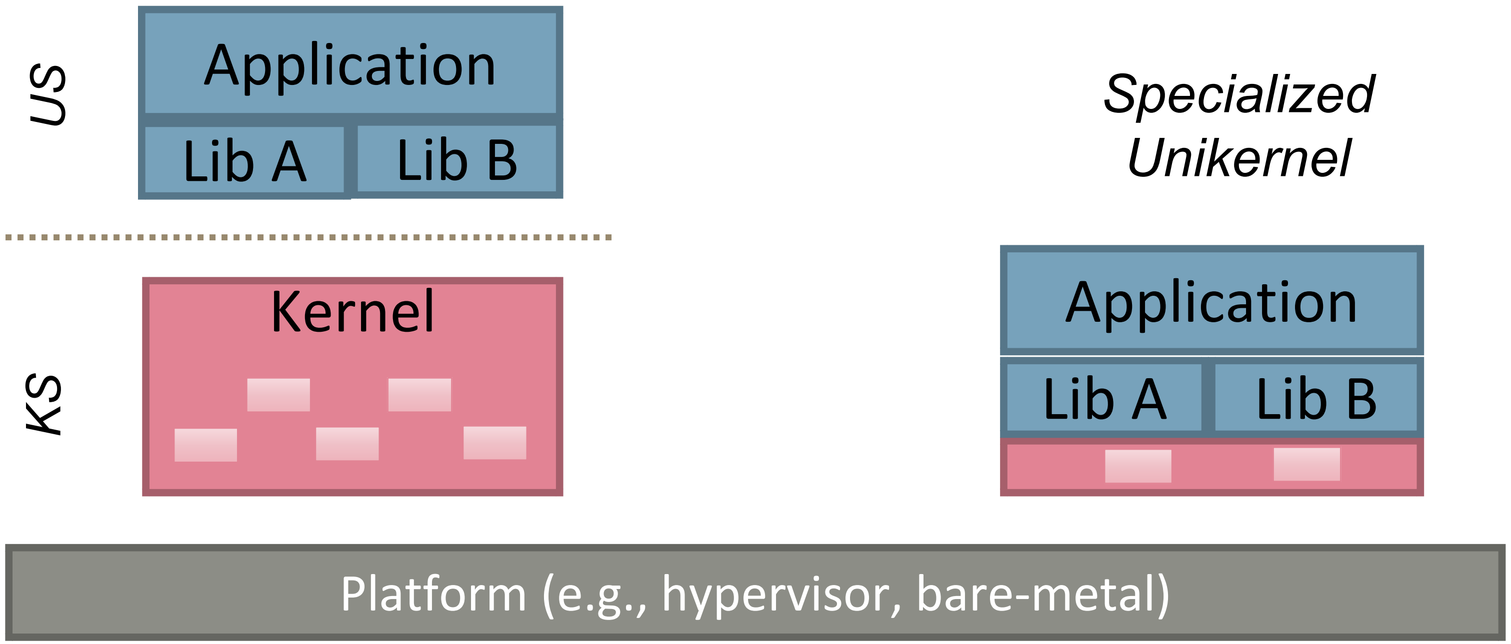
FOSDEM'22
February 5th, 2022
Online Event

Monitoring and Debugging

Simon Kuenzer
CTO at Unikraft UG (haftungsbeschränkt)
Senior Researcher at NEC Laboratories Europe GmbH
simon@unikraft.io

Dr.-Ing. Marc Rittinghaus
Head of Engineering at Unikraft UG (haftungsbeschränkt)
Post-Doc and Researcher at KIT
marc@unikraft.io

Unikraft Unikernel

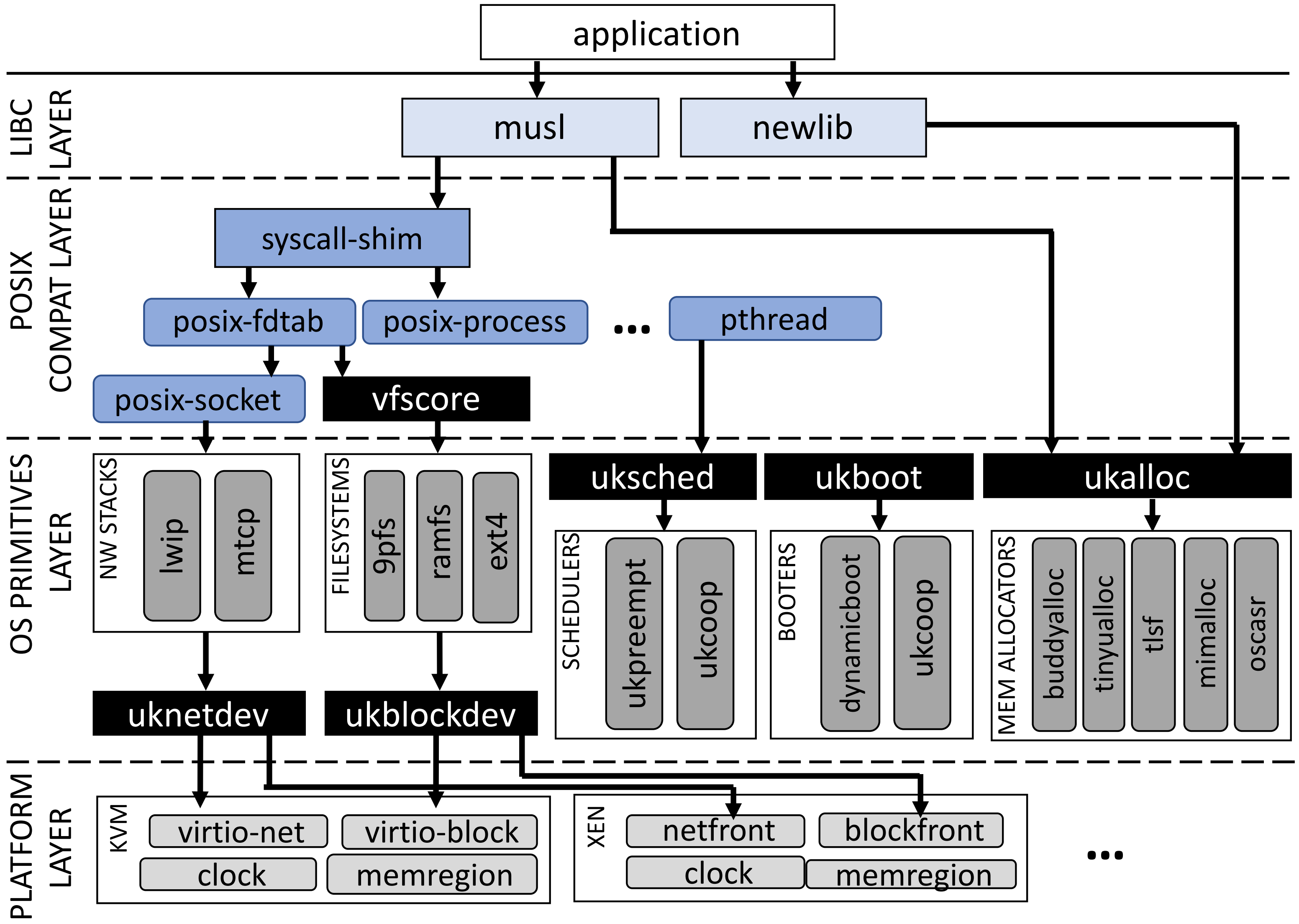


- One application → Flat and single address space
- Single monolithic binary with only necessary kernel components
- Advantages from specialization
  - Performance and efficiency
  - Small TCB and memory footprint
  - Fast boot times

Design Principles

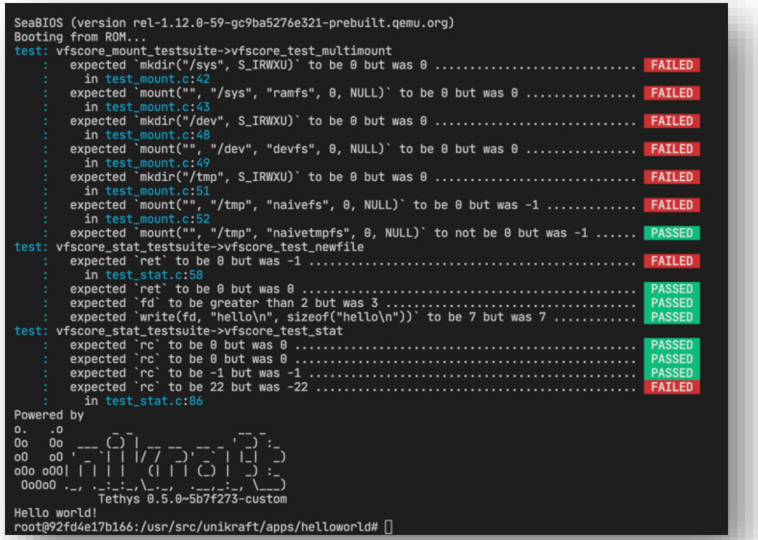
- Specialization as main driving design principle
  - Highly customizable: KPI-driven specialization
- Philosophy: “Everything is a (micro-)library”
  - Decomposed OS primitives
    - Schedulers, memory allocators, VFS, network stacks, …
  - Architectures, platform support, and drivers
    - Virtualization environments, bare-metal
  - Application interfaces
    - POSIX, Linux system call ABI, language runtimes
- Widespread targets
  - Microservices, FaaS, NFV, Edge Computing, (Industrial) IoT and automotive, …

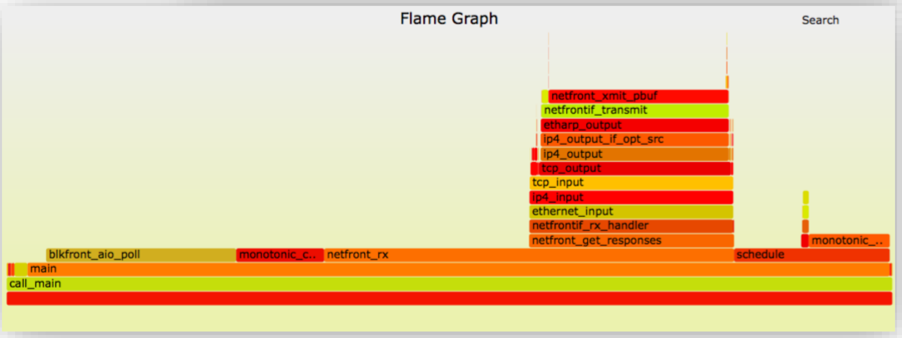
The Unikraft Library Stack



Monitoring and Debugging Features

- ukdebug
  - Logging/Print system
  - Assertions
  - Tracepoints
  - GDB server
- uktest
  - Unit Testing
- ukstore
  - Directory of library getters and setters
- ubsan
  - Detect run-time memory bugs
- Uniprof (tool)
  - Performance analysis with stack snapshots





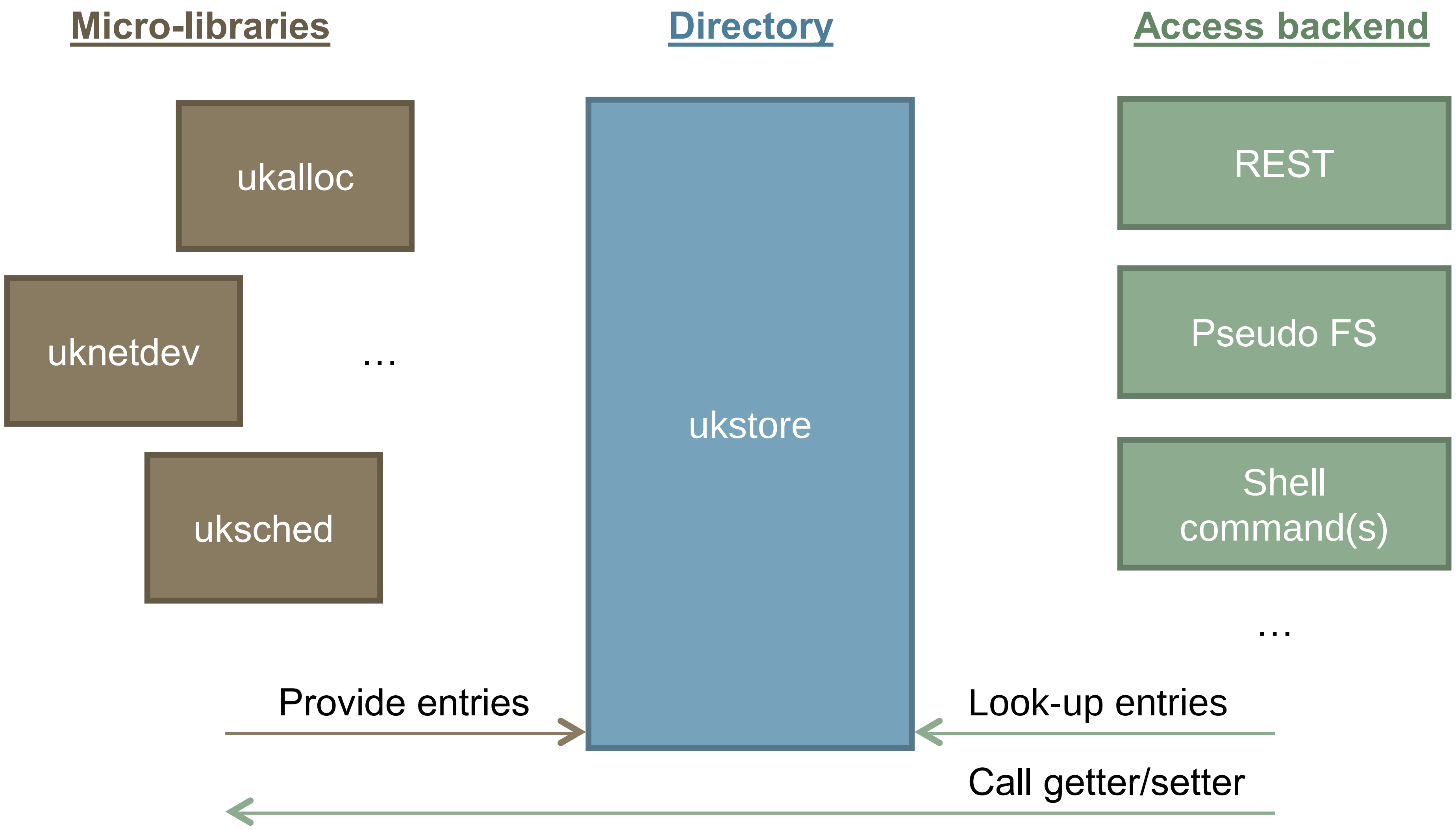
1

Monitoring with ukstore

Requirements

- Re-use (micro-)library instrumentation
- ukstore is optional: Remove unneeded instrumentation at compile-time
- Allow retrieving of data and setting of values
- Getter/Setter interface defined by library
  - Name and data type (e.g., int, string)
- Pull-oriented design
  - Minimal overhead: Compute/parse only when requested
- Enable integration into common visualization/alerting systems
  - e.g., Prometheus, Grafana

Architecture



An ukstore entry

- Name, data type, and function pointer to getter and/or setter
- **Static** entries (always available)
  - Compile-time, no run-time registration

```
[Library ID]/[Entry]
```

Examples:

```
uknetdev/interfaces_count (r-)
ukboot/request_shutdown (-w)
```

- **Dynamic** entries
  - Created and removed at runtime
  - Entries per instance/object (e.g., thread, allocator, network interface)

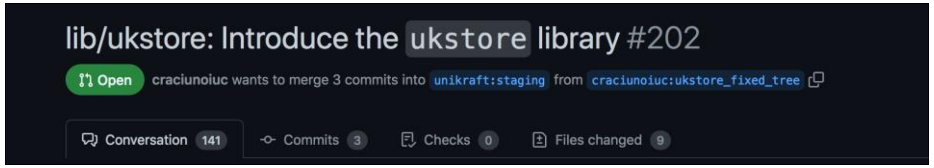
```
[Library ID]/[Object ID]/[Entry]
```

Examples:

```
uknetdev/1/sent_bytes (r-)
ukalloc/0/avail_mem (r-)
```

Example: Grafana/Prometheus with ukstore

Current State and Future Work



- Currently upstreaming ukstore
  - https://github.com/unikraft/unikraft/pull/202

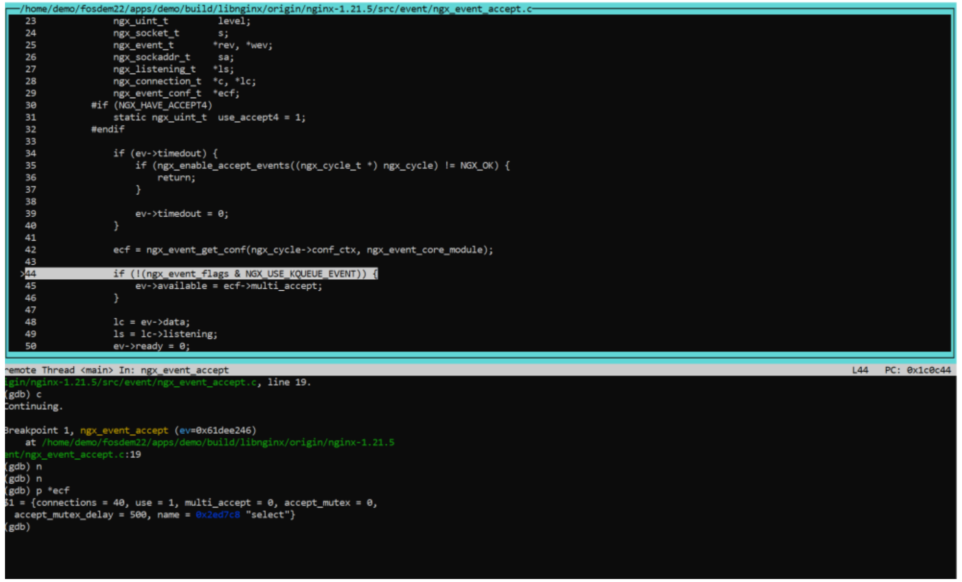
- Next
  - Provide set of initial instrumentation
    - Memory utilization: ukalloc
    - CPU utilization: uksched
    - Network utilization: lwip, uknetdev
    - Storage utilization: vfscore , ukblkdev
    - …
  - Access backends
    - e.g., Prometheus/REST, pseudo-FS, shell

2

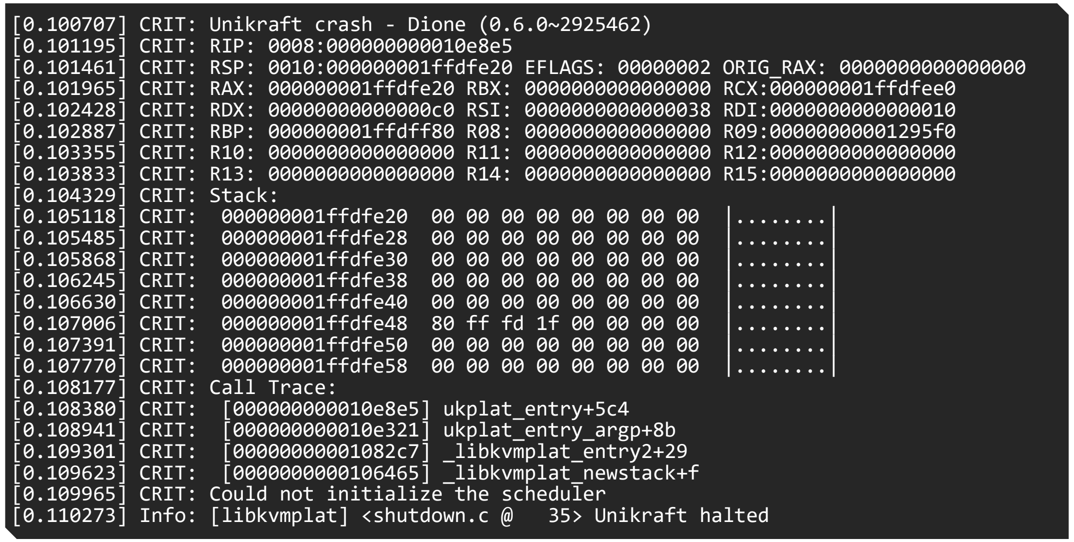
New Debugging Features in Unikraft (ukdebug)

New Debugging Features in Unikraft

Integrated GDB Stub


```
/home/demo/fosdem22/apps/demo/build/libnginx/origin/nginx-1.21.5/src/event/ngx_event_accept.c
        ngx_uint_t         level;
        ngx_socket_t       s;
        ngx_event_t       *rev, *wev;
        ngx_sockaddr_t     sa;
        ngx_listening_t   *ls;
        ngx_connection_t  *c, *lc;
        ngx_event_conf_t  *ecf;
    #if (NGX_HAVE_ACCEPT4)
        static ngx_uint_t  use_accept4 = 1;
    #endif

        if (ev->timedout) {
            if (ngx_enable_accept_events((ngx_cycle_t *) ngx_cycle) != NGX_OK) {
                return;
            }

            ev->timedout = 0;
        }

        ecf = ngx_event_get_conf(ngx_cycle->conf_ctx, ngx_event_core_module);

        if (!(ngx_event_flags & NGX_USE_KQUEUE_EVENT)) {
            ev->available = ecf->multi_accept;
        }

        lc = ev->data;
        ls = lc->listening;
        ev->ready = 0;
remote Thread <main> In: ngx_event_accept                                  L44   PC: 0x1c0c44
igin/nginx-1.21.5/src/event/ngx_event_accept.c, line 19.
(gdb) c
Continuing.

Breakpoint 1, ngx_event_accept (ev=0x61dee246)
    at /home/demo/fosdem22/apps/demo/build/libnginx/origin/nginx-1.21.5
ent/ngx_event_accept.c:19
(gdb) n
(gdb) n
(gdb) p *ecf
$1 = {connections = 40, use = 1, multi_accept = 0, accept_mutex = 0,
  accept_mutex_delay = 500, name = 0x2ed7c8 "select"}
(gdb)
```

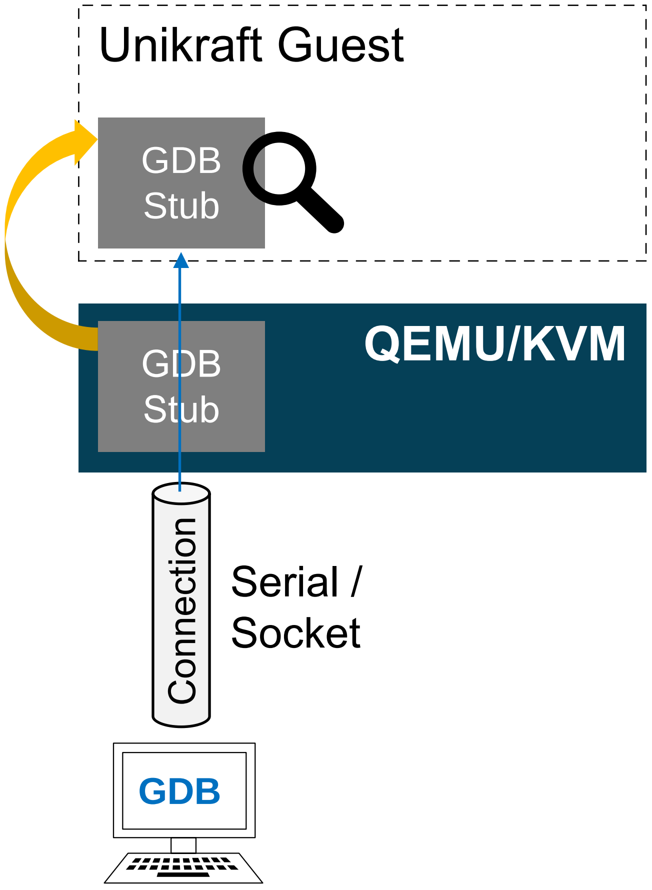
Uniform Crash Screen


```
[0.100707] CRIT: Unikraft crash - Dione (0.6.0~2925462)
[0.101195] CRIT: RIP: 0008:000000000010e8e5
[0.101461] CRIT: RSP: 0010:000000001ffdfe20 EFLAGS: 00000002 ORIG_RAX: 0000000000000000
[0.101965] CRIT: RAX: 000000001ffdfe20 RBX: 0000000000000000 RCX:000000001ffdfee0
[0.102428] CRIT: RDX: 00000000000000c0 RSI: 0000000000000038 RDI:0000000000000010
[0.102887] CRIT: RBP: 000000001ffdff80 R08: 0000000000000000 R09:00000000001295f0
[0.103355] CRIT: R10: 0000000000000000 R11: 0000000000000000 R12:0000000000000000
[0.103833] CRIT: R13: 0000000000000000 R14: 0000000000000000 R15:0000000000000000
[0.104329] CRIT: Stack:
[0.105118] CRIT:  000000001ffdfe20  00 00 00 00 00 00 00 00  |........|
[0.105485] CRIT:  000000001ffdfe28  00 00 00 00 00 00 00 00  |........|
[0.105868] CRIT:  000000001ffdfe30  00 00 00 00 00 00 00 00  |........|
[0.106245] CRIT:  000000001ffdfe38  00 00 00 00 00 00 00 00  |........|
[0.106630] CRIT:  000000001ffdfe40  00 00 00 00 00 00 00 00  |........|
[0.107006] CRIT:  000000001ffdfe48  80 ff fd 1f 00 00 00 00  |........|
[0.107391] CRIT:  000000001ffdfe50  00 00 00 00 00 00 00 00  |........|
[0.107770] CRIT:  000000001ffdfe58  00 00 00 00 00 00 00 00  |........|
[0.108177] CRIT: Call Trace:
[0.108380] CRIT:  [000000000010e8e5] ukplat_entry+5c4
[0.108941] CRIT:  [000000000010e321] ukplat_entry_argp+8b
[0.109301] CRIT:  [00000000001082c7] _libkvmplat_entry2+29
[0.109623] CRIT:  [0000000000106465] _libkvmplat_newstack+f
[0.109965] CRIT: Could not initialize the scheduler
[0.110273] Info: [libkvmplat] <shutdown.c @   35> Unikraft halted
```

GDB Debugger Support

- QEMU/KVM provides GDB debugger stub
  - Source-level guest debugging
  - Single-stepping, breakpoints, etc.

- BUT:
  - No debugging support on other platforms (e.g., Hyper-V, bare metal, cloud)
  - Semantic gap (e.g., no thread-level debugging)
  - No debugger integration in crash handling (e.g., failed asserts, kernel crash)



Want: Guest-level debugger support
GDB is obvious choice

Needed Components

- Communication channel
  - Should be available early in the boot phase
- GDB stub
  - Processing of GDB commands
- Means to react to debugging events
  - Architectural events (traps, breakpoints)
- Debugger invocation in error conditions
  - Failed assert, kernel crash
- But: Adhere to Unikraft philosophy
  - Implement as optional / replaceable micro-library

Communication Channel

- Unikraft has very short boot phase
  - System far into the boot when network is available
- Serial device
  - No requirements on other subsystems (e.g., memory allocator)
  - Available on most platforms
  - Quick to setup
  - Simple to use
- But: Already used for kernel messages
  - Share with serial console
  - Dedicate to debugger

GDB Stub – Protocol Handling

- Responsible for communication with GDB
  - Packet-based protocol
- Packet Data
  - Command + parameters
  - Payload encoded depending on packet type (e.g., as hex string)
- Checksum
  - Two-digit hexadecimal sum of all characters in packet data modulo 256
- GDB stub is mostly parsing of packets
  - 70% (~1000 LoC) for protocol handling
  - 10 commands needed for basic operation

```
$packet-data#checksum
```

Read 8 bytes at memory address `00000000`

```
$m00000000,08#checksum
```

Read CPU register state

```
$g#checksum
```

```
$52AB031F632DC000…
```

EAX EBX

GDB Stub – Architecture Integration

- Responsibilities
  - Save and restore CPU context
  - Read and write memory
  - Set up single-stepping (i.e., set trace flag in EFLAGS register)
  - Implement trap handlers (e.g., breakpoint)

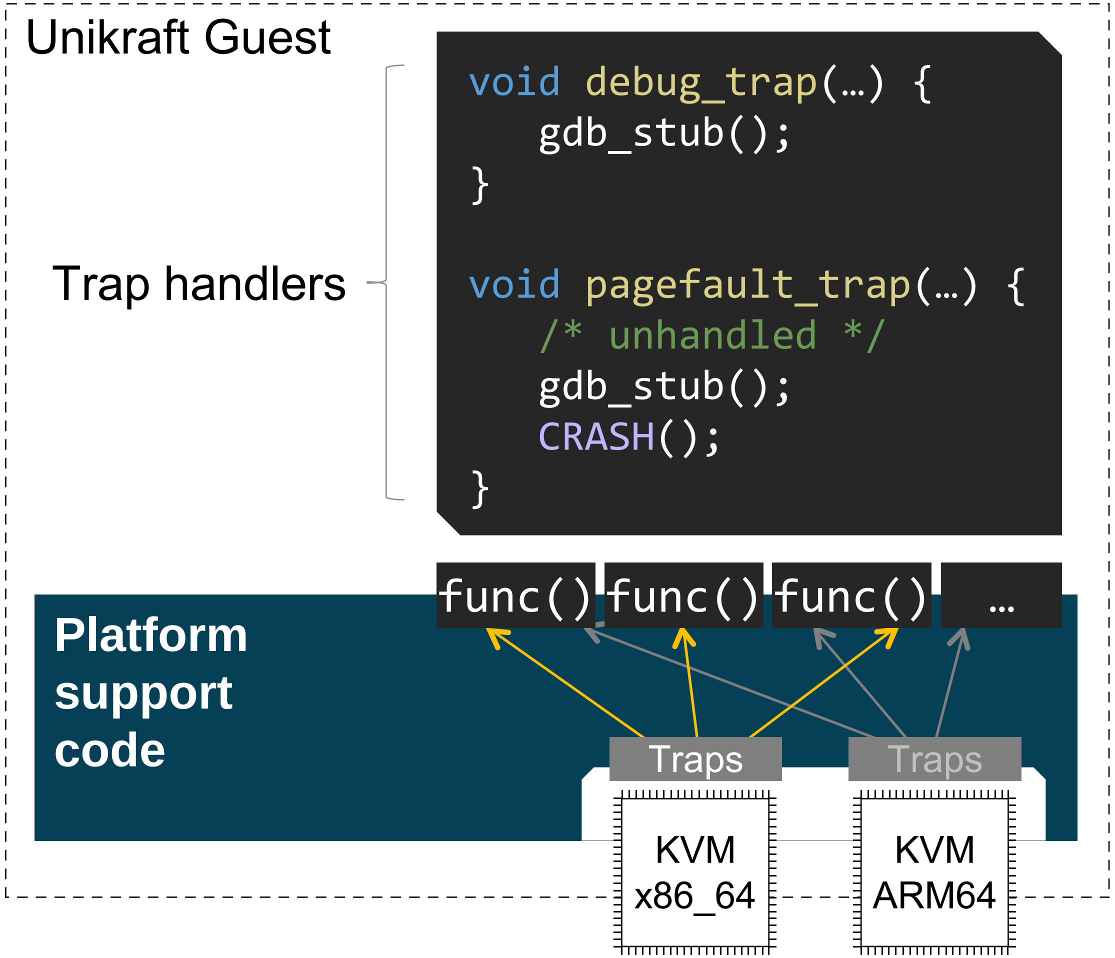
- Setting / unsetting of breakpoints done by GDB client ☺
  - Replace instructions with debug break (memory read/write commands)
  - BUT: HW breakpoints (watchpoints) need support by stub (not yet)

- Unikraft supports: x86-64 (450 LoC) and ARM64 (250 LoC)

Trap Handling

- Debugger must react to traps (e.g., breakpoint, single step)
- Could manually invoke debugger in trap handling code
- BUT: Would create dependency in platform to optional GDB stub library

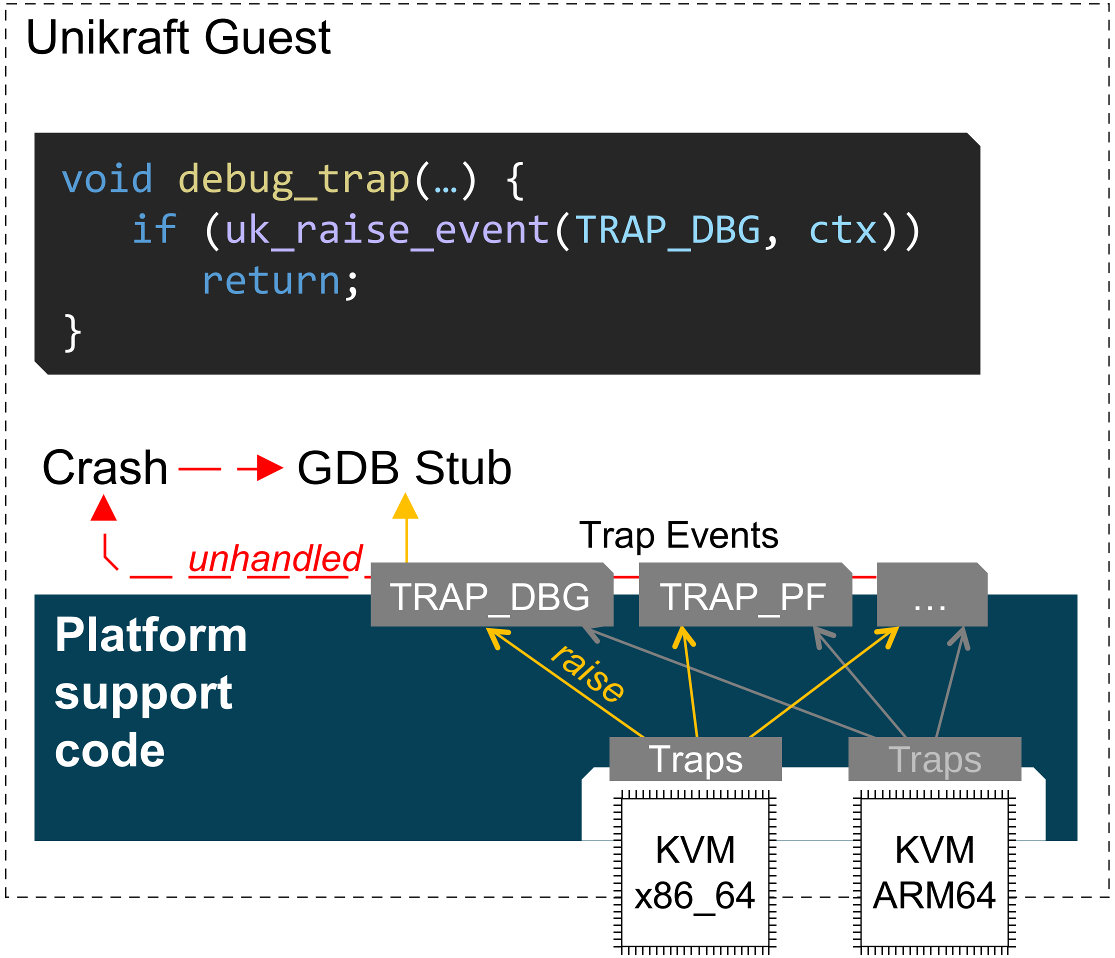
Better: Extensible trap interface



Extensible Trap Interface

- Event-based interface
  - Platform defines and raises events
- Any library can define handlers
  - Link-time handler registration
  - Handler priorities

```
EVENT_HANDLER(TRAP_DBG, gdb_dbg_trap);
```



GDB Stub

- Problem: GDB might access invalid memory addresses
  - Tries to interpret integers as pointers (e.g., in ASM TUI mode)
  - Tries to read from invalid pointer (e.g., during backtrace)
  - User command leads to unintentional invalid memory access

- GDB stub may crash system when performing illegal access

Need way to catch invalid memory accesses

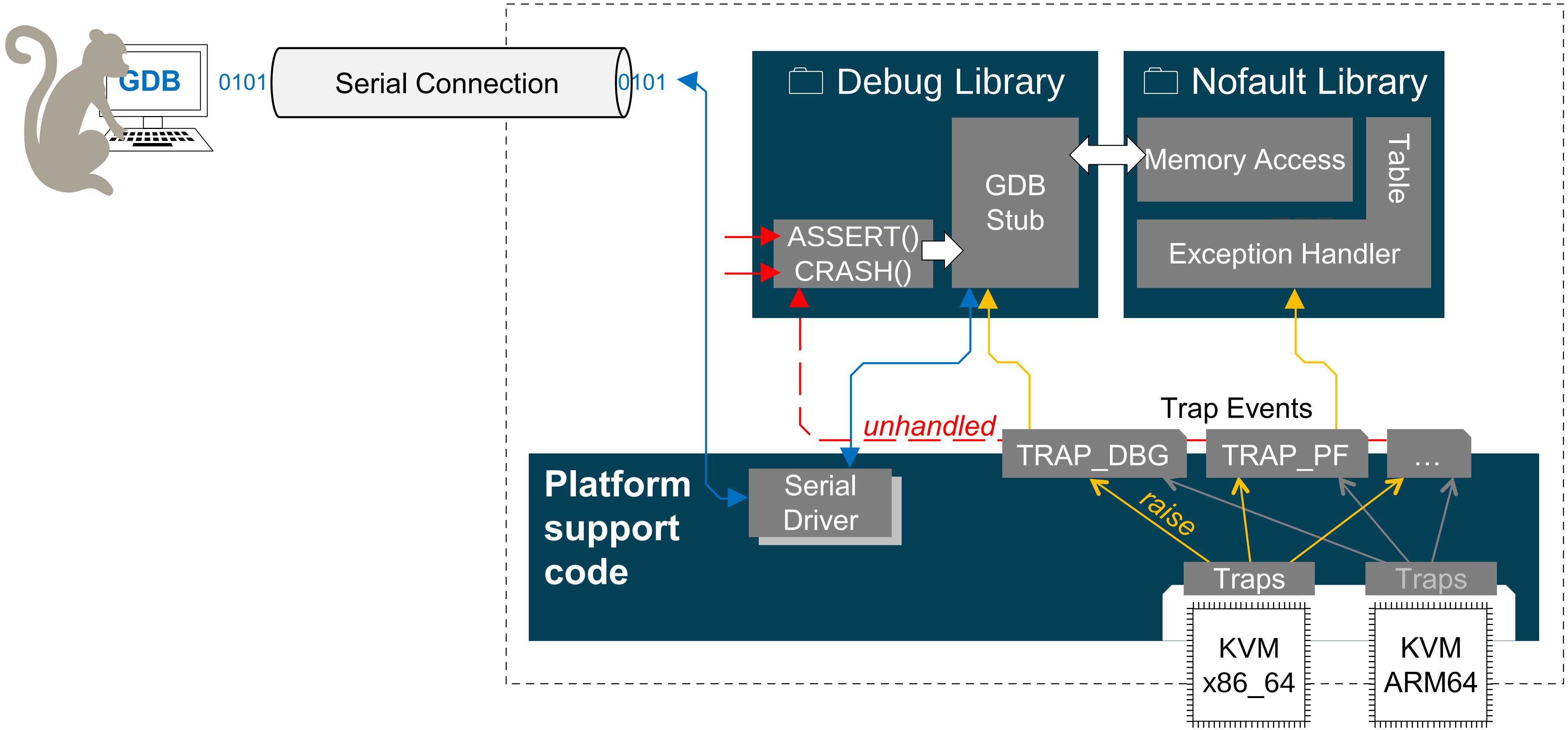
Non-Faulting Memory Accesses

- Want: Try memory access and return error on illegal access (i.e., no crash)
  - Flexible approach: Just try and catch illegal memory accesses

```
int uk_memcpy_nofault(char *dst, const char *src, size_t len) {
    nf_copy_loop(dst, src, len, fault); /* just try memcpy */
    return len;
fault:                                  /* jump here on error */
    return -1;
}
```

- Register low-priority pagefault handler
  - Maps trapped IP to entry in exception table
  - Each `nofault()`-call receives entry in table
  - Entry provides continuation IP for exception handler

Overview


GDB
0101
Serial Connection
0101
Debug Library
Nofault Library
GDB Stub
Memory Access
Table
ASSERT()
CRASH()
Exception Handler
unhandled
Trap Events
TRAP_DBG
TRAP_PF
…
raise
Platform support code
Serial Driver
Traps
Traps
KVM x86_64
KVM ARM64

Uniform Crash Experience

- Previously: Every architecture had its own crash handling code
  - Dumping registers, halting system, etc.

- Want: Uniform experience on all architectures
  - Invoke debugger if available

x86-64

```
[0.100707] CRIT: Unikraft crash - Dione (0.6.0~2925462)
[0.101195] CRIT: RIP: 0008:000000000010e8e5
[0.101461] CRIT: RSP: 0010:000000001ffdfe20 EFLAGS: 00000002 ORIG_RAX: 0000000000000000
[0.101965] CRIT: RAX: 000000001ffdfe20 RBX: 0000000000000000 RCX:000000001ffdfee0
[0.102428] CRIT: RDX: 00000000000000c0 RSI: 0000000000000038 RDI:0000000000000010
[0.102887] CRIT: RBP: 000000001ffdff80 R08: 0000000000000000 R09:00000000001295f0
[0.103355] CRIT: R10: 0000000000000000 R11: 0000000000000000 R12:0000000000000000
[0.103833] CRIT: R13: 0000000000000000 R14: 0000000000000000 R15:0000000000000000
[0.104329] CRIT: Stack:
[0.105118] CRIT:  000000001ffdfe20  00 00 00 00 00 00 00 00  |........|
[0.105485] CRIT:  000000001ffdfe28  00 00 00 00 00 00 00 00  |........|
[0.105868] CRIT:  000000001ffdfe30  00 00 00 00 00 00 00 00  |........|
[0.106245] CRIT:  000000001ffdfe38  00 00 00 00 00 00 00 00  |........|
[0.106630] CRIT:  000000001ffdfe40  00 00 00 00 00 00 00 00  |........|
[0.107006] CRIT:  000000001ffdfe48  80 ff fd 1f 00 00 00 00  |........|
[0.107391] CRIT:  000000001ffdfe50  00 00 00 00 00 00 00 00  |........|
[0.107770] CRIT:  000000001ffdfe58  00 00 00 00 00 00 00 00  |........|
[0.108177] CRIT: Call Trace:
[0.108380] CRIT:  [000000000010e8e5] ukplat_entry+5c4
[0.108941] CRIT:  [000000000010e321] ukplat_entry_argp+8b
[0.109301] CRIT:  [00000000001082c7] _libkvmplat_entry2+29
[0.109623] CRIT:  [0000000000106465] _libkvmplat_newstack+f
[0.109965] CRIT: Could not initialize the scheduler
[0.110273] Info: [libkvmplat] <shutdown.c @   35> Unikraft halted
```

arm64

```
[0.011181] CRIT: Unikraft crash - Dione (0.6.0~2925462)
[0.011750] CRIT: PC : 0000000040107b04
[0.011980] CRIT: LR : 0000000040107ab4
[0.012209] CRIT: SP : 000000005ffffdc0
[0.012418] CRIT: PSTATE: 200003c5
[0.012713] CRIT: X0 : 000000005ffffdd0 X1 : 0000000008000100
…
[0.017126] CRIT: X28: 0000000000000000 X29: 000000005ffffee0
[0.017508] CRIT: Stack:
[0.018804] CRIT:  000000005ffffdc0  e0 fd ff 5f 00 00 00 00  |..._....|
[0.019168] CRIT:  000000005ffffdc8  00 00 00 00 00 00 00 00  |........|
[0.019549] CRIT:  000000005ffffdd0  d0 fd ff 5f 00 00 00 00  |..._....|
[0.019939] CRIT:  000000005ffffdd8  00 01 00 08 00 00 00 00  |........|
[0.020312] CRIT:  000000005ffffde0  00 01 00 00 00 00 00 00  |........|
[0.020679] CRIT:  000000005ffffde8  28 c0 13 40 00 00 00 00  |(..@....|
[0.021022] CRIT:  000000005ffffdf0  cc fd ff 5f 00 00 00 00  |..._....|
[0.021408] CRIT:  000000005ffffdf8  8c fe ff 5f 00 00 00 00  |..._....|
[0.021819] CRIT: Call Trace:
[0.022007] CRIT:  [0000000040107b04] ukplat_entry+424
[0.022724] CRIT:  [00000000401076e0] ukplat_entry_argp+c4
[0.023039] CRIT:  [000000004010450c] _libkvmplat_entry2+3c
[0.023333] CRIT:  [0000000040102080] _libkvmplat_newstack+10
[0.023711] CRIT: Could not initialize the scheduler
[0.024009] Info: [libkvmplat] <shutdown.c @   35> Unikraft halted
```

Uniform Crash Experience

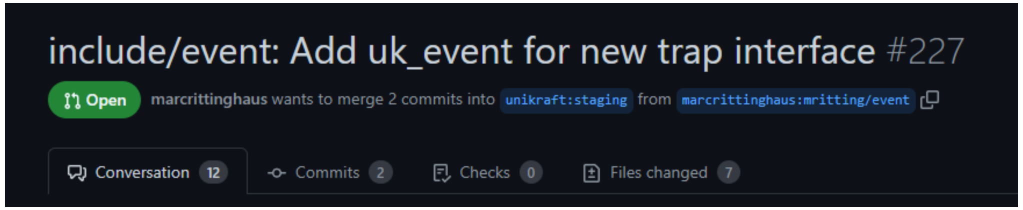
- Symbol resolution
  - Uses <IP, string>-like table
  - Link unikernel three times
    1. Link without table and extract debug symbols
    2. Link with table (might change symbol addresses!). Extract debug symbols again
    3. Link with updated table
  - API to resolve symbols at runtime

```
[0.108177] CRIT: Call Trace:
[0.108380] CRIT:  [000000000010e8e5] ukplat_entry+5c4
[0.108941] CRIT:  [000000000010e321] ukplat_entry_argp+8b
[0.109301] CRIT:  [00000000001082c7] _libkvmplat_entry2+29
[0.109623] CRIT:  [0000000000106465] _libkvmplat_newstack+f
```

```
int uk_resolve_symbol(unsigned long addr, struct uk_symbol *sym);
```

Current State and Future Work

- Currently upstreaming features



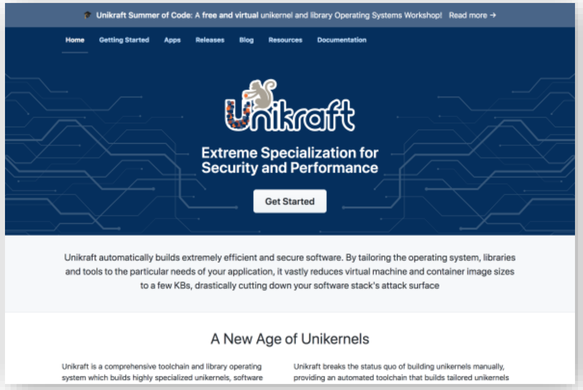
- Next:
  - Thread and SMP support
  - Hardware watchpoints
  - Custom commands
    - Crash dump
    - Inspect state (IRQ, paging, …)
  - Debugging over network connection

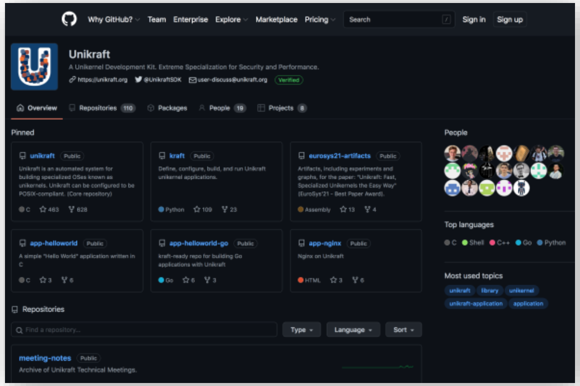
3

The Project, the Company

Join us!

- OSS project
  unikraft.org
- Code & Contributing
  github.com/unikraft
- Documentation
  docs.unikraft.org
- Follow us on
  - Discord: https://bit.ly/UnikraftDiscord
  - Twitter: @UnikraftSDK
  - LinkedIn: https://linkedin.com/company/unikraft-sdk





Want to try out Unikraft at your Company?

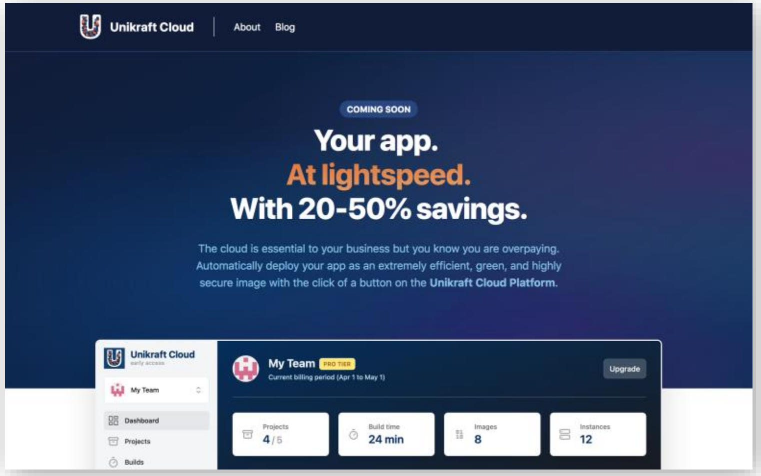
- Please, contact us!
  simon@unikraft.io
  felipe@unikraft.io
  alex@unikraft.io

- **Unikraft UG (haftungsbeschränkt)**
  Im Neuenheimer Feld 582
  69120 Heidelberg
  GERMANY

  https://unikraft.io

Save your free trial!

Heidelberg Old Town, Photo by Brina Blum on Unsplash.com



Thank you!

Unikraft UG (haftungsbeschränkt)
Im Neuenheimer Feld 582
69120 Heidelberg
GERMANY

https://unikraft.io