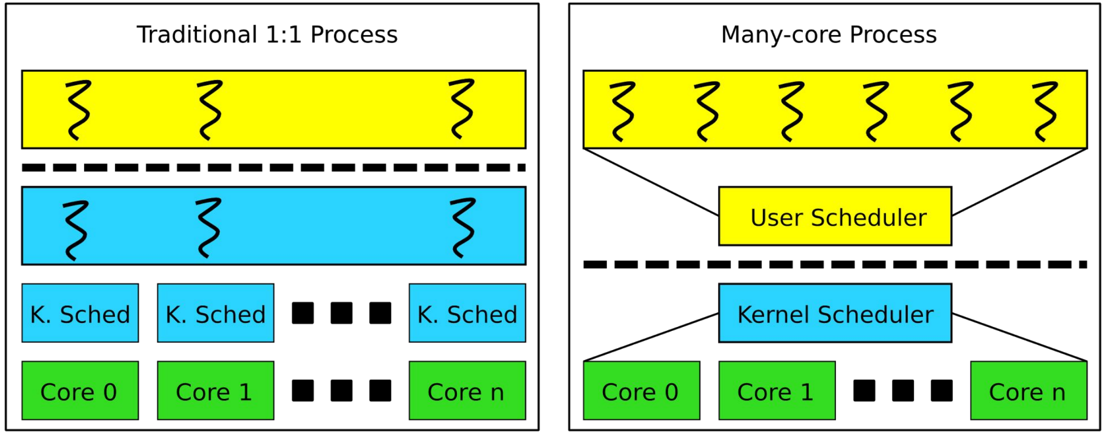
Akaros Virtualization VMs as threads

Akaros team

Akaros

- Research kernel from Berkeley
- Lots of new ideas but we will focus on one: “multi core process”
    - Key to understanding the new Virtual Machine Model
- Multi Core Process (MCP) can be thought of as a set of cores assigned as an entity to a program
    - Has the flavor of gang scheduling
- An MCP manages scheduling of its cores in a “2LS”
    - Second level scheduler, a.k.a. User level thread scheduling
- Thread scheduling is managed in the process, i.e. this is M:N system

MCP



- Treat parallel processes as a single entity
  - Gang scheduled, no kernel thread per “pthread”/core
  - Single address space
- The process is aware of its state
  - Number of cores, which ones are running, etc
- Virtual machine work builds on this model

Virtual Machine support

- First, what’s a VM?
  - Is it an instance of opening a device in /dev?
  - No: that’s *one* way of *implementing* VMs
- How do VMs fit into the Akaros model?
- We decided to take nothing for granted
- A look back is useful
- A VM could be kind of a process
- How did we used to start processes?

Start process: HP MPE

Using CREATEPROCESS

Your program can call the CREATEPROCESS intrinsic to create a child

process. Your program can control the creation of the child process with

the optional parameters itemnums and items. The information you can pass

to MPE XL through itemnums/items parameter pairs include:

 * The names of the files to be used as $STDIN and $STDLIST for the

    child process.

 * An option that indicates if the child process is to be activated

    immediately after it is created, and if the parent process should be

    suspended automatically when the child process is activated.

 A list of user-named executable libraries to be searched at load time

    for required external references.

 * A user-created UNSAT procedure, to which all unsatisfied load-time

    external references may be directed.

This is an example of a CREATEPROCESS intrinsic call:

.

.

.

```
ERRORCODE := 0;

    PIN := 0;

    BNAME := 'MYPROG.PUB.MYACCT';

    ITEMNUMS[1] := 3;

    ITEMNUMS[2] := 0;

    ITEMS[1] := 1;

    CREATEPROCESS
(ERRORCODE,PIN,BNAME,ITEMNUMS,ITEMS);
```

 *

Start a job: IBM

```
//CONCATEX JOB CLASS=6,NOTIFY=&SYSUID

//* Example 1:

//STEP10 EXEC PGM=MYPROG

//IN1    DD DSN=SAMPLE.INPUT1,DISP=SHR

//OUT1   DD DSN=SAMPLE.OUTPUT1,DISP=(,CATLG,DELETE),

//       LRECL=50,RECFM=FB

//SYSIN  DD *

//CUST1  1000

//CUST2  1001
```

Start a job on CDC

```
/JOB
COPYBR,INPUT,INPUT1.
COPYBR,INPUT,INPUT2.
REWIND,INPUT1,INPUT2.
* RUN CHECKPOINT JOB
LINK1,INPUT1,OUTPUT,PUNCH1,UT1.
* MANIPULATE FILES
PACK,PUNCH1.
REWIND,PUNCH1.
RETURN,POOL.
RENAME,OPTP=NPTP.
* RUN RESTART JOB
LINK1,INPUT2,OUTPUT,PUNCH2,UT1.
/EOR
NASTRAN FILES=NPTP
.
. (Data for Checkpoint Job)
.
/EOR
NASTRAN FILES=OPTP
.
$ READ THE RESTART DICTIONARY
READFILE PUNCH1
.
CEND
.
. (Data for Restart Job)
.
/EOF
```

On newer mainframes

```
<devices>
  <redirdev bus='usb' type='tcp'>
    <source mode='connect' host='localhost' service='4000'/>
    <boot order='1'/>
  </redirdev>
  <redirfilter>
    <usbdev class='0x08' vendor='0x1234' product='0xbeef' version='2.56' allow='yes'/>
    <usbdev allow='no'/>
  </redirfilter>
</devices>
```

On Unix, in 1972

- fork()
- Starting a process was *ugly* before this
- Unix made it easy and clean
- It was also controversial because it encouraged people to use it
- I.e. that one could, in DMR’s words, be “profligate” with processes
- Forking a process to list files was considered ridiculous in many places

Virtualization today

- Remember that “newer mainframes” slide?
- That’s from libvirt
    - It’s a tiny fraction of the total giant pile of XML for *one* VM
    - Far more complex than the JCL example!
- To start a virtual machine on linux requires
    - Daemons
    - Special user ids
    - Batch scheduling
    - Gigantic XML files
    - Mountains of code
- We’ve recreated batch systems for VMs
- What’s going on here?

Virtualization model

- Virtualization on Linux/BSD/Unix requires a device
    - Was not possible to go deeper, i.e. rework task and vm structures
- Device access requires privileges (usually root)
    - But simple setuid approaches are dangerous
    - So we have daemons we talk to to do it for us
- We did not want to assume current practice was the right approach
- We ported 5 different VMs to Akaros while evaluating them
    - KVM, Dune, Bhyve, two others you never heard of
- We did not have restrictions on how much we could change
- We did not want to recreate a batch system for VMs
- So, question, what keeps VMs just being another thread in our process?

VMs as threads

```
struct virtual_machine vm;
static int count;

int main(int argc, char **argv)
{
         int  attrs = 0;
         vthread_attr_init(&vm, attrs);
         vthread_create(&vm, 0, vmcall, NULL);

        while(atomic_read(count) < 100000) {
        }

        return 0;
}
```

Thread

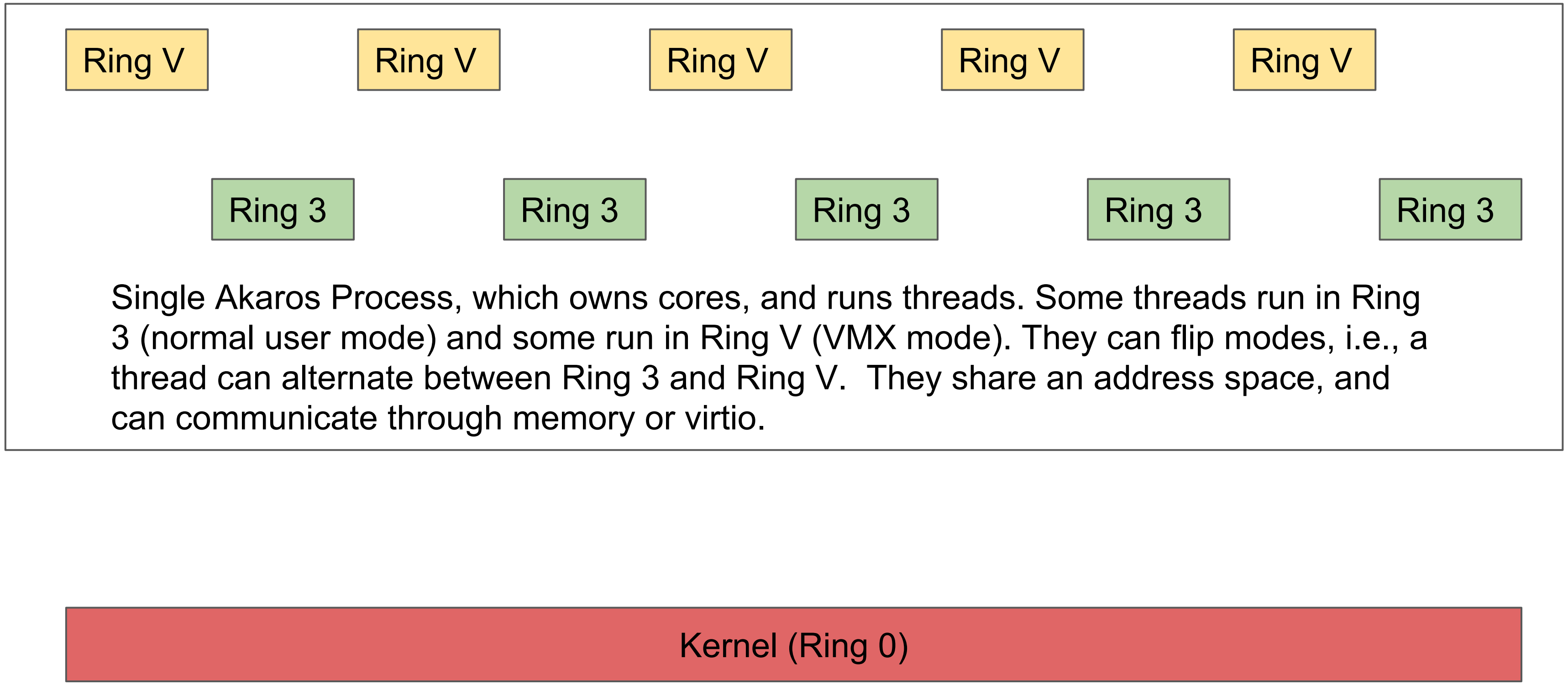
Virtual Machine Thread

```
static void vmcall(void *a)
{
        while (atomic_add(count, 1) < 1000000) {
                __asm__ __volatile__("vmcall\n\t");
        }
        __asm__ __volatile__("hlt\n\t");
}
```

In Akaros, VMs are threads

- Recap:
  - Akaros processes and threads are somewhat different than you’re used to
  - Akaros schedules sets of cores (1 or more), not processes
  - And schedules entities called ‘multi core processes’ onto those cores
  - Threads are scheduled at user level onto cores
- We have extended the thread model to include Virtual Machine Threads
- Virtual Machine Threads (vthreads) can run:
  - Linux 4.8 (with 12 lines of patches so that we can tell it clock rates via cmdline)
  - Any code that shares the Host Ring 3 address space (equivalent to Guest Ring 0)

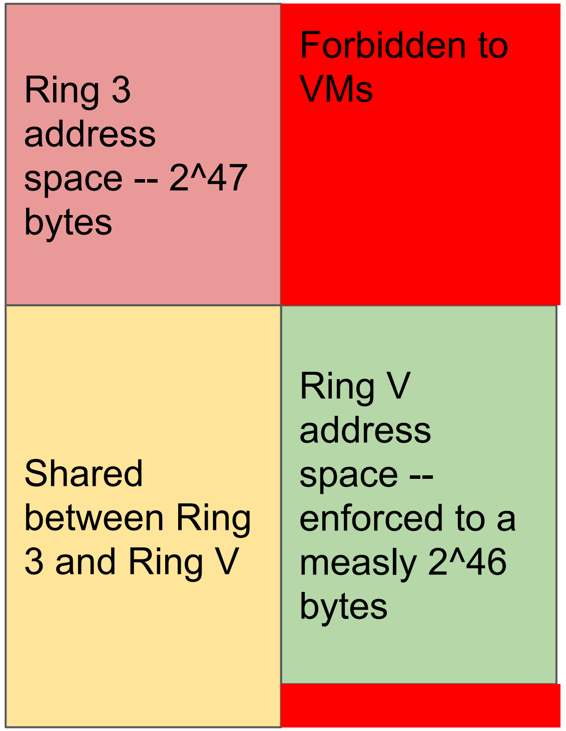
VMs and threads as part of an MCP



Single Akaros Process, which owns cores, and runs threads. Some threads run in Ring 3 (normal user mode) and some run in Ring V (VMX mode). They can flip modes, i.e., a thread can alternate between Ring 3 and Ring V. They share an address space, and can communicate through memory or virtio.

Process address space (x86)

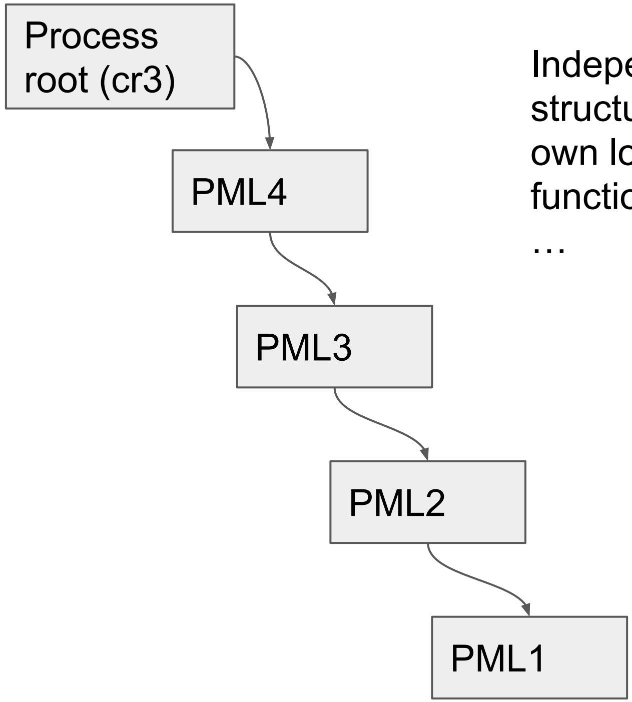
- Ring 3 and Ring V share an address space
- Ring V is limited to 2^46 bytes
- If we are concerned about what's in the VM we can use this limit
  - Run VMM *starting* at 2^46
  - Only leave stuff in the "low" memory that is for the guest
  - Currently due to gcc limits we also enforce only Ring 3 can access low 16 MiB (save for first 1 MiB -- 8086 XXX)


Ring 3 address space -- 2^47 bytes
Forbidden to VMs
Shared between Ring 3 and Ring V
Ring V address space -- enforced to a measly 2^46 bytes

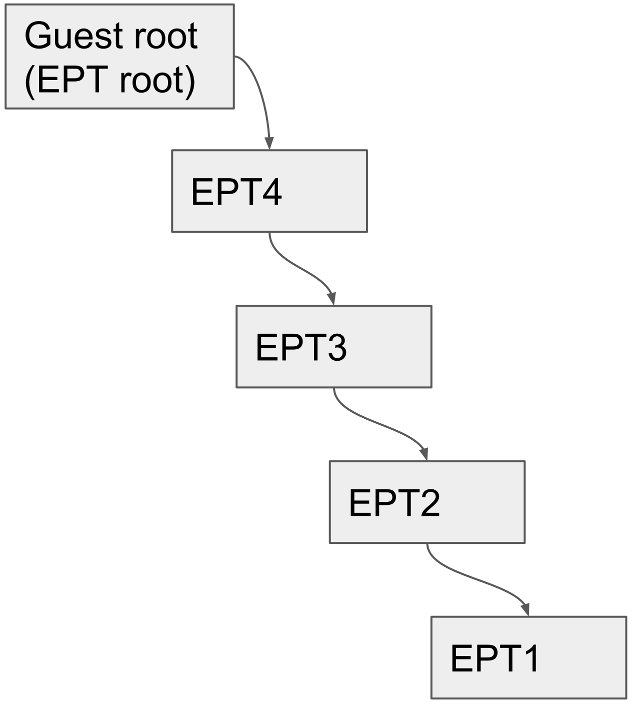
We integrate VM Threads tightly into the kernel

- Paging
  - Page table pages become double-wide (x86-specific, more on this later)
- Entering the kernel
  - Previously, there's only IRQs, Traps, SYSENTER/SYSCALL
  - Added a handler for VMEXIT, treat it like a fault
- Return to user, whether VM or Ring 3
  - OS knew how to run a "context" for IRQs/Traps or SYSENTER
  - Added a handler to run a "VM context"
- No arch-independent changes to scheduler or process management code
- Added a user-level scheduler for VMs
  - Application specific scheduling, where the application is a VMM

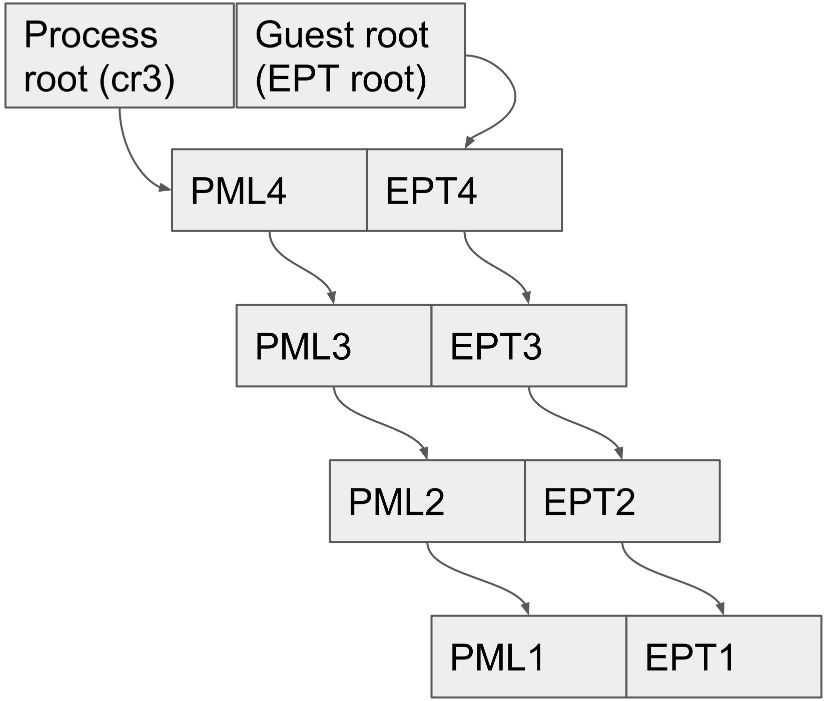
Paging in Linux for virtual machines



Independent structures with their own locks, traversal functions, allocators, …



Akaros paging on x86



- Page table roots are paired, as are page table pages (PTP)
- PTP are 2 * 4K pages
  - Process PTP is lower (“even”) 4k
  - VM PTP is upper (“odd”) 4k
- Lower half is for thread in Ring 3
- Upper half is for thread in Ring V
- Threads can jump back and forth between those modes
  - VMM/VM duality
- Code for VM page fault and process page fault is the same code
- Can always see PTP for VM given a process, or process given a VM
- EPT fault looks like a normal page fault, ugly details buried in HAL (not much code)

Architecture Independent Paging Model for VMs?

- The idea is that the address space for the VM in Ring V is the same as the process in Ring 3. We call this “EPT == KPT” (kernel page table)
- If x86 had used the same format for the EPT as the normal page table, we could have used the process’s normal kernel page table as the EPT.
    - Still need to be careful of permissions - need to respect the User Bit in the KPT
    - The KPT also maps the Host Ring 0 kernel, which Ring 3 cannot access
    - Depending on desired granularity, we can differentiate access at many levels, e.g. could enforce restrictions with a “double page” at PML4
        - One page table page for Ring 3
        - One page table page for Ring V with one or more entries missing

Entering kernel from ring 3/ring V

- The basic API is the same in each case
- And utterly different from, say, /dev/kvm
- In KVM, vmexit is the return value of an ioctl to start a vm
- VMM is responsible for resolving the problem, changing VM state
- In Akaros, vmexit is like any other case where ring 3 drops to ring 0 for services
    - Ring 3: traps, system call, page fault
    - vmexits: vmcall, EPT fault, MSR access, IO, etc
- Same style of handling in the kernel for vmexits as faults:
    - Kernel tries to resolve the fault without blocking (e.g. page fault, lookup in page cache)
    - If the kernel fails, the faulting context is **reflected** to the user-level scheduler (2LS)
    - For userspace, a VM fault is just another reflected fault

Running a ring 3 / ring V context

- Akaros has three context types:
  - Hardware - involuntarily created during a trap/interrupt. Think “every register”
  - Software - voluntarily created during syscalls and **user-level threading**. Think “AMD SYSV ABI”
  - VM - involuntarily created during vmexit. Think “every register”, plus exit info and guest_pcoreid
- Returning to userspace (Ring 3 or Ring V) is always the same: proc_pop_ctx()
  - Whether or not we return to a VM or not is solely dependent on the type of context
- The kernel does maintain, per process, an array of guest physical cores
  - Various x86 accounting not related to the running context
  - VMCS, certain MSRs (STAR, LSTAR, SFMASK, KERNEL_GS_BASE), other fun stuff
- Userspace knows how to run contexts too!
  - Simply build the context as part of a struct uthread and try to run it
  - But on x86, it asks the kernel to run VM contexts, since vmlaunch / vmresume is privileged

Summary

- Akaros VMs are unlike any other VMs
- Threads can easily switch from being a VM to being a host thread
- We invented a new user mode, Ring V
- A VM is a thread in Ring V
    - When a VM Thread is in Ring V, it can run as Guest Ring 0 or Guest Ring 3
    - Can manage its own page tables, interrupt tables, etc.
- Shares memory with Host Ring 3 - easy virtio communication via shared mem!
- In most ways, thread programming models can be used
- Kernels also look like threads and spinning up a core looks like CPU hotplug, accomplished by spinning up a vthread with IP at the 64-bit entry point

Implications for RISCV

- Paging model. How will RISCV do nested paging? Same format for EPT?
- Can Ring 3 pop into “VM mode”?
- It’d be nice to avoid the massive shadow state (VMCS and friends)
- What about interrupts?
  - x86 has mechanisms to inject interrupts without a vmexit
  - Would be nice for guests to be able to IPI other guest cores without an exit
  - Keep in mind things like the interrupt disabling window (x86, IRQs aren’t enabled til the instruction **after** “sti”, so you can halt safely: “sti; hlt;”)
- What about devices and topology?
  - Ideally, the native device discovery method would be easily virtualizable
- Need to be able to inject interrupts without exiting the VM

Summary

- We’ve been a bit concerned that RISCV is trending to “here’s how Linux does it on x86 so do it that way”
- Reinforced by some email threads
  - “Experience with *the kernel* shows that …”
  - “The config string will be tough for *the core* to upstream”
- RISCV is a chance to enable software innovation
- Not get locked into “but we’ve always done it this way”
  - Isn’t that part of what it’s about?
- We want to ensure that RISCV doesn’t make VM Threads impossible
  - Because then we can’t use RISCV :-(
- And hence are interested in your comments/questions