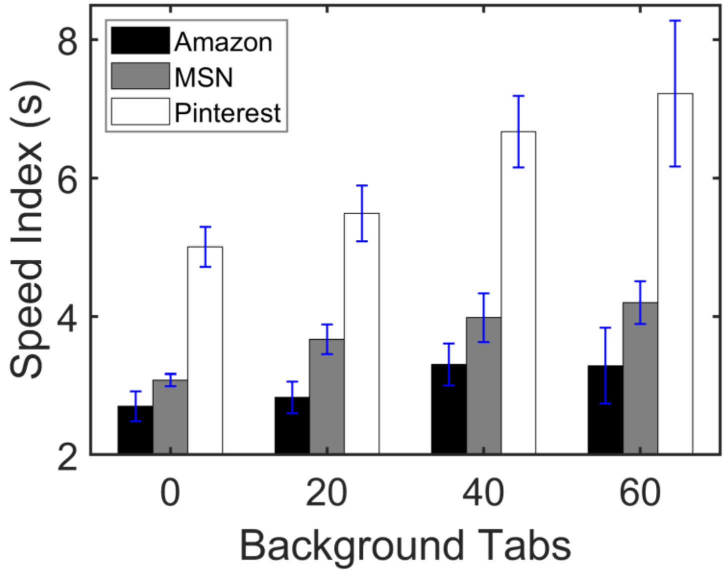




2. Measurement Findings

Browser execution uncertainties

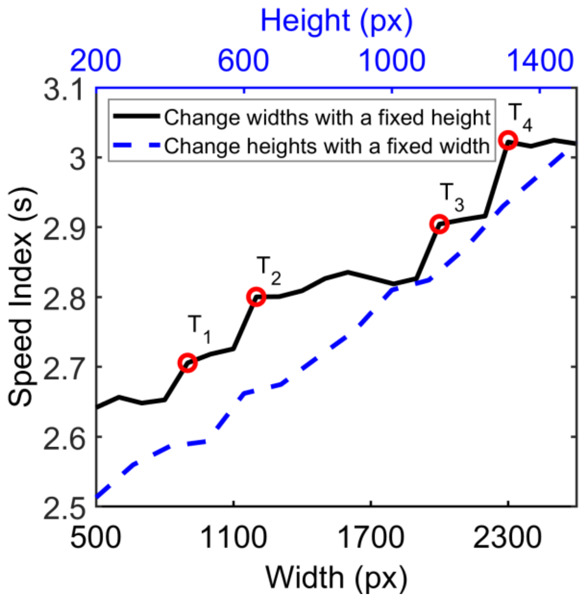
- Client resource contention
- Varied number of background tabs opened by users

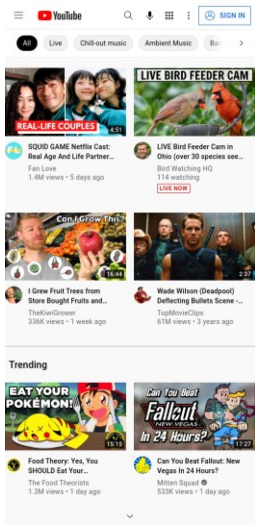


2. Measurement Findings

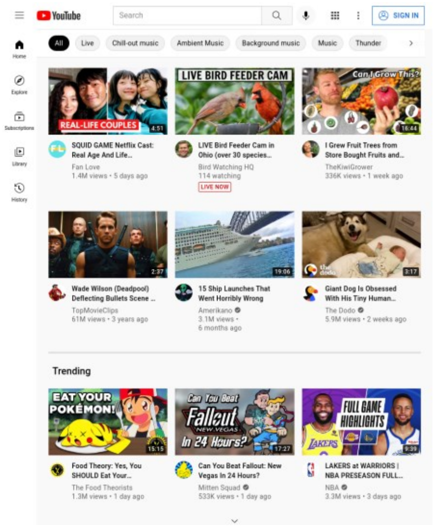
Viewport size uncertainties

- Diverse viewport sizes *v.s.* liquid layouts

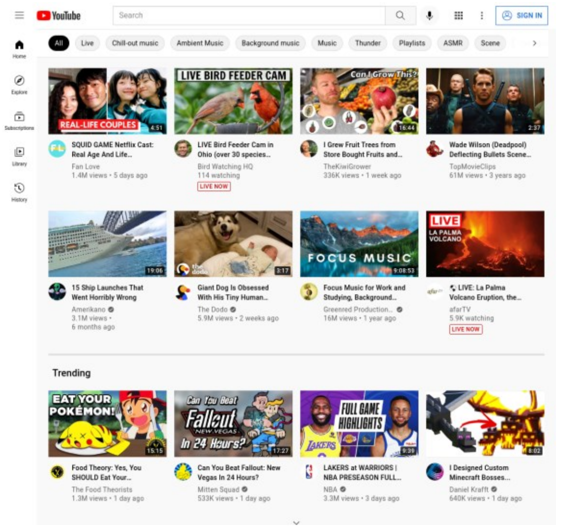




width=500 px



width=872 px



width=1128 px

Different layout schemes of YouTube

2. Motivation

Our Goal Obtaining SI-optimal scheduling

Obstacles

- Network Uncertainties
- Browser Execution Uncertainties
- Viewport Size Uncertainties

Dilemma SI-optimal scheduling cannot be achieved **in advance** or **in one shot**!

Question ***How to handle uncertainties of web page loading?***

3. SipLoader

Reactive scheduling

*"Reactive scheduling **does not try to cope with uncertainty** in creating the baseline schedule but **revises or re-optimizes** the baseline schedule **when an unexpected event occurs**."*

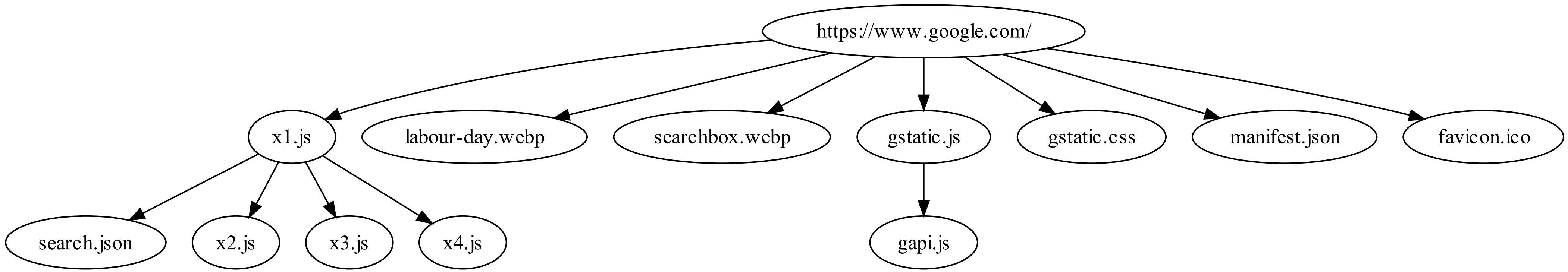
- Adapting to web page loading

1. Create a baseline scheduling (SI-optimal if no uncertainties)
2. Adjust to different viewport sizes (identify crucial elements)
3. Repair the baseline when uncertainties occur

3.1 Creating the Baseline

Dependency-merged greedy inference

- Page loading should obey dependencies
- Modeled as a dependency graph



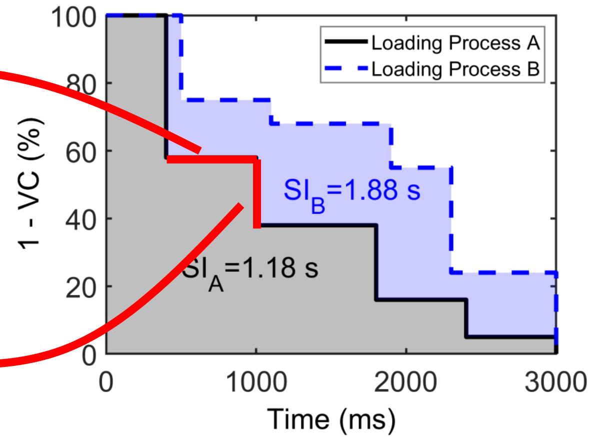
In which order should we load objects to achieve SI-optimal scheduling?

3.1 Creating the Baseline

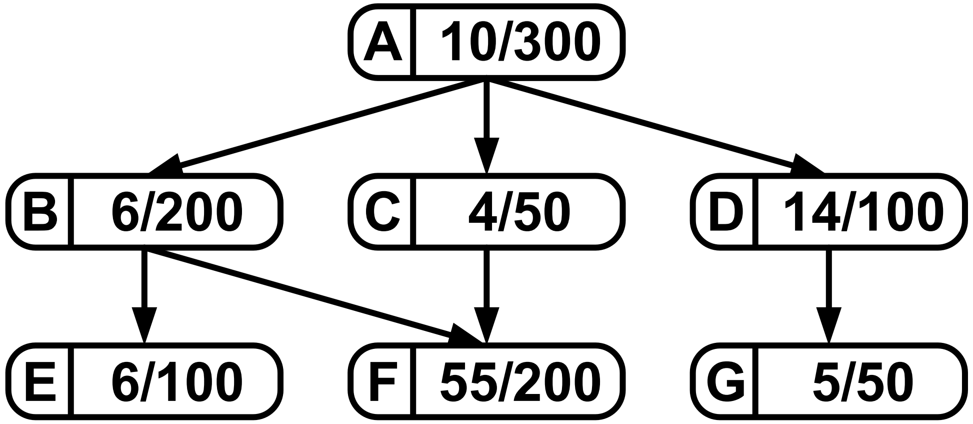
Dependency-merged greedy inference

Each web page object

1. Incurs time overhead to load (Loading cost)
2. Contributes to the visual completeness (SI gain)



SI gain (%) / Loading cost (ms)



A, B, C, D, E, F, G
A, C, B, D, E, F, G
A, D, B, C, E, F, G
…

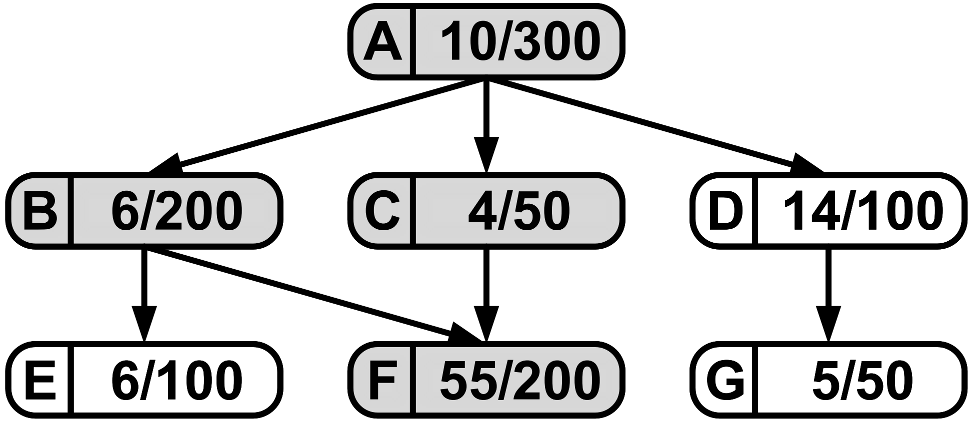
Topological sort-based approach is too slow – O(n!)

3.1 Creating the Baseline

Dependency-merged greedy inference

- Near-optimal solution
- Heuristic: the cumulative nature of SI calculation

$$SI = \int_0^{AFT} \big(1 - VC(t)\big)dt \qquad SI\ efficiency = \frac{gain}{cost} \rightarrow SI\ efficiency = \sum_{i \in group} \frac{gain(i)}{cost(i)}$$



Object group of F

3.2 Identifying Crucial Elements

☐ Predictive element region forest

How to adjust to different viewport sizes efficiently?

Liquid layout ⟶ Crucial elements are uncertain until the page is loaded!

Preload the page ***Server***	**Elements' possible positions** ⟶	Query crucial elements ***Client***

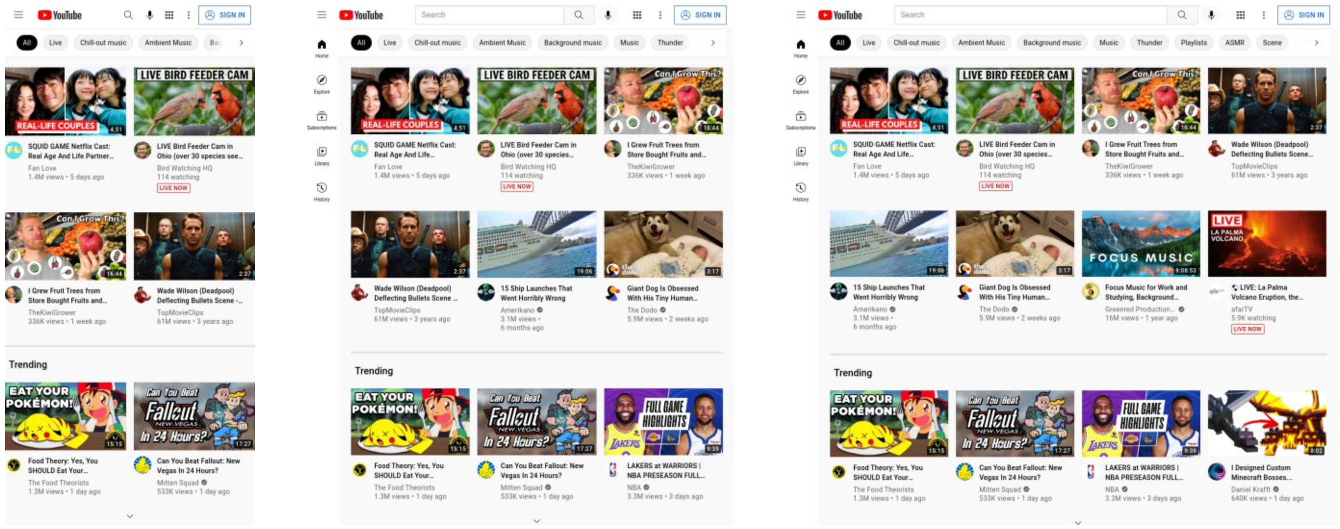
Server-client collaboration

3.2 Identifying Crucial Elements

Predictive element region forest

- Determine elements' possible positions:

  Layout scheme might change when the **viewport width** changes



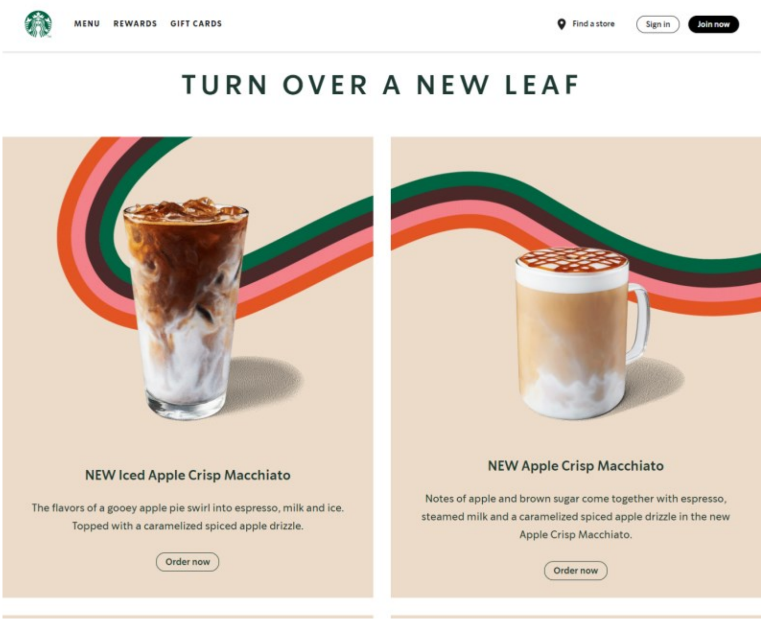
width=500 px　　width=872 px　　width=1128 px

Different layout schemes of YouTube

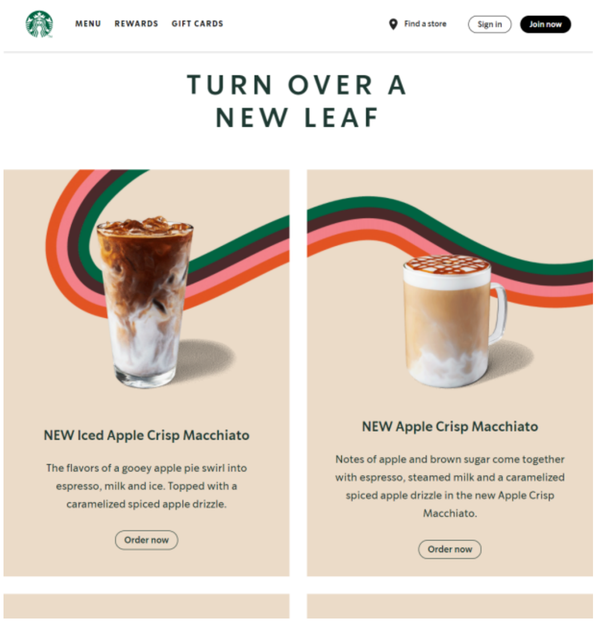
3.2 Identifying Crucial Elements

Predictive element region forest

- Identify layout schemes based on relative angles **(server-side)**



width=1400 px

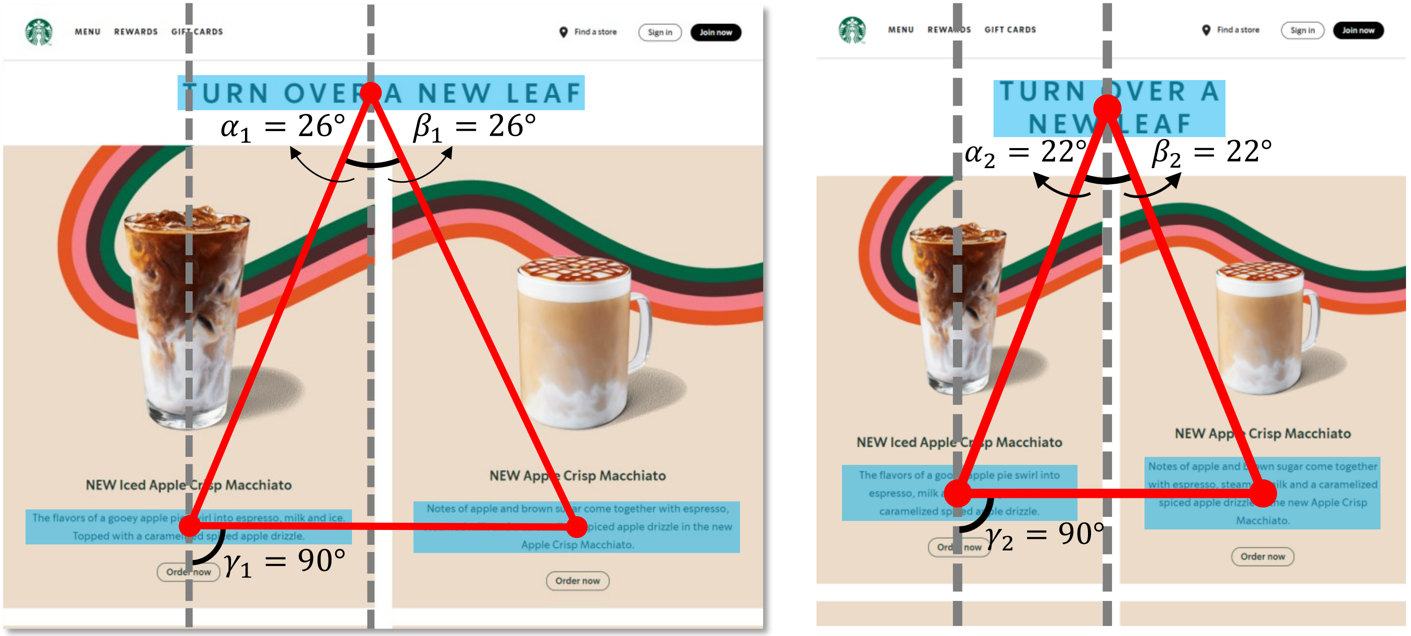


width=1080 px

3.2 Identifying Crucial Elements

Predictive element region forest

- Identify layout schemes based on relative angles **(server-side)**

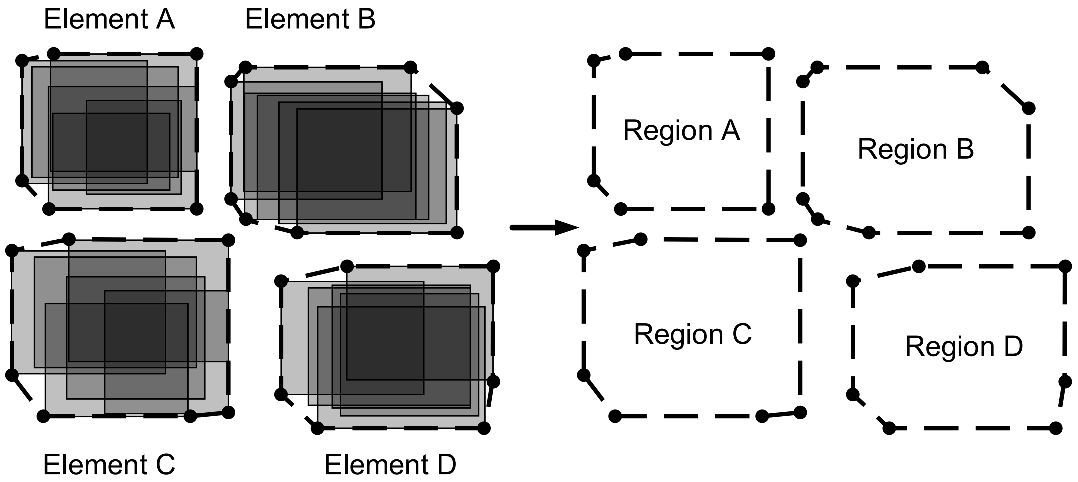


Relative angles change slightly under the same layout scheme

3.2 Identifying Crucial Elements

Predictive element region forest

- Element coverage regions (possible positions) ***in each layout scheme***

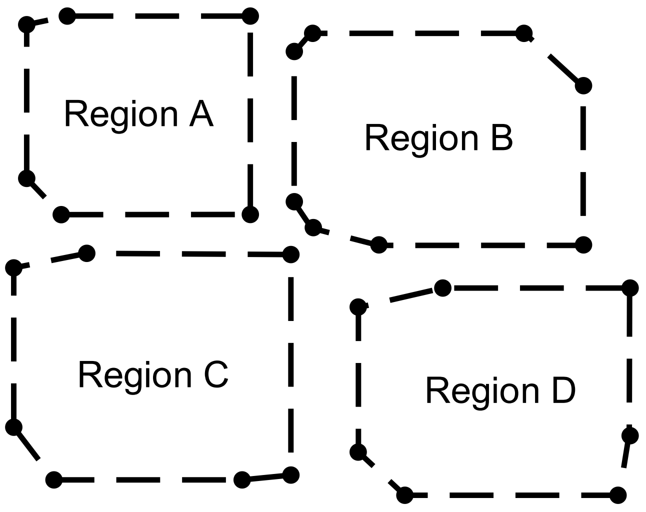


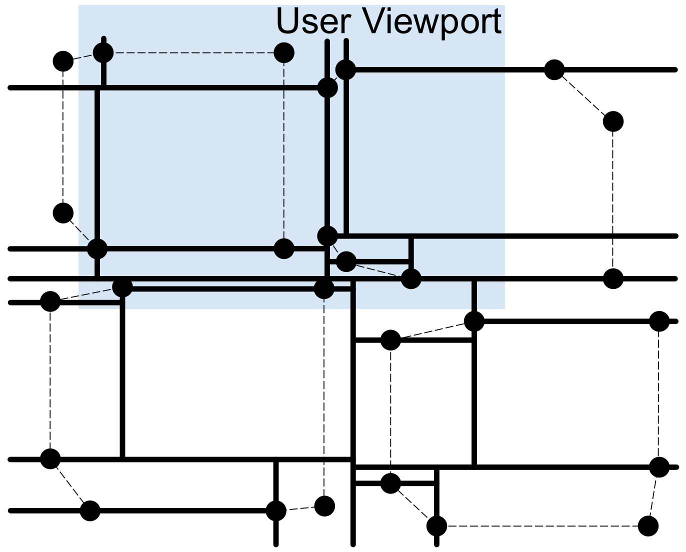
*Record element coverage regions when the viewport **width** changes*
(server-side)

3.2 Identifying Crucial Elements

Predictive element region forest

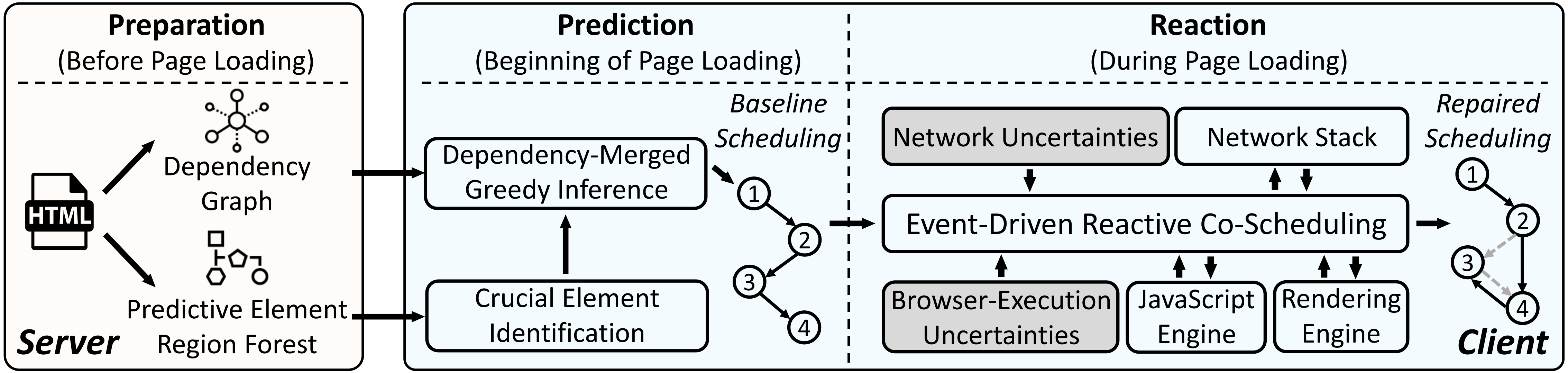
- Build regions' convex hulls into k-d trees (for each layout scheme)
- **The client** selects and efficiently queries a k-d tree based on viewport sizes to identify crucial elements





3. SipLoader

▢ The **cumulative** reactive scheduling framework



Creating the baseline schedule

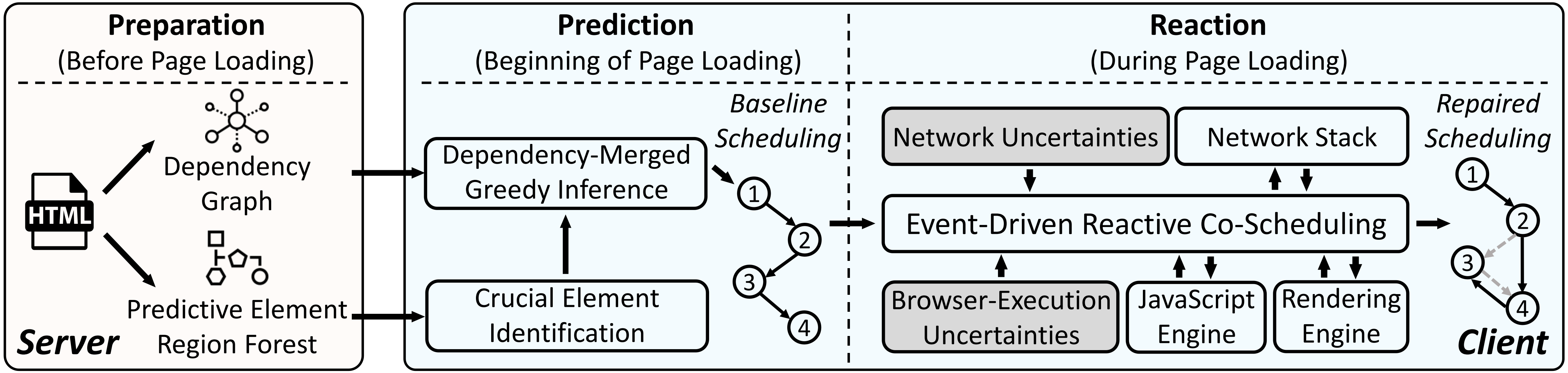
Leverage the cumulative nature of SI calculation

Online repairing

React to the occurrence of uncertainties

3. SipLoader

The cumulative reactive scheduling framework



Creating the baseline schedule

Leverage the cumulative nature of SI calculation

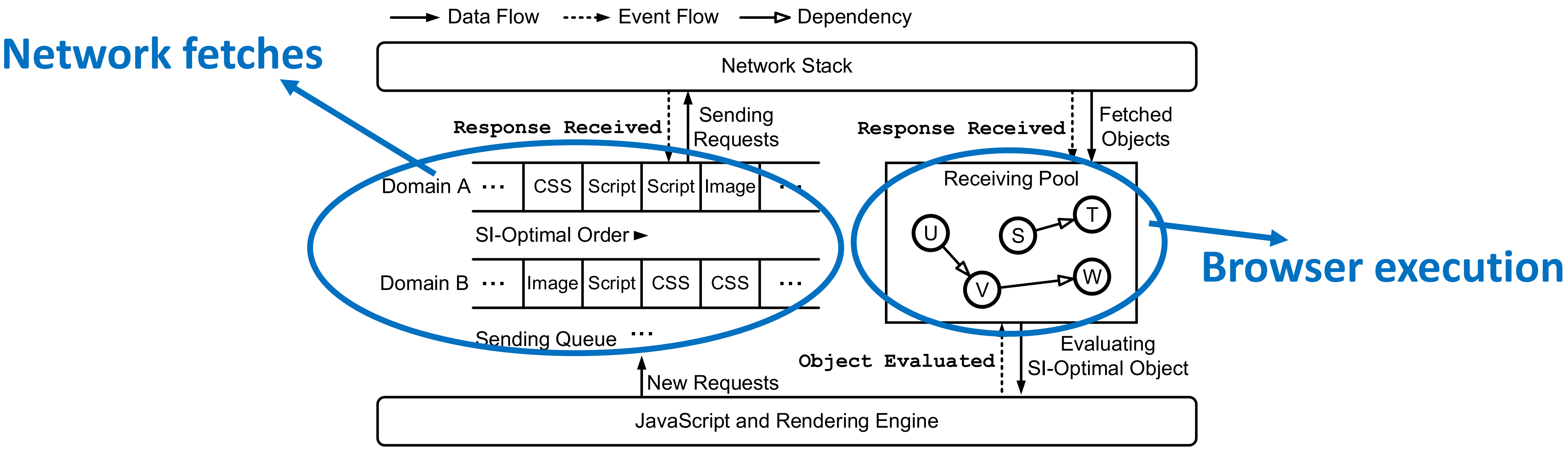
Online repairing

React to the occurrence of uncertainties

3.3 Repairing the Baseline

Event-driven reactive co-scheduling

- Repair the baseline scheduling
- React to network/browser execution uncertainties
- In an event-driven manner



4. Evaluation

Comparing with state-of-the-arts

- **Vroom** [SIGCOMM'17]: Server-aided dependency resolution
- **Fawkes** [NSDI'20]: Static template caching

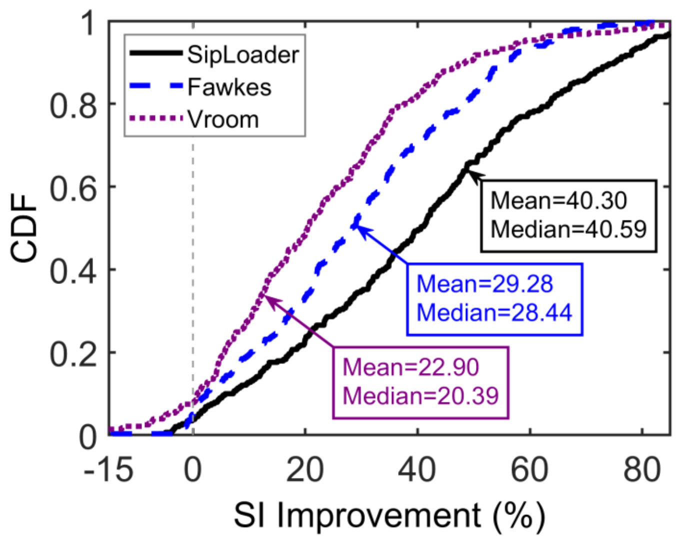
Testbed

- Landing pages of random 300 sites in the Alexa top 1,000 list
- **Network**: {1, 10, 100} Mbps, {10, 25, 50, 75} ms latency
- **Browser**: Cold cache, warm cache
- **Device**: PC, mobile phone

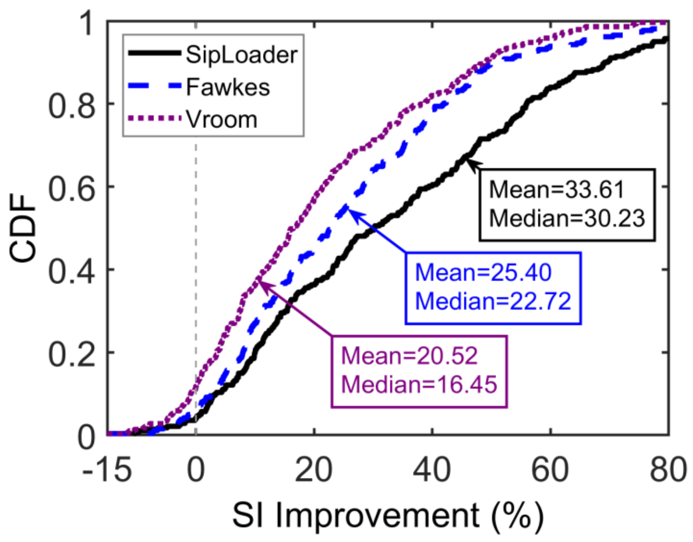
4. Evaluation

Major results

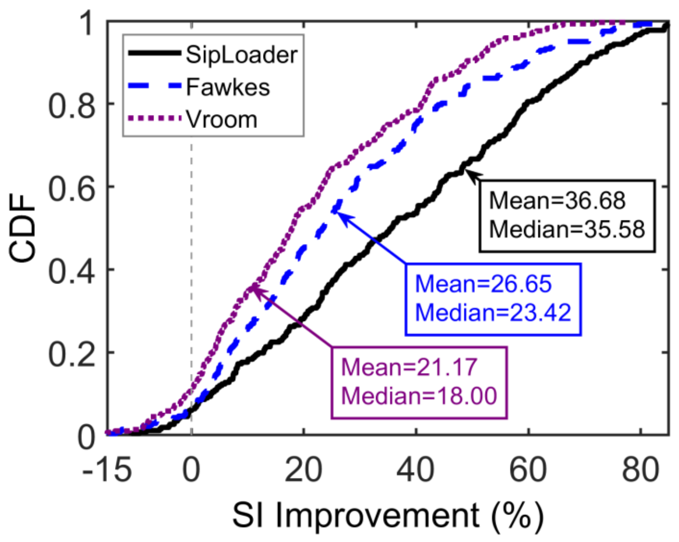
- Cold cache: Improve SI by more than 30%



10 Mbps, **25** ms latency, **2.4** GHz



100 Mbps, **25** ms latency, **2.4** GHz



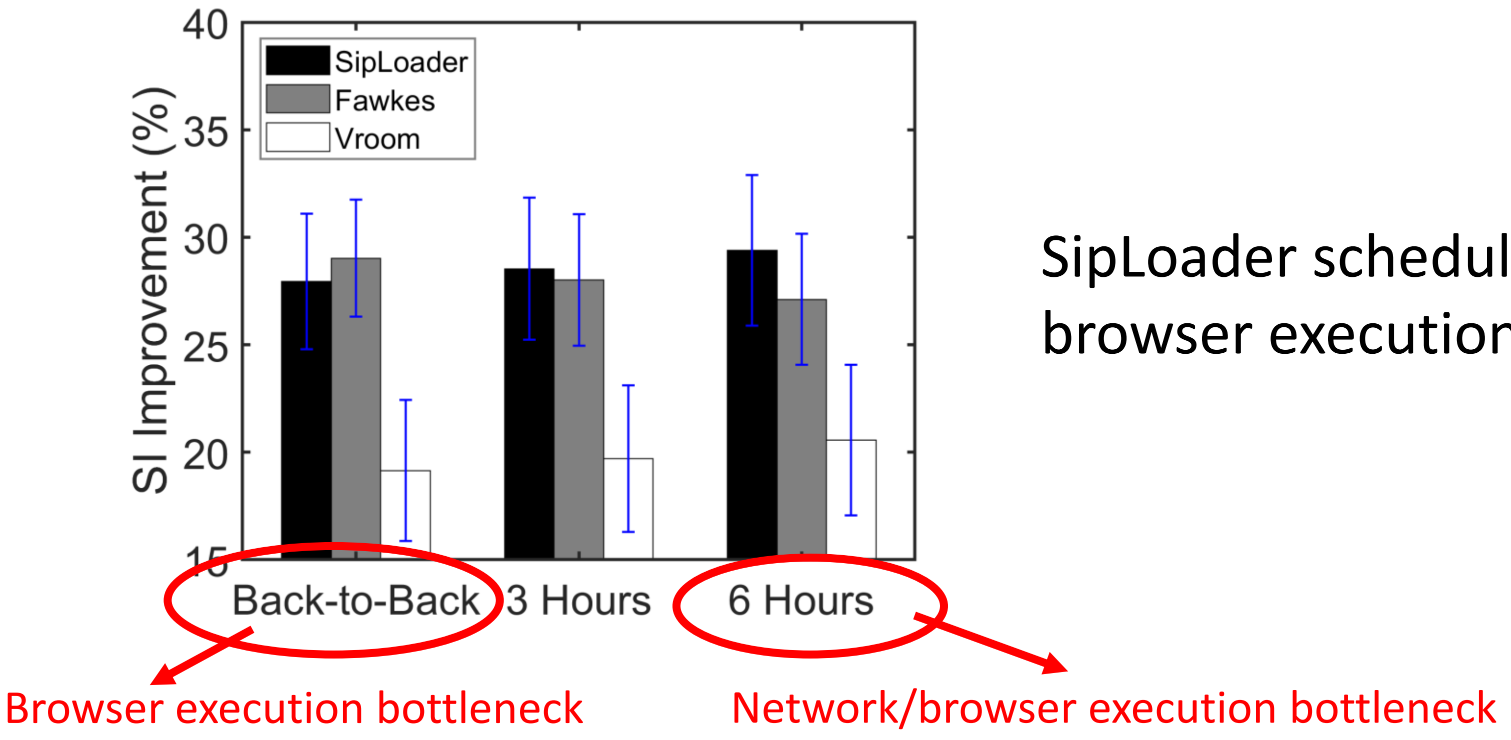
100 Mbps, **25** ms latency, **1.9** GHz

It is important to schedule object loading when network/computation resources are limited!

4. Evaluation

Major results

- Warm cache

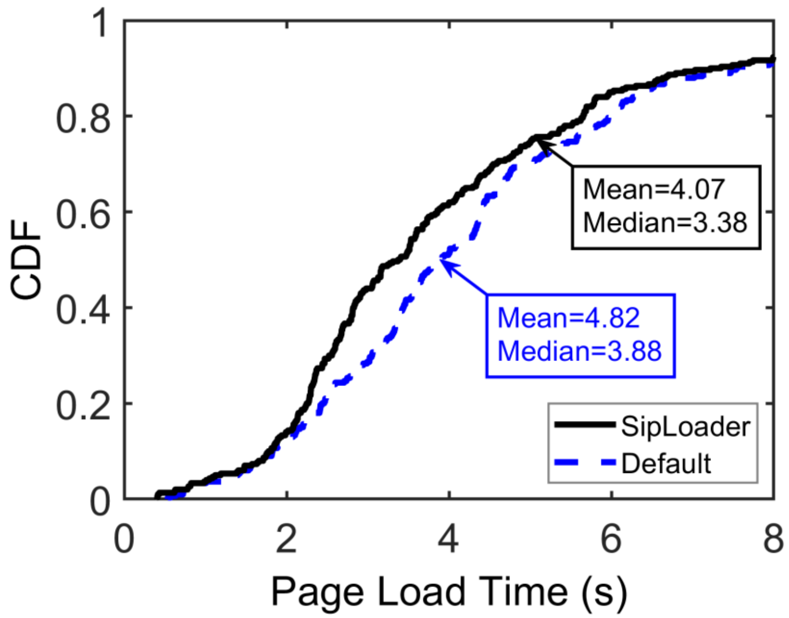


SipLoader schedules both network fetches and browser execution to achieve the (near-)optimal SI

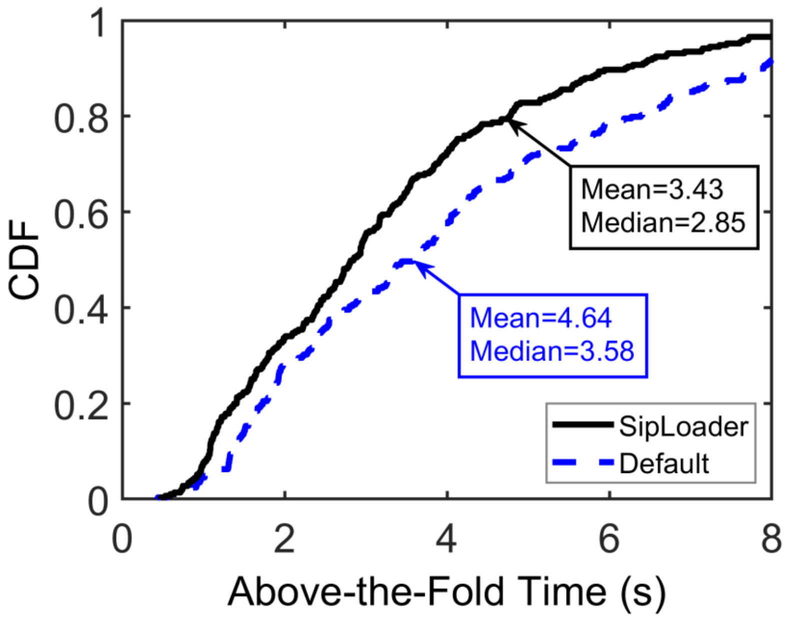
4. Evaluation

◻ Beyond SI

- SipLoader also improves other metrics



Dependency graph is generated in advance



Crucial elements have higher priorities

5. Conclusion

- We uncover the key challenge of using advanced web page performance metrics (such as Speed Index) to guide page loading – the uncertainties during page loading make it impossible to obtain the optimal scheduling in advance or in one shot.

- We present SipLoader, an SI-oriented page load scheduler that leverages the **cumulative reactive scheduling** framework. It does not deal with uncertainties in advance, but repairs the baseline scheduling when uncertainties actually occur. SipLoader improves the average SI by 33.6%.

- Source code and data are available at https://siploader.github.io/

Thanks!

Q & A