
KubeCon
CloudNativeCon
North America 2024

KubeCon | CloudNativeCon

North America 2024

Which GPU Sharing Strategy Is Right for You? - A Benchmark Study using DRA

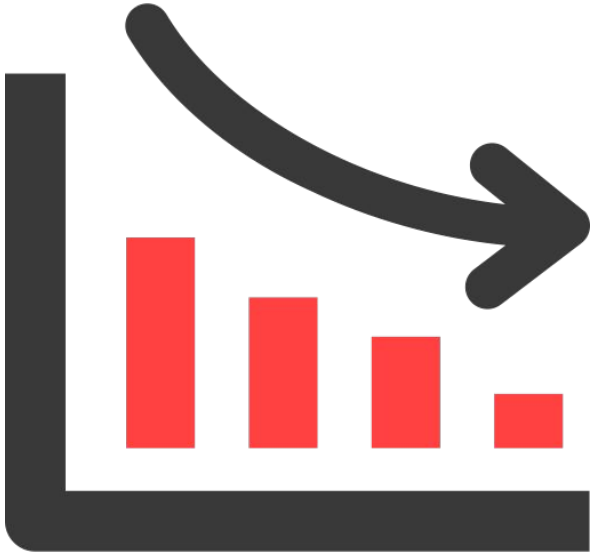
Kevin Klues & Yuan Chen, NVIDIA

Why Share GPUs?

Why Share GPUs?

Increase Utilization

Decrease Costs



More Efficient Use of GPUs → Fewer GPUs to Get Work Done→ Lower Costs

Previous GPU Sharing Talks at KubeCon

KubeCon EU 2024 (Paris)

- Keynote: Accelerating AI Workloads with GPUs in Kubernetes - Kevin Klues & Sanjay Chatterjee, NVIDIA
- Unleashing the Power of DRA for Just-in-Time GPU Slicing - Abhishek Malvankar & Olivier Tardieu, IBM
- Sharing Is Caring: GPU Sharing and CDI in Device Plugins - Evan Lezar, NVIDIA & David Porter, Google
- Increasing GPU Utilisation on K8s Clusters Dedicated for AI/ML Workloads - Maciej Mazur, Canonical & Andreea Munteanu, Canonical
- Maximizing GPU Utilization Over Multi-Cluster: Challenges and Solutions for Cloud-Native AI Platform - William Wang & Hongcai Ren, Huawei

KubeCon NA 2023 (Chicago)

- Unlocking the Full Potential of GPUs for AI Workloads on Kubernetes - Kevin Klues, NVIDIA
- Maximizing GPU Utilization in Kubernetes with Virtual Kubelets - Goutam Verma & Dean Troyer, Salad

Google NVIDIA IBM Canonical HUAWEI salad

Previous GPU Sharing Talks at KubeCon

KubeCon EU 2023 (Amsterdam)

Efficient Access to Shared GPU Resources: Mechanisms and Use Cases - Diogo Filipe Tomas Guerra & Diana Gaponcic, CERN

KubeCon EU 2022 (Valencia)

Improving GPU Utilization using Kubernetes - Maulin Patel & Pradeep Venkatachalam, Google

KubeCon EU 2021 (Virtual)

A Deep Dive on Supporting Multi-Instance GPUs in Containers and Kubernetes - Kevin Klues, NVIDIA

KubeCon EU 2020 (Virtual)

Is Sharing GPU to Multiple Containers Feasible? - Samed Güner, SAP

KubeCon NA 2019 (San Diego)

Lightning Talk: Sharing a GPU Among Multiple Containers - Patrick McQuighan, Algorithmia

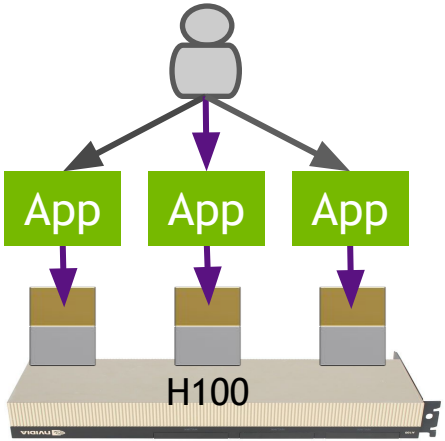
KubeCon EU 2019 (Barcelona)

GPU Sharing for Machine Learning Workload on Kubernetes - Henry Zhang & Yang Yu, VMware

CERN Google NVIDIA SAP ALGORITHMIA vmware

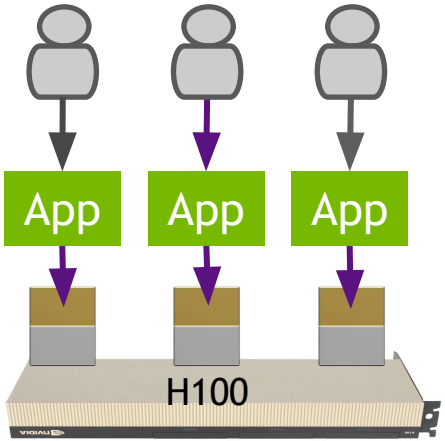
Use Cases for GPU Sharing

Single User → Multiple Apps



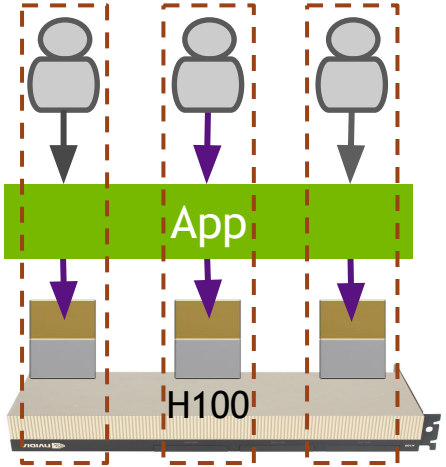
Multiple inference jobs

Single Tenant → Multi-User



Jupyter notebooks for model exploration
CI/CD Pipelines

Multi-Tenant → Multi-User

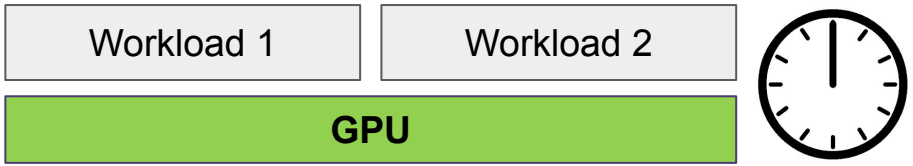


Managed Cloud Services

Space vs. Time Partitioning

Space vs. Time Partitioning

Space



Advantages:

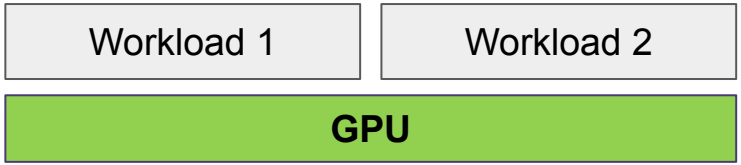
- Fully resident at all times
- No context switch overhead
- Predictable performance

Disadvantages:

- Subset of GPU resources
- Limited number of clients

Space vs. Time Partitioning

Space



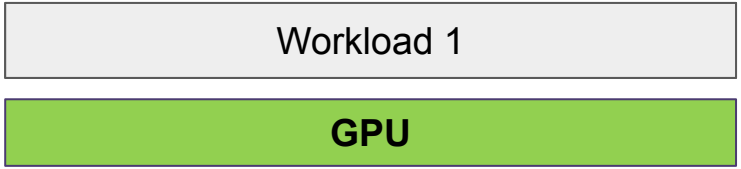
Advantages:

- Fully resident at all times
- No context switch overhead
- Predictable performance

Disadvantages:

- Subset of GPU resources
- Limited number of clients

Time



Advantages:

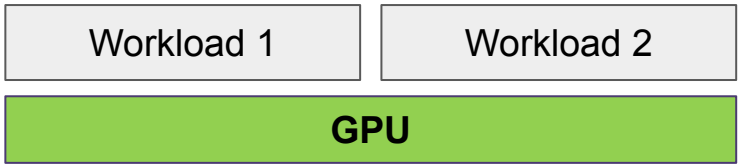
- Full set of GPU resources
- Unlimited number of clients

Disadvantages:

- Only resident a fraction of the time
- Additional context switch overhead
- Unpredictable performance

Space vs. Time Partitioning

Space



Advantages:

- Fully resident at all times
- No context switch overhead
- Predictable performance

Disadvantages:

- Subset of GPU resources
- Limited number of clients

Time

Advantages:

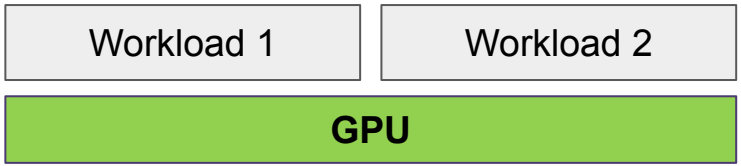
- Full set of GPU resources
- Unlimited number of clients

Disadvantages:

- Only resident a fraction of the time
- Additional context switch overhead
- Unpredictable performance

Space vs. Time Partitioning

Space



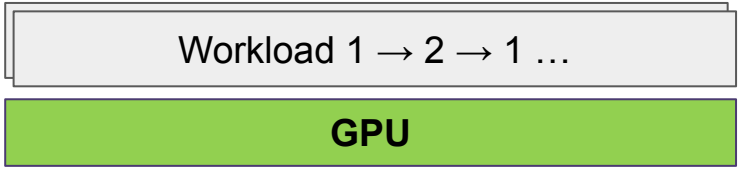
Advantages:

- Fully resident at all times
- No context switch overhead
- Predictable performance

Disadvantages:

- Subset of GPU resources
- Limited number of clients

Time



Advantages:

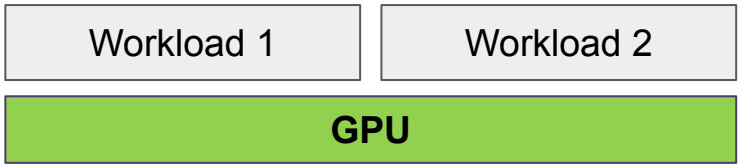
- Full set of GPU resources
- Unlimited number of clients

Disadvantages:

- Only resident a fraction of the time
- Additional context switch overhead
- Unpredictable performance

Space vs. Time Partitioning

Space



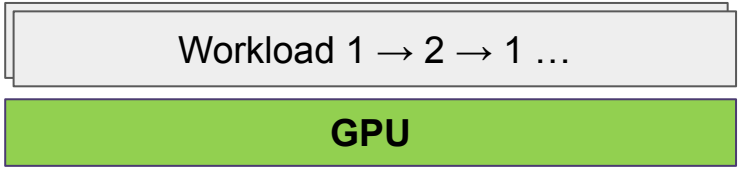
Advantages:

- Fully resident at all times
- No context switch overhead
- Predictable performance

Disadvantages:

- Subset of GPU resources
- Limited number of clients

Time



Advantages:

- Full set of GPU resources
- Unlimited number of clients

Disadvantages:

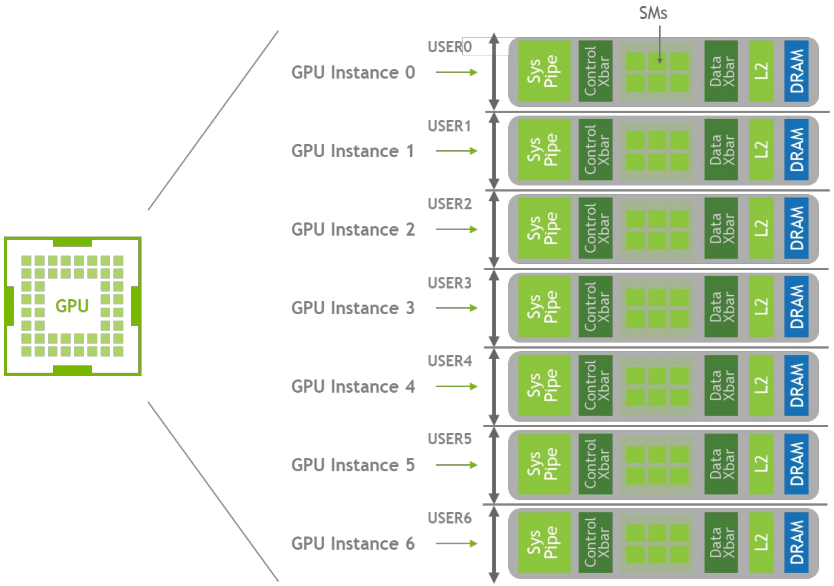
- Only resident a fraction of the time
- Additional context switch overhead
- Unpredictable performance

Inherent Tradeoff in Choosing One Strategy over the Other

Hardware vs. Software-Based Space Partitioning

Hardware vs. Software-Based Space Partitioning

Multi-Instance GPU (MIG)

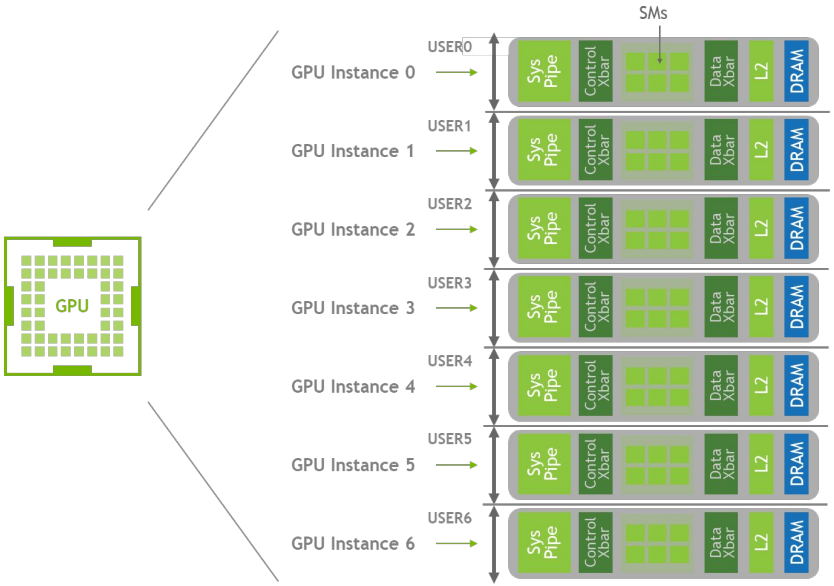


Physical partitioning

Full isolation between workloads

Hardware vs. Software-Based Space Partitioning

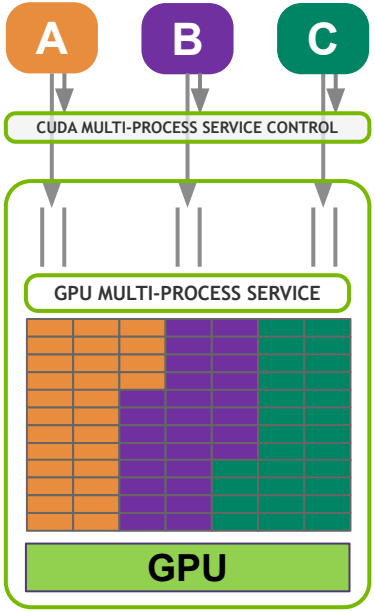
Multi-Instance GPU (MIG)



Physical partitioning

Full isolation between workloads

Multi-Process Service (MPS)



Logical partitioning

Limited isolation between workloads

Hardware vs. Software-Based Space Partitioning

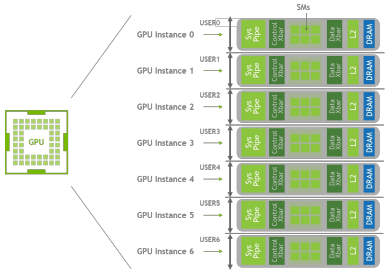
Multi-Instance GPU (MIG)

Advantages:

- Full Fault Isolation
- Guaranteed Memory Bandwidth QoS
- Suitable for Multi-Tenant Environments

Disadvantages:

- Fixed Size Partitions (multiple dimensions)



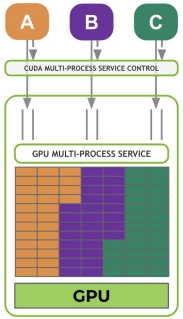
Multi-Process Service (MPS)

Advantages:

- Flexible Partition Size (multiple dimensions)

Disadvantages:

- Limited Fault Isolation
- No Memory Bandwidth QoS
- Not Suitable for Multi-Tenant Environments



Hardware vs. Software-Based Space Partitioning

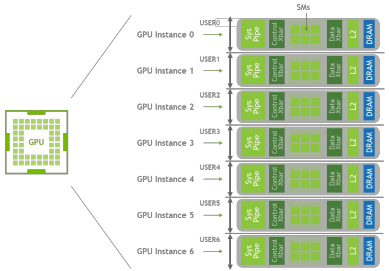
Multi-Instance GPU (MIG)

Advantages:

- Full Fault Isolation
- Guaranteed Memory Bandwidth QoS
- Suitable for Multi-Tenant Environments

Disadvantages:

- Fixed Size Partitions (multiple dimensions)



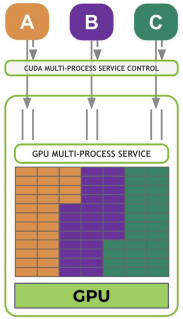
Multi-Process Service (MPS)

Advantages:

- Flexible Partition Size (multiple dimensions)

Disadvantages:

- Limited Fault Isolation
- No Memory Bandwidth QoS
- Not Suitable for Multi-Tenant Environments

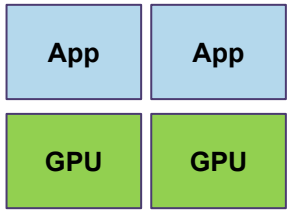


(Layered) Sharing Strategies

(Layered) Sharing Strategies

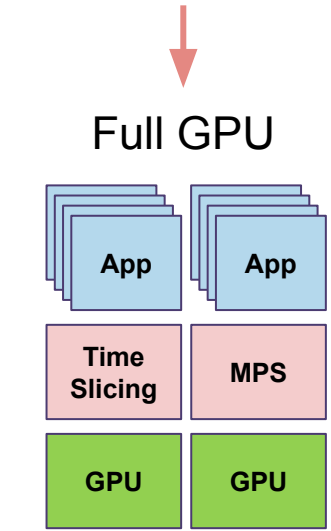
2 Apps
One Per Full GPU

Full GPU



(Layered) Sharing Strategies

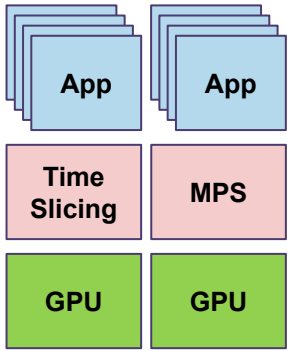
Many Apps
One per full GPU
Shared via either
Time-Slicing or MPS



(Layered) Sharing Strategies

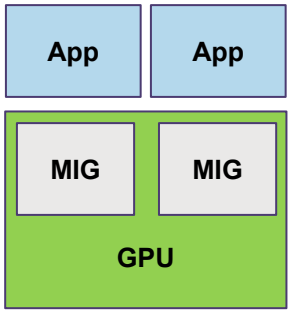
Many Apps
One per full GPU
Shared via either
Time-Slicing or MPS

Full GPU



2 Apps
One Per MIG Device

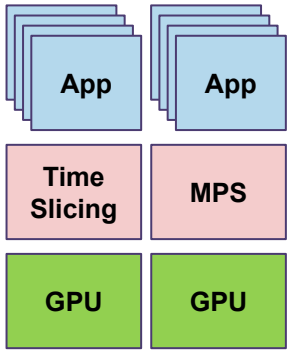
MIG Device



(Layered) Sharing Strategies

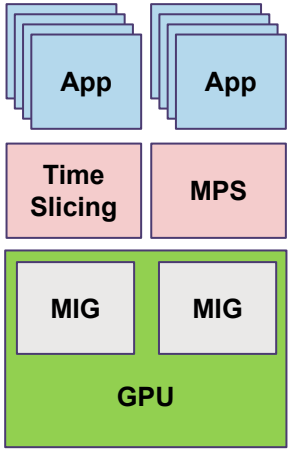
Many Apps
One per full GPU
Shared via either
Time-Slicing or MPS

Full GPU



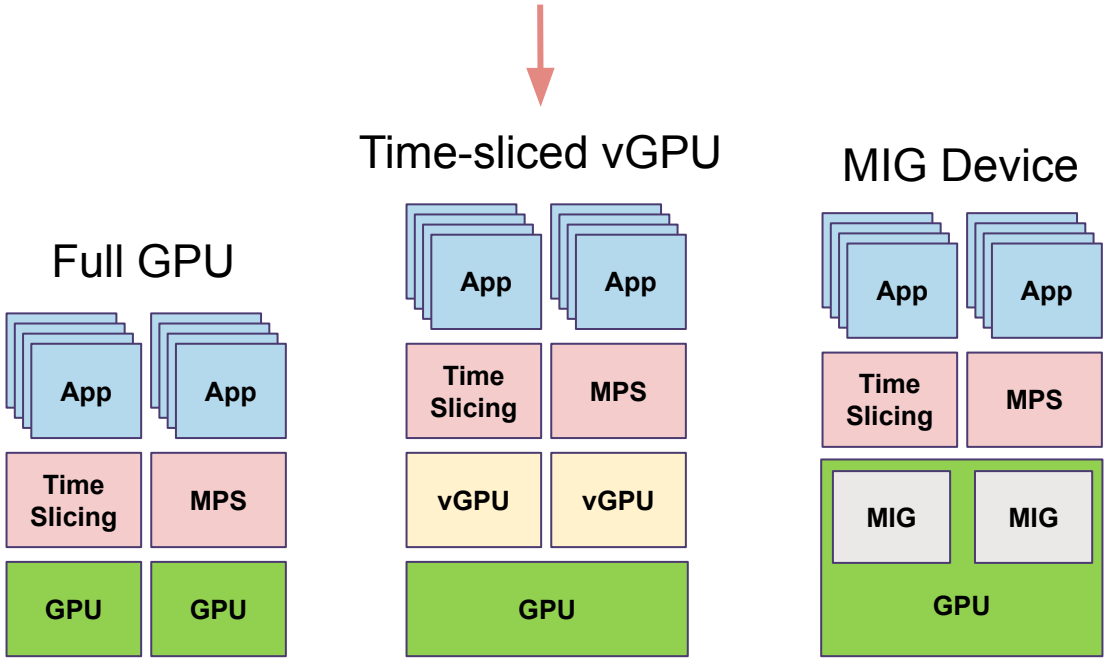
Many Apps
One per MIG Device
Shared via either
Time-Slicing or MPS

MIG Device



(Layered) Sharing Strategies

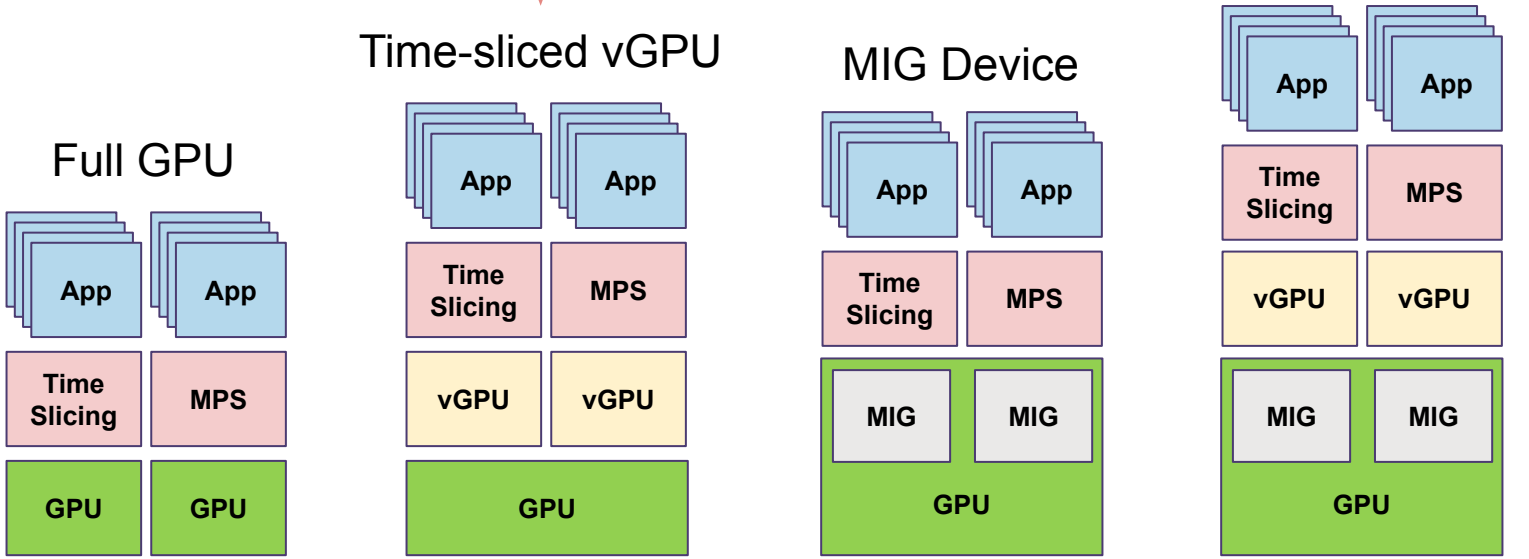
Looks like a full GPU but is Time-Sliced by Hypervisor in a VM



(Layered) Sharing Strategies

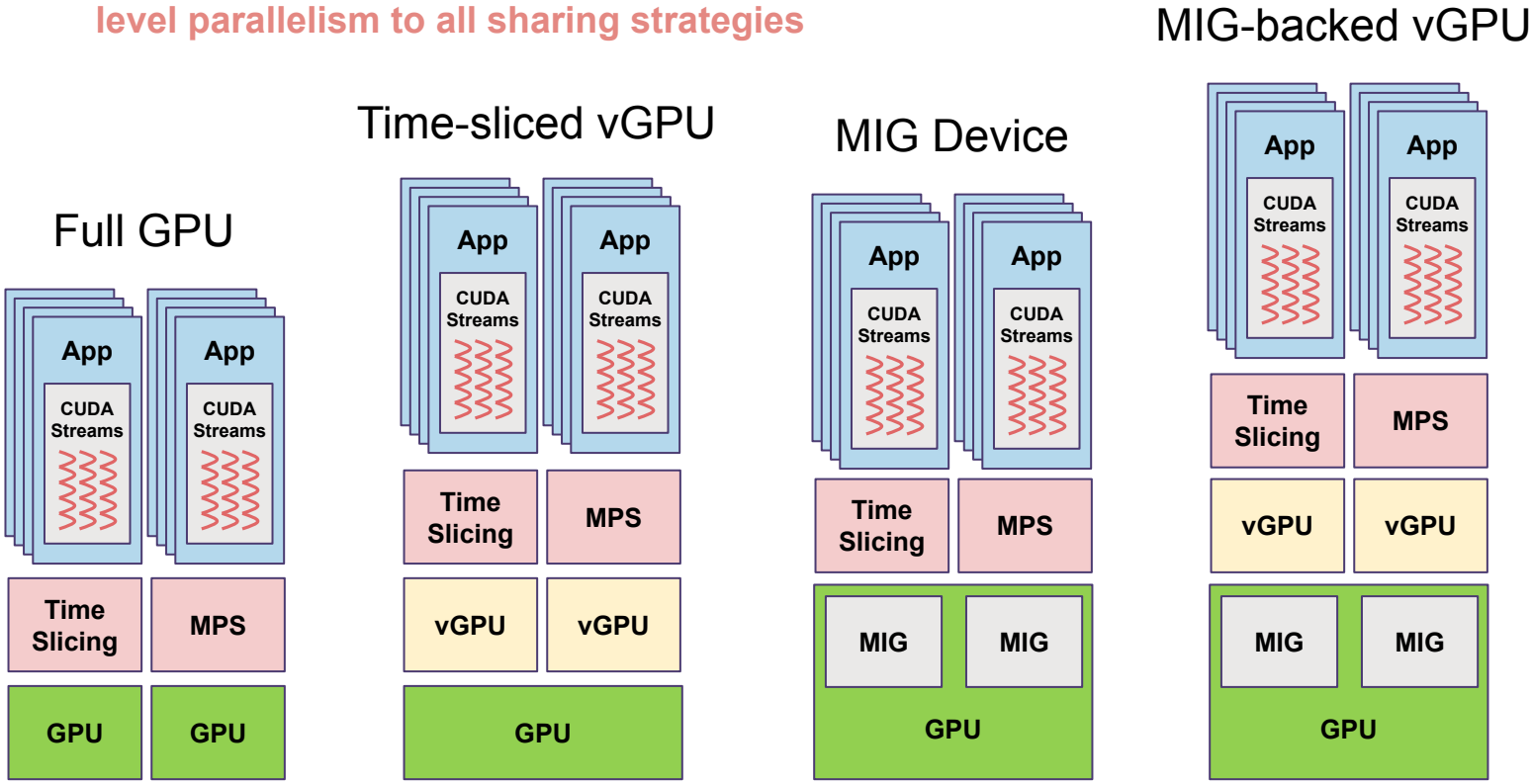
Looks like a full GPU but is Time-Sliced by Hypervisor in a VM

Looks like a MIG Device but is Isolated in a VM



(Layered) Sharing Strategies

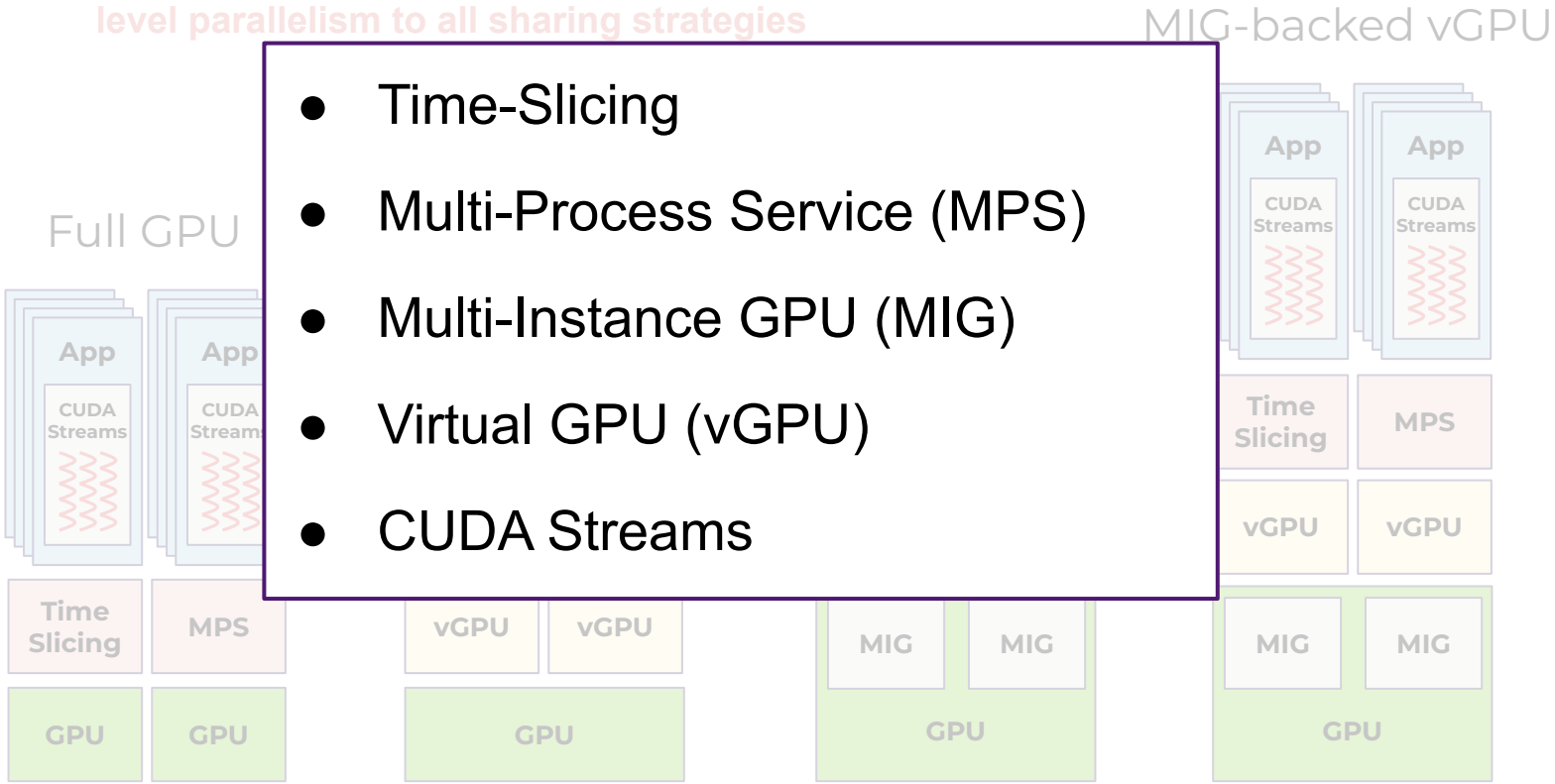
CUDA streams provide an extra-level of application level parallelism to all sharing strategies



(Layered) Sharing Strategies

CUDA streams provide an extra-level of application level parallelism to all sharing strategies

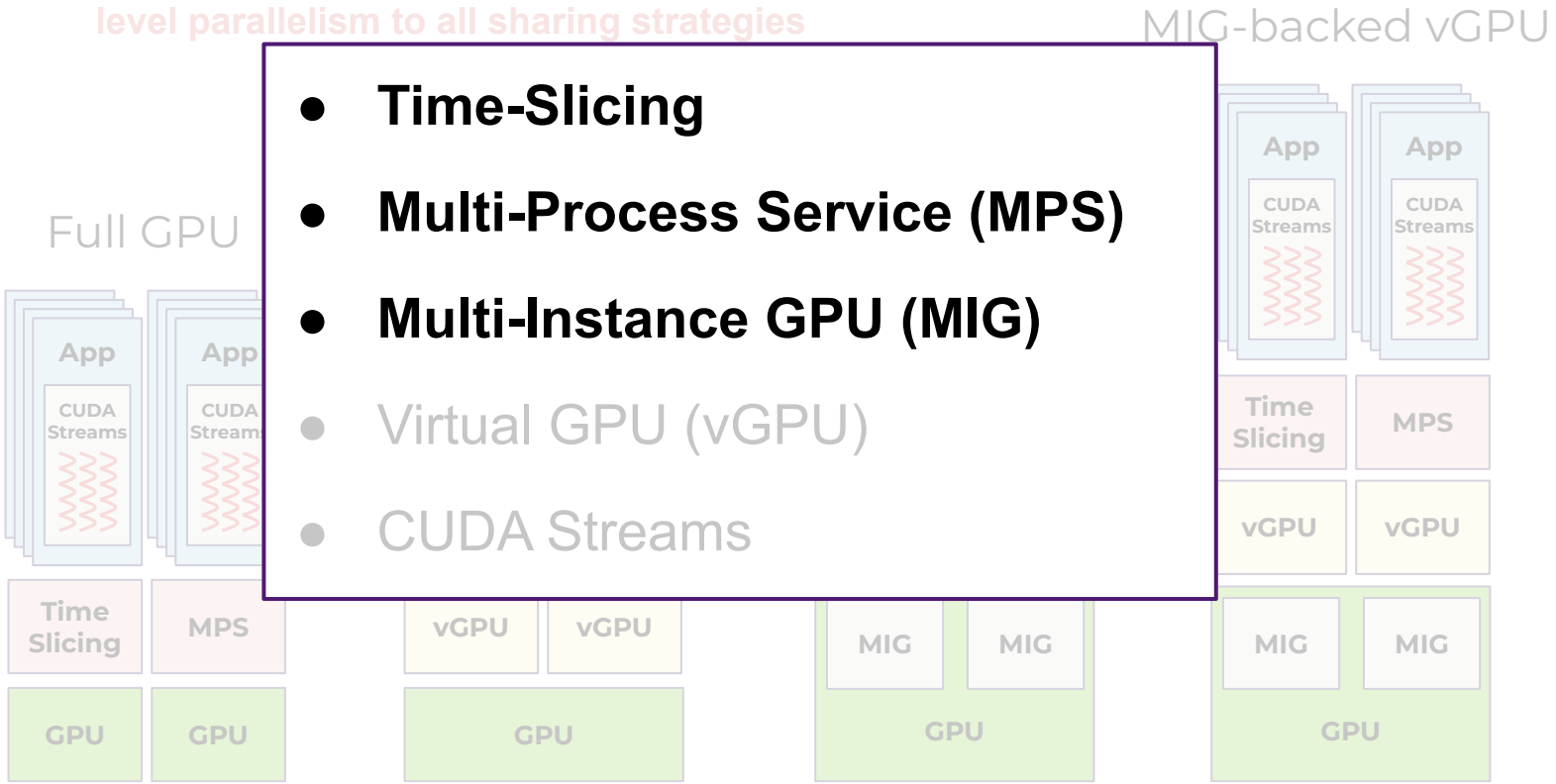
- Time-Slicing
- Multi-Process Service (MPS)
- Multi-Instance GPU (MIG)
- Virtual GPU (vGPU)
- CUDA Streams



(Layered) Sharing Strategies

CUDA streams provide an extra-level of application level parallelism to all sharing strategies

- **Time-Slicing**
- **Multi-Process Service (MPS)**
- **Multi-Instance GPU (MIG)**
- Virtual GPU (vGPU)
- CUDA Streams



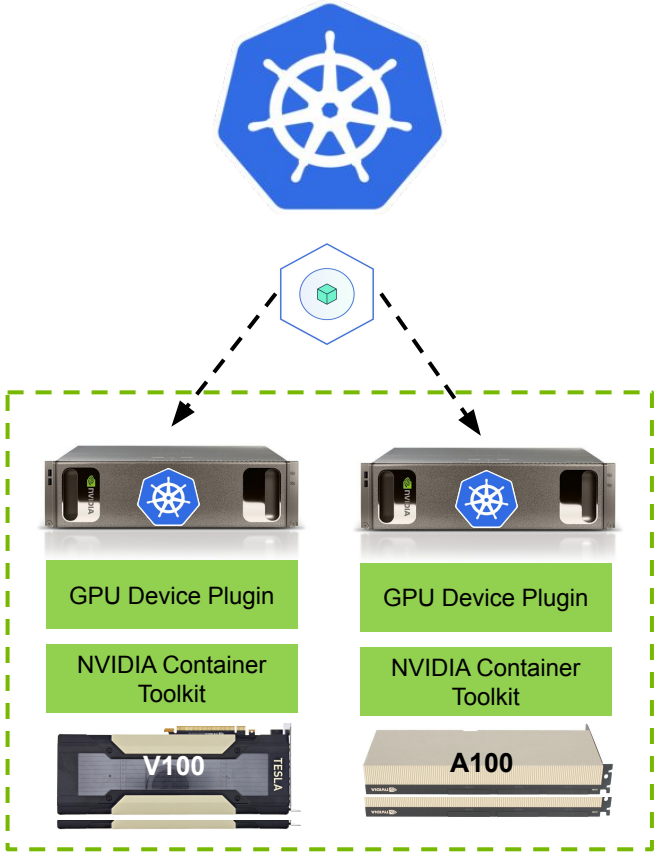
GPU Sharing in Kubernetes Today

Dedicated GPUs


```yaml
apiVersion: v1
kind: Pod
metadata:
  name: gpu-example
spec:
  containers:
    - name: gpu-example
      image: nvidia/cuda
      resources:
        limits:
          nvidia.com/gpu: 1
```

App

GPU

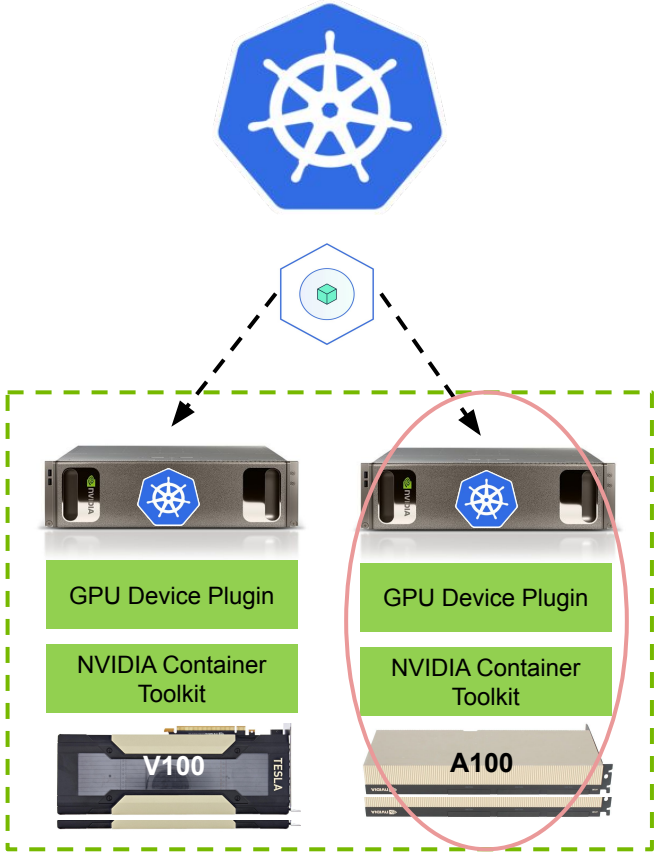


Dedicated GPUs


```yaml
apiVersion: v1
kind: Pod
metadata:
  name: gpu-example
spec:
  containers:
    - name: gpu-example
      image: nvidia/cuda
      resources:
        limits:
          nvidia.com/gpu: 1
  nodeSelector:
    nvidia.com/gpu.product: A100-PCIE-40GB
    nvidia.com/cuda.runtime: 11.4
    nvidia.com/cuda.driver: 470.161.03
```

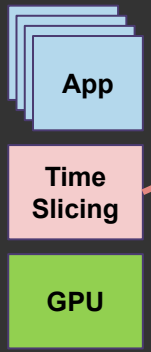
App

GPU



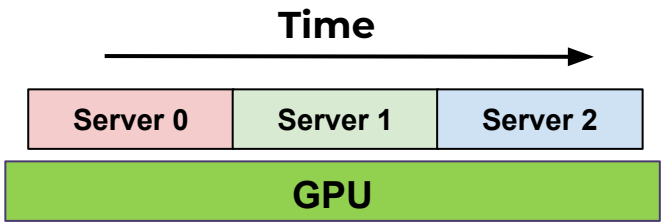
Time-Slicing (On Dedicated GPUs)


```yaml
apiVersion: v1
kind: Pod
metadata:
  name: gpu-example
spec:
  containers:
    - name: gpu-example
      image: nvidia/cuda
      resources:
        limits:
          nvidia.com/gpu.shared: 1
  nodeSelector:
    nvidia.com/gpu.product: A100-PCIE-40GB
    nvidia.com/cuda.runtime: 11.4
    nvidia.com/cuda.driver: 470.161.03
```



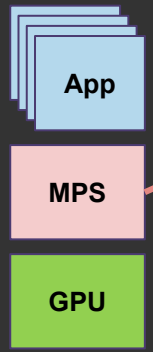

```yaml
version: v1
sharing:
  timeSlicing:
    resources:
    - name: nvidia.com/gpu
      replicas: 3
    ...
```

Per node k8s-device-plugin config file



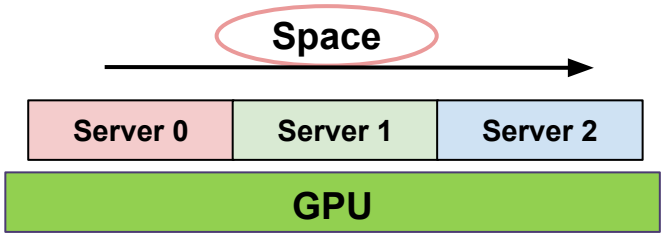
MPS (On Dedicated GPUs)


```yaml
apiVersion: v1
kind: Pod
metadata:
  name: gpu-example
spec:
  containers:
    - name: gpu-example
      image: nvidia/cuda
      resources:
        limits:
          nvidia.com/gpu.shared: 1
  nodeSelector:
    nvidia.com/gpu.product: A100-PCIE-40GB
    nvidia.com/cuda.runtime: 11.4
    nvidia.com/cuda.driver: 470.161.03
```



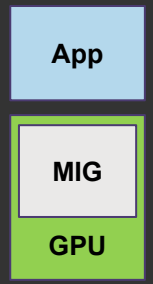

```yaml
version: v1
sharing:
  mps:
    resources:
    - name: nvidia.com/gpu
      replicas: 3
    ...
```

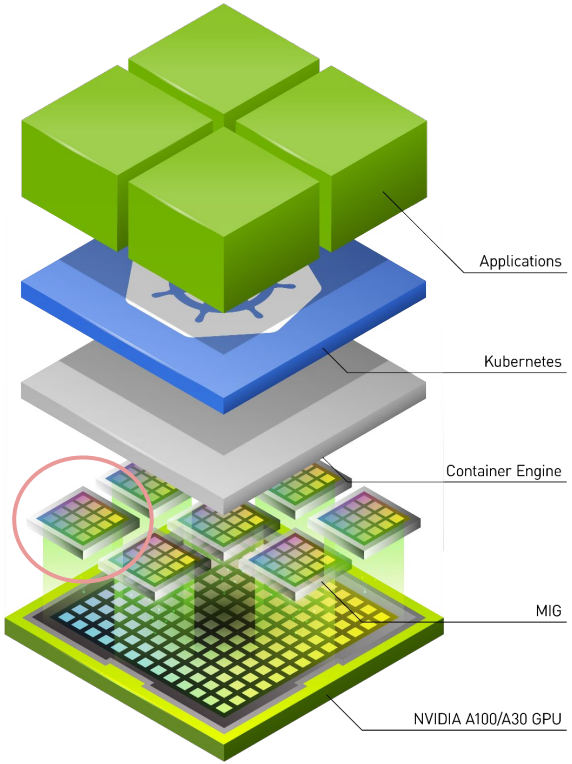
Per node k8s-device-plugin config file



Multi-Instance GPUs (MIG)

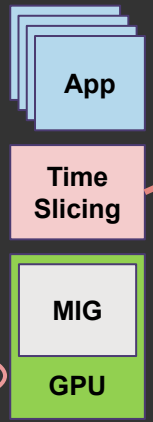

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: gpu-example
spec:
  containers:
    - name: gpu-example
      image: nvidia/cuda
      resources:
        limits:
          nvidia.com/mig-1g.5gb: 1
  nodeSelector:
    nvidia.com/gpu.product: A100-PCIE-40GB
    nvidia.com/cuda.runtime: 11.4
    nvidia.com/cuda.driver: 470.161.03
```





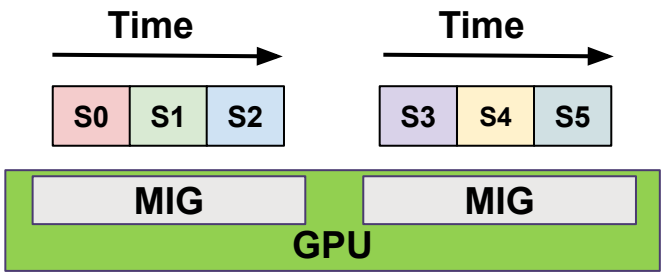
Time-Slicing (On MIG)


```yaml
apiVersion: v1
kind: Pod
metadata:
  name: gpu-example
spec:
  containers:
    - name: gpu-example
      image: nvidia/cuda
      resources:
        limits:
          nvidia.com/mig-1g.5gb.shared: 1
  nodeSelector:
    nvidia.com/gpu.product: A100-PCIE-40GB
    nvidia.com/cuda.runtime: 11.4
    nvidia.com/cuda.driver: 470.161.03
```



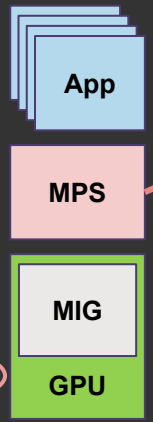

```yaml
version: v1
sharing:
  timeSlicing:
    resources:
    - name: nvidia.com/mig-1g.5gb
      replicas: 3
    ...
```

Per node k8s-device-plugin config file



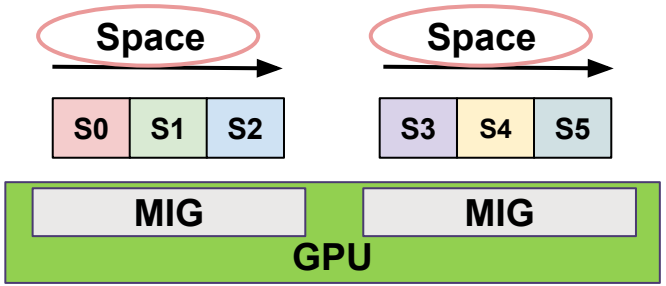
MPS (On MIG)


```yaml
apiVersion: v1
kind: Pod
metadata:
  name: gpu-example
spec:
  containers:
    - name: gpu-example
      image: nvidia/cuda
      resources:
        limits:
          nvidia.com/mig-1g.5gb.shared: 1
  nodeSelector:
    nvidia.com/gpu.product: A100-PCIE-40GB
    nvidia.com/cuda.runtime: 11.4
    nvidia.com/cuda.driver: 470.161.03
```




```yaml
version: v1
sharing:
  mps:
    resources:
    - name: nvidia.com/mig-1g.5gb
      replicas: 3
    ...
```

Per node k8s-device-plugin config file



Limitations

- No control over how time-sliced GPUs are shared between jobs
- No control over how MPS-partitioned GPUs are shared between jobs
- No ability to precisely control fraction of GPU handed out per job
- No ability to dynamically provision MIG devices based on incoming requests

Limitations

- No control over how time-sliced GPUs are shared between jobs
- No control over how MPS-partitioned GPUs are shared between jobs
- **No ability to precisely control fraction of GPU handed out per job**
- No ability to dynamically provision MIG devices based on incoming requests

Scheduling Frameworks with Fractional GPUs

run:ai

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: gpu-pod
spec:
  schedulerName: runai-scheduler
  containers:
    - name: cuda-container
      image: nvcr.io/nvidia/cuda:12.6.2
      resources:
        limits:
          nvidia.com/gpu: "0.5"
```

VOLCANO

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: gpu-pod
spec:
  schedulerName: volcano
  containers:
    - name: cuda-container
      image: nvcr.io/nvidia/cuda:12.6.2
      resources:
        limits:
          volcano.sh/gpu-number: 1
          volcano.sh/gpu-memory: 1024
```

HAMi

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: gpu-pod
spec:
  schedulerName: hami-scheduler
  containers:
    - name: cuda-container
      image: nvcr.io/nvidia/cuda:12.6.2
      resources:
        limits:
          nvidia.com/gpu: 1
          nvidia.com/gpumem: 1024
```

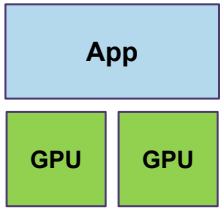
GPU Sharing with DRA

Dynamic Resource Allocation (DRA)

- New way of requesting resources available (as an ***alpha*** feature) in Kubernetes 1.26+
- Graduating to ***beta*** in the upcoming 1.32 release in December
- Provides an ***alternative*** to the “count-based” interface of e.g. `nvidia.com/gpu:2`
- Provides a much richer API for requesting / configuring resources
- Inspired by the persistent volume API
- Key concepts: `DeviceClass` and `ResourceClaim`

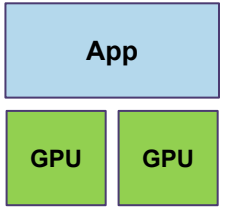
Dedicated GPUs

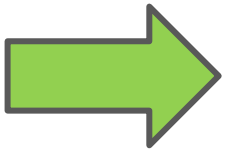

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: gpu-example
spec:
  containers:
  - name: ctr
    image: nvidia/cuda
    resources:
      limits:
        nvidia.com/gpu: 2
```



Dedicated GPUs


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





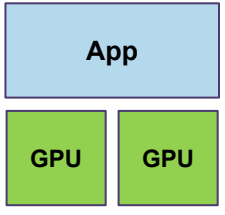
```yaml
apiVersion: resource.k8s.io/v1alpha3
kind: ResourceClaimTemplate
metadata:
  name: unique-gpu
spec:
  spec:
    devices:
      requests:
      - name: gpu
        deviceClassName: gpu.nvidia.com
```

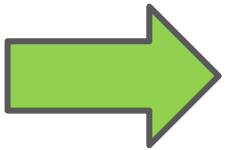
Defines the “template” for creating a ResourceClaim

Associated with the DRA Driver and installed by the cluster admin

Dedicated GPUs


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





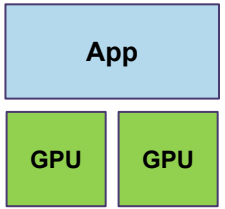
```yaml
apiVersion: resource.k8s.io/v1alpha3
kind: ResourceClaimTemplate
metadata:
  name: unique-gpu
spec:
  spec:
    devices:
      requests:
      - name: gpu
        deviceClassName: gpu.nvidia.com
---
apiVersion: v1
kind: Pod
metadata:
  name: gpu-example
spec:
  containers:
  - name: ctr
    resources:
      claims:
      - name: gpu0
      - name: gpu1
  resourceClaims:
  - name: gpu0
    resourceClaimTemplateName: unique-gpu
  - name: gpu1
    resourceClaimTemplateName: unique-gpu
```

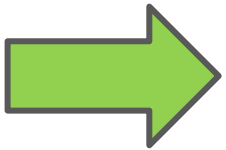
Defines the “template” for creating a ResourceClaim

Associated with the DRA Driver and installed by the cluster admin

Dedicated GPUs


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





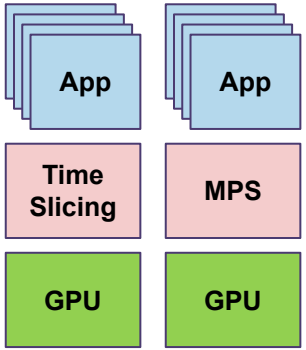
```yaml
apiVersion: resource.k8s.io/v1alpha3
kind: ResourceClaimTemplate
metadata:
  name: unique-gpu
spec:
  spec:
    devices:
      requests:
      - name: gpu
        deviceClassName: gpu.nvidia.com
---
apiVersion: v1
kind: Pod
metadata:
  name: gpu-example
spec:
  containers:
  - name: ctr
    resources:
      claims:
      - name: gpu0
      - name: gpu1
  resourceClaims:
  - name: gpu0
    resourceClaimTemplateName: unique-gpu
  - name: gpu1
    resourceClaimTemplateName: unique-gpu
```

Defines the “template” for creating a ResourceClaim

Associated with the DRA Driver and installed by the cluster admin

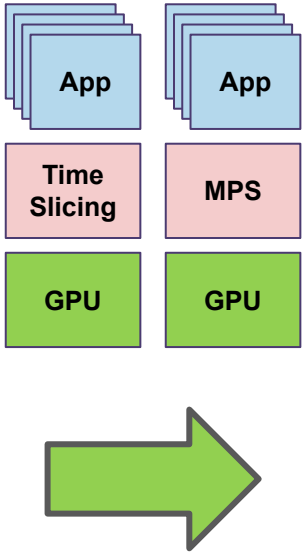
Shared GPUs


```yaml
apiVersion: v1
kind: Pod
metadata:
  name: gpu-example
spec:
  containers:
  - name: ctr0
    image: nvidia/cuda
    resources:
      limits:
        nvidia.com/gpu.shared: 1
  - name: ctr1
    image: nvidia/cuda
    resources:
      limits:
        nvidia.com/gpu.shared: 1
```



Shared GPUs


```yaml
apiVersion: v1
kind: Pod
metadata:
  name: gpu-example
spec:
  containers:
  - name: ctr0
    image: nvidia/cuda
    resources:
      limits:
        nvidia.com/gpu.shared: 1
  - name: ctr1
    image: nvidia/cuda
    resources:
      limits:
        nvidia.com/gpu.shared: 1
```

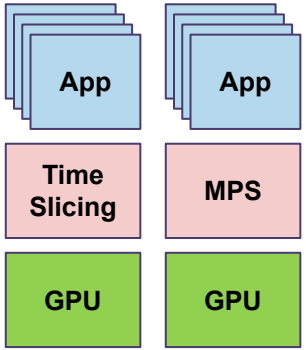



```yaml
apiVersion: resource.k8s.io/v1alpha3
kind: ResourceClaimTemplate
metadata:
  name: unique-gpu
spec:
  spec:
    devices:
      requests:
      - name: gpu
        deviceClassName: gpu.nvidia.com
---
apiVersion: v1
kind: Pod
metadata:
  name: gpu-example
spec:
  containers:
  - name: ctr0
    resources:
      claims:
      - name: gpu
  - name: ctr1
    resources:
      claims:
      - name: gpu
  resourceClaims:
  - name: gpu
    resourceClaimTemplateName: unique-gpu
```

Shared GPUs


```yaml
apiVersion: v1
kind: Pod
metadata:
  name: gpu-example0
spec:
  containers:
  - name: ctr
    image: nvidia/cuda
    resources:
      limits:
        nvidia.com/gpu.shared: 1
```

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: gpu-example1
spec:
  containers:
  - name: ctr
    image: nvidia/cuda
    resources:
      limits:
        nvidia.com/gpu.shared: 1
```




```yaml
apiVersion: resource.k8s.io/v1alpha3
kind: ResourceClaim
metadata:
  name: shared-gpu
spec:
  devices:
    requests:
    - name: gpu
      deviceClassName: gpu.nvidia.com
```

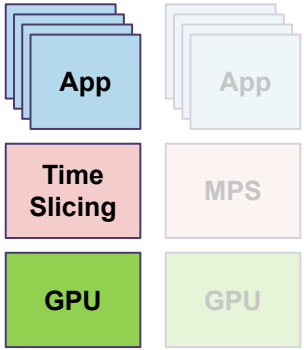
```yaml
---
apiVersion: v1
kind: Pod
metadata:
  name: gpu-example0
spec:
  containers:
  - name: ctr
    resources:
      claims:
      - name: gpu
  resourceClaims:
  - name: gpu
    resourceClaimName: shared-gpu
```

```yaml
---
apiVersion: v1
kind: Pod
metadata:
  name: gpu-example1
spec:
  containers:
  - name: ctr
    resources:
      claims:
      - name: gpu
  resourceClaims:
  - name: gpu
    resourceClaimName: shared-gpu
```

Shared GPUs (Time-Slicing)


```
version: v1
sharing:
  timeSlicing:
    resources:
    - name: nvidia.com/gpu
      replicas: 3
    ...
```

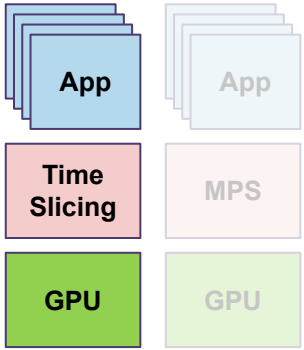
Per node k8s-device-plugin config file

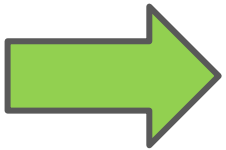


Shared GPUs (Time-Slicing)


```
version: v1
sharing:
  timeSlicing:
    resources:
    - name: nvidia.com/gpu
      replicas: 3
    ...
```

Per node k8s-device-plugin config file



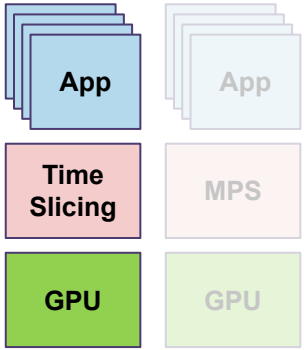


```
apiVersion: resource.k8s.io/v1alpha3
kind: ResourceClaim
metadata:
  name: shared-gpu
spec:
  devices:
    requests:
    - name: gpu
      deviceClassName: gpu.nvidia.com
```

Shared GPUs (Time-Slicing)


```
version: v1
sharing:
  timeSlicing:
    resources:
    - name: nvidia.com/gpu
      replicas: 3
    ...
```

Per node k8s-device-plugin config file



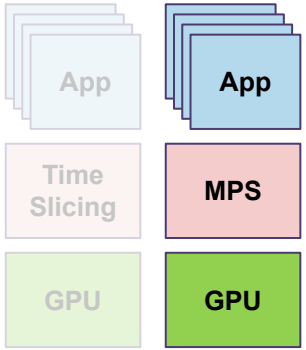

```
apiVersion: resource.k8s.io/v1alpha3
kind: ResourceClaim
metadata:
  name: shared-gpu
spec:
  devices:
    requests:
    - name: gpu
      deviceClassName: gpu.nvidia.com
    config:
    - requests: ["gpu"]
      opaque:
        driver: gpu.nvidia.com
        parameters:
          apiVersion: gpu.nvidia.com/v1alpha1
          kind: GpuConfig
          sharing:
            strategy: TimeSlicing
            timeSlicingConfig:
              interval: <Short | Medium | Long>
```

Configure the time-slice duration

Shared GPUs (MPS)


```
version: v1
sharing:
  mps:
    resources:
    - name: nvidia.com/gpu
      replicas: 3
    ...
```

Per node k8s-device-plugin config file




```
apiVersion: resource.k8s.io/v1alpha3
kind: ResourceClaim
metadata:
  name: shared-gpu
spec:
  devices:
    requests:
    - name: gpu
      deviceClassName: gpu.nvidia.com
    config:
    - requests: ["gpu"]
      opaque:
        driver: gpu.nvidia.com
        parameters:
          apiVersion: gpu.nvidia.com/v1alpha1
          kind: GpuConfig
          sharing:
            strategy: MPS
            mpsConfig:
              defaultActiveThreadPercentage: <fraction>
              defaultPinnedDeviceMemoryLimit: <limit>
```

Configure the default memory / compute limits for each connected client

Shared GPUs (MIG)

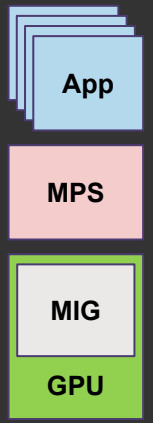

```yaml
apiVersion: resource.k8s.io/v1alpha3
kind: ResourceClaim
metadata:
  name: shared-mig
spec:
  devices:
    requests:
    - name: mig-1g-5gb-0
      deviceClassName: mig.nvidia.com
      selectors:
      - cel:
          expression: "device.attributes['gpu.nvidia.com'].profile == '1g.5gb'"
```

Shared GPUs (MIG + Time-Slicing)


```yaml
apiVersion: resource.k8s.io/v1alpha3
kind: ResourceClaim
metadata:
  name: shared-mig
spec:
  devices:
    requests:
    - name: mig-1g-5gb-0
      deviceClassName: mig.nvidia.com
      selectors:
      - cel:
          expression: "device.attributes['gpu.nvidia.com'].profile == '1g.5gb'"
    config:
    - requests: ["mig-1g-5gb-0"]
      opaque:
        driver: gpu.nvidia.com
        parameters:
          apiVersion: gpu.nvidia.com/v1alpha1
          kind: GpuConfig
          sharing:
            strategy: TimeSlicing
            timeSlicingConfig:
              interval: <Short | Medium | Long>
```

Shared GPUs (MIG + Time-Slicing)


```yaml
apiVersion: resource.k8s.io/v1alpha3
kind: ResourceClaim
metadata:
  name: shared-mig
spec:
  devices:
    requests:
    - name: mig-1g-5gb-0
      deviceClassName: mig.nvidia.com
      selectors:
      - cel:
          expression: "device.attributes['gpu.nvidia.com'].profile == '1g.5gb'"
    config:
    - requests: ["mig-1g-5gb-0"]
      opaque:
        driver: gpu.nvidia.com
        parameters:
          apiVersion: gpu.nvidia.com/v1alpha1
          kind: GpuConfig
          sharing:
            strategy: MPS
            mpsConfig:
              defaultActiveThreadPercentage: <fraction>
              defaultPinnedDeviceMemoryLimit: <limit>
```

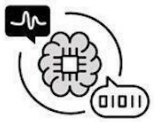


A Benchmark Study

A Benchmark Study

Workloads

Inference Workload

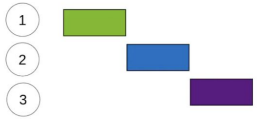


Small GPU Batch Job

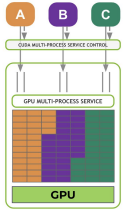
GPU-intensive Jobs

GPU Sharing Strategies

Time-Slicing

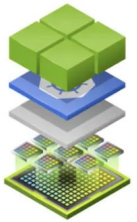


MPS



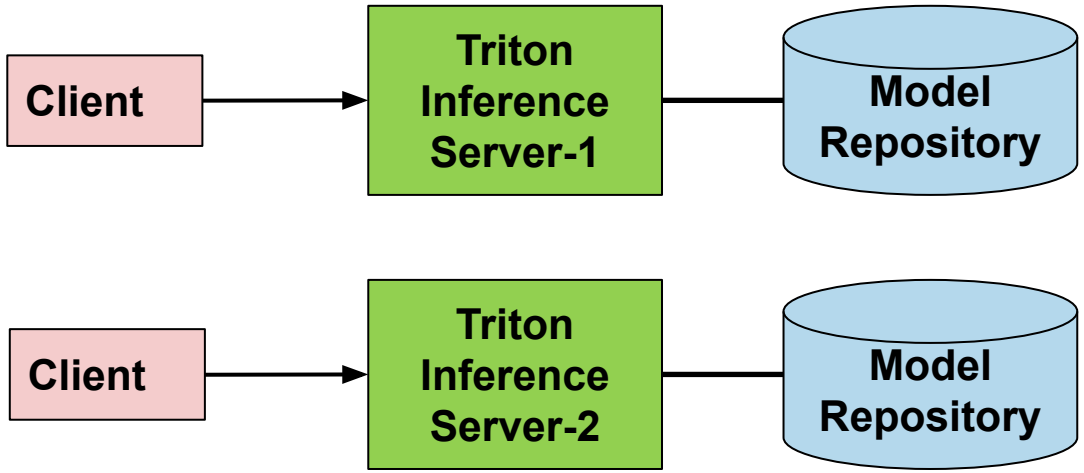
- Without resource limits
- With resource limits

MIG



Inference Workload

Inference Workload: Testbed



Models

resnet50
densenet_onnx
inception_graphdef
simple
simple_dyna_sequence
simple_identity
simple_int8
simple_sequence
simple_string
…

A Linux WorkStation with a single GPU: GeForce RTX 4090

Client (Perf-Analyzer)	Max Threads	Batch Size
Light Load	1	1
Heavy Load	5	5

- Triton inference server
- Performance Analyzer

Sharing A Single GPU: Time-Slicing


```yaml
apiVersion: resource.k8s.io/v1alpha3
kind: ResourceClaim
metadata:
  name: shared-gpu
spec:
  devices:
    requests:
    - name: ts-gpu
      deviceClassName: gpu.nvidia.com
      config:
      - requests: ["ts-gpu"]
        opaque:
          driver: gpu.nvidia.com
          parameters:
            apiVersion: gpu.nvidia.com/v1alpha1
            kind: GpuConfig
            sharing:
              strategy: Time-Slicing
```

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: triton-server-1
  labels:
    app: triton-server-1
spec:
  containers:
    - name: triton-inference-server
      image: nvcr.io/nvidia/tritonserver:23.05-py3
      args: ["tritonserver", "--model-repository=/models"]
      resources:
        claims:
        - name: gpu
  resourceClaims:
    - name: gpu
      resourceClaimName: shared-gpu
```

Sharing A Single GPU: MPS


```yaml
apiVersion: resource.k8s.io/v1alpha3
kind: ResourceClaim
metadata:
  name: shared-gpu
spec:
  devices:
    requests:
    - name: mps-gpu
      deviceClassName: gpu.nvidia.com
      config:
      - requests: ["mps-gpu"]
        opaque:
          driver: gpu.nvidia.com
          parameters:
            apiVersion: gpu.nvidia.com/v1alpha1
            kind: GpuConfig
            sharing:
              strategy: MPS
```

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: triton-server-1
  labels:
    app: triton-server-1
spec:
  containers:
    - name: triton-inference-server
      image: nvcr.io/nvidia/tritonserver:23.05-py3
      args: ["tritonserver", "--model-repository=/models"]
      resources:
        claims:
        - name: gpu
  resourceClaims:
    - name: gpu
      resourceClaimName: shared-gpu
```

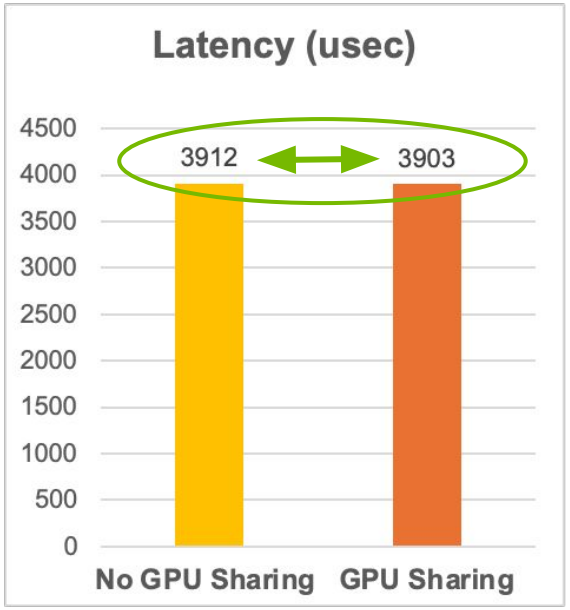
Sharing A Single GPU: MPS + Limits

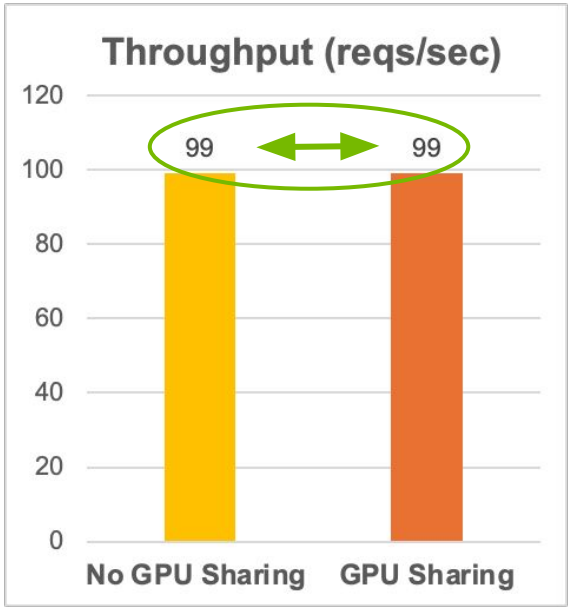

```yaml
apiVersion: resource.k8s.io/v1alpha3
kind: ResourceClaim
metadata:
  name: shared-gpu
spec:
  devices:
    requests:
    - name: mps-gpu
      deviceClassName: gpu.nvidia.com
      config:
      - requests: ["mps-gpu"]
        opaque:
          driver: gpu.nvidia.com
          parameters:
            apiVersion: gpu.nvidia.com/v1alpha1
            kind: GpuConfig
            sharing:
              strategy: MPS
```

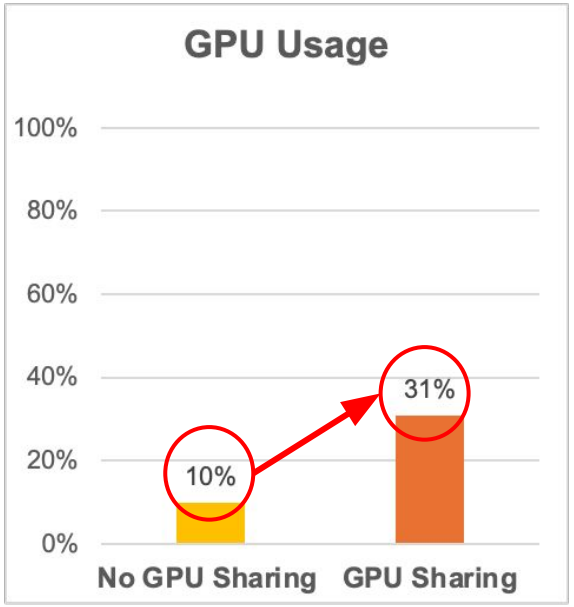
```yaml
apiVersion: v1
kind: Pod
metadata:
  name: triton-server-1
  labels:
    app: triton-server-1
spec:
  containers:
    - name: triton-inference-server
      image: nvcr.io/nvidia/tritonserver:23.05-py3
      args: ["tritonserver", "--model-repository=/models"]
      resources:
        claims:
        - name: gpu
      env:
      - name: CUDA_MPS_PINNED_DEVICE_MEM_LIMIT
        value: "0=500M"
      - name: CUDA_MPS_ACTIVE_THREAD_PERCENTAGE
        value: "5"
  resourceClaims:
    - name: gpu
      resourceClaimName: shared-gpu
```

Light Inference Workload

Time-Slicing







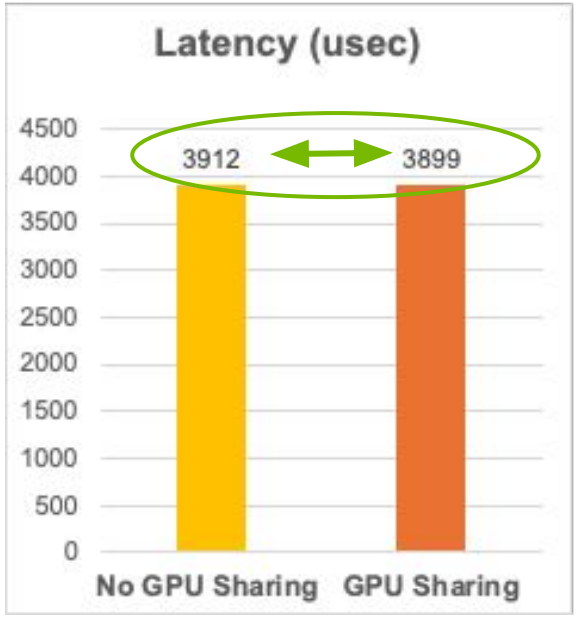
- Improved GPU utilization
- No performance degradation

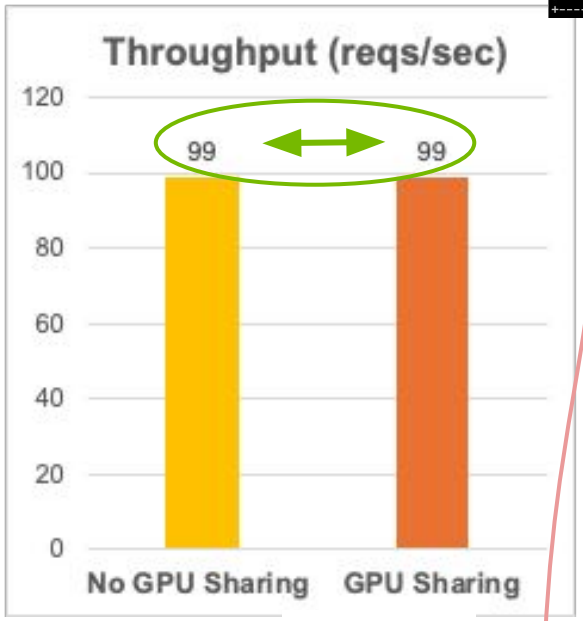
GPU Usage:

- Optimal: 2x increase
- Observed: 3x increase

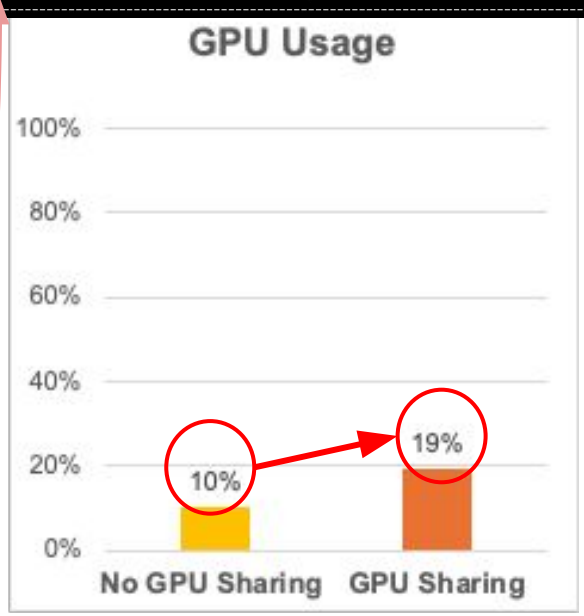
Light Inference Workload

MPS






```
+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|    0   N/A  N/A    882669    M+C   tritonserver                       642MiB |
|    0   N/A  N/A    882677    M+C   tritonserver                       642MiB |
|    0   N/A  N/A    882815      C   nvidia-cuda-mps-server              28MiB |
+-----------------------------------------------------------------------------+
```

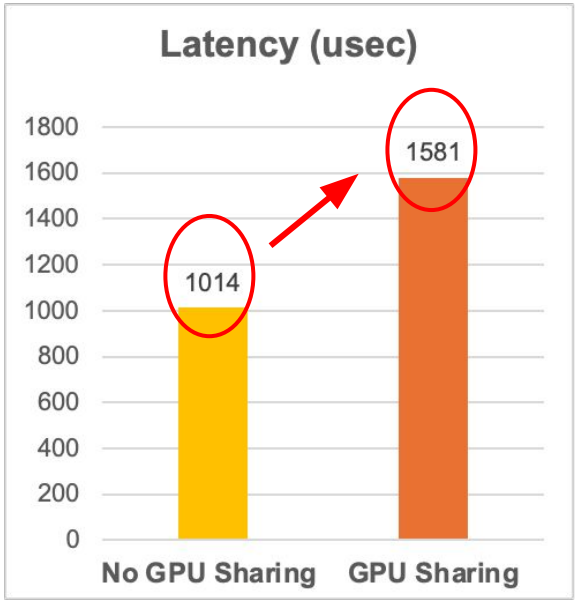


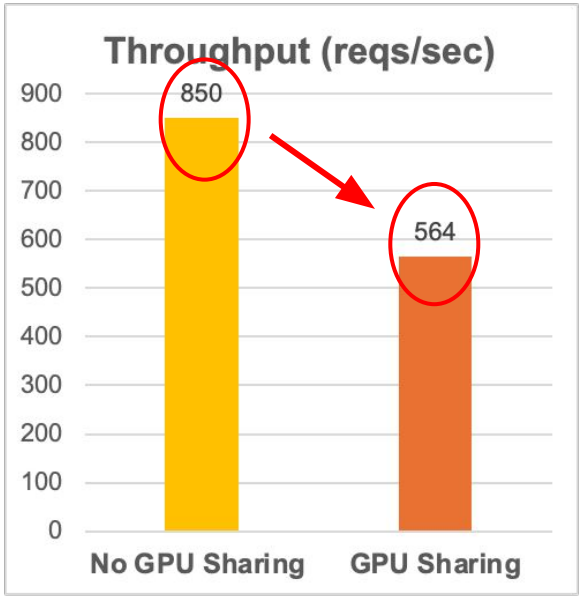
- Improved GPU utilization
- No performance degradation
- Low overhead

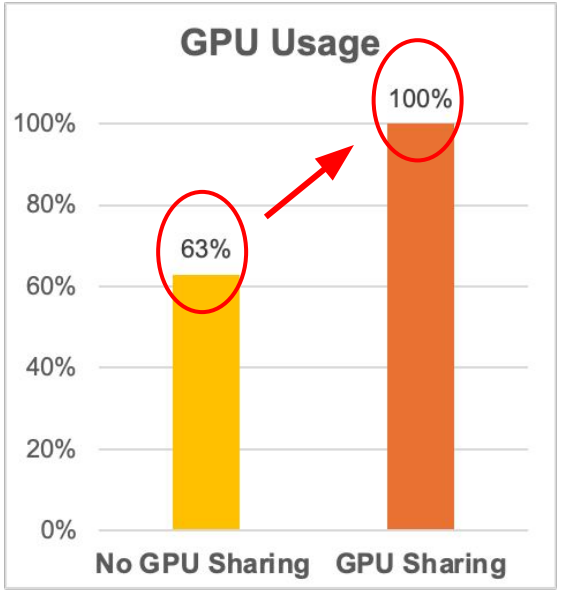
- Limited fault isolation
- Slower startup - MPS: 6s vs TS: 1s
- Additional MPS daemon

Heavy Inference Workload

Time-Slicing



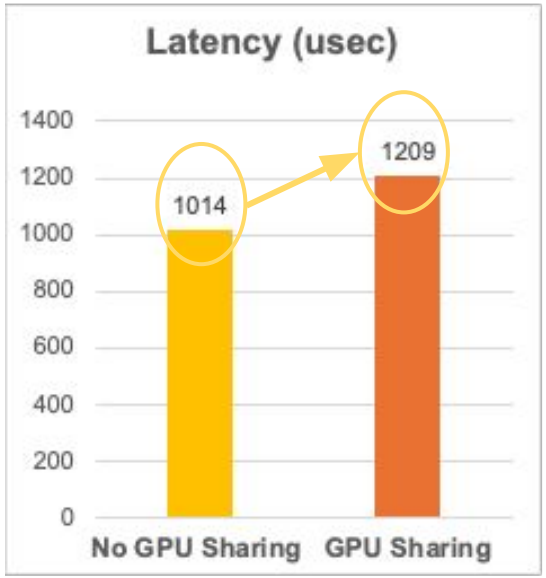


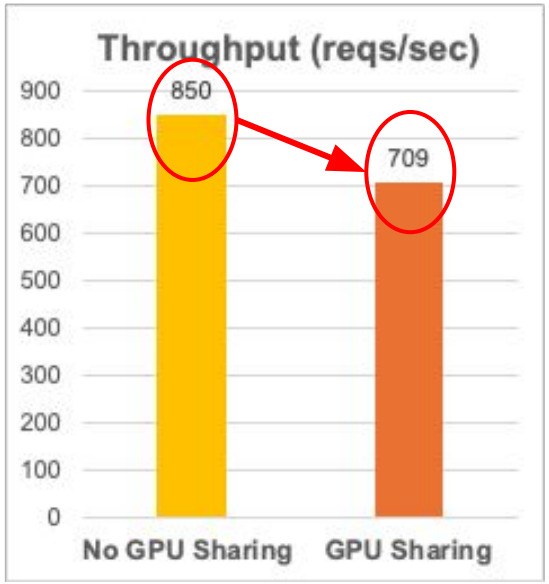


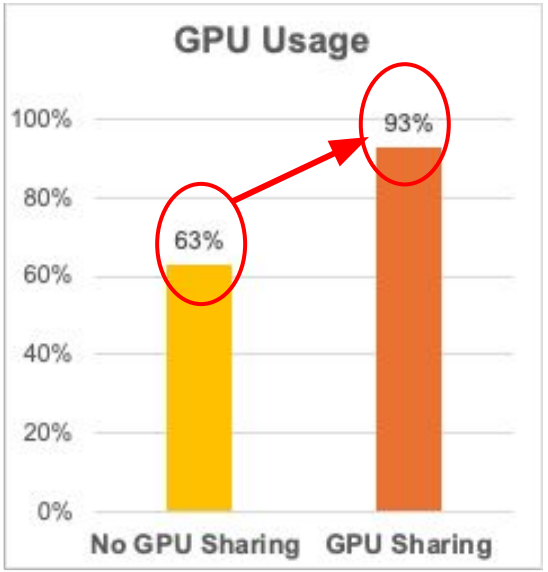
- Performance degradation: higher latency, lower throughput
- Performance interference: cannot limit GPU usage (maxed out at 100%)

Heavy Inference Workload

MPS without Resource Limits





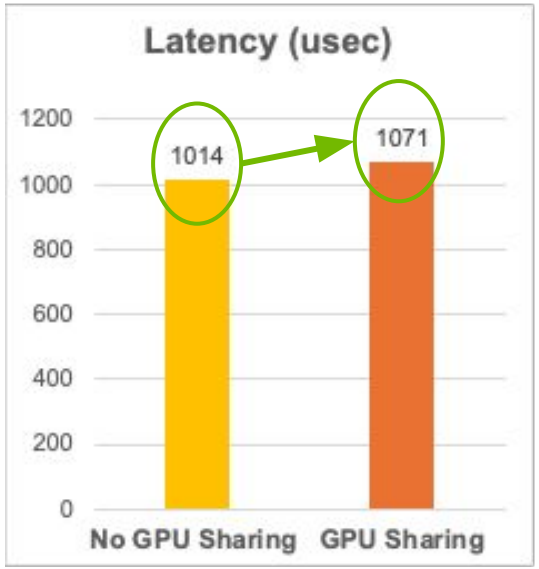


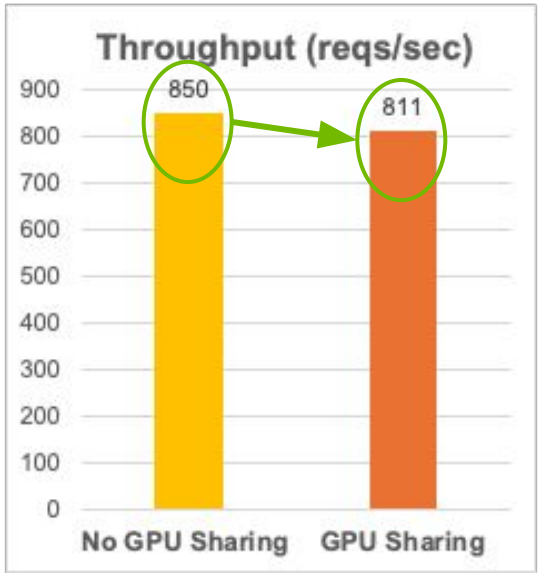
- Moderate performance degradation (better than Time-Slicing)

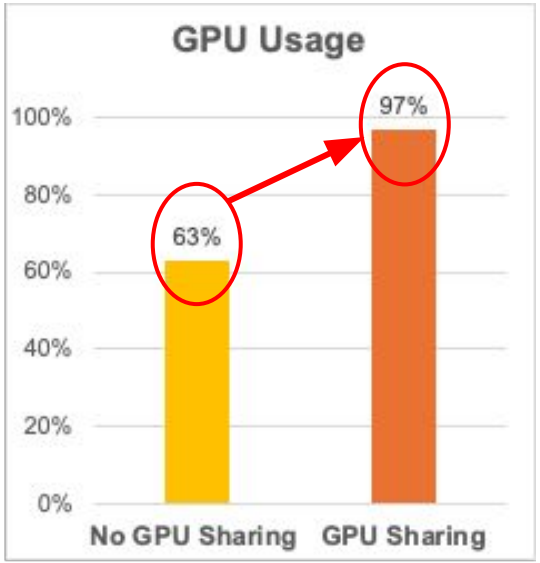
- No performance isolation

Heavy Inference Workload

MPS with Resource Limits







- Slight performance degradation
- Better performance isolation

- Require proper resource limits

Summary of GPU Sharing for Inference Workload

- Light inference workload
  - Time-Slicing and MPS can improve utilization without hurting performance
  - MPS has a lower overhead
- Heavy inference workload
  - Use Time-Slicing if applications are not latency sensitive, e.g., offline inference
  - MPS performs better with moderate performance degradation
  - MPS + Limits can provide better performance isolation

Time-Slicing

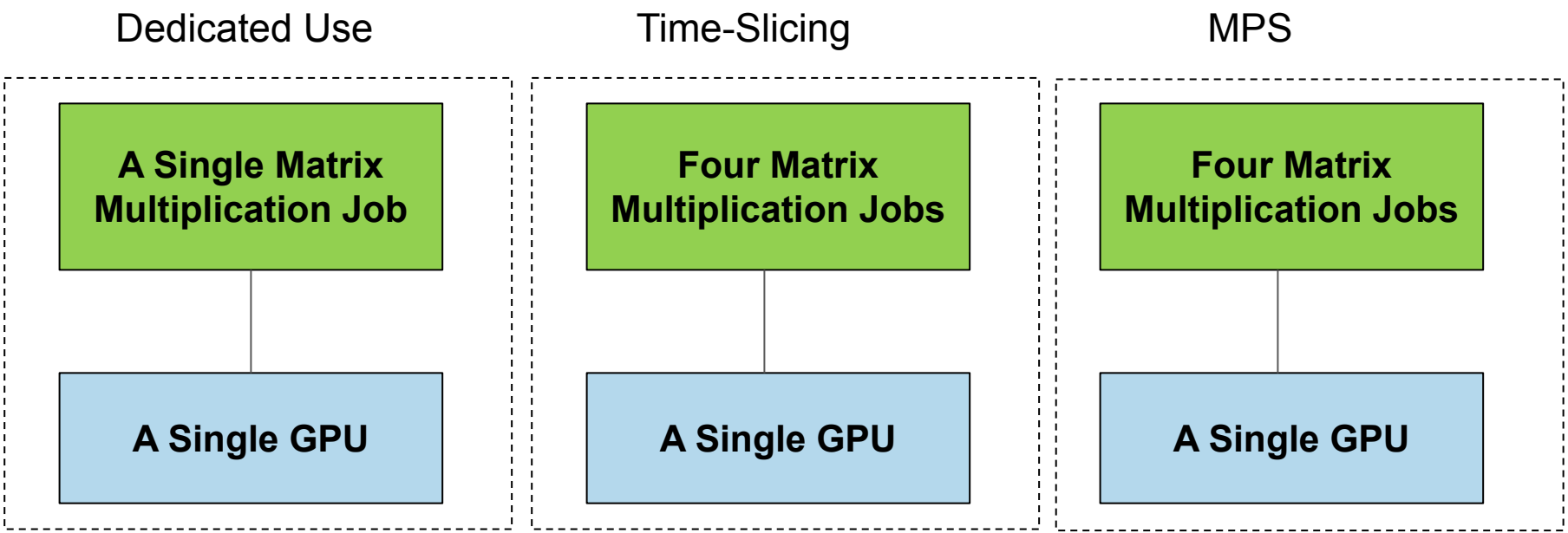
- Context switch overhead
- No resource limit

MPS

- Limited memory fault isolation
- Longer startup time

Small GPU Batch Job

Small GPU Batch Job: Setup



Sharing A Single GPU: Time-Slicing


```yaml
apiVersion: resource.k8s.io/v1alpha3
kind: ResourceClaim
metadata:
  name: shared-gpu
spec:
  devices:
    requests:
    - name: ts-gpu
      deviceClassName: gpu.nvidia.com
      config:
      - requests: ["ts-gpu"]
        opaque:
          driver: gpu.nvidia.com
          parameters:
            apiVersion: gpu.nvidia.com/v1alpha1
            kind: GpuConfig
            sharing:
              strategy: Time-Slicing
```

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: job-1
spec:
  containers:
  - name: matrix-multiplication
    image: cupy/cupy:latest
    command: ["bash", "-c"]
    args:
      - |
        time python3 -c '
        import cupy as cp;
        for i in range(100000):
          a = cp.random.rand(256, 256);
          b = cp.random.rand(256, 256);
          c = cp.dot(a, b);
          print(c)
        '
    resources:
      claims:
      - name: gpu
        request: shared-gpu
```

Sharing A Single GPU: MPS


```yaml
apiVersion: resource.k8s.io/v1alpha3
kind: ResourceClaim
metadata:
  name: shared-gpu
spec:
  devices:
    requests:
    - name: ts-gpu
      deviceClassName: gpu.nvidia.com
      config:
      - requests: ["mps-gpu"]
        opaque:
          driver: gpu.nvidia.com
          parameters:
            apiVersion: gpu.nvidia.com/v1alpha1
            kind: GpuConfig
            sharing:
              strategy: MPS
```

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: job-1
spec:
  containers:
  - name: matrix-multiplication
    image: cupy/cupy:latest
    command: ["bash", "-c"]
    args:
      - |
        time python3 -c '
        import cupy as cp;
        for i in range(100000):
          a = cp.random.rand(256, 256);
          b = cp.random.rand(256, 256);
          c = cp.dot(a, b);
          print(c)
        '
    resources:
      claims:
      - name: gpu
        request: shared-gpu
```

Small GPU Batch Job

Time-Slicing

A single job

GPU usage: **35%**

Execution time:
real 0m26.752s
user 0m31.869s
sys 0m0.380s

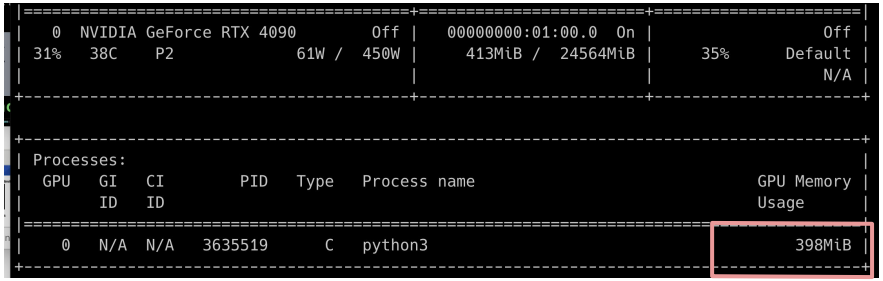
Improved resource utilization

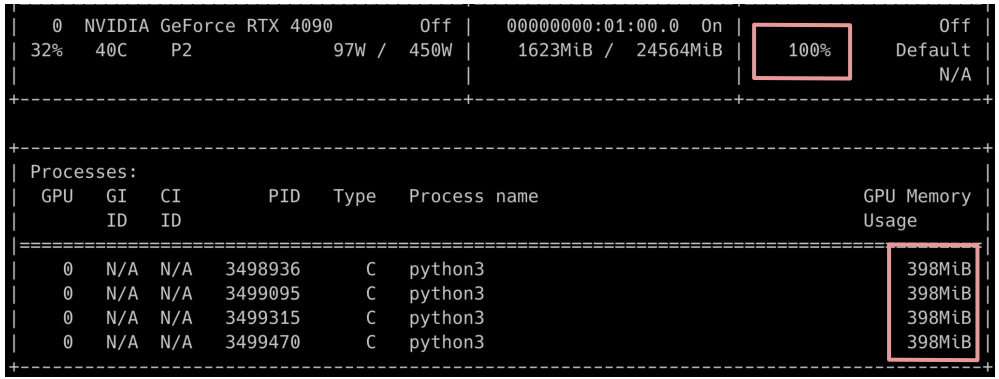
Increased completion time

4 parallel jobs

GPU usage: **100%**

Execution time:
real 0m49.525s
user 0m54.489s
sys 0m0.224s





Small GPU Batch Job

MPS

A single job

GPU usage: **35%**

Execution time:
real 0m26.752s
user 0m31.869s
sys 0m0.380s

Improved resource utilization

Similar performance
Lower overhead

4 parallel jobs

GPU usage: **91%**

Execution time:
real 0m27.213s
user 0m32.101s
sys 0m0.221s

```
|   0  NVIDIA GeForce RTX 4090        Off |   00000000:01:00.0  On |                  Off |
| 31%   38C    P2             61W /  450W |     413MiB /  24564MiB |     35%      Default |
|                                         |                        |                  N/A |
+-----------------------------------------+------------------------+----------------------+

+---------------------------------------------------------------------------------------+
| Processes:                                                                            |
|  GPU   GI   CI        PID   Type   Process name                            GPU Memory |
|        ID   ID                                                             Usage      |
|=======================================================================================|
|    0   N/A  N/A   3635519      C   python3                                     398MiB |
+---------------------------------------------------------------------------------------+
```

```
|   0  NVIDIA GeForce RTX 4090        Off |   00000000:01:00.0  On |                  Off |
| 31%   43C    P2             101W /  450W |    1629MiB /  24564MiB |     94%   E. Process |
|                                         |                        |                  N/A |
+-----------------------------------------+------------------------+----------------------+

+---------------------------------------------------------------------------------------+
| Processes:                                                                            |
|  GPU   GI   CI        PID   Type   Process name                            GPU Memory |
|        ID   ID                                                             Usage      |
|=======================================================================================|
|    0   N/A  N/A   3644684    M+C   python3                                     396MiB |
|    0   N/A  N/A   3644749      C   nvidia-cuda-mps-server                       28MiB |
|    0   N/A  N/A   3644854    M+C   python3                                     396MiB |
|    0   N/A  N/A   3645023    M+C   python3                                     396MiB |
|    0   N/A  N/A   3645191    M+C   python3                                     396MiB |
+---------------------------------------------------------------------------------------+
```

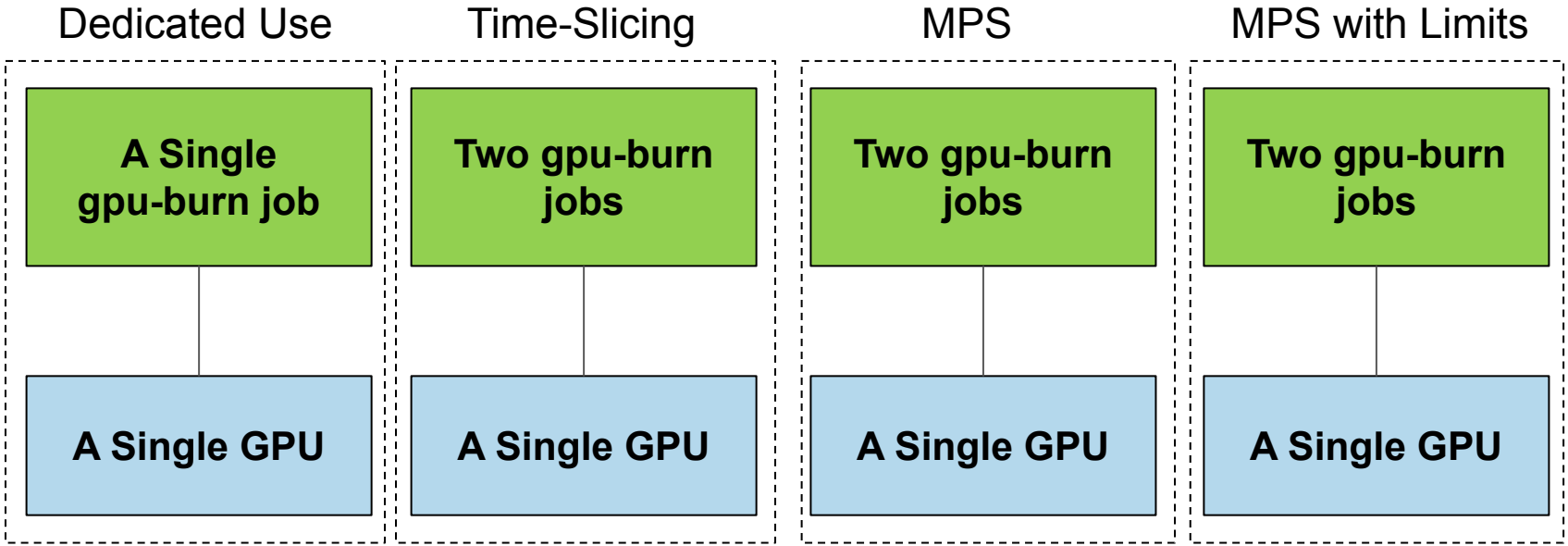
Summary of GPU Sharing for Small GPU Batch Job

- **Time-Slicing**
  - Improve GPU utilization
  - Suboptimal because of higher overhead due to context switch

- **MPS**
  - Flexible partition size
  - Better performance
  - Lower overhead compared to Time-Slicing

GPU-Intensive Batch Job

GPU-intensive Batch Job: Setup



gpu-burn – https://github.com/wilicc/gpu-burn

Sharing A Single GPU: Time-Slicing


```yaml
apiVersion: resource.k8s.io/v1alpha3
kind: ResourceClaim
metadata:
  name: shared-gpu
spec:
  devices:
    requests:
    - name: ts-gpu
      deviceClassName: gpu.nvidia.com
      config:
      - requests: ["ts-gpu"]
        opaque:
          driver: gpu.nvidia.com
          parameters:
            apiVersion: gpu.nvidia.com/v1alpha1
            kind: GpuConfig
            sharing:
              strategy: Time-Slicing
```

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: job-1
spec:
  containers:
  - name: gpu-burn
    image: coguzpastirmaci/gpu-burn
    args: ["-d", "60"]
    resources:
      claims:
      - name: gpu
        request: shared-gpu
```

Sharing A Single GPU: MPS


```yaml
apiVersion: resource.k8s.io/v1alpha3
kind: ResourceClaim
metadata:
  name: shared-gpu
spec:
  devices:
    requests:
    - name: mps-gpu
      deviceClassName: gpu.nvidia.com
      config:
      - requests: ["mps-gpu"]
        opaque:
          driver: gpu.nvidia.com
          parameters:
            apiVersion: gpu.nvidia.com/v1alpha1
            kind: GpuConfig
            sharing:
              strategy: MPS
```

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: job-1
spec:
  containers:
  - name: gpu-burn
    image: coguzpastirmaci/gpu-burn
    args: ["-d", "60"]
    resources:
      claims:
      - name: gpu
        request: shared-gpu
```

Sharing A Single GPU: MPS with Resource Limits


```yaml
apiVersion: resource.k8s.io/v1alpha3
kind: ResourceClaim
metadata:
  name: shared-gpu
spec:
  devices:
    requests:
    - name: ts-gpu
      deviceClassName: gpu.nvidia.com
      config:
      - requests: ["mps-gpu"]
        opaque:
          driver: gpu.nvidia.com
          parameters:
            apiVersion: gpu.nvidia.com/v1alpha1
            kind: GpuConfig
            sharing:
              strategy: MPS
                  mpsConfig:
                    defaultActiveThreadPercentage: 50
                    defaultPinnedDeviceMemoryLimit: 5Gi
```

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: job-1
spec:
  containers:
  - name: gpu-burn
    image: coguzpastirmaci/gpu-burn
    args: ["-d", "60"]
    resources:
      claims:
      - name: gpu
        request: shared-gpu
```

GPU-Intensive Batch Job

Time-Slicing

A single job

- GPU usage: 100%
- Memory usage
  - Job1: 23,702MiB

Two parallel jobs

- GPU usage:100%
- Memory usage
  - Job1: 21,702MiB
  - Job2: 2,182MiB

```
+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|    0   N/A  N/A   1588344      C   ./gpu_burn                      21702MiB |
+-----------------------------------------------------------------------------+
```

```
+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|    0   N/A  N/A   1602484      C   ./gpu_burn                      21702MiB |
|    0   N/A  N/A   1602648      C   ./gpu_burn                       2182MiB |
+-----------------------------------------------------------------------------+
```

- Cannot cap resource usage
- No guarantee for fair sharing

GPU-Intensive Batch Job

MPS without Resource Limits

A single job

- GPU usage: 100%
- Memory usage
  - Job1: 23,702MiB

Two parallel jobs

- GPU usage:100%
- Memory usage
  - Job1: 21,534MiB
  - Job2: 2,270MiB

```
+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|    0   N/A  N/A   1588344      C   ./gpu_burn                      21702MiB |
+-----------------------------------------------------------------------------+
```

```
+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|    0   N/A  N/A   1623188    M+C   ./gpu_burn                      21534MiB |
|    0   N/A  N/A   1623190      C   nvidia-cuda-mps-server             28MiB |
|    0   N/A  N/A   1623280    M+C   ./gpu_burn                       2270MiB |
+-----------------------------------------------------------------------------+
```

- No guarantee for fair sharing
- Additional MPS daemon process

GPU-Intensive Batch Job

MPS with Resource Limits

A single job

- GPU usage: 100%
- Memory usage
  - Job1: 23,702MiB

Two parallel jobs

- GPU usage:100%
- Memory usage
  - Job1: 4,606MiB
  - Job2: 4,606MiB

```
Processes:
  GPU   GI   CI        PID   Type   Process name                            GPU Memory
        ID   ID                                                           Usage
=============================================================================
    0   N/A  N/A   1588344      C   ./gpu_burn                                21702MiB
```

```
Processes:
  GPU   GI   CI        PID   Type   Process name                            GPU Memory
        ID   ID                                                           Usage
=============================================================================
    0   N/A  N/A   1638141    M+C   ./gpu_burn                                4606MiB
    0   N/A  N/A   1638143      C   nvidia-cuda-mps-server                      28MiB
    0   N/A  N/A   1638231    M+C   ./gpu_burn                                4606MiB
```

- Cap resource resource
- Resource fair sharing and isolation

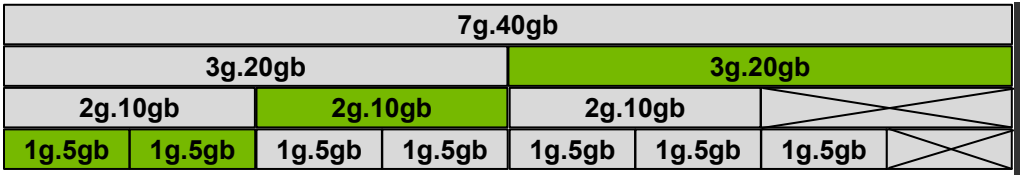
Summary of GPU Sharing for GPU-Intensive Batch Job

- **Time-Slicing** is not an ideal option for GPU-intensive jobs.
- **MPS** with a proper resource limit should be used instead.

MIG

MIG Partition

4 MIG instances on NVIDIA A100 (40GB)



MIG instances:

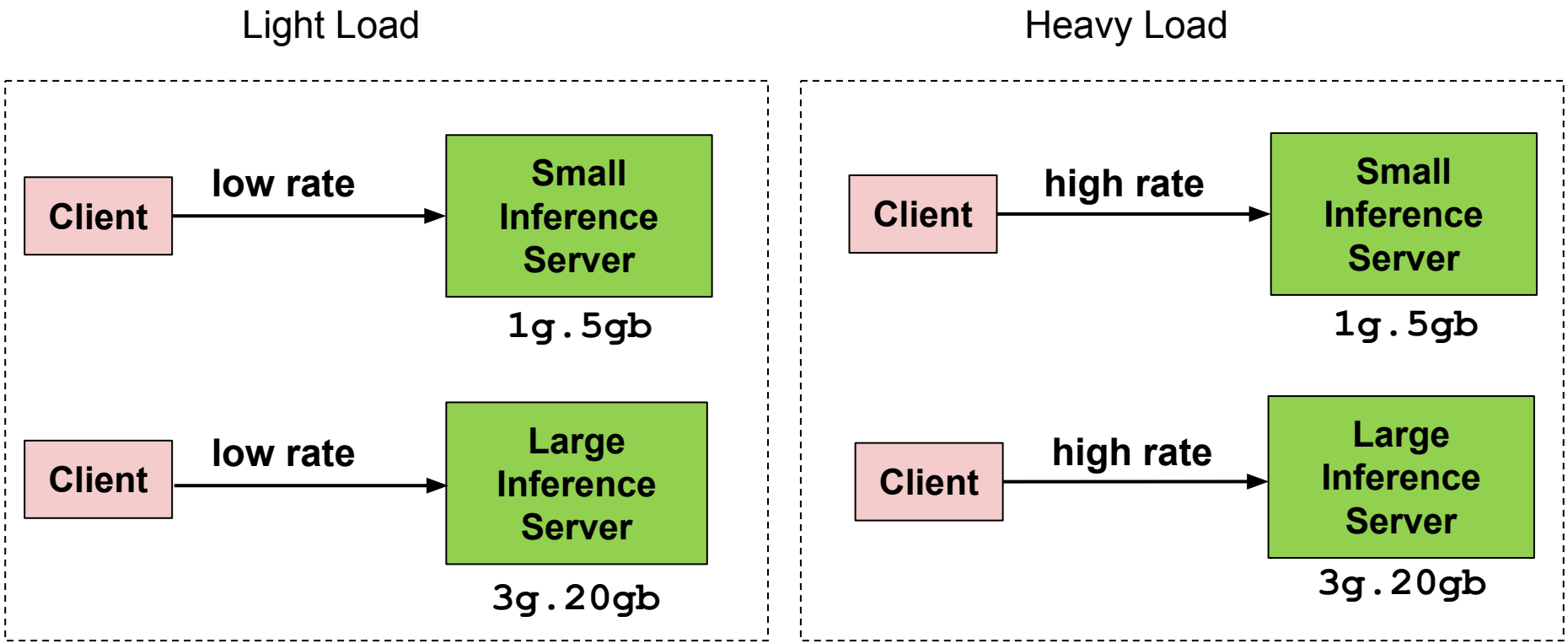
- 2 x 1g.5gb
- 1 x 2g.10gb
- 1 x 3g.20gb

```
apiVersion: v1
mig-configs:
  half-balanced:
    - devices: [0, 1, 2, 3]
      mig-enabled: true
      mig-devices:
        "1g.5gb": 2
        "2g.10gb": 1
        "3g.20gb": 1

    - devices: [4, 5, 6, 7]
      mig-enabled: false
```

```
apiVersion: resource.k8s.io/v1alpha3
kind: ResourceClaimTemplate
metadata:
  name: mig-devices
spec:
  spec:
    devices:
      requests:
      - name: mig-1g-5gb-0
        deviceClassName: mig.nvidia.com
        selectors:
        - cel:
            expression: "device.attributes['gpu.nvidia.com'].profile == '1g.5gb'"
      - name: mig-1g-5gb-1
        deviceClassName: mig.nvidia.com
        selectors:
        - cel:
            expression: "device.attributes['gpu.nvidia.com'].profile == '1g.5gb'"
      - name: mig-2g-10gb
        deviceClassName: mig.nvidia.com
        selectors:
        - cel:
            expression: "device.attributes['gpu.nvidia.com'].profile == '2g.10gb'"
      - name: mig-3g-20gb
        deviceClassName: mig.nvidia.com
        selectors:
        - cel:
            expression: "device.attributes['gpu.nvidia.com'].profile == '3g.20gb'"
      constraints:
      - requests: []
        matchAttribute: "gpu.nvidia.com/parentUUID"
```

Inference Workload: Setup



Inference Workloads on Different MIG Instances

Small Instance: 1g.5gb

```yaml
apiVersion: resource.k8s.io/v1alpha3
kind: ResourceClaim
metadata:
  name: mig-devices
spec:
  devices:
    requests:
    - name: mig-1g-5gb
      deviceClassName: migu.nvidia.com
      selectors:
      - cel:
          expression: "device.attributes['gpu.nvidia.com'].profile == '1g.5gb'"
- name: mig-3g-20gb
      deviceClassName: migu.nvidia.com
      selectors:
      - cel:
          expression: "device.attributes['gpu.nvidia.com'].profile == '3g.20gb'"

      constraints:
      - requests: []
        matchAttribute: "gpu.nvidia.com/parentUUID"
```

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: large-server
  labels:
    app: large-server
spec:
  containers:
    - name: triton-inference-server
      image: nvcr.io/nvidia/tritonserver:23.05-py3
      args: ["tritonserver", "--model-repository=/models"]
      resources:
        claims:
        - name: gpu
          request: mig-1g-5gb
  resourceClaims:
    - name: gpu
      resourceClaimName: mig-devices
```

GPU-intensive Jobs on Different MIG Instances

Large Instance: 3g.20gb

```yaml
apiVersion: resource.k8s.io/v1alpha3
kind: ResourceClaim
metadata:
  name: mig-devices
spec:
  devices:
    requests:
    - name: mig-1g-5gb
      deviceClassName: migu.nvidia.com
      selectors:
      - cel:
          expression: "device.attributes['gpu.nvidia.com'].profile == '1g.5gb'"
- name: mig-3g-20gb
      deviceClassName: migu.nvidia.com
      selectors:
      - cel:
          expression: "device.attributes['gpu.nvidia.com'].profile == '3g.20gb'"

      constraints:
      - requests: []
        matchAttribute: "gpu.nvidia.com/parentUUID"
```

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: large-server
  labels:
    app: large-server
spec:
  containers:
    - name: triton-inference-server
      image: nvcr.io/nvidia/tritonserver:23.05-py3
      args: ["tritonserver", "--model-repository=/models"]
      resources:
        claims:
        - name: gpu
          request: mig-3g-20gb
  resourceClaims:
    - name: gpu
      resourceClaimName: mig-devices
```

Inference Servers on Different MIG Instances

Light Load

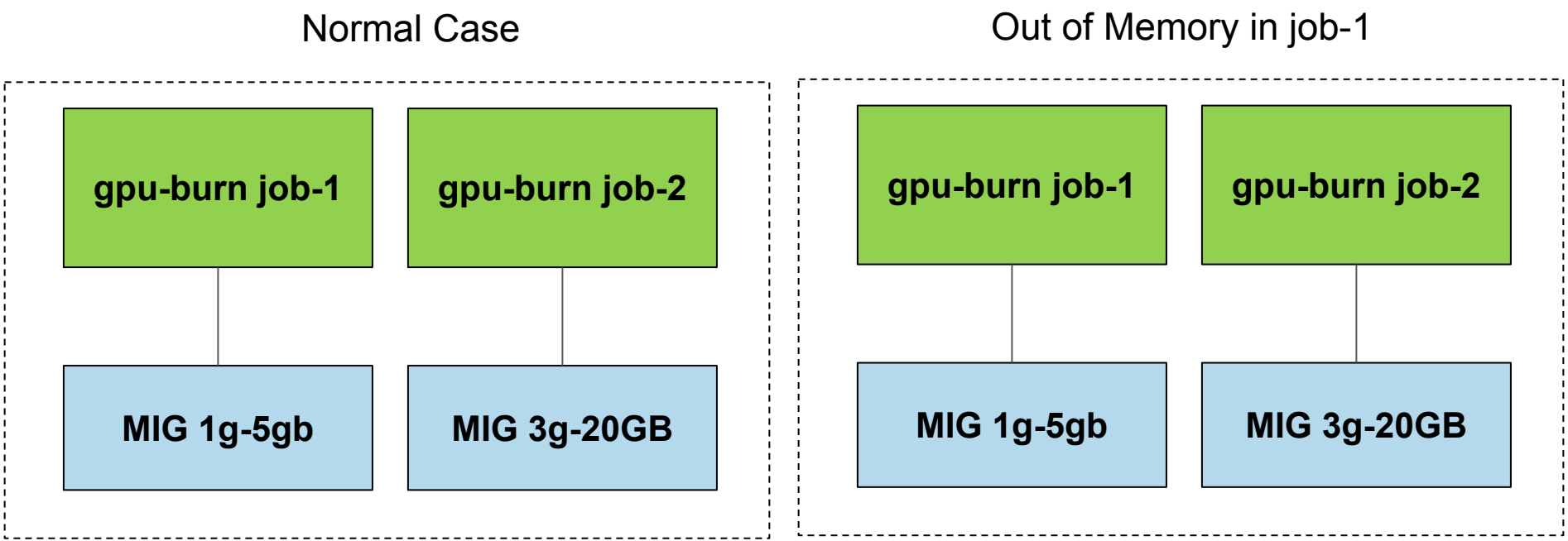
	Latency (usec)	Throughput (reqs/sec)
Small Instance	4,151	99
Large Instance	5,041	755

Performance Isolation

Heavy Load

	Latency (usec)	Throughput (reqs/sec)
Small Instance	14,360	311
Large Instance	5,218	1,718

GPU-intensive Batch Job on MIG: Setup



A100 with 4 MIG Instances

GPU-Intensive Batch Jobs on MIG Instances

Small Instance: 1g.5gb

```yaml
apiVersion: resource.k8s.io/v1alpha3
kind: ResourceClaim
metadata:
  name: mig-devices
spec:
  devices:
    requests:
    - name: mig-1g-5gb
      deviceClassName: migu.nvidia.com
      selectors:
      - cel:
          expression: "device.attributes['gpu.nvidia.com'].profile == '1g.5gb'"
- name: mig-3g-20gb
      deviceClassName: migu.nvidia.com
      selectors:
      - cel:
          expression: "device.attributes['gpu.nvidia.com'].profile == '3g.20gb'"

      constraints:
      - requests: []
        matchAttribute: "gpu.nvidia.com/parentUUID"
```

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: small-server
  labels:
    app: small-server
spec:
  containers:
    - name: gpu-burn
      image: oguzpastirmaci/gpu-burn
      args: ["-d", "60"]
      resources:
        claims:
        - name: gpu
          request: mig-1g-5gb
  resourceClaims:
    - name: gpu
      resourceClaimName: mig-devices
```

GPU-Intensive Batch Jobs on MIG Instances

Large Instance: 3g.20gb

```yaml
apiVersion: resource.k8s.io/v1alpha3
kind: ResourceClaim
metadata:
  name: mig-devices
spec:
  devices:
    requests:
    - name: mig-1g-5gb
      deviceClassName: migu.nvidia.com
      selectors:
      - cel:
          expression: "device.attributes['gpu.nvidia.com'].profile == '1g.5gb'"
- name: mig-3g-20gb
      deviceClassName: migu.nvidia.com
      selectors:
      - cel:
          expression: "device.attributes['gpu.nvidia.com'].profile == '3g.20gb'"

      constraints:
      - requests: []
        matchAttribute: "gpu.nvidia.com/parentUUID"
```

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: large-server
  labels:
    app: larger-server
spec:
  containers:
    - name: gpu-burn
      image: oguzpastirmaci/gpu-burn
      args: ["-d", "60"]
      resources:
        claims:
        - name: gpu
          request: mig-3g-20gb
  resourceClaims:
    - name: gpu
      resourceClaimName: mig-devices
```

GPU-intensive Batch Jobs on MIG Instances

Resource isolation

```
+-----------------------------------------------------------------------------------------+
| Processes:                                                                              |
|  GPU   GI   CI        PID   Type   Process name                              GPU Memory |
|        ID   ID                                                               Usage      |
|=========================================================================================|
|    0   10    0     205309      C   ./gpu_burn                                  4368MiB |
|    2    2    0     205249      C   ./gpu_burn                                 18068MiB |
+-----------------------------------------------------------------------------------------+
```

small instance

large instance

Fault isolation

```
Error in "C alloc": CUDA_ERROR_OUT_OF_MEMORY
```

small instance

```
| Processes:
|  GPU   GI   CI        PID   Type   Process name                              GPU Memory
|        ID   ID                                                               Usage
|=========================================================================================
|    0    2    0     245113      C   ./gpu_burn                                 18068MiB
+-----------------------------------------------------------------------------------------
```

large instance

GPU Partitioning and Sharing using MIG

- Enhanced QoS and fault isolation
- Ideal for multi-tenant environments

- Fixed resource partition size
- Max number of instances = 7

Proper sizing is important!

Summary of the Benchmark Study

Which GPU Sharing Strategy is Right?

	Time-Slicing	MPS w/o Limits	MPS with Limits	MIG
Inference Workload — Light				
Inference Workload — Heavy				
Small GPU Batch Job				
GPU-Intensive Batch Job				

A High Level Comparison of GPU Sharing Strategies

- **Performance isolation**: MIG > MPS > Time-Slicing
- **Fault isolation**: MIG > Time-Slicing > MPS
- **Resource efficiency**: MPS > Time-Slicing
- **Partitioning flexibility**: MPS > MIG
- **Easy to configure**: Time-Slicing > MPS > MIG

Conclusion

Takeaways

GPU sharing can improve resource utilization and reduce costs.

Different GPU sharing strategies, each with its own tradeoffs, are available.

Dynamic Resource Allocation (DRA)

- Enable fine-grained and flexible management of heterogeneous GPUs in a unified and configurable manner
- Significantly simplify GPU resource allocation and sharing
- Beta version available in Kubernetes 1.32+

Takeaways (cont'd)

Time-Slicing

- Suitable for lightly loaded inference servers and less GPU-intensive jobs
- **Easy to configure**
- **Lack of resource guarantees**
- **Context switch overhead**
- Not ideal for heavily loaded interactive workloads; better suited for development environments

MPS (Multi-Process Service)

- **Flexible resource partitioning with application-specific resource limits**
- **Lower overhead**
- Slower startup compared to Time-Slicing
- **Vulnerable to memory-related faults**

MIG (Multi-Instance GPU)

- **Enhanced QoS, fault isolation to accommodate different workload requirements.**
- **Strong isolation; ideal for multi-tenant use cases.**
- **Fixed partition size;** right sizing is critical

References

KubeCon Talks

- Efficient Access to Shared GPU Resources: Mechanisms and Use Cases - Diogo Filipe Tomas Guerra & Diana Gaponcic, CERN
- Keynote: Accelerating AI Workloads with GPUs in Kubernetes - Kevin Klues, Distinguished Engineer & Sanjay Chatterjee, Engineering Manager, NVIDIA

NVIDIA Documents

- GPU sharing: https://developer.nvidia.com/blog/improving-gpu-utilization-in-kubernetes/
- Time-Slicing: https://docs.nvidia.com/datacenter/cloud-native/gpu-operator/latest/gpu-sharing.html
- MPS
  - https://docs.nvidia.com/deploy/mps/index.html#topic_3_3_5_2
  - https://docs.nvidia.com/deploy/pdf/CUDA_Multi_Process_Service_Overview.pdf
- MIG: https://docs.nvidia.com/datacenter/tesla/mig-user-guide/

Workload and Benchmarks

- Triton inference server: https://github.com/triton-inference-server/server
- gpu-burn: https://github.com/wilicc/gpu-burn

Questions & Answers

Give us feedback!

Come find us at our booth!

NVIDIA & Community Talks

KubeCon CloudNativeCon North America 2024

ArgoCon: Building a Cutting-Edge Kubernetes Internal Developer Platform at NVIDIA | **Tuesday** 10:40am - 11:05am

Lightning Talk: Evaluating Scheduler Efficiency for AI/ML Jobs Using Custom Resource Metrics | **Tuesday** 5:45pm - 5:50pm

Lightning Talk: Running Kind Clusters with GPU Support Using Nvkind| **Tuesday** 6:05pm - 6:10pm

Keynote: NVIDIA Case Study: The Many Facets of Building + Delivering AI in the Cloud Native Ecosystem | **Wednesday** 10:05am - 10:20am

All-Your-GPUs-Are-Belong-to-Us: An Inside Look at NVIDIA's Self-Healing GeForce NOW Infrastructure | **Wednesday** 11:15am - 11:50am

Maintainer Track: Kubernetes WG Device Management - Advancing K8s Support for GPUs | **Wednesday** 2:30pm - 3:05pm

AIStore as a Fast Tier Storage Solution: Enhancing Petascale Deep Learning Across Cloud Backends | **Wednesday** 2:30pm - 3:05pm

A Tale of 2 Drivers: GPU Configuration on the Fly Using DRA | **Wednesday** 3:25pm - 4:00pm

Tutorial: Get the Most Out of Your GPUs on Kubernetes with the GPU Operator | **Wednesday** 4:30pm - 6:00pm

From Silicon to Service: Ensuring Confidentiality in Serverless GPU Cloud Functions | **Thursday** 11:00am - 11:35am

Unlocking Potential of Large Models in Production | **Thursday** 2:30pm - 3:05pm

Which GPU Sharing Strategy Is Right for You? a Comprehensive Benchmark Study Using DRA | **Thursday** 4:30pm - 5:05pm

Engaging the KServe Community, The Impact of Integrating a Solutions with Standardized CNCF Projects | **Thursday** 5:25pm - 6:00pm

Maintainer Track: WG Serving: Accelerating AI/ML Inference Workloads on Kubernetes | **Friday** 11:55am - 12:30pm

From Vectors to Pods: Integrating AI with Cloud Native | **Friday** 2:00pm - 2:35pm

Enabling Fault Tolerance for GPU Accelerated AI Workloads in Kubernetes | **Friday** 2:55pm - 3:30pm

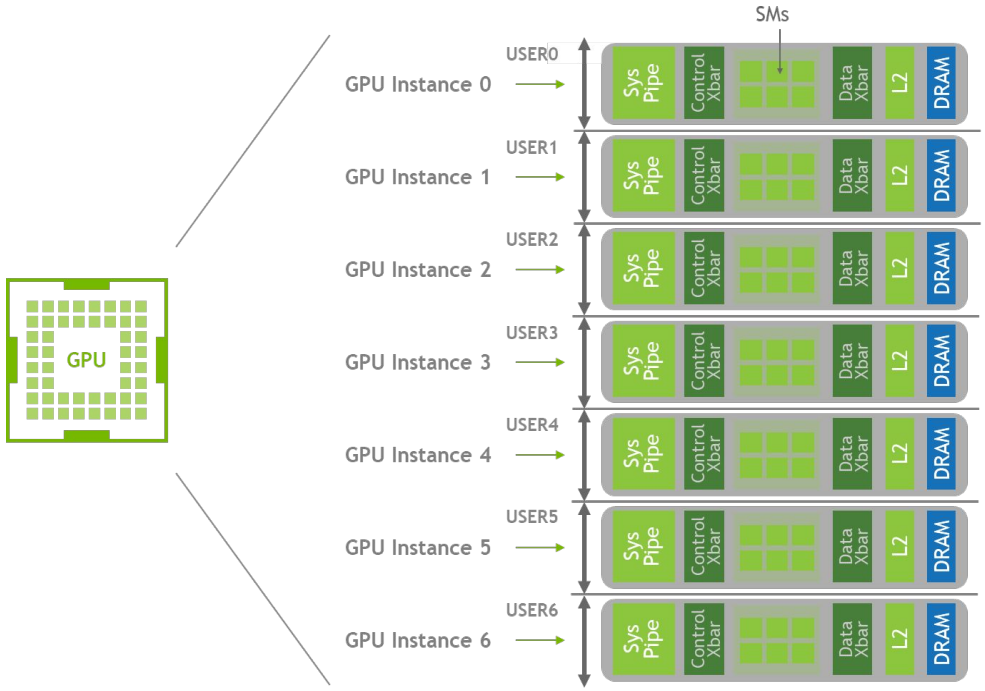
Thousands of Gamers, One Kubernetes Network | **Friday** 2:55pm - 3:30pm

Best Practices for Deploying LLM Inference, RAG and Fine Tuning Pipelines on K8s | **Friday** 4:00pm - 4:35pm

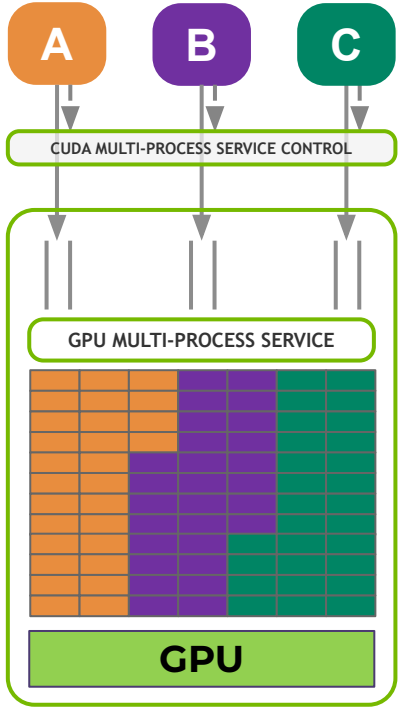
Best of Both Worlds: Integrating Slurm with Kubernetes in a Kubernetes Native Way | **Friday** 4:55pm - 5:30pm

Hardware vs. Software-Based Space Partitioning

Multi-Instance GPU (MIG)



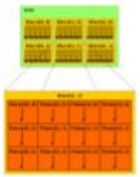
Multi-Process Service (MPS)



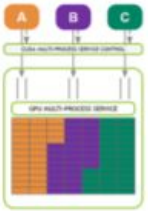
GPU Sharing

GPU "CONCURRENCY"

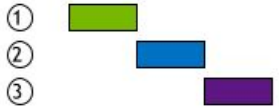
Choices



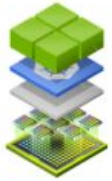
Single Process in CUDA



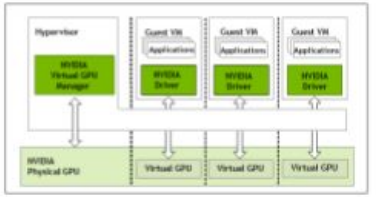
Multi-Process with CUDA MPS



Time-slicing



MIG



Virtualization with vGPU

Application level

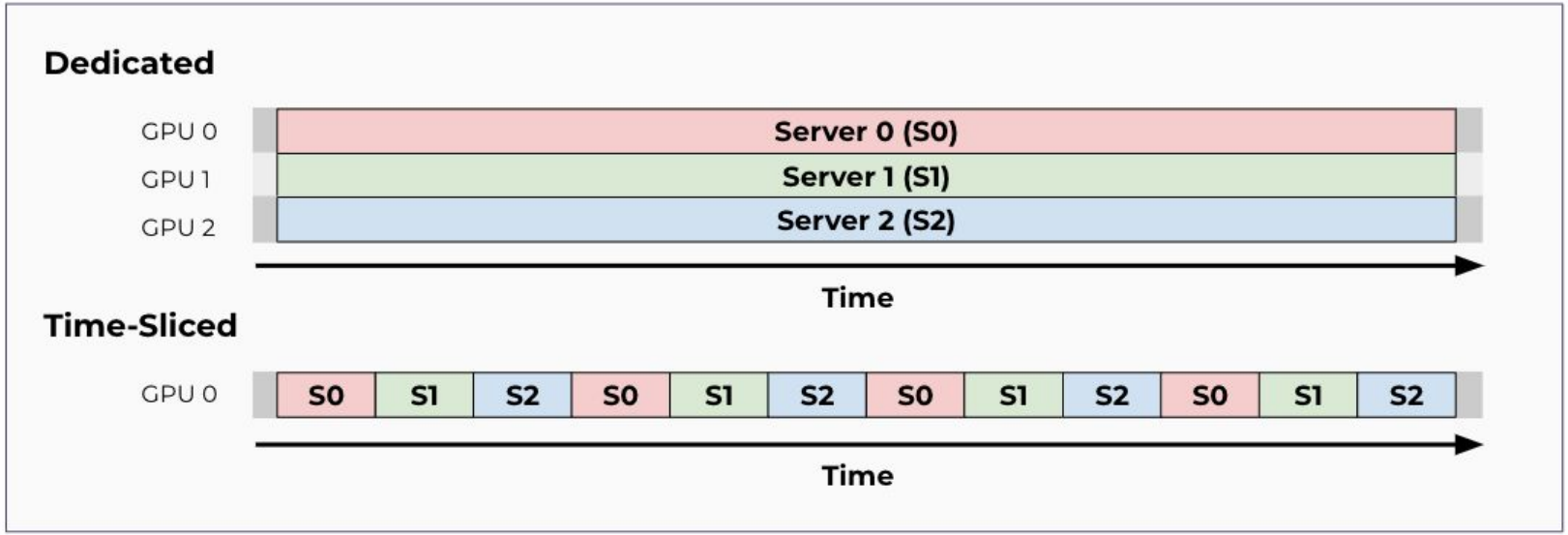
(using the CUDA programming model APIs - CUDA streams)

GPU System Software / Hardware
(Mostly transparent to CUDA applications)

https://developer.nvidia.com/blog/improving-gpu-utilization-in-kubernetes/

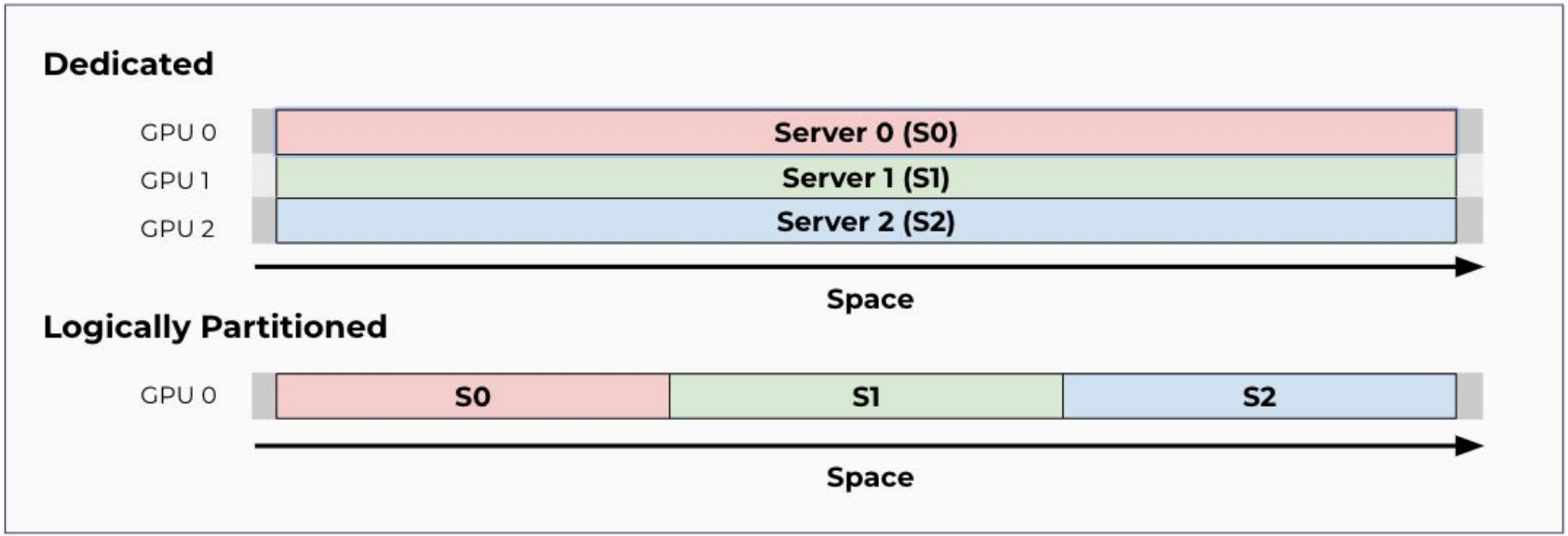
Time-Slicing (TS)

- Simple oversubscription of multiple workloads on a GPU
- Managed by the CUDA runtime
- **Example:** Serve one inference server per time-slice



Multi-Process Services (MPS)

- Logical Partitioning of memory and compute resources between workloads
- Managed by dedicated MPS server running on host
- **Example**: Serve one inference server per logical partition



Multi-Instance GPU (MIG)

- Hardware partitioning of full GPU into fixed set of mini-GPUs
- Full isolation between workloads running on different MIG devices
- **Example**: Serve one inference server, per tenant per hardware partition