REGALE

REGALE architecture and REGALE library

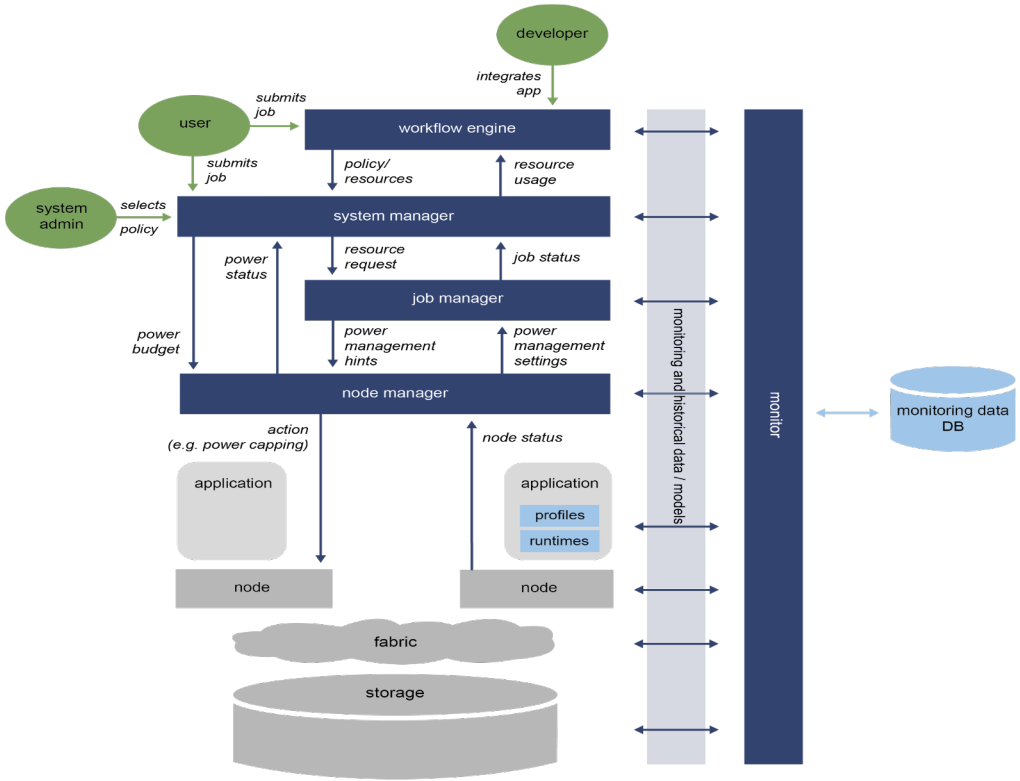
Andrea Bartolini (UNIBO),
Giacomo Madella (UNIBO), Federico Tesser (CINECA),
Julita Corbalan (BSC), Lluis Alonso (BSC), Eishi Arima (TUM)

REGALE Architecture

- Architecture
- Agents/tools

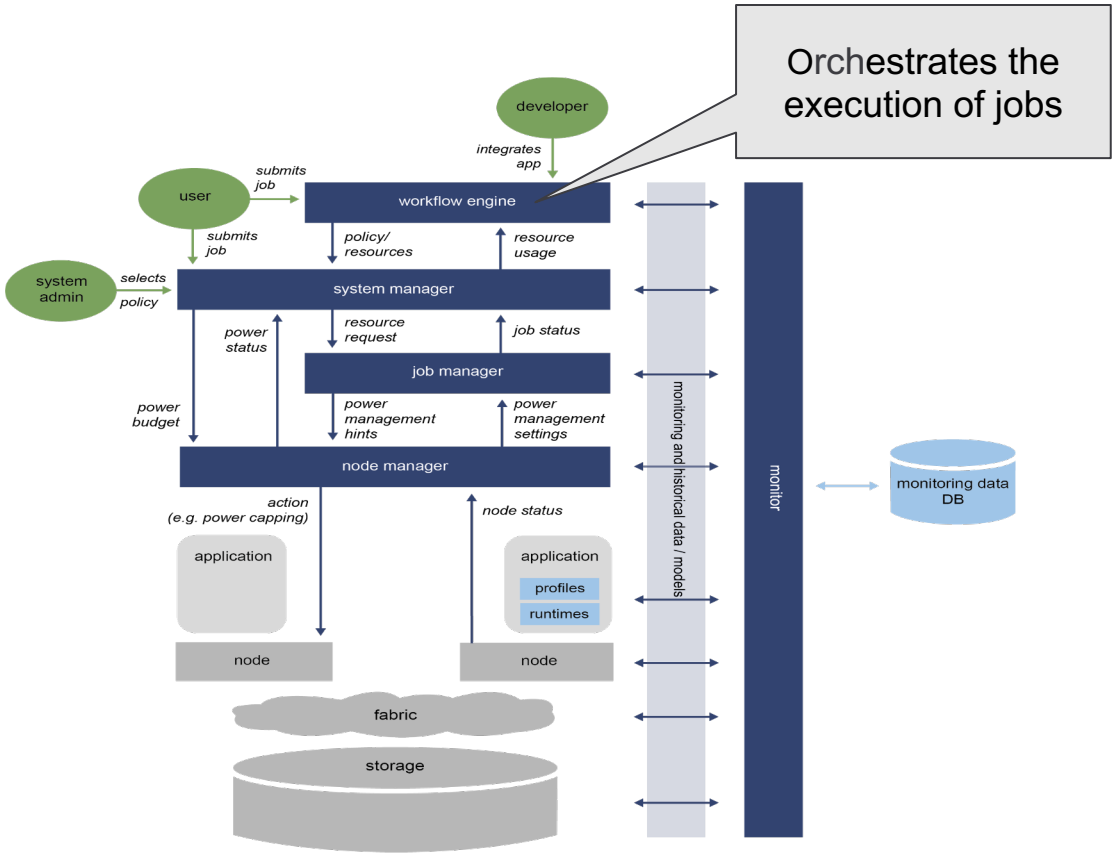
REGALE power management

Architecture



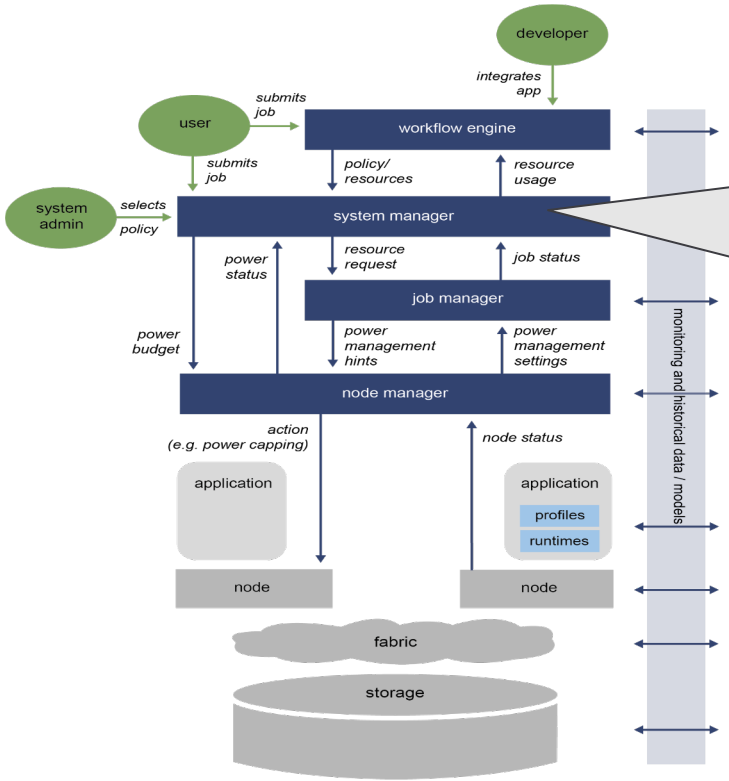
REGALE power management

Architecture



REGALE power management

Architecture



System Job scheduler

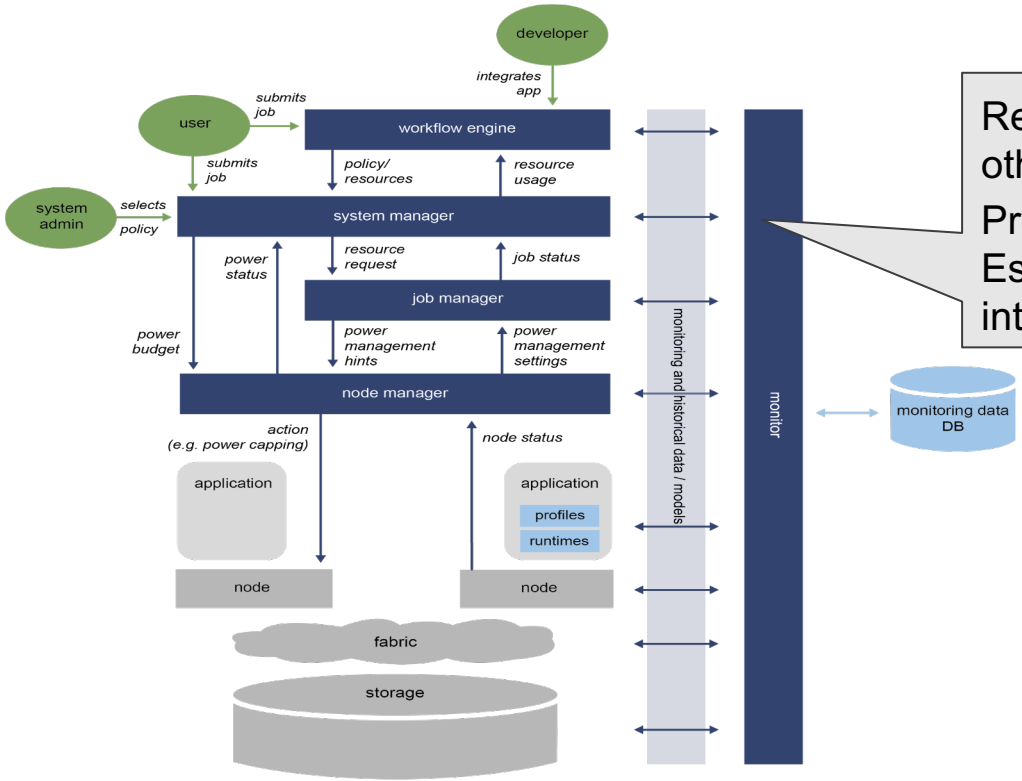
- Controls the job queue, implements the scheduling policy and resource selection, [potentially] decides the job/cluster powercap

System power manager

- Implements the job/cluster powercap. Works with the job scheduler. [Uses system predictor].

REGALE power management

Architecture

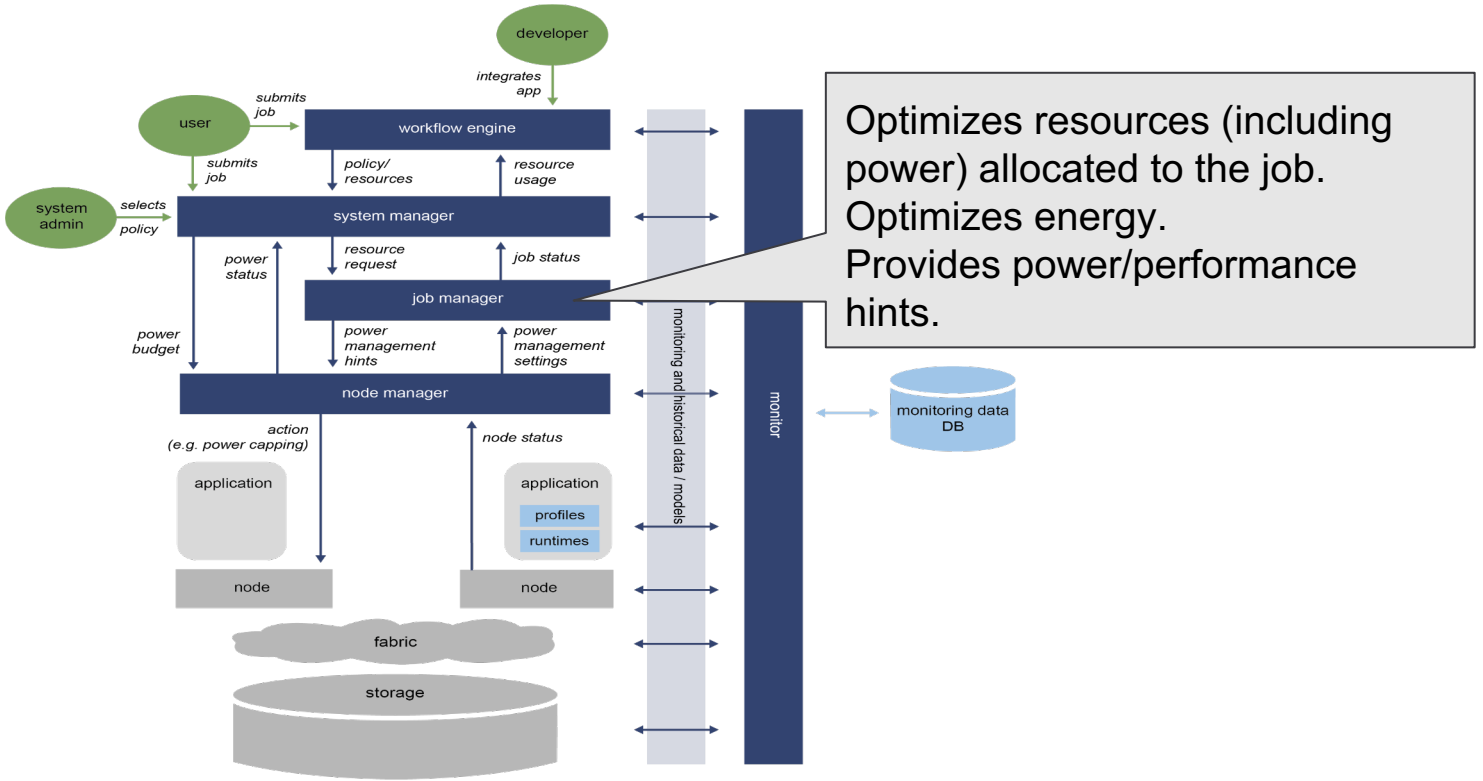


Receives data from all the other services in the systems

Provides Dasboards, Estimators and Query interface to historical data

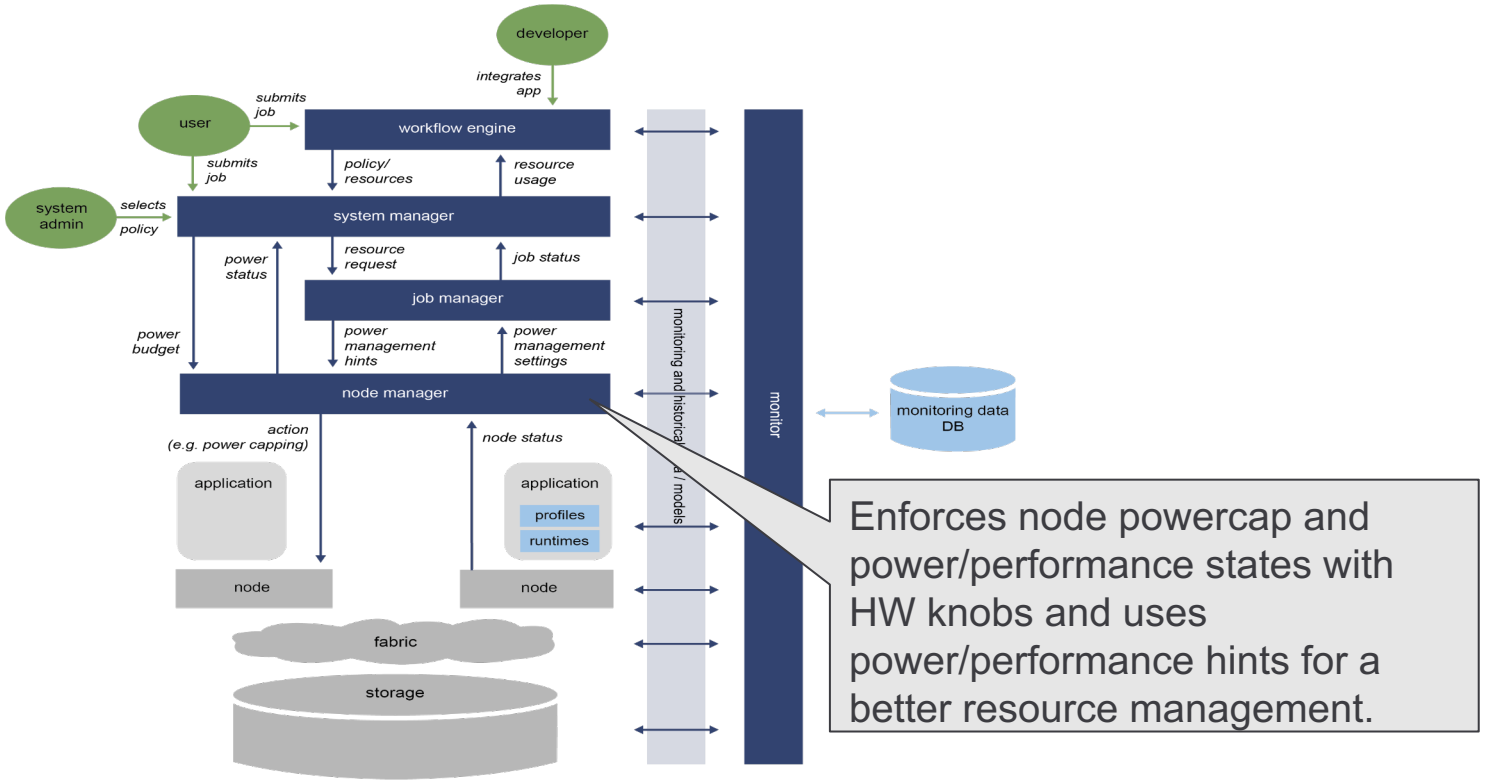
REGALE power management

Architecture

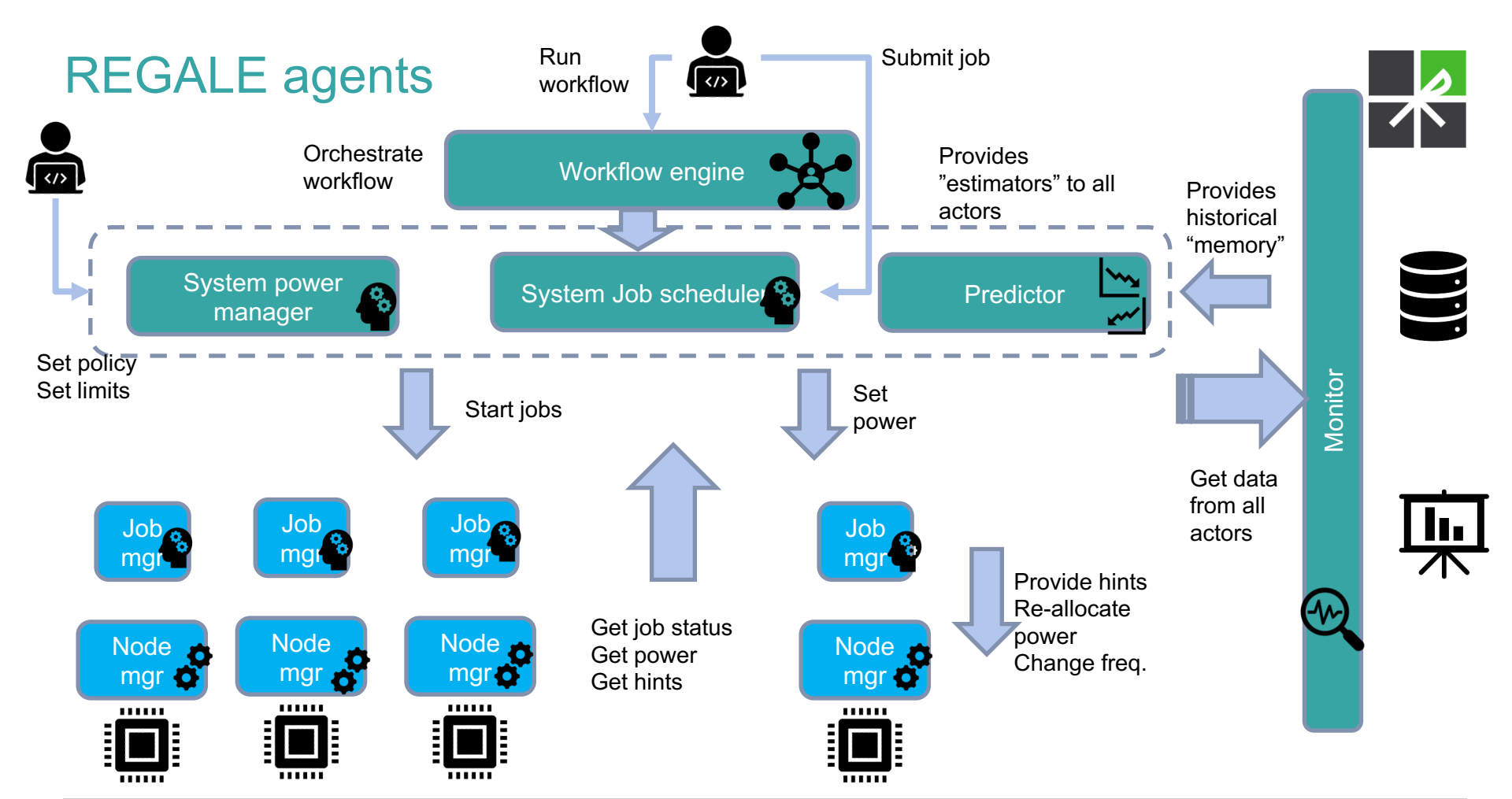


REGALE power management

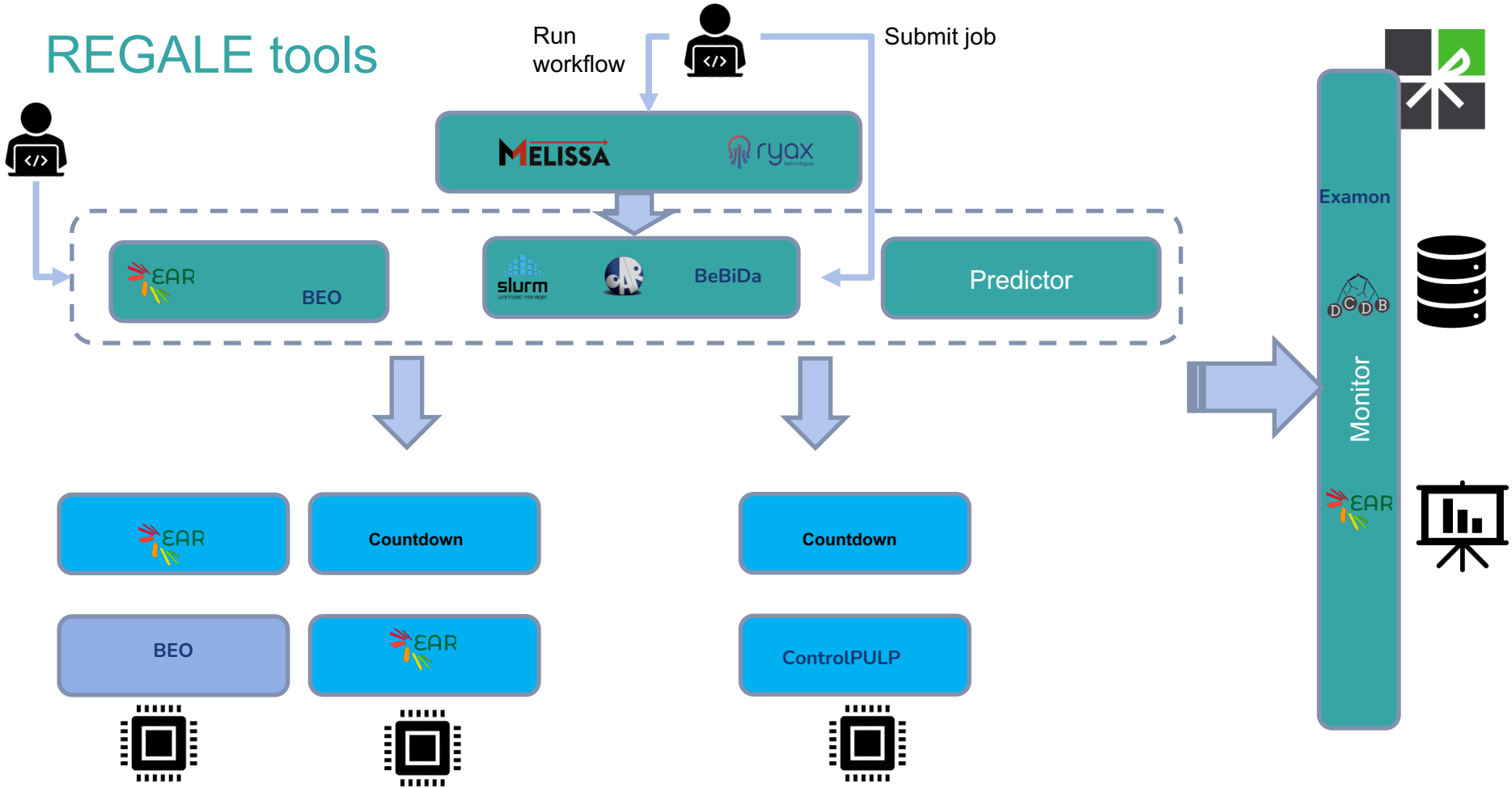
Architecture



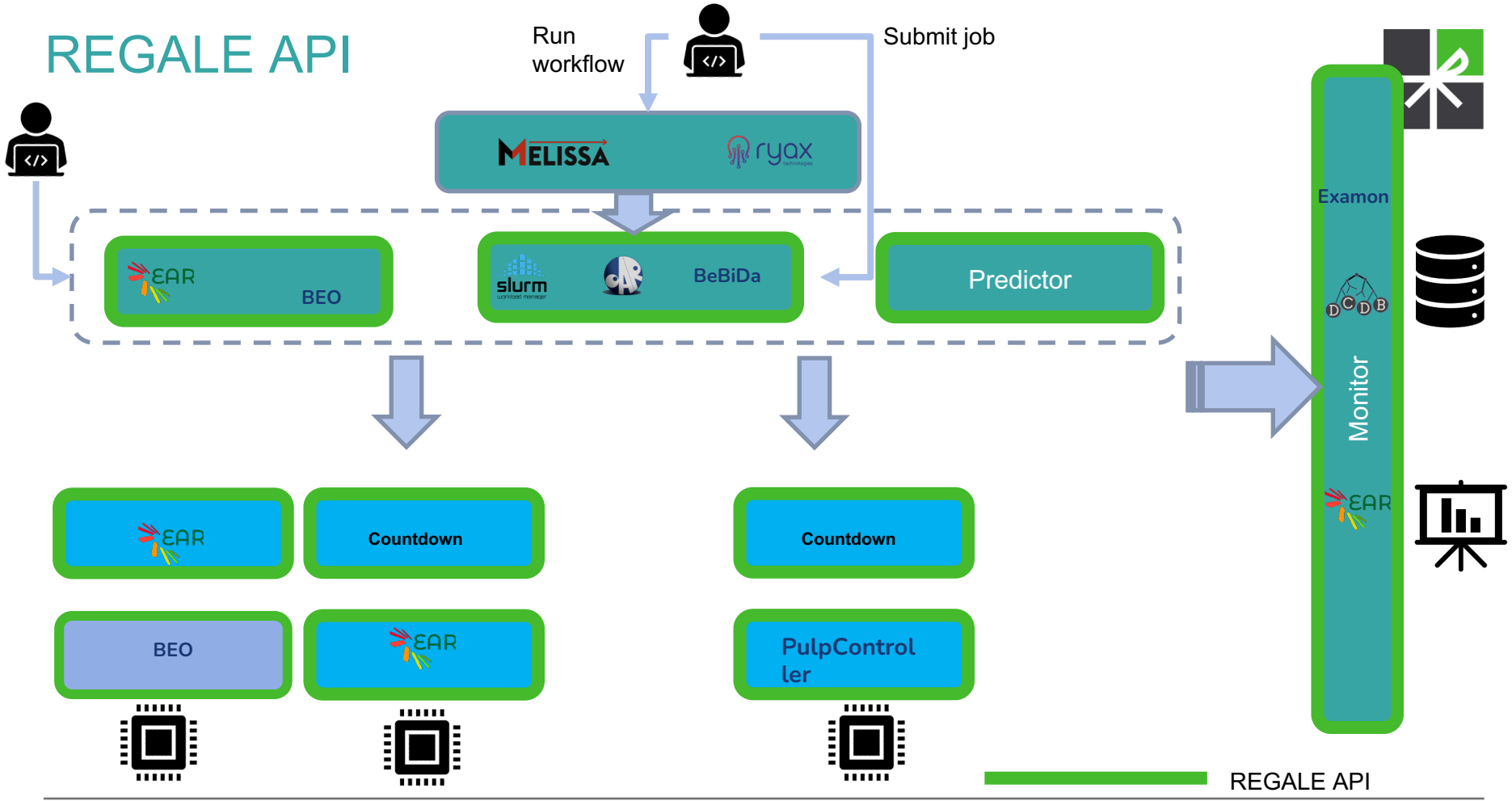
REGALE agents



REGALE tools



REGALE API



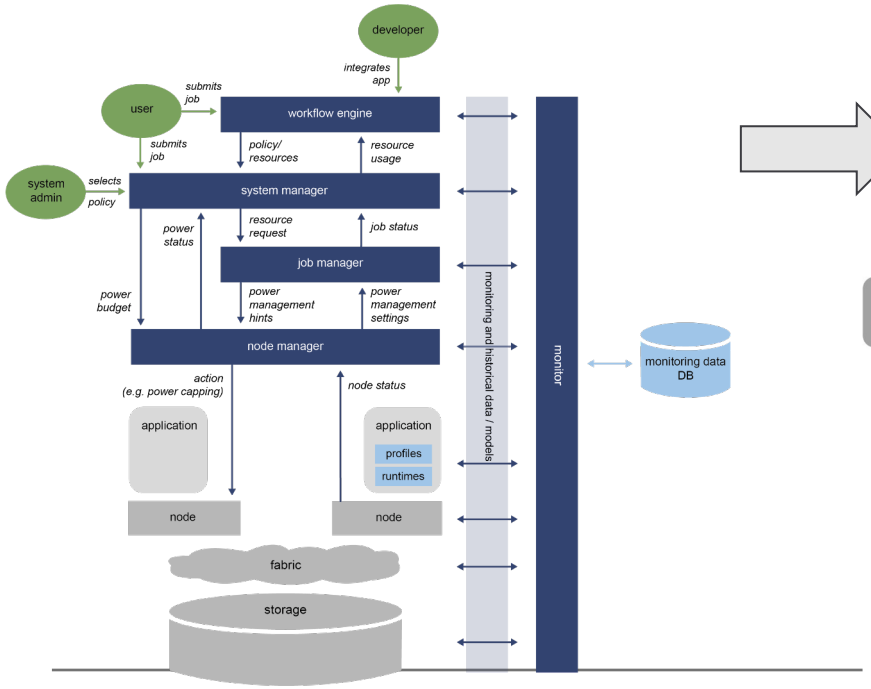
REGALE project

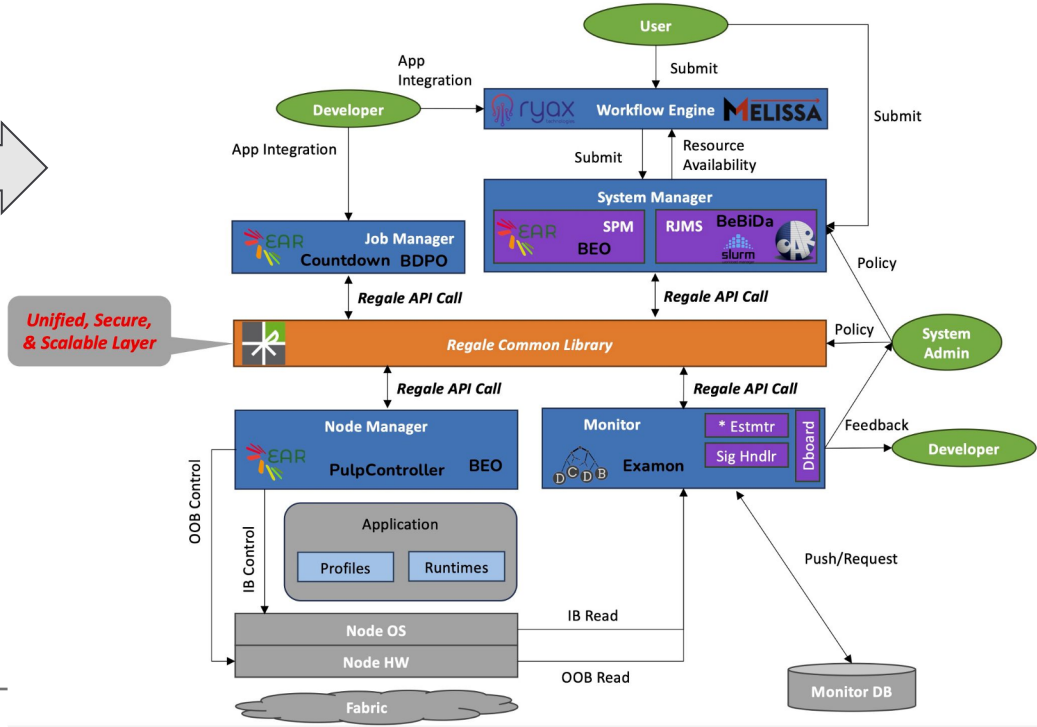
REGALE Library

- Objective & structure
- REGALE Core
- REGALE Client/Server
- REGALE Agents

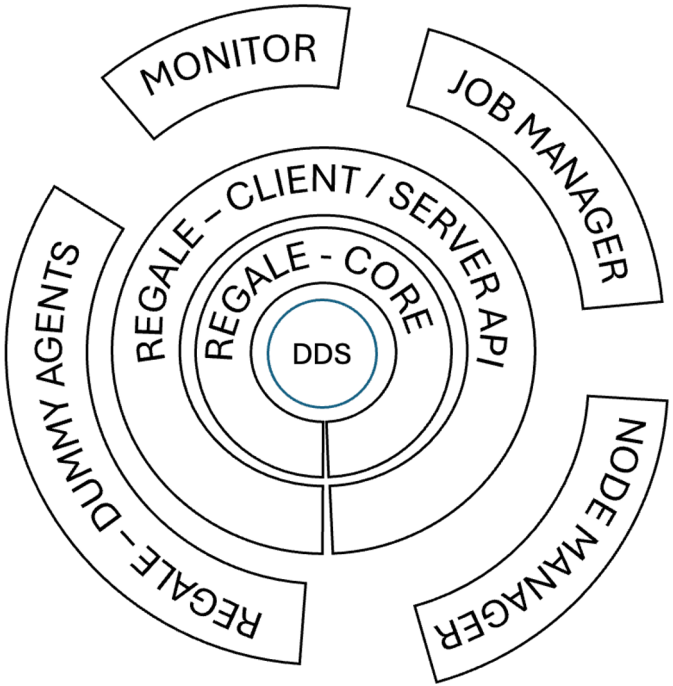
Regale Library goal

- No more specific tricks/hacks/interfaces between just two tools.
- Being able to change, in a future, an implementation for a specific Regale entity, without any issues for the others.
- No multiple invocations for the "same" functions.

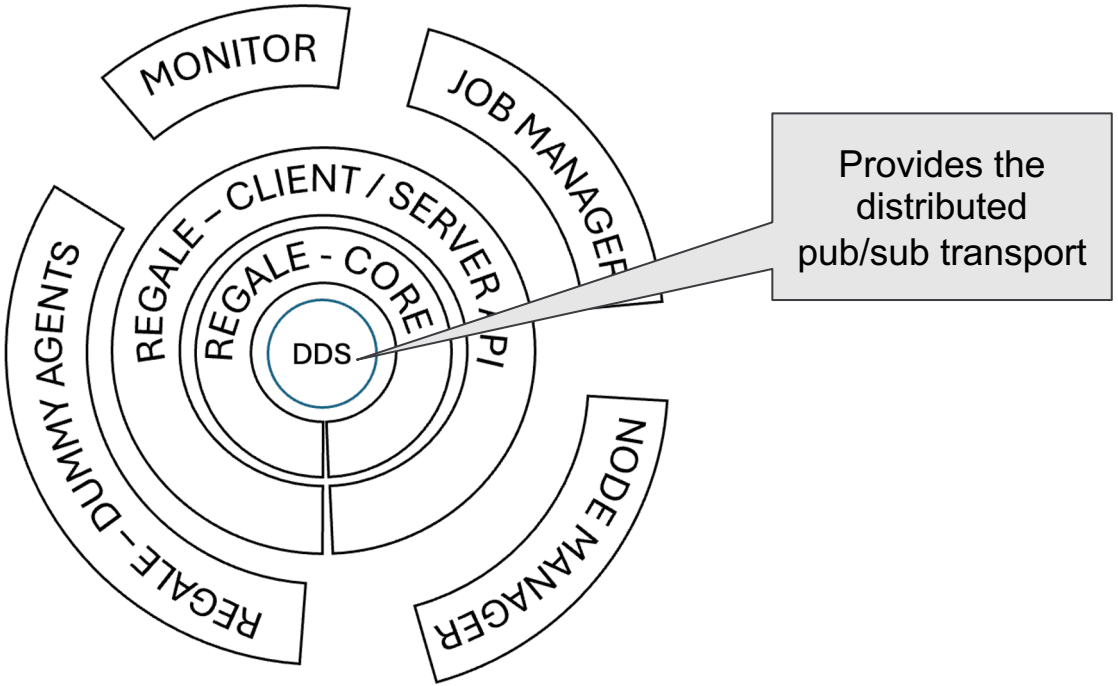




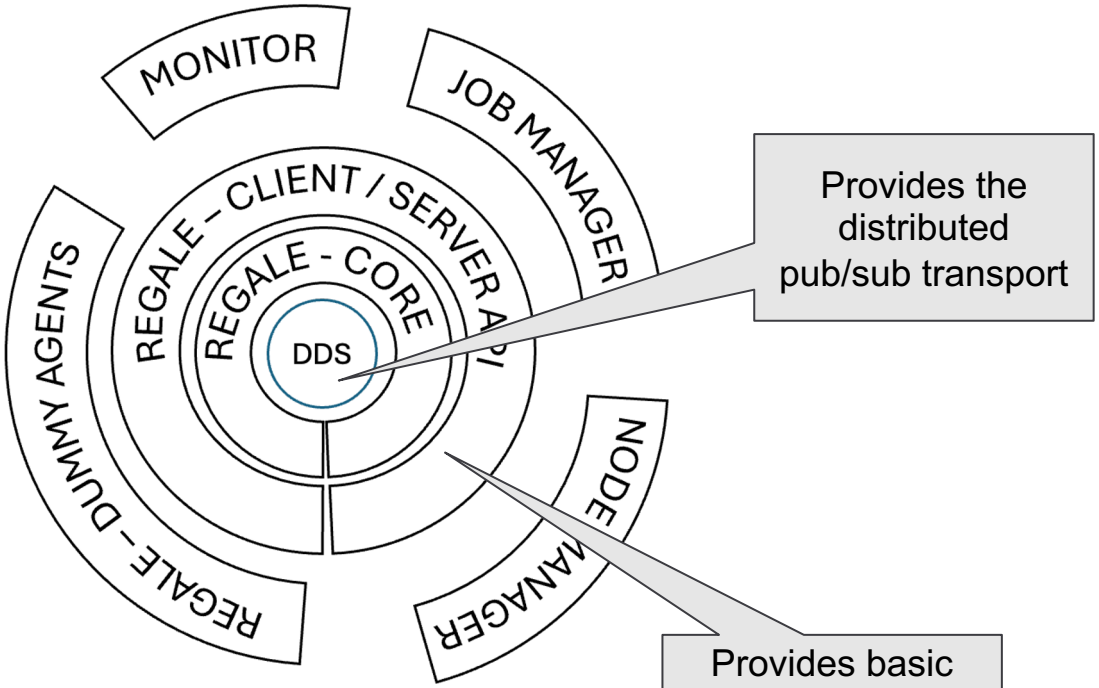
REGALE Library - a bird's-eye view



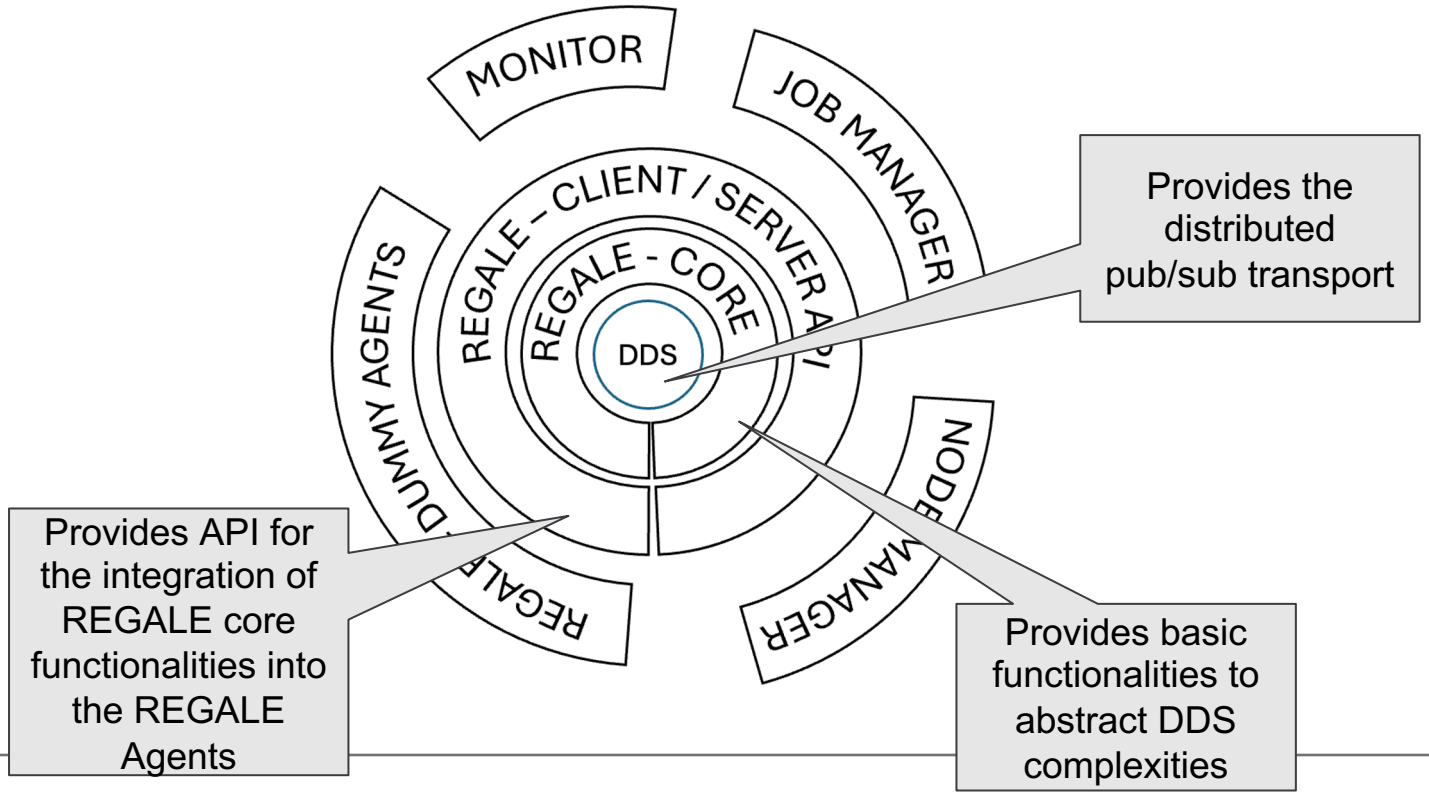
REGALE Library - a bird's-eye view



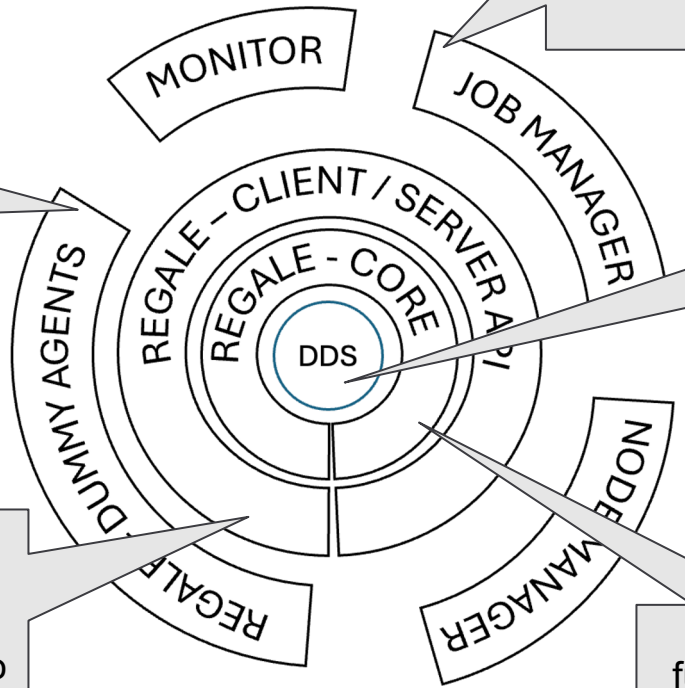
REGALE Library - a bird's-eye view



REGALE Library - a bird's-eye view



REGALE Library - a bird's-eye view



REGALE tools integrated with REGALE library

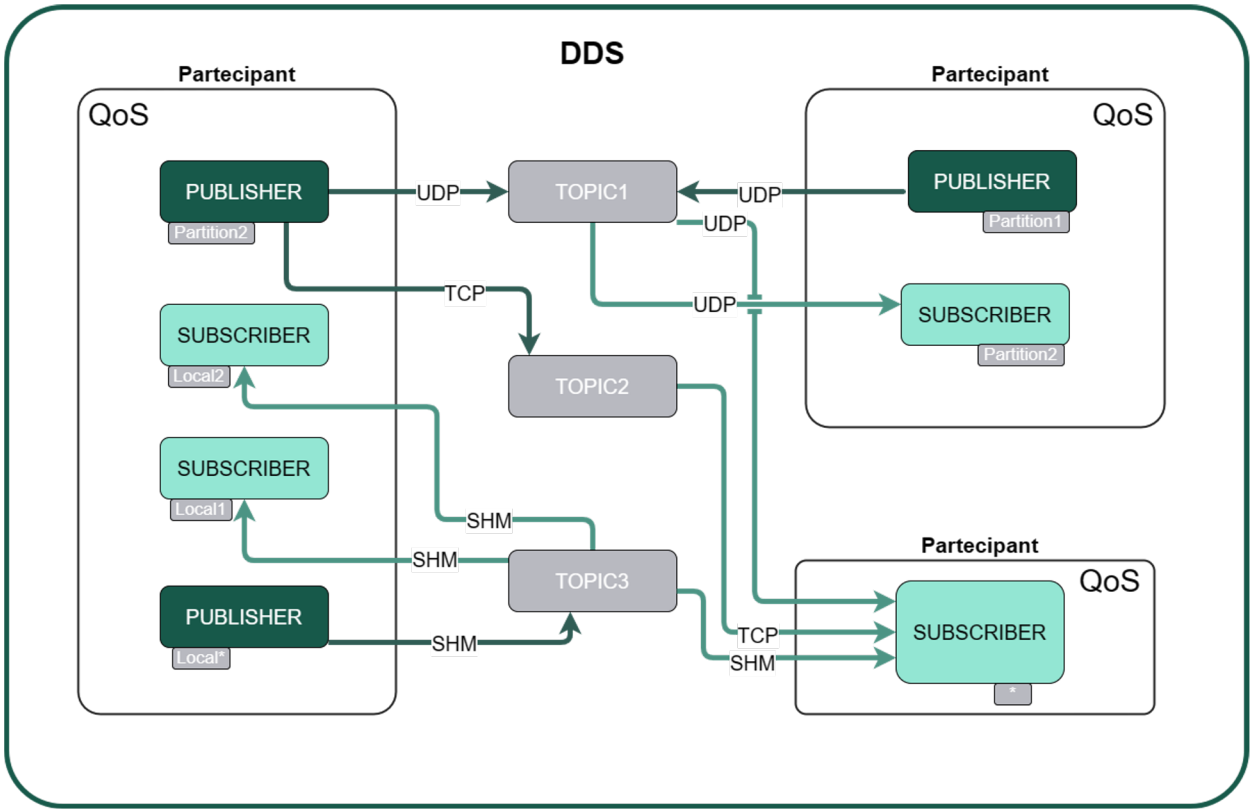
Provides dummy REGALE agents for testing and integration

Provides the distributed pub/sub transport

Provides API for the integration of REGALE core functionalities into the REGALE Agents

Provides basic functionalities to abstract DDS complexities

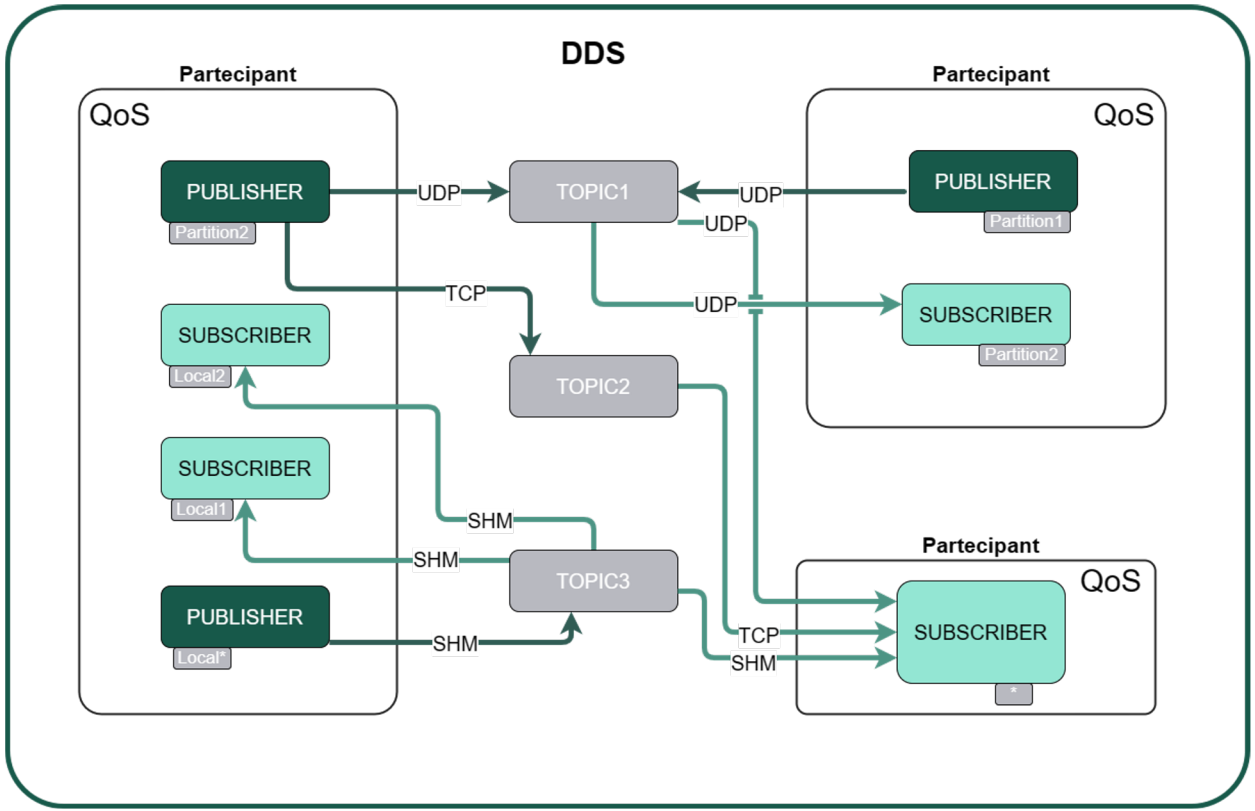
REGALE Library - DDS basics



DDS (Data Distribution Service) – a middleware standard used for real-time systems (e.g., robotics)

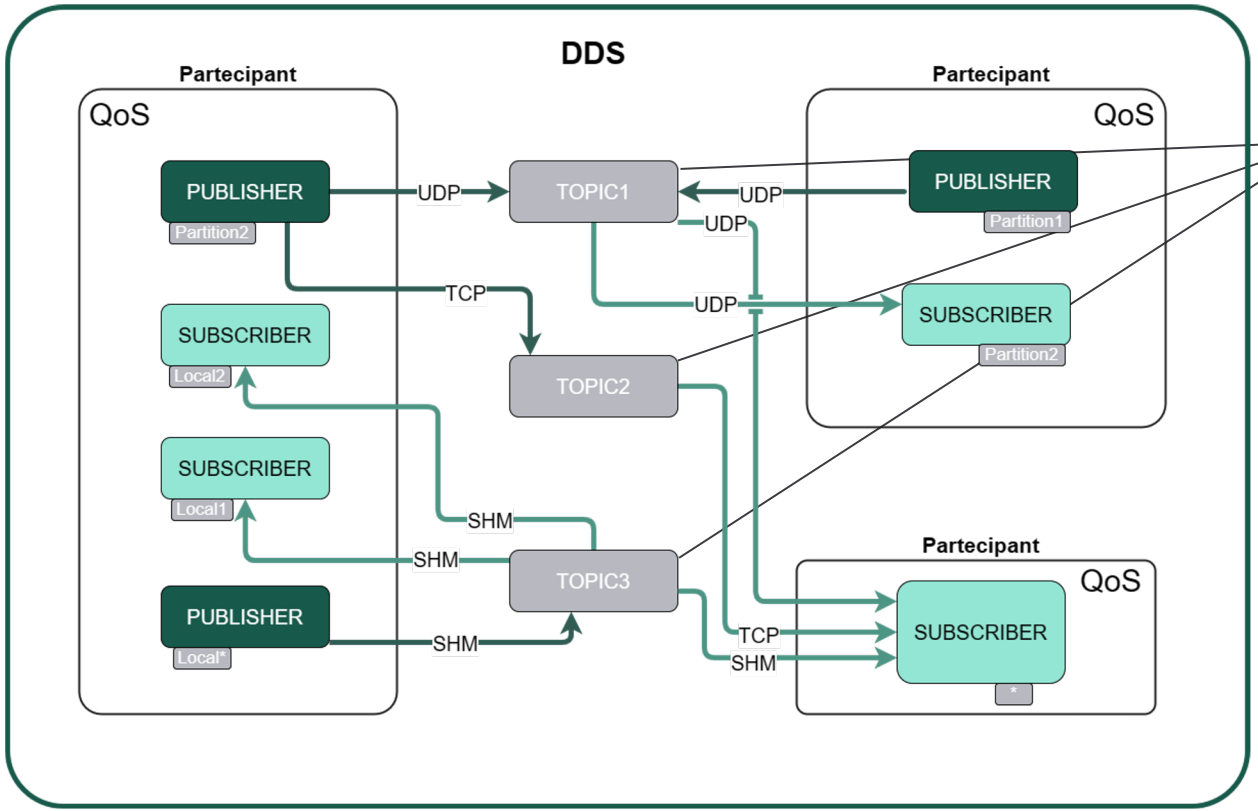
- Publish-subscribe pattern using broadcasting; Define namespace & services
- For dependability, high-performance, interoperability, real-time, scalability, etc.
- Several implementations are available; currently built on top of FAST DDS
- To be verified and validated in production HPC environment

REGALE Library - DDS basics



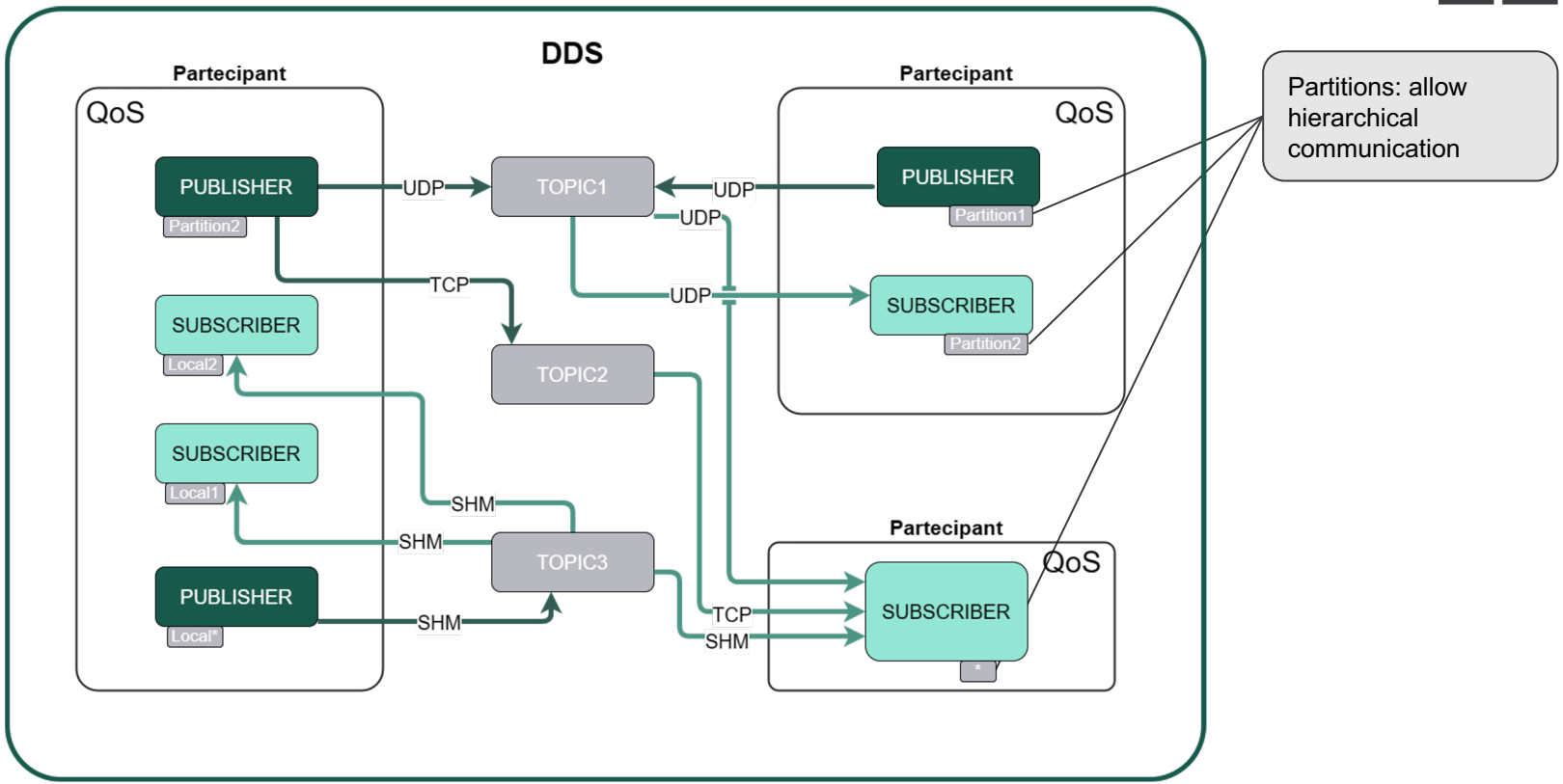
Support different domains with multiple participants

REGALE Library - DDS basics

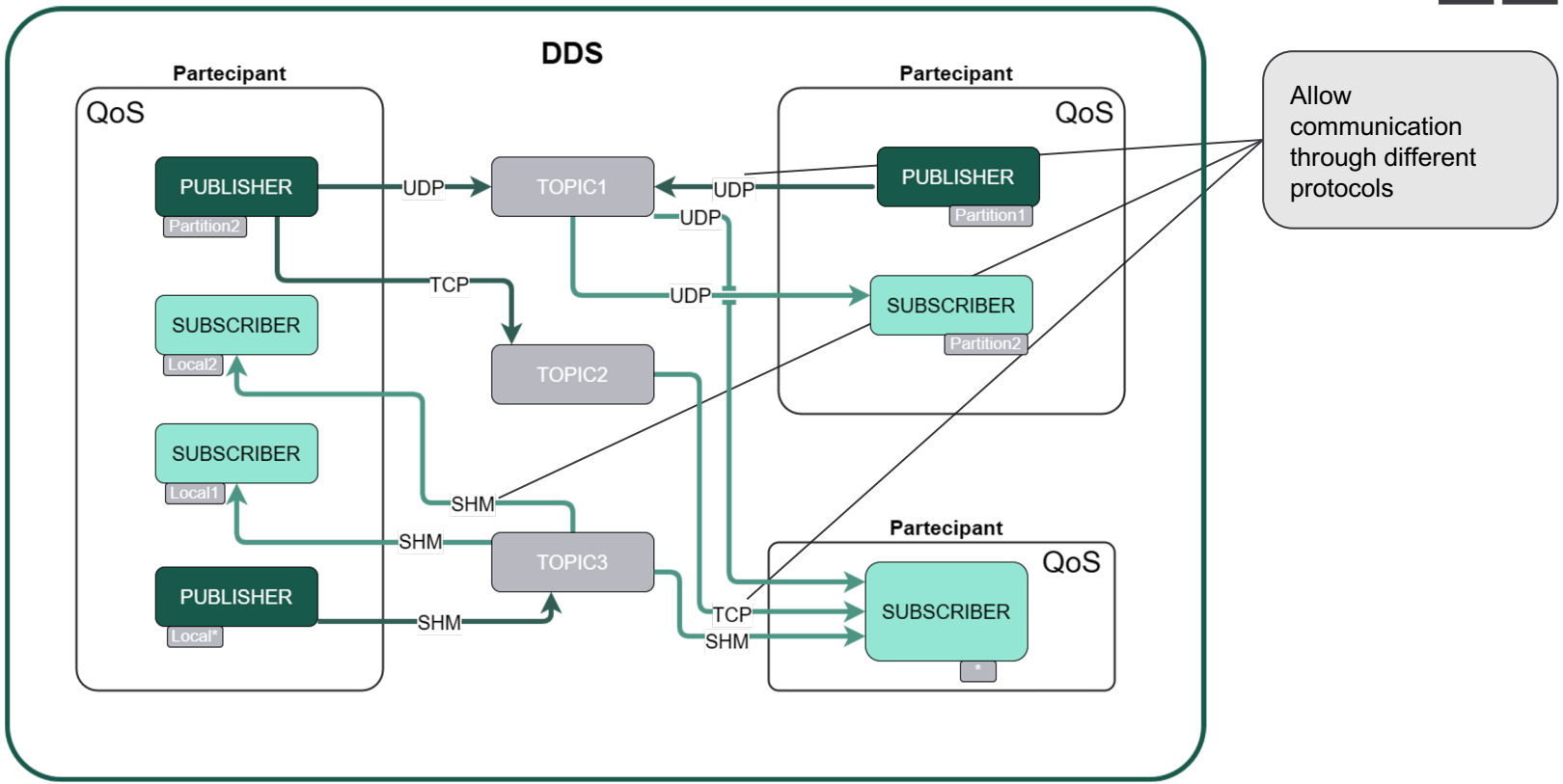


Topics: allow support for different distributed communication channel

REGALE Library - DDS basics



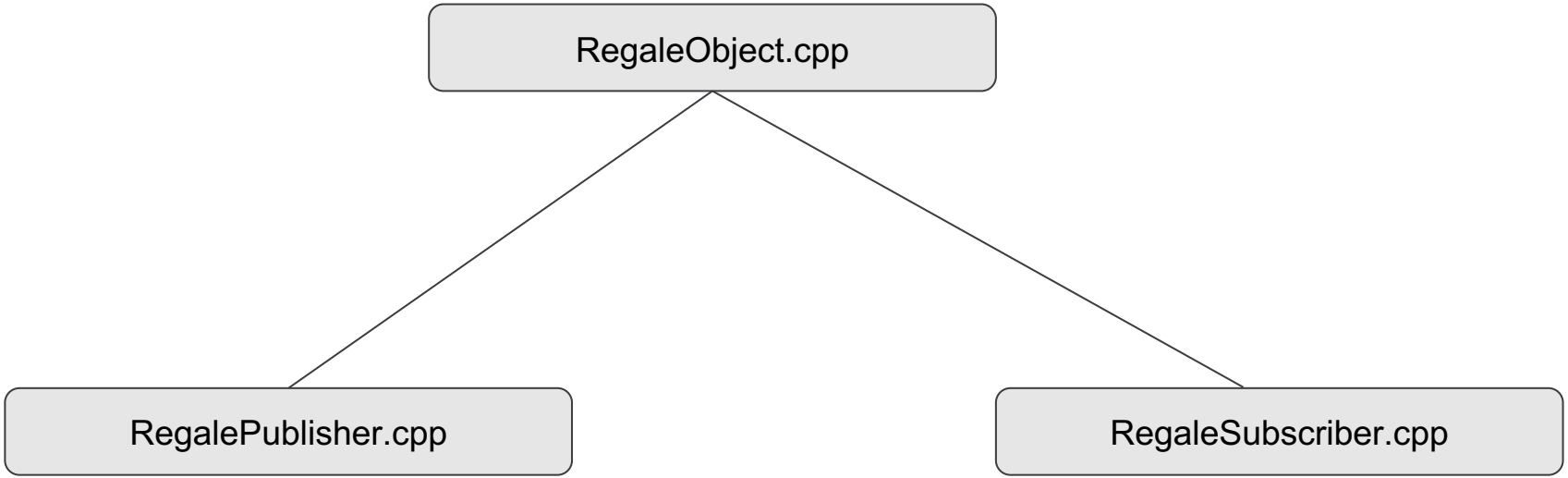
REGALE Library - DDS basics



REGALE Library

- Objective & structure
- **REGALE Core**
- REGALE Client/Server
- REGALE Agents

REGALE Library - CORE



REGALE Library - CORE

Data Types can be stored in an xml file, and parsed dynamically

RegaleObject.cpp

Base class
- QOS
- Topic
- Get/Set Dynamically Data types
- Get transport

RegalePublisher.cpp

```
<!--See: https://fast-dds.docs.eprosima.com/en/latest/fastdds/xml_configuration/making_xml_profiles.html
     section \"Rooted vs Standalone profiles definition\" -->
<dds xmlns="http://www.eprosima.com/XMLSchemas/fastRTPS_Profiles">
    <types>
        <type>
            <struct name="regale_struct">
                <member name="power" type="int32"/>
                <member name="frequency" type="float32"/>
                <member name="name" type="string"/>
                <member name="values" type="float64" arrayDimensions="10,1"/>
            </struct>
        </type>
        <type>
            <struct name="mqtt_string">
                <member name="topic" type="string"/>
                <member name="payload" type="string"/>
            </struct>
        </type>
    </types>
</dds>
```

REGALE Library - CORE

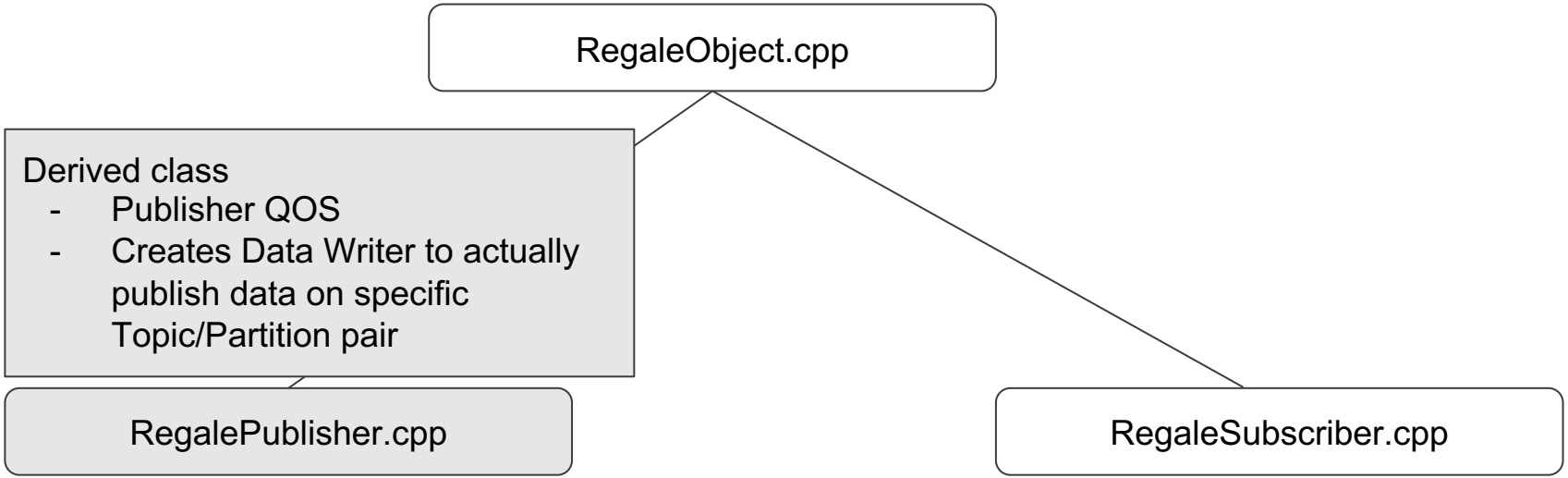
Different configurations can be set in a different xml profiles file. Among these, we can find transport setups.

RegaleObject.cpp

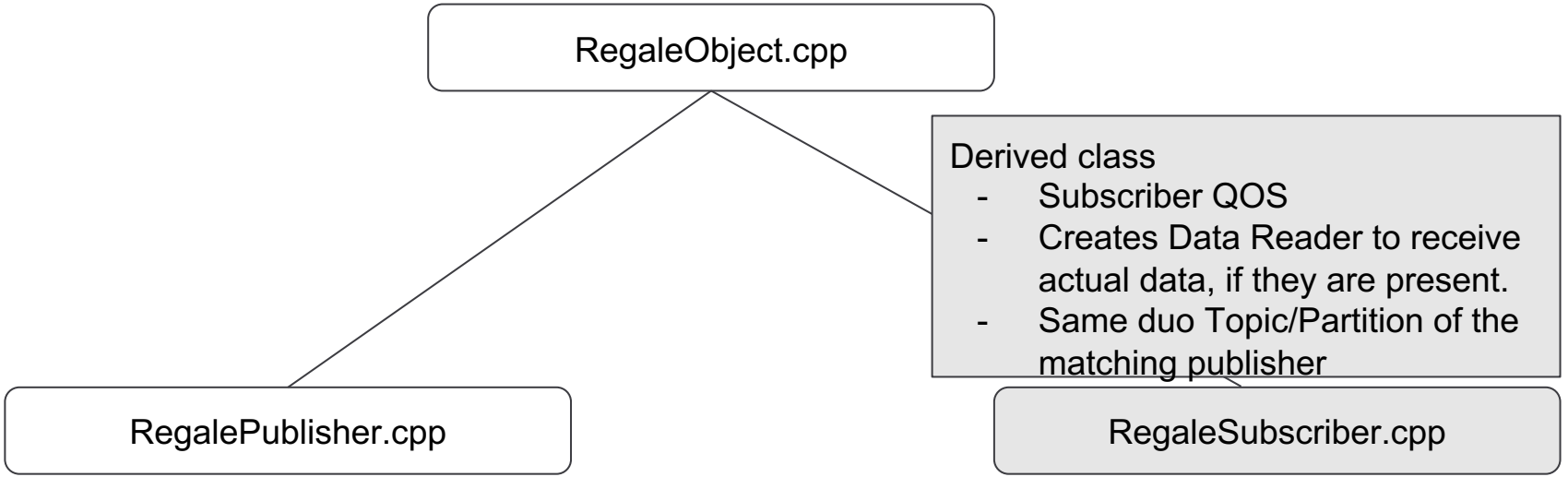
RegalePublisher.cpp

```xml
<!--See: https://fast-dds.docs.eprosima.com/en/latest/fastdds/xml_configuration/making_xml_profiles.html
     section \"Rooted vs Standalone profiles definition\" -->
<dds xmlns="http://www.eprosima.com/XMLSchemas/fastRTPS_Profiles">
    <profiles>
        <transport_descriptors>
            <transport_descriptor>
                <transport_id>udpv4_transport</transport_id>
                <type>UDPv4</type>
            </transport_descriptor>
            <transport_descriptor>
                <transport_id>shm_transport</transport_id>
                <type>SHM</type>
            </transport_descriptor>
            <transport_descriptor>
                <transport_id>tcpv4_server_transport</transport_id>
                <type>TCPv4</type>
                <listening_ports>
                    <port>local_port_for_tcp_acceptor</port>
                </listening_ports>
                <wan_addr>public_wan_address</wan_addr>
            </transport_descriptor>
            <transport_descriptor>
                <transport_id>tcpv4_client_transport</transport_id>
                <type>TCPv4</type>
            </transport_descriptor>
        </transport_descriptors>
        <participant profile_name="tcp_server_participant">
        <rtps>
            <userTransports>
                <transport_id>tcpv4_server_transport</transport_id>
            </userTransports>
            <useBuiltinTransports>false</useBuiltinTransports>
            <defaultUnicastLocatorList>
                <locator>
                    <tcpv4>
                        <wan_address>public_wan_address</wan_address>
                        <address>public_wan_address</address>
                        <physical_port>local_port_for_tcp_acceptor</physical_port>
                        <port>local_port_for_tcp_acceptor</port>
                    </tcpv4>
                </locator>
            </defaultUnicastLocatorList>
            <builtin>
                <metatrafficUnicastLocatorList>
                    <locator>
                        <tcpv4>
                            <wan_address>public_wan_address</wan_address>
                            <address>public_wan_address</address>
                            <physical_port>local_port_for_tcp_acceptor</physical_port>
                            <port>local_port_for_tcp_acceptor</port>
                        </tcpv4>
```

REGALE Library - CORE

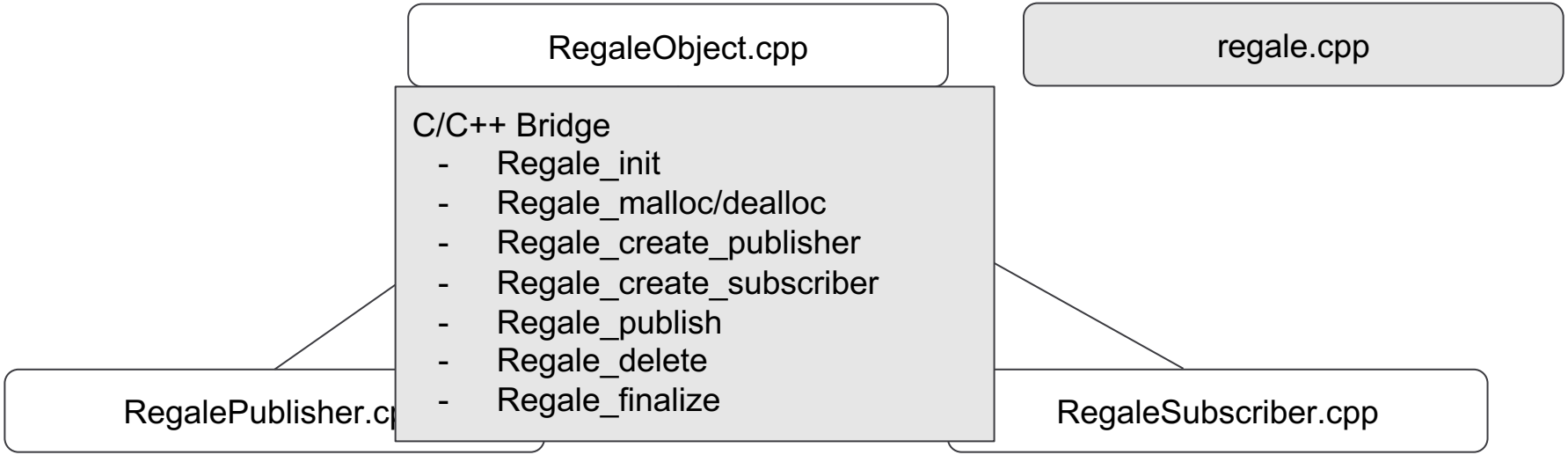


REGALE Library - CORE



REGALE Library - CORE

libregale_core.so



REGALE Library - CORE

regale_init

regale_malloc/dealloc

regale_create_publisher

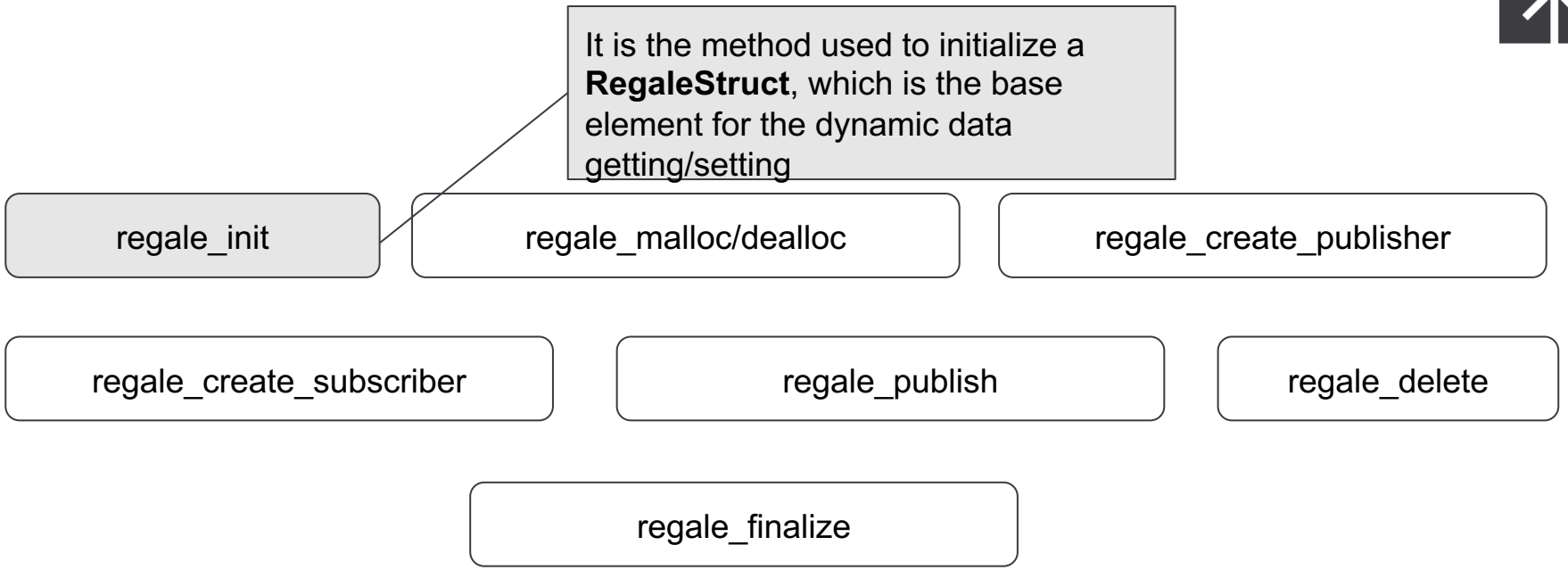
regale_create_subscriber

regale_publish

regale_delete

regale_finalize

REGALE Library - CORE



REGALE Library - CORE

It is the method used to initalize a **RegaleStruct**, which is the base element for the dynamic data getting/setting

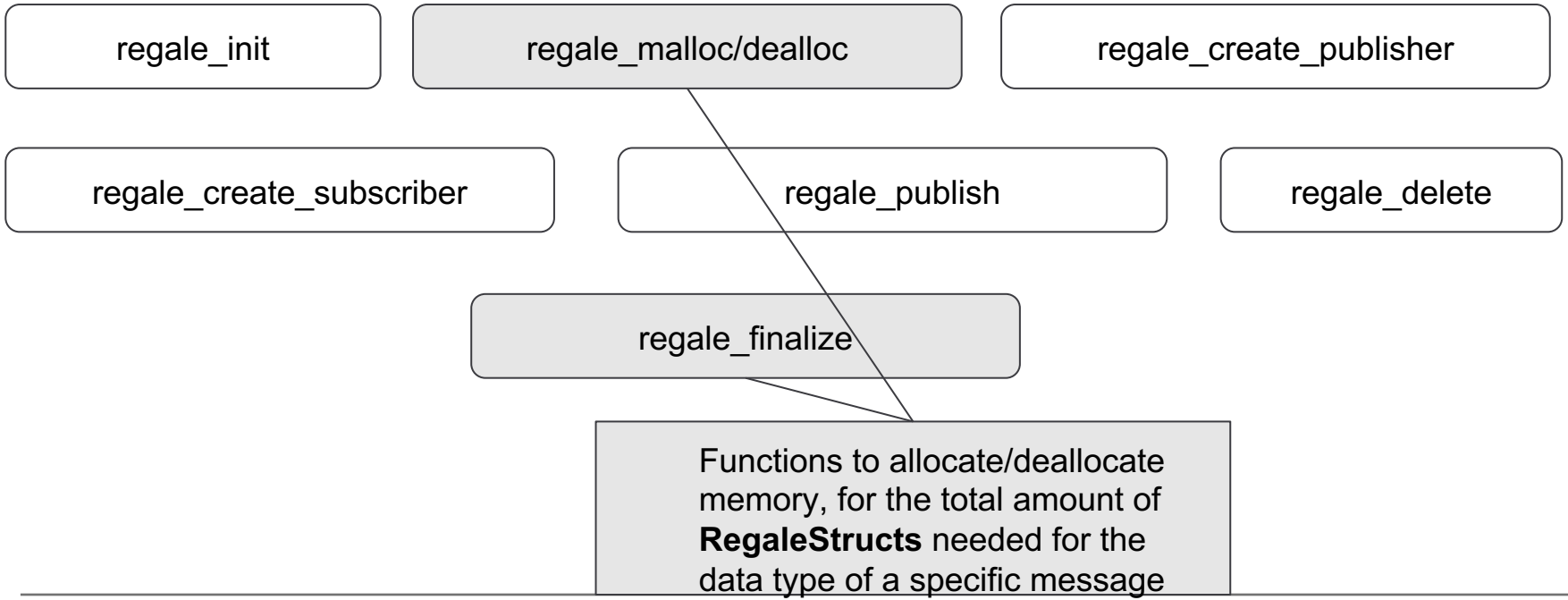
regale_init

regale_malloc/(

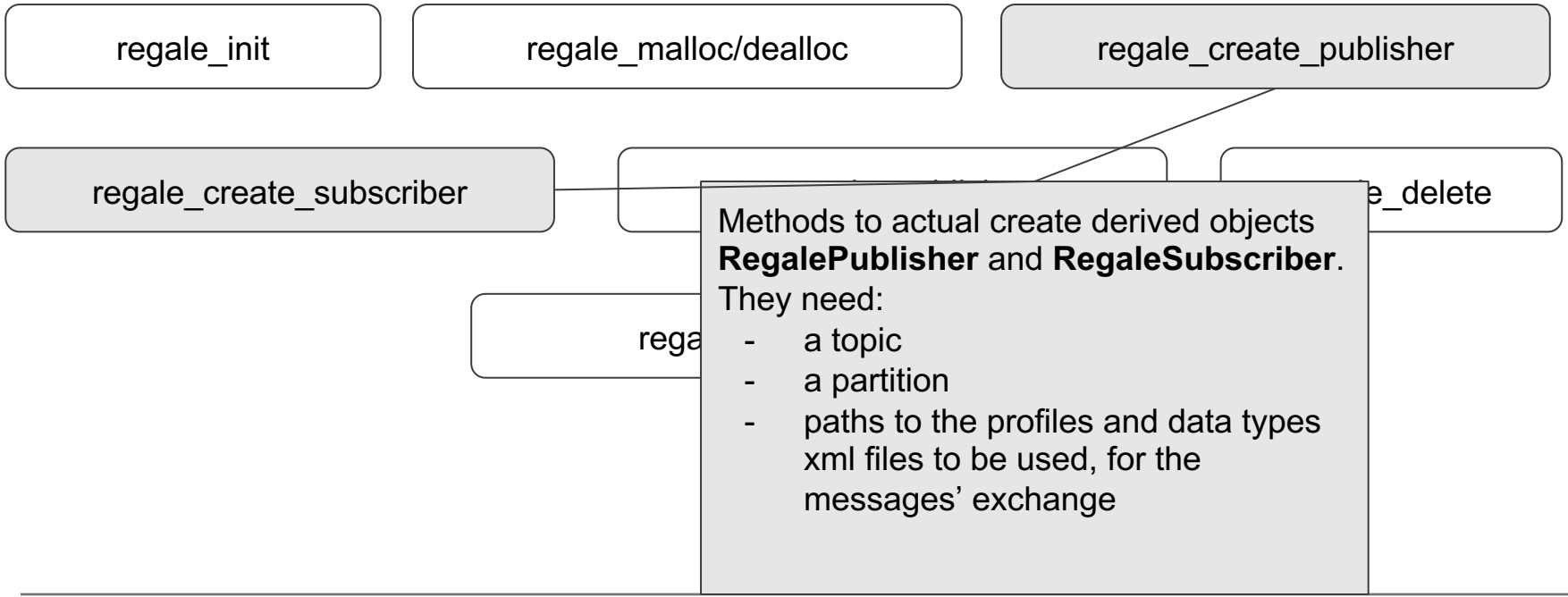
```
RegaleStruct* regale_struct = regale_malloc(4);

regale_init(regale_struct,
            1            ,
            REGALE_INT   ,
            regale_struct + 1);
(*((int*)(regale_struct->elements))) = 14;

regale_init(regale_struct + 1,
            1             ,
            REGALE_FLOAT  ,
            regale_struct + 2);
(*((float*)((regale_struct + 1)->elements))) = 30000.00;

regale_init(regale_struct + 2,
            6             ,
            REGALE_CHAR   ,
            regale_struct + 3);
strcpy((char*)((regale_struct + 2)->elements), "test");

regale_init(regale_struct + 3,
            10            ,
            REGALE_DOUBLE ,
            NULL);
for (int i = 0; i < 10; i++) {
    ((double*)((regale_struct + 3)->elements))[i] = i;
}
```

```
<type>
    <struct name="regale_struct">
        <member name="power" type="int32"/>
        <member name="frequency" type="float32"/>
        <member name="name" type="string"/>
        <member name="values" type="float64" arrayDimensions="10,1"/>
    </struct>
</type>
```

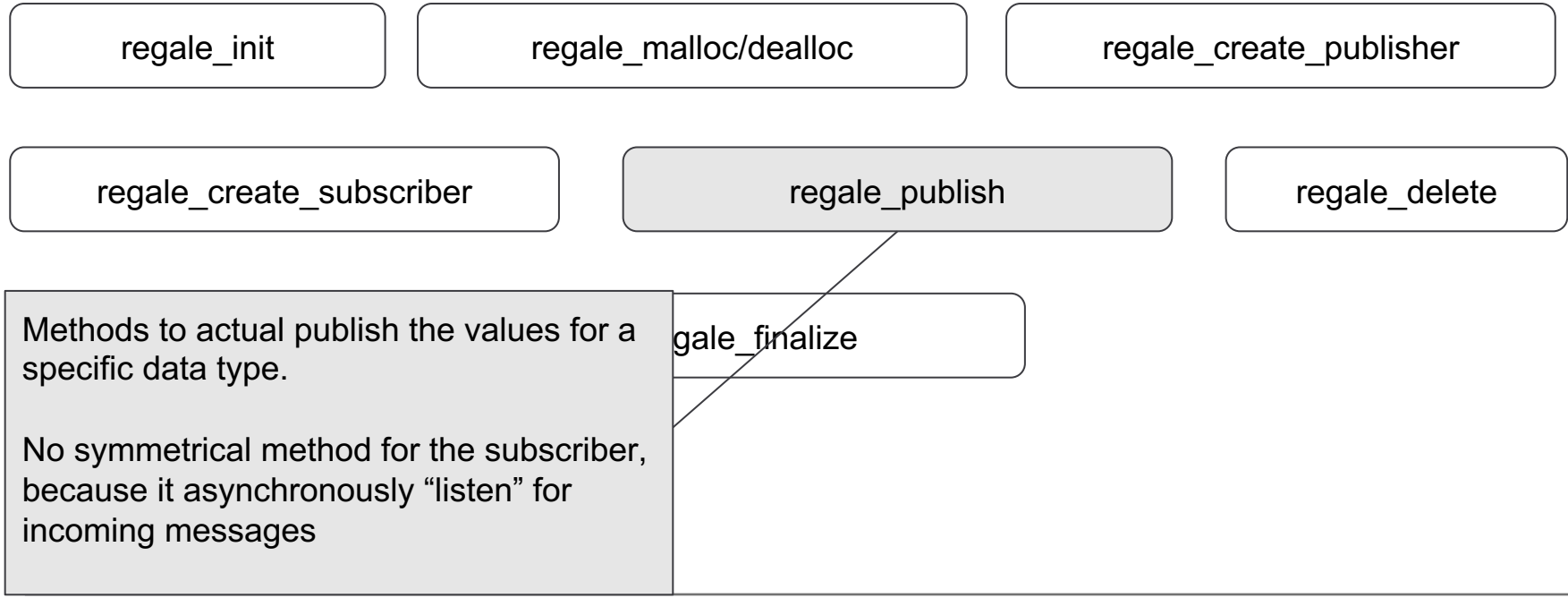
REGALE Library - CORE



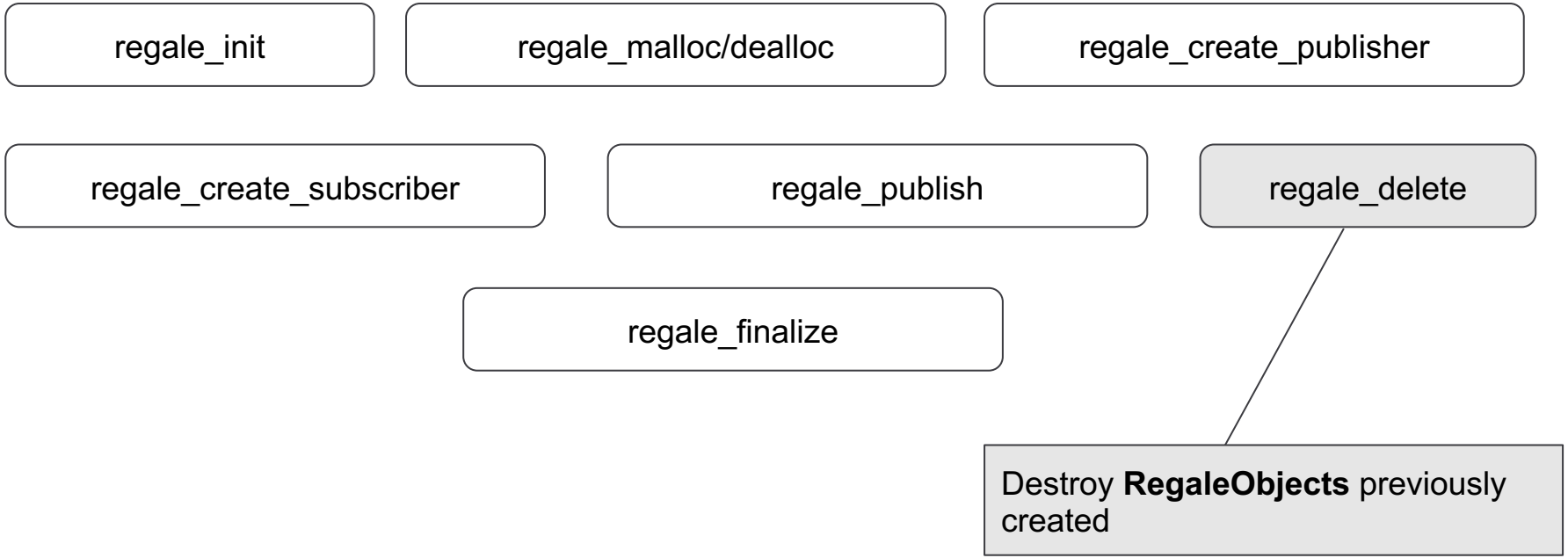
REGALE Library - CORE



REGALE Library - CORE



REGALE Library - CORE



REGALE

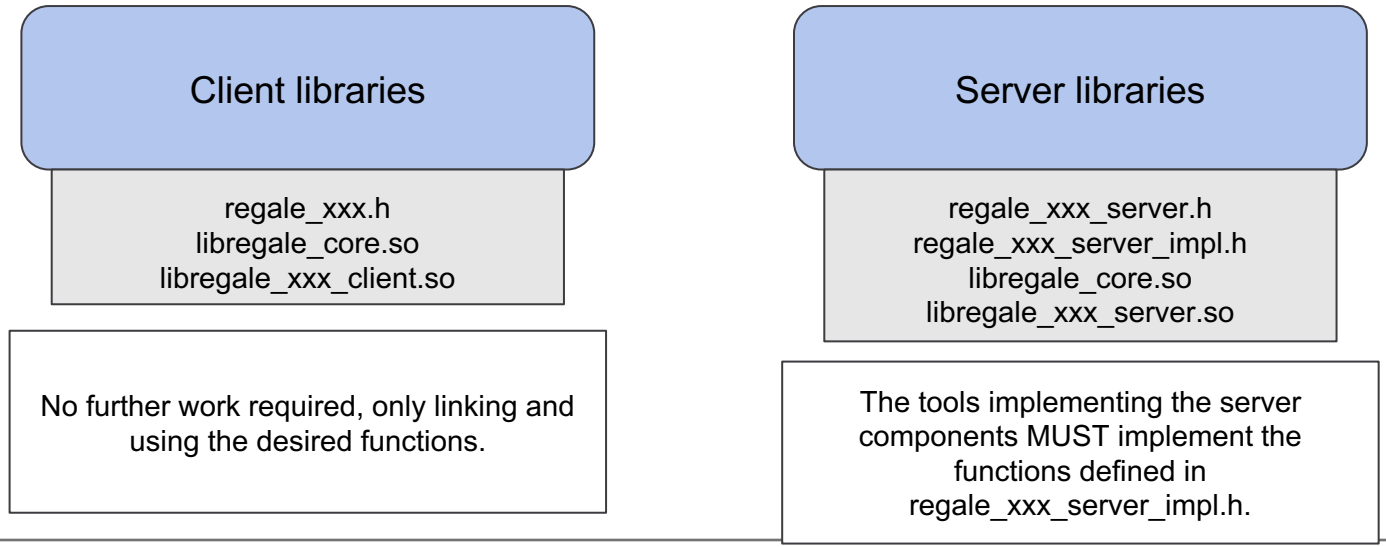
REGALE Library

- Objective & structure
- REGALE Core
- **REGALE Client/Server**
- REGALE Agents

REGALE Library Client/Server

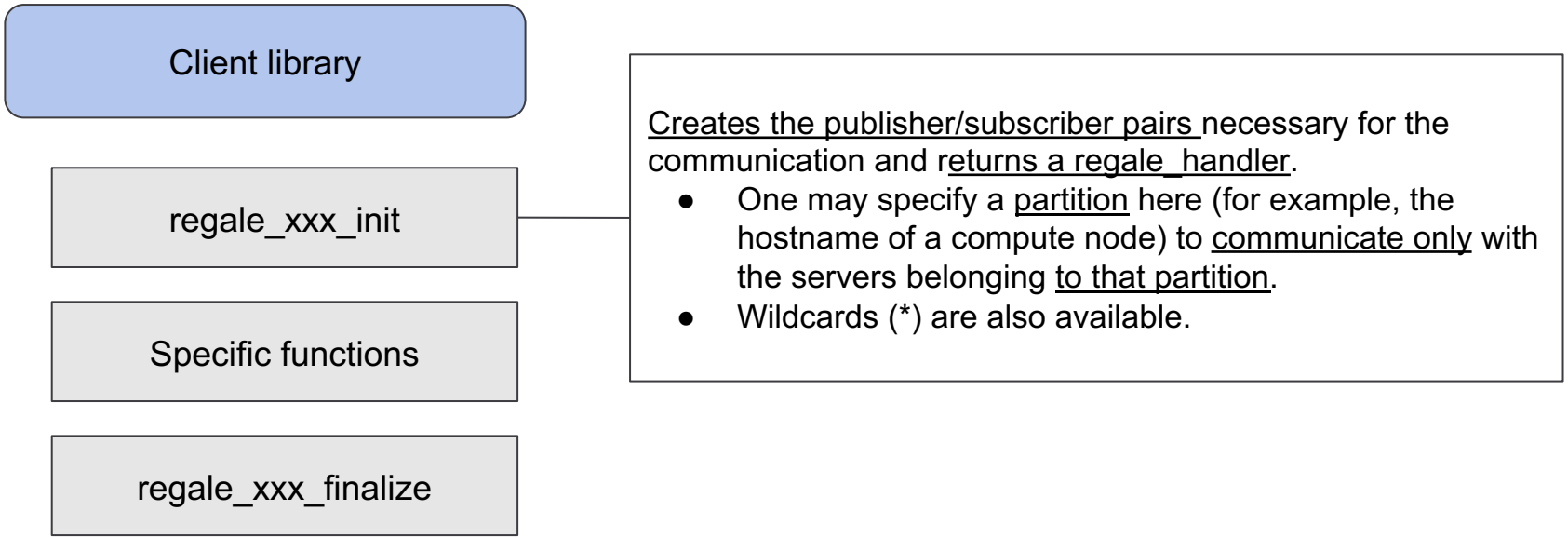
Each REGALE agent has a server and a client library

- Internally uses regale_core.so
- To be compatible with the REGALE library, it only needs to include the relevant parts (e.g. the Monitor server for a monitor)
  - The tool which creates the server MUST implement the functions defined by the spec
  - The tool that uses the client only needs to call the API functions



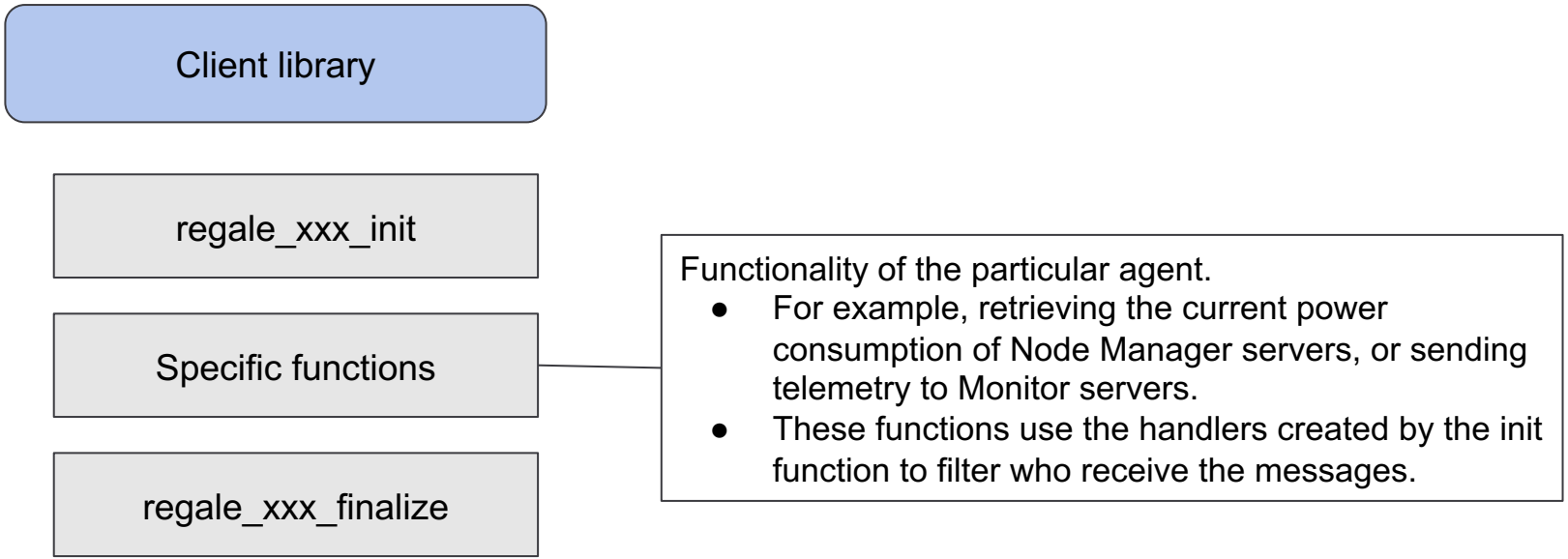
REGALE Library Client/Server

Client Library main components



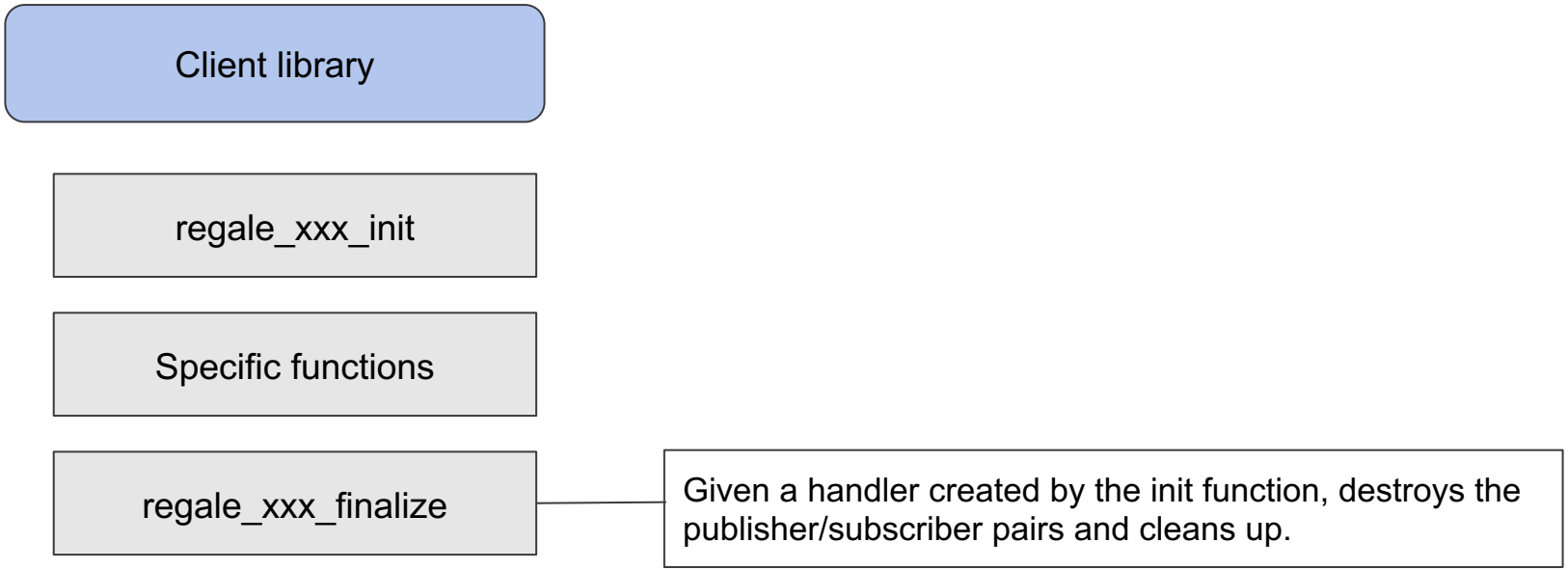
REGALE Library Client/Server

Client Library main components



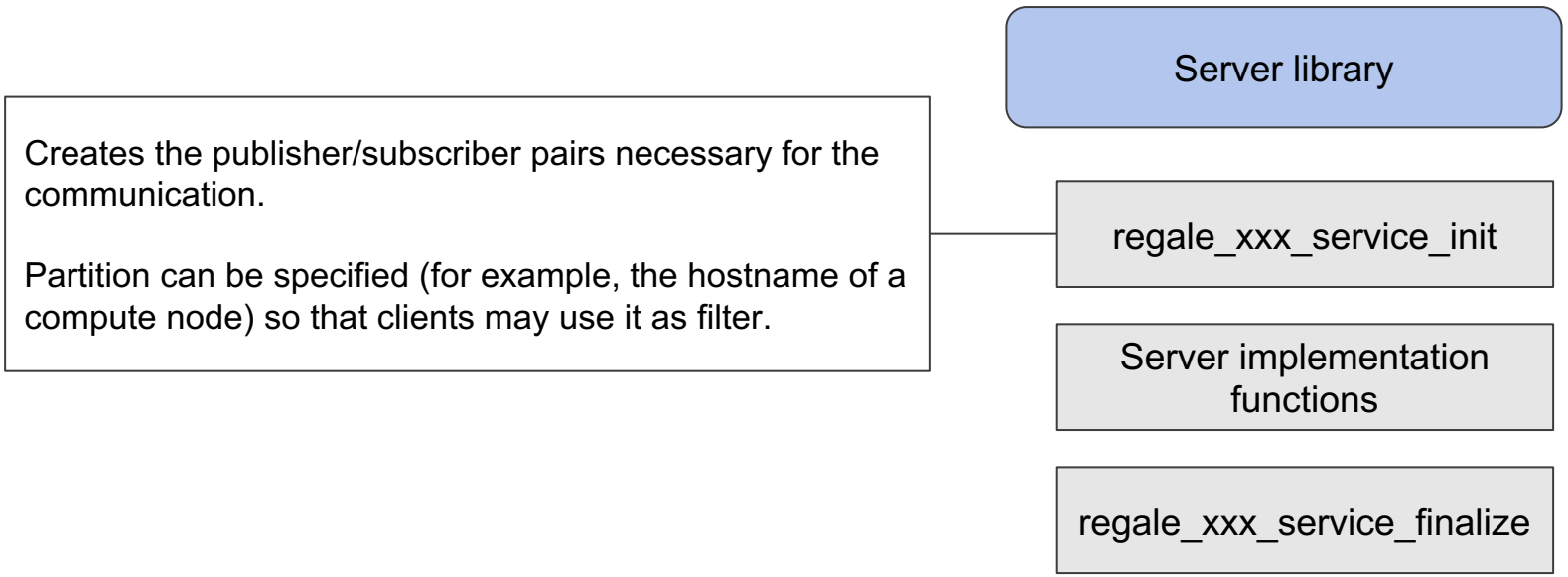
REGALE Library Client/Server

Client Library main components



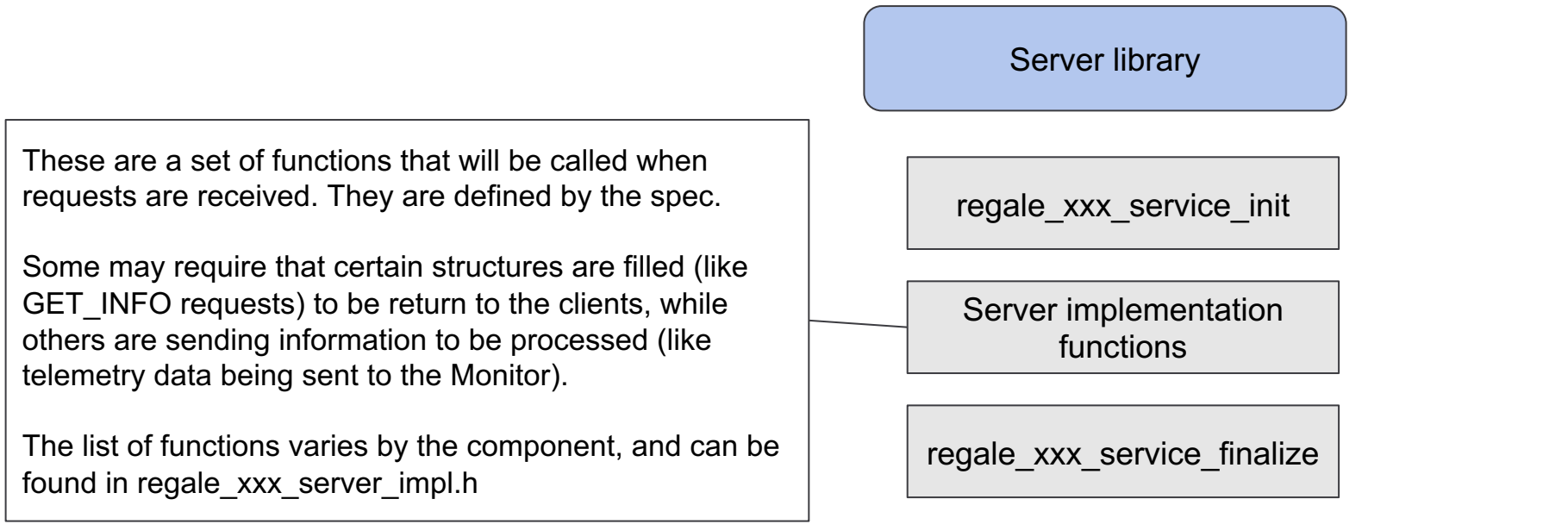
REGALE Library Client/Server

Server Library main components



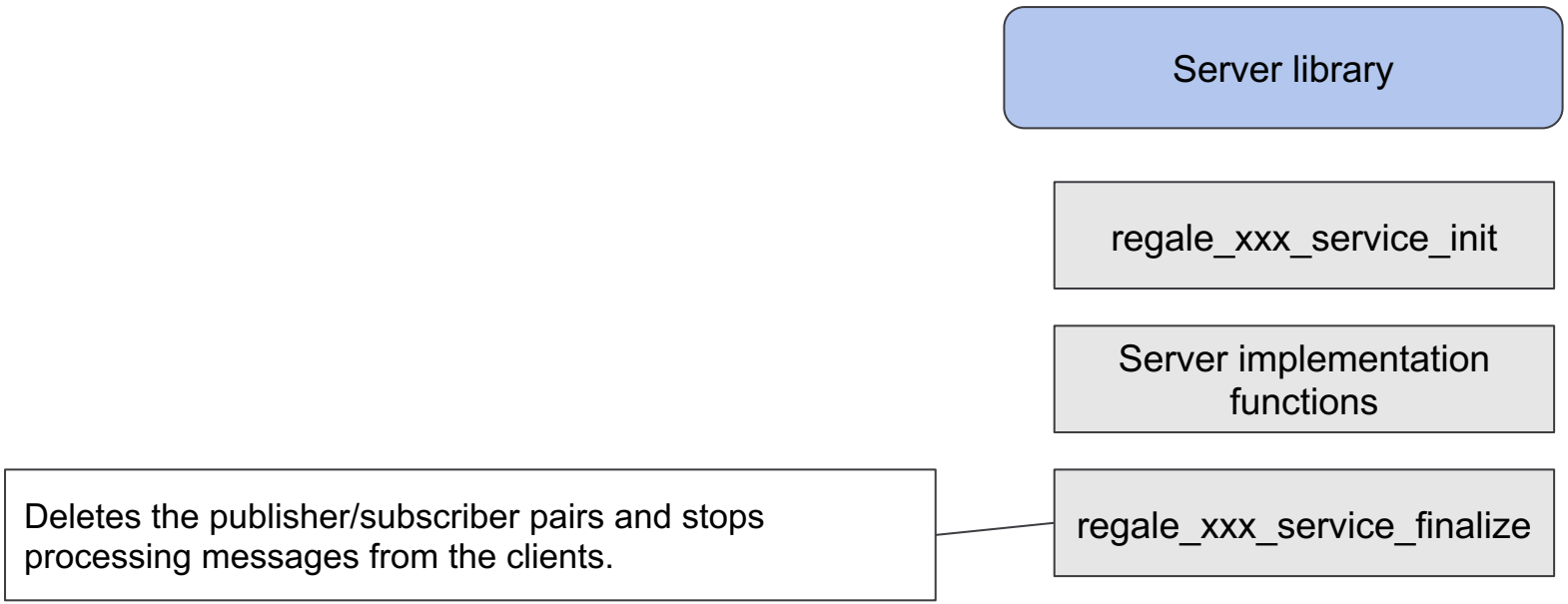
REGALE Library Client/Server

Server Library main components



REGALE Library Client/Server

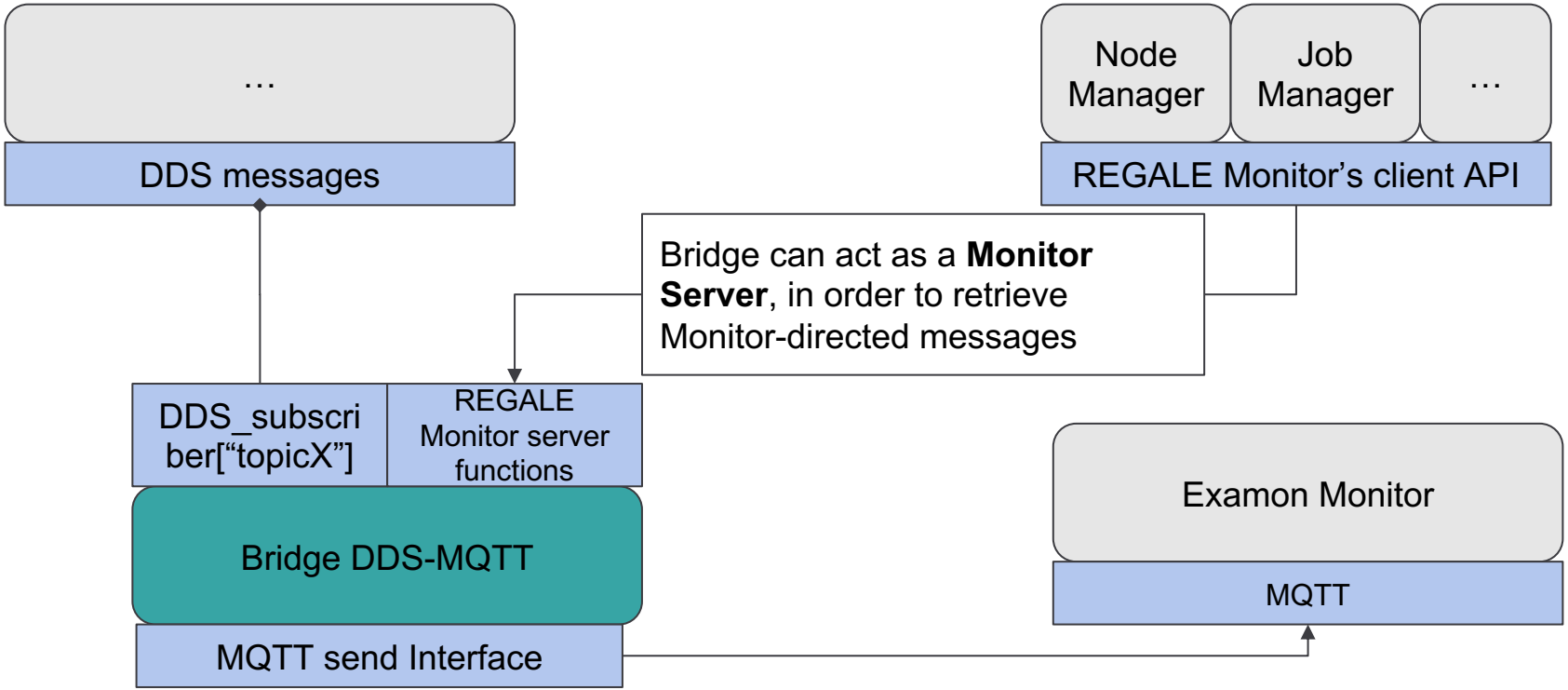
Server Library main components



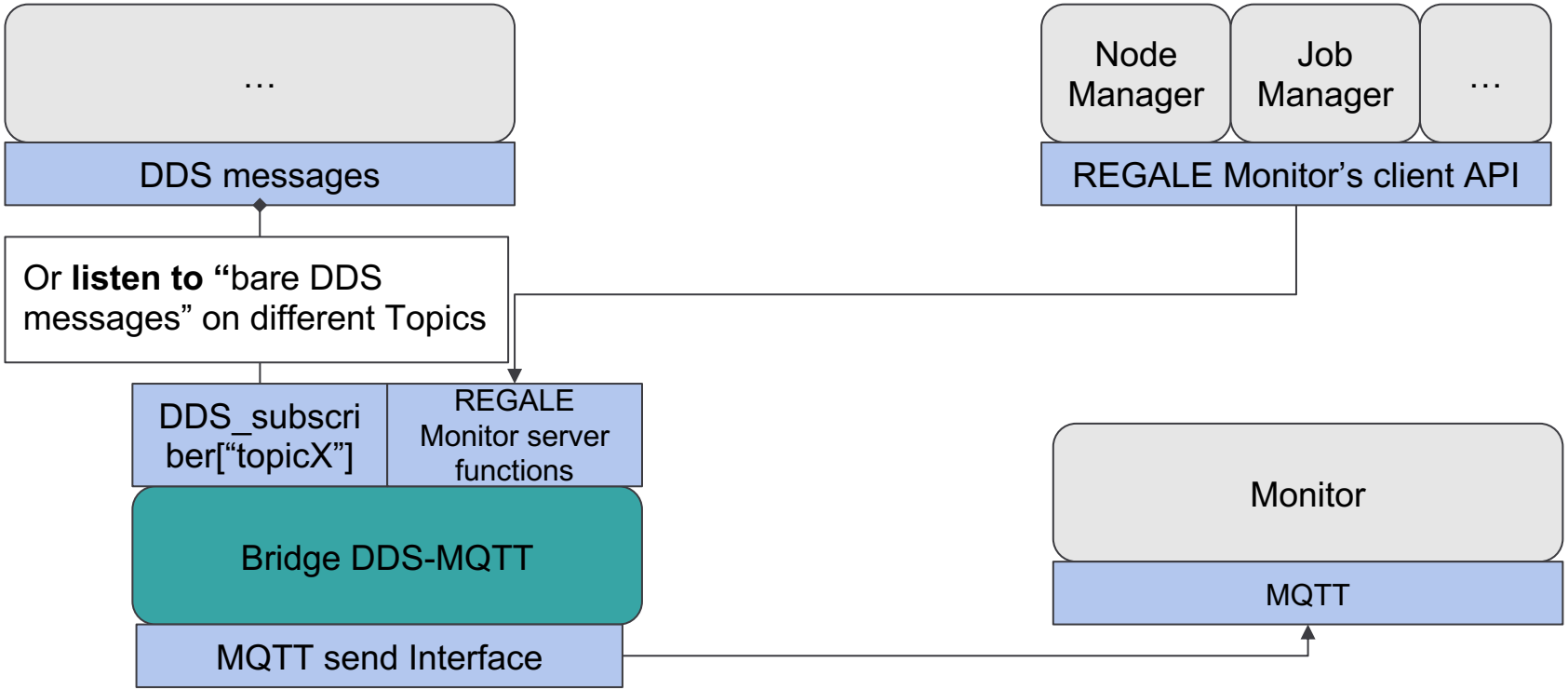
REGALE Library

- Objective & structure
- REGALE Core
- REGALE Client/Server
- **REGALE Agents**

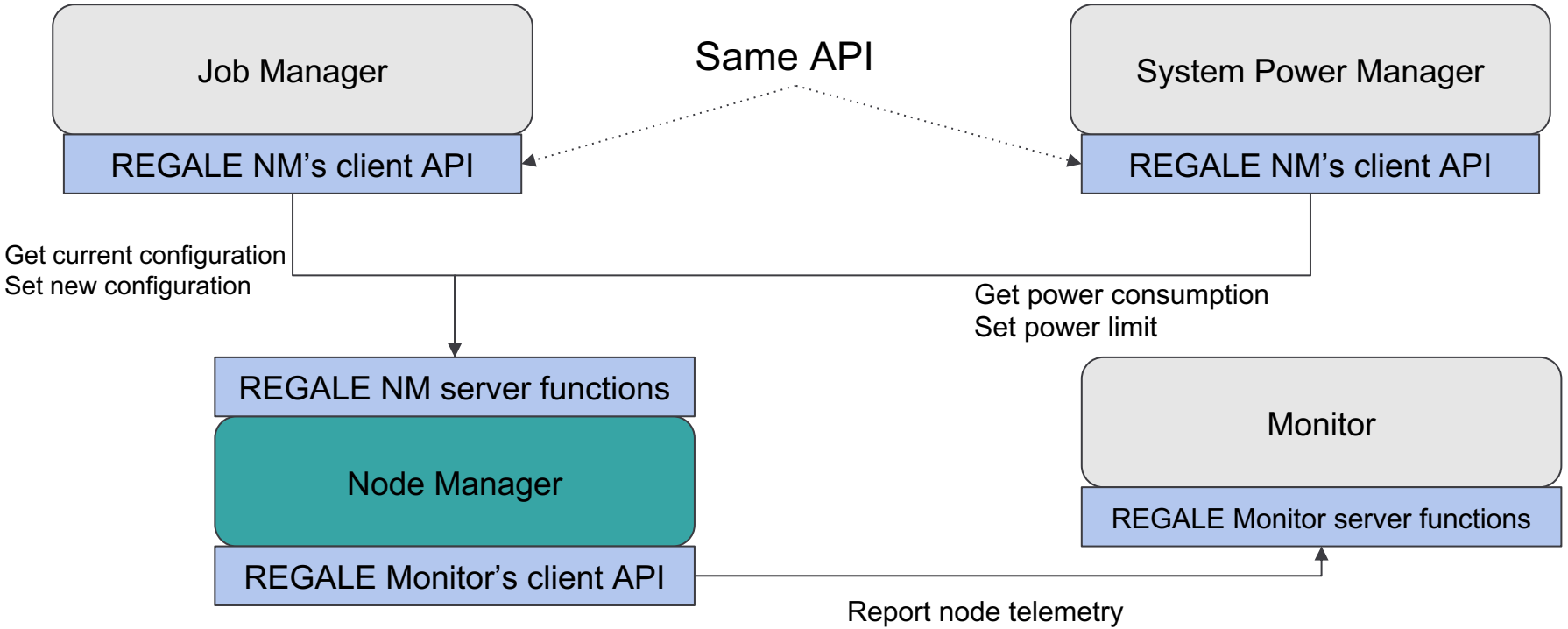
REGALE Agent - Monitor



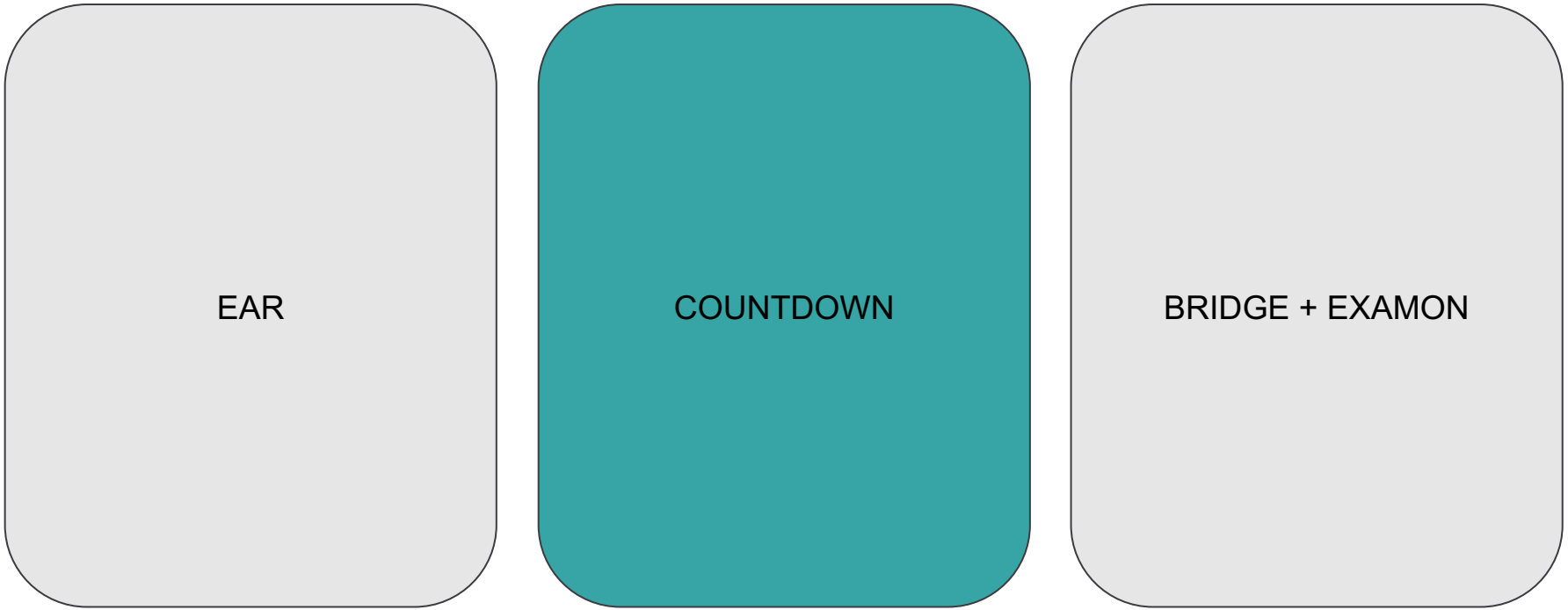
REGALE Agent - Monitor



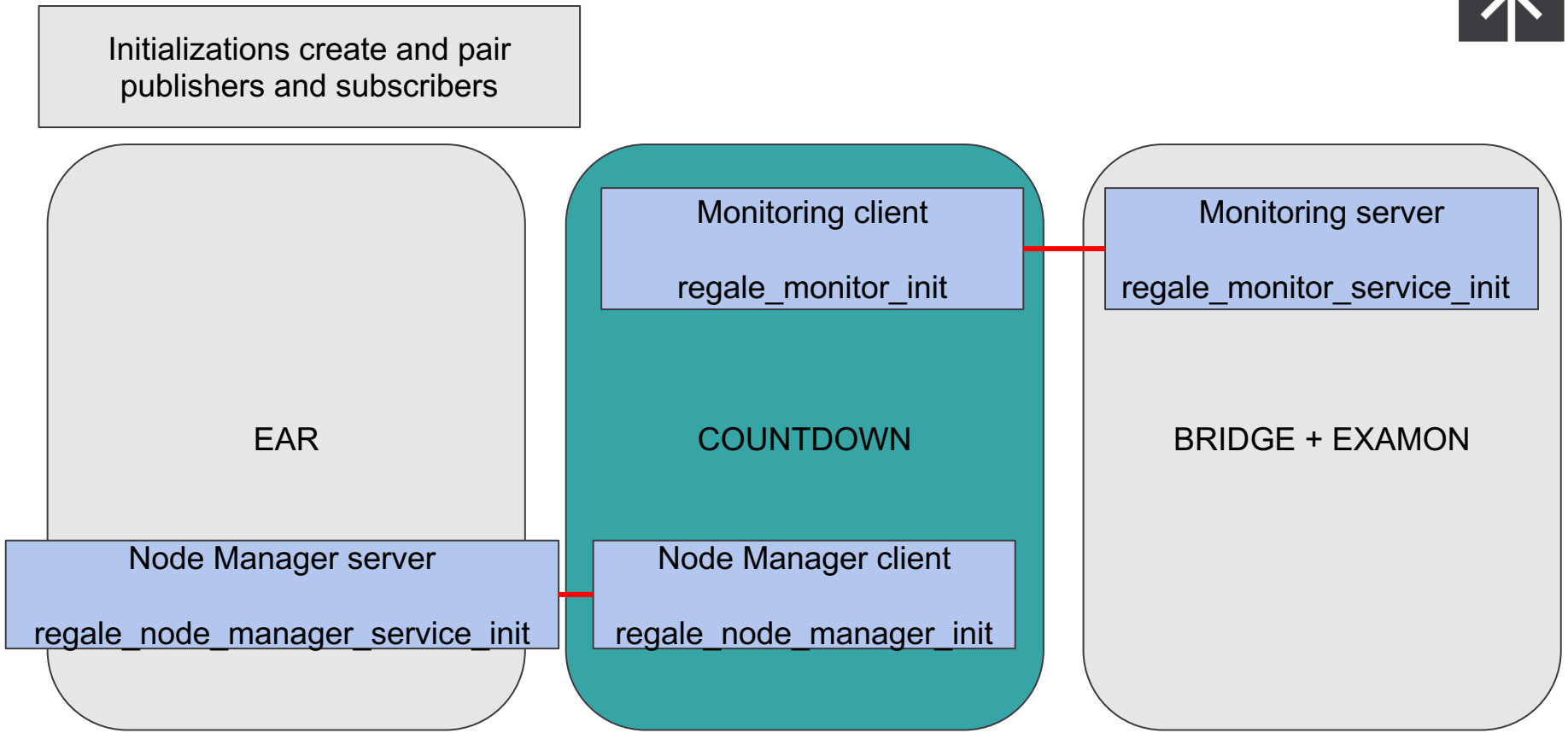
REGALE Agent - Node Manager



REGALE Library - Job Manager - Node Manager - Monitor



REGALE Library - Job Manager - Node Manager - Monitor



REGALE Library - Job Manager - Node Manager - Monitor

Initializations create and pair publishers and subscribers

Client APIs are called, to meet the REGALE's standard to exchange messages

EAR

Node Manager server

COUNTDOWN

Monitoring client

regale_report_job_telemetry

Node Manager client

regale_nm_get_info

BRIDGE + EXAMON

Monitoring server

REGALE Library - Job Manager - Node Manager - Monitor

Initializations create and pair publishers and subscribers

Client APIs are called, to meet the REGALE's standard to exchange messages

EAR

COUNTDOWN

BRIDGE + EXAMON

Monitoring client

Monitoring server

regale_report _job_telemetry_server

Node Manager server

regale_nm_get_info_server

Node Manager client

Server side must define some "interfaces", to specifically decide what the callback will do, with the received information. Again, this respect the REGALE's standard

Conclusions

Large scale computing infrastructure sustainability is of primary importance for today society. Power management is a key ingredient for it.

The REGALE Project has embraced the complexity and fragmentation of power management in large-scale HPC system installation following the HPC PowerStack early results.

The REGALE project during its implementation has studied, conceptualized and implemented an holistic power management view integrating the different power management components features.

The REGALE library is open, modular, extensible and scalable and aims to provide the substrate for power management agents interoperability. Currently a working prototype has been built and integrated with relevant power management software components. We are currently validating the results.

Funding

This project has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 956560. The JU receives support from the European Union's Horizon 2020 research and innovation programme and Greece, Germany, France, Spain, Austria, Italy.