Business Case

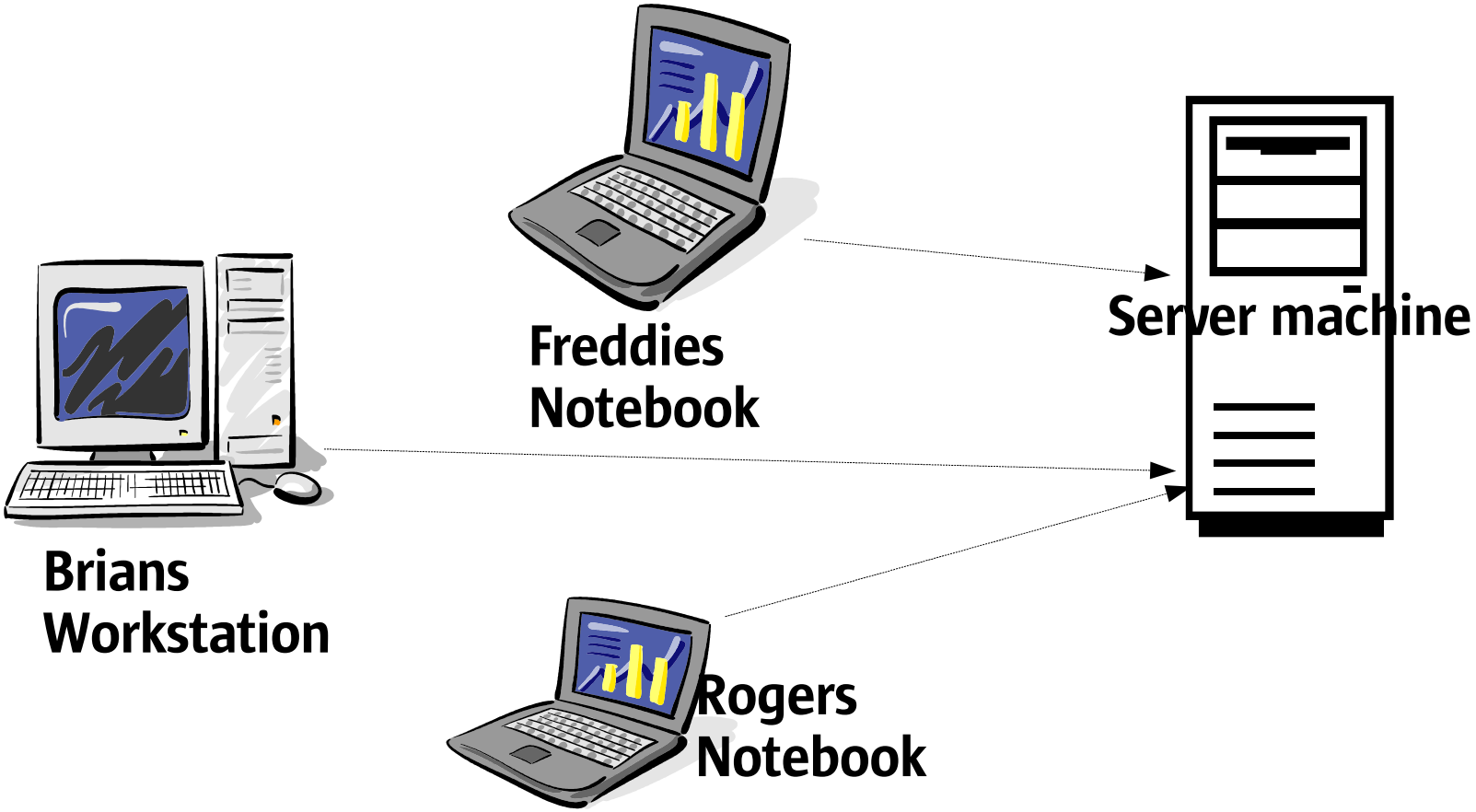
- Developer implements Unit tests based on JUnit, HTTPUnit
- (Some) Unit tests can and should be the basis for
  - Load and Stress tests
  - Concurrency tests (code is threadsafe)
  - Performance measurements

Problems of Unit tests for the upper purposes

- Loops and threads in JUnit tests have to be implemented manually
- Mostly no dedicated hardware for Stress tests, but Hardware of developers can be used
- no orchestrated start of tests on multiple machines (simulate multiple clients executing multiple use cases)
- Test classes must be distributed manually on the dedicated hardware
- Tests must be started / stopped manually by the owner of the test machine
- Test results must be collected manually from the different developer machines
- Test results can only be transformed to a better readable form manually (e.g import results as csv file, use StarOffice or MS Office drawing features)

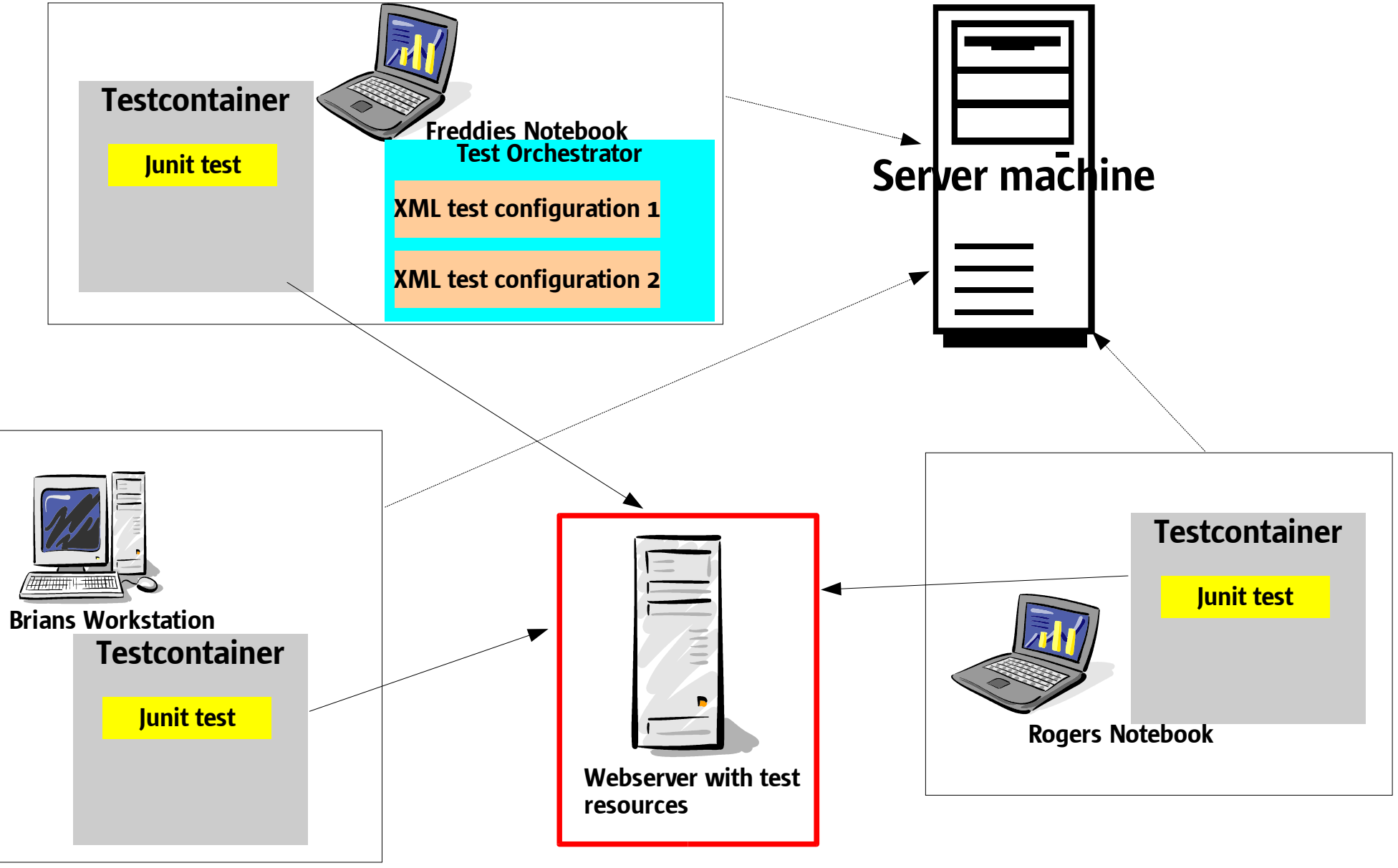
- Following slides outline the idea of a load / stress test tool based on JUnit / HTTPUnit as test drivers to solve the upper problems

Business case - example



- Server machine runs server application to test
- Freddy has developed several unit tests to test server application (use cases)
- JUnit tests should run multithreaded (to simulate multiple clients)
- JUnit tests (multithreaded) should also run on Brians Workstation
- After 10 min Freddie decides to increase load on server application using Rogers Notebook running the tests
- Brian and Roger don't want to be disturbed by Freddie (Freddie uses their Hardware for testing without intervention of Brian or Roger)

Scenario: How does it work 1



Scenario: How does it work 2

- All Hardware which participates in the load test scenario need a Java WebStart installation
- Java WebStart starts a Java-RMI process called “Testcontainer”
- “Testcontainer” Application is taken from “Webserver with test resources”
- Testcontainer reuses the idea of container – they are able to deploy and execute JUnit tests
- Testcontainer are a generic mechanism to deploy and execute any JUnit test in a multithreaded or loop “version”
- Freddie, Brian and Roger start their “Testcontainer” via Java WebStart
- thats all for Brian and Roger – they can continue working while Freddie can use their Hardware for (distributed) testing

Scenario: How does it work 3

- Freddie wants now to execute his Unit tests; multithreaded on his and Brians machine
- He starts the TestOrchestrator via Java WebStart
- Freddie puts the jars with the test classes and all needed supporting jars (e.g. drivers) on the “Webserver with test resources”
- Freddie uses the TestOrchestrator to create an XML test configuration (describes the test in all aspects)
- The XML test configuration contains information like
  - all Jars needed to perform the test
  - the test classes and test methods to perform
  - the number of threads, which run the test
  - the number of iterations (how often should the test be repeated in a for-loop)

Scenario: sample test configuration

```
<?xml version="1.0" ?>
<TestConfiguration orchestrator=”<IP_OF_ORCHESTRATOR>”>
  <JavaResources>
    <JavaResource name="MyTestClasses.jar"/>
    <JavaResource name="driver.jar"/>
  </JavaResources>

  <Test>
    <Class name="com.xyz.SomeTestClass">
    <Method name="testSomeThings">
    <NumberOfThreads count="1">
    <NumberOfRuns count="10">
  </Test>
 <Test>...</Test>
</TestConfiguration>
```

Problems and answers

- Problem: Tests must be started with different JDKs
  - Answer: Java WebStart solves the problem
- Problem: Testcode which can be deployed on a TestContainer can do malicious things
  - Answer: JVM processes can be configured with policy files
  - Answer: Developers in team should trust themselves
- Problem: Testcode can exhaust system resources on TestContainer machine
  - Answer: TestContainer or Testcode must be stopped, either by the TestOrchestrator or by the owner of the Testhardware
- Problem: TestContainer must execute multiple test scenarios without being restarted
  - Answer: implement a custom class loader for the loaded jars and dereference it after the test

Problems and answers

- Problem: Tests code must be instrumented with fine grained measuring points
  - Answer: maybe enhance JUnit framework to deliver such information