# Kubernetes Scheduling with Checkpoint/Restore

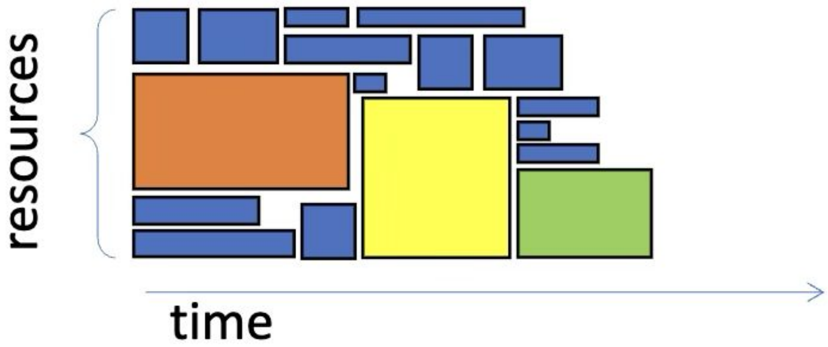
## Challenges & Open Problems

Viktória Spišaková (MU), Radostin Stoyanov (Oxford), Lukáš Hejtmánek (MU),
Dalibor Klusáček (CESNET), Adrian Reber (Red Hat), Rodrigo Bruno (ULisbon)

# Background I.

## Traditional HPC Batch Scheduling

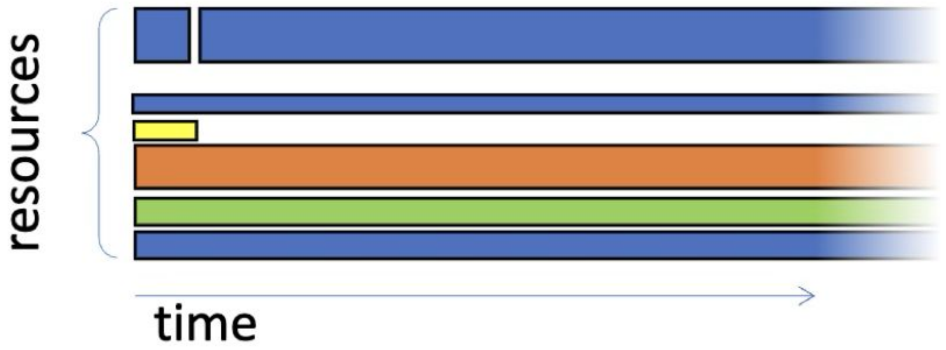
- Workloads are submitted as jobs
    - with strict runtime resource limits and requests
- Jobs are placed in queues
- Queues are ordered by priority
- Scheduler decides when and how resources are allocated to each job



# Background II.

## Kubernetes (K8s) Scheduling

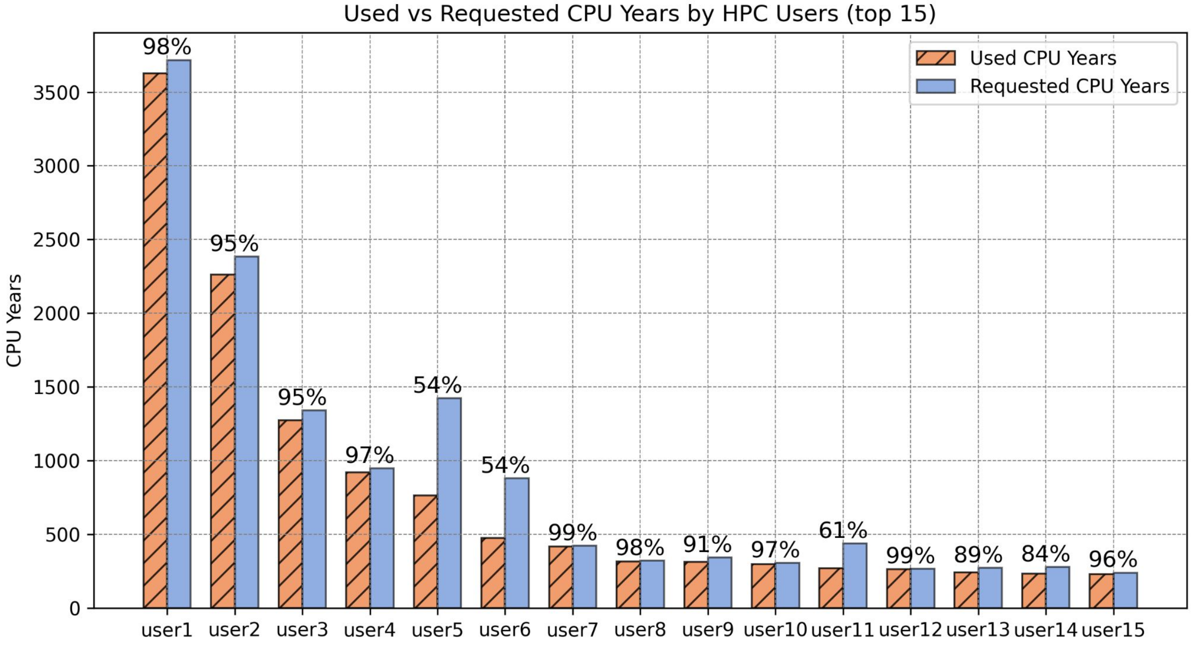
- Users declare resource requests AND limits
- Scheduler immediately tries to place the workload in the cluster
- Workloads may run for seconds, minutes... or months
- Resources remain allocated even if they are not fully used



# Resource Utilization Effectivity

## Traditional HPC Batch Clusters

OpenPBS

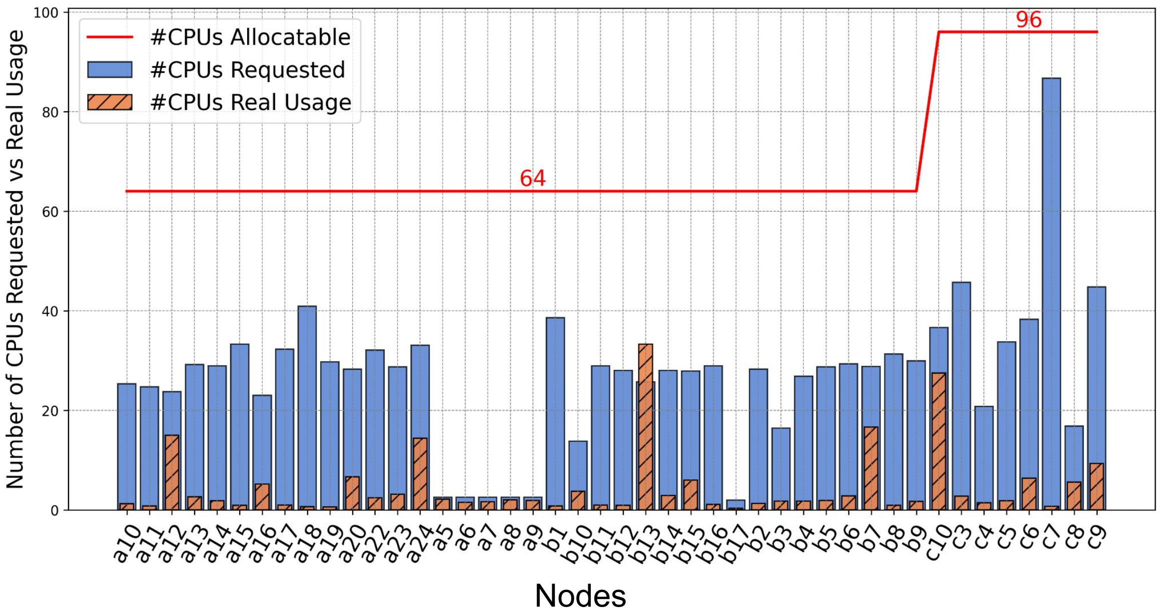



**Average cluster utilization rate overall: ~ 80%**

Real vs. Requested CPU years of the e-INFRA CZ PBSPro cluster users

# Resource Utilization Effectivity

## Kubernetes Clusters




**Average utilization rate: ~ 20% (compared to ~80% in HPC)**

Real vs. Requested CPU utilization of the e-INFRA CZ Kubernetes cluster nodes

# Problem Statement

## Elasticity Without Sacrifices in the Cloud

- Cloud hosts multiclass workloads
  - Interactive, batch, static workloads with “infinite” duration (microservices)
  - Need for **elasticity** — automatically scale up and down in response to demand
- Auto-scaling is not suitable for all workload types
- Clusters stay idle to guarantee SLA compliance → poor real utilization
- Evictions (Preemptions) result in wasted compute & lost workload state
- Default K8s scheduler does not meet the needs of multi-purpose clusters

# What if preemption didn't mean losing progress but new flexibility?

# IASS: Interruption-Aware Scheduling Strategy

**Transparent Checkpoint/Restore + Novel Scheduling Strategy**

# Transparent Checkpoint/Restore

## A New Scheduling Primitive?

- Non-Destructive Interruptions
  - Backfilling: In this context, fill-in unused resources with lower-priority jobs and quickly vacate resources (checkpoint) when they are needed
  - Accommodate high-priority/urgent workloads **without losing progress** of preempted workloads
- Dynamic workload migration across nodes
- Providing infrastructure-level fault tolerance

Improving resource utilization without compromising elasticity

# Transparent Checkpoint/Restore

## Where Are We?

- Provided by the Linux utility CRIU
- Full process tree state is checkpointed to persistent storage
  - CRIU internally uses so-called *parasitic code* (injected via *ptrace()* call)
- Transparent → no modifications to application code or OS kernel required
- After checkpointing, container can be
  - Left running → fault tolerance
  - Left stopped → free resources

# Transparent Checkpoint/Restore

## Why Not Other Alternatives?

- DMTCP, BLCR
  - **Requires workflow modifications**: must be dynamically linked at application startup time
- Application-level C/R
  - **No general solution**: must be implemented for each application
- Micro VMs
  - Promising but **not Kubernetes-native** and introduces extra overhead

➡ Transparent container C/R provided by CRIU

- Fully transparent: no workflow modifications required
- **App-agnostic and lightweight**
- Kubernetes-native

# Interruption-Aware Scheduling Strategy

## Novel Scheduling Strategy + C/R

- Scheduler is dynamic, threats interruptions and changes in the scheduling plan as a core functionality
- Key points
    - Checkpointing workloads to free resources on-demand
    - Restoring workloads when resources are available
    - Preempted applications are *checkpointed* rather than *killed*
    - Supports migration, survivable evictions and fault tolerance

# Interruption-Aware Scheduling Strategy

## CRIU Integration Within the K8s Stack

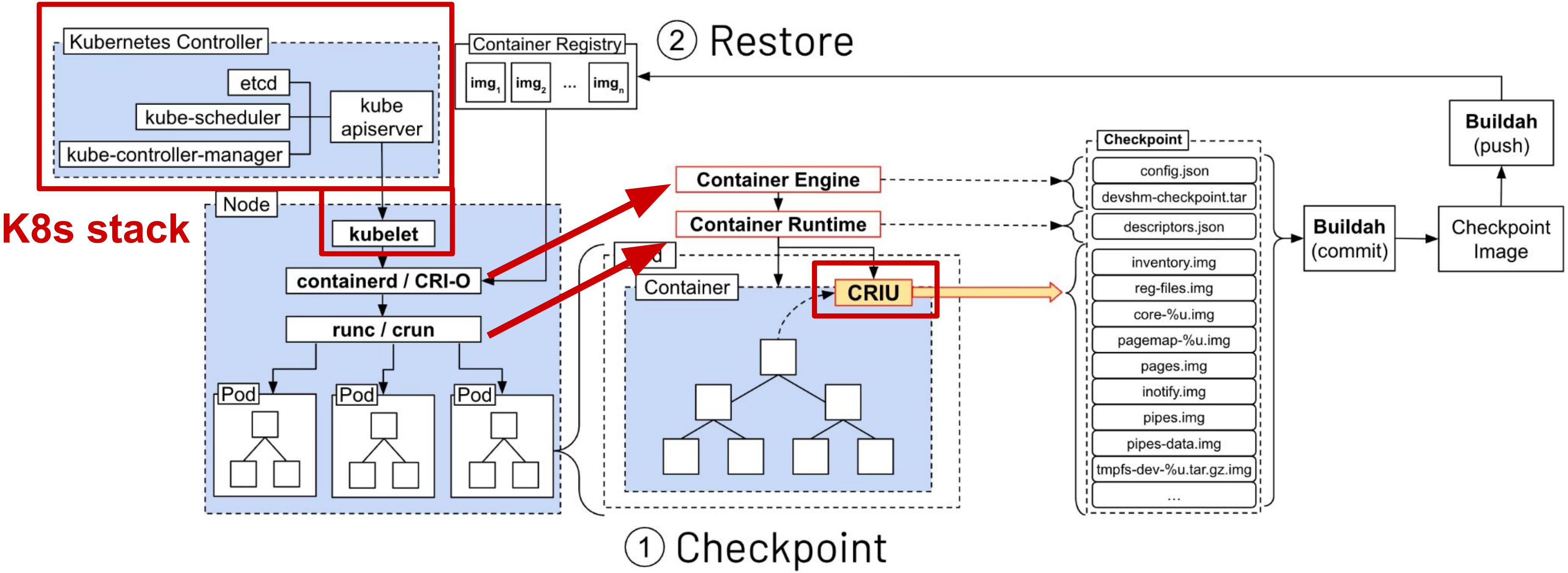



Image Source: Radostin Stoyanov, et. al., "Preemptive Scheduling of Stateful GPU Intensive HPC Applications in Kubernetes", International Workshop on Containers and New Orchestration Paradigms for Isolated Environments in HPC (CANOPIE-HPC). 2023 (full presentation slides)

# Interruption-Aware Scheduling Strategy

## Novel Scheduling Strategy – Queues

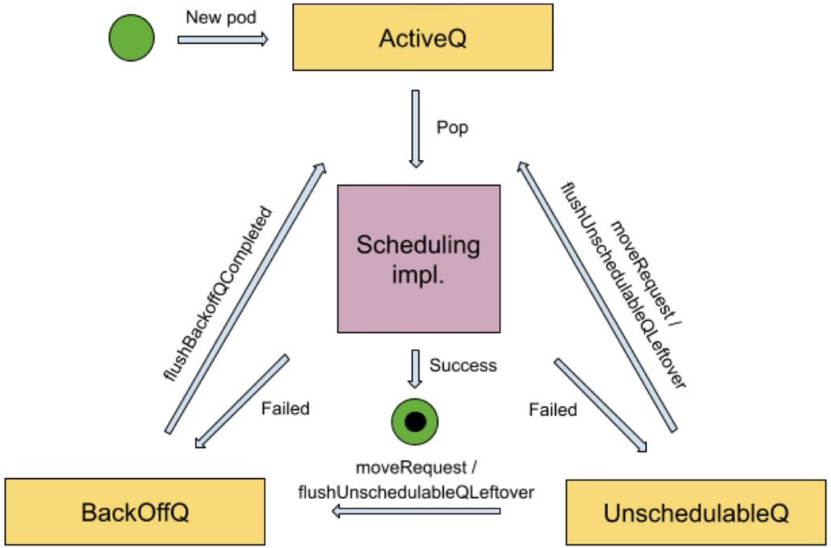



Image source

Kubernetes has 3 scheduling queues: *Active, Backoff, Unschedulable*
Workloads in Backoff queue are expected to be scheduled soon and are polled in exponential manner.

# Interruption-Aware Scheduling Strategy

## Novel Scheduling Strategy – Queues

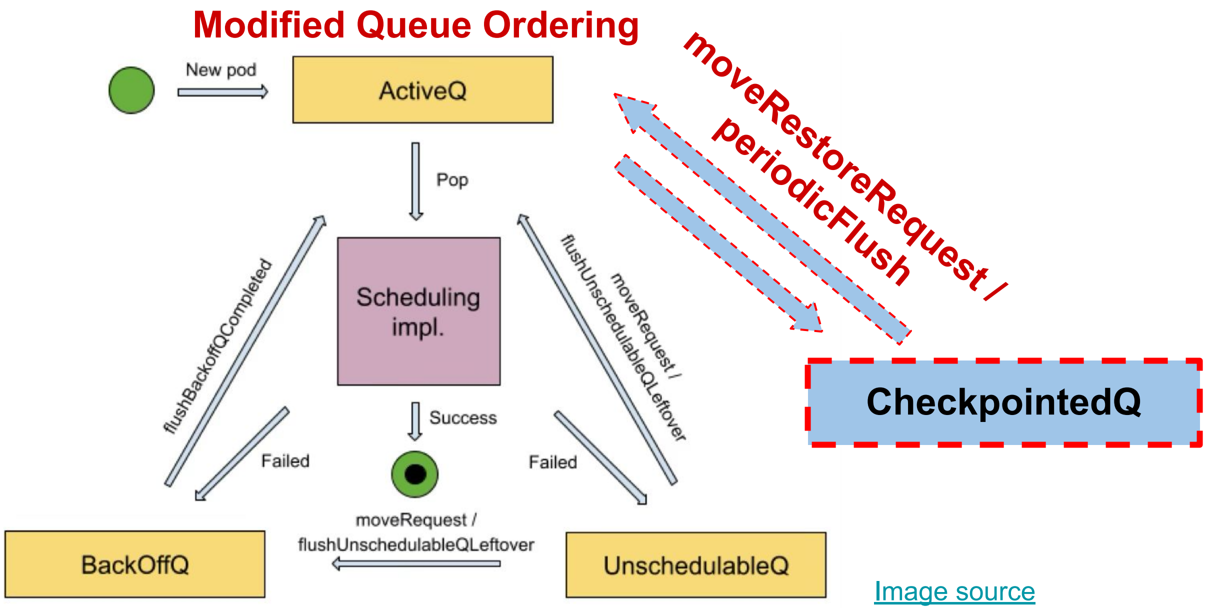



Image source

Kubernetes has 3 scheduling queues: *Active, Backoff, Unschedulable*
Workloads in Backoff queue are expected to be scheduled soon and are polled in exponential manner.

# Interruption-Aware Scheduling Strategy

## Novel Scheduling Strategy – Scheduler

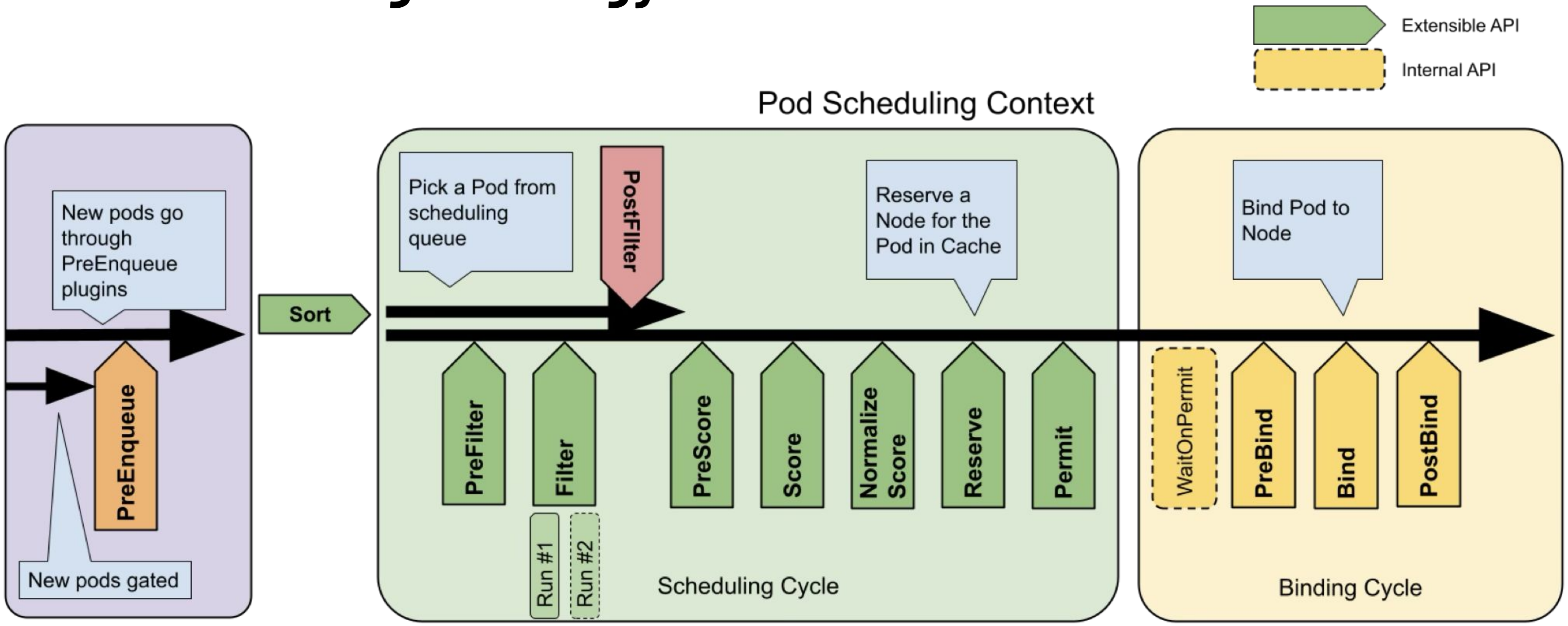



Image source

Kubernetes scheduling is represented as *scheduling framework* consisting of extensible *phases*

# Interruption-Aware Scheduling Strategy

## Novel Scheduling Strategy – Scheduler

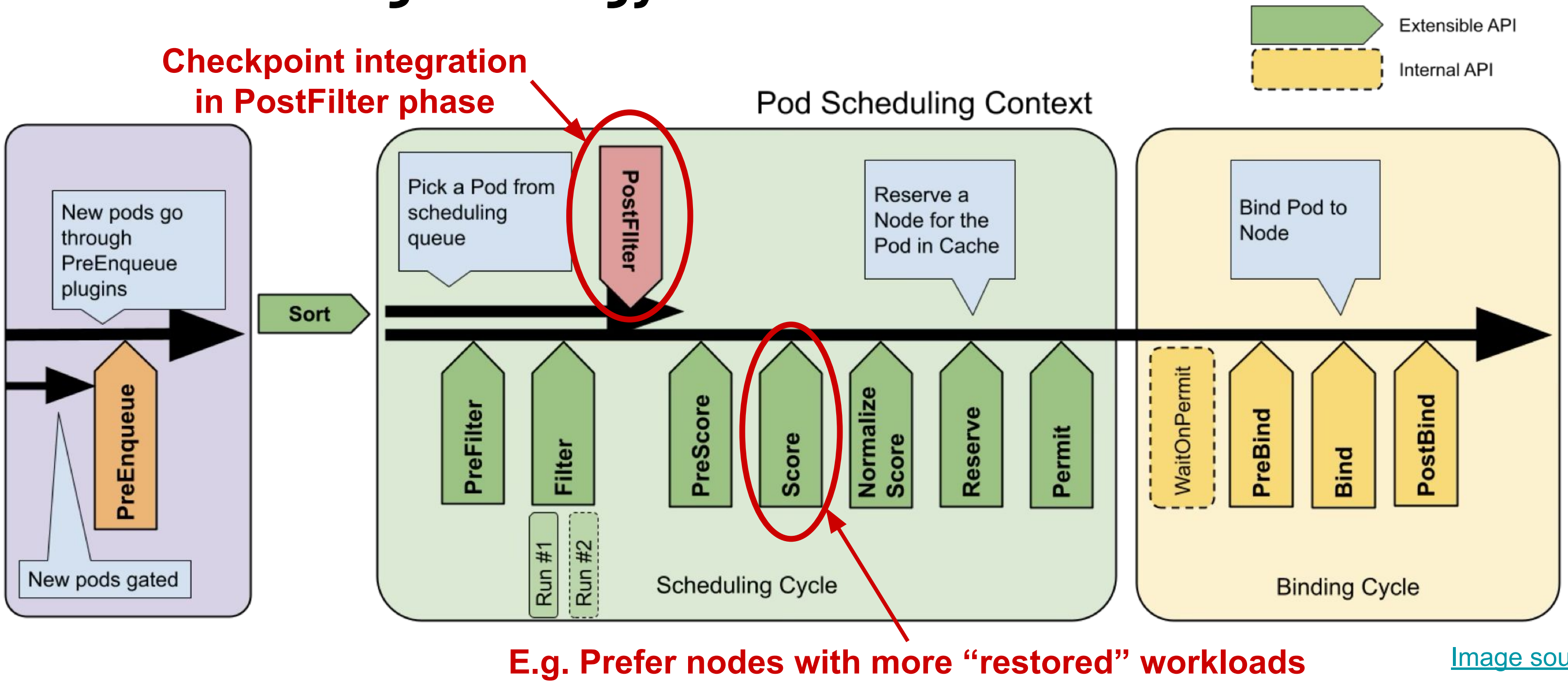



Image source

Kubernetes scheduling is represented as *scheduling framework* consisting of extensible *phases*

# Current State

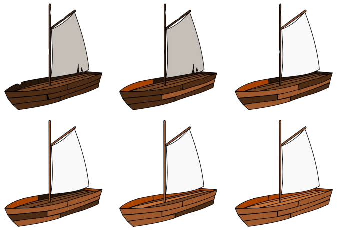
## Can K8s Support C/R-based Scheduling?

- CRIU
  - Stable enough to support simple use-cases
  - Support C/R for **GPU workloads**![1]
- Kubernetes support as forensic container checkpointing
  - Both checkpoint and restore technically work but are not integrated into scheduling
- Additional tools and frameworks supporting the C/R ecosystem
  - Kubernetes checkpoint-restore-operator to help managing checkpoints
  - End-to-end encryption for container checkpoints
  - Coordinated checkpointing of distributed applications

[1] Radostin Stoyanov et al. CRIUgpu: Transparent Checkpointing of GPU-Accelerated Workloads, 2025. arXiv

# Current State

## Challenges to Realizing IASS



Is the restored workload the original one or a new one?
(Ship of Theseus Paradox)

| | |
|---|---|
| Design & Architecture | 💡 Lack of upstream API for restoration<br>💡 Missing Kubernetes design of restored workload |
| Networking | ✅ Container IP address must remain the same to re-establish TCP connections<br>→ Load balancers or provided by overlay network |
| Security | ✅ Security by design<br>→ CRIUSec (E2E encryption scheme)[1]<br>→ Integration into K8s |
| Cost Awareness | ✅ Storage demands of checkpoints, time to take them<br>→ checkpoint-restore operator |
| Policies | 💡 C/R user transparency and relations to external world |

[1] Radostin Stoyanov, Adrian Reber, Daiki Ueno, Michał Clapiński, Andrei Vagin, and Rodrigo Bruno. Towards Efficient End-to-End Encryption for Container Checkpointing Systems

# Opportunities & Future Work

## What's Next?

- Incremental adoption
    - C/R path for limited number of workloads, e.g. for ones with small checkpoint sizes
- Introducing new scheduling policies
    - Preempting long-running jobs for sudden bursts, e.g. new Priority Class
- Collaborations and discussions within the Kubernetes community

# Conclusion

## On the Edge of Tomorrow

- Integrating C/R with the Kubernetes scheduler can improve resource utilization and increase overall flexibility
- Interruptions don’t have to result in wasted resources
- Especially valuable for multi-purpose clusters


Image source

# Thank you!

## Questions?

Contact: k8s@ics.muni.cz

criu.org
docs.cerit-sc.cz
github.com/checkpoint-restore/criu