Tsutomu Katsube

RubyKaigi 2026

Tsutomu Katsube

- ✓ Job: Software Engineer
- ✓ Hobby: OSS Activities
- ✓ GitHub and X: @tikkss

Starting point

きっかけ

About 2 years ago

Red Data Tools

- ✓ A community to provide data processing tools for Ruby
  Ruby 用のデータ処理ツールを提供するコミュニティ

Red Datasets

- A RubyGem for easy use to common (ML) datasets
  機械学習などでよく使われるデータセットを簡単に使える gem
  - Iris, MNIST, CIFAR, etc.

Slow test suite

- ✓ Tests became slower as the supported datasets grew
  サポートされるデータセットが増えるにつれて、テストが遅くなっていた

Powered by Rabbit 4.0.1

Discussion

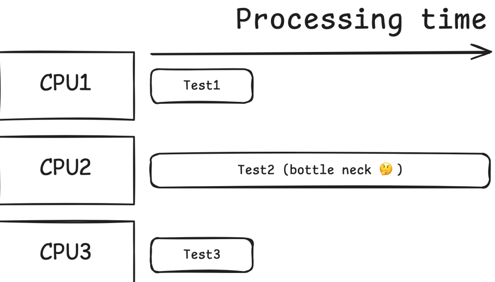
- ✓ How to speed up the test suite such as parallelizing them?
  テストを並列実行するなどして高速化できないか？

First step

- ✓ Speed up each test

Powered by Rabbit 4.0.1

Why?



✓ Test2 is bottleneck even with parallelization

並列化したとしても、遅いテスト（Test2）がボトルネックになるから

Next step

- ✓ Parallelize the tests 👈Today's topic
  テストの並列実行です
  - ✓ test-unit natively adds support for parallel test running😁
    test-unit 本体でテストの並列実行をサポートすることにした

Powered by Rabbit 4.0.1

Why?

- Sutou Kouhei
  - Founder of [Red Data Tools]
  - Maintainer of test-unit (as a bundled gem)

Day3 11:30 - 12:00 Small Hall

Plan

- ✓ [x] Thread
- ✓ [x] Multiprocess 👈Today's topic
- ✓ [ ] Ractor

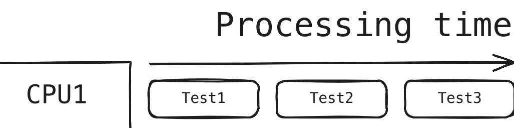
Basics

What is a parallel test running?

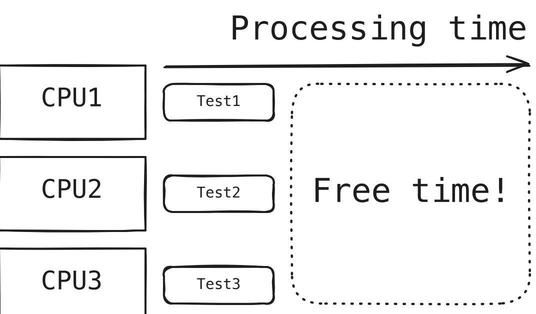
並列テスト実行とはなんでしょうか？

Powered by Rabbit 4.0.1

Sequential running



Parallel running



Why portable?

- ✓ For working across environments
  色々な環境で動いてほしいため

Why fast?

- ✓ For faster feedback loop
  より速いフィードバックループのため

Topics

- ✓ Issues
  課題
- ✓ Solutions
  解決策
- ✓ Demo
  デモ
- ✓ Future
  今後のこと

Issues

- ✓ Portable 👈
- ✓ Fast

Portable Requirements (1)

- ✓ Don't depend on Unix-specific features such as `fork`

  fork のような Unix 固有の機能に依存しない

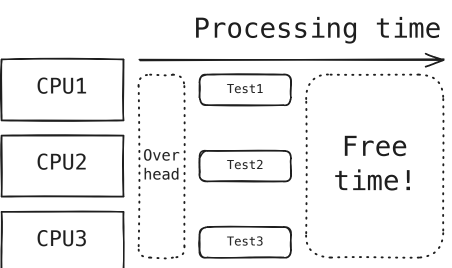
Launch a child process

- ✓ `Kernel.#fork`
- ✓ `Kernel.#spawn`

Portability

- ✓ `Kernel.#fork`
  - ✓ Works only on Unix-based platforms
    fork は Unix 系のみで動く
- ✓ **`Kernel.#spawn`** 👈
  - ✓ Also works on Windows
    spawn は Windows でも動く

Overhead of launching a process



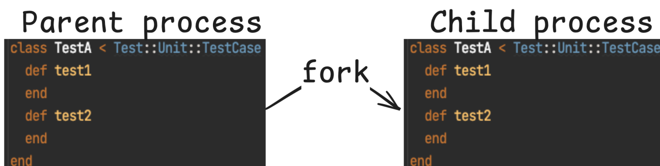
Overhead

- ✓ **`Kernel.#spawn`** has more overhead than `Kernel.#fork`
  spawn は fork よりもオーバーヘッドが大きいので、速さは少々犠牲になる
  - ✓ Because `Kernel.#fork` uses the COW (Copy on Write)
    fork はコピーオンライトを利用するので

Kernel.#fork

- Creates a **copy** of the process

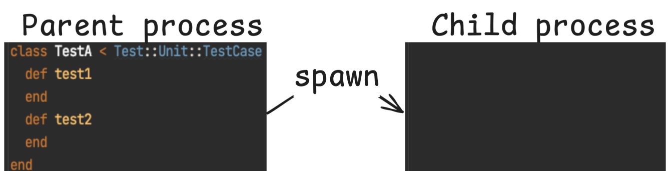
  fork はプロセスのコピーを作る



Kernel.#spawn

- ✓ Creates a clean environment (similar to Ruby::Box)

  spawn はまっさらな環境を作る（Ruby::Box.new に似ている）



Issues (1)

親プロセスの状態をどうやって復元する？

Portable Requirements (2)

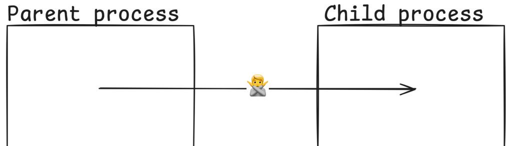
- ✓ Don’t depend on external libraries
  外部ライブラリに依存しない
  - ✓ Including default/bundled gems such as drb
    drb のようなデフォルト/バンドル gem も含む

Multiprocess programming

- ✓ Network communication is required
  ネットワーク通信が必要

How to communicate (1)

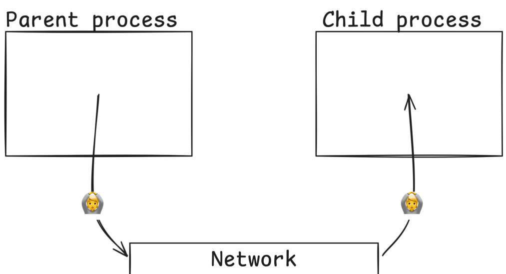
- ✓ Cannot be referenced across processes
  プロセスを超えて参照できない



Powered by Rabbit 4.0.1

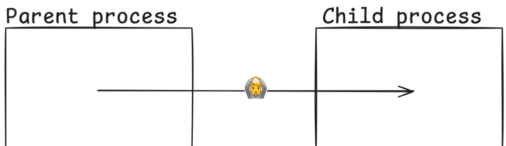
How to communicate (2)

- ✓ Share data over the network
  ネットワークを経由してデータを共有する



drb (as a bundled gem)

✓ Can be referenced like an in-process
同じプロセス内のように参照できる



Lately
最近

- ✓ Standard Library is reduced
  標準ライブラリは減らされて
- ✓ Extract them to default/bundled gems
  デフォルト/バンドル gem に分離されている

Powered by Rabbit 4.0.1

Issues (2)

- ✓ How to design network communication protocol?
  通信プロトコルをどう設計する？

Portable Requirements (3)

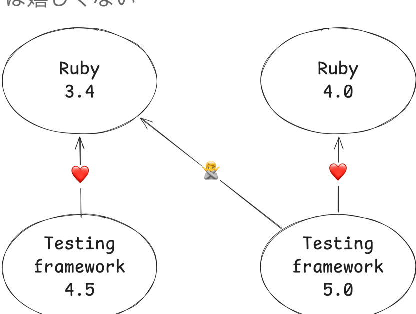
- ✓ Don’t break backward compatibility
  後方互換性を壊さない

Testing framework

- Expected to work on various environments
  色々な環境で動くことが期待されている
  - test-unit of CI: CRuby 2.1 ~ head, JRuby, TruffleRuby

If breaks backward compatibility

- ✓ It's unhappy for users
  ユーザーにとっては嬉しくない



Version up

- Users may need to change their codes and test at once
  ユーザーは自身のコードとテストを同時に変更する必要があるかもしれない
  - For new Ruby and testing framework
    新しい Ruby とテスティングフレームワークに対応するために
- This makes it harder to identify the cause of errors
  エラーの原因を特定することが難しくなる

Issues (3)

- ✓ How to support parallelization with backward compatibility?
  後方互換性を維持しながら、どうやって並列実行をサポートする？

Issues

- ✓ Portable
- ✓ Fast

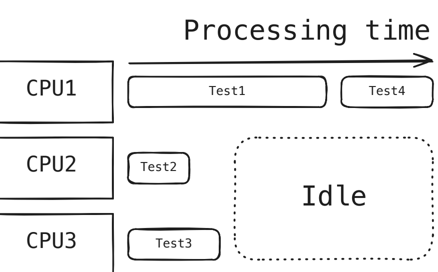
Powered by Rabbit 4.0.1

Fast Requirements

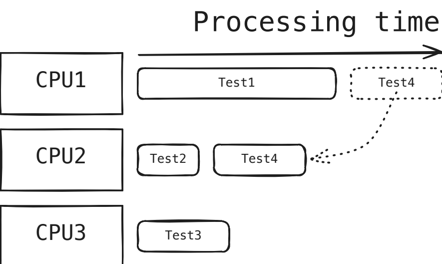
- ✓ **Keep busy** and reduce heavy fixture
  暇をさせずに、重たい準備と後片付けを減らす

Powered by Rabbit 4.0.1

Idle



Keep busy



Fast Requirements

- ✓ Keep busy and **reduce heavy fixture**
  暇をさせずに、**重たい準備と後片付けを減らす**

Fixture

- ✓ Sets up/tears down the state needed to run consistently
  テストを一貫して実行するために必要な準備や後片付けを行う

Test level fixture

- ✓ `#setup`/`#teardown` are called before/after each test
  テスト前後に setup と teardown が呼ばれる

```ruby
class TestA < Test::Unit::TestCase
  def setup; end
  def teardown; end
  def test1; end
  def test2; end
  def test3; end
end
# setup -> test1 -> teardown
# setup -> test2 -> teardown
# setup -> test3 -> teardown
```

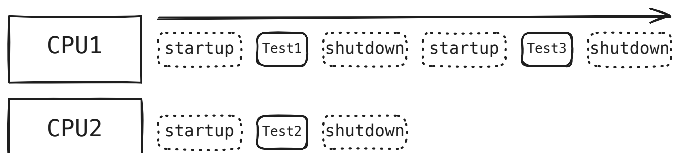
Test case level fixture

- `.startup`/`.shutdown` are called before/after each test case
  テストケース前後に startup と shutdown が呼ばれる
  - Tends to be heavy such as preparing database
    データベースの準備など、処理が重くなりがち

```
class TestA < Test::Unit::TestCase
  class << self
    def startup; end
    def shutdown; end
  end
  def test1; end
  def test2; end
  def test3; end
end
# startup -> test1 -> test2 -> test3 -> shutdown
```

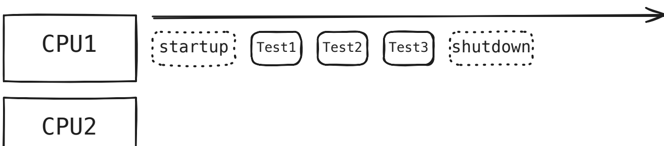
One by one

- ✓ Most overhead, and bad balance



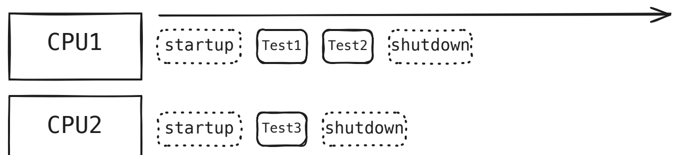
All at once

- ✓ Less overhead, but bad balance



Balance

- ✓ More overhead, but good balance



Issues (4)

- ✓ How to keep busy and reduce heavy fixtures?

  どうやって暇をさせずに、重たい準備や後片付けを減らす？

Issues Summary

課題のまとめ

1. Use `spawn` for Windows: How to restore state?

   spawn は Windows でも動くけどまっさらな環境を作る。状態を復元するには？

2. Don’t depend on drb: Communication protocol?

   drb に依存しない。通信プロトコルはどうする？

3. Keep backward compatibility: How to support parallelism?

   後方互換性を維持しながら並列実行をサポートするには？

4. How to keep busy and reduce heavy fixtures?

Solutions

1. Use spawn for Windows: How to restore state?

spawn は Windows でも動くけどまっさらな環境を作る。状態を復元するには？

test-unit's features

- ✓ **Collect tests** 👈
  テストを集める
- ✓ Run tests
  テストを実行する
- ✓ Report results
  テスト結果をレポートする

Collect tests (1)

- ✓ Example: collects tests under a test directory
  例: test ディレクトリ配下のテストを集める

```
$ tree test
test
└── test-a.rb
```

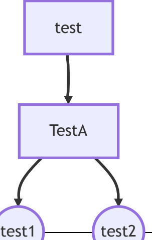
```ruby
class TestA < Test::Unit::TestCase
  def test1; end
  def test2; end
end
```

Collect tests (2)

✓ `test-unit` builds a test suite tree under **given base directory**

test-unit は**指定されたベースディレクトリ**配下からテストスイートのツリーを構築する

```
$ test-unit test
```



Passing to child processes

- Pass **given base directory** to command line arguments in `spawn`
  指定されたベースディレクトリを spawn のコマンドライン引数経由で子プロセスに渡す

Append to $LOAD_PATH

- Users can append directories to $LOAD_PATH
  ユーザーは $LOAD_PATH にディレクトリを追加できる
  - Also pass **given $LOAD_PATH** to command line arguments in spawn
    指定された $LOAD_PATH をspawn のコマンドライン引数経由で子プロセスに渡す

```
$ test-unit test -I lib
or
$ test-unit test --load-path lib
```

Done: Issues (1) 😁

- ✓ A test suite tree can be rebuild in child processes
  テストスイートのツリーが子プロセスで再構築できた
- ✓ Load all required files in each child process
  各子プロセスで必要なファイルをすべてロードしている
  - ✓ Wasteful but likely has minimal impact on total speed
    無駄が多いけど全体の速度にはほとんど影響なさそう
  - ✓ Because collecting tests is not a bottleneck
    テストを集めるのはボトルネックではないから

2. Don’t depend on drb: Communication protocol?

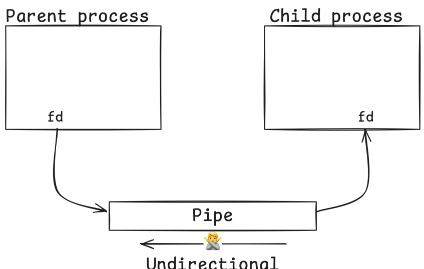
drb に依存しない。通信プロトコルはどうする？

IPC (Inter-Process Communication)

- **Pipe** 👈
  - Portable/Faster/Local
- Unix domain socket
  - Unix only/Fast/Local
- **TCP/IP socket**
  - Portable/Slow/Local and Remote

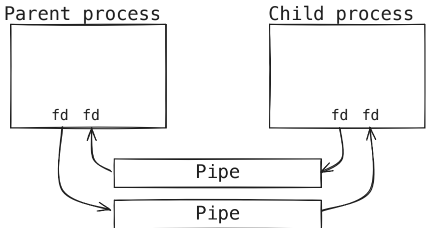
Creates a pipe

- ✓ `IO.pipe`: Creates a pair of pipes
  `IO.pipe` はパイプのペアを作成する
  - ✓ Pipes provide an undirectional IPC
    パイプは一方向の IPC を提供する



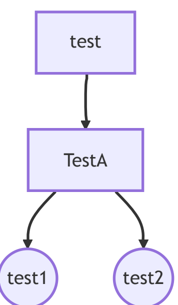
Bidirectional IPC

- ✓ Needs two pipes



Run tests

- ✓ Run tests by traversing a test suite tree
  テストスイートのツリーをたどってテストを実行する



Run a test

- Know path to the leaf of a test suite tree
  テストスイートのツリーの葉までのパスを知っていれば
  - Can find test in a test suite tree and run a test
    テストを探して実行できる
- Path to the leaf
  葉までの経路
  - TestA test1
  - TestA test2

Serializing

- Test name strings
  テスト名の文字列
  - test1(TestA), test2(TestA)
- Test result **object**
  テスト結果のオブジェクト
  - failures, errors, summary
    失敗、エラー、サマリー

Marshalling

- `Marshal.#dump`
  - Can serialize an object
    オブジェクトをシリアライズできる

Unmarshalling

- `Marshal.#load`
  - Can deserialize an object
    オブジェクトをデシリアライズできる
  - We do not need to know the size of data😁
    読み込むデータのバイト数を知る必要がない
  - Blocks process until data is readable
    データが読み込み可能になるまで処理をブロックする

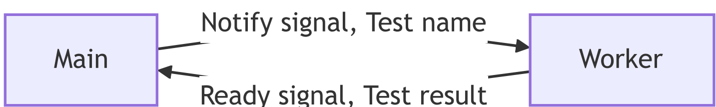
Powered by Rabbit 4.0.1

Role

- A child process named Worker
  - Collects tests
    テストを集める
  - Run tests
    テストを実行する
- A main process named Main
  - Report results
    テスト結果をレポートする

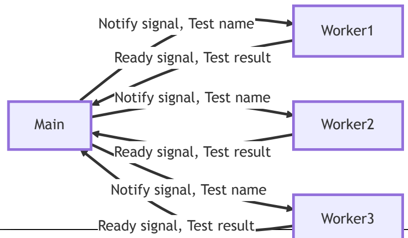
Single worker

- ✓ Also sends notify and ready signal to one another
  実行してー。終了してー。や、準備完了の合図もお互いに送りあう



Multiple workers

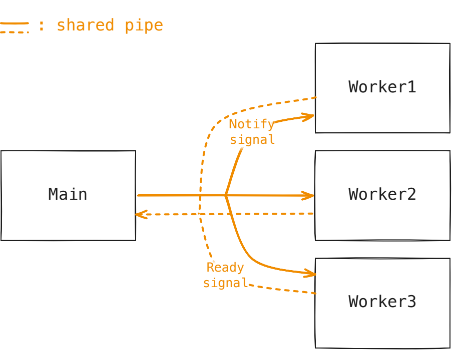
- ✓ Must be exchanged safely between multiple processes
  複数のプロセス間で安全に交換される必要がある
  - ✓ We cannot use `Thread::Queue` because it's not multithread programming
    マルチスレッドプログラミングではないので `Thread::Queue` を使えない



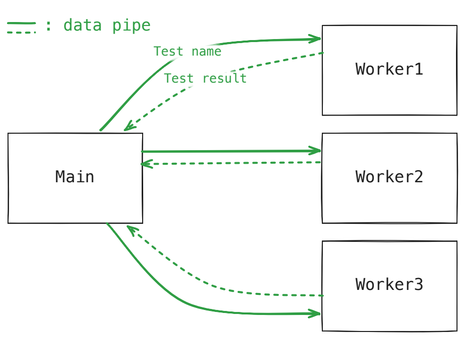
A possible solution

- Share pipes like `Thread::Queue`
  `Thread::Queue` のような共有パイプ
  - Can be exchanged safely if it is one byte writing/reading
    1 バイトの読み書きであれば安全に交換できる
    - **R**un, **F**inish, worker id (~255)
  - OS guarantees atomic operation
    OS がアトミックな操作を保証してくれる

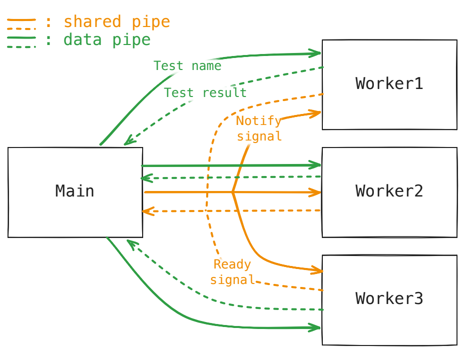
Shared pipes



Data pipes



All pipes



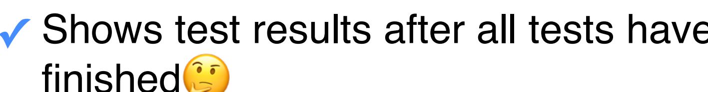
Pipe communication protocol

Main
NotifiedPipe
ReadyPipe
DataPipe
Workers

1 Spawn workers
2 Write `R`, `F`
loop [Repeat until all tests are finished]
3 Read `R`
4 Write a worker id
5 Read a worker id
6 Write a test name
7 Read a test name
8 Find and run a test
9 Read `F`
10 Write result
11 Read result sequentially

Main
NotifiedPipe
ReadyPipe
DataPipe
Workers

Real-time reporting

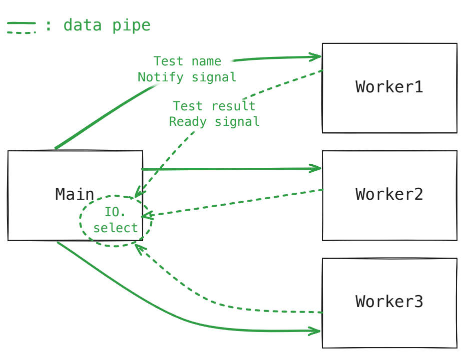


- ✓ Shows test results after all tests have finished🤔
  すべてのテストが終わってからテスト結果を表示する
- ✓ We want to show failures or errors immediately
  テスト実行中、すぐに失敗やエラーを表示したい

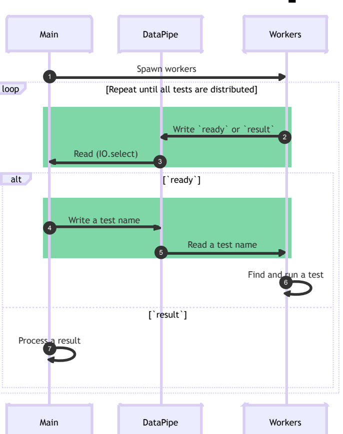
IO.select

- ✓ Allows multiplexing by monitoring multiple IO objects
  複数の IO オブジェクトを監視することで、 I/O の多重化を可能にする
  - ✓ Can remove shared pipes
    共有パイプを削除できる

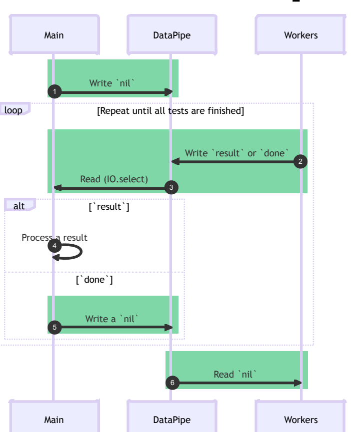
A solution



Pipe communication protocol (1)



Pipe communication protocol (2)



Dumping Issue

- ✓ Cannot dump an object includes `Proc` or `IO`
  `Proc` や `IO` を含むオブジェクトはダンプできない

Test::Unit::TestCase object

- ✓ Should also write to pipe for hook point
  Test::Unit::TestCase オブジェクトもフック用にパイプへ書き込む必要があった
- ✓ Users can set any instance variables in test
  ユーザーはテスト内で任意のインスタンス変数を設定できる

```ruby
class TestA < Test::Unit::TestCase
  def test1
    @input = IO.new(0)
  end
end
```

Customize marshal behavior

- ✓ Hook points for marshaling
  マーシャルのためのフックポイント
  - ✓ `Object#marshal_dump`
  - ✓ `Object#marshal_load`
- ✓ Marshaling only required information
  必須な情報だけをマーシャリングする
  - ✓ Can now perform marshaling 😁
    マーシャリングできるようになった

Cannot dump anonymous class

無名クラスをダンプできない

- Users can be grouping tests using `sub_test_case`
  ユーザーは `sub_test_case` を使ってテストをグループ化できる
  - Like `context` or `describe` in RSpec
- `sub_test_case` creates an anonymous class🤔
  `sub_test_case` は無名クラスを作る

Assign a constant

- ✓ Assign an anonymous class to a constant
  無名クラスを定数に割り当てる
  - ✓ No longer anonymous class 😁
    無名クラスじゃなくなる

Safe constant name

- ✓ `Marshal.#load` cannot restore diffrerent class definitions
  `Marshal.#load` はクラス定義が違うと復元できない
- ✓ Must be a unique constant name
  ユニークな定数名じゃないといけない
  - ✓ Even across processes
    プロセスを超えても同じ
  - ✓ Safe as a constant name
    定数名として安全な名前

Base64 encoding

- ✓ We solved by encoding a sub-test-case name with Base64😁
  Base64 でエンコードすることで解決した
  - ✓ `object_id` is different across processes
    オブジェクト ID は別プロセスだと違う値になる

```
# We can't use "\n", "=", "+" and "/" in base64 as class name.
encoded_name = [sub_test_case.name].pack("m").delete("\n=+/")
const_set(:"TEST_#{encoded_name}", sub_test_case)
```

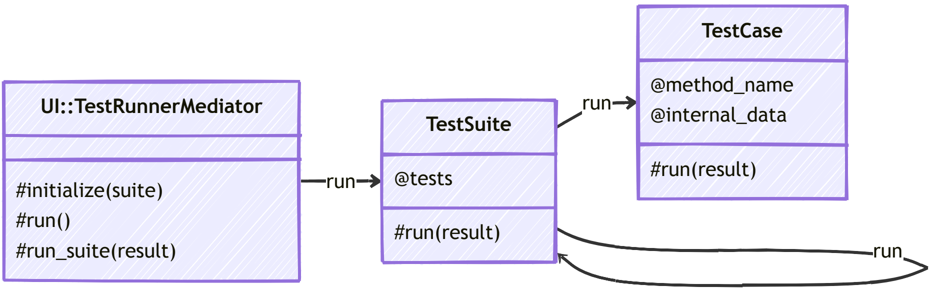
Done:
Issues (2) 😁

3. Keep backward compatibility: How to support parallelism?

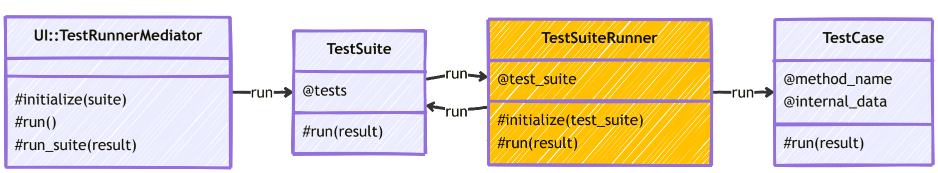
後方互換性を維持しながら並列実行をサポートするには？

Overview of running test suites

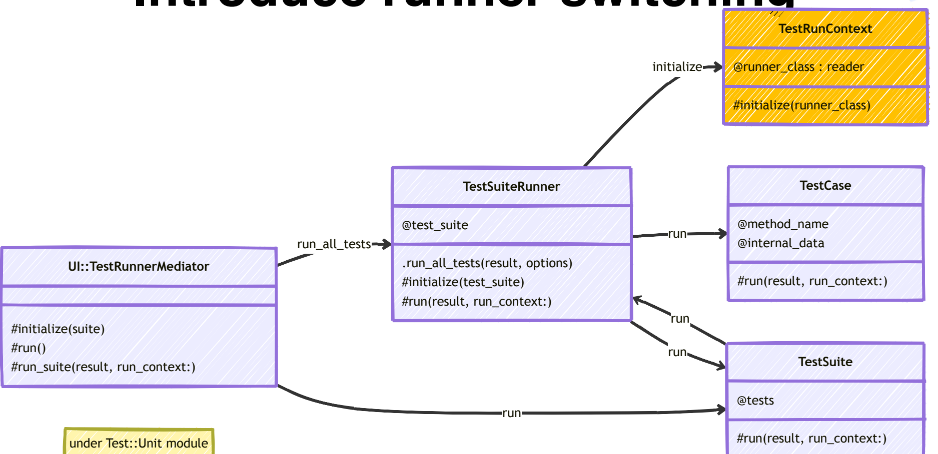
Overview of running test suites (focus)

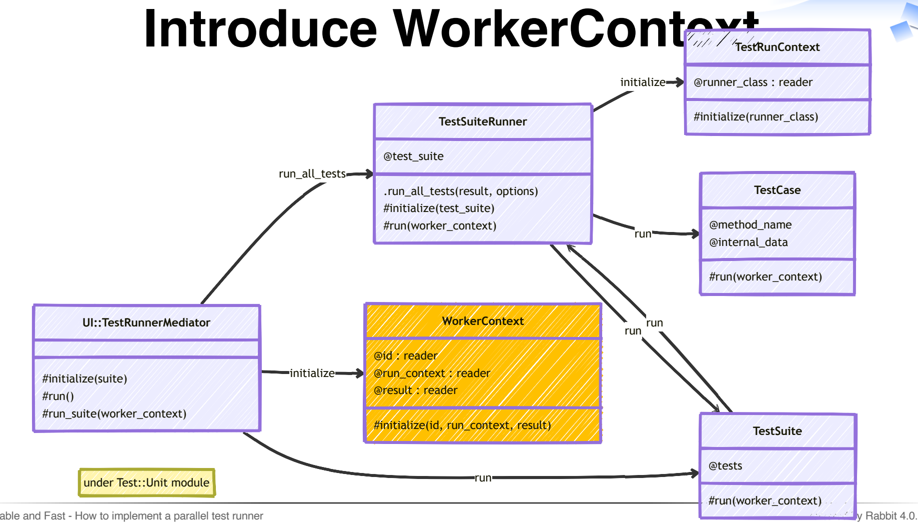


Abstract #run from TestSuite to TestSuiteRunner

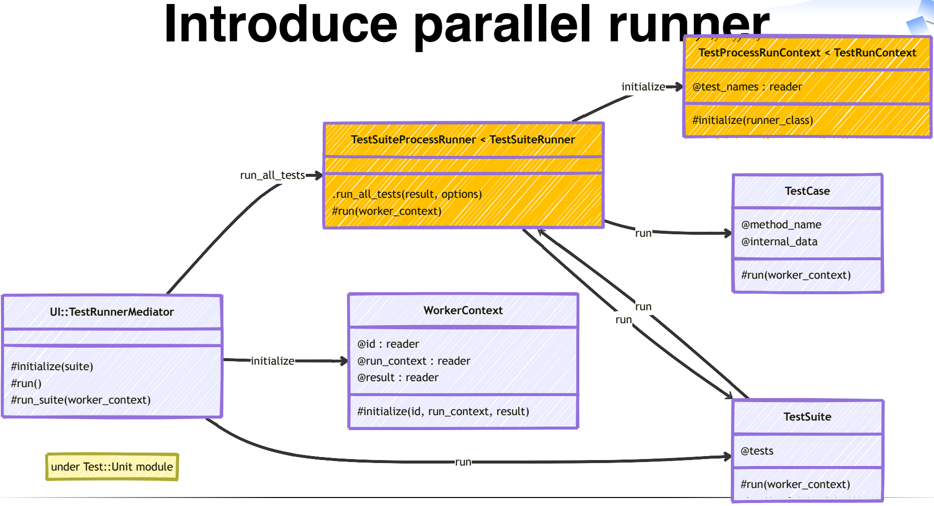


Introduce runner switching




Introduce WorkerContext
TestRunContext
@runner_class : reader
#initialize(runner_class)
initialize
TestSuiteRunner
@test_suite
.run_all_tests(result, options)
#initialize(test_suite)
#run(worker_context)
run_all_tests
TestCase
@method_name
@internal_data
#run(worker_context)
run
UI::TestRunnerMediator
#initialize(suite)
#run()
#run_suite(worker_context)
WorkerContext
@id : reader
@run_context : reader
@result : reader
#initialize(id, run_context, result)
initialize
run
run
TestSuite
@tests
#run(worker_context)
run
under Test::Unit module
Portable and Fast - How to implement a parallel test runner

Introduce parallel runner



Done:
Issues (3) 😁

4. How to keep busy and reduce heavy fixtures?

暇をさせずに、重たい準備や後片付けを減らすには？

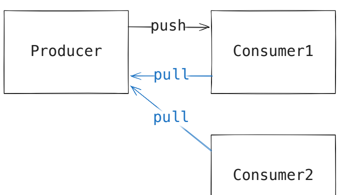
Producer-Consumer model

- Producer: produces tests
  プロデューサーはテストを生成する
  - Main Process (1)
- Consumer: consumes tests
  コンシューマーはテストを消費する
  - Worker Processes (N)

We want consumers to keep busy.
コンシューマーを暇させないようにしたい

Distribute styles

- Push style:
  - Producer pushes tests to consumers
    プロデューサーがコンシューマーにテストをプッシュする
- **Pull style**:
  - Consumers pull a next test from producer
    コンシューマーがプロデューサーから次のテストをプルする



Which leads?

どっちがリードする？

- ✓ Producer does not know consumers are busy
  プロデューサーはコンシューマーが忙しいのか知らない
- ✓ Consumers know own is busy
  コンシューマーは自身が忙しいのか知っている
  - ✓ So start with consumers and then keep them busy
    なので、コンシューマーが起点となって暇かどうかを教えてくれれば暇をさせにくくできる

We use **pull style.**

我々は**プルスタイル**を使う

Balance variable workloads

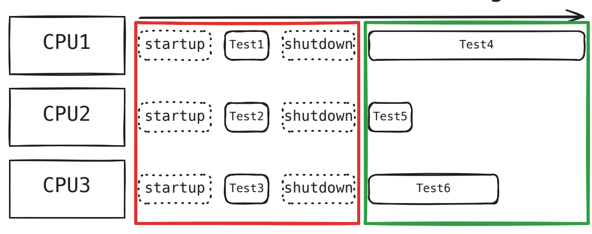
負荷を均す

- ✓ Reduce heavy fixtures as much as possible
  重たい準備と後片付けはできるだけ減らしたい
  - ✓ Heavy fixtures are test case level fixtures
    重たい準備と後片付けは、テスケースレベルのフィクスチャーのことでしたね
- ✓ However, we want to assign tests to another idle consumer
  でも別のコンシューマーが暇しているときは、そちらにテストを割り振りたい

Pass a test one by one

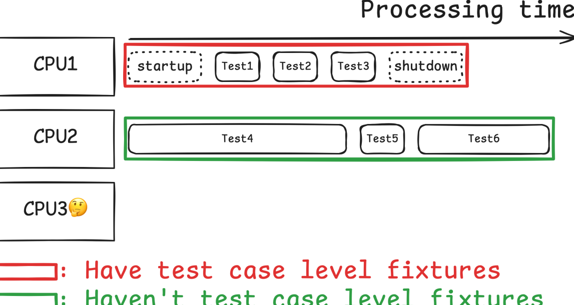
テストを一つずつ渡す

- ✓ Test case level fixtures are called multiple times
  テストケースレベルのフィクスチャーは複数回呼び出される



A possible solution

✓ Pass each test case level
テストケースレベルで渡す



Solution

✓ Pass each test case level **if test case level fixtures exist**

テストケースレベルのフィクスチャーがあるときだけ、テストケースレベルで渡す

Processing time

CPU1: startup, Test1, Test2, Test3, shutdown

CPU2: Test4

CPU3: Test5, Test6

(red box): Have test case level fixtures

(green box): Haven't test case level fixtures

Done: Issues (4) 😁

Works fine!!!

- ✓ But on CI of Windows
  しかし、Windows の CI で
  - ✓ Occurred `'Kernel#spawn': wrong file descriptor` error
    ファイルディスクリプターのエラーが発生した

File descriptors

- 0, 1 and 2 are used by default
  - 0: Standard Input
  - 1: Standard Output
  - 2: Standard Error Output
- Pass 3 and 4 for data pipes to child processes

  データ用パイプのために 3, 4 を子プロセスに渡してる

File descriptors on Windows

- ✓ Cannot pass 3 and above to child processes
  3 以上を子プロセスに渡せない
- ✓ Avoid changing stdin/stdout/stderr as much as possible
  標準入力/標準出力/標準エラー出力を変更するのはできるだけしたくない
  - ✓ Because tests may use them
    テストがそれらを使うかもしれないから

We solved by using TCP/IP socket instead of pipe **if Windows**

Windows の場合、パイプの代わりに TCP/IP のソケットを使うことで解決した

Solutions Summary

解決策のまとめ

1. State? Rebuild a test suite tree
   状態は？テストスイートのツリーを再構築する
2. Communication? Use pipe, `Marshal`, `IO.select`
   通信は？パイプ、`Marshal`、`IO.select`を使う
3. Backward compatibility? Abstract runner and switch it
   後方互換性は？実行するところを抽象化して切り替える
4. Fast? Use pull style. Pass a test suite if test case level fixtures exist
   速さは？Pullスタイルを使う。テストケースレベルのフィクスチャーがあるときはまとめて渡す

Demo

- ✓ `--parallel=process` option is available
  `--parallel=process` オプションでマルチプロセスベースの並列実行を有効にできる
  - ✓ `--n-workers=N` option is available (default: the number of available processors)
    `--n-workers=N` オプションで並列実行するワーカーの数を指定できる
- ✓ Parallel running output looks like a regular test-unit
  並列実行の出力は、通常の test-unit と同様
- ✓ Show the test result in real time
  テスト結果をリアルタイムに表示する

Future

Ractor runner

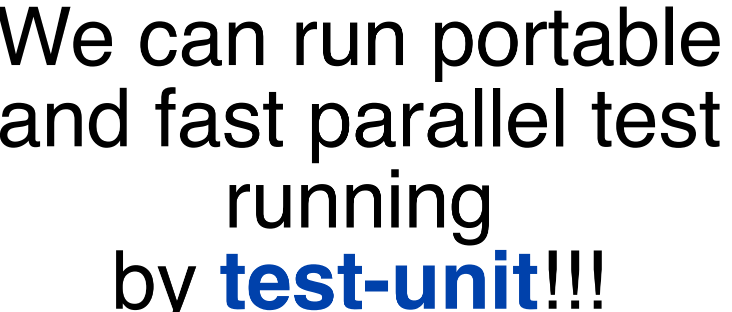
- Now implementing!!!
  - State: Already loaded, but object sharing is limited
    状態: 既にロード済みだが、オブジェクトの共有は制限されている
  - Communication: Use `Ractor::Port`
    通信: `Ractor::Port` を使う
  - Backword compatibility: Switching backend
    後方互換性: バックエンドを切り替える
  - Pull style: `Ractor.receive` not `Ractor.select`
    Pull スタイル: `Ractor.select` ではなく `Ractor.receive` を使う

Ractor::IsolationError

- We met a lot of `Ractor::IsolationError`
  多くの Ractor::IsolationError が発生した
  - Allow reading shareable class variables from non-main Ractors
    非メイン Ractor から共有可能なクラス変数を読み取ることを許可する
  - https://bugs.ruby-lang.org/issues/21942
- Ractor ❤️ Ruby::Box ?
  Ractor と Ruby::Box は相性が良い？

Conclusion

まとめ



test-unit を使うと、ポータブルで速い並列テストを実行できる!!!

Acknowledgement

- Sutou Kouhei
  - Thank you for launching the Red Data Tools
    Red Data Tools を立ち上げてくれてありがとう
  - Thank you for always polite explaining
    いつも丁寧に説明してくれありがとう
  - I wouldn’t be standing here today without him
    彼なしではこの場に立つことはできなかった
- Naoto Ono
  - Thank you for working with us
    一緒に開発してくれてありがとう
  - Your honest feedback are always helpful
    率直な意見にいつも助けられています

Powered by Rabbit 4.0.1

Join us!

Red Data Tools