Garbage Collection in Ruby Extensions

Tobias Peters

Universität Oldenburg

Content

- Extensions

Content

- Extensions
- Garbage Collection in Ruby

Content

- Extensions
- Garbage Collection in Ruby
- Strategies for Garbage Collection in extensions

Content

- Extensions
- Garbage Collection in Ruby
- Strategies for Garbage Collection in extensions
- Using Swig

Content

- Extensions
- Garbage Collection in Ruby
- Strategies for Garbage Collection in extensions
- Using Swig
- Common misconceptions?

Ruby Extensions

make code written in C(++) accessible from Ruby code

Ruby Extensions

Reasons for using extensions:

- Execution speed
- Large existing code base
- Mandatory implementation language

Ruby Extensions

Reasons for using extensions:

- Execution speed
- Large existing code base
- Mandatory implementation language

Considerations:

- Security risk
- Stability risk
- Resource management

Garbage Collection

- frees the programmer from tracking object lifetimes
- inaccessible objects get deleted

```
x = Object.new
# possibly do something with x
x = nil
# the new object is inaccessible here
```

Object accessibility

"root set": Objects referenced by

- global variables
- local variables currently on the stack

Object accessibility

"root set": Objects referenced by

- global variables
- local variables currently on the stack

"reachable set":

- Root set
- All objects referenced another obj. in this set

Object accessibility

"root set": Objects referenced by

- global variables
- local variables currently on the stack

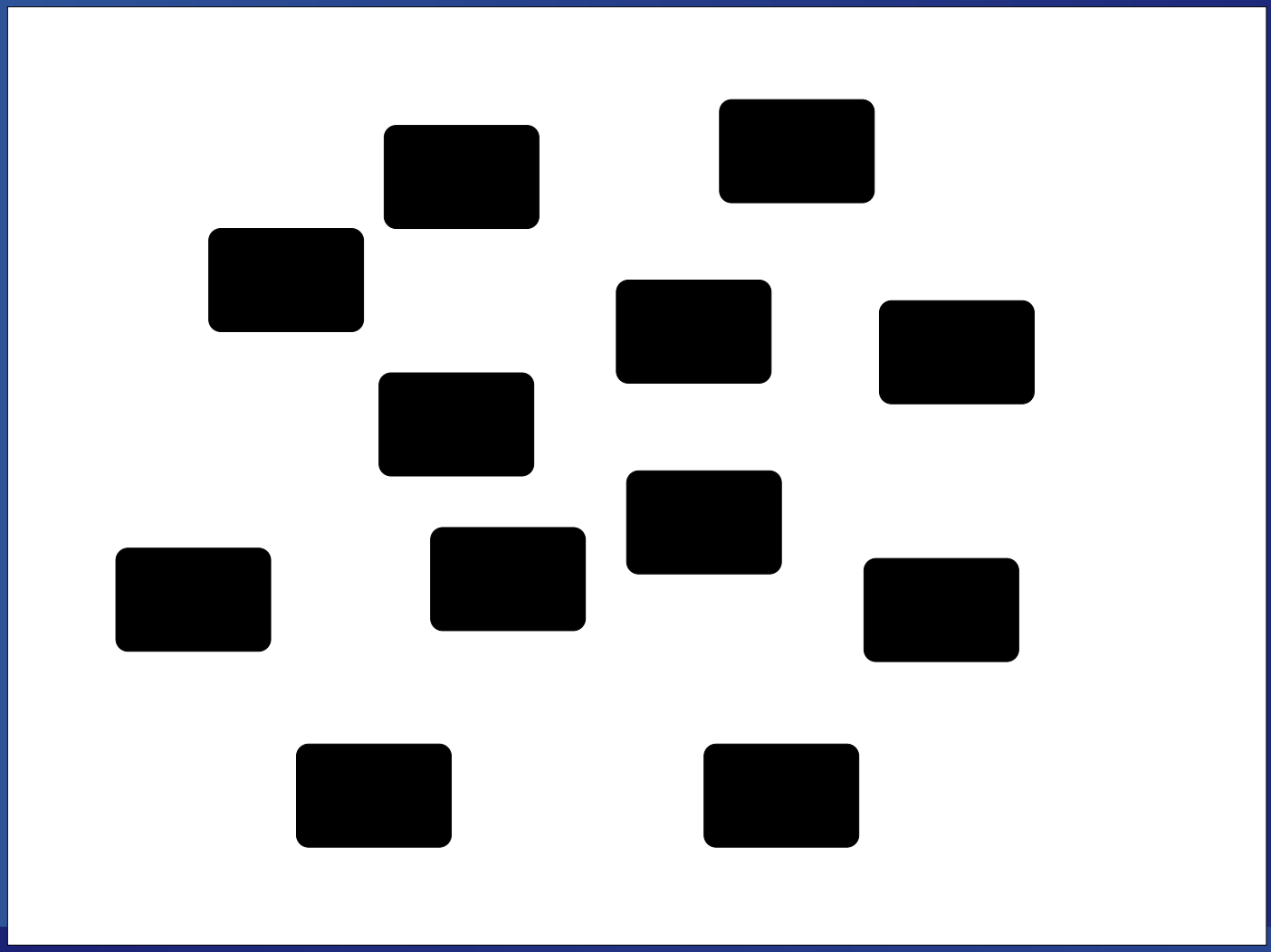
"reachable set":

- Root set
- All objects referenced another obj. in this set

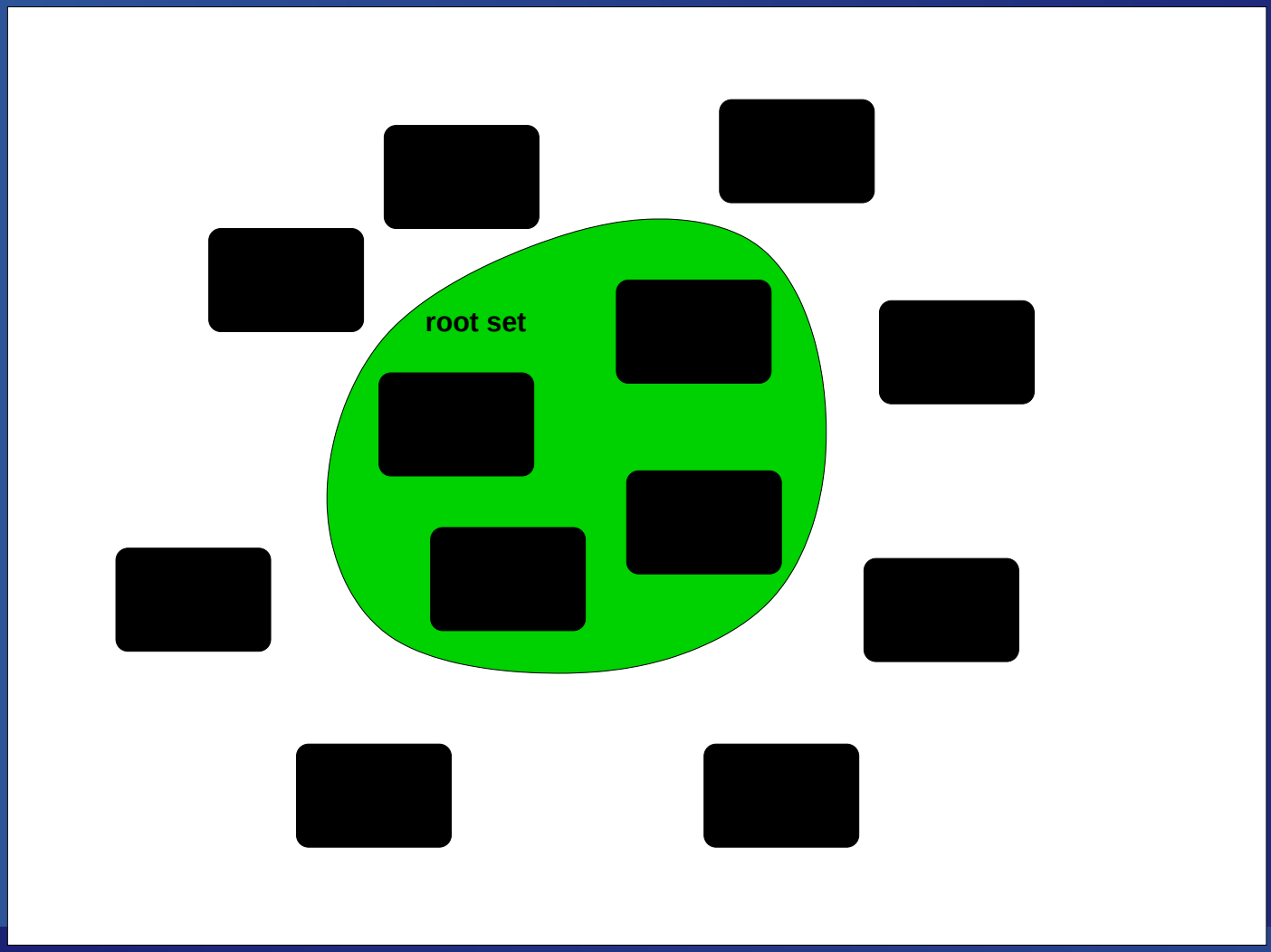
Reference paths: An object references

- its class
- its data members (via instance variables)
- specific references

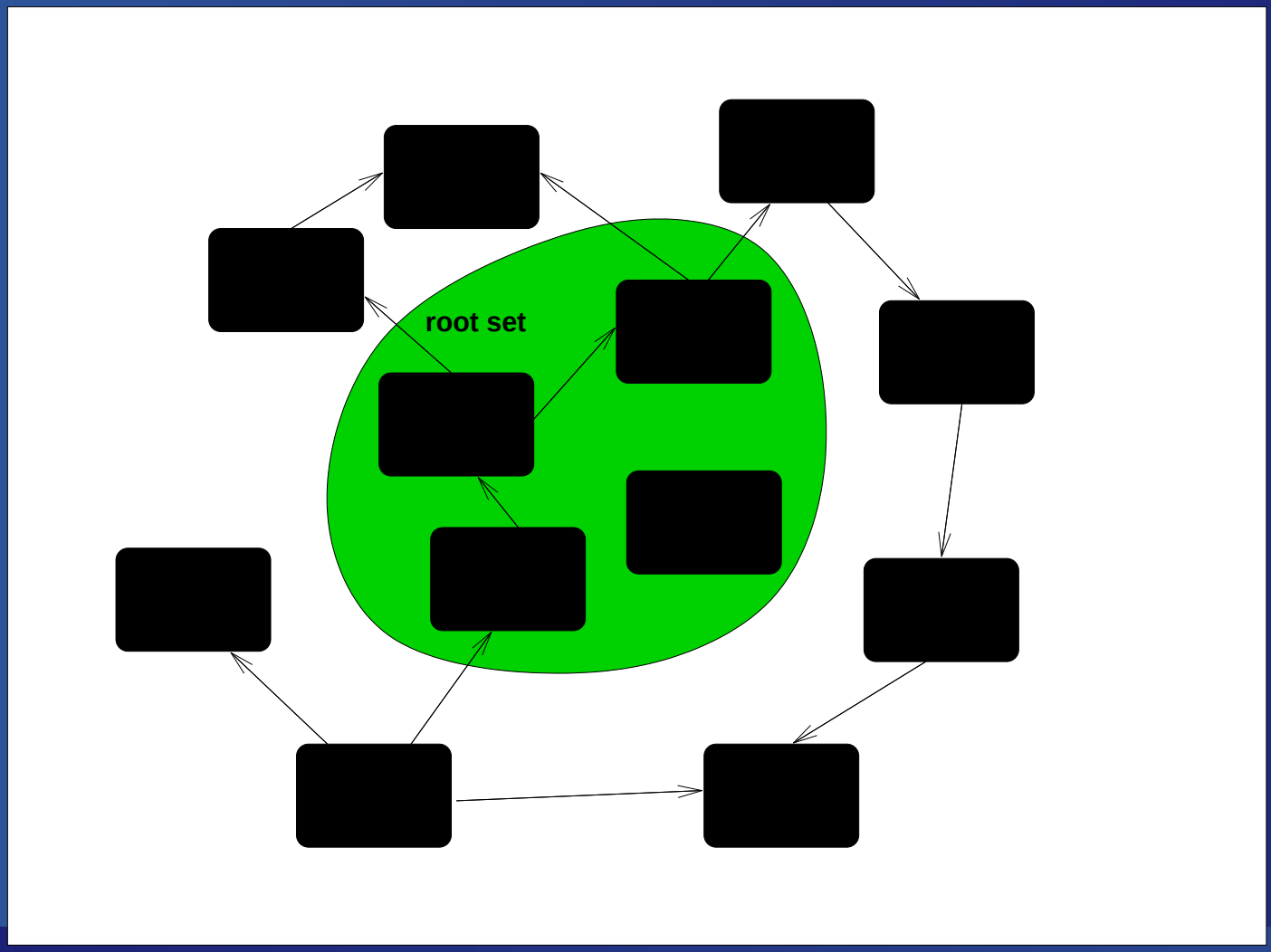
Object accessibility



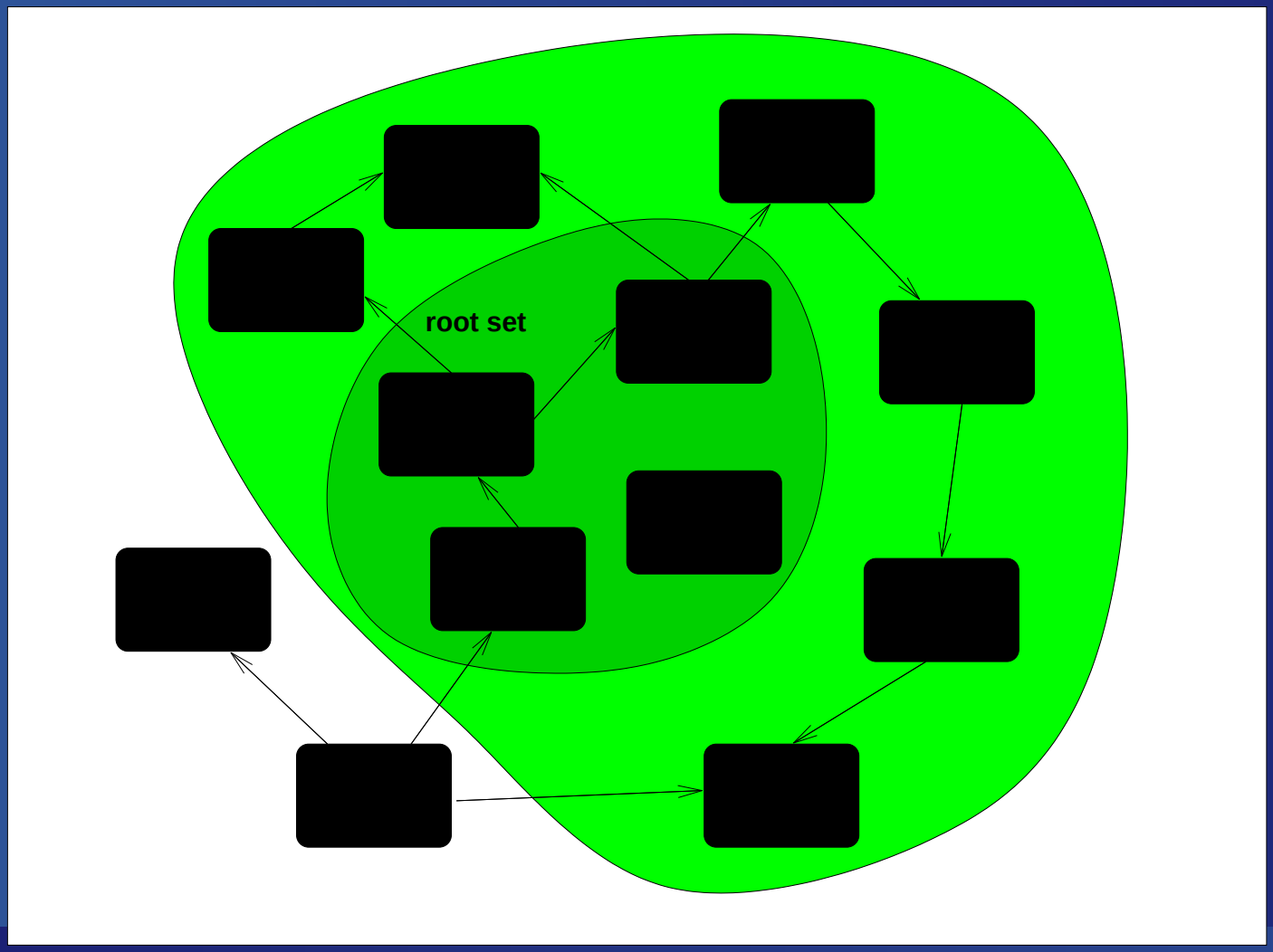
Object accessibility



Object accessibility



Object accessibility



Objects and references implementation

Regular object:

- `struct {long flags;...}`

Objects and references implementation

Regular object:

- `struct {long flags;...}`

Reference:

- A pointer to a regular object, stored in a VALUE

Objects and references implementation

Regular object:

- `struct {long flags;...}`

Reference:

- A pointer to a regular object, stored in a VALUE

Direct Object:

- special VALUE

Mark and Sweep Garbage Collector

Mark phase

- Ruby iterates over all references defining the root set and calls `rb_gc_mark` on these references
- Objects receive marks and recursively call `rb_gc_mark` on all object references they know of

Mark and Sweep Garbage Collector

Mark phase

- Ruby iterates over all references defining the root set and calls `rb_gc_mark` on these references
- Objects receive marks and recursively call `rb_gc_mark` on all object references they know of

Sweep phase

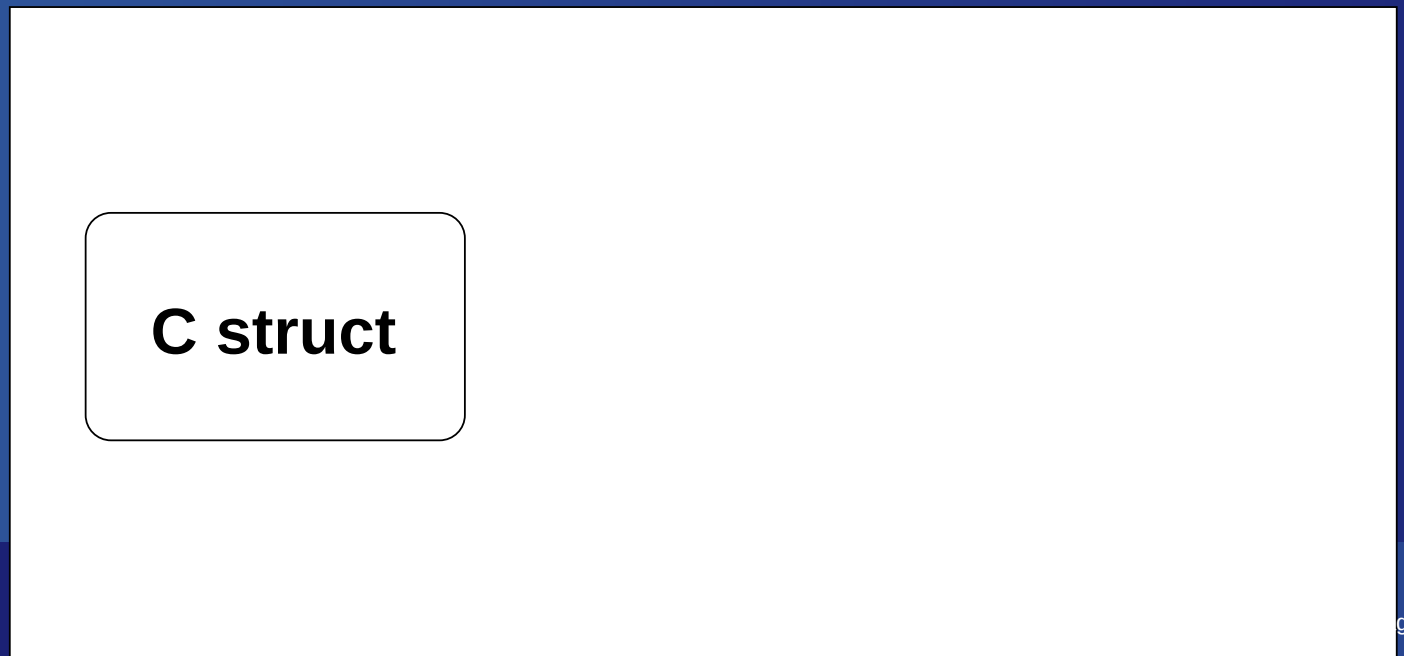
- Ruby iterates over all objects
- Objects that have not received a mark are deleted

Strategies for GC in Extensions

1. Do nothing
2. At least release the memory
3. Consider object relations
4. Revert to explicit resource management

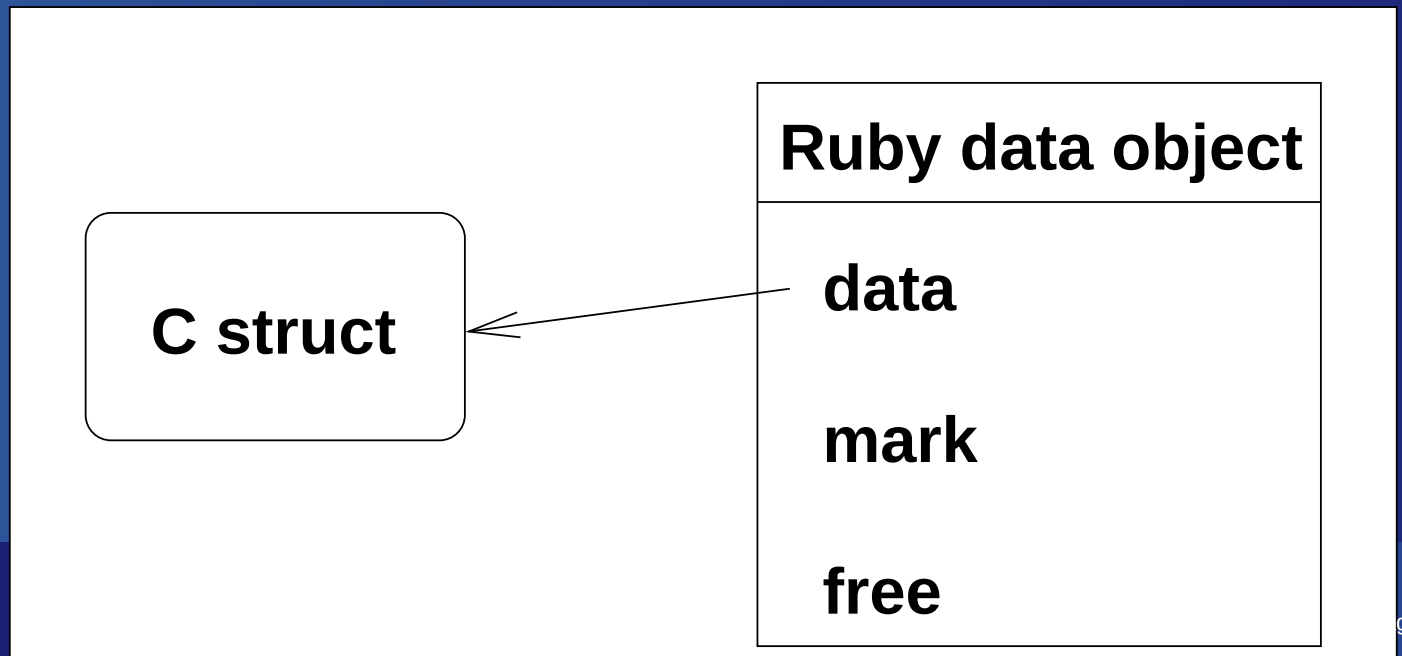
Strategies for GC in Extensions

1. Do nothing
2. At least release the memory
3. Consider object relations
4. Revert to explicit resource management



Strategies for GC in Extensions

1. Do nothing
2. At least release the memory
3. Consider object relations
4. Revert to explicit resource management



GC-strategy 1: Do nothing

- Register `NULL` as the object's "mark" and "free" functions

GC-strategy 1: Do nothing

- Register `NULL` as the object's "mark" and "free" functions
- The right strategy if you don't care for memory leaks

GC-strategy 1: Do nothing

- Register `NULL` as the object's "mark" and "free" functions
- The right strategy if you don't care for memory leaks
- Applicable for small programs

GC-strategy 1: Do nothing

- Register `NULL` as the object's "mark" and "free" functions
- The right strategy if you don't care for memory leaks
- Applicable for small programs
- Not applicable for libraries

GC-strategy 2: Release memory

- Register `NULL` as the object's "mark" function and `free` as the object's "free" function

GC-strategy 2: Release memory

- Register `NULL` as the object's "mark" function and `free` as the object's "free" function
- But watch out for
  - multiple ruby objects wrapping the same C object
  - inter object relations

GC-strategy 2: Release memory

multiple objects wrapping the same C object

- use reference counting for C object if available

GC-strategy 2: Release memory

multiple objects wrapping the same C object

- use reference counting for C object if available
- or use some "user data" field in the C struct to point back to ruby object if available

GC-strategy 2: Release memory

multiple objects wrapping the same C object

- use reference counting for C object if available
- or use some "user data" field in the C struct to point back to ruby object if available
- or have a hash-table that maps C pointers to ruby objects

GC-strategy 3: Object relations

```
x = Ext_Class_1.new(...)
x.learn_about(Ext_Class_2.new(...))
```

GC-strategy 3: Object relations

```
x = Ext_Class_1.new(...)
x.learn_about(Ext_Class_2.new(...))
```

- use reference counting if available
- no need to provide a mark function then

GC-strategy 3: Object relations

```
x = Ext_Class_1.new(...)
x.learn_about(Ext_Class_2.new(...))
```

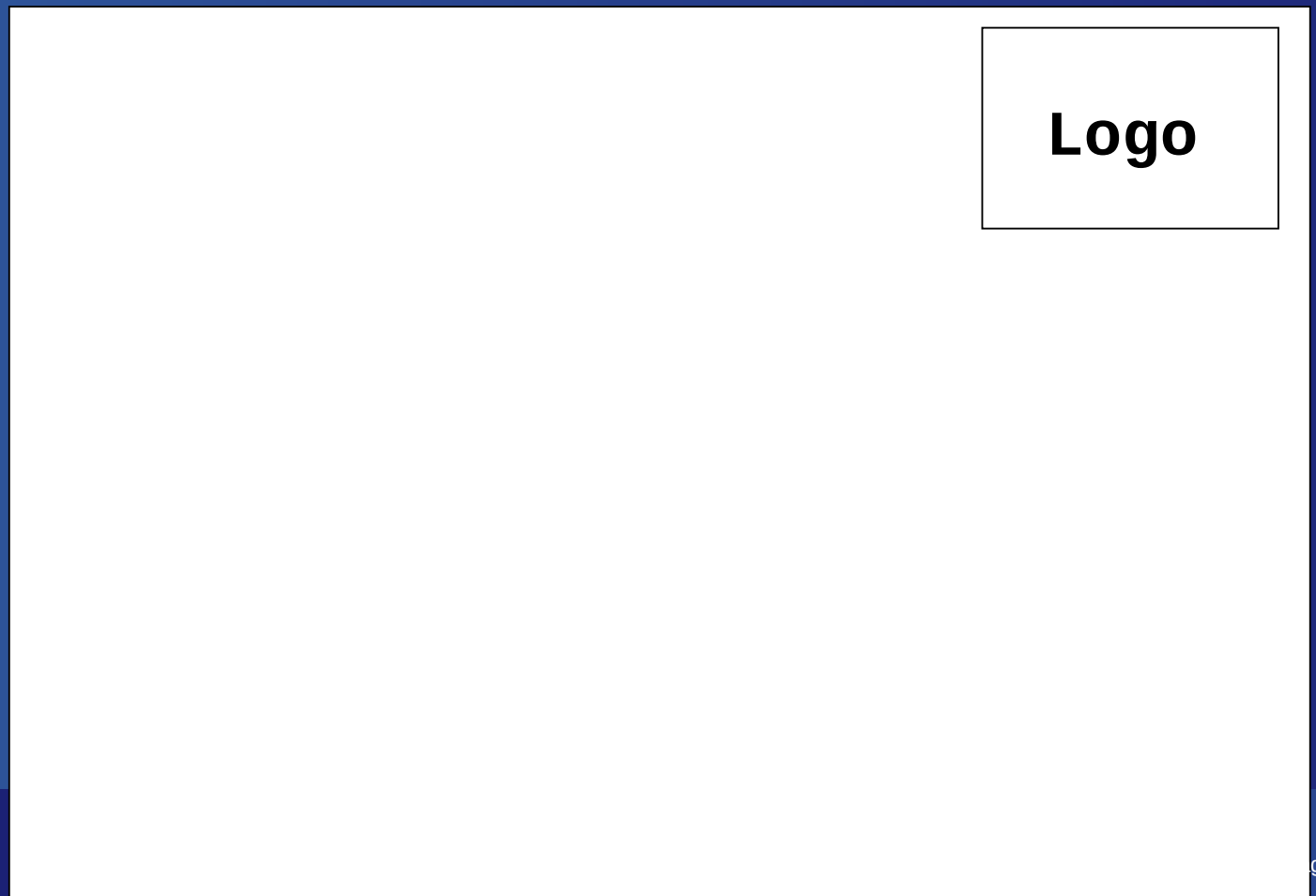
- otherwise unique mapping from C pointers to ruby wrapper objects is necessary
- class-specific mark functions have to be provided

GC-strategy 4: Explicit management

Sometimes garbage collection alone cannot determine the C object's lifetime.

GC-strategy 4: Explicit management

Sometimes garbage collection alone cannot determine the C object's lifetime.



Using SWIG

1. Do nothing
2. Use `%newobject`
3. Use `%typemaps`, `%markfunc` and `%freefunc`
   - Tip: Call the same `ruby_wrapper_for_klass(klass, bool create)` from within the `%typemaps` and the `%markfunctions`

Common Misconceptions

- Not trusting the Garbage Collector, desire to register objects as globals.

Common Misconceptions

- Not trusting the Garbage Collector, desire to register objects as globals.
- Misunderstandings of the purpose of the mark and sweep phases and functions.

Summary

- Quick and dirty programs do not require any GC support from an extension
- Correct GC support in extensions requires either a reference counting framework inside the C library or a reverse mapping of C pointers to ruby objects

Thank you