JPF ’08
where are we - the 360°

Peter C. Mehlitz
PSGS / NASA Ames Research Center
<Peter.C.Mehlitz@nasa.gov>

Roadmap

- Day 1 : Overview and “What is being done with JPF”
  - Overview & Reflection Peter Mehlitz, PSGS/NASA Ames
  - Tales from all corners of the Realm - major infrastructure changes Peter Mehlitz, PSGS/NASA
  - JPF and Google’s Summer of Code John Penix, Google
  - Optimizing Generation of Object Graphs in JPF Sarfraz Khurshid, University of Texas

  ~

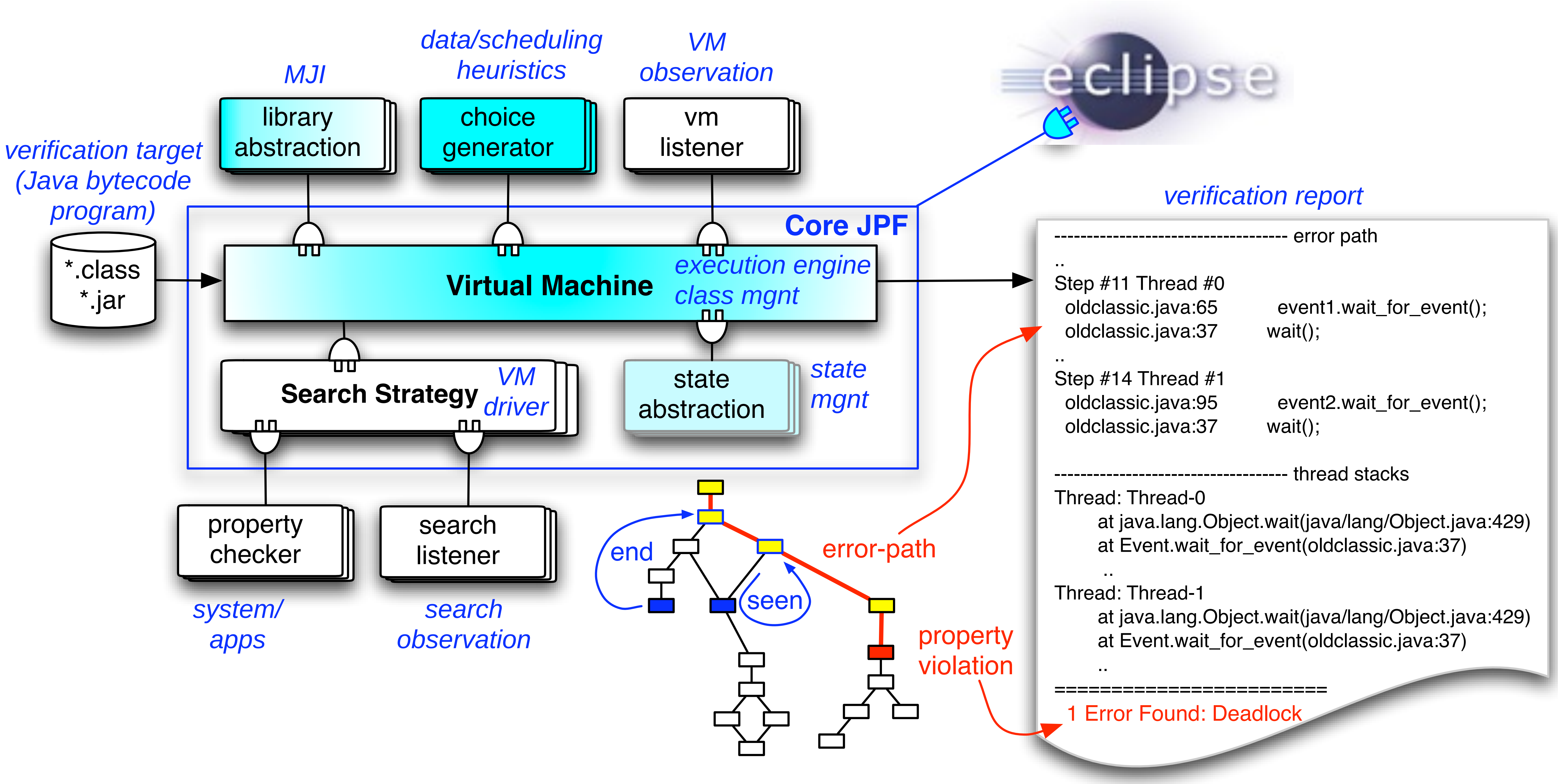
  - JPF State Extensions Darko Marinov, University of Illinois
  - Multi Agent Verification with JPF Berndt Farwer, University of Durham
  - Checking Web Applications with JPF
    - temporal logic model checking Mukul Prasad, Fujitsu
    - environment generation Oksana Tkachuk, Fujitsu
  - wrap up & plan for day 2
- Day 2: Symbolic Execution and Q&A
  - Symbolic Execution
    - using Context Sensitive Relevancy Analysis Indradeep Gosh, Fujitsu
    - Symbolic Execution of Bytecodes Corina Pasareanu, PSGS/NASA Ames
    - Symbolic Execution Engine for C Masahiro Fujita, University of Tokyo
  - Automated Test Generation and Model Checking for C++ Sarah Thompson, RIACS/NASA Ames

  ~

  - JPF Q&A / hands-on sessions
    - JPF introduction David Bushnell, RIACS/NASA Ames

JPF then..

- ✦ Open Sourced 3 years ago (04/26/2005)
- ✦ .. and the plan was: *“make it extensible”*

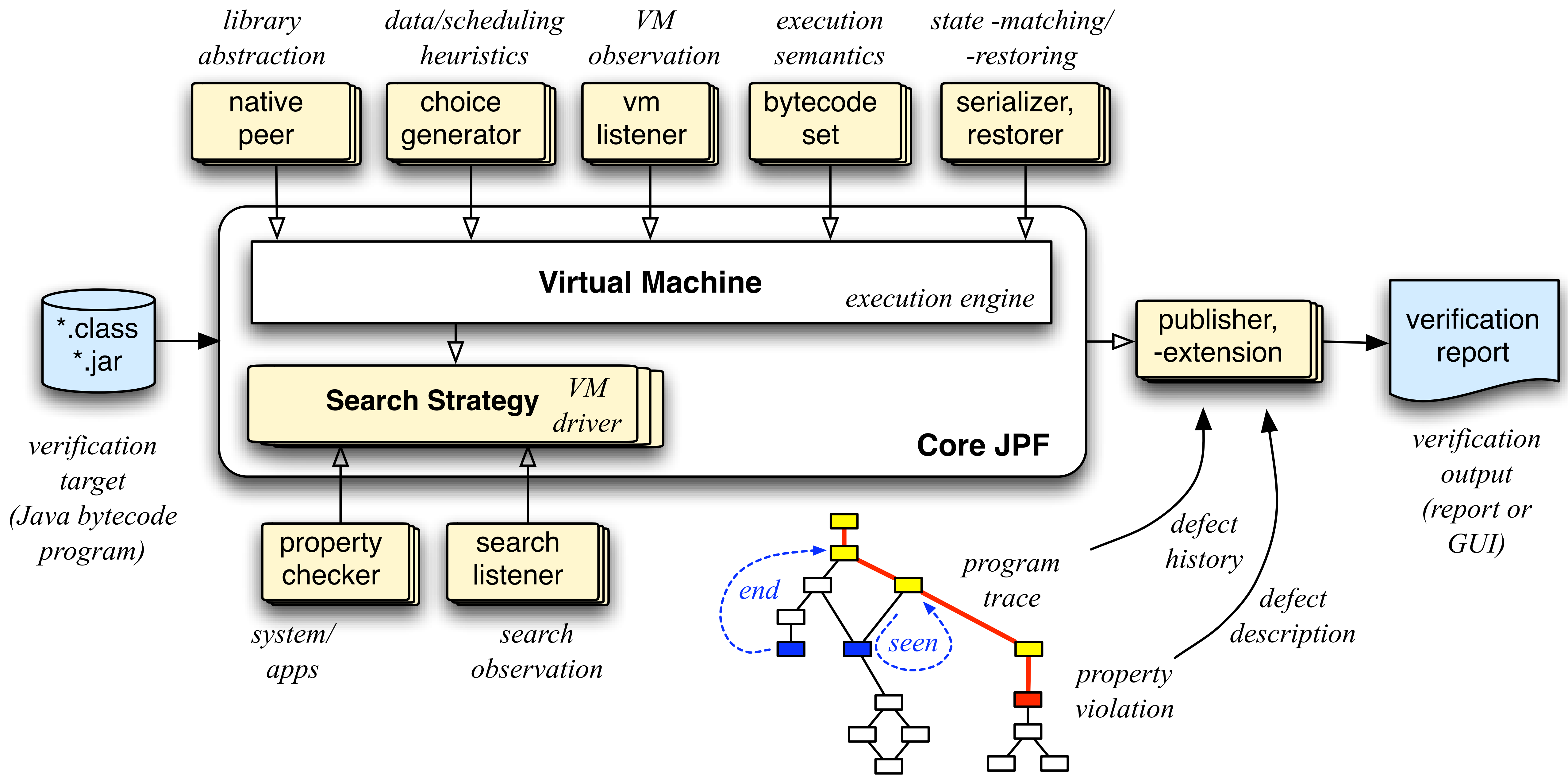


```
------------------------------------ error path
..
Step #11 Thread #0
  oldclassic.java:65         event1.wait_for_event();
  oldclassic.java:37         wait();
..
Step #14 Thread #1
  oldclassic.java:95         event2.wait_for_event();
  oldclassic.java:37         wait();

------------------------------------ thread stacks
Thread: Thread-0
        at java.lang.Object.wait(java/lang/Object.java:429)
        at Event.wait_for_event(oldclassic.java:37)
         ..
Thread: Thread-1
        at java.lang.Object.wait(java/lang/Object.java:429)
        at Event.wait_for_event(oldclassic.java:37)
        ..
========================
  1 Error Found: Deadlock
```

.. and now

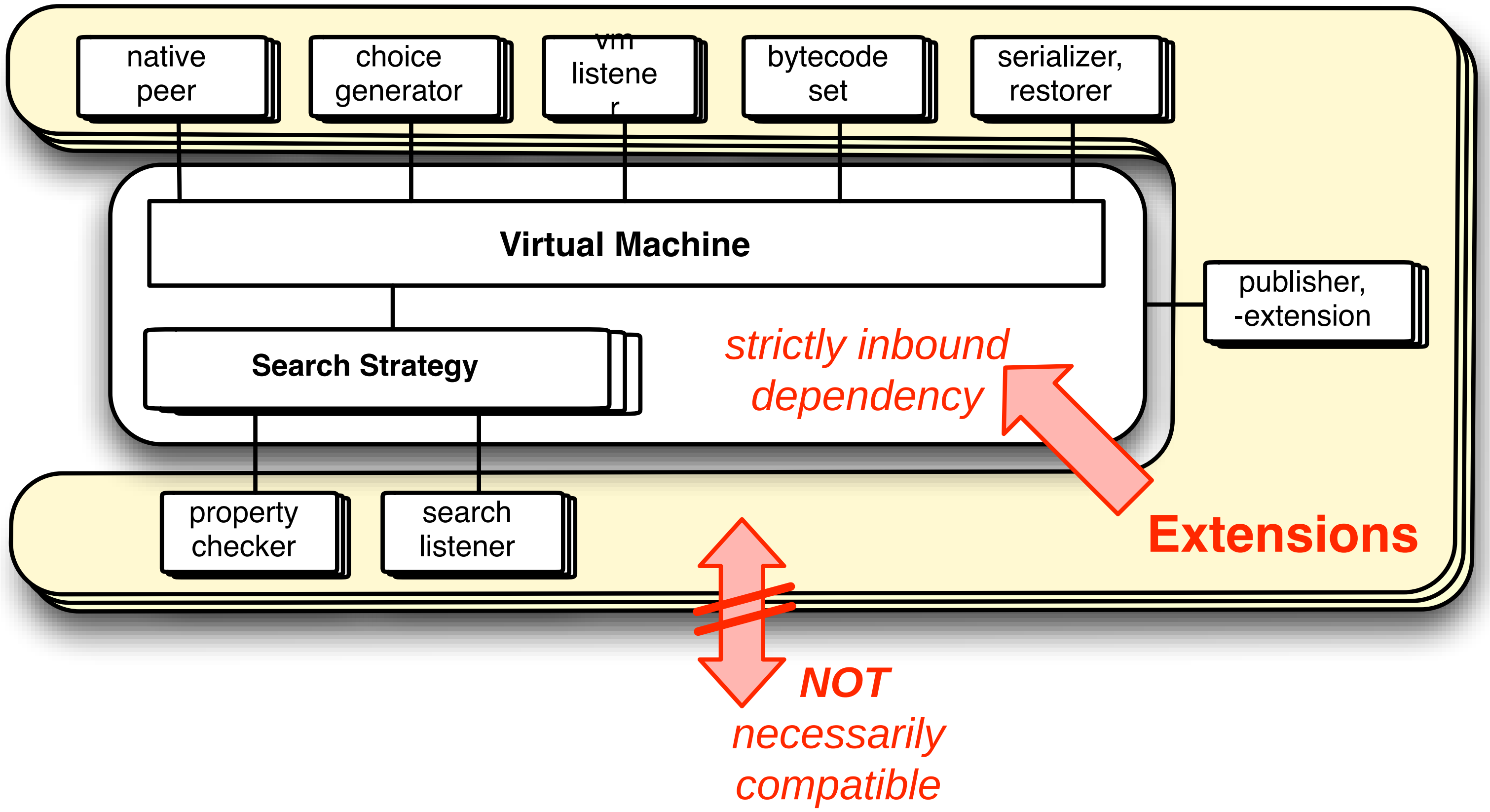
- we stayed on course!
- major extension infrastructure is in place



.. and now: **Extensions**

- mostly a packaging mechanism
- no dependency from core to specific extension allowed
- extensions might not be compatible
  (can be mutually exclusive → Eclipse)
- can even override/replace core classes
  (but shouldn't)

extensions == sandboxes



native peer

choice generator

vm listener

bytecode set

serializer, restorer

Virtual Machine

Search Strategy

publisher, -extension

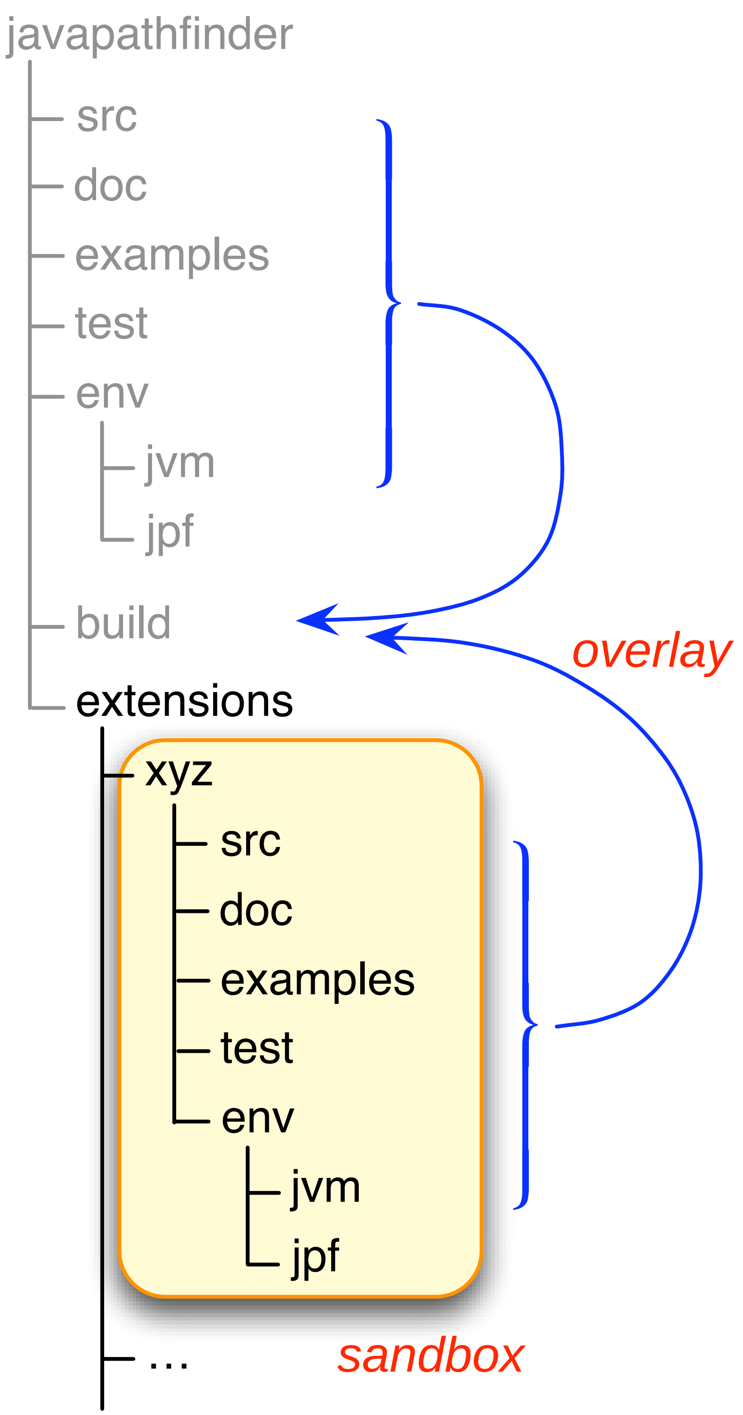
property checker

search listener

strictly inbound dependency

Extensions

NOT *necessarily compatible*



```
javapathfinder
 ├ src
 ├ doc
 ├ examples
 ├ test
 ├ env
 │  ├ jvm
 │  └ jpf
 ├ build
 └ extensions
    ├ xyz
    │  ├ src
    │  ├ doc
    │  ├ examples
    │  ├ test
    │  └ env
    │     ├ jvm
    │     └ jpf
    ├ …
```

overlay

sandbox

JPF then and now: Statistics

- 04/26/2005

	src	ext	env	test	Σ
files	313	34	32	33	453
loc	19546	4301	1549	2145	29310
classes	308	37	33	47	481

52912 w/ cmt
1.8

22 packages

- 04/08/2008

	src	ext	env	test	Σ
files	539	458	90	127	1395
loc	45702 2.3	24865 5.8	5276 3.4	11498 5.4	102308 3.5
classes	617	515	93	192	1634

163991 w/ cmt
1.6

91 packages

JPF then and now: Statistics

- 04/26/2005

	src	ext	env	test	Σ	
files	313	34	32	33	453	
loc	19546	4301	1549	2145	29310	52912 w/ cmt 1.8
classes	308	37	33	47	481	22 packages

☹ maintainable?

- 04/08/2008

	src	ext	env	test	Σ	
files	539	458	90	127	1395	
loc	45702 2.3	24865 5.8	5276 3.4	11498 5.4	102308 3.5	163991 w/ cmt 1.6
classes	617	515	93	192	1634	91 packages

JPF then and now: Statistics

- 04/26/2005

	src	ext	env	test	Σ
files	313	34	32	33	453
loc	19546	4301	1549	2145	29310
classes	308	37	33	47	481

52912 w/ cmt
1.8

22 packages

- 04/08/2008

	src	ext	env	test	Σ
files	539	458	90	127	1395
loc	45702 2.3	24865 5.8	5276 3.4	11498 5.4	102308 3.5
classes	617	515	93	192	1634

163991 w/ cmt
1.6

91 packages

JPF then and now: Statistics

- 04/26/2005

	src	ext	env	test	Σ
files	313	34	32	33	453
loc	19546	4301	1549	2145	29310
classes	308	37	33	47	481

52912 w/ cmt
1.8

22 packages

☺ core stable

- 04/08/2008

	src	ext	env	test	Σ
files	539	458	90	127	1395
loc	45702 2.3	24865 5.8	5276 3.4	11498 5.4	102308 3.5
classes	617	515	93	192	1634

163991 w/ cmt
1.6

91 packages

JPF then and now: Statistics

- 04/26/2005

	src	ext	env	test	Σ
files	313	34	32	33	453
loc	19546	4301	1549	2145	29310
classes	308	37	33	47	481

52912 w/ cmt
1.8

22 packages

- 04/08/2008

	src	ext	env	test	Σ
files	539	458	90	127	1395
loc	45702 2.3	24865 5.8	5276 3.4	11498 5.4	102308 3.5
classes	617	515	93	192	1634

163991 w/ cmt
1.6

91 packages

JPF then and now: Statistics

- 04/26/2005

	src	ext	env	test	Σ
files	313	34	32	33	453
loc	19546	4301	1549	2145	29310
classes	308	37	33	47	481

52912 w/ cmt
1.8

22 packages

☺ sandboxes ☺ maturation

- 04/08/2008

	src	ext	env	test	Σ
files	539	458	90	127	1395
loc	45702 2.3	24865 5.8	5276 3.4	11498 5.4	102308 3.5
classes	617	515	93	192	1634

163991 w/ cmt
1.6

91 packages

Successful Open Sourcing

- serve a real purpose
  - “coolness” helps, but mostly for individual contributors
- get trusted
  - a matter of license & organization
- provide playgrounds
  - design so that people don’t step on each others toes
- be responsive
  - is the project is still alive ?
- give direction
  - steering committee
  - prevent ramification, forks

Successful Open Sourcing

- serve a real purpose
  - “coolness” helps, but mostly for individual contributors
- get trusted
  - a matter of license & organization
- provide playgrounds
  - design so that people don’t step on each others toes
- be responsive
  - is the project is still alive ?
- give direction
  - steering committee
  - prevent ramification, forks

How to *not* get lost

- know what is done where and for what reason: two perspectives
  - implementation centric: **JPF design**
    - Ok for research, developers
    - not suitable for users
  - shift focus from *How* to *Why*: **JPF application types**
    - application centric
- does JPF assume Java as modeling or production language?
  - flipped forth and back for a long time
  - now it’s clear: we do both (and more)

JPF Application Types

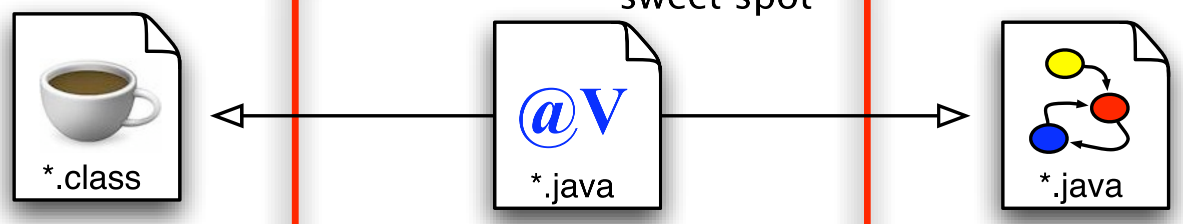
JPF unaware programs

JPF aware programs

JPF dependent programs

"sweet spot"

runs on any JVM



runs only under JPF

annotate program
- requirements
- sequences (UML)
- contracts (PbC)
- tests
- …

analyze program
- symbolic exec
  → test data
- thread safety / races

constraints

runtime costs
- order of magnitude slower
- state storage memory

standard library support
- java.net, javax.swing, .. (needs abstraction models)

functional property impl. costs
- listeners, MJI knowledge

restricted choice types
- scheduling sequences
- java.util.Random

restricted application models
- UML statemachines
- does not run w/o JPF libraries

initial domain impl. costs
- domain libs can be tricky

benefits

non-functional properties
- unhandled exceptions (incl. AssertionError)
- deadlocks
- races

improved inspection
- coverage statistics
- exact object counts
- execution costs

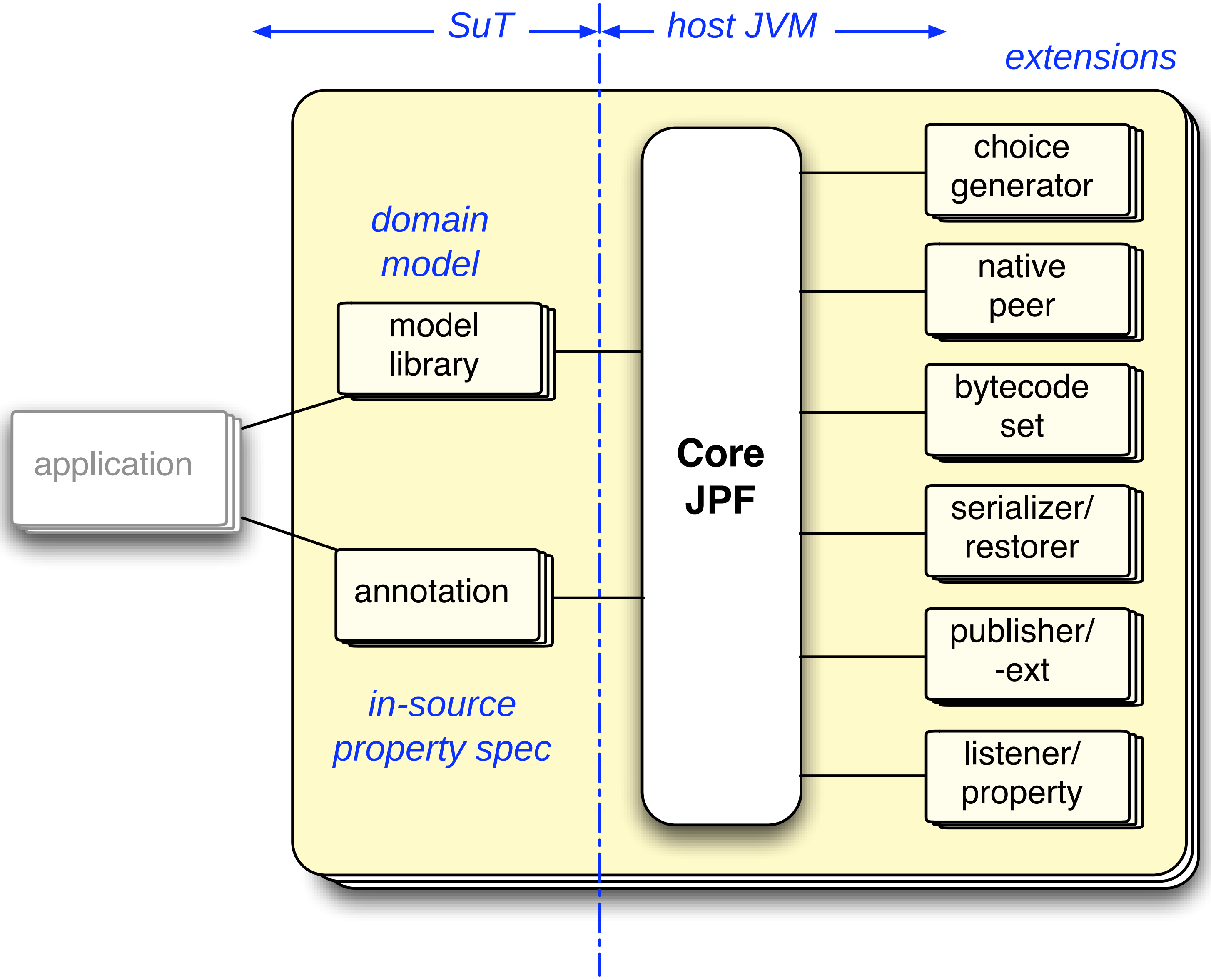
low modeling costs
- statemachine w/o layout hassle,..

functional (domain) properties
- built-in into JPF libraries

flexible state space
- domain specific choices (e.g. UML "enabling events")

runtime costs & library support
- usually not a problem, domain libs can control state space

JPF Components (regrouped)


SuT
host JVM
extensions
domain model
model library
application
annotation
in-source property spec
Core JPF
choice generator
native peer
bytecode set
serializer/ restorer
publisher/ -ext
listener/ property