University of Kent | Computing

# Provenance Tracking in CXXR

Chris A. Silles
Andrew R. Runnalls

Computing Laboratory, University of Kent, UK

# Outline

1 Introduction

2 Provenance

3 CXXR

4 Provenance-Aware CXXR

5 Conclusion

# Motivating Example

## A simple exploration

### R Session

```
> library(MASS)
    # For ‘mammals’ dataset
```

# Motivating Example

## A simple exploration

### R Session

```
> library(MASS)
     # For ‘mammals’ dataset
```

First few rows of ‘mammals’:

```
> mammals
                      body       brain
Arctic fox           3.385       44.50
Owl monkey           0.480       15.50
Mountain beaver      1.350        8.10
Cow                465.000      423.00
Grey wolf           36.330      119.50
...57 rows omitted...
```

# Motivating Example

## A simple exploration

### R Session

```
> library(MASS)
     # For ‘mammals’ dataset
> brain <- mammals[,2]
```

First few rows of ‘mammals’:

```
> mammals
                     body      brain
Arctic fox          3.385      44.50
Owl monkey          0.480      15.50
Mountain beaver     1.350       8.10
Cow               465.000     423.00
Grey wolf          36.330     119.50
...57 rows omitted...
```

# Motivating Example

## A simple exploration

### R Session

```
> library(MASS)
     # For 'mammals' dataset
> brain <- mammals[,2]
> body <- mammals[,1]
```

First few rows of 'mammals':

```
> mammals
                      body      brain
Arctic fox           3.385      44.50
Owl monkey           0.480      15.50
Mountain beaver      1.350       8.10
Cow                465.000     423.00
Grey wolf           36.330     119.50
...57 rows omitted...
```

# Motivating Example

## A simple exploration

### R Session

```
> library(MASS)
     # For 'mammals' dataset
> brain <- mammals[,2]
> body <- mammals[,1]
> plot(body,brain)
```

First few rows of 'mammals':

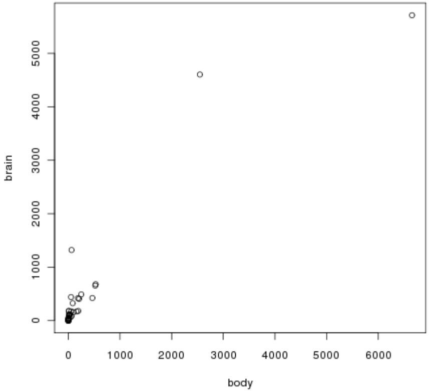
```
> mammals
                      body      brain
Arctic fox           3.385      44.50
Owl monkey           0.480      15.50
Mountain beaver      1.350       8.10
Cow                465.000     423.00
Grey wolf           36.330     119.50
...57 rows omitted...
```

# Motivating Example

## A simple exploration

### R Session

```
> library(MASS)
     # For ‘mammals’ dataset
> brain <- mammals[,2]
> body <- mammals[,1]
> plot(body,brain)
```

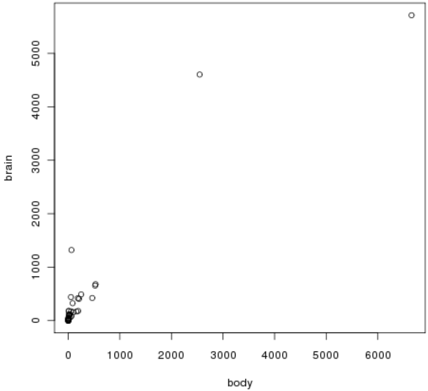


# Motivating Example

## A simple exploration

### R Session

```
> library(MASS)
     # For ‘mammals’ dataset
> brain <- mammals[,2]
> body <- mammals[,1]
> plot(body,brain)
> lbrain <- log(brain)
```

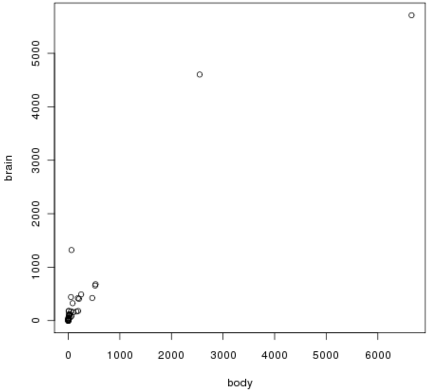


# Motivating Example

## A simple exploration

### R Session

```
> library(MASS)
     # For 'mammals' dataset
> brain <- mammals[,2]
> body <- mammals[,1]
> plot(body,brain)
> lbrain <- log(brain)
> lbody <- log(body)
```

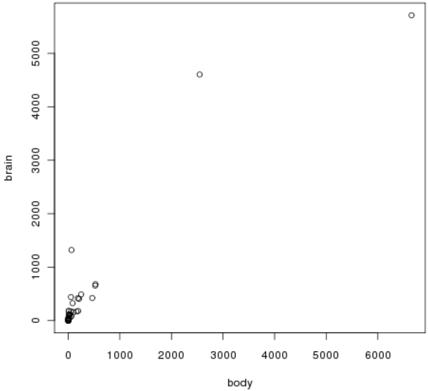


# Motivating Example

## A simple exploration

### R Session

```
> library(MASS)
     # For 'mammals' dataset
> brain <- mammals[,2]
> body <- mammals[,1]
> plot(body,brain)
> lbrain <- log(brain)
> lbody <- log(body)
> plot(lbody,lbrain)
```

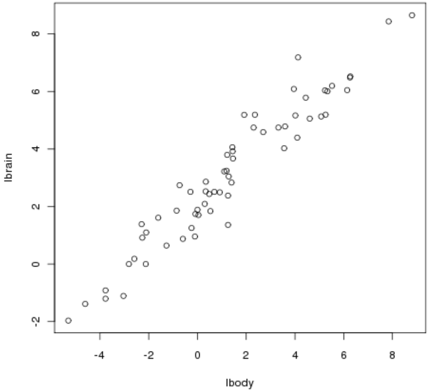


# Motivating Example

## A simple exploration

### R Session

```
> library(MASS)
     # For ‘mammals’ dataset
> brain <- mammals[,2]
> body <- mammals[,1]
> plot(body,brain)
> lbrain <- log(brain)
> lbody <- log(body)
> plot(lbody,lbrain)
```

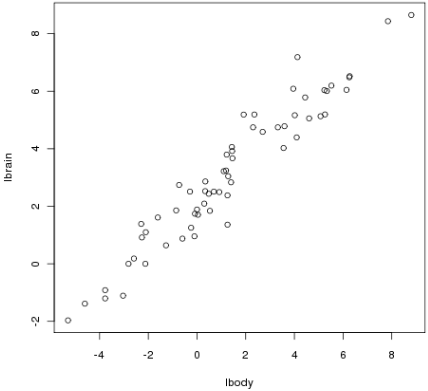


# Motivating Example

## A simple exploration

### R Session

```
> library(MASS)
     # For ‘mammals’ dataset
> brain <- mammals[,2]
> body <- mammals[,1]
> plot(body,brain)
> lbrain <- log(brain)
> lbody <- log(body)
> plot(lbody,lbrain)
> r <- lm(lbrain ∼ lbody)
```

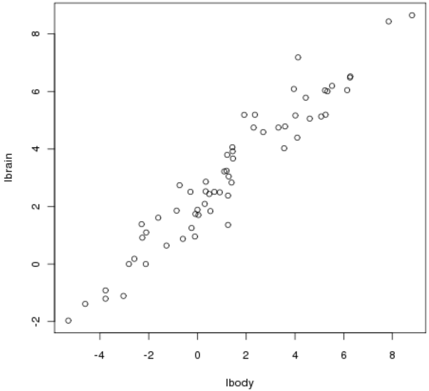


# Motivating Example

A simple exploration

## R Session

```
> library(MASS)
     # For ‘mammals’ dataset
> brain <- mammals[,2]
> body <- mammals[,1]
> plot(body,brain)
> lbrain <- log(brain)
> lbody <- log(body)
> plot(lbody,lbrain)
> r <- lm(lbrain ∼ lbody)
> abline(r)
```

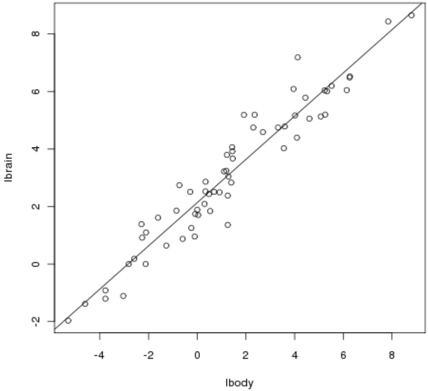


# Motivating Example

## A simple exploration

### R Session

```
> library(MASS)
     # For ‘mammals’ dataset
> brain <- mammals[,2]
> body <- mammals[,1]
> plot(body,brain)
> lbrain <- log(brain)
> lbody <- log(body)
> plot(lbody,lbrain)
> r <- lm(lbrain ∼ lbody)
> abline(r)
```



# What is Provenance?

From the Oxford English Dictionary:

**provenance,** *n*

1. The proceeds from a business. *Obs. rare*.
2. The fact of coming from some particular source or quarter; origin, derivation.
3. The history of the ownership of a work of art or an antique, used as a guide to authenticity or quality; a documented record of this.
4. *Forestry*. The geographic source of tree seed; the place of origin of a tree. Also: seed from a specific location.

Provenance of data objects:

- What primary data items were drawn upon during creation
- What sequence of operations was performed
- How a data object has later been used

# What is Provenance?

From the Oxford English Dictionary:

**provenance,** *n*

1. The proceeds from a business. *Obs. rare*.
2. The fact of coming from some particular source or quarter; origin, derivation.
3. The history of the ownership of a work of art or an antique, used as a guide to authenticity or quality; a documented record of this.
4. *Forestry*. The geographic source of tree seed; the place of origin of a tree. Also: seed from a specific location.

Provenance of data objects:

- What primary data items were drawn upon during creation
- What sequence of operations was performed
- How a data object has later been used

# What is Provenance?

From the Oxford English Dictionary:

**provenance,** *n*

1. The proceeds from a business. *Obs. rare*.
2. The fact of coming from some particular source or quarter; origin, derivation.
3. The history of the ownership of a work of art or an antique, used as a guide to authenticity or quality; a documented record of this.
4. *Forestry*. The geographic source of tree seed; the place of origin of a tree. Also: seed from a specific location.

Provenance of data objects:

- What primary data items were drawn upon during creation
- What sequence of operations was performed
- How a data object has later been used

# The beginning of Provenance-Aware Computing

When, in 1988 *New-S* succeeded *S*, it became one of – if not – the first provenance-aware software application(s) with its novel **S AUDIT** facility.

It is described by Becker and Chambers in their paper *Auditing of Data Analyses*[1].

An **audit file** was maintained by *New-S* which recorded each top-level command issued in this and previous sessions within the workspace, and identified those objects read from and written to.

The audit file was then processed by *S AUDIT*.

[1] SIAM J. Sci. Stat. Comput. 9 [1988] pp. 747–60

# S AUDIT

## Example S AUDIT File

```
#~New session: Time: 542034997; Version: "S Tue Mar 3 10:14:20 EST 1987"
m<-matrix(read("brain.body"),byrow=T,ncol=2)
#~put "/usr/rab/.Data/m" 542035057 "structure"
brain<-m[,1]
#~get "/usr/rab/.Data/m" 542035057 "any"
#~put "/usr/rab/.Data/brain" 542035066 "real"
body<-m[,2]
#~get "/usr/rab/.Data/m" 542035057 "any"
#~put "/usr/rab/.Data/body" 542035072 "real"
plot(body,brain)
#~get "/usr/rab/.Data/body" 542035072 "any"
#~get "/usr/rab/.Data/brain" 542035066 "any"
```

What is recorded in the S AUDIT file:

- Top-level commands
- Data objects read
- Data objects written

# S AUDIT

## Example S AUDIT File

```
#~New session: Time: 542034997; Version: "S Tue Mar 3 10:14:20 EST 1987"
m<-matrix(read("brain.body"),byrow=T,ncol=2)
#~put "/usr/rab/.Data/m" 542035057 "structure"
brain<-m[,1]
#~get "/usr/rab/.Data/m" 542035057 "any"
#~put "/usr/rab/.Data/brain" 542035066 "real"
body<-m[,2]
#~get "/usr/rab/.Data/m" 542035057 "any"
#~put "/usr/rab/.Data/body" 542035072 "real"
plot(body,brain)
#~get "/usr/rab/.Data/body" 542035072 "any"
#~get "/usr/rab/.Data/brain" 542035066 "any"
```

What is recorded in the S AUDIT file:

- Top-level commands
- Data objects read
- Data objects written

# S AUDIT

## Example S AUDIT File

```
#~New session: Time: 542034997; Version: "S Tue Mar 3 10:14:20 EST 1987"
m<-matrix(read("brain.body"),byrow=T,ncol=2)
#~put "/usr/rab/.Data/m" 542035057 "structure"
brain<-m[,1]
#~get "/usr/rab/.Data/m" 542035057 "any"
#~put "/usr/rab/.Data/brain" 542035066 "real"
body<-m[,2]
#~get "/usr/rab/.Data/m" 542035057 "any"
#~put "/usr/rab/.Data/body" 542035072 "real"
plot(body,brain)
#~get "/usr/rab/.Data/body" 542035072 "any"
#~get "/usr/rab/.Data/brain" 542035066 "any"
```

What is recorded in the S AUDIT file:

- Top-level commands
- Data objects read
- Data objects written

# S AUDIT

## Example S AUDIT File

```
#~New session: Time: 542034997; Version: "S Tue Mar 3 10:14:20 EST 1987"
m<-matrix(read("brain.body"),byrow=T,ncol=2)
#~put "/usr/rab/.Data/m" 542035057 "structure"
brain<-m[,1]
#~get "/usr/rab/.Data/m" 542035057 "any"
#~put "/usr/rab/.Data/brain" 542035066 "real"
body<-m[,2]
#~get "/usr/rab/.Data/m" 542035057 "any"
#~put "/usr/rab/.Data/body" 542035072 "real"
plot(body,brain)
#~get "/usr/rab/.Data/body" 542035072 "any"
#~get "/usr/rab/.Data/brain" 542035066 "any"
```

What is recorded in the S AUDIT file:

- Top-level commands
- Data objects read
- Data objects written

# Provenance-Aware Computing Today

## Recent Timeline

- **2006** IPAW'06 International Provenance and Annotation Workshop
- **2006** First Provenance Challenge
- **2006** Second Provenance Challenge
- **2007** Open Provenance Model (OPM) Draft
- **2008** IPAW'08 and OPM Workshop
- **2009** Third Provenance Challenge

The primary goal of the Third Provenance Challenge was to evaluate the Open Provenance Model.

# Provenance-Aware Computing Today

## Recent Timeline

- **2006** IPAW'06 International Provenance and Annotation Workshop
- **2006** First Provenance Challenge
- **2006** Second Provenance Challenge
- **2007** Open Provenance Model (OPM) Draft
- **2008** IPAW'08 and OPM Workshop
- **2009** Third Provenance Challenge

The primary goal of the Third Provenance Challenge was to evaluate the Open Provenance Model.

# Provenance-Aware Computing Today

## Recent Timeline

- **2006** IPAW'06 International Provenance and Annotation Workshop
- **2006** First Provenance Challenge
- **2006** Second Provenance Challenge
- **2007** Open Provenance Model (OPM) Draft
- **2008** IPAW'08 and OPM Workshop
- **2009** Third Provenance Challenge

The primary goal of the Third Provenance Challenge was to evaluate the Open Provenance Model.

# Provenance-Aware Computing Today

## Recent Timeline

- **2006** IPAW'06 International Provenance and Annotation Workshop
- **2006** First Provenance Challenge
- **2006** Second Provenance Challenge
- **2007** Open Provenance Model (OPM) Draft
- **2008** IPAW'08 and OPM Workshop
- **2009** Third Provenance Challenge

The primary goal of the Third Provenance Challenge was to evaluate the Open Provenance Model.

# Provenance-Aware Computing Today

## Recent Timeline

- **2006** IPAW'06 International Provenance and Annotation Workshop
- **2006** First Provenance Challenge
- **2006** Second Provenance Challenge
- **2007** Open Provenance Model (OPM) Draft
- **2008** IPAW'08 and OPM Workshop
- **2009** Third Provenance Challenge

The primary goal of the Third Provenance Challenge was to evaluate the Open Provenance Model.

# Provenance-Aware Computing Today

## Recent Timeline

- **2006** IPAW’06 International Provenance and Annotation Workshop
- **2006** First Provenance Challenge
- **2006** Second Provenance Challenge
- **2007** Open Provenance Model (OPM) Draft
- **2008** IPAW’08 and OPM Workshop
- **2009** Third Provenance Challenge

The primary goal of the Third Provenance Challenge was to evaluate the Open Provenance Model.

# Provenance-Aware Computing Today

## Recent Timeline

- **2006** IPAW'06 International Provenance and Annotation Workshop
- **2006** First Provenance Challenge
- **2006** Second Provenance Challenge
- **2007** Open Provenance Model (OPM) Draft
- **2008** IPAW'08 and OPM Workshop
- **2009** Third Provenance Challenge

The primary goal of the Third Provenance Challenge was to evaluate the Open Provenance Model.

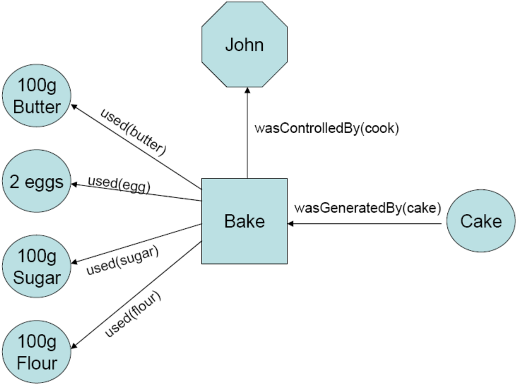
# Open Provenance Model

The OPM has been designed to meet the following requirements:

- To allow provenance information to be exchanged between systems;
- To allow developers to build and share tools that operate on such a model;
- To be technology-agnostic;
- Support a digital representation of provenance for any "thing", produced by computer systems or not;
- Define rules that identify valid inferences on provenance graphs.

# Open Provenance Model

## Example: Victoria Sponge Cake Provenance



Entities

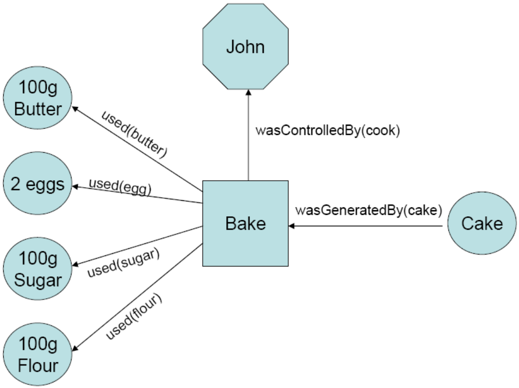
- Artifacts: Cake, 100g butter, 2 eggs, 100g sugar, 100g flour
- Processes: Bake
- Agents: John

Causal Relationships

- wasGeneratedBy
- wasControlledBy
- used

# Open Provenance Model

## Example: Victoria Sponge Cake Provenance



Entities

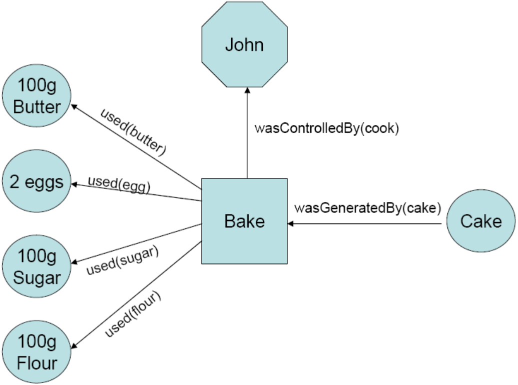
- Artifacts: Cake, 100g butter, 2 eggs, 100g sugar, 100g flour
- Processes: Bake
- Agents: John

Causal Relationships

- wasGeneratedBy
- wasControlledBy
- used

# Open Provenance Model

## Example: Victoria Sponge Cake Provenance



Entities

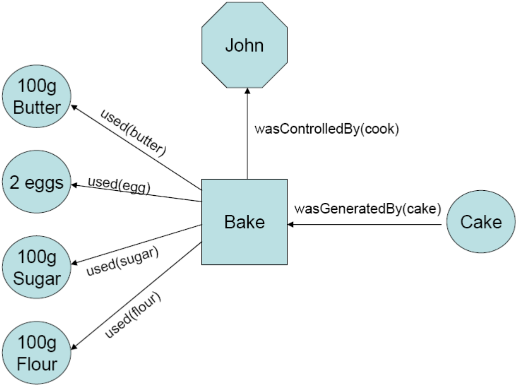
- Artifacts: Cake, 100g butter, 2 eggs, 100g sugar, 100g flour
- Processes: Bake
- Agents: John

Causal Relationships

- wasGeneratedBy
- wasControlledBy
- used

# Open Provenance Model

## Example: Victoria Sponge Cake Provenance



Entities

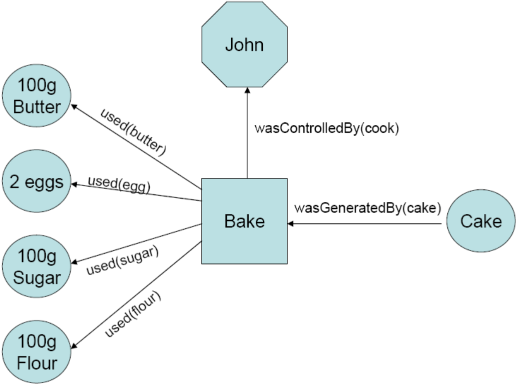
- Artifacts: Cake, 100g butter, 2 eggs, 100g sugar, 100g flour
- Processes: Bake
- Agents: John

Causal Relationships

- wasGeneratedBy
- wasControlledBy
- used

# Open Provenance Model

## Example: Victoria Sponge Cake Provenance



Entities

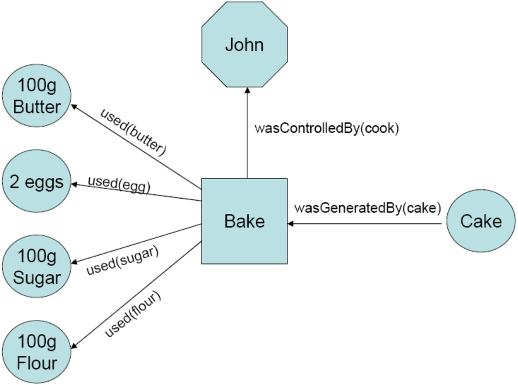
- Artifacts: Cake, 100g butter, 2 eggs, 100g sugar, 100g flour
- Processes: Bake
- Agents: John

Causal Relationships

- wasGeneratedBy
- wasControlledBy
- used

# Open Provenance Model

## Example: Victoria Sponge Cake Provenance



Entities

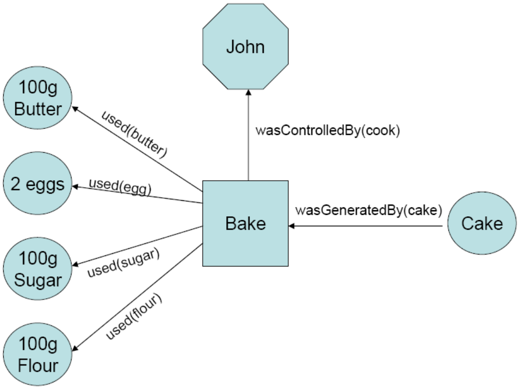
- Artifacts: Cake, 100g butter, 2 eggs, 100g sugar, 100g flour
- Processes: Bake
- Agents: John

Causal Relationships

- wasGeneratedBy
- wasControlledBy
- used

# Open Provenance Model

## Example: Victoria Sponge Cake Provenance



Entities

- Artifacts: Cake, 100g butter, 2 eggs, 100g sugar, 100g flour
- Processes: Bake
- Agents: John

Causal Relationships

- wasGeneratedBy
- wasControlledBy
- used

# The CXXR Project

Founded in 2007, CXXR[2] aims to progressively reengineer the R interpreter from C into C++, with the intention that:

- Full functionality of the standard R distribution is preserved;
- The behaviour of R code is unaffected (unless it probes into the interpreter internals);
- The primary interfaces between the interpreter and C and Fortran code are as far as possible unaffected.

CXXR is intended to make it easier to produce experimental versions of the R interpreter.

[2] www.cs.kent.ac.uk/projects/cxxr

# The CXXR Project

Founded in 2007, CXXR[2] aims to progressively reengineer the R interpreter from C into C++, with the intention that:

- Full functionality of the standard R distribution is preserved;
- The behaviour of R code is unaffected (unless it probes into the interpreter internals);
- The primary interfaces between the interpreter and C and Fortran code are as far as possible unaffected.

CXXR is intended to make it easier to produce experimental versions of the R interpreter.

[2] www.cs.kent.ac.uk/projects/cxxr

# The CXXR Project

Founded in 2007, CXXR[2] aims to progressively reengineer the R interpreter from C into C++, with the intention that:

- Full functionality of the standard R distribution is preserved;
- The behaviour of R code is unaffected (unless it probes into the interpreter internals);
- The primary interfaces between the interpreter and C and Fortran code are as far as possible unaffected.

CXXR is intended to make it easier to produce experimental versions of the R interpreter.

[2] www.cs.kent.ac.uk/projects/cxxr

# The CXXR Project

Founded in 2007, CXXR[2] aims to progressively reengineer the R interpreter from C into C++, with the intention that:

- Full functionality of the standard R distribution is preserved;
- The behaviour of R code is unaffected (unless it probes into the interpreter internals);
- The primary interfaces between the interpreter and C and Fortran code are as far as possible unaffected.

CXXR is intended to make it easier to produce experimental versions of the R interpreter.

[2] www.cs.kent.ac.uk/projects/cxxr

# The CXXR Project

Founded in 2007, CXXR[2] aims to progressively reengineer the R interpreter from C into C++, with the intention that:

- Full functionality of the standard R distribution is preserved;
- The behaviour of R code is unaffected (unless it probes into the interpreter internals);
- The primary interfaces between the interpreter and C and Fortran code are as far as possible unaffected.

CXXR is intended to make it easier to produce experimental versions of the R interpreter.

[2] www.cs.kent.ac.uk/projects/cxxr

# Environments and Bindings

During the evaluation of:

```
x <- 5
```

- *x* is a symbol
- *5* is a vector value
- A binding associates a value with a symbol
- This binding is stored in the global environment
- CXXR provides hooks on bindings, allowing callbacks on
  - Read, i.e. when an object is looked-up in the global environment
  - Write, i.e. when a symbol-to-value binding is created

# Environments and Bindings

During the evaluation of:

```
x <- 5
```

- *x* is a symbol
- *5* is a vector value
- A binding associates a value with a symbol
- This binding is stored in the global environment
- CXXR provides hooks on bindings, allowing callbacks on
  - Read, i.e. when an object is looked-up in the global environment
  - Write, i.e. when a symbol-to-value binding is created

# Environments and Bindings

During the evaluation of:

```
x <- 5
```

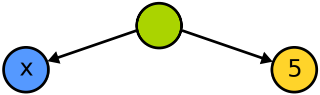
- *x* is a symbol
- *5* is a vector value
- A binding associates a value with a symbol
- This binding is stored in the global environment
- CXXR provides hooks on bindings, allowing callbacks on
  - Read, i.e. when an object is looked-up in the global environment
  - Write, i.e. when a symbol-to-value binding is created

# Environments and Bindings

During the evaluation of:

```
x <- 5
```

- *x* is a symbol
- *5* is a vector value
- A binding associates a value with a symbol
- This binding is stored in the global environment
- CXXR provides hooks on bindings, allowing callbacks on
  - Read, i.e. when an object is looked-up in the global environment
  - Write, i.e. when a symbol-to-value binding is created

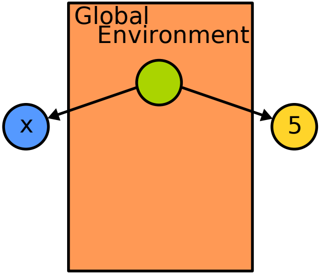


# Environments and Bindings

During the evaluation of:

```
x <- 5
```

- *x* is a symbol
- *5* is a vector value
- A binding associates a value with a symbol
- This binding is stored in the global environment
- CXXR provides hooks on bindings, allowing callbacks on
  - Read, i.e. when an object is looked-up in the global environment
  - Write, i.e. when a symbol-to-value binding is created

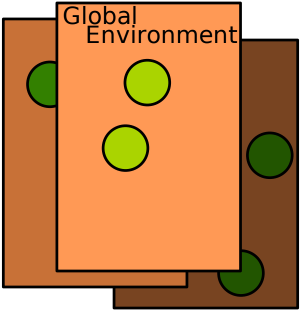


# Environments and Bindings

During the evaluation of:

```
x <- 5
```

- *x* is a symbol
- *5* is a vector value
- A binding associates a value with a symbol
- This binding is stored in the global environment
- CXXR provides hooks on bindings, allowing callbacks on
  - Read, i.e. when an object is looked-up in the global environment
  - Write, i.e. when a symbol-to-value binding is created

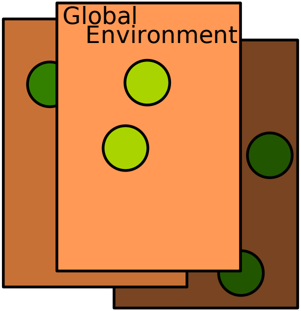


# Environments and Bindings

During the evaluation of:

```
x <- 5
```

- *x* is a symbol
- *5* is a vector value
- A binding associates a value with a symbol
- This binding is stored in the global environment
- CXXR provides hooks on bindings, allowing callbacks on
  - Read, i.e. when an object is looked-up in the global environment
  - Write, i.e. when a symbol-to-value binding is created



# Objectives

Why record provenance?

- Auditing, and accountability
- Informative to the user
- Enabling reproducibility
- Understand how objects are used
    - For instance, identifying all objects which used a given function

# Objectives

Why record provenance?

- Auditing, and accountability
- Informative to the user
- Enabling reproducibility
- Understand how objects are used
  - For instance, identifying all objects which used a given function

# Objectives

Why record provenance?

- Auditing, and accountability
- Informative to the user
- Enabling reproducibility
- Understand how objects are used
    - For instance, identifying all objects which used a given function

# Objectives

Why record provenance?

- Auditing, and accountability
- Informative to the user
- Enabling reproducibility
- Understand how objects are used
    - For instance, identifying all objects which used a given function

# Objectives

Why record provenance?

- Auditing, and accountability
- Informative to the user
- Enabling reproducibility
- Understand how objects are used
  - For instance, identifying all objects which used a given function

# Objectives

Why record provenance?

- Auditing, and accountability
- Informative to the user
- Enabling reproducibility
- Understand how objects are used
    - For instance, identifying all objects which used a given function

# Objectives

Why record provenance?

- Auditing, and accountability
- Informative to the user
- Enabling reproducibility
- Understand how objects are used
    - For instance, identifying all objects which used a given function

What provenance to record?
We want to identify, of a given object:

- Pedigree: The series of commands issued
- Parents: Objects which have been read during its creation
- Children: Objects which have read it during their creation

# Objectives

Why record provenance?

- Auditing, and accountability
- Informative to the user
- Enabling reproducibility
- Understand how objects are used
  - For instance, identifying all objects which used a given function

What provenance to record?
We want to identify, of a given object:

- Pedigree: The series of commands issued
- Parents: Objects which have been read during its creation
- Children: Objects which have read it during their creation

# Objectives

Why record provenance?

- Auditing, and accountability
- Informative to the user
- Enabling reproducibility
- Understand how objects are used
    - For instance, identifying all objects which used a given function

What provenance to record?
We want to identify, of a given object:

- Pedigree: The series of commands issued
- Parents: Objects which have been read during its creation
- Children: Objects which have read it during their creation

# Objectives

Why record provenance?

- Auditing, and accountability
- Informative to the user
- Enabling reproducibility
- Understand how objects are used
  - For instance, identifying all objects which used a given function

What provenance to record?
We want to identify, of a given object:

- Pedigree: The series of commands issued
- Parents: Objects which have been read during its creation
- Children: Objects which have read it during their creation

# Objectives

Why record provenance?

- Auditing, and accountability
- Informative to the user
- Enabling reproducibility
- Understand how objects are used
  - For instance, identifying all objects which used a given function

What provenance to record?
We want to identify, of a given object:

- Pedigree: The series of commands issued
- Parents: Objects which have been read during its creation
- Children: Objects which have read it during their creation

# Strategy

What we need to go about this:

- A mechanism for trapping reads and writes in the user workspace (i.e. the global environment)
  - Recall that CXXR provides monitor hooks on access and mutation of bindings
- Containers for storing provenance information
- New R commands for inspecting provenance
  - provenance(x): Returns a list comprising: expression, symbol, timestamp, parents, children
  - pedigree(x): Displays the sequence of commands issued, which results in x's current state

# Strategy

What we need to go about this:

- A mechanism for trapping reads and writes in the user workspace (i.e. the global environment)
  - Recall that CXXR provides monitor hooks on access and mutation of bindings
- Containers for storing provenance information
- New R commands for inspecting provenance
  - provenance(x): Returns a list comprising: expression, symbol, timestamp, parents, children
  - pedigree(x): Displays the sequence of commands issued, which results in x's current state

# Strategy

What we need to go about this:

- A mechanism for trapping reads and writes in the user workspace (i.e. the global environment)
  - Recall that CXXR provides monitor hooks on access and mutation of bindings
- Containers for storing provenance information
- New R commands for inspecting provenance
  - provenance(x): Returns a list comprising: expression, symbol, timestamp, parents, children
  - pedigree(x): Displays the sequence of commands issued, which results in x's current state

# Associating Provenance with Bindings

- When an object is read from:
  - It is recorded in a Parentage
- When an object is written to:
  - A Provenance object is created, comprising:
  - This Provenance object is then associated with the relevant binding
  - Functions assigned in the global environment are also handled in this way
  - Therefore objects resulting from function calls have the function as a parent

# Associating Provenance with Bindings

- When an object is read from:
  - It is recorded in a Parentage
- When an object is written to:
  - A Provenance object is created, comprising:
    - The being evaluated
    - The current
    - The being written to
    - This objects'
  - This Provenance object is then associated with the relevant binding
  - Functions assigned in the global environment are also handled in this way
  - Therefore objects resulting from function calls have the function as a parent

# Associating Provenance with Bindings

- When an object is read from:
  - It is recorded in a Parentage
- When an object is written to:
  - A Provenance object is created, comprising:
    - The top level expression being evaluated
    - The current timestamp
    - The symbol being written to
    - This objects' parentage
  - This Provenance object is then associated with the relevant binding
  - Functions assigned in the global environment are also handled in this way
  - Therefore objects resulting from function calls have the function as a parent

# Associating Provenance with Bindings

- When an object is read from:
  - It is recorded in a Parentage
- When an object is written to:
  - A Provenance object is created, comprising:
    - The top level expression being evaluated
    - The current timestamp
    - The symbol being written to
    - This objects' parentage
  - This Provenance object is then associated with the relevant binding
  - Functions assigned in the global environment are also handled in this way
  - Therefore objects resulting from function calls have the function as a parent

# Associating Provenance with Bindings

- When an object is read from:
  - It is recorded in a Parentage
- When an object is written to:
  - A Provenance object is created, comprising:
    - The top level expression being evaluated
    - The current timestamp
    - The symbol being written to
    - This objects' parentage
  - This Provenance object is then associated with the relevant binding
  - Functions assigned in the global environment are also handled in this way
  - Therefore objects resulting from function calls have the function as a parent

# Associating Provenance with Bindings

- When an object is read from:
    - It is recorded in a Parentage
- When an object is written to:
    - A Provenance object is created, comprising:
        - The top level expression being evaluated
        - The current timestamp
        - The symbol being written to
        - This objects' parentage
    - This Provenance object is then associated with the relevant binding
    - Functions assigned in the global environment are also handled in this way
    - Therefore objects resulting from function calls have the function as a parent

# Associating Provenance with Bindings

- When an object is read from:
  - It is recorded in a Parentage
- When an object is written to:
  - A Provenance object is created, comprising:
    - The top level expression being evaluated
    - The current timestamp
    - The symbol being written to
    - This objects' parentage
  - This Provenance object is then associated with the relevant binding
  - Functions assigned in the global environment are also handled in this way
  - Therefore objects resulting from function calls have the function as a parent

# Associating Provenance with Bindings

- When an object is read from:
  - It is recorded in a Parentage
- When an object is written to:
  - A Provenance object is created, comprising:
    - The top level expression being evaluated
    - The current timestamp
    - The symbol being written to
    - This objects' parentage
  - This Provenance object is then associated with the relevant binding
  - Functions assigned in the global environment are also handled in this way
  - Therefore objects resulting from function calls have the function as a parent

# Associating Provenance with Bindings

- When an object is read from:
  - It is recorded in a Parentage
- When an object is written to:
  - A Provenance object is created, comprising:
    - The top level expression being evaluated
    - The current timestamp
    - The symbol being written to
    - This objects' parentage
  - This Provenance object is then associated with the relevant binding
  - Functions assigned in the global environment are also handled in this way
  - Therefore objects resulting from function calls have the function as a parent

# Associating Provenance with Bindings

- When an object is read from:
  - It is recorded in a Parentage
- When an object is written to:
  - A Provenance object is created, comprising:
    - The top level expression being evaluated
    - The current timestamp
    - The symbol being written to
    - This objects' parentage
  - This Provenance object is then associated with the relevant binding
  - Functions assigned in the global environment are also handled in this way
  - Therefore objects resulting from function calls have the function as a parent

# Associating Provenance with Bindings

- When an object is read from:
  - It is recorded in a Parentage
- When an object is written to:
  - A Provenance object is created, comprising:
    - The top level expression being evaluated
    - The current timestamp
    - The symbol being written to
    - This objects' parentage
  - This Provenance object is then associated with the relevant binding
  - Functions assigned in the global environment are also handled in this way
  - Therefore objects resulting from function calls have the function as a parent

# Where were we?

## Recall our session...

```
> ls()
```

# Where were we?

## Recall our session...

```
> ls()
[1] "body"   "brain"  "lbody"  "lbrain" "r"
```

# Where were we?

## Recall our session...

```
> ls()
[1] "body"   "brain"  "lbody"  "lbrain" "r"
> provenance(body)
```

# Where were we?

## Recall our session...

```
> ls()
[1] "body"   "brain"  "lbody"  "lbrain" "r"
> provenance(body)
$command
body <- mammals[, 1]

$symbol
body

$timestamp
[1] "07/03/2009 11:33:49 AM.763807"

$parents
NULL

$children
[1] "lbody"
```

# Where were we?

## Recall our session...

```
> ls()
[1] "body"   "brain"  "lbody"  "lbrain" "r"
> provenance(lbrain)
```

# Where were we?

## Recall our session...

```
> ls()
[1] "body"   "brain"  "lbody"  "lbrain" "r"
> provenance(lbrain)
$command
lbrain <- log(brain)

$symbol
lbrain

$timestamp
[1] "07/03/2009 11:33:54 AM.221827"

$parents
[1] "brain"

$children
[1] "r"
```

# Where were we?

## Recall our session...

```
> ls()
[1] "body"   "brain"  "lbody"  "lbrain" "r"
> provenance(r)
```

# Where were we?

## Recall our session...

```
> ls()
[1] "body"   "brain"  "lbody"  "lbrain" "r"
> provenance(r)
$command
r <- lm(lbrain ~ lbody)

$symbol
r

$timestamp
[1] "07/03/2009 11:34:04 AM.117156"

$parents
[1] "lbrain" "lbody"

$children
NULL
```

# Where were we?

## Recall our session...

```
> ls()
[1] "body"   "brain"  "lbody"  "lbrain" "r"
> pedigree(r)
```

# Where were we?

## Recall our session...

```
> ls()
[1] "body"   "brain"  "lbody"  "lbrain" "r"
> pedigree(r)
brain <- mammals[, 2]
body <- mammals[, 1]
lbrain <- log(brain)
lbody <- log(body)
r <- lm(lbrain ~ lbody)
```

# A Further Example

## Function Provenance

```
> sq <- function(x) { x*x }
```

# A Further Example

## Function Provenance

```
> sq <- function(x) { x*x }
> three <- 3
```

# A Further Example

## Function Provenance

```
> sq <- function(x) { x*x }
> three <- 3
> nine <- square(three)
```

# A Further Example

## Function Provenance

```
> sq <- function(x) { x*x }
> three <- 3
> nine <- square(three)
> provenance(nine)$parents
```

# A Further Example

## Function Provenance

```
> sq <- function(x) { x*x }
> three <- 3
> nine <- square(three)
> provenance(nine)$parents
[1] "sq"    "three"
```

# A Further Example

## Function Provenance

```
> sq <- function(x) { x*x }
> three <- 3
> nine <- square(three)
> provenance(nine)$parents
[1] "sq"    "three"
> provenance(sq)$children
```

# A Further Example

## Function Provenance

```
> sq <- function(x) { x*x }
> three <- 3
> nine <- square(three)
> provenance(nine)$parents
[1] "sq"    "three"
> provenance(sq)$children
[1] "nine"
```

# Conclusion and Future Work

We have demonstrated that it is possible to introduce provenance tracking facilities to a statistical environment, and as a result we can identify an object's pedigree, parents and children.
We now need to look into the following

- Reproducing objects from provenance information
- Effectively handle pseudo-random number generation
  - To enable reproducibility of results
- Tracking provenance in other R environments
  - Packages
  - Attached data frames
  - Functions
- Serializing provenance information
  - To enable cross-session provenance-tracking

# Conclusion and Future Work

We have demonstrated that it is possible to introduce provenance tracking facilities to a statistical environment, and as a result we can identify an object's pedigree, parents and children.

We now need to look into the following

- Reproducing objects from provenance information
- Effectively handle pseudo-random number generation
  - To enable reproducibility of results
- Tracking provenance in other R environments
  - Packages
  - Attached data frames
  - Functions
- Serializing provenance information
  - To enable cross-session provenance-tracking

# Conclusion and Future Work

We have demonstrated that it is possible to introduce provenance tracking facilities to a statistical environment, and as a result we can identify an object's pedigree, parents and children.

We now need to look into the following

- Reproducing objects from provenance information
- Effectively handle pseudo-random number generation
  - To enable reproducibility of results
- Tracking provenance in other R environments
  - Packages
  - Attached data frames
  - Functions
- Serializing provenance information
  - To enable cross-session provenance-tracking

## Conclusion and Future Work

We have demonstrated that it is possible to introduce provenance tracking facilities to a statistical environment, and as a result we can identify an object's pedigree, parents and children.

We now need to look into the following

- Reproducing objects from provenance information
- Effectively handle pseudo-random number generation
  - To enable reproducibility of results
- Tracking provenance in other R environments
  - Packages
  - Attached data frames
  - Functions
- Serializing provenance information
  - To enable cross-session provenance-tracking

# Conclusion and Future Work

We have demonstrated that it is possible to introduce provenance tracking facilities to a statistical environment, and as a result we can identify an object's pedigree, parents and children.

We now need to look into the following

- Reproducing objects from provenance information
- Effectively handle pseudo-random number generation
  - To enable reproducibility of results
- Tracking provenance in other R environments
  - Packages
  - Attached data frames
  - Functions
- Serializing provenance information
  - To enable cross-session provenance-tracking