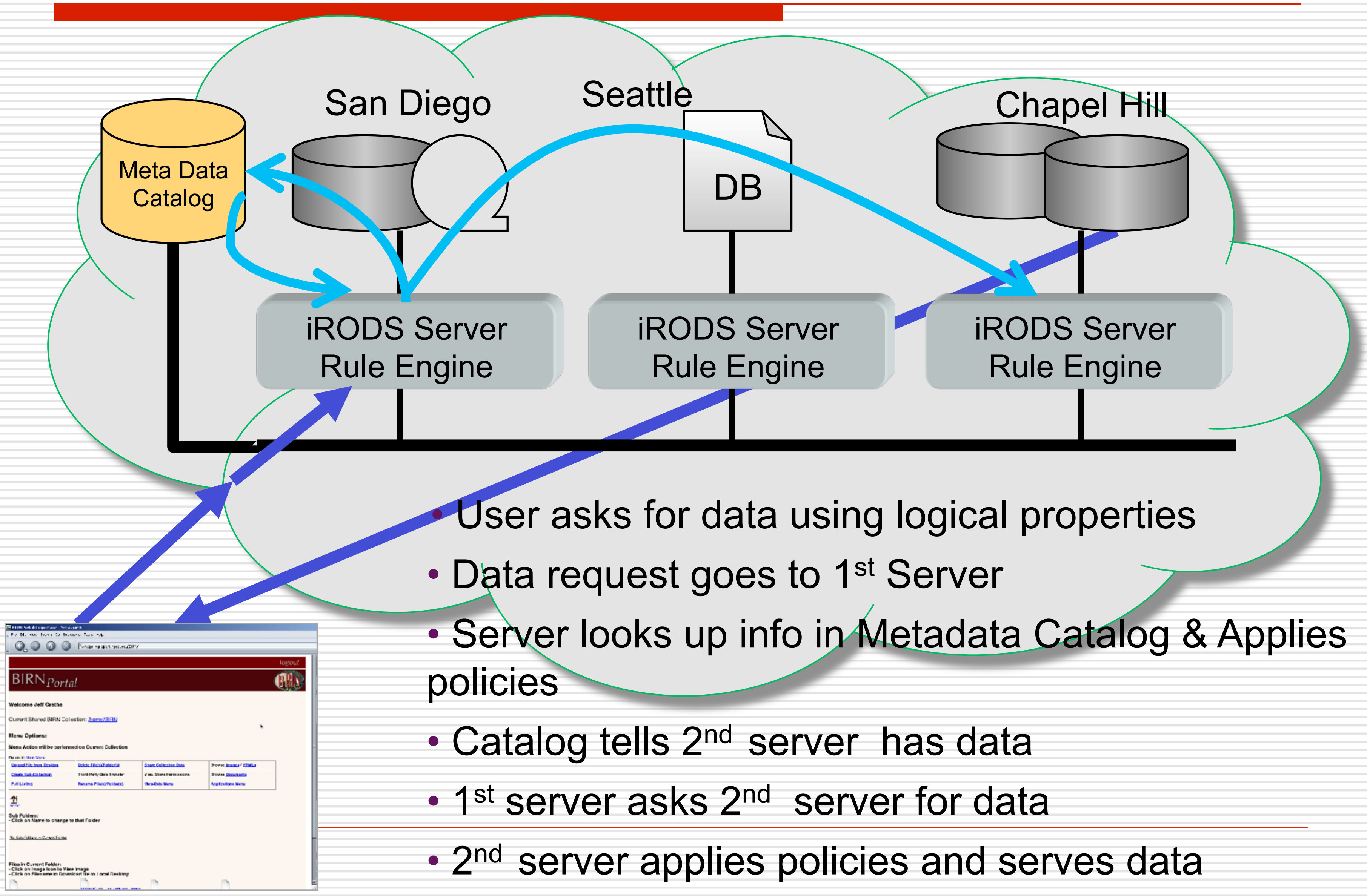
How to Code Policies in iRODS

Arcot (Raja) Rajasekar
Reagan Moore
DICE Center/SILS/RENCI
University of North Carolina
Chapel Hill, NC, USA.

Introduction

- How Policy Is Invoked?
- Policy Points – Where Policy is Invoked in iRODS
- Rules as computed-executable Policies
- Semantics of Rule Execution
- Data Involved in Policy Execution
- Sample Policies through Examples

Using a Policy-EnabledData Grid



- User asks for data using logical properties
- Data request goes to 1st Server
- Server looks up info in Metadata Catalog & Applies policies
- Catalog tells 2nd server has data
- 1st server asks 2nd server for data
- 2nd server applies policies and serves data

Policy Points in iRODS

- Policies are applied at specific points in iRODS
- Examples:
  - Just before data creation (useful to find where to put the data, what access control to apply, what type of streaming to use, versioning of overwritten files,…)
  - Just after data has been transferred into iRODS (useful to perform metadata extraction and registration into Metadata Catalog, Replicating or Copying Data, Computing Checksums, Giving Other user access permissions, setting flags, launching tasks to create derived products, sending email to subscribed users,…)
  - Just before removing a file/collection (useful to stop this operation, putting things into trash bin instead of removing it, notifying someone, delaying the operation for a day, removing derived products,…)
  - Just before creating a user (useful to put in groups, creating system objects such as home collection, trash bin,etc., making more security checks, notifying group managers, creating user profiles, …)

Some Policy Points in iRODS

- acPostProcForDelete
- acPostProcForCollCreate
- acPostProcForRmColl
- acPostProcForModifyUser
- acPostProcForModifyAVUmetadata
- acPostProcForCreateUser
- acPostProcForDeleteUser
- acPostProcForCreateResource
- acPostProcForCreateToken
- acPostProcForModifyUserGroup
- acPostProcForDeleteResource
- acPostProcForDeleteToken
- acPostProcForModifyResource
- acPostProcForModifyResourceGroup
- acPostProcForModifyCollMeta
- acPostProcForModifyDataObjMeta
- acPostProcForModifyAccessControl
- acPostProcForObjRename
- acPostProcForGenQuery

- acPreprocForDataObjOpen
- acPreprocForCollCreate
- acPreprocForRmColl
- acPreProcForModifyUser
- acPreProcForModifyAVUmetadata
- acPreProcForCreateUser
- acPreProcForDeleteUser
- acPreProcForCreateResource
- acPreProcForCreateToken
- acPreProcForModifyUserGroup
- acPreProcForDeleteResource
- acPreProcForDeleteToken
- acPreProcForModifyResource
- acPreProcForModifyResourceGroup
- acPreProcForModifyCollMeta
- acPreProcForModifyDataObjMeta
- acPreProcForModifyAccessControl
- acPreProcForObjRename
- acPreProcForGenQuery

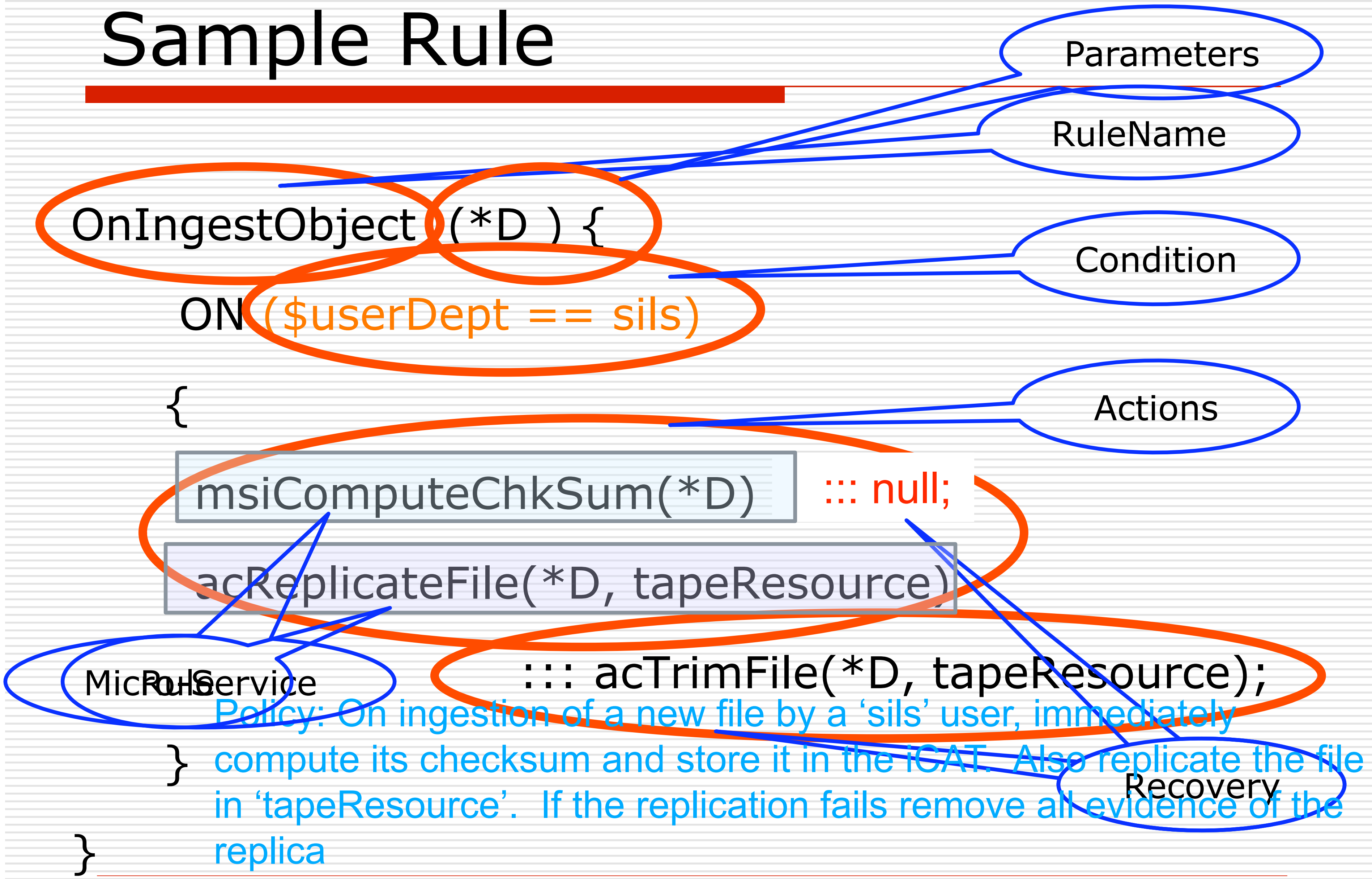
Policy Points in iRODS

- Current Policy Points: 64
- Strategically placed in the server code
- If new ones are needed, we can add them.
- Mostly policy points are dual: before and after some data management task.
- Each policy point invokes a rule stored in core.irb under a given ruleName
- Policies are made of alternative rules but only one policy is executed fully.

iRODS Rules

- Each rule has several parts:
  - RuleName – so that one can invoke a Policy
    - There can be more than one rule for a RuleName
  - Condition – A ‘guard’ which checks if a rule can be fired or not
    - If one rule does not fire, the next rule with same RuleName is tried
  - Action chains - Body of the rule
    - List of Functions (workflow) performed
    - Made of micro-services and other rules
  - Recovery chains – What to do when an action fails
    - Made of micro-services and other rules

Sample Rule



```
OnIngestObject (*D ) {
      ON ($userDept == sils)
      {
       msiComputeChkSum(*D)  ::: null;
       acReplicateFile(*D, tapeResource)
                ::: acTrimFile(*D, tapeResource);
      }
}
```

Policy: On ingestion of a new file by a 'sils' user, immediately compute its checksum and store it in the iCAT. Also replicate the file in 'tapeResource'. If the replication fails remove all evidence of the replica

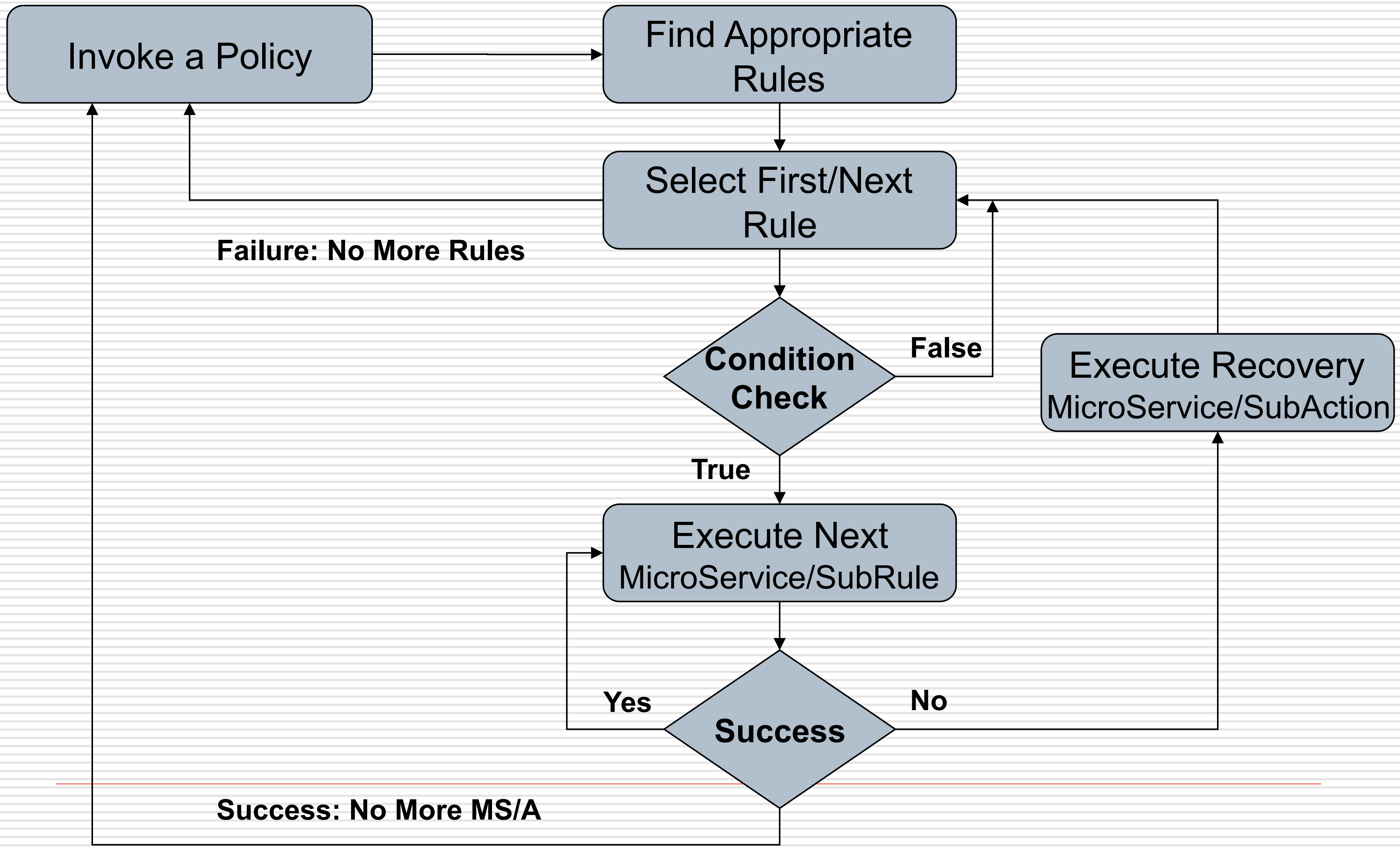
Sample Rule – Internal Form

```
OnIngestObject  (*D ) {
        ON ($userDept == sils)
       {
         msiComputeChkSum(*D) ;
         acReplicateFile(*D, tapeResource)
                                    ::: acTrimFile(*D, tapeResource);
       }
}
```

```
OnIngestObject(*D) | $userDept == sils  |
msiComputeChkSum(*D)##acReplicateFile(*D,tapeResource) |
null##acTrimFile(*D,tapeResource)
```

Conversion done by “rulegen” utility found in icommands/rulegen
See also the “Rules” page in iRODS Wiki

Policy Invocation

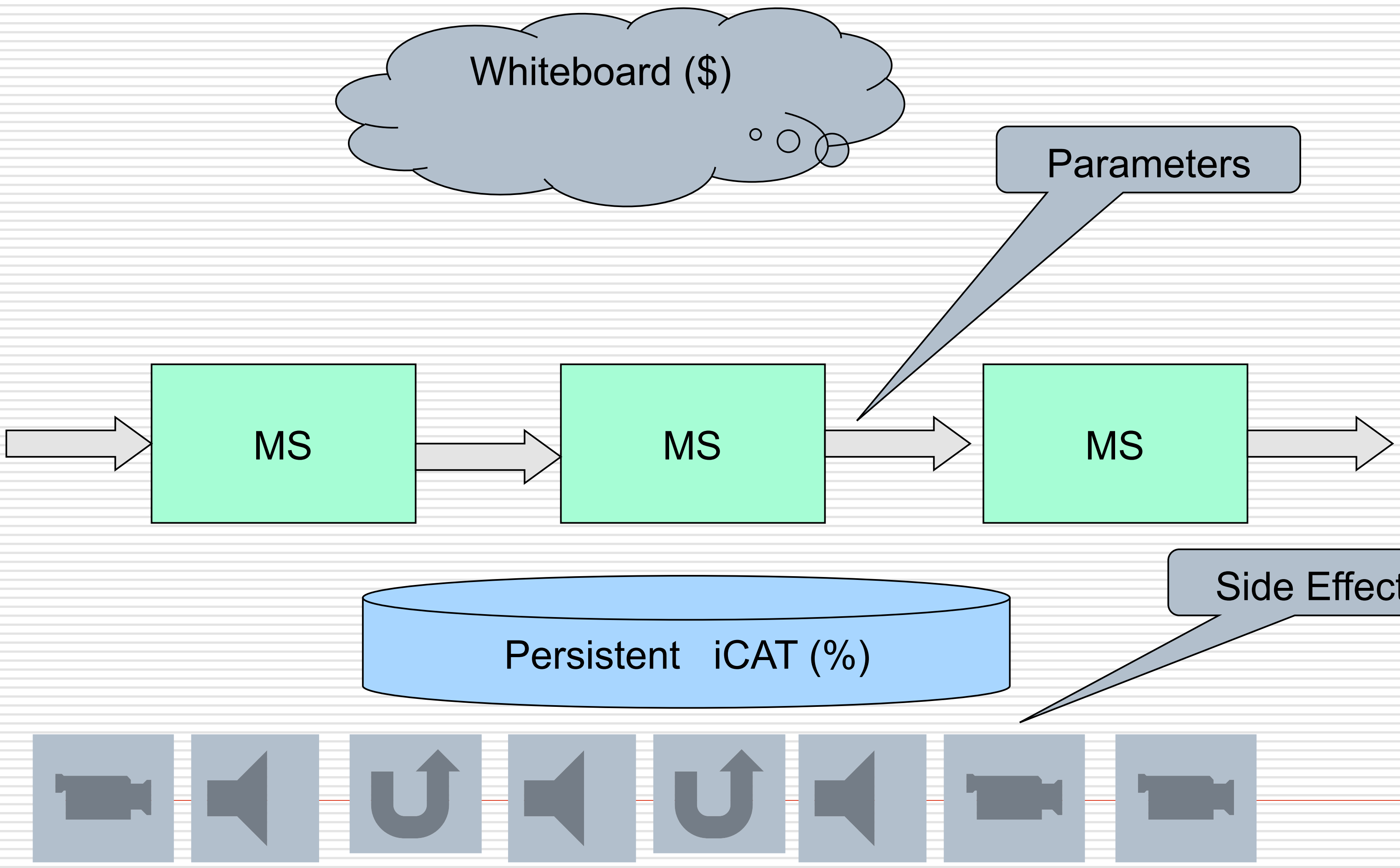


How the Rule Engine Works:

```
A:   C1 | M1 M2      | R1 R2
A:   C2 | M3 M4      | R3 R4
A:   C3 | M5 M6 M7 | R5 R6 R7
A:   C4 | M8 M9      | R8 R9
```

```
Execute A    (Policy Invokes ruleName A)
   Check C1 (success)
     Execute M1 (success)
     Execute M2 (fail)
       Execute R2
       Execute R1   /*R1 is also executed!*/
   Check C2 (fail)
   Check C3 (success)
   Execute M5 (success)
   Execute M6 (success)
   Execute M7 (succes)
A succeeds   (Policy Succeeds)
      /*  C4 is not even checked   */
```

Data Flow between Micro-services



Parameter Passing

- Part of the MicroService Signature

```
int findObjType ( msiParam_t *objInParam ,
                  msiParam_t *typeOutParam ,
                  ruleExecInfo_t *rei );
int ingestBulkMD ( msiParam_t *objInParam,
                   msiParam_t *typInParam,
                   msiParam_t
                       *keyValuePairsInParam,
                   ruleExecInfo *rei);
```

- When used in a rule the “rei” parameter is implicit.

WhiteBoard ($) Variables

- They are stored in a structure: rei
- Some common ones that are of interest
  - $objPath collection-path name of data object
  - $rescName name of resource
  - $userNameClient name of client-user
- How to Use them:
  - Condition checking:
    `$objPath like /zone/home/sekar/nvo/*`
  - Parameter passing:
    `findObjType($objPath,*Type)`
  - assign($rescName, duke-samqfs)
- You can find the $-variable names in:
  - server/config/reConfigs/core.dvm

Policies And $-variables

- Not all $-variables needed for every policy (ruleName).
- We can find what $-variable is available at each policy-point from Table 5.2 and 5.3 in *iRODS Primer*.
- Example:
  - acCreateUser : $otherUserName, $otherUserZone, all connection-level $-variables (called S1).
  - acPreProcForObjRename: $objPath (old path) + S1
  - acPreProcForCollCreate: $collName, $collParentName + S1
  - acPreProcForDataObjOpen: S3 (data object)+ S4 (resources) + S1 (connection) = 50 variable information…..

WhiteBoard: ruleExecInfo (*rei)

- A large data structure shared when invoking a rule
- Implicitly used throughout the rule processing
- MicroServices can access values/structs in the *rei and also set values in the *rei structure
- The structure is defined in reGlobalsExtern.h and it can be extended if necessary
- Contains various important structures used in the iRODS data management:
  - *rsComm - client-server communication structure
  - *doi - dataObject information
  - *rescGrp - resource (group) informations
  - *uoic - client user information
  - and others ….
- The rule invoking function should set the proper values…

Example: ATM (1)

- Let us try to code the policy for an ATM to disburse money.
- Assume that authentication has already been done.
- Policy : payMoney

  If a user asks for money below a ‘ceiling value’ and has a balance above the asking value, disburse the money and debit the user account. Recover from disbursement failure.

Example: ATM (2)

What are the key Policy-Points:

- Get User Information
- Get Amount & Account Information
- Check Balance and Ceiling (CheckMoney)
- Subtract from the User Account Value
- Count Money into the PayBin
- Open the PayBin to Pay the User
- Give Receipt

- Commit/Rollback Changes
- Write Paper Records
- Shutdown
- Notify User

Example: ATM (3)

```
acCheckMoney (*U, *A)
{
       ON (*A < 300)
       {
          display("You cannot withdraw
                         more than $300");
          displayExitMessage;
       }
       OR ON (*A < balance(*U) )
       {
          display("You have insufficient balance");
          displayExitMessage;
       }
}
```

*U = user *A = Amount to pay

From Policies to Rules

- Write the policy with clear “keywords” that define side-effects that can be performed by micro-services.
- Identify recovery mechanisms for failure
- Create high-level signatures for the micro-services – split complicated micro-services
- Form a workflow based on the micro-services and test various paths
- Search existing rules/micro-services which can be used.
- Code micro-services, if needed, and unit test
- Write and test the rules

Some Sample Policies

- acCreateUser (default policy in core.irb)
- acDataDeletePolicy (not a default – can be turned on at admin’s discretion)

- Policies generated in DCAPE Project

acCreateUser

- Used by iRODS when an administrator creates a new user.
- Flexibility to add “new” features when creating users
  - Create a trash bin
  - Add user to groups based on her domain
  - Verify the user in a list or external database or with some community authentication system
  - Allocate storage and quotas
  - Notify someone about this new user (may be the domain manager)
  - Send the new user some emails about how to use irods

acCreateUser – by default

```
acCreateUser {
   ON ($otherUserName == anonymous)
   {
        msiCreateUser                        ::: msiRollback;
        msiCommit;
    }
   OR
    {
        msiCreateUser                        ::: msiRollback;
        acCreateDefaultCollections           ::: msiRollback;
        msiAddUserToGroup(public)            ::: msiRollback;
        msiCommit
    }
```

acCreateDefaultCollections

```
acCreateDefaultCollections
 {
   acCreateUserZoneCollections
 }
acCreateUserZoneCollections
 {
    msiCreateCollByAdmin(/$rodsZoneProxy/home,
                              $otherUserName );
    msiCreateCollByAdmin(/$rodsZoneProxy/trash/home,
                              $otherUserName );
  }
```

- Creates two collections a ‘home’ and a ‘trash’

acDataDeletePolicy

- Can be used to disallow deleting files from a collection

```
acDataDeletePolicy
{
   ON ($objPath like /myzone/home/sekar/*)
   {
     msiDeleteDisallowed;    /*sets a disallow flag */
   }
   OR
   {
      nop;
   }
}
```

DCAPE Policies & Rule Transformation

http://ils.unc.edu/spaces/dcape/index.php/DCAPE_Initial_Capabilities

How To Execute a Rule

Rules get executed

- As part of a server-function invoked by the iRODS server
  - Ex. rsDataPut calls acPostProcForPut
- As part of a scheduled activation
  - Scheduled by other rules/micro-services
  - iRODS has a built-in scheduler/executor
- As part of an explicit user invocation
  - Using the client-side irule command

Scheduling A Rule/Micro-Service

- Using delayExec micro-service as part of a rule in core.irb

```
acPostProcForPut
    | $objPath like /tempzone/home/rods/nvo/*
    | delayExec(<PLUSET>1h</PLUSET>,
            msiReplDataObj(nvoReplResc),
            sendEmail(sekar, $objName failed to replicate) )
    | nop
```

- Pol cy: On ingestion of a new file in "nvo" collection, re cate the file in 'nvoReplResc' after 1 hour; If the fails, s out it.

Recovery for the delayExec micro-service

Recovery for the replication operation in delayExec

iRule Client Command

- Can use a rule-file (*.ir files)
- Rule-files Contain exactly 3 lines
  - First line is the rule, the second line has the input values and third has a variable list to print to screen

```
myTestRule || acGetIcatResults(*Action,*Condition,*B)##forEachExec
(*B,msiDataObjChksum(*B,forceChksum,*C),nop)| nop##nop
*Action=chksum%*Condition= COLL_NAME = '/tempZone/home/rods/tst’
*Condition%*Action%ruleExecOut
```

- Policy: For every file in COLL_NAME (as given by iCAT) force compute its checksum and store in iCAT
- How to run irule:

  irule –v –F myRule.ir

From Policies to Rules

- Write the policy with clear “keywords” that define side-effects that can be performed by micro-services.
- Identify recovery mechanisms for failure
- Create high-level signatures for the micro-services – split complicated micro-services
- Form a workflow based on the micro-services and test various paths
- Search existing rules/micro-services which can be used.
- Code micro-services, if needed, and unit test
- Write and test the rules

Conclusion

- Policies play an important role in iRODS
- They provide a way to customize the iRODS
- Policies translate to rules
- Rules are executed at pre-defined policy points in the data management server code.
- Rules use multiple types of data information to perform their tasks