

iRODS – Advanced user training

DATA REPLICATION AND RULES – S4R WORKSHOP

iRODS

Christine Staiger, Arthur Newton



Agenda

9.30 - 10.00 Recap of icommands

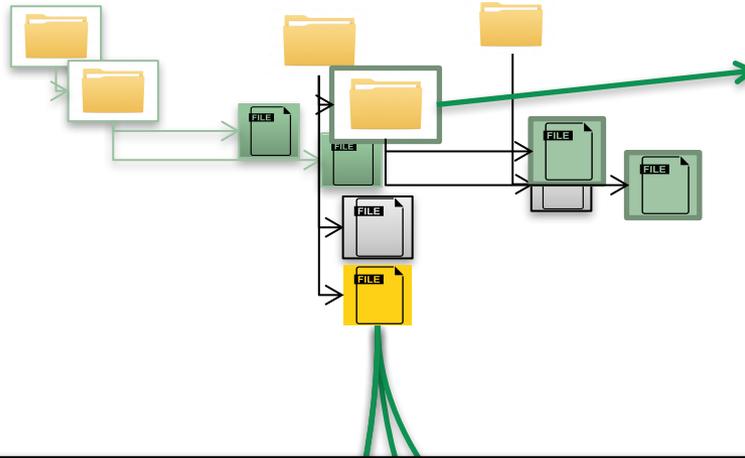
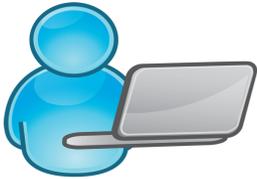
10.00 - 10.30 Data replication

10.30 -12.00 iRODS Federations and data synchronisation

12.00 -13.00 Lunch

13.00 -17.00 Rules, rules rules

Storage – The users' challenge



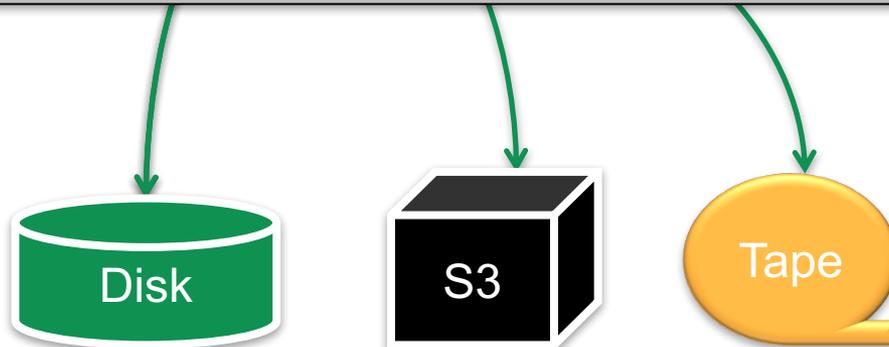
+ Extra information:
attribute: distance
value: 12
units: miles

attribute: author
value: Alice
units:

Abstraction layer:
Mapping from logical
to physical
namespace

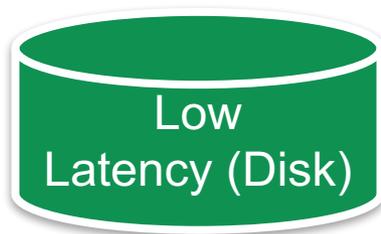


Storage layer:
Different storage media
Different protocols to steer data



In the Background: iRODS resources

- (Storage) Resource is a Software or Hardware system that stores data
- 3 Resource classes:

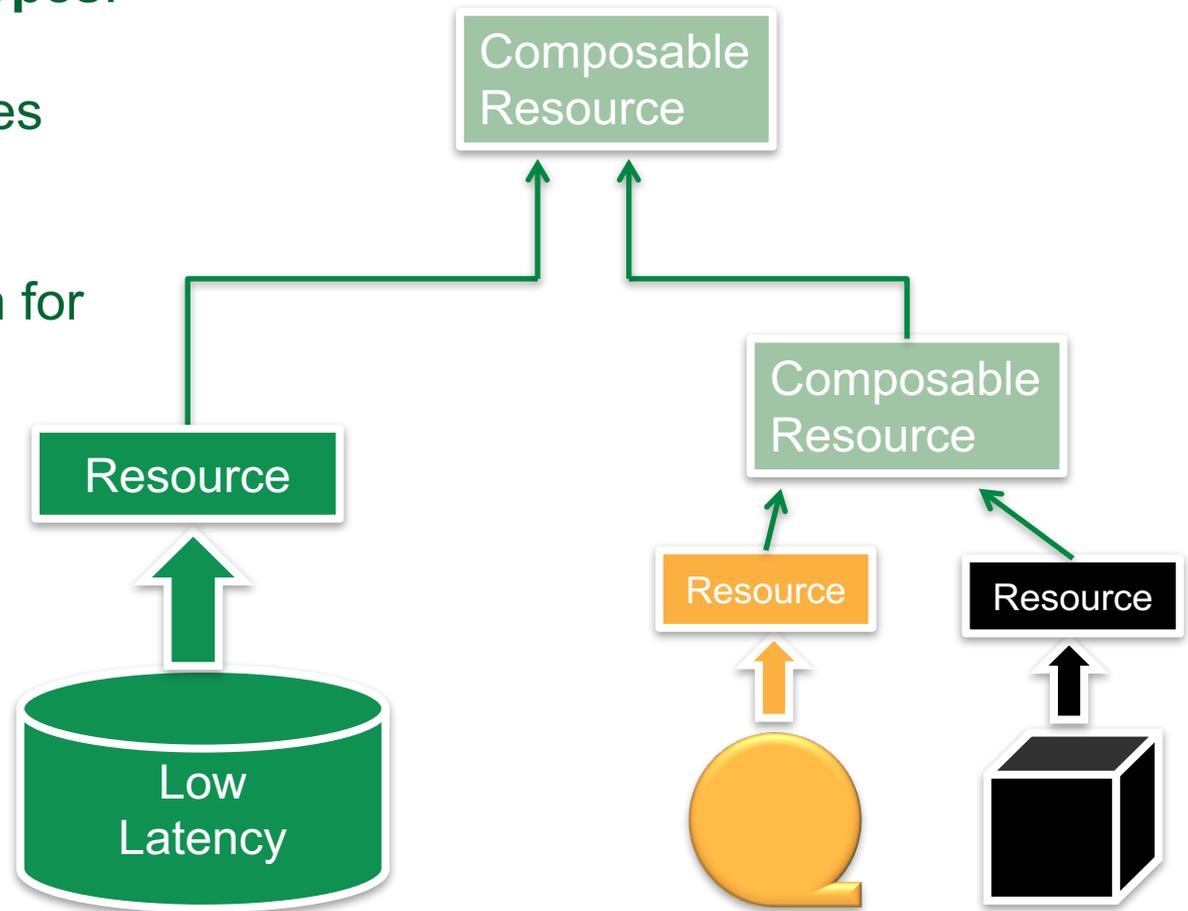


- Storage Resource: unix file system, s3, structured file type univMSS, opendap, tds (THREDDS)

Resource groups

Composable resource Types:

- **Replication**
synchronise resources
- **Round Robin**
rotate through children for uploading
- **Load balance**
- **Compound resource**
cache resource and archive resource



iRODS resources and replicas

- **User** can choose resource: `iput -R <rescname> <file>`
- **User** can create replicas: `irepl -R <rescname> <irods path>`

→ **User** needs to:

- Know setup of the iRODS instance
- Understand the concept of replica
- Know how iRODS handles replicas

→ Suitable for advanced users

- Chose storage medium according to special policies
- Chose suitable storage medium for application

Automatise and standardise the choice of resource as much as possible in the iRODS rulebase (next part of the tutorial).

irepl

iCAT – Zone 1

iCAT entry for file.txt:

Logical path:

/zone1/home/<user>/file.txt

Metadata:

attr1; val1; unit1

attr2; val2; unit2



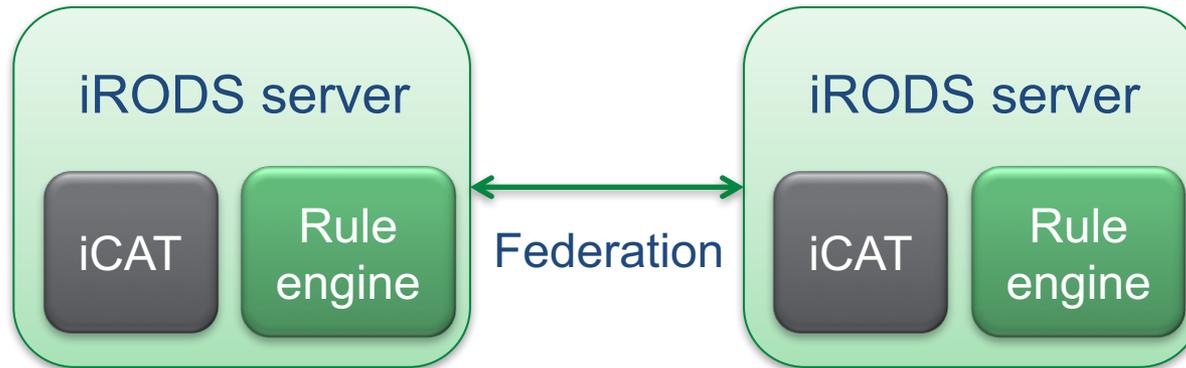
/Vault1/home/<user>/file.txt



/Vault2/home/<user>/file.txt

irepl

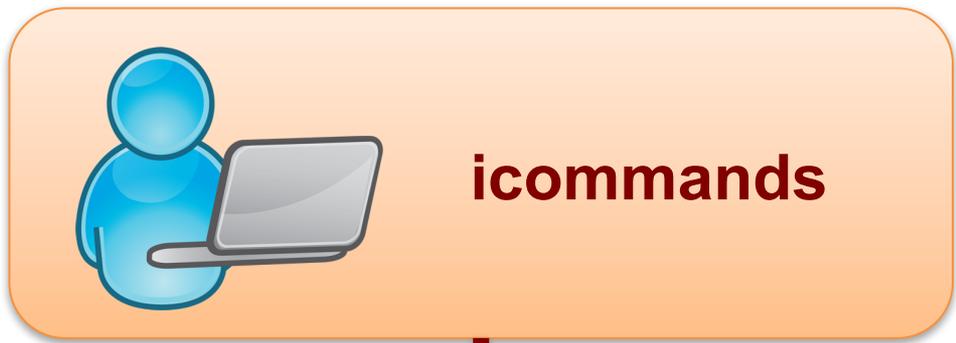
iRODS Federations



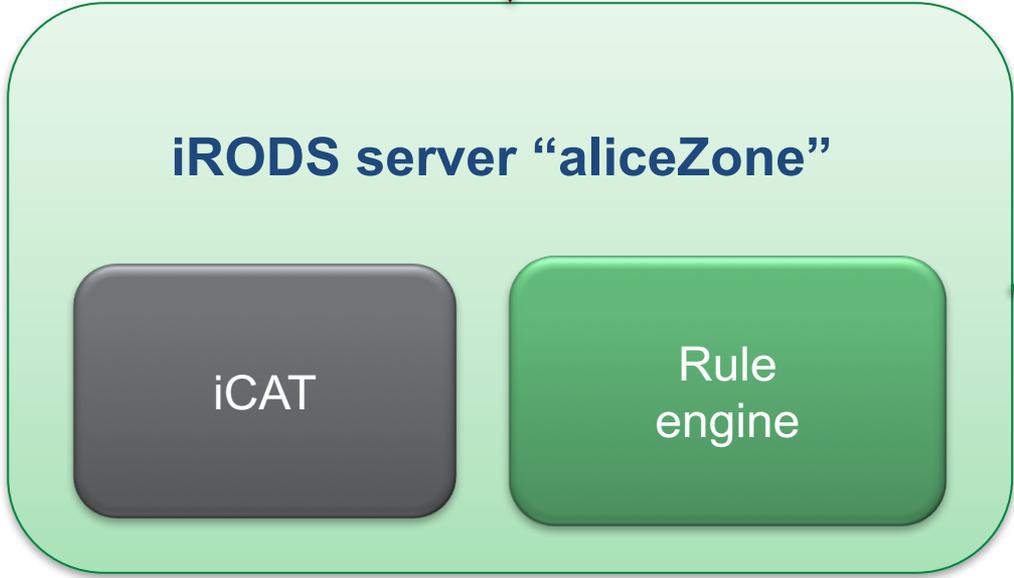
- The iCAT defines the iRODS zone
- Two independent iRODS zones, own rule engine and different rulebases
- Federation on system level
- iRODS admins give access to certain users

User

- Authentication at home iRODS zone (iinit)
- Access to federated zone
`/otherIRODSzone/home/user#homeIRODSzone`

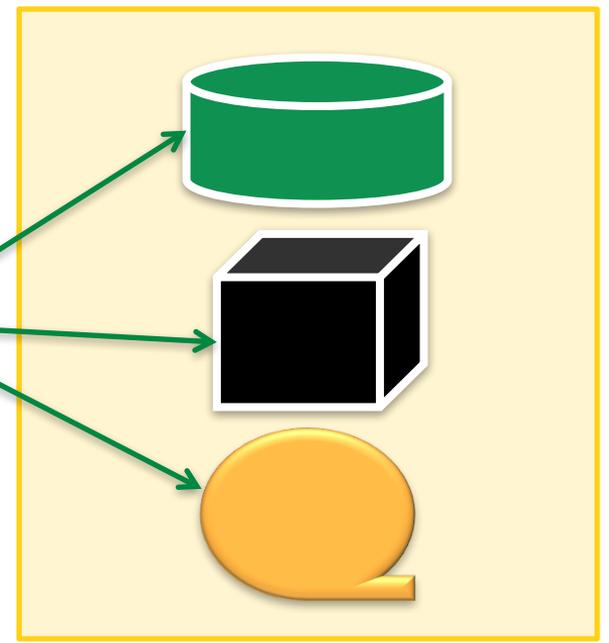


Upload, download ACLs, metadata

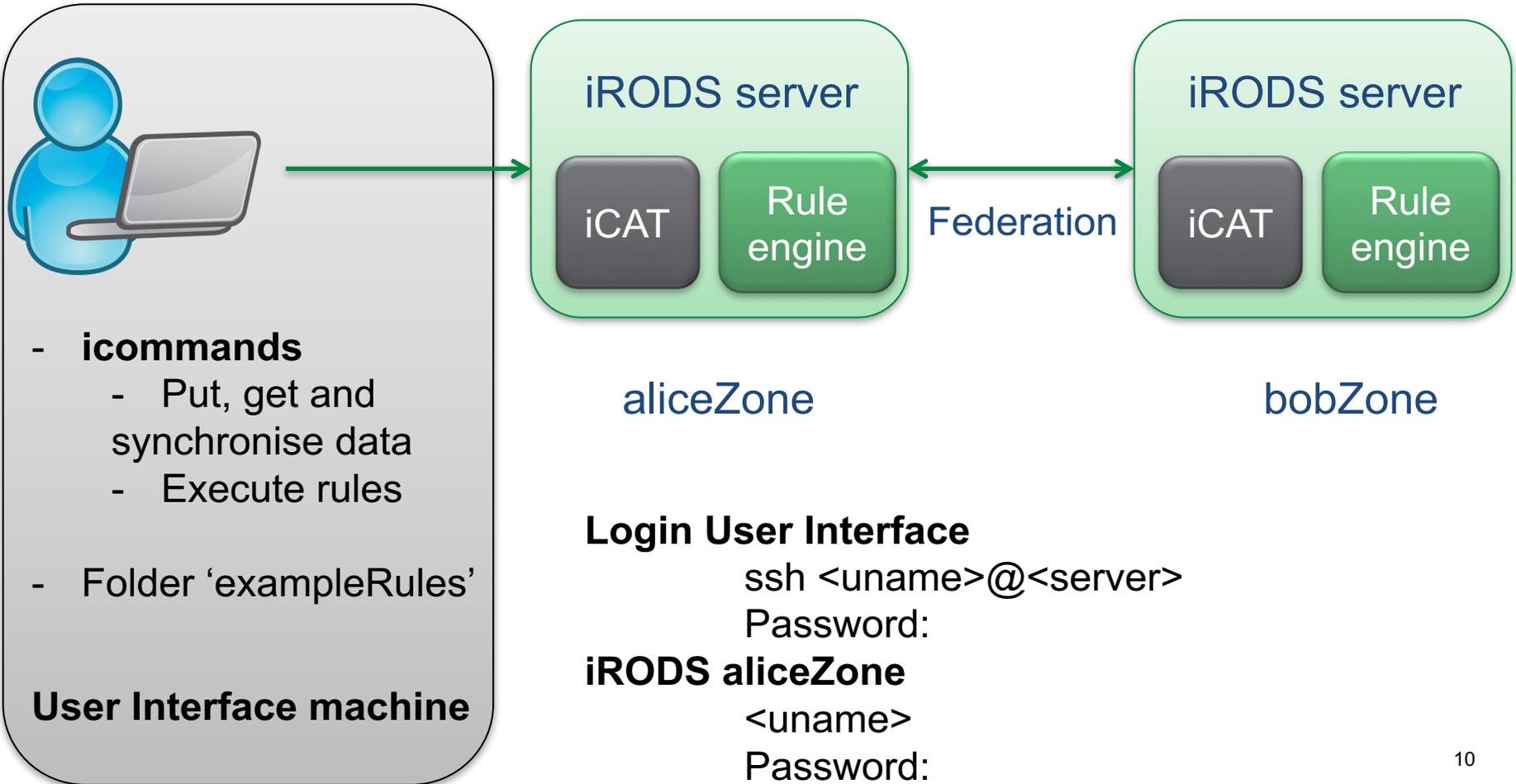


Today:
User Interface machine
Login: **di4r-userX**

Generally:
Lisa/cartesius
module load icommands



Training Setup



10

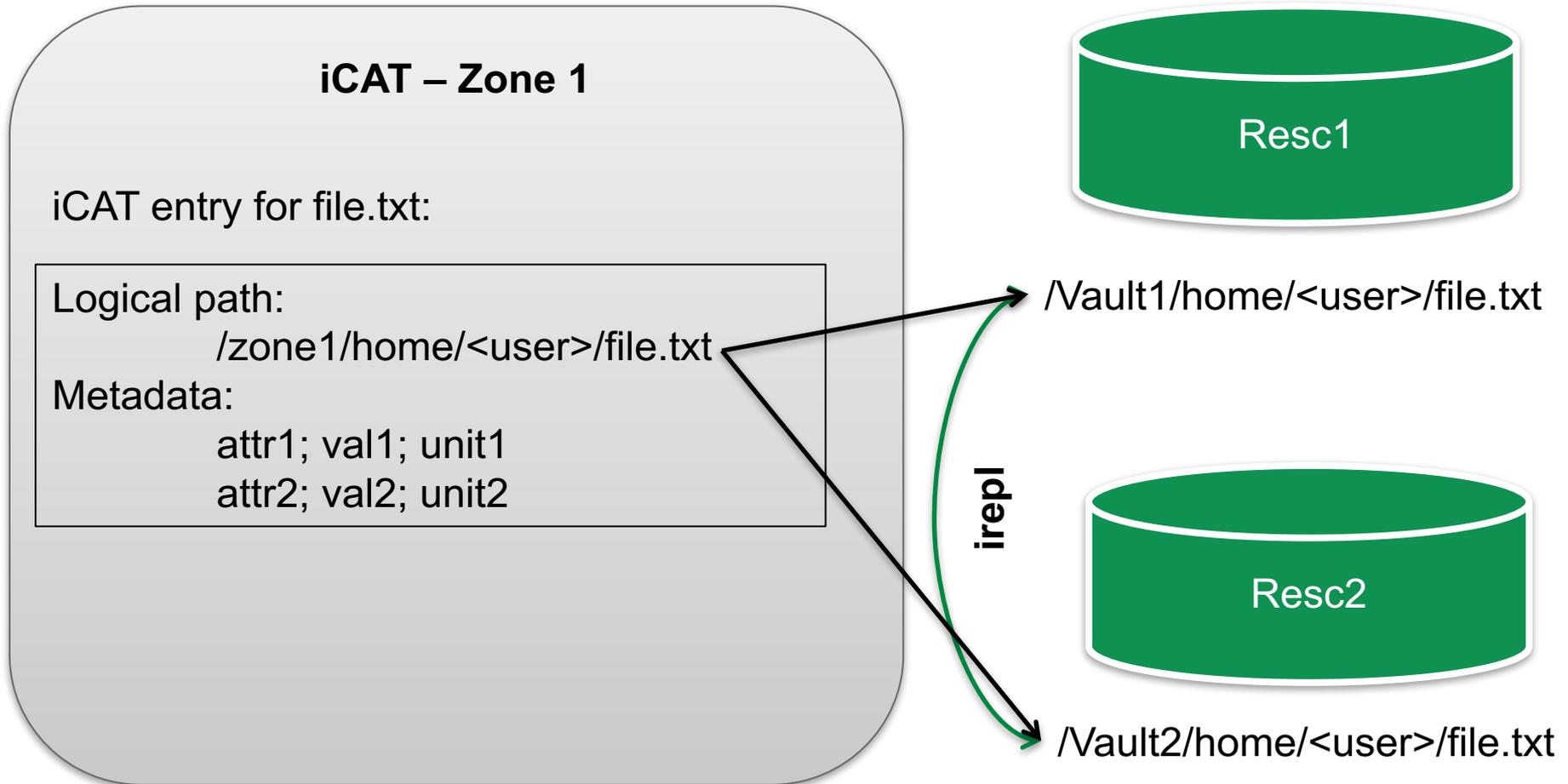


Resources and Federations

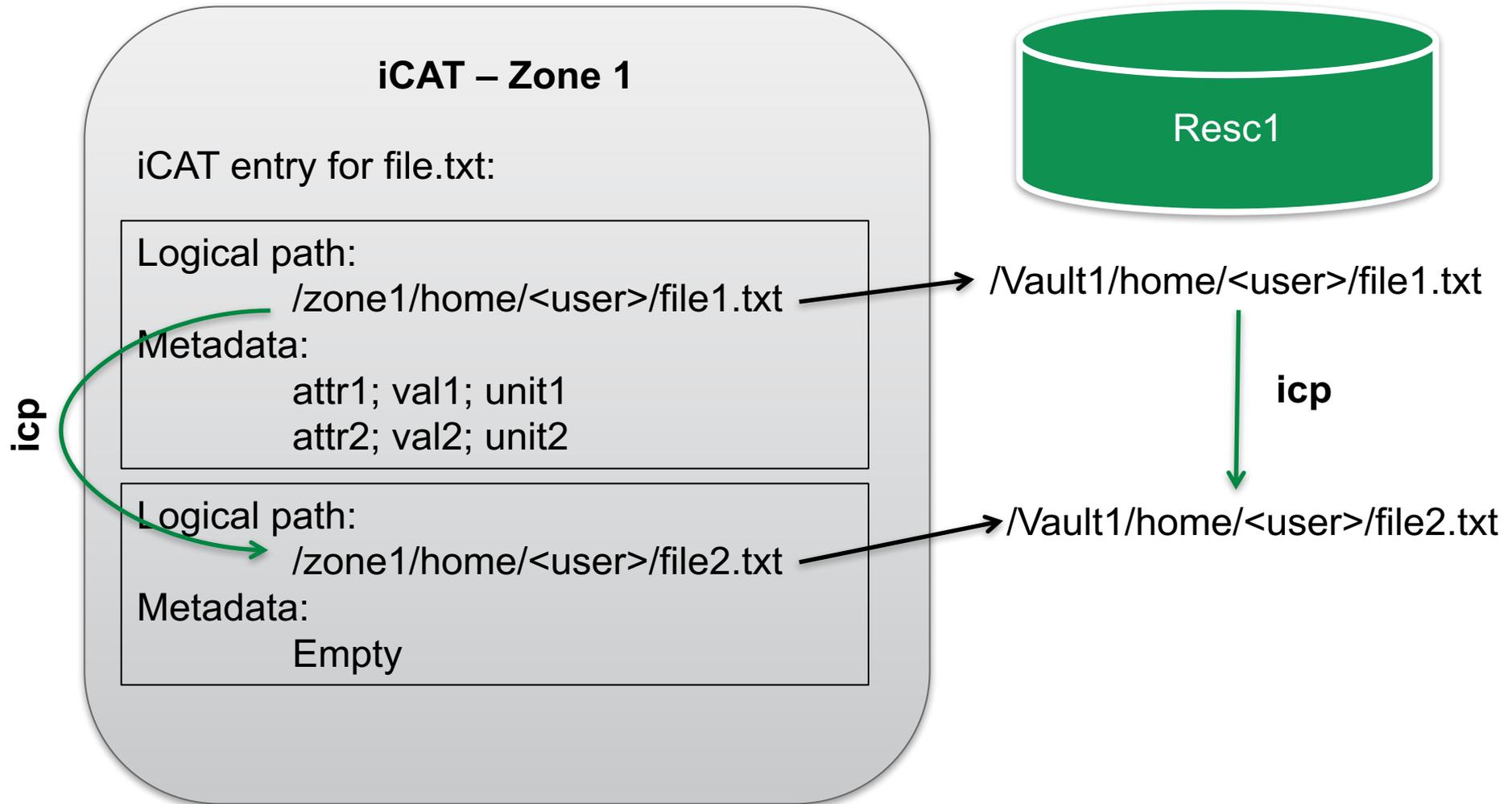
<https://tinyurl.com/iRODS-advanced-HandOut>

Data – metadata relations with imv, icp and irepl

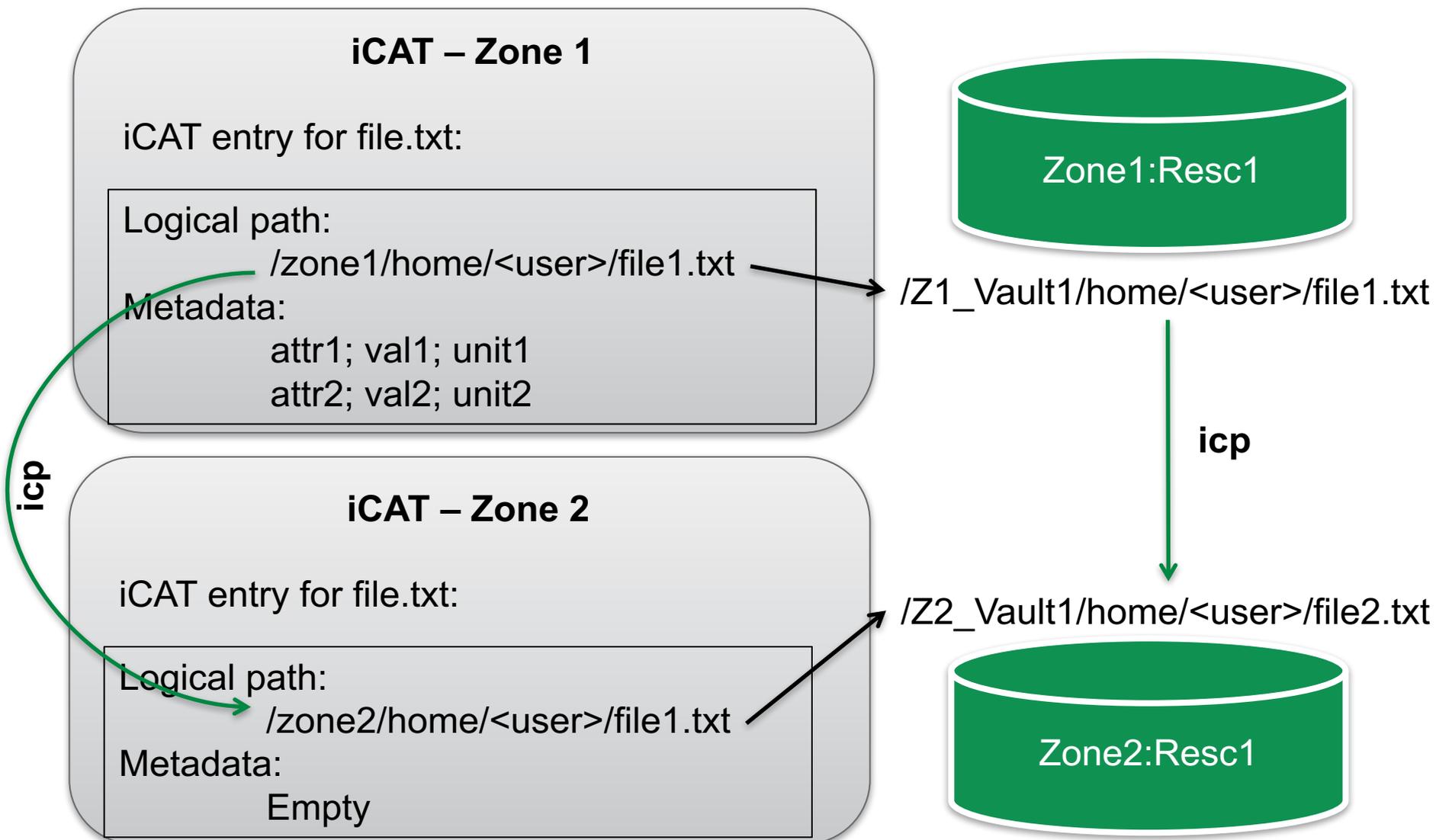
irepl



icp – in one zone



icp/irsync – across zones



imv

iCAT – Zone 1

iCAT entry for file.txt:

Logical path:

imv ↻
/zone1/home/<user>/file.txt → /Vault1/home/<user>/file.txt
/zone1/home/<user>/file_v1.txt → /Vault1/home/<user>/file_v1.txt

Metadata:

attr1; val1; unit1
attr2; val2; unit2



Not possible to do an imv across Zones:

Metadata entry in Zone1 while data resides on resource in Zone 2

Rules and micro services

iRODS micro services

- Define actions on data, resources and users → atomic
- C++ functions, calling external libraries
- Used and combined in workflows/policies → iRODS rules

- Predefined microservices
 - <http://docs.irods.org/4.1.10/doxygen>
- Example: msiCollRsync → synchronises two iRODS collections from different zones

- Own micro services:
 - Written in C++
 - Need to be installed on the iRODS server → root or iRODS service account rights
 - Example: Automatic metadata extraction from HDF5 files

iRODS rules

- iRODS rule engine → built-in interpreter for own language
- Automate data management tasks
- Standard set of pre-implemented rules constitutes default data policies

- Trigger execution of rules by
 - irule → User
 - Delayed or scheduled execution → User & iRODS admin
 - Actions and policy enforcement points extending and overlaying the default rule base → iRODS admin

```
HelloWorld{  
    writeLine("stdout", "Hello *name!");  
}  
INPUT *name="World"  
OUTPUT ruleExecOut, *name
```

iRODS standard data policies

- Event hooks are triggered by actions
 - E.g. put data (client interaction - iput)
 - acPostProcForPut - Rule for post processing the put operation.
- Policy enforcement points (PEPs) are executed by the rule engine

```
acPostProcForPut {  
    msiSysChksumDataObj;  
    msiSysReplDataObj("demoResc","all"); }
```

```
pep_api_data_obj_put_post( *COMM, *DATAOBJINP,  
*BUFFER, *PORTAL_OPR_OUT) {  
    acPostProcForPut; }
```

Extending the standard core.re

- Predefined core.re and also pretty empty in standard setup
 - Placeholder for all event hooks and PEPs
 - Placeholder for own general data management rules
- Place your (carefully tested) rules directly into core.re
 - bad idea
- Write an own policy.re and configure server

```
"re_rulebase_set": [{"filename": "policy"}, {"filename": "core"}]
```

 - policy.re and core.re build the ruleset for this iRODS instance
 - Order matters

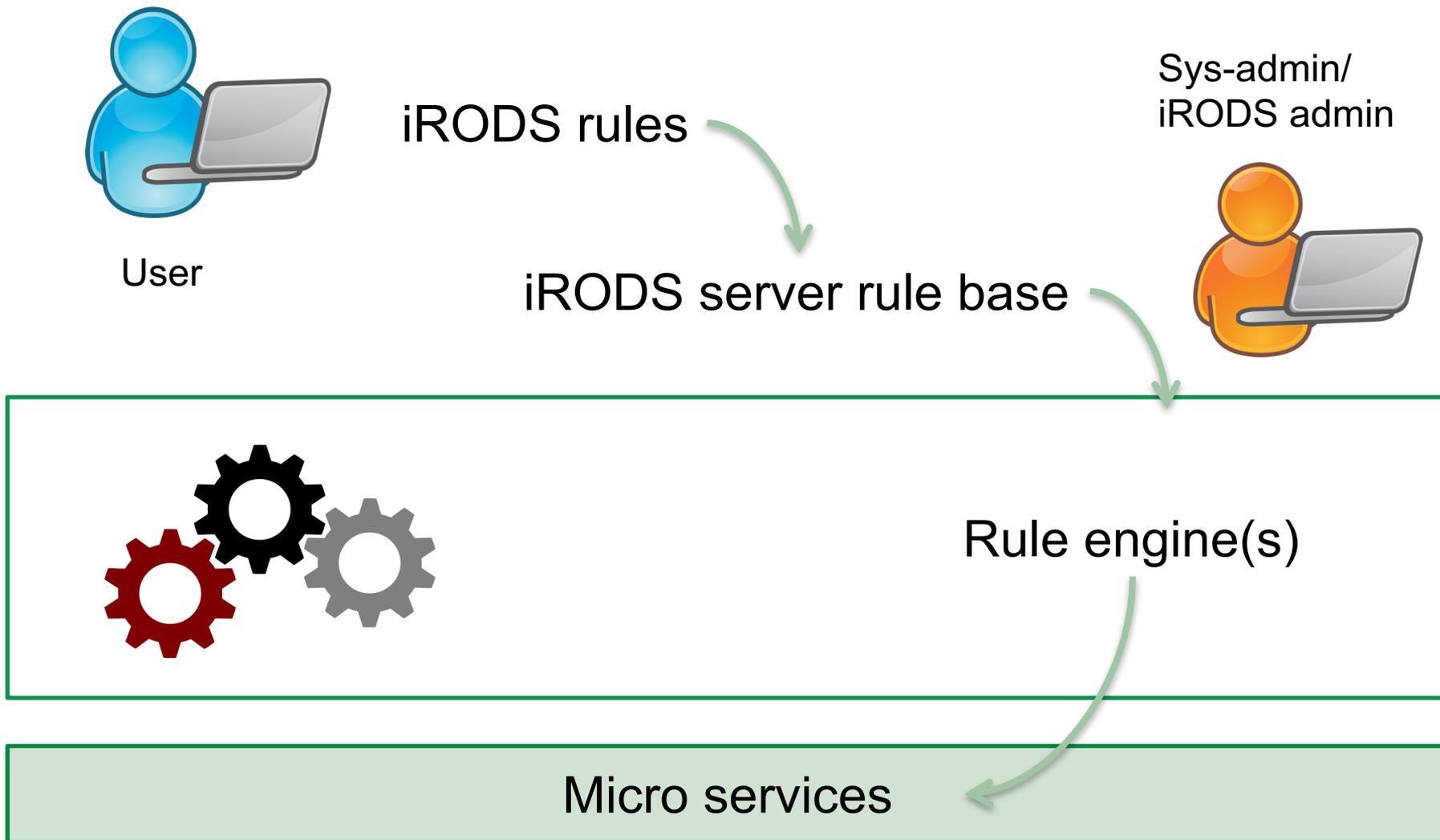
Rules: Order matters

- No namespaces!
- First rule that matches (name and variables) will be executed
- Event hooks and PEPs follow the syntax of rules

Workflow for developing policies/rules

- Write a local rule as iRODS user → `irule <file>`
 - Debugging
- Put rule on top of all rules in the configured rule set
 - Does it still work?
 - Which rules does it inhibit from being executed
- Bit by bit find the right spot for the rule in the rule base

The Hierarchy





Write your own data archiving policy/rule



Thank you! Questions?

Special thanks to:

Manon van Eijden (SURFsara)

Jan Bot (SURFsara)



[arthur.newton\(at\)surfsara.nl](mailto:arthur.newton@surfsara.nl)
[christine.staiger\(at\)surfsara.nl](mailto:christine.staiger@surfsara.nl)

