

Antikernel

A Decentralized Secure Hardware-Software
Operating System

Andrew Zonenberg (@azonenberg)
Senior Security Consultant, IOActive

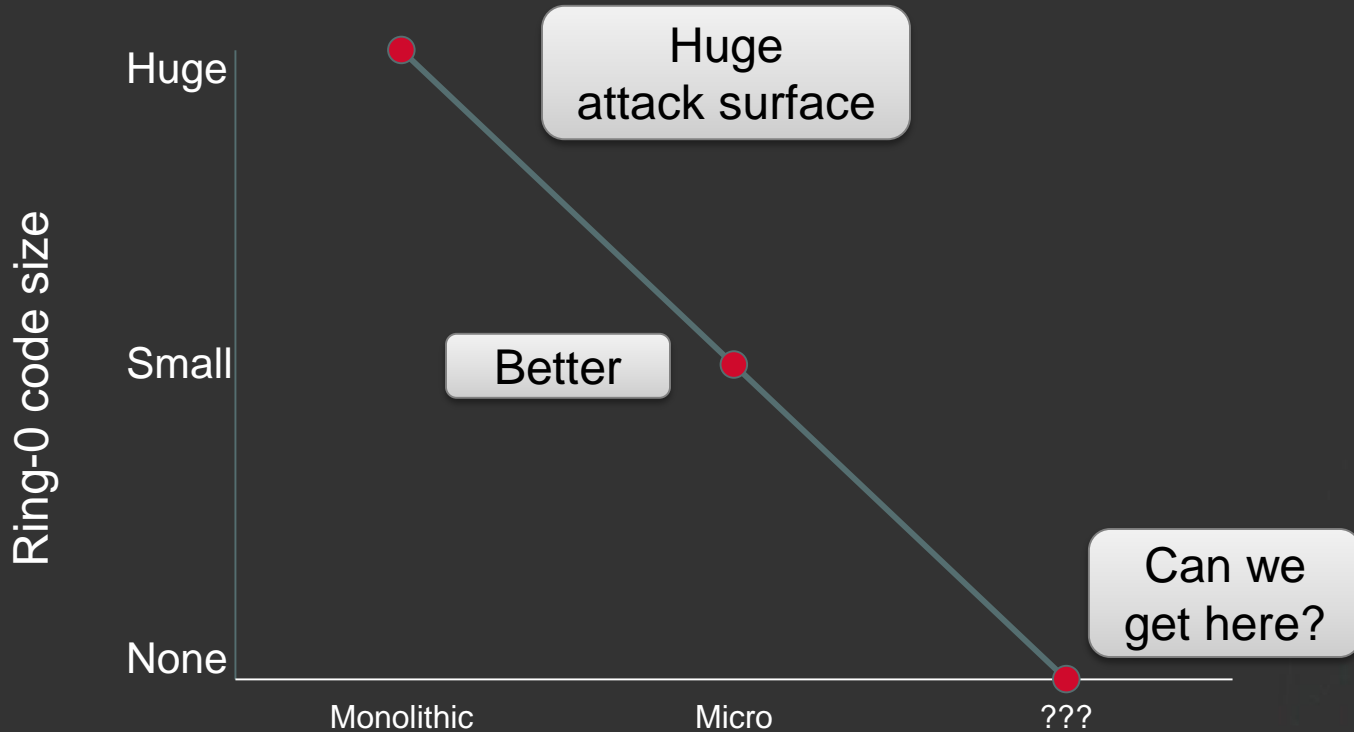
Bülent Yener
Professor, Rensselaer Polytechnic Institute



Kernel mode = full access to *all state*

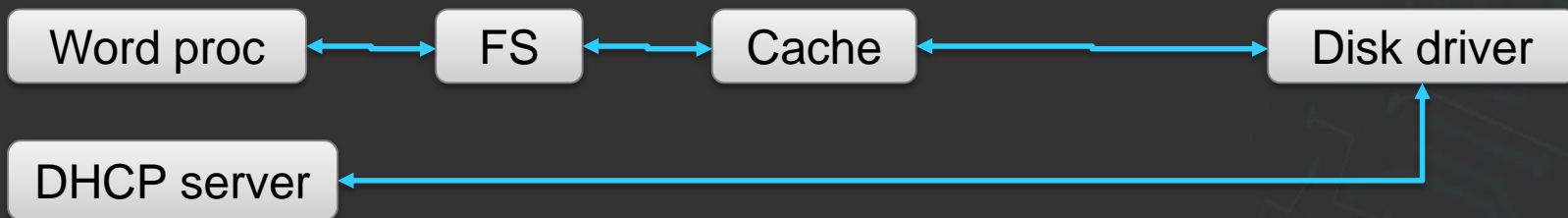
- What OS code *needs* this level of access?
 - Memory manager only needs heap metadata
 - Scheduler only needs run queue
 - Drivers only need their peripheral
 - Nothing needs access to state of user-mode apps
- No single subsystem that *needs* access to all state
- ***Any code with ring 0 privs is incompatible with LRP!***

Monolithic kernel, microkernel, ...



Exokernel (MIT, 1995)

- OS abstractions can often hurt performance
 - You don't need a full FS to store temporary data on disk
- Split protection / segmentation from abstraction



Exokernel (MIT, 1995)

- OS does very little:
 - Divide resources into blocks (CPU time quanta, RAM pages...)
 - Provide controlled access to them

But wait, there's more...

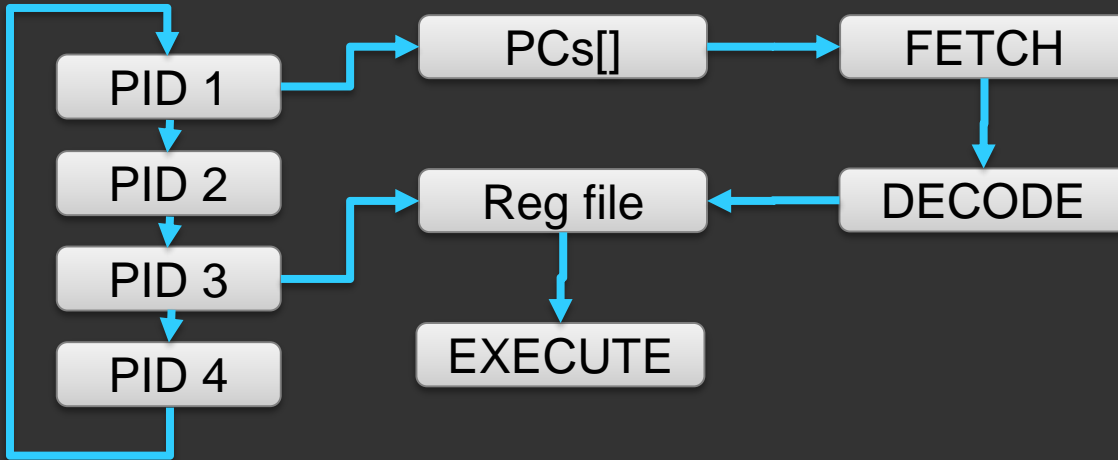
- By removing non-security abstractions from the kernel, we shrink the TCB and thus the attack surface!

So what does the kernel *have* to do?

- Well, obviously a few things...
 - Share CPU time between multiple processes
 - Allow processes to talk to hardware/drivers
 - Allow processes to talk to each other
 - Page-level RAM allocation/access control

Are you *sure*?

- Barrel proc / HT can context switch without S/W help
- What if we moved the whole run queue into the CPU?

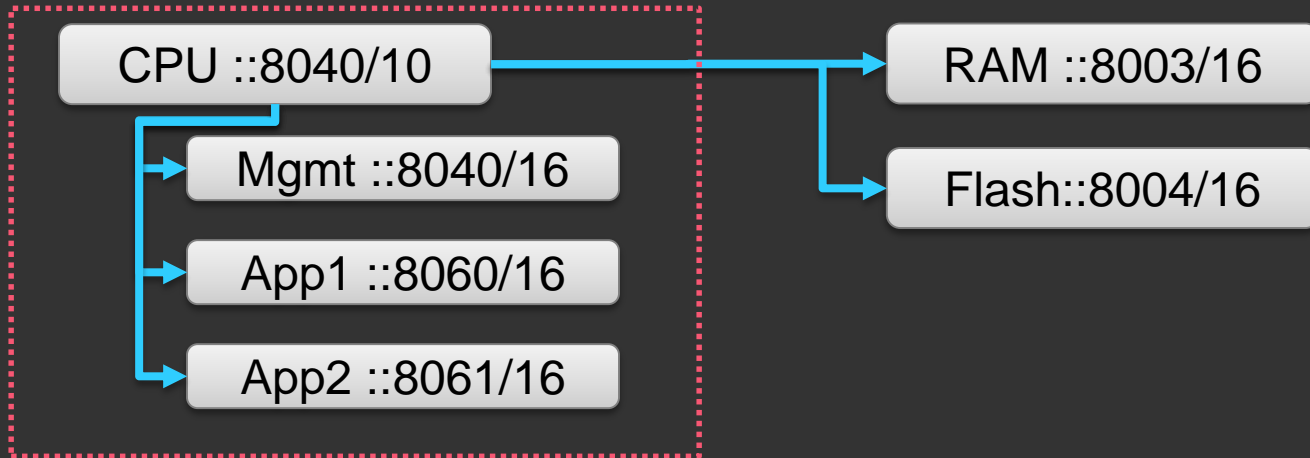


Hardware scheduler

- Circular queue of thread IDs = round robin scheduler
- Minimal gate count (one small small FIFO)
- Deterministic performance (good for hard realtime)
- No possibility of corrupting unrelated state

Access to hardware

- Instead of a bus, connect the CPU to rest of the SoC with a packet-switched NoC
 - Assign addresses as {cpu subnet prefix, PID}



Communication

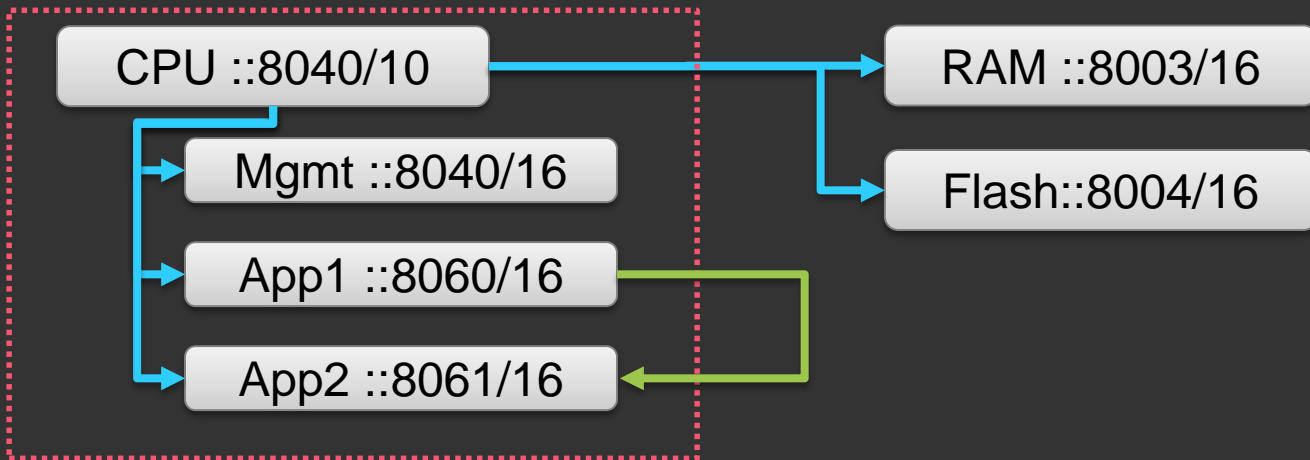
- Two parallel networks
 - RPC network: 4x 32 bit words, \approx ioctl(2)
Typically accessed via CPU registers for low latency
 - DMA network: 1-512 word data plane blocks
Typically memory mapped
- Reliable datagrams
 - In-order delivery between any pair of endpoints
 - Guaranteed minimum QoS for hard realtime systems

Network-based access control

- Provide CPU insn to send/recv message
- Network accessed via formally verified transceiver IP
 - Untrusted 3rd-party IP cores cannot spoof headers
 - Neither can arbitrary code on the CPU
- We can use packet headers for access control!
 - Node can tell what app is accessing it
 - Node makes access control decision based on msg origin

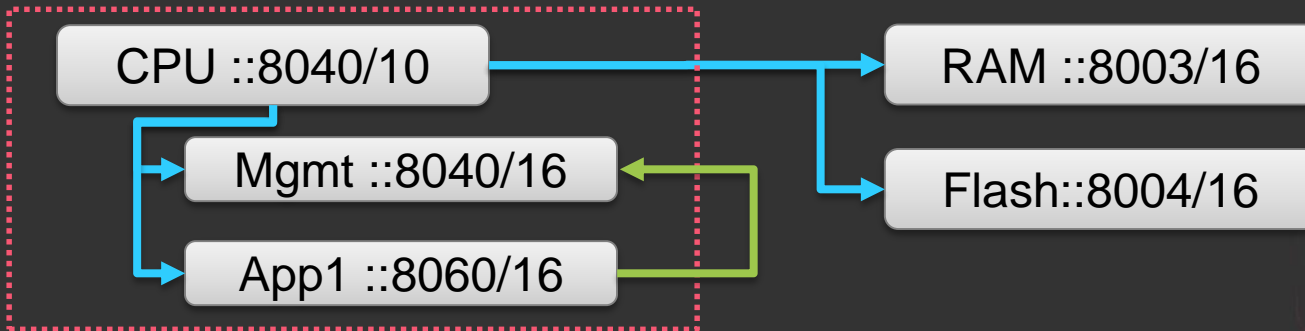
IPC is now trivial

- Each app already has a unique address on the NoC
- Just send a packet to the app's address



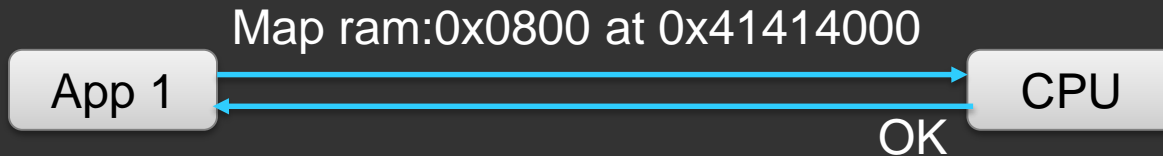
Surely sbrk/malloc must be in ring 0...

- Processor MMU provides *translation only*
- You can map any phyaddr you want, but if the peripheral says “no” the app segfaults
- mmap(2) no longer needs elevated privs!
 - Just send a message to CPU OoB address



Smart RAM controller

- RAM controller has NoC API
 - RPC: Allocate/free/chown page
 - DMA: bulk read/write



Smart RAM controller

- Trivially simple data structures
 - FIFO of free pages
 - Array of page owners
 - Control state machine is ~500 lines including fluff
- Easy to test / verify
 - Thoroughly covered by automated test suite
 - Formal verification in near future

So what's left in ring 0?

- *Nothing!*
- Remove privileged instructions from the ISA
- Run userspace on bare metal
- This is an *antikernel* – an OS with *no kernel at all!*

Key concepts

- This is not just a “hardware microkernel”
- The “OS” is an emergent entity created from many independent state machines
- These subsystems communicate in a limited, formally defined manner (complete encapsulation of state)

Modularity for security

- Each node maintains its own security state
- Your TCB is what you make it
 - A vuln in a node you don't depend on has *zero* impact on your app's security

SARATOGA CPU

- Compatible with mips GCC but not full MIPS
- 8 stage barrel processor, 200 MHz on Artix-7
 - 2 cycles each i-fetch, r-fetch
 - 4 cycles execution
- 2-issue in-order superscalar, 2^N HW threads ($N \geq 3$)
- Set-associative L1 cache, partitioned per thread
- Hardware ELF loader w/ code signing
 - HMAC-SHA256 for prototype to save FPGA resources
 - Will use RSA or ECC in real system

Name server

- Resolve 8-char hostnames to 16-bit addresses
- ROM of hard IP locations
- Writable memory for software apps
 - Signed updates to ensure authenticity

Prototype implementation

- ~187 kLine including test cases, build tools, etc
- Critical stuff is small
 - RPC/DMA networks combined 4.5 kLine
 - Name server 1 kLine
 - SARATOGA CPU 9 kLine
- All code is F/OSS (3-clause BSD)
- Goal is to encourage reproduction of results, industry/academic research, etc

Future work

- Add more peripherals
- Solve name binding problem for NVM
- Formally verify non-reduced-payload DMA protocol
- Formally verify SARATOGA (or its successor)

Questions?