USB メモリの中身
20120404/
vw78jun793mac.zip
vw78jun793win.zip

今回は別の PDF ファイルへの書き込みが主

File Browser から Install.st を File in

KSU, KSU-High, High, Class を見ると、各クラスに対して名前がちゃんと書かれている
前回の復習から
資料参照(処理時間と清掃回数)

次は閉包

次は逐次・並行・約束

KSU, KSU-High, High, Class, page0, page00
page00
```
	"KSU.High page00."
	"準備"

	| aWindow aCollection aRectangle |
	aWindow := (aCollection := Transcript dependents) isEmpty
			ifTrue: [VisualLauncher open window]
			ifFalse: [aCollection first topComponent].
	aRectangle := aWindow displayBox.
	aRectangle := aRectangle translatedBy: (aRectangle origin - (50 @ 50)) negated.
	aRectangle := aRectangle origin extent: aRectangle width @ 400.
"処理の区切れ目としてここで改行を入れたい"
	aWindow displayBox: aRectangle.
	aWindow := JunLauncher launcherWindow ifNil: [JunLauncher install] ifNotNil: [:it | it yourself].
"処理の区切れ目としてここで改行を入れたい"
	aWindow collapse.
	aWindow := (ScheduledControllers scheduledControllers
			detect: [:aController | aController view label = 'Welcome to VisualWorks']
			ifNone: [nil]) ifNil: [nil] ifNotNil: [:aController | aController view].
	aWindow ifNotNil: [aWindow collapse]
```

でも、こうではなくて、資料の p3 の様に各々ブロッククロージャにして、書きましょう。

KSU, KSU-High, High, Class, page0, page00
page01
```
	"KSU.High page01."
	"逐次"

	| aClosure |
	Transcript clear.
	aClosure :=
		[:aString :howMany |
		howMany timesRepeat:
			[Transcript
				nextPutAll: aString;
				nextPutAll: ': ';
				nextPutAll: Time now printString;
				cr;
				flush.
			1000 milliseconds wait]].
	[aClosure value: 'Black' value: 2] value.
	[aClosure value: 'Red' value: 3] value.
	[aClosure value: 'Green' value: 3] value.
	[aClosure value: 'Blue' value: 3] value.
	[aClosure value: 'White' value: 2] value
```

Black: 20:18:55
Black: 20:18:56
Red: 20:18:57
Red: 20:18:58

```
Red: 20:18:59
Green: 20:19:00
Green: 20:19:01
Green: 20:19:02
Blue: 20:19:03
Blue: 20:19:04
Blue: 20:19:05
White: 20:19:06
White: 20:19:07
```

```
page02
	"KSU.High page02."
	"並行"

	| aClosure |
	Transcript clear.
	aClosure :=
		[:aString :howMany |
		howMany timesRepeat:
			[Transcript
				nextPutAll: aString;
				nextPutAll: ': ';
				nextPutAll: Time now printString;
				cr;
				flush.
			1000 milliseconds wait]].
	[aClosure value: 'Black' value: 2] value.
	[aClosure value: 'Red' value: 3] fork.
	[aClosure value: 'Green' value: 3] fork.
	[aClosure value: 'Blue' value: 3] fork.
	[aClosure value: 'White' value: 2] value
```

時刻に注目すると、White, Red, Green, Blue は同時に実行されている

```
Black: 20:19:30
Black: 20:19:31
White: 20:19:32
Red: 20:19:32
Green: 20:19:32
Blue: 20:19:32
White: 20:19:33
Red: 20:19:33
Green: 20:19:33
Blue: 20:19:33
Red: 20:19:34
Green: 20:19:34
Blue: 20:19:34
```

```
page03
	"KSU.High page03."
	"約束：並行と同じ"

	| aClosure |
	Transcript clear.
	aClosure :=
		[:aString :howMany |
		howMany timesRepeat:
			[Transcript
				nextPutAll: aString;
				nextPutAll: ': ';
				nextPutAll: Time now printString;
				cr;
				flush.
			1000 milliseconds wait]].
	[aClosure value: 'Black' value: 2] value.
	[aClosure value: 'Red' value: 3] promise.
	[aClosure value: 'Green' value: 3] promise.
	[aClosure value: 'Blue' value: 3] promise.
	[aClosure value: 'White' value: 2] value
```

上と同様に、同時に実行されている

```
Black: 20:24:01
Black: 20:24:02
White: 20:24:03
Red: 20:24:03
Green: 20:24:03
Blue: 20:24:03
White: 20:24:04
Red: 20:24:04
Green: 20:24:04
Blue: 20:24:04
Red: 20:24:05
Green: 20:24:05
Blue: 20:24:05


page04
	"KSU.High page04."
	"約束：親が子を待ち合わせる"

	| aClosure redPromise greenPromise bluePromise |
	Transcript clear.
	aClosure :=
			[:aString :howMany |
			howMany timesRepeat:
				[Transcript
					nextPutAll: aString;
					nextPutAll: ': ';
					nextPutAll: Time now printString;
					cr;
					flush.
				1000 milliseconds wait]].
	[aClosure value: 'Black' value: 2] value.
	redPromise := [aClosure value: 'Red' value: 3] promise.
	greenPromise := [aClosure value: 'Green' value: 3] promise.
	bluePromise := [aClosure value: 'Blue' value: 3] promise.
"下記の Promise が実行されるまで待ち合わせる"
	redPromise value.
	greenPromise value.
	bluePromise value.
	[aClosure value: 'White' value: 2] value

Black: 20:24:25
Black: 20:24:26
Red: 20:24:27
Green: 20:24:27
Blue: 20:24:27
Red: 20:24:28
Green: 20:24:29
Blue: 20:24:29
Red: 20:24:30
Green: 20:24:30
Blue: 20:24:30
White: 20:24:31 ← Red, Green, Blue が終わった後に実行されている
White: 20:24:32


page05
	"KSU.High page05."
	"約束：親が子を待ち合わせる間に値のやり取り"

	| aBlock aClosure redPromise greenPromise bluePromise |
	Transcript clear.
	aBlock :=
			[:aString |
			Transcript
				nextPutAll: aString;
				nextPutAll: ': ';
				nextPutAll: Time now printString;
				cr;
				flush].
	aClosure := [:aString :howMany | howMany timesRepeat:
```

```
                [aBlock value: aString.
                1000 milliseconds wait]].
    redPromise := greenPromise := bluePromise := nil.
    [aClosure value: 'Black' value: 2] value.
    redPromise :=
            [redPromise value: '赤'."実行前に約束は果たした"
            aClosure value: 'Red' value: 3] promise.
    greenPromise :=
            [aClosure value: 'Green' value: 1.
            greenPromise value: '緑'."1回実行した後に約束は果たした"
            aClosure value: 'Green' value: 2] promise.
    bluePromise :=
            [aClosure value: 'Blue' value: 2.
            bluePromise value: '青'."2回実行した後に約束は果たした"
            aClosure value: 'Blue' value: 1] promise.
    aBlock value: redPromise value.
    aBlock value: greenPromise value.
    aBlock value: bluePromise value.
    [aClosure value: 'White' value: 2] value
```

Black: 20:29:31
Black: 20:29:32
Red: 20:29:33
Green: 20:29:33
Blue: 20:29:33
赤: 20:29:33 ← 赤が先に来そうだが Red が上に来ているのはスケジューラの問題であるだけ(non-preentive で実行権を奪えないため)
Red: 20:29:34
Green: 20:29:34
Blue: 20:29:34
緑: 20:29:34
Red: 20:29:35
Green: 20:29:35
Blue: 20:29:35
青: 20:29:35 ← Blue が2回実行されたので出てきた
White: 20:29:35 ← 全ての約束が果たされたので実行開始(青と時刻が同じであることに注目)
White: 20:29:36

京都産業大学　コンピュータ理工学部　CS・NM・IS学科　特別研究
２０１２年２月２２日(水)　第４・５時限　６５実験室

処理時間と清掃回数

(マイクロ秒とスキャベンジ)

青木 淳
aokisan@qb3.so-net.ne.jp
http://www.cc.kyoto-su.ac.jp/~atsushi/
研究室： 第２実験室棟 ３階 ７３研究室

整数演算と浮動小数点数演算

５億回の比較演算と加算演算そして束縛

整数演算

```
| aClosure |
aClosure :=
          [| result |
          result := 1.
          [result <= 500000000] whileTrue: [result := result + 1].
          result yourself].
^aClosure value
```

浮動小数点数演算

```
| aClosure |
aClosure :=
          [| result |
          result := 1.0d.
          [result <= 500000000.0d] whileTrue: [result := result + 1.0d].
          result yourself].
^aClosure value
```

処理時間

Timeというクラスへ microsecondsToRun: aBlockというメッセージを送る

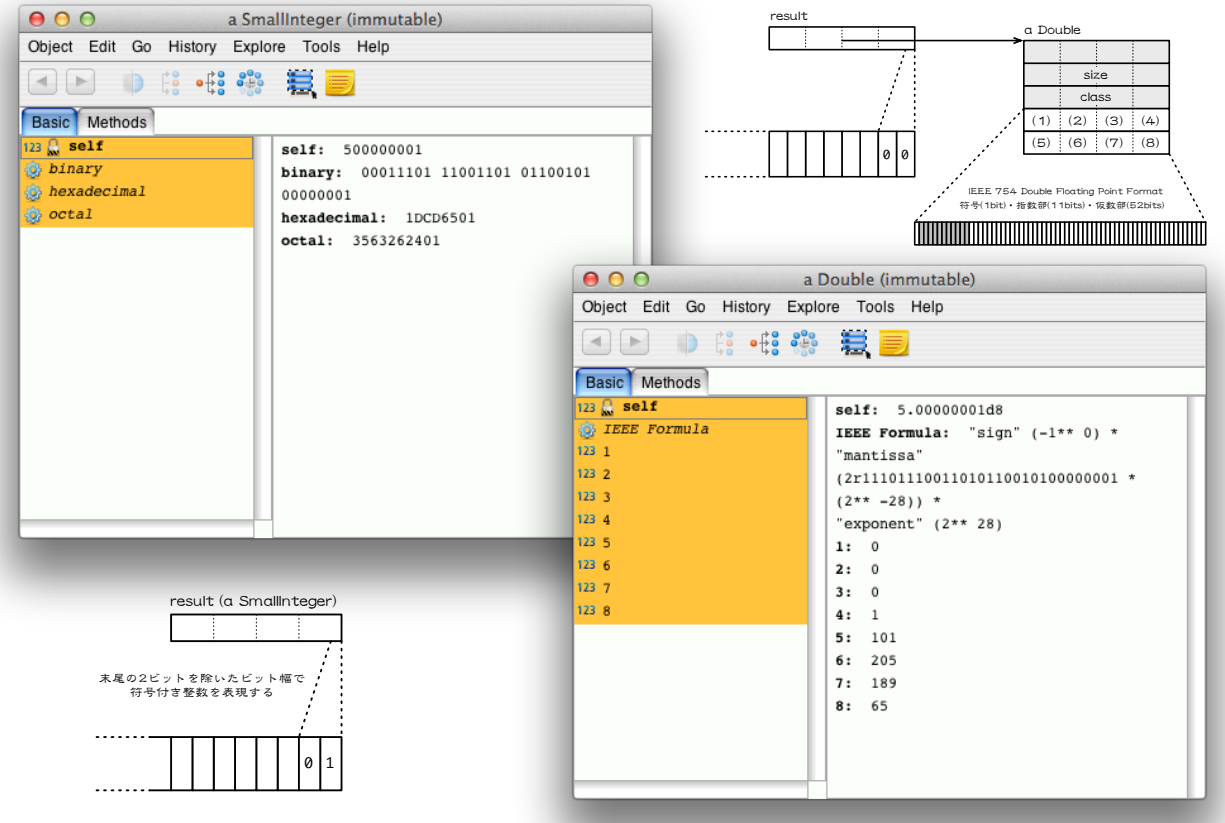
整数演算

```
| aClosure |
aClosure :=
          [| result |
          result := 1.
          [result <= 500000000] whileTrue: [result := result + 1].
          result yourself].
^Time microsecondsToRun: [aClosure value]
```

浮動小数点数演算

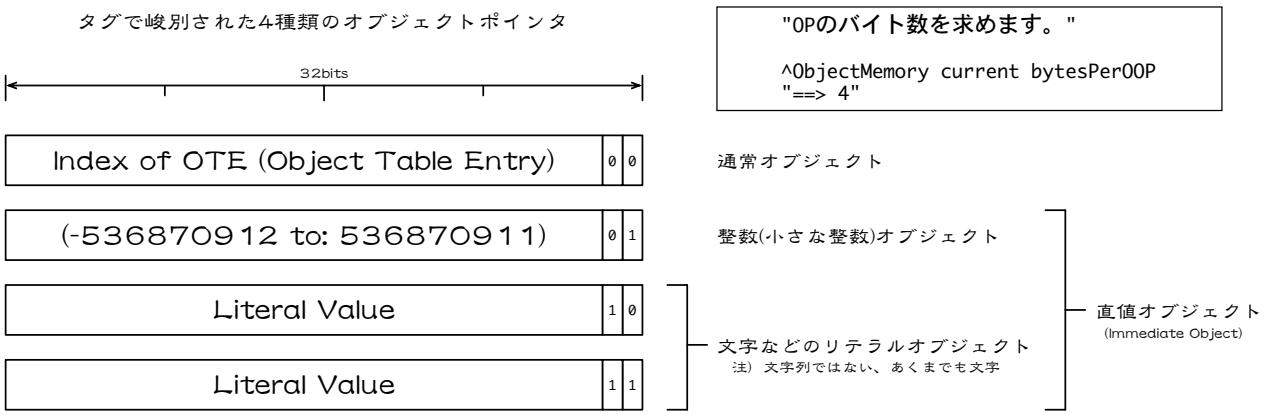
```
| aClosure |
aClosure :=
          [| result |
          result := 1.0d.
          [result <= 500000000.0d] whileTrue: [result := result + 1.0d].
          result yourself].
^Time microsecondsToRun: [aClosure value]
```


整数の構造と浮動小数点数の構造



オブジェクトの構造(1)

まず、オブジェクトポインタ(OP(またはOOP))から...

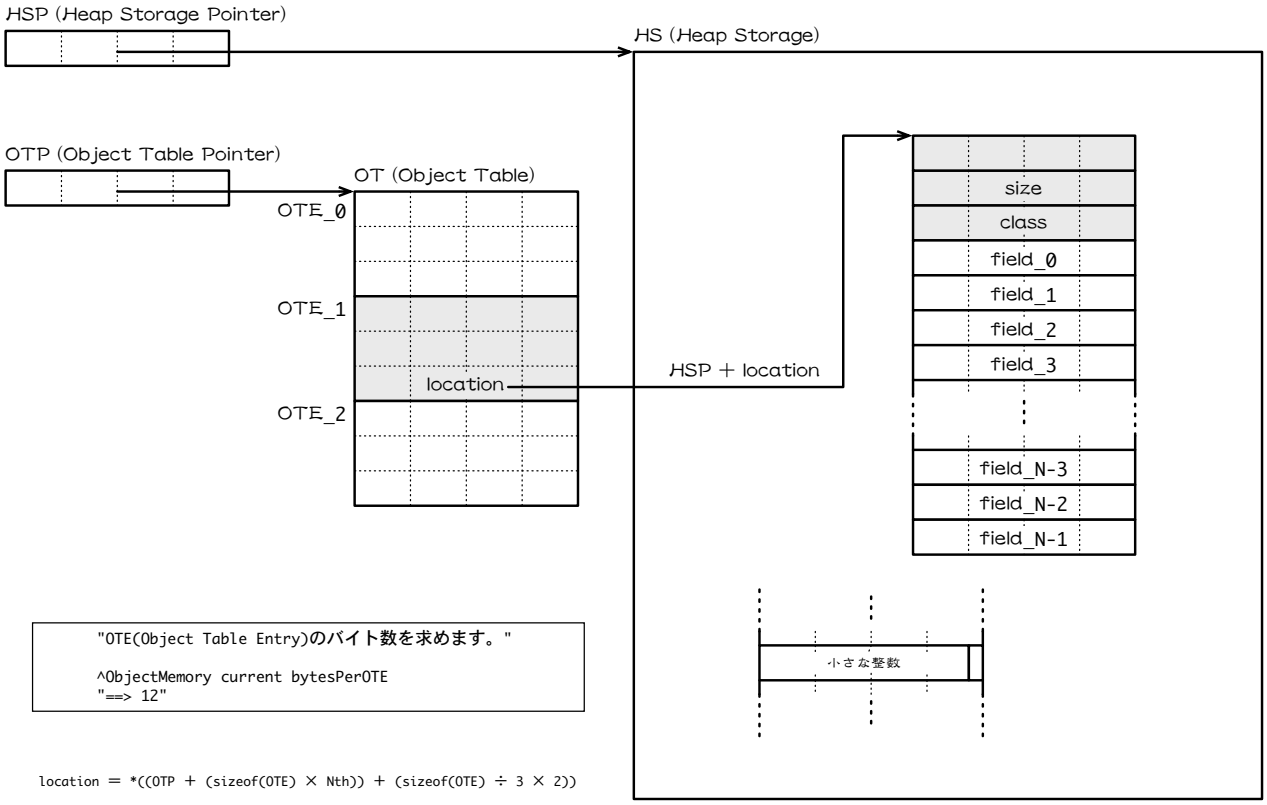



```
"小さな整数(SmallInteger)の範囲を求めます。"

| value |
value := -1.
[(value subtractOrFail: 1) notNil] whileTrue: [value := value + value].
^value to: (value + 1) negated
"==> (-536870912 to: 536870911)"
```

昔は16bits(タグ1bit)
現在は32bits(タグ2bits)から64bits(タグ3bits)へ移行完了
さらに64bitsから128bits(タグ4bits)への移行が試みられている

オブジェクトの構造(2)

次に、オブジェクトテーブル...

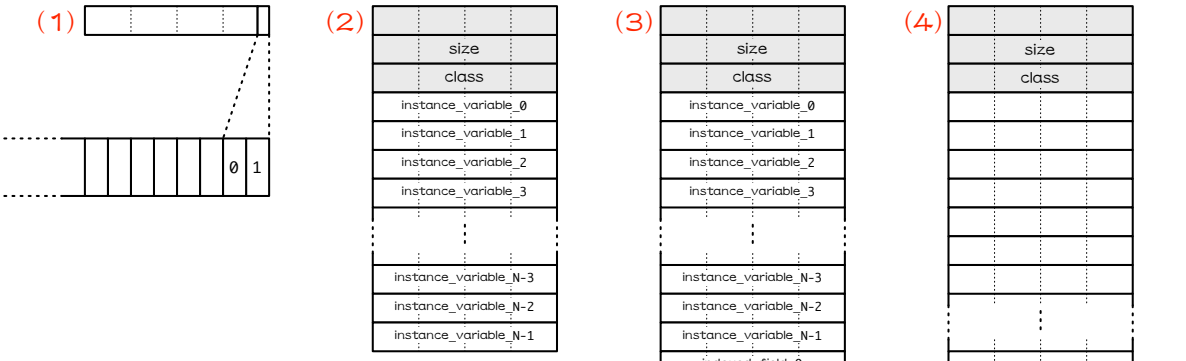



```
"OTE(Object Table Entry)のバイト数を求めます。"

^ObjectMemory current bytesPerOTE
"==> 12"
```

location = *((OTP + (sizeof(OTE) × Nth)) + (sizeof(OTE) ÷ 3 × 2))

オブジェクトの構造(3)

そして、フィールドの使い方によるオブジェクトの根源的な型...




```
"subclass : anObjectが非ポインタ非インデックス型(イミディエイト型)"

| anObject aClass |
anObject := 123.
aClass := anObject class.
^aClass isBits and: [aClass isFixed]
"==> true"
```

(1)

```
"subclass : anObjectがポインタ非インデックス型"

| anObject aClass |
anObject := Point x: 100 y: 200.
aClass := anObject class.
^aClass isPointers and: [aClass isFixed]
"==> true"
```

(2)

```
"variableSubclass : anObjectがポインタインデックス型"

| anObject aClass |
anObject := OrderedCollection new.
aClass := anObject class.
^aClass isPointers and: [aClass isVariable]
"==> true"
```

(3)

```
"variableByteSubclass : anObjectが非ポインタインデックス型"

| anObject aClass |
anObject := 123.0d.
aClass := anObject class.
^aClass isBits and: [aClass isVariable]
"==> true"
```

(4)

バイトコード

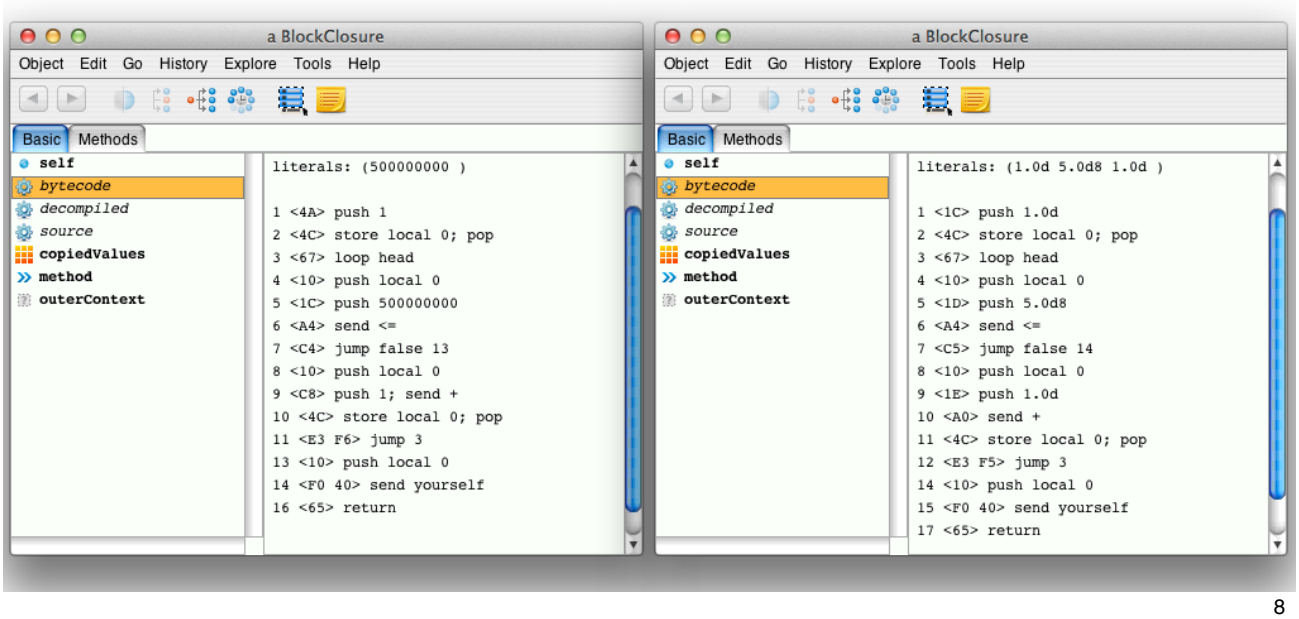
バイト幅のスタックベースの命令群(256種類)を仮想マシンが食べる

整数演算

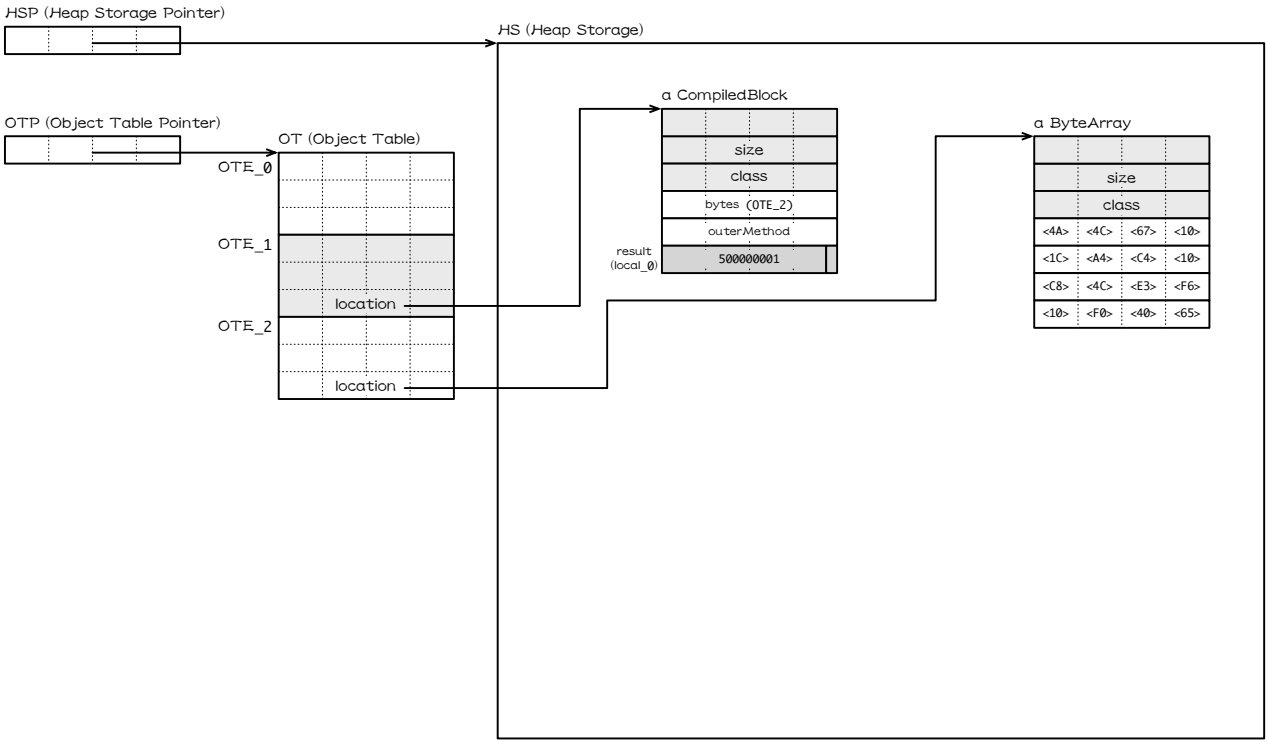
```
| aClosure |
aClosure :=
        [| result |
        result := 1.
        [result <= 500000000] whileTrue: [result := result + 1].
        result yourself].
^aClosure
```

浮動小数点数演算

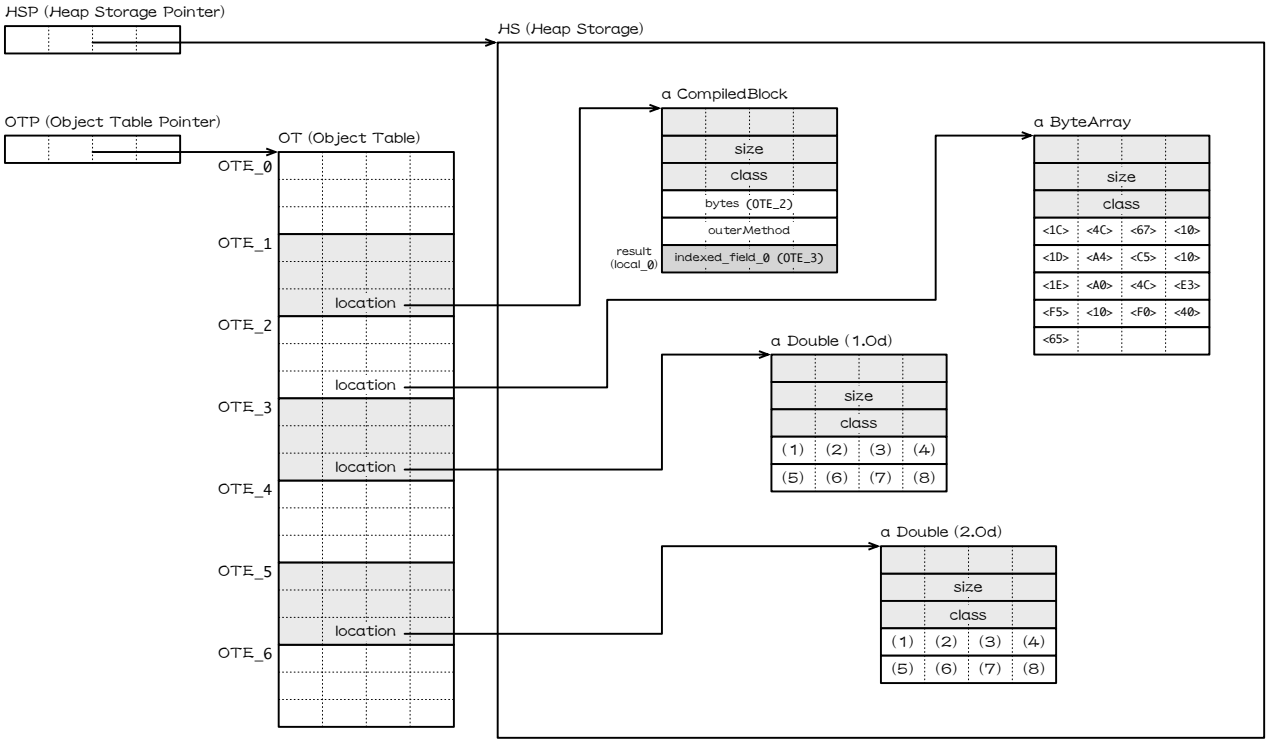
```
| aClosure |
aClosure :=
        [| result |
        result := 1.0d.
        [result <= 500000000.0d] whileTrue: [result := result + 1.0d].
        result yourself].
^aClosure
```



整数演算のオブジェクトメモリ



浮動小数点数演算のオブジェクトメモリ



清掃回数

ObjectMemoryのインスタンスへnumScavengesというメッセージを送る

整数演算

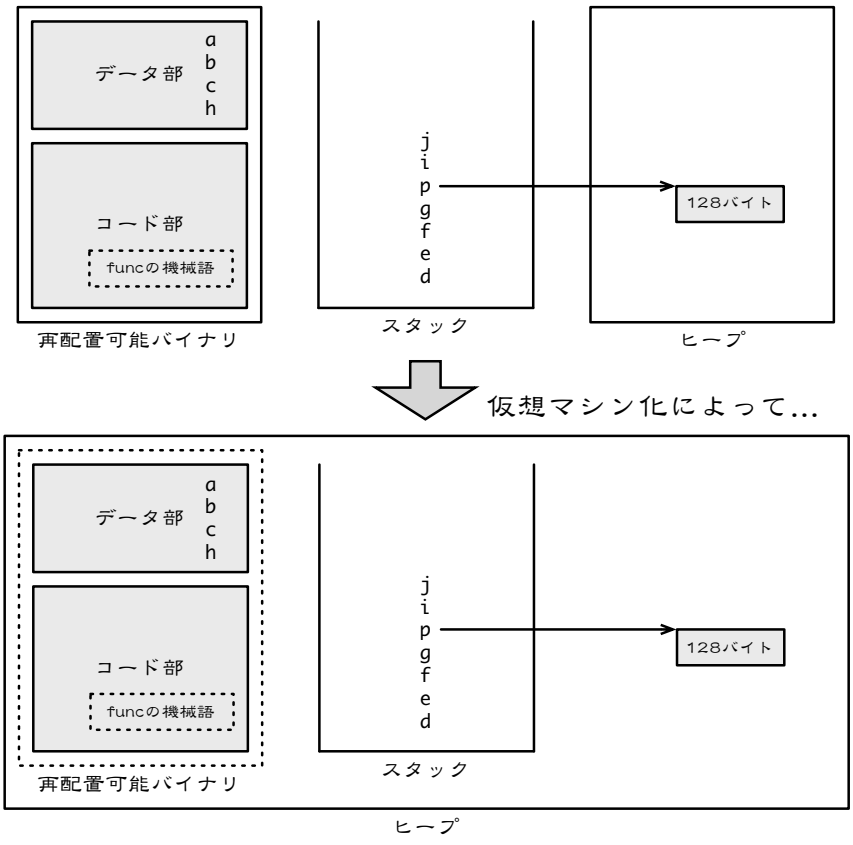
```
| aClosure aValue |
aClosure :=
          [| result |
          result := 1.
          [result <= 500000000] whileTrue: [result := result + 1].
          result yourself].
aValue := ObjectMemory current numScavenges.
^(Time microsecondsToRun: [aClosure value])
     -> (ObjectMemory current numScavenges - aValue)
```

浮動小数点数演算

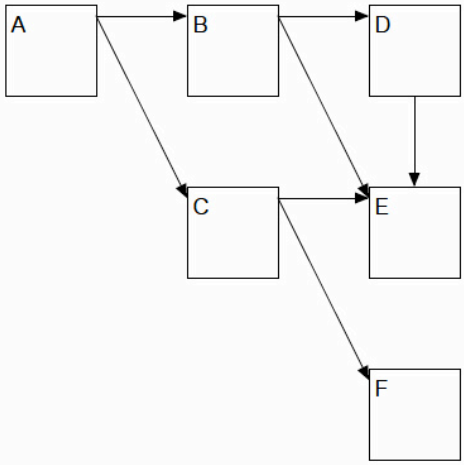
```
| aClosure aValue |
aClosure :=
          [| result |
          result := 1.0d.
          [result <= 500000000.0d] whileTrue: [result := result + 1.0d].
          result yourself].
aValue := ObjectMemory current numScavenges.
^(Time microsecondsToRun: [aClosure value])
     -> (ObjectMemory current numScavenges - aValue)
```


すべてをヒープ領域へ

```
    :

int a;
static int b;
extern int c;

    :

void func(int d, int e, int f)
{
    int g;
    static int h;
    int* p;

    :

    while(true)
    {
        int i;
        int j;

        :

    }

    :

    p = (int*)malloc(128);

    return;
}

    :
```

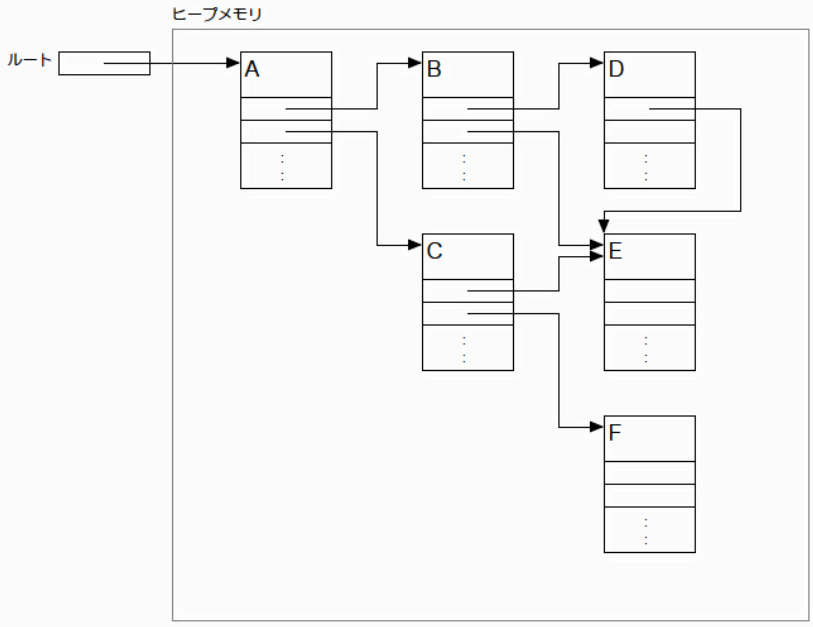


直接と間接



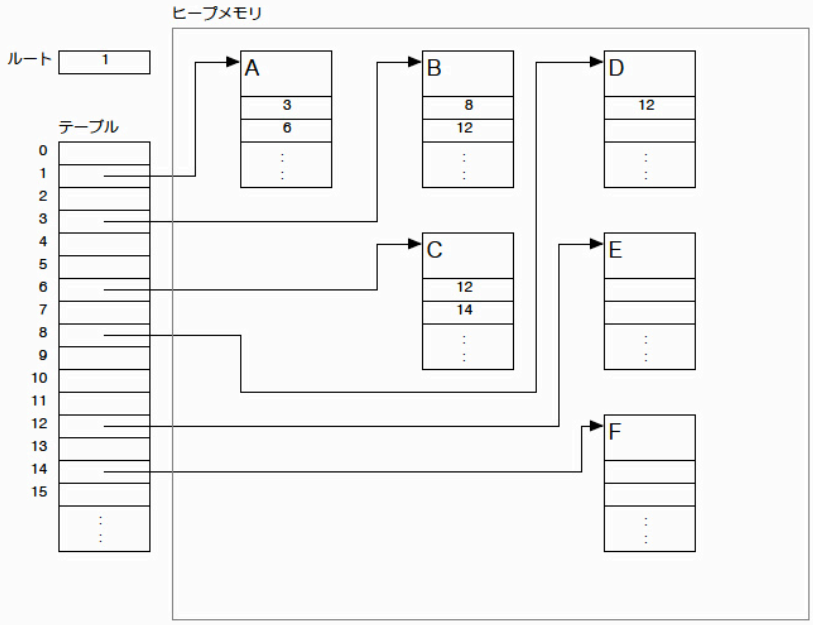
直接と間接

Direct Addressing（直接アドレッシング）



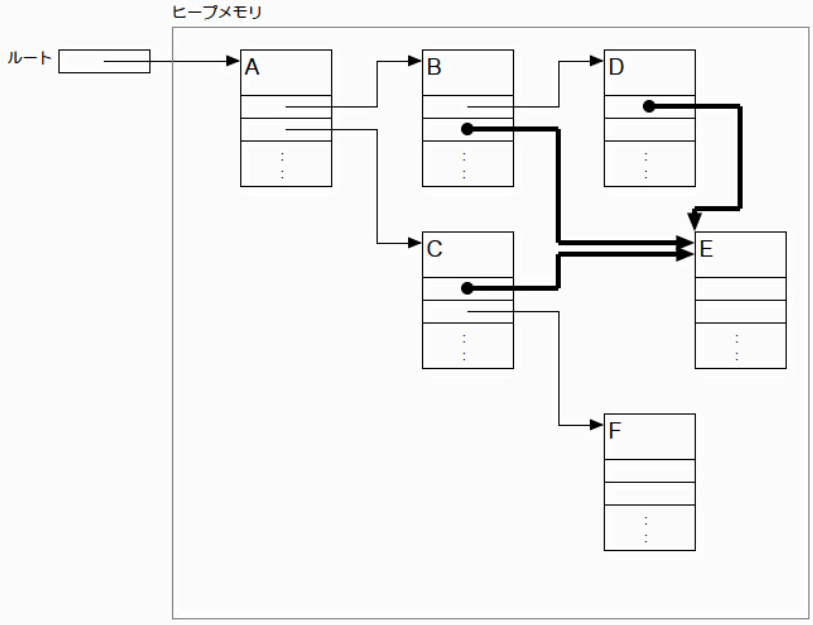
直接と間接

Indirect Addressing（間接アドレッシング）



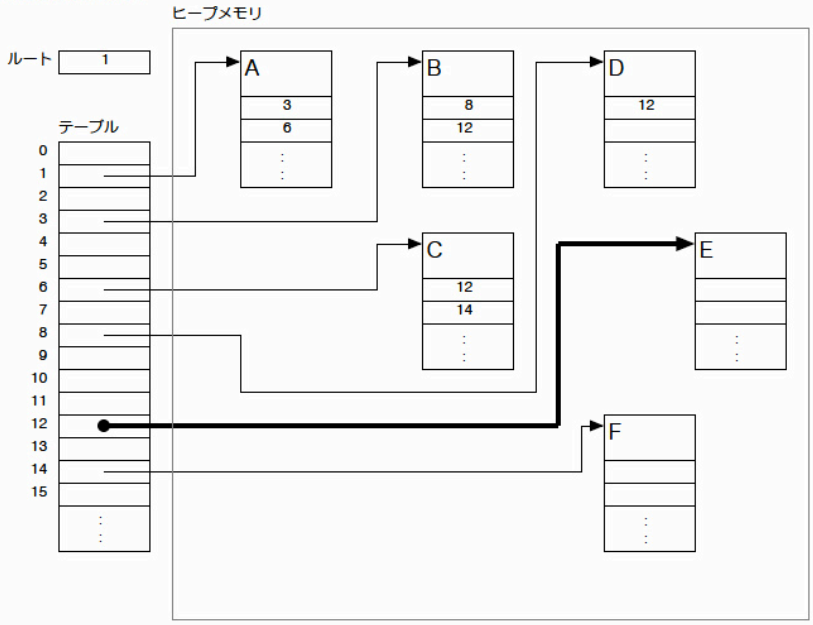
引っ越し

Direct Addressing（直接アドレッシング）
Eのアドレスが変わった！



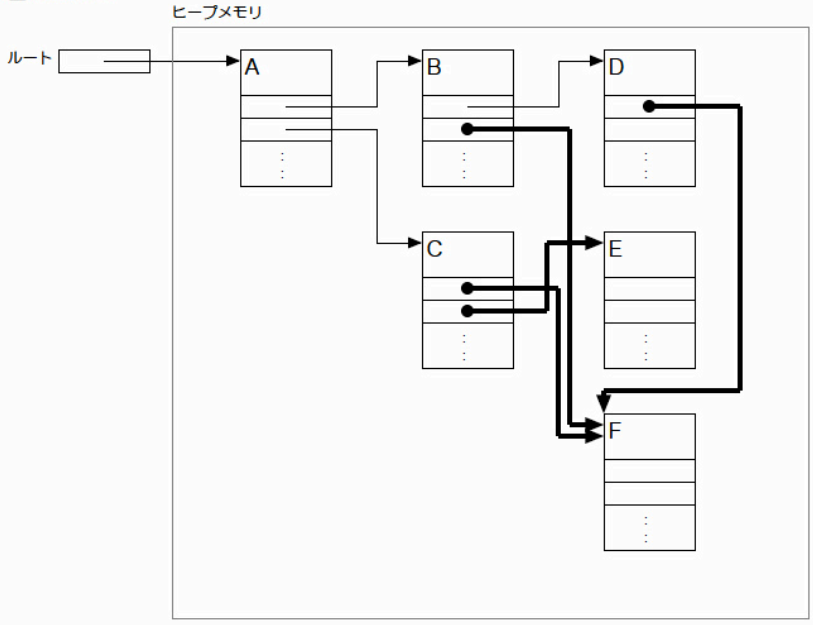
引っ越し

Indirect Addressing（間接アドレッシング）
Eのアドレスが変わった！



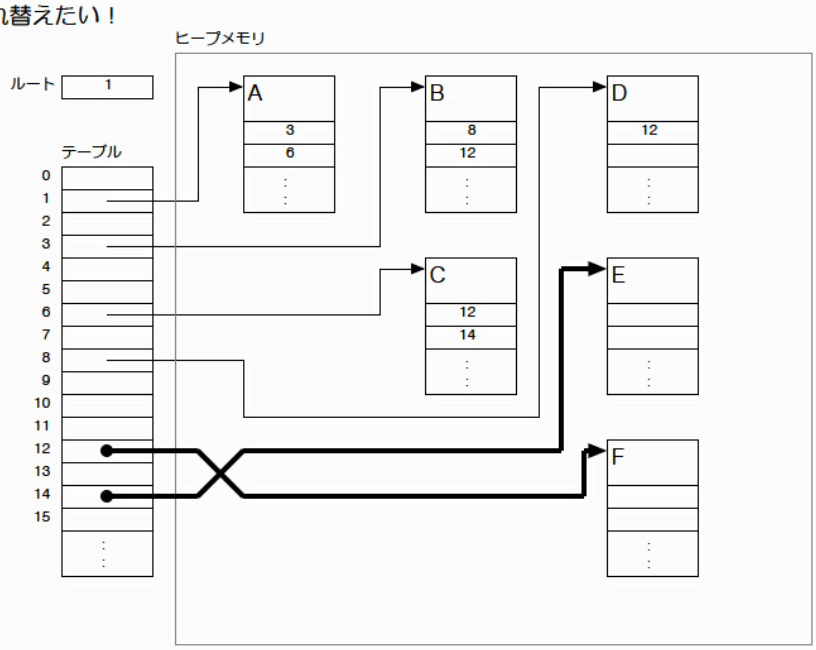
入れ替え

Direct Addressing（直接アドレッシング）
EとFを入れ替えたい！



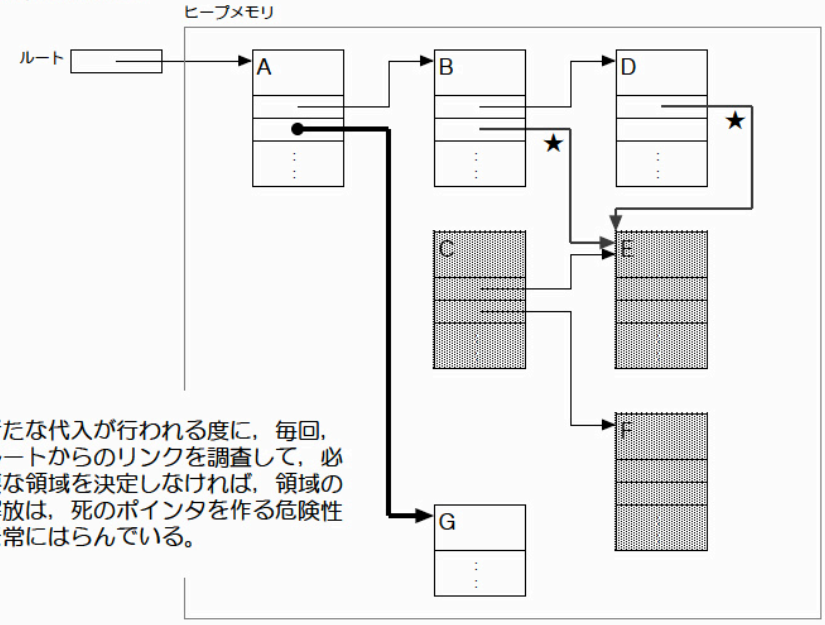
入れ替え

Indirect Addressing（間接アドレッシング）
EとFを入れ替えたい！



死亡と誕生

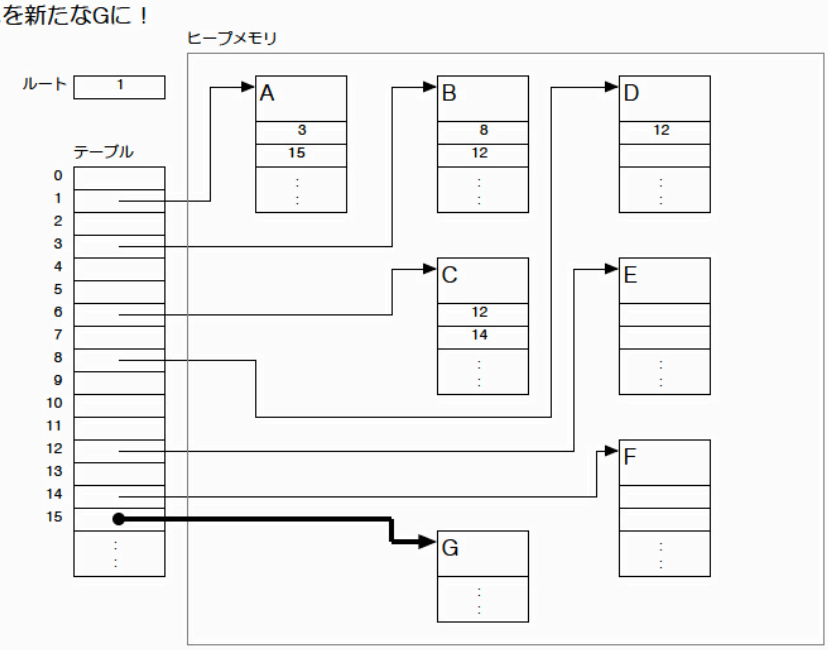
Direct Addressing（直接アドレッシング）
AからCへを新たなGに！



新たな代入が行われる度に，毎回，ルートからのリンクを調査して，必要な領域を決定しなければ，領域の解放は，死のポインタを作る危険性を常にはらんでいる。

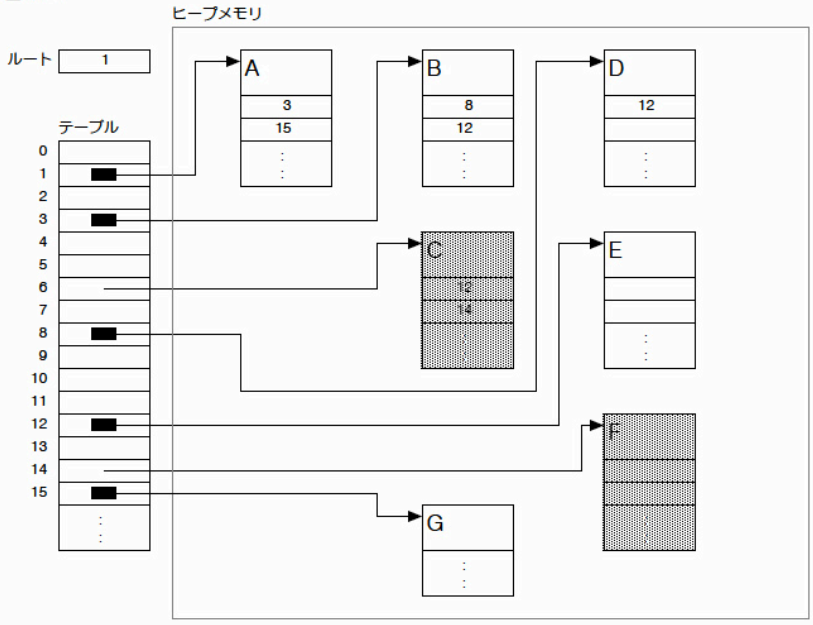
死亡と誕生

Indirect Addressing（間接アドレッシング）
AからCへを新たなGに！



死亡と誕生

Indirect Addressing（間接アドレッシング）
自動メモリ管理！



京都産業大学　コンピュータ理工学部　CS・NM・IS学科　特別研究
２０１２年２月２２日(水)　第４・５時限　６５実験室

閉包

（クロージャ：Smalltalkのブロッククロージャ）

青木 淳
aokisan@qb3.so-net.ne.jp
http://www.cc.kyoto-su.ac.jp/~atsushi/
研究室： 第２実験室棟 3階 73研究室

ブラケット(角形括弧)で囲う

```
3 + 4
```

パレンテシス(丸形括弧)で囲う

```
(3 + 4) inspect
```

ブラケット(角形括弧)で囲う

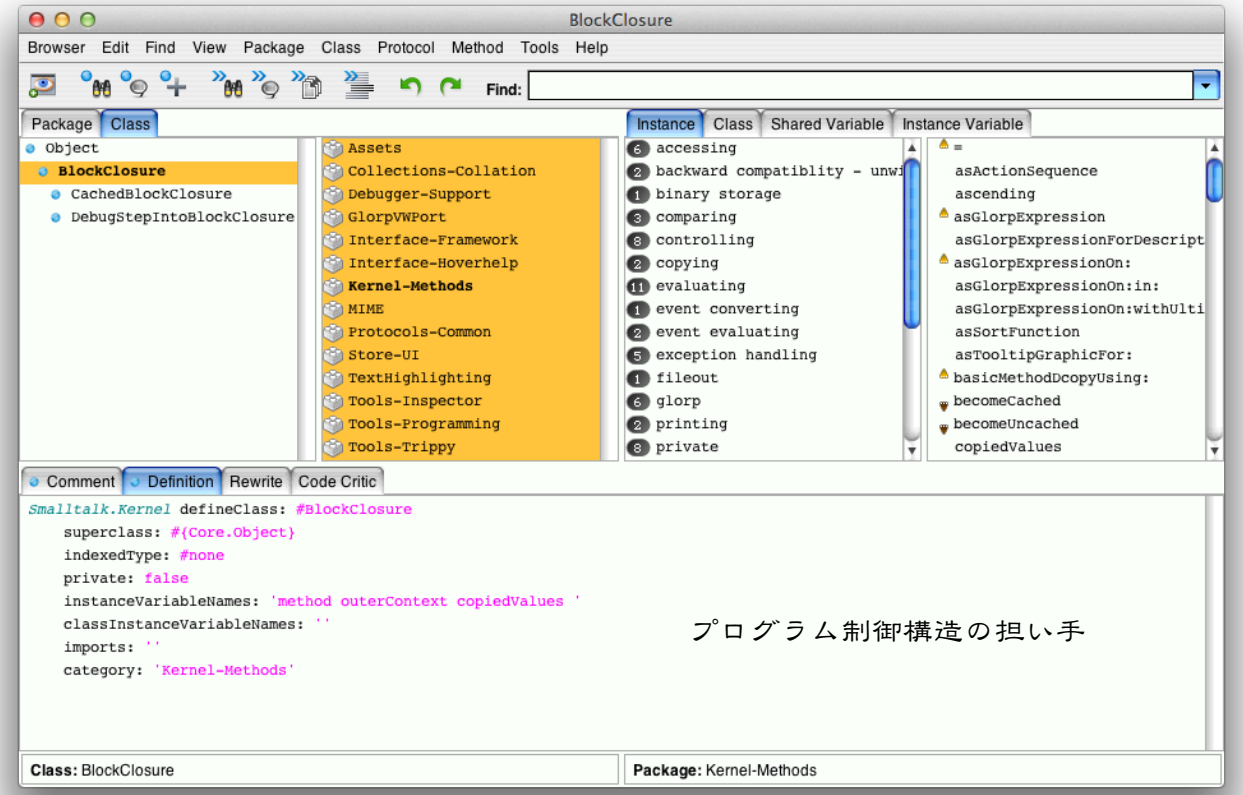
```
[3 + 4] inspect
```

```
[3 + 4] value inspect
```

```
([:x | x + 4] value: 3) inspect
```

```
([:x :y | x + y] value: 3 value: 4) inspect
```

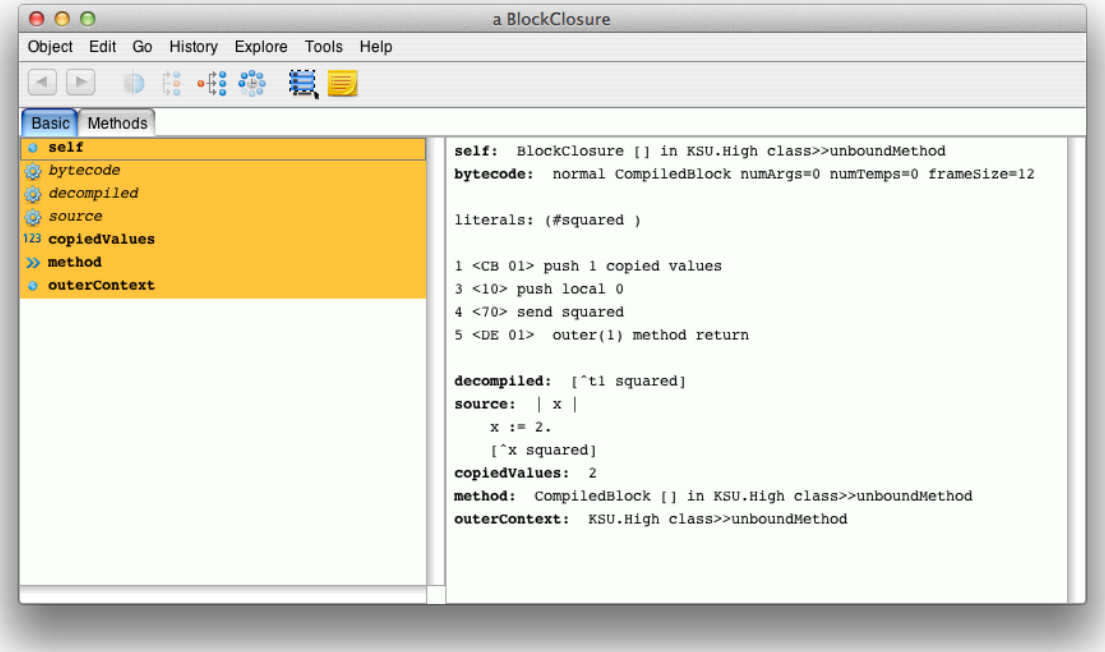
BlockClosure（クラス）



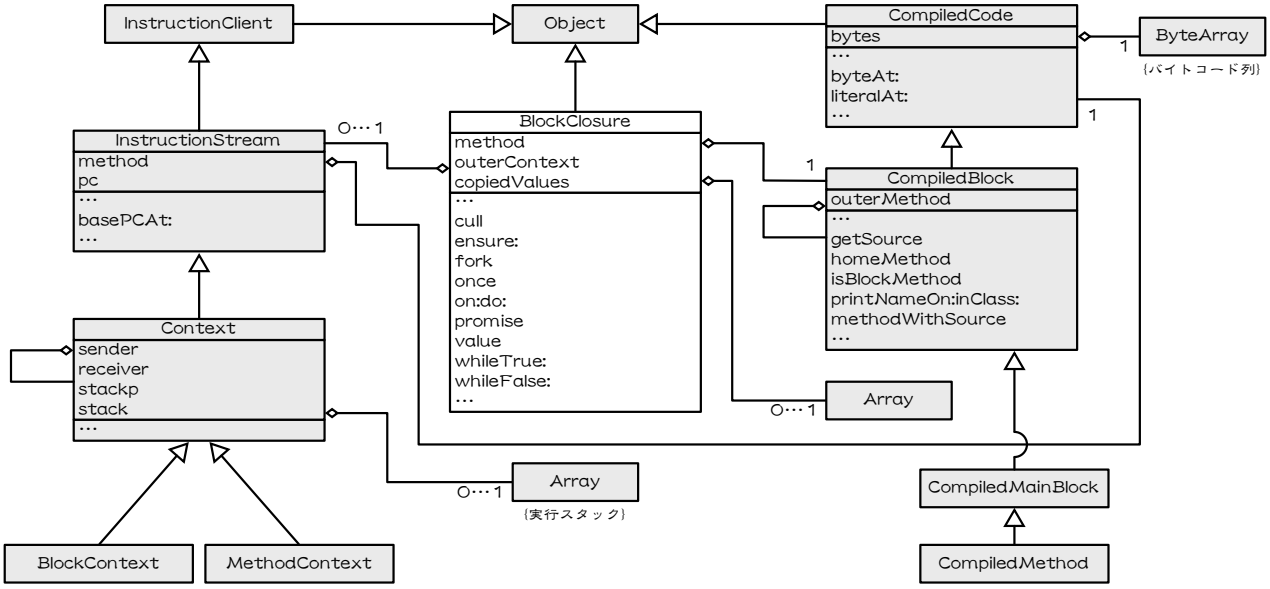
プログラム制御構造の担い手

BlockClosure（インスタンス）

```
| x |
x := 2.
[^x squared]
```



BlockClosureの構造



メソッドはクロージャ(コンパイルドコードとコンテクストの閉包)として処理される

クロージャごとにバイトコード列・スタック・リテラルフレームがある
1本の線形なスタックではない、メソッドが起動されるたびにスタックが作られ、コンテクストのチェーンが形成される

BlockClosureの分類

		copiedValues (外部文脈のリテラルを複製)	
		nilまたはempty	それ以外
outerContext (外部文脈を束縛)	nil	Clean (and faster)	Copying (not clean)
	それ以外	Full (not clean)	Copying / Full (not clean)

```
"Clean (and faster) : (copiedValues isEmpty) AND (outerContext isNil)"

| result |
result := [:x | x squared] value: 2.
^result
```

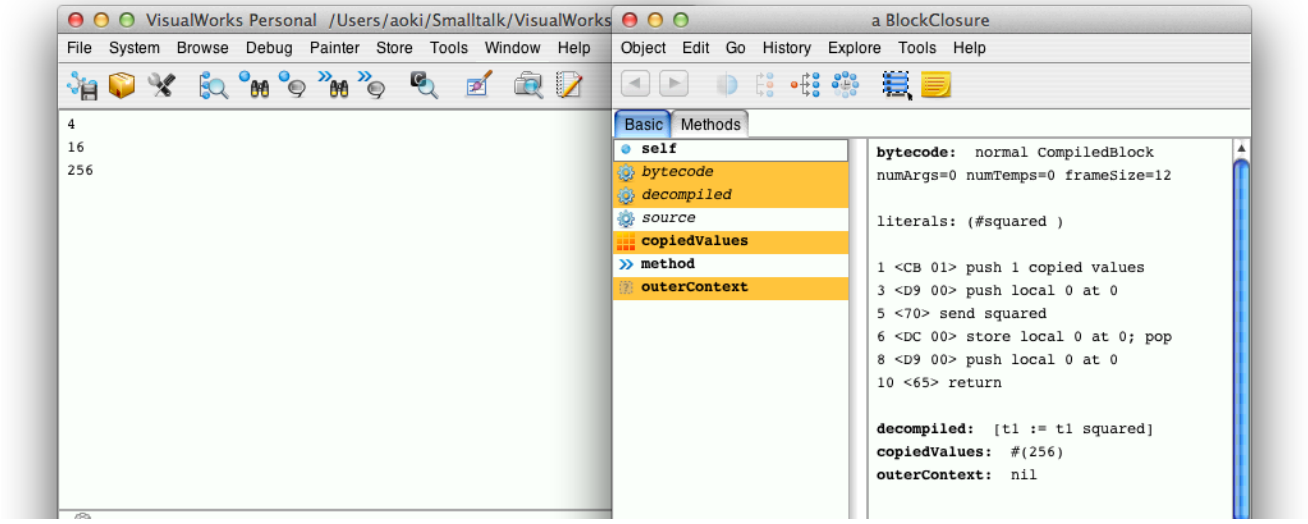
```
"Full : (copiedValues isNil) AND (outerContext notNil)"

[:x | ^x squared] value: 2
```

```
"Copying : (copiedValues notNil) AND (outerContext isNil)"

| x |
x := 2.
^[x squared] value
```

```
"Copying / Full : (copiedValues notNil) AND (outerContext notNil)"

| x |
x := 2.
[^x squared] value
```

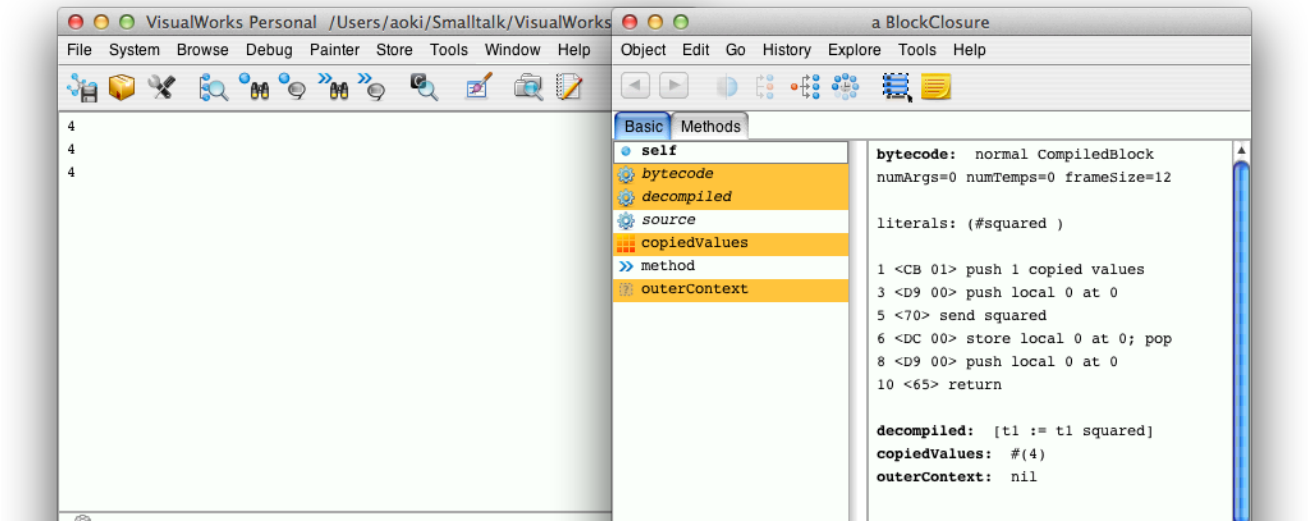
BlockClosureの評価

```
| continuation x closure |
Transcript clear.
continuation :=
          [:v |
          Transcript
               nextPutAll: v printString;
               cr;
               flush].
x := 2.
closure := [x := x squared]
               inspect;
               yourself.
3 timesRepeat: [continuation value: closure value]
```



BlockClosureの１度限りの評価

```
| continuation x closure |
Transcript clear.
continuation :=
          [:v |
          Transcript
               nextPutAll: v printString;
               cr;
               flush].
x := 2.
closure := [x := x squared]
               inspect;
               yourself.
3 timesRepeat: [continuation value: closure once]
```



京都産業大学　コンピュータ理工学部　CS・NM・IS学科　特別研究
２０１２年２月２２日(水)　第４・５時限　６５実験室

逐次・並行・約束

(シーケンス・コンカレンス・プロミス)

青木 淳
aokisan@qb3.so-net.ne.jp
http://www.cc.kyoto-su.ac.jp/~atsushi/
研究室： 第２実験室棟 3階 73研究室

value・fork・promise

逐次 (シーケンス)	並行 (コンカレンス)	約束 (プロミス)
[…A…] value.	[…A…] fork.	[…A…] promise.
[…B…] value.	[…B…] fork.	[…B…] promise.
[…C…] value.	[…C…] fork.	[…C…] promise.
[…D…] value.	[…D…] fork.	[…D…] promise.
[…E…] value	[…E…] fork	[…E…] promise

Smalltalk仮想マシンにおいて、アプリケーションは並列に実行される複数のスレッド(ライトウェイトプロセス)を使用することができます。

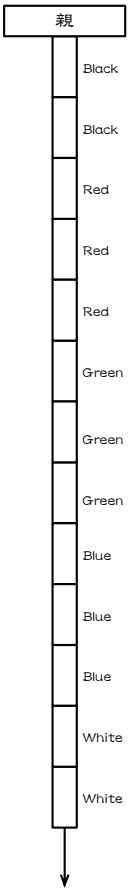
Thread (a nonpreemptive light-weight process) represents an independent path of control in the system. This path of control may be stopped (by sending the instance the message suspend) in such a way that it can later be restarted (by sending the instance the message resume).

いずれも閉包(クロージャ)が担い手

以下のプログラムの中にBlockClosureはいくつあるか？

```
[| aWindow aCollection aRectangle |
aWindow := (aCollection := Transcript dependents) isEmpty
               ifTrue: [VisualLauncher open window]
               ifFalse: [aCollection first topComponent].
aRectangle := aWindow displayBox.
aRectangle := aRectangle translatedBy: (aRectangle origin - (50 @ 50)) negated.
aRectangle := aRectangle origin extent: aRectangle width @ 400.
aWindow displayBox: aRectangle]
          value.    "逐次"

[| aWindow |
aWindow := JunLauncher launcherWindow ifNil: [JunLauncher install] ifNotNil: [:it | it yourself].
aWindow collapse]
          fork.     "並行"

[| aWindow |
aWindow := (ScheduledControllers scheduledControllers
               detect: [:aController | aController view label = 'Welcome to VisualWorks']
               ifNone: [nil]) ifNil: [nil] ifNotNil: [:aController | aController view].
aWindow ifNotNil: [aWindow collapse]]
          promise   "約束"
```

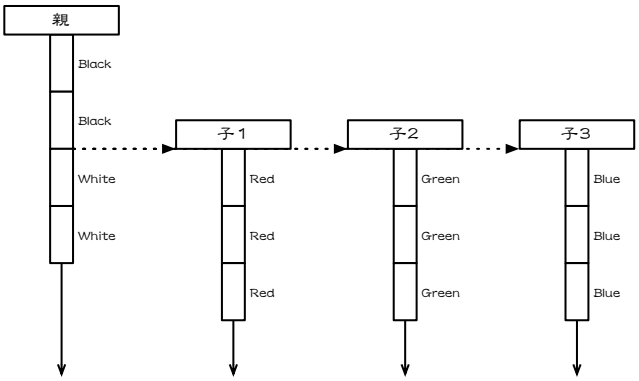

逐次




```
| aClosure |
Transcript clear.
aClosure :=
            [:aString :howMany |
            howMany timesRepeat:
                        [Transcript
                              nextPutAll: aString;
                              nextPutAll: ': ';
                              nextPutAll: Time now printString;
                              cr;
                              flush.
                        1000 milliseconds wait]].
[aClosure value: 'Black' value: 2] value.
[aClosure value: 'Red' value: 3] value.
[aClosure value: 'Green' value: 3] value.
[aClosure value: 'Blue' value: 3] value.
[aClosure value: 'White' value: 2] value
```

```
Black: 11:46:44
Black: 11:46:45
Red: 11:46:46
Red: 11:46:47
Red: 11:46:48
Green: 11:46:49
Green: 11:46:50
Green: 11:46:51
Blue: 11:46:52
Blue: 11:46:53
Blue: 11:46:54
White: 11:46:55
White: 11:46:56
```

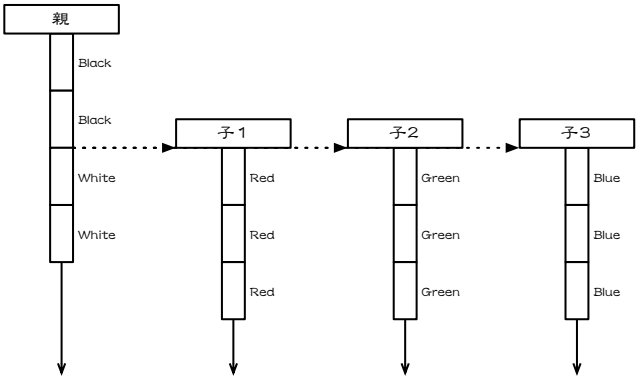
並行




```
| aClosure |
Transcript clear.
aClosure :=
            [:aString :howMany |
            howMany timesRepeat:
                        [Transcript
                              nextPutAll: aString;
                              nextPutAll: ': ';
                              nextPutAll: Time now printString;
                              cr;
                              flush.
                        1000 milliseconds wait]].
[aClosure value: 'Black' value: 2] value.
[aClosure value: 'Red' value: 3] fork.
[aClosure value: 'Green' value: 3] fork.
[aClosure value: 'Blue' value: 3] fork.
[aClosure value: 'White' value: 2] value
```

```
Black: 11:49:57
Black: 11:49:58
White: 11:49:59
Red: 11:49:59
Green: 11:49:59
Blue: 11:49:59
White: 11:50:00
Red: 11:50:00
Green: 11:50:00
Blue: 11:50:00
Red: 11:50:01
Green: 11:50:01
Blue: 11:50:01
```

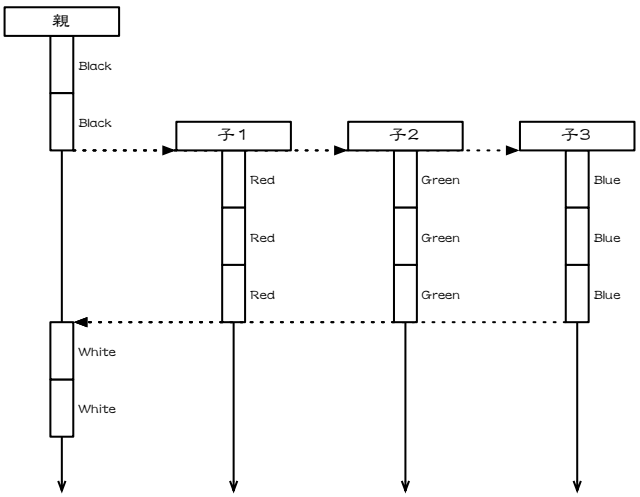

約束(1)




```
| aClosure |
Transcript clear.
aClosure :=
            [:aString :howMany |
            howMany timesRepeat:
                        [Transcript
                              nextPutAll: aString;
                              nextPutAll: ': ';
                              nextPutAll: Time now printString;
                              cr;
                              flush.
                        1000 milliseconds wait]].
[aClosure value: 'Black' value: 2] value.
[aClosure value: 'Red' value: 3] promise.
[aClosure value: 'Green' value: 3] promise.
[aClosure value: 'Blue' value: 3] promise.
[aClosure value: 'White' value: 2] value
```

```
Black: 11:49:57
Black: 11:49:58
White: 11:49:59
Red: 11:49:59
Green: 11:49:59
Blue: 11:49:59
White: 11:50:00
Red: 11:50:00
Green: 11:50:00
Blue: 11:50:00
Red: 11:50:01
Green: 11:50:01
Blue: 11:50:01
```

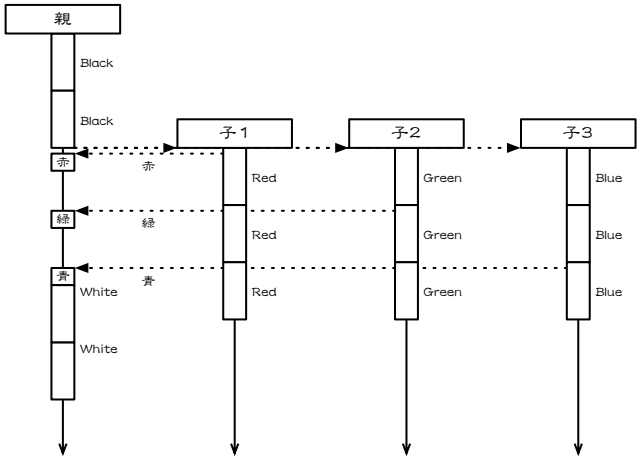
約束(2)




```
| aClosure redPromise greenPromise bluePromise |
Transcript clear.
aClosure :=
            [:aString :howMany |
            howMany timesRepeat:
                        [Transcript
                              nextPutAll: aString;
                              nextPutAll: ': ';
                              nextPutAll: Time now printString;
                              cr;
                              flush.
                        1000 milliseconds wait]].
[aClosure value: 'Black' value: 2] value.
redPromise := [aClosure value: 'Red' value: 3] promise.
greenPromise := [aClosure value: 'Green' value: 3] promise.
bluePromise := [aClosure value: 'Blue' value: 3] promise.
redPromise value.
greenPromise value.
bluePromise value.
[aClosure value: 'White' value: 2] value
```

```
Black: 11:52:25
Black: 11:52:26
Red: 11:52:27
Green: 11:52:27
Blue: 11:52:27
Red: 11:52:28
Green: 11:52:28
Blue: 11:52:28
Red: 11:52:29
Green: 11:52:29
Blue: 11:52:29
White: 11:52:30
White: 11:52:31
```


約束(3)




```
| aBlock aClosure redPromise greenPromise bluePromise |
Transcript clear.
aBlock :=
            [:aString |
            Transcript
                  nextPutAll: aString;
                  nextPutAll: ': ';
                  nextPutAll: Time now printString;
                  cr;
                  flush].
aClosure :=
            [:aString :howMany |
            howMany timesRepeat:
                  [aBlock value: aString.
                  1000 milliseconds wait]].
redPromise := greenPromise := bluePromise := nil.
[aClosure value: 'Black' value: 2] value.
redPromise :=
            [redPromise value: '赤'.
            aClosure value: 'Red' value: 3] promise.
greenPromise :=
            [aClosure value: 'Green' value: 1.
            greenPromise value: '緑'.
            aClosure value: 'Green' value: 2] promise.
bluePromise :=
            [aClosure value: 'Blue' value: 2.
            bluePromise value: '青'.
            aClosure value: 'Blue' value: 1] promise.
aBlock value: redPromise value.
aBlock value: greenPromise value.
aBlock value: bluePromise value.
[aClosure value: 'White' value: 2] value
```

```
Black: 11:52:50
Black: 11:52:51
Red: 11:52:52
Green: 11:52:52
Blue: 11:52:52
赤: 11:52:52
Red: 11:52:53
Green: 11:52:53
Blue: 11:52:53
緑: 11:52:53
Red: 11:52:54
Green: 11:52:54
Blue: 11:52:54
青: 11:52:54
White: 11:52:54
White: 11:52:55
```