Omniscient debugging with TOD

Guillaume Pothier
Pleiad Lab. - University of Chile

http://pleiad.dcc.uchile.cl/tod

The problem with debugging

For developers, debugging is **tedious**

For companies, debugging is **costly**

2002 NIST study:

"Software developers already spend approximately
80% of development costs
on identifying and correcting defects."

National Institute of Standards and Technologies,
"Software errors cost U.S. economy $59.5 billion annually", June 2002
(http://www.nist.gov/public_affairs/releases/n02-10.htm)

Why is it hard?

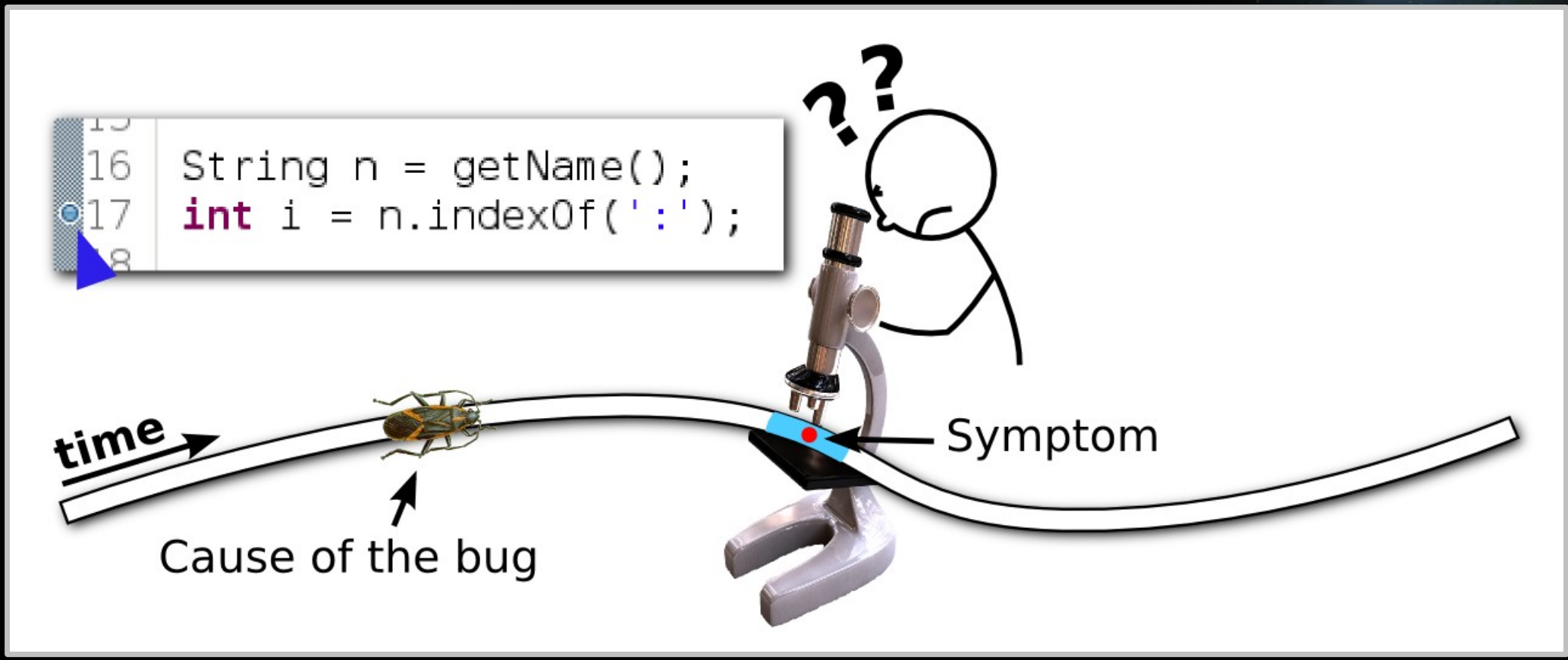
You see the **symptom** (crash, wrong result...)

You must find its **root cause**.

(which occurred before, maybe a long time ago, in a different module, a different thread... and there probably is a whole chain of errors)

Unfortunately, commonly used debugging approaches are not very helpful...

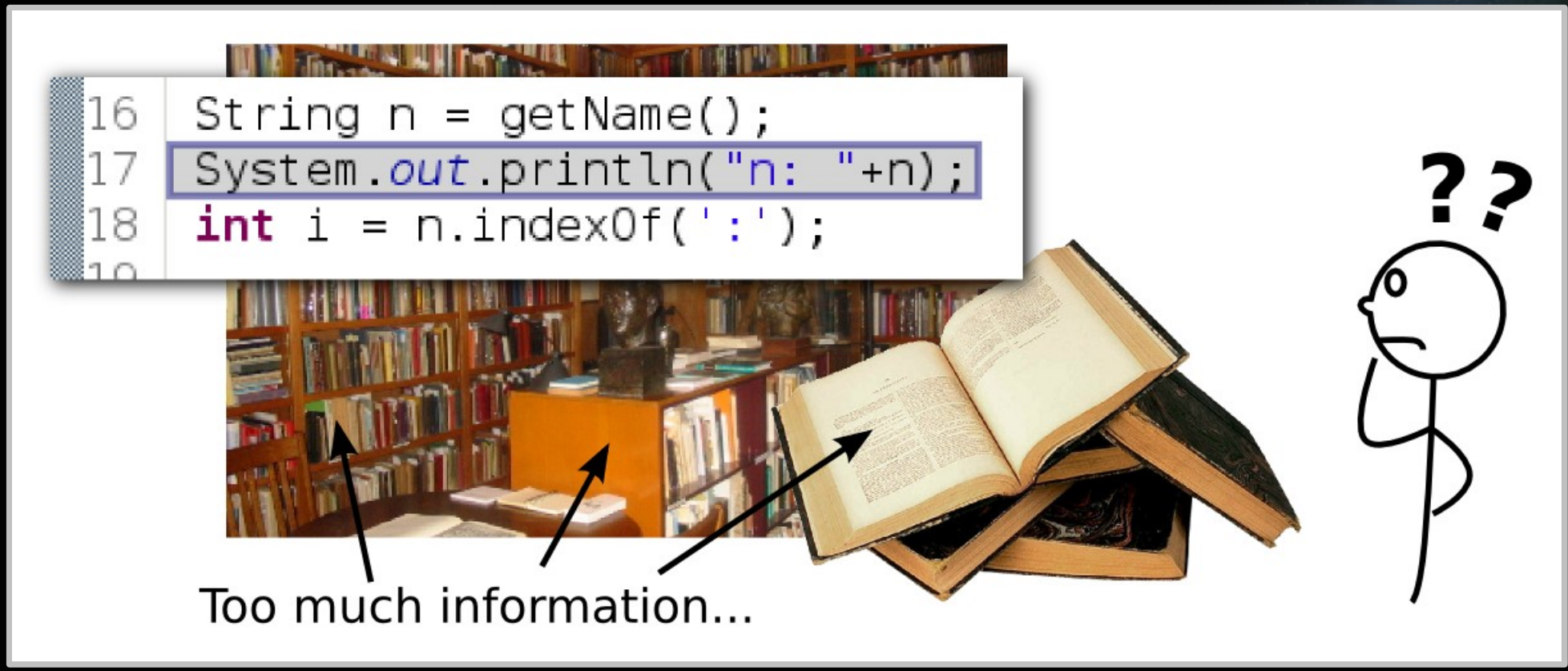
Breakpoint-based debugging



Lots of details about the state of the program at the breakpoint

But what happened before?

Log-based debugging



You *might* have the full program history
(*if* you have print statements at all potentially interesting locations)

But how do you analyze it?

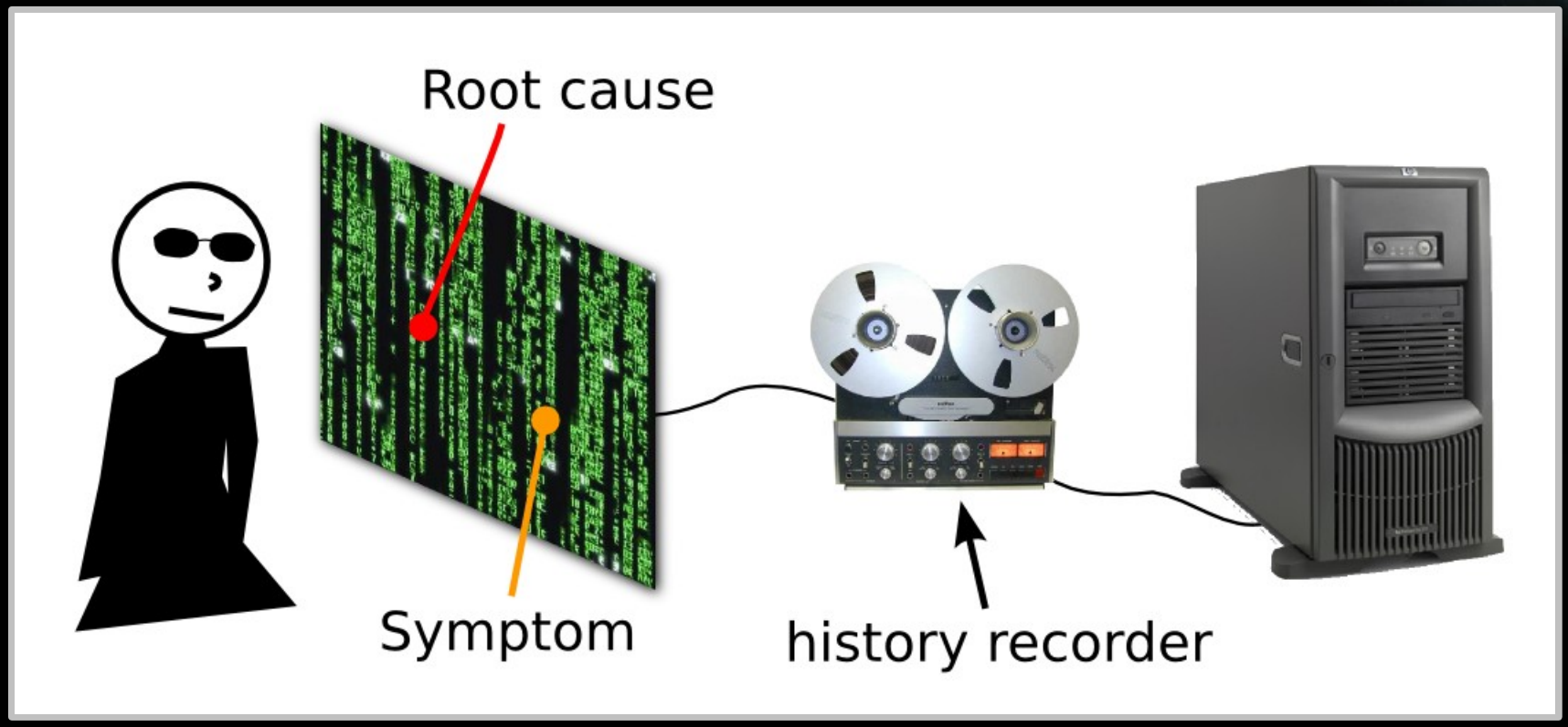
Omniscient debugging

Automatic recording
of program execution

Interactive navigation
in execution history

Instantaneous traversal
of causal links

Omniscient debugging



You know **everything**

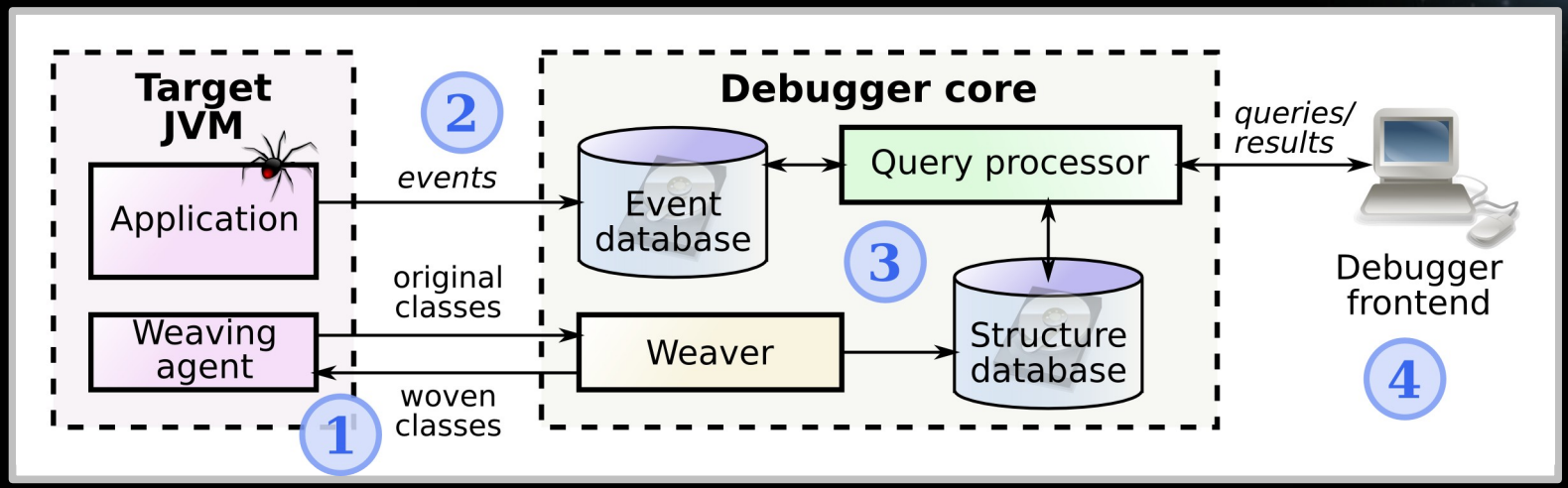
(you are the one, Neo ;-)

What is TOD?

Scalable omniscient debugger
for Java & AspectJ
(and initial support for Python)

Integrated into Eclipse

Architecture



1. At load-time, classes are instrumented.
2. At run-time, events produced by the execution of instrumented classes are sent to a database.
3. The specialized high-performance, parallelizable database stores and indexes the events.
4. The debugger front-end (Eclipse plugin) lets the programmer navigate in the execution trace.