Rinda, dRuby

answering the RubyConf, RubyKaigi

Masatoshi SEKI

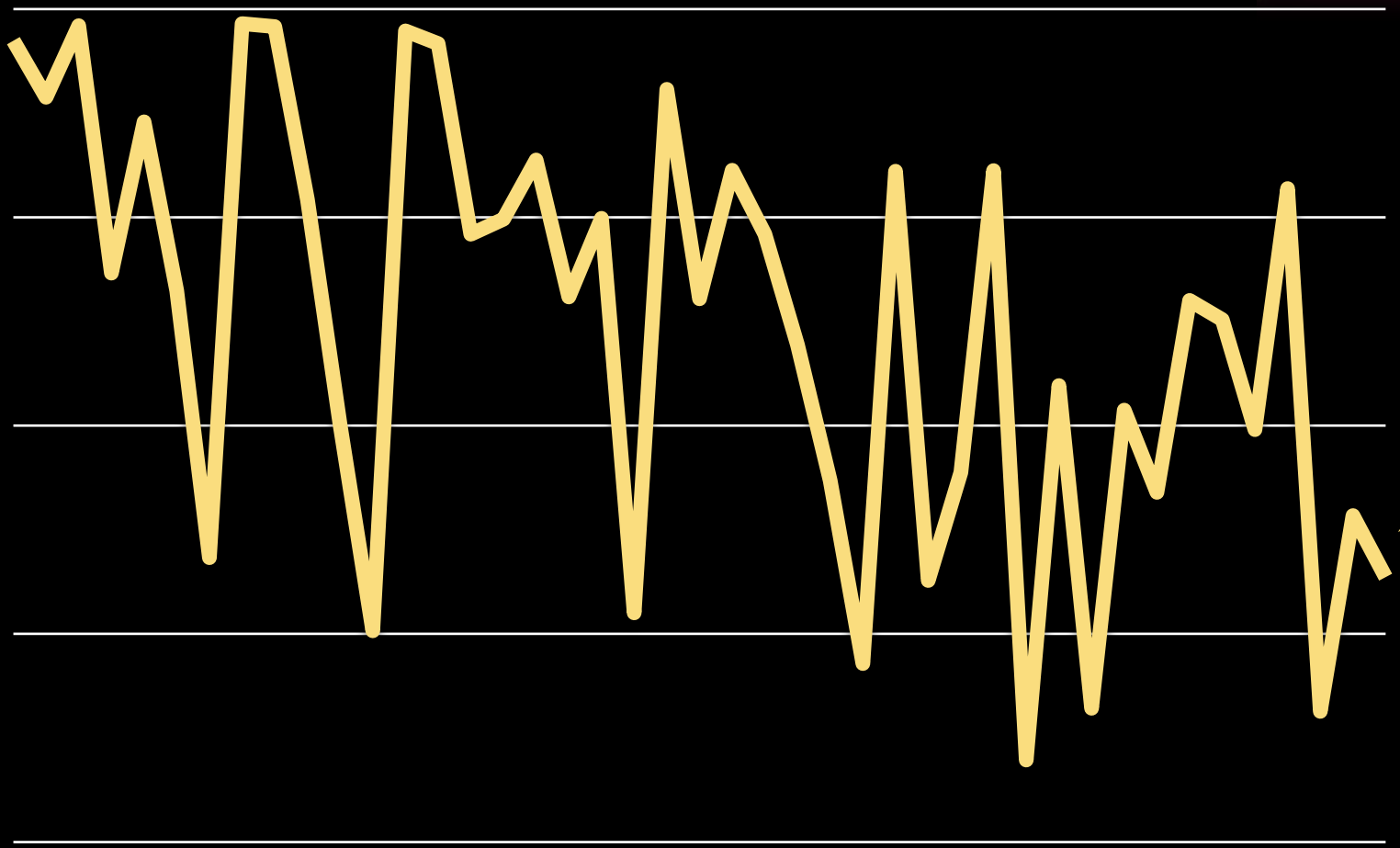
重要なことを先に

- 先に

重要なこと

- '05 夏リリース
- まだ初版買えます

amazon.co.jp



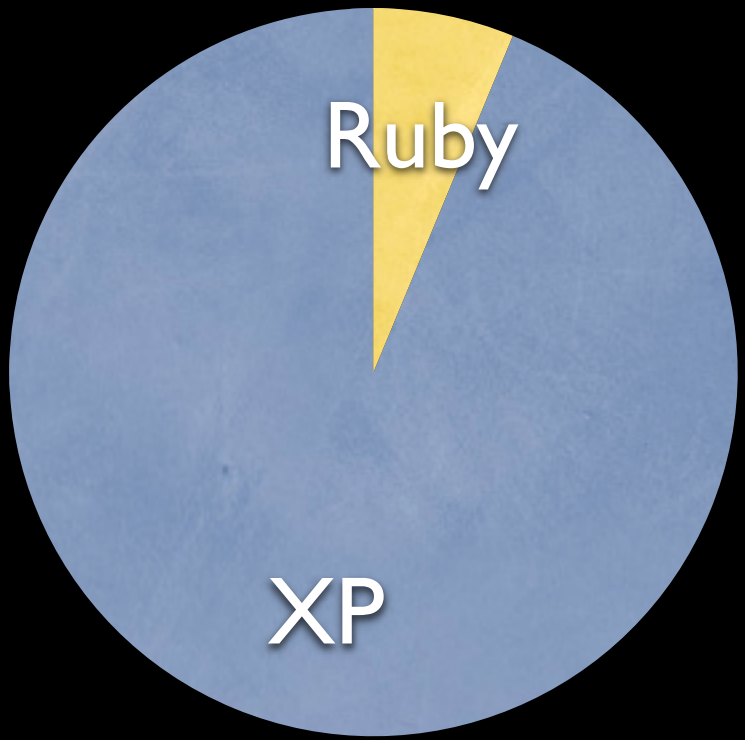
buy now!

私について

- 自称プログラマ
  - 今日はAgileの人じゃないよ
- 打たれ弱い

私について

- Rubyの人になりたい
  - さっぱりRubyな依頼こないよ？

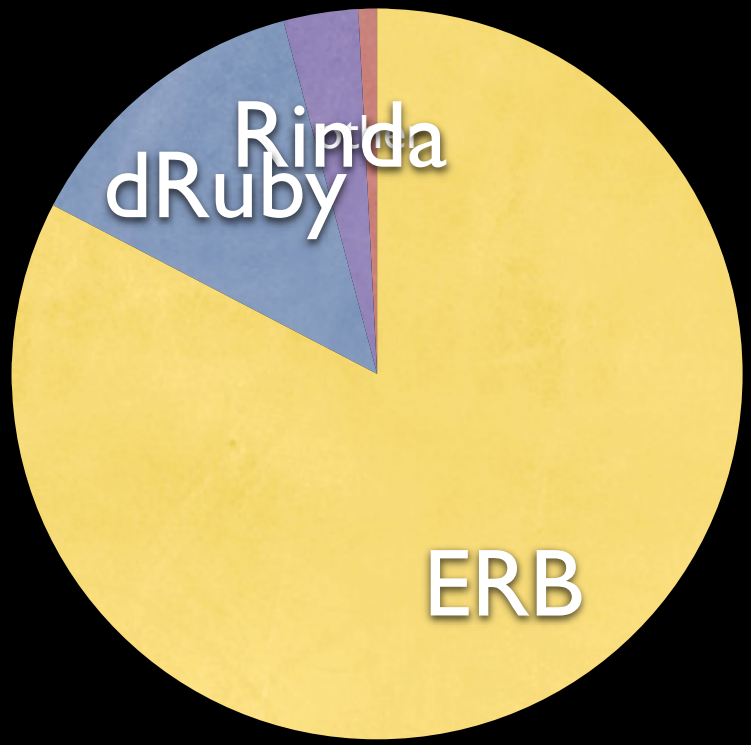


products

- ERB
- dRuby
- Rinda
- RWiki
- Div
- Koya
- ...

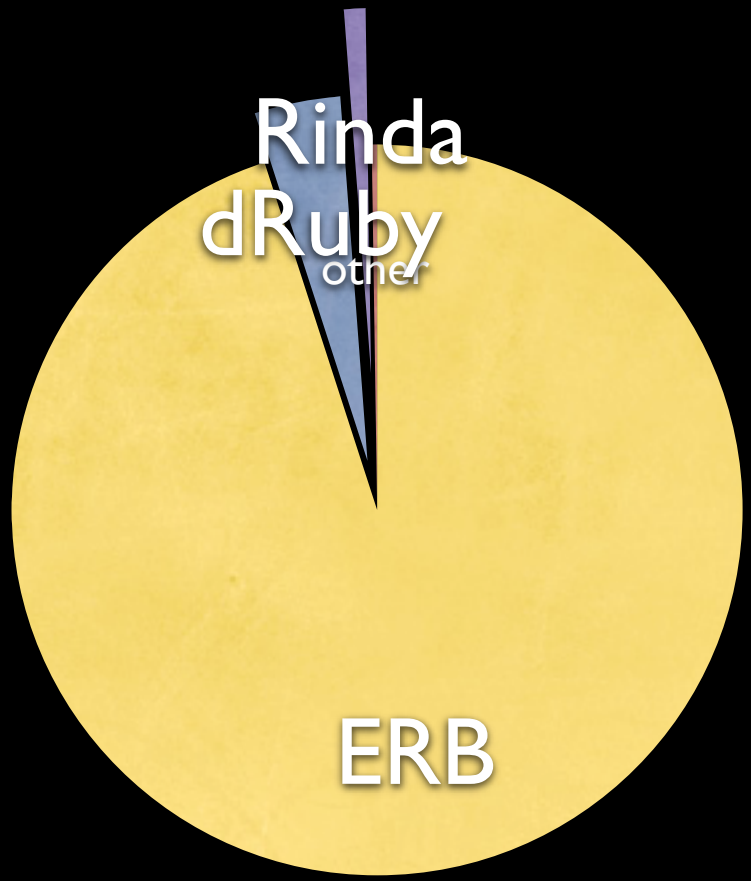
users

- ERB
- dRuby
- Rinda
- RWiki
- Div
- Koya
- ...



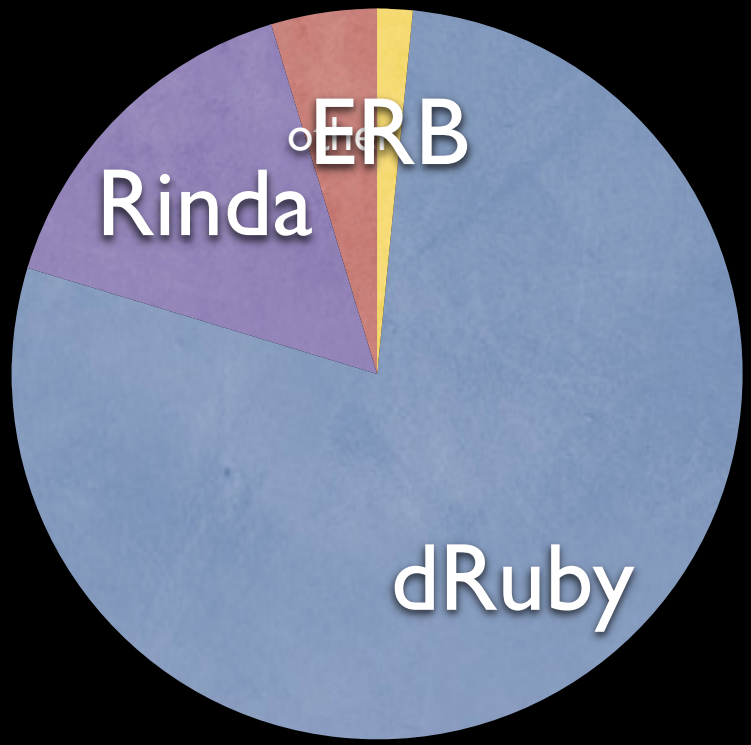
users

- ERB
- dRuby
- Rinda
- RWiki
- Div
- Koya
- ...

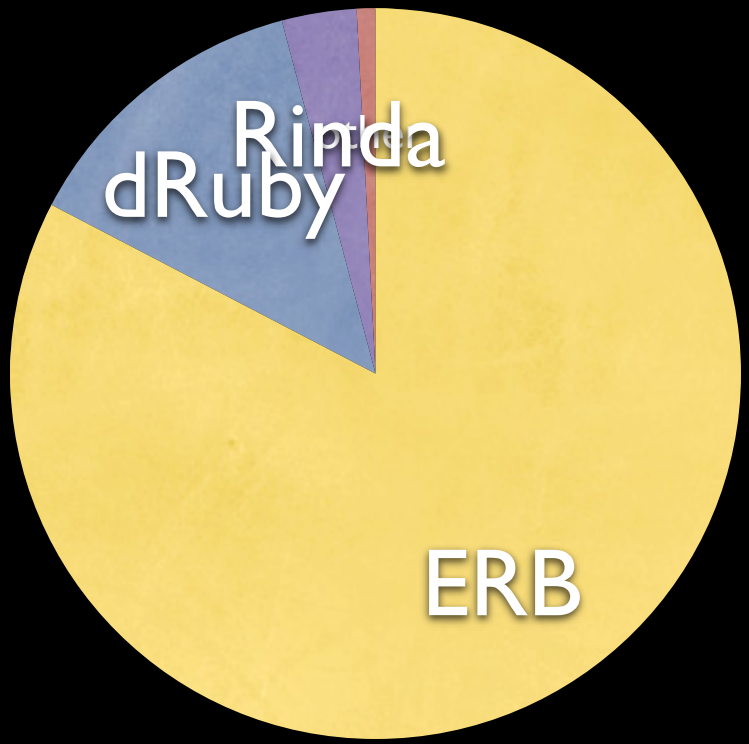


ego-search

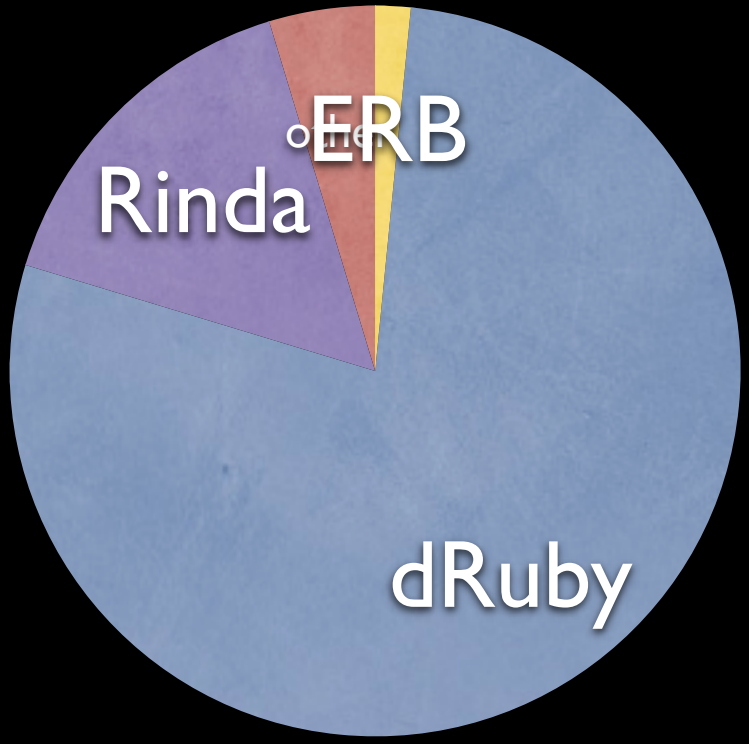
- ERB
- dRuby
- Rinda
- RWiki
- Div
- Koya
- ...



報われない愛



ユーザ数



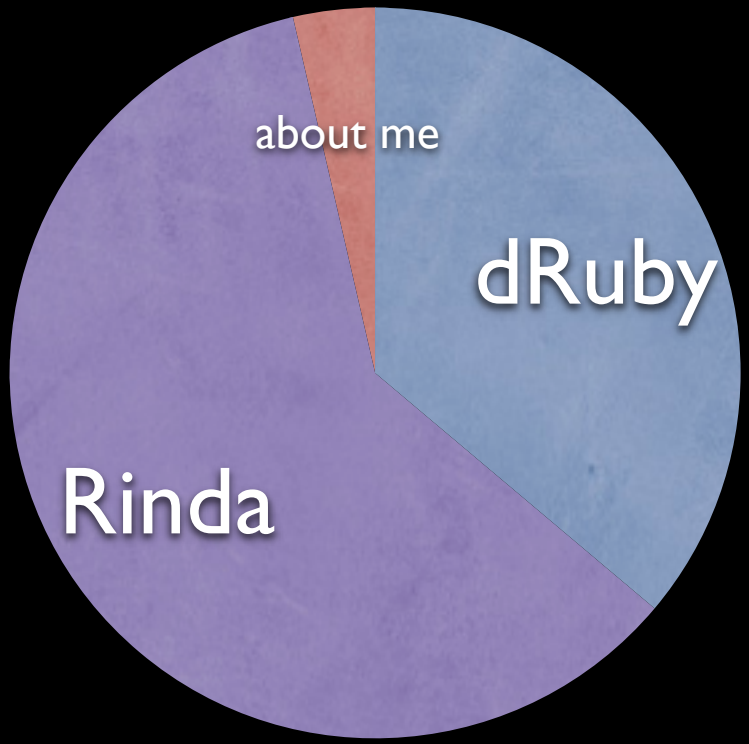
作者の愛情

愛の反対は無関心

- ERBを愛してないわけじゃない

でも今日は

- Rindaを中心に
- ERBはまた今度



agenda

- dRuby, Rindaの紹介
- いくつかの発表に言い訳
  - RubyKaigi 2006
  - Rinda and DRb in the Real World.
  - Scaling Twitter

言い訳駆動

- 最初のリリース以降のモチベーション
- コンセプト製品みたいのばっかり
- そのくせ指摘されると打たれ弱い
- 言い訳をするように本を書いたり..

RubyKaigi 2006

dRuby, Again.

dRubyをもう一度思い出してみよう
今日も古い話ばっかり

dRuby

dRubyのやること

- Rubyのメッセージングを拡張
- 別のプロセスのオブジェクトのメソッドを起動する

ブロック付きメソッド

- なぜ動くのか
  - Procは参照渡し
  - yieldはProcへの操作

Rubyなので

- IDL不要
- proxyは一種類だけ

で?

- dRubyをなにに使うの?

いろんな分散

- 配置
  - プロセス
  - マシン
- 機能
- ちょっと豪華な通信の代替手段

寿命といえば永続化

- 永遠の永続化
- 比較的に永続化
  - 一方よりは長生きだ

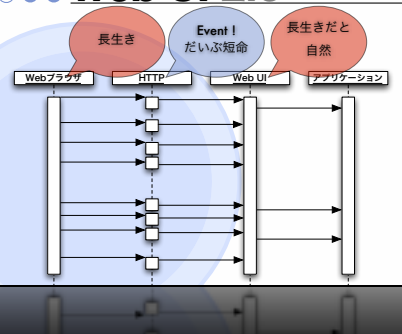
長命なオブジェクト

- 長生きするオブジェクトの方が自然なとき
- Web UIとか

遺言に苦労するなら

- はじめから長命なオブジェクトはいかが?
- つーかそのMVCでいいの?

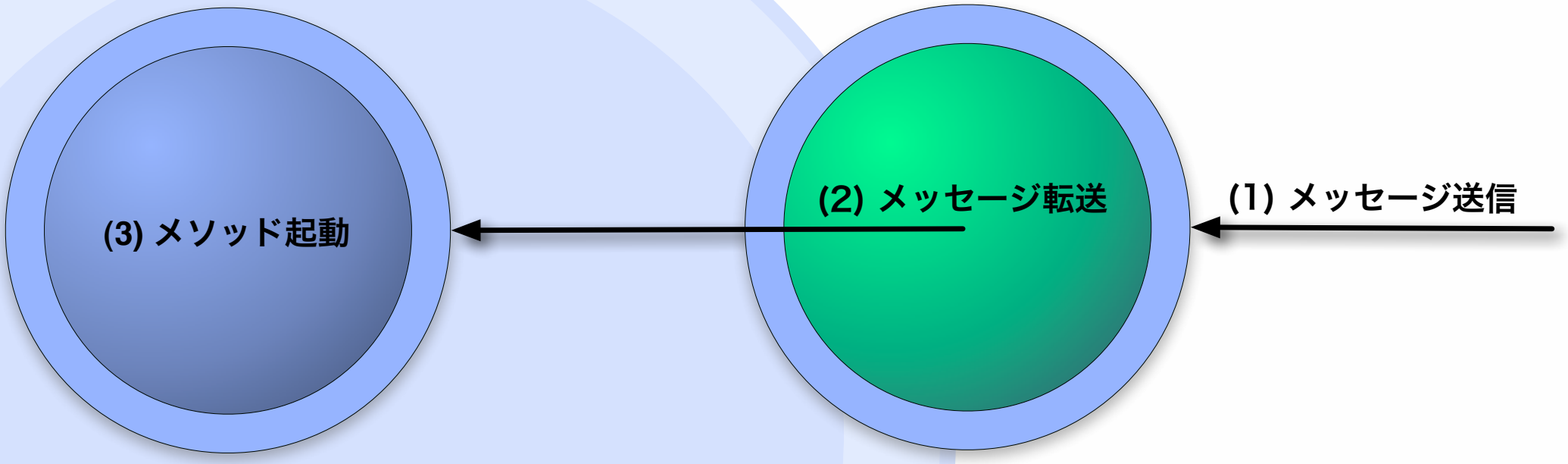
Web UI 2.0



dRubyのやること

- Rubyのメッセージングを拡張
- 別のプロセスのオブジェクトのメソッドを起動する
- オブジェクトの交換

プロセスを越える



実験

- 二つのirbを使って話し合う
- 一方はHashを公開

front

- システムへの入り口、きっかけ
- URIで指定する
- dRubyのシステムに一つ以上

irb

OSXで
Threadがスイッチし
ないときに

```
cinq:~ mas$ irb --simple-prompt --noreadline
>> require 'drb/drb'
=> true
>>
```

```
cinq:~ mas$ irb --simple-prompt --noreadline
>> require 'drb/drb'
=> true
>>
```

{}をfrontに


```
>> DRb.start_service('druby://localhost:12345', {})
=> #<DRb::DRbServer:0x1c1e48....>
>>
```

frontを参照

URIを参照する

```
>> DRb.start_service
=> #<DRb::DRbServer:0x1c97c4...>
>> ro = DRbObject.new_with_uri('druby://localhost:
12345')
=> #<DRb::DRbObject:....>
>>
```

{}をいじる

```
>> ro['client'] = 'Hello, Again'
=> "Hello, Again"
>>
```

```
>> DRb.front
=> {"client"=>"Hello, Again"}
>> DRb.front['server'] = 'Hello, World.'
=> "Hello, World."
>>
```

```
>> ro['server']
=> "Hello, World."
>> ro
=> #<DRb::DRbObject:...">
>> ro.keys
=> ["client", "server"]
>> ro.values
=> ["Hello, Again", "Hello, World."]
>>
```

オブジェクトの交換

Marshal	
dumpable Number, Boolean, String	clone
can’t dump Proc, Thread, File	reference

Threadは参照渡し

```
>> DRb.front['thread'] = Thread.current
=> #<Thread:0x1df7cc run>
>>
```

```
>> ro['thread']
=> #<DRb::DRbObject:...>
>> ro['thread'].status
=> "sleep"
>>
```

値渡しはコピー

```
>> ro['client']
=> "Hello, Again"
>> ro['client'].reverse!
=> "niagA ,olleH"
>> ro['client']
=> "Hello, Again"
>>
```

```
>> DRb.front['client']
=> "Hello, Again"
>>
```

同期

もちろん
TupleSpaceも

- Thread同期の仕組みもそのまま

で?

- dRubyをなにに使うの?

いろんな分散

- 配置
  - プロセス
  - マシン
- 機能
- ちょっと豪華な通信の代替手段

配置と寿命

- オブジェクトはプロセスに在る
- 住いが異なる = 寿命が異なる
- プロセスより長命なオブジェクト

寿命といえば永続化

- 永遠の永続化
- 比較的に永続化
  - 一方よりは長生きだ

長命なオブジェクト

- 長生きするオブジェクトの方が自然なとき
- Web UIとか

Rinda

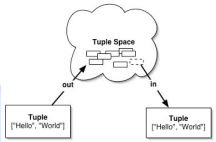
もっとかっこいいやつ

- 並列処理糊言語 LindaとRinda
- 動的な構成変更 Ring
- 使わなくても素振りしておけ

Linda

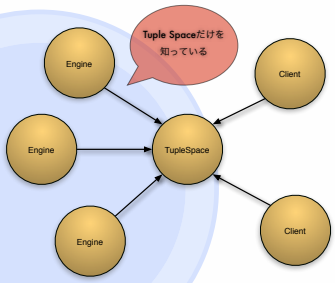
- 並列処理糊言語
- タプルとタプルスペース
- パターンマッチング

Lindaの協調



- out
- inとrd
  - パターンで指定
  - ブロックできる
- 参加者はタプルスペースだけを知ってる

かっこいい並列処理



Rinda

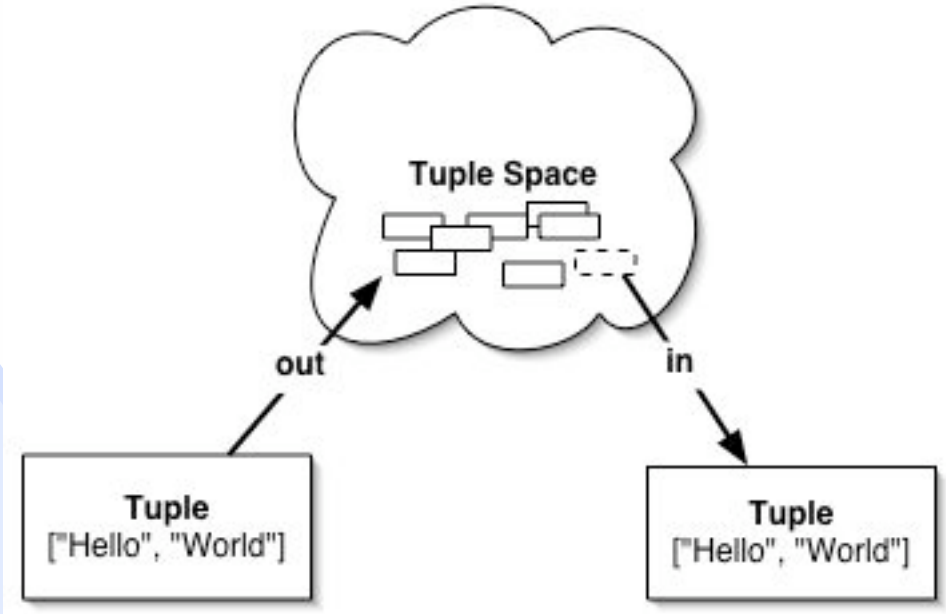
- RubyによるLindaの実装
- dRubyを意識してくれる
- TupleはArray|Hashで

Linda

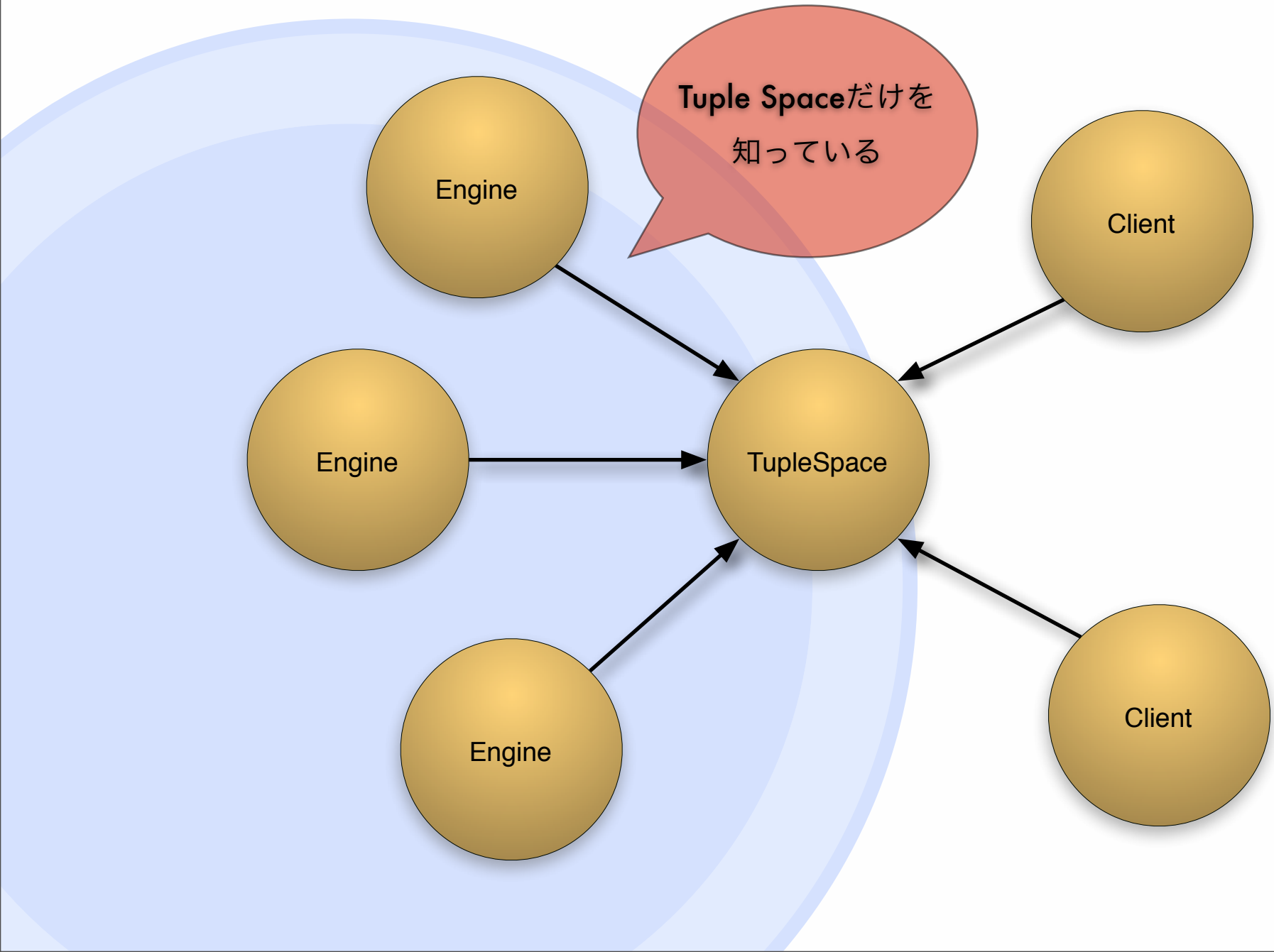
- 並列処理糊言語
- タプルとタプルスペース
- パターンマッチング

Lindaの協調

- out
- inとrd
  - パターンで指定
  - ブロックできる
- 参加者はタプルスペースだけを知ってる



かっこいい並列処理


Tuple Spaceだけを
知っている
Engine
Engine
Engine
TupleSpace
Client
Client

Rinda

- RubyによるLindaの実装
- dRubyを意識してくれる
- TupleはArray|Hashで

TupleSpace

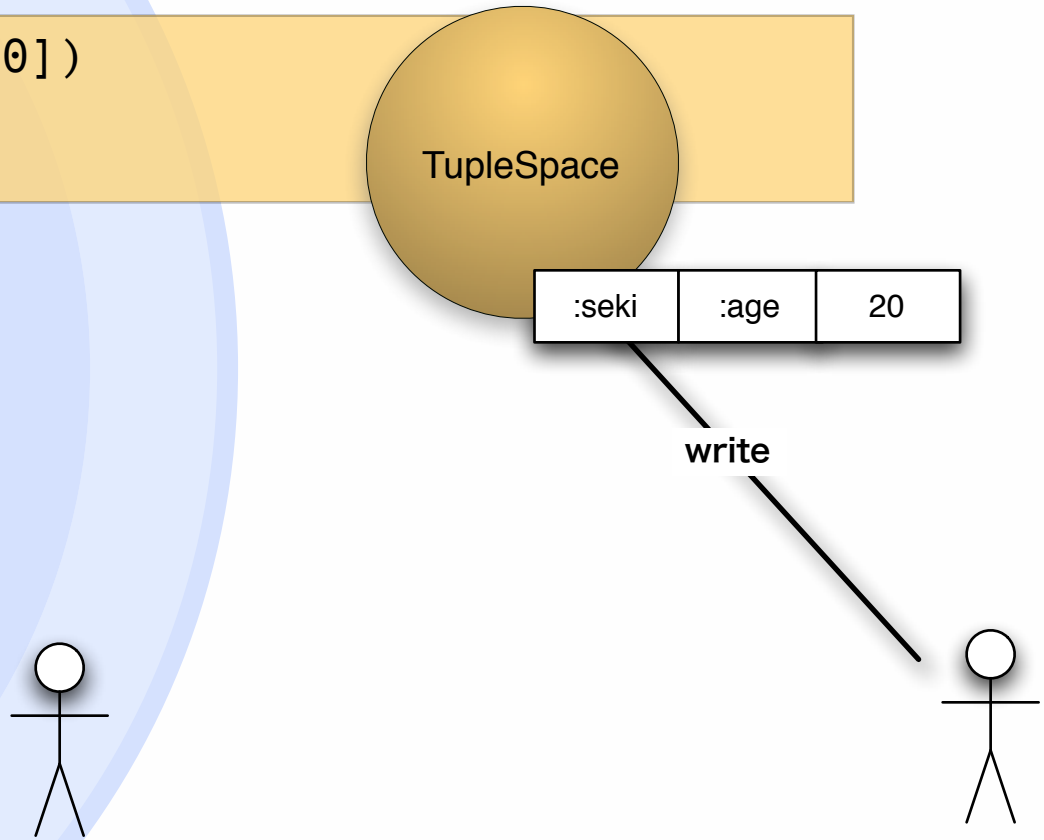
```
>> ts = Rinda::TupleSpace.new
=> #<Rinda::TupleSpace:...>
>> DRb.front['ts'] = ts
=> #<Rinda::TupleSpace:...>
>>
```

```
>> it = DRbObject.new_with_uri('druby://localhost:12345?
ts')
=> #<DRb::DRbObject:...>
>> ts = Rinda::TupleSpaceProxy.new(it)
=> #<Rinda::TupleSpaceProxy:...>
>>
```

TupleSpaceProxyは
dRuby下で安全に動作
させるためのもの

writeとtake

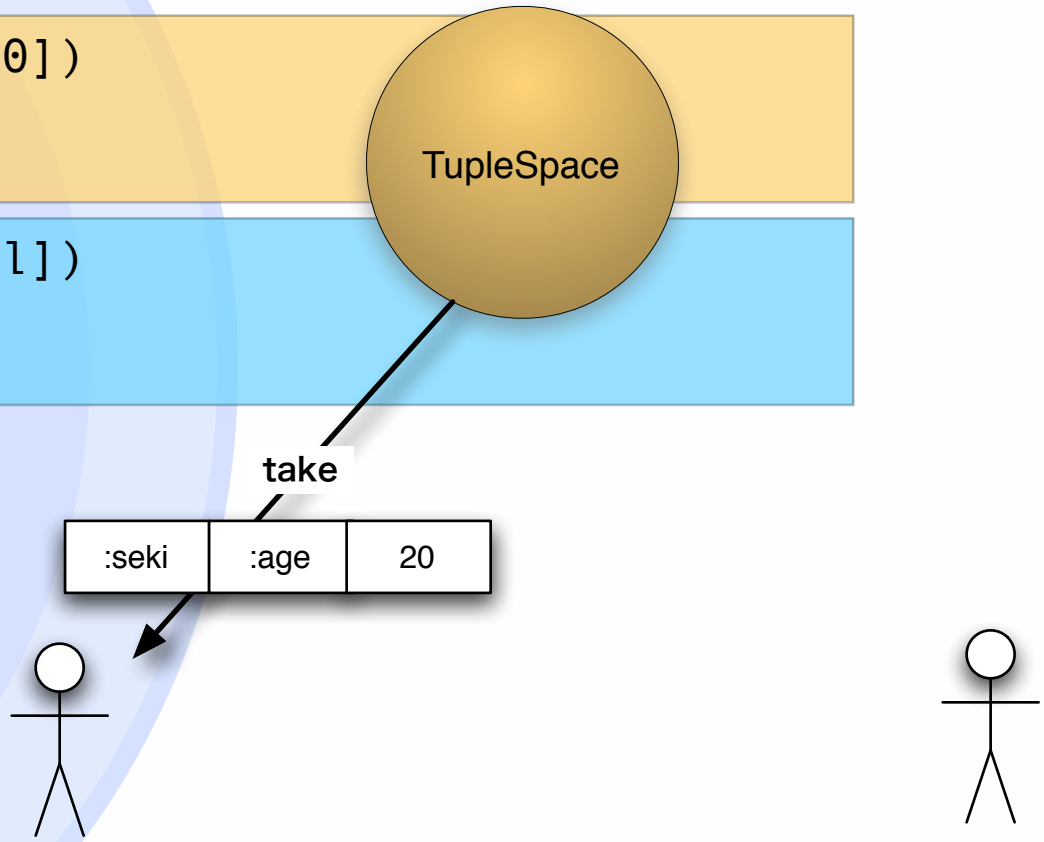
```
>> ts.write([:seki, :age, 20])
=> #<Rinda::TupleEntry:...>
>>
```



writeとtake

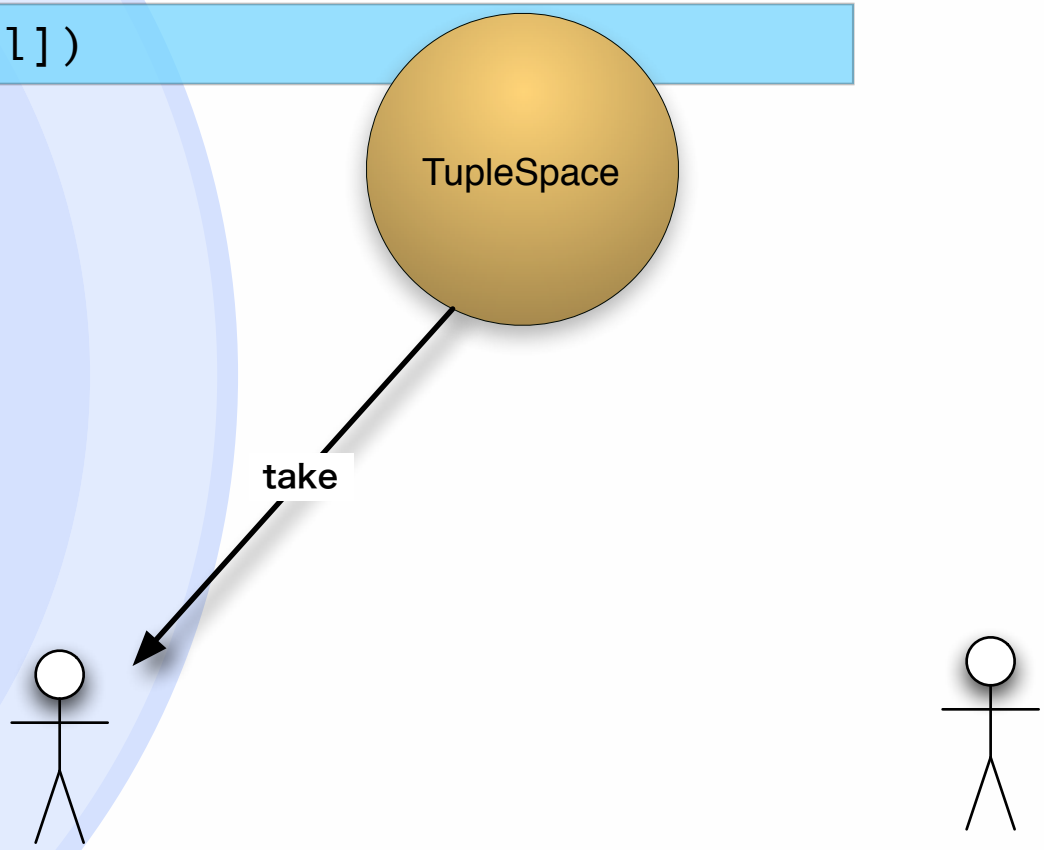
```
>> ts.write([:seki, :age, 20])
=> #<Rinda::TupleEntry:...>
>>
```

```
>> ts.take([:seki, :age, nil])
=> [:seki, :age, 20]
>>
```



待ち合わせ

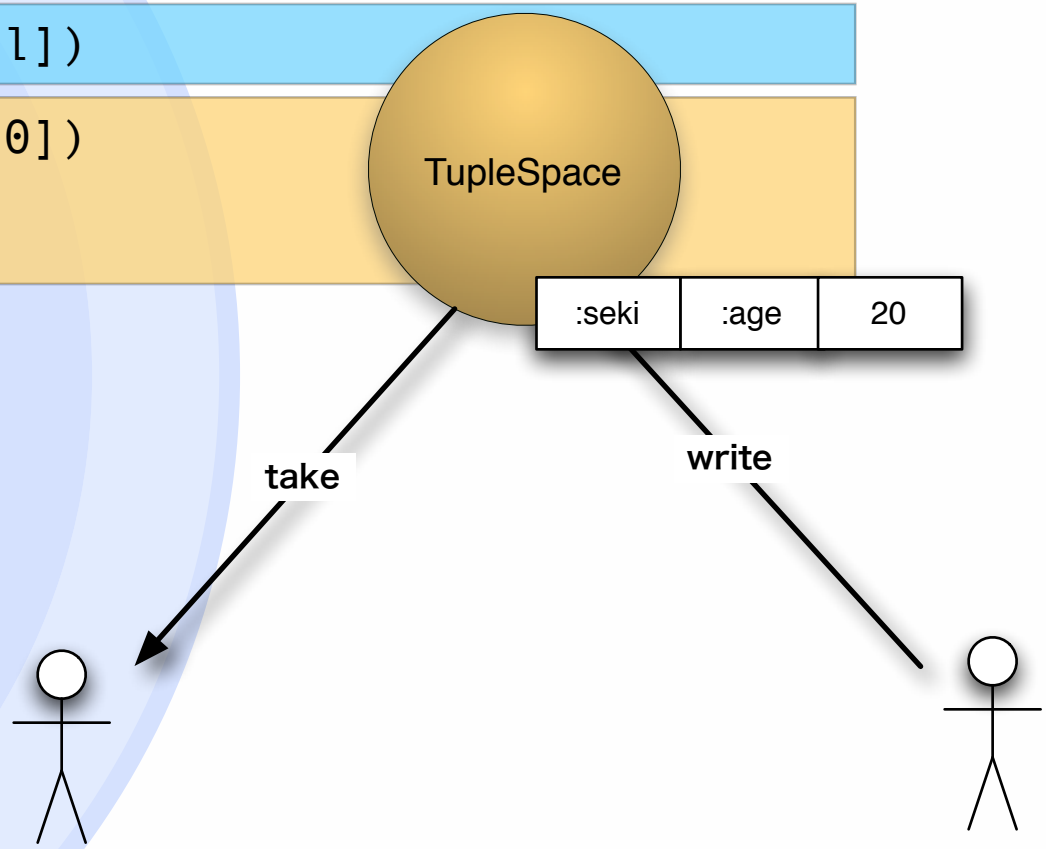
```
>> ts.take([:seki, :age, nil])
```



待ち合わせ

```
>> ts.take([:seki, :age, nil])
```

```
>> ts.write([:seki, :age, 20])
=> #<Rinda::TupleEntry:...>
>>
```

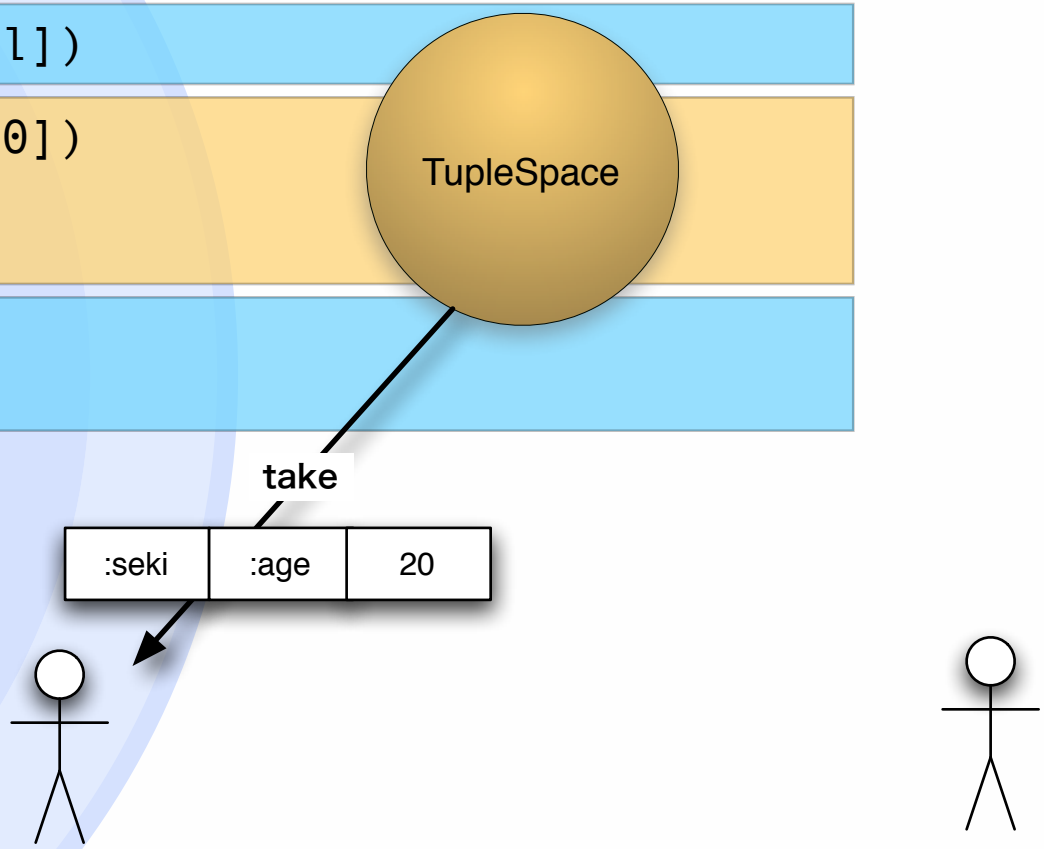


待ち合わせ

```
>> ts.take([:seki, :age, nil])
```

```
>> ts.write([:seki, :age, 20])
=> #<Rinda::TupleEntry:...>
>>
```

```
=> [:seki, :age, 20]
>>
```



Answering

- dRuby, Again.
- Scaling Twitter
- Rinda and DRb in the Real World
  - mput.dip.jp
  - Nick Sieger's blog
    - kiwamu訳

Rinda and DRb in the Real World

- Glenn Vanderburg 返事もらえなかった..
- mputさんやkiwamuさんの日記を参照

Rinda and DRb in the Real World

- Tupleの設計はプロトコル設計
- SPOFだよ
- RingServerは鬼門
- 不揮発性のオブジェクトID
- 2004年から開発停まってるじゃん

Rinda and DRb in the Real World

- Tupleの設計はプロトコル設計
- SPOFだよ
- RingServerは鬼門
- 不揮発性のオブジェクトID
- 2004年から開発停まってるじゃん

Tupleの設計

- Tupleの設計はプロトコルの設計
  - 伝票
- RDBMSのテーブルの設計は?
  - なんだろ。帳簿??

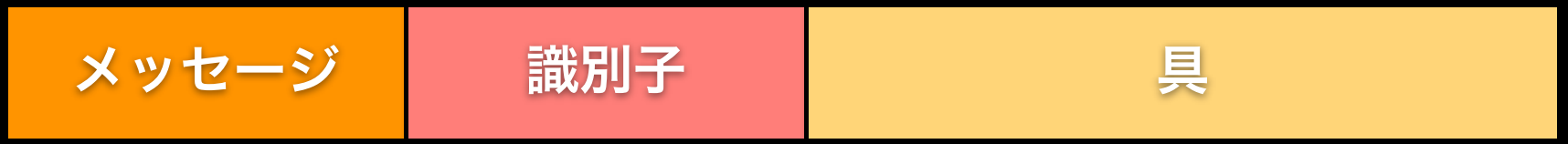
Tupleで表現するもの

- RPCの要求とその返信
- 分散データ構造
  - ストリーム
  - キャッシュ
  - ...

テンプレート

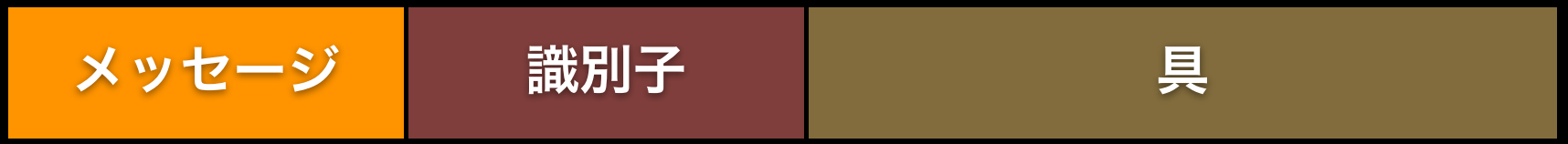
- Tupleをパターンで指定
  - read, takeで
- ===で比較
- nilはワイルドカード

RPCなTuple



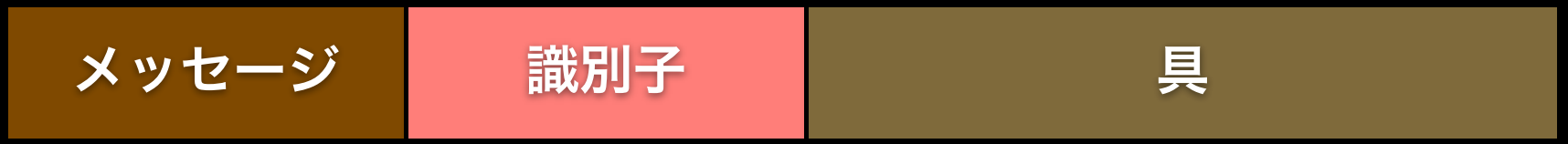
- メッセージ
- 識別子
- 具 / データ

RPCなTuple



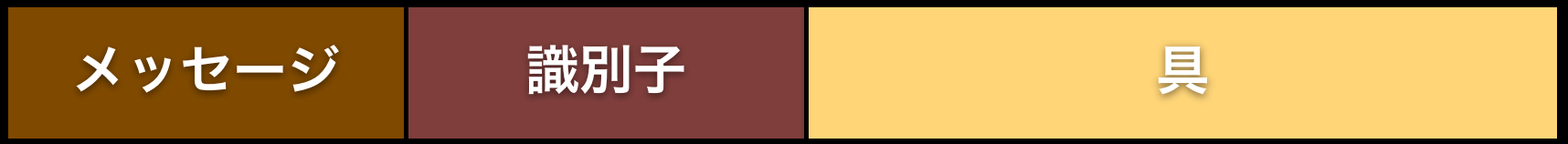
- 値で指定
- ワイルドカードはまずない

RPCなTuple



- ほとんどの場合nil
- 返信待ちでは値の場合も
- ときどき範囲 (Class, Regexp, Range)

RPCなTuple



- ほとんどの場合nil
- 要素数は合わせてね

RPCなTuple

メッセージ	識別子	具
:snapshot	12345	“http://foo/bar”, 320, 240

RPCなTuple

メッセージ	識別子	具
:snapshot	12345	“http://foo/bar”, 320, 240
:snapshot	nil	nil, nil, nil

RPCなTuple

メッセージ	識別子	具
:snapshot	12345	“http://foo/bar”, 320, 240
:snapshot	nil	nil, nil, nil
:snapshot_done	12345	“img_320_240_12345.jpg”

RPCなTuple

メッセージ	識別子	具
:snapshot	12345	“http://foo/bar”, 320, 240
:snapshot	nil	nil, nil, nil
:snapshot_done	12345	“img_320_240_12345.jpg”
:snapshot_done	12345	nil

Scaling Twitter

Big Bird.

(scaling twitter)

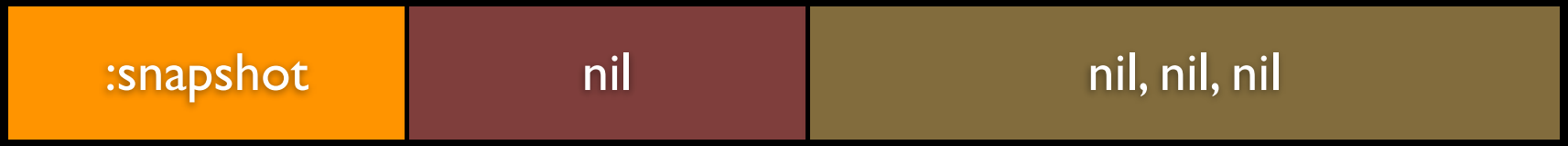
Rinda

- Shared Queue (TupleSpace)
- Built with DRb
- RingyDingy makes it stupid easy
- See Eric Hodel's documentation
- O(N) for take(). Sigh.

Sigh.

- Shared Queue (TupleSpace)
- O(N) for take(). Sigh.

O(N)

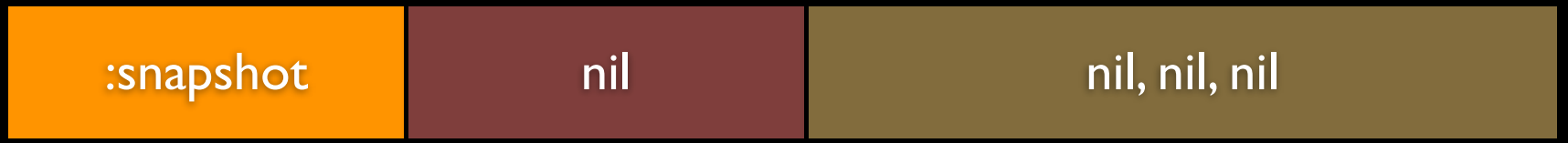


- メッセージだけで探す
- 要求が溜まってるとき、すぐにマッチ
- まぁまぁO(1)じゃない?

まぁまぁO(1)?

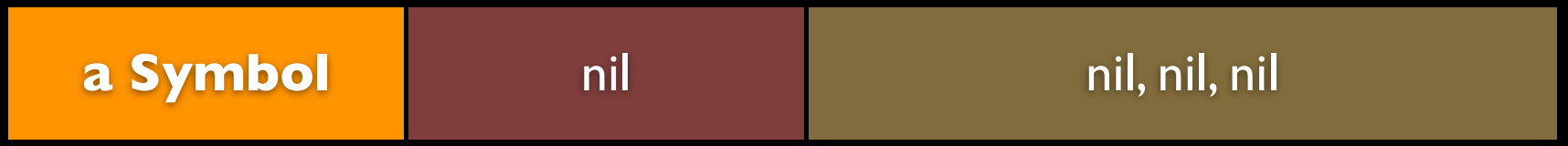
- あれれ
  - Array#deleteしてた
  - delete_atに修正しました
    - おまけ: rindexで末尾を先に削除

不公平な検索



- 事前にメッセージごとに分類
  - 先頭の要素がSymbolの場合のみ
  - StringやIntegerは===の扱いが難しい
- ほんとにO(1)っぽい

不公平な検索



- tuple[0]はSymbolにしておくと効率よい
- そういうケースだけ特別扱いするよ

Rinda遅いんですけど

- さっきロビーで
- ruby_1_8を試して

Rinda and DRb in the Real World

- Tupleの設計はプロトコル設計
- SPOFだよ
- RingServerは鬼門
- 不揮発性のオブジェクトID
- 2004年から開発停まってるじゃん

RubyKaigi 2006

- サーバ側がマシンがトラブるとエラーになります。マスターを複数に出来るといいなと思うんですが。(森脇さん)
- 今聞いたんで、これから考えます。(咳)

考えた

- さっきの状態で生き返るTupleSpace
  - ただしTupleの状態だけ
  - take, readで待ってた人は再試行plz
- 自動的な再起動は別途
  - Ring ?

永続化

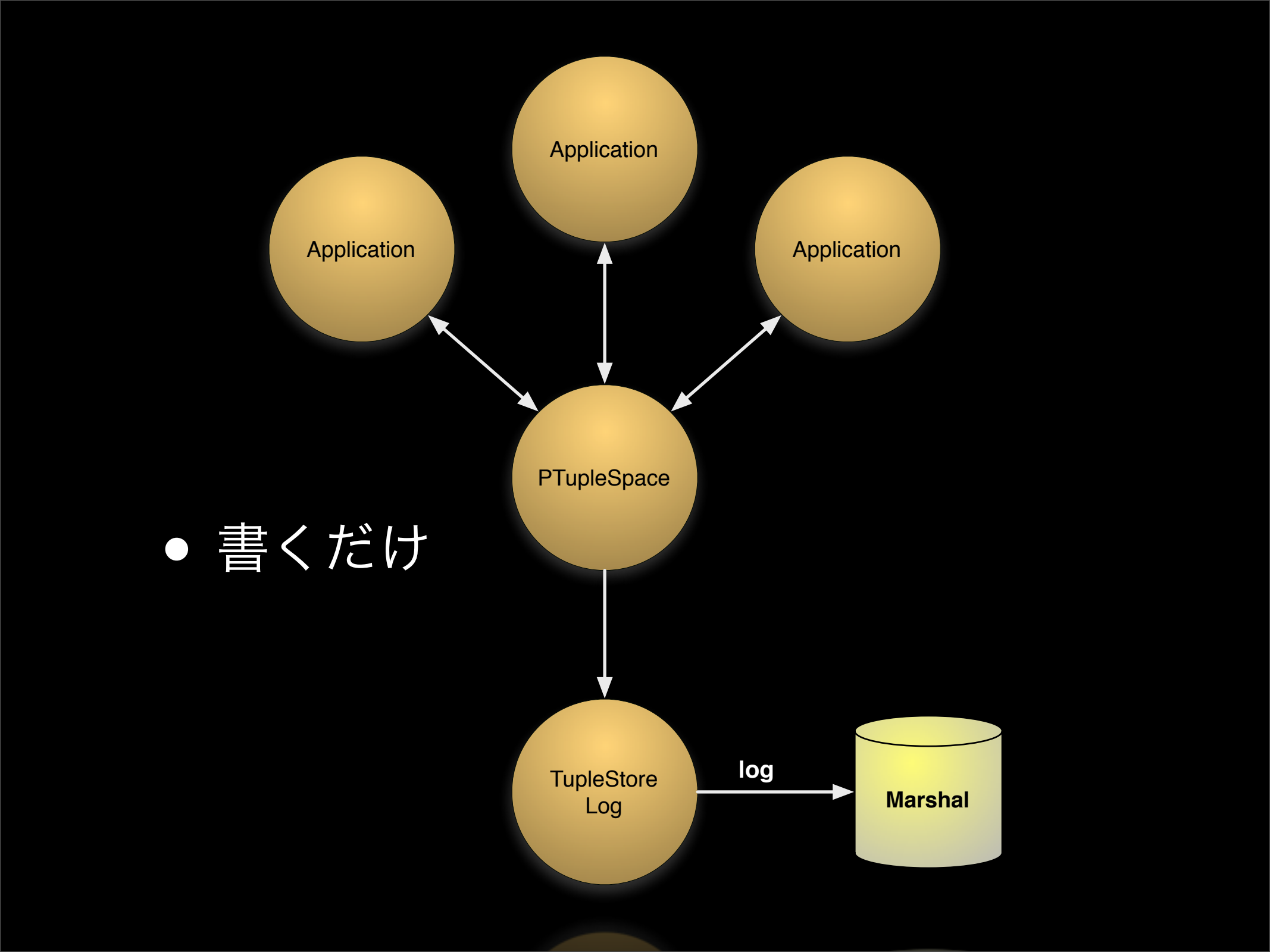
- タプルの内部表現の状態を外部に記録
- 変更も記録
- 再起動時に記録された情報で再構築

PTupleSpace

- 永続版TupleSpace
- TupleとTupleEntryの永続化
  - Tupleのcancelなどのハンドル
- Tupleを保存するTupleStoreと一緒に

TupleStore

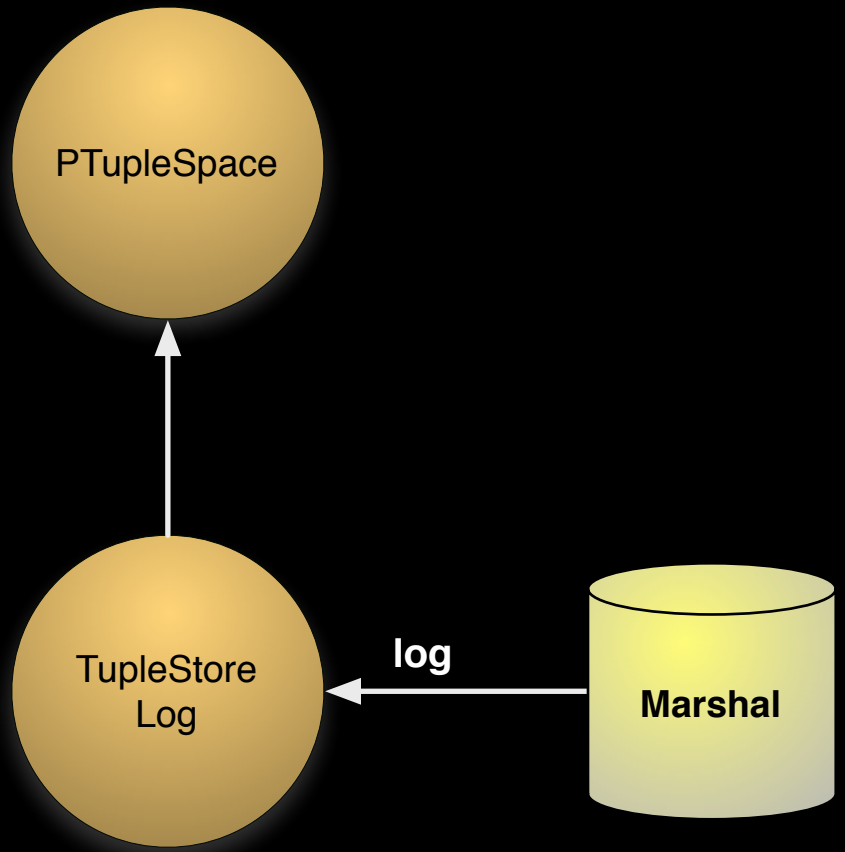
- Tupleの情報をメモしとく
- ログファイルのTupleStoreLog
- HashのTupleStoreSimple
  - プロセスの寿命の違いを利用
  - 開発中向け
- RDBなら安心する人は自分で書いてね


Application
Application
Application
PTupleSpace
TupleStore
Log
log
Marshal

- 書くだけ

ログから生き返る

- 読むだけ



再起動の通知は

- Ringが使えるかも

TupleStoreの配置は

- PTupleSpaceと同じマシンに
- dRubyで好きなところに

Rinda and DRb in the Real World

- Tupleの設計はプロトコル設計
- SPOFだよ
- RingServerは鬼門
- 不揮発性のオブジェクトID
- 2004年から開発停まってるじゃん

Rinda and DRb in the Real World

- Tupleの設計はプロトコル設計
- SPOFだよ
- RingServerは鬼門
- 不揮発性のオブジェクトID
- 2004年から開発停まってるじゃん

不揮発な参照

- object_idはそのプロセスだけで有効
- 再起動時に困る

不揮発な参照

- DRbIdConvをカスタマイズ
- ログのIDを用いた特別な参照を作る
- 再起動してもTupleEntryの参照が有効

Rinda and DRb in the Real World

- Tupleの設計はプロトコル設計
- SPOFだよ
- RingServerは鬼門
- 不揮発性のオブジェクトID
- 2004年から開発停まってるじゃん

2004年で停まってる

- 変えたよ。
  - 変えないと、ダメなの?

まとめ

- 関心をもってくれたみなさまに感謝
  - RubyKaigi 2006
  - Rinda and DRb in the Real World.
  - Scaling Twitter

それから

- たまにはERBの言い訳をしたい
  - なぜ行に分割するのか
  - メソッド化して使うのがERB way

toRuby 4h

- とちぎRubyの勉強会
- 6/23 西那須野公民館
- 豪華ゲスト
- 豪華レギュラー