POSITION PAPER

MULTI-VERSION SOFTWARE UPDATES

CRISTIAN CADAR **PETR HOSEK**

"The fundamental problem with program maintenance is that fixing a defect has a substantial (20*-50%) chance of introducing another. So the whole process is two steps forward and one step back."

—**F. Brooks,** 1975
The Mythical Man-Month

*__More than 14.8~24.4% for major operating system patches__

Yin, Z., Yuan, D., Zhou, Y., Pasupathy, S., and Bairavasundaram, L. *How Do Fixes Become Bugs?* ESEC/FSE'11

Users Refuse to Upgrade

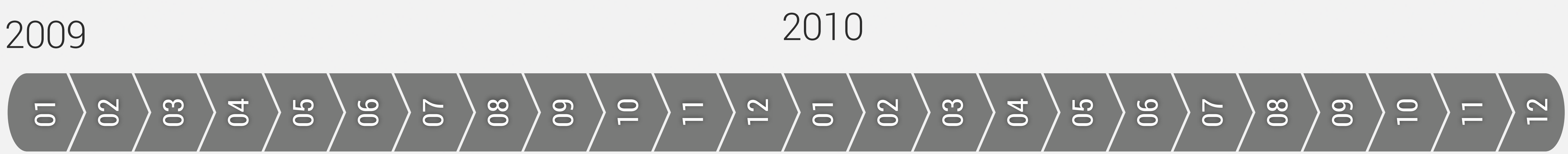
Because software updates often present a high risk:

- Many users refuse to upgrade their software
- Reliance on outdated versions flawed with vulnerabilities

*Crameri, O., Knezevic, N., Kostic, D., Bianchini, R., Zwaenepoel, W. *Staged deployment in Mirage, an integrated software upgrade testing and distribution system.* SOSP'07

LIGHTTPD WEB SERVER

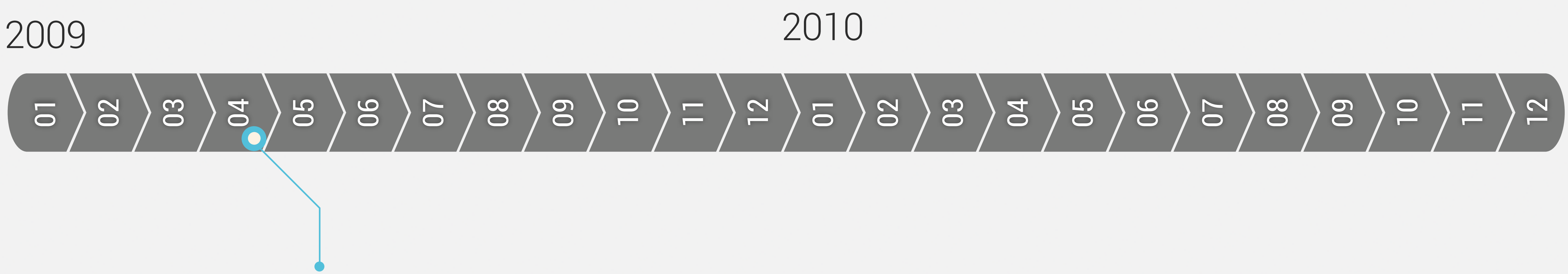
Popular web-server used by YouTube, Wikipedia or Meebo



LIGHTTPD
fly light.

LIGHTTPD WEB SERVER

Popular web-server used by YouTube, Wikipedia or Meebo



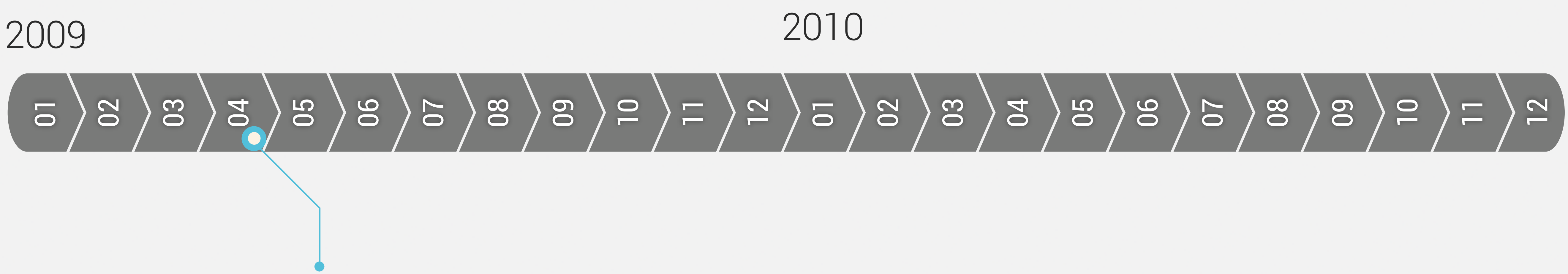

```
for (h = 0, i = 0; i < etag->used; ++i)
  h = (h << 5) ^ (h >> 27) ^ (etag->ptr[i]);
```

HTTP ETag hash value computation in `etag_mutate`

LIGHTTPD fly light.

LIGHTTPD WEB SERVER

Popular web-server used by YouTube, Wikipedia or Meebo



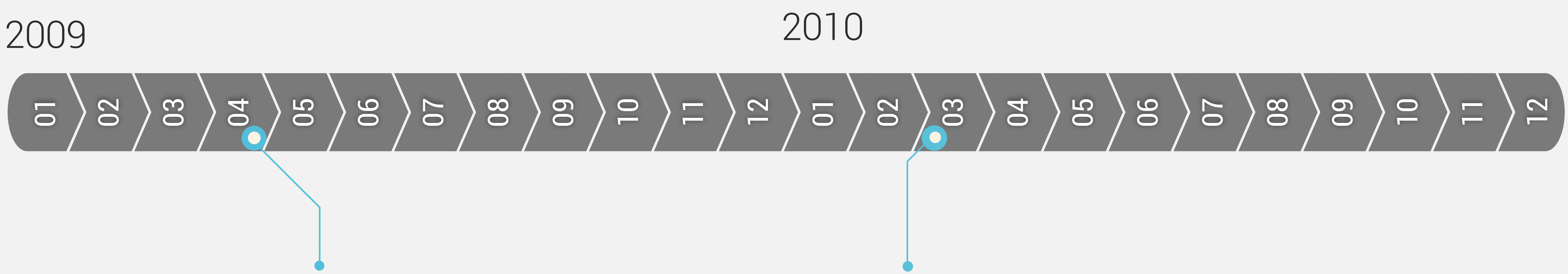

```
for (h = 0, i = 0; i < etag->used - 1; ++i)
  h = (h << 5) ^ (h >> 27) ^ (etag->ptr[i]);
```

HTTP ETag hash value computation in `etag_mutate`

LIGHTTPD fly light.

LIGHTTPD WEB SERVER

Popular web-server used by YouTube, Wikipedia or Meebo




```
for (h = 0, i = 0; i < etag->used - 1; ++i)
  h = (h << 5) ^ (h >> 27) ^ (etag->ptr[i]);
```

HTTP ETag hash value computation in `etag_mutate`

Bug reported in issue tracker

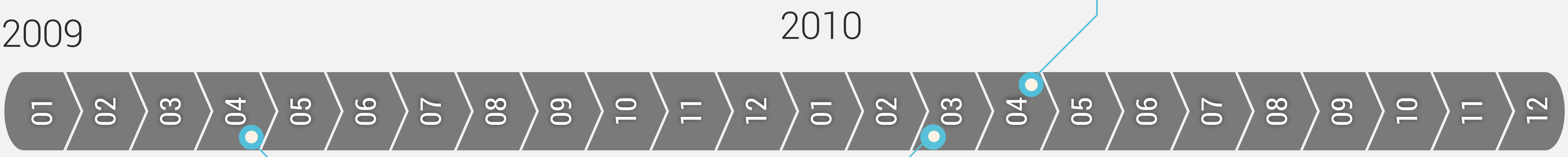
LIGHTTPD
fly light.

LIGHTTPD WEB SERVER

Popular web-server used by YouTube, Wikipedia or Meebo

```
etag_mutate(con->physical.etag, srv->tmp_buf);
```

File (re)compression in `mod_compress_physical`




```
for (h = 0, i = 0; i < etag->used - 1; ++i)
  h = (h << 5) ^ (h >> 27) ^ (etag->ptr[i]);
```

HTTP ETag hash value computation in `etag_mutate`

Bug reported in issue tracker

LIGHTTPD fly light.

LIGHTTPD WEB SERVER

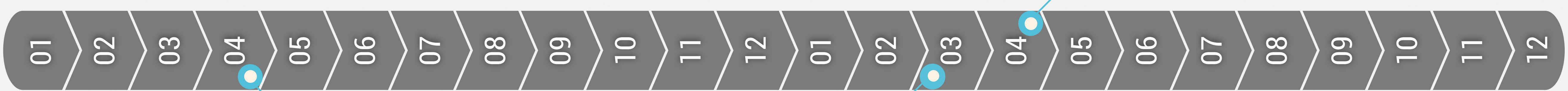
Popular web-server used by YouTube, Wikipedia or Meebo

```
if (use_etag) {
    etag_mutate(con->physical.etag, srv->tmp_buf);
}
```

File (re)compression in `mod_compress_physical`

2009

2010




```
for (h = 0, i = 0; i < etag->used - 1; ++i)
  h = (h << 5) ^ (h >> 27) ^ (etag->ptr[i]);
```

HTTP ETag hash value computation in `etag_mutate`

Bug reported in issue tracker

LIGHTTPD fly light.

LIGHTTPD WEB SERVER

Popular web-server used by YouTube, Wikipedia or Meebo

```
if (use_etag) {
    etag_mutate(con->physical.etag, srv->tmp_buf);
}
```

File (re)compression in `mod_compress_physical`

Bug reported in issue tracker

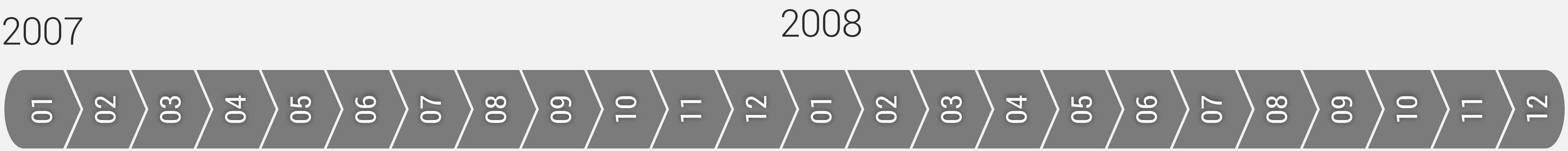
```
for (h = 0, i = 0; i < etag->used - 1; ++i)
  h = (h << 5) ^ (h >> 27) ^ (etag->ptr[i]);
```

HTTP ETag hash value computation in `etag_mutate`

LIGHTTPD
fly light.

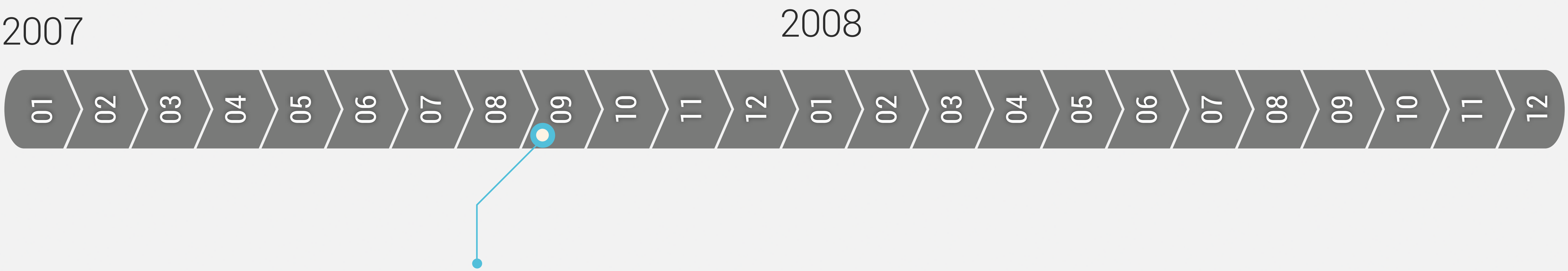
VIM TEXT EDITOR

Arguably the most popular editor for UNIX systems



VIM TEXT EDITOR

Arguably the most popular editor for UNIX systems




```
if (orig != orig_save)
    vim_free(orig);
```

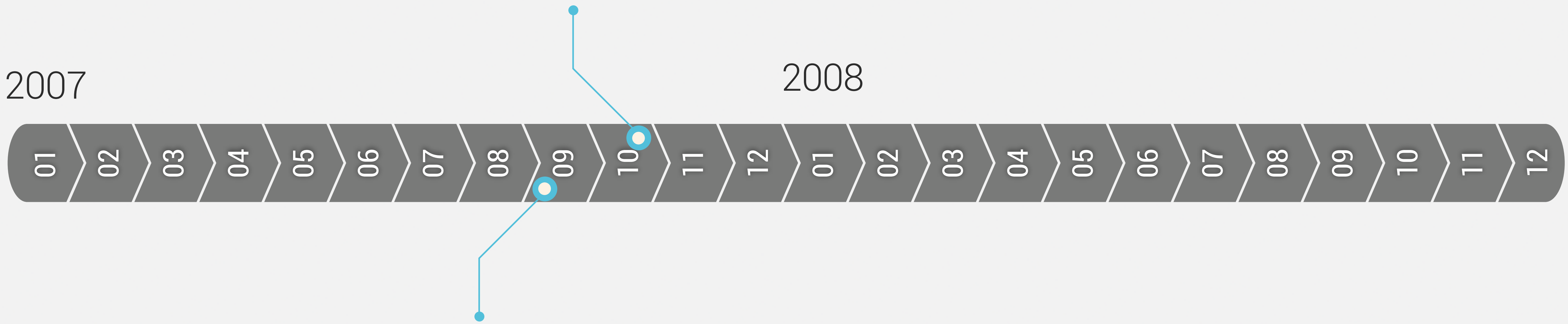
Vim 7.1.127 fixed memory leak when doing path completion

VIM TEXT EDITOR

Arguably the most popular editor for UNIX systems

```
if (orig != orig_save)
    vim_free(orig);
```

Vim 7.1.147 fixed memory freeing on user name completion




```
if (orig != orig_save)
    vim_free(orig);
```

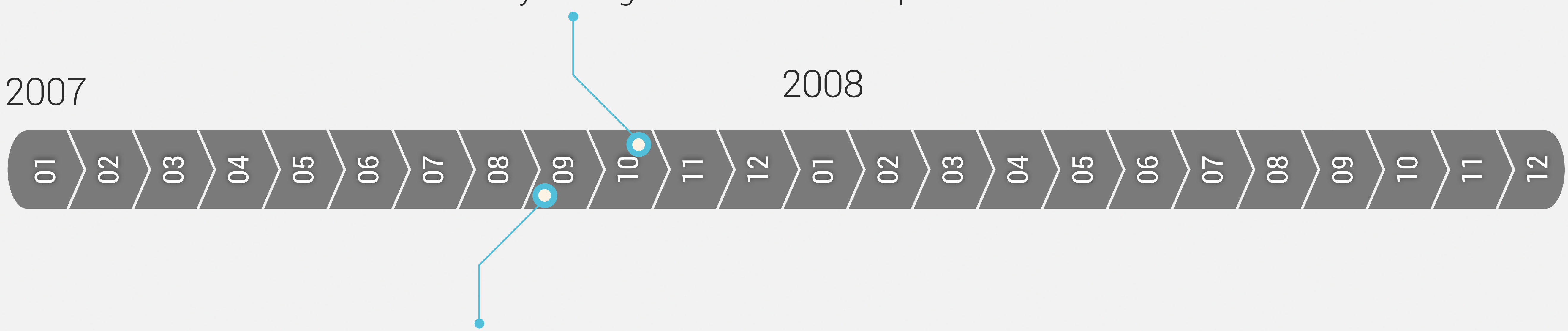
Vim 7.1.127 fixed memory leak when doing path completion

VIM TEXT EDITOR

Arguably the most popular editor for UNIX systems

```
vim_free(orig_save);
orig_saved = TRUE;
...
if (!orig_saved)
    vim_free(orig);
```

Vim 7.1.147 fixed memory freeing on user name completion




```
if (orig != orig_save)
    vim_free(orig);
```

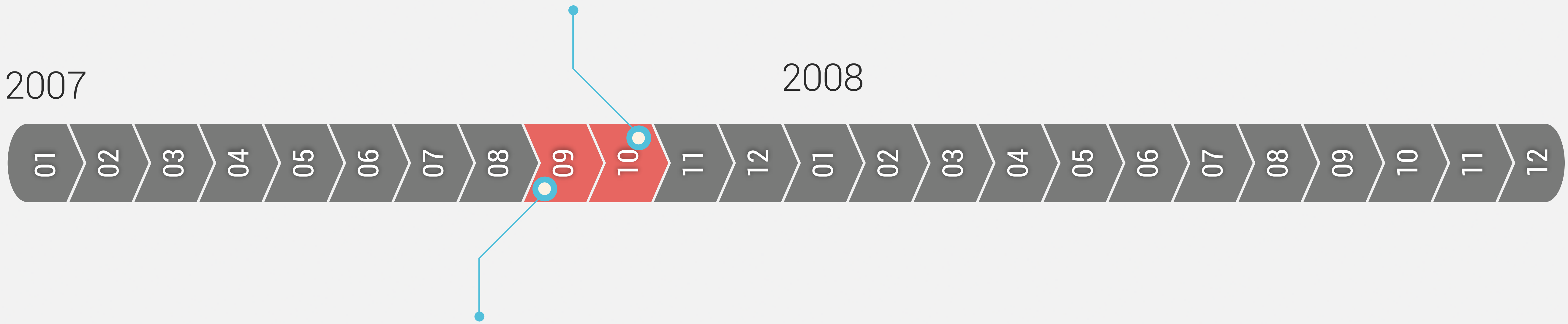
Vim 7.1.127 fixed memory leak when doing path completion

VIM TEXT EDITOR

Arguably the most popular editor for UNIX systems

```
vim_free(orig_save);
orig_saved = TRUE;
...
if (!orig_saved)
    vim_free(orig);
```

Vim 7.1.147 fixed memory freeing on user name completion




```
if (orig != orig_save)
    vim_free(orig);
```

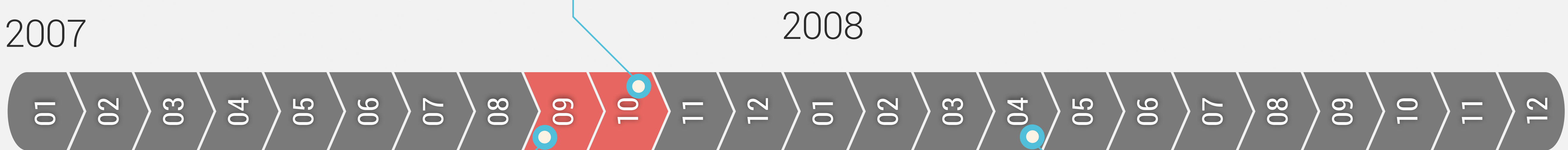
Vim 7.1.127 fixed memory leak when doing path completion

VIM TEXT EDITOR

Arguably the most popular editor for UNIX systems

```
vim_free(orig_save);
orig_saved = TRUE;
...
if (!orig_saved)
    vim_free(orig);
```

Vim 7.1.147 fixed memory freeing on user name completion




```
if (orig != orig_save)
    vim_free(orig);
```

Vim 7.1.127 fixed memory leak when doing path completion

Ubuntu 8.04 LTS released

Including the **Vim 7.1.138** affected by the path completion bug

VIM TEXT EDITOR

Arguably the most popular editor for UNIX systems

```
vim_free(orig_save);
orig_saved = TRUE;
...
if (!orig_saved)
    vim_free(orig);
```

Vim 7.1.147 fixed memory freeing on user name completion


```
if (orig != orig_save)
    vim_free(orig);
```

Vim 7.1.127 fixed memory leak when doing path completion

Ubuntu 8.04 LTS released

Including the **Vim 7.1.138** affected by the path completion bug

Existing Approaches

Verification and validation

Testing, static analysis, model checking, symbolic execution, etc.

Deployment strategies

Staged or delayed deployment

Software fault isolation

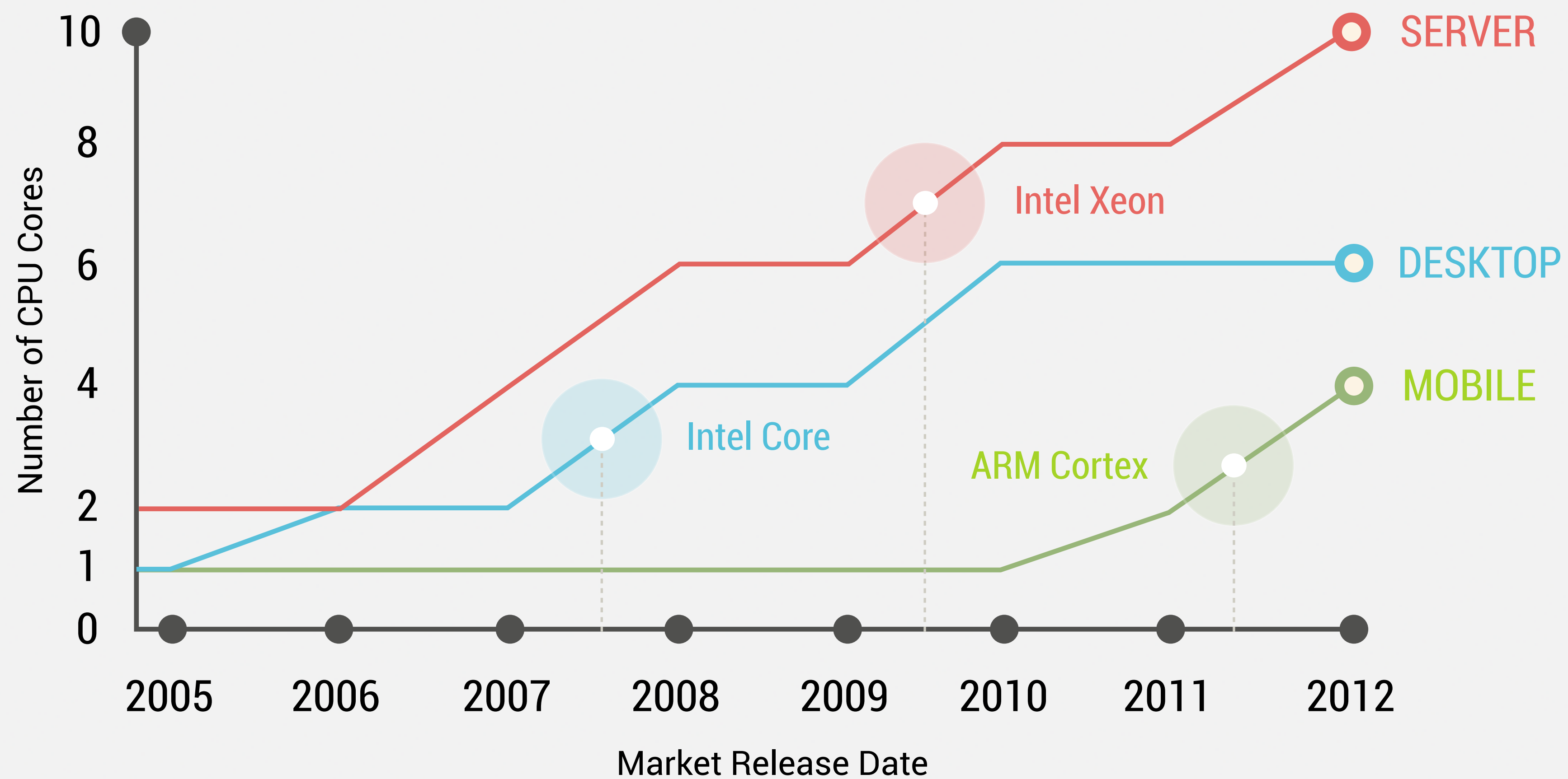
Virtualization and sandboxing

Our Proposal

Improve the execution of upgraded software to provide:

- Benefits of the newer version
- Stability of the older version

Multicore CPUs becoming a standard



Abundance of resources and a high degree of parallelism

Cadar, C., Pietzuch P., Wolf, A. L. *Multiplicity computing: A vision of software engineering for next-generation computing platform applications*. FoSER'10

Our Proposal

Multi-version execution based approach:

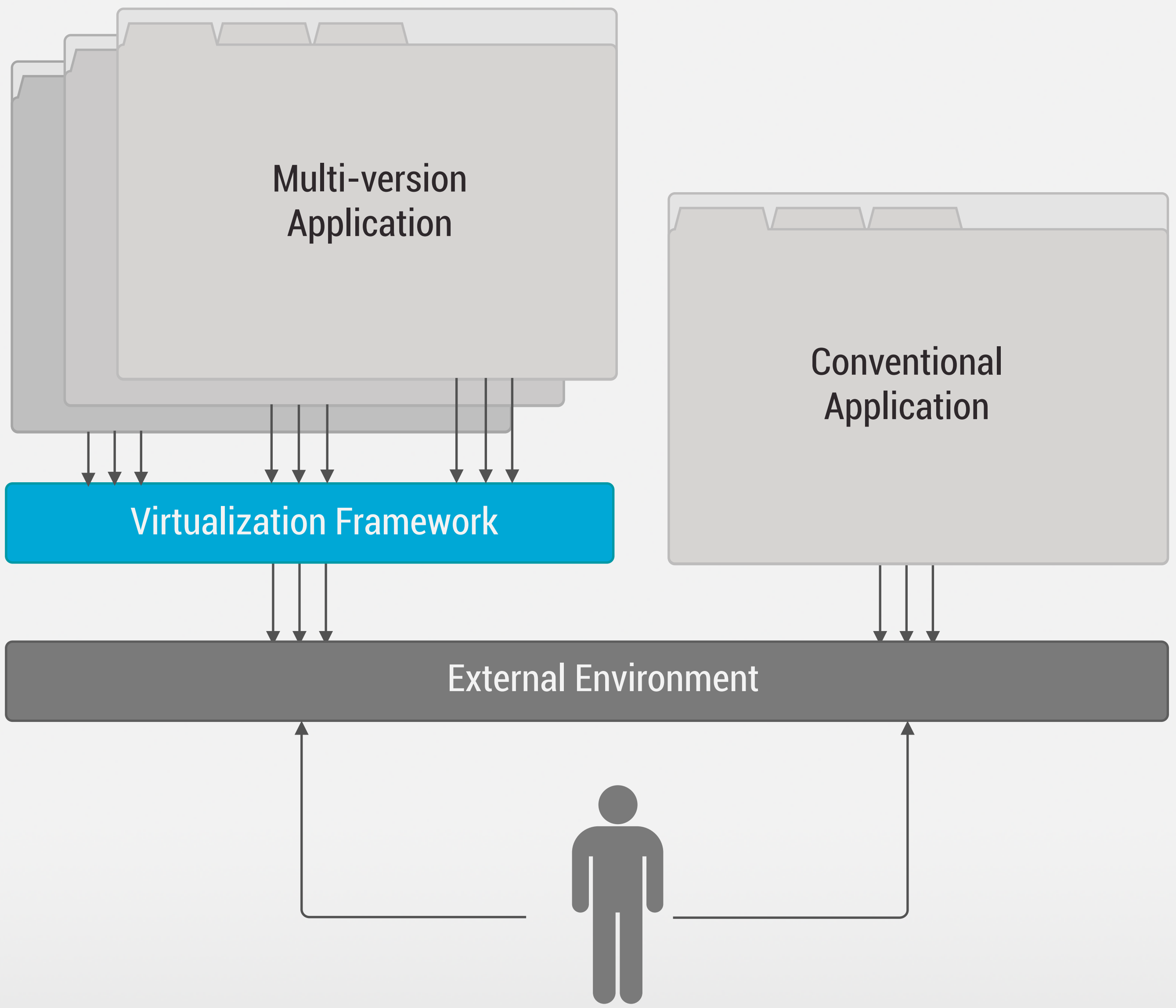
Run the new version in parallel with the existing one

Coordinate the execution of the two versions

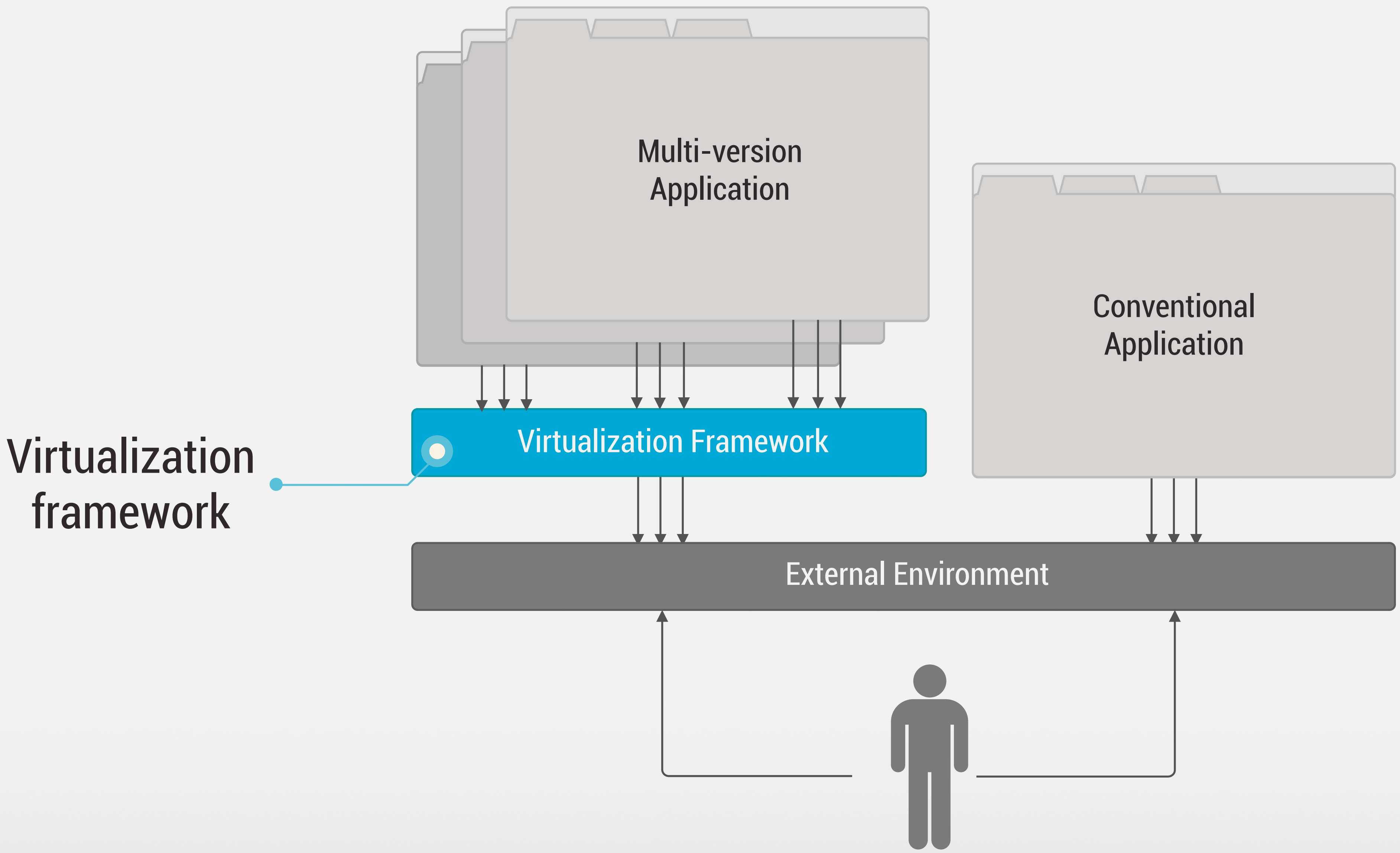
Use output of correctly executing version at any given time

Can be extended to work with multiple versions

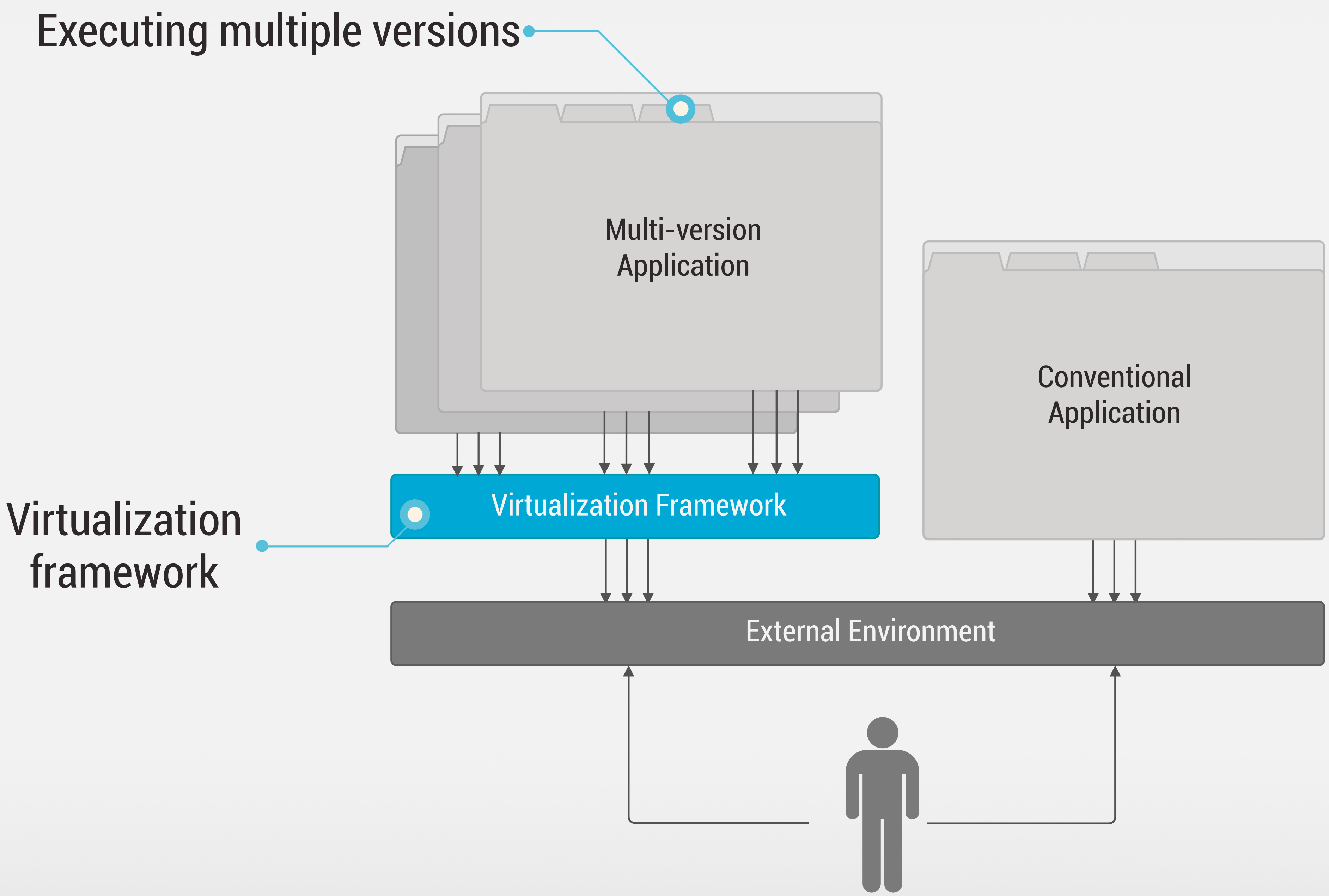
Goals and Challenges



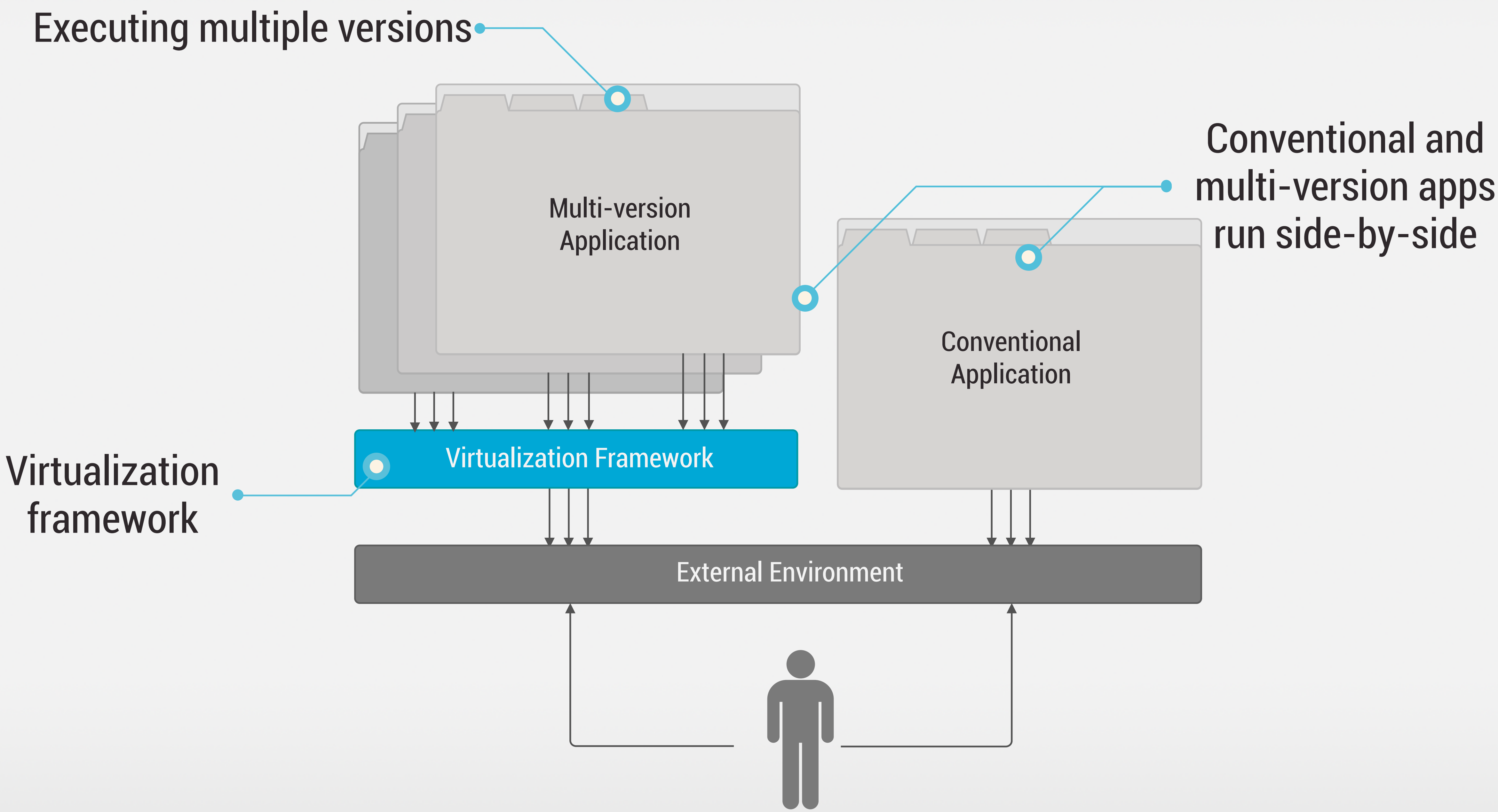
Goals and Challenges



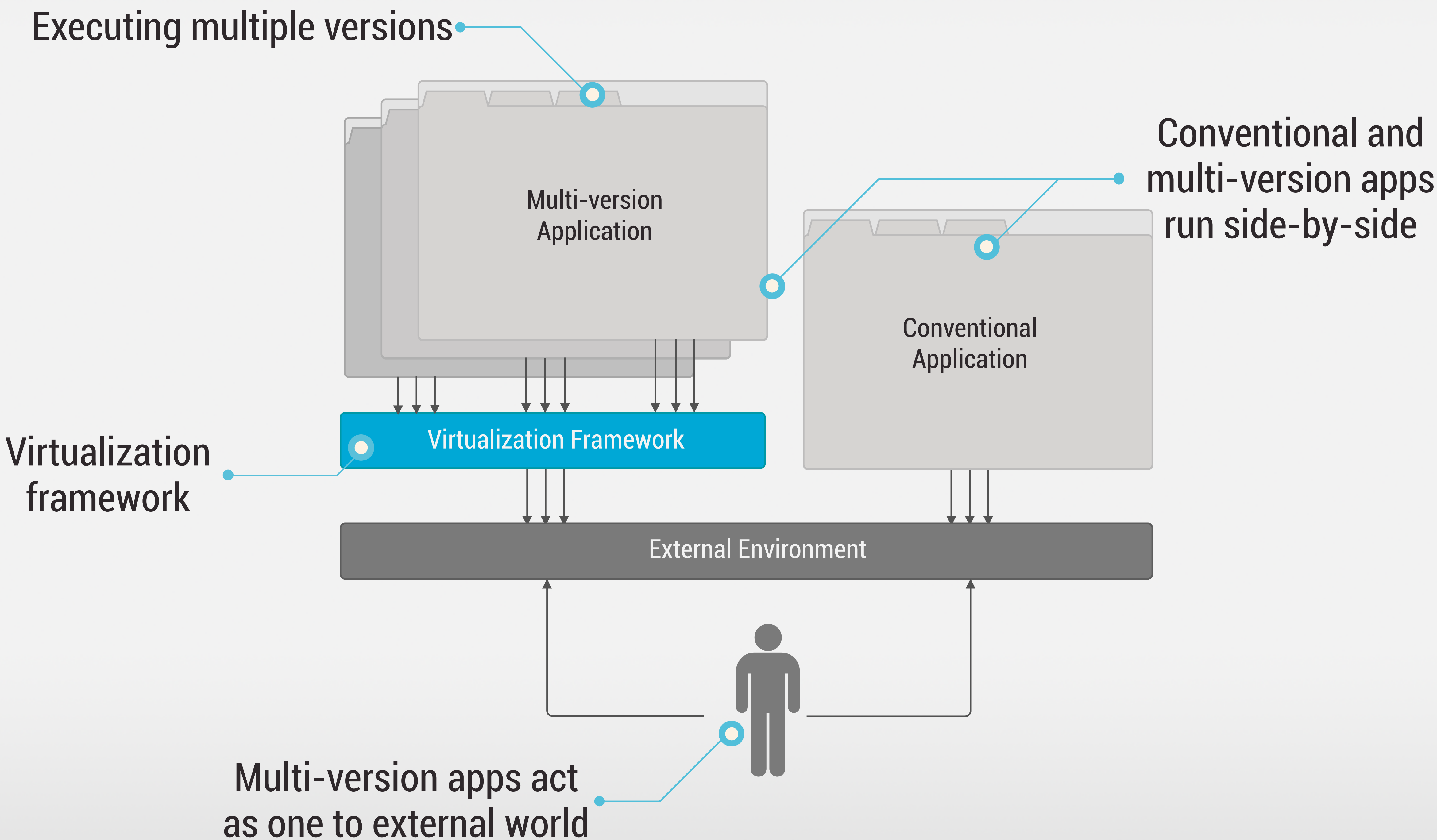
Goals and Challenges



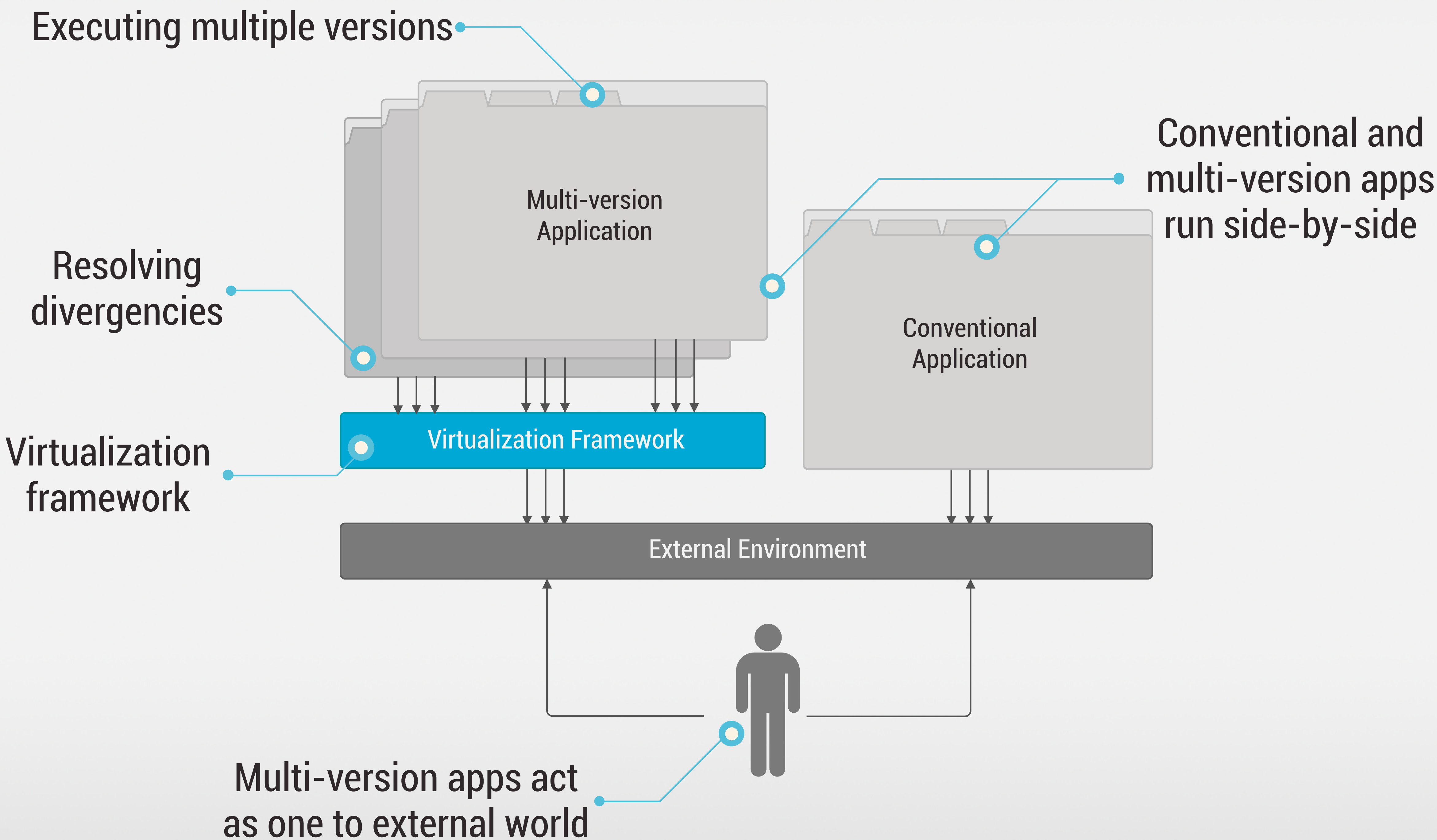
Goals and Challenges



Goals and Challenges



Goals and Challenges



LIGHTTPD **1.4.22**

LIGHTTPD **1.4.23**

LIGHTTPD
fly light.


LIGHTTPD 1.4.22
LIGHTTPD 1.4.23
Synchronization


LIGHTTPD 1.4.22
LIGHTTPD 1.4.23
Synchronization
GET /index.html HTTP/1.1
Host: srg.doc.ic.ac.uk
Accept-Encoding: gzip

LIGHTTPD **1.4.22**

LIGHTTPD **1.4.23**

```
GET /index.html HTTP/1.1
Host: srg.doc.ic.ac.uk
Accept-Encoding: gzip
```

Synchronization

```
for (h = 0, i = 0; i < etag->used; ++i)
  h = (h << 5) ^ (h >> 27) ^ (etag->ptr[i]);
```

LIGHTTPD **1.4.22**

LIGHTTPD **1.4.23**

```
GET /index.html HTTP/1.1
Host: srg.doc.ic.ac.uk
Accept-Encoding: gzip
```

Synchronization

```
for (h = 0, i = 0; i < etag->used; ++i)
  h = (h << 5) ^ (h >> 27) ^ (etag->ptr[i]);
```

```
for (h = 0, i = 0; i < etag->used - 1; ++i)
  h = (h << 5) ^ (h >> 27) ^ (etag->ptr[i]);
```

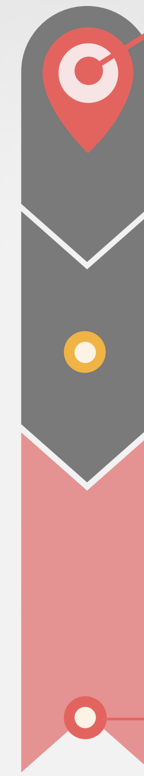
Crash
Segmentation fault

LIGHTTPD
fly light.

LIGHTTPD **1.4.22**

LIGHTTPD **1.4.23**

```
GET /index.html HTTP/1.1
Host: srg.doc.ic.ac.uk
Accept-Encoding: gzip
```



Synchronization

```
for (h = 0, i = 0; i < etag->used; ++i)
  h = (h << 5) ^ (h >> 27) ^ (etag->ptr[i]);
```

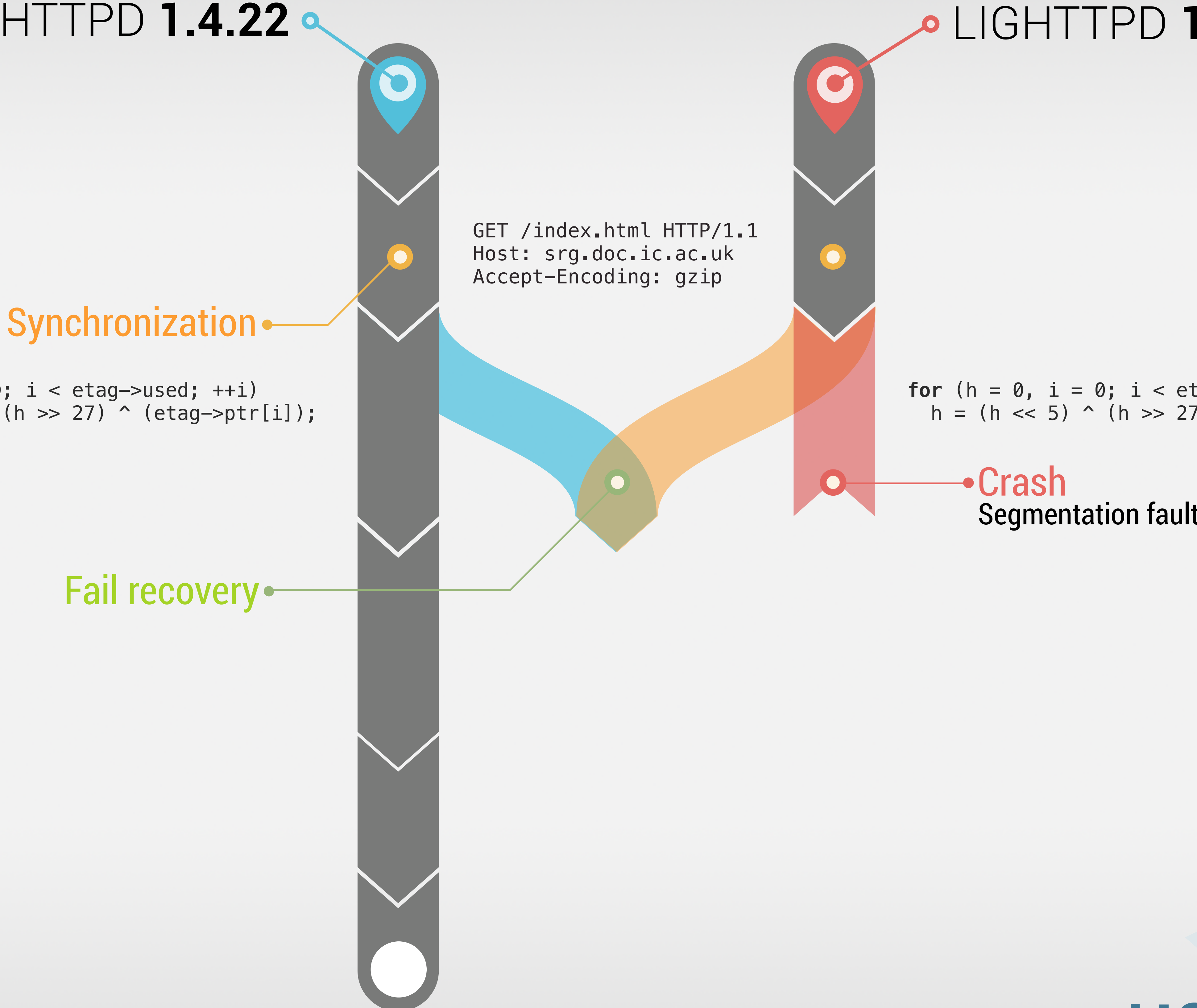
```
for (h = 0, i = 0; i < etag->used - 1; ++i)
  h = (h << 5) ^ (h >> 27) ^ (etag->ptr[i]);
```

Crash
Segmentation fault

LIGHTTPD
fly light.

LIGHTTPD **1.4.22**

LIGHTTPD **1.4.23**

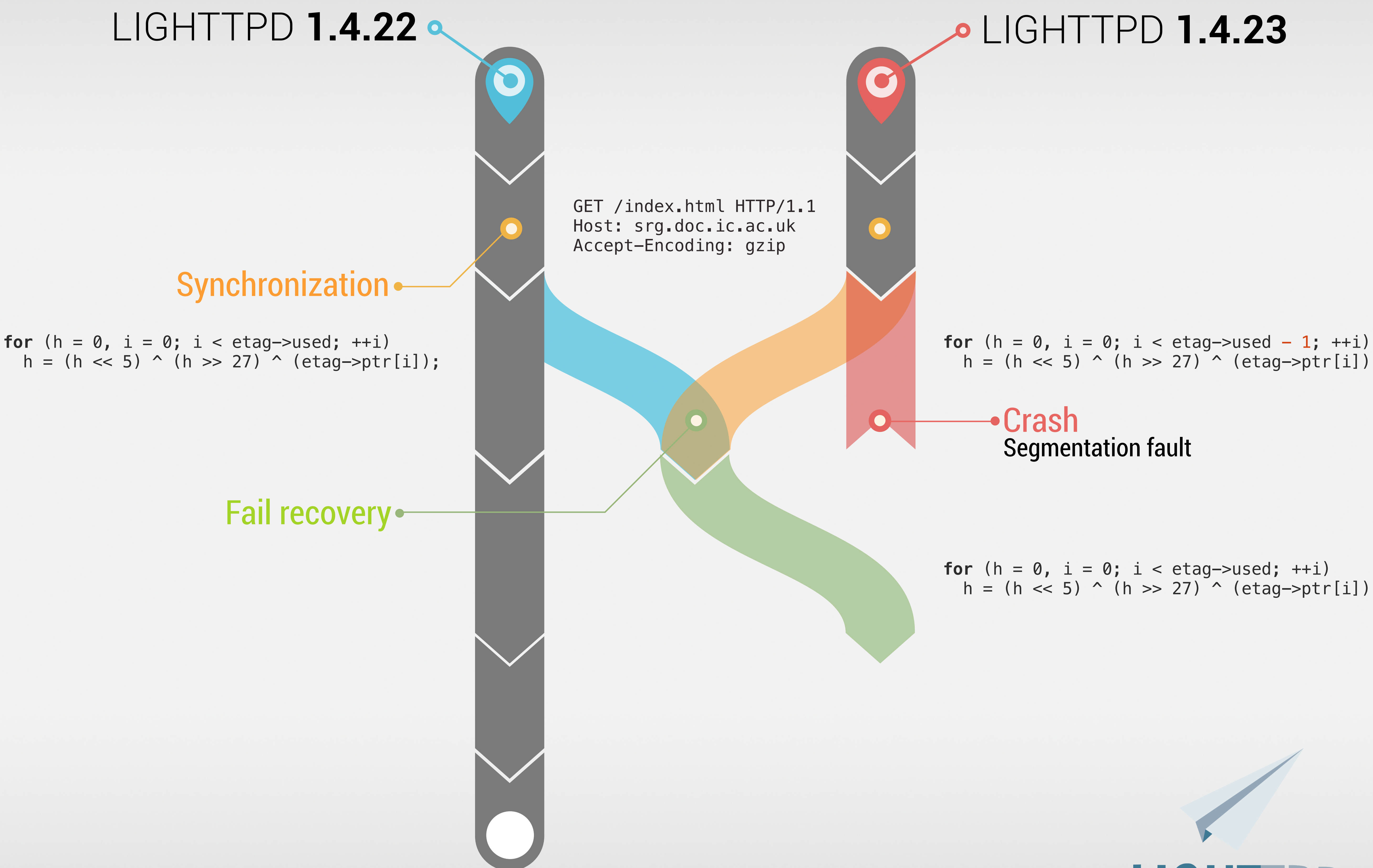



```
GET /index.html HTTP/1.1
Host: srg.doc.ic.ac.uk
Accept-Encoding: gzip
```

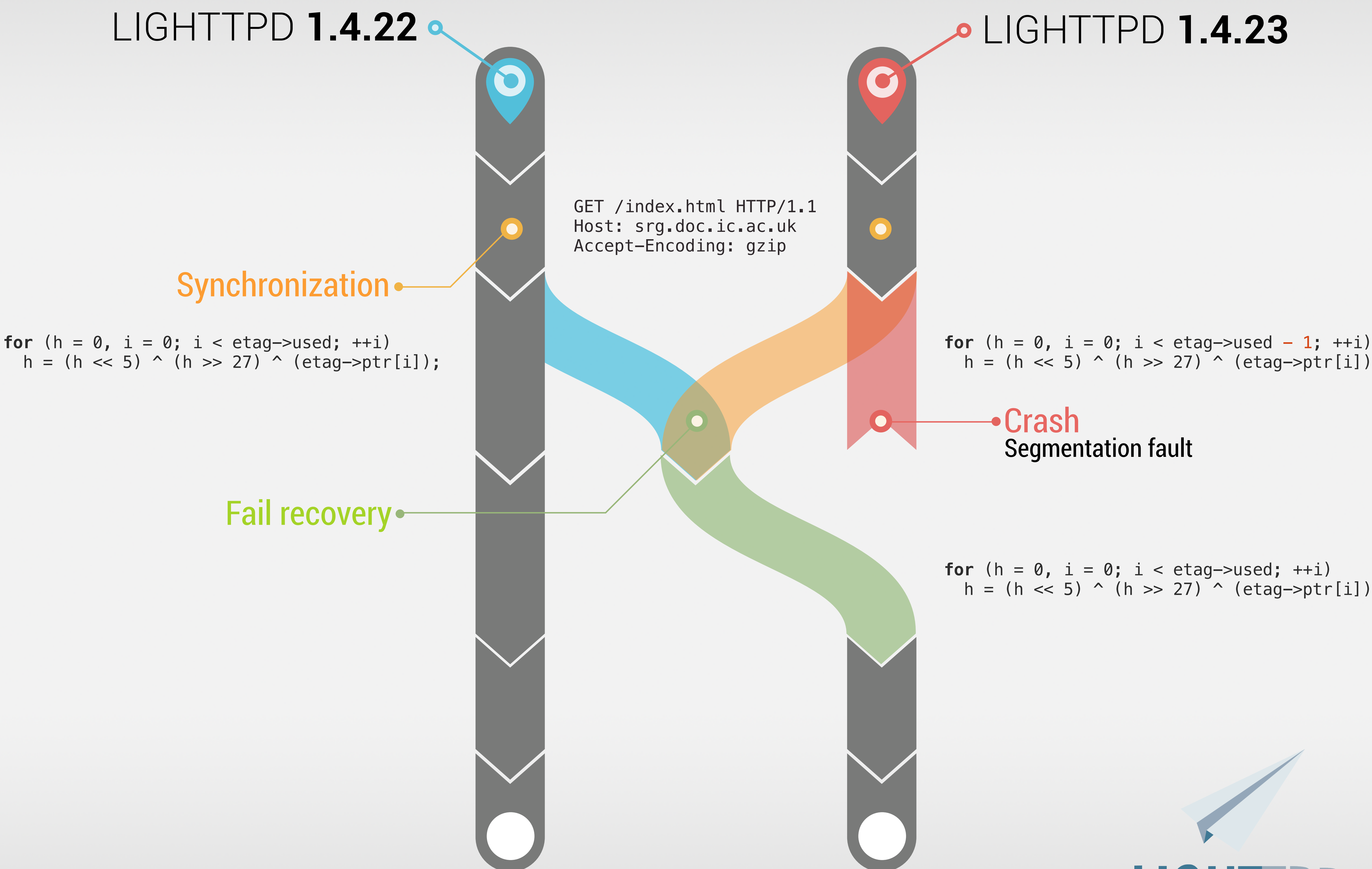
```
for (h = 0, i = 0; i < etag->used; ++i)
  h = (h << 5) ^ (h >> 27) ^ (etag->ptr[i]);
```

```
for (h = 0, i = 0; i < etag->used - 1; ++i)
  h = (h << 5) ^ (h >> 27) ^ (etag->ptr[i]);
```

LIGHTTPD fly light.



LIGHTTPD
fly light.



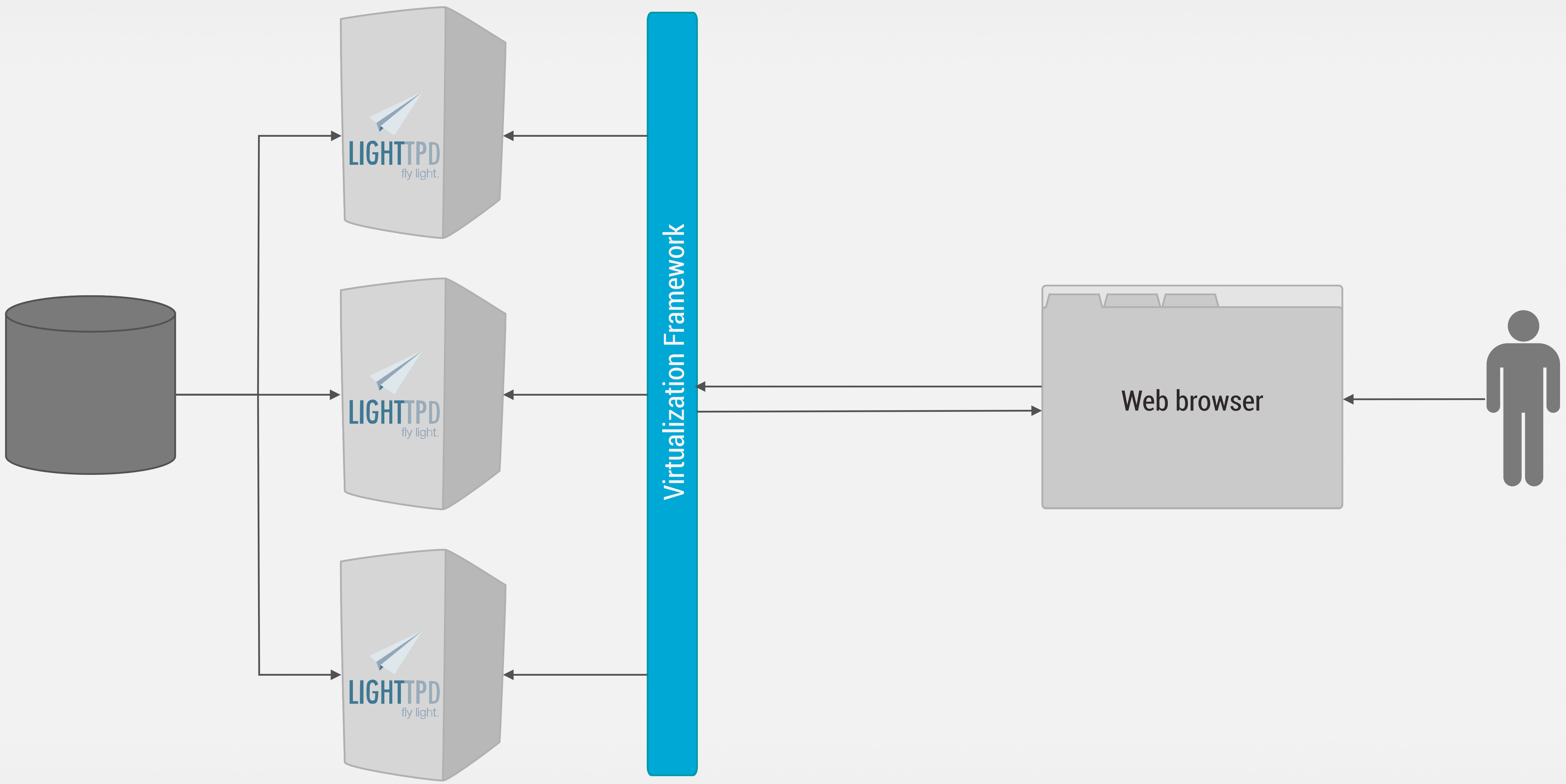
LIGHTTPD
fly light.

Synchronization Mechanism

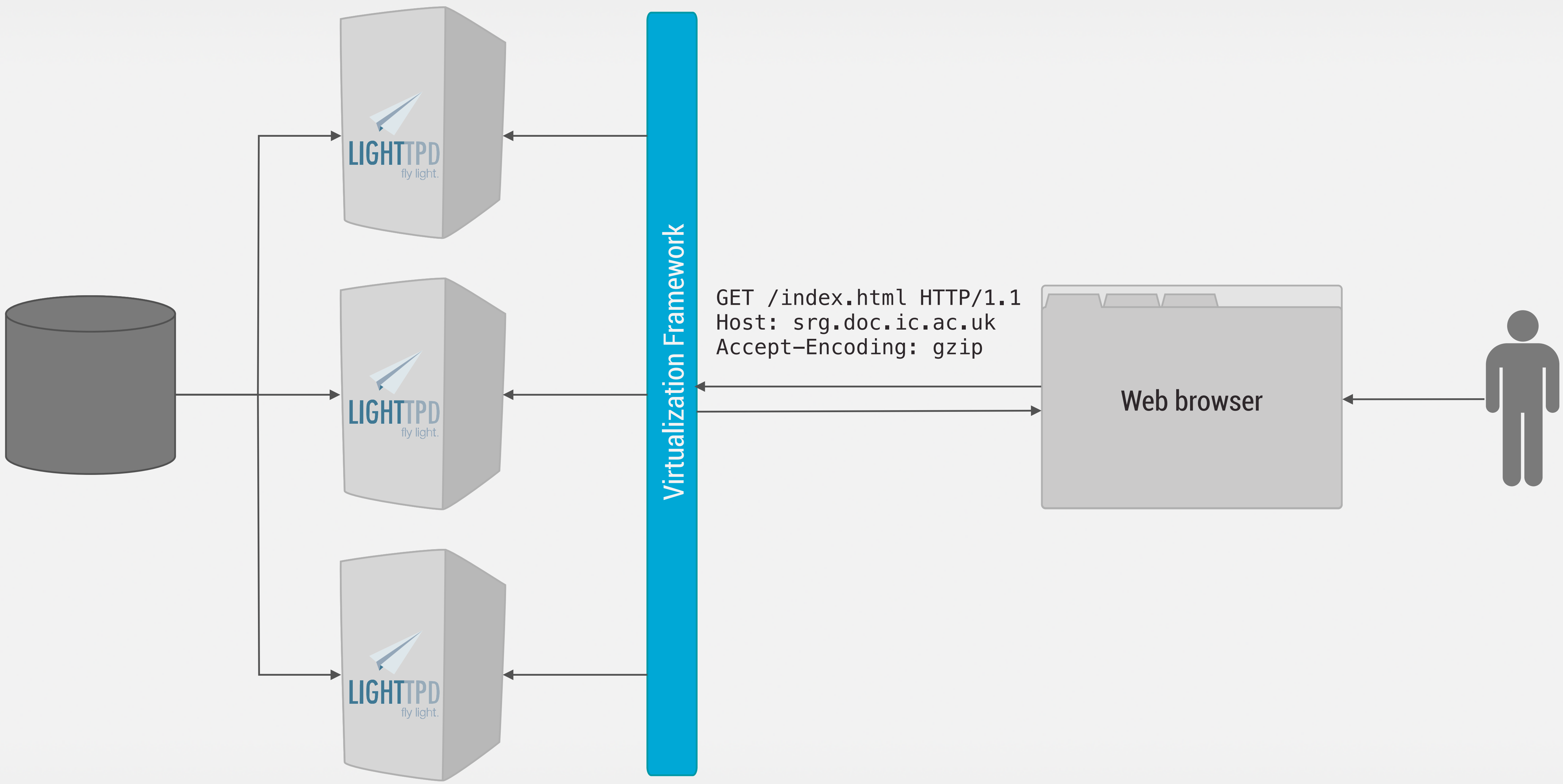
Synchronization at multiple levels of abstraction:

- Application inputs-outputs
- Library calls
- System calls

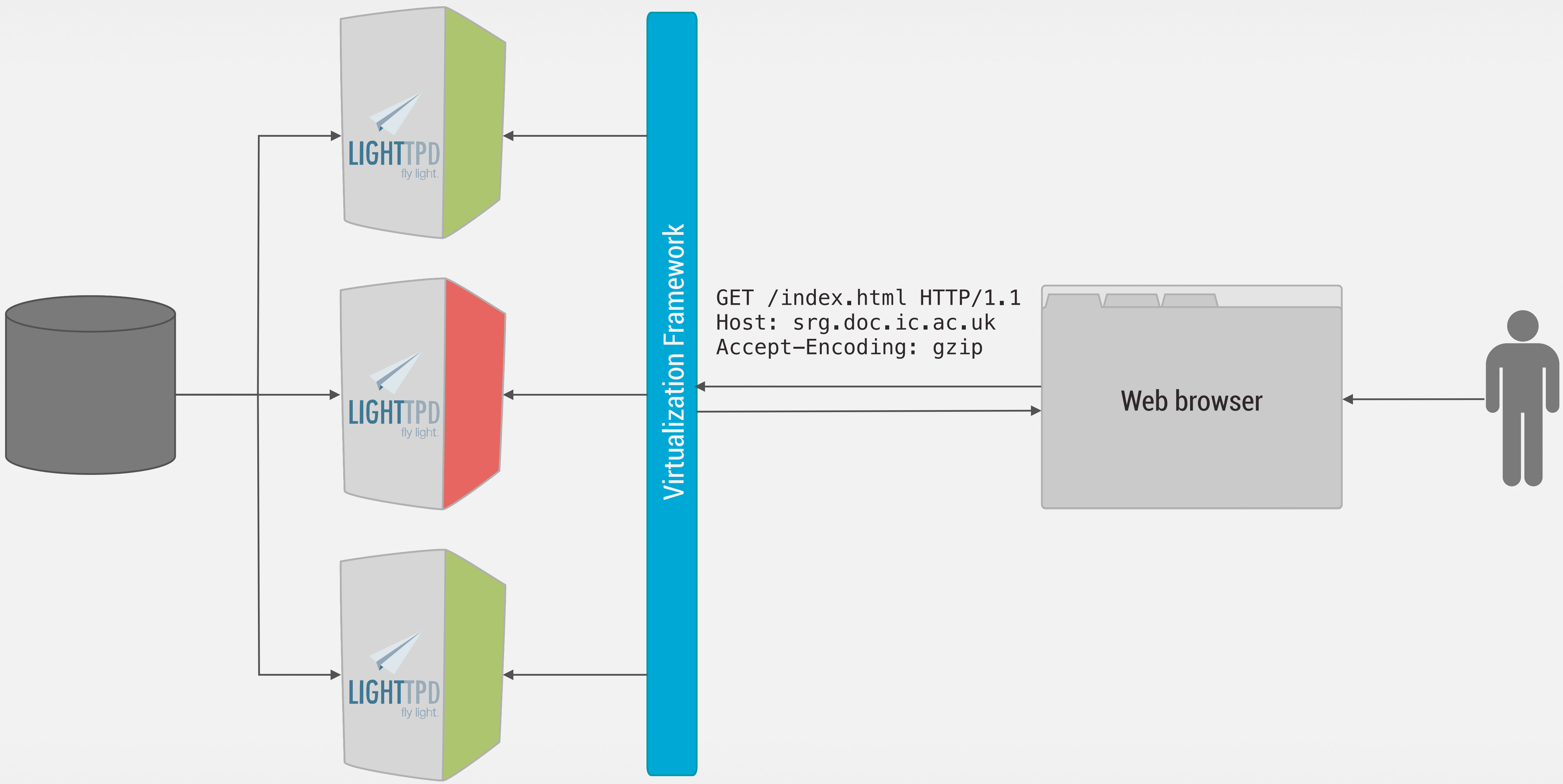
Synchronization at the **protocol level**



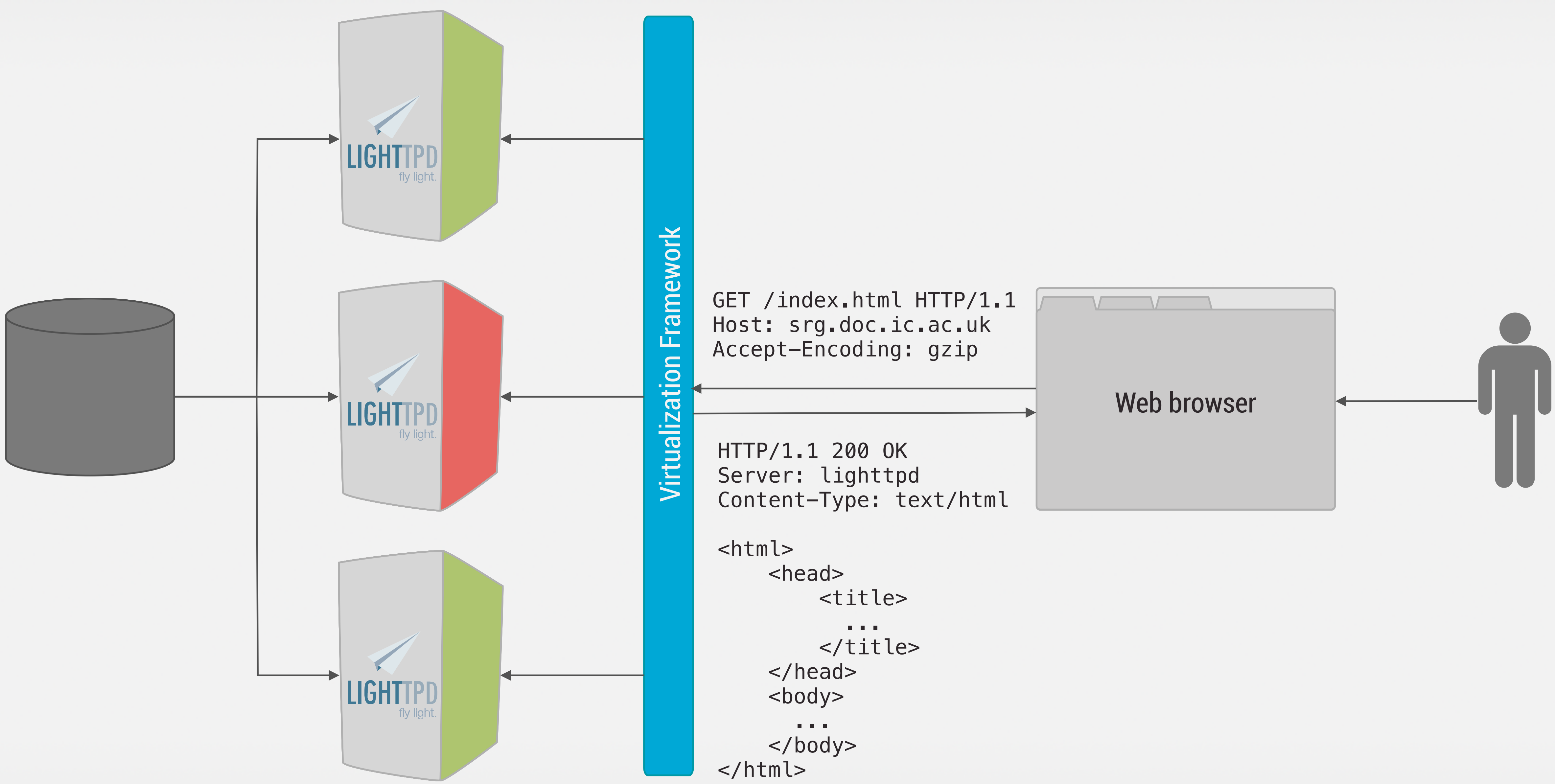
Synchronization at the **protocol level**



Synchronization at the **protocol level**



Synchronization at the **protocol level**



Synchronization at the level of **system calls**

VERSION 1

```
void print_sorted(int *arr, size_t len)
{
    int sarr[len];
    memcpy(sarr, arr, sizeof(sarr));

    bsort(sarr, len, sizeof(int), cmp);
    for (int i = 0; i < len; ++i)
        printf("%d\n", sarr[i]);
}
```

VERSION 2

```
void print_sorted(int *arr, size_t len)
{
    int sarr[len];
    memcpy(sarr, arr, sizeof(sarr));

    qsort(sarr, len, sizeof(int), cmp);
    for (int i = 0; i < len; ++i)
        printf("%d\n", sarr[i]);
}
```

```
int main(int argc, char **argv)
{
    int arr[] = { 6, 4, 3, 7 };
    print_sorted(arr, 4);
}
```

Example testing code
Tested with both implementations

Synchronization at the level of **system calls**

VERSION 1

```
void print_sorted(int *arr, size_t len)
{
    int sarr[len];
    memcpy(sarr, arr, sizeof(sarr));

    bsort(sarr, len, sizeof(int), cmp);
    for (int i = 0; i < len; ++i)
        printf("%d\n", sarr[i]);
}
```

VERSION 2

```
void print_sorted(int *arr, size_t len)
{
    int sarr[len];
    memcpy(sarr, arr, sizeof(sarr));

    qsort(sarr, len, sizeof(int), cmp);
    for (int i = 0; i < len; ++i)
        printf("%d\n", sarr[i]);
}
```

```
int main(int argc, char **argv)
{
    int arr[] = { 6, 4, 3, 7 };
    print_sorted(arr, 4);
}
```

Example testing code
Tested with both implementations

Synchronization at the level of **system calls**

VERSION 1

```
void print_sorted(int *arr, size_t len)
{
    int sarr[len];
    memcpy(sarr, arr, sizeof(sarr));

    bsort(sarr, len, sizeof(int), cmp);
    for (int i = 0; i < len; ++i)
        printf("%d\n", sarr[i]);
}
```

VERSION 2

```
void print_sorted(int *arr, size_t len)
{
    int sarr[len];
    memcpy(sarr, arr, sizeof(sarr));

    qsort(sarr, len, sizeof(int), cmp);
    for (int i = 0; i < len; ++i)
        printf("%d\n", sarr[i]);
}
```

```
...
write(1, "3\n", 23) = 2
write(1, "4\n", 24) = 2
write(1, "6\n", 26) = 2
write(1, "7\n", 27) = 2
...
```

Snippet of system call trace
Obtained using *strace* tool

```
int main(int argc, char **argv)
{
    int arr[] = { 6, 4, 3, 7 };
    print_sorted(arr, 4);
}
```

Example testing code
Tested with both implementations

Synchronization at the level of **system calls**

VERSION 1

```
void print_sorted(int *arr, size_t len)
{
    int sarr[len];
    memcpy(sarr, arr, sizeof(sarr));

    bsort(sarr, len, sizeof(int), cmp);
    for (int i = 0; i < len; ++i)
        printf("%d\n", sarr[i]);
}
```

VERSION 2

```
void print_sorted(int *arr, size_t len)
{
    int sarr[len];
    memcpy(sarr, arr, sizeof(sarr));

    qsort(sarr, len, sizeof(int), cmp);
    for (int i = 0; i < len; ++i)
        printf("%d\n", sarr[i]);
}
```

```
...
write(1, "3\n", 23) = 2
write(1, "4\n", 24) = 2
write(1, "6\n", 26) = 2
write(1, "7\n", 27) = 2
...
```

Snippet of system call trace
Obtained using *strace* tool

```
int main(int argc, char **argv)
{
    int arr[] = { 6, 4, 3, 7 };
    print_sorted(arr, 4);
}
```

Example testing code
Tested with both implementations

Synchronization at the level of **system calls**

VERSION 1

```
void print_sorted(int *arr, size_t len)
{
    int sarr[len];
    memcpy(sarr, arr, sizeof(sarr));

    bsort(sarr, len, sizeof(int), cmp);
    for (int i = 0; i < len; ++i)
        printf("%d\n", sarr[i]);
}
```

```
...
write(1, "3\n", 23) = 2
write(1, "4\n", 24) = 2
write(1, "6\n", 26) = 2
write(1, "7\n", 27) = 2
...
```

Snippet of system call trace
Obtained using *strace* tool

VERSION 2

```
void print_sorted(int *arr, size_t len)
{
    int sarr[len];
    memcpy(sarr, arr, sizeof(sarr));

    qsort(sarr, len, sizeof(int), cmp);
    for (int i = 0; i < len; ++i)
        printf("%d\n", sarr[i]);
}
```

```
...
write(1, "3\n", 23) = 2
write(1, "4\n", 24) = 2
write(1, "6\n", 26) = 2
write(1, "7\n", 27) = 2
...
```

Snippet of system call trace
Obtained using *strace* tool

```
int main(int argc, char **argv)
{
    int arr[] = { 6, 4, 3, 7 };
    print_sorted(arr, 4);
}
```

Example testing code
Tested with both implementations

Synchronization at the level of **system calls**

VERSION 1

```
void print_sorted(int *arr, size_t len)
{
    int sarr[len];
    memcpy(sarr, arr, sizeof(sarr));

    bsort(sarr, len, sizeof(int), cmp);
    for (int i = 0; i < len; ++i)
        printf(“%d\n”, sarr[i]);
}
```

```
...
write(1, “3\n”, 23) = 2
write(1, “4\n”, 24) = 2
write(1, “6\n”, 26) = 2
write(1, “7\n”, 27) = 2
...
```

Snippet of system call trace
Obtained using *strace* tool

VERSION 2

```
void print_sorted(int *arr, size_t len)
{
    int sarr[len];
    memcpy(sarr, arr, sizeof(sarr));

    qsort(sarr, len, sizeof(int), cmp);
    for (int i = 0; i < len; ++i)
        printf(“%d\n”, sarr[i]);
}
```

```
...
write(1, “3\n”, 23) = 2
write(1, “4\n”, 24) = 2
write(1, “6\n”, 26) = 2
write(1, “7\n”, 27) = 2
...
```

Snippet of system call trace
Obtained using *strace* tool

```
int main(int argc, char **argv)
{
    int arr[] = { 6, 4, 3, 7 };
    print_sorted(arr, 4);
}
```

Example testing code
Tested with both implementations

Handling Divergencies

How to resolve divergences across versions?

- Use output of the new version by default
- Use output of old version when new version crashes
  - Fail-recovery in case of crash
- Majority voting/Byzantine fault tolerance

Deployment Strategy

How to deal with limited resources?

Use the last *n* released versions

Keep several older (stable) versions

Trade-off between ease of synchronization and stability

Prototype

Implementation for x86 and x86-64 Linux systems:

Combines binary static analysis, lightweight checkpointing and runtime code patching

Synchronization at the level of system calls (via `ptrace`)

Run two versions with small differences in behavior

Focus on application crashes and recovery (i.e. `SIGSEGV`)

Hosek, P., Cadar, C. *Safe Software Updates via Multi-version Execution*. Imperial College London Technical Report

Survived a number of crash bugs
in several popular server applications

redis

Popular in-memory NoSQL database

```
robj *o = lookupKeyRead(c->db, c->argv[1]);
if (o == NULL) {
    addReplySds(c,sdscatprintf(sdsempty(),
      "*%d\r\n",c->argc-2));
    for (i = 2; i < c->argc; i++) {
        addReply(c,shared.nullbulk);
    }
    return;
} else {
    if (o->type != REDIS_HASH) {
        addReply(c,shared.wrongtypeerr);
        return;
    }
}
addReplySds(c,sdscatprintf(sdsempty(),
  "*%d\r\n",c->argc-2));
```

Redis regression bug #344 introduced during refactoring

HMGET command implementation in hmgetCommand function

Survived a number of crash bugs

in several popular server applications

redis

Popular in-memory NoSQL database

```
robj *o, *value;
o = lookupKeyRead(c->db,c->argv[1]);
if (o != NULL && o->type != REDIS_HASH) {
    addReply(c,shared.wrongtypeerr);
    return;
}
addReplySds(c,sdscatprintf(sdsempty(),
  "*%d\r\n",c->argc-2));
for (i = 2; i < c->argc; i++) {
    if (o != NULL && (value =
          hashGet(o,c->argv[i])) != NULL) {
        addReplyBulk(c,value);
        decrRefCount(value);
    } else {
        addReply(c,shared.nullbulk);
    }
}
```

Missing return statement

Redis regression bug #344 introduced during refactoring

HMGET command implementation in hmgetCommand function

Related Work

Distinct code bases, manually-generated

N-version programming: A fault-tolerance approach to reliability of software operation

Chen, L., and Avizienis, A. FTCS'78

Using replicated execution for a more secure and reliable web browser

Xue, H., Dautenhahn, N., and King, S. T. NDSS'12

Variants of the same code, automatically generated

N-variant systems: a secretless framework for security through diversity

Cox, B., Evans, D., Filipi, A., Rowanhill, J., Hu, W., Davidson, J., Knight, J., Nguyen-Tuong, A., and Hiser, J. USENIX Security'06

Run-time defense against code injection attacks using replicated execution

Salamat, B., Jackson, T., Wagner, G., Wimmer, C., and Franz, M. IEEE Transactions 2011

Validation of different manually-evolved versions

Efficient online validation with delta execution

Tucek, J., Xiong, W., Zhou, Y. ASPLOS'09

Summary

Novel approach for improving software updates:

- Based on multi-version execution
- Our prototype can survive crash bugs in real apps

Many opportunities for future work:

- Better performance overhead
- Support for more complex code changes
- Support for non-crashing type of divergences

Discussion Topics

Types of applications and scenarios suitable for multi-version execution

- User interactive applications vs servers with stringent availability requirements

Deployment strategies for multi-version execution

- Running recent consecutive versions vs keeping older (and more stable) ones

Performance and energy consumption overhead acceptable by providers/end-users

- Overall processor performance is often not a limiting factor
- Power consumption is not always proportional to useful work performed

Identifying meaningful divergencies

- Functional equivalence does not imply equivalence of external behaviour