A program logic for system verification

Matt Brecknell

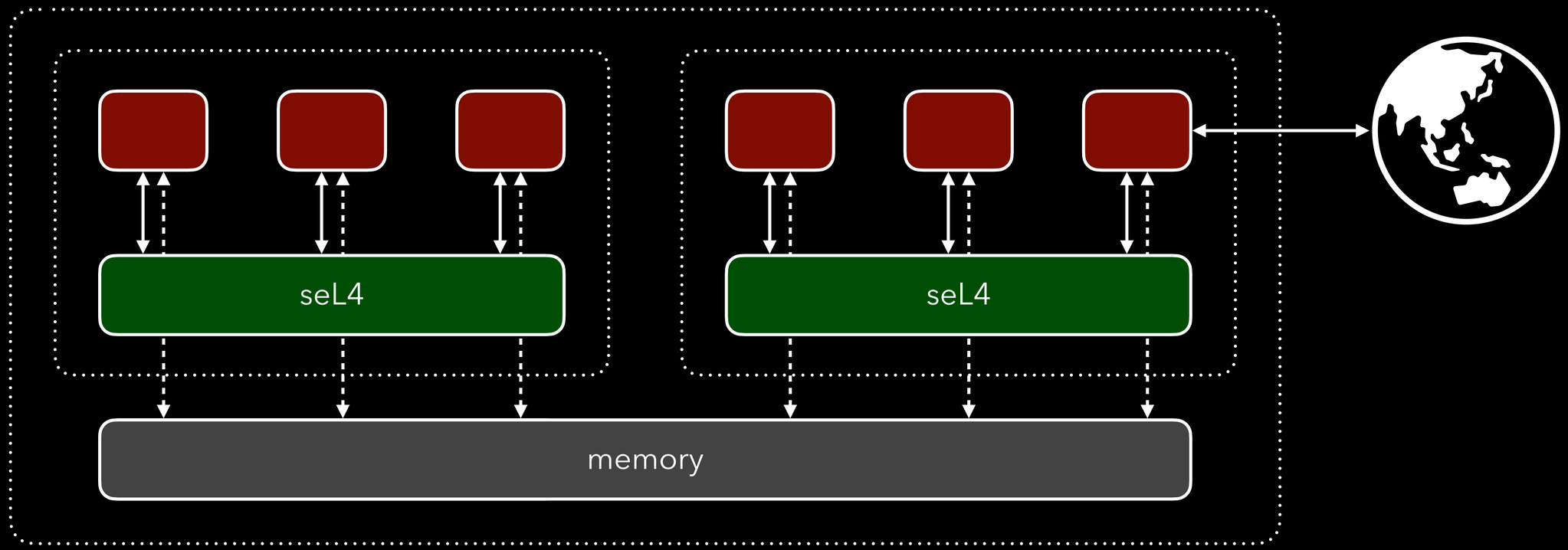
Kry10

seL4 Summit – 3 September 2025 – Prague

Funded by the Agentur für Innovation in Cybersicherheit GmbH
Ecosystem of Formally Verifiable IT - Provable Cybersecurity

www.cyberagentur.de/en/programs/evit/

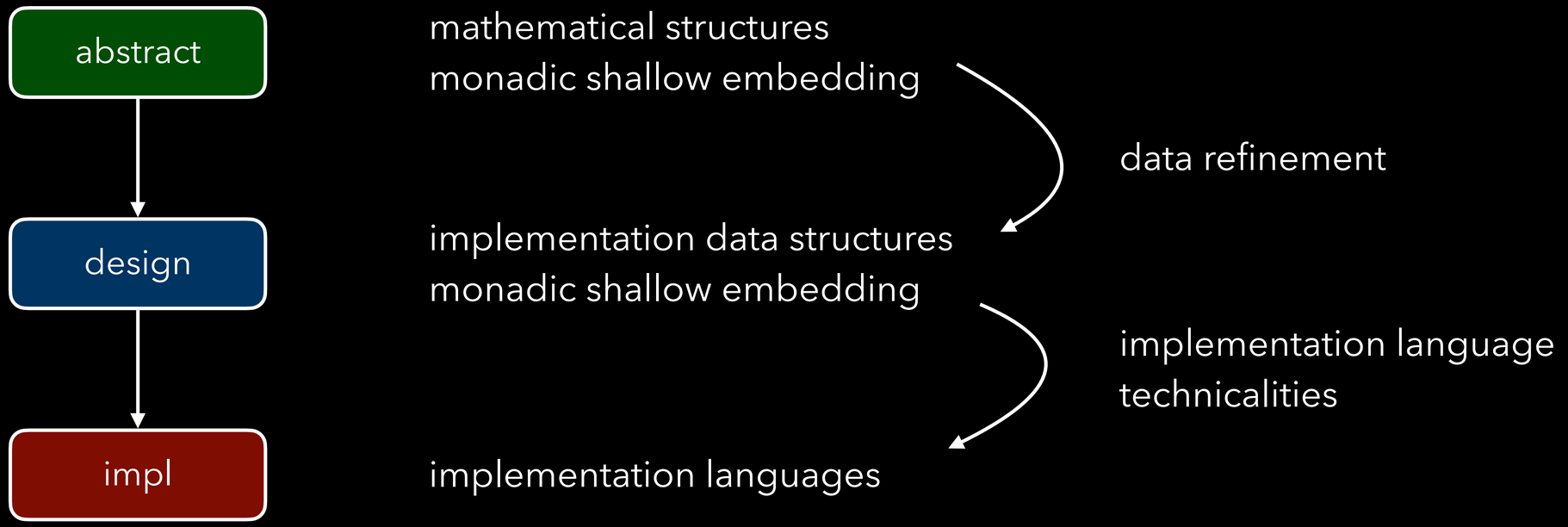
What is a system?



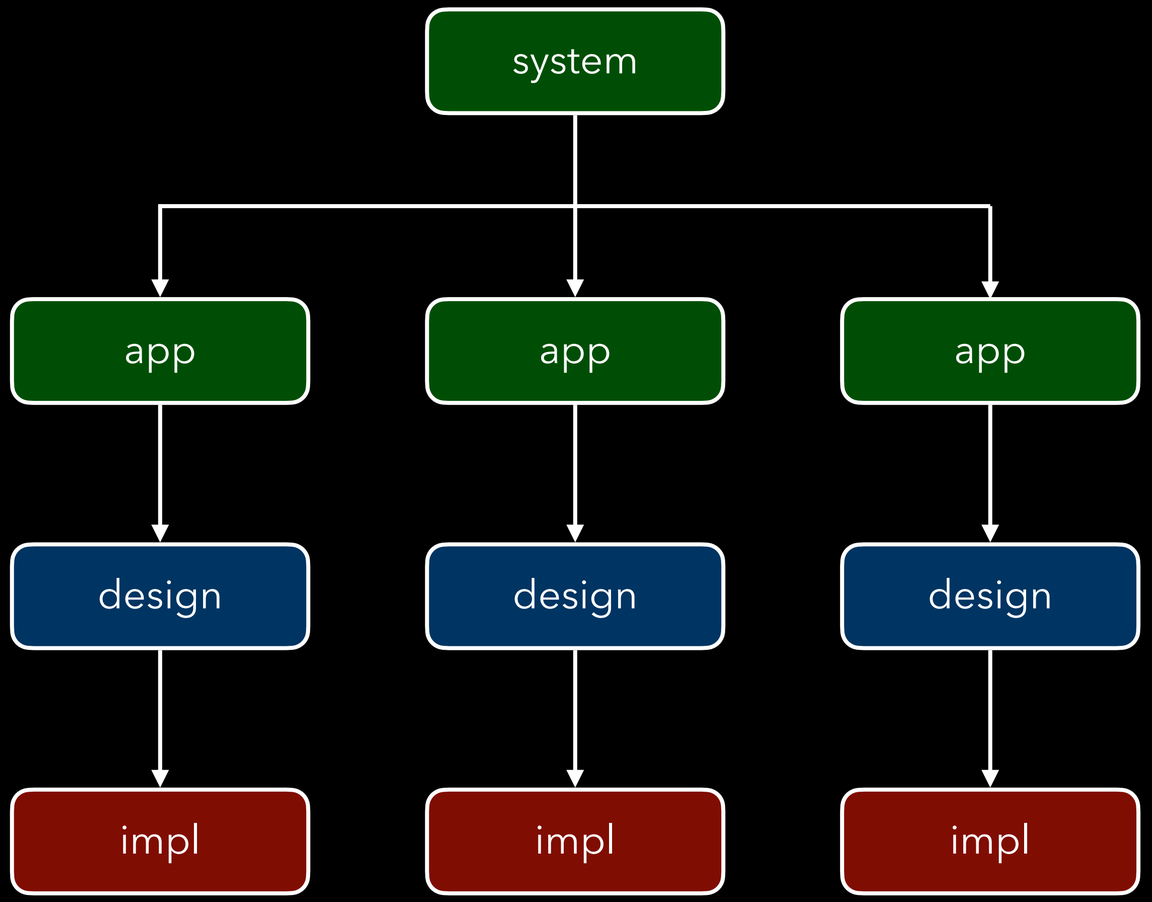
- Composition
- Internal & external interaction

What is system verification?

(seL4 verification recap)



What is system verification?



refinement from
sequential system spec to
concurrent app specs

What is system verification?

1. Specification language
   - system and application specifications
2. Semantics
   - define system states
   - define how they execute
3. Program logic
   - reason about applications and systems
   - prove refinement

Interaction trees
(Xia et al. 2020)

ROCQ

Iris separation logic
(Jung et al. 2015)

Specifications as interaction trees


```
adder (ep_slot reply_slot : slot) (sum : word) :=
  x ← ep_recv ep_slot reply_slot ;
  ep_reply reply_slot sum ;
  adder ep_slot reply_slot (sum + x)
```

Specifications as interaction trees


```
adder (ep_slot reply_slot : slot) (sum : word) :=
  x ← ep_recv ep_slot reply_slot ;
  ep_reply reply_slot sum ;
  adder ep_slot reply_slot (sum + x)
```

```
Variant invocation : Type → Type :=
  | EP_Recv (ep_slot reply_slot : slot)
    : invocation word
  | …

ep_recv (ep_slot reply_slot : slot)
        : itree invocation word :=
  trigger (EP_Recv ep_slot reply_slot)
```

Specifications as interaction trees


```
adder (ep_slot reply_slot : slot) (sum : word) :=
  x ← ep_recv ep_slot reply_slot ;
  ep_reply reply_slot sum ;
  adder ep_slot reply_slot (sum + x)
```

```
client (ep_slot mmio_slot : slot) :=
  x ← read mmio_slot ;
  r ← ep_call ep_slot x ;
  write mmio_slot r ;
  client ep_slot mmio_slot
```

Specifications as interaction trees


```
adder (ep_slot reply_slot : slot) (sum : word) :=
  x ← ep_recv ep_slot reply_slot ;
  ep_reply reply_slot sum ;
  adder ep_slot reply_slot (sum + x)
```

```
client (ep_slot mmio_slot : slot) :=
  x ← read mmio_slot ;
  r ← ep_call ep_slot x ;
  write mmio_slot r ;
  client ep_slot mmio_slot
```

```
spec (mmio_slot : slot) (sum : word) :=
  x ← read mmio_obj ;
  write mmio_obj sum ;
  spec mmio_slot (sum + x)
```

```
adder || client ⩽ spec
```

What is system verification?

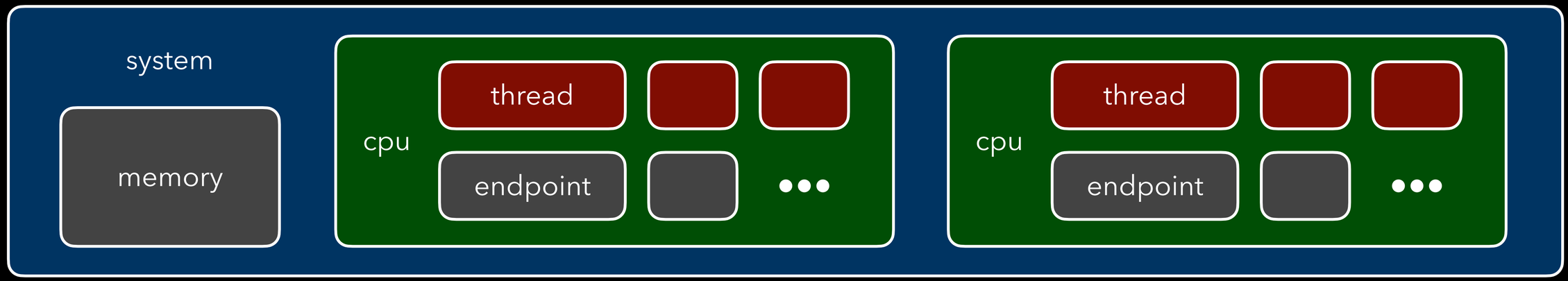
1. Specification language
   - system and application specifications
2. Semantics
   - define system states
   - define how they execute
3. Program logic
   - reason about applications and systems
   - prove refinement

Interaction trees
(Xia et al. 2020)

ROCQ

Iris separation logic
(Jung et al. 2015)

System states

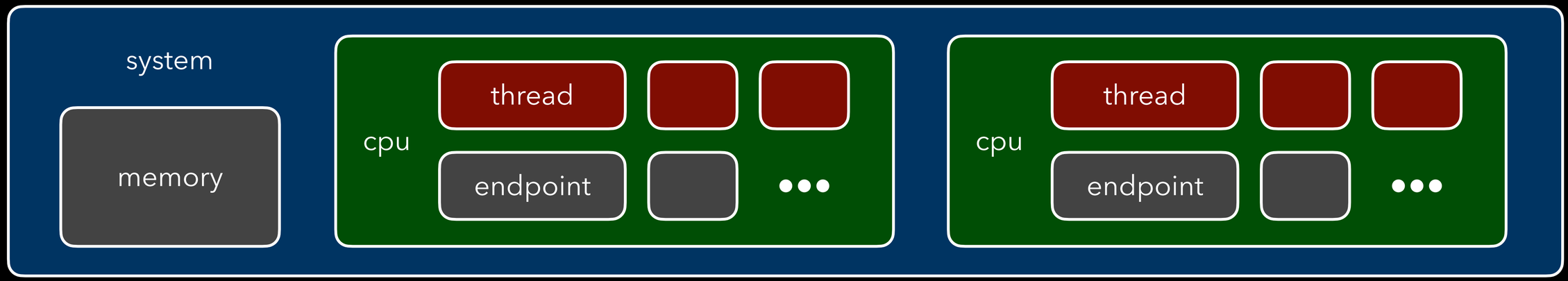



```
Record thread := {
  prog : itree thread_event void ;
  caps …
}.

Record system_state := {
  cpus : gmap cpu_id cpu_state ;
  memory …
}.
```

```
Record cpu_state := {
  threads : gmap thread_id thread ;
  endpoints …
}.
```

Semantics as interaction trees




```
run_cpu (s : cpu_state) : itree cpu_event void.
```

- Interpret seL4 invocations
- Interleave thread steps
- Expose memory and I/O operations

```
run_system (s : system_state) : itree system_event void.
```

- Interpret memory operations
- Interleave CPU steps
- Expose I/O operations

What is system verification?

1. Specification language
   - system and application specifications
2. Semantics
   - define system states
   - define how they execute
3. Program logic
   - reason about applications and systems
   - prove refinement

Interaction trees
(Xia et al. 2020)

ROCQ

Iris separation logic
(Jung et al. 2015)

Program logic

From Iris

- Base logic
- Step indexed model
- Resource algebra library
- Ghost resources
- Invariants
- Proof mode
- ~~Language framework~~
- ~~Program logic~~

New

- Specifications as interaction trees
- Weakest-precondition program logic
- seL4-specific assertions and rules

Program logic

- Ownership assertions
  - *thread-points-to* — $\mathit{thread\_id} \Longmapsto_{\mathrm{thread}} \mathit{itree\_program}$
  - *endpoint-points-to* — $\mathit{ep\_id} \Longmapsto_{\mathrm{ep}} \mathit{ep\_state}$
  - *slot-points-to* — $\mathit{slot\_id} \mathbin{@} \mathit{thread\_id} \Longmapsto_{\mathrm{slot}} \mathit{capability}$

- Weakest precondition assertion
  - resources needed to execute safely — $\mathtt{WP}\ \mathit{thread\_id}\ \{\!\{\ \varphi\ \}\!\}$

Program logic

Proof by symbolic execution

ep_obj $\Longmapsto_{ep}$ EP_Call_Queue [(t_{client}, x)] ⟵ Assume: temporary ownership of kernel objects

ep @ t_{adder} $\Longmapsto_{slot}$ Cap_EP ep_obj {[EP_Recv]}
reply @ t_{adder} $\Longmapsto_{slot}$ Cap_Null ⟵ Assume: ownership of capability slots

t_{adder} $\Longmapsto_{thread}$ (r ← ep_recv ep reply; k r) ⟵ Assume: ownership of the executing program

——————————————————————— *

WP t_{adder} {{ False }} ⟵ Goal: prove that t_{adder} is safe to execute

Program logic

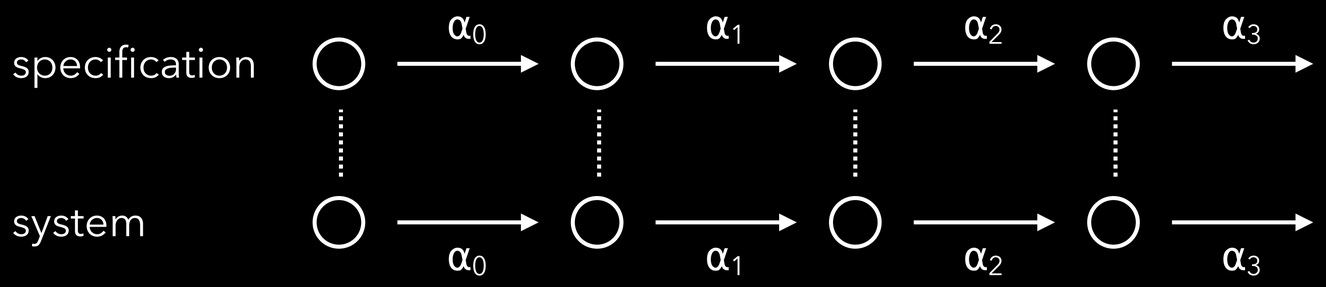
Proof by symbolic execution

$$\frac{\begin{array}{l}\texttt{ep\_obj} \Longmapsto_{\text{ep}} \texttt{EP\_Call\_Queue [(}t_{\text{client}}\texttt{, x)]} \\ \texttt{ep @ }t_{\text{adder}} \Longmapsto_{\text{slot}} \texttt{Cap\_EP ep\_obj \{[EP\_Recv]\}} \\ \texttt{reply @ }t_{\text{adder}} \Longmapsto_{\text{slot}} \texttt{Cap\_Null} \\ t_{\text{adder}} \Longmapsto_{\text{thread}} \texttt{(r ← ep\_recv ep reply; k r)}\end{array}}{\texttt{WP }t_{\text{adder}}\texttt{ \{\{ False \}\}}}*$$

$$\frac{\begin{array}{l}\texttt{ep\_obj} \Longmapsto_{\text{ep}} \texttt{EP\_Idle} \\ \texttt{ep @ }t_{\text{adder}} \Longmapsto_{\text{slot}} \texttt{Cap\_EP ep\_obj \{[EP\_Recv]\}} \\ \texttt{reply @ }t_{\text{adder}} \Longmapsto_{\text{slot}} \texttt{Cap\_Reply }t_{\text{client}} \\ t_{\text{adder}} \Longmapsto_{\text{thread}} \texttt{(k x)}\end{array}}{\texttt{WP }t_{\text{adder}}\texttt{ \{\{ False \}\}}}*$$

Program logic — refinement

Every system I/O event can be matched by the specification



Program logic — refinement

Every system I/O event can be matched by the specification

- I/O specifications
  - *exclusive right to perform specified sequence of I/O operations*

`io_spec` *itree_program*

Program logic – refinement

Every system I/O event can be matched by the specification

$$\frac{\texttt{io\_spec}\ (x \leftarrow \texttt{read}\ \texttt{mmio}_{\text{obj}};\ \texttt{k}_{\text{spec}}\ x) \quad \texttt{mmio}_{\text{slot}}\ @\ \texttt{t}_{\text{adder}} \Longmapsto_{\text{slot}} \texttt{Cap\_MMIO}\ \texttt{mmio}_{\text{obj}} \quad \texttt{t}_{\text{client}} \Longmapsto_{\text{thread}} (x \leftarrow \texttt{read}\ \texttt{mmio}_{\text{slot}};\ \texttt{k}_{\text{client}}\ x)}{\texttt{WP}\ \texttt{t}_{\text{client}}\ \{\{\ \texttt{False}\ \}\}}\ *$$

- ⟵ Assume: ownership of I/O trace specification
- ⟵ Assume: ownership of capability slots
- ⟵ Assume: ownership of the executing program
- ⟵ Goal: prove that $\texttt{t}_{\text{client}}$ is safe to execute

Program logic – refinement

Every system I/O event can be matched by the specification

$$\frac{\texttt{io\_spec}\ (\texttt{x} \leftarrow \texttt{read}\ \texttt{mmio}_{\text{obj}};\ \texttt{k}_{\text{spec}}\ \texttt{x}) \qquad \texttt{mmio}_{\text{slot}}\ @\ \texttt{t}_{\text{adder}} \Longmapsto_{\text{slot}} \texttt{Cap\_MMIO}\ \texttt{mmio}_{\text{obj}} \qquad \texttt{t}_{\text{client}} \Longmapsto_{\text{thread}} (\texttt{x} \leftarrow \texttt{read}\ \texttt{mmio}_{\text{slot}};\ \texttt{k}_{\text{client}}\ \texttt{x})}{\texttt{WP}\ \texttt{t}_{\text{client}}\ \{\{\ \texttt{False}\ \}\}}\ *$$

$$\frac{\texttt{io\_spec}\ (\texttt{k}_{\text{spec}}\ \texttt{x}) \qquad \texttt{mmio}\ @\ \texttt{t}_{\text{adder}} \Longmapsto_{\text{slot}} \texttt{Cap\_MMIO}\ \texttt{mmio}_{\text{obj}} \qquad \texttt{t}_{\text{client}} \Longmapsto_{\text{thread}} (\texttt{k}_{\text{client}}\ \texttt{x})}{\texttt{WP}\ \texttt{t}_{\text{client}}\ \{\{\ \texttt{False}\ \}\}}\ *$$

What is system verification?

1. Specification language
   - system and application specifications
2. Semantics
   - define system states
   - define how they execute
3. Program logic
   - reason about applications and systems
   - prove refinement

Interaction trees
(Xia et al. 2020)

ROCQ

Iris separation logic
(Jung et al. 2015)

Conclusion

- Status
  - Implementing semantics and logic for a simplified model of seL4
- Topics for another talk
  - Reasoning about shared resources using invariants and ghost resources
  - Robustness: proving properties in the presence of untrusted apps
  - Nondeterminism and failure
- Scalability
  - Requires modularity, compositionality, abstraction

Funded by the Agentur für Innovation in Cybersicherheit GmbH
Ecosystem of Formally Verifiable IT - Provable Cybersecurity

www.cyberagentur.de/en/programs/evit/