Tutorial - Distributed Data Analysis using R

Fraunhofer Institut Intelligente Analyse- und Informationssysteme

Stefan Rüping, Michael Mock, Dennis Wegener

Dortmund, August 2008

The Lecturers

- Stefan Rüping
- Michael Mock
- Dennis Wegener

Fraunhofer Institut Intelligente Analyse- und Informationssysteme

Goals of the tutorial

- Get to know different concepts and applications of distributed systems
- Get an overview of available R packages in the context of distributed computing
- Introduction to the GridR package
- See some real life examples

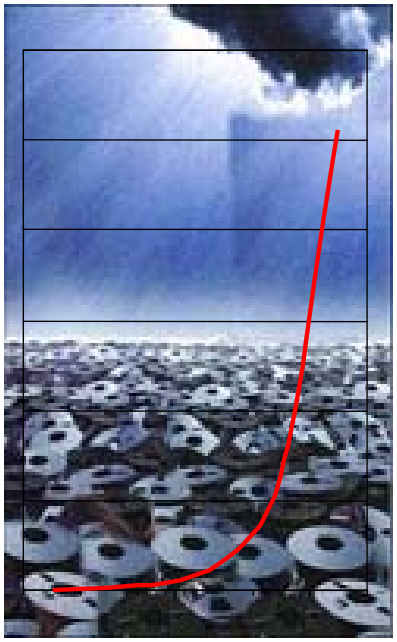
Outline & Timetable

- 9:00 – 9:15 Introduction
- 9:15 – 10:15 R in the Context of Distributed Systems
- 10:15 – 10:30 GridR Installation
- 10:30 – 11:00 Coffee Break
- 11:00 – 11:45 The GridR Package
- 11:45 – 12:15 Real-World Examples
- 12:15 – 12:30 Discussion

Introduction

Why Grid-R?

Some Trends in Technology and Society



Convergence

universal digital representation,
digital photos, MP3, web, podcasts, internet-tv

Ubiquitous Intelligent Systems

mobile phones and PDAs, RFID, sensor networks, embedded systems

Users as Producers

web, wikipedia, blogs, podcasts, social software (Flickr, del.icio.us, Upcoming)

Autonomous Agents

Trading agents (ebay, B2B), self-learning mail filters, virtual opponents in games, autonomous robots

- **Need for state-of-the-art, rapid data analysis** → **R**
- **Need for easy integration and distribution** → **Grid**

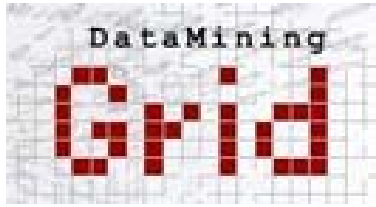
What is the Grid?

- Using a distributed system should be as easy as plugging in a toaster
  - No need to think about where electricity is generated
  - One type of plug works everywhere
- Grid is about making distributed computing
  - Easy
  - Secure
  - “sellable”
- Related Buzzwords
  - Cloud Computing
  - Software as a Service
  - Web 2.0

Ian Foster and Carl Kesselman: "The Grid: Blueprint for a new computing infrastructure"

Our History in Data Mining on the Grid

- **DataMiningGrid** (EU 2004-2006) – Data Mining and Grid Computing

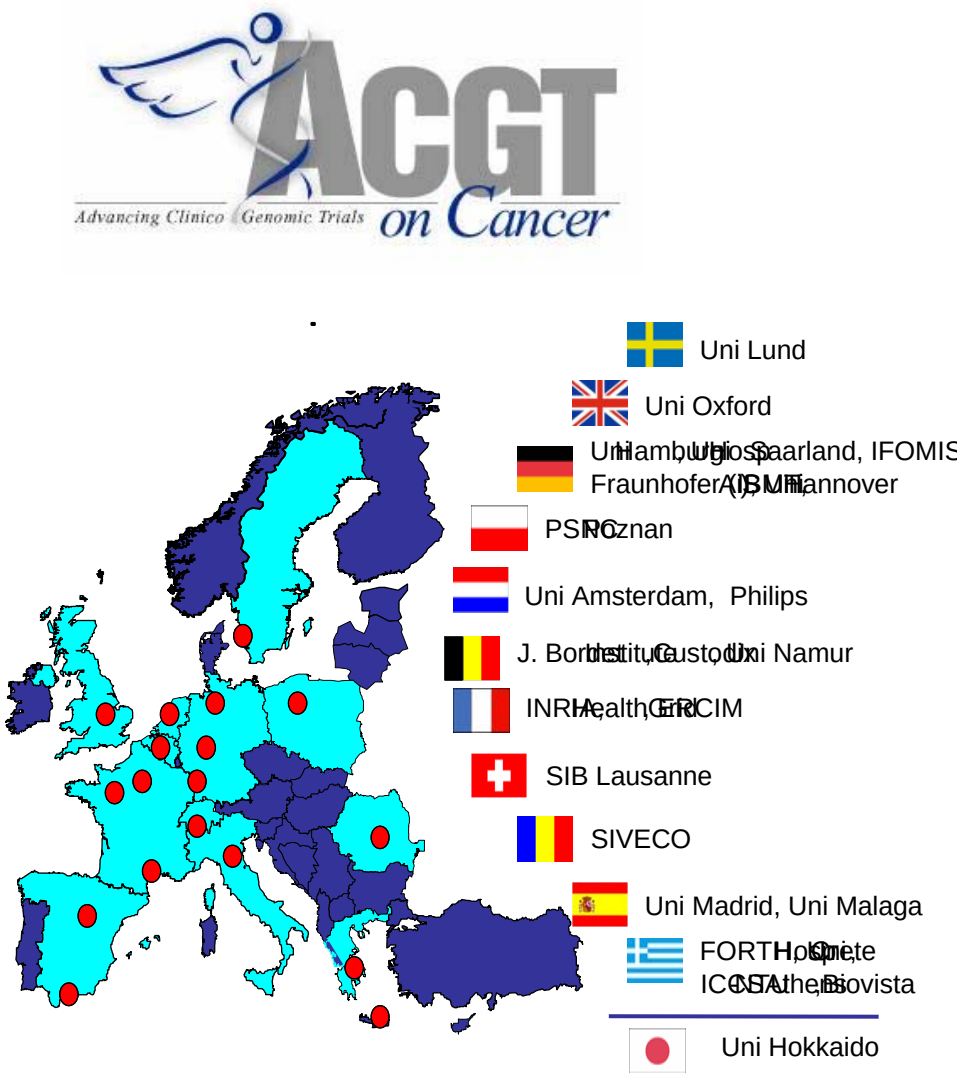


- **SIMDAT** (EU, 2004-2008) – Data Mining and Grid Computing for Complex Industrial Applications

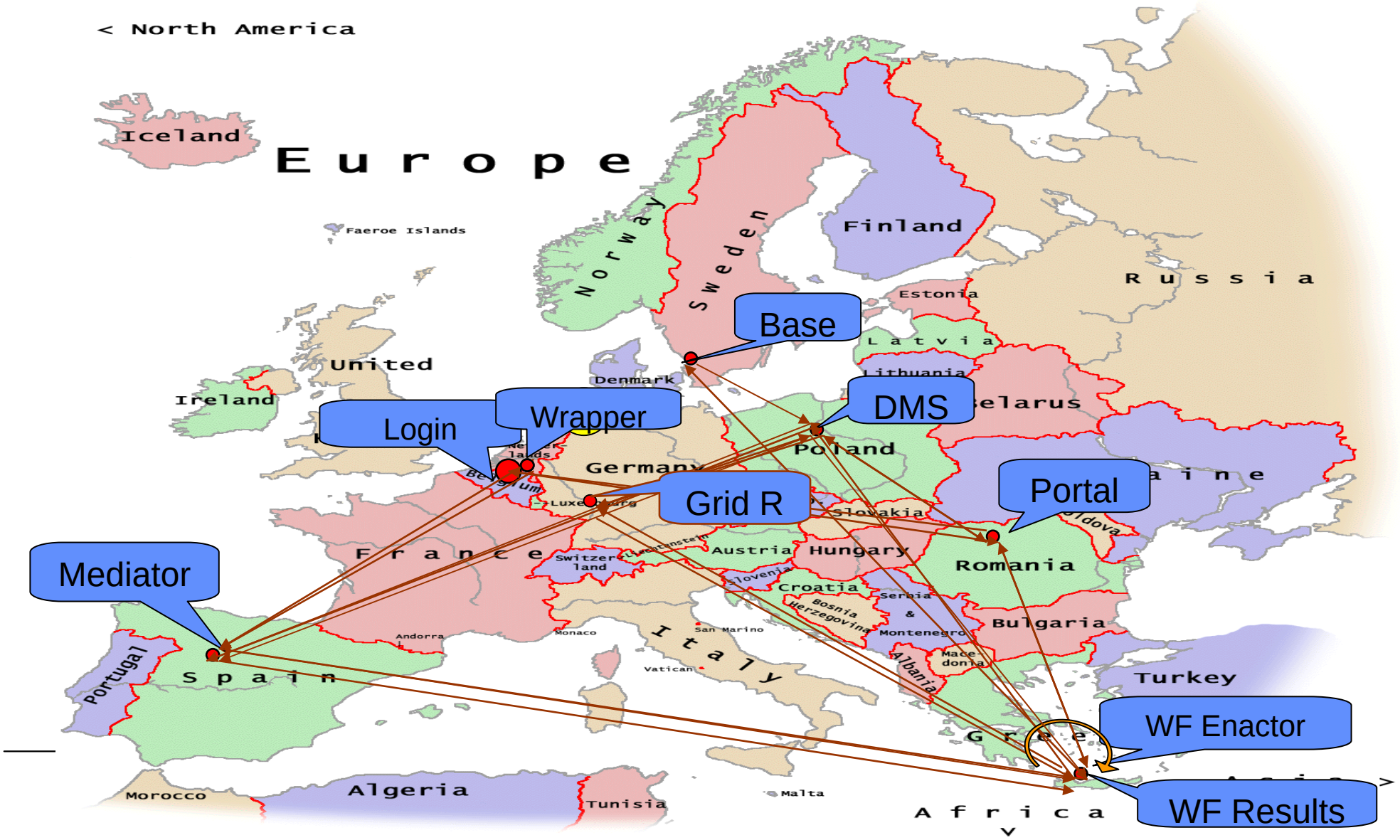
- **ACGT** (EU 2005-2009) – Advancing Clinico-Genomic Trials on Cancer: Open Grid Services for Improving Medical Knowledge Discovery

The ACGT Project

- Goal: to support clinico-genomic trials
- Challenges
  - Large data sets – terabytes of data
  - Distributed data sets – multiple hospitals and organizations involved in a trial
  - Genomic data is *very* privacy-sensitive
  - High computational demands
  - Semantics
- Approach
  - Grid architecture for distributed data management and security
  - Ontologies for common semantics
  - R / Bioconductor as workhorse for analysis of genomic data

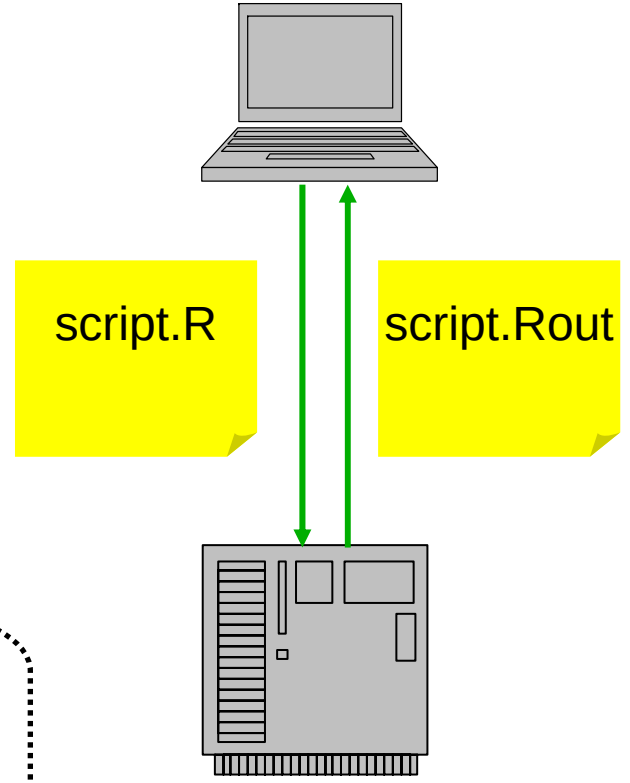


Real-Life Example: Analysis of Clinico-Genomic Data



Scenarios of Distributed Computing (1/5)

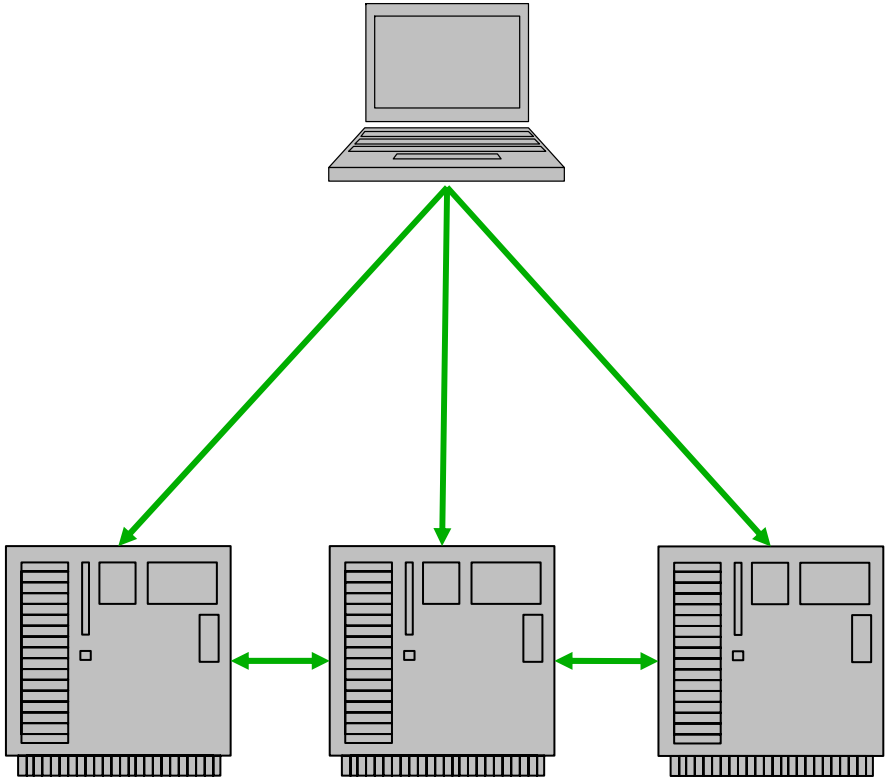
- Using external resources: run big R script on some server instead of own laptop
- Trivial solution
  - Remote login, or
  - copy R script to server, run, copy results back
- Problems
  - Interactive mode problematic
  - Need to tranfer scripts
  - Need to coordinate with other people on server



Quick tipp:
if this is all you need, on a linux /unix machine the commands `ssh / scp`, `nohup` and `screen` may already do the trick.

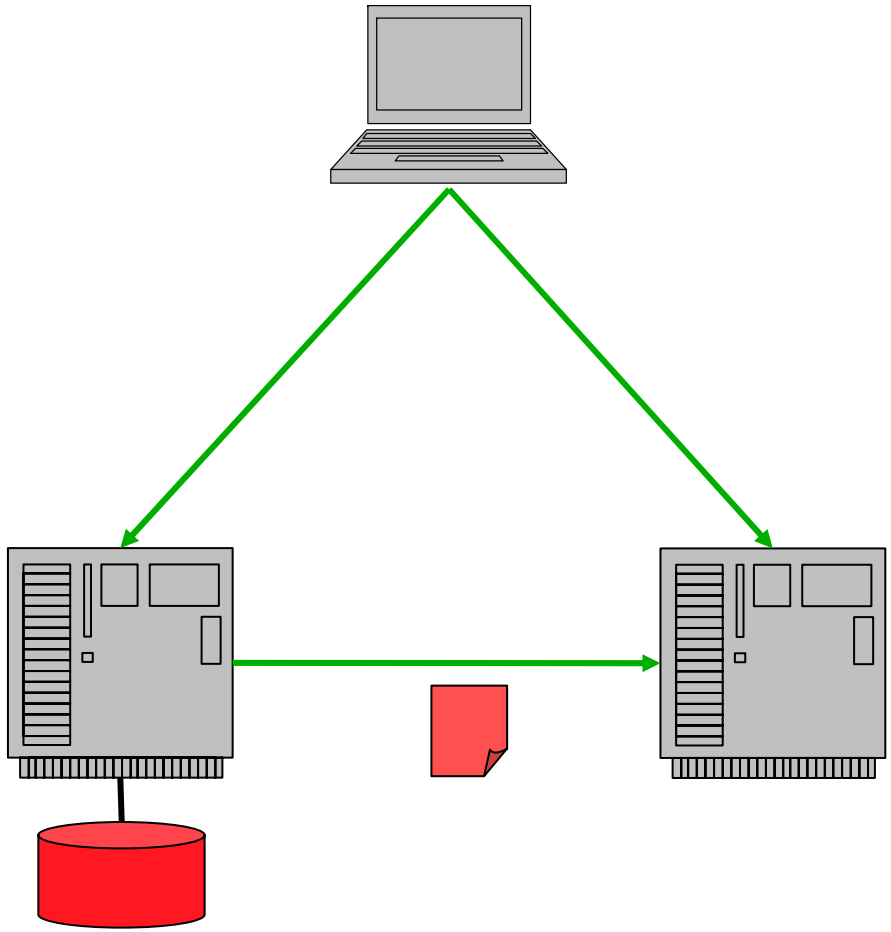
Scenarios of Distributed Computing (2/5)

- Using multiple resources
  - Run n R scripts on n computers - “embarrassingly easy parallelization”, e.g. cross-validation, parameter-tuning, …
  - Run n distinct parts of one R script on n computers – hard
- Could start all parts by hand, but
  - How to coordinate all servers?
  - Need to generate n scripts
  - Need to integrate results by hand



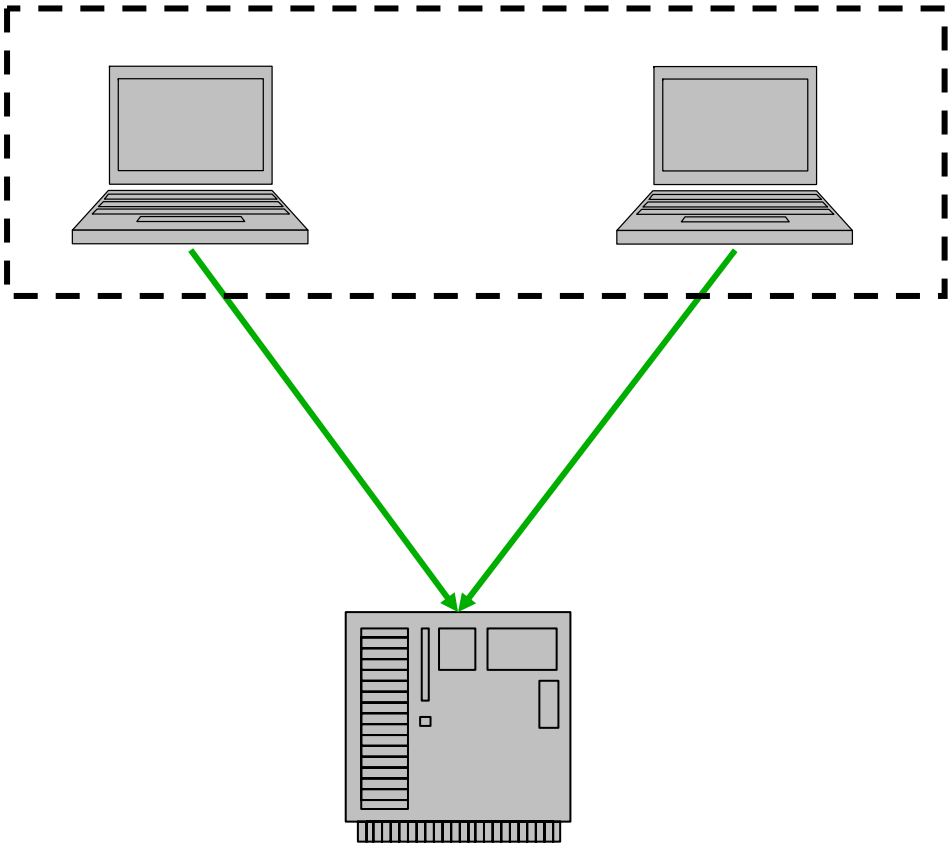
Scenarios of Distributed Computing (3/5)

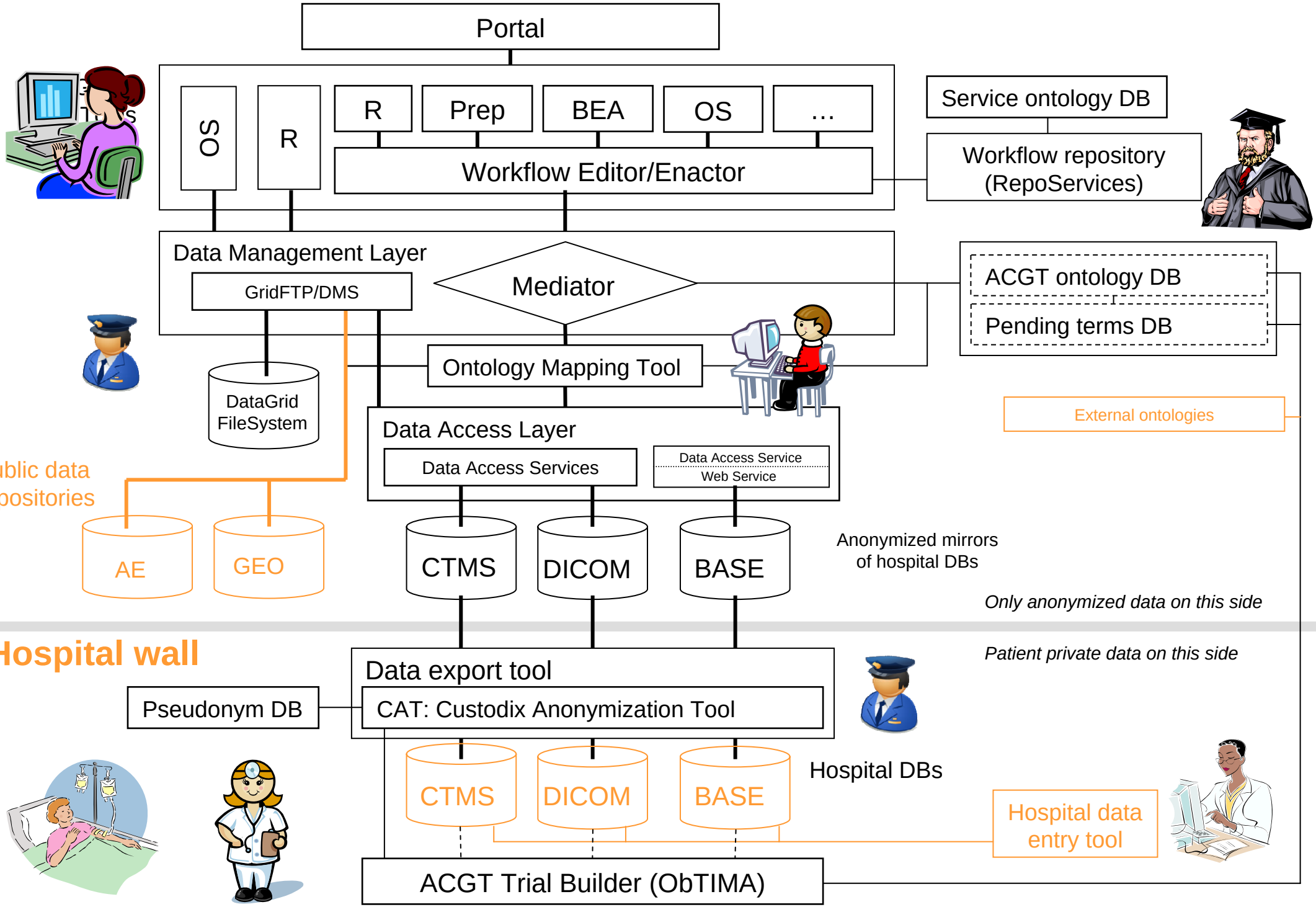
- Using distributed data
  - Access remote data sources
  - Shipment of algorithms – bring R script to the data, not vice versa
  - Access multiple data sources
- Security gets very tricky here



Scenarios of Distributed Computing (4/5)

- Multiple-User Collaboration
  - Exchange data with colleagues
  - Exchange code with colleagues




Portal
OS
R
R
Prep
BEA
OS
…
Workflow Editor/Enactor
Service ontology DB
Workflow repository
(RepoServices)
Data Management Layer
GridFTP/DMS
Mediator
ACGT ontology DB
Pending terms DB
Ontology Mapping Tool
DataGrid
FileSystem
External ontologies
Data Access Layer
Data Access Services
Data Access Service
Web Service
Public data
repositories
AE
GEO
CTMS
DICOM
BASE
Anonymized mirrors
of hospital DBs
Only anonymized data on this side
Hospital wall
Patient private data on this side
Data export tool
Pseudonym DB
CAT: Custodix Anonymization Tool
CTMS
DICOM
BASE
Hospital DBs
Hospital data
entry tool
ACGT Trial Builder (ObTIMA)

Scenarios of Distributed Computing

- Using external resources
  - Run big R script on some server instead of own laptop
- Using multiple resources
  - Run n R scripts on n computers - “embarrassingly easy parallelization”, e.g. cross-validation, parameter-tuning, …
  - Run n distinct parts of one R script on n computers Using distributed data
- Access multiple data sources
  - Shipment of algorithms – bring R script to the data, not vice versa
- Multiple-User collaboration
  - Exchange data and code with colleagues
- Setting up a large system
  - R as small part of the architecture

Outline & Timetable

- 9:00 – 9:15 Introduction
- 9:15 – 10:15 R in the context of Distributed Systems
- 10:15 – 10:30 GridR Installation
- 10:30 – 11:00 coffee break
- 11:00 – 11:45 The GridR package
- 11:45 – 12:15 Real world examples
- 12:15 – 12:30 Discussion

Part I - R in the context of distributed systems

- **Overview on distributed systems**
- Explicit message passing
  - R-packages Rpvm, Rmpi
- Shared workspaces
  - R-package nsw
- Remote object invocations and web-services
  - R-packages Rserve, Rweb, Snow
- Cluster- and grid-based computing
  - Condor and Globus toolkit, R-package gridR

Distributed Systems

A distributed system is a collection of autonomous computing nodes connected by a network that work on a common task.

- Advantages
  - Inherently redundant, potentially fault tolerant
  - Scalable, cheap, powerful
  - Application driven distribution: Computing nodes and data reside in different locations
- Complexity
  - Exponential number of failure modes
  - Concurrency, no global state or memory
  - Heterogenous, dynamic
  - Topology of the network influences programming
  - Programming model must integrate communication

→ models, algorithms and system software provide abstractions that „hide“ the complexity

Characteristics of distributed systems

- Partial Failure Property
  - The failure of any component of the system does not lead to the failure of the complete system
- Leslie Lamport’s definition:“A distributed system is one in which the failure of a computer you didn't even know existed can render your own computer unusable.” as quoted in CACM, June 1992
- System software introduces “transparancy”
  - Failure transparency (partial failures are hidden)
  - Location transparency (resources can be accessed as if they were local)
  - Concurrency transparency (resources can be accessed as if there was only one user)
  - …
- System models define assumptions, under which distributed algorithms operate

Synchronous System Model

- A (potentially unknown) number of processes p,q, …
- Communicate by with send(msg, p) and recv(msg, p) primitives
- Fixed message latencies
- No message loss
- No process failure
- Process work in rounds in lock step
  - Receive messages
  - Do work
  - Send messages

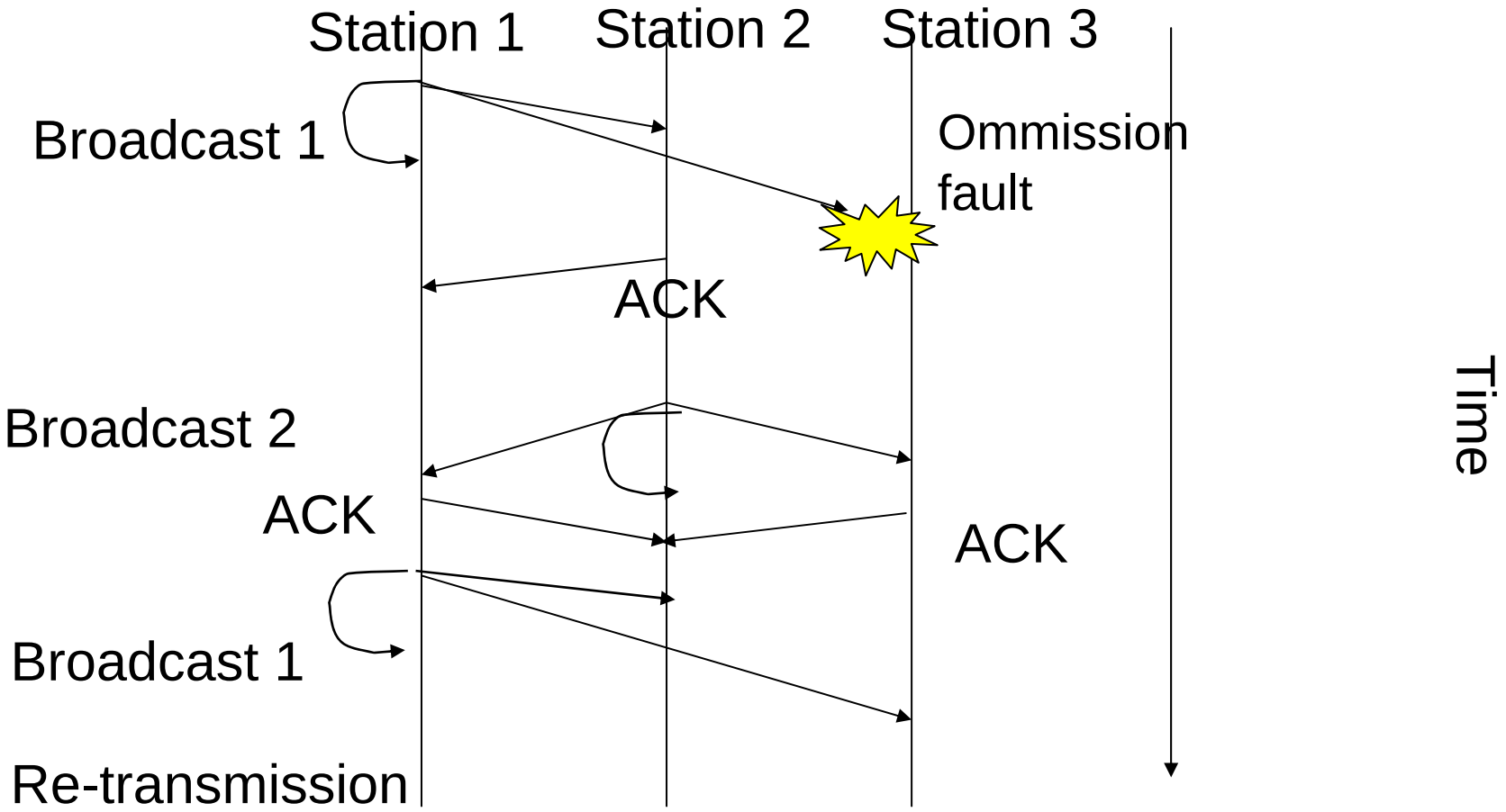
Asynchronous System Model

- A (potentially unknown) number of processes p,q, …
- Communicate by with send(msg, p) and recv(msg, p) primitives
- Unknown message latencies
- No message loss
- No process failure
- Process work in rounds, but not synchronized
  - Receive messages
  - Do work
  - Send messages

Abstract Model for Group Communication

Property	Meaning
Broadcast	Messages are sent to a group of processes. All processes in the group receive the same messages
Ordering	All messages are received in the same order in all processes
...	

Interference of group communication properties



➔ **Station 2: BC 1 before BC 2, Station 3: BC 2 before BC 1**

➔ **ordering is violated, more protocol mechanisms are needed**

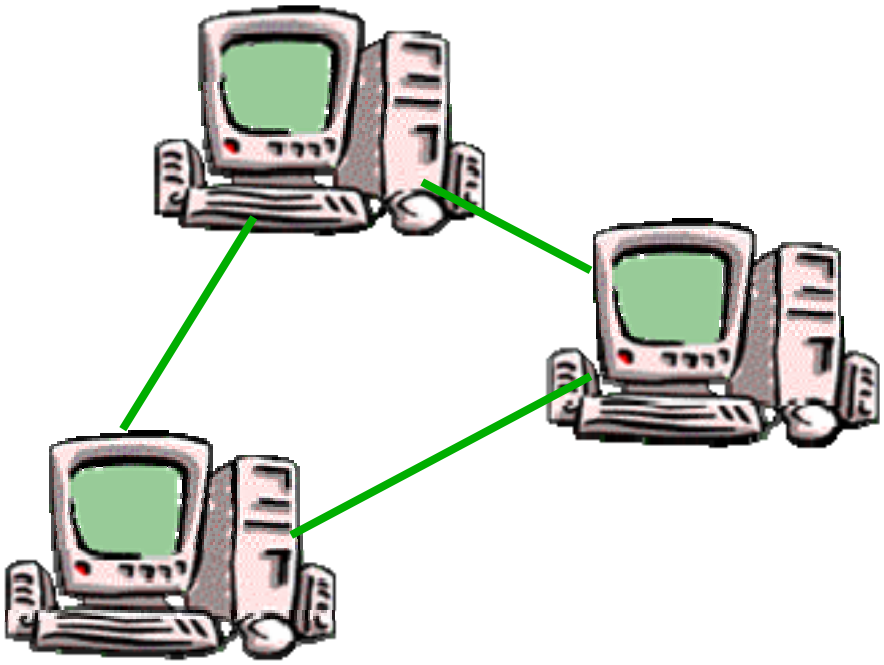
Summary

- Distributed systems offer scalable CPU and storage facilities
- Distributed systems have no shared memory and are implemented on top inherently unreliable communication
- Abstract models of distributed systems provide easy to use programming paradigms
- These models are implemented by network communication protocols, operating systems, middleware and language libraries
  - message passing
  - shared workspaces
  - remote object invocations

Part I - R in the context of distributed systems

- Overview on distributed systems
- Explicit message passing
  - Socket connections in R
  - R-packages Rpvm, Rmpi
- Shared workspaces
  - R-package nsw
- Remote object invocations and web-services
  - R-packages Rserve, Rweb, Snow
- Cluster- and grid-based computing
  - Condor and Globus toolkit, R-package gridR

Distributed systems with explicit message passing

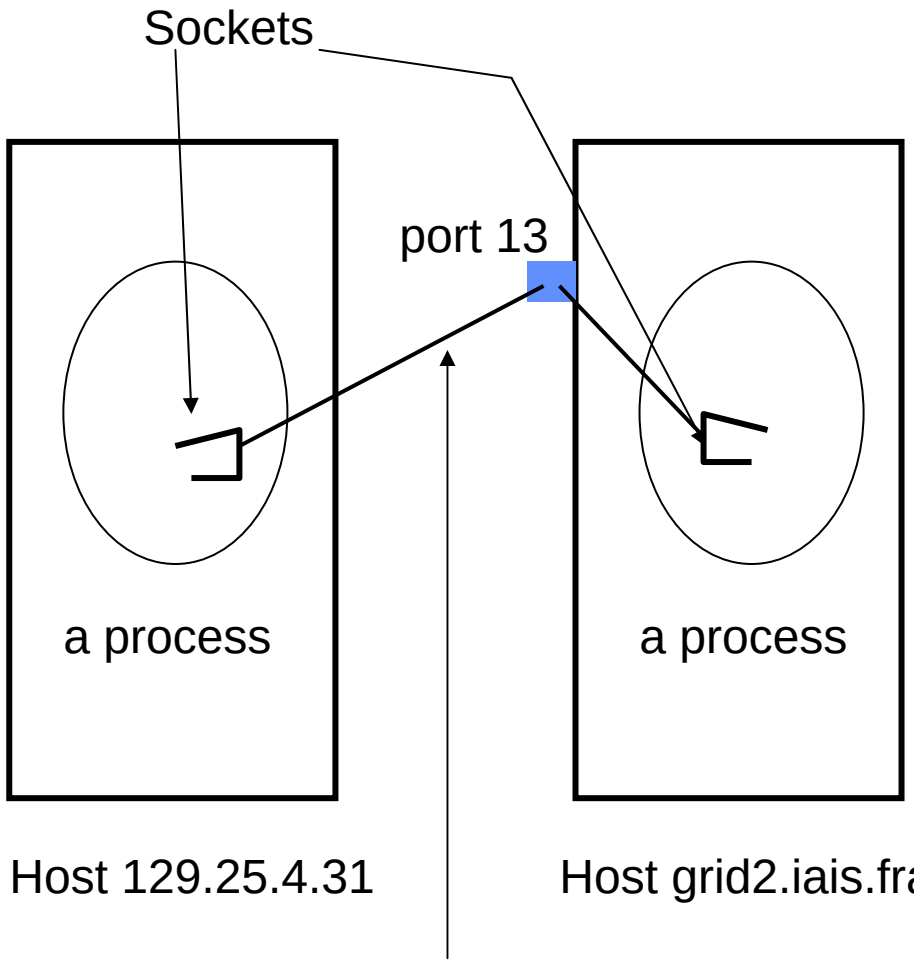


Things that matter

- Processes
- Tcp/IP Messages
- Host Names/Adresses
- Port Numbers
- ...

➔Operating system provides primitives for sending and receiving messages between nodes

Socket connections as messaging abstraction



- Unix based stream abstraction
- Sockets are 1:1 communication end-points
- Networks connections can be read/written like files
- Host names/adresses identify computing nodes
- Port numbers identify connections
- ...

Socket connections in R (from R man pages)

```
## Not run:
## two R processes communicating via non-blocking sockets

# R process 1
con1 <- socketConnection(port = 6011, server=TRUE)
writeLines(LETTERS, con1)
close(con1)

# R process 2
con2 <- socketConnection(Sys.info()["nodename"], port = 6011)
# as non-blocking, may need to loop for input
readLines(con2)
while(isIncomplete(con2)) {Sys.sleep(1); readLines(con2)}
  close(con2)   ## End(Not run)
```

Evaluation of socket communication

- Good
  - Very flexible, can use **cat** and **scan** to transmit any R object between different processes
  - Supports text transmissions (Readlines, Writelines)
  - No restrictions on when to communicate
- Bad
  - Processes must be started/stopped manually
  - Addressing with hostnames and port numbers is very tedious
  - no group communication
  - program structure hard to understand, mix of computation and communication

R packages for explicit message passing

- Approved standard interfaces for message passing for parallel computations
- Different language bindings: Fortran, C, Phython, R, ...
- PVM: Parallel Virtual Machine (http://www.csm.ornl.gov/pvm/)
- MPI: Message Passing Interface (http://www-unix.mcs.anl.gov/mpi/)
- Quite similar approaches, differences in details, CRAN recommends MPI

- Packages can be mostly downloaded from CRAN: http://ftp5.gwdg.de/pub/misc/cran/web/packages/
- Rpvm: interface to PVM
  - Requires installation of PVM package on the system (See R package README, complex setup for windows: http://www.csm.ornl.gov/pvm/NTport.html)
- Rmpi: interface to MPI (http://www.stats.uwo.ca/faculty/yu/Rmpi/)
  - Requires the installation of an MPI package on the system (see R package README, e.g. MPICH2 on windows: http://www.mcs.anl.gov/research/projects/mpich2/)
  - Rmpi tutorial (including examples): http://ace.acadiau.ca/math/ACMMaC/Rmpi/index.html

MPI Basic Features

- Dynamic process management (usually, a master task spawns other, slave tasks)
- Task IDs are used for addressing
- Point-to-point communication:
  - a task can send a message to another task
  - a task can receive a message from another task
- Task groups: Tasks can join/leave groups
  - Group communication: a task can send a message to a group
  - Synchronization: a task can receive messages from all group members

Rmpi basic functions (see http://ace.acadiau.ca/math/ACMMaC/Rmpi/methods.html)

- mpi.spawn.Rslaves([nslaves=#])
  - This function spawns a number of slave processes to perform work.
  - Slaves are numbered 1..n, the master has number 0
- mpi.send.Robj(object,destnumber,tag)
  - Send an R object to the slave process destnumber.
  - This type of send must be processed by a call to mpi.recv.Robj in the receiving process.
- object <- mpi.recv.Robj(mpi.any.source(),mpi.any.tag())
  - Receive an R object and assign it to “object”.
  - Blocks until an object is received.
- mpi.bcast.Robj2slave(object)
  - This function broadcasts/pushes an object, or even a function, out to all the slaves.
- results <- mpi.remote.exec("R code")
  - This causes all slave processes to run the R code, and return the result.

Rmpi example (see http://ace.acadiau.ca/math/acmmac/Rmpi/brute_force.R)

```
# Initialize the Rmpi environment
library("Rmpi")
# It's 10-fold cross-validation.
# we spawn 10 slaves
mpi.spawn.Rslaves(nslaves=10)
...
# Function to be executed by the slaves
foldslave <- function() {
    result <- # work on “thedata”
   ...
    }

# We're in the parent.
# Create data set
thedata <- data.frame(y=y,x=x)
```

```
# Now, send the data to the children
mpi.bcast.Robj2slave(thedata)
...

# Send the function to the children
mpi.bcast.Robj2slave(foldslave)

# Call the function in all the children,
# and collect the results
rssresult <- mpi.remote.exec(foldslave())

# plot the results
plot(apply(rssresult,1,mean))

# close slaves and exit
mpi.close.Rslaves()
mpi.quit(save = "no")
```

Evaluation of explicit message passing with pvm, mpi

- Good
  - easy addressing of processes
  - point-to-point and group operations
  - No restrictions on when to communicate
- Bad
  - requires installation of pvm/mpi runtime systems
  - message passing must be programmed explicitly
  - program structure hard to understand, mix of computation and communication

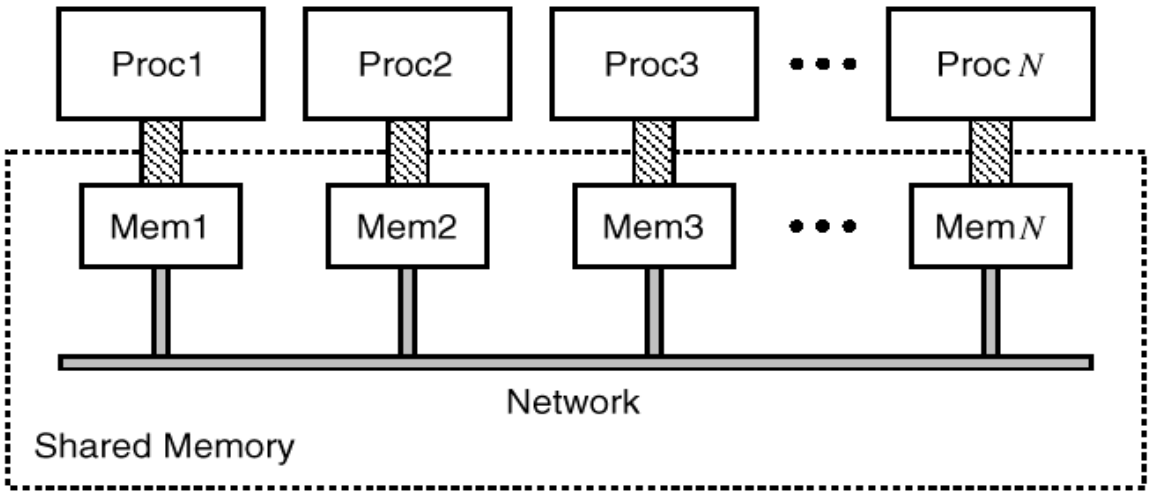
Part I - R in the context of distributed systems

- Overview on distributed systems
- Explicit message passing
  - R-packages Rpvm, Rmpi
- Shared workspaces
  - Shared files and databases
  - R-package nws
- Remote object invocations and web-services
  - R-packages Rserve, Rweb, Snow
- Cluster- and grid-based computing
  - Condor and Globus toolkit, R-package gridR

Virtual global memory

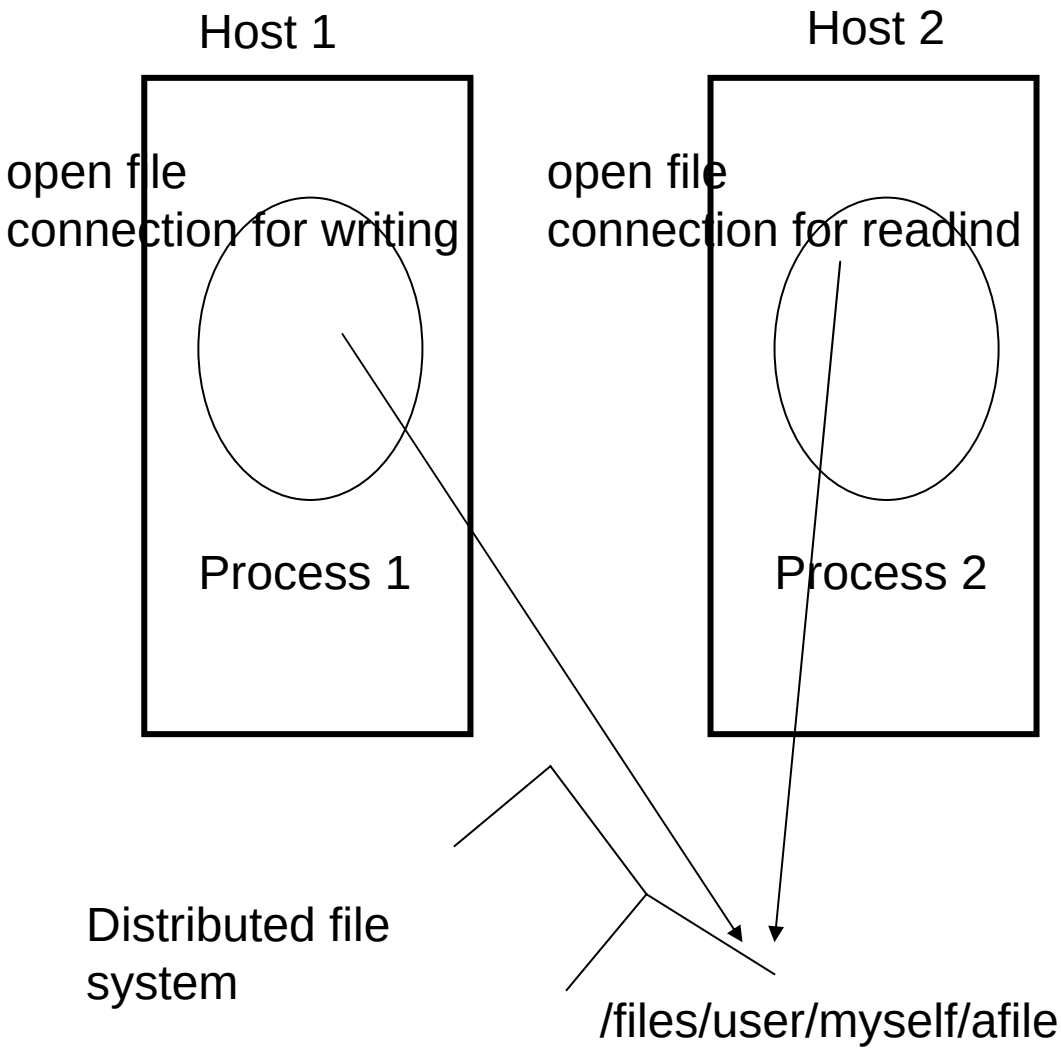
- Idea: the system software provides the illusion that the nodes of a distributed system can access (read/write) a globally shared memory system
- Various flavors
  - DSM - Distributed Shared Memory
  - TupleSpace (Linda, JavaSpace)
  - Blackboard systems (Distributed AI)
  - Process Images (Industrial automation)
- Processes work, execute without explicit message passing, but exchange data and synchronize themselves via the global memory

A virtual shared memory system



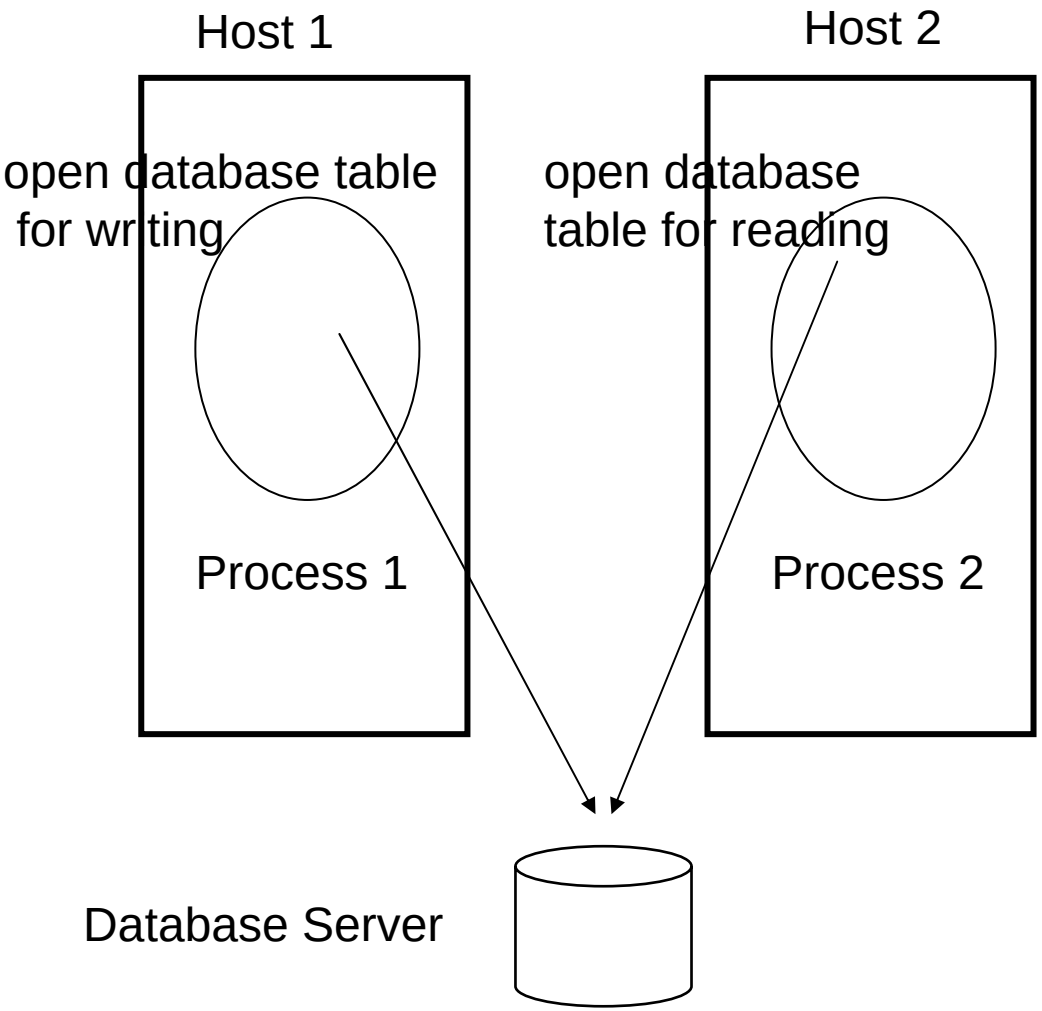
Each processor is provided with the illusion to read/write all memory locations in the network

Using a distributed file system as global memory



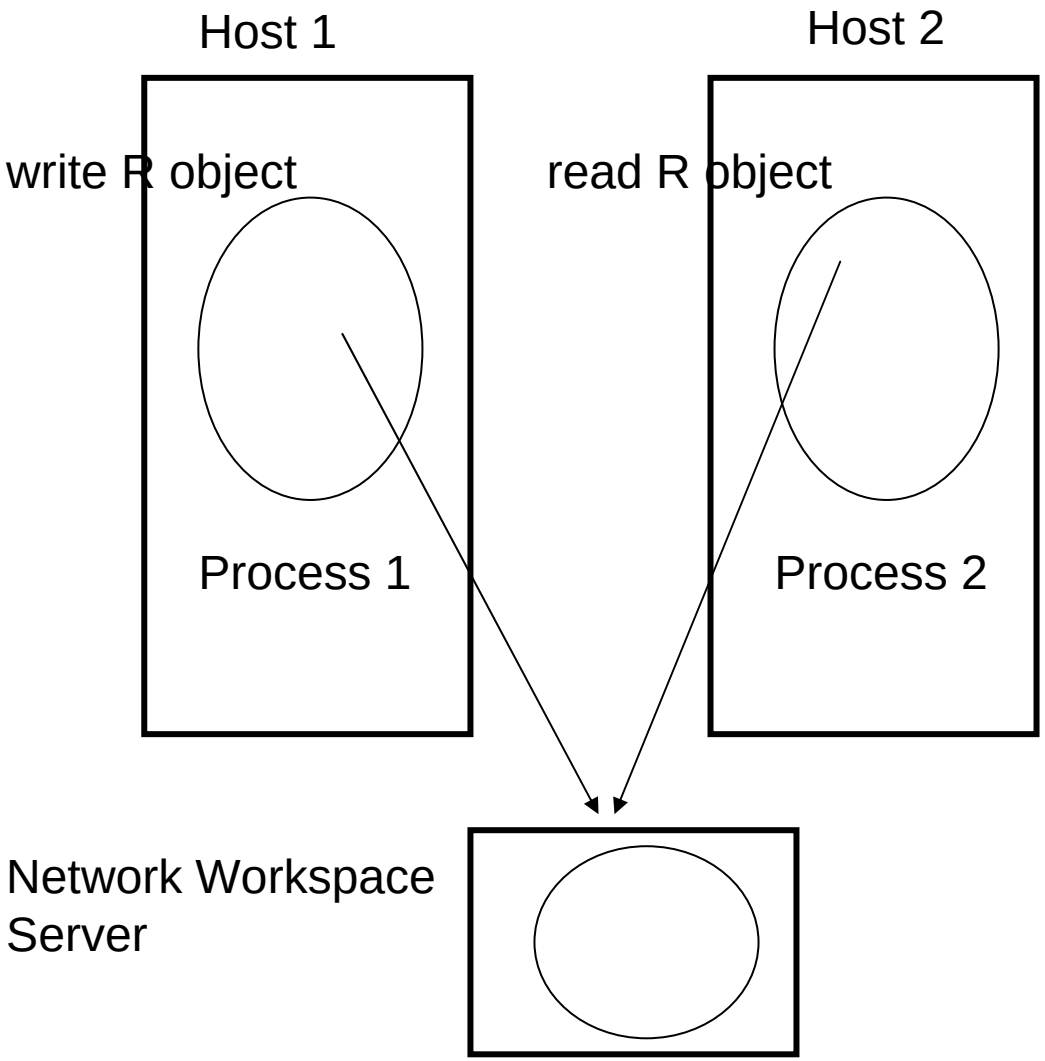
- Host 1 and host 2 mount the same shared file system from a file server
- Process 1 opens a connection for writing and writes data or R objects to the file
- Process 2 opens the file for reading and reads the output of process 1
- Example (R manual)
  - con <- file(" /files/user/myself/afile", "w") # open an output file connection
  - cat("TITLE extra line", "2 3 5 7", "", "11 13 17", file = con, sep = "\n")
  - close(con)

Using a central database system as global memory



- Host 1 and host 2 access the same database server with RODBC
- http://cran.r-project.org/web/packages/RODBC/index.html
- opening connection
  - channel = odbcConnect(dsn, uid = "", pwd = "", ...)
- wiriting table
  - sqlSave(data, sqltable)
- reading table
  - sqlFetch(channel, sqltable)

Using a network workspace as global memory



- Host 1 and host 2 access the same network workspace server with NWS
- NWS: NetWorkSpace (http://nws-r.sourceforge.net/)
- Examples: http://nws-r.sourceforge.net/doc/nwsR-1.5.0.pdf
- Can write and read individual R objects into / from the shared workspace
- blocking and non-blocking read
- write/read several values sequentially on the same R object (FIFO)

NWS Basic Operations

- ws = netWorkSpace(‘R space’)
  - opens a connection to the shared workspace
- nwsStore(ws, ‘x’, 1)
  - creates an R object named x and store the value 1 in it
- val = nwsFind(ws, ‘x’)
  - reads the value of the R object x and stores it in val (here 1)
  - blocks until x exists
- val = nwsFindTry(ws, ‘x’) - non-blocking version, returns NULL if x is not found
- val = nwsFetch(ws, ‘x’)
  - same as nwsFind, but removes the value of x afterwards
- val = nwsFetchTry(ws, ‘x’) - non-blocking version,

NWS Operations with sequential values (http://nws-r.sourceforge.net/doc/nwsR-1.5.0.pdf)

- > n = c(16, 19, 25, 22)
- > for (x in n) {
- + nwsStore(ws, 'biff', x)
- + }
  - writes the values 16,19,25,22 sequentially into the R object biff
- > n = vector()
- > i = 1
- > while (!is.null(tmp <- nwsFetchTry(ws, 'biff') {
- + n[i] = tmp + i = i + 1
- + }
- > n
- [1] 16 19 25 22
  - reads these values sequentially

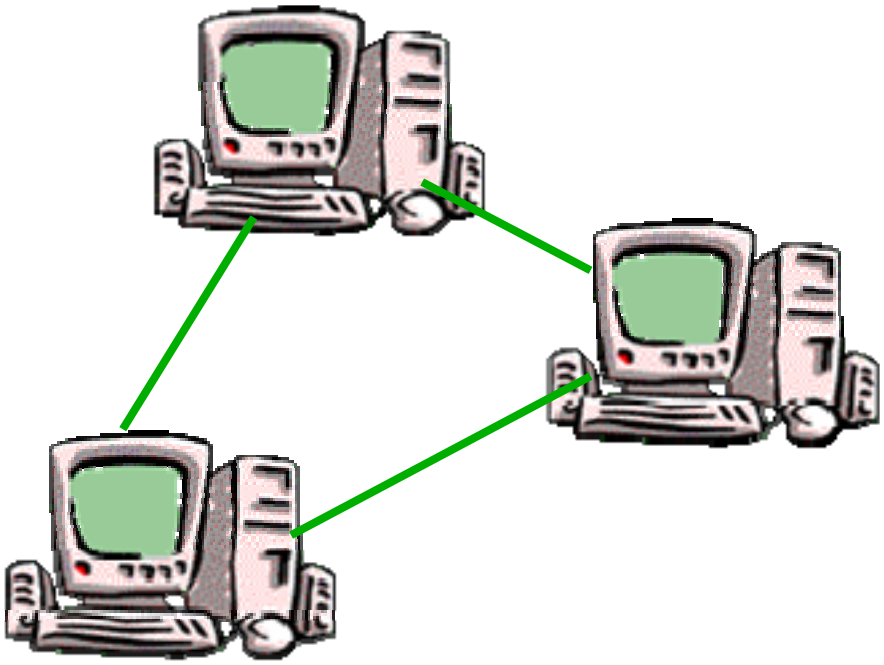
Evaluation of global memory

- Good
  - easy to use
  - good for sharing data
  - synchronization possible (easy with NWS)
- Bad
  - file/database access is slow - only useful for large data
  - synchronization of tasks is difficult with files/database
  - no direct support for parallel executions
    (note: there is an extension of NWS called sleigh for parallel jobs)

Part I - R in the context of distributed systems

- Overview on distributed systems
- Explicit message passing
  - R-packages Rpvm, Rmpi,
- Shared workspaces
  - R-package nws
- Remote object invocations and web-services
  - R-packages Rserve, Rweb, Snow
- Cluster- and grid-based computing
  - Condor and Globus toolkit, GridR

A Distributed System

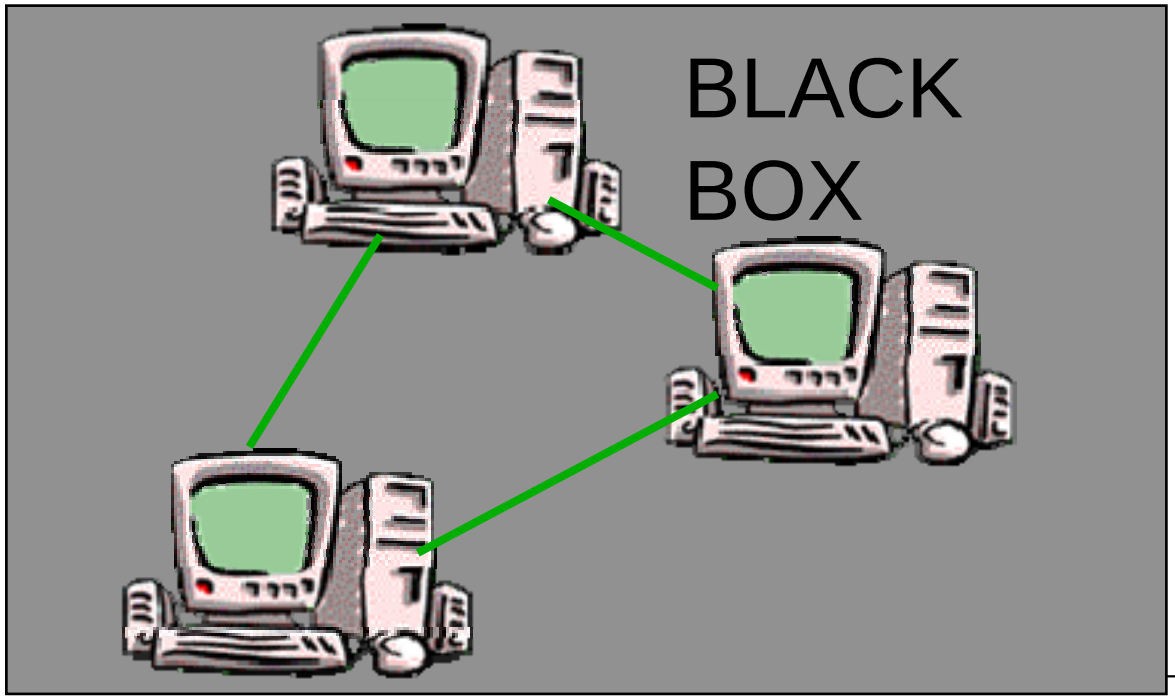


Things that matter

- Processes
- Tcp/IP Messages
- Host Names/Adresses
- Port Numbers
- ...

➔The application developer is disturbed by many nasty details

A Distributed object-oriented system



Rising the level of abstraction!

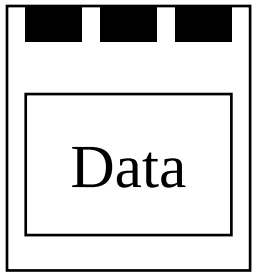
Things that matter

- Objects
- Methods
- Object Invocations

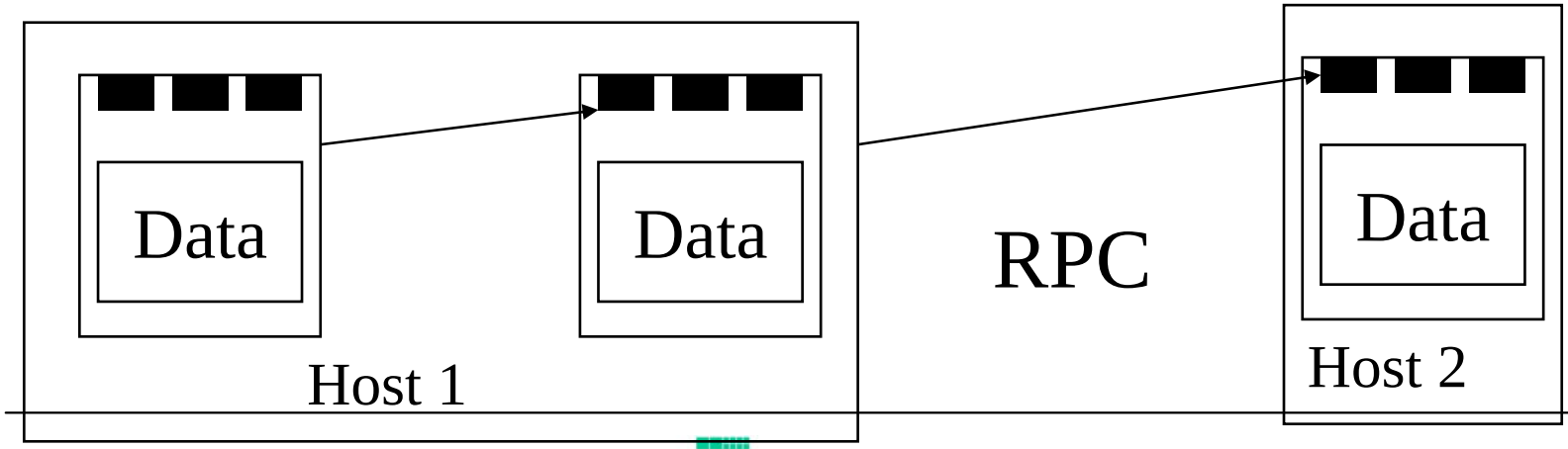
➔The application developer is relieved from many nasty details

Remote Procedure Calls (RPC)

- Applications are structured in objects
- An object encapsulates data which can be accessed by a set of operations (methods, procedures, services)
- Objects interact with procedure calls, these can be local or remote



An object



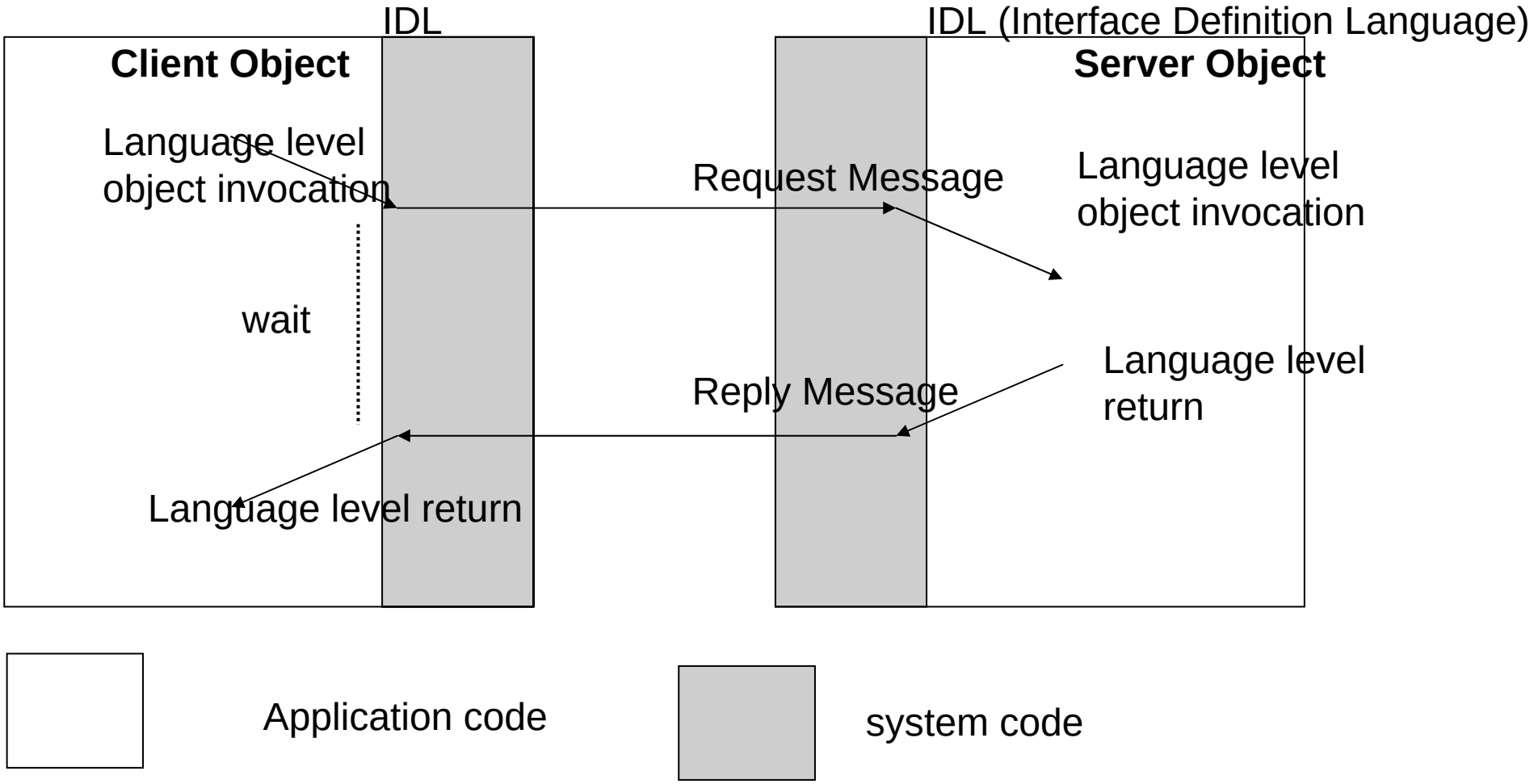
Advantages of the Remote Procedure Call

- Location transparency: a remote object is invoked in the same way as a local object. The location of the remote object is hidden to the application.

- Language transparency: The objects can be programmed in different programming languages, but still can invoke each other (even across different operating systems)

- Network transparency: All parameter/results (data) of a procedure called are serialized and sent automatically over the network

Application vs. system code in the RPC

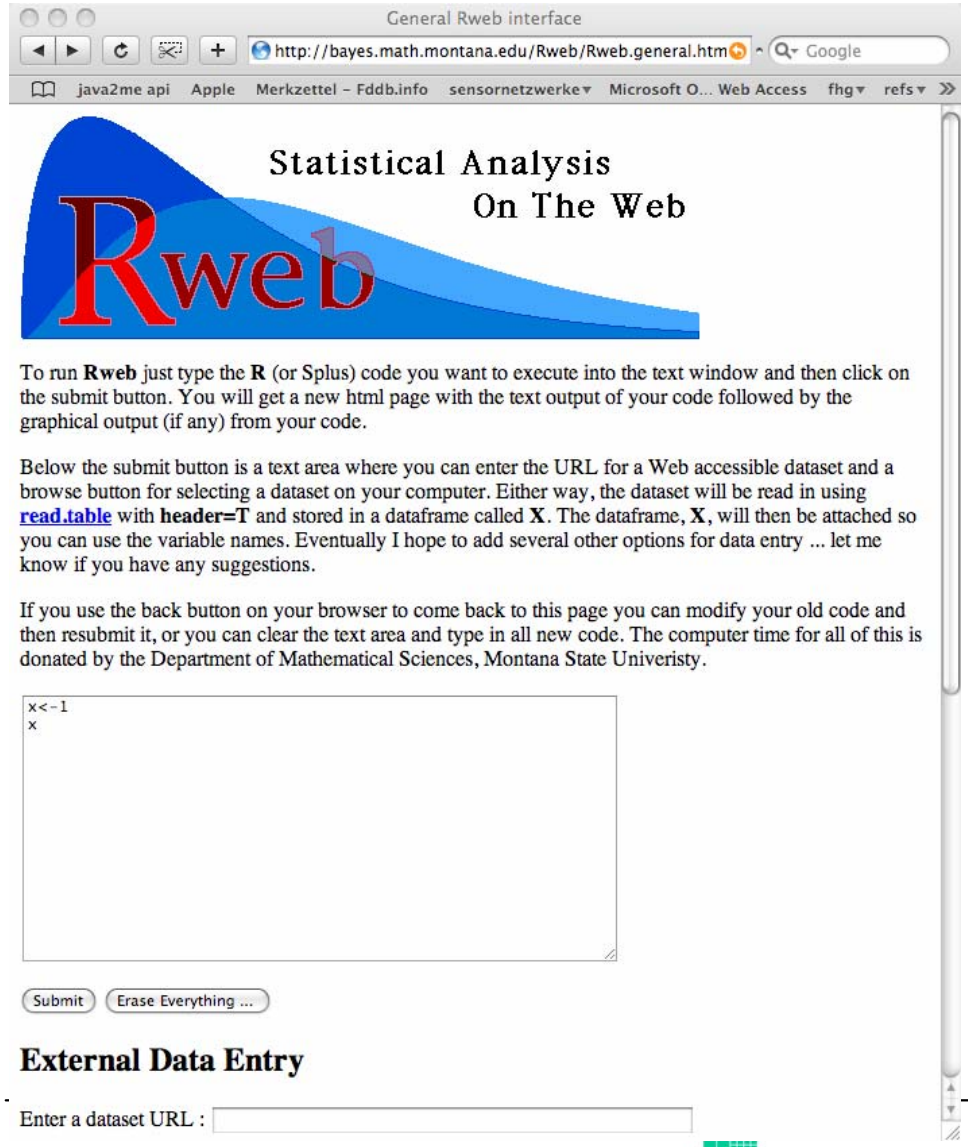


Examples for the remote procedure call modell

- CORBA - Common Object Request Broker Architecture: the first object oriented RPC standard, defined by the OMG in the 90‘s (IBM, Sun, HP, Microsoft, GMD, ... Über 300 Mitglieder)
- DCOM - form Microsoft
- Java - Single-Language System from Sun
- .NET - successor of DCOM
- SOAP - Simple Object Access Protocol for the Word Wide Web
- Web-Services - Remote object invocations over SOAP using furhter Web-Standards

- the most dominant and successful model in distributed systems - what about R ?

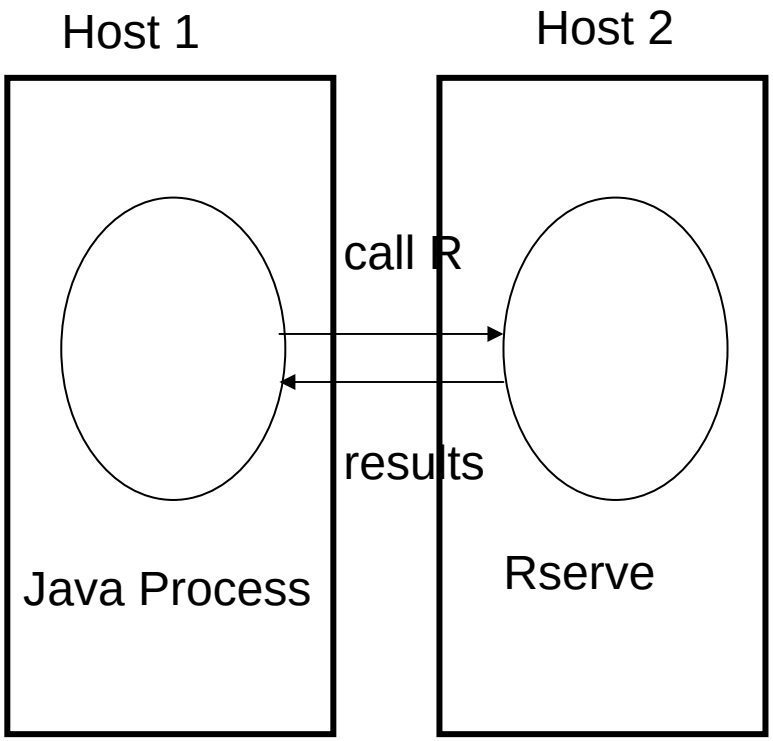
Rweb



- Rweb: web based interface to R (http://www.math.montana.edu/Rweb/)
- Allows to connect to a remote R session via a web-browser
- Similar to a remote terminal functionality
- useful for trying R without installing it
- not useful (or intended) for distributed computations

Fraunhofer IAIS Institut Intelligente Analyse- und Informationssysteme

Rserve

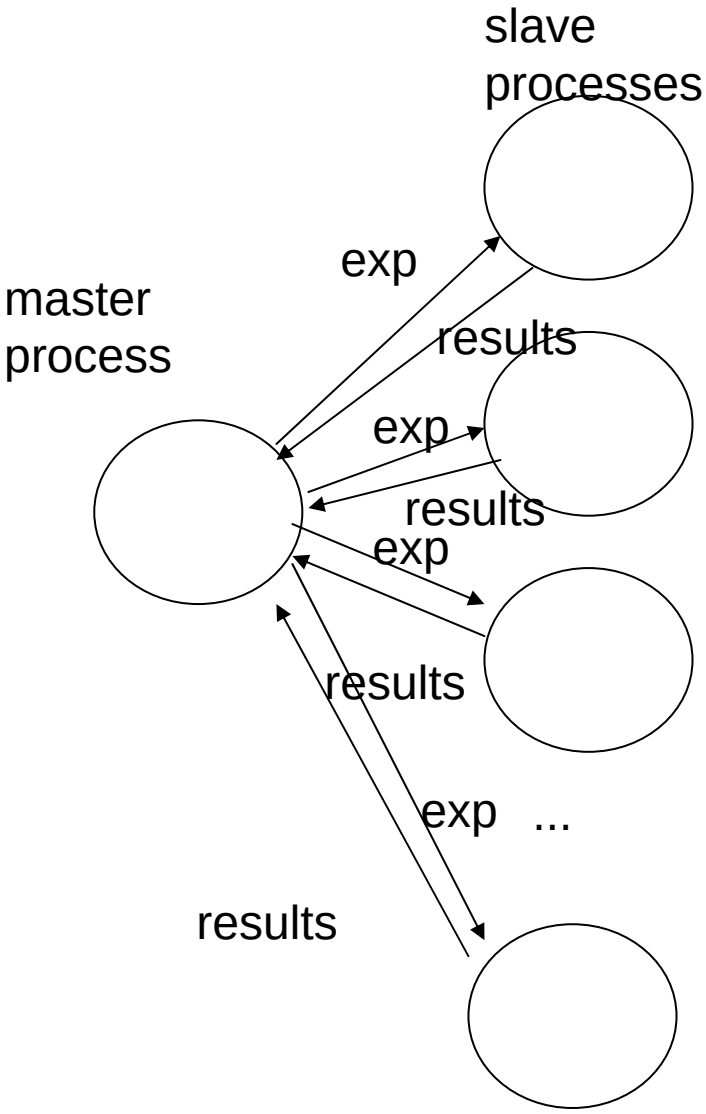


- Rserve: socket-based R server (http://rosuda.org/Rserve/)
- A server runs R, a client can access the server via a client library
- Client libraries available for Java and C++ (and R)
- Functionality is similar to loading R as library into a Java or C++ program, but the R code is actually executed on a remote machine
- useful to integrate R-statistics into Java or C++
- not useful (or intended) for distributed computations

The R Snow package

- Simple Network of Workstations
- Programming model similar to RPC, but supports parallelism
- Runs on top of PVM, MPI or directly on socket connections
- (http://www.sfu.ca/~sblay/R/snow.html)
- Tutorial on http://www.stat.uiowa.edu/~luke/talks/uiowa03.pdf

Snow Basic Functions



- cl <- makeCluster(10)
  - creates 10 slave processes
- clusterCall(cl, exp, ...)
  - evaluates exp on all slaves in cl
- clusterApply(cl, list, func, ...)
  - applies function func to list, one element of the list is processed on one slave

- Snow example follows in comparison with GridR !

SNOW examples

- Examples (from http://www.sfu.ca/~sblay/R/snow.html and R Help for package snow)
  - cl <- makeCluster(c("localhost","localhost"), type = "SOCK")
  - clusterApply(cl, 1:2, get("+"), 3)
  - stopCluster(cl)
  - Output:
    [[1]]
    [1] 4
    [[2]]
    [1] 5

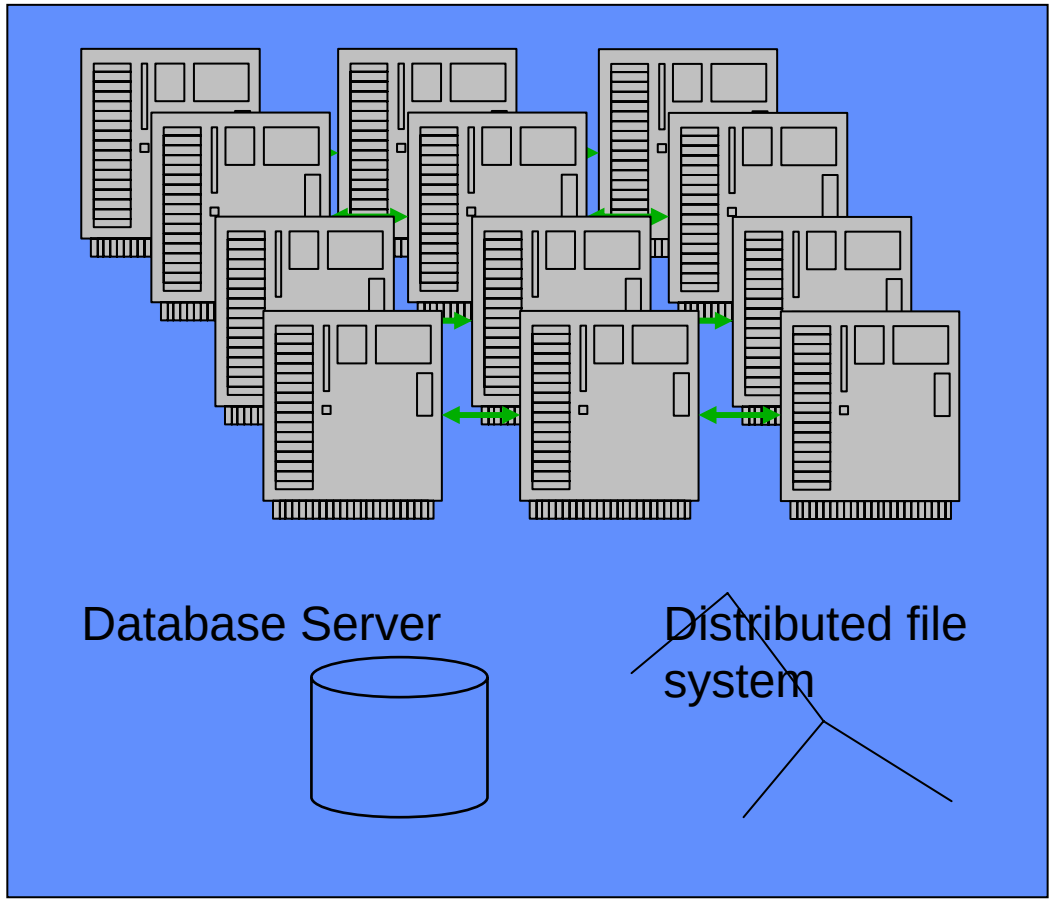
Evaluation of remote procedure call

- Good
  - extremely easy to use
  - widely accepted for distributed systems
- Bad
  - no direct support for parallel executions
  - except for SNOW
  - but SNOW does not integrate cluster and grid management systems

Part I - R in the context of distributed systems

- Overview on distributed systems
- Explicit message passing
  - R-packages Rpvm, Rmpi,
- Shared workspaces
  - R-package nws
- Remote object invocations and web-services
  - R-packages Rserve, Rweb, Snow
- Cluster- and grid-based computing
  - Condor and Globus toolkit, GridR

Cluster computing



- A cluster is a set of computers providing computing and storage facilities in a single network and a single administration domain
- Data is stored on database systems or distributed file systems in the same network
- Authentification and data access control is handled by the network administrator using standard operating system functionality
- A cluster management system handles job submissions and load balancing

Condor cluster management system

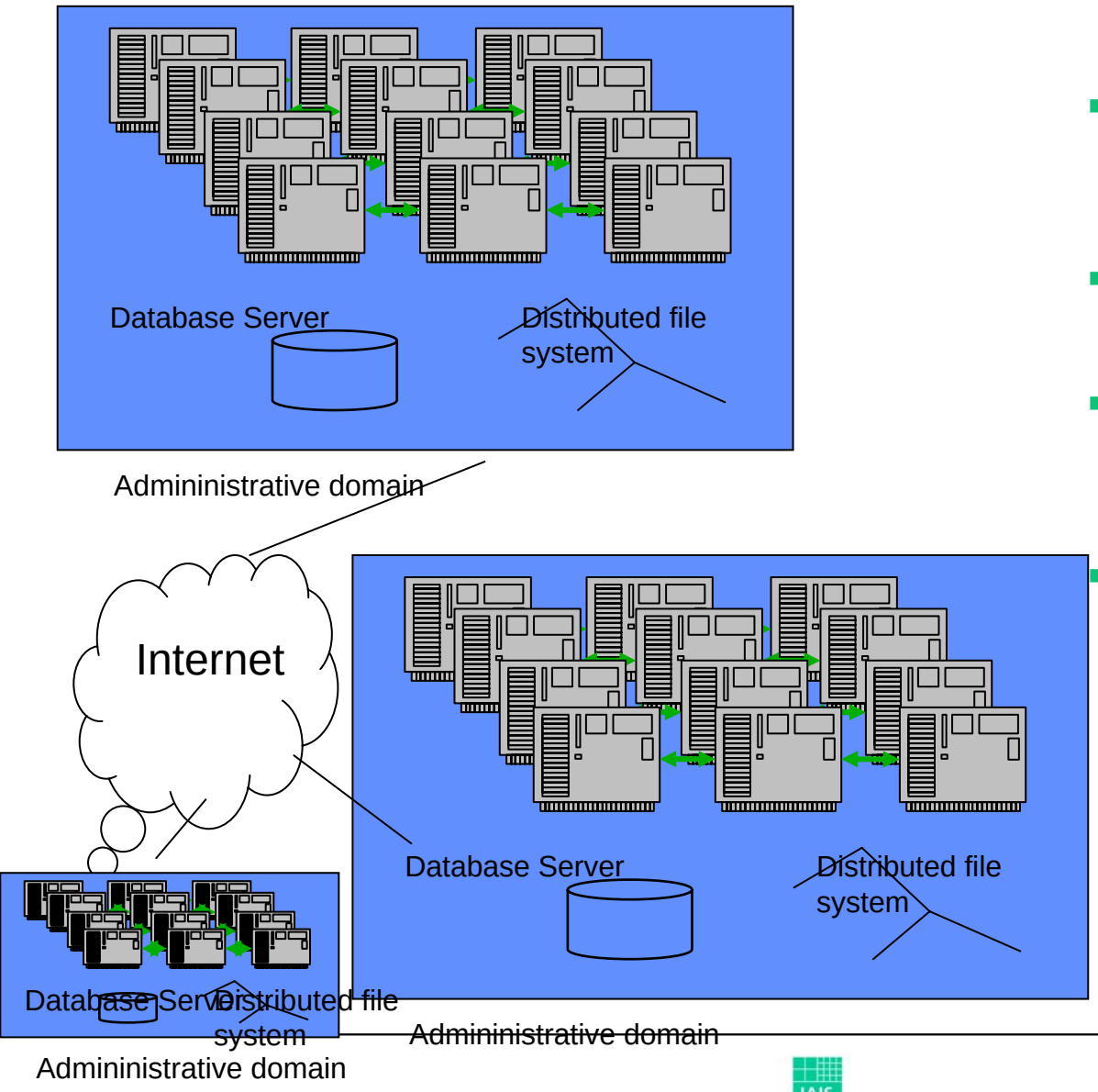
- http://www.cs.wisc.edu/condor/
- Prominent and free management system for Unix and Windows clusters
- Handles job submission with load balancing: selects automatically the node to which a job should be sent
- Job descriptions allow easy submission of parallel jobs

Condor job description

```
#####################
# simple condor job description
#####################

Executable     = /home/user/myApp.sh
Universe = vanilla
input    = test.data
output   = myApp.out
error    = myApp.error
Log      = myApp.log

Queue 3
```

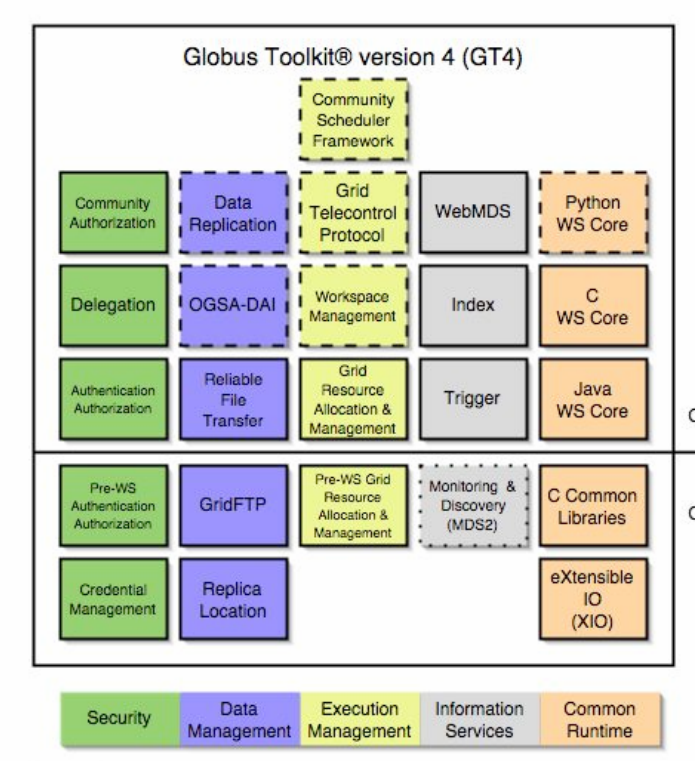
- Submit command: „condor_submit myApp.con“
- runs the script App.sh using input test.data 3 times in parallel on the cluster

Grid computing



- Grid computing provides parallel computing and collaboration across different organizational domains
- Data is stored and computing clusters reside in different administrative domains
- Authentification and data access control is handled according to unified internet/grid standards
- A grid management translates grid standards to the local administrative domain

Globus Toolkit (GT4) grid management system



- http://www.globus.org/toolkit/
- Prominent and free management system for Unix and Windows grid integration
- Handles across organizations over the internet
  - job submission
  - access to data including access control
  - authentification
- Job descriptions allow easy submission jobs

http://www.globus.org/toolkit/about.html

GT4 job description

```
<job>
   <executable>my_echo</executable>
   <directory>${GLOBUS_USER_HOME}</directory>
   <argument>Hello</argument>
   <argument>World!</argument>
   <stdout>${GLOBUS_USER_HOME}/stdout</stdout>
   <stderr>${GLOBUS_USER_HOME}/stderr</stderr>
   <fileStageIn>
      <transfer>
         <sourceUrl>gsiftp://job.submitting.host:2811/bin/echo</sourceUrl>
         <destinationUrl>file:///${GLOBUS_USER_HOME}/my_echo</destinationUrl>
      </transfer>
   </fileStageIn>
   <fileStageOut>
      <transfer>
         <sourceUrl>file:///${GLOBUS_USER_HOME}/stdout</sourceUrl>
         <destinationUrl>gsiftp://job.submitting.host:2811/tmp/stdout</destinationUrl>
      </transfer>
   </fileStageOut>
</job>
```

- Submit command: „globusrun-ws -submit f myJob.xml“
- runs the application my_echo on the remote machine

Part I - R in the context of distributed systems

- Overview on distributed systems
- Explicit message passing
  - R-packages Rpvm, Rmpi,
- Shared workspaces
  - R-package nws
- Remote object invocations and web-services
  - R-packages Rserve, Rweb, Snow
- Cluster- and grid-based computing
  - Condor and Globus toolkit, GridR

  - --> SUMMARY

Part I - Summary - Comparison of R packages

	Rpvm	Rmpi	nws	Rweb	Rserve	Snow	GridR
complex parallel algorithms	X	X					
easy parallelization			(X)			X	X
data sharing and collaboration			X				X
Grid integration		X					X

GridR installation

- Installation of GridR
  - Download from ACGT website(?) / Copy the GridR package
  - On Linux: run „R CMD install GridR.tgz“
  - On Windows: extract archive and run „R CMD install GridR“
  - you might need the Rtools (perl etc.): http://www.murdoch-sutherland.com/Rtools/index.html
  - For running the examples: Create a file named „.gridR.conf“ in /home/user/ (Linux) or C:\users\username\Documents (Windows) with the following content:

```
<GRIDR>
<LOCALTMPDIR>D:\\tmp</LOCALTMPDIR>
<SERVICE>local</SERVICE>
<SSHKEY>D:\\tmp</SSHKEY>
<NFSDIR>D:\\tmp\\share</NFSDIR>
</GRIDR>
```

- Installation of Snow
  - Install package with name „snow“ through the R package installer from CRAN

- <coffee break>

Part II - The GridR package

Outline & Timetable

- 9:00 – 9:15 Introduction
- 9:15 – 10:15 R in the Context of Distributed Systems
- 10:15 – 10:30 GridR Installation
- 10:30 – 11:00 Coffee Break
- 11:00 – 11:45 The GridR Package
- 11:45 – 12:15 Real-World Examples
- 12:15 – 12:30 Discussion

The GridR package - outline

- Motivation - The R user‘s point of view
- The GridR package development
- Functions, Configuration and internal details
- Parallelization of computational tasks – example scenes (GridR & Snow)
- Collaboration among multiple users

Scenarios of Distributed Computing

- Using external resources
  - Run big R script on some server instead of own laptop
- Using multiple resources
  - Run n R scripts on n computers - “embarrassingly easy parallelization”, e.g. cross-validation, parameter-tuning, …
  - Run n distinct parts of one R script on n computers - hard, not further discussed in this presentation
- Using distributed data
  - Access multiple data sources
  - Shipment of algorithms – bring R script to the data, not vice versa
- Multiple-User Collaboration
  - Exchange data and code with colleagues

Goal: Make this possible with GridR

GridR - Motivation

- Primary Goal: Usability
  - Parallel execution should not interfere with running R session
  - R user should not need to know details of distributed computation (software, standards, …)
  - Transparent execution of any R code
  - Support for all needs in a typical distributed computing scenario – here: analysis of clinico-genomic data
- Implementation Requirements
  - No modification of R itself
  - Independent of underlying distributed architecture

Motivation & process of development

- Our approach: Compile an R package for computation in distributed environments that is easy to use (from th R user‘s point of view)
- Imagine there is an R function that does some kind of big computation
- Process of thinking:
  - Execute the function in the R session on the local machine
    - Create a function process <- function (..) {..}
    - Call it from the command promt: >process(..)
  - Execute the function as own process on the local machine
    - Store the function into an R script file process.R
    - Launch an R process by calling System(„R CMD BATCH process.R“)
  - Execute the function (manually) on a remote machine (e.g. via ssh)
    - Copy the R script file to the remote machine
    - Launch an R process on the remote machine by calling System(„ssh remotemachine R CMD BATCH process.R“)

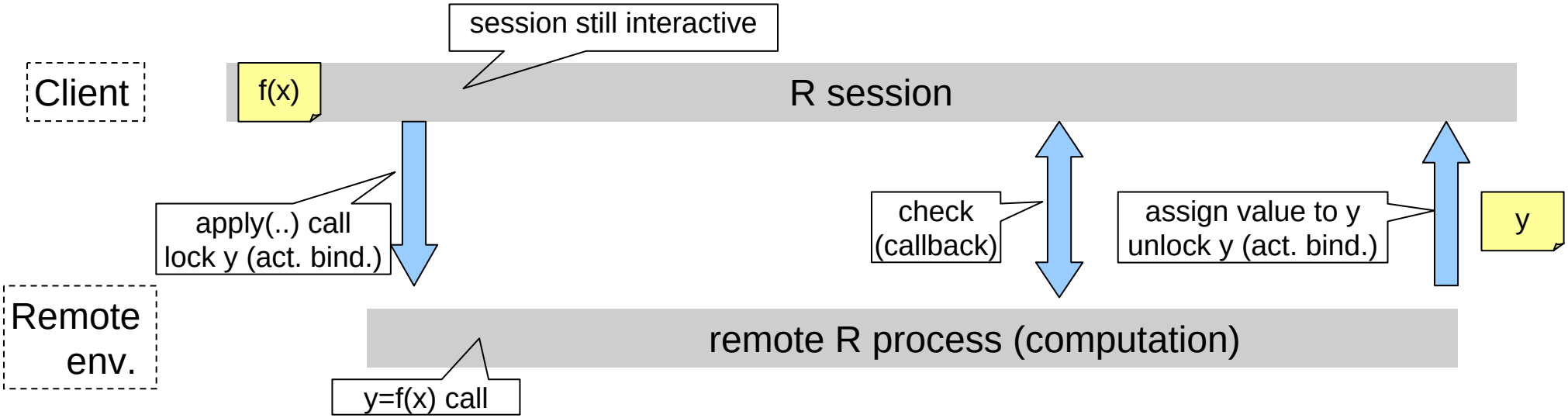
Motivation & process of development (cont.)

- Execute the function on a pool of remote machines (a Cluster, e.g. managed by Condor)
  - Create a job description for the cluster management system
  - Copy the R script file and the job description to the cluster master remote machine
  - Launch an R process as job on one of the remote machines by calling System(„ssh remotemachine condor_submit process.condor“)
- Execute the function on a pool of remote machines that are not maintained by yourself (Grid)
  - Create a job description for the resource management system
  - Contact a client of the resource management system
  - Launch an R process …

- More than one user
  - share data and functions

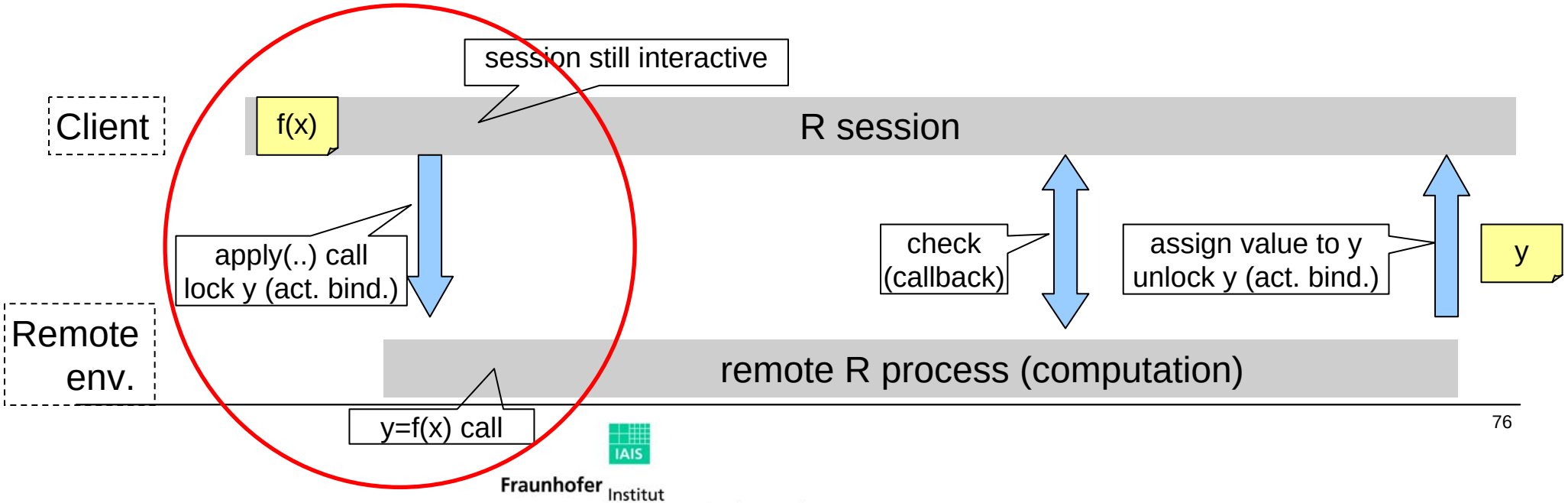
Basic idea of execution with GridR

- Goal:
  - make use of the grid technology in a transparent way
  - passing the functions to be executed in the grid as input predefined function (grid.apply) in their local code
  - make use of available R features & create a package (no changes in the core R implementation)



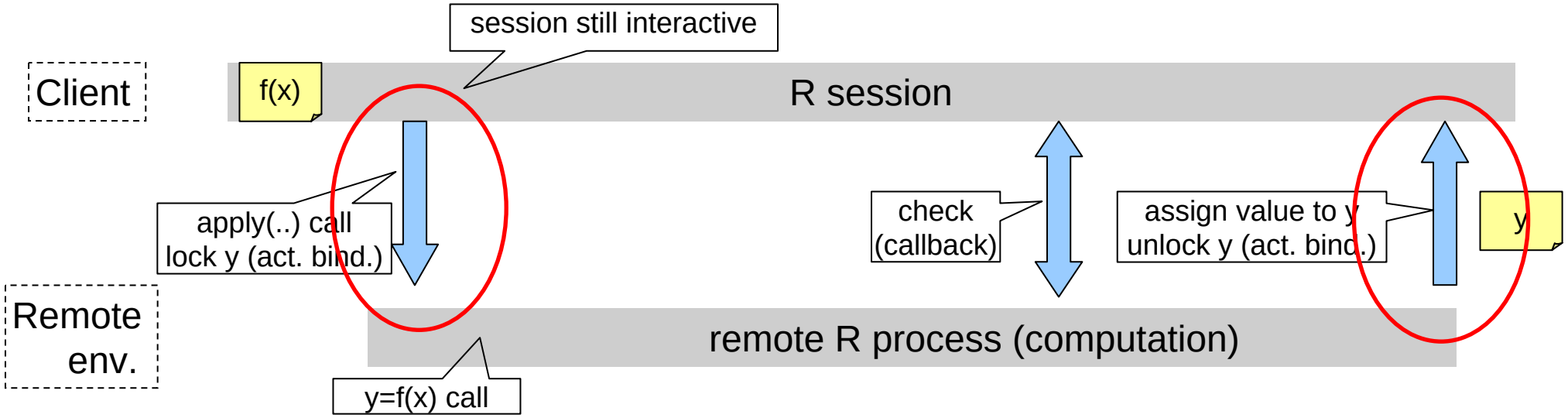
GridR – Functionality

- Parallel function application
  - **`grid.apply("y",f,x)`** computes **`y <- f(x)`** remotely
  - A set of files is created for starting up the remote execution
  - Control is directly returned to R session, user can continue work
- Catching user errors
  - GridR tries to analyze dependencies between **`f`** and functions and variables used by **`f`** to automatically upload them together with R (**`check=TRUE)`**
  - Errors thrown by missing data are caught, value is retrieved and computation is restarted



GridR – Functionality (cont.)

- Locking
  - User can not modify value of **`y`** while computation is running
  - Avoids accidental overwriting of data (consistency of variables)
  - Using R's **`makeActiveBinding()`**



GridR – Active Bindings

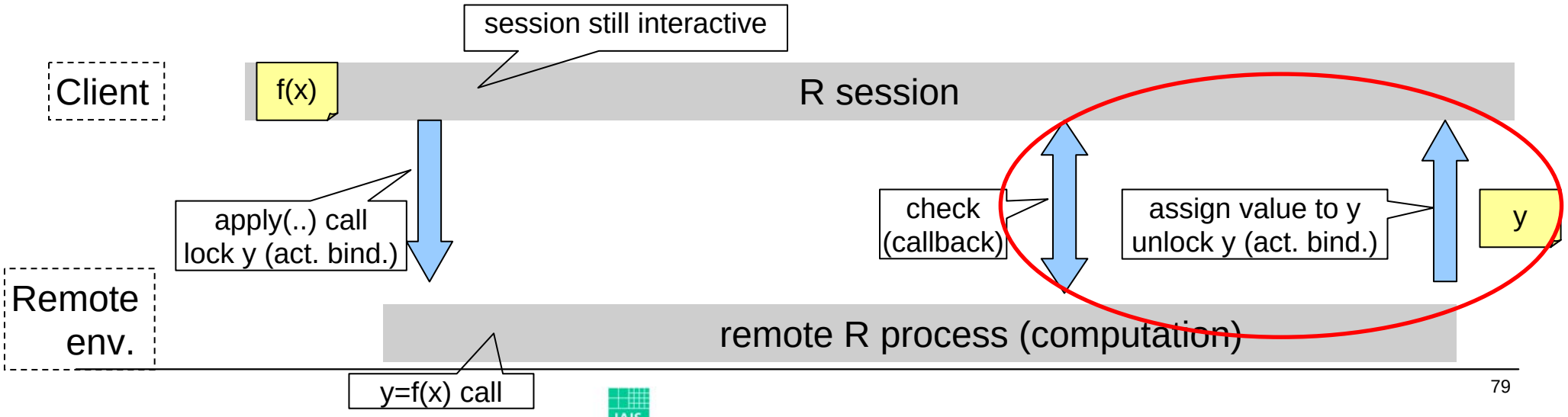
- A variable is replaced by a function call
- Predefined function handles the locking system and allows working interactively with variable
  - When the variable is read: function returns the value associated to the variable (or an error code if the variable is locked)
  - When a value is assigned: function is called with the value as parameter for storage in an internal structure

```
f <- local( {
    x <- 1
    function(v) {
      if (missing(v))
        cat("get\n")
      else {
        cat("set\n")
        x <<- v
      }
      x
    }
})
makeActiveBinding("myVar", f, .GlobalEnv)
bindingIsActive("myVar", .GlobalEnv)
myVar
myVar <- 2
myVar
```

GridR – Functionality (cont.)

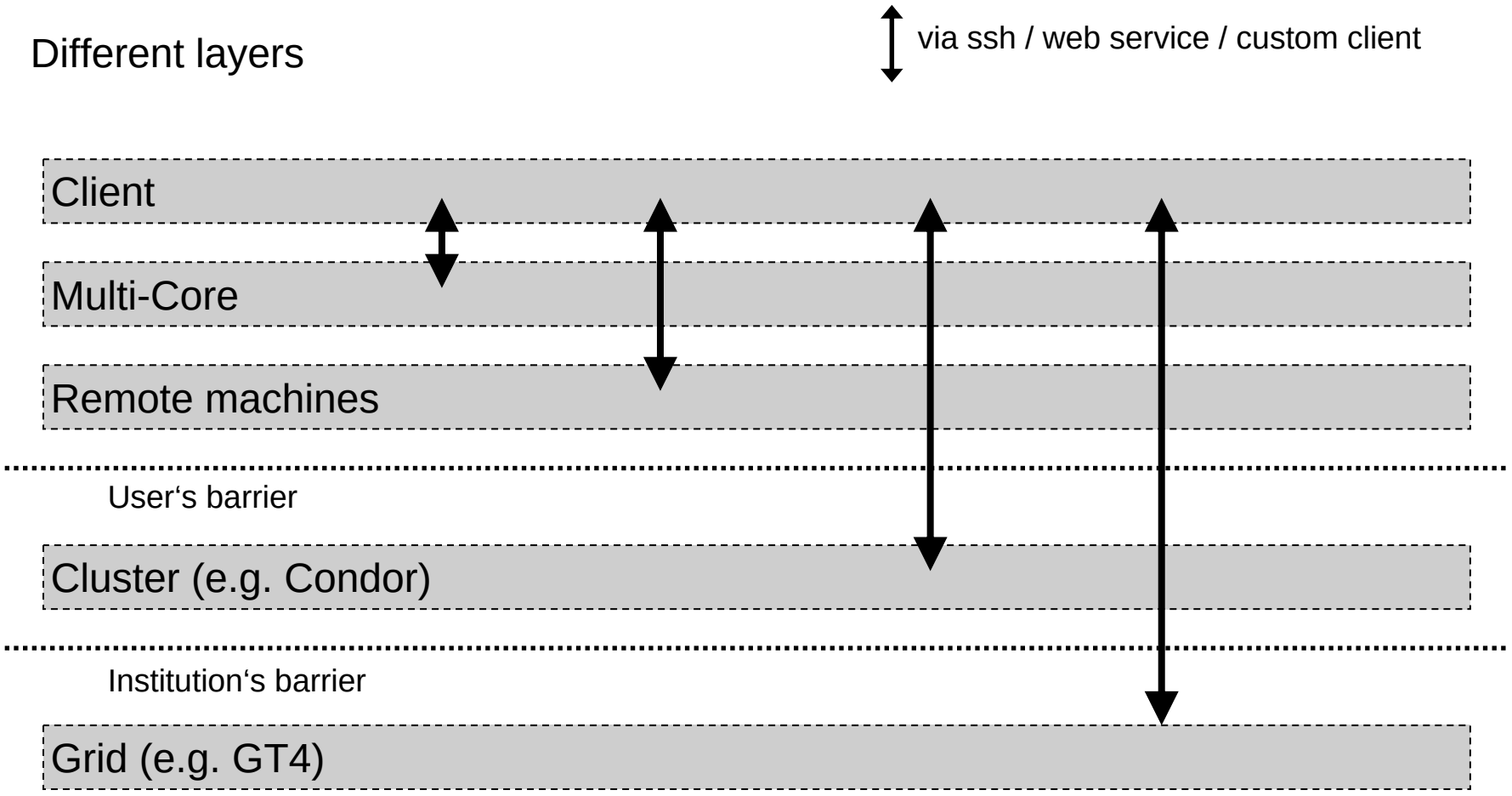
- Automatic retrieval of results
  - When remote instance's computation is finished, value of **y** is automatically loaded back into the running session
  - Using R's task callbacks - functions that are executed automatically by R after the user has issued a command

```
cb <- function() {
     function(...) {
        cat("executing callback!\n")
        return(TRUE)
     }
}
# add the callback
addTaskCallback(cb())
```



GridR interfaces

- Different layers



GridR interfaces – detail

- Submission and execution modes:
- local
  - Execute a job on the local (multicore) machine
- remote.ssh
  - Send a job to a remote machine via ssh
- condor.ssh
  - Send a job to a Condor cluster via ssh
- condor.ws
  - Send a job to a Condor cluster via a web service
- globus.ws
  - Send a job to a GT4 GRAM via a web service
- globus.cog
  - Send a job to a GT4 GRAM via a client side cog-kit
- acgt
  - Send a job to the ACGT system

GridR configuration (init)

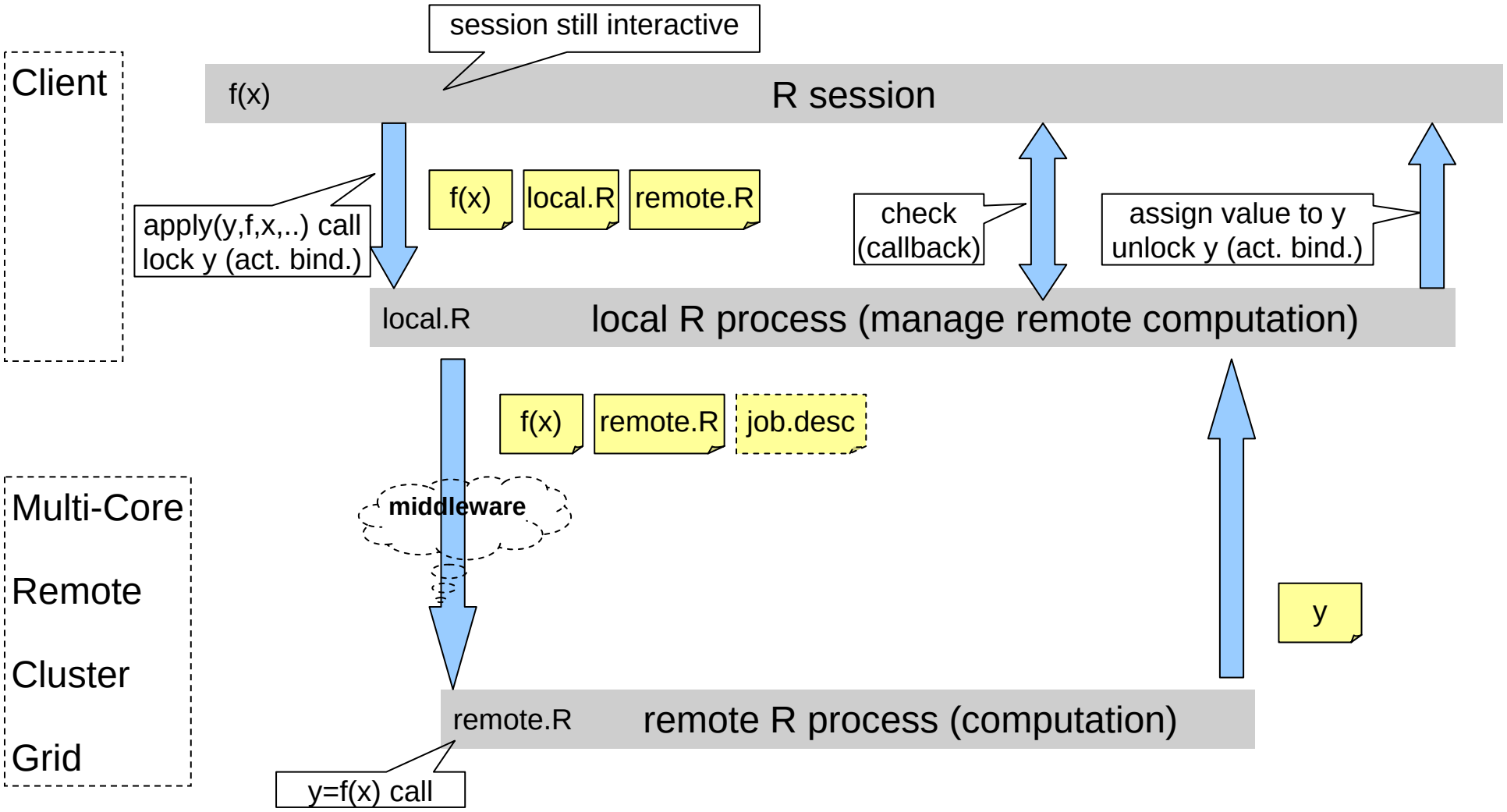
- A set of parameters has to be specified (depending on the mode choosen)
  - e.g., local and remote paths, shared dir, web service URIs, Myproxy info, etc.
- These parameters can be either
  - Passed when calling grid.init(..)
  - Specified in a config file that is loaded automatically when calling grid.init(..)
- Examples:
  - grid.init(service=„globus.cog“,..)
  - grid.init() + config file

```
<GRIDR>
<LOCALTMPDIR>/tmp/</LOCALTMPDIR>
<SERVICE>remote.ssh</SERVICE>
<SSHREMOTEDIR>/home/user/</SSHREMOTEDIR>
<SSHREMOTEIP>192.168.58.128</SSHREMOTEIP>
</GRIDR>
```

Process of executing a single R function with GridR

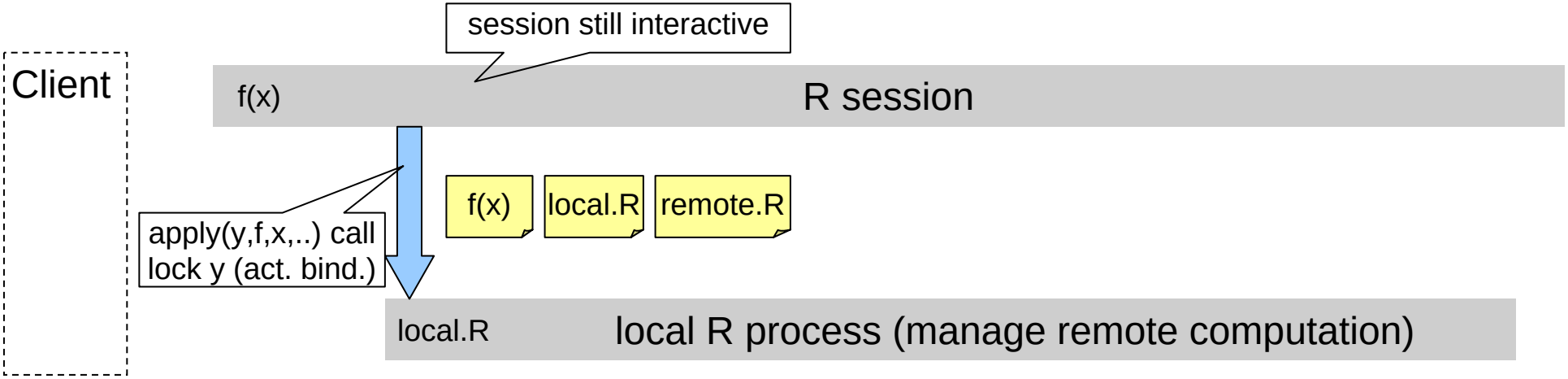
- **Function loading**
  - GridR functions are loaded from the GridR package into the workspace of the R client
- **Grid initialization**
  - grid environment is initialized by calling the function grid.init(..)
    - sets all needed settings for the client side components of the resource management systems to contact (e.g., including the information on the Myproxy certificate in case GT4 or Gridge are used as grid middleware)
    - sets temporary directories to use on the remote execution machine(s)
- **Code writing**
  - The R code which is to be executed in the grid is written and wrapped, e.g., as single R function in the local R environment

Process of execution



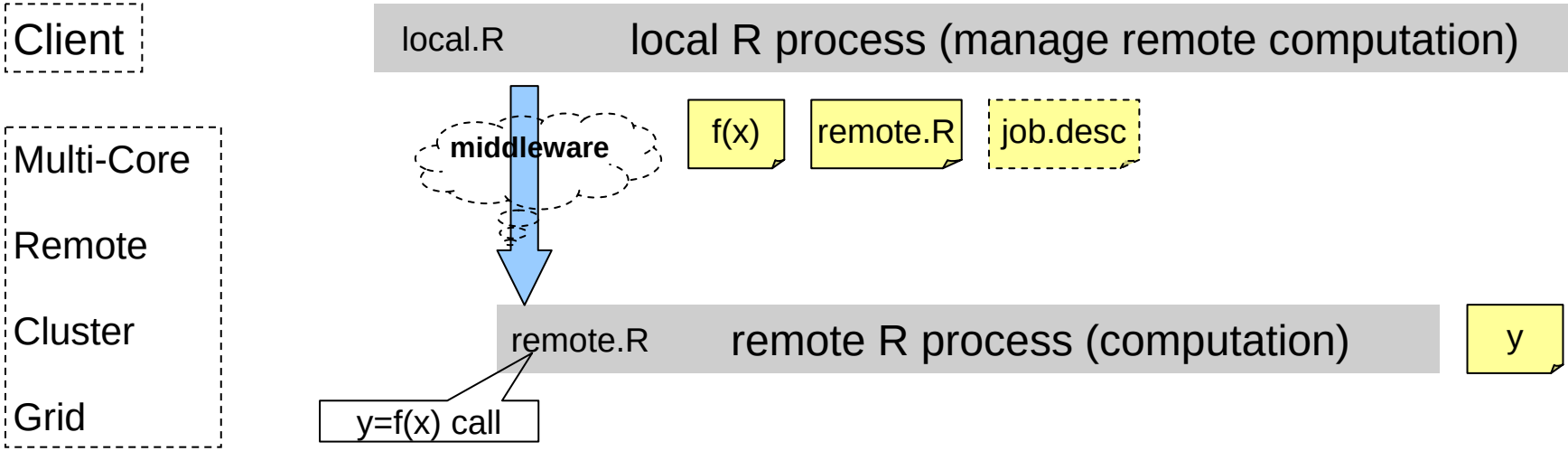
Process of executing a single R function with GridR (cont.)

- **Grid submission –** through grid.apply(..) the process of submission is started
  - Function to be executed in the grid and the needed parameters are written into a file (uniqueID-fx)
  - R script which is executed on the remote machine is generated (uniqueID-RemoteScript.R)
  - R script specifying the “workflow” on the client side is generated (uniqueID-LocalScript.R)
  - The result variable y is locked
  - The local script is executed as separate process in another R session and performs tasks depending on the mode chosen (ssh, web service, cog-kit, etc.)



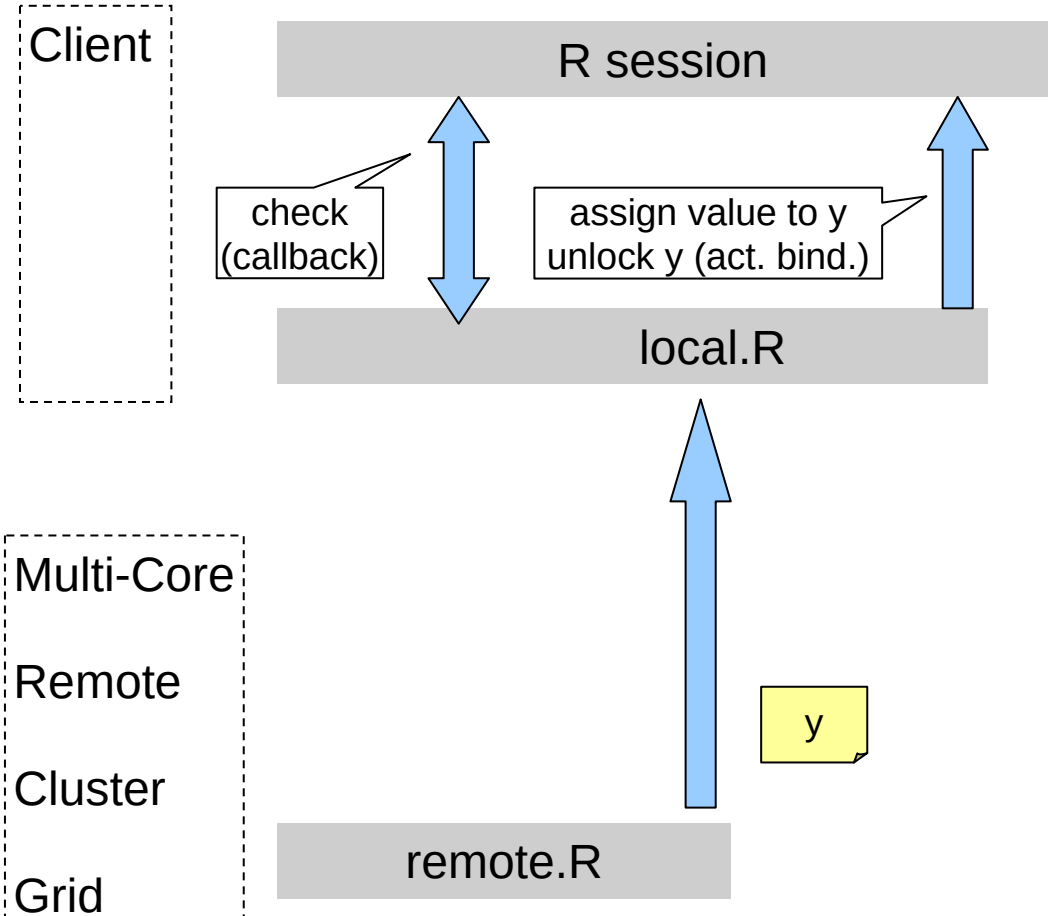
Process of executing a single R function with GridR (cont.)

- The local script performs in detail:
- Stage-in phase - files needed for the job submission (uniqueID-fx and uniqueID-RemoteScript.R) are uploaded to the grid
- A job is generated - Resource management system takes care of
  - staging files to the execution machine
  - launching the processing remote R script (uniqueID-RemoteScript.R) on an execution machine
    - read in uniqueID-fx
    - execute y=f(x)
    - writes the result or errors into a result file (uniqueID-y.dat)



Process of executing a single R function with GridR (cont.2)

- **Waiting for result –** During remote execution y is locked and the R client checks frequently if the file y.dat was created
  - depending on the waiting mode:
    - wait=TRUE: grid.apply(..) blocks until the results are there
    - wait=FALSE: user can use the console and gets a message when the results are there
- **Result processing**
  - Result file (uniqueID-y.dat) is transferred back to the client (Stage-out phase)
  - Result file is loaded and exit status checked
  - value to y is assigned or error information displayed
  - y is unlocked



GridR – important functions

- grid.init
  - Initializes setting necessary for the execution
- grid.apply (y,f,param1, param2, wait=TRUE,check=FALSE, ..)
  - Performs a remote execution of an R function; waits (grid.callback) or sets a lock (grid.lock)
- grid.waitForResult (varlist)
  - Waits until all results are accessible (grid.isLocked, grid.callback)

GridR Example Scene – Shipping of algorithm 1/2

- Simple learning task (locally):

- Create normal 100x3 matrix
- Create normal 10x3 test-matrix
- Create vector length 100 (linear model with noise)
- Bind vector and matrix
- Learn model - predict Y taking V1, V2 and V3 (column names of X) as input
- Make a prediction with the learned model on the test-matrix
- Print out the prediction

```
X <- as.data.frame(array(rnorm(300),c(100,3)))

Xtest <- as.data.frame(array(rnorm(30),c(10,3)))

Y <- X[,1]+X[,2]-2*X[,3]+rnorm(nrow(X))

YX <- cbind(Y,X)

m <- lm(Y~V1+V2+V3,YX)

prediction = predict.lm(m,Xtest)

prediction
```

GridR Example Scene – Shipping of algorithm 2/2

```
generateData <- function() {
    path <- "/tmp/dwegener_matrices.Rdata"
    X <- as.data.frame(array(rnorm(300),c(100,3)))
    Xtest <- as.data.frame(array(rnorm(30),c(10,3)))
    Y <- X[,1]+X[,2]-2*X[,3]+rnorm(nrow(X))
    YX <- cbind(Y,X)
    save(file=path,Xtest,Y,YX)
    return(path)
}
learAndApplyModel <- function(path) {
    load(path)
    m <- lm(Y~V1+V2+V3,YX)
    prediction = predict.lm(m,Xtest)
    return(prediction)
}
```

Remote execution:

- Create two functions
  - Data generation
  - Model learn and apply
- Execute the functions in a way that
  - The data is created remotely
  - The algorithm for learning is shipped to that data

```
library(GridR)
grid.init()
grid.apply("path",generateData,wait=TRUE)
grid.apply("prediction",learAndApplyModel,
        path,check=FALSE)
```

SNOW Example Scene – Shipping of algorithm

- Shipping – code is quite similar

```
library(GridR)
grid.init()
grid.apply("path",generateData,wait=TRUE)
grid.apply("prediction",learAndApplyModel,
        path,check=FALSE)
```

```
library(snow)
cl <- makeCluster(c("localhost"), type = "SOCK")
path=clusterCall(cl,generateData)[[1]]
prediction=clusterCall(cl,learAndApplyModel,path)[[1]]
stopCluster(cl)
```

GridR Example Scene – Distributed CV 1/3

- CV task

```
crossvalidate <- function(X,Y) {
  YX <- cbind(Y,X)
  err <-0
  n <- nrow(X)
  for (i in 1:10) {
    YXtrain <- YX[which(1:n %% 10 != i-1),]
    YXtest <- YX[which(1:n %% 10 == i-1),]
    m <- lm(Y~V1+V2+V3, YXtrain)
    p <- predict.lm(m,YXtest)
    err <- err+mean((p-YXtest[,1])^2)
  }
  err <- err/10
  return(err)
}
X <- as.data.frame(array(rnorm(300),c(100,3)))
Y <- X[,1]+X[,2]-2*X[,3]+rnorm(nrow(X))
err = crossvalidate(X,Y)
```

- CV task prepared for distributed execution

```
cv_single_fold <- function(i,n_fold,X,Y,YX) {
  n <- nrow(X)
  YXtrain <- YX[which(1:n %% n_fold != i-1),]
  YXtest <- YX[which(1:n %% n_fold == i-1),]
  m <- lm(Y~V1+V2+V3,YXtrain)
  p <- predict.lm(m,YXtest)
  err <- mean((p-YXtest[,1])^2)
  return(err)
}
X <- as.data.frame(array(rnorm(300),c(100,3)))
Y <- X[,1]+X[,2]-2*X[,3]+rnorm(nrow(X))
YX <- cbind(Y,X)
n_folds=10
err <-0
for(i in 1:n_folds) {
    err <- err+cv_single_fold(i,n_folds,X,Y,YX)
}
err=err/n_folds
```

GridR Example Scene – Distributed CV 2/3

- Crossvalidation task computed in parallel

```
cv_single_fold <- function(i,n_fold,X,Y,YX) {
  n <- nrow(X)
  YXtrain <- YX[which(1:n %% n_fold != i-1),]
  YXtest <- YX[which(1:n %% n_fold == i-1),]
  m <- lm(Y~V1+V2+V3,YXtrain)
  p <- predict.lm(m,YXtest)
  err <- mean((p-YXtest[,1])^2)
  return(err)
}
```

```
library(GridR)
grid.init()
X <- as.data.frame(array(rnorm(300),c(100,3)))
Y <- X[,1]+X[,2]-2*X[,3]+rnorm(nrow(X))
YX <- cbind(Y,X)

n_folds=10
err <-0
vars=paste("tmp",1:n_folds,sep="")
for(i in 1:n_folds) {
  grid.apply(vars[i],cv_single_fold,i,n_folds,X,Y,YX, wait=FALSE)
}

grid.waitForResult(vars)
for(i in 1:n_folds) {
  err <- err+get(vars[i])
}
err=err/n_folds
```

GridR Example Scene – Distributed CV 3/3

- Crossvalidation task computed in parallel using parameter sweep

```
cv_single_fold <- function(i,n_fold,X,Y,YX) {
  n <- nrow(X)
  YXtrain <- YX[which(1:n %% n_fold != i-1),]
  YXtest <- YX[which(1:n %% n_fold == i-1),]
  m <- lm(Y~V1+V2+V3,YXtrain)
  p <- predict.lm(m,YXtest)
  err <- mean((p-YXtest[,1])^2)
  return(err)
}
```

```
library(GridR)
grid.init(verbose=FALSE)

X <- as.data.frame(array(rnorm(300),c(100,3)))
Y <- X[,1]+X[,2]-2*X[,3]+rnorm(nrow(X))
YX <- cbind(Y,X)
n_folds=10
err<-0

grid.apply("result",cv_single_fold,
  c(1:n_folds),list(n_folds),list(X),list(Y),list(YX), batch=c(1),wait=TRUE)

for(i in 1:n_folds) {
  err <- err+result[[i]][1]
}
err=err/n_folds
```

Snow Example Scene – Distributed CV

- Distribution – code is quite similar

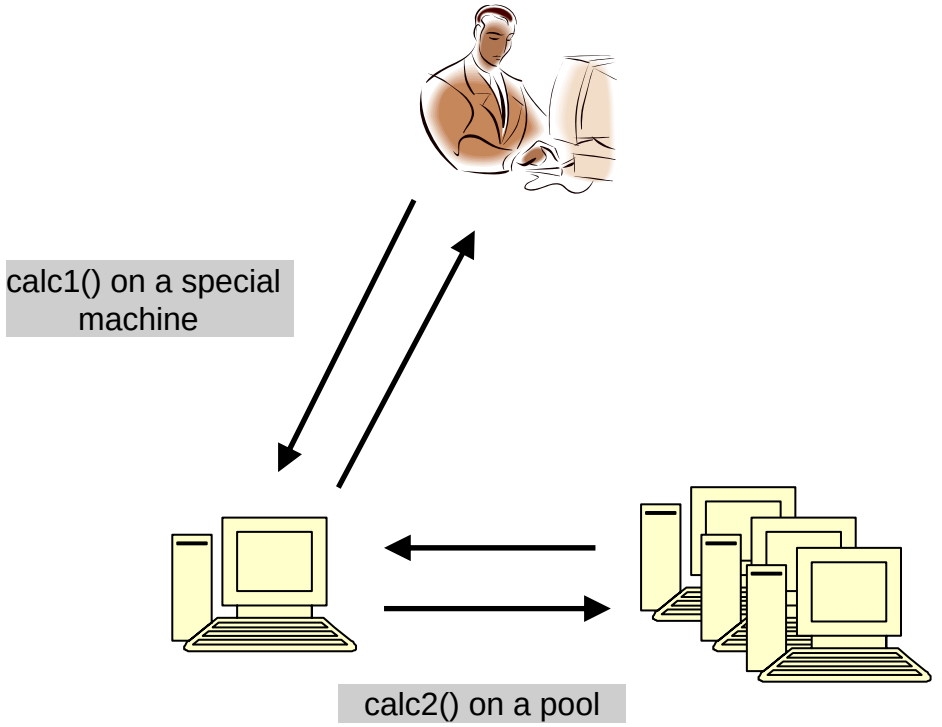
```
library(GridR)
grid.init(verbose=FALSE)

X <- as.data.frame(array(rnorm(300),c(100,3)))
Y <- X[,1]+X[,2]-2*X[,3]+rnorm(nrow(X))
YX <- cbind(Y,X)

n_folds=10
err<-0
grid.apply("result",cv_single_fold,
  c(1:n_folds),list(n_folds),list(X),list(Y),list(YX),
        batch=c(1),wait=TRUE)

for(i in 1:n_folds) {
  err <- err+result[[i]][1]
}
err=err/n_folds
```

```
library(snow)
hosts=rep("localhost",10)
cl <- makeCluster(hosts, type = "SOCK")

X <- as.data.frame(array(rnorm(300),c(100,3)))
Y <- X[,1]+X[,2]-2*X[,3]+rnorm(nrow(X))
YX <- cbind(Y,X)

n_folds=10
err <-0
result = clusterApply(cl, 1:n_folds,cv_single_fold,
       n_folds,X,Y,YX)
stopCluster(cl)

for(i in 1:n_folds) {
  err <- err+ result[[i]]
}
err=err/n_folds
```

GridR parallel

- For expert users that know about the distributed environment
  - Using the GridR client on server side
  - Going through different layers with one submit
  - Recursive implementation possible

```
library(GridR)
grid.init(..some params..)
calc1 <- function(a) {
      library(GridR)
      grid.init(..different params..)
      calc2<-function(b) {return(2*b)}
      grid.apply(„result2“,calc2,a)
      result1<-result2+2
      return(result1)
}
grid.apply(„result“,calc1)
```



Multi-user Collaboration

- By using the same technology as for remote execution (callbacks and active bindings) it is possible to
  - Export variables in a file that is stored in a shared directory
  - Automatically get the new value for a variable that was exported or updated by another user
- Enables easy sharing of data and functions across multiple R sessions on distributed sites
- During the GridR initialization: Specification of a shared directory

```
library(GridR)
grid.init()
x=5
grid.share("x")
```

The GridR package

- License: GPL v2
- Available at: ACGT homepage & CRAN (planned)
- Functionality
  - Remote execution
  - Shipping
  - Parallelization
- Interfaces to
  - SSH machines
  - Condor clusters
  - GT4 grid middleware
- Limitations (currently): side effect output (e.g. plots)

PART III

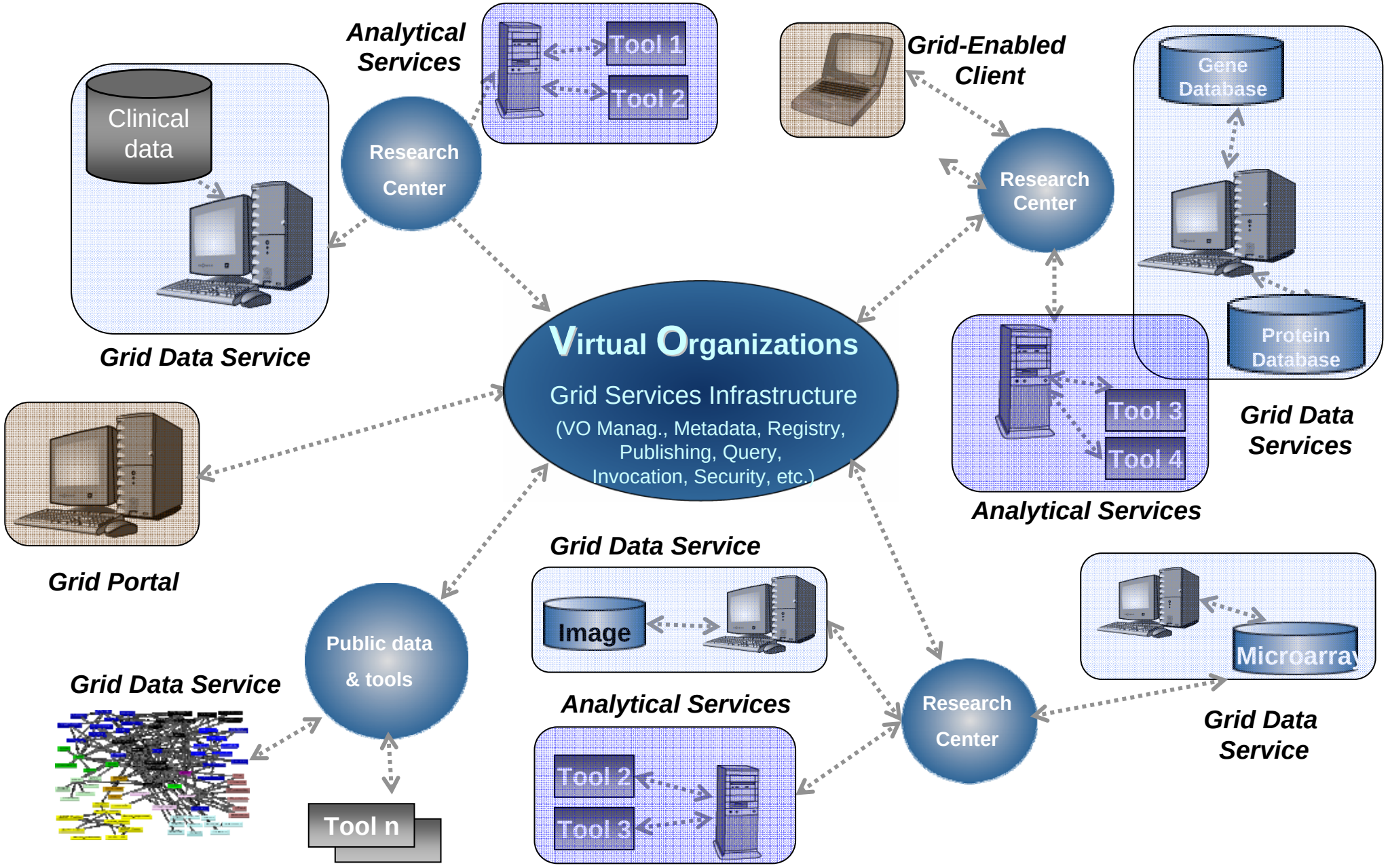
Outline & Timetable

- 9:00 – 9:15 Introduction
- 9:15 – 10:15 R in the Context of Distributed Systems
- 10:15 – 10:30 GridR Installation
- 10:30 – 11:00 Coffee Break
- 11:00 – 11:45 The GridR Package
- 11:45 – 12:15 Real-World Examples
- 12:15 – 12:30 Discussion

Scenarios of Distributed Computing

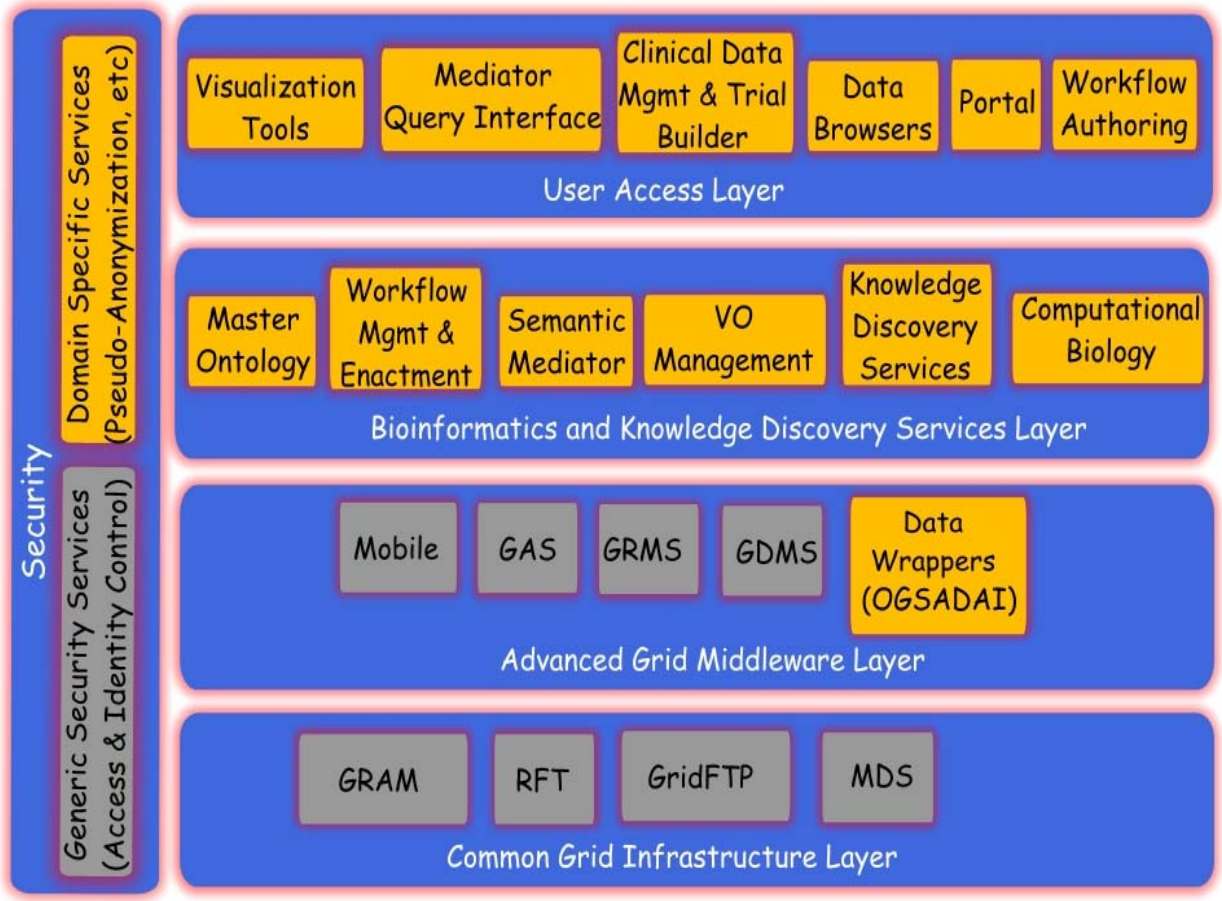
- Using external resources
  - Run big R script on some server instead of own laptop
- Using multiple resources
  - Run n R scripts on n computers - “embarrassingly easy parallelization”, e.g. cross-validation, parameter-tuning, …
  - Run n distinct parts of one R script on n computers Using distributed data
- Access multiple data sources
  - Shipment of algorithms – bring R script to the data, not vice versa
- Multiple-User collaboration
  - Exchange data and code with colleagues
- Setting up a large system
  - R as small part of the architecture

The ACGT Virtual Organizations



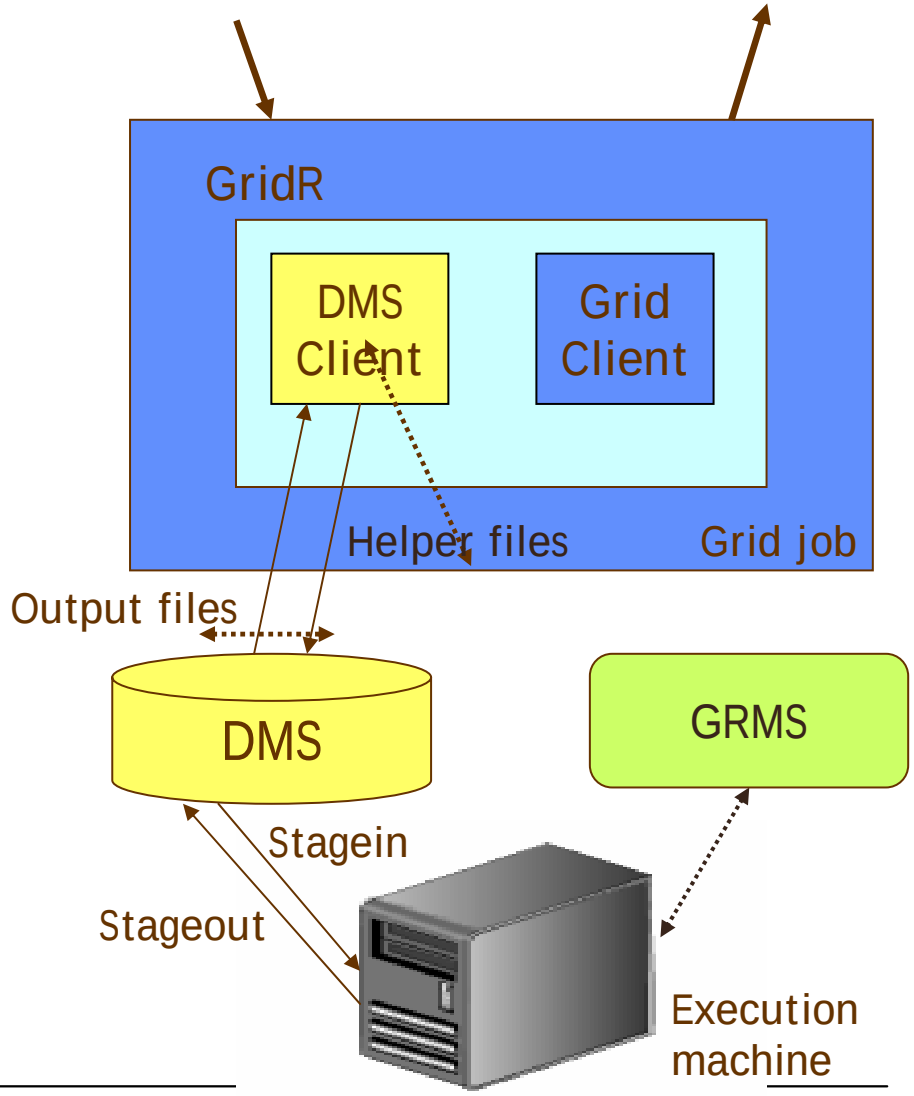
ACGT Architecture

- A platform that is
  - *Service oriented*
  - *Ontology driven*
  - *Grid enabled*



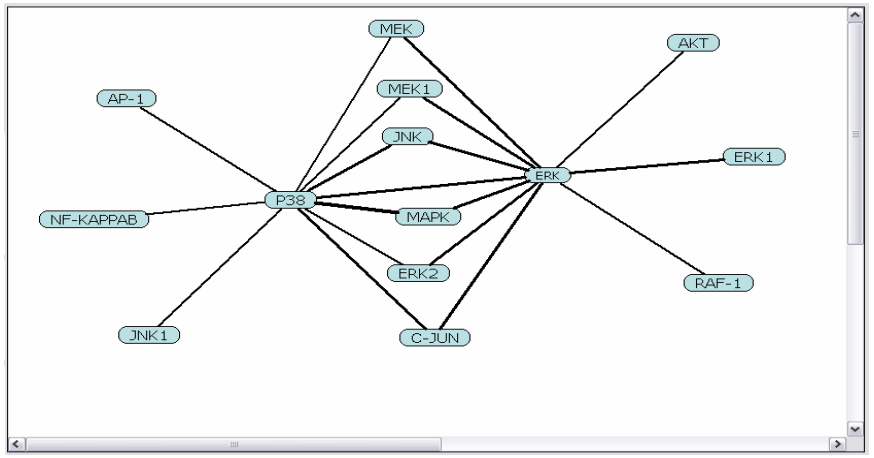
GridR

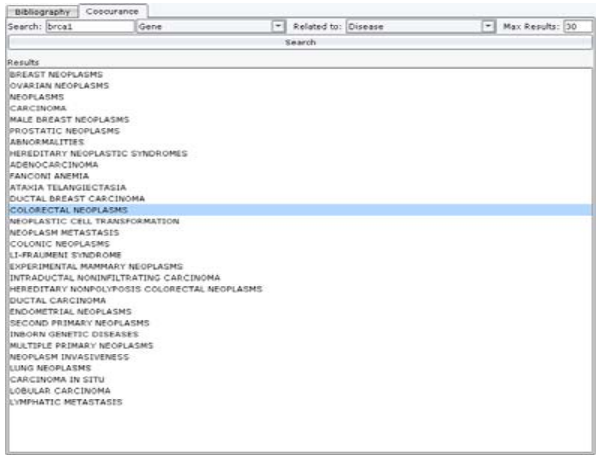
- Remote execution of R code in the Grid.
- New
  - Wrapper to make R look like generic ACGT service
  - Interface to distributed Data Management System and Resource Management System
  - Interface to Meta Data Repository



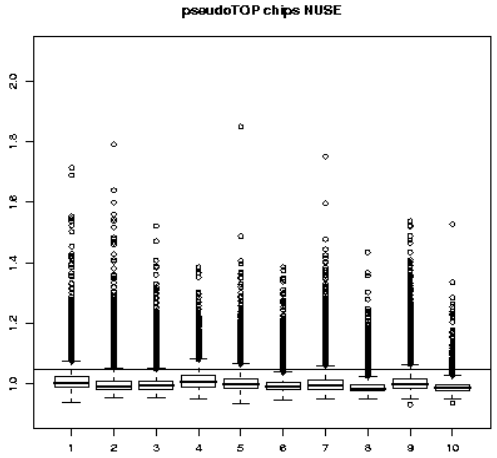
Literature Mining

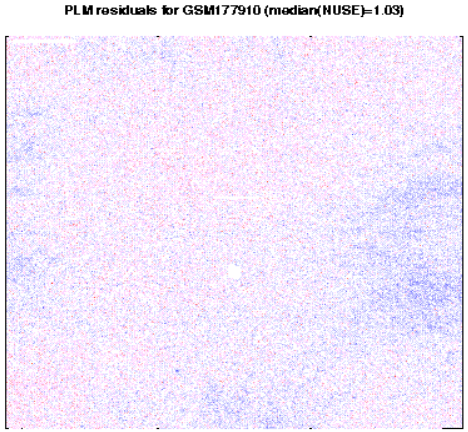
- Extraction of
  - Links from genes to deseases
  - Links from genes to articles
- Based on an database of relations between concept
  - Constructed via literature mining techniques

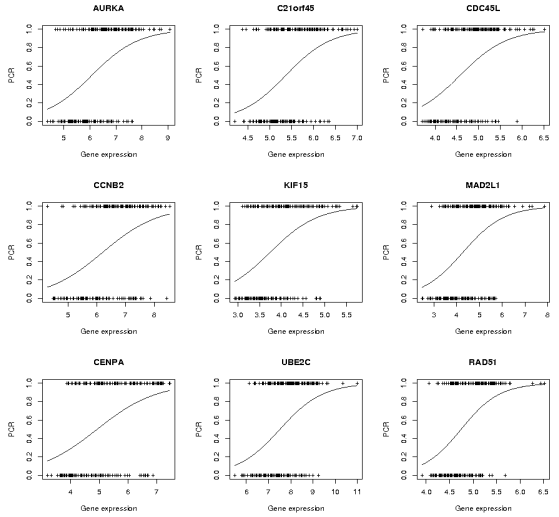


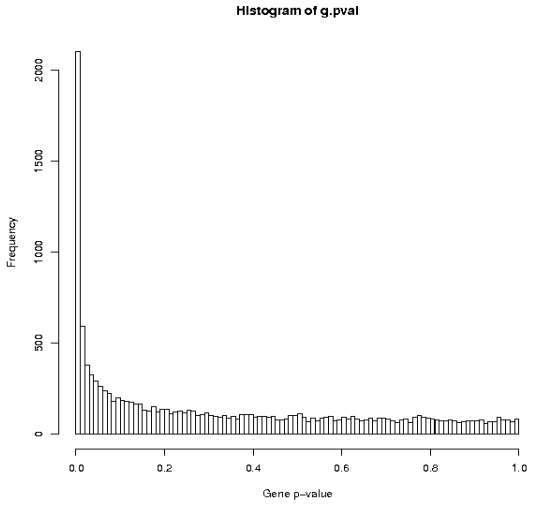


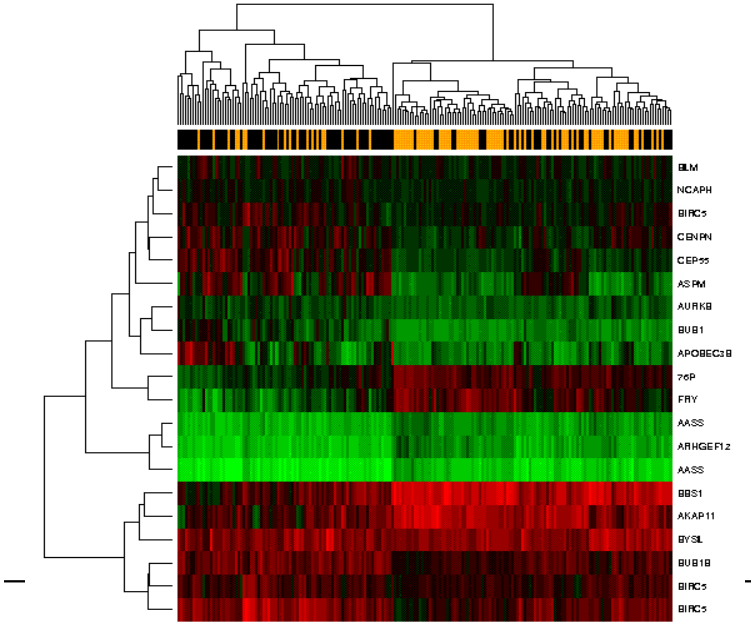
Clinico-Genomic Data

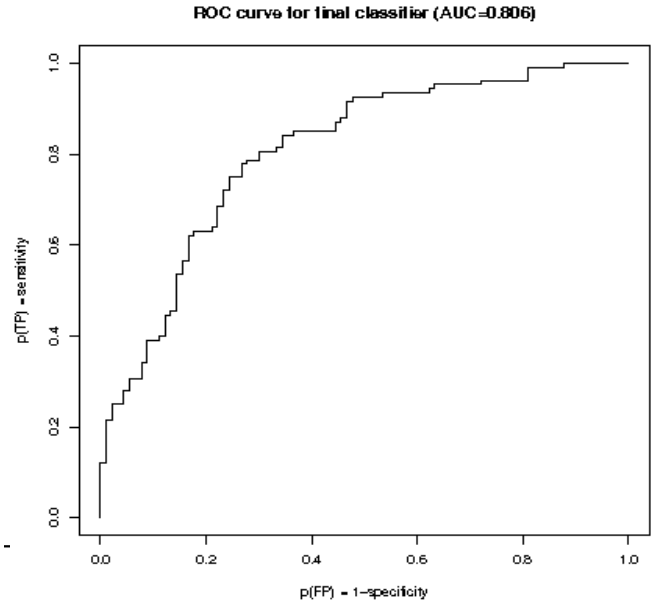












Fraunhofer Institut
Intelligente Analyse- und
Informationssysteme

Scenario: Farmer et al.

Oncogene (2005) 24, 4660–4671

www.nature.com/onc

Identification of molecular apocrine breast tumours by microarray analysis

Pierre Farmer[1,2], Herve Bonnefoi[3,4,5], Veronique Becette[6], Michele Tubiana-Hulin[6], Pierre Fumoleau[7], Denis Larsimont[8], Gaetan MacGrogan[9], Jonas Bergh[10], David Cameron[11], Darlene Goldstein[1,2], Stephan Duss[2], Anne-Laure Nicoulaz[2], Cathrin Brisken[2], Maryse Fiche[12], Mauro Delorenzi[1,2] and Richard Iggo*[,2]

[1]Swiss Institute of Bioinformatics (SIB), Lausanne, Switzerland; [2]National Centre of Competence in Research (NCCR) Molecular Oncology, Swiss Institute for Experimental Cancer Research (ISREC), Epalinges, Switzerland; [3]Hôpitaux Universitaires de Genève, Geneva, Switzerland; [4]for the Swiss Group for Clinical Cancer Research (SAKK), Bern, Switzerland; [5]European Organisation for Research and Treatment of Cancer (EORTC), Brussels, Belgium; [6]Centre René Huguenin, St-Cloud, France; [7]Centre René Gauducheau, Nantes, France; [8]Institut Jules Bordet, Brussels, Belgium; [9]Institut Bergonié, Bordeaux, France; [10]for the Swedish Breast Cancer Group (SweBCG), Karolinska Institute, Stockholm, Sweden; [11]for the Anglo-Celtic Cooperative Oncology Group (ACCOG), Edinburgh University, Edinburgh, UK; [12]Centre Hospitalier Universitaire Vaudois, Lausanne, Switzerland

Previous microarray studies on breast cancer identified multiple tumour classes, of which the most prominent, named luminal and basal, differ in expression of the oestrogen receptor α gene (ER). We report here the identification of a group of breast tumours with increased androgen signalling and a 'molecular apocrine' gene expression profile. Tumour samples from 49 patients with large operable or locally advanced breast cancers were

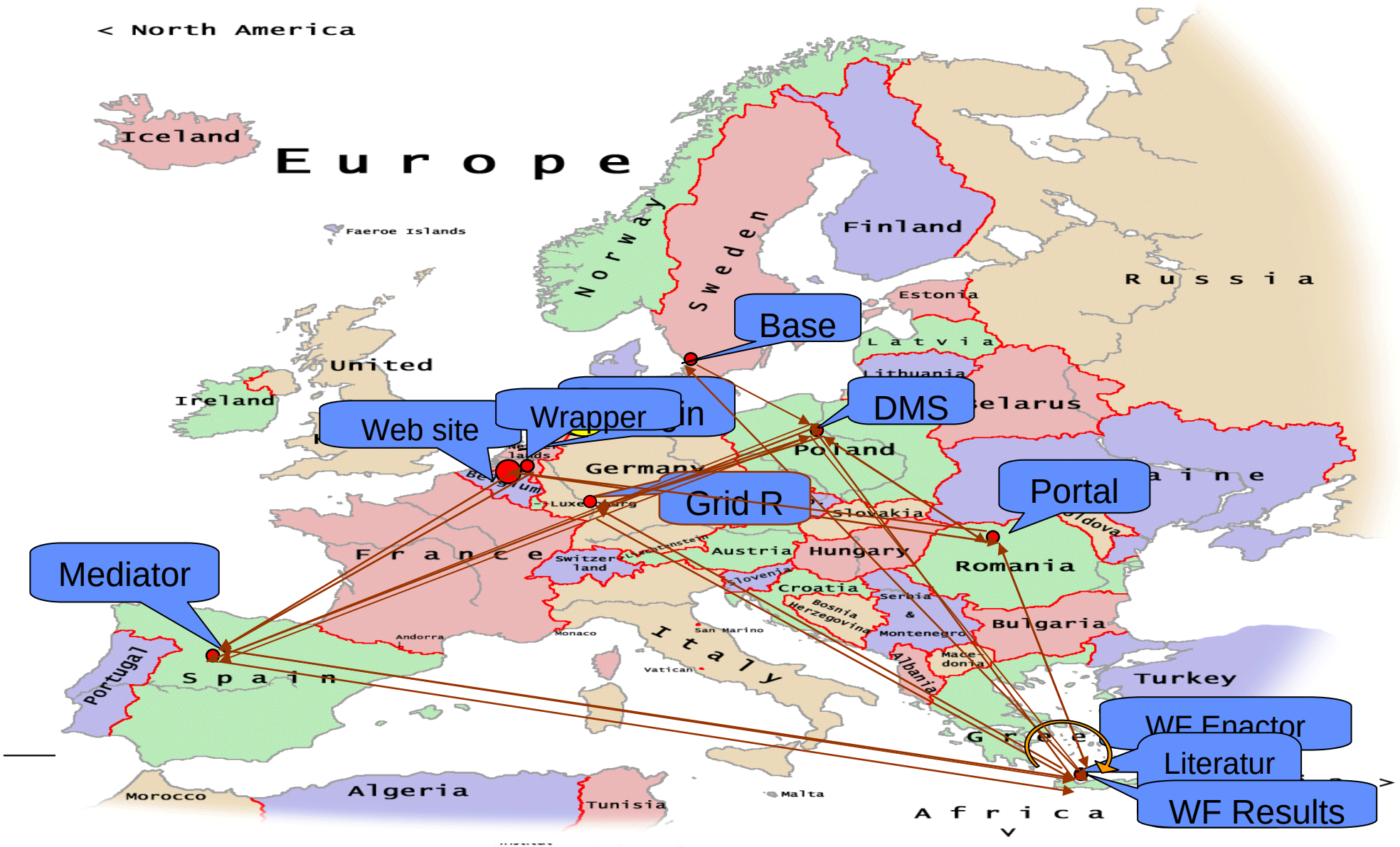
Introduction

Despite the existence of 18 different histopathological types of breast cancer (Ellis, 2003), the major distinctions important in clinical practice are based on oestrogen receptor (ER) and ERBB2 status rather than histopathological tumour type. Unsupervised hierarchical clustering of microarray data readily identifies

Use Case

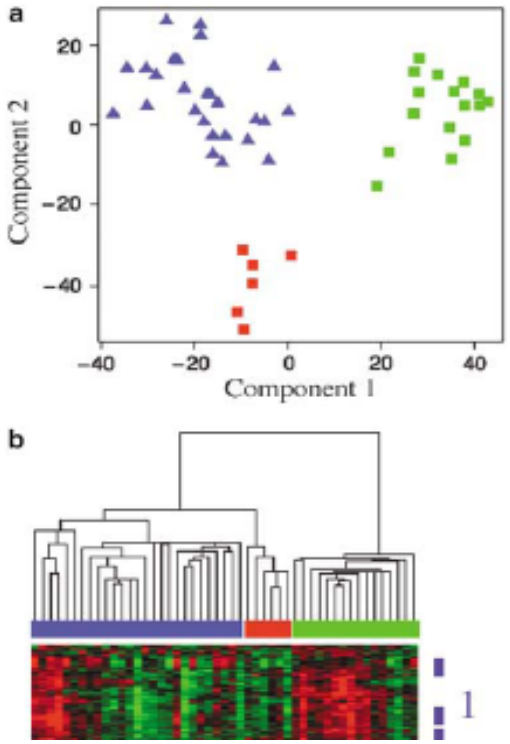
- Researcher are analysing data in R
- Local Laptop
  - R as user interface
  - Development of algorithms
  - Monitoring of Execution
- Remote Grid Machine
  - Data (~800 MB)
  - Execution of algorithms
- Transfer of algorithms & results only!

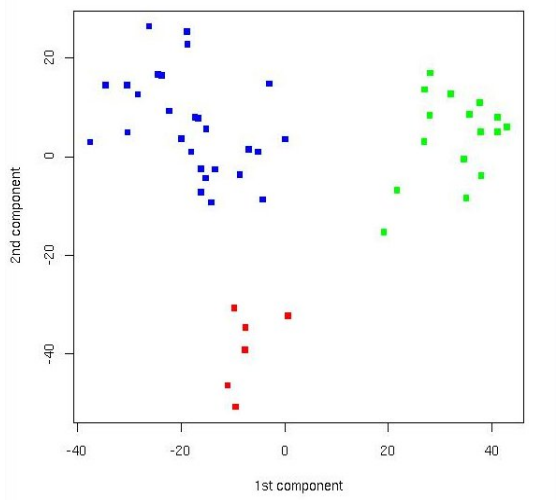
Execution Track of the Workflow

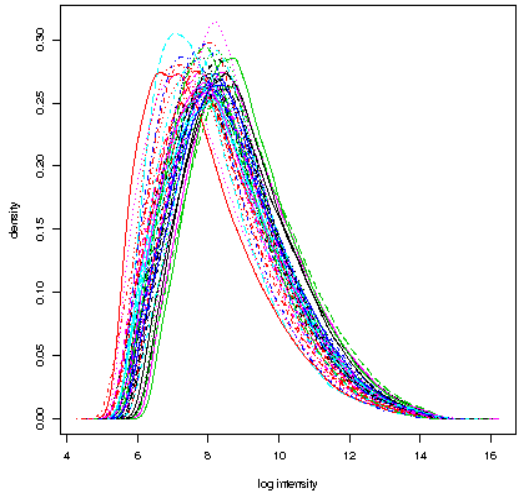


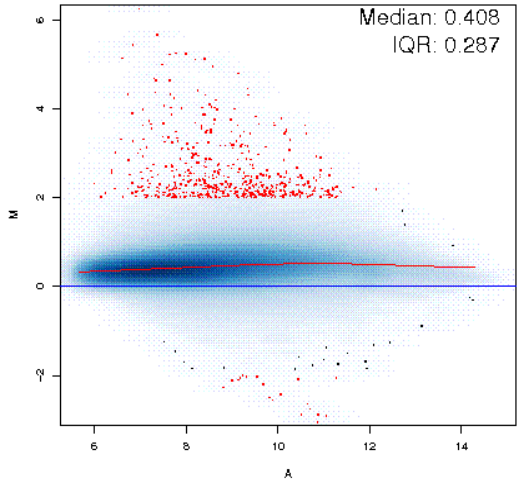
GridR results

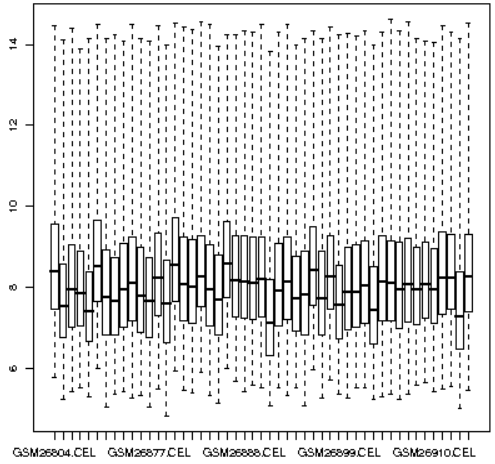
Comparison with the PCA plot showed that the first component splits the tumours into basal (green in Figure 1a) and luminal (blue in Figure 1a) groups. As





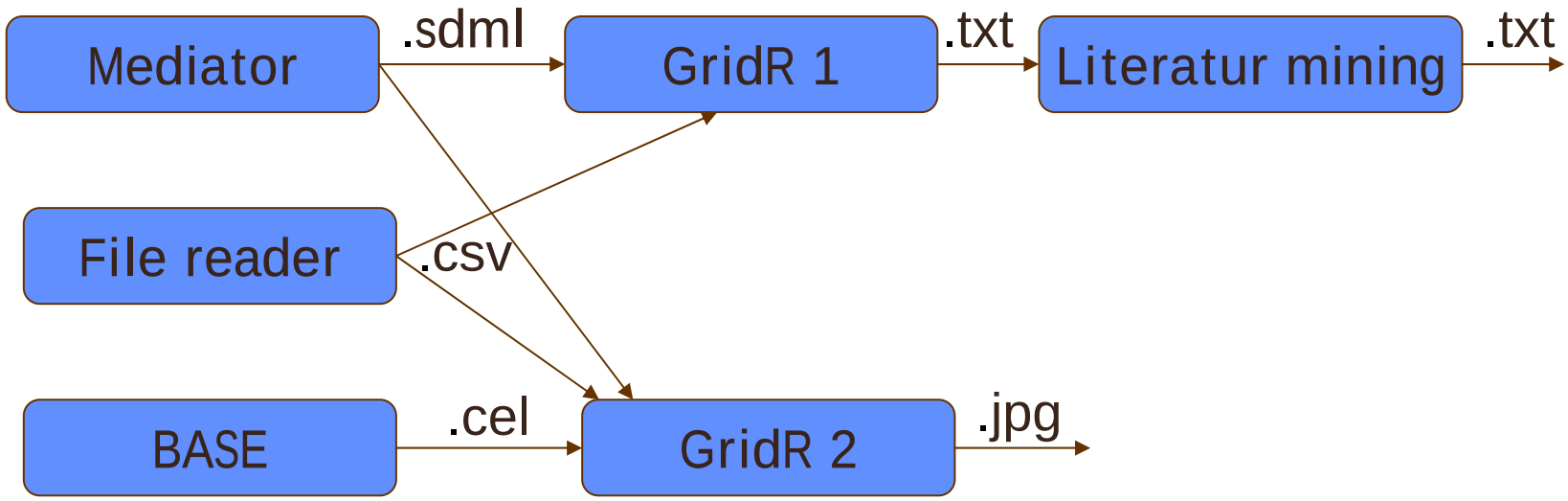






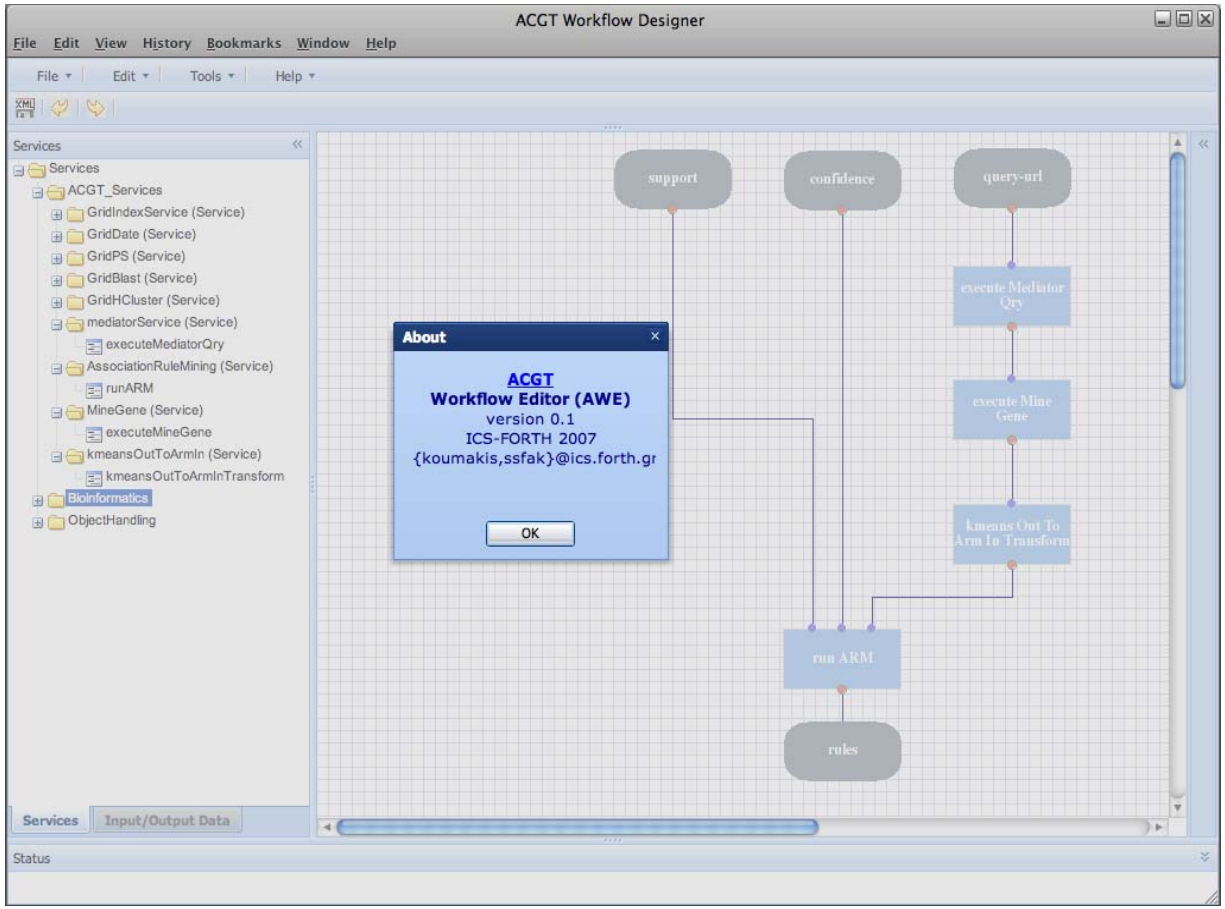
Workflows

- Representation of the single steps of data processing
  - Nodes = services
  - Arrows = data flow
- Necessary to control the complexity of distributed execution



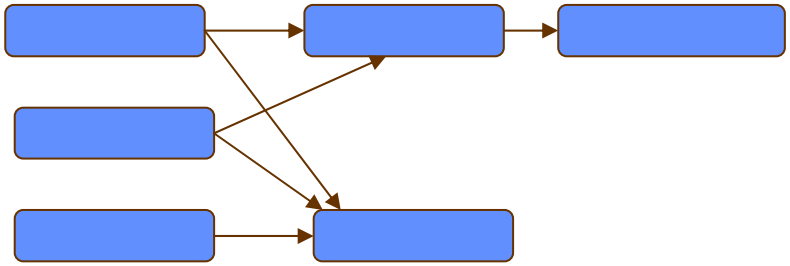
ACGT Workflow Editor

- Web based and integrated into the ACGT portal.
- Efficient discovery & browsing of available tools and workflows.
- Strive for “intelligent support” of the user
- Validation of the correctness (Syntactic and Semantic) of the analysis pipeline.



How Workflows Differ from Scripts

- Limited language – no loops, no if-then-else
- Limited data types – need to be defined in advance, or are ambiguous
- Limited complexity - good thing!
  - Easy to execute
  - Easy to check
  - Easy to re-use
- R is great for rapidly prototyping statistical analysis
- Experience shows: problems arise when
  - Analysis is to be re-executed
  - Approach is to be transferred to new data



The Advantage of Workflows

- Workflows allow an analysis of workflow properties
  - “users who have used this workflow found also this workflow interesting”
  - “in a workflow with these services, also these services were frequently used”
  - “on this type of data, these workflows / Services are usually used”
- Meta data repository
  - Repository of
    - R scripts and functions
    - meta data (partially automatically generated from R help texts)
    - use cases (data sets, literature, etc)
  - Immediate execution of R scripts with GridR

Future Development of GridR

- Automatic construction of Grid workflows from R command history
- Executing functions / packages that are not installed locally
- More fine-grained access rights in Grid workspace
- R in a browser
- R on a mobile device

And Now it's Time For…

Questions and Discussion!

The End…

Thank you for your attention!