偉大なBigTableとぼくのおもちゃ

RubyKaigi2009のテーマはたぶんCOBOL

seki@druby.org

自己紹介

- 省略

今年も

- いつも同じ話を聴いてくれてありがとう

￥3360

- 初刷まだ買えます
- 英語版はPragmatic Bookshelfより
  - いつでるの？

$32.00

- International Journal of PARALLEL PROGRAMING

Int J Parallel Prog
DOI 10.1007/s10766-008-0086-1

dRuby and Rinda: Implementation and Application of Distributed Ruby and its Parallel Coordination Mechanism

Masatoshi Seki

Received: 20 March 2008 / Accepted: 13 August 2008

Abstract The object-oriented scripting language Ruby is admired by many programmers for being easy to write in, and for its flexible, dynamic nature. In the last few years, the Ruby on Rails web application framework, popular for its productivity benefits, has brought about a renewed attention to Ruby for enterprise use. As the focus of Ruby has broadened from small tools and scripts, to large applications, the demands on Ruby's distributed object environment have also increased, as has the need for information about its usage, performance and examples of common practices. dRuby and Rinda were developed by the author as the distributed object environment and shared tuplespace implementation for the Ruby language, and are included as part of Ruby's standard library. dRuby extends method calls across the network while retaining the benefits of Ruby. Rinda builds on dRuby to bring the functionality of Linda, the glue language for distributed co-ordination systems, to Ruby. This article discusses the design policy and implementation points of these two systems, and demonstrates their simplicity with sample code and examples of their usage in actual applications. In addition to dRuby and Rinda's appropriateness for prototyping distributed systems, this article will also demonstrate that dRuby and Rinda are building a reputation for being suitable infrastructure components for real-world applications.

Keywords Ruby · Linda · dRuby · Rinda · TupleSpace · RMI

M. Seki (✉)
Tochigi, Japan
e-mail: m_seki@mva.biglobe.ne.jp
URL: www.druby.org

Springer

とちぎRuby会議02

- もうすぐ裏番組で発表!?

Agenda

- おことわり
- MapReduceとRinda編
- マッチポンプ編

電波

- 「電波です」と宣言するとムリな展開も許される風潮がある

電波な講演の例

kakutani.com「時を超えたプログラミングの道への道」より

おことわり

- 今日は電波
- 技術的なことを期待しないで

MapReduceとRinda編

MapReduce

- 単語数を調べるシステム
- ついでに転置インデックスまで
- かっこいいバッチ処理
  - 私は本物を知らないので想像

かっこいいバッチ処理

- 二つの関数(map, reduce)の組み合わせで大規模データを計算するバッチ処理
- mapとreduceの周りがおもしろそう

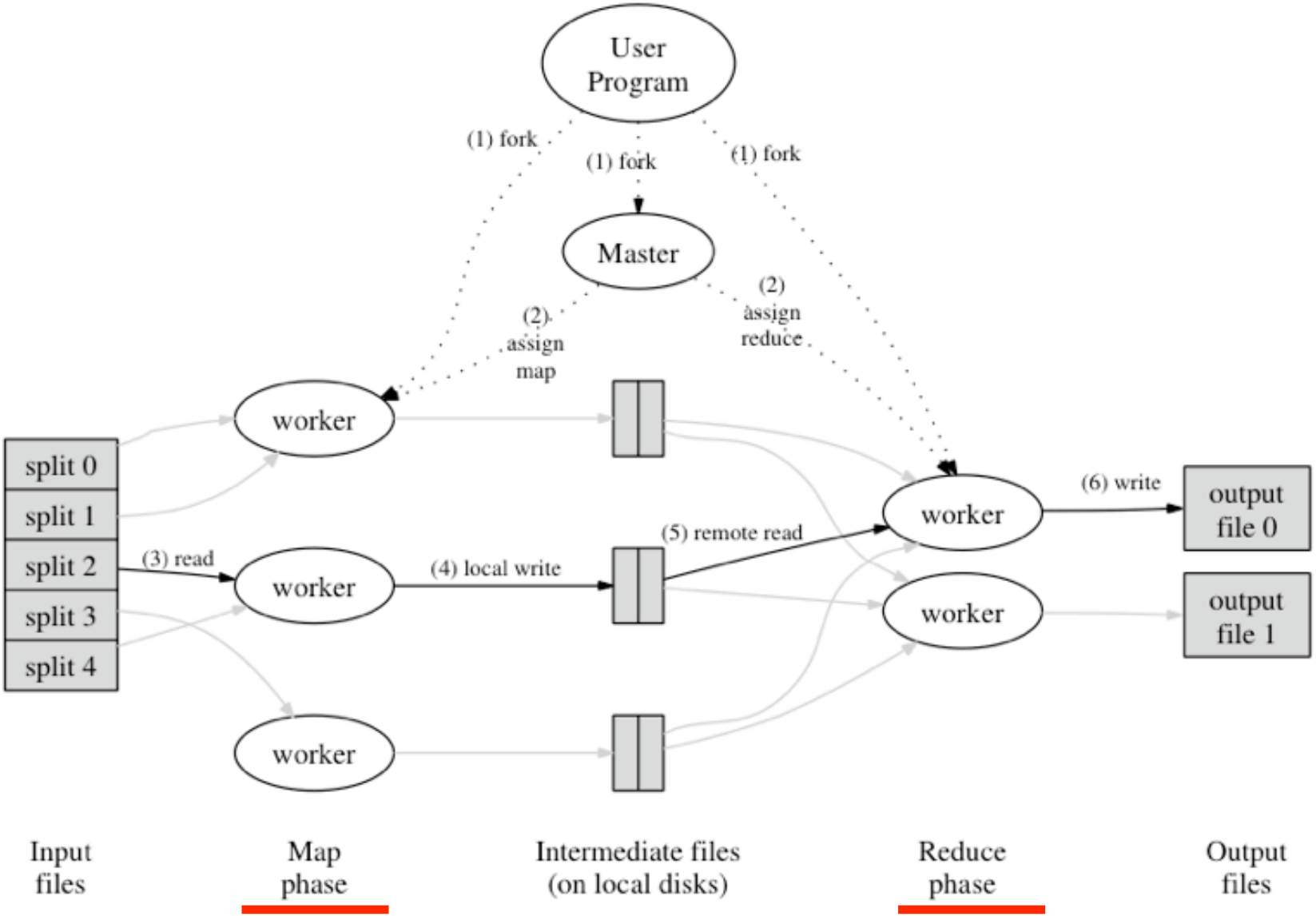


Figure 1: Execution overview

Figure 1: Execution overview

MapReduce : Simplified Data Processing on Large Clustersより

map

- keyとvalueの組を出力
  - [単語, 単語数]や[単語, 文書ID]など

map()

["eval", 1]
["c", 1]
["Author", 1]
["nobu", 1]
["created", 1]
["at", 1]
["Thu", 1]
["Jun", 1]
["10", 1]
["14", 1]
["22", 1]
["17", 1]
["JST", 1]
["1993", 1]
["Copyright", 1]
["C", 1]
["1993", 1]
["2007", 1]
["Yukihiro", 1]
["Matsumoto", 1]
["Copyright", 1]
["C", 1]
["2000", 1]
["Network", 1]

["Applied", 1]
["Communication", 1]
["Laboratory", 1]
["Inc", 1]
["Copyright", 1]
["C", 1]
["2000", 1]
["Information", 1]
["technology", 1]
["Promotion", 1]
["Agency", 1]
["Japan", 1]
["include", 1]
["eval_intern", 1]
["h", 1]
["include", 1]
["iseq", 1]
["h", 1]
["VALUE", 1]
["proc_invoke", 1]
["VALUE", 1]
["VALUE", 1]
["VALUE", 1]
["VALUE", 1]
["VALUE", 1]

["rb_binding_new", 1]
["void", 1]
["NORETURN", 1]
["void", 1]
["rb_raise_jump", 1]
["VALUE", 1]
["ID", 1]
["rb_frame_callee", 1]
["void", 1]
["VALUE", 1]
["rb_eLocalJumpError", 1]
["VALUE", 1]
["rb_eSysStackError", 1]
["define", 1]
["exception_error", 1]
["GET_VM", 1]
["special_exceptions", 1]
["ruby_error_reenter", 1]
["include", 1]
["eval_error", 1]
["c", 1]
["include", 1]
["eval_jump", 1]
["c", 1]
["initialize", 1]

["ruby", 1]
["if", 1]
["defined", 1]
["__APPLE__", 1]
["define", 1]
["environ", 1]
["_NSGetEnviron", 1]
["elif", 1]
["defined", 1]
["_WIN32", 1]
["extern", 1]
["char", 1]
["environ", 1]
["endif", 1]
["char", 1]
["rb_origenviron", 1]
["void", 1]
["rb_clear_trace_func", 1]
["void", 1]
["void", 1]
["rb_thread_stop_timer_thread", 1]
["void", 1]
["void", 1]
["rb_call_inits", 1]
["void", 1]

["void", 1]
["Init_heap", 1]
["void", 1]
["void", 1]
["Init_BareVM", 1]
["void", 1]
["void", 1]
["ruby_init", 1]
["void", 1]
["static", 1]
["int", 1]
["initialized", 1]
["0", 1]
["int", 1]
["state", 1]
["if", 1]
["initialized", 1]
["return", 1]
["initialized", 1]
["1", 1]
["rb_origenviron", 1]

出力はなにもの?

- map()の出力は順序づけられreduce()へ
- どう格納されてるかはともかくそれを観察するときには順序がついてる
  - 巨大なordered listであるBigTableで作られてるのかなあと想像

reduce

- mapフェーズが終わったら呼ばれる
  - たぶん辞書順
- keyとたくさんのvalueを受け取る
- あらたなkeyとvalueを出力
  - [単語, 単語数]とか

reduce()

["ANYARGS", 1]
["ANYARGS", 1]
["ANYARGS", 1]
["ANYARGS", 1]
["ANYARGS", 1]
["ANYARGS", 1]
["ANYARGS", 6]
["ARGV", 1]
["ARGV", 1]
["Adds", 1]
["Adds", 1]
["Agency", 1]
["Agency", 1]
["Applied", 1]
["Applied", 1]
["ArgumentError", 1]
["ArgumentError", 1]
["ArgumentError", 2]
["Array", 1]
["Array", 1]
["Array", 1]
["Array", 3]

["Author", 1]
["Author", 1]
["Bignum", 1]
["Bignum", 1]
["Binding", 1]
["Binding", 1]
["C", 1]
["C", 1]
["C", 1]
["C", 3]
["CONST_ID", 1]
["CONST_ID", 1]
["Call", 1]
["Call", 1]
["Can", 1]
["Can", 1]
["Can", 2]
["Check_Type", 1]
["Check_Type", 1]
["Check_Type", 1]
["Check_Type", 3]

["Communication", 1]
["Communication", 1]
["Copyright", 1]
["Copyright", 1]
["Copyright", 1]
["Copyright", 3]
["EXEC_EVENT_HOOK", 1]
["EXEC_EVENT_HOOK", 1]
["EXEC_TAG", 1]
["EXEC_TAG", 1]
["EXEC_TAG", 1]
["EXEC_TAG", 1]
["EXEC_TAG", 1]
["EXEC_TAG", 1]
["EXEC_TAG", 1]
["EXEC_TAG", 1]
["EXEC_TAG", 1]
["EXEC_TAG", 1]
["EXEC_TAG", 1]

reduceを呼ぶ係は

- どうやって動くんだろう
- 要素を一つずつ見てkeyが同じ間だけ働くイテレータを渡す係っぽい
  - 関数型方面の心に響く

SQLでいえば

- group_byと集約関数とか

COBOLでいえば

- 「コントロールブレイク」とか
  - COBOLの授業で教わったよ
  - RubyKaigi2009のテーマはCOBOL

コントロールブレイク

- 一つずつ読んでkeyが変わったら処理を切り替えるパターン
- 言語に依らない処理ですね

reduceを呼ぶ係は

- それをイテレータにしたりかっこよく見せてくれる係
  - と想像される

二つの周りにあるもの

- [key, value]のデータを共有する空間
- 並列処理の待ち合わせ
- 障害時の再起動
  - ほんとはココ↑がすごいんだと思うけどね

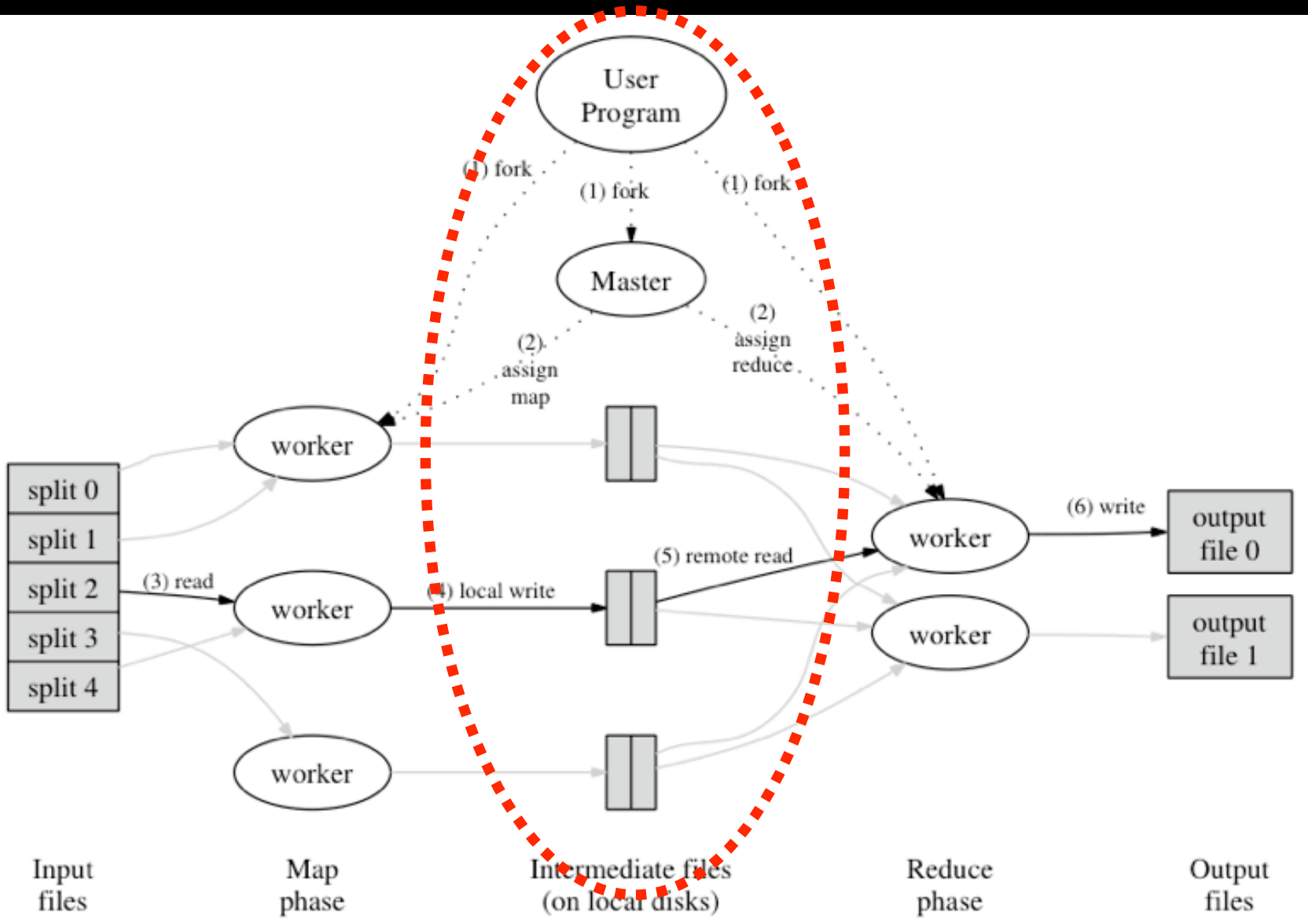


Figure 1: Execution overview

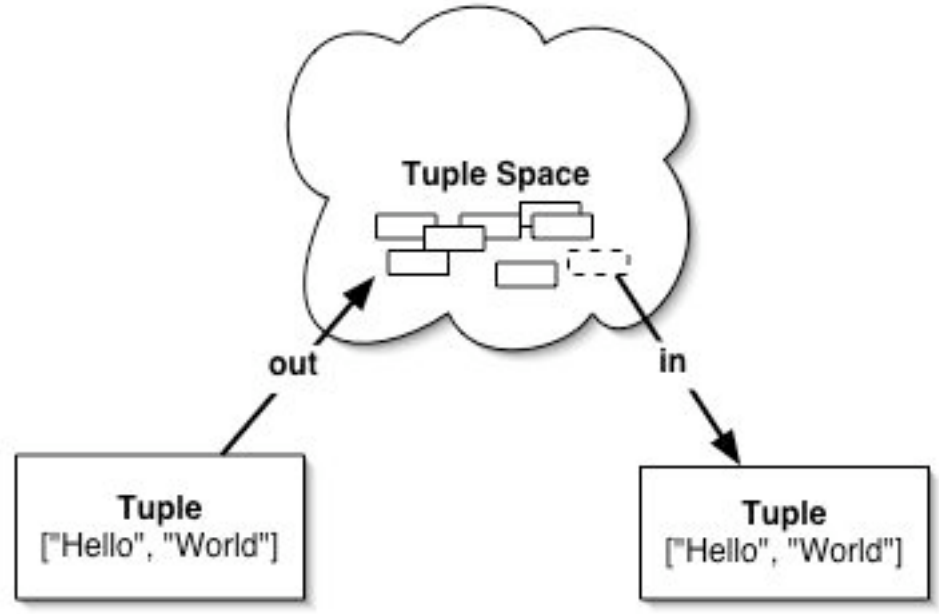
MapReduce : Simplified Data Processing on Large Clustersより

そういえば

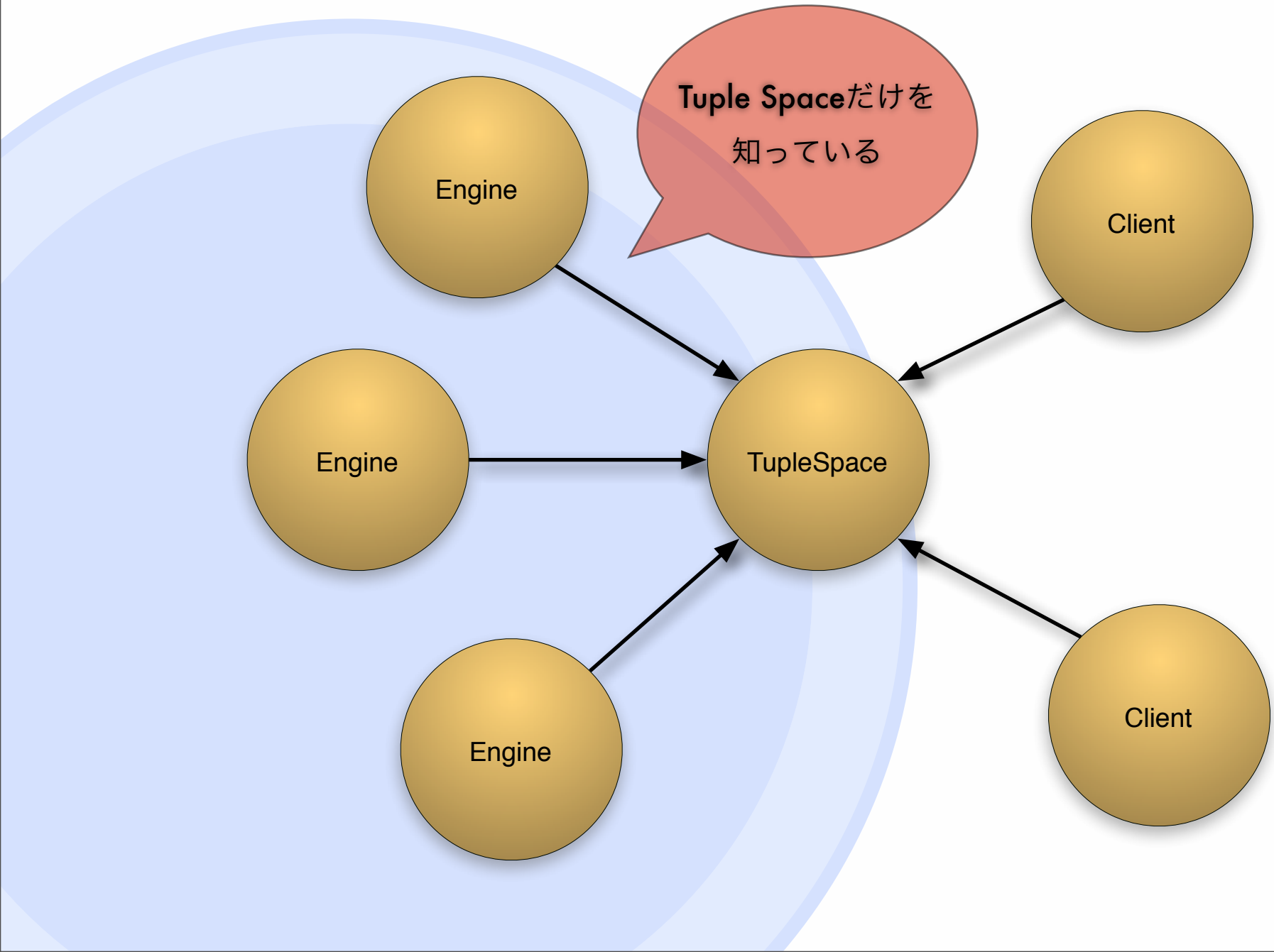
- Rindaのキーワードと似てる
  - 並列処理とか
  - タプルを共有する空間

Linda

- tuple, tuple space
- out, inとrd
  - パターンで指定
  - ブロックできる



かっこいい並列処理


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

Rinda

- RubyによるLindaの実装
- dRuby向けにちょっと調整
- TupleはArrayで表現

Tuple

```
[:chopstick, 2]
[:room_ticket]
['abc', 2, 5]
[:matrix, 1.6, 3.14]
['family', 'is-sister', 'Carolyn', 'Elinor']
```

Pattern

- Tupleの要素と===で比較する
  - case equalsなのでパターンっぽく動く
  - Regexp, Range, Classとか
- nilはワイルドカード

Pattern

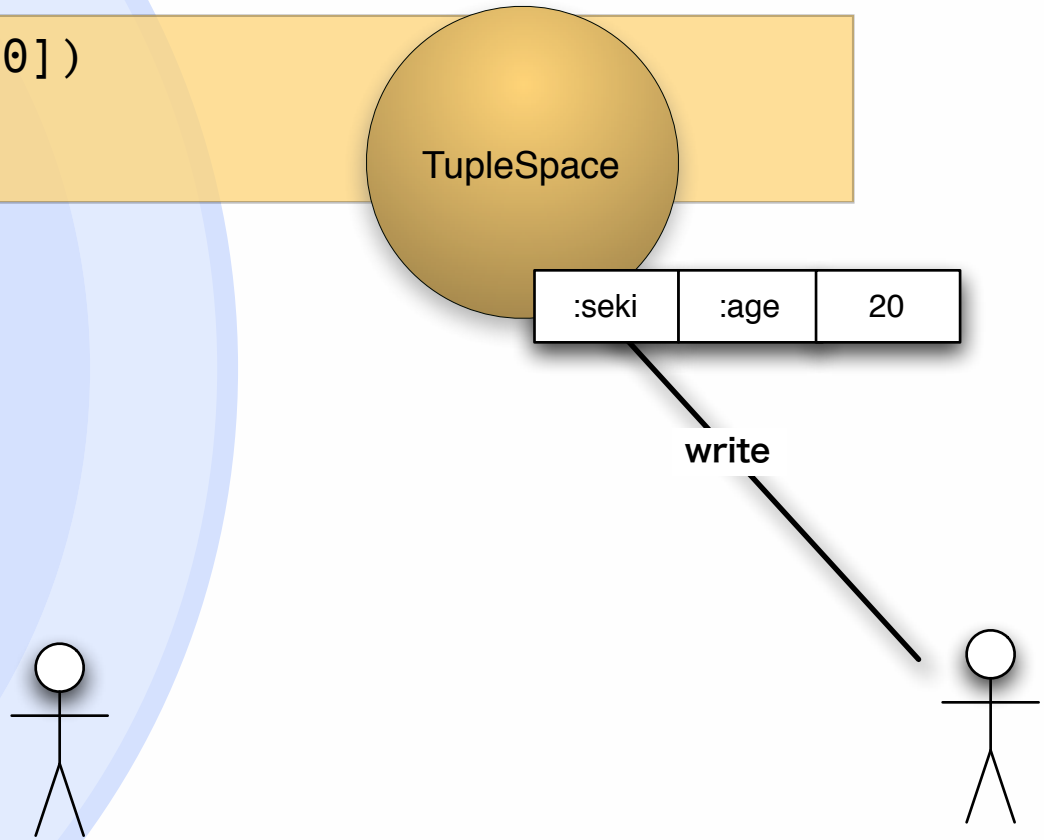
```
[/^A/, nil, nil]
[:matrix, Numeric, Numeric]
['family', 'is-sister', 'Carolyn', nil]
[nil, 'age', (0..18)]
```

['seki', 'age', 20]

```
>> require 'rinda/tuplespace'
>> ts = Rinda::TupleSpace.new
>> ts.write(['seki', 'age', 20])
>> ts.write(['sougo', 'age', 18])
>> ts.write(['leonard', 'age', 18])
>> ts.read_all([nil, 'age', 0..19])
=> [["sougo", "age", 18], ["leonard", "age", 18]]
>> ts.read_all([/^s/, 'age', Numeric])
=> [["seki", "age", 20], ["sougo", "age", 18]]
```

writeとtake

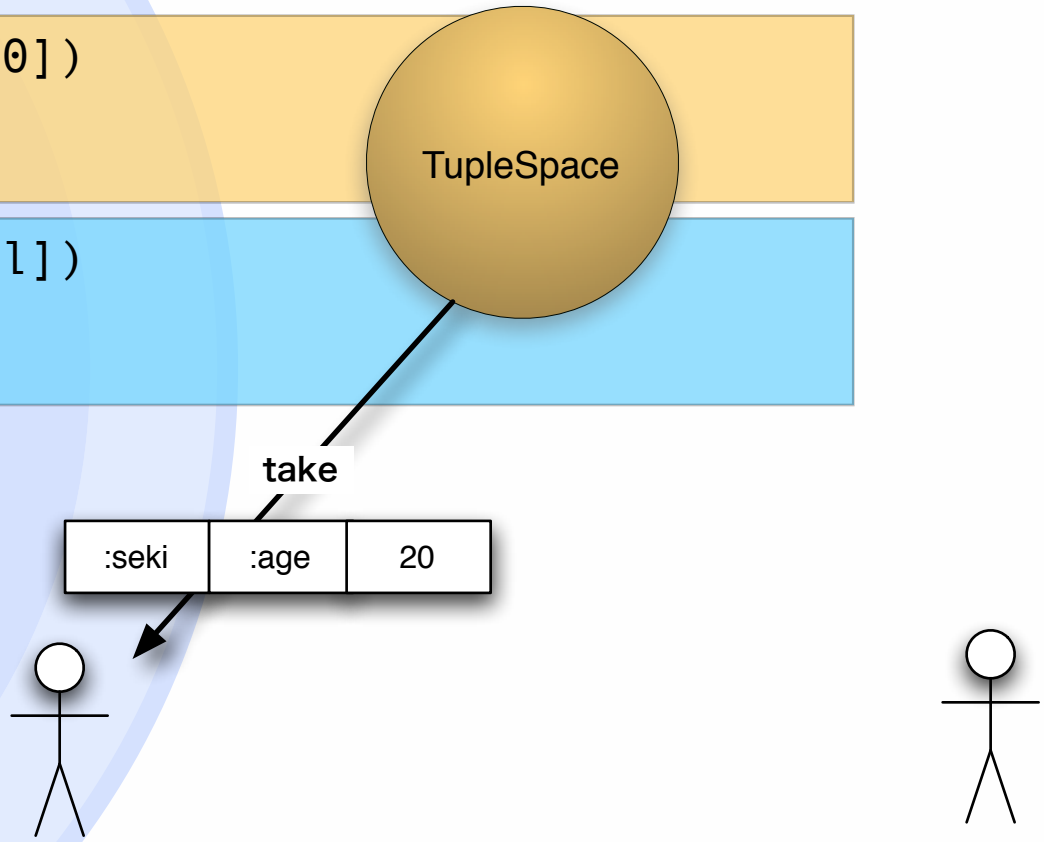
```
>> ts.write([:seki, :age, 20])
=> #<Rinda::TupleEntry:...>
>>
```



writeとtake

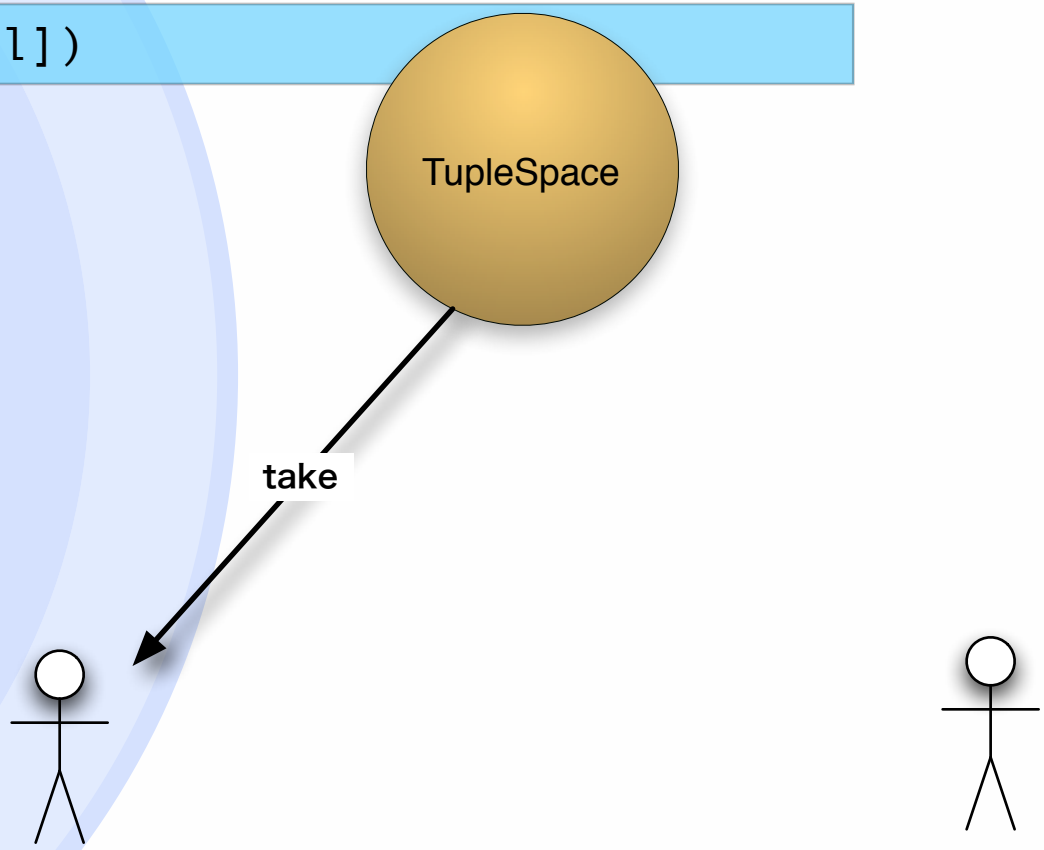
```
>> ts.write([:seki, :age, 20])
=> #<Rinda::TupleEntry:...>
>>
```

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



待ち合わせ

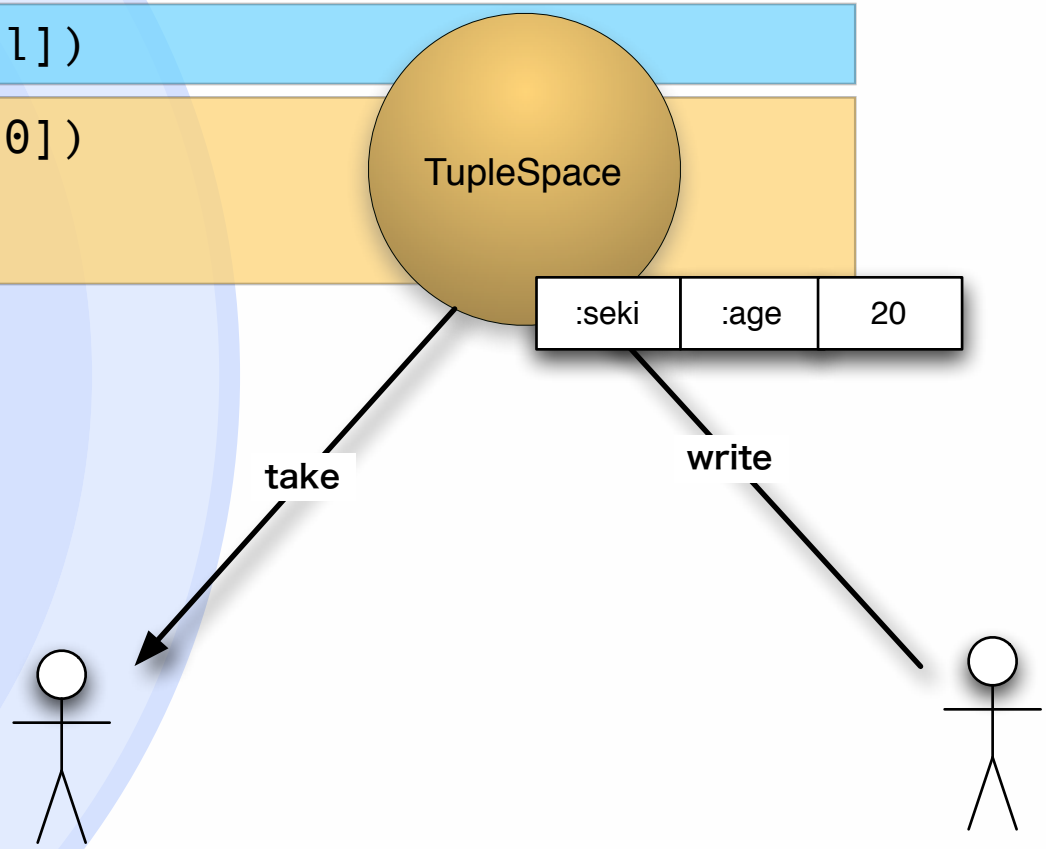
```
>> ts.take([:seki, :age, nil])
```



待ち合わせ

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

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

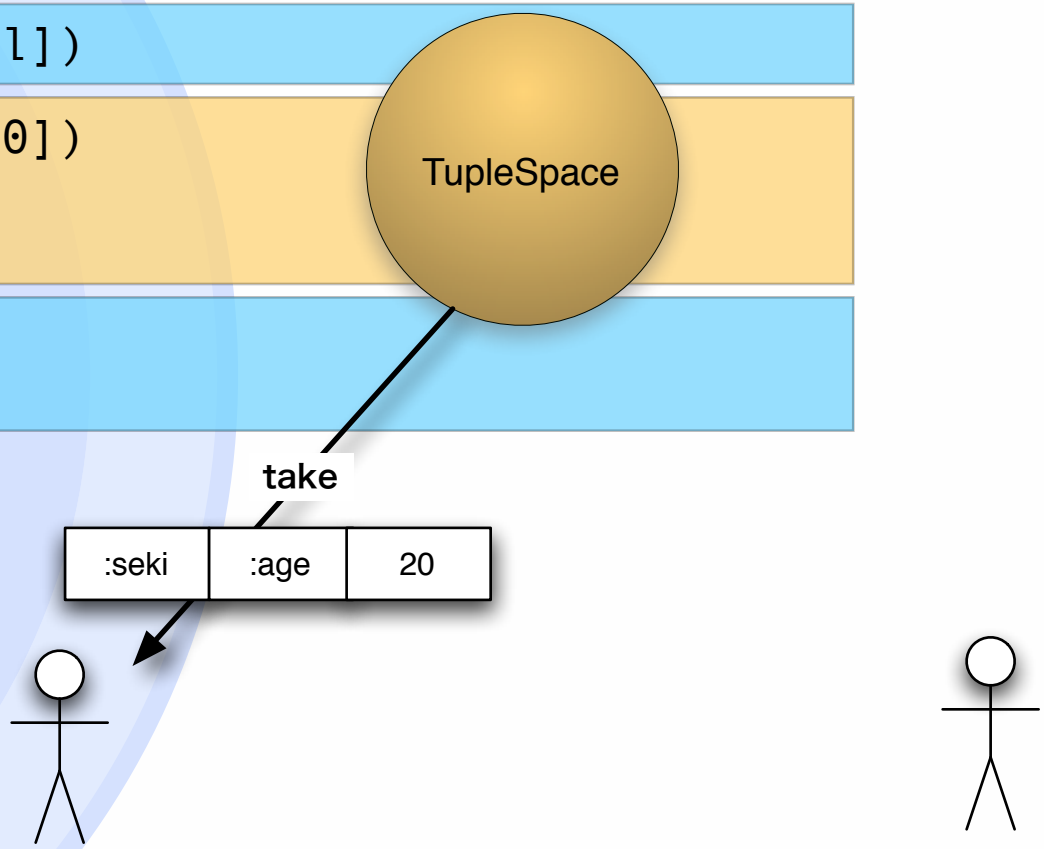


待ち合わせ

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

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

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



くわしくは勉強会で

- toRuby勉強会
  - 毎月第一水曜日
  - 西那須野公民館

さっきのアレ

- Rindaで単語を数えるシステムを書くぞ
  - MapReduceっぽいの実装ブームに乗り遅れないぞ

タプルと処理の設計

- [String, Integer]かな
  - 単語と単語数のタプル
- mapとreduceの二つのプロセスでやるか

map簡単

```
while line = gets
  line.scan(/\w+/) do |w|
    ts.write([w, 1])
  end
end
```

さて数えよう

```
word = ts.read_all.sort_by {|t| t[0]}.first[0]
```

- 一番最初の単語はなにかな
- 苦手..
- O(n)ですよ

その単語を全部集める

```
ary = ts.read_all([word, nil])
```

- 全部一度に読む？

ではその単語の次は?

```
ts.read_all.sort_by {|t| t[0]}.first
```

- これも苦手!!
- またO(n)

reduce?

- だれが同じ単語をまとめてくれるの？
- 並べるのは誰？

Hashも同様に微妙

```
hash.keys.sort
```

- keys.sortかよ
- 次のkeyが表現できたらいいのに
- hashは知ってるkeyに対してうまく動く

keyに対する操作

- HashやTupleSpaceにkeyの順序はない
- 知ってるkeyの有無を調べるのは得意
- 次のkeyを求めるのは苦手
  - read_all, keysなど全部集める操作で代用

Lindaは

- ほんとは並列処理糊言語だよ
- 並列処理の協調の仕組み
- もっと適したデータ構造があると思う

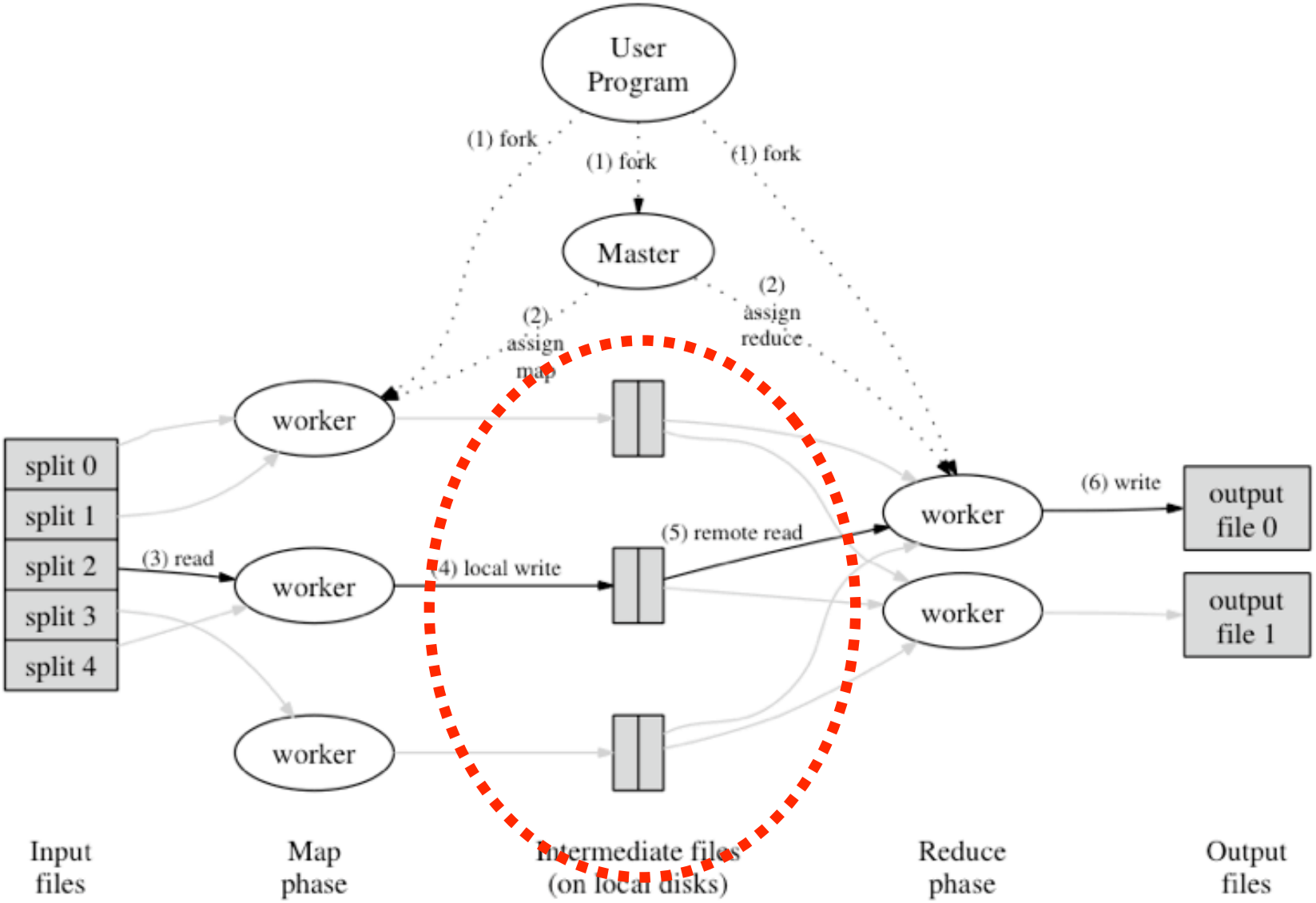


Figure 1: Execution overview

Figure 1: Execution overview

MapReduce : Simplified Data Processing on Large Clustersより

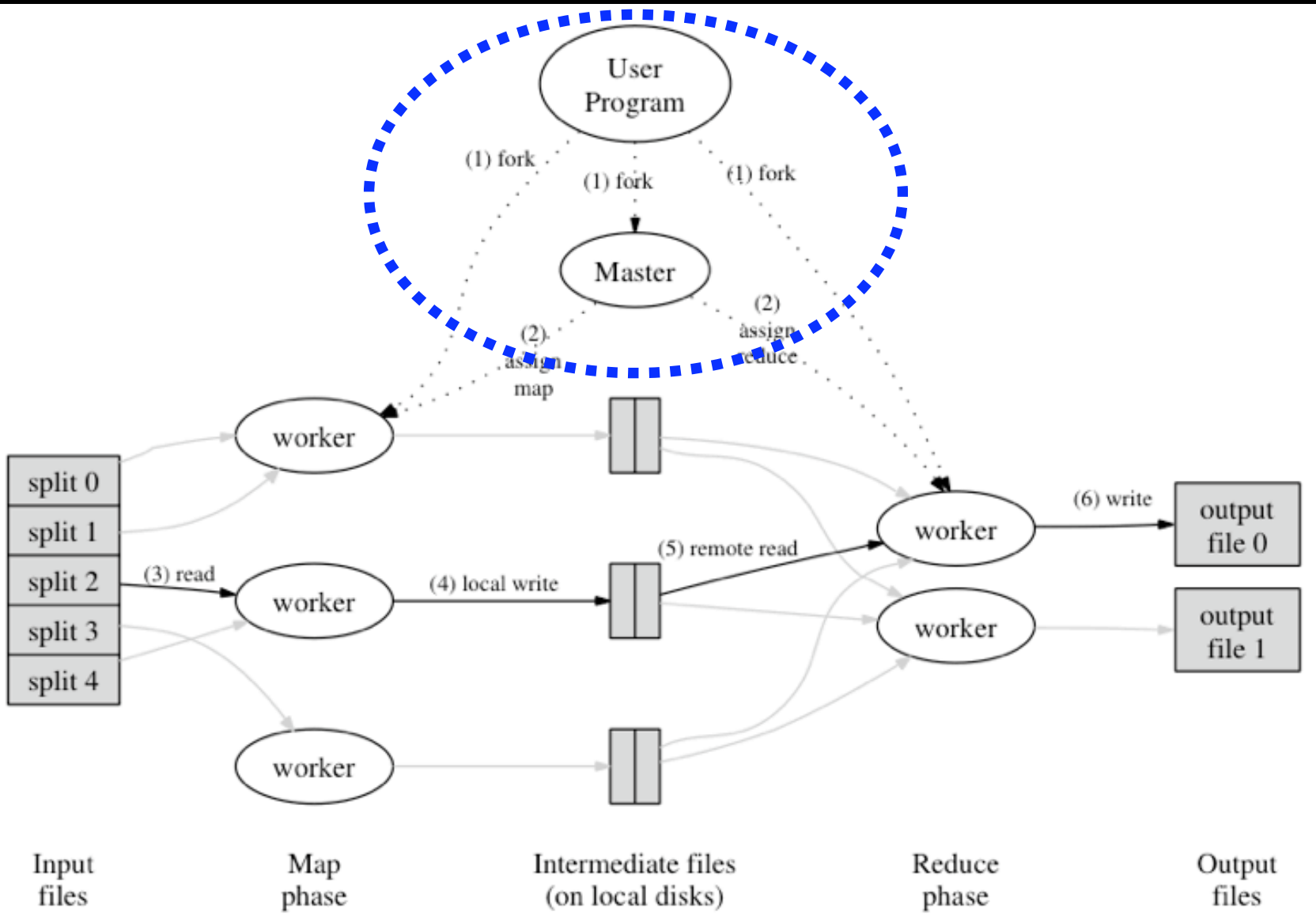


Figure 1: Execution overview

MapReduce : Simplified Data Processing on Large Clustersより

遷移するには

- reduceには分類が必要
  - 全部そろわないとダメ
- みんなが完了するまで待ち合わせ
  - こういうときこそLindaの出番

そういえばKVS

- 定義がさっぱりわかりません
- 二要素からなるタプルなんだろか
- RDB対抗の製品かどうか？

KVSっぽいってHash?

- 抽象データ型のように操作から考える
- RubyのHashみたいかどうか？

mapとreduceの間

- それらはどうもHashではないみたい
- key-valuesだし
- keyを順に辿れるし
  - 知らないkeyを扱う

むしろリアル辞書っぽい

- keyで並んでる
- 高速な検索と連続的なアクセス

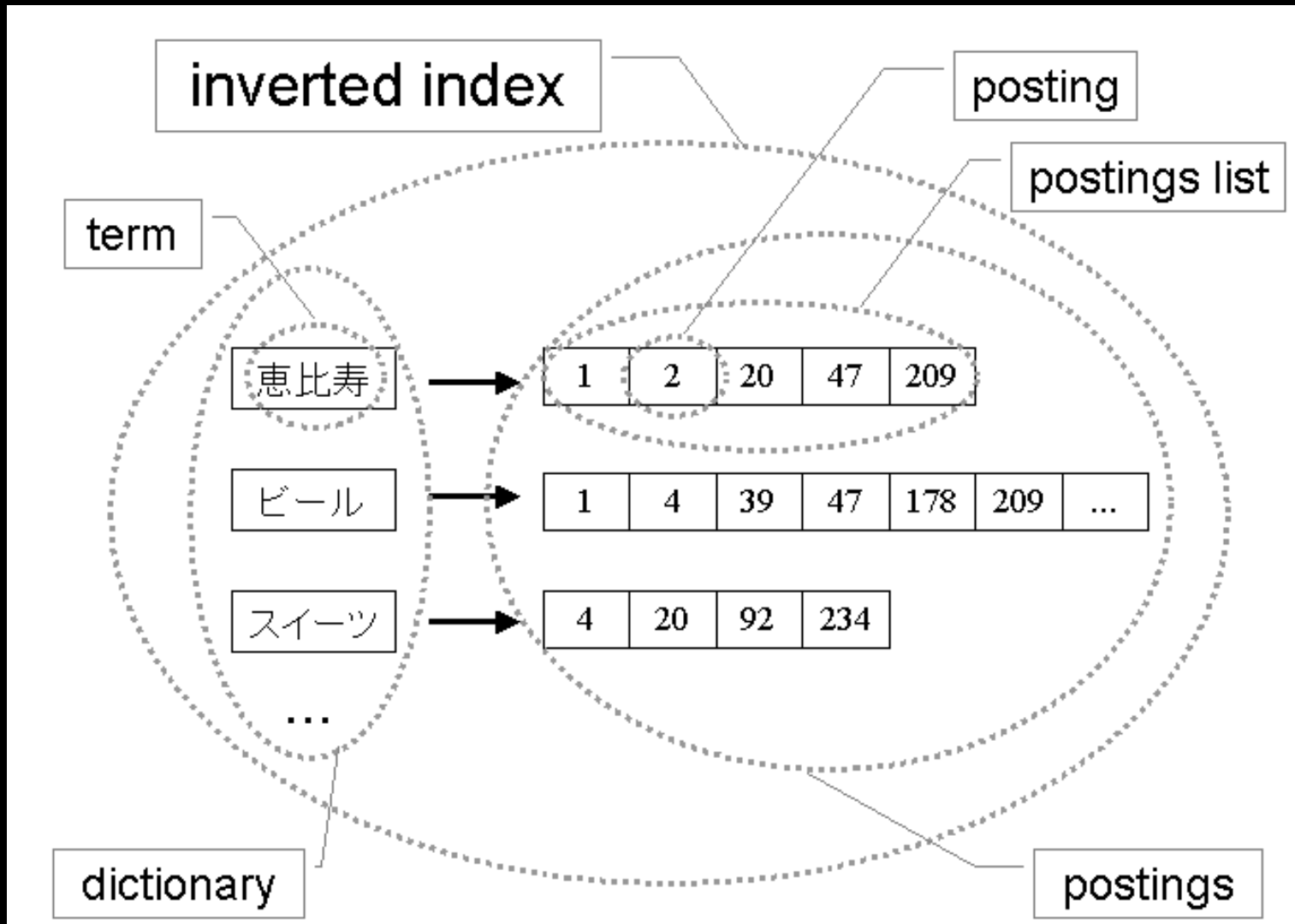
keyってなんだろう

- 単なるID
- 意味のある情報
  - 文字列
  - オブジェクト

例: 転置インデックス

[単語，文書IDや位置]

- 単語の現れた文書、位置
- 索引



http://chalow.net/2008-01-18-1.html
[を] 転置インデックスの構成とブーリアン検索

boolean検索

- and, or検索の実現
- 位置の順に並んでいると効率よい
- 小さい方のカーソルを進めてく

この場合のkeyは

- 単語と文書IDで並んでるとうれしい

keyもまたvalueだ

- [[単語, 文書ID], nil]がうれしい？
- keyそのものが情報だ
  - valueいらんかも
- keyもまた意味のある情報である

実験してみる

keyに順序がある集合

- たとえばBigTable
- あるいはTokyo Cabinet
- それからRBTree
- 二つはすぐに使えるよ

Tokyo Cabinet

- QDBMの人の作品。B+木。前に使った
- keyはStringで順序がある
- データの重複を許す
- ファイルベース
- 並列性よくわからん
  - 書き込みモードで二人が開くと後側が待たされちゃうんだけどCで書けば大丈夫なの？

Koya

- sqlite→QDBM→Tokyo Cabinetを使って書いたOODB
- Rubyのオブジェクト空間をkey, valueで表現
- 世界はkeyとvalueで表現できる
  - 番地と値で実装されてるんだから当然

Koyaの中身

```
# memory
[“m.#{oid}”, class.to_s]

# property
[“p.#{oid}@#{name}”, Marshal.dump([klass, value])]
```

- 全て一つの表に入ってる
- 階層的なkeyでオブジェクトの属性を表現する

Koyaの狙い

- サーバ・クライアント形態のOODBはdRubyでやればいいのでつまんない
- それ以外の実装のOODBにどんな意味があるのかを確かめる
- Koyaは世界をファイルに置いて共有することをテーマとした

Koyaの失敗

- Rubyはロールバックを持たないのにtransactionを導入してしまった
  - シームレスじゃない
- プログラミングスタイル違うよね
- そもそも応用が思いつかない

RBTree

- RB木 (≠Ruby木)
- KeyはRuby Objectで順序がある
- データの重複を許す - MultiRBTree
- インメモリ
- わりとRubyっぽい

インメモリ

- dRubyファンはまず最初に検討すべき
- ファイルに書く必然性がないときに
- たいていin-memory + loggingで済むよ
- フォロワーたくさん
  - memcachedとか(ちょっと嘘)

メモリに入らない量

- アドレス空間の限界
  - dRubyでプロセスを分ければいい
- 物理メモリの限界
  - dRubyでマシンを分ければいい
- その先は二次記憶にするのかな

欠点

- なんとなく不安
- FUDに弱い

問題を考えよう

- とりあえず転置インデックスを作る？
- 工夫する必然性がある量
- マッチポンプ万歳

けっこう大変

- ご家庭や職場で生成するテキストの量には限りがある
  - 楽天うらやましい!!!
- ふつうのデータでは工夫しなくても困らない
- なんとかして大きな問題を作ろう！

今回のネタ

- ファイルに時間をかけ算して水増し
- CVSリポジトリからデータを作ろう!
- 変更ごとにadd/deleteされた単語を追跡
- ある単語がadd/deleteされたcommitを検索できるようにする

データの生成

- リポジトリをakrさんのRuby/CVSで展開
- add/deleteされた行を分析して転置インデックスを作る
- 履歴は減らないのでデータの追記だけ
  - 更新にまつわる難しい問題がない

やっと

- 工夫しないと高速に検索できない問題を手に入れた

それって

- ,vファイルを直接検索して表示で工夫すればいいじゃん？
- そういうことは言わない

転置インデックス

```
[[word, doc_id, rev_id, mode, lineno], nil]
```

- 単語, 文書ID, revision ID, add|del|log, 行
- 実際にはいろいろ圧縮するしかけも
  - 今回は割愛

実装

- Tokyo CabinetのBDD
- ちょっとしたキャッシュつきデーモン
  - WEBrick::CGIと4行のcgiスクリプト

うごきました

- あたりまえだけど
- 数千ファイル過去10年分の変更履歴を瞬時に検索
- デーモンはまったく落ちない

開発中に気付いたこと

- 試行錯誤が続く時期に全部処理するの面倒
- 各フェーズの途中からやりなおしたい
- 領域ごとにやりなおしたい
  - あー、こういうことのサポートが充実してるんだろな > Googleの内側

重要なことをもう一度

- 天丼

￥3360

- 初刷まだ買えます
- 英語版はPragmatic Bookshelfより
  - いつでるの？

まとめ

- まだ初刷買えます。
- ご家庭ででっかい問題を作るのは大変
- RubyKaigi2009のテーマはCOBOL
- 世界をコントロールブレイク