

KubeCon | CloudNativeCon

North America 2026



#KubeCon #CloudNativeCon

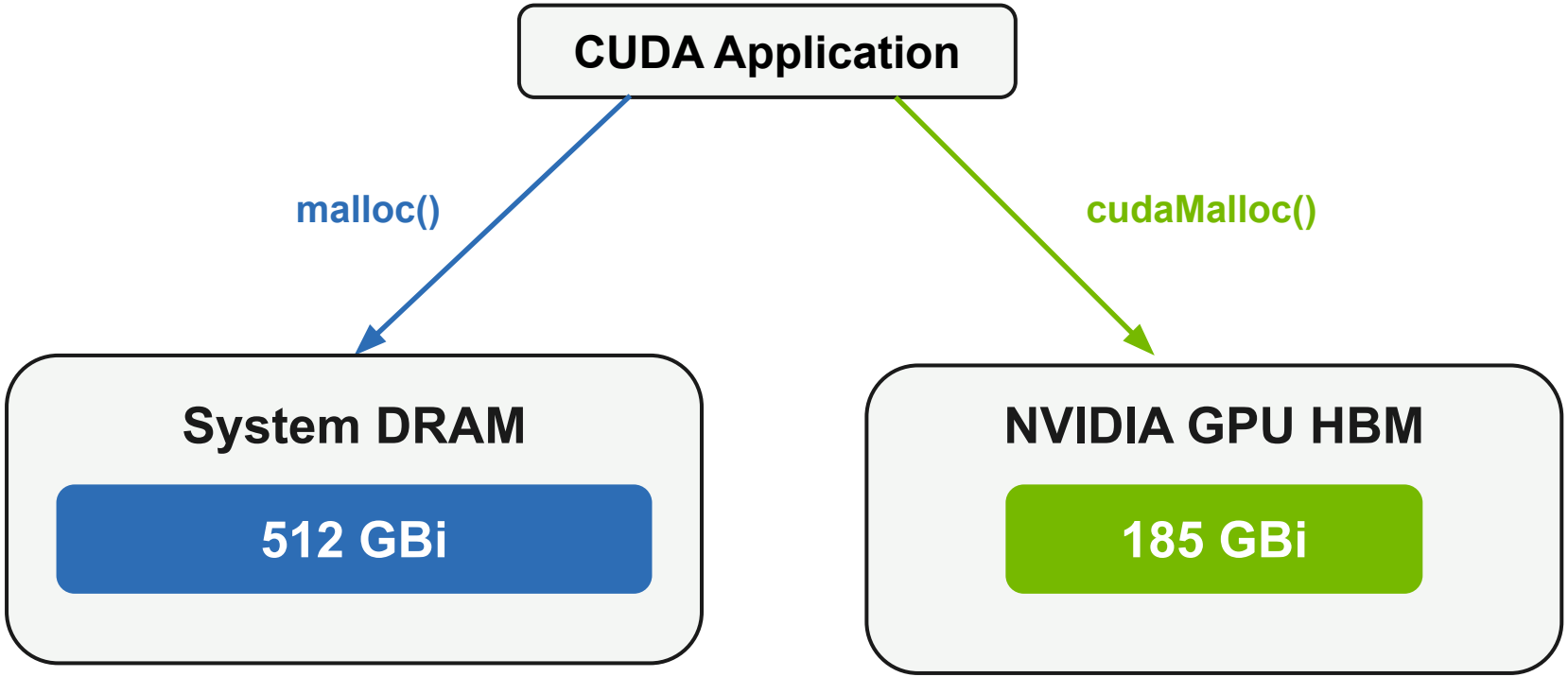
When GPUs Become NUMA Nodes: Challenges Running Kubernetes with Memory-Coherent Hardware

Kevin Klues & Balbir Singh

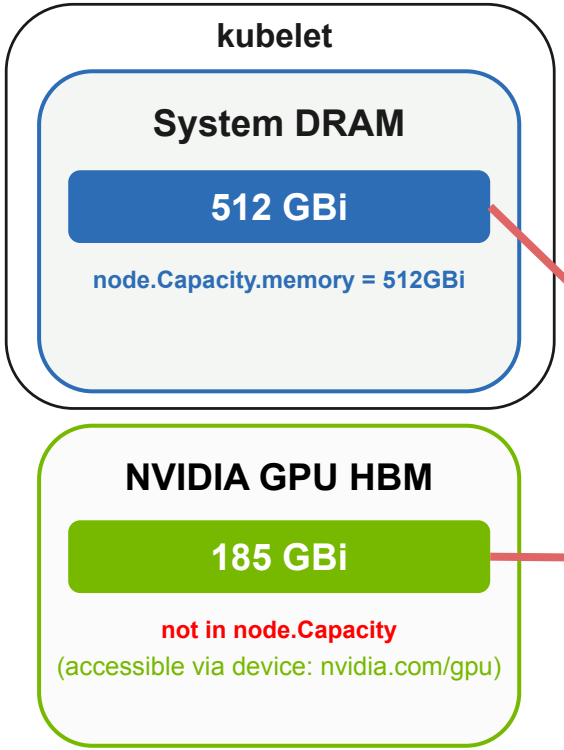
NVIDIA



CPU and GPU Memory in Kubernetes



CPU and GPU Memory in Kubernetes



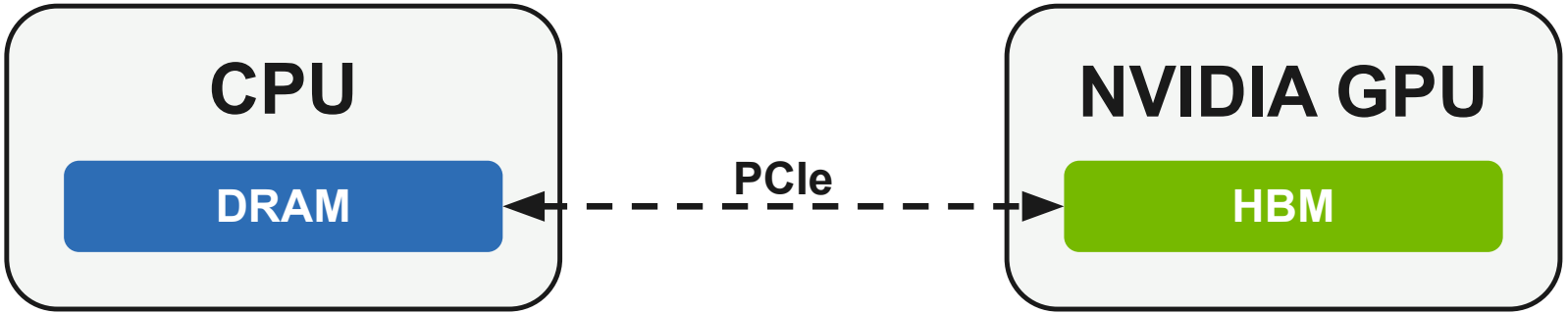

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: gpu-example
spec:
  containers:
    - name: gpu-example
      image: nvidia/cuda
      resources:
        limits:
          memory: 2GBi
          nvidia.com/gpu: 2
```

System memory is charged to the pod's **`cgroup`** and counted toward **`memory.limit`**

GPU memory is invisible to **`cgroups`** and only counted inside the NVIDIA driver

Non-Hardware-Coherent Memory Platforms

Traditional platforms connect the CPU and GPU over PCIe



DGX A100, DGX H100, HGX B200, …

Hardware-Coherent Memory Platforms

Hardware-coherent platforms connect the CPU and GPU over NVLink chip-to-chip (C2C) rather than PCIe



GH100, GB200, GB300, Vera Rubin …

NUMA Mode in the NVIDIA GPU Driver

- Exposes all GPU memory as standard NUMA nodes
  - Each GPU gets its own dedicated NUMA node
  - MIG devices get their own unique NUMA node as well

- NUMA node numbering is (essentially) predictable
  - Nodes 0-1 are system memory
  - Nodes 2-N are GPU memory

NUMA Mode in the NVIDIA GPU Driver


```
available: 34 nodes (0-33)
node 0 cpus: 0-71
node 0 size: 490533 MB
node 0 free: <some amount>
node 1 cpus: 72-143
node 1 size: 490813 MB
node 1 free: <some amount>

node 2     size: 188416 MB # GPU 0
node 2     free: 188416 MB
node 3-9   size: 0 MB      # GPU 0's reserved slots for MIG

node 10    size: 188416 MB # GPU 1
node 10    free: 188416 MB
node 11-17 size: 0 MB      # GPU 1's reserved slots for MIG

node 18    size: 188416 MB # GPU 2
node 18    free: 188416 MB
node 19-25 size: 0 MB      # GPU 2's reserved slots for MIG

node 26    size: 188416 MB # GPU 3
node 26    free: 188416 MB
node 27-33 size: 0 MB      # GPU 3's reserved slots for MIG
```

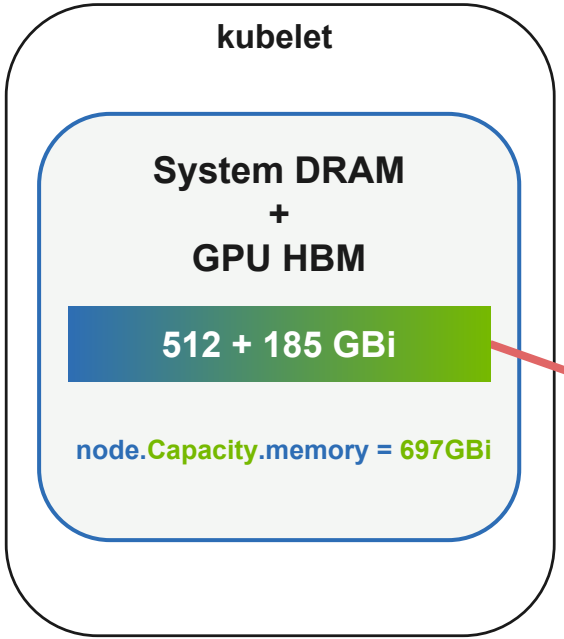
- Single GB200 as example
  - 34 NUMA nodes total
  - 4 for full GPUs
  - 7 per GPU for MIG

- NUMA nodes always present
  - Get populated by GPU driver when loaded
  - Memory moved as MIG devices come and go

NUMA Mode in the NVIDIA GPU Driver

- Benefits of NUMA mode
  - **One unified memory pool exposed via NUMA:** GPU workloads can overflow into system memory instead of crashing, and vice versa
  - **Automatic memory migration:** the OS moves data to wherever it's actively used without requiring the NVIDIA driver to do anything
  - **Manual placement control:** via standard Linux tools (`numactl`, `mbind`)
  - **Unified pointers:** a `malloc()` pointer is valid on the GPU, a `cudaMalloc()` pointer is valid on the CPU — no `cudaMemcpy()` required

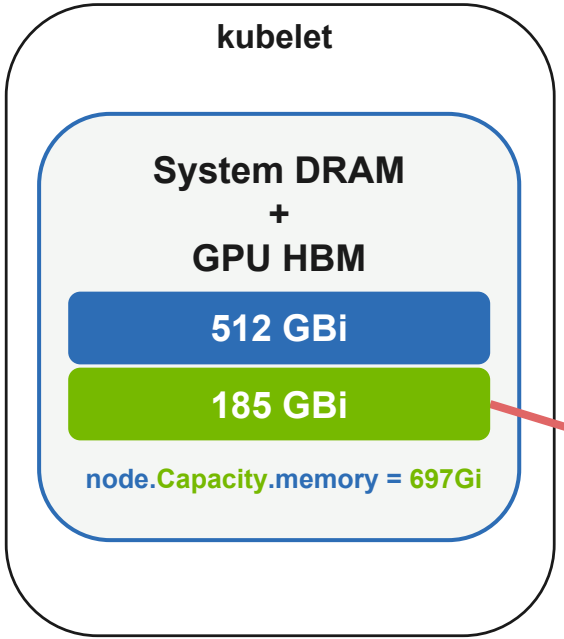
NUMA Mode and Kubernetes




```yaml
apiVersion: v1
kind: Pod
metadata:
  name: gpu-example
spec:
  containers:
    - name: gpu-example
      image: nvidia/cuda
      resources:
        limits:
          memory: 2GBi
          nvidia.com/gpu: 2
```

`malloc()` pulls from ***either*** system memory ***or*** GPU memory and gets charged to the pod's `cgroup` and counted toward `memory.limit`

NUMA Mode and Kubernetes




```yaml
apiVersion: v1
kind: Pod
metadata:
  name: gpu-example
spec:
  containers:
    - name: gpu-example
      image: nvidia/cuda
      resources:
        limits:
          memory: 2GBi
          nvidia.com/gpu: 2
```

`cudaMalloc()` pulls ***only*** from GPU memory and ***doesn't*** get charged to the pod's `cgroup` or counted toward `memory.limit`

NUMA Mode and Kubernetes

- Implications
  - **Memory over-reporting:** If even one `cudaMalloc()` call is made by any pod in the system, it throws off all `kubelet` accounting and pods may be scheduled to run instead of queuing on memory pressure
  - **The Pod API contract breaks:** `memory.limit` was never meant to bound GPU memory, but GPU memory allocated via standard **malloc()** now counts against it, potentially causing a pod to get OOM-killed for legitimate GPU usage
  - **Isolating GPU memory among pods:** Any container can **malloc()** from any GPU's memory, even if it wasn't assigned a GPU!

NUMA Mode and Kubernetes

- Implications
  - **Memory over-reporting:** If even one `cudaMalloc()` call is made by any pod in the system, it throws off all `kubelet` accounting and pods may be scheduled to run instead of
  - **Goal:** Address these issues with no changes to Kubernetes and no loss of CPU-GPU coherence.
  - **Isolating GPU memory among pods:** Any container can **malloc()** from any GPU's memory, even if it wasn't assigned a GPU!

Outline

- What we tried first and why it didn’t work
- Coherent Driver-based Memory Management (CDMM)
- Residual changes needed in Kubernetes even with CDMM
- Kubernetes changes required to support original NUMA mode
- Questions & Feedback

Hide GPU NUMA Nodes from Kubelet

Idea: Use existing **`memoryManager`** flags to hide GPU memory from the **kubelet**

```
reservedMemory:
  - numaNode: 2       # GPU 0
    limits:
      memory: 185Gi
  - numaNode: 10      # GPU 1
    limits:
      memory: 185Gi
  - numaNode: 18      # GPU 2
    limits:
      memory: 185Gi
  - numaNode: 26      # GPU 3
    limits:
      memory: 185Gi
systemReserved:
  memory: 745Gi       # 4*185Gi
```

Hide GPU NUMA Nodes from Kubelet

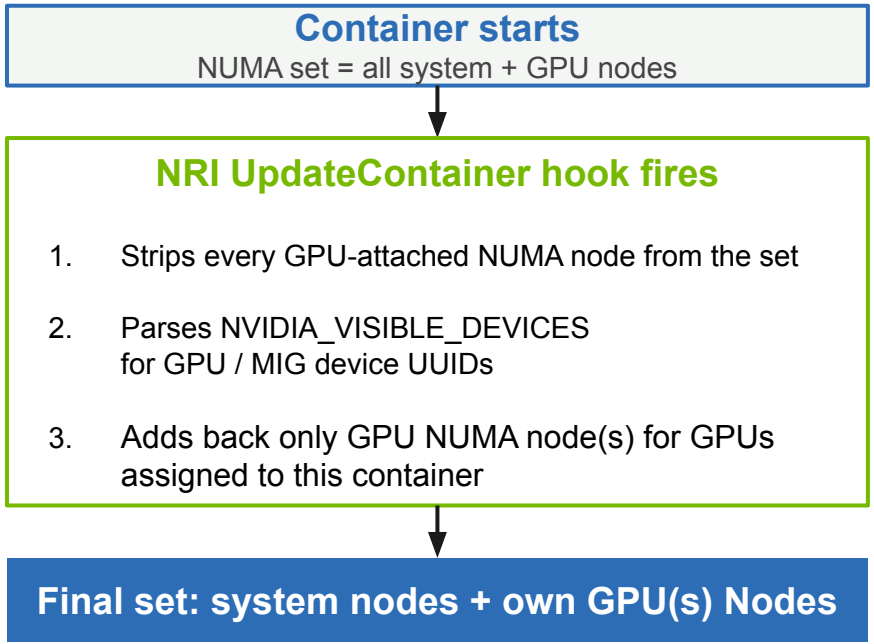
Idea: Use existing **`memoryManager`** flags to hide GPU memory from the **kubelet**

```
reservedMemory:
  - numaNode: 2       # GPU 0
    limits:
      memory: 185Gi
  - numaNode: 10      # GPU 1
    limits:
      memory: 185Gi
  - numaNode: 18      # GPU 2
    limits:
      memory: 185Gi
  - numaNode: 26      # GPU 3
    limits:
      memory: 185Gi
systemReserved:
  memory: 745Gi       # 4*185Gi
```

- Fragile to count memory exactly
- Must be added *after* GPU driver loaded (problematic for GPU Operator and GKE)
- No way to account for MIG as devices get created / destroyed dynamically
- Only fixes **memory over-reporting**, not the other two issues

Hide GPU NUMA Nodes from Workload

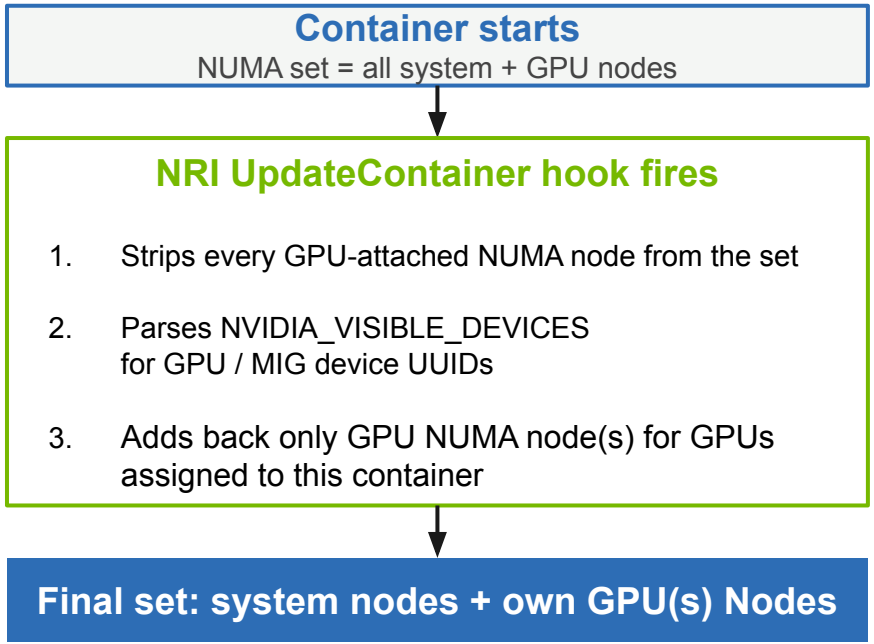
Idea: Run an NRI plugin to dynamically restrict NUMA node access



Hide GPU NUMA Nodes from Workload

KubeCon CloudNativeCon North America 2026

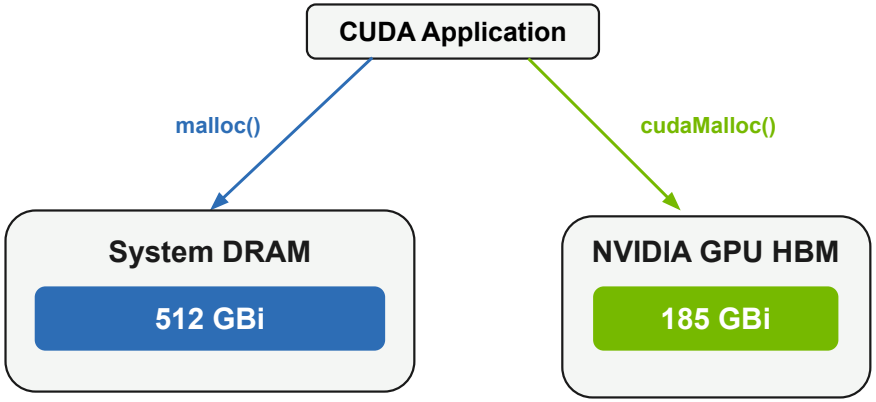
Idea: Run an NRI plugin to dynamically restrict NUMA node access



- Gives us a way to selectively isolate GPU memory in appropriate pods
- Requires extra component installed on the system to run NRI
- Gets complicated if we want to support CDI to discover injected GPUs
- Still doesn't address `memory.limits` issue within pods that have access to GPU NUMA nodes

Introducing CDMM

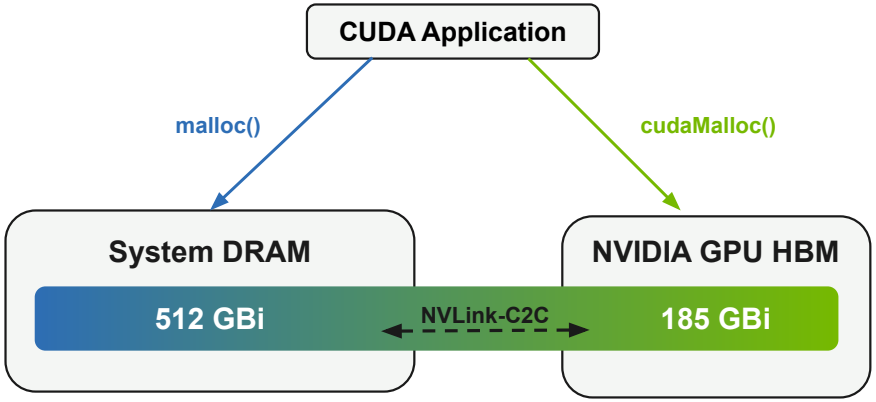
- Coherent Driver-based Memory Management
- An new mode in the underlying NVIDIA GPU driver
- Avoids onlining GPU memory as NUMA nodes



- Restores existing semantics expected by Kubernetes

Introducing CDMM

- Coherent Driver-based Memory Management
- An new mode in the underlying NVIDIA GPU driver
- Avoids onlining GPU memory as NUMA nodes



- Restores existing semantics expected by Kubernetes
- Retains coherency between GPU and system memory over NVLink C2C

Introducing CDMM

- Benefits of NUMA mode
  - ~~**One unified memory pool exposed via NUMA:** GPU workloads can overflow into system memory instead of crashing, and vice versa~~
  - ~~**Automatic memory migration:** the OS moves data to wherever it's actively used without requiring the NVIDIA driver to do anything~~
  - ~~**Manual placement control:** via standard Linux tools (`numactl`, `mbind`)~~
  - **Unified pointers:** a `malloc()` pointer is valid on the GPU, a `cudaMalloc()` pointer is valid on the CPU — no `cudaMemcpy()` required

Residual effects of NUMA mode


```
available: 34 nodes (0-33)
node 0 cpus: 0-71
node 0 size: 490533 MB
node 0 free: <some amount>
node 1 cpus: 72-143
node 1 size: 490813 MB
node 1 free: <some amount>

node 2     size: 0 MB  # GPU 0
node 3-9   size: 0 MB  # GPU 0's reserved slots for MIG

node 10    size: 0 MB  # GPU 1
node 11-17 size: 0 MB  # GPU 1's reserved slots for MIG

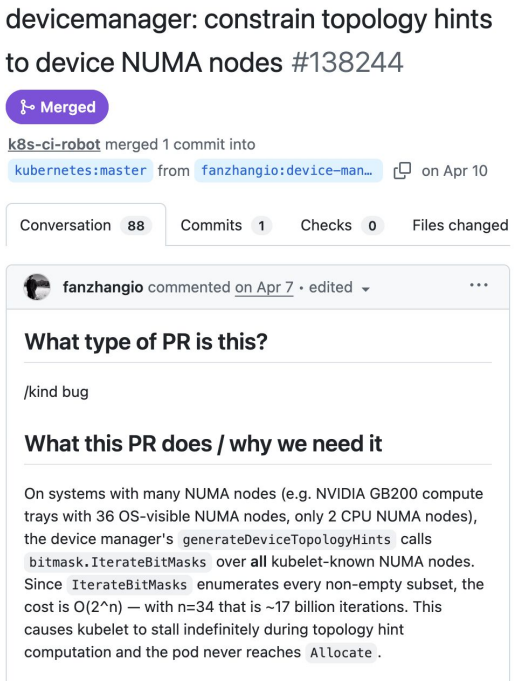
node 18    size: o MB # GPU 2
node 19-25 size: 0 MB # GPU 2's reserved slots for MIG

node 26    size: 0 MB # GPU 3
node 27-33 size: 0 MB # GPU 3's reserved slots for MIG
```

- Phantom NUMA nodes for all GPUs and MIG devices with 0 memory
- Set by ACPI before the NVIDIA driver ever comes online
- Required to support option for selecting NUMA mode at all

Residual effects of NUMA mode

Phantom NUMA nodes triggered a state explosion in the `TopologyManager`



devicemanager: constrain topology hints to device NUMA nodes #138244

Merged

k8s-ci-robot merged 1 commit into kubernetes:master from fanzhangio:device-man… on Apr 10

Conversation 88 Commits 1 Checks 0 Files changed

fanzhangio commented on Apr 7 • edited

What type of PR is this?

/kind bug

What this PR does / why we need it

On systems with many NUMA nodes (e.g. NVIDIA GB200 compute trays with 36 OS-visible NUMA nodes, only 2 CPU NUMA nodes), the device manager's `generateDeviceTopologyHints` calls `bitmask.IterateBitMasks` over **all** kubelet-known NUMA nodes. Since `IterateBitMasks` enumerates every non-empty subset, the cost is O(2^n) — with n=34 that is ~17 billion iterations. This causes kubelet to stall indefinitely during topology hint computation and the pod never reaches `Allocate`.

- **`TopologyManager`** iterates all combinations of NUMA nodes looking for possible device–CPU alignment
- With 34 NUMA nodes that becomes $2^{34} \approx 17B$ iterations causing the **`kubelet`** to stall forever
- Most of those nodes are empty GPU HBM, irrelevant to the device being allocated
- Fix: only iterate NUMA nodes that actually host a device for that resource; merged in v1.36

Long Term Solution

Distinguish between NUMA memory onlined as CPU memory vs. GPU memory #5726

Closed as not planned



klueska opened on Dec 8, 2025 — Last edited by klueska — Contributor

Enhancement Description

- **One-line enhancement description (can be used as a release note):**
  Build a comprehensive solution to ensure Kubernetes can distinguish between NUMA nodes brought online with CPU memory vs. GPU memory (or more generally any device memory). This will likely take the form of changes to `cadvisor` at the lowest layers.

https://github.com/kubernetes/enhancements/issues/5726

Long Term Solution

Distinguish between NU… …onlined as CPU memory vs. GPU memory #5726

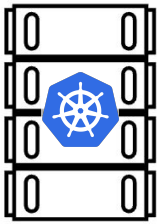
Closed as not planned

klueska opened on Dec 8, 2025 — Last edited by klueska — Contributor

Enhancement Description

- One-line enhancement description (can be…
  Build a comprehensive solution to ensure Ku… …ween NUMA nodes brought online with CPU memory vs. GPU memory (or more generally any device … …the form of changes to `cadvisor` at the lowest layers.

https://github.com/ku… …ments/issues/5726



4x DGX Spark

512 GB
Up to 700B
parameters

Timeline

R535 (535.54.03) · Jun 2023

GH200 Ships

- GPU memory onlined as a NUMA node by default
- No CDMM option exists

R580 (580.65.06) · Aug 2025

CDMM introduced

- Driver option to switch between NUMA and CDMM mode introduced
- GPU Operator defaults to CDMM for GB200

R615 (615.71.10) · Oct 2026

CDMM becomes default

- Default across all hardware-coherent platforms
- NUMA mode still fully supported

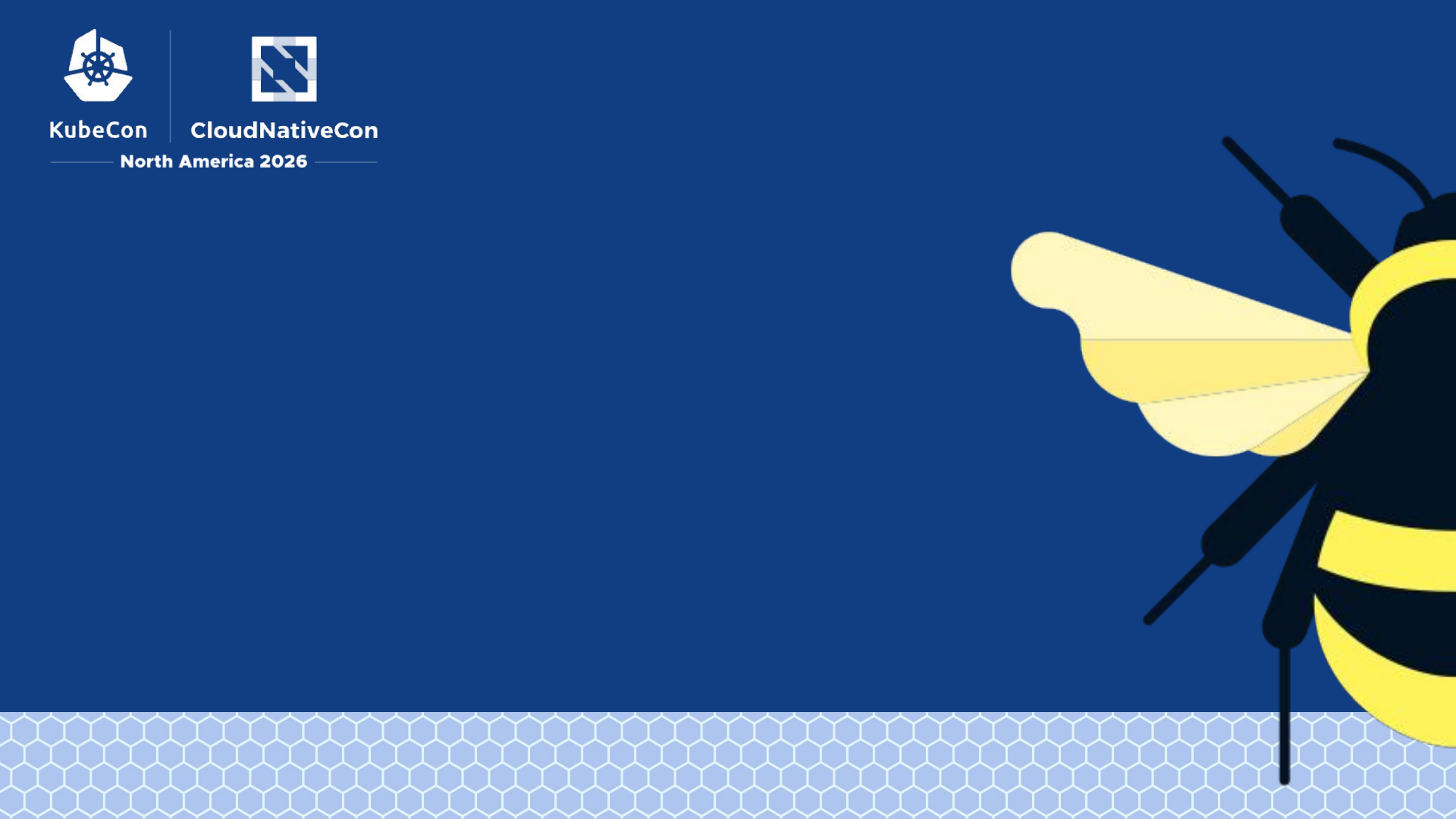
```
echo "options nvidia NVreg_CoherentGPUMemoryMode=<driver | numa>" > /etc/modprobe.d/nvidia-openrm.conf
```

Questions & Feedback

KubeCon CloudNativeCon North America 2026



When GPUs Become NUMA Nodes: Challenges Running Kubernetes with Memory-Coherent Hardware


KubeCon
CloudNativeCon
North America 2026