python kVerifier ライブラリの開発

kVerifeir Lab: 小林憲次

**テストを
ソースから独
立 させよう！**

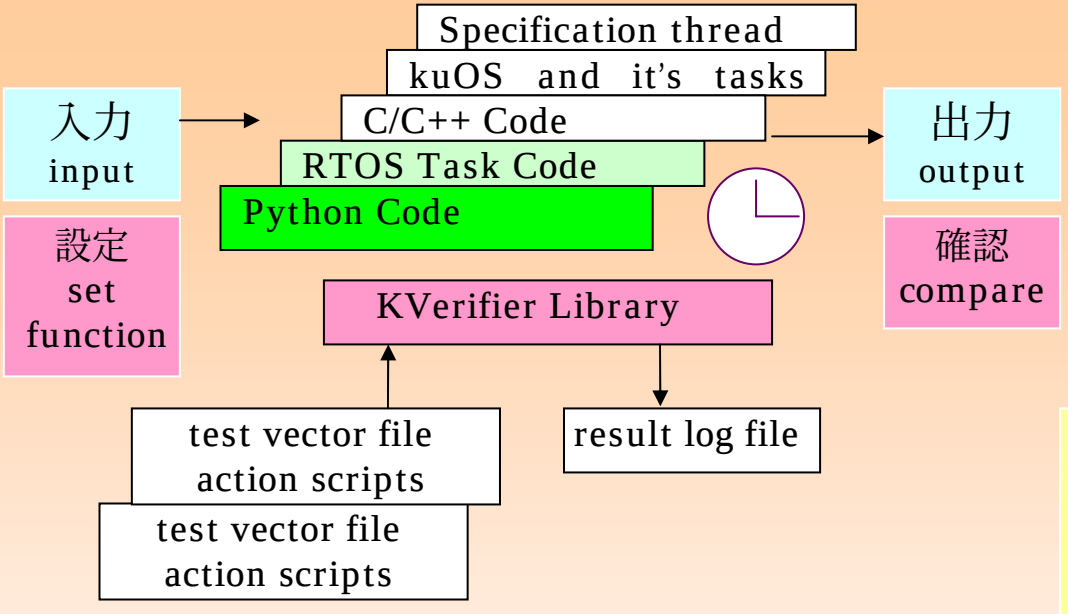
テストはテスターが主に書くべき

テストは詳細仕様でもあるべき

組み込み開発でもテストを使うべき

python の書きやすさ読みやすさを活用しよう
python を動く仕様モデルの記述に使おう
python で作ったテストを C/C++ でも使おう

What does the kVerifier Library ?



Test C/C++
Python
Program

Can Control
Time

kVerifier vs
TestBuilder
Sugar,CppUnit

Can Test SystemC circuits and/or RTOS tasks,Spec.

Set input, Call interrupt and then Compare output

Separated action scripts describe program actions

Total Test: circuits, tasks specifications all together

Example of Calculator's Action Scripts

```
#time name  command   parameter
+1c  入力 __set         3 + 4
+0   結果 __compare  7

+1c  入力 __set         2**0.5
+0   結果 __compare  1.41421356

# data1.val == 7
+1c  入力 __set         data7 * 4 - 3
+0   結果 __compare  25

+1c  入力 __set         "10 + 1 / sin( pi/4 )"
+0   結果 __compare  11.41421

+1c  入力 __set         3.1 + 4.2
+0   結果 __compare  7.3

+1c  入力 __set         3 ** 2
+0   結果 __compare  9

+0 __end
```

時刻／回数　変数名　__command　引数

具体例の羅列：非プログラマーも記述

誰でも読める：computer, 外国の人

アルゴリズムを記述しない：明晰

テスターによるテストの作成

Action Script Language

詳細仕様記述：自然言語に勝る
文才を必要としない

ソースから独立：単なる Text File
Python と C でテストを共用できます

四則演算だけだとしても、行列・ベクトルまで含むと、そのテストは膨大になります

kVerifier ⊃ unit test:テストもソフト資産

ブラック･ボックス	ホワイト･ボックス
トータル･テスト	ユニット･テスト
ソースからの独立	ソースと一体
関係者全員が作成	プログラマが作成
時間･順序を基準	順序･時間がない

テストをソフト詳細仕様に！コード実装に依存させない

テスターが主体になりカバレッジ 100% の全体テストを

再利用可能なテストを！　テストのソフト資産化を！

Light Weight Language Python とは

読みやすさ

Python　Ruby　Perl

書きやすさ

bisect モジュールを使っての読みやすさの例示

```
bisect_right(...)
        bisect_right(a, x[, lo[, hi]]) -> index

        Return the index where to insert item x in list a, assuming a is sorted.

        The return value i is such that all e in a[:i] have e <= x, and all e in
        a[i:] have e > x.  So if x already appears in the list, i points just
        beyond the rightmost x already there

        Optional args lo (default 0) and hi (default len(a)) bound the
        slice of a to be searched.
```

リスト a がソート済みだとして、引数 x がリスト a の中で収まるべきインデックスを返す。

戻り値 i は、a[:i] に属する全ての e に対して e<=x であり、また a[i:] に属する全ての e に対して e > x である。だから x がリスト引数で分類済みならば、インデックス i は、リスト上の x があるべき位置に対して右側で最も近い位置を示す。

オプション引数 lo(デフォルトで 0) と hi(デフォルトで len(a) は探索するときのスライスを限定する。

```
def bisect_right(a, x, lo=0, hi=None):
    if hi is None:
        hi = len(a)
    while lo < hi:
        mid = (lo+hi)//2
        if x < a[mid]: hi = mid
        else: lo = mid+1
    return lo
```

ソースが一番解りやすい！

Python には多くのモジュールが揃っている

python sf:python モジュール蓄積の豊富さと python kVerifeir の実用プログラムへの適用の例示

行列;mt=~[[1,2,3],[4,5,6],[7,8,10]]

```
==========================
[[  1.   2.   3.]
 [  4.   5.   6.]
 [  7.   8.  10.]]
```

mt=~[[1,2,3],[4,5,6],[7,8,10]];1/mt

```
================================
[[-0.66666667 -1.33333333  1.        ]
 [-0.66666667  3.66666667 -2.        ]
 [ 1.         -2.          1.        ]
```

mt=~[[1,2,3],[4,5,6],[7,8,10]];mt^-2 + 3 mt^2

```
================================
[[  92.33333333  102.          138.        ]
 [ 194.          261.33333333  296.        ]
 [ 328.66666667  391.33333333  513.        ]]
```

多項式poly1d([1,1,0,-2])

```
======================
   3     2
1 x + 1 x - 2
```

根;;f=poly1d([1,1,0,-2]);f.r

```
======================
[-1.+1.j -1.-1.j  1.+0.j]
```

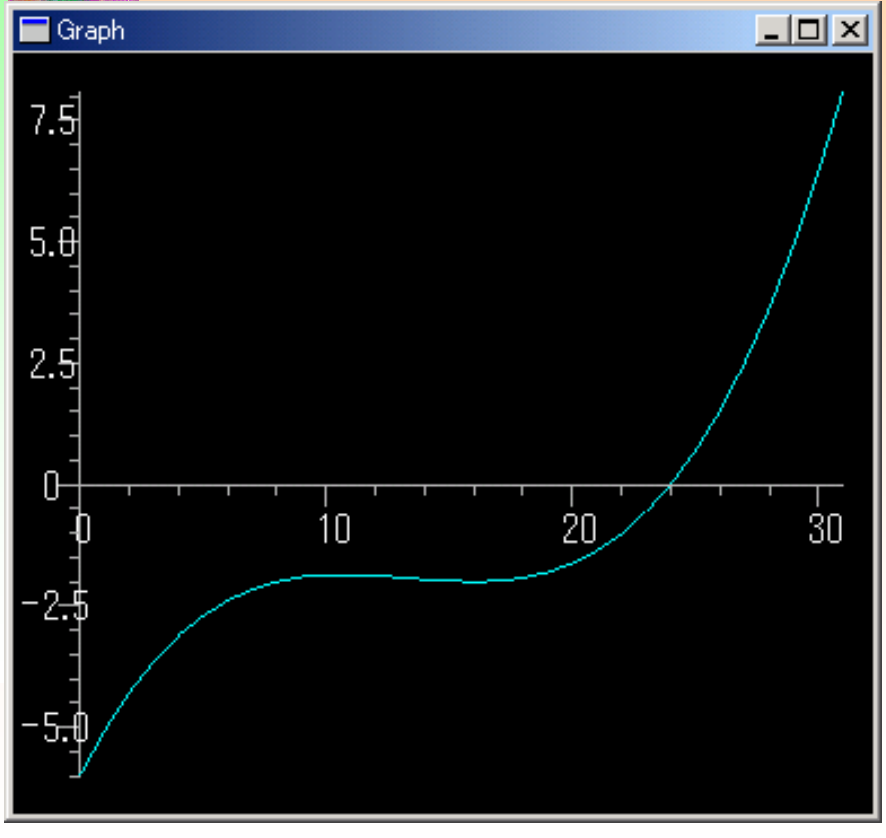
N=32; f=poly1d([1,1,0,-2]);[f(x) for x in arsq(-2,N,4/N)]

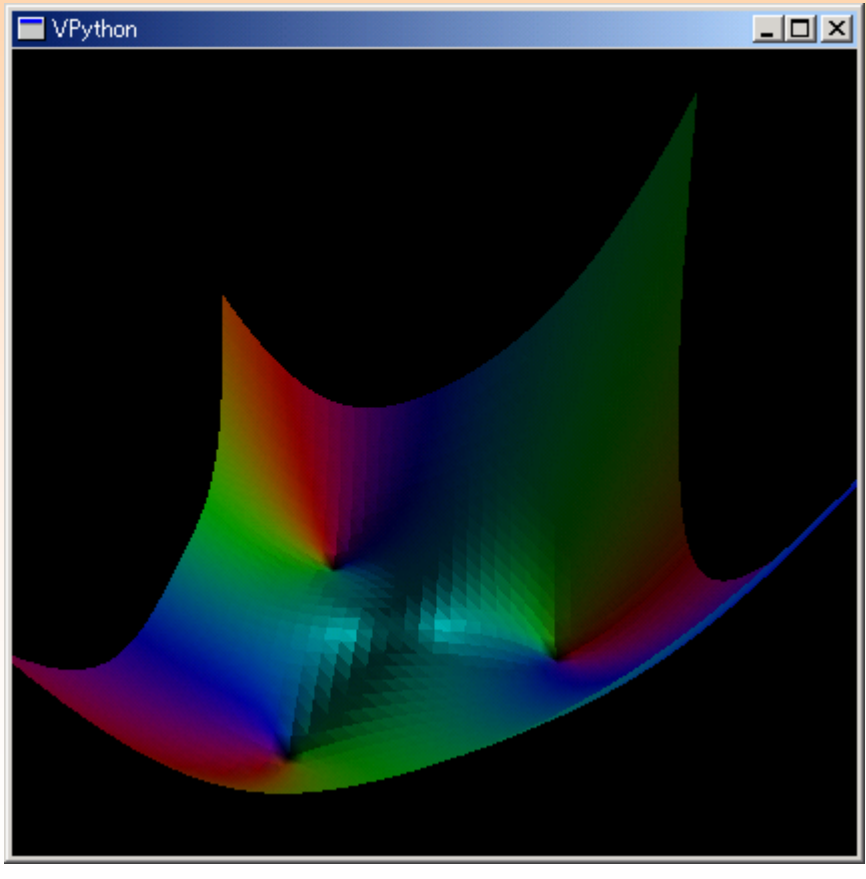
```
==========================
[-6.0, -5.076171875, -4.296875, -3.650390625, -3.125, -2.708984375, -2.390625, -2.158203125, -2.0,'''
2.490234375, 3.625, 4.931640625, 6.421875, 8.107421875]
```

python sf:2d/3dグラフ表示

```
N=32; f=poly1d([1,1,0,-2]);
plotGr([f(x) for x in arsq(-2,N,4/N)])
```

```
N= 32; f=poly1d([1,1,0,-2]);
renderMtCplxWithRGB(
[[f(x+`i y) for x in arsq(-2,N,4/N)]
 for y in arsq(2,N,-4/N)])
```





python sf:シンボリック計算

微分;;∂x(1/ts.sqrt(`x^2+`y^2+`z^2))

==========

-x/(x**2 + y**2 + z**2)**(3/2)

ベクトル微分;;∇(1/ts.sqrt(`x^2+`y^2+`z^2))

==========

(-x/(x**2 + y**2 + z**2)**(3/2), -y/(x**2 + y**2 + z**2)**(3/2), -z/(x**2 + y**2 + z**2)**(3/2))

python sf⇔ プリプロセッサ + scipy + sympy + vpython

Mathematica/Matlab/Maple 等の有名ソフトと同等！

!!! 数学ソフトの L.L !!!　grepの感覚で数式･数値計算を

python sf ベリファイ変数／関数の登録

```
#calculator 関数
def calculateLineString(strAg):
    #import pdb; pdb.set_trace()
    global strExceptionStt
    try:
        __execLine(strAg )
    except StandardError, valAt:
        strExceptionStt = str(valAt)


#ベリファイ変数登録コードを最後に追加します
import kv
clTestGlb = kv.ClActnScrpts(globals())

clTestGlb.rgstVrfyVar( inStt     = ('@kv.a2i',)
        , blStt          = ('@bool',)
        , scalarStt      = (kv.tolerantType, '@complex')
        , calculateLineString   = (kv.fnType, )
        , converFileAndExecute  = (kv.fnType, '@str')
        , krryStt            = ('@sf.krry',)
        , strExceptionStt       = ('@str',)
        , checkResultVal        = (kv.fnType, '@str')
        , checkResultArr        = (kv.fnType, '@str')
)
```

変数のアドレスと型と変数名文字列
(結果、入力)の組を kVerifier に登録

関数もベリファイ変数にできます

変数：型とアドレス <== 文字列

変数名を介して
kVerifier Library が
action scripts と application
を相互に作用させる

既存の汎用デバッグ・ツール
IDE, Debugger, Profiler 等
を利用できます

python sf のテストから、次のことが解ります

python ライブラリ蓄積の凄さ、Mathematica/Matlab 以上

テストを書くこと自体は簡単

テストの難しさは、ソフト仕様の正しさを検証することにある

下の全てを組み合わせた仕様の整合性を保つのは難しい

- ユーザー拡張を含むプリプロセッサ拡張
- 四則演算+べき乗
- ベクトル行列演算
- 整数:含bigNum
  浮動小数点(含 NaN)
  多項式･有理式
  ユーザー拡張による体･環
- シンボリック演算

python kVerifier による交通信号機制御

```
# -*- encoding: cp932 -*-
import kv.VrfyYThrd as vy
#  8 bit output port controlling 2 traffic signal
# 7   6th     5th     4th   v  2nd   1th     oth
#     Red     Yellow  Blu      Red   Yellow  Blue
#|--|--- East/West pair ---{-|- South/North  --|
class ClInt(int):
    def __str__(self):
        return hex(self)
outPort0 = ClInt()
def setSN(inAg):
    """' set South/North traffic signal)
    '"""
    global  outPort0
    outPort0 = ClInt((outPort0 & 0x70)
                + (inAg & 0x07))
def setEW(inAg):
    """' set South/North traffic signal)
    '"""
    global  outPort0
    outPort0 = ClInt(((inAg & 0x07) << 4)
                + (outPort0 & 0x07))
def setEWSN(inAg):
    """' set South/North traffic signal)
    '"""
```

```
global  outPort0
    outPort0 = ClInt(inAg & 0x77)
# 7   6   5  4  3  2  1th     oth
#     n   n  n  n  n  none    emergency button
#|-----------------------|-----|
inPort0 = 0
#calculator 関数
class  ClTrffcLt(vy.ClYThrd):
    """' Traffic light control class
    '"""
    def __init__(self):
        #vy.ClYThrd.__init__(self)
        vy.ClYThrd.__init__(self)
        self.Start()
    def onOffSub(self):
        while True:
            # put on EW_red:, put on SN_red:
            setEWSN(0x44)
            yield vy.Wait(1 )
            #put off EW_all, put off SN_all
            setEWSN(0x00)
            yield vy.Wait(1 )
            if inPort0 & 0x01 == False:
                yield vy.Return()
```

```
def emergencyTask(self):
    #put off EW_all, put off SN_all
    setEWSN(0x00)
    yield vy.Wait(10)
    yield vy.Call(self.onOffSub())
    #put off EW_all, put off SN_all
    setEWSN(0x00)
    yield vy.Wait(10)
    yield vy.Transfer(self.mainVl())
def mainVl(self):
    while(True):
        #put on SN_blue
        setSN( 0x01 )
        yield vy.Wait(20)
        #put on SN_yellow
        setSN( 0x02 )

        yield vy.Wait(3)
        # put on SN_red:
        setSN( 0x04 )

        yield vy.Wait(1 )
        # put on EW_blue
        setEW( 0x01 )
        yield vy.Wait(20)
        #put on EW_yellow
        setEW( 0x02 )
```

```
        #put on EW_red
        setEW(0x04)
        yield vy.Wait(1 )
        if inPort0 & 1 == True:
            yield vy.Transfer(self.emergencyTask())
    def haveGottenException(self):
        import traceback;traceback.print_exc()

clAt = ClTrffcLt()
clAt.Start()
def setInPort0(inAg):
    global inPort0
    inPort0 = inAg
if _name_ == "_main_":
    #import pdb; pdb.set_trace()
    import kv
    clAt = vy.ClVrfyYThrdActn(globals())
    clAt.rgstVrfyVar(inPort0=())
    clAt.rgstMntrVar(outPort0=())

    clAt.VrfyThis("""# action scripts
+0S  _TmSendVariableName outPort0
+0S  _TmSendBitName outPort0 South_North:Blue    bit
+0S  _TmSendBitName outPort0 South_North:Yellow  bit
+0S  _TmSendBitName outPort0 South_North:Red     bit
+0S  _TmSendBitName outPort0 East_West:Blue     bi
+0S  _TmSendBitName outPort0
```

入力ポート：__set

出力ボート：__compare

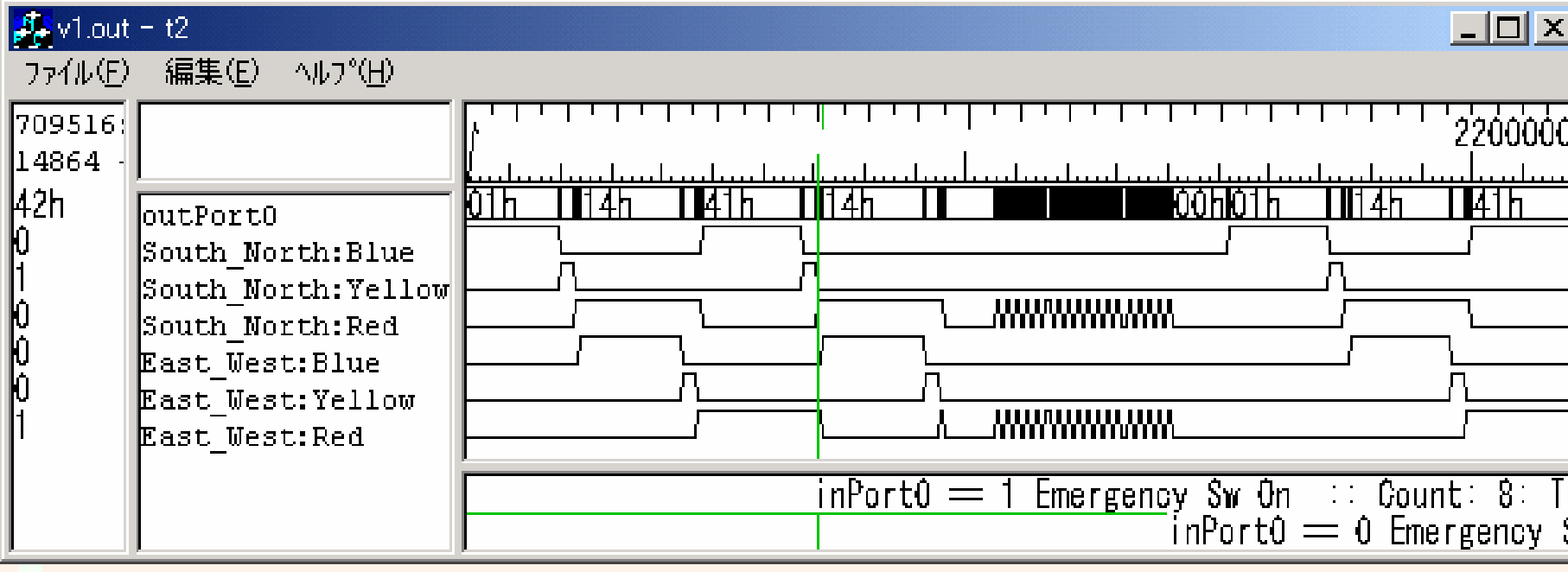
```
East_West:Yellow    bit5
+0S   __TmSendBitName   outPort0   East_West:Red     bit0
## __TmSendText   inPort0 == 1 Emergency Sw On
+70S  inPort0 __set    1    #
## __TmSendText   inPort0 == 0 Emergency Sw Off
+70S  inPort0 __set    0    #0x00
+80S __end
""")
    kv.ClVrfySglt.GetStt().Main()
```

clAt.rgstMntrVar(outPort0=())：モニタ変数登録により変数が変化した時刻とタイミングを verify.out に記録する ➔ タイム･チャート

python モジュールの大部分は thread safe でない
<= python の thread/threading モジュールは実質的に使えない

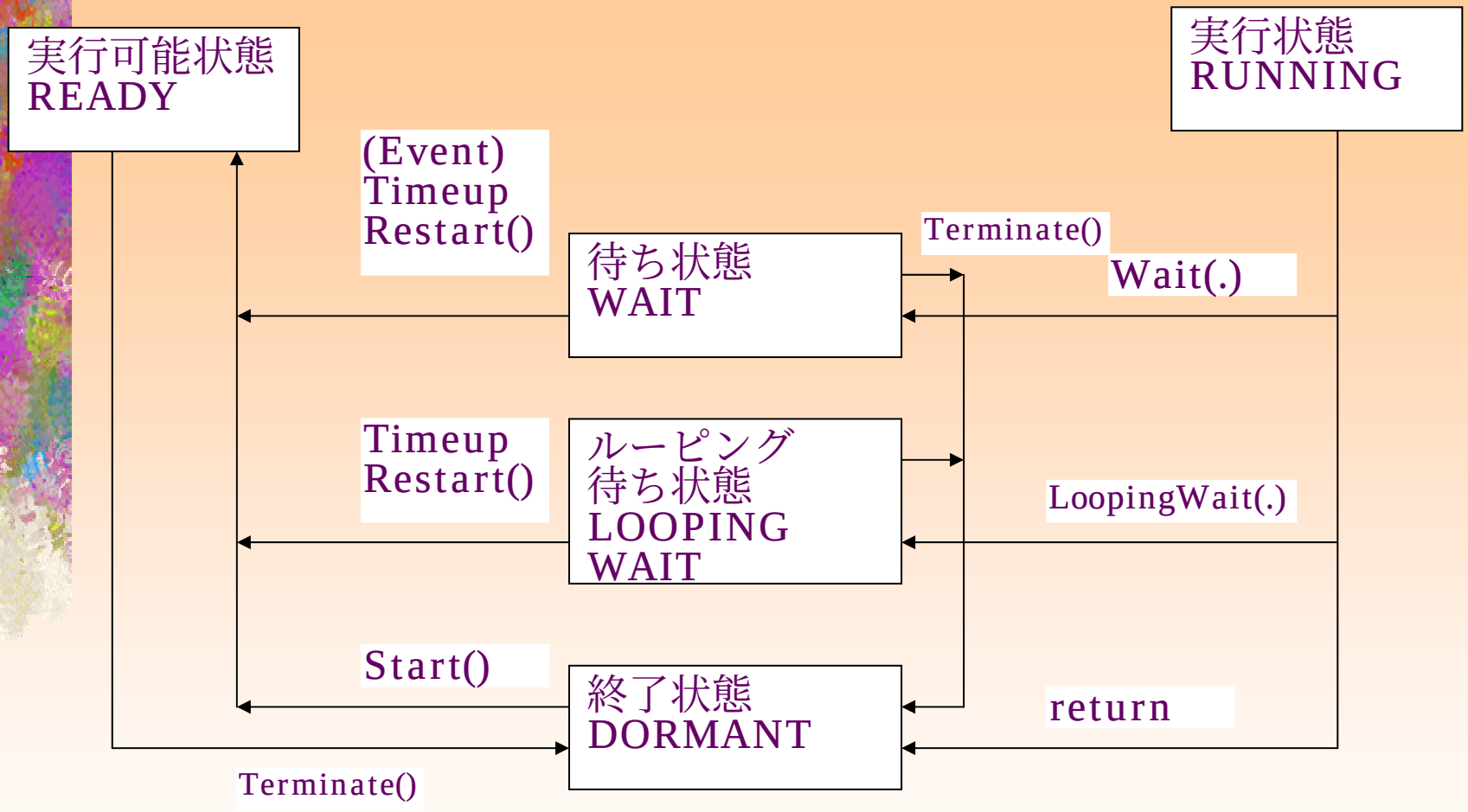
pre-emptive multi thread には無理があります
もっと co-operative multi thread の活用を！

Time Chart



Converted from Log __value data

Spec. thread, k-uOS 状態遷移



k-uOS はスタックを共用しながらcontext switch を実装

co-operative であることを**利用して単純化する**

最後に

ソースから独立したテスト、再利用可能な action scripts

使い捨てのデバッグ工数 → ソフト資産：詳細仕様

テストはテスターが主に書くべき

python による仕様記述・モデリングとテスト

関係者全員の総力によるカバレッジ100%テスト

組み込みソフトでも c/python kVerifier を

kVerifier ⊃ CppUnit

テストをソース・コード／プログラマから独立させよう！

Python でテストまで済ませてから C/C++ に移植しよう！

Download URL

http://www.nasuinfo.or.jp/FreeSpace/kenji

python sf

kVerifier