Open Source Performance Testing

tools and ideas for performance testing

OneDayTalk 1. October 2010
15:00 - 15:45

Volker Bergmann

Targets of the talk

- NOT an introduction to performance testing in general
- BUT presenting you useful testing and measurement approaches
- Presenting small, handy and proven open source tools
- Focus on development and developers
- Increasing awareness of sometimes neglected topics
- (Very) brief introductions

Agenda

- If and how to test performance
- Setting up test data — *Benerator, Jailer, Talend*
- Continuous Performance Testing — *JUnitPerf, ContiPerf*
- Monitoring & Profiling — *log4jdbc, VisualVM, perf4j*
- (Load Generation) — *(JMeter, Grinder)*

Have you seen this guy?

- 13 years of professional software development with Java
- numerous large scale projects
- analysis, design, implementation, testing
- Striving to assure software quality in early project stages
- Performance focus esp. for J2EE server software
- Development of open source test tools
- His name: Volker Bergmann

If and how to test performance

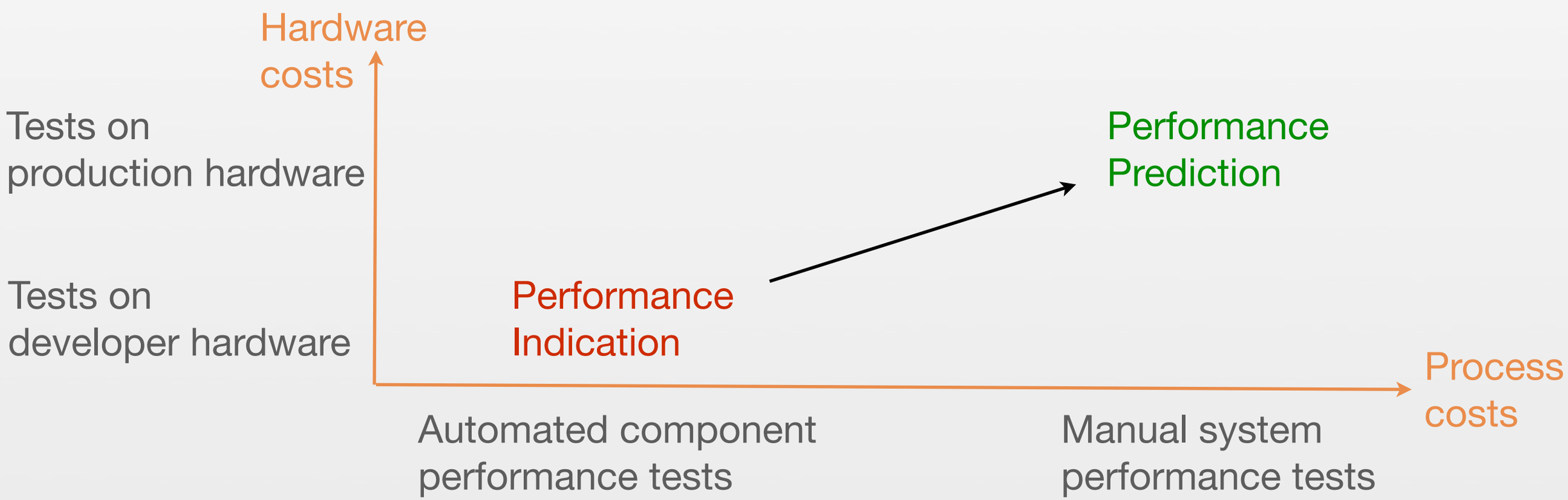
About the performance testing process and tool selection

We‘ll do it later, ...maybe

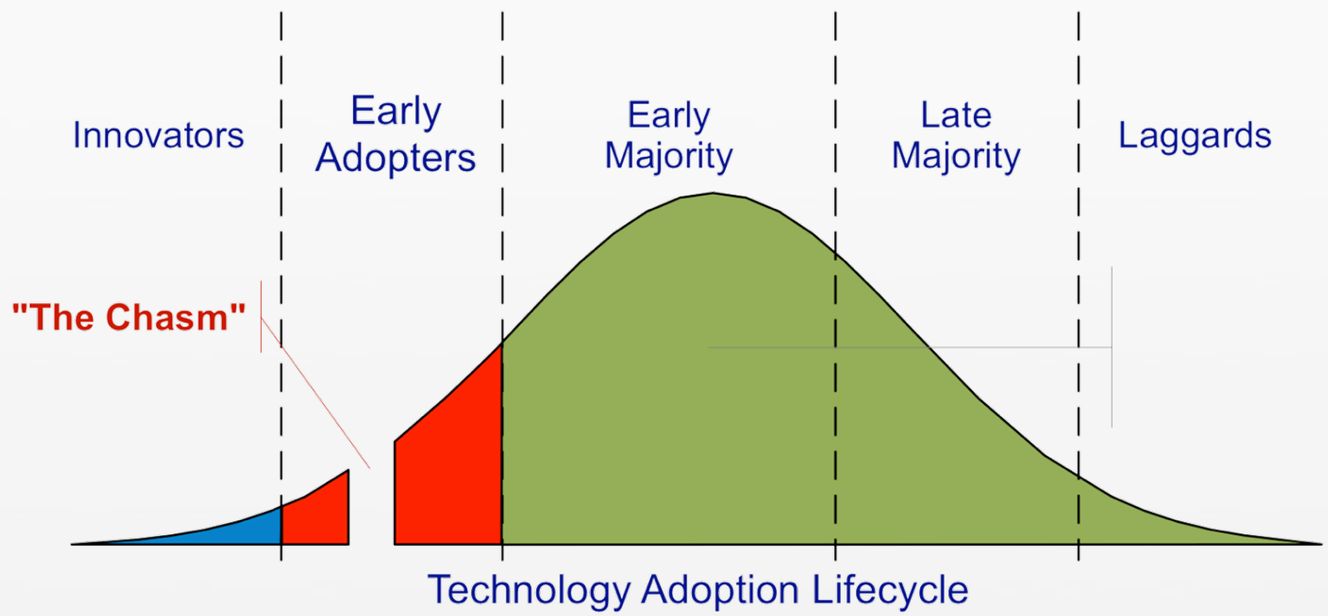
- Do you really need performance testing?
- No - until it is to late!
- How embarrassing / expensive will it be when it is too late?
  - Extra project costs
  - image damage
  - missing revenue?
- You might be able to cope with this failure

Scaling your test effort

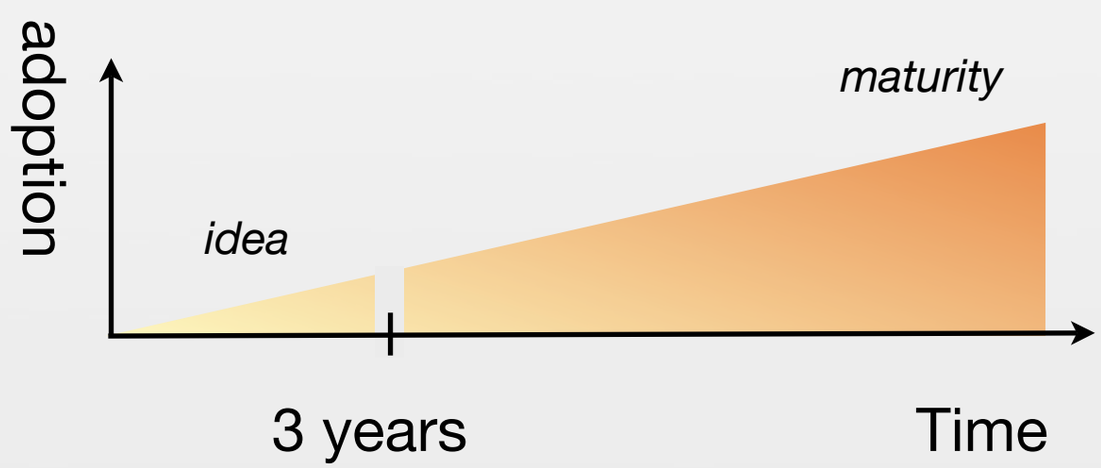
- Regard performance testing like an **insurance**:
  - **spending** a **small effort**
  - for **reducing the risk** of having a **large damage**
- Test at least the ‚risky' parts of your application



Use mature tools and frameworks



Crossing Moore‘s chasm



Test tool fitness

- How useful is the tool for you?

Usefulness

- Maturity
- Fitness for the task (shooting flies with cannons? shooting elephants with peas?
- Extendibility

Cost

- License cost

Risk

- Learning curve (lost work time)
- Training necessity and cost
- Productivity (lost work time)
- Probability that bugs get fixed (open source? dormant/discontinued project?)

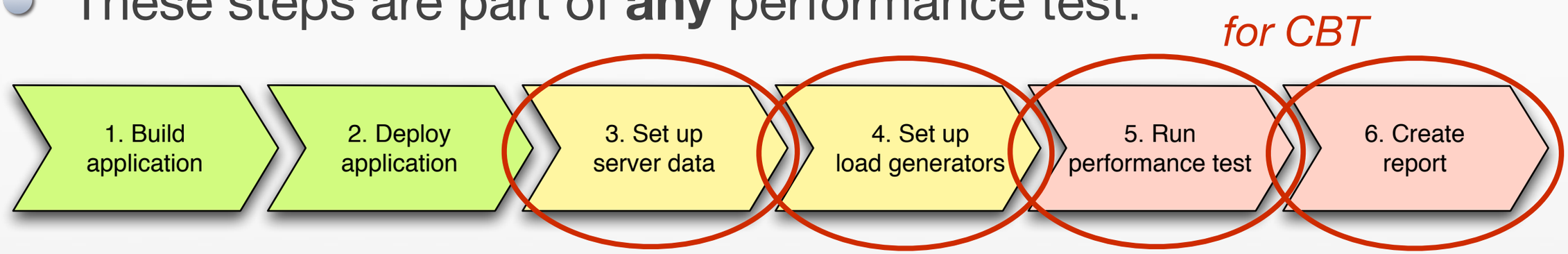
- **A ‚cheaper' product may be more expensive in the long run!**

Predicting Production Performance

- Manual task with high requirements
  - Production-like hardware
  - Production-like data
  - Production-like client behavior
  - Resource and application monitoring for all related systems: CPU, memory, disk I/O, network traffic, VM Heap, Garbage Collections, database load, database query times
    --> Monitoring
- **--> Interactive Process!**
- **--> Requires expertise, not a tool!**

Performance Testing Process

- These steps are part of **any** performance test:

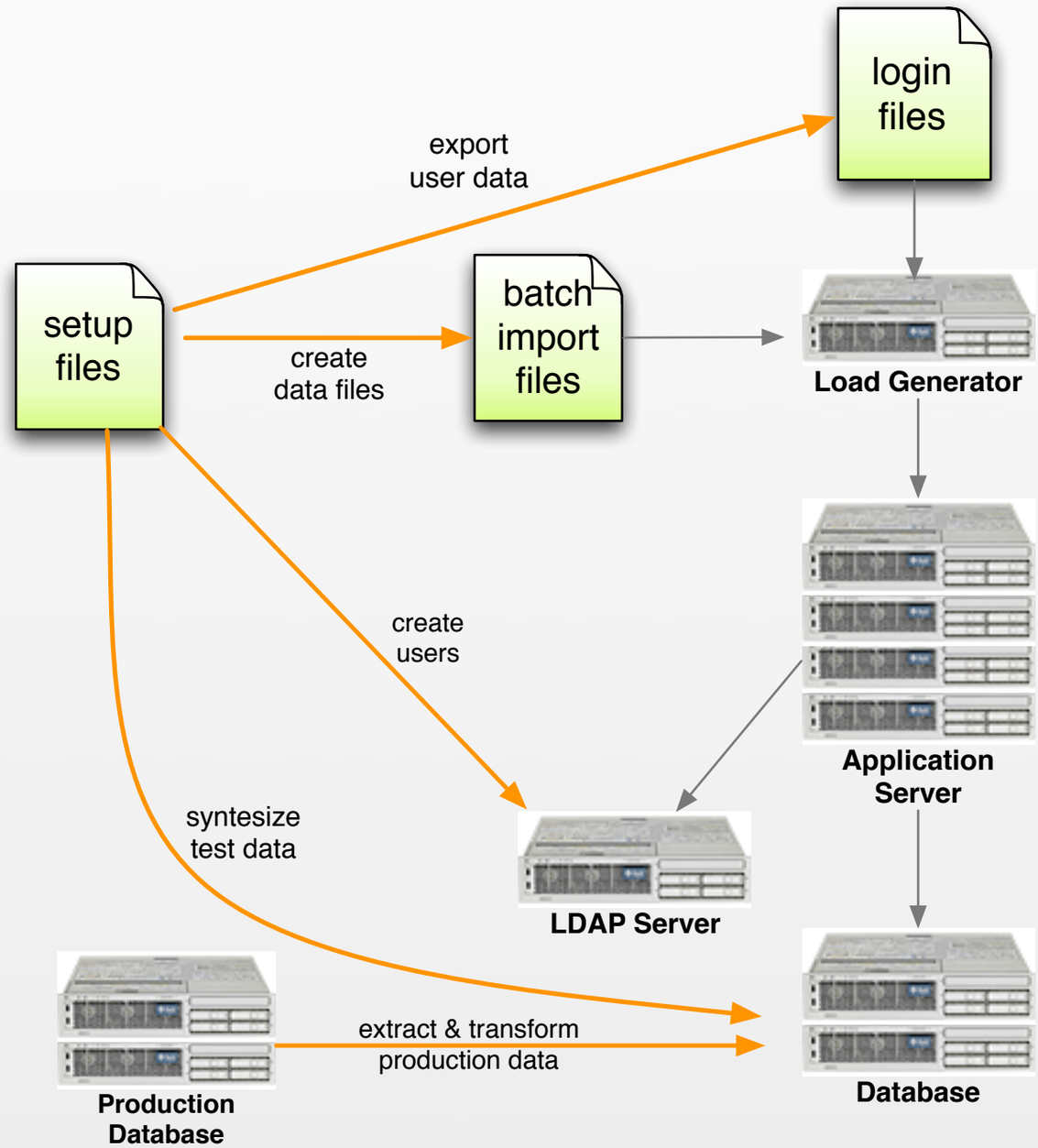


- **Continuous Performance Testing** (CBT) aims at automating performance test process execution
- Steps 1 and 2 are the classic domain of **Continuous integration** tools (like Hudson), steps 3 to 6 may be integrated
- Step 3 is **often neglected**
- Steps 4 to 6 require **different tools** for CBT or manual performance testing

Setting up test data

Preparing data for the tested system
and the load generators

Data related test preparations

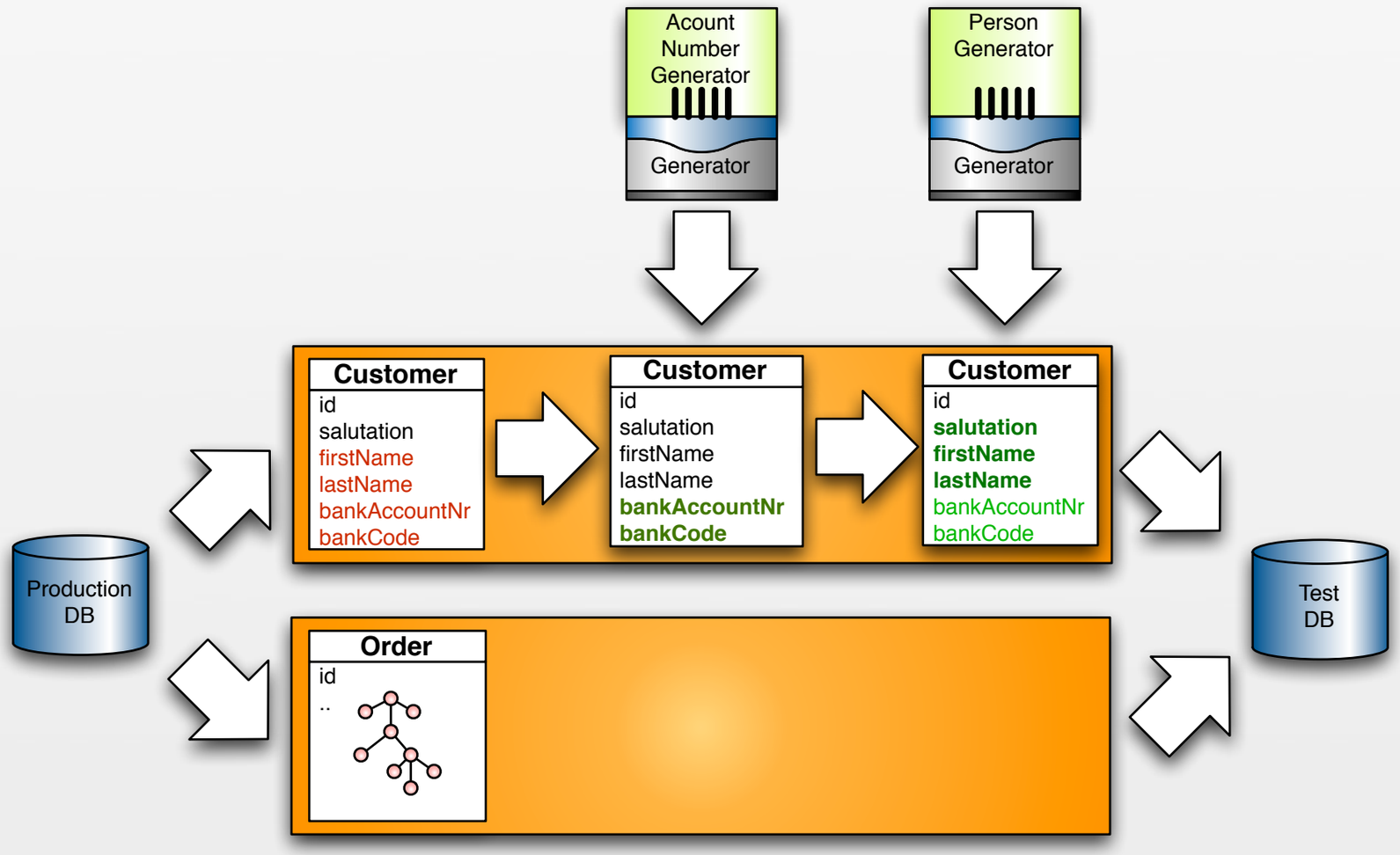


Use Production Data?

- Is production data
  - existing/available?
  - anonymized?
  - applicable?
  - good enough?
- Anyway you need to
  - check validity
  - extract subsets
  - generate additional data

Anonymization

- Data might be transferred 1:1
- Only confidential and personal data needs to be anonymized



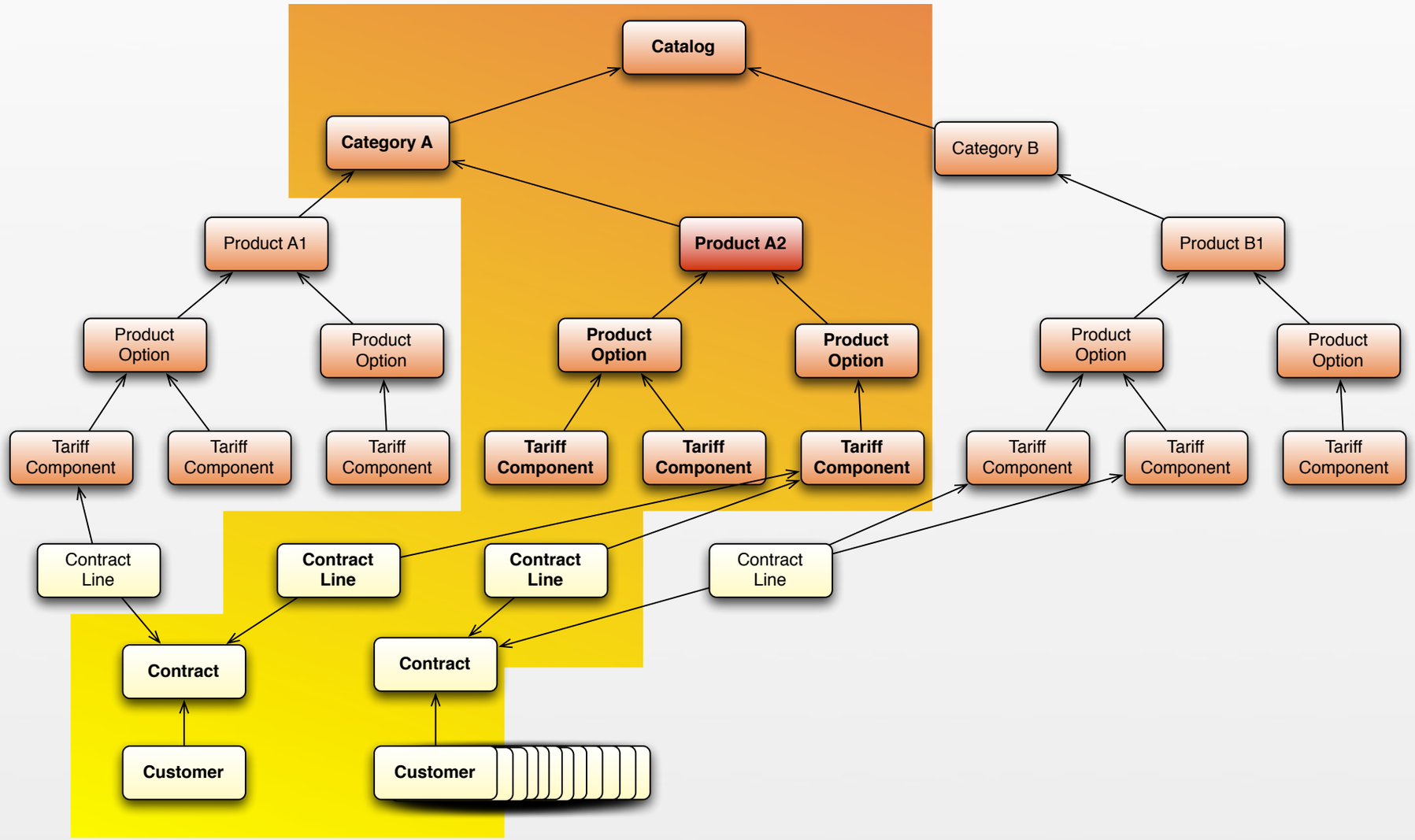
Anonymization with Benerator

- Generator classes create composite consistent data graphs (e.g. BankAccountGenerator creates BankAccount --> Bank)
- Graph parts can be mapped to database columns

```
<database id="prod" readOnly="true" ... />
<database id="test" ... />

<iterate source="prod" type="CUSTOMER" consumer="test">
    <variable name="acct" generator="BankAccountGenerator"/>
    <attribute name="BANK_ACCOUNT_NR" script="acct.accountNumber" />
    <attribute name="BANK_CODE" script="acct.bank.bankCode" />
</iterate>

<iterate source="prod" type="ORDER" consumer="test" />
```

Database subsetting

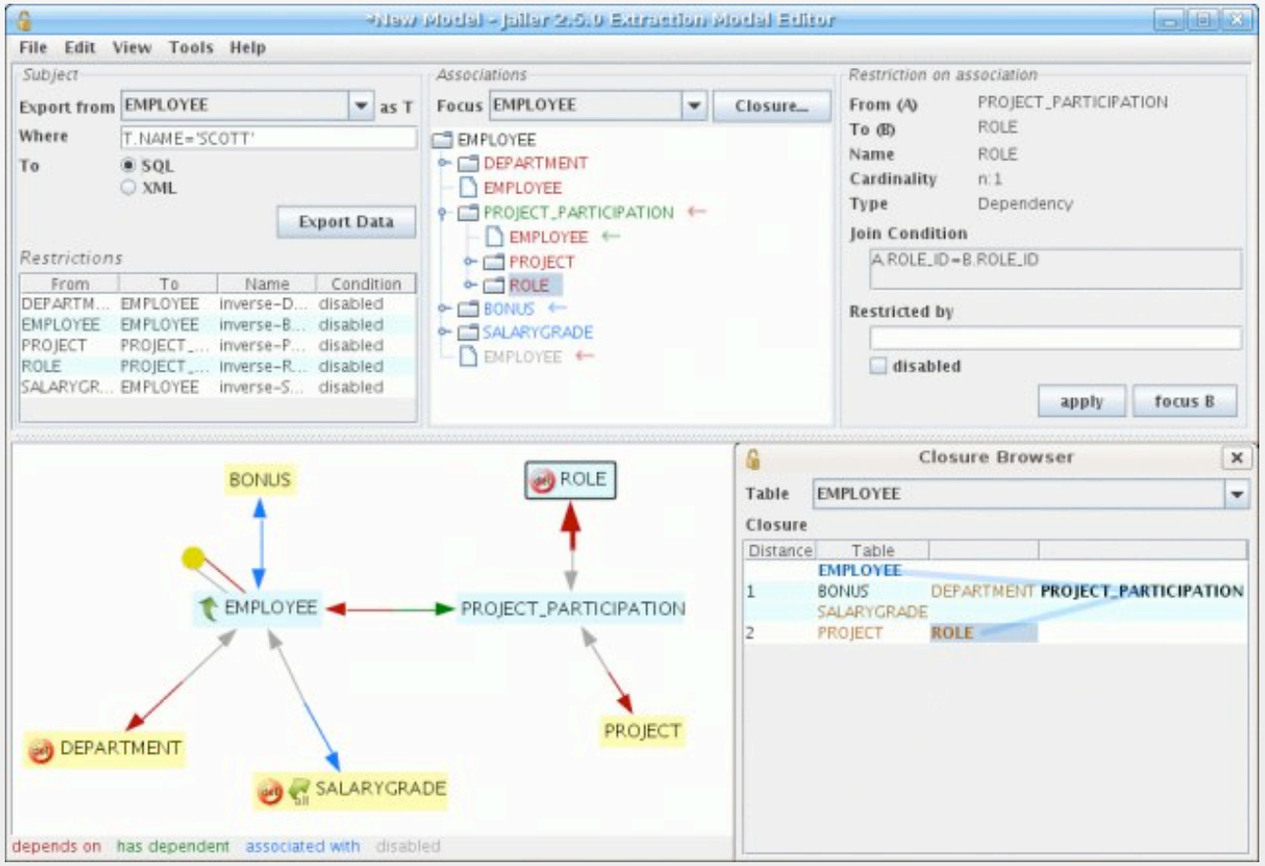
- If you want to restrict the products to ‚Product A2'...



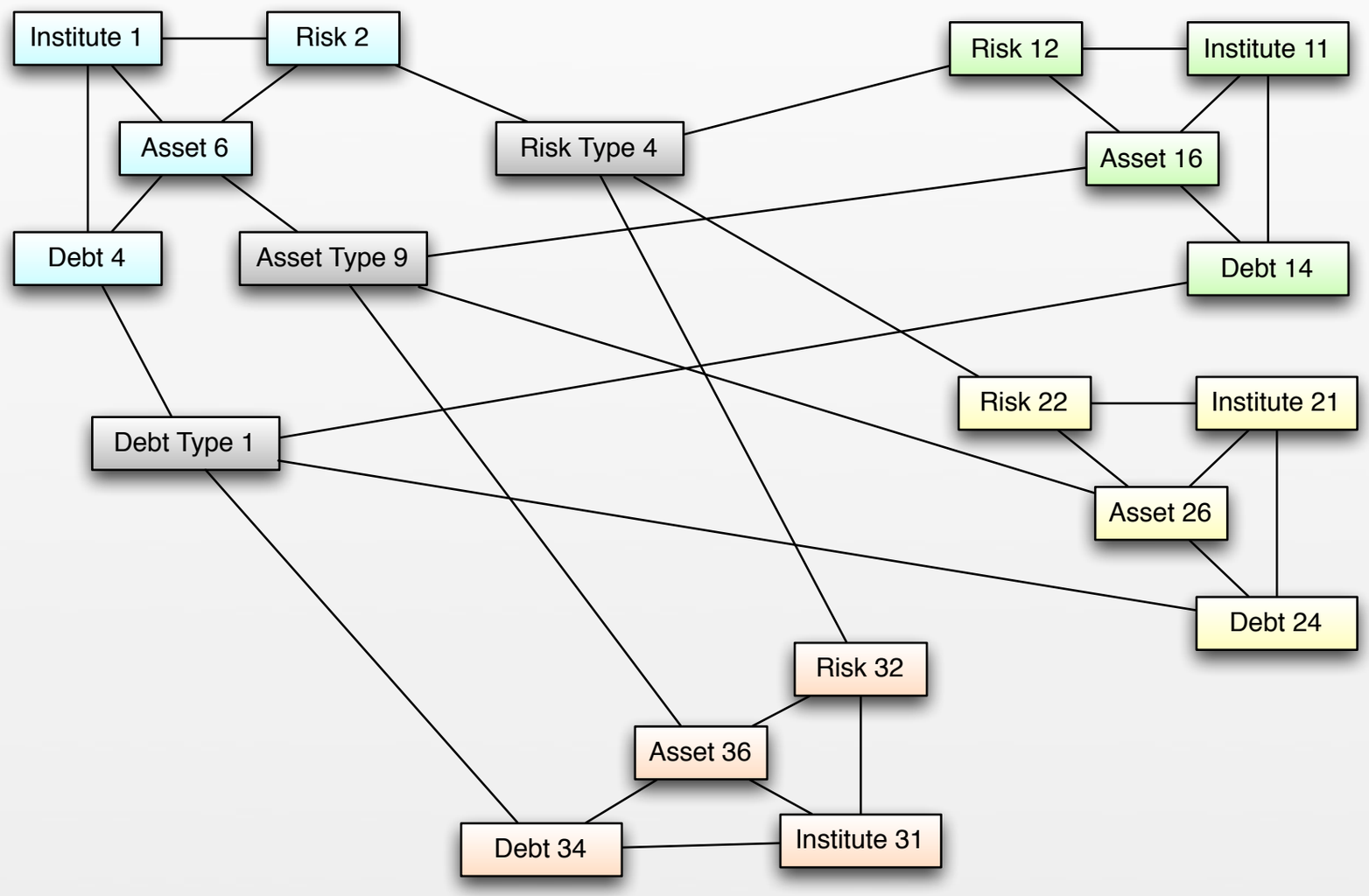
- ...how to keep referential integrity?

Jailer

- Database subsetting tool
- Active, current version 3.4.7 (2010)
- Extracts database subsets with referential integrity
- implicit/explicit foreign keys
- composition/reference
- Export in
  - SQL
  - XML
  - DbUnit



Data Replication



Data Generation

- Defining data synthetically
  - ETL features: extract and transform data from databases or files
  - Generator features: Generating random data
- Core issue: Matching explicit and implicit constraints in
  - database
  - application
- For most performance tests, at least a part of data must be generated
- Experience: Typically stupid random data of little usefulness
- until... Benerator was there

Data Generation with Benerator

Define a database ‚db‘ -->

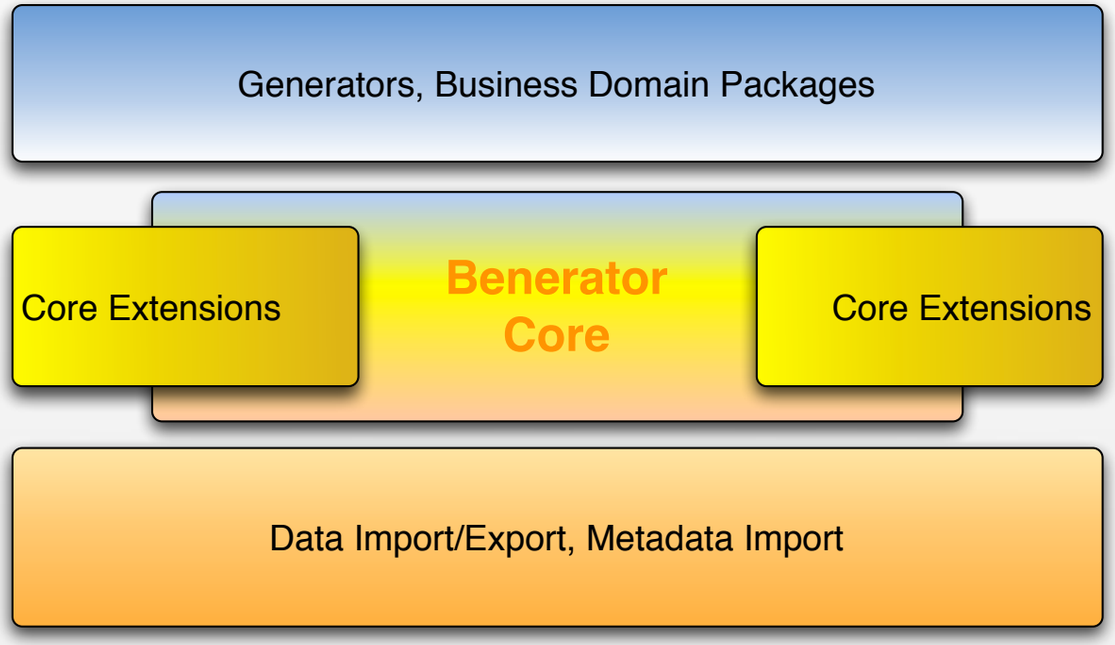
execute DDL/SQL scripts -->

import data from DbUnit files -->

generate valid data -->

```
<database id="db" ... />

<execute uri="create_tables.sql" target="db" />

<iterate source="products.dbunit.xml" consumer="db" />

<generate type="db_order" consumer="db">
  <id name="id"
    generator="new DBSequenceGenerator('SEQ_ORDER',db)"/>
  <reference name="customer" targetType="db_customer"
    distribution="random" />
  <attribute name="created_at"
    generator="CurrentDateGenerator" />
  <attribute name="created_by" script="this.customer" />
</generate>
```

Benerator Architecture



person, address, finance, net, ...

generate, anonymize, replicate

database, LDAP, JCR, CSV, XML

Data related tools

	Benerator	Jailer	Talend Open Studio
Anonymization	✓	✗	✓
Database Subsetting	✗	✓	✗
Replication	✓	✗	✓
Data Generation	✓	✗	☹

Continuous Performance Testing

Targets, process and tools

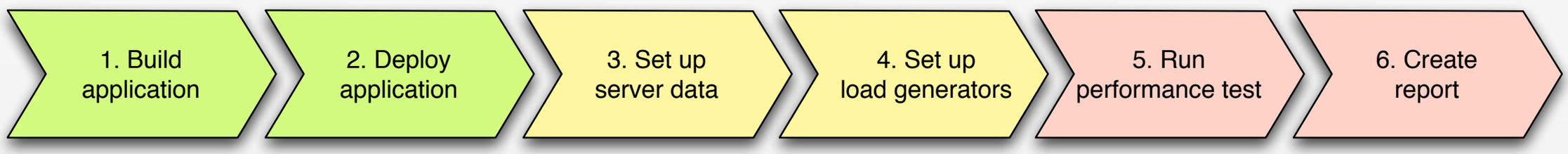
Continuous Performance Testing

- Does **not** replace a carefully monitored manual performance test
- Usually not aimed at predicting production performance, but at realizing **performance trends** over development cycles.
- **Nightly builds** provide immediate feedback when a change impacts performance
- Excellent for performance assurance of **stable applications**
- The less stable your application interface is, the more expensive. It may become a ***Continuous Reengineering***:
  - Client code (Interface + Data constraints)
  - Data (Client side and server side)

Continuous Performance Testing

Idea: automated (nightly?) performance testing

Controller: Hudson, CruiseControl, Cron Job



Maven Ant	SQL Maven Plugin Ant Plugin	Benerator Jailer Open Studio	Benerator Open Studio	JUnitPerf ContiPerf	Maven Surefire report XSLT -> HTML

JUnitPerf

- Integrates with the JUnit Test Framework
- Mechanism to create JUnit test suites with multithreaded execution and time limit
- TimedTest

```
public static Test suite() {
    long maxElapsedTime = 1000;
    Test testCase = new ExampleTestCase("testExecution");
    return new TimedTest(testCase, maxElapsedTime);
}
```

- LoadTest

```
public static Test suite() {
    int maxUsers = 10;
    long maxElapsedTime = 1500;
    Test testCase = new ExampleTestCase("testExecution");
    Test loadTest = new LoadTest(testCase, maxUsers);
    return new TimedTest(loadTest, maxElapsedTime);
}
```

JUnitPerf

- Cons:
  - Made for JUnit 3
  - needs a special runner to work with JUnit4
  - Unnecessary programmatical ‚configuration‘:

```
public static Test suite() {
    int maxUsers = 10;
    long maxElapsedTime = 1500;
    Test testCase = new ExampleTestCase("testOneSecondResponse");
    Test loadTest = new LoadTest(testCase, maxUsers);
    return new TimedTest(loadTest, maxElapsedTime);
}
```

  - ...and dormant since 2008
  - But there is something new:...

ContiPerf

- Time measurement and evaluation library
- Integrates with JUnit 4 like JUnitPerf with JUnit 3
- Uses Java annotations
- Stable release
- Active development
- Can be used standalone for running and evaluating tests
  - Invoking code with given concurrency and time characteristics
  - Measuring execution times
  - Calculating averages, percentiles, max, ...
  - Reporting failure if imposed requirements are violated (throughput, max, average, percentiles)

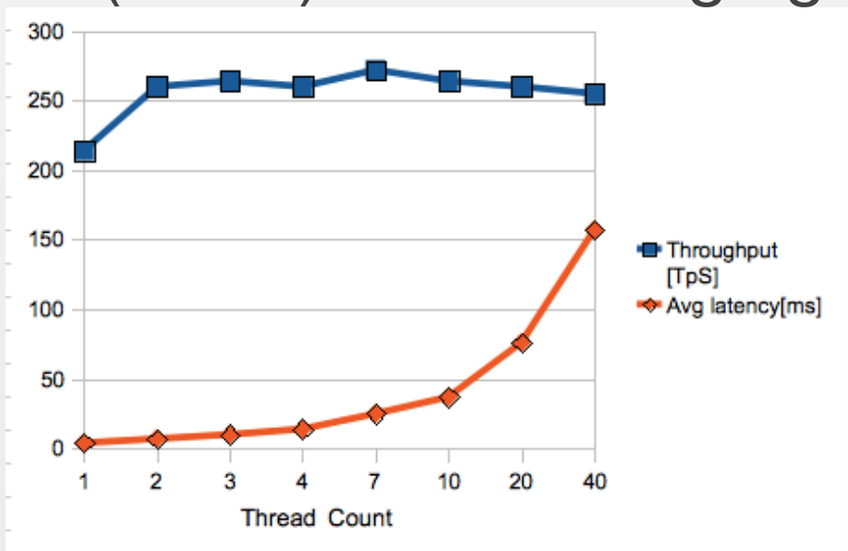
ContiPerf example

```
public class SampleTest {

    @Rule public ContiPerfRule rule = new ContiPerfRule();

    @Test
    @PerfTest(duration = 3600000, threads = 20)
    @Required(throughput = 40, max = 1200, percentile90 = 220)
    public void test() throws Exception {
        ...
    }
}
```

- The sample code invokes the test() method for one hour with 20 concurrent threads. It will report test failure if
  - an exception occurs
  - the achieved throughput is below 40 invocations per second
  - the maximum execution time is more than 1200 ms
  - more than 90% of the invocations take more than 220 ms

Pitfalls in CPT

- Don‘t be too simplistic:
  - Have your database filled with volumes comparable to production
  - Vary your request data
  - Use a reasonable concurrency for testing throughput
  - Use high concurrency for finding locks and race conditions
  - Use long test runs (>= 2h) for checking against memory leaks



Monitoring and Profiling

Where to look at with which tool

Focus your attention

- For web apps, 90% of performance issues result from **logging**, **O/R Mapping** and **database**, (given that you are using mature frameworks)
- Learn about monitoring features of your
  - O/R mapper (Hibernate: Statistics MBean)
  - database
  - application server

log4jdbc

- JDBC driver proxy
- Logs SQL and/or JDBC calls
- resolves prepared statement params
- timing feature
- JDBC config:
  - **Driver: net.sf.log4jdbc.DriverSpy**
  - **URL: jdbc:log4jdbc:hsqldb:hsql://localhost/hsql/relative**
- Output format:

```
... INFO  [jdbc.sqltiming] insert into Person (first_name, family_name, gender, id)
values ('Hans', 'Elstner', 'm')  {executed in 5 msec}
```

```
<category name="jdbc">
   <priority value="error"/>
</category>

<category name="jdbc.sqltiming">
   <priority value="info"/>
</category>
```

JVM Profiling

- **VisualVM** (free, extendible)
- Eclipse TPTP (open source)
- **NetBeans Profiler** (open source)
- JBoss Profiler (open source, 2.0 beta since 2008)
- **JProfiler**, **YourKit** ($$$)
- JProbe Enterprise ($$,$$$)
- dynatrace ($$,$$$)

Usefulness (~ Price)

VisualVM

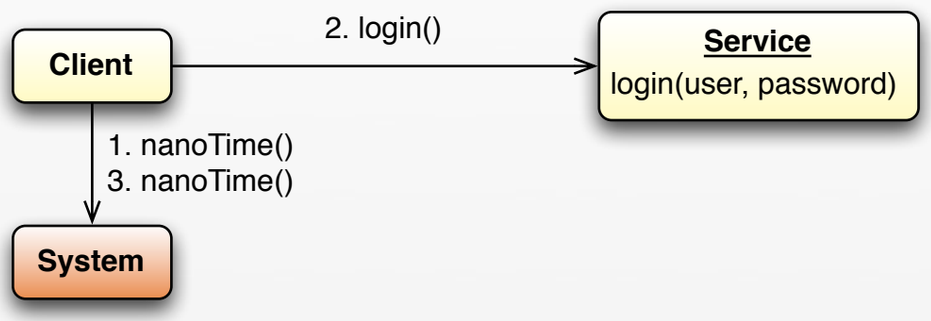
- Comes with the JDK since 1.6_07
- Invoke `jvisualvm`
- Profiling features: Heap Memory, CPU, Garbage Collector stats
- Extendible
- Plugins: JMX Monitoring, GC analyis, Thread Dump Analyzer

Selfmade Monitoring

- In mature applications there individual weaknesses which are known with time
- Often they deserve the insertion of measurement points in the application code.
- Production-readyness, low overhead
- runtime activation/deactivation
- Applied by
  - bytecode instrumentation (aspects, dynatrace)
  - dynamic proxies (e.g. PreparedStatementLogger from jdbacl)
  - explicit code

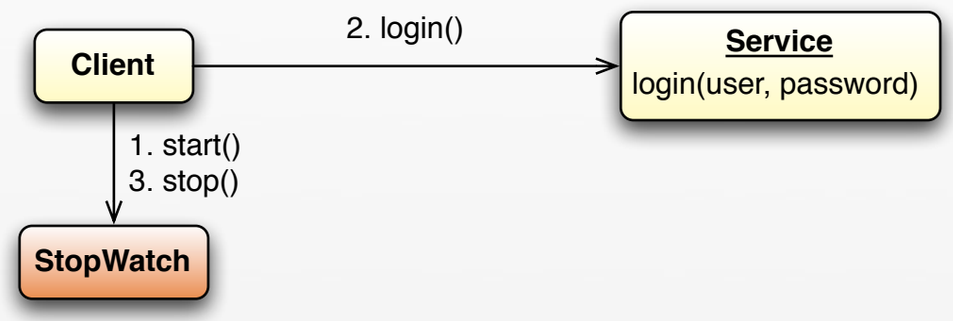
Monitoring Approaches

- Explicit

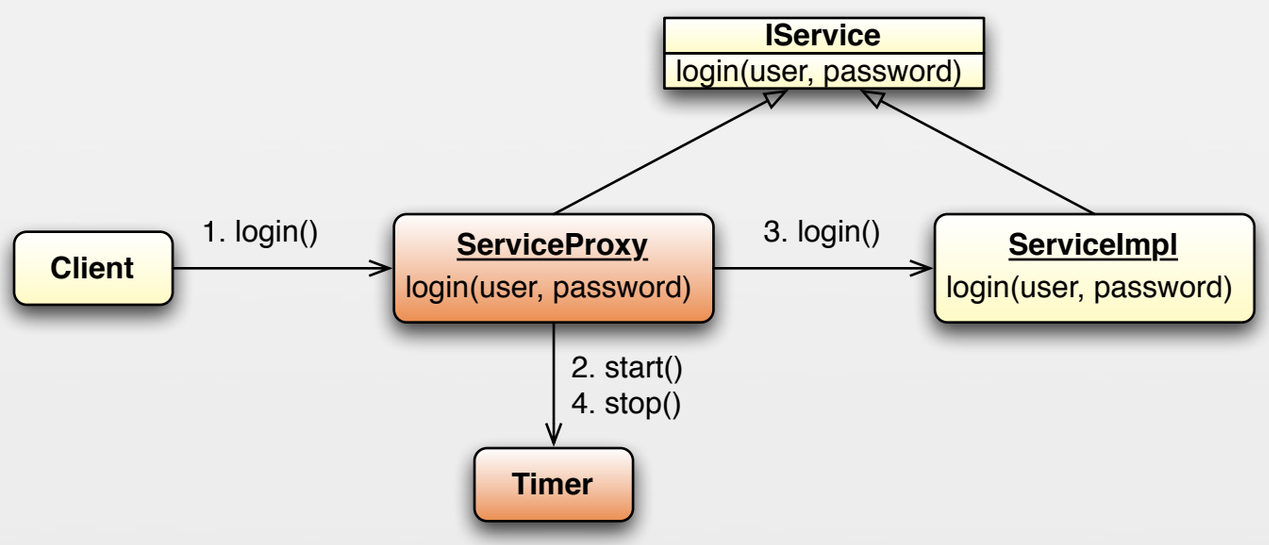


Don‘t use System.currentTimeMillis()

- StopWatch Tool



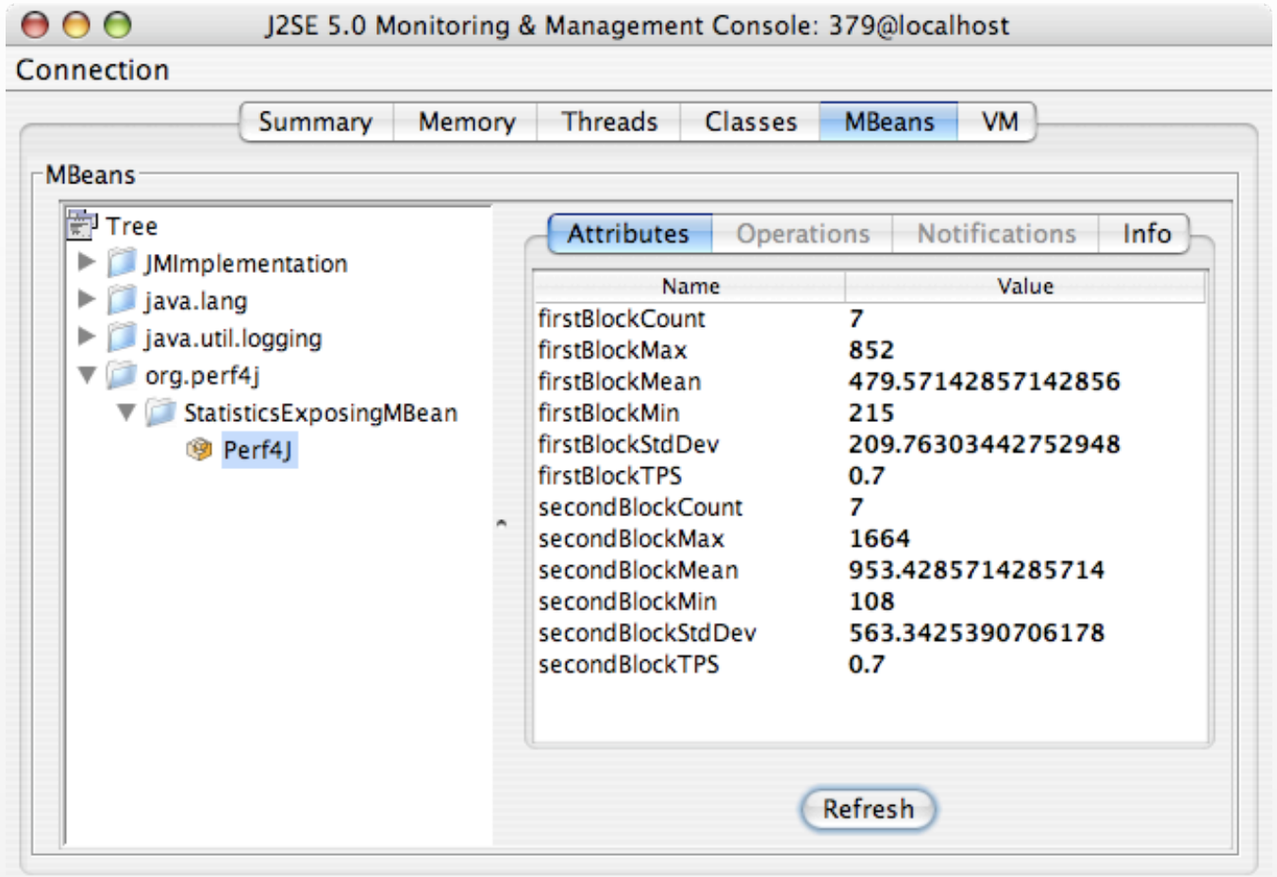
- Monitoring Proxy

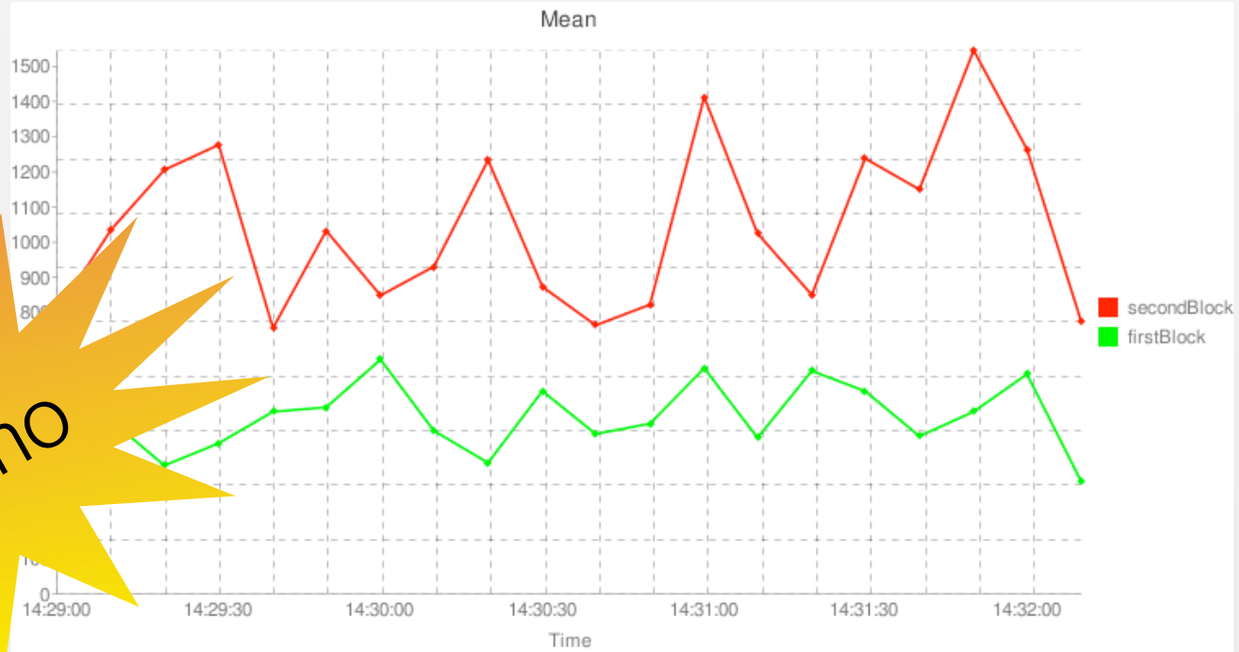


- OS Tools:
  - JAMon API - dormant (2007)
  - Java Simon - active (2010)
  - **Perf4J - active (2010)**

Perf4J

- Similar idea to Simon, plus:
  - Charting
  - Better JMX monitoring
  - Time Slicing
- Active
- Version 0.9.3 (2010)
- Use:
  - Explicit StopWatch or
  - Annotations + AOP
- No JDBC proxy yet





Demo

Monitoring EJBs with Perf4J

```
@Stateless
public class HelloWorldBean implements HelloWorld {

    @Interceptors(org.perf4j.log4j.aop.EjbTimingAspect.class)
    public void sayHello() {
        System.out.println("Hello!");
    }
}
```

Load Generation for manual performance tests

Introducing
JMeter and The Grinder

- Load Generator GUI
- **Tree-Based graphical configuration of test plans**
- Multithreaded, distributed execution
- Predefined connectors for HTTP(S), SOAP, JDBC, LDAP, JMS, POP3(S) + IMAP(S)
- Result aggregation and visualization
- Extensible: samplers, timers, visualization plugins, functions, scriptable samplers

Apache JMeter

Share Test Plan.jmx (/Users/sjavey/Desktop/Test Plans/Share Test Plan.jmx) – Apache JMeter (2.3.1)

File Edit Run Options Help

0 / 0

- Share Test Plan
  - Share Users
    - User Defined Variables
    - HTTP Request Defaults
    - HTTP Cookie Manager
    - HTTP Header Manager
    - Login Page
    - Login
    - 1timeToLive
    - 2getProperties
    - 3getUser
    - 4getHomeSpaceData
    - 5getRecipients
    - 6getUsersByEmail
    - 7getAllowedPDFConversionExtensions
    - 8getIllegalFilesExtensions
    - 9getNodeList
    - Graph Results
  - WorkBench

HTTP Request

Name: 2getProperties

Comments:

Web Server

Server Name or IP: Port Number:

HTTP Request

Protocol (default http): Method: POST Content encoding:

Path: adc/messagebroker/amf

☑ Redirect Automatically ☐ Follow Redirects ☑ Use KeepAlive ☐ Use multipart/form-data for HTTP POST

Send Parameters With the Request:

Name:	Value	Encode?	Include Eq...

Add Delete

Send a File With the Request:

Filename: /Users/sjavey/Desktop/amf/2.getProperties.amf Browse...

Value for "name" attribute:

MIME Type: application/x-amf

Optional Tasks

☐ Retrieve All Embedded Resources from HTML Files ☐ Use as Monitor

Embedded URLs must match:

The Grinder

- Load generator tool
- generic: can call anything that has a Java API: HTTP, SOAP, REST, CORBA, RMI, JMS, EJB, custom protocols
- mature HTTP support
- multi-threaded, multi-process, distributed execution
- **script based (Python)**
- more flexible than JMeter
- steeper learning curve
- record & replay
- GUI: script editor, execution controller & monitor

The Grinder

Grinder Konsole

Datei Aktion Verteilung Hilfe

Proben Interval: 1000 ms

Ignoriere 0 Proben 0

Endlose Probensammlung! 0

Warte auf Proben

0,00 TPS

Gesamt

---- ms (Mittel)
0,00 TPS (Mittel)
0,00 TPS (Spitze)
0 Transaktionen
0 Fehler

Graphen | Resultate | Prozesse | Skript

- examples
  - ClientForm$py.class
  - amazon.py
  - composite.py
  - console.py
  - cookies.py
  - ejb.py
  - email.py
  - fba.py
  - form.py
  - grinder.properties
  - helloworld$py.class
  - helloworld.py
  - helloworldfunctions$py.c
  - helloworldfunctions.py
  - http.py
  - httpg2.py
  - httpunit.py
  - jaxrpc.py
  - jdbc.py
  - jmsreceiver.py
  - jmssender.py
  - proportion.py
  - scenario.py
  - scriptlifecycle.py
  - statistics.py
  - sync.py
  - testRandomise$py.class
  - urllib2$py.class
  - xml-rpc.py

helloworld.py

```
# A minimal script that tests The Grinder logging facility.
#
# This script shows the recommended style for scripts, with a
# TestRunner class. The script is executed just once by each worker
# process and defines the TestRunner class. The Grinder creates an
# instance of TestRunner for each worker thread, and repeatedly calls
# the instance for each run of that thread.

from net.grinder.script.Grinder import grinder
from net.grinder.script import Test

# A shorter alias for the grinder.logger.output() method.
log = grinder.logger.output

# Create a Test with a test number and a description. The test will be
# automatically registered with The Grinder console if you are using
# it.
test1 = Test(1, "Log method")

# Wrap the log() method with our Test and call the result logWrapper.
# Calls to logWrapper() will be recorded and forwarded on to the real
# log() method.
logWrapper = test1.wrap(log)

# A TestRunner instance is created for each thread. It can be used to
# store thread-specific data.
class TestRunner:

    # This method is called for every run.
    def __call__(self):
```

Thanks for your attention

Volker Bergmann
volker@databene.org