CERTIFIED INTERRUPTIBLE OS KERNELS AND DEVICE DRIVERS

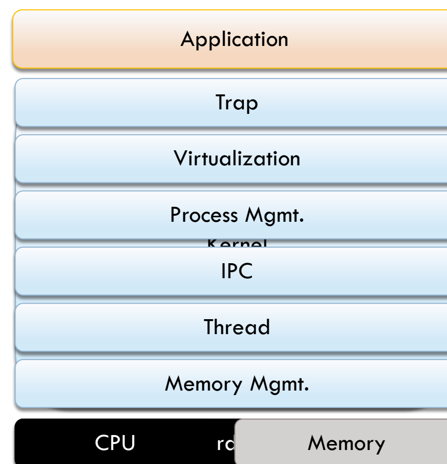
Zhong Shao
Joint work with Hao Chen, Newman Wu, Joshua Lockerman, and Ronghui Gu
Yale University

CPSC 422 Operating Systems October 15, 2020

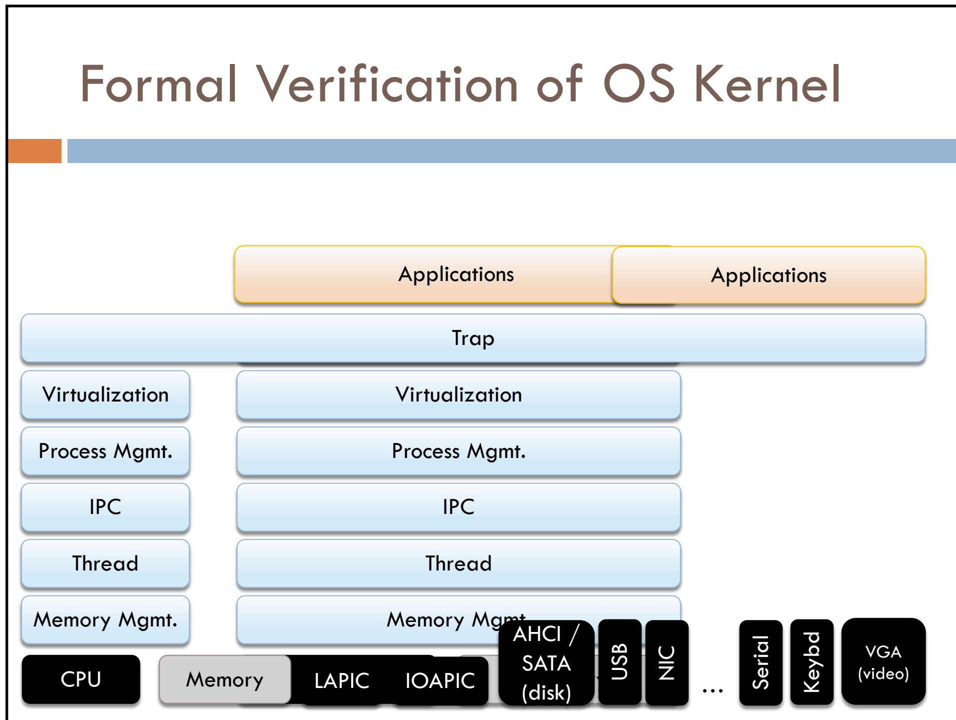
1

Formal Verification of OS Kernel

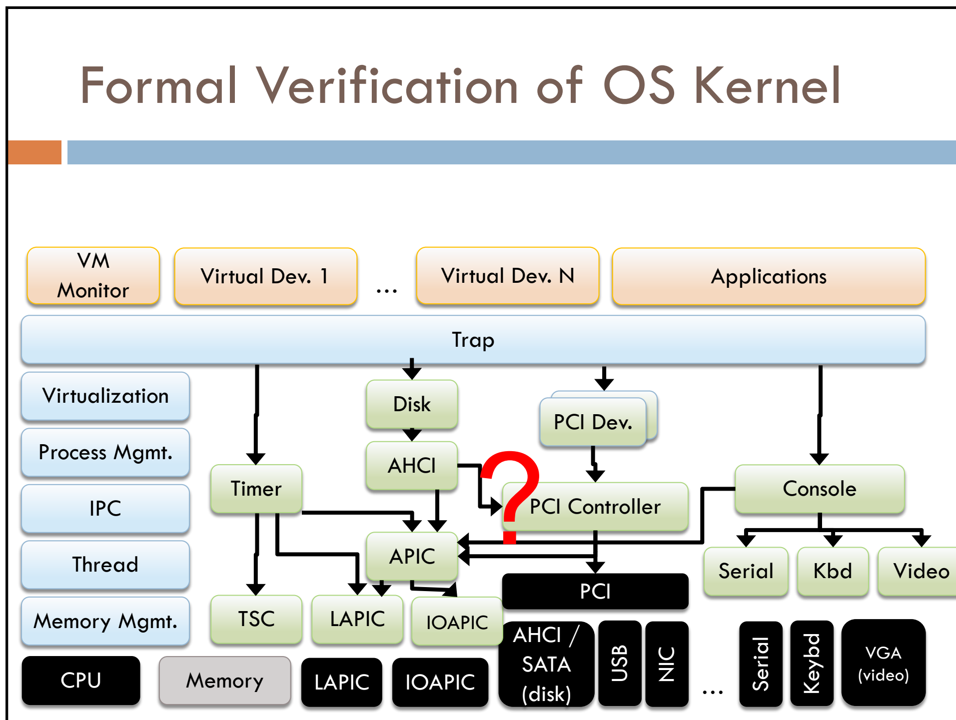
- seL4
- CertiKOS
- Verve



2



3



4

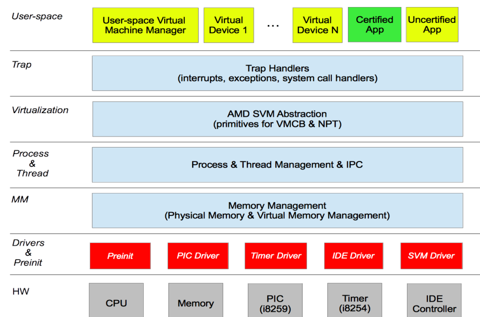
Device Drivers in Mainstream OS

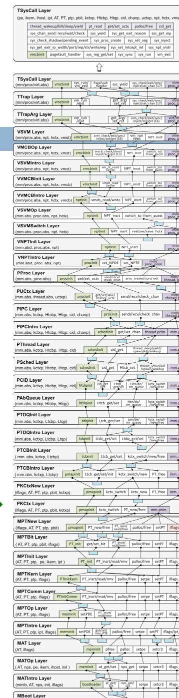
- 70% of Linux 2.4.1 kernel are device drivers.
- 70% of Windows crash are caused by third-party driver code.

5

mCertiKOS Overview [POPL'15]

- Single-core version of CertiKOS.
- 3k LOC, can boot Linux as guest.
- Aggressive use of abstraction over deep specification (37 layers).

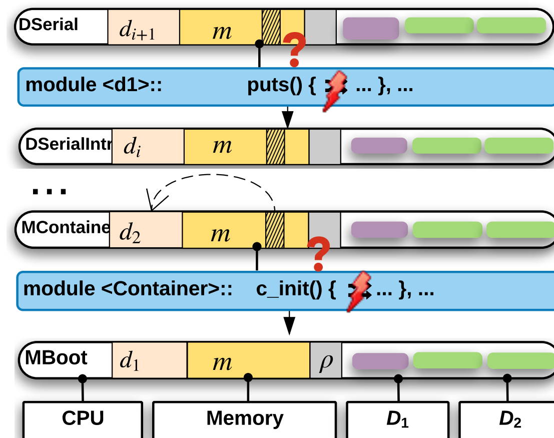




6

Main Challenge

Every fine-grained processor step could be interrupted.



7

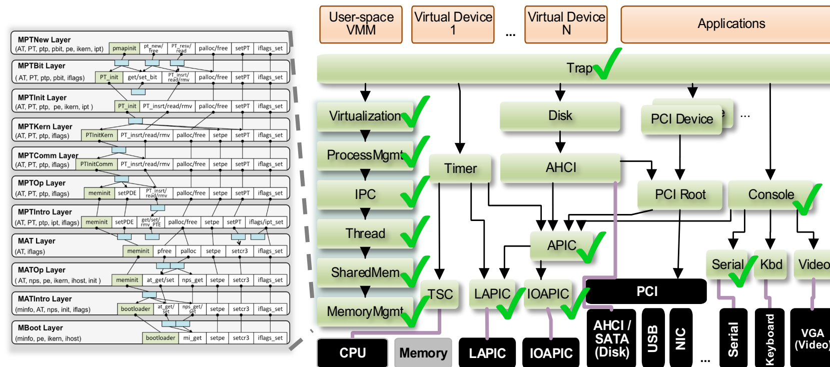
Other Challenges

- Interrupt hardware can be *dynamically* configured.
- Devices and CPU run in *parallel*.
- Device drivers are written in both C and *assembly*.
- The correctness results of different components should be *linked formally*.

8

Our Contributions [PLDI'16]

The *first* formally verified *interruptible* OS kernel with *device drivers.*



9

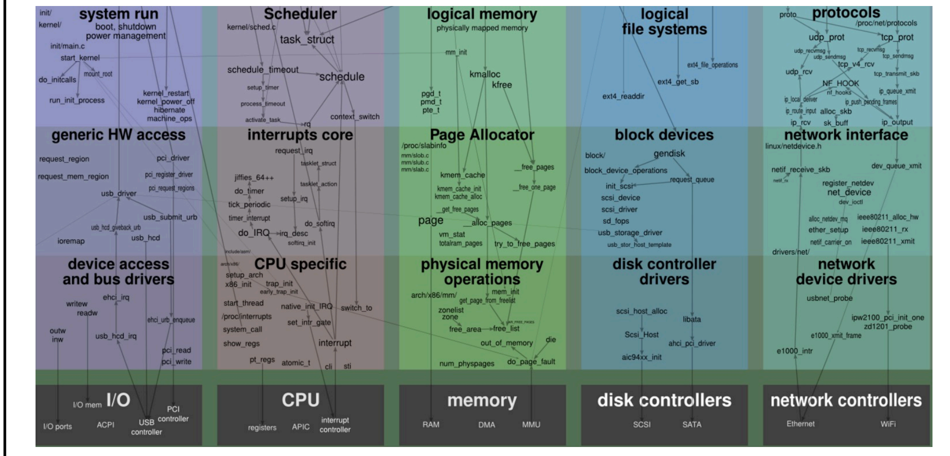
Our Contributions [PLDI'16]

- New techniques for certifying abstraction layers with multiple ***logical CPUs*** and devices.
- New techniques for building formal ***certified device hierarchies***.
- An abstraction-layer-based approach for reasoning about ***interrupts***.
- ***Case study***: interruptible mCertiKOS with device drivers.

10

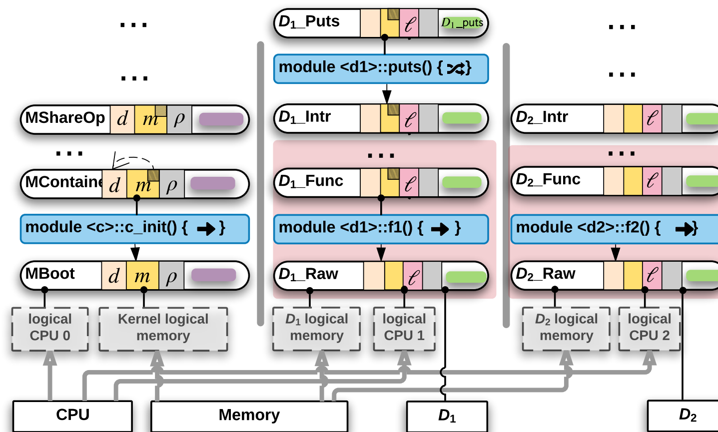
Linux Kernel Map

Kernel components are sorted into different stacks of abstraction layers based on their underlying hardware device.



11

New Machine Model



12

MShareOp d m ρ
...
MContain d m ρ
module <c>::c_init() { }
MBoot d m ρ
logical CPU 0
Kernel logical memory

D_1_Intr ℓ
...
D_1_Func ℓ
module <d1>::f1() { }
D_1_Raw ℓ
D_1 logical memory
logical CPU 1

D_2_Intr ℓ
...
D_2_Func ℓ
module <d2>::f2() { }
D_2_Raw ℓ
D_2 logical memory
logical CPU 2

Legend

Hardware
logical hw
module
Context
Layer
States
primitive
critical area
link
interleave
sequential
implement
use
refine
logical separation

δ^{env} intr_handler p_i
arrival
intr_disable
Primitive called

$intr_{CPU}$ sti cli iret
$intr_{IC}$ eoi mask unmask
intr_handler p_i
arrival
mask Primitive called

$\sqsubseteq$

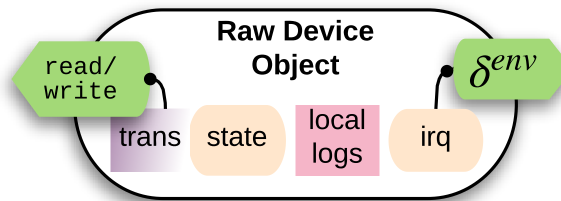
al events.

δ^{CPU}

14

Raw Device Object

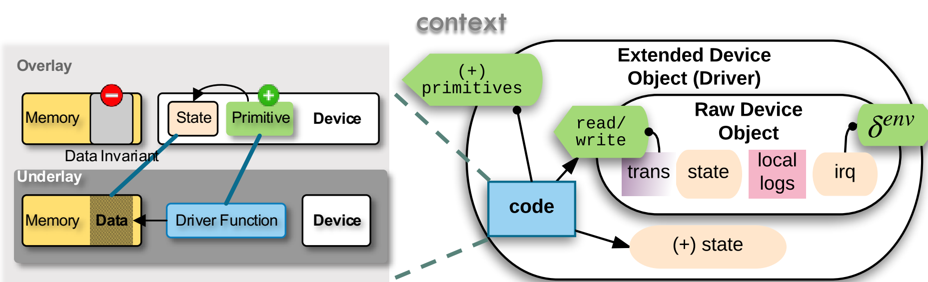
- Local log for the list of observed external events.
- Multiple local logs to handle disjoint set of external events asynchronously.
- Read/Write instructions: IN/OUT, memory mapped I/O, *etc*.



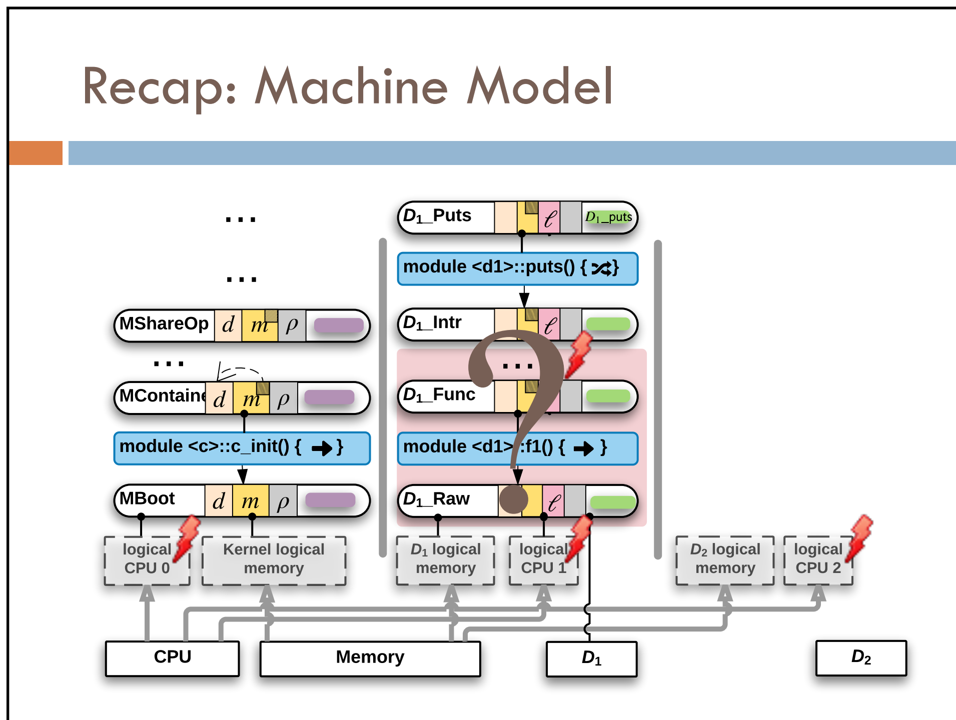
15

Extended Device Object

Driver as a logical device.



16

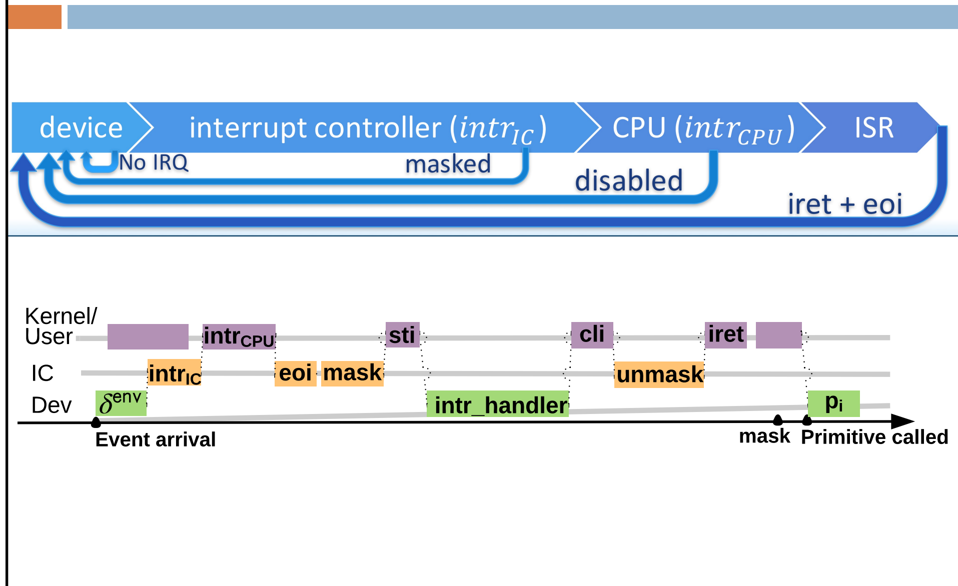


17

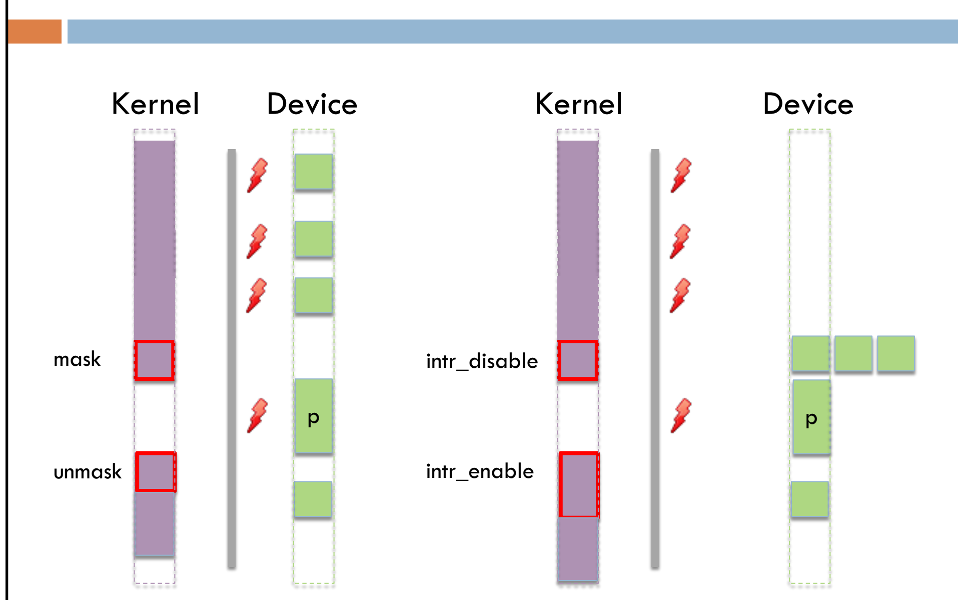
Our Contributions

- New techniques for certifying abstraction layers with multiple ***logical CPUs*** and devices.
- New techniques for building formal ***certified device hierarchies***.
- An abstraction-layer-based approach for reasoning about ***interrupts***.
- ***Case study***: interruptible mCertiKOS with device drivers.

18



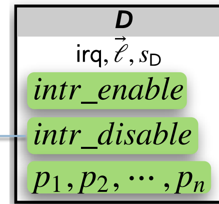
19



20

Semantics of *intr_disable*

- Scans external events.
- Recursively performs the environmental transition.
- Synchronizes unhandled interrupts.



details in the paper

DisableNoIntr: Disable with no unhandled interrupt

$$\frac{(e, \ell_i') = \mathsf{next}(\ell^{env}, \ell_i) \qquad s_{\mathsf{tmp}} = \delta^{\mathsf{env}}(s, e) \qquad s_{\mathsf{tmp}}.irq = \mathsf{false} \qquad s' = s[\mathsf{iFlag} \leftarrow 0]}{\mathtt{intr_disable}(s, \ell_i, \ell^{env}) = (s', \ell_i)}$$

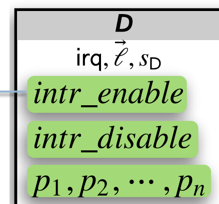
DisableIntr: Disable with unhandled interrupts

$$\frac{(e, \ell_i') = \mathsf{next}(\ell^{env}, \ell_i) \qquad s' = \delta^{\mathsf{env}}(s, e) \qquad s'.irq = \mathsf{true} \qquad (s'', \ell_i'') = \mathtt{intr_handler}(s', \ell_i', \ell^{env}) \qquad (s''', \ell_i''') = \mathtt{intr_disable}(s'', \ell_i'', \ell^{env})}{\mathtt{intr_disable}(s, \ell_i, \ell^{env}) = (s''', \ell_i''')}$$

21

Semantics of *intr_enable*

- Recursively discharges pending interrupts.
- Delayed interrupts that occur while the interrupt is disabled.



details in the paper

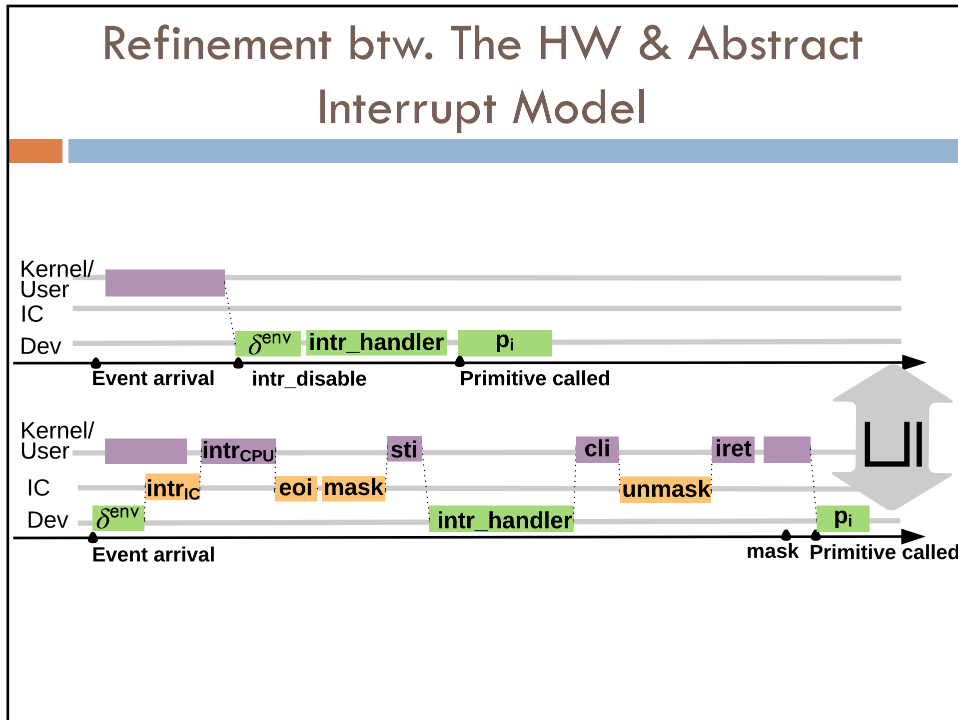
EnableNoIntr: Enable with no pending interrupt

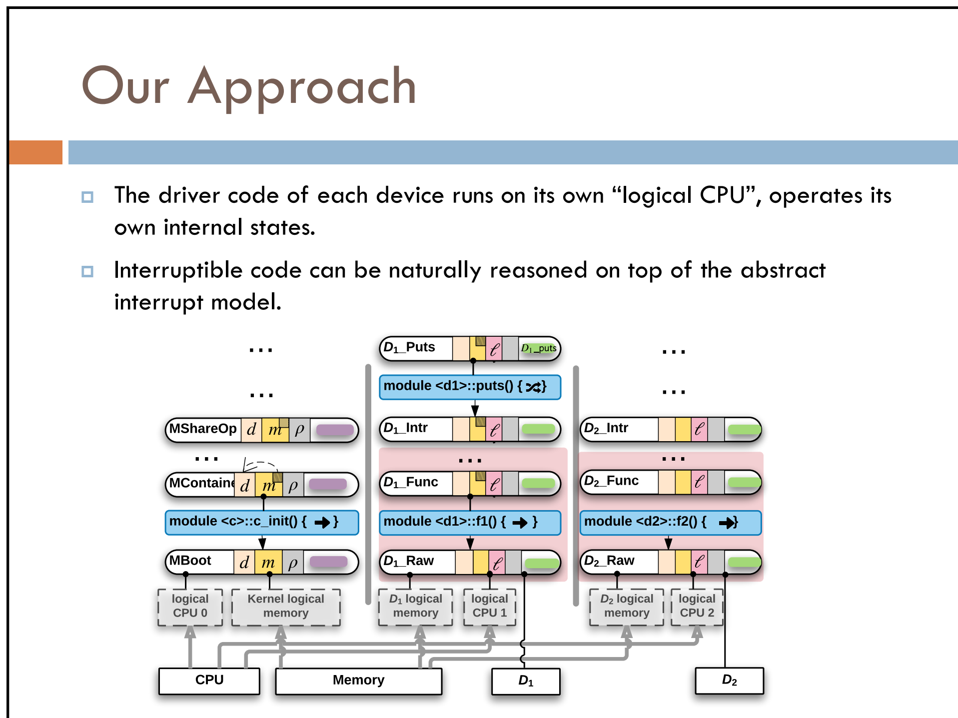
$$\frac{s.irq = \mathsf{false} \qquad s' = s[\mathsf{iFlag} \leftarrow 1]}{\mathtt{intr_enable}(s, \ell_i, \ell^{env}) = (s', \ell_i)}$$

EnableIntr: Enable with pending interrupts

$$\frac{s.irq = \mathsf{true} \qquad (s', \ell_i') = \mathtt{intr_handler}(s, \ell_i, \ell^{env}) \qquad (s'', \ell_i'') = \mathtt{intr_enable}(s', \ell_i', \ell^{env})}{\mathtt{intr_enable}(s, \ell_i, \ell^{env}) = (s'', \ell_i'')}$$

22


Refinement btw. The HW & Abstract Interrupt Model
Kernel/User
IC
Dev
δ^{env}
intr_handler
p_i
Event arrival
intr_disable
Primitive called
Kernel/User
$intr_{CPU}$
sti
cli
iret
IC
$intr_{IC}$
eoi
mask
unmask
Dev
δ^{env}
intr_handler
p_i
Event arrival
mask
Primitive called
⊑
23

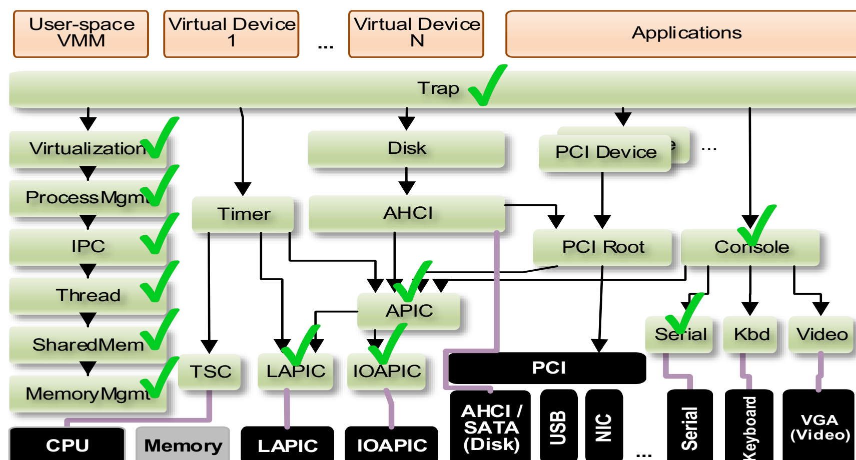

Our Approach
The driver code of each device runs on its own "logical CPU", operates its own internal states.
Interruptible code can be naturally reasoned on top of the abstract interrupt model.
D_1_Puts
module <d1>::puts() {
D_1_Intr
D_2_Intr
MShareOp
MContainer
D_1_Func
D_2_Func
module <c>::c_init() {
module <d1>::f1() {
module <d2>::f2() {
MBoot
D_1_Raw
D_2_Raw
logical CPU 0
Kernel logical memory
D_1 logical memory
logical CPU 1
D_2 logical memory
logical CPU 2
CPU
Memory
D_1
D_2
24

Our Contributions

- New techniques for certifying abstraction layers with multiple ***logical CPUs*** and devices.
- New techniques for building formal ***certified device hierarchies***.
- An abstraction-layer-based approach for reasoning about ***interrupts***.
- ***Case study***: interruptible mCertiKOS with device drivers.

25

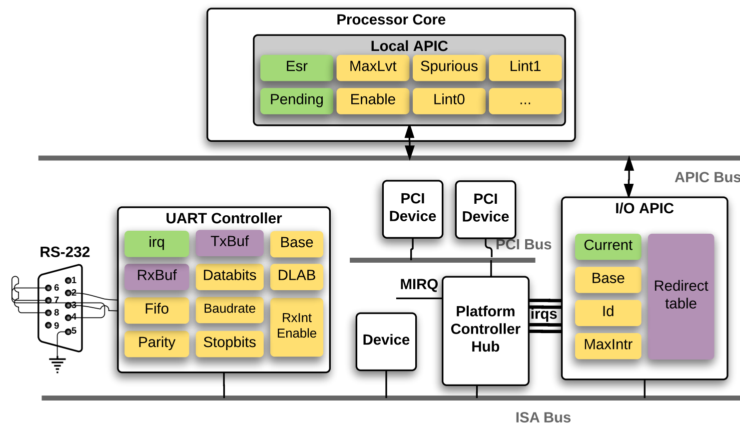
Interruptible mCertiKOS with Drivers



26

Case Study: Modeling HW Devices

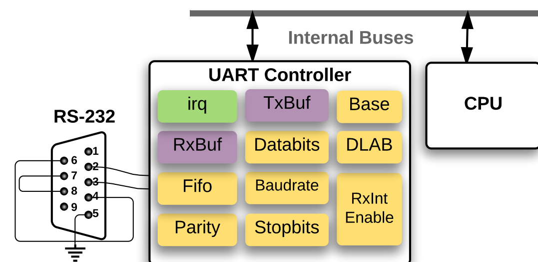
- Serial Port, I/O APIC, Local APIC, CPU interrupt handling.



27

Case Study: Serial Device

- States: see figure
- Transitions: serial_trans_env + serial_trans_IO
- Read/Write primitives: serial_read / serial_write



28

Serial Interrupt Handler

serial_intr

serial_getc

serial_getc

```
1 unsigned int serial_getc () {
```



```
1 void serial_puts(char * s, int len) {
2   int i = 0;
3   while (i < len && s[i] != 0) {
4     serial_intr_disable ();
5     serial_putc (s[i]);
6     serial_intr_enable ();
7     i++;
8   }
9 }
```

serial_read

serial_write

What We Have Proved

- Total functional correctness.
- Safety.
- *Contextual refinement* between the lowest and the top level abstract machine:
  $\forall P,\ [\![K \bowtie P]\!]_{\mathrm{x86}} \sqsubseteq [\![P]\!]_{\mathrm{mCertiKOS}}$
- Data invariants:
  - Console's circular buffer is always well-formed.
  - Interrupt controller states are always consistent.
- The framework also ensures that:
  - No code injection attacks, buffer overflow, integer overflow, null pointer access, *etc.*

31

Size of TCB and Spec/Proof

- In the TCB
  - X86 hardware model
  - Hardware device/interrupt model (510 LOC)
  - System call specification (126 LOC)
  - Bootloader
  - Coq proof checker
  - Pretty-printing phase of the CompCert compiler
- Rest of the spec/proof (about 20k LOC)
  - Intermediate and auxiliary specifications and definitions
  - Coq proof scripts

32

Conclusion

- Compositional framework for building certified interruptible kernel with device drivers.
  - Certified abstraction layers with multiple logical CPUs.
  - An abstraction-layer-based approach for expressing interrupts.
- The first formally verified interruptible OS kernel with device drivers.
- Extensions:
  - Other drivers
  - Concurrency
  - Larger kernel