

Bringing Subversion to the Java(TM) World

SubConf 2009

Alexander Kitaev, Alexander Sinyushkin

TMate Software Ltd. © 2009

<http://svnkit.com/subconf2009/>

Welcome & Outline

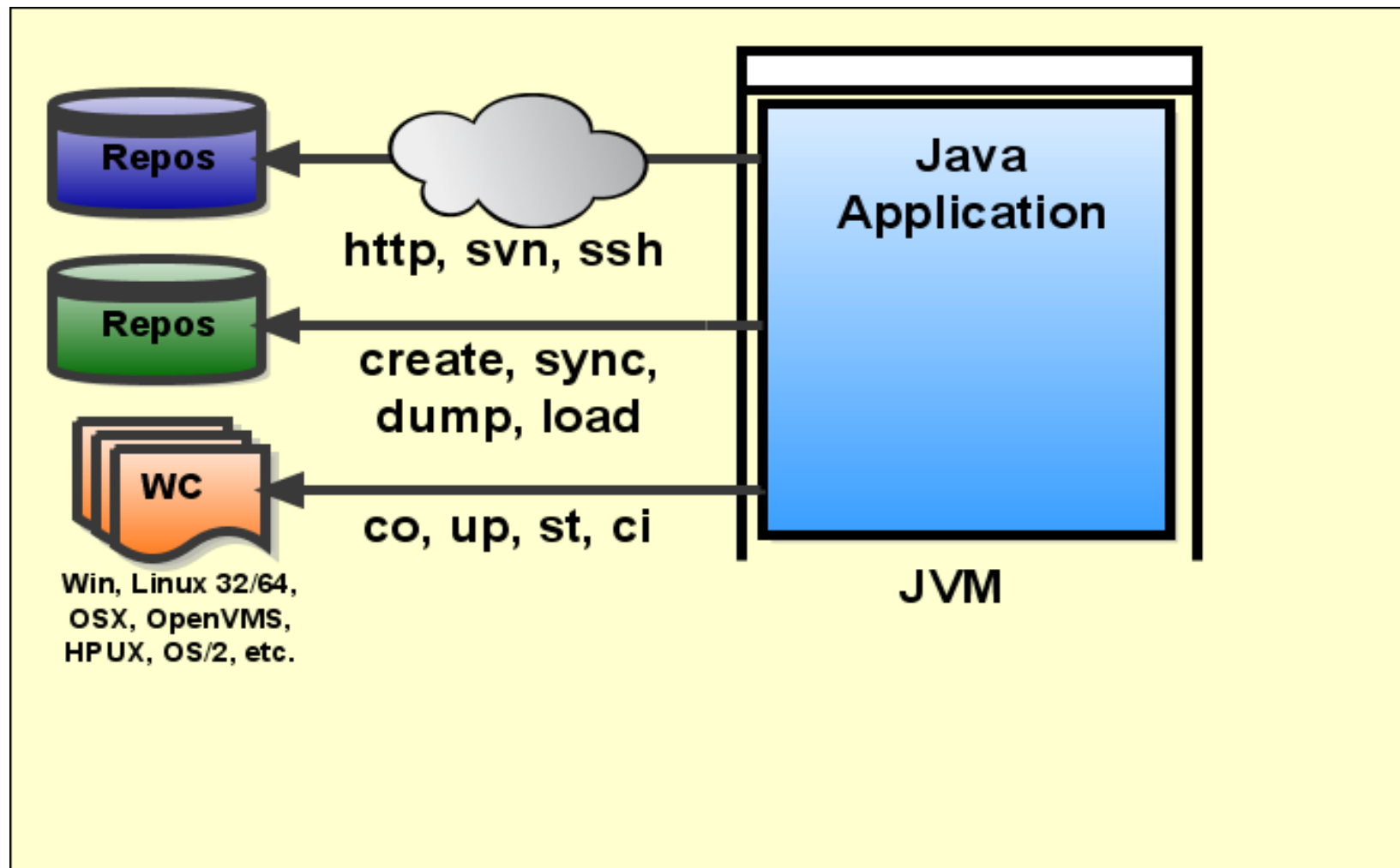
Subversion and Java

- Introduction
- Integration approaches overview
- JavaHL and Command Line
- Integrating using SVNKit
- Combined approach
- Non-technical aspects

Company Profile

- TMate Software: software development and consulting company
- Founded in 2003, headquarters in Prague, 5 employees
- SVNKit - barrier-free access to all Subversion features for Java developers and their applications

Java meets Subversion



Java meets Subversion

- IDEs – IDEA, Subclipse, Subversive, JDeveloper
- Standalone clients – SmartSVN, SyncroSVN
- Configuration Management tools – Ant, Maven, TeamCity, Bamboo, JIRA
- Content management systems – Groowiki
- Less usual use cases – JBoss Shotoku

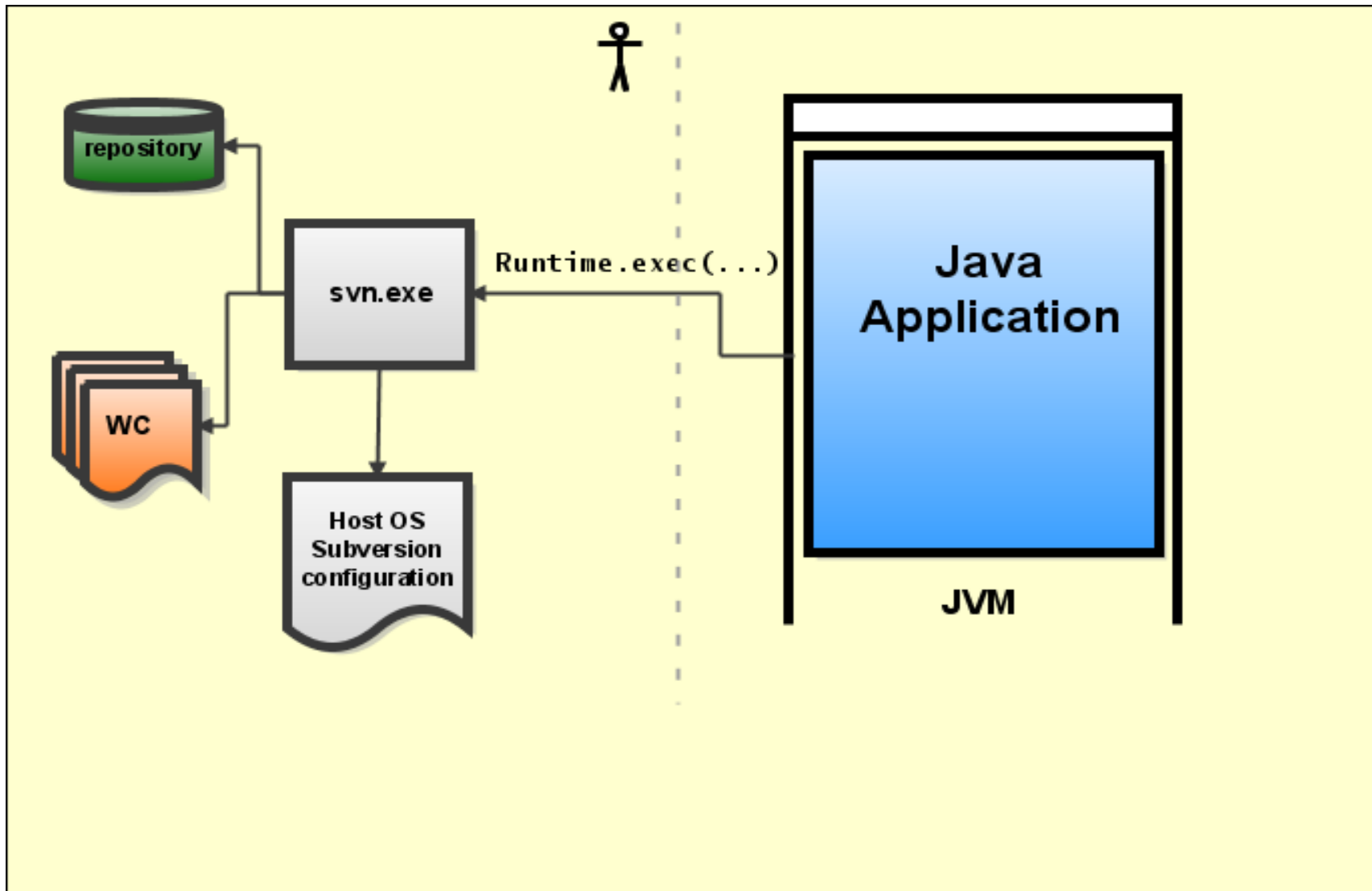
Integration Aspects

- Target platform diversity
- End-user assistance availability
- Features needed
- Configuration and authorization control
- Development environment aspects

Integration Approaches

- Command Line Client
- JavaHL (JNI bindings)
- SVNKit: independent pure Java Subversion implementation
- SVNKit and JavaHL combined

Command line client



Command line client

Sane command syntax and parseable output

```
$ svn co URL /home/path
```

```
A    /home/path/file.txt
```

```
Checked out revision 10.
```

```
$ svn st --xml /home/path
```

```
<status>
```

```
<XML document describing working copy status>
```

```
</status>
```

```
$ svn ci -m "commit message" /home/path
```

```
Sending /home/path/file.txt
```

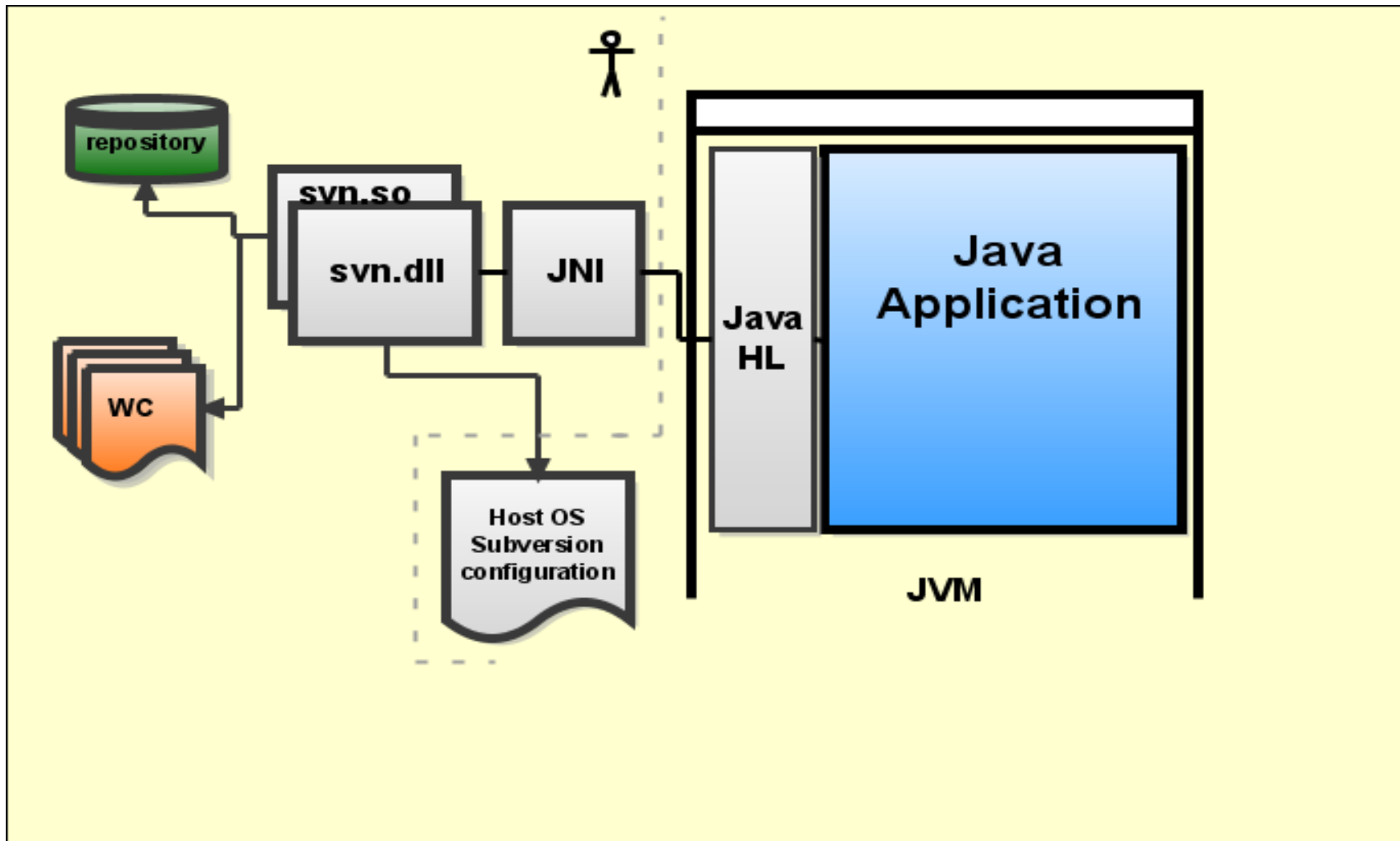
```
Transmitting file data .
```

```
Committed revision 11.
```

Command line client

- Works anywhere where Subversion command line client is installed
- End-user assistance required
- All command line client features
- Mostly no control on configuration
- Version-dependent code, external process management

JavaHL – JNI Bindings



JavaHL – JNI Bindings

```
SVNClientInterface client = new SVNClient();

// checking out
client.checkout(URL, path, Revision.HEAD, Revision.HEAD,
               Depth.infinity, false, true);

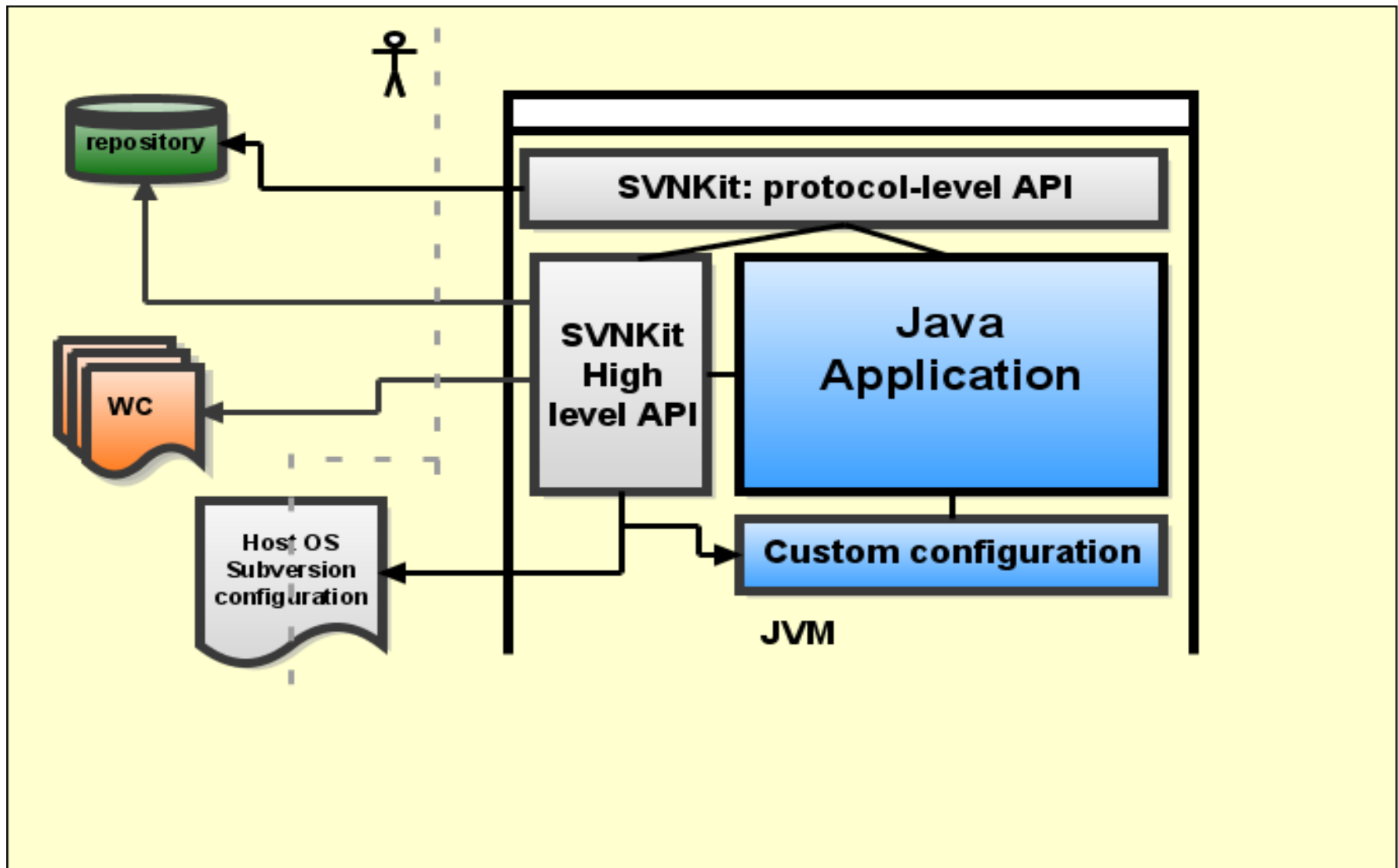
// status
client.status(path, Depth.infinity, false, true, false, false, null,
             new StatusCallback() {
                 public void doStatus(Status status) {
                     }
             });

// commit
client.commit(new String[] {path}, "commit message",
             Depth.infinity, false, false, null, null);
```

JavaHL – JNI Bindings summary

- JavaHL native library and Subversion binaries for every target platform
- May work out of the box, or may require experienced end-user or administrator assistance
- Full high-level features set
- Same as for the command line client
- Heterogenous development environment

SVNKit – Pure Java approach



SVNKit – Pure Java approach

- Independent pure Java Subversion implementation
- Supports Subversion 1.6.5 and older versions
- Works with local FSFS and remote repositories
- Runs on JVM 1.4 or newer

SVNKit Example: high-level API

```
// cheking out
manager.getUpdateClient().
    doCheckout(url, path,
        SVNRevision.HEAD, SVNRevision.HEAD, SVNDepth.INFINITY, true);
// status
manager.getStatusClient().
    doStatus(path, SVNRevision.WORKING, SVNDepth.INFINITY,
        false, true, false, false,
        new ISVNStatusHandler() {
            public void handleStatus(SVNStatus status)
                throws SVNException {
            }
        }, null);
// commit
manager.getCommitClient().
    doCommit(new File[] {path}, false, "commit message", null,
        null, false, false, SVNDepth.INFINITY);
```

SVNKit Example: protocol-level API

We'll use two different approaches for getting repository tree:

- Standard recursive list of SVNKit 'trunk' with `SVNRepository.getDir()` : about **1.5 minutes**
- Tricky method with 'status' command `SVNRepository.status()` : about **5 seconds**

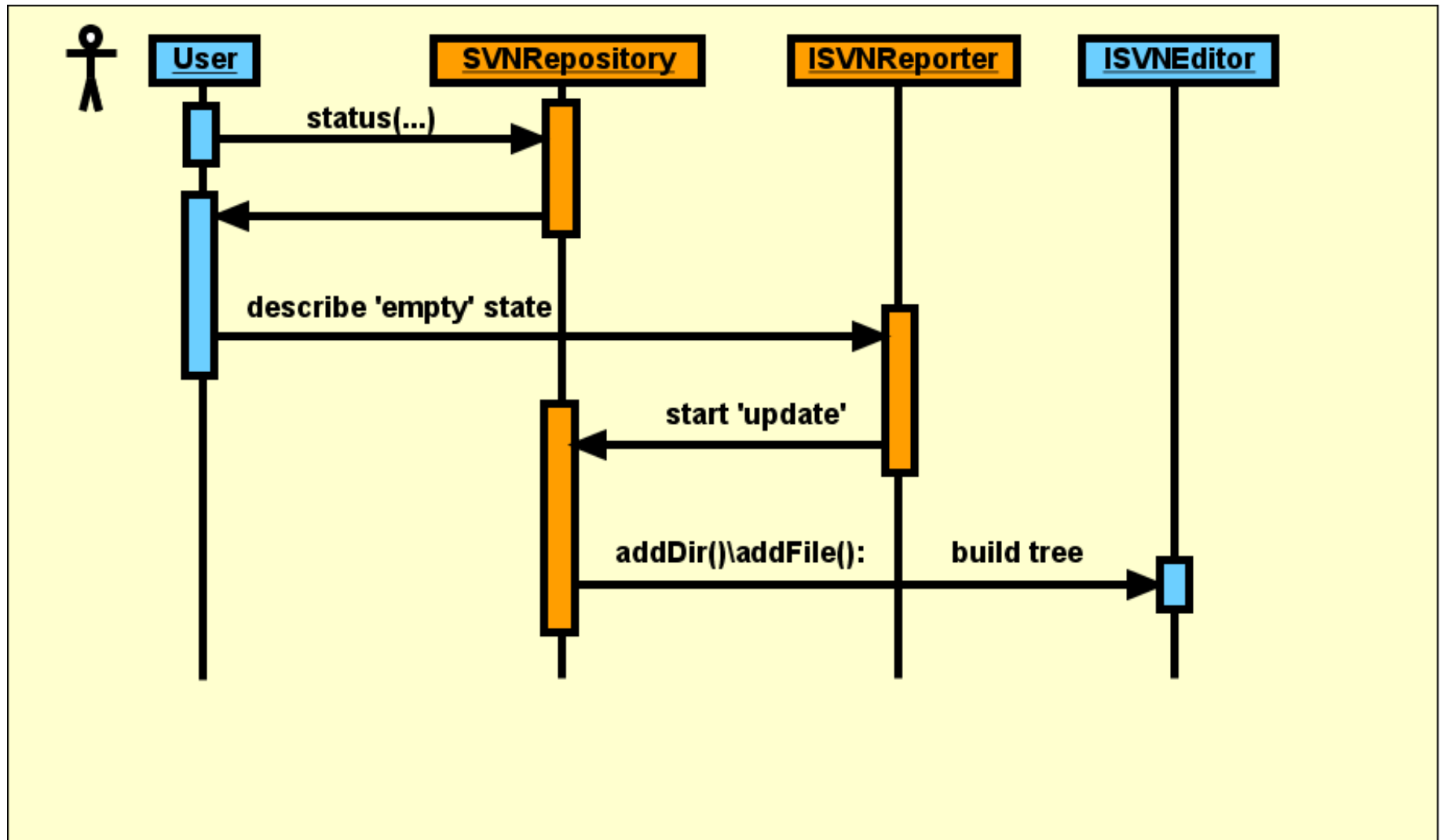
Full source code with comments is available at <http://svnkit.com/subconf2009/> web page.

Getting repository tree: slow

```
SVNRepository repos = SVNRepositoryFactory.create(URL);
...
list(repos, true);
...

public void list(SVNRepository repos, boolean recursive, ...) {
    ...
    // a full request-response cycle goes from here
    repos.getDir(...);
    // to here
    ...
    if (recursive) {
        // and repeated for each directory
        list(repos, recursive, ...);
    }
    ...
}
```

Getting repository tree: fast



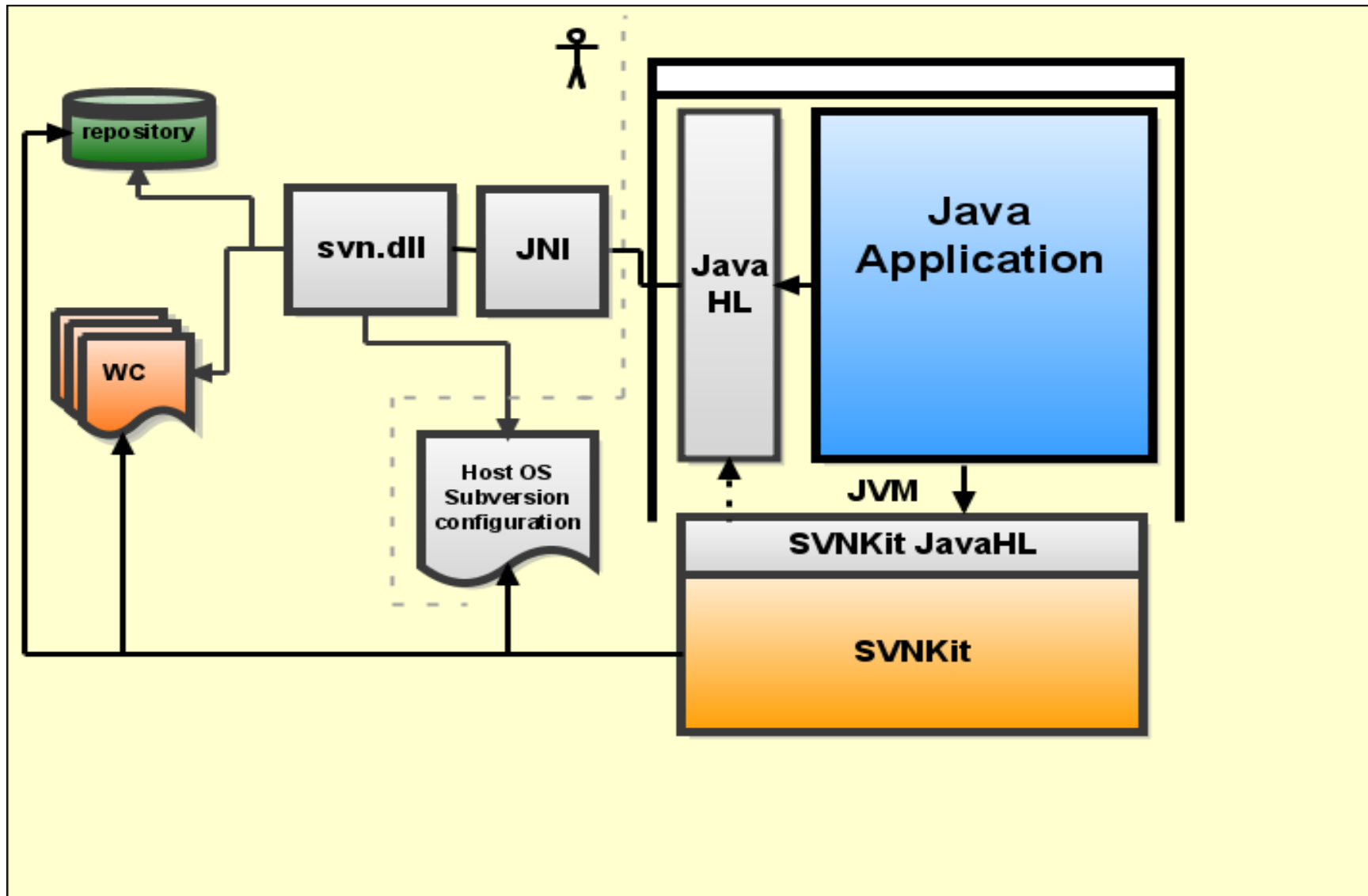
Getting repository tree: fast

```
ISVNEditor editor = new ISVNEditor() {
    public void addDir(...) {
        // collect directory paths here
    }
    public void addFile(...) {
        // collect file paths here
    }
    ....
};

ISVNReporterBaton reporterCallback = new ISVNReporterBaton() {
    public void report(ISVNReporter reporter) throws SVNException {
        // report empty 'local' state.
        reporter.setPath("", null, rev, true);
        reporter.finishReport();
    }
};

// ask repository - what will be 'updated'?
repos.status(rev, null, true, reporterCallback, editor);
```

Combining SVNKit and JavaHL



Combining SVNKit and JavaHL

SVNKit implements JavaHL SVNClient API

```
import org.tigris.subversion.javahl.SVNClientInterface;
import org.tigris.subversion.javahl.SVNClient;

import org.tmatesoft.svn.core.javahl.SVNClientImpl;

public class SVNTest {
    public static void main(String[] args) {
        SVNClientInterface javaHLClient = new SVNClient();
        SVNClientInterface svnkitClient =
            SVNClientImpl.newInstance();
        ...
    }
}
```

SVNKit approach summary

- Portable: runs in JRE 1.4 or newer
- May require zero end-user assistance
- Rich Java API suitable for different task
- Full control over configuration and authorization
- Homogeneous development environment, JavaHL API implementation, unique features

SVNKit Unique Features

- Different working copy formats without forced upgrade of the working copy
- Atomic commits from disjoint working copies
- Performance improvements: connection pools for svn and svn over ssh protocols; fast update
- API to perform `svnlook` and `svnsync` operations
- Special handling of missing files on commit operation
- and more...

SVNKit: disadvantages

- Independent implementation:
 - Specific bugs
 - Delayed releases (not later than one month after Subversion GA release)
 - Possible incompatibility issues
- Isolated from the target platform
- Specific licensing policy

SVNKit Development Process

- Open Source
- Continuous integration
- Native Subversion Python tests suite: more than **800** tests for each protocol on different JVM versions.
- Compatible stable version of SVNKit not later than in a month after Subversion release
- We do eat our own dog-food :)

Support and Licensing

- SVNKit features **Dual Licensing** scheme:
 - No limitations for Open Source products
 - No limitations for internal, in-house use
 - Reasonable licensing fee for closed-source products
- Dedicated **technical support**:
 - Highest priority with support and bugs fixing
 - Custom features implementation upon request
- **Fast turn-around** for all SVNKit users

SVNKit Roadmap

- Under the hood (SVNKit 1.4):
 - WC-NG support with the help of SQLJet Library
 - Support of other Subversion 1.7 features
 - Pure Java Subversion server implementation as a web application
- Right Now (SVNKit 1.3.x):
 - Keep fixing bugs and implementing new features of 1.6.x Subversion version line

SVNKit – you are welcome!

- SVNKit web site: <http://svnkit.com/>
- SVNKit mailing list: [@Nabble](#)
- SVNKit support on technical and licensing questions: **support@svnkit.com**

Thank you!

Thank you for your attention,
we wish you a nice day at the conference!