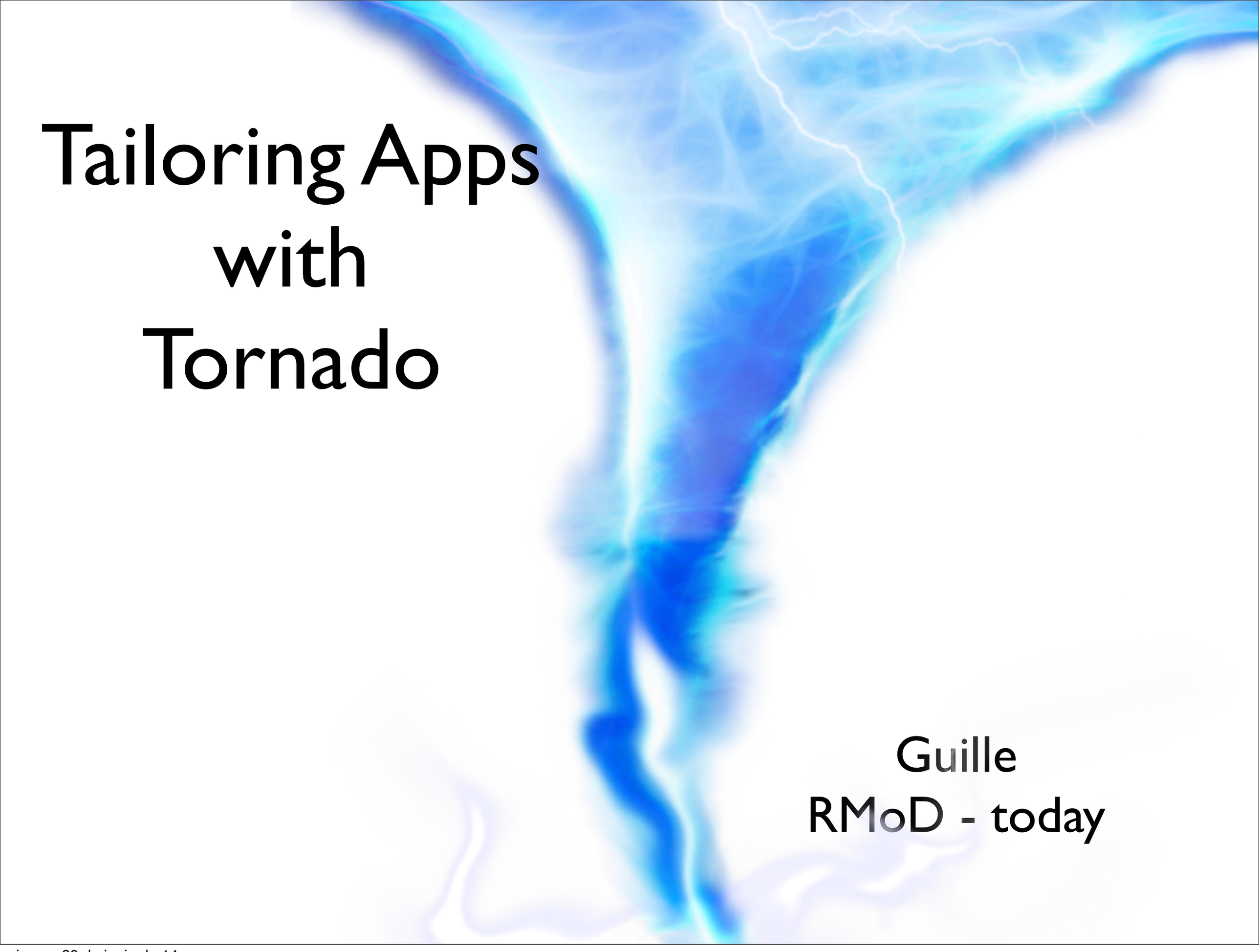
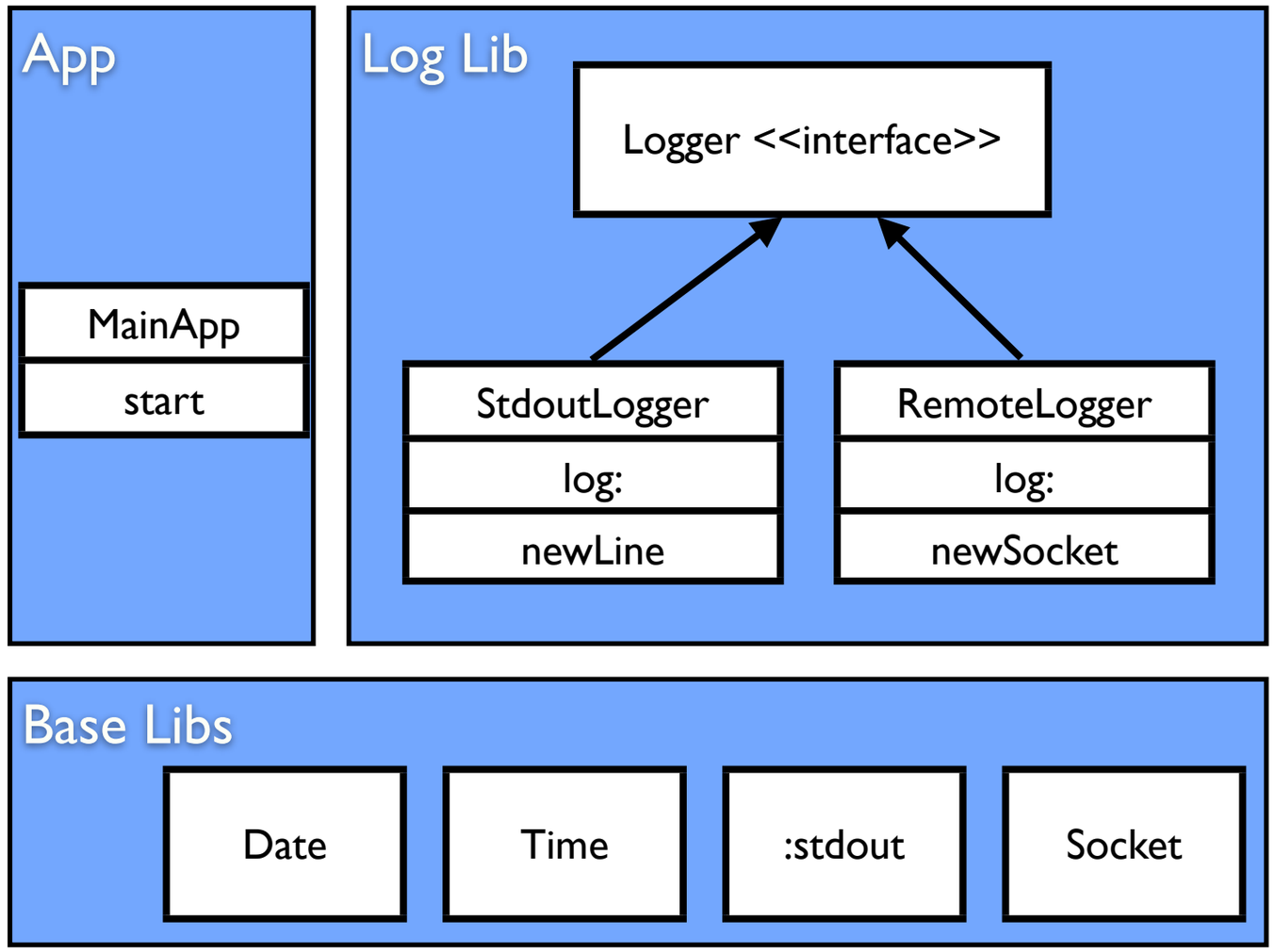
Tailoring Apps with Tornado



Guille
RMoD - today

Lets look at this application...

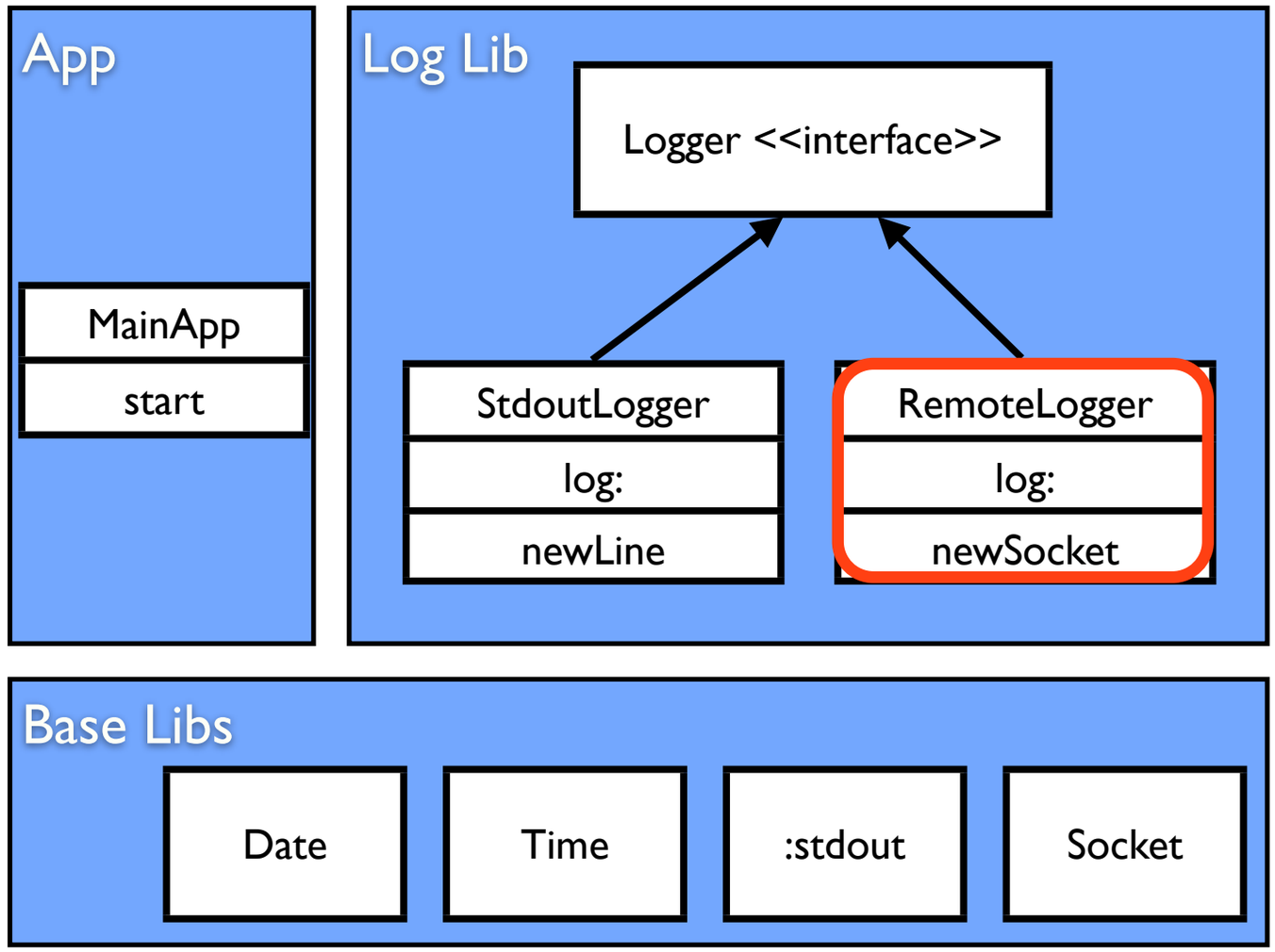
```
MainApp>>start
    logger := StdoutLogger new.
    logger log: ’Application has started’.
    "do something"
    logger log: ’Application has finished’.

StdoutLogger»newLine
    stdout newLine.

StdoutLogger>>log: aMessage
    stdout nextPutAll: Time now printString.
    stdout nextPutAll: aMessage.
    stdout newLine.
```




```
RemoteLogger»log: aMessage
    | socket |
    socket := self newSocket.
    socket nextPutAll: Time now printString.
    socket nextPutAll: aMessage.
    socket newLine.

RemoteLogger»newSocket
    "...."
    "creates an instance of socket given some configuration"
```

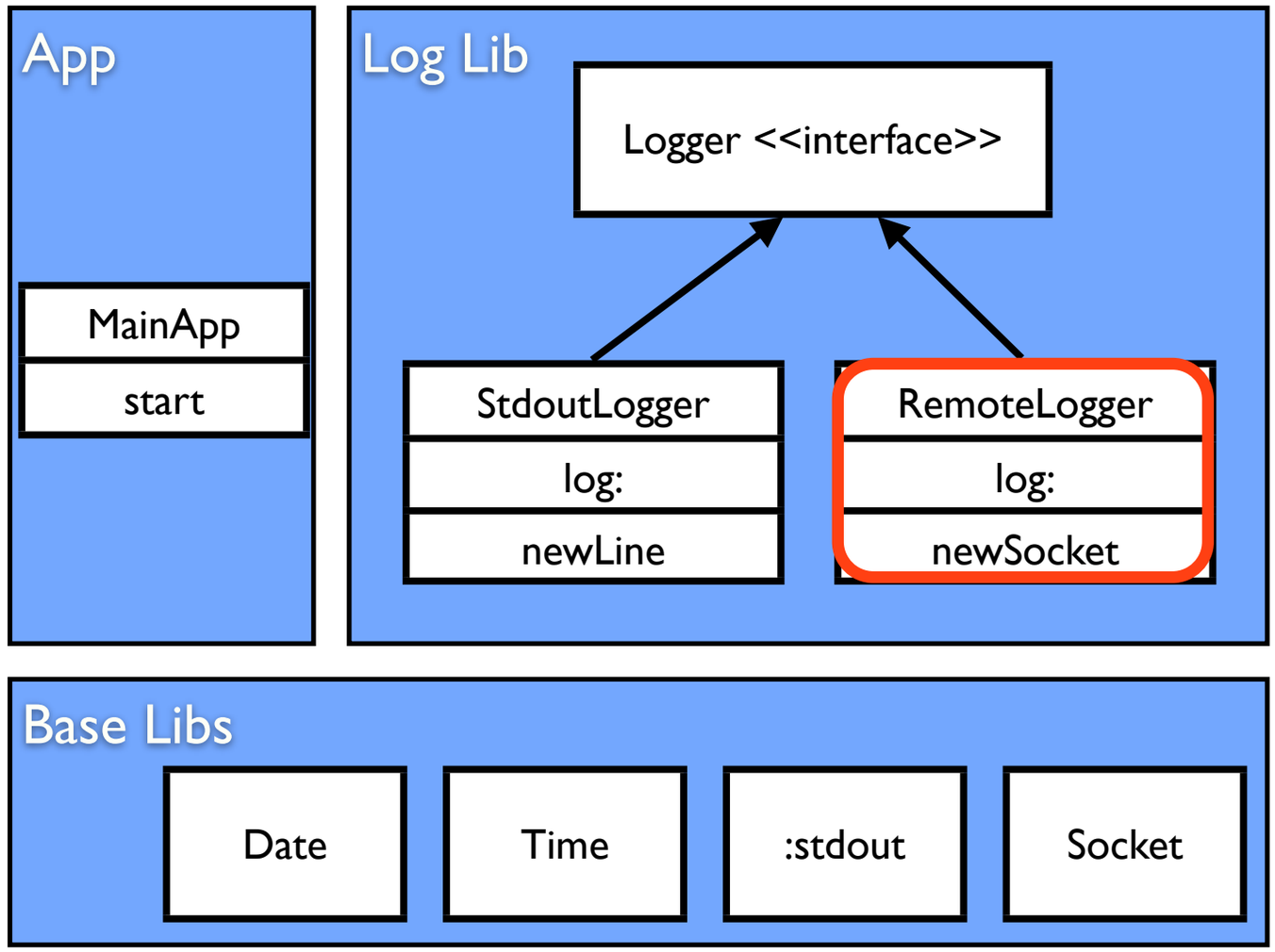
There is *extra* stuff!

```
MainApp>>start
    logger := StdoutLogger new.
    logger log: ’Application has started’.
    "do something"
    logger log: ’Application has finished’.

StdoutLogger»newLine
    stdout newLine.

StdoutLogger>>log: aMessage
    stdout nextPutAll: Time now printString.
    stdout nextPutAll: aMessage.
    stdout newLine.
```




```
RemoteLogger»log: aMessage
    | socket |
    socket := self newSocket.
    socket nextPutAll: Time now printString.
    socket nextPutAll: aMessage.
    socket newLine.

RemoteLogger»newSocket
    "...."
    "creates an instance of socket given some configuration"
```

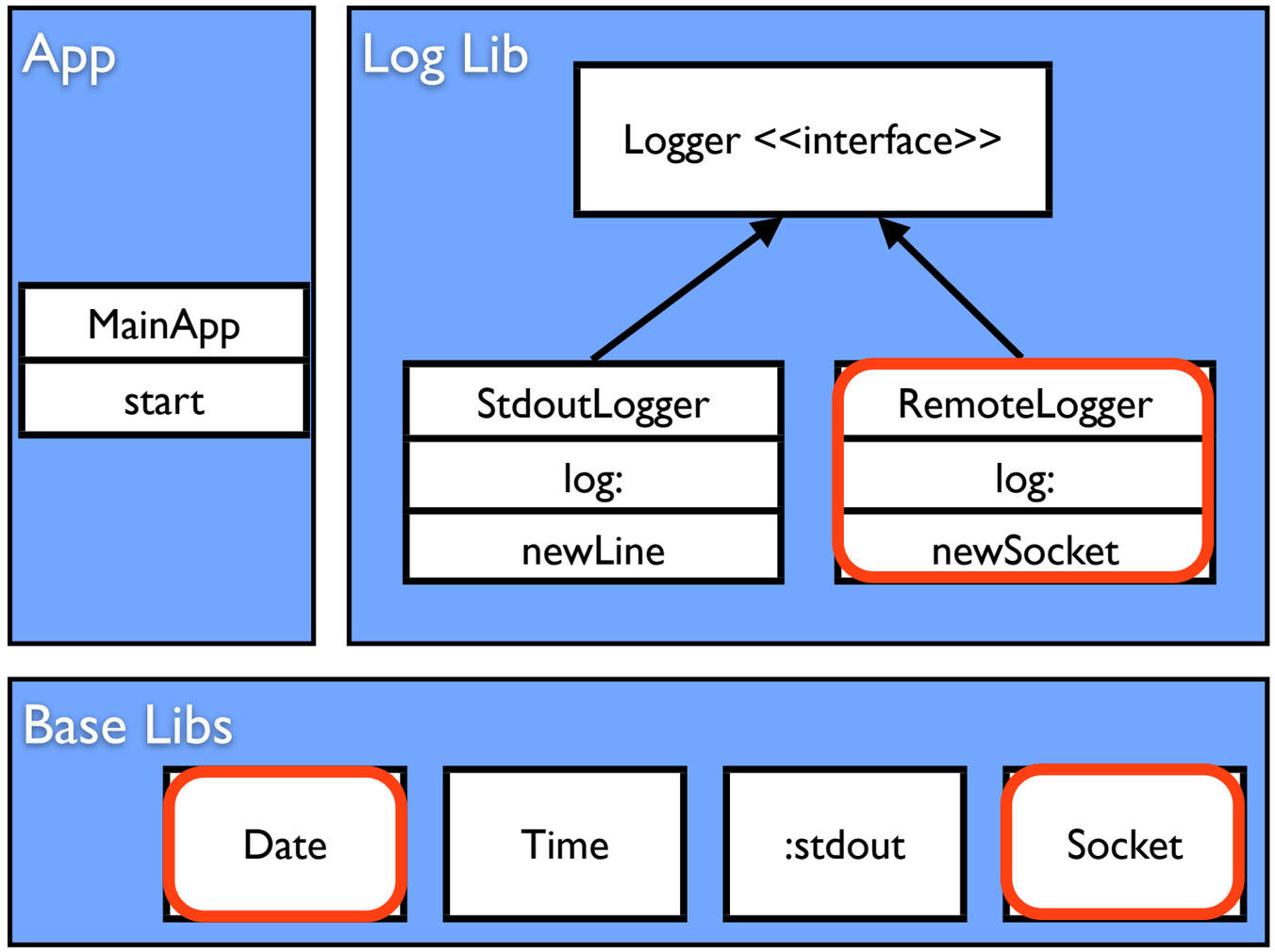
```
MainApp>>start
    logger := StdoutLogger new.
    logger log: ’Application has started’.
    "do something"
    logger log: ’Application has finished’.

StdoutLogger»newLine
    stdout newLine.

StdoutLogger>>log: aMessage
    stdout nextPutAll: Time now printString.
    stdout nextPutAll: aMessage.
    stdout newLine.
```




```
RemoteLogger»log: aMessage
    | socket |
    socket := self newSocket.
    socket nextPutAll: Time now printString.
    socket nextPutAll: aMessage.
    socket newLine.
```

```
RemoteLogger»newSocket
    "...."
    "creates an instance of socket given some configuration"
```

```
MainApp>>start
    logger := StdoutLogger new.
    logger log: ’Application has started’.
    "do something"
    logger log: ’Application has finished’.

StdoutLogger»newLine
    stdout newLine.

StdoutLogger>>log: aMessage
    stdout nextPutAll: Time now printString.
    stdout nextPutAll: aMessage.
    stdout newLine.
```



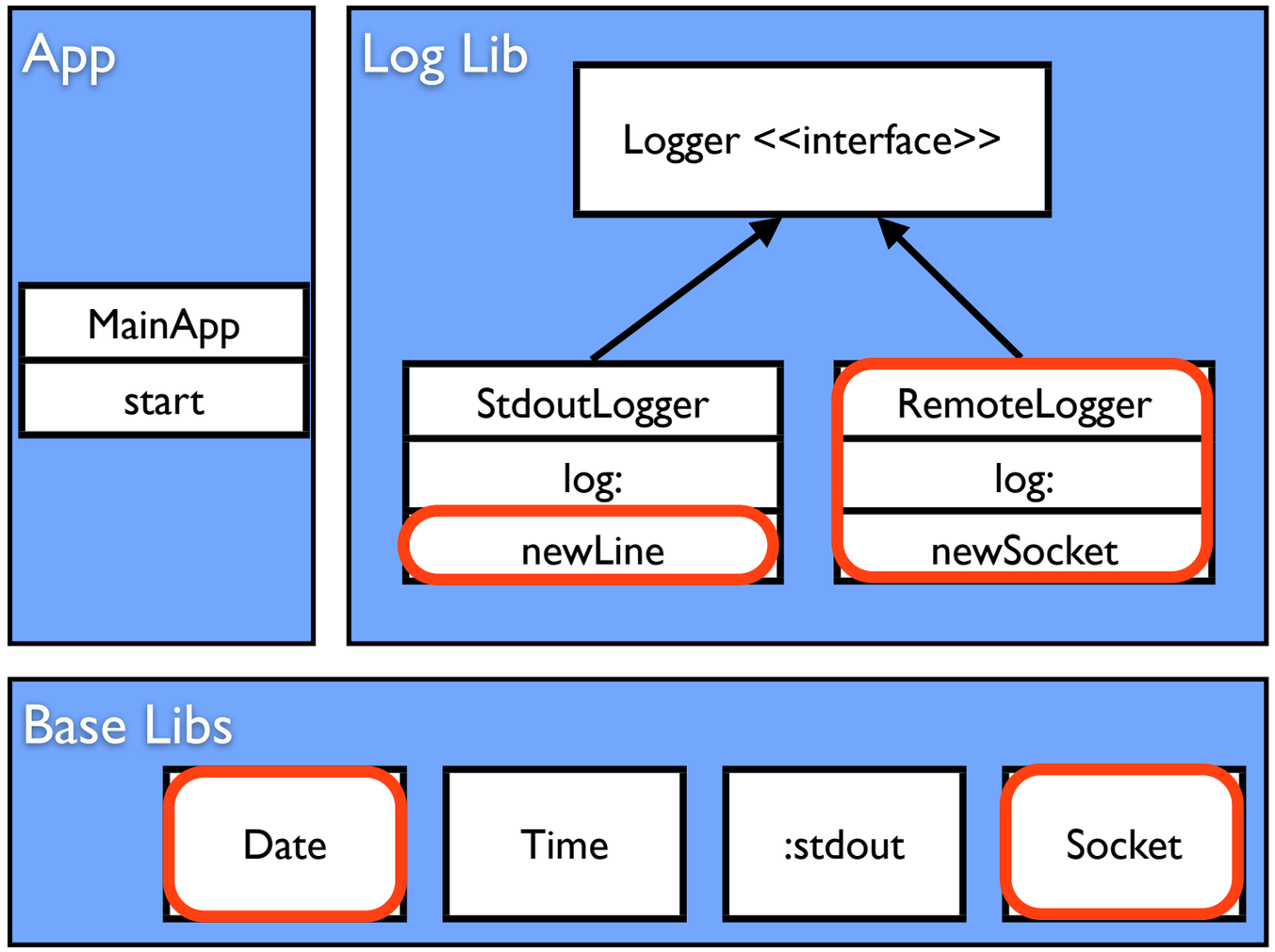

```
RemoteLogger»log: aMessage
    | socket |
    socket := self newSocket.
    socket nextPutAll: Time now printString.
    socket nextPutAll: aMessage.
    socket newLine.
```

```
RemoteLogger»newSocket
    "...."
    "creates an instance of socket given some configuration"
```

```
MainApp>>start
    logger := StdoutLogger new.
    logger log: ’Application has started’.
    "do something"
    logger log: ’Application has finished’.
```

```
StdoutLogger»newLine
    stdout newLine.
```

```
StdoutLogger>>log: aMessage
    stdout nextPutAll: Time now printString.
    stdout nextPutAll: aMessage.
    stdout newLine.
```




```
RemoteLogger»log: aMessage
    | socket |
    socket := self newSocket.
    socket nextPutAll: Time now printString.
    socket nextPutAll: aMessage.
    socket newLine.
```

```
RemoteLogger»newSocket
    "...."
    "creates an instance of socket given some configuration"
```

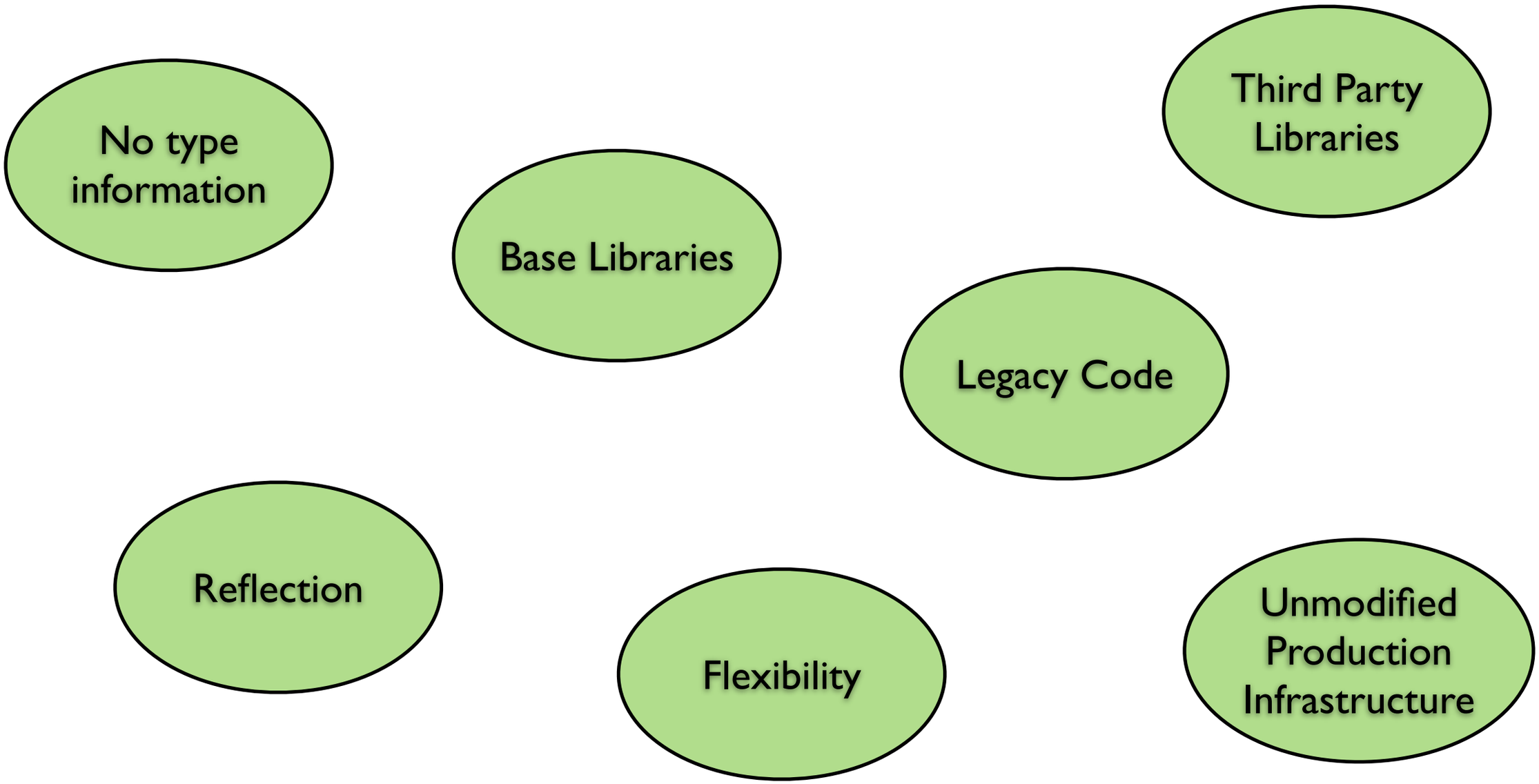
And now think about this

- Pharo 2.0 (prepared for production) = 12.87Mb
- Seaside (prepared for production) = 4.33Mb

Not counting source files, nor VM

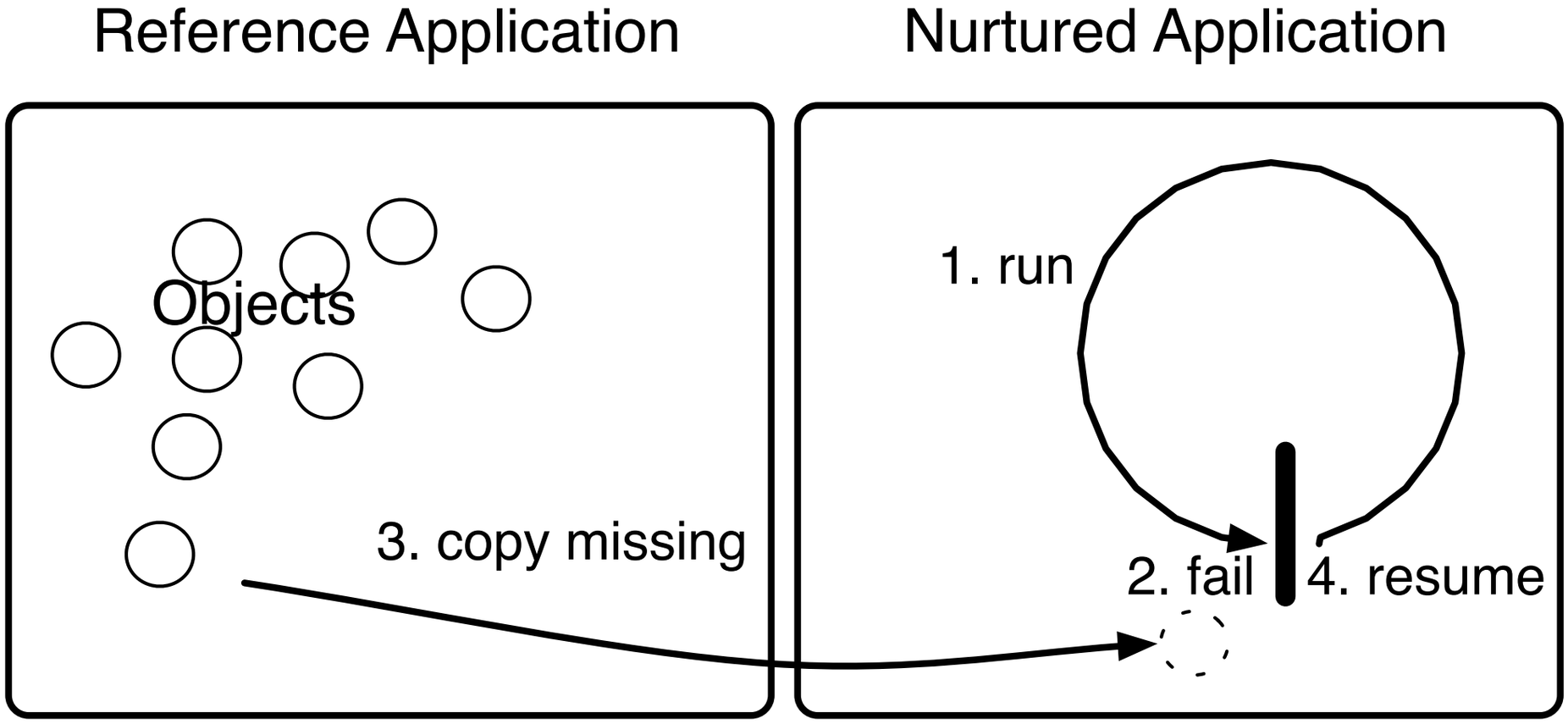
Can we get rid of the unnecessary stuff?
(tailoring)
and...
How much do we gain?

The challenges



So we made Tornado...

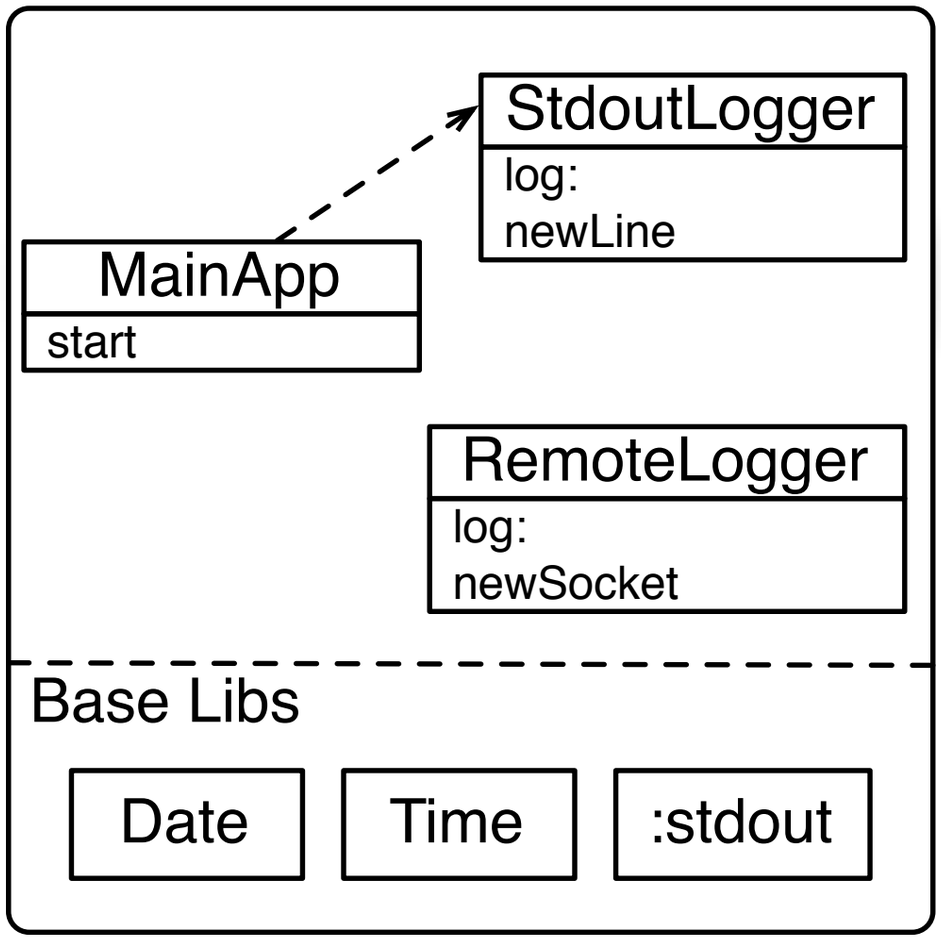
A run-fail-grow approach...



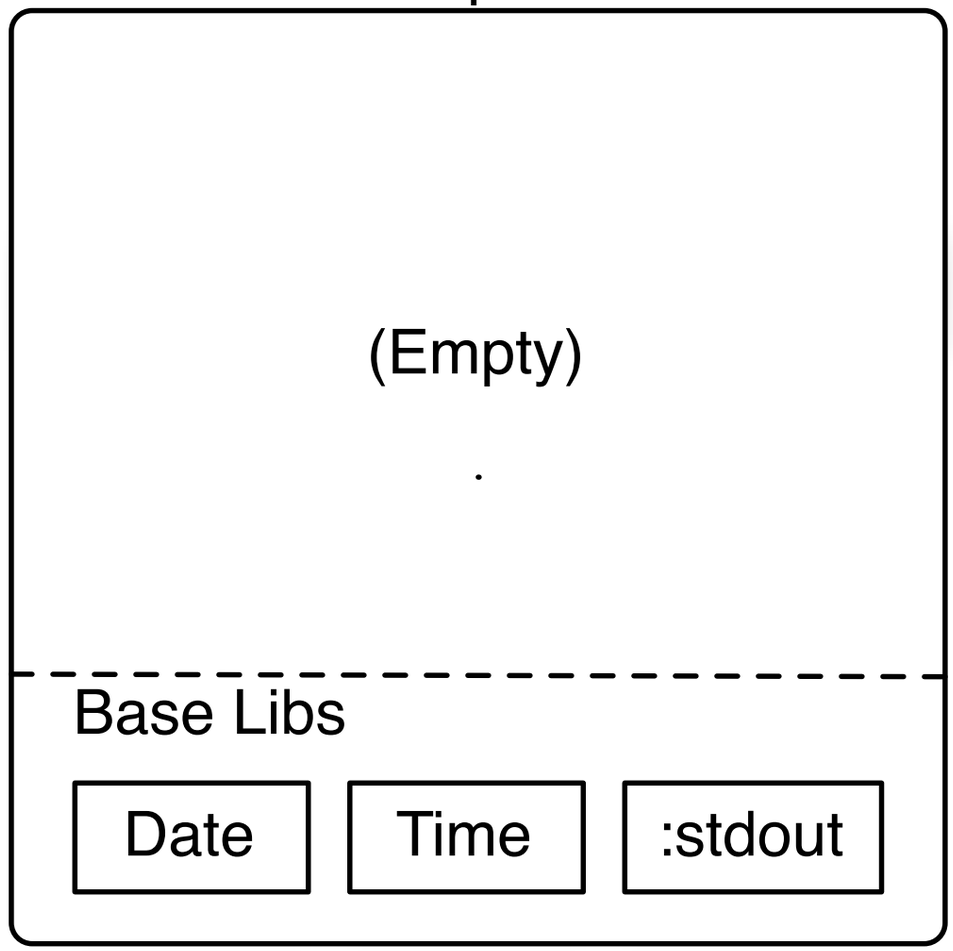
A run-fail-grow approach...

Let's see how this works

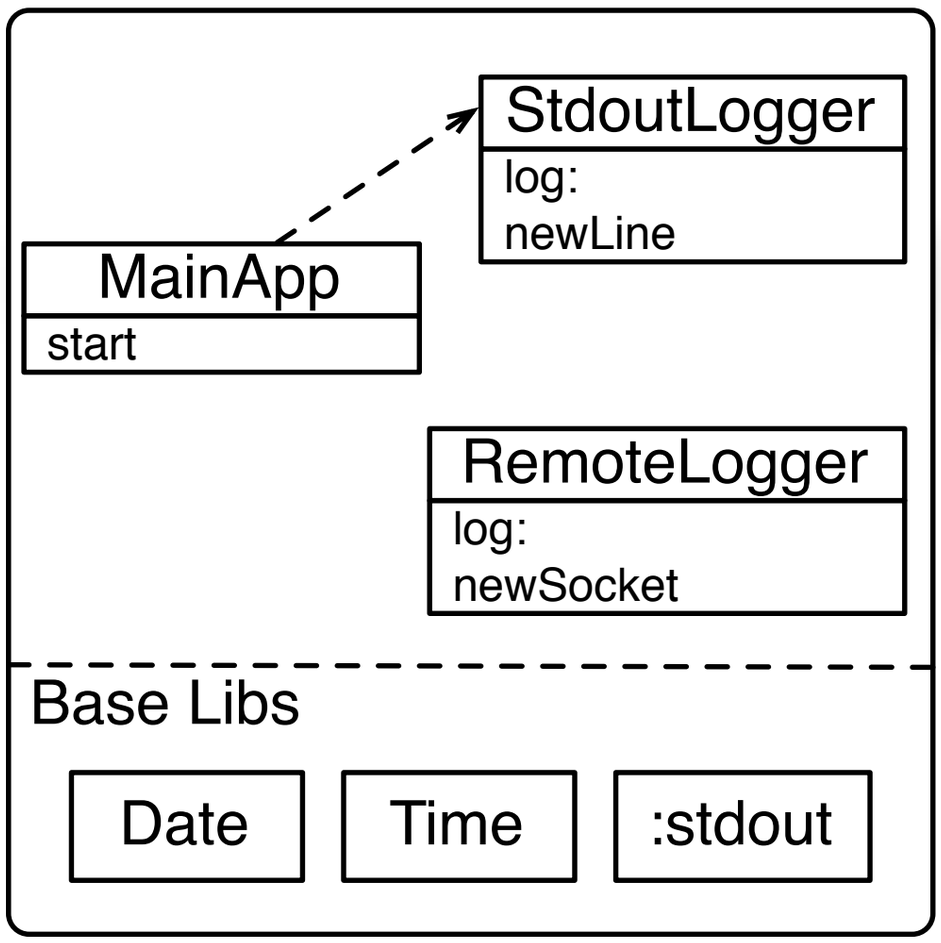
Reference Application



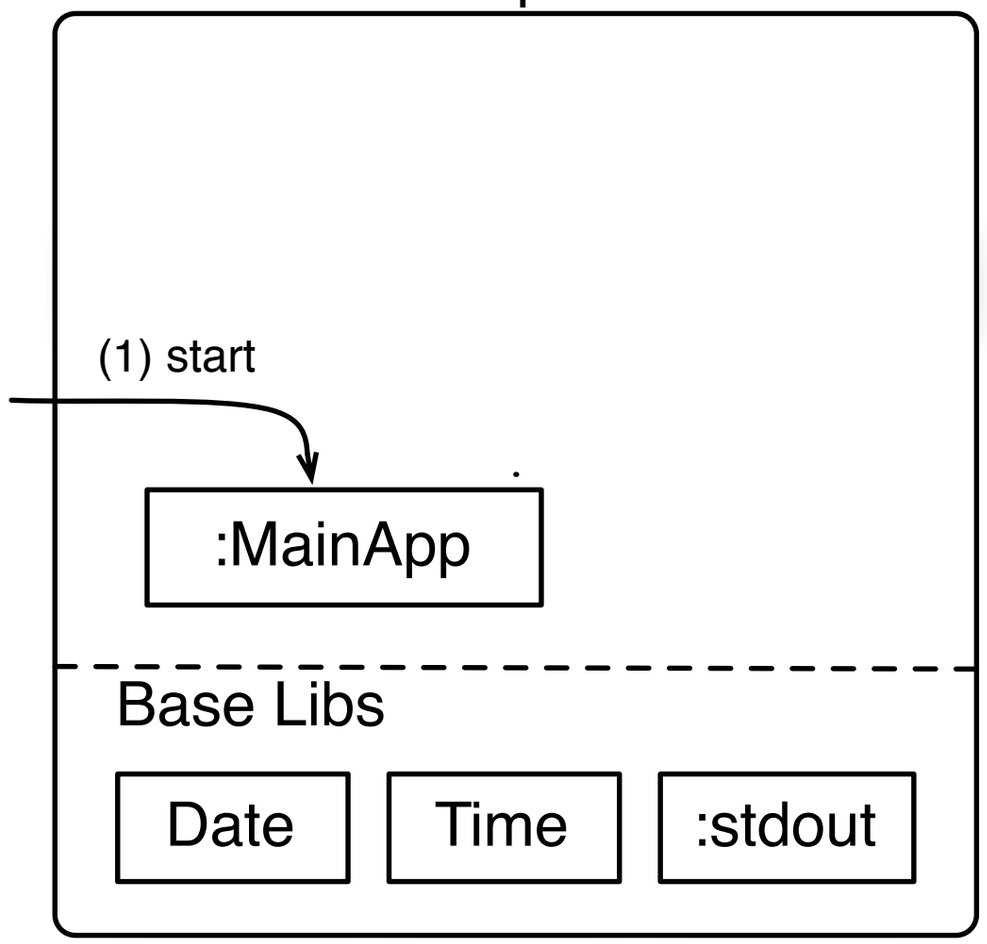
Step 0



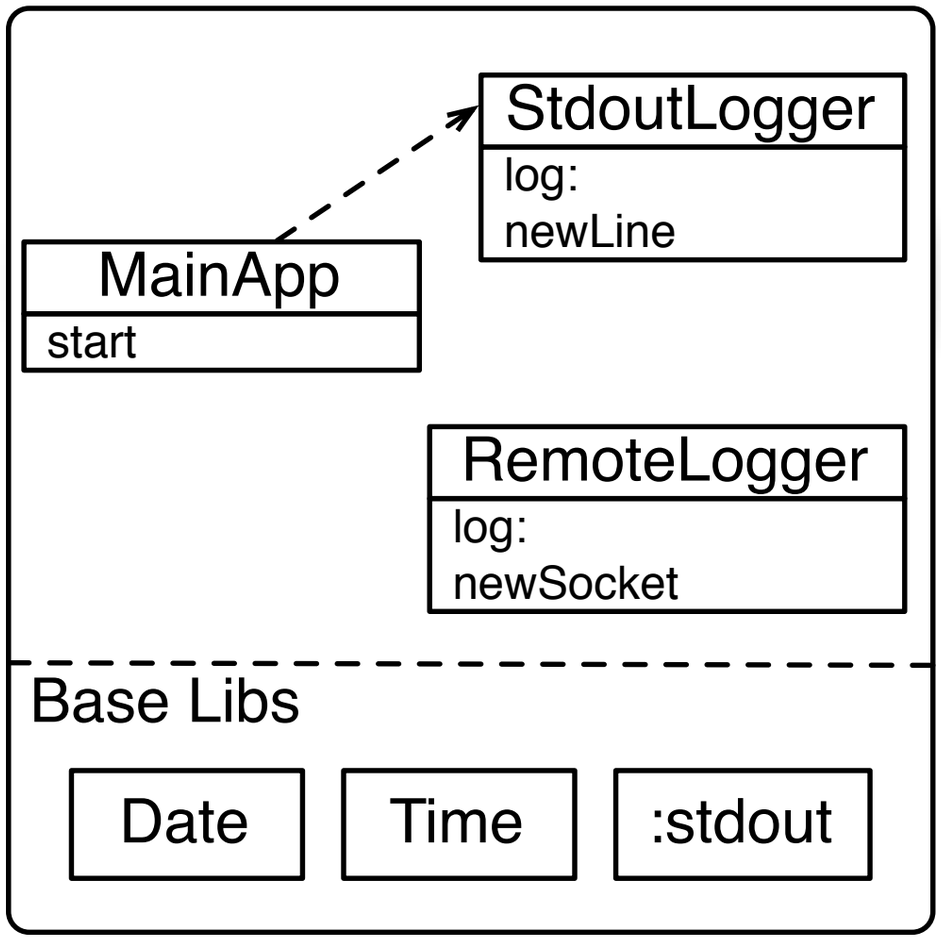
Reference Application



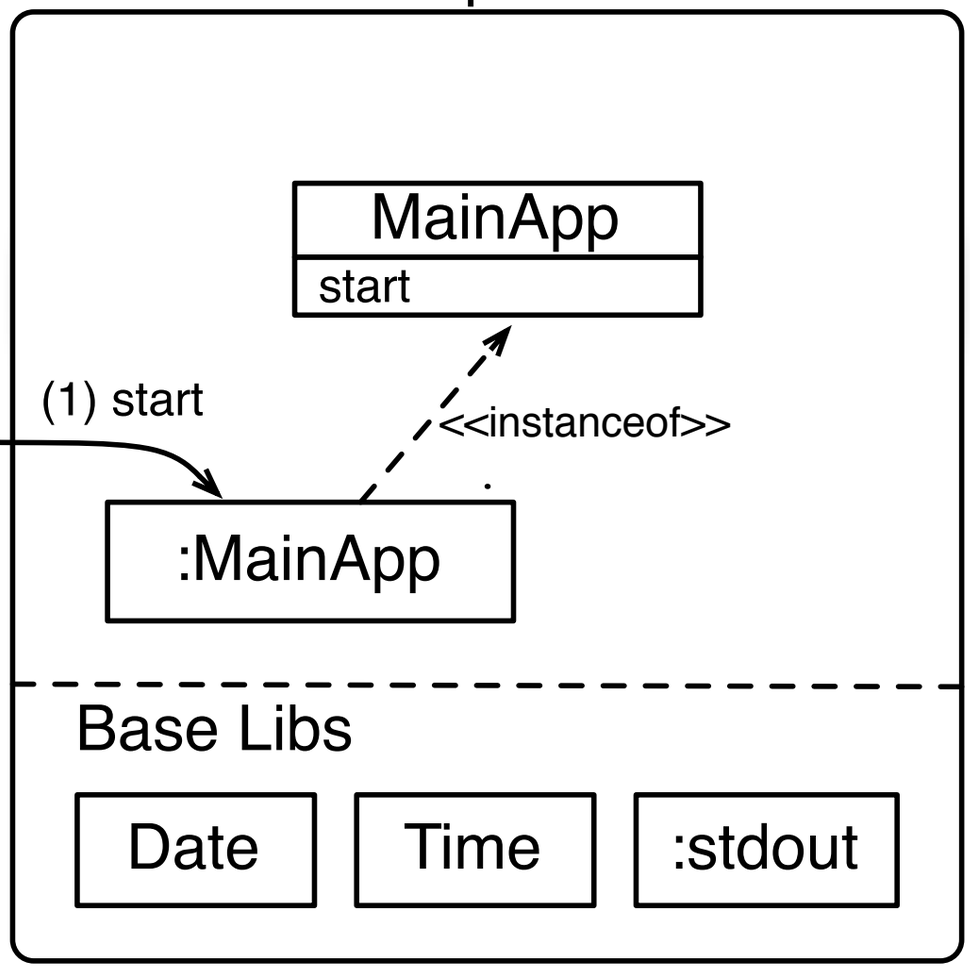
Step 1



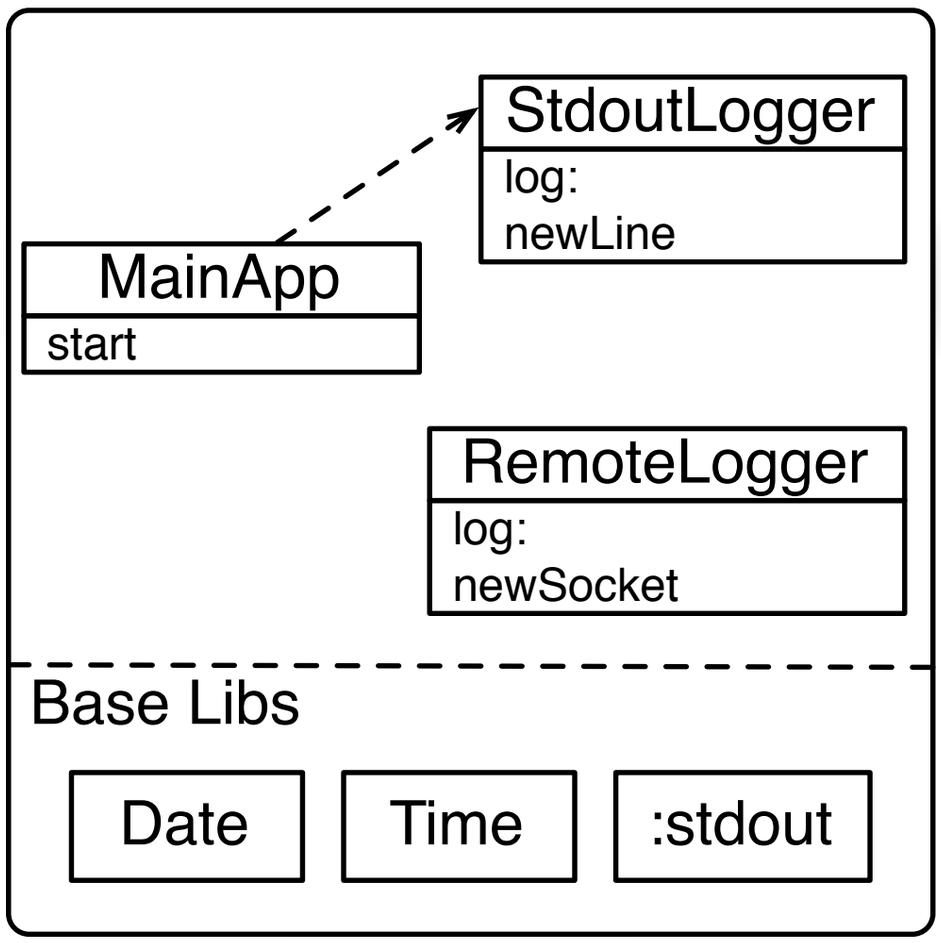
Reference Application



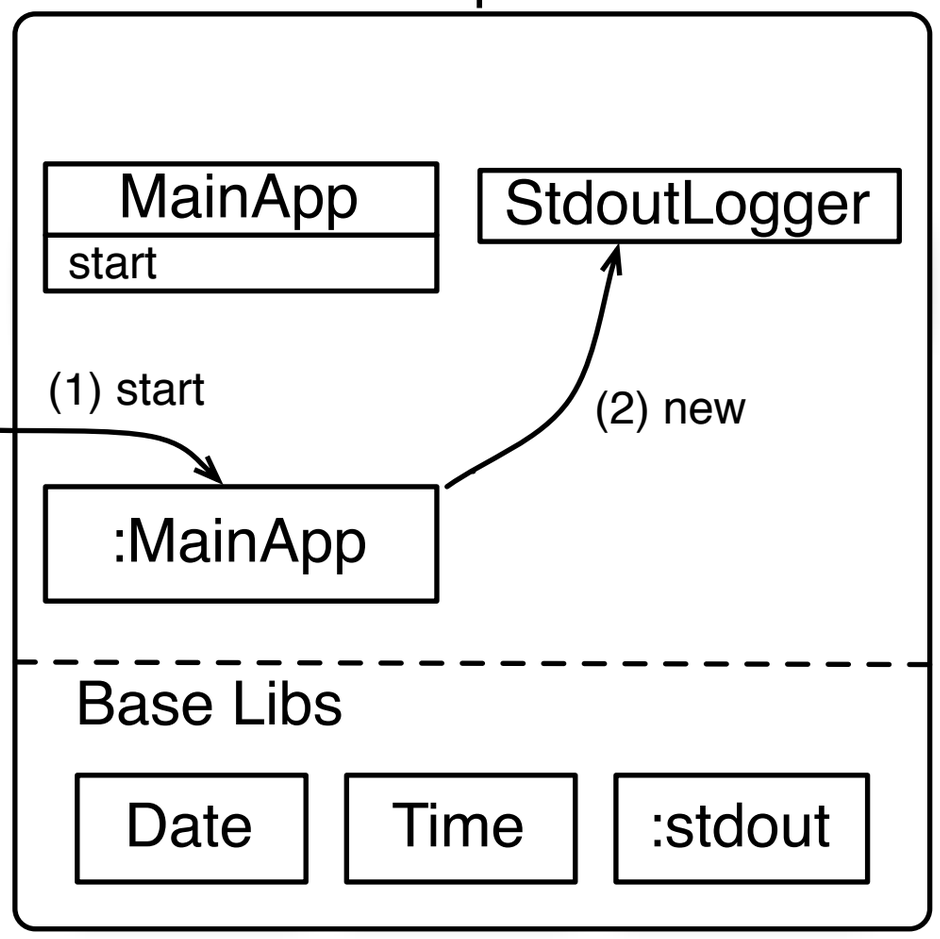
Step 2



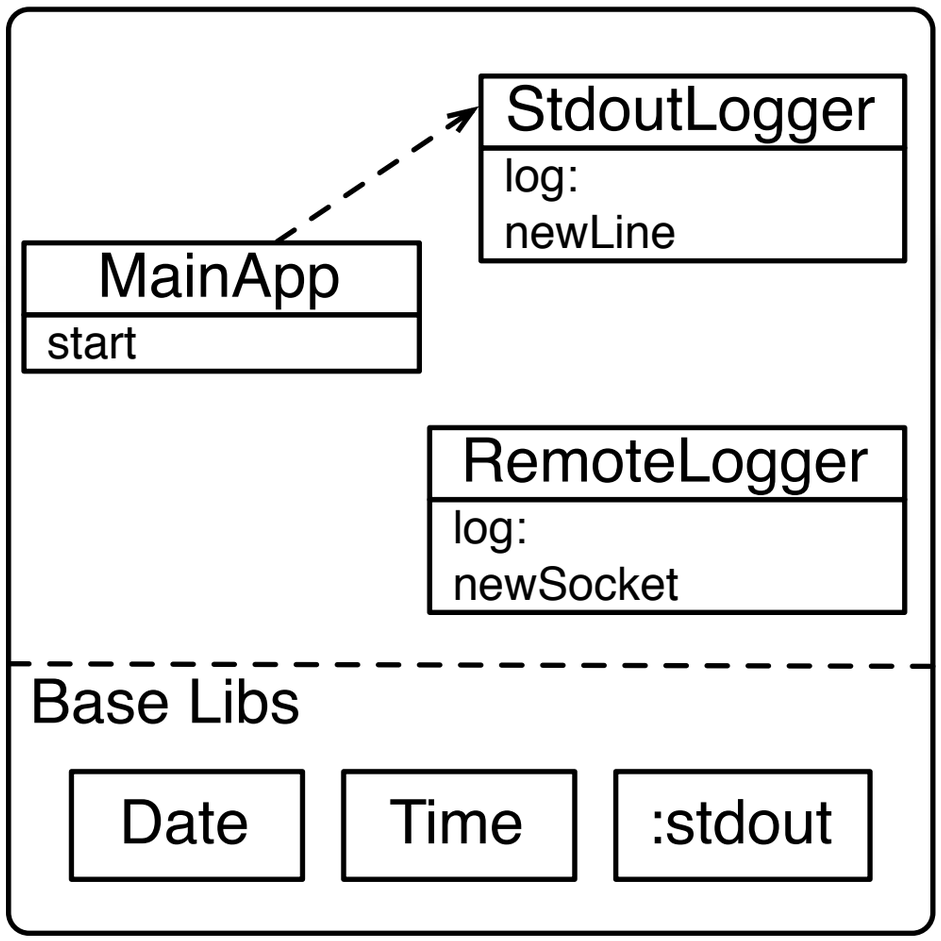
Reference Application



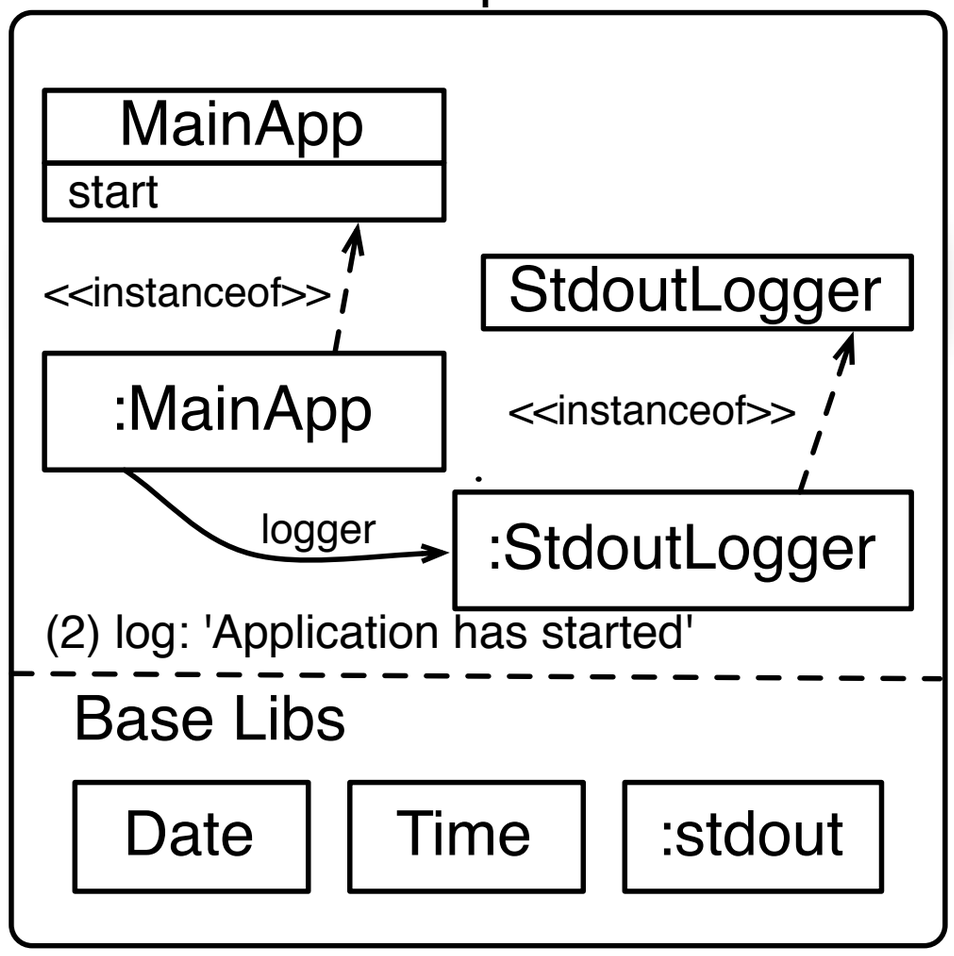
Step 3



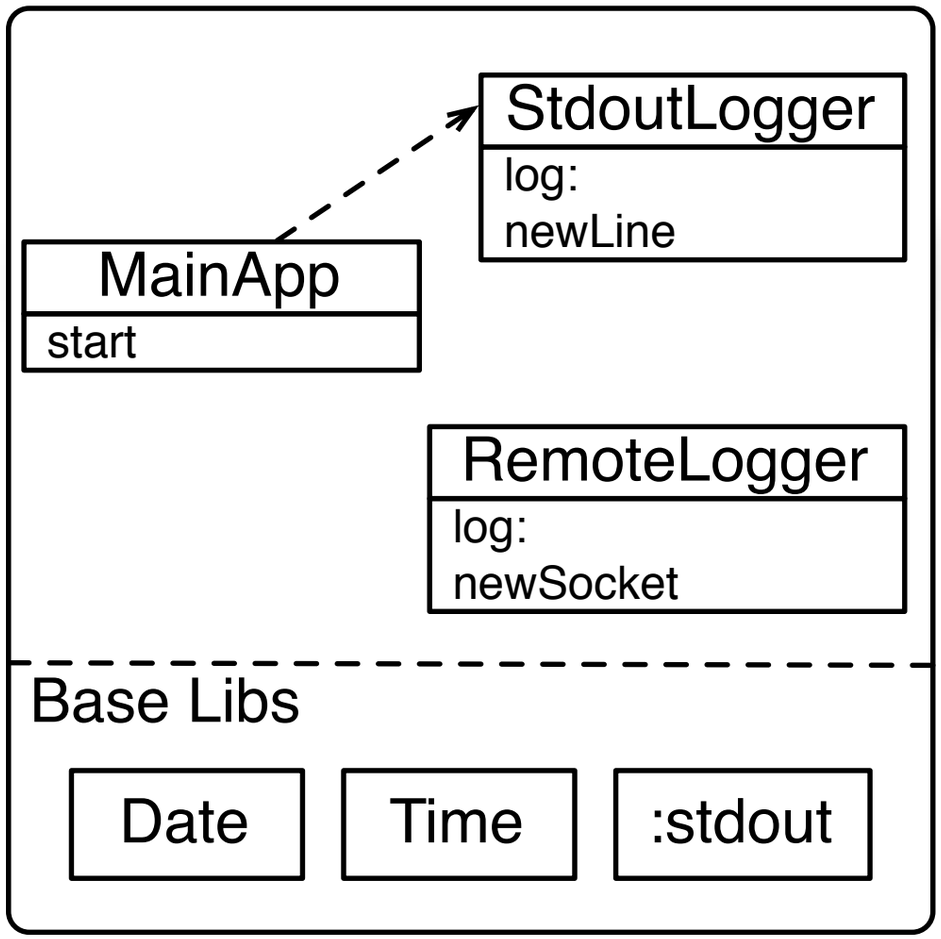
Reference Application



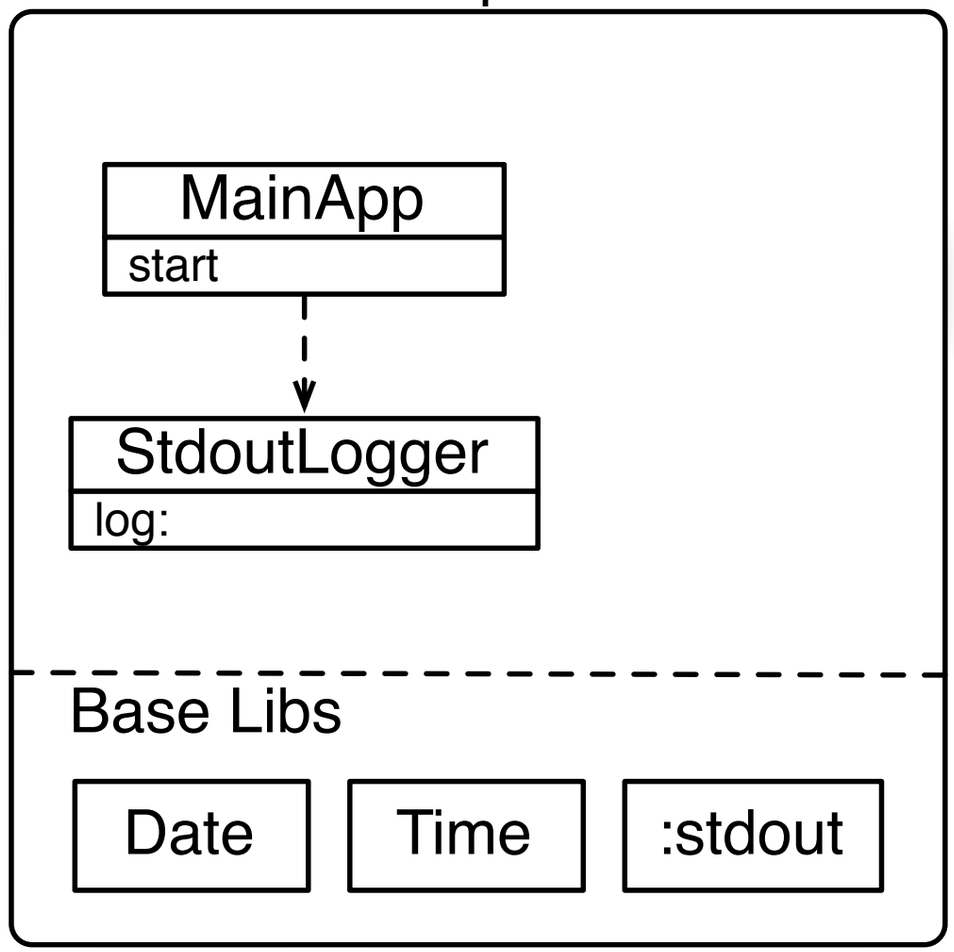
Step 4



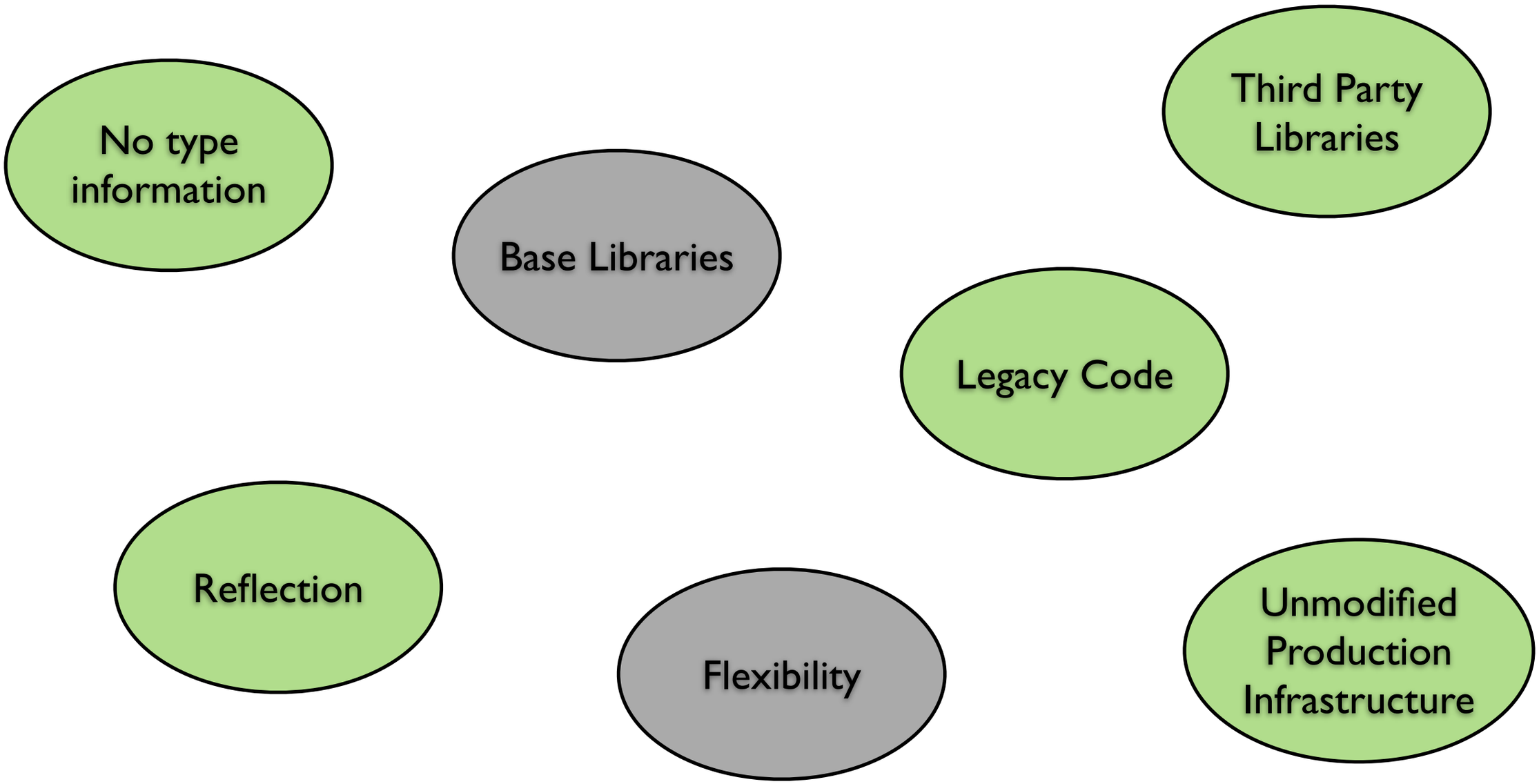
Reference Application



Step 5



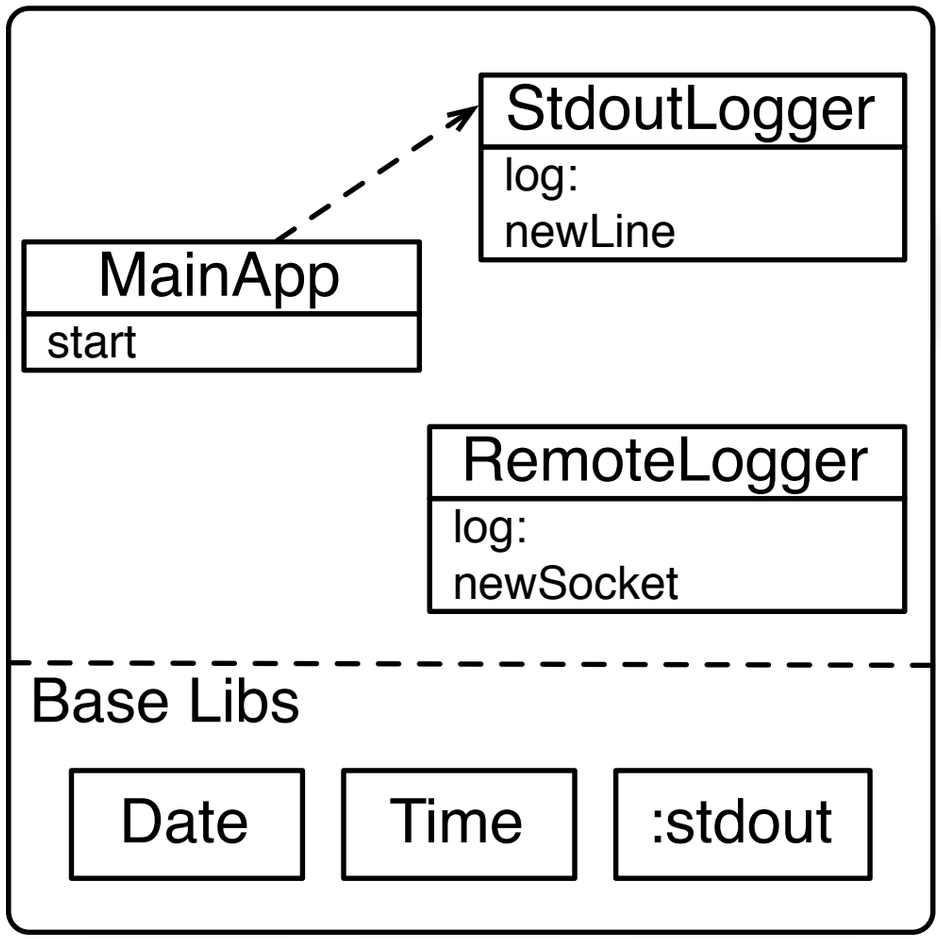
And so with that we tackled...



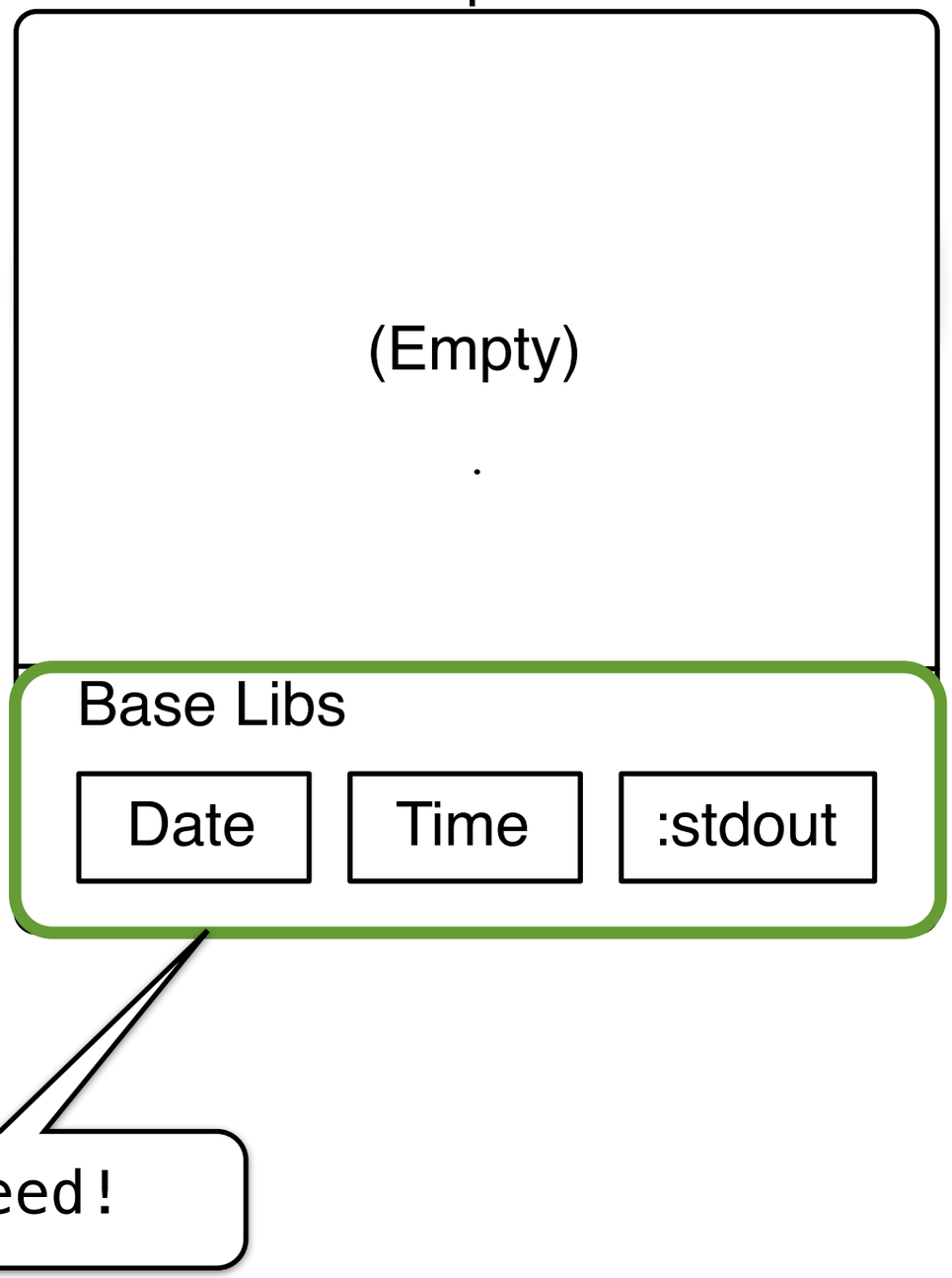
The Seeds

or... how to tackle Base Libraries

Reference Application

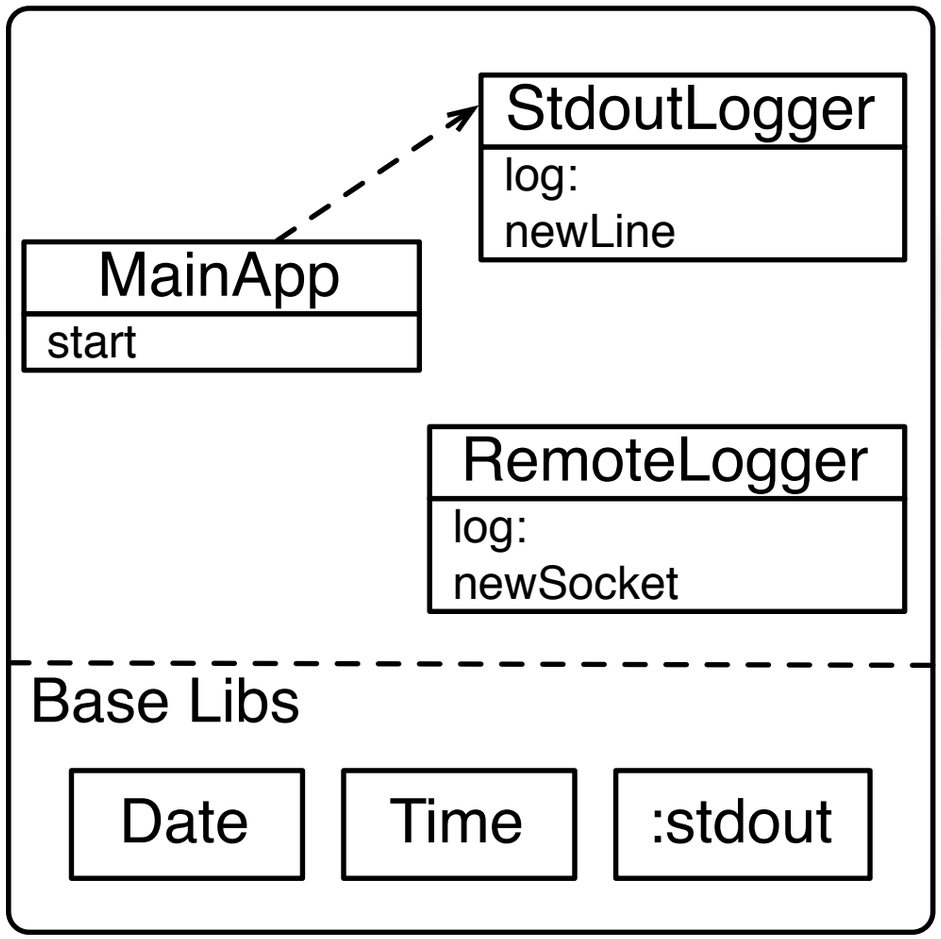


Step 0

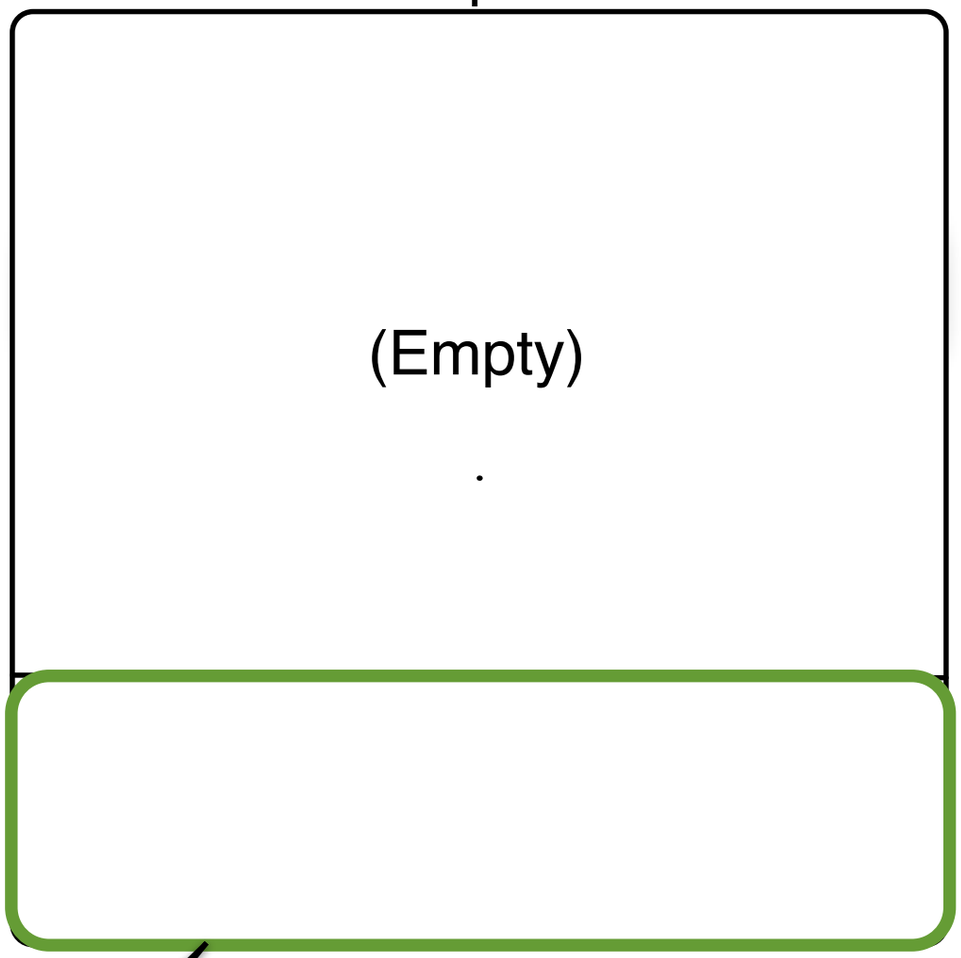


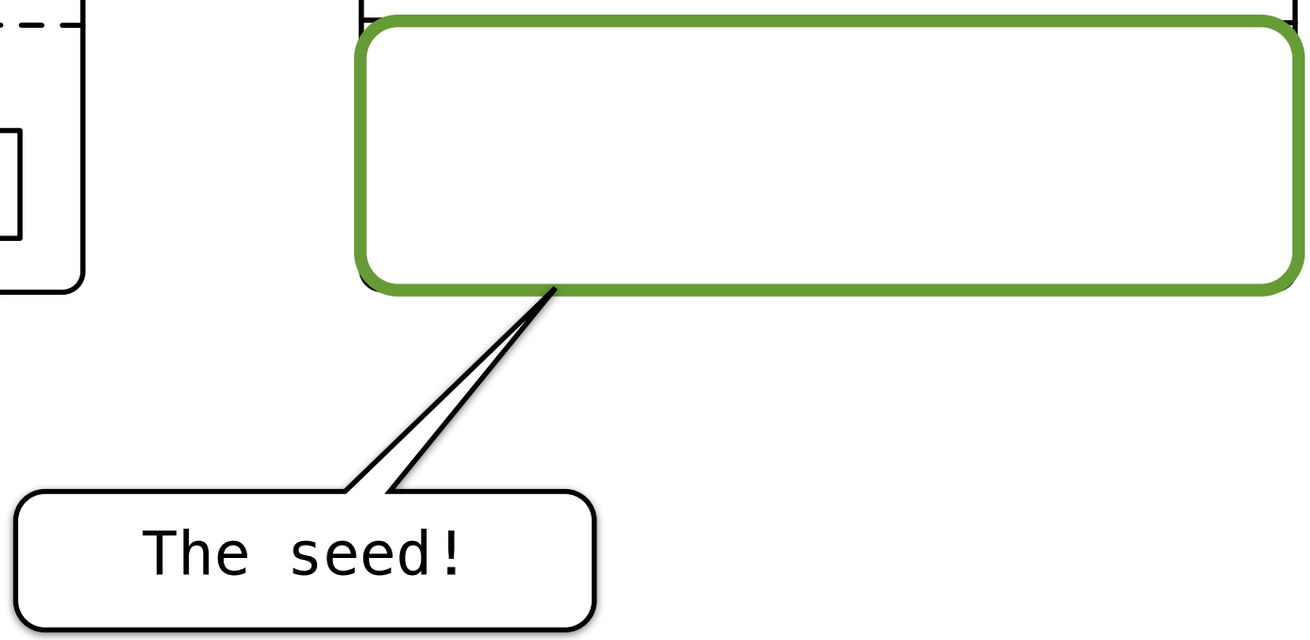
What should have happened if...

Reference Application



Step 0





About seeds

Allow to customize the minimal ensured code units

Should contain VM minimal requirements (not completely empty)

And so with that we tackled...

No type information

Base Libraries

Third Party Libraries

Legacy Code

Reflection

Flexibility

Unmodified Production Infrastructure

Does it really work? (Results)

HelloWorld, tailoring all:

12872 Kb => 131 Kb

Seaside Counter tailoring all:

17250 Kb => 573 Kb

Seaside Counter not tailoring base libraries:

17250 Kb => 13090 Kb

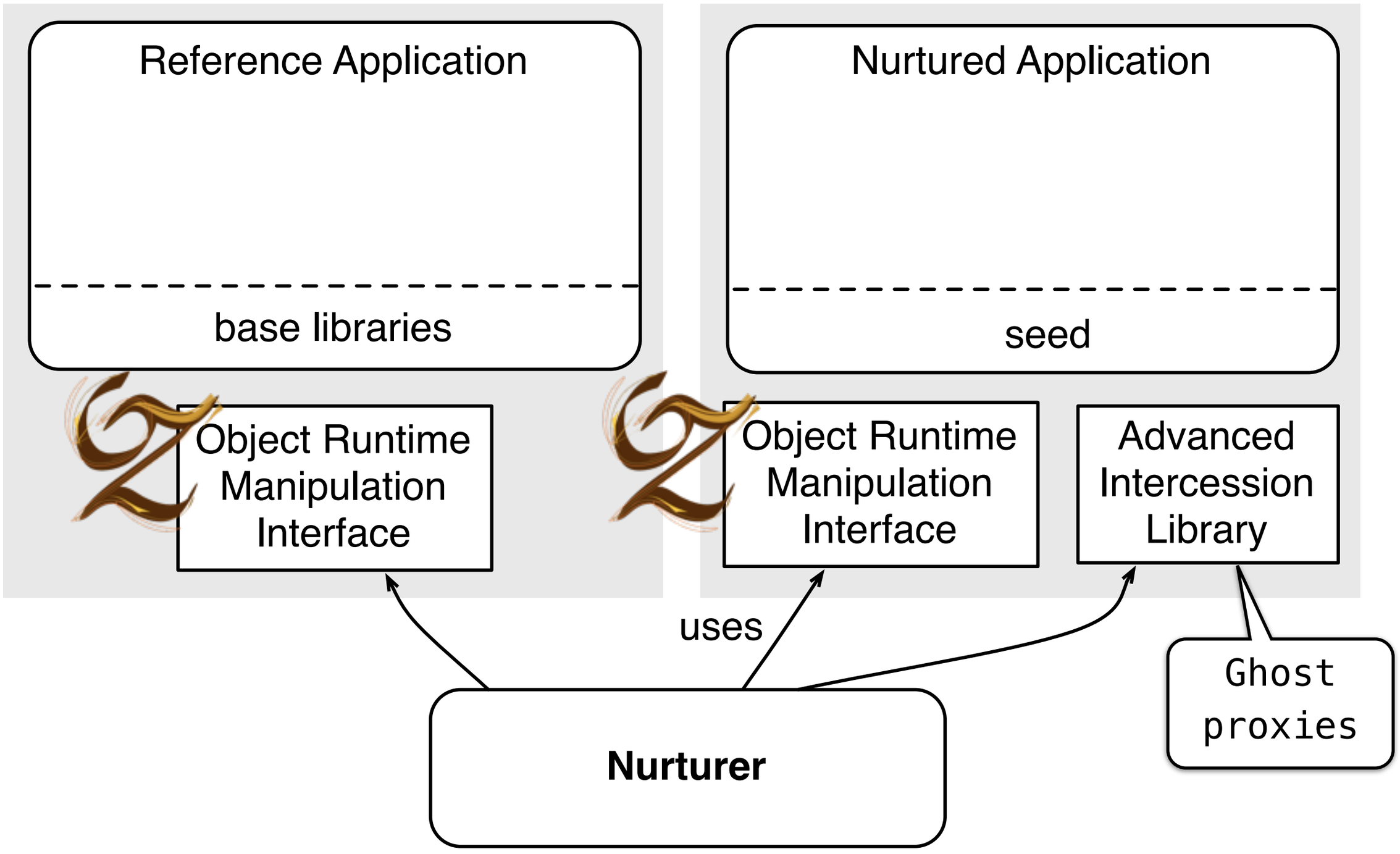
Opening a discussion

- Ensuring the completeness of the tailoring
- Tornado is not friendly with all kind of designs
- Reflection, and program meta-data shrinking for free
- Your ideas?

Tailoring Apps with Tornado

- Run-fail-grow approach for tailoring
- Works on dynamic languages
- Flexible
- We made it work with different applications (including a seaside app)
- Paper sent to DLS two weeks ago

In Details...


Reference Application
base libraries
Object Runtime Manipulation Interface
Nurtured Application
seed
Object Runtime Manipulation Interface
Advanced Intercession Library
uses
Nurturer
Ghost proxies

The Object Runtime Manipulation Interface

- Starting / Stopping the runtime
- Installing / Terminating Processes
- Inspect the objects and classes available
- Create and modify new objects and classes

Ghost Proxies: one model to capture them all

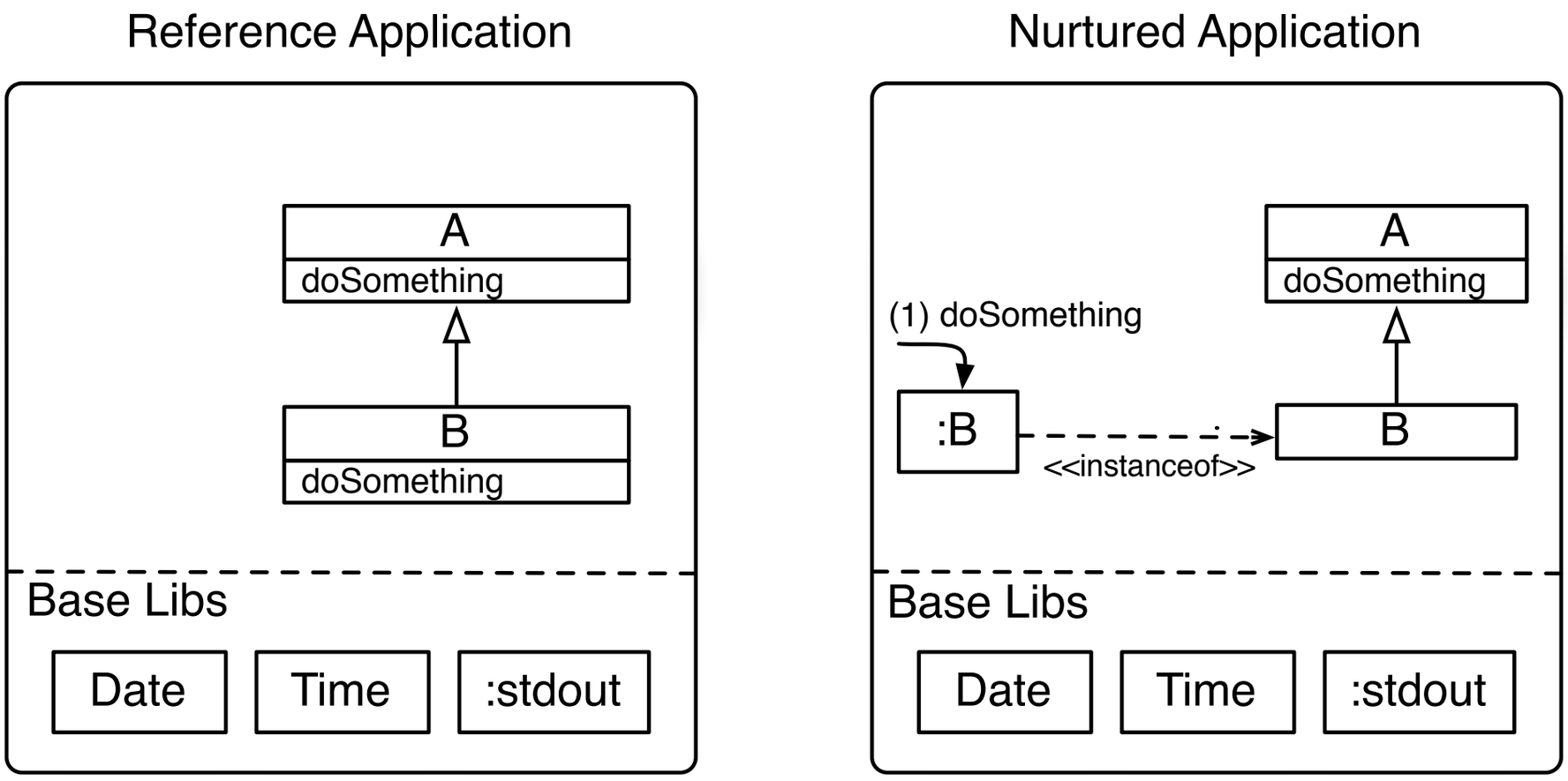
Used to Implement execution traps:

- Detect the usage of not-yet installed code units
- Suspend the execution
- Return control to the nurturer
- Finally, replaced by the newly installed object

Execution traps

- Missing Object
- Missing Method
- Missing Override
- Primitive Method

Why Missing Overrides?



Why Primitive Traps?