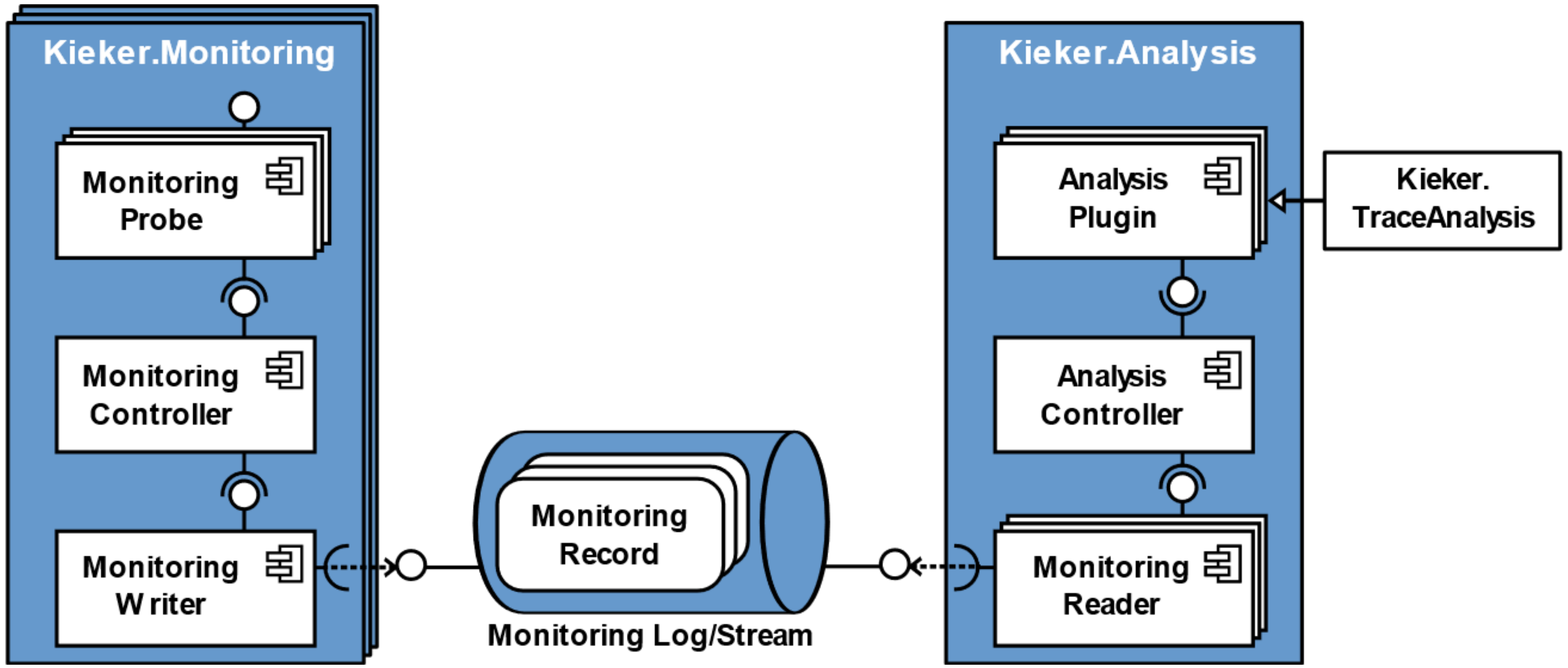
Instrumenting Python with Kieker

Serafim Simonov, Thomas Duellmann, Reiner Jung, Sven Gundlach

Kieker Architecture



Kieker Instrumentation

Manually

- Type the API at every place in the code by yourself.

Automatic

- Code weaving/ Aspect Oriented Programming(AOP)
- AspectJ or AspectC
- Compiler weaving

AOP in Python

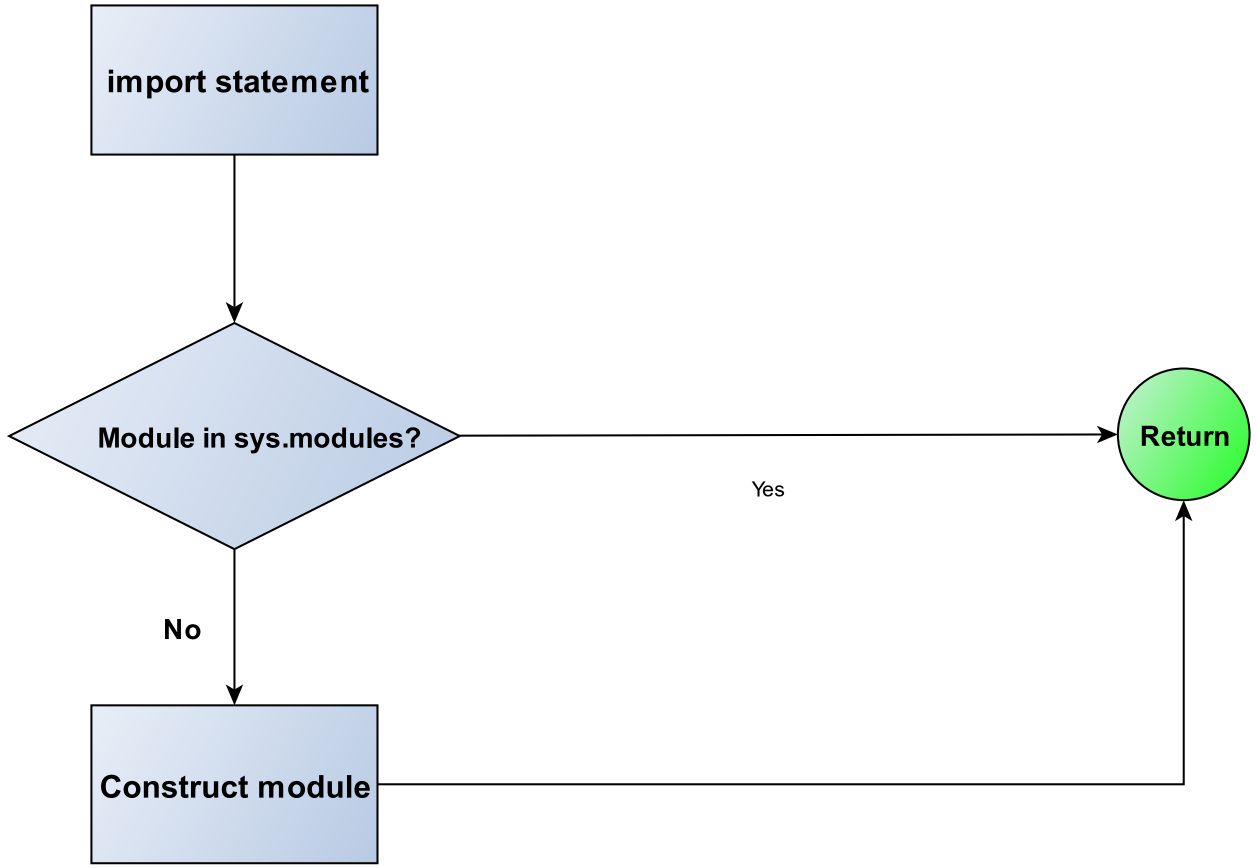
Python Features

- Decorators
- Meta classes
- Everything is an object

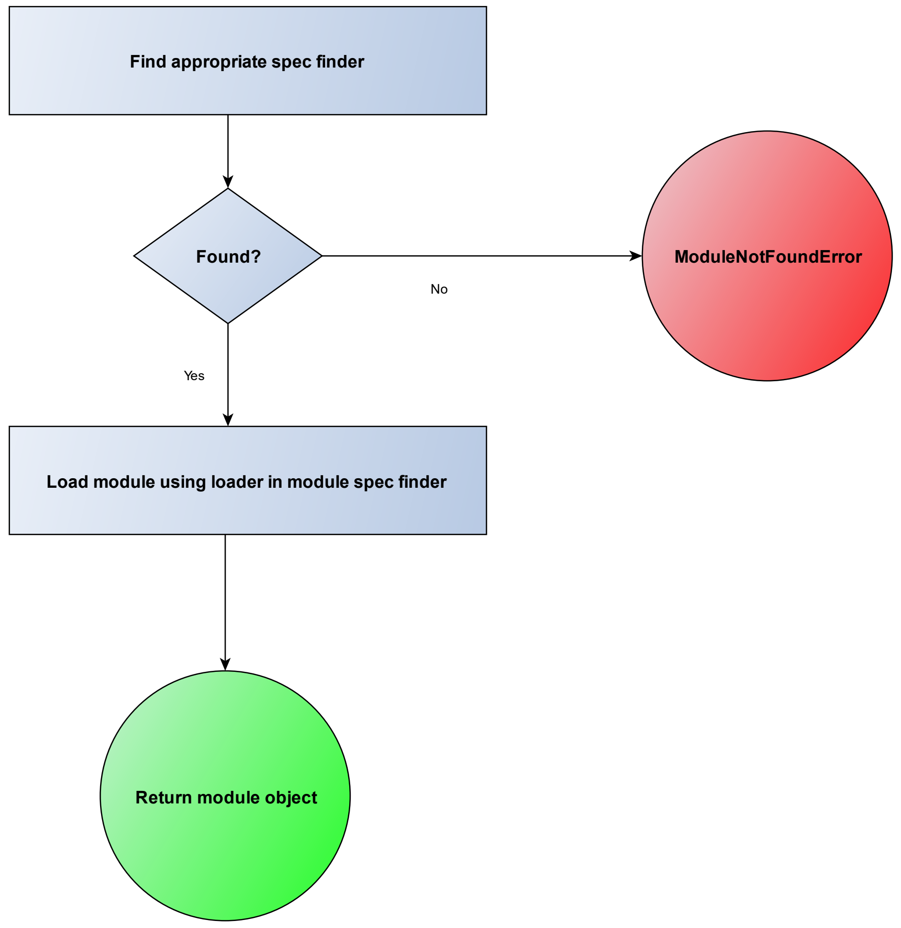
Frameworks

- aspectLib
  - Testing
- Spring Python
  - Last commit in 2014

Import Process in Python



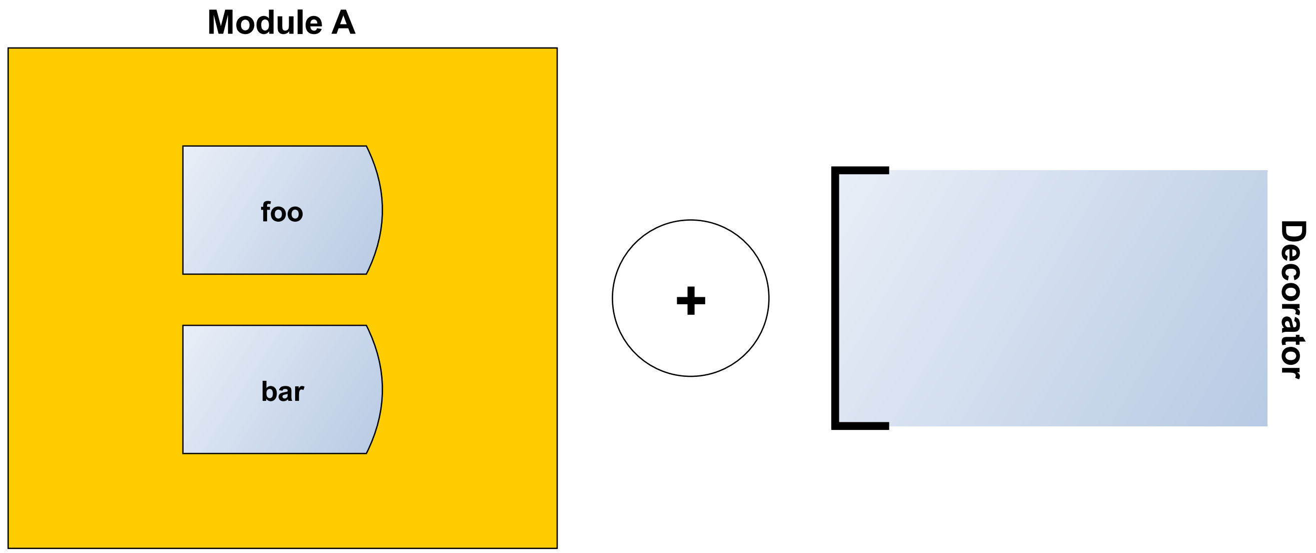
Find and Load Modules in Python



Find appropriate spec finder
Found?
No
ModuleNotFoundError
Yes
Load module using loader in module spec finder
Return module object

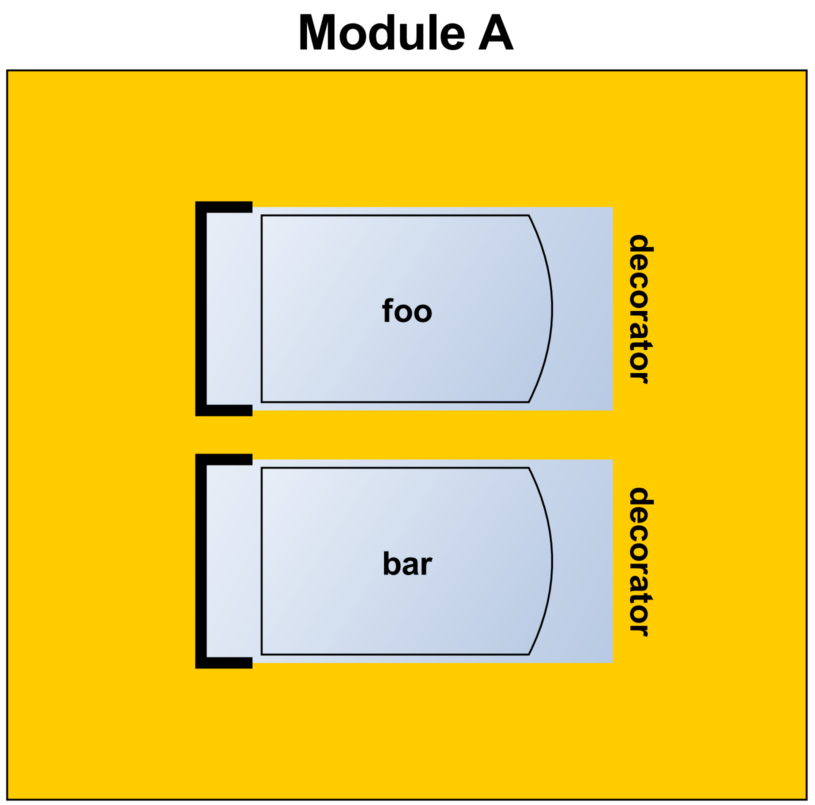
sys.meta_path	• Module spec finders
Module Spec	• Module name, path, loader
Loader	• load the module

Post Import Weaving



1. Load the module normally
   - The source code is executed
2. Apply the decorator to functions

Issues



Problems:

- Information about original objects is lost
- Side effects
- Overwriting by the user code is possible

Pre Import Weaving

1. Find the module
2. Apply changes
    1. Parse the source code in to AST
    2. Add the decorators „manually“
    3. Compile and execute the modified version of the module
3. Execute

Old Problems Fixed

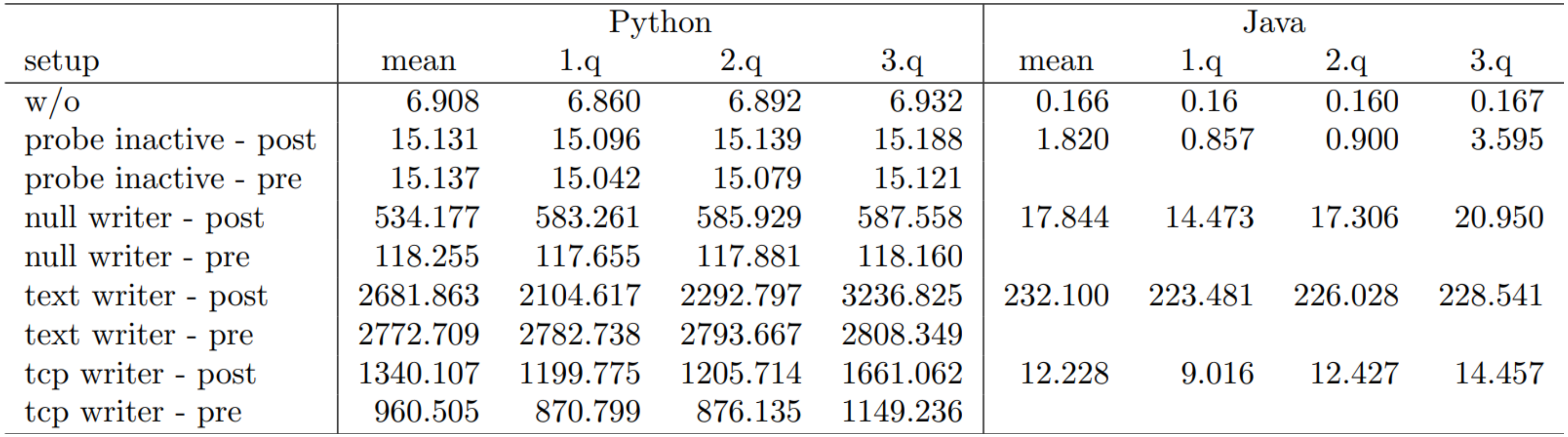
- No information loss
- No overwriting at actual runtime
- Less side effects
- C Error exception if we instrument “google.protobuf.descriptor” in Tensorflow
  - Some side effectss persist
  - Possibly hidden interaction

Case Studies

Evaluation

- Moobench
  - The same principle as in Moobench for Java
- Configurations:
  - Inactive Instrumentation
  - TCP Writer
  - File Writer
  - Null Writer
  - No Instrumentation

Evaluation Results



	Python				Java			
setup	mean	1.q	2.q	3.q	mean	1.q	2.q	3.q
w/o	6.908	6.860	6.892	6.932	0.166	0.16	0.160	0.167
probe inactive - post	15.131	15.096	15.139	15.188	1.820	0.857	0.900	3.595
probe inactive - pre	15.137	15.042	15.079	15.121				
null writer - post	534.177	583.261	585.929	587.558	17.844	14.473	17.306	20.950
null writer - pre	118.255	117.655	117.881	118.160				
text writer - post	2681.863	2104.617	2292.797	3236.825	232.100	223.481	226.028	228.541
text writer - pre	2772.709	2782.738	2793.667	2808.349				
tcp writer - post	1340.107	1199.775	1205.714	1661.062	12.228	9.016	12.427	14.457
tcp writer - pre	960.505	870.799	876.135	1149.236				

Summary

- Current State
  - The instrumentation techniques can be applied in other contexts
  - Two automatic instrumentation modes
  - Assumption: entry point of the Python program is known
- Limitations
  - Relies on „monkey patching“
  - The instrumentation can be overwritten and turned off
    - Client code overwrites `__import__()`
    - Client code changes `sys.meta_path`
  - No instrumentation of modules written in C

Future Work

- Improve efficiency
  - Writing of the records blocks the execution of the whole program
- Monitoring of `sys.meta_path`
- Instrumentation before execution
  - Requires extensive knowledge of Python grammar and return behavior
- Instrument modules written in C