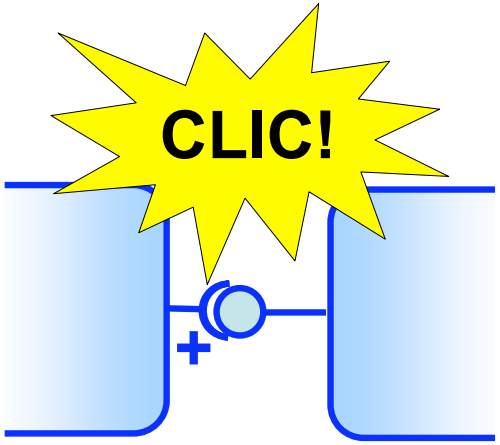


CLIC

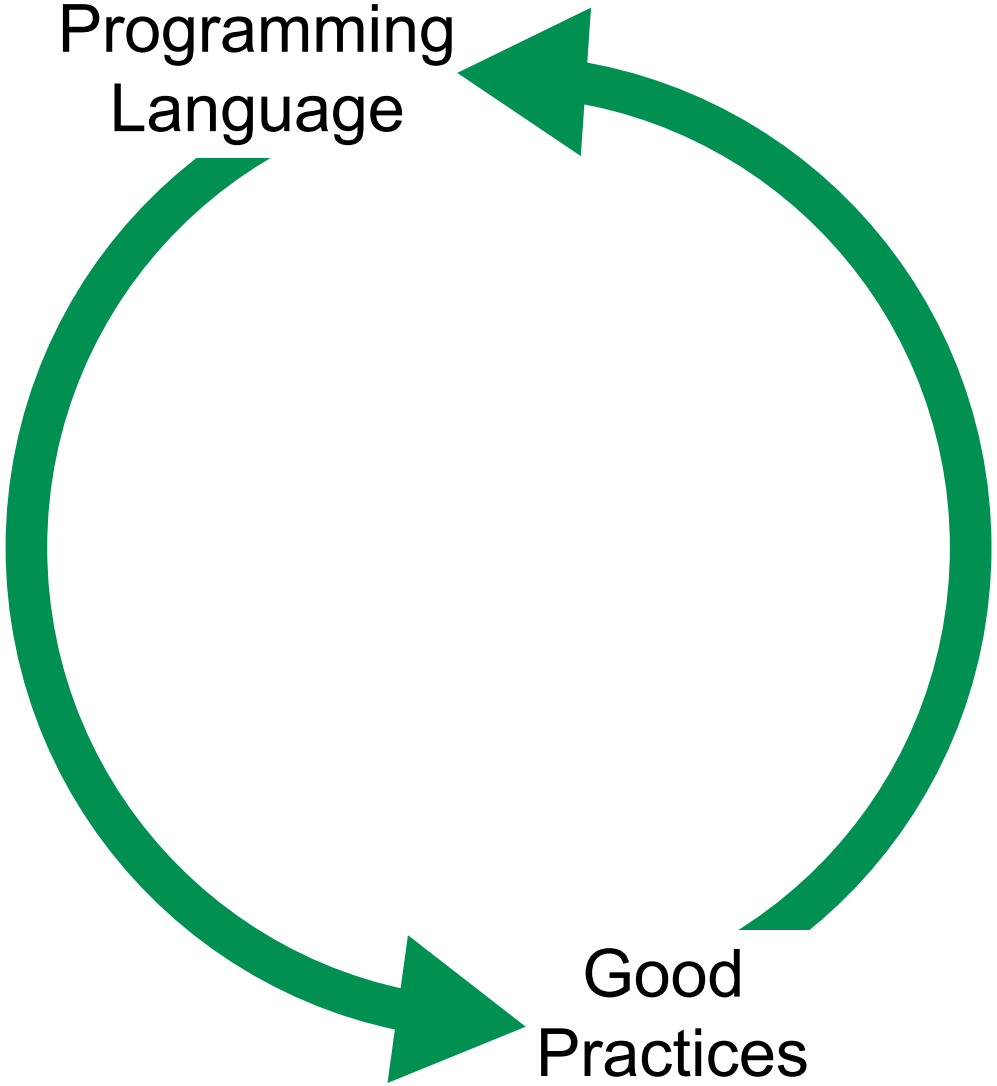
a Component Model Symbiotic with Smalltalk

N. Bouraqadi and L. Fabresse

Ecole des Mines de Douai

http://vst.ensm-douai.fr/Clic

Software Engineering Virtuous Circle



Some Good Practices

- Document the code
  - Design drawings
  - Comments
- Uncoupling software "parts"
  - Inversion of control
  - Use of design patterns such as Observer

Some Issues with OOP

- Out of sync documentation and code
  - Code change not reflected in documentation
- Implicit dependencies
  - Dependencies hidden inside methods

Components

- A step towards enforcing:
  - Documentation
  - Loose Coupling

Programming with components

1. Describe the architecture
   - Part of the software
2. Get the appropriate components
   - Implemented or picked out of some library
3. Assemble the components
   - According to the architecture

A Component is…

a piece of software

- *Standalone or not*

self-documented

- *Requirements / Dependencies*
- *Provided services*
- *Parameters*
- *Architecture*

easy to deploy

- Explicit connections to the rest of the software

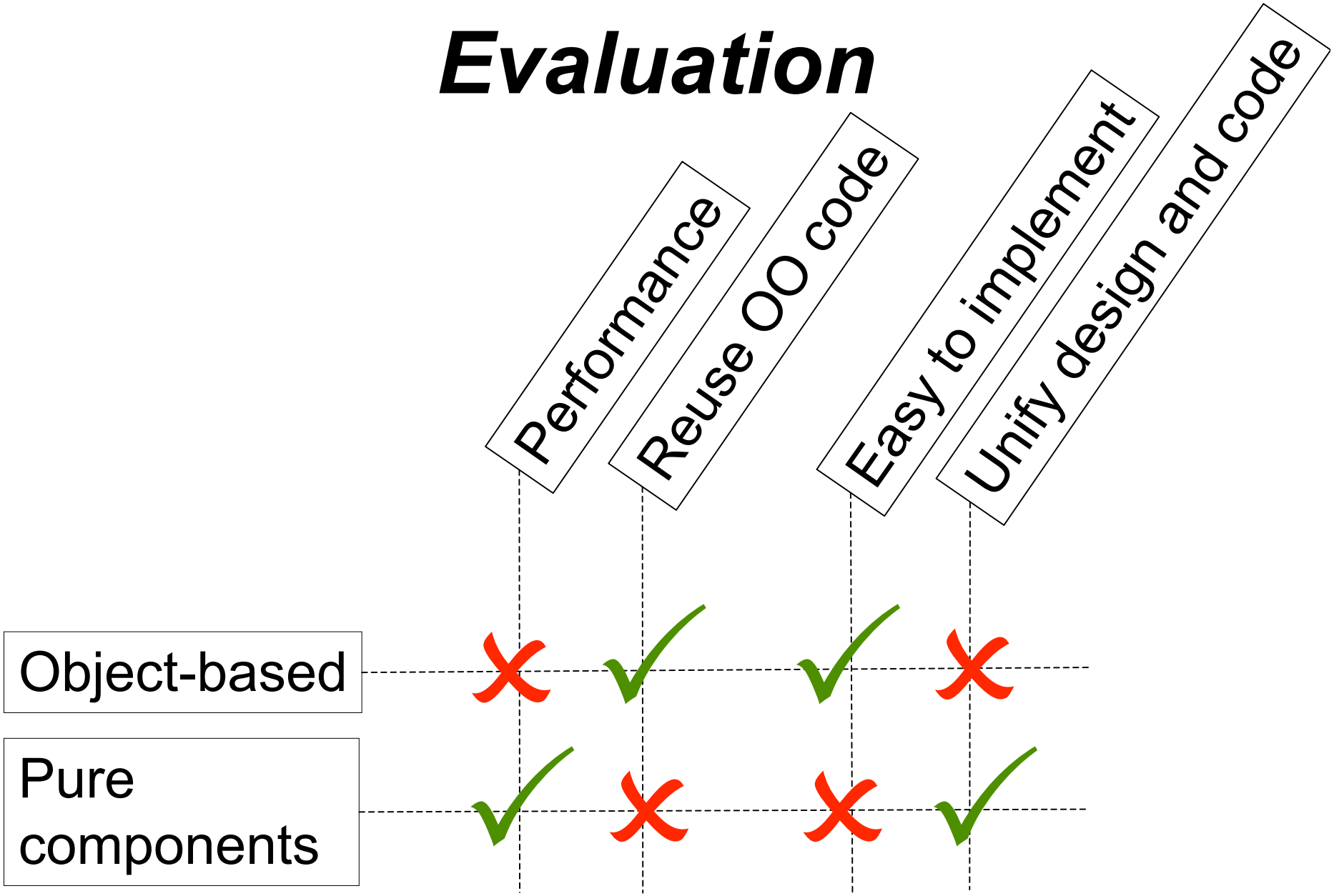
Introducing Components into a Programming Language

- "Objects-based" components
  - Objects = building blocks for components
    - Reification of component related concepts
    - Ex: FracTalk, MalevaST

Introducing Components into a Programming Language

- Pure component-based programming language
  - No objects
  - Ex: SCL

Evaluation


Performance
Reuse OO code
Easy to implement
Unify design and code
Object-based
Pure components

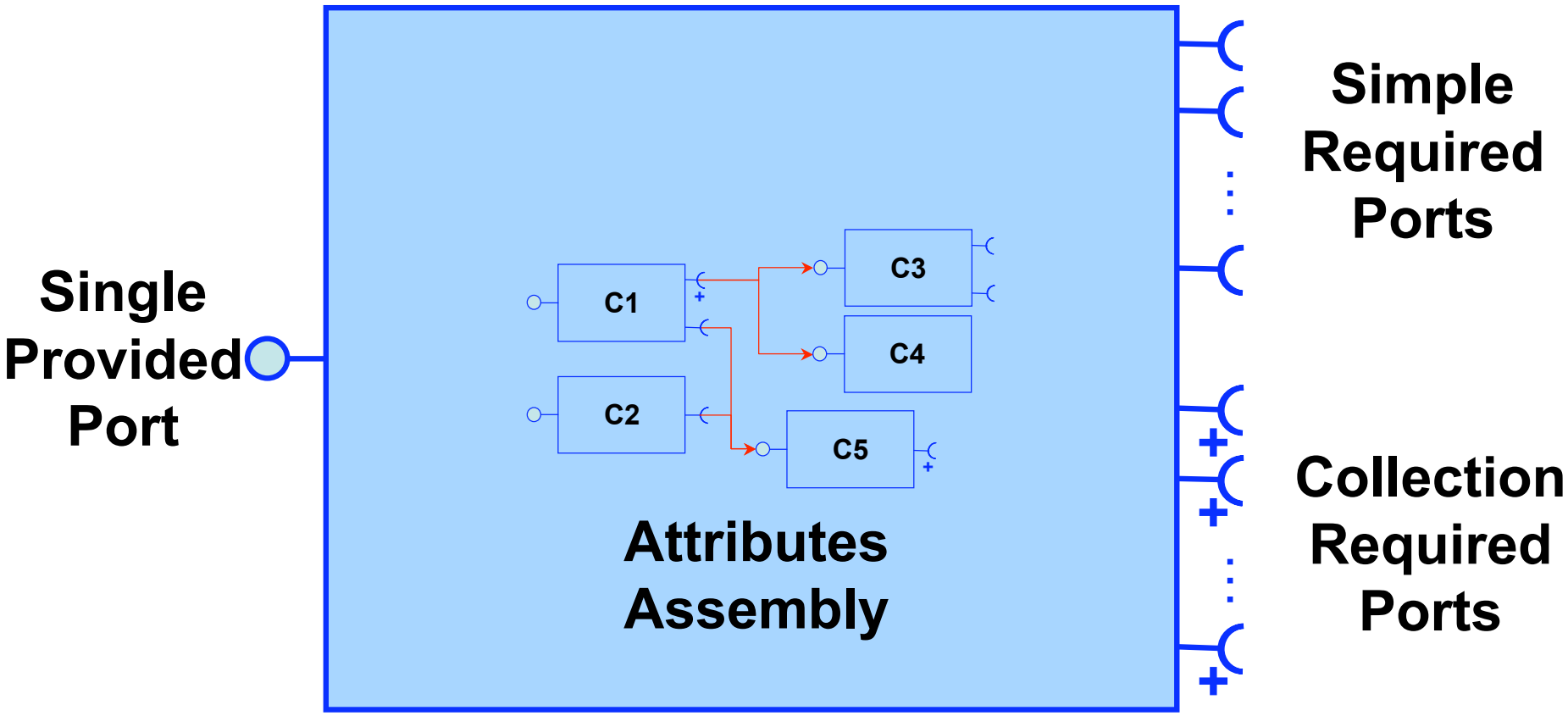
Proposal

Performance
Reuse OO code
Easy to implement
Unify design and code
CLIC

Object-Components Symbiosis

- CLIC Components ARE objects
  - Can be used by objects
- Smalltalk objects ARE *"dirty"* components
  - Can be connected to components
  - Possibly with implicit / tight dependencies
- Same run-time (VM, image)
  - Same performances

CLIC Component

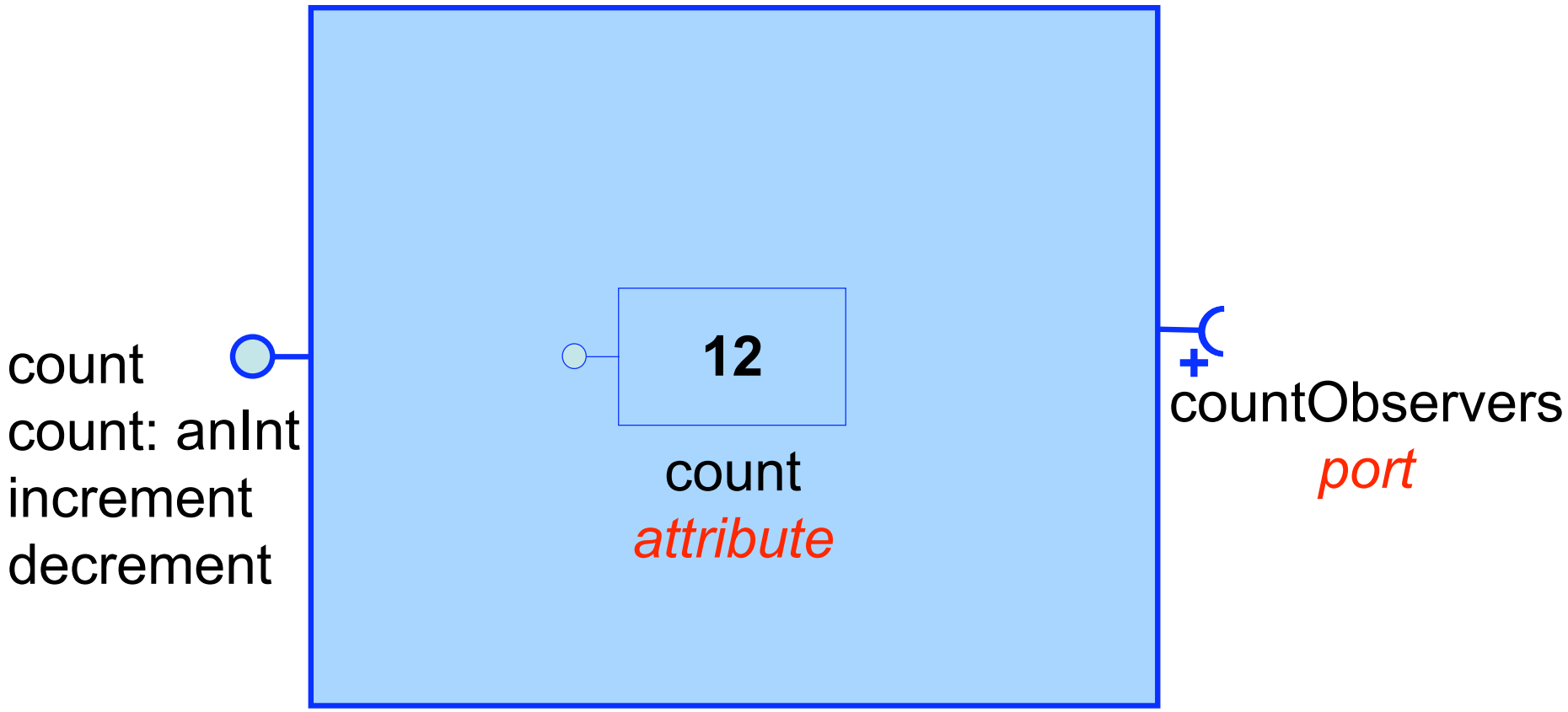


N. Bouraqadi & L. Fabresse - IWST ESUG 2009

Attributes

- Private or shared
- Handled through accessors only
- Observables
  - 1 attribute => 1 collection required port

Counter in CLIC



Counter in CLIC

```
CLComponent subclass: #Counter
    localPrivateAttributeNames: #(count)
    privateAttributesInitDict: {
        #count -> 0}
    sharedAttributeNames: #()
    sharedAttributesInitDict: {}
    localRequiredPortsDict: {}
    category: #'ClicExamples-Clock'
```

Counter in CLIC

```
CLComponent subclass: #Counter
    localPrivateAttributeNames: #(count)
    privateAttributesInitDict: {
        #count -> 0}
    sharedAttributeNames: #()
    sharedAttributesInitDict: {}
    localRequiredPortsDict: {}
    category: #'ClicExamples-Clock'
```

Counter in CLIC

```
CLComponent subclass: #Counter
    localPrivateAttributeNames: #(count)
    privateAttributesInitDict: {
        #count -> 0}
    sharedAttributeNames: #()
    sharedAttributesInitDict: {}
    localRequiredPortsDict: {}
    category: #'ClicExamples-Clock'
```

Counter in CLIC

```
CLComponent subclass: #Counter
    localPrivateAttributeNames: #(count)
    privateAttributesInitDict: {
        #count -> 0}
    sharedAttributeNames: #()
    sharedAttributesInitDict: {}
    localRequiredPortsDict: {}
    category: #'ClicExamples-Clock'
```

Counter in CLIC

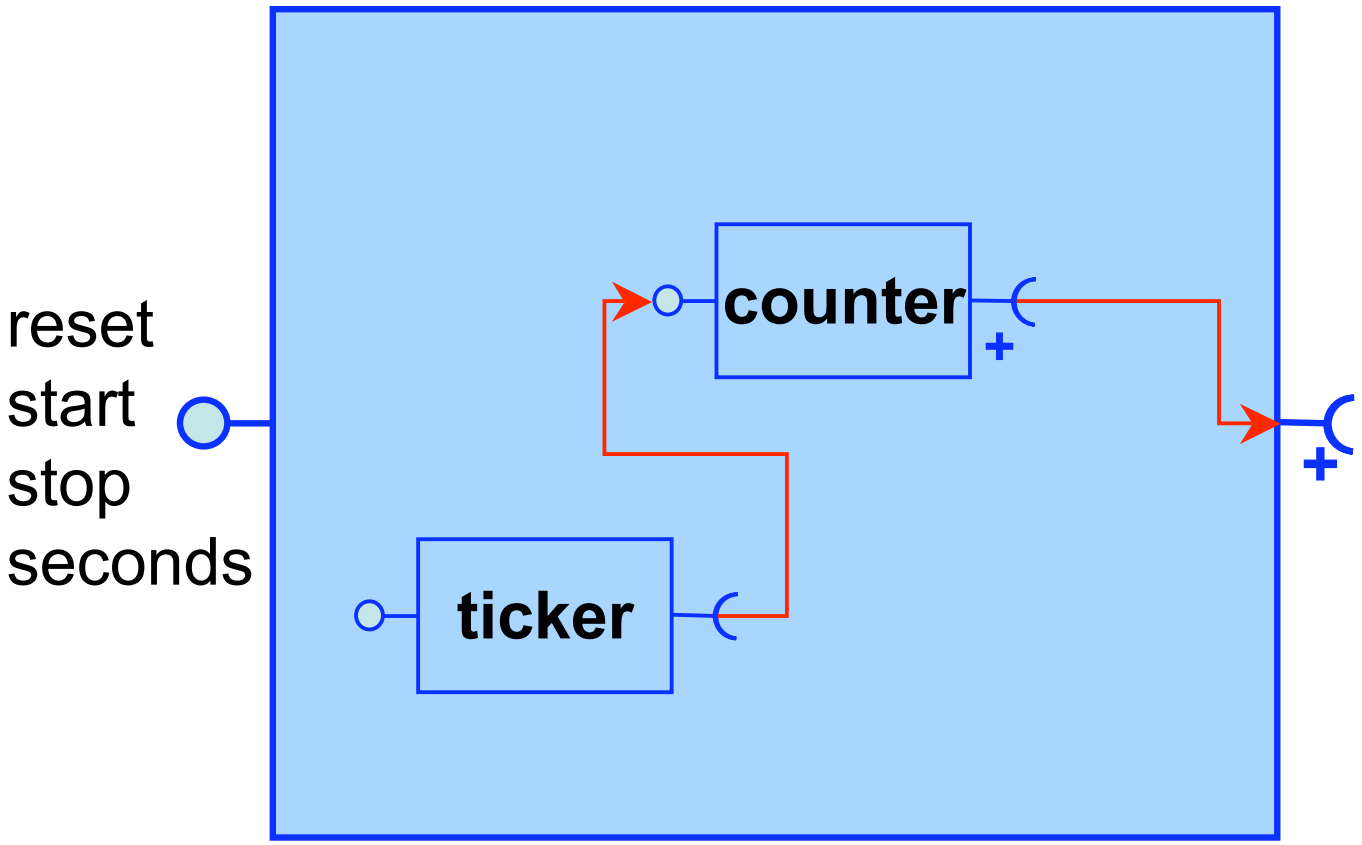
```
Counter >> increment
    self count: self count + 1

Counter >> decrement
    self count: self count - 1
```

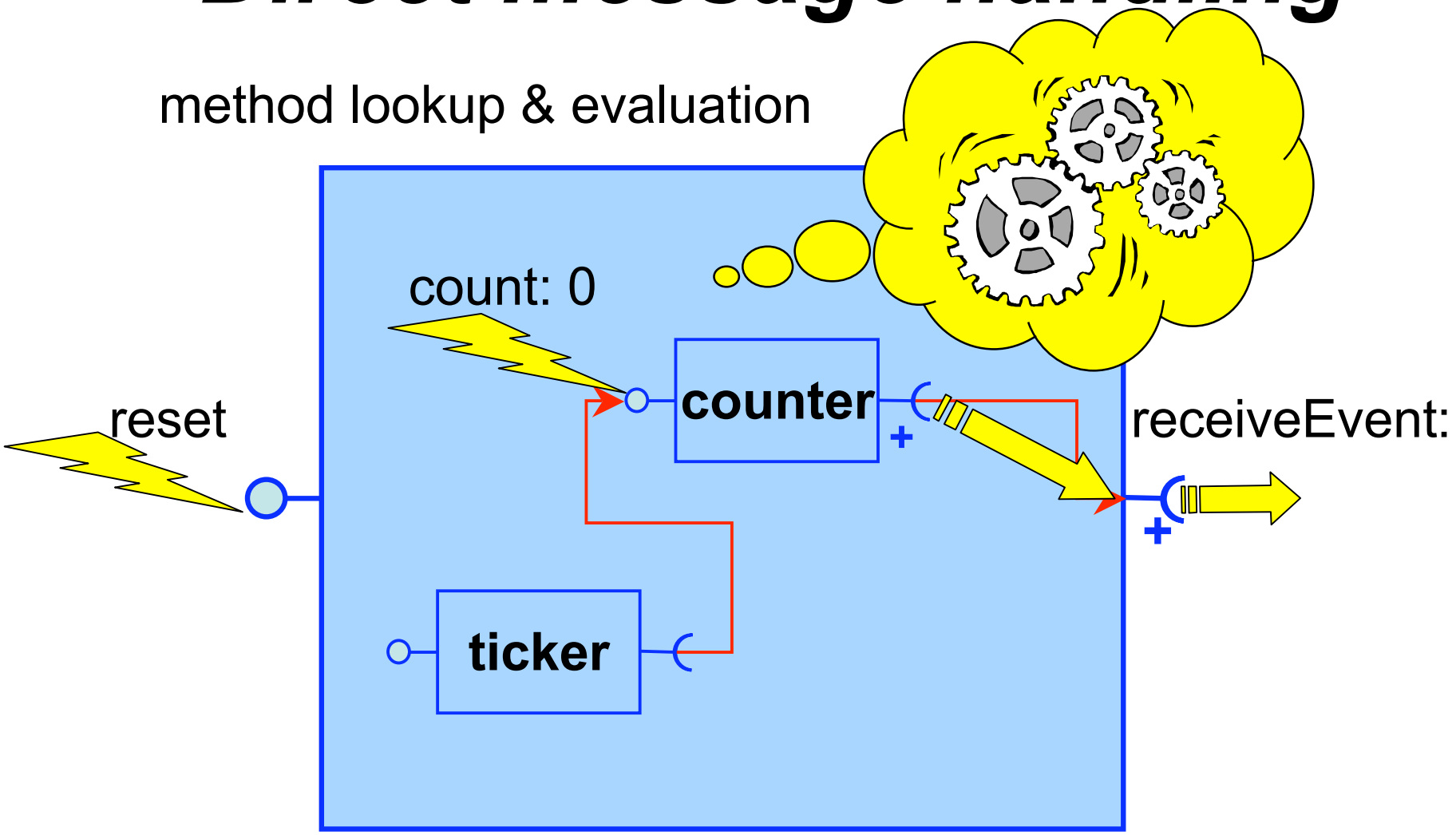
Counter in CLIC

```
|myCounter|
myCounter := Counter new.
myCounter increment.
Transcript cr; show: myCounter count
```

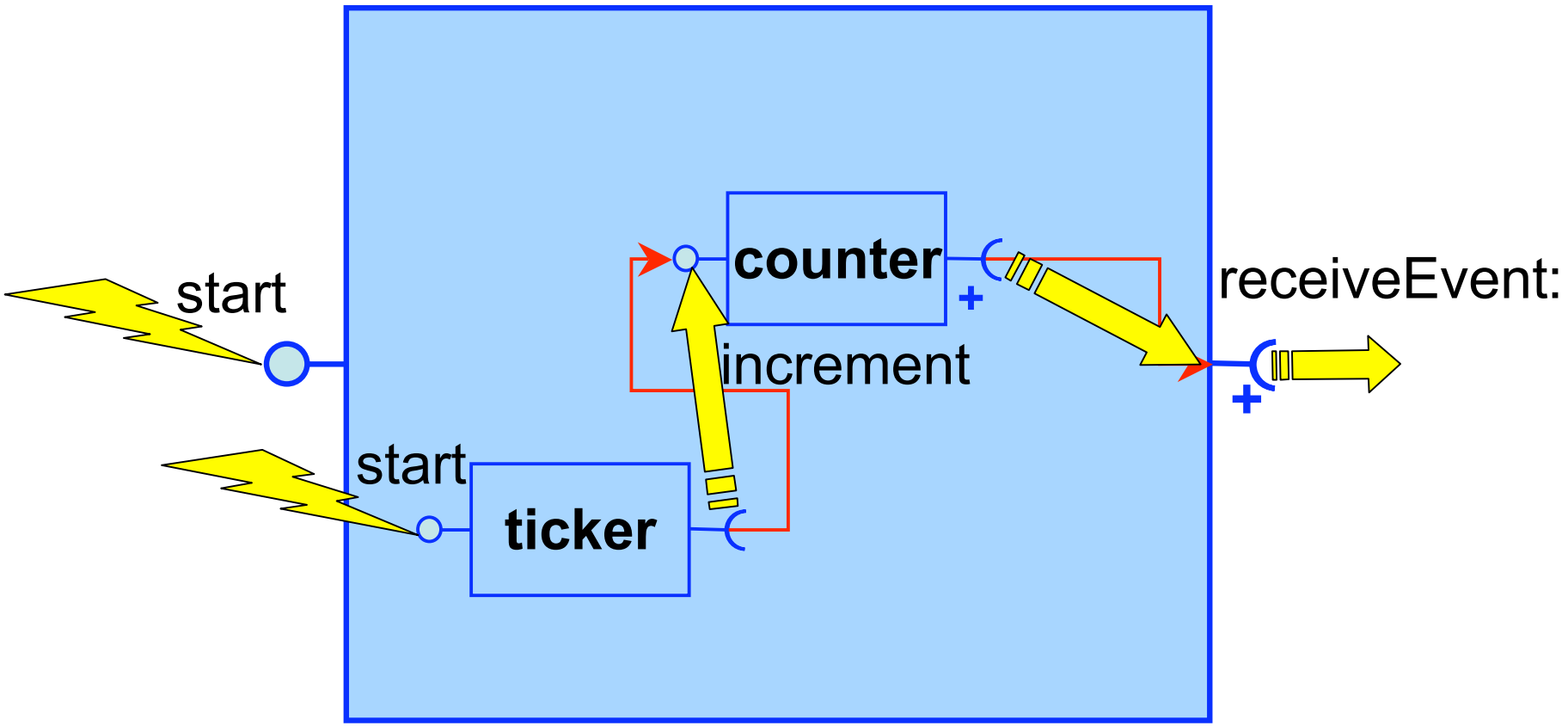
StopWatch in CLIC



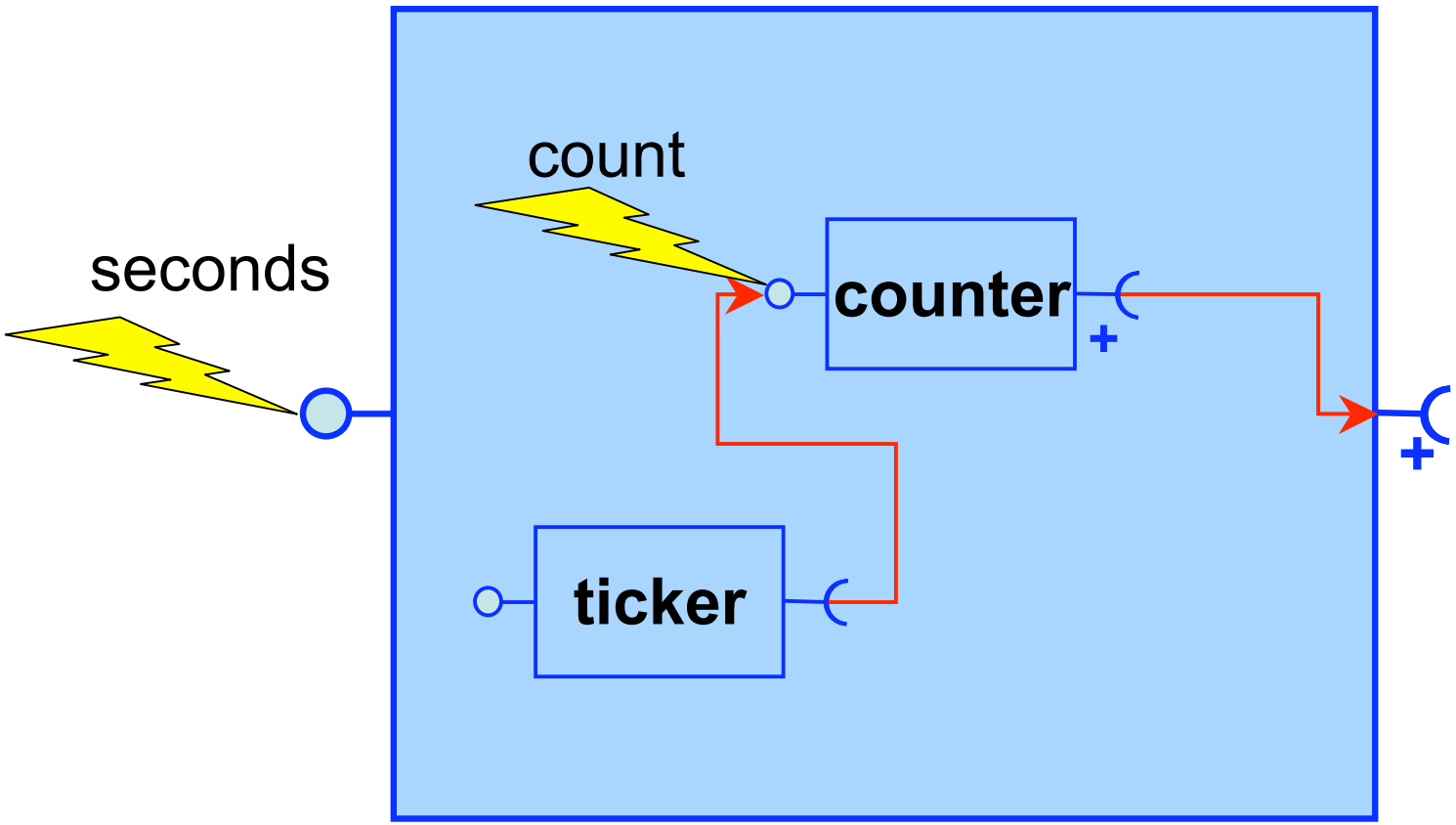
Direct message handling



Message forwarding



Message translation



StopWatch in CLIC

```
CLComponent subclass: #StopWatch
    localPrivateAttributeNames: #(counter ticker)
    privateAttributesInitDict: {
        #counter -> Counter @ #new.
        #ticker -> GenericTicker @ #new}
...
```

StopWatch in CLIC

```
...
sharedAttributeNames: #(Scheduler)
sharedAttributesInitDict: {
   #Scheduler -> Processor}
...
```

StopWatch in CLIC

```
…
architectureFrom: {
   (#ticker @ #notifiedComponents)
                                  => #counter};
…
```

StopWatch in CLIC

```
…
operationsExportDictFrom: {
   #counter @ #(count) -> #(seconds).
   #ticker @ #(start stop)};
…
```

StopWatch in CLIC

```
…
exportedRequiredPortsDictFrom: {
    #counter @ #(countObservers)
                                -> #(secondsObservers)}
```

StopWatch in CLIC

```
StopWatch>>reset
    self counter count: 0


StopWatch>>initialize
    super initialize.
    self ticker
        tickSelector: #increment;
        stepDelayDuration: 1000;
        processPriority: self Scheduler timingPriority.
```

Conclusion

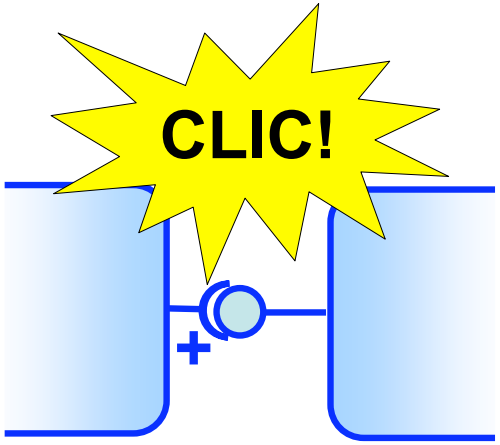
- Components enforce some programming good practices
  - Documentation
    - Architecture
    - Requirements / Dependencies
    - Provided services
    - Parameters
  - Loose coupling
    - Explicit dependencies

Conclusion

- Smalltalk enables object-component symbiosis
  - Easy implementation
    - Reflection
    - No language change
  - Components as first class entities
  - Same run-time => Same performance
  - Ability to use objects ("dirty" components)

Future Works

- Version management
- Import / Export support
- Large scale experiments



CLIC

a Component Model Symbiotic with Smalltalk

N. Bouraqadi and L. Fabresse

Ecole des Mines de Douai

http://vst.ensm-douai.fr/Clic