Rust support in seL4 userspace

Nick Spinale <nick@nickspinale.com>
seL4 Summit
September 20th, 2023

Colias Group

Rust

Enforces memory safety using compile-time analysis, without the overhead of a heavyweight language runtime

Rust

Enforces memory safety, without the overhead of a heavyweight language runtime, using compile-time analysis

```rust
fn main() {
    let r;

    {
        let x = 5;
        r = &x;
    }

    println!("r: {}", r);
}
```

```
error[E0597]: `x` does not live long enough
 --> src/main.rs:6:13
  |
6 |         r = &x;
  |             ^^ borrowed value does not live long enough
7 |     }
  |     - `x` dropped here while still borrowed
8 |
9 |     println!("r: {}", r);
  |                       - borrow later used here
```

Colias
Group

Rust

Enforces memory safety, without the overhead of a heavyweight language runtime, using compile-time analysis

Aims to provide “zero cost abstractions”

Rust

Enforces memory safety, without the overhead of a heavyweight language runtime, using compile-time analysis

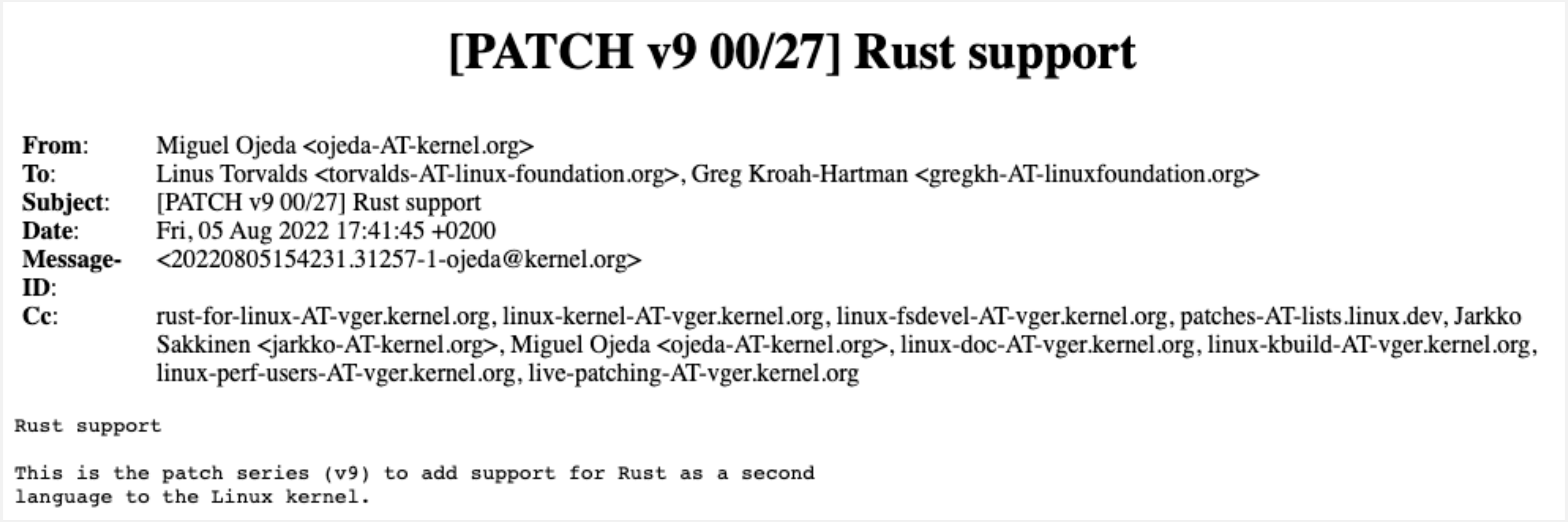
Aims to provide “zero cost abstractions”

Suitable for use cases from embedded to server, from OS kernel to application

Colias Group

Rust

Linux now supports Rust in the kernel (merged October 3rd, 2022)[1,2]



[PATCH v9 00/27] Rust support

From:	Miguel Ojeda <ojeda-AT-kernel.org>
To:	Linus Torvalds <torvalds-AT-linux-foundation.org>, Greg Kroah-Hartman <gregkh-AT-linuxfoundation.org>
Subject:	[PATCH v9 00/27] Rust support
Date:	Fri, 05 Aug 2022 17:41:45 +0200
Message-ID:	<20220805154231.31257-1-ojeda@kernel.org>
Cc:	rust-for-linux-AT-vger.kernel.org, linux-kernel-AT-vger.kernel.org, linux-fsdevel-AT-vger.kernel.org, patches-AT-lists.linux.dev, Jarkko Sakkinen <jarkko-AT-kernel.org>, Miguel Ojeda <ojeda-AT-kernel.org>, linux-doc-AT-vger.kernel.org, linux-kbuild-AT-vger.kernel.org, linux-perf-users-AT-vger.kernel.org, live-patching-AT-vger.kernel.org

```
Rust support

This is the patch series (v9) to add support for Rust as a second
language to the Linux kernel.
```

[1]https://lwn.net/ml/linux-kernel/20220805154231.31257-1-ojeda@kernel.org/
[2]https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/commit/?id=8aebac82933ff1a7c8eede18cab11e1115e2062b

Colias Group

Rust is a good fit for seL4 userspace

A high level language…

- Memory safety
- Abstraction
- Developer productivity

Rust is a good fit for seL4 userspace

A high level language…

- Memory safety
- Abstraction
- Developer productivity

…even for components without access to OS services

Colias Group

Rust is a good fit for seL4 userspace

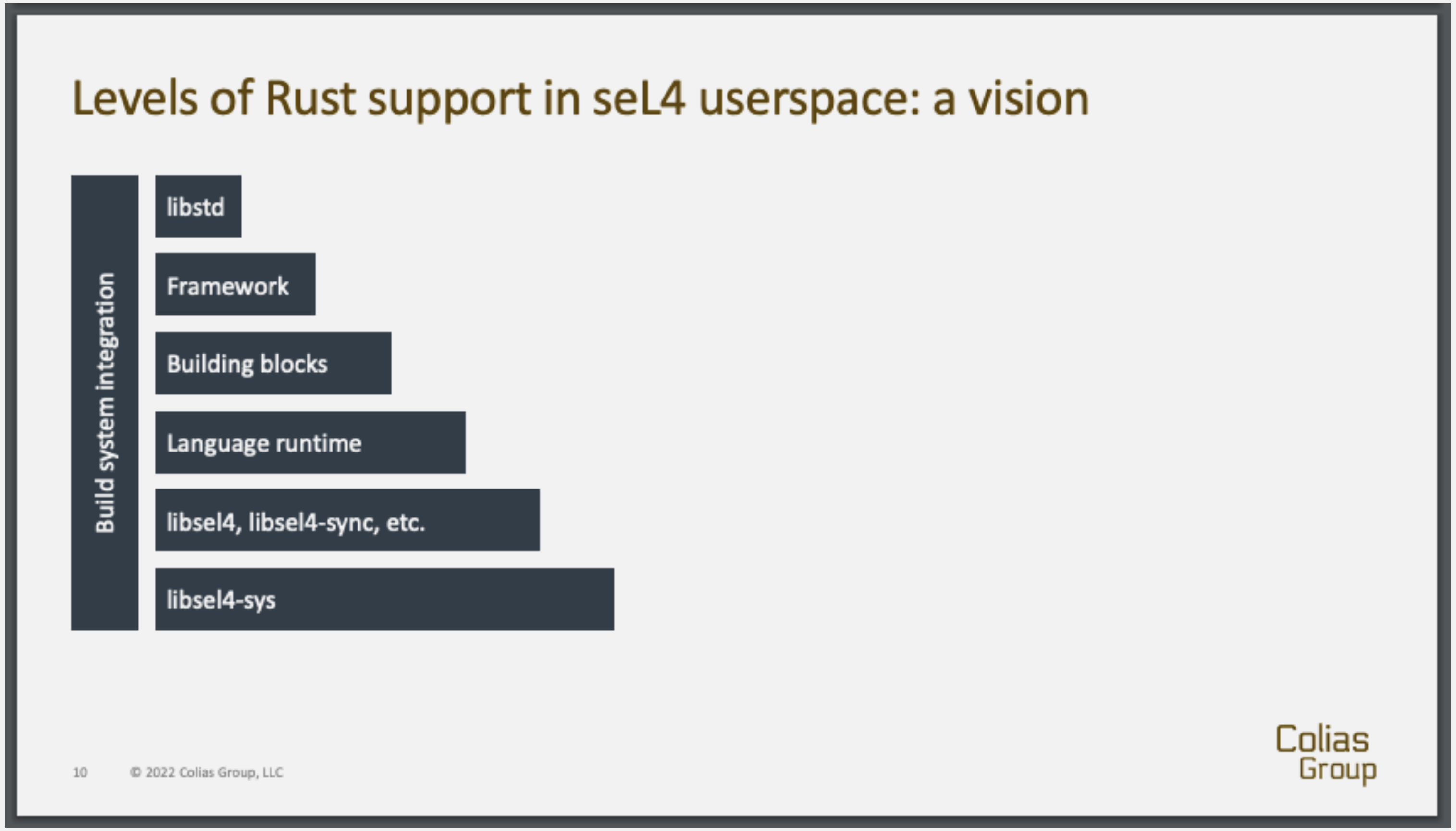
A high level language…

- Memory safety
- Abstraction
- Developer productivity

…even for components without access to OS services

…even for resource-constrained systems

Last year



Levels of Rust support in seL4 userspace: a vision

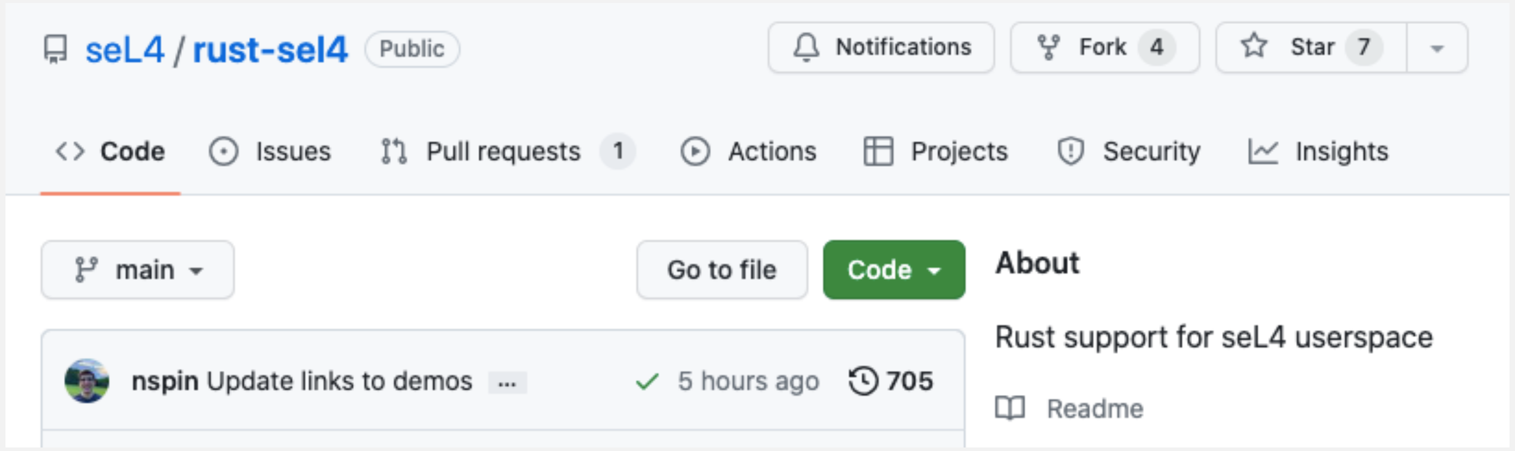
10 © 2022 Colias Group, LLC

Colias Group

Colias Group

This year

https://github.com/seL4/rust-sel4 (since last week)



nspin Update links to demos ... ✓ 5 hours ago 705

Rust support for seL4 userspace

Readme

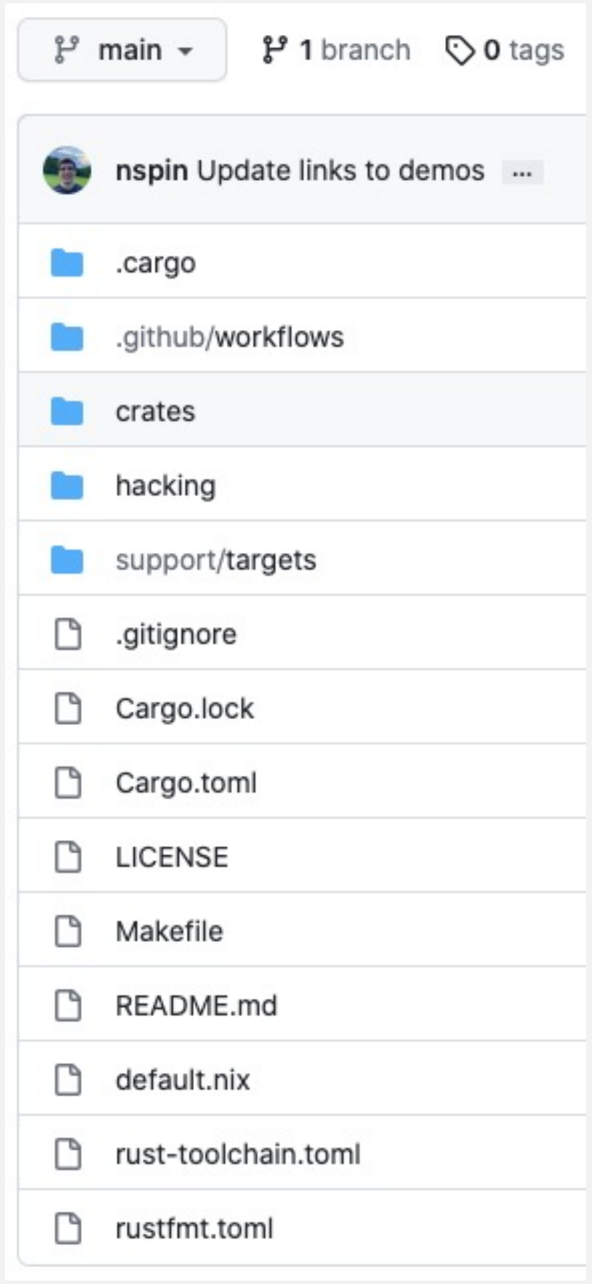
Thanks to funding by the seL4 Foundation

Colias Group

Repository contents

https://github.com/seL4/rust-sel4

- Rust libraries (aka “crates”)
  - Rust bindings for the seL4 API
  - A runtime for root tasks
  - A runtime for seL4 Microkit protection domains
  - …and many more
- CapDL-based system initializer
- General-purpose kernel loader
- Rustc target specs
- Examples
- Tests



main · 1 branch · 0 tags

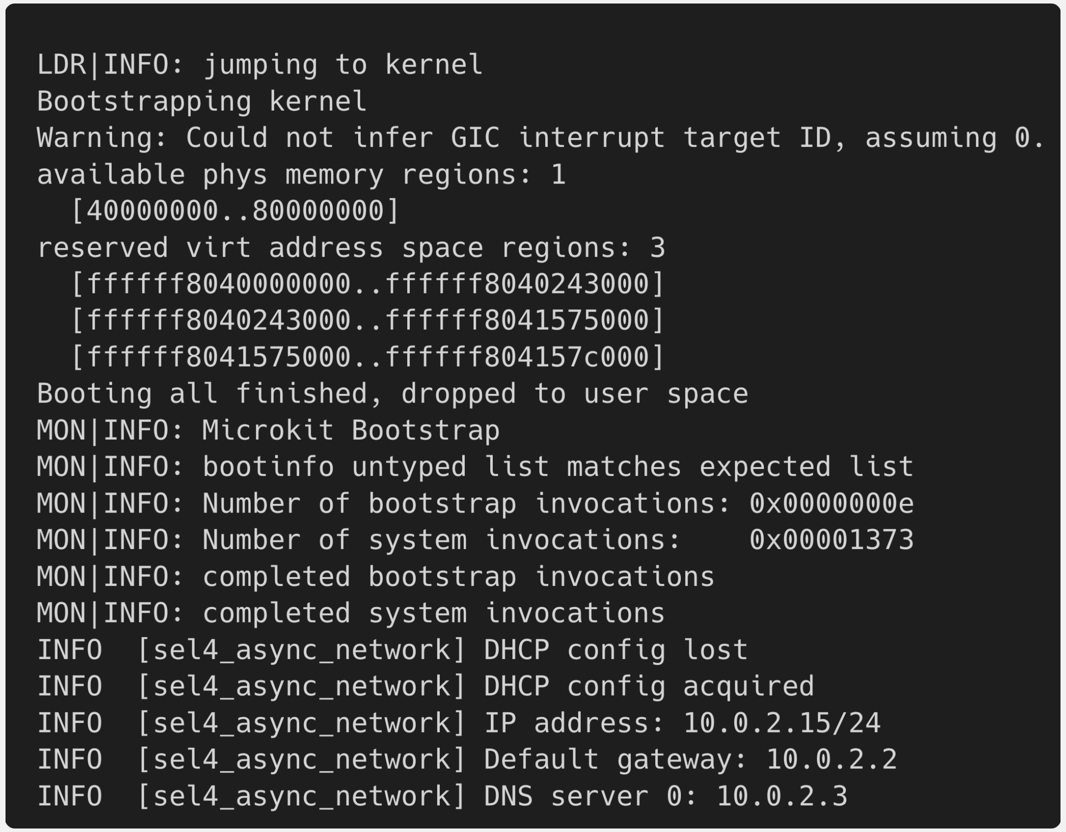
nspin Update links to demos …

.cargo
.github/workflows
crates
hacking
support/targets
.gitignore
Cargo.lock
Cargo.toml
LICENSE
Makefile
README.md
default.nix
rust-toolchain.toml
rustfmt.toml

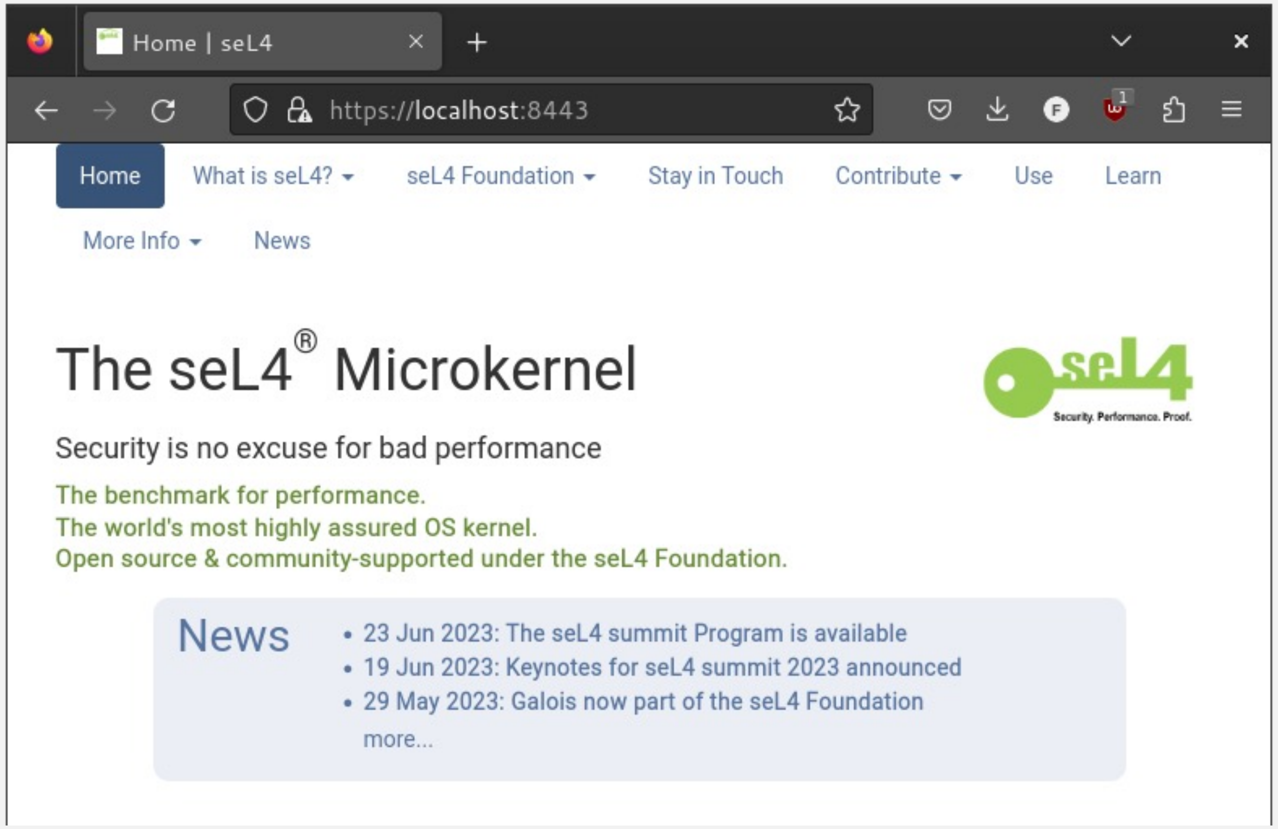
Colias Group

Example: HTTP server using seL4 Microkit

https://github.com/seL4/rust-microkit-http-server-demo



```
LDR|INFO: jumping to kernel
Bootstrapping kernel
Warning: Could not infer GIC interrupt target ID, assuming 0.
available phys memory regions: 1
  [40000000..80000000]
reserved virt address space regions: 3
  [ffffff8040000000..ffffff8040243000]
  [ffffff8040243000..ffffff8041575000]
  [ffffff8041575000..ffffff804157c000]
Booting all finished, dropped to user space
MON|INFO: Microkit Bootstrap
MON|INFO: bootinfo untyped list matches expected list
MON|INFO: Number of bootstrap invocations: 0x0000000e
MON|INFO: Number of system invocations:    0x00001373
MON|INFO: completed bootstrap invocations
MON|INFO: completed system invocations
INFO  [sel4_async_network] DHCP config lost
INFO  [sel4_async_network] DHCP config acquired
INFO  [sel4_async_network] IP address: 10.0.2.15/24
INFO  [sel4_async_network] Default gateway: 10.0.2.2
INFO  [sel4_async_network] DNS server 0: 10.0.2.3
```



The seL4® Microkernel

Security is no excuse for bad performance

The benchmark for performance.
The world's most highly assured OS kernel.
Open source & community-supported under the seL4 Foundation.

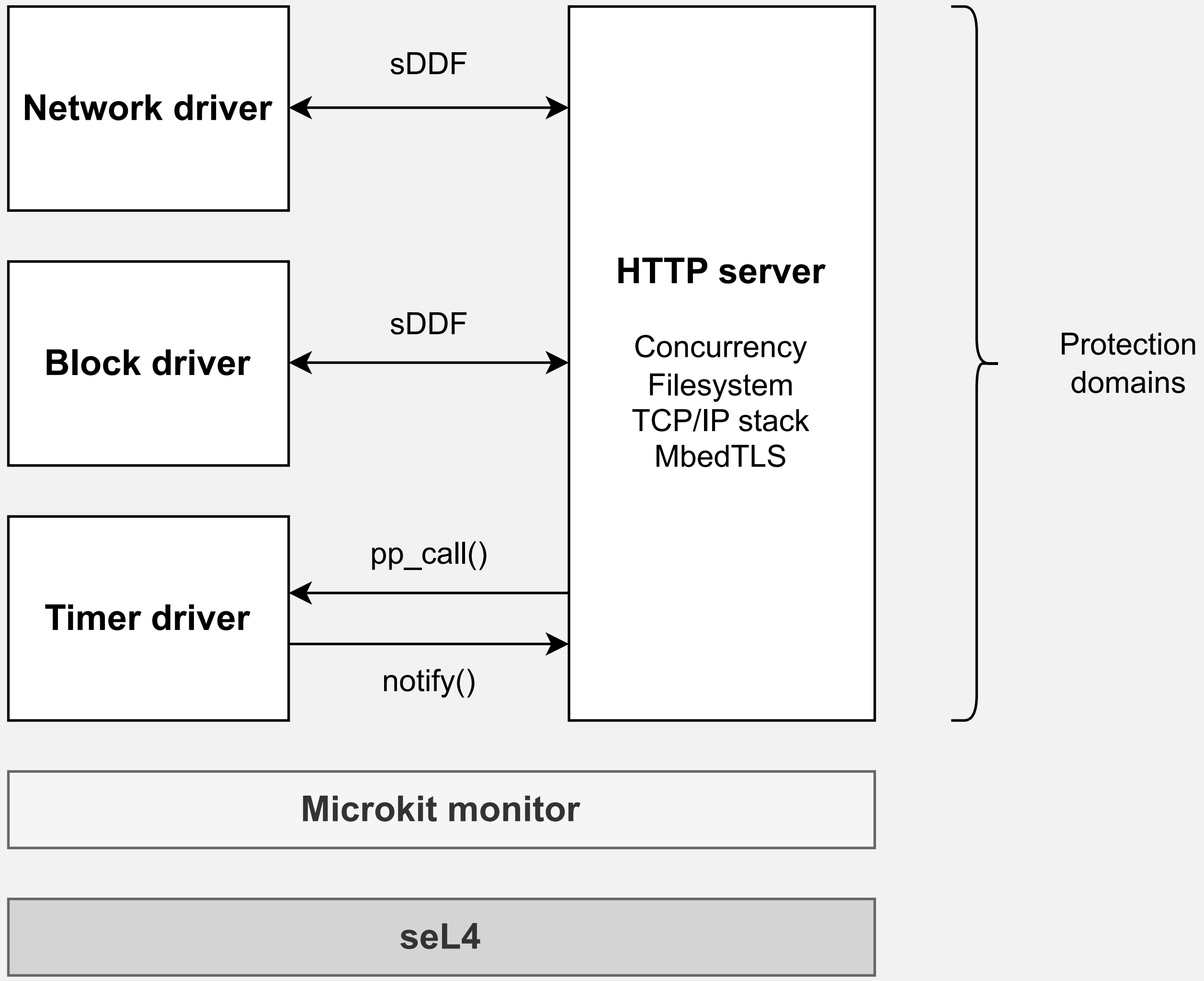
News
- 23 Jun 2023: The seL4 summit Program is available
- 19 Jun 2023: Keynotes for seL4 summit 2023 announced
- 29 May 2023: Galois now part of the seL4 Foundation

more...

Colias Group

Example: HTTP server using seL4 Microkit

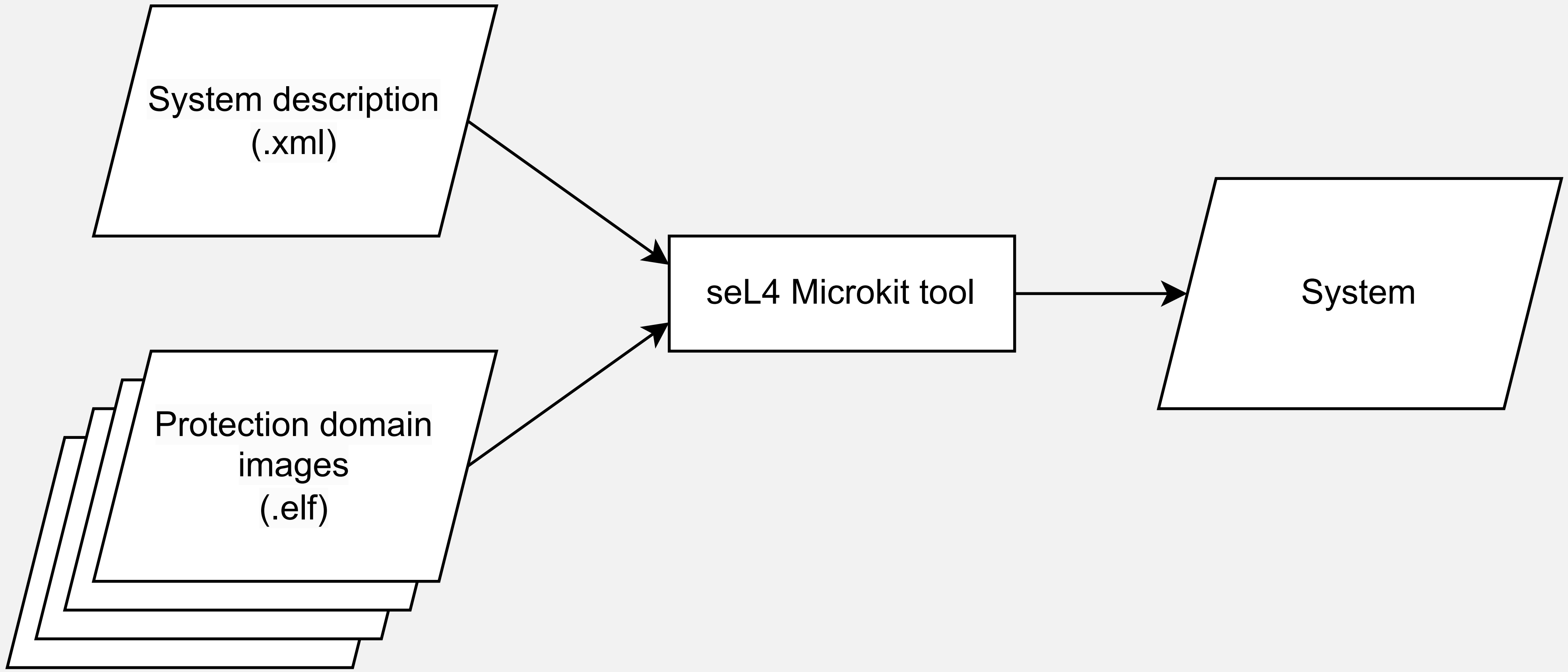
https://github.com/seL4/rust-microkit-http-server-demo



Colias Group

Example: HTTP server using seL4 Microkit

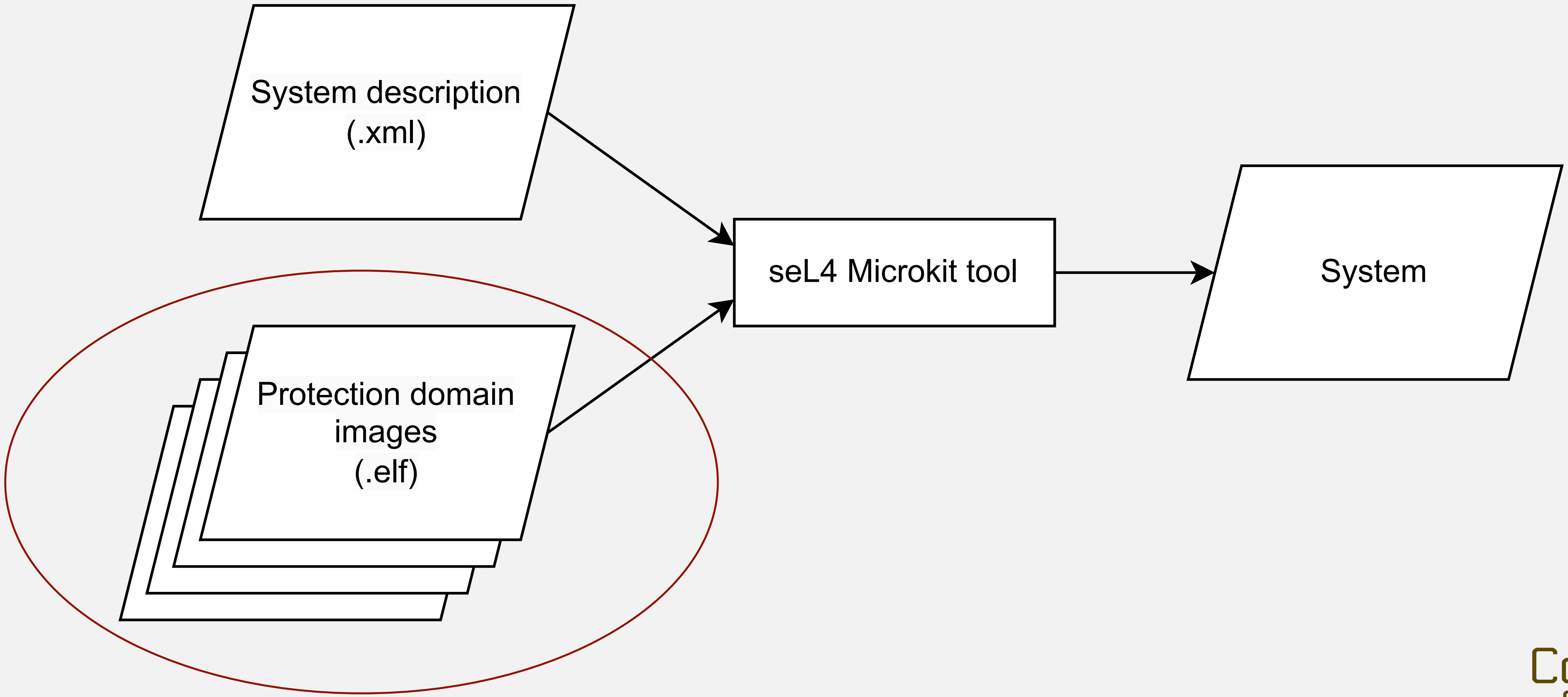
https://github.com/seL4/rust-microkit-http-server-demo



Colias Group

Example: HTTP server using seL4 Microkit

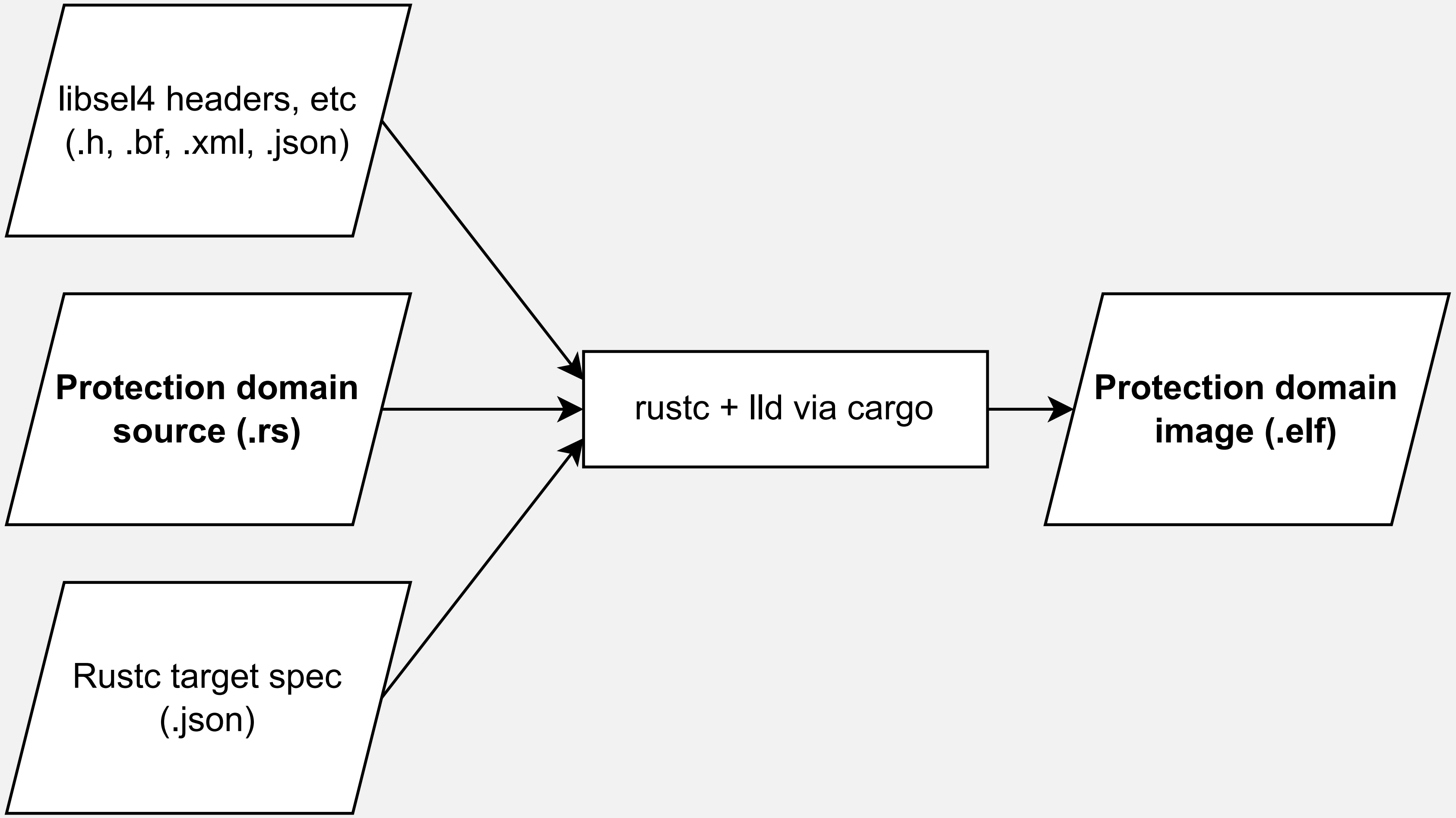
https://github.com/seL4/rust-microkit-http-server-demo



Colias Group

Example: HTTP server using seL4 Microkit

https://github.com/seL4/rust-microkit-http-server-demo


libsel4 headers, etc (.h, .bf, .xml, .json)
Protection domain source (.rs)
Rustc target spec (.json)
rustc + lld via cargo
Protection domain image (.elf)

Colias Group

Example: HTTP server using seL4 Microkit

https://github.com/seL4/rust-microkit-http-server-demo

```
RUST_TARGET_PATH=$(abspath $(rust_target_dir) \
SEL4_INCLUDE_DIRS=$(abspath $(microkit_sdk_dir)/include) \
   cargo build \
      -Z build-std=core,alloc,compiler_builtins \
      -Z build-std-features=compiler-builtins-mem \
      --target aarch64-sel4
```

Colias Group

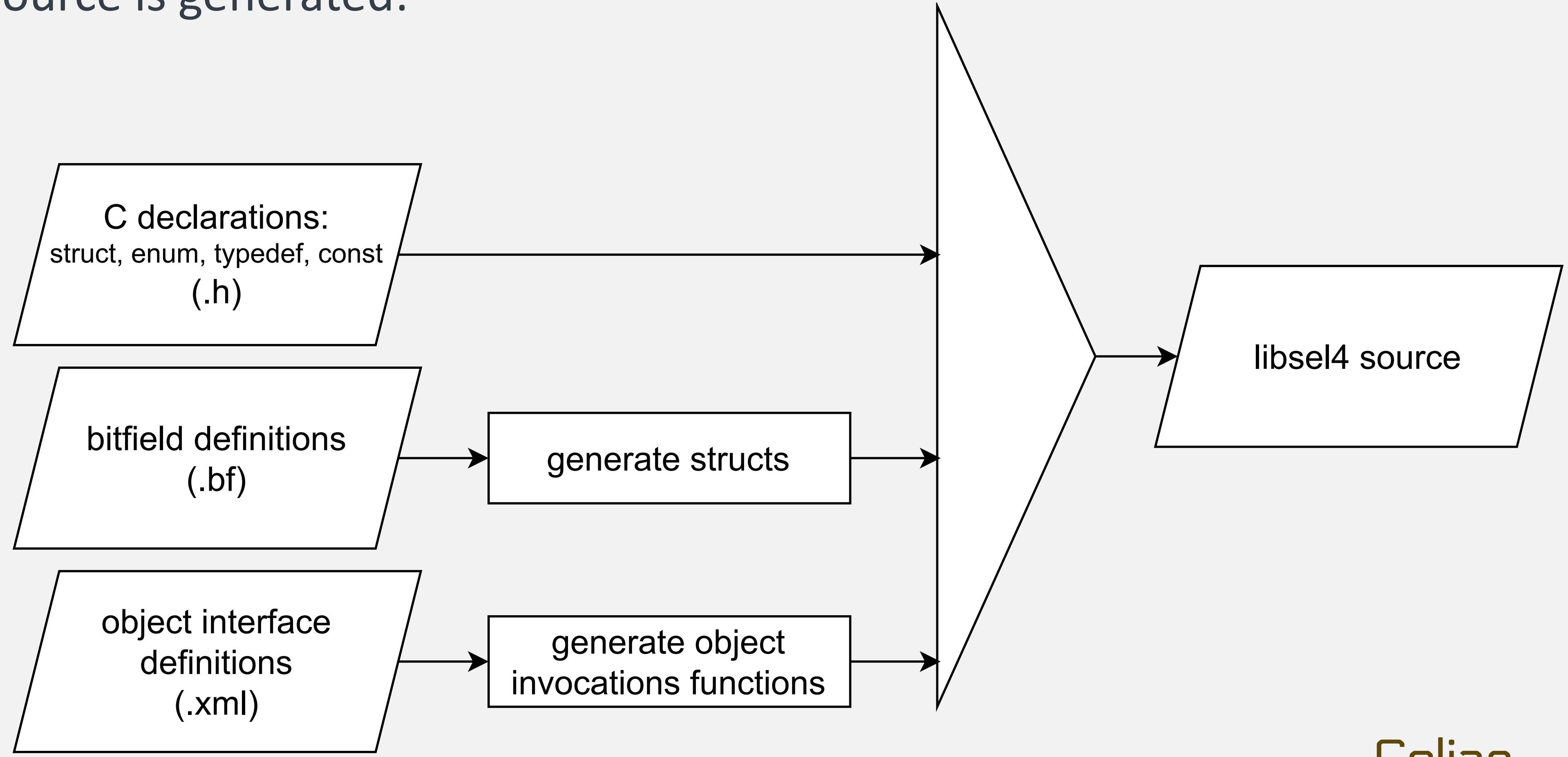
Crate: `sel4-sys`

Low-level libsel4

Crate: `sel4-sys`

Low-level libsel4

How the C libsel4 source is generated:

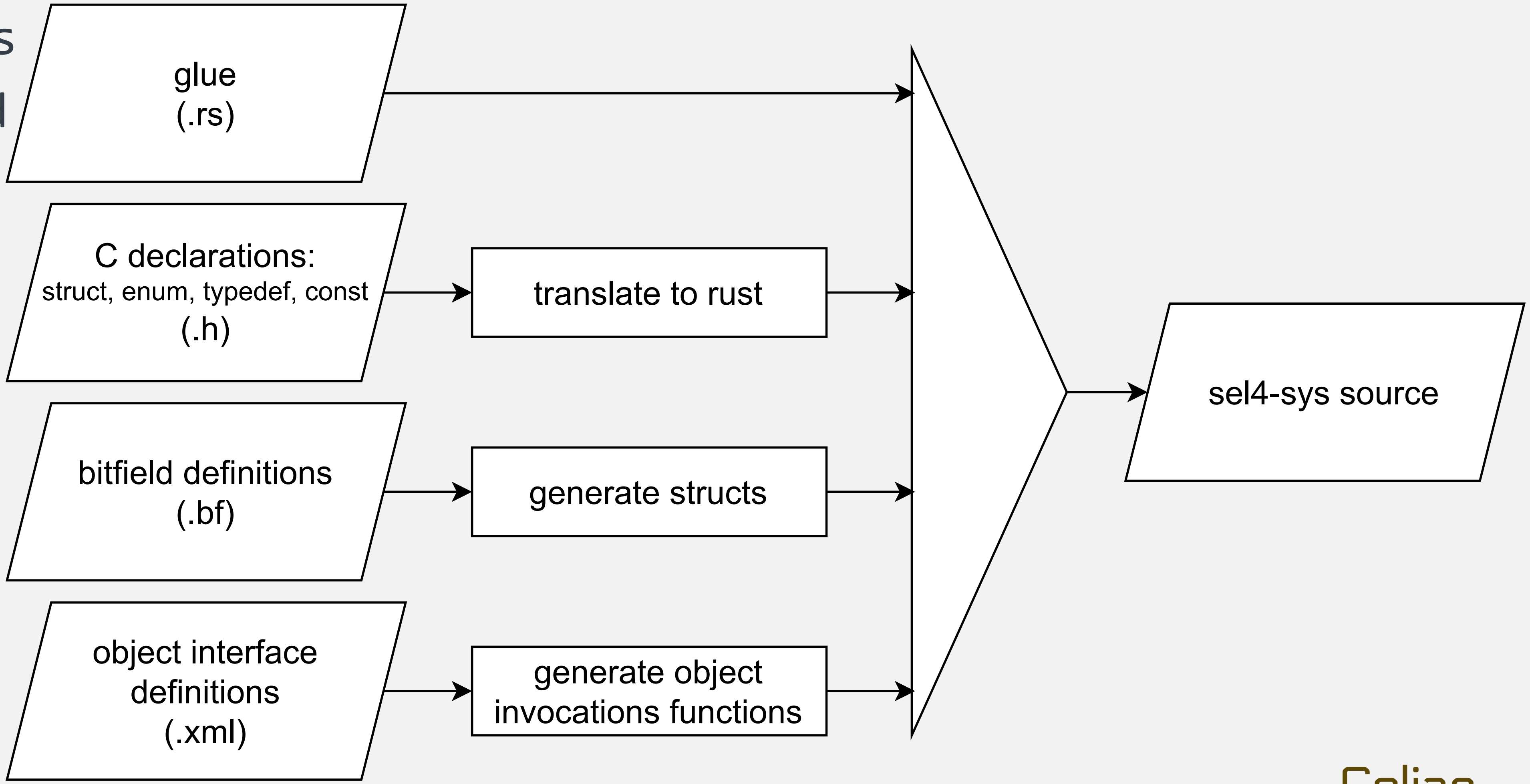


Colias
Group

Crate: `sel4-sys`

Low-level libsel4

How the `sel4-sys` source is generated



Colias Group

Crate: `sel4-sys`

Low-level libsel4

- Pure Rust
- The only build-time dependency outside of Rust and libsel4 headers is libclang
- Simple to build:
  - Supply libsel4 headers (including .bf and .xml) via `$SEL4_INCLUDE_DIRS`
  - Crate `build.rs` takes care of code generation
- TLS is optional
- Testing: masquerade as `libsel4.a` and link against `sel4test`

Crate: `sel4`

Higher-level libsel4

The “real” Rust libsel4: wraps `sel4-sys`, leveraging the Rust type system and idioms to present a cleaner and more ergonomic API

- No additional dependencies
- TLS is still optional

Rustdoc:

https://sel4.github.io/rust-sel4/views/aarch64-root-task/aarch64-sel4/doc/sel4/index.html

Colias Group

Crate: `sel4`

Higher-level libsel4

```
// implicit (TLS, or global in single-threaded case)

untyped_cap.untyped_retype(
    &blueprint,
    &cnode,
    slot,
    1,
);

// explicit

untyped_cap.with(&mut ipc_buffer).untyped_retype(
    &blueprint,
    &cnode,
    slot,
    1,
);
```

Colias Group

The Rust Standard Library

Layer	**Provides**	**Requires**

`libstd`	`std::fs`
	`std::net`
	`std::thread`
	`std::process`
	Language runtime
		OS services

`liballoc`	`alloc::vec`
	`alloc::collections`
	`alloc::string`
		heap allocator

`libcore`	`core::mem`
	`core::num`
	`core::iter`
	`core::ffi`
		nothing!
		(except panic handler)

depends on ↓

Colias Group

The Rust Standard Library

	Layer	**Provides**	**Requires**

#![no_std]	~~`libstd`~~	`std::fs`
		`std::net`
		`std::thread`
		`std::process`
		Language runtime
			OS services

	`liballoc`	`alloc::vec`
		`alloc::collections`
		`alloc::string`
			heap allocator

	`libcore`	`core::mem`
		`core::num`
		`core::iter`
		`core::ffi`
			nothing!
			(except panic handler)

depends on ↓

Colias Group

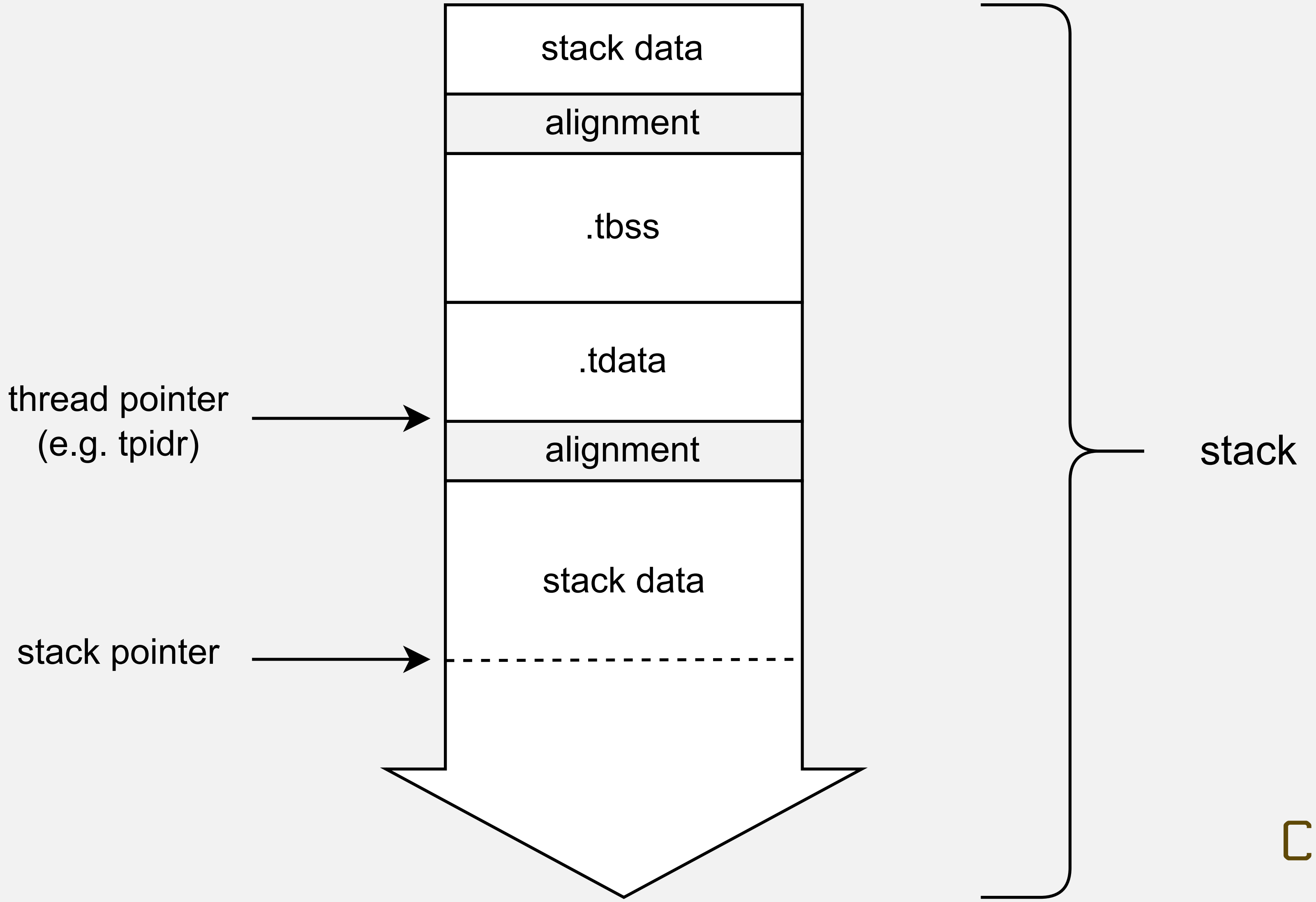
Language runtime

- Entrypoint: `_start` *(required)*
- Stack *(required)*
- Thread local storage *(optional)*
- Heap allocator: `#[global_allocator]` *(optional)*
- Panic handler: : `#[panic_handler]` *(required)*
- Exception handling *(optional)*

Colias Group

Crate: `sel4-reserve-tls-on-stack`

Internal language runtime building block



Colias Group

Crate: `sel4-panicking`

Internal language runtime building block

Configurable exception handling (±TLS, ±heap)

Heavy lifting done by external dependency: `unwinding` crate

```
use sel4_panicking::catch_unwind;

let result = catch_unwind(|| {
    debug_println!("hello!");
});
assert!(result.is_ok());

let result = catch_unwind(|| {
    panic!("oh no!");
});
assert!(result.is_err())
```

Colias Group

Crate: `sel4-backtrace`

Internal language runtime building block

Flexible backtrace collection for debugging

Crate: `sel4-backtrace`

Internal language runtime building block

Defer symbolization:

```
Bootstrapping kernel
available phys memory regions: 1
  [60000000..80000000]
reserved virt address space regions: 3
  [8060000000..8060246000]
  [807e157000..807e1590a6]
  [807e15a000..8080000000]
Booting all finished, dropped to user space
collecting stack backtrace
sending stack backtrace
    0001b09a910101f0f88a0101d4d6900101a09f8b0101bce5900101a0e8900101e8e7900101c0f1900101a8ee90010180c0a9100000
```

Colias Group

Crate: `sel4-backtrace`

Internal language runtime building block

Defer symbolization:

```
$ cargo run -p sel4-backtrace-cli -- -f ./result/root-task.elf
0001b09a910101f0f88a0101d4d6900101a09f8b0101bce5900101a0e8900101e8e7900101c0f1900101a8ee90010180c0a9100000
    Finished dev [unoptimized + debuginfo] target(s) in 0.15s
     Running `target/debug/sel4-symbolize-backtrace -f ./result/root-task.elf
0001b09a910101f0f88a0101d4d6900101a09f8b0101bce5900101a0e8900101e8e7900101c0f1900101a8ee90010180c0a9100000
backtrace: ./result/root-task.elf
    0:              0x244d30 - sel4_backtrace::collect
                               sel4_backtrace::BacktraceSendWithToken::collect
                               sel4_backtrace_simple::SimpleBacktracing::collect
                                 at /nix/store/chv0yzjhbrih0ghvfr420qi2bph9d7pw-workspace/src/sel4-backtrace/
    1:              0x22bc70 - tests_root_task_backtrace::g
                               core::ops::function::FnMut::call_mut
                               <core::slice::iter::Iter<T> as core::iter::traits::iterator::Iterator>::for_ea
                               tests_root_task_backtrace::f
                               tests_root_task_backtrace::main::{{closure}}
                               sel4_panicking::catch_unwind::do_call
                               sel4_panicking::catch_unwind
                                 at /nix/store/chv0yzjhbrih0ghvfr420qi2bph9d7pw-workspace/src/tests-root-task
    2:              0x242b54 - tests_root_task_backtrace::main
```

Crate: `sel4-backtrace`

Internal language runtime building block

Symbolize on-device:

```
Bootstrapping kernel
available phys memory regions: 1
  [60000000..80000000]
reserved virt address space regions: 3
  [8060000000..8060246000]
  [807e184000..807e1860a6]
  [807e187000..8080000000]
Booting all finished, dropped to user space
printing backtrace:
    0:            0x22b7cc - sel4_backtrace::collect_with
                             sel4_backtrace::collect
                             tests_root_task_backtrace::g
                             core::ops::function::FnMut::call_mut
                             <core::slice::iter::Iter<T> as core::iter::traits::iterator::Iterator>::for_
                             tests_root_task_backtrace::f
                             tests_root_task_backtrace::main::{{closure}}
                             sel4_panicking::catch_unwind::do_call
                             sel4_panicking::catch_unwind
```

Crate: `sel4-root-task`

Language runtime #1

Configurable (±TLS, ±heap, ±unwinding)

Glues together:

- `sel4`
- `sel4-reserve-tls-on-stack`
- `sel4-panicking`
- `sel4-dlmalloc`
- …and more

Colias Group

Crate: `sel4-root-task`

Language runtime #1

```
#![no_std]
#![no_main]
#![feature(never_type)]

use sel4_root_task::root_task;

#[root_task]
fn main(_bootinfo: &sel4::BootInfo) -> ! {
    sel4::debug_println!("Hello, World!");

    sel4::BootInfo::init_thread_tcb().tcb_suspend().unwrap();

    unreachable!()
}
```

Colias Group

Crate: `sel4-microkit`

Language runtime #2

Configurable (±TLS, ±heap, ±unwinding)

Colias Group

Crate: sel4-microkit

Language runtime #2

```rust
#![no_std]
#![no_main]

use sel4_microkit::{debug_println, protection_domain, Channel, Handler, MessageInfo};

#[protection_domain(stack_size = 4096 * 4, heap_size = 4096 * 12)]
fn init() -> HandlerImpl {
    debug_println!("Hello, World!");
    HandlerImpl {}
}

struct HandlerImpl {}

impl Handler for HandlerImpl {
    fn notified(&mut self, channel: Channel) -> Result<(), Self::Error> {
        todo!()
    }

    fn protected(
        &mut self,
        channel: Channel,
        msg_info: MessageInfo,
    ) -> Result<MessageInfo, Self::Error> {
        todo!()
    }
}
```

Higher-level crates

- `sel4-logging`
- `sel4-sync`
- `sel4-externally-shared`
- `sel4-shared-ring-buffer`
- `sel4-microkit-message`
- `sel4-async-*`

Colias Group

Asynchronous programming in Rust

Concurrent programming model for lightweight green threads at the library level

Asynchronous programming in Rust

Futures:

```rust
pub trait Future {
    type Output;

    fn poll(self: Pin<&mut Self>, cx: &mut Context<'_>) -> Poll<Self::Output>;
}

pub enum Poll<T> {
    Ready(T),
    Pending,
}
```

Colias Group

Asynchronous programming in Rust

Composing futures

```
fn recv_request() -> impl Future<Request>;

fn send_response(req: &Request) -> impl Future<()>;

fn serve() -> impl Future<()> {
    recv_request.then(|req| {
        send_response(&req)
    })
}
```

Colias Group

Asynchronous programming in Rust

Composing futures

```
fn recv_request() -> impl Future<Request>;

fn send_response(req: &Request) -> impl Future<()>;

fn serve() -> impl Future<()> {
    recv_request.then(|req| {
        send_response(&req)
    })
}
```

async/await

```
async fn recv_request() -> Request;

async fn send_response(req: &Request);

async fn serve() {
    let req = recv_request().await;
    send_response(req)
}
```

Colias Group

Asynchronous programming in Rust on seL4

```rust
async fn send_response_header<U: AsyncIo>(
    &self,
    conn: &mut U,
    name: &str,
    value: &[u8],
) -> Result<(), U::Error> {
    conn.send_all(name.as_bytes()).await?;
    conn.send_all(b": ").await?;
    conn.send_all(value).await?;
    conn.send_all(b"\r\n").await?;
    Ok(())
}
```

Asynchronous programming in Rust on seL4

```
for f in [use_socket_for_http, use_socket_for_https].map(Rc::new) {
    for _ in 0..MAX_NUM_SIMULTANEOUS_CONNECTIONS {
        spawner
            .spawn_local({
                let network_ctx = network_ctx.clone();
                let f = f.clone();
                async move {
                    loop {
                        let socket = network_ctx.new_tcp_socket();
                        f(TcpSocketWrapper::new(socket)).await;
                    }
                }
            })
            .unwrap()
    }
}
```

Colias Group

Asynchronous programming in Rust on seL4

PD event handler is centered around an “executor”:

- Maintains pool of futures (i.e. green threads)
- Responds to external events by polling futures which have been woken up

Colias Group

Asynchronous programming in Rust on seL4

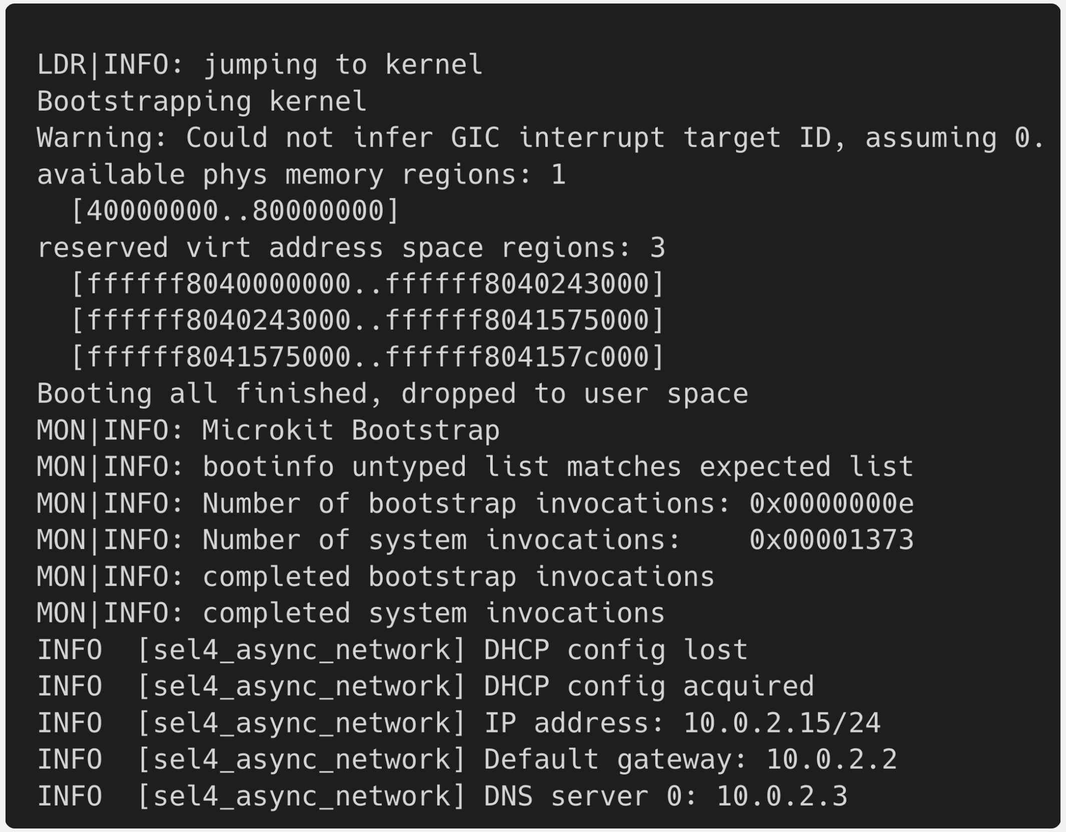
Crates:

- `sel4-async-timers`
- `sel4-async-network`
- `sel4-async-block-io`
- `sel4-async-single-threaded-executor`

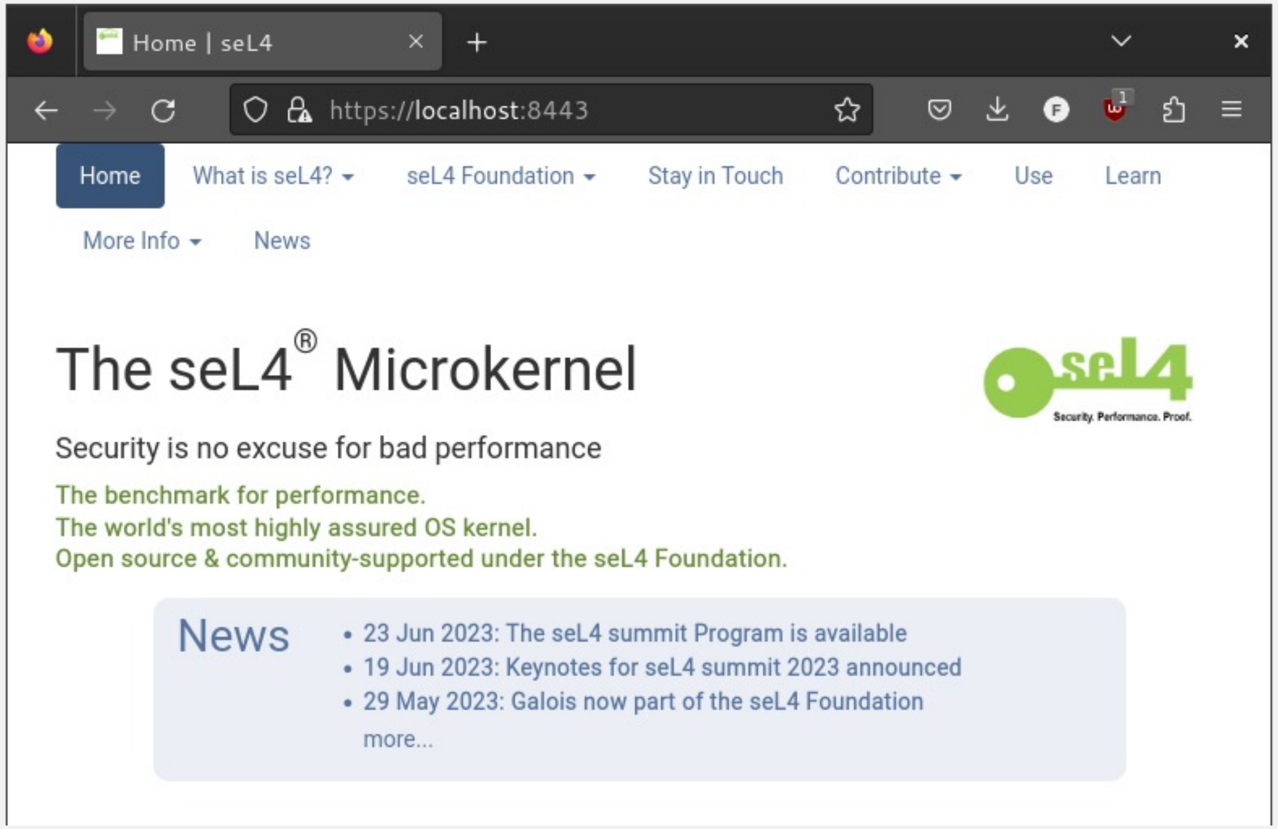
Colias Group

Example: HTTP server using seL4 Microkit

https://github.com/seL4/rust-microkit-http-server-demo



```
LDR|INFO: jumping to kernel
Bootstrapping kernel
Warning: Could not infer GIC interrupt target ID, assuming 0.
available phys memory regions: 1
  [40000000..80000000]
reserved virt address space regions: 3
  [ffffff8040000000..ffffff8040243000]
  [ffffff8040243000..ffffff8041575000]
  [ffffff8041575000..ffffff804157c000]
Booting all finished, dropped to user space
MON|INFO: Microkit Bootstrap
MON|INFO: bootinfo untyped list matches expected list
MON|INFO: Number of bootstrap invocations: 0x0000000e
MON|INFO: Number of system invocations:    0x00001373
MON|INFO: completed bootstrap invocations
MON|INFO: completed system invocations
INFO  [sel4_async_network] DHCP config lost
INFO  [sel4_async_network] DHCP config acquired
INFO  [sel4_async_network] IP address: 10.0.2.15/24
INFO  [sel4_async_network] Default gateway: 10.0.2.2
INFO  [sel4_async_network] DNS server 0: 10.0.2.3
```



The seL4® Microkernel

Security is no excuse for bad performance

The benchmark for performance.
The world's most highly assured OS kernel.
Open source & community-supported under the seL4 Foundation.

News
- 23 Jun 2023: The seL4 summit Program is available
- 19 Jun 2023: Keynotes for seL4 summit 2023 announced
- 29 May 2023: Galois now part of the seL4 Foundation

more...

Colias Group

CapDL-based system initializer

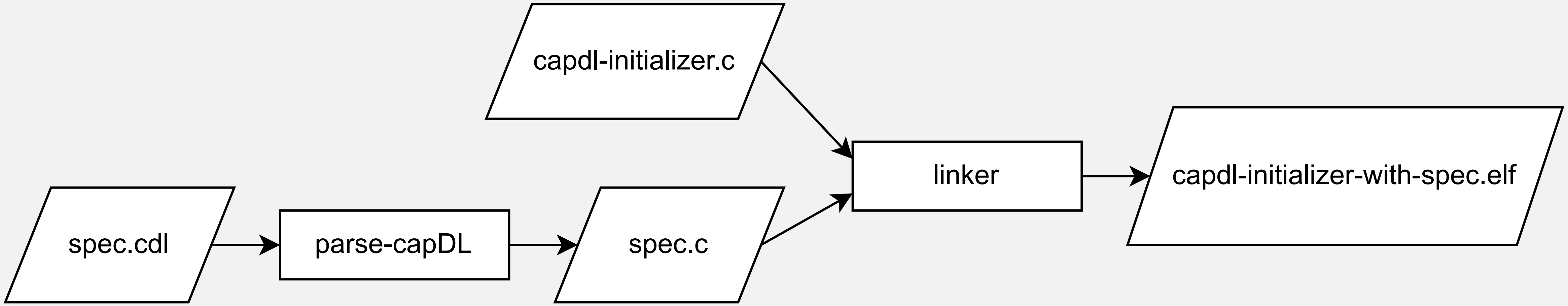
Supports zero-copy frame content: backing root-task frames themselves are mapped into child VSpaces

Supports a usage mode where initializer is built ahead of time and then later coupled with serialized CapDL spec

Suitable for Microkit, to compliment verified CASE initializer for use on unverified configurations and during development

CapDL-based system initializer

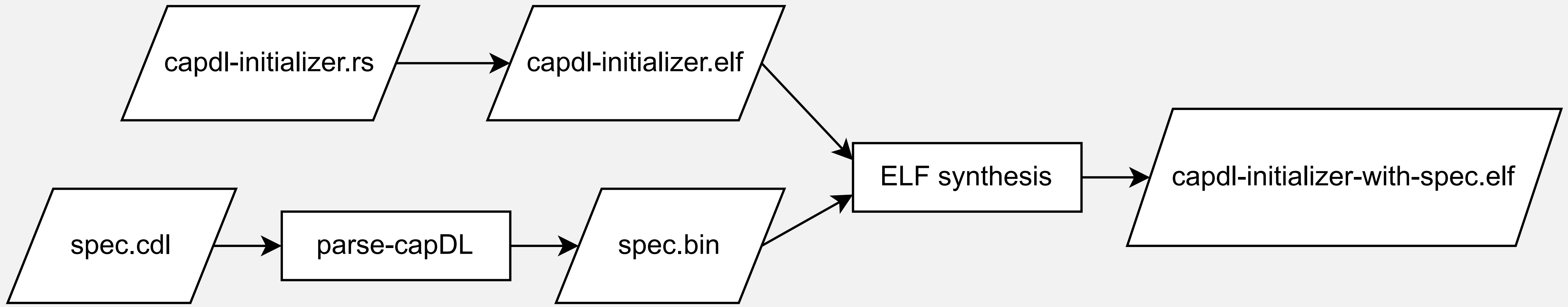
Using the C CapDL initializer



Colias Group

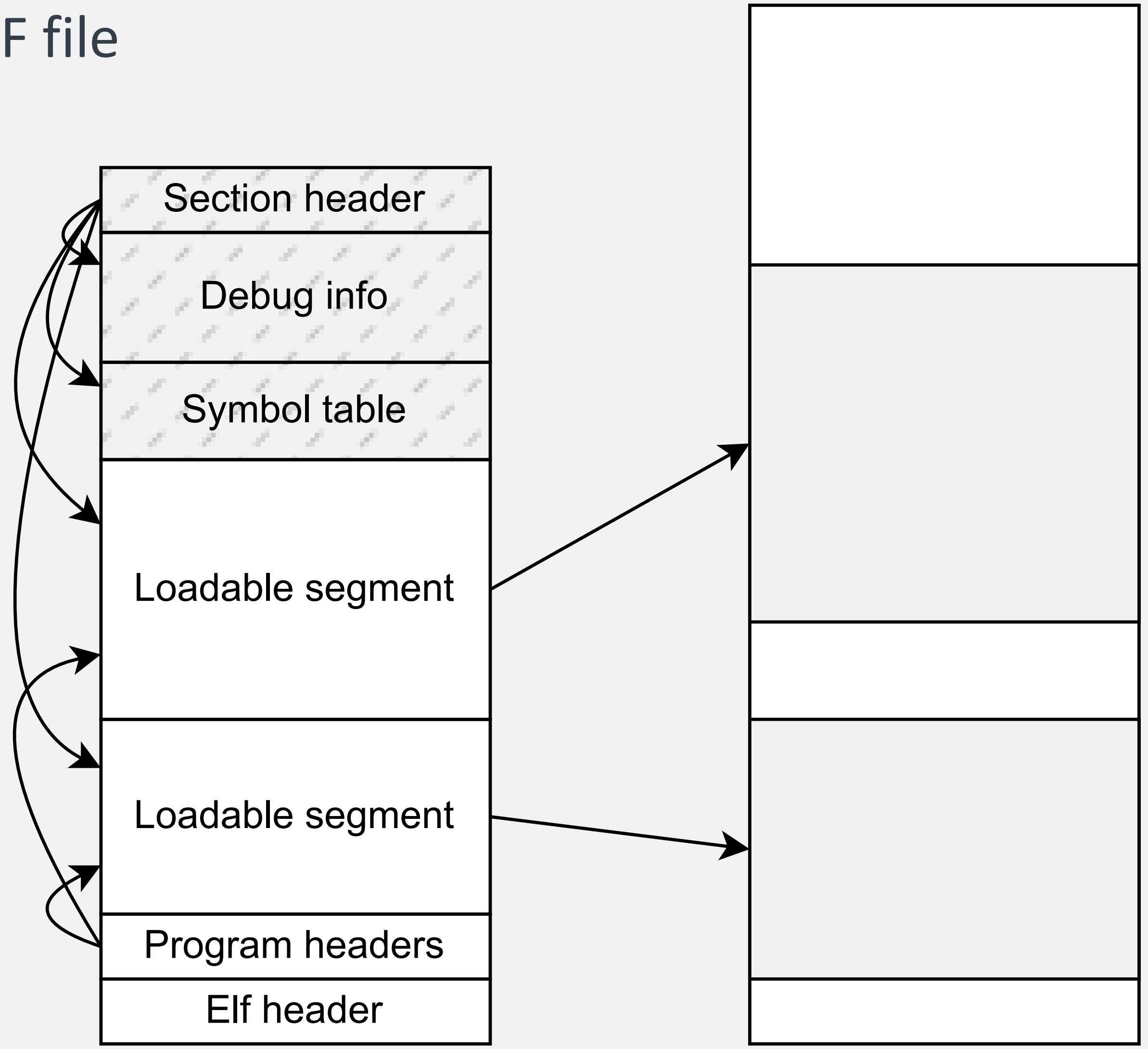
CapDL-based system initializer

Using the Rust CapDL initializer



CapDL-based system initializer

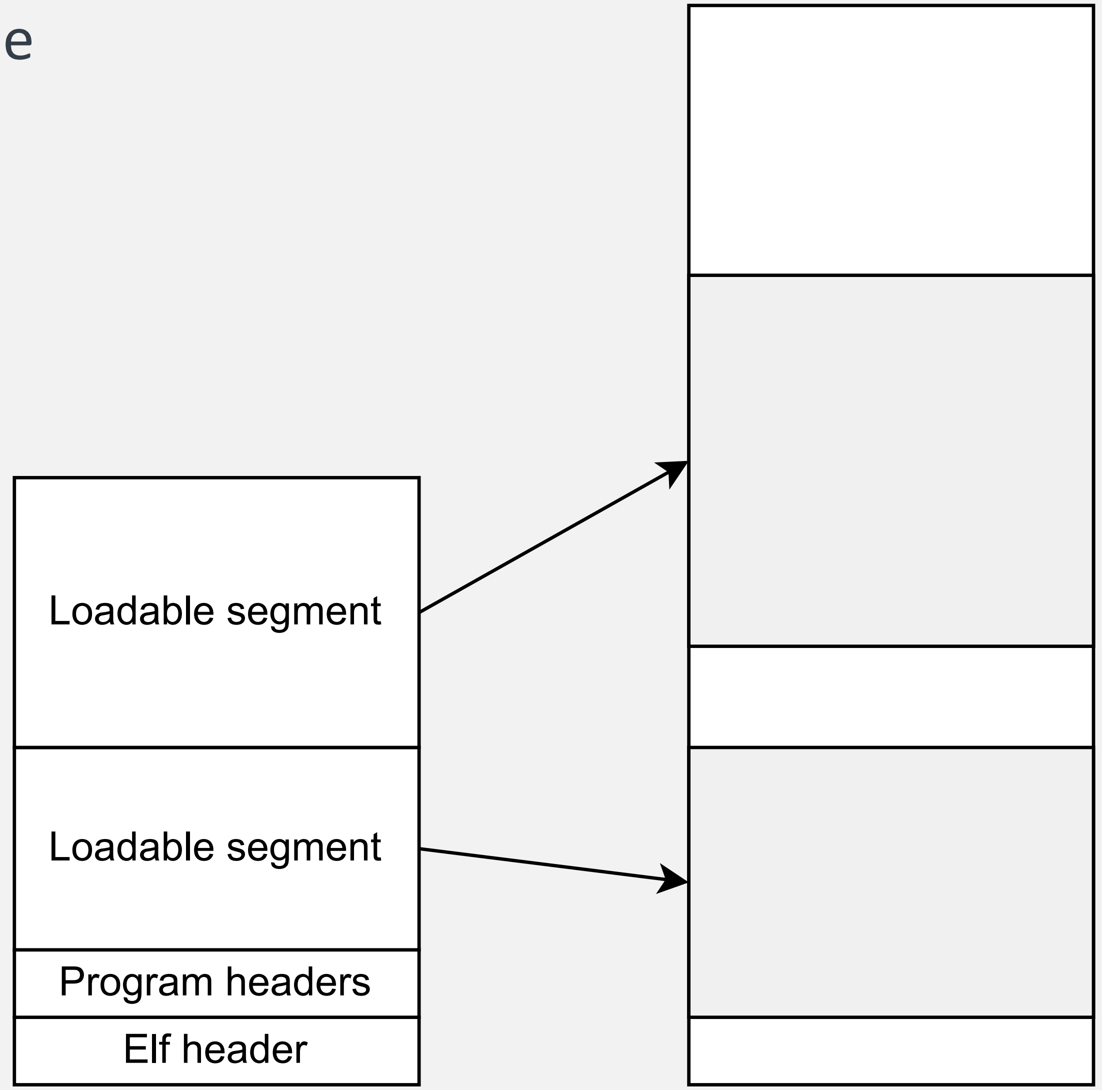
Creating a synthetic ELF file



Colias Group

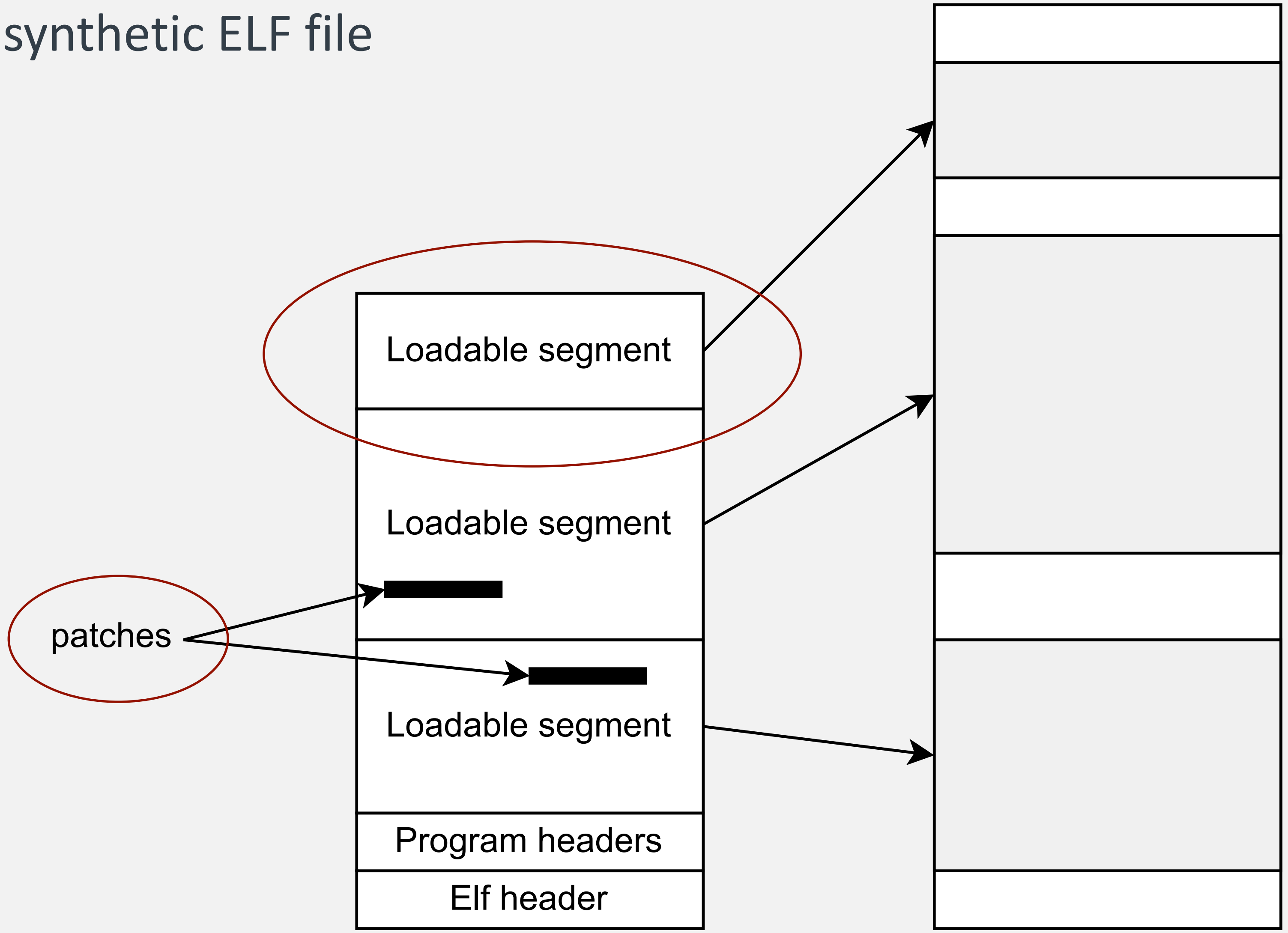
CapDL-based system initializer

Creating a synthetic ELF file



CapDL-based system initializer

Creating a synthetic ELF file



Colias Group

CapDL-based system initializer

Supports zero-copy frame content: backing root-task frames themselves are mapped into child VSpaces

Supports a usage mode where initializer is built ahead of time and then later coupled with serialized CapDL spec

Suitable for Microkit, to compliment verified CASE initializer for use on unverified configurations and during development

General-purpose kernel loader

Uses synthetic ELF technique

Suitable for binary distribution

What’s next

- Support adoption
- Support more configurations
- Benchmark
- Host on https://crates.io
- More high-level constructs
- More reusable components

https://github.com/seL4/rust-sel4

mailto:nick@nickspinale.com

Colias Group

Discussion

https://github.com/seL4/rust-sel4

Colias Group