Northeastern University
Khoury College of Computer Sciences

HPC Scheduling with Kubernetes

Siyuan Chen, Yilin Xu, Nidhi Shah, Soufiane Jounaid, Juhi Paliwal

Claudia Misale(IBM), Carlos Eduardo Arango Gutierrez (RedHat),
Daniel Milroy (Lawrence Livermore National Laboratory)

Agenda

- Project goals
  - Introduction to Kubernetes and plug-in schedulers
  - Problem description
  - Problem demonstration
- Accomplishments
  - Job cancellation management: Pod Informer solution
  - Pod Informer demonstration
  - Pod Informer Performance analysis
  - First steps for co-scheduling: Json Graph Format (JGF) Operator solution
- Conclusion and future improvements

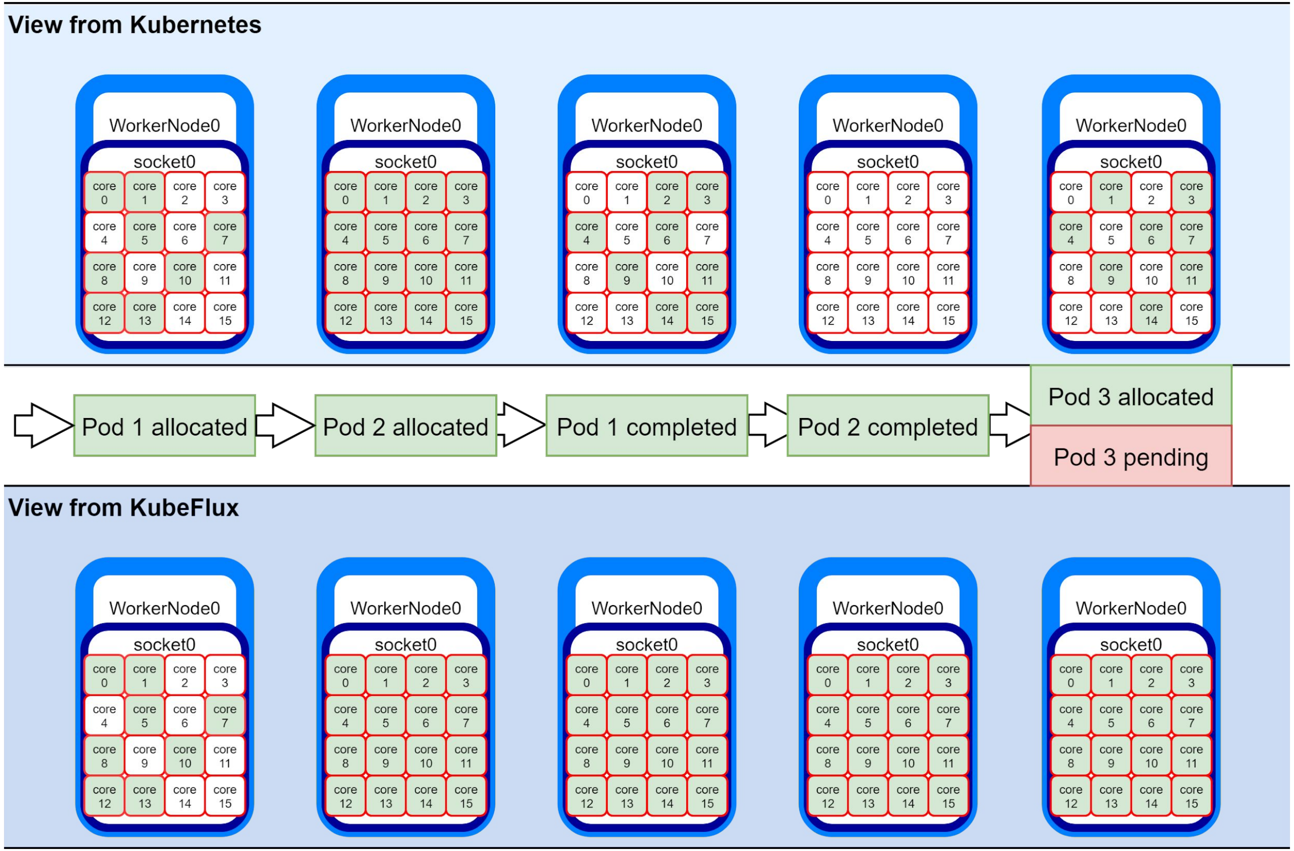
Kubernetes and KubeFlux

- Kubernetes is a portable, extensible, open-source platform for managing containerized workloads and services.
- While Kubernetes excels at orchestrating containers, high-performance computing (HPC) applications can be tricky to deploy on Kubernetes.

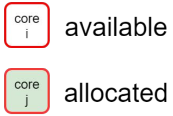
- Kube-Flux is an HPC scheduler that employs the Fluxion library to take scheduling decisions on Kubernetes.

State Inconsistency between KubeFlux and Kubernetes

Northeastern University
Khoury College of Computer Sciences



- Cluster has 1 worker node with 1 cpu socket. It has 16 cores
- Pod 1, 2 ,3 requires 8 cores

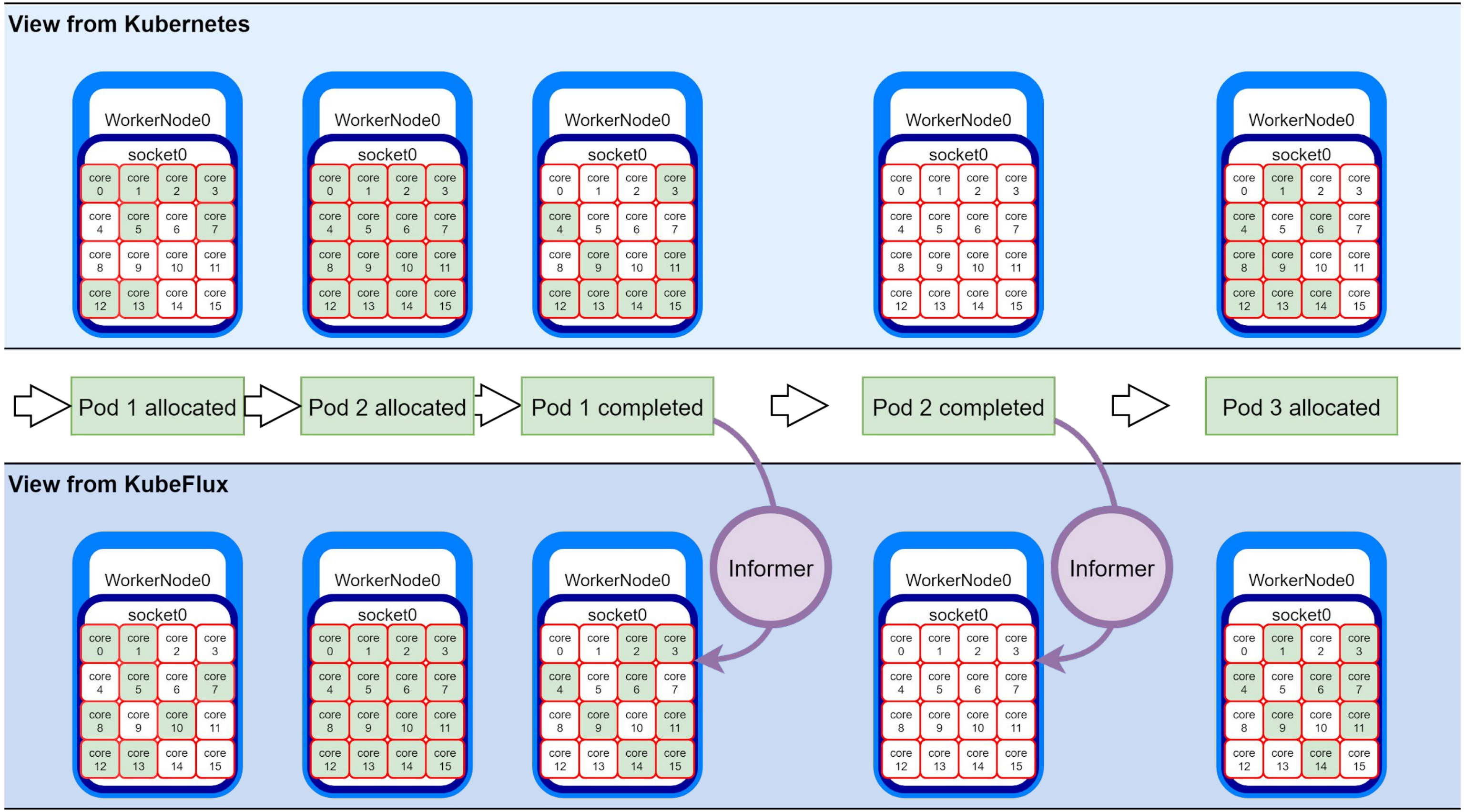


Problem demonstration

- Environment: A cluster running on local machine
  - 2 compute nodes
    - 1 Master node: Kubernetes control plane
    - 1 worker node
      - 16 virtual cpu cores
- Tools:
  - Kind
  - Docker
- Task:
  - Deploy 4 pods scheduled by KubeFlux, each pod requires 8 cpu cores
- Goal
  - Demonstrate the original KubeFlux can not reuse resources

Solution - Informer

Northeastern University
Khoury College of Computer Sciences

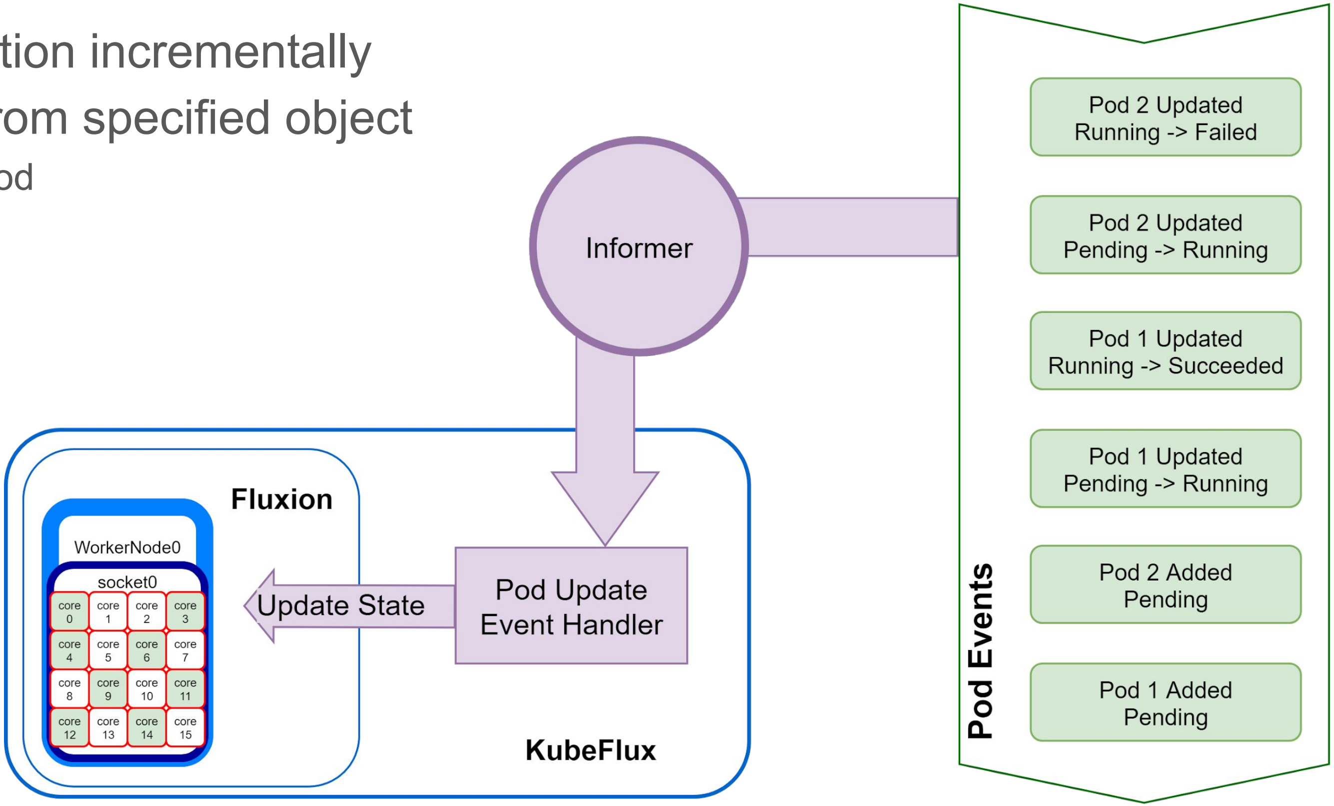


Cluster has 1 worker node with 1 cpu socket. It has 16 cores

Pod 1, 2 ,3 requires 8 cores

Closer look: Kubernetes Informer

- Update information incrementally
- Watch events from specified object
    - In our case: pod

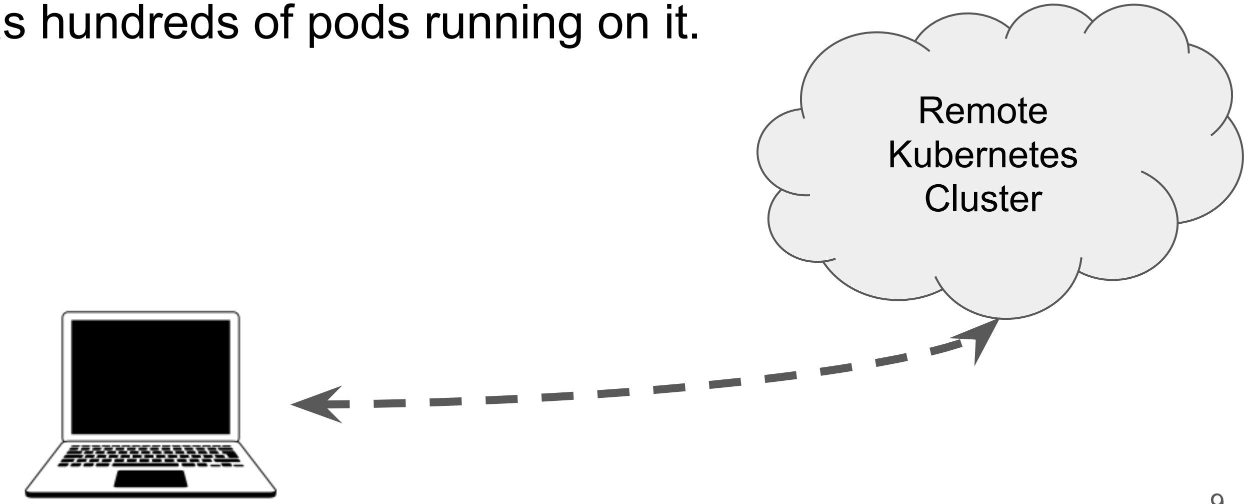


Informer Demo Setup

- Environment: Cluster running on local machine
  - 2 compute nodes
    - 1 Master node: Kubernetes control plane
    - 1 worker node
      - 16 virtual cores
- Tools:
  - Kind
  - Docker
- Task:
  - Deploy 4 pods scheduled by KubeFlux, each pod requires 8 cpu cores
- Goal
  - Demonstrate job cancellation feature with 4 succeeded pods
  - Demonstrate informer with 4 failed pods

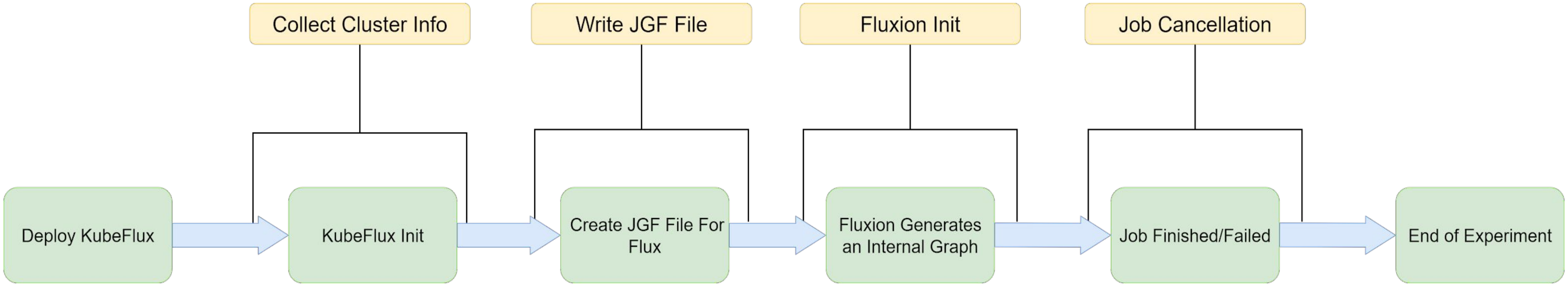
Experiment - KubeFlux Operations Time Consumption

- Deploy Pi calculation program on a remote cluster
  - Originally planned with the GROMACS application test, but it was broken
- The cluster has 8 accessible worker nodes
- Each worker node has: 4 cpu cores, 16 Gb memory
- The cluster already has hundreds of pods running on it.



Performance Metrics

- Collect Cluster Information
- Write JGF (Json Graph Format) File
- Fluxion Initialization
- Job Cancellation

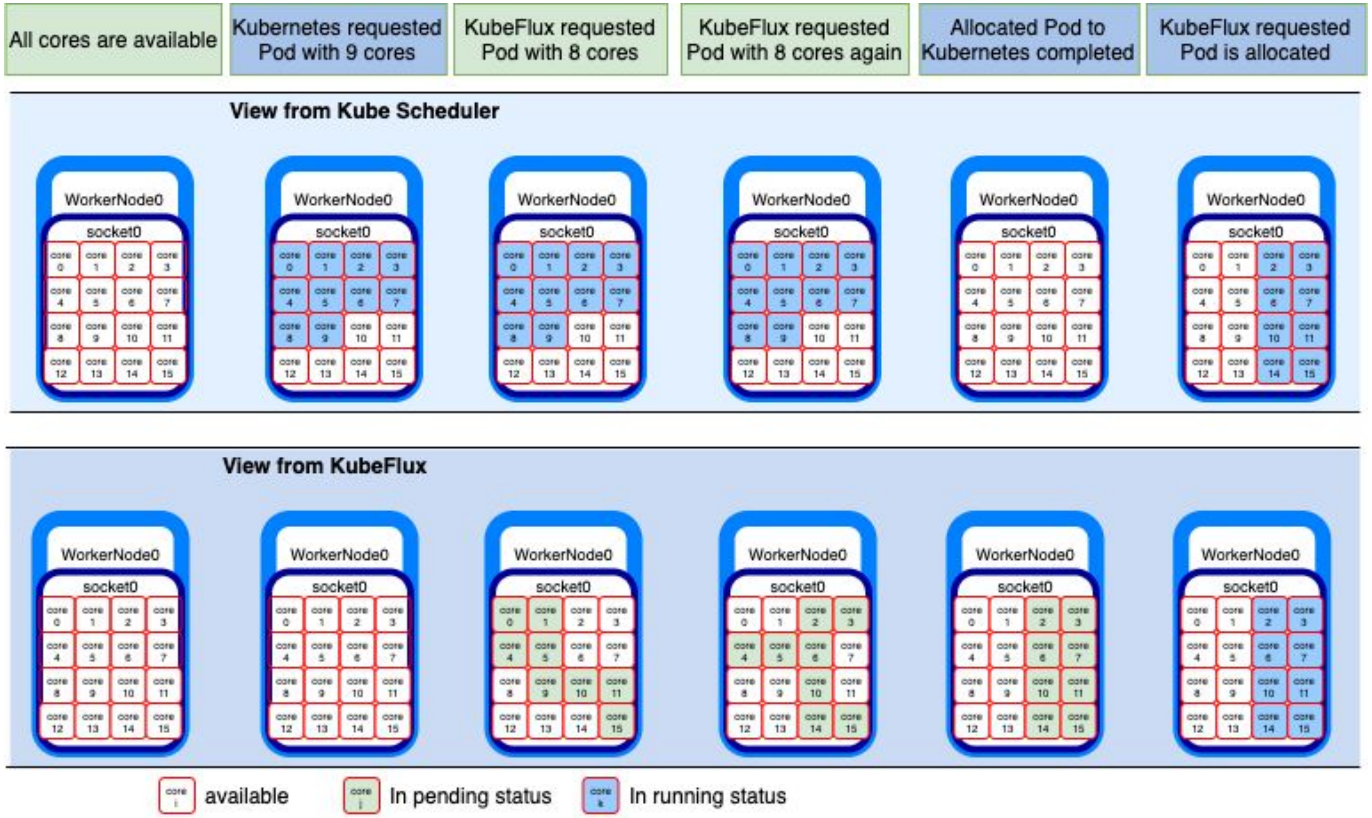


Experiment - Result

Overhead Measurements in Experiments (Unit: Second)					
No	Resource Graph Creation				Job Cancellation
	Collect Cluster Information	Write JGF File	Fluxion Initialization	Overall Time Consumption	
1	2.15588139	0.00185015	0.002944827	2.160676367	0.000160137
2	0.424259821	0.001822241	0.003250035	0.429332097	0.000161866
3	0.281901809	0.002125131	0.004601823	0.288628763	0.000138845
4	0.382038727	0.006396838	0.003113534	0.391549099	0.000142303
5	0.211644054	0.004054272	0.003107828	0.218806154	0.000151261
6	0.45694305	0.001359202	0.003051238	0.46135349	0.000130882
7	0.216843598	0.007000077	0.003323509	0.227167184	0.000113033

The challenge of Co-scheduling

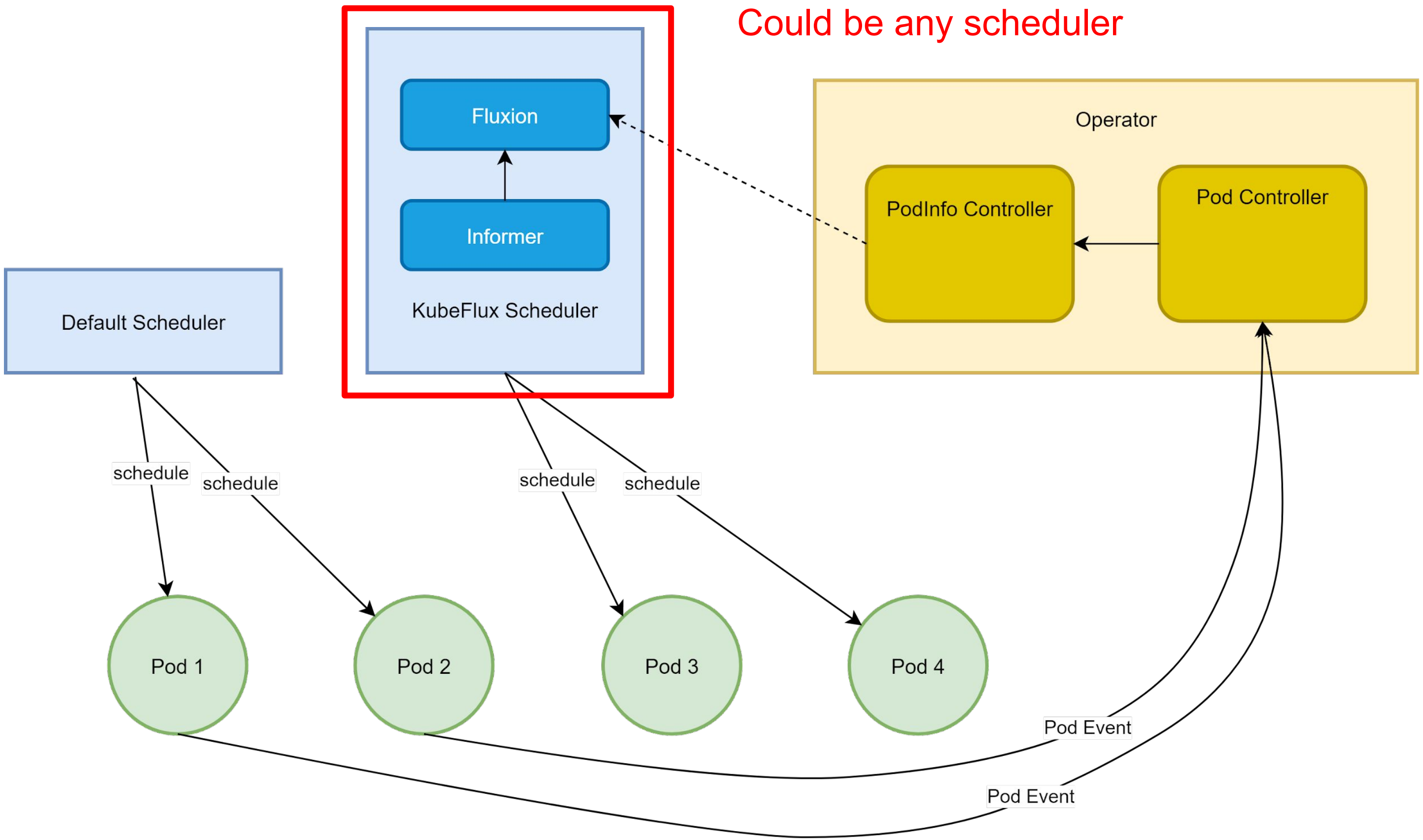
Northeastern University
Khoury College of Computer Sciences



Main conceptual challenges

- Abstraction: Enabling co-scheduling of any two scheduler plugins.
  - The solution should offer a representation of how it records changes
  - Schedulers wanting to make use of co-scheduling to become aware of other schedulers can translate the changes recorded by the operator and update their own internal state accordingly.
  - Each component participating in co-scheduling is specific to a certain level. The interfaces have to be prescriptive of how to implement co-scheduling but not too restrictive.

Potential Solution: Pod Controller

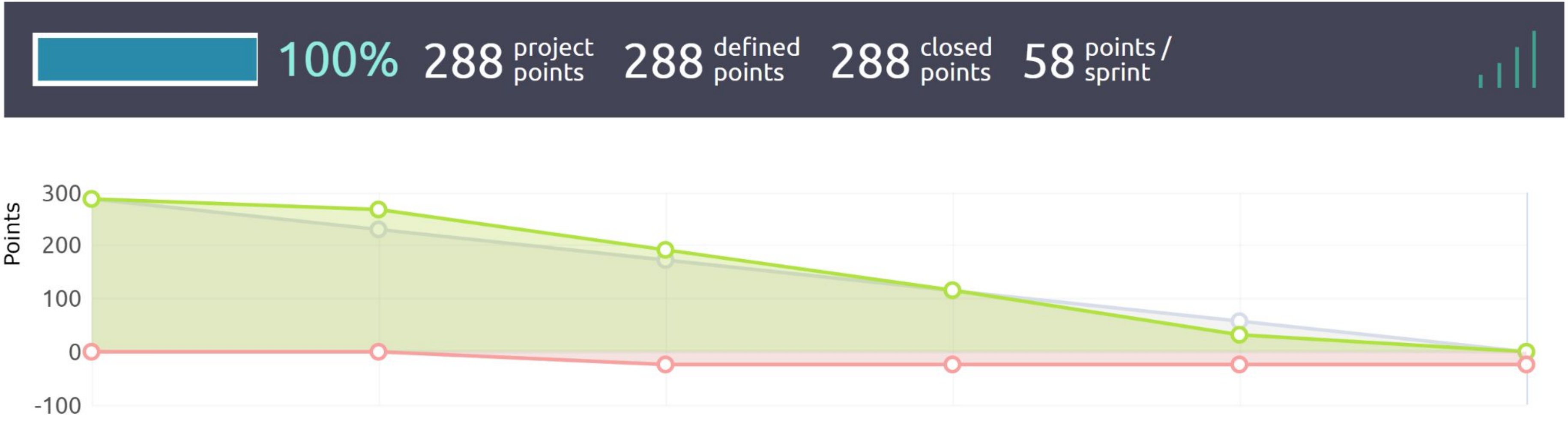


Main technical challenges

- Kubernetes has 3 different ways of specifying pod resource needs.
  - Priority 1: Guaranteed Pod
  - Priority 2: Burstable Pod
  - Priority 3: Best-Effort Pod
- Resources usage fluctuations
- Imperative HPC management philosophy doesn’t align with the declarative K8s management philosophy

Conclusion

- Studied about Kubernetes, Flux and Golang basics
- Implemented an informer, which is now part of the KubeFlux plugin
- Started the implementation of an operator
- Performed analysis on the time consumed by the informer



Future Scope

Extend the project to help KubeFlux become a competent HPC scheduling solution for Kubernetes:

- Designing a standard process for synchronizing a plug-in scheduler's internal state with the Kubernetes cluster state
    - Getting a proper figure of resource utilization from running pods (translating Kubernetes cpu measurements to tangible info)
    - Designing interface with a generic representation of the kubernetes cluster state that plug-in schedulers can access.

Thank You!