A Declarative Model of Smalltalk Programs

Allen Wirfs-Brock

ParcPlace-Digitalk

allen@parcplace.com

Collaborators

- Co-authors
  - Brian Wilkerson
  - Juanita Ewing
  - Harold Williams
- Other Key Contributors
  - Steve Messick
  - John Wiegand
  - Dale Henrichs
  - Carl McConnell
  - Pat Caudill
  - Tim O'Connor

What is Smalltalk?

- The system software of the "Interim Dynabook"
- "The purpose of the Smalltalk project is to provide computer support for the creative spirit in everyone"
  - ⑨Ingalls, 1981
- "COBOL for the 90's"?

Evolution of Smalltalk

- Smalltalk was originally:
  - A programming language
  - A development environment
  - A GUI environment
  - An operating system
- all rolled into one...

Smalltalk has changed and evolved over 25 years

- lost control of hardware
- lost control of screen
- adapted to standard operating systems and GUI environment
- learned to inter-operate with languages, databases, and communications protocols

Smalltalk Today

- A comprehensive object-oriented programming language used to build complex, mission critical, enterprise and technical applications.
- The benchmark against which all other object-orient programming languages and development environments are compared.

A Declarative Model of Smalltalk Programs

How Smalltalk continues to evolved to "be more normal while remaining special"

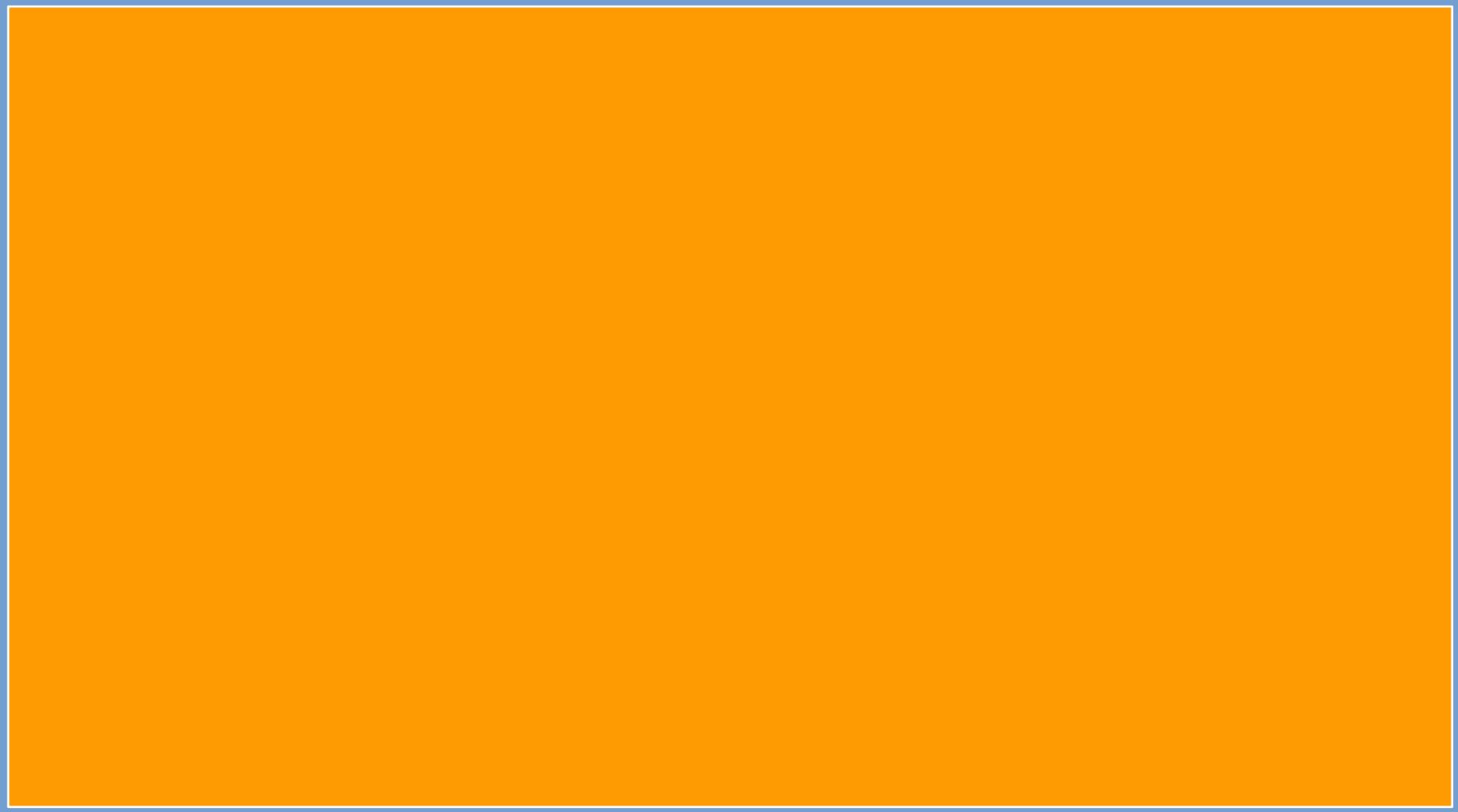
Smalltalk: Strengths

- Pure Object-oriented language
- Rich class library
- Incremental development
- High programmer productivity
- It's a pleasure to work with

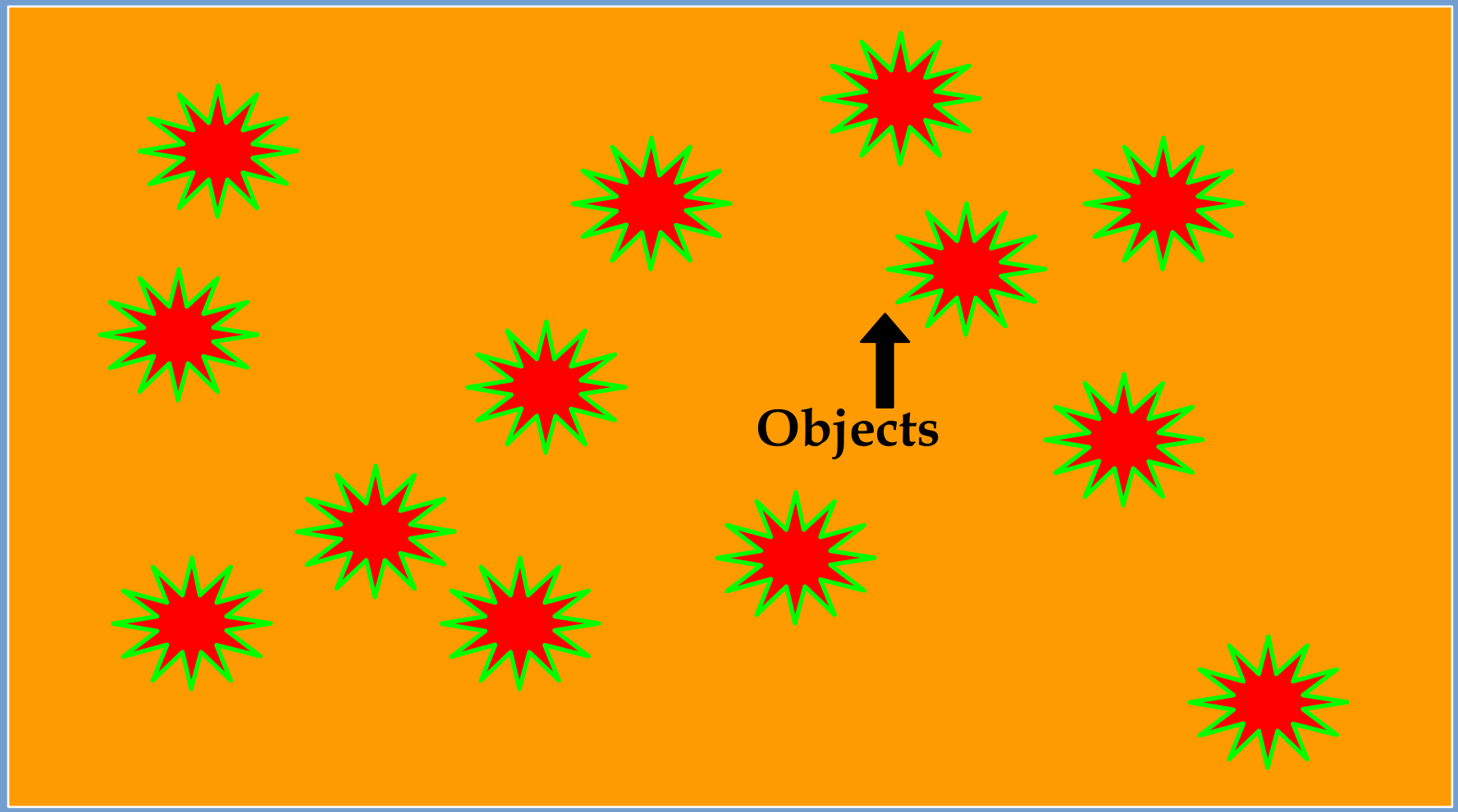
... and Weaknesses

- Size
- Performance (?)
- Application Delivery
- Maintenance

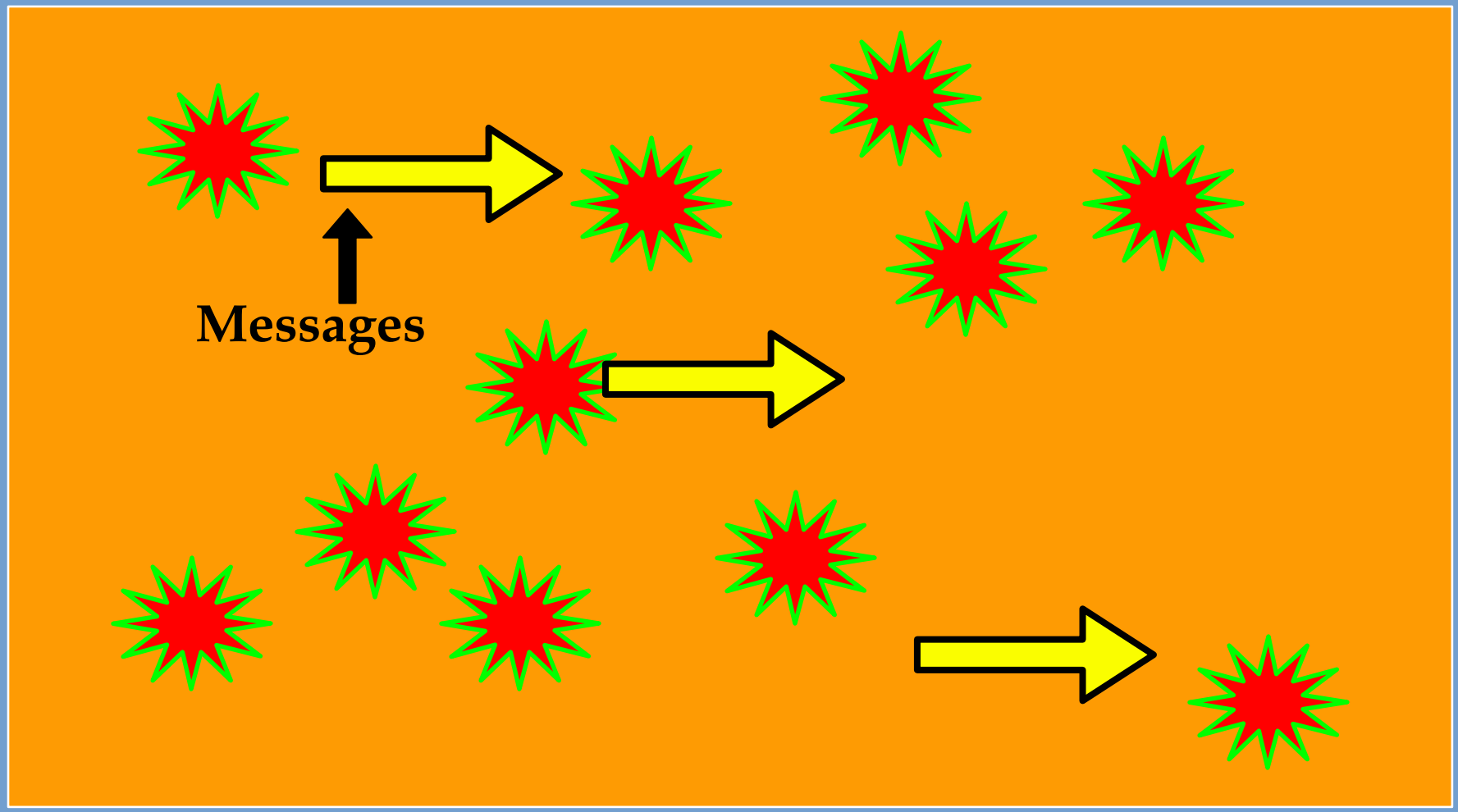
Review:The Smalltalk Image



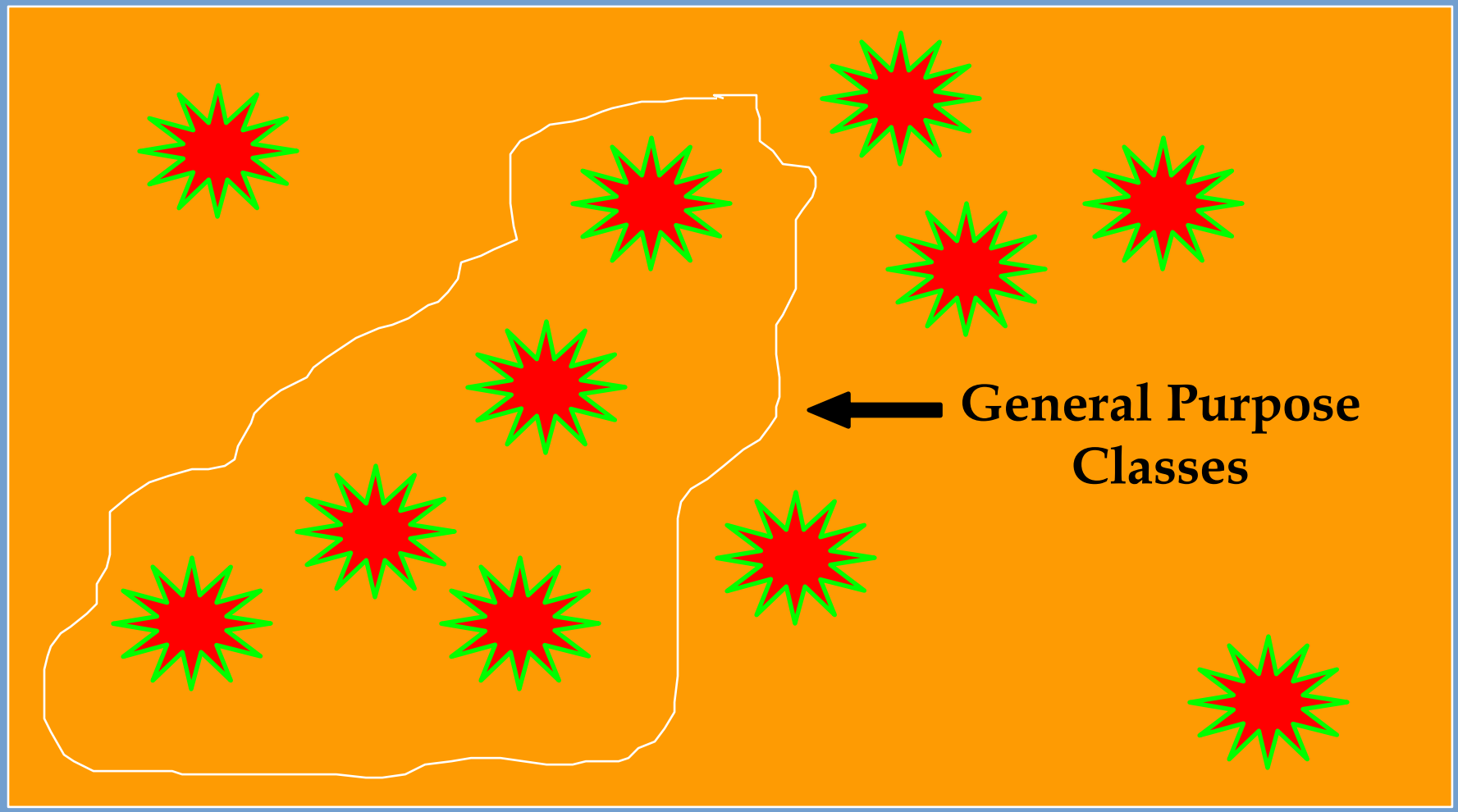
Review:The Smalltalk Image



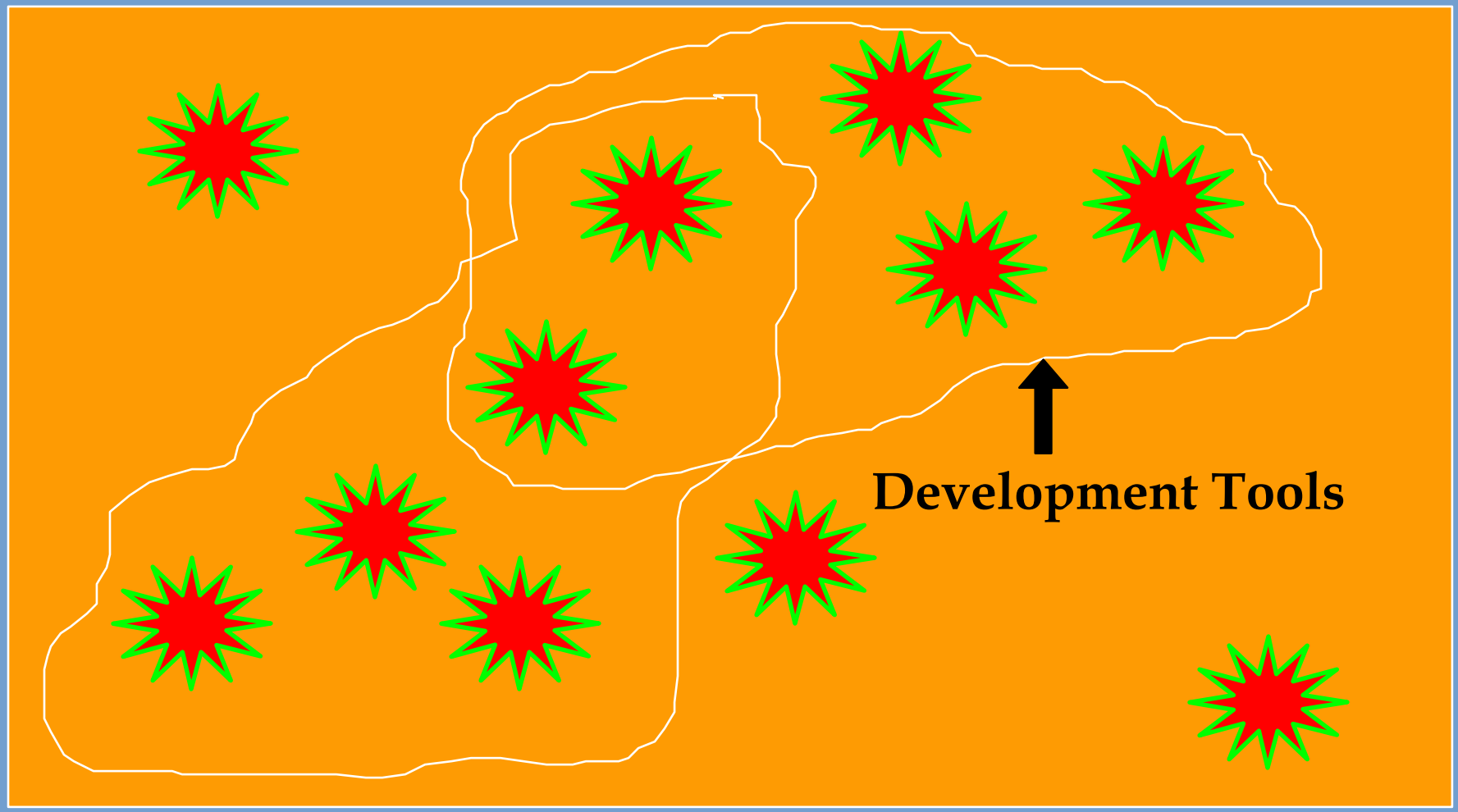
Review:The Smalltalk Image



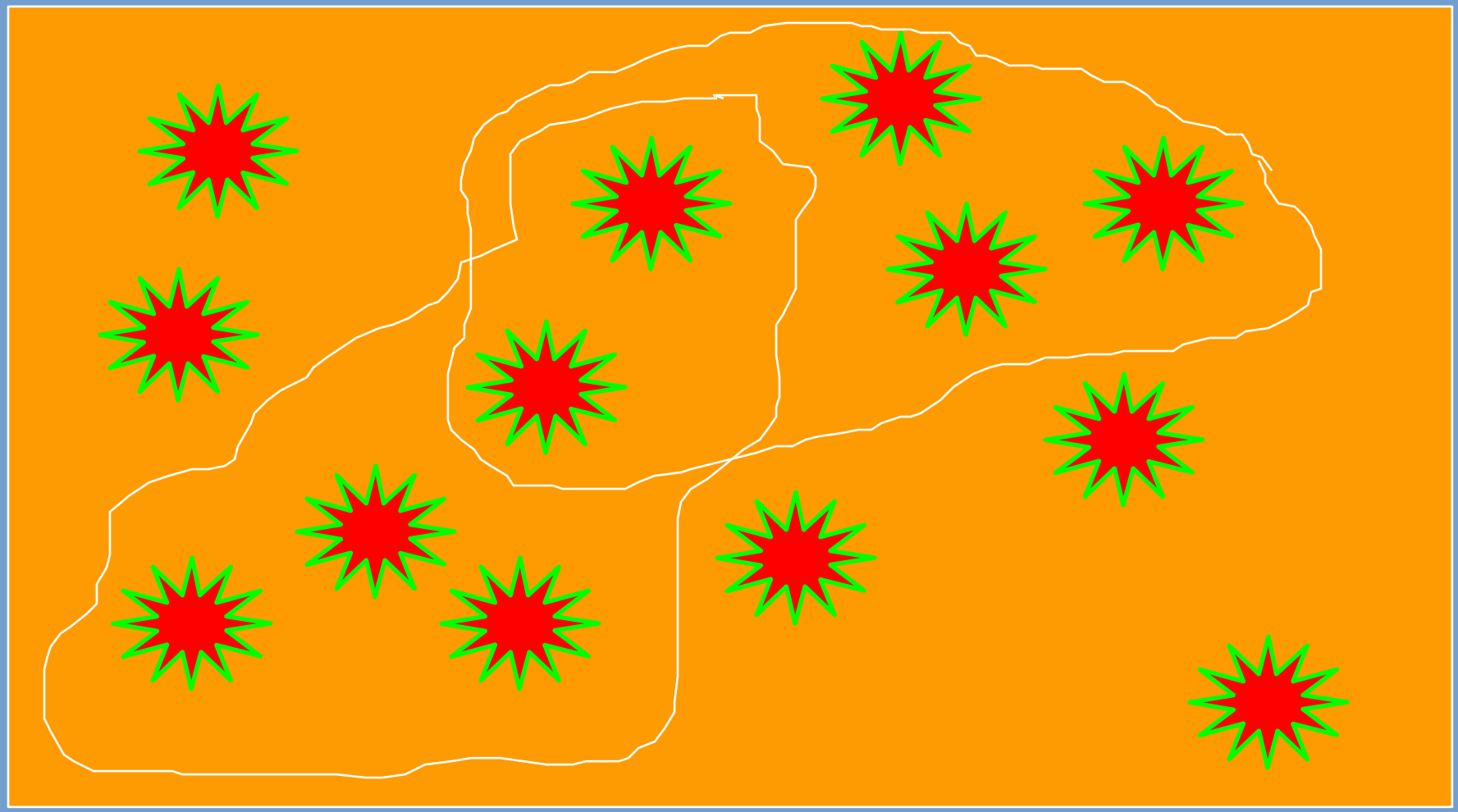
Review:The Smalltalk Image



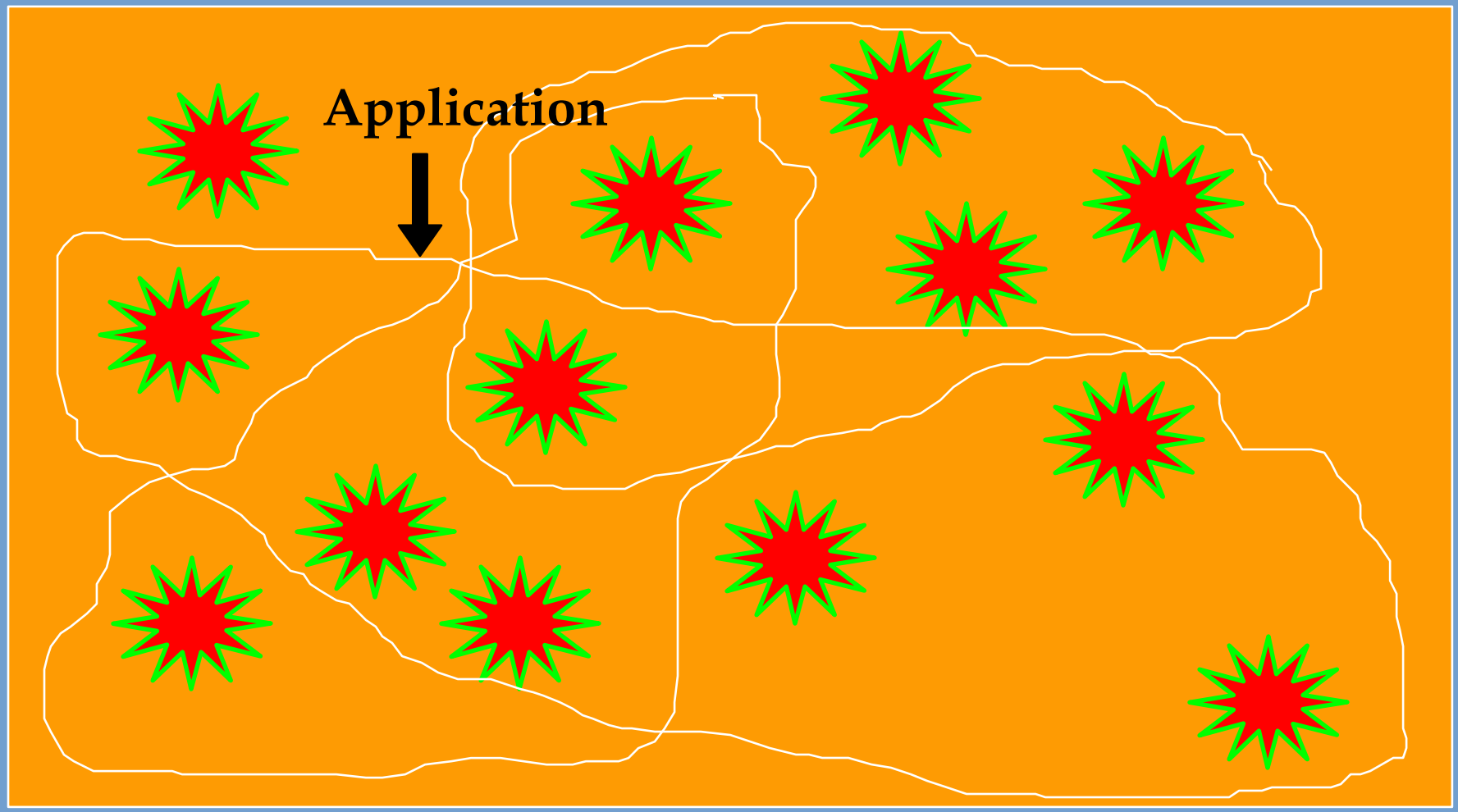
Review:The Smalltalk Image



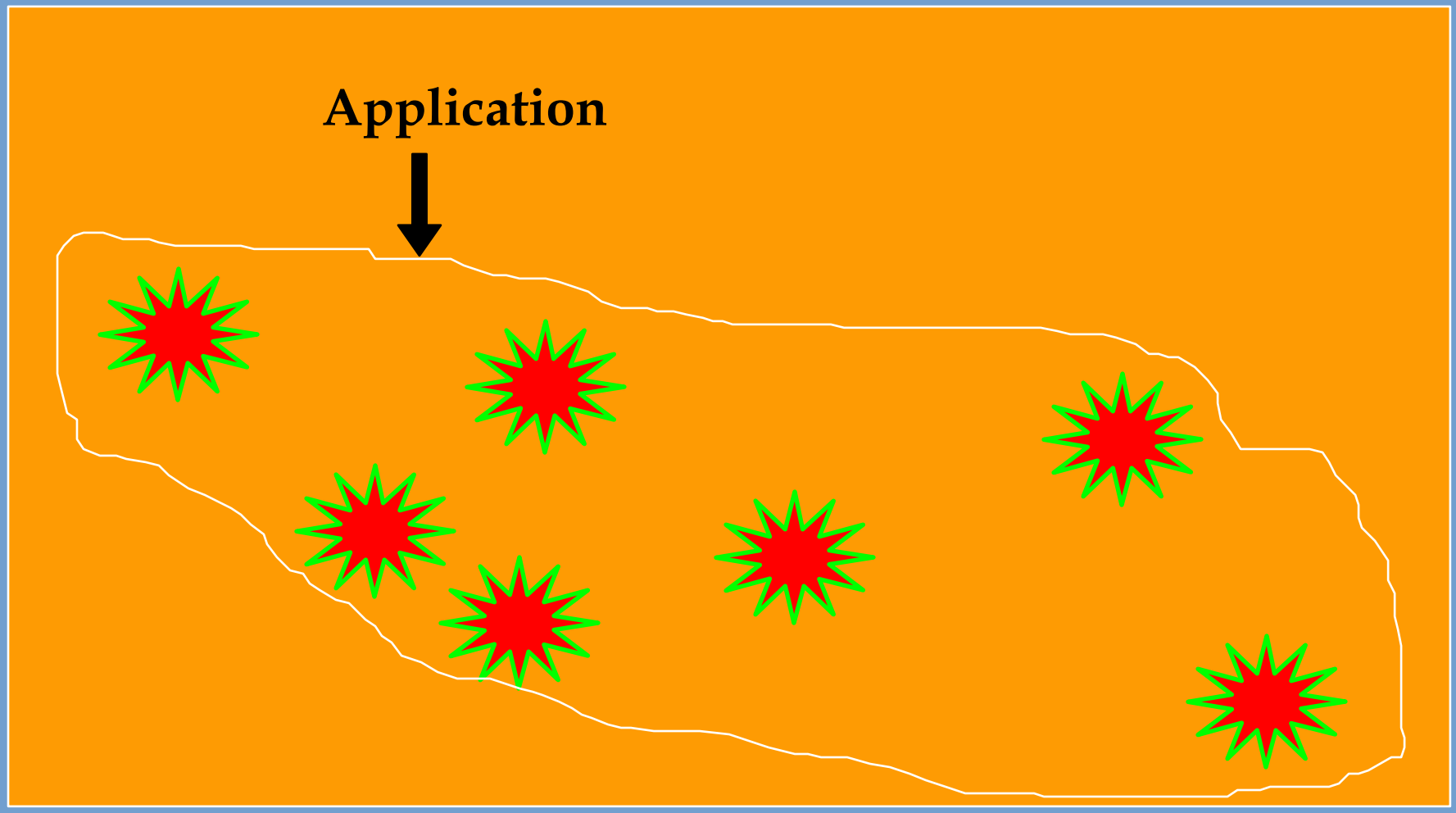
Review:The Smalltalk Image



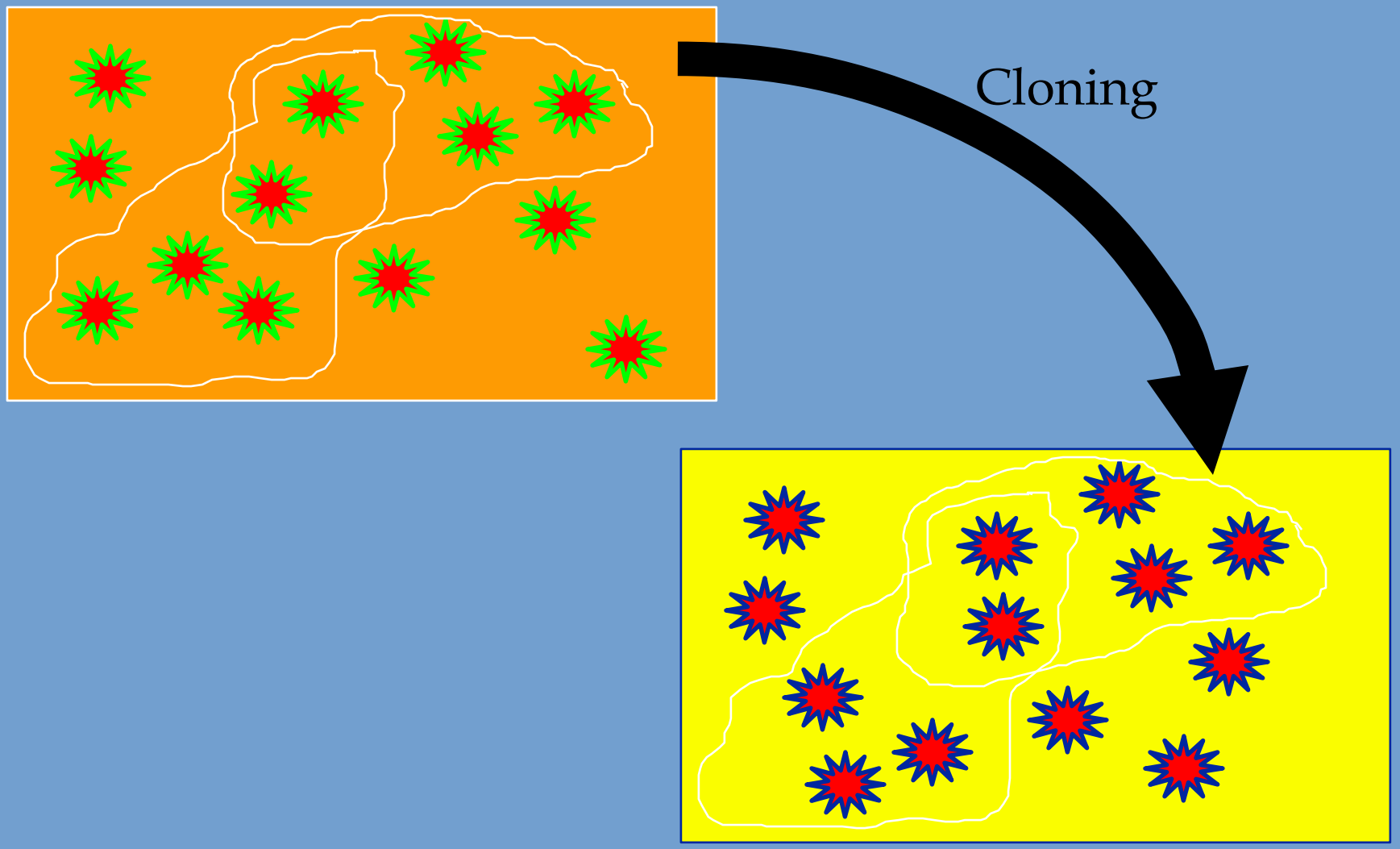
Review:The Smalltalk Image



Application Delivery: Stripping



Smalltalk Maintenance: Cloning



Defining Smalltalk Programs

- Messages are sent whose side-effects are the creation of new Program Elements
  - Classes
  - Methods
  - Global Variables
  - Pools and Pool Variables

Defining Smalltalk Programs

Creating Program Elements

- Interactive using browsers
- Batched using a file containing a sequences to expressions to evaluate
  - ❼"File-in" Format

Defining Classes

```
ApplicationModel
        subclass: #UIPalette
        instanceVariableNames:
            'activeSpecs toolName'
        classVariableNames:  'PaletteOffsets'
        poolDictionaries: '' !

    UIPalette class
        instanceVariableNames: 'selectIcon'!
```

Defining Methods

```
!UIPalette methods!

makeSticky
    UIPainterController modelsSticky
        ifFalse: [UIPainterController modelsSticky: true] !

toolName
    ^toolName ! !
```

Global Variables

- Definition:

  Smalltalk at:#TaskTable put: nil !

- Reference:

  TaskTable == nil

  ifTrue:[TaskTable :=Dictionary new] !

Defining Variable Pools

```
| p |
p := Dictionary new.
p at: 'Red' put: Color red.
p at: 'Blue' put: Color blue.
p at: 'Green' put: Color green.
Smalltalk at: #ColorConstants put: p !
```

Traditional Smalltalk uses an imperative model of program definition.

- An imperative model is a description of an entity that consists of a set of commands that when executed in sequence will reproduce the entity.

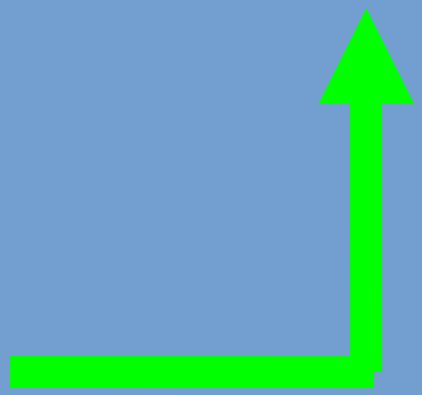
An Imperative Definition of a Geometric Figure

An Imperative Definition of a Geometric Figure

- Move the pen 5 units to the right

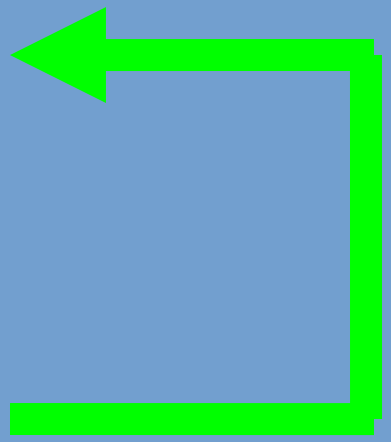
An Imperative Definition of a Geometric Figure

- Move the pen 5 units to the right
- Move the pen 5 units upwards



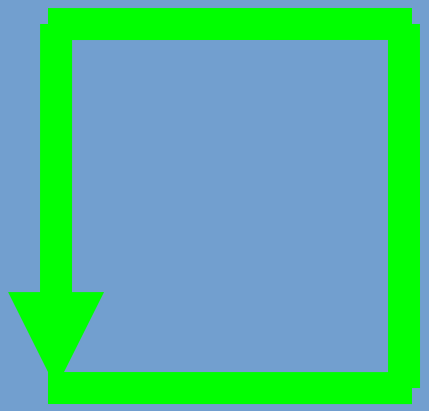
An Imperative Definition of a Geometric Figure

- Move the pen 5 units to the right
- Move the pen 5 units upwards
- Move the pen 5 units to the left



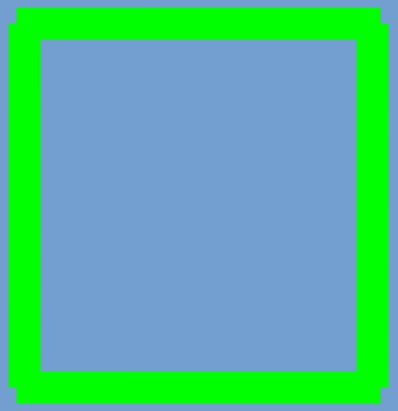
An Imperative Definition of a Geometric Figure

- Move the pen 5 units to the right
- Move the pen 5 units upwards
- Move the pen 5 units to the left
- Move the pen 5 units downward



An Imperative Definition of a Geometric Figure

- Move the pen 5 units to the right
- Move the pen 5 units upwards
- Move the pen 5 units to the left
- Move the pen 5 units downward

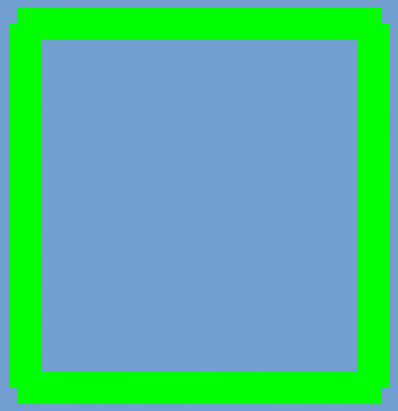


Declarative Models

- A declarative model is a description of an entity that consists of a set of existential statements that enumerate the distinguishing characteristics of the entity.

A Declarative Definition of the Geometric Figure

- A square with sides 5 units in length with green lines 0.1 units thick.



Some Observations

- Imperative Models
  - Describes”how to build it”
  - Constrains the implementation
  - Have initial state dependencies
  - Difficult to analyze

Some Observations

- Declarative Models
  - Describes “what it is”
  - Implementation independent
  - No state dependencies
  - Direct analysis

Many issues arise from Smalltalk's imperative model of program definition.

Issues: Software Engineering

- What is the program?
- Change management.
- Change merging.
- Initial state dependencies.
- Pre-load, post-load and unload actions.

Issues: Software Maintenance

- Can the program be re-generated?
- What is its initial state?
- Class libraries and development environment version dependencies.

Issues: Standardization

- What is an integral part of the Smalltalk language that all implementations must support?
- In what ways are implementations allowed to vary?

Pool implementation example

```
| p |
p := Dictionary new.
p at: 'Red' put: Color red.
p at: 'Blue' put: Color blue.
p at: 'Green' put: Color green.
Smalltalk at: #ColorConstants put: p !
```

Issues: Complexity

- Too many implementation artifacts are "in the programmer's face"
  - Metaclass hierarchy
  - CompiledMethods
  - Method Dictionaries
  - Pool Dictionaries
  - Symbols
  - Development Environment classes

Declarative Specification of Programs

- "Normal" programming language use a declarative model:
  - FORTRAN
  - Pascal
  - C
  - COBOL
- Could a declarative model be used for Smalltalk programs?

Should a Declarative Model be used for Smalltalk

"Unfortunately, to describe a system on paper, a noninteractive linear mode of presentation is needed. To this end, a *basic class template* is provided..."

⑨From "The Smalltalk-80 System", The Xerox Learning Research Group, *Byte*, August 1981:

The Smalltalk-80 Class Template

class name	Point
instance variable names	x y
methods	

```
x: xCoordinate y: yCoordinate
    x := xCoordinate.
    y := yCoordinate

+ aPoint |sumX sumY|
    sumX := x + aPoint x.
    sumY := y + aPoint y.
    ^Point newX: sumX Y: sumY
```

Making Smalltalk Declarative

- Identify all language elements and define syntax
- Define static and runtime semantics
- Map onto existing and future implementations

Smalltalk Execution Environment

- Objects - state+behavior
  - Described by class definitions
  - Statically created
    - Literals
    - Class objects
  - Dynamically created
- Variables - store object references
- Thread(s) of execution

Smalltalk Language Elements

- A Smalltalk Program consists of
  - ❼Class and method definitions
  - ❼Global Variable Definitions
  - ❼Pool Definitions
  - ❼An initialization sequence

Abstract versus Concrete Syntax

- We choose to only define an abstract syntax for Smalltalk program and to not require single linear concrete syntax.
- The abstract syntax provides a means for describing all elements of a Smalltalk program.

Class Definition: Syntax

<class definition> ::=
    <class name> [<superclass name>]
        [<instance variables>]
        [<class instance variables>]
        [<class variables>]
        [<imported pools>]
        [<instance methods>]
        [<class methods>]
        [<class initializer>]

Class Definition: Semantics

- <class name> is defined as a global name
- The execution time binding of <class name> is to the class object
- The binding of <class name> is fixed
- The behavior of instances consists of the instance behavior in the class definition named <superclass name> augmented by the <instance methods> ...
- etc.

Class Definition: Static Semantics

- Errors:
  - <class name> is duplicately defined
  - <superclass name> is not defined as a <class name>
  - <superclass name> and <class name> are the same name
  - <superclass name> is the name of a class that inherits from this class
  - etc.

Class Definition: Implementation Options

- Representation and location of methods
- Metaclasses?
- Inheritance (lookup or copy down?)
- Representation of message selectors
- etc.

Global and Pool Definitions

- <global definition> ::=
  <global variable names>
  [<variable initializer>]
- <pool definition> ::= <pool name>
  <pool variable definition>*

  <pool variable definition> ::=
  < pool variable names>
  [<variable initializer>]

Smalltalk Programs

- <Smalltalk program> ::= <program element>+
  <program element> ::= <class definition> |
  <global definition> |
  <pool definition>
- Element ordering determines execution time initialization order.

Unnecessary Implementation Assumptions

- A "system dictionary" named Smalltalk exists
- All class, global variables, and pools are elements of the system dictionary
- Pools are implemented using class Dictionary
- Global and pool variables are implemented as instances of class Association

More Unnecessary Assumptions

- Methods are objects
- Methods are stored in a method dictionary
- An object's behavior is implemented by a class object
- Each class has an associated metaclass
- The definition of a program may dynamically change through reflection

Smalltalk Standardization

- X3J20 - The "ANSI Smalltalk" committee
- Targeted Completion 1997
- Will use the "declarative model" of Smalltalk programs

Program Interchange

- X3J20 is defined in terms of an abstract declarative program syntax
- For interchange purposes it defines a concrete, *textual,* interchange format
- The abstract syntax could also be the basis for an implementation independent, *binary,* interchange format

What about Reflection?

- Reflection - the ability of a program to dynamically inspect (and modify?) its own implementation.
- Smalltalk was reflective before before any of us know what “reflection” was!

Reflection versus the Declarative Model

- The imperative model of Smalltalk is inherently reflective.
- Reflection occurs dynamically as a program executes.
- The declarative model describes a program staticly prior to execution.
- The declarative model neither requires nor precludes reflection.

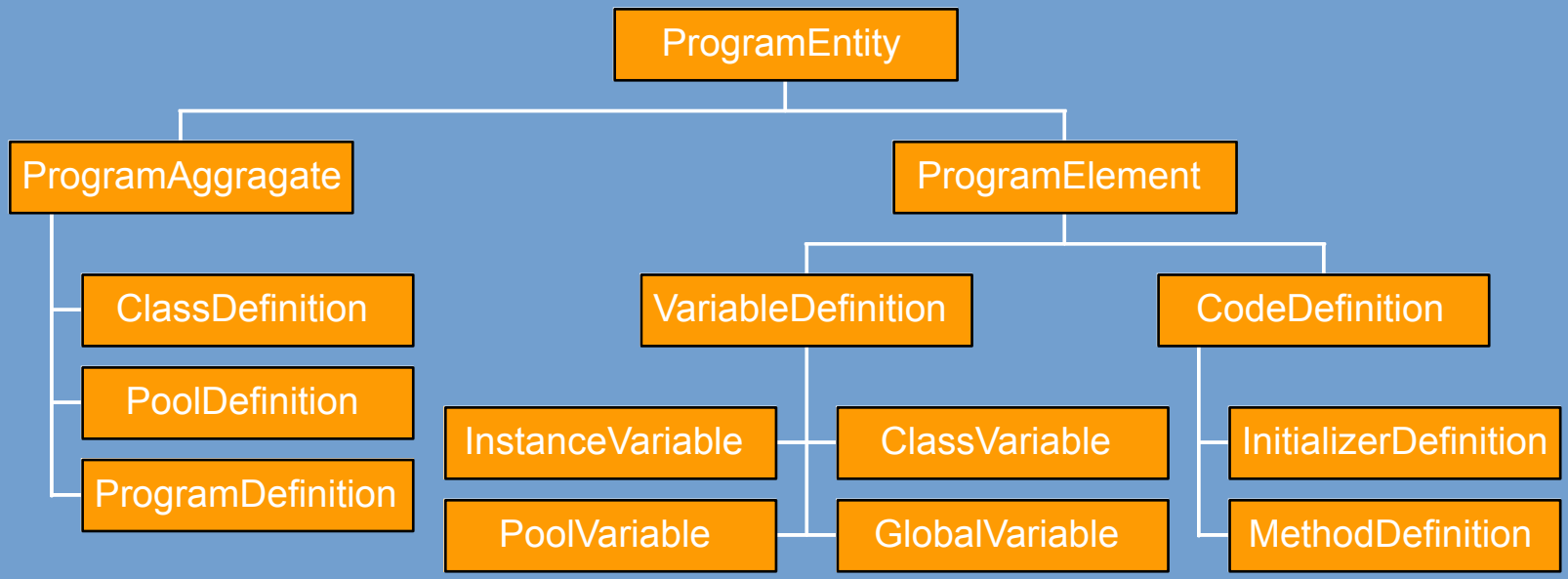
Reflection: No change required

- An implementation may continue to use traditional object models to represent the implementation artifacts of a Smalltalk program
  - ❼Metaclasses
  - ❼Method Dictionaries
  - ❼Variable Dictionaries
  - ❼etc.
- They may be reflectively manipulated

Reflection: Doing Better

- Traditional Smalltalk reflection is inherently implementation dependent
  - ❼ An object model of the implementation artifacts
- Why not objectify the abstract declarative description of a Smalltalk program?

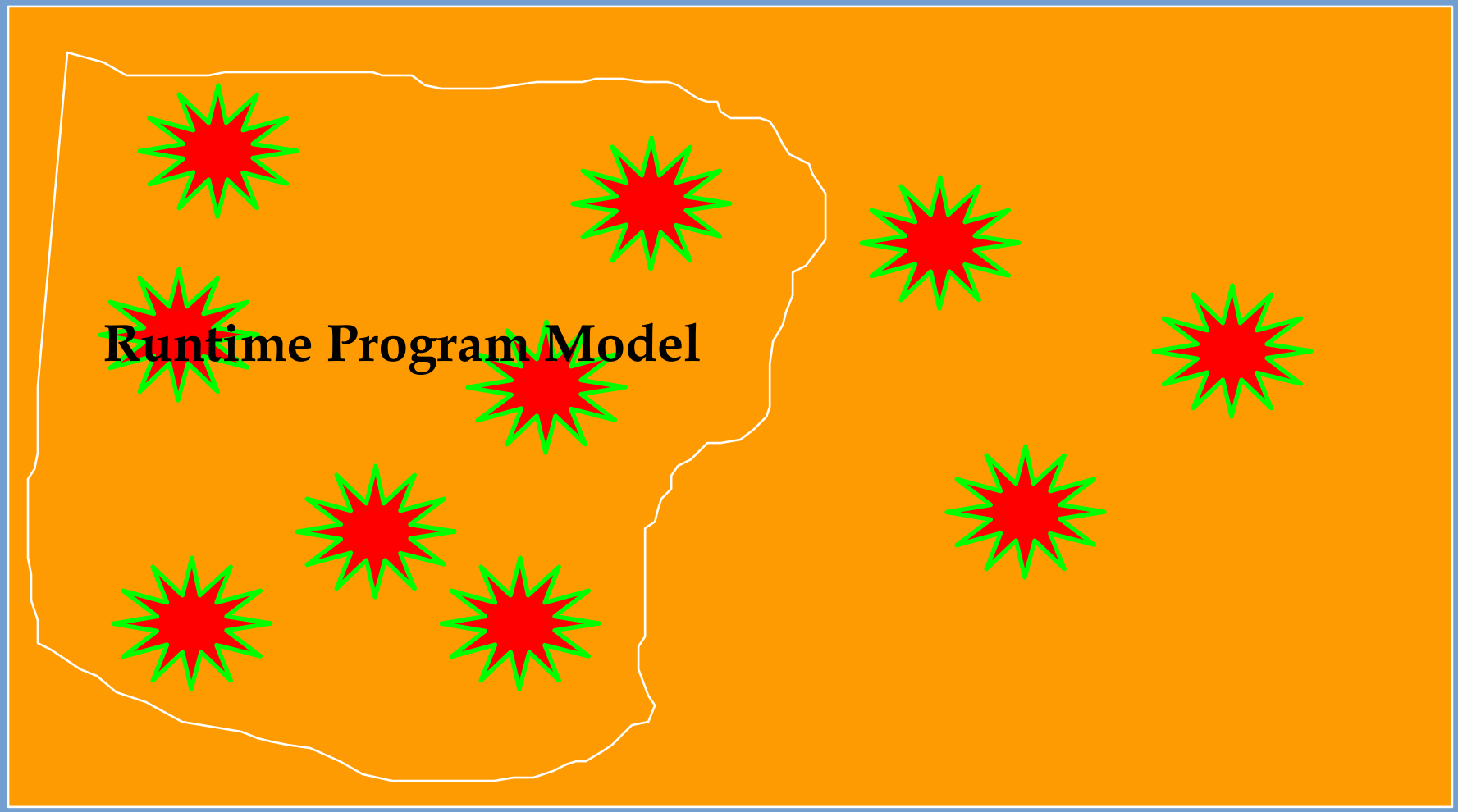
An Abstract Object Model for Smalltalk Programs



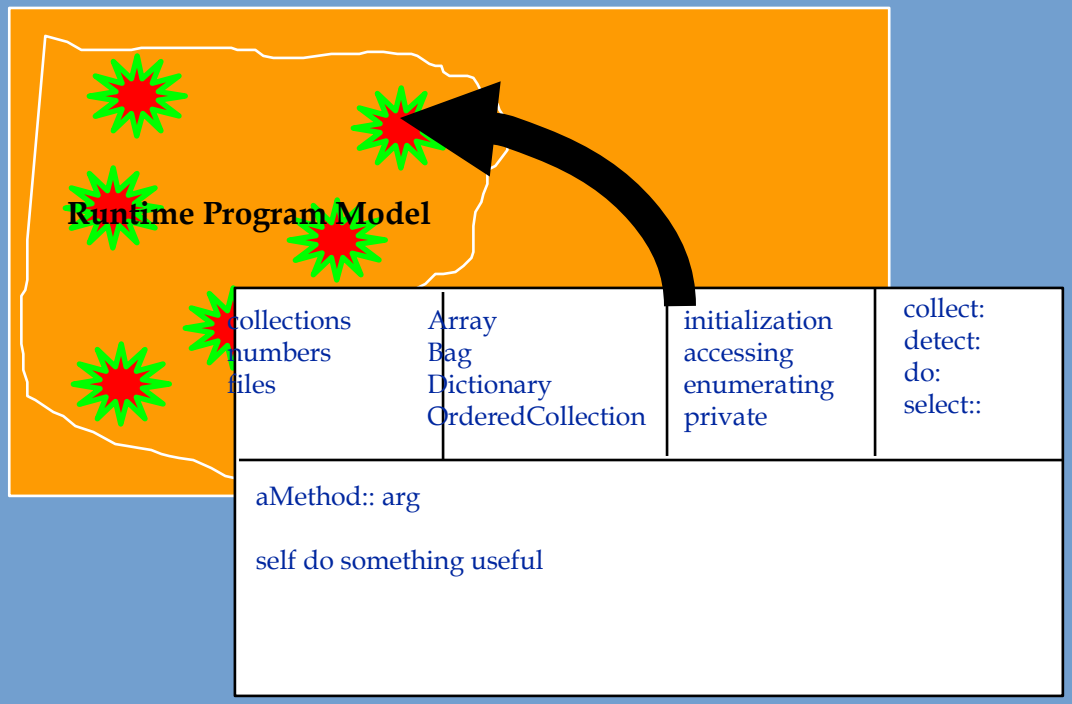
Improving the Development Environment

- The primary use of reflection in Smalltalk has been the implementation of the Smalltalk development environment.
- What happens when we apply the declarative program model for reflection within the development environment.

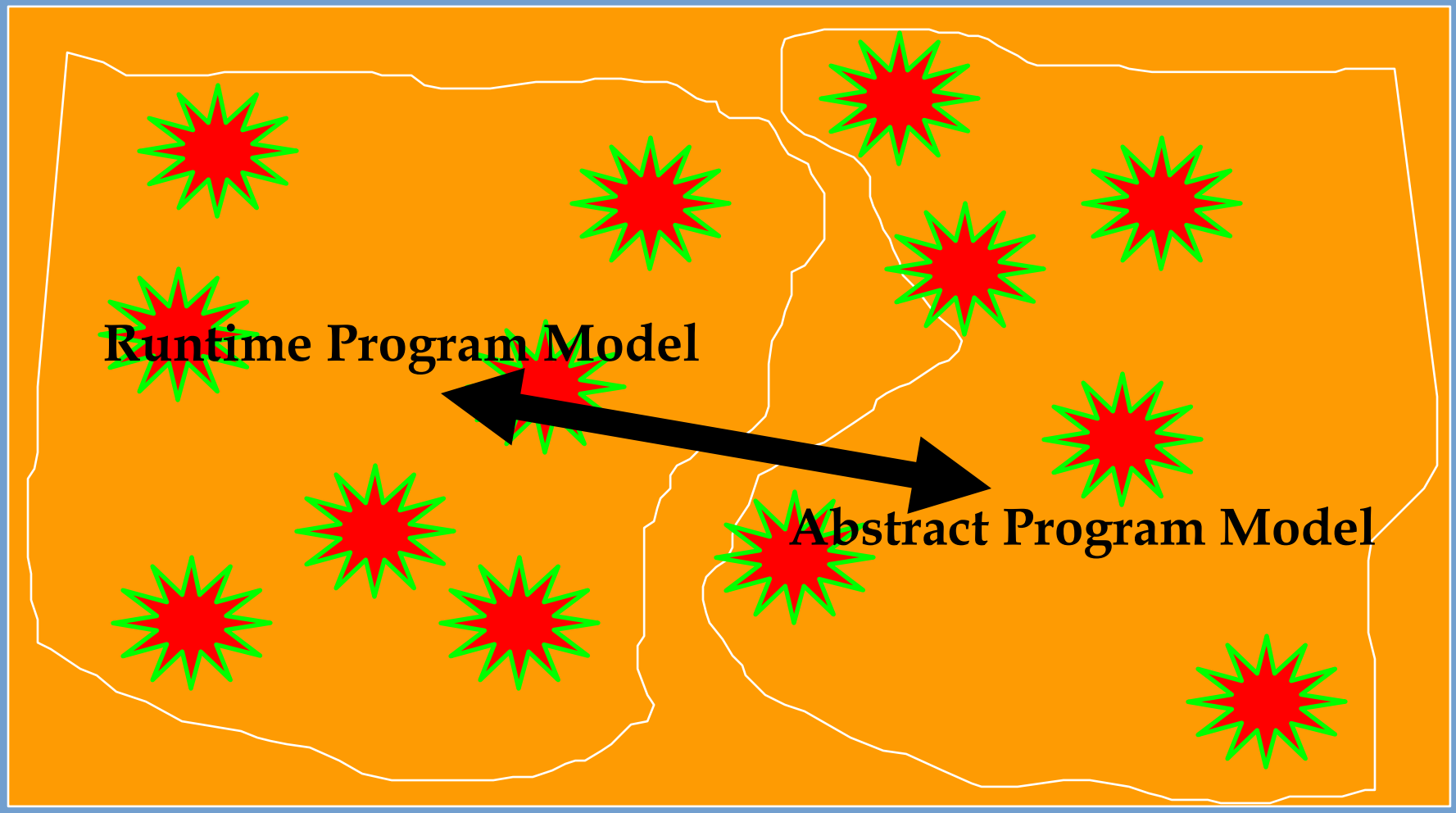
Smalltalk Image Program Model



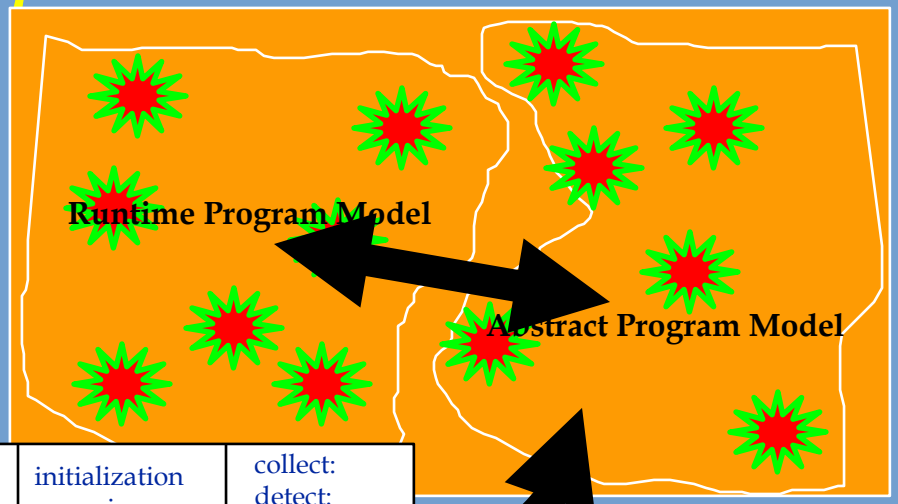
Tradition Smalltalk Tools Manipulate the Runtime Model



Alternative Program Object Models



Better Smalltalk Tools Manipulate the Abstract Model

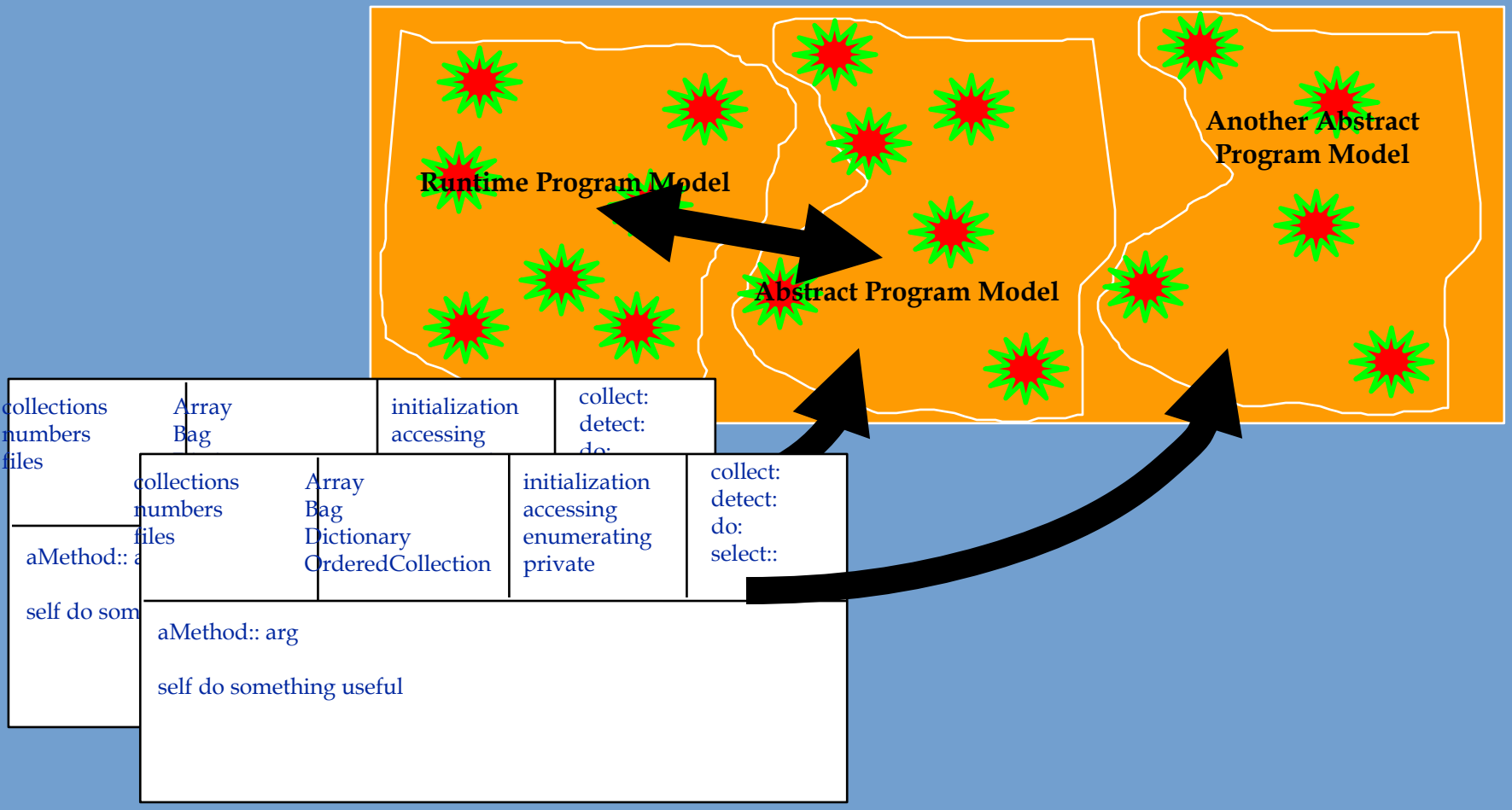


collections numbers files	Array Bag Dictionary OrderedCollection	initialization accessing enumerating private	collect: detect: do: select::

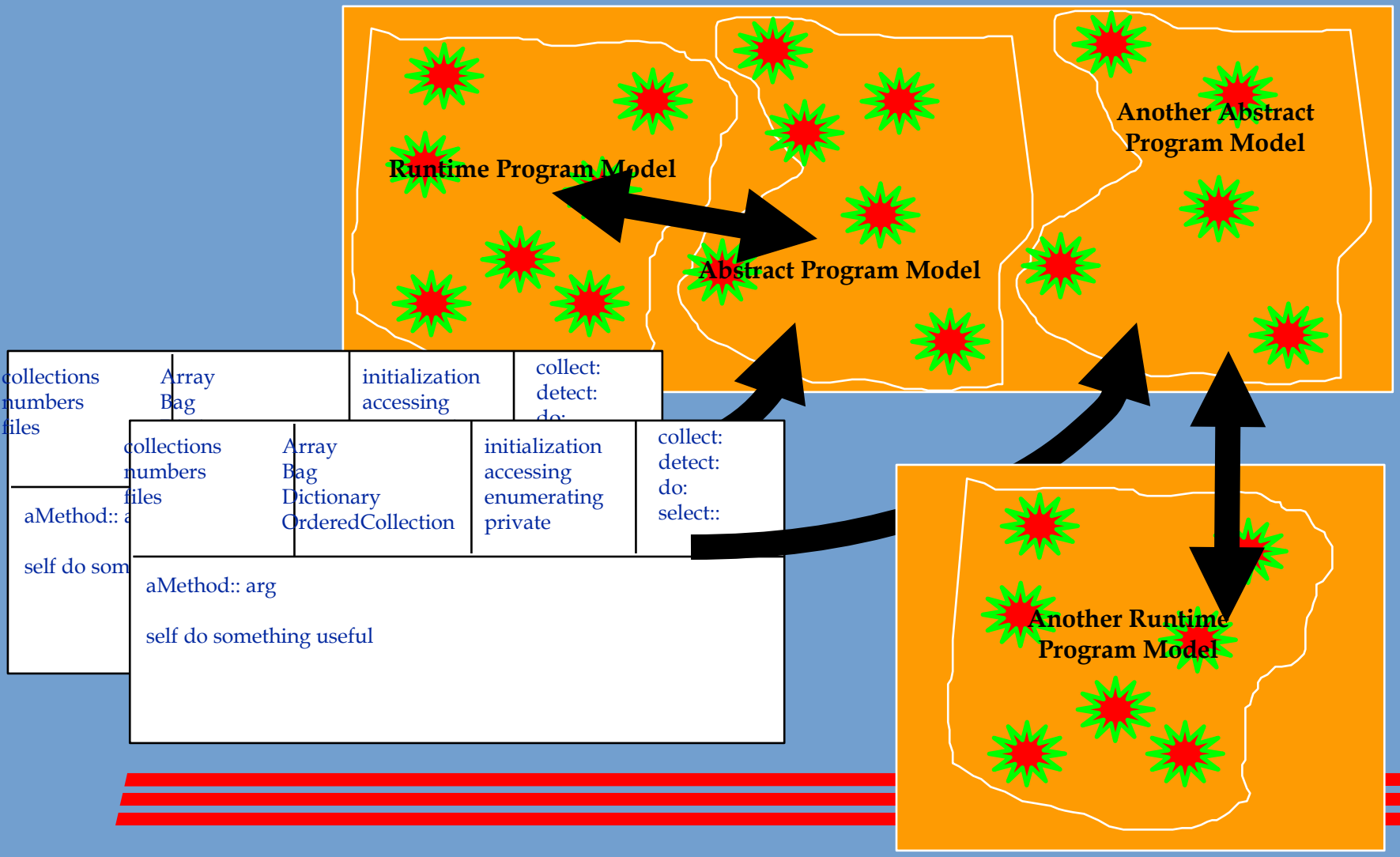
```
aMethod:: arg

self do something useful
```

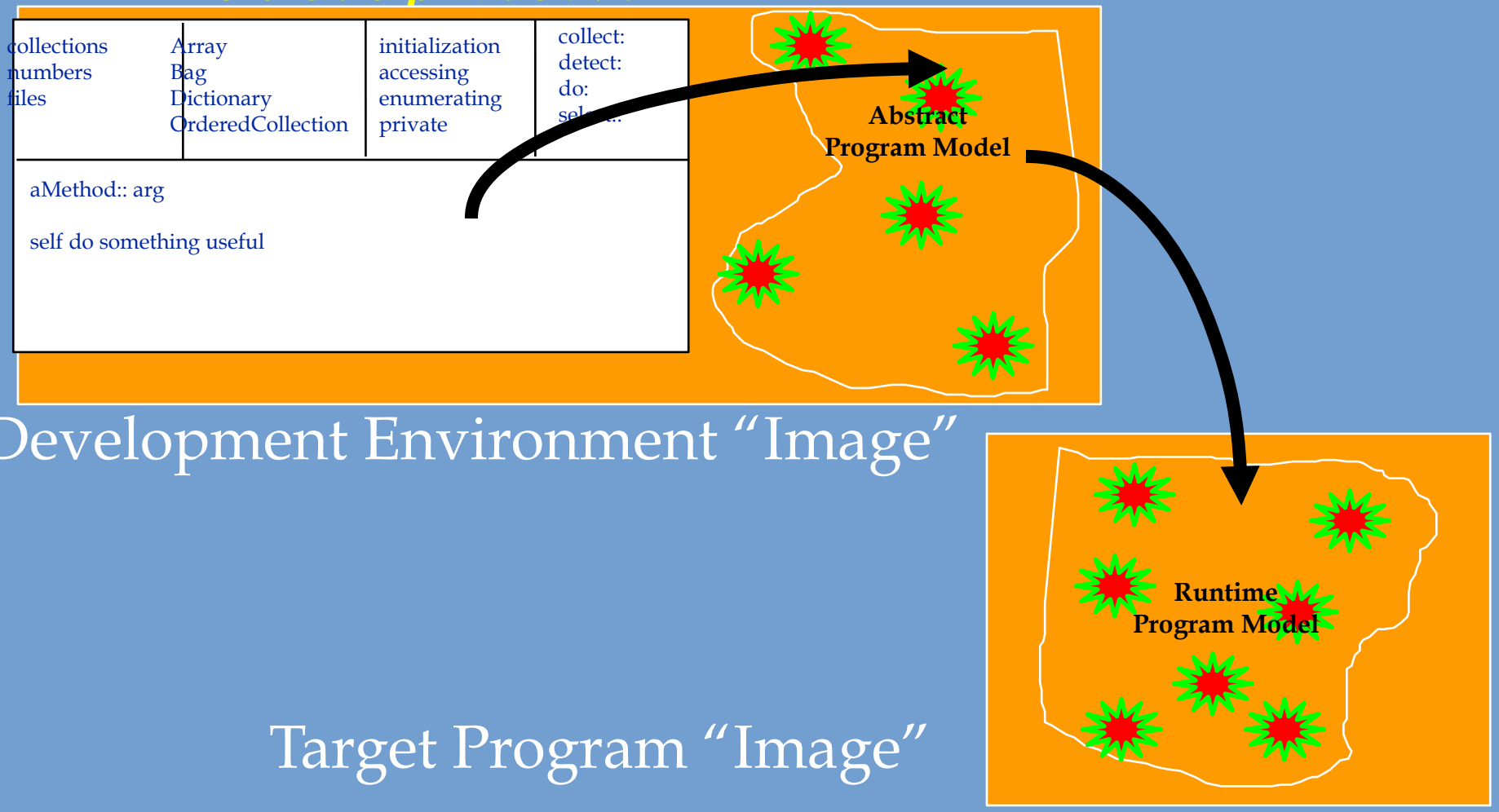
Multiple Abstract Program Models



Making It Executable



A New Architecture for Smalltalk Development



Development Environment "Image"

Target Program "Image"

New Architecture Characteristics

- Users construct a declarative definition of a Smalltalk program instead of editing an image.
  - ❼Programs are completely specified
  - ❼Reproducible from source code
  - ❼Non-loadable programs are editable
  - ❼No “stripping” required for delivery

New Architecture Characteristics

- Target program class library is separate and distinct from the class library used to implement the development environment.
  - Target program changes do not impact development tools.
  - Development tool changes do not impact target program
  - Release and/or vendor class library decoupling

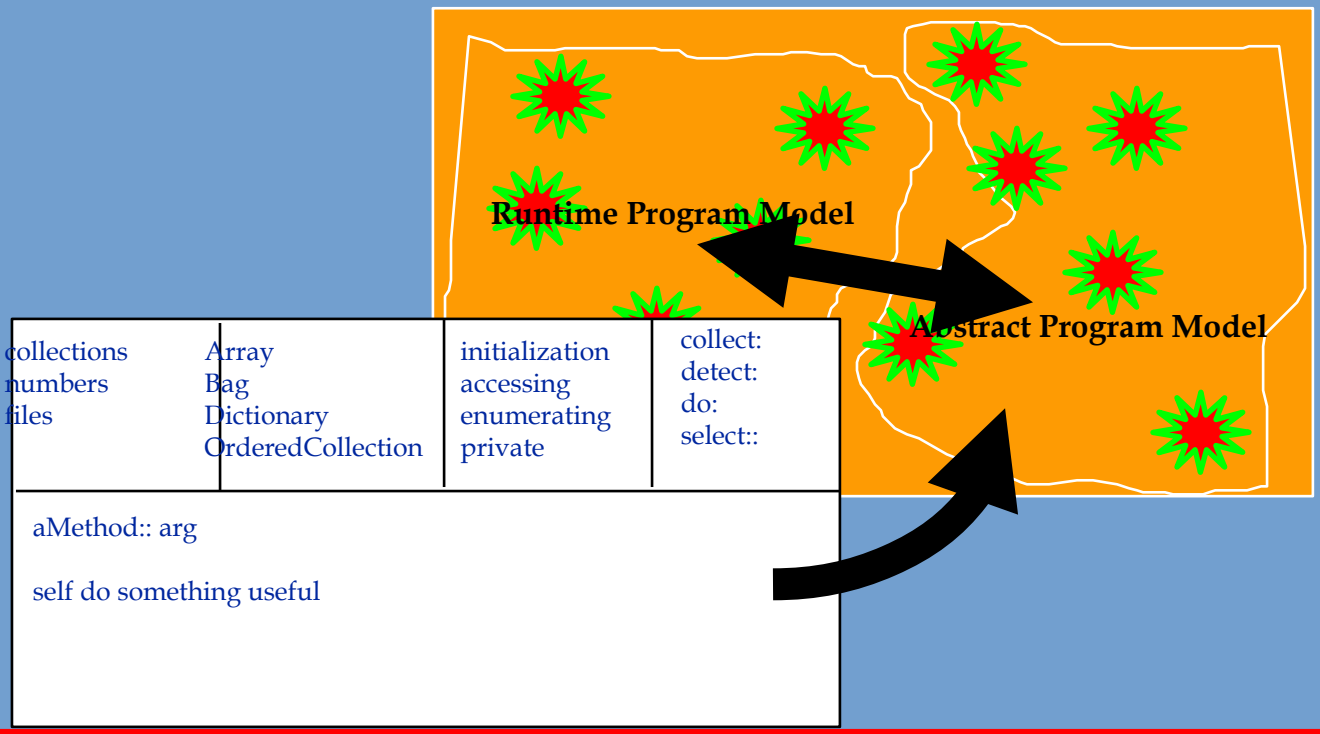
New Architecture Characteristics

- Simplified Class library - No tools or runtime implementation classes visible to application programmer.
- Traditional Smalltalk fully incremental, interactive program creation, testing, and debugging.
- Target program failure will not crash development environment

Is the Architecture Feasible?

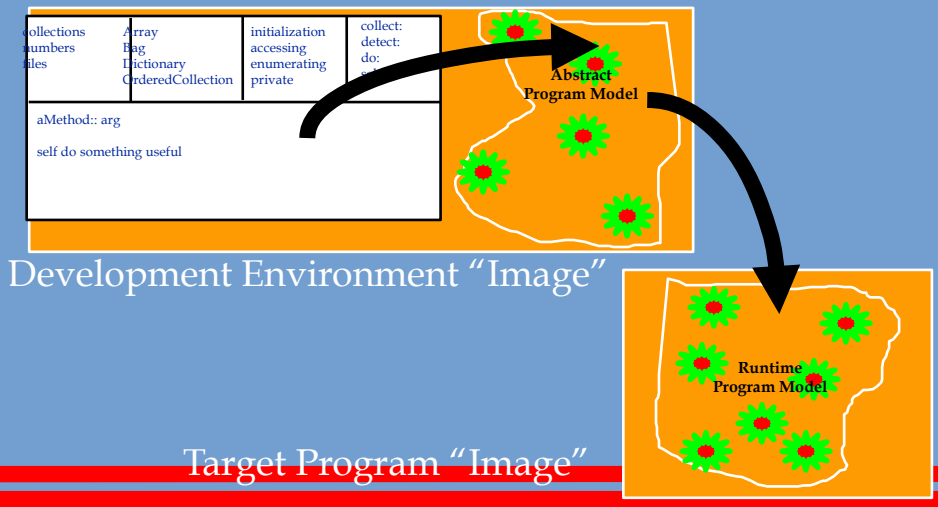
Is the Architecture Feasible?

- "Team/V" in production since 1993
  - Declarative program specification
  - Abstract Program Object Model



Is the Architecture Feasible?

- "Firewall" prototype operational
  - Target program fully decoupled from development tools
  - Target program executes in separate process.
  - Full incremental programming and debugging



Development Environment "Image"

Target Program "Image"

"Firewall" Accomplishments

- Very small application program images
  - "3+4" image < 10K
  - Utilities & applets 30K - 200k
  - Full GUI Applications 500k-2m
- "Digitalk" Smalltalk application edited in a "ParcPlace" Smalltalk hosted development environment

"Firewall" Accomplishments

- First complete regeneration of a "Xerox Smalltalk" system from source code since 1976?

Conclusions

- The adoption of the declarative model is the latest example of Smalltalk's ability to evolve and adapt.

Conclusions

- Smalltalk will continue to be the benchmark against which other object-oriented programming language and environments are measured.