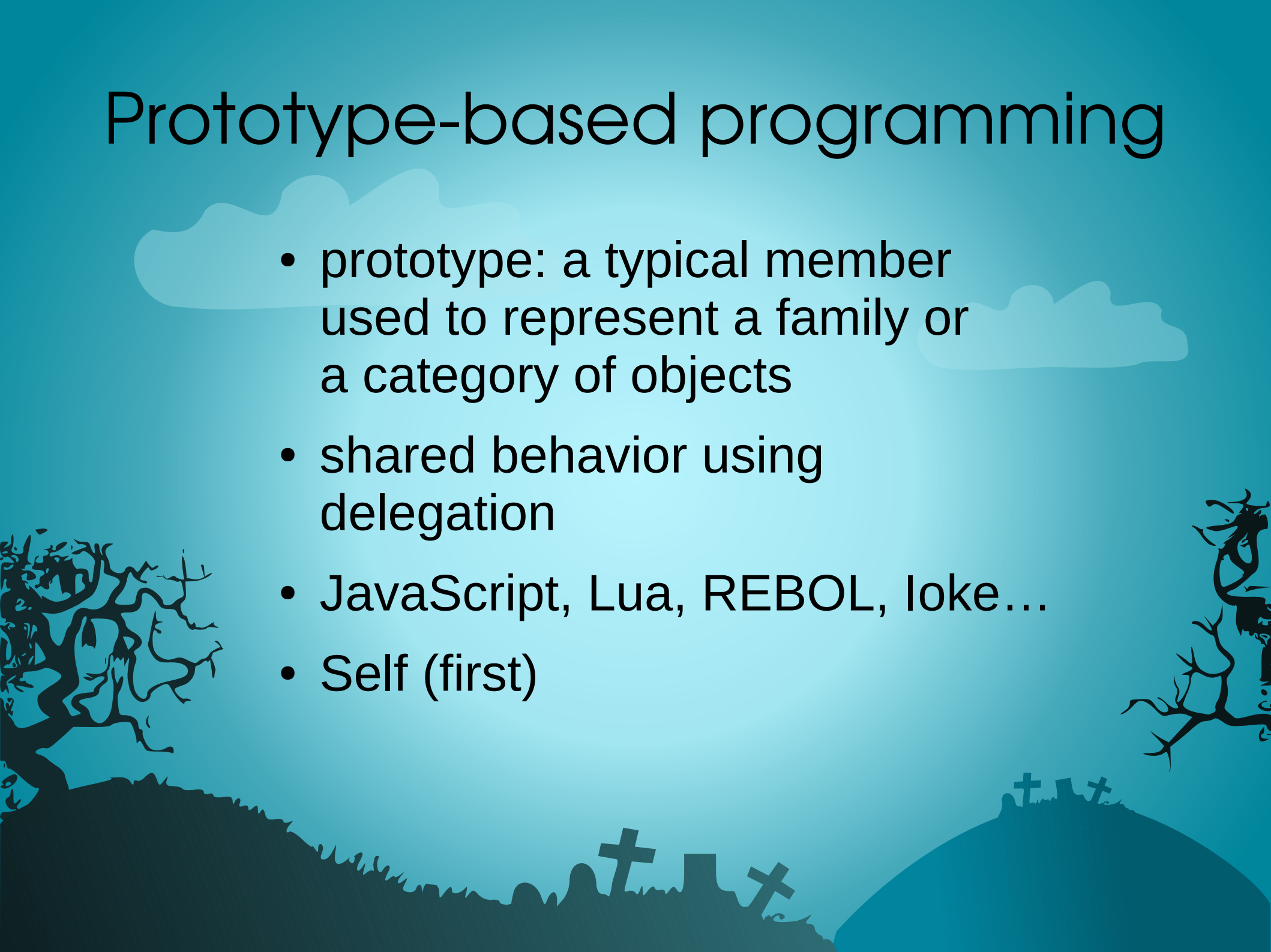
Prototypes in Pharo

Pavel Krivanek, Inria Nord Europe, RMoD

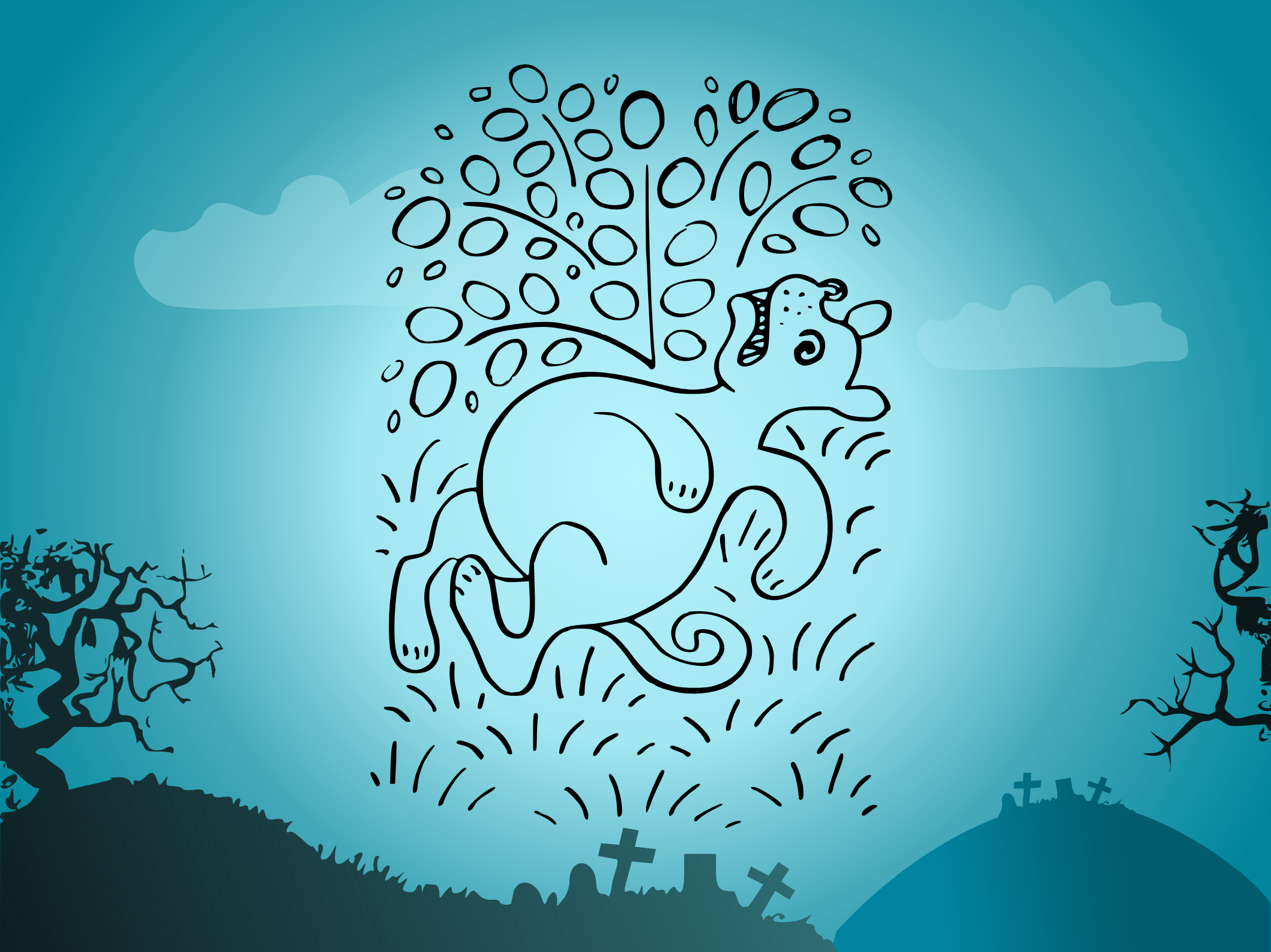
Prototype-based programming

- prototype: a typical member used to represent a family or a category of objects
- shared behavior using delegation
- JavaScript, Lua, REBOL, Ioke…
- Self (first)



- 1986 Xerox PARC (David Ungar, Randall Smith)
- Standford University, Sun Microsystems (1990)
- slow progress since 1995, latest release May 2017

http://www.selflanguage.org/




Self
How can we improve Smalltalk?
Let's make it even simpler!

- prototype-based Smalltalk
- no classes, no metaclasses
- more focused on objects
- more Smalltalkish than Smalltalk
- only one keyword
- no symbols
- uniform messages and variables

- multiple dynamic inheritance
- namespaces
- modules
- full closures
- total object encapsulation
- was faster (2x)

Objects: composition of slots

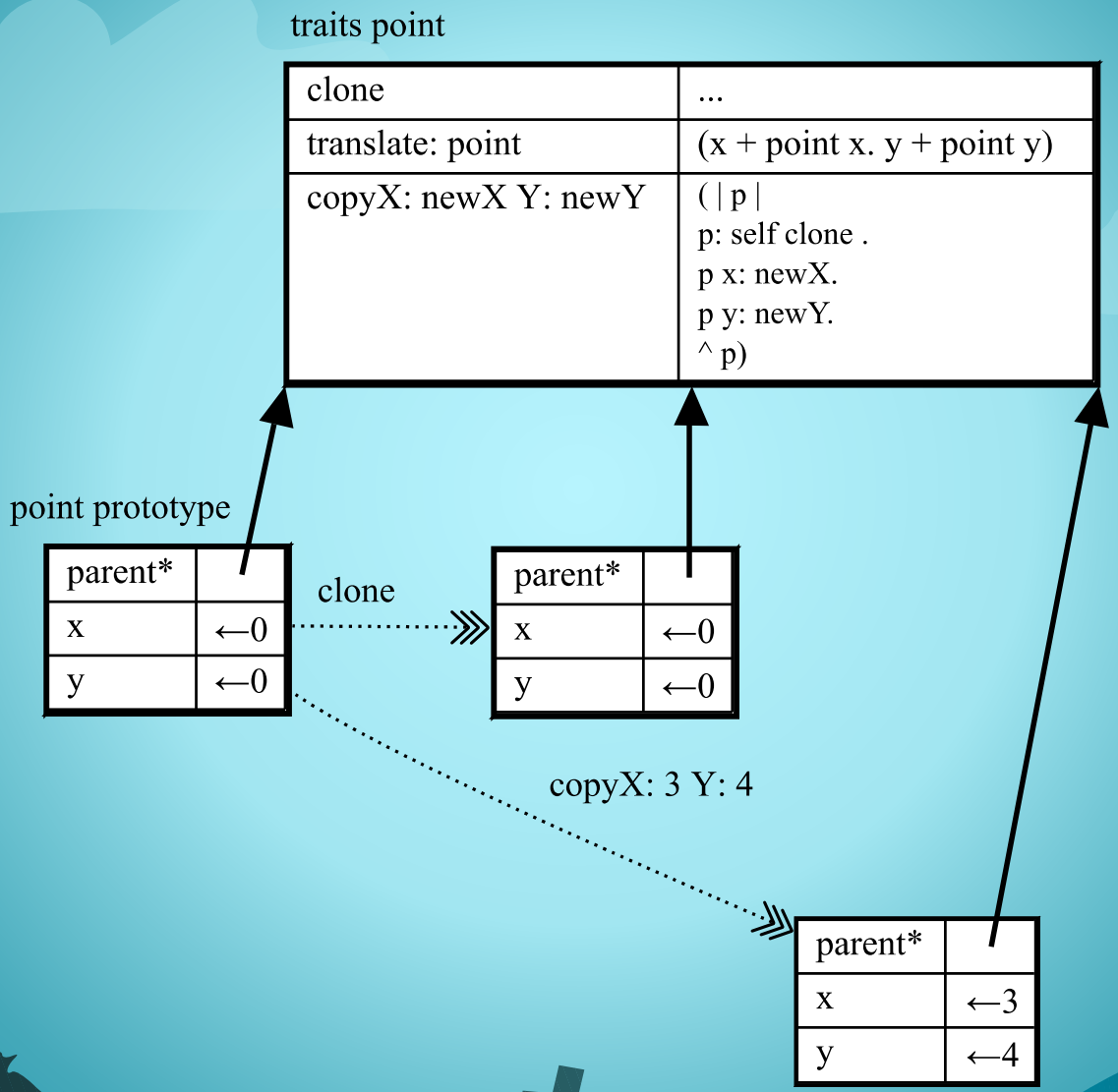
- named references
  - data slots (read-only / read-write)
  - methods
  - argument slots
  - parent slots (read-only /read-write)

  - auxiliary (unnamed slots, unnamed parent slots…)

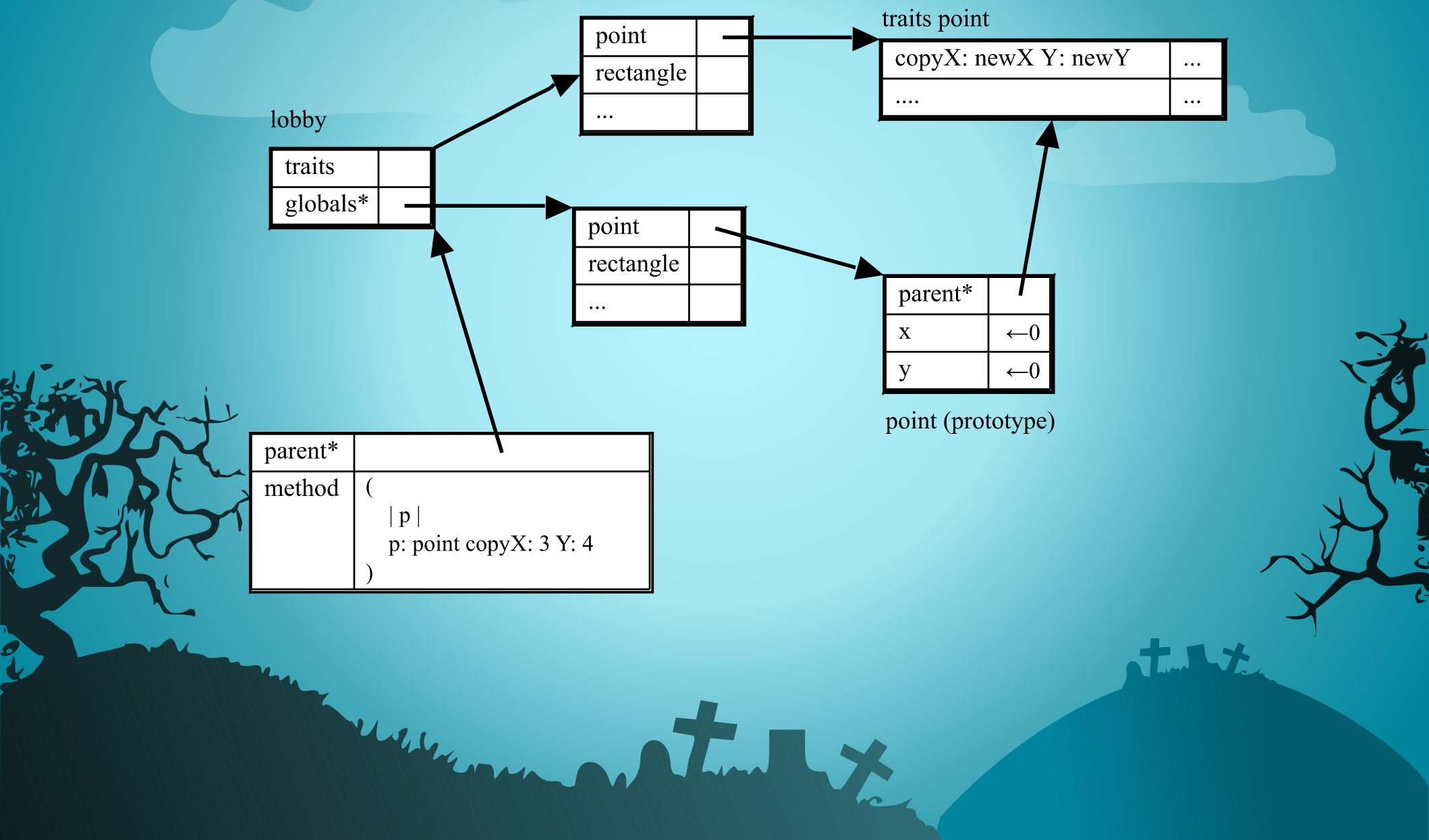
Delegation

DEMO

Prototypes and traits



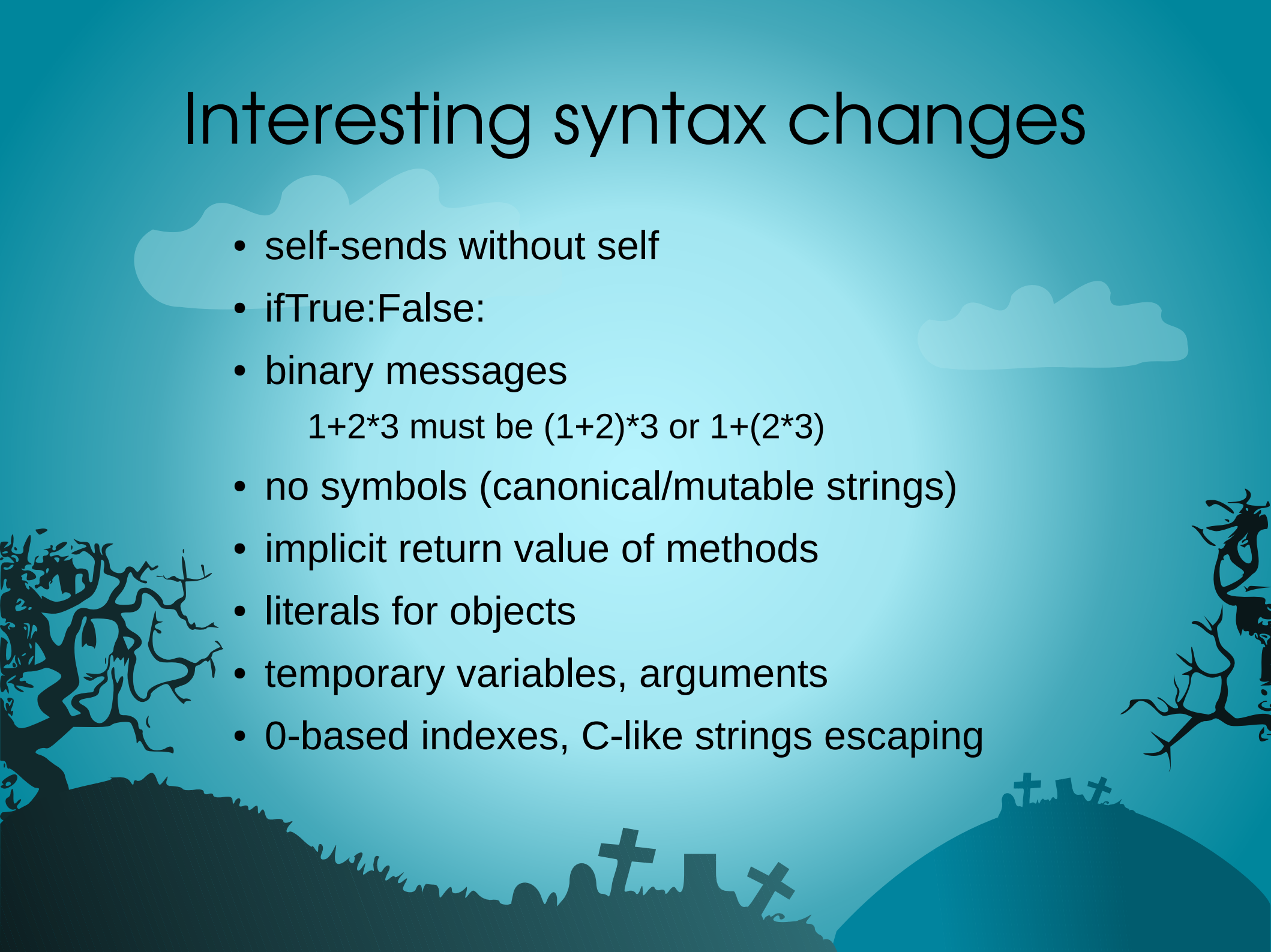
Prototypes and traits



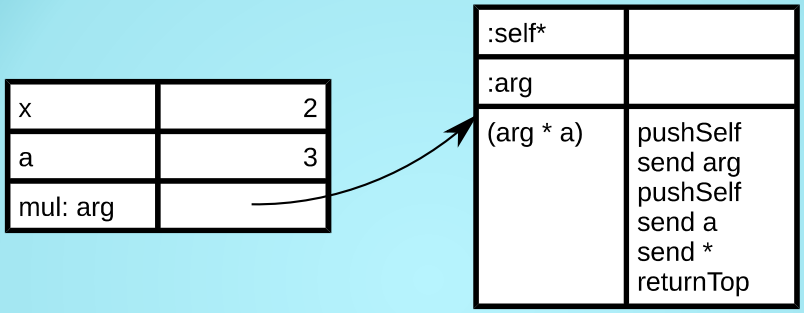
Interesting syntax changes

- self-sends without self
- ifTrue:False:
- binary messages

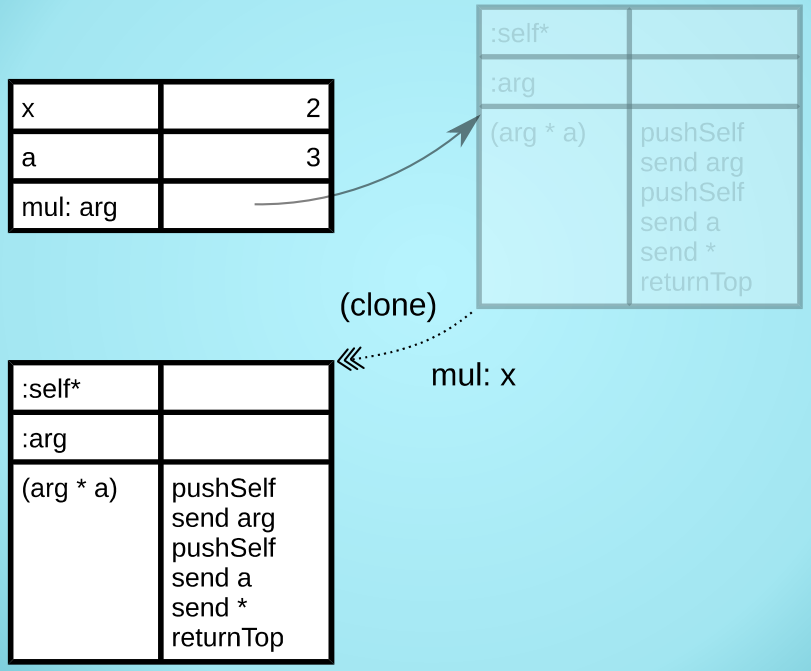
  1+2*3 must be (1+2)*3 or 1+(2*3)
- no symbols (canonical/mutable strings)
- implicit return value of methods
- literals for objects
- temporary variables, arguments
- 0-based indexes, C-like strings escaping



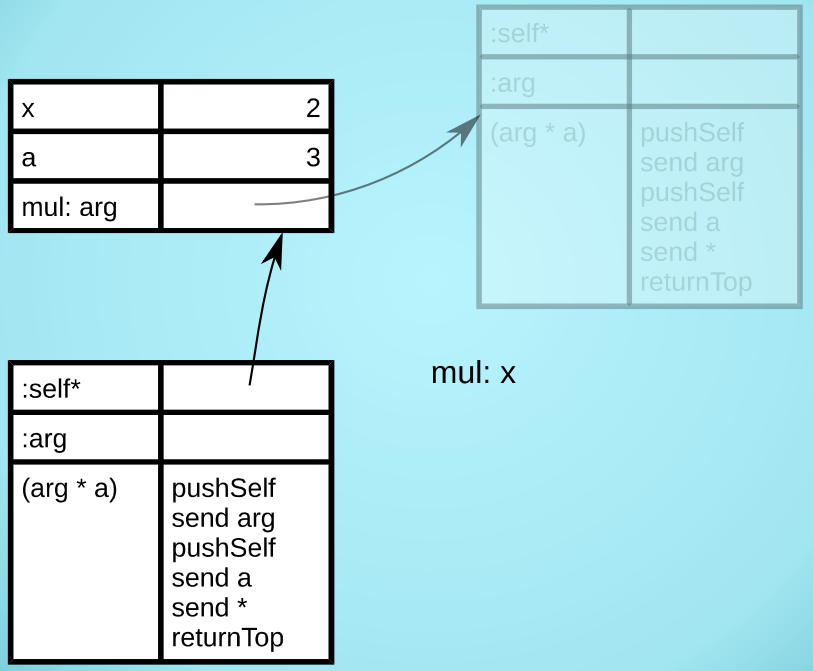
Method activation



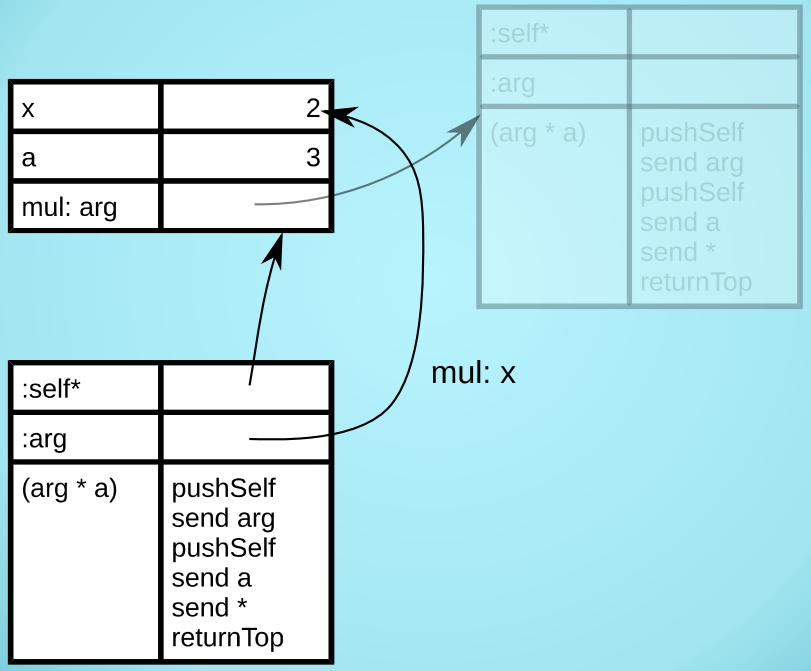
Method activation



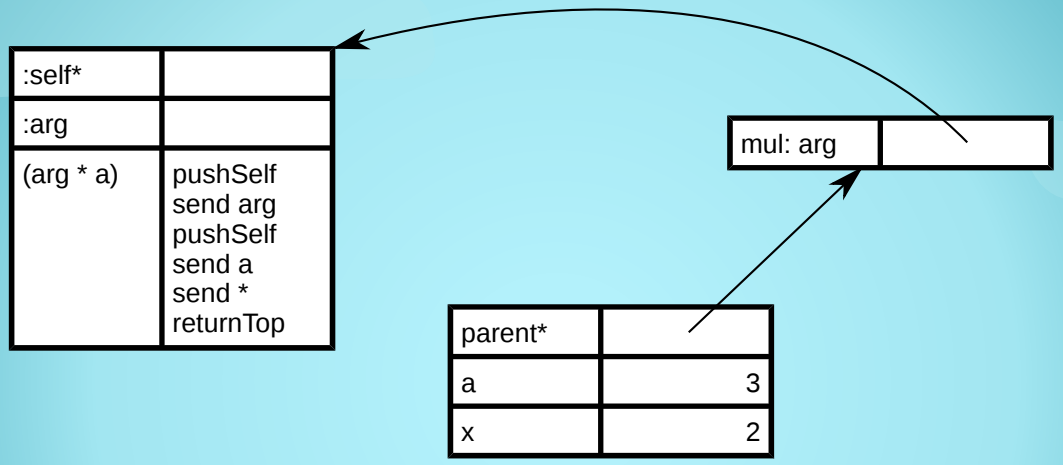
Method activation



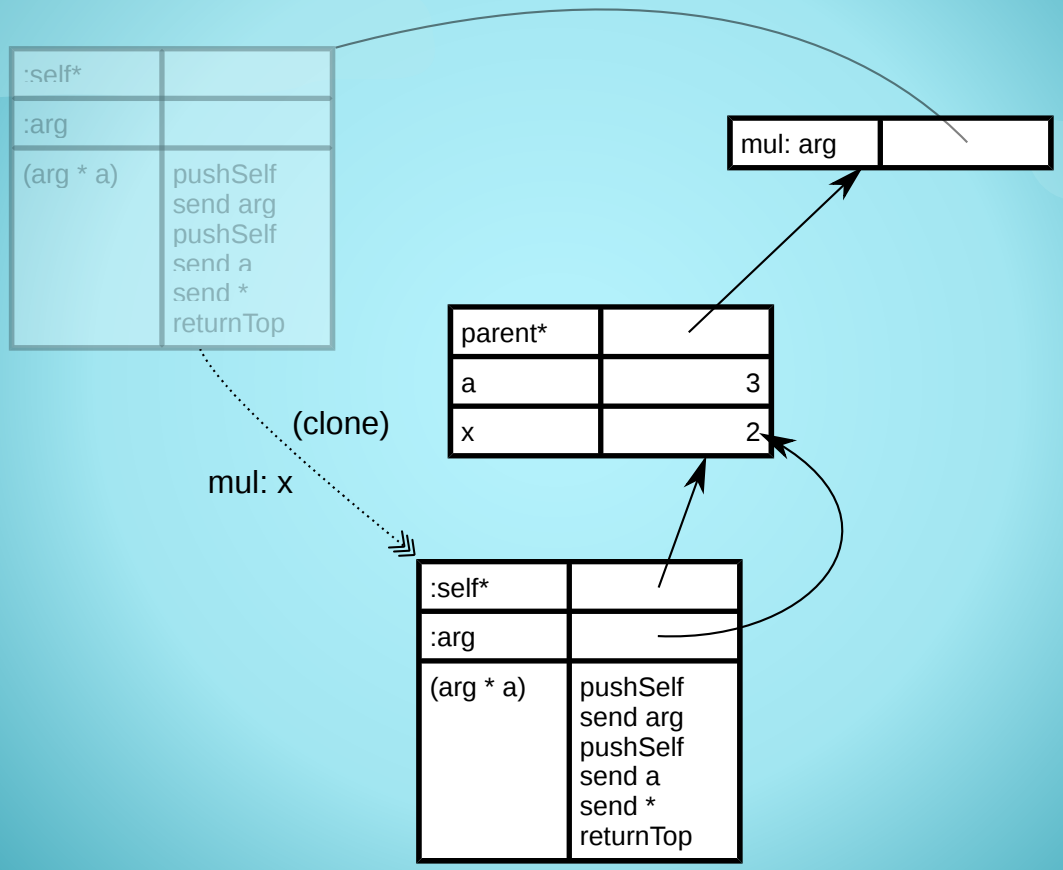
Method activation



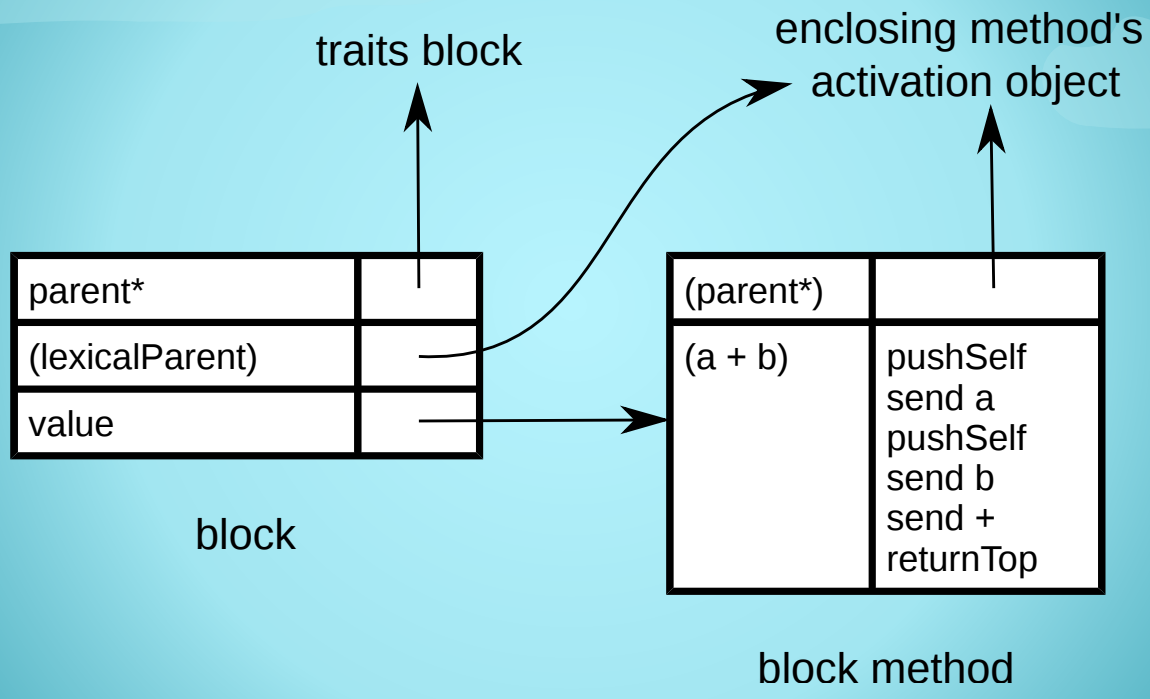
Delegation


:self*
:arg
(arg * a)
pushSelf
send arg
pushSelf
send a
send *
returnTop
mul: arg
parent*
a
3
x
2

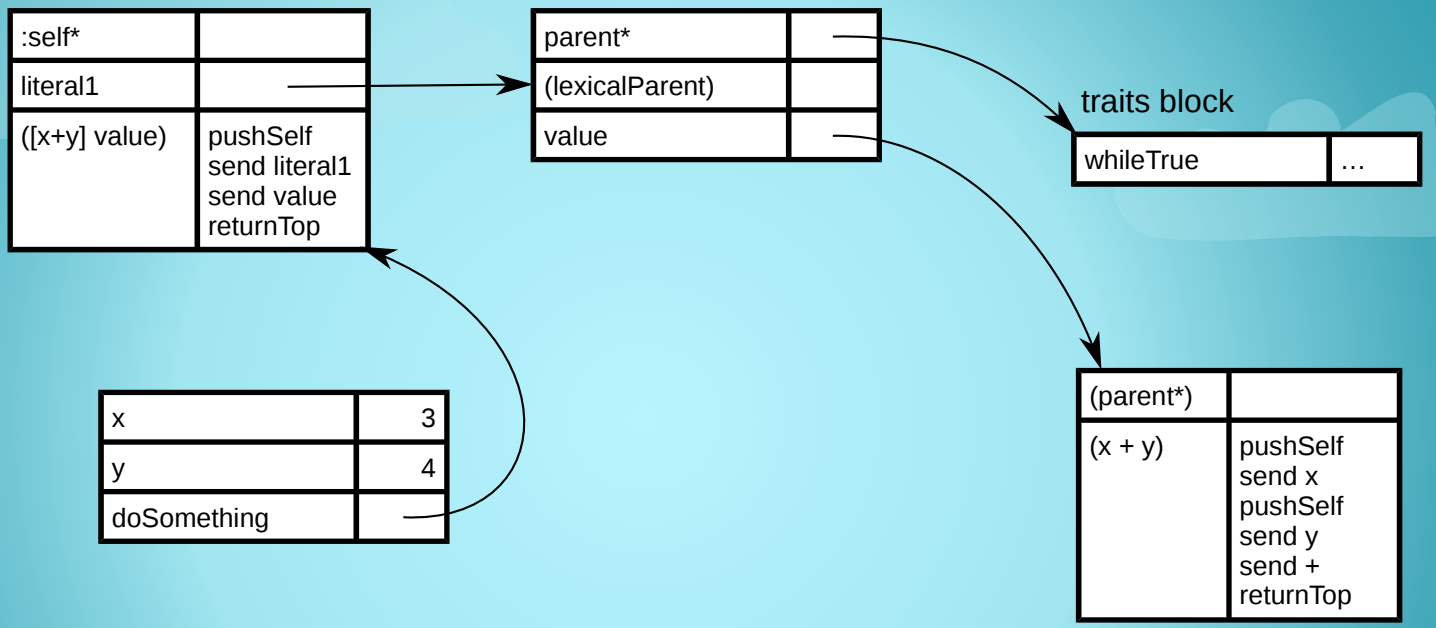
Delegation


:self*
:arg
(arg * a)
pushSelf
send arg
pushSelf
send a
send *
returnTop
mul: arg
parent*
a
3
x
2
(clone)
mul: x
:self*
:arg
(arg * a)
pushSelf
send arg
pushSelf
send a
send *
returnTop

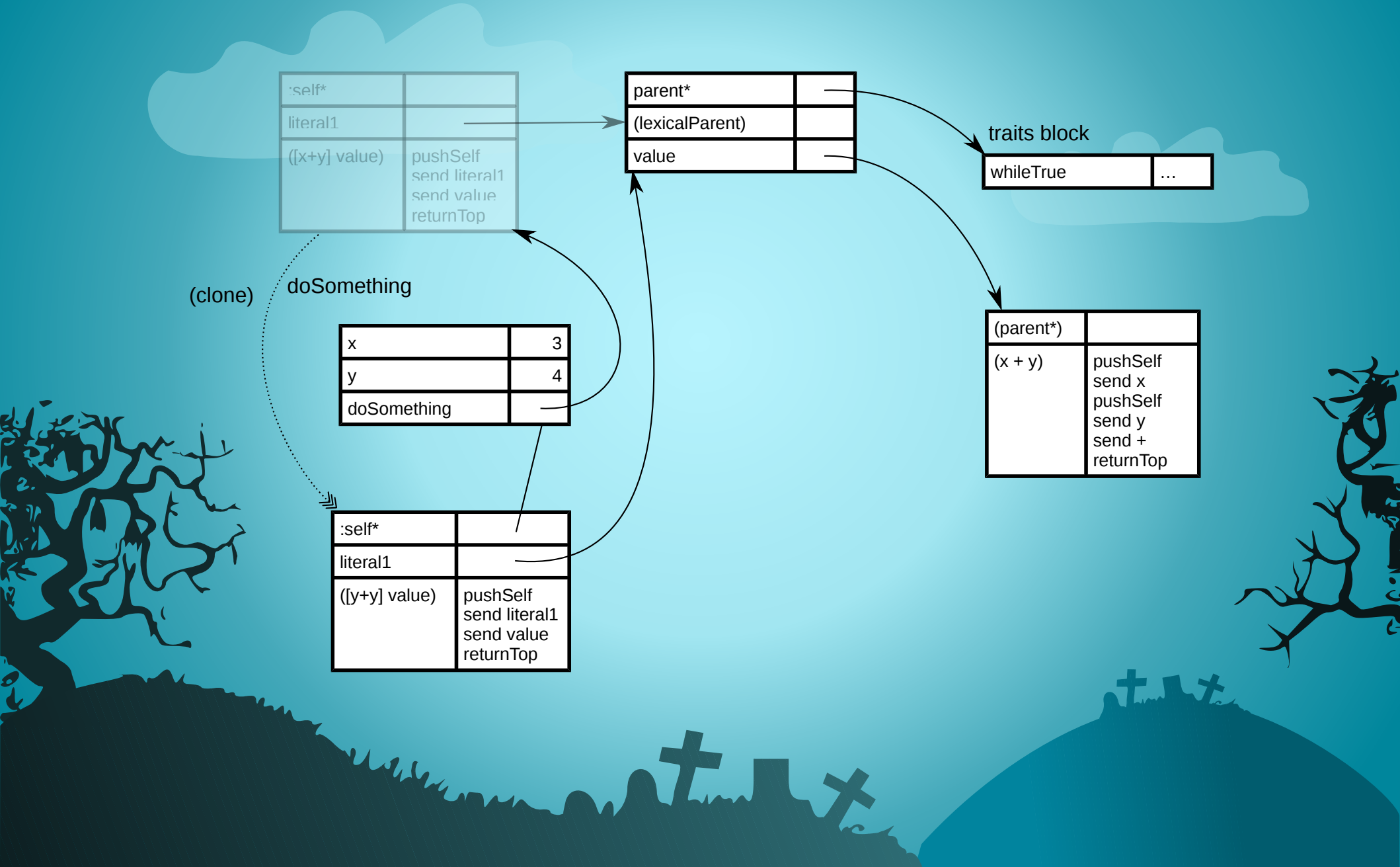
Blocks


traits block
enclosing method's
activation object
parent*
(lexicalParent)
value
block
(parent*)
(a + b)
pushSelf
send a
pushSelf
send b
send +
returnTop
block method

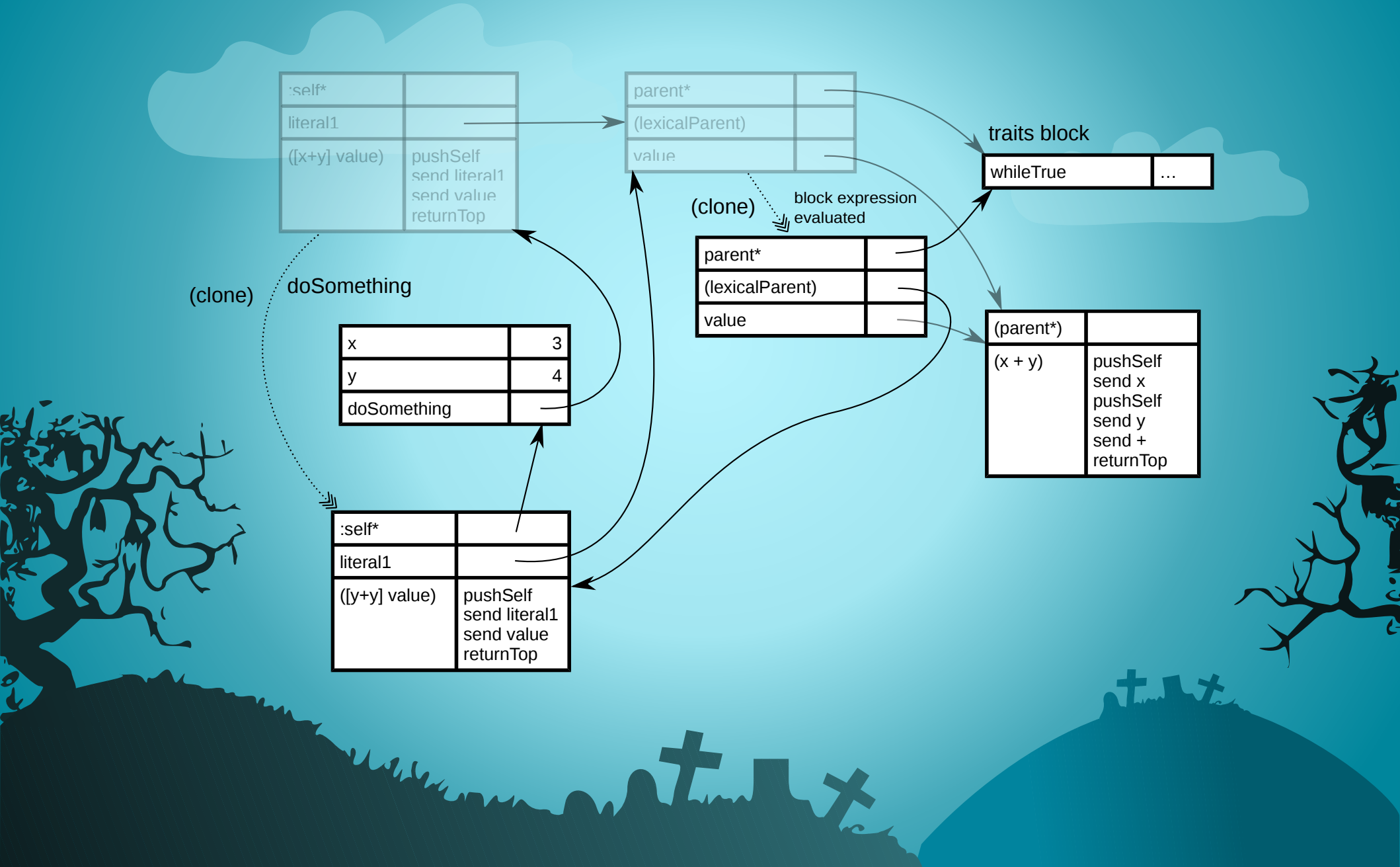
Blocks


:self*
literal1
([x+y] value)
pushSelf
send literal1
send value
returnTop
parent*
(lexicalParent)
value
traits block
whileTrue
…
x
3
y
4
doSomething
(parent*)
(x + y)
pushSelf
send x
pushSelf
send y
send +
returnTop

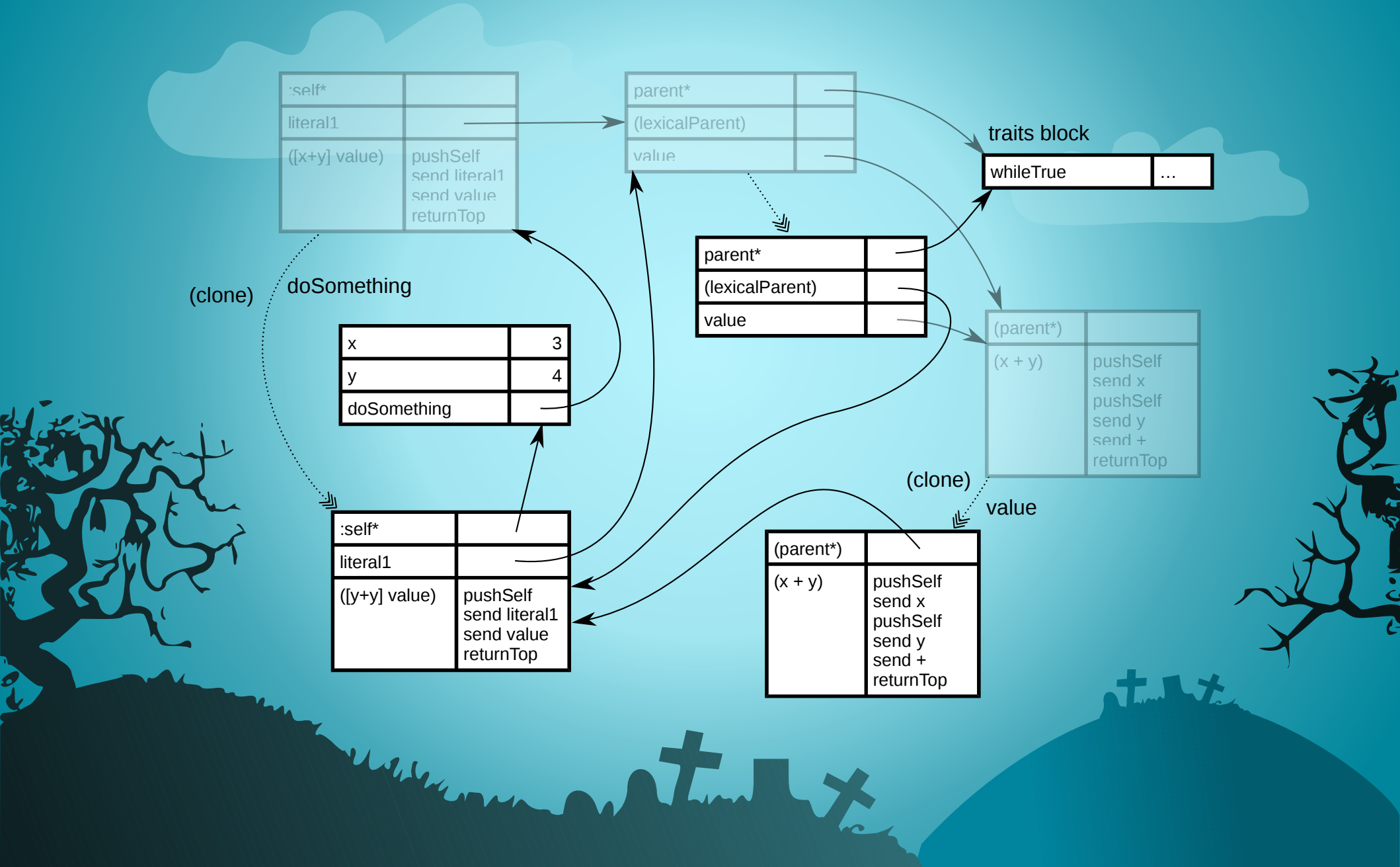
Blocks


:self*
literal1
([x+y] value)
pushSelf
send literal1
send value
returnTop
parent*
(lexicalParent)
value
traits block
whileTrue
...
(clone)
doSomething
x
3
y
4
doSomething
(parent*)
(x + y)
pushSelf
send x
pushSelf
send y
send +
returnTop
:self*
literal1
([y+y] value)
pushSelf
send literal1
send value
returnTop

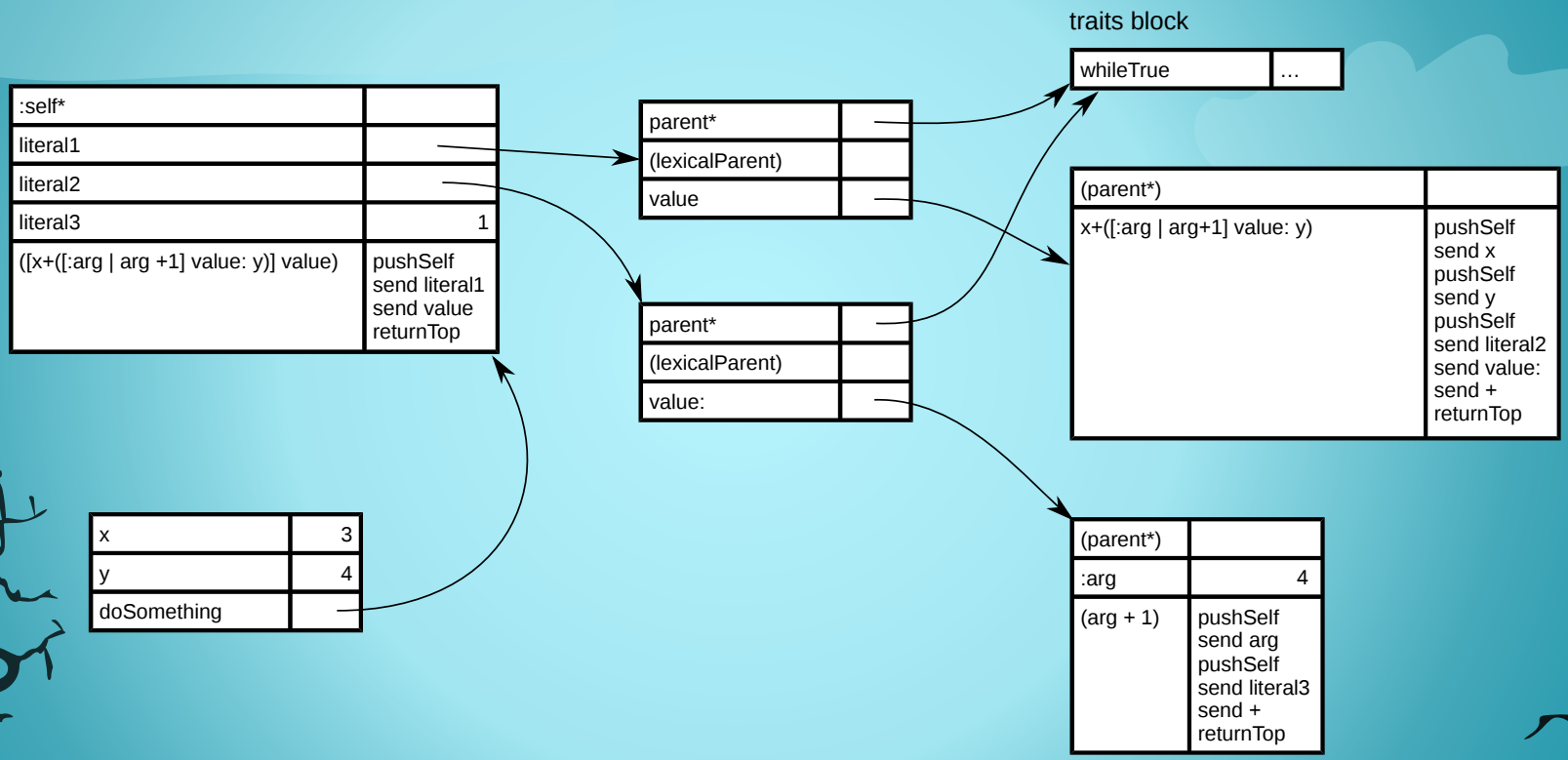
Blocks


:self*
literal1
([x+y] value)
pushSelf
send literal1
send value
returnTop
parent*
(lexicalParent)
value
traits block
whileTrue
...
(clone)
block expression
evaluated
parent*
(lexicalParent)
value
(clone)
doSomething
x
3
y
4
doSomething
(parent*)
(x + y)
pushSelf
send x
pushSelf
send y
send +
returnTop
:self*
literal1
([y+y] value)
pushSelf
send literal1
send value
returnTop

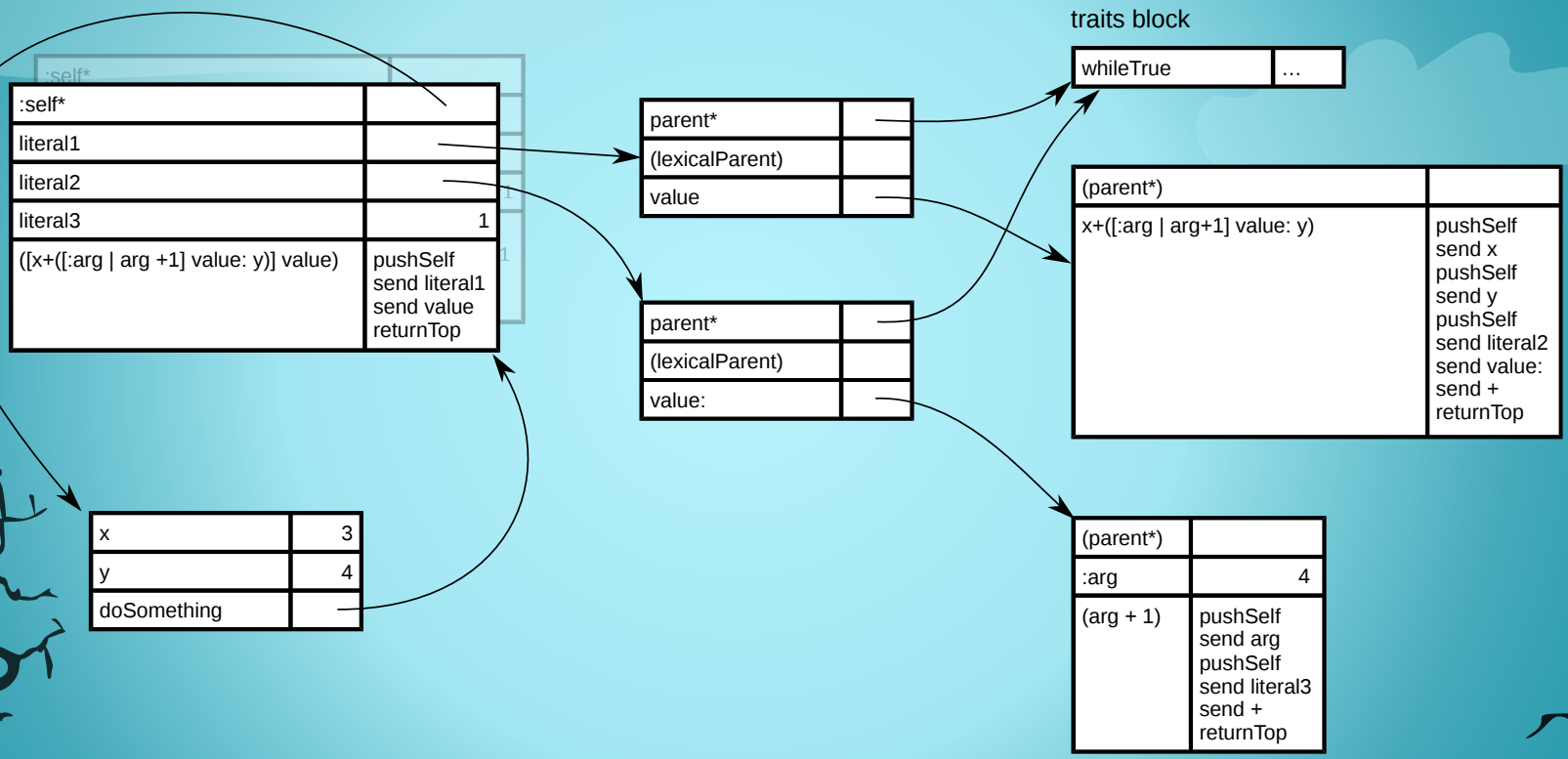
Blocks


:self*
literal1
([x+y] value)
pushSelf
send literal1
send value
returnTop
parent*
(lexicalParent)
value
traits block
whileTrue
...
parent*
(lexicalParent)
value
(clone)
doSomething
x
3
y
4
doSomething
(parent*)
(x + y)
pushSelf
send x
pushSelf
send y
send +
returnTop
(clone)
value
:self*
literal1
([y+y] value)
pushSelf
send literal1
send value
returnTop
(parent*)
(x + y)
pushSelf
send x
pushSelf
send y
send +
returnTop

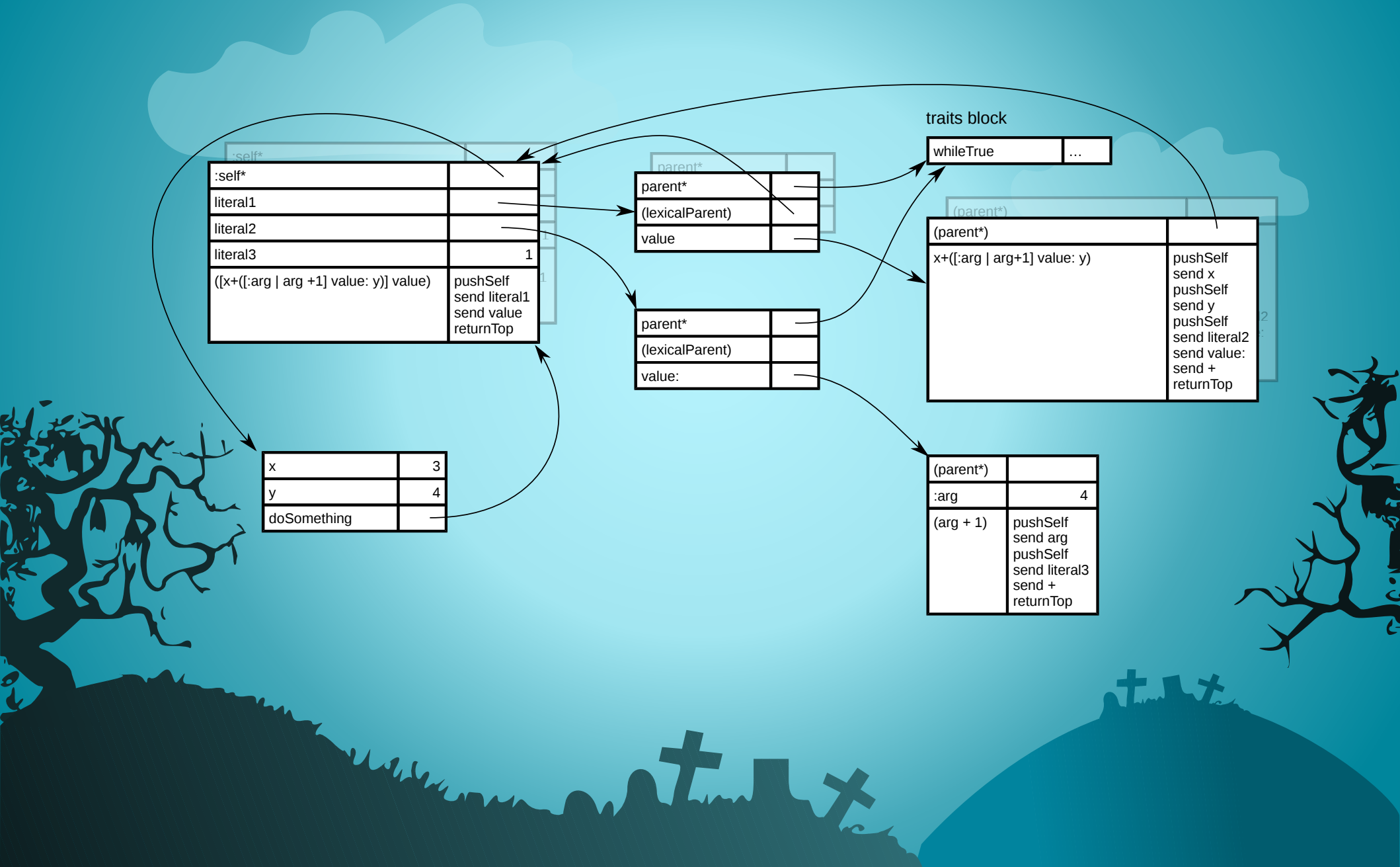
Embedded blocks



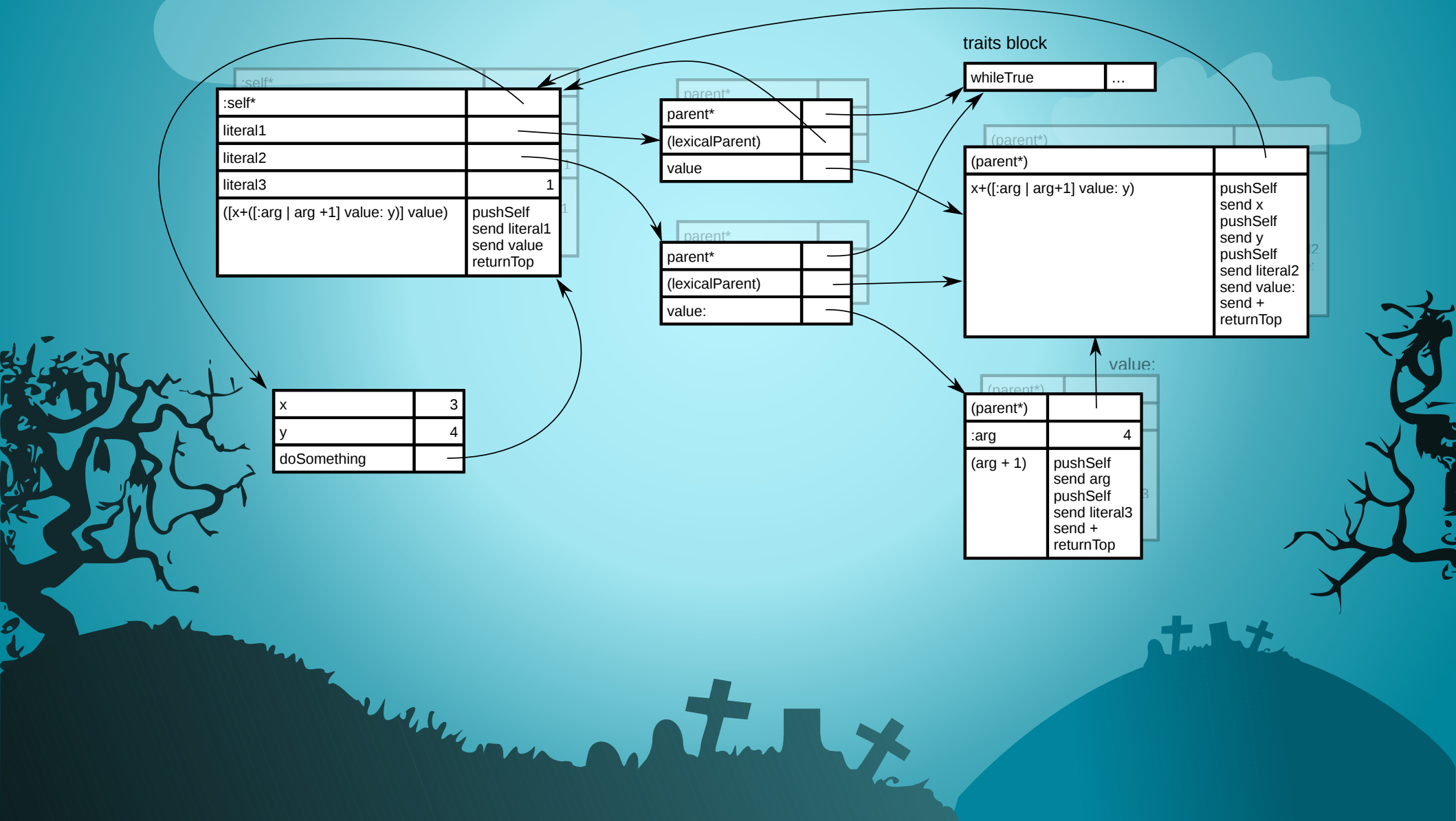
Embedded blocks



Embedded blocks

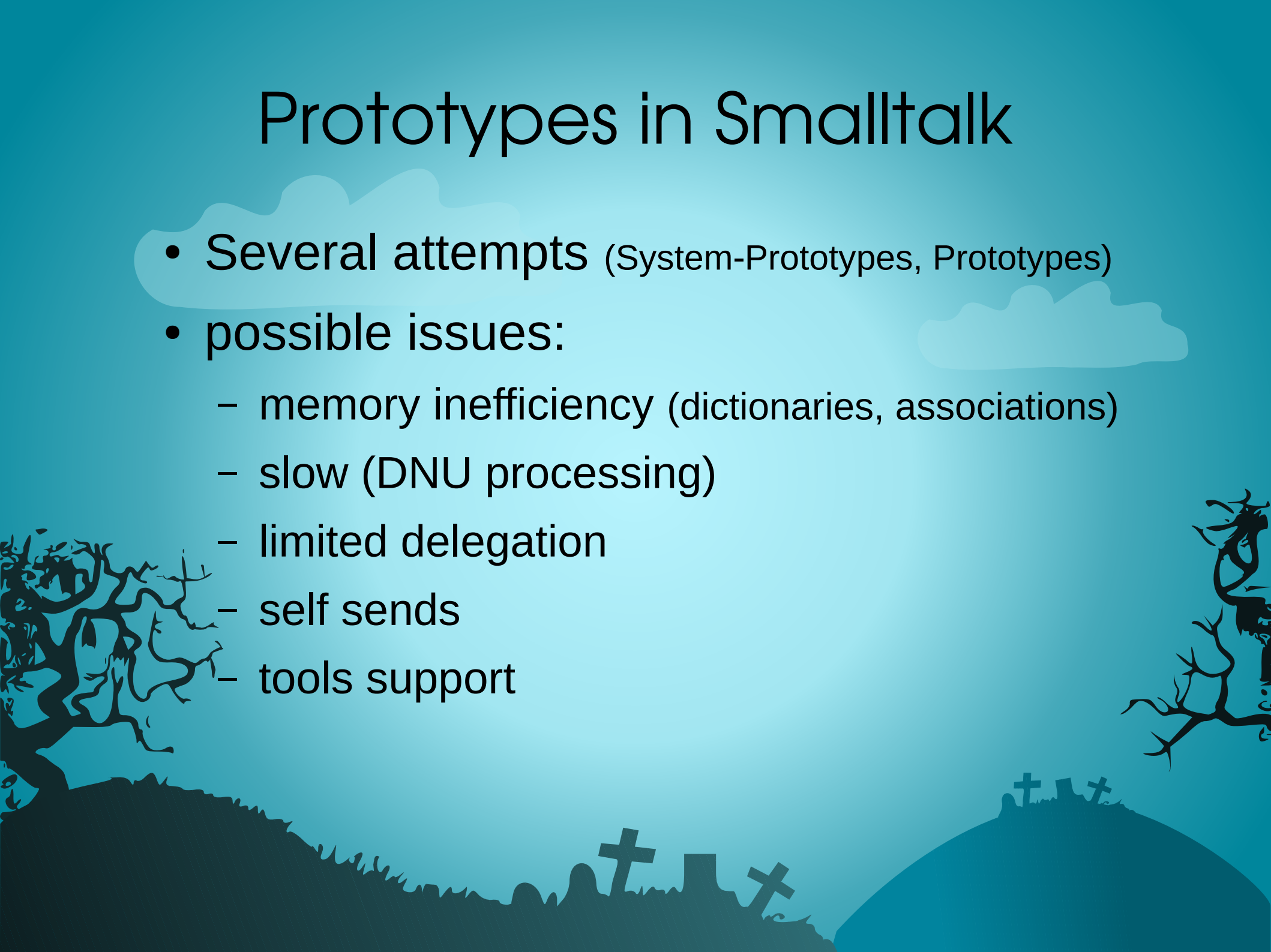


Embedded blocks



Prototypes in Smalltalk

- Several attempts (System-Prototypes, Prototypes)
- possible issues:
  - memory inefficiency (dictionaries, associations)
  - slow (DNU processing)
  - limited delegation
  - self sends
  - tools support



Russel Allen's dark magic (2005)

- create instance of Behavior (parent)
- set its superclass to Behavior
- create prototype as instance of the parent
- clone instance variables of the parent into the prototype (superclass, method dict, format)
- parent gets identity of the prototype
- extend methods dictionary from Prototype (addSlot:…)

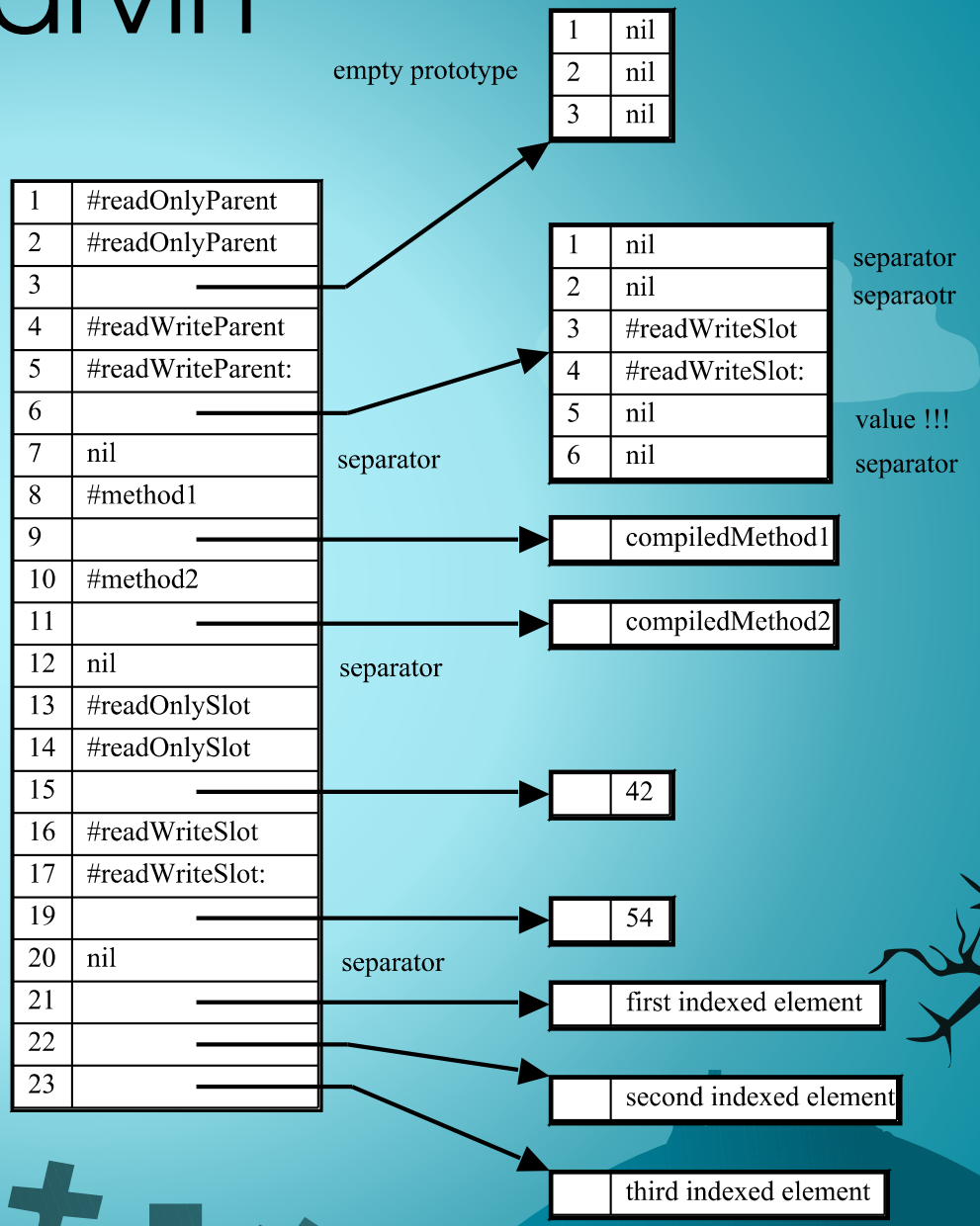
Russel Allen's dark magic (2005)

- create instance of Behavior (parent)
- set its superclass to Behavior
- create prototype as instance of the parent
- clone instance variables of the parent into the prototype (superclass, method dict, format)
- parent gets identity of the prototype
- extend methods dictionary from Prototype (addSlot:…)

→ the object is its own instance

Marvin

- objects as arrays
- no associations
- nil as separators
  - parent slots
  - methods
  - data slots
  - indexed content



Marvin

- Delegation implemented in VM
  - Depth First Search (unlike Self)
  - easy in that time
- Follow the Smalltalk infrastructure
  - Smalltalk blocks
  - Symbols

Marvin

- own syntax (SmaCC)
- mix of Self and Smalltalk

Self:

```
| p1 |
p1: (|
      parent* = (|
            parent* = (|
                  a = ( ^a ).
                  b = 'symbol' |).
            a = ( resend.a ) |).
      a = (|
            a = 3.
            b = 4 |).
      method = ( resend.a a ) |).
p1 method
```

Marvin:

```
| p1 |
p1: (|
      parent* = (|
            parent* = (|
                  a = { ^a }.
                  b = #symbol |).
            a = { ^resend a } |).
      a = (|
            a = 3.
            b = 4 |).
      method = { ^resend a a } |).
p1 method
```

current Marvin

- Delegation based on DNU
- Easier combination with regular objects
- Indexed content at the beginning
  - Indexed access to inst. variables in CM
- Smalltalk objects injection

https://github.com/pavel-krivanek/Marvin

Objects injection

Date today (instance variables)

```
objectToExtend := Date today.
```

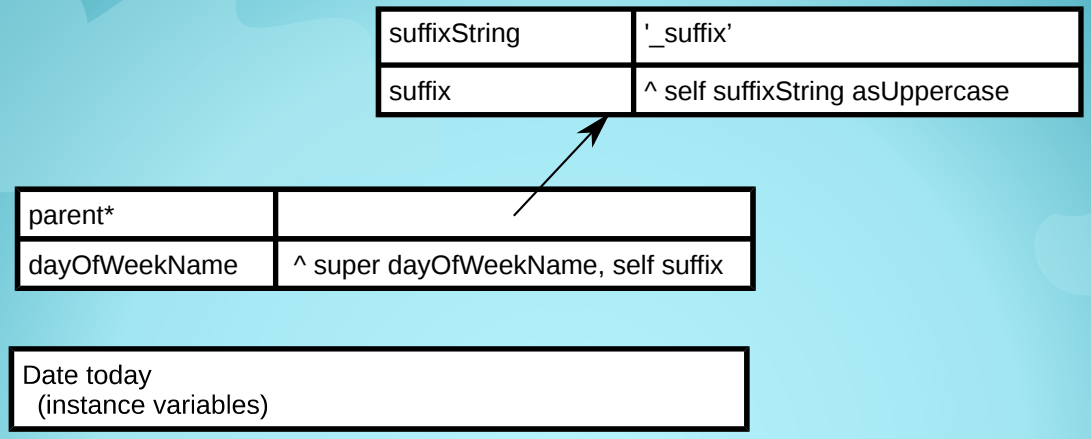
Objects injection

suffixString	'_suffix’
suffix	^ self suffixString asUppercase

Date today (instance variables)

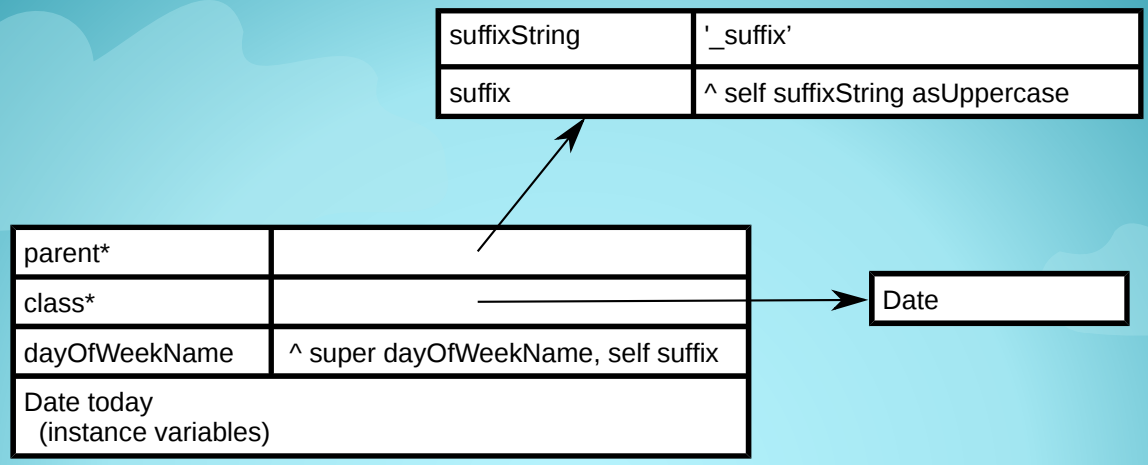
```
objectToExtend := Date today.
parent := MarvinPrototype new.
parent _AddMethod: 'suffix ^ self suffixString asUppercase'.
parent _AddReadSlot: #suffixString value: '_suffix'.
```

Objects injection




```
objectToExtend := Date today.
parent := MarvinPrototype new.
parent _AddMethod: 'suffix ^ self suffixString asUppercase'.
parent _AddReadSlot: #suffixString value: '_suffix'.
object := MarvinPrototype new.
object _AddParentSlot: #parent value: parent.
object _AddMethod: 'dayOfWeekName
      ^ super dayOfWeekName, self suffix'.
```

Objects injection




```
objectToExtend := Date today.
parent := MarvinPrototype new.
parent _AddMethod: 'suffix ^ self suffixString asUppercase'.
parent _AddReadSlot: #suffixString value: '_suffix'.
object := MarvinPrototype new.
object _AddParentSlot: #parent value: parent.
object _AddMethod: 'dayOfWeekName
      ^ super dayOfWeekName, self suffix'.
object _Inject: objectToExtend.

      object dayOfWeekName >>> 'Thursday_SUFFIX'
```

Why Self

is not more widely known and used?

What about Smalltalkers?

Self disadvantages

- lost commercial support (Sun)
- complicated VM
- platforms (no Windows VM, late Linux, 32-bits only)
- Morphic, UI, tools (outliners)
- Documentation
- Community

788x50844

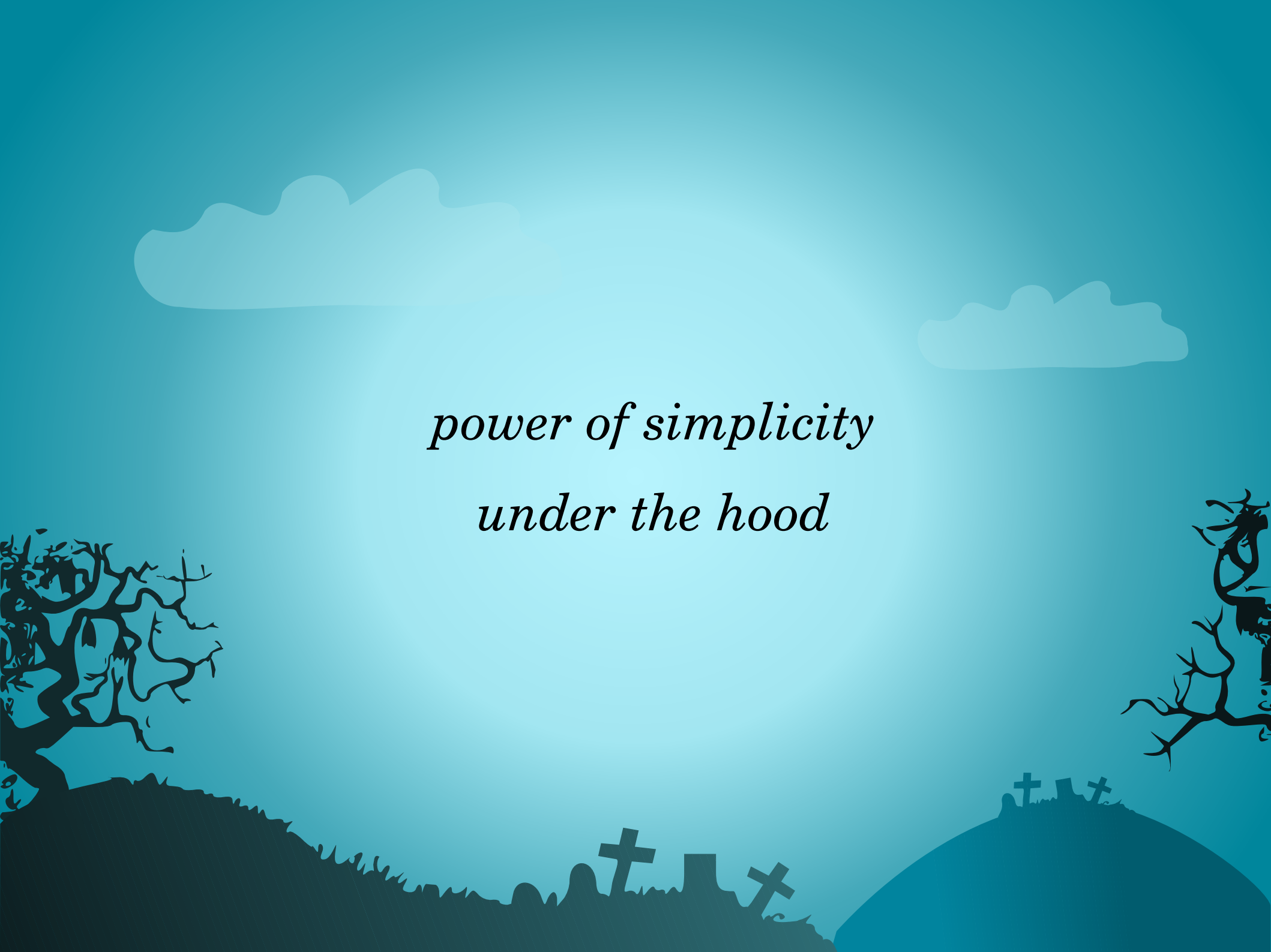
Self disadvantages

Does it remove from Smalltalk things that make people productive?

Easy to loose control?

power of simplicity

under the hood




In
loving
Memory