
KubeCon
CloudNativeCon
North America 2023

Operator design for HPC:
patterns for orchestrating large scale compute intensive applications

Luca Montechiesi
Min Tsao

About us

SIEMENS *Luca Montechiesi*

Senior Software Engineer
Calibre Advanced Infrastructure team

https://www.linkedin.com/in/lucamontechiesi/

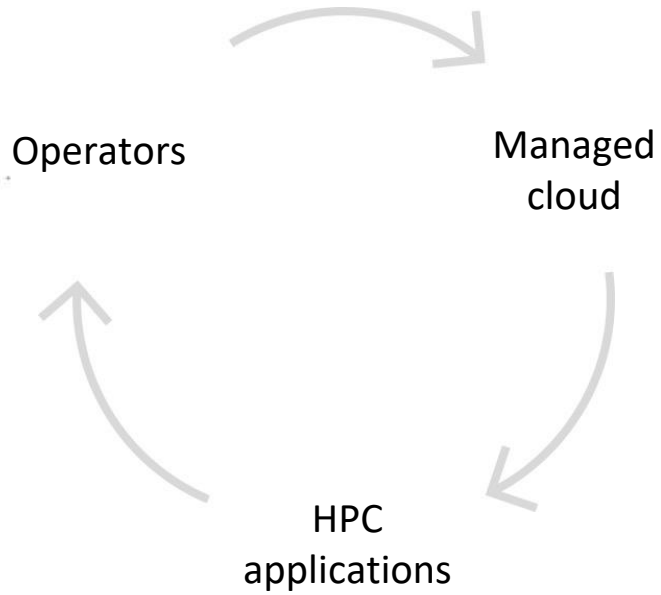
https://github.com/lumontec

https://lumontec.com/

SIEMENS *Min Tsao*

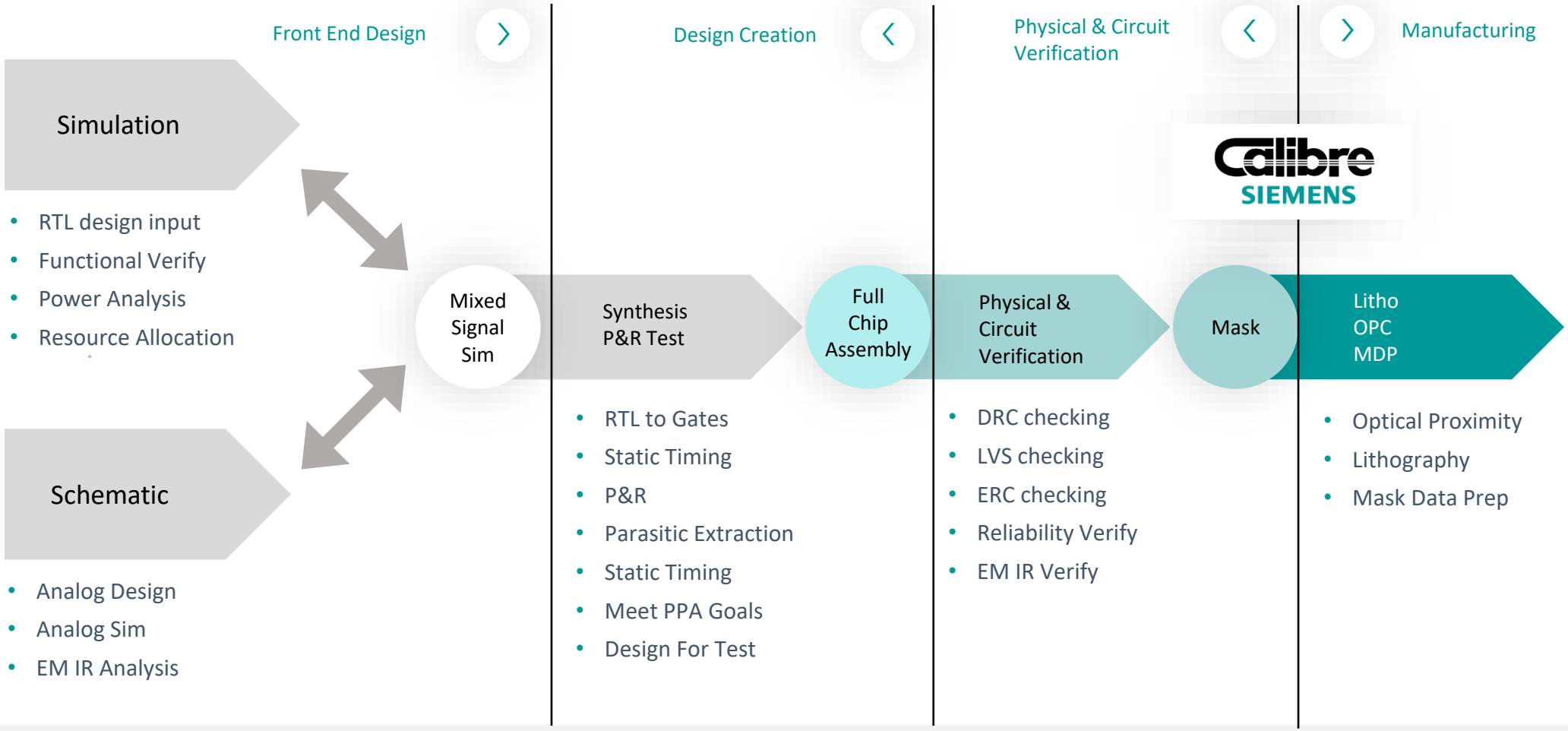
Engineering Director
Calibre Advanced Infrastructure team

https://www.linkedin.com/in/mintsao/



1. Overview of HPC/EDA applications/workloads
2. Designing controllers for HPC on managed cloud
3. Optimizations and performance

A Typical IC Design Flow And EDA Applications



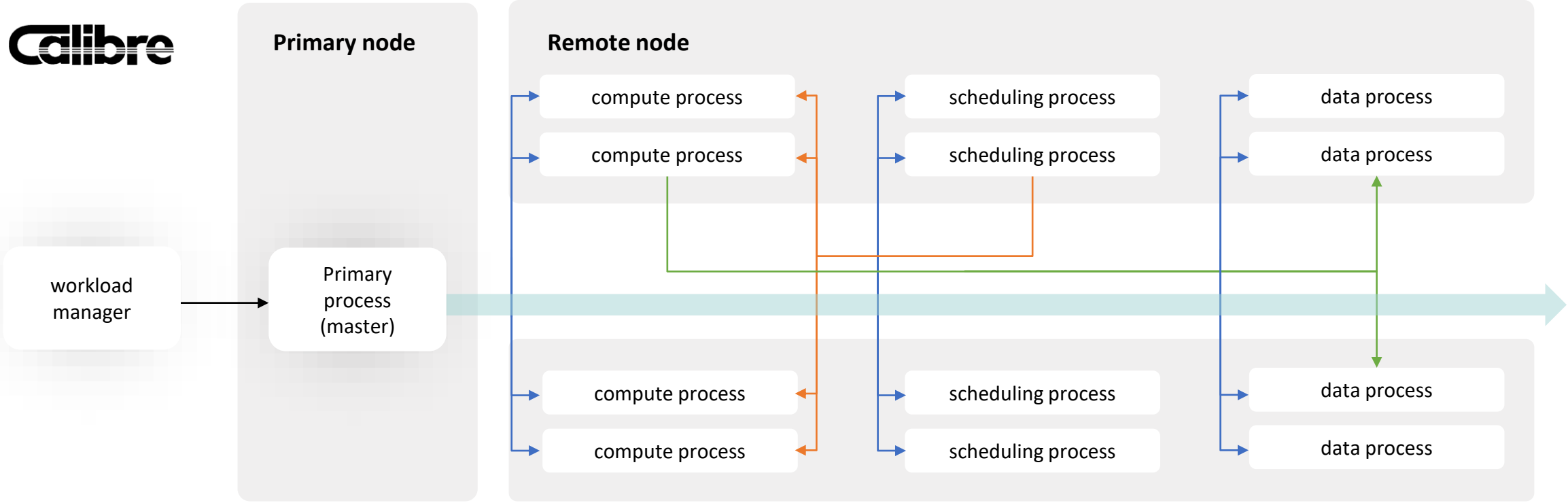
- Interactive editing
- On-demand simulation
- Data analysis

HPC

- Batch run with huge demand of computing
- Complex distributed architecture
- Tight coupling between components

Some numbers:

- Distributed runs up to **20000** cores per job
- Can allocate up to **1TB** RAM on primary node
- Burst network utilization up to **10 Gpbs**



Massive distributed parallel processing software

- Basic use model is “batch-run”. Compute and memory intensive.
- Many different components that serve different roles in a single run. Lots of network traffic.
- Different components have different levels of criticality. Some can “afford” to fail. Some cannot.

Built in dynamic resource control

- Built in logic of scheduling and orchestration
- Dynamic resource control. It will make its own decisions of acquiring and releasing resources.

Long turn around time

- Usually, a single run takes hours to days.
- Failed run needs to be repeated which impacts user’s productivity.
- Observability is very important.

Operators for HPC:

- Implement workload orchestration to satisfy internal state machine
- Abstract batch submission and configuration
- Expose k8s scheduling primitives
- Implement flexible workload observability spec

Critical challenges:

- Heaps of processes = heaps of pods and containers = heavy on k8s api-server / etcd
- HPC = compute, memory cpu intensive = heavy on infrastructure and hardware

Just lift and shift !

Mapping application state to k8s

Stateful application orchestration => Controller logic depends on **INTERNAL APPLICATION STATE**

Job controller:

-watch, build and cache the current state for a set of k8s resources

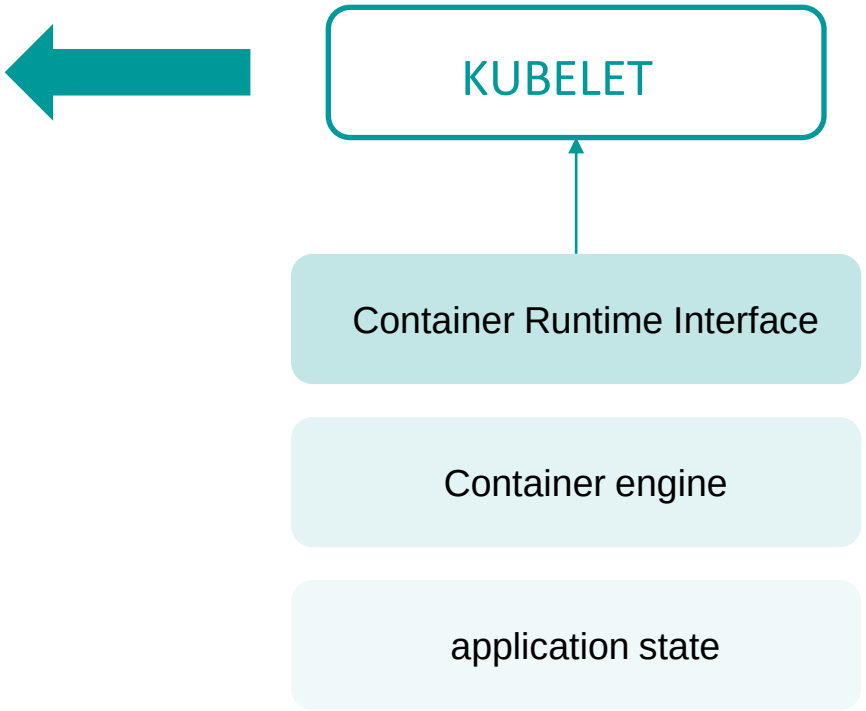
-reconcile against the desired state

-stateless and idempotent

Kubelets solve the same problem !

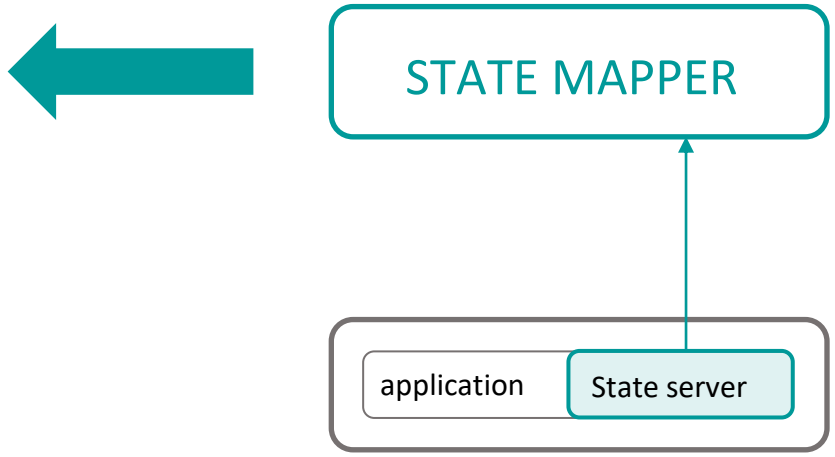
K8s state ← **Application state**

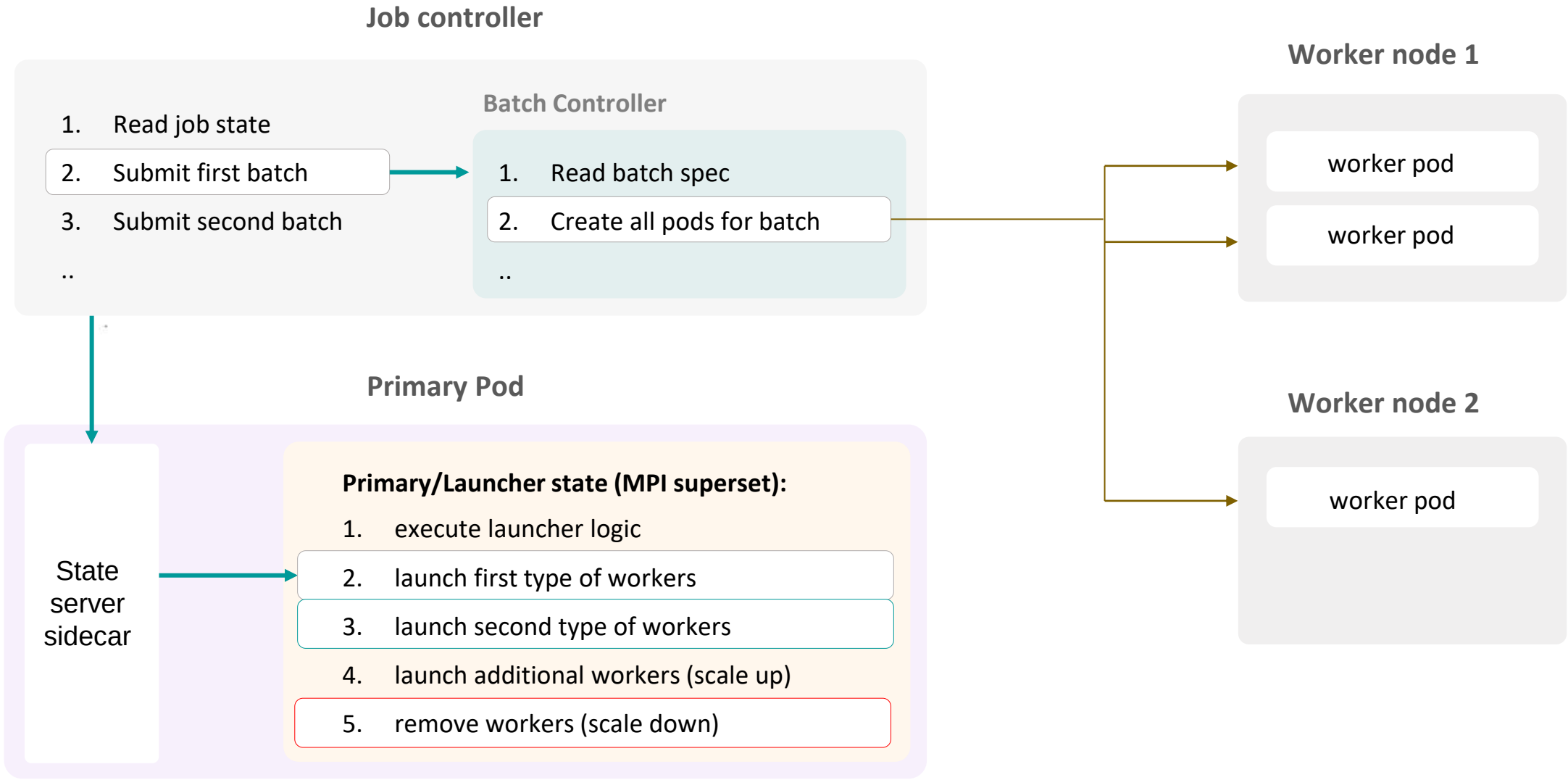
```
kind: Pod
...
spec:
  readinessGates:
    - conditionType: "www.example.com/feature-1"
status:
  conditions:
    - type: Ready
      status: "False"
      lastProbeTime: null
      lastTransitionTime: 2018-01-01T00:00:00Z
      …
containerStatuses:
    - containerID: docker://abcd...
      ready: true
...
```



K8s state ← **Application state**

```
apiVersion: kube-calibre.eda.siemens.com/v1
kind: CalibreJob
...
spec:
  calibre:
  ...
status:
  serverStatus: Connected
  state:
    hdb0Connection: 0.0.0.0:0
    phdbAccepted: 0
    phdbNeeded: 0
    rcsAccepted: 6
    rcsNeeded: 6
    rdsAccepted: 0
    rdsNeeded: 0
    stage: phdbPrepare
  status: Running
```




Job controller
1. Read job state
2. Submit first batch
3. Submit second batch
..
Batch Controller
1. Read batch spec
2. Create all pods for batch
..
Worker node 1
worker pod
worker pod
Worker node 2
worker pod
Primary Pod
State server sidecar
Primary/Launcher state (MPI superset):
1. execute launcher logic
2. launch first type of workers
3. launch second type of workers
4. launch additional workers (scale up)
5. remove workers (scale down)

Native batch job controller: batch/v1 as workqueue

```
apiVersion: batch/v1
kind: Job
metadata:
  name: pi
spec:
  template:
    metadata:
    …
    spec:
      containers:
      …
  restartPolicy: Never
  backoffLimit: 4
  parallelism: 4
  podFailurePolicy:
    rules:
    - action: FailJob
    onExitCodes:
      containerName: main
      operator: In
      values: [42]
    - action: Ignore
    onPodConditions:
    - type: DisruptionTarget
```

Kubernetes 1.26: Job Tracking, to Support Massively Parallel Batch Workloads, Is Generally Available

https://kubernetes.io/blog/2022/12/29/scalable-job-tracking-ga/

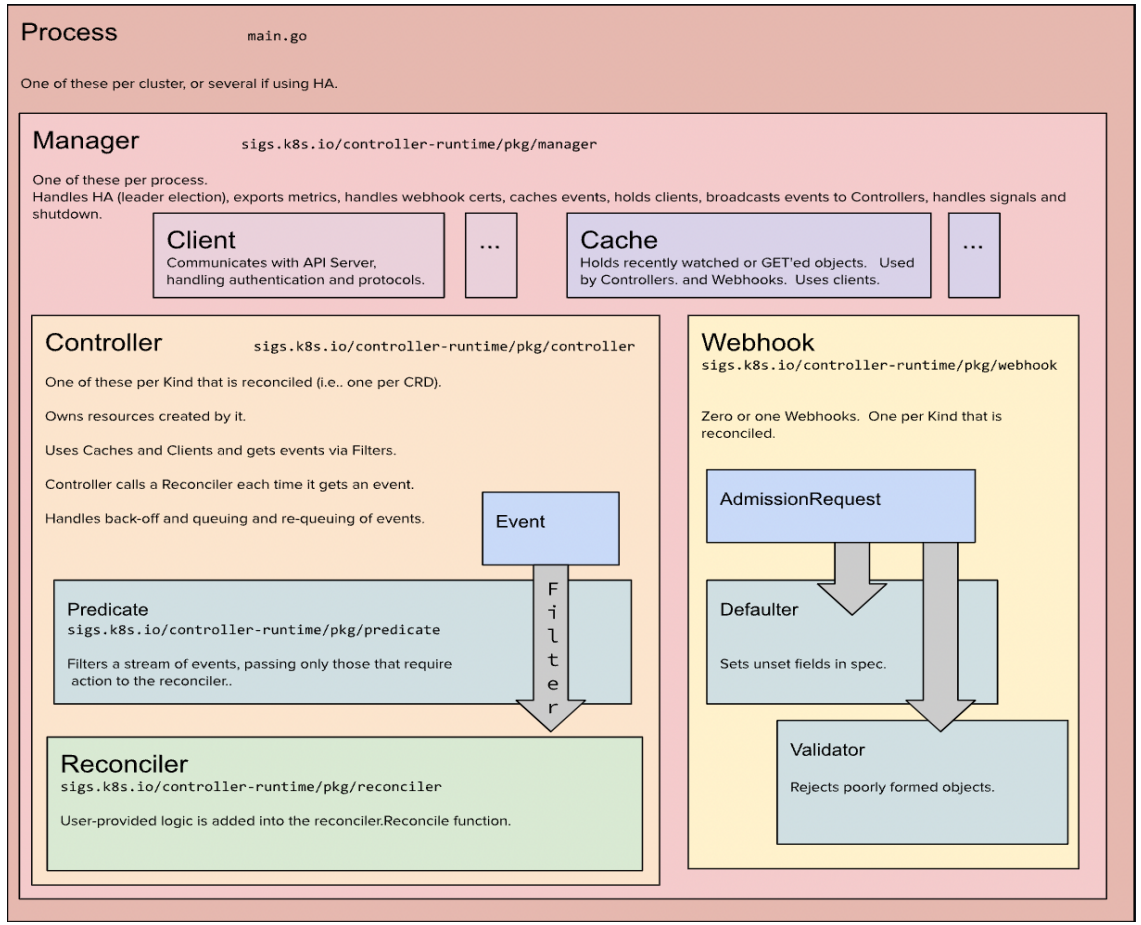
Kubernetes 1.28: Improved failure handling for Jobs

https://kubernetes.io/blog/2023/08/21/kubernetes-1-28-jobapi-update/

Custom batch controller

- implement custom job scale down behavior (chose which remotes to scale down first)
- offload batch controller from api-server (managed environment)
- implement custom status information (e.g. characterize ramp up transitory)
- older environments work with earlier k8s versions (managed environment)

sigs.k8s.io/controller-runtime



K8s api-server

wss:// or HTTP/2

client.go/tools/cache.**Reflector**

client.go/tools/cache.**Store** (Informer)

controller-runtime/pkg/**Predicate** (optional)

controller-runtime/pkg/**Reconciler**

Observe api-server

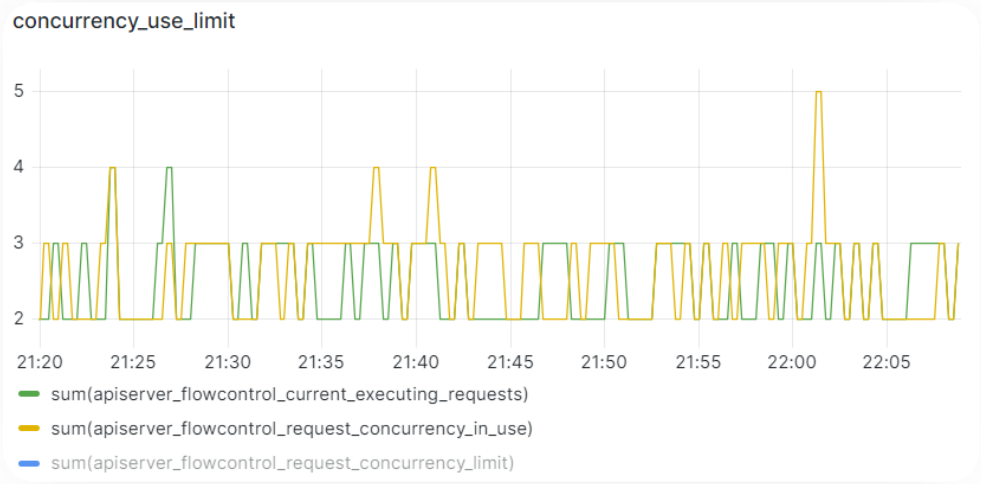
Auditing

```
apiVersion: audit.k8s.io/v1
kind: Policy
rules:
  - level: RequestResponse
    verbs: ["get", "list", "watch"]
    resources:
      - group: "" # core
        resources: ["pods"]
```

..
{"kind":"Event","apiVersion":"audit.k8s.io/v1","level":"RequestResponse","auditID":"c3284a70-8286-4c0c-a233-8a4f7b093e80","stage":"RequestReceived","requestURI":"/api/v1/pods?labelSelector=calibre-owner%3Dcalibre\u0026limit=500\u0026resourceVersion=0","verb":"list“ ..”}

Server metrics

- apiserver_request_duration_seconds_bucket
- apiserver_flowcontrol_request_concurrency_in_use
- apiserver_flowcontrol_request_concurrency_limit
  ..



Observe client and cache

Verbose client logging

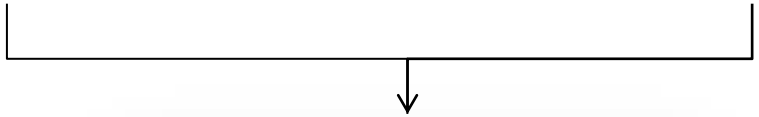
```
klogv2 "k8s.io/klog/v2"
..
klogv2.InitFlags(nil)
flag.Set("v", "10")
```

```
I0808 13:43:44.213465 2122057
round_trippers.go:466] curl -v -
XPOST -H "Accept:
application/json" -H "Content-
Type: application/json"
'https://127.0.0.1:43889/api/v1/
namespaces/default/pods'
```

Dump watch events through predicates

```
type OnlyObserve struct {
 predicate.Funcs
}

func (OnlyObserve) Delete(e
event.DeleteEvent) bool {
...
}
```



```
{"kind":"PodList","apiVersion“..
{"type":"ADDED","object":{"apiVersion“..
{"type":"ADDED","object":{"kind":"Pod"
```

Tap client RoundTripper interface

```
"k8s.io/client-go/rest"
..
cfg.WrapTransport = func(rt
http.RoundTripper) http.RoundTripper {
}
```

And client side metrics

- **workqueue**_depth
- workqueue_queue_duration_seconds
- ..
- **rest_client**_request_latency_seconds
- ..
- **reflector**_list_duration_seconds
- reflector_watches_total
- ..

https://book.kubebuilder.io/reference/metrics

Watch the right way !

```
apiVersion: v1
kind: List
items:
- apiVersion: v1
  kind: Pod
  name: a-drc-0-example8c9dd950-48c8p
  metadata:
    labels:
      calibre-owner: calibre
  spec:
    containers:
    - name: calibre
      command:
      - /bin/bash ..
- apiVersion: v1
  kind: Pod
  name:  a-drc-0-example8c9dd950-hq944
  metadata:
    labels:
      calibre-owner: calibre
  spec:
    containers:
    - name: calibre
      command:
      - /bin/bash ..
- apiVersion: v1
  kind: Pod
  name: test-pod
  metadata:
    labels:
      some-key: value
  spec:
    containers:
    - name: some
      command:
      - /bin/bash ..
```

Client side event filtering


```
ctrl.NewControllerManagedBy(mgr).
   For(&kcalv1.Batch{}).
   Watches(
      &corev1.Pod{},
      handler.EnqueueRequestForOwner(
         mgr.GetScheme(),
         mgr.GetRESTMapper(),
         &kcalv1.Batch{},
         handler.OnlyControllerOwner(),
      ),
      builder.WithPredicates(predDelete),
   ).
```

Reconcile on kcalv1.Batch events

And reconcile on corev1.Pod events, which are found to be child resources of Batch and owned by its controller (*ownerReference*)

And reconcile only on events that pass this custom predicate filtering logic

suite_test.go:117] REQ:&{GET https://127.0.0.1:39205**/api/v1/pods?limit=500**&resourceVersion=0 HTTP/1.1

```
apiVersion: v1
kind: List
items:
- apiVersion: v1
  kind: Pod
  name: a-drc-0-example8c9dd950-48c8p
  metadata:
    labels:
      calibre-owner: calibre
  spec:
    containers:
    - name: calibre
      command:
      - /bin/bash ..
- apiVersion: v1
  kind: Pod
  name:  a-drc-0-example8c9dd950-hq944
  metadata:
    labels:
      calibre-owner: calibre
  spec:
    containers:
    - name: calibre
      command:
      - /bin/bash ..
- apiVersion: v1
  kind: Pod
  name: test-pod
  metadata:
    labels:
      some-key: value
  spec:
    containers:
    - name: some
      command:
      - /bin/bash ..
```

Server side event filtering: labelSelector


```
cache.Options{
   ByObject: map[client.Object]cache.ByObject{
      &kcalv1.Batch{}: {..},
      &corev1.Pod{}: {
         Label: labels.SelectorFromSet(
            labels.Set{"calibre-owner": "calibre"}
         ),
      },
   },
}
```

Receive events for kcalv1.Batch objects with any label

Receive events for kcalv1.Pods objects with only with specific labels

..
suite_test.go:117] REQ:&{GET https://127.0.0.1:38103**/api/v1/pods?labelSelector=calibre-owner**%3Dcalibre&limit=500&resourceVersion=0 HTTP/1.1
..

```
apiVersion: v1
kind: List
items:

- apiVersion: v1
  kind: Pod
  name: a-drc-0-example8c9dd950-48c8p
  metadata:
    labels:
      calibre-owner: calibre
  spec:
    containers:
    - name: calibre
      command:
      - /bin/bash .
.
- apiVersion: v1
  kind: Pod
  name:  a-drc-0-example8c9dd950-hq944
  metadata:
    labels:
      calibre-owner: calibre
  spec:
    containers:
    - name: calibre
      command:
      - /bin/bash ..

- apiVersion: v1
  kind: Pod
  name: test-pod
  metadata:
    labels:
      some-key: value
  spec:
    containers:
    - name: some
      command:
      - /bin/bash ..
```

Server side event filtering: PartialObjectMetadata

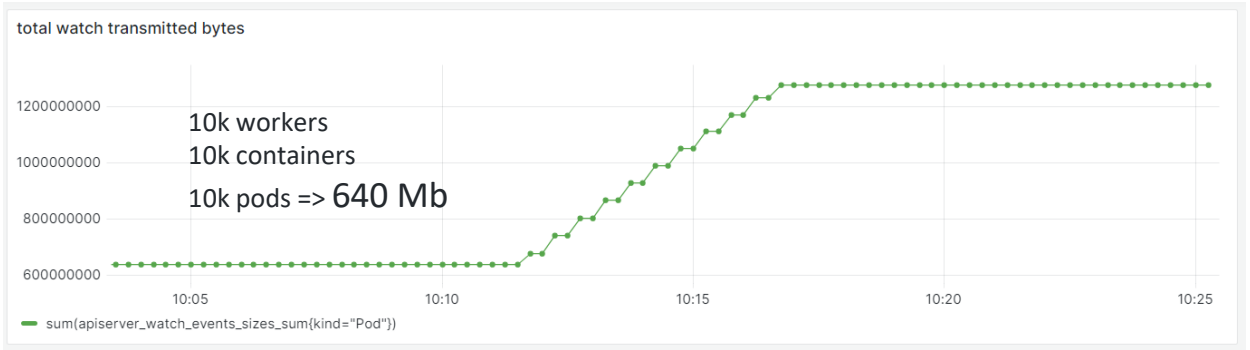

```
ctrl.NewControllerManagedBy(mgr).
   WatchesMetadata (
      &corev1.Pod{},
   ).
```

→ Only watch metadata subresource for Pods

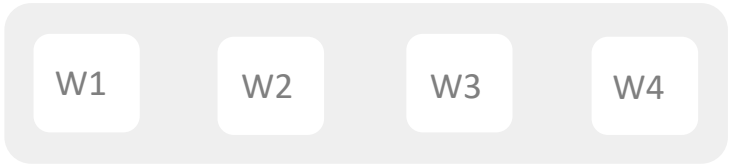
..
suite_test.go:117] REQ:&{GET https://127.0.0.1:38103/api/v1/pods..

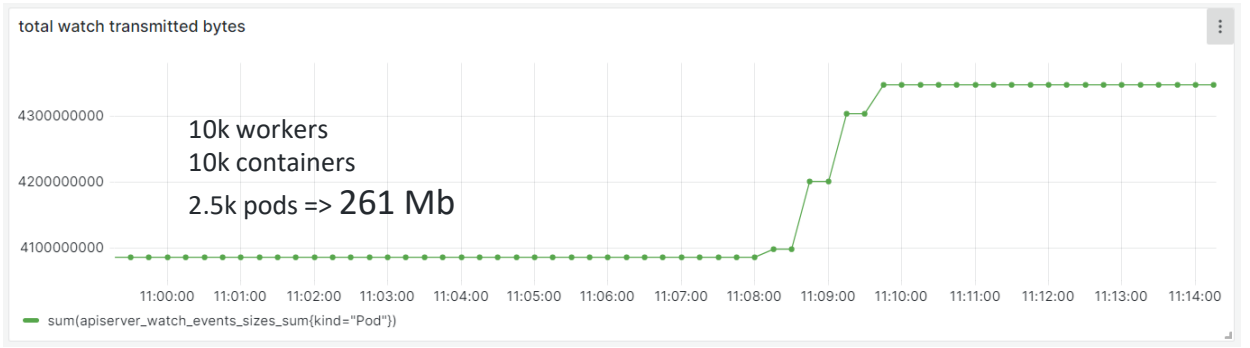
Accept: application/vnd.kubernetes.protobuf;as=PartialObjectMetadata

n workers / n containers / n pods

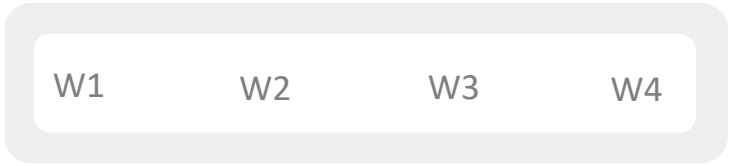


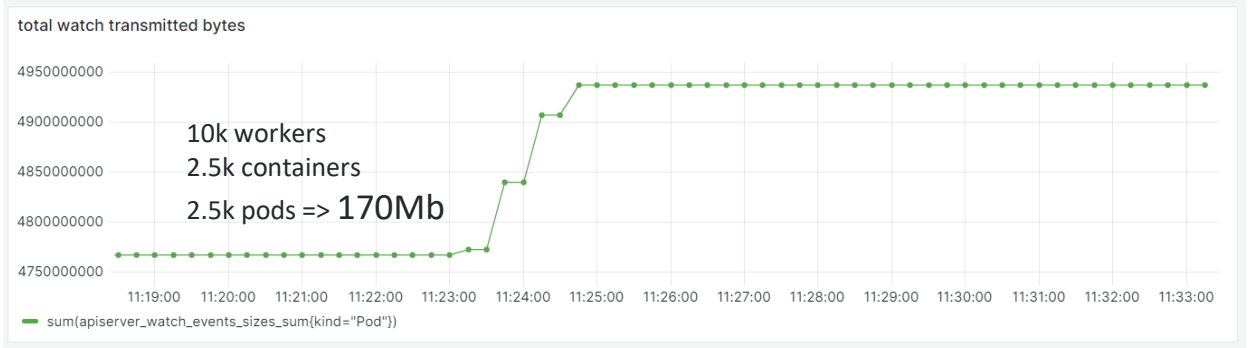
n workers / n containers / 1 pod

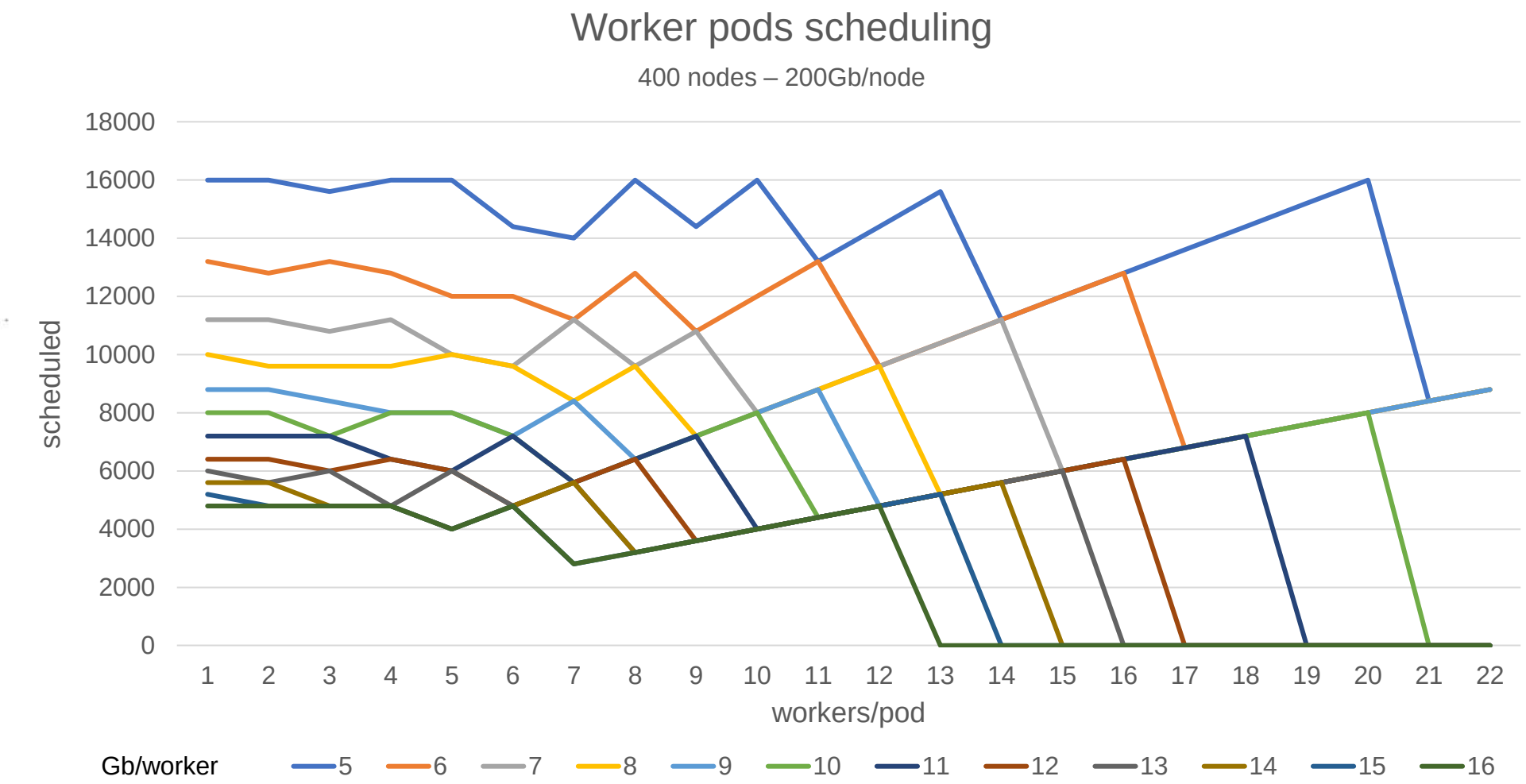




n workers / 1 container / 1 pod






Worker pods scheduling
400 nodes – 200Gb/node
scheduled
18000
16000
14000
12000
10000
8000
6000
4000
2000
0
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22
workers/pod
Gb/worker
5 6 7 8 9 10 11 12 13 14 15 16

Job Spec design

```
apiVersion: kube-calibre.eda.siemens.com/v1
kind: CalibreJob
metadata:
 name: drc-0-example
spec:
 calibre:
  logFile: "transcript"
  ruleFile: "rules"
  runMode:
  ..
 primary:
  nodeSelector:
   calibre-static-role: "primary"
  container:
   resources:
    limits:
     cpu: "1"
    requests:
     cpu: “1“
     ..
```

Apply to primary pod

```
 common:
  volumes:
  ..
  securityContext:
  ..
  container:
   image: aoj_cal_2021.3_35.19:latest
   env:
    - name: LM_LICENSE_FILE
      value: “.."
  tolerations:
  ..
```

Apply to all pods

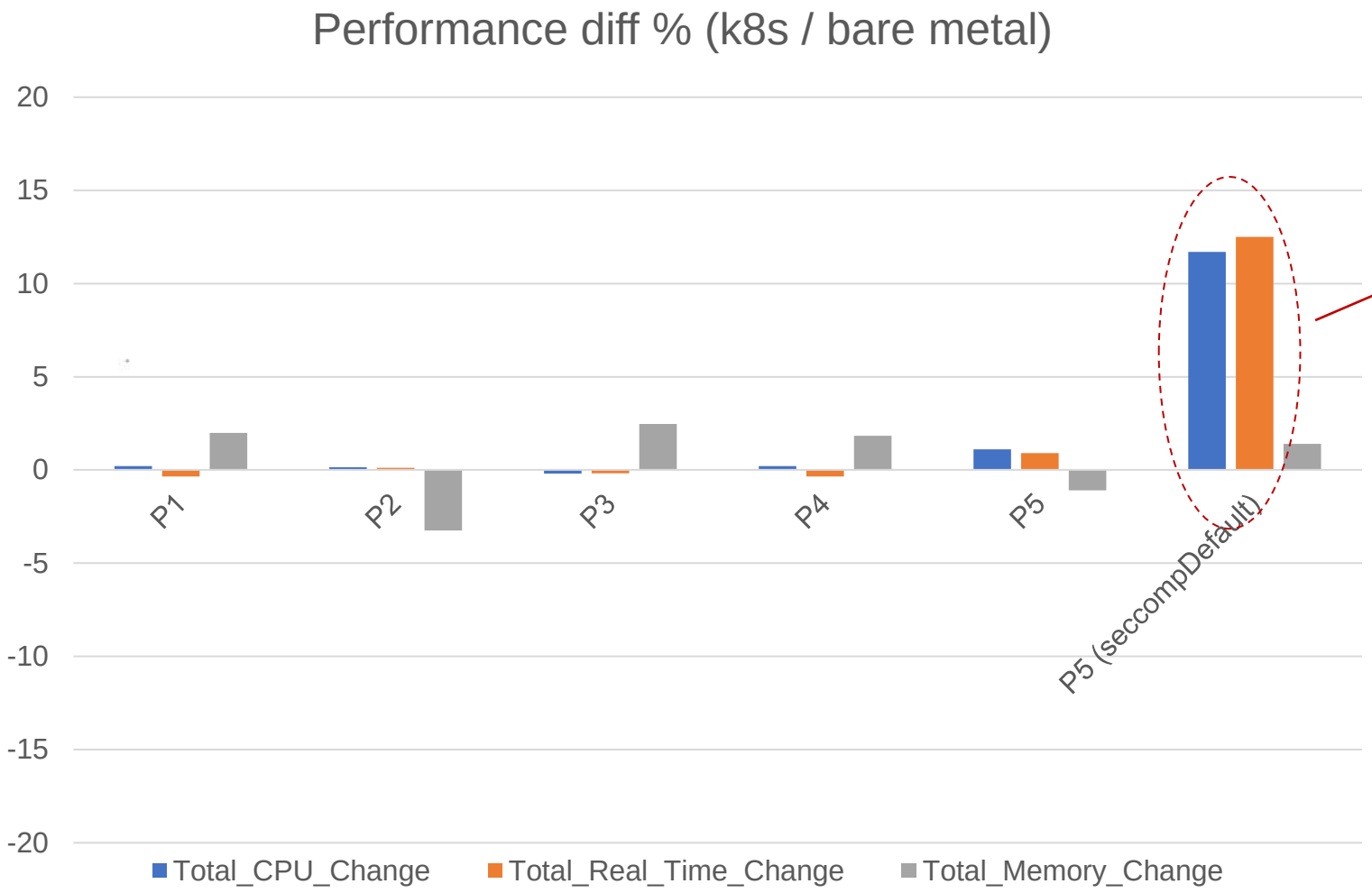
```
 compute:
  nodeSelector:
   calibre-static-role: "remote"
  container:
   resources:
   ..
```

Apply to compute pods

```
apiVersion: v1
kind: Pod
spec:
 ..
 nodeSelector:
    calibre-static-role: “primary”
 ..
```

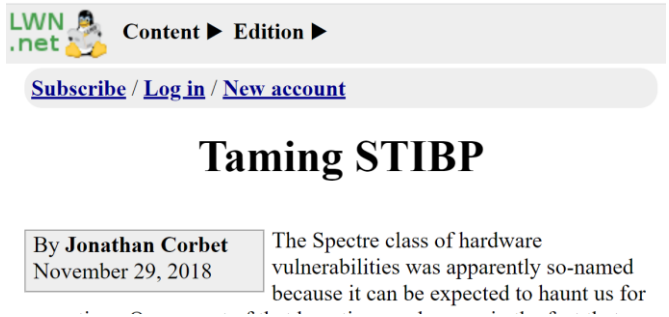
```
apiVersion: v1
kind: Pod
spec:
 ..
 nodeSelector:
    calibre-static-role: “remote”
 ..
```

- Expose k8s scheduling primitives (podSpec)
- Expose generic sidecar injection
- Concise (D.R.Y.) -> inheritance
- Fine grained remote type configuration



seccomp

$ cat /sys/devices/system/cpu/vulnerabilities/spec_store_bypass
Mitigation: **Speculative Store Bypass** disabled via prctl and **seccomp**


LWN.net Content ▶ Edition ▶

Subscribe / Log in / New account

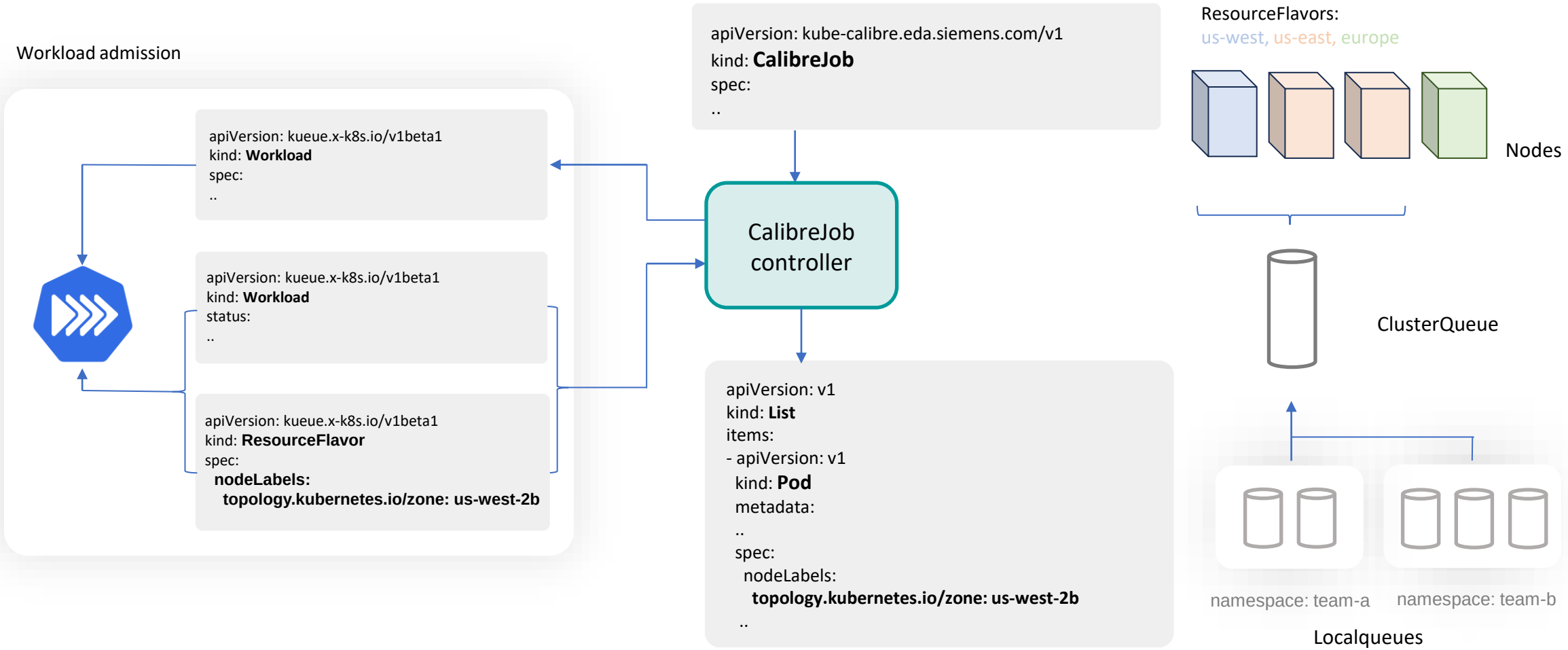
Taming STIBP

By **Jonathan Corbet**
November 29, 2018

The Spectre class of hardware vulnerabilities was apparently so-named because it can be expected to haunt us for

https://lwn.net/Articles/773118/

Multi tentancy and queueing: kubernetes-sigs/kueue



Key takeways

HPC and EDA
- definition of workloads

Mapping application state to k8s
- observable and predictable orchestration

Observe controllers/api-server interaction
- server side
- client side

Optimize controller runtime watches
- *https://github.com/kubernetes-sigs/controller-runtime*
- client side: predicates
- server side: labelSelectors and PartialObjectMetadata

Performance parity and pitfalls
- same performance
- seccomp pitfalls

Expose multi tenancy through job queues
- custom workloads with
https://github.com/kubernetes-sigs/kueue

Thank you !

Luca Montechiesi
luca.montechiesi@siemens.com

Min Tsao
min.tsao@siemens.com