I Am ALTREP (And So Can You!)

Gabriel Becker, Work Joint with L Tierney, M Lawrence and T Kalibera

The Year: R v0-3.4 (1997- 2018)

You Know What The R C API Needs?

~ Me probably, circa 2016

An R Vector Viewed From C

Two primary sections

- SEXP header

An R Vector Viewed From C

Two primary sections

- SEXP header
  - Length

An R Vector Viewed From C

Two primary sections

- SEXP header
  - Length
  - SEXP type

An R Vector Viewed From C

Two primary sections

- SEXP header
  - Length
  - SEXP type
  - Various other info

An R Vector Viewed From C

Two primary sections

- SEXP header
  - Length
  - SEXP type
  - Various other info
- Payload (Data)

An R Vector Viewed From C

Two primary sections

- SEXP header
  - Length
  - SEXP type
  - Various other info
- Payload (Data)
  - The values of the vector elements

Tightly Coupled

Atomic vector objects were *tightly coupled* with their data

- header+payload contiguous in memory

Tightly Coupled

Atomic vector objects were *tightly coupled* with their data

- header+payload contiguous in memory
- payload data in simple array format

An Incomplete History of (Not) Duplicating Data in R $< 3.6.0$

Copy on Write

- Pass-by-value semantics

Copy on Write

- Pass-by-value semantics
  - R *behaves* as if it duplicates data every time it is

Copy on Write

- Pass-by-value semantics
  - R *behaves* as if it duplicates data every time it is
    - Assigned to a new variable

Copy on Write

- Pass-by-value semantics
  - R *behaves* as if it duplicates data every time it is
    - Assigned to a new variable
    - Passed passed as argument to function

Copy on Write

- Pass-by-value semantics
  - R *behaves* as if it duplicates data every time it is
    - Assigned to a new variable
    - Passed passed as argument to function
  - Without actually duplicating

Copy on Write

- Pass-by-value semantics
  - R *behaves* as if it duplicates data every time it is
    - Assigned to a new variable
    - Passed passed as argument to function
  - Without actually duplicating
- Only matters how many pointers we have to it if it changes

Copy on Write

- Pass-by-value semantics
  - R *behaves* as if it duplicates data every time it is
    - Assigned to a new variable
    - Passed passed as argument to function
  - Without actually duplicating
- Only matters how many pointers we have to it if it changes
  - Duplicate then, and only then

Shallow Duplication

- Lists, S4 objects are “separable”

Shallow Duplication

- Lists, S4 objects are “separable”
  - Modifying elements forces duplication of only those elements

Shallow Duplication

- Lists, S4 objects are “separable”
  - Modifying elements forces duplication of only those elements
  - Introduced in R 3.1.0, Michael Lawrence and R-Core

Shallow Duplication

- Lists, S4 objects are “separable”
  - Modifying elements forces duplication of only those elements
  - Introduced in R 3.1.0, Michael Lawrence and R-Core
  - Modifying attributes duplicates only the “container” list

Deep Duplication

- Atomic vectors were *not* separable

Deep Duplication

- Atomic vectors were *not* separable
  - Modifying any element forces full data duplication

Deep Duplication

- Atomic vectors were *not* separable
  - Modifying any element forces full data duplication
  - Modifying attributes forces full data duplication

This Worked Well, Obviously

But there were limitations

- No way for compressed/shared/out of core data to interact with R internals

But there were limitations

- No way for compressed/shared/out of core data to interact with R internals
- Full duplication on modifying of atomic vector attributes

But there were limitations

- No way for compressed/shared/out of core data to interact with R internals
- Full duplication on modifying of atomic vector attributes
- No way for vectors to retain information about themselves

But there were limitations

- No way for compressed/shared/out of core data to interact with R internals
- Full duplication on modifying of atomic vector attributes
- No way for vectors to retain information about themselves
  - Sortedness, presence of NAs, etc

The Idea of ALTREP

Atomic Vectors, By Way Of

Design Intent

- Generalize storage of data payload for atomic vector SEXPs

Design Intent

- Generalize storage of data payload for atomic vector SEXPs
- Implement “Smart Vectors”

Design Intent

- Generalize storage of data payload for atomic vector SEXPs
- Implement “Smart Vectors”
- Decouple data and attributes

Design Intent

- Generalize storage of data payload for atomic vector SEXPs
- Implement "Smart Vectors"
- Decouple data and attributes
- *Completely transparent* at the R level

Generalized Data Storage

- Location

Generalized Data Storage

- Location
  - In memory

Generalized Data Storage

- Location
  - In memory
  - Out of core

Generalized Data Storage

- Location
  - In memory
  - Out of core
  - Owned by another process/object

Generalized Data Storage

- Location
  - In memory
  - Out of core
  - Owned by another process/object
- Format

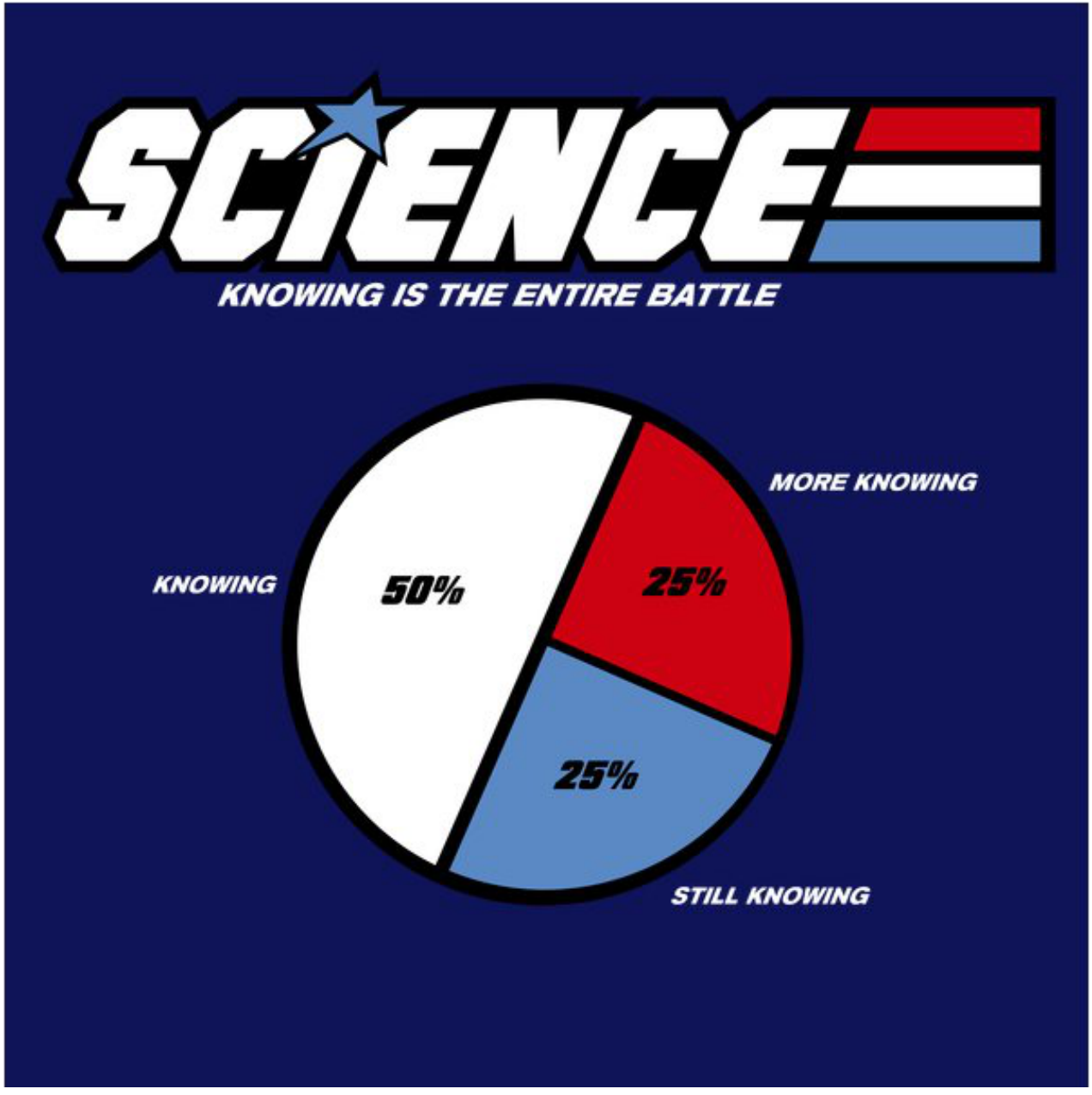
Generalized Data Storage

- Location
  - In memory
  - Out of core
  - Owned by another process/object
- Format
  - Efficient representations

Generalized Data Storage

- Location
  - In memory
  - Out of core
  - Owned by another process/object
- Format
  - Efficient representations
  - E.g., compact integer/real sequences

Smart Vectors



Smart Vectors

- Know metadata about themselves

Smart Vectors

- Know metadata about themselves
  - sortedness

Smart Vectors

- Know metadata about themselves
  - sortedness
  - lack of NAs

Smart Vectors

- Know metadata about themselves
  - sortedness
  - lack of NAs
- Makes certain computations very easy

Smart Vectors

- Know metadata about themselves
  - sortedness
  - lack of NAs
- Makes certain computations very easy
- **Fully compatible with R internals**

Decoupling Attributes and Data

- No reason to copy data when just changing object class

Decoupling Attributes and Data

- No reason to copy data when just changing object class
- Originally “stretch goal”

Decoupling Attributes and Data

- No reason to copy data when just changing object class
- Originally “stretch goal”
  - Implemented by Luke for 3.6.0 for vectors $>$ certain size

How to Spot an ALTREP - R Code Edition

ALTREP R Objects Are Just R Objects

- R code should never know the difference

ALTREP R Objects Are Just R Objects

- R code should never know the difference
- "normal" C code should not know the difference

ALTREP R Objects Are Just R Objects

- R code should never know the difference
- "normal" C code should not know the difference
  - exception: hooks to call ALTREP methods

How?

- ALTREP framework implements an abstraction *underneath* traditional R C API

How?

- ALTREP framework implements an abstraction *underneath* traditional R C API
  - Generalizes whats underneath the API

How?

- ALTREP framework implements an abstraction *underneath* traditional R C API
  - Generalizes whats underneath the API
    - Without changing how data are accessed

How?

- ALTREP framework implements an abstraction *underneath* traditional R C API
  - Generalizes whats underneath the API
    - Without changing how data are accessed
  - Compatible with all C code which uses the API

How?

- ALTREP framework implements an abstraction *underneath* traditional R C API
  - Generalizes whats underneath the API
    - Without changing how data are accessed
  - Compatible with all C code which uses the API
  - Compatible with R internals

The Deets

ALTREP

One Bit To Rule Them All

- Named bit `alt` in header struct that SEXP is an ALTREP

One Bit To Rule Them All

- Named bit `alt` in header struct that SEXP is an ALTREP
  - `ALTREP(x)` function checks the bit

One Bit To Rule Them All

- Named bit `alt` in header struct that SEXP is an ALTREP
  - `ALTREP(x)` function checks the bit
  - `SETALTREP(x,v)` not provided ... don't do that

All ALTREP Objects are defined by 3 SEXP fields

- Data 1

All ALTREP Objects are defined by 3 SEXP fields

- Data 1
  - `R_altrep_data1` and `R_set_altrep_data1`

All ALTREP Objects are defined by 3 SEXP fields

- Data 1
  - `R_altrep_data1` and `R_set_altrep_data1`
  - "Usually" the alternative representant

All ALTREP Objects are defined by 3 SEXP fields

- Data 1
  - `R_altrep_data1` and `R_set_altrep_data1`
  - "Usually" the alternative representant
- Data 2

All ALTREP Objects are defined by 3 SEXP fields

- Data 1
  - `R_altrep_data1` and `R_set_altrep_data1`
  - "Usually" the alternative representant
- Data 2
  - `R_altrep_data2` and `R_set_altrep_data2`

All ALTREP Objects are defined by 3 SEXP fields

- Data 1
  - `R_altrep_data1` and `R_set_altrep_data1`
  - "Usually" the alternative representant
- Data 2
  - `R_altrep_data2` and `R_set_altrep_data2`
  - "Often" placeholder for "Expanded" version

All ALTREP Objects are defined by 3 SEXP fields

- Data 1
  - `R_altrep_data1` and `R_set_altrep_data1`
  - "Usually" the alternative representant
- Data 2
  - `R_altrep_data2` and `R_set_altrep_data2`
  - "Often" placeholder for "Expanded" version
- ALTREP Class

All ALTREP Objects are defined by 3 SEXP fields

- Data 1
  - `R_altrep_data1` and `R_set_altrep_data1`
  - "Usually" the alternative representant
- Data 2
  - `R_altrep_data2` and `R_set_altrep_data2`
  - "Often" placeholder for "Expanded" version
- ALTREP Class
  - Contains method table

All ALTREP Objects are defined by 3 SEXP fields

- Data 1
  - `R_altrep_data1` and `R_set_altrep_data1`
  - "Usually" the alternative representant
- Data 2
  - `R_altrep_data2` and `R_set_altrep_data2`
  - "Often" placeholder for "Expanded" version
- ALTREP Class
  - Contains method table
  - `R_altrep_inherits` only API provided, **no** getter/setter

All ALTREP Objects are defined by 3 SEXP fields

- Data 1
  - `R_altrep_data1` and `R_set_altrep_data1`
  - "Usually" the alternative representant
- Data 2
  - `R_altrep_data2` and `R_set_altrep_data2`
  - "Often" placeholder for "Expanded" version
- ALTREP Class
  - Contains method table
  - `R_altrep_inherits` only API provided, **no** getter/setter
- Currently Implemented as CONS cells, **but this may change without warning**

How R Internals Interact With Vectors

Overview

- Access data (payload)

Overview

- Access data (payload)
- Modify data^^

Overview

- Access data (payload)
- Modify data^^
- Access length

Overview

- Access data (payload)
- Modify data^^
- Access length
- Coerce to another SEXP type

Overview

- Access data (payload)
- Modify data^^
- Access length
- Coerce to another SEXP type
- Duplicate

Overview

- Access data (payload)
- Modify data^^
- Access length
- Coerce to another SEXP type
- Duplicate
- (Un)Serialize

ALTREP Classes

Define Methods Which

- Support all of these actions

ALTREP Classes

Define Methods Which

- Support all of these actions
- Interact with the alternative representation

ALTREP Classes

Define Methods Which

- Support all of these actions
- Interact with the alternative representation
- Provide “escape-hatch” to create non-ALTREP version of themselves

ALTREP Classes

Define Methods Which

- Support all of these actions
- Interact with the alternative representation
- Provide “escape-hatch” to create non-ALTREP version of themselves
  - Or throw error when they would need to

ALTREP Classes

Define Methods Which

- Support all of these actions
- Interact with the alternative representation
- Provide "escape-hatch" to create non-ALTREP version of themselves
  - Or throw error when they would need to
- Remember, ALTREPS should be passable to all R internal functions

Select ALTREP Class Methods

We Are Going Way Down

We Are Going Way Down

- **Always** use provided accessor functions

We Are Going Way Down

- **Always** use provided accessor functions
- **Never** poke around at bits the API doesn't provide access to

We Are Going Way Down

- **Always** use provided accessor functions
- **Never** poke around at bits the API doesn't provide access to
- The API is defined as what is documented in Writing R Extensions

We Are Going Way Down

- **Always** use provided accessor functions
- **Never** poke around at bits the API doesn't provide access to
- The API is defined as what is documented in Writing R Extensions
  - Exception is ALTREP things, not documented there yet

We Are Going Way Down

- **Always** use provided accessor functions
- **Never** poke around at bits the API doesn't provide access to
- The API is defined as what is documented in Writing R Extensions
  - Exception is ALTREP things, not documented there yet
  - Only things starting with `R_altrep` or `R_set_altrep`

We Are Going Way Down

- **Always** use provided accessor functions
- **Never** poke around at bits the API doesn't provide access to
- The API is defined as what is documented in Writing R Extensions
  - Exception is ALTREP things, not documented there yet
  - Only things starting with `R_altrep` or `R_set_altrep`
- **Always** respect MAYBE_SHARED

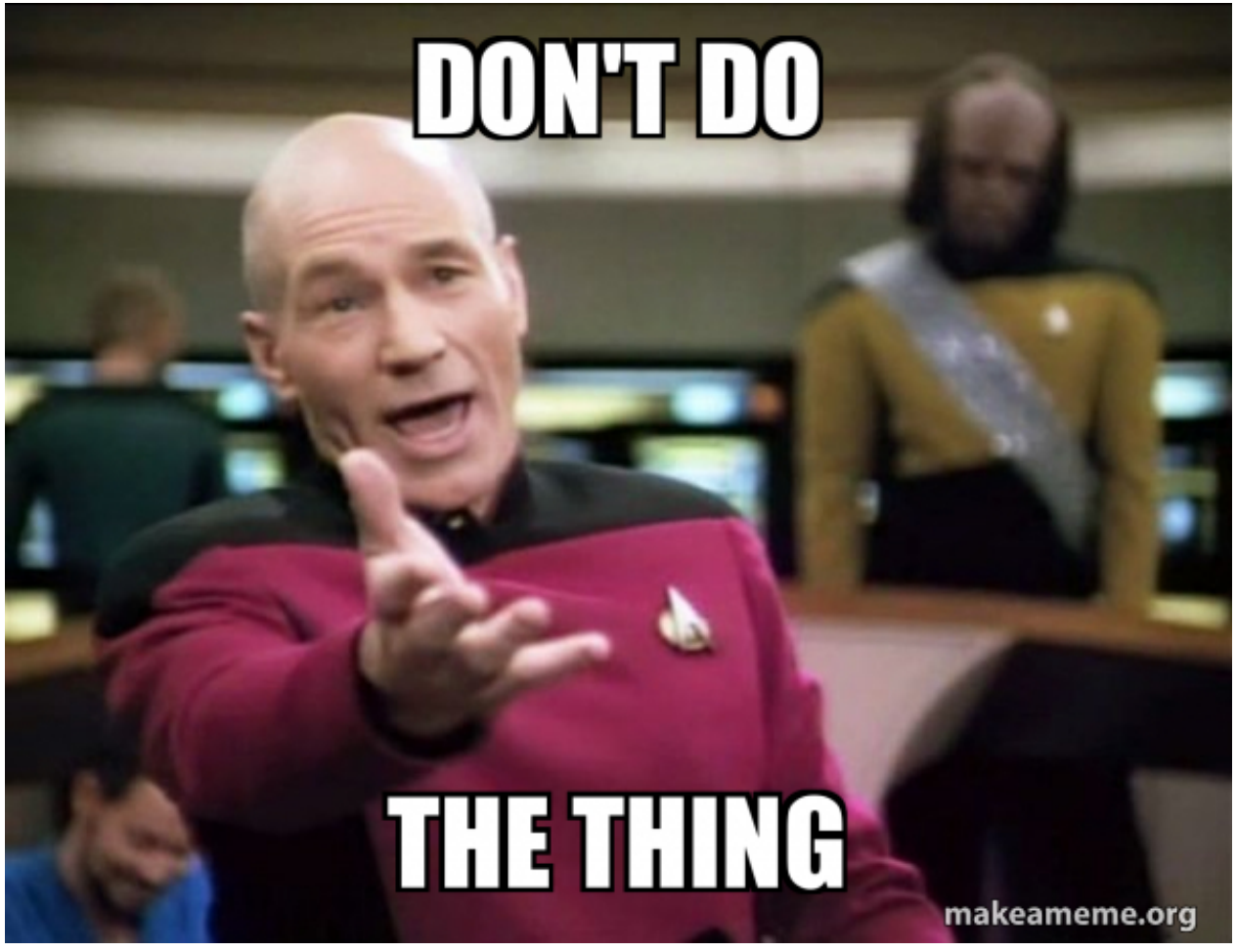
We Are Going Way Down

- **Always** use provided accessor functions
- **Never** poke around at bits the API doesn't provide access to
- The API is defined as what is documented in Writing R Extensions
  - Exception is ALTREP things, not documented there yet
  - Only things starting with `R_altrep` or `R_set_altrep`
- **Always** respect MAYBE_SHARED
  - Your responsibility to duplicate before modification if it returns true

We Are Going Way Down

- **Always** use provided accessor functions
- **Never** poke around at bits the API doesn't provide access to
- The API is defined as what is documented in Writing R Extensions
  - Exception is ALTREP things, not documented there yet
  - Only things starting with `R_altrep` or `R_set_altrep`
- **Always** respect MAYBE_SHARED
  - Your responsibility to duplicate before modification if it returns true
- Don't define `USE_RINTERNALS`

If Someone on R-Core Tells You Not To Do Something in C Code



Duplicate

- `SEXP Duplicate(SEXP x, Rboolean deep)`

Duplicate

- `SEXP Duplicate(SEXP x, Rboolean deep)`
  - **MUST** return a SEXP which is modifiable via DATAPTR or fail

Duplicate

- `SEXP Duplicate(SEXP x, Rboolean deep)`
  - **MUST** return a SEXP which is modifiable via DATAPTR or fail
  - No matter what.

Duplicate

- `SEXP Duplicate(SEXP x, Rboolean deep)`
  - **MUST** return a SEXP which is modifiable via DATAPTR or fail
  - No matter what.
  - Yes, even you.

Dataptr (Mandatory No Default)

- `void *Dataptr(SEXP x, Rboolean writeable)` - Access full data pointer

Dataptr (Mandatory No Default)

- `void *Dataptr(SEXP x, Rboolean writeable)` - Access full data pointer
  - Must **always** return ptr to **full** data in array form (or fail)

Dataptr (Mandatory No Default)

- `void *Dataptr(SEXP x, Rboolean writeable)` - Access full data pointer
  - Must **always** return ptr to **full** data in array form (or fail)
- if `writeable`,

Dataptr (Mandatory No Default)

- `void *Dataptr(SEXP x, Rboolean writeable)` - Access full data pointer
  - Must **always** return ptr to **full** data in array form (or fail)
- if `writeable`,
  - modifications to array data **must** be reflected in R object

Dataptr (Mandatory No Default)

- `void *Dataptr(SEXP x, Rboolean writeable)` - Access full data pointer
  - Must **always** return ptr to **full** data in array form (or fail)
- if `writeable`,
  - modifications to array data **must** be reflected in R object
  - any metadata (sortedness, No_NA) **must** be dropped/set to unknown

Dataptr (Mandatory No Default)

- `void *Dataptr(SEXP x, Rboolean writeable)` - Access full data pointer
  - Must **always** return ptr to **full** data in array form (or fail)
- if `writeable`,
  - modifications to array data **must** be reflected in R object
  - any metadata (sortedness, No_NA) **must** be dropped/set to unknown
  - Often just duplicate into std SEXP vector and use that from now on

Dataptr_or_null

- `const void *Dataptr_or_null(SEXP x)` - Access full data ptr “if thats ok”

Dataptr_or_null

- `const void *Dataptr_or_null(SEXP x)` - Access full data ptr “if thats ok”
  - Return full data ptr if already available

Dataptr_or_null

- `const void *Dataptr_or_null(SEXP x)` - Access full data ptr "if thats ok"
  - Return full data ptr if already available
  - E.g., if `Dataptr` was prev. called with writeable as `TRUE`

Dataptr_or_null

- `const void *Dataptr_or_null(SEXP x)` - Access full data ptr "if thats ok"
  - Return full data ptr if already available
  - E.g., if `Dataptr` was prev. called with writeable as `TRUE`
  - If not already available, return

Dataptr_or_null

- `const void *Dataptr_or_null(SEXP x)` - Access full data ptr "if thats ok"
  - Return full data ptr if already available
  - E.g., if `Dataptr` was prev. called with writeable as `TRUE`
  - If not already available, return
  - `NULL` if your altrep class "doesn't want to" populate full data array

Dataptr_or_null

- `const void *Dataptr_or_null(SEXP x)` - Access full data ptr "if thats ok"
  - Return full data ptr if already available
  - E.g., if `Dataptr` was prev. called with writeable as `TRUE`
  - If not already available, return
  - `NULL` if your altrep class "doesn't want to" populate full data array
  - pointer to full data array

Elt

- `int Elt(SEXP x, R_xlen_t i)`

Elt

- `int Elt(SEXP x, R_xlen_t i)`
  - Return *value* of vector at single position

Sortedness in ALTREP

```
enum {SORTED_DECR_NA_1ST = -2,
      SORTED_DECR = -1,
      UNKNOWN_SORTEDNESS = INT_MIN, /*INT_MIN is NA_INTEGE
      SORTED_INCR = 1,
      SORTED_INCR_NA_1ST = 2,
      KNOWN_UNSORTED = 0};
```

Is_sorted

- `int Is_sorted(SEXP)`

Is_sorted

- `int Is_sorted(SEXP)`
  - Always return an enum value by name

Is_sorted

- `int Is_sorted(SEXP)`
  - Always return an enum value by name
  - **Always** return `UNKNOWN_SORTEDNESS` once `DATAPTR` has been called with `writeable` true

Is_sorted

- `int Is_sorted(SEXP)`
  - Always return an enum value by name
  - **Always** return `UNKNOWN_SORTEDNESS` once `DATAPTR` has been called with `writeable` true
  - `KNOWN_UNSORTED` ***only*** if vector has > 3 distinct values

Is_sorted

- `int Is_sorted(SEXP)`
  - Always return an enum value by name
  - **Always** return `UNKNOWN_SORTEDNESS` once `DATAPTR` has been called with `writeable` true
  - `KNOWN_UNSORTED` ***only*** if vector has > 3 distinct values
  - **and is not sorted in either directoin**

Creating ALTREP Class

```
static void InitVWindowRealClass(DllInfo *dll)
{
    R_altrep_class_t cls =
    R_make_altreal_class("vwindow_real", "vectorwindow", dl

    /* ALTREP methods */
    R_set_altrep_Inspect_method(cls, vwindow_Inspect);
    /* etc */

    /* ALTVEC methods */
    R_set_altvec_Dataptr_method(cls, vwindow_Dataptr);
    /* etc */

    /* ALTREAL methods */
    R_set_altreal_Elt_method(cls, vwindow_real_Elt);
    /* etc */
}
```

ALTREP Writing Guidelines

- Be **extremely** careful and conservative

ALTREP Writing Guidelines

- Be **extremely** careful and conservative
  - metadata returned must be correct **100% of the time**

ALTREP Writing Guidelines

- Be **extremely** careful and conservative
  - metadata returned must be correct **100% of the time**
  - Use the API even in ALTREP method code

ALTREP Writing Guidelines

- Be **extremely** careful and conservative
  - metadata returned must be correct **100% of the time**
  - Use the API even in ALTREP method code
- Methods which return SEXPs can return NULL to decline to do do something

ALTREP Writing Guidelines

- Be **extremely** careful and conservative
  - metadata returned must be correct **100% of the time**
  - Use the API even in ALTREP method code
- Methods which return SEXPs can return NULL to decline to do do something
  - Exception: Duplicate

ALTREP Writing Guidelines

- Be **extremely** careful and conservative
  - metadata returned must be correct **100% of the time**
  - Use the API even in ALTREP method code
- Methods which return SEXPs can return NULL to decline to do do something
  - Exception: Duplicate
- Be *very* wary of violating pass-by-value semantics

ALTREP Writing Guidelines

- Be **extremely** careful and conservative
  - metadata returned must be correct **100% of the time**
  - Use the API even in ALTREP method code
- Methods which return SEXPs can return NULL to decline to do do something
  - Exception: Duplicate
- Be *very* wary of violating pass-by-value semantics
  - Mark things as not-mutable to get read-only shared access to memory

ALTREP Writing Specifics

- Don't write methods for the _EX variants

ALTREP Writing Specifics

- Don't write methods for the _EX variants
  - Default calls down to non _EX variant

ALTREP Writing Specifics

- Don't write methods for the _EX variants
  - Default calls down to non _EX variant
- `Duplicate` method **MUST** return a SEXP which can be modified by interaction with writeable dataptr

ALTREP Writing Specifics

- Don't write methods for the _EX variants
  - Default calls down to non _EX variant
- `Duplicate` method **MUST** return a SEXP which can be modified by interaction with writeable dataptr
  - or fail by throwing an error

ALTREP Writing Specifics

- Don't write methods for the _EX variants
  - Default calls down to non _EX variant
- `Duplicate` method **MUST** return a SEXP which can be modified by interaction with writeable dataptr
  - or fail by throwing an error
- Write functions/macros which abstract details of whats in data1/data2

ALTREP Writing Specifics

- Don't write methods for the _EX variants
  - Default calls down to non _EX variant
- `Duplicate` method **MUST** return a SEXP which can be modified by interaction with writeable dataptr
  - or fail by throwing an error
- Write functions/macros which abstract details of whats in data1/data2
  - Always use those even in your own methods

ALTREP Writing Specifics

- Don't write methods for the _EX variants
  - Default calls down to non _EX variant
- `Duplicate` method **MUST** return a SEXP which can be modified by interaction with writeable dataptr
  - or fail by throwing an error
- Write functions/macros which abstract details of whats in data1/data2
  - Always use those even in your own methods
- Do not write C code which calls `R_altrep_data*` or **especially** `R_set_altrep_data*` outside of ALTREP methods

R Internal Data Access API

Accessing the Data

Accessing Full Data (Integer Vector)

- `INTEGER` - returns `int *` to full data in array form

(*) indicates additions for ALTREP support

Accessing Full Data (Integer Vector)

- `INTEGER` - returns `int *` to full data in array form
  - *must always succeed or throw e.g. memory error* regardless of ALTREPness

(*) indicates additions for ALTREP support

Accessing Full Data (Integer Vector)

- `INTEGER` - returns `int *` to full data in array form
  - *must always succeed or throw e.g. memory error* regardless of ALTREPness
- (*) `INTEGER0` - efficiently return pointer for non-ALTREPs

(*) indicates additions for ALTREP support

Accessing Full Data (Integer Vector)

- `INTEGER` - returns `int *` to full data in array form
  - *must always succeed or throw e.g. memory error* regardless of ALTREPness
- (*) `INTEGER0` - efficiently return pointer for non-ALTREPs
- (*) `INTEGER_RO` - returns `const` pointer

(*) indicates additions for ALTREP support

Accessing Full Data (Integer Vector)

- `INTEGER` - returns `int *` to full data in array form
  - *must always succeed or throw e.g. memory error* regardless of ALTREPness
- (*) `INTEGER0` - efficiently return pointer for non-ALTREPs
- (*) `INTEGER_RO` - returns `const` pointer
- (*) `INTEGER_OR_NULL` - returns `NULL` pointer if ALTREP "prefers not to" populate full data array

(*) indicates additions for ALTREP support

AFTER INTEGER

- modifications in the addressed memory must be reflected in R object

AFTER INTEGER

- modifications in the addressed memory must be reflected in R object
  - This can't be detected

AFTER INTEGER

- modifications in the addressed memory must be reflected in R object
  - This can't be detected
  - ALTREP representation/metdata is invalidated

AFTER INTEGER

- modifications in the addressed memory must be reflected in R object
  - This can't be detected
  - ALTREP representation/metdata is invalidated
  - Often data2 of ALTREP object stores standard vector SEXP once this happens

AFTER INTEGER

- modifications in the addressed memory must be reflected in R object
  - This can't be detected
  - ALTREP representation/metdata is invalidated
  - Often data2 of ALTREP object stores standard vector SEXP once this happens
    - Further calls to INTEGER, etc just hit that instead

AFTER INTEGER

- modifications in the addressed memory must be reflected in R object
  - This can't be detected
  - ALTREP representation/metdata is invalidated
  - Often data2 of ALTREP object stores standard vector SEXP once this happens
    - Further calls to INTEGER, etc just hit that instead
- `INTEGER_RO` and `INTEGER_OR_NULL` prevent this destructive access

AFTER INTEGER

- modifications in the addressed memory must be reflected in R object
  - This can't be detected
  - ALTREP representation/metdata is invalidated
  - Often data2 of ALTREP object stores standard vector SEXP once this happens
    - Further calls to INTEGER, etc just hit that instead
- `INTEGER_RO` and `INTEGER_OR_NULL` prevent this destructive access
  - Should be used in your C code where possible

Retrieving Partial Data

- (*) `INTEGER_ELT` - return c value (`int`, `double`) for single data element

Retrieving Partial Data

- (*) `INTEGER_ELT` - return c value (`int`, `double`) for single data element
- (*) `INTEGER_GET_REGION` - populate provided buffer with values from contiguous region

Retrieving Partial Data

- (*) `INTEGER_ELT` - return c value (`int`, `double`) for single data element
- (*) `INTEGER_GET_REGION` - populate provided buffer with values from contiguous region
  - Copies data so not ALTREP destructive

How Not To Talk To ALTREPs

- `INTEGER` (often) destroys aspects of ALTREPness

How Not To Talk To ALTREPs

- `INTEGER` (often) destroys aspects of ALTREPness
- `INTEGER_ELT` in tight loop painfully slow

ALTREP-Safe Full Data Access (`include/R_ext/Itermacros.h`)

- `ITERATE_BY_REGION`

ALTREP-Safe Full Data Access (`include/R_ext/Itermacros.h`)

- `ITERATE_BY_REGION`
  - Grabs full dataptr if possible via `*_OR_NULL`

ALTREP-Safe Full Data Access (`include/R_ext/Itermacros.h`)

- `ITERATE_BY_REGION`
  - Grabs full dataptr if possible via `*_OR_NULL`
  - Wraps repeated `*_GET_REGION` calls

ALTREP-Safe Full Data Access (`include/R_ext/Itermacros.h`)

- `ITERATE_BY_REGION`
  - Grabs full dataptr if possible via `*_OR_NULL`
  - Wraps repeated `*_GET_REGION` calls
  - ALTREP safe

ALTREP-Safe Full Data Access (`include/R_ext/Itermacros.h`)

- `ITERATE_BY_REGION`
  - Grabs full dataptr if possible via `*_OR_NULL`
  - Wraps repeated `*_GET_REGION` calls
  - ALTREP safe
  - Allows for efficient tight loop over region pointer

ALTREP-Safe Full Data Access (`include/R_ext/Itermacros.h`)

- `ITERATE_BY_REGION`
  - Grabs full dataptr if possible via `*_OR_NULL`
  - Wraps repeated `*_GET_REGION` calls
  - ALTREP safe
  - Allows for efficient tight loop over region pointer
- `ITERATE_BY_REGION0`

ALTREP-Safe Full Data Access (`include/R_ext/Itermacros.h`)

- `ITERATE_BY_REGION`
  - Grabs full dataptr if possible via `*_OR_NULL`
  - Wraps repeated `*_GET_REGION` calls
  - ALTREP safe
  - Allows for efficient tight loop over region pointer
- `ITERATE_BY_REGION0`
  - **Always** uses repeated `*_GET_REGION` chunks

ALTREP-Safe Full Data Access (`include/R_ext/Itermacros.h`)

- `ITERATE_BY_REGION`
  - Grabs full dataptr if possible via `*_OR_NULL`
  - Wraps repeated `*_GET_REGION` calls
  - ALTREP safe
  - Allows for efficient tight loop over region pointer
- `ITERATE_BY_REGION0`
  - **Always** uses repeated `*_GET_REGION` chunks
- `ITERATE_BY_REGION_PARTIAL(|0)`

ALTREP-Safe Full Data Access (`include/R_ext/Itermacros.h`)

- `ITERATE_BY_REGION`
  - Grabs full dataptr if possible via `*_OR_NULL`
  - Wraps repeated `*_GET_REGION` calls
  - ALTREP safe
  - Allows for efficient tight loop over region pointer
- `ITERATE_BY_REGION0`
  - **Always** uses repeated `*_GET_REGION` chunks
- `ITERATE_BY_REGION_PARTIAL(|0)`
  - Same as above but specify starting position and count

An Example - `which` Internals

(Part of) the C code implementing the `which` R function:

```
int ioffset = 1;
int *buf = (int *) R_alloc(len, sizeof(int));
/* use iteration macros to be ALTREP safe <snip> */
ITERATE_BY_REGION(v, ptr, idx, nb, int, LOGICAL, {
    for(int i = 0; i < nb; i++) {
    if(ptr[i] == TRUE) {
        buf[j] = ioffset + i; // offset has +1 built i
        j++;
    }

    }
    ioffset += nb; // move to beginning of next buffer
});

len = j;
// buf has ints in it and we're returning ints, <snip>
PROTECT(ans = allocVector(INTSXP, len));
```

Example ALTREP packages

https://github.com/ALTREP-examples

Acknowledgements

- Luke Tierney
- Michael Lawrence
- Tomas Kalibera
- Mike Smith and Bioc Devel Forum
- You

Full List of ALTREP Methods

ALTREP Class Methods (All ALTREP Types)

- UnserializeEX
- Unserialize
- Serialized_state
- DuplicateEX
- Duplicate
- Coerce
- Inspect
- **Length**

ALTVEC Class Methods (Vectors)

ALTREP methods, plus

- **`Dataptr`**
- `Dataptr_or_null`
- `Extract_subset`

ALTINTEGER, ALTREAL Class Methods

ALTVEC methods, plus

- `Elt`
- `Get_region`
- `Is_sorted`
- `No_NA`
- `Sum`
- `Min`
- `Max`

ALTLOGICAL Class Methods

ALTVEC methods, plus

- `Elt`
- `Get_region`
- `Is_sorted`
- `No_NA`
- `Sum`

ALTRAW/ALTCOMPLEX Class Methods

ALTVEC methods, plus

- `Elt`
- `Get_region`

ALTSTRING

ALTVEC methods, plus

- `Elt`
- `Set_elt`
- `Is_sorted`
- `No_NA`