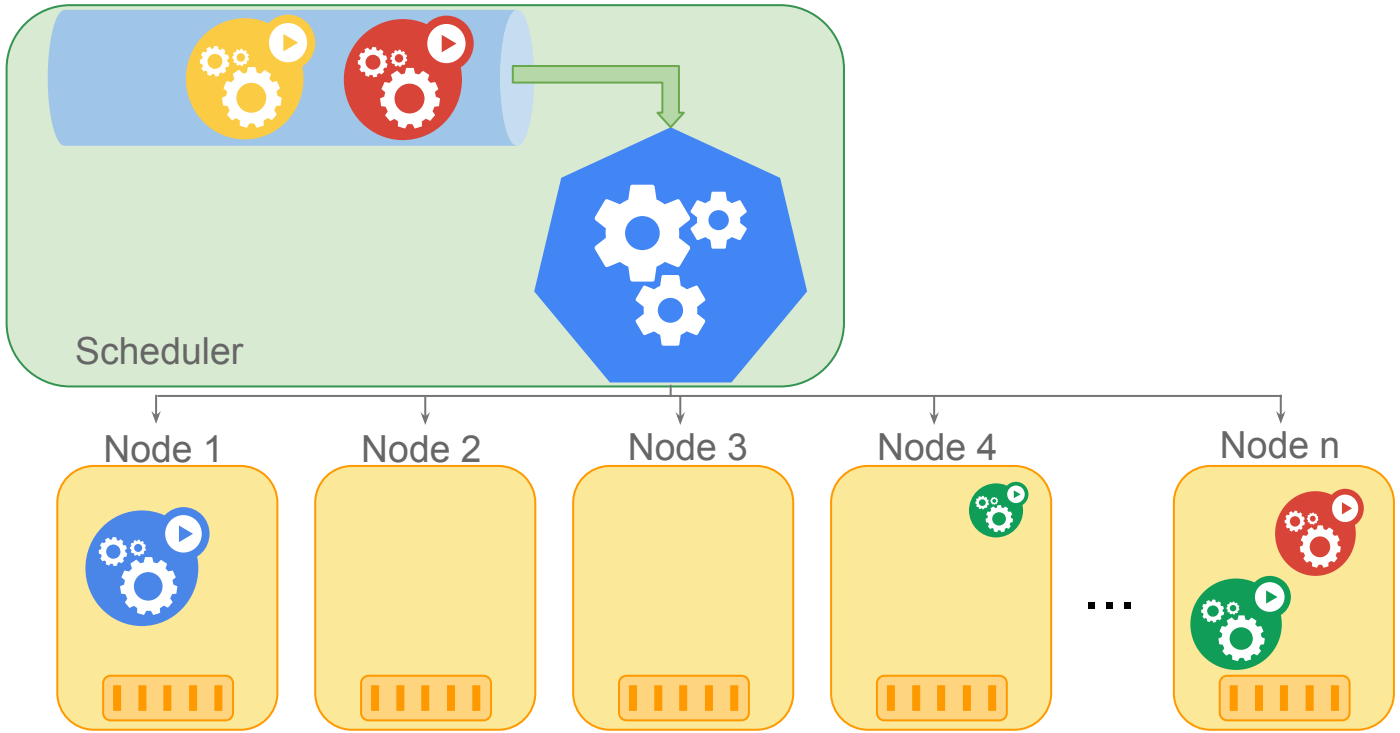
Deep Dive: Sig Scheduling

Abdullah Gharaibeh, Google

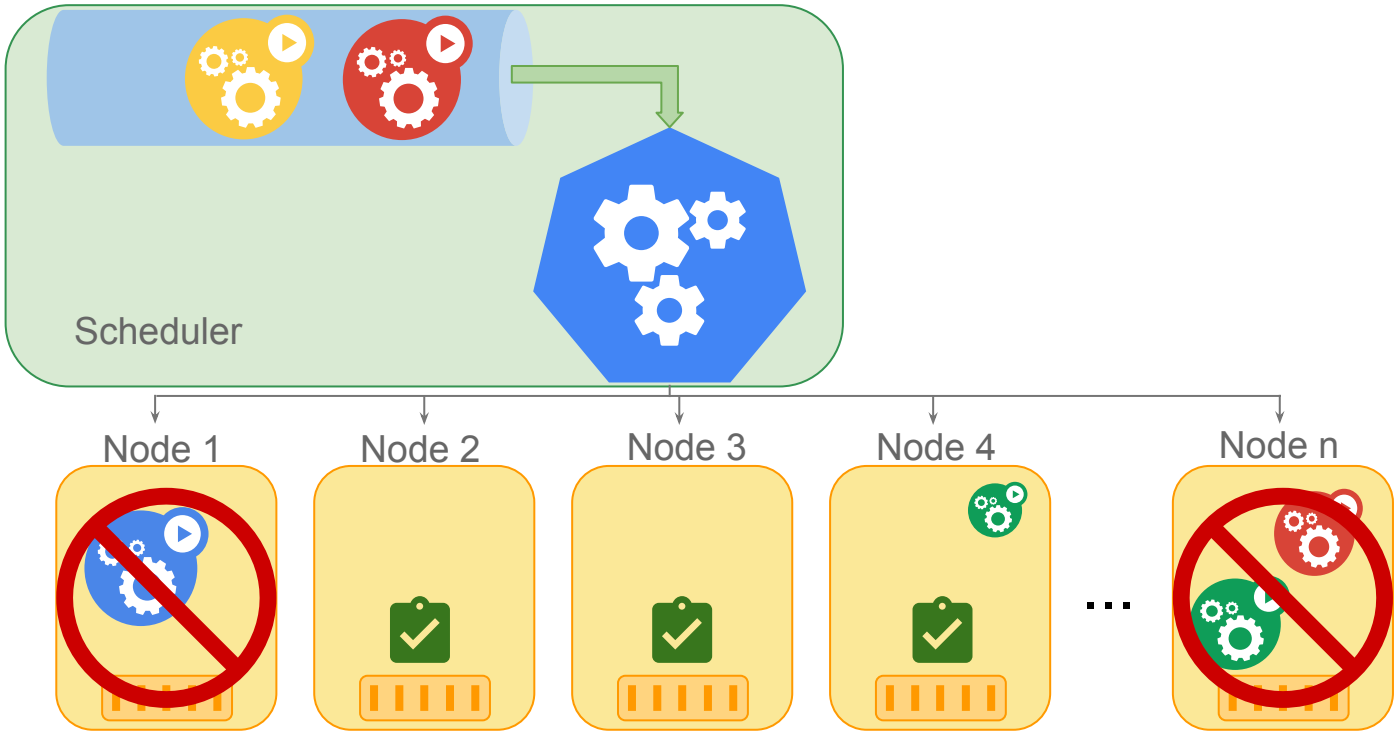
Introduction

Scheduler assigns Pods to Nodes



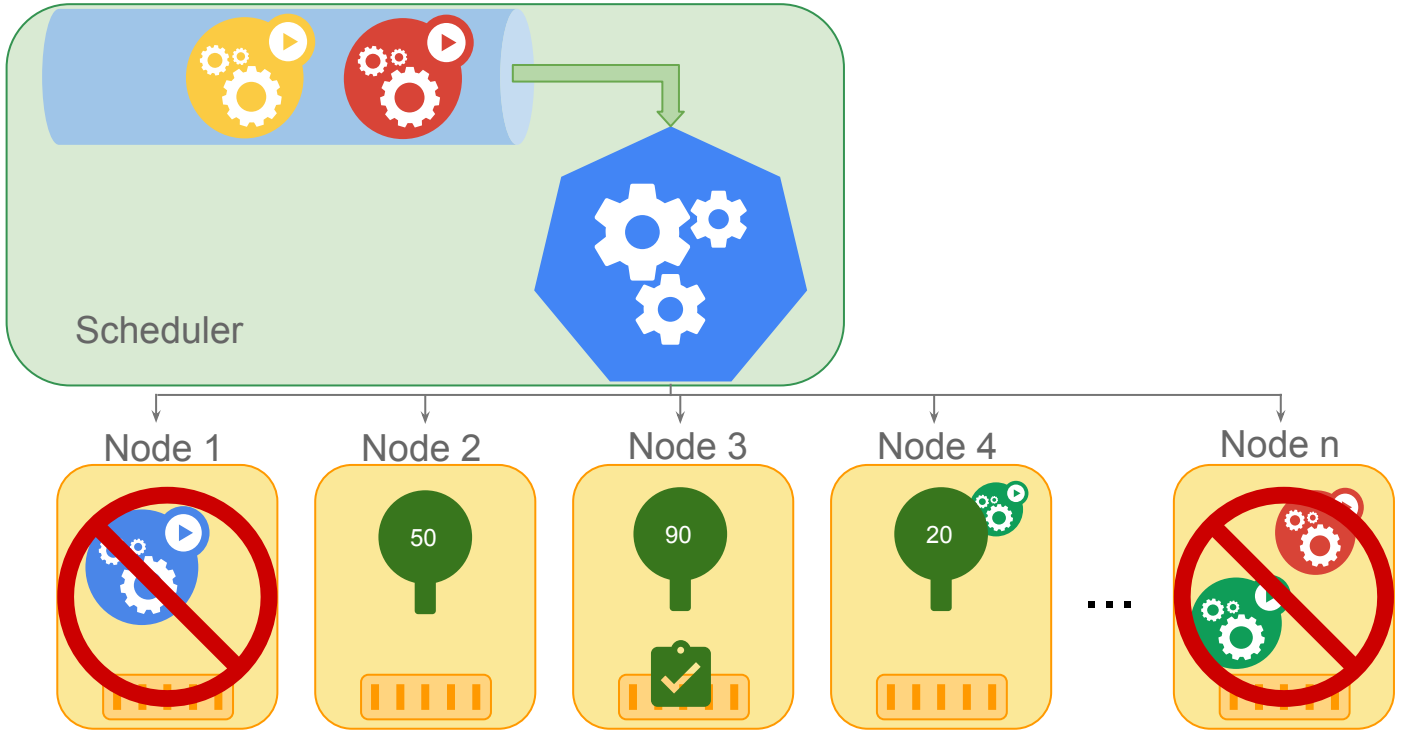
Introduction

Filters identify feasible Nodes



Introduction

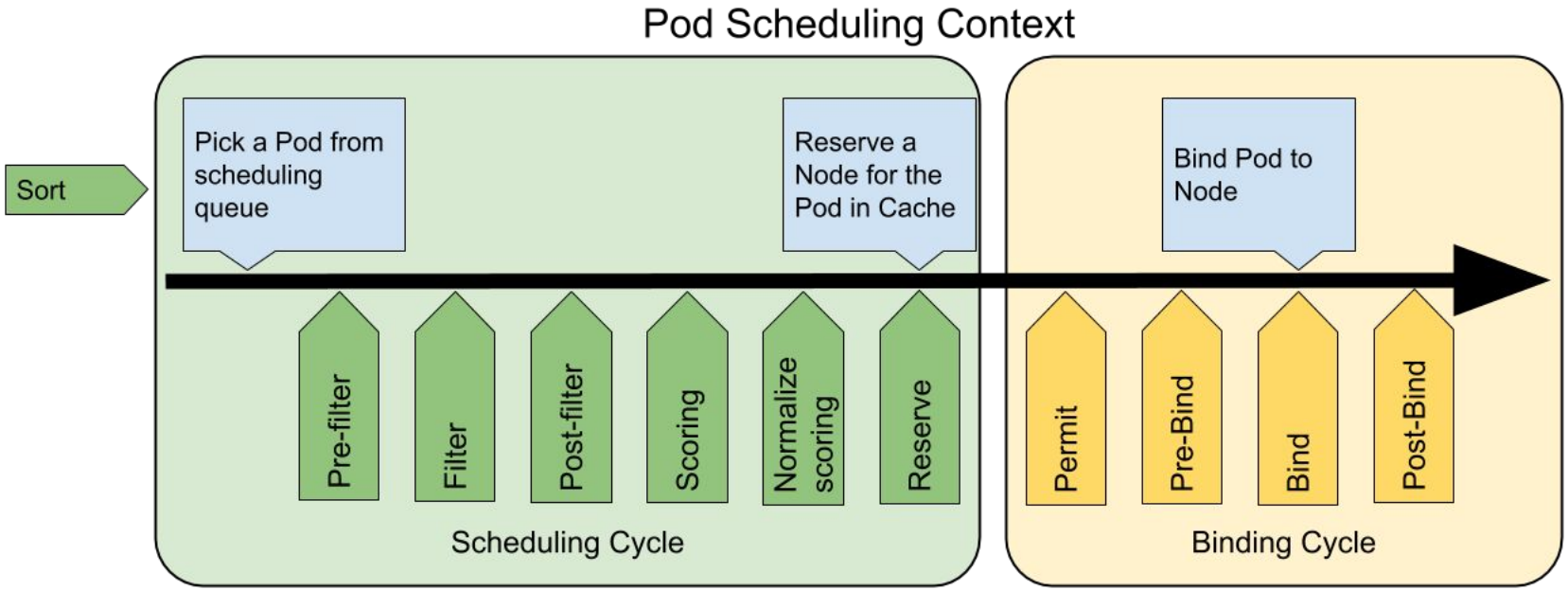
Score functions rank feasible Nodes



Recent Developments

Scheduling Framework

- Plugins add scheduling behaviors, they can be invoked at multiple extension points.
- ComponentConfig allows plugins to be enabled, disabled, and reordered.



Scheduling Framework

- Makes the k8s scheduler easier to extend and isolate features
  - A plugin corresponds to a feature and it can implement several extension points. In the past, a feature would be spread across different files
  - The core scheduler becomes simpler: run callbacks at pre-defined extension points in each execution cycle
- Custom schedulers don’t have to maintain patches to support custom algorithms
- Previously we had only “predicates” and “priorities”. New extension points allow implementing more complex features, for example gang-scheduling

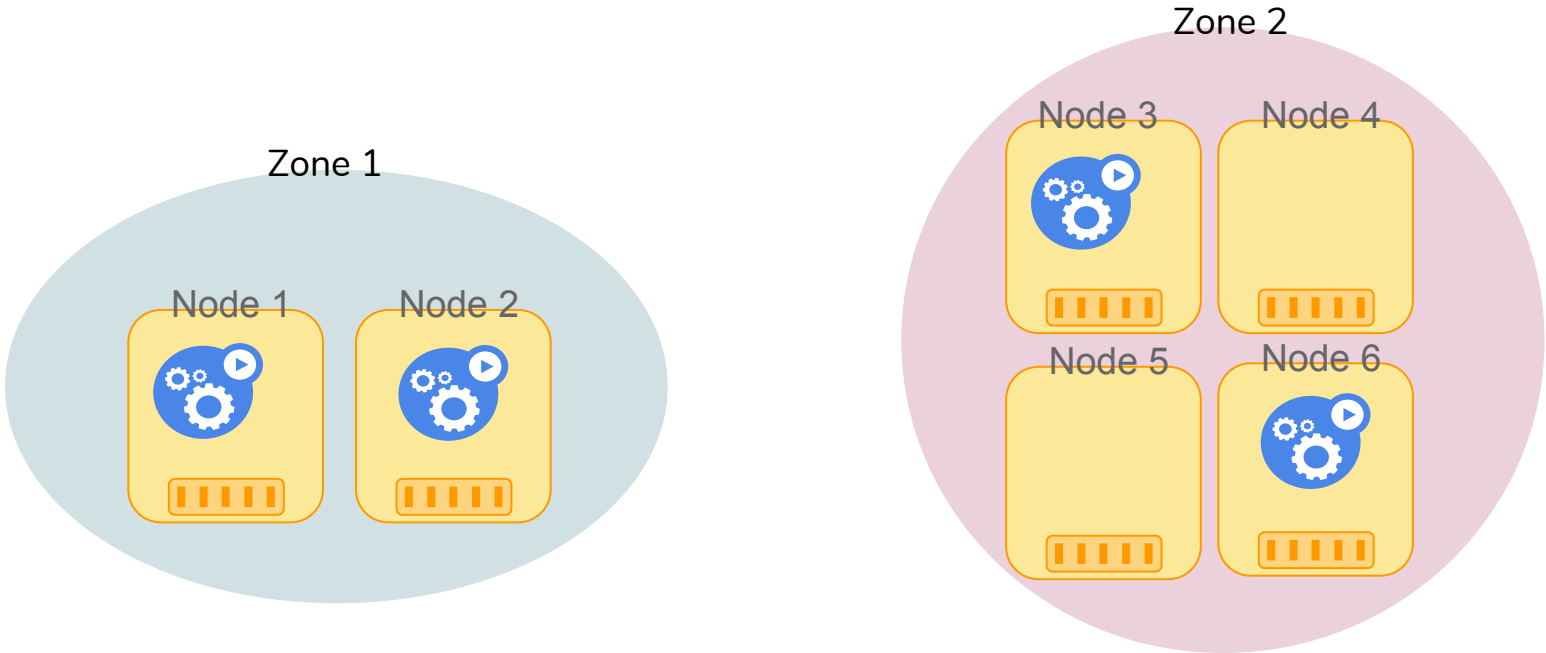
Scheduling Framework

go/scheduling-framework-migration

- Milestone 1 (Done): 1.17
  - Wrapped existing predicate and priorities functions in Plugins.
  - Translation layer from predicate/priority “policies” into Plugin configurations.
  - Create an interface for CA/Daemonset/Kubelet to call Filter plugins.
- Milestone 2 (Not started): 1.18
  - Move predicates and priorities code to run natively as plugins.
  - Clean up calls from core scheduler to predicates and priorities.
  - Declare Policy API deprecated, Plugins API in ComponentConfig is the replacement
- Milestone 3 (high-level idea): 1.19
  - Actually Deprecate Policy API and remove translation layer
  - Framework in GA

Pod Topology Spreading

- A Pod-level API that allows spreading pods in arbitrary topology domains
- Can be hard or soft requirement
- Alpha in 1.17



Improved Observability

- Latency
  - Scheduling latency per pod
  - Latency breakdown of each scheduling step/plugin
- Traffic
  - Incoming pods per second
  - Scheduling attempts per second
- Saturation
  - Binding goroutines
  - Cache size (pods and nodes in scheduler cache)

Graduated to GA in 1.17

- Schedule DaemonSet Pods
- Taint Nodes by Condition

Planned Features

Pod Overhead

- Problem
  - Pods have non-negligible resource overhead not accurately accounted for
  - The current approach reserves pre-defined amount of resources on the node for system components, but ignores per-pod overhead (like pause container)
  - With sandbox pods, the pod overhead potentially becomes much larger and can't be ignored (e.g., Kata agent, gVisor sentry)

- Solution
  - Augment the RuntimeClass definition and the PodSpec to introduce the field `Overhead *ResourceList`
  - Scheduler, Kubelet and resource quota management takes this overhead into account

In-place Update of Pod Resources

- Problem
  - Currently, changing resource allocation requires the Pod to be recreated since the PodSpec's Container Resources is immutable
  - This is disruptive when trying to vertically scale the Pod, especially for stateful workloads, or ones with lower replica count

- Solution
  - Make PodSpec mutable with regards to Resources
  - Practically, Kubelet will decide whether or not the change is accepted
  - If accepted, the Scheduler also accounts for the increased in resource usage of the node

Questions