Three Ruby usages

Kouhei Sutou

ClearCode Inc.

RubyKaigi 2014
2014/09/20

Silver sponsor

SILVER SPONSORS

ClearCode Inc.

ClearCode runs free software business and contributes to free software.

Goal

- ✓ You know three Ruby usages
  - ✓ High-level interface
  - ✓ Glue
  - ✓ Embed
- ✓ You can remember them later

Targets

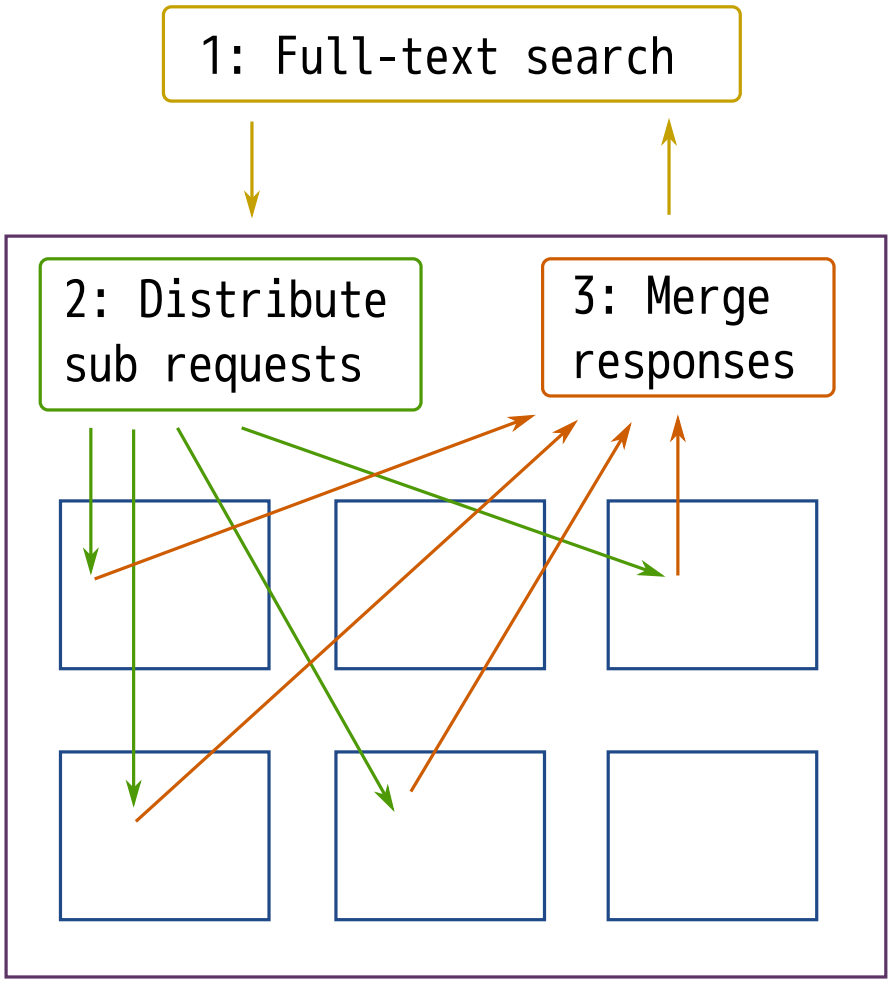
- ✓ High-level interface
  - ✓ Pure Rubyists
- ✓ Glue
  - ✓ Rubyists who can write C/C++
- ✓ Embed
  - ✓ Rubyists who also write C/C++

Case study

> *Implement distributed full-text search engine in Ruby*

Abbreviation: DFTSE = Distributed Full-Text Search Engine

DFTSE?



Distrubuted Full-Text Search Engine

Full-Text Search Engine

Why do we use DFTSE?

I'm developing Droonga

(A DFTSE implementation in Ruby)

High-level interface

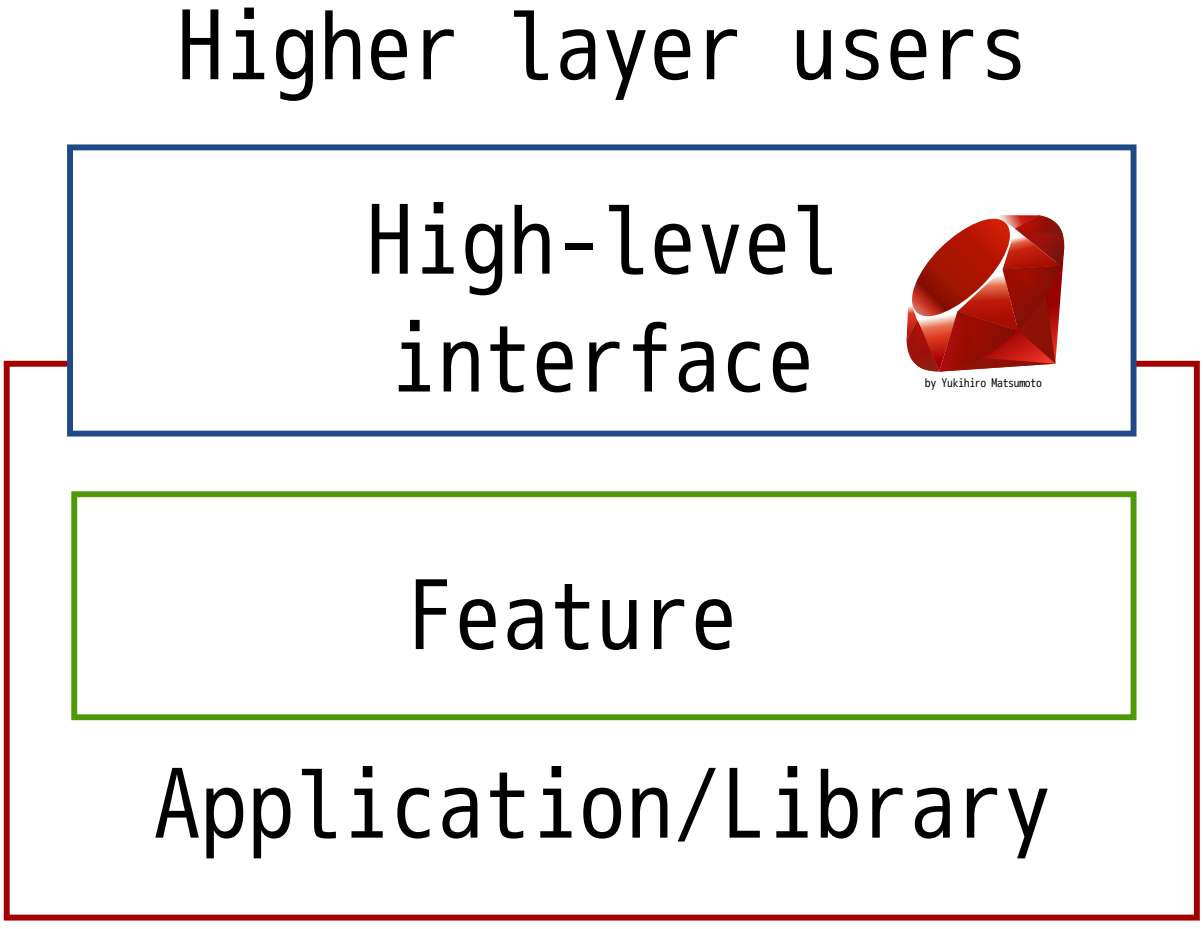
Three Ruby usages

- ✓ **High-level interface**
  - ✓ Target: Pure Rubyists
- ✓ Glue
- ✓ Embed

High-level interface

- ✓ Provides lower layer feature to higher layer
- ✓ With simpler/convenience API

High-level interface



Example

Vagrant

Developers

Vagrantfile

Build development environment

Vagrant

Active Record

Developers

Object based API

Access data in RDBMS

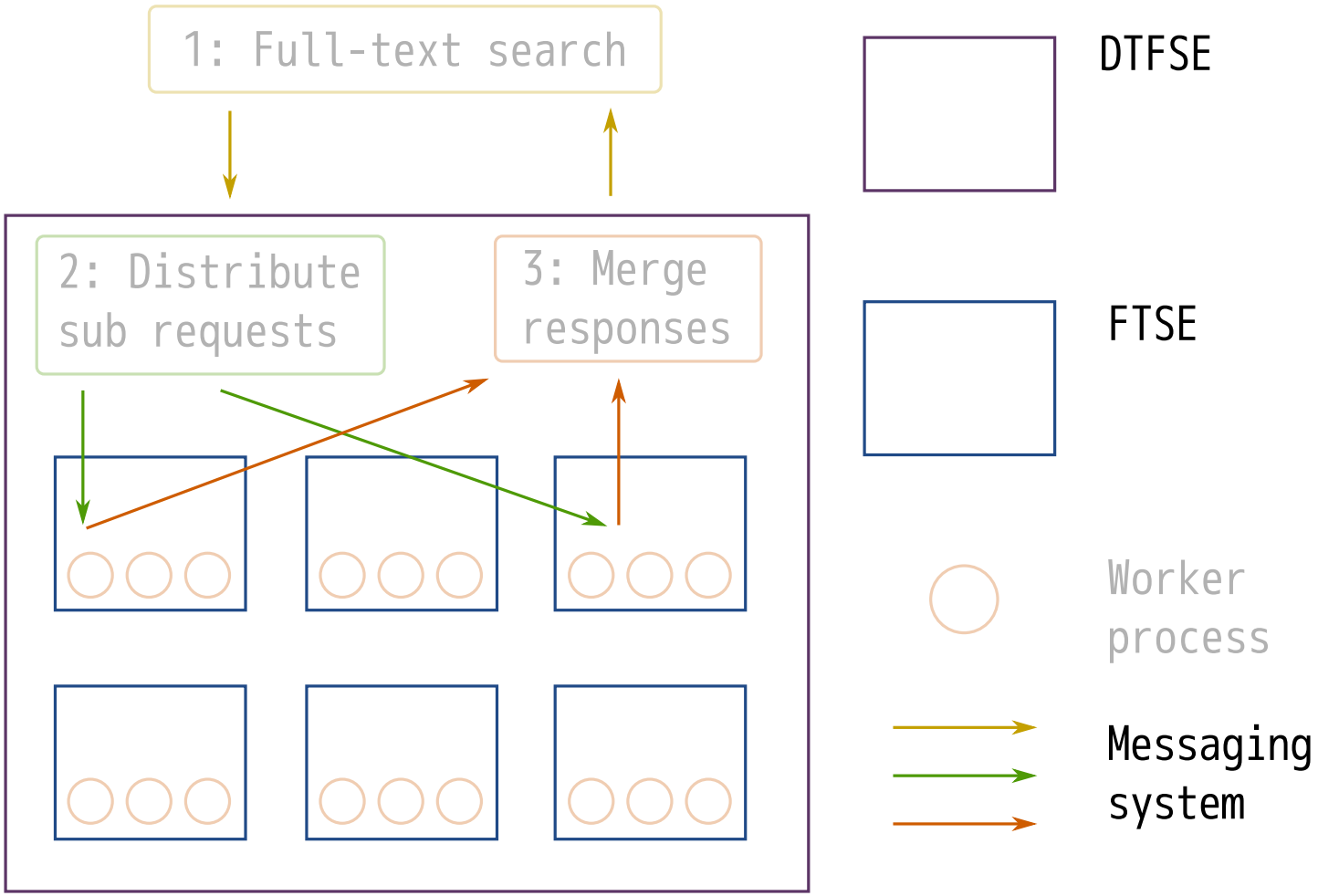
Active Record

Droonga: High-level IF

DFTSE components

- ✓ Full-text search engine
- ✓ **Messaging system**
- ✓ Cluster management
- ✓ Process management

Messaging system



Messaging system

- ✓ Provides distributed search feature
  - ✓ Plan how to search
  - ✓ Distribute requests
  - ✓ Merge responses
- ✓ Users don't know details

Characteristic

- ✓ Plan how to search
  - ✓ May speed up/down over 100 times
- ✓ Distribute requests
  - ✓ Network bound operation
- ✓ Merge responses
  - ✓ CPU and network bound operation

Point

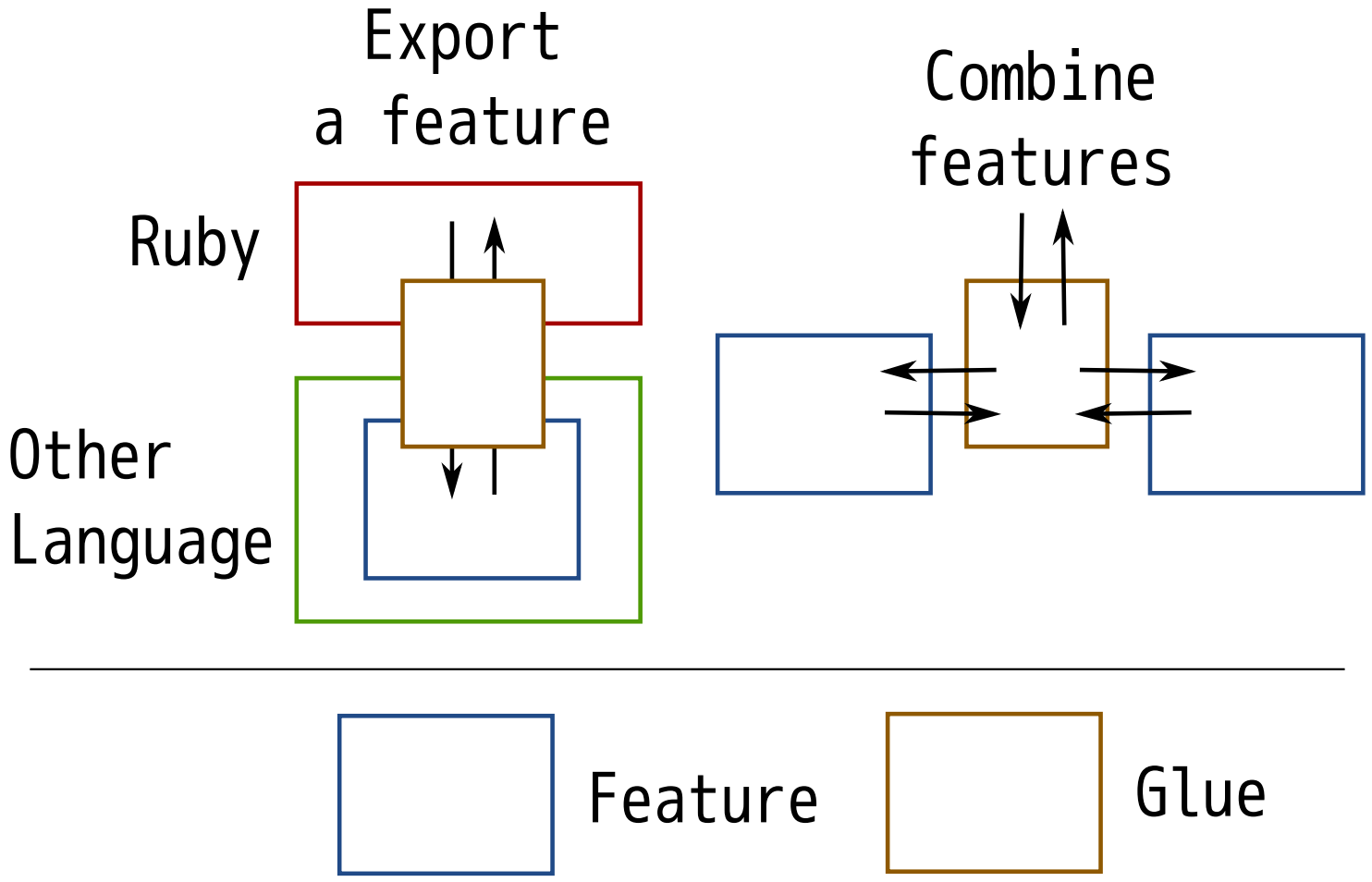
- Algorithm is important
  - Need to find new/existing better algorithm
  - "Rapid prototype and measure" feedback loop is helpful
  - Ruby is good at rapid dev.

Glue

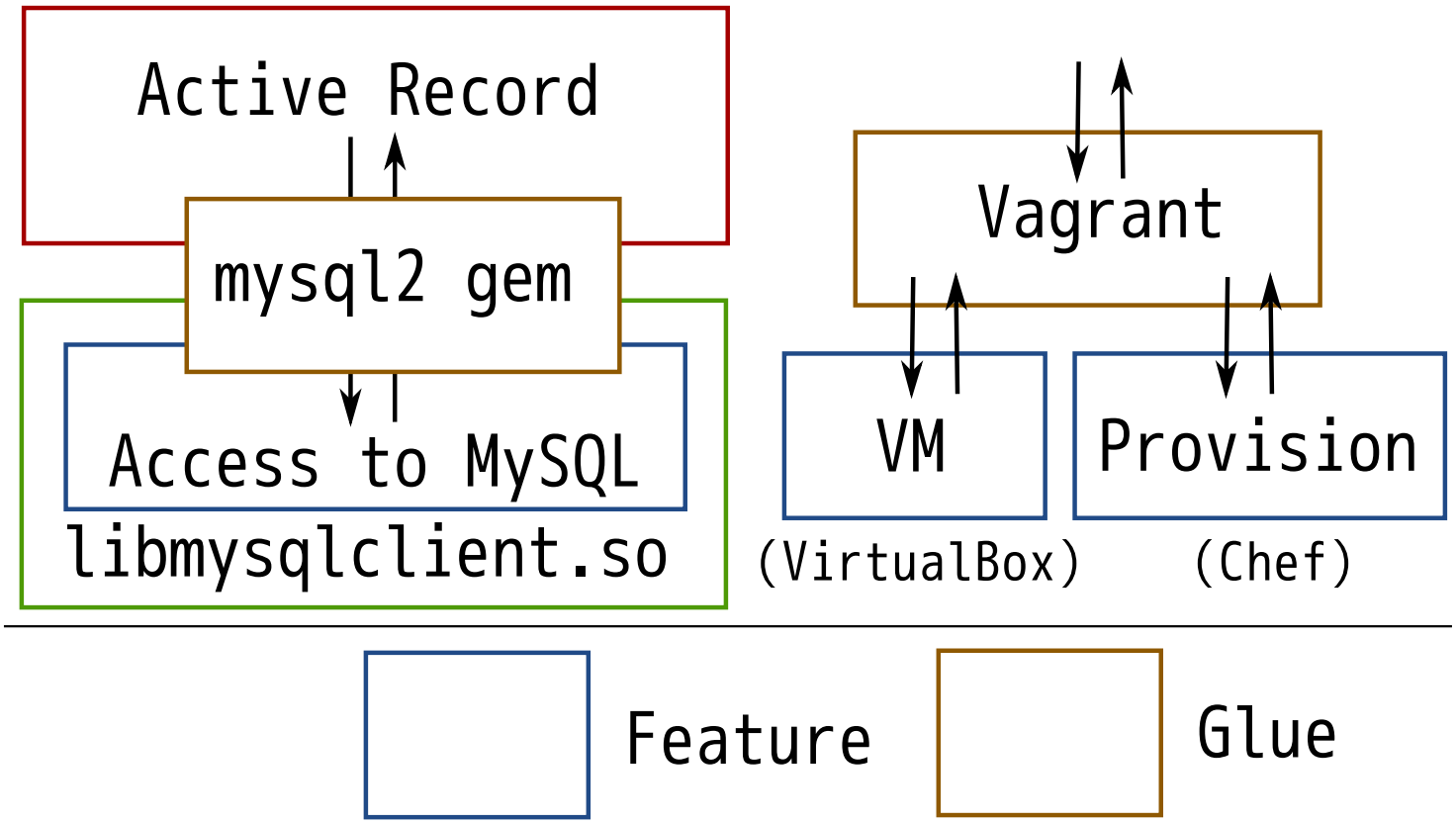
Three Ruby usages

- ✓ High-level interface
- ✓ **Glue**
  - ✓ Target:
    Rubyists who can write C/C++
- ✓ Embed

Glue


Export
a feature
Ruby
Other
Language
Combine
features
Feature
Glue

Example


Active Record
mysql2 gem
Access to MySQL
libmysqlclient.so
Vagrant
VM
Provision
(VirtualBox)
(Chef)
Feature
Glue

Why do we glue?

- ✓ Reuse existing features

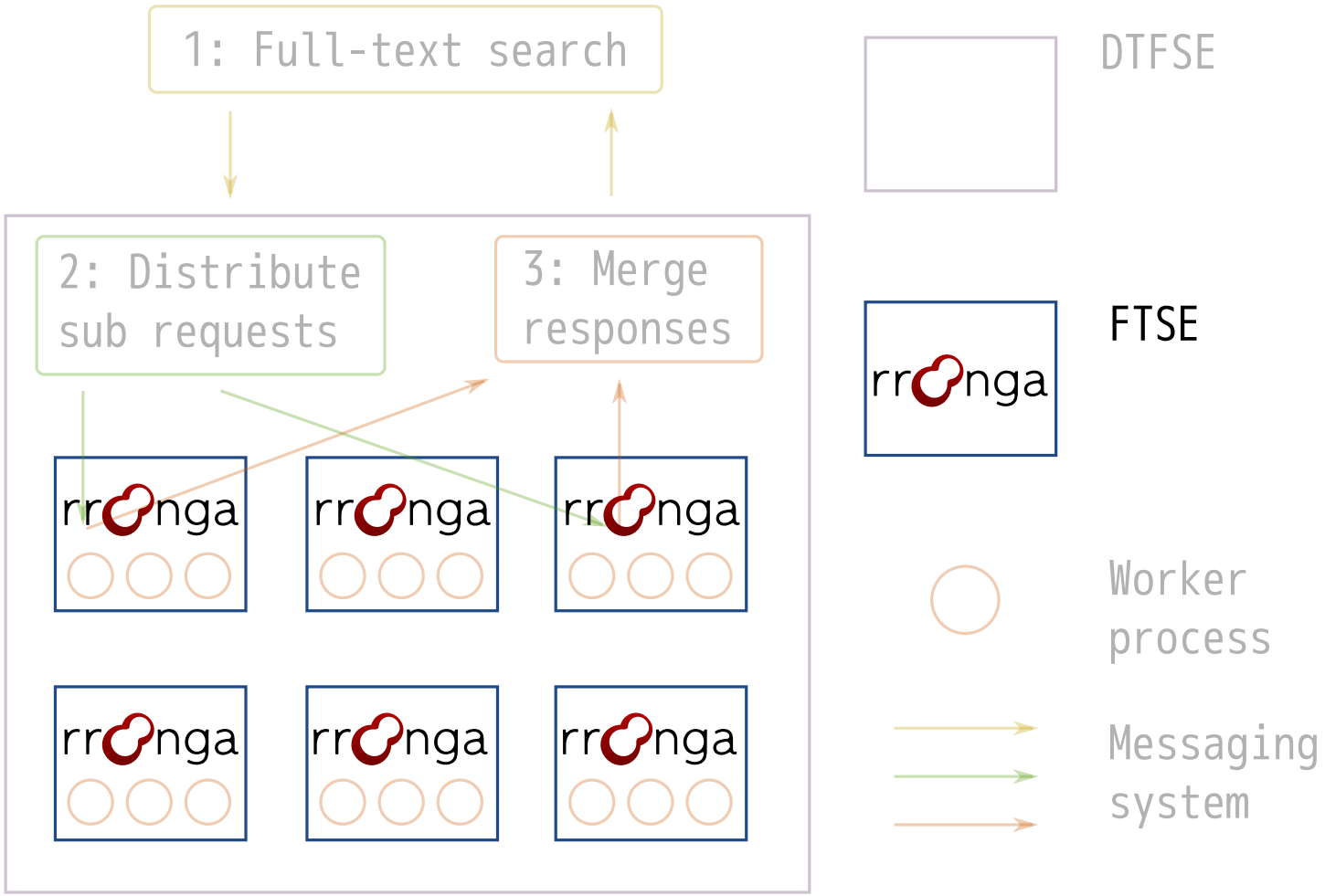
How to glue

- ✓ Use external library
  - ✓ Implement bindings (mysql2 gem)
- ✓ Use external command
  - ✓ Spawn command (Vagrant)
- ✓ Use external service
  - ✓ Implement client

Glue in Droonga

- ✓ **Rroonga**: Groonga bindings
  - ✓ Groonga: FTSE C library (and server)
- ✓ Cool.io: libev bindings
  - ✓ libev: Event loop C library
    (Based on I/O multiplexing and non-blocking I/O)
- ✓ Serf: Clustering tool (in Droonga)

Rroonga in Droonga



FTSE in Droonga

- ✓ Must be fast!
- ✓ CPU bound processing

For fast Rroonga

- ✓ **Do heavy processing in C**
  - ✓ Nice to have Ruby-ish API
- ✓ Less memory allocation
  - ✓ Cache internal buffer
- ✓ Multiprocessing
  - ✓ Groonga supports multiprocessing

Search

```
Groonga::Database.open(ARGV[0])
entries = Groonga["Entries"]

entries.select do |record|
  record.description =~ "Ruby"
end
```

Search - Pure Ruby (ref)

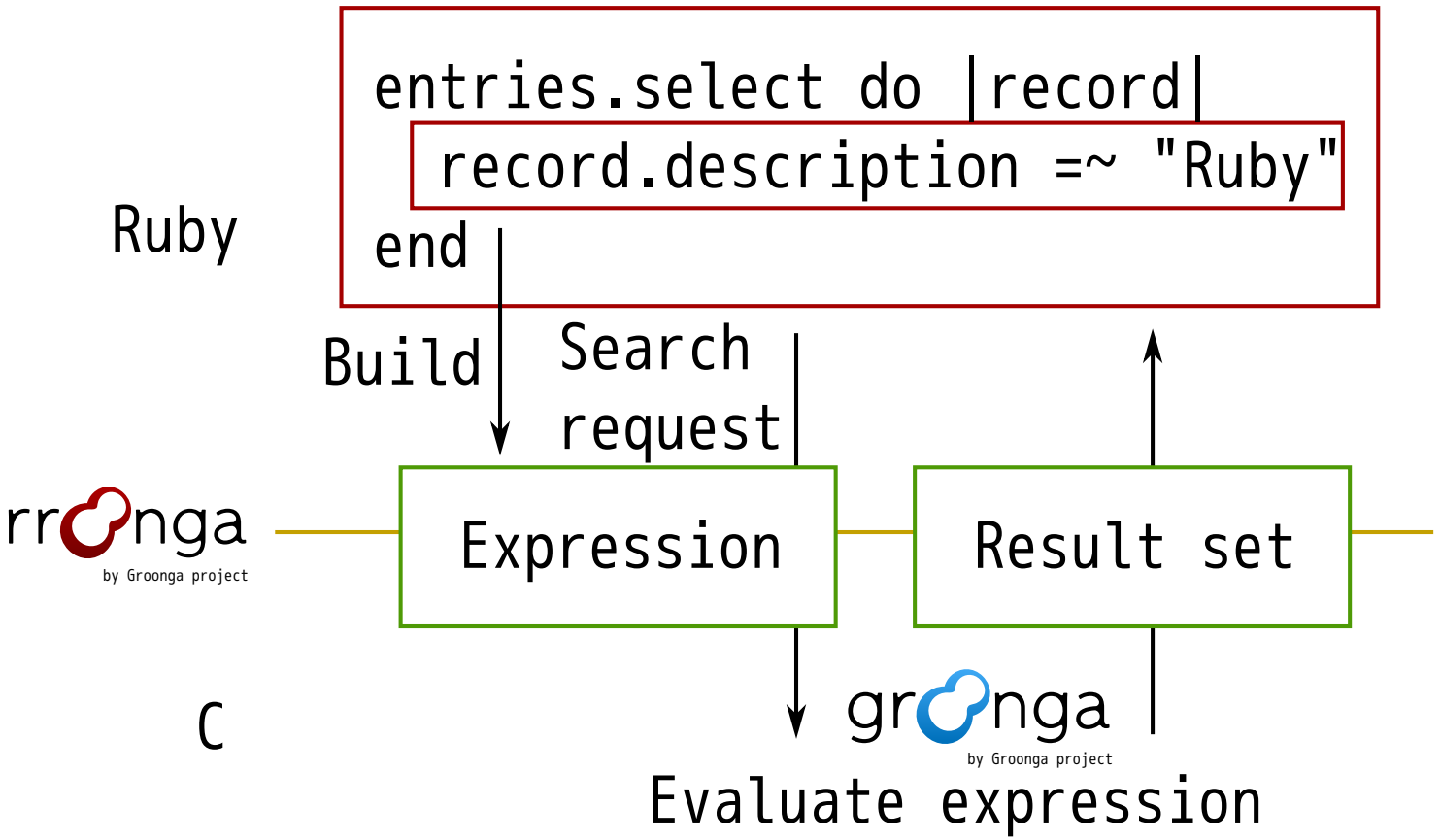
```ruby
Groonga::Database.open(ARGV[0])
entries = Groonga["Entries"]

entries.find_all do |record|
  # This block is evaluated for each record
  /Ruby/ =~ record.description
end
```

Search impl.

```ruby
# (2) Evaluate expression in C
entries.select do |record|
  # (1) Build expression in Ruby
  # This block is evaluated only once
  record.description =~ "Ruby"
end
```

Search impl. - Fig.



Search - Benchmark

- ✓ Ruby (It's already showed)
- ✓ C

Search - C

```
grn_obj *expr;
grn_obj *variable;
const gchar *filter = "description @ \"Ruby\"";
grn_obj *result;

GRN_EXPR_CREATE_FOR_QUERY(&ctx, table, expr, variable);
grn_expr_parse(&ctx, expr,
               filter, strlen(filter), NULL,
               GRN_OP_MATCH, GRN_OP_AND,
               GRN_EXPR_SYNTAX_SCRIPT);
result = grn_table_select(&ctx, table, expr, NULL, GRN_OP_OR);
grn_obj_unlink(&ctx, expr);
grn_obj_unlink(&ctx, result);
```

Search - Benchmark

Ruby impl. is fast enough 😀

Impl.	Elapsed time
C	0.6ms
Ruby	0.8ms

(Full-text search with "Ruby" against 72632 records)

Embed

Three Ruby usages

- ✓ High-level interface
- ✓ Glue
- ✓ **Embed**
  - ✓ Target:
    Rubyists who also write C/C++

Embed

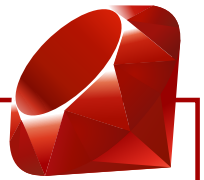
Internal engine

C/C++ application
C/C++ library

Implement
some features
in Ruby

Interface

Plugin API
Conifugration



C/C++ application
C/C++ library

Examples

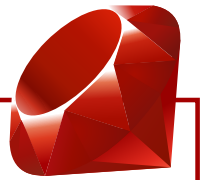
Internal engine

groonga

Implement query optimizer in Ruby

by Yukihiro Matsumoto

Interface

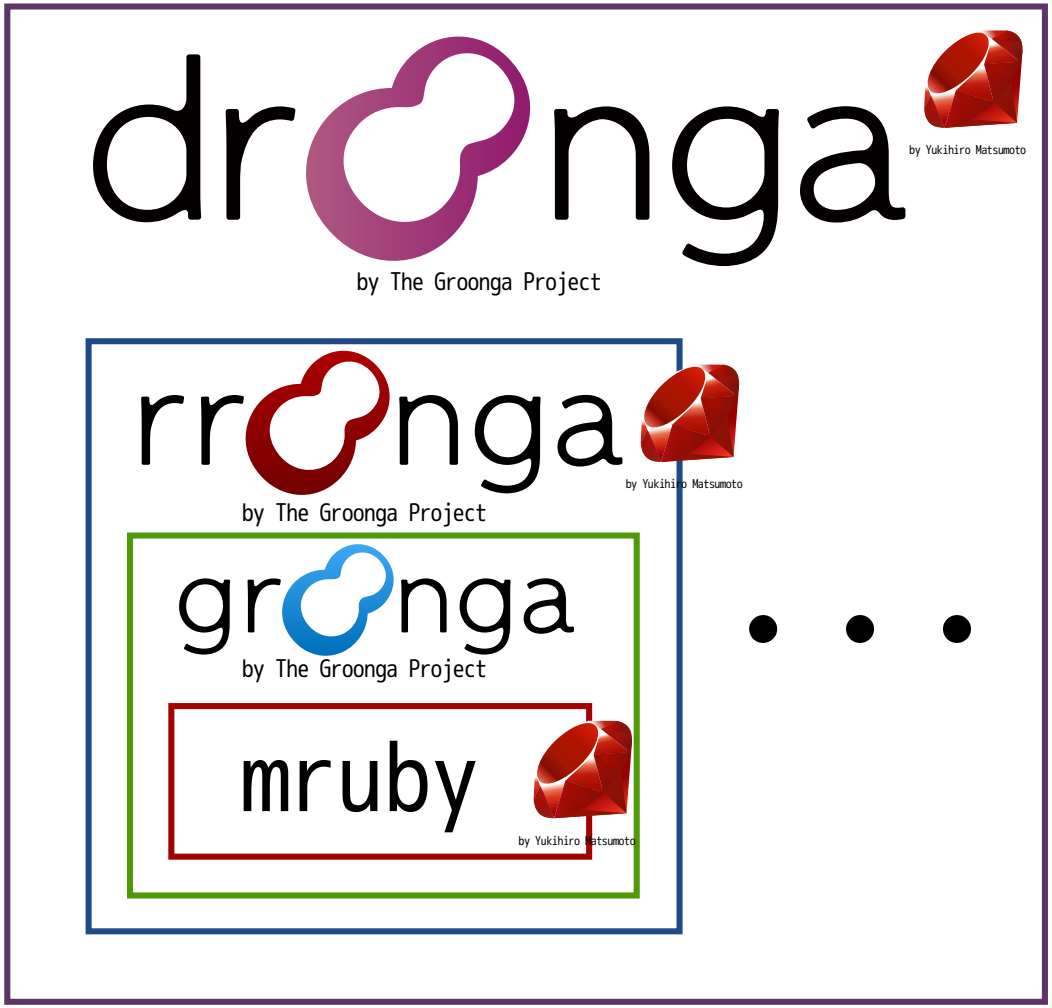


vim-ruby

by Yukihiro Matsumoto

VIM

Embed in Droonga



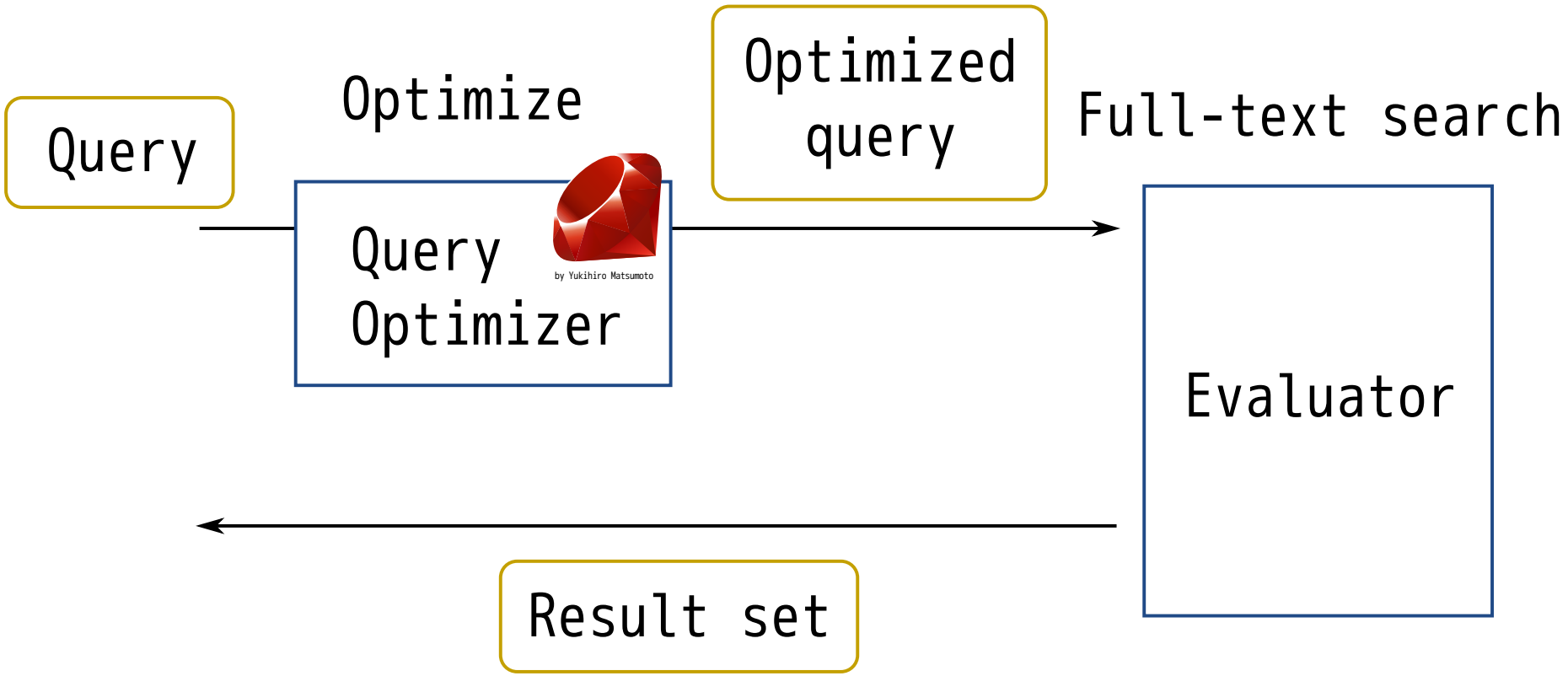
CRuby vs. mruby

- ✓ CRuby
  - ✓ Full featured!
  - ✓ Signal handler isn't needed ☹
- ✓ mruby
  - ✓ Multi-interpreters in a process!
  - ✓ You may miss some features ☹

mruby in Groonga

- ✓ Query optimizer
- ✓ Command interface (plan)
  - ✓ Interface and also high-level interface!
- ✓ Plugin API (plan)
  - ✓ Interface!

Query optimizer



Query optimizer

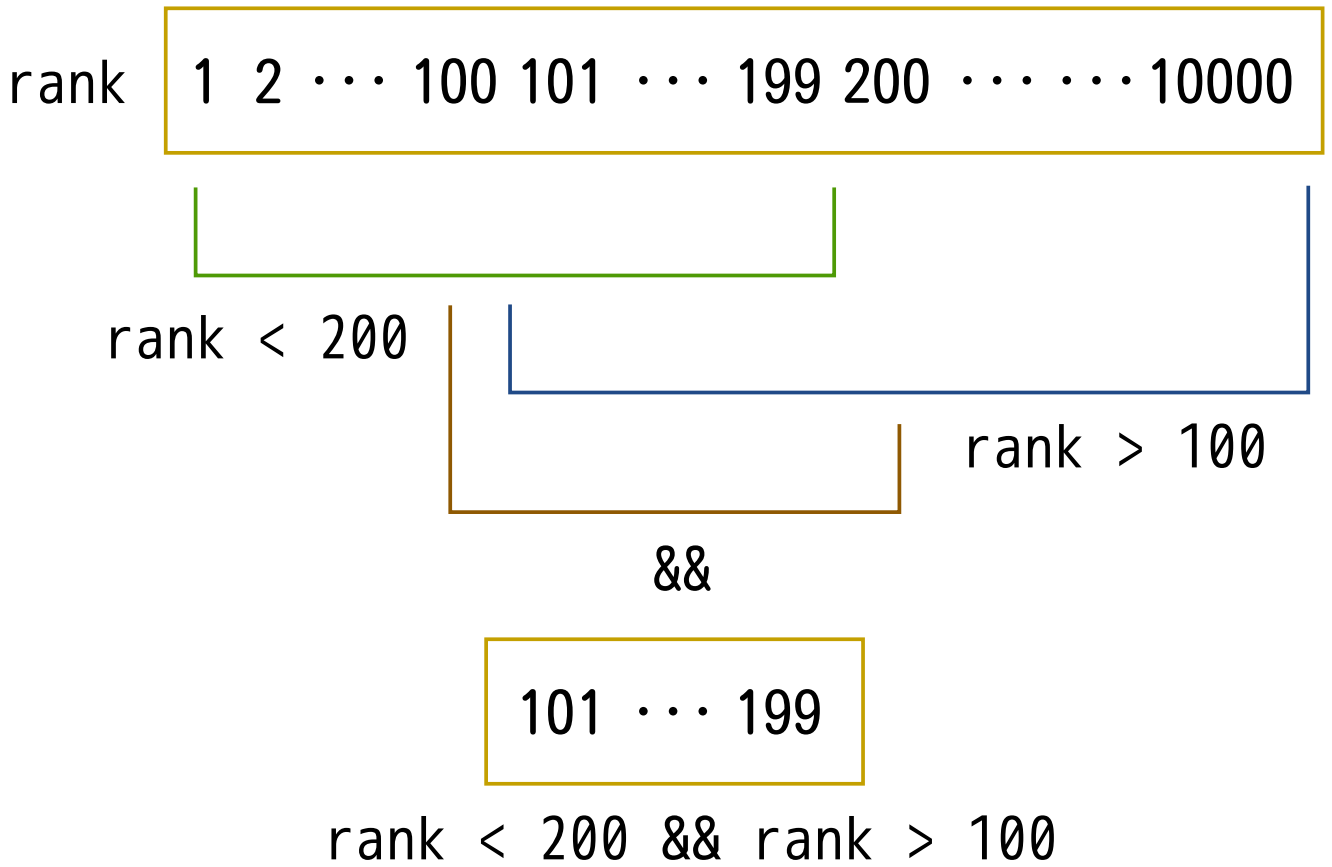
- ✓ Plan how to search
  - ✓ It's a bother ☹
- ✓ Light operation than FTS
- ✓ Depends on data

  (Choose effective index, use table scan and so on)

Example

```
rank < 200 && rank > 100
```

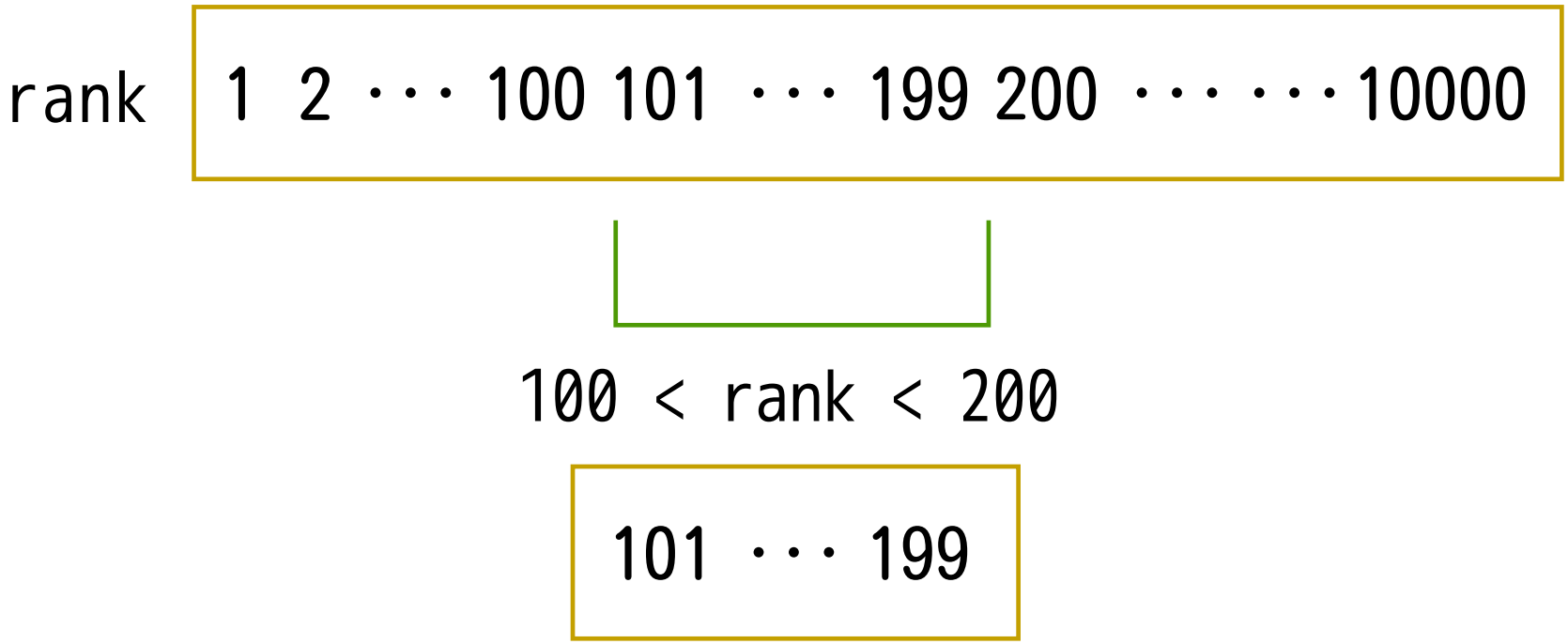
Simple impl.


rank
1 2 ··· 100 101 ··· 199 200 ··· ··· 10000
rank < 200
rank > 100
&&
101 ··· 199
rank < 200 && rank > 100

Simple impl.

- ✓ Slow against many out of range data

Optimized impl.



Is embedding reasonable?

Measure

Measure

- ✓ mruby overhead
- ✓ Speed-up by optimization

Overhead

Small overhead: Reasonable😀

# conds	mruby	Elapsed
1	○	0.24ms
1	×	0.16ms
4	○	0.45ms
4	×	0.19ms

Speed-up

Fast for many data:Reasonable😀

# records	mruby	no mruby
1000	0.29ms	0.31ms
10000	0.31ms	2.3ms
100000	0.26ms	21.1ms
1000000	0.26ms	210.2ms

Note

- Embedding needs many works
  - Write bindings, import mruby your build system and ...
- How to test your mruby part?
  - And how to debug?

Conclusion

Conclusion 1

- ✓ Describe three Ruby usages
  - ✓ High-level interface
  - ✓ Glue
  - ✓ Embed

Conclusion 2

- ✓ High-level interface
  - ✓ Target: Pure Rubyists
  - ✓ Provides lower layer feature to higher layer w/ usable interface
  - ✓ Ruby's flexibility is useful

Conclusion 3

- ✓ Glue
  - ✓ Target:
    Rubyists who can write C/C++
  - ✓ Why: Reuse existing feature
  - ✓ To be fast, do the process in C

Conclusion 4

- ✓ Embed
  - ✓ Target:
    Rubyists who also write C/C++
  - ✓ Why:
    Avoid bother programming by Ruby

Conclusion 5

- ✓ Embed
  - ✓ Is it reasonable for your case?
  - ✓ You need many works
  - ✓ Very powerful
    if your case is reasonable😀

Announcement

- ClearCode Inc.
  - A silver sponsor
  - Is recruiting
  - Will do readable code workshop
- The next Groonga conference
  - It's held at 11/29