Ruby向け並列化機構 Guildの試作

笹田 耕一

クックパッド株式会社
<ko1@cookpad.com>

本発表の概要

- Ruby3 で並列処理に対応させたい
  - スレッドプログラミング == 共有メモリモデルは難しい
  - 共有メモリモデルではないプログラミングモデルが欲しい
- Guild の提案（笹田2016）
  - オブジェクトはほぼ共有しない（！共有メモリモデル）
    - でも、たまには共有したい → 共有可能オブジェクト
  - メールボックスによるGuild間通信
    - 複製　全部コピー
    - 移籍　ヘッダだけコピー（中身はコピーしない）
  - 今回は少し進めて試作を報告

前の発表
（2016年10月　第111回プログラミング研究会）

Ruby 3 に向けた 新しい並行実行モデルの提案

笹田 耕一
松本 行弘

heroku

背景
Rubyって？

- オブジェクト指向スクリプト言語
  - かなり厳密なOO（vs. Python）
  - 記述量が少ない
  - 世界中で、結構人気
- ウェブアプリケーション開発によく利用
  - Ruby on Rails というフレームワークが大流行（2005年〜）
  - クックパッドとか
- 性能は遅い
  - でも、もっと速くして欲しい（クックパッドとか）
  - 次のリリースで、JITコンパイラが試験的に導入予定

CM：Ruby Hack Challenge
9/10, 9/13 の二日間でRubyをハッキング

https://cookpad.connpass.com/event/95127/

背景
Ruby でも並列処理書きたい

- 並列計算機をきちんと使いたい（SWoPPっぽい）
  - 例：ウェブのリクエストごとに並列に処理したい
  - 現状は Process を fork するしかない
  - スレッド（Threadクラス）は並列処理非対応
    - ジャイアントロックがあるため。並行処理は OK
    - (1) 処理系の実装が楽だから
    - (2) 並列処理なスレッドプログラミングは大変だから

背景
スレッド難しい…

- **共有メモリモデル**：共有データに同期が必須
  - 発見が難しい。特に大規模な逐次プログラムの並列化
  - 同期処理を忘れずにいれるのが難しい
- デバッグ難しい（非決定的）
- パフォーマンス出すの難しい

Ruby 的に許しがたい（個人の感想です）

まぁ、Rubyだし、そこそこでいいんじゃない（個人の感想です）

Ruby3 の目標（並行並列処理）

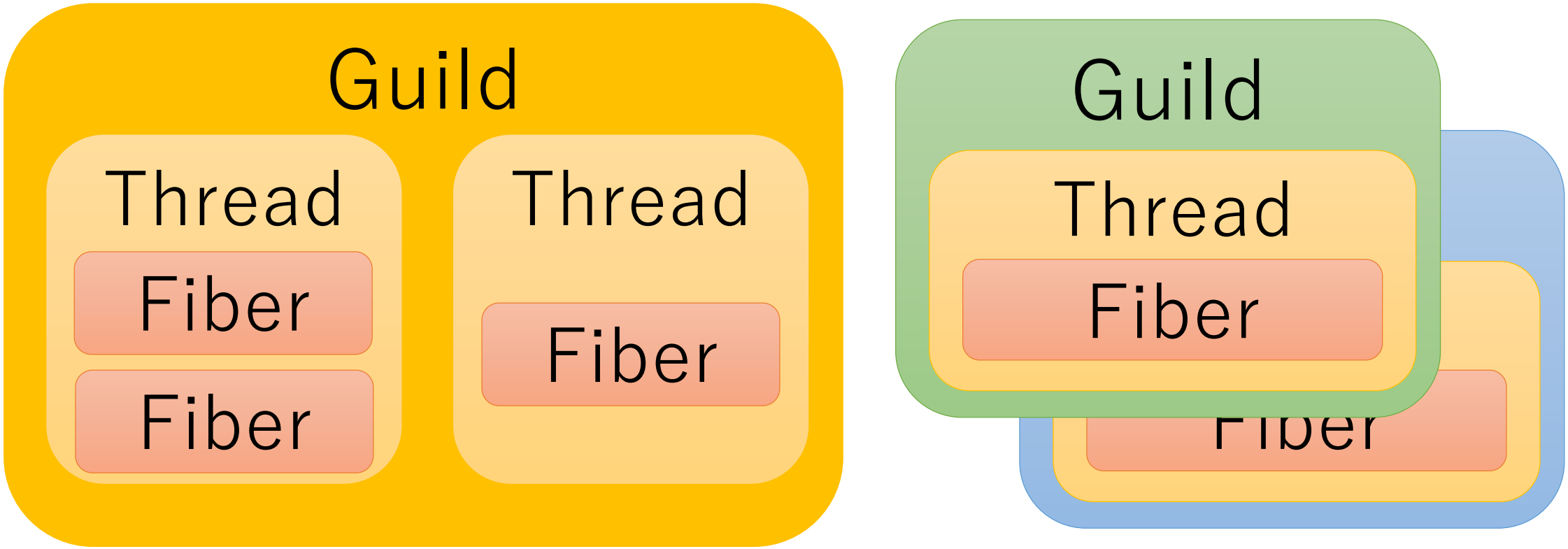
- Ruby 2 との互換性の維持
- 並列処理のサポート
- 共有メモリモデルを用いない安全なプログラミングの実現
- 高速な通信の実現
- 現実的な処理系開発・維持コストの実現
  - 処理系のそこら中に細粒度同期を必要としない

Guildの提案

- Guildという並行実行単位を用意
  - 異なるGuildに属するスレッドは**並列処理可能**
- **共有メモリモデルではない**（なんて言うんだろう？）
  - ほとんどのオブジェクトは、1つのGuildに所属
  - 特別にいくつかの**共有可能オブジェクト**
- 複製と移籍によるGuild間通信

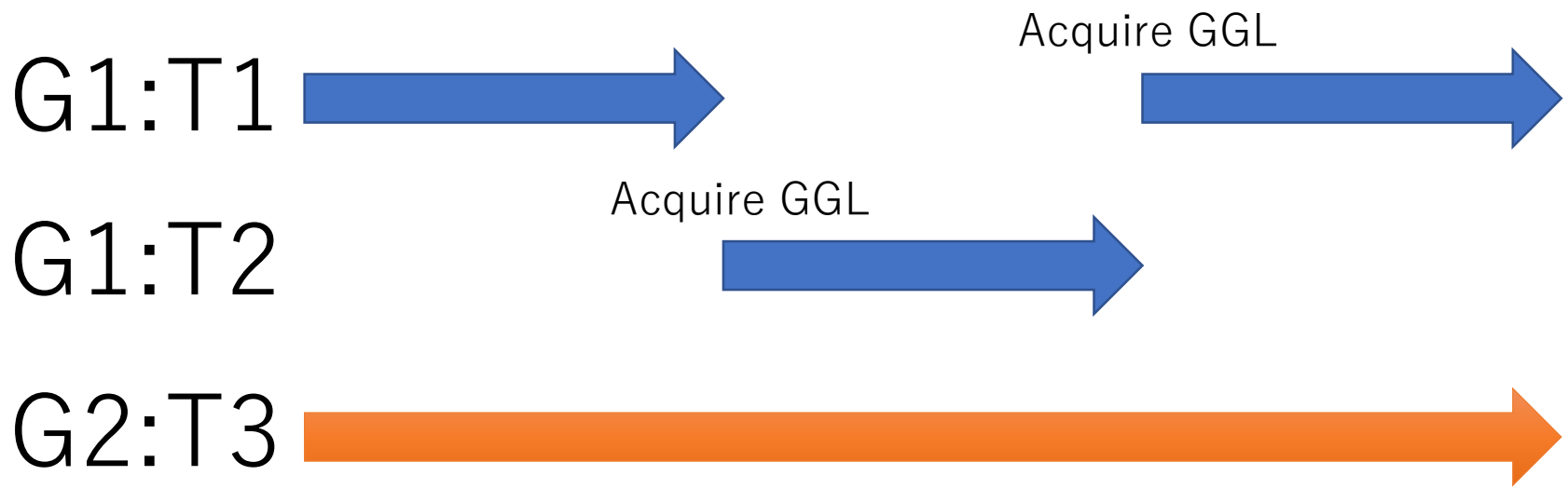
Guildとスレッド

- Guildは1つ以上のスレッドを持つ
- スレッドは1つ以上のFiber（コルーチン）を持つ



Guildとスレッド

- 同じGuildに所属するスレッドは並列処理しない（byジャイアントロック）→ Ruby2 と互換
- 所属するGuildの異なるスレッドは並列に実行

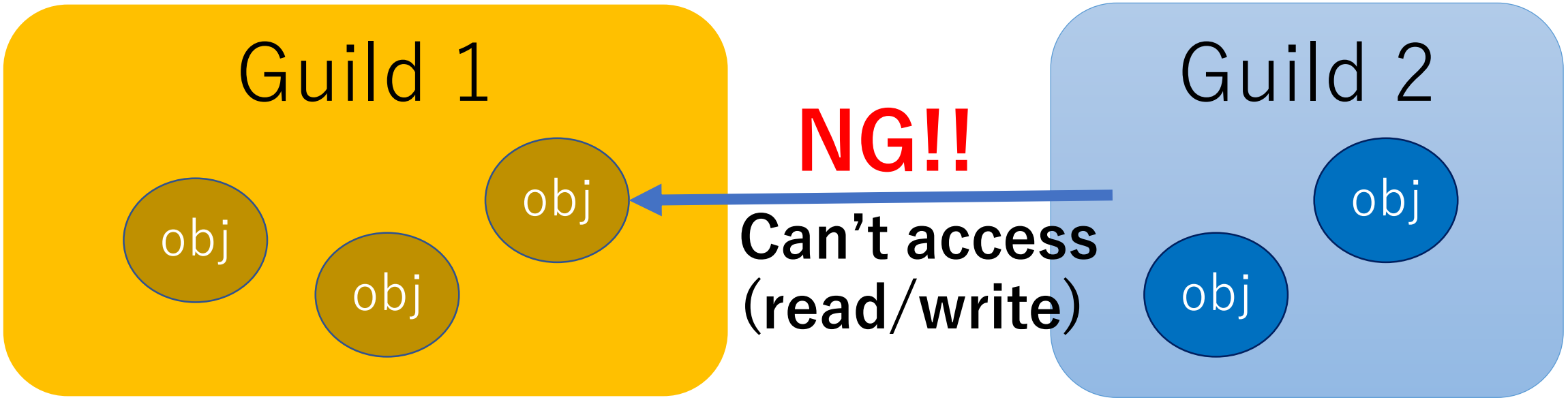


Guildの作成

```
g1 = Guild.new do
  expr1
end
g2 = Guild.new do
  expr2
end
# 2つのGuild（と暗黙にスレッド）を作成
# expr1 と expr2 は並列に実行
```

Guildとオブジェクトの所属

- ほとんどのオブジェクトは「**共有不可オブジェクト**」
- 他のGuildからアクセスはできない



共有可能オブジェクトと共有不可オブジェクト

- 並行処理プログラムを観察すると、共有するデータはほんの一部（特にOO言語）
  - その一部に同期が必須
- 逐次プログラム作成→共有不可オブジェクトを利用
- 共有する場合、共有可能オブジェクトを利用
  - 利用は同期が必須で面倒だが、利用しないとプログラムが動かない
    → フェイルセーフ

共有可能オブジェクト
今のところ5種類

- 共有コンテナオブジェクト
- 不変オブジェクト
- クラス・モジュールオブジェクト
- 隔離Procオブジェクト
- Guild 制御オブジェクト

共有可能オブジェクト
共有コンテナオブジェクト

- いわゆるコンテナオブジェクト
- アクセスするのに同期が必須（例：ロック）
- Software Transactional Memory (STM)が良さそう
  - ロック順による同期が不要
- （具体的には何も決めていない）

- どれだけ、ミスがないようなデザインができるか？

ミスが起きそうな例（1）

```
# 間違ったコード
n = s.transaction{ s[:n] }
s.transaction{ s[:n] = n+1 }

# 正しいコード
s.transaction{ s[:n] = s[:n] +1 }
```

ミスが起きそうな例（2）

```
# 間違いそうな例
def get_num(s)
  s.transaction{ s[:n] }
end
def set_num s, n
  s.transaction{ s[:n] = n}
end
```

```
# 間違い
set_num( s, get_num(s) + 1 )

# 正しいコード
s.transaction do
  set_num( s, get_num(s) + 1 )
end
```

Open issue

どうすれば言語デザインでこんなミスが排除できるか？

共有可能オブジェクト
不変オブジェクト

- 不変オブジェクトは変らないので、データレースを起こさないため、共有しても問題ない
- freeze （書き込み禁止）とは違う
  - 自身が freeze されており、参照しているオブジェクトが共有可能オブジェクト

共有可能オブジェクト
クラス・モジュールオブジェクト

- 検討した結果、これらは共有可能に
  - クラス（モジュール）もオブジェクト
  - 共有可能オブジェクト含む全てのオブジェクトはクラスへの参照を持つ
  - 同じオブジェクトIDだが実体を変える、という案も検討したが、Guild間でメソッドリストが異なると使いづらそう、ということになった
- mutable なので、適切なプロトコルが必須（詳細は報告参照）

共有可能オブジェクト
隔離Procオブジェクト

- Proc（クロージャ）オブジェクトは自由変数への参照を持つ→共有できない（ニーズあり）
- 自由変数へのアクセスができないProcを Proc#isolate で生成
  - アクセスするようなら例外
- Guild.new では暗黙にこれを利用

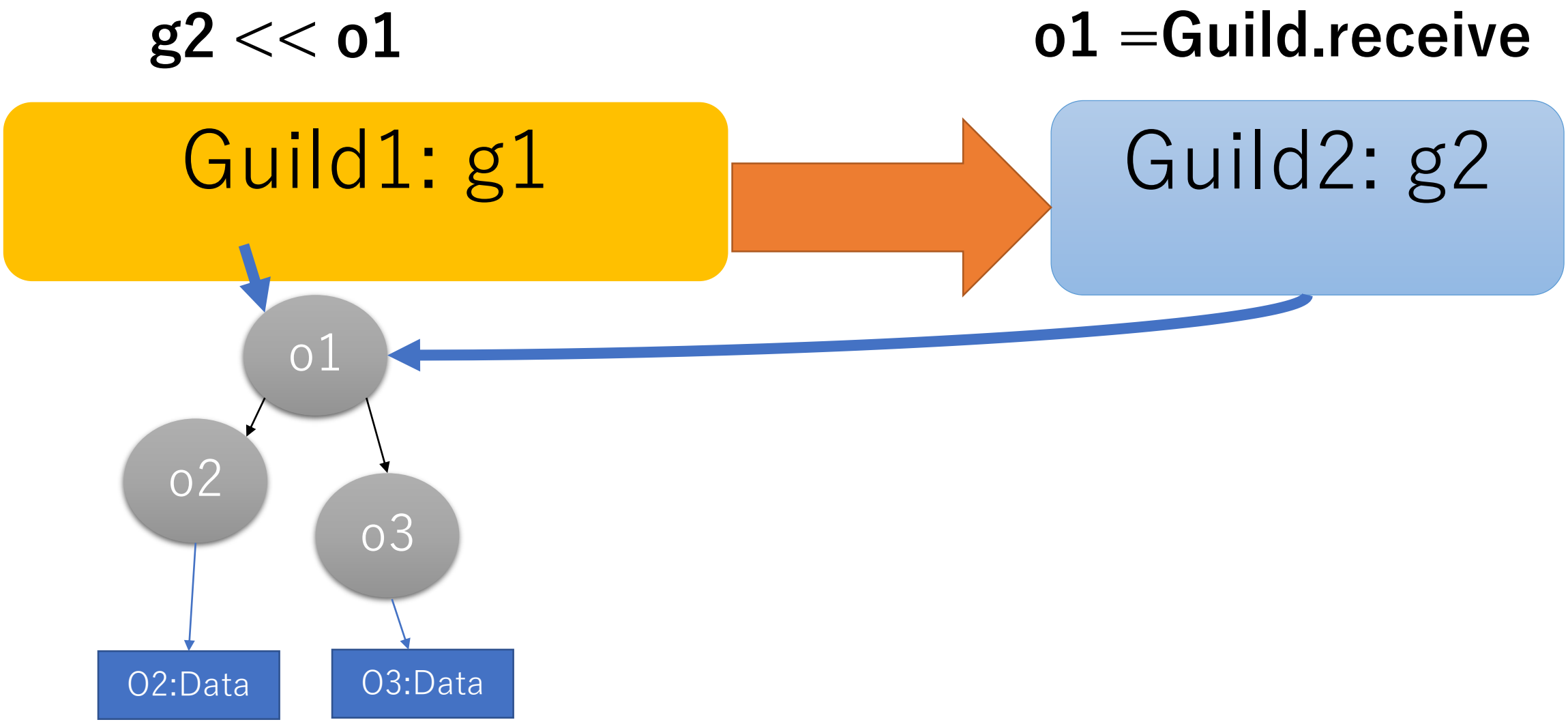
共有可能オブジェクト
隔離Procオブジェクト

```
# Initial block for Guild is isolated proc
g1 = Guild.new do
  expr1  # Make isolated block and invoke
end
g2 = Guild.new do
  p g1 #=> RuntimeError (can’t access “g1”)
       #   because block is isolated
end
```

Guild間通信

- 通信路どうする？
  - チャンネル（like Go-lang）を使う？
  - メールボックス（like Erlang）を使う？
  - 今はメールボックス
- 二つの送信手法
  - 複製
  - 移籍
  - （共有可能オブジェクトはリファレンスを送信）

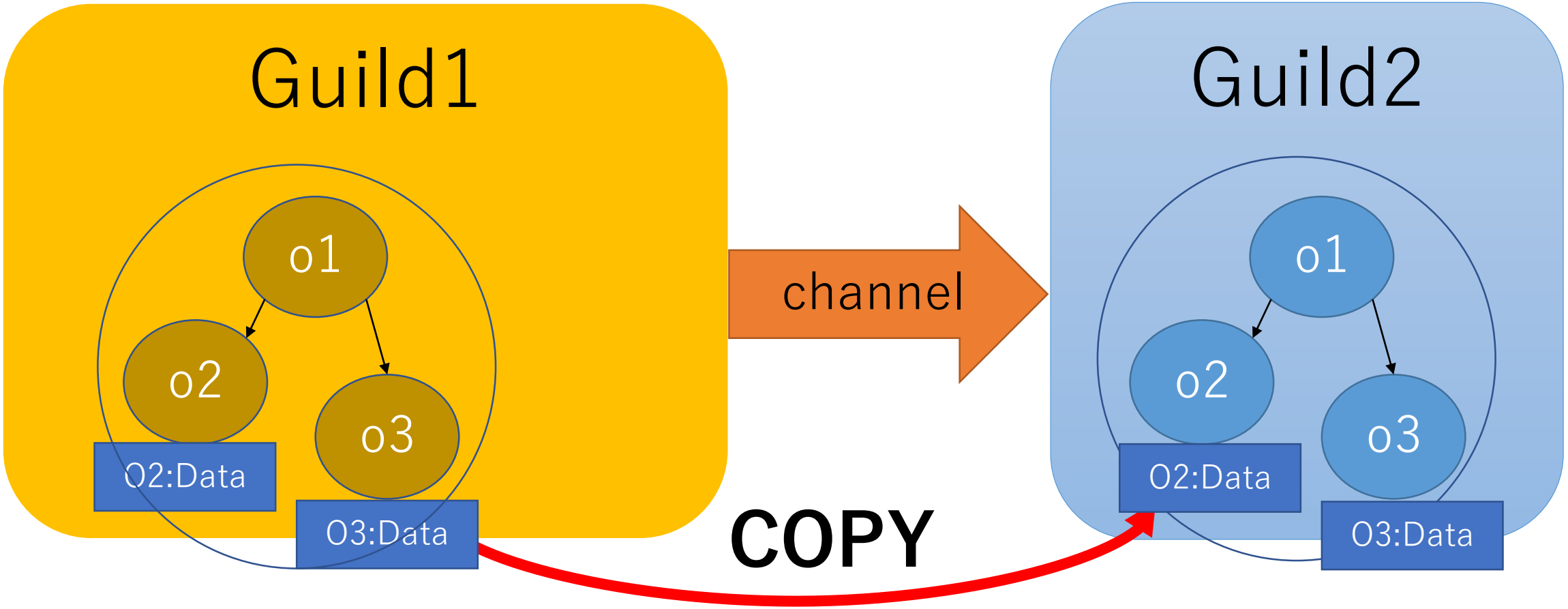
共有可能オブジェクトの送信



共有不可オブジェクトの送信
(1) 複製による送信

g2 << o1

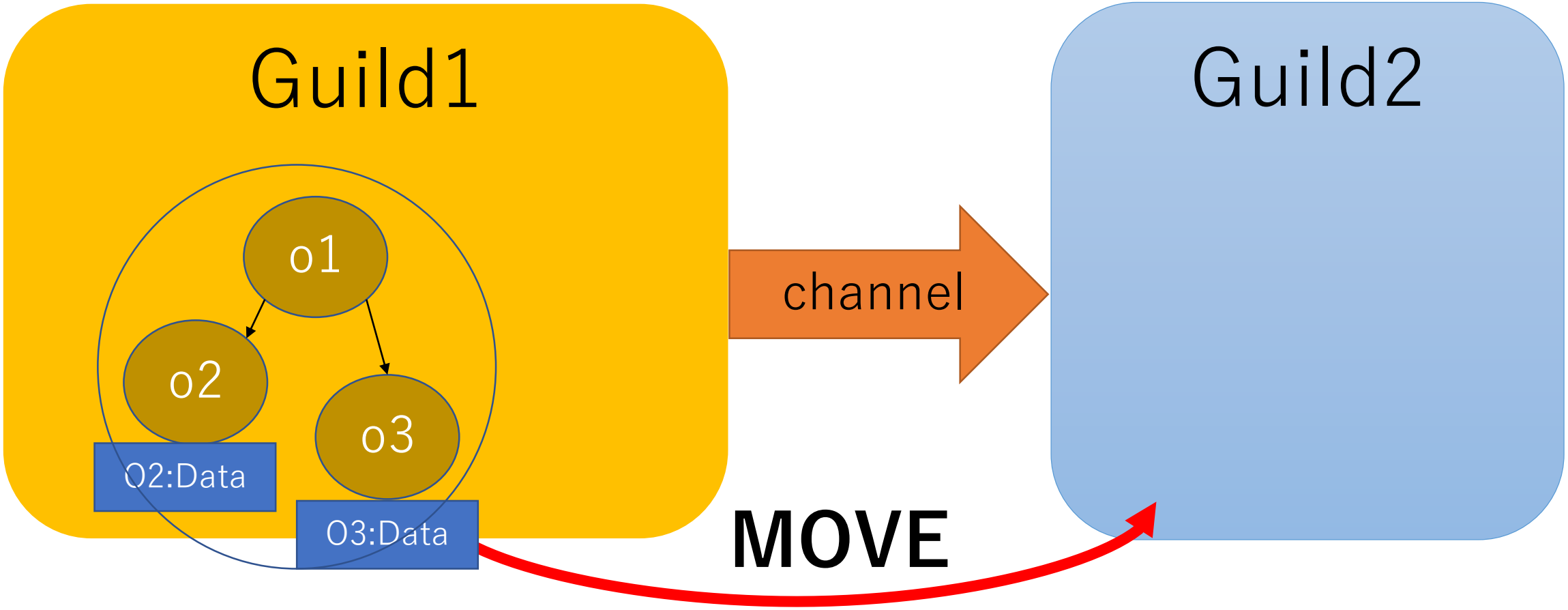
o1 = Guild.receive



共有不可オブジェクトの送信
(2) 移籍による送信

g2.move(o1)

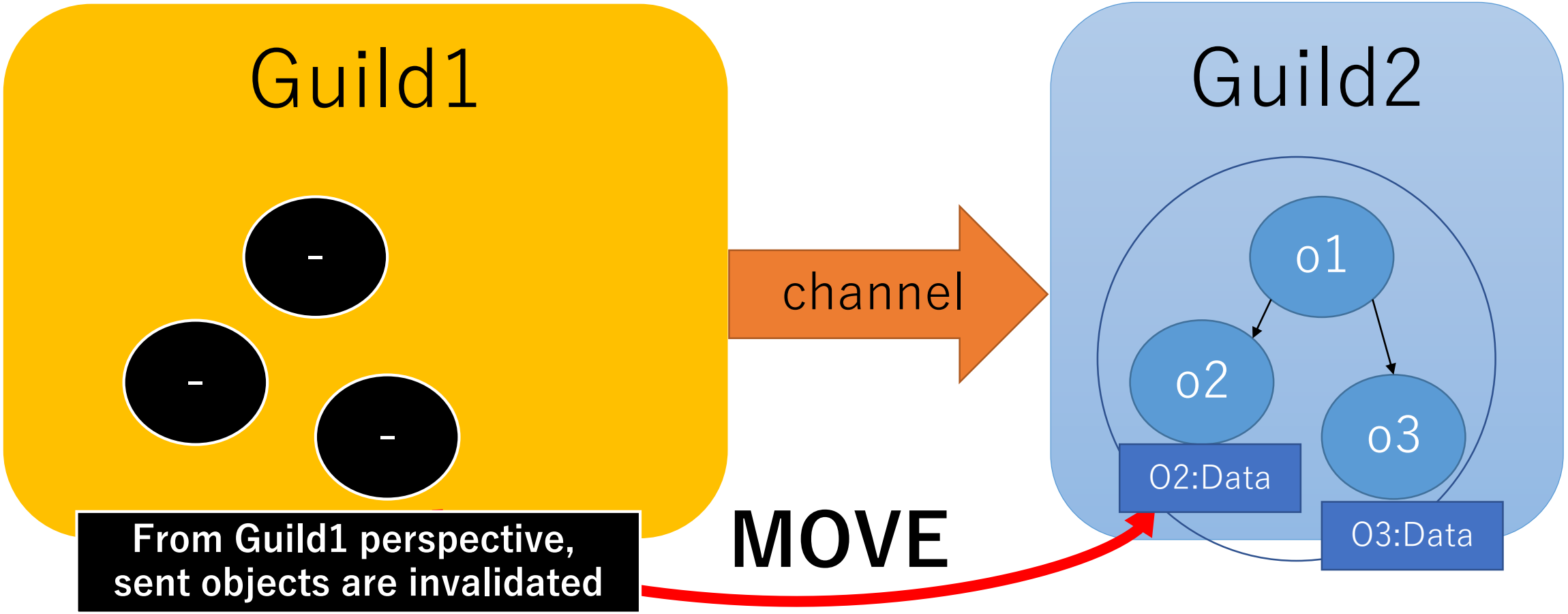
o1 =Guild.receive



共有不可オブジェクトの送信
(2) 移籍による送信

g2.move(o1)

o1 = Guild.receive



共有不可オブジェクトの送信
(2) 移籍による送信

- 送ったあと、送ったオブジェクトが不要であれば（そういうケースはそこそこありそう）、移籍の方が高速
- 例
  - 大きな文字列
  - I/O オブジェクト（ソケットなど）

Guildの実装

- 素直に Guild + Thread を実装
  - 共有データ用に、処理系に排他制御を導入
  - 不完全（難しい→スレッドなんて使うもんじゃない）
- GC: 停止GC
  - 全Guildを止めて marking（sweep は各Guildで）
  - 同期漏れがあり、うまく動いていない（難しい）

実験

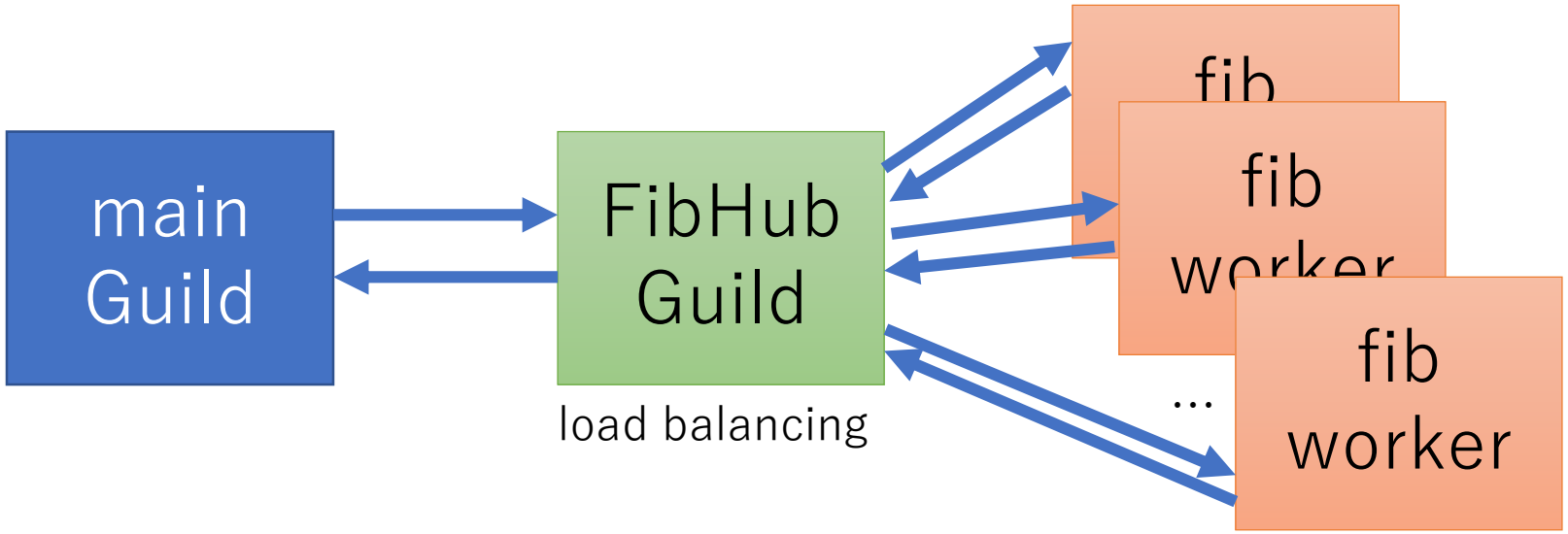
- いくつかの実験
  - (1) 生成・通信（ピンポン）・終了
  - (2) 数値計算タスクの並列処理
  - (3) 文字列処理
- 実験環境
  - Intel(R) Xeon(R) CPU E5-2630 v4 @ 2.20GHz
    - 10 cores x 2 Hyper Threading
    - x 2 CPUs
    - = 40 Hardware threads

実験：生成・通信（ピンポン）・終了

	Execution time（10万回）
Proc	0.09
Fiber	0.24
Thread	3.23
Guild	6.48
Process	63.40

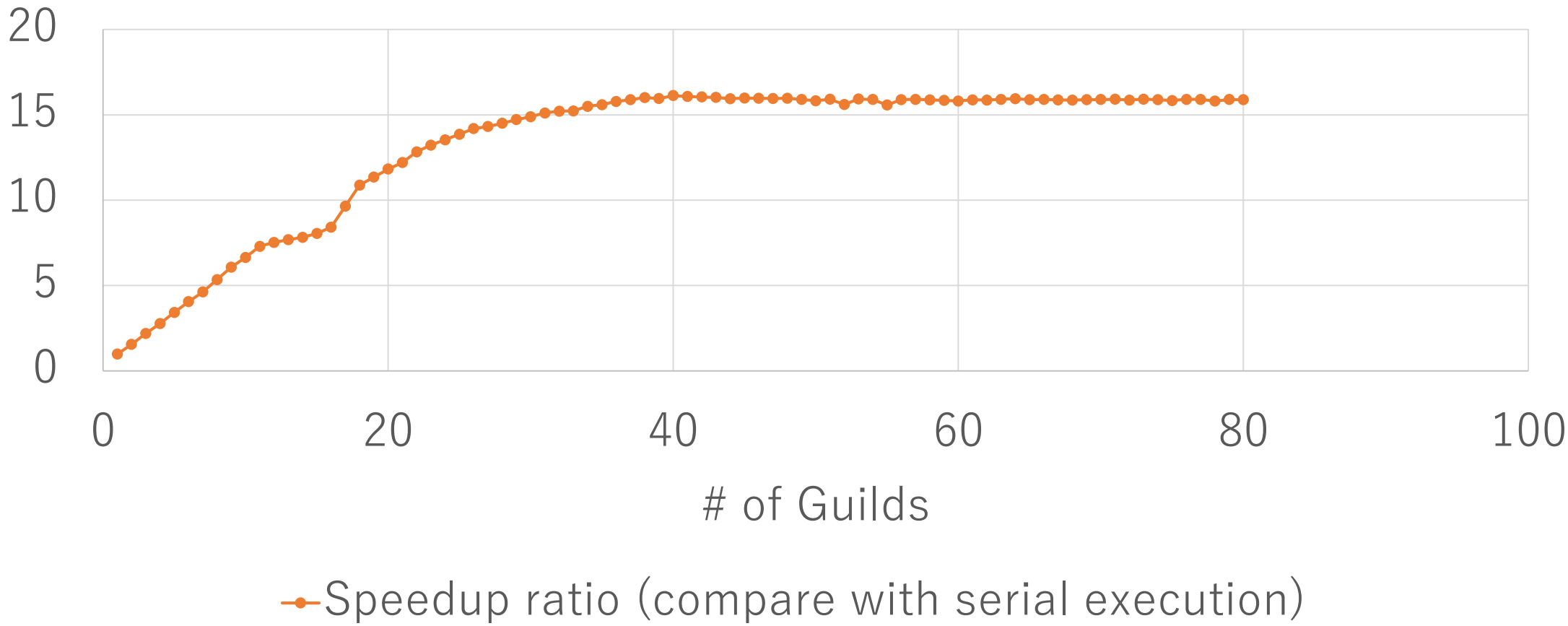
実験 (on 40 vCPUs)

- Workload
  - Calculate **fib(23)** x 100_000 times
    - Serial version: `100_000.times{ fib(23) }`
    - Guild version:



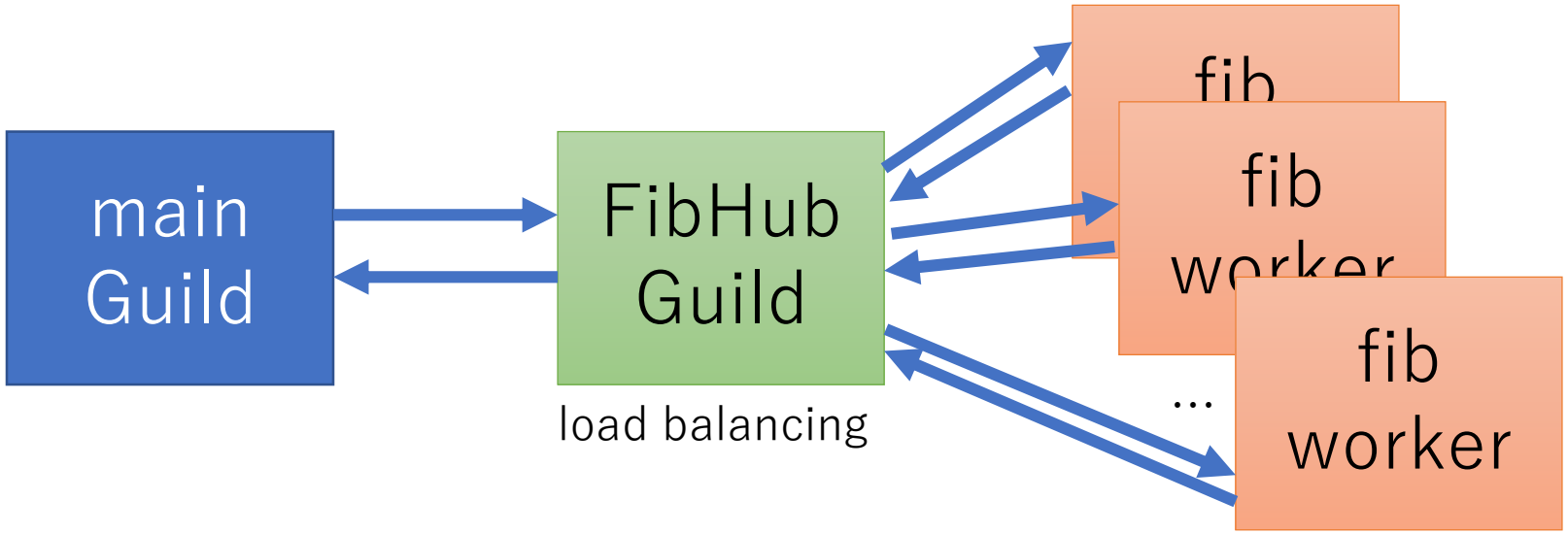
We can change # of workers

fib(23) with # of Guilds on 40 vCPUs

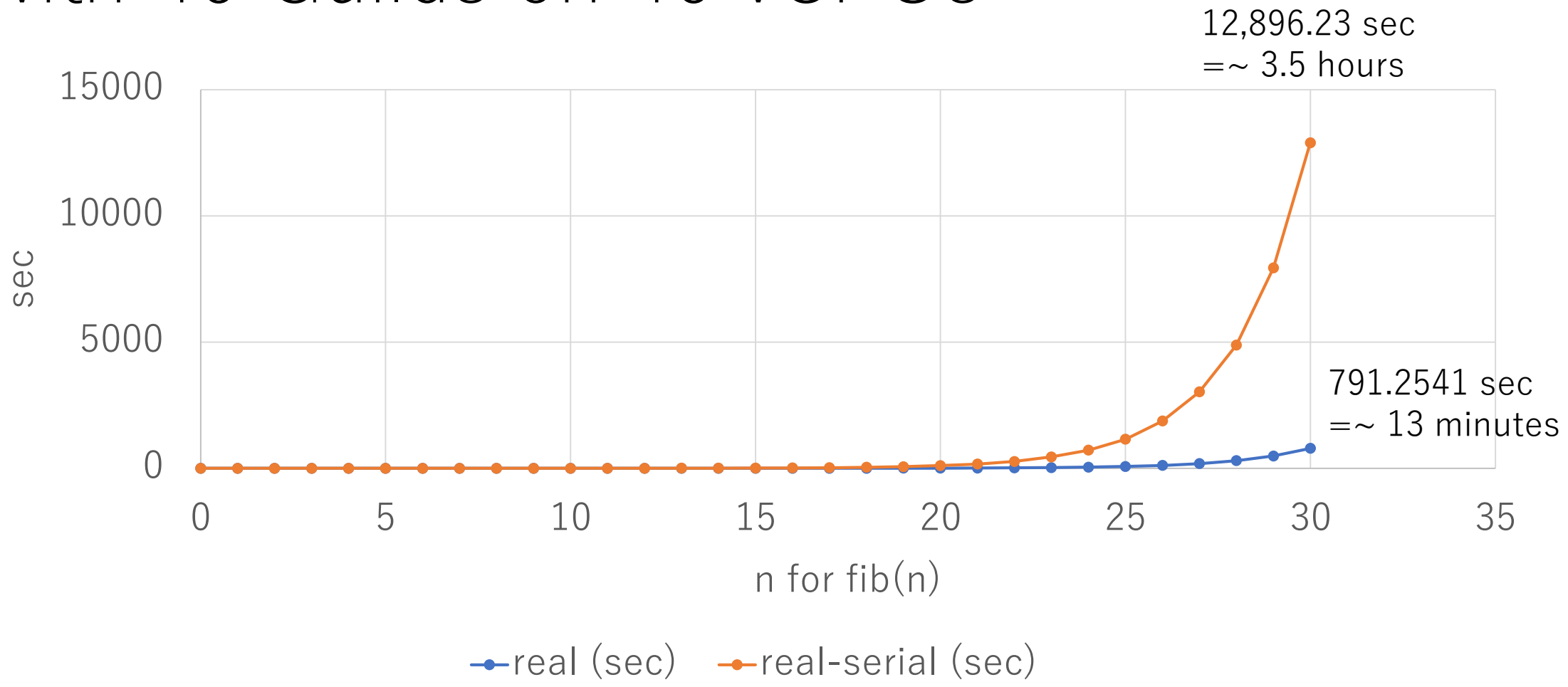


実験 (on 40 vCPUs)

- Workload
  - Calculate **fib(n)** x 100_000 times (0 ≦n≦30)
    - Serial version: 100_000.times{ fib(23) }
    - Guild version: 40 Guilds

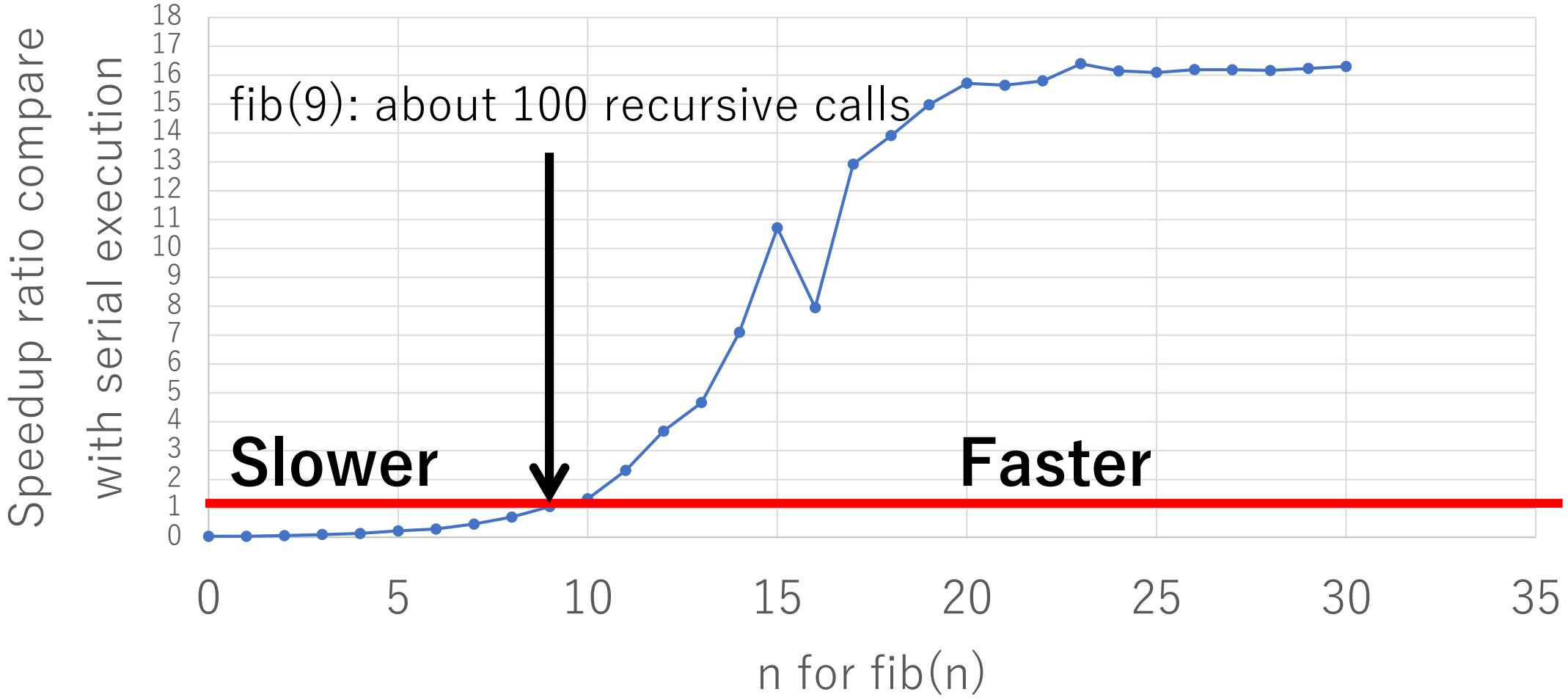


Execution time (sec) of **fib(n) x 100_000** with 40 Guilds on 40 vCPUs



Order of fib(n) is “O(2^n)”

fib(n) with 40 Guilds on 40 vCPUs



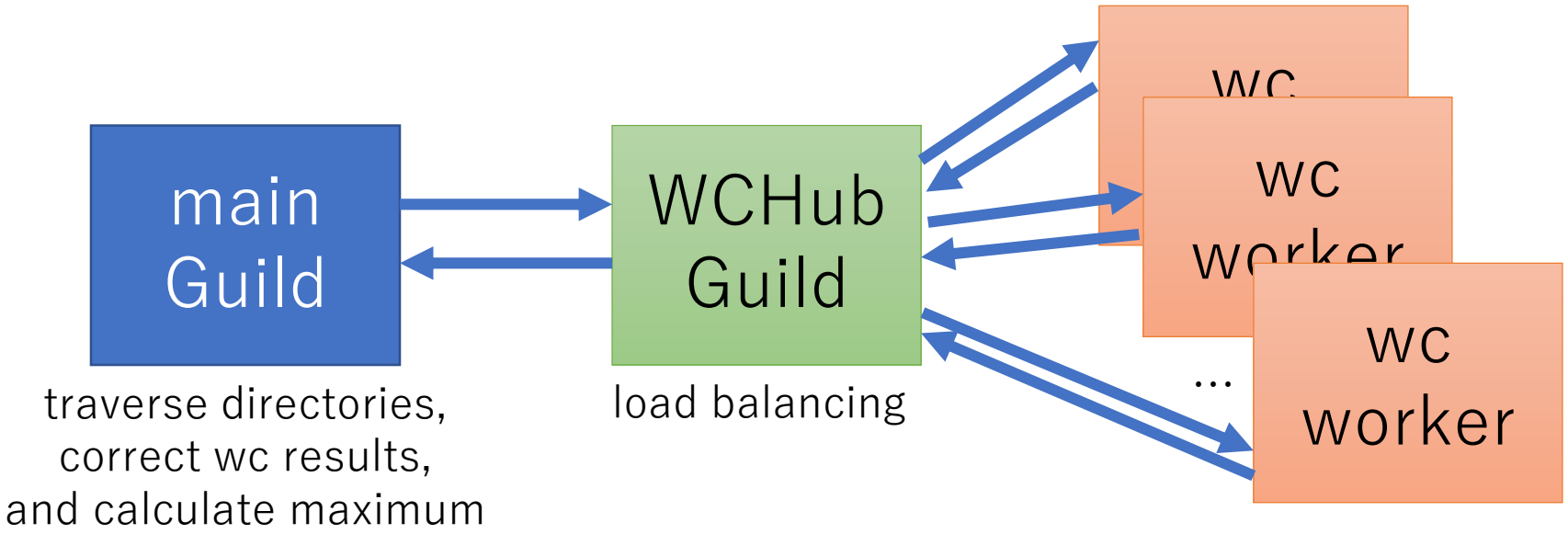
実験 (on 40 virtual CPU)

- Workload
  - Calculate wordcount for files and find a file which contains maximum number of words.
    - on “ruby/test/**/*” files (1,108 files)

```
def word_count file
  r = File.read(file).b.upcase.split(/¥W/).uniq.size
end
```

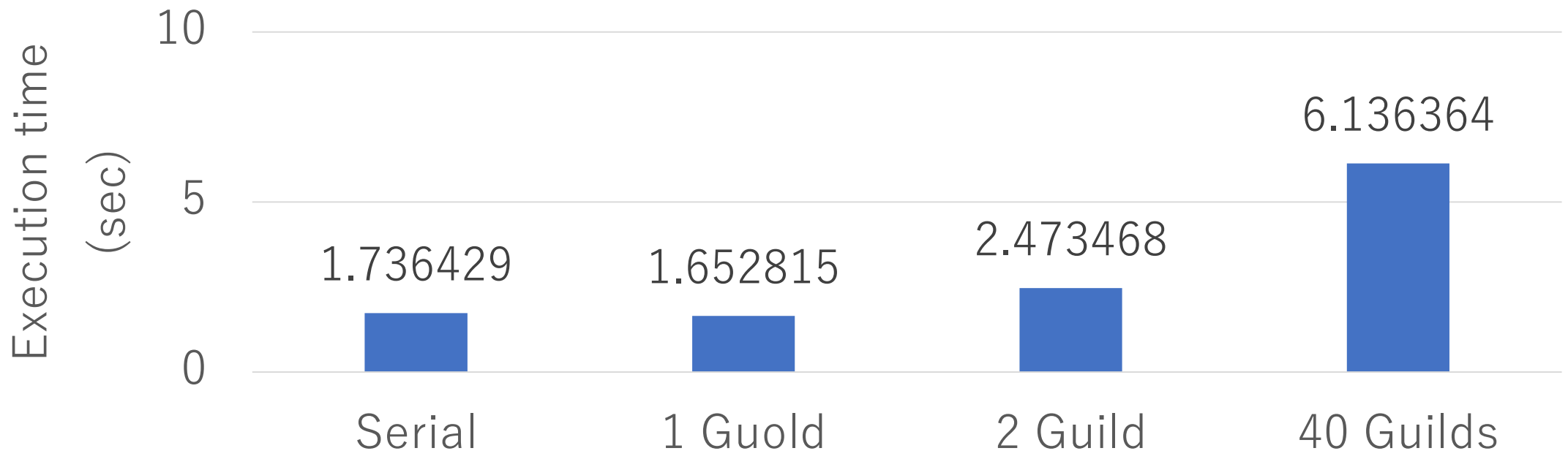
実験 (on 40 virtual CPU)

- Workload
  - Calculate wordcount for files and find a file which contains maximum number of words.
    - on “ruby/test/**/*” files (1,108 files)



We can change # of workers

実験 (on 40 virtual CPU)



It is **SLOW** with multiple Guilds
because GC/object allocation require naïve global locking
(current implementation limitation) and huge contentions.

今後の課題（イッパイ）

- フィージビリティテスト with 実装
- Guild 終了時の挙動
- C API の変更

本発表の概要

- Ruby3 で並列処理に対応させたい
  - スレッドプログラミング == 共有メモリモデルは難しい
  - 共有メモリモデルではないプログラミングモデル
- Guild の提案（笹田2016）
  - オブジェクトはほとんど共有しない（！共有メモリモデル）
    - でも、たまには共有したい → 共有可能オブジェクト
  - メールボックスによるGuild間通信
    - 複製　全部コピー
    - 移籍　ヘッダだけコピー（中身はコピーしない）
  - 今回は少し進めて試作を報告

Thank you for your attention

Ruby向け並列化機構 Guildの試作

We are hiring!

笹田 耕一

クックパッド株式会社
<ko1@cookpad.com>