DDS
Dynamic Deployment System

Andrey Lebedev
Anar Manafov
GSI, Darmstadt, Germany
2017-07-06

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.

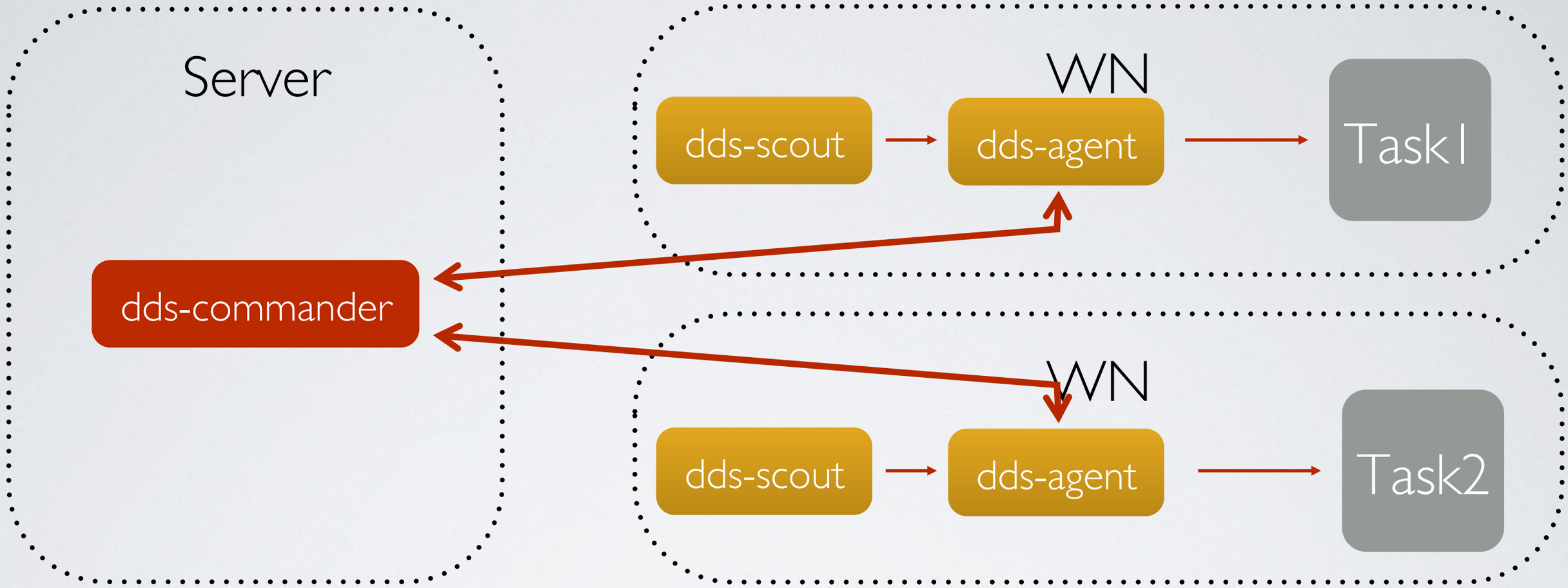
```
<topology id="myTopology">

  <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 *–activate topology.xml*

dds-topology *–update new_topology.xml*

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 message queue to deliver key-value to a user task.*

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

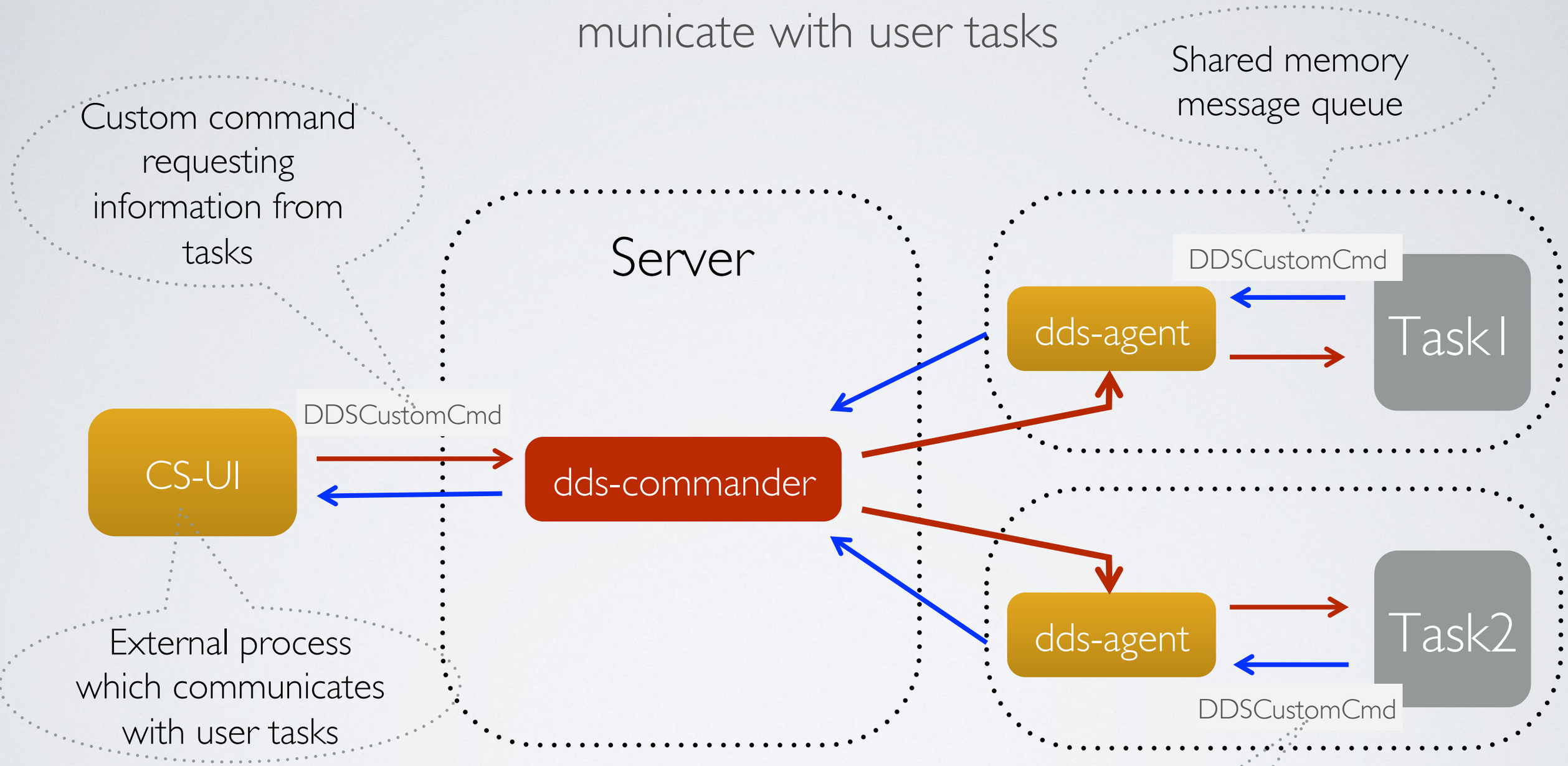
dds-intercom-lib and header "dds_intercom.h"

```
1  #include "dds_intercom.h"
2
3  CIntercomService service;
4  CKeyValue keyValue(service);
5
6  // Subscribe on error events
7  service.subscribeOnError([](EErrorCode _errorCode, const string& _msg) {
8      // Service error
9  });
10
11 // Subscribe on key update events
12 keyValue.subscribe([](const string& _propertyID, const string& _key, const string& _value) {
13      // Key-value update event received
14 });
15
16 // Subscribe on delete key notifications
17 keyValue.subscribeOnDelete([](const string& _propertyID, const string& _key) {
18      // Key-value delete event received
19 });
20
21 // Start listening to events we have subscribed on
22 service.start();
23
24 keyValue.putValue("prop_1", "prop_1_value");
```

For more information refer to Tutorial1 of DDS.

Custom commands (1)

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




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

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

Custom commands (2)

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

Two use cases:

1. User task communicates with DDS agent via shared memory
2. Ext. utility which connects to DDS commander via network

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 API

dds-intercom-lib and header "dds_intercom.h"

```
#include "dds_intercom.h"

// DDS custom command API
CIntercomService service;
CCustomCmd customCmd(service);

// first subscribe on errors, before doing any other function calls.
service.subscribeOnError([](
  const EErrorCode _errorCode, const string& _errorMsg) {
    // Report error to the user
});

// Subscribe on custom commands
customCmd.subscribe([&customCmd](
  const string& _command, const string& _condition, uint64_t _senderId) {
    string senderIdStr = to_string(_senderId);
    // Reply to the sender
    customCmd.send("Reply message", senderIdStr);
});

// Subscribe on reply from DDS commander server
customCmd.subscribeOnReply([](const string& _msg) {
   // Process reply message from server
});

service.start();
```

For more information refer to Tutorial2 of DDS.

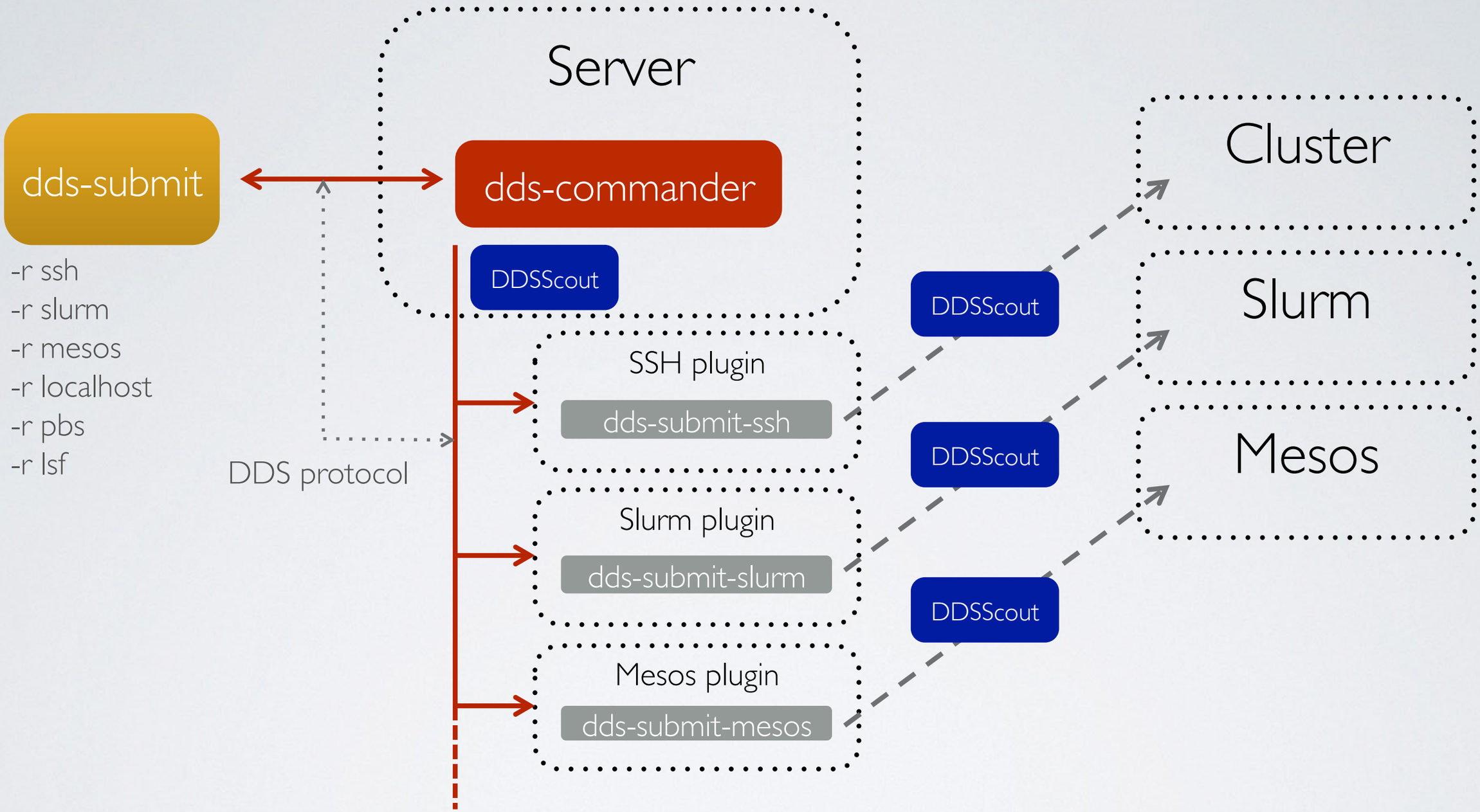
RMS plug-in architecture
Motivation

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** (CERN)

#5: **PBS**

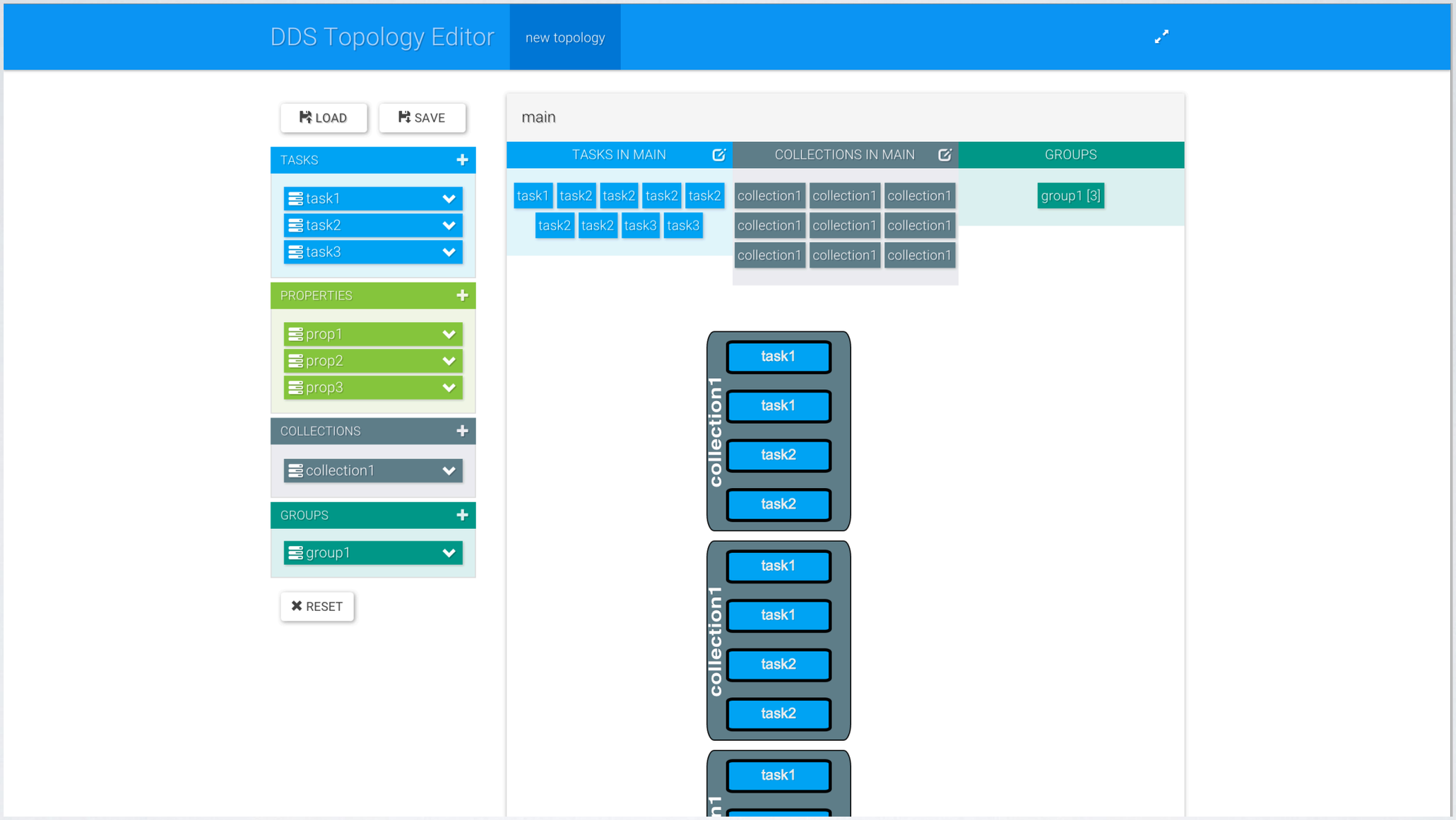
#6: **LSF**

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 v1.6

  (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.

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.

```
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 –activate <topology_file #1>
dds-topology –stop
dds-topology –activate <topology_file #2>
```

(1)

Reserve resources first, then deploy different topologies on it.

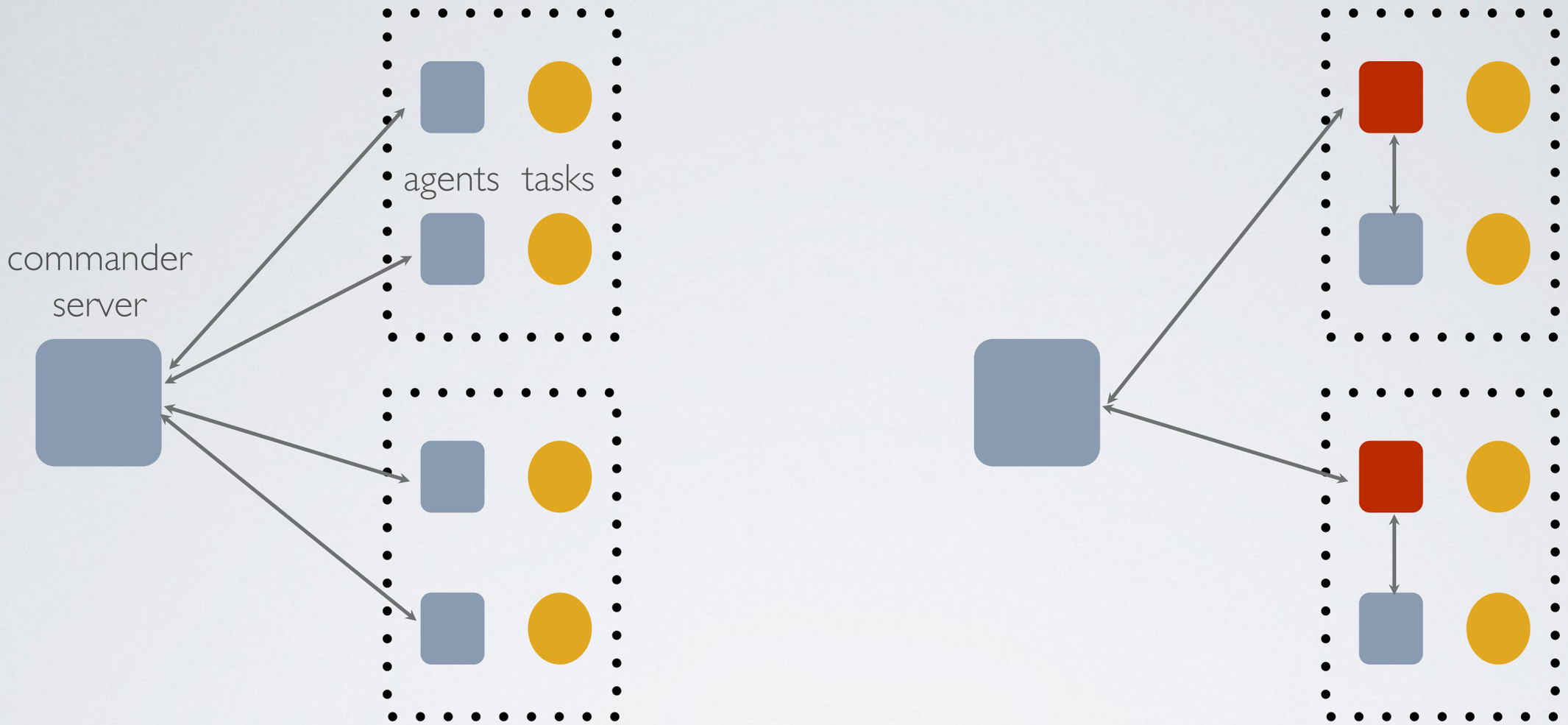
```
dds-submit -r RMS --topo <topology_file>
```

(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

Shared memory communication

Shared memory channel:

- Similar API as DDS network channel;
- Two way communication;
- Asynchronous read and write operations;
- dds-protocol.

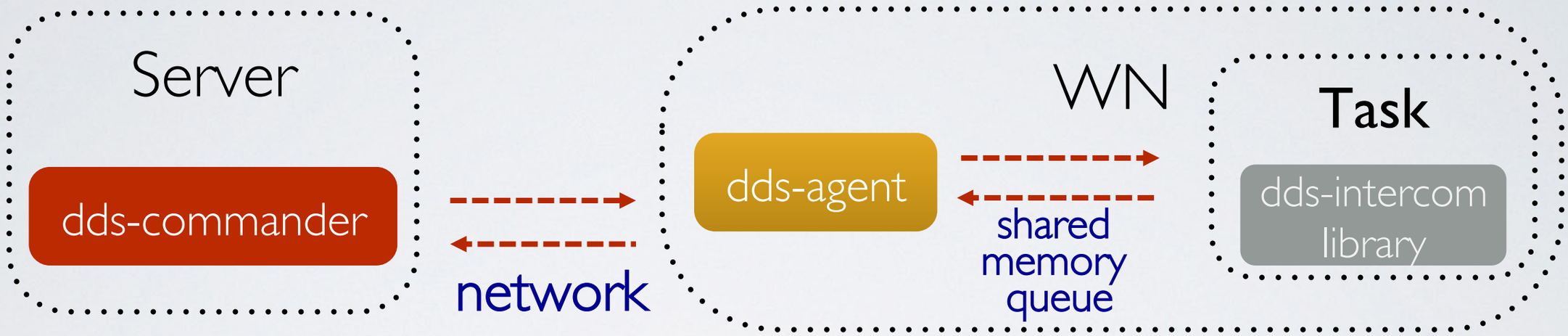
Implementation:

1. **boost::message_queue:** message transport via shared memory;
2. **dds-protocol:** message encoding and decoding;
3. **boost::asio:** proactor design pattern, thread pool.

key-value and custom commands

NEW

Shared memory communication
between dds-agent and user task



No need to cache messages in the dds-intercom-lib – we guarantee that the message will be delivered. Messages are stored directly in the shared memory and managed by the message queue.

2x better performance for our test case:
40 tasks intensively exchanging key-values on a single node with 40 logical cores.

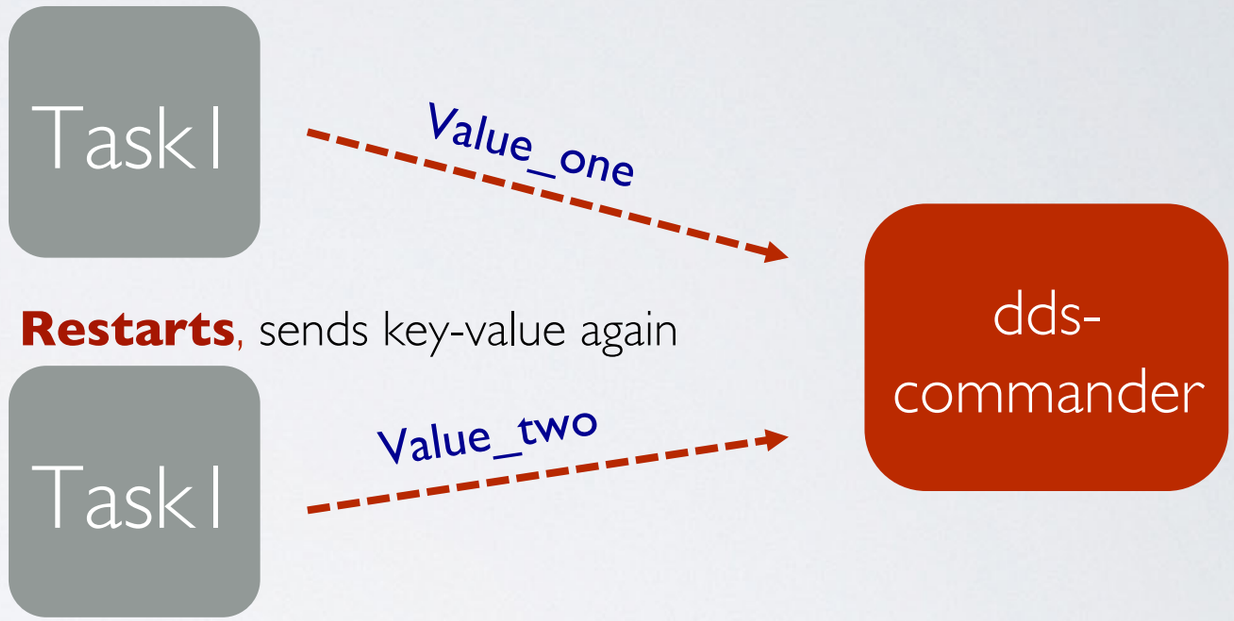
Versioning in key-value propagation [1]

Single property in the topology – multiple keys at runtime.

A certain key can be changed only by one task instance.

Why do we need versioning?

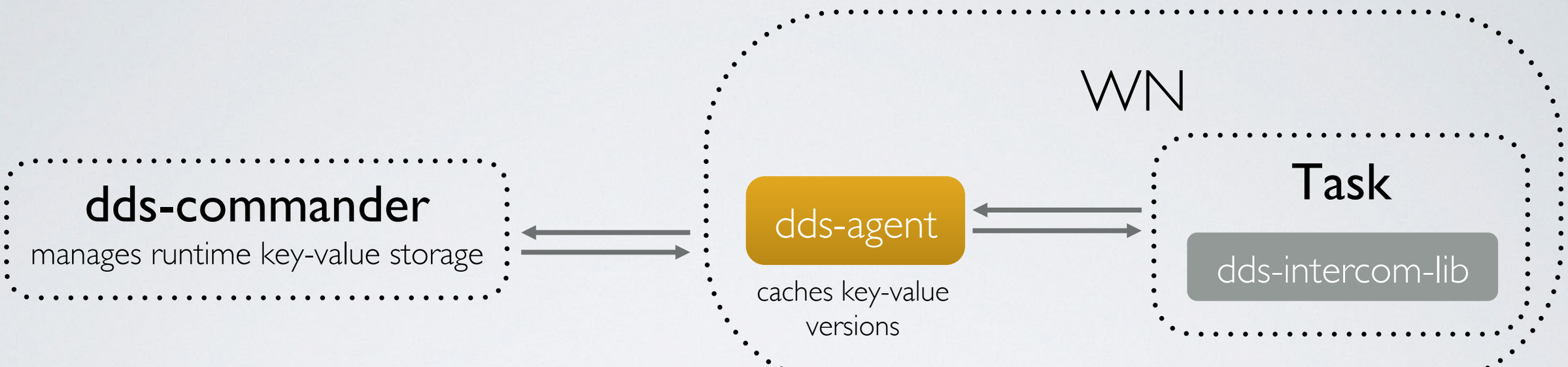
Starts, sends key-value and dies



If Value_two arrives first, than it will be overwritten by Value_one and all tasks will be notified with the wrong value.

Versioning in key-value propagation [2]

Versioning is completely hidden from the user.



1. Task sets key-value using dds-intercom-lib.
2. dds-agent sets version for key-value and sends it to dds-commander.
3. dds-commander checks version:

a) if version is correct than it updates version in storage and broadcasts the update to all related dds-agents;

b) in case of version mismatch it sends back an error containing current key-value version in the storage. dds-agent receives error, updates version cache and forces the key update with the latest value set by the user.

Runtime topology update [1]

Update of the currently running topology without stopping the whole system.

Steps:

1. Get the **difference** between current and new topology. The algorithm calculates hashes for each task and collection in the topology based on the full path and compares them. As a result a list of removed tasks and collections and a list of added tasks and collections are obtained.
2. **Stop** removed tasks and collections.
3. **Schedule and activate** added tasks and collections.

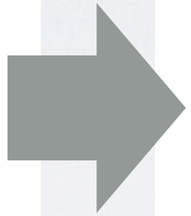
Limitation:

Declaration of tasks and colle ctions can't be changed.

```
<decltask id="task1">
   <exe>/Users/andrey/DDS/task1.sh</exe>
   <properties>
      <id access="”write">property1</id>
   </properties>
</decltask>
```

Runtime topology update [2]

```
<main id="main">
    <task>task1</task>
    <collection>collection1</collection>
        <group id="group1" n="2">
                <task>task1</task>
                <task>task2</task>
                <collection>collection1</collection>
        </group>
        <group id="group2" n="3">
                <task>task1</task>
        </group>
</main>
```



```
<main id="main">
        <task>task3</task>
        <group id="group1" n="3">
                <task>task1</task>
                <task>task2</task>
                <collection>collection1</collection>
        </group>
        <group id="group2" n="1">
                <task>task1</task>
                <task>task3</task>
        </group>
</main>
```

Runtime topology update [3]

dds-topology *--update new_topology.xml*

```
dds-topology: Contacting DDS commander on visitor-16386626.dyndns.cern.ch:20001  ...
dds-topology: Connection established.
dds-topology: Requesting server to update a topology...
dds-topology: Updating topology to /Users/andrey/DDS/1.3.25.gbf0d0fc/../topology_test_diff_2.xml
dds-topology:
Removed tasks:5
1 x main/collection1/task1
1 x main/collection1/task2
2 x main/group2/task1
1 x main/task1
Removed collections:1
1 x main/collection1
Added tasks:6
1 x main/group1/collection1/task1
1 x main/group1/collection1/task2
1 x main/group1/task1
1 x main/group1/task2
1 x main/group2/task3
1 x main/task3
Added collections:1
1 x main/group1/collection1

dds-topology: Stopping removed tasks...
[==================================================] 100 % (5/5)
Stopped tasks: 5
Errors: 0
Total: 5
Time to Stop: 1.003 s
dds-topology: Activating added tasks...
[==================================================] 100 % (6/6)
Activated tasks: 6
Errors: 0
Total: 6
Time to Activate: 0.099 s
```

dds-octopus

Motivation: Growing complexity of DDS requires powerful functional tests. Unit tests can't cover all cases. Most of issues can be only detected during run-time when multiple agents are in use.

dds-octopus: A full blown functional test machinery for DDS.
Test DDS using DDS.

dds-octopus: core components

dds-octopus-start - a steering script. Starts DDS, deploys agents and activates predefined topologies. It validates return values of all used DDS commands and timeouts on each of them.
dds-octopus-task - an executable. Acts as a regular "user" task. This the same task is part of all test topologies.
dds-octopus - an executable. Acts as external test manager. It is not a part of topology.