Pure Ruby Apache Arrow reader/writer

Sutou Kouhei

ClearCode Inc.

RubyKaigi 2026

2026-04-24

Sutou Kouhei
A president Ruby committer

Silver Sponsors

ClearCode Inc.

https://www.clear-code.com/

Free software is important in ClearCode. We develop/support software with our free software development experiences.
We feed back our business experiences to free software.

20th anniversary!

今年で20周年！7月29日に東京でお祝いイベントをするよ！

Details: https://www.clear-code.com/anniversaries/20th.html

Apache Arrow

- ✓ Fast columnar format
  for large in-memory data
  メモリー上の大量データ用の高速なカラムナーフォーマット
- ✓ Fast data processing library
  for large in-memory data
  メモリー上の大量データ用の高速なデータ処理ライブラリー
- ✓ ...

Pure Ruby Apache Arrow reader/writer

- ✓ **Fast columnar format for large in-memory data**
  メモリー上の大量データ用の高速なカラムナーフォーマット
- ✓ Fast data processing library for large in-memory data
  メモリー上の大量データ用の高速なデータ処理ライブラリー
- ✓ ...

Wait! Do we already have it...?
あれ、もうなかった。。。？

- ✓ Red Arrow (The official bindings of the official C++ impl)
  （公式C++実装の公式バインディング）
  - ✓ Fast reader/writer, data processor, ...
    （高速な読み書き・データ処理など）
- ✓ Ruby Polars (The bindings of a Rust library)
  （Apache Arrow対応Rustライブラリーのバインディング）
  - ✓ Fast reader/writer, data processor, ...
    （高速な読み書き・データ処理など）
- ✓ ...

Motivation
なんで作ったの？

Spread Apache Arrow in Ruby ecosystem!

Rubyエコシステム内でApache Arrowをもっと広めたい！

Why pure Ruby implementation?
なぜpure Ruby実装なの？

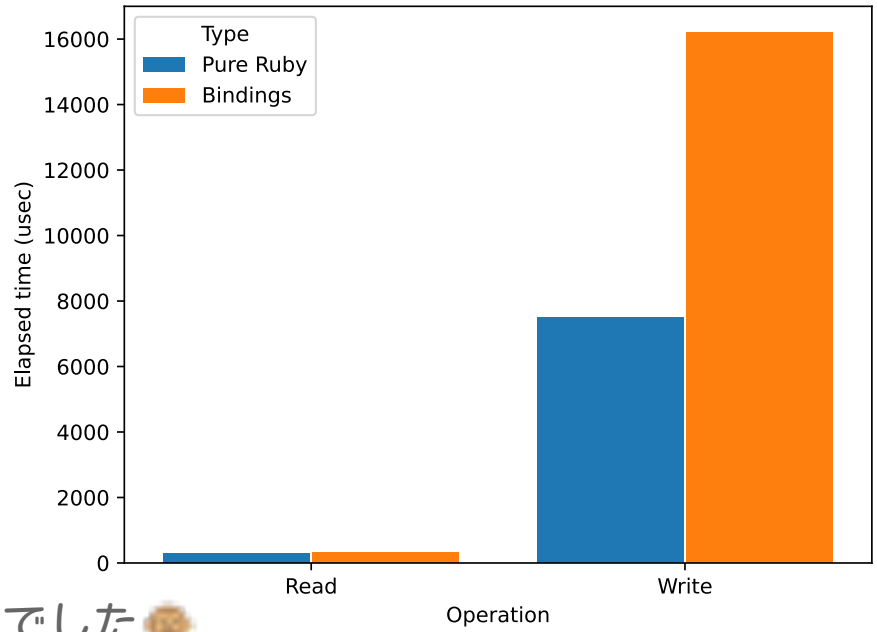
- ✓ Bindings are faster but:
  バインディングは速いけど
  - ✓ more difficult to install/maintain
    インストールやメンテナンスが難しくなりがち
  - ✓ larger install size
    インストールサイズが大きくなりがち
- ✓ Pure Ruby impl can solve these cons
  pure Ruby実装ではこれらの欠点を解消できうる

Red Arrow Format

- ✓ Pure Ruby Apache Arrow reader/writer
  Apache Arrowフォーマットを読み書きできるRubyライブラリー
  - ✓ + Pure Ruby FlatBuffers reader/writer
    FlatBuffersフォーマットもなかったので作った
  - ✓ Apache Arrow is built on FlatBuffers
    Apache Arrowフォーマット内でFlatBuffersフォーマットを使っている
- ✓ A 2025 Ruby Association Grant project
  2025年度Rubyアソシエーション開発助成プロジェクトの1つ

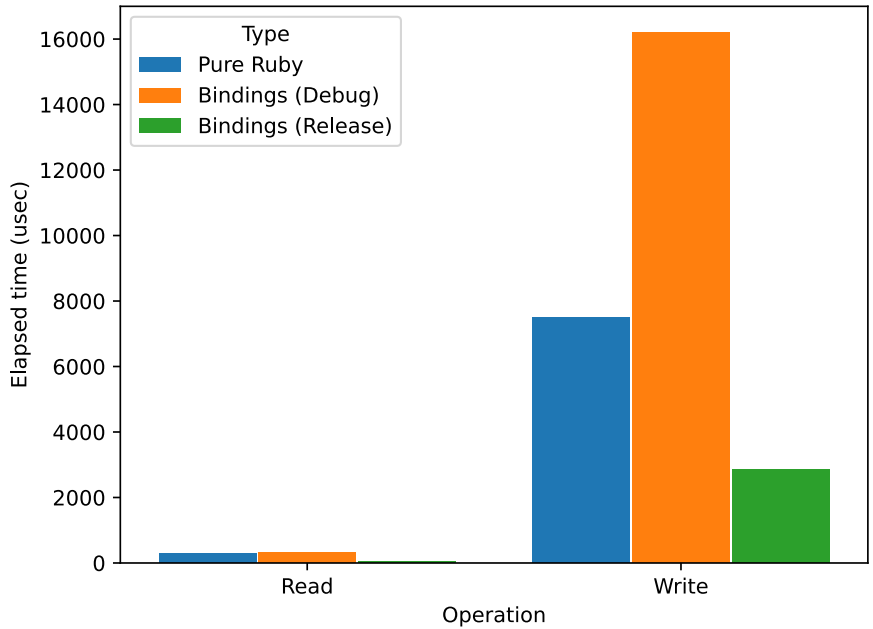
How much slower?
どのくらい遅いの？

- ✓ Lower is faster
  低いほど速い
- ✓ The pure Ruby is faster!!!
  pure Rubyの方が速い！
- ✓ The bindings use debug build🙈
  バインディングの方はデバッグビルドでした🙈



Release build
リリースビルド

- Lower is faster
  低いほど速い
- The bindings are 5.5x/1.5x faster!!!
  Bindingsの方が5.5x/1.5x倍速い！
- But... it's enough fast, isn't it!?
  でも、十分速くない！？



Why is the pure Ruby fast?

どうしてpure Ruby実装は速いの？

- ✓ I'm awesome! 
  私がすごい！ 
- ✓ Apache Arrow is awesome!
  Apache Arrowがすごい！

Why is Apache Arrow awesome?

なんでApache Arrowがすごいの？

Zero copy!

ゼロコピーですよ！！！

What factors slow down read/write?

読み書きが遅くなる原因は？

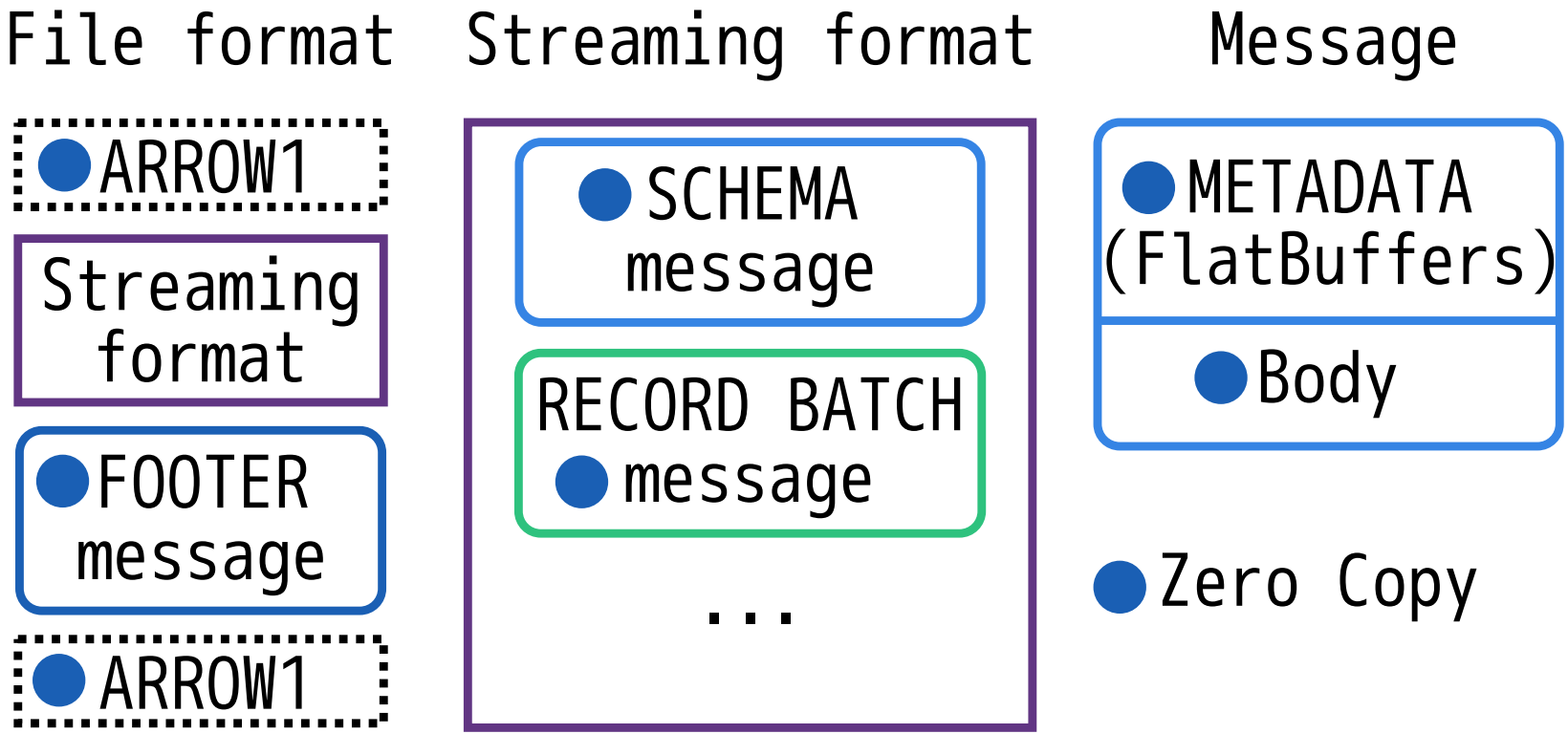
- ✓ Data copy
  データコピー
  - ✓ Memory allocation/copy are heavier operations
    メモリー確保とコピーは重めの操作
- ✓ Format conversion
  フォーマット変換
  - ✓ CPU is consumed
    CPUを使っちゃう

Zero copy
ゼロコピー

- ✓ No memory allocation/copy
  メモリー確保もコピーもない
- ✓ No format conversion
  フォーマット変換もない
- ✓ Just refer existing data
  単に既存データを参照するだけ

Zero copy in Apache Arrow format

Apache Arrowフォーマット内のゼロコピー



String zero copy in Ruby

Rubyで文字列をゼロコピー

```ruby
require "objspace"
data = "large data" * 1000
# Reuse leading string
ObjectSpace.memsize_of(data[0..-2]) # 10040
# Reuse trailing string
ObjectSpace.memsize_of(data[1..-1]) # 40 (Zero copy!)
# Reuse middle string
ObjectSpace.memsize_of(data[1..-2]) # 10039
```

Integer zero copy in Ruby

Rubyで整数をゼロコピー

- We can't zero copy integer in Ruby
  Rubyで整数をゼロコピーすることはできない
- Internal format is different
  内部表現が違う
  - Apache Arrow: C compatible
    Apache Arrow：C互換
  - Ruby: (c_format << 1) + 0x01
    Ruby：Cの表現を1ビット左シフトして空いたビットを1にする
  - 00000001 (1 in C) ->
    00000011 (1 in Ruby)

Zero copy parsing: Leading data

ゼロコピーでのパース：先頭部分

```ruby
# length(int32_t) + string + ...
data = [5].pack("l") + "abcde" + "..."
# Parse
length = data.unpack1("l") # Not zero copy
# Not zero copy! (Middle string can't be shared)
string = data[4, length]
```

Zero copy parsing: Trailing data

ゼロコピーでのパース：最後の部分

```ruby
# ... + length(int32_t) + string
data = "..." + [5].pack("l") + "abcde"
# Parse
# data[3..-1] is zero copy but
# temporary string is allocated
length = data[3..-1].unpack1("l")
# No temporary string
length = data.unpack1("l", offset: 3)
# Zero copy! (Trailing string can be shared)
string = data[7, length]
```

IO::Buffer

- Since Ruby 3.1 (still experimental)
  Ruby 3.1で導入（まだ実験的扱い）
- Efficient zero-copy buffer
  効率的なゼロコピーバッファー
- Use cases:
  用途：
  - Fiber::Scheduler
  - Parsing binary protocols!!!
    バイナリープロトコルのパース！

Buffer zero copy in Ruby

Rubyでバッファーをゼロコピー

```ruby
data = IO::Buffer.for("large data" * 1000)
# Reuse leading buffer
data.slice(0, data.size - 1) # Zero copy!
# Reuse trailing buffer
data.slice(1, data.size - 1) # Zero copy!
# Reuse middle buffer
data.slice(1, data.size - 2) # Zero copy!

data.get_string(1, 2) # Not zero copy...
```

Zero copy parsing with IO::Buffer

IO::Bufferでゼロコピーパース

```ruby
# "..." + length(int32_t) + string + ...
data = "..." + [5].pack("l") + "abcde" + "..."
buffer = IO::Buffer.for(data)
# Parse
length = buffer.get_value(:s32, 3) # Not zero copy
# Zero copy! But it's buffer not string...
string_buffer = buffer.slice(7, length)
# Not zero copy
string = buffer.get_string(7, length)
```

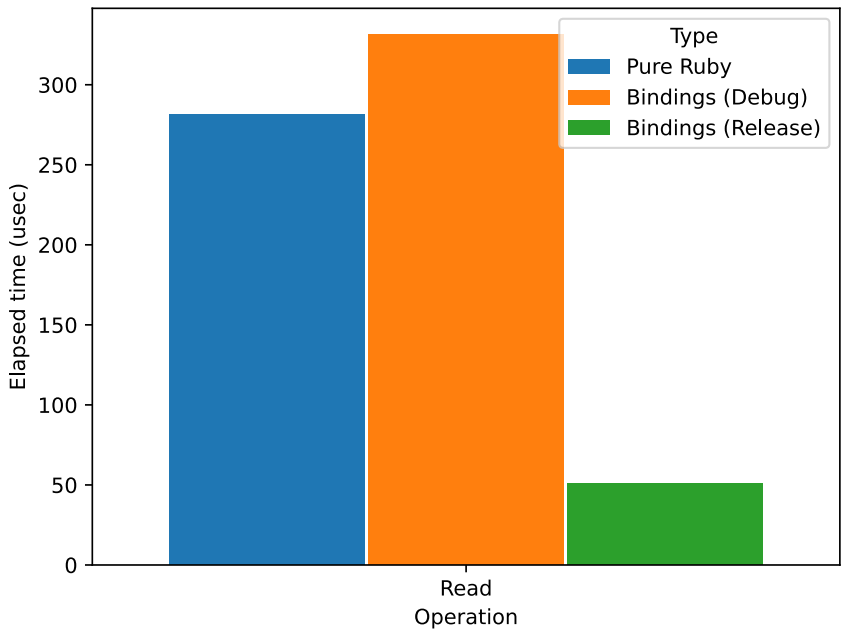
Zero copy parsing data in file

ファイル内のデータをゼロコピーでパース

```ruby
File.open("data.arrow", "rb") do |file|
  readonly = IO::Buffer::READONLY
  buffer = IO::Buffer.map(file, nil, 0, readonly)
  # file.read copies data from file but...
  length = buffer.get_value(:s32, 3) # doesn't copy
  string_buffer =
    buffer.slice(7, length) # doesn't copy
end
```

Zero copy parsing result
ゼロコピーでパースした結果

- ✓ Lower is faster
  低いほど速い
- ✓ Faster than debug build
  デバッグビルドよりは速い
- ✓ Slower than release build
  リリースビルドよりは遅い

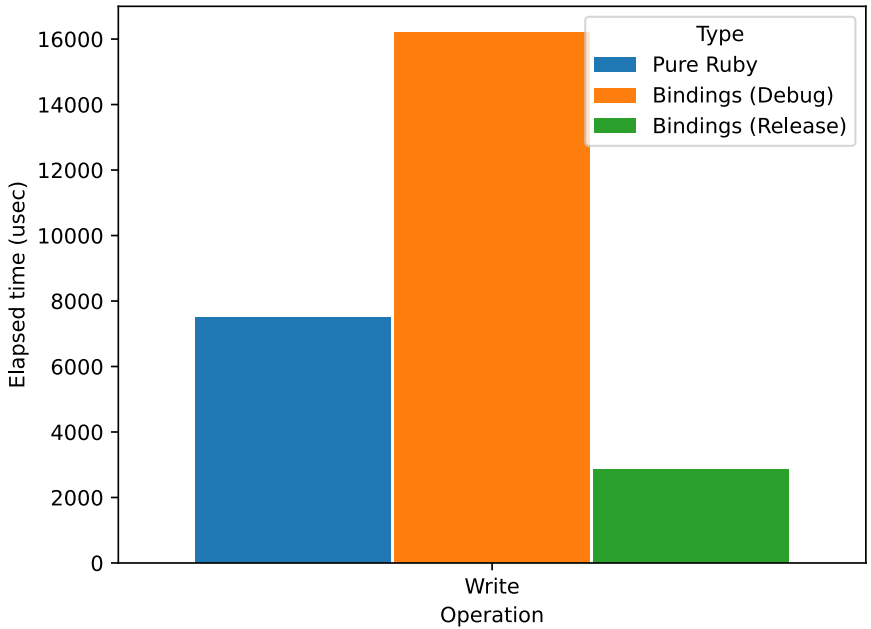


Zero copy writing
ゼロコピーで書き込み

- ✓ Can't do it😞
  できない😞
- ✓ Need to concatenate all buffers
  すべてのバッファーを結合しないといけない
- ✓ How to implement fast writer...?
  どうやって高速な書き込みライブラリーを実装したの？

Writing result
書き込み結果

- ✓ Lower is faster
  低いほど速い
- ✓ Faster than debug build
  デバッグビルドよりは速い
- ✓ Slower than release build
  リリースビルドよりは遅い



How to implement?
どう実装したの？

- Nothing special but fast...
  特別なことはしていないのに速い
  - output.append_as_bytes(string)
  - output.append_as_bytes([integer].pack("L<"))
- Want to eliminate temporary objects
  一時オブジェクトを減らしたい
  - e.g. [integer].pack("L<")
  - Thinking good API but no idea...
    いいAPIを考えているけど思いつかない。。。

Motivation
なんで作ったの？

Spread Apache Arrow in Ruby ecosystem!

Rubyエコシステム内でApache Arrowをもっと広めたい！

What do we do for it?
なにをすればいいの？

- ✓ Just add support for Arrow in/out
  Apache Arrowでデータを入出力できるようにするだけでいいっす
  - ✓ e.g. JSON/Apache Arrow output Web app
    例：JSONでもApache Arrowでも出力できるWebアプリ
  - ✓ e.g. CSV/Apache Arrow input Web app
    例：CSVでもApache Arrowでもインポートできるアプリ
- ✓ Existing tools integrate with your apps
  既存のデータ処理ツールがあなたのアプリと連携するはず

Future work
今後の展望

If you're interested in this, I can support you!
興味があるならサポートできるよ！

- ✓ Add fast bit operations to Ruby
  Ruby本体に高速なビット演算を追加
  - ✓ Apache Arrow uses bitmap
    Apache Arrowはビットマップを使っている
  - ✓ e.g. popcount: Feature #20163
- ✓ Add fast binary write API to Ruby
  Ruby本体に高速なバイナリーデータ書き出しAPIを追加

Future work
今後の展望

If you're interested in this, I can support you!
興味があるならサポートできるよ！

- ✓ Add SIMD JIT to Ruby
  Ruby本体にSIMD用コードを生成するJITを追加
  - ✓ Apache Arrow is SIMD friendly (aligned and continuous)
    Apache ArrowはSIMDしやすい（アラインされた連続データ）
  - ✓ MemoryView can be used for this
    MemoryViewを使えるはず
  - ✓ MemoryView is a typed array
    MemoryViewは型付きの配列

SIMD JIT

```ruby
sum = 0
# Vectorize automatically by JIT
int32_array_memory_view.each do |value|
  sum += value
end
sum
```

Wrap up
まとめ

- Pure Ruby reader/writer is fast enough
  Apache ArrowのRuby実装は十分速い
  - Let's make your apps Apache Arrow ready
    みんなのアプリをApache Arrow対応にしてね
- Let's improve it together!
  Join Red Data Tools! https://red-data-tools.github.io/
- I'll be at @.bookstore today's breaks
  今日のお昼休みとおやつ休みは本屋さんにいるよ
  - Come there if you want to talk to me!
    お話したい人は来てねー！