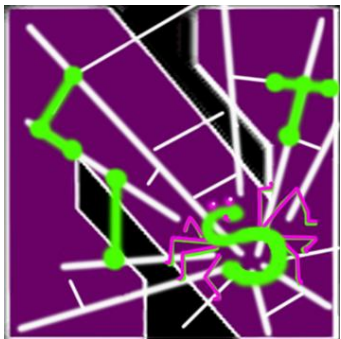


WebProphet: Automating Performance Prediction for Web Services

Zhichun Li, Ming Zhang, **Zhaosheng Zhu**, Yan Chen,
Albert Greenberg and Yi-min Wang
Lab of Internet and Security Technology (LIST)
Northwestern University
Microsoft Research

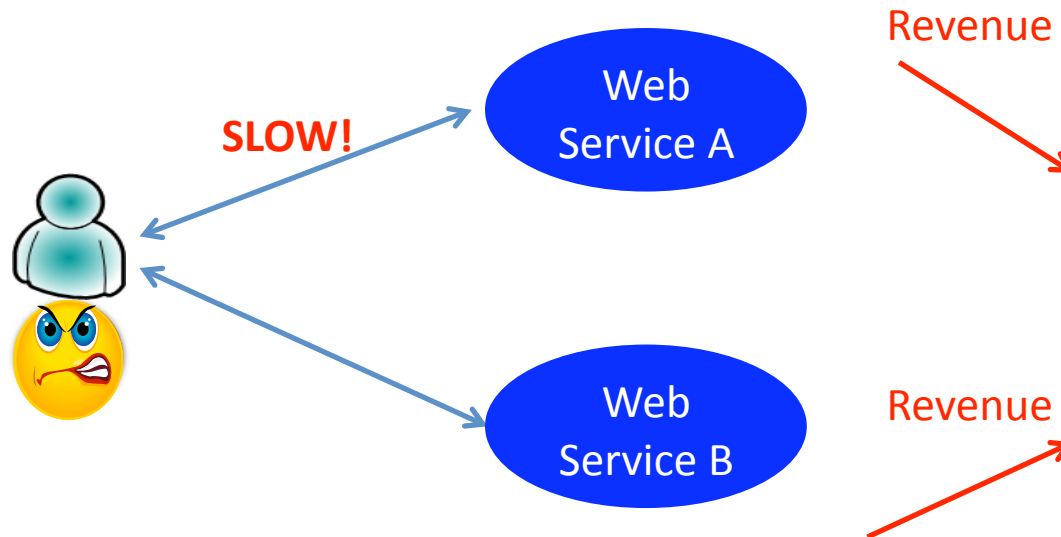


Microsoft®
Research

Web Services Are Prevalent

- Almost everything is related to Web
 - Web search
 - Web mail
 - Online shopping
 - Online Social network
 - Calendar

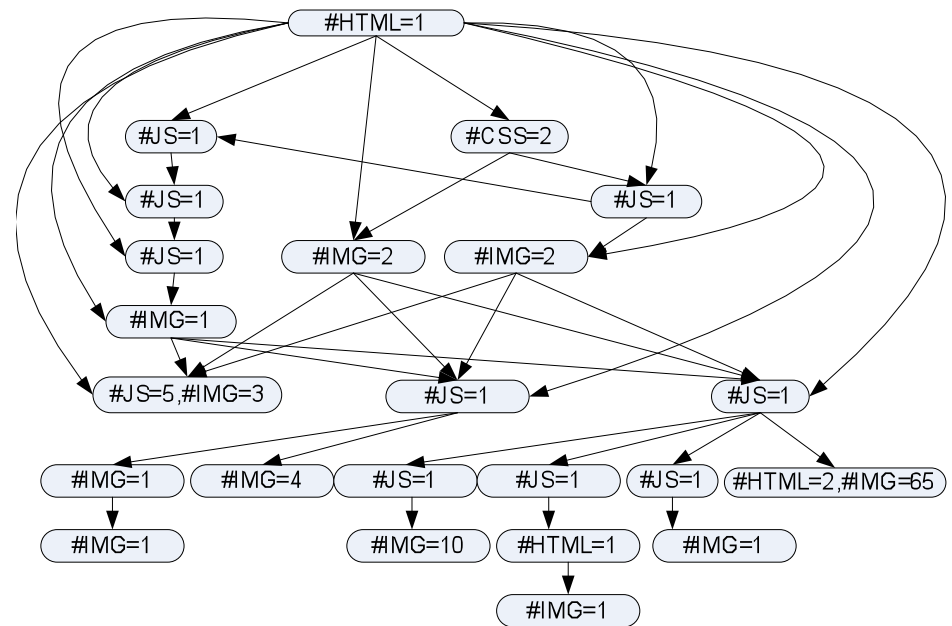
Performance Is Important



- Amazon: 100ms extra delay → 1% sale loss
- Google search results: 500 ms extra delay → reduce display ads revenues by up to 20%

Web Services Are Complicated

- Example of Yahoo Maps
 - 110 embedded objects
 - Complex object dependencies
 - 670KB JavaScript
 - Hosted by multiple data-centers around the world



Performance Optimization is Hard

Page Load Time

User perceived PLT:
whole page or the portion
with most visual effects

Object

A large number of possible
optimization strategies

Limitations with Existing Techniques

- A/B test (controlled experiments)
 - Idea: set up an experiment setting and try on a group of users
 - Problems with A/B test
 - Hard to fully automated
 - Expensive to set up
 - Quite slow!

Limitations with Existing Techniques

- Service provider based techniques (WISE SIGCOMM2008)
 - Problems
 - multiple data sources
 - Object dependencies
 - Client side delays, e.g. JavaScript execution time
- Regression based techniques (LinkGradient INFOCOM2009)
 - Usually require the independence assumption on delay factors of each object. **Problematic!**

Our Contributions

- A tool for automated performance prediction



- Fast prediction on the user perceived performance
- Timing perturbation based dependency discovery
- Dependency driven page load simulation

Outline

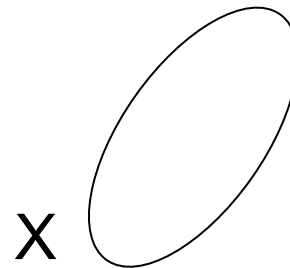
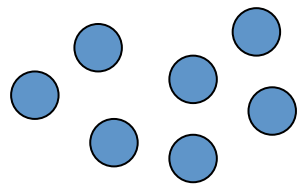
- Motivation & Design
- **Dependency Extraction**
- Performance Prediction
- Implementation
- Evaluation
- Conclusion

Why Are Dependency Discovery Difficult?

- Simple HTML parsing/DOM traversal is not enough
 - Object requests generated by JavaScript depend on the corresponding .JS files
 - Event triggers, such as when image B trigger “onload” event, then image A will be load by JavaScript
- Extensive browser instrumentation is heavy-weight and browser dependent

Our Approach

- Goal:
 - Light-weight black box based approach
 - Browser independent
- Timing perturbation based technique
 - Inject delay
 - See how delay propagate.



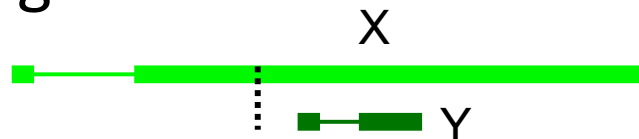
Objects depend
on X

Take Care HTML Objects

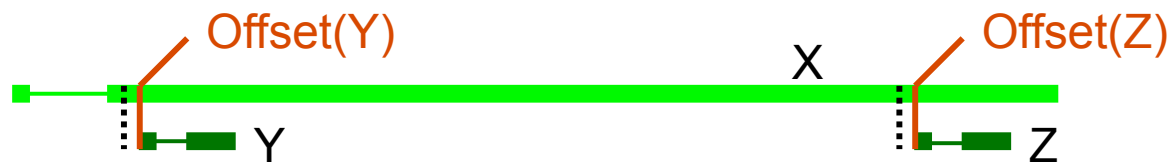
- Regular Objects
 - Regular objects have to be fully loaded before their descendants



- HTML Objects are special
 - HTML is stream objects, allowing incremental rendering



Measure the Offset

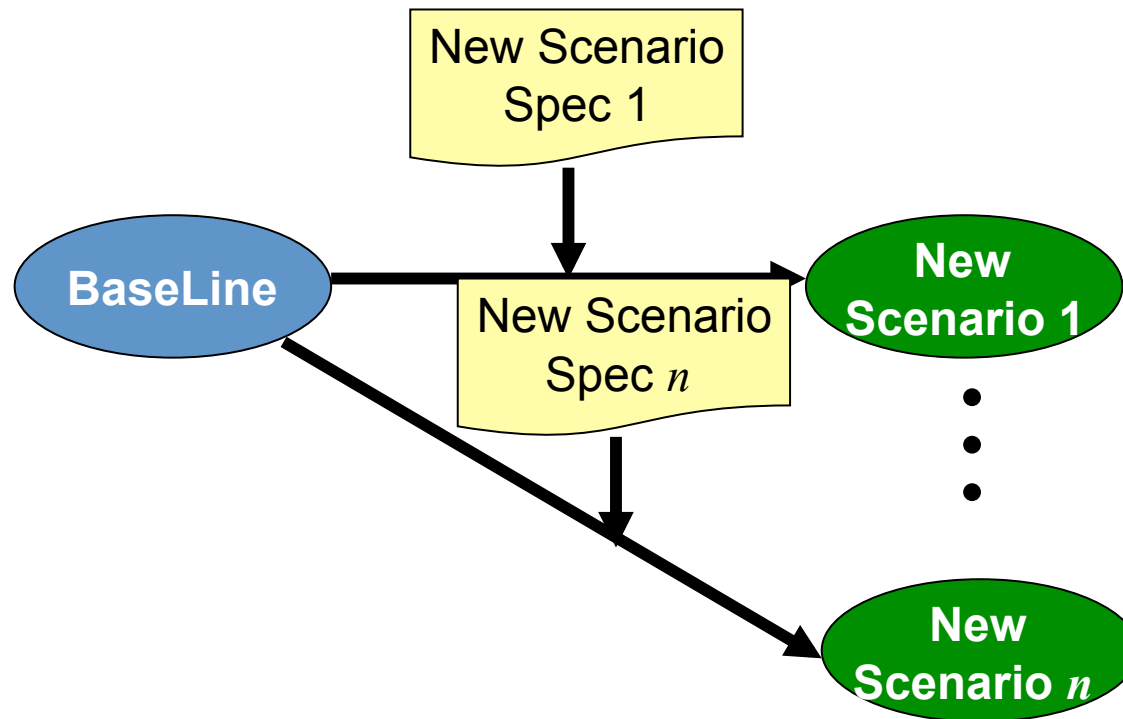


Outline

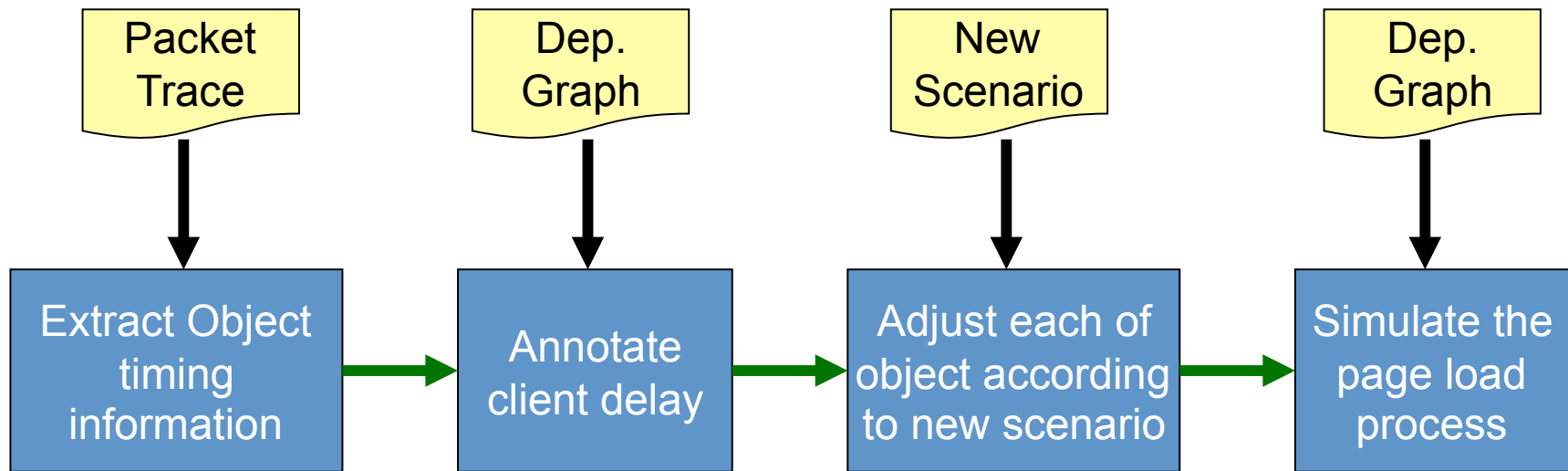
- Motivation
- Design
- Dependency Extraction
- **Performance Prediction**
- Implementation
- Evaluation
- Conclusion

Performance Prediction Problem

- Evaluate different new scenarios



Performance Prediction Procedure




Extract Object Timing information

- Extract Timing from packet traces
- Basic object timing info

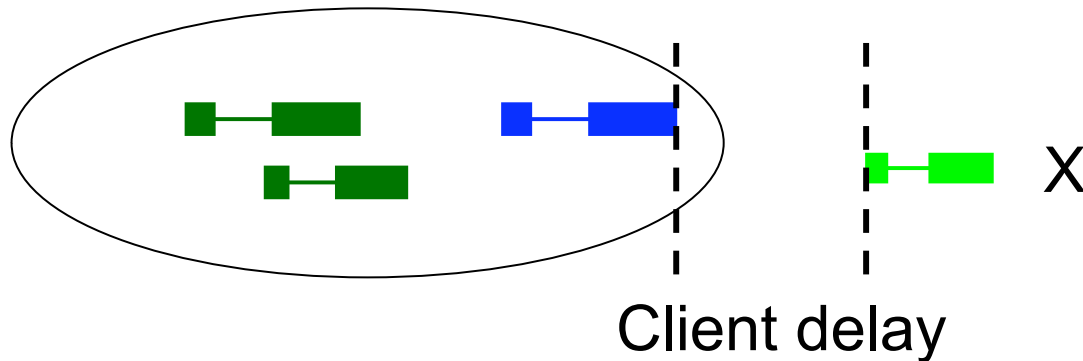
DNS  DNS lookup time

TCP  TCP handshaking time

HTTP  Response time
Request transfer time Reply transfer time

Annotate client delay

- Browser processing time after dependency solved

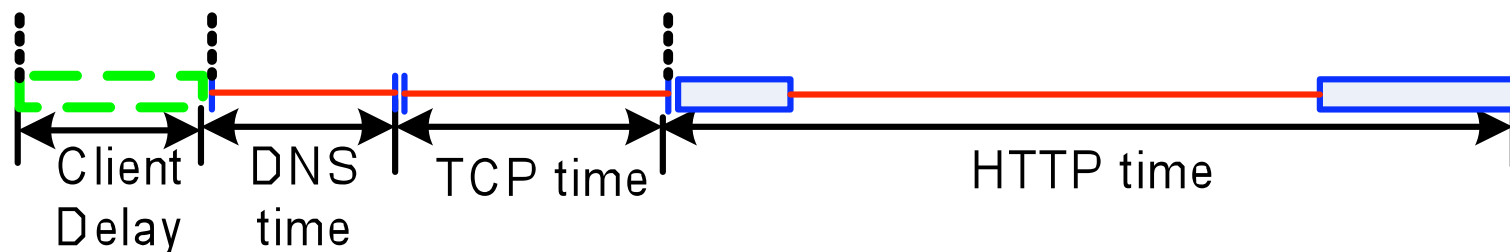


Adjust Object Timing Info

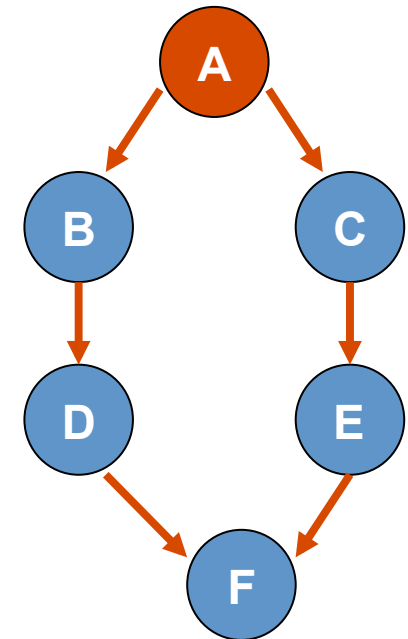
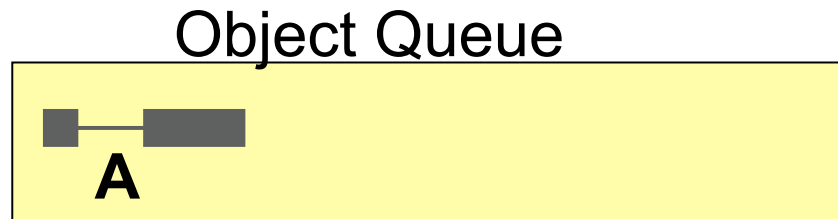
- Consider four delay factors: **client delay**, **server delay**, **RTT** and **DNS lookup time**
- Adjust timing
 - Adjust Client delay, DNS lookup time, and server response time directly
 - RTT: adjust $\Delta\text{RTT} * \text{number of round trips}$

Factors Affected Object Loading

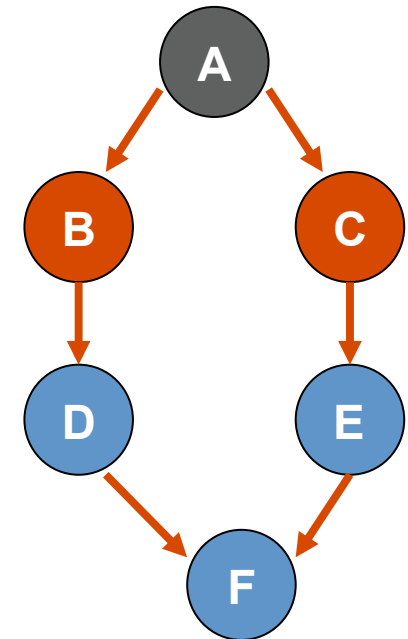
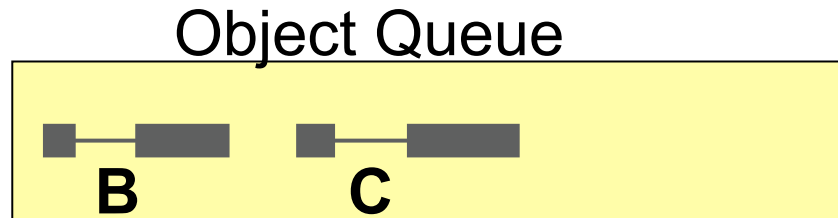
- Add DNS lookup time based on DNS cache
- Add TCP handshaking time for new connections
- Add TCP waiting time when all connections are not available



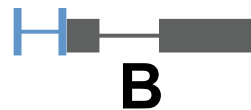
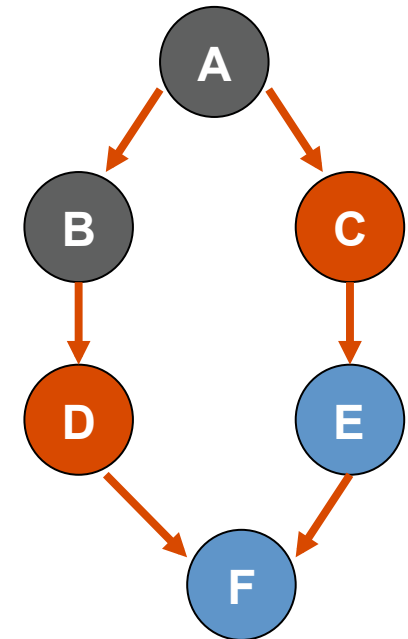
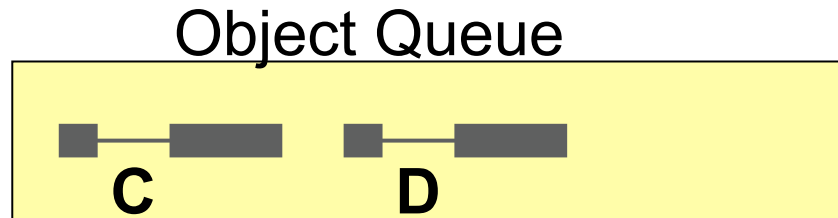
Simulate Page Load Process



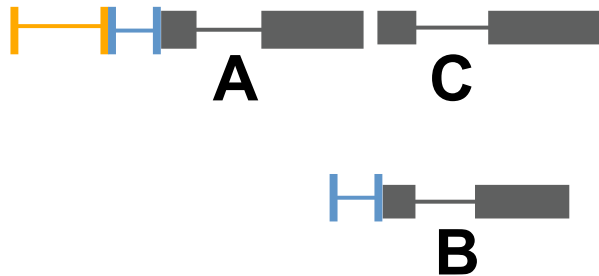
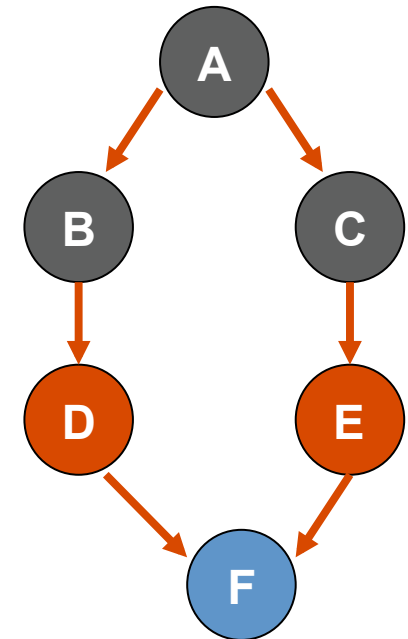
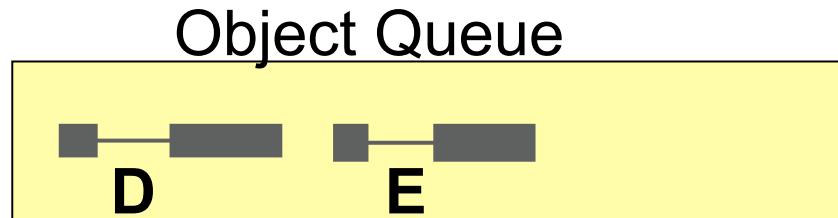
Simulate Page Load Process



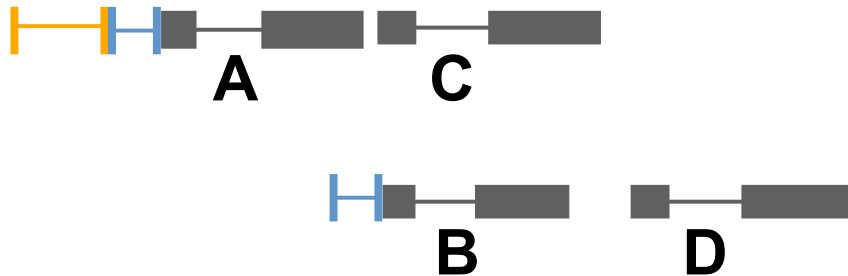
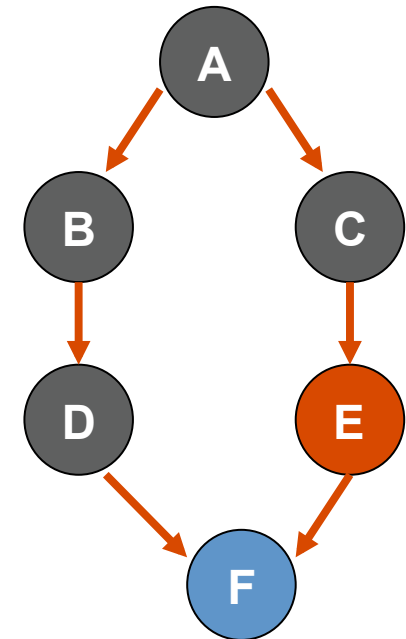
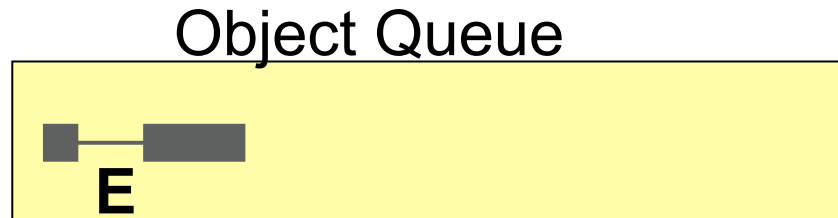
Simulate Page Load Process



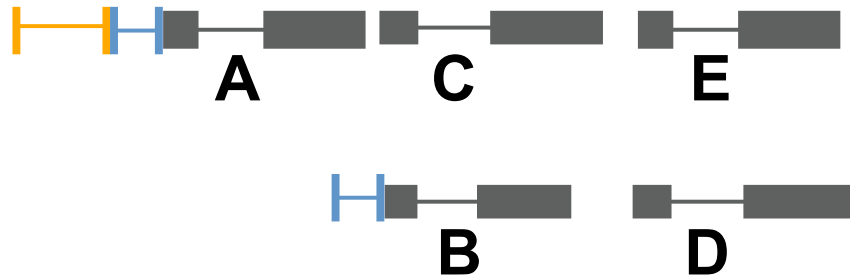
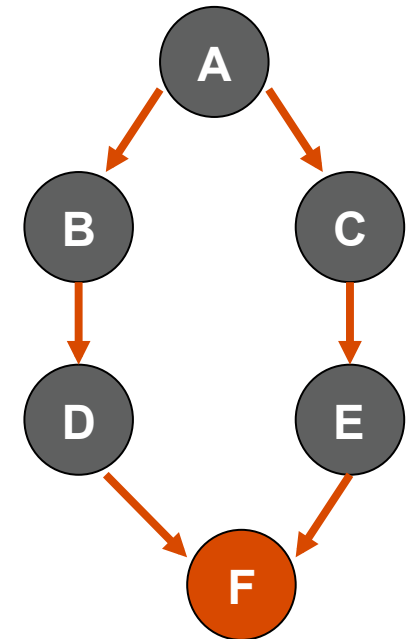
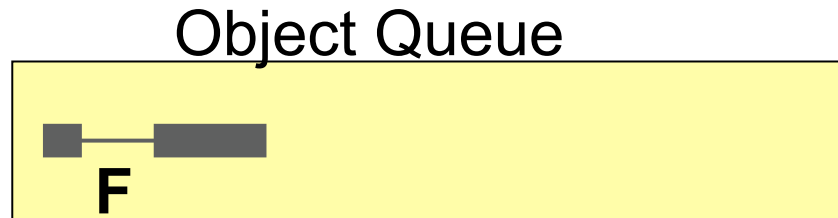
Simulate Page Load Process



Simulate Page Load Process

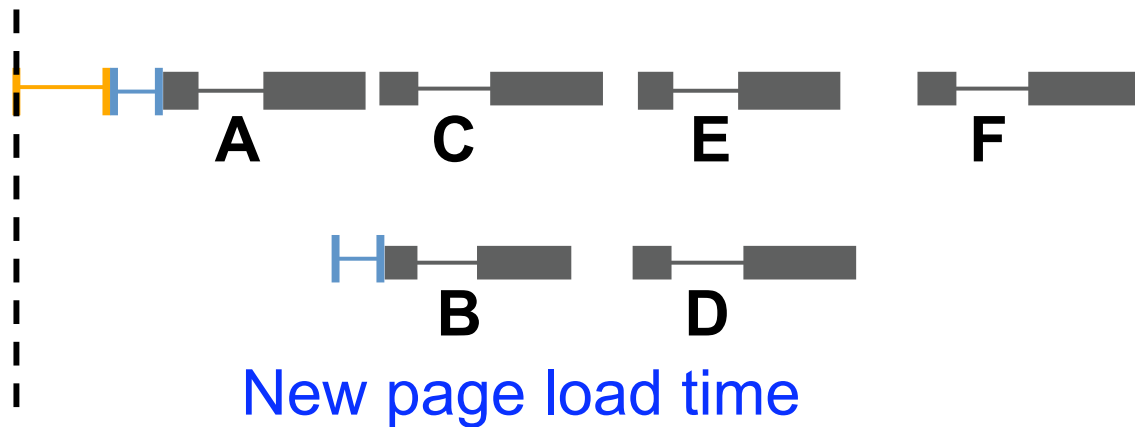
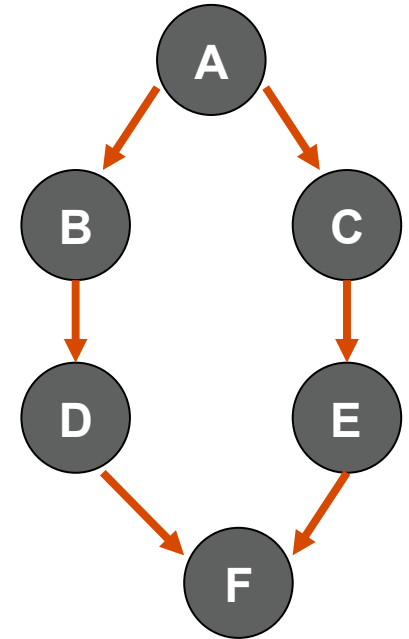
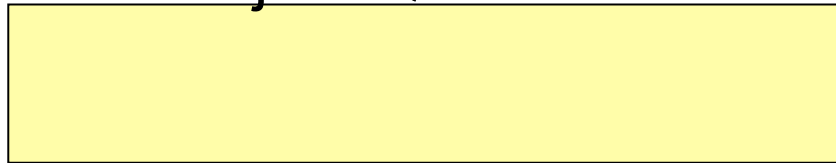


Simulate Page Load Process



Simulate Page Load Process

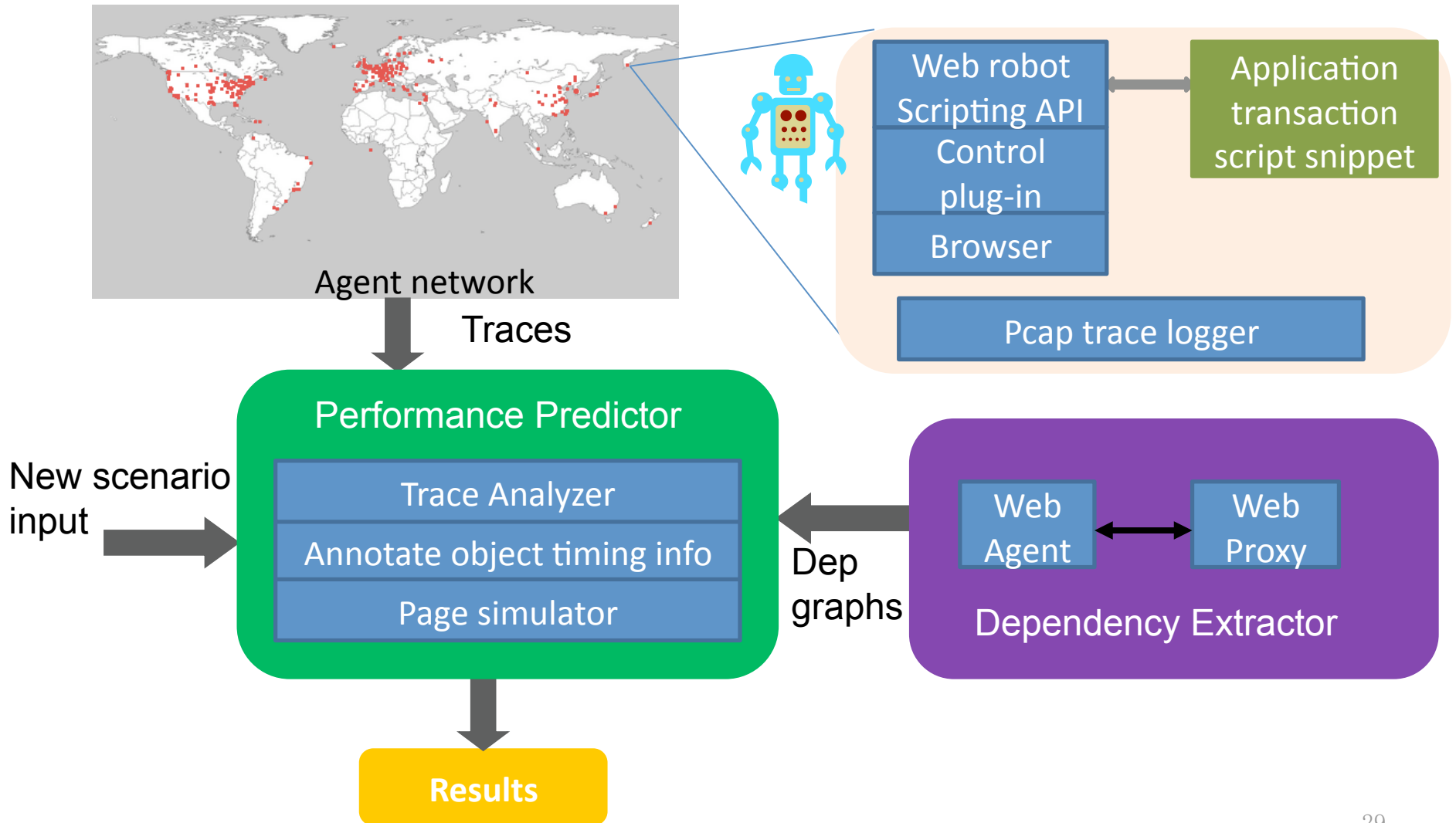
Object Queue



Outline

- Motivation
- Design
- Dependency Extraction
- Performance Prediction
- **Implementation**
- Evaluation
- Conclusion

WebProphet Framework

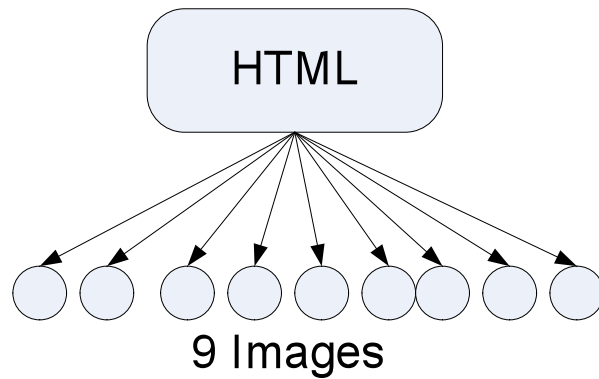


Outline

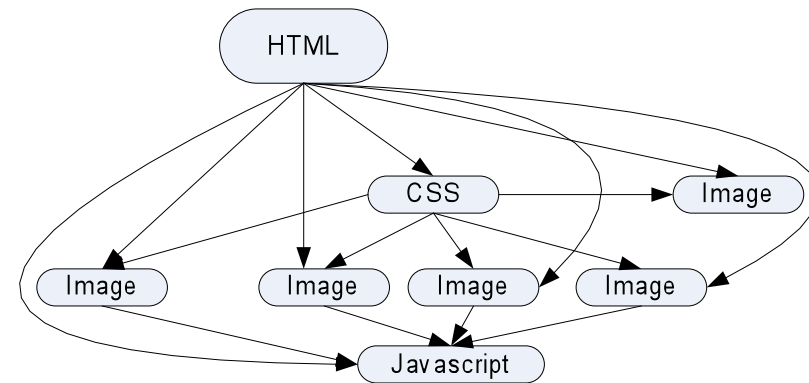
- Motivation
- Design
- Dependency Extraction
- Performance Prediction
- Implementation
- **Evaluation**
- Conclusion

Dependency Extraction Results

- Google and Yahoo Search



Google

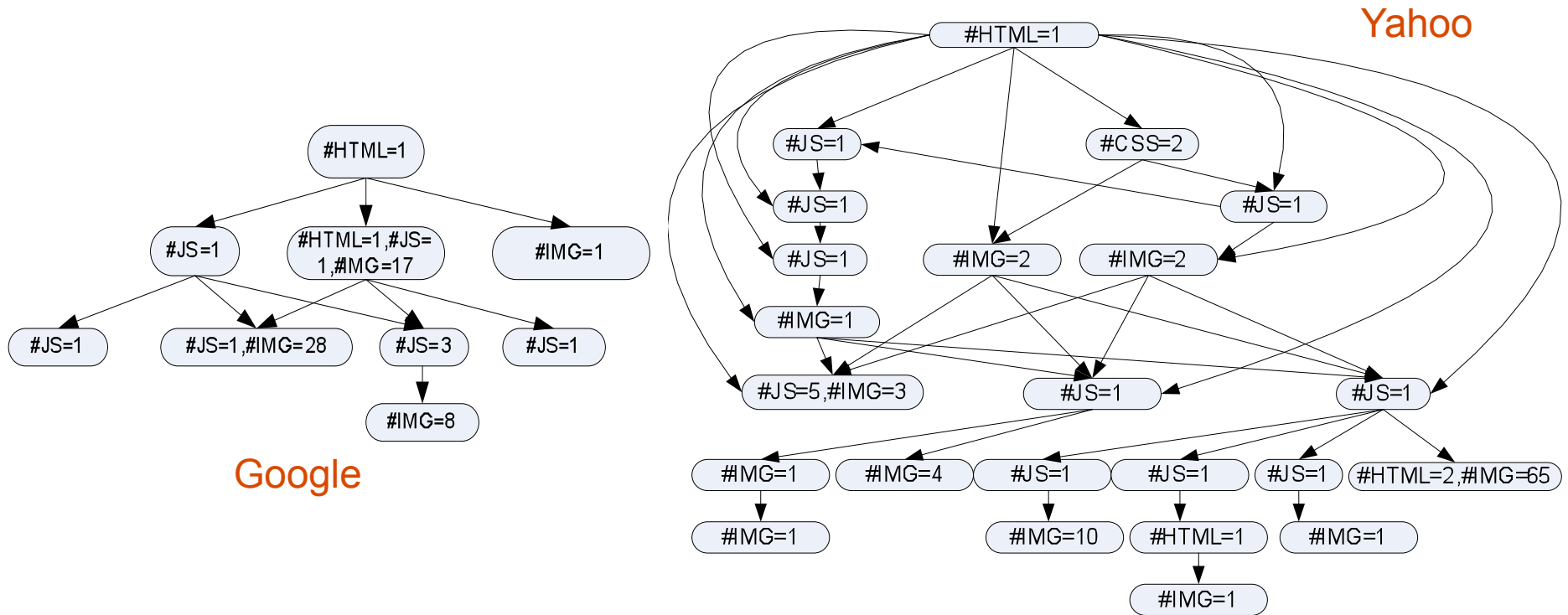


Yahoo

- Validation: manual code analysis

Dependency Extraction Results

- Google and Yahoo Maps



- Validation: create pages with the same dep. graph and validate the crafted pages

Prediction Experiment Setup

- Reduce latency see the improvement on PLT
- Controlled experiments
 - Baseline: high latency
 - New Scenario: low latency
 - Use control gateway to inject and remove delays
- Planetlab experiments
 - Baseline: International nodes
 - New scenario: US nodes
 - Improve all delay factors to be the same as the US node.

Controlled Experiment

- Setup: visit Yahoo Maps from Northwestern
- Baseline: inject 100ms RTT to one DC
- New Scenario: removing the 100ms RTT injected

DC	Err (median)	Err (P95)
Akamai	16.0%	11.8%
YDC1	6.5%	9.7%
YDC2	14.8%	6.0%

Planetlab Experiment

- Baseline: A International node with relative poor performance
- New Scenario: a US node

Service	Baseline	New	Err(median)	Err(P95)
Gsearch	Singapore	US	2.0%	10.7%
Ysearch	Japan	US	6.1%	0.3%
Gmap	Sweden	US	1.2%	1.8%
Ymap	Poland	US	0.7%	1.3%

Usage Scenarios

- Analyze how to improve **Yahoo Maps**
 - Only want to optimize a small number of objects
 - Use a greedy based search
 - Evaluate **2,176** hypothetical scenarios in **20** secs, find that
 - Move 5 objects to CDN: **14.8%**
 - Reduce client delays of 14 objects to half: **26.6%**
 - Combine both: **40.1%** (4secs to 2.4secs)

Outline

- Motivation
- Design
- Dependency Extraction
- Performance Prediction
- Implementation
- Evaluation
- Conclusion

Conclusions

- Web service performance prediction is hard
 - Modern web services are complicated
 - Object dependencies are very important
- Design an automated tool for performance prediction
 - Dependency discovery
 - Dependency driven performance predication
 - Evaluation on the accuracy and usefulness of our tool

Q & A

Thanks!