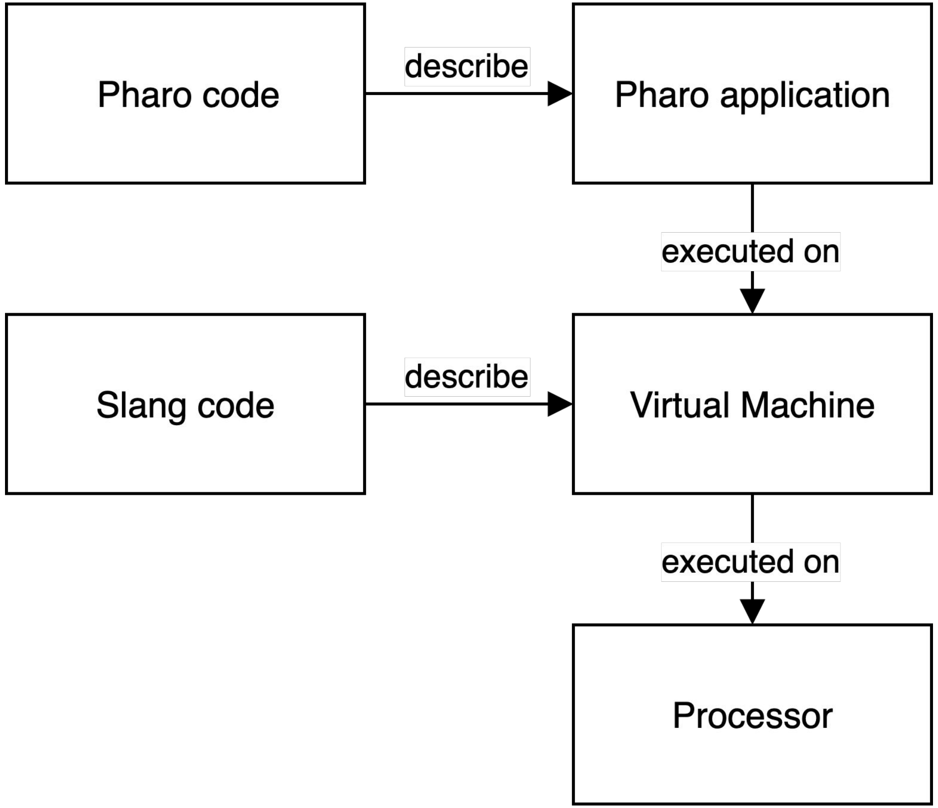
Illicium

Compiling Pharo to C

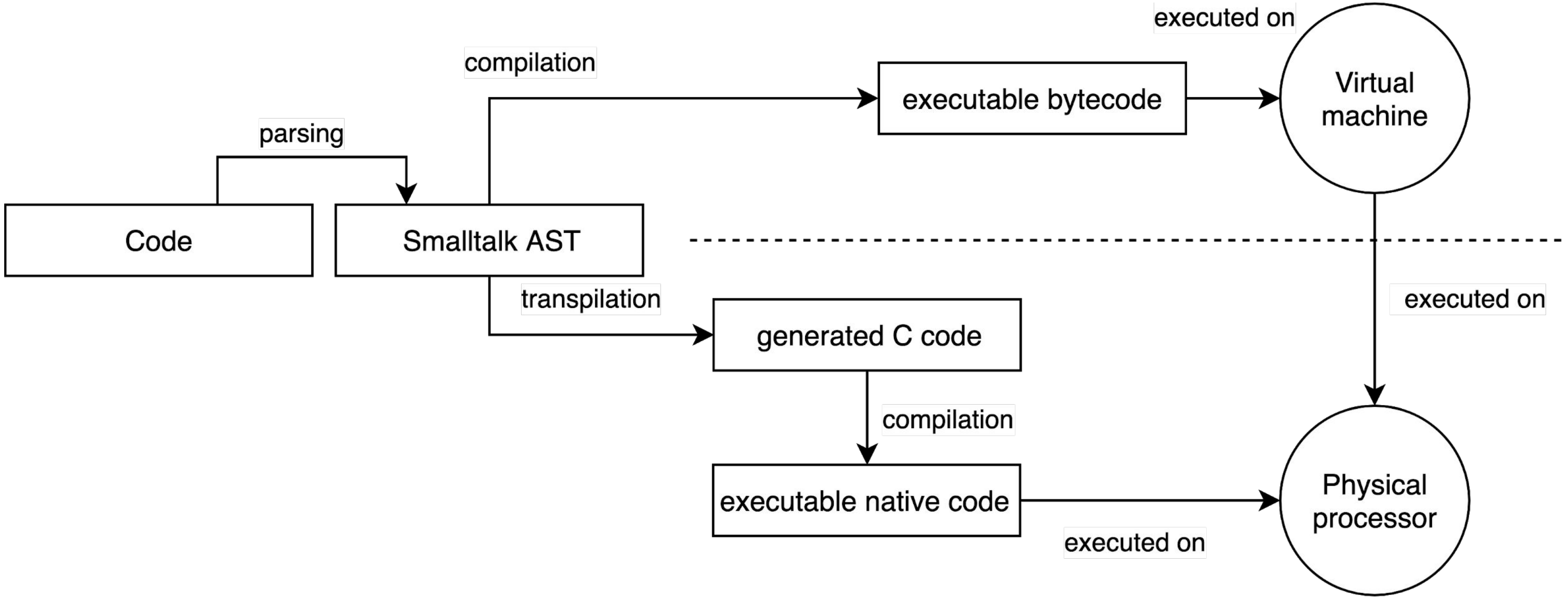
Pierre Misse-Chanabier

Vincent Aranega
Guillermo Polito
Stéphane Ducasse

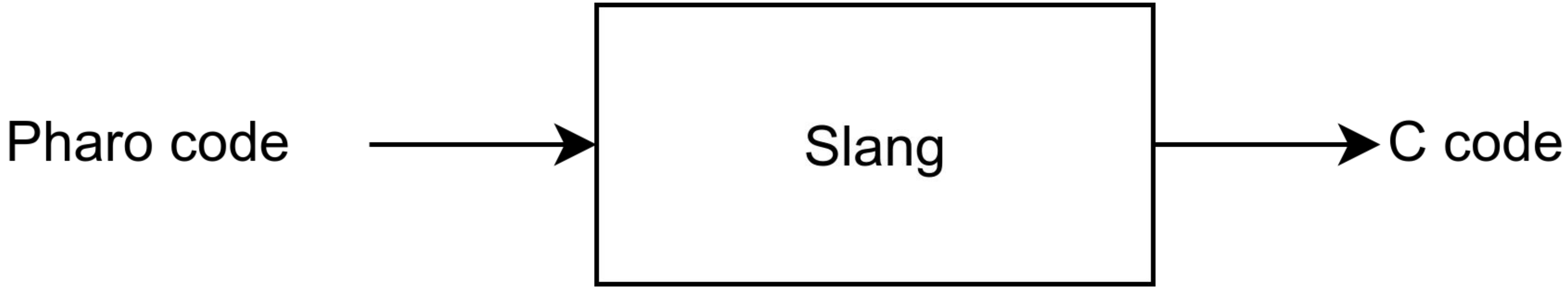
Pharo development



Code compilation



Slang



Slang: Basis

```
anOperator
    ^ 1 + 2
```

```
int anOperator(void)
{
    return 1 + 2;
}
```

Slang: Control flow

```
anIf
    true ifTrue:[ 1 + 2 ]
```

```
int anIf(void)
{
    if(true){
        1 + 2;
    }
    return 0;
}
```

Slang: a Macro

```
aMacro
    ^ 1 between: 2 and: 3
```

```
int aMacro(void)
{
    return ((1>=2) && (1<=3));
}
```

Slang: an Unknown Message

```
anUnknownMessage
    ^ 1 even
```

```
int anUnknowMessage(void)
{
    return even(1);
}
```

Slang: an Unknown Message

```
anUnknownMessage
    ^ 1 class
```

```
int anUnknowMessage(void)
{
    return class(1);
}
```

Slang: a Weird Message

```
aWeirdMessage
    ^ self
        between: 1 and: false
```

```
int aWeirdMessage(void)
{
    return ((self>=1)
        && (self<=0));
}
```

Slang: assign a value to a class variable

```
assignToClassVariable
    aClassVariable := 5
```

```
#define aClassVariable null

void assignToClassVariable(void)
{
    aClassVariable = 5;
}
```

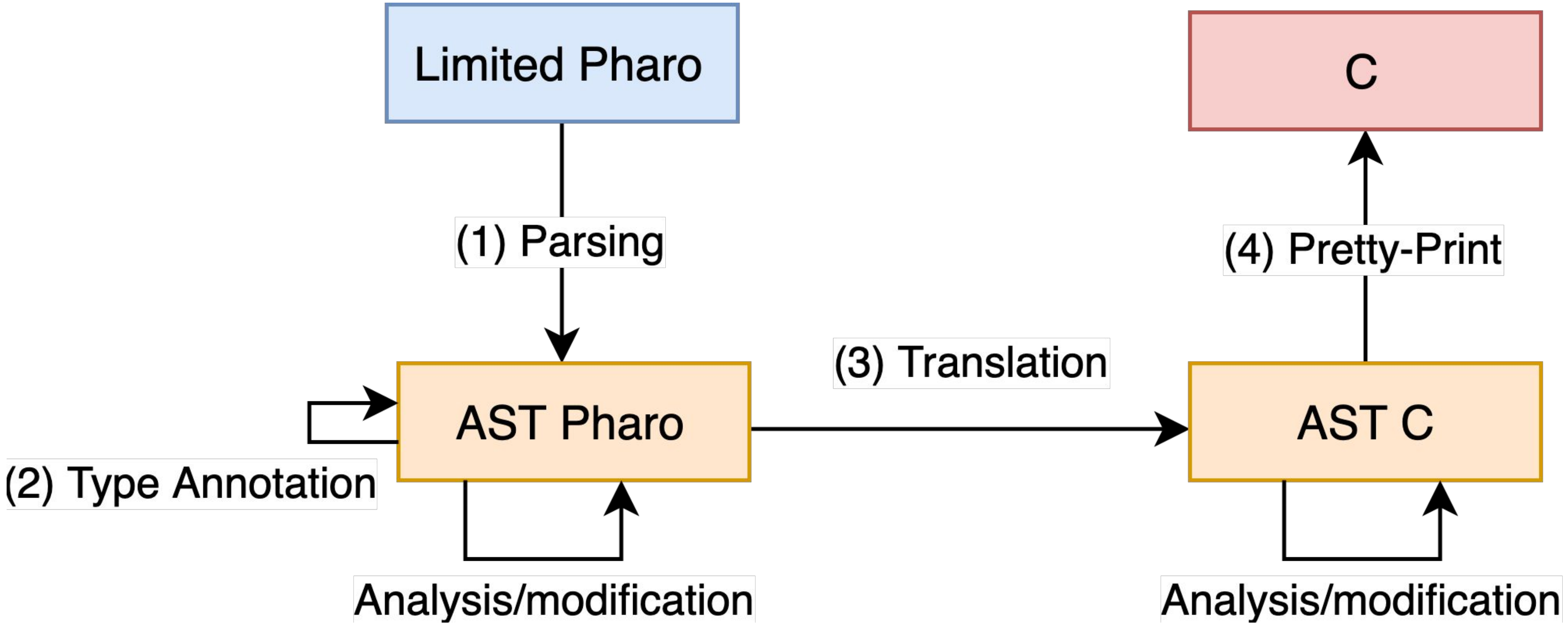
Slang: Code generation

```
generateModulo: msgNode on: aStream indent: level
	"Generate the C code for this message onto the given stream."

	self emitCExpression: msgNode receiver on: aStream.
	aStream nextPutAll: ' % '.
	self emitCExpression: msgNode args first on: aStream
```

Problems

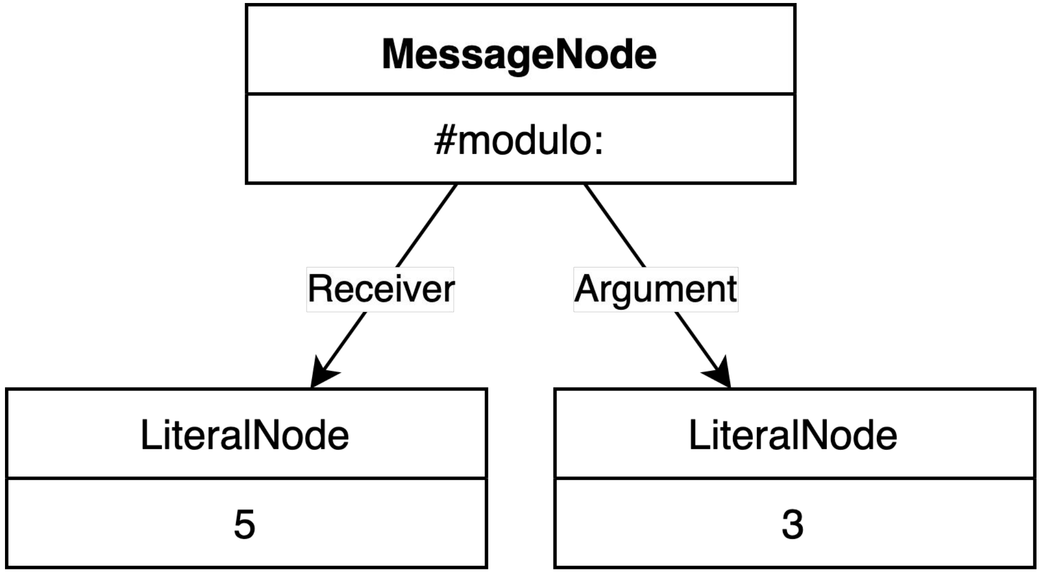
1. No intermediary representation
2. Modularity
3. Blurry language boundaries

My solution: Illicium



What’s an AST? A Visitor?

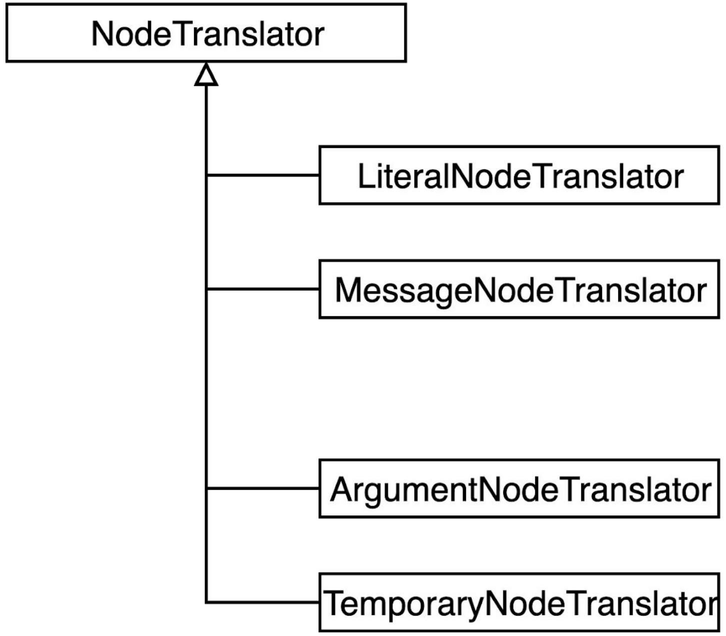
5 modulo: 3

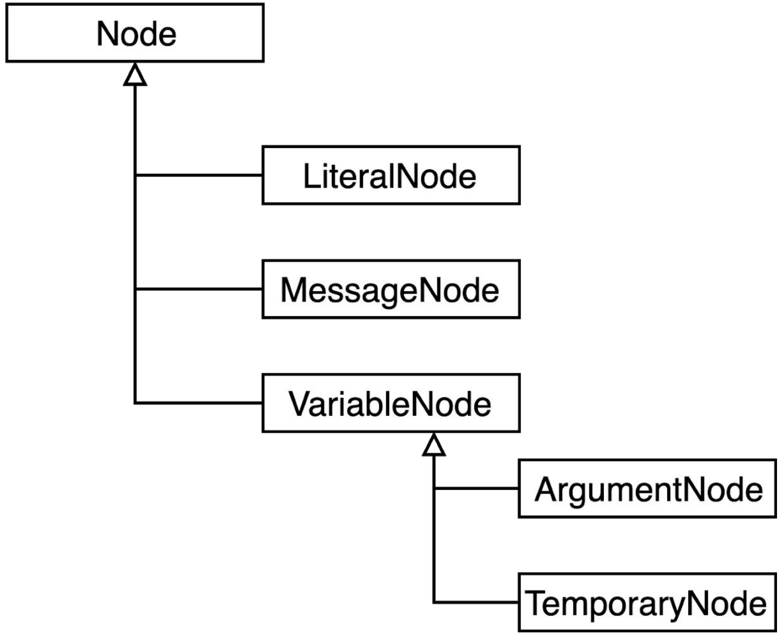


Intermediary representation: AST C

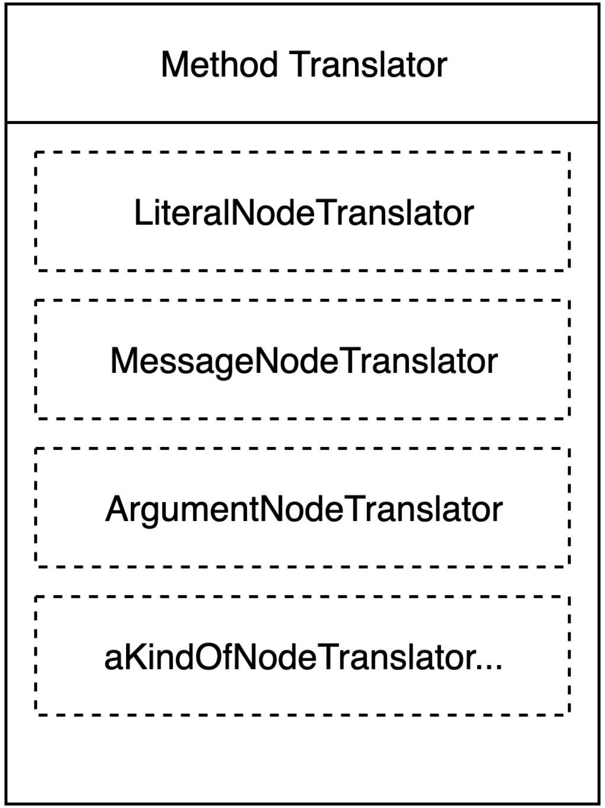
- Described by a meta model
- Code generation
  - Class, attributes, accessors [...]
  - Visitors
  - Consistency

Modularity: Node Translators





Modularity: Method Translator, a composition



Modularity: Method Translator visit

```
MethodTranslator >> visitLiteralValueNode: aLiteralValueNode
      ^ (translators at: #literalValueNodeTranslator)
            translateNode: aLiteralValueNode
            withMethodTranslator: self
```

Modularity: LiteralNodeTranslator

```
LiteralNodeTranslator >>
 TranslateNode: aLiteralNode withMethodTranslator: aTranslator
     ^ ASTCLiteral new
          value: aLiteralNode value
```

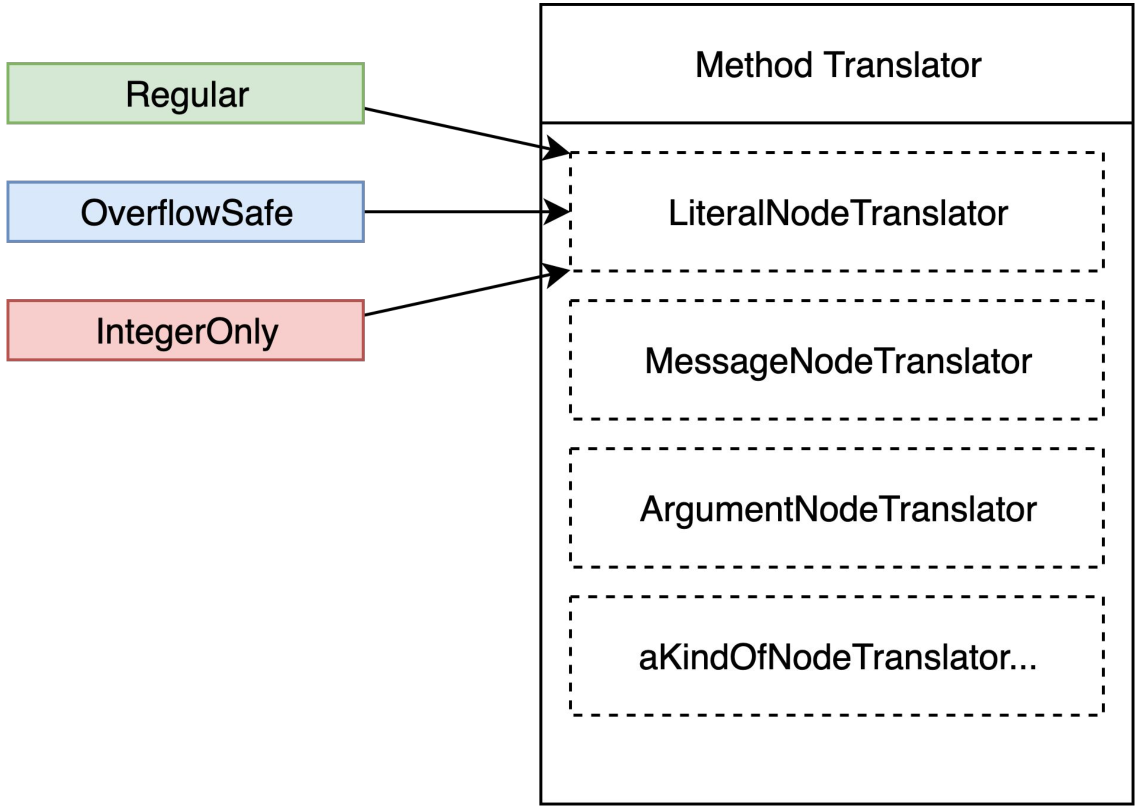
Modularity: OverflowSafeLiteralNodeTranslator

```
OverflowSafeLiteralNodeTranslator >> TranslateNode: aLiteralNode
    aLiteralNode value > 255
    ifTrue:[ self error: ‘not going to fit in a byte’ ].

    ^ ASTCLiteral new
        value: aLiteralNode value
```

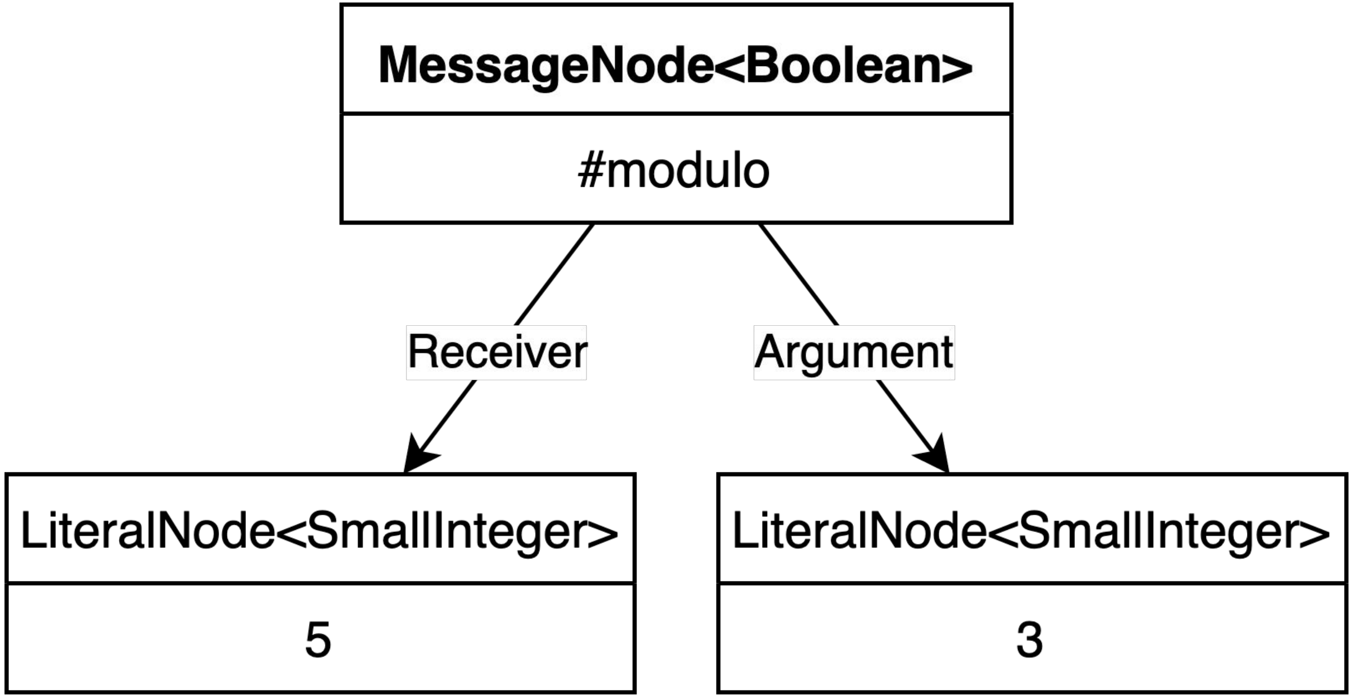
Modularity: IntegerOnlyLiteralNodeTranslator

```
IntegerOnlyLiteralNodeTranslator >>
  TranslateNode: aLiteralNode withMethodTranslator: aTranslator
     aLiteralNode isInteger
     ifFalse:[ self error: ‘Integers are the only real literals!’ ].

     ^ ASTCLiteral new
           value: aLiteralNode value
```

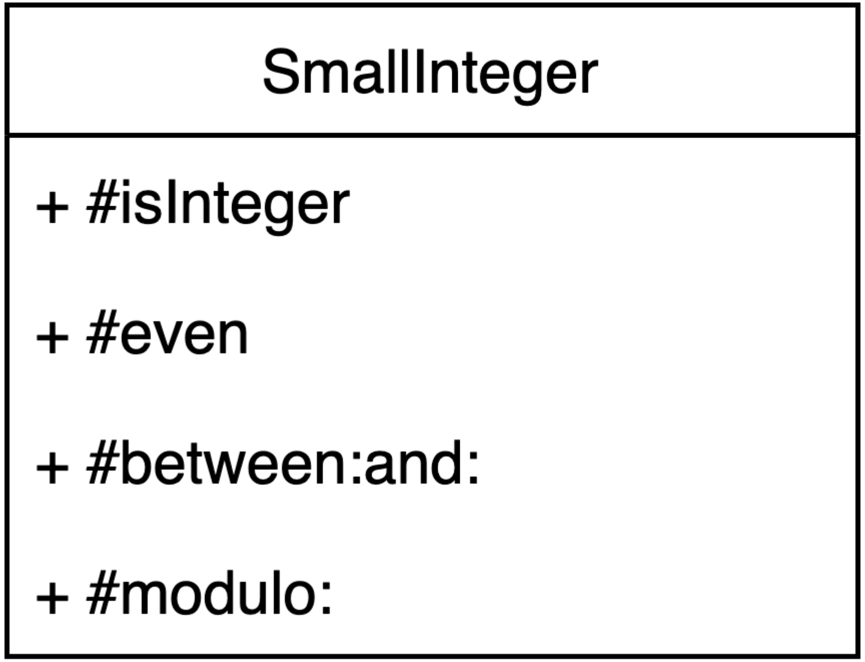
Modularity: Configurable Method Translator

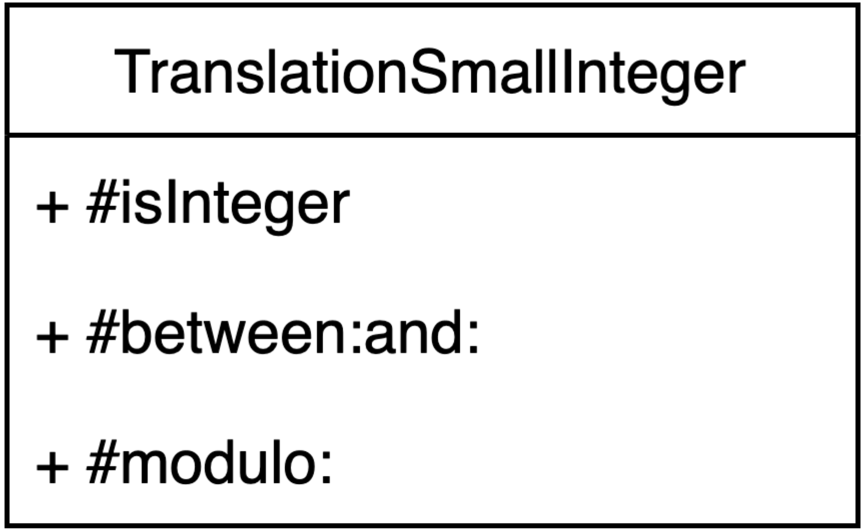


Boundaries: Translation of a MessageNode



Boundaries: Translation Classes





Boundaries: MessageNodeTranslator

```
MessageNodeTranslator >>
   translateNode: aMessageNode withMethodTranslator: aTranslator
       | newReceiver |
       newReceiver := TranslationSmallInteger new
              value: aMessageNode receiver;
              methodTranslator: aTranslator.

       ^ newReceiver perform: aMessageNode selector
              withArguments: aMessageNode arguments
```

Boundaries: Regular vs Translation classes

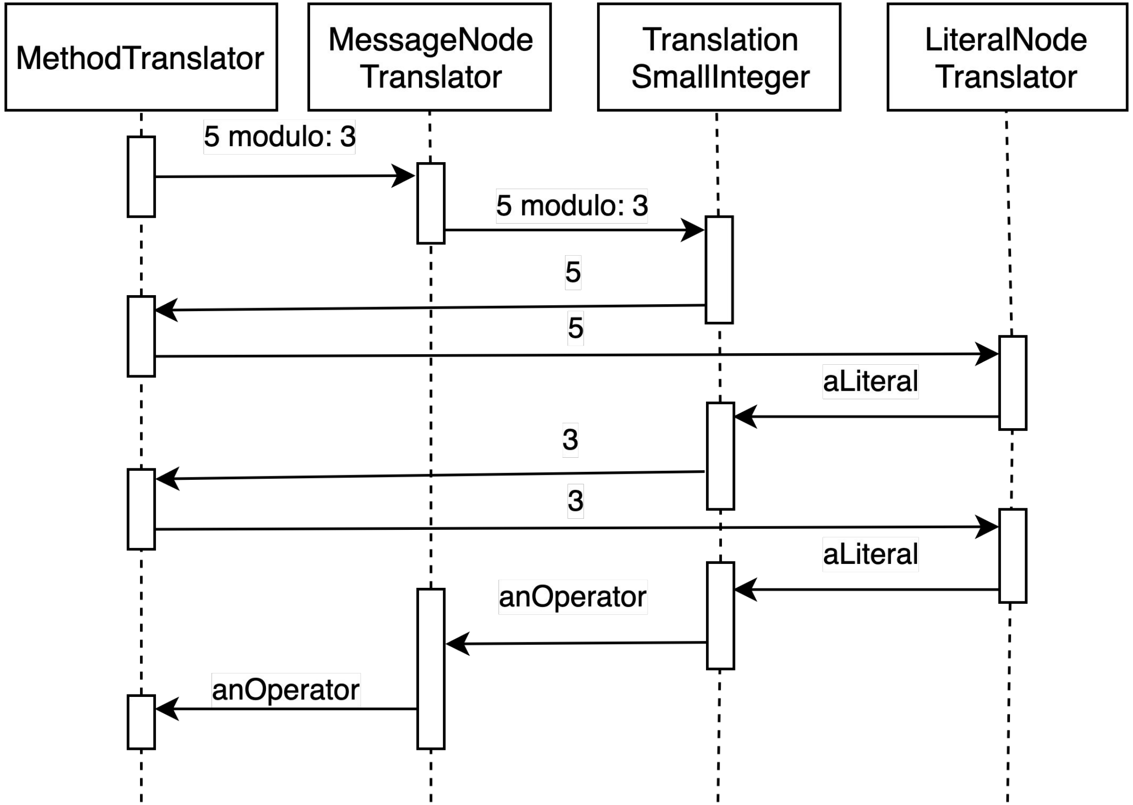
SmallInteger

```
modulo: aNumber
   ^ self - (self // aNumber * aNumber)
```

TranslationSmallInteger

```
modulo: aNumber
   ^ ASTCModuloOperator new
      leftOperand: (self value acceptVisitor: visitor);
      rightOperand: (aNumber acceptVisitor: visitor);
      yourself.
```

Boundaries: Translation process



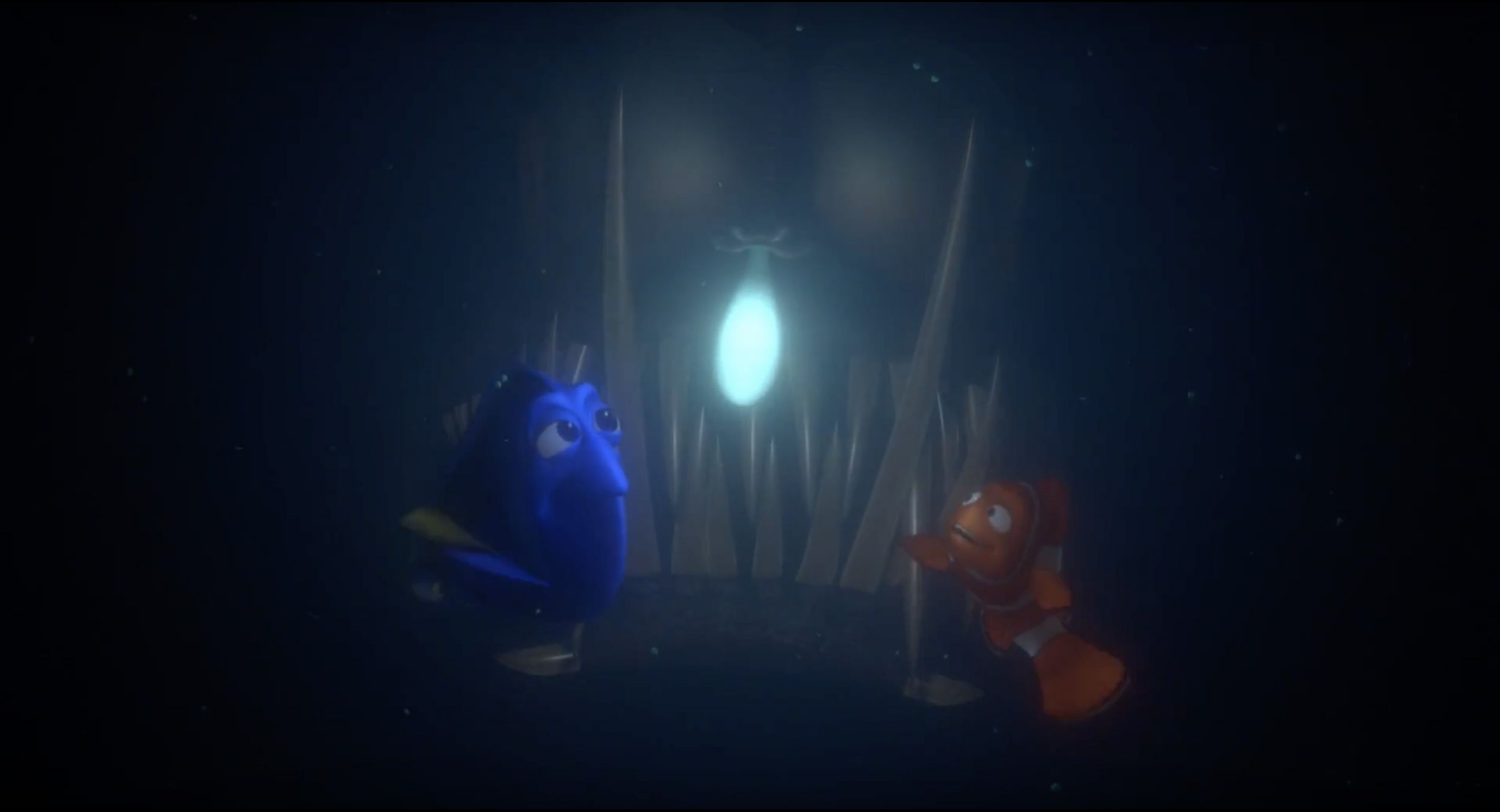
Solution

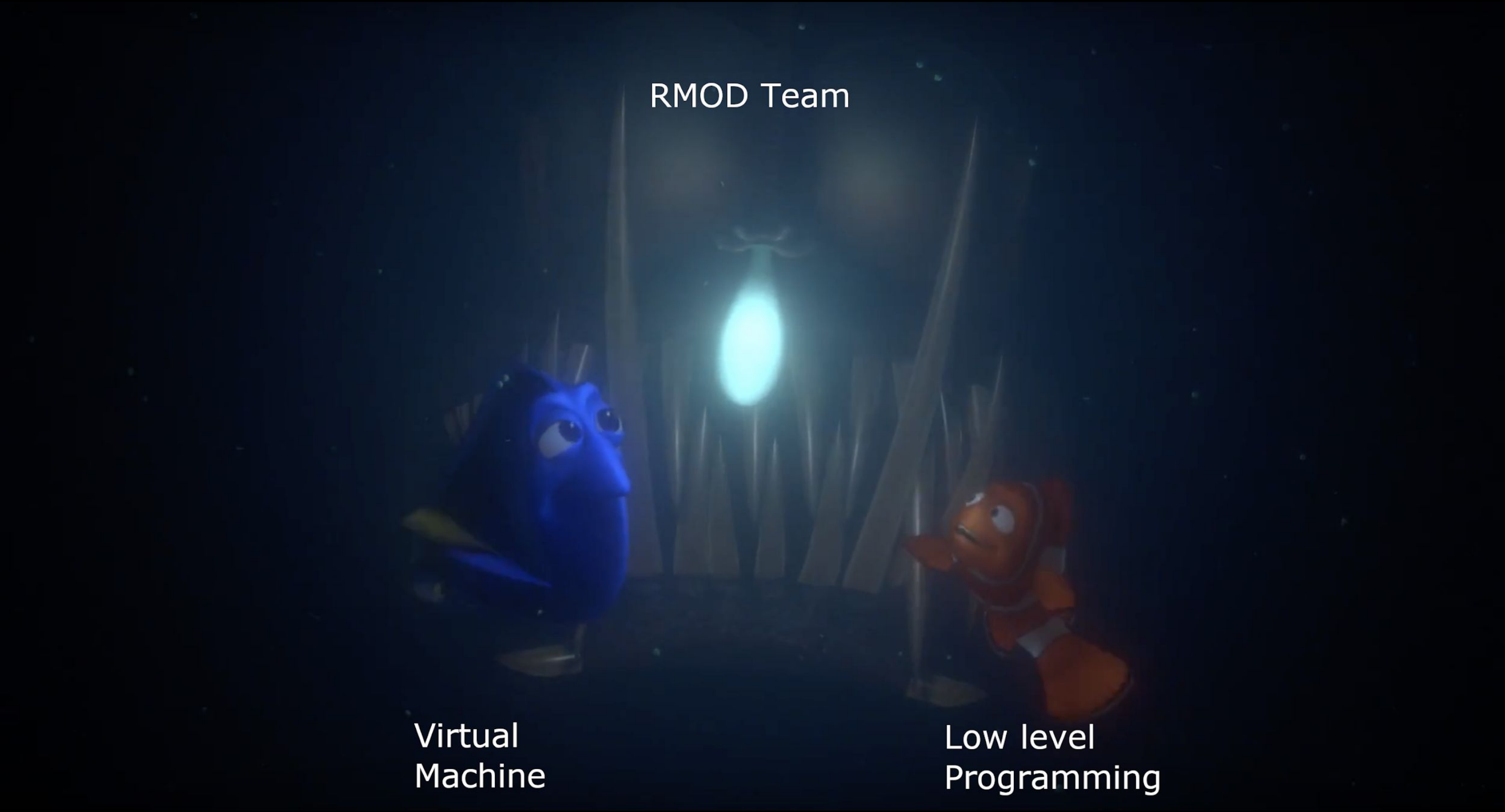
- Better language delimitation
  - Type dependent
  - Browsable
  - Extensible
- Two modularity point
  - Node specialized translator
  - Translation classes

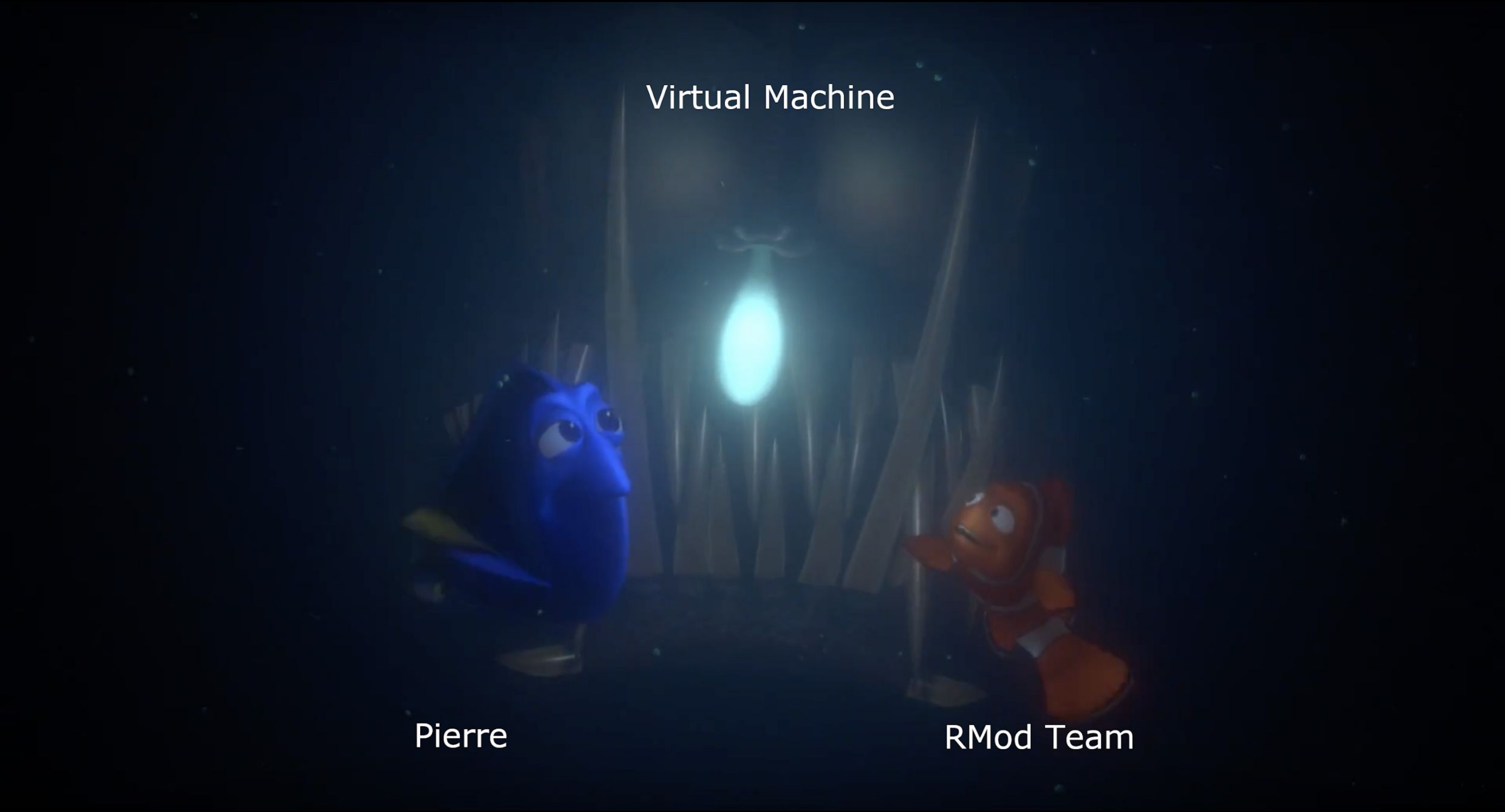
Conclusion

- Slang
- (IR) Metamodel approach
- (Modularity) Small, replaceable translators
- (Boundaries) Message translation based on type
- (Modularity + Boundaries) Translation classes

Pierre Misse-Chanabier
pierre.misse-chanabier@inria.fr
Hogoww on discord/Github




RMOD Team
Virtual Machine
Low level Programming


Virtual Machine
Pierre
RMod Team