LOOM: Bypassing Races in Live Applications with Execution Filters

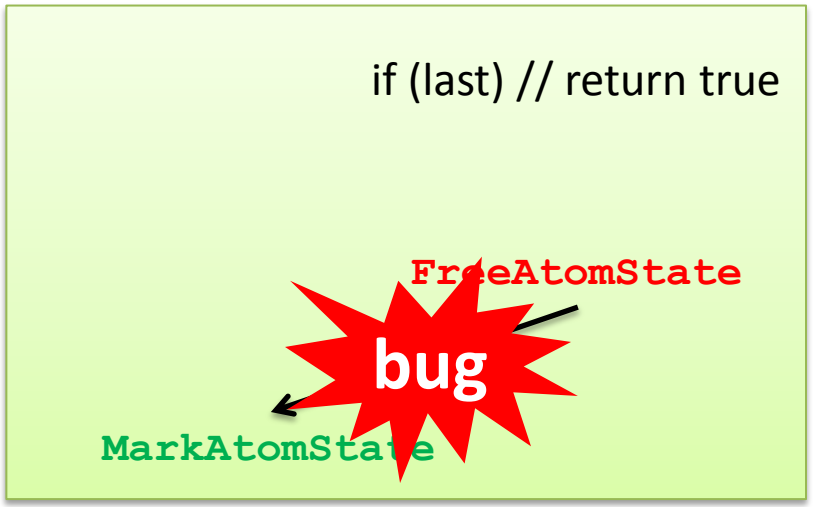
Jingyue Wu, Heming Cui, Junfeng Yang
Columbia University

Mozilla Bug #133773

```
void js_DestroyContext(
       JSContext *cx) {
  JS_LOCK_GC(cx->runtime);
  MarkAtomState(cx);
  if (last) { // last thread?
    ...
    FreeAtomState(cx);
    ...
  }
  JS_UNLOCK_GC(cx->runtime);
}
```

A buggy interleaving

Non-last Thread Last Thread



Complex Fix

```
void js_DestroyContext() {
  if (last) {
    state = LANDING;
    if (requestDepth == 0)
      js_BeginRequest();
    while (gcLevel > 0)
      JS_AWAIT_GC_DONE();
    js_ForceGC(true);
    while (gcPoke)
      js_GC(true);
    FreeAtomState();
  } else {
    gcPoke = true;
    js_GC(false);
  }
}
void js_BeginRequest() {
  while (gcLevel > 0)
    JS_AWAIT_GC_DONE();
}
```

```
void js_ForceGC(bool last)
{
  gcPoke = true;
  js_GC(last);
}
void js_GC(bool last) {
  if (state == LANDING &&
!last)
    return;
  gcLock.acquire();
  if (!gcPoke) {
    gcLock.release();
    return;
  }
  if (gcLevel > 0) {
    gcLevel++;
    while (gcLevel > 0)
      JS_AWAIT_GC_DONE();
    gcLock.release();
    return;
  }
```

```
  gcLevel = 1;
  gcLock.release();
restart:
  MarkAtomState();
  gcLock.acquire();
  if (gcLevel > 1) {
    gcLevel = 1;
    gcLock.release();
    goto restart;
  }
  gcLevel = 0;
  gcPoke = false;
  gcLock.release();
}
```

- 4 functions; 3 integer flags
- Nearly a month
- Not the only example

LOOM: Live-workaround Races

- Execution filters: temporarily filter out buggy thread interleavings

```
void js_DestroyContext(JSContext *cx) {
  MarkAtomState(cx);
  if (last thread) {
    ...
    FreeAtomState(cx);
    ...
  }
}
```

A mutual-exclusion execution filter to bypass the race on the left

```
js_DestroyContext <> self
```

- Declarative, easy to write

LOOM: Live-workaround Races

- Execution filters: temporarily filter out buggy thread interleavings
- Installs execution filters to live applications
  - Improve server availability
  - STUMP [PLDI ‘09], Ginseng [PLDI ‘06], KSplice [EUROSYS ‘09]
- Installs execution filters safely
  - Avoid introducing errors
- Incurs little overhead during normal execution

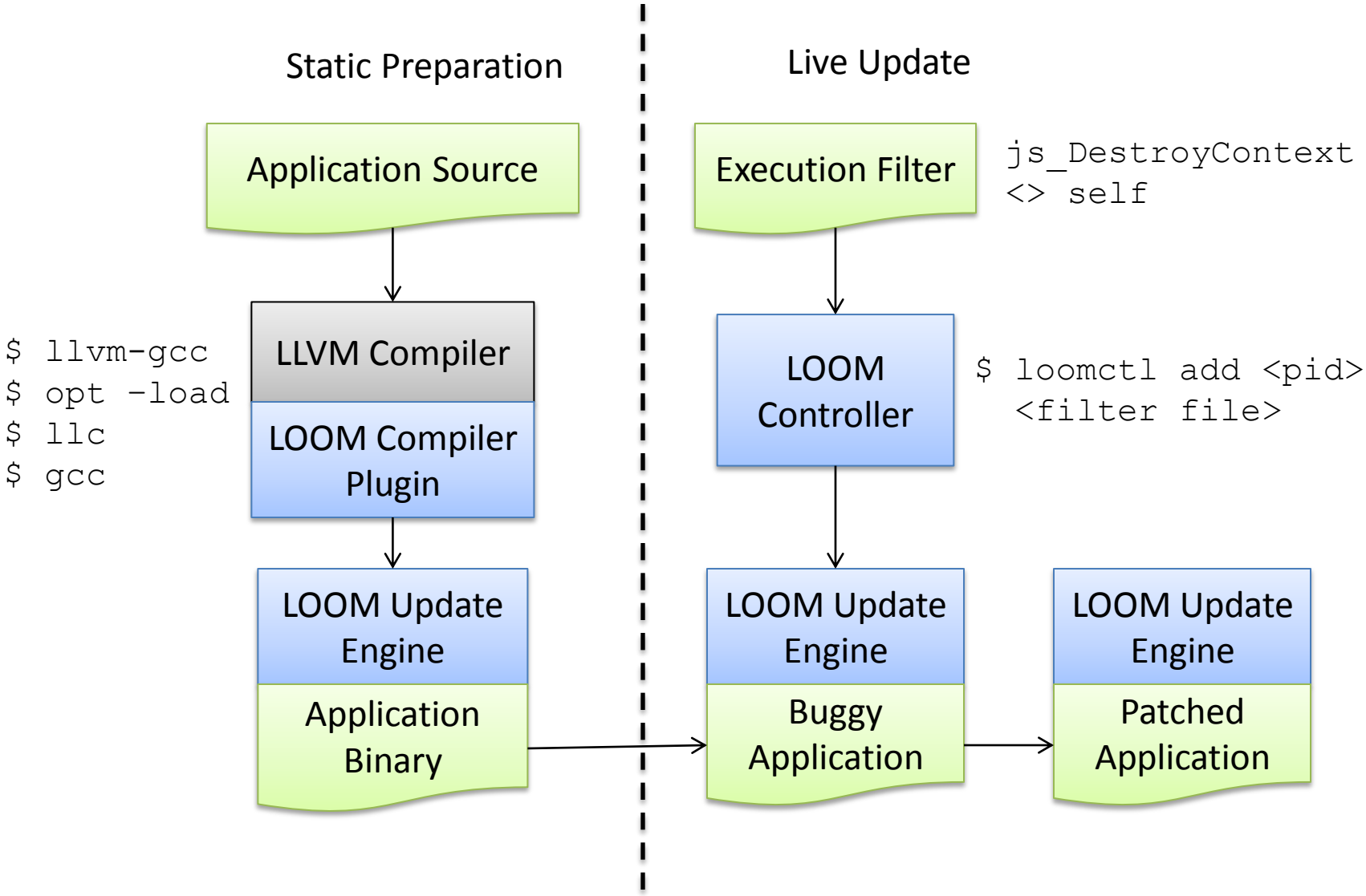
Summary of Results

- We evaluated LOOM on nine real races.
  - Bypasses all the evaluated races safely
  - Applies execution filters immediately
  - Little performance overhead (< 5%)
  - Scales well with the number of application threads (< 10% with 32 threads)
  - Easy to use (< 5 lines)

Outline

- Architecture
  - Combines static preparation and live update
- Safely updating live applications
- Reducing performance overhead
- Evaluation
- Conclusion

Architecture

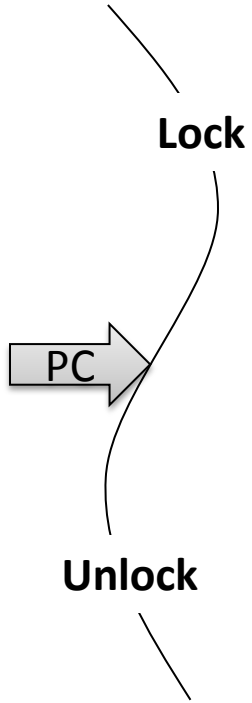


Outline

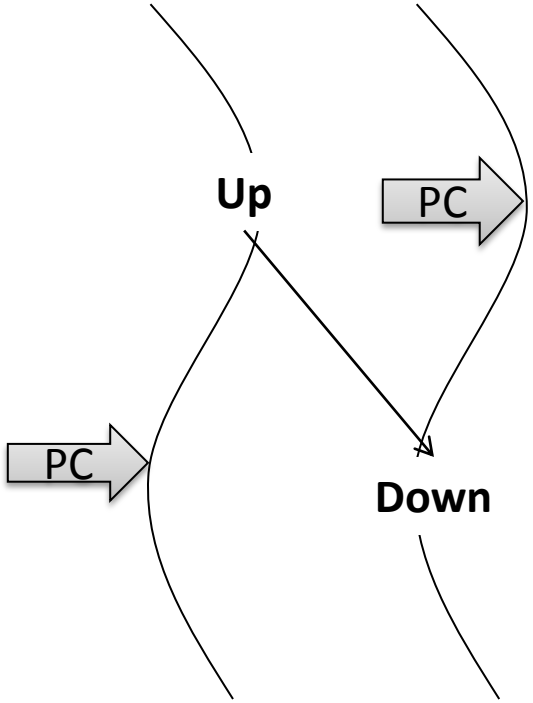
- Architecture
  - Combines static preparation and live update
- Safely updating live applications
- Reducing performance overhead
- Evaluation
- Conclusion

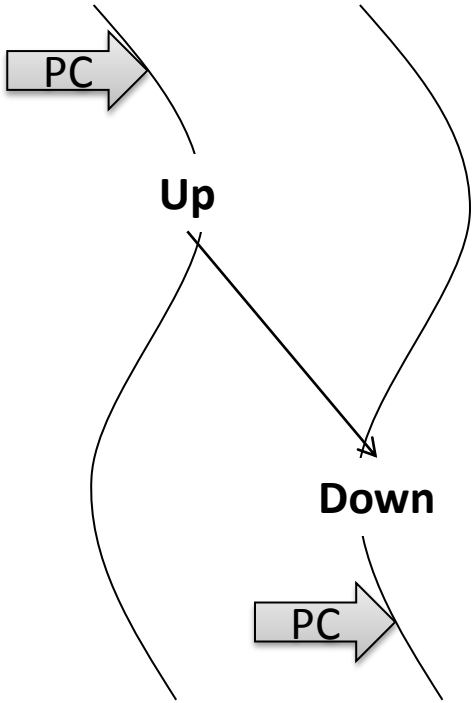
Safety: Not Introducing New Errors

Mutual Exclusion



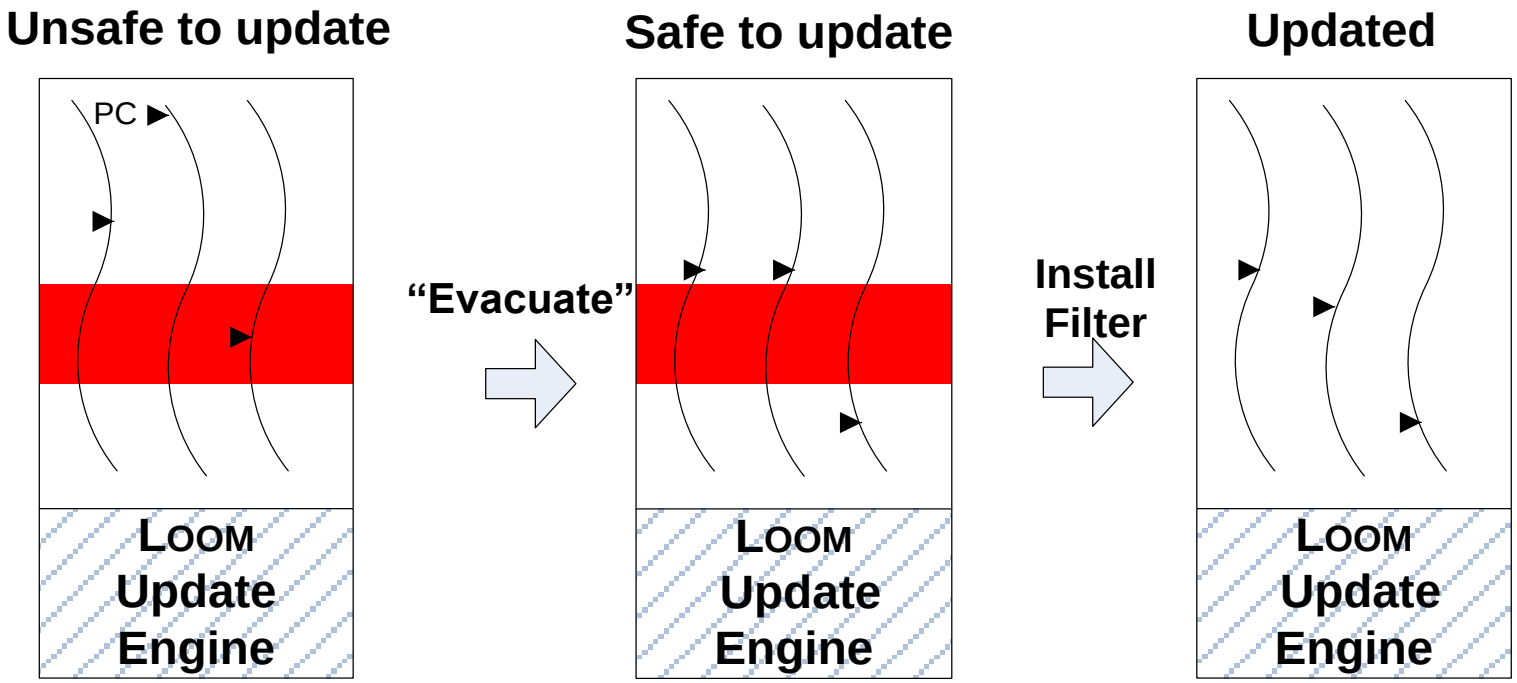
Order Constraints





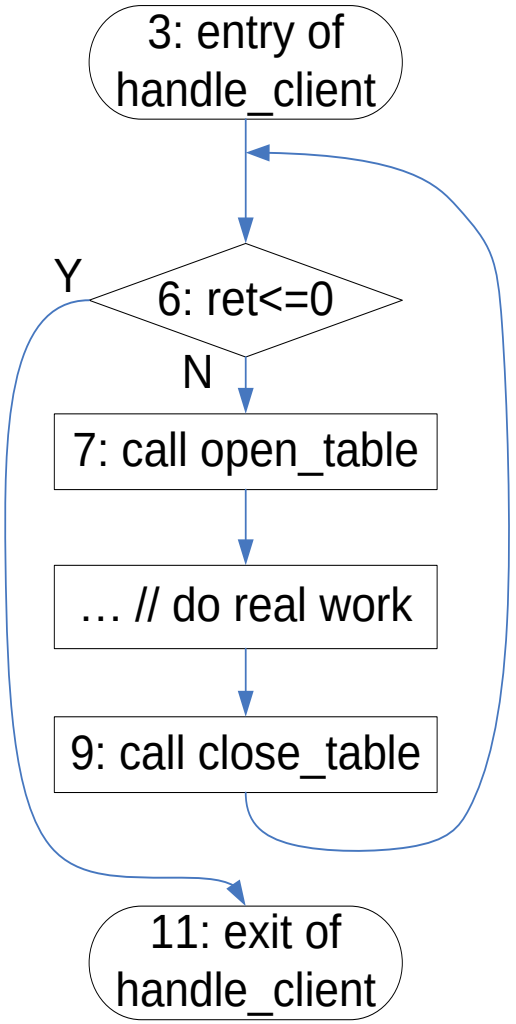
Evacuation Algorithm

1. Identify the dangerous region using static analysis
2. Evacuate threads that are in the dangerous region
3. Install the execution filter

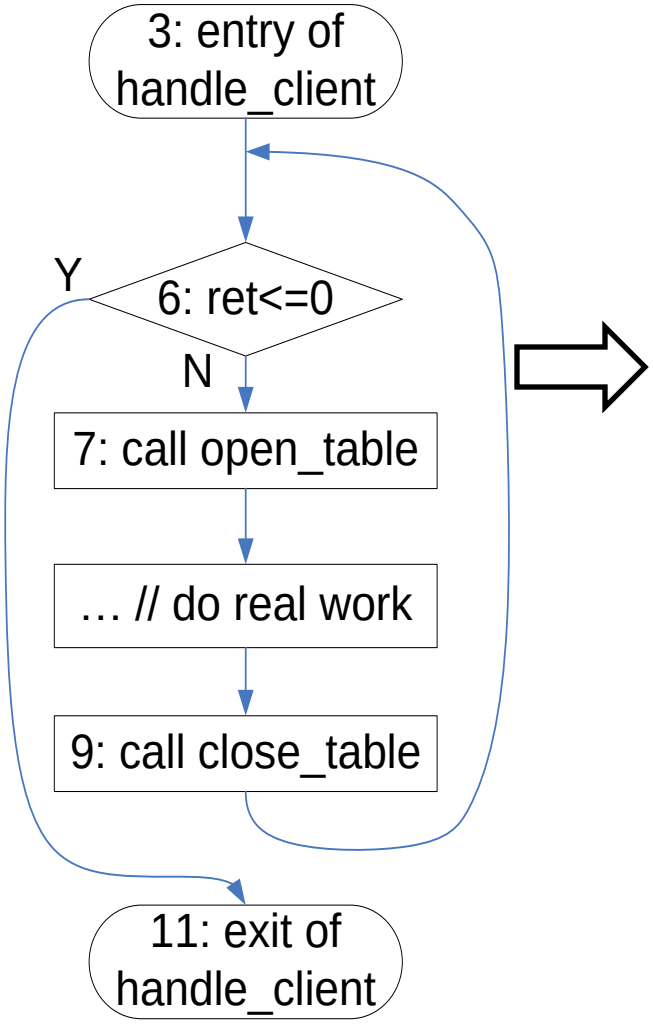


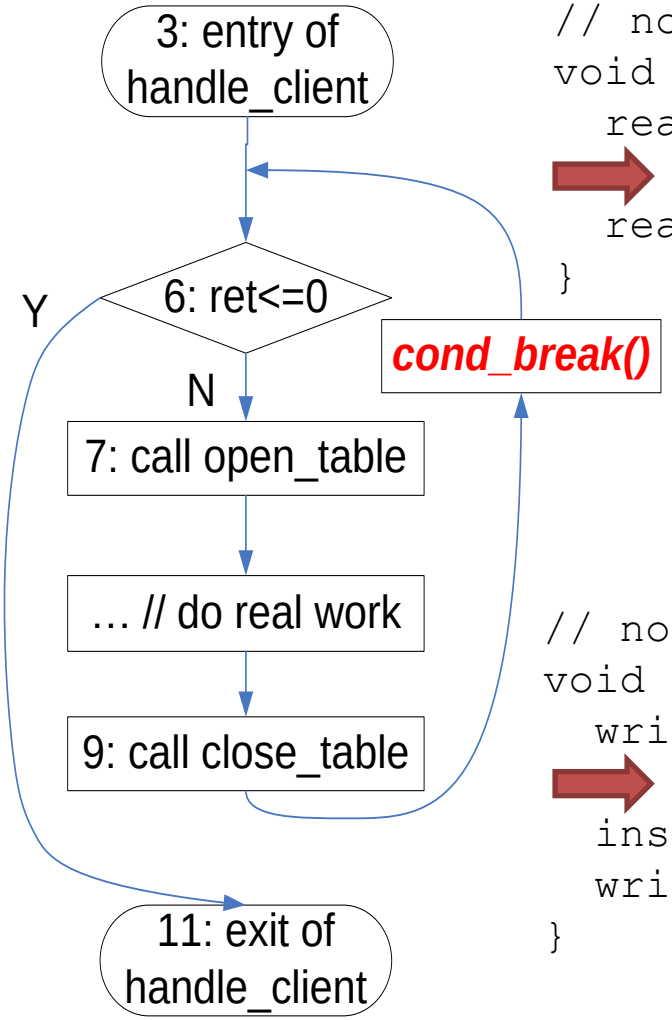
Control Application Threads

```
1 : // database worker thread
2 : void handle_client(int fd) {
3 :   for(;;) {
4 :     struct client_req req;
5 :     int ret = recv(fd, &req, ...);
6 :     if(ret <= 0) break;
7 :     open_table(req.table_id);
8 :     ... // do real work
9 :     close_table(req.table_id);
10:   }
11: }
```



Control Application Threads (cont'd)

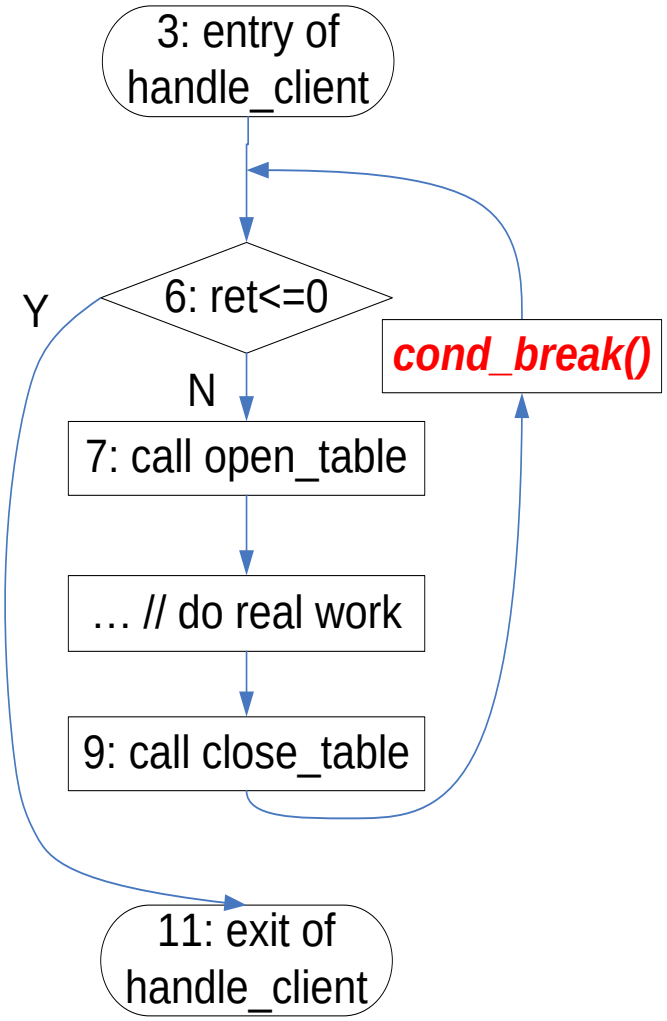





```
// not the final version
void cond_break() {
  read_unlock(&update);
  read_lock(&update);
}
```

```
// not the final version
void loom_update() {
  write_lock(&update);
  install_filter();
  write_unlock(&update);
}
```

Pausing Threads at Safe Locations



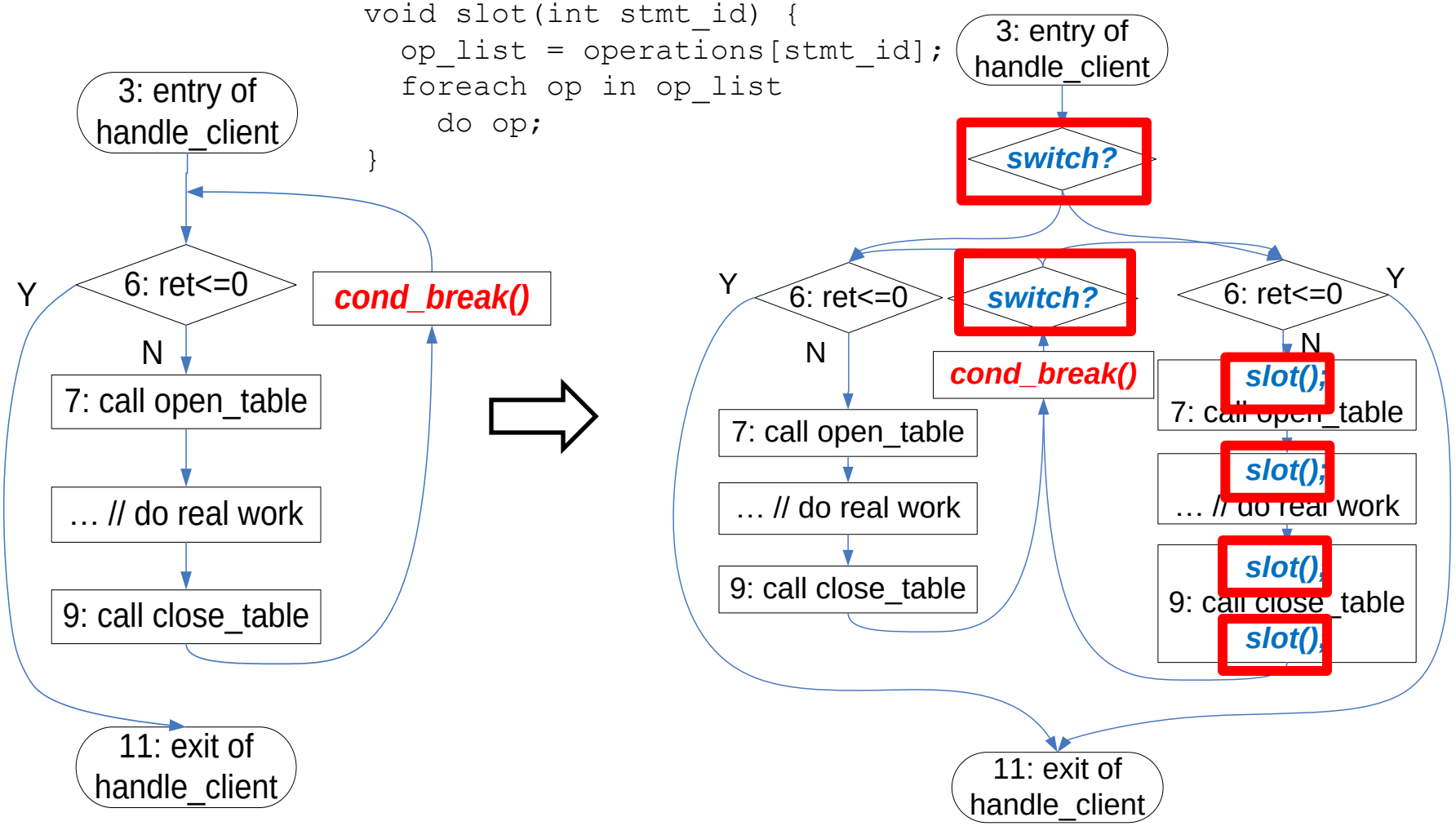

```
        cmpl 0x0, 0x845208c
        je 0x804b56d

void cond_break() {
  if [wait backedge_id]) {
    read_unlock(&update);
    while [wait backedge_id]);
    read_lock(&update);
  }
}

void loom_update() {
  identify_safe_locations();
  for each safe backedge E
    [wait E] = true;
  write_lock(&update);
  install_filter();
  for each safe backedge E
    [wait E] = false;
  write_unlock(&update);
}
```

Outline

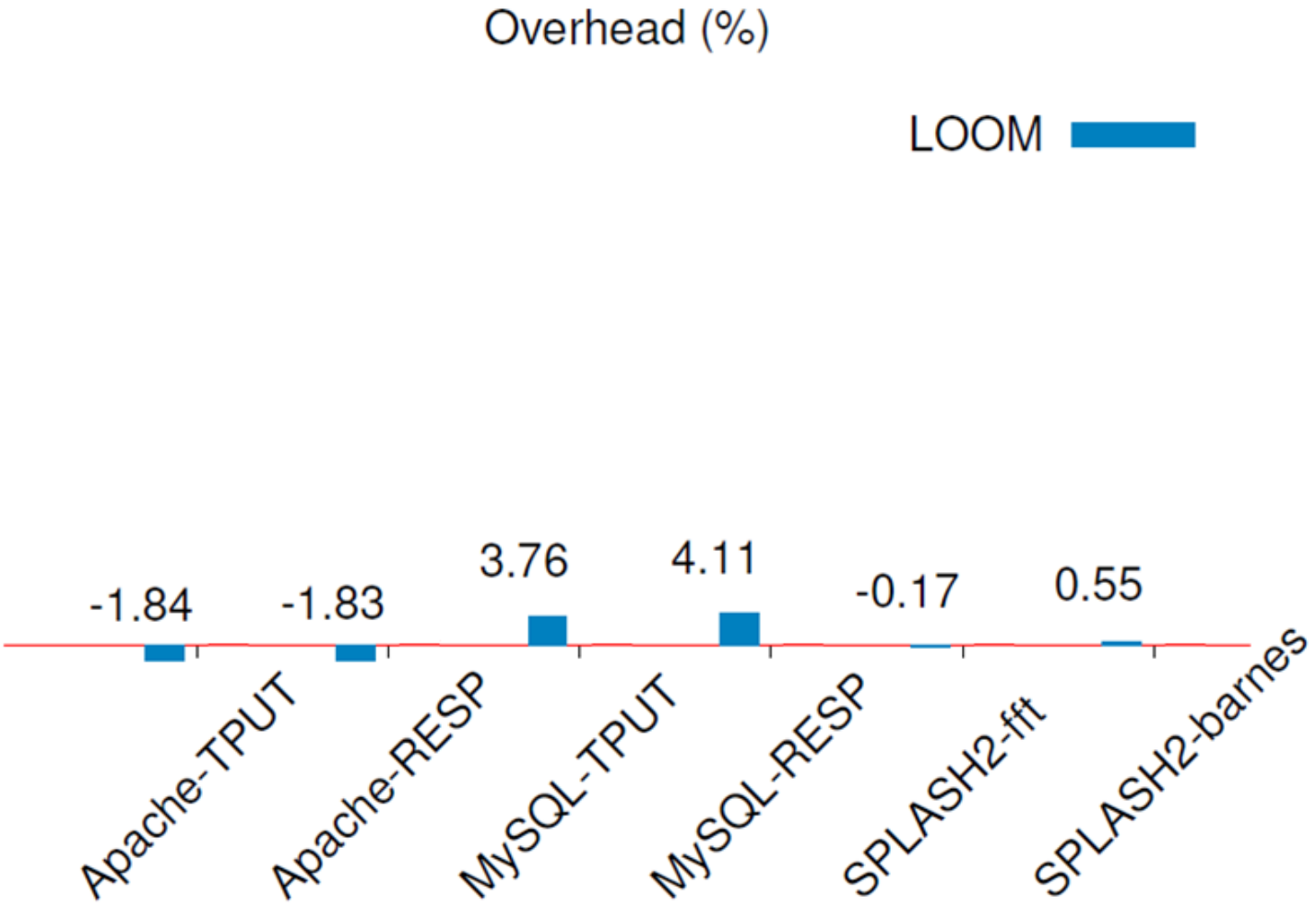
- Architecture
  - Combines static preparation and live update
- Safely updating live applications
- Reducing performance overhead
- Evaluation
- Conclusion

Hybrid Instrumentation

```
void slot(int stmt_id) {
  op_list = operations[stmt_id];
  foreach op in op_list
    do op;
}
```

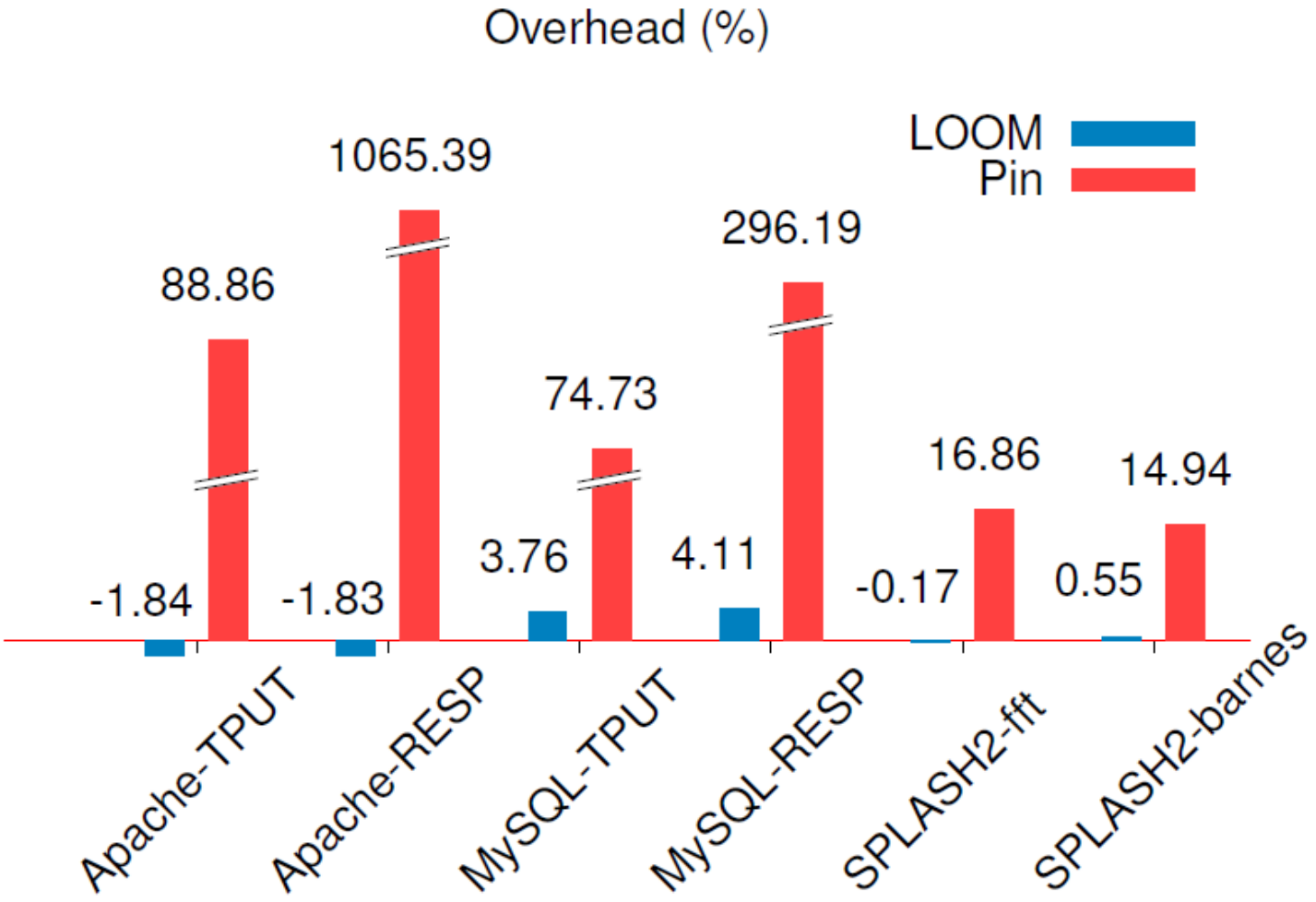


Bare Instrumentation Overhead



Performance overhead < 5%

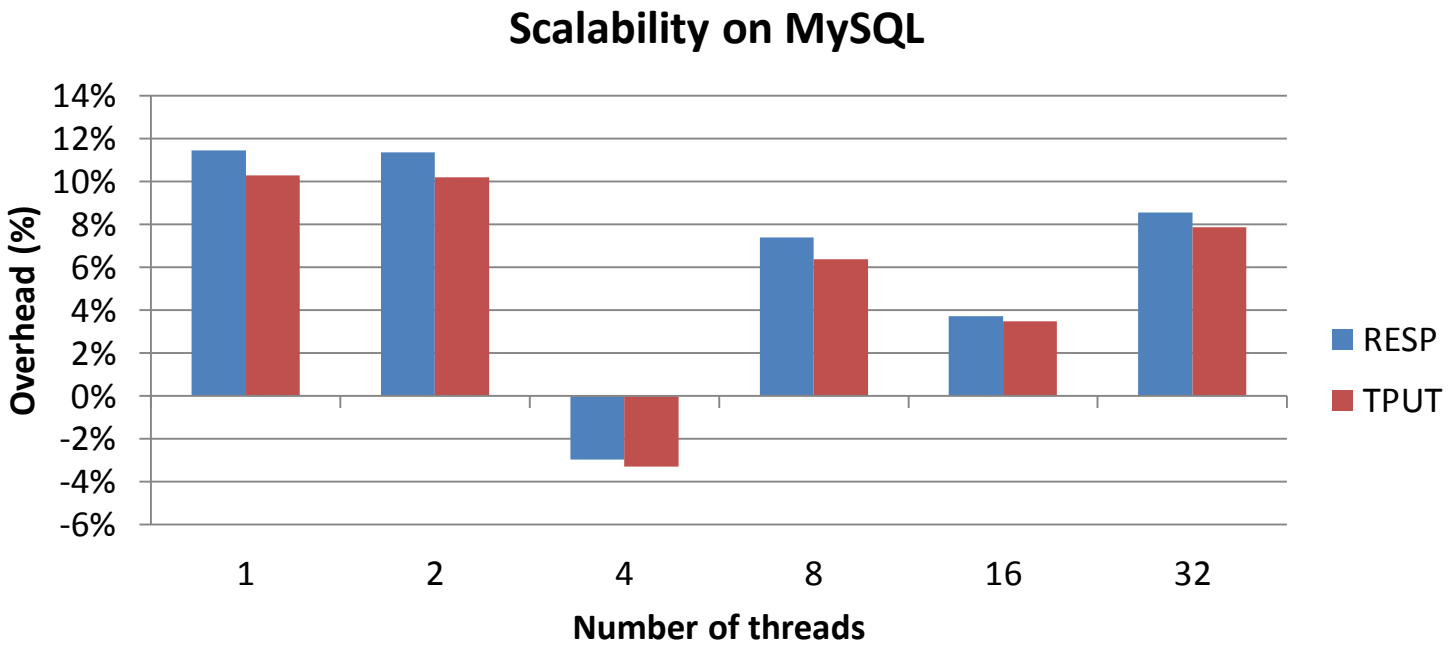
Bare Instrumentation Overhead



Performance overhead < 5%

Scalability

- 48-core machine with 4 CPUs; Each CPU has 12 cores.
- Pin the server to CPU 0, 1, 2, and the client to CPU 3.



Performance overhead does not increase

Conclusion

- LOOM: A live-workaround system designed to quickly and safely bypass races
  - **Execution filters:** easy to use and flexible (< 5 lines)
  - **Evacuation algorithm:** safe
  - **Hybrid instrumentation:** fast (overhead < 5%) and scalable (overhead < 10% with 32 threads)
- Future work
  - Generic hybrid instrumentation framework
  - Extend the idea to other classes of errors

Questions?