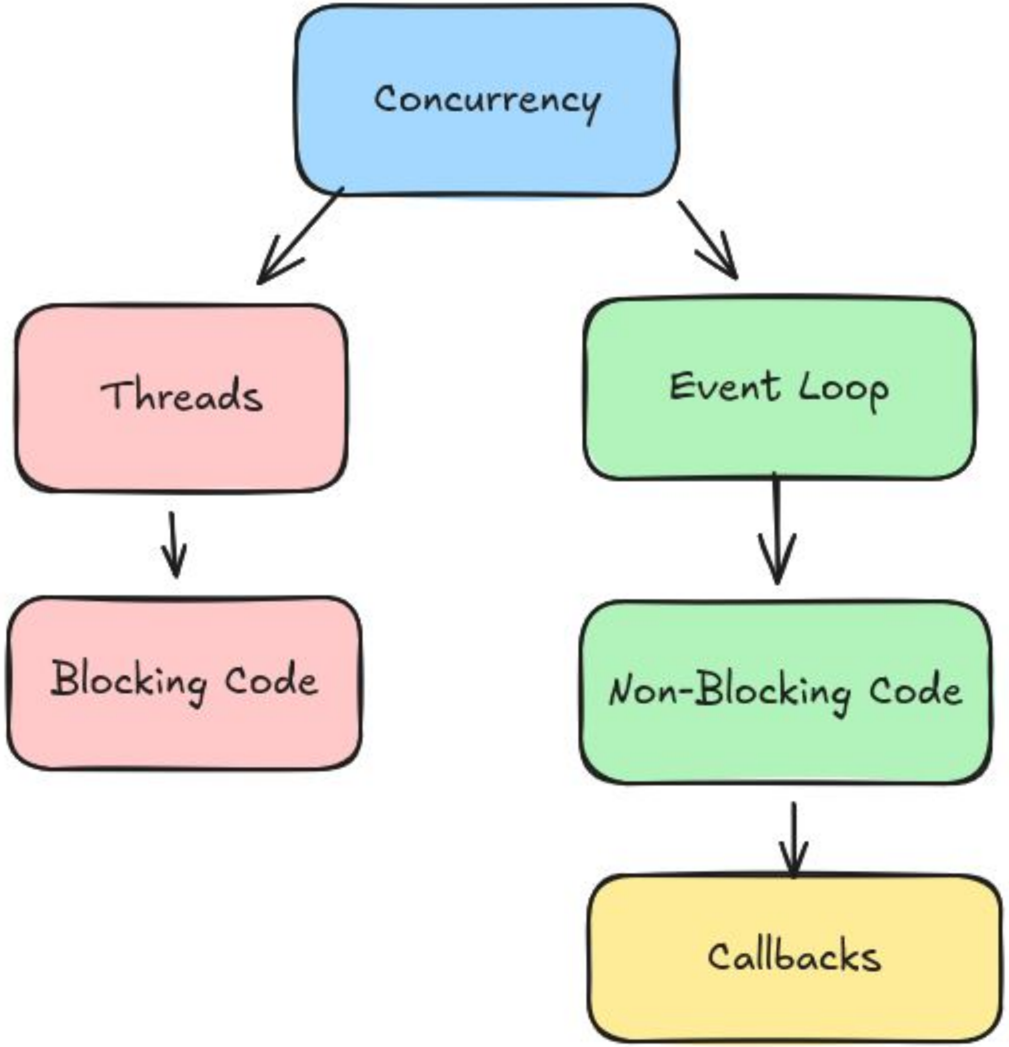
Designing Fibers for systemd

Structured POSIX avoidance in PID 1

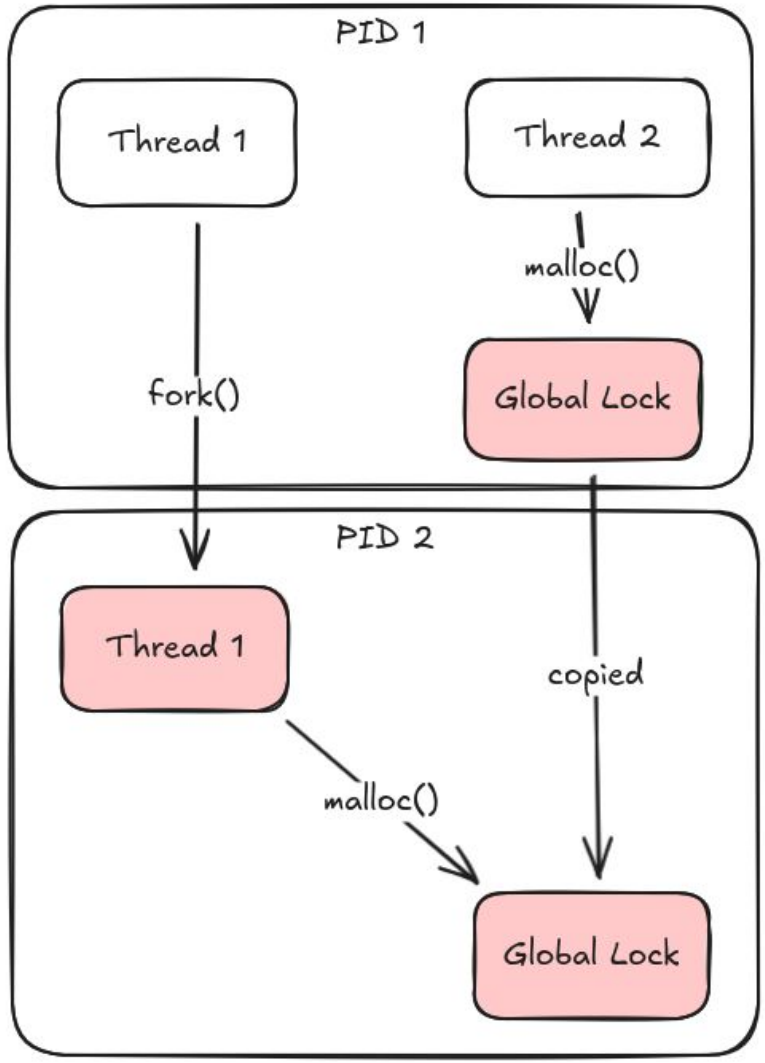
Background

- systemd has a lot of daemons
- These daemons have to handle requests concurrently

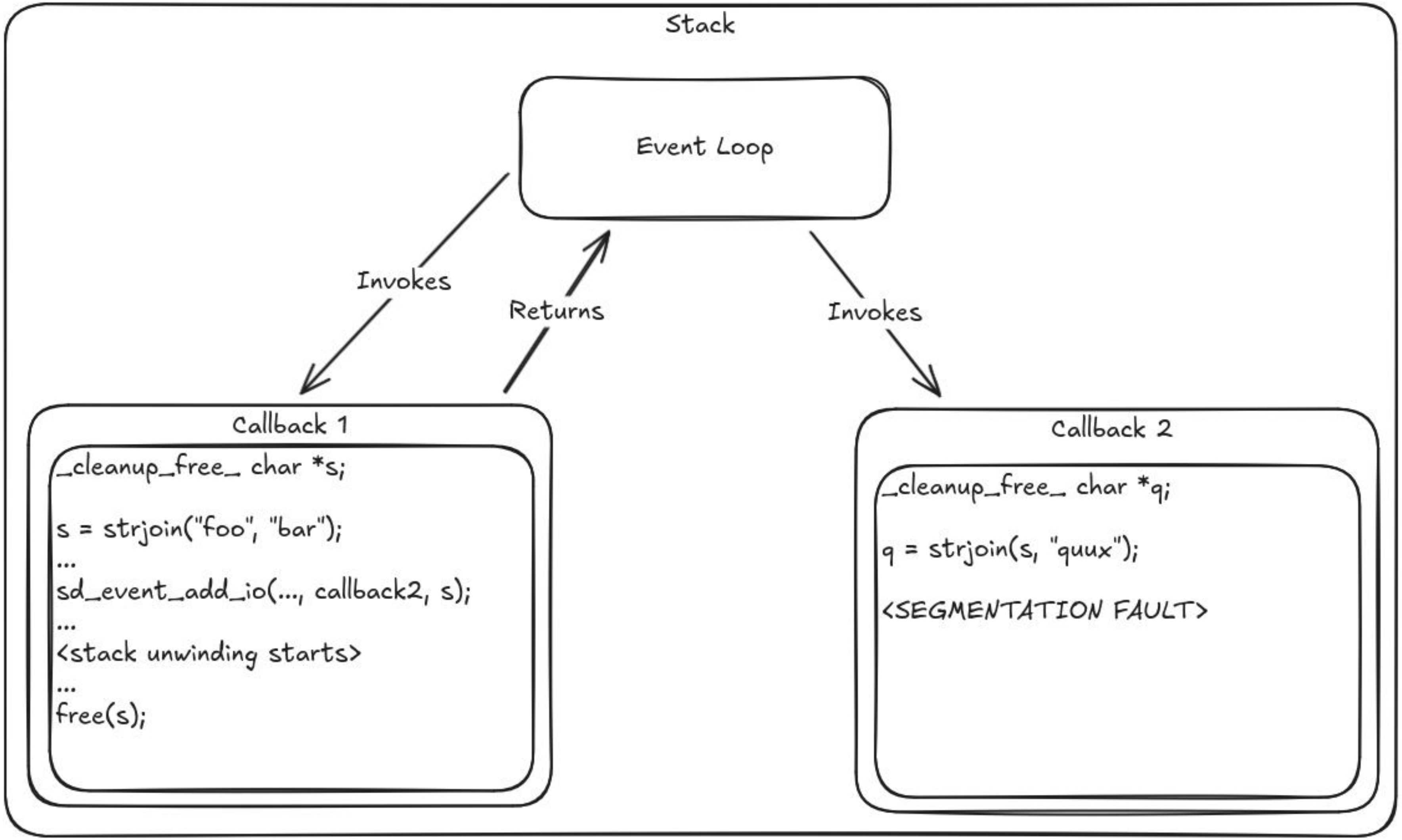

Concurrency
Threads
Event Loop
Blocking Code
Non-Blocking Code
Callbacks

Threads + fork() == pain

- Signal handlers face similar problems

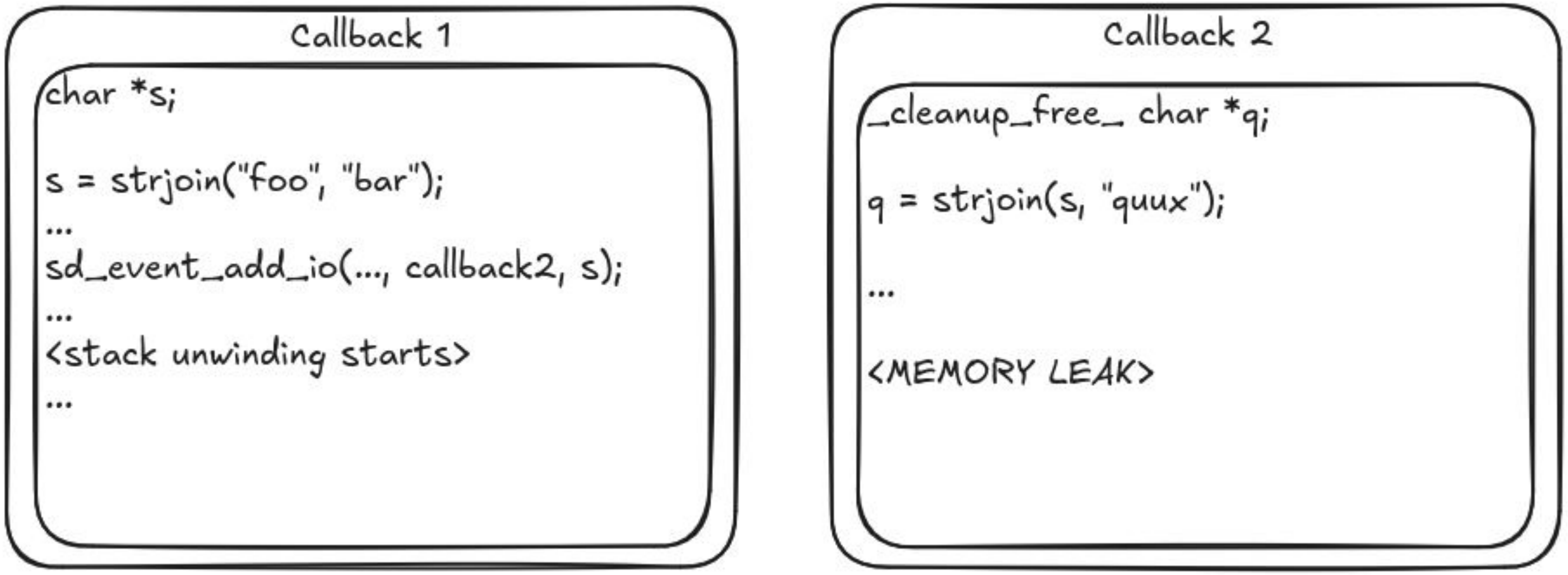


Event Loop + Callbacks == pain



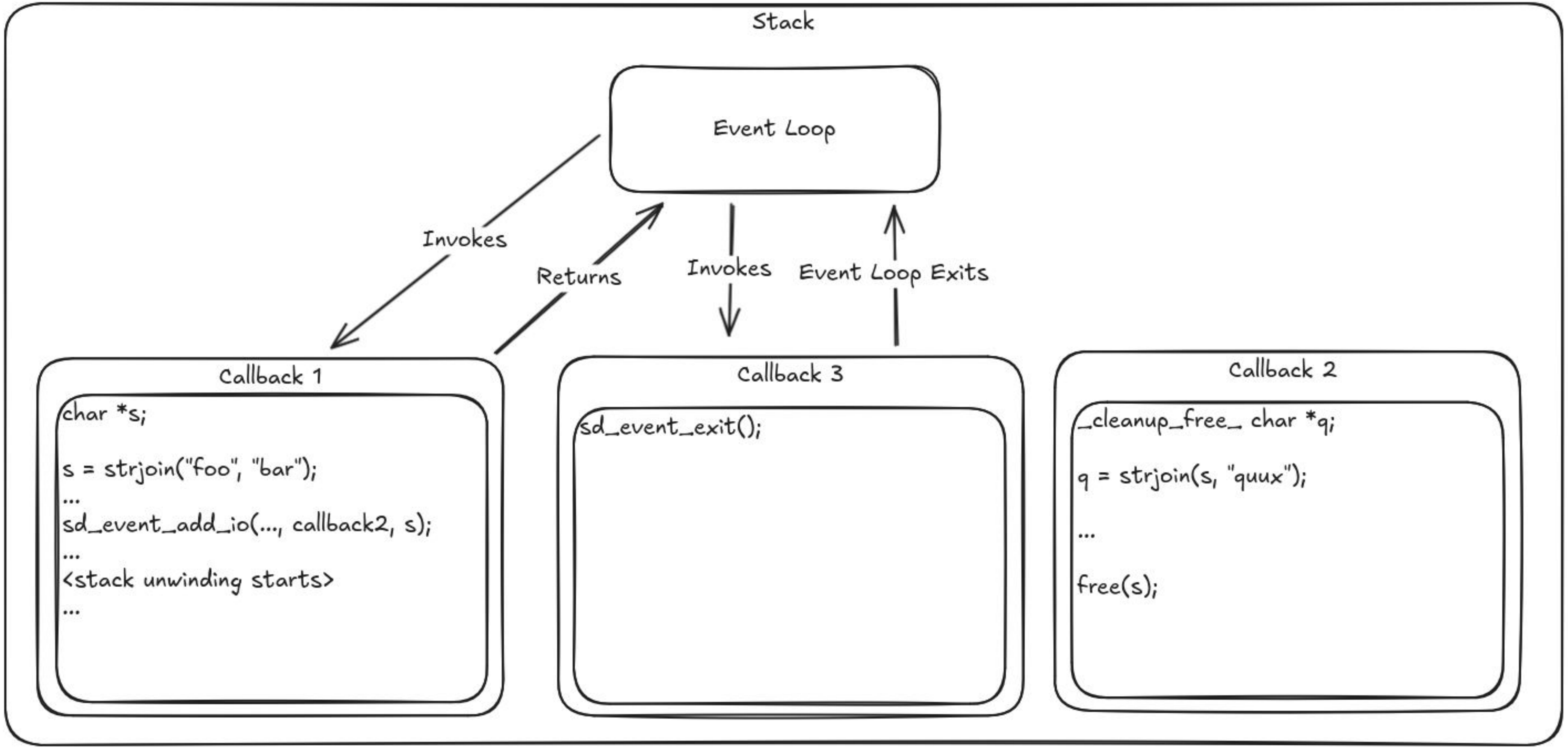
```
_cleanup_free_ char *s;

s = strjoin("foo", "bar");
...
sd_event_add_io(..., callback2, s);
...
<stack unwinding starts>
...
free(s);
```

```
_cleanup_free_ char *q;

q = strjoin(s, "quux");

<SEGMENTATION FAULT>
```



```
char *s;

s = strjoin("foo", "bar");
...
sd_event_add_io(..., callback2, s);
...
<stack unwinding starts>
...
```

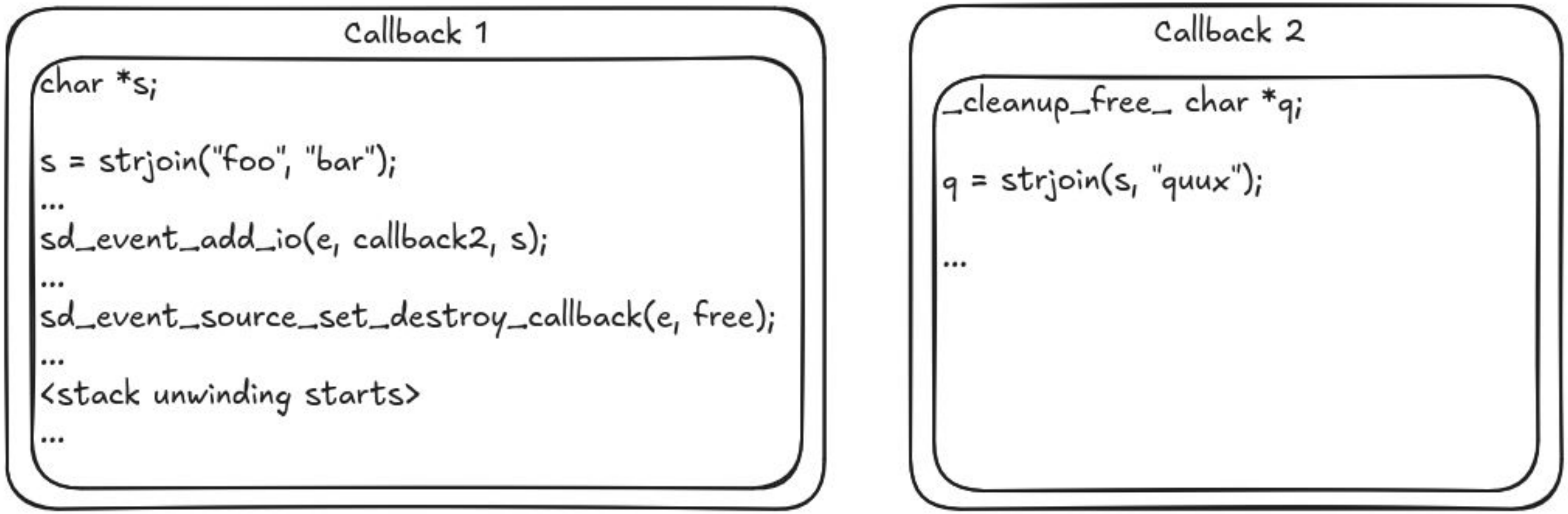
```
_cleanup_free_ char *q;

q = strjoin(s, "quux");

...

<MEMORY LEAK>
```



```
char *s;

s = strjoin("foo", "bar");
...
sd_event_add_io(..., callback2, s);
...
<stack unwinding starts>
...
```

```
sd_event_exit();
```

```
_cleanup_free_ char *q;

q = strjoin(s, "quux");

...

free(s);
```



Callback 1

```
char *s;

s = strjoin("foo", "bar");
...
sd_event_add_io(e, callback2, s);
...
sd_event_source_set_destroy_callback(e, free);
...
<stack unwinding starts>
...
```

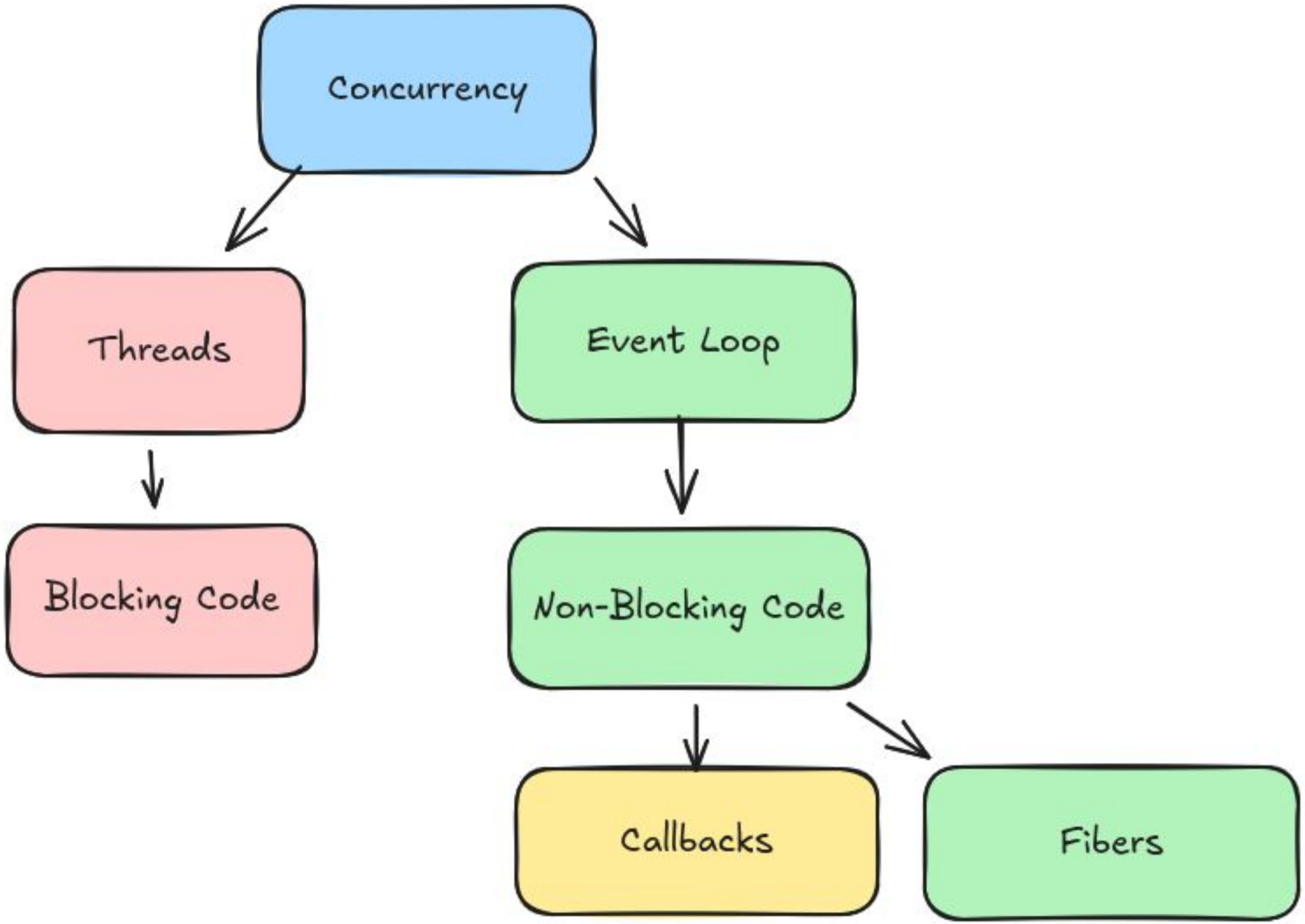
Callback 2

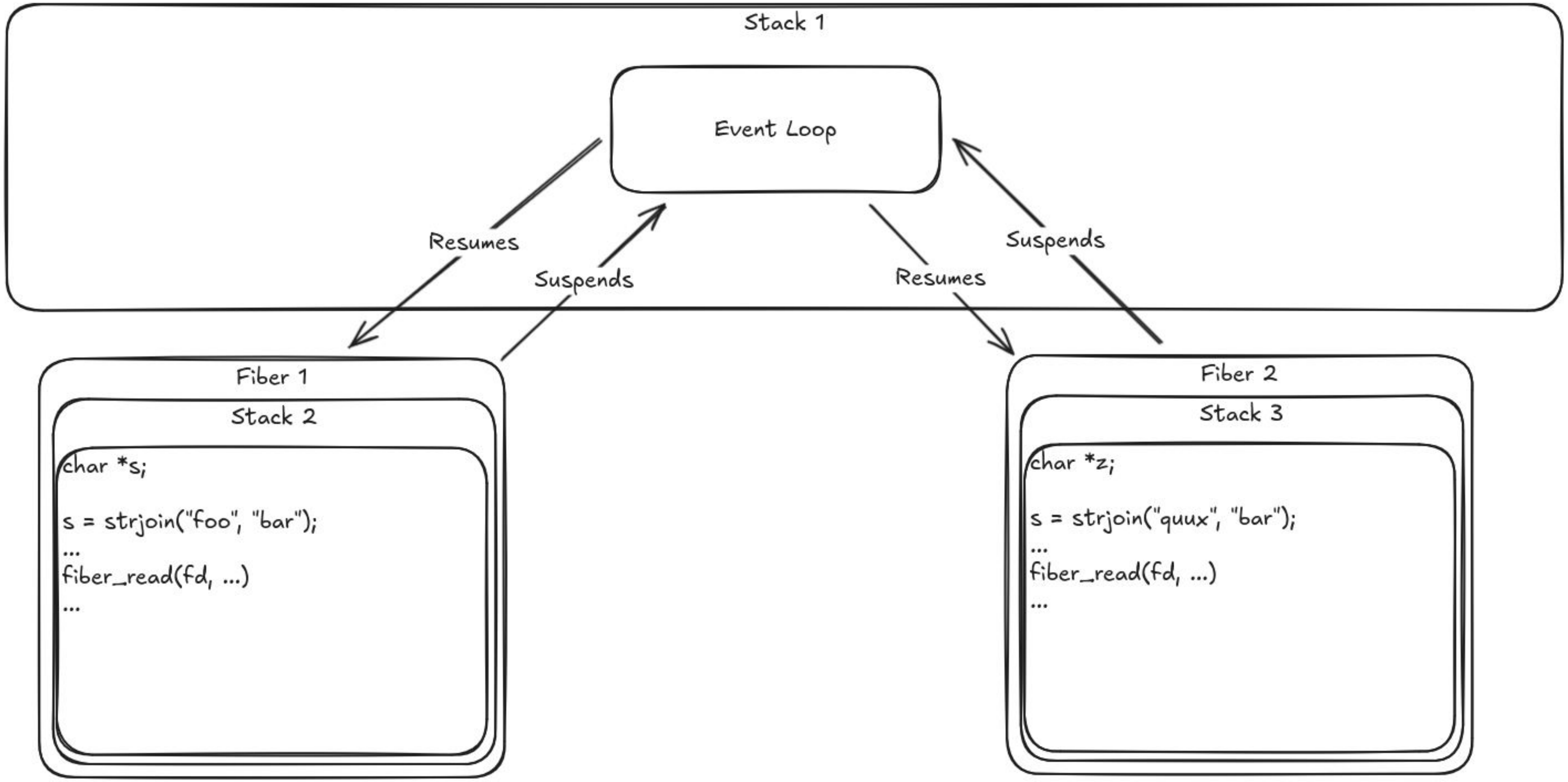
```
_cleanup_free_ char *q;

q = strjoin(s, "quux");

...
```

Threads vs Callbacks
=> Stack Unwinding

Can we have the best of both worlds?

- Can we have separate stacks without separate threads?
- Can we pass around an entire stack between individual callbacks?


Concurrency
Threads
Event Loop
Blocking Code
Non-Blocking Code
Callbacks
Fibers



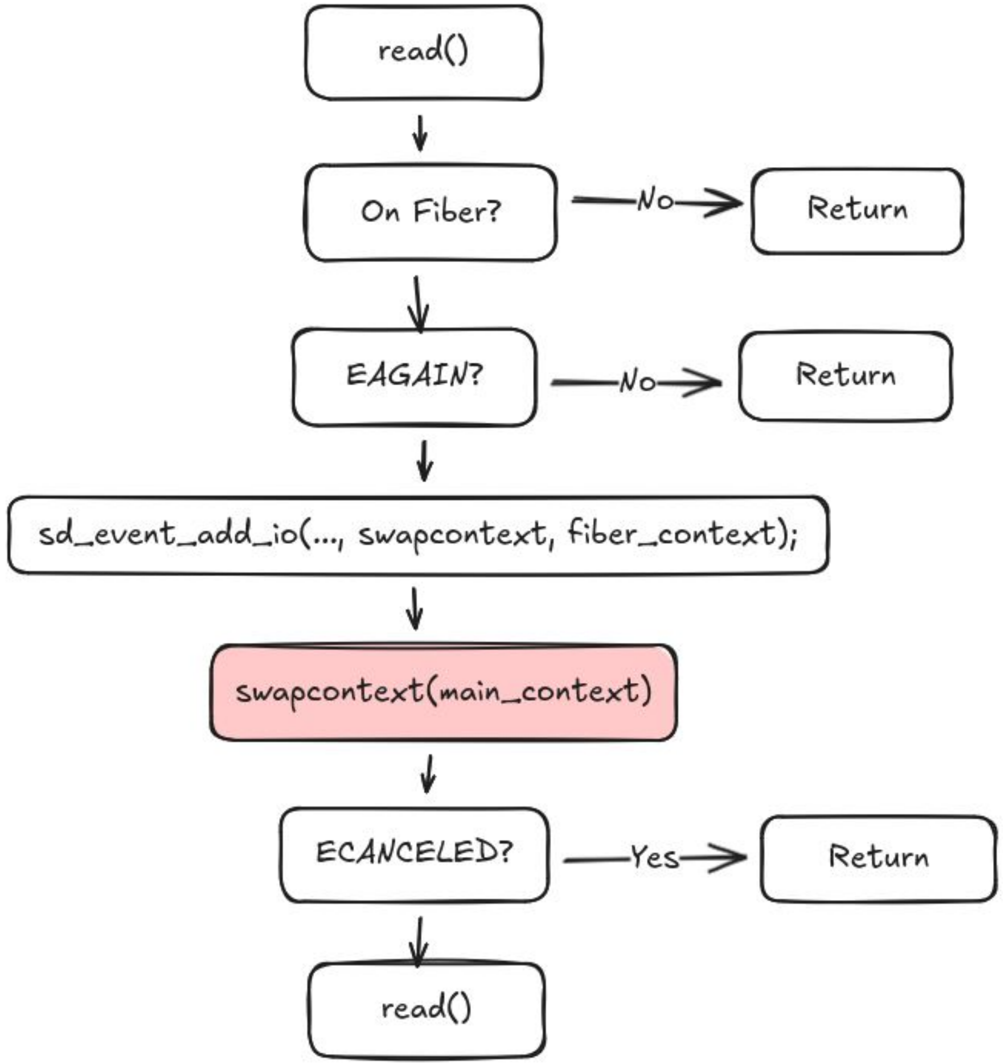
```
char *s;

s = strjoin("foo", "bar");
...
fiber_read(fd, ...)
...
```

```
char *z;

s = strjoin("quux", "bar");
...
fiber_read(fd, ...)
...
```

How to implement stack suspend/resume?

- ucontext.h
- getcontext()
- makecontext()
- swapcontext()

fiber_read()


read()
On Fiber?
No
Return
EAGAIN?
No
Return
sd_event_add_io(..., swapcontext, fiber_context);
swapcontext(main_context)
ECANCELED?
Yes
Return
read()

Cancellation

- To ensure proper cleanup for fibers, stack unwinding is essential
- When shutting down a daemon, running fibers have to be interrupted
- Implemented by injecting the ECANCELED error into running fibers
    - And then waiting for the fibers to finish running

Considerations when spawning background fibers

- An existing fiber can spawn more fibers to run tasks in the background
- Thinking about ownership of data passed into background fibers is essential
    - Shared state!
- When in doubt, copy input arguments to each background fiber

Disadvantages?

- No threads, but still need locking, concurrency primitives, …
- Less efficient than callbacks
- Not POSIX! (deprecated)
- swapcontext() not available on musl (need libucontext)
- Worse integration than threads (gdb, coredumps, …)
- Performance?
  - POSIX mandates saving/restoring per thread signal mask for every swapcontext()
- Still no non-blocking disk I/O
  - But, Jens Axboe is working behind the scenes to make io-uring more widely available!

Current status

- PR open: github.com/systemd/systemd/pull/39771
- Introduces new libsystemd interface sd-future.h
  - Internal for now, but might become public in the future
  - Adds support for using fibers with sd-event.h
- Basic operations implemented
  - Socket IO
  - Child processes
  - Sleeping
  - Waiting for other fibers
  - …
- Concurrency primitives still missing

Questions?