Challenges Revisited

Supporting

Virtual CPU Hotplug

On

Architectures That ~~Support~~ CPU Hotplug

(like ARM64)

Challenges & Workarounds In Qemu

https://kvm-forum.qemu.org/2023/talk/9SMPDQ/

A Follow-up Presentation of KVMForum 2020: https://sched.co/eE4m

Salil Mehta (Huawei, Cambridge, UK)

James Morse (ARM, Cambridge, UK)

Date: 15th June 2023, 15:45–16:15 (Europe/Prague/Brno), Room 1

KVM Forum 2023

HUAWEI

Overview

Motivation: of Supporting VCPU Hot-plug on ARM like Platforms

- ❑ Required by Cloud Vendors as part of the orchestration framework which could adjust resource requests (CPU and Mem requests) for the containers in a pod, dynamically based on usage or SLA(Pay-as-you-grow).

- ❑ Should have look-and-feel of x86_64 platform VCPU hot-plug feature

- ❑ For example, to support Kata-containers (Light Weight VMs) with Qemu/ARM64
  - o Kata Containers VM starts with a minimum amount of resources, (hot-plug everything)
    - ▪ allowing for **faster boot time** and
    - ▪ reduction in **memory footprint**.
  - o As the container launch progresses, devices are hot-plugged to the VM.

Terminology: From the ACPI Perspective

- ❑ **Possible** VCPUs = Maximum possible VCPUs in a VM
- ❑ **Present** VCPUs = which are present but further need to be 'enabled'. Cannot be used,
- ❑ **Enabled** VCPUs = which are 'present' and can be 'onlined' (PSCI) for use

Brief Anatomy: How Do We **Setup ARM VCPU** During VM Init? (1)

- **QOM ARM CPU** Object & **KVM VCPU** object
  - Allocation & Initialization
    - Various per-cpu features (PMU, SVE etc.)
  - Realization
    - Creation KVM VCPU (KVM_CREATE_VCPU)
      - Alloc of 'struct kvm_vcpu'
      - Initialization of various per-cpu data-structures
      - Setup Timer, PMU, debug etc
      - Config of SGIs, PPIs
      - Associated VGIC CPU interface initialization
    - Initialization of KVM VCPU (KVM_ARM_VCPU_INIT)
    - Qemu CPU Thread Exec (KVM_RUN)

Brief Anatomy: How Do We **Setup ARM GIC** During VM Init (2)

- **GIC** Initialization (**Qemu** & **KVM**)
  - Initialize GICV3 data structures (distributor, redistributor, cpu) in QEMU
  - KVM VGIC CPU interface initialization (KVM_CREATE_VCPU)
  - Initialize the GIC Distributor in KVM
    - Various SPIs, vITS alloc etc. (KVM_DEV_ARM_VGIC_CTRL_INIT)
    - IO Devs and memory regions – only once! (KVM_RUN)
      - Requires all KVM vCPU redistributors have been setup. (?)
      - vgic_v3_map_resources() and declare VGIC ready.
    - Base address (KVM_VGIC_V3_ADDR_TYPE_DIST)
  - Initialize the GIC redistributor in KVM
    - IO Devs, memory regions, Base address (KVM_VGIC_V3_ADDR_TYPE_REDIST_REGION)
  - Initialize GIC CPU Interface Register Info (like for ICC_CTLR_EL1) in Qemu Model
    - Required to get actual supported reg config from KVM

VCPU Hotplug Impediments: **KVM** Init **(Recap)**

1) All ARM VCPUs MUST be created in the Host KVM at VM Init
   - ❑ Number of VCPUs in the System cannot change after Init (ARM CPU Arch Req)
     - ❖ VCPU related features Init: Timers, PMU, Debug, private IRQs(= SGIs, PPIs...) etc. (KVM_CREATE_VCPU)
   - ❑ Sizing of VGIC in Host KVM can only happen once (Number of VCPUs) (ARM CPU Arch Req?)
     - ❖ Like VGIC Redistributor KVM iodevs, Memory Regions etc. (KVM_VGIC_V3_ADDR_TYPE_REDIST_REGION)
     - ❖ Association: VGIC Redistributor <=> VGIC CPU

2) VCPUs once created in KVM cannot be destroyed. (Limitation of KVM)

VCPU Hotplug Impediments: **Guest** Init **(Recap)**

1) Guest Kernel MUST see all ARM VCPUs specified by ACPI MADT GICC at Boot time (ARM CPU Arch Req)
   - ❑ Number of VCPUs in the Guest Kernel cannot change after Boot
   - ❑ Number of Redistributors exposed to Guest Kernel cannot change after boot
   - ❑ Association: VGIC Redistributor <=> VGIC CPU cannot change

2) VCPUs specified at Guest Kernel Boot by ACPI cannot be ‘un-plugged’ (ARM CPU Arch Req)

Workaround: to **KVM** Init Impediments **(Recap)**

1) **No workarounds!** Host KVM Initialization Should remain unchanged (Requirement from ARM Maintainers)
2) Fake VCPU Hot-{un}plug at VMM/Qemu level
3) Hence, No other option than to Create all KVM VCPUs at VM Init
   - Only Change power state of VCPUs When 'faking' hot-unplug in Qemu
4) Size GICV3 once
   - GIC Redistributor KVM iodevs, Memory Regions etc. – Yah! All of that!
   - Initialize GICR and GIC CPU Interface association in KVM once at VM Init
5) No need to destroy 'struct kvm_vcpu'. Keep them parked in powered down state

Workaround: to **Guest Kernel** Init Impediments **(Recap)**

What we Can't Do?

1) Number of Booted VCPUs & GIC redistributors Should remain unchanged (ARM Arch Req)

What we Can do?

2. Support to defer 'Online'ing' the VCPUs to a later time when Guest has booted (Details in James Morse ppt)
   - ❑ Introduce new 'online-capable' Bit in **ACPI MADT Table GIC CPU Interface** 'flag' field [1]
     - ❖ GICC = 'Enabled' (must be 'online'd' during boot) or
     - ❖ GICC = 'online-capable' (can be deferred 'online'ing' after Guest Kernel boots)
3. Enabled or Disabled CPU in Guest Kernel? evaluate(_STA) for CPU
   - ❑ Enabled CPU (In Guest) = Plugged VCPU (In Qemu) = _STA.PRESENT=1 _STA.ENABLED**=1 (ACPI Status)**
   - ❑ Disabled CPU (In Guest) = Un-Plugged VCPU (In Qemu) = _STA.PRESENT=1 _STA.ENABLED**=0 (ACPI Status)**

[1] ACPI specification 6.5 "Table 5.37: GICC CPU Interface Flags"

Long Story Short: Feels Like Hotplug. But It Isn't!

1. **Guest Kernel** (MADT/GICC, online-capable = 1)

- **Plugged** (cold, hot)? Present = 1 , **Enabled** = 1 (ACPI Status)
- **Unplugged** (cold, hot)? Present = 1, **Enabled** = 0
- **ACPI Enumerate** only **Plugged** i.e. register_cpu() ➡ sysfs
- Attempt **Online (via PSCI)** (Subject to VMM Policy)

Covered In Previous Next Slides
+
Accompanying James Morse's Slides

2. **Qemu** (Workarounds)

- **Realize** only **Plugged:** (cold, hot) vCPUs (Qemu Threads)
- **Park** KVM **VCPUs:** Unplugged QOM CPUs (KVM Power Off)
- **Lie** to the **Guest:** about the '**Presence**' of vCPUs
- **Fake Hotplug:** at QOM
- **Deny Online'ing:** Unplugged vCPUs (SMCC/HYP Handle)

Coming-up In Next Slides

3. **Host KVM** (No Workarounds: Arch Constraints)

- **Possible vCPUs** : Fully Initialize
- **Unplugged vCPUs:** Powered Down
- **VGIC** : Fully Initialize
- **Exit HVC/SMC:** CPU_{ON,OFF} to VMM (Policy Check)

Covered In Previous Next Slides
+
Accompanying James Morse's Slides

Qemu Challenges: to Realize VCPU Hotplug (1)

1. Customer requirement: **x86_64 look-and-feel of VCPU Hotplug** on ARM (Different Architectures)
   - ❑ How to **fake VCPU Hot-plug at Qemu** when KVM & Guest Kernel don't support it? **(Problem-1)** >>
   - ❑ QMP Command 'device_{add, del}' – How to use same interface?

     ```
     $ device_add   host-arm-cpu, id=core4, core-id=4
     $ device_del   core-id=4
     ```
   - ❑ VM boot time should be comparable to x86_64 (hot-plug should not degrade)

2. At VM init, Qemu CPU Objects MUST be created for each Host KVM VCPUs. (KVM Constraint)
   - ❑ What should we do to CPU Objects for '**yet-to-be-plugged'** VCPUs (after Init)? **(Problem-2)** >>
     - ❖ Keep 'not-plugged' VCPUs realized? (CPU Threads will always exist – **Boot time**?)
     - ❖ Unrealized 'not-plugged' VCPU CPU objects can't be migrated (VMSD registration?)
     - ❖ Delete or keep them disabled after KVM initialization has finished?

Qemu Challenges: to Realize VCPU Hotplug (2)

3. Qemu MUST **fully 'realize' GIC V3** to correctly initialize Host KVM VGIC (KVM Constraint ➡ CPU Arch)
   - ❑ GIC V3 State MUST be pre-sized with all the 'possible' VCPUs (no escape?)
   - ❑ All redistributors state must pre-exists (even for not-plugged VCPUs)
   - ❑ GIC **init**, **access operations** or **updates** can only be performed on 'Enabled' VCPUs **(Problem-3)** >>

4. Guest MUST see all VCPUs as 'present' in MADT GICC CPU interface at boot (ARM CPU Arch Req)
   - ❑ Present ACPI _STA.PRESENT=1 _STA.ENABLED={0,1} to Guest, Always! **(Problem-4)** >>
   - ❑ 'not-plugged' Qemu VCPUs: show as 'disabled' i.e. ACPI _STA.ENABLED=0 (Conflicts with QOM State)

Qemu Challenges: to Realize VCPU Hotplug (3)

5. What if in future ARM CPU Arch supports Physical CPU Hotplug? (ARM CPU Arch says 'assume nothing')
   - Can Qemu implementation be generic enough in the following cases **(Problem-5)** >>
     - Virtual CPU Hotplug Only
     - Physical CPU Hotplug Only
     - Both Virtual & Physical Hotplug co-exists
   - How should Qemu distinguish these cases and tell Guest about compatibility?
     - Should Guest evaluate(_OSC)?

Qemu Workarounds: Creative Hack – Which Works! Really? (1)

- Oh Yes! That thin Red line between **Bravery/Madness**, **Bug/Feature**, **Creativity/Hack**
  - Allow me to call it a **Creative Hack**? But is it Acceptable to the community? You decide!

Approach-1: To mitigate the 5 Qemu problems discussed earlier

1. Use existing **device_{add, del}** hot-plug device interface **(Mitigation-1)** <<
   - a new CPU object in Qemu shall be created for a hot-plugged VCPU
   - On VCPU hot-unplug, CPU object shall be destroyed
   - Update the GICV3 about this event (re-patching GICC <-> VCPU) (Deviation from ARM Arch. Okay?)

[1] Rough Qemu Boot Time Figures Presented At Linaro Open Discussion Meeting

Qemu Workarounds: Creative Hack – Which Works! Really? (2)

2. Creation, initialization and realization for {cold, hot-}plug VCPUs **(Mitigation-2)** <<

- ❑ During VM Init, create and initialize all 'possible' CPU objects

- ❑ Only 'realize' VCPUs which are cold plugged (**-smp cpus 4** maxcpus 6) (CPU Threads)
  - ❖ Including cold-plugged VCPUs added using '-device' option
  - ❖ Will end up creating, initializing and running KVM VCPUs (KVM_{CREATE_VCPU, ARM_CPU_INIT, RUN})

- ❑ **Do not** 'realize' un-plugged (will be 'hot plugged') VCPUs CPU Objects (i.e. No CPU Threads)
  - ❖ Only create & initialize KVM VCPU at Host KVM (KVM_CREATE_VCPU, KVM_ARM_VCPU_INIT)
  - ❖ Above strategy also drastically saves Boot time [1] (a requirement for Kata Containers)

[1] Rough Qemu Boot Time Figures Presented At Linaro Open Discussion Meeting

Qemu Workarounds: Creative Hack – Which Works! Really? (3)

3. Qemu GICV3 **full Realization (Mitigation-3)** <<

- ❑ **No Workarounds** for below (ARM CPU Architecture Constraint)
  - ❖ Pre-size GICv3State with 'possible' vCPUs but use only 'enabled' (no escape here)
  - ❖ Create all the GICV3 Redistributors for all 'possible' vCPUs both in Qemu & KVM
    - ▪ All related Memory Regions will be created in Qemu and initialized in KVM

- ❑ Some **subtle changes** are required for realization, initialization, access and updates for GICV3
  - ❖ GICV3 realization: only assign 'enabled' vCPUS [GICv3State->GICv3CPUState->CPUState]
  - ❖ Initialize the GICV3 CPU interface registers to KVM values (on vCPU reset)
    - ▪ GICv3 CPU interface Reginfo (ICC_CTLR_EL1) for newly plugged vCPUs need to be init again.
    - ▪ .resetfn(): fetch value of ICC_CTLR_EL1 from KVM.

Qemu Workarounds: Creative Hack – Which Works! Really? (4)

- Some **subtle changes** are required for realization, initialization, access and updates for GICV3 **(Contd.)**
  - Restrict VCPU updates for re-patching GICv3CPUState to 'enabled' VCPUs (plug/hotplug event)
  - Restrict KVM device access (get/put) of GICC registers to 'enabled' VCPUs (kvm_gic{c,d,r}_access)
    - Why? State sync up for example for pre_save, post_load, reset_hold (putting back to kvm) etc.

4. What **update** does on VCPU {Un-}Plug events?

- Match GIC CPU Interface: **'mp-affinity'** ⇔ **GICR_TYPER** (All the GIC CPU Interface)
  - Match Found? VCPU Hot Unplug Event : VCPU Hotplug Event
- Adjust: GIC CPU Interface VCPU = Hot plugged VCPU (**Deviation** from ARM CPU Arch. Okay?)
- Initialize GIC CPU Interface Registers info again in Qemu (VCPU Hotplug)

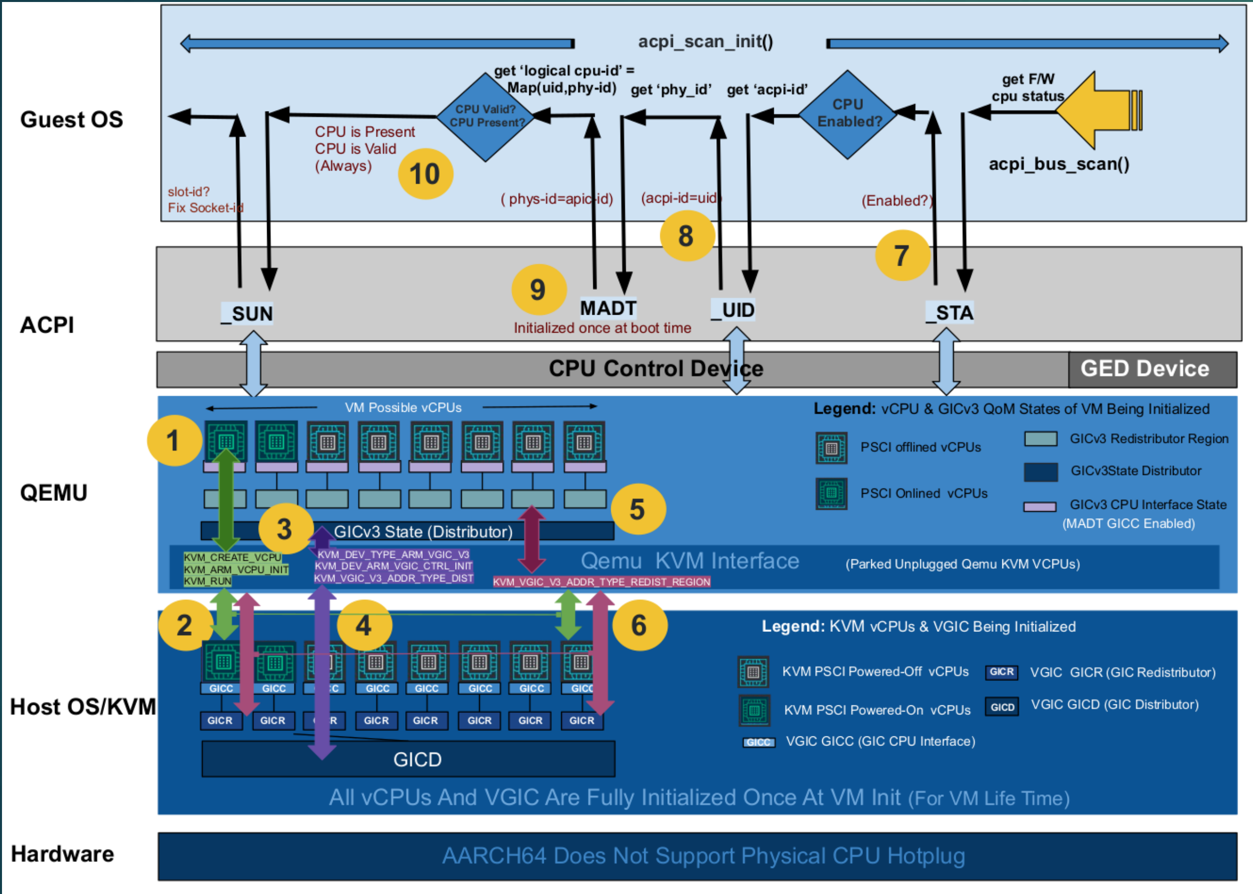
Qemu Workarounds: Creative Hack – Which Works! Really? (5)

5. Guest can defer online'ing/enabling the VCPU after boot. [ARM Arch Complaint] **(Mitigation-4)** <<
   - Use ACPI MADT GICC CPU Interface flag **'online-capable'** Bit [1] in Qemu.
   - VCPUs in Guest can be brought '{on, off}line' when Qemu hot-{un}plugs them
     - Will involve ACPI hot-plug handshake between Qemu and Guest Kernel
     - Handshake events: **Device-Check/Eject-Request/Eject Ej0/_STA**
6. Qemu MUST distinguish between ACPI _STA.PRESENT and _STA.ENABLED. Update Guest Kernel as,
   - _STA.PRESENT=1 and _STA.ENABLED=1 for 'plugged' VCPUs in Qemu
   - _STA.PRESENT=1 and _STA.ENABLED=0 for 'not-plugged' VCPUs in Qemu **(Faking Present – Okay?)**
7. Implement _OSC AML method to return to the guest about **(Mitigation-5)** <<
   - Hotplug 'Enabled' or 'Present' Support

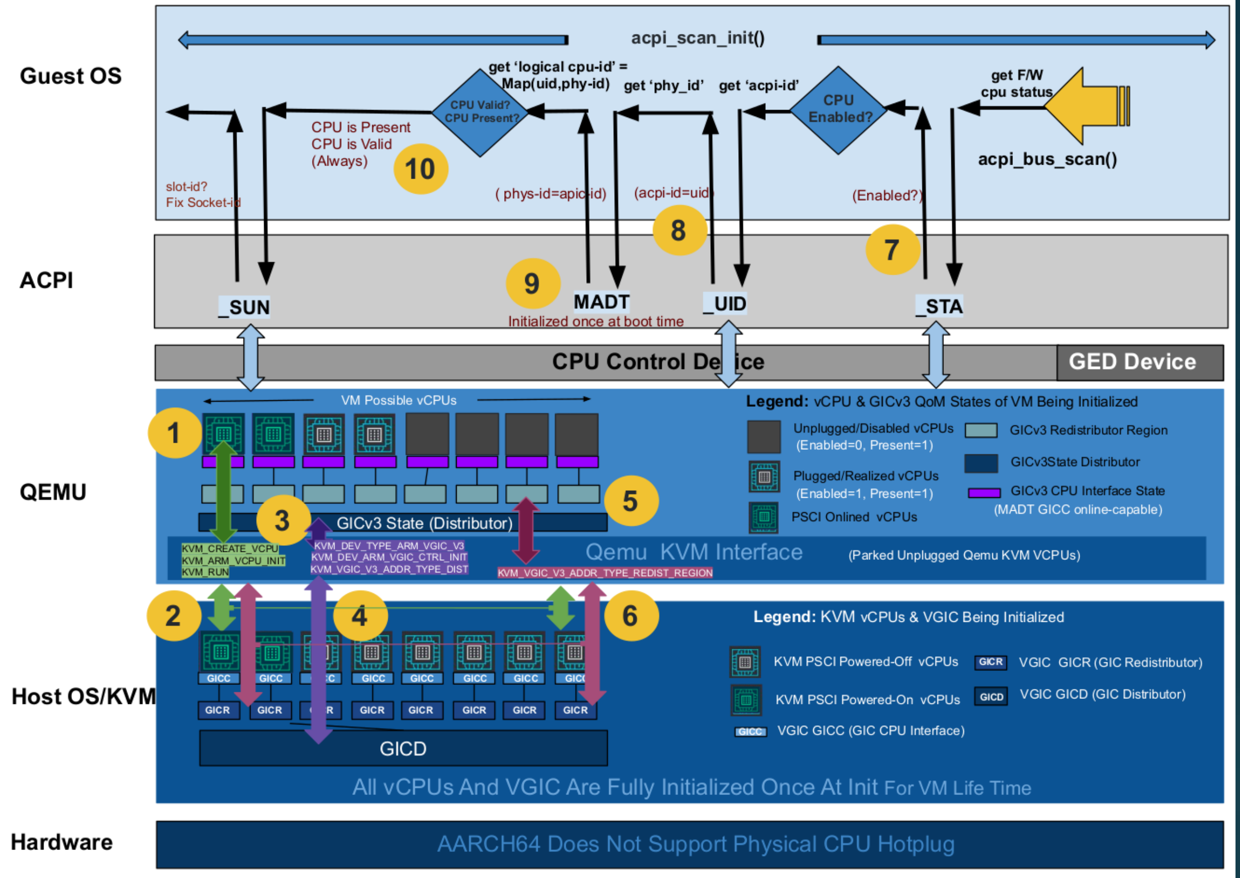
[1] ACPI specification 6.5 "Table 5.37: GICC CPU Interface Flags"

Hidden Backup Slide For Discussion Only: (Not To Be Presented)

VM Init in A Nutshell: Qemu, KVM, Kernel (AARCH64)


Guest OS
acpi_scan_init()
get 'logical cpu-id' = Map(uid,phy-id)
get 'phy_id'
get 'acpi-id'
CPU Valid? CPU Present?
CPU Enabled?
get F/W cpu status
acpi_bus_scan()
CPU is Present
CPU is Valid
(Always)
10
slot-id?
Fix Socket-id
(phys-id=apic-id)
(acpi-id=uid)
(Enabled?)
ACPI
8
7
9
_SUN
MADT
Initialized once at boot time
_UID
_STA
CPU Control Device
GED Device
QEMU
VM Possible vCPUs
Legend: vCPU & GICv3 QoM States of VM Being Initialized
PSCI offlined vCPUs
PSCI Onlined vCPUs
GICv3 Redistributor Region
GICv3State Distributor
GICv3 CPU Interface State (MADT GICC Enabled)
1
3
5
GICv3 State (Distributor)
KVM_CREATE_VCPU
KVM_ARM_VCPU_INIT
KVM_RUN
KVM_DEV_TYPE_ARM_VGIC_V3
KVM_DEV_ARM_VGIC_CTRL_INIT
KVM_VGIC_V3_ADDR_TYPE_DIST
KVM_VGIC_V3_ADDR_TYPE_REDIST_REGION
Qemu KVM Interface
(Parked Unplugged Qemu KVM VCPUs)
2
4
6
Host OS/KVM
Legend: KVM vCPUs & VGIC Being Initialized
KVM PSCI Powered-Off vCPUs
KVM PSCI Powered-On vCPUs
VGIC GICC (GIC CPU Interface)
VGIC GICR (GIC Redistributor)
VGIC GICD (GIC Distributor)
GICC
GICR
GICD
All vCPUs And VGIC Are Fully Initialized Once At VM Init (For VM Life Time)
Hardware
AARCH64 Does Not Support Physical CPU Hotplug

Sequence Depicting VM & Guest Kernel Initialization (AARCH64) (No Hotplug Support)

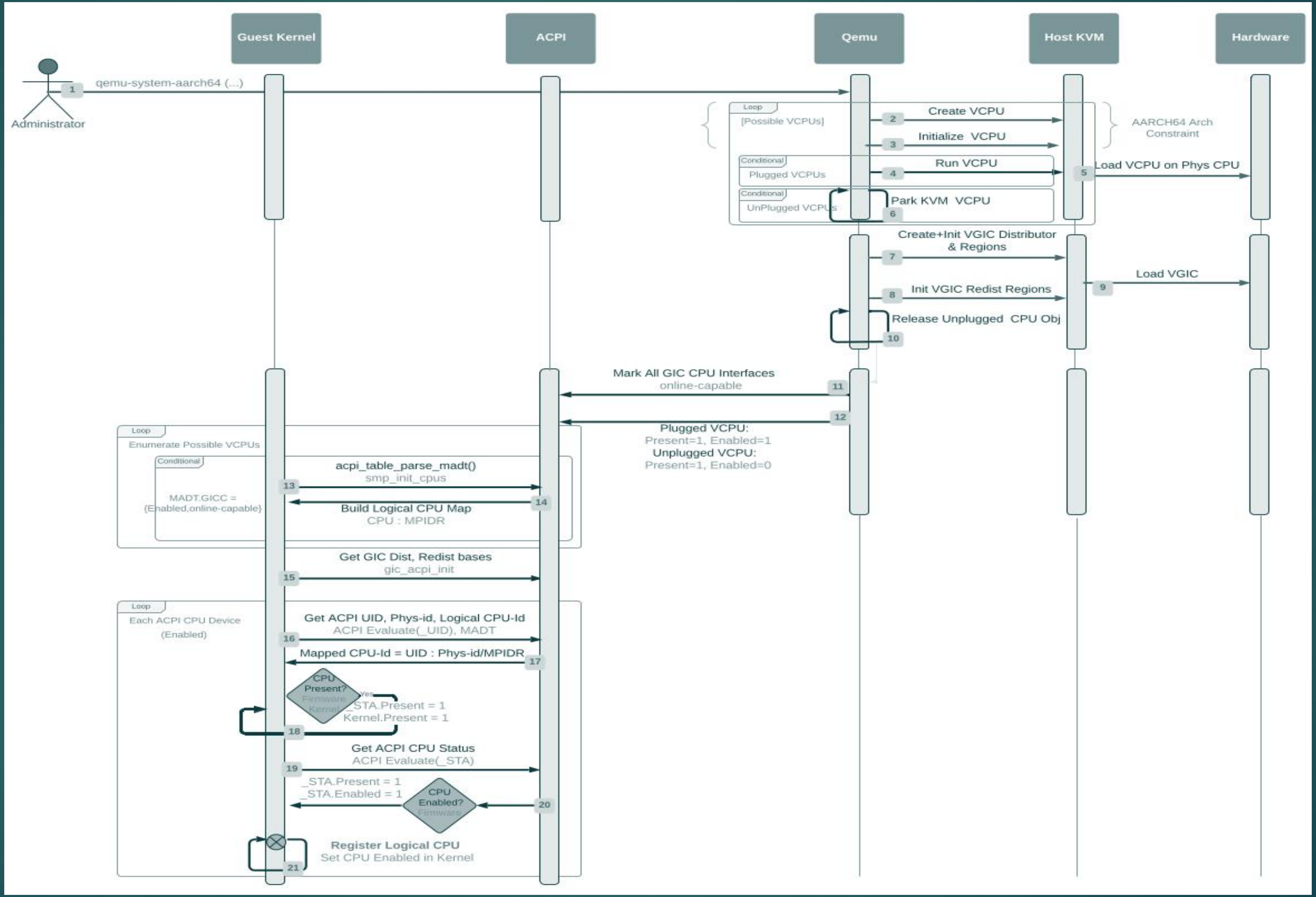

Guest OS
acpi_scan_init()
get 'logical cpu-id' = Map(uid,phy-id)
get 'phy_id'
get 'acpi-id'
CPU Valid? CPU Present?
CPU Enabled?
get F/W cpu status
acpi_bus_scan()
CPU is Present
CPU is Valid
(Always)
10
slot-id?
Fix Socket-id
(phys-id=apic-id)
(acpi-id=uid)
(Enabled?)
ACPI
8
7
9
_SUN
MADT
Initialized once at boot time
_UID
_STA
CPU Control Device
GED Device
QEMU
VM Possible vCPUs
Legend: vCPU & GICv3 QoM States of VM Being Initialized
Unplugged/Disabled vCPUs (Enabled=0, Present=1)
Plugged/Realized vCPUs (Enabled=1, Present=1)
PSCI Onlined vCPUs
GICv3 Redistributor Region
GICv3State Distributor
GICv3 CPU Interface State (MADT GICC online-capable)
1
3
5
GICv3 State (Distributor)
KVM_CREATE_VCPU
KVM_ARM_VCPU_INIT
KVM_RUN
KVM_DEV_TYPE_ARM_VGIC_V3
KVM_DEV_ARM_VGIC_CTRL_INIT
KVM_VGIC_V3_ADDR_TYPE_DIST
KVM_VGIC_V3_ADDR_TYPE_REDIST_REGION
Qemu KVM Interface
(Parked Unplugged Qemu KVM VCPUs)
2
4
6
Host OS/KVM
Legend: KVM vCPUs & VGIC Being Initialized
KVM PSCI Powered-Off vCPUs
KVM PSCI Powered-On vCPUs
VGIC GICC (GIC CPU Interface)
VGIC GICR (GIC Redistributor)
VGIC GICD (GIC Distributor)
GICC
GICR
GICD
All vCPUs And VGIC Are Fully Initialized Once At Init For VM Life Time
Hardware
AARCH64 Does Not Support Physical CPU Hotplug

Sequence Depicting VM & Guest Kernel Initialization (AARCH64) (With Hotplug Support)

[1] KVMForum 2020 VCPU Hotplug Slide: VCPU Hotplug Events

Hidden Backup Slide For Discussion Only: (Not To Be Presented)

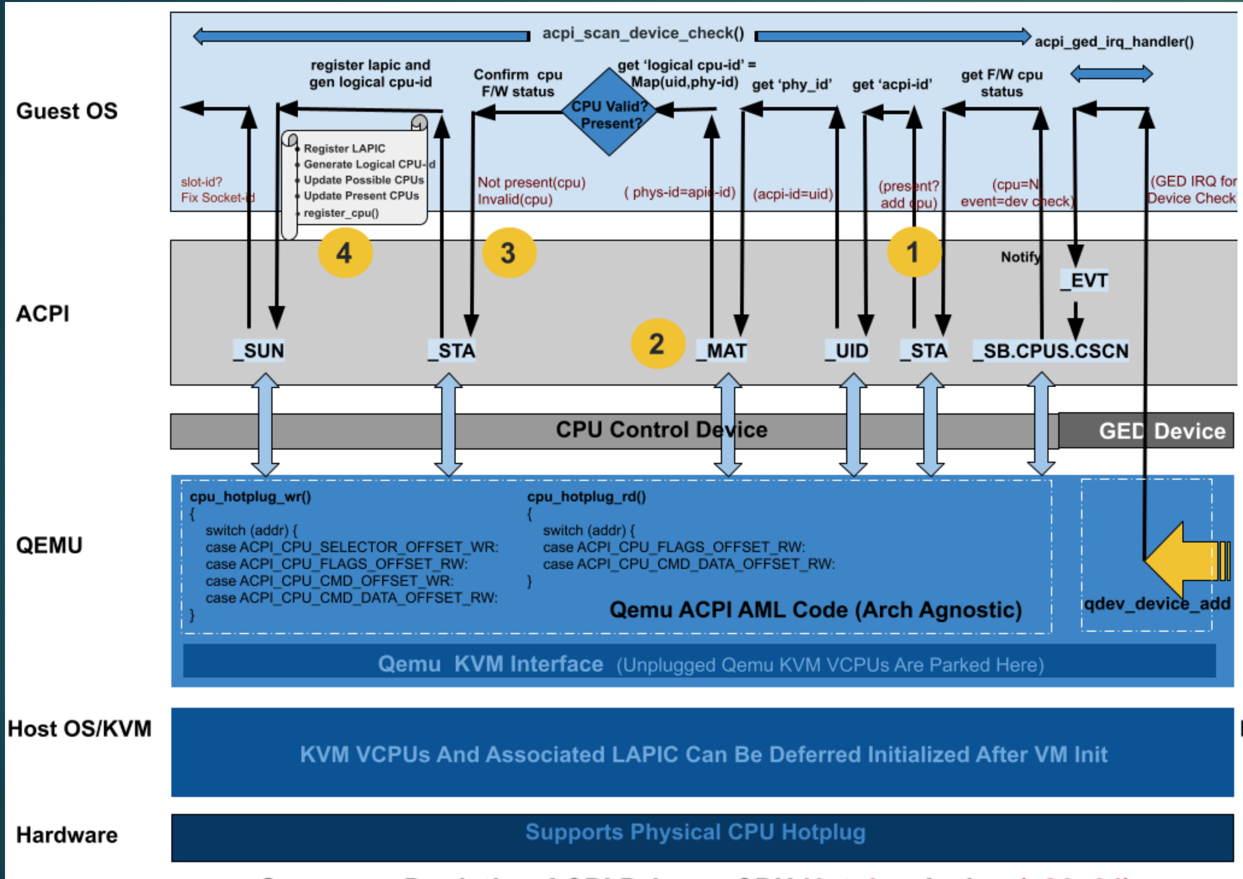
VM Init Sequence: Qemu, KVM, ACPI, Kernel **(AARCH64)**


Guest Kernel
ACPI
Qemu
Host KVM
Hardware
Administrator
1 qemu-system-aarch64 (...)
Loop
[Possible VCPUs]
2 Create VCPU
3 Initialize VCPU
AARCH64 Arch Constraint
Conditional
Plugged VCPUs
4 Run VCPU
5 Load VCPU on Phys CPU
Conditional
UnPlugged VCPUs
6 Park KVM VCPU
7 Create+Init VGIC Distributor & Regions
9 Load VGIC
8 Init VGIC Redist Regions
10 Release Unplugged CPU Obj
11 Mark All GIC CPU Interfaces online-capable
12 Plugged VCPU: Present=1, Enabled=1 Unplugged VCPU: Present=1, Enabled=0
Loop
Enumerate Possible VCPUs
Conditional
MADT.GICC = {Enabled,online-capable}
13 acpi_table_parse_madt() smp_init_cpus
14 Build Logical CPU Map CPU : MPIDR
15 Get GIC Dist, Redist bases gic_acpi_init
Loop
Each ACPI CPU Device (Enabled)
16 Get ACPI UID, Phys-id, Logical CPU-Id ACPI Evaluate(_UID), MADT
17 Mapped CPU-Id = UID : Phys-id/MPIDR
CPU Present? Firmware Kernel
Yes
_STA.Present = 1
Kernel.Present = 1
18
19 Get ACPI CPU Status ACPI Evaluate(_STA)
_STA.Present = 1
_STA.Enabled = 1
CPU Enabled? Firmware
20
21 Register Logical CPU Set CPU Enabled in Kernel

HUAWEI

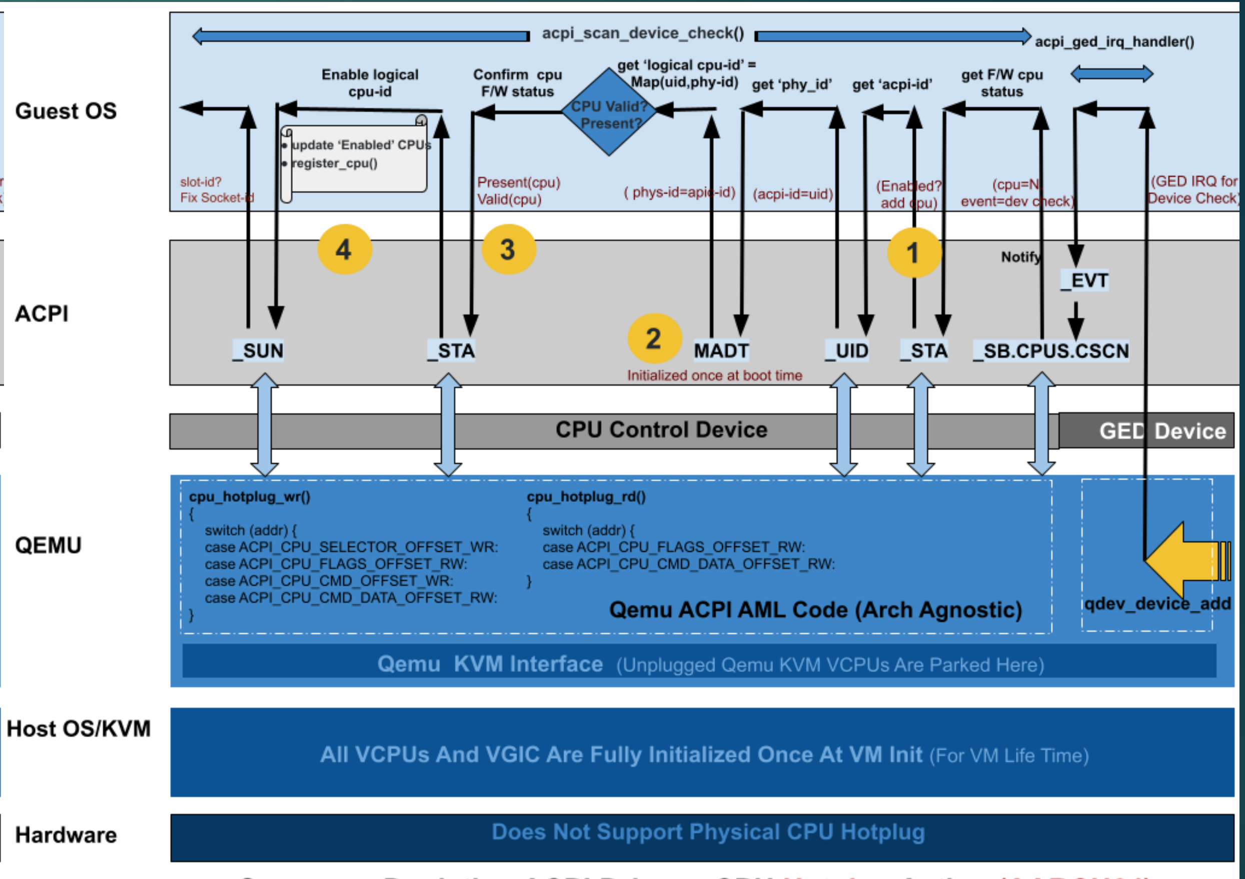
Hidden Backup Slide For Discussion Only: (Not To Be Presented)

Comparison: Virtual CPU Hotplug ACPI Event Seq **(x86_64 vs AARCH64)**



```
cpu_hotplug_wr()
{
   switch (addr) {
   case ACPI_CPU_SELECTOR_OFFSET_WR:
   case ACPI_CPU_FLAGS_OFFSET_RW:
   case ACPI_CPU_CMD_OFFSET_WR:
   case ACPI_CPU_CMD_DATA_OFFSET_RW:
}

cpu_hotplug_rd()
{
   switch (addr) {
   case ACPI_CPU_FLAGS_OFFSET_RW:
   case ACPI_CPU_CMD_DATA_OFFSET_RW:
}
```

Sequence Depicting ACPI Driven vCPU Hotplug Action (x86_64)



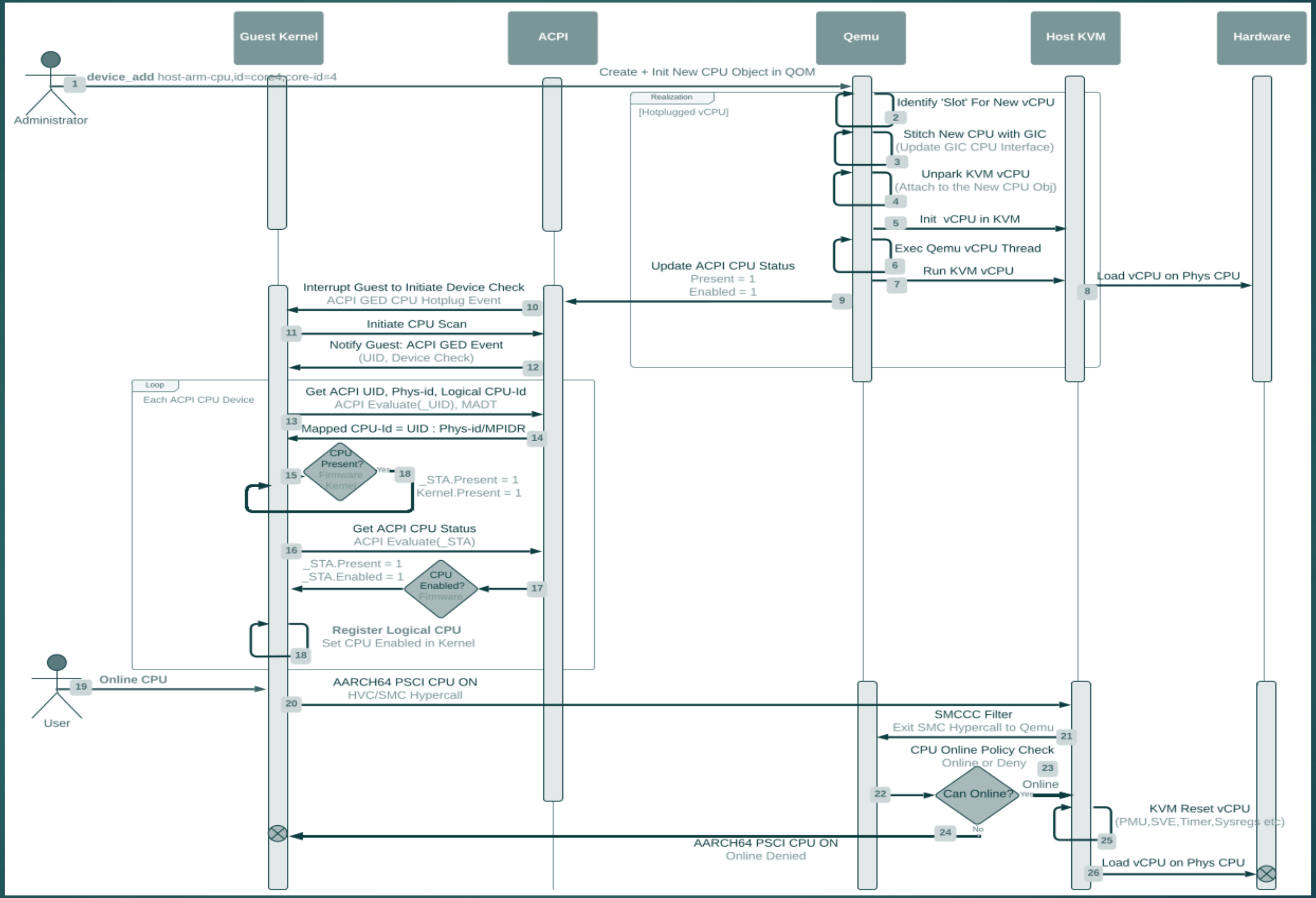
```
cpu_hotplug_wr()
{
   switch (addr) {
   case ACPI_CPU_SELECTOR_OFFSET_WR:
   case ACPI_CPU_FLAGS_OFFSET_RW:
   case ACPI_CPU_CMD_OFFSET_WR:
   case ACPI_CPU_CMD_DATA_OFFSET_RW:
}

cpu_hotplug_rd()
{
   switch (addr) {
   case ACPI_CPU_FLAGS_OFFSET_RW:
   case ACPI_CPU_CMD_DATA_OFFSET_RW:
}
```

Sequence Depicting ACPI Driven vCPU Hotplug Action (AARCH64)

[1] KVMForum 2020 VCPU Hotplug Slide: VCPU Hotplug Events

Hidden Backup Slide For Discussion Only: (Not To Be Presented)

Hotplug Sequence: Qemu, KVM, ACPI, Kernel (AARCH64)

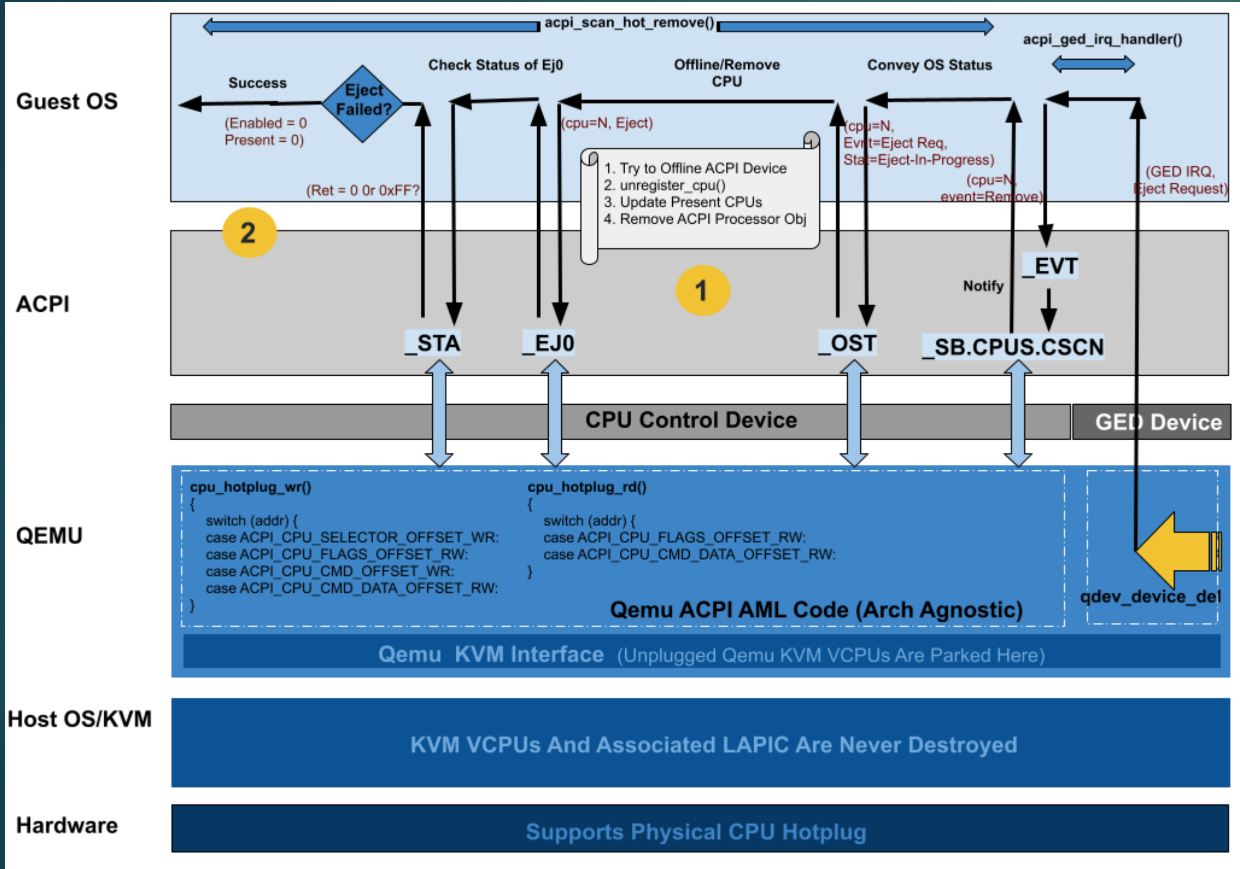


Guest Kernel
ACPI
Qemu
Host KVM
Hardware
Administrator
1 device_add host-arm-cpu,id=core4,core-id=4
Create + Init New CPU Object in QOM
Realization
[Hotplugged vCPU]
2 Identify 'Slot' For New vCPU
3 Stitch New CPU with GIC (Update GIC CPU Interface)
4 Unpark KVM vCPU (Attach to the New CPU Obj)
5 Init vCPU in KVM
6 Exec Qemu vCPU Thread
7 Run KVM vCPU
8 Load vCPU on Phys CPU
9 Update ACPI CPU Status Present = 1 Enabled = 1
10 Interrupt Guest to Initiate Device Check ACPI GED CPU Hotplug Event
11 Initiate CPU Scan
12 Notify Guest: ACPI GED Event (UID, Device Check)
Loop
Each ACPI CPU Device
13 Get ACPI UID, Phys-id, Logical CPU-Id ACPI Evaluate(_UID), MADT
14 Mapped CPU-Id = UID : Phys-id/MPIDR
15 CPU Present? Firmware Kernel
Yes
18 _STA.Present = 1 Kernel.Present = 1
16 Get ACPI CPU Status ACPI Evaluate(_STA)
_STA.Present = 1 _STA.Enabled = 1
17 CPU Enabled? Firmware
18 Register Logical CPU Set CPU Enabled in Kernel
User
19 Online CPU
20 AARCH64 PSCI CPU ON HVC/SMC Hypercall
21 SMCCC Filter Exit SMC Hypercall to Qemu
23 CPU Online Policy Check Online or Deny
22 Can Online?
Yes
Online
No
25 KVM Reset vCPU (PMU,SVE,Timer,Sysregs etc)
24 AARCH64 PSCI CPU ON Online Denied
26 Load vCPU on Phys CPU

[1] KVMForum 2020 VCPU Hotplug Slide: VCPU Hotplug Events

Hidden Backup Slide For Discussion Only: (Not To Be Presented)

Comparison: Virtual CPU Hot-Unplug ACPI Event Seq (x86_64 vs AARCH64)

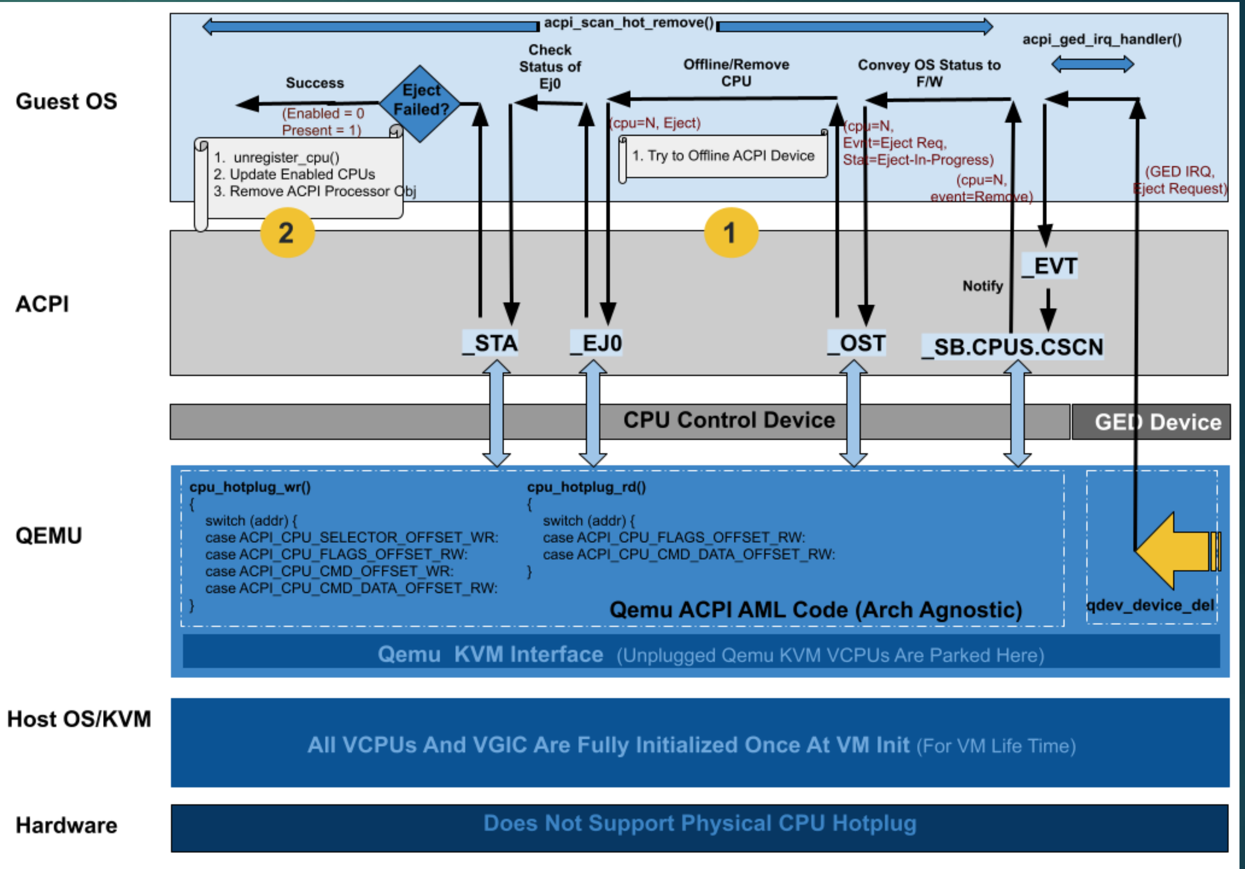


1. Try to Offline ACPI Device
2. unregister_cpu()
3. Update Present CPUs
4. Remove ACPI Processor Obj

```
cpu_hotplug_wr()
{
   switch (addr) {
   case ACPI_CPU_SELECTOR_OFFSET_WR:
   case ACPI_CPU_FLAGS_OFFSET_RW:
   case ACPI_CPU_CMD_OFFSET_WR:
   case ACPI_CPU_CMD_DATA_OFFSET_RW:
}
```

```
cpu_hotplug_rd()
{
   switch (addr) {
   case ACPI_CPU_FLAGS_OFFSET_RW:
   case ACPI_CPU_CMD_DATA_OFFSET_RW:
   }
```

Sequence Depicting ACPI Driven vCPU Hot-Unplug Action (x86_64)



1. unregister_cpu()
2. Update Enabled CPUs
3. Remove ACPI Processor Obj

1. Try to Offline ACPI Device

```
cpu_hotplug_wr()
{
   switch (addr) {
   case ACPI_CPU_SELECTOR_OFFSET_WR:
   case ACPI_CPU_FLAGS_OFFSET_RW:
   case ACPI_CPU_CMD_OFFSET_WR:
   case ACPI_CPU_CMD_DATA_OFFSET_RW:
}
```

```
cpu_hotplug_rd()
{
   switch (addr) {
   case ACPI_CPU_FLAGS_OFFSET_RW:
   case ACPI_CPU_CMD_DATA_OFFSET_RW:
   }
```

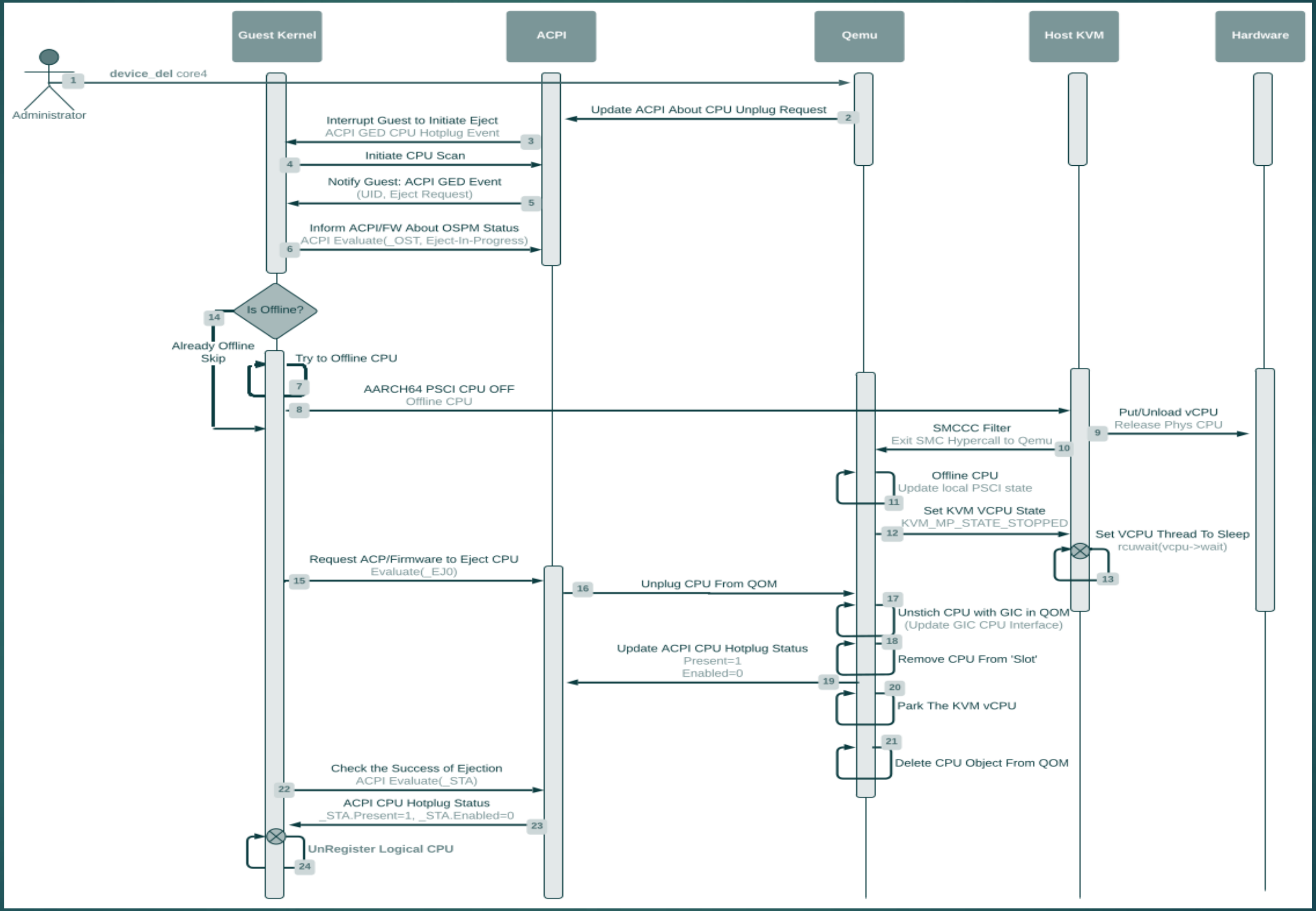
Sequence Depicting ACPI Driven vCPU Hot-Unplug Action (AARCH64)

[1] KVMForum 2020 VCPU Hotplug Slide: VCPU Hotplug Events

Hidden Backup Slide For Discussion Only: (Not To Be Presented)

Hotunplug Sequence: Qemu, KVM, ACPI, Kernel **(AARCH64)**

19F



Guest Kernel
ACPI
Qemu
Host KVM
Hardware
Administrator
1 device_del core4
2 Update ACPI About CPU Unplug Request
3 Interrupt Guest to Initiate Eject
ACPI GED CPU Hotplug Event
4 Initiate CPU Scan
5 Notify Guest: ACPI GED Event
(UID, Eject Request)
6 Inform ACPI/FW About OSPM Status
ACPI Evaluate(_OST, Eject-In-Progress)
Is Offline?
14 Already Offline
Skip
Try to Offline CPU
7
8 AARCH64 PSCI CPU OFF
Offline CPU
9 Put/Unload vCPU
Release Phys CPU
10 SMCCC Filter
Exit SMC Hypercall to Qemu
11 Offline CPU
Update local PSCI state
12 Set KVM VCPU State
KVM_MP_STATE_STOPPED
Set VCPU Thread To Sleep
rcuwait(vcpu->wait)
13
15 Request ACP/Firmware to Eject CPU
Evaluate(_EJ0)
16 Unplug CPU From QOM
17 Unstich CPU with GIC in QOM
(Update GIC CPU Interface)
18 Remove CPU From 'Slot'
19 Update ACPI CPU Hotplug Status
Present=1
Enabled=0
20 Park The KVM vCPU
21 Delete CPU Object From QOM
22 Check the Success of Ejection
ACPI Evaluate(_STA)
23 ACPI CPU Hotplug Status
_STA.Present=1, _STA.Enabled=0
24 UnRegister Logical CPU

[1] KVMForum 2020 VCPU Hotplug Slide: VCPU Hotplug Events

KVM Forum 2023 HUAWEI

Approx. VM Init Time: With Varying VCPUs

- Qemu VCPU Realization involves spawning of vCPU threads and KVM_RUN IOCTL calls which can be avoided for unplugged VCPUs. This improves VM Init time and mem footprint. Can we can defer spawning of Qemu threads even for plugged till they are online'd using SMCC Exit calls to Qemu? (scope of optimization? – an experiment-in-progress!)
- Below figures were taken to measure the impact of the VCPU realization i.e. spawning of Qemu threads and KVM_RUN IOCTL on the VM init time. Total possible vCPUS = 128. 'N' Realized VCPUs = 'N' vCPU Threads in Qemu.

SN	Stage of VM Init	Realized = 1, Unrealized = 127	Realized = 128, Unrealized = 0
1	**Pre vCPU Init**	~25 msec	~40 msec
2	**Post vCPU Init** (But Pre-VGIC Init)	~40 msec (+15) ⬅	~240 msec (+200) ⬅
3	**Post GIC Init**	~70 msec (+30) ⬅	~340 msec (+100) ⬅
4	**Pre Linux Kernel Load**	~90 msec (+20)	~360 msec (+20)
5	**Post Kernel Load** (But Pre SMBIOS setup)	~120 msec (+30)	~410 msec (+50)

[1] VM Init Time Discussion: Linaro Open Discussions Date: 22-03-2021

Prototype Code: Repositories & Upstreaming

1. [PATCH RFC 0/4] Changes to Support *Virtual* CPU Hotplug for ARM64 **(Salil Mehta/Xiongfeng Wang)**
   - https://lore.kernel.org/all/20200625133757.22332-1-salil.mehta@huawei.com/
2. [PATCH RFC 00/22] Support of Virtual CPU Hotplug for ARMv8 Arch **(Salil Mehta/Keqian Zhu)**
   - https://lore.kernel.org/qemu-devel/20200613213629.21984-1-salil.mehta@huawei.com/
3. Latest **Qemu** Prototype **(Pre RFC V2)**
   - https://github.com/salil-mehta/qemu.git virt-cpuhp-armv8/rfc-v1-port11052023.dev-1 **(Salil Mehta)**
4. Latest **Kernel** Prototype **(Pre RFC V2 = RFC V1 + Fixes)**
   - https://git.gitlab.arm.com/linux-arm/linux-jm.git virtual_cpu_hotplug/rfc/v2 **(James Morse)**

Acknowledgements

1. Marc Zyngier (Google)
2. James Morse (ARM)
3. Jean-Philippe Brucker (Linaro)
4. Lorenzo Pieralisi (Linaro)
5. Keqian Zhu (Huawei)
6. Xiongfeng Wang (Huawei)
7. Jonathan Cameron (Huawei)
8. Shameerali Kolothum Thodi (Huawei)
9. Will Deacon (Google)
10. Catalin Marinas (ARM)
11. Russel King (Oracle)
12. Miguel Luis (Oracle)
13. Vishnu Pajjuri (Ampere)
14. Sudeep Holla (ARM)
15. Andrew Jones (Redhat)
14. Igor Mamedov (Redhat)
15. Gavin Shan (Redhat)
16. Darren Hart (Ampere)
17. Ilkka Koskinen (Ampere)
18. Karl Heubaum (Oracle) and all those I have missed out!

Comments: We Can Discuss?

Q & A

Thanks

To the Wonderful Audience & All The Organizers of The KVMForum2023