NICTA

Low-Overhead Virtualization of Mobile Systems

Gernot Heiser

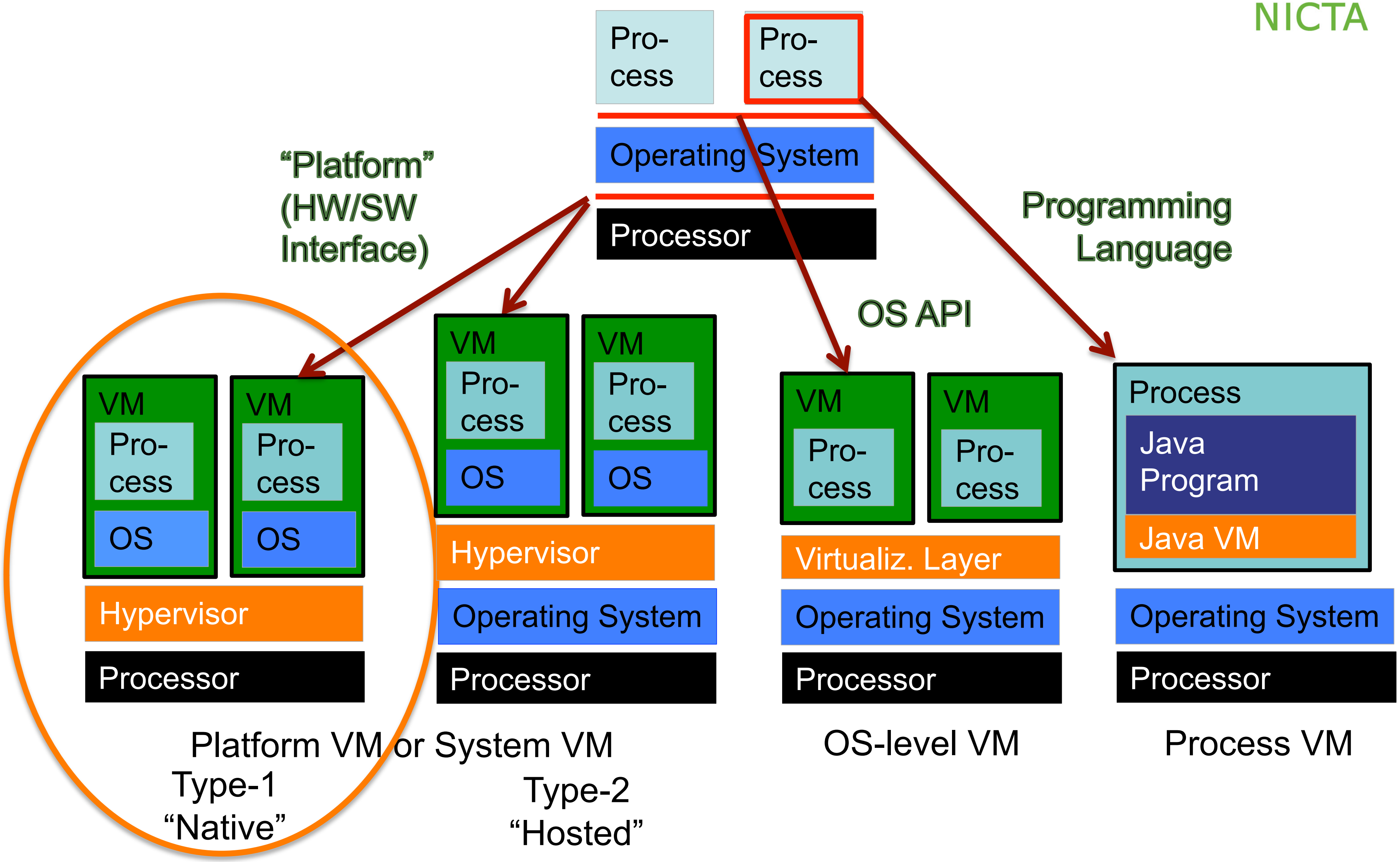
NICTA and University of New South Wales
Sydney, Australia

Australian Government
Department of Broadband, Communications and the Digital Economy
Australian Research Council

NICTA Funding and Supporting Members and Partners

Australian National University
UNSW THE UNIVERSITY OF NEW SOUTH WALES
NSW GOVERNMENT
ACT GOVERNMENT
State Government Victoria
THE UNIVERSITY OF MELBOURNE
THE UNIVERSITY OF SYDNEY
Queensland Government
Griffith UNIVERSITY
QUT Queensland University of Technology
THE UNIVERSITY OF QUEENSLAND AUSTRALIA

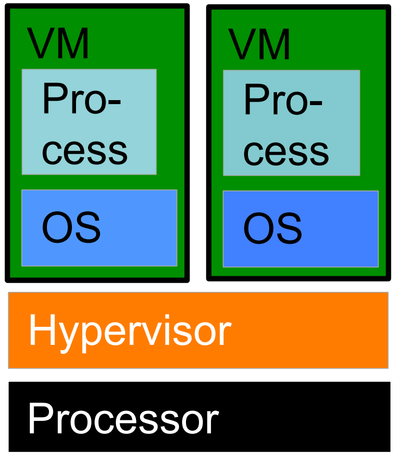
Types of Virtualization



Why Virtual Machines?

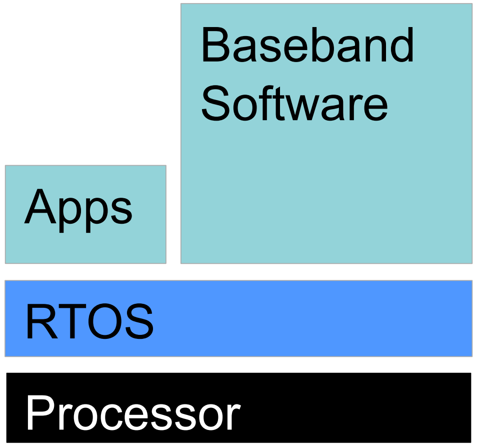
Traditional (enterprise) uses:

- Server consolidation
  - Hardware & energy savings with QoS isolation
  - Migrating, checkpointing, debugging
  - Concurrent use of multiple OSes
    - … or OS versions
- Security
  - Partitioning to limit reach of intrusions
  - Sandboxing untrusted apps

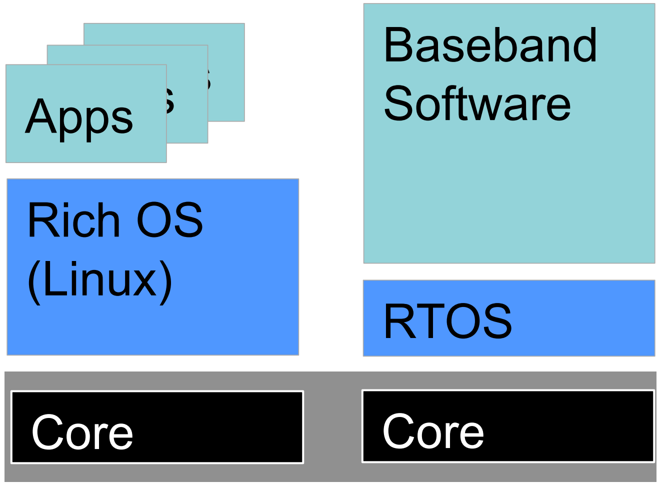


Virtualizing mobile systems – crazy idea?

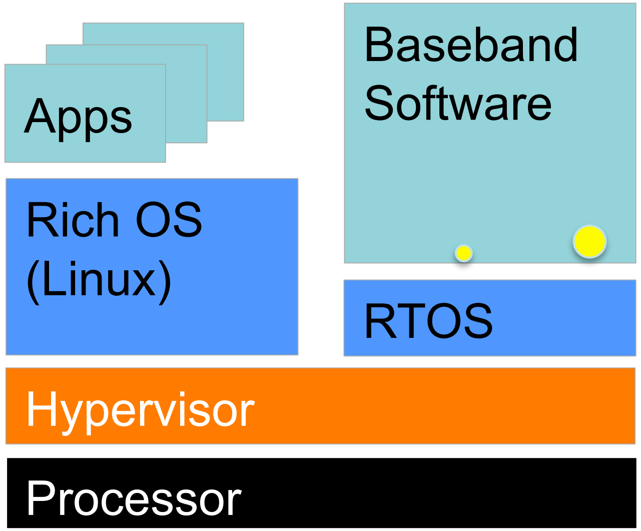
Mobile Phones



Dumb phone



Smartphone

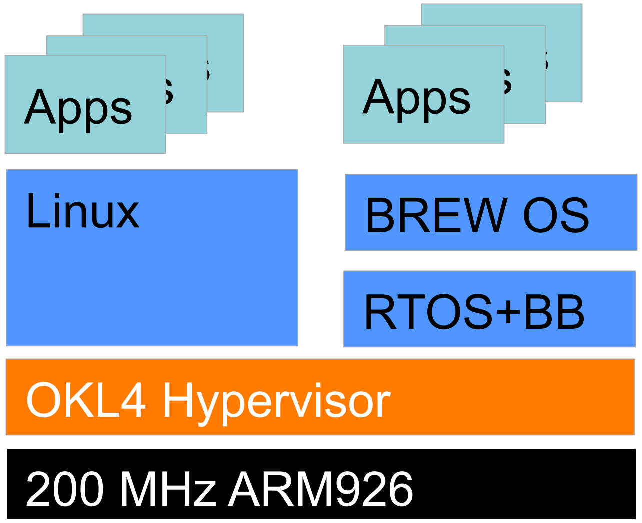


Consolidated phone

Heterogenous Operating Systems!

Consolidated Phone: Motorola Evoke

- Linux+BREW OS
- Linux+BREW apps
- Seamless UI integration
- Released April 2009



Apps

Apps

Linux

BREW OS

RTOS+BB

OKL4 Hypervisor

200 MHz ARM926

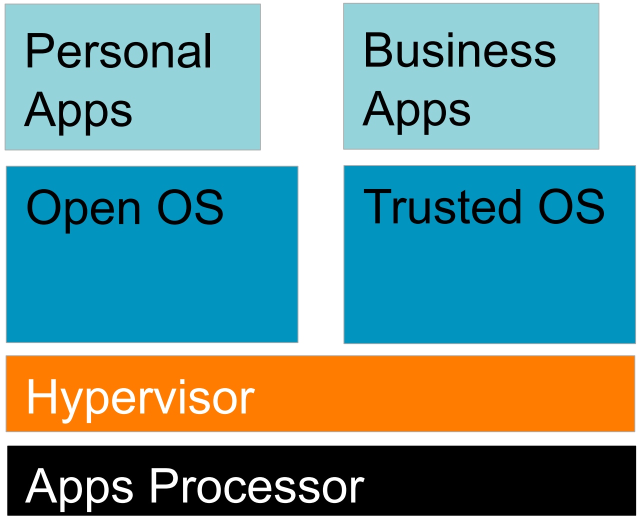
Dual-Persona Smartphone

- Phones increasingly used to access business data
  - Companies lock down phones, no arbitrary apps
  - Employees end up carrying two phones

- Integrate two virtual phones into one physical
  - Locked-down business phone
  - Open personal phone

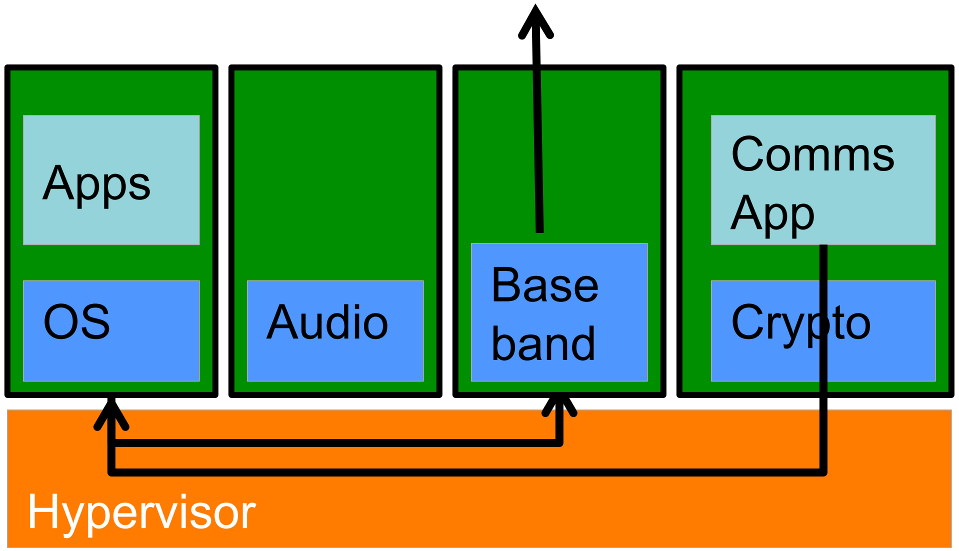
- Only one used at a time
  - Perfect use of virtualization

Will reach market soon



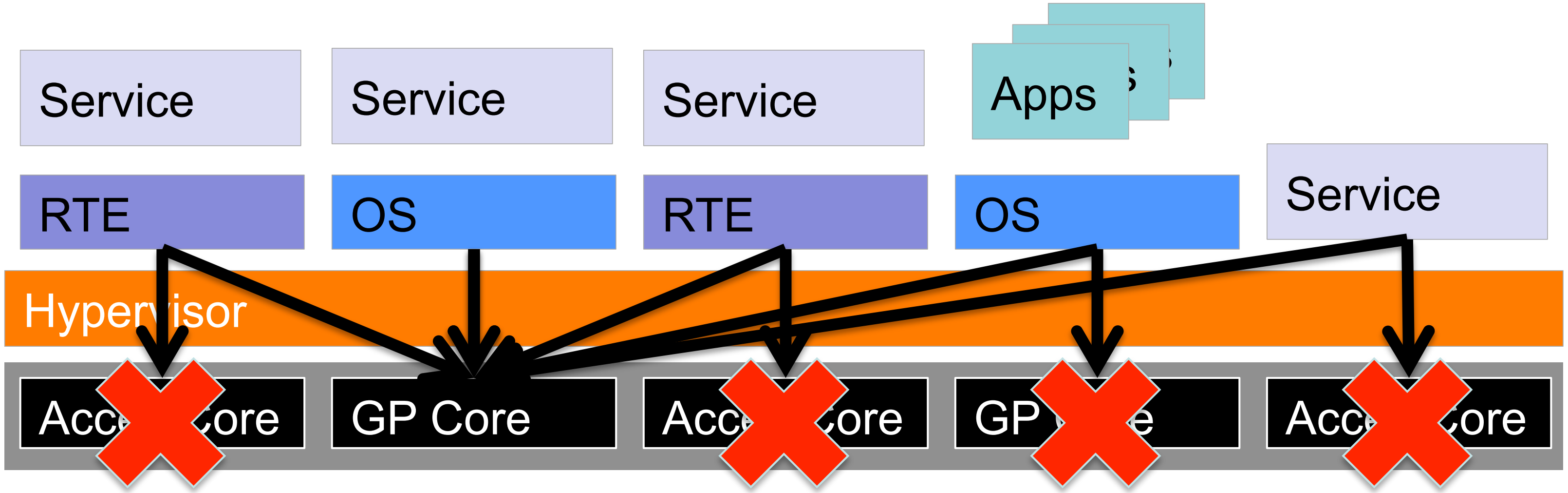
Secure Communication on COTS Phone

- Secure phones are expensive (small product runs)
  - Strong push for COTS devices in defence etc
- Use virtualization to provide secure communication on standard smartphone
  - Encrypt voice, data and tunnel through open OS
- Hypervisor guarantees isolation
  - With controlled communication
- Small *trusted computing base*



Presently under evaluation by various agencies

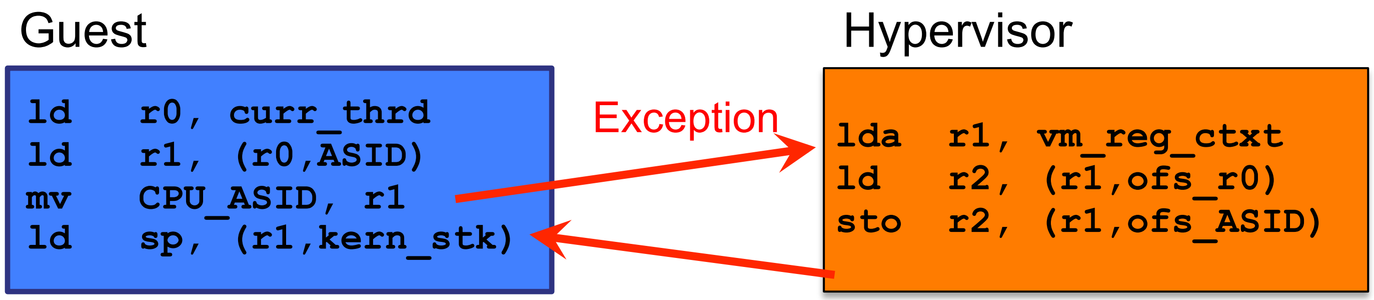
Energy Management in Future Devices



- Load-based dynamic re-mapping of activities to cores

Virtualization Mechanics: Instruction Emulation

- “Pure” virtualization: *Trap and emulate* approach:
  - Guest attempts to access physical resource
  - Hardware raises exception (trap), invoking hypervisor’s handler
  - Hypervisor emulates result, based on access to virtual resources
- Most instructions do not trap
  - Makes efficient virtualization possible
  - Requires that VM ISA is (almost) same as physical processor ISA
- Works as long as architecture is “virtualizable”:
  - All instructions exposing or modifiying physical resources must trap
  - Not the case e.g. for ARM



Guest

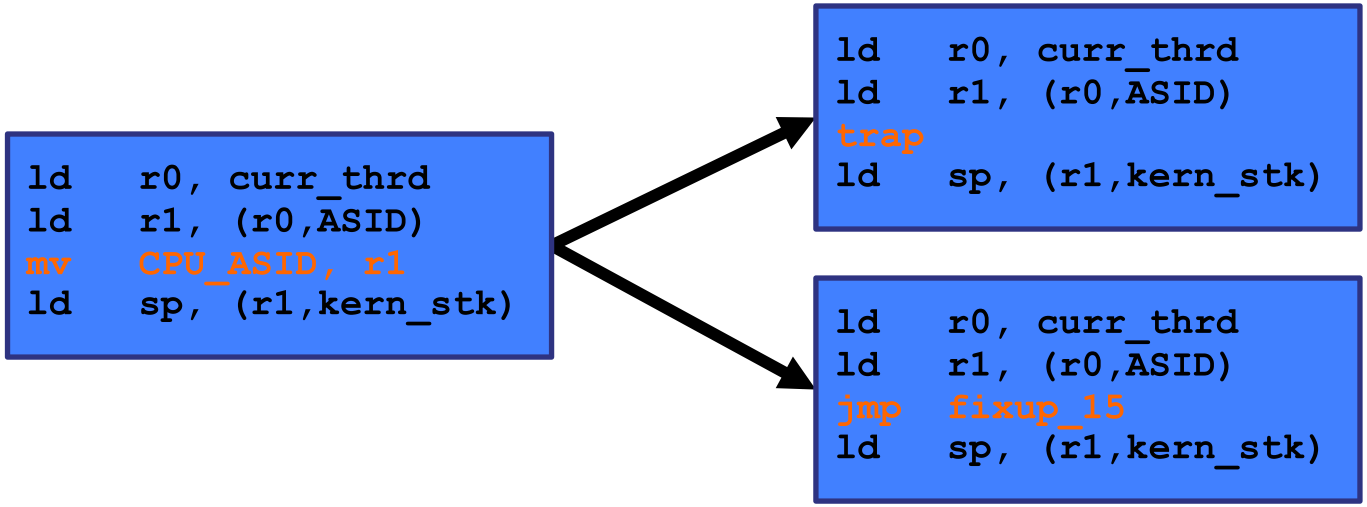
```
ld   r0, curr_thrd
ld   r1, (r0,ASID)
mv   CPU_ASID, r1
ld   sp, (r1,kern_stk)
```

Hypervisor

```
lda  r1, vm_reg_ctxt
ld   r2, (r1,ofs_r0)
sto  r2, (r1,ofs_ASID)
```

Para-Virtualization

- Manual modification of guest OS source
  - Port from hardware ISA to hypervisor API
    - Replace ISA instructions by trapping code (“hypercalls”)
  - Expensive in terms of engineering time (& error prone)
- Mandatory for non-virtualizable architecture (eg. ARM)
- Optionally for performance improvements
  - Minimise costly hypervisor entries
  - Amortize hypercall cost over many instructions



```
ld   r0, curr_thrd
ld   r1, (r0,ASID)
mv   CPU_ASID, r1
ld   sp, (r1,kern_stk)
```

```
ld   r0, curr_thrd
ld   r1, (r0,ASID)
trap
ld   sp, (r1,kern_stk)
```

```
ld   r0, curr_thrd
ld   r1, (r0,ASID)
jmp  fixup_15
ld   sp, (r1,kern_stk)
```

Minimising Overheads

- Hypervisor design and implementation is important
  - Para-virtualization requires well-designed API
    - Minimise hypervisor entries
  - Tight implementation as hypervisor is on critical path
    - Small cache footprint
    - “Fast paths” for optimising common case
    - Many processor-specific optimisations
  - Keeping it small helps:
    - 10 kLOC is much easier to optimise than 100 kLOC!

Overheads: lmbench Microbenchmarks

Benchmark	Native	Virtualized	Overhead		
null syscall	0.6 µs	0.96 µs	0.36 µs	180 cy	60 %
read	1.14 µs	1.31 µs	0.17 µs	85 cy	15 %
stat	4.73 µs	5.05 µs	0.32 µs	160 cy	7 %
fstat	1.58 µs	2.24 µs	0.66 µs	330 cy	42 %
open/close	9.12 µs	8.23 µs	-0.89 µs	-445 cy	-10 %
select(10)	2.62 µs	2.98 µs	0.36 µs	180 cy	14 %
sig handler	1.77 µs	2.05 µs	0.28 µs	140 cy	16 %
pipe latency	41.56 µs	54.45 µs	12.89 µs	6.4 kcy	31 %
UNIX socket	52.76 µs	80.90 µs	28.14 µs	14 kcy	53 %
fork	1,106 µs	1,190 µs	84 µs	42 kcy	8 %
fork+execve	4,710 µs	4,933 µs	223 µs	112 kcy	5 %
system	7,583 µs	7,796 µs	213 µs	107 kcy	3 %

OKL4 Microvisor on Beagle Board (500 MHz Cortex A8 ARMv7)

Overheads: Networking

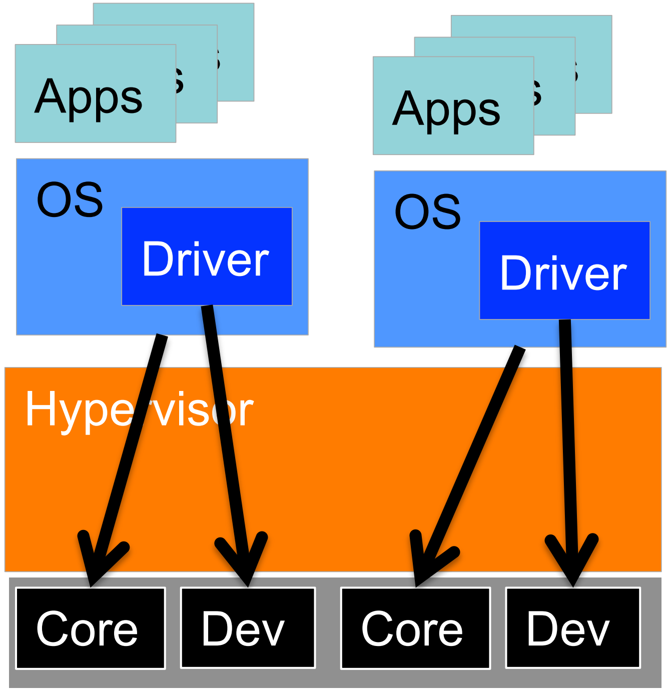
Netperf networking benchmark on Linux

Type	Measure	Native	Virtualized	Overhead
TCP	Throughput [Mib/s]	651	630	3 %
	CPU load [%]	99	99	0 %
	Cost [µs/KiB]	12.5	12.9	3 %
UDP	Throughput [Mib/s]	537	516	4 %
	CPU load [%]	99	99	0 %
	Cost [µs/KiB]	15.2	15.8	4 %

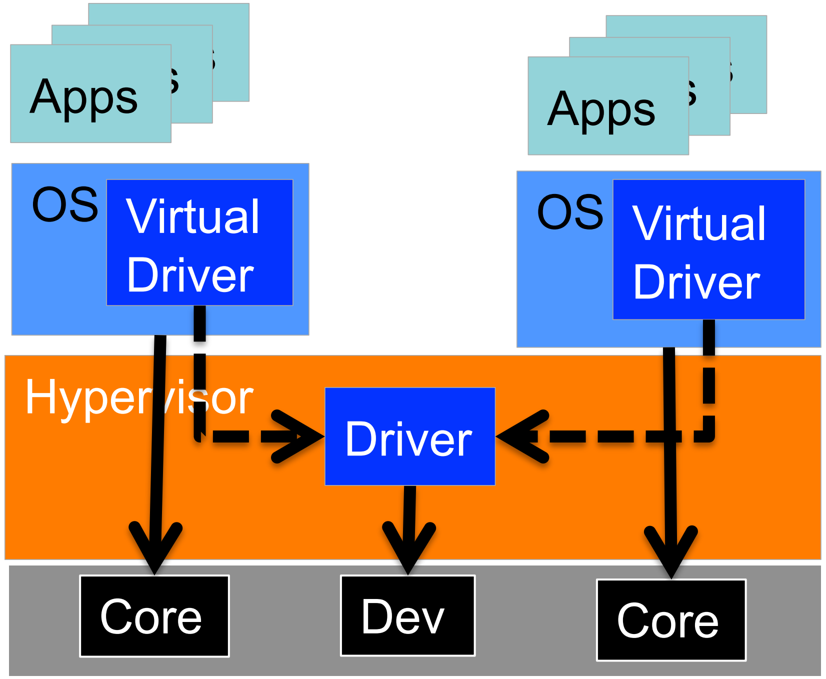
OKL4 Microvisor on Beagle Board (500 MHz Cortex A8 ARMv7)

VIrtualizing Devices: Two Possibilities

VM-Owned



Shared



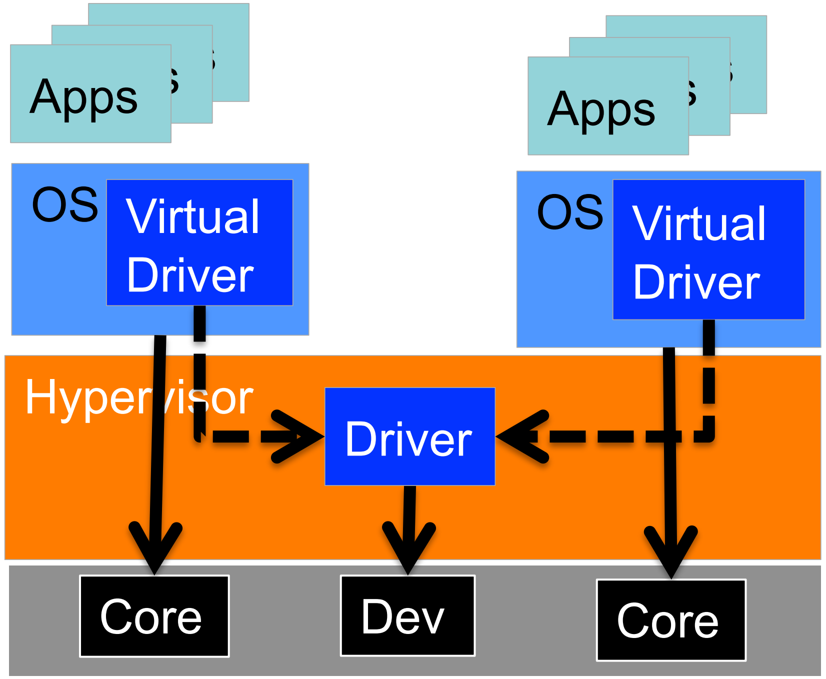
- Device regs exposed to VM
  - unmodified native guest driver accesses device directly

- Virtual device exposed to VM
  - Virtual driver communicates with real driver in hypervisor

Shared Devices: Pure vs Para-Virtualized

Pure: Unmodified guest driver

- Each device register access by guest driver traps to hypervisor
  - – real driver emulates
- Many traps – expensive!

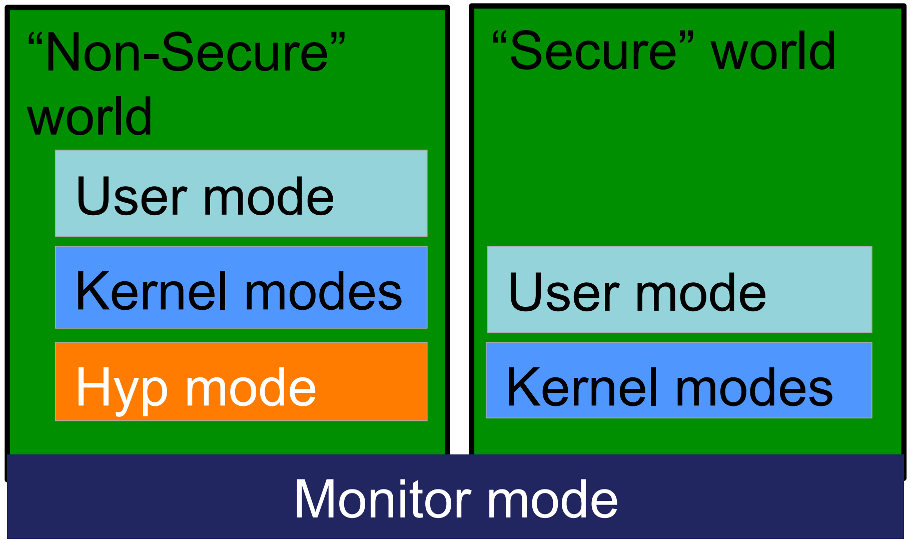


Para: Modified device API

- Virtual device is simplified
  - – possibly explicit driver communication API
  - – virtual driver is very simple
- Can dramatically reduce traps
- **But**: *need new driver*
  - – real driver ported to hypervisor
- Real driver can be
  - – inside hypervisor
  - – separate driver VM
    - one for all drivers
    - separate for each driver

Coming Up: Hardware Support

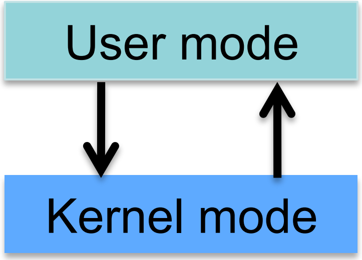
- ARMv7 virtualization extensions announced Q3/2010
- Anticipate Si samples in 2011, products in 2012
- Presently only simulator (**not** cycle accurate!)



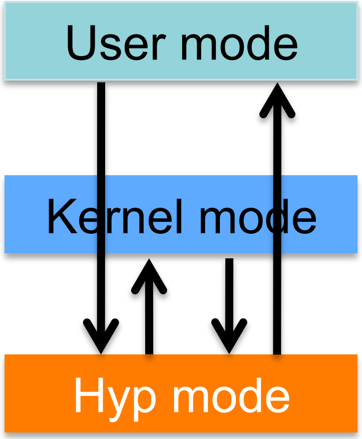
- New privilege level: hyp
  - Strictly higher than kernel
  - Virtualizes or traps *all* sensitive instructions
  - Only available in ARM TrustZone "non-secure" mode
- Note: different from x86
  - VT-x "root" mode is orthogonal to x86 protection rings

ARM Virtualization Extensions (1)

Configurable Traps

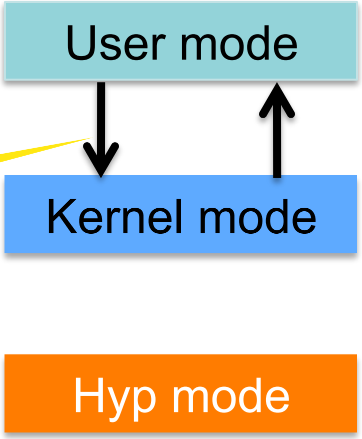


Native syscall



Virtual syscall

Can configure traps to go directly to guest OS

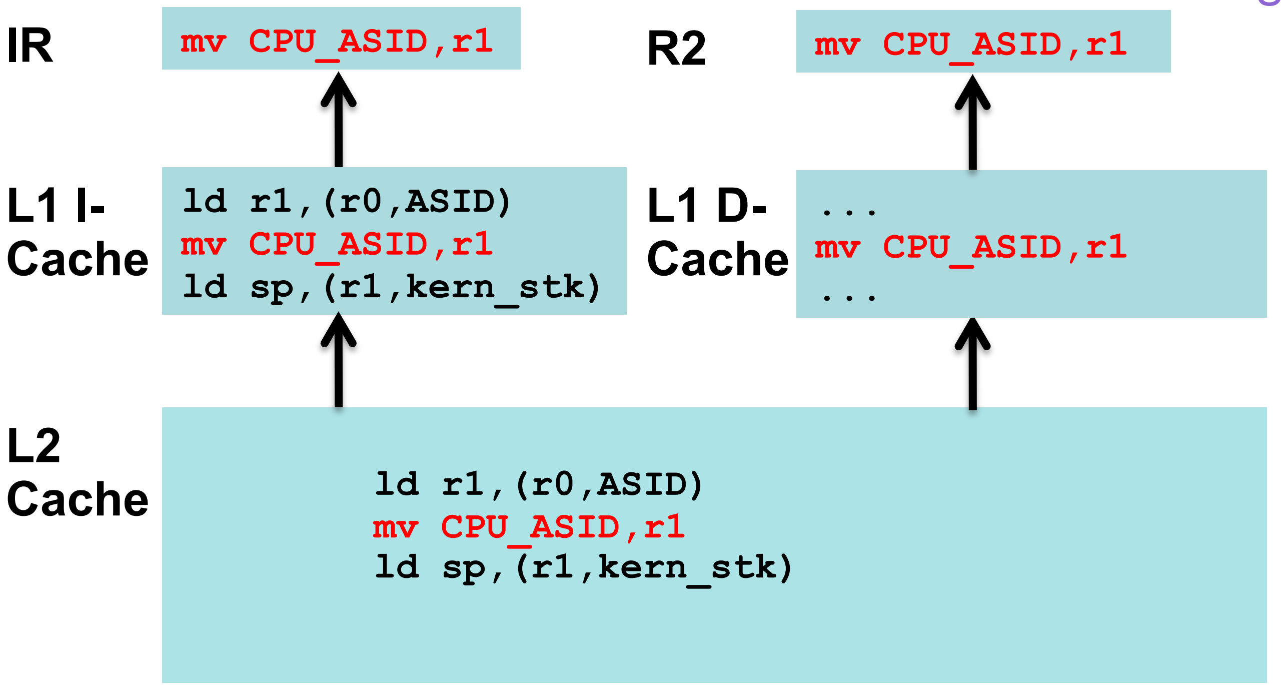


Virtual syscall
Trap to guest

ARM Virtualization Extensions (2)

Emulation

1) Load faulting instruction
   - Compulsory L1-D miss!
2) Decode instruction
   - Complex logic
3) Emulate instruction
   - Usually straightforward



IR

```
mv CPU_ASID,r1
```

R2

```
mv CPU_ASID,r1
```

L1 I-Cache

```
ld r1,(r0,ASID)
mv CPU_ASID,r1
ld sp,(r1,kern_stk)
```

L1 D-Cache

```
...
mv CPU_ASID,r1
...
```

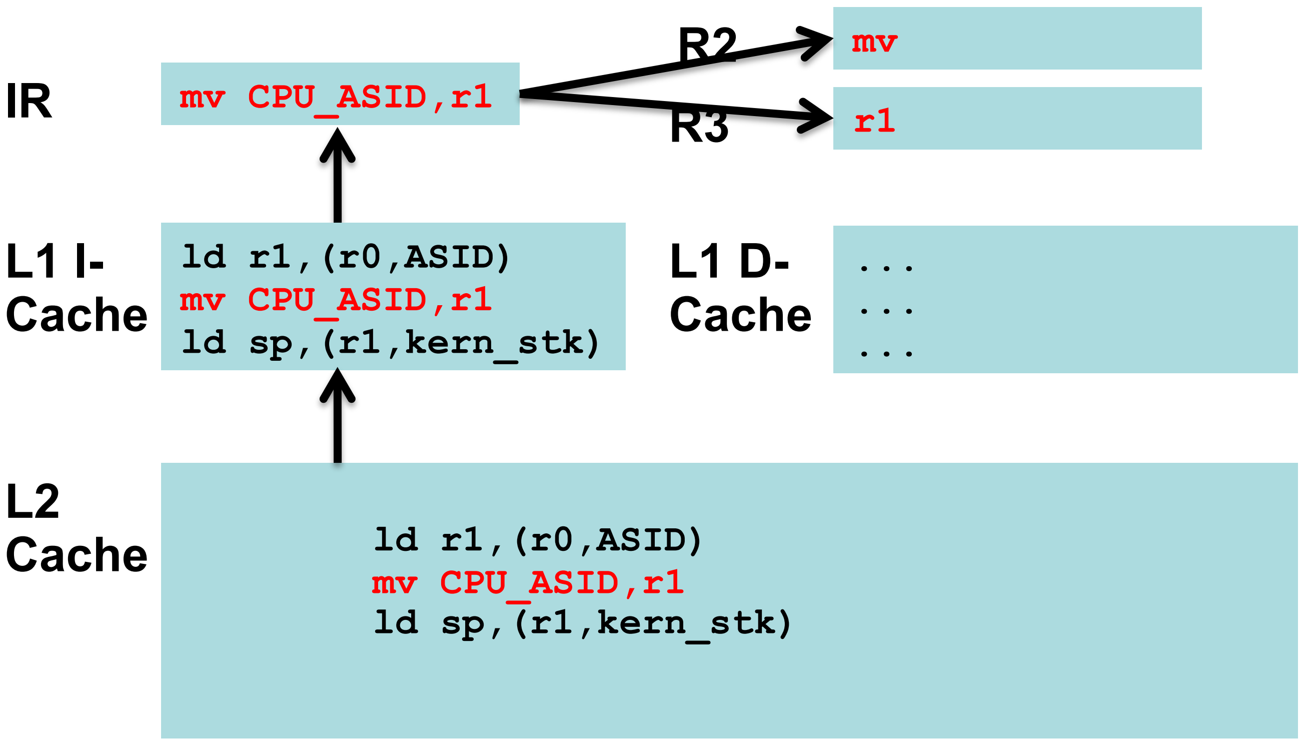
L2 Cache

```
ld r1,(r0,ASID)
mv CPU_ASID,r1
ld sp,(r1,kern_stk)
```

ARM Virtualization Extensions (2)

Emulation Support

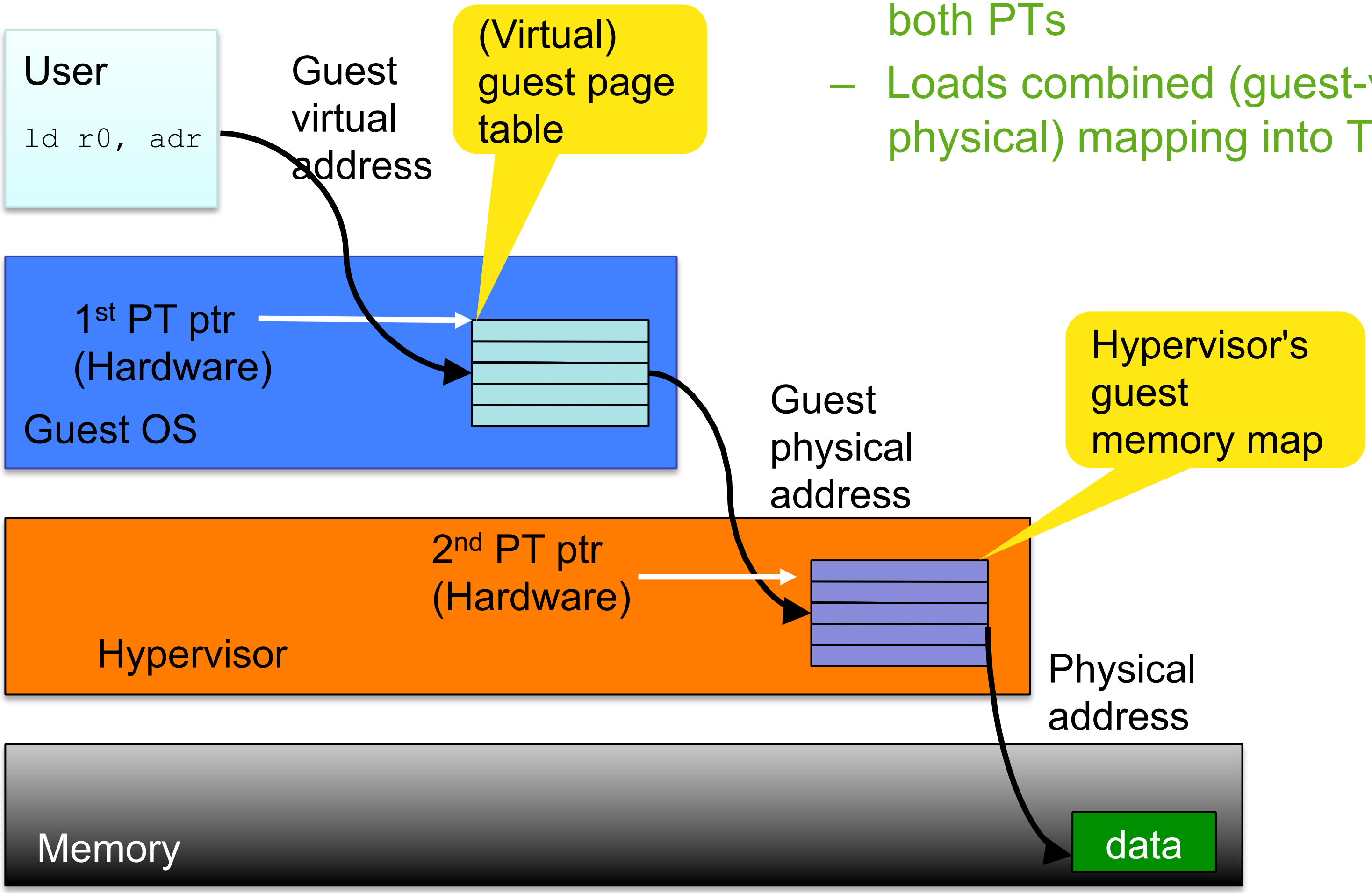
- HW decodes instruction
  - No L1 miss
  - No software decode
- SW emulates instruction
  - Usually straightforward



```
ld r1,(r0,ASID)
mv CPU_ASID,r1
ld sp,(r1,kern_stk)
```

ARM Virtualization Extensions (3)

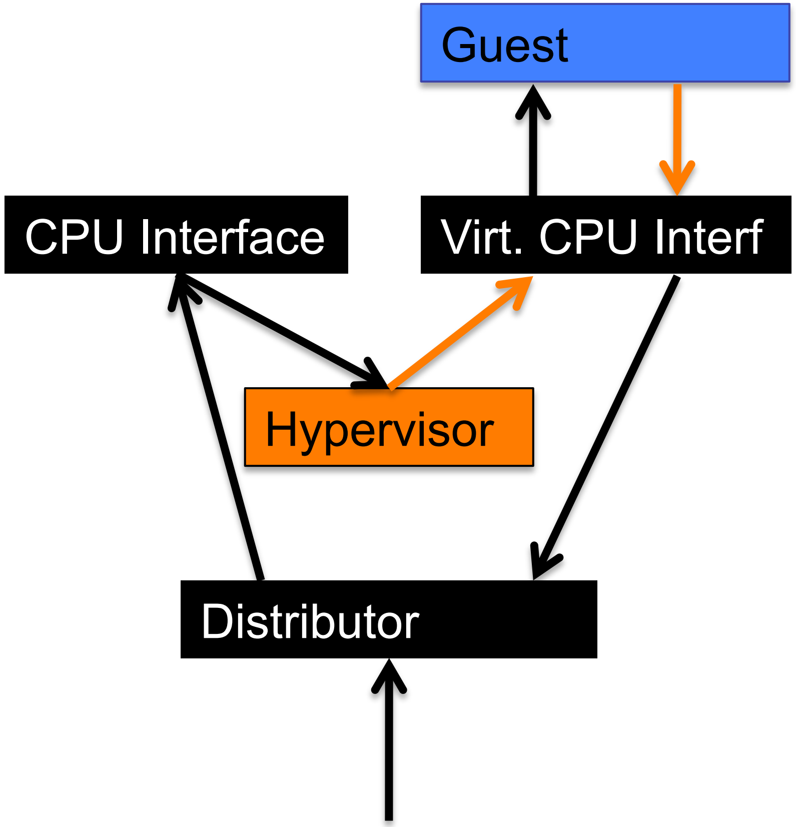
2-stage translation



- Hardware PT walker traverses both PTs
- Loads combined (guest-virtual to physical) mapping into TLB

ARM Virtualization Extensions (4)

Virtual Interrupts



- ARM has 2-part IRQ controller
  - Global “distributor”
  - Per-CPU “interface”
- New H/W “virt. CPU interface”
  - Mapped to guest
  - Used by HV to forward IRQ
  - Used by guest to acknowledge
- Reduces hypervisor entries for interrupt virtualization

Experience: Hypervisor Size

- Resonably complete prototype hypervisor utilising extensions
  - Runs Linux
  - Simulator only (no hardware)

Hypervisor	ISA	Type	Kernel	User
OKL4	ARMv7	para-virtualization	9.8 kLOC	0
Prototype	ARMv7	pure virtualization	6 kLOC	0
Nova	x86	pure virtualization	9 kLOC	27 kLOC

- Much smaller than x86 pure-virtualization hypervisor
  - Mostly due to greatly reduced need for instruction emulation
- Size (& complexity) reduced about 40% wrt to para-virtualization

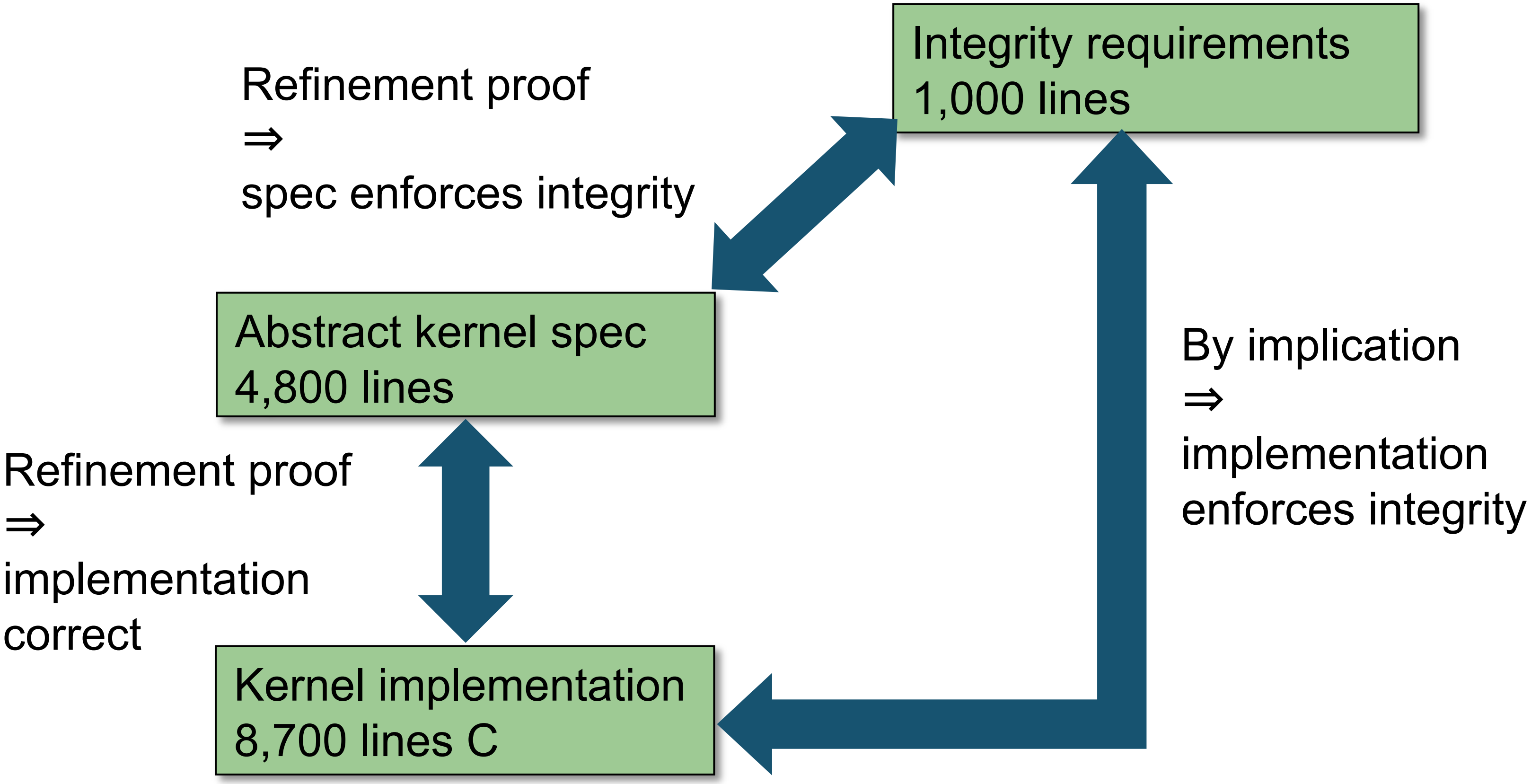
Overheads (Estimated)

Operation	Pure virtualization		Para-virtualiz.
	Instruct	Cycles (est)	Cycles (approx)
Guest system call	0	0	300
Hypervisor entry + exit	120	650	150
IRQ entry + exit	270	900	300–400?
Page fault	356	1500	700
Device emul.	249	1040	N/A
Device emul. (accel.)	176	740	N/A
World switch	2824	7555	200

- Note: *Rough* estimates due to lack of cycle-accurate simulation
- Interesting tradeoffs:
  - Fast syscalls (no emulation)
  - slower hypervisor invocation, world switch
- Pure virtualization almost certainly unsuitable for device drivers

Future of Hypervisors: seL4 Microkernel

- Q: Can you trust separation by the hypervisor?
- A: Yes: we have proof!



Integrity requirements
1,000 lines

Refinement proof
⇒
spec enforces integrity

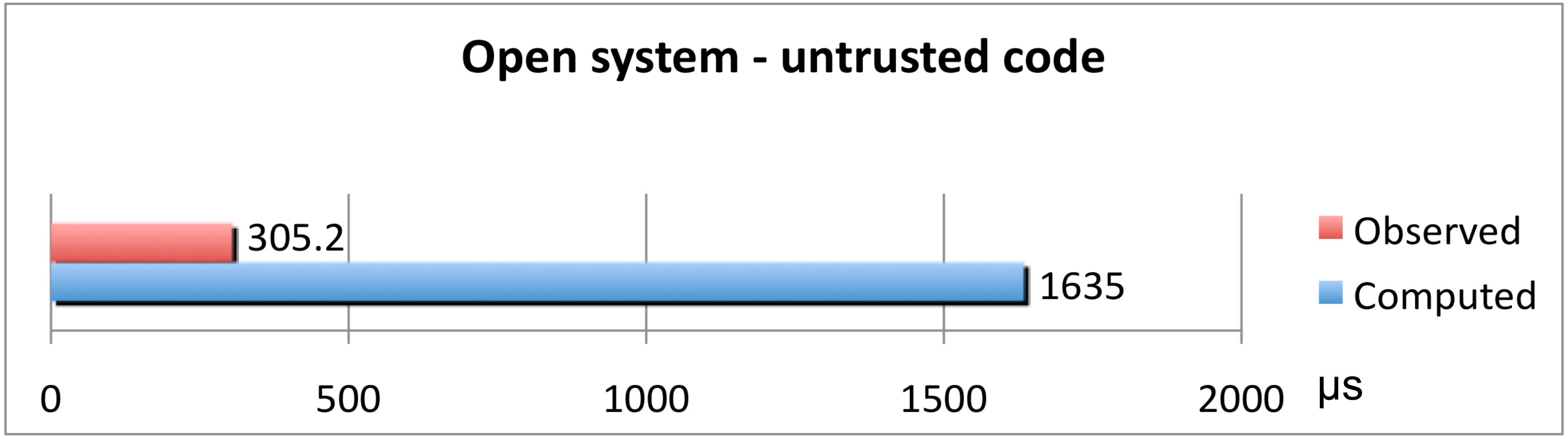
Abstract kernel spec
4,800 lines

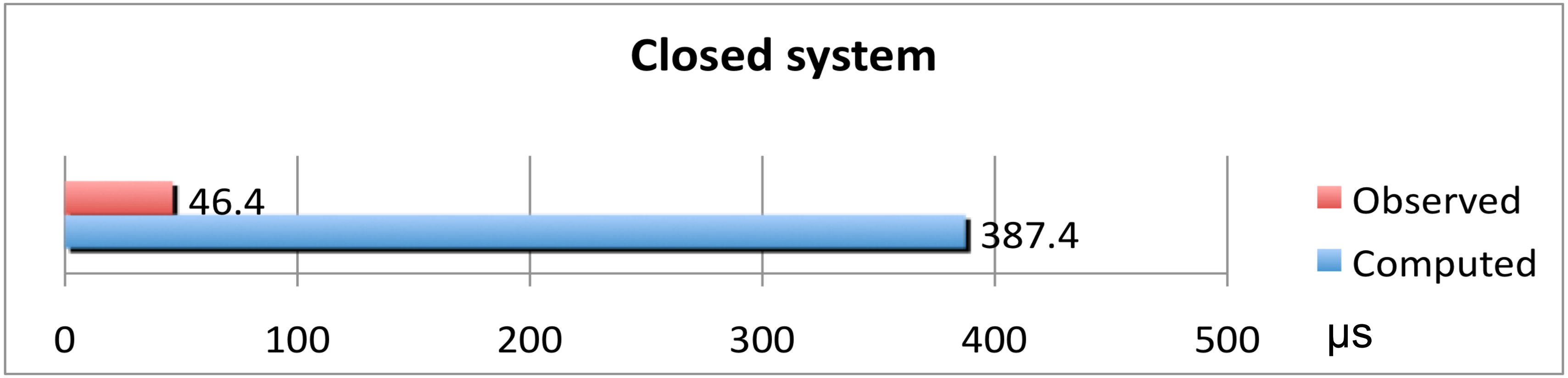
By implication
⇒
implementation enforces integrity

Refinement proof
⇒
implementation correct

Kernel implementation
8,700 lines C

seL4 WCET Analysis





Clearly early days, aiming for 10 µs WCET

Conclusions

- Virtualization is coming to mobile devices!
  - Hardware utilization
  - Security
  - Energy management
- Manufacturers are providing extensions to accelerate
- The art of para-virtualization is far from dying
- Isolation can have the strength of mathematical proof