Device Plugins 2.0: How to Build a Driver for Dynamic Resource Allocation (DRA)

Kevin Klues (NVIDIA), Alexey Fomenko (Intel)

What is Dynamic Resource Allocation?

- New way of requesting access to resources available in Kubernetes 1.26+
- Provides an alternative to the “count-based” interface of e.g. `nvidia.com/gpu: 2`
- Puts full control of the API to request resources in the hands of 3rd-party developers
- Generalization of the **`PersistentVolume`** API for all types of resources
- Key concepts:

  `ResourceClass` → **`ClassParameters`**

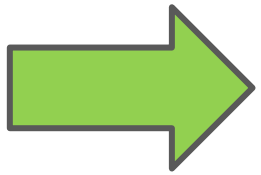
  `ResourceClaim(Template)` → **`ClaimParameters`**

What is Dynamic Resource Allocation?


```yaml
apiVersion: v1
kind: Pod
metadata:
  name: gpu-example
spec:
  containers:
  - name: ctr
    image: nvidia/cuda
    command: ["nvidia-smi", "-L"]
    resources:
      limits:
        nvidia.com/gpu: 1
```

What is Dynamic Resource Allocation?


```yaml
apiVersion: v1
kind: Pod
metadata:
  name: gpu-example
spec:
  containers:
  - name: ctr
    image: nvidia/cuda
    command: ["nvidia-smi", "-L"]
    resources:
      limits:
        nvidia.com/gpu: 1
```

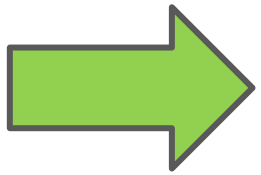


```yaml
---
apiVersion: resource.k8s.io/v1alpha2
kind: ResourceClaimTemplate
metadata:
  name: gpu-template
spec:
  spec:
    resourceClassName: gpu.nvidia.com

---
apiVersion: v1
kind: Pod
metadata:
  name: gpu-example
spec:
  containers:
  - name: ctr
    image: nvidia/cuda
    command: ["nvidia-smi" "-L"]
    resources:
      claims:
      - name: gpu
  resourceClaims:
  - name: gpu
    source:
      resourceClaimTemplateName: gpu-template
```

What is Dynamic Resource Allocation?


```yaml
apiVersion: v1
kind: Pod
metadata:
  name: gpu-example
spec:
  containers:
  - name: ctr
    image: nvidia/cuda
    command: ["nvidia-smi", "-L"]
    resources:
      limits:
        nvidia.com/gpu: 1
```

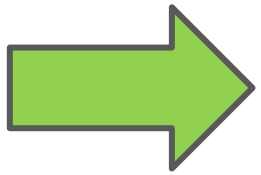


```yaml
---
apiVersion: resource.k8s.io/v1alpha2
kind: ResourceClaimTemplate
metadata:
  name: gpu-template
spec:
  spec:
    resourceClassName: gpu.nvidia.com

---
apiVersion: v1
kind: Pod
metadata:
  name: gpu-example
spec:
  containers:
  - name: ctr
    image: nvidia/cuda
    command: ["nvidia-smi" "-L"]
    resources:
      claims:
      - name: gpu
  resourceClaims:
  - name: gpu
    source:
      resourceClaimTemplateName: gpu-template
```

What is Dynamic Resource Allocation?


```yaml
apiVersion: v1
kind: Pod
metadata:
  name: gpu-example
spec:
  containers:
  - name: ctr
    image: nvidia/cuda
    command: ["nvidia-smi", "-L"]
    resources:
      limits:
        nvidia.com/gpu: 1
```

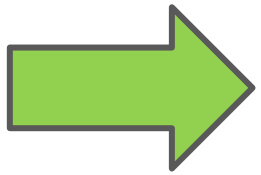


```yaml
---
apiVersion: resource.k8s.io/v1alpha2
kind: ResourceClaimTemplate
metadata:
  name: gpu-template
spec:
  spec:
    resourceClassName: gpu.nivida.com

---
apiVersion: v1
kind: Pod
metadata:
  name: gpu-example
spec:
  containers:
  - name: ctr
    image: nvidia/cuda
    command: ["nvidia-smi" "-L"]
    resources:
      claims:
      - name: gpu
  resourceClaims:
  - name: gpu
    source:
      resourceClaimTemplateName: gpu-template
```

What is Dynamic Resource Allocation?


```yaml
apiVersion: v1
kind: Pod
metadata:
  name: gpu-example
spec:
  containers:
  - name: ctr
    image: nvidia/cuda
    command: ["nvidia-smi", "-L"]
    resources:
      limits:
        nvidia.com/gpu: 1
```



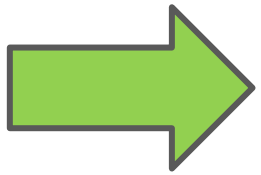
```yaml
---
apiVersion: resource.k8s.io/v1alpha2
kind: ResourceClaimTemplate
metadata:
  name: gpu-template
spec:
  spec:
    resourceClassName: gpu.nvidia.com

---
apiVersion: v1
kind: Pod
metadata:
  name: gpu-example
spec:
  containers:
  - name: ctr
    image: nvidia/cuda
    command: ["nvidia-smi" "-L"]
    resources:
      claims:
      - name: gpu
  resourceClaims:
  - name: gpu
    source:
      resourceClaimTemplateName: gpu-template
```

Associated with the DRA Driver and installed by the cluster admin

What is Dynamic Resource Allocation?


```yaml
apiVersion: v1
kind: Pod
metadata:
  name: gpu-example
spec:
  containers:
  - name: ctr
    image: nvidia/cuda
    command: ["nvidia-smi", "-L"]
    resources:
      limits:
        nvidia.com/gpu: 1
```



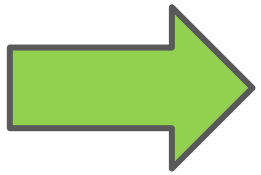
```yaml
---
apiVersion: resource.k8s.io/v1alpha2
kind: ResourceClaimTemplate
metadata:
  name: gpu-template
spec:
  spec:
    resourceClassName: gpu.nvidia.com

---
apiVersion: v1
kind: Pod
metadata:
  name: gpu-example
spec:
  containers:
  - name: ctr
    image: nvidia/cuda
    command: ["nvidia-smi" "-L"]
    resources:
      claims:
      - name: gpu
  resourceClaims:
  - name: gpu
    source:
      resourceClaimTemplateName: gpu-template
```

Associated with the DRA Driver and installed by the cluster admin

What is Dynamic Resource Allocation?


```yaml
apiVersion: v1
kind: Pod
metadata:
  name: gpu-example
spec:
  containers:
  - name: ctr
    image: nvidia/cuda
    command: ["nvidia-smi", "-L"]
    resources:
      limits:
        nvidia.com/gpu: 1
```



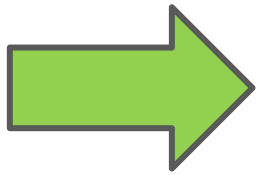
```yaml
---
apiVersion: resource.k8s.io/v1alpha2
kind: ResourceClaimTemplate
metadata:
  name: gpu-template
spec:
  spec:
    resourceClassName: gpu.nvidia.com

---
apiVersion: v1
kind: Pod
metadata:
  name: gpu-example
spec:
  containers:
  - name: ctr
    image: nvidia/cuda
    command: ["nvidia-smi" "-L"]
    resources:
      claims:
      - name: gpu
  resourceClaims:
  - name: gpu
    source:
      resourceClaimTemplateName: gpu-template
```

Associated with the DRA Driver and installed by the cluster admin

What is Dynamic Resource Allocation?


```yaml
apiVersion: v1
kind: Pod
metadata:
  name: gpu-example
spec:
  containers:
  - name: ctr
    image: nvidia/cuda
    command: ["nvidia-smi", "-L"]
    resources:
      limits:
        nvidia.com/gpu: 1
```



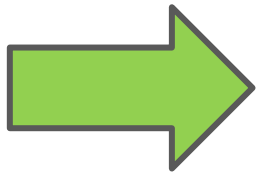
```yaml
---
apiVersion: resource.k8s.io/v1alpha2
kind: ResourceClaimTemplate
metadata:
  name: gpu-template
spec:
  spec:
    resourceClassName: gpu.nvidia.com

---
apiVersion: v1
kind: Pod
metadata:
  name: gpu-example
spec:
  containers:
  - name: ctr
    image: nvidia/cuda
    command: ["nvidia-smi" "-L"]
    resources:
      claims:
      - name: gpu
  resourceClaims:
  - name: gpu
    source:
      resourceClaimTemplateName: gpu-template
```

Associated with the DRA Driver and installed by the cluster admin

What is Dynamic Resource Allocation?


```yaml
apiVersion: v1
kind: Pod
metadata:
  name: gpu-example
spec:
  containers:
  - name: ctr
    image: nvidia/cuda
    command: ["nvidia-smi", "-L"]
    resources:
      limits:
        nvidia.com/gpu: 2
```

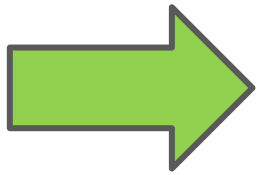


```yaml
---
apiVersion: resource.k8s.io/v1alpha2
kind: ResourceClaimTemplate
metadata:
  name: gpu-template
spec:
  spec:
    resourceClassName: gpu.nvidia.com

---
apiVersion: v1
kind: Pod
metadata:
  name: gpu-example
spec:
  containers:
  - name: ctr
    image: nvidia/cuda
    command: ["nvidia-smi" "-L"]
    resources:
      claims:
      - name: gpu0
      - name: gpu1
  resourceClaims:
  - name: gpu0
    source:
      resourceClaimTemplateName: gpu-template
  - name: gpu1
    source:
      resourceClaimTemplateName: gpu-template
```

What is Dynamic Resource Allocation?


```yaml
apiVersion: v1
kind: Pod
metadata:
  name: gpu-example{0,1,2}
spec:
  containers:
  - name: ctr
    image: nvidia/cuda
    command: ["nvidia-smi", "-L"]
    resources:
      limits:
        nvidia.com/gpu: 2
```



```yaml
apiVersion: resource.k8s.io/v1alpha2
kind: ResourceClaimTemplate
metadata:
  name: gpu-template
spec:
  spec:
    resourceClassName: gpu.nvidia.com
```

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: gpu-example{0,1,2}
spec:
  containers:
  - name: ctr
    image: nvidia/cuda
    command: ["nvidia-smi" "-L"]
    resources:
      claims:
      - name: gpu0
      - name: gpu1
  resourceClaims:
  - name: gpu0
    source:
      resourceClaimTemplateName: gpu-template
  - name: gpu1
    source:
      resourceClaimTemplateName: gpu-template
```

What is Dynamic Resource Allocation?

- **`ResourceClass`**
  - Associates a named resource with its corresponding resource driver
  - Created by a sys-admin to define the set of allocatable resources

```
apiVersion: resource.k8s.io/v1alpha2
kind: ResourceClass
metadata:
  name: gpu.nvidia.com
driverName: gpu.resource.nvidia.com
```

What is Dynamic Resource Allocation?

- **`ResourceClass`**
  - Associates a named resource with its corresponding resource driver
  - Created by a sys-admin to define the set of allocatable resources
  - Can include optional set of **`ClassParameters`** with custom API

```
apiVersion: resource.k8s.io/v1alpha2
kind: ResourceClass
metadata:
  name: gpu.nvidia.com
driverName: gpu.resource.nvidia.com
parametersRef:
    apiGroup: <api-group>
    kind: <class-parameters-kind>
    name: <class-parameters-name>
```

What is Dynamic Resource Allocation?

- **`ResourceClass`**
  - Associates a named resource with its corresponding resource driver
  - Created by a sys-admin to define the set of allocatable resources
  - Can include optional set of **`ClassParameters`** with custom API

```yaml
apiVersion: resource.k8s.io/v1alpha2
kind: ResourceClass
metadata:
  name: gpu.nvidia.com
driverName: gpu.resource.nvidia.com
parametersRef:
    apiGroup: gpu.resource.nvidia.com
    kind: GpuClassParameters
    name: non-sharable
```

```yaml
apiVersion: gpu.resource.nvidia.com/v1alpha1
kind: GpuClassParameters
metadata:
  name: non-sharable
spec:
  shareable: false
```

What is Dynamic Resource Allocation?

- **`ResourceClass`**
  - Associates a named resource with its corresponding resource driver
  - Created by a sys-admin to define the set of allocatable resources
  - Can include optional set of **`ClassParameters`** with custom API

```yaml
apiVersion: resource.k8s.io/v1alpha2
kind: ResourceClass
metadata:
  name: t4.nvidia.com
driverName: gpu.resource.nvidia.com
parametersRef:
    apiGroup: gpu.resource.nvidia.com
    kind: GpuClassParameters
    name: only-t4-gpus
```

```yaml
apiVersion: gpu.resource.nvidia.com/v1alpha1
kind: GpuClassParameters
metadata:
  name: only-t4-gpus
spec:
  selector:
    - productName: "*t4*"
```

What is Dynamic Resource Allocation?

- **`ResourceClaim(Template)`**
  - Represents an actual resource allocation to be made by a resource driver
  - Created by an end-user

```
apiVersion: resource.k8s.io/v1alpha2
kind: ResourceClaimTemplate
metadata:
  name: unique-gpu
spec:
  spec:
    resourceClassName: gpu.nvidia.com
```

`ResourceClaimTemplates` create a *new* **`ResourceClaim`** each time they are referenced (e.g. a unique GPU for each reference)

```
apiVersion: resource.k8s.io/v1alpha2
kind: ResourceClaim
metadata:
  name: shared-gpu
spec:
  resourceClassName: gpu.nvidia.com
```

`ResourceClaims` refer to the *exact* same object each time they are referenced (e.g. the *same* GPU for each reference)

What is Dynamic Resource Allocation?

- **`ResourceClaim(Template)`**
  - Represents an actual resource allocation to be made by a resource driver
  - Created by an end-user
  - Can include optional set of **`ClaimParameters`** with custom API

```yaml
apiVersion: resource.k8s.io/v1alpha2
kind: ResourceClaim
metadata:
  name: shared-gpu
spec:
  resourceClassName: gpu.nvidia.com
  parametersRef:
    apiGroup: <api-group>
    kind: <claim-parameters-kind>
    name: <claim-parameters-name>
```

What is Dynamic Resource Allocation?

- **`ResourceClaim(Template)`**
  - Represents an actual resource allocation to be made by a resource driver
  - Created by an end-user
  - Can include optional set of **`ClaimParameters`** with custom API

```yaml
apiVersion: resource.k8s.io/v1alpha2
kind: ResourceClaim
metadata:
  name: shared-gpu
spec:
  resourceClassName: gpu.nvidia.com
  parametersRef:
    apiGroup: gpu.resource.nvidia.com
    kind: GpuClaimParameters
    name: t4-or-16gb-v100-mps-shared
```

```yaml
apiVersion:
gpu.resource.nvidia.com/v1alpha1
kind: GpuClaimParameters
metadata:
  name: t4-or-16gb-v100-mps-shared
spec:
  count: 1
  selector:
    orExpression:
    - productName: "*t4*"
    - andExpression:
      - productName: "*v100*"
      - memorySize:
          value: 16Gi
          operator: LessThanOrEqualTo
  sharing:
    strategy: MPS
```

What is Dynamic Resource Allocation?

- **`ResourceClaim(Template)`**
  - Represents an actual resource allocation to be made by a resource driver
  - Created by an end-user
  - Can include optional set of **`ClaimParameters`** with custom API

```yaml
apiVersion: resource.k8s.io/v1alpha2
kind: ResourceClaim
metadata:
  name: shared-gpu
spec:
  resourceClassName: gpu.nvidia.com
  parametersRef:
    apiGroup: gpu.resource.nvidia.com
    kind: GpuClaimParameters
    name: t4-or-16gb-v100-mps-shared
```

```yaml
apiVersion:
gpu.resource.nvidia.com/v1alpha1
kind: GpuClaimParameters
metadata:
  name: t4-or-16gb-v100-mps-shared
spec:
  count: 1
  selector:
    orExpression:
    - productName: "*t4*"
    - andExpression:
      - productName: "*v100*"
      - memorySize:
          value: 16Gi
          operator: LessThanOrEqualTo
  sharing:
    strategy: MPS
```

What is Dynamic Resource Allocation?


```yaml
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
    source:
      resourceClaimName: shared-gpu
```

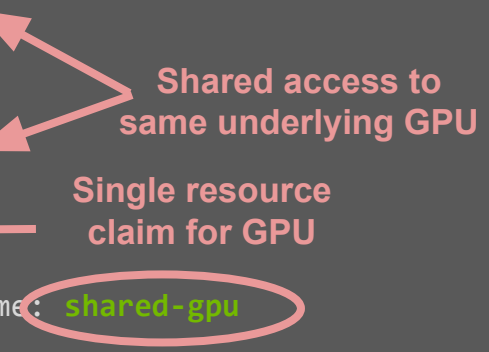
What is Dynamic Resource Allocation?


```
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
    source:
      resourceClaimName: shared-gpu
```

Single resource claim for GPU

What is Dynamic Resource Allocation?


```
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
    source:
      resourceClaimName: shared-gpu
```



What is Dynamic Resource Allocation?


```yaml
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
    source:
      resourceClaimName: shared-gpu
```

Shared access to same underlying GPU

Single resource claim for GPU

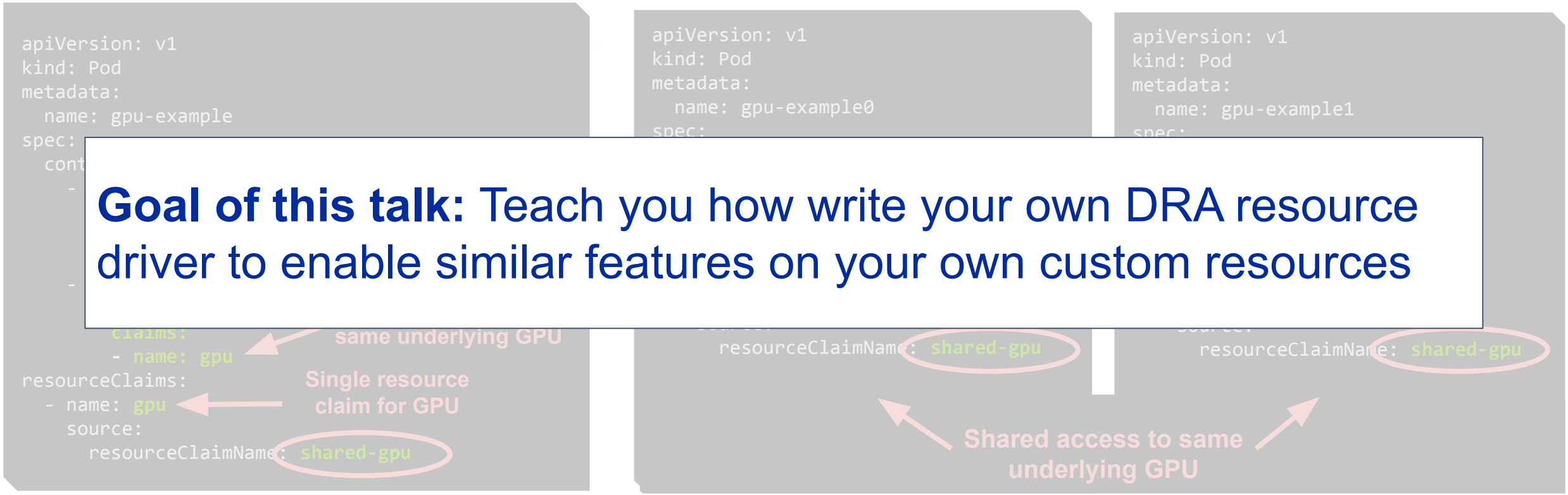
```yaml
apiVersion: v1
kind: Pod
metadata:
  name: gpu-example0
spec:
  containers:
    - name: ctr
      resources:
        claims:
        - name:gpu
resourceClaims:
  - name: gpu
    source:
      resourceClaimName: shared-gpu
```

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

Shared access to same underlying GPU

What is Dynamic Resource Allocation?

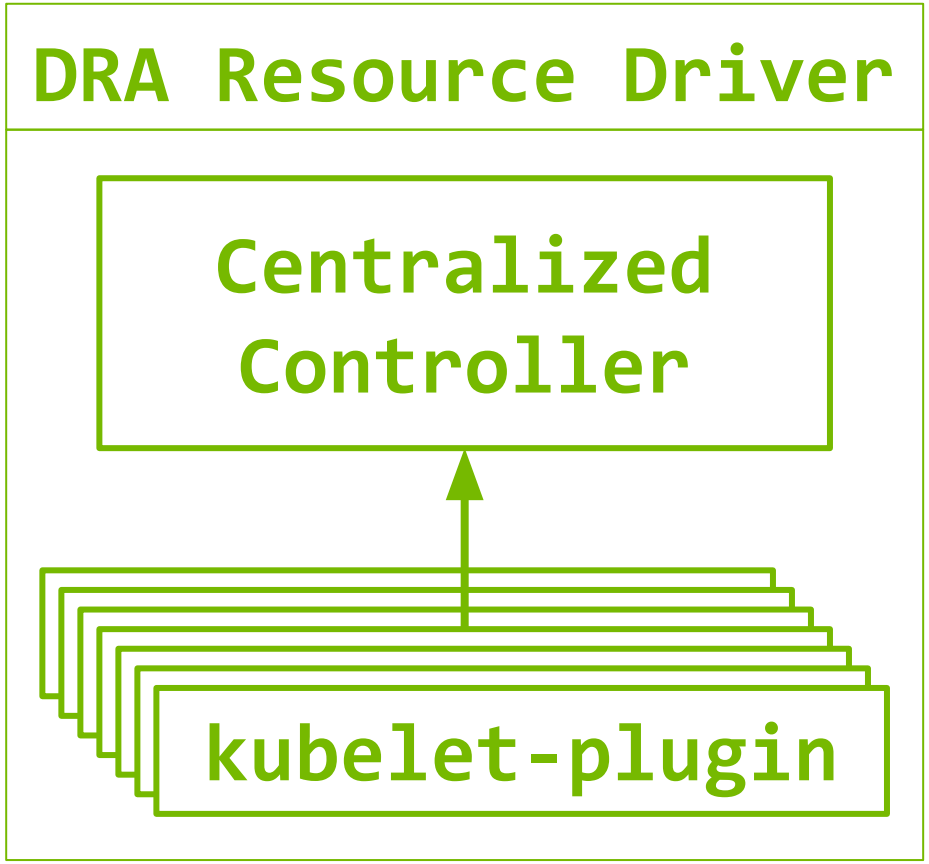
Goal of this talk: Teach you how write your own DRA resource driver to enable similar features on your own custom resources



Outline

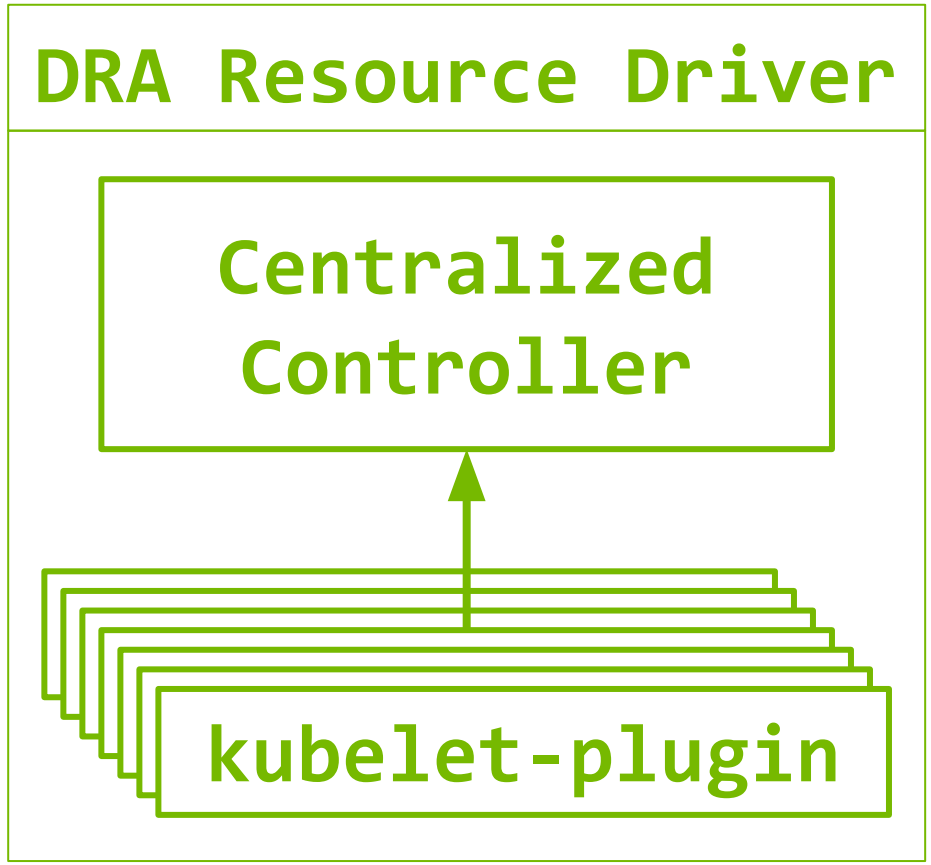
- Anatomy of a DRA Resource Driver
- Allocating Resources with DRA
- How to Build Your Own DRA Resource Driver
- New and Upcoming Features
- Demo on NVIDIA GPUs
- Demo on Intel GPUs

Anatomy of a DRA Resource Driver



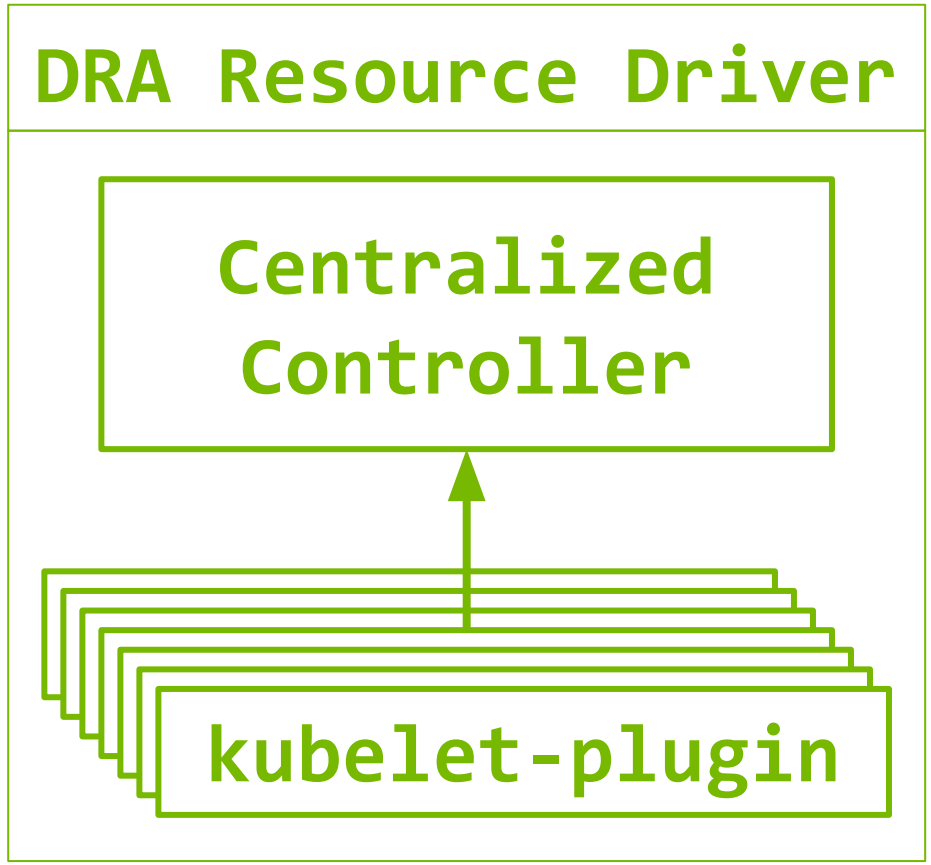
- Two separate but coordinating components:
  - A centralized controller running with high-availability
  - A node-local kubelet plugin running as a daemonset

Anatomy of a DRA Resource Driver



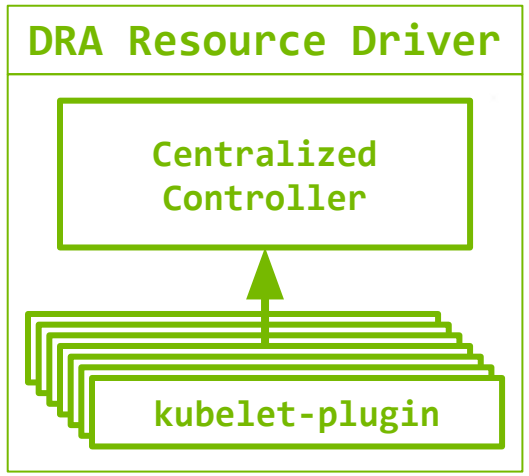
- Two separate but coordinating components:
  - A centralized controller running with high-availability
  - A node-local kubelet plugin running as a daemonset
- The centralized controller serves to:
  - Coordinate with the K8s scheduler to decide which nodes an incoming `ResourceClaim` can be serviced on
  - Perform the actual `ResourceClaim` allocation once the scheduler picks a node to allocate it on
  - Perform deallocation of a `ResourceClaim` once deleted

Anatomy of a DRA Resource Driver

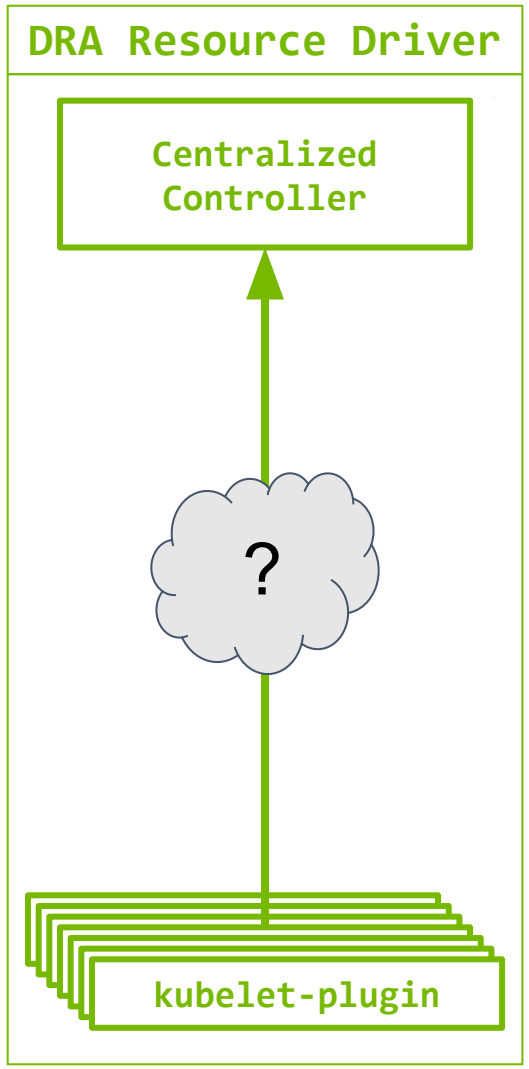


- Two separate but coordinating components:
  - A centralized controller running with high-availability
  - A node-local kubelet plugin running as a daemonset
- The centralized controller serves to:
  - Coordinate with the K8s scheduler to decide which nodes an incoming **`ResourceClaim`** can be serviced on
  - Perform the actual **`ResourceClaim`** allocation once the scheduler picks a node to allocate it on
  - Perform deallocation of a **`ResourceClaim`** once deleted
- The node-local kubelet plugin serves to:
  - Advertise any node-local state required by the centralized controller to make its allocation decisions
  - Perform any node-local operations required as part of preparing and unpreparing a **`ResourceClaim`** on a node
  - Pass the set of devices associated with a prepared **`ResourceClaim`** to the kubelet so it can forward them to the underlying container runtime

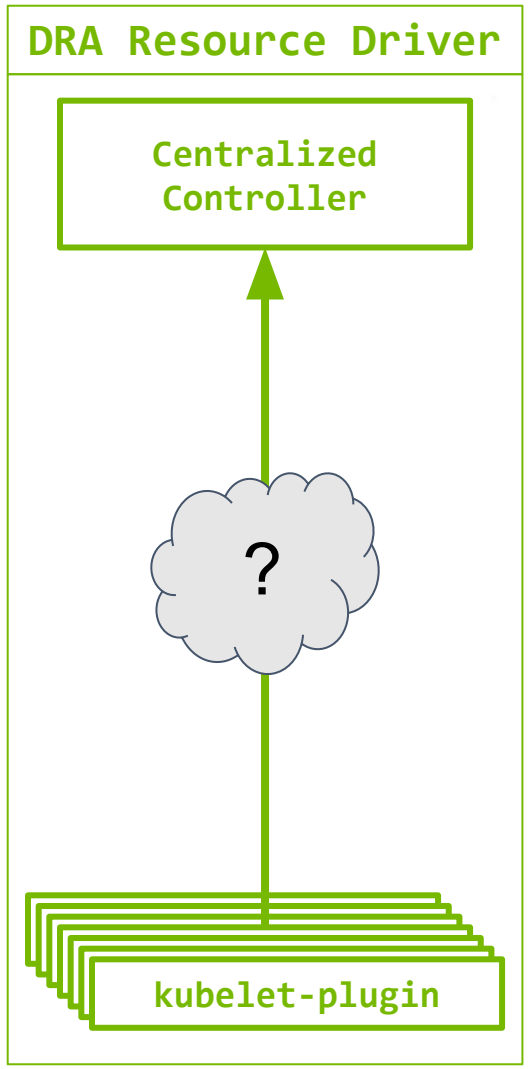
Anatomy of a DRA Resource Driver



Anatomy of a DRA Resource Driver

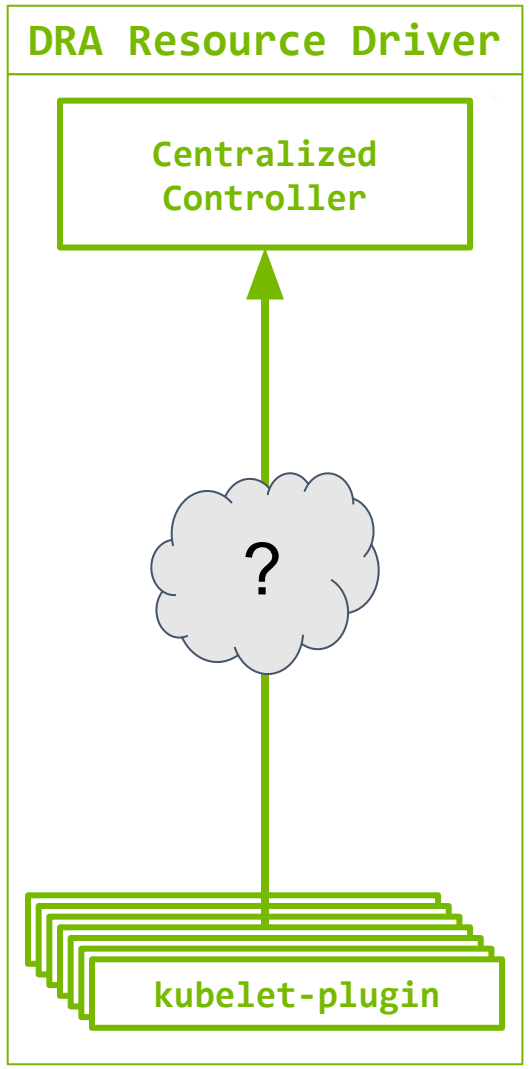


Anatomy of a DRA Resource Driver



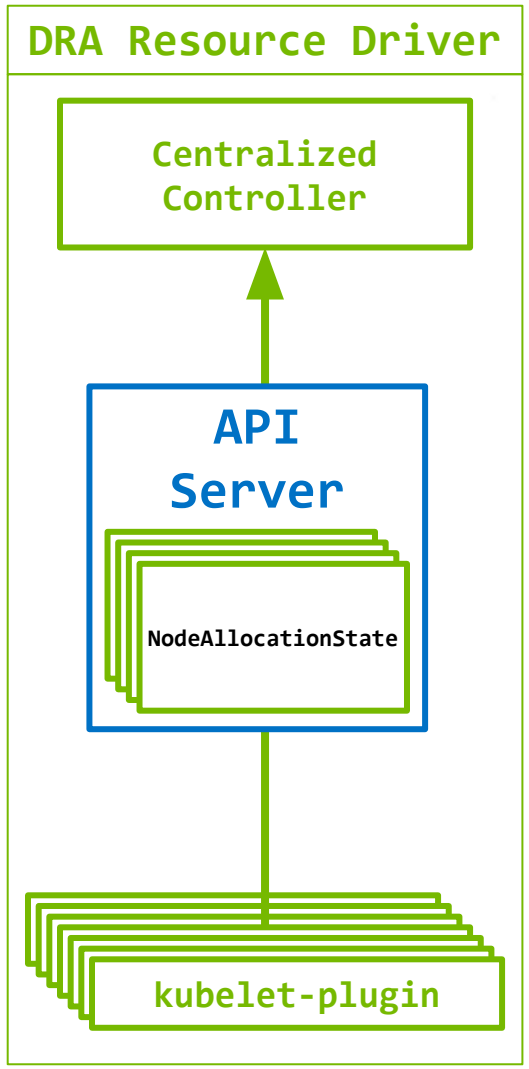
- Single, all-purpose, per-node CRD
  - Advertise available resources
  - Track resources allocated by the controller
  - Track resources prepared by the kubelet plugin

Anatomy of a DRA Resource Driver



- Single, all-purpose, per-node CRD
  - Advertise available resources
  - Track resources allocated by the controller
  - Track resources prepared by the kubelet plugin
- Split-purpose communication
  - Advertise available resources via per-node CRD
  - Track allocated resources using **`ResourceHandles`** inside the **`ResourceClaim`** itself
  - Track resource prepared by the kubelet plugin in a checkpoint file on the node-local filesystem

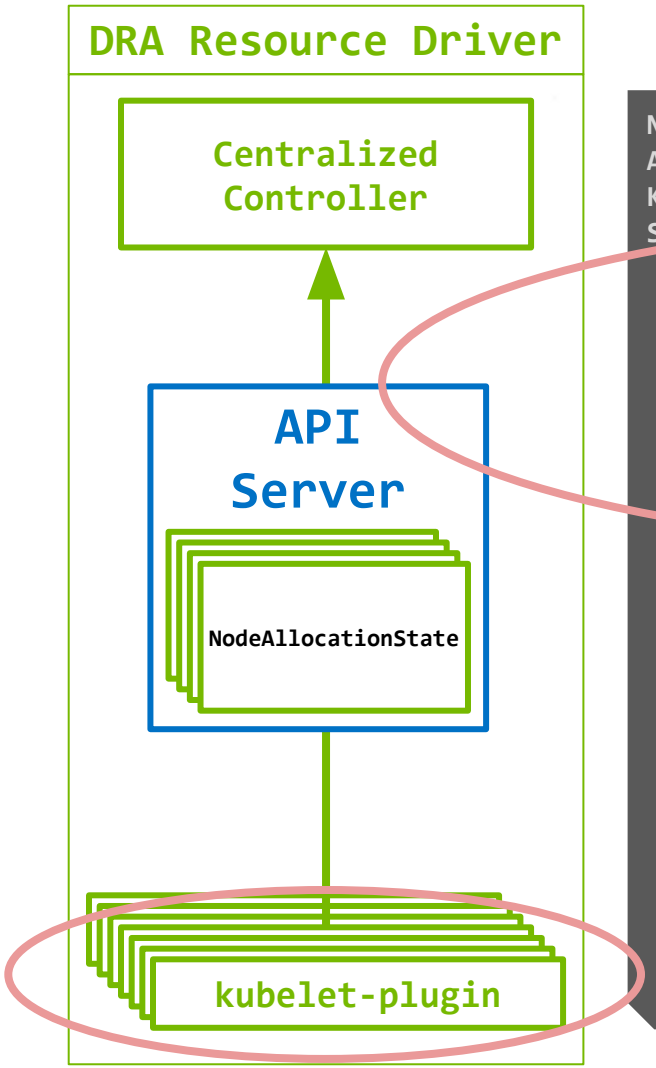
Anatomy of a DRA Resource Driver



- Single, all-purpose, per-node CRD
  - Advertise available resources
  - Track resources allocated by the controller
  - Track resources prepared by the kubelet plugin

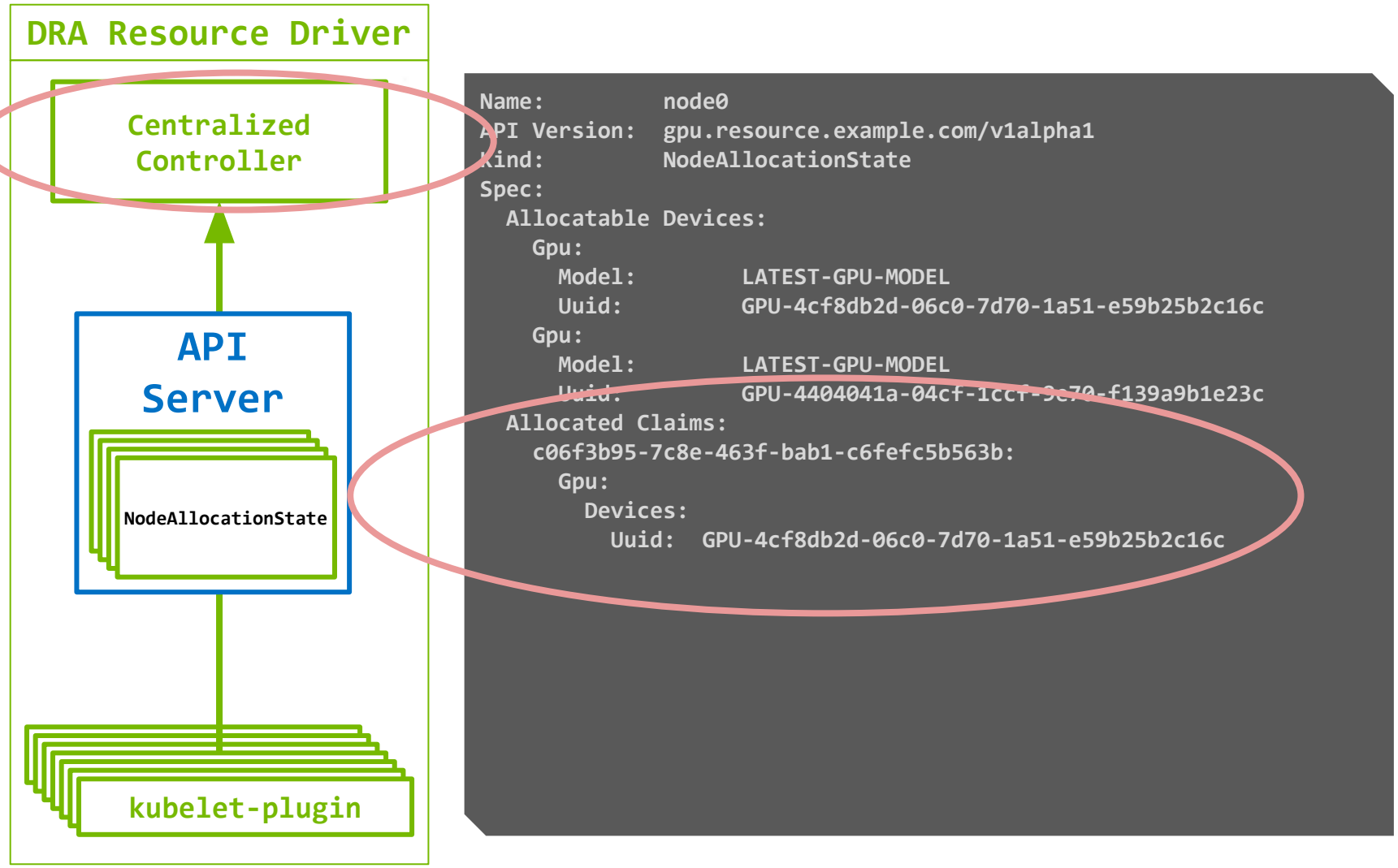
- Split-purpose communication
  - Advertise available resources via per-node CRD
  - Track allocated resources using `ResourceHandles` inside the `ResourceClaim` itself
  - Track resource prepared by the kubelet plugin in a checkpoint file on the node-local filesystem

Anatomy of a DRA Resource Driver

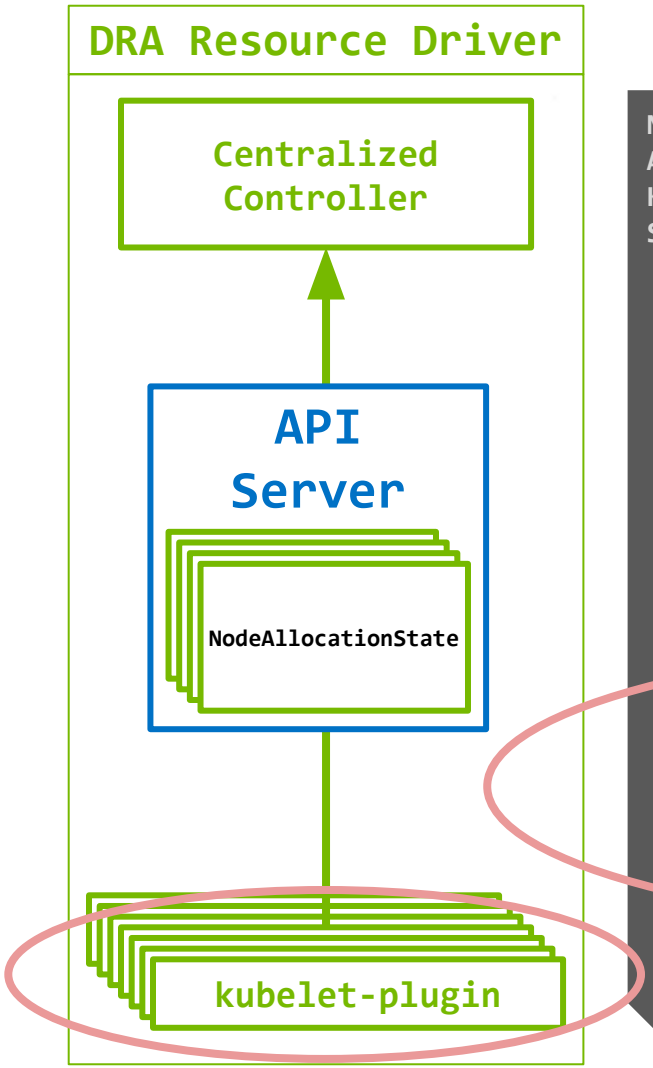



```
Name:         node0
API Version:  gpu.resource.example.com/v1alpha1
Kind:         NodeAllocationState
Spec:
  Allocatable Devices:
    Gpu:
      Model:        LATEST-GPU-MODEL
      Uuid:         GPU-4cf8db2d-06c0-7d70-1a51-e59b25b2c16c
    Gpu:
      Model:        LATEST-GPU-MODEL
      Uuid:         GPU-4404041a-04cf-1ccf-9e70-f139a9b1e23c
```

Anatomy of a DRA Resource Driver

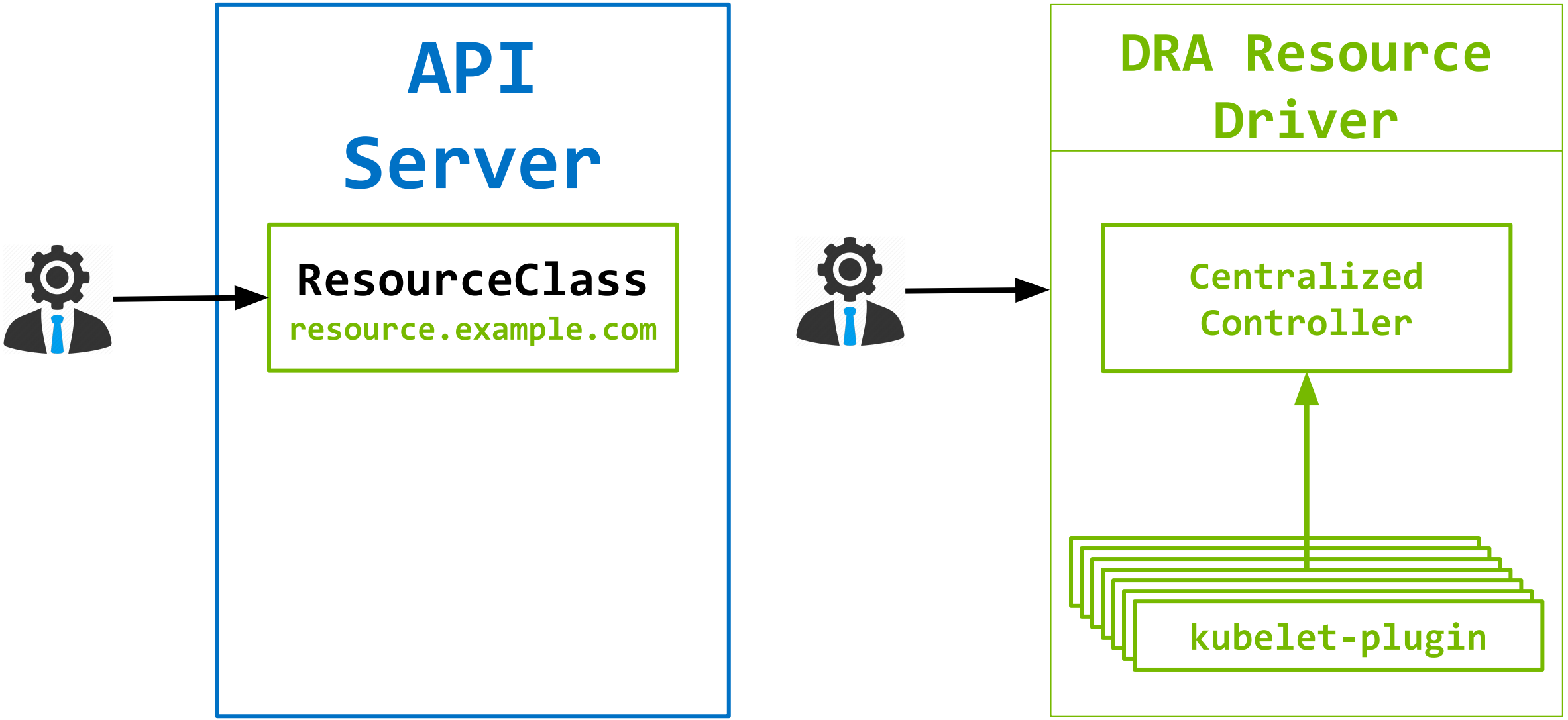


Anatomy of a DRA Resource Driver

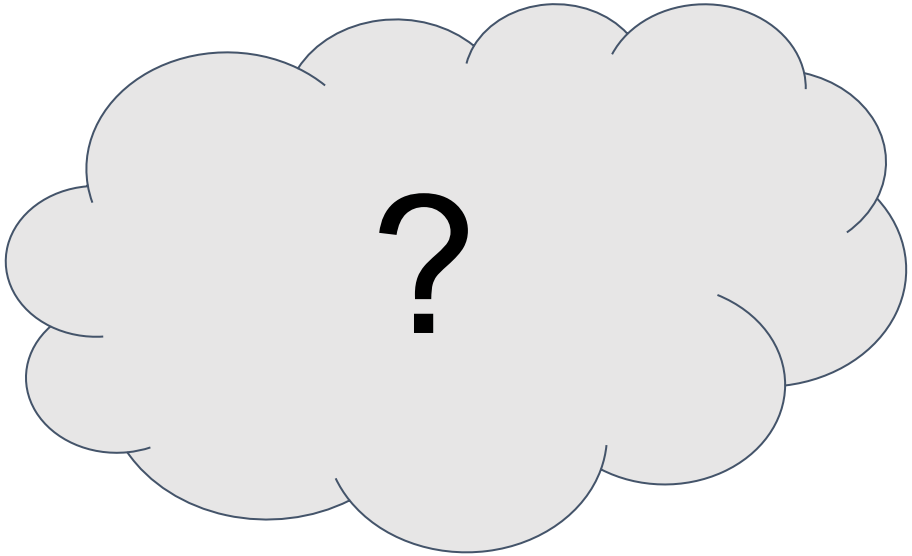



```
Name:         node0
API Version:  gpu.resource.example.com/v1alpha1
Kind:         NodeAllocationState
Spec:
  Allocatable Devices:
    Gpu:
      Model:        LATEST-GPU-MODEL
      Uuid:         GPU-4cf8db2d-06c0-7d70-1a51-e59b25b2c16c
    Gpu:
      Model:        LATEST-GPU-MODEL
      Uuid:         GPU-4404041a-04cf-1ccf-9e70-f139a9b1e23c
  Allocated Claims:
    c06f3b95-7c8e-463f-bab1-c6fefc5b563b:
      Gpu:
        Devices:
          Uuid:  GPU-4cf8db2d-06c0-7d70-1a51-e59b25b2c16c
  Prepared Claims:
    c06f3b95-7c8e-463f-bab1-c6fefc5b563b:
      Gpu:
        Devices:
          Uuid:  GPU-4cf8db2d-06c0-7d70-1a51-e59b25b2c16c
```

Anatomy of a DRA Resource Driver



Allocating Resources with DRA



Allocating Resources with DRA

Immediate

Allocate resources *immediately* upon the creation of a **`ResourceClaim`**

Pods that reference the claim will be restricted to nodes where those resources have been allocated

Delayed

(a.ka. WaitForFirstConsumer)

Delay allocation of a **`ResourceClaim`** until the first pod that references it is being scheduled

Resource availability will be considered as part of the pod scheduling decision

Allocating Resources with DRA

Immediate

Allocate resources *immediately* upon the creation of a **`ResourceClaim`**

Pods that reference the claim will be restricted to nodes where those resources have been allocated

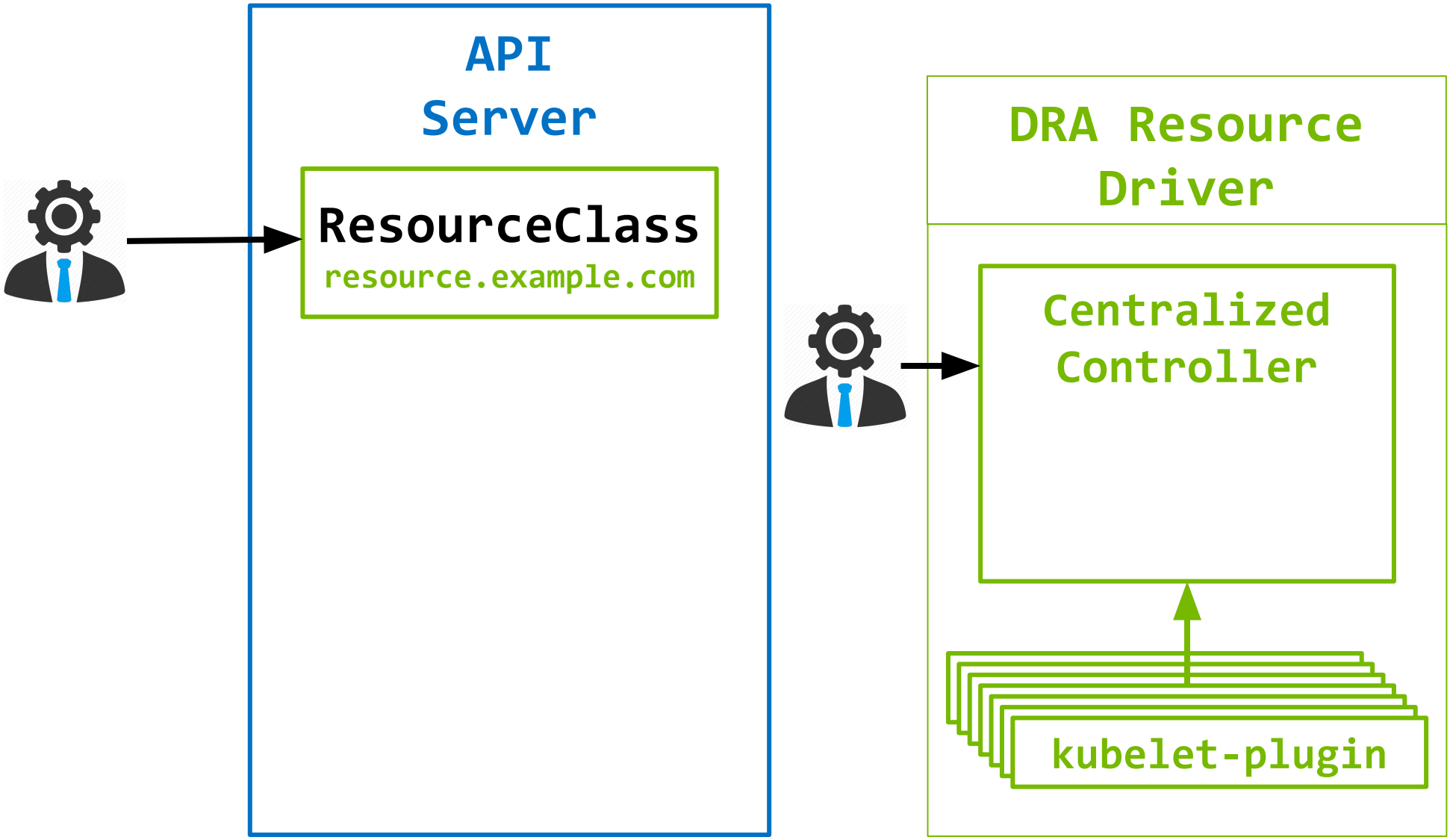
Delayed

(a.ka. WaitForFirstConsumer)

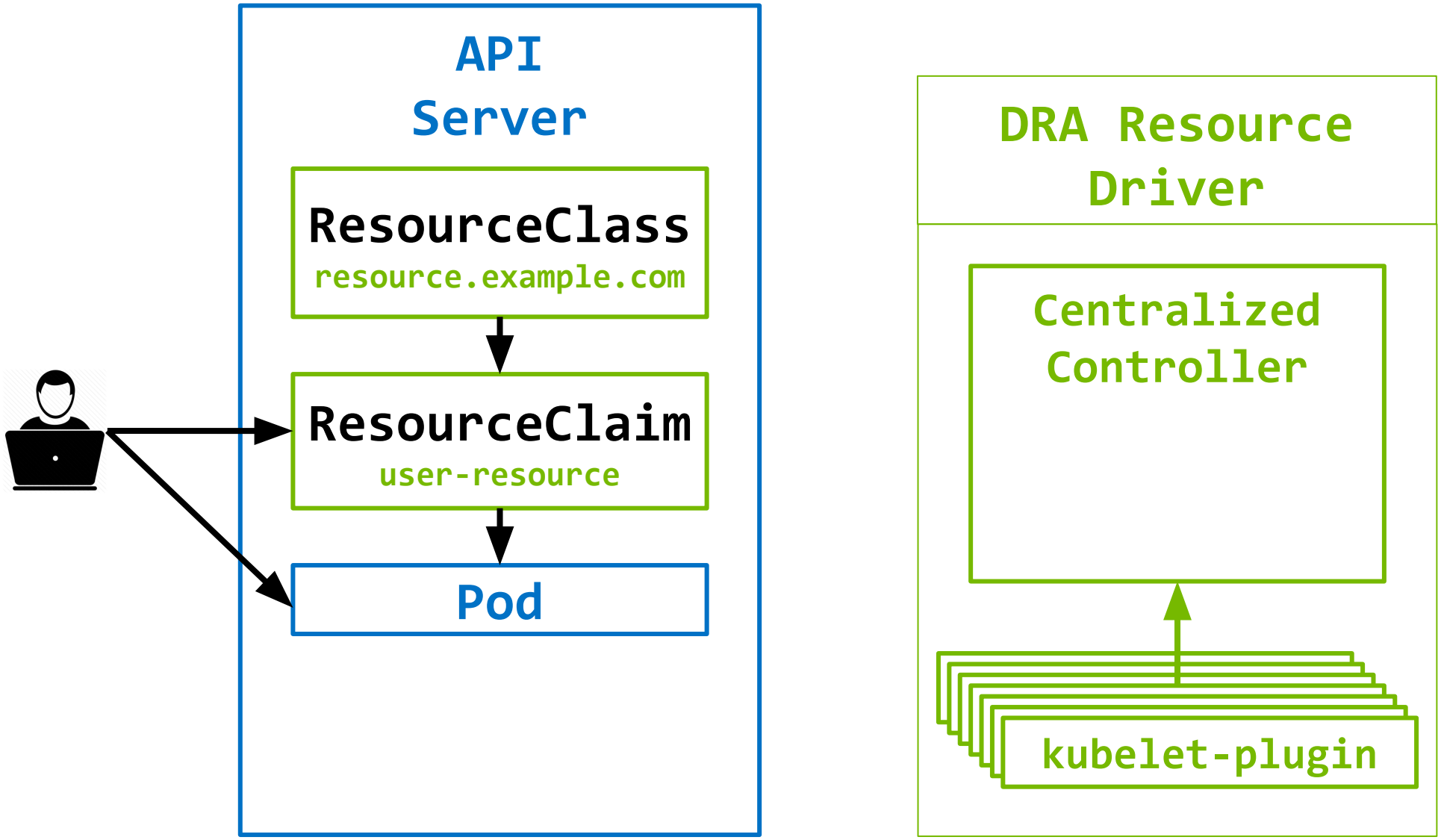
Delay allocation of a **`ResourceClaim`** until the first pod that references it is being scheduled

Resource availability will be considered as part of the pod scheduling decision

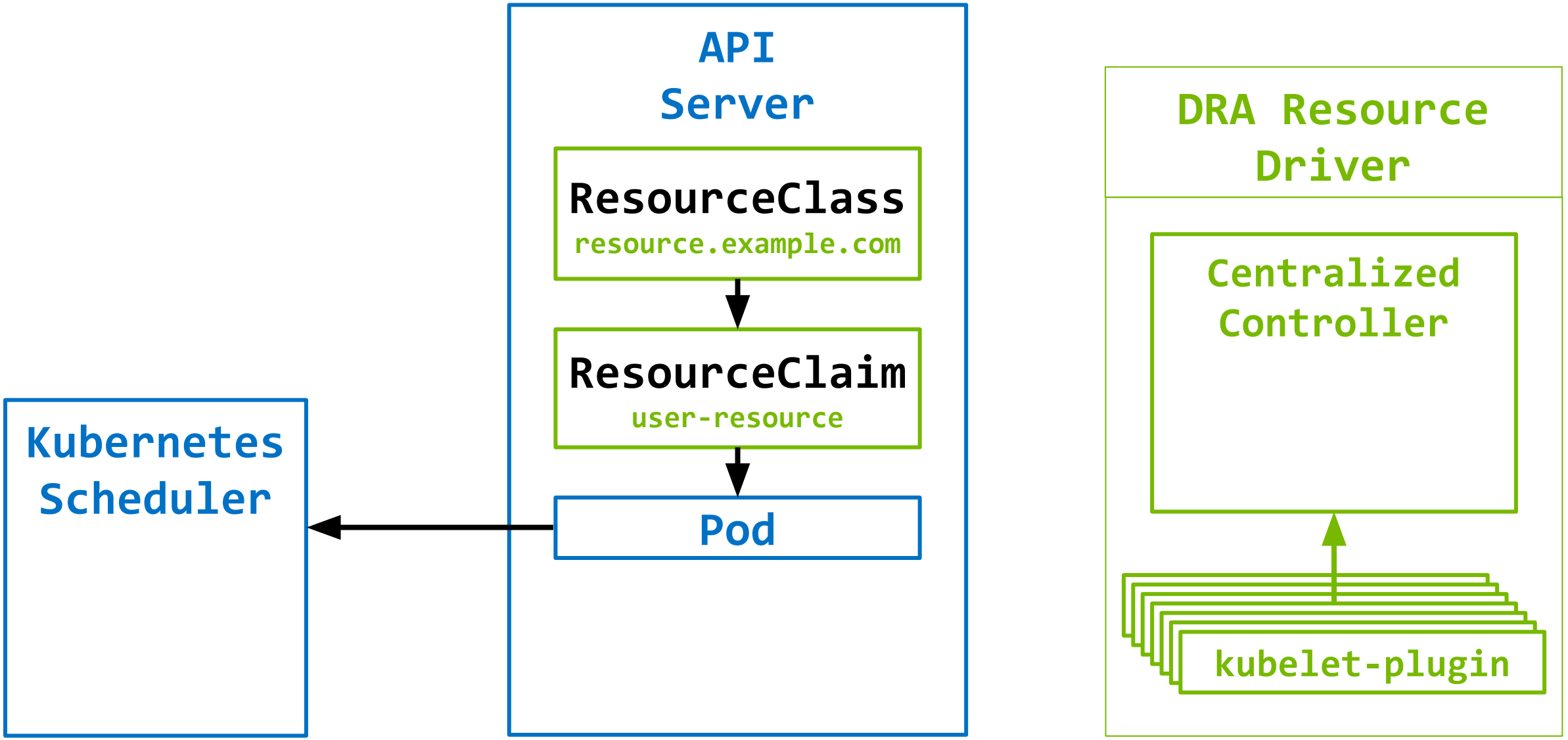
Delayed Allocation with DRA



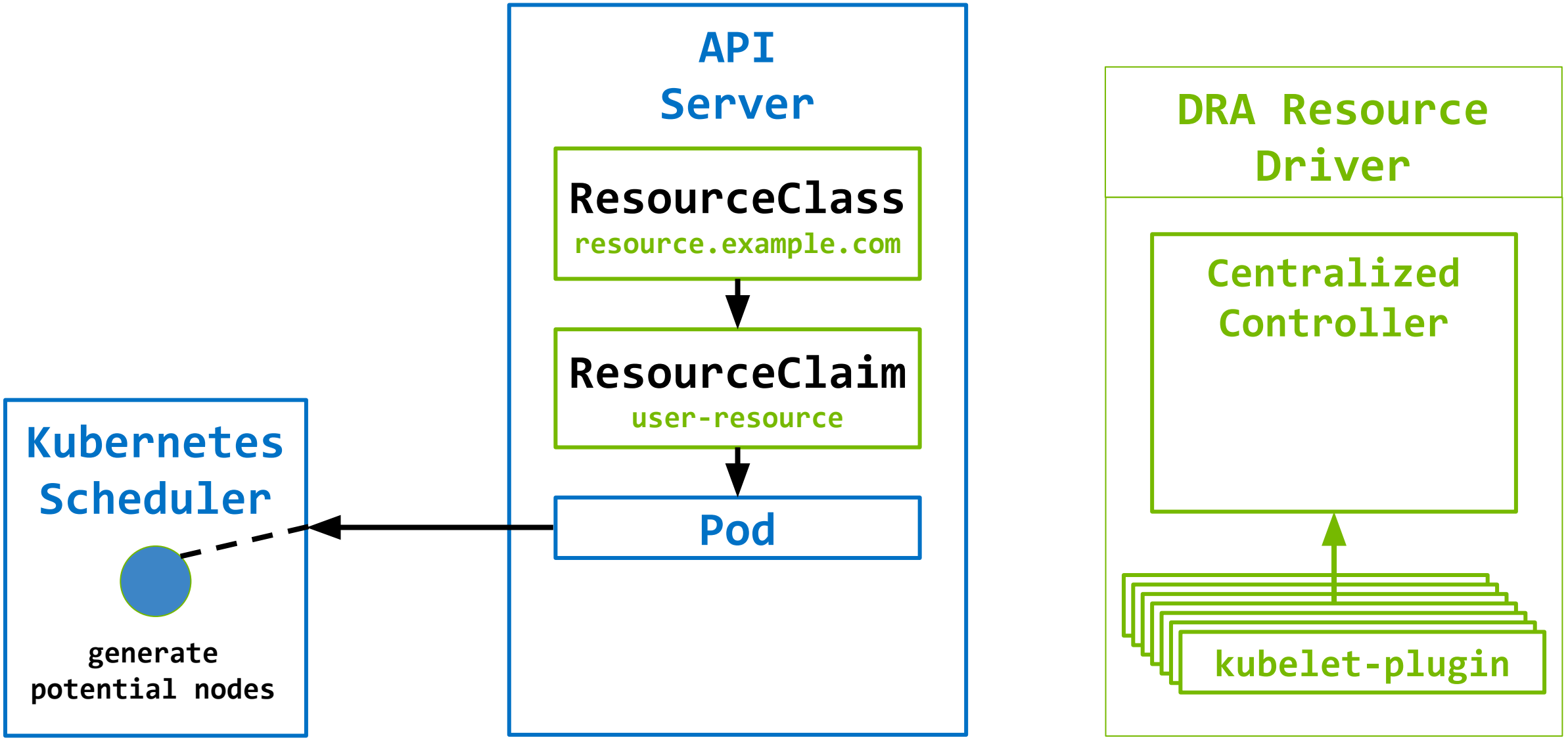
Delayed Allocation with DRA



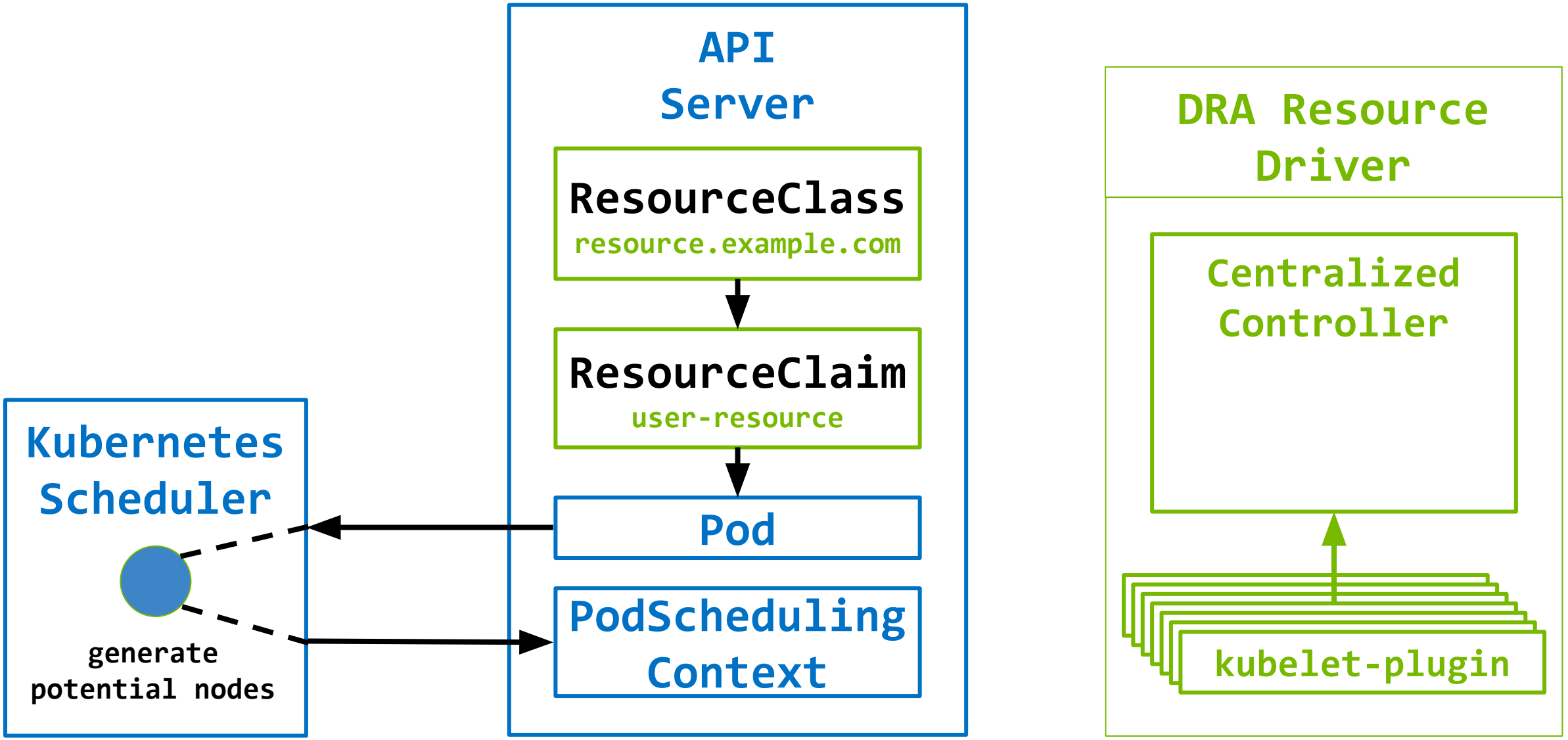
Delayed Allocation with DRA



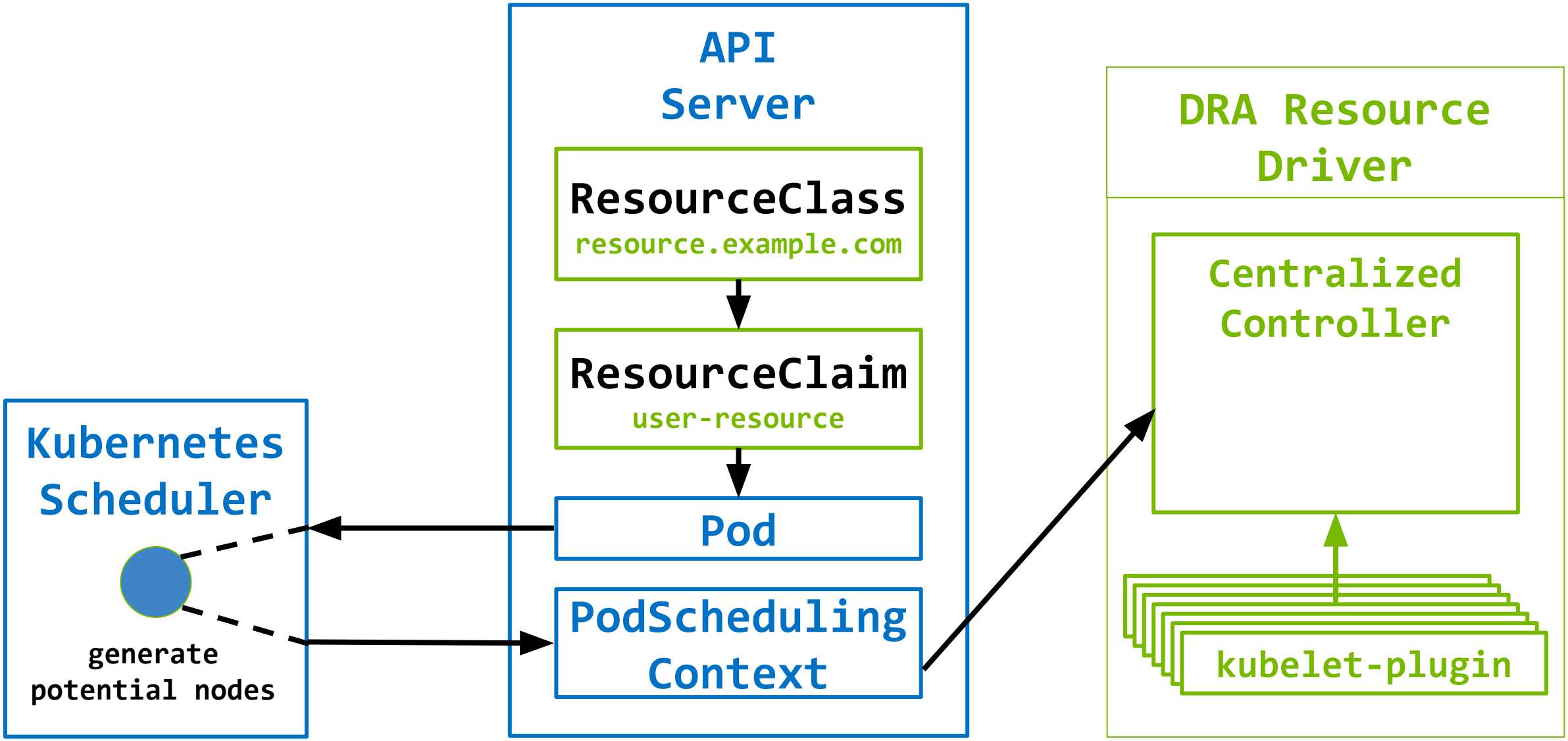
Delayed Allocation with DRA



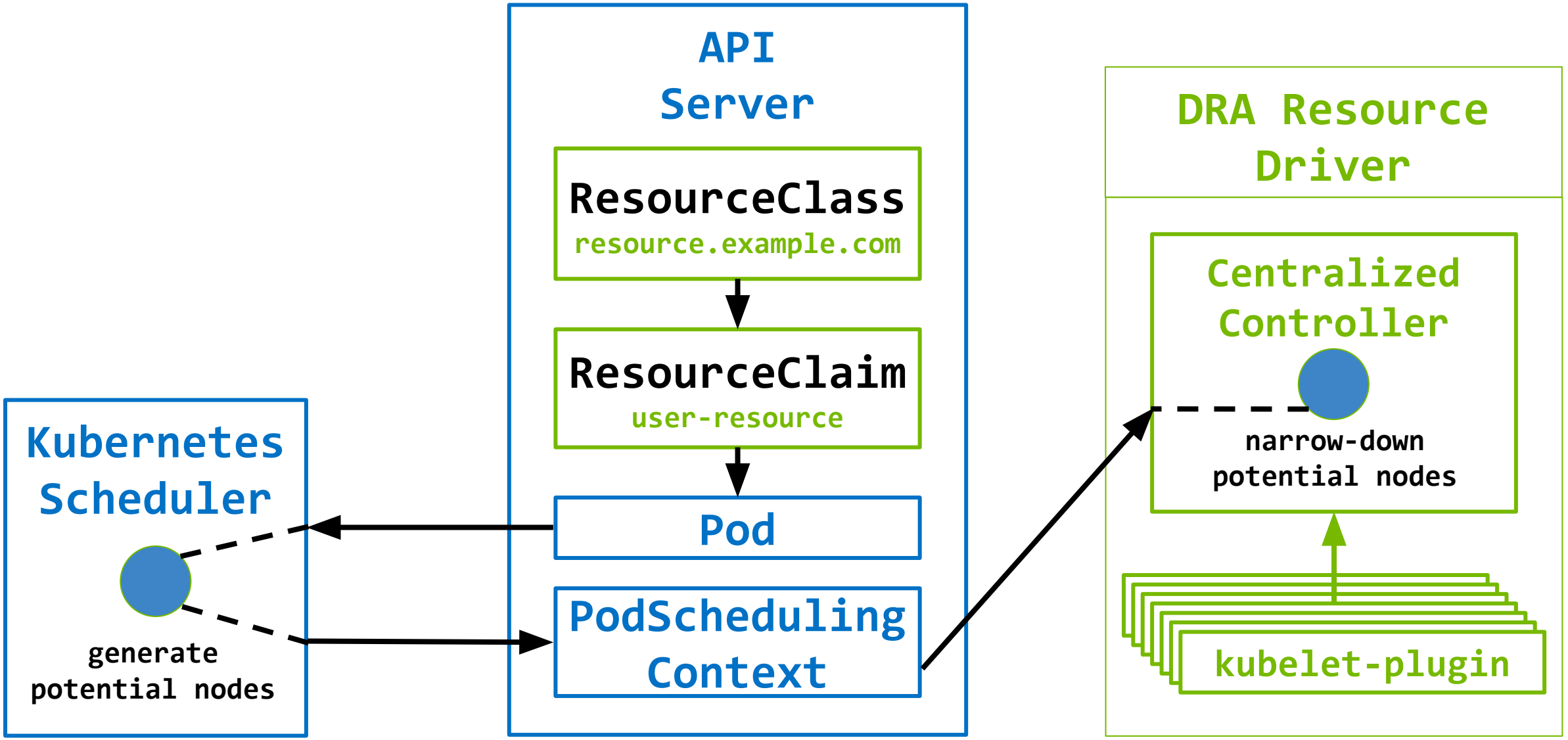
Delayed Allocation with DRA



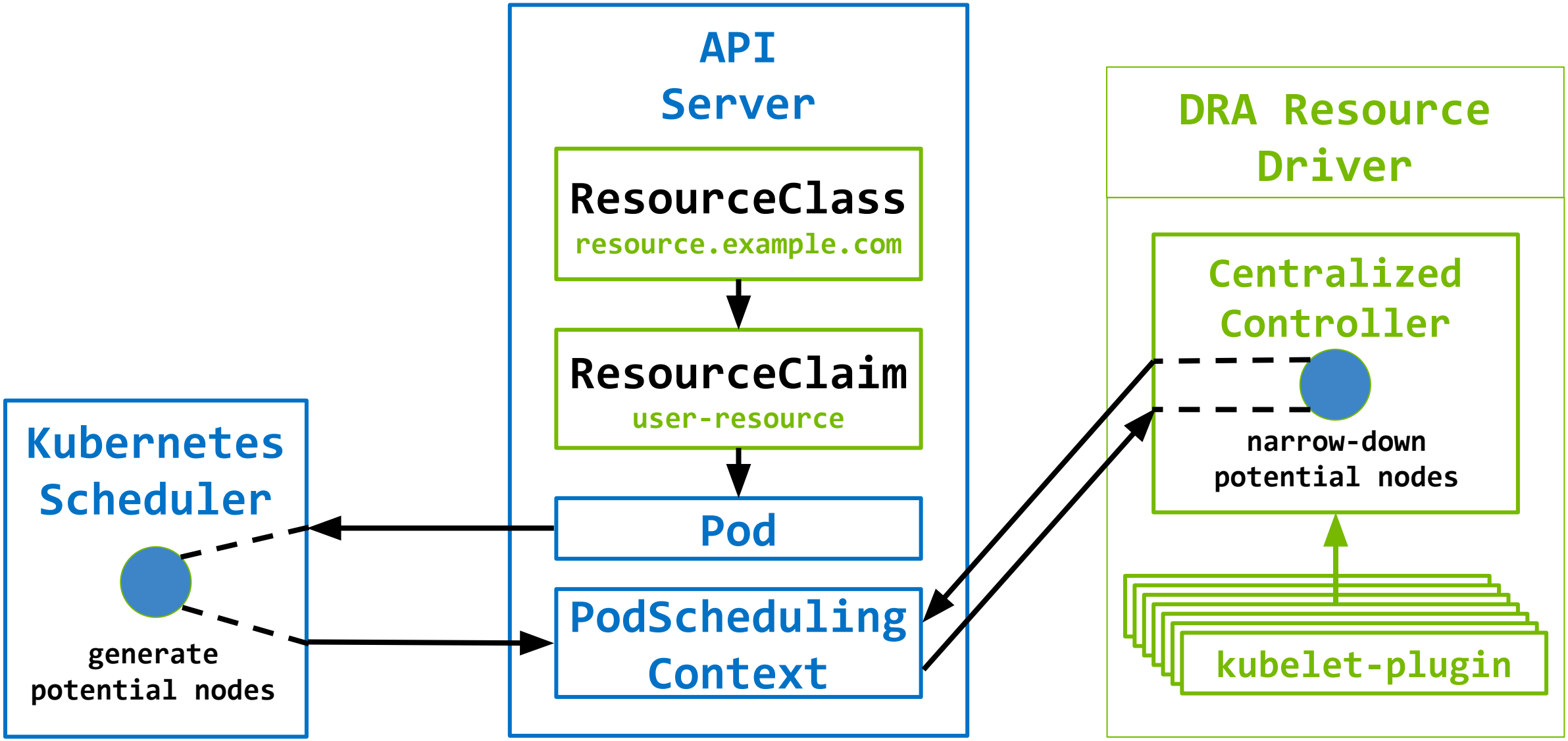
Delayed Allocation with DRA



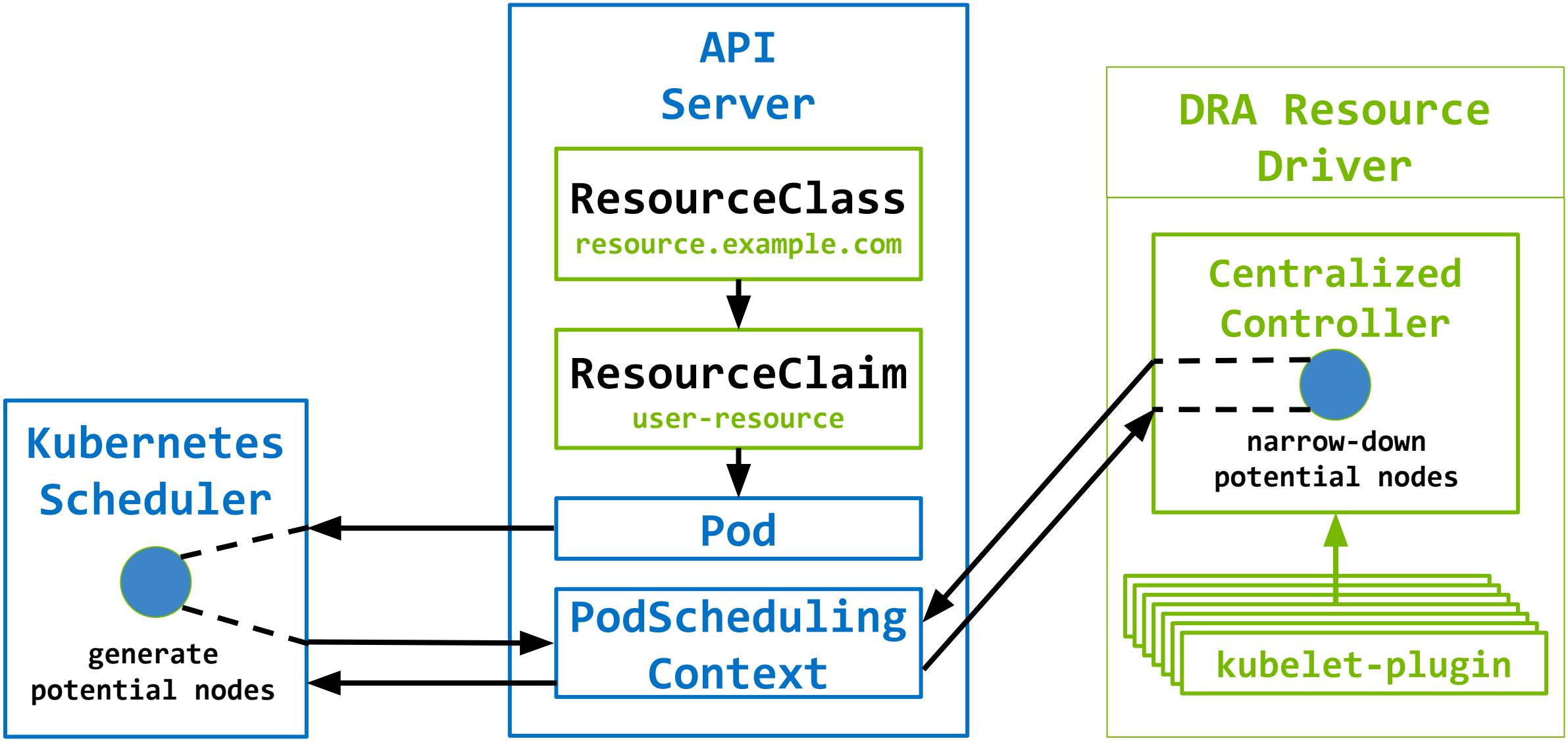
Delayed Allocation with DRA



Delayed Allocation with DRA

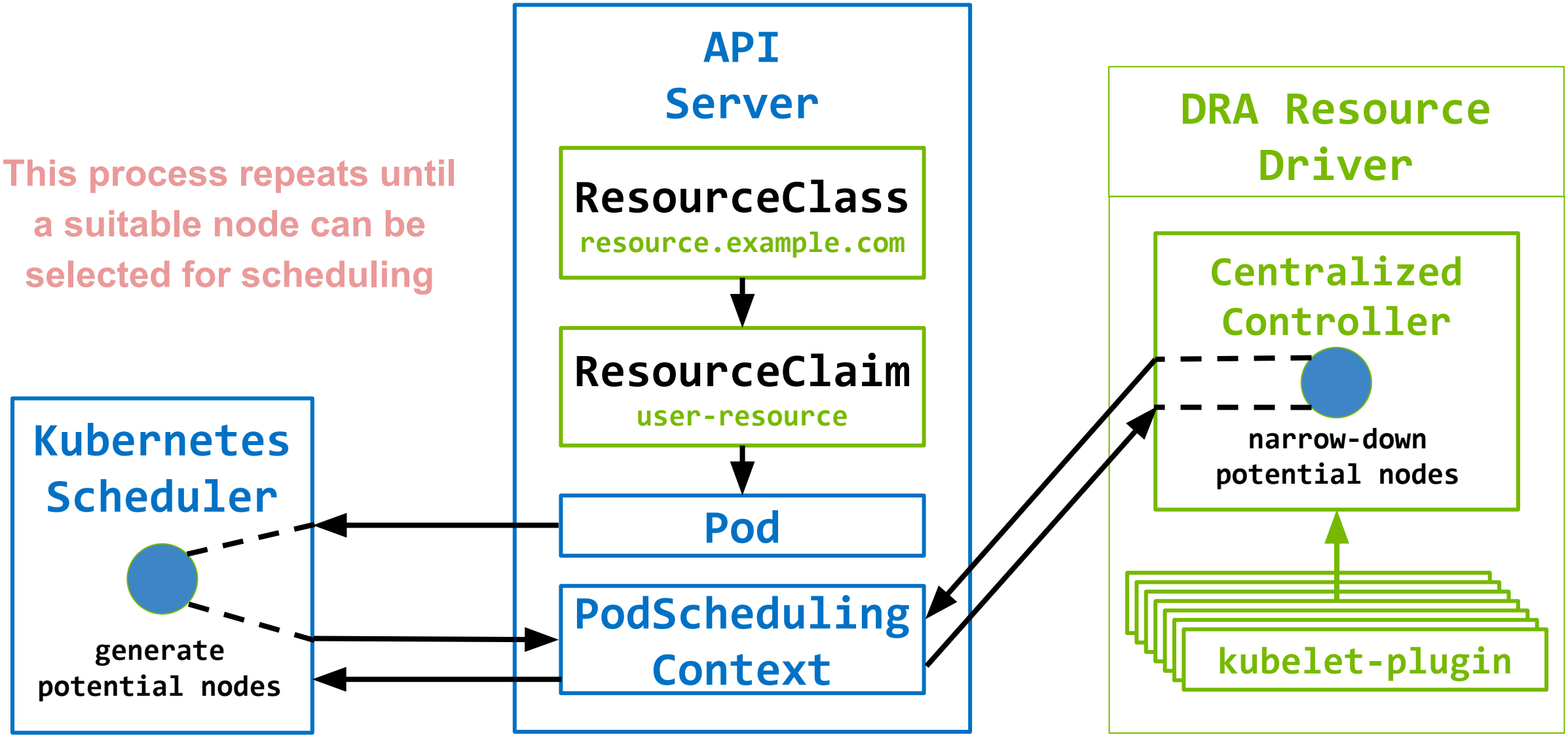


Delayed Allocation with DRA

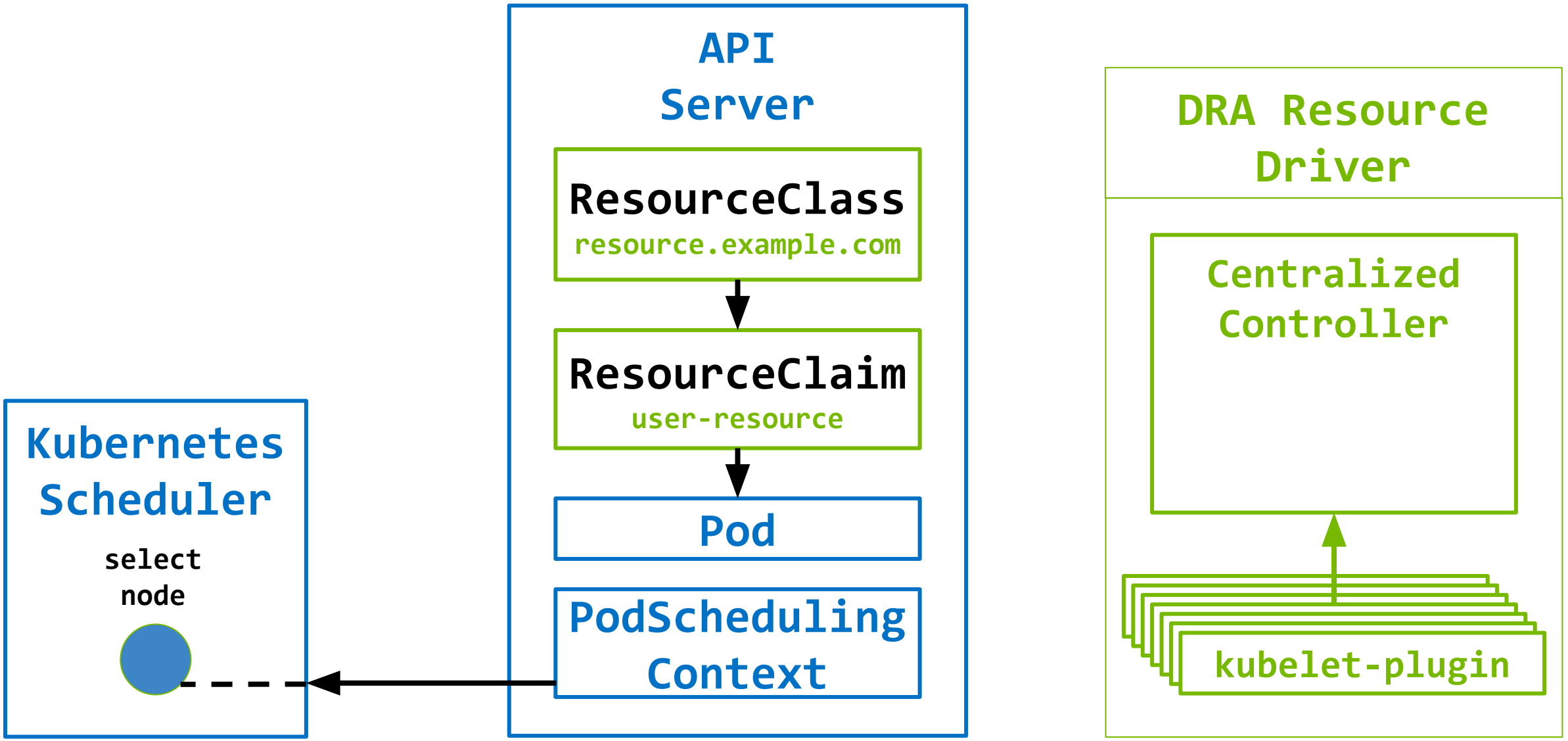


Delayed Allocation with DRA

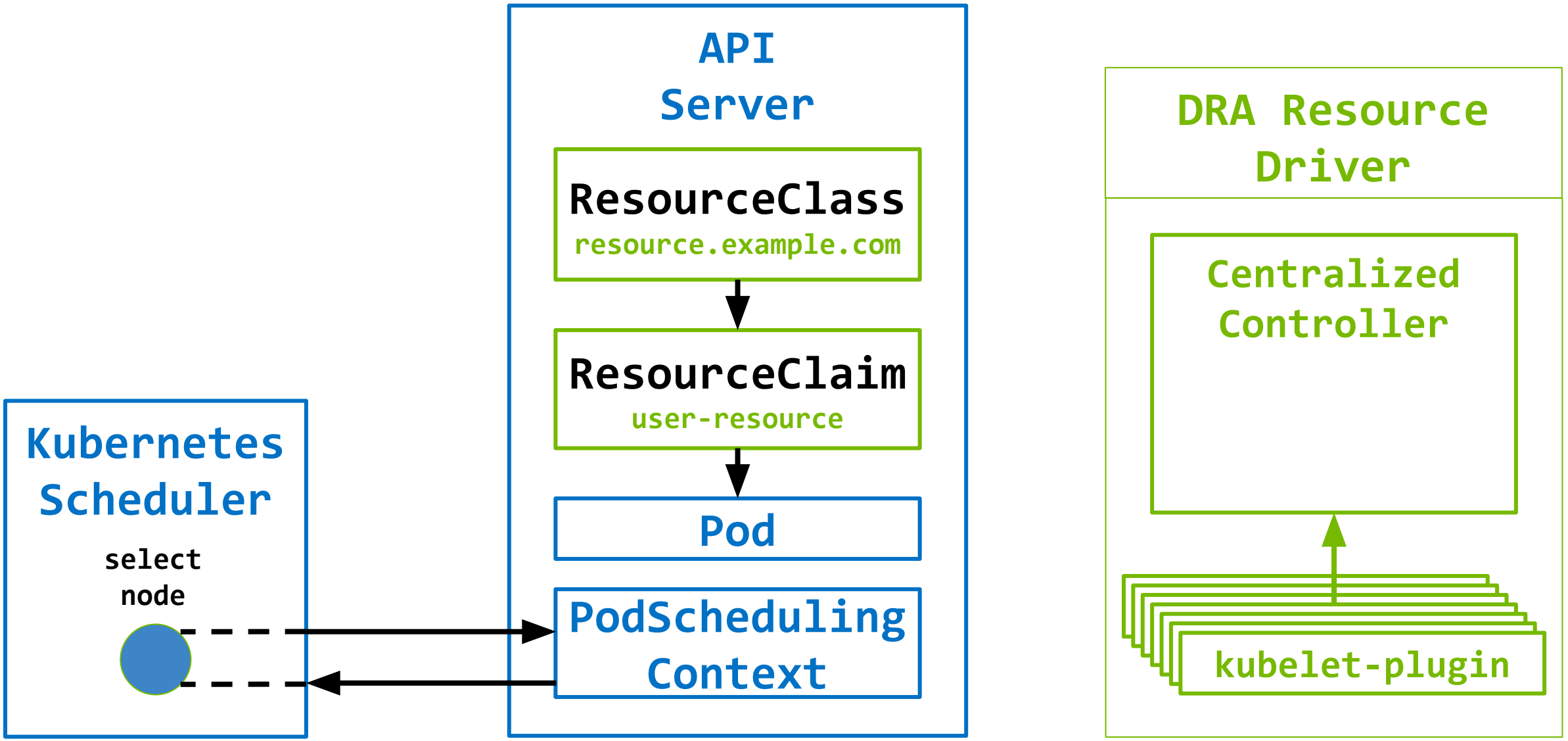
This process repeats until a suitable node can be selected for scheduling



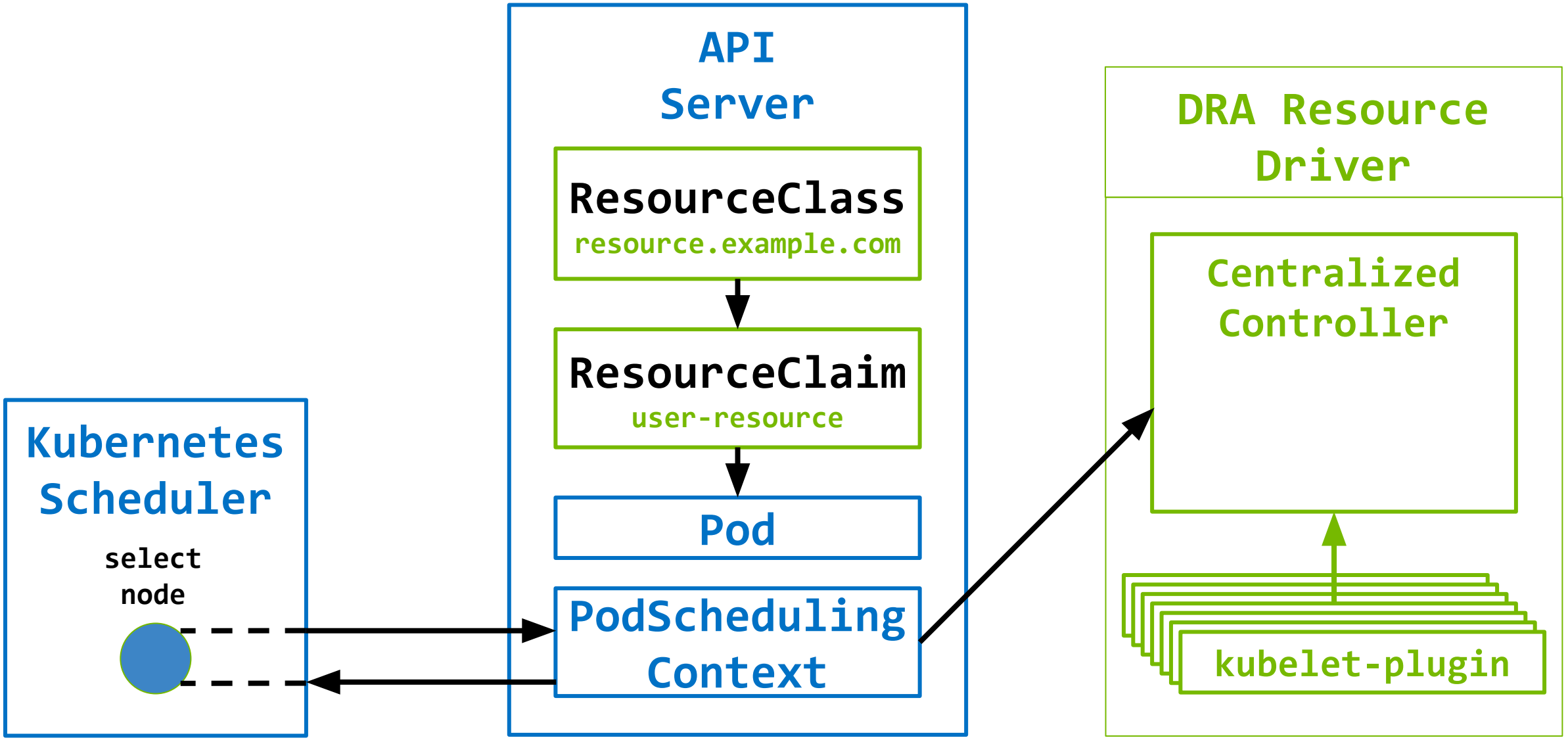
Delayed Allocation with DRA



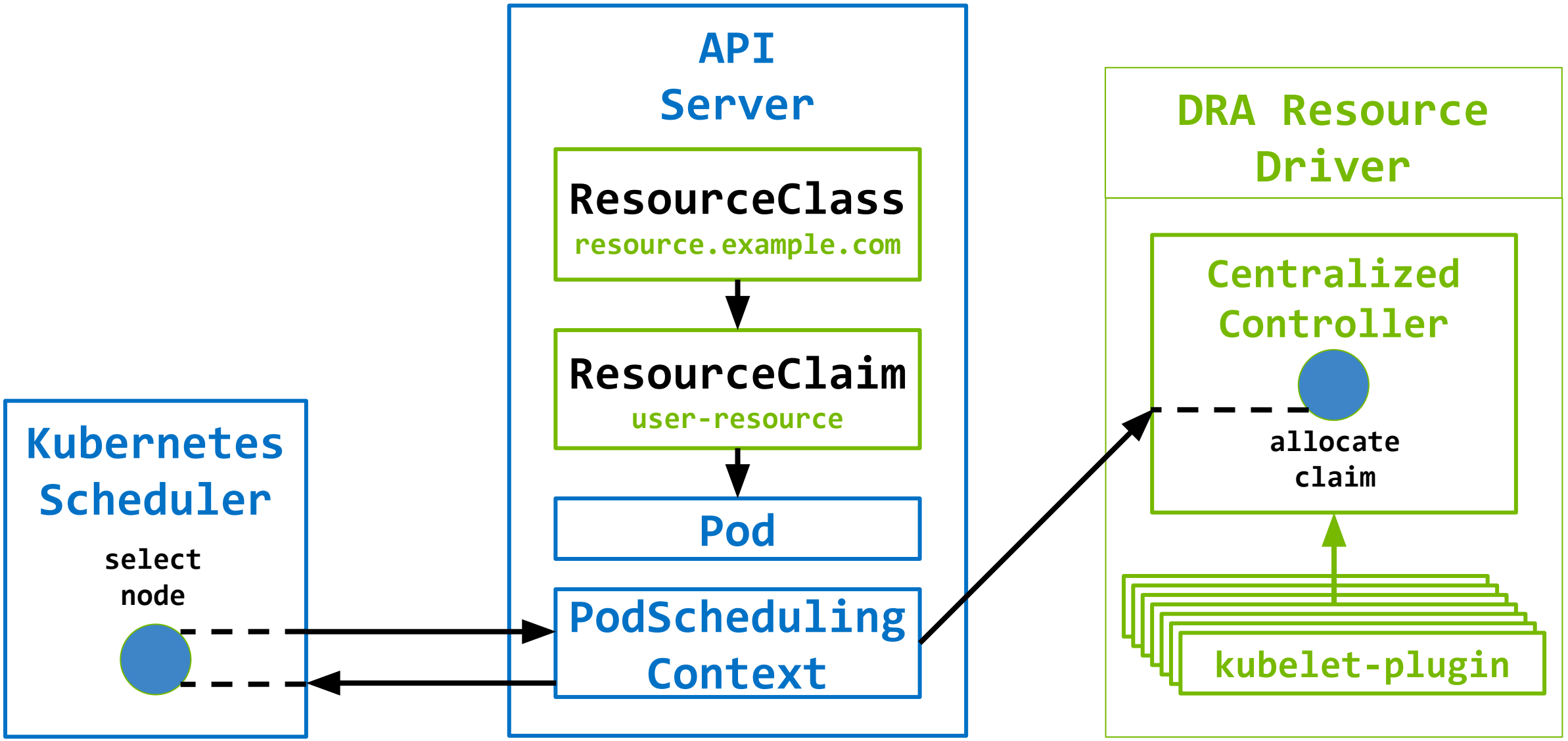
Delayed Allocation with DRA



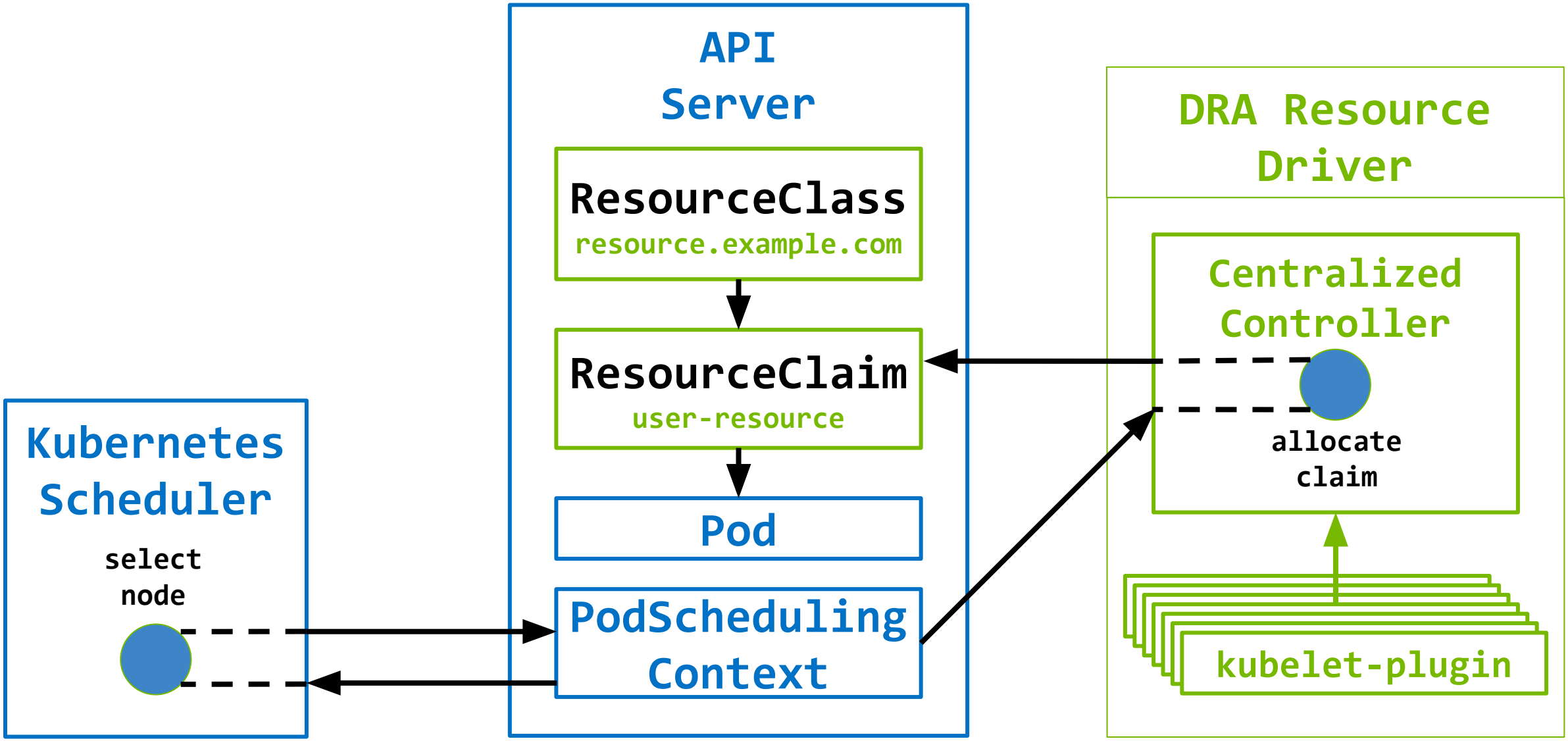
Delayed Allocation with DRA



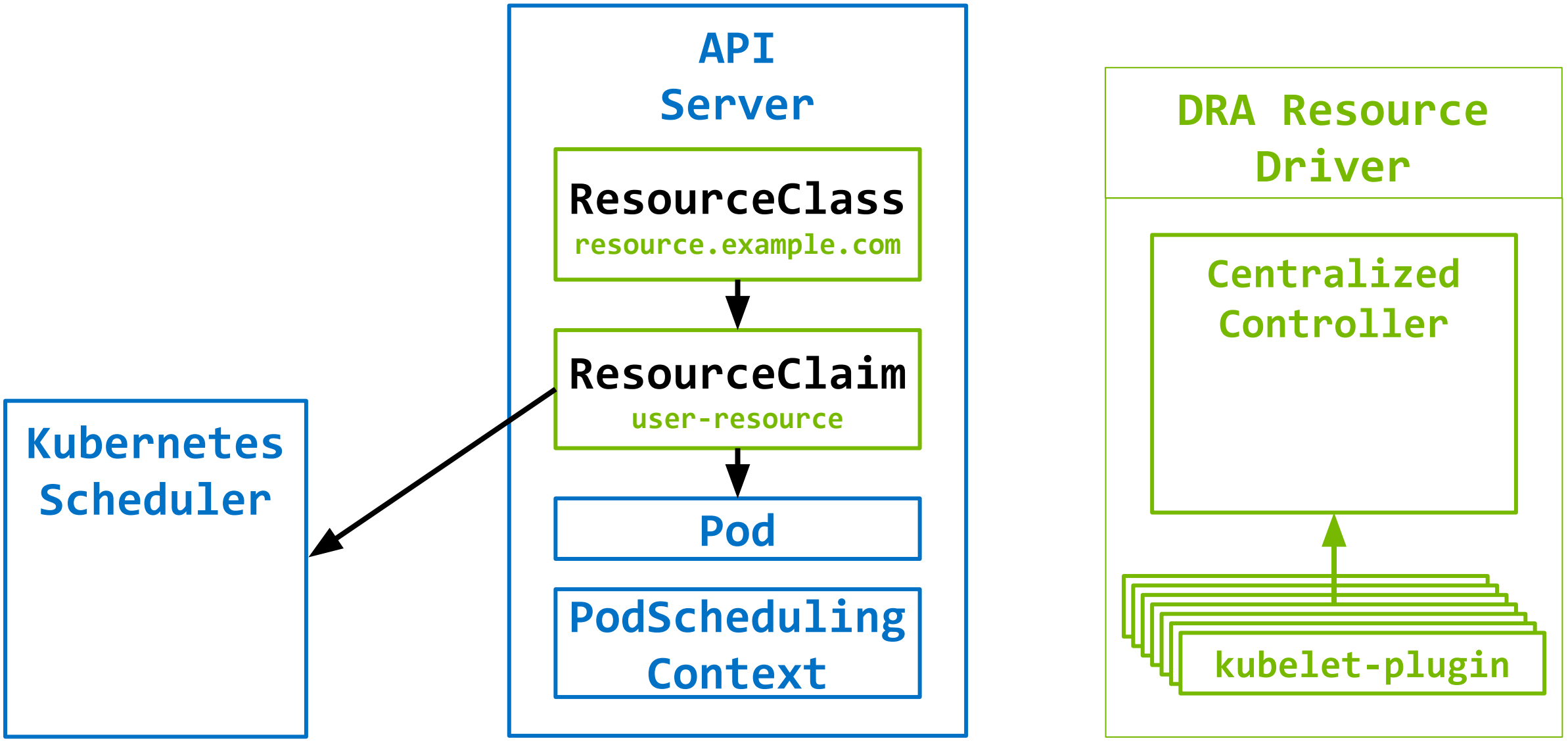
Delayed Allocation with DRA



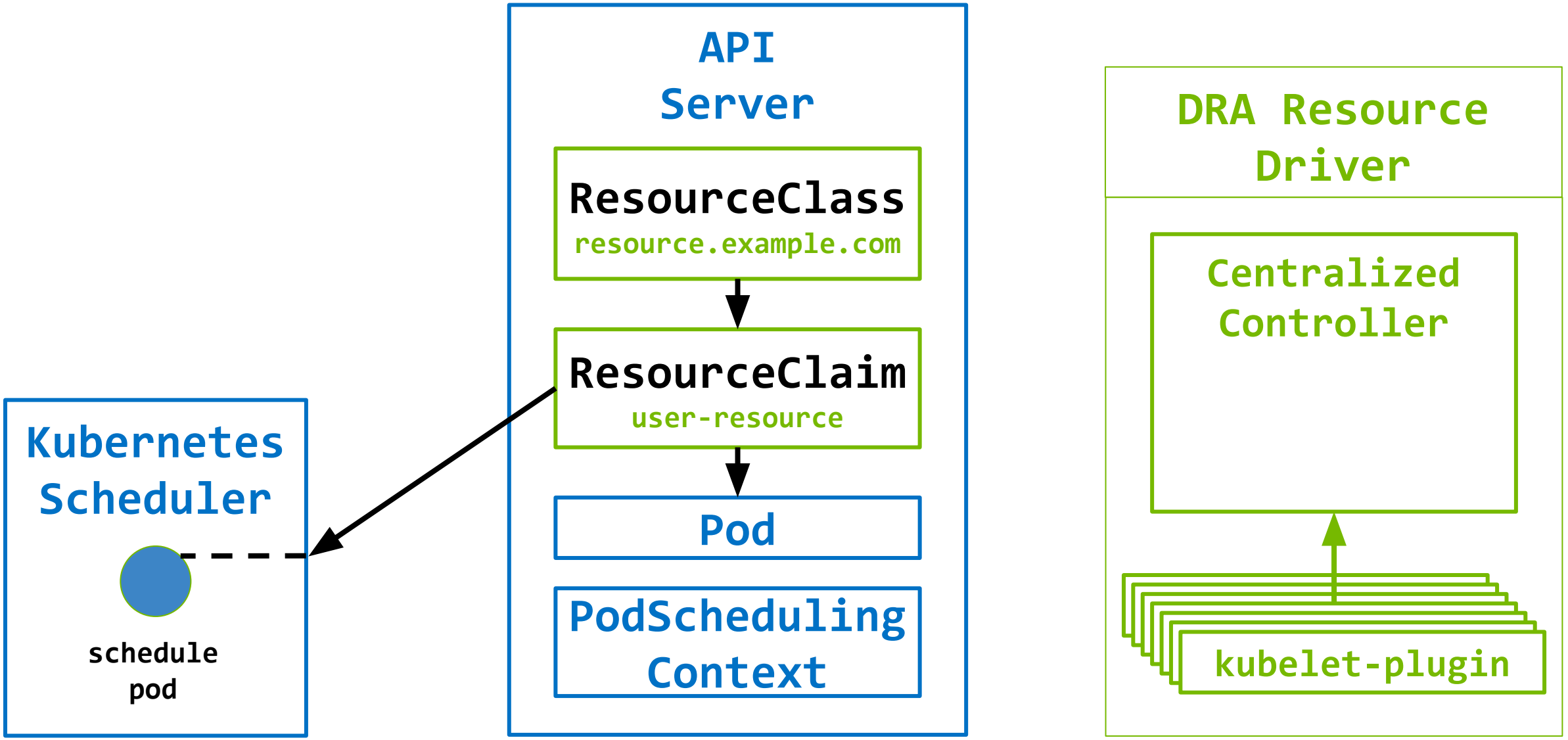
Delayed Allocation with DRA



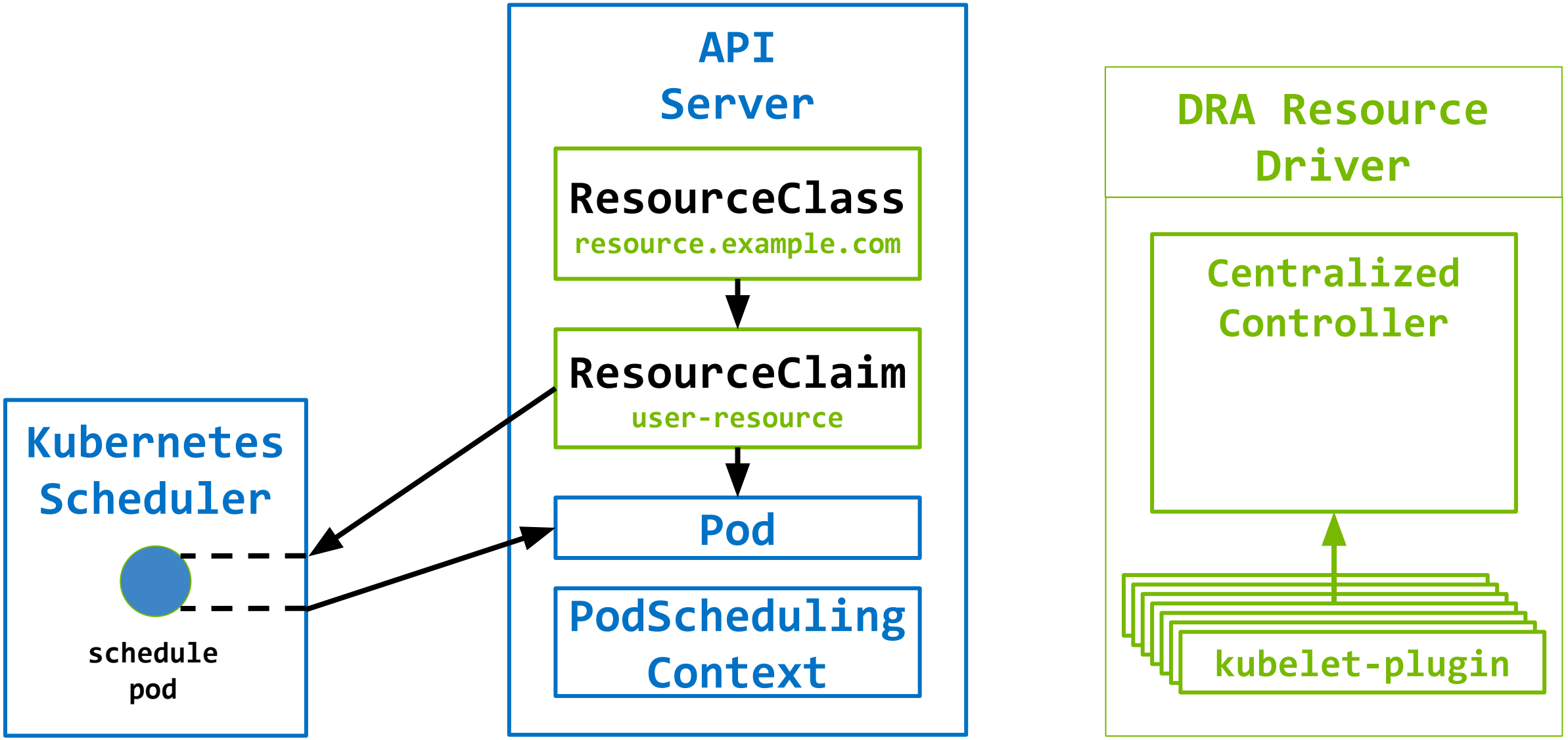
Delayed Allocation with DRA



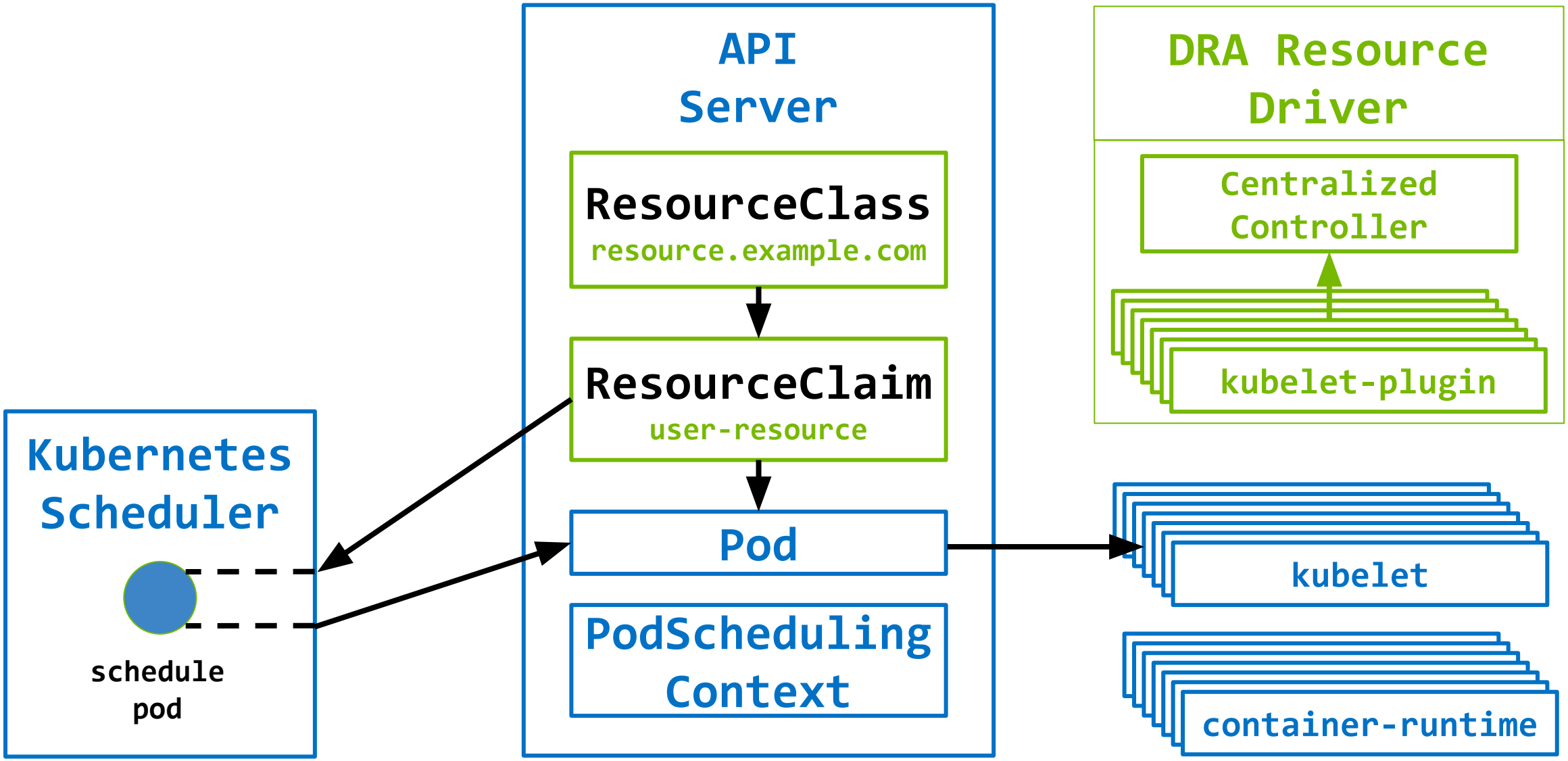
Delayed Allocation with DRA



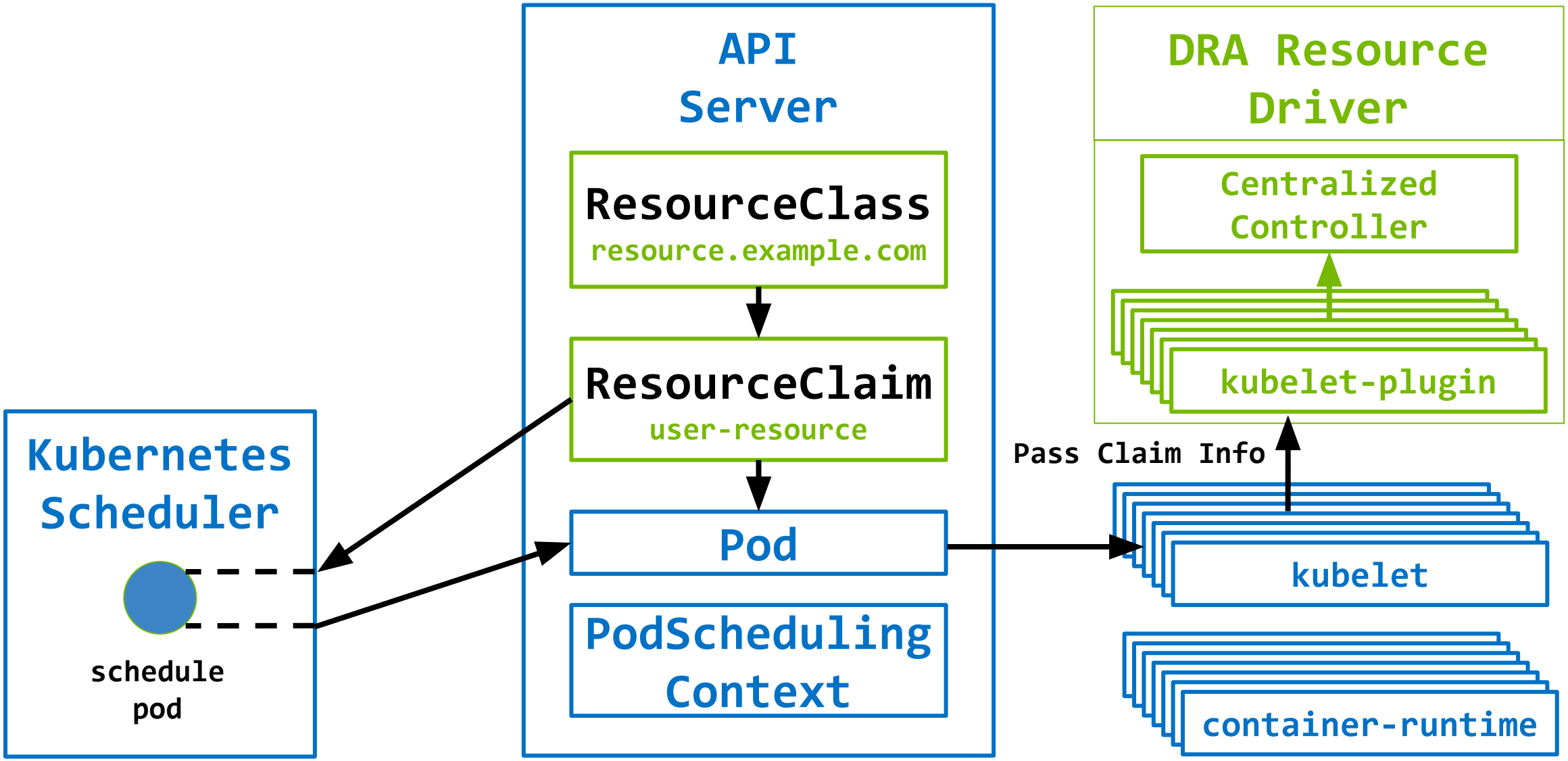
Delayed Allocation with DRA



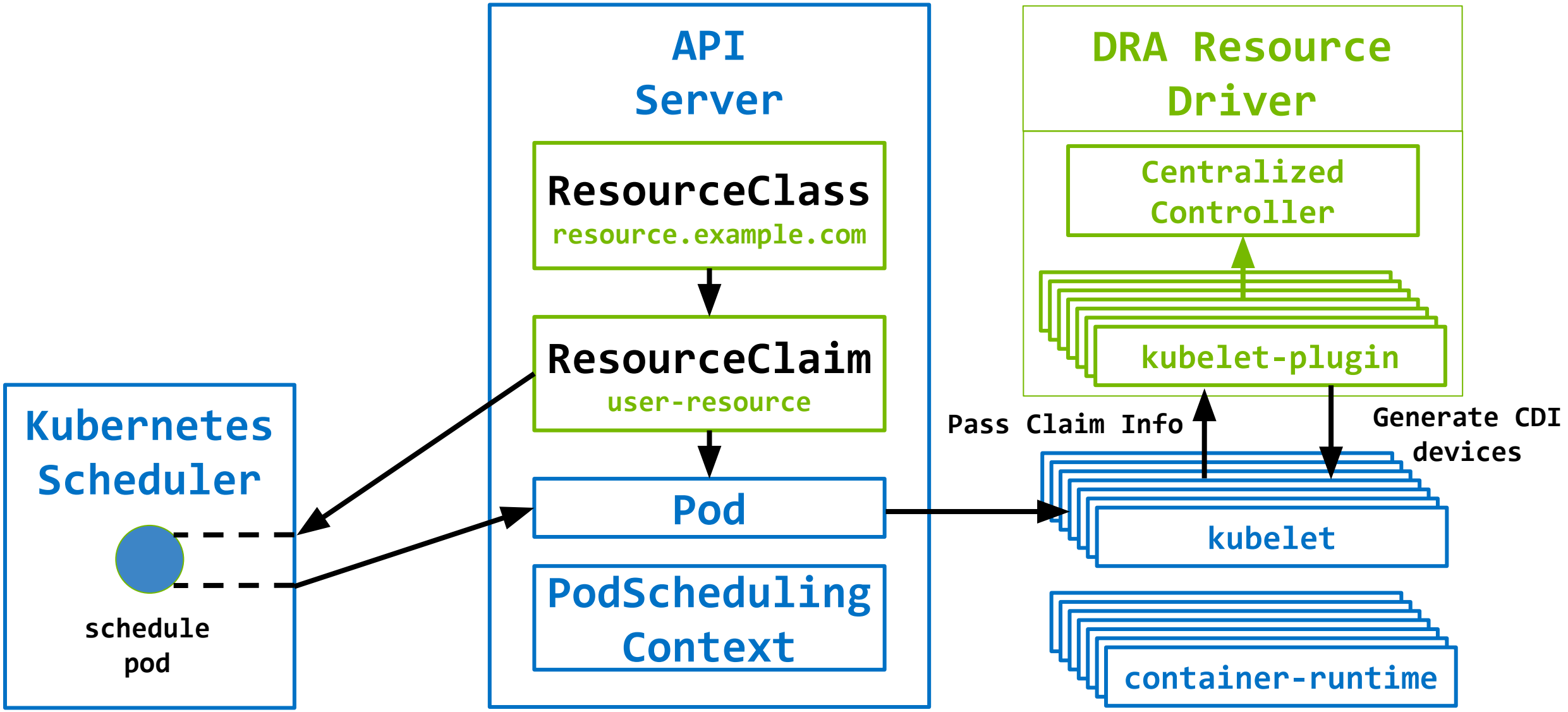
Delayed Allocation with DRA



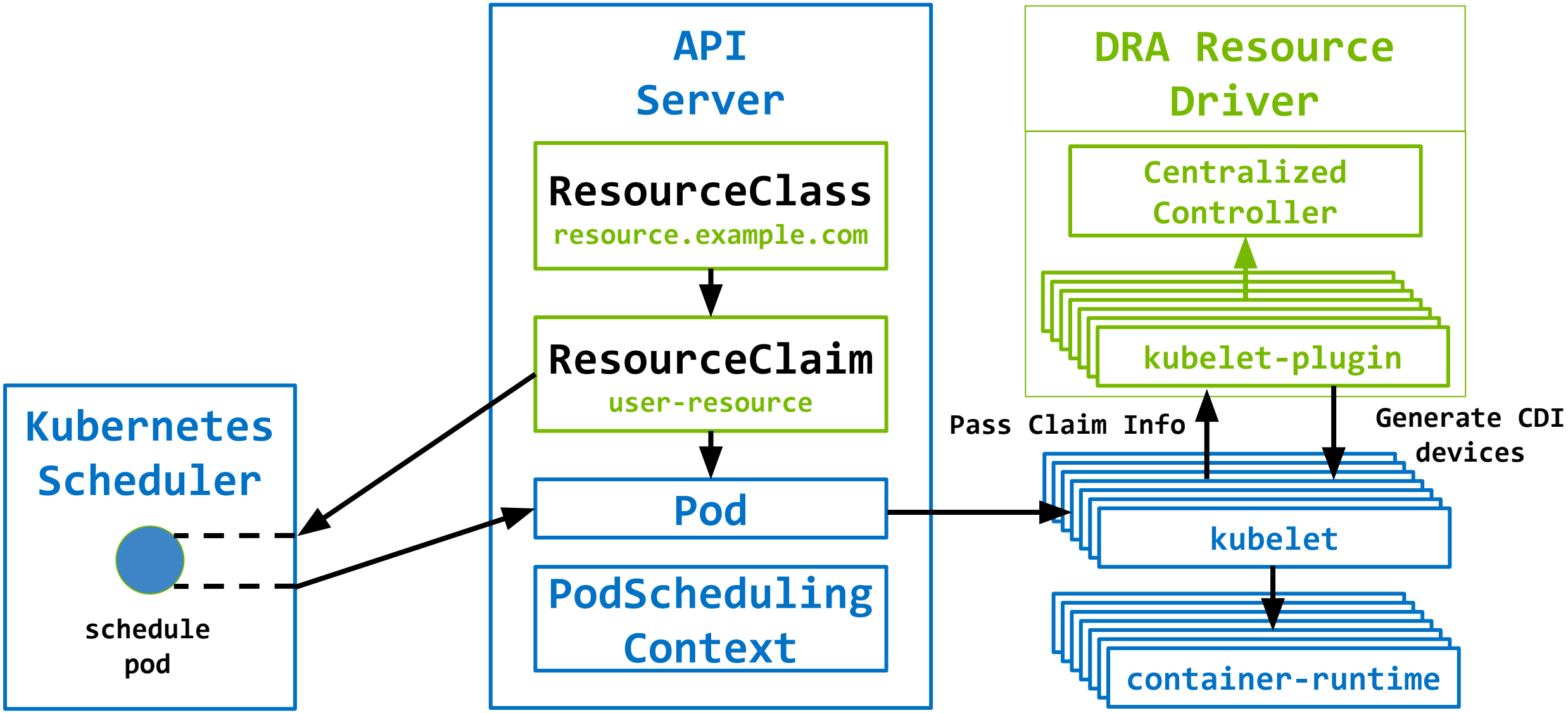
Delayed Allocation with DRA



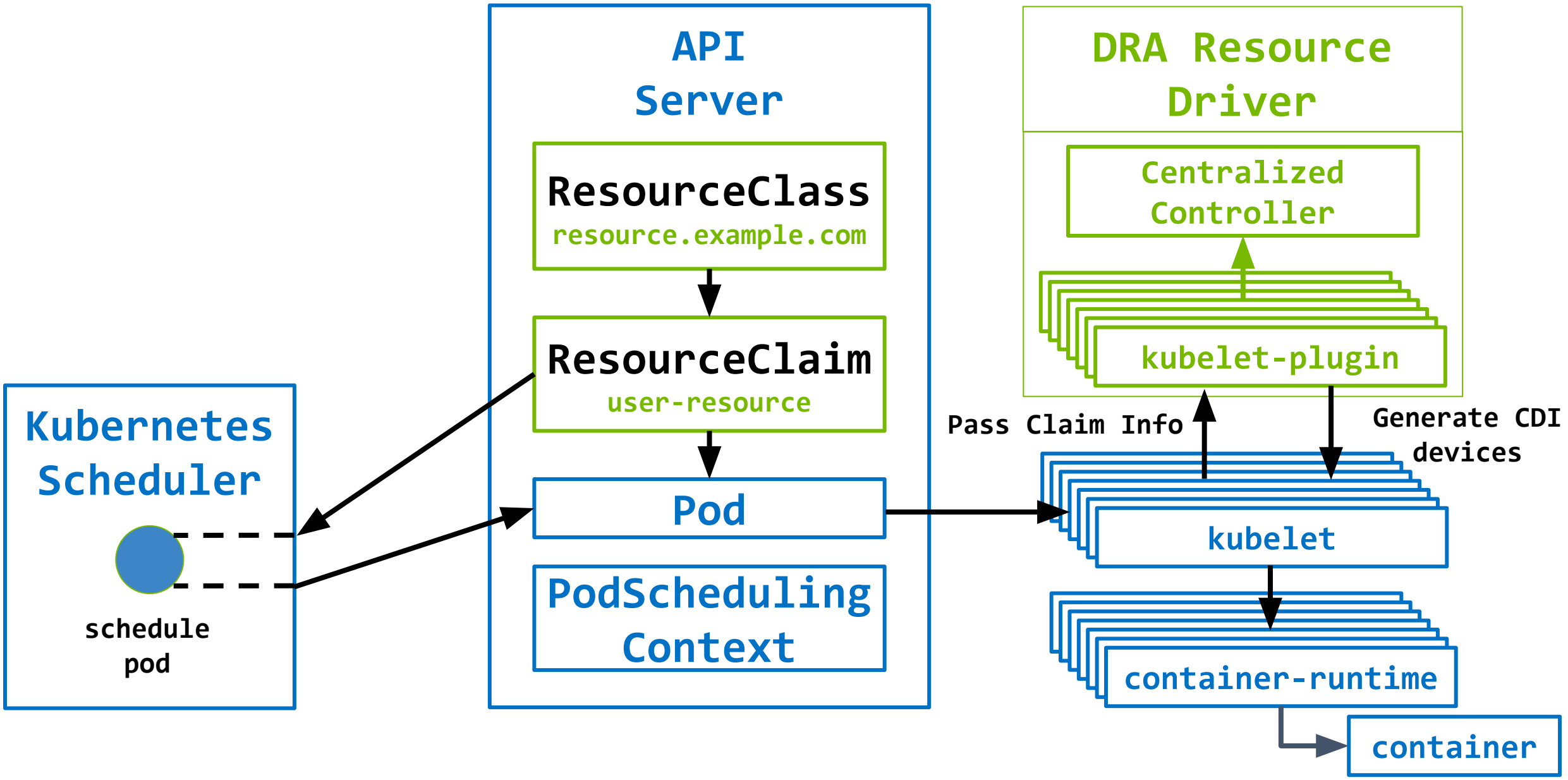
Delayed Allocation with DRA



Delayed Allocation with DRA



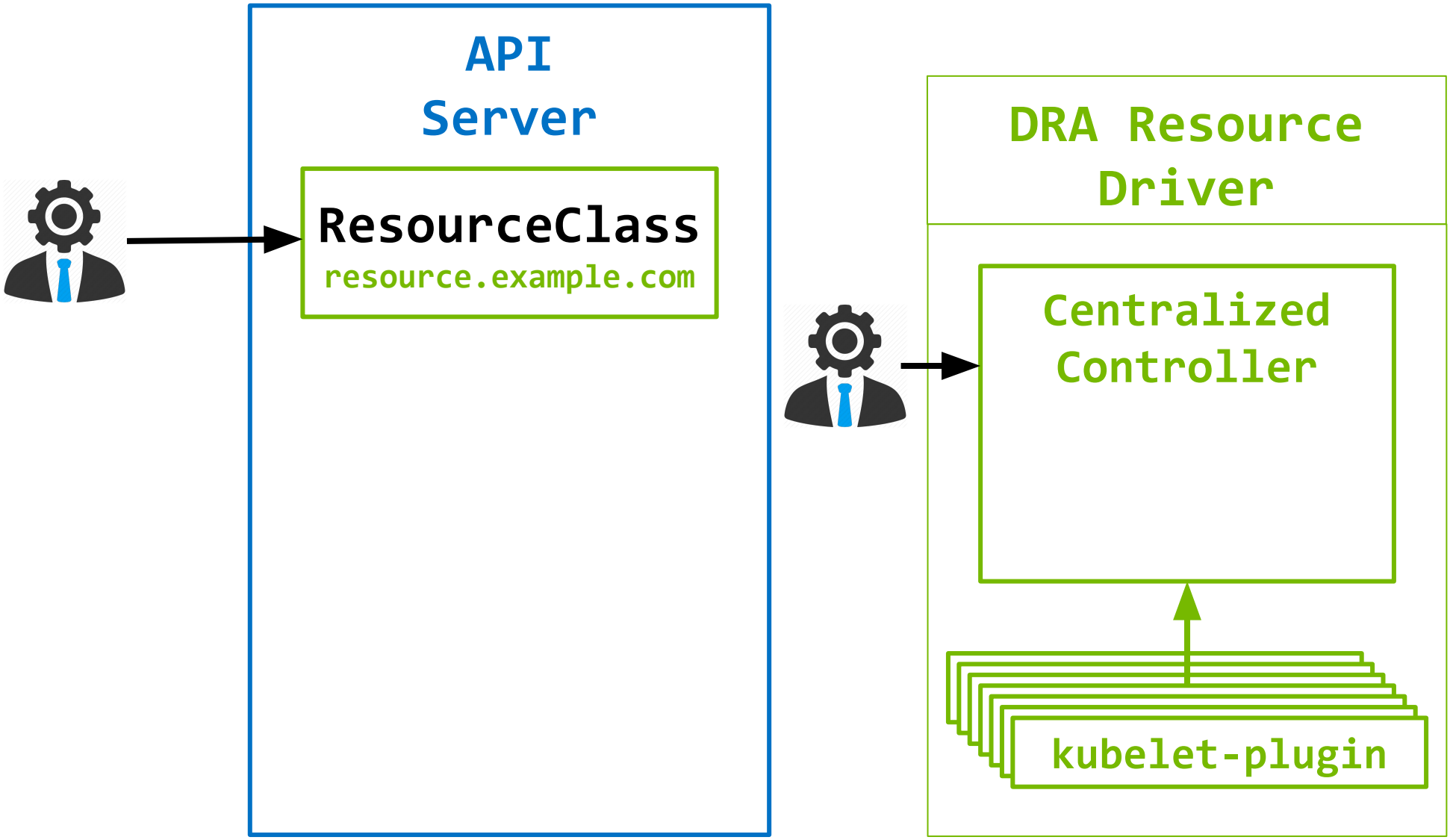
Delayed Allocation with DRA



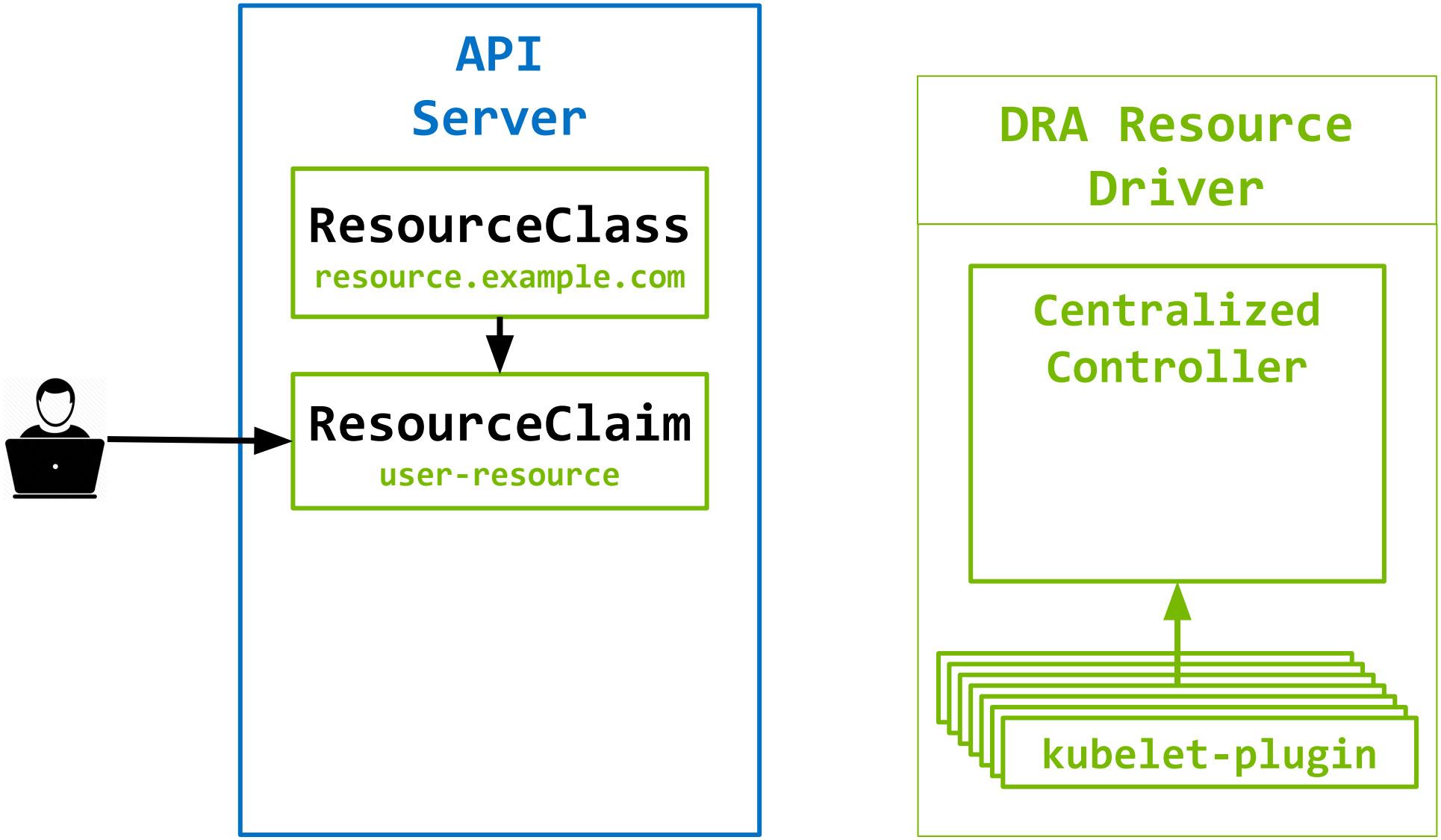
Immediate Allocation with DRA

Immediate

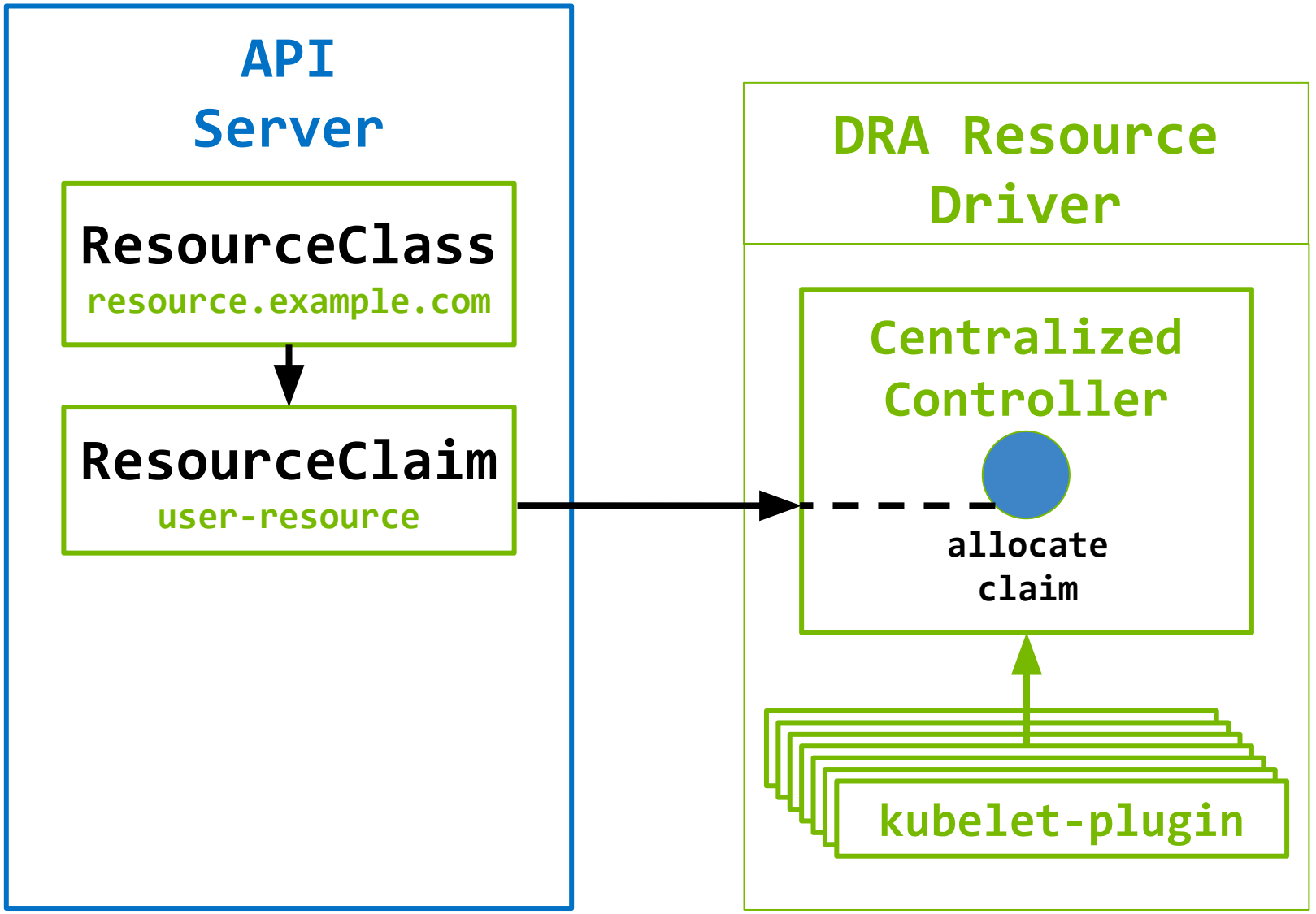
Immediate Allocation with DRA



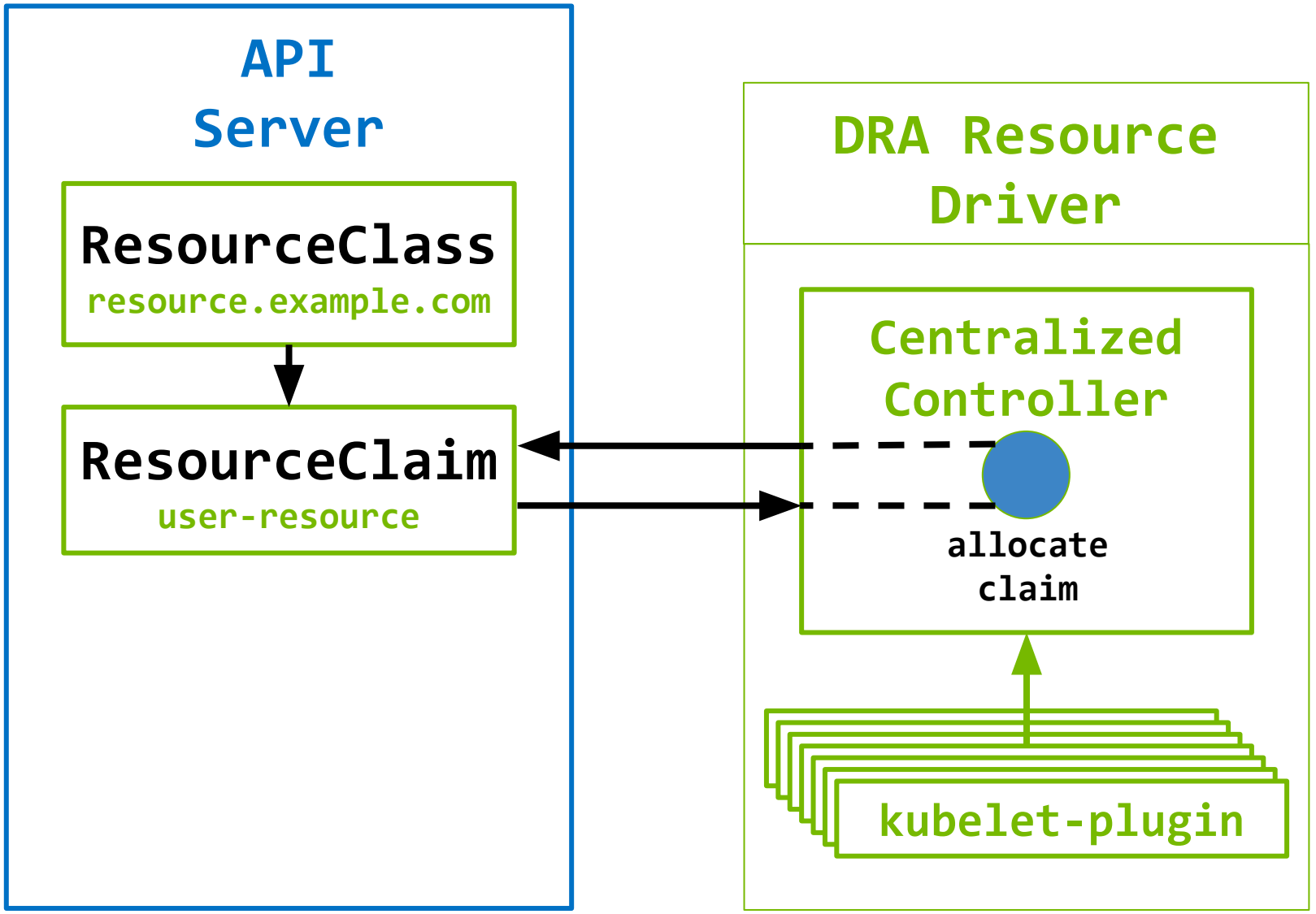
Immediate Allocation with DRA



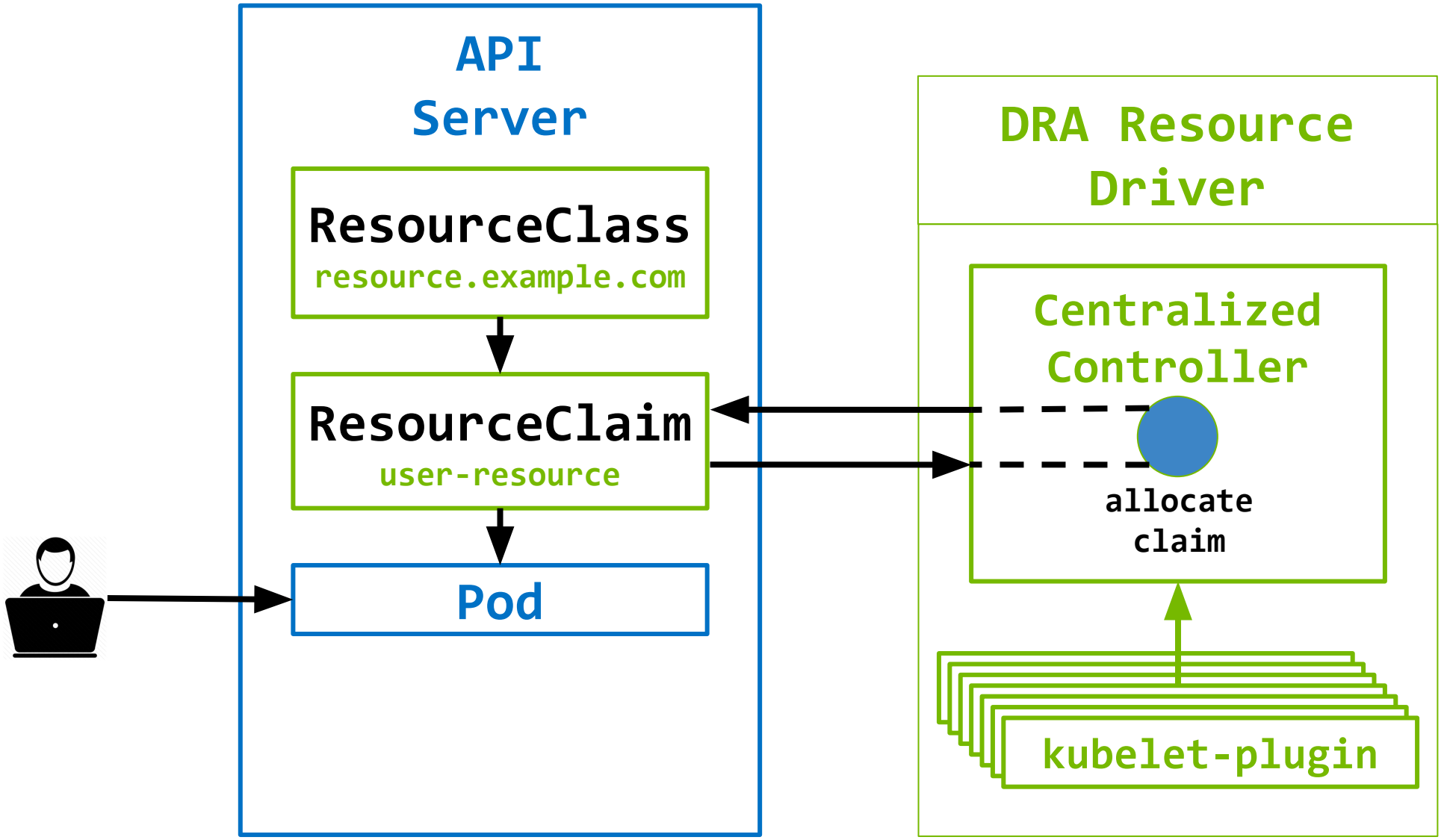
Immediate Allocation with DRA



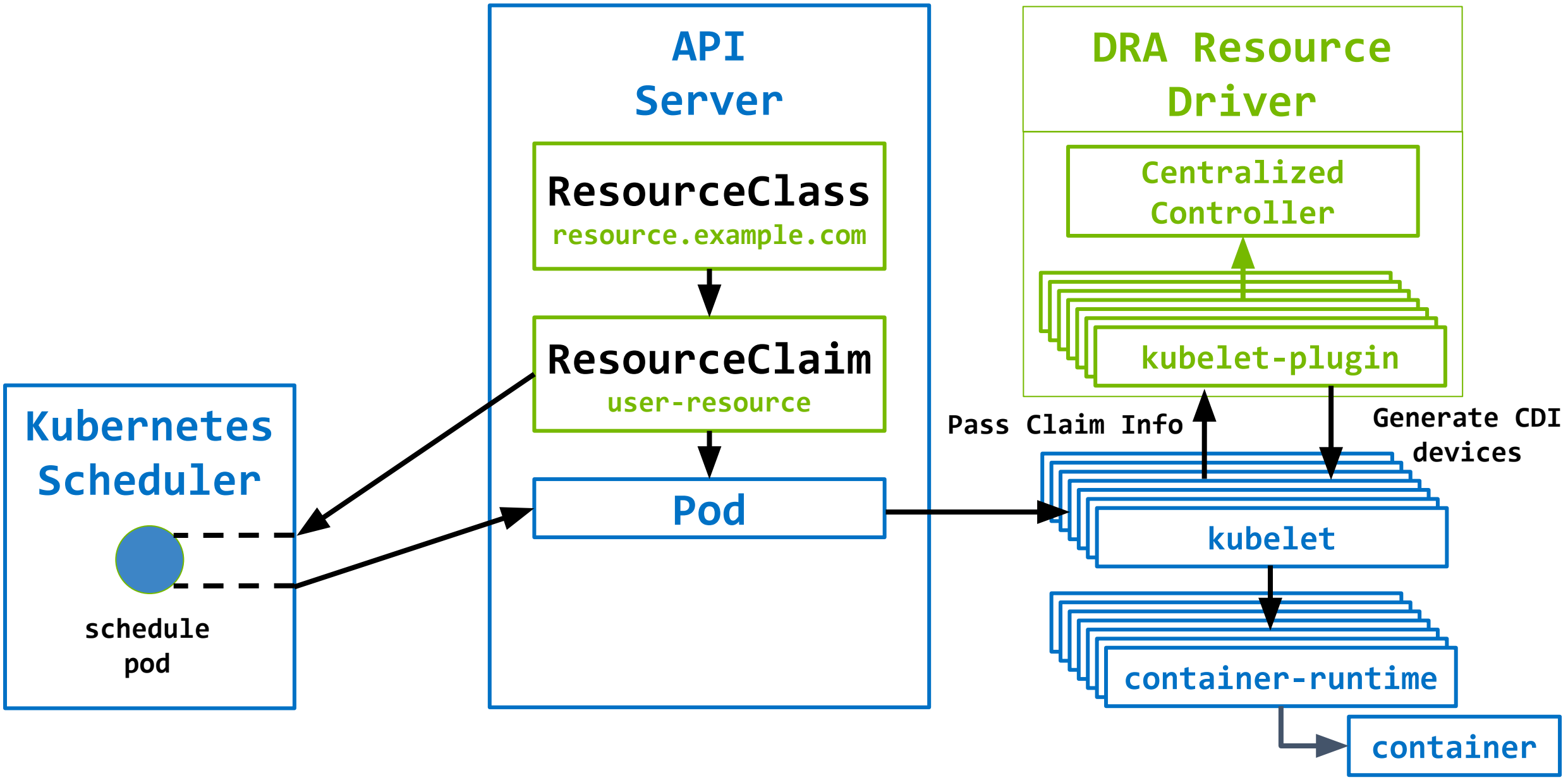
Immediate Allocation with DRA



Immediate Allocation with DRA

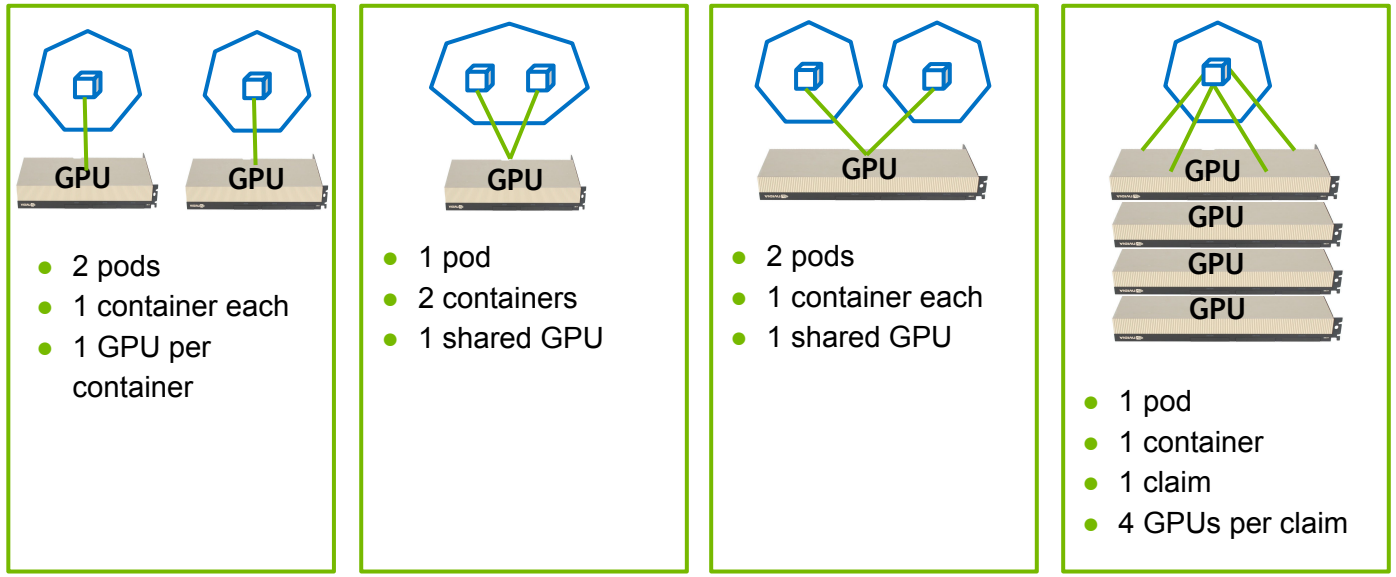


Immediate Allocation with DRA



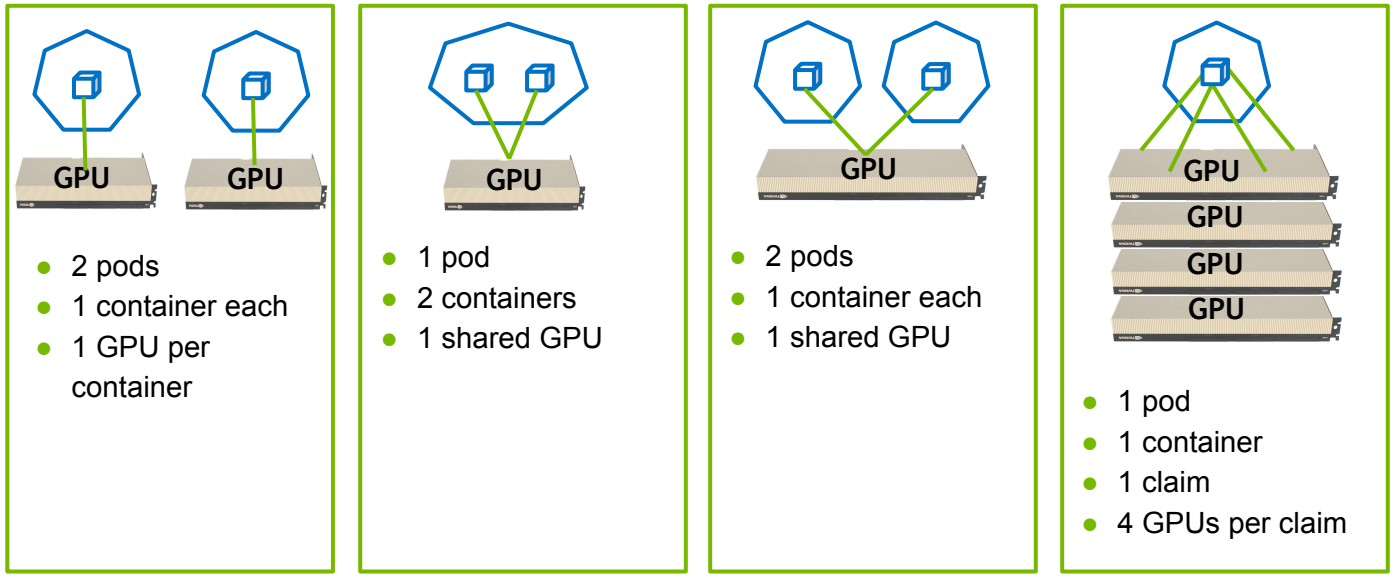
Build Your Own DRA Resource Driver

- Example DRA resource driver
  - https://github.com/kubernetes-sigs/dra-example-driver
  - Provides fully functional DRA resource driver on a set of mock GPUs
  - Wraps everything in a helm chart for ease of deployment
  - Provides scripts for bringing up a **`kind`** cluster to test in a multi-node setup
  - Runs on Mac OS and Linux without requiring specialized hardware
  - README includes a demo with four example deployments



Build Your Own DRA Resource Driver

- Example DRA resource driver
  - https://github.com/kubernetes-sigs/dra-example-driver
  - Provides fully functional DRA resource driver on a set of mock GPUs
  - Wraps everything in a helm chart for ease of deployment
  - Provides scripts for bringing up a **`kind`** cluster to test in a multi-node setup
  - Runs on Mac OS and Linux without requiring specialized hardware
  - README includes a demo with four example deployments



Build Your Own DRA Resource Driver

- In a Nutshell:
  - Decide on a name for your driver
  - Decide on a communication strategy (single purpose CRD vs. split-purpose communication)
  - Define types to represent your allocatable resources, allocated resources, and prepared resources
  - Define types to represent any `ClassParameters` you want for your resources
  - Define types to represent any `ClaimParameters` you want for your resources
  - Prepare at least one default `ResourceClass` for distribution with your resource driver
  - Write the boilerplate code to register your controller with the scheduler
  - Write the boilerplate code to register your kubelet-plugin with the kubelet
  - Write the business logic for your controller
  - Write the business logic for your kubelet-plugin

Build Your Own DRA Resource Driver

- In a Nutshell:
  - Decide on a name for your driver
  - Decide on a communication strategy (single purpose CRD vs. split-purpose communication)
  - Define types to represent your allocatable resources, allocated resources, and prepared resources
  - Define types to represent any `ClassParameters` you want for your resources
  - Define types to represent any `ClaimParameters` you want for your resources
  - Prepare at least one default `ResourceClass` for distribution with your resource driver
  - Write the boilerplate code to register your controller with the scheduler
  - Write the boilerplate code to register your kubelet-plugin with the kubelet
  - Write the business logic for your controller
  - Write the business logic for your kubelet-plugin

Build Your Own DRA Resource Driver

- Controller Helper Library
  - k8s.io/dynamic-resource-allocation/controller

Build Your Own DRA Resource Driver

- Controller Helper Library
  - k8s.io/dynamic-resource-allocation/controller

```
type Driver interface {
      GetClassParameters(context, class) (interface{}, error)
      GetClaimParameters(context, claim, class, classParameters) (interface{}, error)
      UnsuitableNodes(context, pod, []claimAllocation, []potentialNodes) error
      Allocate(context, claim, claimParameters, class, classParameters, selectedNode string) (allocationResult, error)
      Deallocate(context, claim) error
}
```

Build Your Own DRA Resource Driver

- Controller Helper Library
  - k8s.io/dynamic-resource-allocation/controller

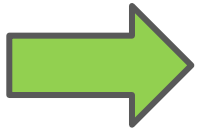
```
type Driver interface {
    GetClassParameters(context, class) (interface{}, error)
    GetClaimParameters(context, claim, class, classParameters) (interface{}, error)
    Deallocate(context, claim) error
}
```

Build Your Own DRA Resource Driver

- Controller Helper Library
  - k8s.io/dynamic-resource-allocation/controller

```
type Driver interface {
    GetClassParameters(context, class) (interface{}, error)
    GetClaimParameters(context, claim, class, classParameters) (interface{}, error)
    Deallocate(context, claim) error
}
```

```
apiVersion: resource.k8s.io/v1alpha2
kind: ResourceClass
metadata:
  name: gpu.nvidia.com
driverName: gpu.resource.nvidia.com
parametersRef:
    apiGroup: gpu.resource.nvidia.com
    kind: GpuClassParameters
    name: non-sharable
```

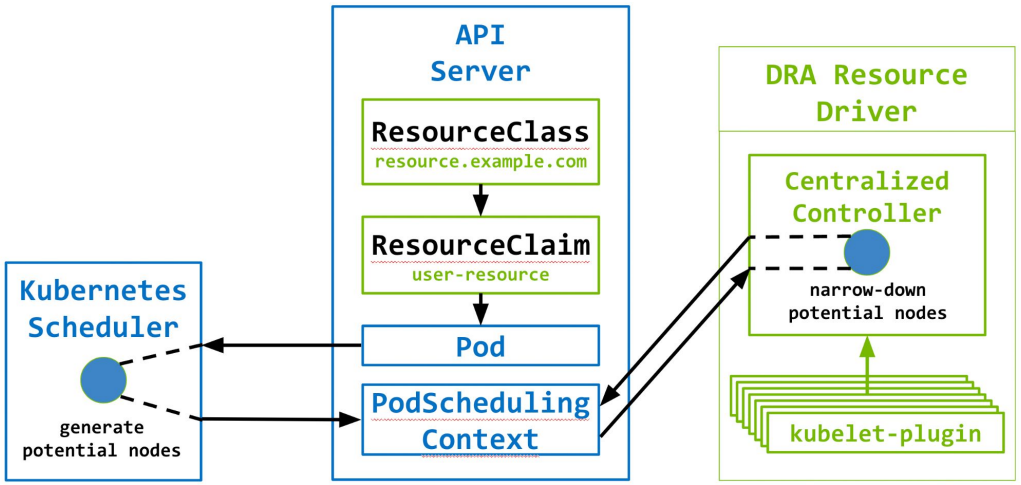


```
apiVersion: gpu.resource.nvidia.com/v1alpha1
kind: GpuClassParameters
metadata:
  name: non-sharable
spec:
  shareable: false
```

Build Your Own DRA Resource Driver

- Controller Helper Library
  - k8s.io/dynamic-resource-allocation/controller

```
type Driver interface {
    GetClassParameters(context, class) (interface{}, error)
    GetClaimParameters(context, claim, class, classParameters) (interface{}, error)
    UnsuitableNodes(context, pod, []claimAllocation, []potentialNodes) error
    Allocate(context, claim, claimParameters, class, classParameters, selectedNode string) (allocationResult, error)
    Deallocate(context, claim) error
}
```



Build Your Own DRA Resource Driver

- Controller Helper Library
  - k8s.io/dynamic-resource-allocation/controller

```
type Driver interface {
      GetClassParameters(context, class) (interface{}, error)
      GetClaimParameters(context, claim, class, classParameters) (interface{}, error)
      UnsuitableNodes(context, pod, []claimAllocation, []potentialNodes) error
      Allocate(context, claim, claimParameters, class, classParameters, selectedNode string) (allocationResult, error)
      Deallocate(context, claim) error
}
```

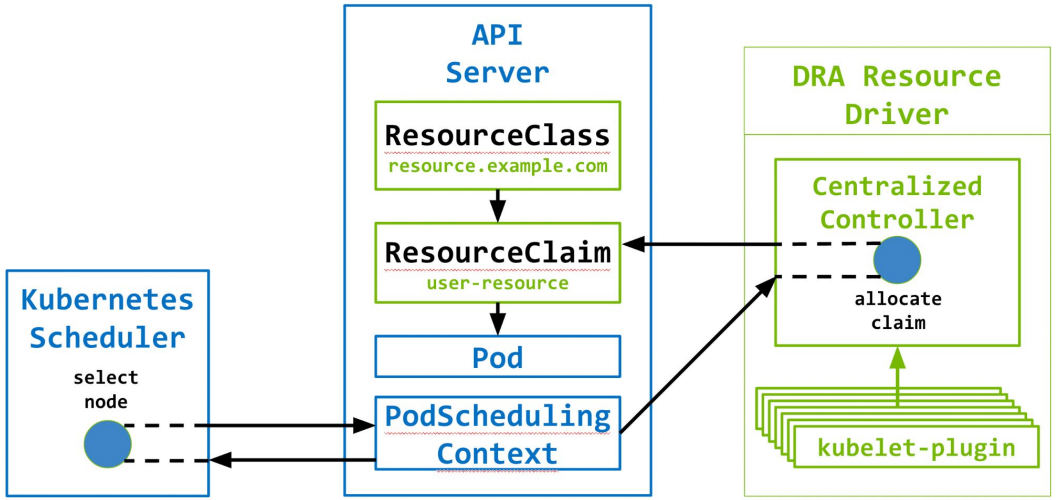
```
type ClaimAllocation struct {
      PodClaimName     string
      Claim            *resourcev1alpha2.ResourceClaim
      Class            *resourcev1alpha2.ResourceClass
      ClaimParameters  interface{}
      ClassParameters  interface{}
      UnsuitableNodes  []string
}
```

- Loop through **`potentialNodes`** in search of available resources
- Write back narrowed down list of nodes where resources *unavailable*

Build Your Own DRA Resource Driver

- Controller Helper Library
  - k8s.io/dynamic-resource-allocation/controller

```
type Driver interface {
    GetClassParameters(context, class) (interface{}, error)
    GetClaimParameters(context, claim, class, classParameters) (interface{}, error)
    Allocate(context, claim, claimParameters, class, classParameters, selectedNode string) (allocationResult, error)
    Deallocate(context, claim) error
}
```



Build Your Own DRA Resource Driver

- Controller Helper Library
  - k8s.io/dynamic-resource-allocation/controller

```
type Driver interface {
      GetClassParameters(context, class) (interface{}, error)
      GetClaimParameters(context, claim, class, classParameters) (interface{}, error)
      Allocate(context, claim, claimParameters, class, classParameters, selectedNode string) (allocationResult, error)
      Deallocate(context, claim) error
}
```

```
type AllocationResult struct {
      ResourceHandles []ResourceHandle
      AvailableOnNodes *v1.NodeSelector
      Shareable bool
}
```

Opaque data attached to claim to be passed to kubelet plugin(s) for interpretation

Build Your Own DRA Resource Driver

- Controller Helper Library
  - k8s.io/dynamic-resource-allocation/controller

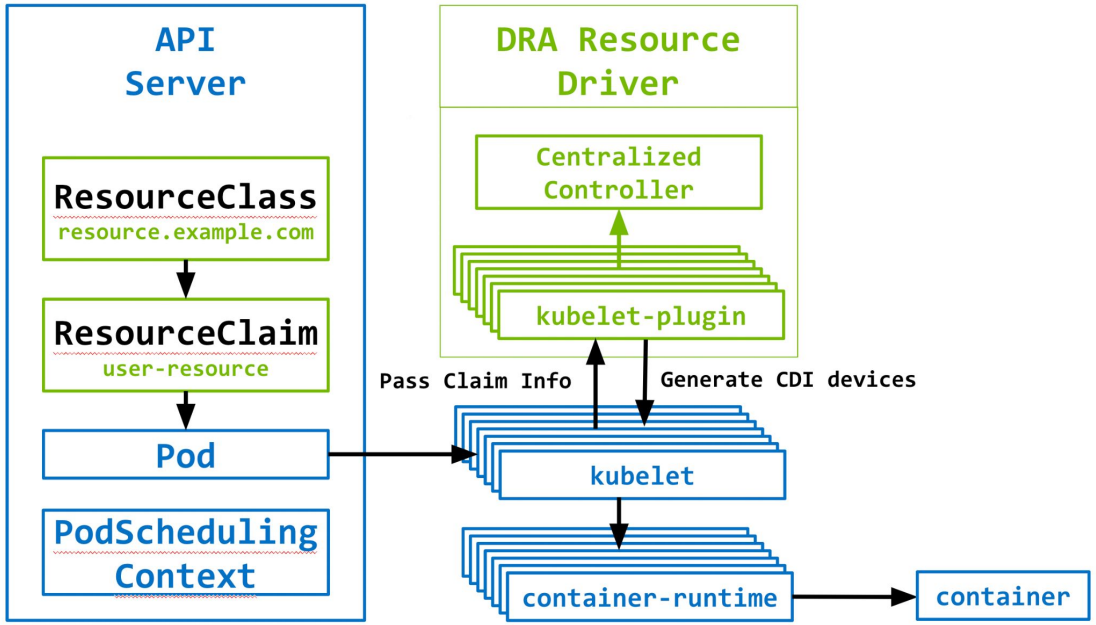
```
type Driver interface {
      GetClassParameters(context, class) (interface{}, error)
      GetClaimParameters(context, claim, class, classParameters) (interface{}, error)
      Allocate(context, claim, claimParameters, class, classParameters, selectedNode string) (allocationResult, error)
      Deallocate(context, claim) error
}
```

Deallocate(context, claim) error

Build Your Own DRA Resource Driver

- Kubelet Plugin API and Helper Library
  - k8s.io/kubelet/pkg/apis/dra/v1alpha2
  - k8s.io/dynamic-resource-allocation/kubeletplugin

```
type NodeServer interface {
      NodePrepareResource(context.Context, *NodePrepareResourceRequest) (*NodePrepareResourceResponse, error)
      NodeUnprepareResource(context.Context, *NodeUnprepareResourceRequest) (*NodeUnprepareResourceResponse, error)
}
```



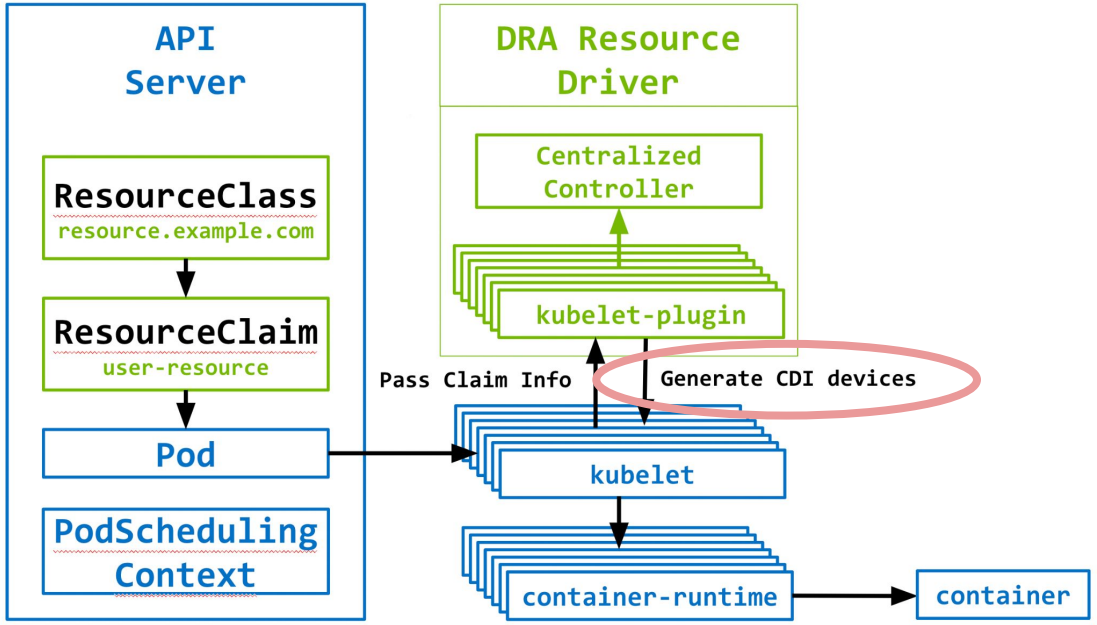
Build Your Own DRA Resource Driver

- Kubelet Plugin API and Helper Library
  - k8s.io/kubelet/pkg/apis/dra/v1alpha2
  - k8s.io/dynamic-resource-allocation/kubeletplugin

```
type NodeServer interface {
      NodePrepareResource(context.Context, *NodePrepareResourceRequest) (*NodePrepareResourceResponse, error)
      NodeUnprepareResource(context.Context, *NodeUnprepareResourceRequest) (*NodeUnprepareResourceResponse, error)
}
```

```
type NodePrepareResourceRequest struct {
      Namespace       string
      ClaimUid        string
      ClaimName       string
      ResourceHandle string
}

type NodePrepareResourceResponse struct {
      CdiDevices              []string
}
```



Build Your Own DRA Resource Driver

- Kubelet Plugin API and Helper Library
  - k8s.io/kubelet/pkg/apis/dra/v1alpha2
  - k8s.io/dynamic-resource-allocation/kubeletplugin

```
type NodeServer interface {
      NodePrepareResource(context.Context, *NodePrepareResourceRequest) (*NodePrepareResourceResponse, error)
      NodeUnprepareResource(context.Context, *NodeUnprepareResourceRequest) (*NodeUnprepareResourceResponse, error)
}
```

```
type NodePrepareResourceRequest struct {
      Namespace      string
      ClaimUid       string
      ClaimName      string
      ResourceHandle string
}

type NodePrepareResourceResponse struct {
      CdiDevices          []string
}
```

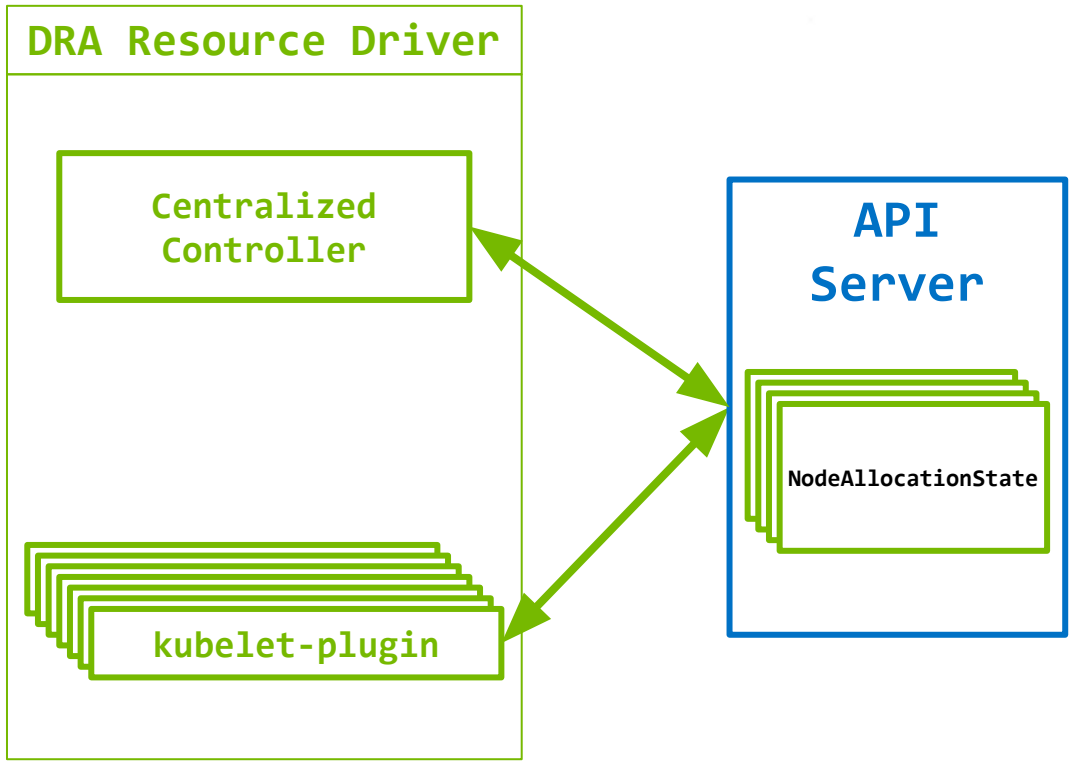
```
type NodeUnprepareResourceRequest struct {
      Namespace      string
      ClaimUid       string
      ClaimName      string
      ResourceHandle string
}

type NodeUnprepareResourceResponse struct {}
```

New And Upcoming Features

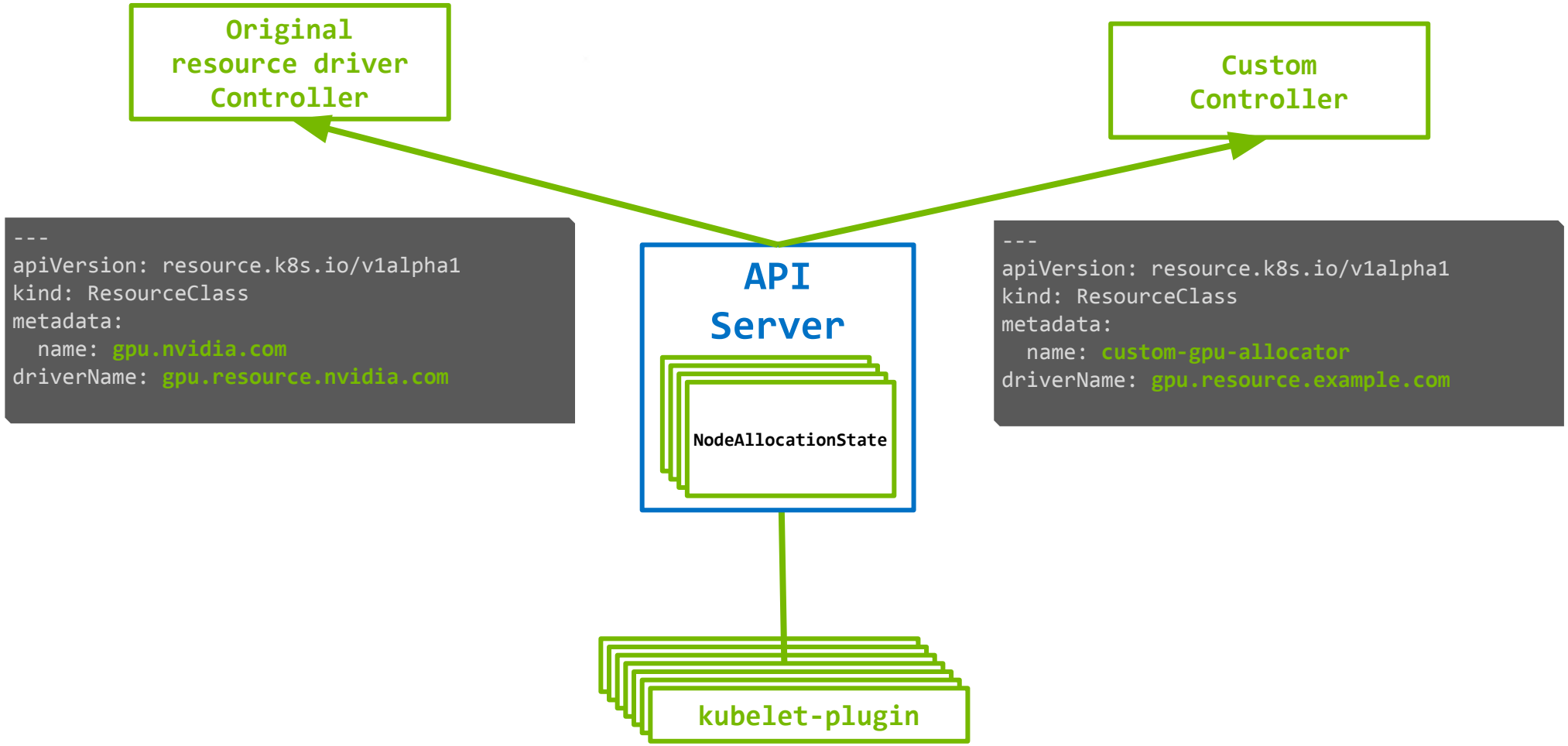
- Custom resource driver controllers for joint allocation of multiple kinds of devices
- Batching of resource claims for resource driver API

Custom DRA Controllers

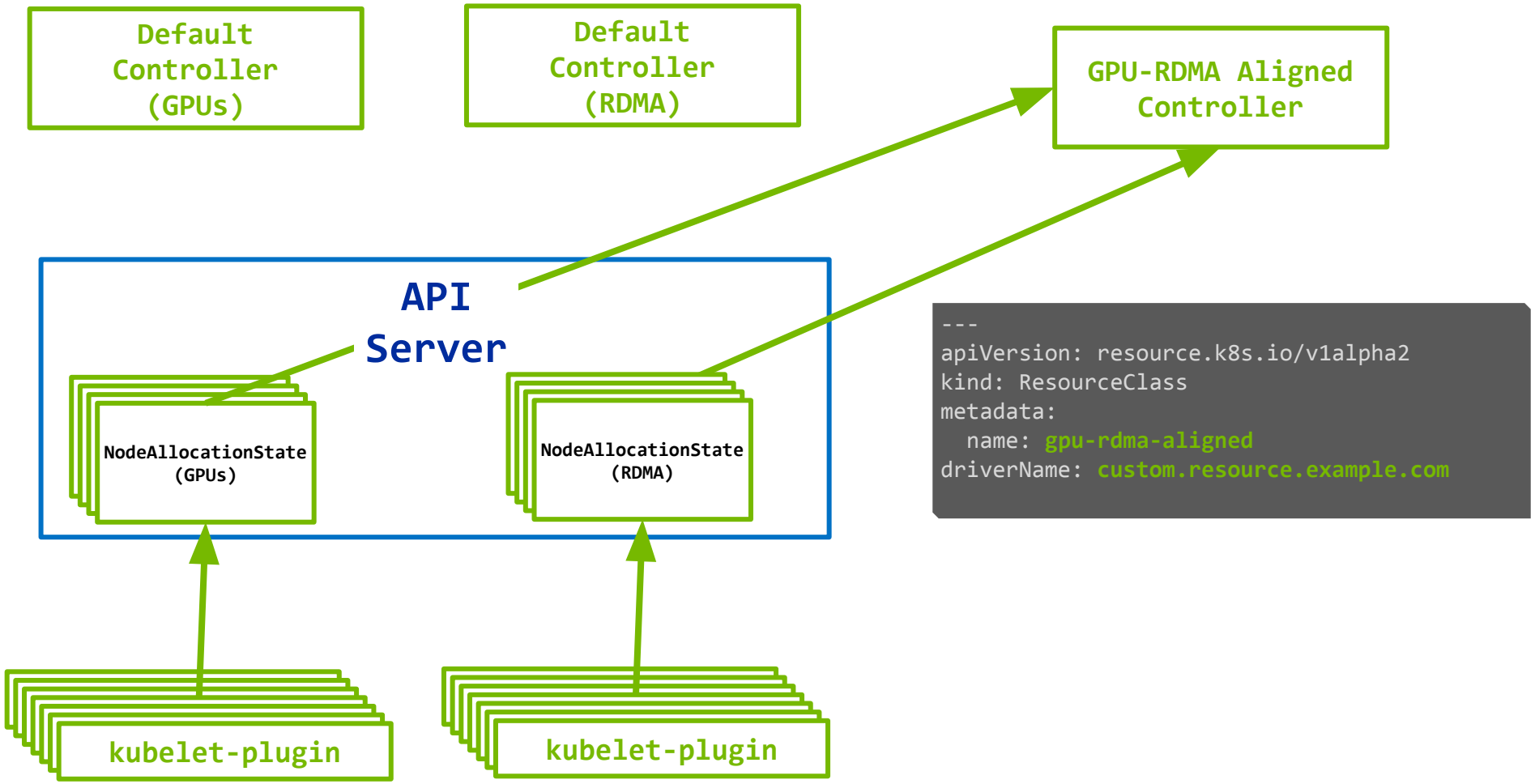


Bookkeeping has to be in CRD if resource driver's device is expected to be allocated by other resource driver controllers. E.g. GPU & NIC

Custom DRA Controllers

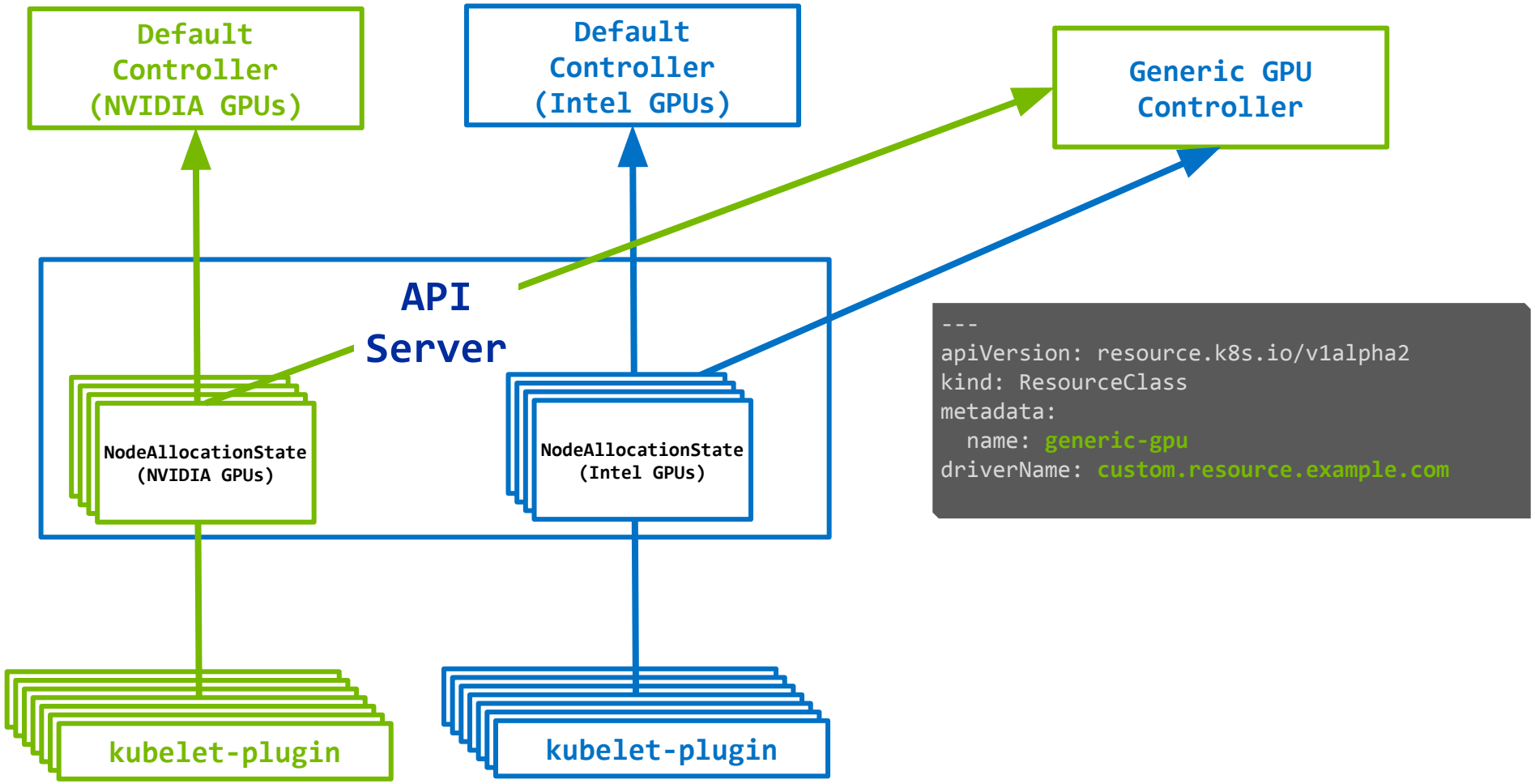


Custom DRA Controllers



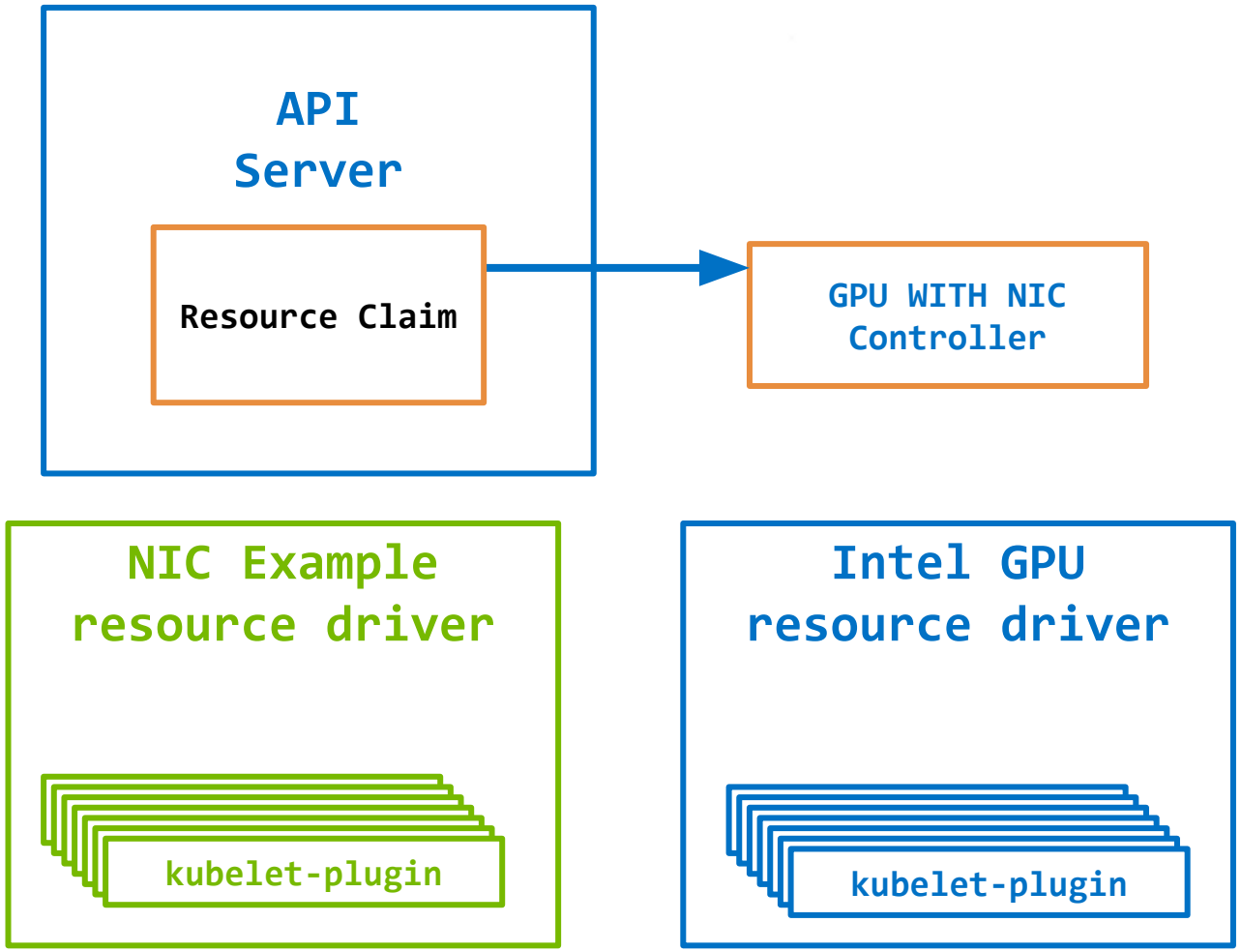

```
---
apiVersion: resource.k8s.io/v1alpha2
kind: ResourceClass
metadata:
  name: gpu-rdma-aligned
driverName: custom.resource.example.com
```

Custom DRA Controllers



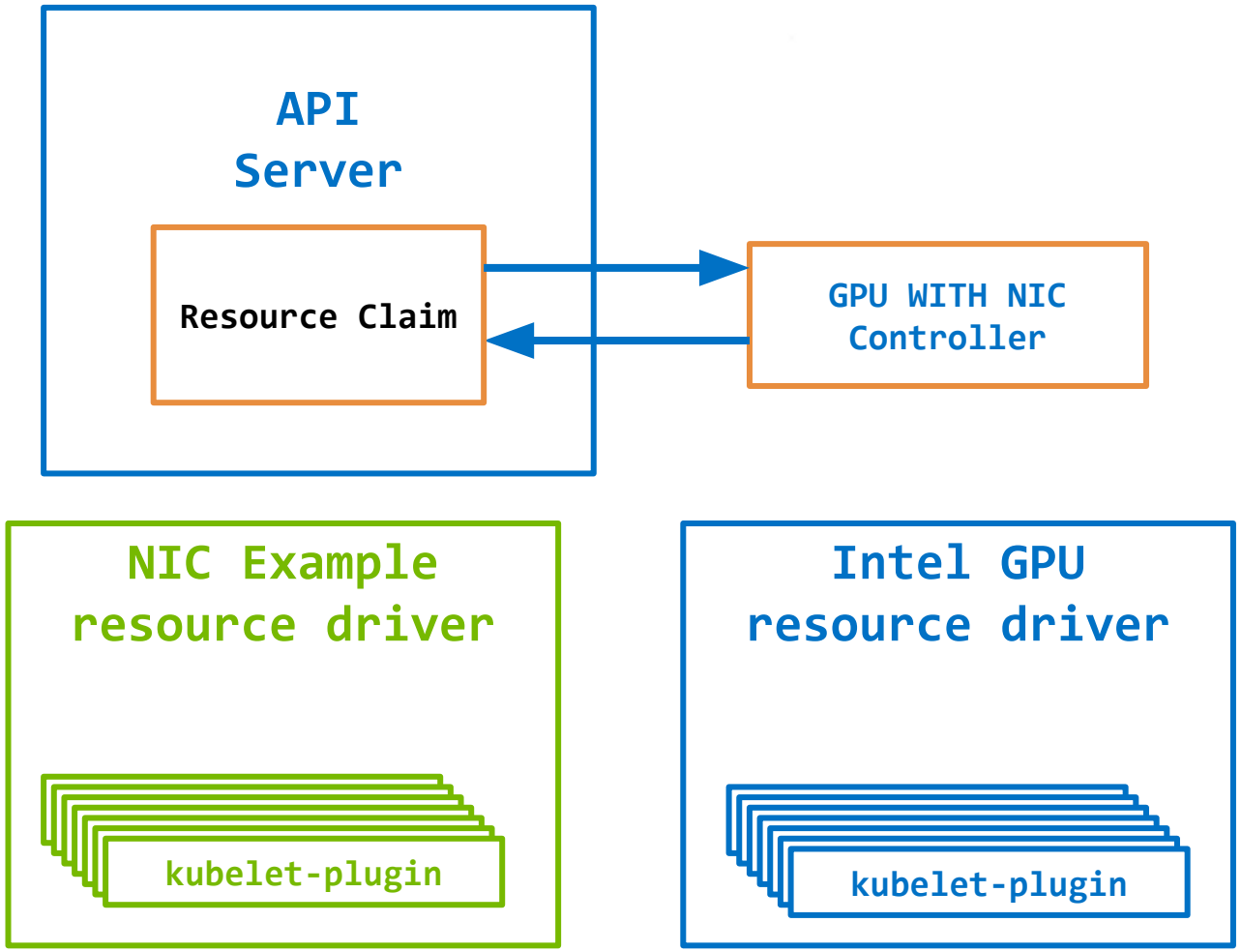

```
---
apiVersion: resource.k8s.io/v1alpha2
kind: ResourceClass
metadata:
  name: generic-gpu
driverName: custom.resource.example.com
```

Custom DRA Controllers



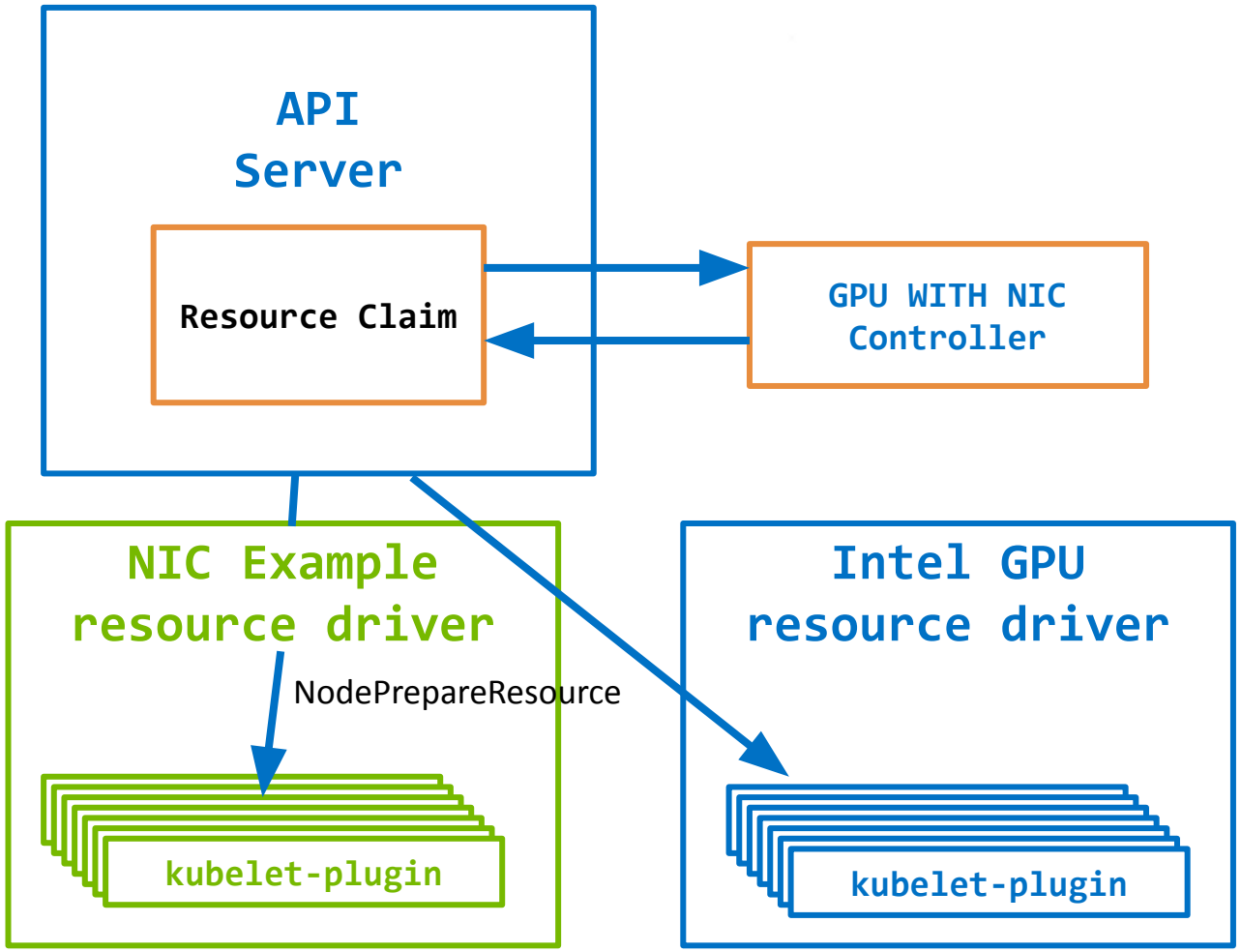

```
---
apiVersion: resource.k8s.io/v1alpha2
kind: ResourceClass
metadata:
  name: gpu-with-nic
driverName: custom.resource.example.com
```

Custom DRA Controllers




```
AllocationResult {
    ResourceHandles []ResourceHandle{
        ResourceHandle {
            DriverName: gpu.resource.intel.com,
            Data: “...”,
        },
        ResourceHandle {
            DriverName: nic.resource.example.com,
            Data: “...”,
        },
    },
    AvailableOnNodes &v1.NodeSelector{...},
    Shareable: <bool>,
}
```

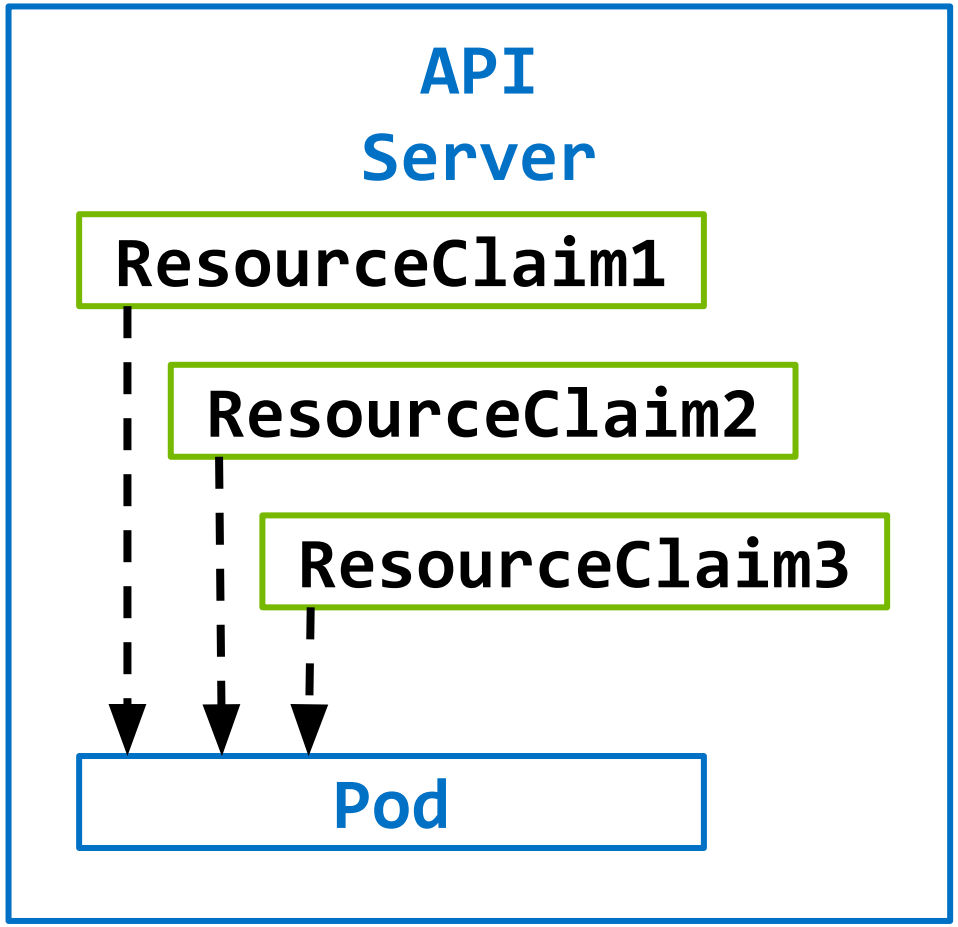
Custom DRA Controllers




```
AllocationResult {
    ResourceHandles []ResourceHandle{
        ResourceHandle {
            DriverName: gpu.resource.intel.com,
            Data: “...”,
        },
        ResourceHandle {
            DriverName: nic.resource.example.com,
            Data: “...”,
        },
    },
    AvailableOnNodes &v1.NodeSelector{...},
    Shareable: <bool>,
}
```

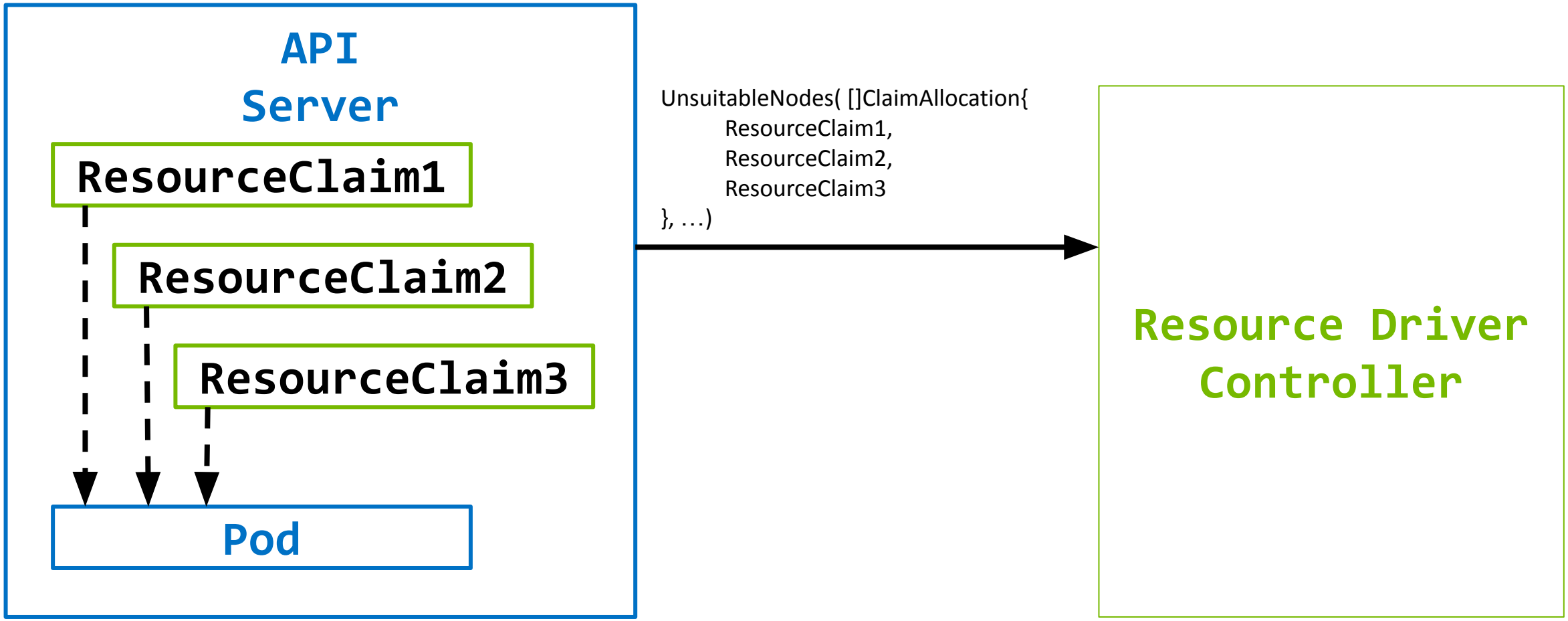
Resource driver API change in K8s 1.28

Now (K8s 1.26+)



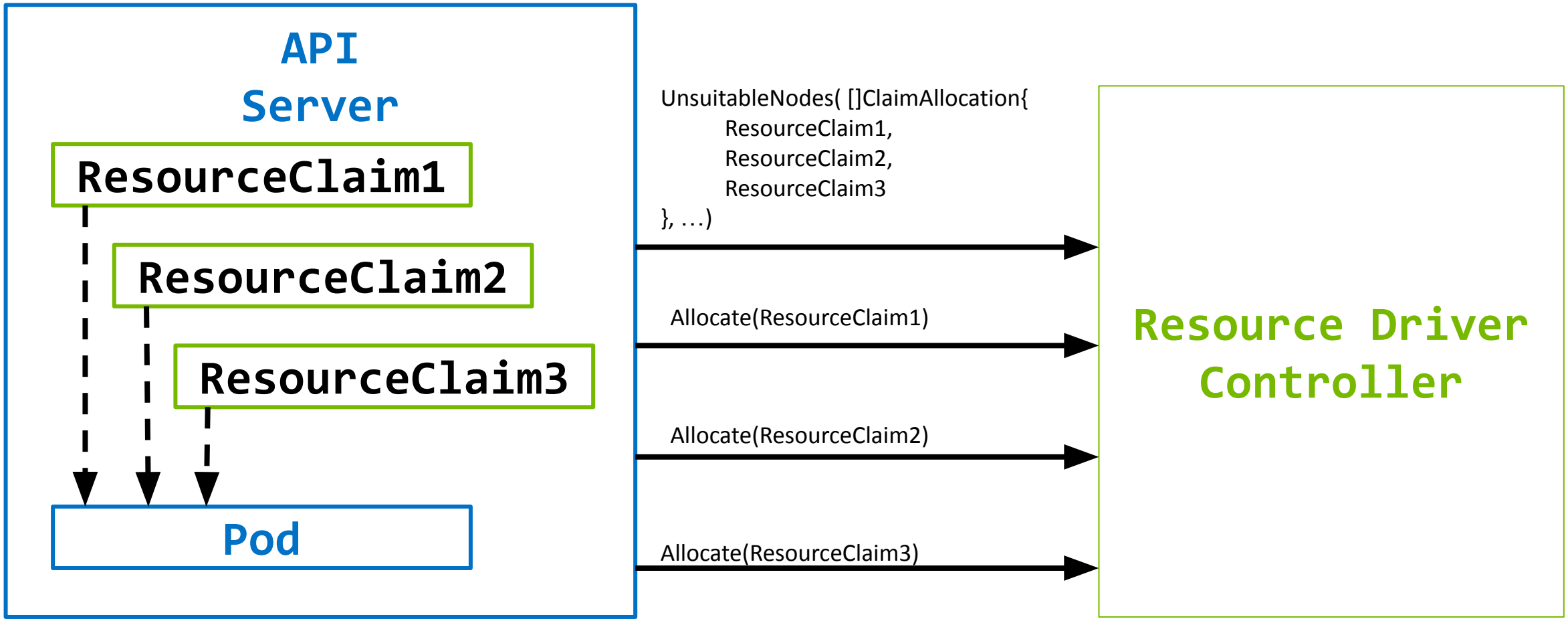
Resource driver API change in K8s 1.28

Now (K8s 1.26+)



Resource driver API change in K8s 1.28

Now (K8s 1.26+)



Resource driver API change in K8s 1.28

The problem: potential resources availability change

- Initial calculation done for list of claim allocations
- The actual allocation is done for single claim allocation
- Either disregard the problem and handle in kubelet plugin
- Or keep track of Pod owning the resource claim

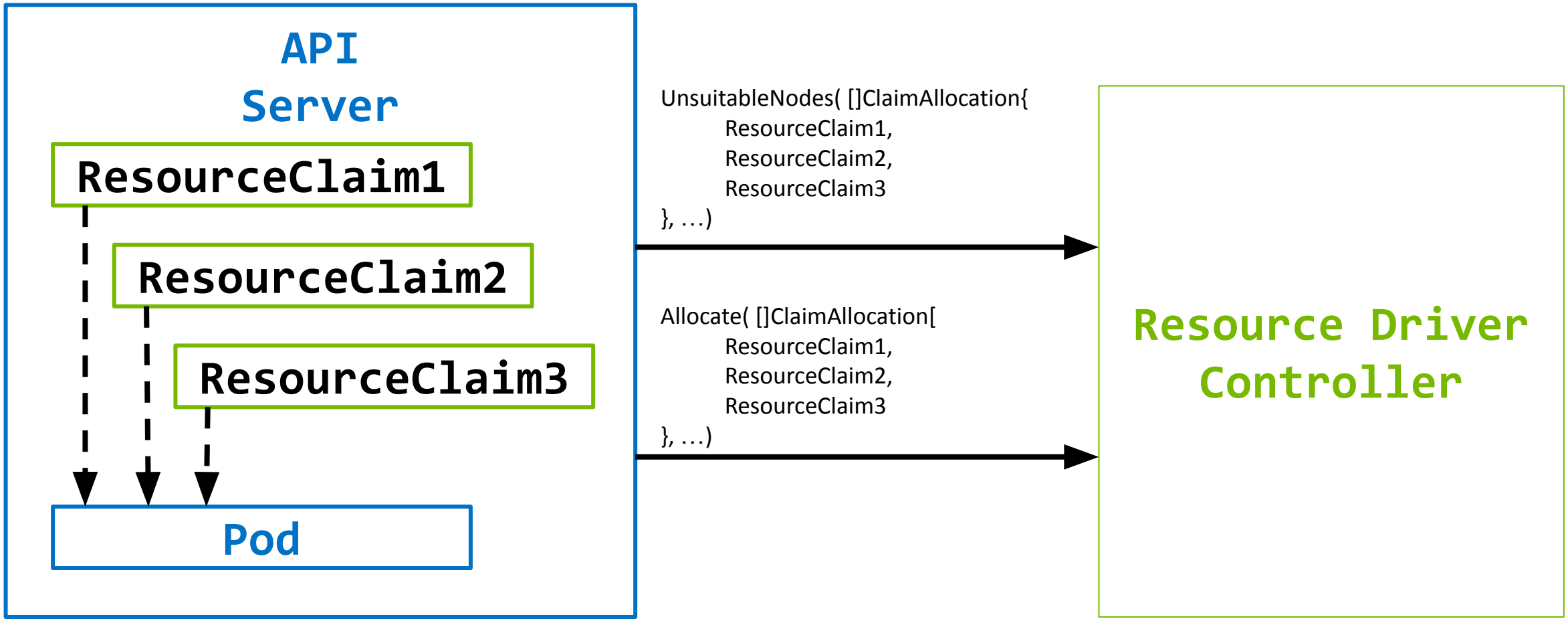
The solution: call allocation for a list of claim allocations

- Improves allocation decision making in certain situations
- Decreases allocation complexity: no need to track owner

Resource driver API change in K8s 1.28

KubeCon CloudNativeCon Europe 2023

Proposal (K8s 1.28+)



Resource driver API change in K8s 1.28

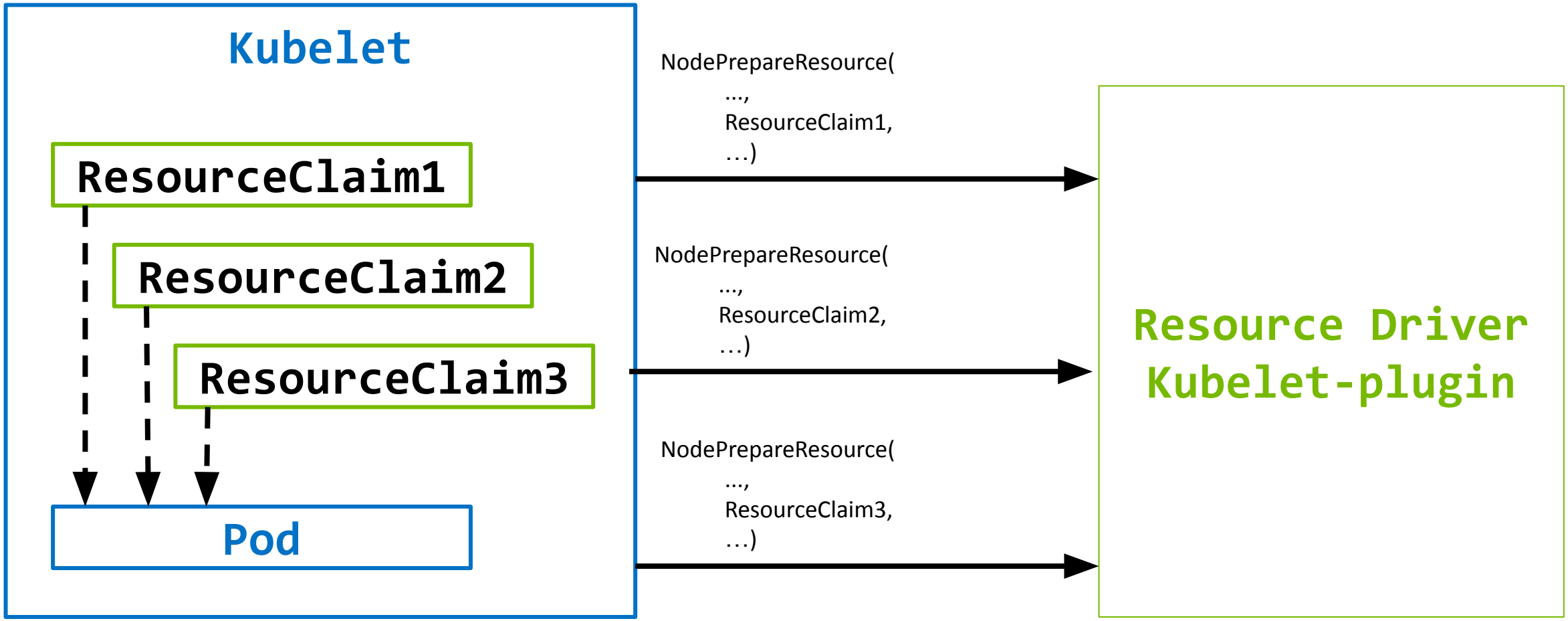
Same for Kubelet-plugin

- Some HW allows only one immutable configuration: preparing one resource claim on such HW would prohibit preparing another resource claim that controller allocated to the same HW - all resource claims allocated / planned for such HW have to be prepared at the same time

- Unpreparing is easier - resource can be tracked internally as unused until all the configurations applied are out of use - then all configurations can be removed at the same time

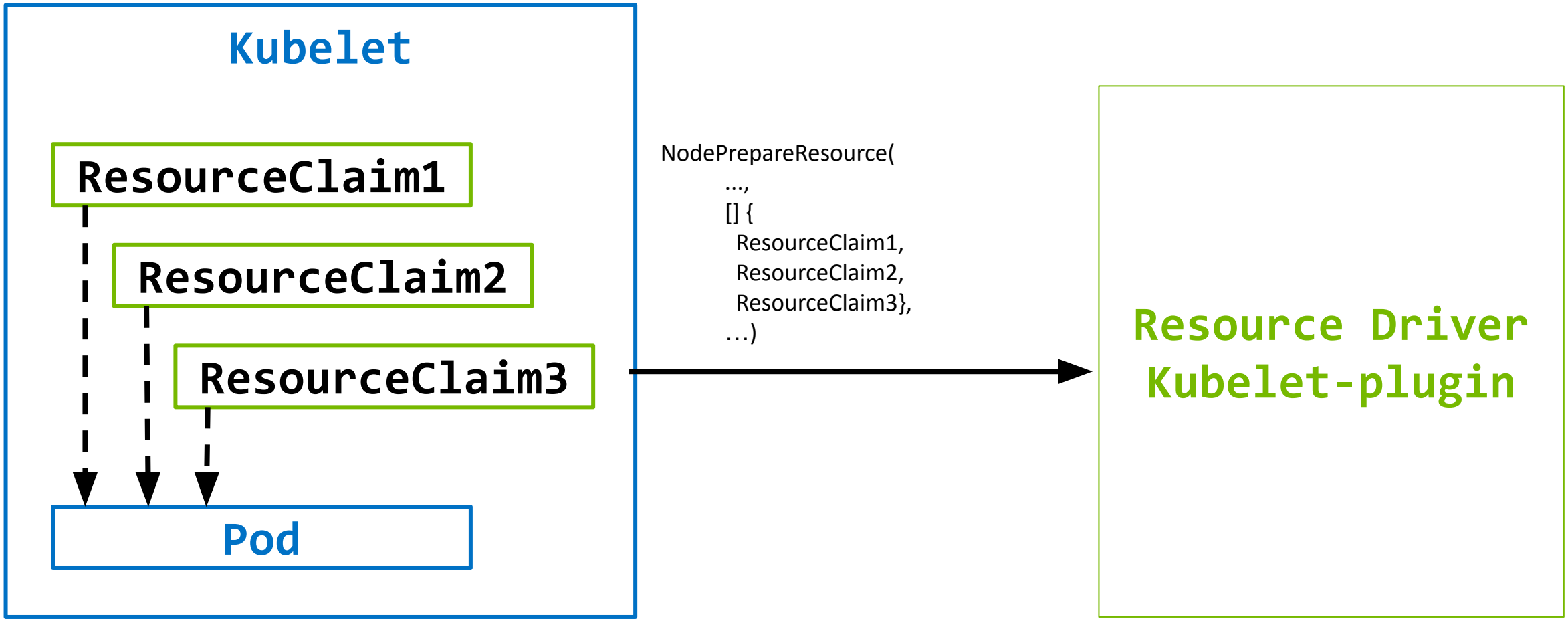
Resource driver API change in K8s 1.28

Now (K8s 1.26+)



Resource driver API change in K8s 1.28

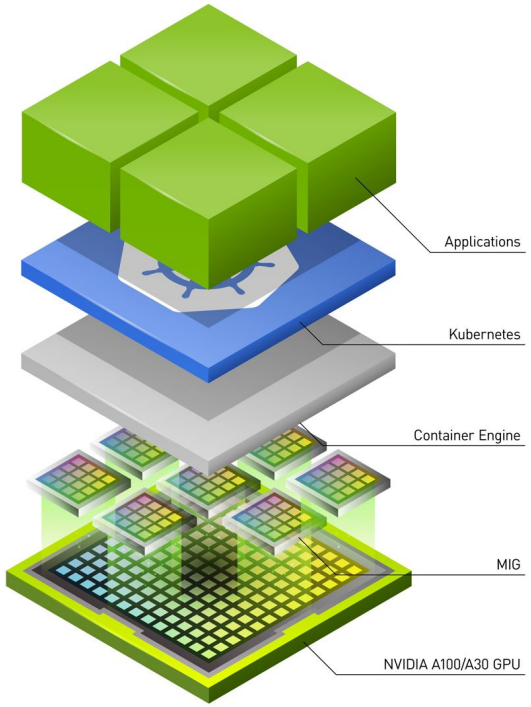
Proposal (K8s 1.28+)



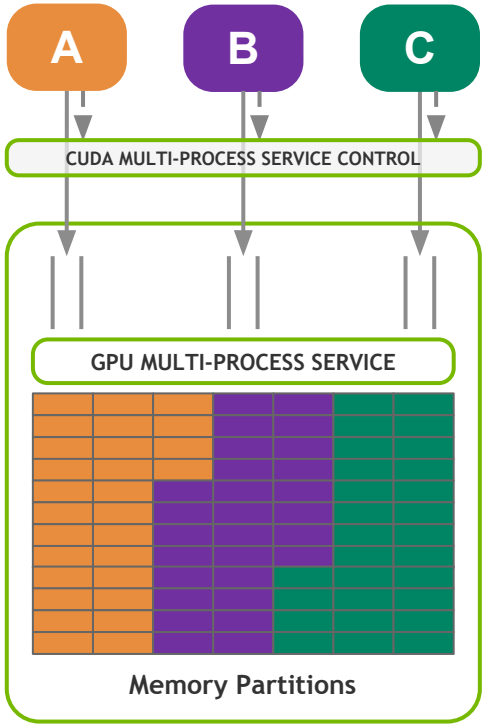
Demo on NVIDIA GPUs

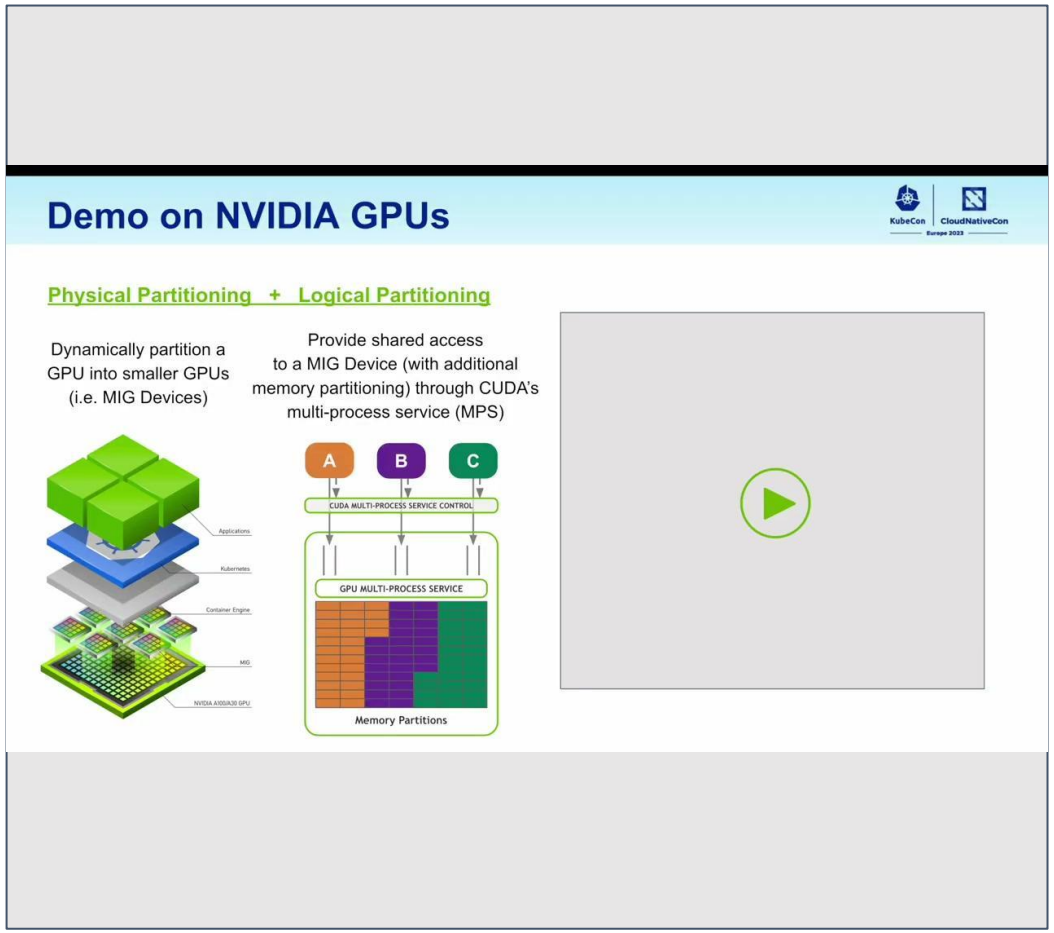
Physical Partitioning + Logical Partitioning

Dynamically partition a GPU into smaller GPUs (i.e. MIG Devices)



Provide shared access to a MIG Device (with additional memory partitioning) through CUDA's multi-process service (MPS)

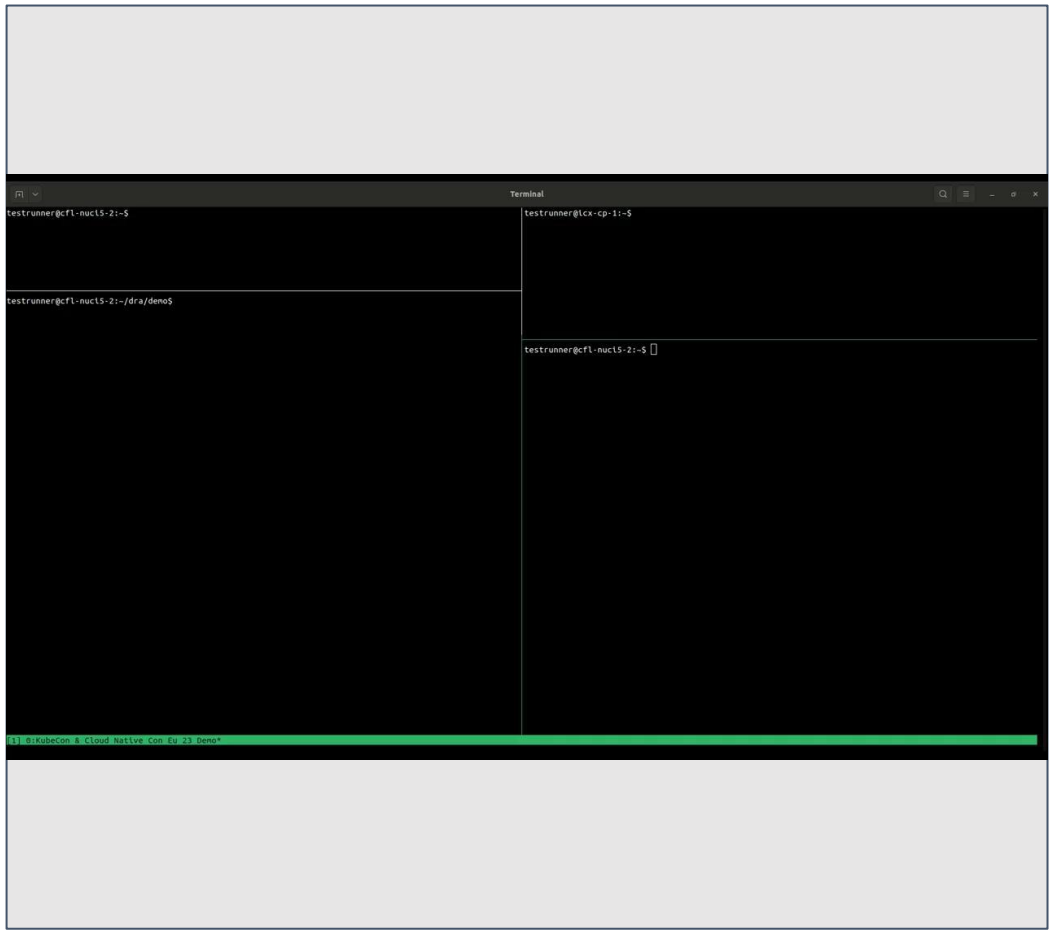




Demo on Intel GPUs

K8s 1.26
CRI-O
GPU resource driver v0.1.0-beta

Dynamic SR-IOV Virtual Functions for memory partitioning and isolation



Feedback & Questions

KubeCon CloudNativeCon Europe 2023

Please scan the QR Code below to leave feedback on this session

- Dynamic Resource Allocation Roadmap
  - Kubernetes 1.25 - KEP accepted
  - Kubernetes 1.26 - Alpha1 Release
  - Kubernetes 1.27 - Alpha2 Release
  - Kubernetes 1.29 - Beta Release
  - TBD - GA Release
- Shout Outs
  - Patrick Ohly (Intel)
  - Alexander Kanevskiy (Intel)
  - Evan Lezar (NVIDIA)
  - Ed Bartosh (Intel)
  - Krisztian Litkey (Intel)
- Resources:
  - Dynamic Resource Allocation KEP
  - Example DRA Resource Driver
  - NVIDIA DRA resource driver for GPUs
  - Intel DRA resource driver for GPUs


KubeCon
CloudNativeCon
Europe 2023
TiKV