FACULTY OF INFORMATION TECHNOLOGY
CZECH TECHNICAL UNIVERSITY IN PRAGUE

A Virtual-Machine Metaobject Protocol for Run-Time Software Adaptation

Guido Chari - BeHapi ’19

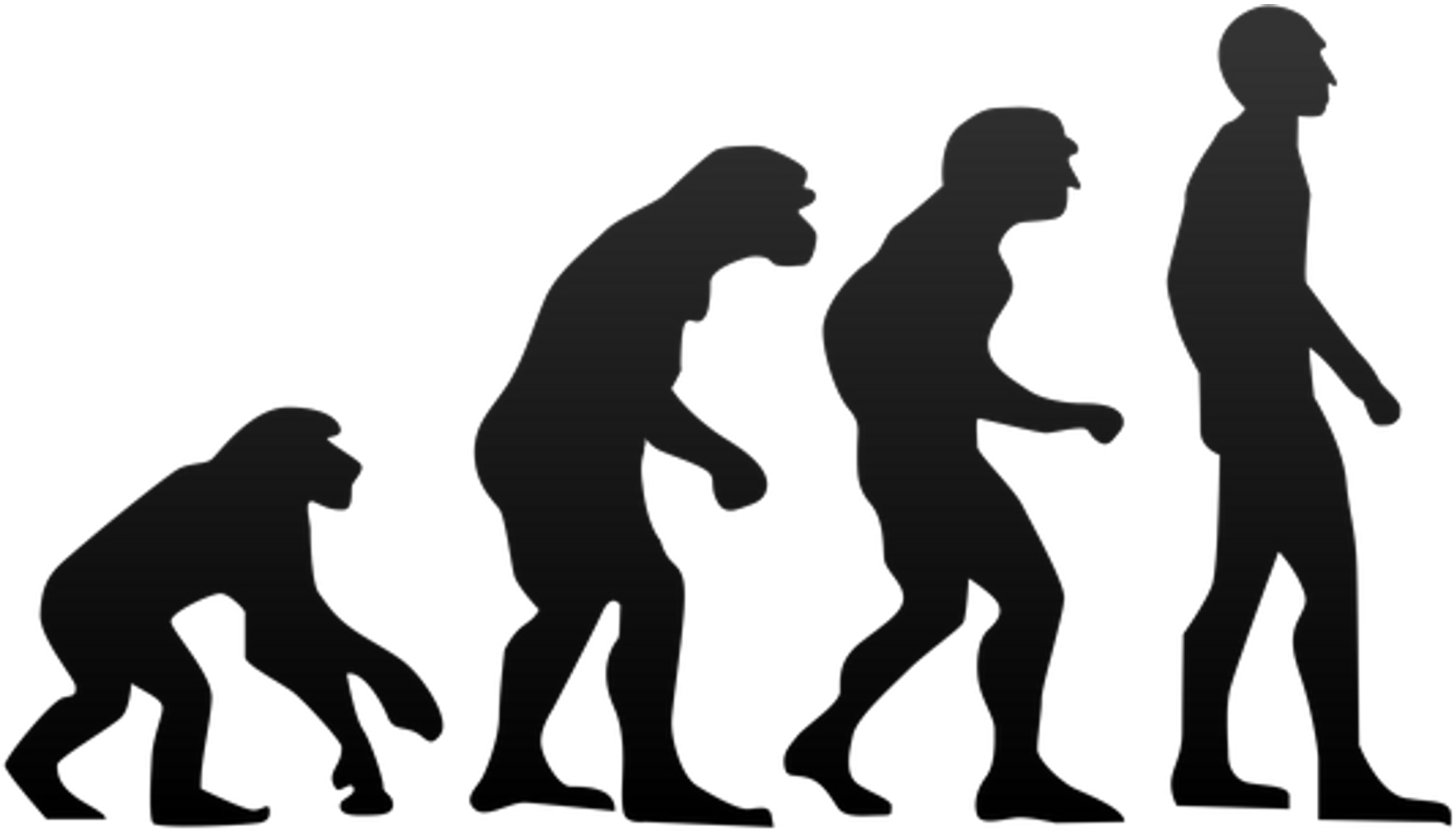
Diego Garbervetsky, Stefan Marr, Stepane Ducasse

FACULTY OF INFORMATION TECHNOLOGY
CZECH TECHNICAL UNIVERSITY IN PRAGUE

An (H)API for Run-Time Software Adaptation

Guido Chari - BeHapi ’19

Diego Garbervetsky, Stefan Marr, Stepane Ducasse



"…research should focus on providing support to software at run-time, **breaking today's rigid boundary between development-time and run-time…**"

The disappearing boundary between development-time and run-time, Luciano Baresi and Carlo Ghezzi FOSER '10.

Approaches

Dynamic adaptation
programatically

- **Code/Class Loading**
- **Eval**
- **Reification**
- **AOP/COP**

Approaches

Dynamic adaptation
programatically

- **Code/Class Loading**
- **Eval**
- **Reification**
- **AOP/COP**

Reflection

"Knowing yourself is the beginning of all wisdom."

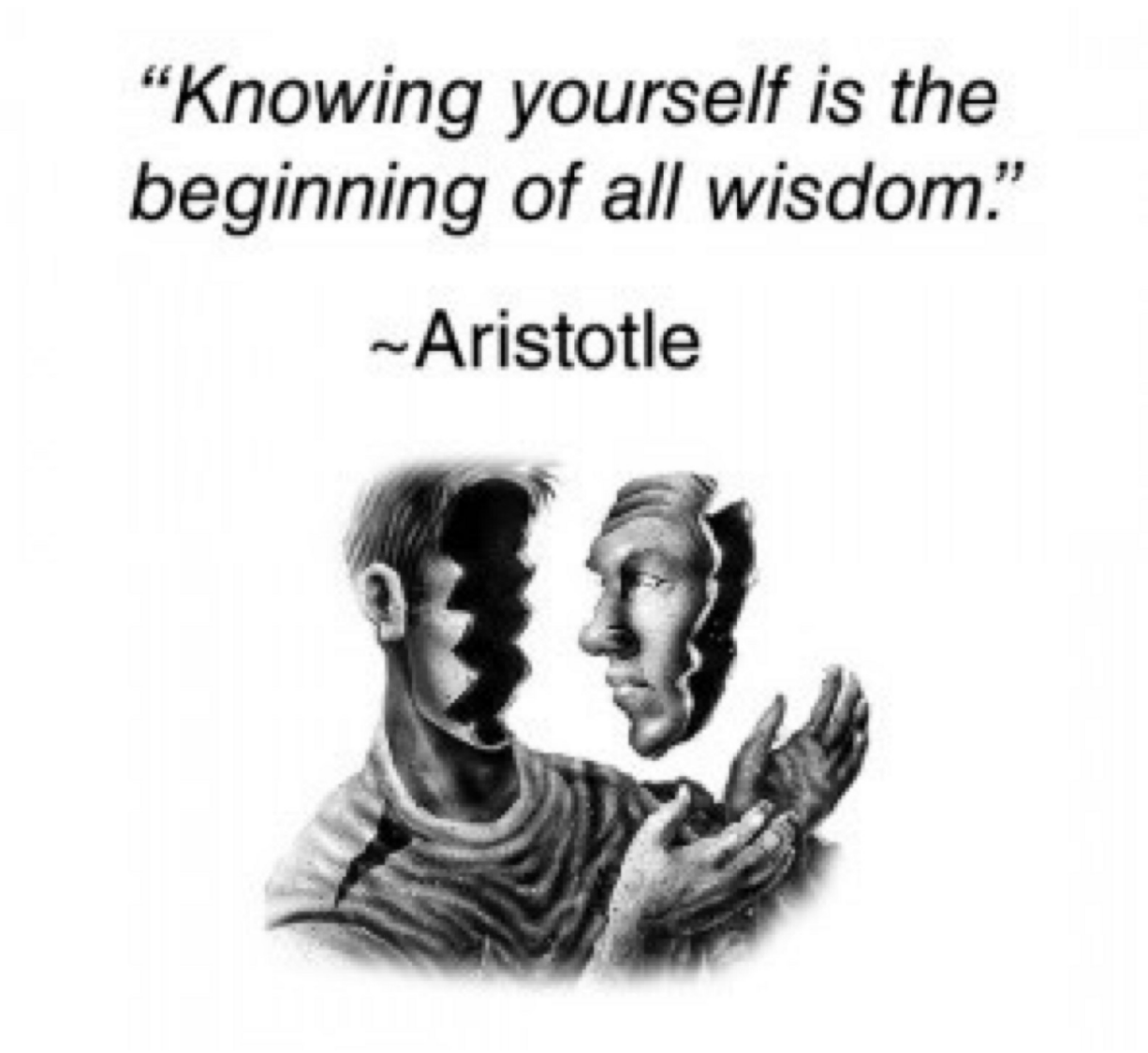
~Aristotle

Reflection

- The ability of programs to:
  - Observe (**introspect**) and/or
  - Adapt (**intercede**)

their own

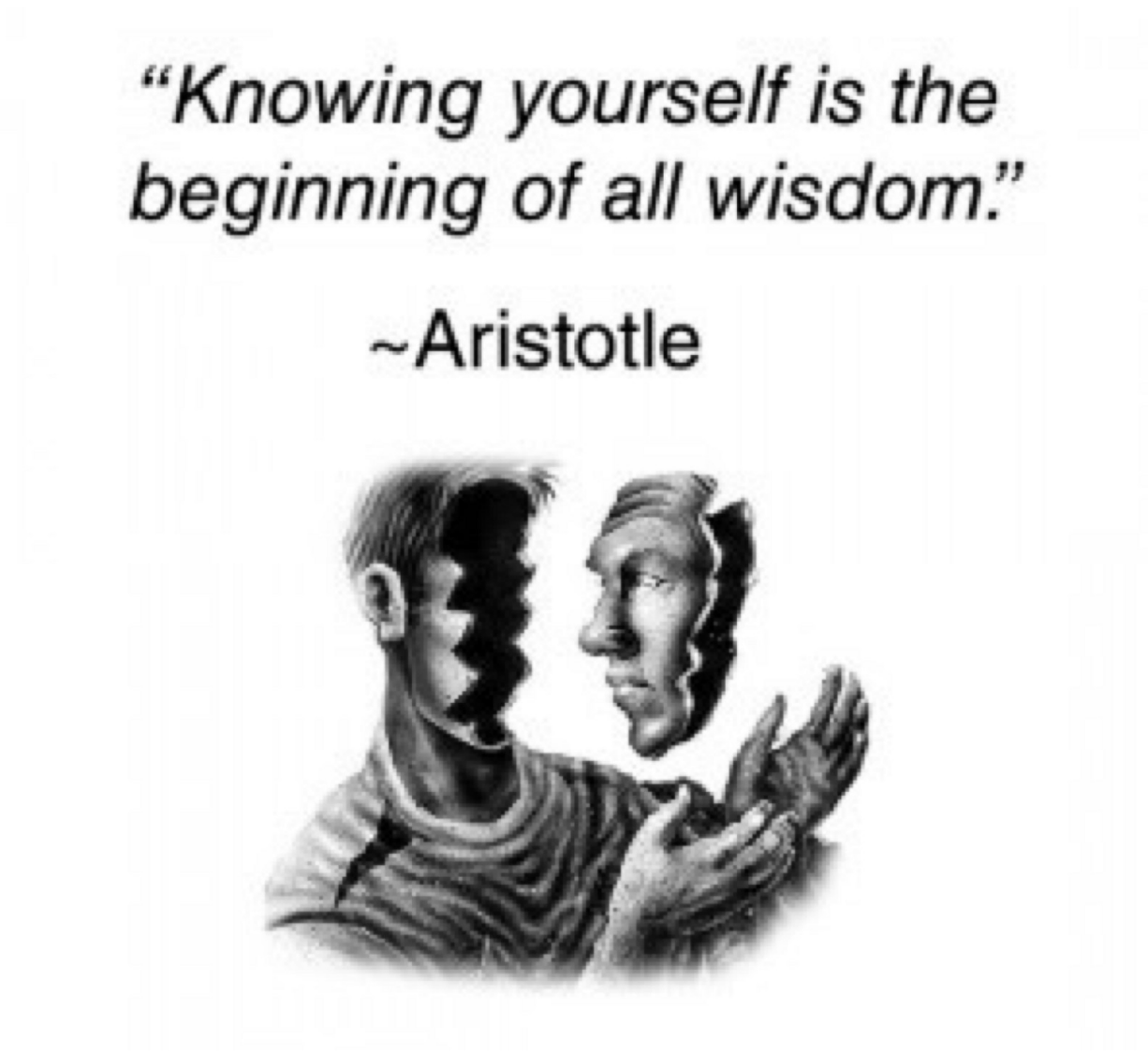
  - **Behavior** and/or
  - **Structure**.



Reflection

- The ability of programs to:
  - Observe (**introspect**) and/or
  - Adapt (**intercede**)

  their own
  - **Behavior** and/or
  - **Structure**.



Javascript, Python, Ruby, Java, C#, Lua, PHP, Rust?, etc.

Limitations

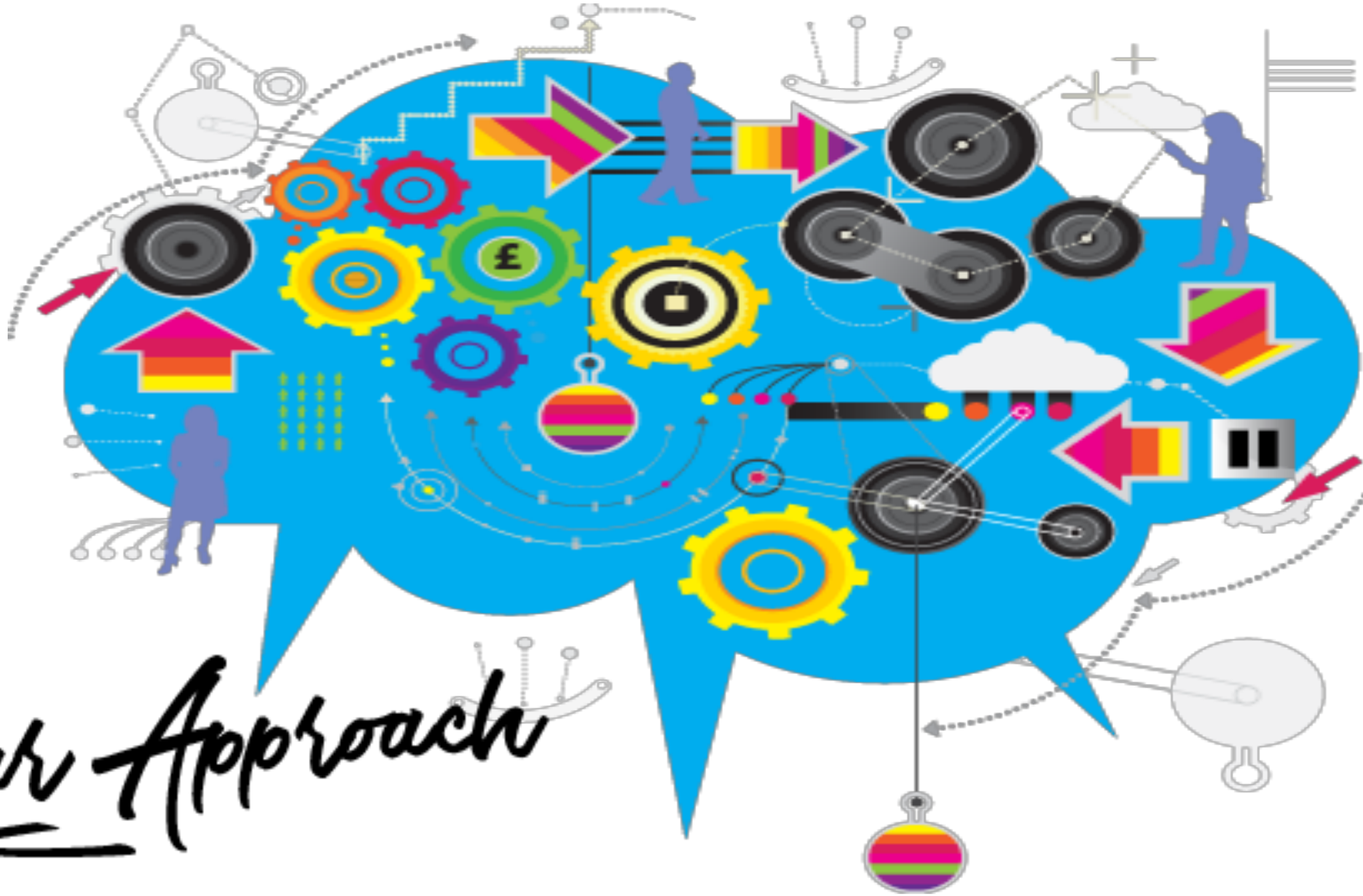
Run-Time Adaptation Scenarios

- **Security**: Immutability, tainting
- **Maintenance**: Profiling, debugging
- **Performance:** Columnar objects (time), sparse objects (space)
- **Language evolution:** typing schemes (gradual typing)

Limitations

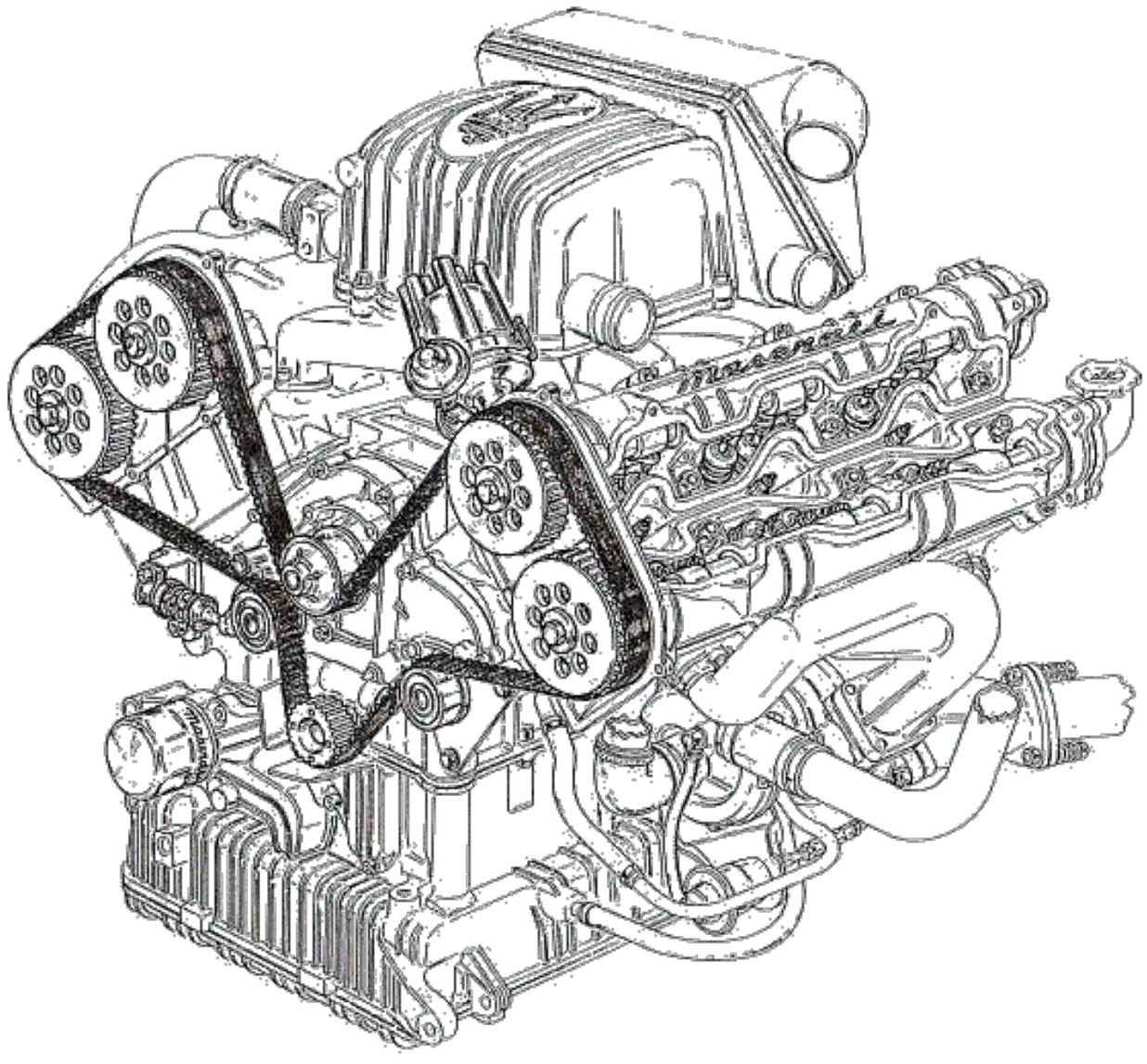
Semantics ***Semantics***

Semantics



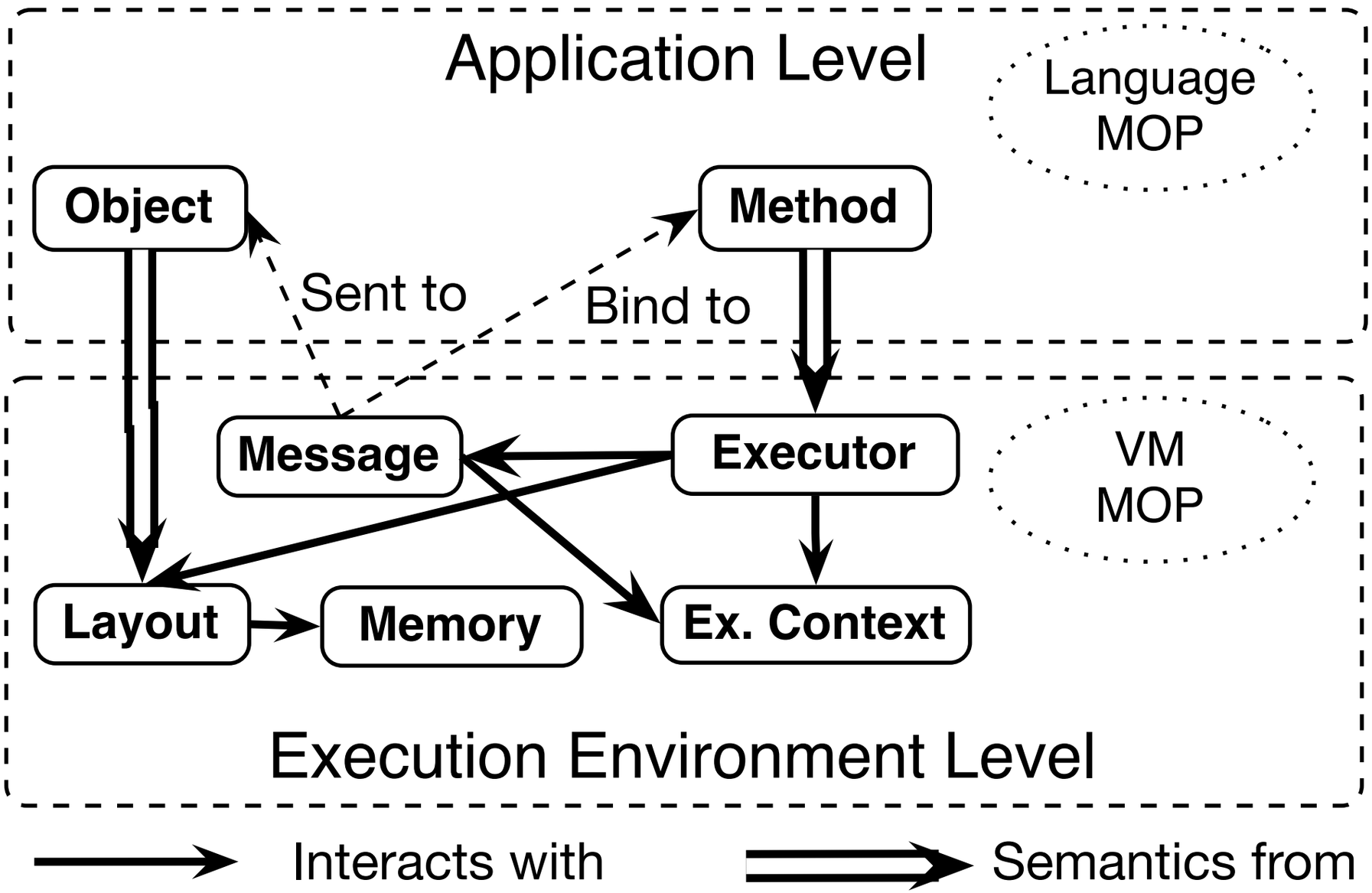
Our Approach

Virtual Machine

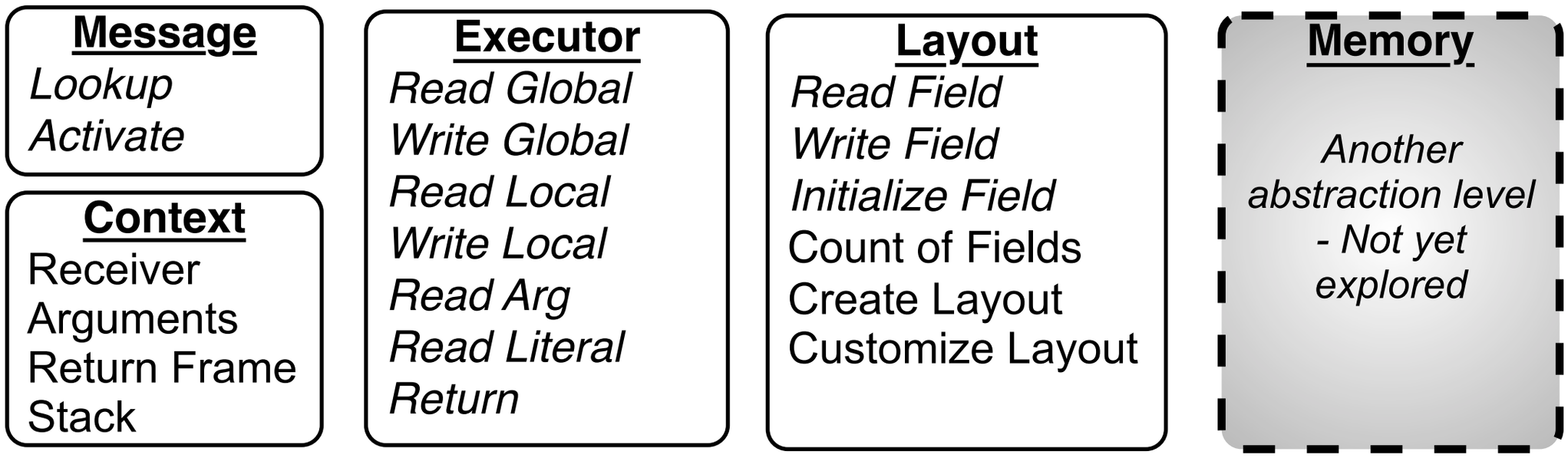


Fully Reflective Virtual Machine

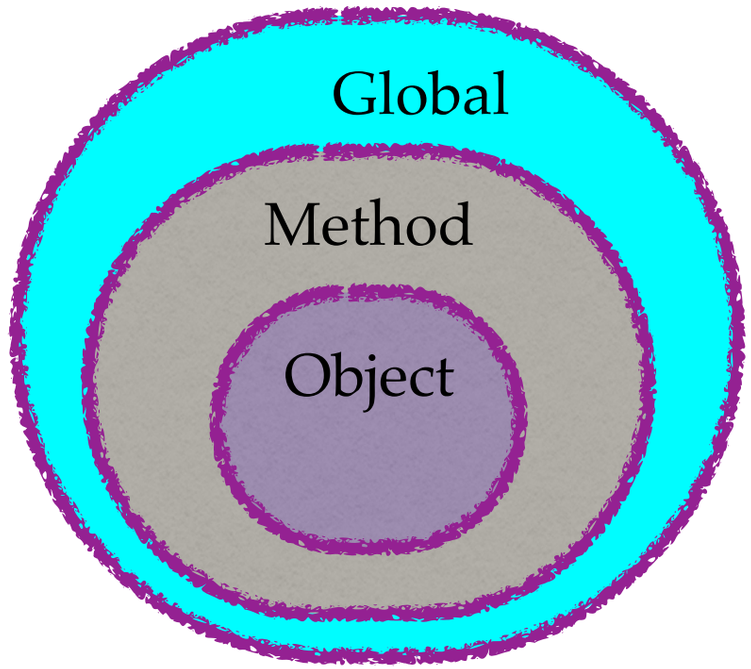
Fully Reflective Virtual Machine



API for Semantics Adaptation



API for Semantics Adaptation



Message
Lookup
Activate

Context
Receiver
Arguments
Return Frame
Stack

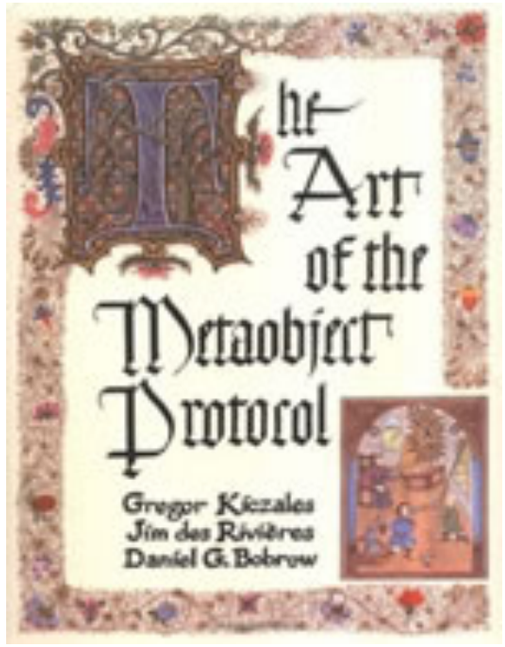
Executor
Read Global
Write Global
Read Local
Write Local
Read Arg
Read Literal
Return

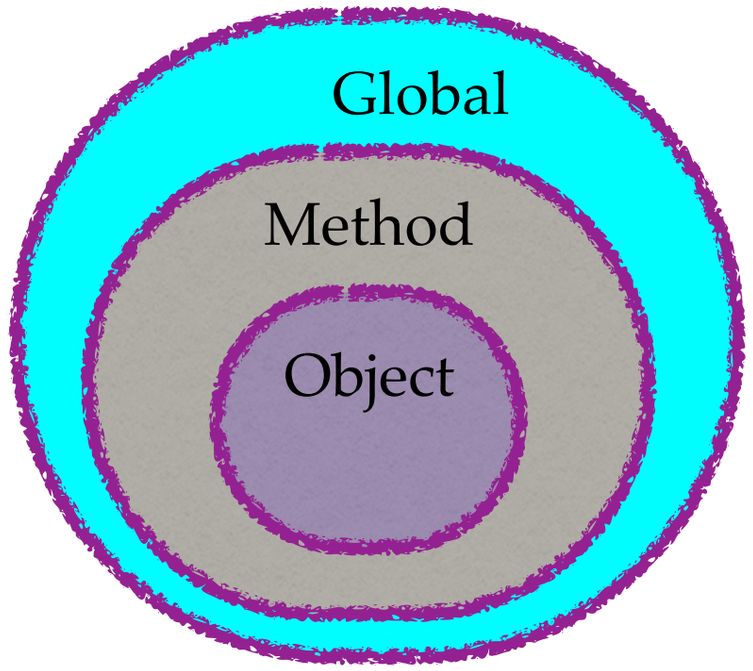
Layout
Read Field
Write Field
Initialize Field
Count of Fields
Create Layout
Customize Layout

Memory

Another abstraction level - Not yet explored

API for Semantics Adaptation





Message
Lookup
Activate

Context
Receiver
Arguments
Return Frame
Stack

Executor
Read Global
Write Global
Read Local
Write Local
Read Arg
Read Literal
Return

Layout
Read Field
Write Field
Initialize Field
Count of Fields
Create Layout
Customize Layout

Memory

Another abstraction level - Not yet explored

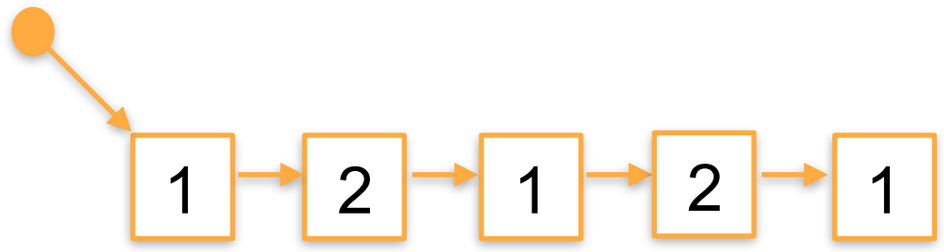
Readonly Protection Scenario

- Readonly Reference


```
def evenToOdd(){
  current = this.first();
  while (current) {
    if (current->isEven()) {
      current->add(1);
    }
    current = current->next;
  }
}
```

Readonly Protection Scenario

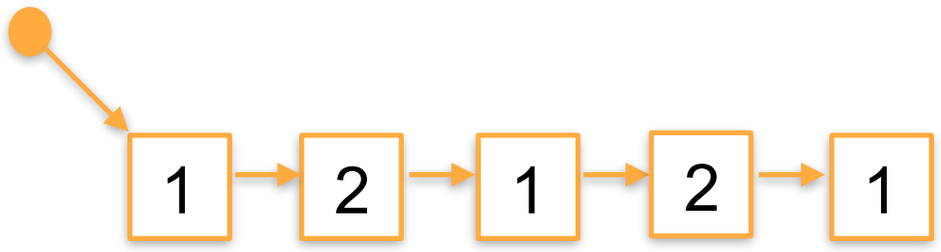
- Readonly Reference




```
def evenToOdd(){
 current = this.first();
  while (current) {
    if (current->isEven()) {
      current->add(1);
    }
    current = current->next;
  }
}
```

Readonly Protection Scenario

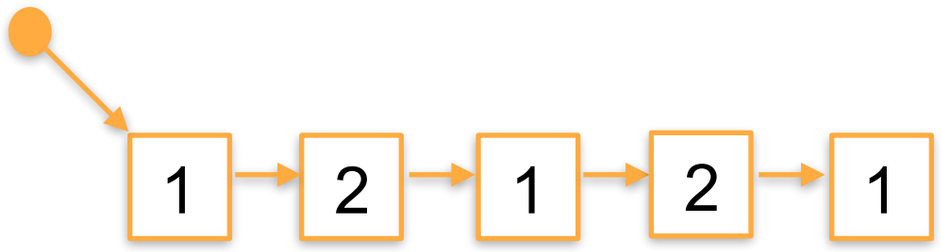
- Readonly Reference




```
def evenToOdd(){
  current = this.first();
  while (current) {
    if (current->isEven()) {
      current->add(1);
    }
    current = current->next;
  }
}
```

Readonly Protection Scenario

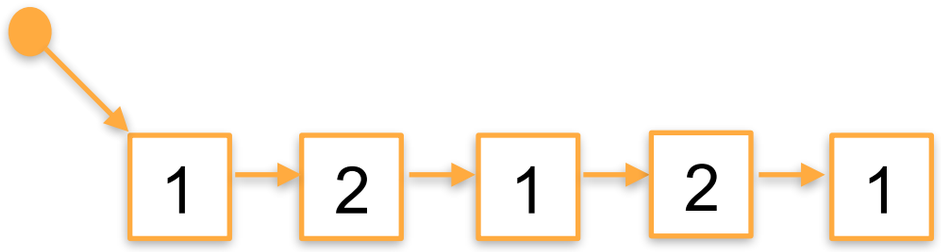
- Readonly Reference




```
def evenToOdd(){
  current = this.first();
  while (current) {
    if (current->isEven()) {
      current->add(1);
    }
    current = current->next;
  }
}
```

Readonly Protection Scenario

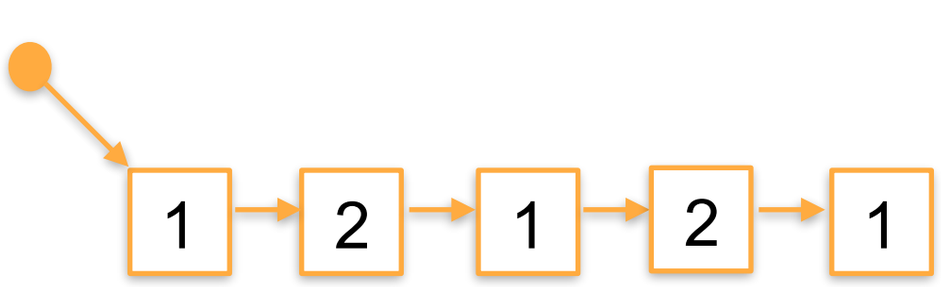
- Readonly Reference




```
def evenToOdd(){
  current = this.first();
  while (current) {
    if (current->isEven()) {
      current->add(1);
    }
    current = current->next;
  }
}
```

Readonly Protection Scenario

- Readonly Reference

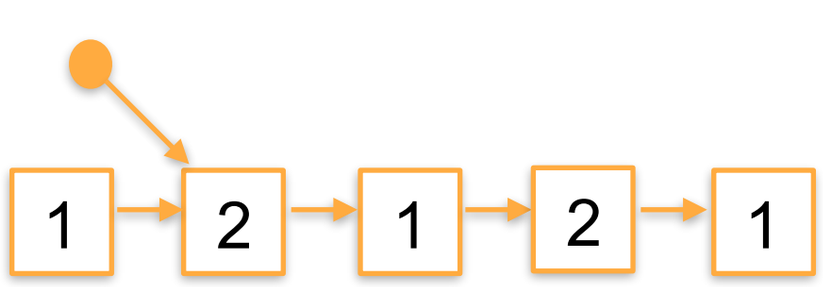



```
def evenToOdd(){
  current = this.first();
  while (current) {
    if (current->isEven()) {
      current->add(1);
    }
    current = current->next;
  }
}
```

Readonly semantics **propagation**

Readonly Protection Scenario

- Readonly Reference

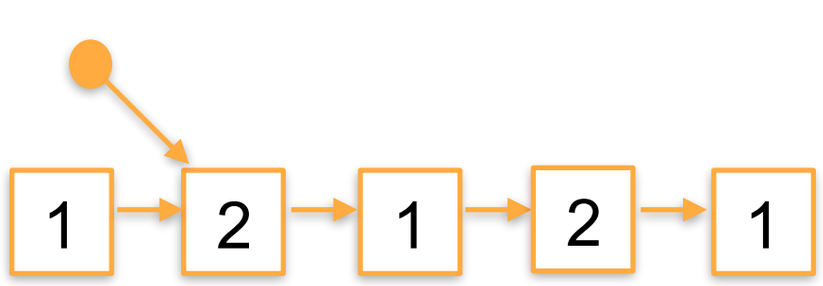



```
def evenToOdd(){
  current = this.first();
  while (current) {
    if (current->isEven()) {
      current->add(1);
    }
    current = current->next;
  }
}
```

Readonly semantics **propagation**

Readonly Protection Scenario

- Readonly Reference

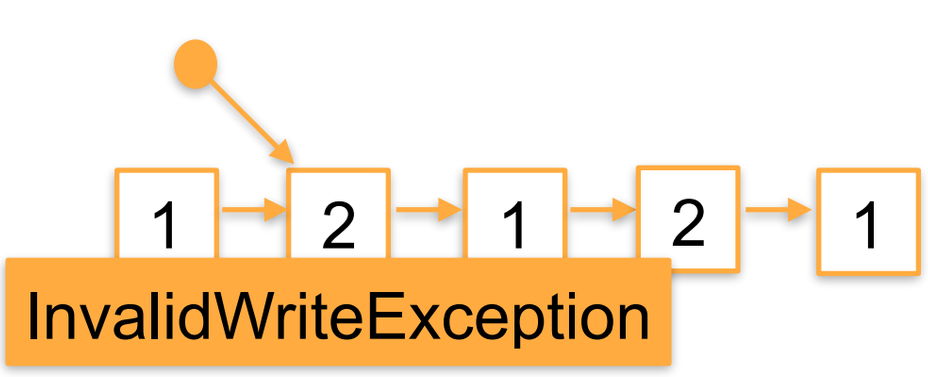



```
def evenToOdd(){
  current = this.first();
  while (current) {
    if (current->isEven()) {
      current->add(1);
    }
    current = current->next;
  }
}
```

Readonly semantics **propagation**

Readonly Protection Scenario

- Readonly Reference




```
def evenToOdd(){
  current = this.first();
  while (current) {
    if (current->isEven()) {
      current->add(1);
    }
    current = current->next;
  }
}
```

Readonly semantics **propagation**

Mutations must *be* **forbidden**

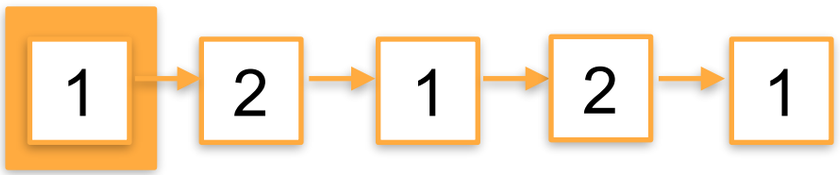
Readonly Protection Scenario

Proxy (Handle)


```
def evenToOdd(){
  current = this.first();
  while (current) {
    if (current->isEven()) {
      current->add(1);
    }
    current = current->next;
  }
}
```

Readonly Protection Scenario

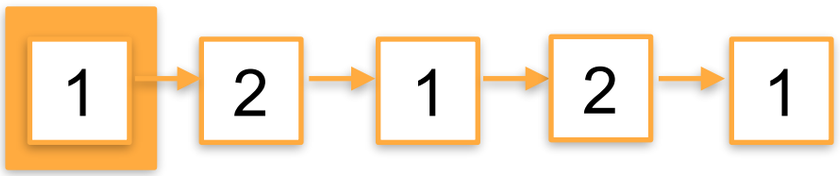
Proxy (Handle)




```
def evenToOdd(){
  current = this.first();
  while (current) {
    if (current->isEven()) {
      current->add(1);
    }
    current = current->next;
  }
}
```

Readonly Protection Scenario

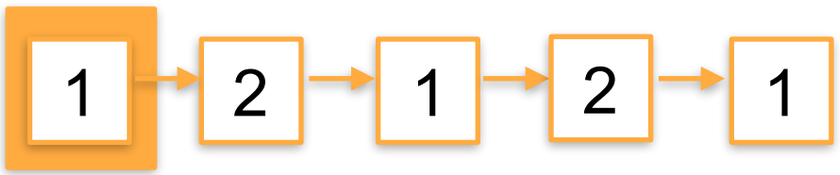
Proxy (Handle)




```
def evenToOdd(){
  current = this.first();
  while (current) {
    if (current->isEven()) {
      current->add(1);
    }
    current = current->next;
  }
}
```

Readonly Protection Scenario

Proxy (Handle)

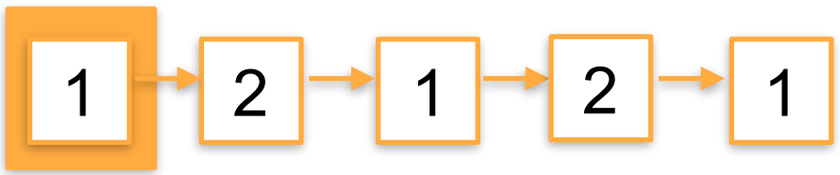



```
def evenToOdd(){
  current = this.first();
  while (current) {
    if (current->isEven()) {
      current->add(1);
    }
    current = current->next;
  }
}
```

Delegate the isEven() method to **list items**

Readonly Protection Scenario

Proxy (Handle)

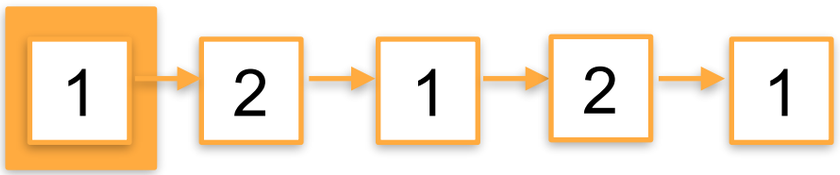



```
def evenToOdd(){
  current = this.first();
  while (current) {
    if (current->isEven()) {
      current->add(1);
    }
    current = current->next;
  }
}
```

Delegate the isEven() method to **list items**

Readonly Protection Scenario

Proxy (Handle)



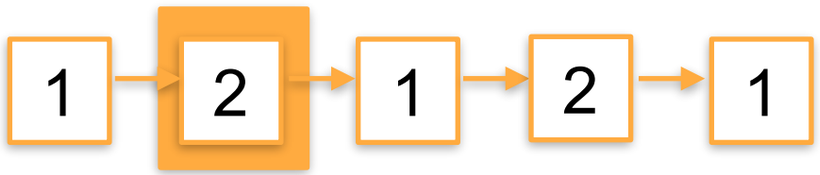

```
def evenToOdd(){
  current = this.first();
  while (current) {
    if (current->isEven()) {
      current->add(1);
    }
    current = current->next;
  }
}
```

Delegate the isEven() method to **list items**

handles semantics **propagation**

Readonly Protection Scenario

Proxy (Handle)



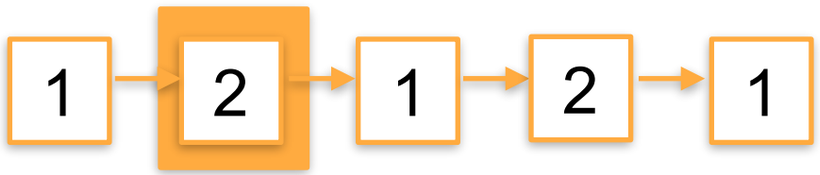

```
def evenToOdd(){
  current = this.first();
  while (current) {
    if (current->isEven()) {
      current->add(1);
    }
    current = current->next;
  }
}
```

Delegate the isEven() method to **list items**

handles semantics **propagation**

Readonly Protection Scenario

Proxy (Handle)



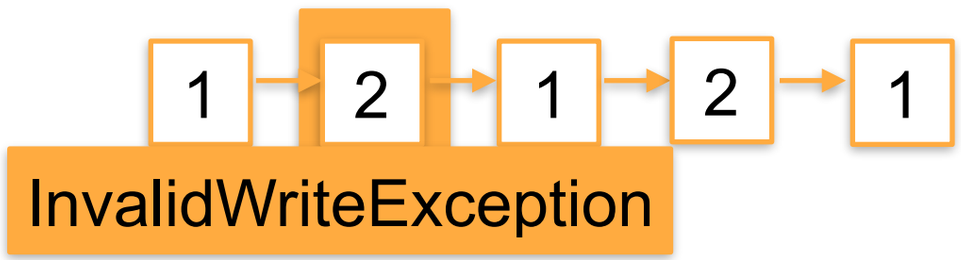

```
def evenToOdd(){
  current = this.first();
  while (current) {
    if (current->isEven()) {
      current->add(1);
    }
    current = current->next;
  }
}
```

Delegate the isEven() method to **list items**

handles semantics **propagation**

Readonly Protection Scenario

Proxy (Handle)




```
def evenToOdd(){
  current = this.first();
  while (current) {
    if (current->isEven()) {
      current->add(1);
    }
    current = current->next;
  }
}
```

Delegate the isEven() method to **list items**

handles semantics **propagation**

handles must **forbid** mutations

Handles Implementation

- **Method Lookup**: the lookup must be performed in the target
- **Read Field:** must return a handle to the field to propagate immutability
- **Write Field:** must be forbidden

Handles Using the API

```
class ReadonlyMsg extends Message (
  def lookup(handle, methodName) {
    return super.lookup(handle.getTarget(), methodName);
  }
)

class ReadonlyExec extends Executor (
  def read(handle, anIndex) {
    return Handle(super.read(handle.getTarget(), anIndex));
  }

  def write(handle, anIndex, aValue) {
    throw new InvalidWriteException();
  }
)
```

Handles Using the API

```
class ReadonlyMsg extends Message (
 def lookup(handle, methodName) {
   return super.lookup(handle.getTarget(), methodName);
  }
)

class ReadonlyExec extends Executor (
  def read(handle, anIndex) {
    return Handle(super.read(handle.getTarget(), anIndex));
   }

  def write(handle, anIndex, aValue) {
    throw new InvalidWriteException();
   }
)
```

Evaluation

Quality-Based Criteria

- **Succinctness:** the number of lines of code (#LOCs) needed for the adaptation.
- **Forward compatibility:** whether an adaptation persists during the evolution of the application.
- **Scoping:** whether the approach provides explicit ways to apply changes at different levels of granularity.
- **Modularity:** indicates whether it is possible to implement the adaptation without polluting the application’s logic.

Results

- Reflective VM handled all the cases while alternatives don't
- Reflective VMs subsume language-level alternatives
- Reflective VM >= language-level alternatives (quality criteria)

Behavioral Types

Can formal reasoning around behavioral types (typestates?) provide (weak/strong) guarantees to advanced reflective models?