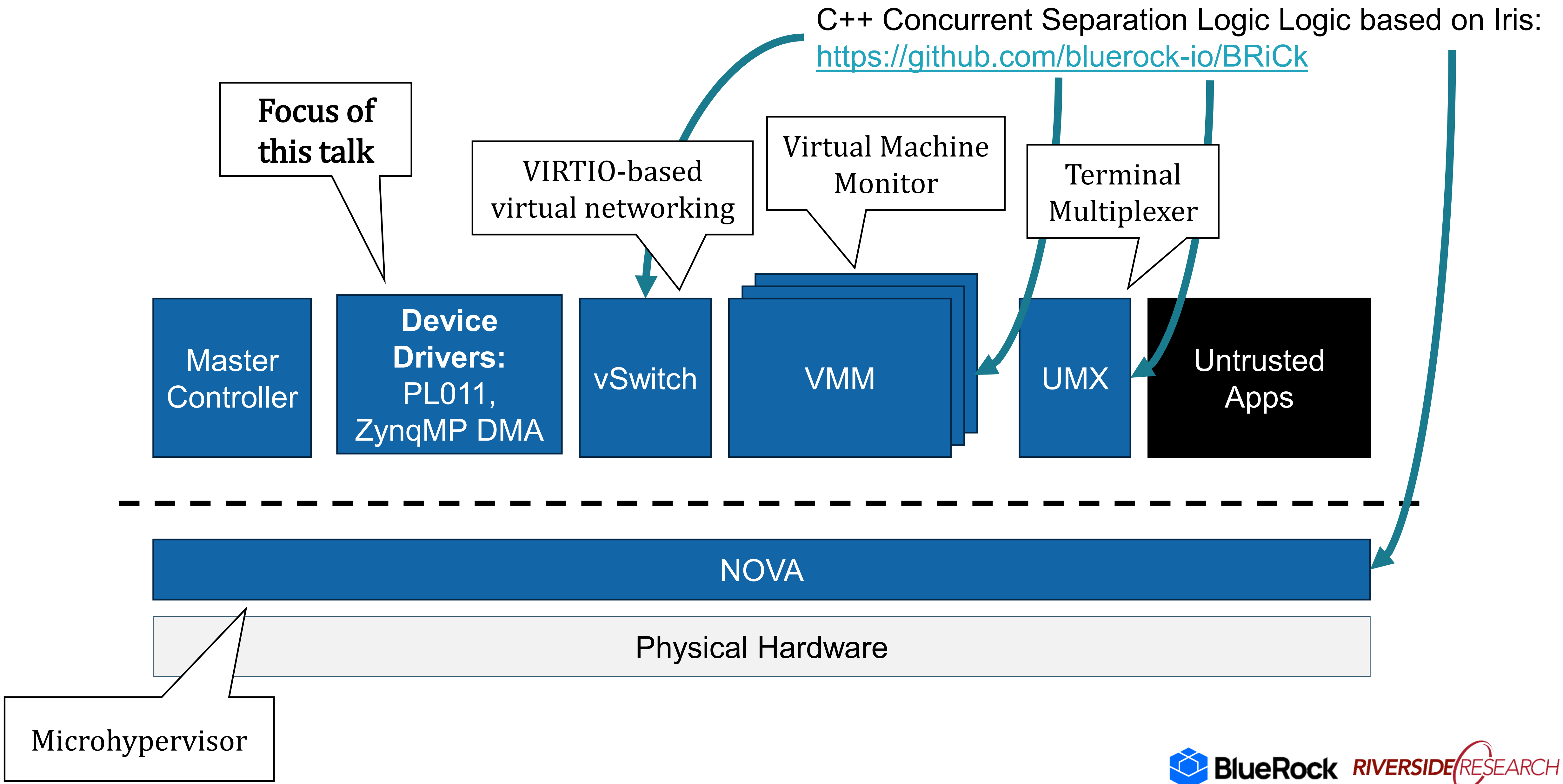
Verified ZynqMP DMA Driver in Concurrent Separation Logic

Gordon Stewart
Riverside Research

Gregory Malecha
BlueRock Security

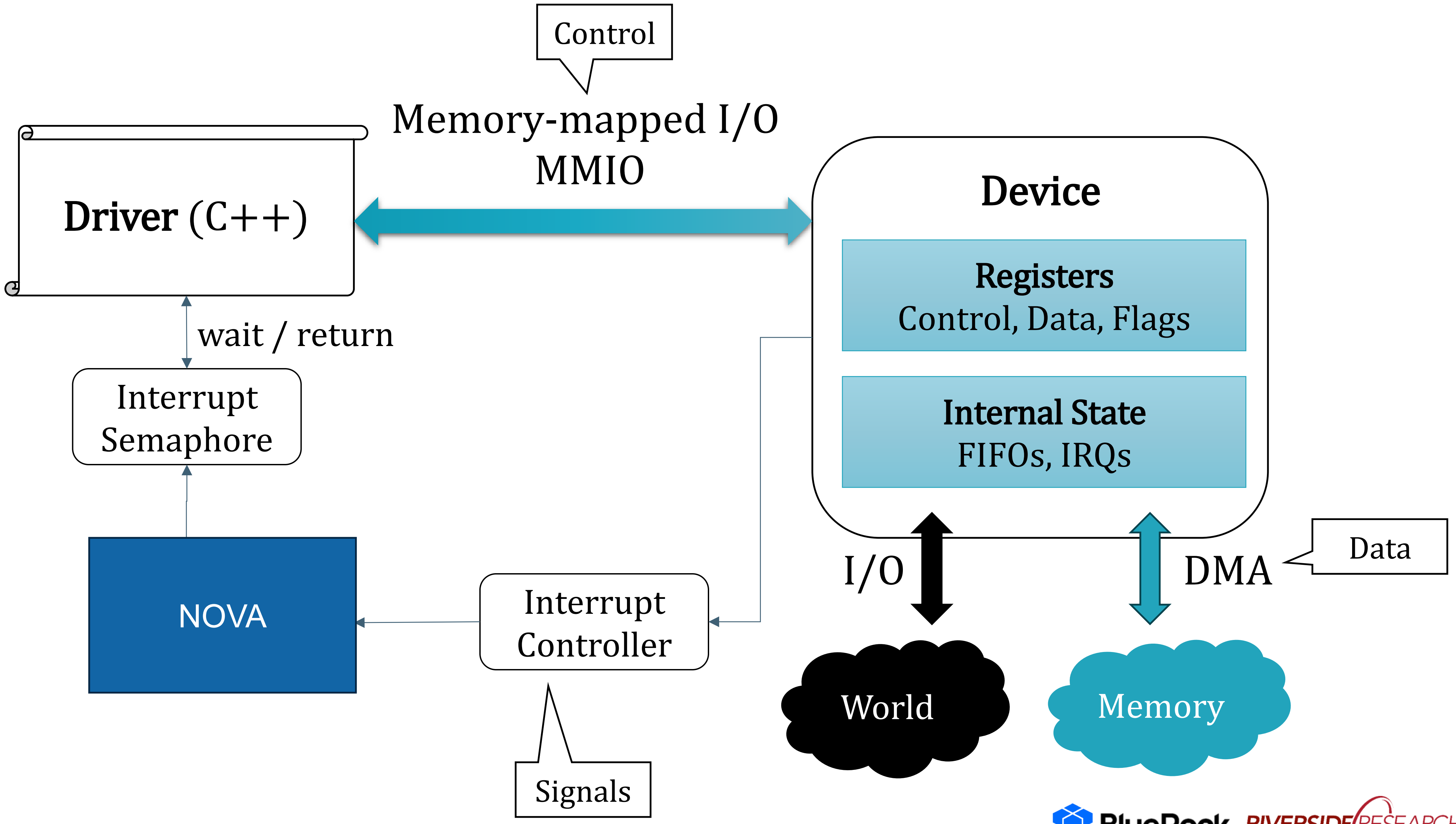
seL4 Summit, Prague, Czechia
3 Sep 2025

Operating Systems Verification at BlueRock Security


C++ Concurrent Separation Logic Logic based on Iris:
https://github.com/bluerock-io/BRiCk
Focus of this talk
VIRTIO-based virtual networking
Virtual Machine Monitor
Terminal Multiplexer
Master Controller
Device Drivers: PL011, ZynqMP DMA
vSwitch
VMM
UMX
Untrusted Apps
NOVA
Physical Hardware
Microhypervisor

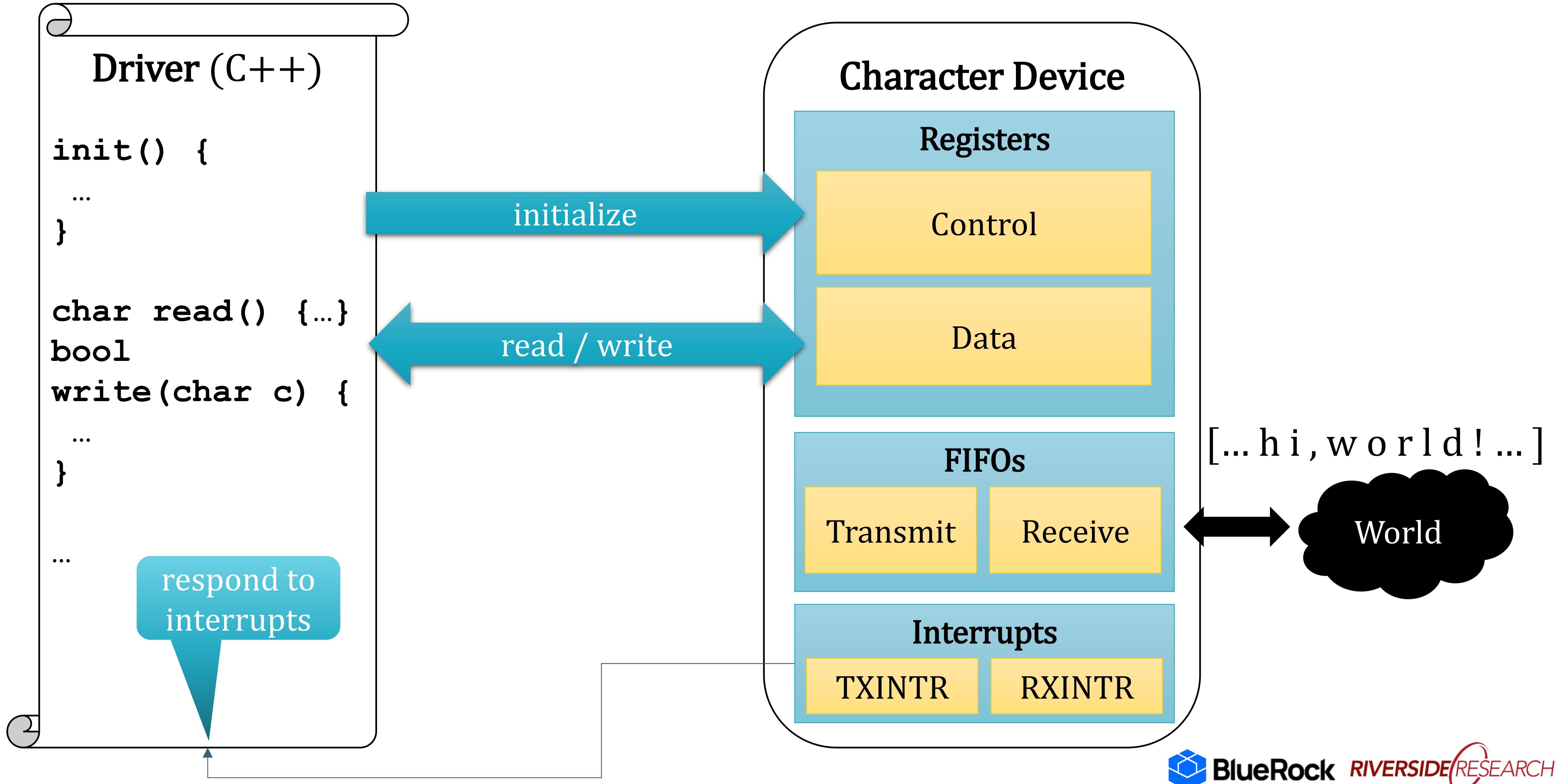
BlueRock RIVERSIDE RESEARCH

Device Drivers



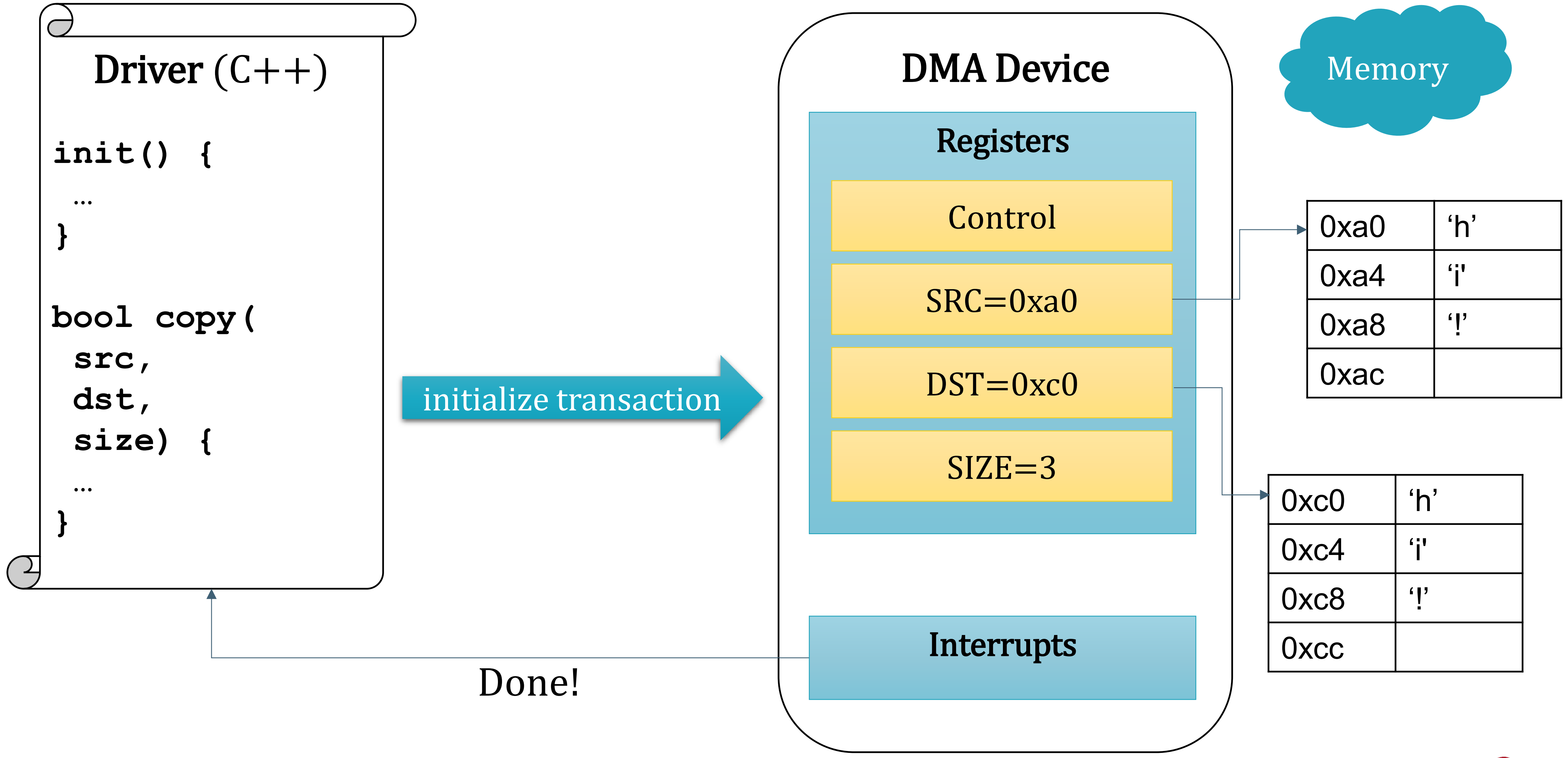
BlueRock RIVERSIDE RESEARCH

Example #1: Character Device (UART)



BlueRock RIVERSIDE RESEARCH

Example #2: Direct-Memory Access (DMA, simple mode)



```
init() {
  …
}

bool copy(
  src,
  dst,
  size) {
  …
}
```

0xa0	‘h’
0xa4	‘i’
0xa8	‘!’
0xac	

0xc0	‘h’
0xc4	‘i’
0xc8	‘!’
0xcc	

Outline

➡ Protocol-based verification by example:

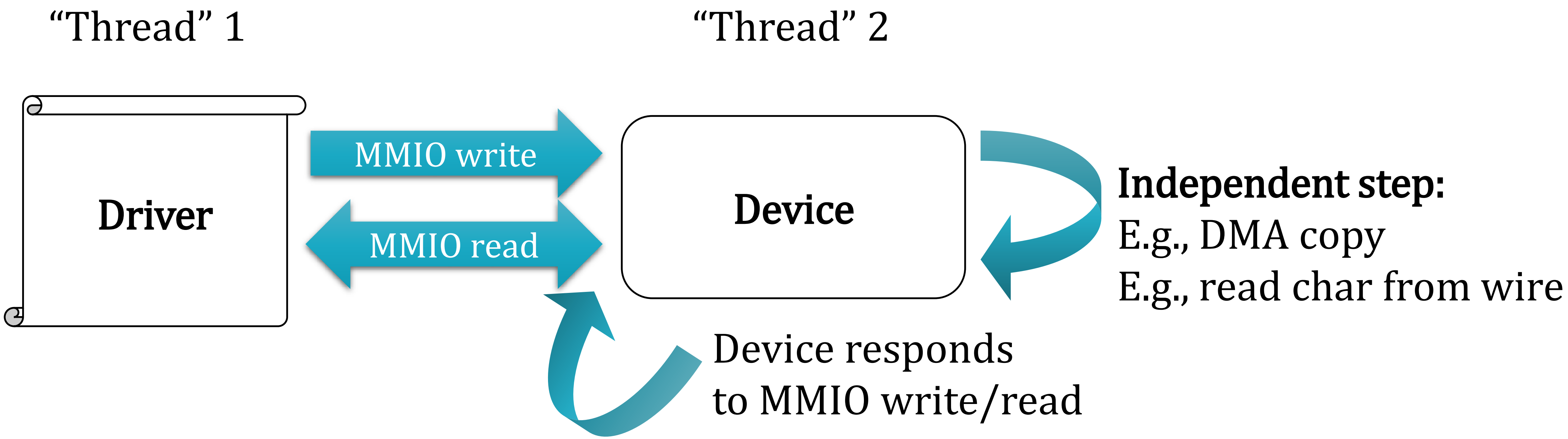
PL011

ZynqMP DMA

seL4

RIVERSIDE RESEARCH

Protocol-based Driver Verification

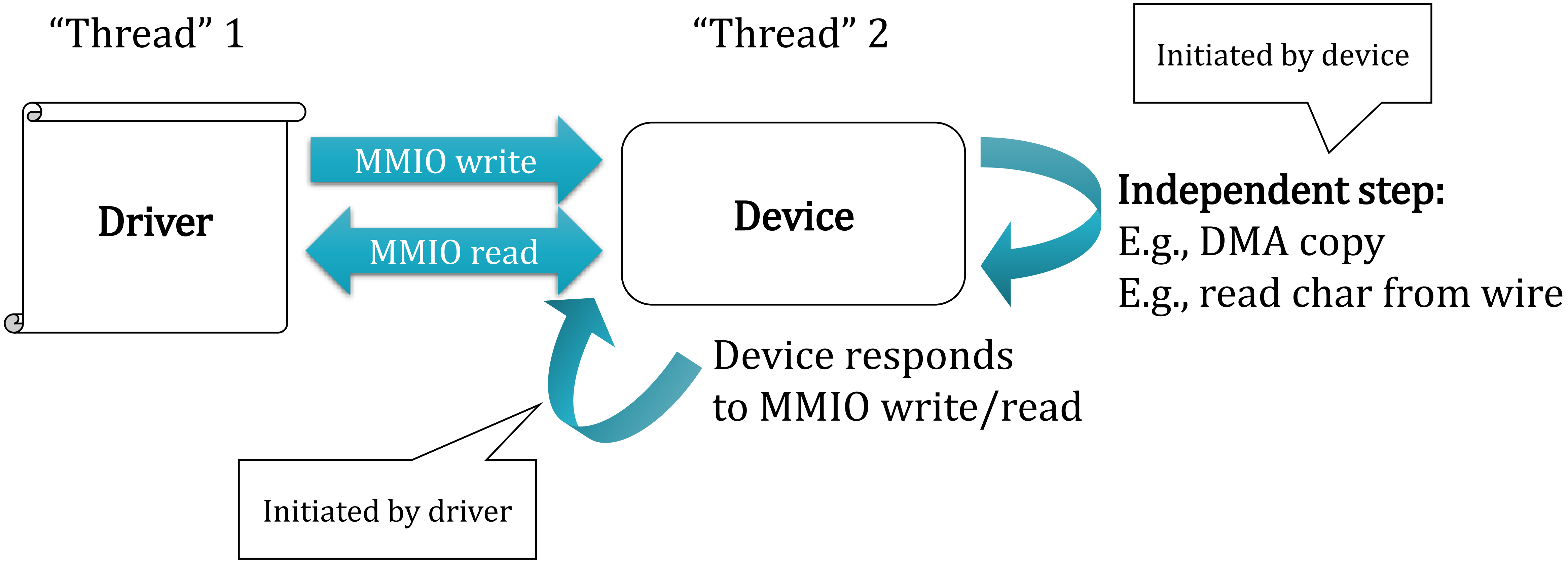


Device and driver run concurrently.

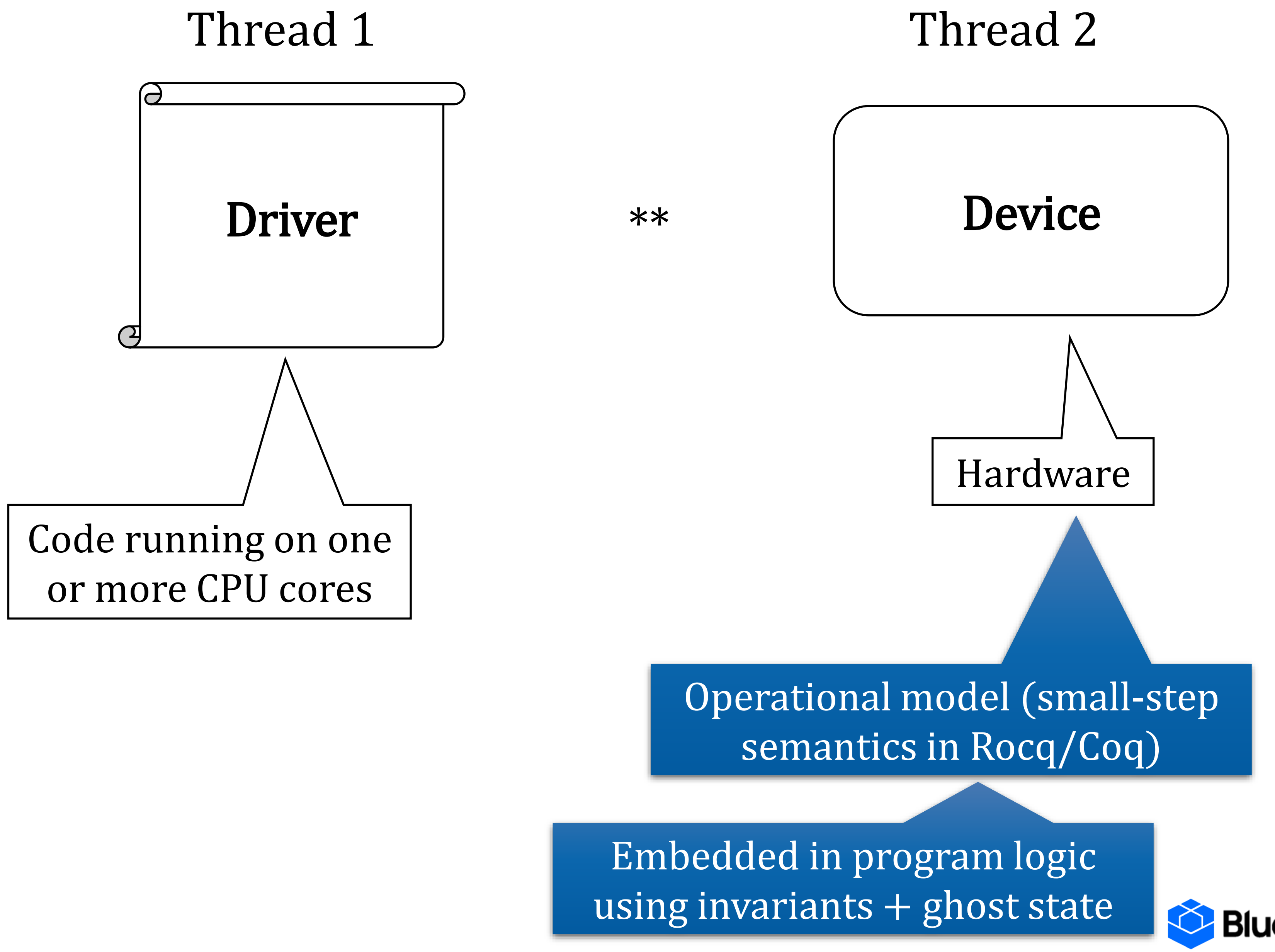
Treat each as a thread.

Interactions are on the shared state of the device.

Device and driver run concurrently



Treat driver and device as independent “threads”


Thread 1
Thread 2
Driver
**
Device
Code running on one or more CPU cores
Hardware
Operational model (small-step semantics in Rocq/Coq)
Embedded in program logic using invariants + ghost state

BlueRock RIVERSIDE RESEARCH

Embedding Device States

Model state of a UART device

```
(* Device State: *)
#[local] Notation RxQ := (Queue.t N queue_size) (only parsing).
#[local] Notation TxQ := (Queue.t N queue_size) (only parsing).
Record State : Type :=
  { rx : RxQ (* the receive buffer *)
  ; tx : TxQ (* the transmit buffer *)
  ; regs : @map Reg N (* the register state *)
  ; irq_raised : @map Irq bool (* interrupts currently asserted *)
  ; irqs : list IntAction }. (* pending irqs (not yet delivered) *)
```

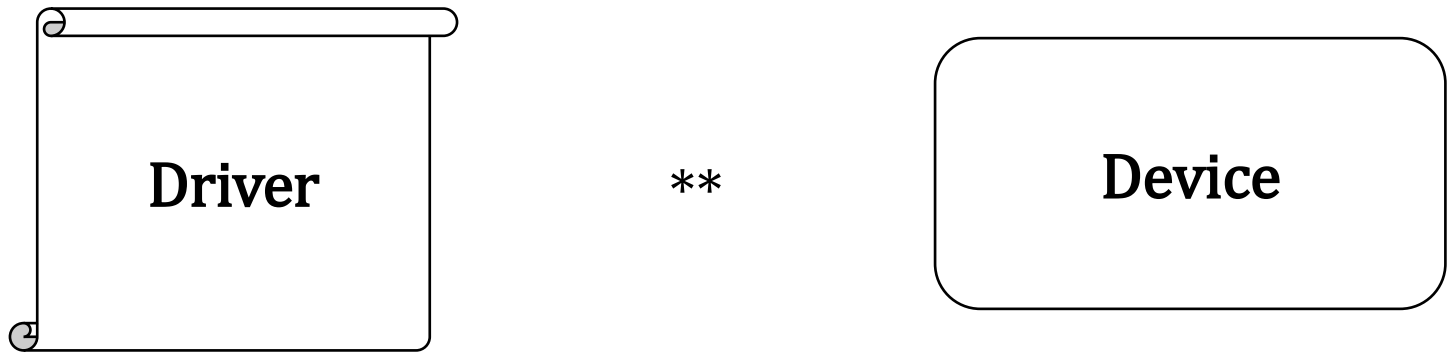
Device

```
| WRITE_DR s c (pf : ~Queue.full s.(tx))
          (_ : s .^ _uart_enabled) (_ : trimN8 c = c) :
  let new_tx := Queue.enqueue c s.(tx) pf in
  step s
    (System (CpuWrite (natural_size_of DR) (offset_of DR) c))
    (s &:         _tx .= new_tx
       &:     _txintr .= false)
```

Example MMIO write step

BlueRock RIVERSIDE RESEARCH

Embedding Device States



Iris authoritative camera

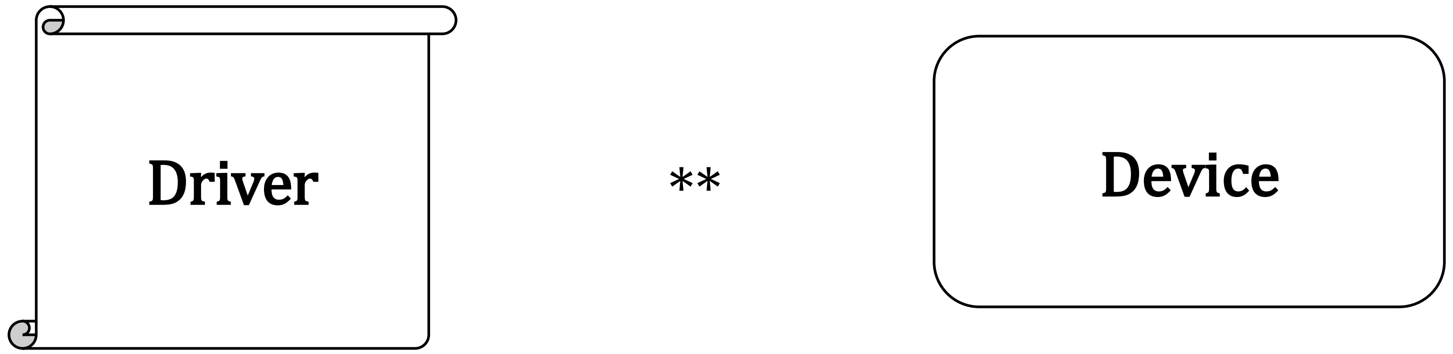
Ghost location containing device state

```
State.frag γ st    **    State.auth γ st’
```

Driver view of state `st`

Device view of state `st’`

Embedding Device States



Consistency:

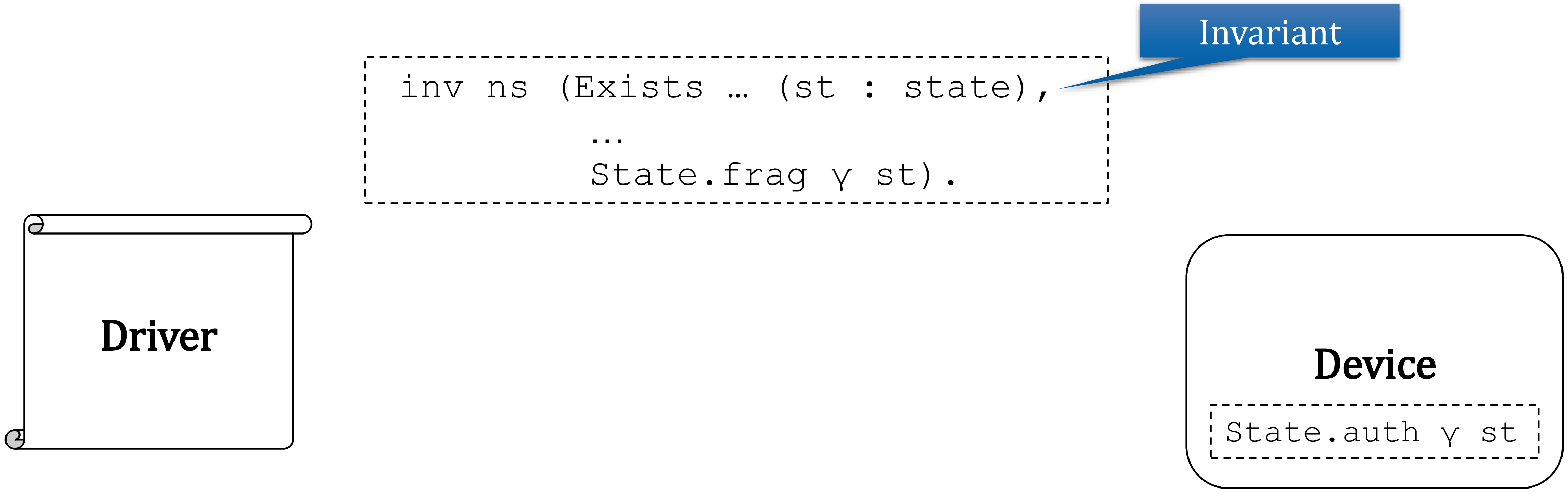
```
    State.frag γ st ** State.auth γ st' -* [|st=st'|] ** …
```

Update:

```
        State.frag γ st ** State.auth γ st' -*
    |=> State.frag γ new_st ** State.auth γ new_st
```

Arbitrary new device state…

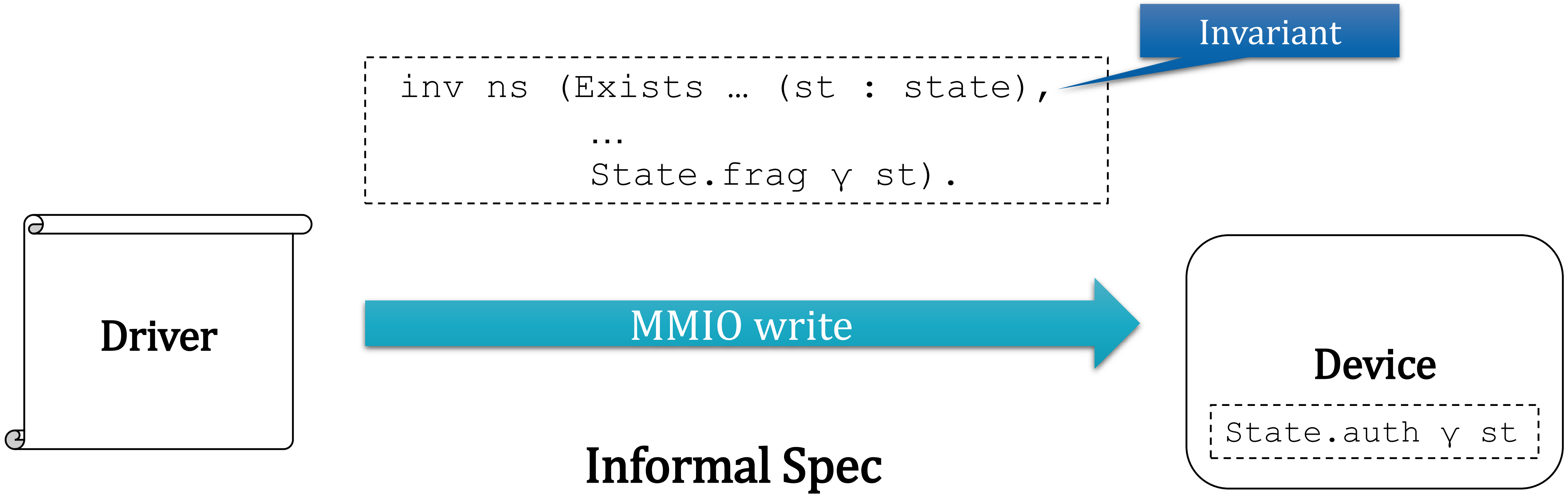
Embedding Device States



```
inv ns (Exists … (st : state),
          …
          State.frag γ st).
```

```
State.auth γ st
```

MMIO Operations



```
inv ns (Exists … (st : state),
           …
           State.frag γ st).
```

```
State.auth γ st
```

Informal Spec

If `MMIO::write` is safe for current device state `st`,
Then `MMIO::write` updates device state to some `st’` reachable via a `CpuWrite` event.

Overall postcondition is some consequence `Q`.

MMIO write spec, formally

```
Definition state_inv γ (ns : namespace) : mpred :=
 inv ns (Exists … (st : state),
         …
         State.frag γ.(_state) st).

Definition mmio_write_spec (sz : bitsize) : WpSpec_cpp :=
  let ty := Tint sz Unsigned in
  \with γdev offset Q ns
  \arg{port} "port" (Vptr port)
  \arg{val} "val" (Vn val) \prepost port |-> MMIOReg sz offset dev
  \pre AU
    <<∀ st, State.frag γdev st ** [| write_safe st sz offset val |]>>
      @ protocol_mask ns, protocol_mask ns \ ↑stateinv_ns ns
    <<∃ st',
      [| dev_step st (System (CpuWrite (bitsN sz) offset val)) st' |] **
      State.frag γdev st',
    COMM Q>>
  \post Q.
```

Assuming some device state (will be constrained in a bit)

… and it is safe to write to the device register in this state …

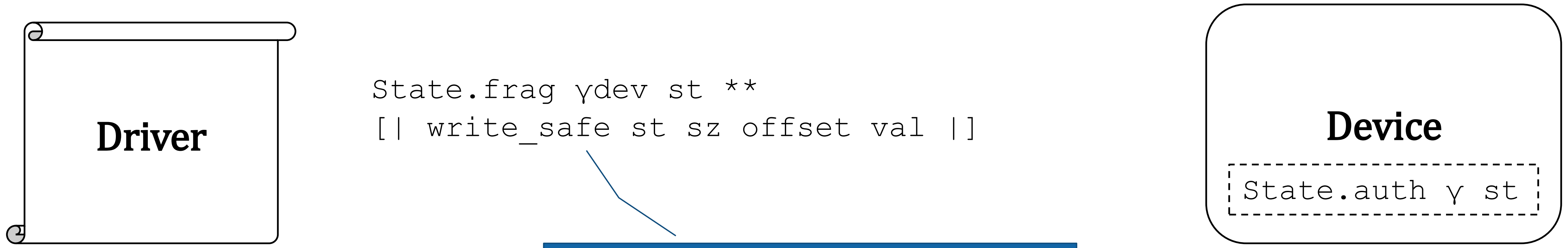
… then the device will update its state to `st'`, in accord with its operational model.

BlueRock RIVERSIDE RESEARCH

Protocols to constrain device states

```
inv ns (Exists … (st : state),
             ...
             State.frag γ st).
```

Arbitrary state



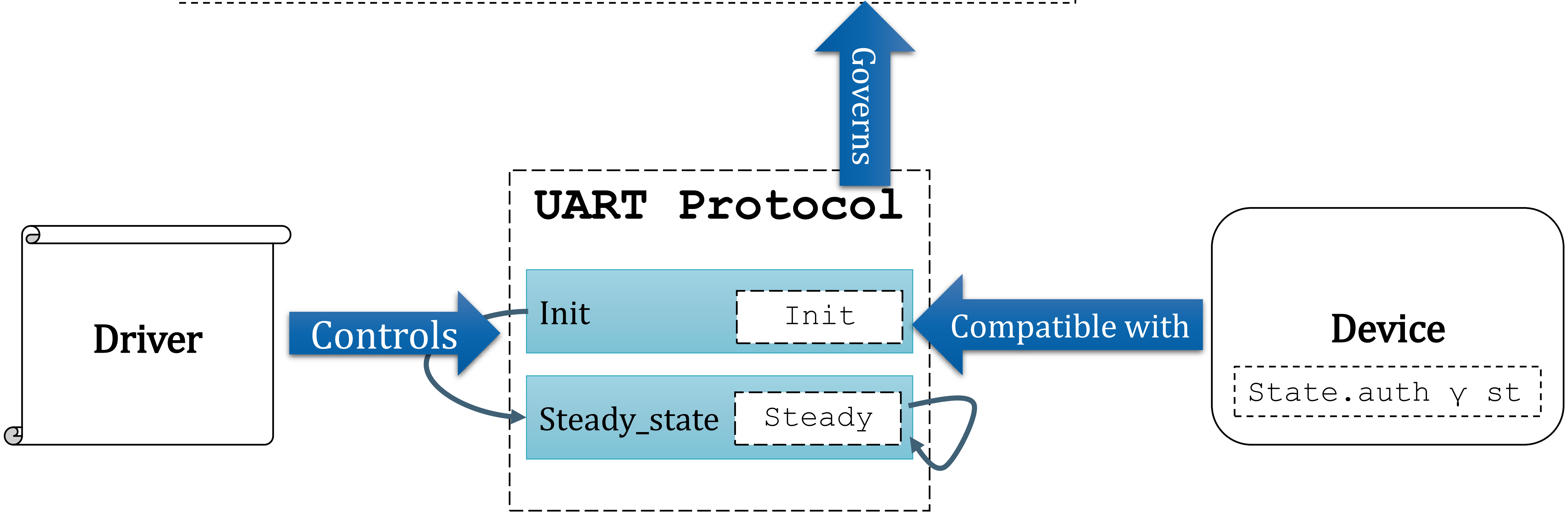
```
State.frag γdev st **
[| write_safe st sz offset val |]
```

```
State.auth γ st
```

How to prove safety (and other deeper properties)?

Protocols to constrain device states

```
inv ns (Exists (P : T) (st : state),
          Proto.frag γ.(_proto) (1/2) P **
          interp P st **
          State.frag γ.(_state) st).
```

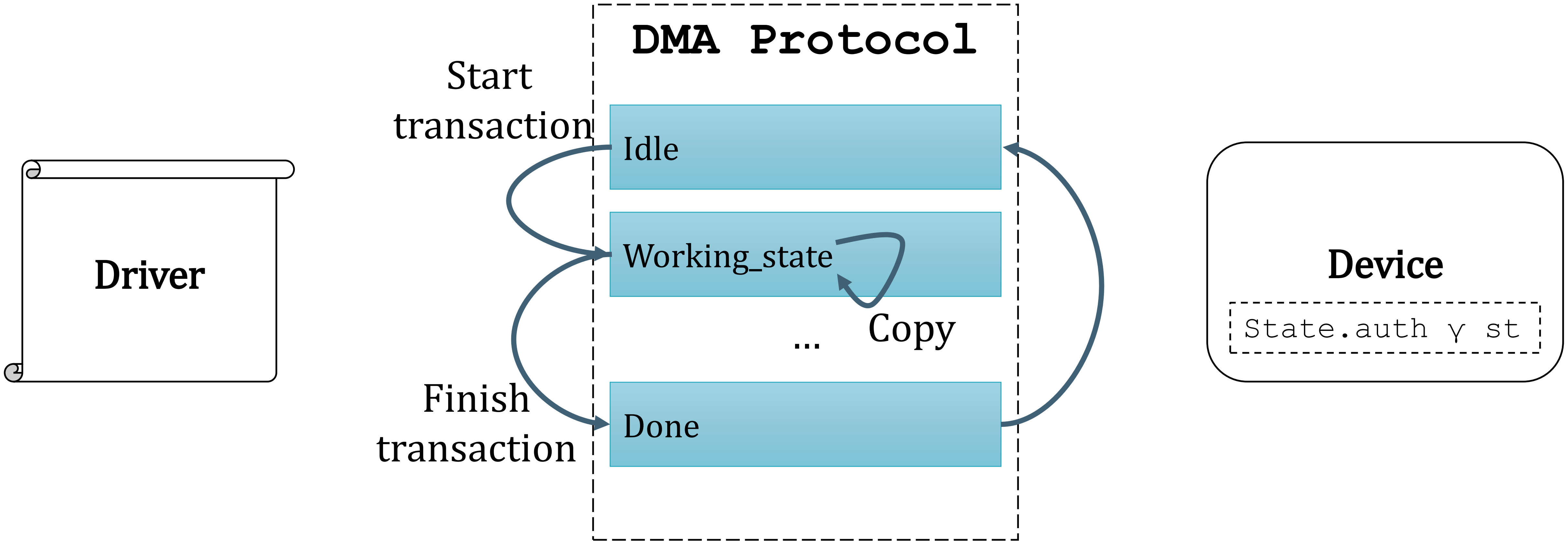


PL011: Interpretation of `Steady_state`

```
Definition Steady_state (γ : Chardev_model dev)
  : device_state -d> mpredO := fun st =>
 [| Initialized st |] **
 ...
 (Exists l : leibnizO (list N),
   [| l ++ fmap trimN8 (filter (negb ∘ receive_error)
                                (Queue.list $ rxq st))
    = in_trace _ st |] **
    own γ.(_intrace) (A:=excl_authR (leibnizO (list N)))
      (excl_auth_auth l)) **
...
```

Characters received so far, up to transmission errors and characters still in RX queue

ZynqMP DMA Protocol



Working_state

Protocol phase corresponding to "currently performing a copy"

```
Definition wp_copy
 (c : copy) (* Copy continuation *)
 (sptr dptr : ptr) (* Ptr start of src/dest arrays *)
 (l : list N) (* Data to be copied *)
 (q : Qp) (* Ownership of src transferred to device *)
: mpred :=
  let s := c.(src_base) in
  let d := c.(dst_base) in
  …
  let to_write := (sizen - num_written)%N in
  pinned_bytes size s sptr ** pinned_bytes size d dptr **
  [| c.(len) = N.of_nat (List.length l) |] **
  [| (s <= c.(src) /\ c.(src) <= s + sizen)%N |] **
  [| (d <= c.(dst) /\ c.(dst) <= d + sizen)%N |] **
  [| l = take (N.to_nat num_written) l ++
     c.(data) ++
     drop (N.to_nat num_read) l |] **
  [| l = map (fun n => (n `mod` 2 ^ 8)%N) l |] **
  sptr |-> bytesR q l **
  dptr |-> bytesR 1 (take (N.to_nat num_written) l) **
  dptr .[ T_uchar ! num_written ] |-> anyR (Tarray T_uchar to_write) 1.
```

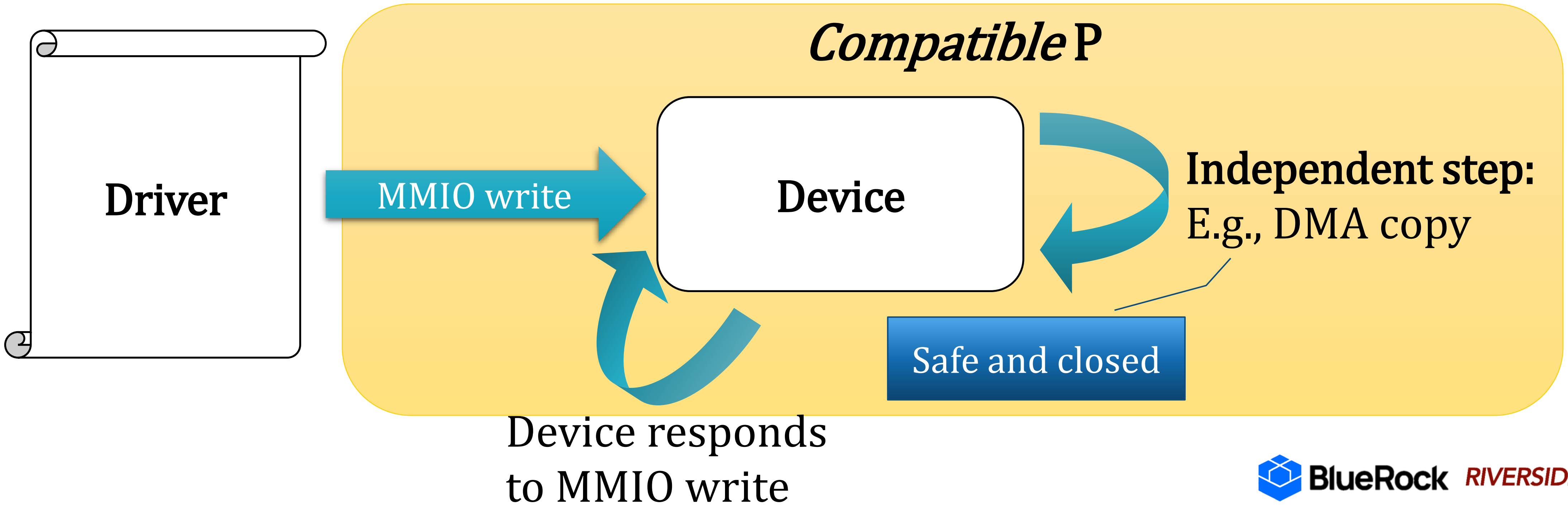
`l` is overall list of bytes to be copied

Device is somewhere in the middle of a copy

Protocols cannot block devices

```
Definition state_inv γ (ns : namespace) : mpred :=
 inv ns (Exists (P : T) (st : state),
         Proto.frag γ.(_proto) (1/2) P **
         interp P st **
         State.frag γ.(_state) st).
```

Arbitrary P could constrain the device state in ways that are *incompatible* with steps device takes independently of driver.



BlueRock RIVERSIDE RESEARCH

Protocols cannot block devices

```
Definition state_inv γ (ns : namespace) : mpred :=
 inv ns (Exists (P : T) (st : state),
          Proto.frag γ.(_proto) (1/2) P **
          interp P st **
          State.frag γ.(_state) st).
```

Constrain the current protocol to be **Compatible** with the device

```
Definition compatible γ (ns : namespace) :=
 inv ns (Exists (P : T),
   Proto.auth γ.(_proto) P **
   Compatible (interp P)).
```

```
Compatible (I : state -> mpred) : mpred :=
 (□ (Forall st, I st -* wp_step st I))%I.
```

I is closed and safe for device steps

Weakest precondition predicate transformer for device semantics

PL011 read_byte

```
Definition read_byte_spec (this : ptr) : WpSpec_cpp :=
 \with γ q before
 \arg{out} "out" (Vptr out)
 \arg{timeout_tsc} "timeout_tsc" (Vn timeout_tsc)
 \prepost this |-> chardev_drvR γ q
 \pre In_trace γ before
 \pre out |-> anyR T_uchar 1
 \post{r}[Verrno r]
 if decide (r = ENONE) then
   Exists c, In_trace γ (before ++ [c]) ** out |-> uint8R 1 c
 else In_trace γ before ** out |-> anyR T_uchar 1.
```

No error → Input trace extension

Simple-mode ZynqMP DMA

```
Definition memcopy_spec_ (this : ptr) : WithPrePost _ :=
 \with γ_proto γ γ_dma sptr dptr q l
 \arg{dest_pa} "dest_pa" (Vn dest_pa)
 \arg{source_pa} "source_pa" (Vn source_pa)
 \arg "len" (Vn (N.of_nat (List.length l)))
 \arg{timeout_tsc} "timeout_tsc" (Vn timeout_tsc)
 \require 0 < List.length l
 \require bound W32 Unsigned (Z.of_nat (List.length l))
 \prepost this |-> ZynqdmaR (Proto_idle γ) γ_proto γ_dma proto_full
 \prepost driver_regs γ
 \prepost pinned_bytes (List.length l) source_pa sptr
 \prepost pinned_bytes (List.length l) dest_pa dptr
 \prepost sptr |-> bytesR q l
 \pre dptr |-> anyR (Tarray (Tint char_bits Unsigned) (N.of_nat (length l))) 1
 \post{r}[Verrno r]
  if decide (r = ENONE)
  then dptr |-> bytesR 1 l
  else dptr |-> anyR (Tarray (Tint char_bits Unsigned) (N.of_nat (length l))) 1.
```

Device starts and ends in `Proto_idle` state

Bytes `l` copied to destination buffer

```
Definition memcopy_spec := [CHECK]
 SPECIFY (exact "_ZN10Zynqmp_dma7memcopyEmmmy") memcopy_spec_.
```

BlueRock RIVERSIDE RESEARCH

seL4 Connections

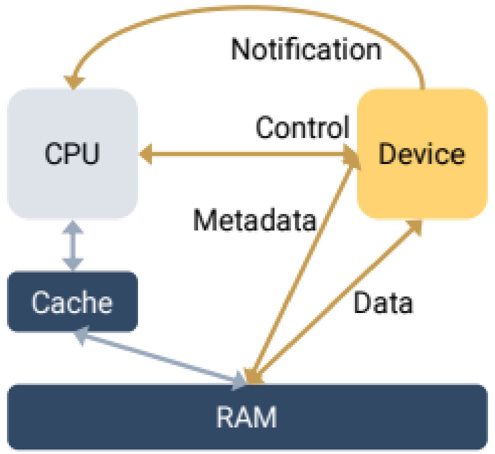
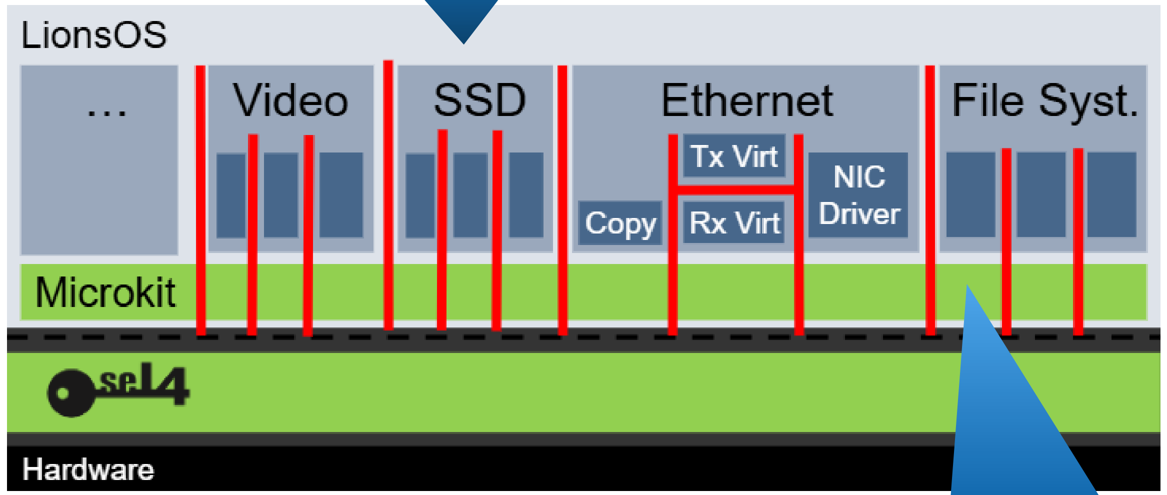


Figure 3.1: Device model.

https://trustworthy.systems/projects/drivers/sddf-design.pdf

Share device models?



https://lionsos.org/

Microkit interfaces to support pluggable proof engines?

E.g., device drivers proved in concurrent separation logic

BlueRock RIVERSIDE RESEARCH

Thanks!

Gordon Stewart
Riverside Research
gstewart@riversideresearch.org

Gregory Malecha
BlueRock Security
gregory@bluerock.io