Seamless Kernel Update

https://gitee.com/openeuler/nvwa

Longjun Luo (luolongjun@huawei.com)

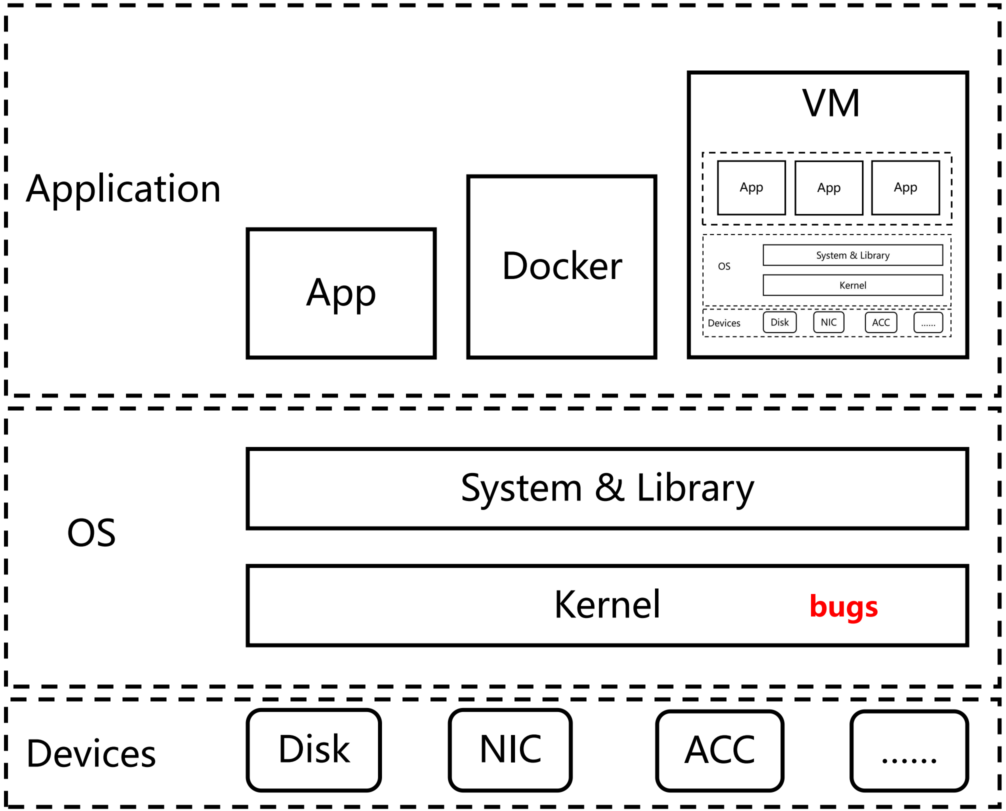
OpenEuler ops-sig

Contents

01 Background

- Present Situation

Background



Kernel Bugs

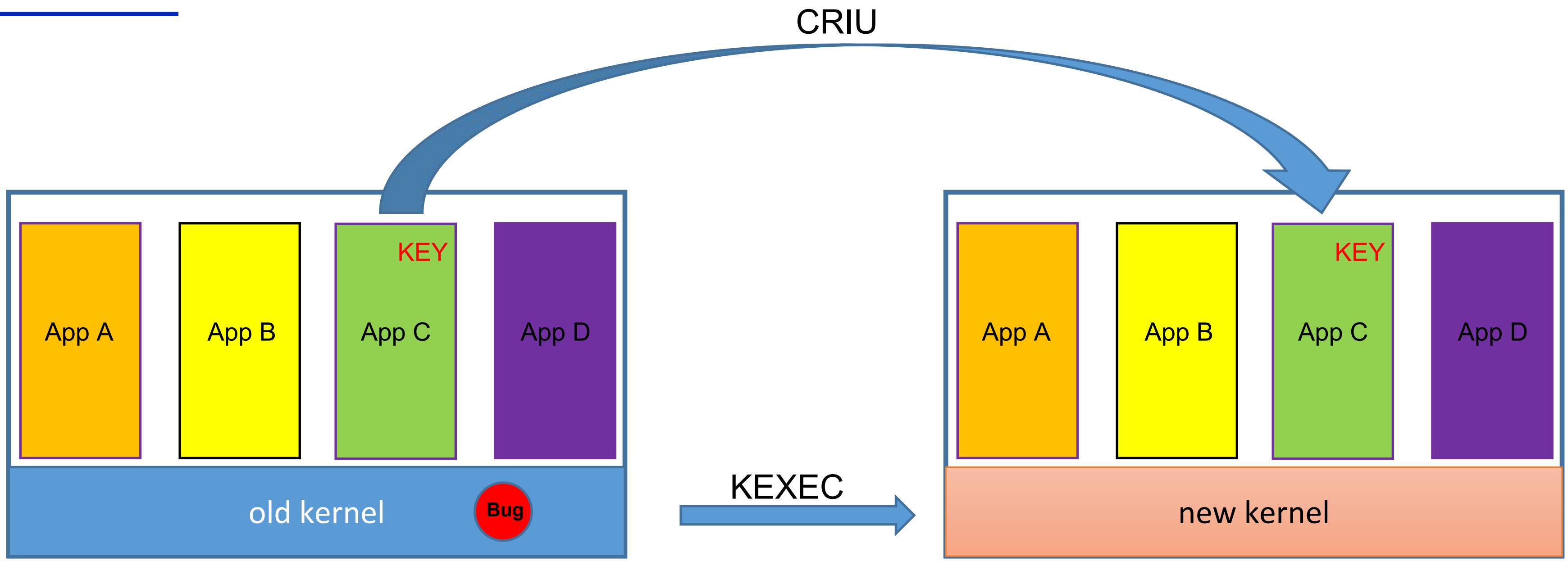
- Kernel live patch to fix the bug
  - No live patch for some bugs
  - Require significant programming effort
- Live migration for APP/VM & Reboot
  - No method for the pass-thought device
  - Difficult to transmit large memory

For Example

Machine: Bare-Metal Server Kunpeng 920
Memory： 380G
Application： MySQL (DB)

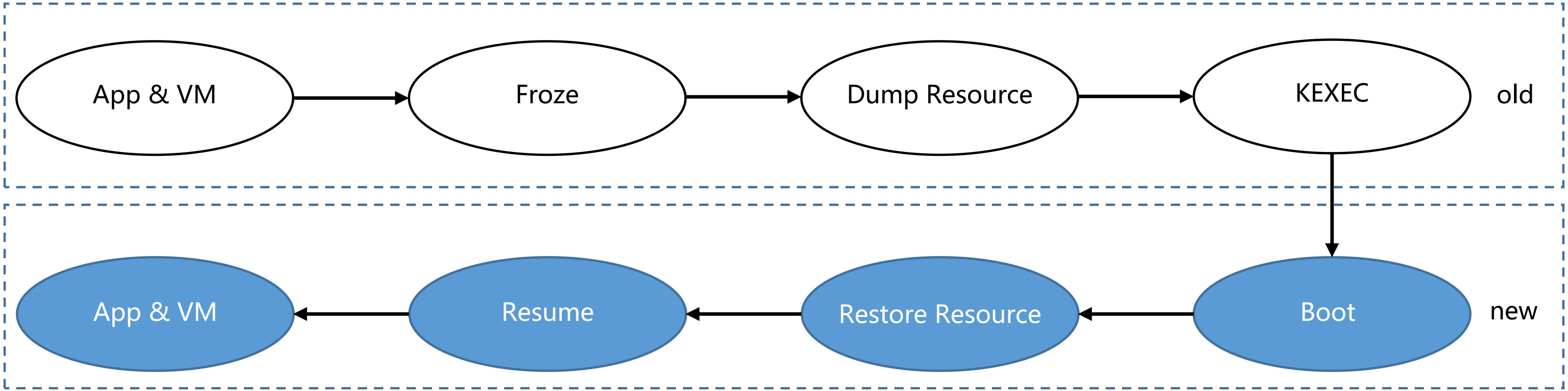
* Difficult to transmit large memory data

Background



- Time cost is unacceptable
- Limited support for kernel driver

Acceleration

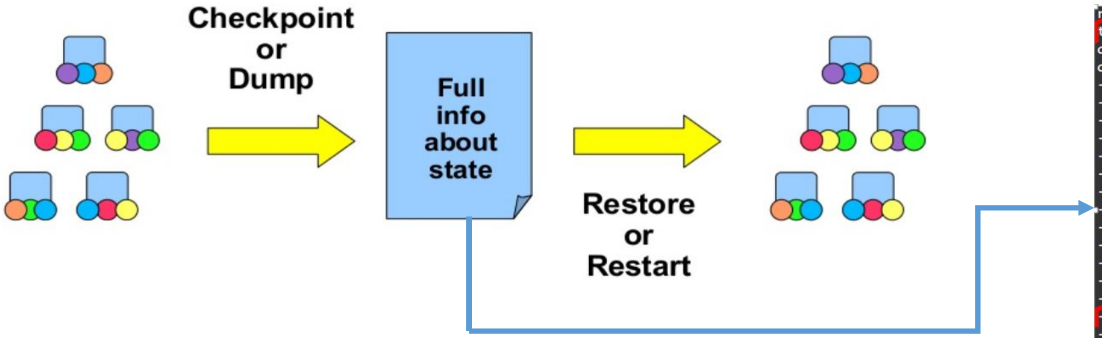


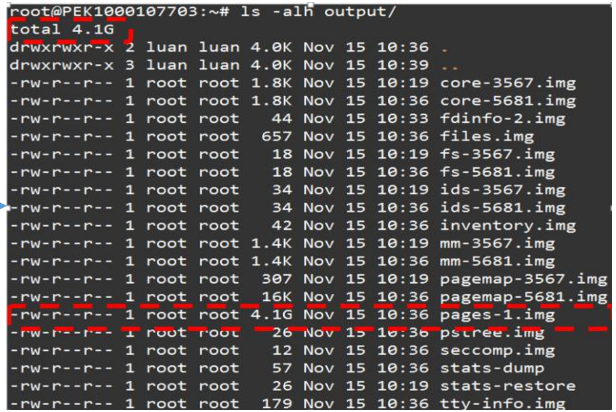
* time cost = dumping time + kernel switch time(KEXEC) + restoration time

02 Froze/Resume Application

- Pin Process Memory
- Pin Kernel Memory

Keep memory

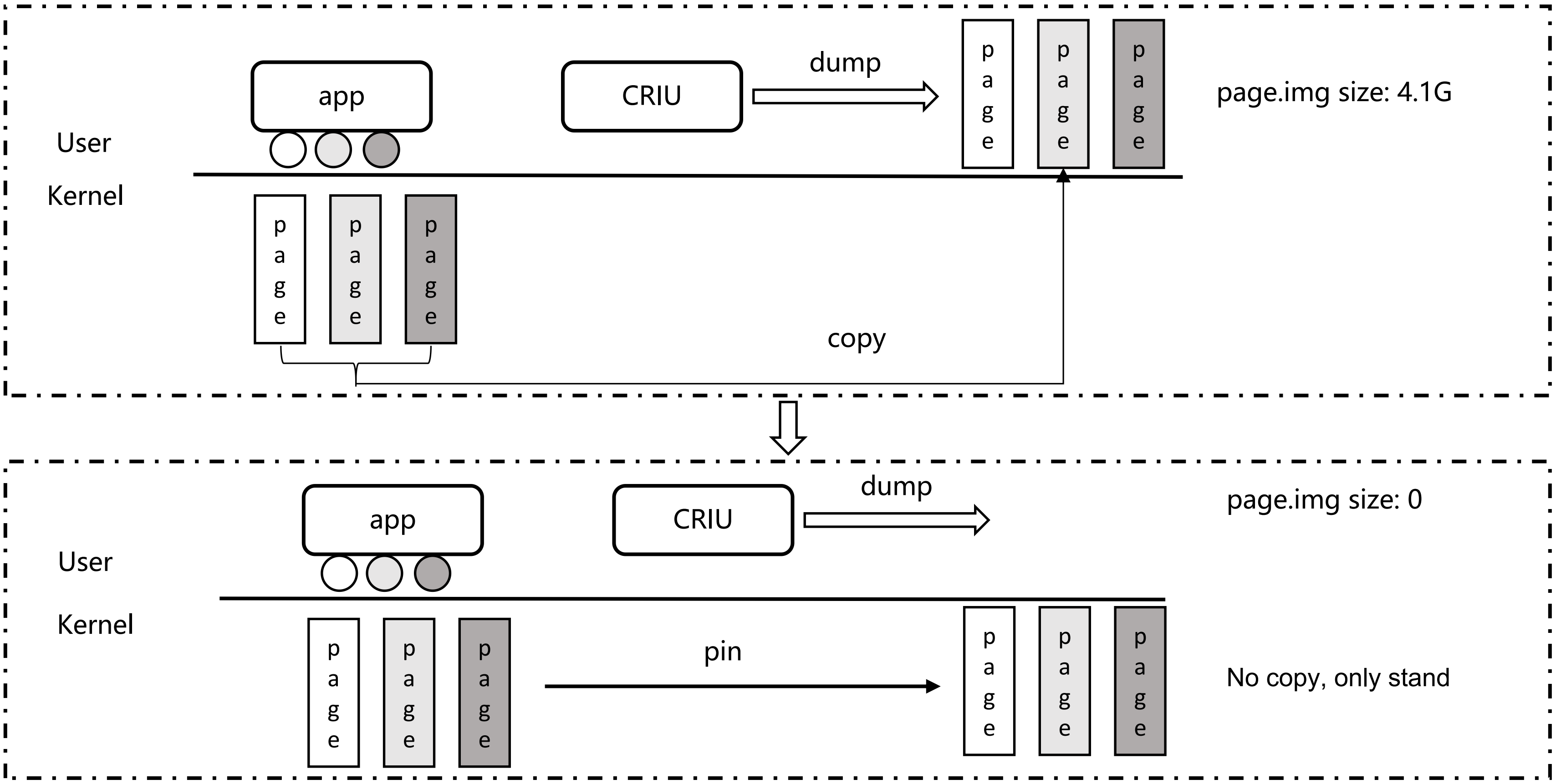




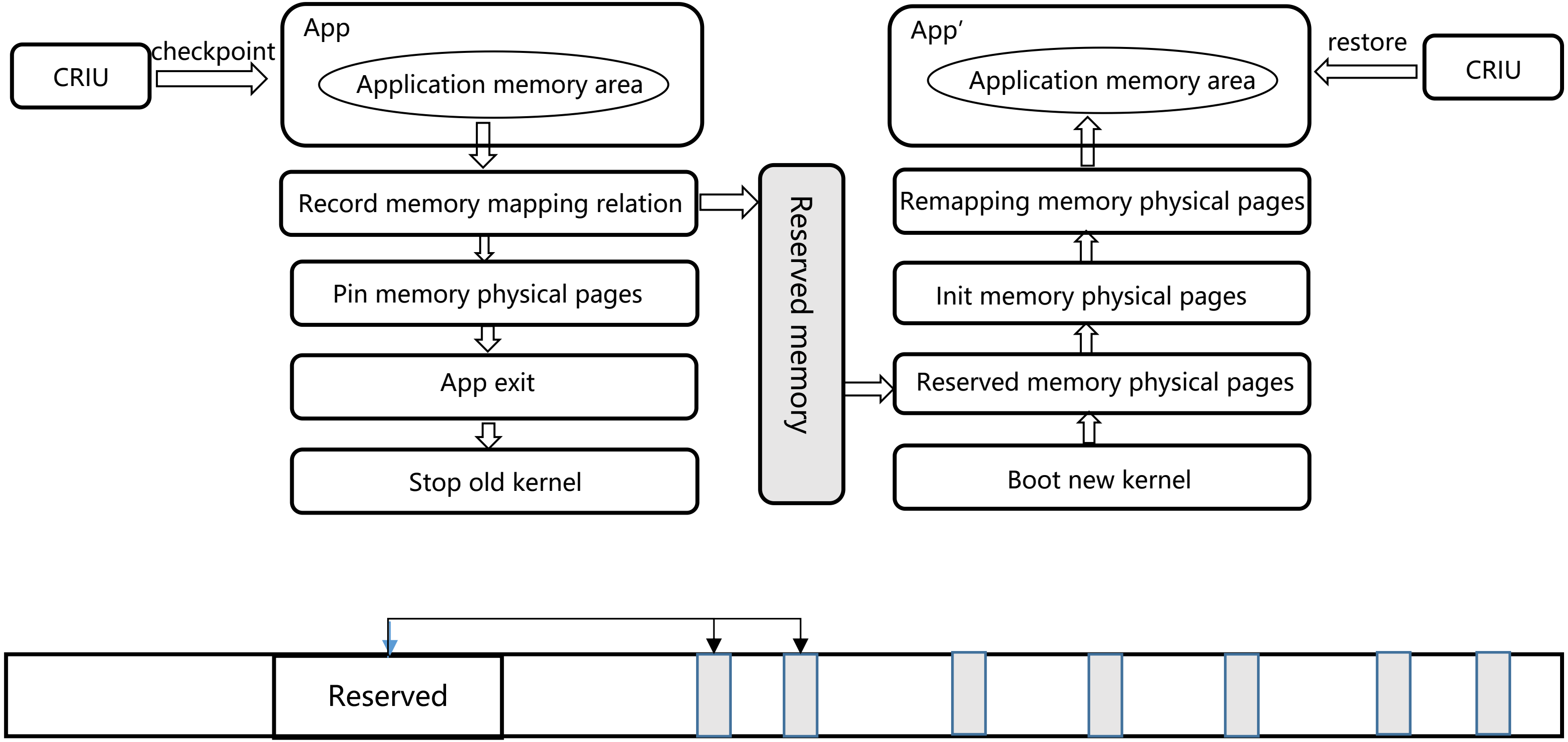
CRIU writes the copy of application memory into disk file for restoring the application. When the data is large, the copy operation will cost too much time.

In order to avoid copy and read operation, we can keep the memory(pin memory) and remap the memory to the restoring task.

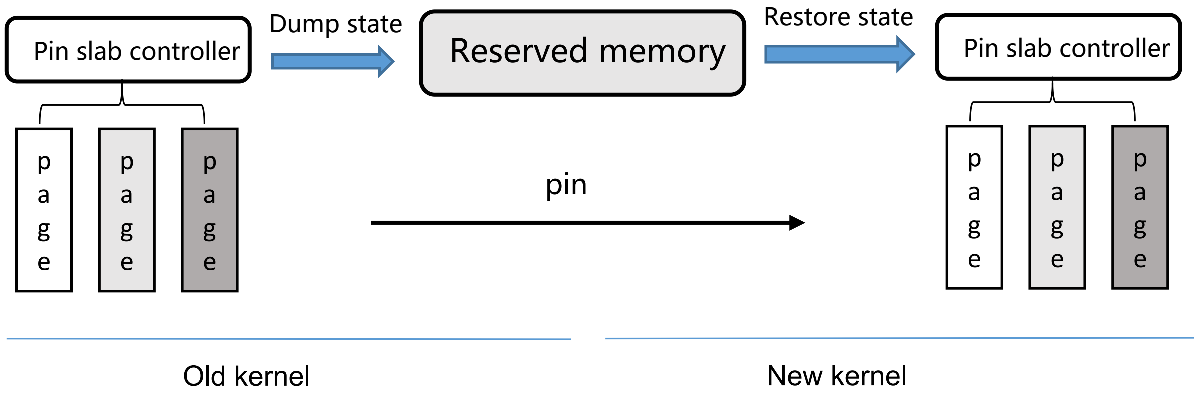
Pin application memory



Keep user memory unchanged in new kernel



Keep kernel memory unchanged in new kernel



* Create a pin slab controller to manage the old kernel pages which need to keep constant while booting the new kernel.

* Support kmalloc/vmalloc.

03 Kernel Fast Reboot

- CPU Park
- Preload and Decompress kexec Images
- Defer and parallelize Initialization

CPU Park

➢ process

S1：Reserve Memory

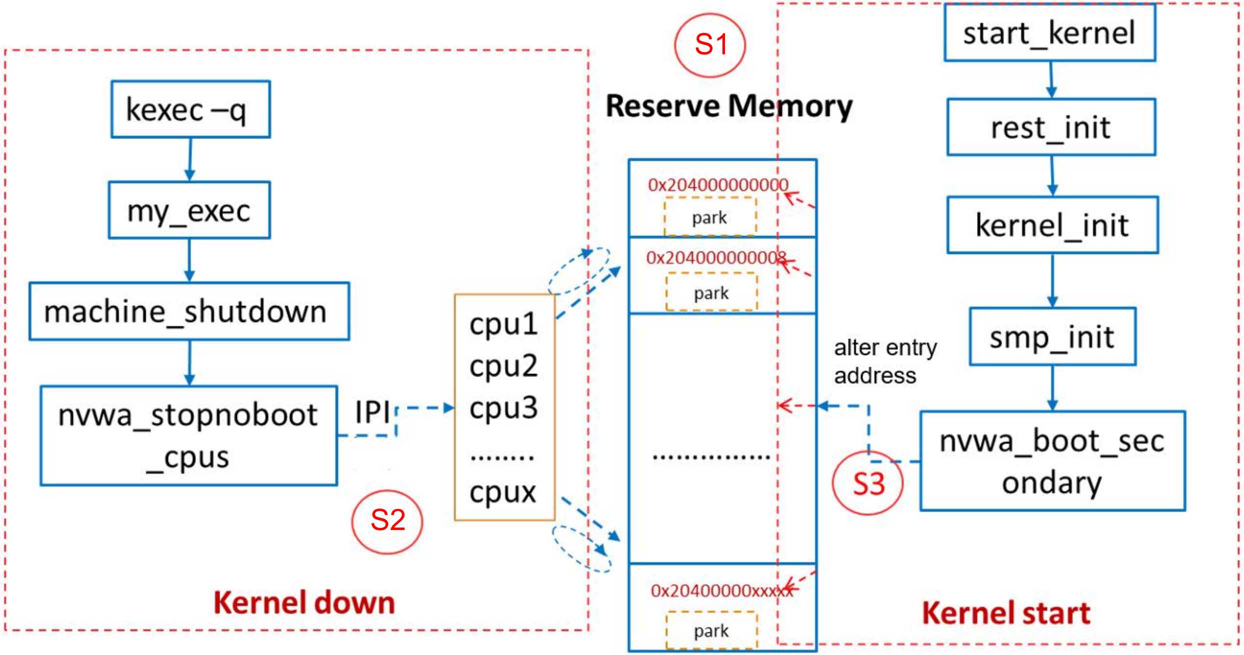
Reserve some space for CPU park code and data.

S2：Kernel Down

When APs get reboot IPI, they will execute CPU park code where APs will spin and wait on an address.

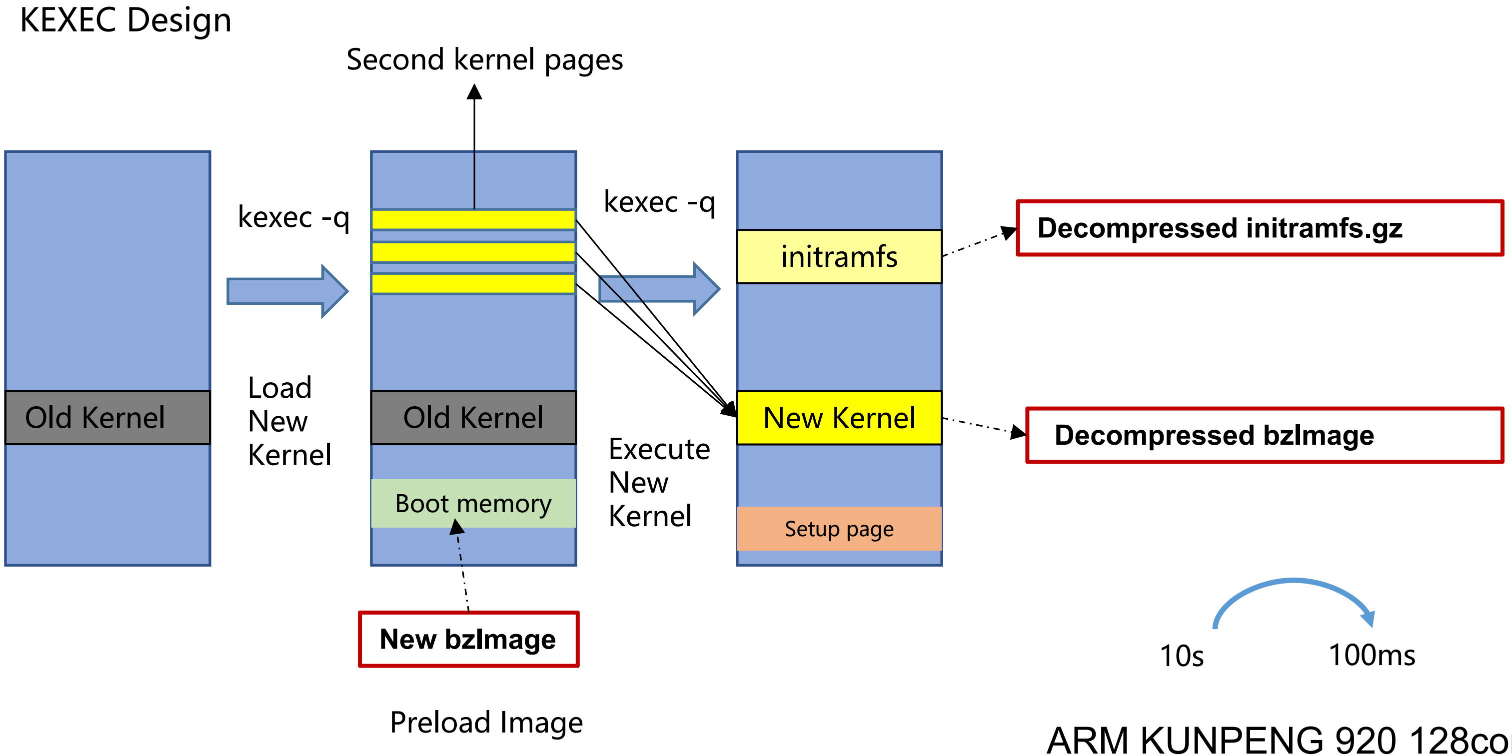
S3：Kernel Up

After BSP reboot successfully, it will wake up APs by writing an entry address to the spin-read address so that APs can jump to the normal boot-up procedure.



	X86	ARM	ARM(nvwa)
Startup time(s/per cpu)	0.003~0.004s	0.03~0.04s	**0.0003 s**
ARM(nvwa)/other	1/10+	1/100+	**1**

Preload and decompress kexec images



ARM KUNPENG 920 128core

Defer and parallelize initialization

commit e44431498f5fbf427f139aa413cf381b4fa3a600

Author: Daniel Jordan <daniel.m.jordan@oracle.com>

Date: Wed Jun 3 15:59:51 2020 -0700

mm: parallelize deferred_init_memmap()

Deferred struct page init is a significant bottleneck in kernel boot. Optimizing it maximizes availability for large-memory systems and allows spinning up short-lived VMs as needed without having to leave them running. It also benefits bare metal machines hosting VMs that are sensitive to downtime. In projects such as VMM Fast Restart[1], where guest state is preserved across kexec reboot, it helps prevent application and network timeouts in the guests.

On Josh's 96-CPU and 192G memory system:

Without this patch series:
[0.487132] node 0 initialised, 23398907 pages in 292ms
[0.499132] node 1 initialised, 24189223 pages in 304ms
...
[0.629376] Run /sbin/init as init process

With this patch series:
[0.231435] node 1 initialised, 24189223 pages in 32ms

		kernel boot			deferred init		
node%	(thr)	speedup	time_ms	(stdev)	speedup	time_ms	(stdev)
	(0)	--	4089.7	(8.1)	--	1785.7	(7.6)
2%	(1)	1.7%	4019.3	(1.5)	3.8%	1717.7	(11.8)
12%	(6)	34.9%	2662.7	(2.9)	79.9%	359.3	(0.6)
25%	(13)	39.9%	2459.0	(3.6)	91.2%	157.0	(0.0)
37%	(19)	39.2%	2485.0	(29.7)	90.4%	172.0	(28.6)
50%	(26)	39.3%	2482.7	(25.7)	90.3%	173.7	(30.0)
75%	(39)	39.0%	2495.7	(5.5)	89.4%	190.0	(1.0)
100%	(52)	40.2%	2443.7	(3.8)	92.3%	138.0	(1.0)

*Intel(R) Xeon(R) Platinum 8167M CPU @ 2.00GHz (Skylake, bare metal)
2 nodes * 26 cores * 2 threads = 104 CPUs
384G/node = 768G memory

Defer and parallelize initialization

commit d1c3414c2a9d10ef7f0f7665f5d2947cd088c093
Author: Grant Likely <grant.likely@secretlab.ca>
Date: Mon Mar 5 08:47:41 2012 -0700

drivercore: Add driver probe deferral mechanism

Allow drivers to report at probe time that they cannot get all the resources required by the device, and should be retried at a later time.

This should completely solve the problem of getting devices initialized in the right order. Right now this is mostly handled by mucking about with initcall ordering which is a complete hack, and doesn‘t even remotely handle the case where device drivers are in modules. This approach completely sidesteps the issues by allowing driver registration to occur in any order, and any driver can request to be retried after a few more other drivers get probed.

module.async_probe [KNL]
Enable asynchronous probe on this module.

driver_async_probe= [KNL]
List of driver names to be probed asynchronously.
Format: <driver_name1>,<driver_name2>...

deferred_probe_timeout=

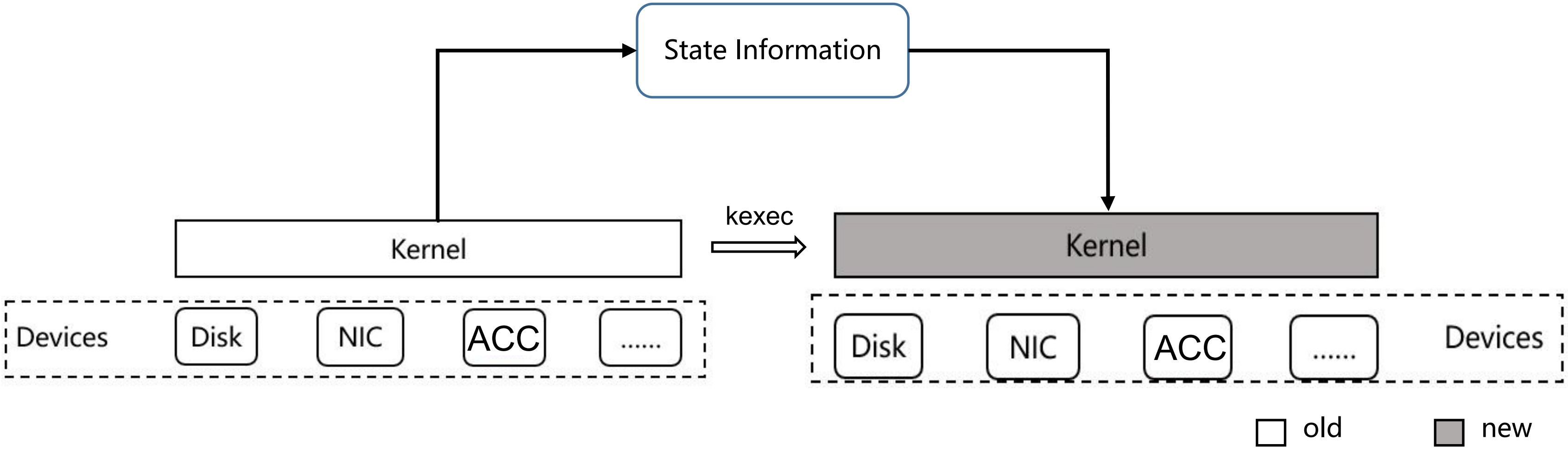
[KNL] Debugging option to set a timeout in seconds deferred probe to give up waiting on dependencies to probe. Only specific dependencies (subsystems or drivers) that have opted in will be ignored. A timeout of 0 will timeout at the end of initcalls. This option will also dump out devices still on the deferred probe list after retrying.

*Deferring device driver probe can only defer the entire device initialization , however, the kernel booting only uses part device function at sometime , we can defer the other part of the device initialization.

04 Keep Device State

- Keep PCI Device
- Keep Driver State

Keep Device State



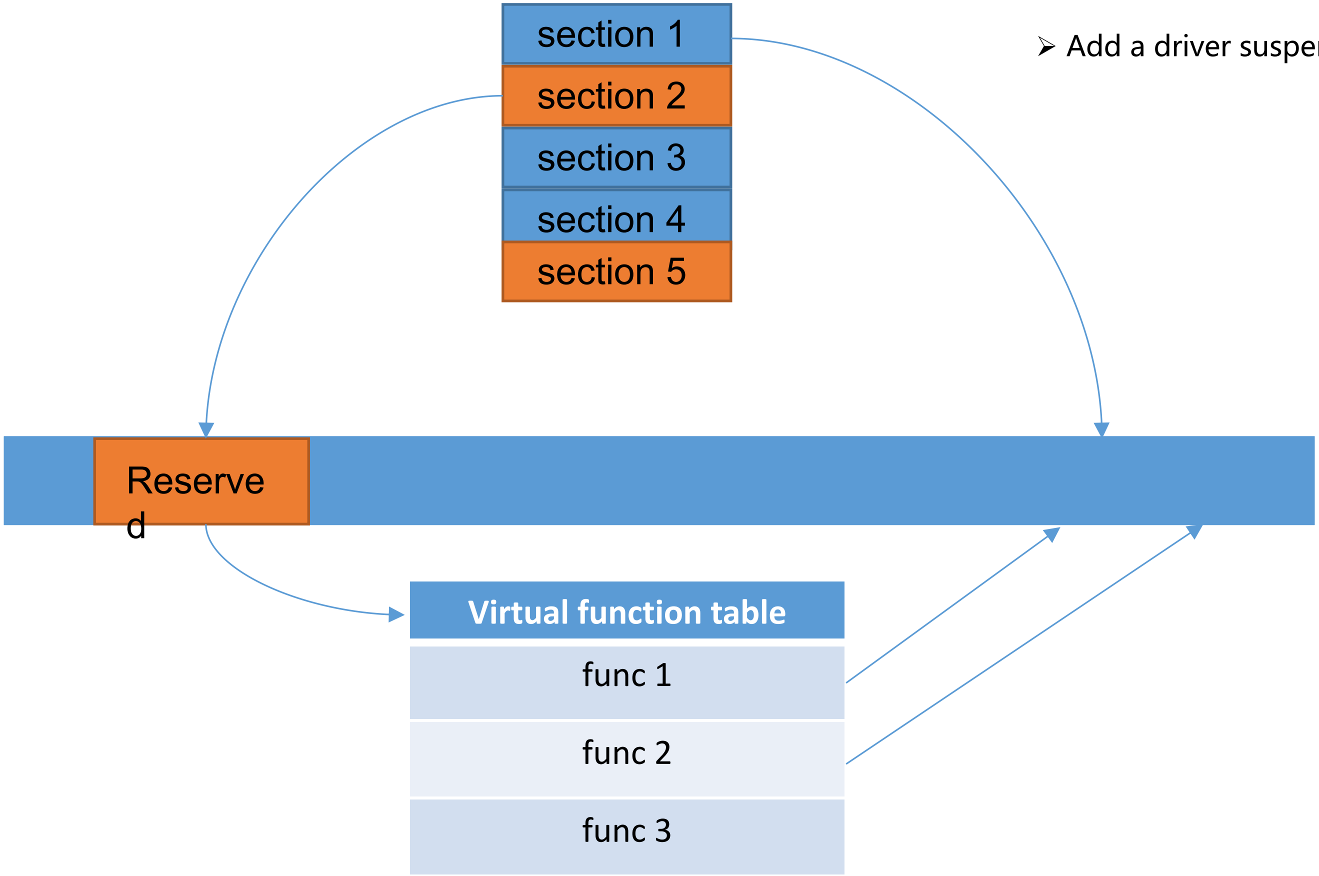
- Device state information
  - position in PCI tree
  - BAR resource usage
  - IRQ and DMA config
  - memory, threads, kernel objs …

Keep PCI Device

- Skip PCI Enumeration
- Restore PCI tree from old kernel
- Do not read/write HW registers
- Restore BAR resource allocated from old kernel
- Skip HW reset and device init
  - Keep device alive
- Reload IRQ and DMA config from old kernel
- Reload memory and IO mapping

Keep Driver State —— Try and Challenge

- Isolated kernel driver from kernel space partly
- Add a driver suspend/restore API



05 Future Attempt

- Conclusion
- Development plan

Conclusions

- Save/restore status in memory or keep resources in memory directly
- Rewrite of driver code is necessary
- Many work needed to accelerate linux reboot process

Development plan

- A new and easy-to-use tool to checkpoint/restore apps (kernel module + userspace)
- Combine livepatch and seamless kernel update
- Standard and universal method for drivers save/restore
- Support for VM host update natively

NVWA Developers

Lin Fu (fulin10@huawei.com)
Longjun Luo (luolongjun@huawei.com)
Lin Zhu (zhuling8@huawei.com)
Jingxian He (hejingxian@huawei.com)
Jianhai Ruan (luanjianhai@huawei.com)
Yan Sang (sangyan@huawei.com)