CoreFi: An Orchestration Language for Embedded Systems

Bridging Implementation to Specification

Yan Liu Alex Potanin

Australian National University

submitted to OOPSLA '26

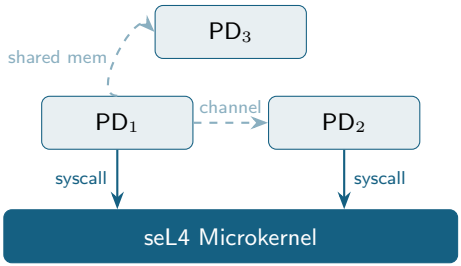
`fiducia-lang.github.io`

seL4 MicroKit

seL4: formally verified microkernel: Enforces spatial isolation via *capabilities*

MicroKit: builds on seL4 with programmable constructs

- **Protection Domains (PDs)**: isolated processes, each runs a C/Pancake program
- **Channels**: async (bounded shared-memory buffer) or sync (priority-based rendezvous)
- **Memory regions**: mapped into PDs' address spaces at boot, fixed for lifetime
- System topology is *declared statically* — no dynamic changes at runtime

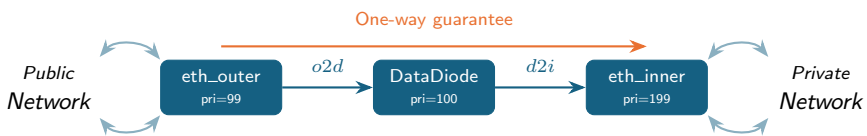


What MicroKit does NOT provide

No language-level structure around channels or memory — everything is C integers and raw pointers.

Running Example: Data Diode

A **network data diode** enforces *one-way* data flow: packets travel from the public (untrusted) network to the private (trusted) network — no reverse channel, ever.



DEMO

Show `datadiode.system` and `ddiode.c` · `make build && make run`

What We Saw: The Verification Gap

Channels are bare integers

```
#define OUTER_OUTPUT_CH   1
#define INNER_INPUT_CH    2
```

No direction. No type. Nothing stops a PD from sending on a channel it should only receive.

Memory is raw pointers

```
uintptr_t outer_output_vaddr;
uintptr_t inner_input_vaddr;
```

Any PD the system grants a mapping to can read *and* write. Accidental aliasing is one mapping declaration away.

Write anything to any memory region

```
struct buffer_descriptor {
  uint16_t data_length; // attacker-controlled
  uint16_t flags;       // 0=empty, 1=full (convention only)
};

case INNER_INPUT_CH:
  /* nothing prevents uncommenting: */
  volatile struct buffer_descriptor *obd =
    (void*)(INNER_INPUT
            + BUFFER_SIZE * inner_input_index);
  if (obd->flags == 1) { /* silent drop */ }
  else {
    obd->data_length = bd->data_length; // attacker-controlled
    mycpy(opkt, pkt, bd->data_length);  // no bounds check
    obd->flags = 1; // convention only, not enforced
    microkit_notify(OUTER_OUTPUT_CH);   // reverse channel!
  }
  break;

void mycpy(volatile void *dst, volatile void *src,
           unsigned int length) {
  volatile uint64_t *d = dst; // raw pointer
  int l = length / 64;
  while (l) { d[i]=s[i]; ... d[i+7]=s[i+7];
              l--; i += 8; }
}
```

Why Verification — Especially Automated — Is Hard

Top-down: choreographic programming

- Write a global choreography, project to local code
- Deadlock freedom by construction
- **Assumes:** clean separation of computation and communication; no shared state between participants

⇒ Does not fit MicroKit: PDs share memory *and* communicate

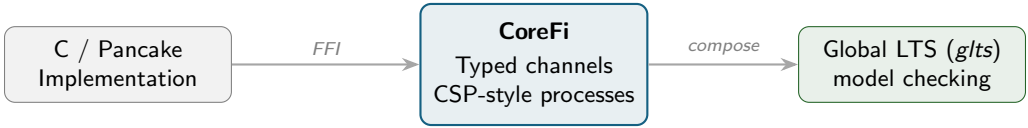
Bottom-up: model extraction

- Manually extract a formal model from C
- Verify the model with UPPAAL / SPIN / FDR
- **Problems:** expert-intensive, fragile (model rots with every code change), state-space explosion from unconstrained shared memory

What we need

Structure at the **language level**: typed channels with direction, typed memory access, process control flow that can be *automatically* lifted into a formal model.

CoreFi: A Middle-Layer Language



What CoreFi adds to MicroKit:

- Channels carry *direction* and *type* — no bare integers
- Memory mappings are *access-controlled* — no accidental aliasing
- Process control flow is *explicit* — CSP-style events, guards, recursion
- C implementations linked via FFI *without change*

`datadiode.fis` — **channel declaration:**

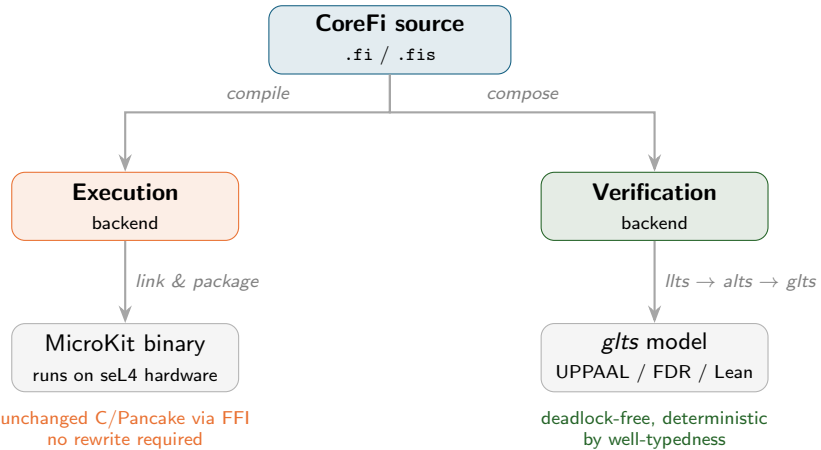
```
Async Chan d2i = dd --> eth_inner
           via eth_inner_input<PKT>;
```

`eth_inner.fi` — **event and process:**

```
evt send_frame.|pkt: PKT|:
    eth.send_frame(pkt);

proc Notified() {
    irq?155 -> Handle_eth()
 [] d2i?pkt -> send_frame.pkt -> Skip;
    Notified();
}
```

CoreFi: Two Backends



Event Block Semantics

An *event block* α is a named sequence of statements executed **atomically** as a single labelled step. The *effect* function computes the resulting store update:

$$\textsc{Evt-Assign}\ \frac{v = [\![e]\!]_{\sigma,\gamma}}{\mathit{effect}(x \leftarrow e,\ \sigma,\ \gamma) = \sigma[x \mapsto v]} \qquad \textsc{Evt-Call}\ \frac{\bar{r} = \overline{\mathit{ref}(x_i,\ \sigma)} \qquad \sigma' = [\![g]\!](\bar{r},\ \sigma)}{\mathit{effect}(\mathit{call}(g,\ \bar{x}),\ \sigma,\ \gamma) = \sigma'}$$

$$\textsc{Evt-Seq}\ \frac{\sigma_0 = \sigma \qquad \forall\ i \in 1..n. \quad \mathit{effect}(\alpha_i,\ \sigma_{i-1},\ \gamma) = \sigma_i}{\mathit{effect}(\alpha_1\ ;\cdots;\ \alpha_n,\ \sigma,\ \gamma) = \sigma_n}$$

$\mathit{ref}(x_i, \sigma)$ produces a by-reference handle: a scalar slot (m, l) or a range $(m, [l, l{+}k))$ within region m. Evt-Call passes these handles to the C function g, which may update any referenced location in σ.

Key property

All intermediate stores are hidden from the rest of the system — only the final σ' is observable.

Local LTS (*llts*): Process Semantics

$$\textsc{Skip}\ \frac{}{\langle SKIP, \sigma, \gamma\rangle \xrightarrow{\checkmark} \langle STOP, \sigma, \gamma\rangle}$$

$$\textsc{Rec}\ \frac{\langle \rho[\mu X.\rho/X], \sigma, \gamma\rangle \xrightarrow{\ell} \langle \rho', \sigma', \gamma'\rangle}{\langle \mu X.\rho, \sigma, \gamma\rangle \xrightarrow{\ell} \langle \rho', \sigma', \gamma'\rangle}$$

$$\textsc{ExtL}\ \frac{\langle \rho_1, \sigma, \gamma\rangle \xrightarrow{\ell} \langle \rho_1', \sigma', \gamma\rangle}{\langle \rho_1 \,\square\, \rho_2, \sigma, \gamma\rangle \xrightarrow{\ell} \langle \rho_1', \sigma', \gamma\rangle} \qquad \textsc{ExtR}\ \frac{\langle \rho_2, \sigma, \gamma\rangle \xrightarrow{\ell} \langle \rho_2', \sigma', \gamma\rangle}{\langle \rho_1 \,\square\, \rho_2, \sigma, \gamma\rangle \xrightarrow{\ell} \langle \rho_2', \sigma', \gamma\rangle}$$

$$\textsc{Send}\ \frac{v = [\![e]\!]_{\sigma,\gamma} \quad |\gamma(c)| < N}{\langle c!e \to \rho, \sigma, \gamma\rangle \xrightarrow{c!v} \langle \rho, \sigma, \gamma[c \mapsto \gamma(c)\cdot v]\rangle}$$

$$\textsc{Loss}\ \frac{v = [\![e]\!]_{\sigma,\gamma} \quad |\gamma(c)| = N}{\langle c!e \to \rho, \sigma, \gamma\rangle \xrightarrow{c!v} \langle \rho, \sigma, \gamma\rangle}$$

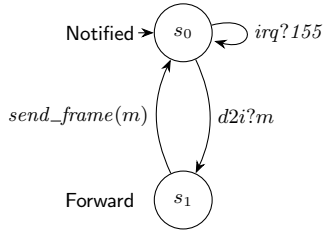
$$\textsc{Recv}\ \frac{\gamma(c) = v \cdot q'}{\langle c?x \to \rho, \sigma, \gamma\rangle \xrightarrow{c?v} \langle \rho, \sigma[x \mapsto v], \gamma[c \mapsto q']\rangle}$$

$$\textsc{Guard-T}\ \frac{[\![b]\!]_{\sigma,\gamma} = true \quad \langle \rho, \sigma, \gamma\rangle \xrightarrow{\ell} \langle \rho', \sigma', \gamma'\rangle}{\langle [b]\rho, \sigma, \gamma\rangle \xrightarrow{[b]\ell} \langle \rho', \sigma', \gamma'\rangle}$$

Local state: $s_p = \langle \rho_p,\ \sigma_p,\ \gamma_p \rangle$

ρ_p process · σ_p store · γ_p buffer view

llts of eth_inner:



Rec unfolds `Notified()` in-place. ExtL/ExtR select the branch. Evt fires $send\_frame(m)$ atomically. Recv blocks when $\gamma(d2i) = \varepsilon$. Loss: buffer full $\Rightarrow$ value silently dropped.

llts $\to$ *alts*: Abstraction Rules

$$\textsc{Hide}\ \frac{\langle \rho,\sigma,\gamma\rangle \xrightarrow{\alpha} \langle \rho',\sigma',\gamma\rangle \quad \alpha \in \Sigma_\alpha}{\langle \rho,\sigma,\Gamma\rangle \xrightarrow{\tau} \langle \rho',\sigma',\Gamma\rangle}$$

$$\textsc{Hide-G}\ \frac{\langle \rho,\sigma,\gamma\rangle \xrightarrow{[b]\alpha} \langle \rho',\sigma',\gamma\rangle \quad \alpha \in \Sigma_\alpha}{\langle \rho,\sigma,\Gamma\rangle \xrightarrow{[b]\tau} \langle \rho',\sigma',\Gamma\rangle}$$

$$\textsc{Collapse}\ \frac{\langle \rho,\sigma,\gamma\rangle \xrightarrow{\tau}{}^{+} \langle \rho',\sigma',\gamma\rangle}{\langle \rho,\sigma,\Gamma\rangle \Longrightarrow \langle \rho',\sigma',\Gamma\rangle}$$

$$\textsc{Collapse-G}\ \frac{\begin{array}{c}\langle \rho,\sigma,\gamma\rangle \xrightarrow{[b]\tau} \langle \rho',\sigma',\gamma\rangle \\ \langle \rho',\sigma',\gamma\rangle \xrightarrow{\tau}{}^{+} \langle \rho'',\sigma'',\gamma\rangle\end{array}}{\langle \rho,\sigma,\Gamma\rangle \xRightarrow{[b]\tau} \langle \rho'',\sigma'',\Gamma\rangle}$$

$$\textsc{Seq-Pre}\ \frac{\begin{array}{c}\langle \rho,\sigma,\gamma\rangle \xrightarrow{\tau}{}^{*} \langle \rho',\sigma',\gamma\rangle \\ \langle \rho',\sigma',\gamma\rangle \xrightarrow{\ell} \langle \rho'',\sigma'',\gamma'\rangle \quad \ell \in \Sigma_c\end{array}}{\langle \rho,\sigma,\Gamma\rangle \Longrightarrow \langle \rho'',\sigma'',\Gamma'\rangle}$$

$$\textsc{Seq-Post}\ \frac{\begin{array}{c}\langle \rho,\sigma,\gamma\rangle \xrightarrow{\ell} \langle \rho',\sigma',\gamma'\rangle \\ \langle \rho',\sigma',\gamma'\rangle \xrightarrow{\tau}{}^{*} \langle \rho'',\sigma'',\gamma'\rangle \quad \ell \in \Sigma_c\end{array}}{\langle \rho,\sigma,\Gamma\rangle \Longrightarrow \langle \rho'',\sigma'',\Gamma'\rangle}$$

$$\textsc{Seq-G}\ \frac{\langle \rho,\sigma,\gamma\rangle \xrightarrow{[b]\tau} \langle \rho',\sigma',\gamma\rangle \quad \langle \rho',\sigma',\gamma\rangle \xrightarrow{\ell} \langle \rho'',\sigma'',\gamma'\rangle \quad \ell \in \Sigma_c}{\langle \rho,\sigma,\Gamma\rangle \xRightarrow{[b]\ell} \langle \rho'',\sigma'',\Gamma'\rangle}$$

Hide/Hide-G: relabel $\alpha \in \Sigma_\alpha$ as τ (hiding computation). Collapse/Collapse-G: contract τ^+-chains to a single big-step $\Longrightarrow$.

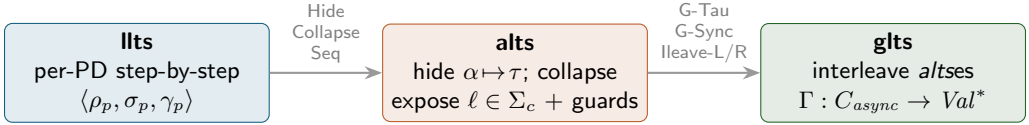
alts → *glts*: Global Composition Rules

Global state: $\langle \mathbf{S},\, \Gamma \rangle$ $\quad$ $\mathbf{S}$: PD local states · $\Gamma : C_{async} \to Val^*$: async channel queues

$$\textsc{G-Sync}\ \frac{s_{p_1} = \langle [b_1]\,(c!e \to \rho_1),\, \sigma_1 \rangle \quad s_{p_2} = \langle [b_2]\,(c?x \to \rho_2),\, \sigma_2 \rangle \quad [\![b_1]\!]_{\sigma_1} = \mathit{true} \quad [\![b_2]\!]_{\sigma_2} = \mathit{true} \quad v = [\![e]\!]_{\sigma_1} \quad c \notin \mathit{dom}(\Gamma)}{\langle \mathbf{S},\, \Gamma \rangle \xrightarrow{c.v} \langle \mathbf{S}[p_1 \mapsto \langle \rho_1, \sigma_1 \rangle,\ p_2 \mapsto \langle \rho_2, \sigma_2[x \mapsto v] \rangle],\ \Gamma \rangle}$$

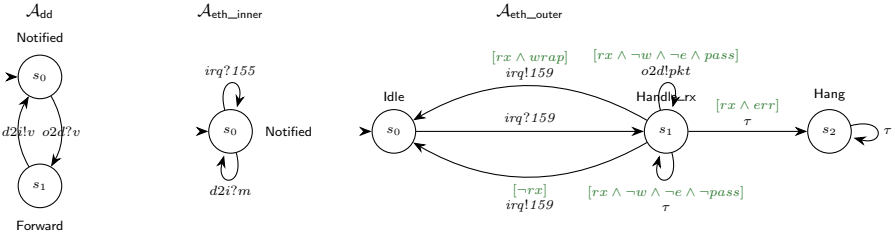
$$\textsc{G-Tau}\ \frac{s^p \xrightarrow{\tau} s^{p\prime}}{\langle \mathbf{S},\, \Gamma \rangle \xrightarrow{\tau} \langle \mathbf{S}[p \mapsto s^{p\prime}],\ \Gamma \rangle} \qquad \textsc{Ileave-L}\ \frac{\mathcal{A}_1 \xrightarrow{\ell} \mathcal{A}_1'}{\mathcal{A}_1 \,|||\, \mathcal{A}_2 \xrightarrow{\ell} \mathcal{A}_1' \,|||\, \mathcal{A}_2} \qquad \textsc{Ileave-R}\ \frac{\mathcal{A}_2 \xrightarrow{\ell} \mathcal{A}_2'}{\mathcal{A}_1 \,|||\, \mathcal{A}_2 \xrightarrow{\ell} \mathcal{A}_1 \,|||\, \mathcal{A}_2'}$$

G-Sync: synchronous rendezvous on $c \in C_{sync}$; both guards must hold; $c \notin dom(\Gamma)$ (sync channels have no buffer). G-Tau: lifts a local τ-step to the global level; Γ unchanged. Ileave-L/Ileave-R: async send/recv (Send/Loss/Recv) are local rules lifted to global via interleaving; they update Γ as they fire.



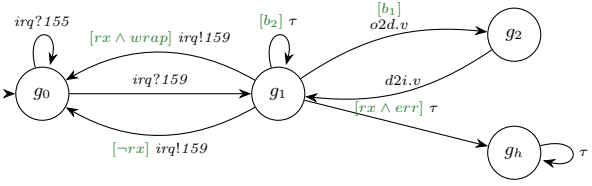
From Local Processes to a Global Model

Data diode altses — after hiding all computation, only channel events remain:



Computation events hidden as τ; IRQ labels kept observable; intermediate states eliminated by Seq-Post and Seq-G.

Composed glts ζ — interleaving the three *alts*es:



DEMO 3

Open VSCode · Show `dd.fi` and `datadiode.fis`

Summary and Future Work

Takeaway

- A **unified infrastructure** for execution and verification.
- Formalisation and Verification **Automation**
- Minimal change to already implemented applications
- **Pluggable** various verification backend
  Future work

Future Work

- **Compilation pipeline**
  Integration with libmicrokit and FFIs to low-level implementation
- **Model checking integration**
  Abstract the data, reduce infinite data range to finite domain
- **Information-flow security**
  Language-based information security; enforcing security policy by type checking

Vision

Verified kernel (seL4) + verified compiler (CompCert/Pancake) + CoreFi application correctness = **end-to-end verified embedded system**.
$\longrightarrow$ minimal TCB

Thank You

Questions?