PowerOn and Checkpointing

Wednesday, August 18, 2010

Outline

- PowerOn of VMX and VMM
- PowerOn with checkpoint restore
  - Complication: vMotion
- Checkpoint save: stun, quiesce, unstun
  - Complication: lazy save

PowerOn

- Managed
  - hostd / WS via VMHS: vim-cmd, etc.
- Unmanaged
  - Visor: /bin/vmx ++options
  - Hosted: vmware-vmx
- Hand-off (FSR)
  - authd + cmd line

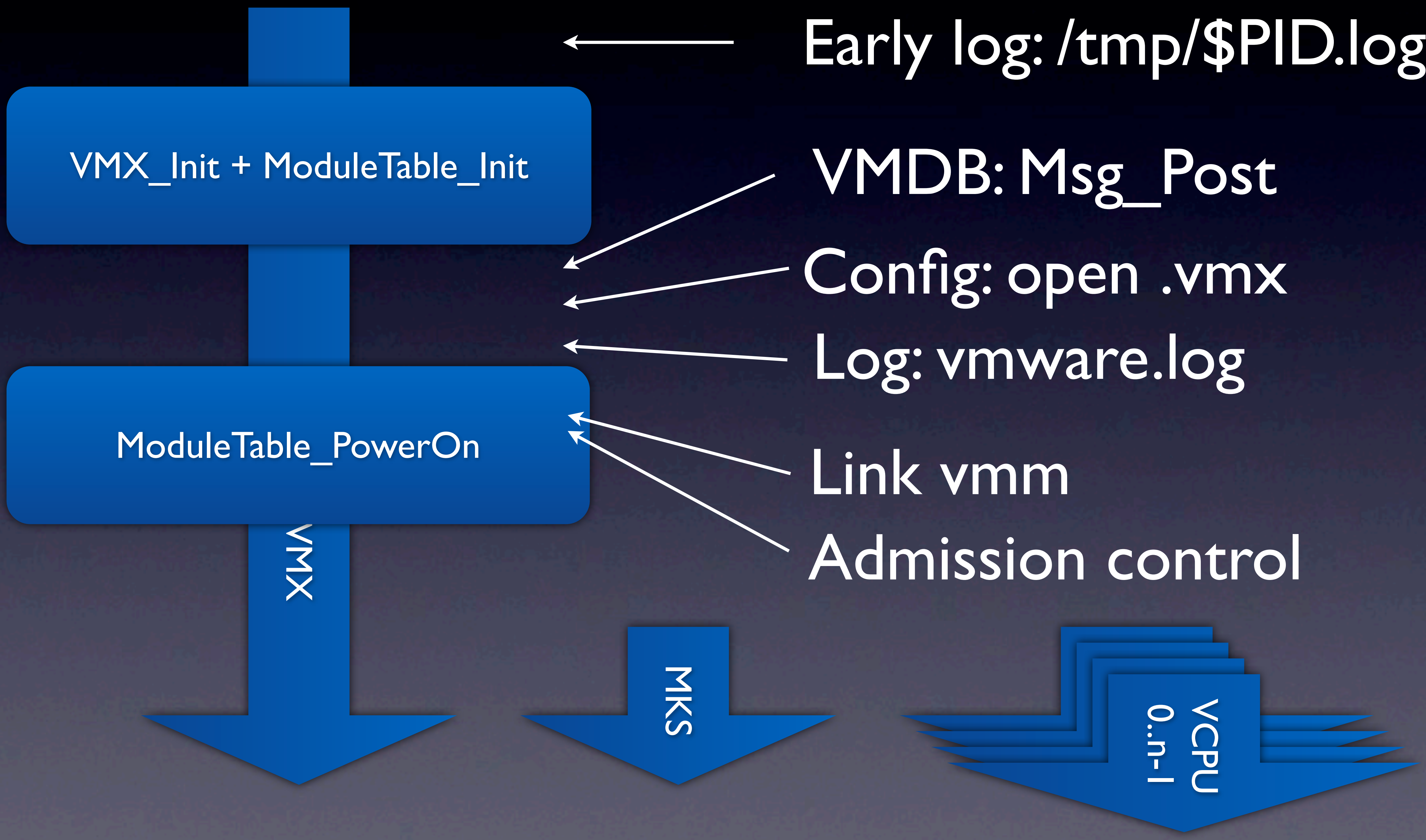
Different ways of starting the VMX process.

Hostd and the Workstation local UI link in lib/vmhostsvcs (VMHS), a library for managing VMs. Contains code to start a new VMX. Posix does double-fork, Windows does CreateProcess, Visor does VMKernel_ForkExec. This is the only “supported” way to start VMs for our customers.

Unmanaged / command line: used by automation tools (Frobos, etc.), always start headless VM. Visor cmd line can accept ++ options to vmkernel, used to control swap behavior.

FSR path is for starting a new VMX given an existing VMX; it has special code to handle different build types (talk to Ariel). VMHS knows enough about this path to convert abrupt disconnect into FSR hand-off (talk to Petr).

Logging and Msgs



Wednesday, August 18, 2010

Initialization and error reporting.

DO NOT RACE TO THE BEGINNING. Only one thing can be initialized first, some initialization is necessary for even malloc() / Log() to work, significant initialization is necessary to report errors to end user.

Init: VMX's init table is for initialization that does not require a config file and does not report errors. It can use host configuration (/etc/vmware/config).

PowerOn: initialization dependent upon a config file. By this point, full error message reporting to end user is available and log persists in VM's directory.

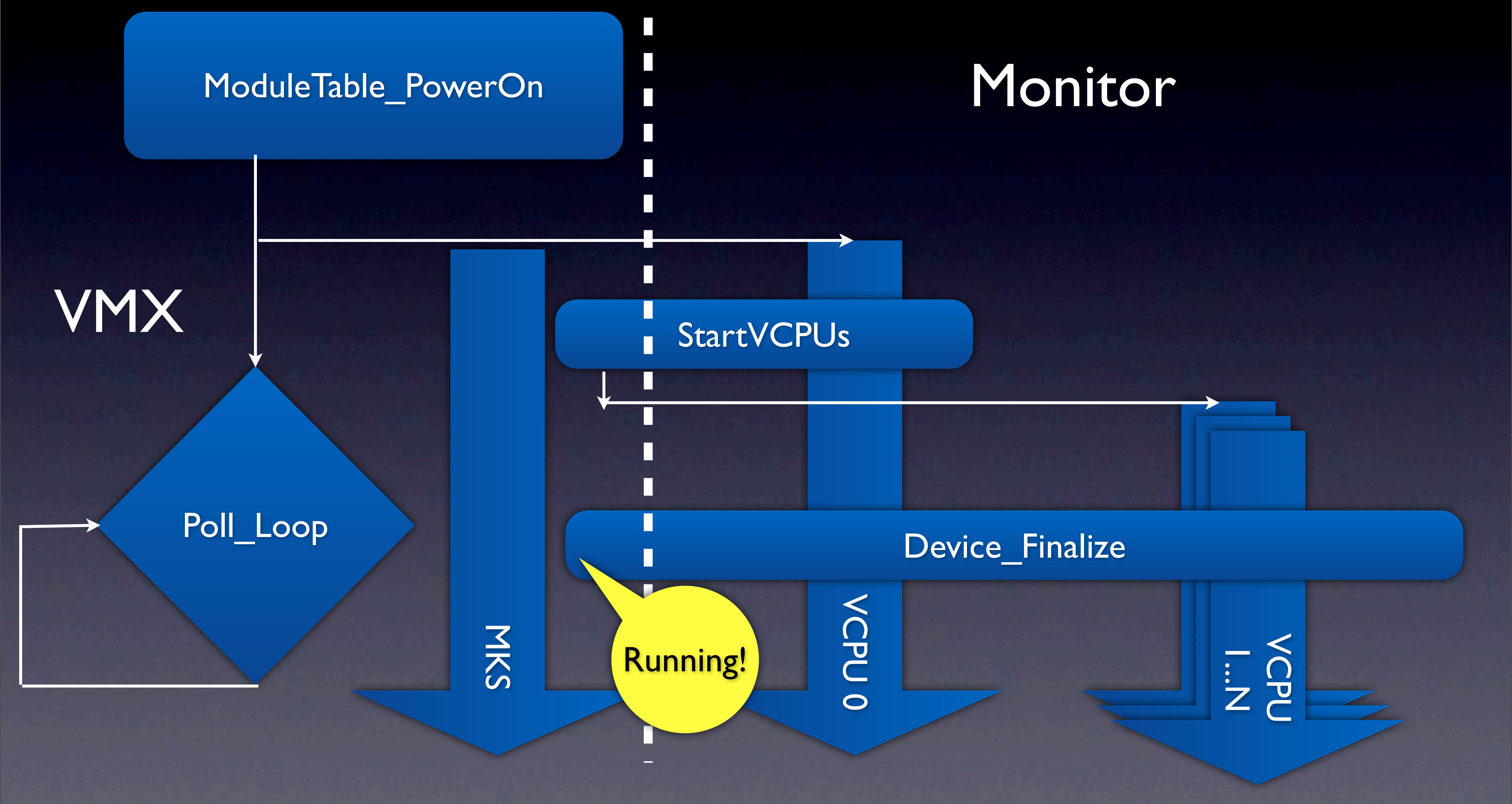
ESX and memory: we start with a small, (usually) swappable amount of memory (currently 3MB), which we use to link the monitor and otherwise compute a real reservation, which then gets used for the real admissions control.

The entire power-on sequence is presently single-threaded and NOT locked. This may change for O/P. There is only one thread in the process at this time, so we are not processing Poll - callbacks will not fire until PowerOn completes.

After all PowerOn functions complete successfully, we start up MKS and VCPU threads. More detail on VCPU threads on the next slide.

Future work: today, resetting a VM runs all PowerOff functions followed by all PowerOn functions. Init only runs once. We hope to change this for O/P (talk to Dmitriy / Jesse Pool).

Starting VCPUs



Wednesday, August 18, 2010

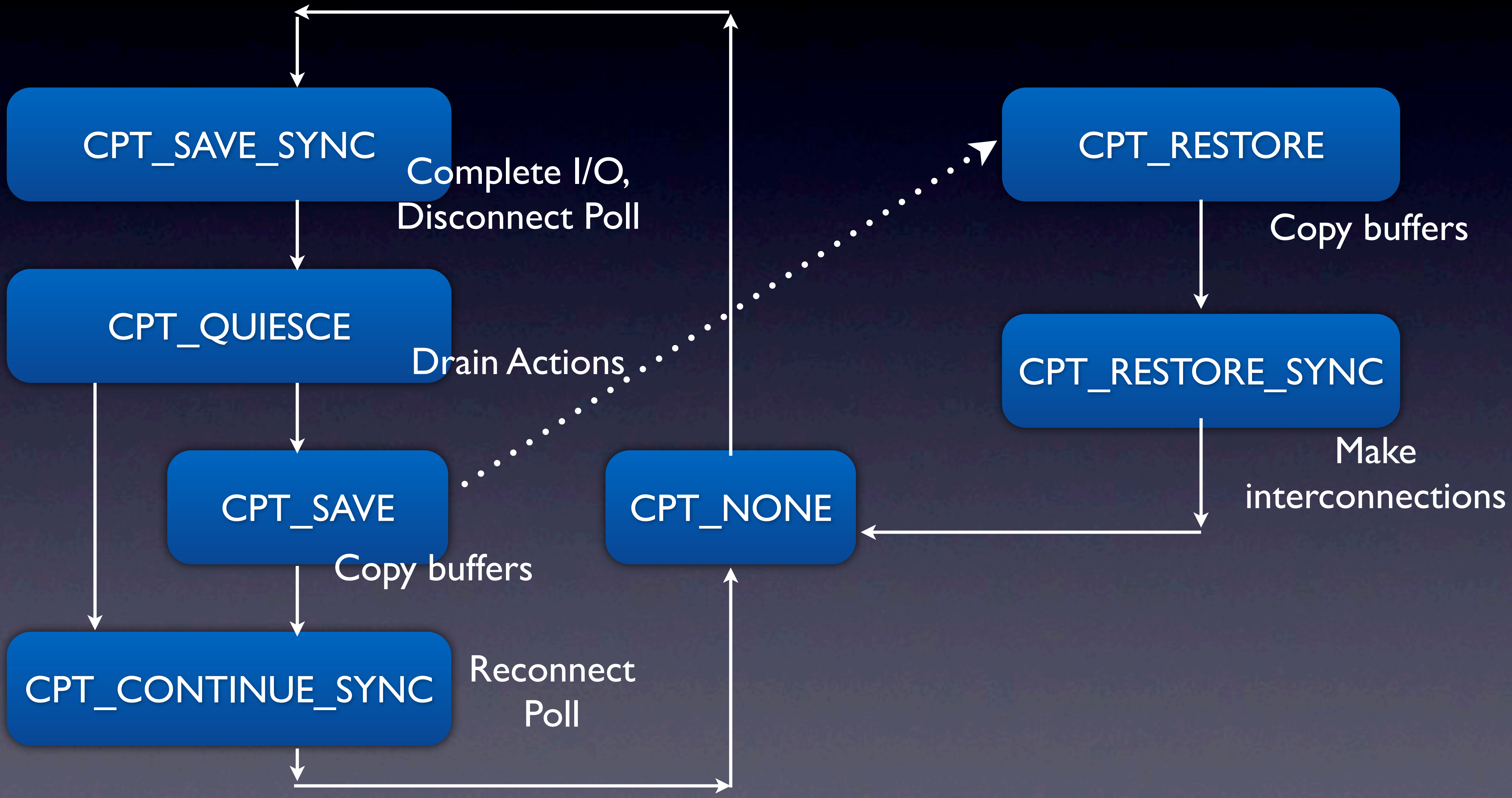
More detailed look at starting VCPU threads, in the normal case (e.g. start in BIOS).

After PowerOn completes, VMX thread starts two more threads: MKS and VCPU0. VMX thread then proceeds to a poll loop, which runs until the VM exits.

VCPU0 does some initialization in the monitor, then makes a usercall to start other VCPUs, then enters a barrier. Other VCPUs initialize, arrive at barrier. Then all proceed in parallel through more initialization. Eventually, they rendezvous; VCPU0 calls Device_Finalize, which switches to userlevel and finishes device initialization (e.g. connect removable devices, assign PCI slots).

The first action drain occurs after Device_Finalize completes.

Checkpoint phases

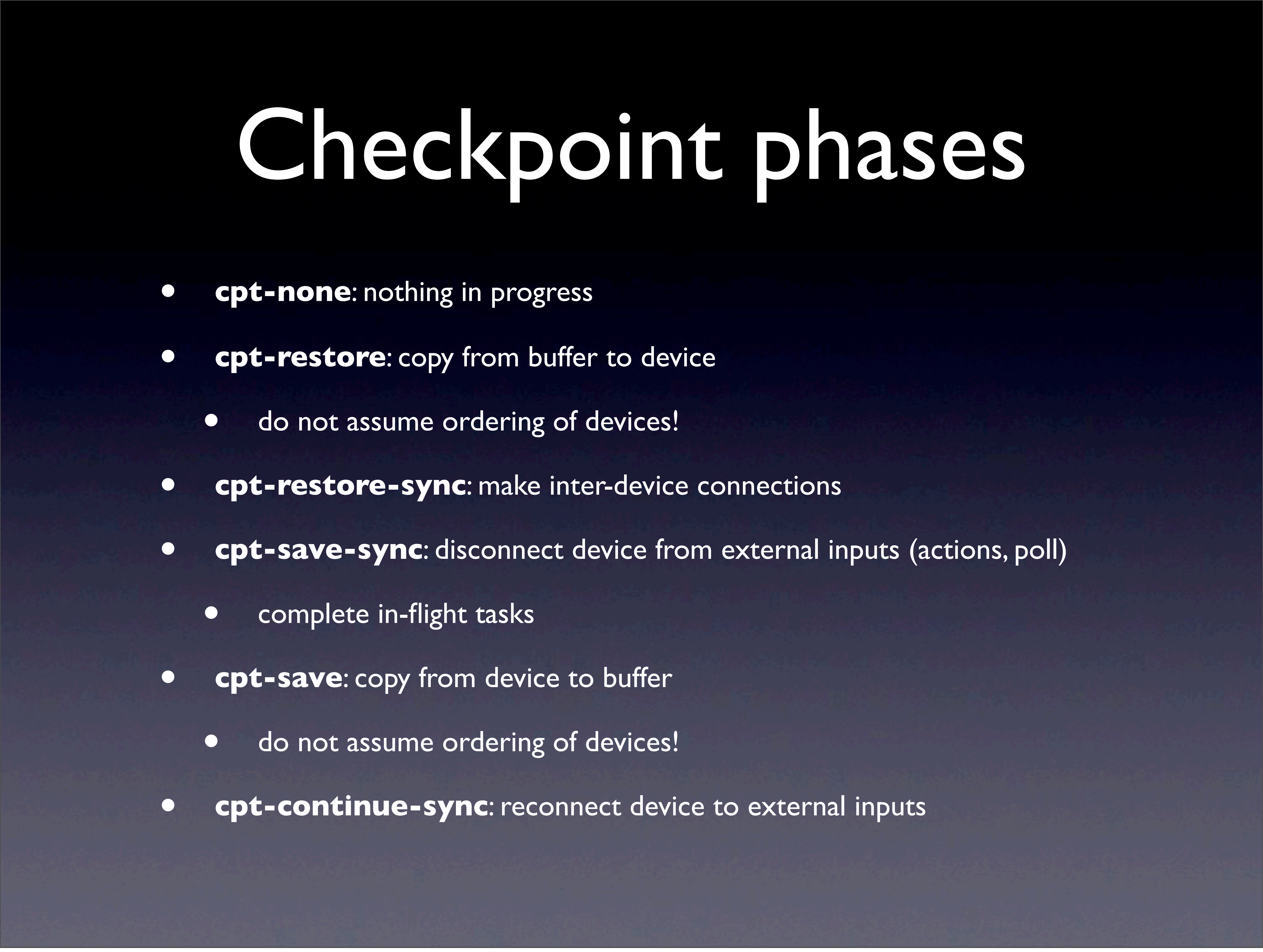


The various phases of checkpoint.

Of note:

cpt-restore and cpt-save are purely copy operations local to one device. Do not make ordering assumptions about which devices are saved/restored first.

cpt-save-sync is an opportunity to flush any pending I/O. I/O must either be completed synchronously in cpt-save-sync, or saved for re-issue during cpt-restore. Completing I/O synchronously is generally easier to implement.



Checkpoint phases

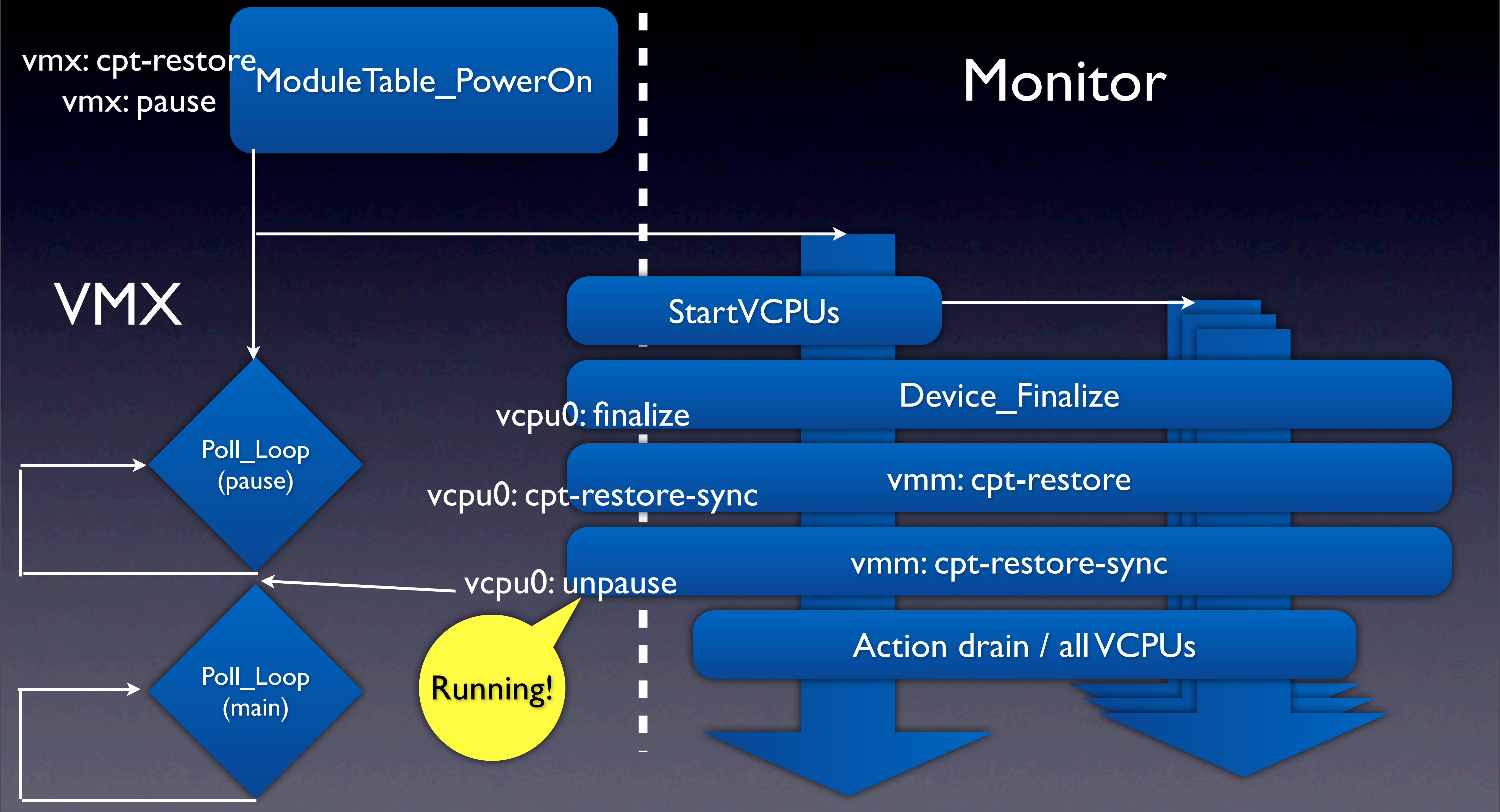
- **cpt-none**: nothing in progress
- **cpt-restore**: copy from buffer to device
  - do not assume ordering of devices!
- **cpt-restore-sync**: make inter-device connections
- **cpt-save-sync**: disconnect device from external inputs (actions, poll)
  - complete in-flight tasks
- **cpt-save**: copy from device to buffer
  - do not assume ordering of devices!
- **cpt-continue-sync**: reconnect device to external inputs

Wednesday, August 18, 2010

The various phases of checkpoint.

Of note:
cpt-restore and cpt-save are purely copy operations local to one device. Do not make ordering assumptions about which devices are saved/restored first.
cpt-save-sync is an opportunity to flush any pending I/O. I/O must either be completed synchronously in cpt-save-sync, or saved for re-issue during cpt-restore. Completing I/O synchronously is generally easier to implement.

CPT Resume



Wednesday, August 18, 2010

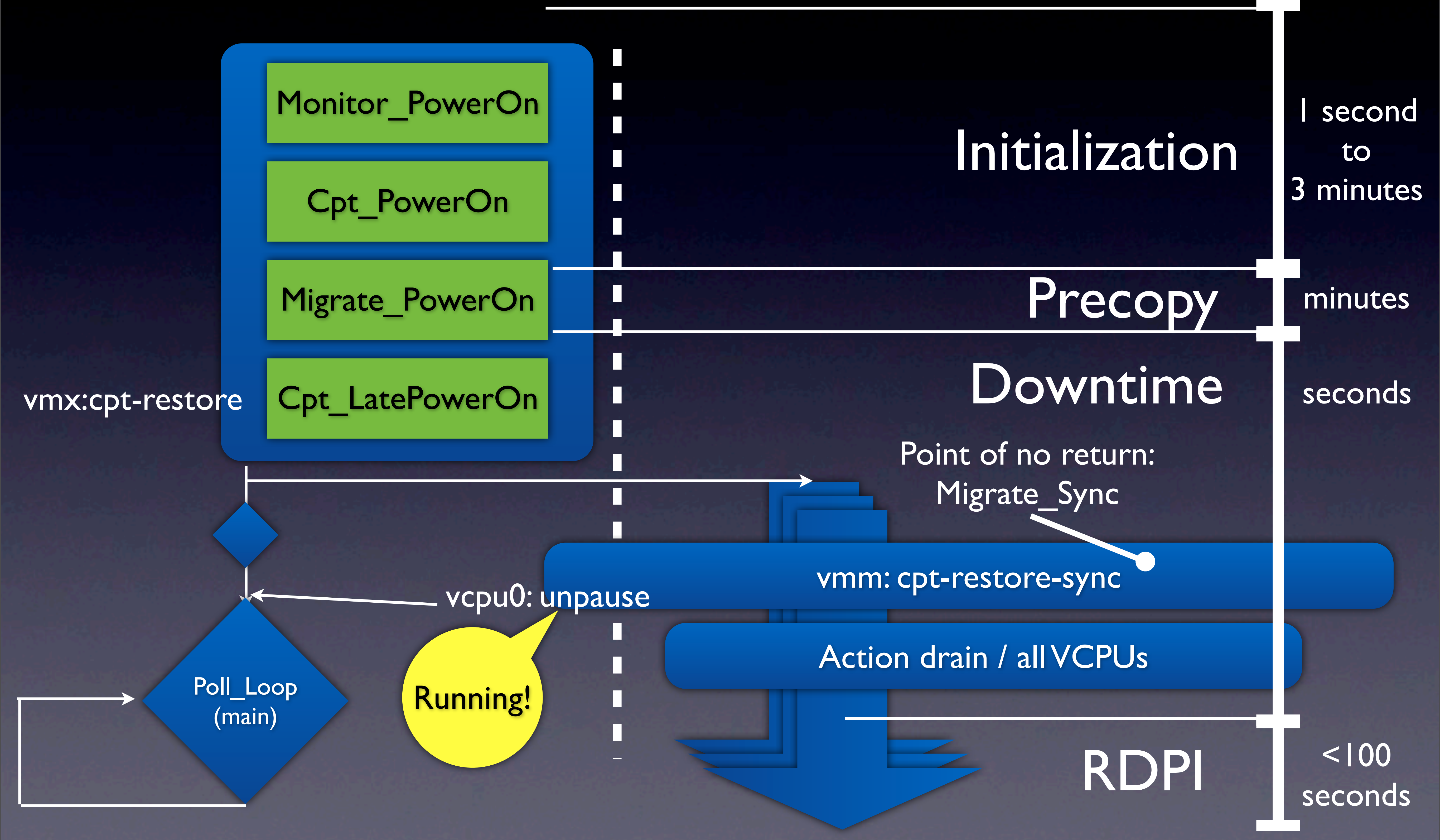
Same diagram, with checkpoint restore.

During PowerOn (specifically, Checkpoint_PowerOn), VMX discovers it is restoring. Later, during Checkpoint_LatePowerOn, it calls a restore function at userlevel for each device. (While still single-threaded / no locks). VMX also sets "pause", which restricts the set of poll events fired during the eventual poll loop and (later) puts vmm in a continue-processing loop where it drains actions but does not run guest instructions.

VCPU threads start as before. After Finalize, a few more steps occur: a cpt-restore in vmm to set up some more state, then a cpt-restore-sync first in vmx then in vmm to allow cross-device connections to be established. The end of the cpt-restore-sync handler triggers an unpause, so poll loop and action draining can run normally.

Special guarantee: monitor guarantees all VCPUs will drain actions to completion at least once before any VCPU runs guest code. This ensures all cpt-restore-sync side effects are seen synchronously.

Resume + vMotion



Wednesday, August 18, 2010

Checkpoint restore, with vMotion added

Initially, a VMX comes up as if it were a normal VM, until it reaches Migrate_PowerOn. (Note: no checkpoint data is available until after Migrate_PowerOn!) Once at that point, VMX tells Vmkernel destination is ready; VMX then loops in “precopy” until Vmkernel decides enough data has transferred to perform a switchover. Vmkernel stuns source, then destination is allowed to proceed through rest of PowerOn (esp. Checkpoint_LatePowerOn that actually restores vMotion checkpoint). Sequence continues as per previous slide. After checkpoint is restored, VM is allowed to run, but some pages are still on source: this is new feature Resume During Page In (RDPI), such pages are demand-faulted by destination.

In vmm’s restore-sync function for Migrate, there is a VMKCall that is the point of no return. Before this point, dest VMX can error out (Msg_Post / clean error during PowerOn, Panic during restore-sync) and VM continues running on source. After this point, VM is running on dest and source VMX is powering off.

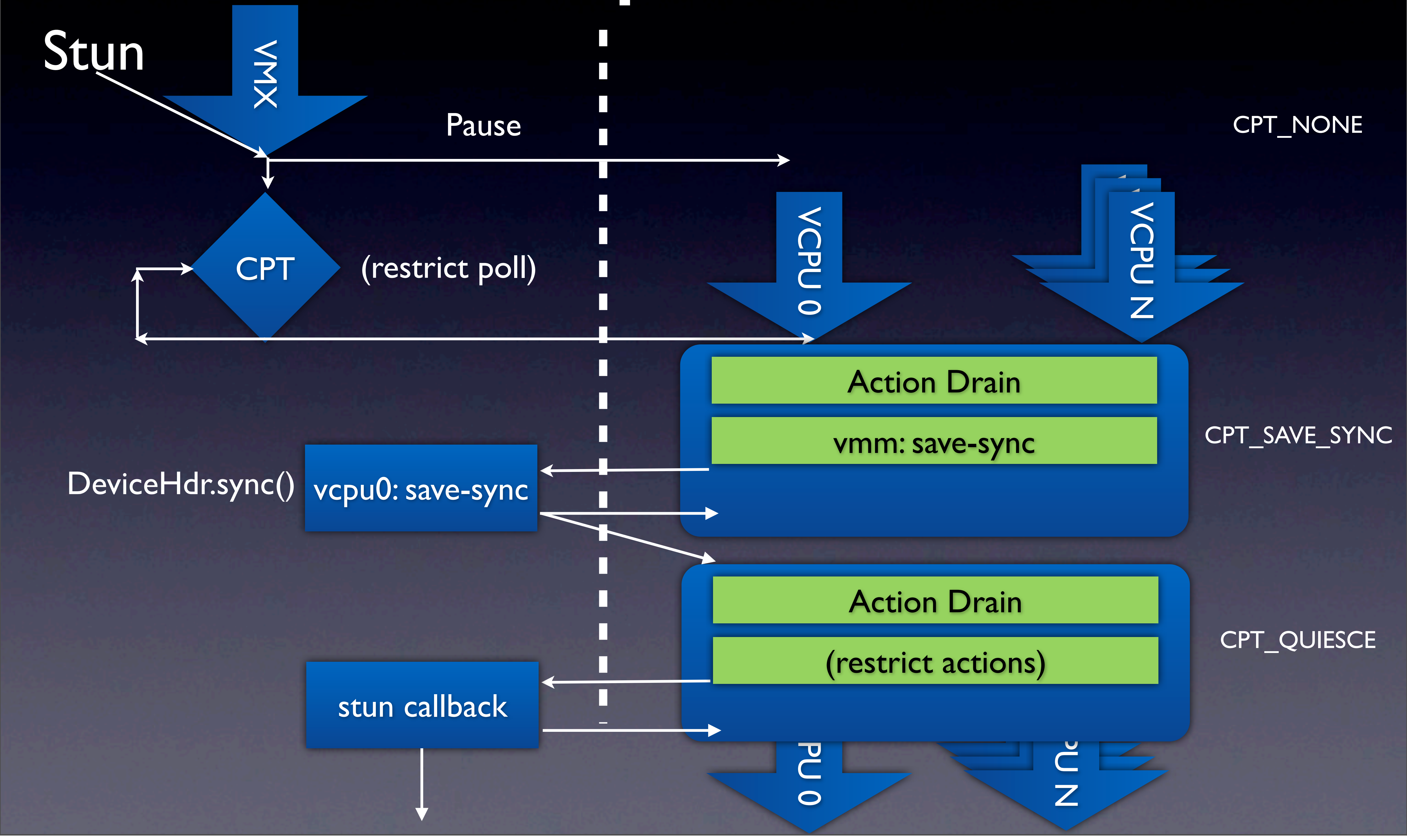
Initialization takes seconds to minutes; length is determined by hostd load / VMHS+VMDB overheads (4 threads for 100s of VMs) and storage latency. Times out after 3 minutes.

Precopy can take as long as vmkernel wishes to take. 256GB VM / 10Gbps -> 1’s of minutes idle, 10s of minutes loaded. Also, 1-minute timeout if network drops.

Downtime is fast: mostly CPU-bound (~1/2 second of straight-line code), dominant cost is size of (non-precopied parts of) checkpoint, and VMFS overhead of opening .vmdk files (optimized by OID cache) / reading .vmdk metadata. 1 second typical, >5s rare. 60s timeout for network failure.

Comment about timeouts: there is pressure to keep timeouts low, timeouts directly affect failure-path latency for vMotions / High Availability restart times.

Checkpoint Stun



Wednesday, August 18, 2010

Checkpoint stun is the process of quiescing all VM activity, either for saving a checkpoint or for a major change to runtime state (e.g. device hot-add).

First step is pausing VMM to stop guest progress as immediately as possible.
- if posted by VMX, stops at next (IPI-accelerated) action drain
- if posted by VCPU, stops at next guest instruction

Next is restricting the poll loop to POLL_CLASS_CPT, which means “only fire poll callbacks that move checkpoint forward). Notably excluded: POLL_CLASS_MAIN (device events) and POLL_CLASS_PAUSE (external APIs / VMDB / Foundry).

After poll becomes restricted, no new external events should show up. (E.g. no new network packets, no new mouse movements.)

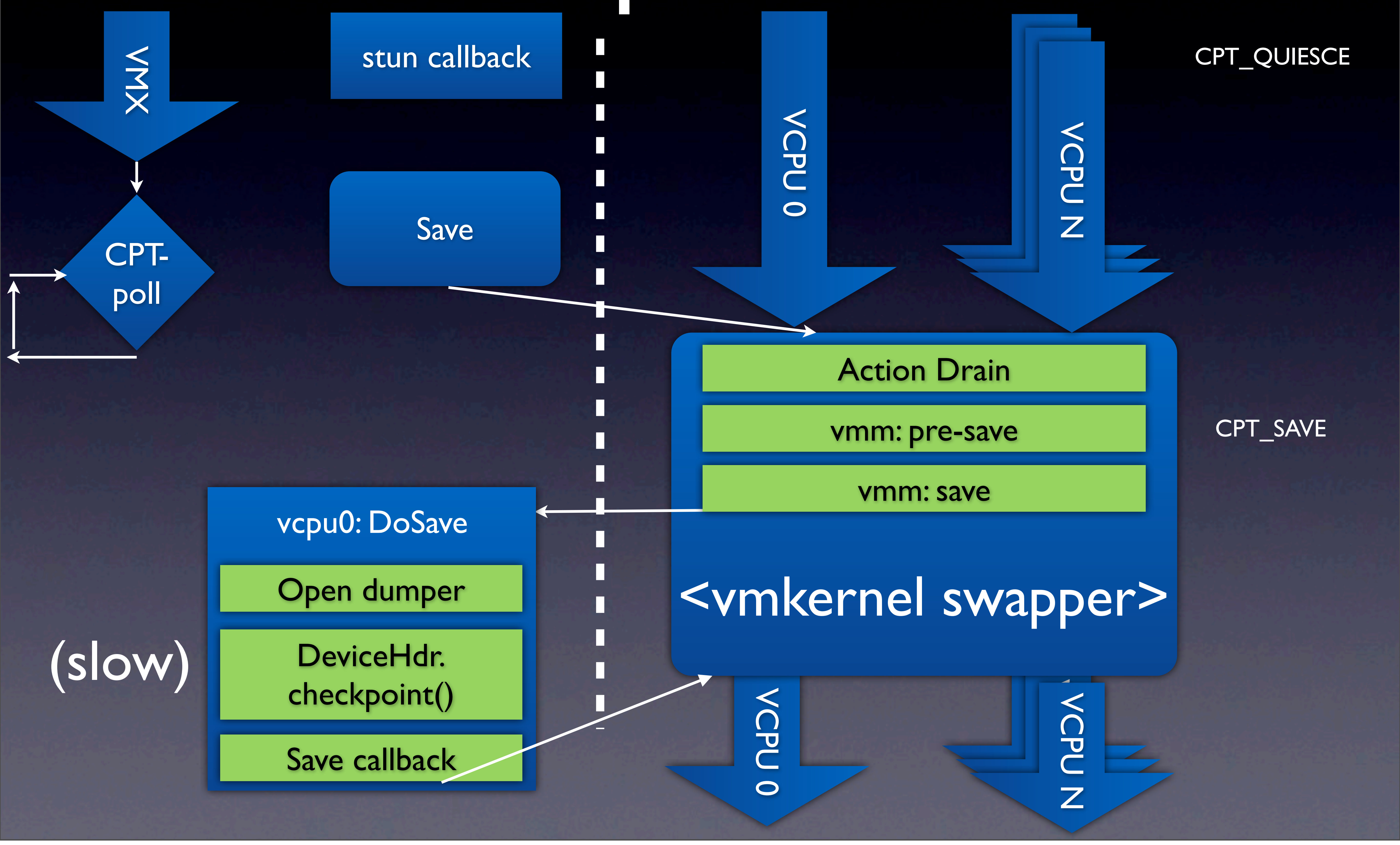
Then, NOP gets posted to the monitor to cause it to shift into CPT_SAVE_SYNC. VMM drains all pending actions once, then does ST_Stop to join all VCPUs, then runs sync handlers in monitor. Then vcpu0 calls userlevel and runs sync handlers in vmx. VMM sync handlers are very fast; VMX sync handlers are dominated by cost of blocking until I/Os complete.

Save-sync should complete all “pending” events - after save-sync completes, additional OS-supplied completions are not allowed.

Then back to the monitor, which continuously drains actions until no restricted actions are present. This is “quiesced”. After, a restriction mask is installed, and we call to userlevel to report “stun” completed.

Loosely, this sequence restricts out various sources of event:
- poll restricts new events
- save-sync restricts completion of existing events

Checkpoint Save



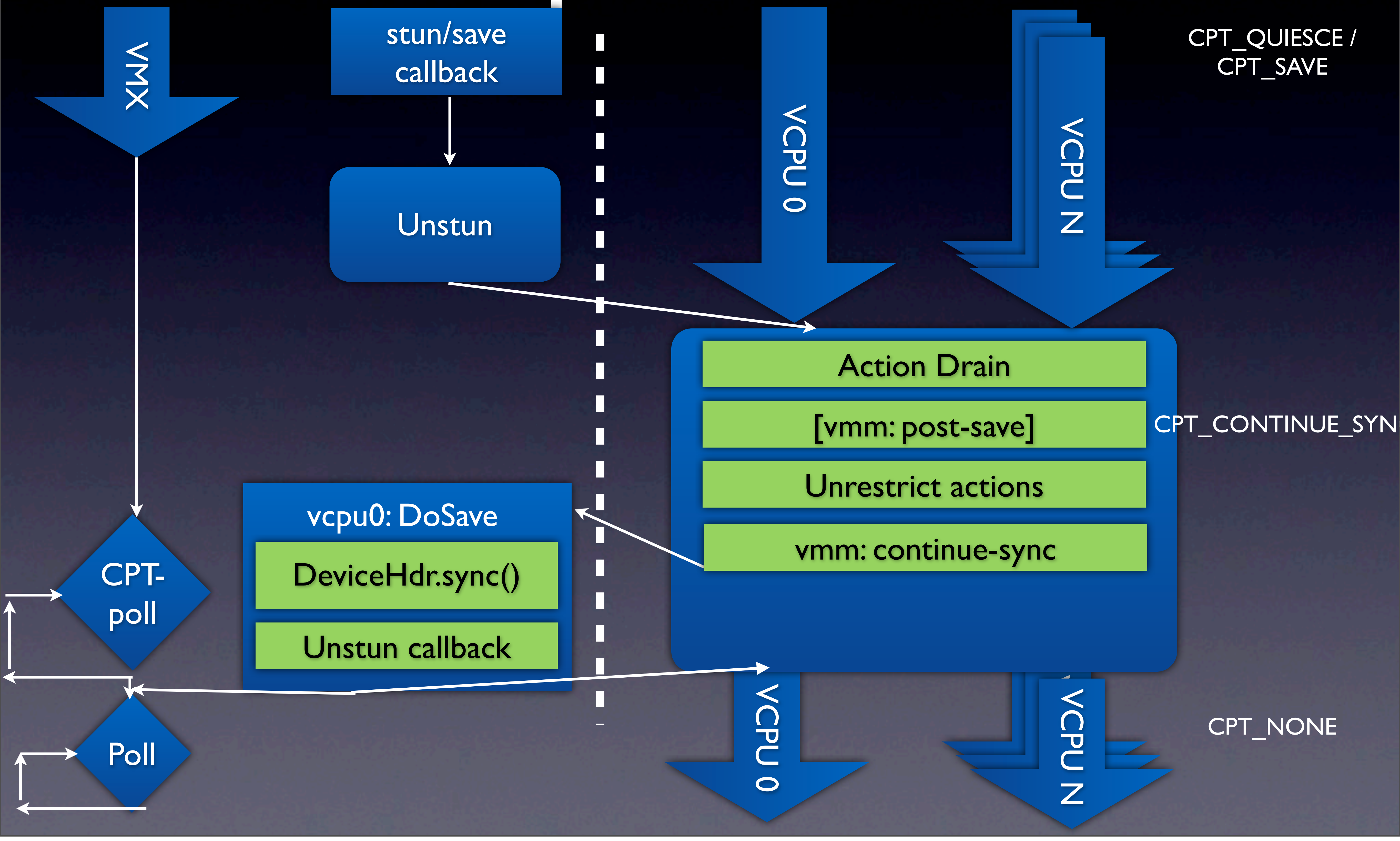
Wednesday, August 18, 2010

Stun callbacks could unstun immediately, but more likely to call Checkpoint_DoSave to actually write out the checkpoint. Starts by switching to VMM to do "pre-save", which unpins all guest memory and (for ESX) switches to vmkernel swapper. Effectively, detaches vmm from guest memory for duration of the checkpoint, flushing all caches in the process.

The VMM runs save handlers; these mostly copy state out of monitor into shared area, so that userlevel save handler can write them into checkpoint. The checkpoint file is only opened at this point. Then the save callback fires.

Wrinkle: lazy-save may also occur. This means a device reports success, but the checkpoint is left open so that device can continue writing to file in background while guest starts running.

Checkpoint Unstun



Wednesday, August 18, 2010

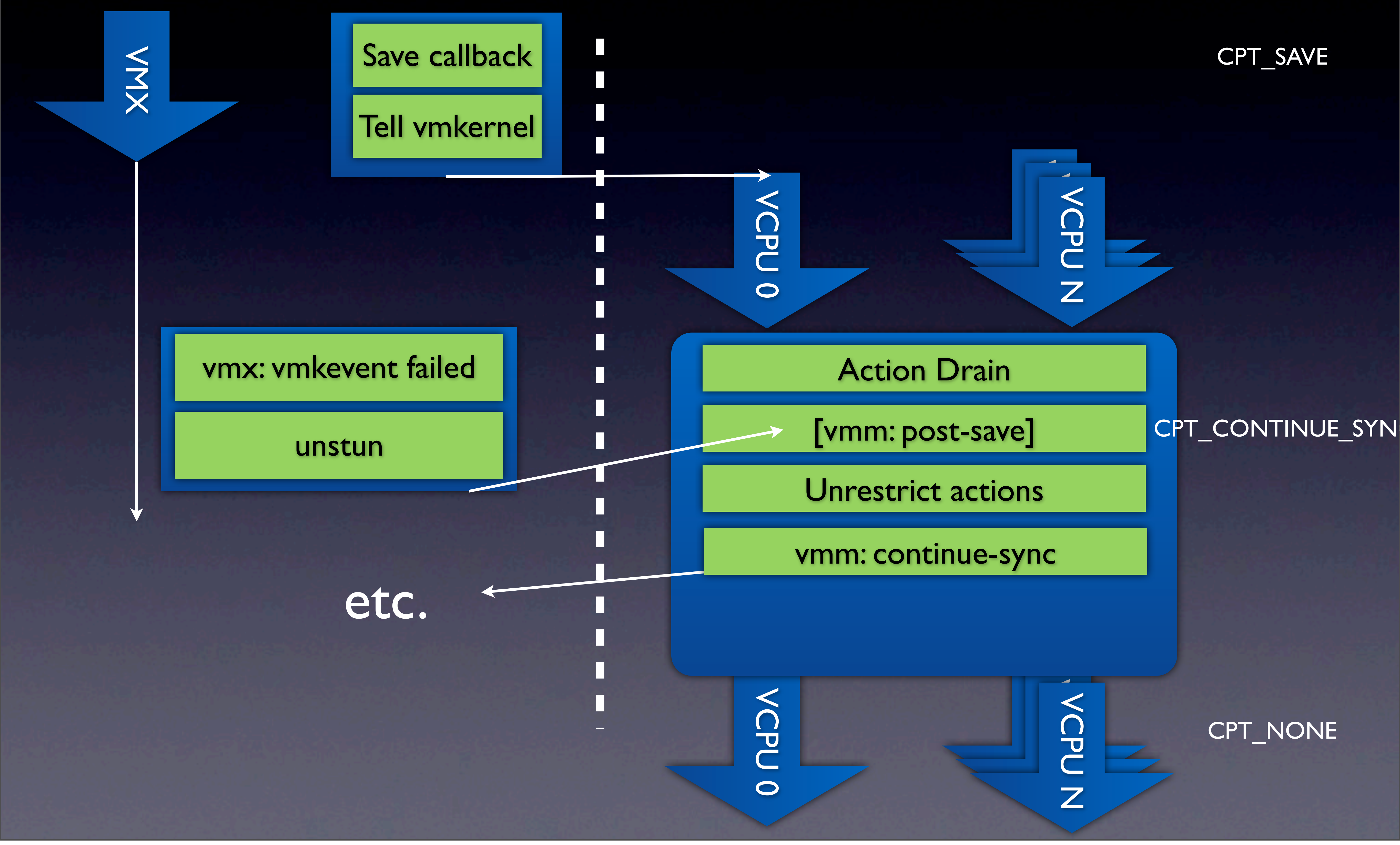
Generally, the stun or save callback triggers an unstun.

This posts a NOP to trigger a monitor change. If VMM pre-saved, then there is a post-save to re-establish key monitor caches and pinned pages. Once caches are re-established, action restriction is lifted.

Then continue-sync runs in the monitor (ST_Stopped to be synchronized); this attaches / synchronizes monitor-side devices. (Many call vmkernel to reconnect here). After, continue-sync runs at userlevel to connect non-monitor devices.

Once the unstun callback returns, vcpu threads can immediately start running guest instructions.

vMotion Source

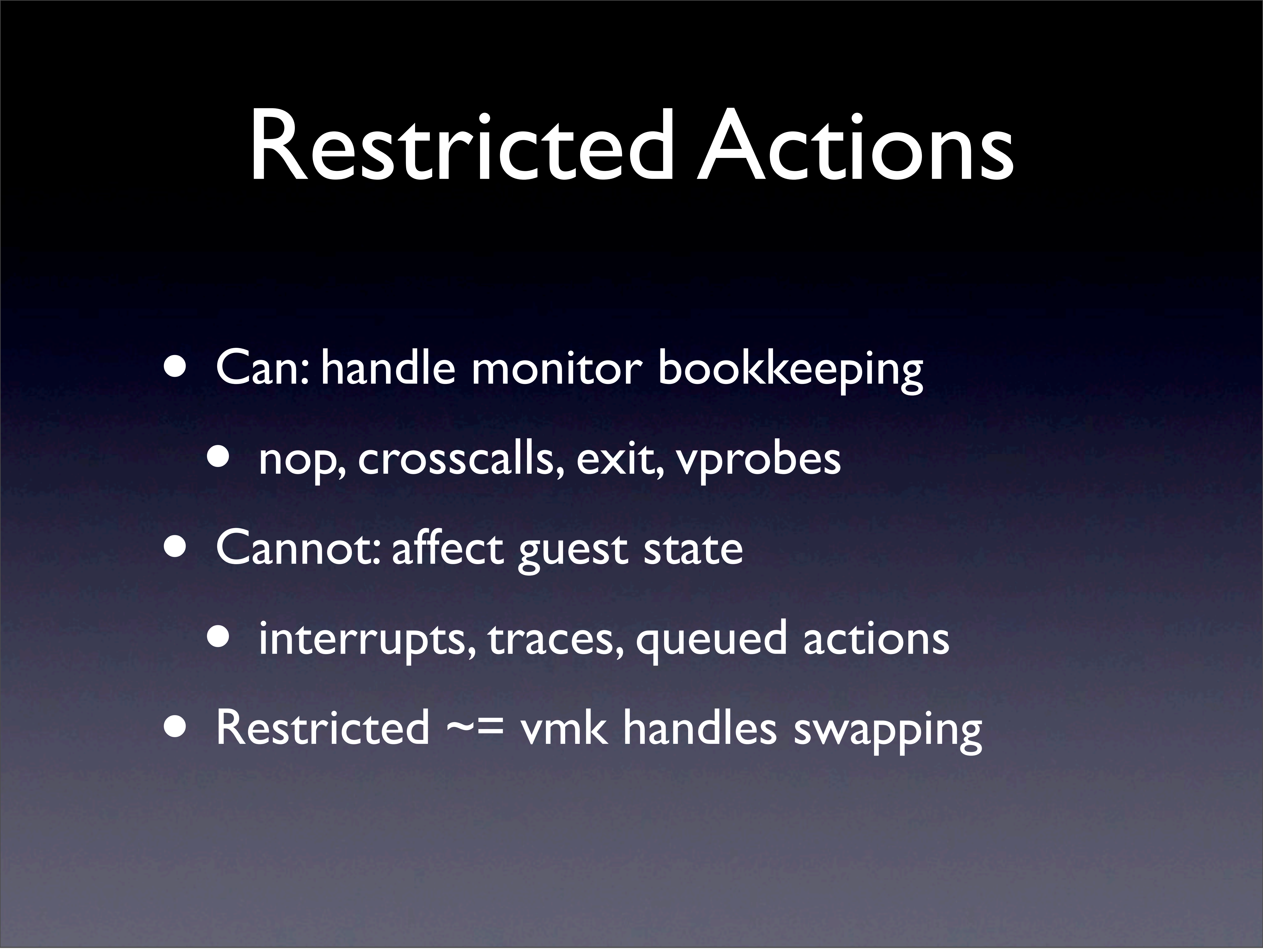


Wednesday, August 18, 2010

vMotion has a quirk on the source. The save callback just returns; vcpus go back to draining (restricted) actions and vmx stays in (restricted) poll. Eventually vmkernel reports either success or failure. On success, poweroff (much like suspend). On failure, unstun.

Note that there are no blocked threads while waiting for vMotion result.

General comment: while a Migrate is in progress, Checkpoint cannot be used for anything else. (No GoLive, no snapshot-consolidate, etc.)



Restricted Actions

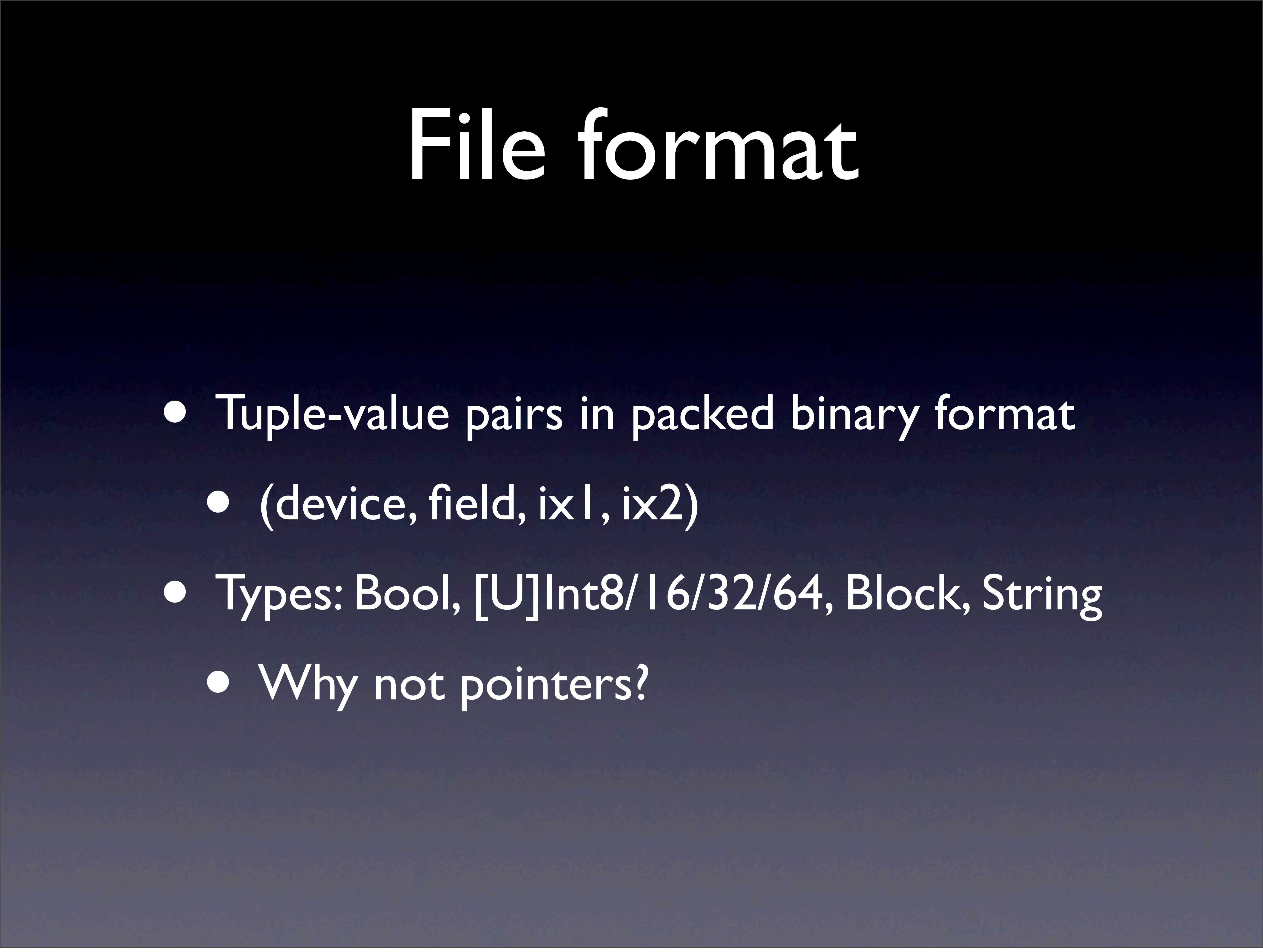
- Can: handle monitor bookkeeping
  - nop, crosscalls, exit, vprobes
- Cannot: affect guest state
  - interrupts, traces, queued actions
- Restricted ~= vmk handles swapping

Wednesday, August 18, 2010

What is a “restricted action”?

Loosely, any action that affects guest-visible state is disallowed during action restriction, because the checkpoint would be inconsistent if state were changed while being written.

Restriction is loosely (though not exactly) analogous to vmk-swapper being active: things that require pinned guest memory are disallowed.



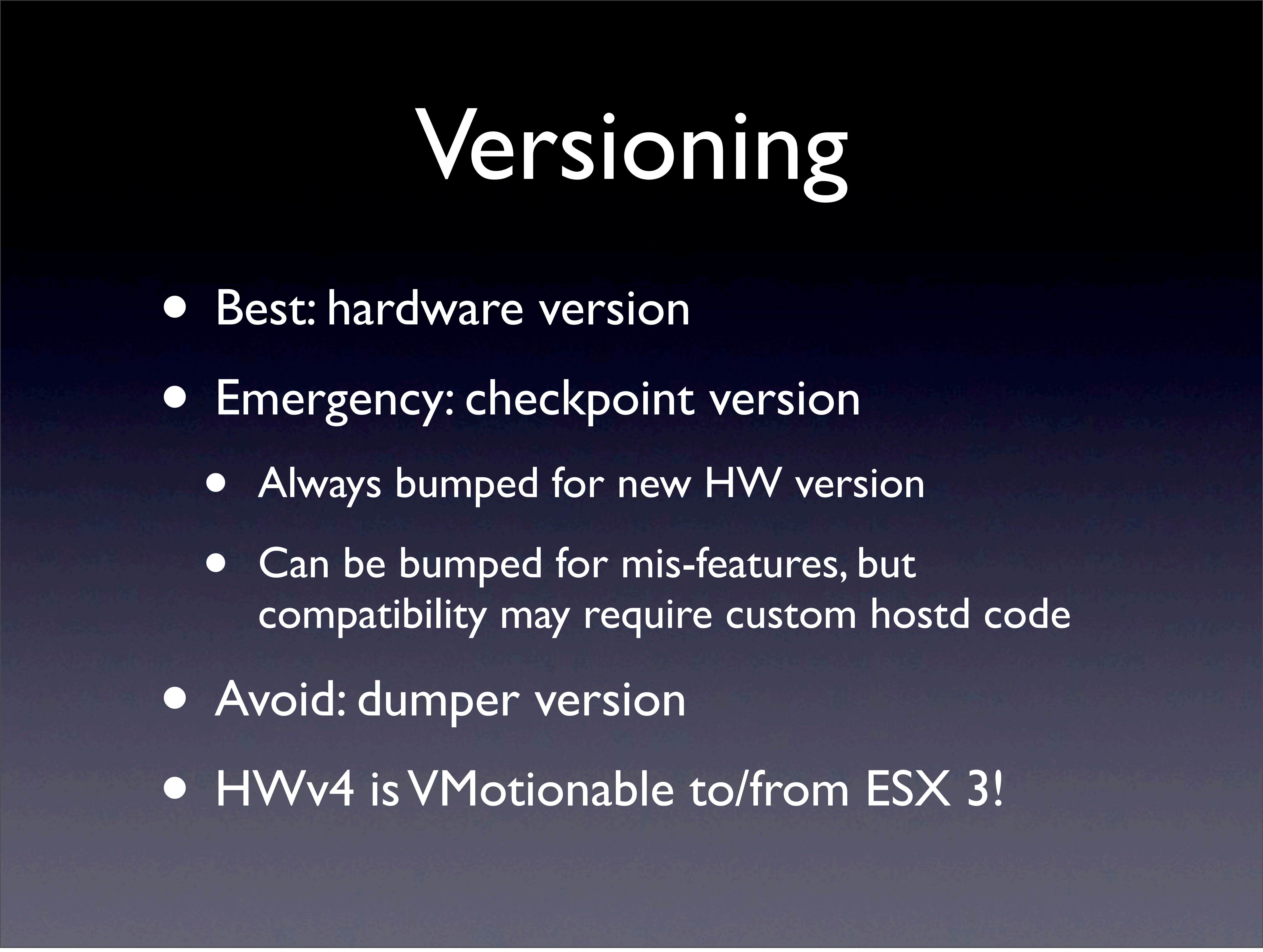
File format

- Tuple-value pairs in packed binary format
  - (device, field, ix1, ix2)
- Types: Bool, [U]Int8/16/32/64, Block, String
  - Why not pointers?

Wednesday, August 18, 2010

Checkpoints are stored in a packed binary format, indexed by tuples.

Only primitive types are stored in a checkpoint. Complex types (structures) change!



Versioning

- Best: hardware version
- Emergency: checkpoint version
  - Always bumped for new HW version
  - Can be bumped for mis-features, but compatibility may require custom hostd code
- Avoid: dumper version
- HWv4 is VMotionable to/from ESX 3!

Wednesday, August 18, 2010

Checkpoint versioning and compatibility

THERE ARE NO AUTOMATED TESTS! Even QA has difficulty testing; we mostly ensure compatibility via careful code review.

If checkpoint format changes:
- prefer to write out old and new format, read old or new format. (Old format can be deleted in ~5 years.)
- if VM would fail to resume on old VMX (VMX panic or guest BSOD/crash), bump checkpoint version. This is a check of last resort: vSphere / VC (and especially DRS) do not know about such requirements so will attempt VMotions which will fail. Avoiding ugly failures requires custom hostd code; ~2 such cases exist. Don't do it.
- Usually OK to write (a little bit) more data.

Current compatibility constraints: HWv4 VMs can be VMotioned to/from ESX3. HWv7 VMs can be VMotioned to/from ESX4. HWv8 VMs can assume ESX5 (M/N).

Questions?