

# Horcrux:

## Automatic JavaScript Parallelism for Resource-Efficient Web Computations

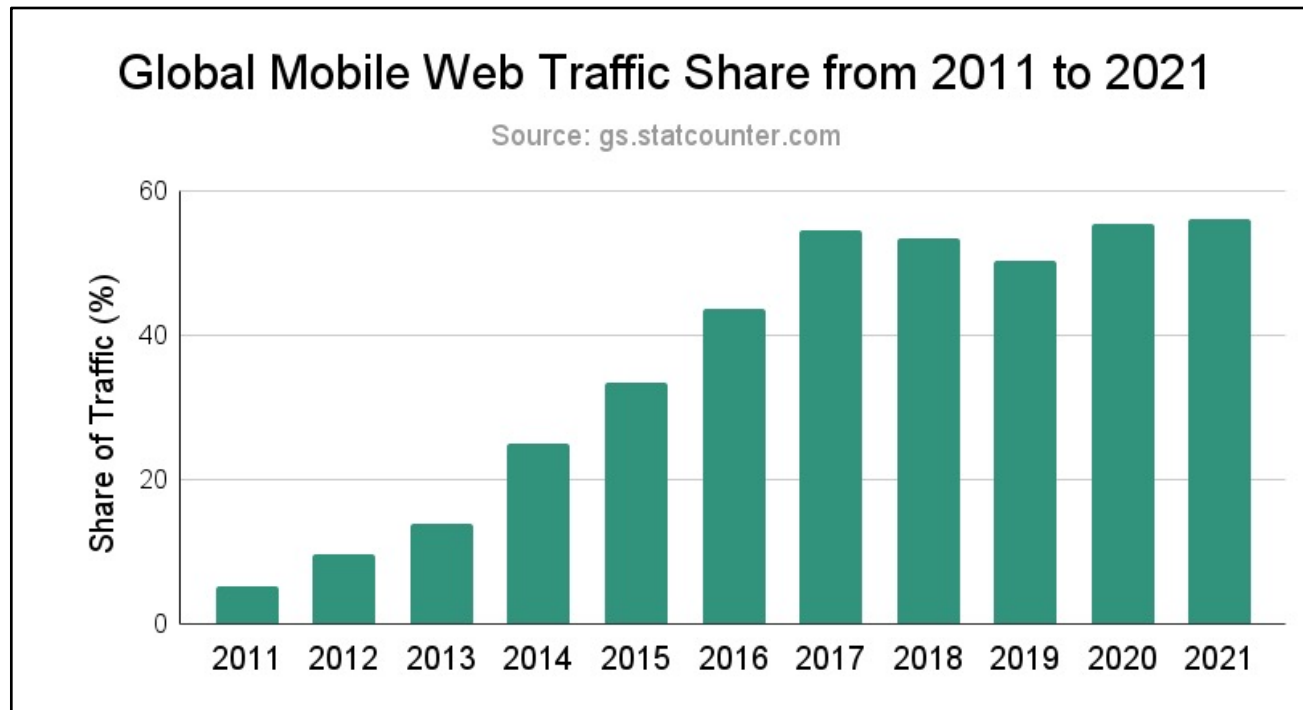
---

**Shaghayegh Mardani**<sup>1</sup>, Ayush Goel<sup>2</sup>, Ronny Ko<sup>3</sup>, Harsha Madhyastha<sup>2</sup>, Ravi Netravali<sup>4</sup>

<sup>1</sup>UCLA, <sup>2</sup>University of Michigan, <sup>3</sup>Harvard University, <sup>4</sup>Princeton University

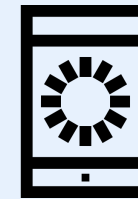
# Modern Web Browsing

## Web Traffic is Dominated by Mobile



## Performance Matters

Users



53% of visits are abandoned if a mobile site takes longer than 3 seconds to load.

Source: thinkwithgoogle.com

Content Providers



If your site makes \$100,000/day, 1 sec improvement in page load increases revenue by \$7,000 daily.

Source: bluecorona.com

# The Problem: Computation Delays

- Evaluation setup

Performance metrics:

- Page Load Time (PLT)
- Speed Index (SI)

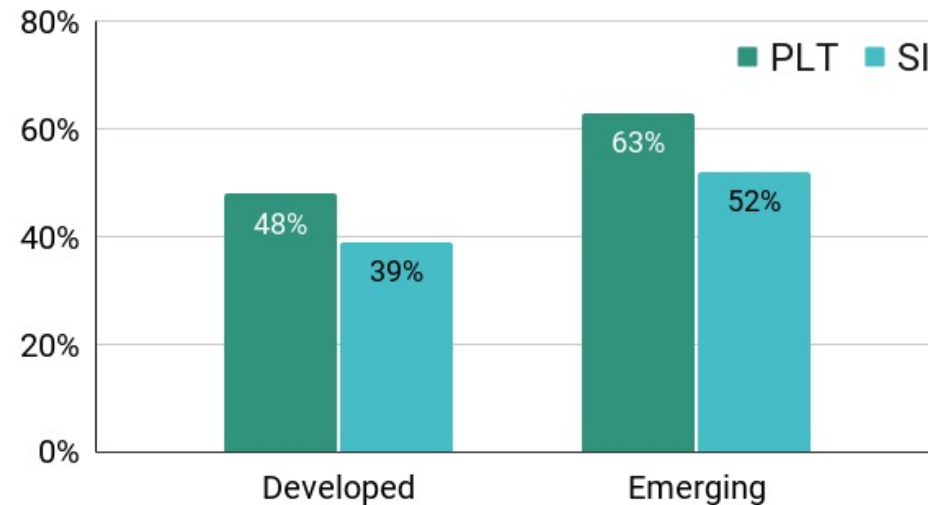
- With *NO* network delays:

## Developed Region

- 582 pages in US
- Google Pixel 3
  - 2.0 GHz octa-core

## Emerging Region

- 91 pages in Pakistan
- Redmi 6A
  - 2.0 GHz quad-core

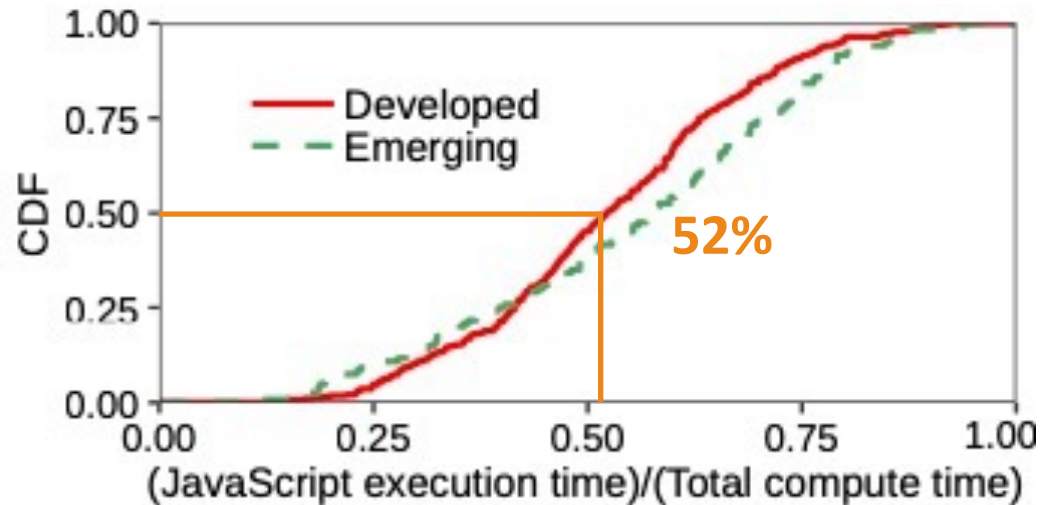


% of pages with load times > 3

# Why Are Computation Delays Significant?

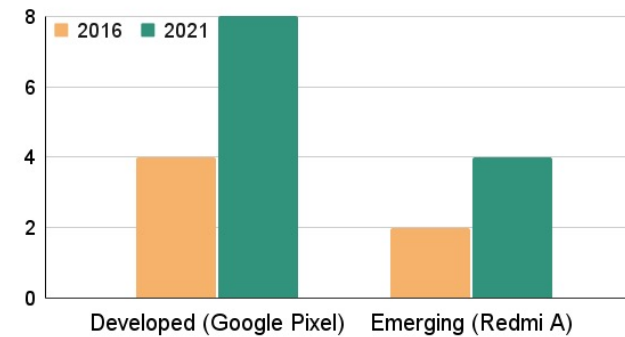
## JavaScript

- 80% more JavaScript over the last 5 years

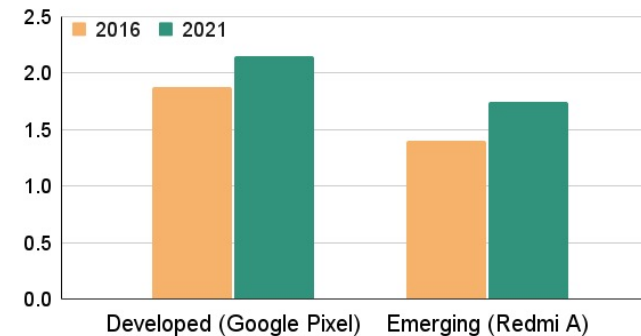


## Mobile Devices

Increasing CPU Core Counts

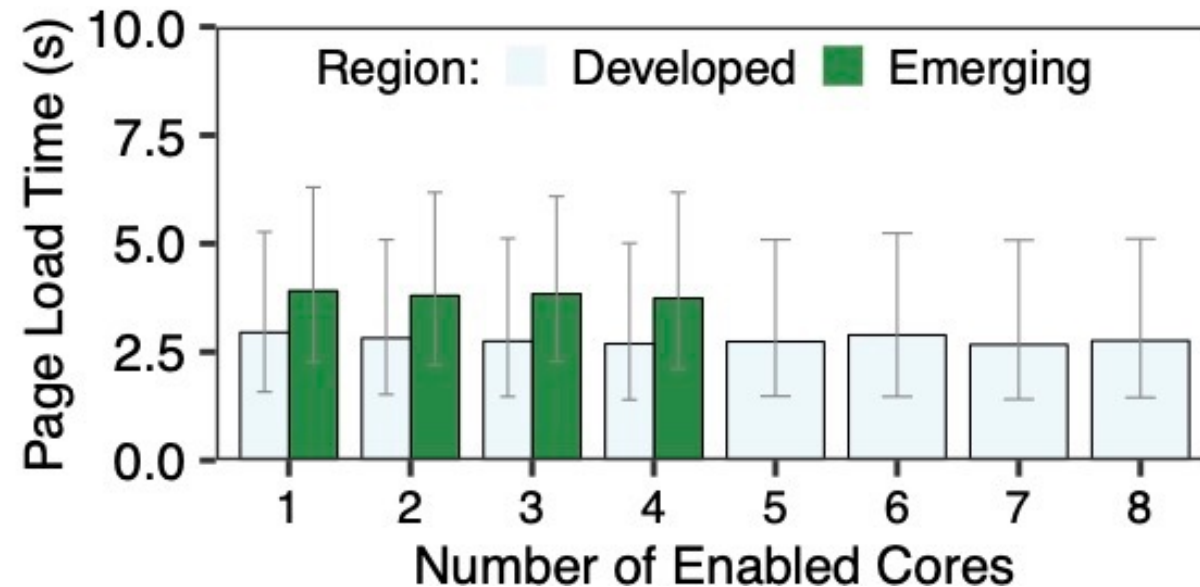


Increasing CPU Clock Speed (GHz)



primary compute  
improvements  
=  
more cores

# More cores != Better performance



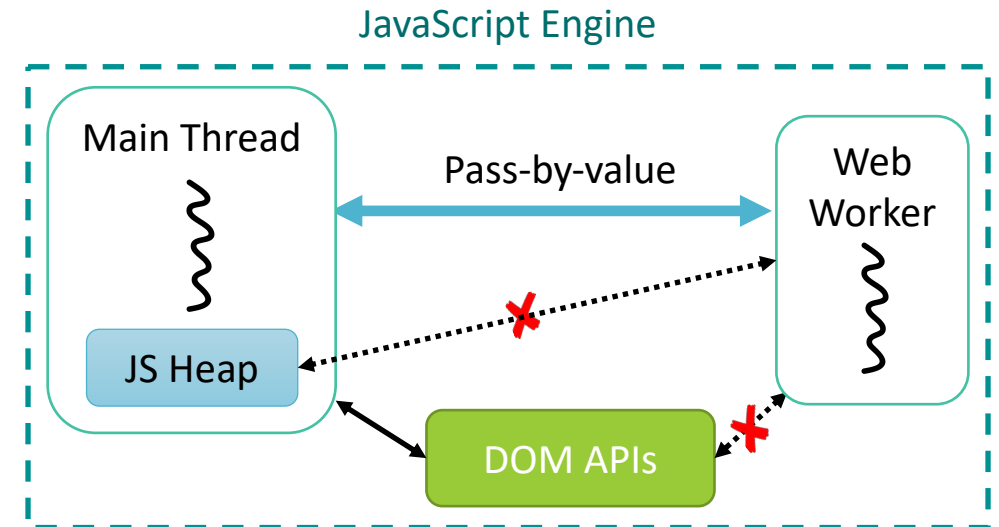
Reason: Single-thread execution



Solution: Parallelizing JavaScript

# Parallelization Opportunities

- Web workers are widely supported by browsers
- Constrained APIs:
  - No access to DOM APIs
  - No access to the main global state
- Legacy pages are highly amenable to *safe* parallelization



# of cores	Speedup in JS Runtimes
2 cores	49%
4 cores	75%
8 cores	87%

# Challenges


## C1: Ensuring Correctness

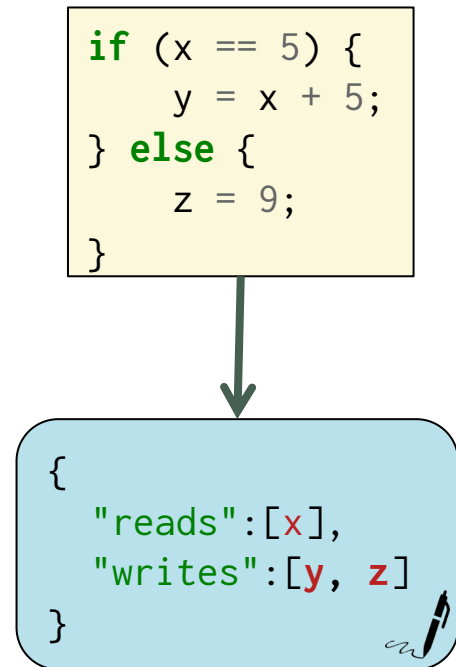
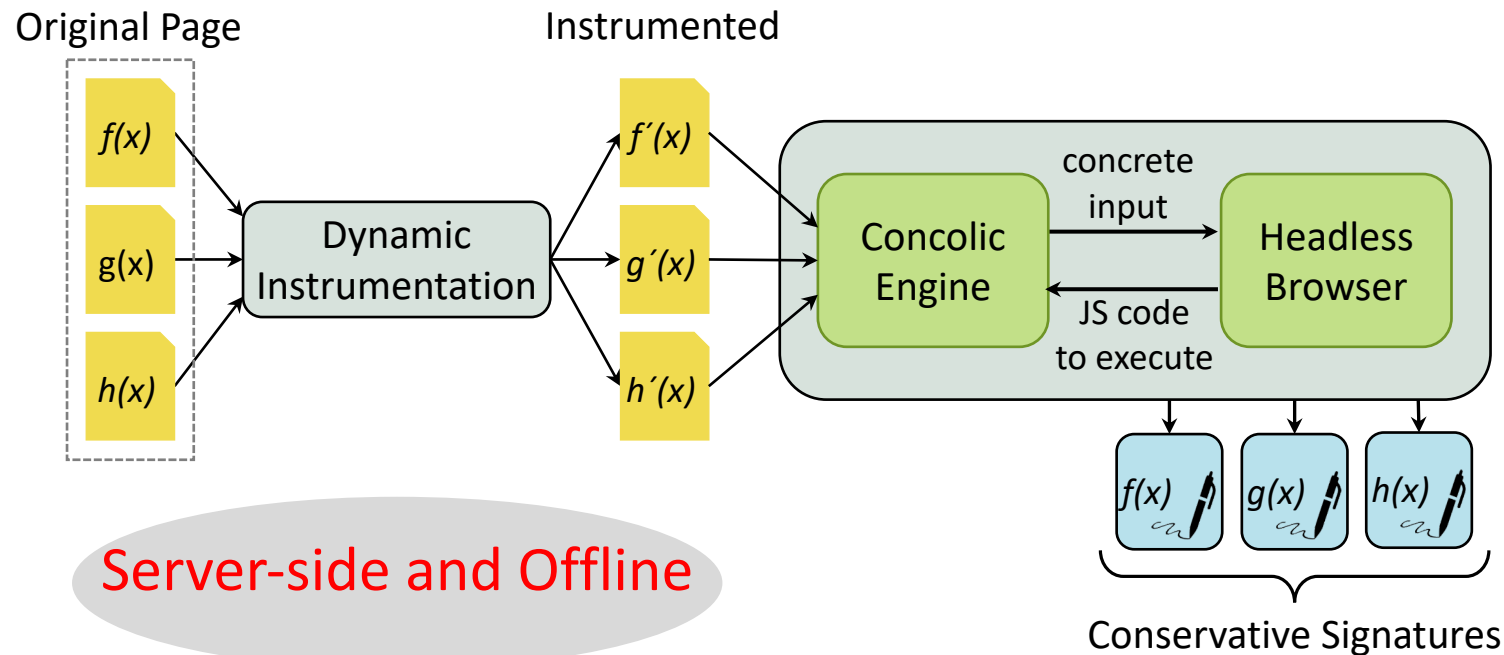
The exact state accessed by parallelized JavaScript  
- Despite non-determinism and across all possible control paths



Conservative  
Signatures

# Signature Generation

Conservative signatures  Dynamic analysis: track read/writes to page state  
Concolic execution: explore all possible control paths





# Challenges

## C1: Ensuring Correctness

The exact state accessed by parallelized JavaScript  
- Despite non-determinism and across all possible control paths



Conservative  
Signatures

## C2: Web Workers Constrained APIs

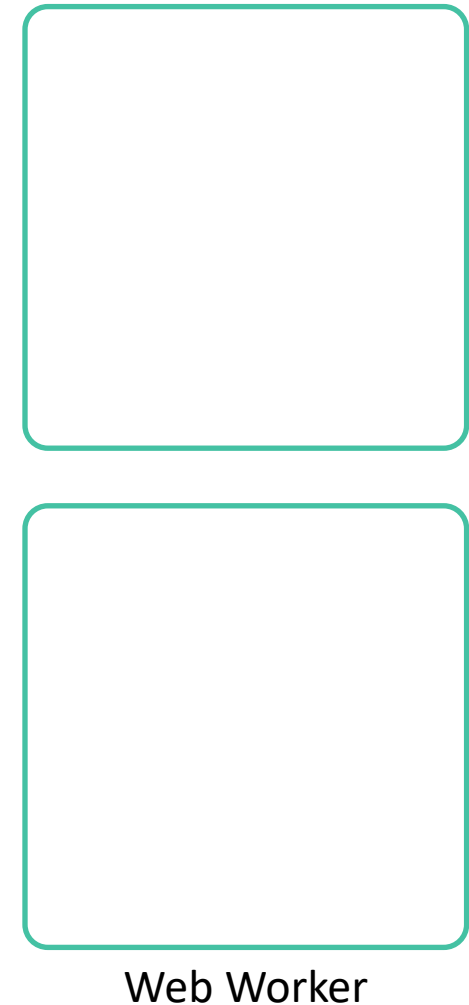
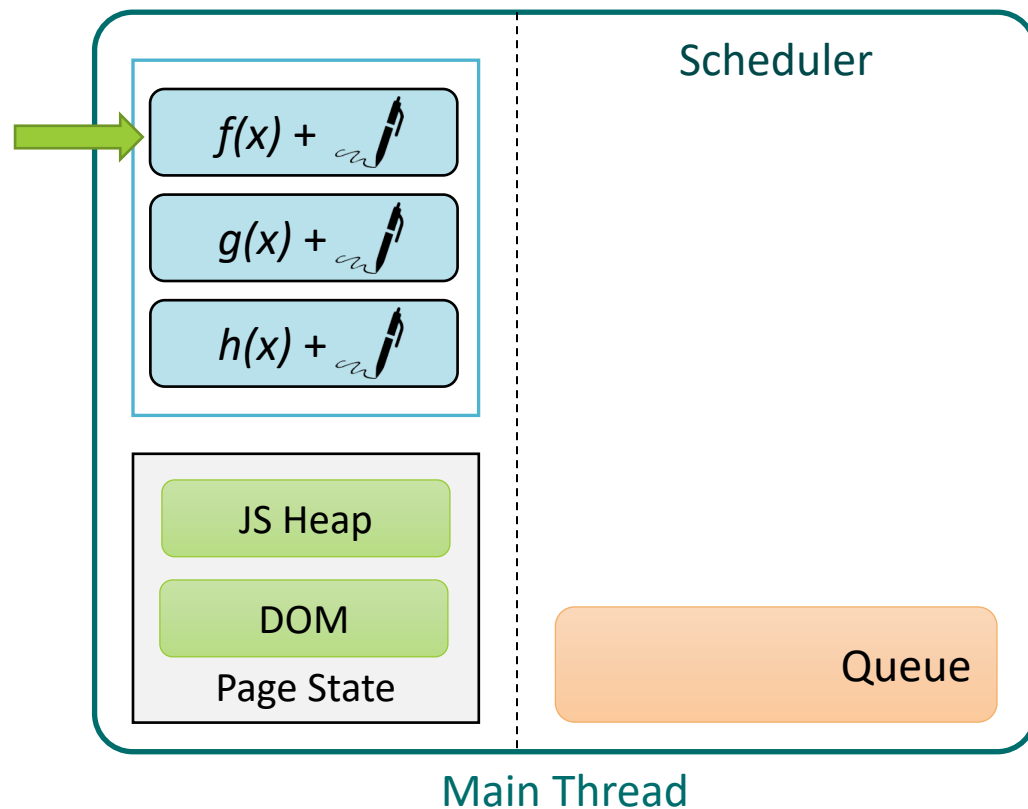
- How to parallelize execution with constrained APIs?



Client-side  
Parallelization  
Scheduler

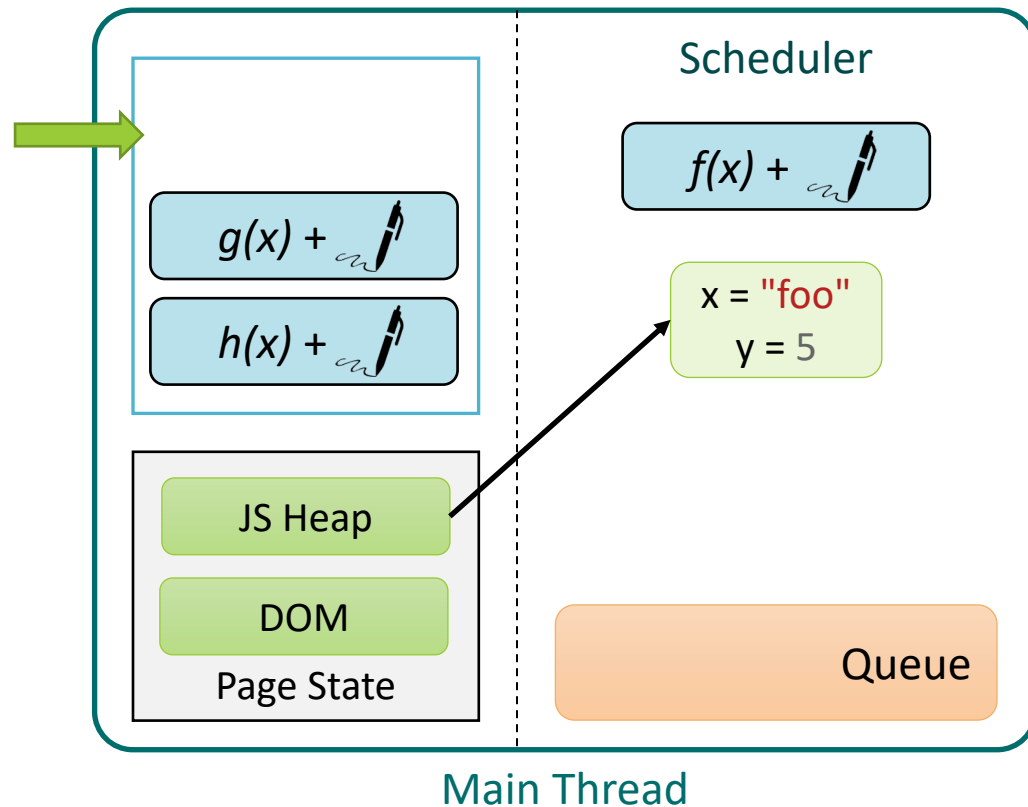
# Horcrux Dynamic Scheduler

- Runs in the main browser thread
- Only task: manage offloads in event-driven mode



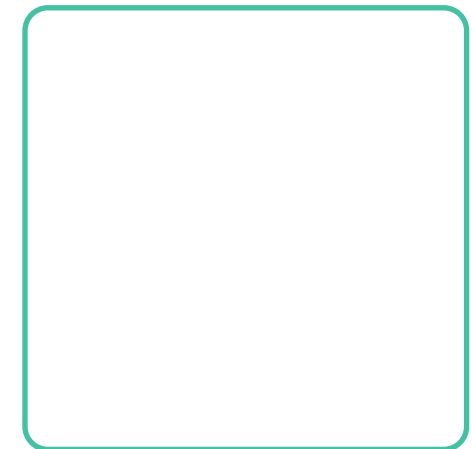
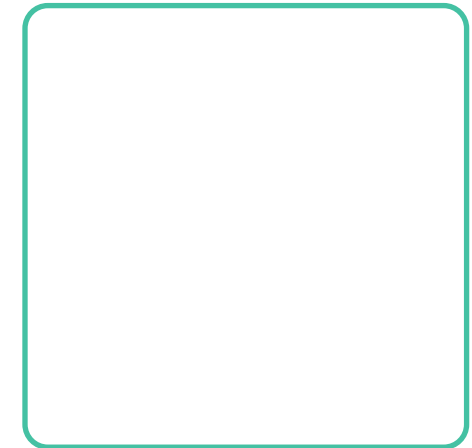
# Horcrux Dynamic Scheduler

- Runs in the main browser thread
- Only task: manage offloads in event-driven mode



$f(x)$  reads & writes to  $x, y$

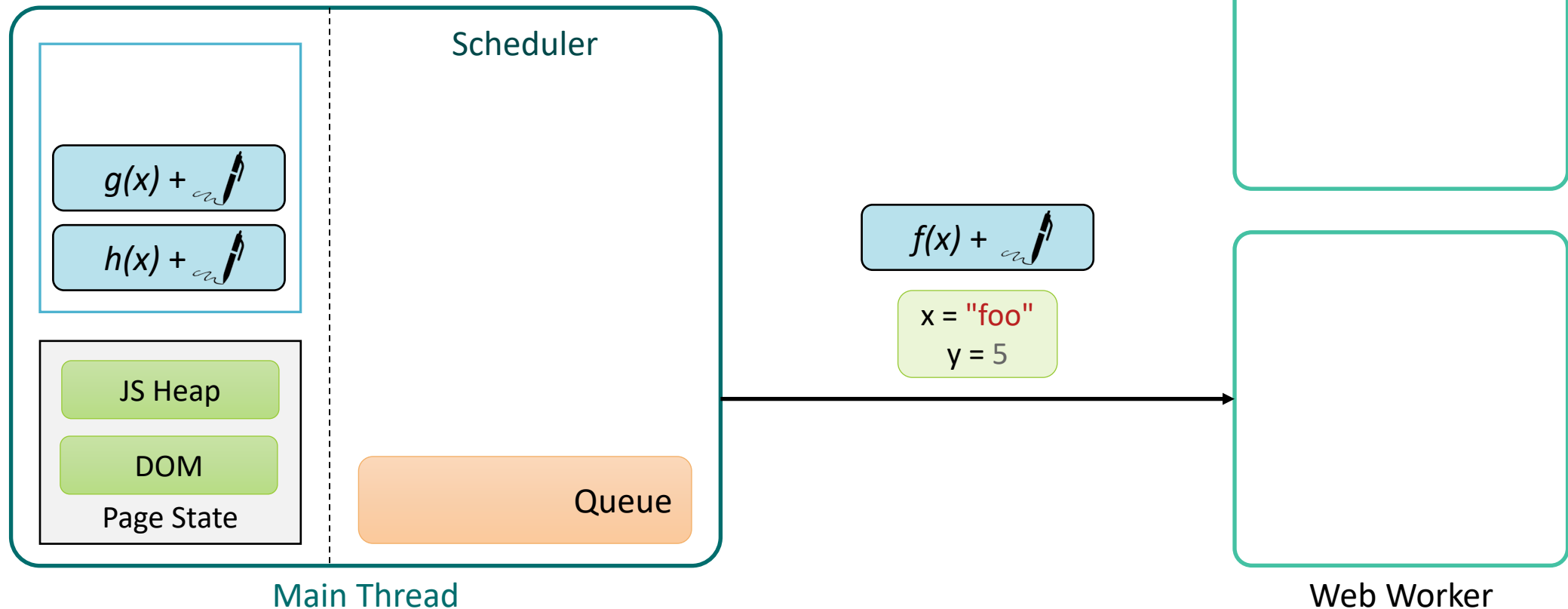
no state dependencies



Web Worker

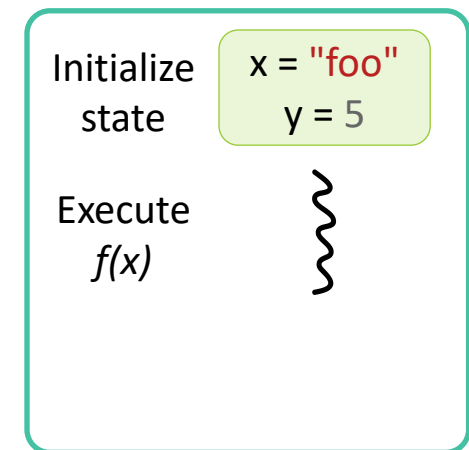
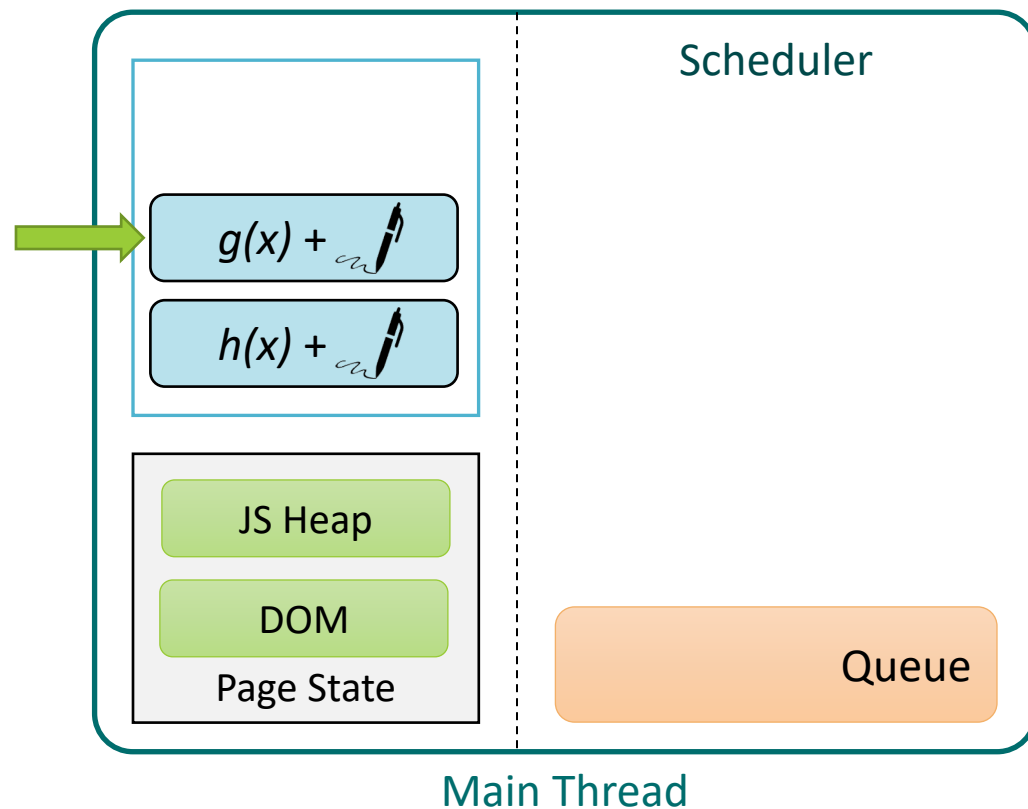
# Horcrux Dynamic Scheduler

- Runs in the main browser thread
- Only task: manage offloads in event-driven mode



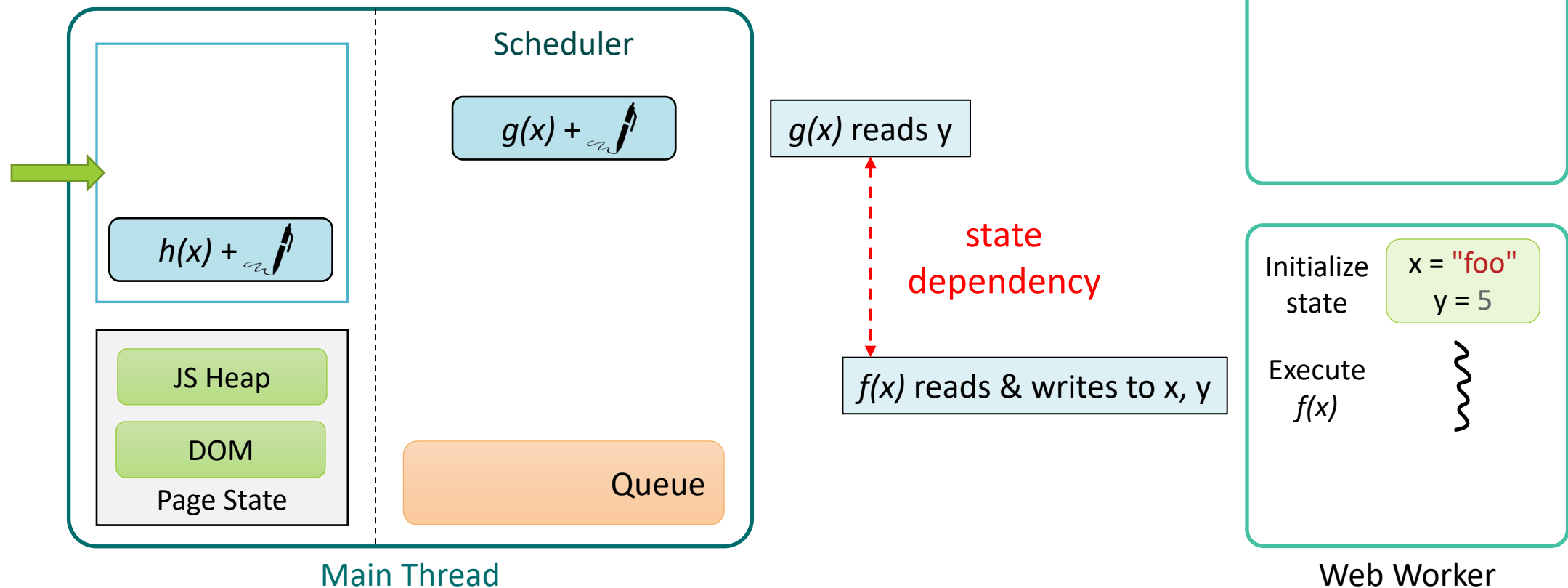
# Horcrux Dynamic Scheduler

- Runs in the main browser thread
- Only task: manage offloads in event-driven mode



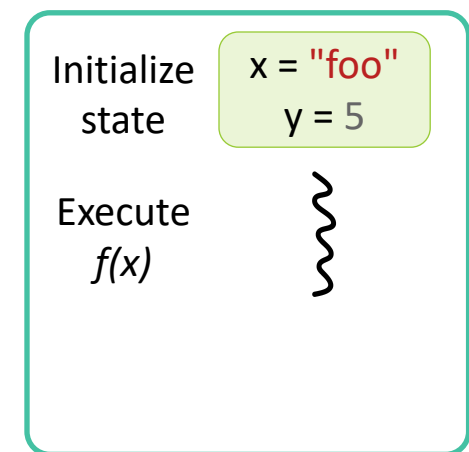
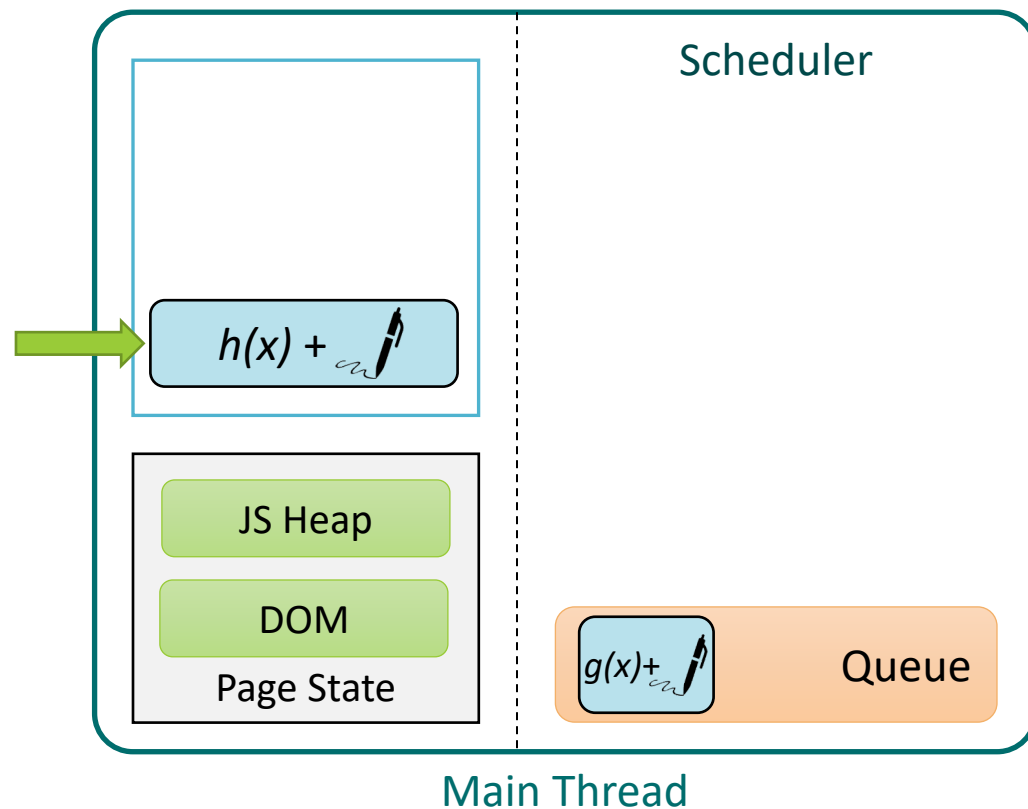
# Horcrux Dynamic Scheduler

- Runs in the main browser thread
- Only task: manage offloads in event-driven mode



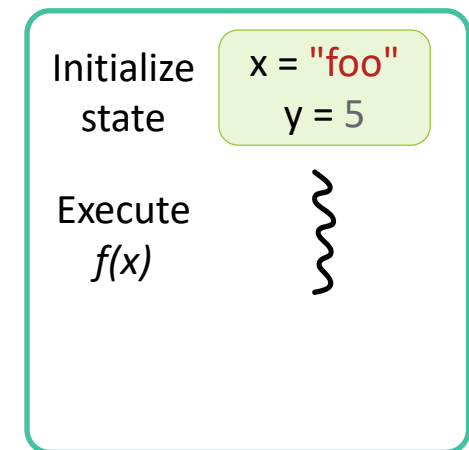
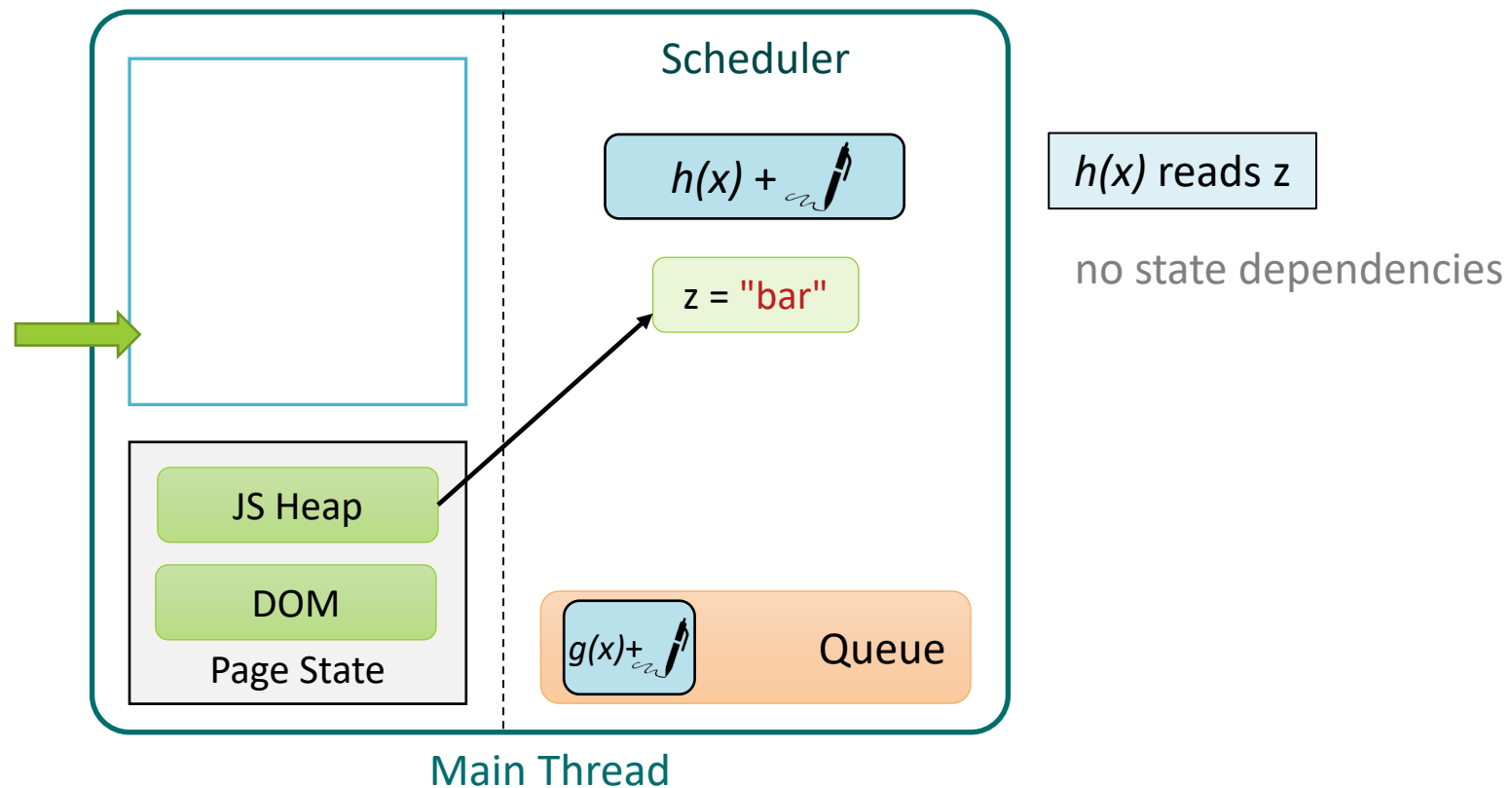
# Horcrux Dynamic Scheduler

- Runs in the main browser thread
- Only task: manage offloads in event-driven mode



# Horcrux Dynamic Scheduler

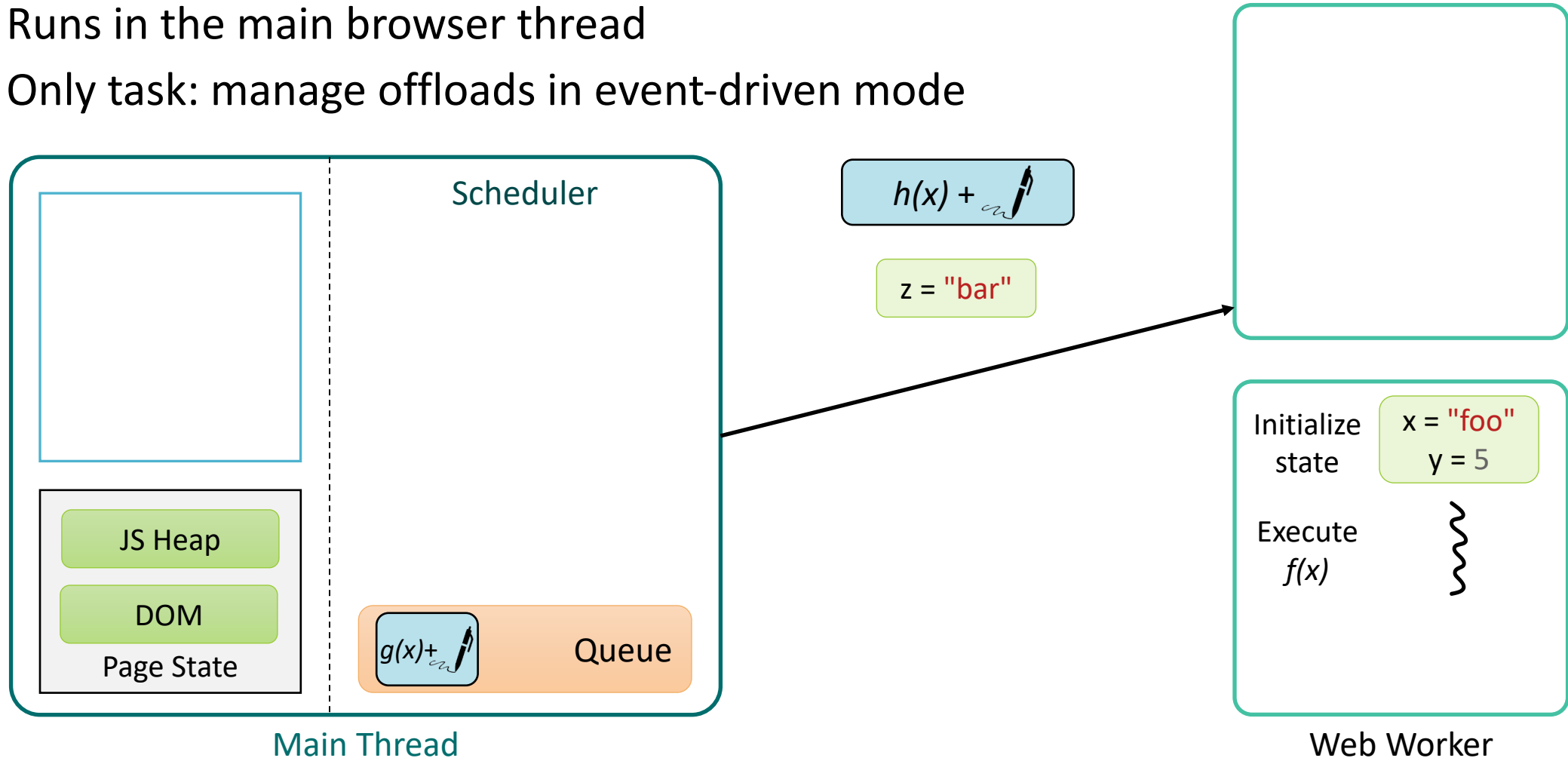
- Runs in the main browser thread
- Only task: manage offloads in event-driven mode





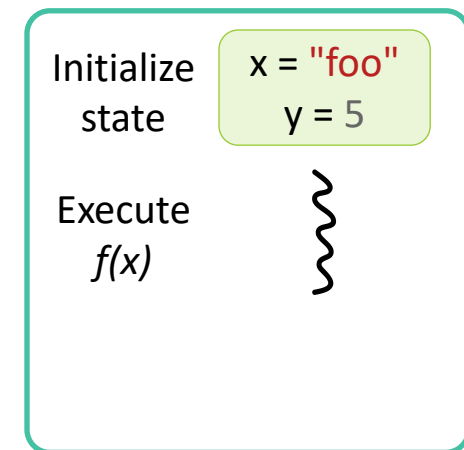
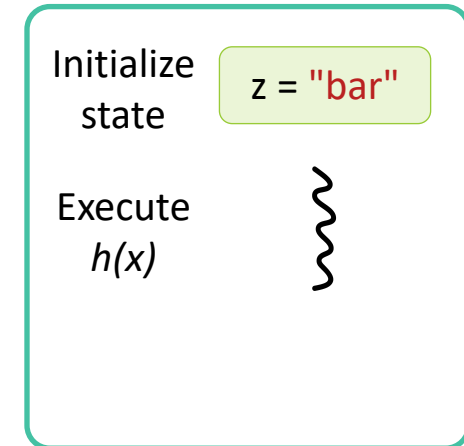
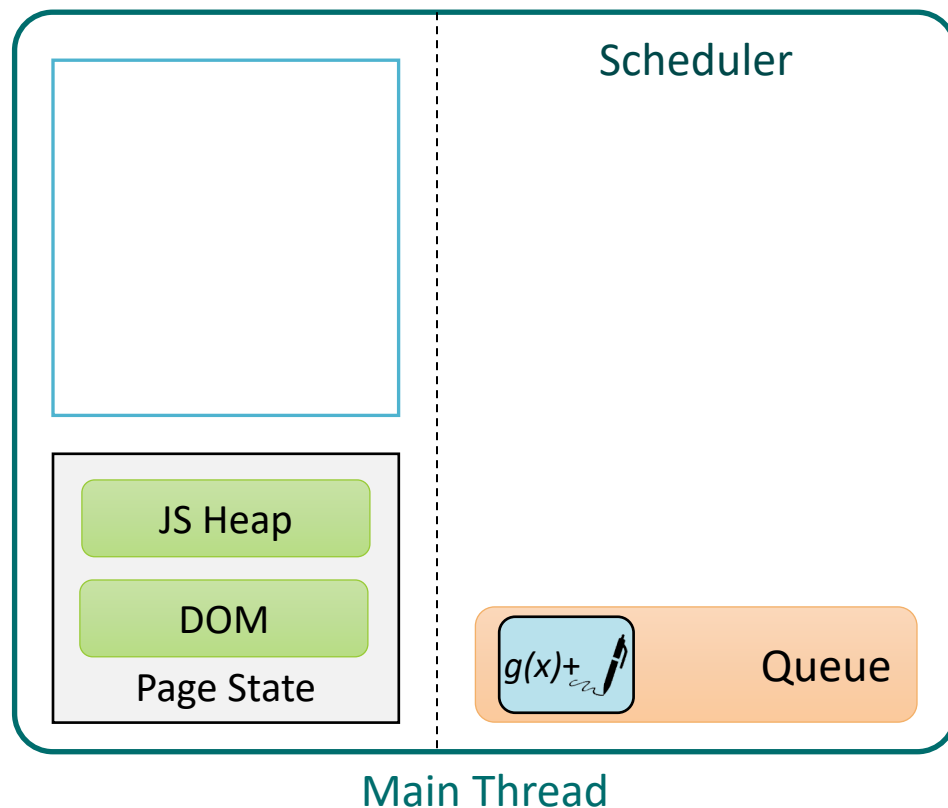
# Horcrux Dynamic Scheduler

- Runs in the main browser thread
- Only task: manage offloads in event-driven mode



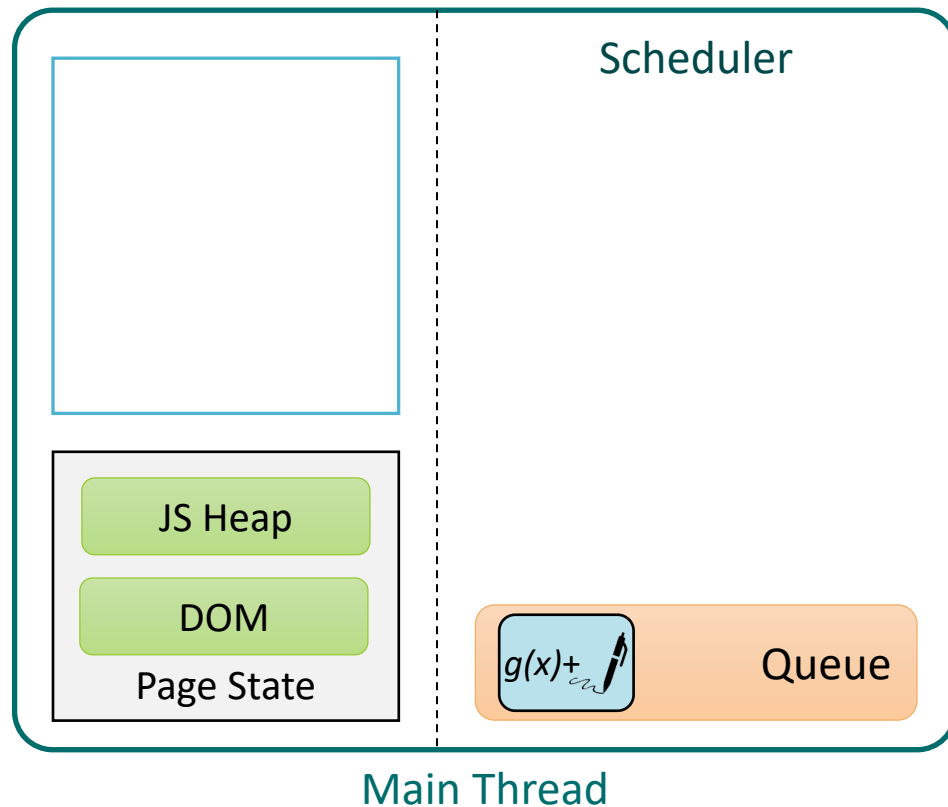
# Horcrux Dynamic Scheduler

- Runs in the main browser thread
- Only task: manage offloads in event-driven mode

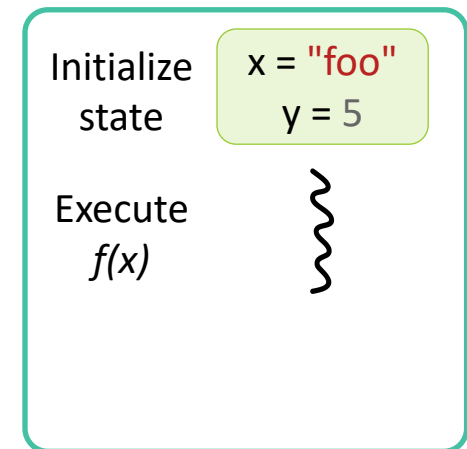
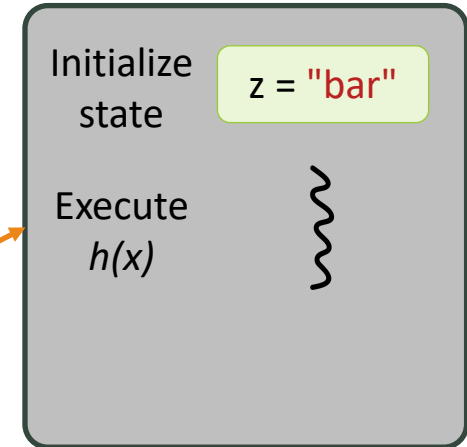


# Horcrux Dynamic Scheduler

- Runs in the main browser thread
- Only task: manage offloads in event-driven mode



Pauses the execution



$getElementById()$

# Challenges

## C1: Ensuring Correctness

- The exact state accessed by parallelized JavaScript
- Despite non-determinism and across all possible control paths



Conservative Signatures

## C2: Web Workers Constrained APIs

- How to parallelize execution with constrained APIs?



Client-side Parallelization Scheduler

## C3: Offloading overheads

- Pass-by-value I/O can take ~0-10 ms



Root-function Granularity

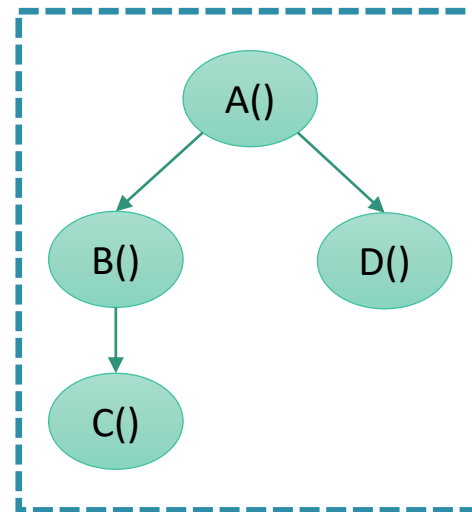
# Offloading Granularity

- Trade-off: parallelization benefits vs. offloading overheads
- Solution: offloading root-function invocations

Original Page



```
<script>  
  function A() {  
    ...  
    B(); // invokes C()  
    D();  
    ...  
  }  
  A();  
</script>
```

Root-function



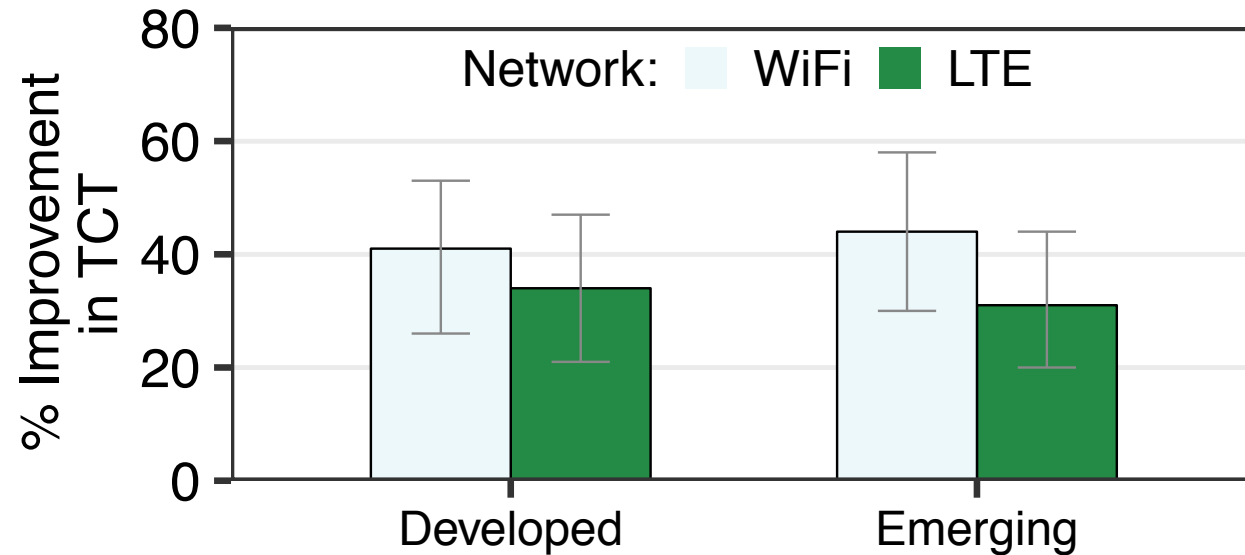
4x fewer offloads and  
73% of parallelization  
benefits

# Evaluation Questions

- Impact on browser computation delays? 
- Impact on end-to-end performance? 
- Horcrux comparison to prior compute optimizations?
- What do conservative signatures forgo?
- How much are the server-side overheads?

# Computation Delay Reductions

- Total Computation Time (TCT)



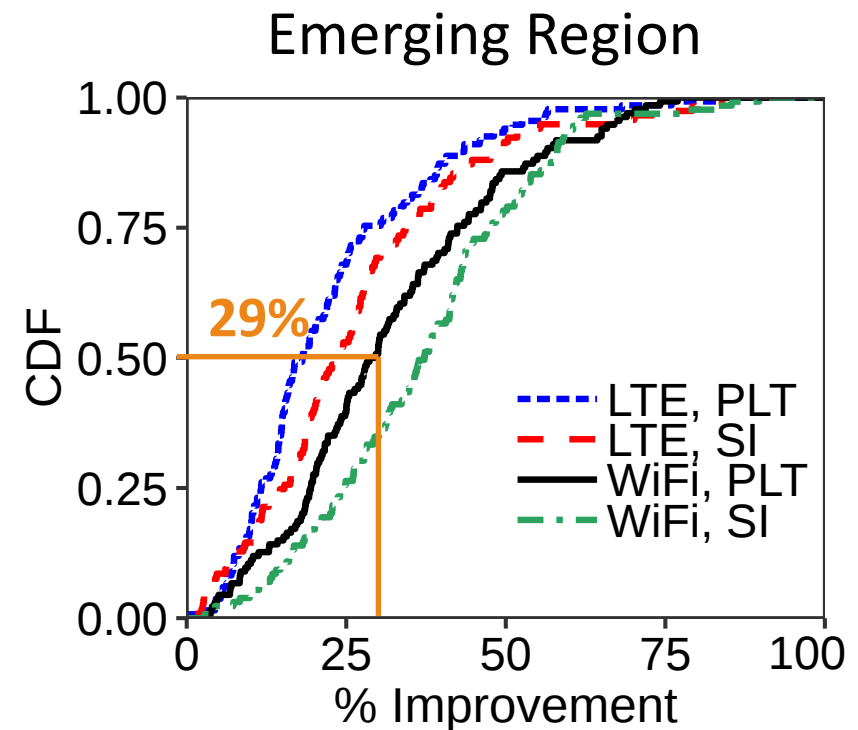
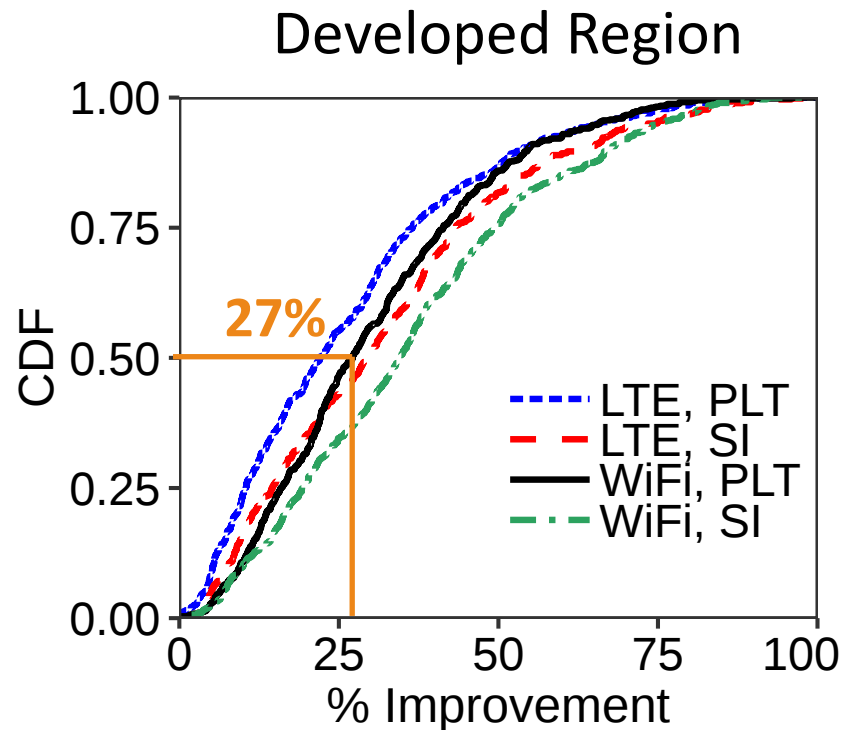
Median Improvements:

Developed: WiFi (41%), LTE (34%)

Emerging: WiFi (44%), LTE (31%)

# End-to-end Performance Improvements

- Page Load Time (PLT) and Speed Index (SI)





# Conclusion

- More cores != Better performance
- Horcrux automatically parallelizes JavaScript execution using concolic execution to take advantage of phones multi-core CPUs



[github.com/ShaghayeghMrdn/horcrux-osdi21](https://github.com/ShaghayeghMrdn/horcrux-osdi21)



[shaghayegh@cs.ucla.edu](mailto:shaghayegh@cs.ucla.edu)