# UNIC: Secure Deduplication of General Computations

Yang Tang *and* Junfeng Yang
Columbia University



The world's data is fast exploding…

A significant portion of the data is redundant.

Data deduplication can hugely save storage and simplify management.

Not only is the data redundant…

The computations on top of the data are also redundant.

Can we deduplicate computations as well?



# Challenges

How can we deduplicate computations done by mutually distrusting users?

- Users can’t put wrong results in the cache.
- Users can’t query the cache if they shouldn’t.

Don’t authenticate users. Authenticate computations!

The UNIC way: leverage code attestation.

# More challenges

How can we deduplicate general computations?

*Examples: virus scanning, compiling source code, compression, big-data analysis, CommonCrawl…*

Impossible to understand every computation.

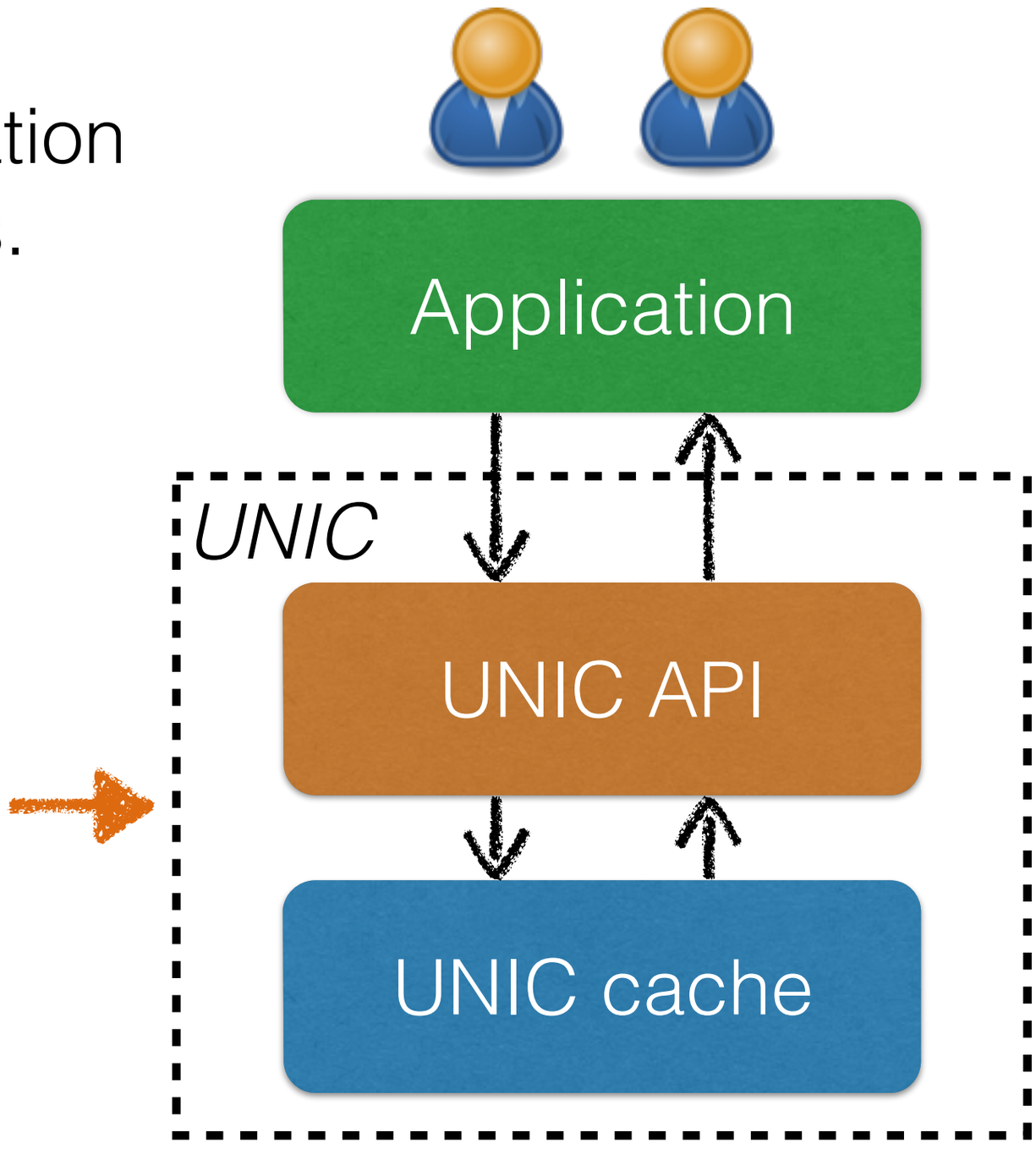
With developer's support, much more can be done.

The UNIC way: provide a simple yet expressive API.

# UNIC overview

UNIC: secure deduplication of general computations.

Code attestation for integrity and secrecy



# Outline

- Introduction
- Security design
- Example
- UNIC API
- Evaluation

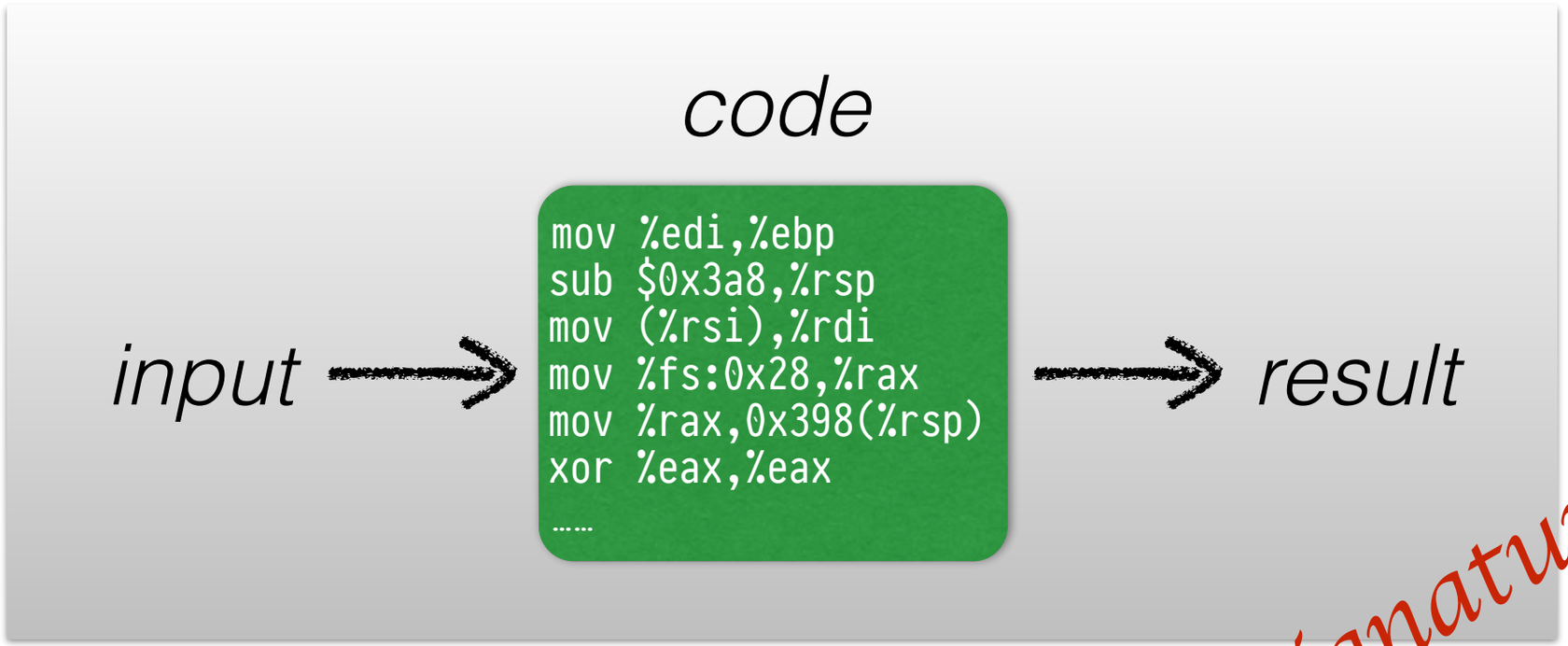
# Threats

- Cache poisoning
  - A malicious user may try to put wrong results.
- Query forging
  - A malicious user may try to query entries in the result cache that she cannot access.

UNIC leverages code attestation to enforce both integrity and secrecy of the result cache.

# Code attestation

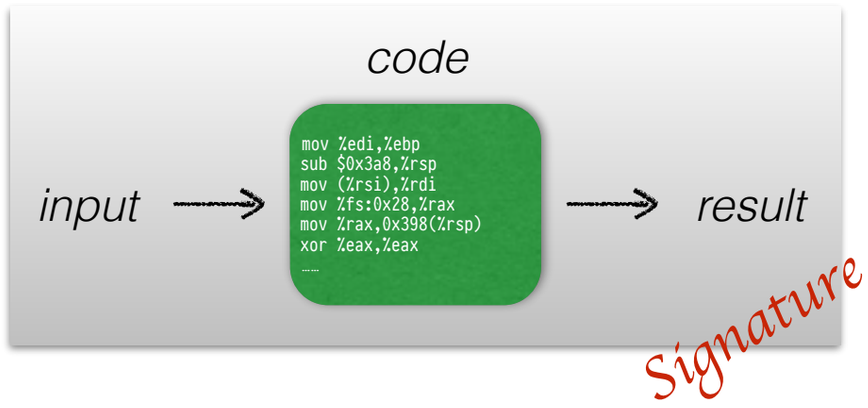
A computation is…




Code attestation cryptographically binds the *result* with the *code* and *input* that produced the *result*.

# UNIC design

UNIC computes…




(1) *result* = *code*(*input*)

Run *code* on *input* to compute *result*.

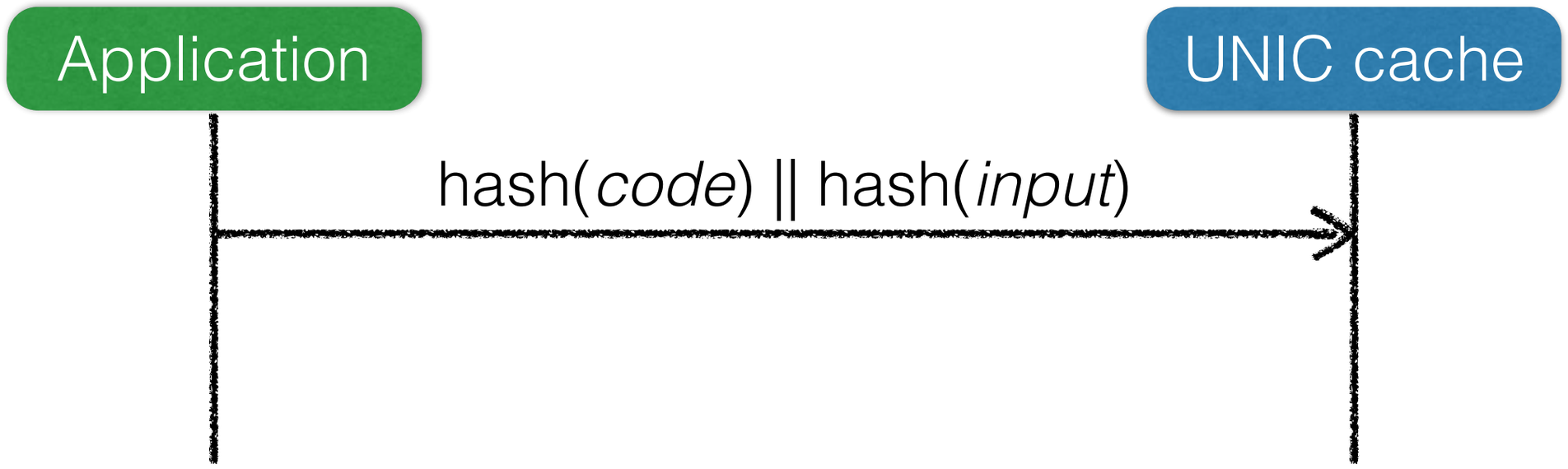
(2) *sig* = HMAC(hash(*code*) || hash(*input*) || *result*, *K*)

Bind *result* with *code* and *input*.

*K* is distributed among the trusted OSes.

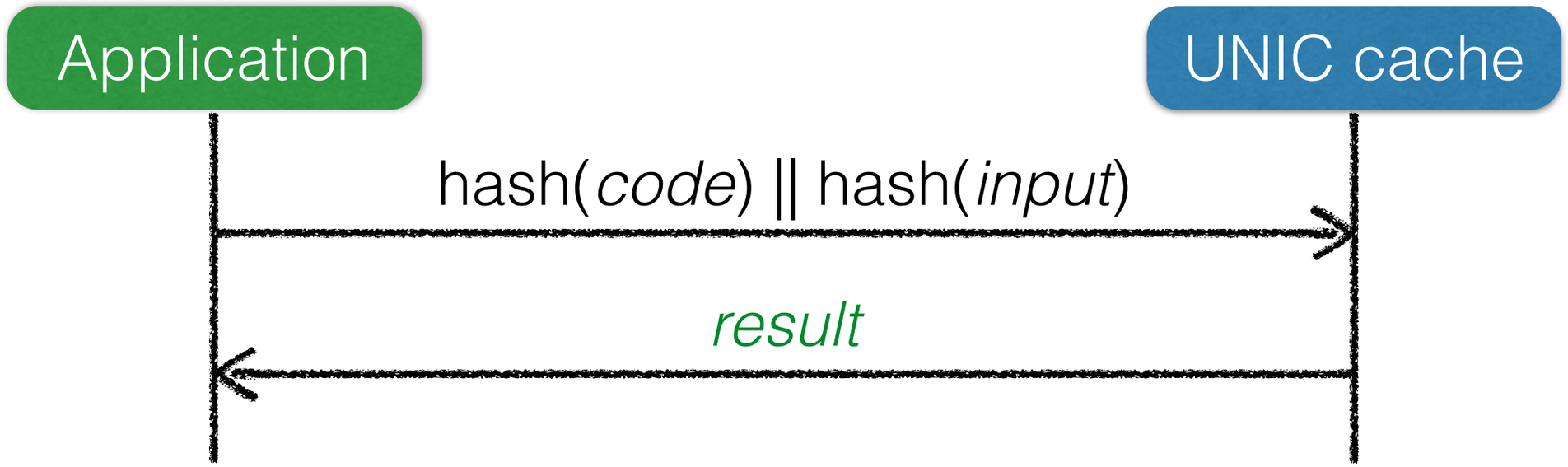
# UNIC protocol

When an application wants to compute *code*(*input*)…



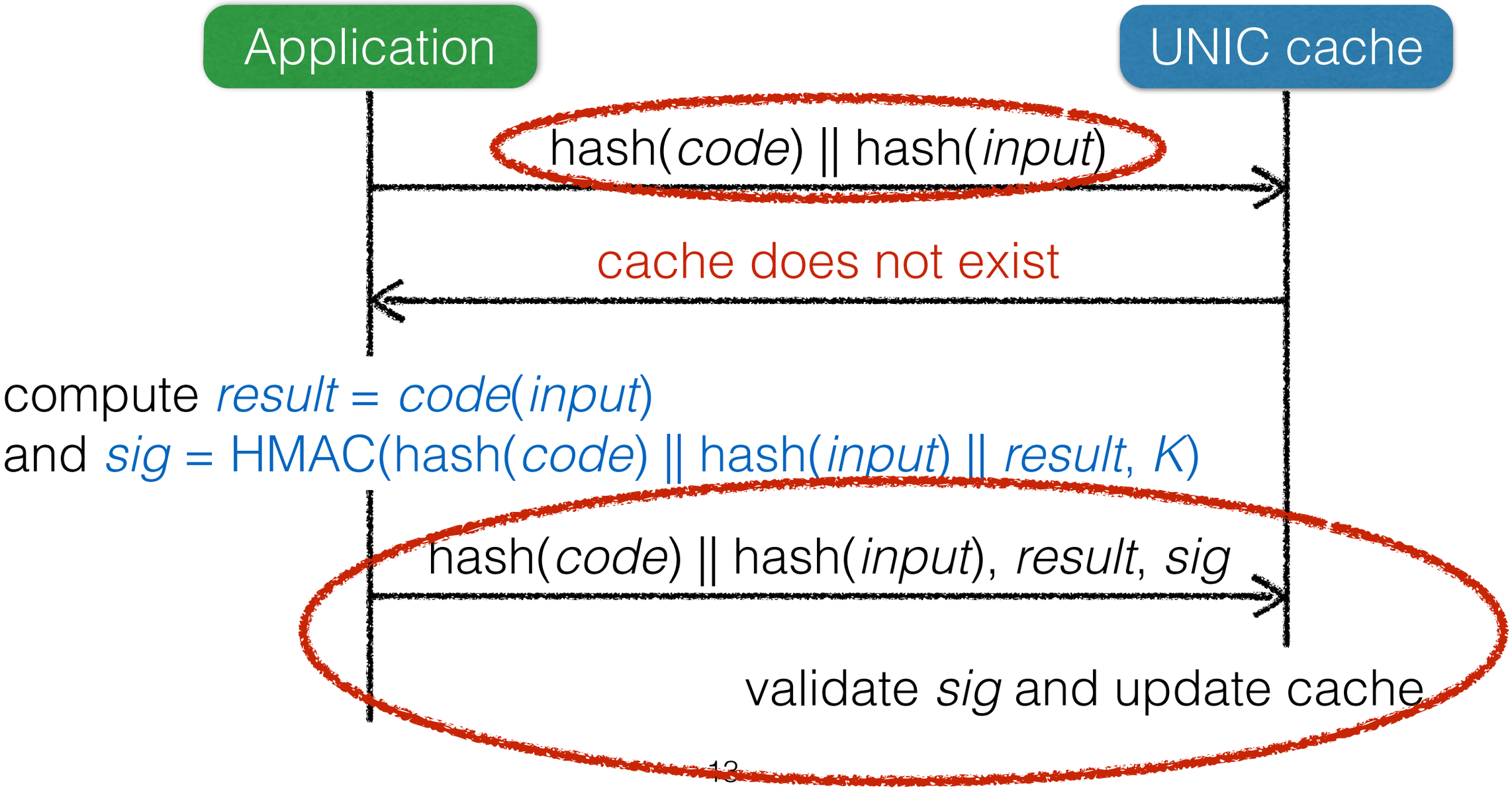
# UNIC protocol

If cache hits…



# UNIC protocol

If cache misses…



# Example

```
void simple_virus_scanner(file, options) {
  buffer = read(file);
  result = scan_signature(buffer, options);
  print(result);
}
```

*A simple virus scanning application*

# Example

```
void simple_virus_scanner(file, options) {
  buffer = read(file);
  if (exists(scan_signature, buffer, options)) {
    result = get(scan_signature, buffer, options);
  } else {
    result = scan_signature(buffer, options);
    put(scan_signature, buffer, options, result);
  }
  print(result);
}
                                    // red font: UNIC API
```

# Example

```
void simple_virus_scanner(file, options) {
   buffer = read(file);
   if (exists(scan_signature, buffer, options)) {
      result = get(scan_signature, buffer, options);
   } else {
      result = scan_signature(buffer, options);
      put(scan_signature, buffer, options, result);
   }
   print(result);
}                                    // red font: UNIC API
```

# Example

```
void simple_virus_scanner(file, options) {
  hash = get_file_hash(file);
  if (exists(scan_signature, hash, options)) {
    result = get(scan_signature, hash, options);
  } else {
    buffer = read(file);
    result = scan_signature(buffer, options);
    put(scan_signature, hash, options, result);
  }
  print(result);
}                                  // red font: UNIC API
```

# UNIC API

- **init**()
- **get_file_hash**(*file*)
- **exists**(*computation*, *hash*, *id*)
- **get**(*computation*, *hash*, *id*)
- **put**(*computation*, *hash*, *id*, *result*, *ttl*)

# UNIC API

- **`init`**`()`
- **`get_file_hash`**`(`*`file`*`)`

Leverages deduplication metadata from the underlying storage

- **`exists`**`(`*`computation`*`, `*`hash`*`, `*`id`*`)`
- **`get`**`(`*`computation`*`, `*`hash`*`, `*`id`*`)`
- **`put`**`(`*`computation`*`, `*`hash`*`, `*`id`*`, `*`result`*`, `*`ttl`*`)`

# UNIC API

- **init**()
- **get_file_hash**(*file*)
- **exists**(*computation*, *hash*, *id*)
- **get**(*computation*, *hash*, *id*)
- **put**(*computation*, *hash*, *id*, *result*)

Application-defined string identifier



# Evaluation

Our experiments aim to answer:

- Is UNIC easy to use?
- Does UNIC reduce computation time?
- What is UNIC's storage overhead?

# Evaluation

We evaluated UNIC on 4 popular open-source apps:

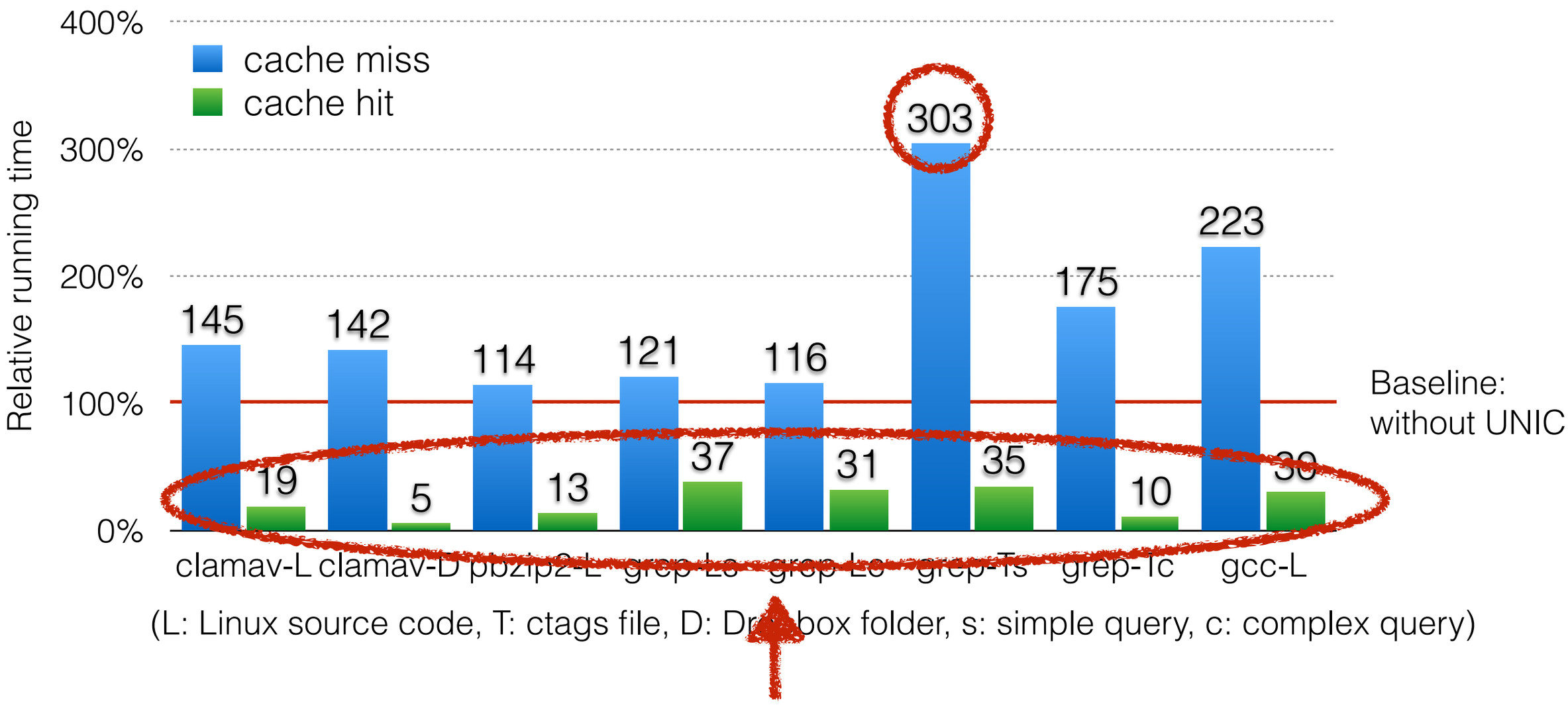
- clamav: anti-virus software.
- pbzip2: multi-threaded compression utility.
- grep: pattern searching tool.
- gcc: compiler (using ccache).

# Application adaptation effort

| Application | Total LoC | Changes | Percentage |
|---|---|---|---|
| clamav | 1,732,762 | 12 | <0.01% |
| pbzip2 | 4,376 | 18 | 0.41% |
| grep | 9,658 | 35 | 0.36% |
| gcc (ccache) | 29,023 | 30 | 0.10% |

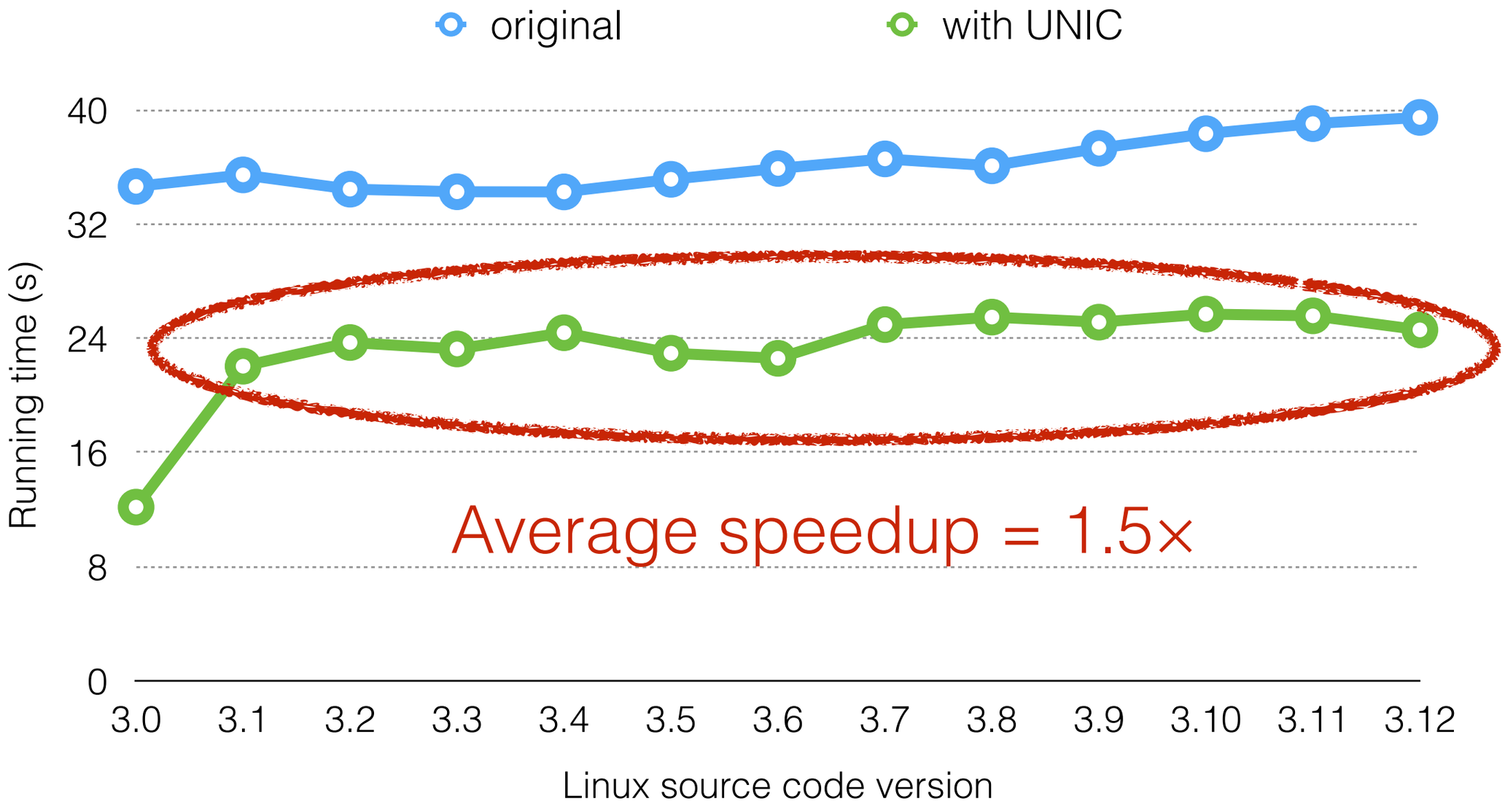
UNIC is easy to use.

# Application performance




Reusing cached result is much faster.
Average speedup = 7.6×

# Working with changing data




UNIC also works well with partially duplicate data.

# Storage overhead

| Application | Dataset size | Cache size | Percentage |
|---|---|---|---|
| clamav-L | 508.1MB | 2.8MB | 0.55% |
| clamav-D | 10.8GB | 4.4MB | 0.04% |
| pbzip2-L | 544.0MB | 106.4MB | 19.55% |
| grep-Ls | 508.1MB | 11.2MB | 2.21% |
| grep-Lc | 508.1MB | 4.2MB | 0.83% |
| grep-Ts | 250.0MB | 5.3MB | 2.13% |
| grep-Tc | 250.0MB | 4.5MB | 1.80% |
| gcc-L | 508.1MB | 2.3MB | 0.46% |

(L: Linux source code, T: ctags file, D: Dropbox folder, s: simple query, c: complex query)

UNIC incurs little storage overhead.

# Conclusion

UNIC: secure deduplication of general computations.

- It uses code attestation for secrecy and integrity.
- It exports a simple yet expressive API.
- It explores a cross-layer design to leverage storage deduplication.