DDS
Dynamic Deployment System

Andrey Lebedev
Anar Manafov
GSI, Darmstadt, Germany
2016-07-07

Motivation

Create a system, which is able to spawn and control

hundreds of thousands of different user tasks

which are tied together by a topology,

can be run on different resource management systems

and can be controlled by external tools.

The Dynamic Deployment System

is a tool-set that automates and significantly simplifies a deployment of user defined processes and their dependencies on any resource management system using a given topology

Basic concepts

DDS:

- implements a single-responsibility-principle command line tool-set and APIs,
- treats users' tasks as black boxes,
- doesn't depend on RMS (provides deployment via SSH, when no RMS is present),
- supports workers behind FireWalls (outgoing connection from WNs required),
- doesn't require pre-installation on WNs,
- deploys private facilities on demand with isolated sandboxes,
- provides a key-value properties propagation service for tasks,
- provides a rules based execution of tasks.

The contract

The system takes so called “topology file” as the input.
Users describe desired tasks and their dependencies using this file.
Users are also provided with a Web GUI to create topologies.

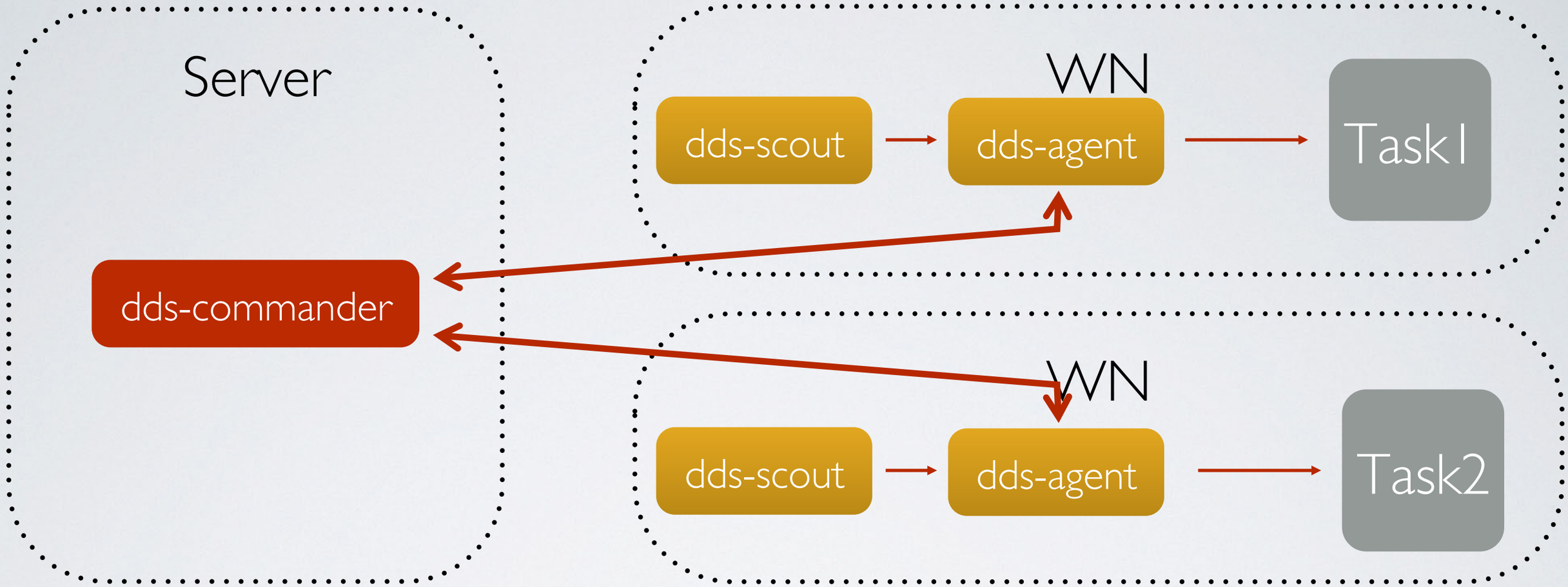
```
<topology id="myTopology">

  <decltask id="task1">
      <exe reachable="false">/Users/andrey/Test1.sh -l</exe>
  </decltask>

  <decltask id="task2">
      <exe>/Users/andrey/DDS/Test2.sh</exe>
  </decltask>

  <main id="main">
      <task>task1</task>
      <task>task2</task>
  </main>

</topology>
```

Declaration of user tasks. Commands with command line argument are supported.

Main block defines which tasks has to be deployed to RMS.

More info: http://dds.gsi.de/doc/nightly/topology.html

DDS Workflow



dds-server *start*

dds-submit -r *ssh* -c *ssh_hosts.cfg*

dds-topology --set *topology_test.xml*

dds-topology --*activate*

DDS SSH plugin cfg file

ssh_hosts.cfg

@bash_begin@
@bash_end@

flp, lxi0234.gsi.de, , /tmp/dds_wrk, 8
epn, lxi235.gsi.de, , /tmp/dds_wrk, 610

Highlights of the DDS features

1. key-value propagation,
2. custom commands for user tasks and ext. utils,
3. RMS plug-ins,
4. Watchdogging

… many more other features

more details here: https://github.com/FairRootGroup/DDS/blob/master/ReleaseNotes.md

key-value propagation

2 tasks → static configuration with shell script
100k tasks → **dynamic configuration with DDS**

Allows user tasks to exchange and synchronize the information dynamically at runtime.

Use case:
In order to fire up the FairMQ devices they have to exchange their connection strings.

- *DDS protocol is highly optimized for massive key-value transport. Internally small key-value messages are accumulated and transported as a single message.*
- *DDS agents use shared memory for local caching of key-value properties.*

key-value in the topology file

```
<topology id="myTopology">
  <property id="property_1" />
  <property id="property_2" />

  <decltask id="task1">
      <exe reachable="false">/Users/andrey/Test1.sh –l</exe>
      <properties>
          <id access=”read">property_1</id>
          <id access="write">property_2</id>
      </properties>
  </decltask>

  <decltask id="task2">
      <exe>/Users/andrey/DDS/Test2.sh</exe>
      <properties>
          <id access=”write">property_1</id>
          <id access=”readwrite">property_2</id>
      </properties>
  </decltask>
  …
</topology>
```

Property declaration with a key.

Task defines a list of dependent properties with access modifier: “read”, “write” or “readwrite”.

When property is received user task will be notified about key-value update.

key-value API

dds-intercom-lib and header file "dds_intercom.h" with user API

```
#include "dds_intercom.h"

CKeyValue ddsKeyValue;

// Subscribe key-value updates
ddsKeyValue.subscribe([](const string& _key, const string& _value) {
    // User code
});

// Subscribe on error messages from DDS commander server
ddsKeyValue.subscribeError([](const string& _msg){
   // Handle error message here
});

// Get list of properties
CKeyValue::valuesMap_t values;
ddsKeyValue.getValues("property_1", &values);

// Write property
ddsKeyValue.put("property_1", "property_1_value");
```

For more information refer to Tutorial1 of DDS.

key-value performance stats

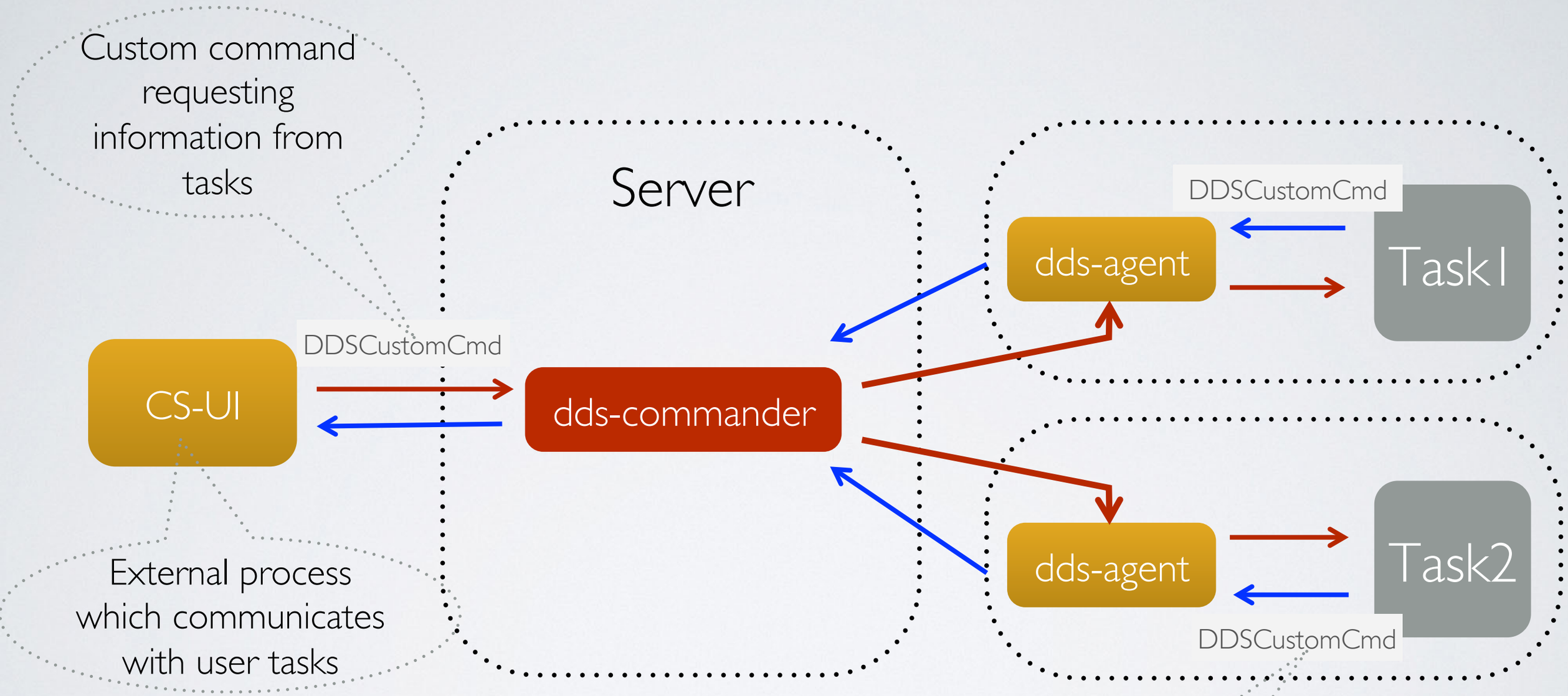
Tested on kronos @ GSI

- **10081 devices** (5040 FLP + 5040 EPN + 1 Sampler);
- Startup time 207 sec (3:27);
- DDS propagated **~77 Millions** key-value properties.

Device – in this context is a user executable. FLP, EPN and Sampler are concrete device types for the Alice O2 framework.

Custom commands (1)

A possible use case: Control System for Experiment which is able to com
municate with user tasks




```
// Subscribe on custom commands
ddsCustomCmd.subscribeCmd(...);

// Send custom command
ddsCustomCmd.sendCmd(...);
```

Reply with requested information

Custom commands (2)

Sending of custom commands from user tasks and ext. utilities.

Two use cases:

1. User task which connects to DDS agent
2. Ext. utility which connects to DDS commander

A custom command is a standard part of the DDS protocol.
From the user perspective a command can be any text, for example, JSON or XML.
A custom command recipient is defined by a condition.

Broadcast custom command
to all tasks with this path.

Task index.

Condition types:

1. Internal channel ID which is the same as sender ID.
2. Path in the topology: main/RecoGroup/TrackingTask.
3. Hash path in the topology: main/RecoGroup/TrackingTask_23.

Custom commands (3)

dds-intercom-lib and header file “dds_intercom.h” with user API

```
#include “dds_intercom.h”

CCustomCmd ddsCustomCmd;

// Subscribe on custom commands
ddsCustomCmd.subscribeCmd(
    [](const string& _command, const string& _condition, uint64_t _senderId)
{
    cout << ”Command: " << _command << " condition: " << _condition
         << " senderId: " << _senderId << endl;

    // Send message back to sender
    if (_command == "please-reply”)
        ddsCustomCmd.sendCmd("reply“, to_string(_senderId));
});

// Subscribe on reply from DDS commander server
ddsCustomCmd.subscribeReply([](const string& _msg)
{
    cout << ”Message: " << _msg << endl;
});
```

For more information refer to Tutorial2 of DDS.

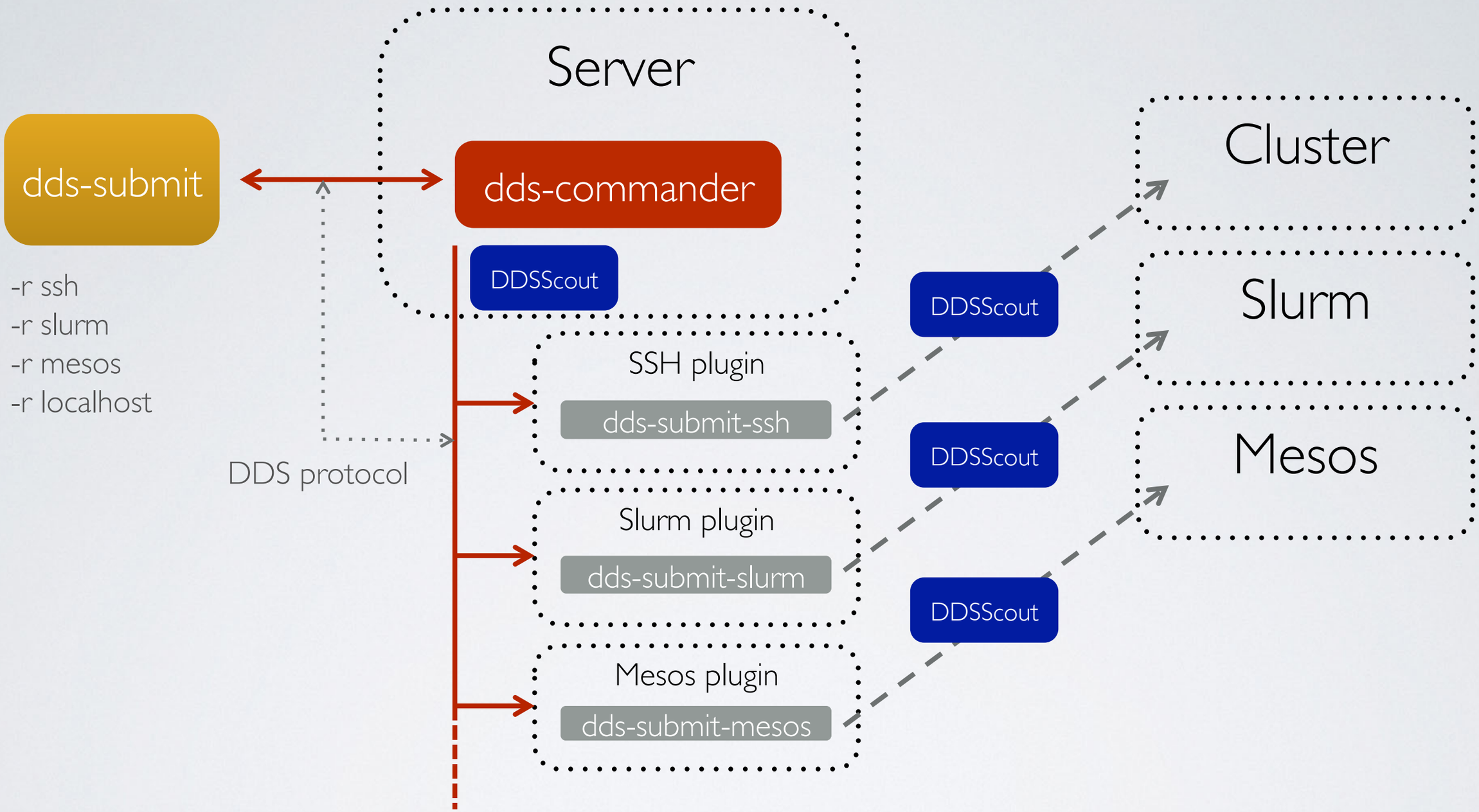
RMS plug-in architecture
Motivation

Give external devs. a possibility to create DDS plug-ins - to cover different RMS.

Isolated and safe execution. A plug-in must be a standalone processes - if segfaults, won't effect DDS.

Use DDS protocol for communication between plug-in and commander server - speak the same language as DDS.

RMS plug-in architecture



1. dds-commander starts a plug-in based on the dds-submit parameter,
2. plug-in contact DDS commander server asking for submissions details,
3. plug-in deploy DDSScout fat script on target machines,
4. plug-in execute DDSScout on target machines.

List of available RMS plug-ins

#1: **SSH**

#2: **localhost**

#3: **Slurm**

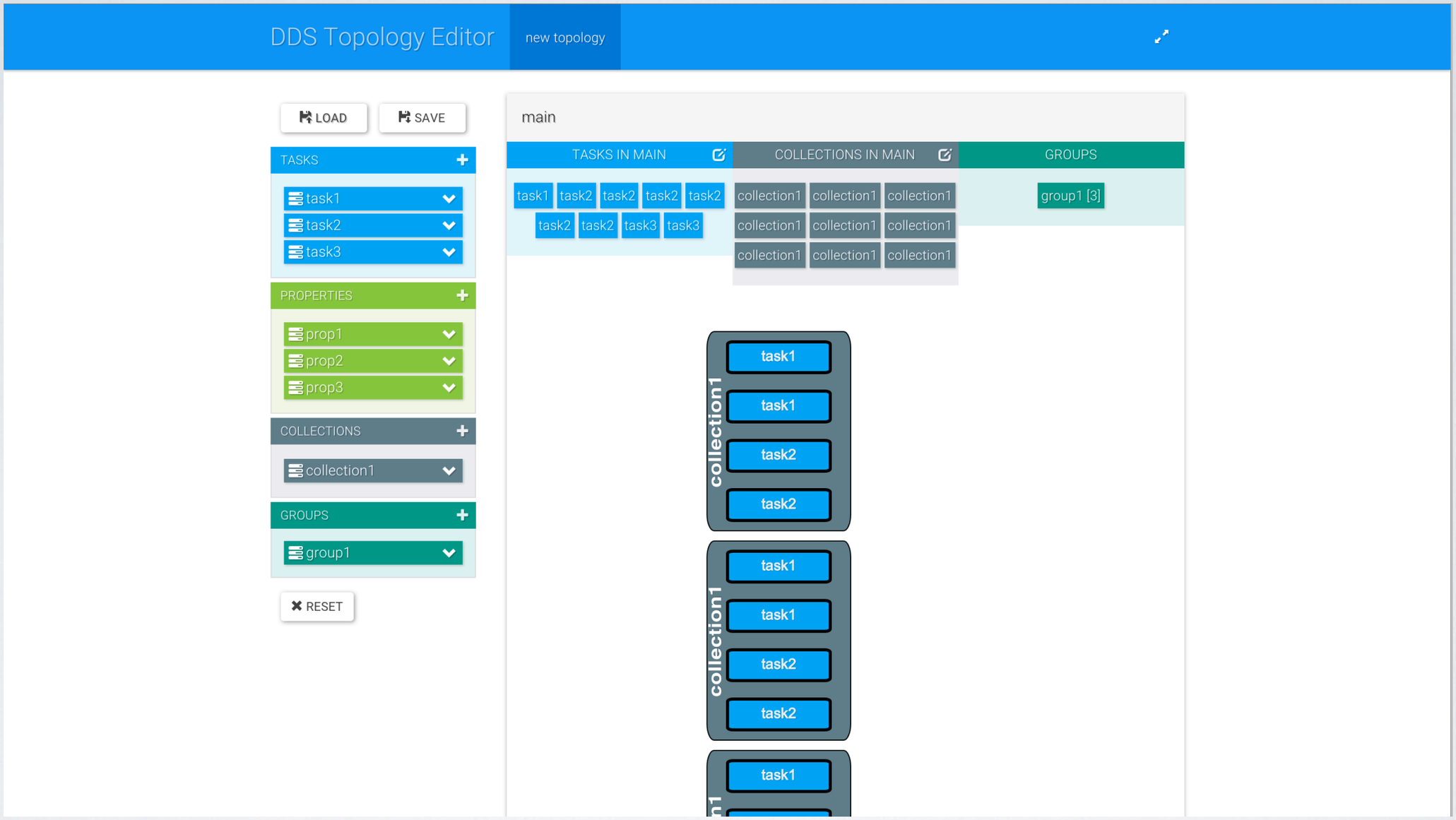
#4: **MESOS** (work of Giulio Eulisse and Kevin Napoli from CERN)

Documentation and tutorials

- User manual
- API documentation
- Tutorial1: key-value propagation
- Tutorial2: custom commands

For more information refer to DDS documentation:
http://dds.gsi.de/documentation.html

Topology editor



http://rbx.github.io/DDS-topology-editor/

By Alexey Rybalchenko (GSI, Darmstadt)

- Releases - DDS v1.2

  (http://dds.gsi.de/download.html),

- DDS Home site: http://dds.gsi.de

- User's Manual: http://dds.gsi.de/documentation.html

- Continues integration:
  http://demac012.gsi.de:22001/waterfall

- Source Code:
  https://github.com/FairRootGroup/DDS
  https://github.com/FairRootGroup/DDS-user-manual
  https://github.com/FairRootGroup/DDS-web-site
  https://github.com/FairRootGroup/DDS-topology-editor

BACKUP

Elements of the topology

Task

- A task is a single entity of the system.
- A task can be an executable or a script.
- A task is defined by a user with a set of props and rules.
- Each task will have a dedicated DDS watchdog process.

####Collection

- A set of tasks that have to be executed on the same physical computing node.

####Group

- A container for tasks and collections.
- Only main group can contain other groups.
- Only group define multiplication factor for all its daughter elements.

```
CRMSPluginProtocol prot("plugin-id");
```

(1)

```
prot.onSubmit([](const SSubmit& _submit) {
        // Implement submit related functionality here.

        // After submit has completed call stop() function.
        prot.stop();
});
```

(2)

```
// Let DDS commander know that we are online and start listen for messages.
prot.start(bool _block = true);
```

(3)

```
// Report error to DDS commander
proto.sendMessage(dds::EMsgSeverity::error, “error message here”);

// or send an info message
proto.sendMessage(dds::EMsgSeverity::info, “info message here”);
```

RMS plug-in architecture

```
$ dds-submit -r localhost -n 10


dds-submit: Contacting DDS commander on pb-d-128-141-130-162.cern.ch:20001 ...
dds-submit: Connection established.
dds-submit: Requesting server to process job submission...
dds-submit: Server reports: Creating new worker package...
dds-submit: Server reports: RMS plug-in: /Users/anar/DDS/1.1.52.gfb2d346/plugins/dds-submit-
localhost/dds-submit-localhost
dds-submit: Server reports: Initializing RMS plug-in...
dds-submit: Server reports: RMS plug-in is online. Startup time: 17ms.
dds-submit: Server reports: Plug-in: Will use the local host to deploy 10 agents
dds-submit: Server reports: Plug-in: Using '/var/folders/ng/vl4ktqmx3y93fq9kmtwktpb40000gn/
T/dds_2016-03-31-15-33-32-090' to spawn agents
dds-submit: Server reports: Plug-in: Starting DDSScout in '/var/folders/ng/
vl4ktqmx3y93fq9kmtwktpb40000gn/T/dds_2016-03-31-15-33-32-090/wn'
dds-submit: Server reports: Plug-in: DDS agents have been submitted
dds-submit: Server reports: Plug-in: Checking status of agents...
dds-submit: Server reports: Plug-in: All agents have been started successfully
```

Two ways to activate a topology

```
dds-submit -r RMS -n 100
dds-topology --set <topology_file #1>
dds-topology --activate
dds-topology --stop
dds-topology --set <topology_file #2>
dds-topology --activate
```

(1)

Reserve resources first, then deploy different topologies on it.

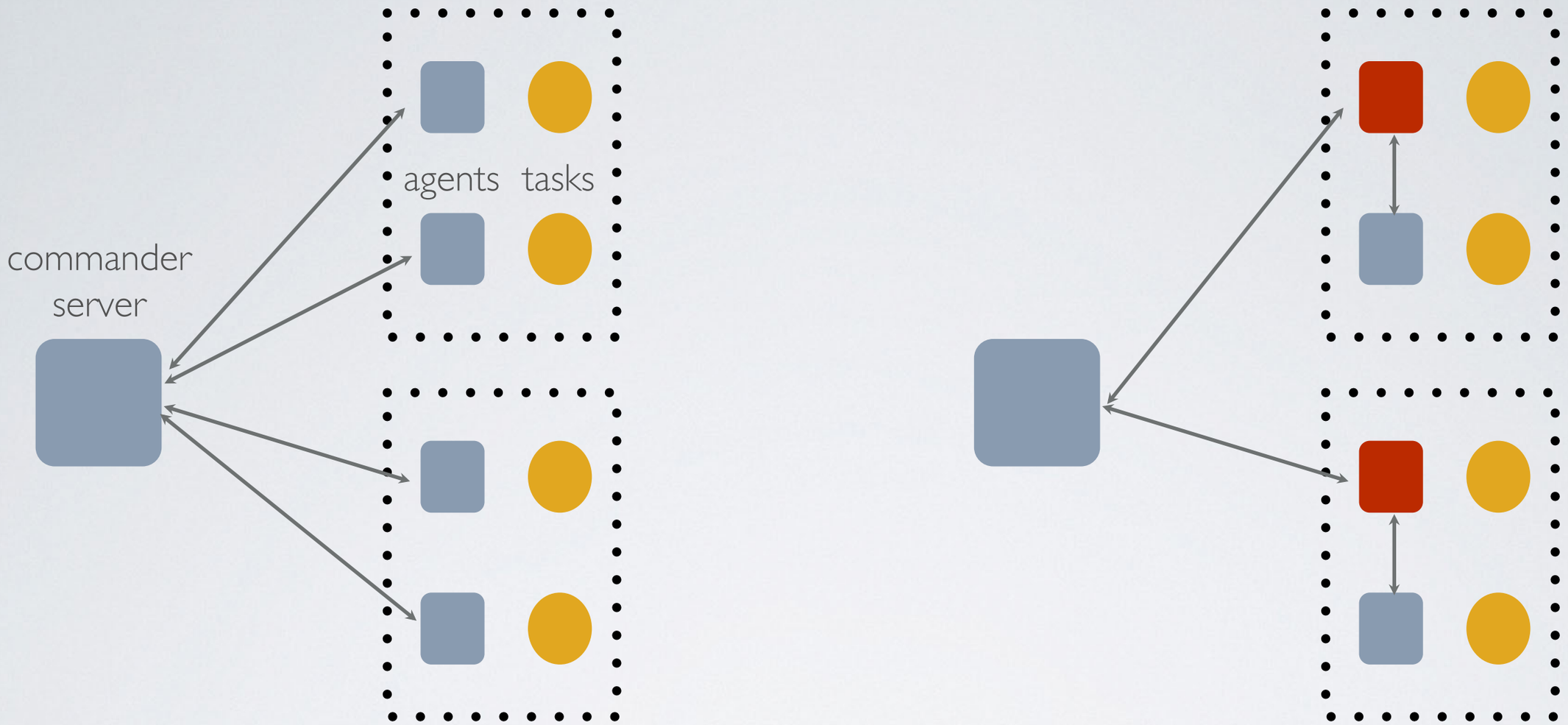
```
dds-submit -r RMS --topo <topology_file>
```

(2)

Reserve resources according to requirements of a given topology.

We aim to delegate the complex work of requirements analysis and corresponding resource allocation to RMS.

Lobby based deployment



1. DDS Commander will have one connection per host (lobby),
2. lobby host agents (master agents) will act as dummy proxy services, no special logic will be put on them except key-value propagation inside collections,
3. key-value will be either global or local for a collection