プログラミング言語 Ruby に GMP を組み込む

田中 哲

GNUプロジェクト30周年記念
FSIJワークショップ
2013-09-29

発表の概要

- Ruby に GMP を組み込んで
  多倍長整数演算を高速化
- 自由ソフトウェア運動として
  どのような意味があるか

プログラミング言語 Ruby

- オブジェクト指向スクリプト言語
- 2-clause BSDL と独自ライセンスのデュアルライセンス
  (Ruby 1.9.2 までは GPLv2 と独自だった)
- Bignum が組み込まれていて
  任意多倍長整数演算が自然に使える
- 31bit/63bit までは Fixnum (ポインタ埋め込み)
- Ruby 2.1.0 (2013-12-25 リリース予定) では
  GMP による Bignum の高速化が組み込まれる

GNU 多倍長演算ライブラリ GMP

- GNU プロジェクトのひとつ
- LGPLv3 or later
- 高速な多倍長整数演算の実装
  - 様々なアルゴリズムの実装
  - アセンブラによる実装
- 有理数演算も実装されている

乗算アルゴリズム

Ruby 2.0.0

- 筆算
- Karatsuba
- Toom-3

GMP 5.1.2

- 筆算
- Karatsuba
- Toom-3
- Toom-4
- Toom-6.5
- Toom-8.5
- FFT

除算アルゴリズム

Ruby 2.0.0

- 筆算

GMP 5.1.2

- 筆算
- Divide and Conquer Division
- Block-Wise Barrett Division

GCDアルゴリズム

Ruby 2.0.0

- ユークリッドの互除法

GMP 5.1.2

- Binary GCD
- Lehmer's algorithm
- Subquadratic GCD

考えられる方針

a) Bignum のデータ構造を mpz_t に変えてしまう
侵襲性高 (C レベル API 非互換、GMP必須)

b）データ構造をコンパイル時選択
侵襲性中 (C レベル API 非互換)

c）高コストな演算で、演算時に mpz_t に変換して演算し、結果を逆変換する
侵襲性低 (API 互換。本体に GMP をリンク)

d）メソッドの再定義して c を行う
侵襲性無 (本体は変化なし。内部呼び出しのすり替えは困難)

今回の方針

- 高コストな演算で、演算時に mpz_t に変換して演算し、結果を逆変換する
- 変換は数のビット長に対して線形コストなので、漸近的に Ruby よりも GMP のほうが速い演算については、数が大きくなれば変換を含めても GMP のほうが速い
- 演算
  乗算、除算、GCD、
  基数変換（2〜36進文字列との相互変換)

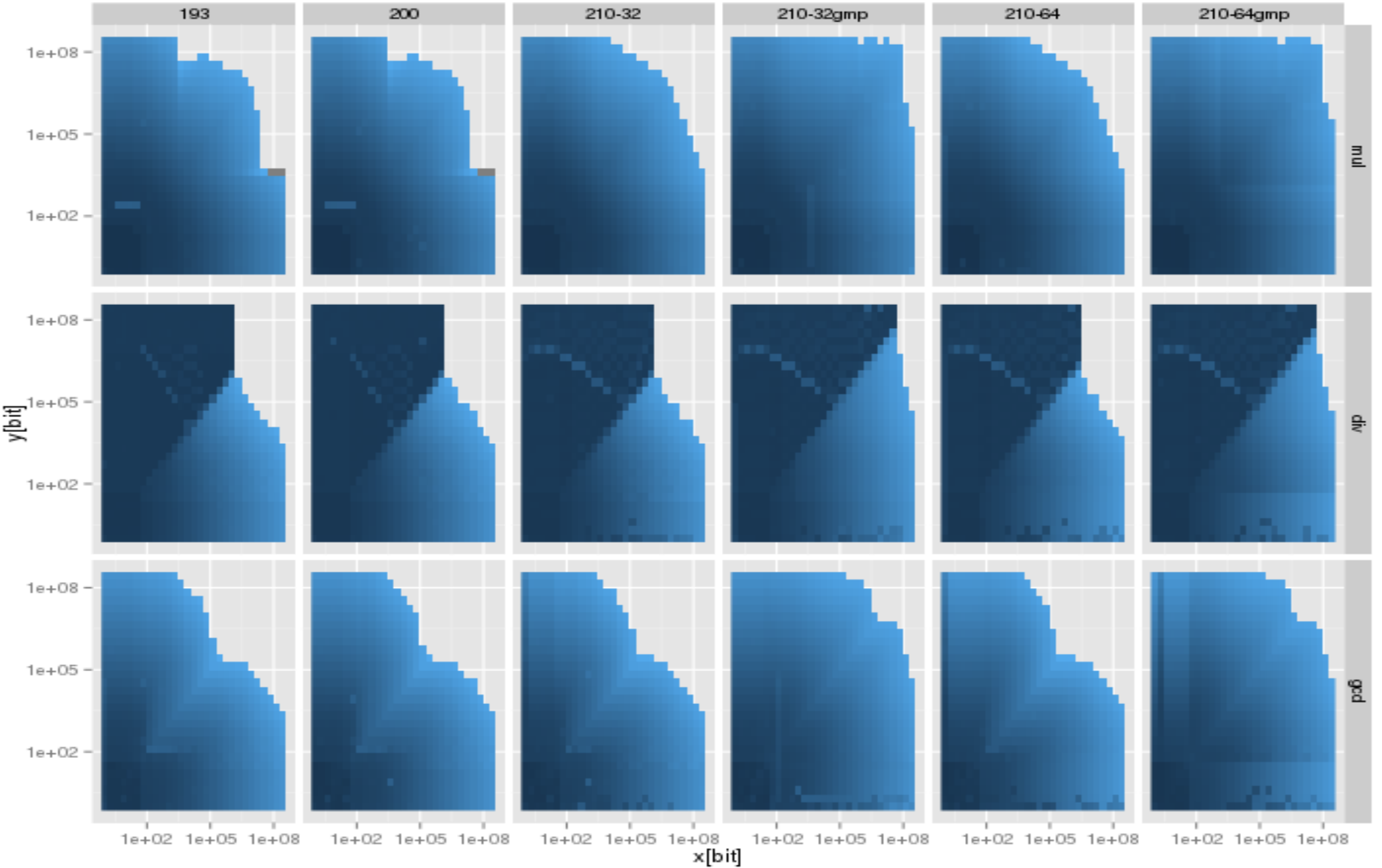
GMP以外の変更

- 可能なら __int128 を利用する (gcc 4.6 から)
  Ruby の Bignum は配列要素の整数型と
  その倍の長さの整数型を使う
  - BDIGIT=16bit　BDIGIT_DBL=32bit　(現在はまず使われない)
  - BDIGIT=32bit　BDIGIT_DBL=64bit　(普通はこれ)
  - BDIGIT=64bit　BDIGIT_DBL=128bit　(New)
  - BDIGIT=64bit　BDIGIT_DBL=64bit　(昔のCray)
- Bignum のリファクタリング
  中間オブジェクト生成の除去
- 基数変換 (10進→2進、文字列読み込み) で
  分割統治法を利用

ベンチマーク環境

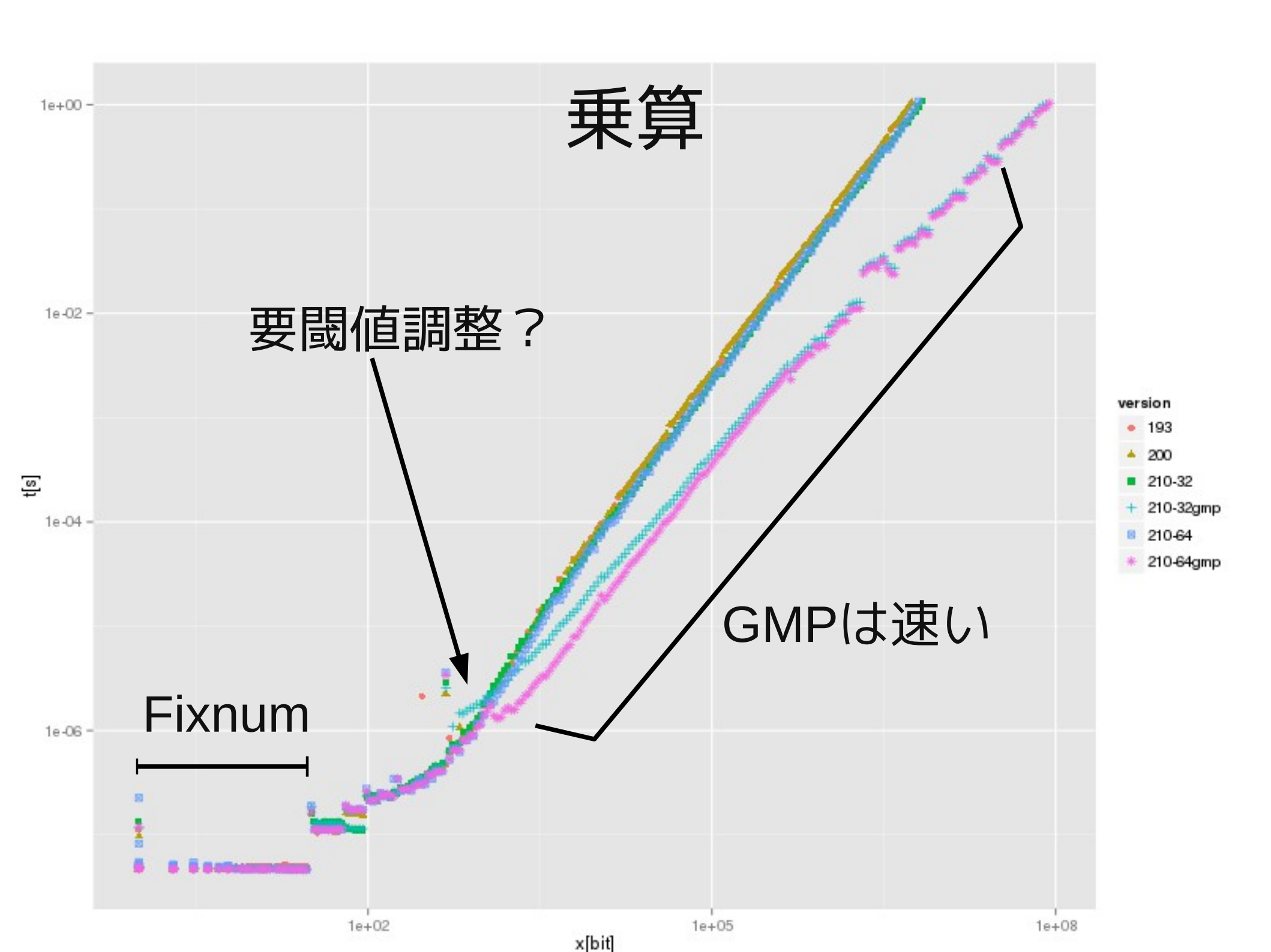
- Core i7-3517U 1.90GHz
- Debian GNU/Linux 7.1
  - gcc 4.7.2
  - gmp 5.0.5
- ruby versions
  - 1.9.3
  - 2.0.0
  - 2.1.0
    - BDIGIT: 32bit, 64bit
    - without GMP, with GMP

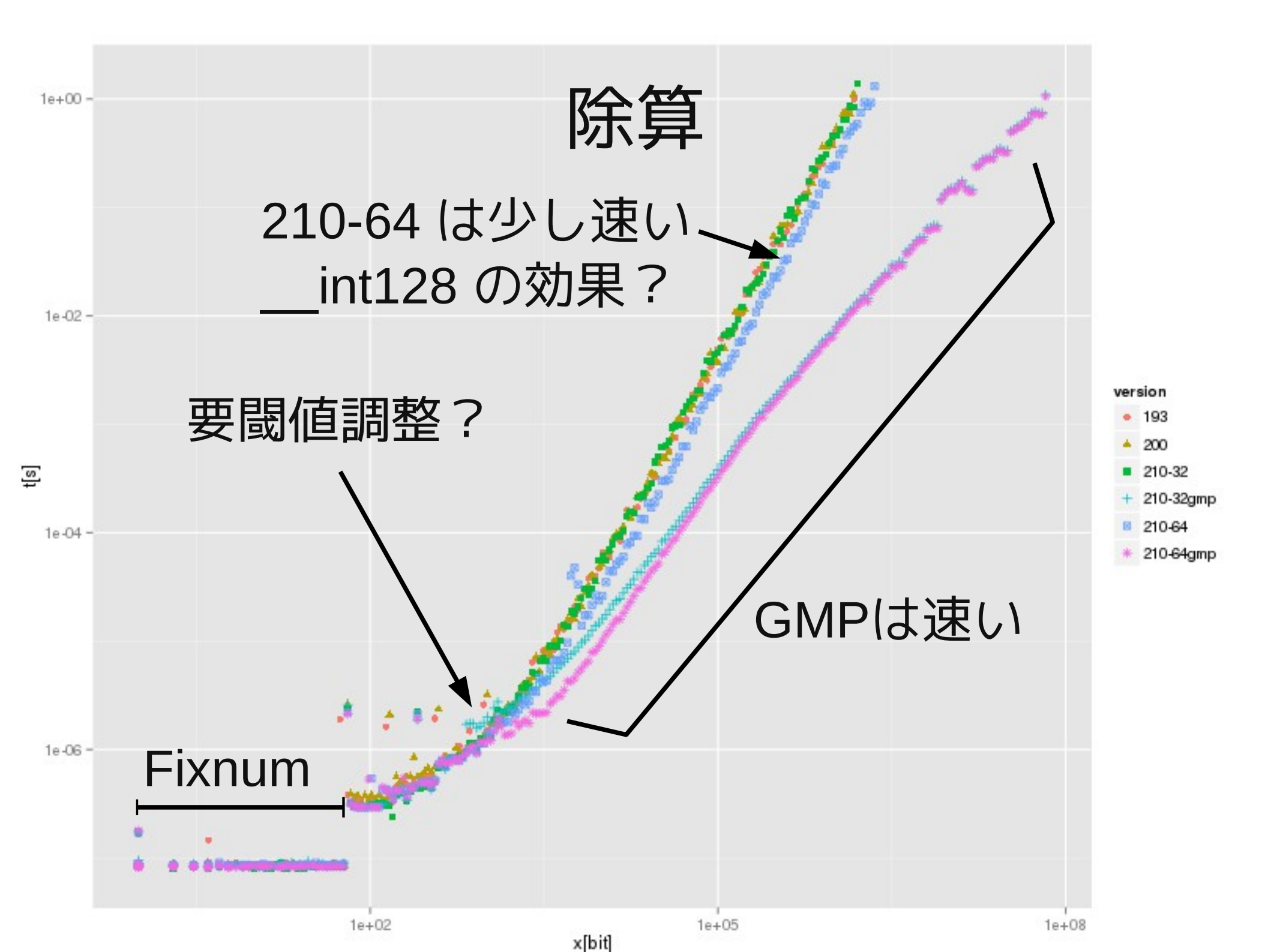
2項演算の傾向

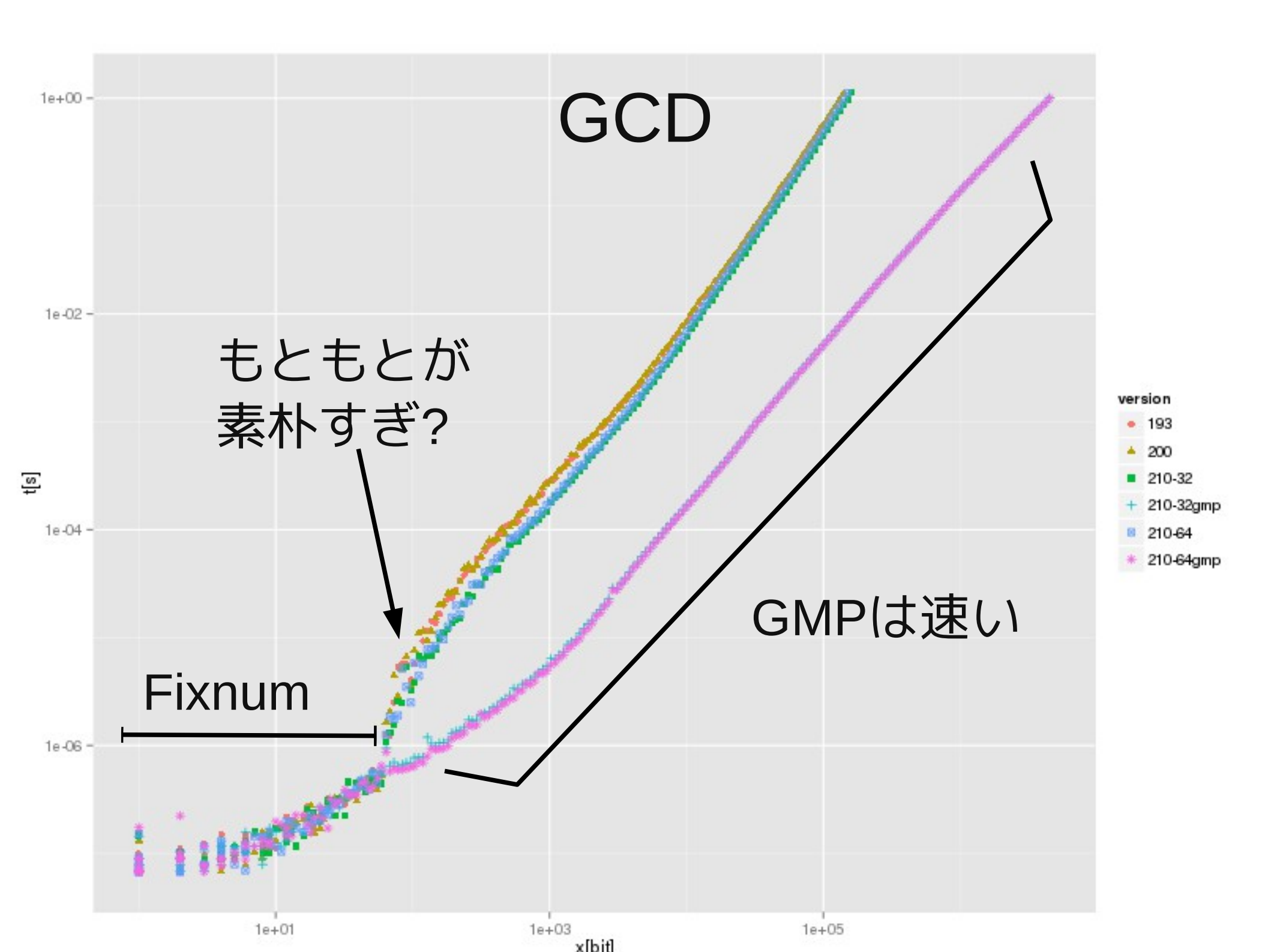

193
200
210-32
210-32gmp
210-64
210-64gmp
mul
div
gcd
y[bit]
x[bit]
1e+02
1e+05
1e+08

ベンチマーク

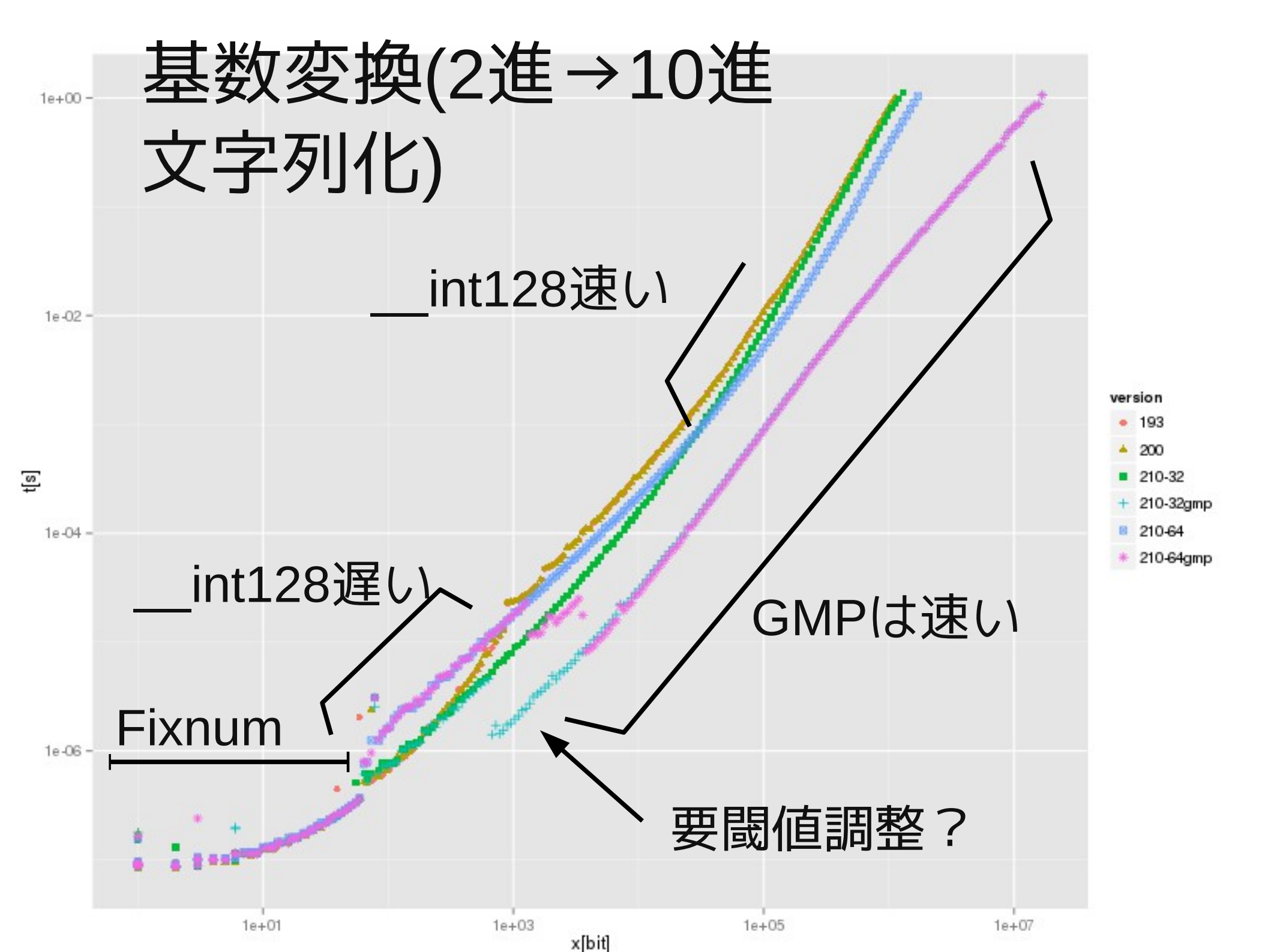
- n ビットの数を生成
  m = 1 << (n-1)
  m | rand(m)
- 2項演算についてはもうひとつ別の数を生成
  除算は半分のビット数、他は同じビット数
- 演算して時間を測る
  clock_gettime(CLOCK_THREAD_CPUTIME_ID)
- 1秒くらい、あるいは 100回繰り返して時間を測る


乗算
要閾値調整？
GMPは速い
Fixnum
1e+00
1e-02
1e-04
1e-06
t[s]
1e+02
1e+05
1e+08
x[bit]
version
193
200
210-32
210-32gmp
210-64
210-64gmp

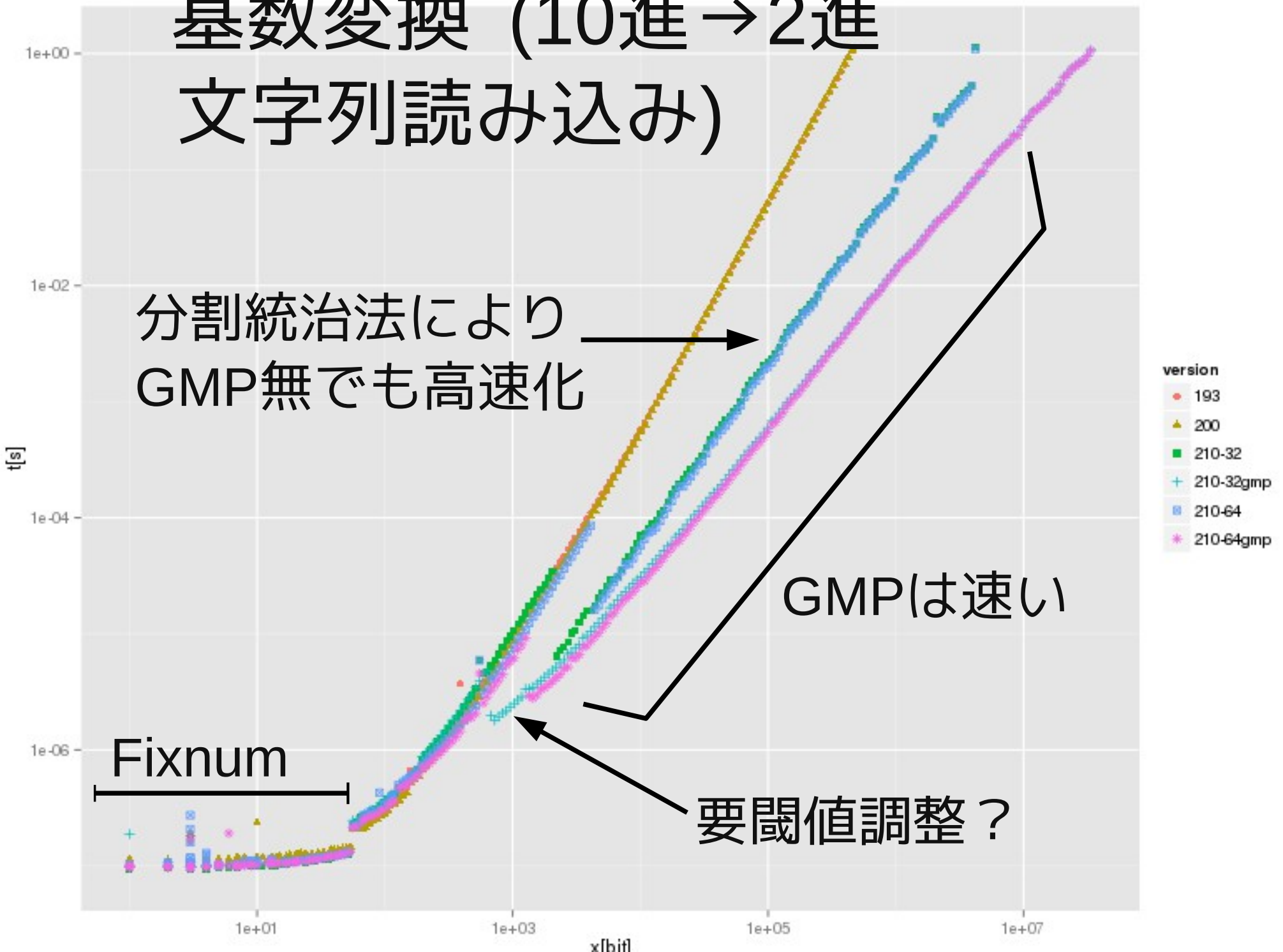

除算
210-64 は少し速い
__int128 の効果？
要閾値調整？
GMPは速い
Fixnum
1e+00
1e-02
1e-04
1e-06
t[s]
1e+02
1e+05
1e+08
x[bit]
version
193
200
210-32
210-32gmp
210-64
210-64gmp


GCD
もともとが
素朴すぎ?
GMPは速い
Fixnum
1e+00
1e-02
1e-04
1e-06
t[s]
1e+01
1e+03
1e+05
x[bit]
version
193
200
210-32
210-32gmp
210-64
210-64gmp

基数変換(2進→10進文字列化)



基数変換 (10進→2進 文字列読み込み)



技術的まとめ

- 大きな数について GMP でいくつかの演算が高速化
  乗算、除算、GCD、基数変換
- もうすこし調整するほうが望ましい

Ruby と GMP および自由ソフトウェア運動

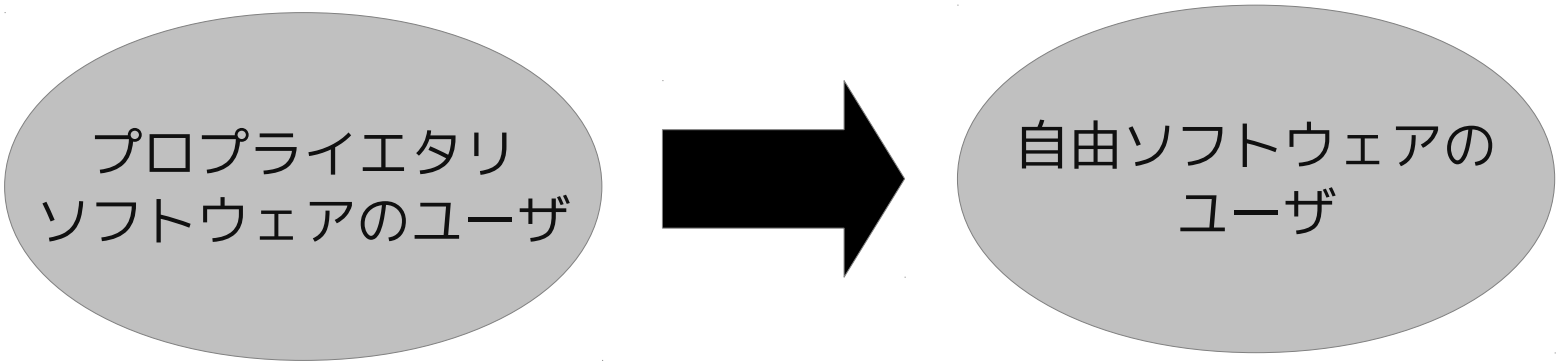
- Ruby は BSD (と独自のデュアルライセンス)
- GMP は LGPLv3 or later
- 高速化と LGPLv3 の抱き合わせ
- GMP を組み込めるようにしたことの影響は?

自由ソフトウェア運動

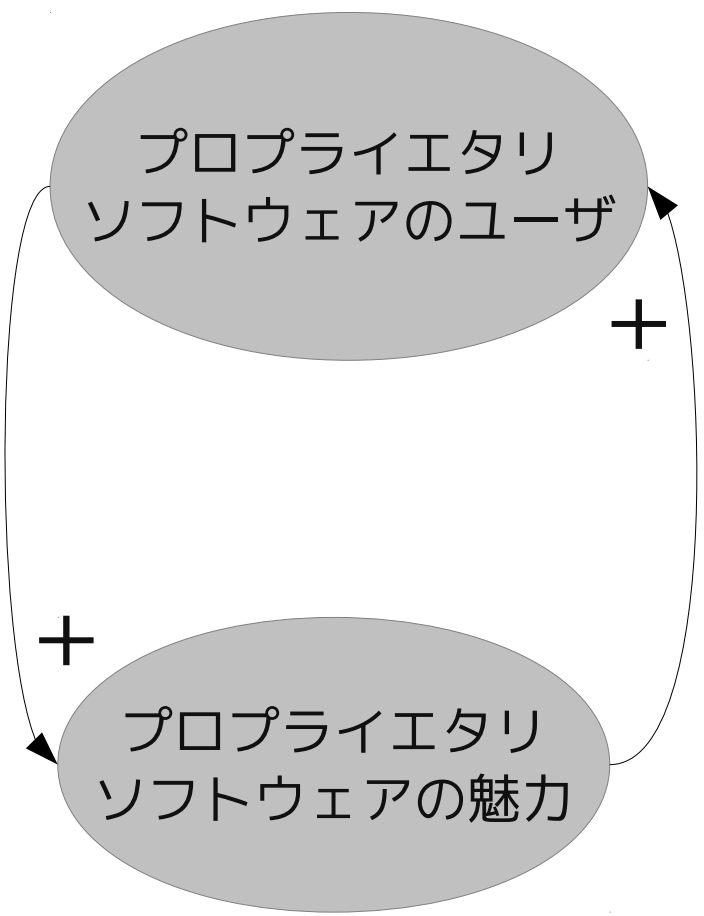
ソフトウェアに関する自由を実現する運動

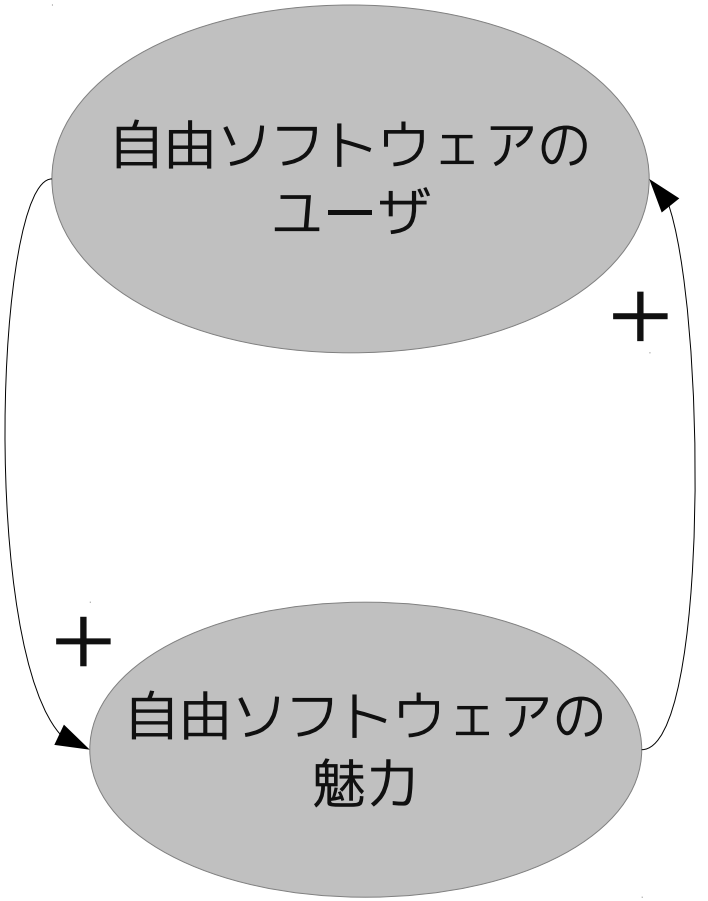
- いかなる目的に対しても、プログラムを実行する自由 (第零の自由)。
- プログラムがどのように動作しているか研究し、必要に応じて改造する自由 (第一の自由)。ソースコードへのアクセスは、この前提条件となります。
- 身近な人を助けられるよう、コピーを再配布する自由 (第二の自由)。
- 改変した版を他に配布する自由 (第三の自由)。これにより、変更がコミュニティ全体にとって利益となる機会を提供できます。ソースコードへのアクセスは、この前提条件となります。

ユーザの移行を促す



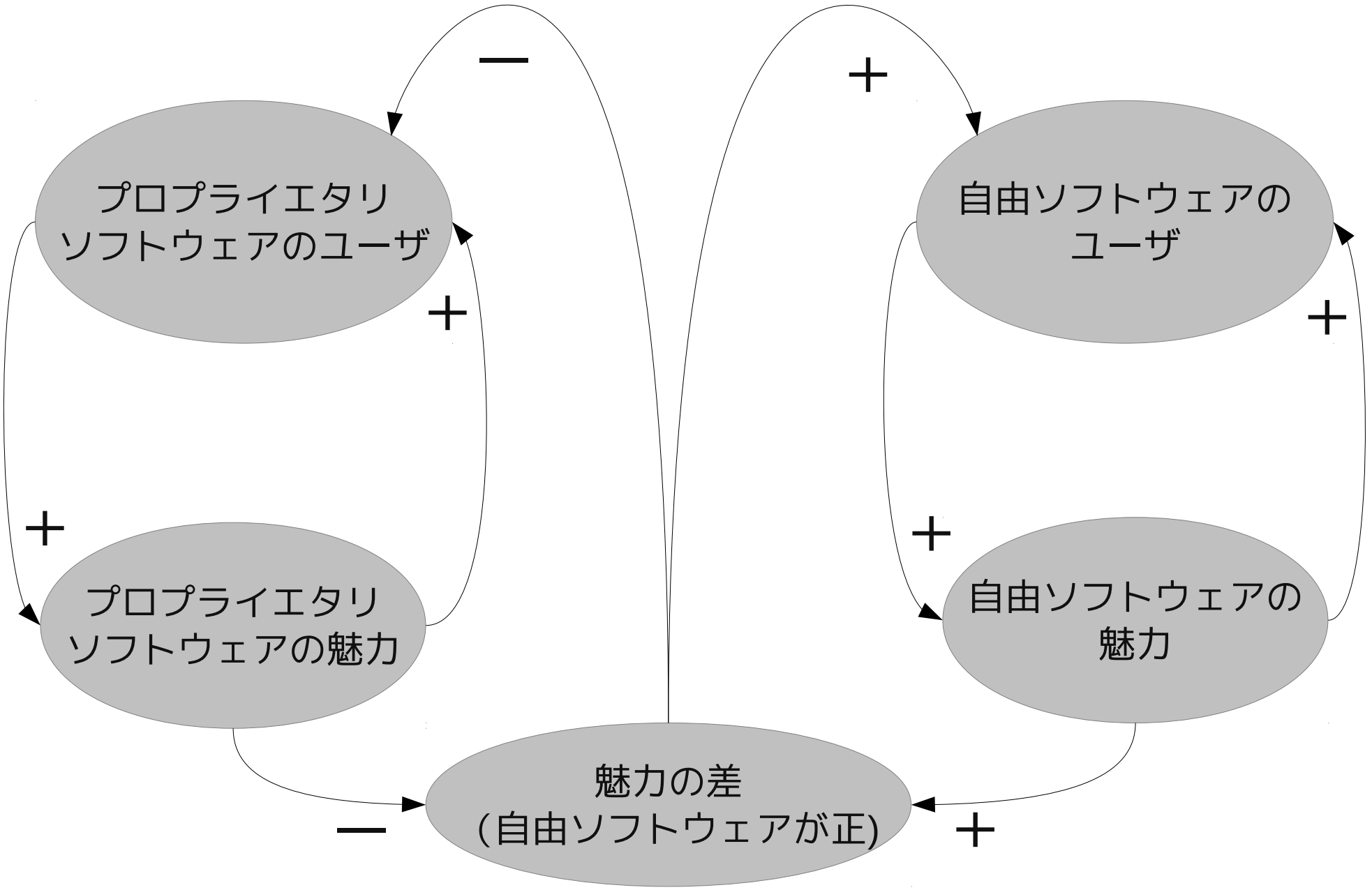
ユーザ数と魅力の拡張ループ



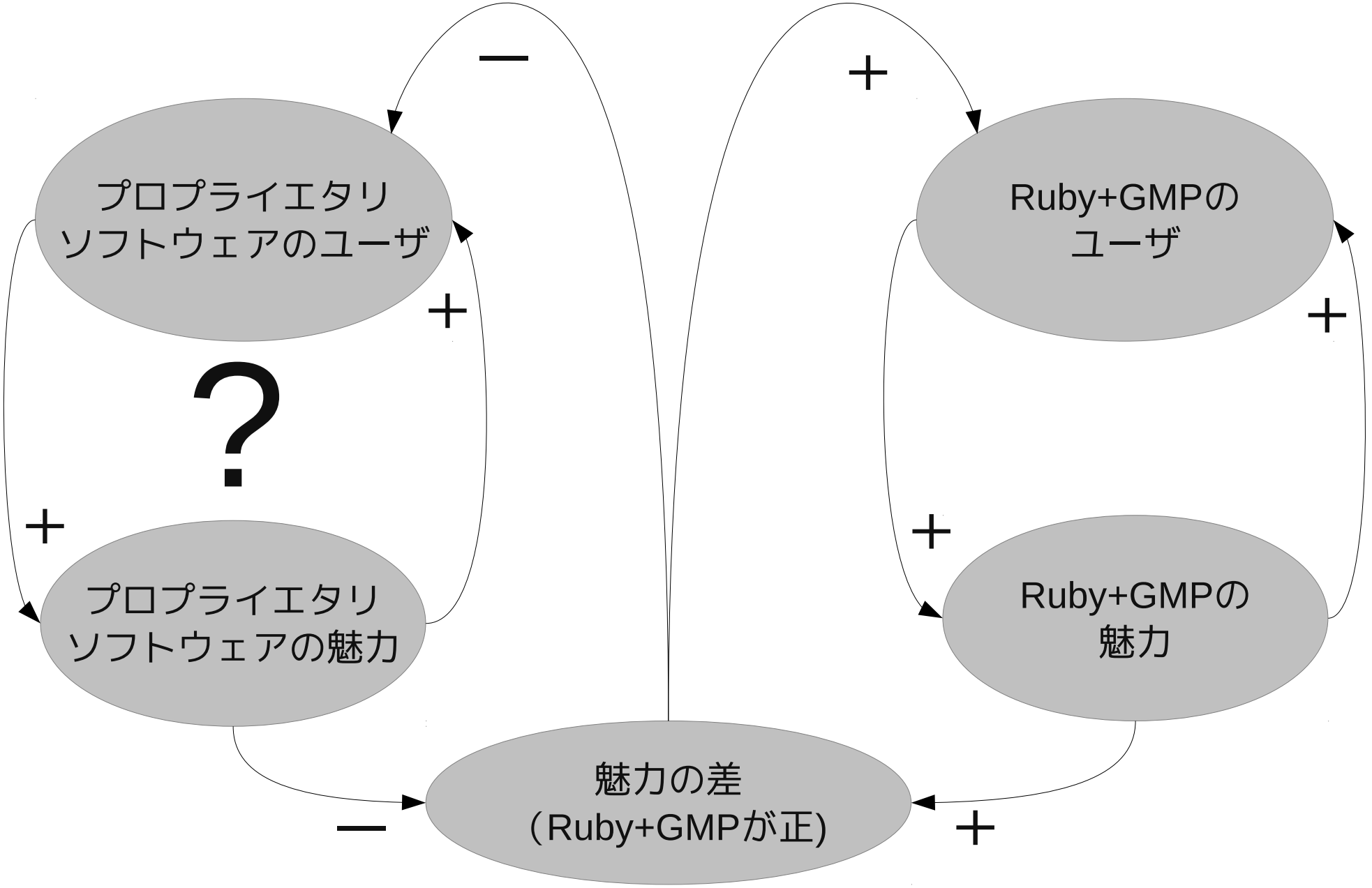


正帰還: どんどん増えていくか、どんどん減っていく

ユーザの移行


プロプライエタリ
ソフトウェアのユーザ
自由ソフトウェアの
ユーザ
プロプライエタリ
ソフトウェアの魅力
自由ソフトウェアの
魅力
魅力の差
(自由ソフトウェアが正)
−
+
+
+
+
+
−
+

Ruby+GMPへの適用



プロプライエタリソフトウェア?

- 「多倍長演算が高速なRuby」がユーザを獲うターゲットとなるプロプライエタリソフトウェア?
  あまり思い当たらない感じ?
  Java, C#, Visual Basic あたり?
- 自由ソフトウェアの魅力を増やしているという点はある
  他に移りにくくなるかも?

自由ソフトウェア運動まとめ

- 自由ソフトウェア運動としてはあまり効果的な領域でなかった?

参考文献

- システム・シンキング入門、西村行功、2004, 日経文庫