

# Slinky: The Missing Link Between Slurm and Kubernetes

Skyler Malinowski, Alan Mutschelknaus,  
Marlow Warnicke, Tim Wickberg

Cray User Group 2025



# Introduction

# What is Slurm?

- Leading HPC Workload Manager
  - Workload Manager = Scheduler + Resource Manager
    - Roughly equivalent to "Orchestrator"
  - Scheduler:
    - Prioritize and decide which jobs to run on which parts of the system
  - Resource Manager:
    - Track node state and resources
    - Launch jobs
- Manages the majority of the TOP500 supercomputers
  - Also manages most AI/ML training workloads
  - Scales beyond 15,000 nodes in the cluster
- Open-Source
  - GPL-v2+



# Who are SchedMD?

- Developers of Slurm – and Slinky
- Spun off from LLNL in 2012 to support Slurm's rapid adoption
  - Founders are Moe and Danny, the "MD" in SchedMD
- SchedMD provides commercial support for Slurm, alongside
  - Training
  - Consultation
  - Custom Development

# What is Slinky?



# What is Slinky?

- Toolkit of projects to integrate Slurm into Kubernetes
- Open Source
  - Apache-2.0
- Three major components:
  - Slurm-operator
  - Slurm-bridge
  - Associated tooling



# What is Slinky?

- Slurm-operator
  - Kubernetes Operator for managing Slurm clusters
  - Manage Slurm compute nodes through Kubernetes pods
    - Autoscale in response to Slurm system load
  - Released in November 2024
    - v0.1.0 - November 2024
    - v0.2.0 - March 2025
    - v0.3.0 - June 2025

# What is Slinky?

- Slurm-bridge
  - Kubernetes Scheduling Plugin
  - Enable Slurm scheduling of both Kubernetes Pods and Slurm Jobs on converged clusters
  - Will be released in June 2025
    - After Slurm 25.05 release (May 2025)



# What is Slinky?

- Associated Tooling
  - Slurm Client
    - Golang Client Library for Slurm's REST API
  - Slurm Exporter
    - Prometheus Exporter for Slurm's REST API
    - Metrics to enable autoscaling
  - Helm Charts
  - Container Images

# Slinky Repositories



<https://github.com/SlinkyProject>

# Batch Workload Classification

# Batch workload classification

- Massively parallel
  - AI/ML training
  - Capability HPC workloads
  - Requires close coordination between application, MPI / interconnect, scheduler
    - Commonly involves esoteric libraries such as libfabrics, PMI2 / PMIx
  - Kubernetes does not manage these workloads well today
    - Multi-node work is an afterthought, all approaches are kludges on top of the existing architecture
  - Defacto standard solution is Slurm
    - Not expecting to change within the short term
  - Slurm operator is the intended solution here
    - Longer term other Slinky projects may start to address Kubernetes limitations

# Batch workload classification

- Loosely-coupled sub-node batch workloads
  - AKA "embarrassingly parallel"
  - HPC capacity workloads
  - Slurm bridge can improve scheduling performance
    - Allow for better job queuing and prioritization
  - DRA resource allocation model used to sub-divide resources
  - SchedMD is pushing for similar support for CPUs
    - Will likely publish a reference DRA CPU driver

# Batch workload classification

- Full-node
  - Slurm Bridge can provide better prioritization
  - "Easiest" jobs to manage, both in traditional batch schedulers and Kubernetes

# Batch workload classification

- Lightly multi-node
  - AI/ML inference workloads that no longer fit within single nodes
  - Kubernetes supports these workloads, but not efficiently
  - Target for first release of the Slurm Bridge

# Slurm Operator



# Slurm Operator Use Cases

- Manage Slurm clusters within a Kubernetes environment
- Each compute node maps to a Kubernetes pods running the slurmd process
- Support autoscaling based on cluster utilization metrics
- Run Slurm jobs natively
  - Users interact with Slurm through traditional CLI tools
    - Through one or more "login node" pods they can SSH into
- Kubernetes is not involved in scheduling or managing compute jobs
  - Slurm runs Slurm workloads directly
    - Allows for fine-grained resource limits
    - Backfill scheduling
    - Respect network topology - especially for NVIDIA NVL interconnects
  - Allow large training workloads to run efficiently
  - Provide access to traditional HPC tooling such as PMI/PMIx

# Documentation

- Initial documentation – <https://slinky.schedmd.com/>

# Slurm Operator Demo Screenshots

Every 1.0s: kubectl exec -n slurm statefulset/slurm-controller -- squeue; echo; kubectl... bluemachine: Mon Jul 29 19:19:24 2024

| JOBID | PARTITION | NAME | USER  | ST | TIME | NODES | ODELIST(REASON)          |
|-------|-----------|------|-------|----|------|-------|--------------------------|
| 221   | purple    | wrap | slurm | PD | 0:00 | 2     | (Resources)              |
| 224   | purple    | wrap | slurm | PD | 0:00 | 2     | (Resources)              |
| 226   | purple    | wrap | slurm | PD | 0:00 | 2     | (Resources)              |
| 227   | purple    | wrap | slurm | PD | 0:00 | 2     | (Resources)              |
| 229   | purple    | wrap | slurm | PD | 0:00 | 2     | (Resources)              |
| 231   | purple    | wrap | slurm | PD | 0:00 | 2     | (Resources)              |
| 232   | purple    | wrap | slurm | PD | 0:00 | 2     | (Resources)              |
| 234   | purple    | wrap | slurm | PD | 0:00 | 2     | (Resources)              |
| 235   | purple    | wrap | slurm | PD | 0:00 | 1     | (Resources)              |
| 236   | purple    | wrap | slurm | PD | 0:00 | 2     | (Resources)              |
| 237   | purple    | wrap | slurm | PD | 0:00 | 2     | (Resources)              |
| 238   | purple    | wrap | slurm | PD | 0:00 | 1     | (Resources)              |
| 216   | purple    | wrap | slurm | R  | 0:38 | 2     | kind-worker,kind-worker2 |

| PARTITION | AVAIL | TIMELIMIT | NODES | STATE | ODELIST                  |
|-----------|-------|-----------|-------|-------|--------------------------|
| purple*   | up    | infinite  | 2     | alloc | kind-worker,kind-worker2 |

| NAME                           | READY | STATUS  | RESTARTS      | AGE | IP          | NODE         | NOMINATED | NODE | READINESS | GATES |
|--------------------------------|-------|---------|---------------|-----|-------------|--------------|-----------|------|-----------|-------|
| slurm-compute-purple-55gch     | 1/1   | Running | 0             | 4d  | 10.244.2.11 | kind-worker2 | <none>    |      | <none>    |       |
| slurm-compute-purple-xgdnb     | 1/1   | Running | 5 (3d23h ago) | 4d  | 10.244.1.9  | kind-worker  | <none>    |      | <none>    |       |
| slurm-controller-0             | 2/2   | Running | 0             | 4d  | 10.244.2.12 | kind-worker2 | <none>    |      | <none>    |       |
| slurm-metrics-79c86f5978-s5wdv | 1/1   | Running | 0             | 4d  | 10.244.2.9  | kind-worker2 | <none>    |      | <none>    |       |
| slurm-restapi-79f44bff7d-9pmqr | 1/1   | Running | 0             | 4d  | 10.244.1.7  | kind-worker  | <none>    |      | <none>    |       |





# Slurm Bridge

# Why Slurm Bridge

- Kubernetes lacks fine-grained control of native resources (CPU, Memory)
  - HPC and AI training workloads are generally more efficient when dedicated resources are assigned
    - Avoid jitter and cache contention
- Ability to have fast scheduling that is not possible in kubelet
- Ability to use both Kubernetes and Slurm workloads on the same set of nodes
  - Allow researchers to use their preferred tooling, without needing separate dedicated compute systems

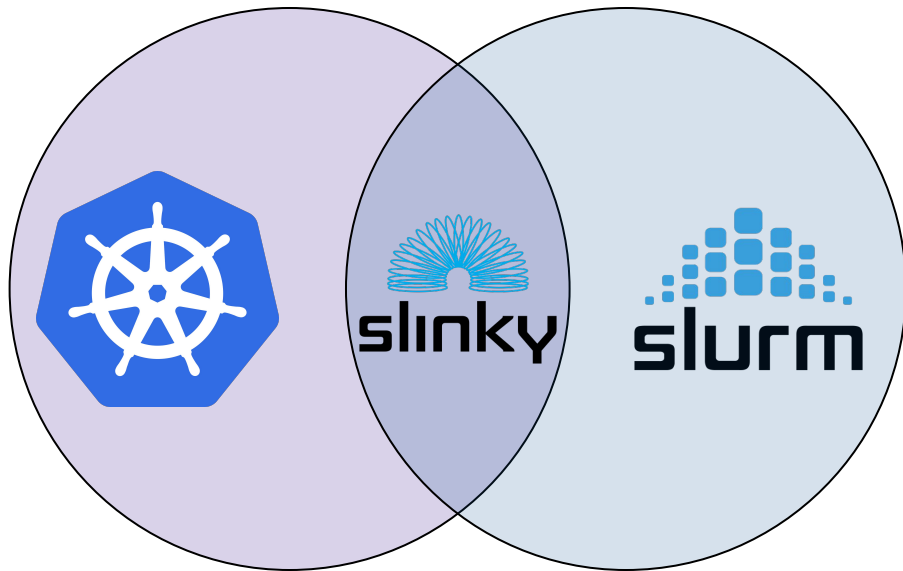
# Why Not Slurm Bridge

- Slurm Bridge is not meant to replace the default scheduler
  - Another alternative
    - Kubernetes API makes it possible to provision multiple schedulers
    - Same approach taken by Kueue, Volcano, MPI Operator, ...
  - However... as the Kubernetes API doesn't provide a clean way to sub-divide resources within a node, it does assume that - for any node it's meant to schedule - that is is the only workload scheduler
    - Disregard core infrastructure - such as daemon sets - that are still scheduled through the default scheduler
- Slurm Bridge may not be appropriate for your system
  - Intended for clusters that are predominantly dedicated to batch-oriented process
    - Or closely related domains - such as AI/ML inference
      - Especially for managing multi-node inference workloads

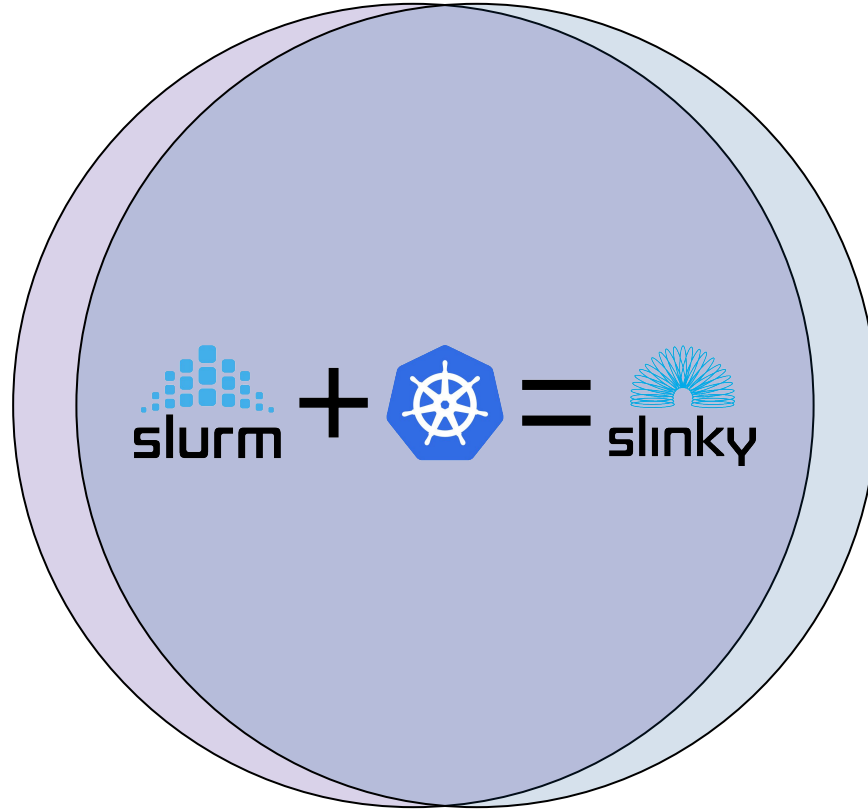


# Domain Pools

- Kubernetes manages its nodes
  - Running kubelet
- Slurm manages its nodes
  - Running slurmd
- The Slurm-Bridge manages workloads running on converged nodes shared by both
- Nodes are not required to run both, but for most deployments they likely will



## Domain Pools - Expected Deployment Pattern



# Design Goals

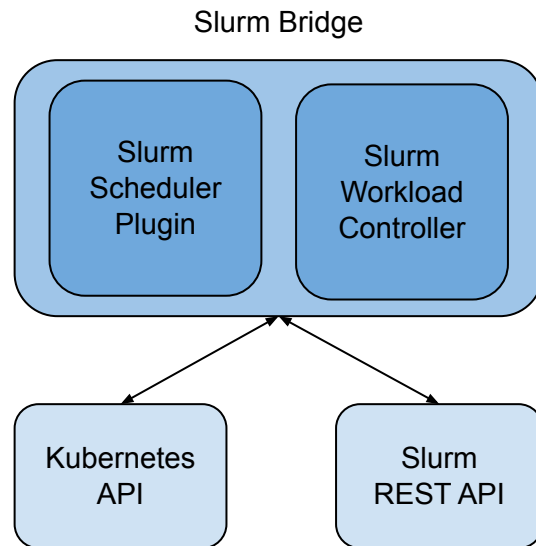
- Run both Slurm and Kubernetes workloads on pools of nodes
- Slurm bridge will translate resource requirements for Kubernetes workloads into Slurm jobs
  - Reconstruct multi-node workloads, and submit single job to Slurm
    - PodGroup and JobSet currently
      - Likely LeaderWorkerSet as well
- Handle Device Plugins, such as GPUs
- Filter out nodes that Slurm is not to manage, through the current set of labels provided
- Filter out pods out via designated namespaces
  - Will have an allow-list of namespaces we handle
    - "slurm-bridge" in our demo

# Restrictions

- Each node can run Slurm **or** Kubernetes workloads, not both concurrently
  - The kubelet will manage Kubernetes pods
  - The slurmd will manage Slurm jobs
- Configure the Slurm-bridge plugin as Kubernetes scheduling profile
  - Plugin will take control of all workloads in allow-list of namespaces
  - The Default Scheduler will handle all other workloads
- Slurm can only schedule to nodes with slurmd running
  - Even if you don't want to run native Slurm workloads
  - Need detailed CPU information that the Kubernetes API doesn't provide
    - Can use the Slurm Operator to manage these slurmd processes
      - Or run slurmd directly on base-metal

# Slurm Bridge Scheduler + Controller

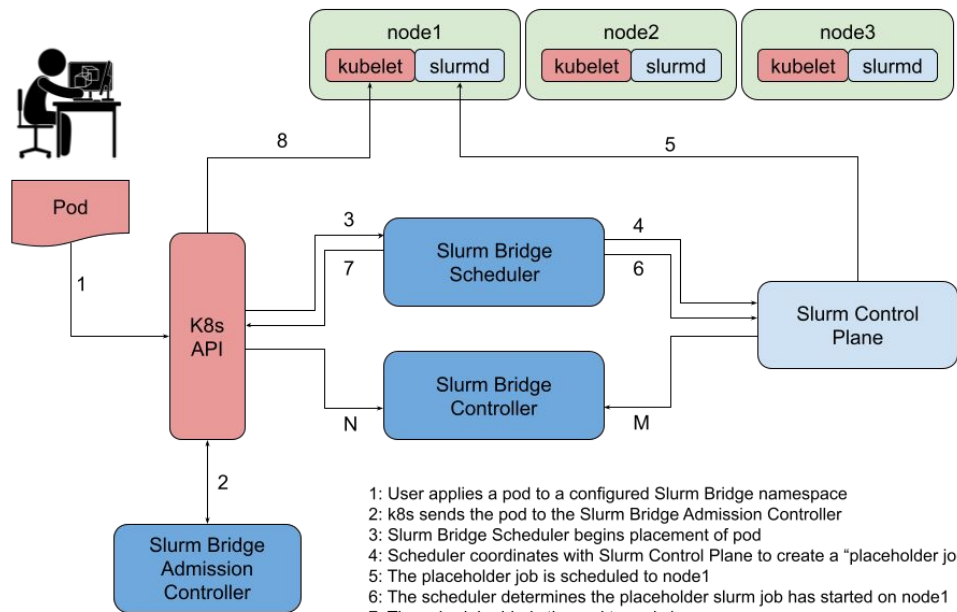
- Responsible for managing Slurm as the source of truth and enforcing scheduling decisions from Slurm
- Slurm Scheduler Plugin
  - Hooks into the Kubernetes scheduling API to utilize the Slurm Control Plane to make scheduling decisions
- Slurm Workload Controller
  - Reconciles pod drift/desync using Slurm as the source-of-truth for Slurm scheduled workloads



# Slurm Bridge

## User's Perspective

# Slurm Bridge - User's Perspective



- 1: User applies a pod to a configured Slurm Bridge namespace
- 2: k8s sends the pod to the Slurm Bridge Admission Controller
- 3: Slurm Bridge Scheduler begins placement of pod
- 4: Scheduler coordinates with Slurm Control Plane to create a "placeholder job"
- 5: The placeholder job is scheduled to node1
- 6: The scheduler determines the placeholder slurm job has started on node1
- 7: The scheduler binds the pod to node1
- 8: kubelet starts the pod on node1

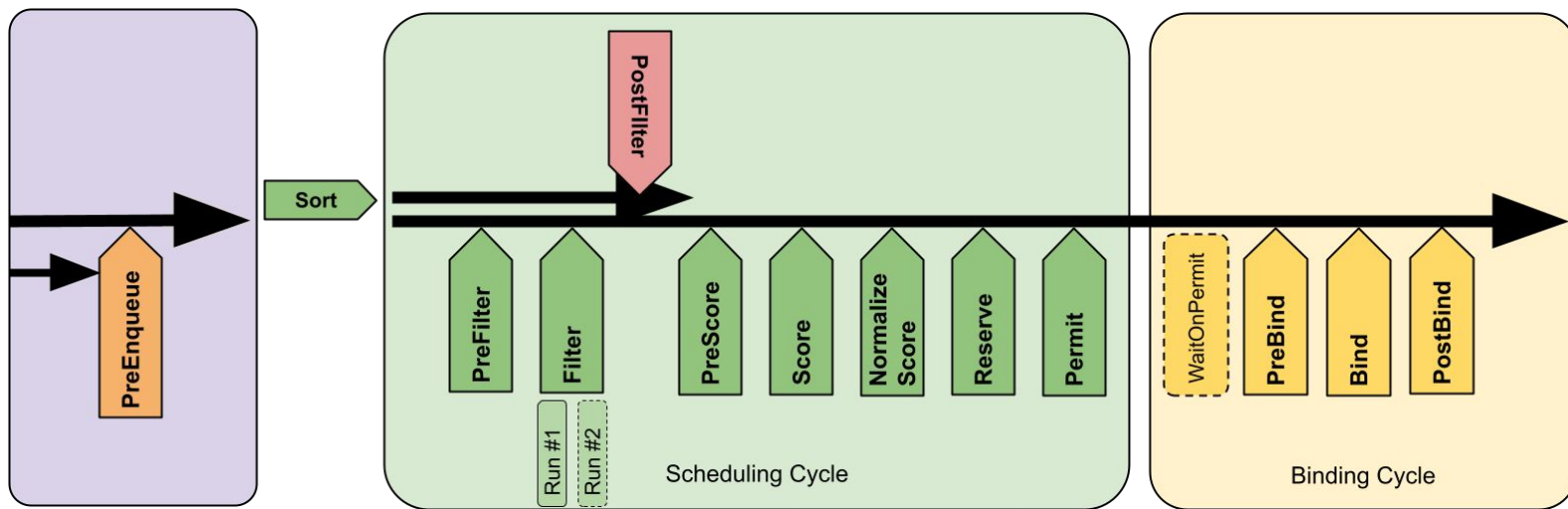
N: Slurm Bridge Controller reconciles k8s node and pod events  
M: Slurm Bridge Controller reconciles Slurm node and job events

# Slurm Bridge

## Kubernetes Scheduler Plugin

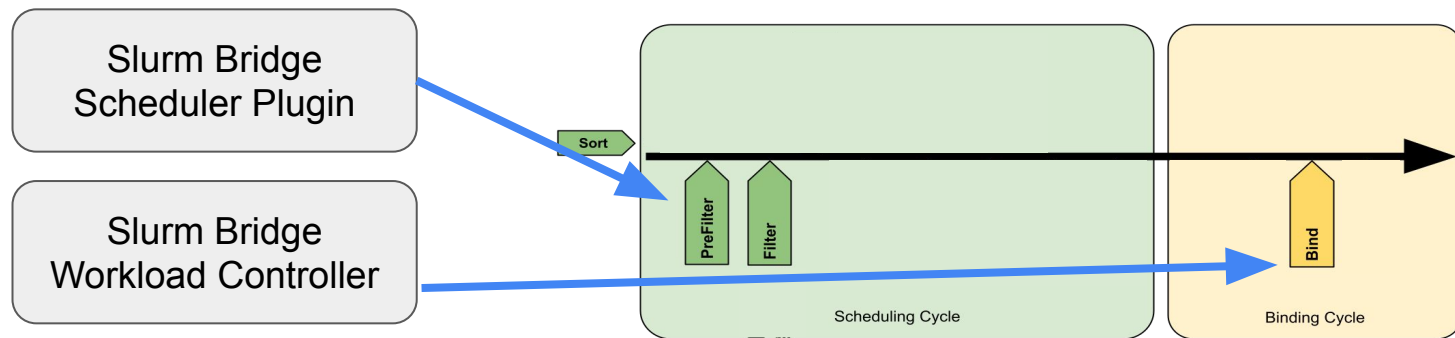


# Kubernetes Scheduler Framework



# Slurm Scheduler Plugin

- Only implement PreFilter/Filter and Bind
- PreFilter to capture new pod requests
  - To translate Pod into Slurm job and submit into Slurm's queues
- Bind to communicate the node allocation back to Kubernetes
  - Technically managed by the workload controller, not the scheduler plugin
- Does not implement all Kubernetes scheduling primitives
  - E.g., affinity/anti-affinity aren't available
  - Avoids some performance pitfalls of the Kubernetes scheduling API



# Slurm Bridge

## Demo Screenshots

```
apiVersion: v1
kind: Pod
metadata:
  name: pause-pod
  namespace: slurm-bridge
  annotations:
    slinky.slurm.net/job-name: "pausepod"
spec:
  containers:
  - name: pause-pod
    image: registry.k8s.io/pause:3.6
```

```
$ kubectl apply -f pause-pod.yaml.debug
pod/pause-pod created
```

```
$ squeue
```

| JOBID | PARTITION | NAME     | USER  | ST | TIME | NODES | NODELIST(REASON) |
|-------|-----------|----------|-------|----|------|-------|------------------|
| 16    | slurm-bri | pausepod | slurm | R  | 0:11 | 1     | slurm-bridge-1   |

```
$ kubectl get pods -o wide -n slurm-bridge
```

| NAME      | READY | STATUS  | RESTARTS | AGE | IP          | NODE           | NOMINATED | NODE | READINESS | GATES |
|-----------|-------|---------|----------|-----|-------------|----------------|-----------|------|-----------|-------|
| pause-pod | 1/1   | Running | 0        | 17s | 10.244.2.12 | slurm-bridge-1 | <none>    |      | <none>    |       |

```
apiVersion: v1
kind: Pod
metadata:
  annotations:
    kubectl.kubernetes.io/last-applied-configuration: ...
    slinky.slurm.net/job-name: pausepod
    slinky.slurm.net/slurm-node: slurm-bridge-1
  creationTimestamp: "2025-03-26T12:38:17Z"
  finalizers:
  - scheduler.slurm.net/finalizer
  labels:
    scheduler.slinky.slurm.net/slurm-jobid: "16"
  name: pause-pod
  namespace: slurm-bridge
  ...
spec:
  containers:
  ...
  schedulerName: slurm-bridge-scheduler
  tolerations:
    key: slinky.slurm.net/managed-node
    operator: Equal
    value: slurm-bridge-scheduler
```



```

apiVersion: scheduling.x-k8s.io/v1alpha1
kind: PodGroup
metadata:
  name: nginx-pg
  namespace: slurm-bridge
  annotations:
    slinky.slurm.net/job-name: pgReplicaset
spec:
  minMember: 2
  ---
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: nginx-pg
  namespace: slurm-bridge
  labels:
    app: nginx-pg
spec:
  replicas: 2
  selector:
    matchLabels:
      app: nginx-pg
  template:
    metadata:
      name: nginx-pg
      namespace: slurm-bridge
      labels:
        app: nginx-pg
        scheduling.x-k8s.io/pod-group: nginx-pg
    spec:
      containers:
        - name: nginx-pg
          image: nginx
          resources:
            limits:
              cpu: 3000m
              memory: 500Mi
            requests:
              cpu: 3000m
              memory: 500Mi

```

```
# Slurm Bridge Scheduler Pods
NAME          READY  STATUS   RESTARTS   AGE   NODE
nginx-pg-fwcdc 1/1    Running  0          14s   slurm-bridge-1
nginx-pg-rq2kk 1/1    Running  0          14s   slurm-bridge-2

# PodGroup Status
NAME          PHASE    MINMEMBER  RUNNING  SUCCEEDED  FAILED  AGE
nginx-pg      Running  2          2        0           0       14s

# Slurm sinfo
JOBID  PARTITION  NAME          USER  ST  TIME  NODES  NODELIST(REASON)
17     slurm-bridge  pgReplicaset  slurm  R   0:13  2      slurm-bridge-[1-2]

# Slurm squeue
PARTITION  AVAIL  TIMELIMIT  NODES  STATE  NODELIST
slurm-bridge  up     infinite   2      alloc  slurm-bridge-[1-2]
slurm-bridge  up     infinite   1      idle   slurm-bridge-0
```



```

$ cat podgroup.yaml.debug
---
apiVersion: scheduling.x-k8s.io/v1alpha1
kind: PodGroup
metadata:
  name: sleep-pg
  namespace: slurm-bridge
  annotations:
    slinky.slurm.net/account: slurm
    slinky.slurm.net/job-name: podgroupSleep
spec:
  minMember: 2
---
apiVersion: v1
kind: Pod
metadata:
  name: sleep1
  namespace: slurm-bridge
  labels:
    app: sleep-pg
    scheduling.x-k8s.io/pod-group: sleep-pg
spec:
  restartPolicy: Never
  containers:
  - name: my-container
    image: busybox
    command: ["sh", "-c", "sleep 20 && exit 0"]
---
apiVersion: v1
kind: Pod
metadata:
  name: sleep2
  namespace: slurm-bridge
  labels:
    app: sleep-pg
    scheduling.x-k8s.io/pod-group: sleep-pg
spec:
  restartPolicy: Never
  containers:
  - name: my-container
    image: busybox
    command: ["sh", "-c", "sleep 20 && exit 0"]

```



```

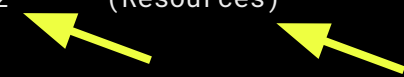
# Slurm Bridge Scheduler Pods
NAME          READY  STATUS   RESTARTS  AGE  NODE
nginx-pg-fwcdc 1/1    Running  0          91s  slurm-bridge-1
nginx-pg-rq2kk 1/1    Running  0          91s  slurm-bridge-2
sleep1         0/1    Pending  0          4s   <none>
sleep2         0/1    Pending  0          4s   <none>

# PodGroup Status
NAME          PHASE      MINMEMBER  RUNNING  SUCCEEDED  FAILED  AGE
nginx-pg      Running    2          2        0           0       91s
sleep-pg      Scheduling 2          0        0           0       5s

# Slurm sinfo
JOBID  PARTITION  NAME          USER  ST  TIME  NODES  NODELIST(REASON)
17     slurm-bridge  pgReplicaset  slurm  R   1:30  2      slurm-bridge-[1-2]
18     slurm-bridge  podgroupSleep  slurm  PD  0:00  2      (Resources)

# Slurm squeue
PARTITION  AVAIL  TIMELIMIT  NODES  STATE  NODELIST
slurm-bridge  up     infinite   2      alloc  slurm-bridge-[1-2]
slurm-bridge  up     infinite   1      idle   slurm-bridge-0

```



Note that this second workload is pending - insufficient nodes available. Slurm will schedule this once resources are available.

```
# Slurm Bridge Scheduler Pods
NAME    READY  STATUS   RESTARTS  AGE  NODE
sleep1  1/1    Running  0          44s  slurm-bridge-1
sleep2  1/1    Running  0          44s  slurm-bridge-2

# PodGroup Status
NAME      PHASE    MINMEMBER  RUNNING  SUCCEEDED  FAILED  AGE
sleep-pg  Running  2          2        0           0       45s

# Slurm sinfo
JOBID  PARTITION  NAME           USER  ST  TIME  NODES  NODELIST(REASON)
18     slurm-bridge  podgroupSleep  slurm  R   0:10  2      slurm-bridge-[1-2]

# Slurm squeue
PARTITION  AVAIL  TIMELIMIT  NODES  STATE  NODELIST
slurm-bridge  up    infinite   2      alloc  slurm-bridge-[1-2]
slurm-bridge  up    infinite   1      idle   slurm-bridge-0
```

```
# Slurm Bridge Scheduler Pods
NAME    READY  STATUS    RESTARTS  AGE  NODE
sleep1  0/1    Completed  0          75s  slurm-bridge-1
sleep2  0/1    Completed  0          75s  slurm-bridge-2

# PodGroup Status
NAME    PHASE    MINMEMBER  RUNNING  SUCCEEDED  FAILED  AGE
sleep-pg  Finished  2          0        2          0      77s

# Slurm queue
PARTITION  AVAIL  TIMELIMIT  NODES  STATE  NODELIST
slurm-bridge  up    infinite   3      idle  slurm-bridge-[0-2]
```

# Future Work

# Future Work

- Further refinement, documentation, and testing of the Slurm Operator
- Work with the Kubernetes community to be able to handle fine-grained control and understanding of native resources
  - "DRA-for-Cores"
  - Publish CPU affinity mapping for other DRA devices
- Allow for Slurm to operate as a pure Kubernetes scheduler
  - Remove requirement for slurmd daemon on nodes managed by the Slurm Bridge
    - Requires new "external" node status within Slurm to indicate Slur's own resource management layer is disabled
  - Requires extension to the Slurm Workload Controller to automatically create "external" nodes within Slurm
- Investigation into better coordination with Autoscaler

# CPU affinity - HPC requirements

- HPC workloads have a broad range of ways to model their internal application layouts
- HPC workload managers evolved to support a huge range of options
- Subset of these allocation options:
  - number-of-tasks, number-of-nodes, number-of-tasks-per-node
  - cpus, cpus-per-gpu, cpus-per-node, cpus-per-task
  - gpus, gpus-per-node, gpus-per-task, gpus-per-socket
  - sockets-per-node, threads-per-core
  - gpu-to-cpu-pinning

# CPU resource management

- CPU resource management
  - Significant functional gap compared to Slurm's native resource management
  - CPU affinity has significant performance impacts on most workloads
    - Managed by through the Linux cpuset cgroup controller
      - Kubernetes lacks centralized planning for CPUs
        - Delegated to the runtime
          - But precludes effective backfill scheduling
  - Discussing different models with the device management wg and others
    - May publish a POC DRA driver for CPUs while discussing whether something should be pushed into core Kubernetes

**Questions?**



Thank You



<https://github.com/SlinkyProject>

**SCHEDMD**

The Slurm Company