合同分科会
合同分科会 2011 年度会合 より

基調講演

セルフデベロップメントとプログラミング

IIJ 技術研究所

和田 英一

サイエンティフィック・システム研究会 2011 年 10 月 20 日

セルフデベロップメントとプログラミング

和田英一

IIJ イノベーションインスティテュート

講義を聞いたり本を読んだりしても, その内容がすぐに理解できるわけではない. 思い返し, 考え直し, 理解したいという努力の継続の結果, 少しずつ身につく. ところで, その理解の努力を強力に援助してくれるのが, 計算機である. 計算機のプログラムを書き, 実験を繰り返すと, それまで中途半端だった理解が急に深まることが少なくない.

手で繰り返す実験はたかが知れているが, 計算機を使えば膨大な処理がたちどこに出来, 多くのケースについて知見が得られる. そしてそれぞれのテーマが興味深いことを知る.

そういう面白い例のいくつかを紹介したい.

Self Development, Programming, Numerical Experiment, Puzzle, Computer Simulation, Graphic Languages

はじめに

富士通研究所初代社長の尾見半左右氏は 70 歳を過ぎてから学位論文「電子計算機の巨大化の限界に関する研究」を提出, 1973 年 9 月に東京工業大学から工学博士の学位を得られた. 論文を指導した東工大の川上正光教授から, その快挙に対し, 佐藤一斎の言志四録の「少にして学ばば即ち壮にして為すこと有り, 壮にして学ばば即ち老いて衰へず, 老いて学ばば即ち死して朽ちず」の言葉を贈られた.

いつまでも学ぶ態度を持ち続けるのが, セルフデベロップメント, 自己啓発であろう. 自己啓発は, 個人ごとに多くのやり方があって然るべきである. 私の場合, 仕事のプログラムは書くこともあるが, 自分のために書くことが殆んどである. 自分のため, すなわち自己啓発のためであって, 僭越ながら, 今日はそういう計算機活用の簡単で面白い例を紹介したい.

私がプログラミングに取り掛かるのは, 本を読んでいて, これは一体どういうことかという疑問を解こうとする時が多い. 特に翻訳をしている時は, 内容が分からないと訳せないわけだが, そういう場合に計算機が活躍する.

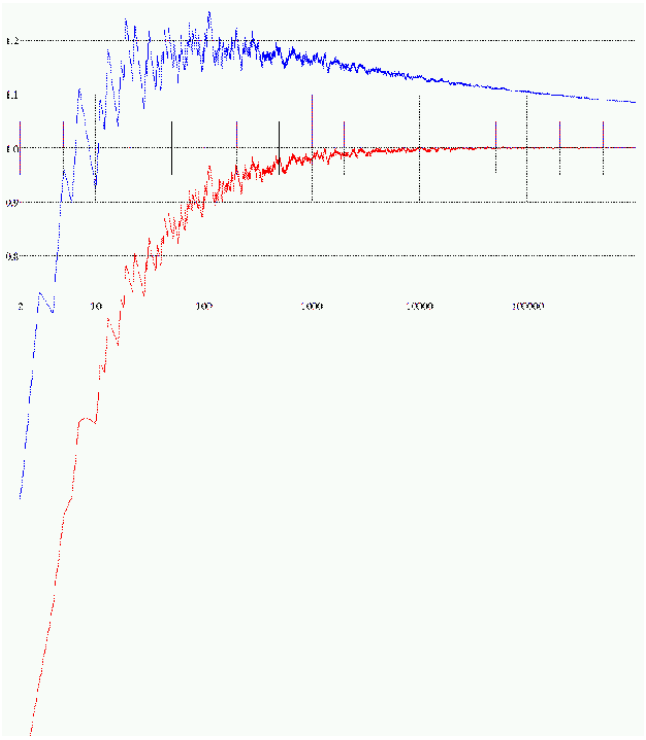
以下, あれこれやってみたのは, 元富士通社長 小林大祐氏の「ともかくやってみろ」精神のプログラミング版である.

Eratosthenes の篩

Eratosthenes の篩の話は, 子供の頃に読んだような気がするが, 大人になってからでも, やってみられるのは, せいぜい 100 くらいまでである. しかし, 素数定理を知ったとして, それを確かめるには, 1 万, 10 万とか 100 万くらいまで素数の個数を知る必要がある. 手元に計算機があれば, 篩は簡単に実行出来る.

100 万までの Eratosthenes の篩をやってみよう.

```
(define length 500000)                          ; 篩は奇数だけなので半分でよい.
(define sieve (make-bit-string length #t))      ; 篩 sieve を作る 最初全部ビットをたてる
(bit-string-clear! sieve 0)                     ;1 の場所をクリアする
(define k 0) (define b 10) (define c 1)

(define (loop)
 (set! k (bit-substring-find-next-set-bit sieve (+ k 1) length))
 (if k (let ((p (+ k k 1)))                     ;k の場所が 1 なら 2k+1 が素数
    (if (> p b) (begin (display (list b c)) (set! b (* b 10))))
    (set! c (+ c 1))                            ; カウントを増やす
    (do ((q (/ (- (* p p) 1) 2) (+ q p))) ((>= q length))
       (bit-string-clear! sieve q)) (loop))     ;(p*p-1)/2 から p 置きにビットをクリア
  c))

(loop)

(10 4)(100 25)(1000 168)(10000 1229)(100000 9592)
;Value: 78498
```



100 万までには, 78498 個の素数のあることが分かる.
Gauss の素数定理では (/ 1000000 (log 1000000)) なので計算してみると
⇒ 72382.41365054197
一方 Legendre の定理では
(/ 1000000 (- (log 1000000) 1.08366))
⇒ 78543.1776352779
この 2 つの素数定理を図にしたのが左だ.

Wilson の定理 ($(p-1)! \equiv -1 (\mathrm{mod}\, p)$ iff p is prime or 1, 従って $\frac{(x-1)!+1}{x}$ は, x が素数か 1 の時に限り整数) を読んだら, プログラムを書く.

```
(define (wilson x)
 (/ (+ (factorial (- x 1)) 1) x))

(map wilson (a2b 2 20))
 ;(a2b a b) は a から b-1 までのリスト
(1 1 7/4 5 121/6 103 5041/8 40321/9 362881/10 329891
 39916801/12 36846277 6227020801/14 87178291201/15
 1307674368001/16 1230752346353 355687428096001/18
 336967037143579)
```

Carmichael 数

Fermat のテストを騙す数 (合成数なのに, 素数の Fermat テストを通過する) があるという. さっそくやってみる. 下の表の最左端は 4 から 28 までの合成数 n で, その右に $1 \le a < n$ について, n を法として a^n を計算したものである. その値が丁度 a になるものは赤字で示す.

```
 4  1  0  1
 6  1  4  3  4  1
 8  1  0  1  0  1  0  1
 9  1  8  0  1  8  0  1  8
10  1  4  9  6  5  6  9  4  1
12  1  4  9  4  1  0  1  4  9  4  1
14  1  4  9  2 11  8  7  8 11  2  9  4  1
15  1  8 12  4  5  6 13  2  9 10 11  3  7 14
16  1  0  1  0  1  0  1  0  1  0  1  0  1  0  1
18  1 10  9 10  1  0  1 10  9 10  1  0  1 10  9 10  1
20  1 16  1 16  5 16  1 16  1  0  1 16  1 16  5 16  1 16  1
21  1  8  6  1 20  6  7  8 15 13  8  6 13 14 15  1 20 15 13 20
```

```
22  1  4  9 16  3 14  5 20 15 12 11 12 15 20  5 14  3 16  9  4  1
24  1 16  9 16  1  0  1 16  9 16  1  0  1 16  9 16  1  0  1 16  9 16  1
25  1  7 18 24  0  1  7 18 24  0  1  7 18 24  0  1  7 18 24  0  1  7 18 24
26  1  4  9 16 25 10 23 12  3 22 17 14 13 14 17 22  3 12 23 10 25 16  9  4  1
27  1 26  0  1 26  0  1 26  0  1 26  0  1 26  0  1 26  0  1 26  0  1 26  0  1 26
28  1 16 25  4  9  8 21  8  9  4 25 16  1  0  1 16 25  4  9  8 21  8  9  4 25 16  1
```

さて, Carmichael 数, 例えば $n = 561$ では, その段全体が赤字になるものである.

```
(map (lambda (a) (modulo (expt a 561) 561)) (a2b 1 561))
=> (1 2 3 4 5 6 7 8 9 10 ... 560)
```

面白い! Carmichael 数には, 1105, 1729, 2465, 2821, 6601,... などがあるらしい.

関数を使った分類ルーチン

Sussman さんたちの Struncture and Interpretation of Computer Programs を訳していた時であった. 普通なら左のように書くプログラムが右のように書いてあった.

```
(define (extract sequence)
  (if (null? sequence) (cons '() '())
    (let ((result (extract (cdr sequence)))
          (next-item (car sequence)))
      (let ((a (car result)) (b (cdr result)))
        (if (even? next-item)
           (cons (cons next-item a) b)
           (cons a (cons next-item b)))))))

(let ((result (extract '(3 1 4 1 5 9 2))))
  (display (car result)) (display (cdr result)))
```

```
(define (extract sequence receive)
 (if (null? sequence)
   (receive '() '())
   (extract (cdr sequence)
    (lambda (a b)
      (let ((next-item (car sequence)))
        (if (even? next-item)
          (receive (cons next-item a) b)
          (receive a (cons next-item b))))))))
(extract '(3 1 4 1 5 9 2)
 (lambda (a b) (display a) (display b)))
(4 2) (3 1 1 5 9)
```

これなどは, 実行してみなければ, どう動くのか分からない. extract が渡す関数の中で引数を出力するようにして走らせてみる.

```
(define (extract sequence receive)
 (if (null? sequence)
   (receive '() '())
   (extract (cdr sequence)
    (lambda (a b)
      (display a) (display b) (newline) ←この行挿入
      (let ((next-item (car sequence)))
        (if (even? next-item)
          (receive (cons next-item a) b)
          (receive a (cons next-item b))))))))
```

出力
```
()()
(2)()
(2)(9)
(2)(5 9)
(2)(1 5 9)
(4 2)(1 5 9)
(4 2)(1 1 5 9)
(4 2)(3 1 1 5 9)
```

出力を見ると, どう動いたかが判明する. つまり, 関数 extract は, 順次に短くなるリストとある関数を引数として, 繰り返し呼び出される. 最後に空リストで呼び出されると, 渡された関数の引数 a と b に空リストを渡して呼び出す.

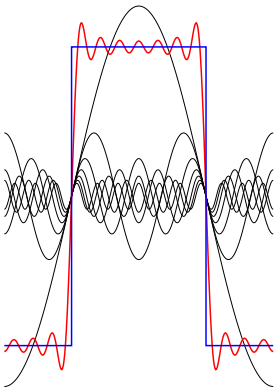
すると, 呼び出された関数 (1 段外で渡した引数の関数) が, 2 つの空リストを引数として呼ばれたので, その本体が実行され, 自分が渡されたリストの先頭が偶数か奇数かにより, 次に渡された関数を呼び出すための引数を用意し, 呼び出す.

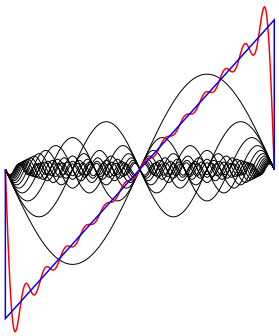
最後は, 最初に extract を呼び出した時に渡した関数が呼び出され, その引数が本体内で使えることになる. 巧妙な関数呼出し方である.

Fourier 変換

大学 3 年の時, 坪井忠二先生の演習で, 周期関数の Fourier 変換を計算する課題があった. 学部学生は手回し計算機も使わせて貰えず, 三角関数との掛算は計算尺, 合計には算盤を使った.

しかし, パラメトロン計算機が自由に使えるようになると, Fourier 変換は何でもなかった. 本に書いてあるように分解出来, 合成出来た.





f(0)	cos(6π/N)
f(1)	cos(2π/N)
f(2)	cos(4π/N)
f(3)	cos(6π/N)
f(m)	cos(2mπ/N)
f(N-1)	cos(2(N-1)π/N)

上の図で左はステップ関数, 中は鋸波関数. どちらも青線が元の関数, 黒が各周期の三角関数, 赤が合成した結果である.

Fourier 変換は大体次のようなプログラムを書くであろう.

```
(define nn 64)

(define (a n)
 (let ((an 0))
  (do ((m 0 (+ m 1))) ((= m nn))
    (set! an (+ (* (f m) (cos (/ (* 2 pi (+ (* m n) 0.5)) nn))) an)))
   (/ an nn)))
```

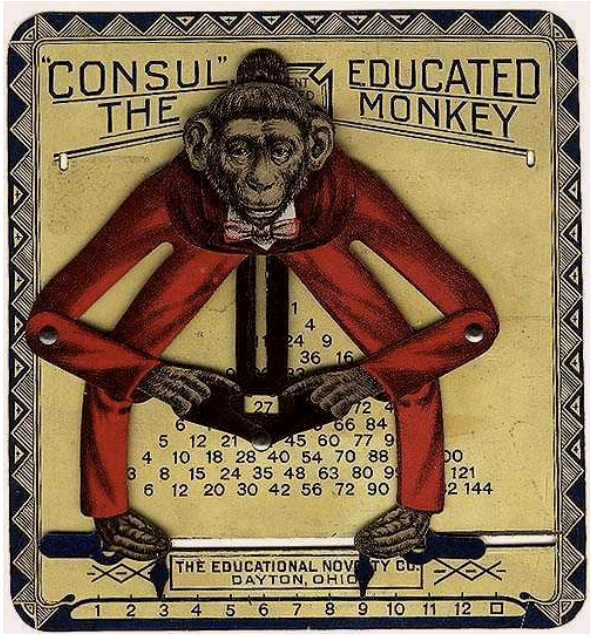
しかし, 演習の方法は違って, 上の図の右にあるような周期関数の表と, 三角関数の表をまず用意する. そして対応に注意しながら計算を続けるのであった.

最近は計算機が高速なので, 三角関数を何度計算しても痛痒を感じないが, ちゃんとしたプログラムを書くなら, 考えなければならない.

```
(define (a n)
 (/ (apply + (map (lambda (m) (* (vector-ref cs (modulo (* n m) 64))
  (vector-ref fs m))) (a2b 0 64))) 64))
```

高橋秀俊先生のお奨めは, 三角関数の表も加法定理で作るのだが, これは 1 回きりなので, そう凝ることもあるまい.

学者猿コンサル

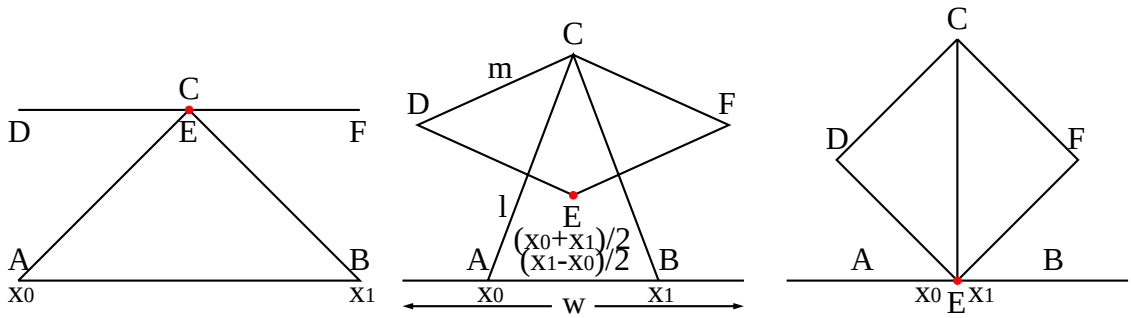


Consul, the Educated Monkey という計算関連の教育玩具がある. ウェブページで調べると, 1916 年頃にアメリカで売り出されたものらしい.

使い方はこうだ. 猿の両足を, 図のように下の目盛の 3 と 9 に合わせると, 猿の手が示す枠の中に積の 27 が現れるのである. 目盛の右端の 12 の右には四角が見えるが, 片足をそこに合わせると, 他方の足の二乗が現れる.

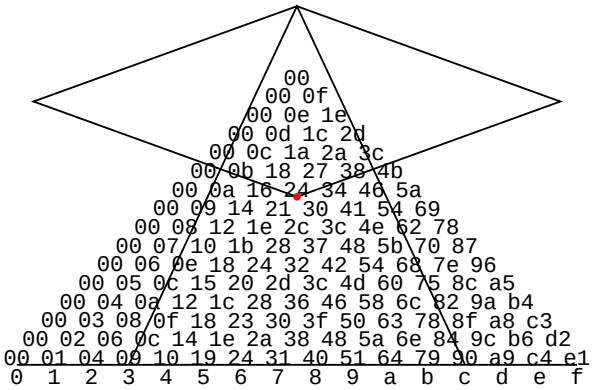
これは乗算表であるが, この表は差し換えることが出来, 加算にも使える.

次の図でその原理を示す. まず中の図で, 猿の足の位置が A, B(座標は x_0, x_1) とする. 足の動く範囲を w とする. 猿の頭が C で, AC の長さを l, C から肘 D までの長さを m とし, 頭と肘と解の位置関係は菱形とする. また∠ ACD は固定で, 今は 45° にし. たまた $l = w/\sqrt{2}$, $m = l/\sqrt{2}$ とする. そうすると, 解の点 E の座標は $(x_0 + x_1)/2, (x_1 - x_0)/2$ となる.



左は足を一杯に開いた時, 右は足を揃えた時の様子を示す.

右の図は, 上のパターンで十六進法の乗算表を作ってみたところである. 3 に c(つまり 12) を掛けると 24(十進で 36) になることを示す. x_1 が x_0 より左へ来ると, 解の位置は基線より下になる.

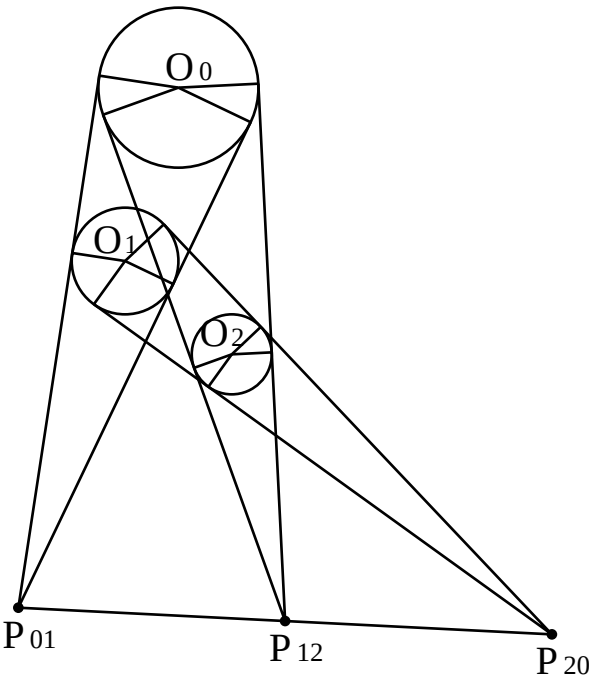


三つの円

夏の始め頃, 「とっておきのパズル」という本を送ってくれた人がいる. 面白く, しかも難しい問題が山のように書いてある本だ. うっかりはまると大変なので, なるべく近寄らないようにしてはいるが, 誘惑も絶ちがたく, ときどき, ちらと眺めては考え始めたりする.

そこに三つの円という幾何学の問題があった. 問題は簡単だが, どこから取り掛かったらよいかわからない. こういうパズルだ.

平面上の 2 つの円の共通接線の交点を, それらの円の焦点という. 3 つの円があると, 焦点は 3 個出来るわけだが, その 3 点は一直線上にあることを証明せよというのである.

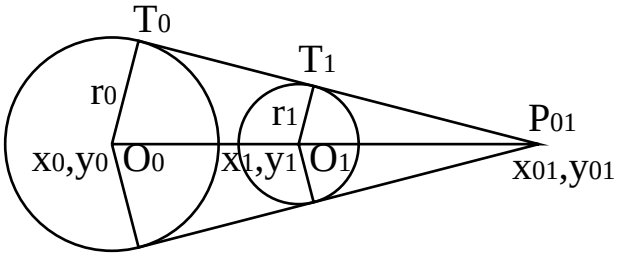


手元に計算機があるから, とりあえず強引にやってみることにした. 図では確かにそうなる.

下の図の円 O_0 は中心の座標が x_0, y_0, 半径が r_0, 円 O_1 は座標が x_1, y_1, 半径が r_1 であるとする. また共通接線の 1 本と, 円との接点を T_0, T_1, 共通接線の交点を P_{01} とする. するとその座標 x_{01}, y_{01} が得られる. 同様にして

$x_{01} = (r_0x_1 - r_1x_0)/(r_0 - r_1)$, $y_{01} = (r_0y_1 - r_1y_0)/(r_0 - r_1)$
$x_{12} = (r_1x_2 - r_2x_1)/(r_1 - r_2)$, $y_{12} = (r_1y_2 - r_2y_1)/(r_1 - r_2)$
$x_{20} = (r_2x_0 - r_0x_2)/(r_2 - r_0)$, $y_{20} = (r_2y_0 - r_0y_2)/(r_2 - r_0)$

これから, この三角形の面積を計算すればよい. 3 点が一直線上にあれば, 面積は 0 になる. 私は risa/asir で計算した.



プログラムを調べるためのシミュレータやトレーサ

昔の計算機で使っていた面白そうなプログラムに出会うことがある. アセンブリ言語や機械語で書いてあると, どのように動くのか, 理解しにくい. 昔でなくても現今でも TAOCP の MMIX 言語を見ると, 途方に暮れることがある.

その時, 役に立つのが, 元の計算機のシミュレータである. 昔の計算機はアーキテクチャが簡単だから, シミュレータを書くのは何でもない.

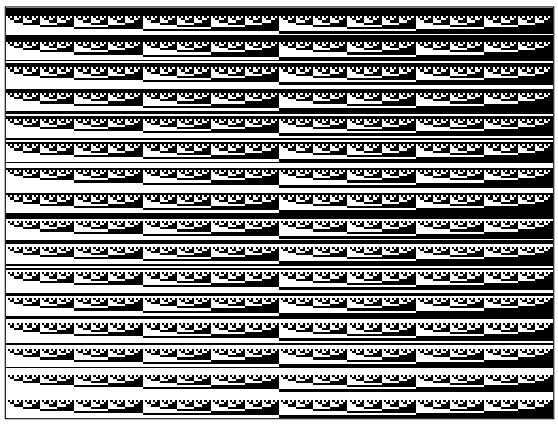
2 つの例を紹介したい. 慶応義塾大学名誉教授の大駒誠一氏は, 古い計算機のソフトウェア的復刻と称し, いくつものシミュレータを書いて, 昔のプログラムをシミュレートしている.

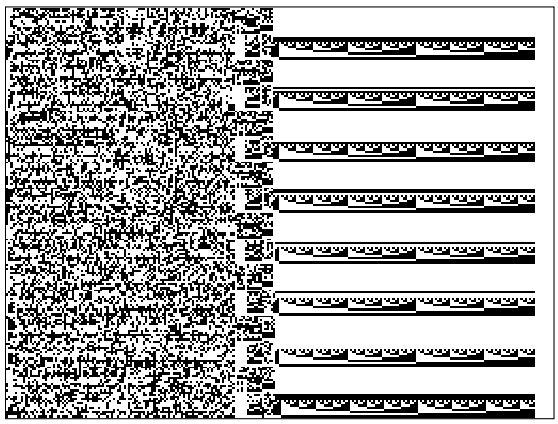
その大駒氏が, 私が 1958 年に書いた, パラメトロン計算機 PC-1 のイニシアルオーダーを解読しようとして, PC-1 のシミュレータを書き, イニシアルオーダーの機能が完全に理解できたといわれた. そこまですれば, そうであろうと思う.

私の場合は, DEC(Digital Equipment Corporation) のミニコン, PDP-8 のシミュレータである. ミニコンだけあって, 書くのは簡単であった.

私はそのシミュレータの上で, オランダの van der Poel 先生の作られた Lisp を走らせるのが目的であった. 通常のシミュレータだと, プログラムカウンタ, レジスタの内容が順次に出力されるが, このシミュレータでは, リスト処理のゴミ集め (Garbage Collection) を視覚化する.

ゴミ集めの視覚化は, 遥か昔, 慶応義塾大学の中西さんが, Apple コンピュータでやって見せたことがあり, そんなことがやってみたかったのである.





この図のそれぞれは, 1 語 12 ビットで 4096 語のメモリーを, 縦 192 ビット, 横 256 ビットで示す. 192 ビットは 12 ビットの語が下から 0 番地, 1 番地, ..., 15 番地と 16 語並び, そういう 16 語が, 256 段あって, 都合 4096 語になっている.

左は 0 番地には 0, 1 番地には 1, ..., 4095 番地には 4095 を格納したところである. 右は左半分が Lisp のプログラム, 右半分がヒープ領域で, ヒープ領域の各セルは, 2n 番地と 2n+1 番地の 2 語からなり, 奇数番地の car 部は 0, 偶数番地の cdr 部には, 次の自由セルの番地が入っている. Lisp のプログラムを入力するとちゃんと動く.

実際に動く様子は, この話の当日にお目にかける.

参考文献

佐藤一斎 川上正光注, 言志晩録 60, 言志四録 (三) 講談社学術文庫 276, 1980.
高橋秀俊, フーリエ変換よもやま話, 数理と現象, 岩波書店, 1975.
ピーター・ウィンクラー 坂井公他訳, とっておきの数学パズル, 日本評論社, 2011.

セルフデベロップメントとプログラミング

和田英一
IIJ イノベーションインスティテュート
技術研究所

尾見半左右氏(富士通研究所初代所長 1901-1985)
の情報処理学会のコンピュータ博物館パイオニアのページ
http://museum.ipsj.or.jp/pioneer/omi.html

少にして学ばば即ち壮にして為すこと有り,
壮にして学ばば即ち老いて衰へず,
老いて学ばば即ち死して朽ちず　　(佐藤一斎 言志四録)

少而学 即壮而有為
壮而学 即老而不衰
老而学 即死而不朽

パソコンが手元にあると, 面白そうな話題が試せる.

Knuth の The Art of Computer Programming を読み, 訳し,
多くのアルゴリズムと問題をテストした.

お詫び

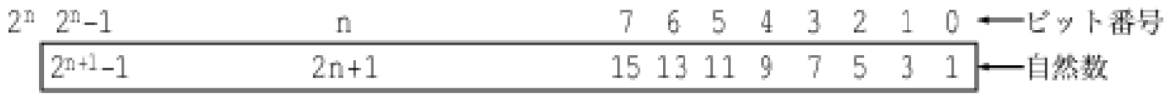
以下のプログラムはLisp言語の1つ, Schemeで書いてある.
私はプログラムのほとんどをLispとPostScriptでしか書かないから
なので, 済みません.

The Art of Computer Programmingは, Stanford大学の
Donald Knuthが1968年から書き続けているアルゴリズムの大著で,
この春4A巻が出た.

そのこだわりから, アルゴリズムは自然言語で書いていて,
曖昧なところがあり, 多くのことを書こうとするあまり,
説明をはしょっていて, 実装して走らせてみてやっと理解出来たりする.

Eratosthenes の篩

昔から知られている素数表の作り方

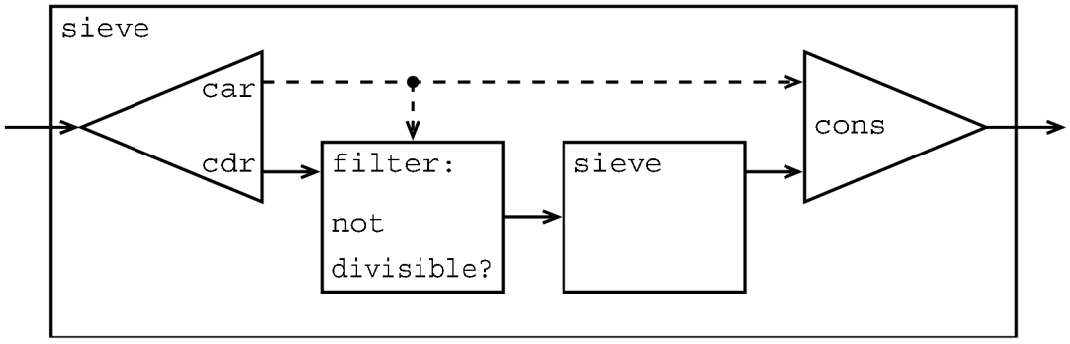


計算機で Eratosthenes の篩を計算するには, 0 から $2^n - 1$ の, 長さ 2^n で, 各ビットが 1 のビット列を用意し, 1 から $2^{n+1} - 1$ の奇数に対応させる.
1 は素数ではないので, 1 に対応するビットを 0 にする.
番号の小さい方から 1 のビットを探し, そのビット番号が n なら $p = 2n + 1$ が次の素数. そこから p 置きにビットを 0 にしていく.
p^2 より小さいビットは, $< p$ の素数で篩われているから, p^2 に対応するビット位置 $(p^2 - 1)/2$ 番目から p 置きに篩う.

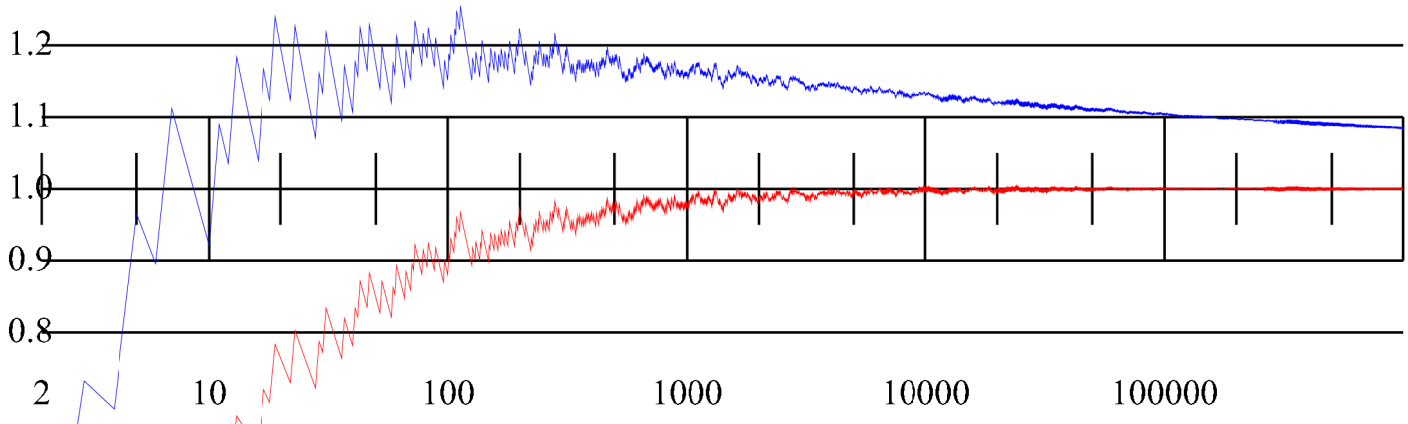
手作業では100くらいまで．パソコンでは，100万くらいまで出来る．

```
(define length 524288)  ;2^19
(define sieve (make-bit-string length #t))  ;篩を作る
(bit-string-clear! sieve 0)       ;1の場所をクリアする
(define k 0)
(define (next k)
 (let ((n (bit-substring-find-next-set-bit sieve
        (+ k 1) length)))
  (if n (let ((p (+ n n 1))) ;nの場所が1なら2n+1が素数
   (do ((q (/ (- (* p p) 1) 2) (+ q p))) ((>= q length))
    (bit-string-clear! sieve q)) (next n))
 'ok)))       ;(p*p-1)/2からp置きにクリア
(next 0)      ;起動
```

上の篩の実行に，MacBook Airで2.8秒くらいかかる．



ストリームの信号処理系の入れ子によるEratosthenesの篩
一番外の処理系には左から2,3,4,5,...が入る
右から2,3,5,7,11,..が出る



100万までの素数表があると, Gaussと Legendreの素数定理の図を描くことが出来る.

前のページの図は $\pi(n)/(n/\ln(n))$ をプロットしてある.
Gaussの式の値は

2の時 $1/(2/\ln(2))$=.34657359027997264
3の時 $2/(3/\ln(3))$=.73240819244540 64
4の時 $2/(4/\ln(4))$=.69314718055994 53
5の時 $3/(5/\ln(5))$=.96566274746046 01
6の時 $3/(6/\ln(6))$=.89587973461402 75
...
1000000の時 $78498/(1000000/\ln(1000000))$=1.08448994777907 95

Wilson の定理

$((x-1)!+1)/x$ は, x が素数か 1 の時に限り整数

Scheme には bignum があるのでやってみる.

```
(define (wilson x)
 (/ (+ (factorial (- x 1)) 1) x))
        ;Scheme の整数除算は整除できないと分数にする
(map wilson (a2b 2 20))
 ;(a2b a b) は a から b-1 までのリスト
(1 1 7/4 5 121/6 103 5041/8 40321/9 362881/10 329891
 39916801/12 36846277 6227020801/14 87178291201/15
 1307674368001/16 1230752346353 355687428096001/18
 336967037143579)
```


Carmichael 数 Fermat のテスト 素数 p で $a < p$ について $a^p \bmod p = p$.

Carmichael 数: 合成数なのに, Fermat テストを通過する

```
 4  1  0  1
 6  1  4  3  4  1
 8  1  0  1  0  1  0  1
 9  1  8  0  1  8  0  1  8
10  1  4  9  6  5  6  9  4  1
12  1  4  9  4  1  0  1  4  9  4  1
14  1  4  9  2 11  8  7  8 11  2  9  4  1
15  1  8 12  4  5  6 13  2  9 10 11  3  7 14
16  1  0  1  0  1  0  1  0  1  0  1  0  1  0  1
18  1 10  9 10  1  0  1 10  9 10  1  0  1 10  9 10  1
20  1 16  1 16  5 16  1 16  1  0  1 16  1 16  5 16  1 16  1
21  1  8  6  1 20  6  7  8 15 13  8  6 13 14 15  1 20 15 13 20
22  1  4  9 16  3 14  5 20 15 12 11 12 15 20  5 14  3 16  9  4  1
24  1 16  9 16  1  0  1 16  9 16  1  0  1 16  9 16  1  0  1 16  9 16  1
25  1  7 18 24  0  1  7 18 24  0  1  7 18 24  0  1  7 18 24  0  1  7 18 24
26  1  4  9 16 25 10 23 12  3 22 17 14 13 14 17 22  3 12 23 10 25 16  9  4  1
27  1 26  0  1 26  0  1 26  0  1 26  0  1 26  0  1 26  0  1 26  0  1 26  0  1 26
28  1 16 25  4  9  8 21  8  9  4 25 16  1  0  1 16 25  4  9  8 21  8  9  4 25 16  1
```

前ページの表は下のような latex の原稿で構成した

```
\verb+ 4 +\color{red}\verb+ 1+\color{black}\verb+  0  1 +\\
\verb+ 6 +\color{red}\verb+ 1+\color{black}\verb+  4 +
\color{red}\verb+ 3+\color{black}\verb+ +\color{red}\verb+ 4+
\color{black}\verb+  1 +\\
\verb+ 8 +\color{red}\verb+ 1+\color{black}\verb+  0  1  0  1  0  1 +\\
```

この原稿は次のページの Scheme のプログラムで作る.


```
(define (fermattest n a)
 (modulo (expt a (- n 0)) n))    ;a^(n-1)mod nはもう1つのFermatテスト

(define (disp2 n)
 (if (< n 10) (display " ")) (display n))

(define (dispred n)
 (display "+\\color\{red\}\\verb+")
 (disp2 n)
 (display "+\\color\{black\}\\verb+"))

(define (foo n)
 (if (> (length (factorize n)) 1)
  (begin (display "\\verb+")
   (disp2 n) (display " ")
   (do ((a 1 (+ a 1))) ((= a n))
    (let ((f (fermattest n a)))
     (if (= f a) (dispred f) (disp2 f)) (display " ")))
   (display "+\\\\") (newline))))

(do ((n 2 (+ n 1))) ((= n 30)) (foo n))
```

```
(map (lambda (a) (modulo (expt a 561) 561)) (a2b 1 561))
=> (1 2 3 4 5 6 7 8 9 10 ... 560)
```

Carmichael数には, 1105, 1729, 2465, 2821, 6601, ...
などがあるらしい.

このうち, 1729 は別のことで有名

Hardy の Ramanujan 追悼文 (Hardy 1921) から引用すると「『すべての正の整数は彼の友人であった』といったのは Littlewood だった (と思う.) 彼がパトニーで療養していた時, 見舞いにいったのを覚えている. 1729 番というタクシーに乗ったので, その数には何の面白味もないといい, それが悪い予兆でないことを望むといった. 『いや』彼は答えた『それは非常に興味のある数だ. それは二通りで二つの立方数の和で表現出来る最小の数だ』」.

$9^3 + 10^3 = 1^3 + 12^3 = 1729 \qquad 9^3 = 729, \quad 12^3 = 1728$

$x^2 + x + 41$ **の式と同じように気になる数である.**

Fourier 変換

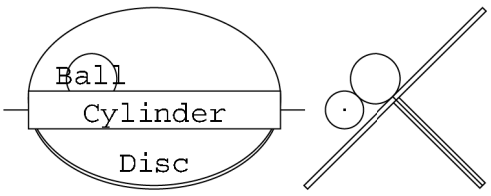
2π を周期とする周期関数 f を

$f(x) = a_0 + \Sigma_{n=1}^{N} a_n \cos nx + \Sigma_{n=1}^{N} b_n \sin nx$

のように sin や cos の和と表現したい時, 係数 a_n, b_n を計算する.

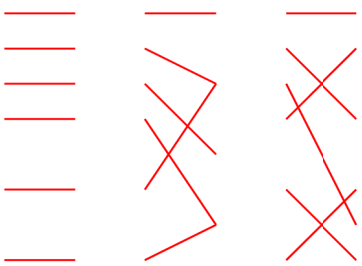
$$a_n = \frac{1}{N} \Sigma_m f\left(\frac{m\pi}{N}\right) \cos \frac{mn\pi}{N}$$

$$b_n = \frac{1}{N} \Sigma_m f\left(\frac{m\pi}{N}\right) \sin \frac{mn\pi}{N}$$



人力計算の場合

f(0)	cos(0π/N)
f(1)	cos(2π/N)
f(2)	cos(4π/N)
f(3)	cos(6π/N)
f(m)	cos(2mπ/N)
f(N-1)	cos(2(N-1)π/N)

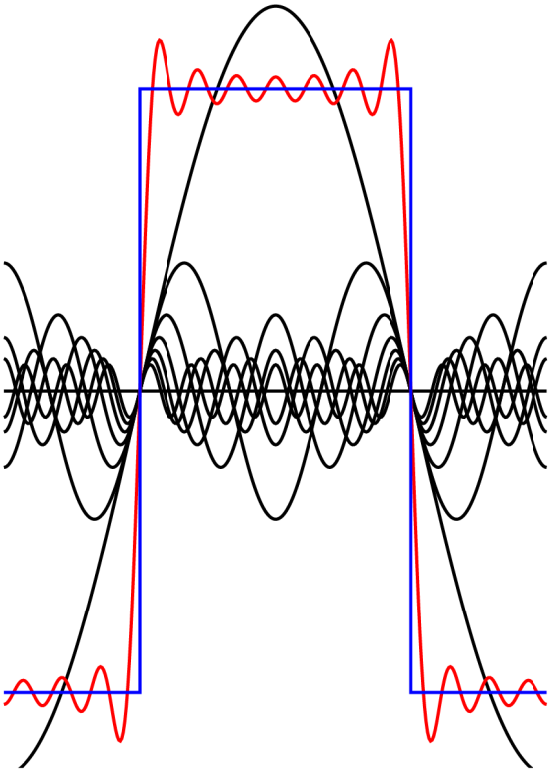


手で計算する時は, このような表を作る.
a_1 の計算は, 0 行目同士, 1 行目同士, ... を掛け合わせて足す.
a_2 の計算は, 0 行目と 0 行目, 1 行目と 2 行目, ... k 行目と 2k 行目を掛け合わせて足す.
a_3 の計算は, 0 行目と 0 行目, 1 行目と 3 行目, ... k 行目と 3k 行目を掛け合わせて足す.
のようにするが, 結構めんどうくさい.

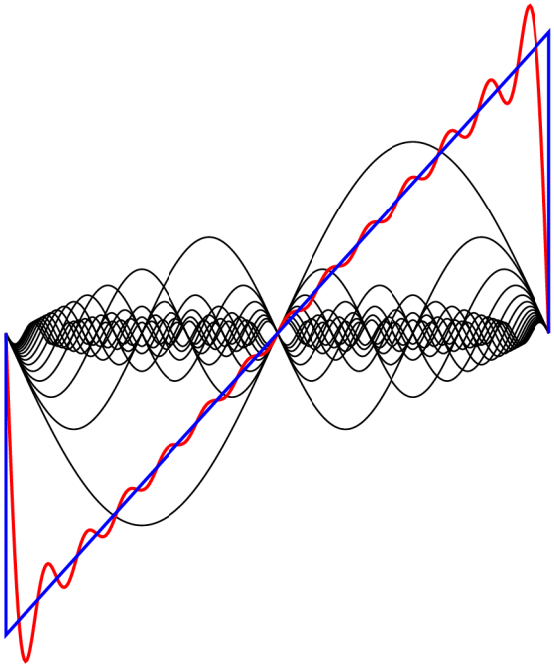
計算機があれば, すぐ試すことが出来る.

```
(define nn 64)

(define (a n)
 (let ((an 0))
  (do ((m 0 (+ m 1))) ((= m nn))
    (set! an (+ (* (f m) (cos (/ (* 2 pi (+ (* m n) 0.5)) nn))) an)))
   (/ an nn)))

(define (f n)     ; ステップ関数の時
 (cond ((< n (/ nn 4)) -1) ;1/4 より小さい時 -1
       ((>= n (/ (* nn 3) 4)) -1) ;3/4 より大きい時 -1
       (else 1)))          ; それ以外は 1

(define (f n)     ; 鋸歯状波関数
  (+ (/ (* 2 (+ n 0.5)) nn) -1))
```

ステップ関数の場合



鋸歯状波関数の場合



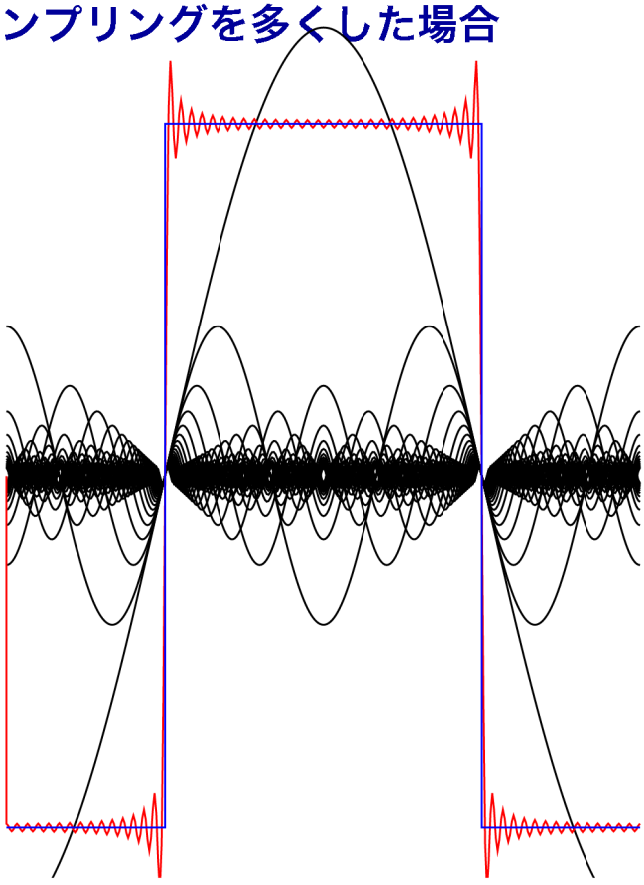
ステップ関数の係数

0
-.6368755077217536
0
.21194999078372936
0
-.12614008280347488
0
.08876606163859643
0
-.06752635118750022
0
.05360803812501698
0
-.04361835776624559
0
.03597089502453032

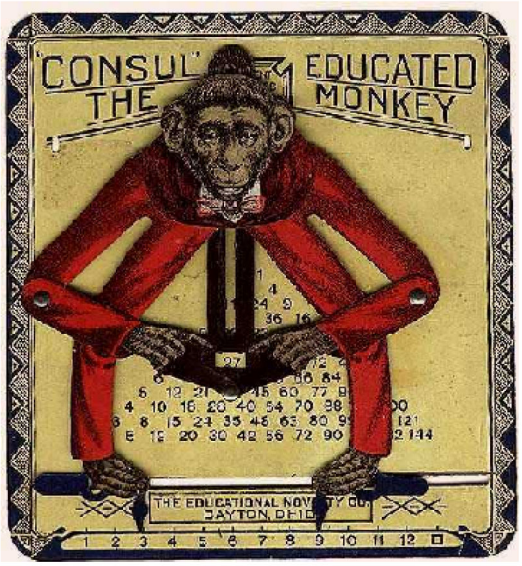
鋸歯状波関数の係数

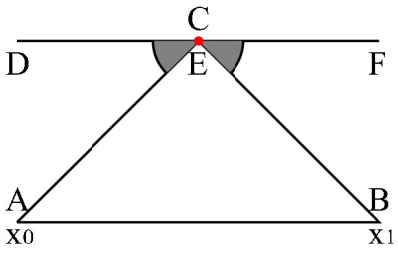
-.3184377538608768 6
-.1592188769304384 3
-.1059749953918646 5
-.0792242424353677 4
-.063070041401737 5
-.0522133599145449 1
-.0443830308192982 5
-.0384433314177275 5
-.0337631755937502 2
-.0299637897371944 56
-.0268040190625084 62
-.0241229798407529 3
-.0218091788831225 15
-.0197828665477294 76
-.0179854475122653 08

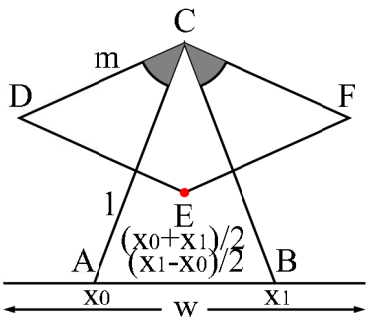
ステップ関数でサンプリングを多くした場合

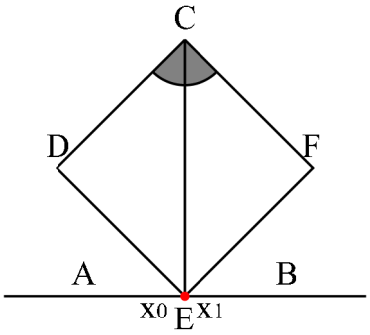


学者猿コンサル Consul the Educated Monkey

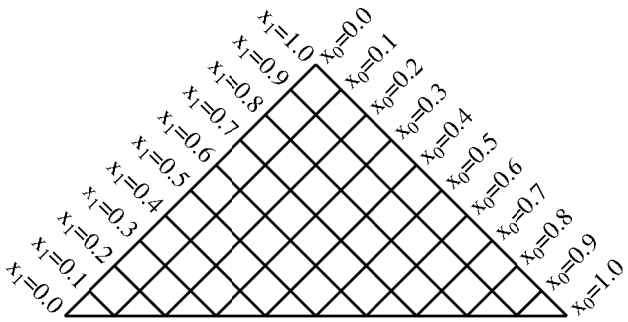




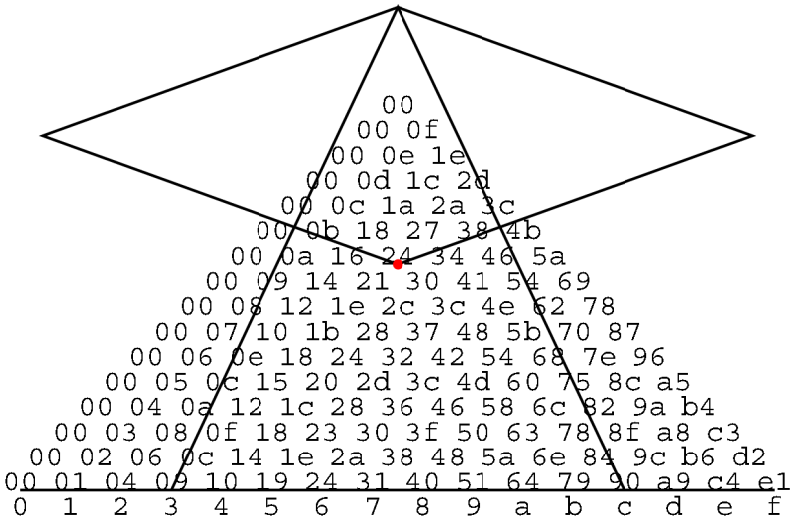




これは一種の計算図表



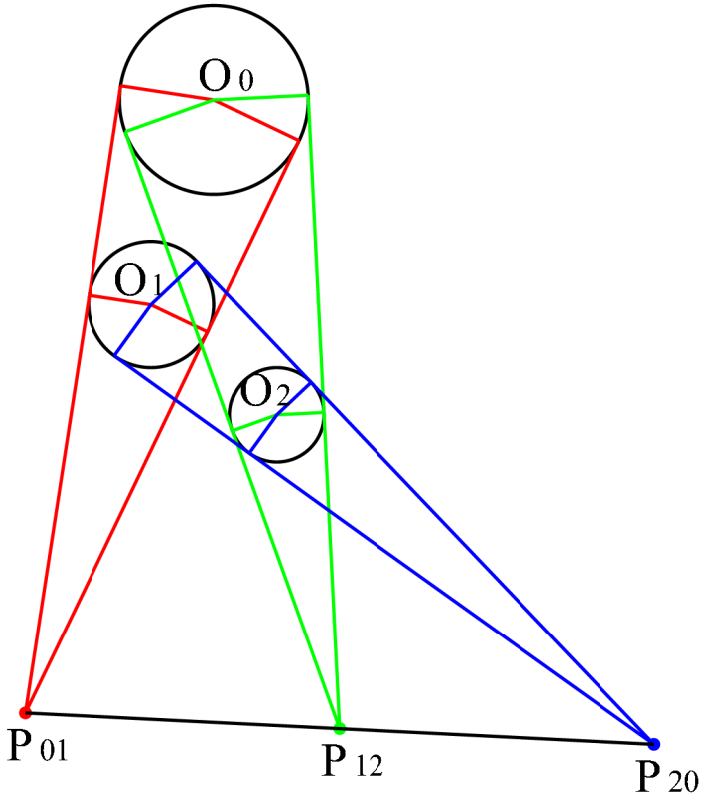
十六進の乗算表を作ることが出来る.

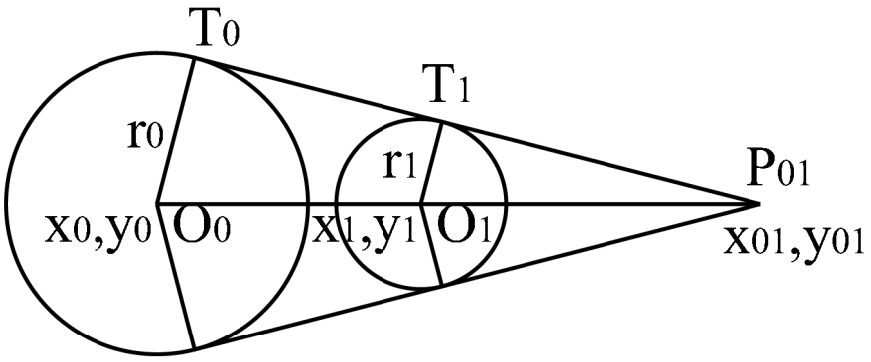


三つの円

平面上の２つの円の共通接線の交点を, それらの円の焦点という.

３つの円で出来る３つの焦点は一直線上にある.





$x_{01} = (r_0x_1 - r_1x_0)/(r_0 - r_1),\ y_{01} = (r_0y_1 - r_1y_0)/(r_0 - r_1)$

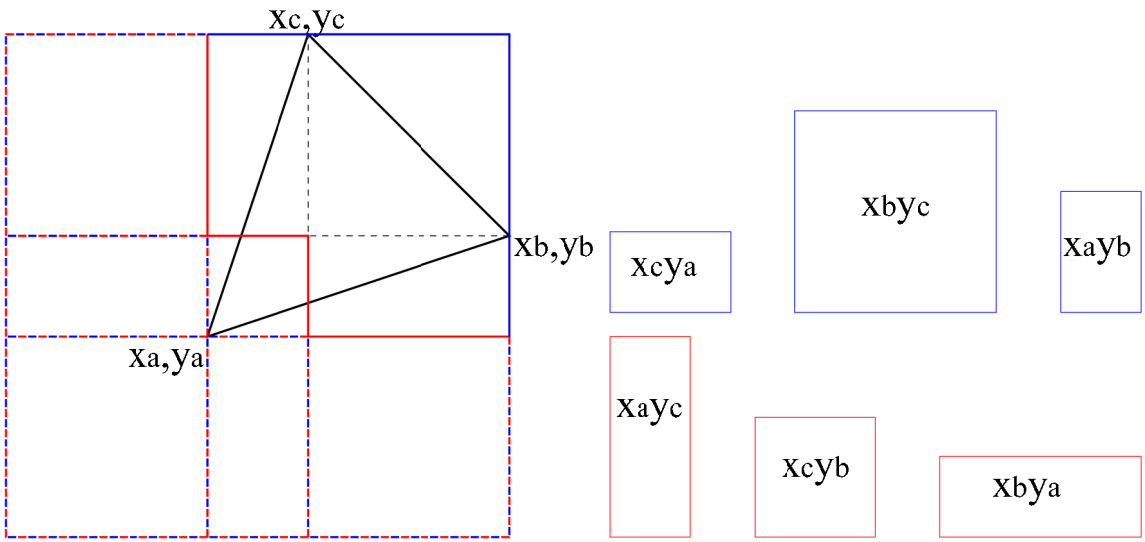
$x_{12} = (r_1x_2 - r_2x_1)/(r_1 - r_2),\ y_{12} = (r_1y_2 - r_2y_1)/(r_1 - r_2)$

$x_{20} = (r_2x_0 - r_0x_2)/(r_2 - r_0),\ y_{20} = (r_2y_0 - r_0y_2)/(r_2 - r_0)$

3点 (x_a, y_a), (x_b, y_b), (x_c, y_c) で構成する三角形の面積 S は

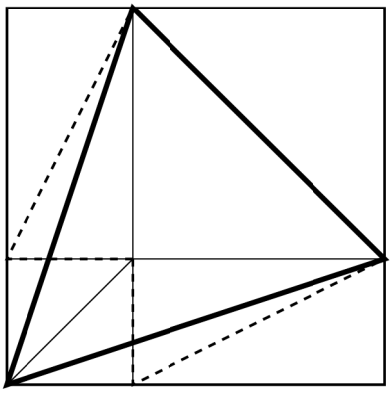
$$S = \frac{1}{2}\begin{vmatrix} x_a, y_a, 1 \\ x_b, y_b, 1 \\ x_c, y_c, 1 \end{vmatrix} \qquad \text{又は } S = \frac{1}{2}((x_b - x_a)(y_c - y_a) - (x_c - x_a)(y_b - y_a))$$

上の行列式の値は



右上の三角形に外接する欠けた四角の面積で,
その1/2が三角形の面積になる.

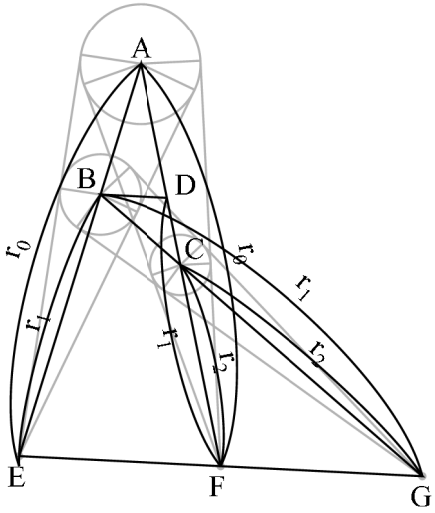
四角形と三角形の面積の関係



risa/asir では

```
X01=(r0*x1-r1*x0)/(r0-r1);
X12=(r1*x2-r2*x1)/(r1-r2);
X20=(r2*x0-r0*x2)/(r2-r0);

Y01=(r0*y1-r1*y0)/(r0-r1);
Y12=(r1*y2-r2*y1)/(r1-r2);
Y20=(r2*y0-r0*y2)/(r2-r0);

X01*Y12+X12*Y20+X20*Y01-X12*Y01-X20*Y12-X01*Y20;
```


幾何学的証明



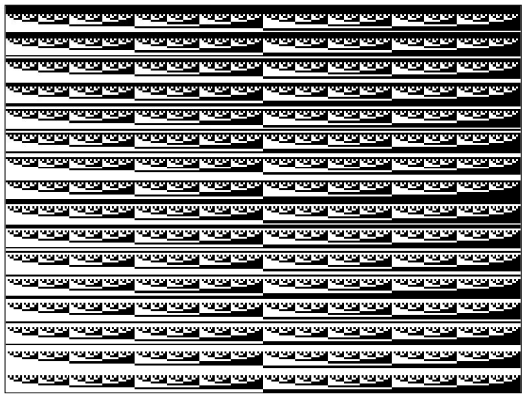
A,B,C の半径を r_0, r_1, r_2 とする.
$\frac{\mathrm{BE}}{\mathrm{AE}} = \frac{r_1}{r_0}$, $\frac{\mathrm{CF}}{\mathrm{AF}} = \frac{r_2}{r_0}$, $\frac{\mathrm{CG}}{\mathrm{BG}} = \frac{r_2}{r_1}$.
AF 上に D をとり, BD//EF とすると $\frac{\mathrm{DF}}{\mathrm{AF}} = \frac{r_1}{r_0}$.

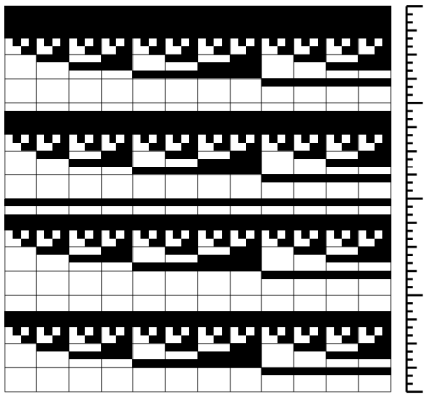
従って BD//FG QED.

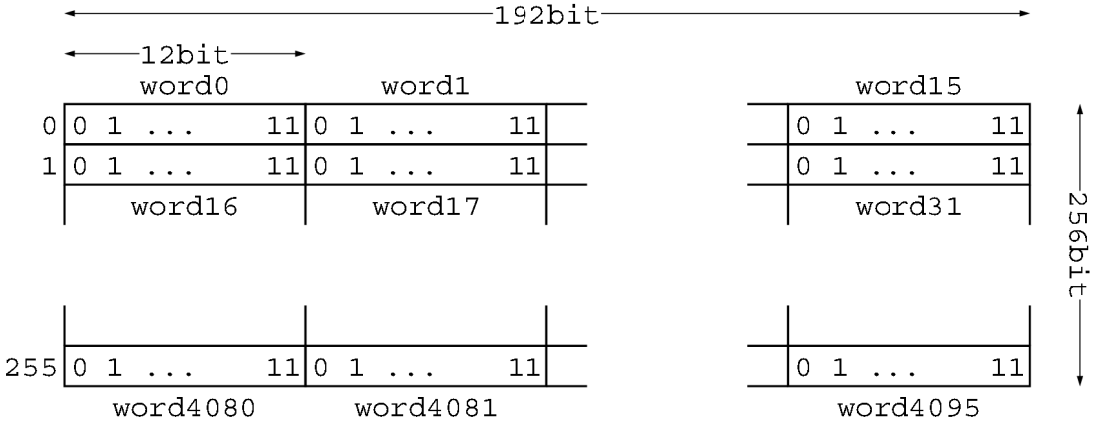
シミュレータ

PDP-8 のメモリーを表示するシミュレータ

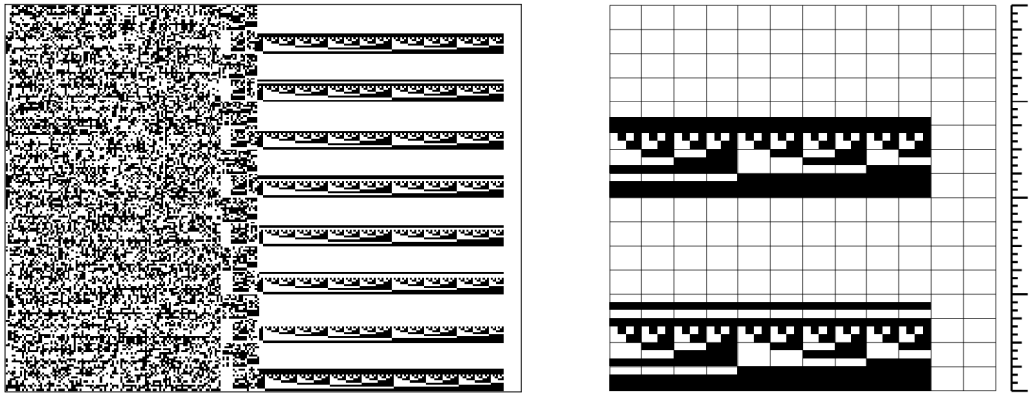
$0 \leq n < 4096$ の n 番地に n を書き込んだところ







van der Poel先生のPDP-8 Lispを読み込んだところ
左がプログラム領域, 右がヒープ領域



Programming should be fun.
Programs should be beautiful.

合 同 分 科 会 選 出
合同分科会 2011 年度会合 より

ICT は新しい社会基盤と どのように協働するか

奈良先端科学技術大学院大学
山口 英

情報通信技術は新しい社会基盤とどのように協働するか

山口英
奈良先端科学技術大学院大学
情報科学研究科

概要

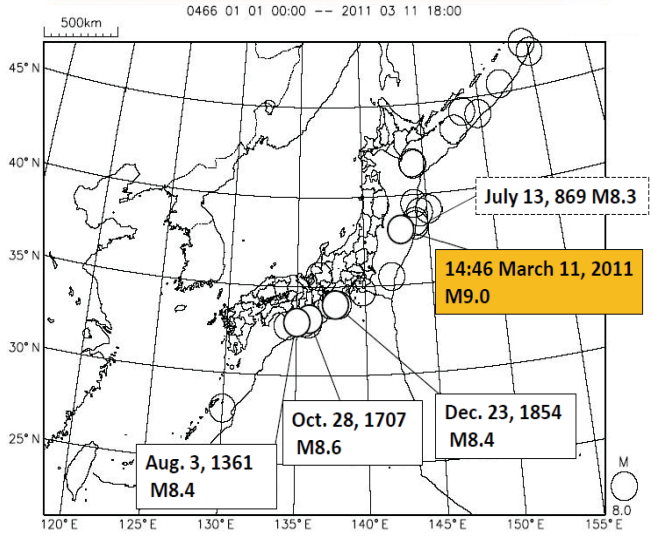
● 東日本大震災以後、わが国の将来像を広く議論されている。その中で ICT の役割について様々なアイディアが提示され、また、他の社会基盤との関係を検討する取り組みも広がっている。特に、大震災からの復興を通して新たな日本の姿を打ち出すという、現在の政府の意欲的な復興方針から、さまざまな技術、制度、経済基盤を対象に検討が広範に行われているのは大変興味深い。本講では、新しい日本への再生プロセスの中でのICTの役割とその将来展望について述べる。

● キーワード
– 高度情報通信ネットワーク社会, インターネット, 共通番号制度、クラウドコンピューティング

3.11 & ICT DEPLOYMENT TO OUR SOCIETY

Overview of "the Tohoku Region Pacific Coast Earthquake"

Huge earthquakes in Japan



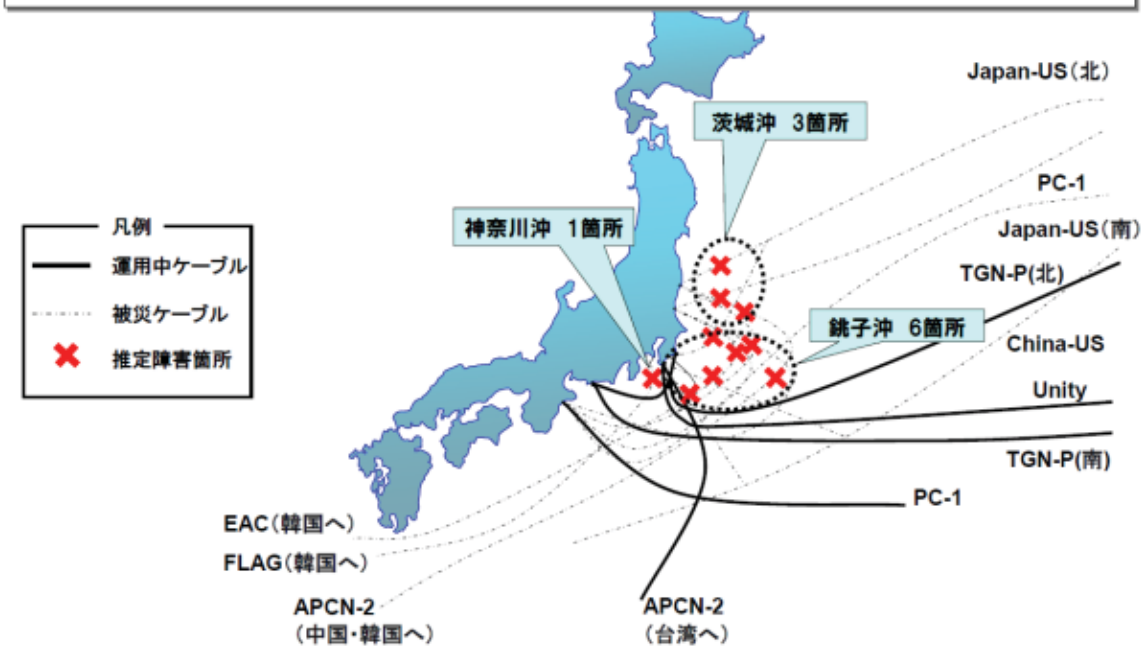
(source: Japan Meteorological Agency)

Outline of the Earthquake	
Day and Time	11 March 2011 14:46 JST (05:46 UTC)
Earthquake Center	The Coast of Sanriku in the Tohoku Area of Japan
Magnitude	9.0
Maximum Seismic Intensity	Level 7 (Northern Miyagi Prefecture)
Number of Deaths	About 15,000
Evacuee (Max.)	About 450,000
Damage	Building Collapse, Fire Disaster, Tsunami, Nuclear Accident

震災の情報通信インフラへのインパクト

- 有線、無線を問わず通信基盤への壊滅的な打撃
- 情報の破壊、喪失
- 情報処理提供不能
  - 特に自治体における情報処理不能は大きなインパクト
- 業務のシステム化による、サービス提供不能
  - 何をしてよいか分からない
  - 機材が無ければ何も出来ない

・海底ケーブルの故障により、国際専用線・国際IP－VPN・国際電話付加サービスにサービス影響が発生したが、3月15日15時9分に復旧。



東日本大震災復興構想

- 「復興への提言　～悲惨のなかの希望～」
  - 復興構想会議によってまとめられ、2011年6月25日に発表
  - II.第2章「しごととくらしの再生」
    - (6) 地域経済活動を支える基盤の強化、③人を活かす情報通信技術の活用
    - 避難民への迅速な支援情報の提供
    - 地理的に分散されてしまった避難民と、避難民の旧居住地の自治体の円滑なコミュニケーション環境の確保
    - 復興進捗状況の公開、「見える化」、正確かつ迅速な情報発進
    - 行政、医療、教育におけるデジタル化の推進と、クラウドサービス導入の強力な推進
    - 地域医療、介護、農林水産業、中小企業再建、地域経済の再生・創出における情報通信技術の積極的活用
    - 原子力災害の記録のデジタル化と積極的公開

東日本大震災からの復興の基本方針

- 現在の最新版は「2011年8月11日改定」稿
- 復興構想会議提言を踏まえ、政府が具体的な施策を1ヶ月でまとめたもの
  - 政府の具体的な計画を列記
  - この復興基本方針をベースに、各プログラムを推進するための補正予算、通常予算が編成されていく

復興のための各プログラム

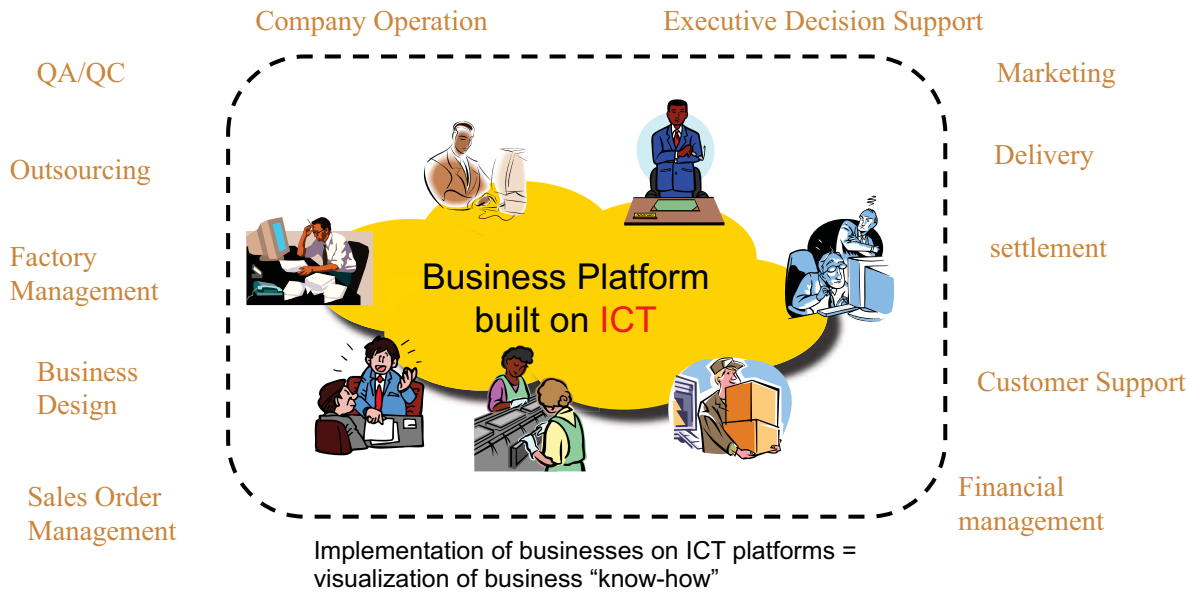
- 高度医療機関と地域の医療機関の連携・協力を確保し、医療・健康情報の電子化・ネットワーク化を推進。
- 地方公共団体をはじめ幅広い分野へのクラウドサービスの導入推進
- 情報通信基盤の復旧、復興等の環境整備を進め、まちづくりと一体となった災害に強い情報通信ネットワークの構築に取り組む
- 被災地域の自治体と住民のコミュニケーション環境の確保と、被災者政策支援での円滑化の取り組み促進。
- 復興進捗状況などの、政府保有データの公開
- 緊急時の物資輸送の情報化等の推進
- 防災教育におけるインターネットの活用
- 総合防災情報システムの機能拡充と、その情報通信網である衛星通信ネットワークの機能強化
- 災害時に強靭な情報システムの構築等、大規模災害時における安全性・信頼性の向上を図る
- 地震、津波災害、原子力事故被害の記録のデジタル化と一元管理、さらには広く国民の活用を可能とする方策（デジタルミュージアム等）の適用

本日の話題

- クラウドコンピューティングの展開
- 番号制度の現状と課題

REALITY OF CLOUD COMPUTING

Business processes and ICT

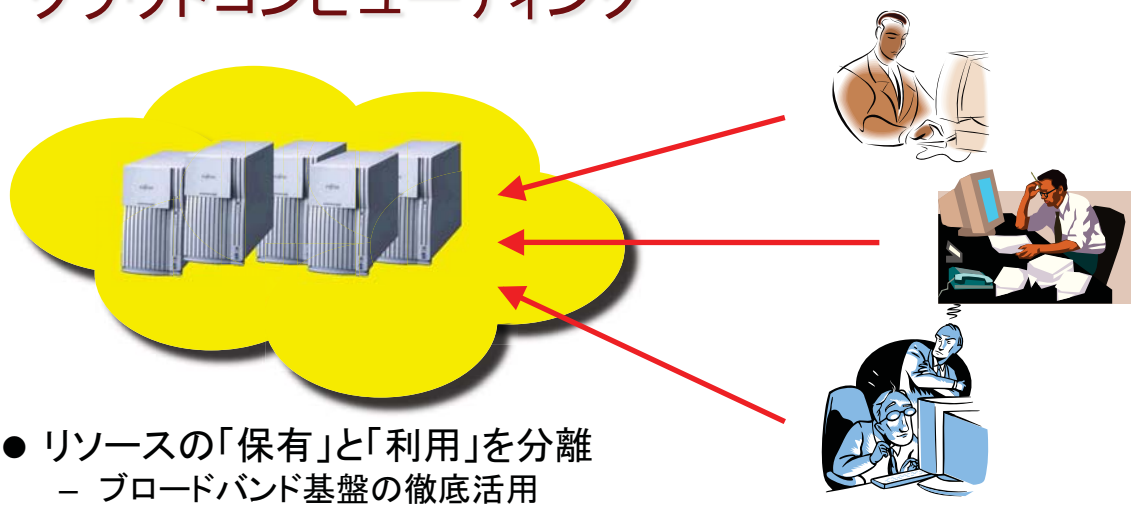


Implementation of businesses on ICT platforms = visualization of business “know-how”

情報処理基盤の整備

- リソースを自己保有
  - 管理ポリシ適用の徹底
  - リソースの追加拡張、ソフトウェアの導入更新の実施が大変
- 運用コスト (TCO) の圧縮と品質の高い運用が常に課題
  - 専門家による運用が期待される
  - IT基盤はどんな業務にも必要になる
  - 運用コストは下げたい

クラウドコンピューティング



- リソースの「保有」と「利用」を分離
  - ブロードバンド基盤の徹底活用
  - 保有と管理のアウトソース
  - 複数のカスタマによる共有で、コスト圧縮をはかる
- ネットワークを介したサービスとして提供
  - 仮想化による多重化により商業化成功
  - ホスティングサービスと比較して高い利益率

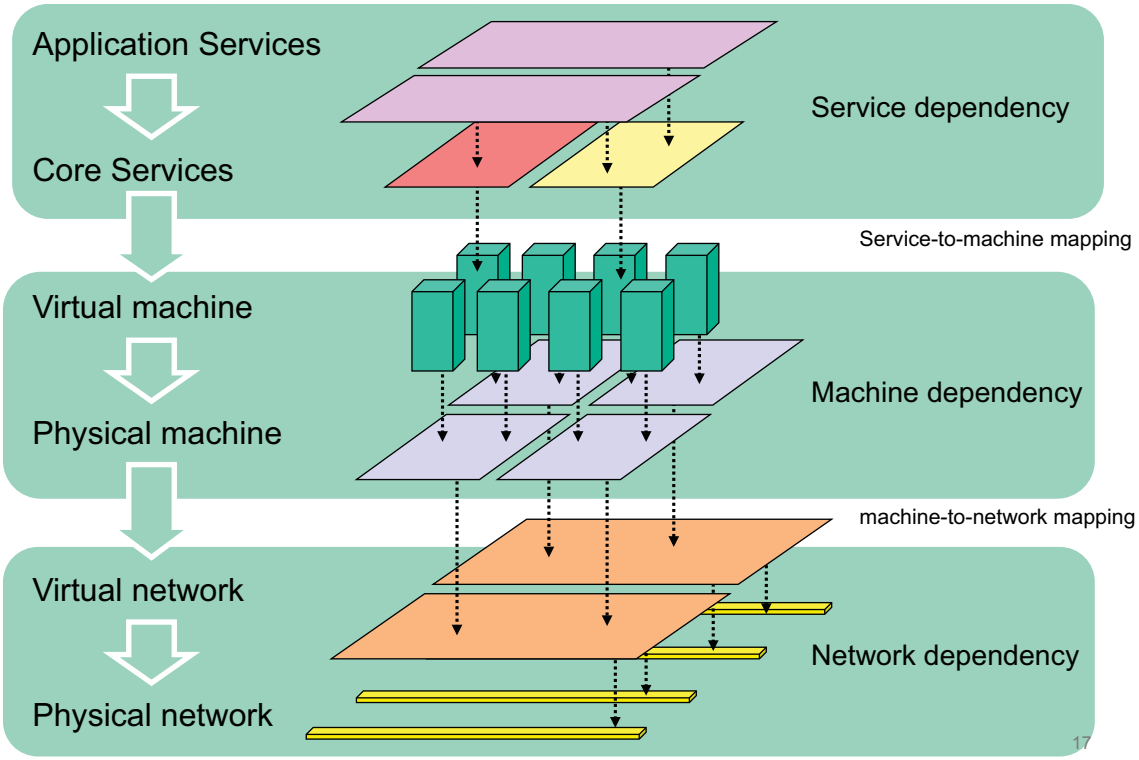
Cloud Computing - 新しいインフラ

- 知見の積極的な注入/専門性の向上
  - 専門家の知見の展開が簡単に実施できる
  - 知恵と知見の集約によるインテリジェント化を達成
  - より洗練された機能提供
- 統合管理 (integrated management)
- 計測性を確保し「みえる化」を推進 (measurable infra.)
- 相互接続 (interconnected)
- 他者による利用を想定
  - Openな基盤構成
  - セキュリティ機能を強化した状態でのユーティリティ化
- 資源制限への調整機能と最適化
  - 代表的なものはグリーン機能
  - 仮想化 (virtualization) の活用が積極的
- 耐故障性能の高さ
  - クラスタ構成によるバックアップ構成

Key Elements

- Virtualization（仮想化技術）
  - Optimization of resource usage
  - From exclusive assignment to shared assignment
- Off-site processing / storage（情報処理の集中化）
  - Professional management
  - On time basis of resource assignment
  - Robust & Resilient system operation are in the hand of operators.
- Open Innovation（オープン性）
  - Mainly by industries.
  - Less risk of “discontinue” incident.

Components in Information System



数多くのメリット

- データ処理の移動性を確保できる
  - 処理(process)とデータのシステム間転送が可能
    - Process migration
  - 物理的・地理的な分散が可能になる
  - 新たな分散処理基盤の構築と実装が可能
  - BCPへの貢献大
- On-demand resource assignment
  - 処理に必要な資源を動的に割り当てることができる
  - 処理の最適化
  - 夢の「適時適切な資源割り当て」
  - 可用性の改善

地球規模での分散処理

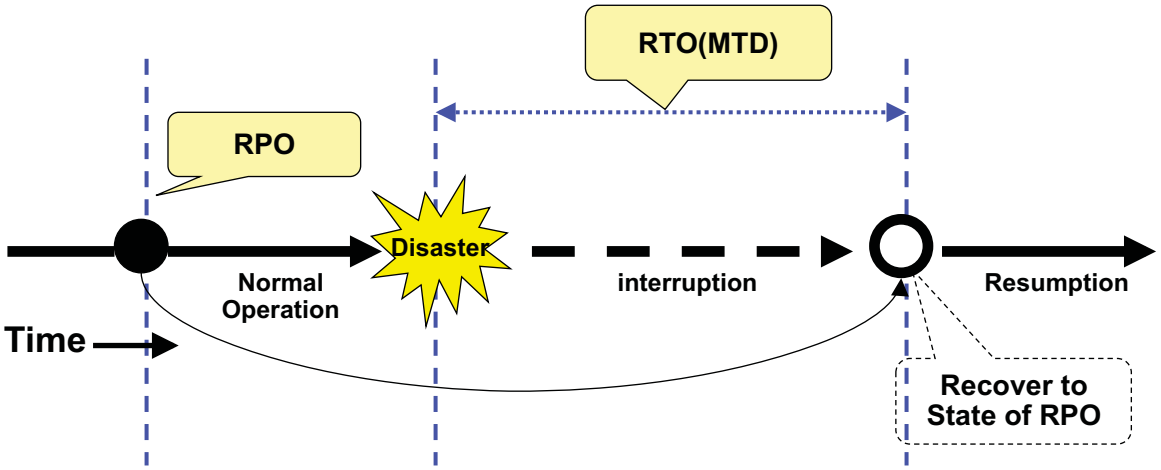
- 地球上に分散した複数のクラスタを10Gbps以上の回線で結ぶ
  - データ共有基盤を構築
  - 数万台のクラスタを仮想的に構成
- 地球規模の分散処理に挑戦
  - 異機種プラットフォーム接続も挑戦
    - Lace Project (ISI, NAIST, NICT)

BCPと Cloud Computing

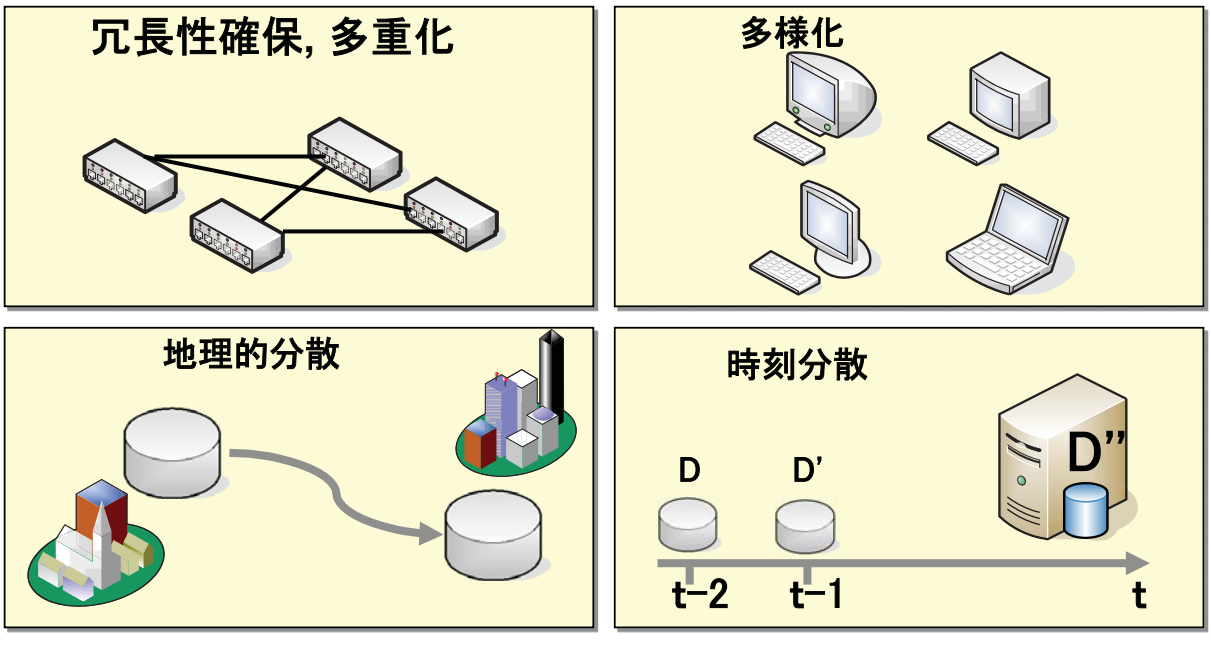
- Cloud computing はリスク集中を引き起こす
  - 単一のHW上に、複数の処理コンポーネントを実装
  - 単一の処理機構が、大量のデータ処理を実施
- トラブル発生時のインパクト軽減が必須
  - 処理コンポーネントの分散→積極的な処理分散
  - 充実したBCP & Disaster Recovery Management (DRM)
- 仮想化技術により、データ、情報処理の移転は容易
  - BCP実装は比較的容易になる
  - 地理的制約も受けにくい
  - ここにチャンスを見いだしている

RTO(MTD) and RPO

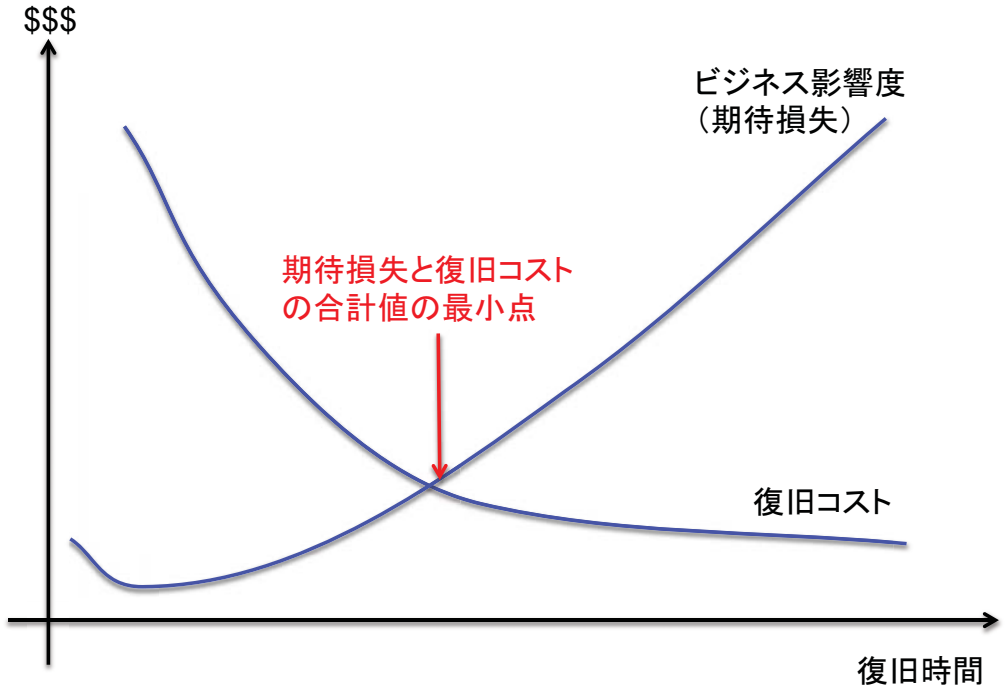
RTO (Required Time Objective) = 目標復旧時間
MTD (Maximum Tolerable Downtime)
RPO (Recovery Point Objective) : 目標復旧時点



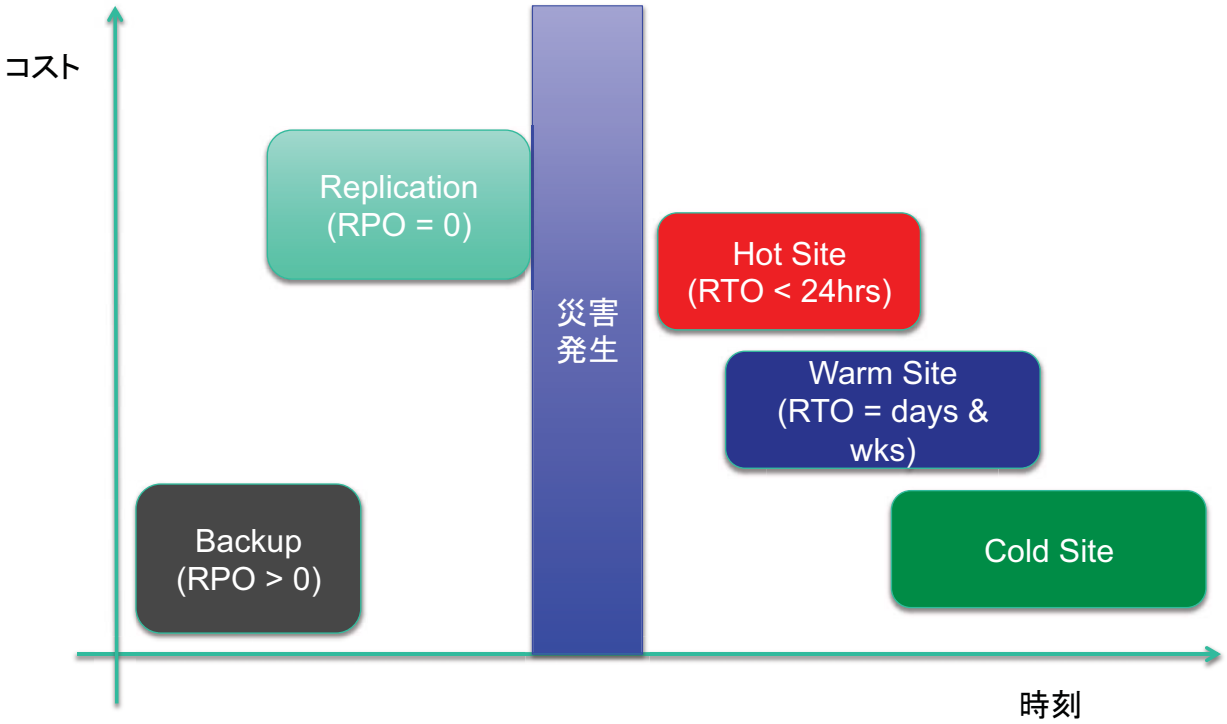
基本作戦



コストと復旧時間



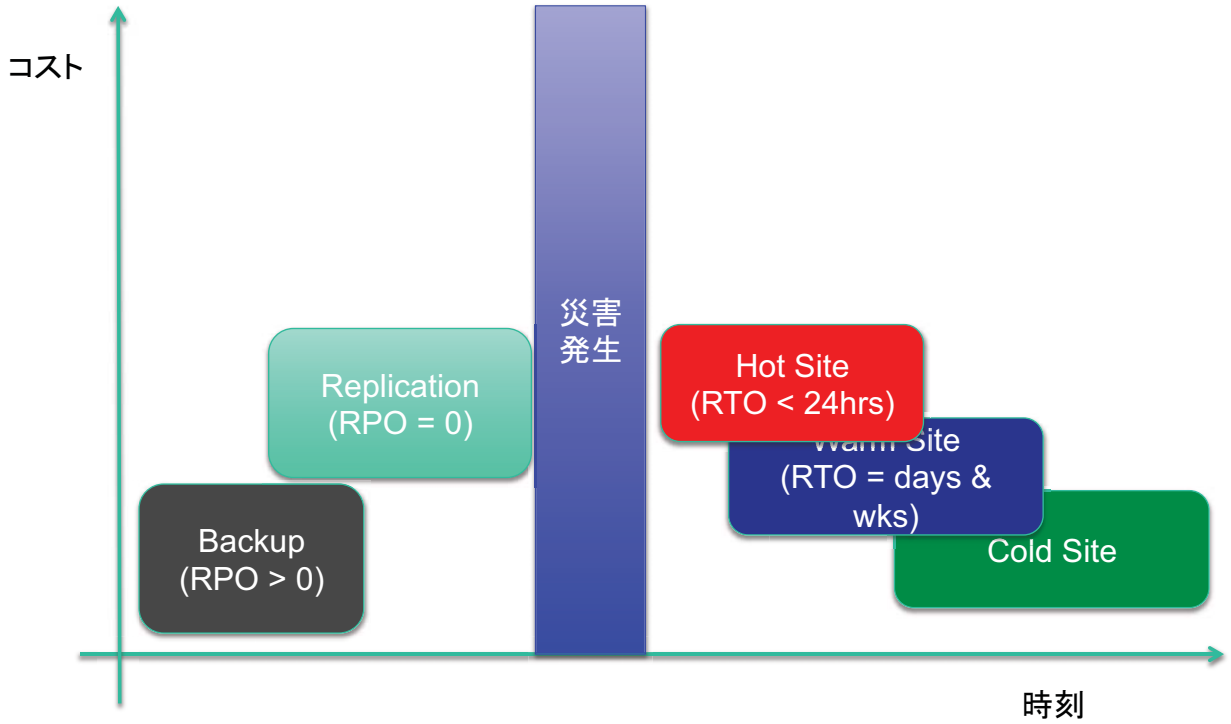
RPOとRTOのバランス



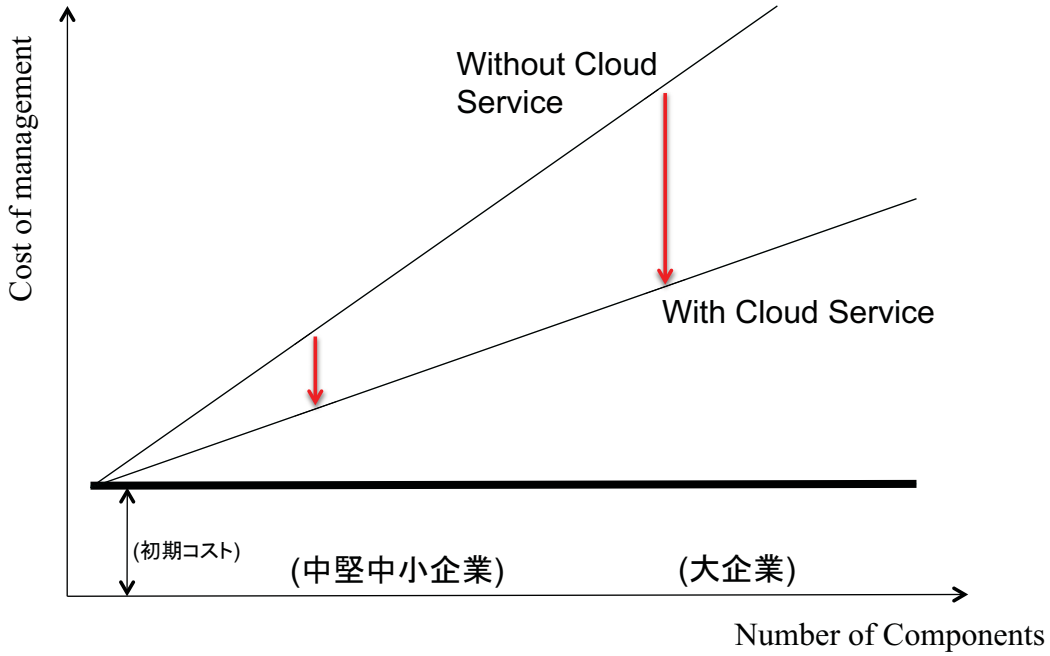
新たなチャンス

- Cloud Computing を活用した、複数のDCを活用しながらのBCPの実現
  - 仮想化されたシステム資源は、複製、移動が簡単にできることが、BCP実現の柔軟性を与えた
  - 実現性がある
    - 従来では想定しえなかった（資源が潤沢に無かった）
    - 現在ではDC間のネットワークが充実している
      - 40Gbps, 100Gbps, or Bus extension
    - Akamai.com は実現している
  - 地理的に離れたDCへの積極的な分散も、ビジネス的に検討可能なオプションになっている

コストは下げられる



TCO & Cloud Computing



JAPANESE RESIDENTS ID SYSTEM AND GOVERNMENT APPLICATIONS

共通番号制度

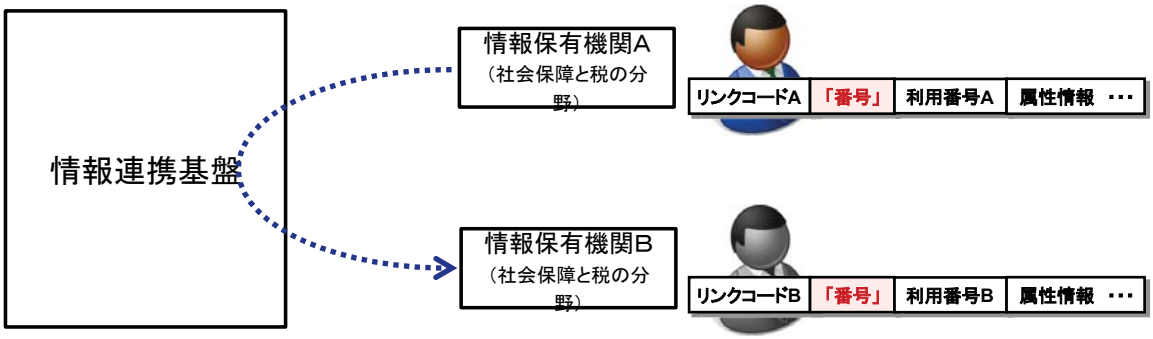
- 国民に関わる情報は、地方自治体、国の行政機関等で、広く分散して蓄積されている
- 実は、これらのデータのフォーマットも、付番もバラバラ
  - 効率的で一体的な行政サービス提供が困難
  - 世界各国で様々な取組があるが、日本では納税者番号制度（日本版グリーンカード）や住民基本台帳番号の導入などで、様々な物議を醸してきて、全面的な導入が行われてこなかった。
    - 米国 SSN や欧州各国における住民登録カードの例は有名
  - 民主党政権では、税と社会保障の一体化のために、「共通番号制度」の導入を公約として掲げている

現在の検討状況

- 社会保障・税に関わる番号制度についての基本方針（2011年1月31日決定）
- 社会保障・税番号大綱（2011年6月30日決定）
- 現在、具体的なシステムおよび法律の検討作業を政府において実施。同時に国会においても審議中。
  - 管政権から野田政権への政権交代で、関係する閣僚・政務が大幅に入れ替わったため、法案検討は引き続き役所で進んでいるが、政治的な観点からの議論・検討は遅延している
  - 次期通常国会への法案提出を目指す

番号制度と情報連携

- 番号制度導入で具体的に起きること
  - 各機関ごとに分散して管理されている同一人の情報を紐付けし、参照可能にする
    - 行政機関での「名寄せ」を、「情報連携」機能を使って実現する
  - きめ細やかな制度設計や行政事務効率化を図る
    - 例えば、国民一人一人の所得等の状況に応じた制度

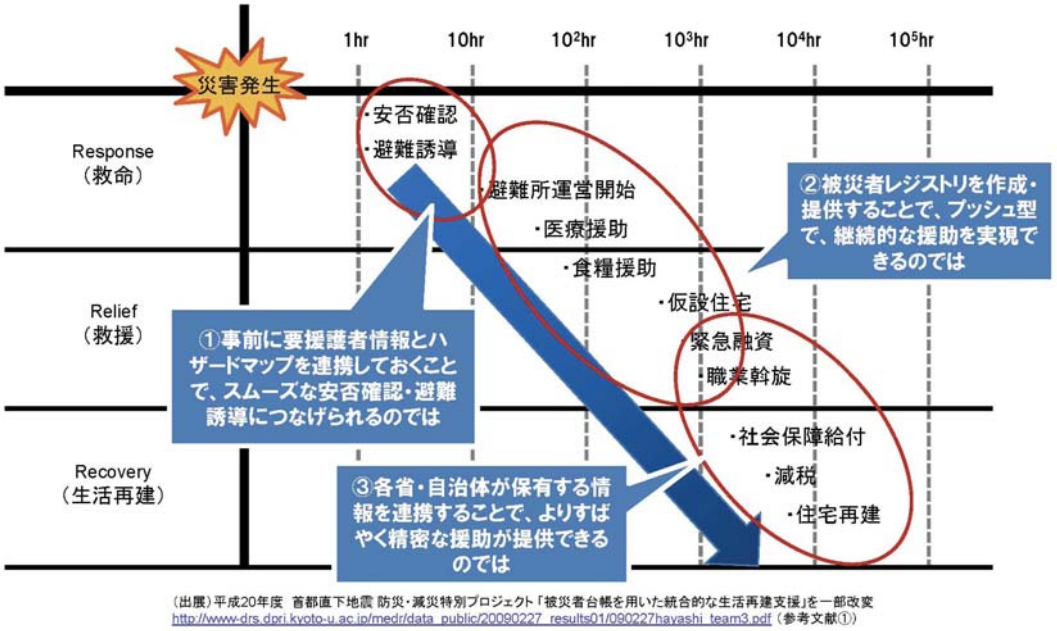


震災と番号制度

- 震災の被害者に対する様々な援助が円滑に行われないことの一つに、地方自治体における個人情報の名寄せが上手く出来ないことが原因と言われた
  - 罹災証明書の発行
  - 被災者台帳作成

- 既に地方自治体が持っている情報を相互にリンクするだけでも、かなり色々なことができるはずであると考えた人達が多かった
  - 共通番号制度の導入に期待

被災者支援のフェーズ



（出展）平成20年度　首都直下地震 防災・減災特別プロジェクト「被災者台帳を用いた統合的な生活再建支援」を一部改変
http://www-drs.dpri.kyoto-u.ac.jp/medr/data_public/20090227_results01/090227hayashi_team3.pdf（参考文献①）

2

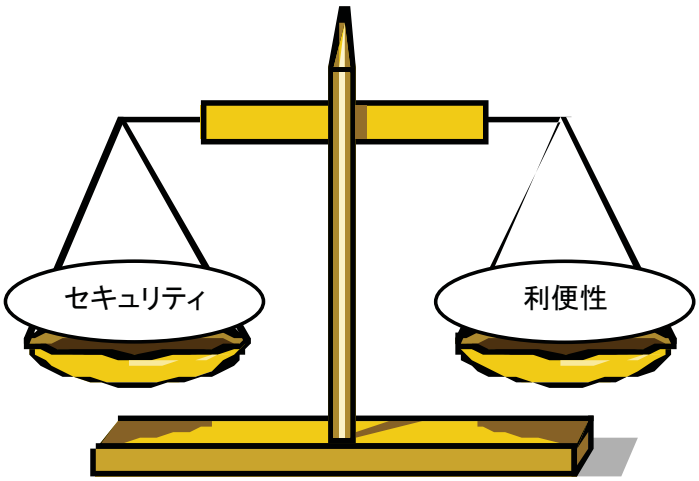
（情報連携基盤技術WG第4回[2011.04.23]配布資料13/ユースケース（災害）崎村委員作成資料より抜粋）

番号制度におけるセキュリティリスク

- 想定されるセキュリティリスク
  - データ漏洩
    - 情報保有機関が管理するデータベース（DB）
    - ネットワークで通信される電文
  - 本人確認情報の漏洩
  - 行政機関による名寄せ、目的外使用
  - 民間機関による名寄せ、目的外使用
  - 番号の不正利用、それに伴う経済的損失等
  - 行政職員による不正閲覧、データ改ざん、データ破壊
  - 国家による一元管理への不安
  - 情報連携機能の障害によるサービス全体への影響

セキュリティリスクへの対応

- 番号制度におけるセキュリティリスクへの対応
  - 各情報保有機関における適正なセキュリティ管理の徹底
    - DBからの直接漏洩、本人確認情報の漏洩、行政職員による不正行為
    - しかし、リスクゼロにはできないので、救済制度を設ける
      - 既存の救済制度：行政機関における個人情報保護法による枠組み
  - 通信路における電文漏洩には暗号化
  - 民間機関の名寄せ、目的外利用には、法律で禁止（罰則付き）
    - それでも各国番号制度の状況をみると、結果として名寄せは発生する
    - 現在の源泉徴収業務（法定業務）での利用を考えれば当然
    - 政治的判断が必須
  - 行政機関の名寄せ、目的外利用に対抗するには、法律で禁止（罰則付き）し、同時に透明性を持った運用を実現する
    - 国民にとっては、ここが番号制度における最大の関心事



SUMMARY

ICTの社会展開

- 社会展開では、さまざまな課題を解決する必要が有る
  - Cloud computing & BCP
  - コスト圧縮、効率的な運用、災害対策
  - Engineering の力
- 相反する考え方にも配慮
  - 効率 vs. プライバシ保護
  - コスト圧縮 vs. 災害対応
  - トレードオフの関係にあるものを、どのように扱うか
- 技術の変化、制度改正などで解決できるものも多い