GPU Acceleration for Container on Intel Processor Graphics

Zhenyu Wang
zhenyuw@linux.intel.com

Agenda

GPU Containers

GPU Namespace

GPU Control Group

Integration with Container Runtime

VM Containers

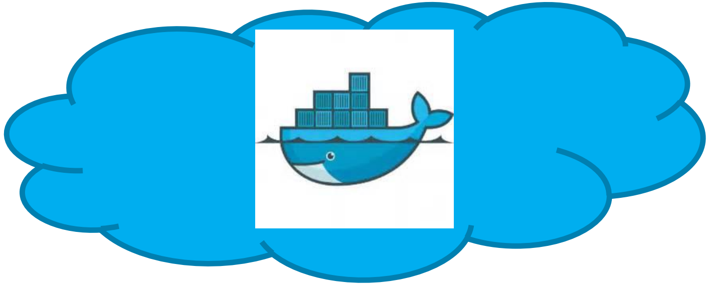
Status

GPU Containers

Cloud Graphics
(Remote desktop/gaming)

Video Delivery
(Encode/Decode/Transcode)

HPC/Machine Learning
(AI, Analytic, …)



Fast provisioning

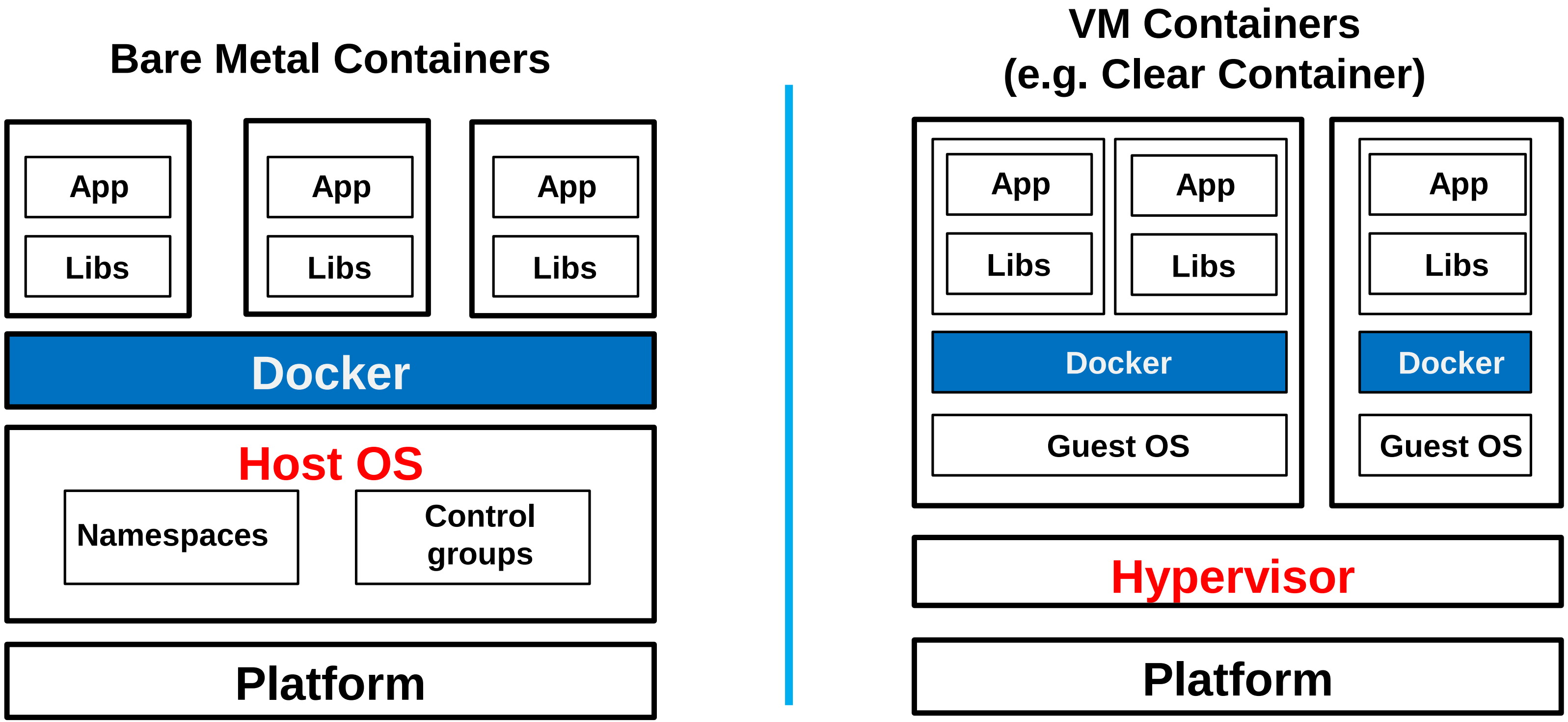
High Density

Little Overhead

E3-G

Discrete GPU

What Makes a Container?



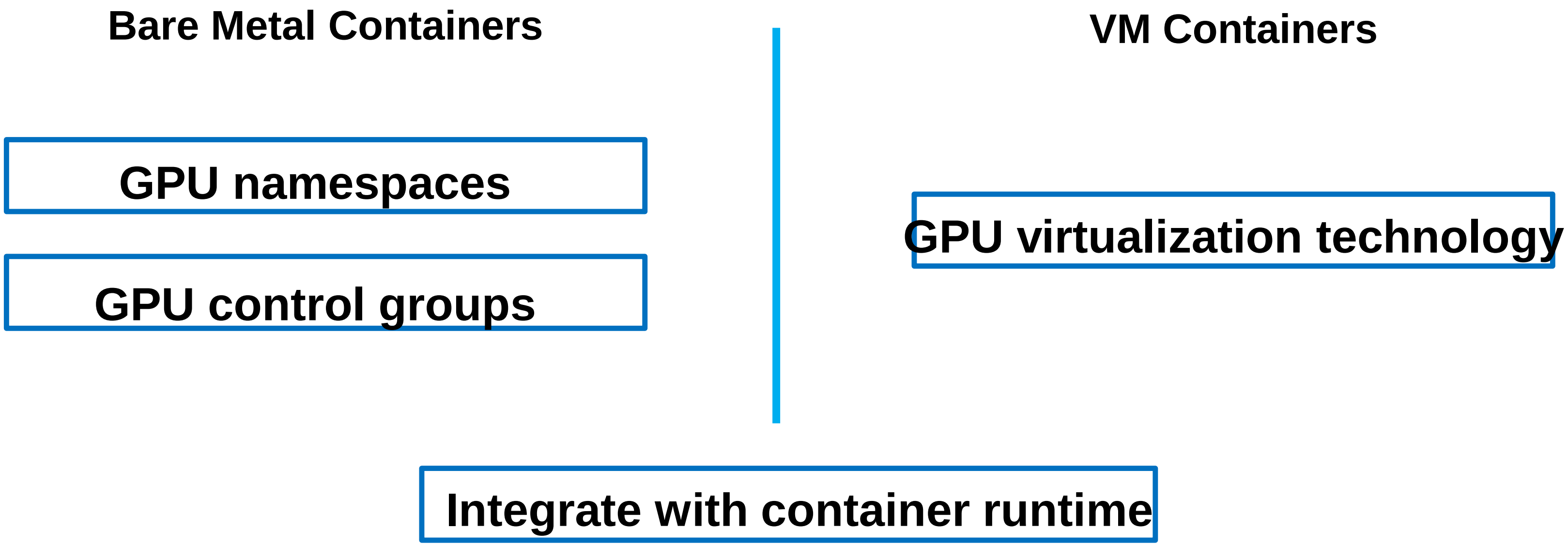
Bare Metal Containers

App
Libs
App
Libs
App
Libs
Docker
Host OS
Namespaces
Control groups
Platform

VM Containers (e.g. Clear Container)

App
Libs
App
Libs
Docker
Guest OS
App
Libs
Docker
Guest OS
Hypervisor
Platform

Path to GPU Containers



Bare Metal Containers

GPU namespaces

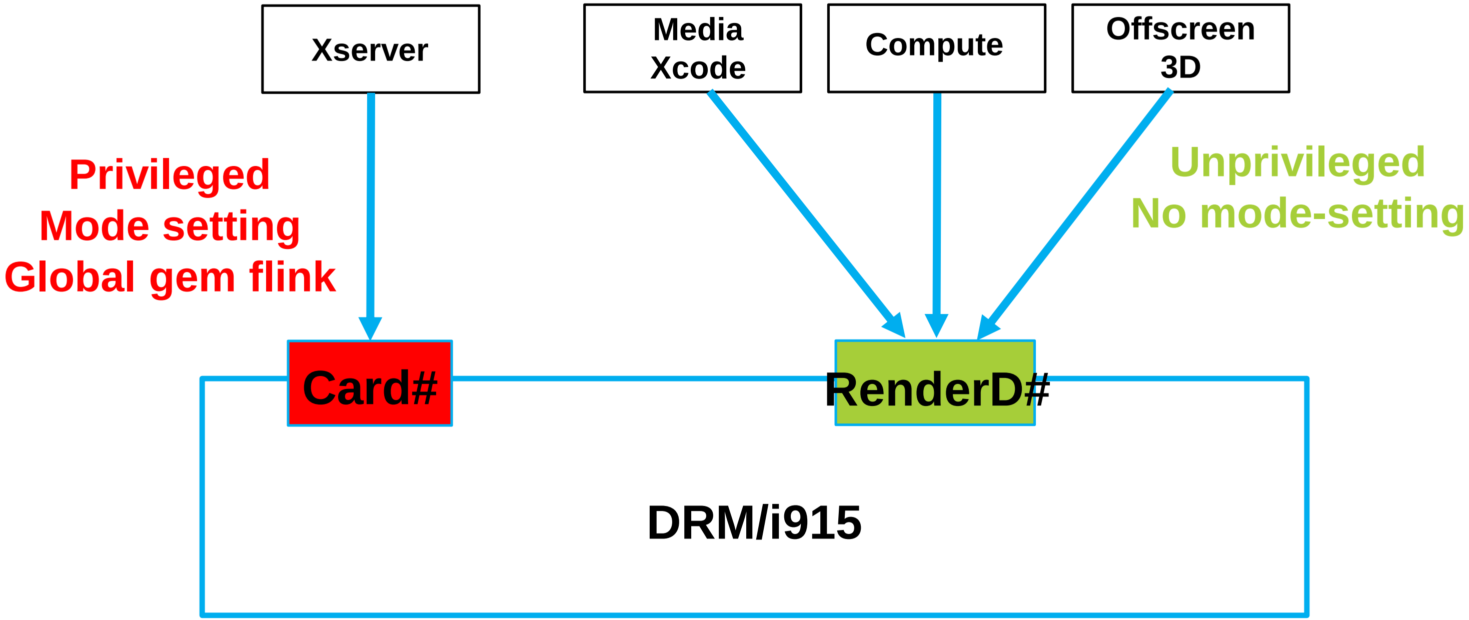
GPU control groups

VM Containers

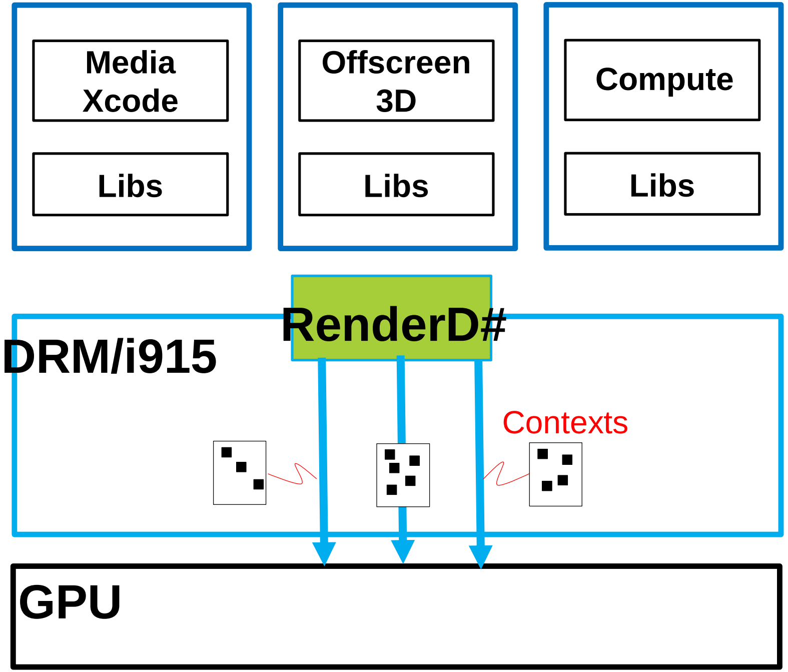
GPU virtualization technology

Integrate with container runtime

GPU Namespace:
DRM Render Node



GPU Namespace: DRM Render Node



Sharing happen with dmabuf only (no global gem object on card0)

No need of creating another namespace

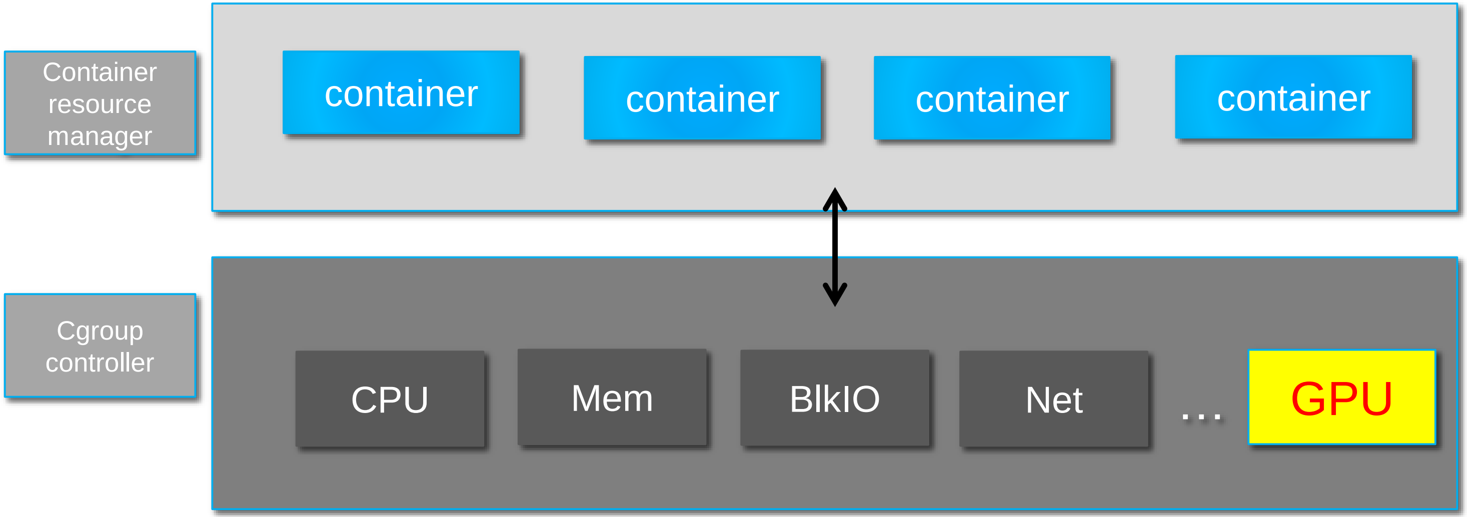
Expose DRM render nodes to provide isolated views through device cgroup

- ✓ e.g “*docker run –device=/dev/dri/renderD128 –it debian*”

Per-process hardware context in GPU

- ✓ Hardware managed render context switch

GPU CGroup



Container-granule GPU resource control

GPU CGroup

New subsystem: gpu_cgrp_subsys

Control knobs

- ✓ gpu.shares (% share of GPU cycles, work in progress)
- ✓ gpu.priority (workload priority in the system)
- ✓ gpu.memory (maximum GPU memory size)

DRM/i915

- ✓ Favor ‘shares/priority’ in workload scheduler
- ✓ Enforce memory limitation (optionally for UMA graphics)

Container Runtime Integration

RunC: an universal container runtime

- ✓ Understand new gpu cgroup
- ✓ Add new Linux resources config.json options for gpu
- ✓ Runtime-tools helper to generate config stubs

New Docker GPU control parameters

e.g *docker –gpu-mem=xxx –gpu-priority=xxx –gpu-share=xxx …*

```
{
 “linux”: {
   “devices”: [
     {
        “path”: “/dev/dri/renderD128”,
        “type”: “c”,
        “major”: 226,
        “minor”: 128,
        “fileMode”: 432,
        “uid”: 0,
        “gid”: 0
      }
    ],
    “resources”: {
      …
      “gpu”: {
        “memory”: <max_mem_in_bytes>,
        “prio”: <workload_priority>
      }
     }
}
```

Image Portability

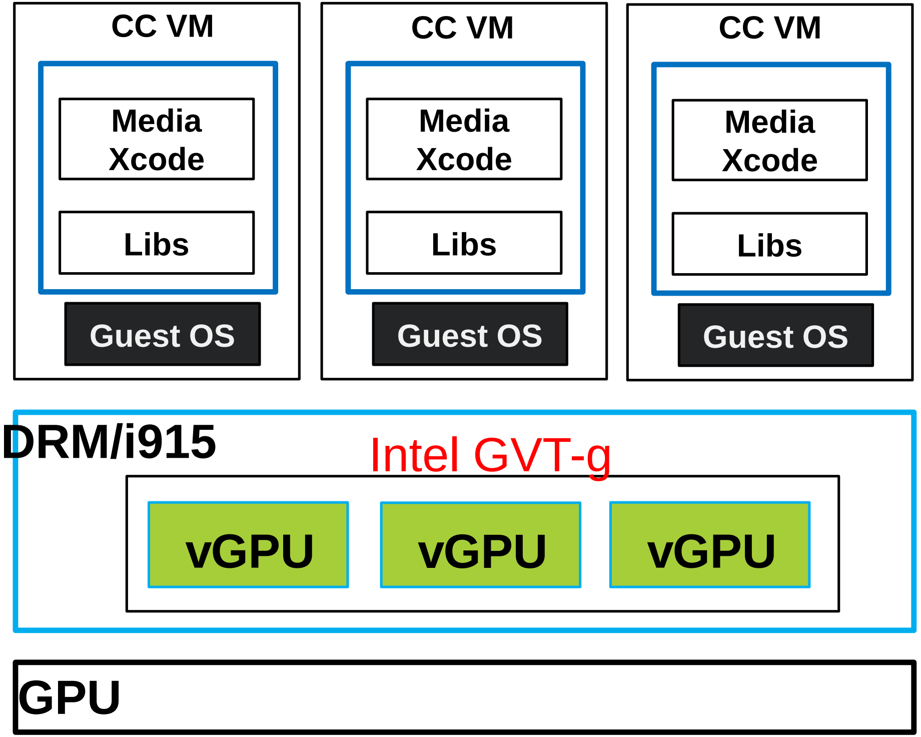
Main challenge - library compatibility

- ✓ User level libraries MUST match underlying kernel/HW

Introduce Docker helper for image inspection

- ✓ Compare image libraries with host environment
- ✓ Reject to launch container upon any mismatch

VM GPU Containers



Clear Container (CC)

Intel graphics virtualization technology (Intel GVT-g)

Assign vGPU to VM container

In upstream Linux since v4.10

https://01.org/igvt-g

VM GPU Containers

Nested containers in VM also have GPU access

Need Improvement:

Scalability

- ✓ Only support 8vGPUs today due to resource partitioning
- ✓ Need some enlightened way to further scale in the short term

Boot time

- ✓ Full guest graphics driver load takes >1.5s
- ✓ Fast optimization to <0.5s in progress

Status

✓ GPU cgroup PoC

https://github.com/zhenyw/linux

https://github.com/zhenyw/runc

https://github.com/zhenyw/moby

https://github.com/zhenyw/cli

✓ GPU virtualization for Clear container

https://clearlinux.org/features/intel%C2%AE-clear-containers

https://github.com/01org/gvt-linux/wiki/Clear-Container-with-GVTg-Setup-Guide

Q & A