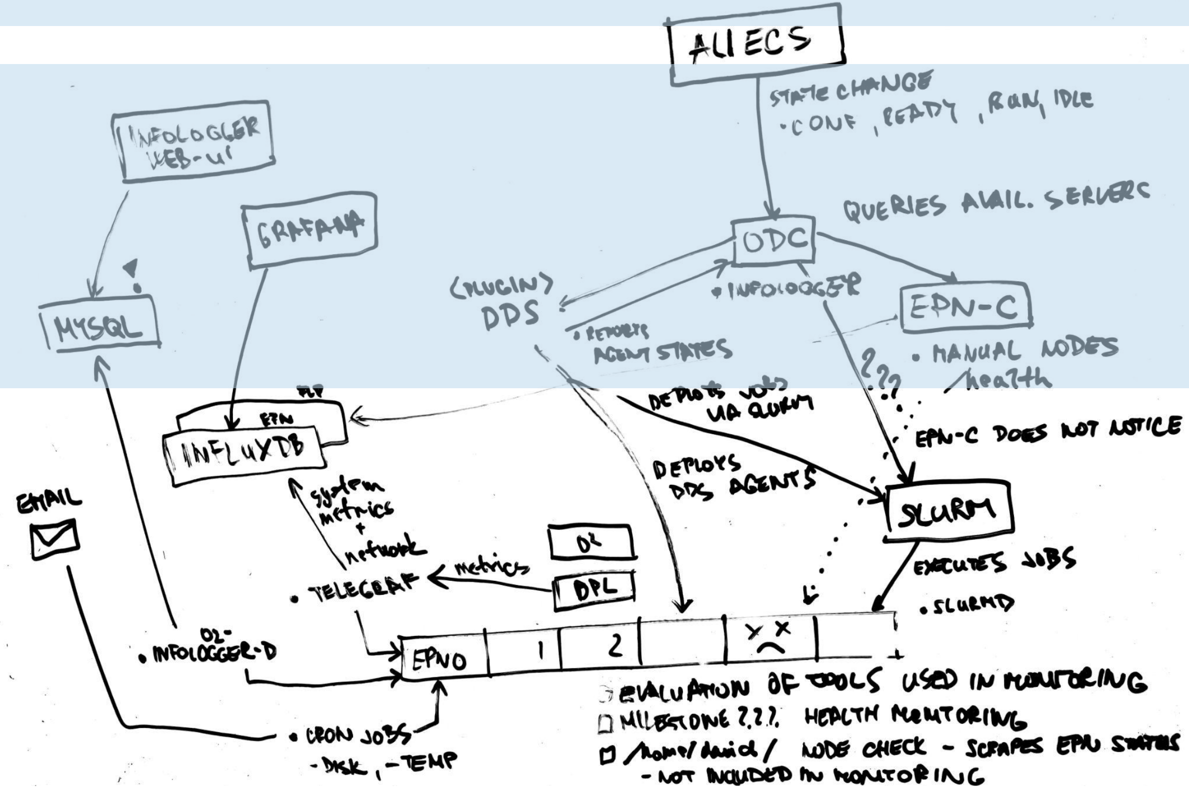
ALICE Orchestration - Lessons Learned

Luboš Krčál
lubos.krcal@cern.ch

Piotr Konopka
piotr.jan.konopka@cern.ch

ALICE / O2 | 8th April 2025



ALICE DAQ Overview

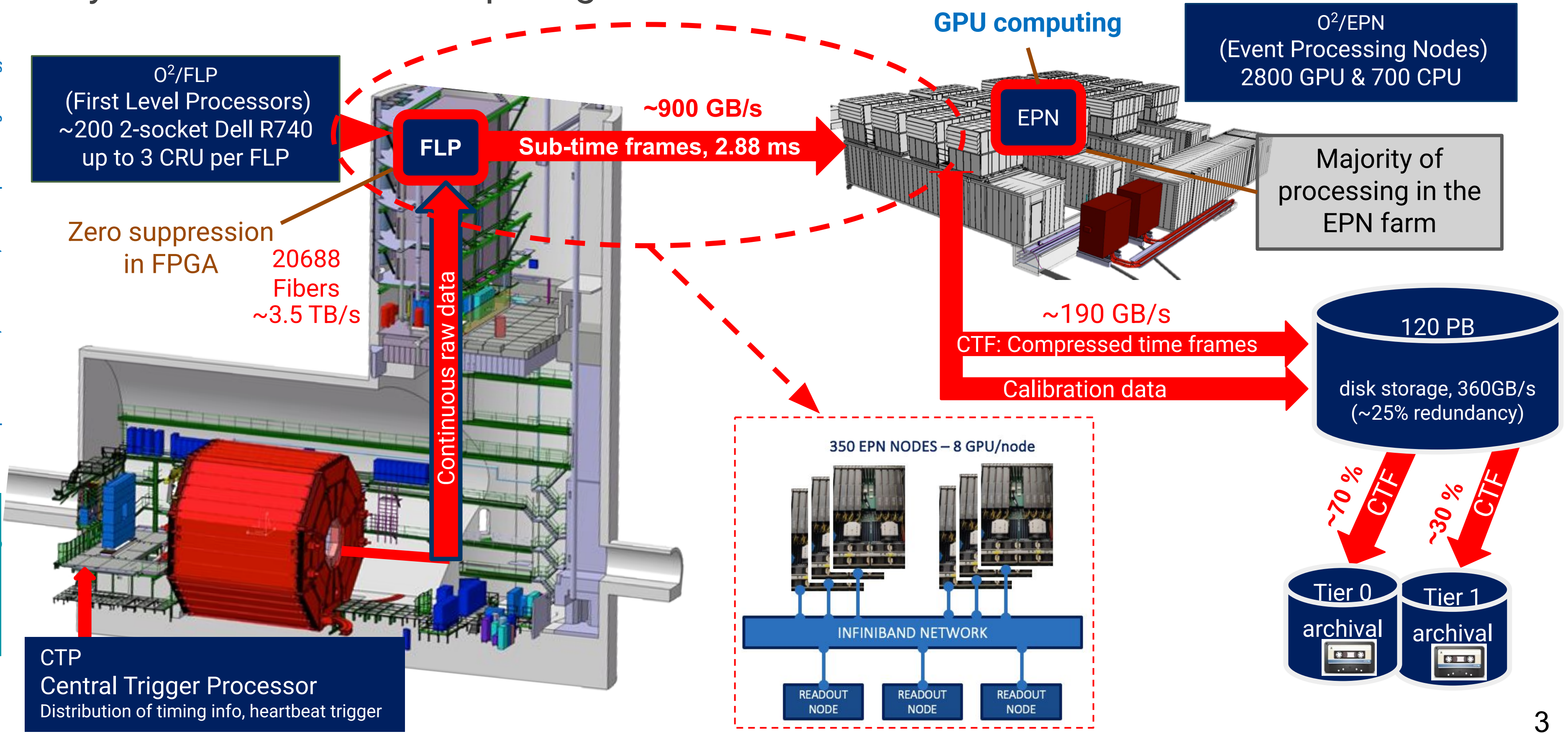
Readout farm

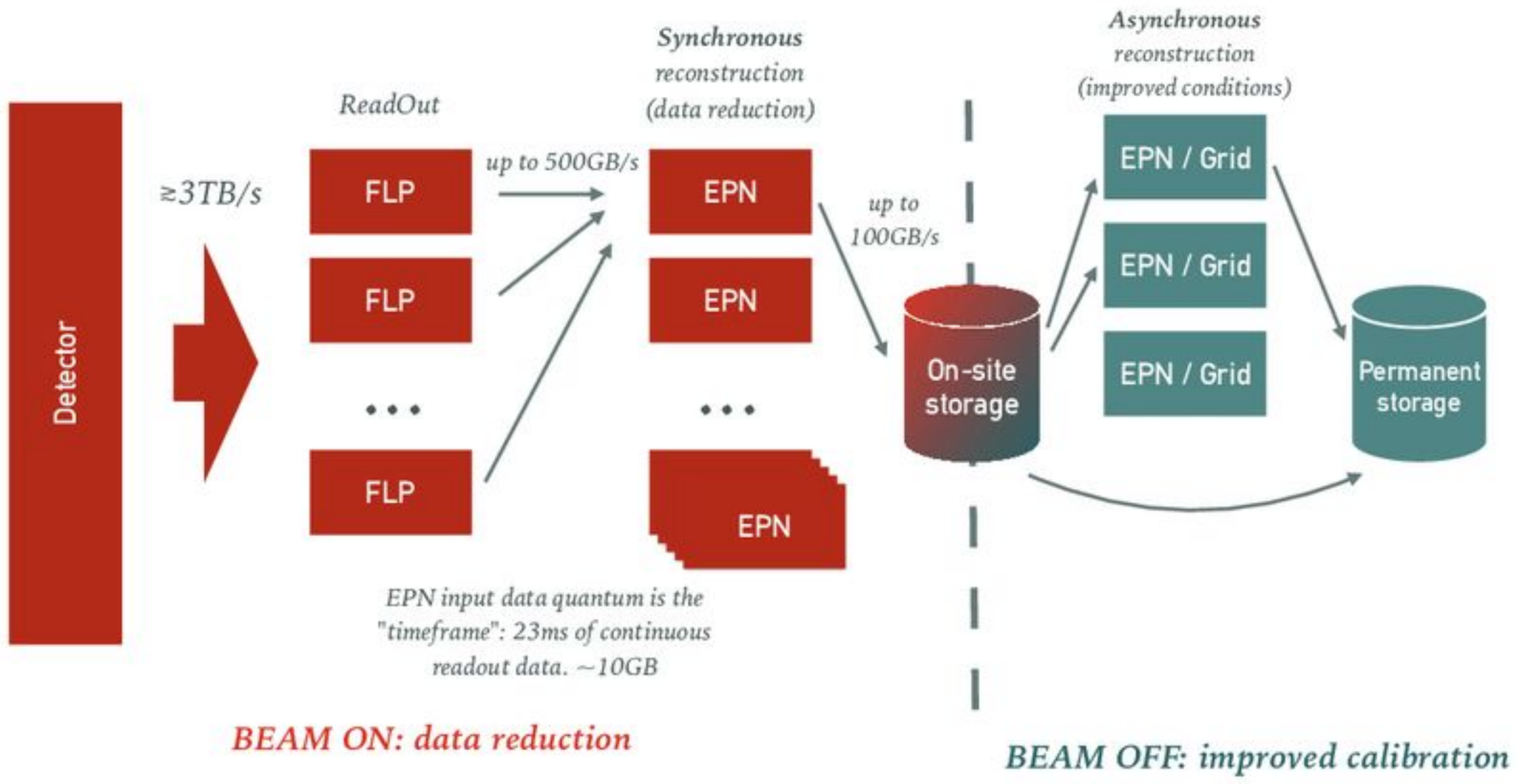
Network / data distribution

HPC farm

THE ALICE RUN 3 DATA FLOW

Layout of the ALICE computing at the LHC P2




Detector
≳3TB/s
ReadOut
FLP
FLP
FLP
up to 500GB/s
Synchronous
reconstruction
(data reduction)
EPN
EPN
EPN
up to
100GB/s
On-site
storage
Asynchronous
reconstruction
(improved conditions)
EPN / Grid
EPN / Grid
EPN / Grid
Permanent
storage
EPN input data quantum is the
"timeframe": 23ms of continuous
readout data. ~10GB
BEAM ON: data reduction
BEAM OFF: improved calibration

Orchestration & Data processing in ALICE

EPN (HPC farm)

- **Global processing**
- Up to 150k tasks / processes
- Across 350 GPU nodes
  - And ~15 service nodes
- Buffering of processed data before semi-permanent storage

Data Distribution

- Millisecond scheduling across 200+350 nodes
- Buffer management on source (<= 1 sec) and destination nodes (<= 1 min)

FLP & Quality Control nodes:

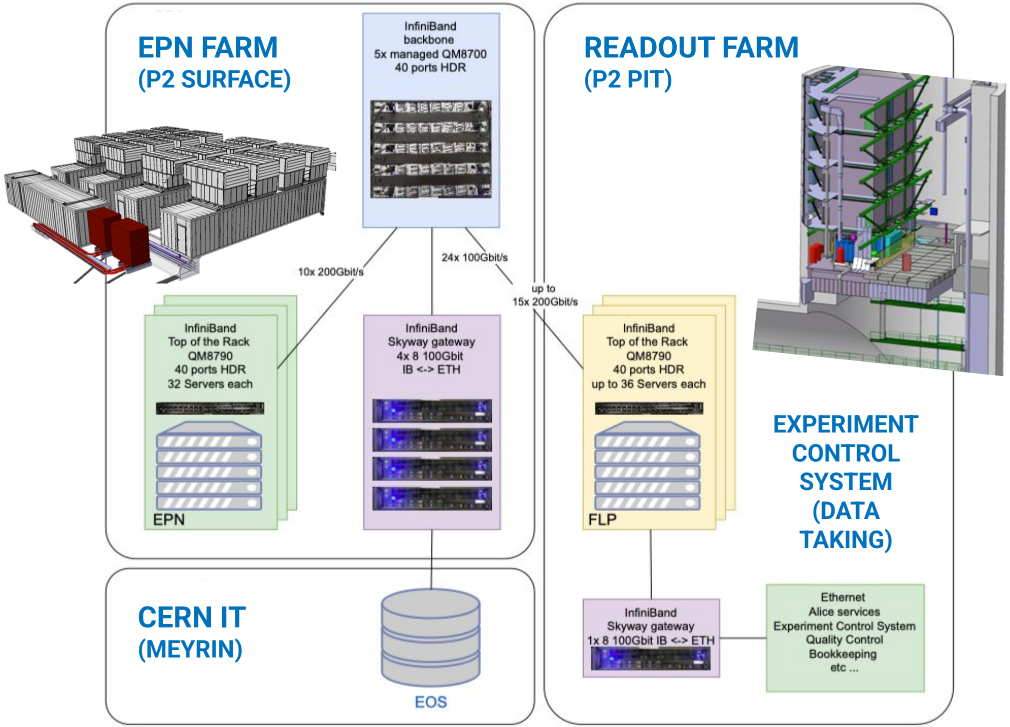
- **Local processing**
- Tasks statically assigned to nodes due to detector readout links distribution
- ~200 FLPs, 5-20 tasks each
- ~15 QC nodes, 5-50 tasks each

Networking:

- FLP -> EPN
- FLP -> QC nodes
- EPN -> QC nodes

THE EPN NETWORK TOPOLOGY

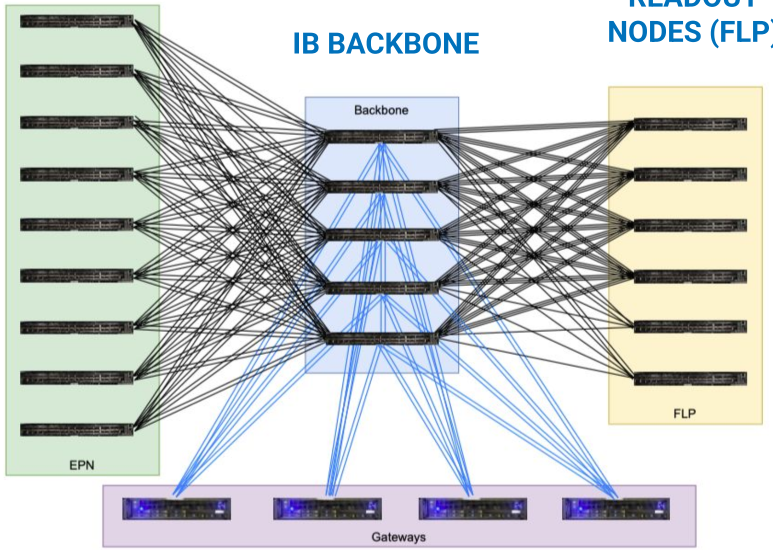
The backbone of the EPN farm is based on HDR Infiniband



EPN BUILDING BLOCKS (IB TOR)

IB BACKBONE

DETECTOR READOUT NODES (FLP)

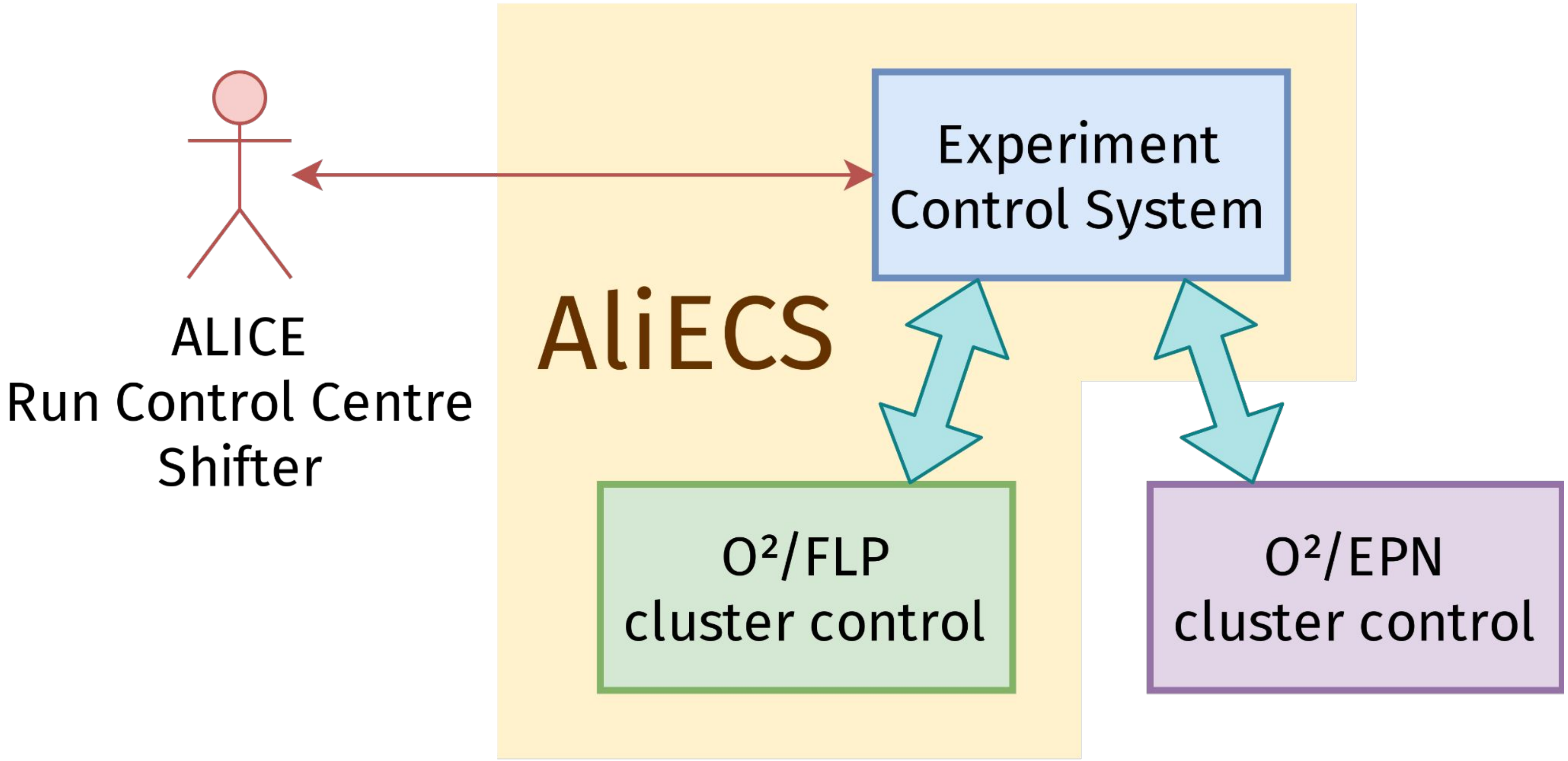


IB TO ETHERNET GATEWAYS (to CERN IT mass storage)

ALICE Experiment Control System

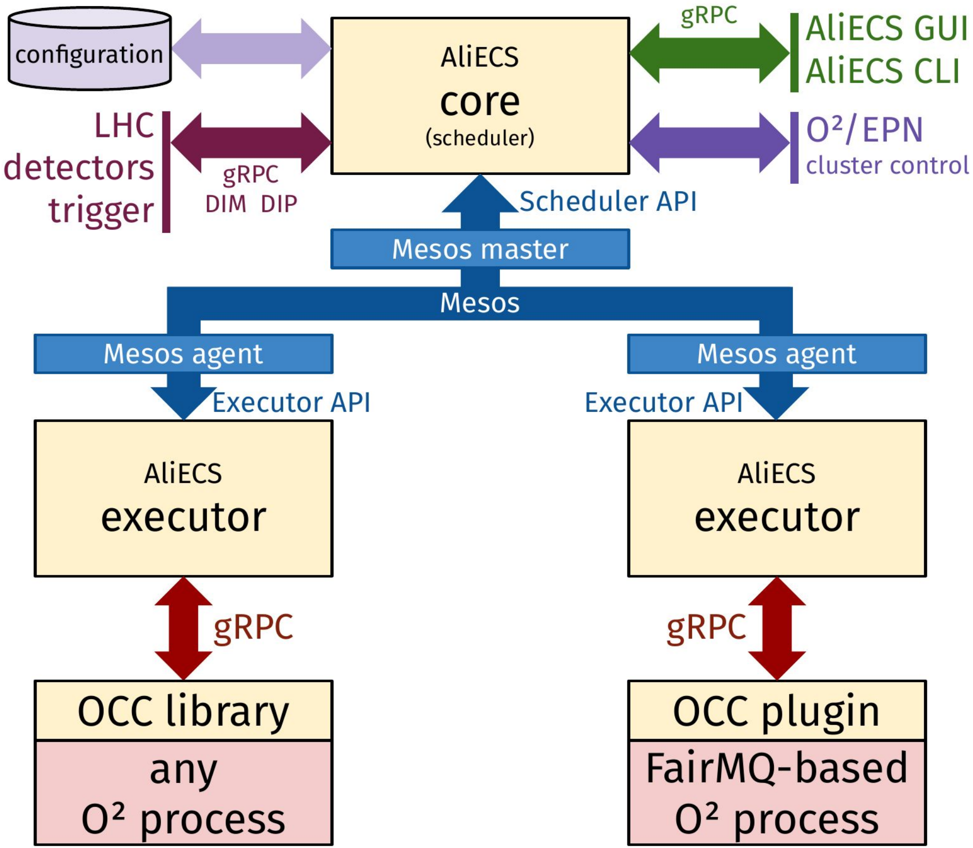
ALICE Experiment Control System

Overview



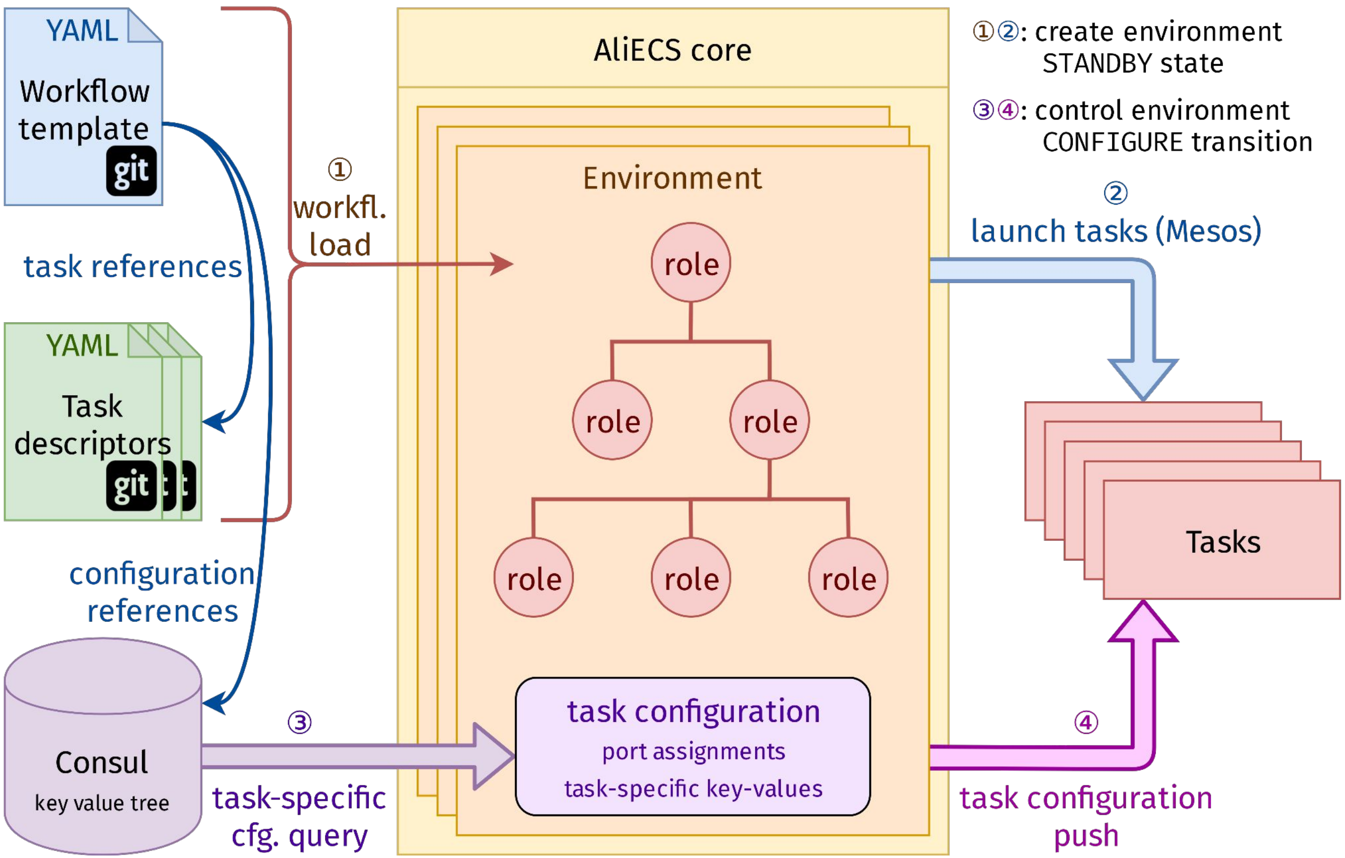
ALICE Experiment Control System

Architecture



ALICE Experiment Control System

Configuration



ALICE Experiment Control System

“Manifest” (workflow template) example

```
...
- name: "data-distribution-dpl"                                              # role name
  enabled: "{{!strings.IsFalsy(dpl_workflow) && dd_enabled == 'true'}}"  # enables the role if conditions apply
  defaults:
    fmq_rate_logging: "10"                                                   # defining a default value for a key
  roles:
    - name: "stfb"
      enabled: "{{stfb_standalone == 'false'}}"
      vars:
        dd_discovery_stfb_id: stfb-{{ flp_host }}-{{ uid.New() }}            # a var which will overwrite a default
      connect:                                                               # connection parameters
      - name: readout
        type: pull
        target: "{{ Up(2).Path }}.readout:readout"
        rateLogging: "{{ fmq_rate_logging }}"
      bind:
        - name: dpl-chan
          type: push
          rateLogging: "{{ fmq_rate_logging }}"
          transport: shmem
          addressing: ipc
          sndBufSize: "4"
          global: "readout-proxy-{{ flp_host }}”
      task:                                                                 # loads a task template
        load: stfbuilder
...
```

ALICE Experiment Control System

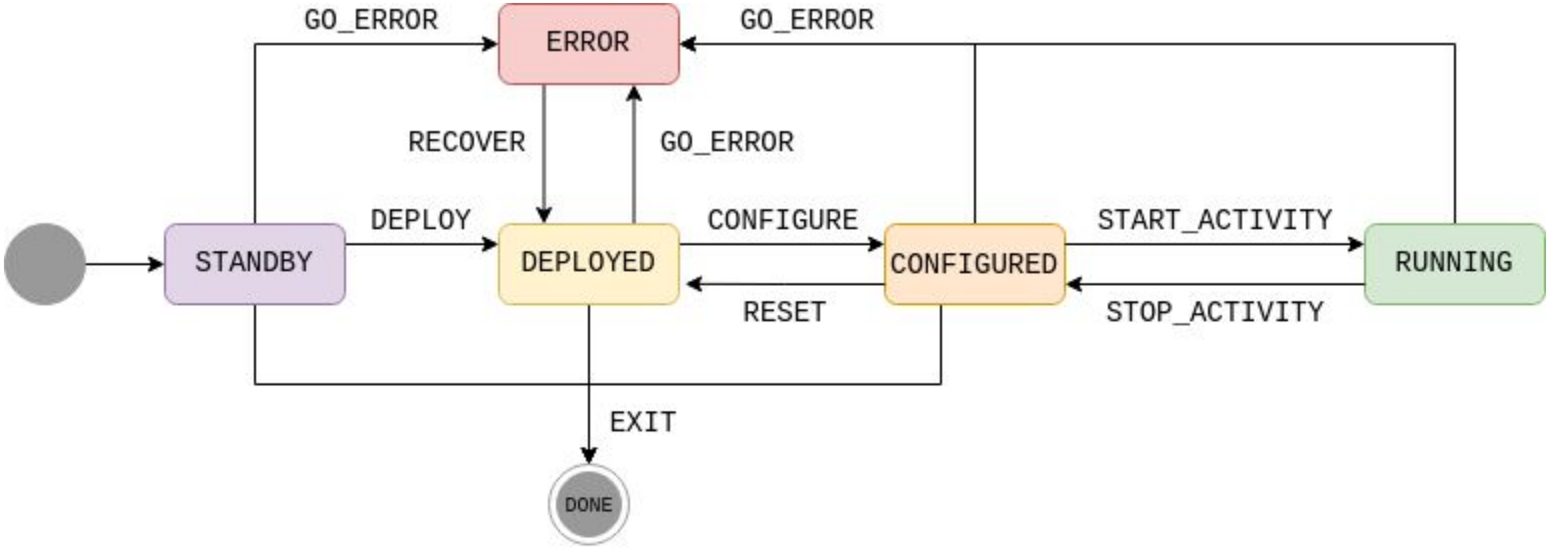
Task-specific configuration example

```
{
  "qc": {
      "config": {
      {% include "ZDC/configuration/config-production" %}
      },
      "tasks": {
      {% include "ZDC/tasks/QcZDCTask" %},
      {% include "ZDC/tasks/QcZDCTask-Trending" %},
      {% include "ZDC/tasks/QcZDCRecTask" %}
      },
      "checks": {
      {% include "ZDC/checks/QcZDCRawCheck" %},
      {% include "ZDC/checks/QcZDCRecCheck" %},
      {% include "ZDC/checks/QcZDCRefCheck" %}
      },
      "aggregators": {
      {% include "ZDC/aggregators/ZDCQuality" %}
      }
  }
}
```

```
{
  "qc": {
    "config": {
      "database": {
        "implementation": "CCDB",
        "host": "alice-qcdb:12345"
      },
      "monitoring": {
        "url": "alice-monitoring:12345"
      },
      "consul": {
        "url": "alice-consul.cern.ch:12345"
      },
      "conditionDB": {
        "url": "alice-ccdb.cern.ch:12345"
      },
      "bookkeeping": {
        "url": "alice-bookkeeping.cern.ch:12345"
      }
    },
    "tasks": {
      "QcZDCTask": {
        "active": "true",
        "critical": "false",
        "className": "o2::qc::zdc::ZDCRawDataTask",
        "moduleName": "QcZDC",
        "detectorName": "ZDC",
...
```

ALICE Experiment Control System

State Machine



ALICE Experiment Control System

Integration with other services

- During state transitions, multiple operations in the experiment's subsystem should be performed in a specific order
- Order of operations is configurable
  - before/after specified state transition
  - can be ordered with a weight
  - can start and finish at different points
  - can be critical or non-critical for a successful transition
- gRPC/protobuf widely used to communicate between ALICE services

```
- name: dcs
  enabled: "{{dcs_enabled == 'true'}}"
  roles:
  - name: pfr
    call:
      func: dcs.PrepareForRun()
      trigger: before_CONFIGURE
      await: after_CONFIGURE-1
      timeout: "{{ dcs_pfr_timeout }}"
      critical: false
  - name: sor
    call:
      func: dcs.StartOfRun()
      trigger: before_START_ACTIVITY+100
      timeout: "{{ dcs_sor_timeout }}"
      critical: true
...
- name: odc
  enabled: "{{odc_enabled == 'true'}}"
  roles:
    - name: part-init
      call:
        func: odc.PartitionInitialize()
        trigger: before_DEPLOY
        await: after_DEPLOY-1
        timeout: "{{ odc_partitioninitialize_timeout }}"
        critical: true
```

ALICE Experiment Control System

Event streaming service

- GUIs and other services benefit from “real-time” updates of the knowledge available to ECS e.g. task state, progress of transitions, integrated services state
- We use Kafka to distribute events, the ECS takes the responsibility to do it for integrated services as well
- Events are encoded with protobuf

```
message Ev_TaskEvent {
  string name = 1;
  string taskid = 2;
  string state = 3;
  string status = 4;
  string hostname = 5;
  string className = 6;
  Traits traits = 7;
  string environmentId = 8;
  string path = 9;
}
```

Job Orchestration @ HPC farm

Job Orchestration @ HPC farm

Overview

- Separate control system from the ALICE Experiment Control System
- Built “in-house” at another institute which is part of the ALICE collaboration

Main components:

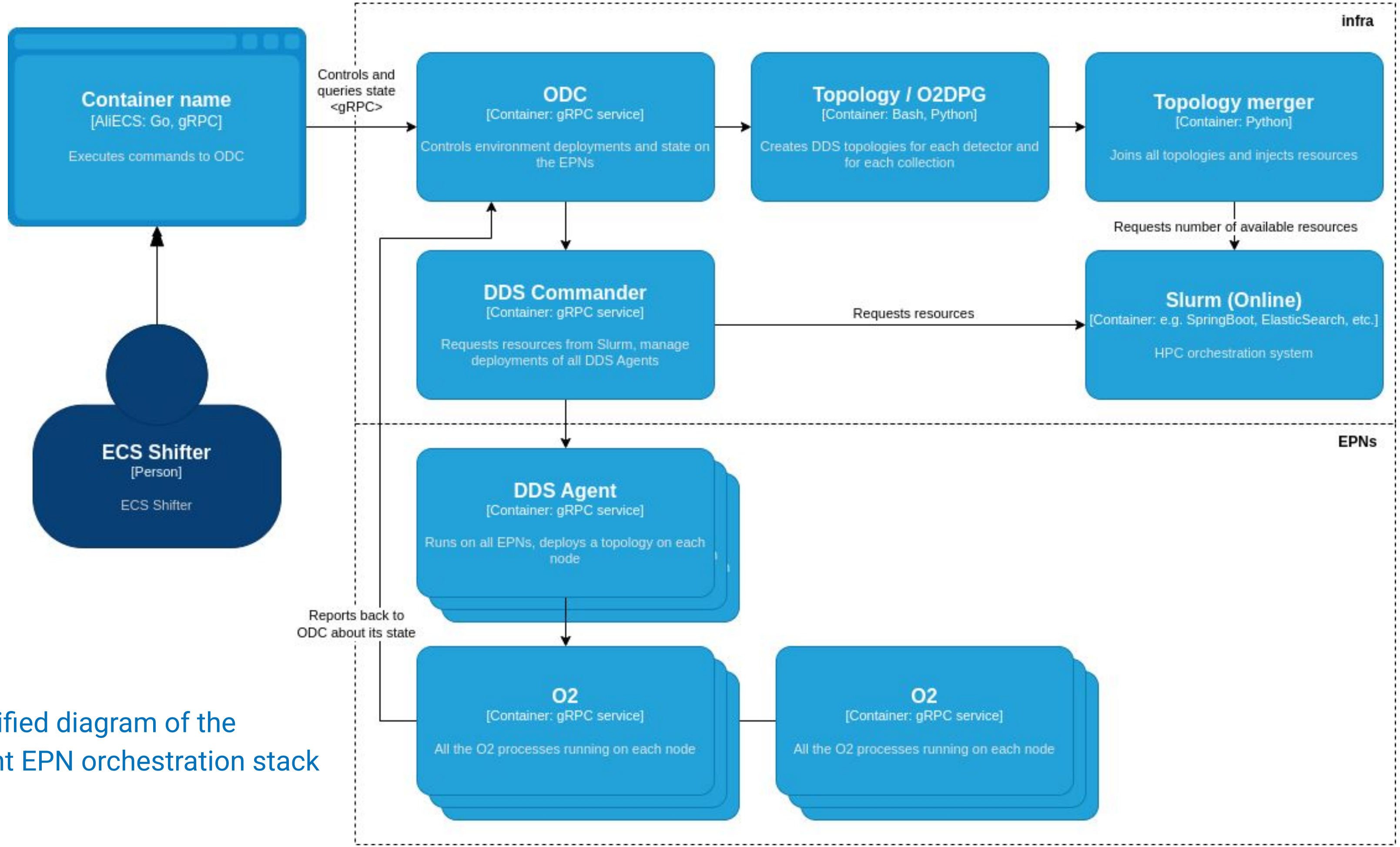
- Online Device Control (ODC)
- Dynamic Deployment System (DDS)
- Resource management
  - Slurm
- Topology tools
  - O2DPG
  - Merger
    - Data distribution
    - Physics processing software
    - System configuration
- Data Processing Layer (DPL)
- FairMQ

HPC farm shared across:

- online processing (real time data taking)
- async processing

Constraints

- Two Slurms - one for each use-case
- Static node allocation
  - Requires expert intervention
  - Requires planning based on LHC operational schedule
- Bad resource utilization as a result



Simplified diagram of the current EPN orchestration stack

Orchestration @ HPC farm - ODC

Online Device Control

- Controls a graph (topology) of processes = FairMQ devices using DDS
  - Deployment
  - State management
- Components
  - The core library odc-core-lib.
  - The gRPC server odc-grpc-server is a sample implementation of the server based on the odc-core-lib.
  - The gRPC client odc-grpc-client is a sample implementation of client.
- Plugin required in all processes

ODC Challenges

- Difficult operations and debugging
  - Thanks to excellent maintenance all issues were resolved quickly
  - Relying on a single part-time developer
- Missing features
  - Lack of resiliency (NMIN)
    - Missing in most stages
    - Across collections
  - No resource reservations before submission - racing on Slurm resources
  - Slow deployments
  - No active monitoring
    - All processes may die and ODC will report everything as happy
- Completely different from ECS!

Orchestration @ HPC farm - DDS

Dynamic Deployment System

- Automates deployment of user defined processes
- Handles dependencies
- Modular resource management
  - We are using Slurm
- Service (DDS Commander) and clients (DDS Agents)
- Different task specification language

```
<topology name="myTopology">
  [... Definition of tasks, properties, and collections ...]
  <main name="main">
    [... Definition of the topology itself, where also groups can
be defined ...]
  </main>
</topology>
```

DDS Challenges

- No active maintainers
- **Many issues with NMIN deployment**
  - Specific node going down in Slurm can take down the entire job
  - Underallocated Slurm job or a slow node will cause a timeout of the deployment
- Slurm drains some nodes when shutting down the environment via KILL transition
- Nearly impossible to debug all the issues
- Slow deployments
  - We had a case of topology distribution that took ~3 minutes

Orchestration @ HPC farm - DPL + FairMQ

Data Processing Layer (DPL)

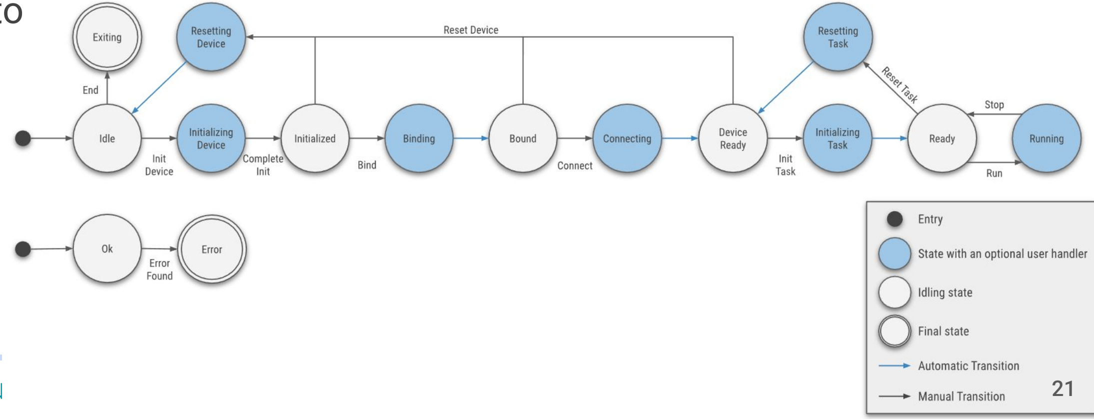
- Developed in ALICE
- Creates and uses a static topology to manage process IO
  - Needs to be set before starting processes
  - No service discovery

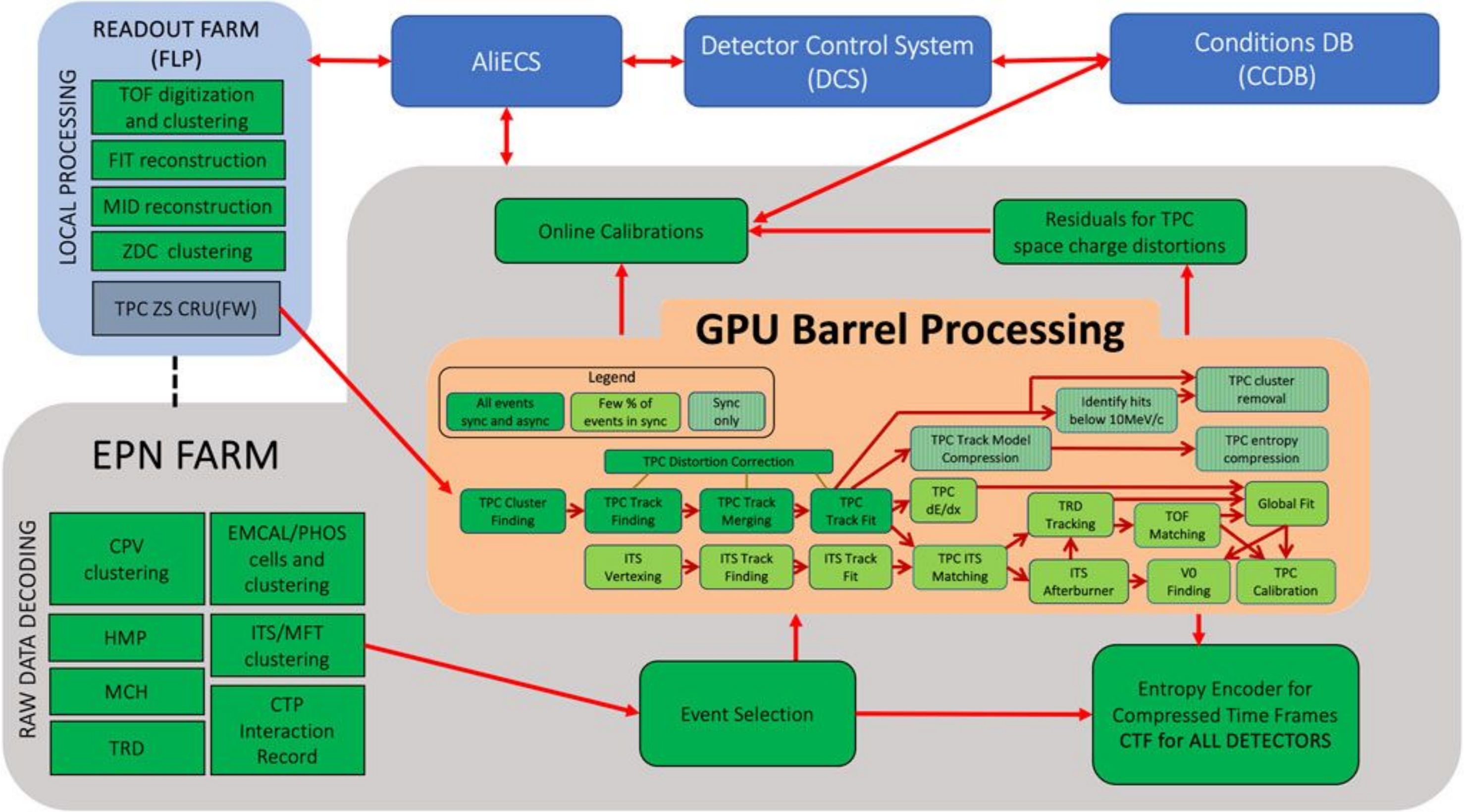
Challenges

- Most issues faced by O2/PDP are due to topology handling

FairMQ

- Developed at GSI - collaborating institute
- Provides an asynchronous message passing API
  - IPC - Inter Process Communication
  - Using shared memory
- Provides a state machine for the processes





High level view of a topology, focused on GPU barrell processing tasks

Experience with our orchestration systems

Apache Mesos (used in the readout farm)

ODC/DDS (FairRootGroup) and Slurm (used in the event processing farm / HPC)

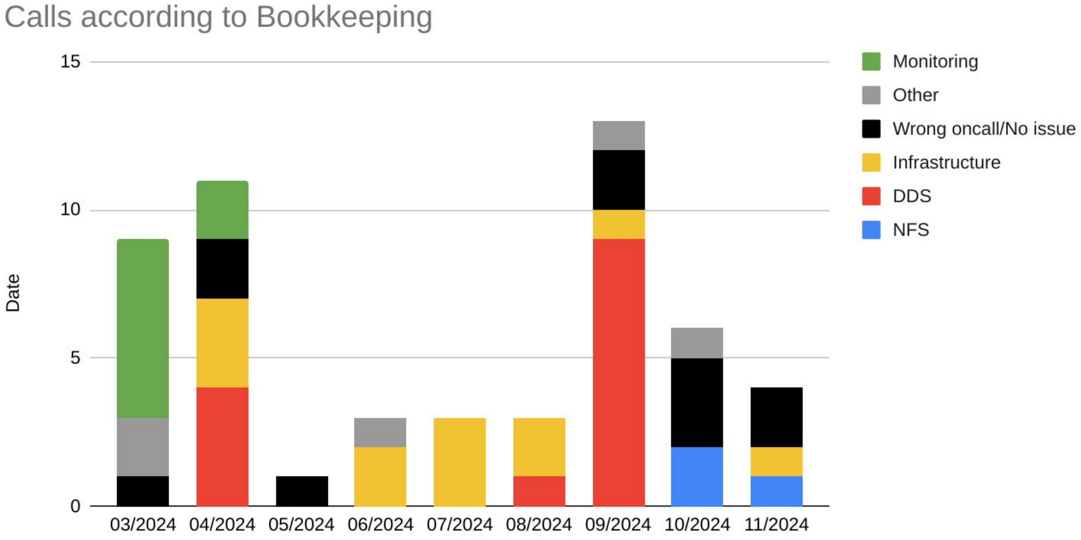
FairMQ (our IPC framework based on ZeroMQ, with state machine)

DPL (workflow framework for FairMQ processes)

Maintainability and operational constraints

Incidents in the HPC farm

- **Highest number of incidents caused by the orchestration stack**



Component (external facing)	**Calls**	**Total: 37**
Not EPN problem	**11**	
Access Management	0	
Alerting	0	not in production
Backups	0	
DataDistribution	0	
DDS	**14**	
Docs	0	
EPN-CTL	0	not in production
EPN2EOS	0	
FairMQ	0	
Infrastructure / CR0	**12**	
Logging	0	
Monitoring	**8**	Mostly InfluxDB
Network	0	
ODC	0	
Provisioning	**3**	Only NFS
SHM-Tool	0	
Slurm	0	
Topology	0	

Lessons Learned

Experience with Apache Mesos

- Why Mesos and not Kubernetes?
  - The decision was taken in 2018, when Kubernetes was less mature and Mesos still popular
  - Mesos allowed to run applications bare-metal, while we did not know whether it would be possible to run everything in containers
  - FLP software is highly static - little benefit from Kubernetes orchestration
- Experience
  - Now - abandonware
  - Often insufficient documentation
  - Did not solve our resource allocation/isolation issues (cgroups)

Lessons Learned

Experience with Golang in AliECS implementation

- What is Go?
  - “Modern C” - minimal syntax and feature set
  - Goroutines and channels
  - Garbage collection
  - Easy build system and package manager
  - Free and open source
- Nice because:
  - Quick to learn and read with C/C++ experience
  - Fast building and deployment (just copy over locally built binary to prod)
  - Rich set of available packages
  - Nice tooling
- Not-always-nice because:
  - Simplicity implies a lot of boiler-plate code
  - Writing multi-threaded applications is still error-prone
  - Unfamiliarity in the HEP community
- Would I still choose Go?
  - Probably yes

Lessons Learned

Experience in reusing tasks across data-taking runs

- Our data-taking and processing systems use a state machine that AliECS controls
  - DEPLOYED -> CONFIGURED -> RUNNING -> CONFIGURED -> DEPLOYED -> DONE
- If deployment takes a long time, one may attempt to reuse the tasks across multiple runs
  - DEPLOYED -> CONFIGURED -> RUNNING -> CONFIGURED -> RUNNING -> CONFIGURED -> …
- In practice we did not manage to achieve this, because:
  - The effort was "postponed" during early global commissioning and revived a few years later
  - Varying code quality (>200 contributors, C++, ROOT), leading to all kinds of memory corruption
  - Difficult to debug locally, as we have no tools to drive the state machine on a laptop setup
  - Fixing and testing is slow, as typically one issue hides others and long deployment cycle slows down discovering next issues in the line
- When deployment becomes faster, reusing tasks is less needed
- If this is a requirement:
  - ensure there is a streamlined environment to test and fix the processing software
  - …or consider approaches without a state machine

Lessons Learned

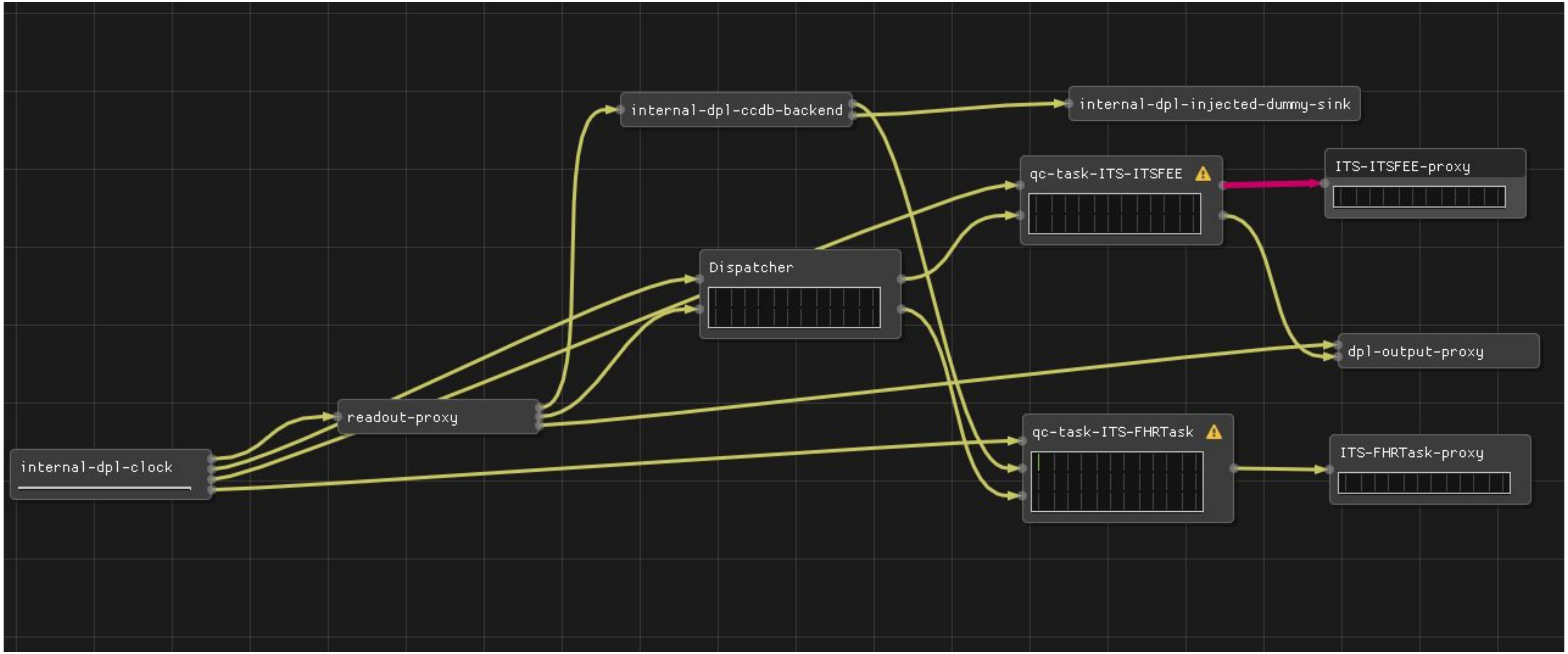
Understanding the dataflow

alio2-cr1-flp190 | InfoLogger FLP | Mesos

Name	PID	Locked	Status	State	Host Name	More
readout	11259	🔒	ACTIVE	RUNNING	alio2-cr1-flp190	⌄
stfbuilder	11260	🔒	ACTIVE	RUNNING	alio2-cr1-flp190	⌄
stfsender	11261	🔒	ACTIVE	RUNNING	alio2-cr1-flp190	⌄
jit-0a5ab59d9be6ed5c736f63d145954d153601ebcc-internal-dpl-clock	11267	🔒	ACTIVE	RUNNING	alio2-cr1-flp190	⌄
jit-0a5ab59d9be6ed5c736f63d145954d153601ebcc-readout-proxy	11272	🔒	ACTIVE	RUNNING	alio2-cr1-flp190	⌄
jit-0a5ab59d9be6ed5c736f63d145954d153601ebcc-internal-dpl-ccdb-backend	11277	🔒	ACTIVE	RUNNING	alio2-cr1-flp190	⌄
jit-0a5ab59d9be6ed5c736f63d145954d153601ebcc-Dispatcher	11283	🔒	ACTIVE	RUNNING	alio2-cr1-flp190	⌄
jit-0a5ab59d9be6ed5c736f63d145954d153601ebcc-qc-task-ITS-ITSFEE	11288	🔒	ACTIVE	RUNNING	alio2-cr1-flp190	⌄
jit-0a5ab59d9be6ed5c736f63d145954d153601ebcc-qc-task-ITS-FHRTask	11293	🔒	ACTIVE	RUNNING	alio2-cr1-flp190	⌄
jit-0a5ab59d9be6ed5c736f63d145954d153601ebcc-ITS-ITSFEE-proxy	11298	🔒	ACTIVE	RUNNING	alio2-cr1-flp190	⌄
jit-0a5ab59d9be6ed5c736f63d145954d153601ebcc-ITS-FHRTask-proxy	11303	🔒	ACTIVE	RUNNING	alio2-cr1-flp190	⌄
jit-0a5ab59d9be6ed5c736f63d145954d153601ebcc-dpl-output-proxy	11313	🔒	ACTIVE	RUNNING	alio2-cr1-flp190	⌄
jit-0a5ab59d9be6ed5c736f63d145954d153601ebcc-internal-dpl-injected-dummy-sink	11317	🔒	ACTIVE	RUNNING	alio2-cr1-flp190	⌄
shell-command		🔒	ACTIVE	RUNNING	alio2-cr1-flp190	⌄
fairmq-shmmonitor		🔒	ACTIVE	RUNNING	alio2-cr1-flp190	⌄
shell-command		🔒	ACTIVE	RUNNING	alio2-cr1-flp190	⌄

Lessons Learned

Understanding the dataflow - much more helpful



Lessons learned from incidents - Control systems

- Fragmented systems
  - Currently ECS + Mesos vs ODC + DDS + Slurm + DPG + Topogen + Data Distribution service
  - Mesos
    - Previously popular Apache project, many production users
    - Single principal maintainer before - superhero → red flag
    - Now 4 years unmaintained
  - DDS
    - Single production user - ALICE EPN
    - Unmaintained since the start of operations
    - Unsupported since the start of operations
- Extremely difficult to debug issues in fragmented systems
  - Impossible to teach new operators
  - Even experienced operators invited to make mistakes frequently
- Inherently unreliable due to lack of standards

Lessons learned from incidents - Control systems

- **Unified control system is needed across all the computing resources used**
- Clear requirements necessary
    - NMIN, Scale up / down, Heterogeneous compute, Statefulness, Background processes, Inter-process communication, Initial state (calibrations), Init jobs, etc.
- Same system to be used for services, online data taking, async processing,...
- Momentum and support
    - No superhero project, no one-man project
    - Used by many production systems
    - Backed by a large community
- Some development needed
    - To inject experiment / project semantics
    - Special features not available in the baseline
    - >=95% to be done in the framework, <5% developed

Lessons Learned - FairMQ, Data Processing Layer (DPL)

- FairMQ has limitations
  - Message passing, stateful, single process, fixed topology, 2 superheros project
  - Check other experiments, lots of publications from CHEP
  - Is message passing obsolete and too rigid?
    - Rigid topologies (or necessary custom dynamic topology handling)
    - Not safe interprocess communication
- Instead using message bus, data driven queues, in-memory data grid?
  - Seastar, Libfabric,...
- What such framework needs?
  - Focus on developer deployability + integration testing
  - Safe interprocess communication
  - Multithreading
- Ideally stateless data-driven system
  - Stateless data-driven tasks (always running, no fixed topology)

Lessons learned - Software in general

Software is not a croissant - to be baked, eaten and forgotten.
Every software project needs a dedicated maintenance throughout its lifespan!

- Data distribution
- ODC, DDS, FairMQ
- Apache Mesos

Software is are not just applications

- Configuration management
- Integrations, DevOps, CI/CD
- Test suites and automated testing frameworks

Lessons Learned - Release management & Verification

- Release management - called release coordination at ALICE
  - Absolutely essential to gatekeep releases
  - Participation has holes
  - Missing decision making based on comprehensive test results
- Staging system
  - Essential for system verification prior to rolling out to production
  - Integration of ALL components
  - All interfaces look identical to production system
  - Separate network, resources, access management, configuration, databases, K8s clusters,...
  - Does not allow scaling tests, but all other verification, including soaking tests
- Development system
  - All software must run on developers' machines
  - How to test a single FairMQ task locally?
  - Framework must support this

Lessons Learned - Resiliency

- Too many SPOFs in across multiple dependent components
  - Cumulative downtime on the entire system impacted
  - Several known components are SPOFs: NFS, ECS, Data Distribution, ODC, Subnet manager
  - Each SPOF needs to be identified and: {ignored, defined recovery process, removed SPOF)
- Resiliency vs. High-availability vs. Time to recovery
  - Not resilient: one node rebooting itself breaks the entire data taking
  - Not highly available: Data distribution scheduler is not automatically backed up by another
  - Long time to recovery: Need to investigate 10 components to find a root cause just to resume
- Hardware resiliency and recovery
  - Failing hardware necessitates frequent reactions from the team
  - HW failure → investigation → replacement → burn-in test → resume
  - Even once a week hardware recovery takes a lot of time
  - Suggestion: recover hardware in a batch every time 5% of resources are unavailable

General requirements of orchestration systems in ALICE

- online/async/batch jobs
  - horizontal scaling
    - configurability

Orchestration - Requirements

Failure resiliency

- The ability to lose tasks / collections / nodes
- In a managed and transparent way

State persisting across multiple environments

- Ex: Shared memory on the EPNs, only reset across runs
- Calibration objects to kickstart the environment

Background processes

- EPN2EOS writer - can take hours to empty the disks

Performance

- It takes ~20-25s to deploy all the processes on each EPN
- ODC+DDS takes ~60-100s to configure, start, deploy and transition all the tasks across all the EPNs
- There can be > 150k tasks in one env on all the epns

Horizontal scaling

- Scaling online data-taking environments down and up
- Dynamic provisioning and releasing of online nodes to improve utilization
  - Based on a load or manual pre-allocations

Multiple users

- Online, Async, OpenStack / CERN IT
- Unify orchestration and scheduling across O2 projects and online/async

Batch/Slurm interoperability (optional)

- If we want to preserve Slurm interface
- Poor man's: exclusive kubelet / slurmd
- Kqueue, Volcano, SUNK,...

LS3 plans

Long Shutdown 3 (LS3) Plans

ALICE

- Rework configuration versioning
  - Improve on reconfigurability and fail-safety
- Seek to unify the control systems in ALICE
  - Get rid of Mesos and DDS
  - Investigate containerization and Kubernetes (see the next slides)
- Improve on automatic testing

HCP farm only

- Shared HPC farm between real-time (online) processing, batch processing (async, GRID jobs), and OpenStack Compute (VMs provided to other CERN users)
- Dynamic re-allocation / preemption of resources based on priority classes

Maybe after LS3?

- Always running systems with dynamic configuration, service discovery, etc.

Kubernetes Orchestration

Operator pattern (preliminary experience of what the ATLAS experiment is pursuing)

Implementation requirements & complexity, Project specific semantics

Integrations with control systems

Integrations with observability systems

Alternative architectures on Kubernetes (to avoid having own operators and CRDs)

Kubernetes Orchestration

Possible architecture

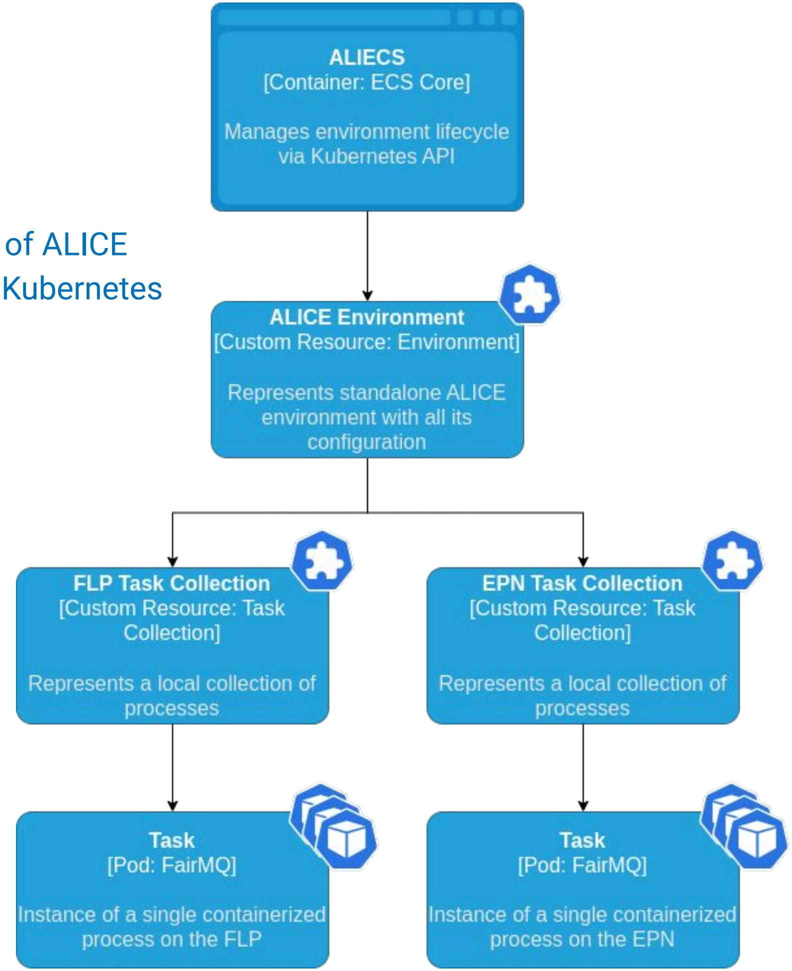
Operator pattern

- Controllers for Environments and Task Collections
- ALICE Environments (CR)
- Task Collections (CR)
- Task (rich Pods)
- Node-based objects (DaemonSets)
- Standalone legacy scripts (Jobs)
- Scale up and down deployment dynamically

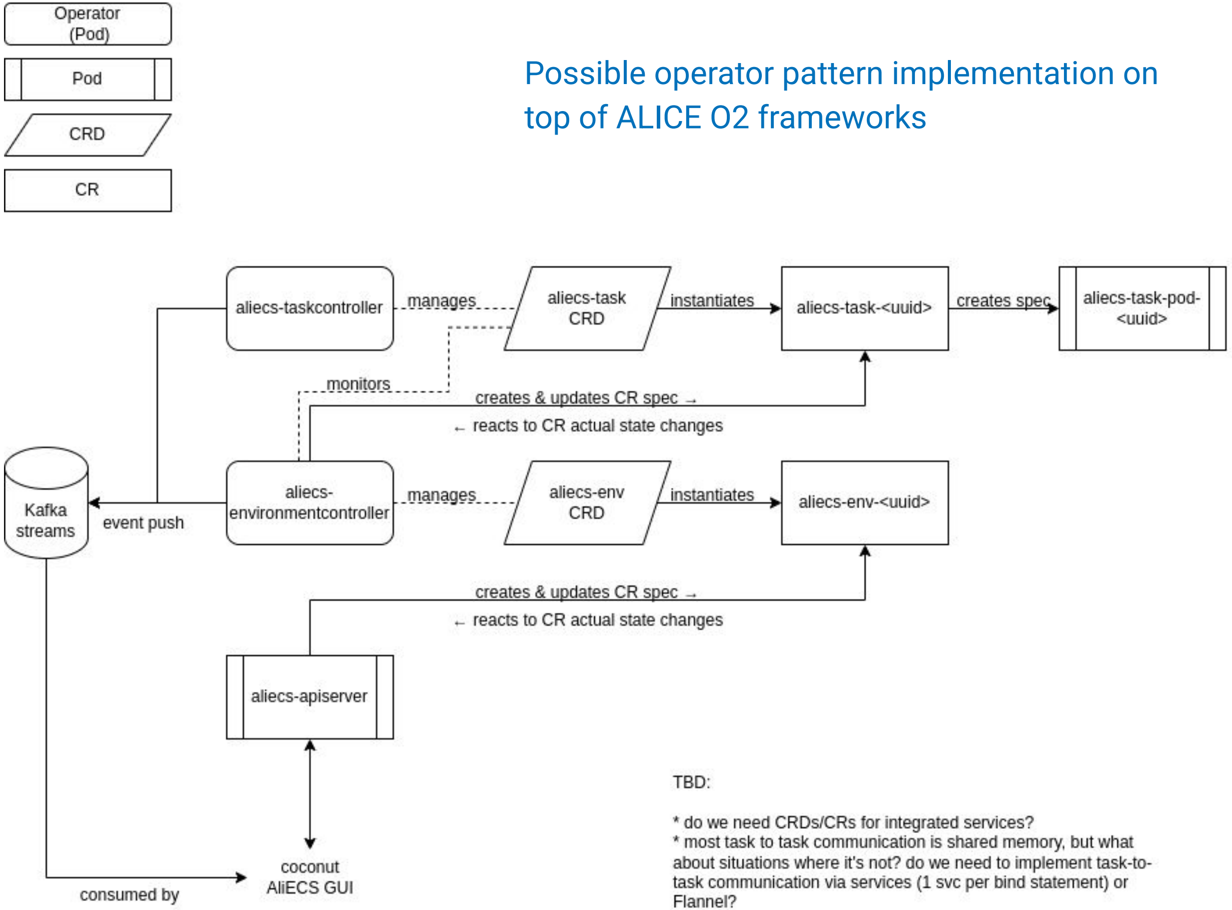
Scaling up and down deployment dynamically

- Horizontal Pod Autoscaler (HPA)
- Kubernetes Event-driven Autoscaling (KEDA)

Representation of ALICE Environment in Kubernetes



Possible operator pattern implementation on top of ALICE O2 frameworks



* detect new aliecs-task CRs, spawn pods and (if needed) services for TCP-based FairMQ channels
* detect changes to spec of aliecs-task CRs, apply changes by transitioning tasks
* catch input from tasks and task pods, react
* detect deletion of aliecs-task CRs, destroy pods

* detect new aliecs-env CRs, generate new aliecs-task CRs in response
* detect changes to spec of aliecs-env CRs (spcifically required state), write modified aliecs-task CRs in response
* detect changes to aliecs-task CRs actual state, including errors, end-of-processing and similar events, and react
* detect faults or absence in aliecs-task CRs
* connect to integrated services, spawn custom aliecs-integratedservice CRs with state information (open question: do integrated service clients need to be represented as CRs? if not, they are a component within environmentcontroller but not represented as distinct CRs, only as part of an aliecs-env CR) and handle WFT calls to services

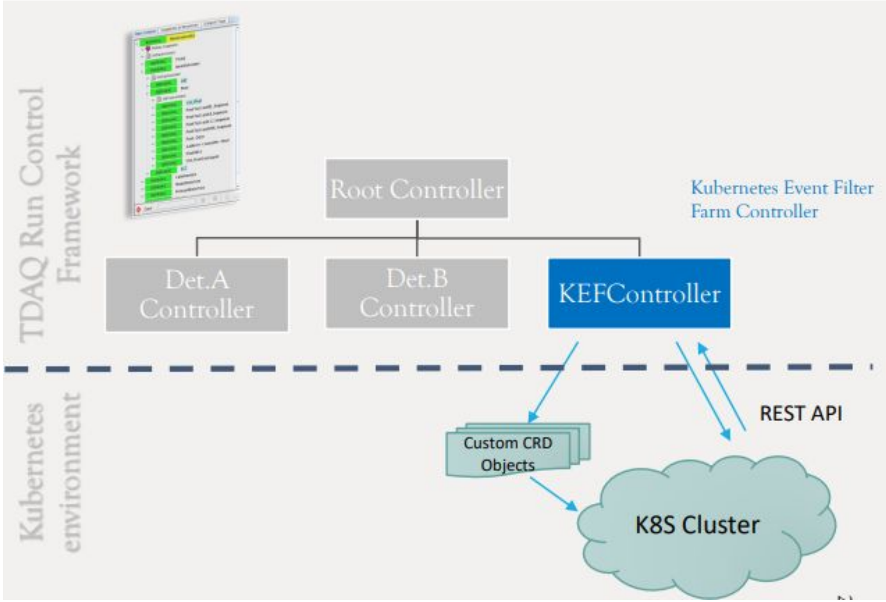
* input from coconut, legacy API support
* workflow processing
* repo system
* poll for changes in aliecs-environment and aliecs-integratedservice translate them to legacy GUI API

TBD:

* do we need CRDs/CRs for integrated services?
* most task to task communication is shared memory, but what about situations where it's not? do we need to implement task-to-task communication via services (1 svc per bind statement) or Flannel?

Kubernetes Orchestration at CERN - Operator Pattern

Prototyped already at ATLAS



Development needed

- To add the semantics to the Kubernetes Operator and Custom Resources framework
- Integrating into control system
- Integrating operations and observability

Knowhow from CERN IT

- Running many clusters
- Consulting (including design of CRD's)

Maybe not needed

- Alternatives exist without extending K8s
- Helm + CD, Crossplane, Kubevela, Kyverno

Kubernetes Orchestration at CERN - Atlas experience

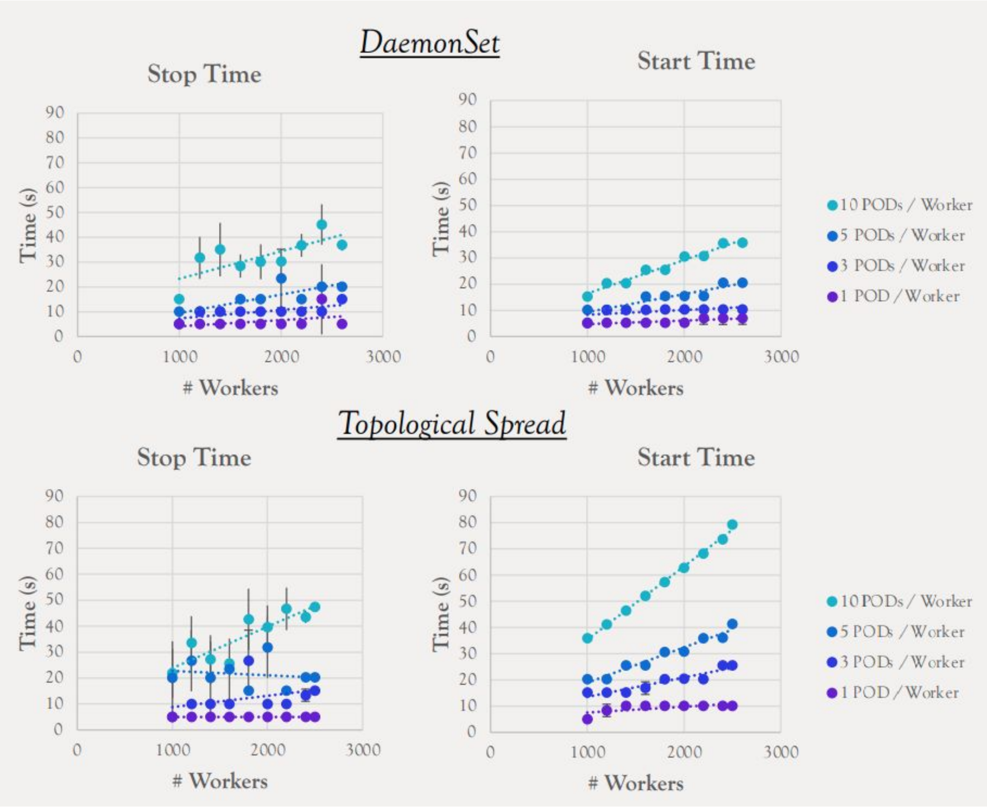
ATLAS performance

- They have 2600 nodes in the EF farm (we have 350)
- Simpler deployment on each node (10s of processes, we have up to 1000)
- They have no GPUs, no parallel gather (data dist.)

Promising startup times using a primitive deployment (pods with native scheduler)

Resources

- **ATLAS Tests** - presentation
- https://cds.cern.ch/record/2923931/files/ATL-DAQ-PROC-2025-004.pdf
- Orchestrating Quasi-Real Time Data Processing in the Computing Farm of the ATLAS Experiment - G. Avolio: https://www.youtube.com/watch?v=vUB3NzqMAzo



Kubernetes for HPC workloads

- We are not just running services and data taking jobs (like ATLAS in their EF farm)
- But we also need typical **batch workflows**
  - Async physics processing
  - External customers from CERN IT

Job Controller	Pod Scheduler
Kueue(0.10.3)	kube-scheduler
	Coscheduling(0.30.6 with a bugfix)
kube-controller-manager	YuniKorn(1.6.2)
Volcano(1.12.0-alpha.0)	

Native K8S schedulers

- Kueue + Coscheduler (CNCF)
- Volcano (CNCF)
- YuniCorn (Apache)

Slurm compatibility

- SUNK (Coreweave)
  - Proprietary
- SLINKY (SchedMD)
  - Marketing gimmick to capture K8s customers by SchedMD - the Slurm company

Kubernetes for HPC workloads - Volcano

A CNCF Cloud-Native Batch Scheduling System Designed for compute-intensive workloads.

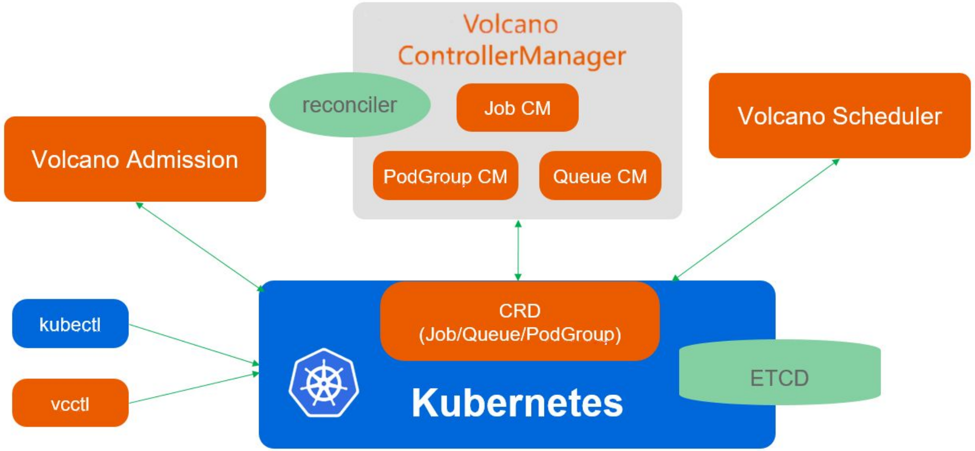
● Provides tight control over resource usage, ideal for batch processing scenarios.

● Can handle extensive resource allocation accommodating large-scale and dynamic deployments.

● Suitable for massive multi-tenant clusters, ensuring efficient resource allocation and fair balance.

● Offers priority-based scheduling, resource partitioning, and specialized metrics for detailed monitoring.

● Adapted for specialized hardware, including GPUs.



Kubernetes - Status & Maintenance Considerations

EPN work on Kubernetes so far

- **Development cluster available and deployable via kubespray**
  - **For services only at the moment**
- Alternative deployments
  - Crossplane + Talos
- Prototyping services migration e.g. InfluxDB, Infologger, Grafana
- Integration with distributed storage

Next steps

- Full staging cluster
- Ingress for services

Maintenance Pitfalls

- Many components: Container engine, SDN (Software-Defined networking) overlay, Ingress Controllers, Load Balancing, Kubelet, Kube-proxy, Kube-Apiserver
- Upgrading Kubernetes not always trivial
- Sharing of HW resources across multiple Pods
  - Cannot virtualize AMD GPUs for example
- Stateful workflows on Kubernetes may have sharp edges

Non-Kubernetes alternatives

Nomad

Slurm

Nomad

Pros

- Simple architecture (single binary)
  - Easy to deploy and operate
  - Low resource usage
- Supports various workloads
  - Containers
  - VMs
  - Binaries
- Integration with Vault (secrets) and Consul (service discovery).
- Batch and scheduled jobs, including HPC-style tasks
- Faster learning curve compared to Kubernetes.

- Much smaller community and ecosystem than Kubernetes
- Lack of built-in features (ingress, network policies, persistent volumes)
- Less multi-tenancy and policy enforcement than Kubernetes
- Fewer third-party integrations
- **Vendor lock-in**
- **HashiCorp has de-opensourced Terraform**
  - **Unpredictable**

Slurm

Possible to run the ALICE O2 framework directly on top of Slurm

Pros

- Simple setup
- Designed for HPC - batch jobs
- Efficient scheduling
- Highly scalable for large clusters
- Good accounting and resource tracking

Cons

- No built-in service orchestration
- Slurm stability issues
- Less available expertise
- Lack of documentation
- Lack of integrations with modern CI/CD
- **Vendor lock-in**
- **SchedMD has never provided good support unless paying highest premium**
- SchedMD attempts to capture K8s audience with SLINKY
  - Shallow marketing gimmick

Other Lessons Learned

Lessons learned - Flying blind

- Observability != Monitoring + Logging
- Observability needs to be designed and engineered as an integral part of the project
- The objective
  - Full visibility, transparency, availability, internal SLAs
  - Quick and clear responses to incidents
- The components
  - More robust logging system
    - Using ML for outlier ident and log-based metrics
  - APM / tracing
  - Monitoring
  - Events, alerting
  - Incident management
  - Service and node status

Lessons learned - Operations

- Operations consuming extremely high portion of the time
- Rarely does the overall system improve as a result of incident investigations
  - Example: Bug in PDP processing solved in the next release, but new bugs will be introduced
  - Solution: Improve testing process
- Incident investigation takes too much time due to lack of observability
  - No incident tracking, no professional on-call system (bookkeeping is a lackluster attempt)
- Shifters (operators) are not sufficiently trained
  - Missing up to date set of training materials, operational handbook, etc.
  - Refresher should be done before every shift block (operational handbook), not every 3-5 years

Lessons learned - Human Resources

- Avoid single person with responsibilities and knowhows - bus factor > 1
- Collapsed silos - single person with knowhow is no longer available
- Knowledge transfer and sharing necessary
  - DAQ team has knowledge transfer and rotations as integral process
  - External (non-CERN) team often lack any such processes
- Clearly defined responsibility areas (overlapping)
  - It is unclear to us who can do what, and is able to do what based on their project allowance
- Clearly defined participation
  - On-calls, Incident resolution, Bugfixes
  - Example: 20 people in EPN team in SAMS, only 3-4 actively contributing
- Early allocation of prototyping
  - Use early all components (avoid putting together all the pieces late and at the same time)
  - Commissioning will require more resources than you expect

Thank you!

References

- AliECS
  - Teo Mrnjavac et al., “AliECS: A New Experiment Control System for the ALICE Experiment”, CHEP 2023, https://doi.org/10.1051/epjconf/202429502027
  - https://github.com/AliceO2Group/Control