Caching in Mobile Apps

COS 316: Principles of Computer System Design

Lecture 10

10/6/2022 | Murali Ramanujam | muralisr.com

Outline

- Use of caches
- Libraries in mobile apps
- Cache performance in mobile apps
  - Reasons for suboptimal caching
  - Improving cache performance
- Caching more things
  - Computations?

What are caches useful for?

Reusing prior content to improve performance

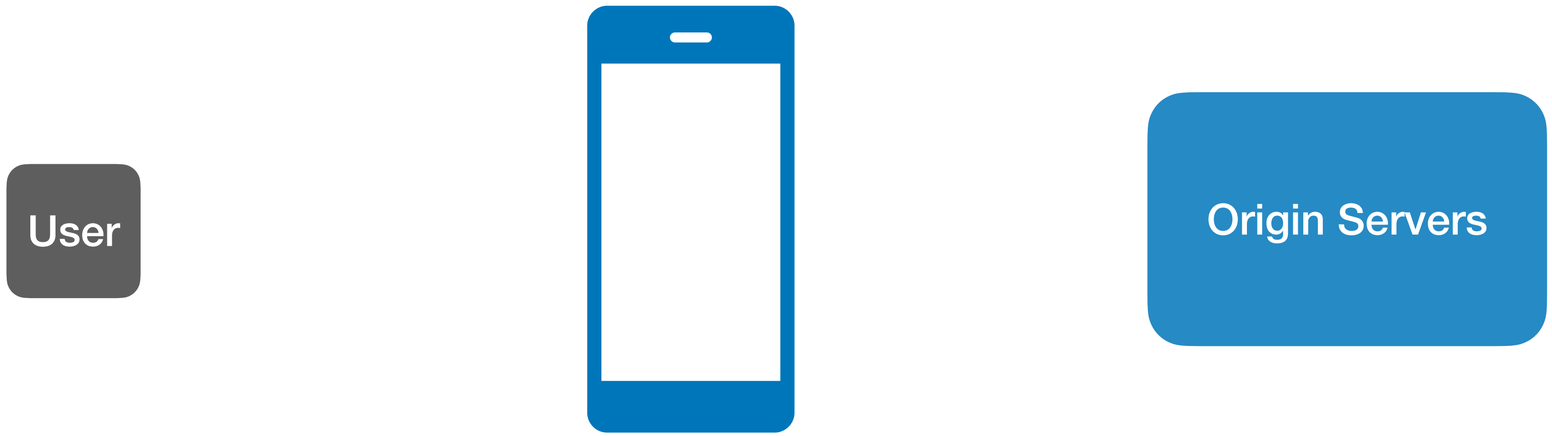
What are caches useful for?

Reusing prior content to improve performance

- CPU memory
- Databases
- CDNs
- Browsers
- And even mobile apps

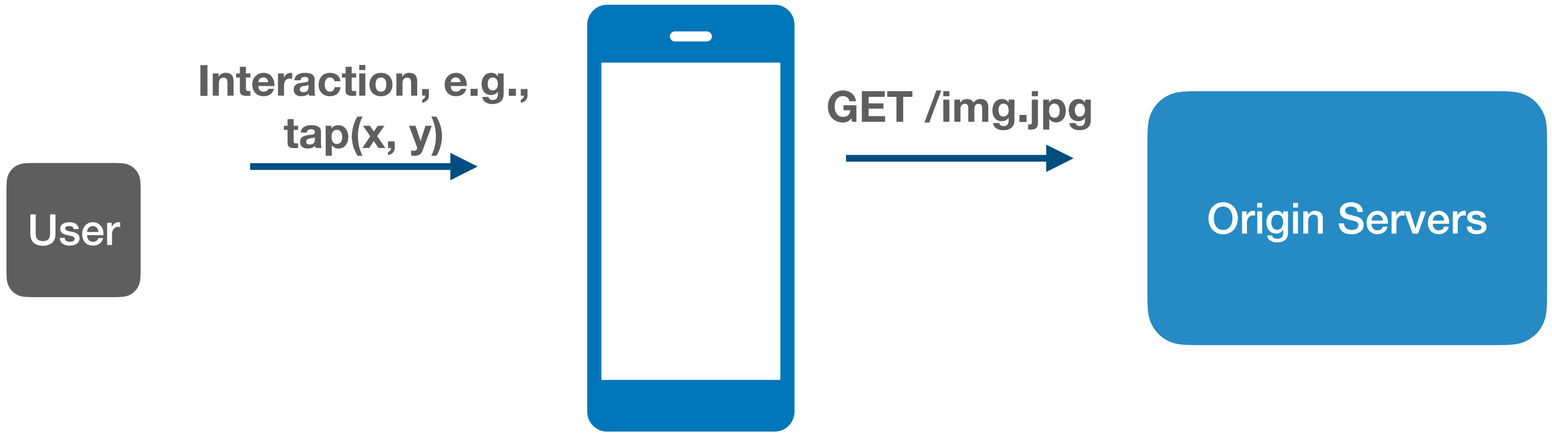
Interacting with mobile apps

Apps are similar to web browsers



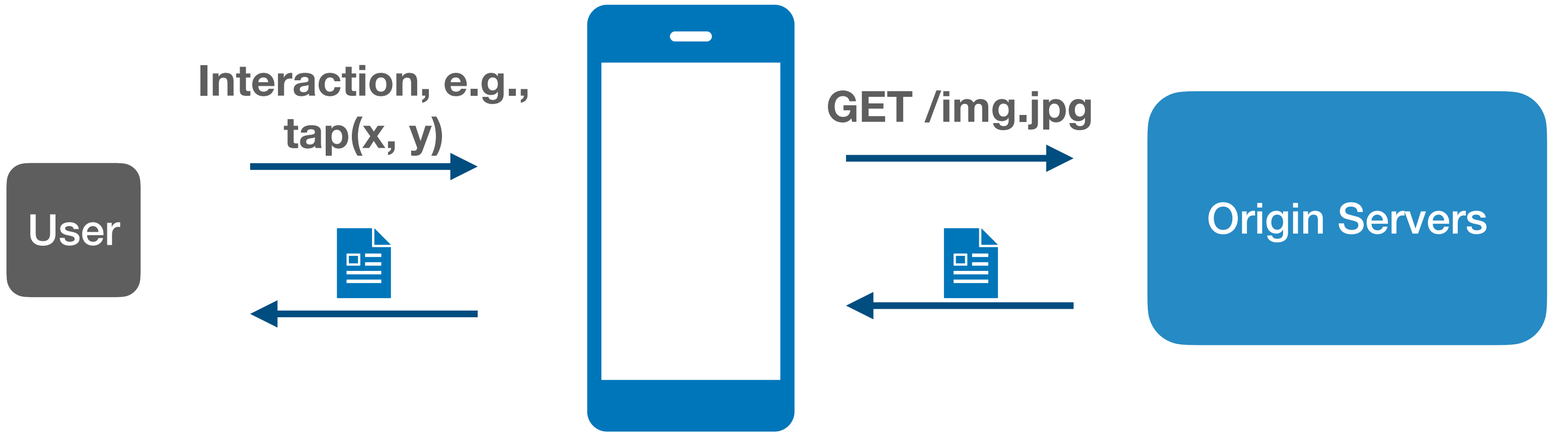
Interacting with mobile apps

Apps are similar to web browsers



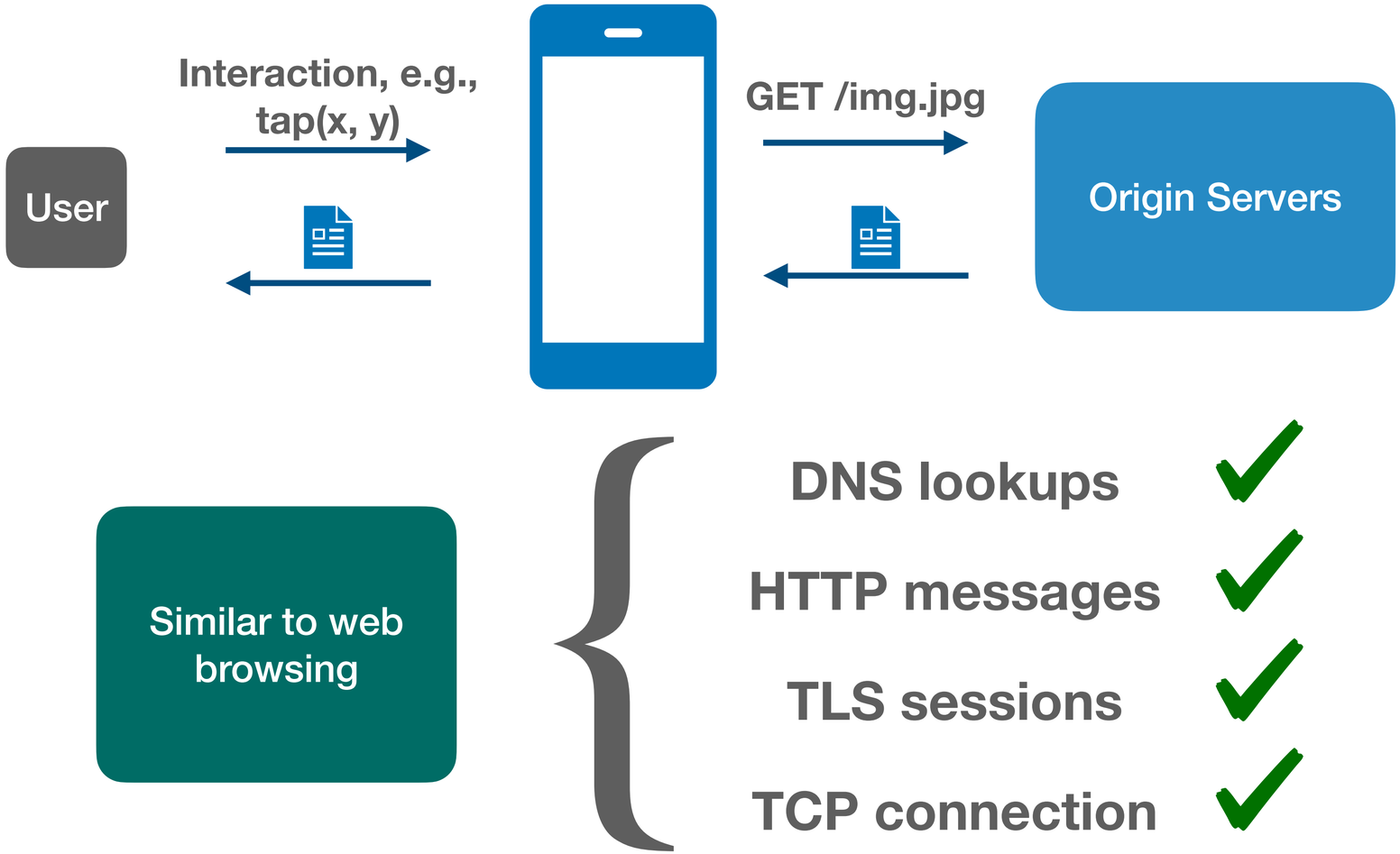
Interacting with mobile apps

Apps are similar to web browsers



Interacting with mobile apps

Apps are similar to web browsers



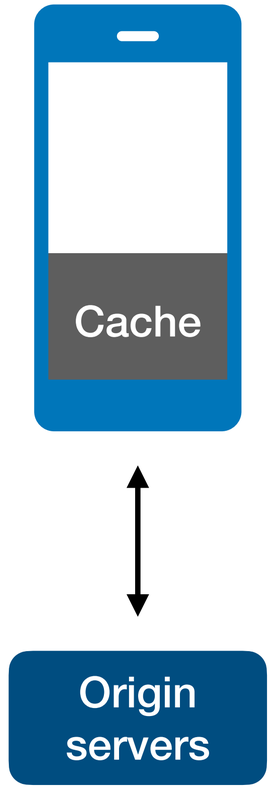
Client side caches for apps

Client side caches for apps

- Apps incorporate web caches
  - Similar to browsers

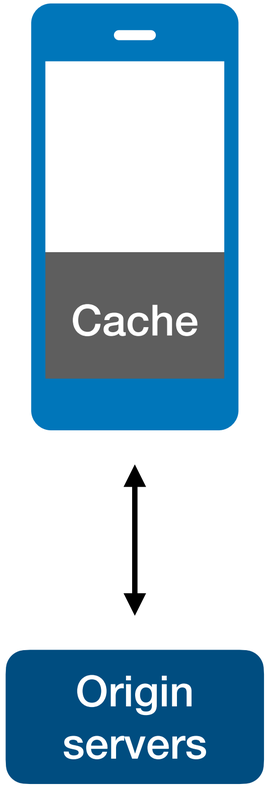
Client side caches for apps

- Apps incorporate web caches
  - Similar to browsers



Client side caches for apps

- Apps incorporate web caches
  - Similar to browsers
- Caching libraries
  - OkHttp
  - Ion
  - Picasso
  - .. and more



Caching libraries

- HTTP request/response handling
  - Generate appropriate request headers
  - Parse and handle response headers
- Cache consistency
- Caching algorithm
  - LRU, LFU, etc.

Caching libraries

- HTTP request/response handling
  - Generate appropriate request headers
  - Parse and handle response headers
- Cache consistency
- Caching algorithm
  - LRU, LFU, etc.

How do these caching libraries perform today?

State of cache performance today

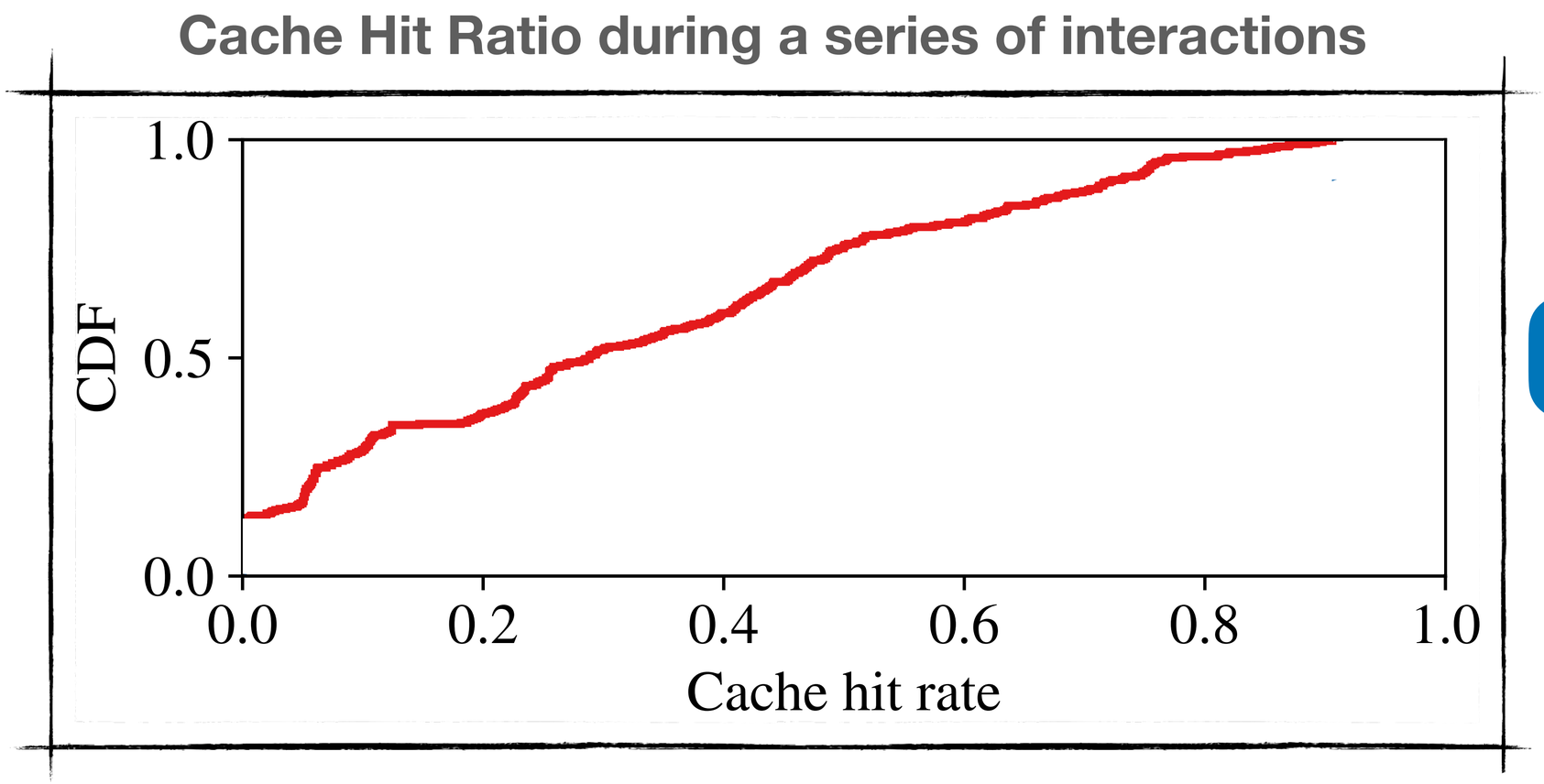
Cache Hit Ratio during a series of interactions

State of cache performance today

Cache Hit Ratio during a series of interactions

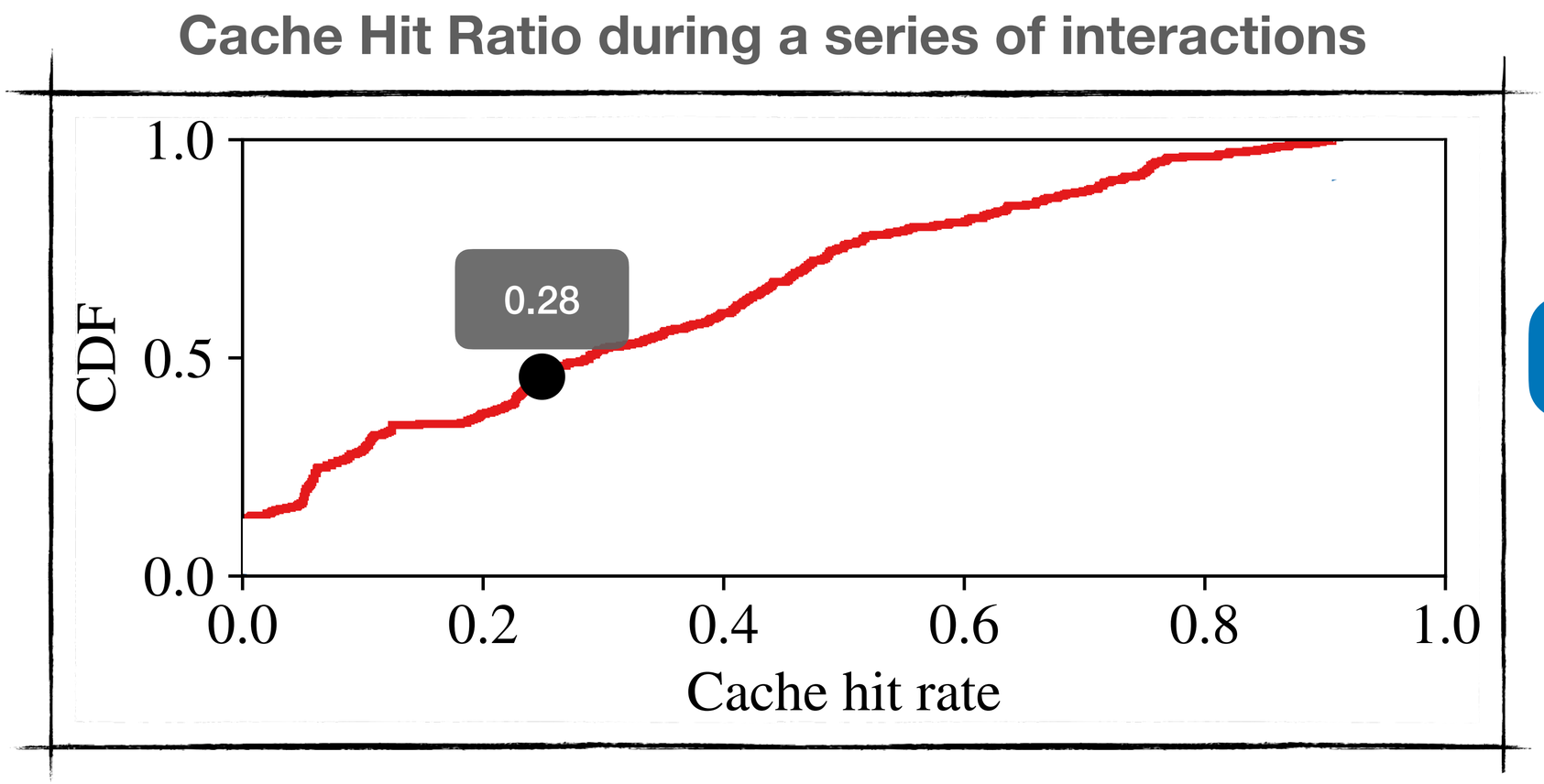
Subsequent series of interactions

State of cache performance today



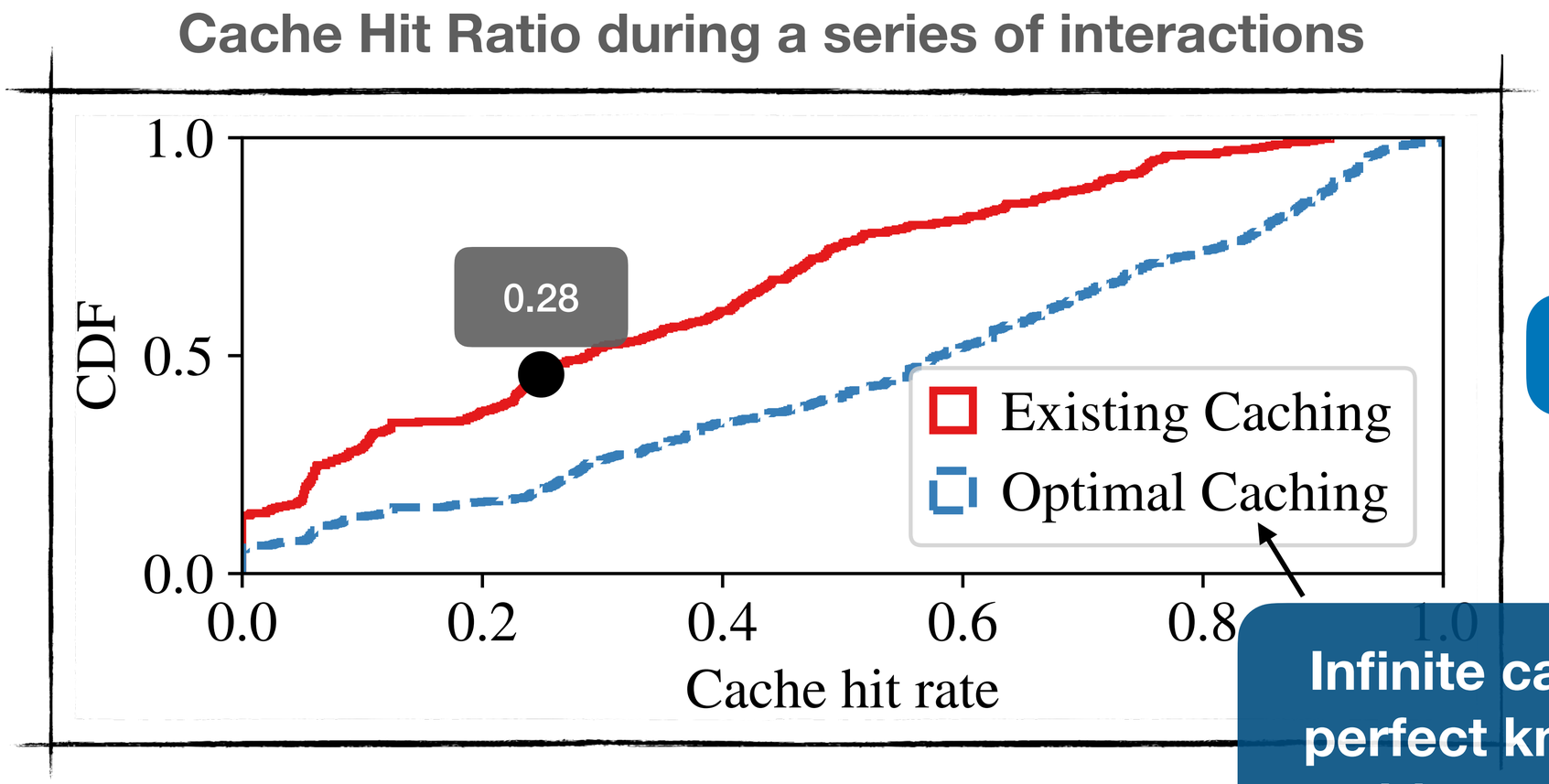
Subsequent series of interactions

State of cache performance today



Subsequent series of interactions

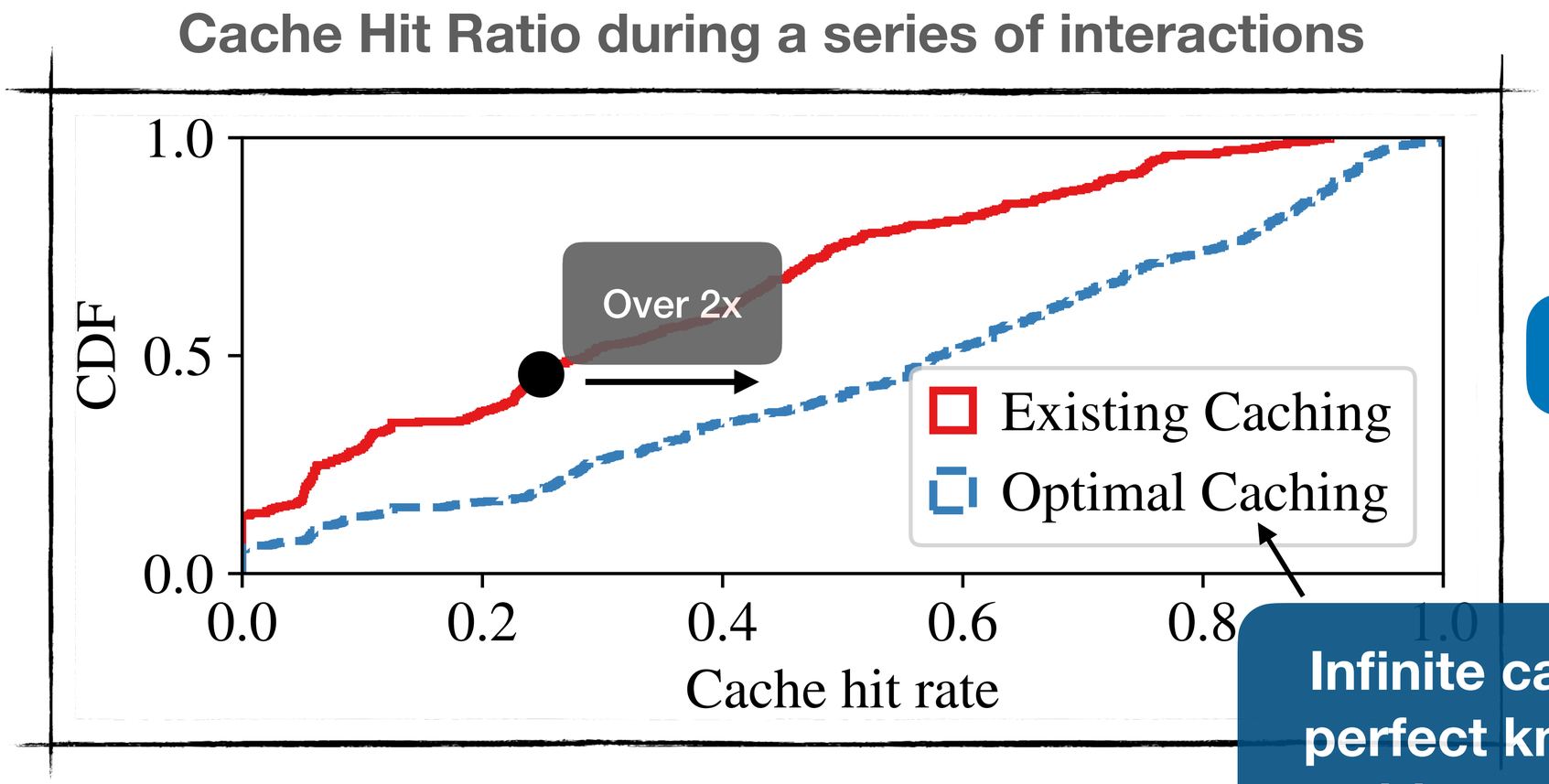
State of cache performance today



Subsequent series of interactions

Infinite capacity and perfect knowledge of object expirations

State of cache performance today

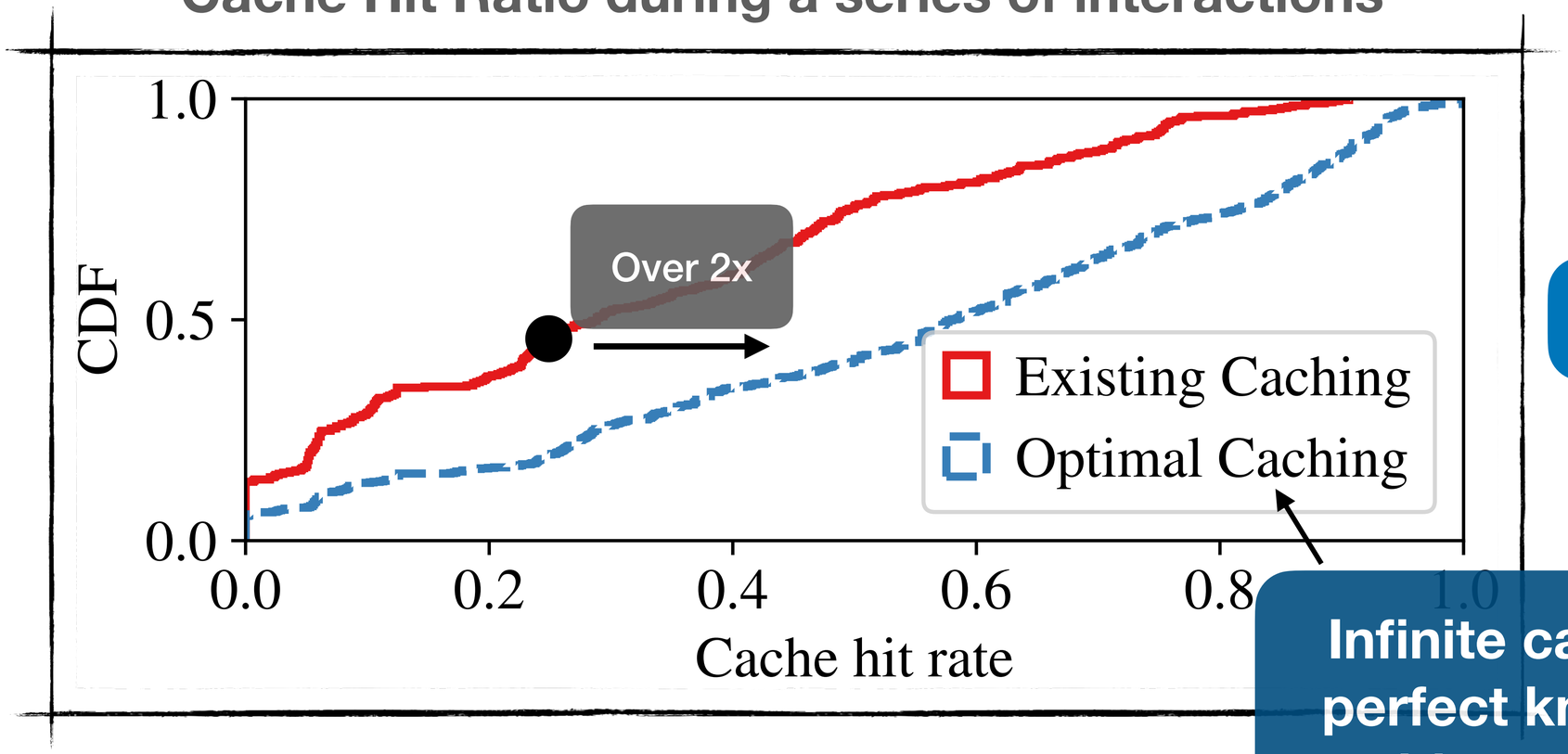


Subsequent series of interactions

Infinite capacity and perfect knowledge of object expirations

State of cache performance today

Cache Hit Ratio during a series of interactions



Subsequent series of interactions

Infinite capacity and perfect knowledge of object expirations

Current caching is suboptimal

App responsiveness

App responsiveness

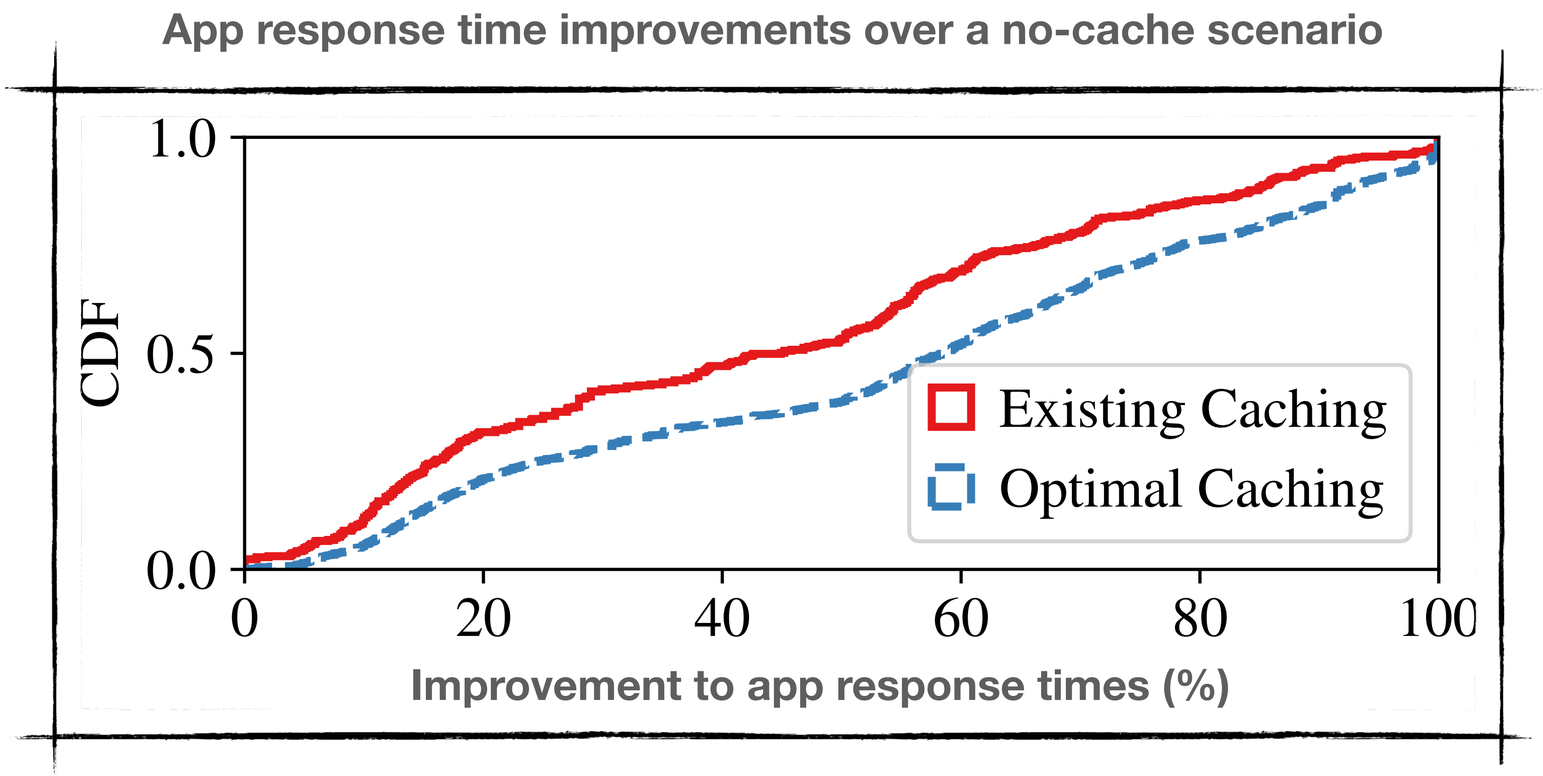
How does cache hit rate affect responsiveness?

Subsequent series of interactions

Optimal version has infinite capacity, perfect expiration

App responsiveness

How does cache hit rate affect responsiveness?

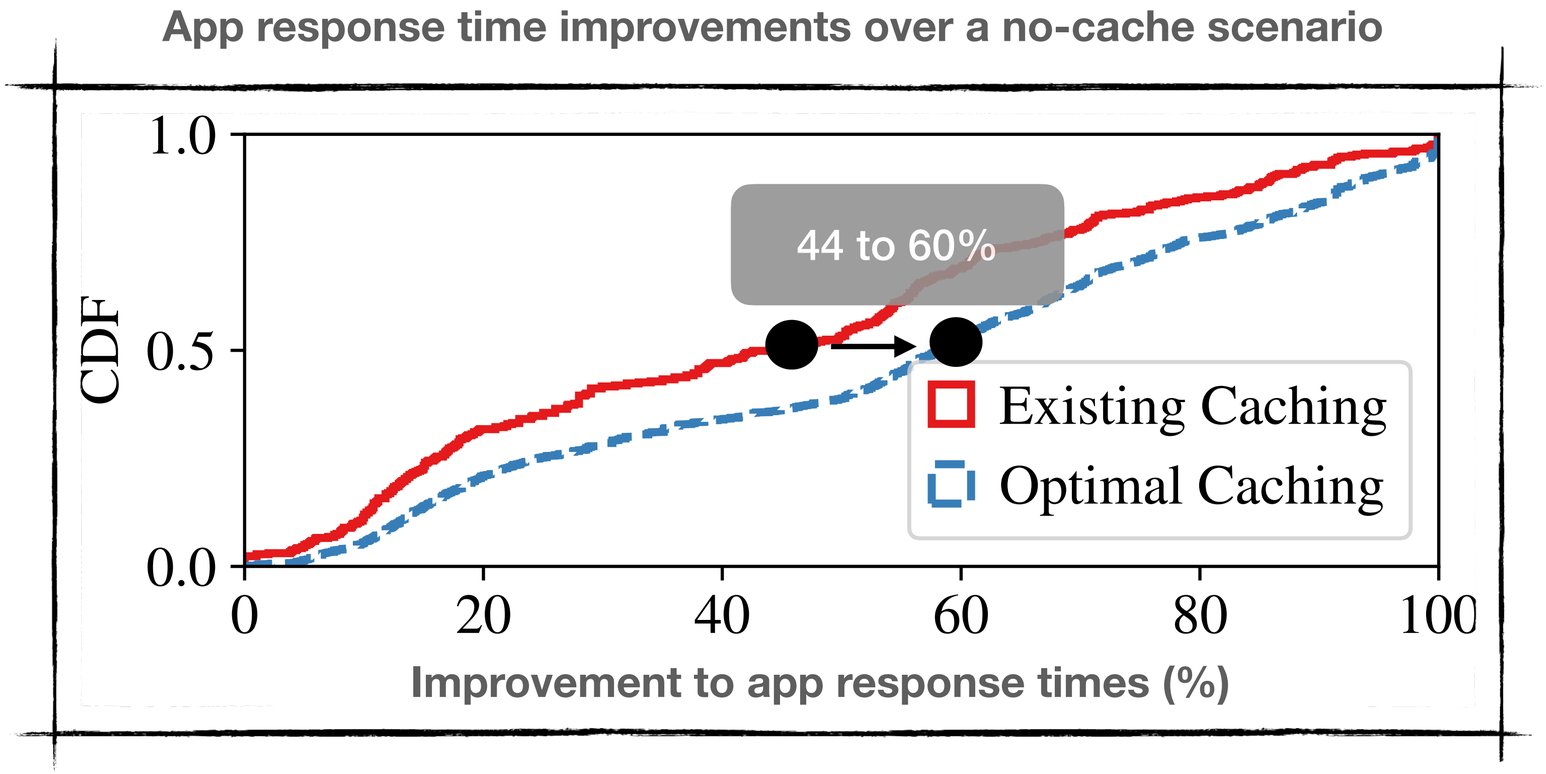


Subsequent series of interactions

Optimal version has infinite capacity, perfect expiration

App responsiveness

How does cache hit rate affect responsiveness?

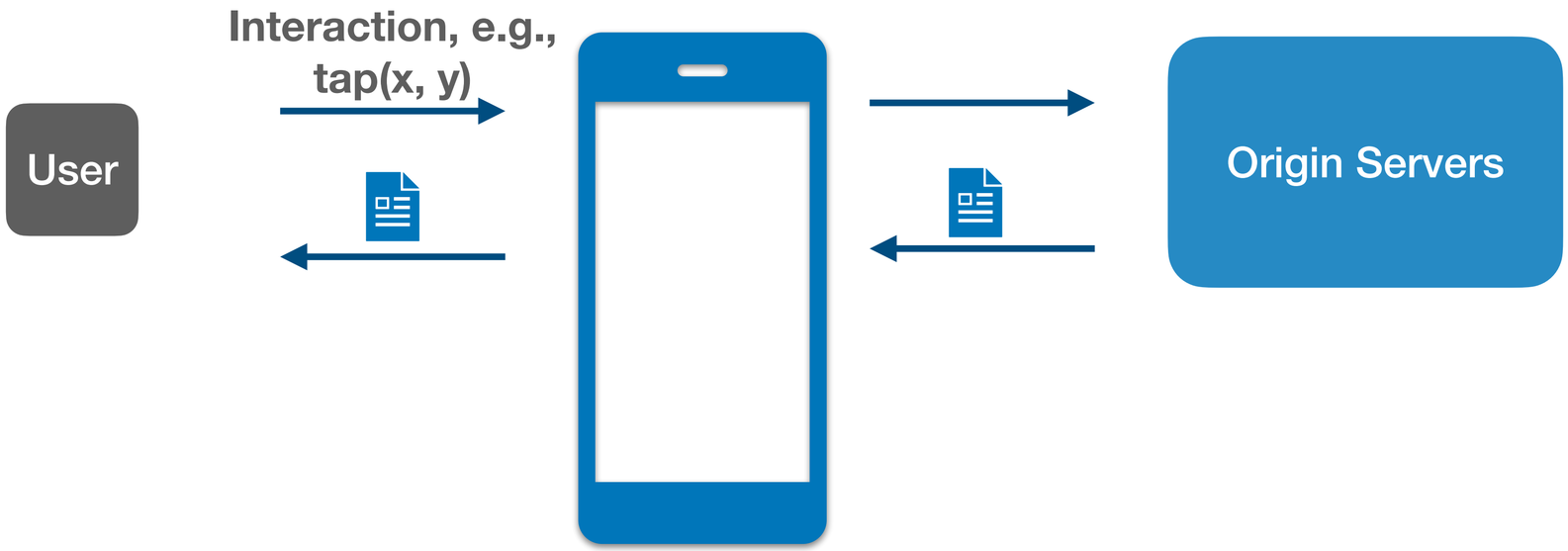


Subsequent series of interactions

Optimal version has infinite capacity, perfect expiration

Cache performance

Why isn’t cache hit rate optimal?



Cache performance

From last lecture

HTTP Response Header for Cache Control

- Whether to cache
  - no store: no cache should store it
- Who should cache
  - private: only a private cache (e.g., browser)
  - public: any cache, including shared ones
- How long to cache
  - max-age=N: for N seconds
  - must-revalidate: check with the server (don't return stale item)

Cache-Control: public, max-age=86400, must-revalidate

Cache performance

From last lecture

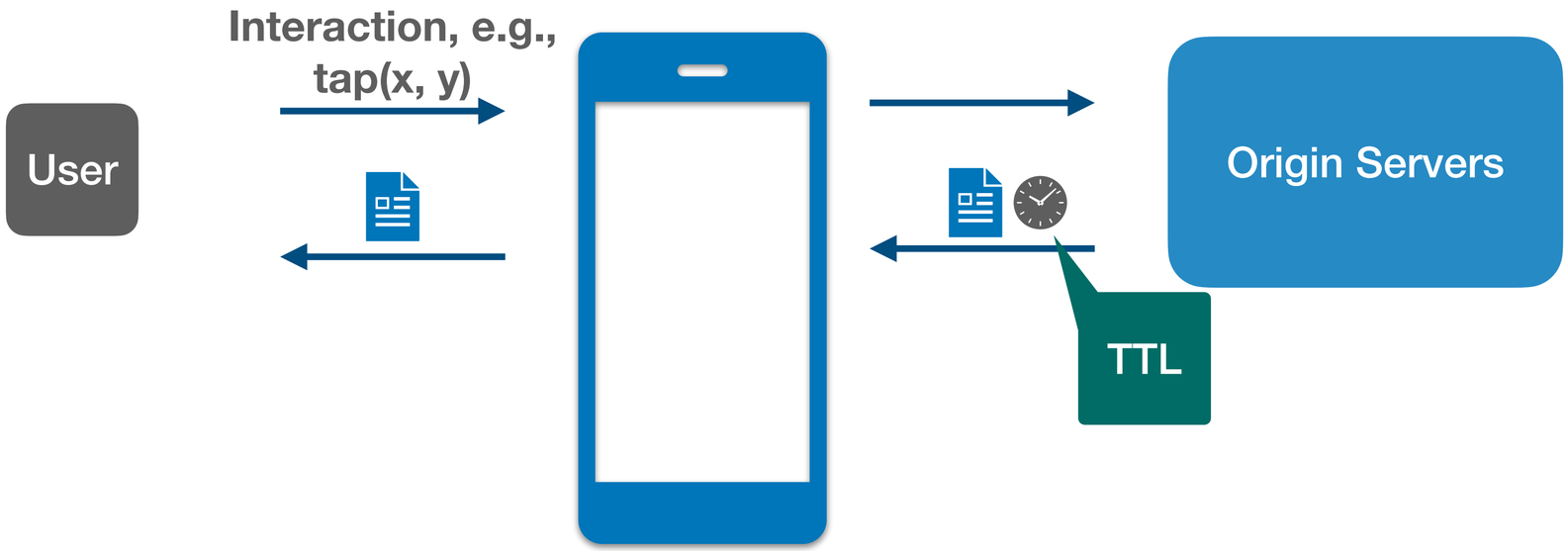
HTTP Response Header for Cache Control

- Whether to cache
  - no store: no cache should store it
- Who should cache
  - private: only a private cache (e.g., browser)
  - public: any cache, including shared ones
- How long to cache
  - max-age=N: for N seconds
  - must-revalidate: check with the server (don't return stale item)

Cache-Control: public, max-age=86400, must-revalidate

Cache performance

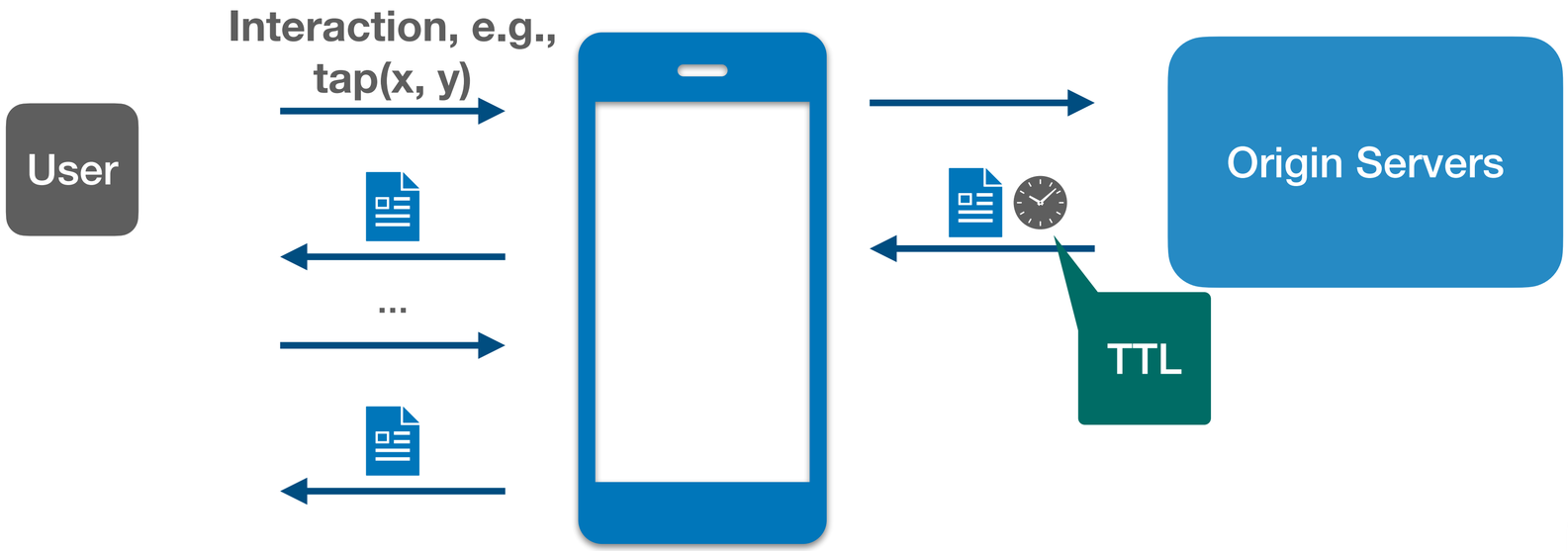
Why isn’t cache hit rate optimal?



TTL: Time to Live

Cache performance

Why isn’t cache hit rate optimal?



TTL: Time to Live

Caching correctly is difficult

As TTLs are hard to choose

Caching correctly is difficult

As TTLs are hard to choose

Developers need to set TTLs precisely such that they expire exactly when the content changes

Caching correctly is difficult

As TTLs are hard to choose

Developers need to set TTLs precisely such that they expire exactly when the content changes

wsj.com homepage at 9am
Ideal TTL: 30 seconds

wsj.com homepage at 2am
Ideal TTL: 3 hours

Caching correctly is difficult

As TTLs are hard to choose

Developers need to set TTLs precisely such that they expire exactly when the content changes

wsj.com homepage at 9am
Ideal TTL: 30 seconds

wsj.com homepage at 2am
Ideal TTL: 3 hours

Repeatedly fetch asset every second

Track when content changes

Caching correctly is difficult

As TTLs are hard to choose

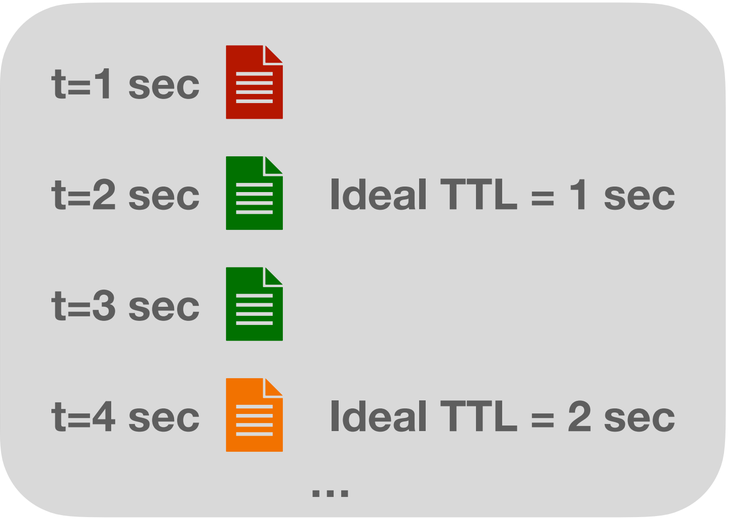
Developers need to set TTLs precisely such that they expire exactly when the content changes

wsj.com homepage at 9am
Ideal TTL: 30 seconds

wsj.com homepage at 2am
Ideal TTL: 3 hours

Repeatedly fetch asset every second

Track when content changes



Caching correctly is difficult

As TTLs are hard to choose

Developers need to set TTLs precisely such that they expire exactly when the content changes

wsj.com homepage at 9am
Ideal TTL: 30 seconds

wsj.com homepage at 2am
Ideal TTL: 3 hours

Repeatedly fetch asset every second

Track when content changes

Ideal TTLs
{1, 2, ..}

Caching correctly is difficult

As TTLs are hard to choose

Developers need to set TTLs precisely such that they expire exactly when the content changes

wsj.com homepage at 9am
Ideal TTL: 30 seconds

wsj.com homepage at 2am
Ideal TTL: 3 hours

Repeatedly fetch asset every second

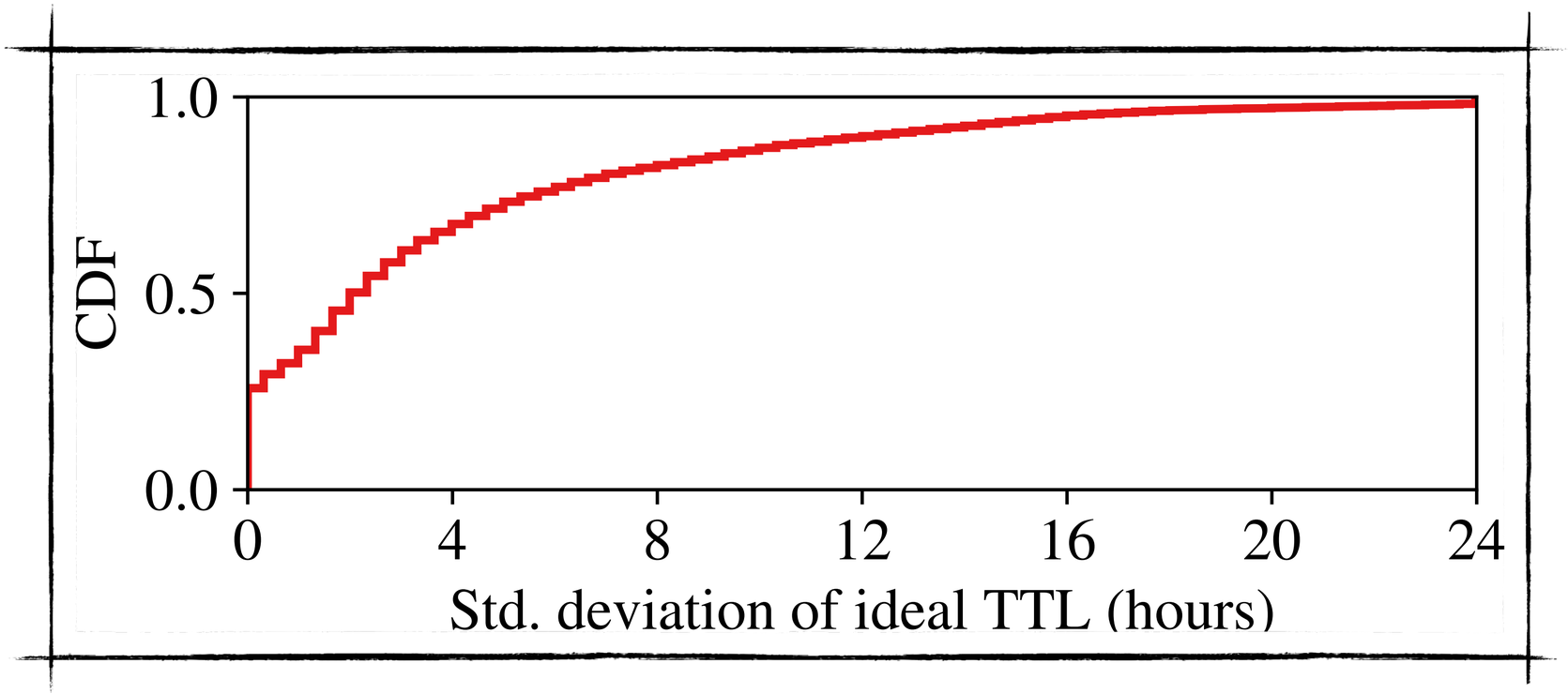
Track when content changes

Ideal TTLs
{1, 2, ..}

Let’s plot the std. dev.

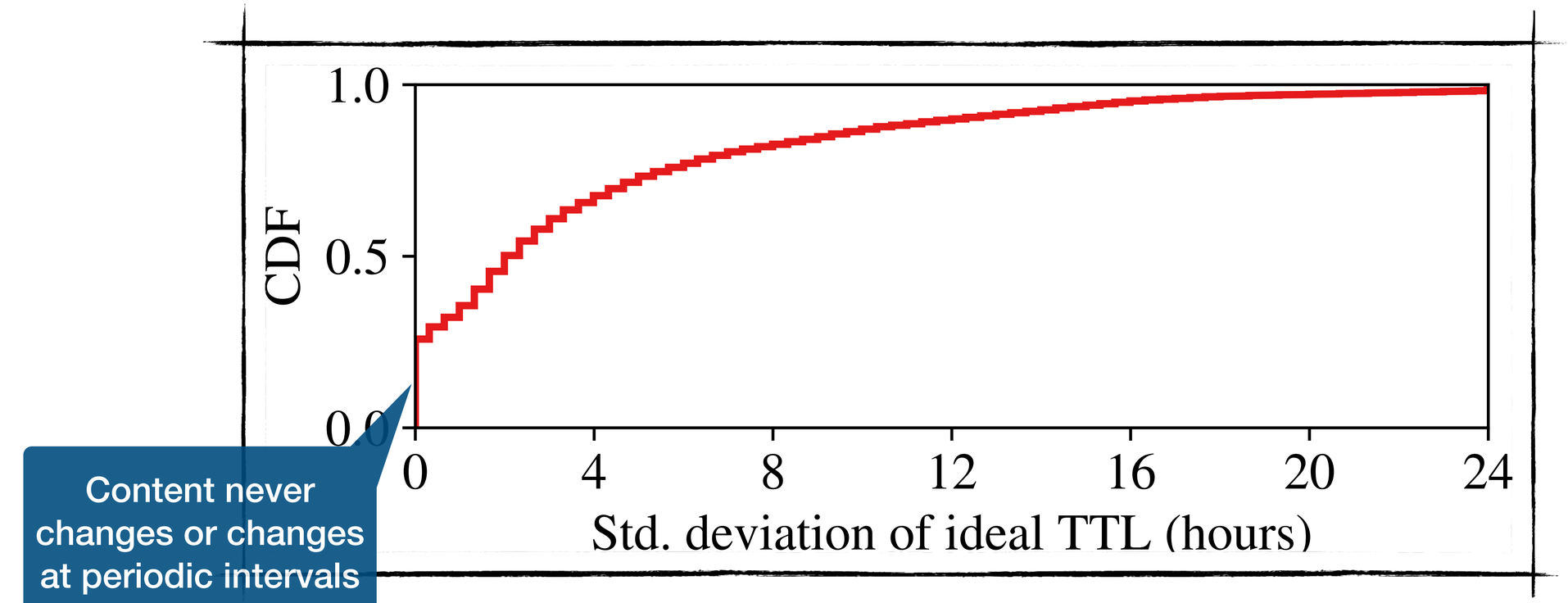
Caching correctly is difficult

Variations in ideal TTL

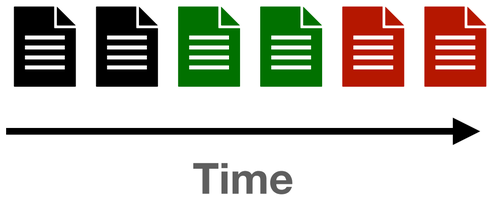


Caching correctly is difficult

Variations in ideal TTL

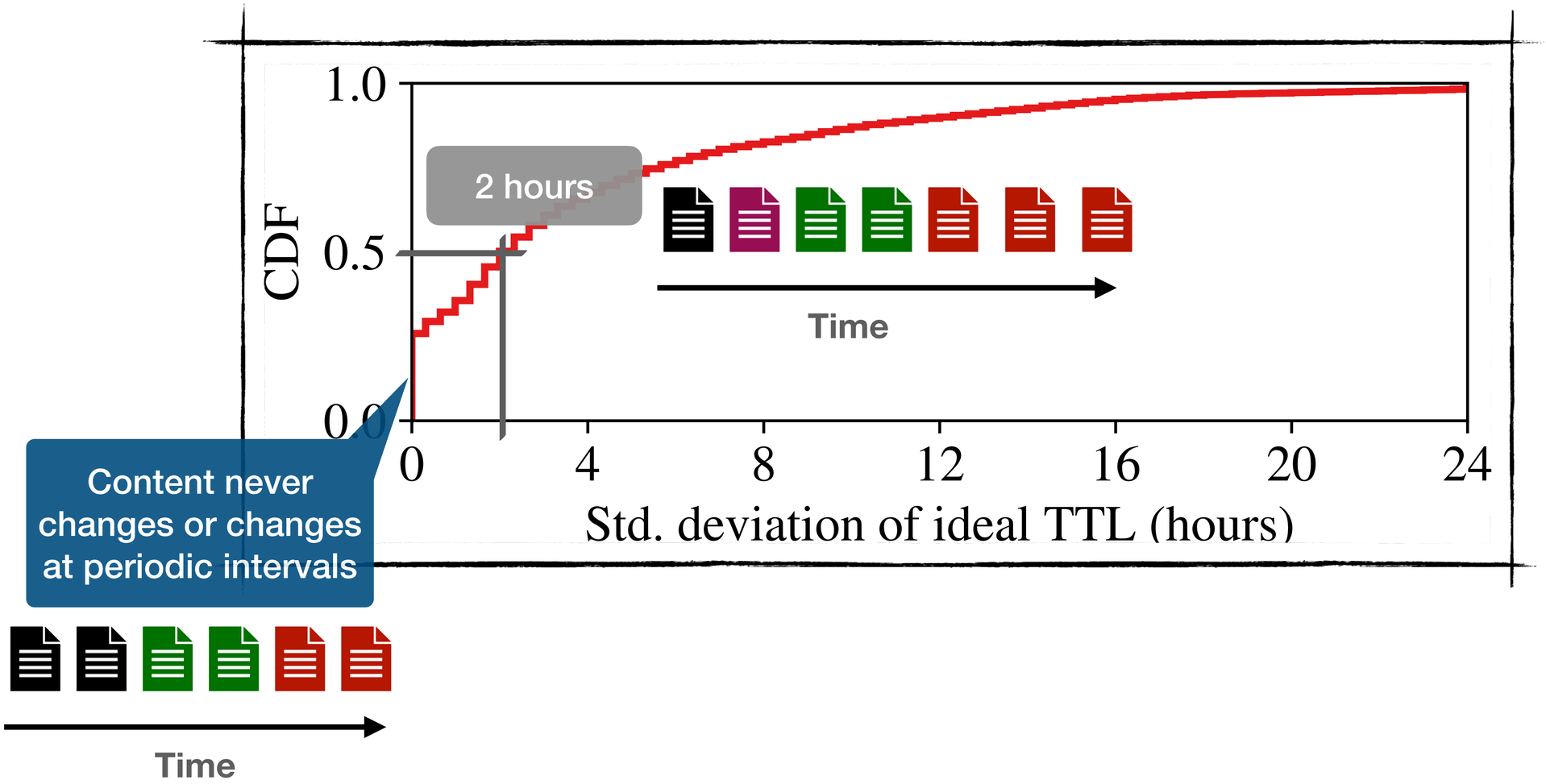


Content never changes or changes at periodic intervals



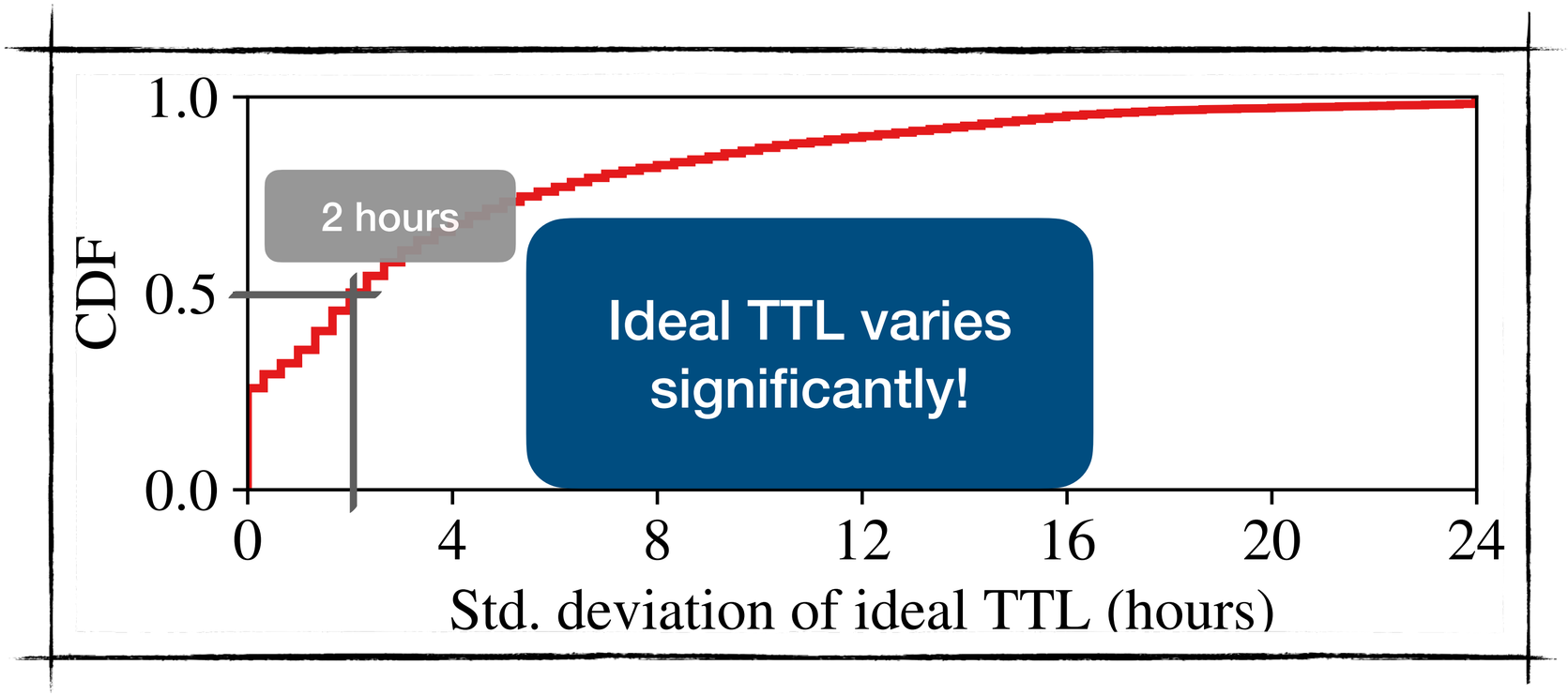
Caching correctly is difficult

Variations in ideal TTL



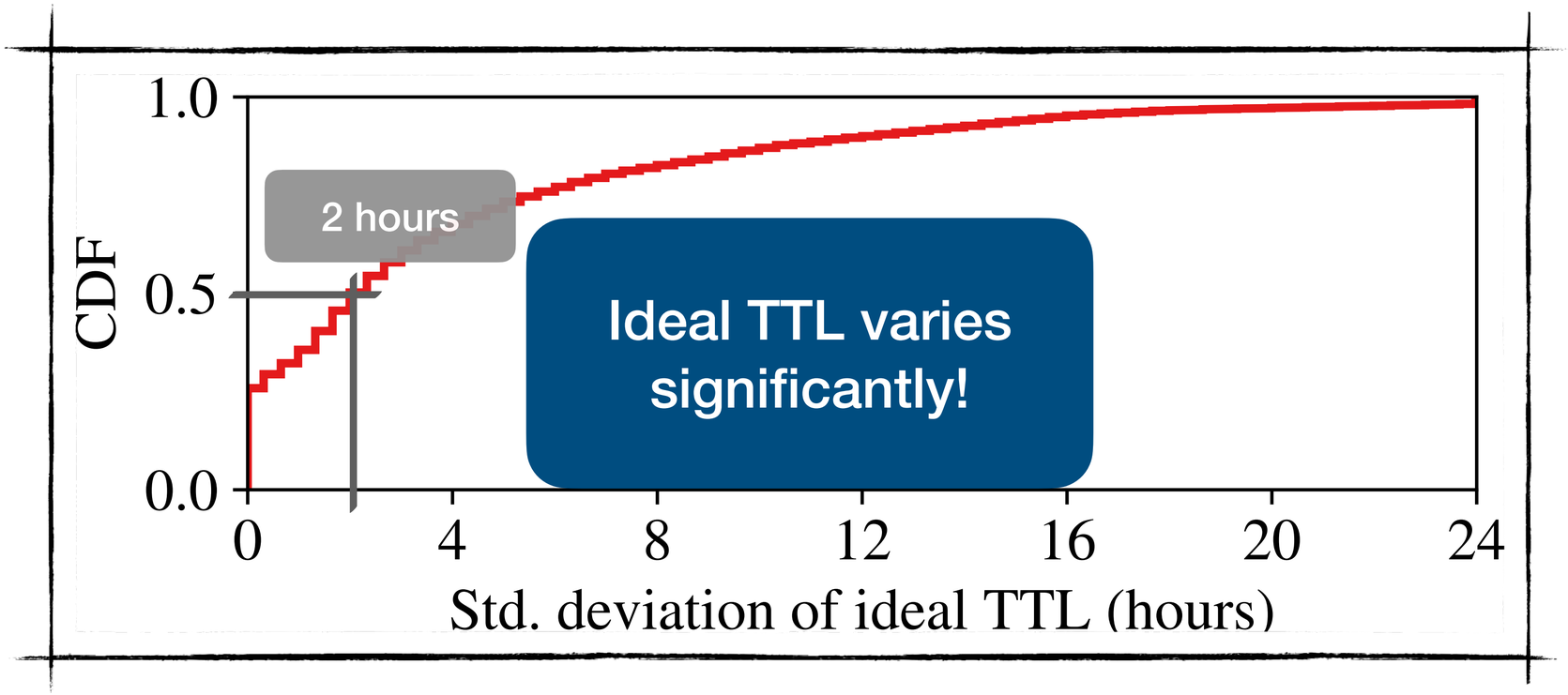
Caching correctly is difficult

Variations in ideal TTL



Caching correctly is difficult

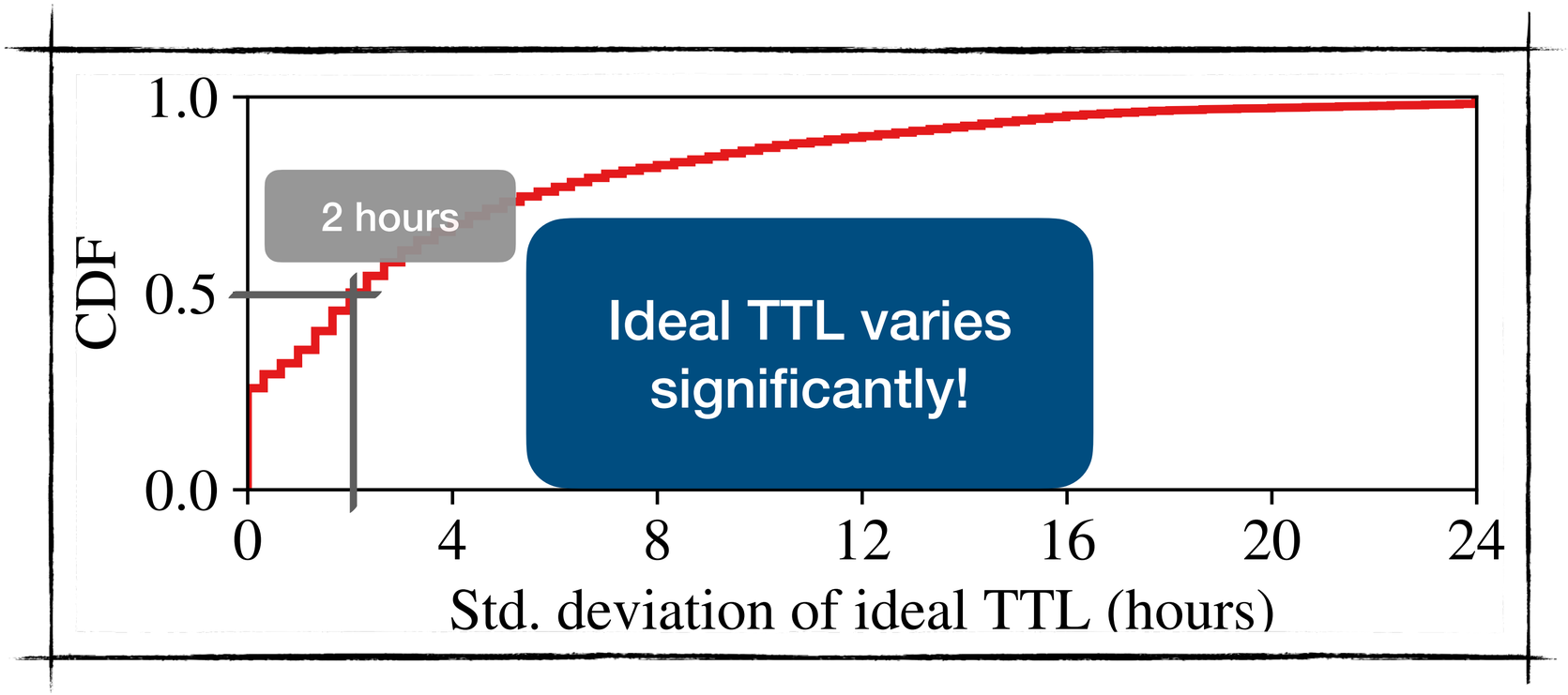
Variations in ideal TTL



High TTLs → good cache performance → stale content

Caching correctly is difficult

Variations in ideal TTL

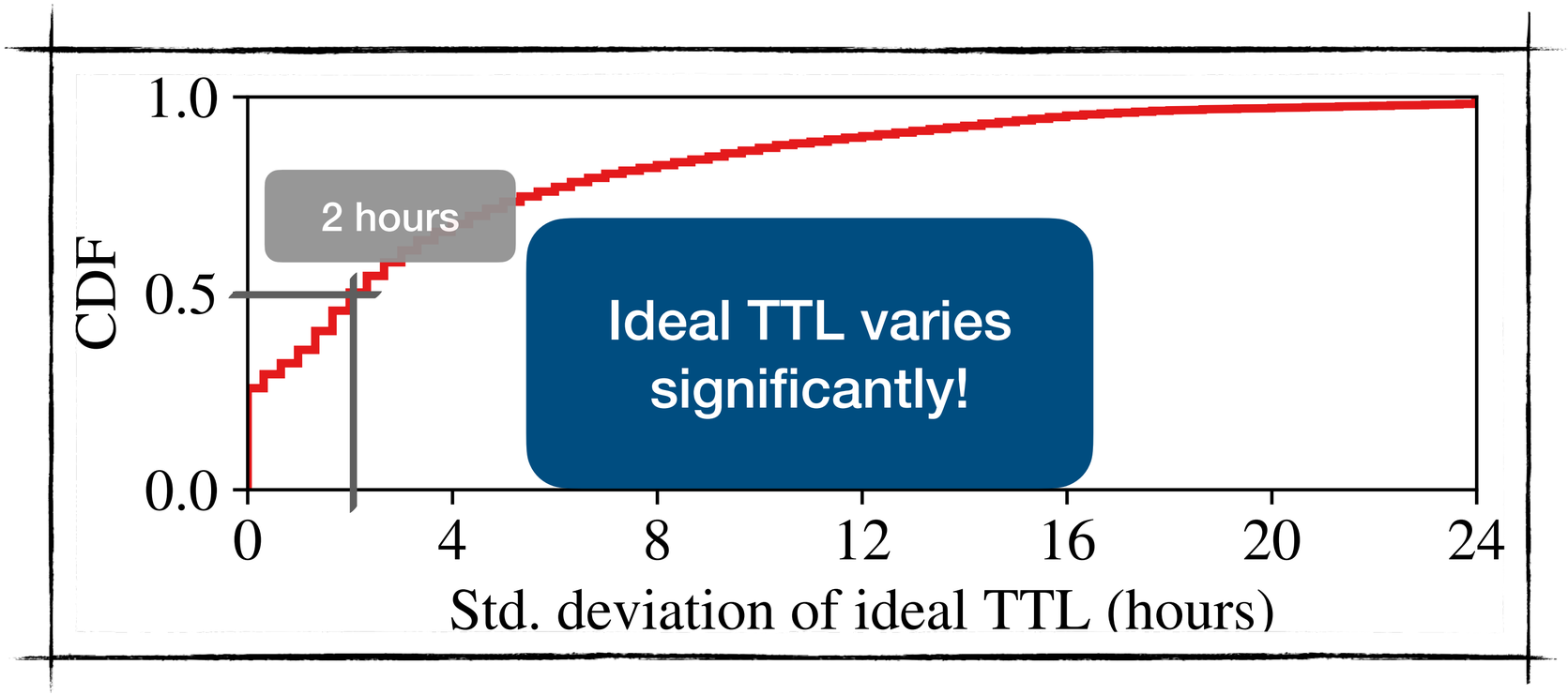


High TTLs → good cache performance → stale content

Low TTLs → fresh content → poor cache performance

Caching correctly is difficult

Variations in ideal TTL

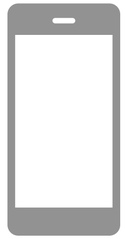


High TTLs → good cache performance → stale content

Low TTLs → fresh content → poor cache performance

Making caching better

Anatomy of an app interaction



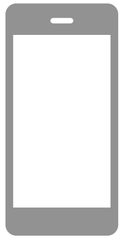
Origin server

Making caching better

Anatomy of an app interaction

Task: Loading a news article

Actions by user after opening app



Origin server

Making caching better

Anatomy of an app interaction

Task: Loading a news article

Actions by user after opening app

Tap on “Entertainment News” category

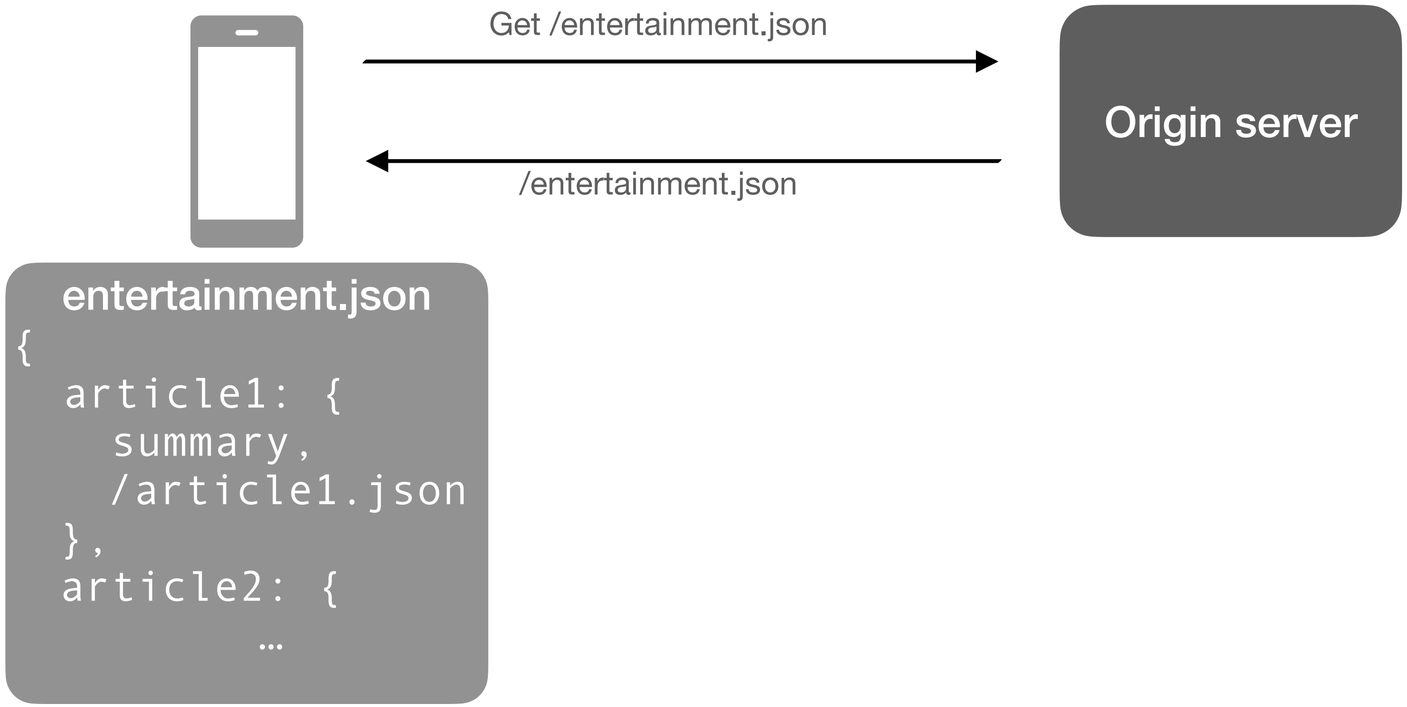
Making caching better

Anatomy of an app interaction

Task: Loading a news article

Actions by user after opening app

Tap on “Entertainment News” category



Making caching better

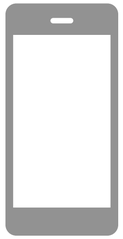
Anatomy of an app interaction

Task: Loading a news article

Actions by user after opening app

Tap on “Entertainment News” category

Tap on news article



entertainment.json

```
{
  article1: {
    summary,
    /article1.json
  },
  article2: {
          …
```

Origin server

Making caching better

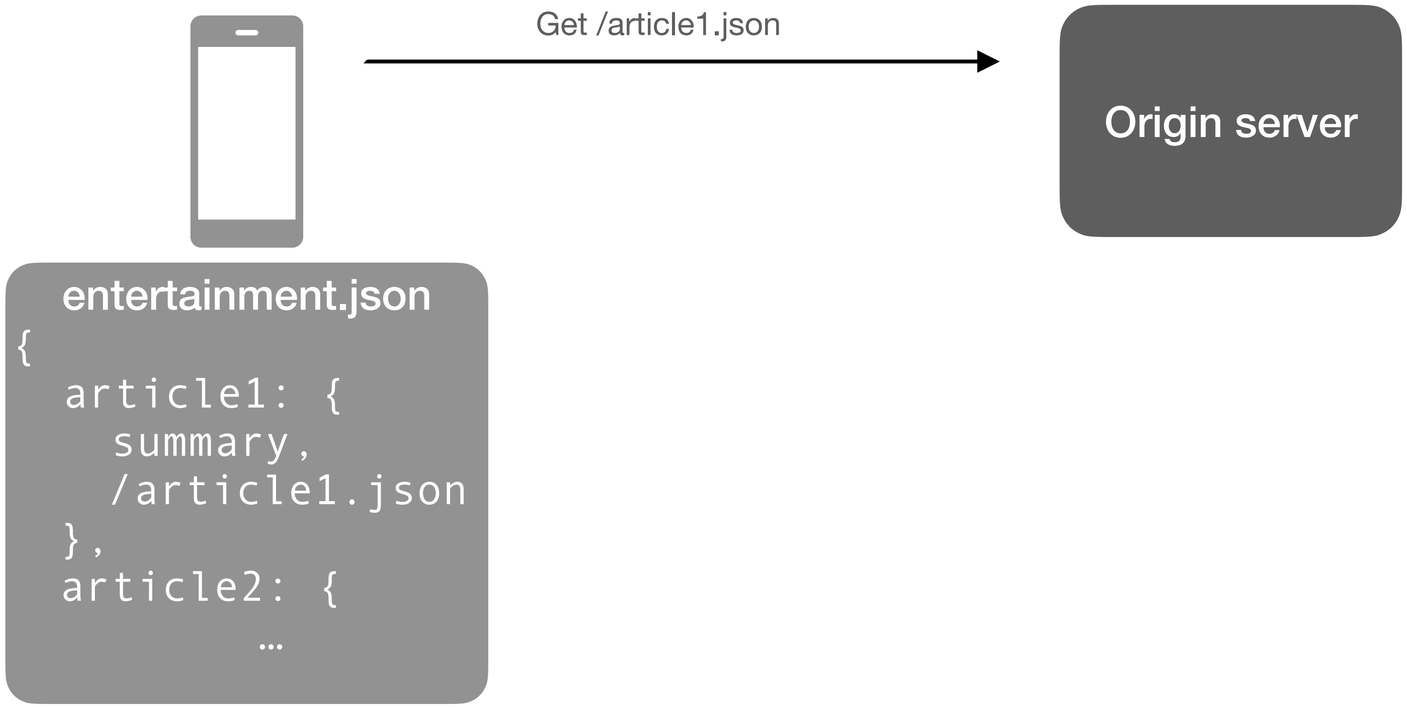
Anatomy of an app interaction

Task: Loading a news article

Actions by user after opening app

Tap on “Entertainment News” category

Tap on news article



Making caching better

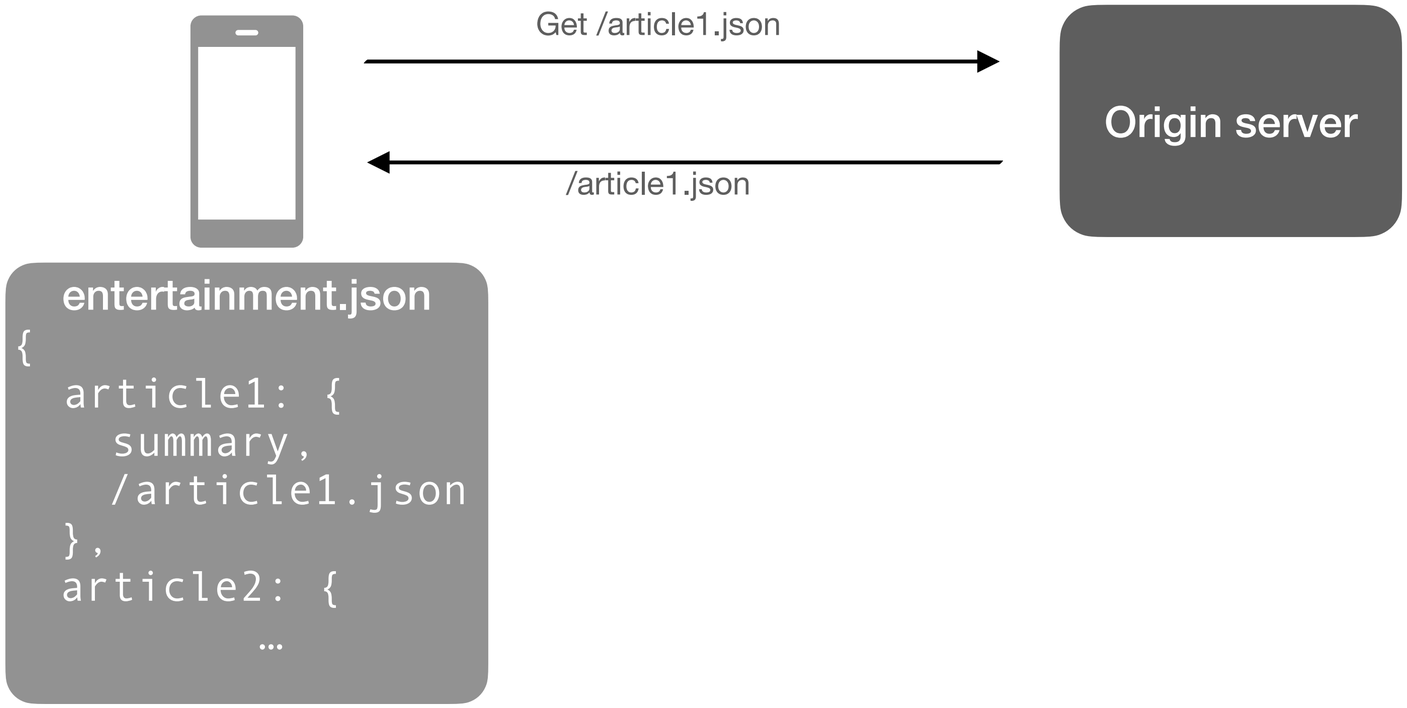
Anatomy of an app interaction

Task: Loading a news article

Actions by user after opening app

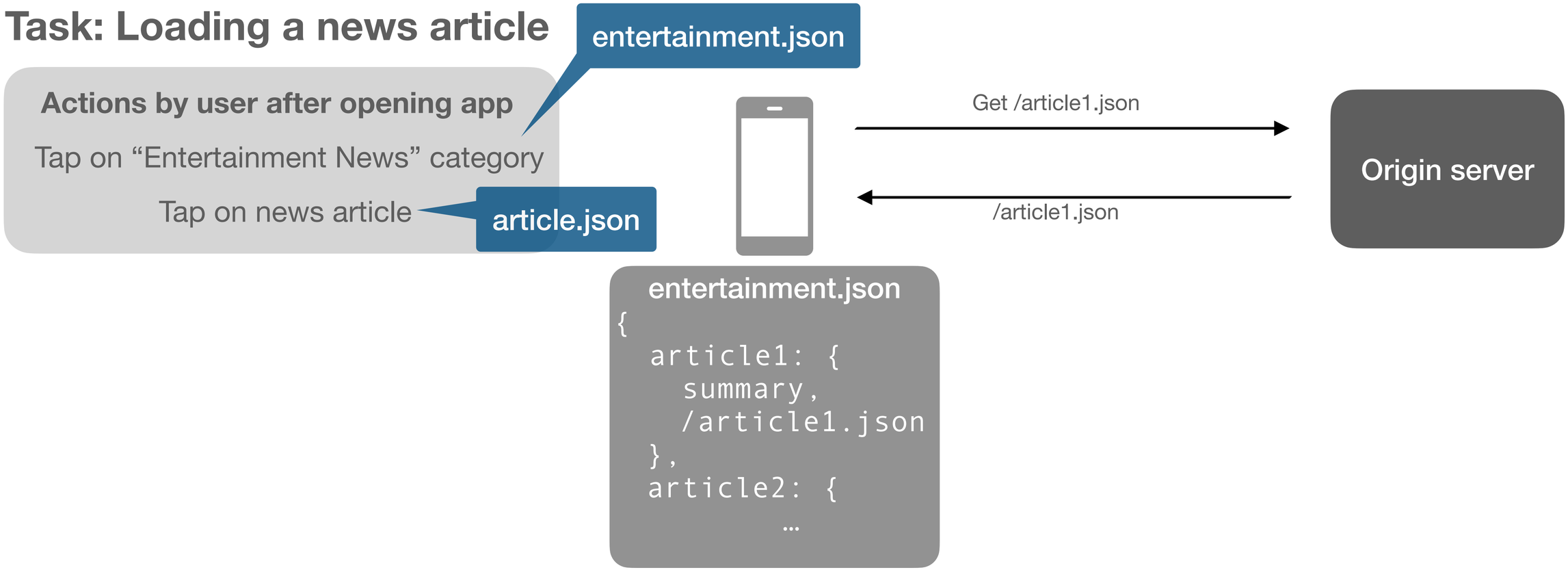
Tap on “Entertainment News” category

Tap on news article



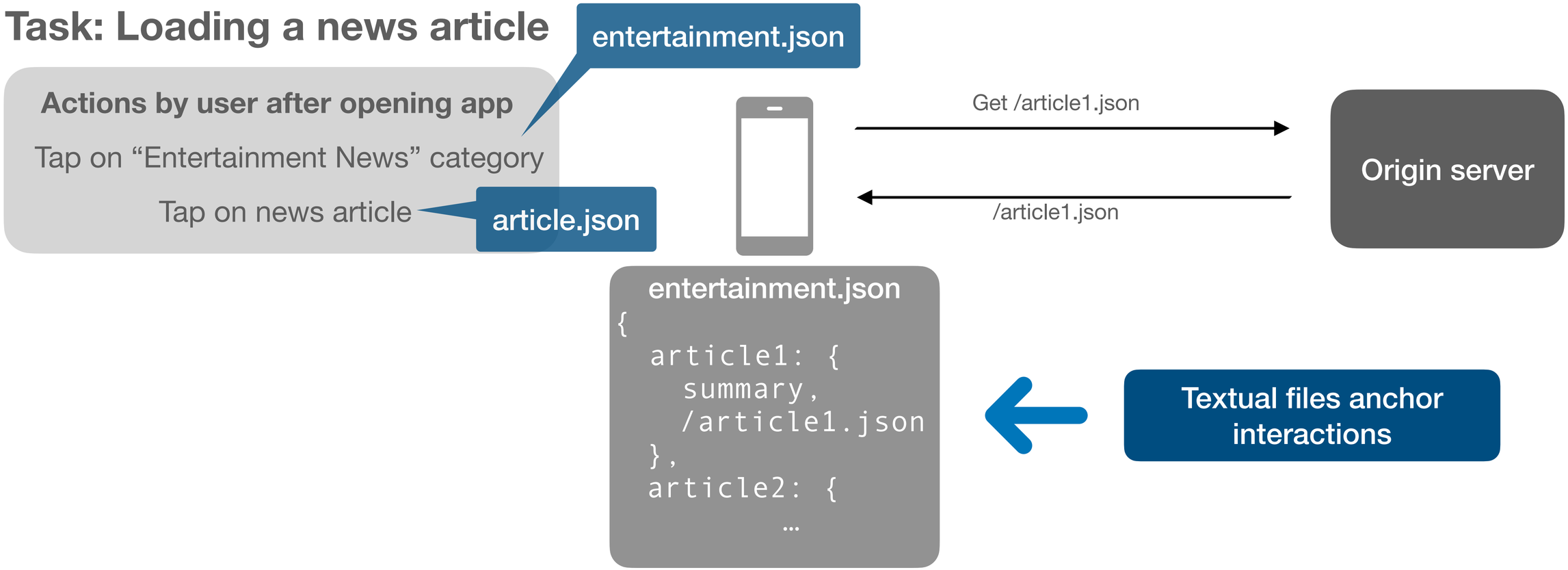
Making caching better

Anatomy of an app interaction



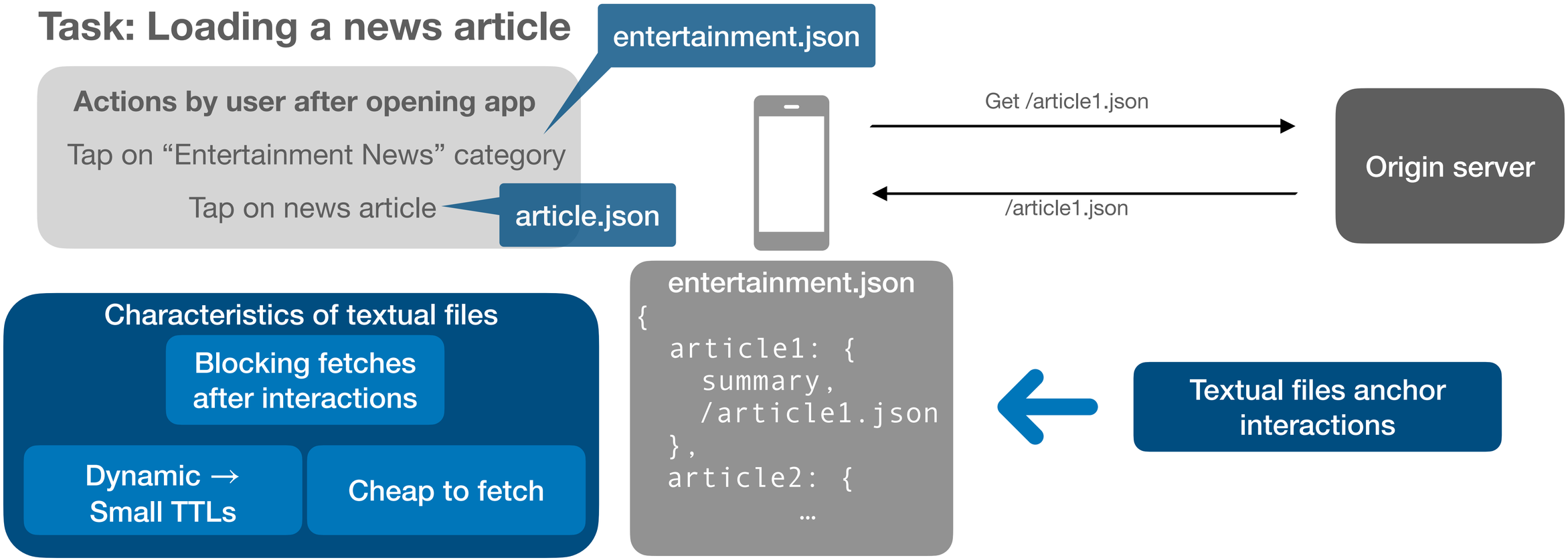
Making caching better

Anatomy of an app interaction



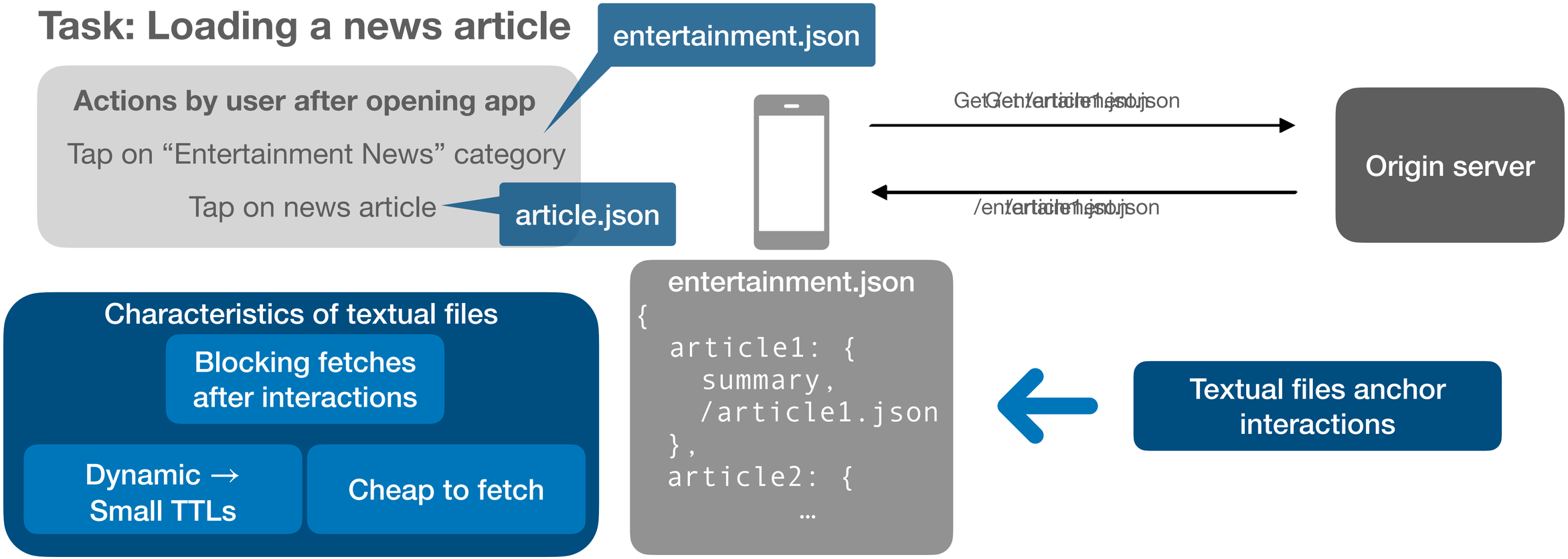
Making caching better

Anatomy of an app interaction



Making caching better

Anatomy of an app interaction



Making caching better

Insights about textual files

Blocking fetches after interactions

Dynamic → Small TTLs

Cheap to fetch

Making caching better

Insights about textual files

Blocking fetches after interactions

Cache hits are impactful

Dynamic → Small TTLs

Cheap to fetch

Making caching better

Insights about textual files

Blocking fetches after interactions

Cache hits are impactful

Dynamic → Small TTLs

Cheap to fetch

Making caching better

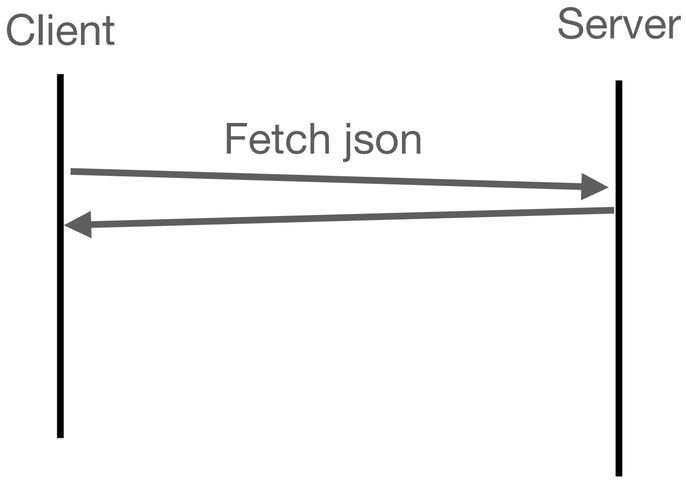
Insights about textual files

Blocking fetches after interactions

Cache hits are impactful

Dynamic → Small TTLs

Cheap to fetch



Making caching better

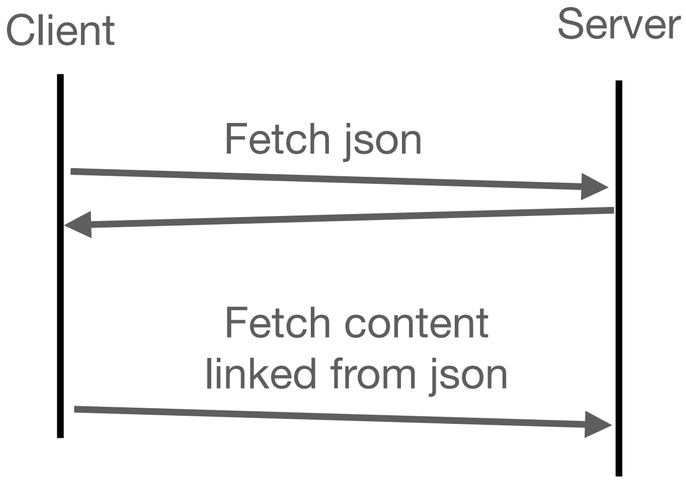
Insights about textual files

Blocking fetches after interactions

Cache hits are impactful

Dynamic → Small TTLs

Cheap to fetch



Making caching better

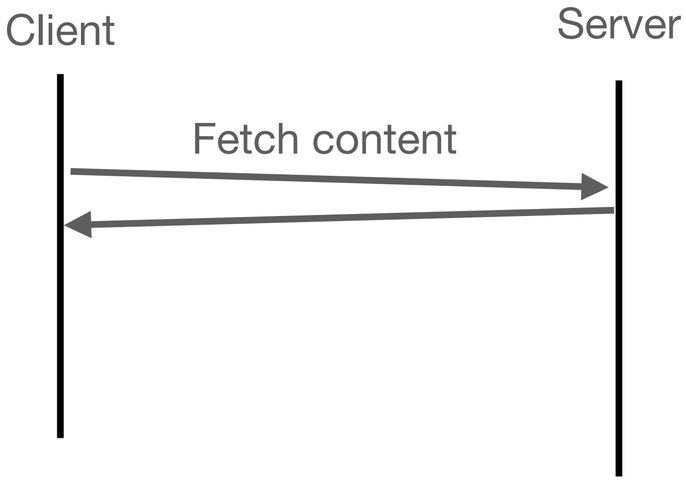
Insights about textual files

Blocking fetches after interactions

Cache hits are impactful

Dynamic → Small TTLs

Cheap to fetch



Making caching better

Insights about textual files

Blocking fetches after interactions

Cache hits are impactful

Dynamic → Small TTLs

Cheap to fetch

Making caching better

Insights about textual files

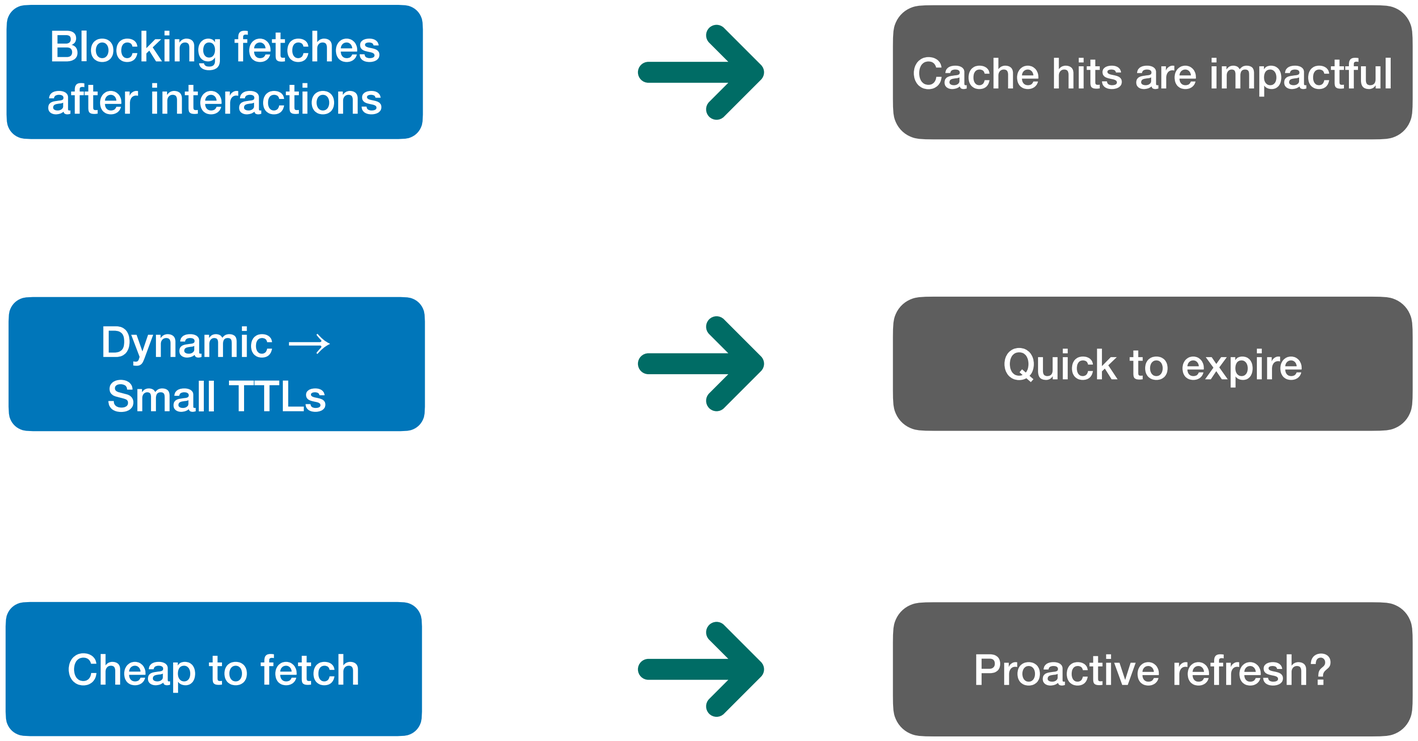
Blocking fetches after interactions → Cache hits are impactful

Dynamic → Small TTLs → Quick to expire

Cheap to fetch

Making caching better

Insights about textual files



Making caching better

Insights about textual files

Blocking fetches after interactions → Cache hits are impactful

Dynamic → Small TTLs → Quick to expire

Cheap to fetch → Proactive refresh?

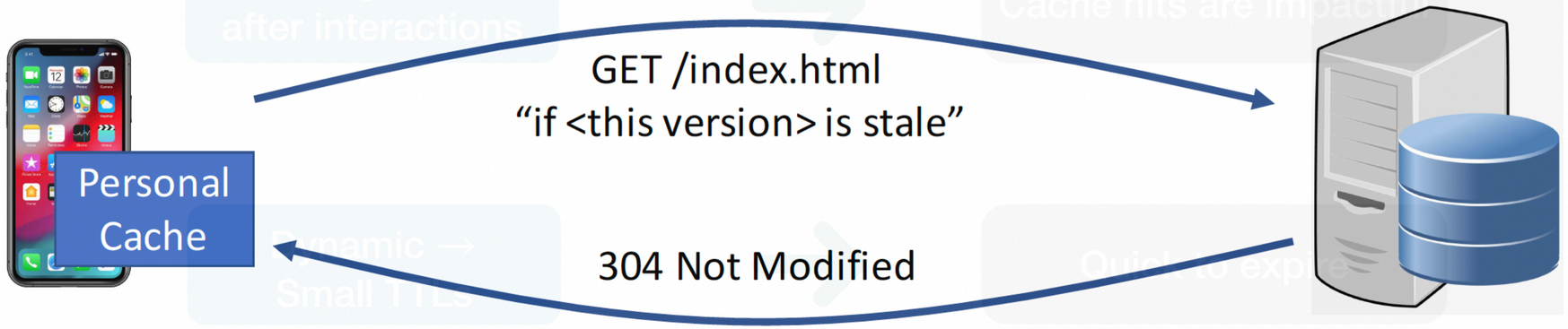
Conditional GET

Making caching better

Insights about textual files

From last lecture

Cache Validation: Client Checks Freshness



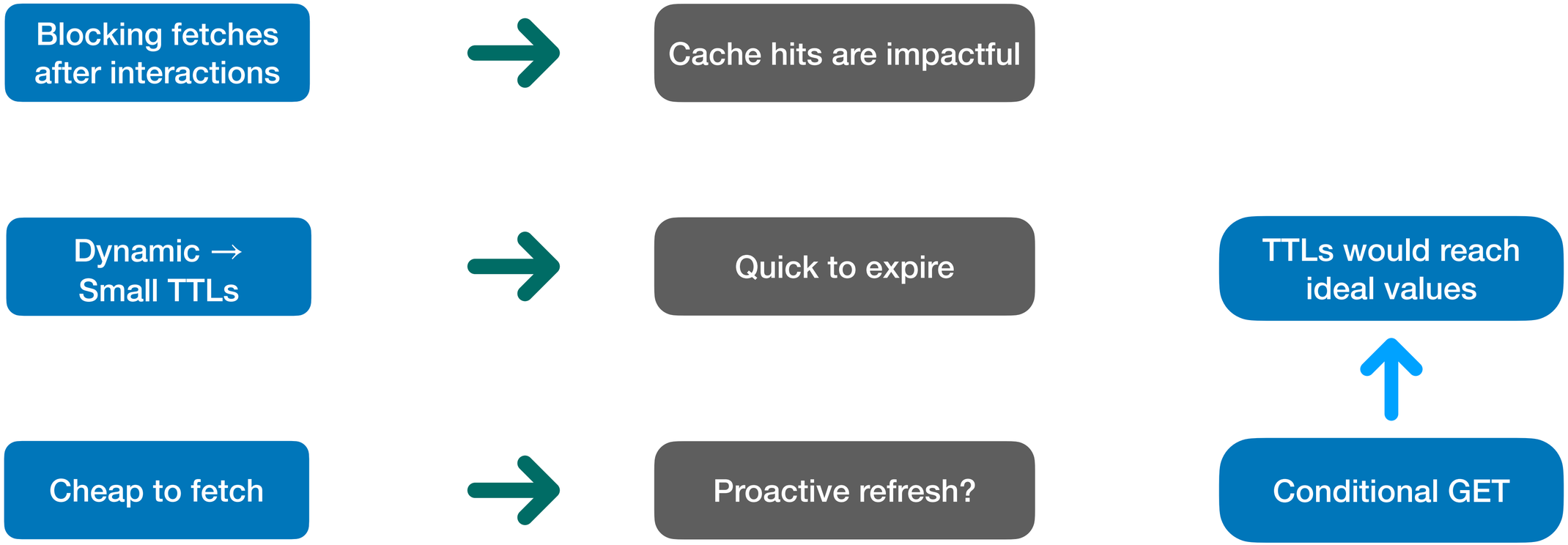
How do they identify the “version”?

- Timestamp
  - When the item was modified by the server
  - E.g., Last-Modified: Wed, 21 Oct 2015 07:28:00 GMT
- Version number
  - Entity tag provided by the server
  - E.g., ETag: "33a64df551425fcc55e4d42a148795d9f25f89d4"

Conditional GET

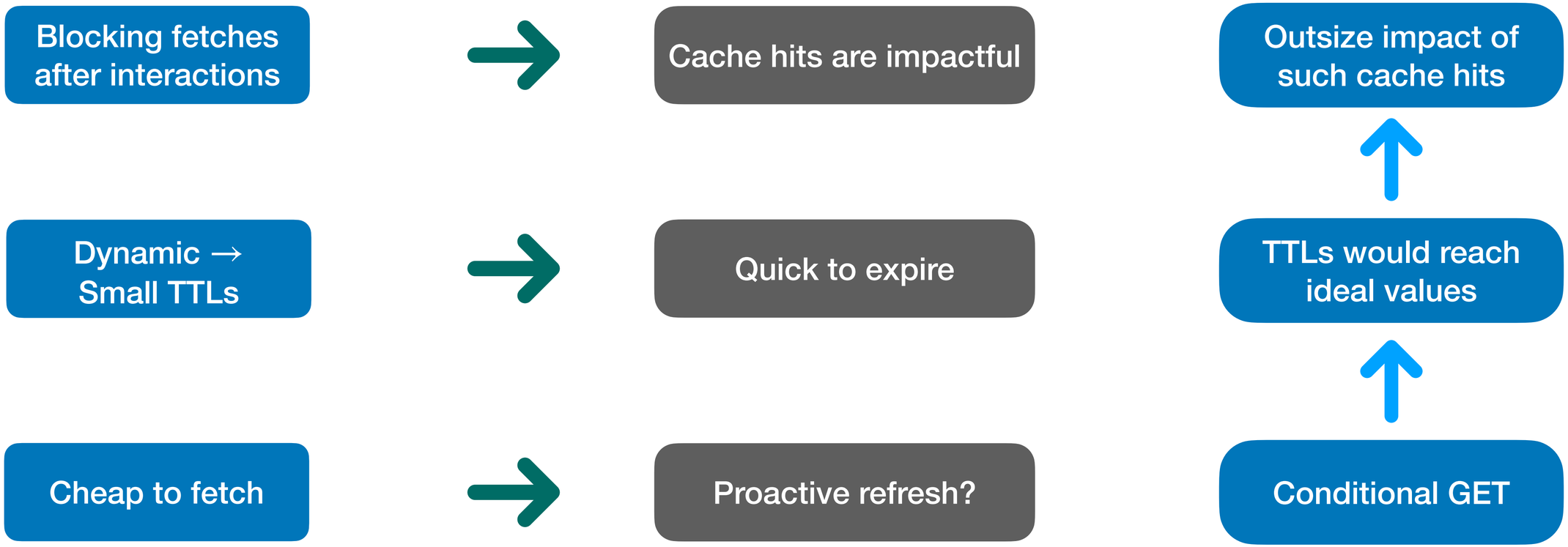
Making caching better

Insights about textual files



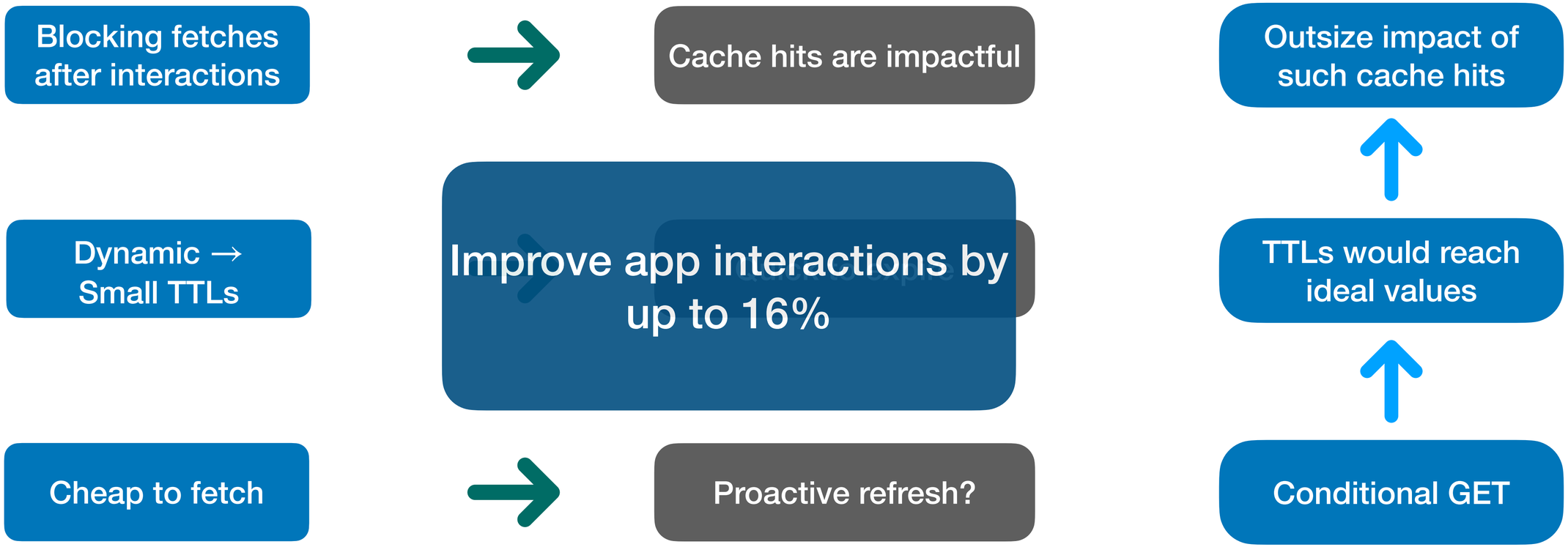
Making caching better

Insights about textual files



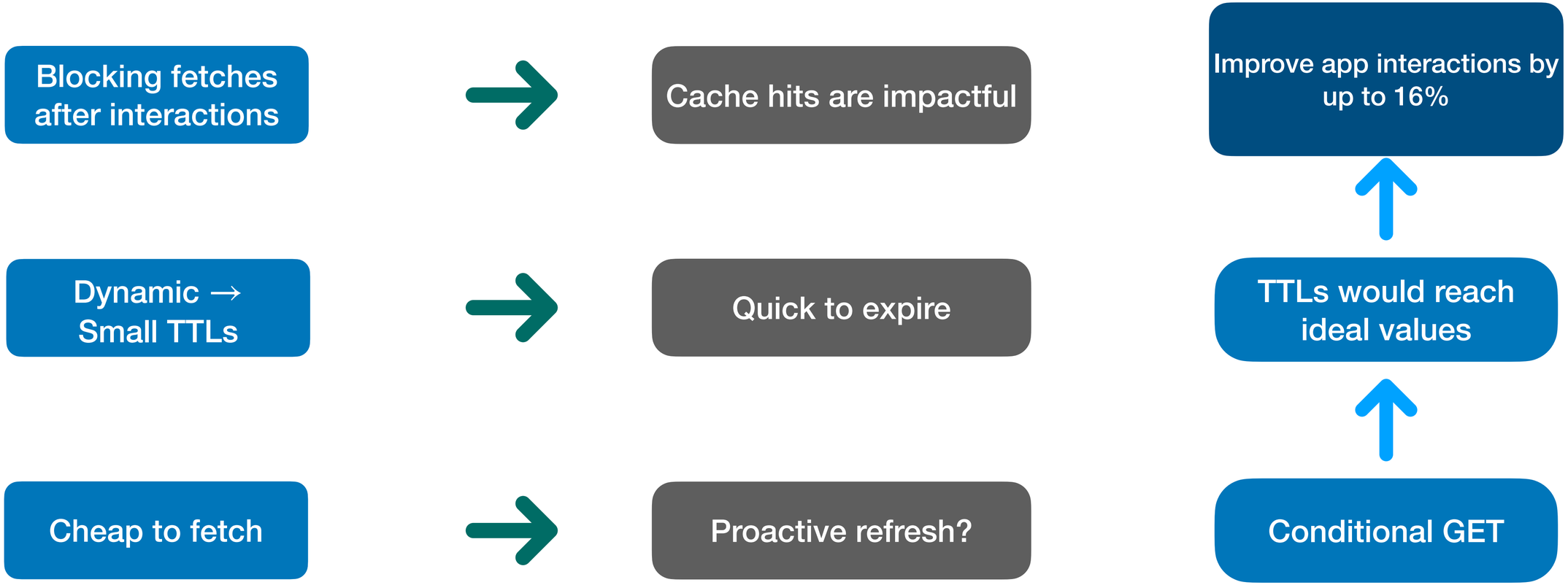
Making caching better

Insights about textual files



Making caching better

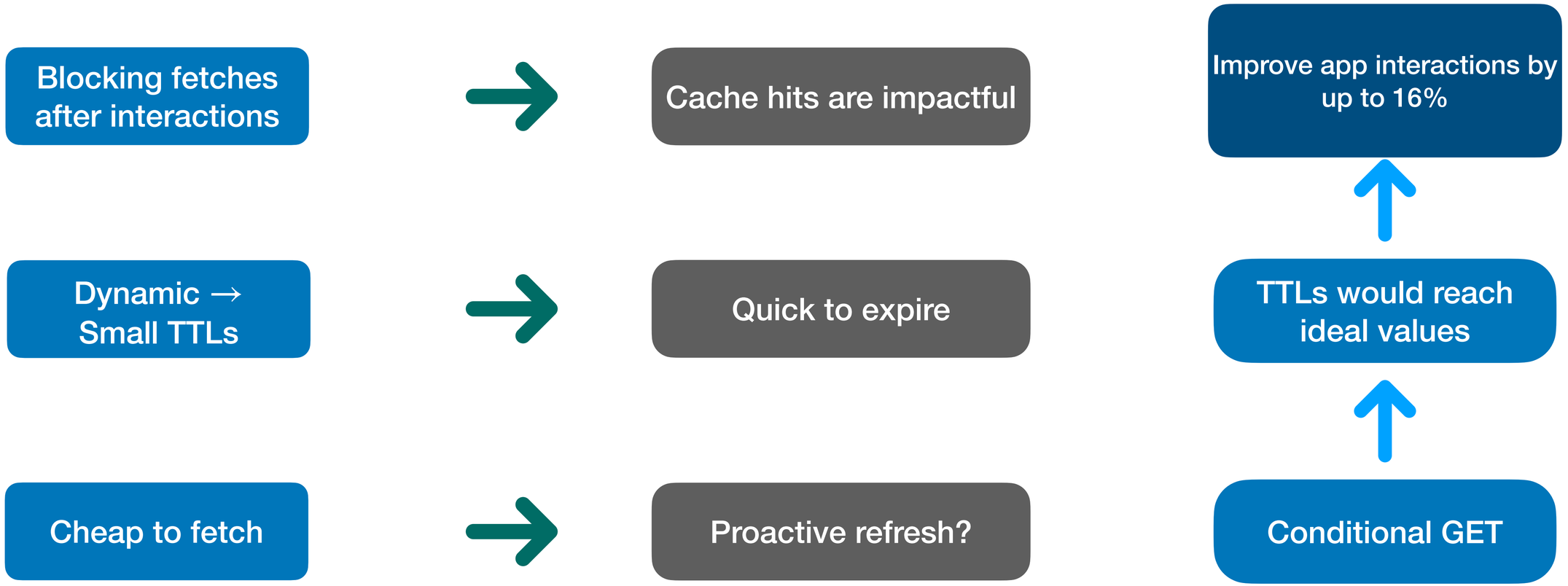
Insights about textual files



What about other assets in the cache, like images?

Making caching better

Insights about textual files



What about other assets in the cache, like images?

HEAD requests

A different way of caching

Why wait to observe requests?

Caches don’t help first (cold) loads

A different way of caching

Why wait to observe requests?

Caches don't help first (cold) loads

From last lecture

Client Machine (e.g., Browser)

Advantages

- Very low latency
- Preserves access bandwidth
- Available when disconnected

Disadvantages

- Low hit rate due to "cold" misses
- Many cache consistency checks
- Incomplete logs at the server

A different way of caching

Why wait to observe requests?

Caches don't help first (cold) loads

From last lecture

Client Machine (e.g., Browser)

Advantages

- Very low latency
- Preserves access bandwidth
- Available when disconnected

Disadvantages

- Low hit rate due to "cold" misses
- Many cache consistency checks
- Incomplete logs at the server

Personal Cache

A different way of caching

Why wait to observe requests?

Caches don’t help first (cold) loads

aka “*pre*fetching”

Could we “warm” the cache?

A different way of caching

Why wait to observe requests?

Caches don’t help first (cold) loads

Could we “warm” the cache?

aka “*pre*fetching”

Near-instantaneous response to users

A different way of caching

Why wait to observe requests?

Caches don’t help first (cold) loads

Could we “warm” the cache?

aka “*pre*fetching”

Near-instantaneous response to users

But what would the user want?

Prefetching correctly is difficult

Load content in the background at periodic intervals
Avoid network fetches during interactions

Prefetching correctly is difficult

Load content in the background at periodic intervals
Avoid network fetches during interactions

4 of top 50
store apps
prefetch

Prefetching correctly is difficult

Load content in the background at periodic intervals
Avoid network fetches during interactions

4 of top 50 store apps prefetch

Speedups above 60%

Prefetching correctly is difficult

Load content in the background at periodic intervals
Avoid network fetches during interactions

Speedups
above 60%

4 of top 50
store apps
prefetch

4x data usage

Prefetching correctly is difficult

Load content in the background at periodic intervals
Avoid network fetches during interactions

Speedups above 60%

4 of top 50 store apps prefetch

4x data usage

Difficult to predict users

Prefetching correctly is difficult

Load content in the background at periodic intervals
Avoid network fetches during interactions

Speedups above 60%

4 of top 50 store apps prefetch

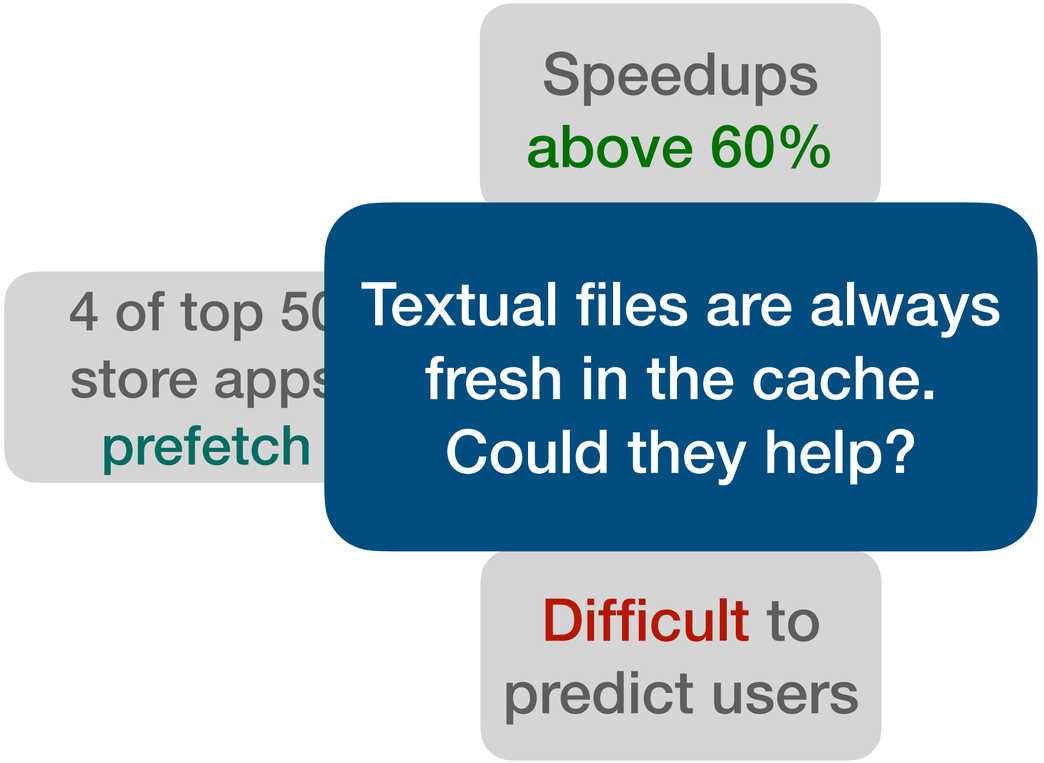
4x data usage

Difficult to predict users

Coupling of speedups and overheads

Prefetching correctly is difficult

Load content in the background at periodic intervals
Avoid network fetches during interactions



Coupling of speedups and overheads

Analyzing textual files

Opportunities for prefetching

Analyzing textual files

Opportunities for prefetching

entertainment.json

```
{
  article1: {
    summary,
    /article1.json
  },
  article2: {
          …
```

article1.json

```
{
  title: “Article”
  banner: “/x.jpg”,
  content: “..”
}
```

Analyzing textual files

Opportunities for prefetching

entertainment.json

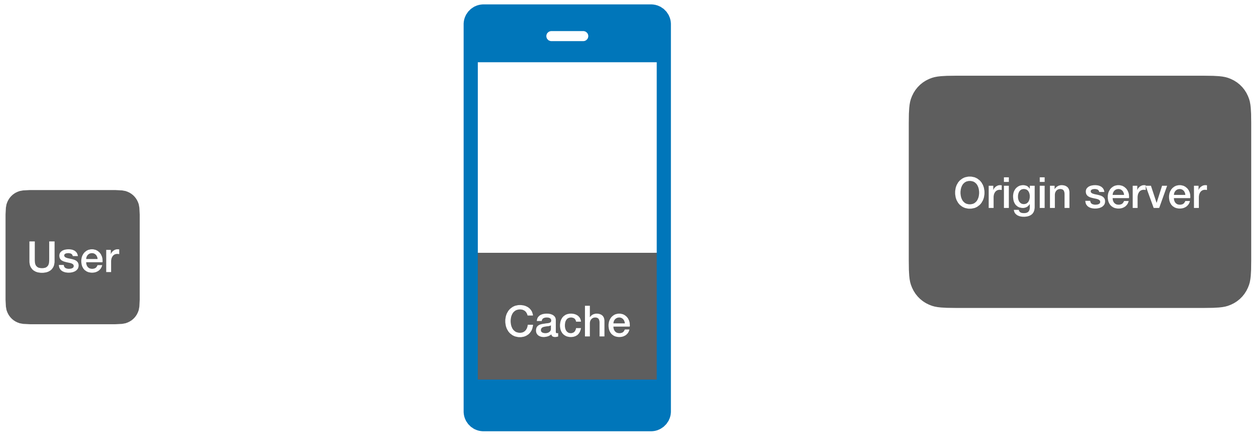
```
{
  article1: {
    summary,
    /article1.json
  },
  article2: {
            …
```

article1.json

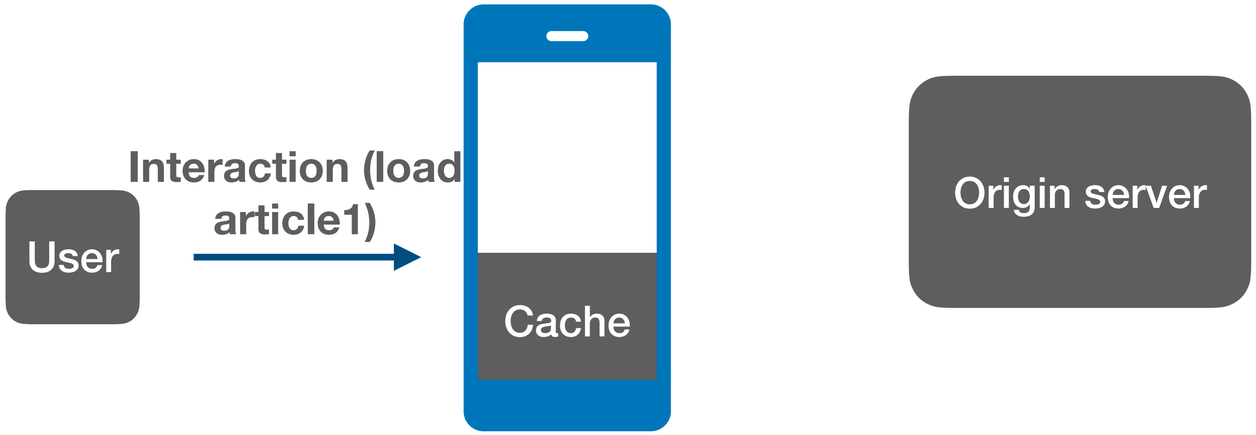
```
{
  title: “Article”
  banner: “/x.jpg”,
  content: “..”
}
```

Text files contain pointers to upcoming requests

Just-in-time prefetching

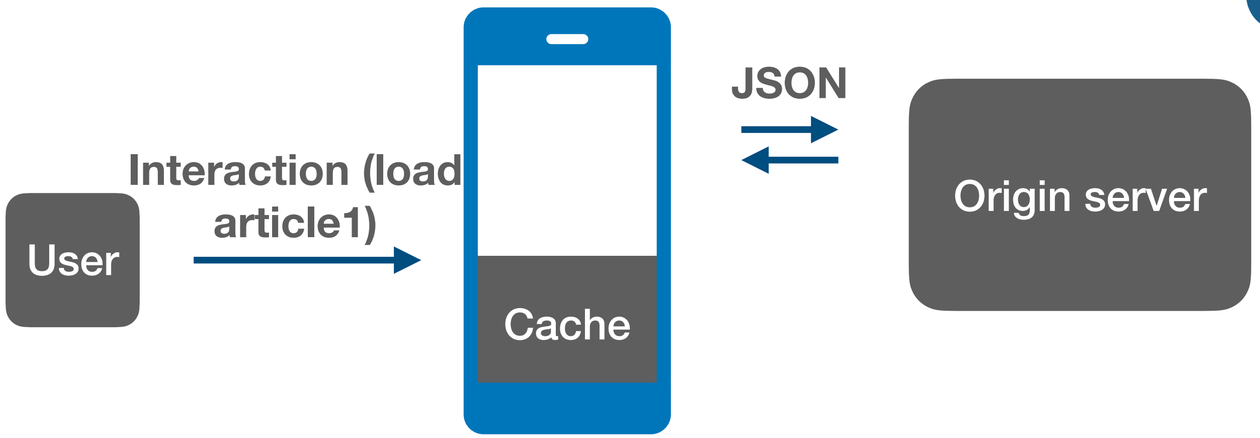


Just-in-time prefetching



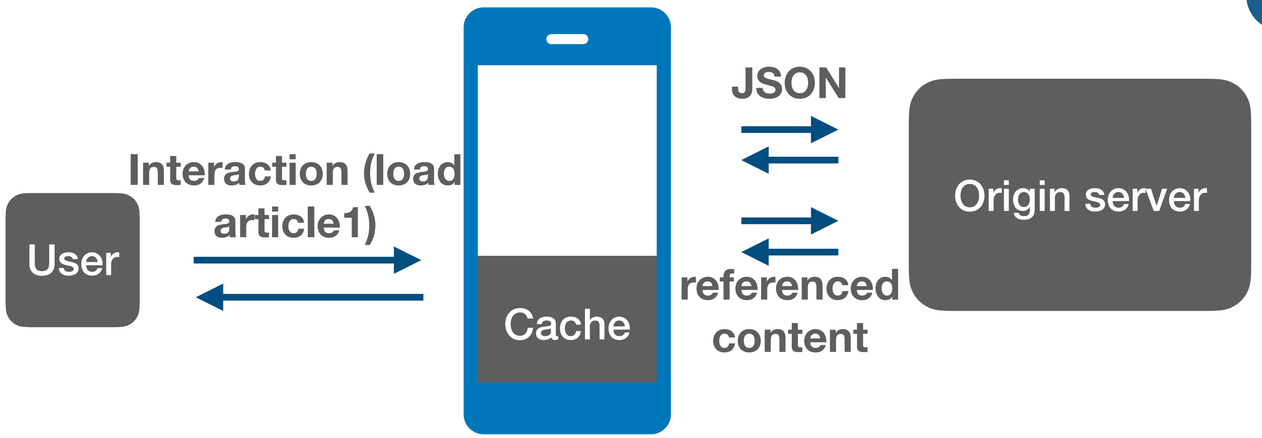
Just-in-time prefetching

JSON: article1.json
Referenced content: x.jpg



Just-in-time prefetching

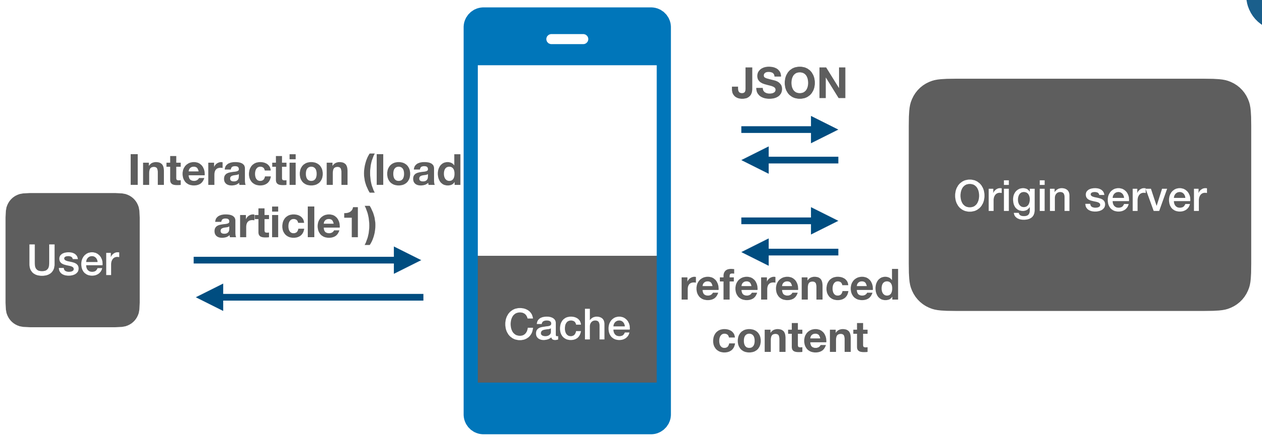
JSON: article1.json
Referenced content: x.jpg



Just-in-time prefetching

JSON: article1.json
Referenced content: x.jpg

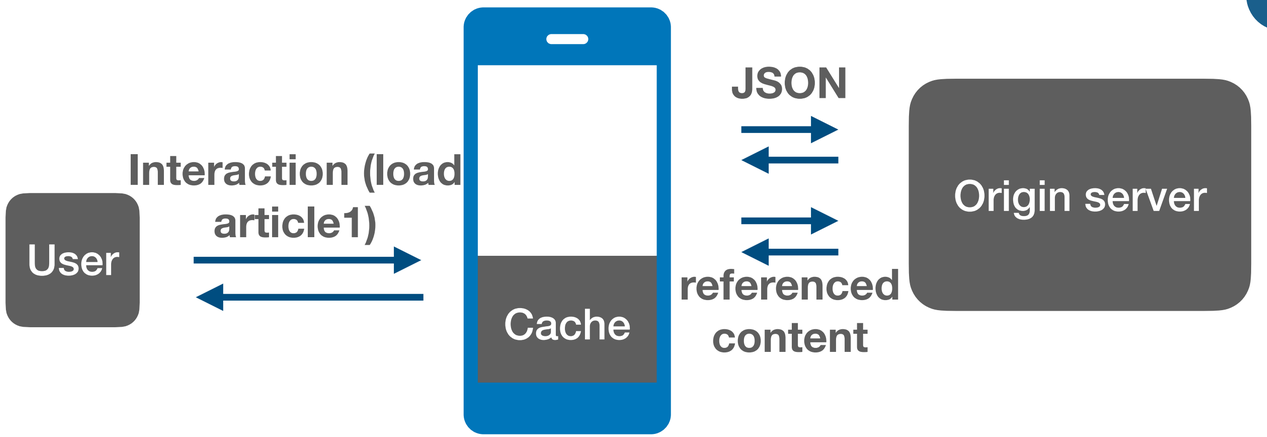
Normal Operation



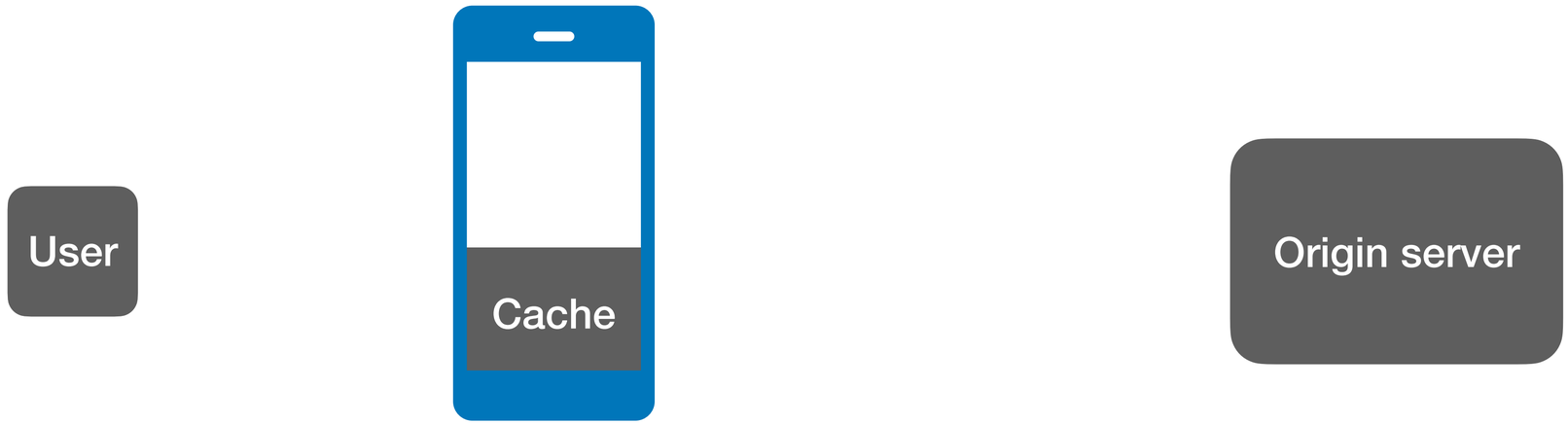
Just-in-time prefetching

JSON: article1.json
Referenced content: x.jpg

Normal Operation



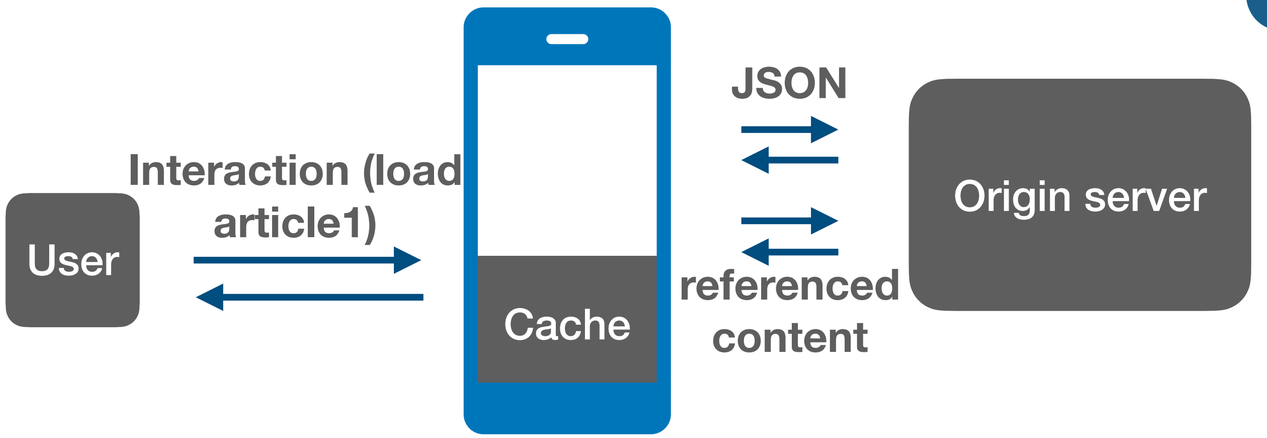
Optimization: Parse JSONs and determine referenced content in advance



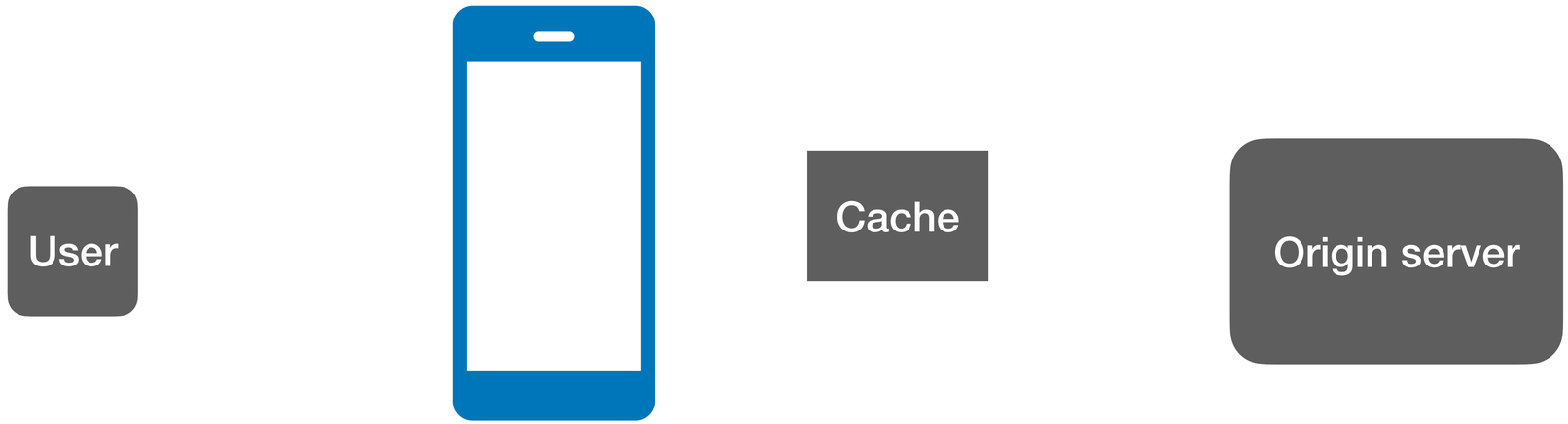
Just-in-time prefetching

JSON: article1.json
Referenced content: x.jpg

Normal Operation



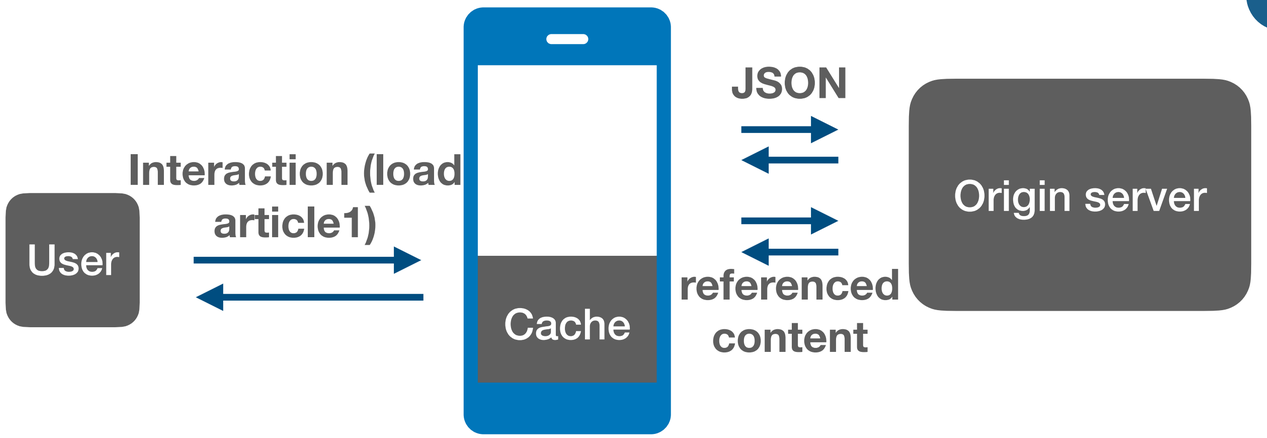
Optimization: Parse JSONs and determine referenced content in advance



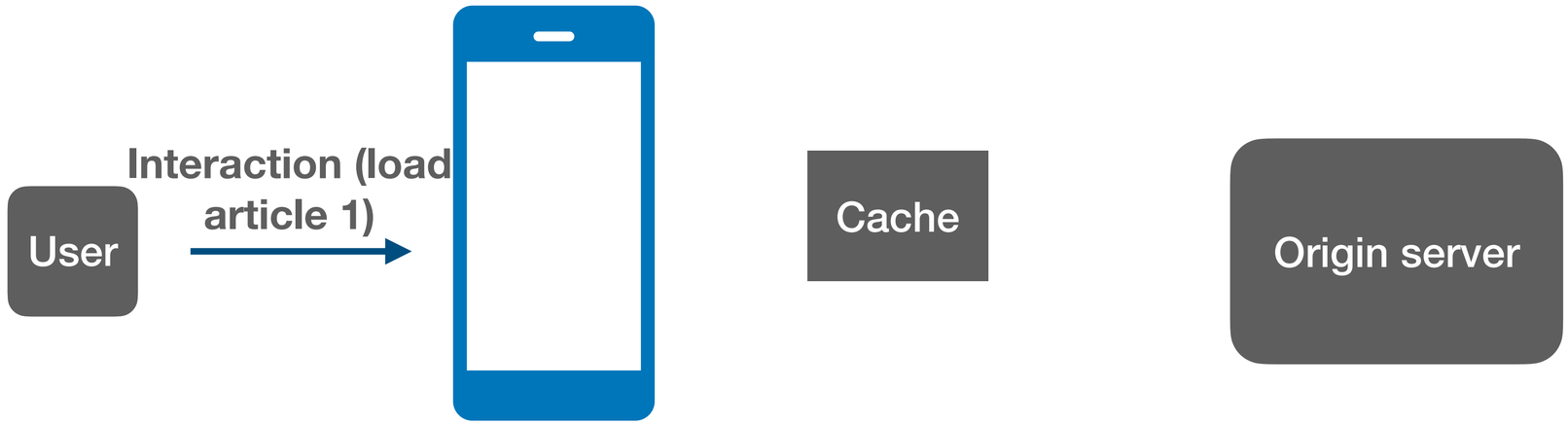
Just-in-time prefetching

JSON: article1.json
Referenced content: x.jpg

Normal Operation



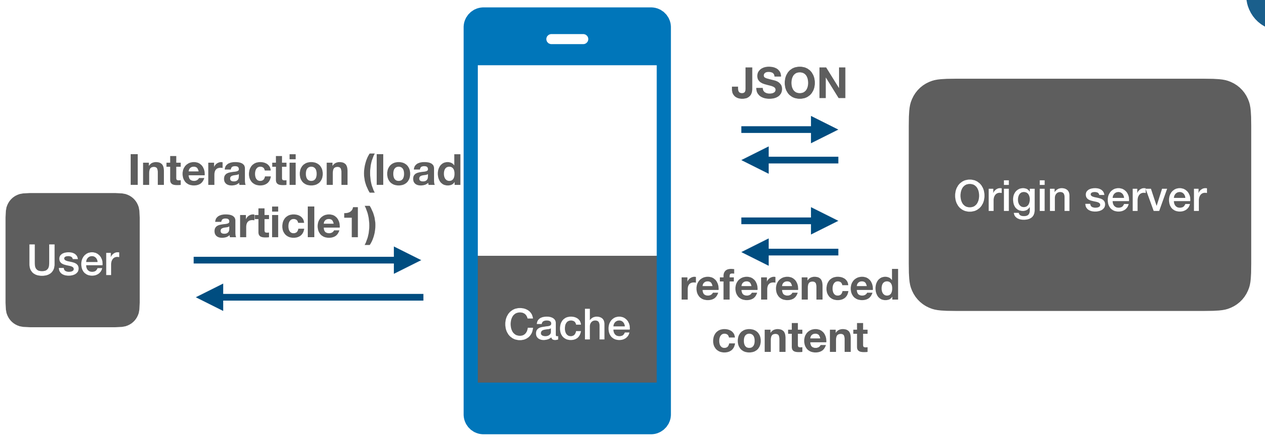
Optimization: Parse JSONs and determine referenced content in advance



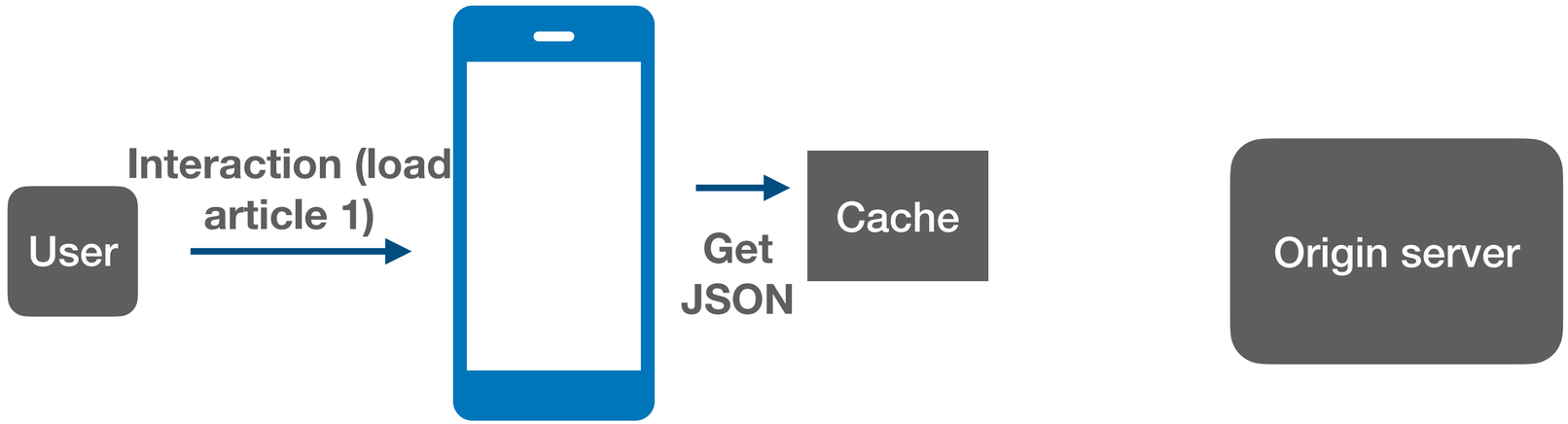
Just-in-time prefetching

JSON: article1.json
Referenced content: x.jpg

Normal Operation



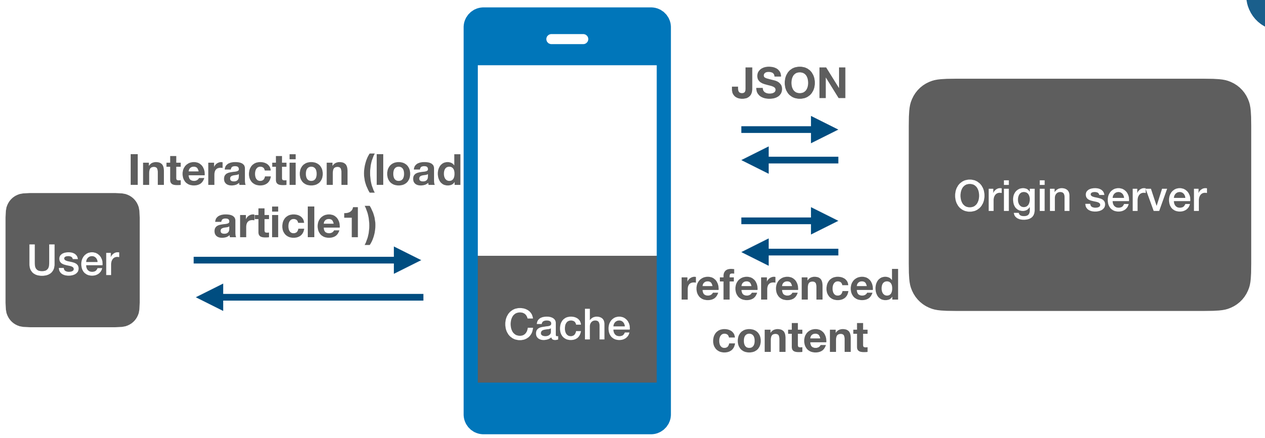
Optimization: Parse JSONs and determine referenced content in advance



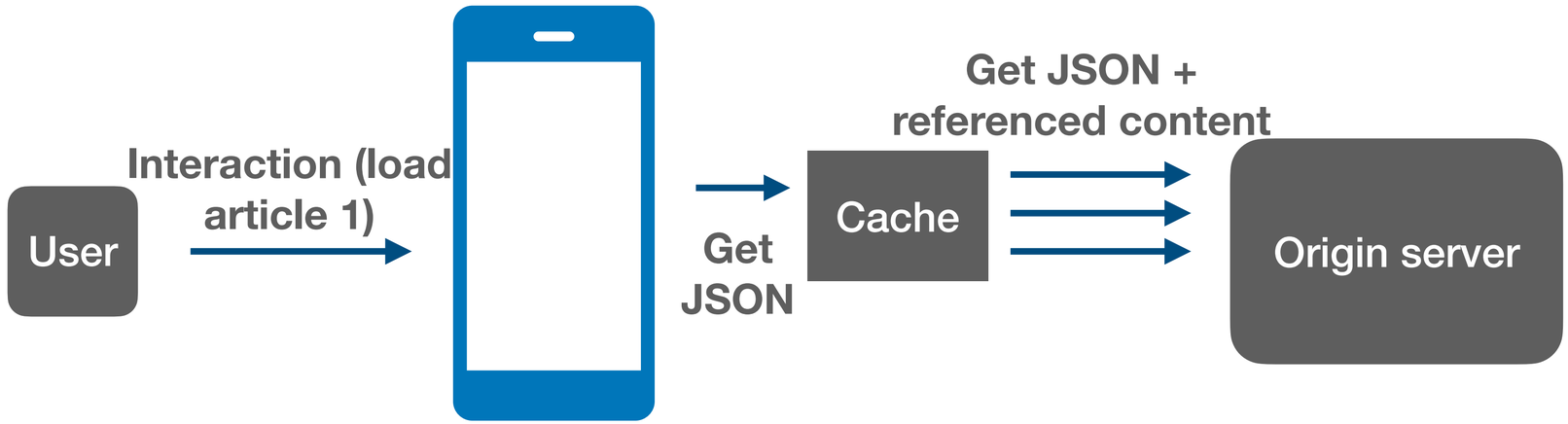
Just-in-time prefetching

JSON: article1.json
Referenced content: x.jpg

Normal Operation



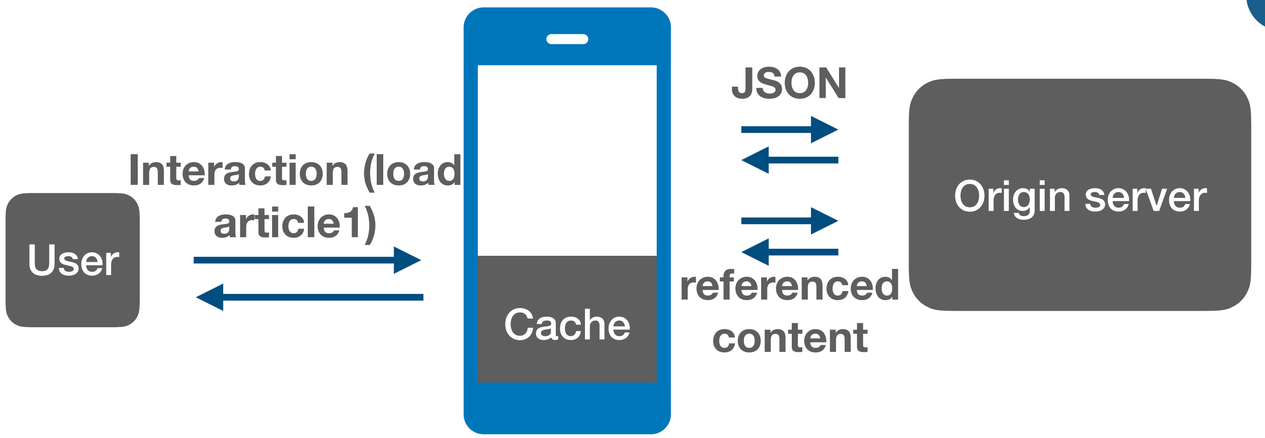
Optimization: Parse JSONs and determine referenced content in advance



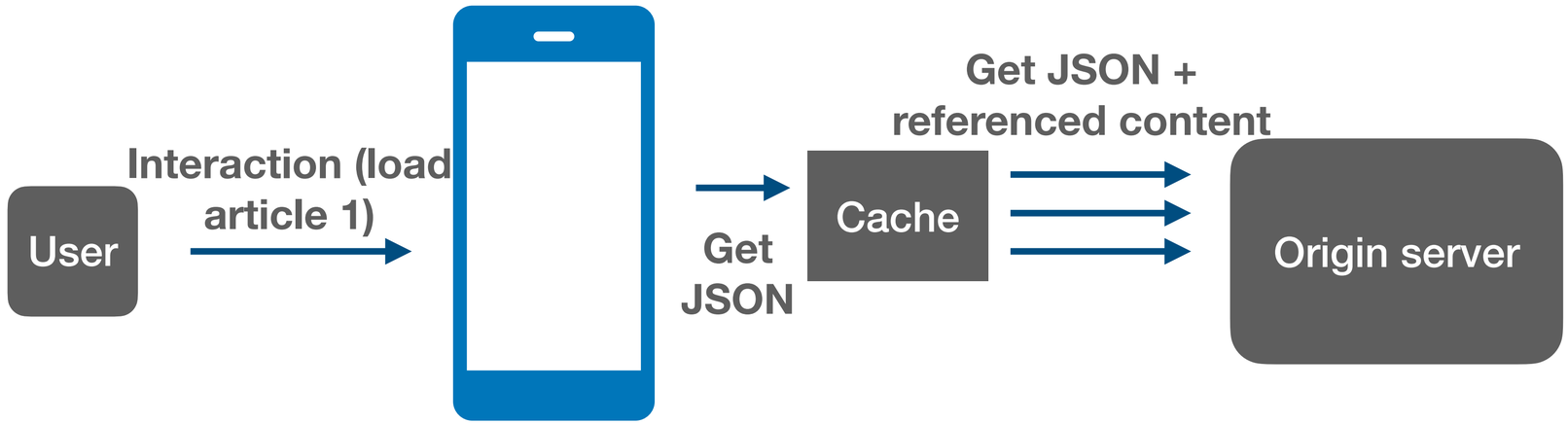
Just-in-time prefetching

JSON: article1.json
Referenced content: x.jpg

Normal Operation



Optimization: Parse JSONs and determine referenced content in advance

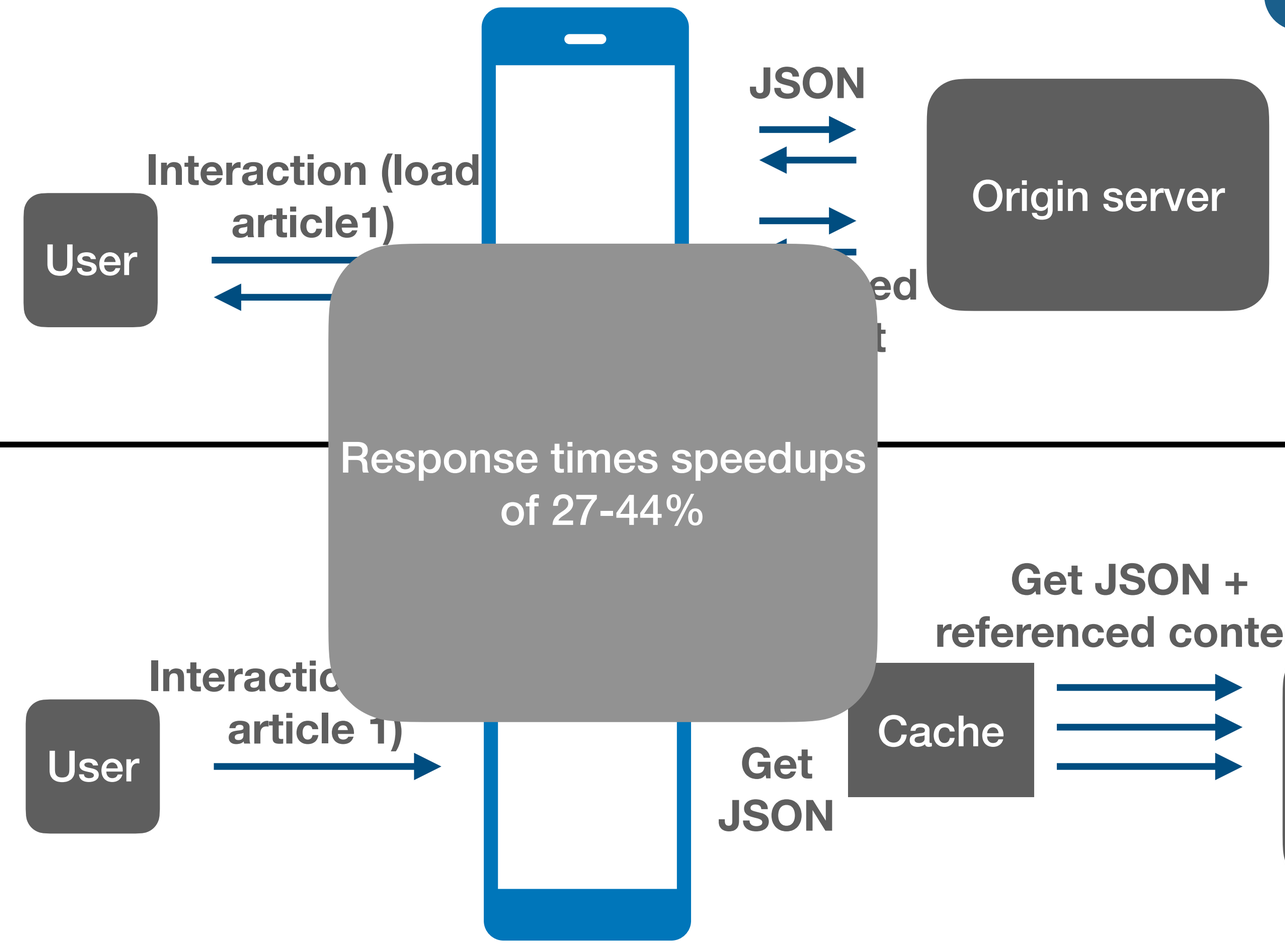


Sidestep the prediction problem

“User requested article1.json so they’ll need the images in it as well”

Just-in-time prefetching

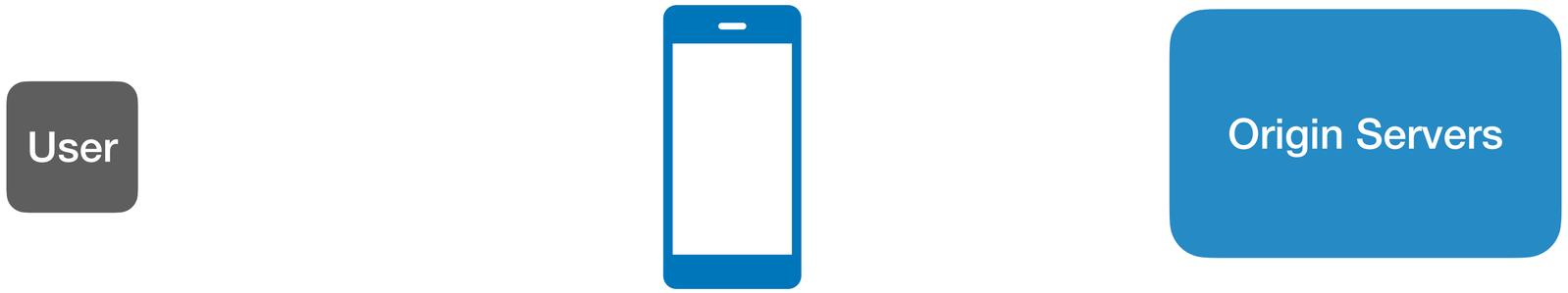
JSON: article1.json
Referenced content: x.jpg



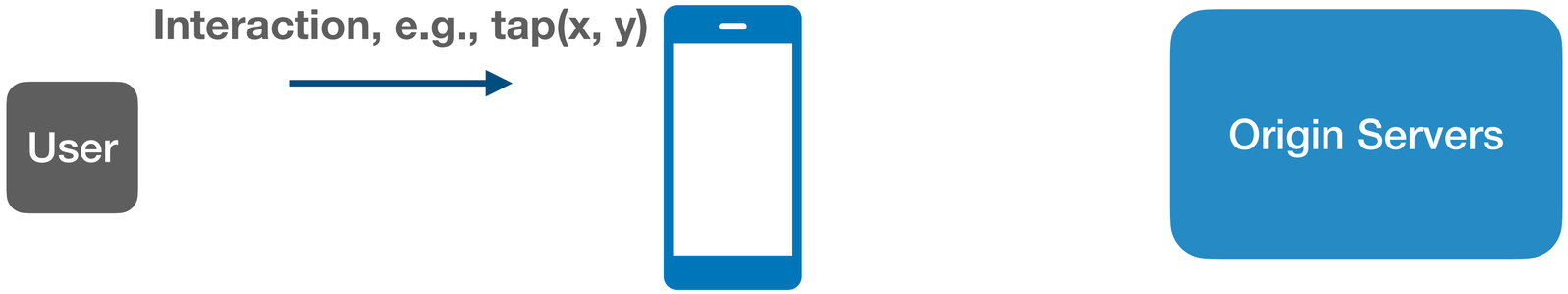
Sidestep the prediction problem

“User requested article1.json so they’ll need the images in it as well”

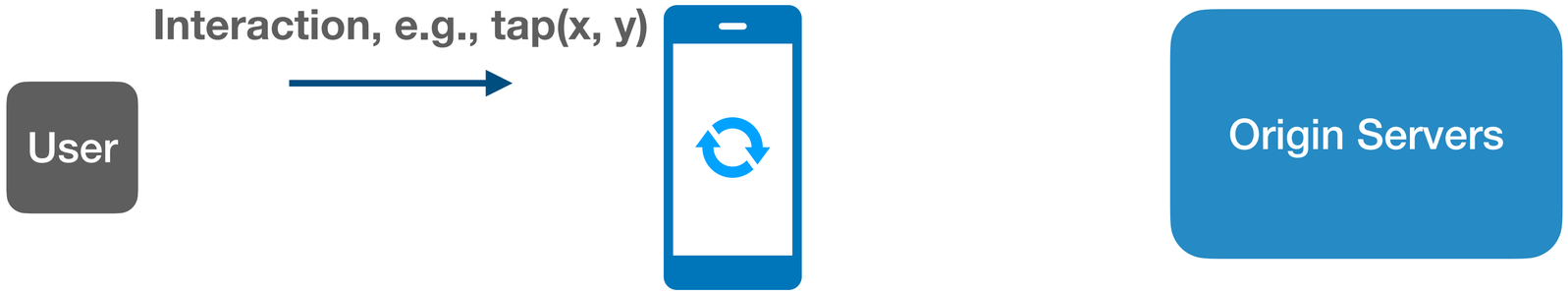
Other caching opportunities?



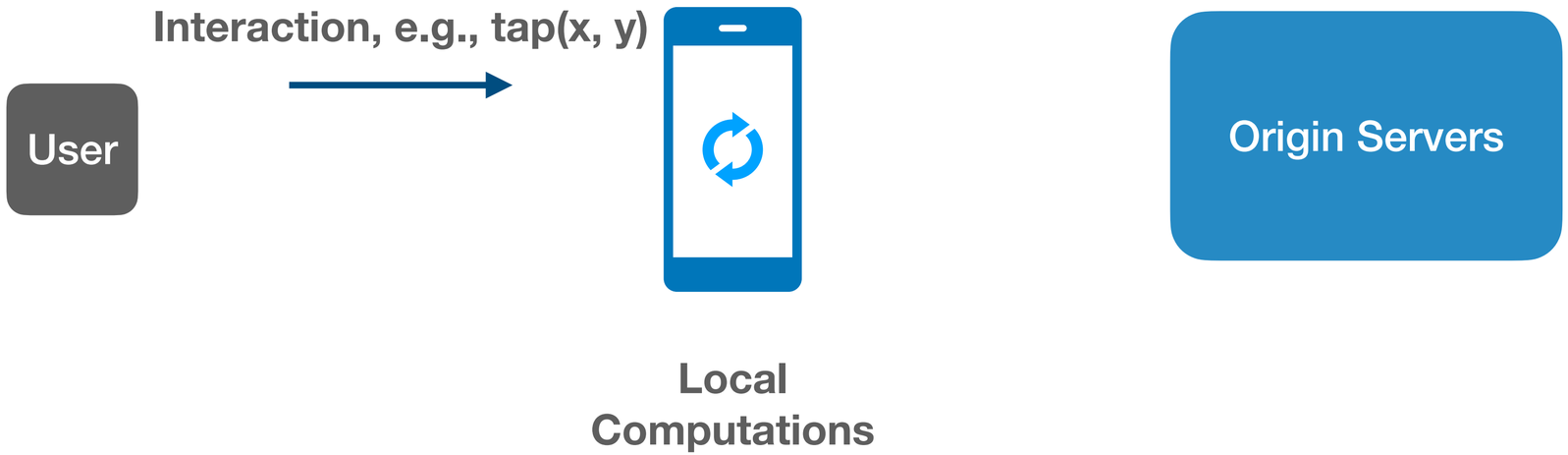
Other caching opportunities?



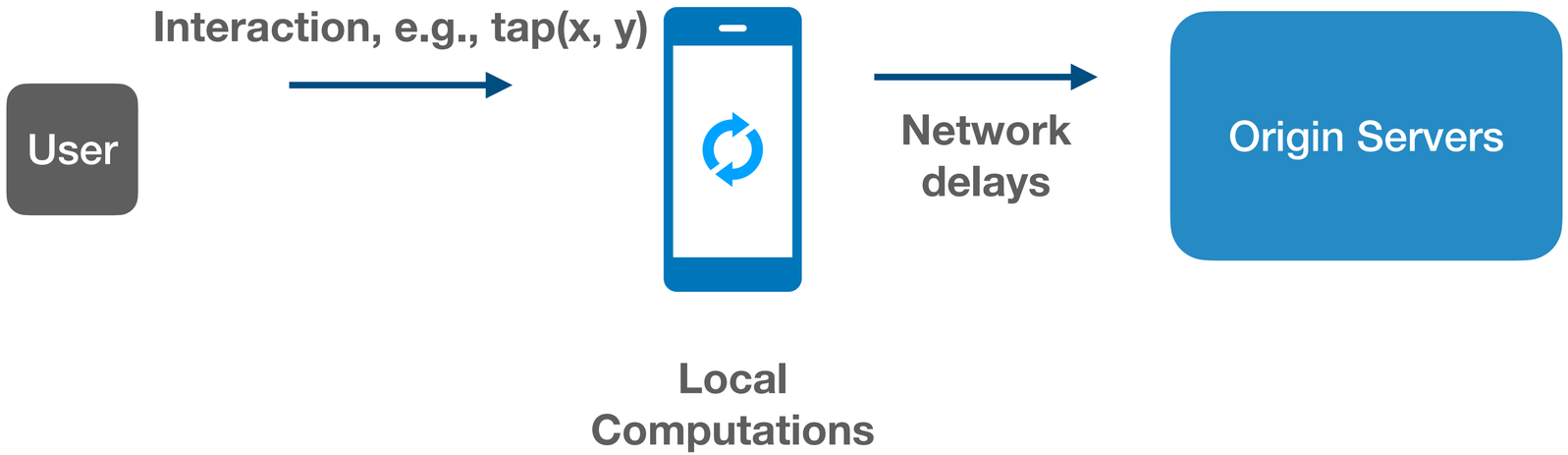
Other caching opportunities?



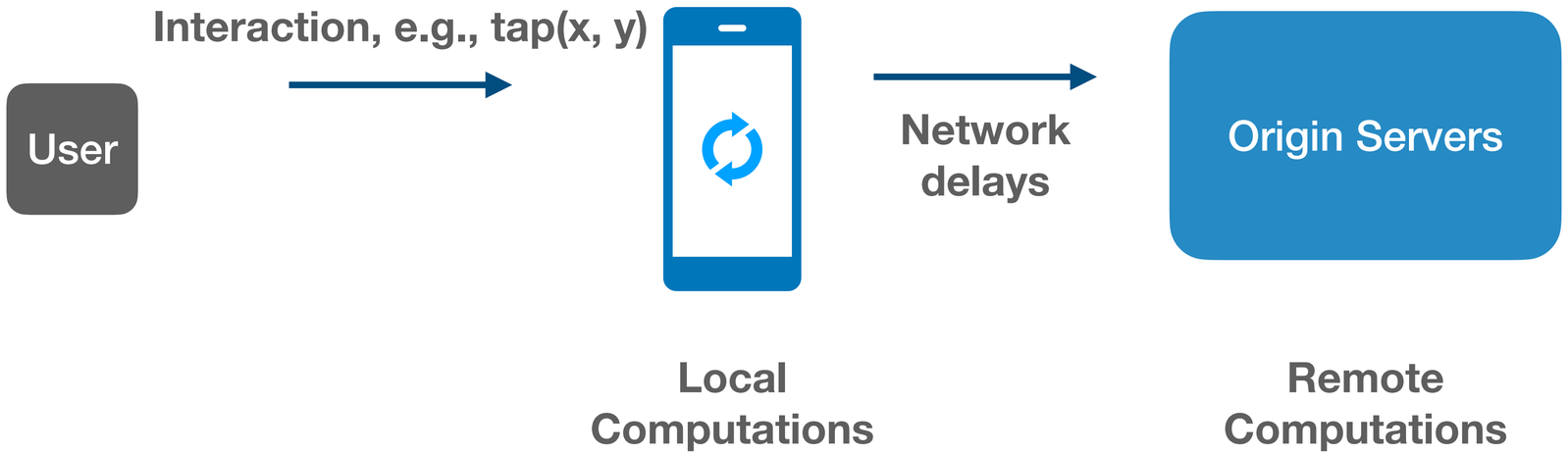
Other caching opportunities?



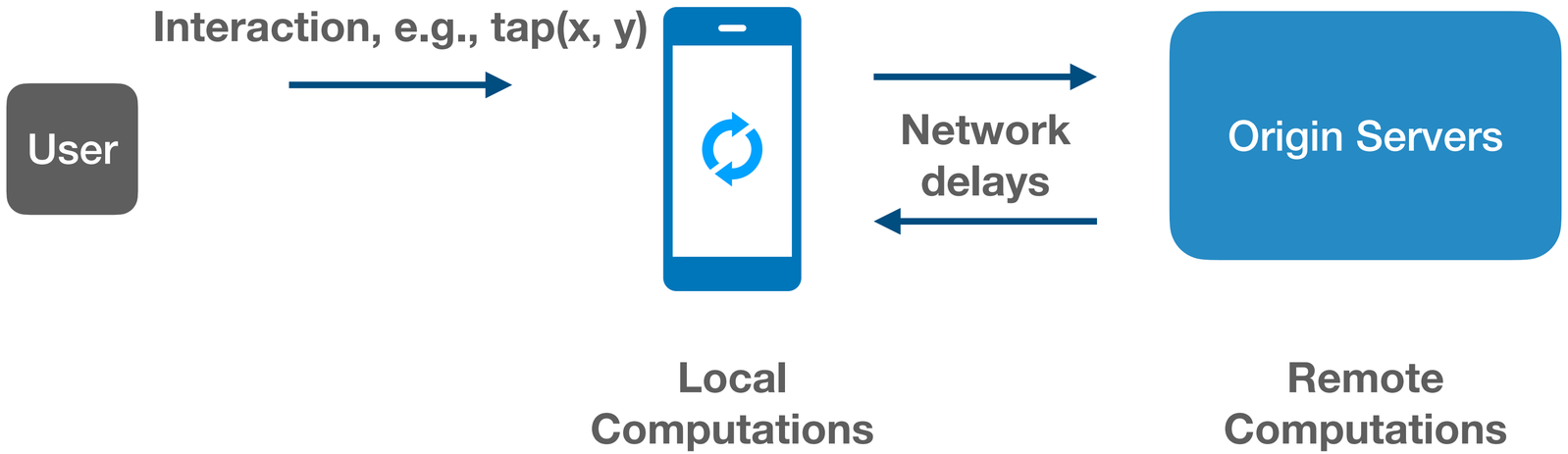
Other caching opportunities?



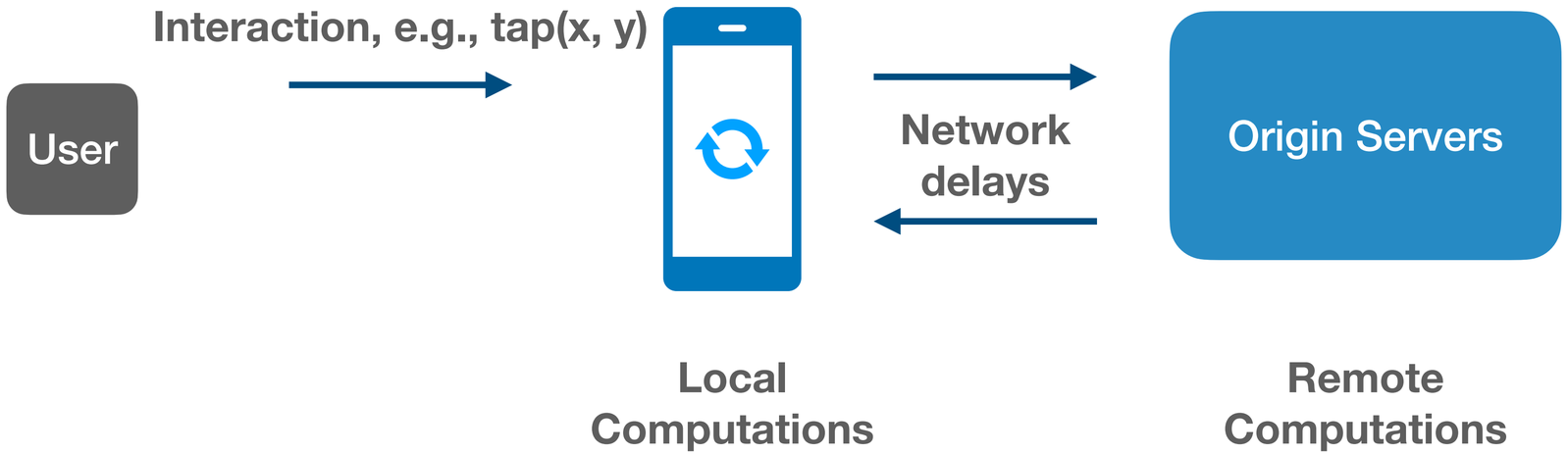
Other caching opportunities?



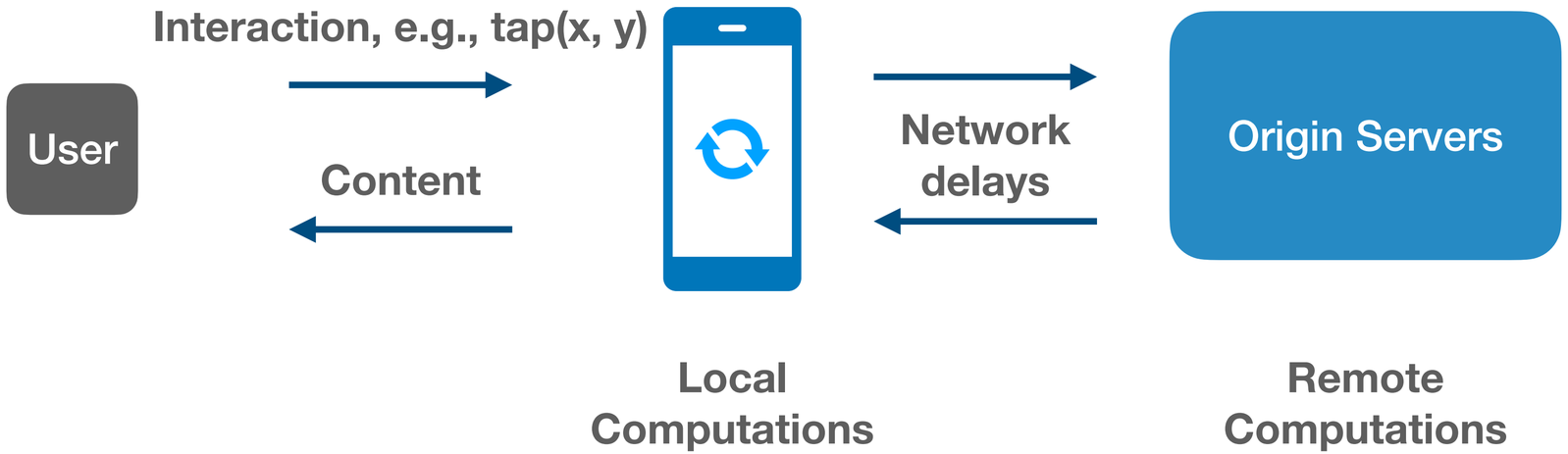
Other caching opportunities?



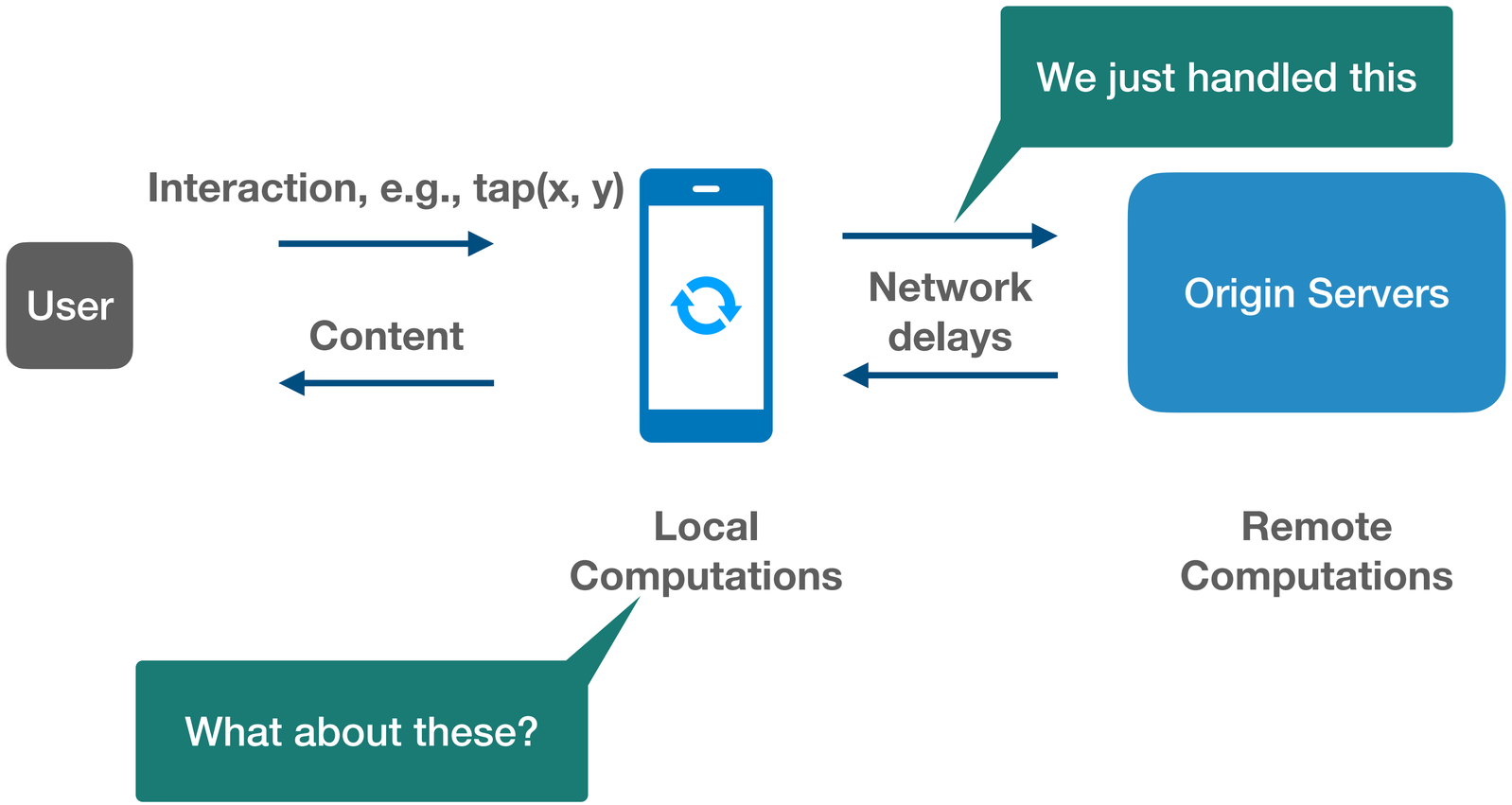
Other caching opportunities?



Other caching opportunities?



Other caching opportunities?



Intro to memoization

Intro to memoization

```
int fn(int input) {
  return input*2;
}
```

Intro to memoization

```
int fn(int input) {
  return input*2;
}
```

```
int fn(int input) {
  if (memo.contains(input)) {
    return memo.get(input);
  } else {
    int tmp = input * 2;
    memo.set(input, tmp);
    return tmp;
  }
}
```

Intro to memoization

```
int fn(int input) {
  return input*2;
}
```

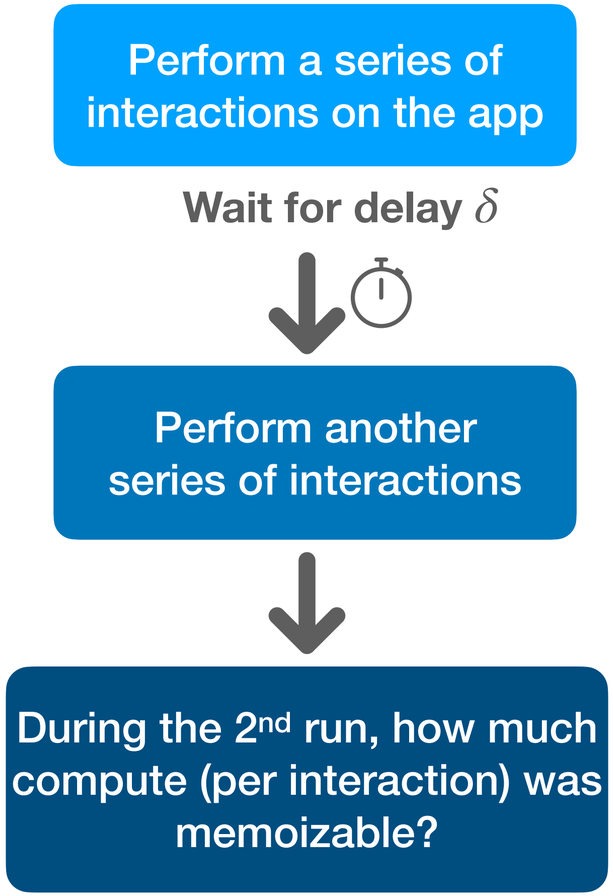
```
int fn(int input) {
  if (memo.contains(input)) {
    return memo.get(input);
  } else {
    int tmp = input * 2;
    memo.set(input, tmp);
    return tmp;
  }
}
```

If read state matches a prior run

Apply prior writes

Potential of memoization

Potential of memoization



Potential of memoization

Perform a series of interactions on the app

Wait for delay δ

Perform another series of interactions

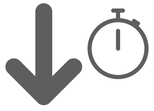
During the 2nd run, how much compute (per interaction) was memoizable?

Upper bound of memoization benefits with zero overhead

Potential of memoization

Perform a series of interactions on the app

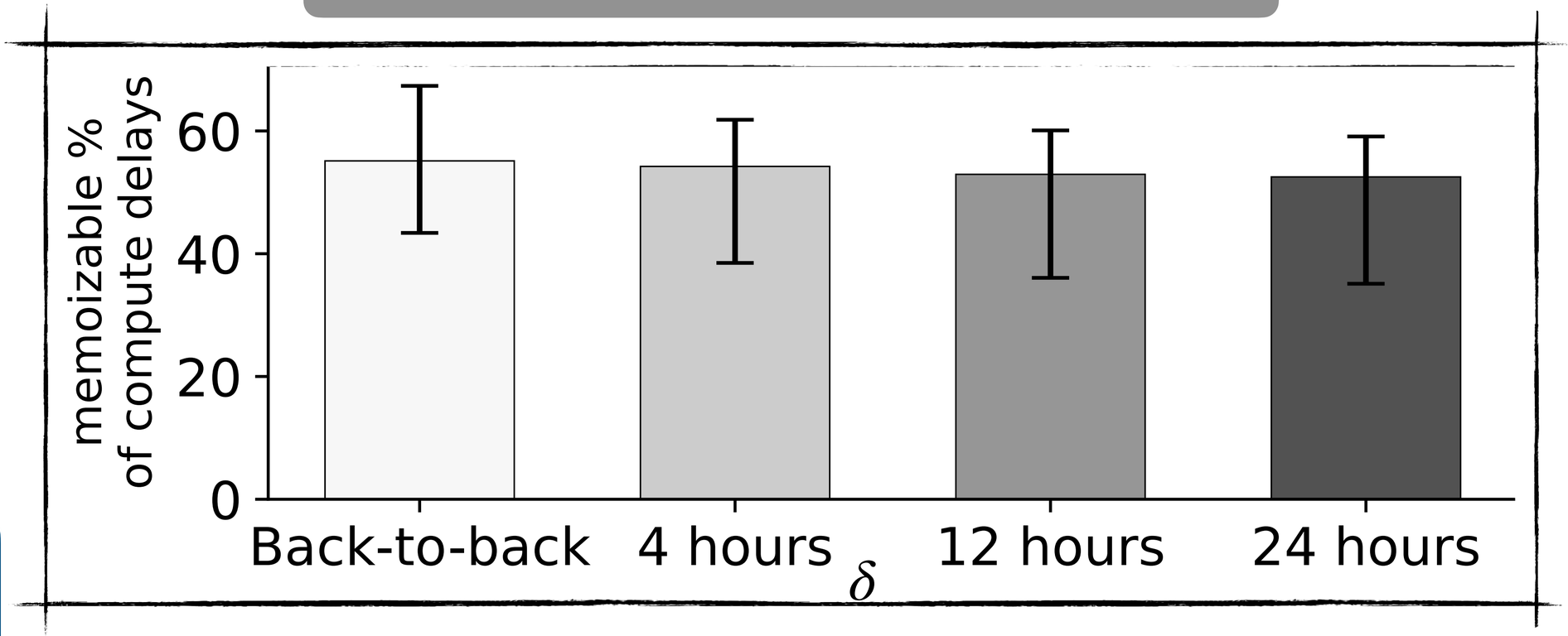
Wait for delay δ



Perform another series of interactions

During the 2nd run, how much compute (per interaction) was memoizable?

Upper bound of memoization benefits with zero overhead



Potential of memoization

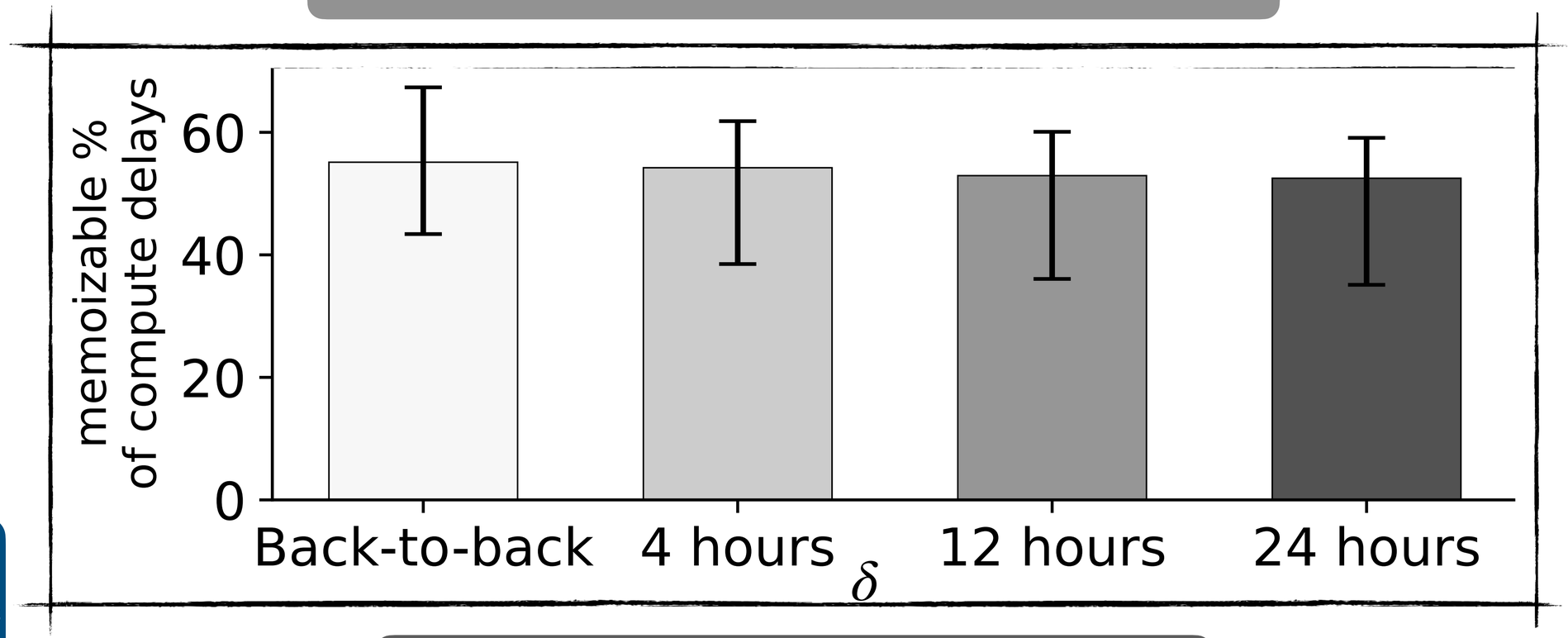
Perform a series of interactions on the app

Wait for delay δ

Perform another series of interactions

During the 2nd run, how much compute (per interaction) was memoizable?

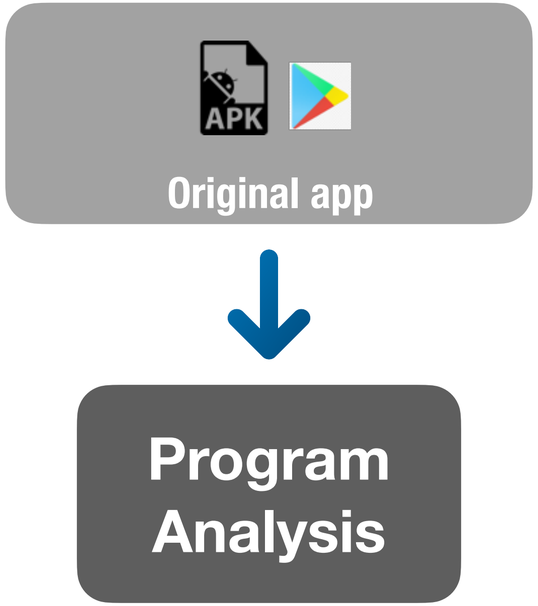
Upper bound of memoization benefits with zero overhead



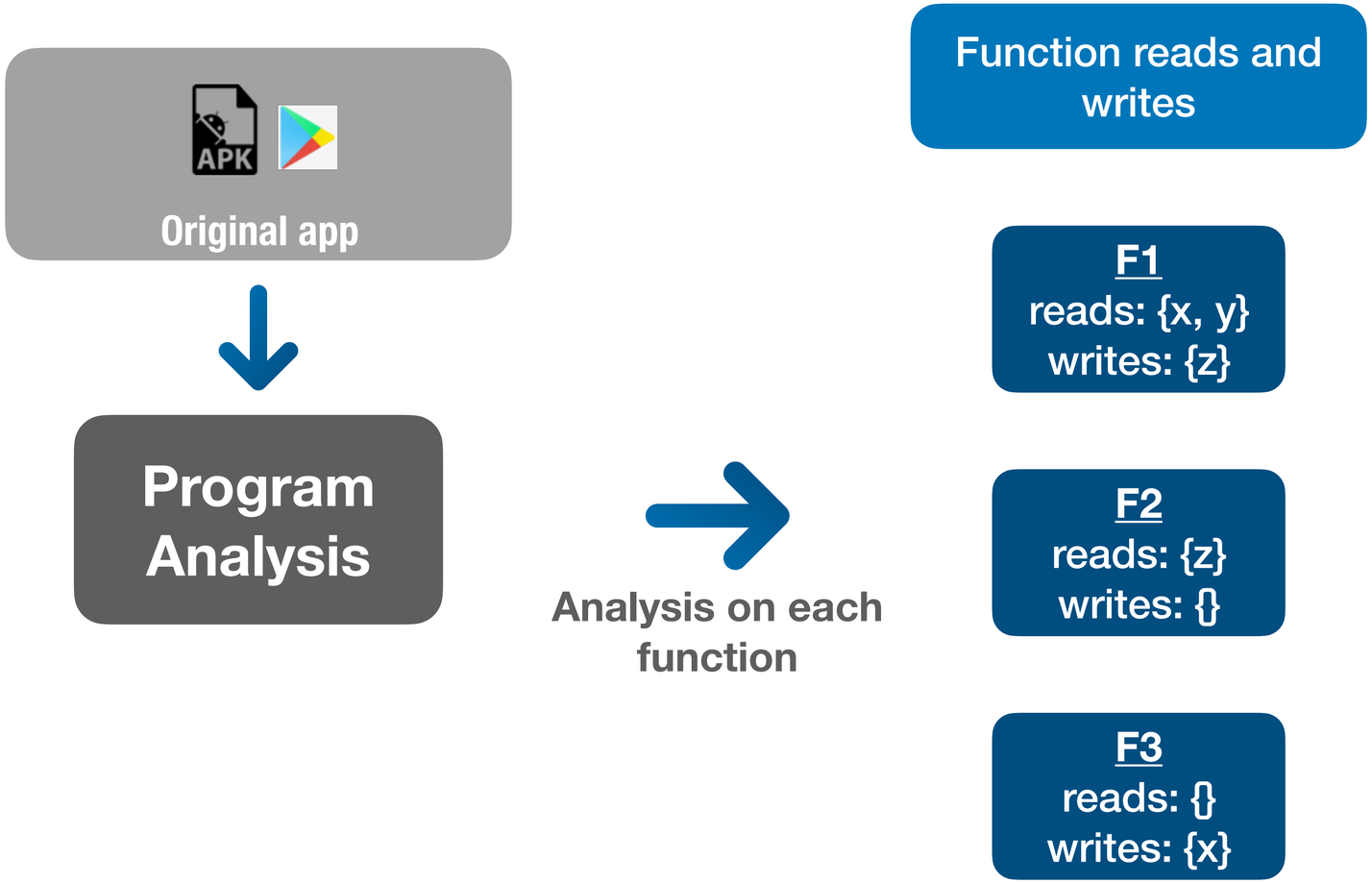
Benefits persist over time due to stable computations

Implementing memoization

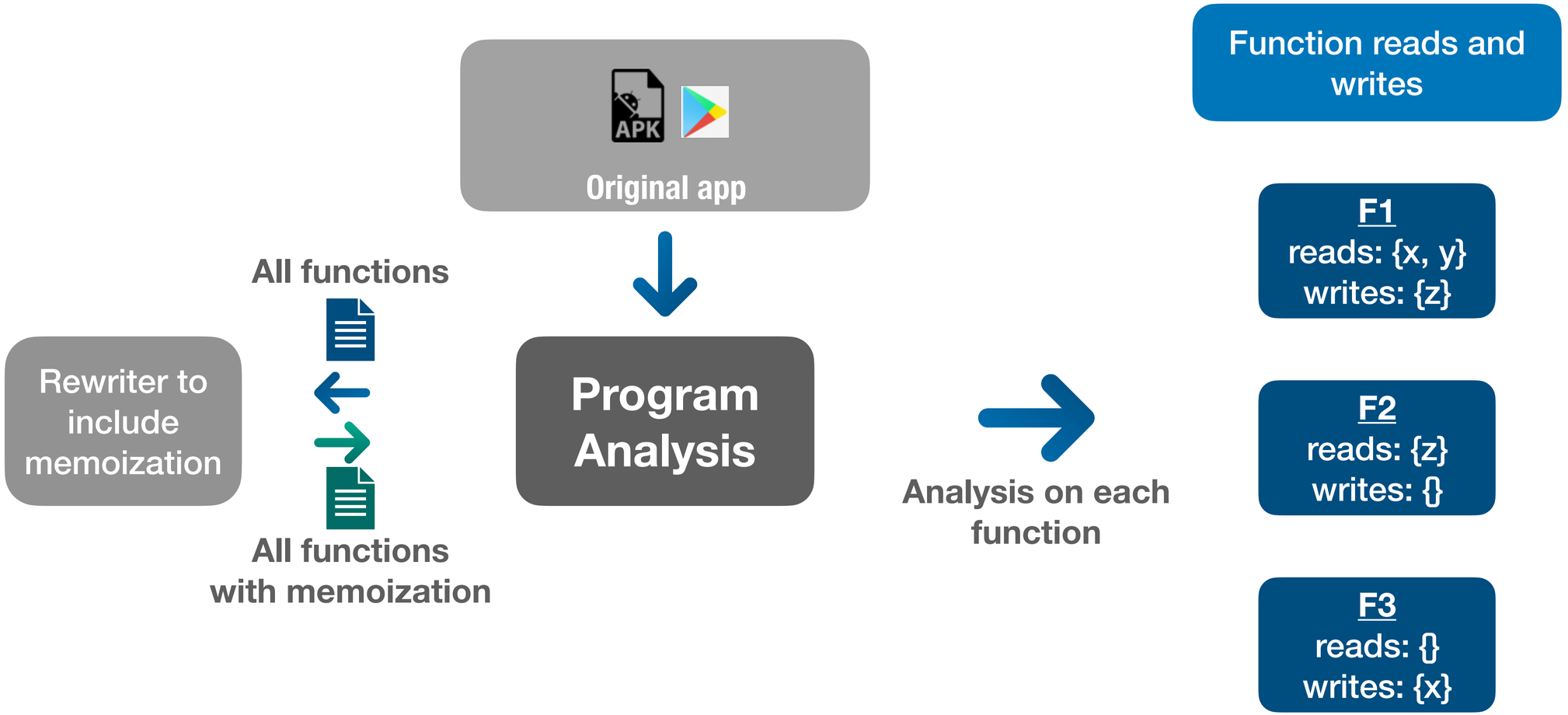
Implementing memoization



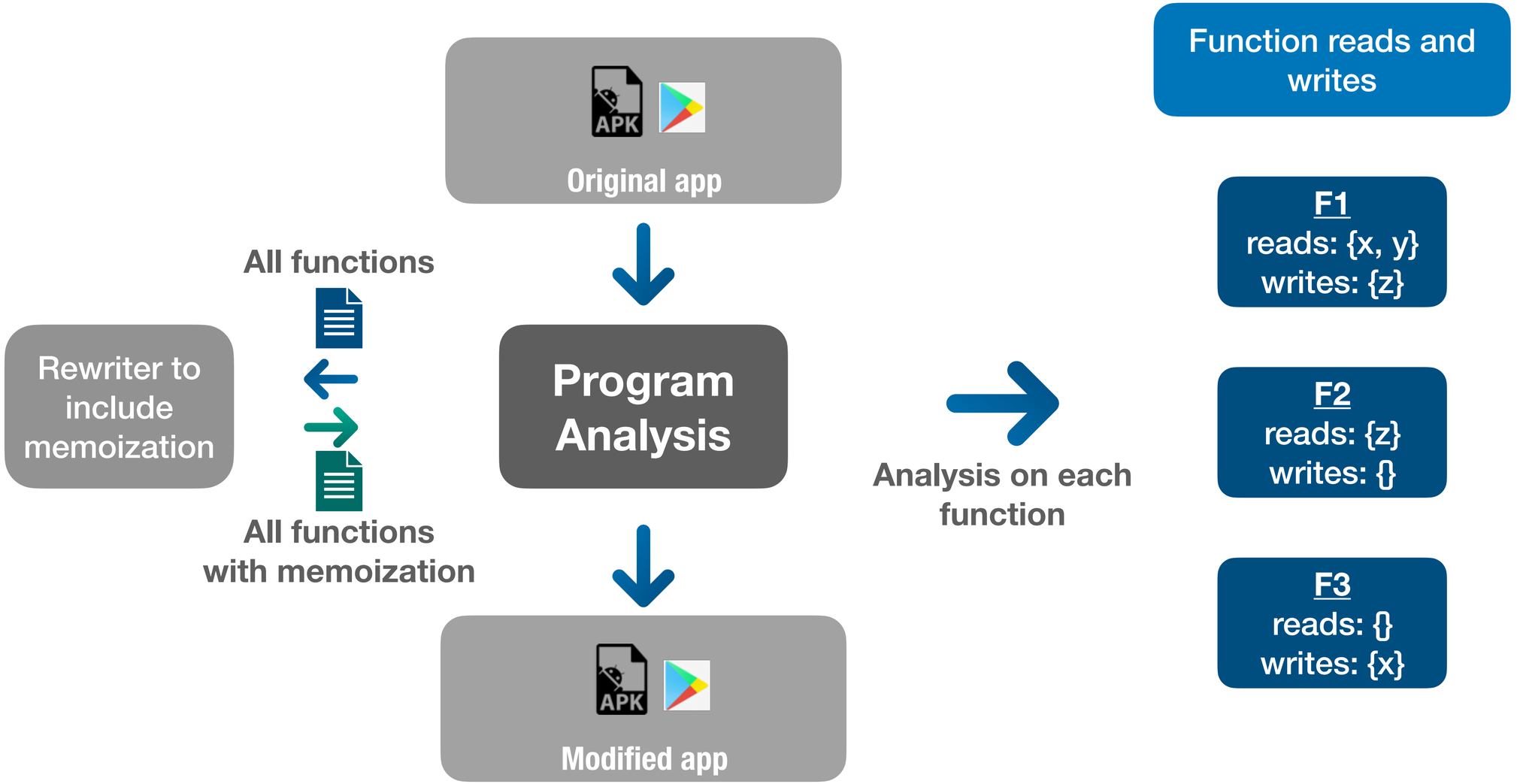
Implementing memoization



Implementing memoization



Implementing memoization

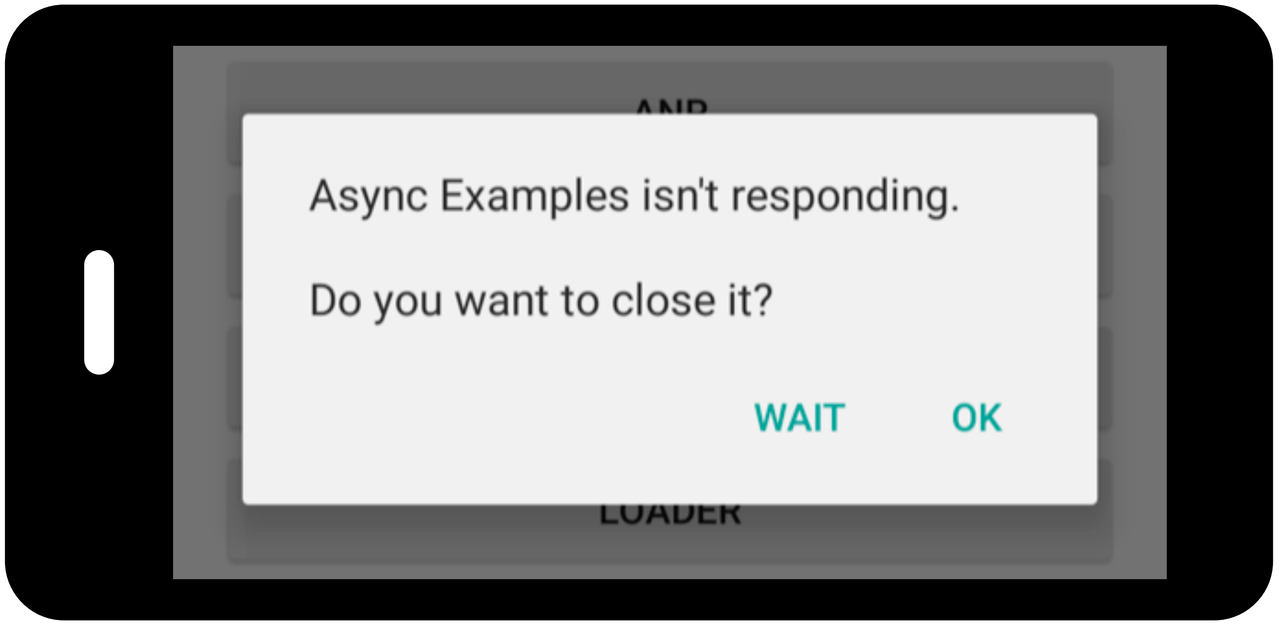


Straightforward memoization

The result

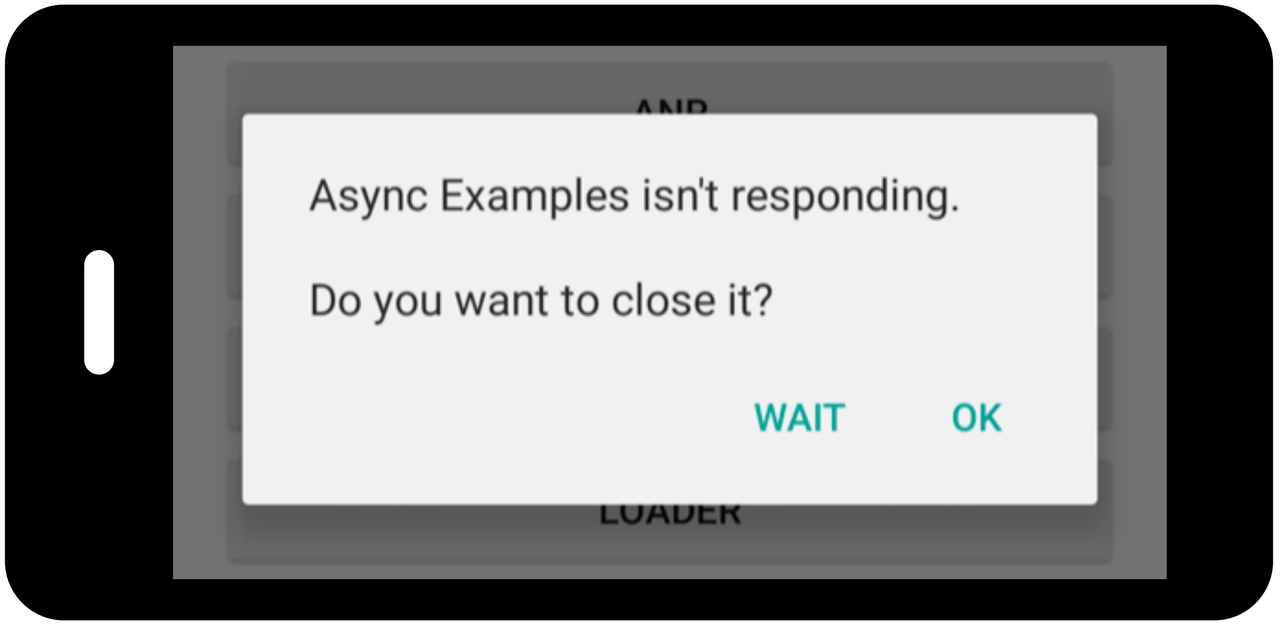
Straightforward memoization

The result



Straightforward memoization

The result



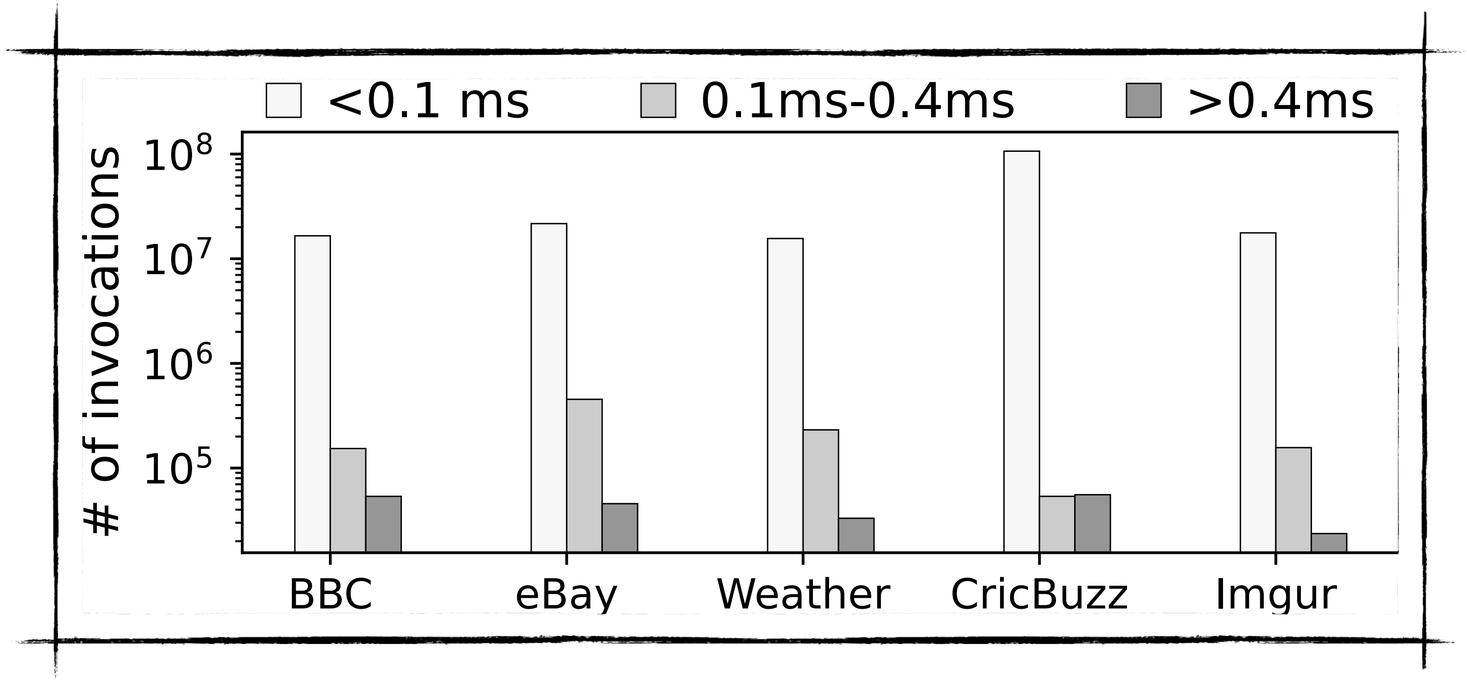
Blocking, expensive, potentially unnecessary cache queries

Challenges with memoization

An experimental result

Challenges with memoization

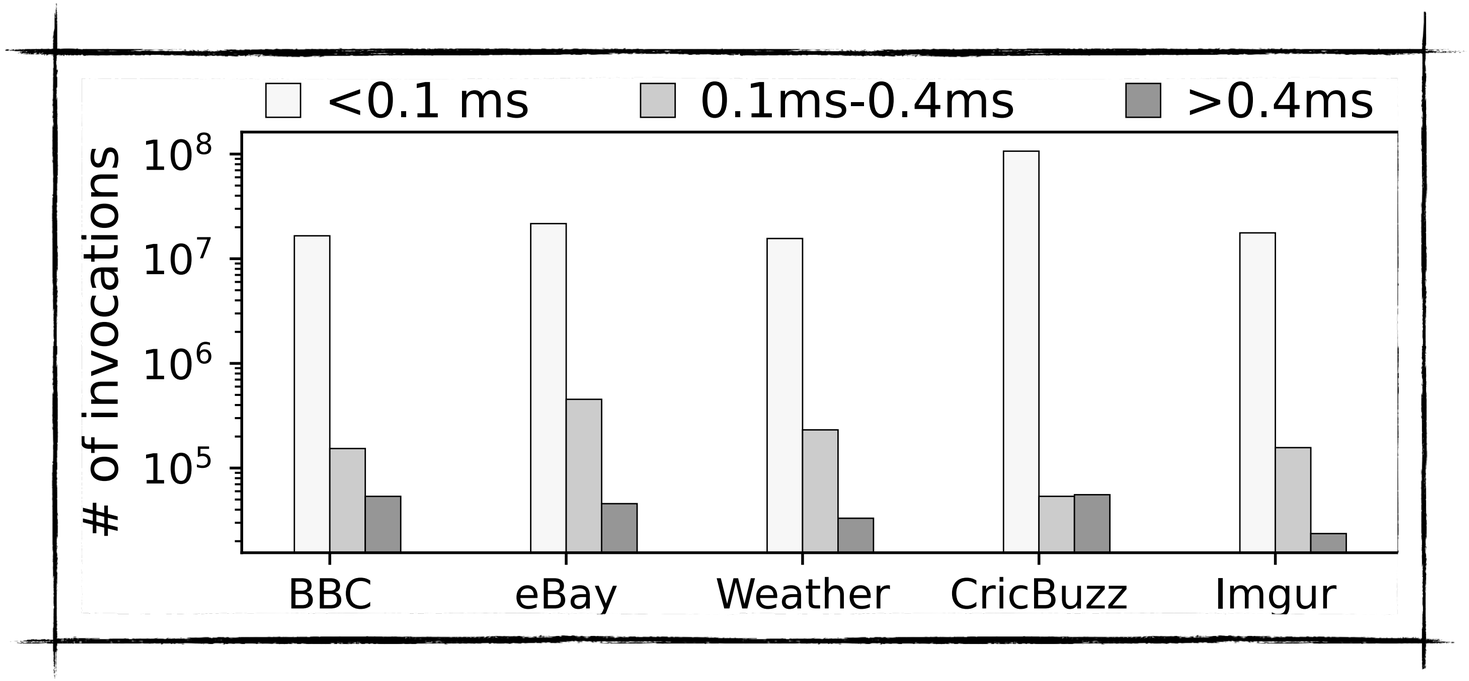
An experimental result



Function invocation runtimes for 5 exemplar apps

Challenges with memoization

An experimental result



Function invocation runtimes for 5 exemplar apps

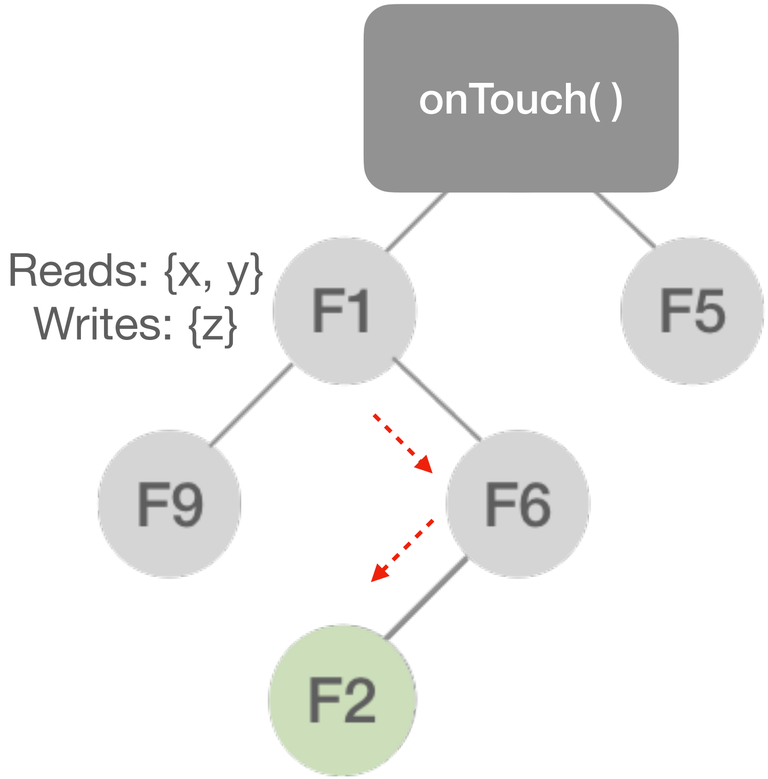
Large number of invocations → Queries can overwhelm resources

Low function runtimes → Overheads can exceed runtimes

Lookaheads

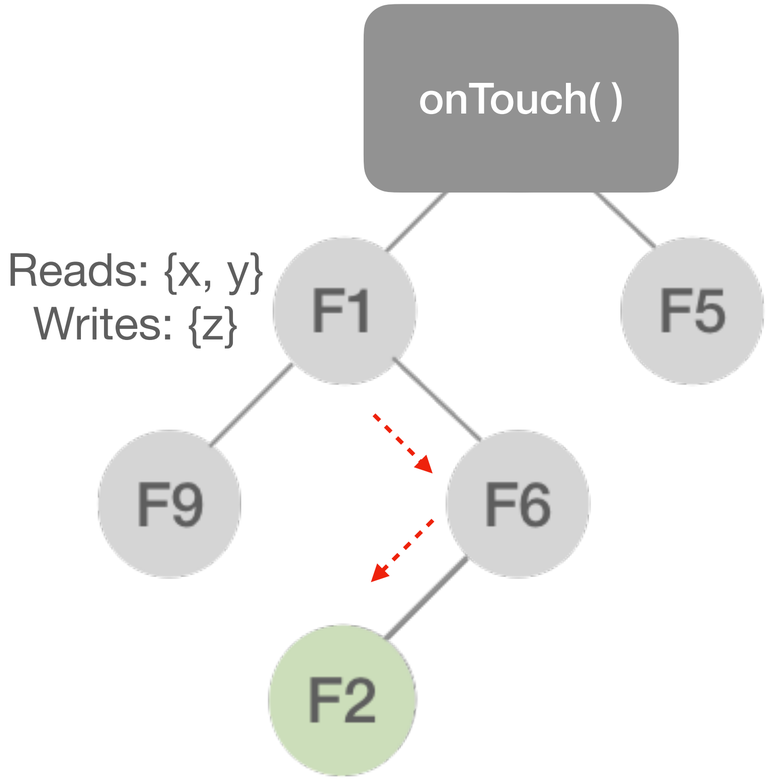
Making cache lookups fast!

Lookaheads
Making cache lookups fast!



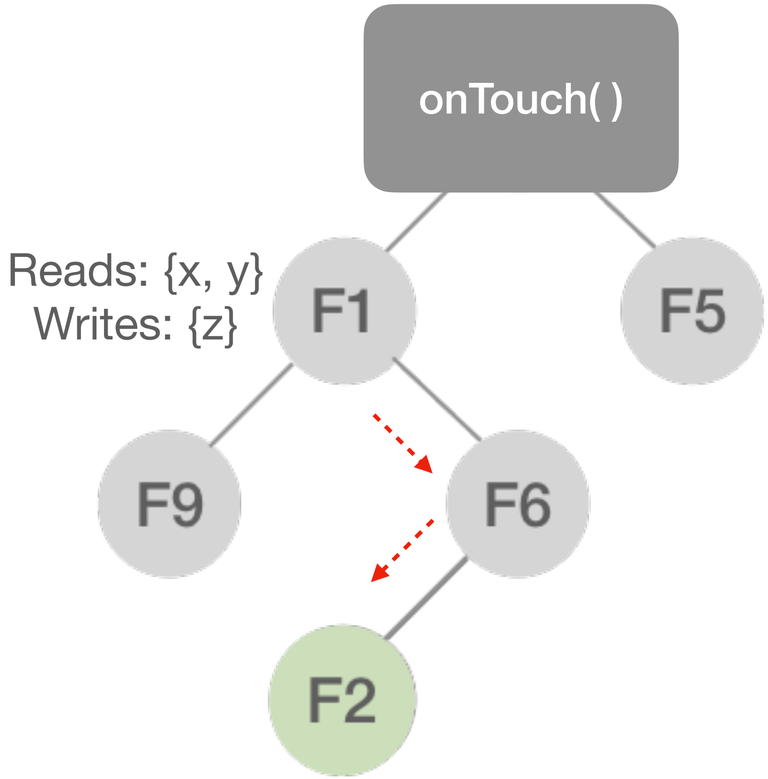
- Offline callgraph generation. Used Online.
- Annotate nodes.
  - Variable dependencies

Lookaheads
Making cache lookups fast!



- Offline callgraph generation. Used Online.
- Annotate nodes.
  - Variable dependencies
- Identify *potentially* upcoming functions.
- Query these functions → Warm the cache

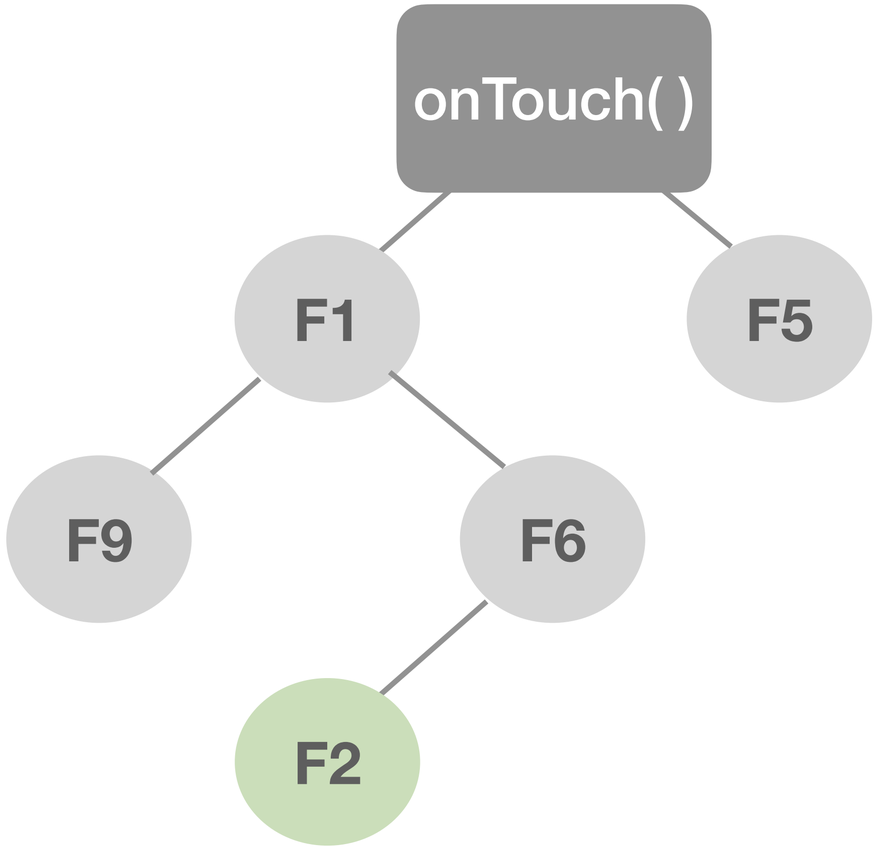
Lookaheads
Making cache lookups fast!



- Offline callgraph generation. Used Online.
- Annotate nodes.
  - Variable dependencies
- Identify *potentially* upcoming functions.
- Query these functions → Warm the cache
  - ***Read state of upcoming functions may vary!***

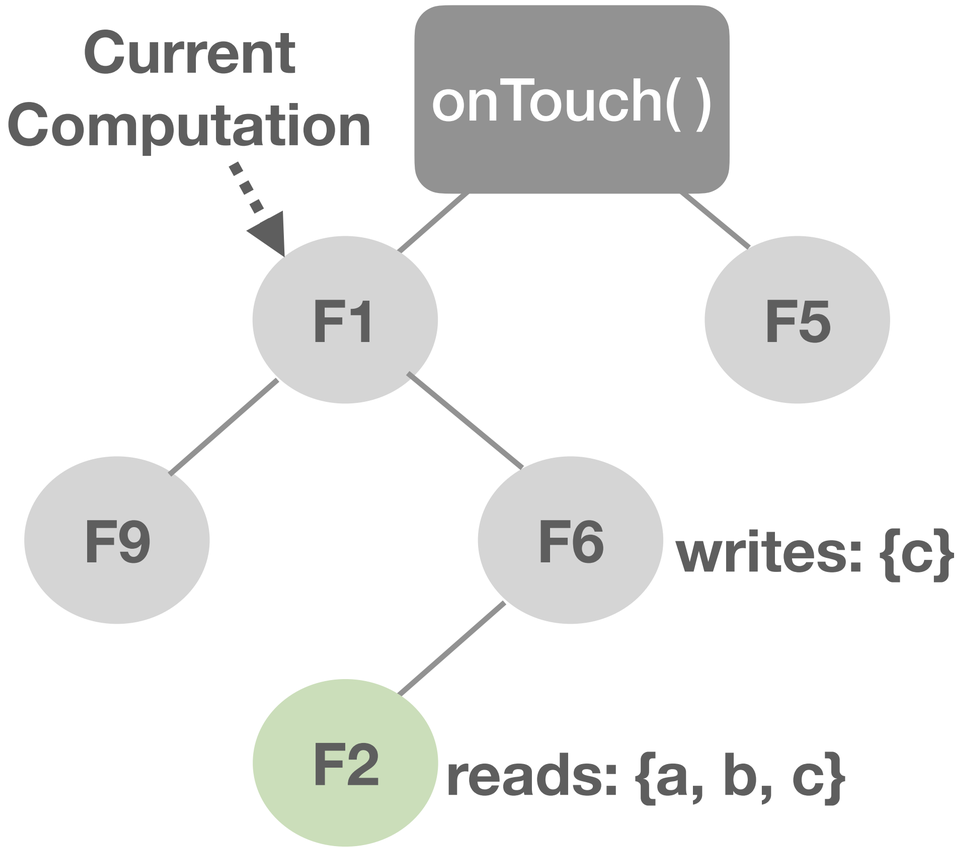
Lookaheads

Lookaheads



Static callgraph

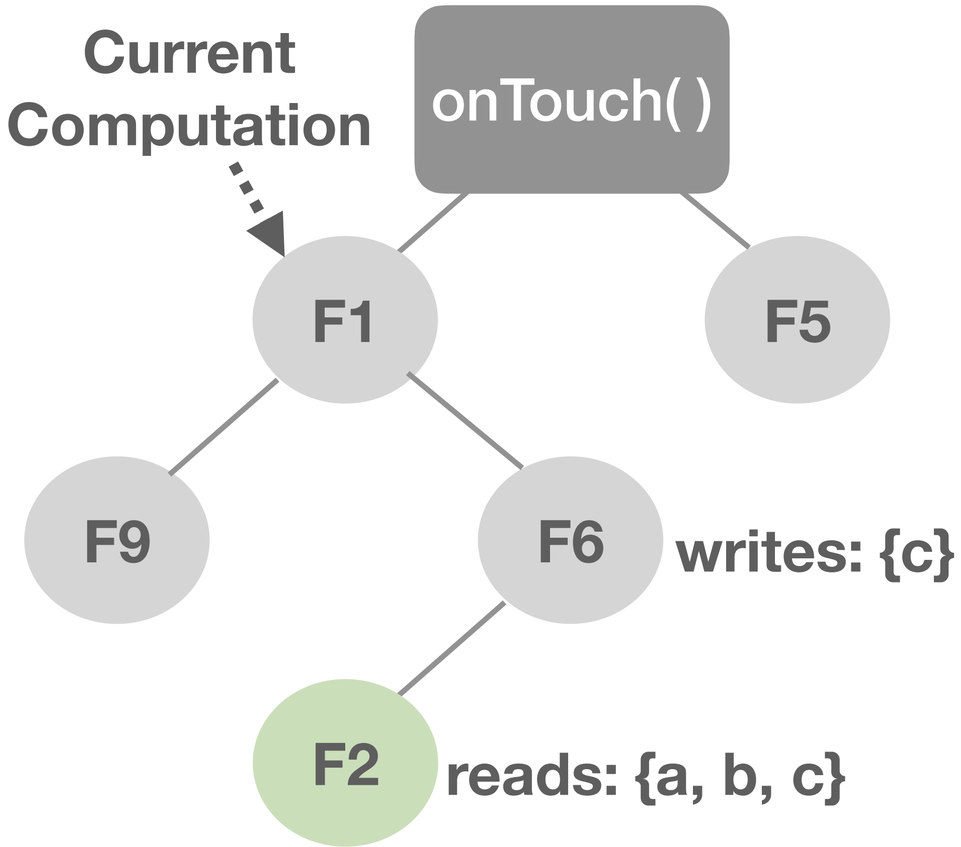
Lookaheads



Static callgraph

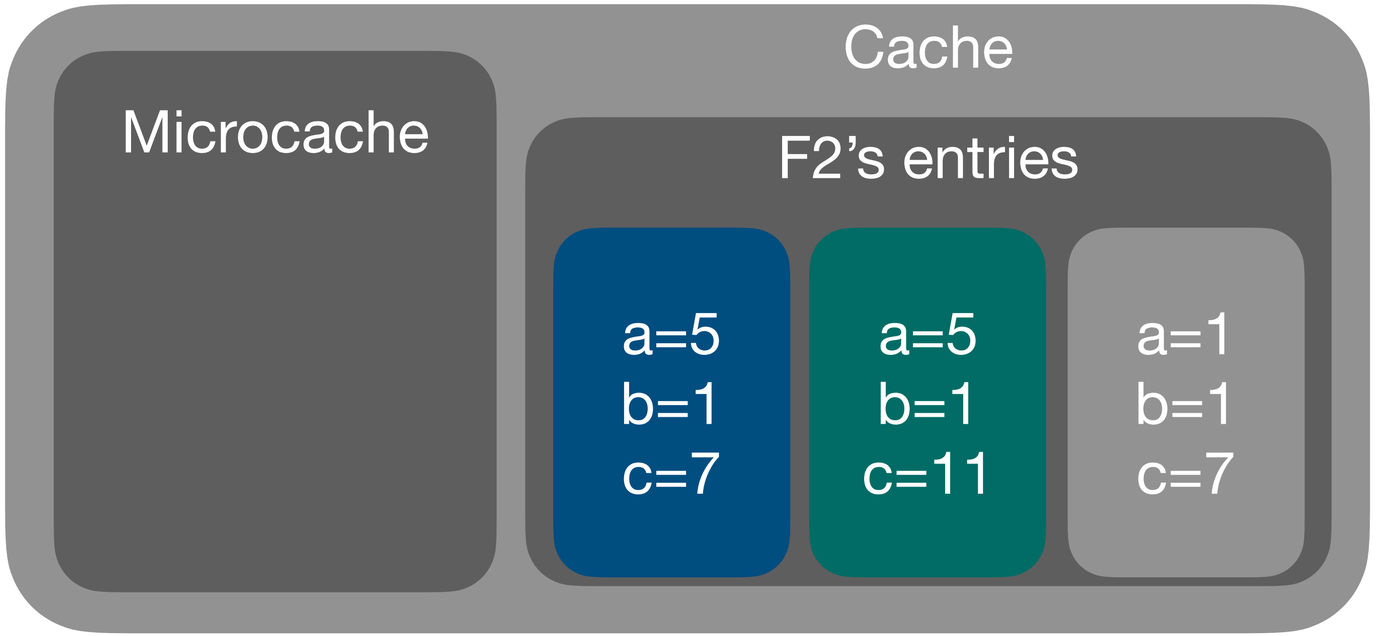
- F2 may run shortly. F2's reads: {a, b, c}
- F6 may write to {c}
  - No writes to {a, b} before F2
- Lookup {a, b, c=*}
  - Use current values {a=5, b=1}
  - Results → Microcache

Lookaheads

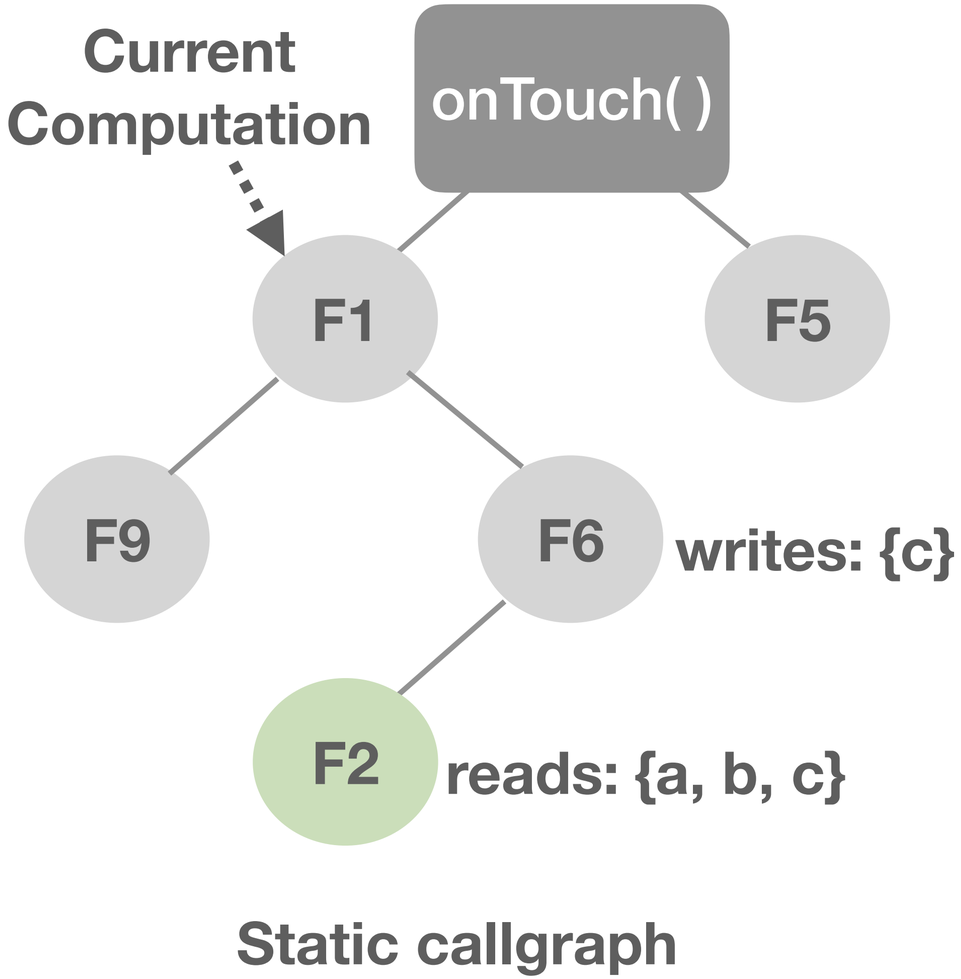


Static callgraph

- F2 may run shortly. F2's reads: {a, b, c}
- F6 may write to {c}
  - No writes to {a, b} before F2
- Lookup {a, b, c=*}
  - Use current values {a=5, b=1}
  - Results → Microcache

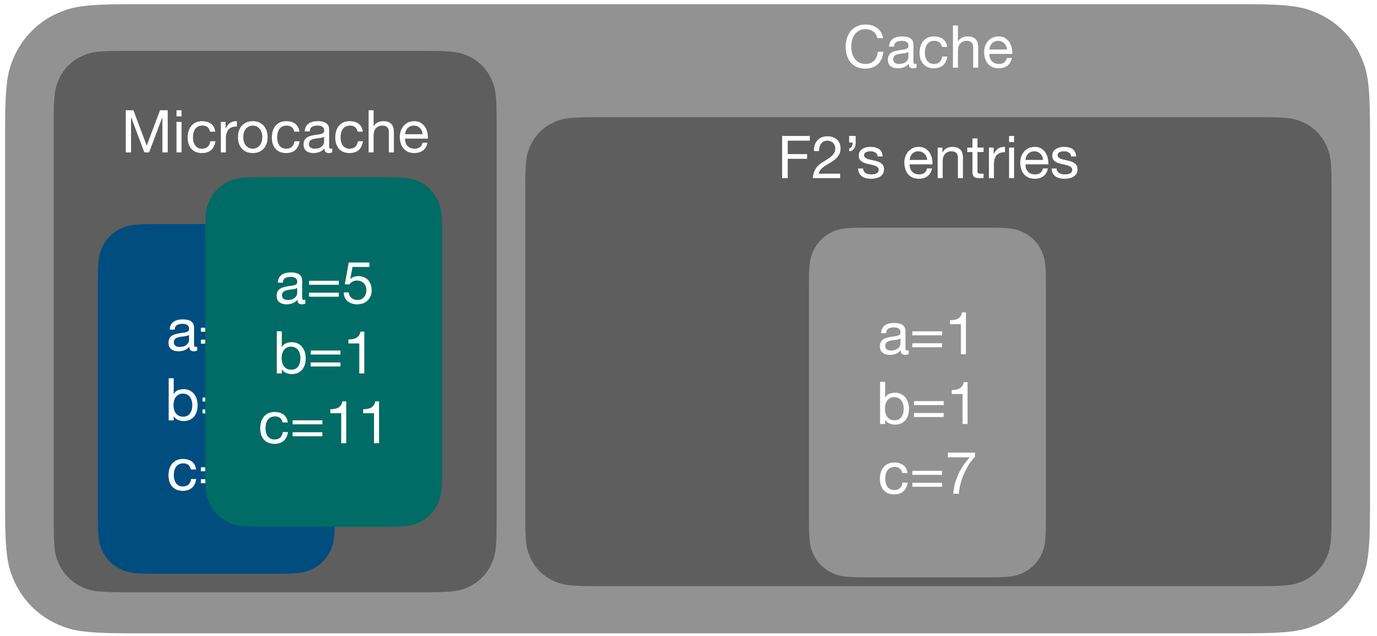


Lookaheads

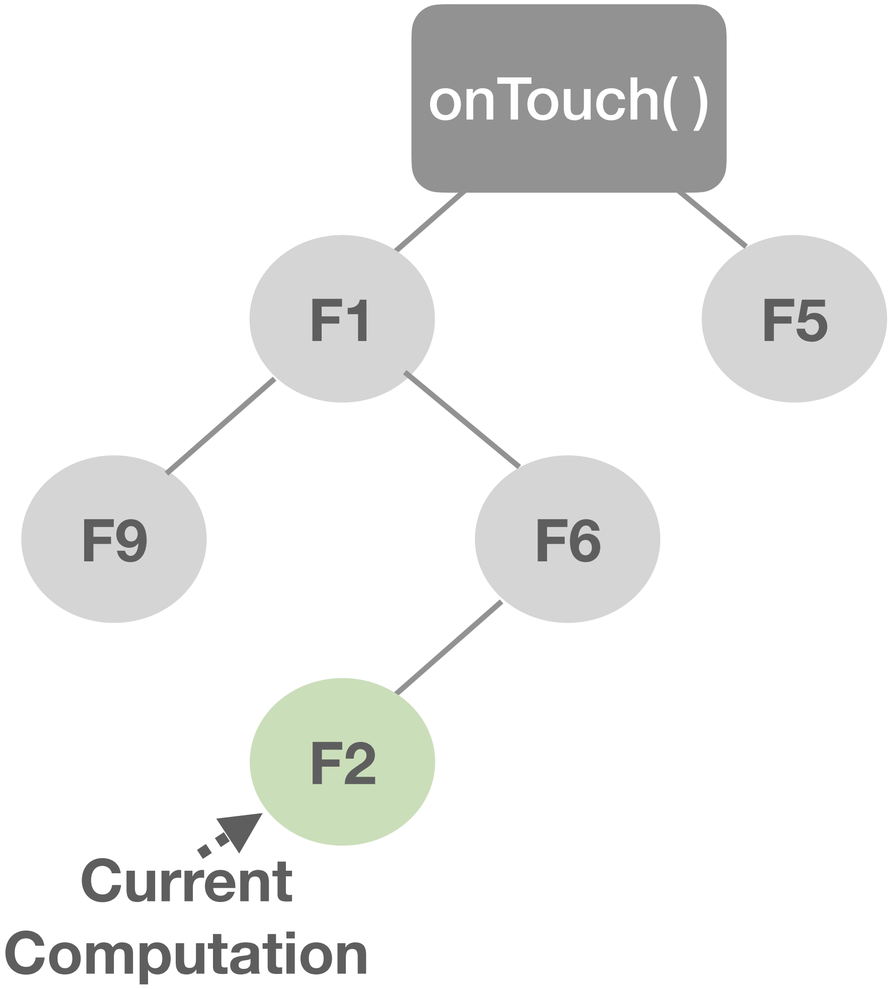


Static callgraph

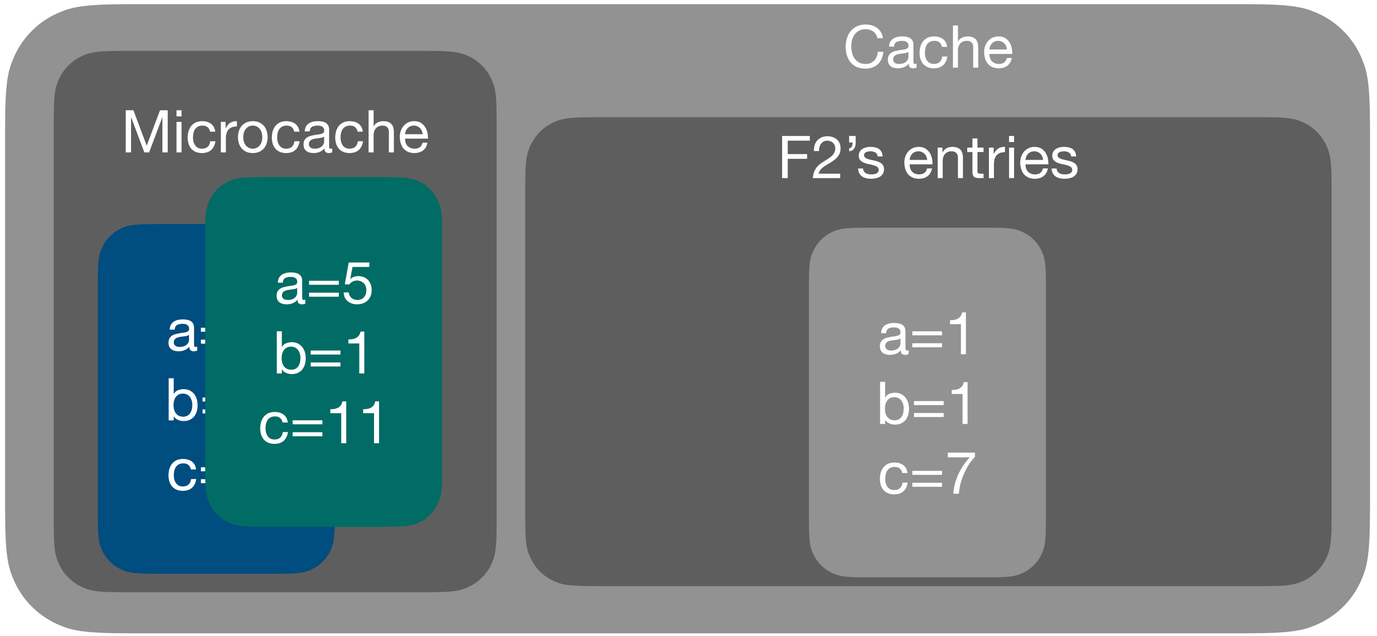
- F2 may run shortly. F2's reads: {a, b, c}
- F6 may write to {c}
  - No writes to {a, b} before F2
- Lookup {a, b, c=*}
  - Use current values {a=5, b=1}
  - Results → Microcache



Lookaheads

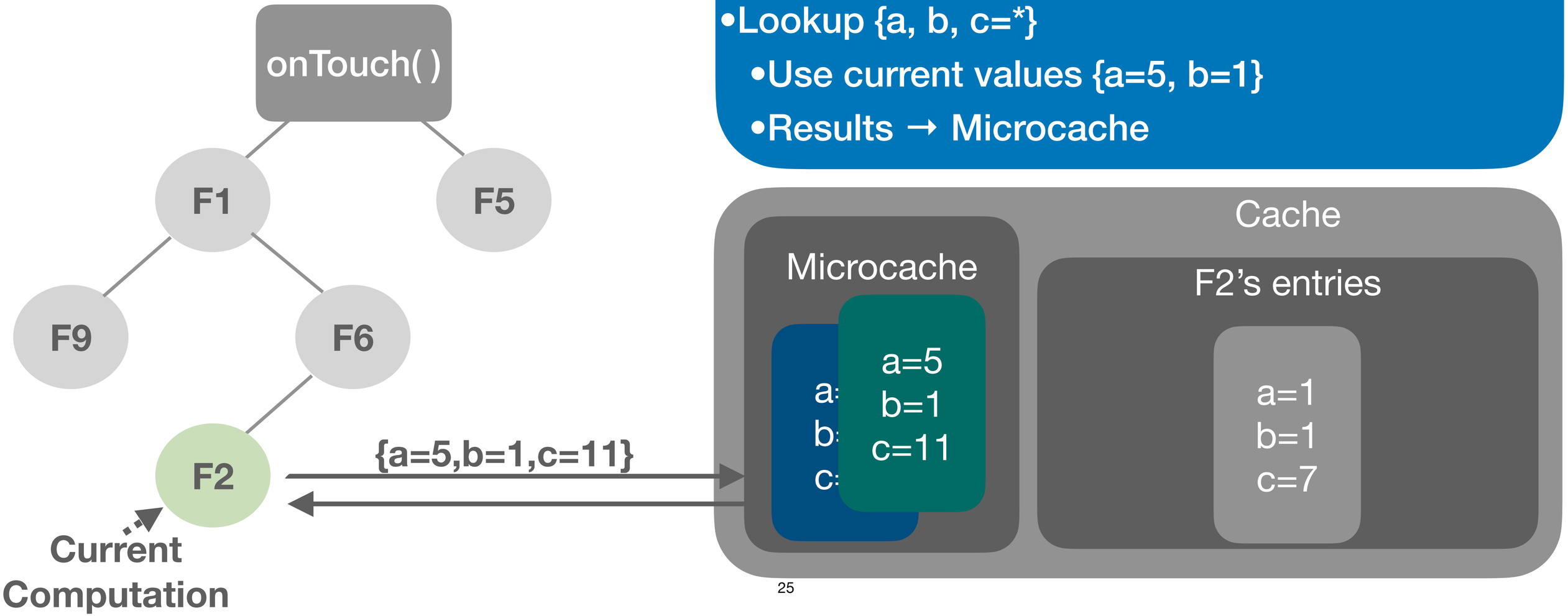


- F2 may run shortly. F2's reads: {a, b, c}
- F6 may write to {c}
  - No writes to {a, b} before F2
- Lookup {a, b, c=*}
  - Use current values {a=5, b=1}
  - Results → Microcache



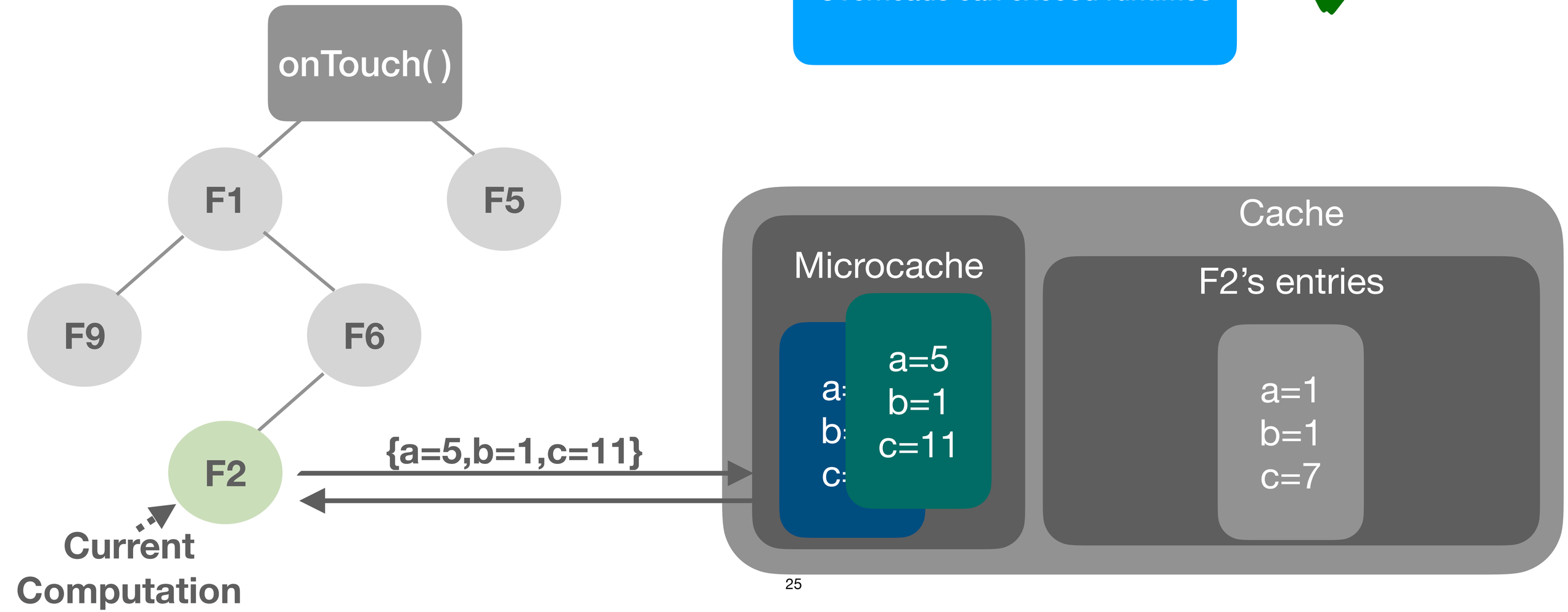
Lookaheads

- F2 may run shortly. F2's reads: {a, b, c}
- F6 may write to {c}
  - No writes to {a, b} before F2
- Lookup {a, b, c=*}
  - Use current values {a=5, b=1}
  - Results → Microcache



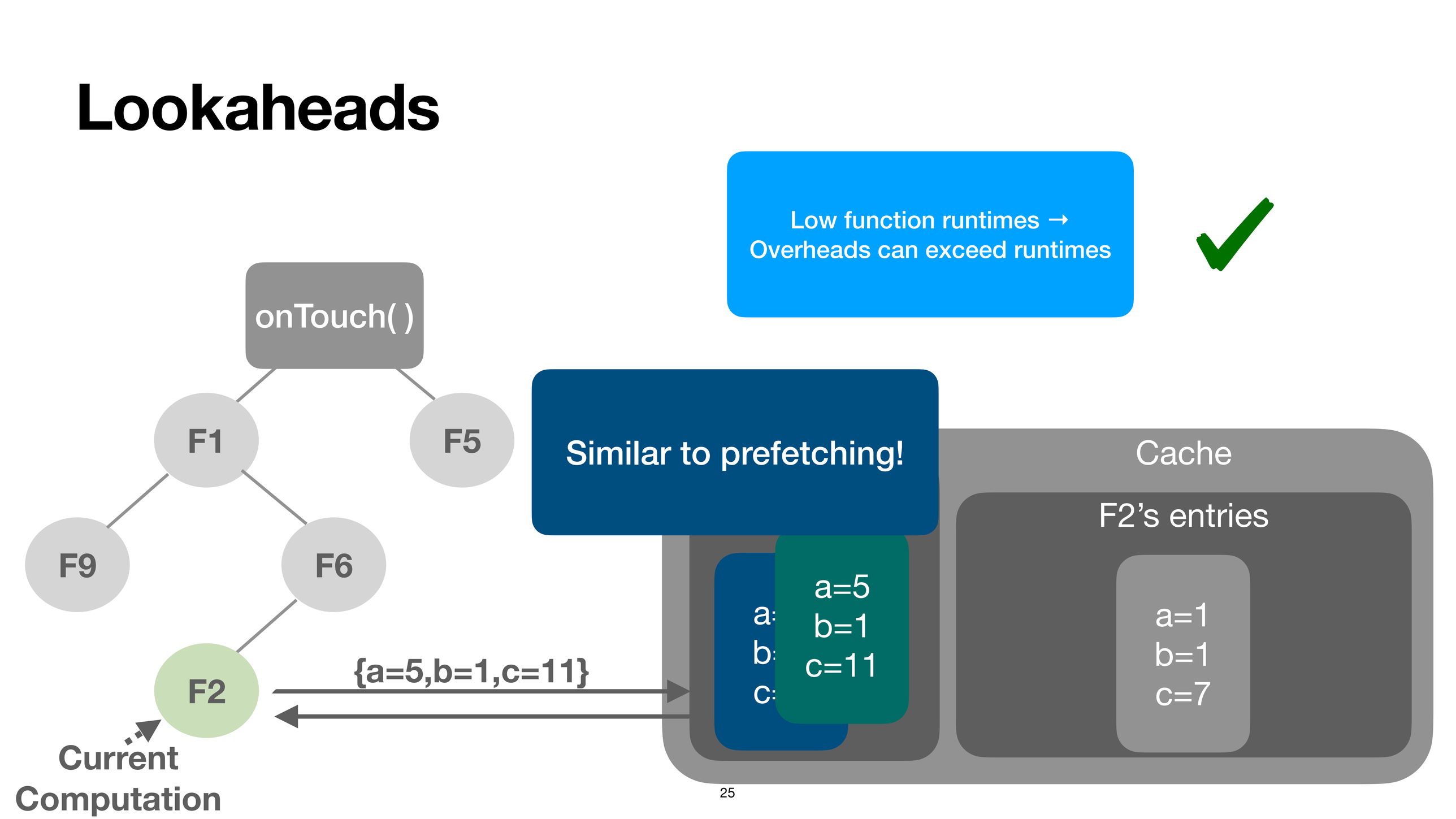
Lookaheads

Low function runtimes →
Overheads can exceed runtimes



Lookaheads

Low function runtimes →
Overheads can exceed runtimes



Caching Policies

Caching Policies

Adding to the cache

Removing from the cache

Caching Policies

What functions do we memoize?

Removing from the cache

Caching Policies

What functions do we memoize?

How to evict cache entries?

Caching Policies

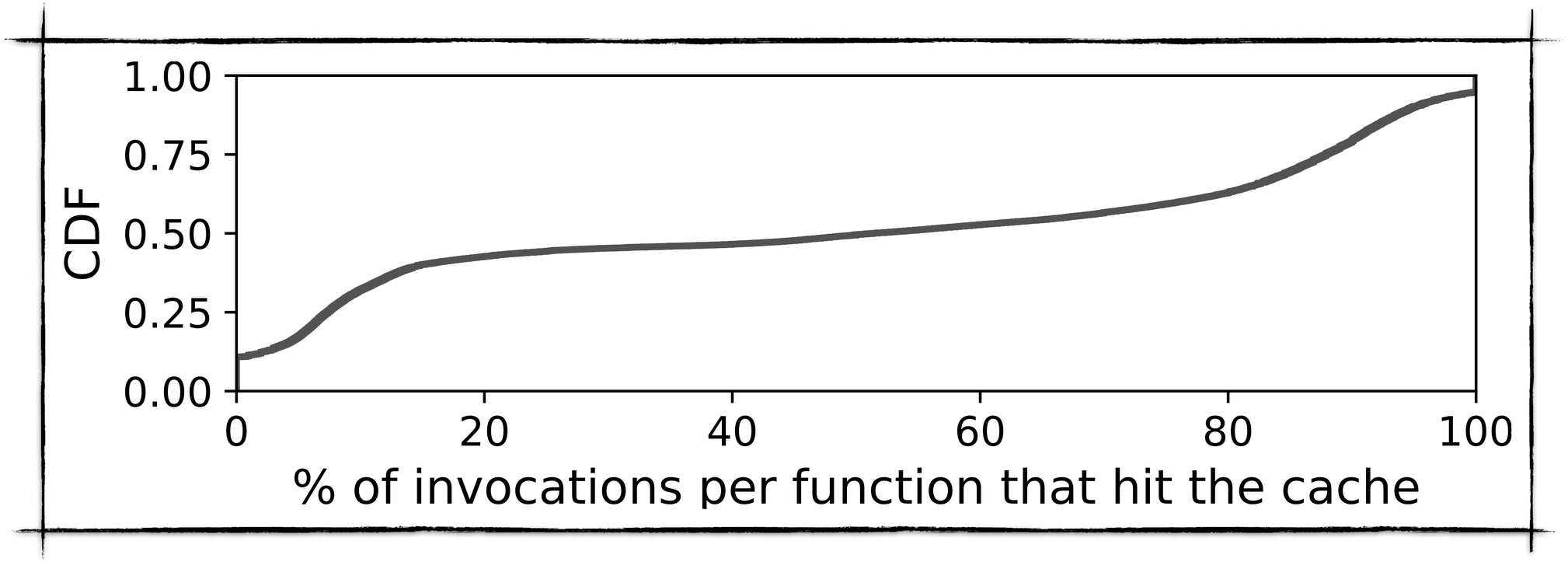
What functions do we memoize?

Given a function, what fraction of its invocations were memoized?

Caching Policies

What functions do we memoize?

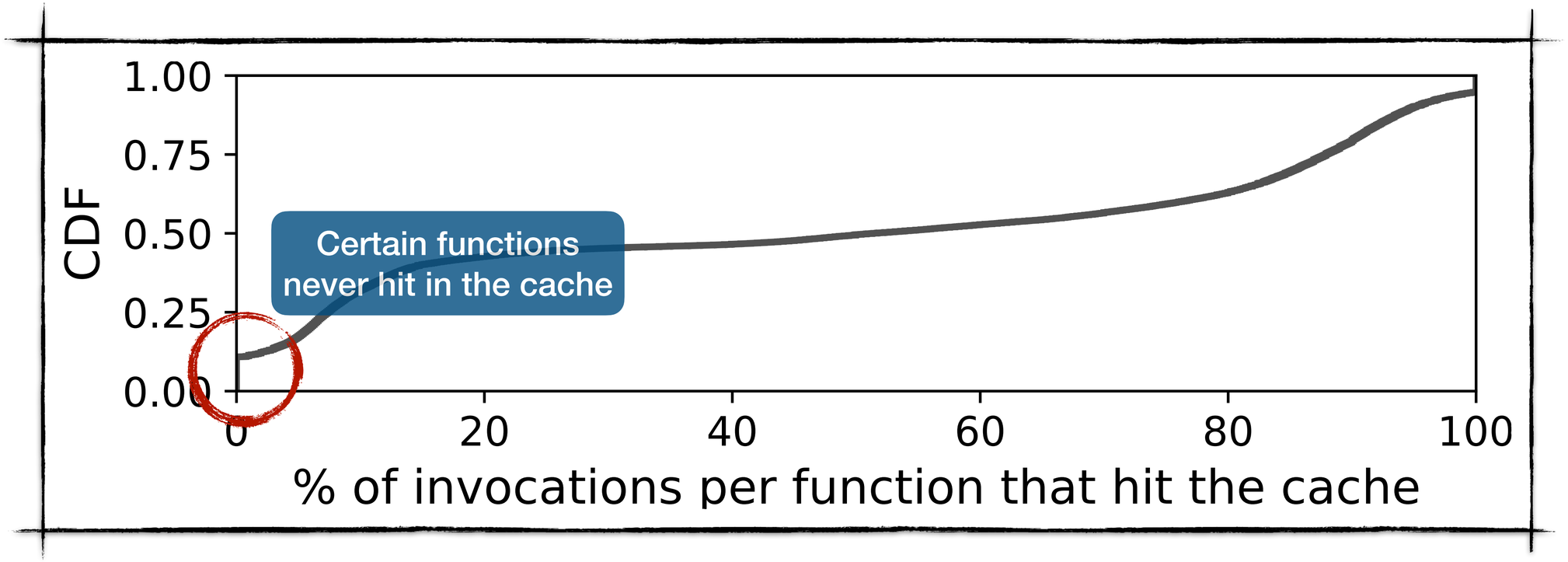
Given a function, what fraction of its invocations were memoized?



Caching Policies

What functions do we memoize?

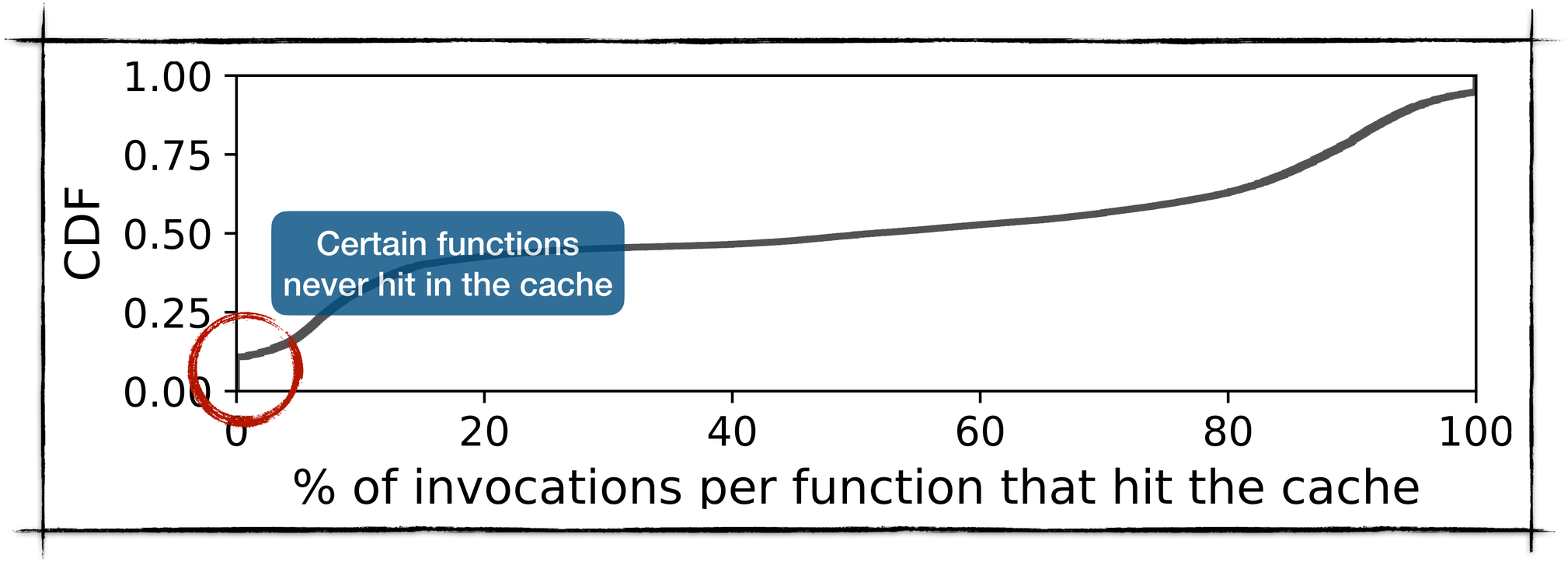
Given a function, what fraction of its invocations were memoized?



Caching Policies

What functions do we memoize?

Given a function, what fraction of its invocations were memoized?



Track hit rate with cache entries: Deactivate low hit-rate functions

Caching Policies

What functions do we memoize?

Given a function, what fraction of its invocations were memoized?

Large number of invocations → Queries can overwhelm resources

Track hit rate with cache entries: Deactivate low hit-rate functions

Caching Policies

Caching Policies

How to evict cache entries?

Caching Policies

How to evict cache entries?

Annotate cache entries with utility/potential benefits

Rank cache entries by utility

Caching Policies

Annotate cache entries with utility/potential benefits

Rank cache entries by utility

Caching Policies

Annotate cache entries with utility/potential benefits

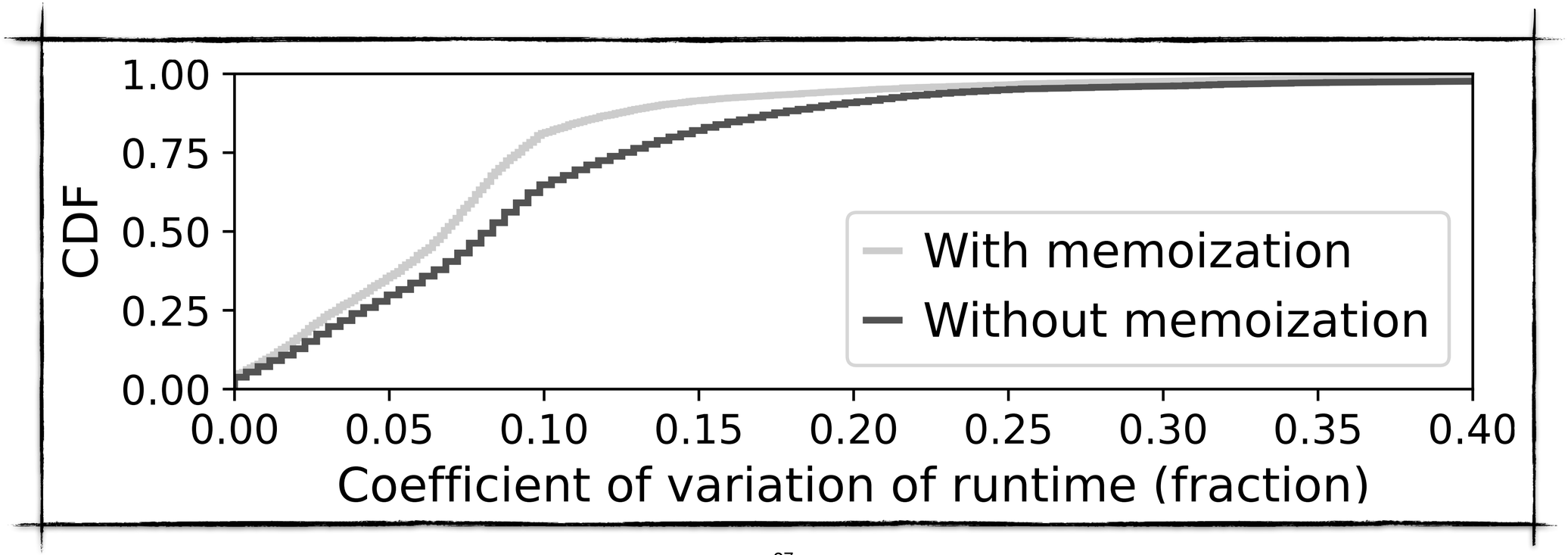
Rank cache entries by utility

How does the runtime of a particular function vary across its invocations?

Caching Policies

Annotate cache entries with utility/potential benefits → Rank cache entries by utility

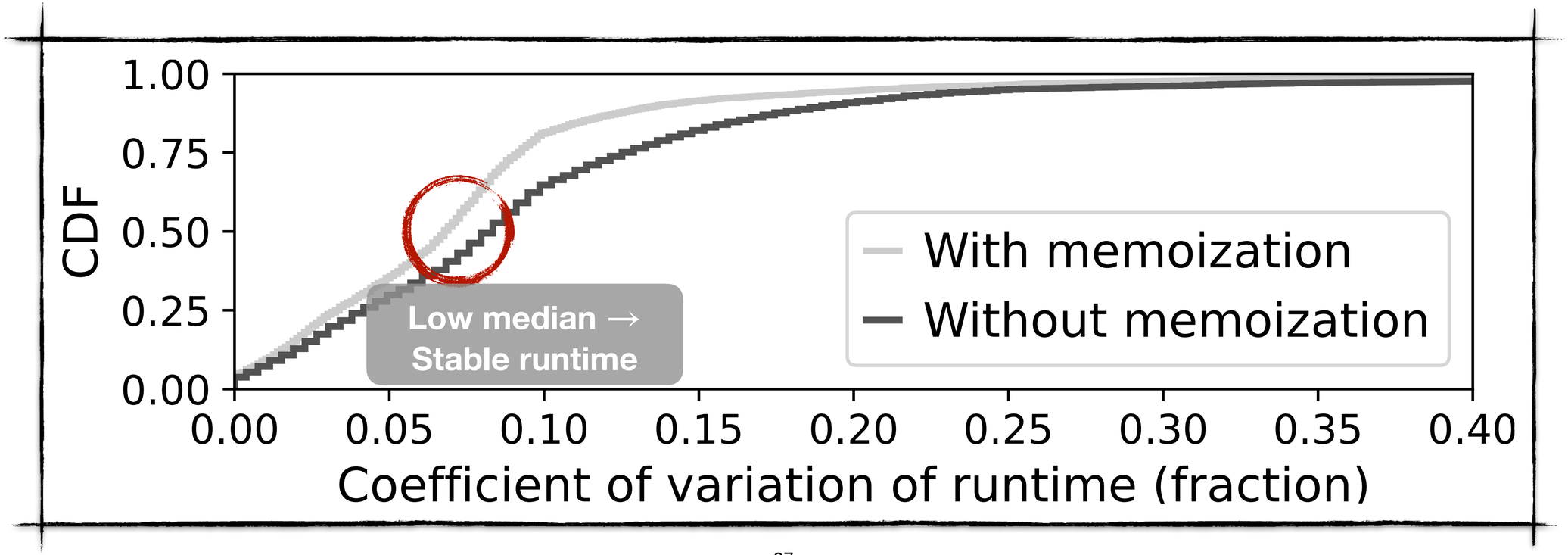
How does the runtime of a particular function vary across its invocations?



Caching Policies

Annotate cache entries with utility/potential benefits → Rank cache entries by utility

How does the runtime of a particular function vary across its invocations?



Caching Policies

Annotate cache entries with utility/potential benefits

Rank cache entries by utility

How does the runtime of a particular function vary across its invocations?

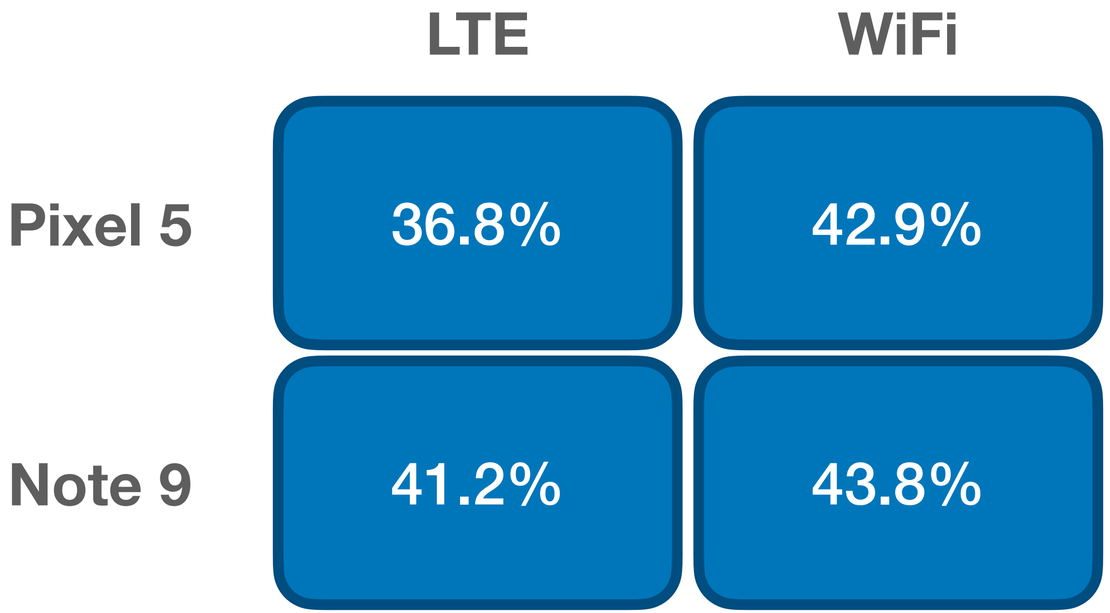
Low median → Stable runtime

Store first runtimes → Diff is the speedup

First memoized and unmemoized invocation

Computation Improvements

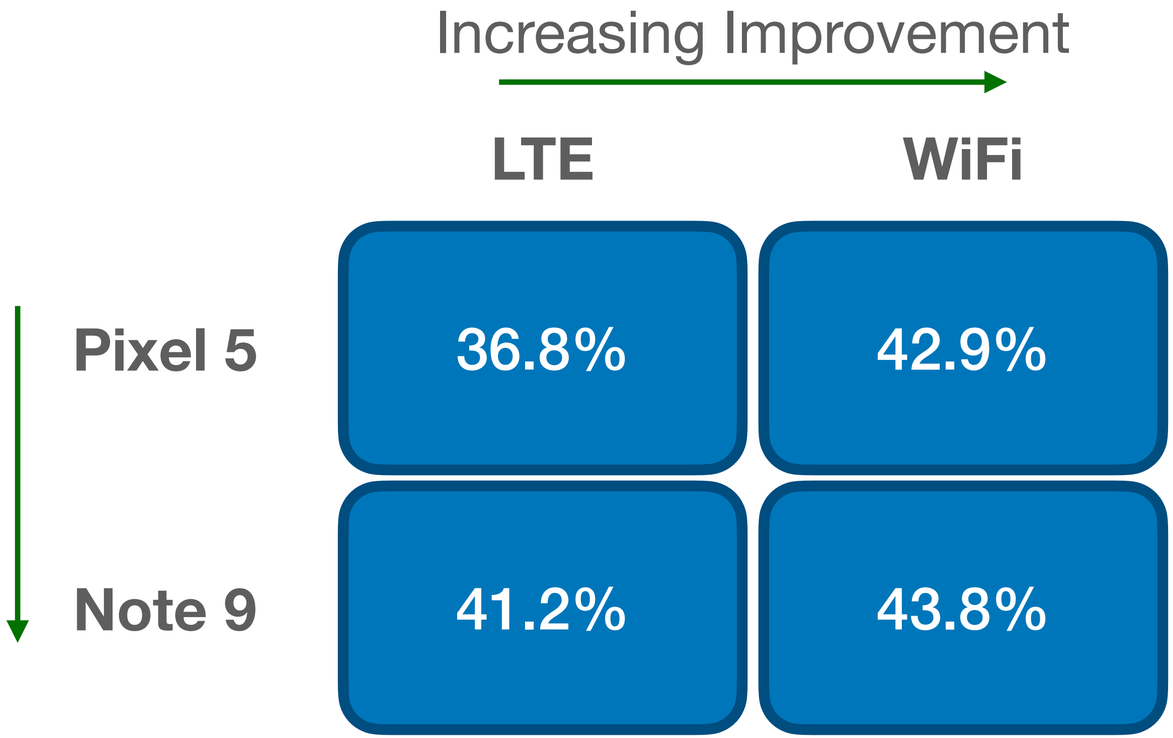
Devices: Note 9, Pixel 5. Networks: WiFi, LTE.



Median computation improvements per interaction

Computation Improvements

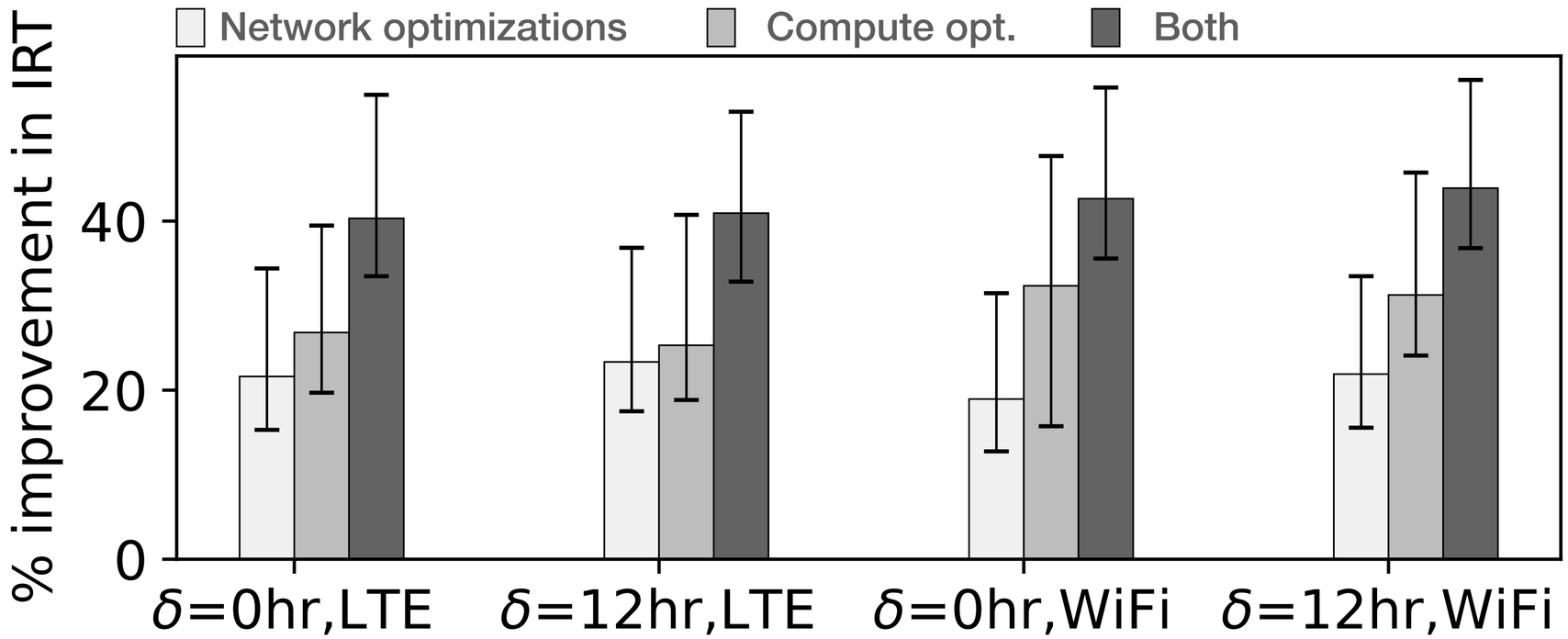
Devices: Note 9, Pixel 5. Networks: WiFi, LTE.



Median computation improvements per interaction

Performance Improvements

With Network and Compute optimizations



Caching in mobile apps

Summary

- Apps could cache content, just like web browsers and CDN. But this potential is not utilized optimally
    - Techniques exist to optimize caching and prefetching
    - For web content:
        - improve TTLs
        - prefetch just-in-time
    - For computations:
        - improve cache lookup time with lookaheads
        - optimized cache entries
        - domain specific caching policies

More content on compute caching (Floo)

Floo: Kinds of local computations

Floo: Kinds of local computations

Compute bottlenecks are critical

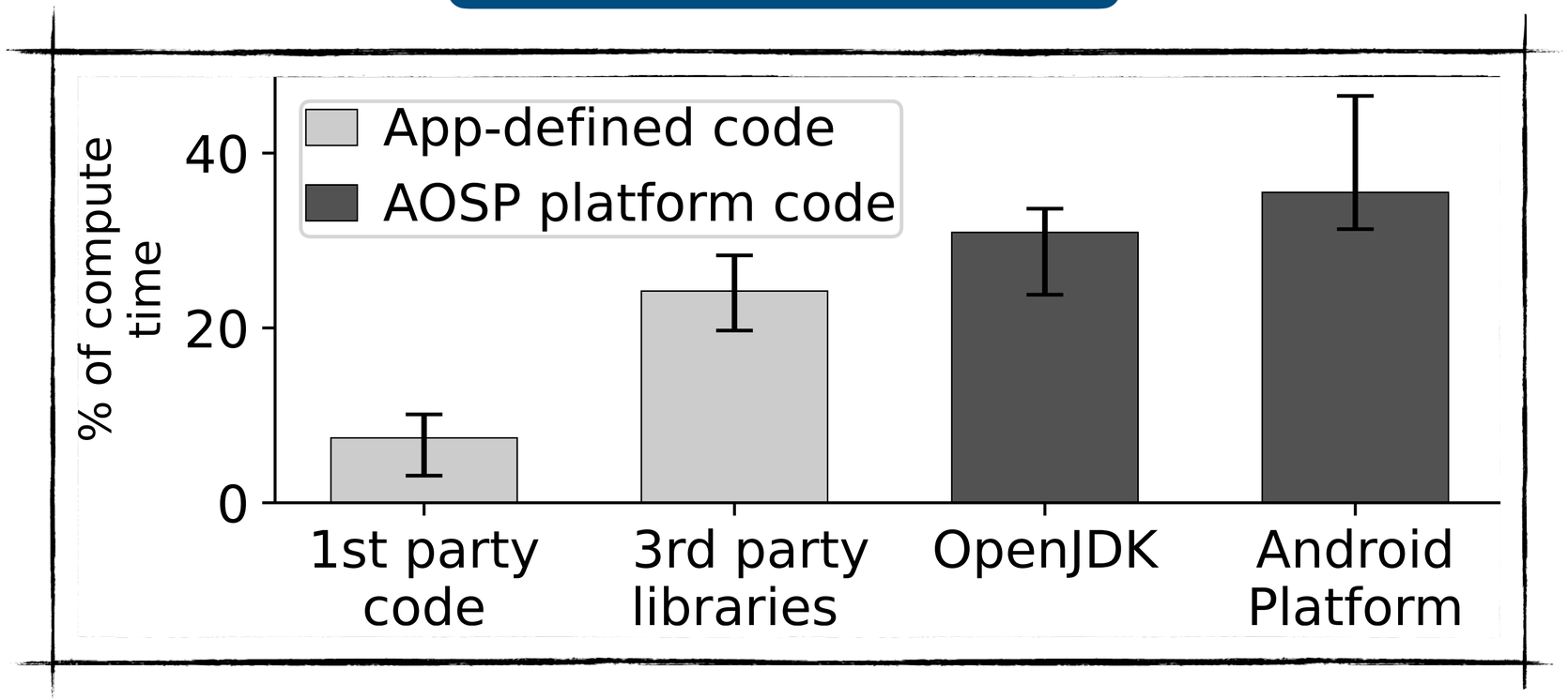
Floo: Kinds of local computations

Compute bottlenecks are critical

But what are these computations exactly?

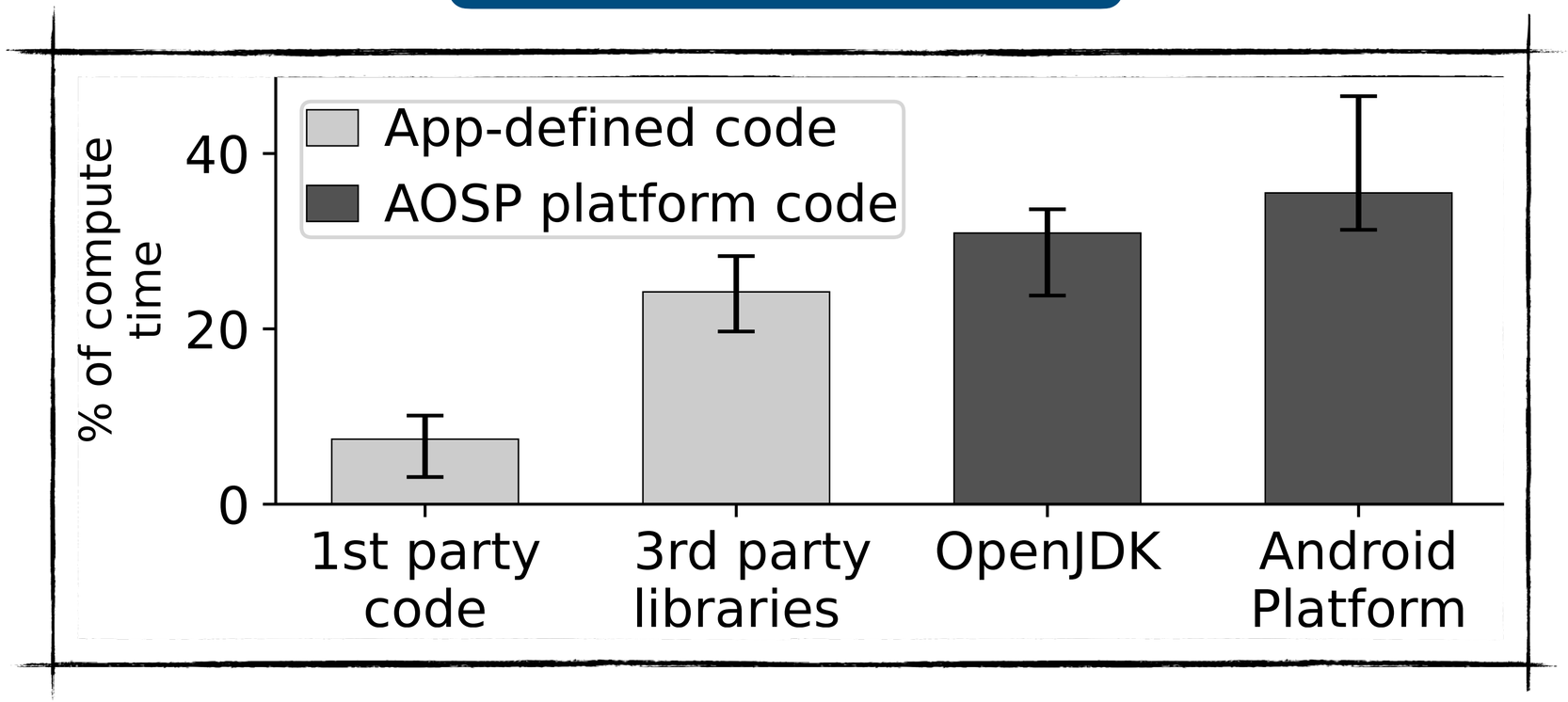
Floo: Kinds of local computations

But what are these computations exactly?



Floo: Kinds of local computations

But what are these computations exactly?



App-defined *and* AOSP functions are substantial

Implementing memoization

Implementing memoization

Exclude functions using nondeterministic or native APIs

```
Fn f() {
  return Time.now() + 1
}
```

Implementing memoization

Exclude functions using nondeterministic or native APIs

```
Fn f() {
  return Time.now() + 1
}
```

Correctness

Implementing memoization

Exclude functions using nondeterministic or native APIs

```
Fn f() {
  return Time.now() + 1
}
```

Correctness

Memo writes include side effects

```
Fn f() {
  this.var = 5;
  print(this.var);
}
```

Implementing memoization

Exclude functions using nondeterministic or native APIs

```
Fn f() {
  return Time.now() + 1
}
```

Memo writes include side effects

```
Fn f() {
  this.var = 5;
  print(this.var);
}
```

Comprehensive read state in addition to parameters

```
Fn f(int param) {
  if (this.x == 5)
    return 10;
  else return param;
}
```

Correctness

Implementing memoization

Exclude functions using nondeterministic or native APIs

```
Fn f() {
  return Time.now() + 1
}
```

Correctness

Memo writes include side effects

```
Fn f() {
  this.var = 5;
  print(this.var);
}
```

Comprehensive read state in addition to parameters

```
Fn f(int param) {
  if (this.x == 5)
    return 10;
  else return param;
}
```

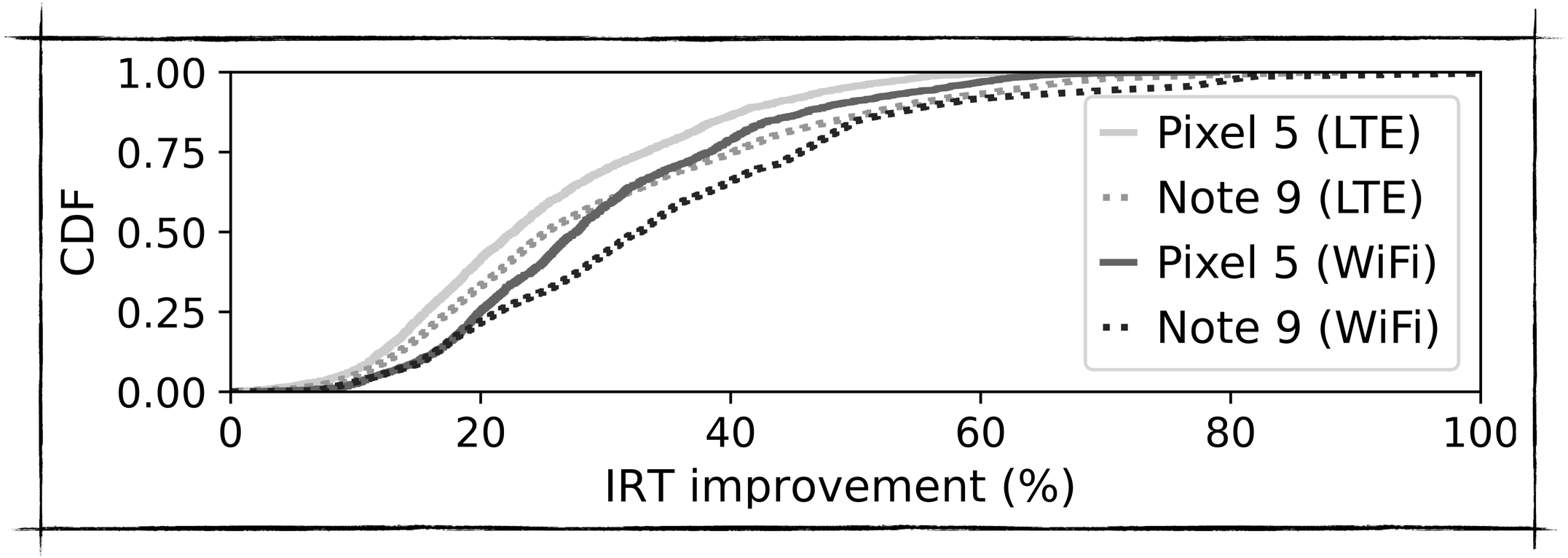
Support for impure functions

Evaluation

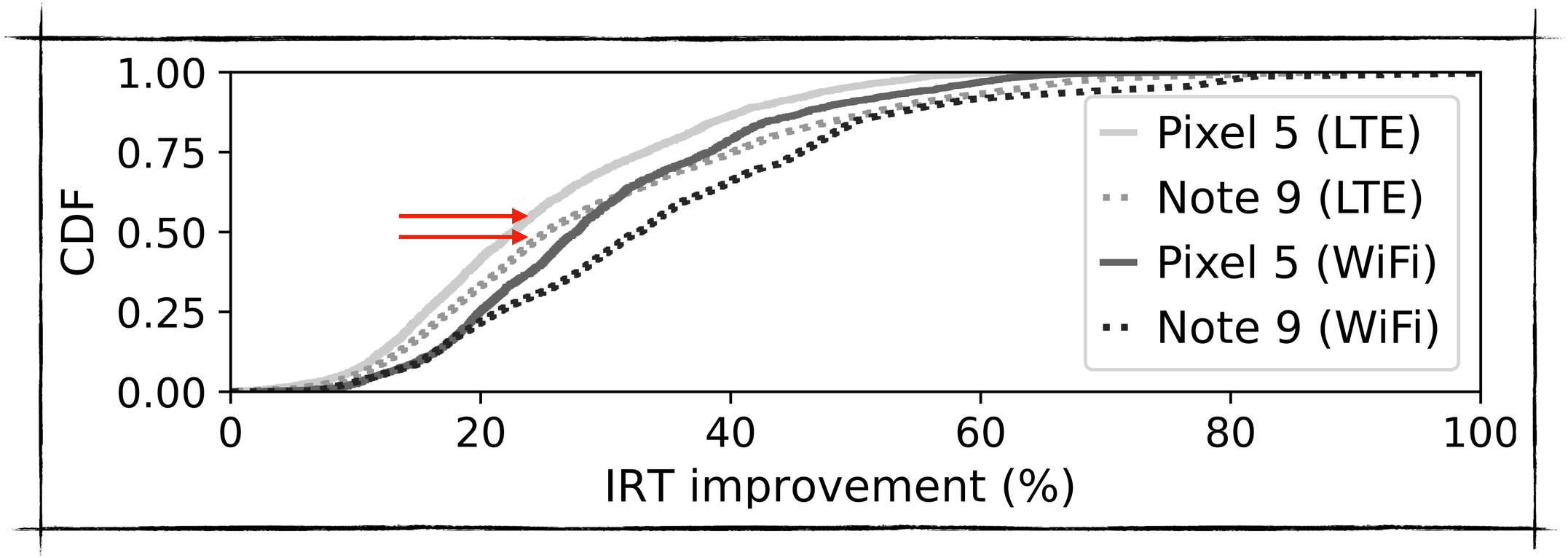
Floo - mobile computation caching

- How does Floo **improve the compute** time?
- How does this improvement translate into **responsiveness**?
- How does Floo perform with **fewer device resources**?
- Do Floo's transformations **preserve correct app behavior**?
- What is the **source of Floo's wins**?
- How does Floo compare against **prior work**?
- Does the improvement change based on **user access delays**?
- What are the **typical app characteristics** that enable memoization?

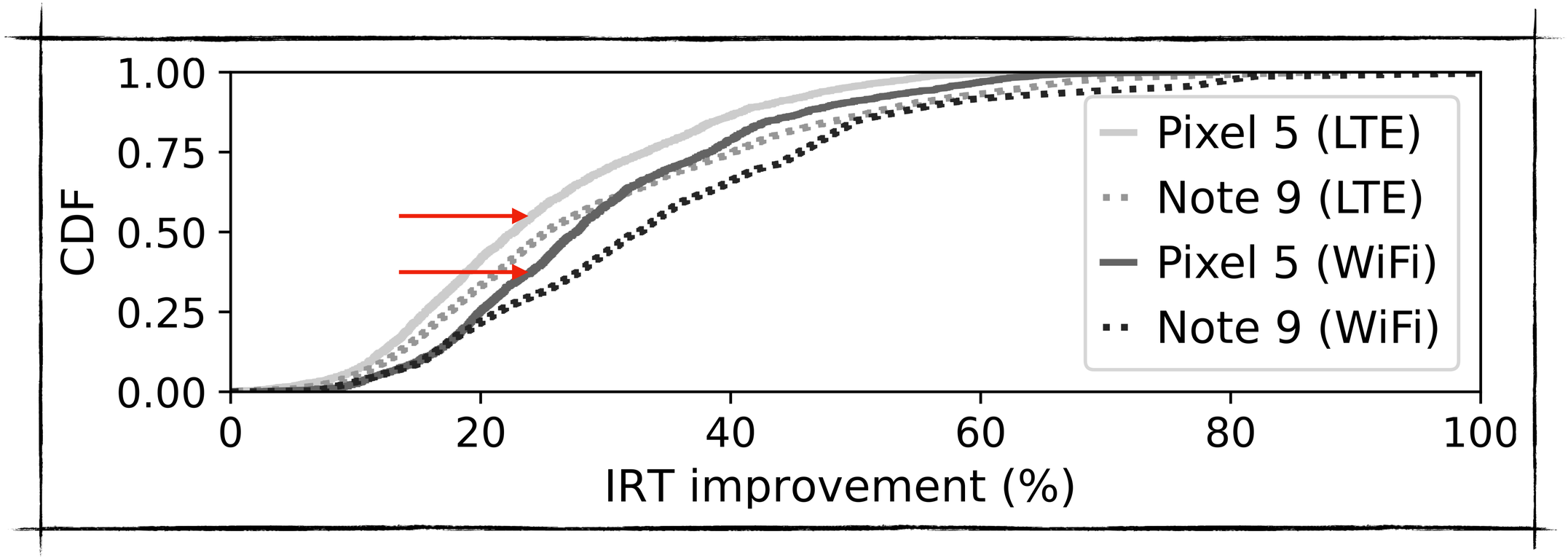
Responsiveness (IRT) Improvements



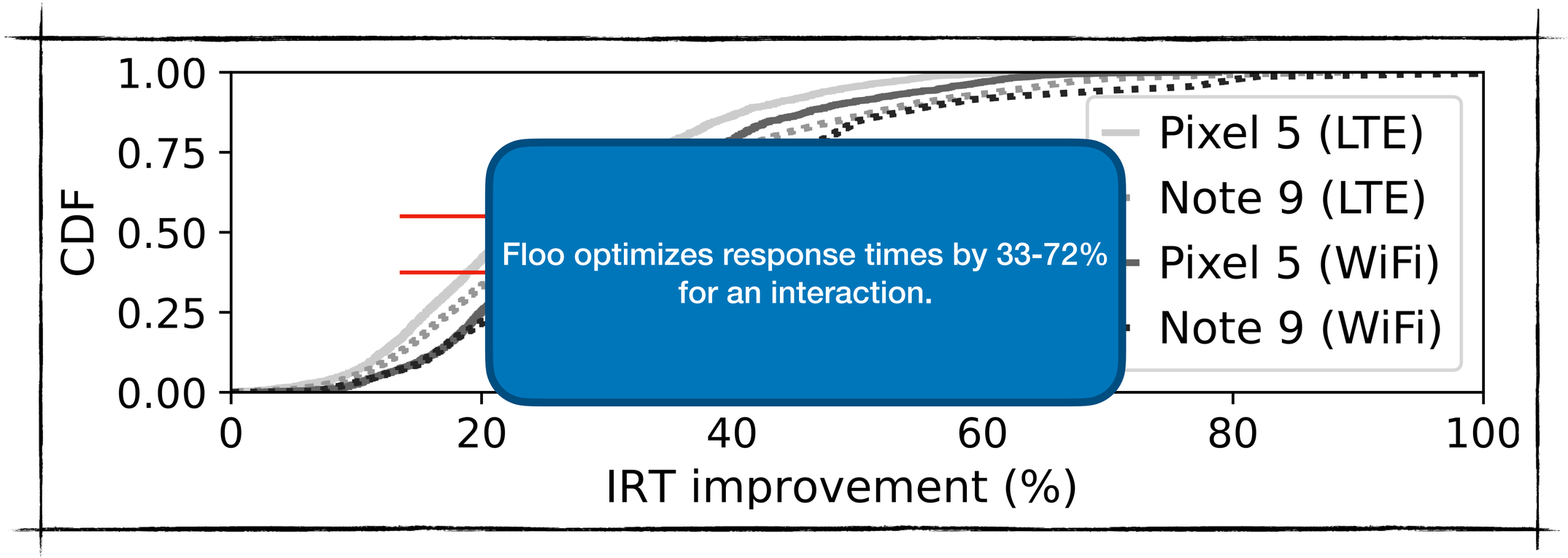
Responsiveness (IRT) Improvements



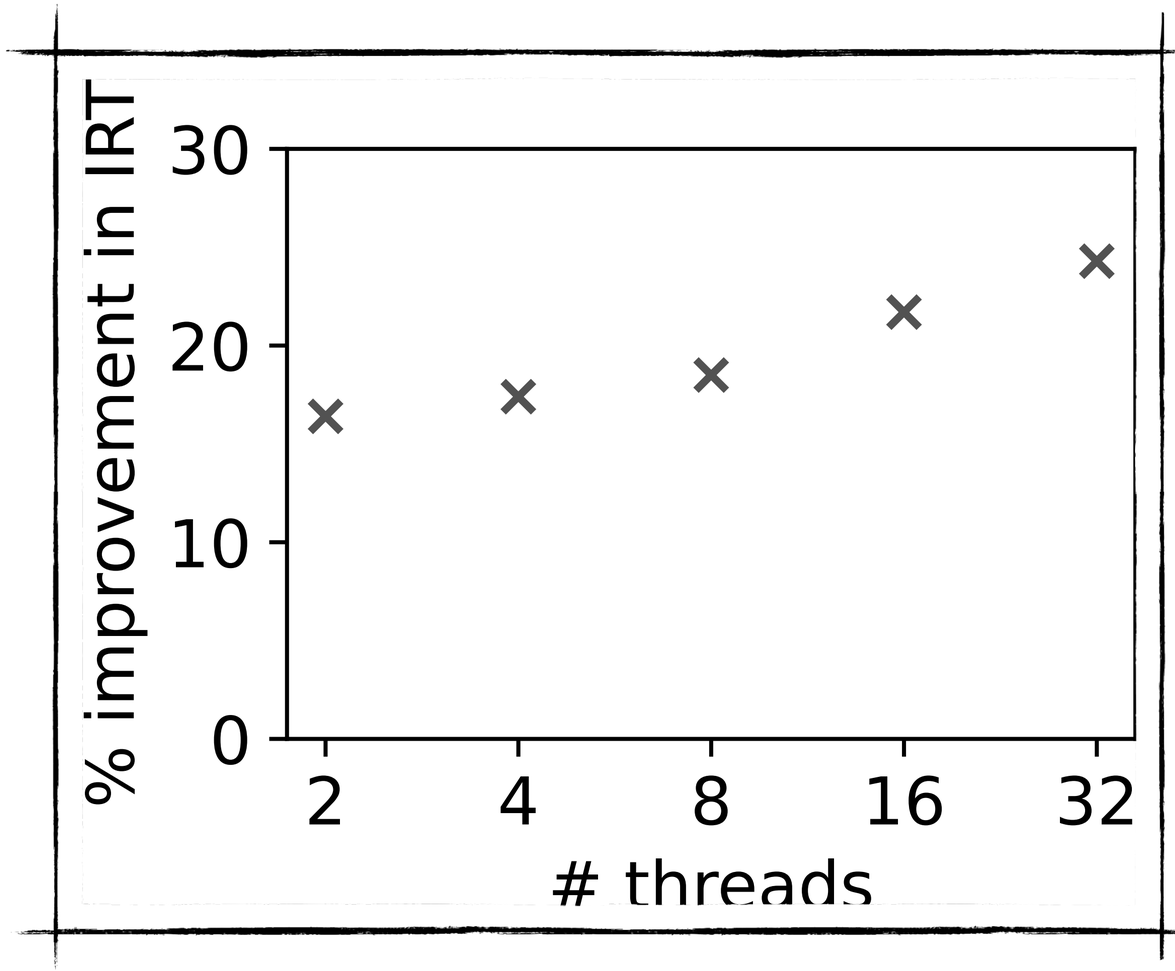
Responsiveness (IRT) Improvements

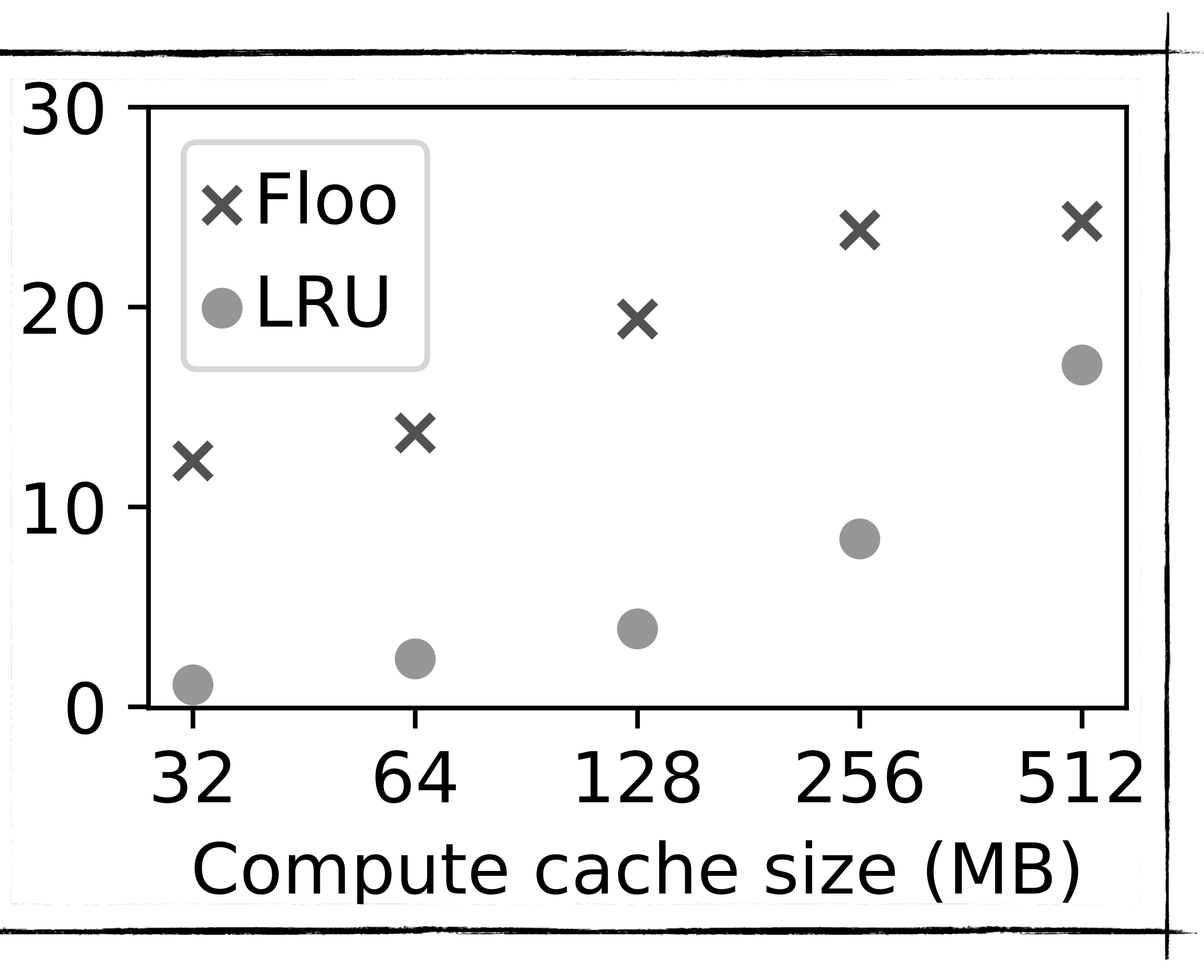


Responsiveness (IRT) Improvements



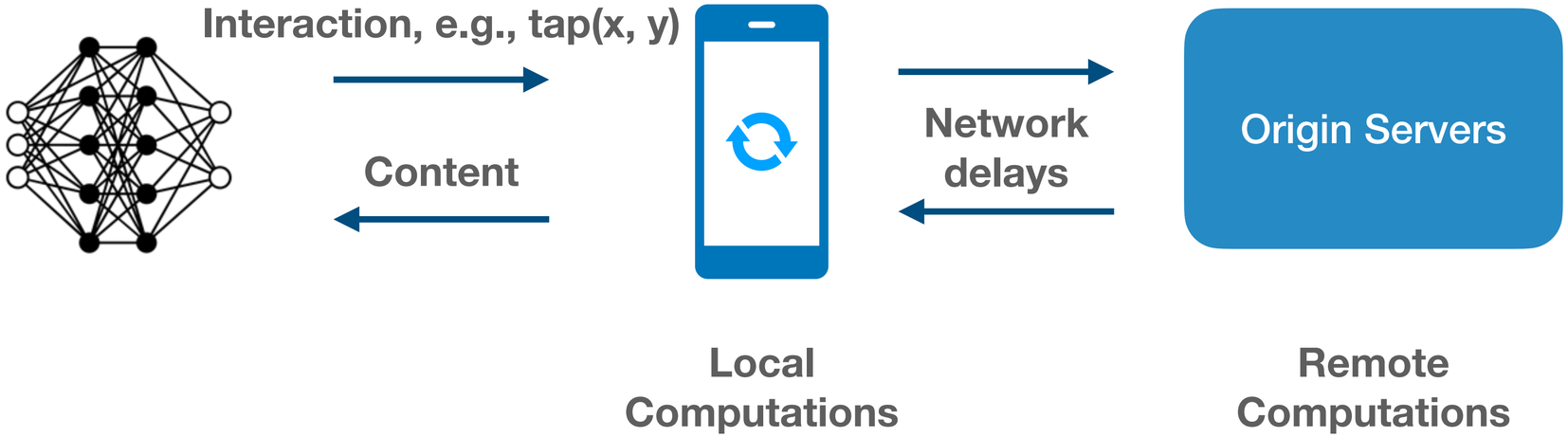
Varying device resources





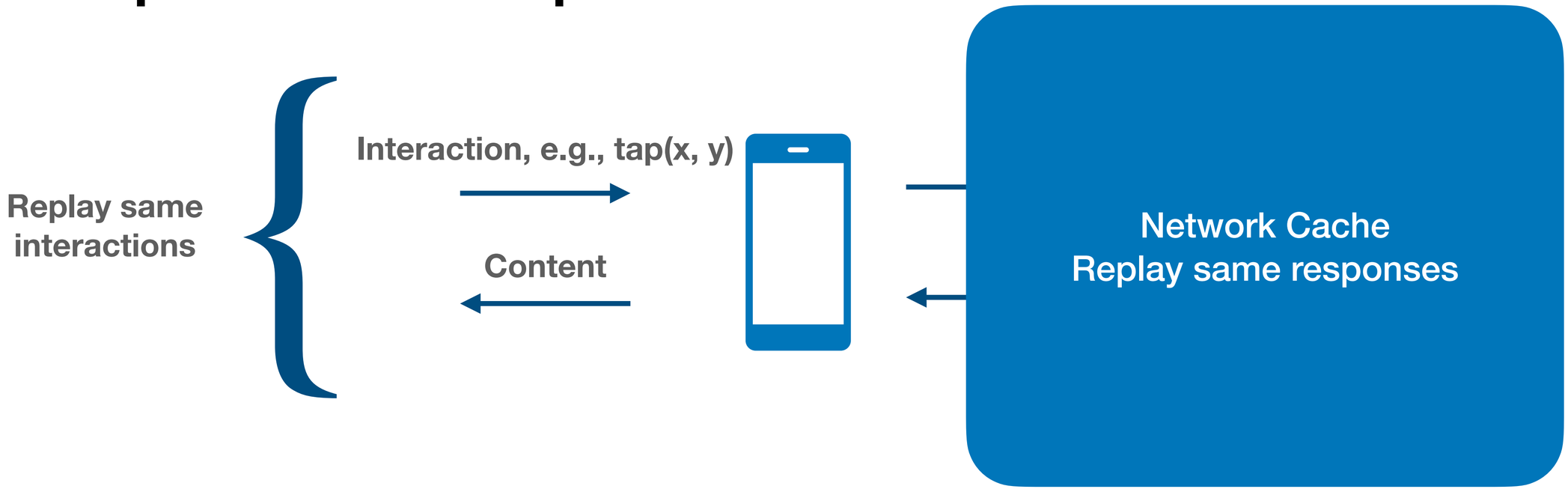
Correctness

Does Floo preserve developer intent?



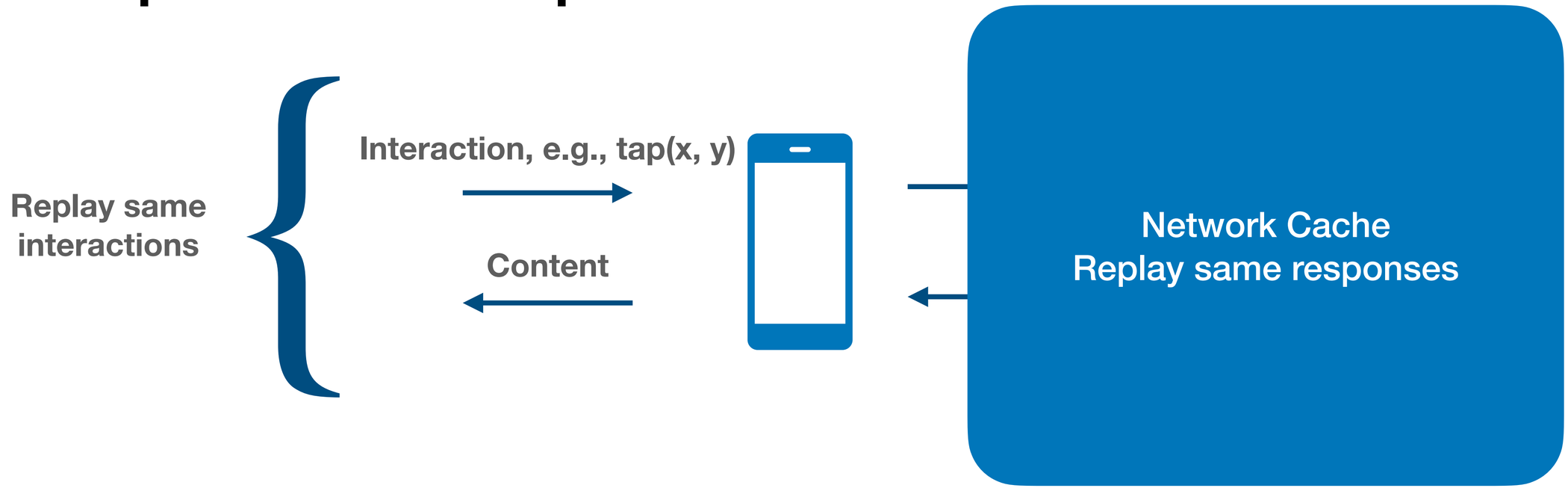
Correctness

Does Floo preserve developer intent?



Correctness

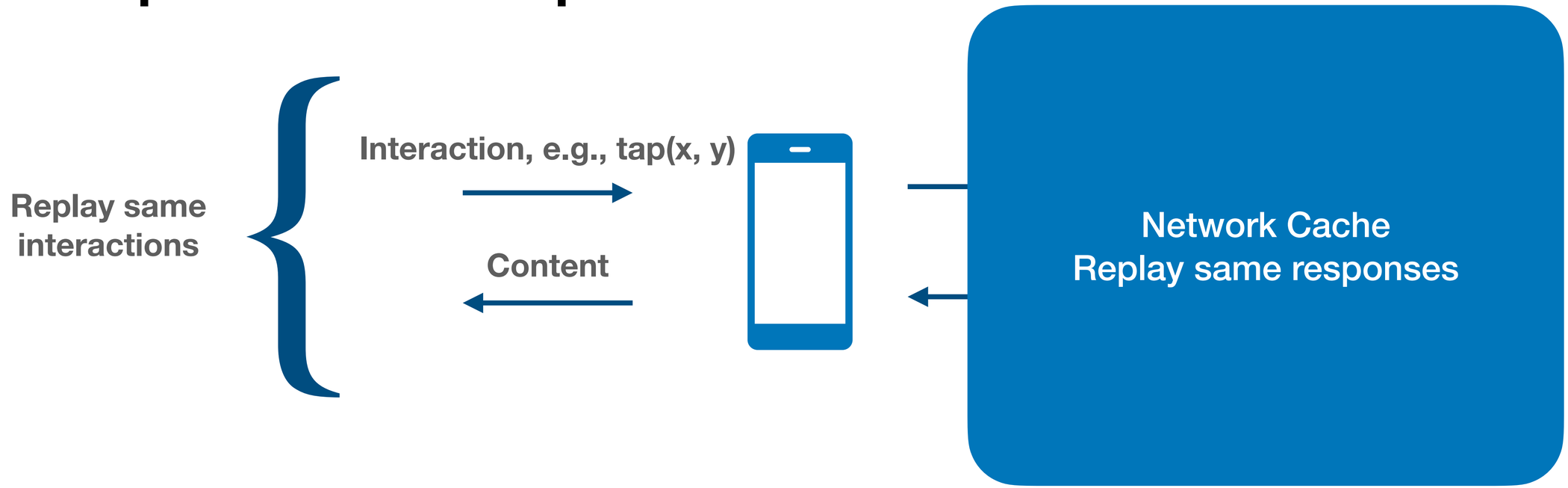
Does Floo preserve developer intent?



- Same server side content
- Same user interactions
- Single thread to prevent races
- Enforce determinism

Correctness

Does Floo preserve developer intent?



- Same server side content
- Same user interactions
- Single thread to prevent races
- Enforce determinism

Compare behavior with and without Floo

Correctness

Does Floo preserve developer intent?

Correctness

Does Floo preserve developer intent?

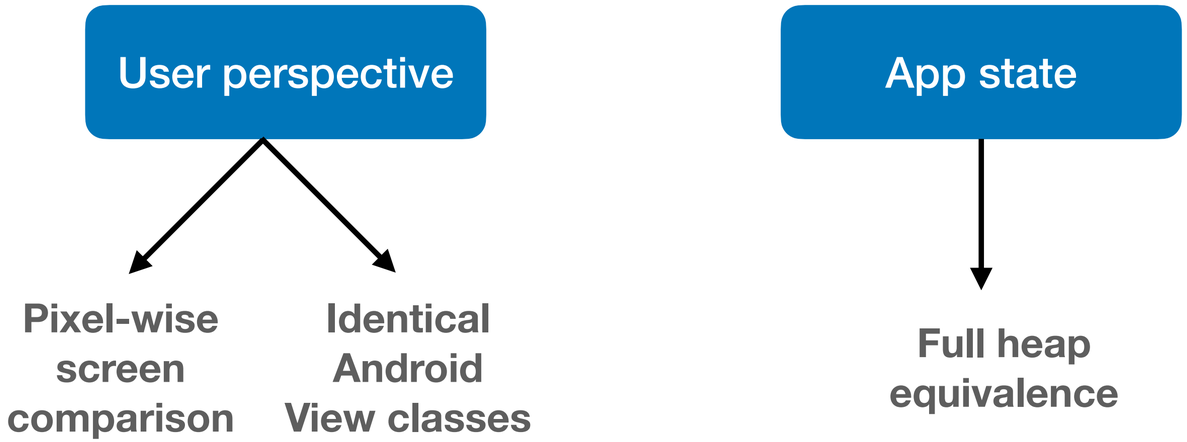
User perspective

App state

Correctness

Does Floo preserve developer intent?

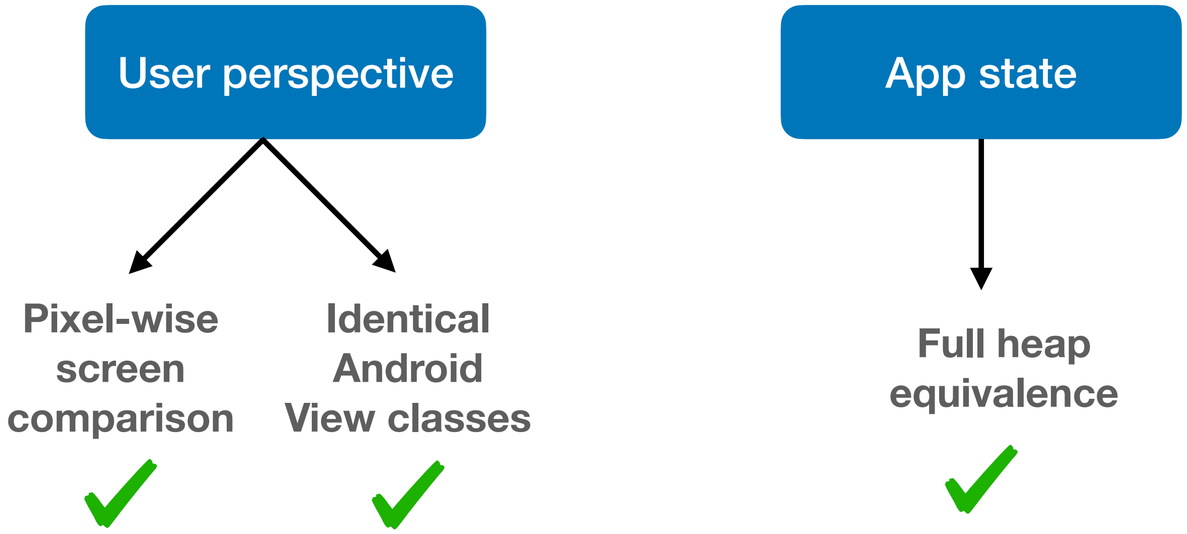
Enforce determinism
Force single thread



Correctness

Does Floo preserve developer intent?

Enforce determinism
Force single thread

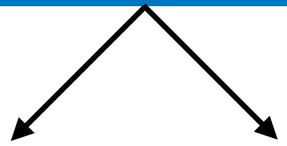


Correctness

Does Floo preserve developer intent?

Enforce determinism
Force single thread

User perspective



Pixel-wise screen comparison

Identical Android View classes

Match after each interaction

App state

Full heap equivalence

Match after a series of interactions

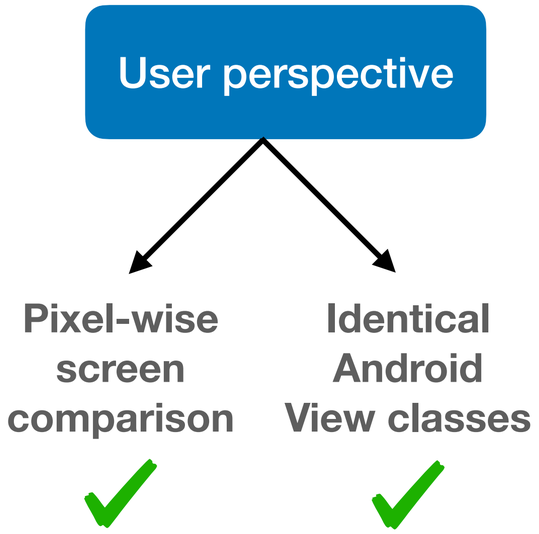
Enforce determinism
Allow multiple threads

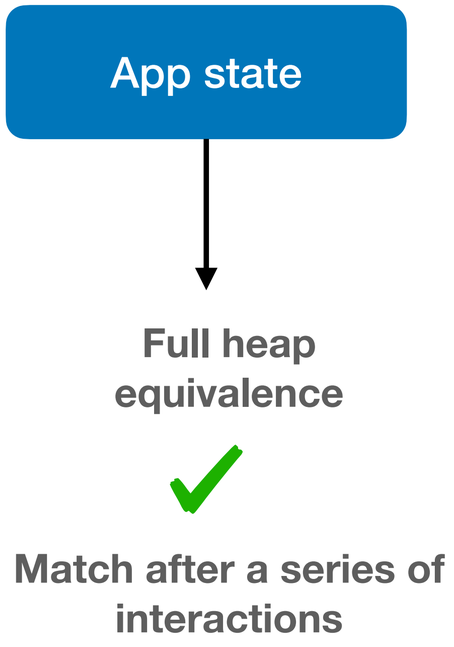
App state

Correctness

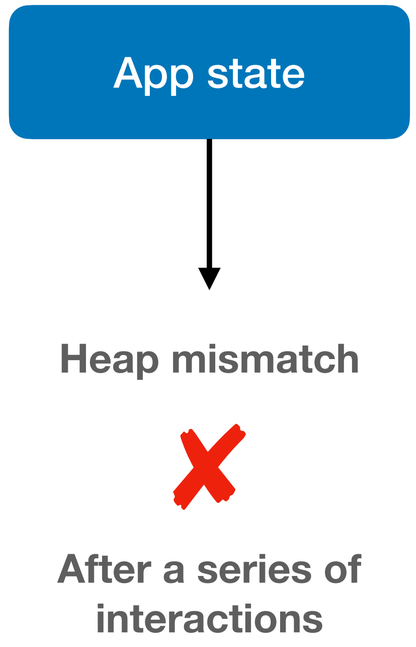
Does Floo preserve developer intent?

Enforce determinism
Force single thread





Enforce determinism
Allow multiple threads



Correctness

Investigating app state mismatches

Correctness

Investigating app state mismatches

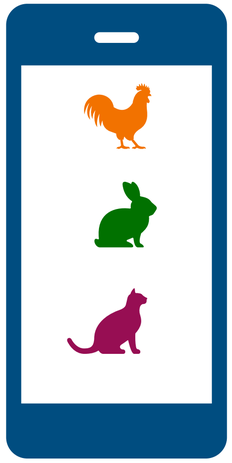
Thread schedule variations

Correctness

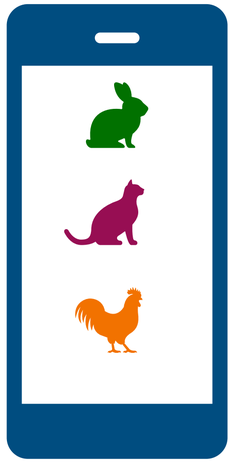
Investigating app state mismatches

Thread schedule variations

Run 1:
Without Floo



Run 2:
With Floo



Heap and potential visual mismatch across two runs

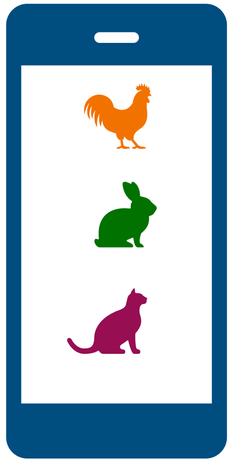
Correctness

Investigating app state mismatches

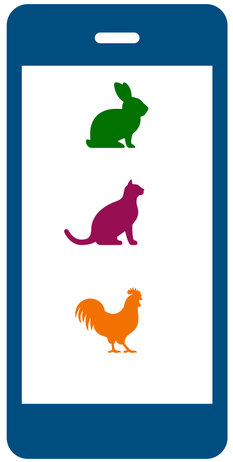
Thread schedule variations

Undetected synchronizations

Run 1:
Without Floo



Run 2:
With Floo



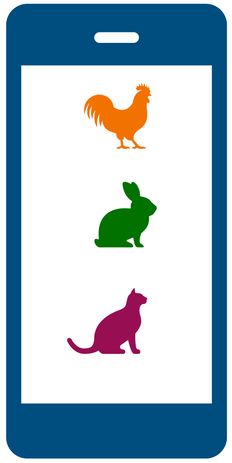
Heap and potential visual mismatch across two runs

Correctness

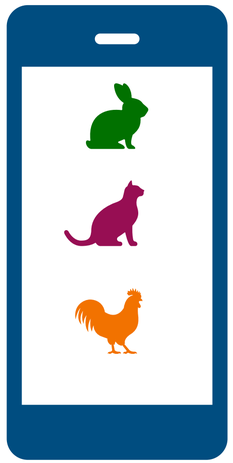
Investigating app state mismatches

Thread schedule variations

Run 1:
Without Floo



Run 2:
With Floo



Heap and potential visual mismatch across two runs

Undetected synchronizations

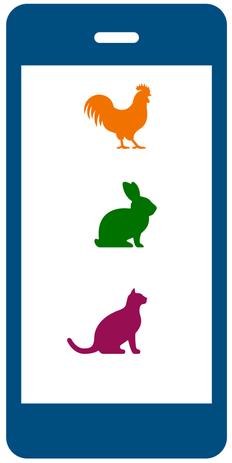
```
while(!x.contentReady); // busy wait
y = x.contentReady; // y must be true
```

Correctness

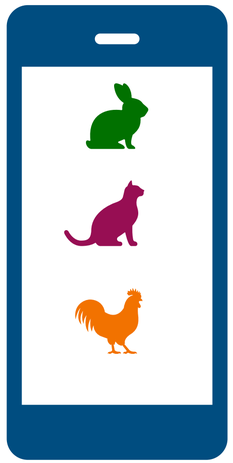
Investigating app state mismatches

Thread schedule variations

Run 1:
Without Floo



Run 2:
With Floo



Heap and potential visual mismatch across two runs

Undetected synchronizations

```
while(!x.contentReady); // busy wait
y = x.contentReady; // y must be true
```

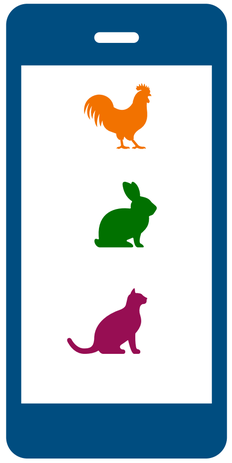
We entered the function with x.contentReady as false and set y to be true

Correctness

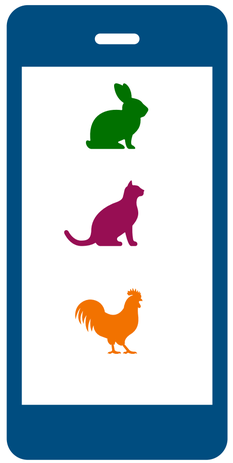
Investigating app state mismatches

Thread schedule variations

Run 1:
Without Floo



Run 2:
With Floo



Heap and potential visual mismatch across two runs

Undetected synchronizations

```
while(!x.contentReady); // busy wait
y = x.contentReady; // y must be true
```

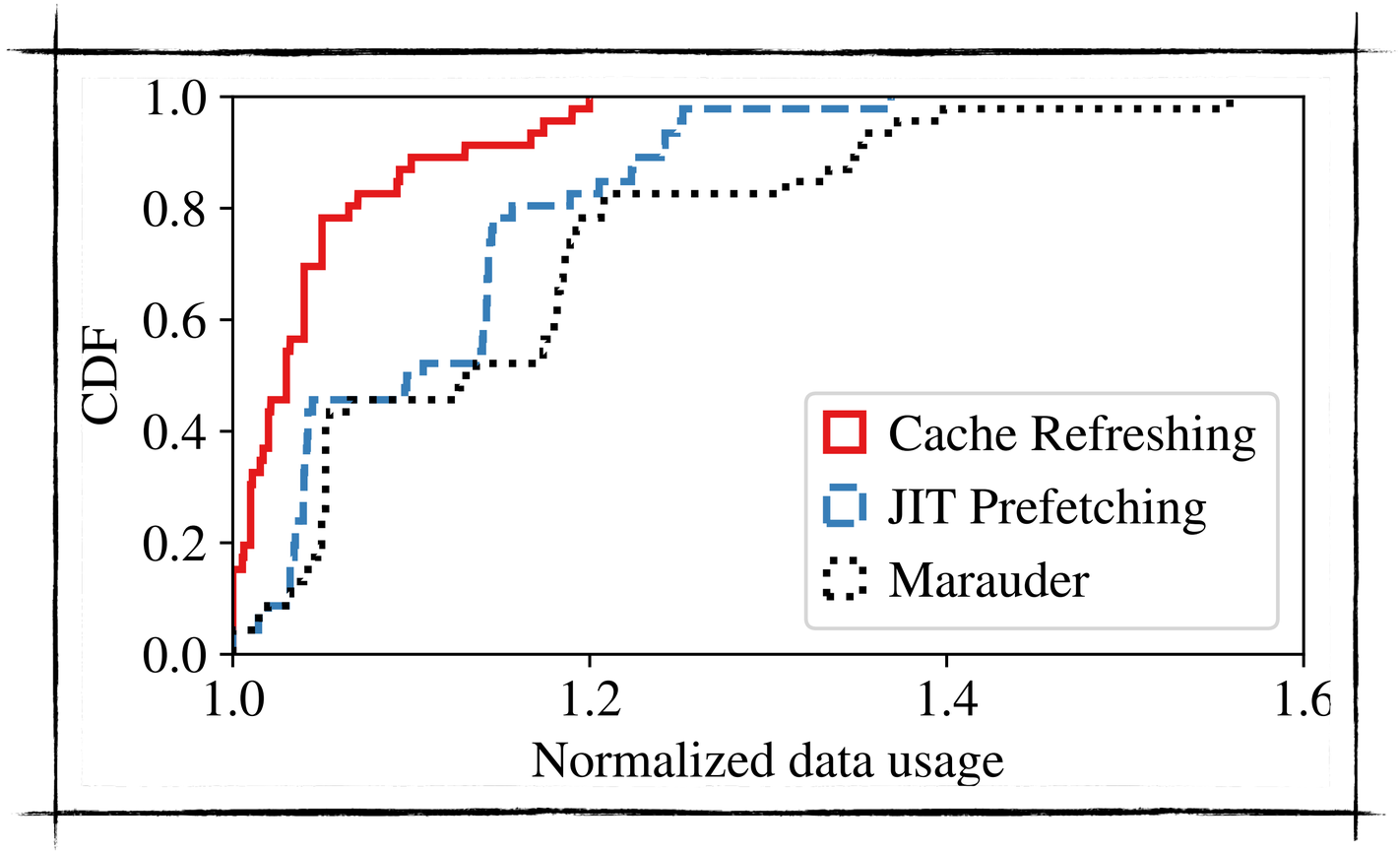
We entered the function with x.contentReady as false and set y to be true

```
if x.contentReady is false
set y=true
```

Incorrect behavior

More content on mobile web caching (Marauder)

Bandwidth overheads



Constituent benefits