

---

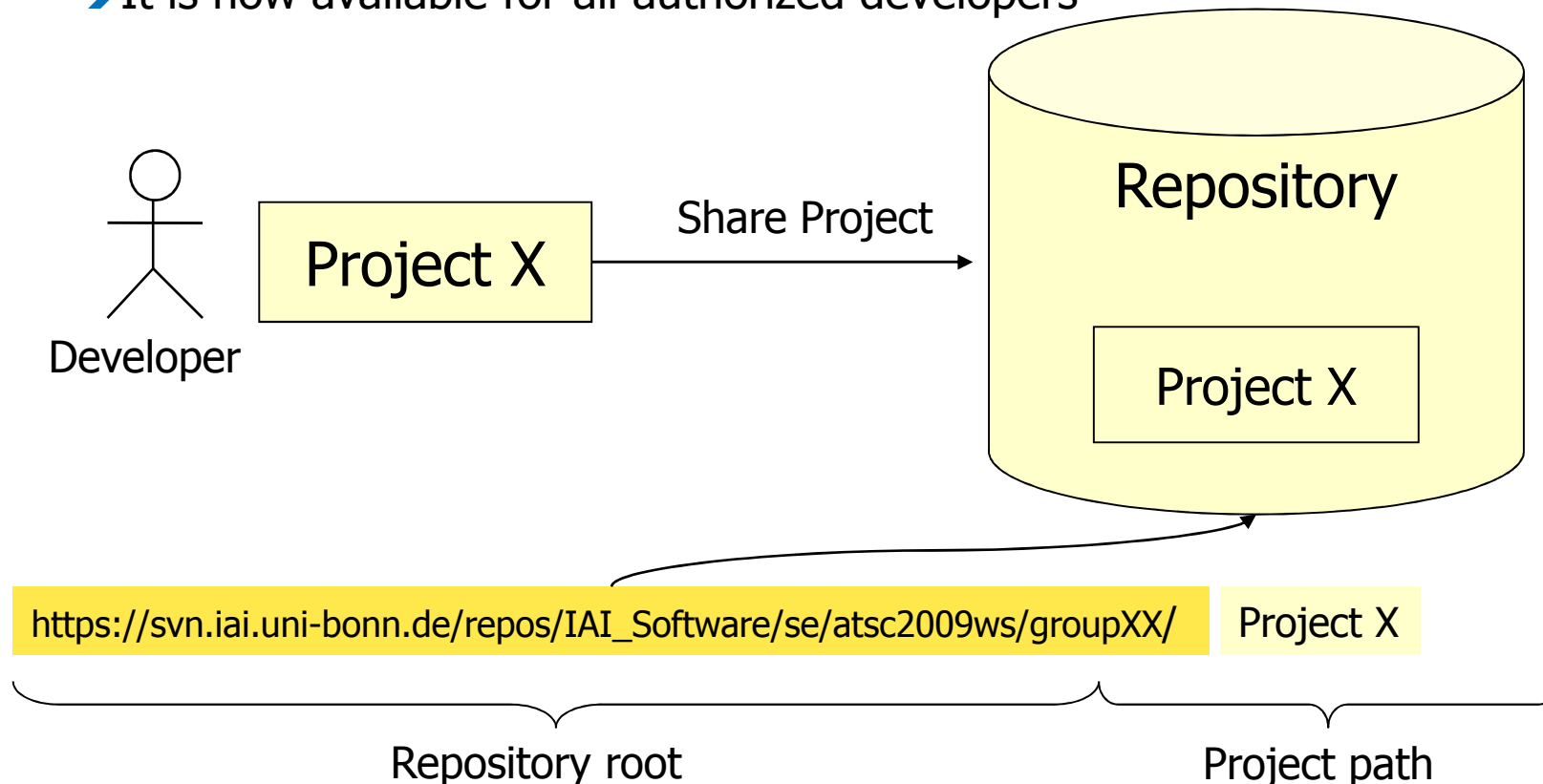
# Introduction to Subversion and the Eclipse Subversion plug-in

Prof. Armin B. Cremers, Daniel Speicher, Tobias Rho



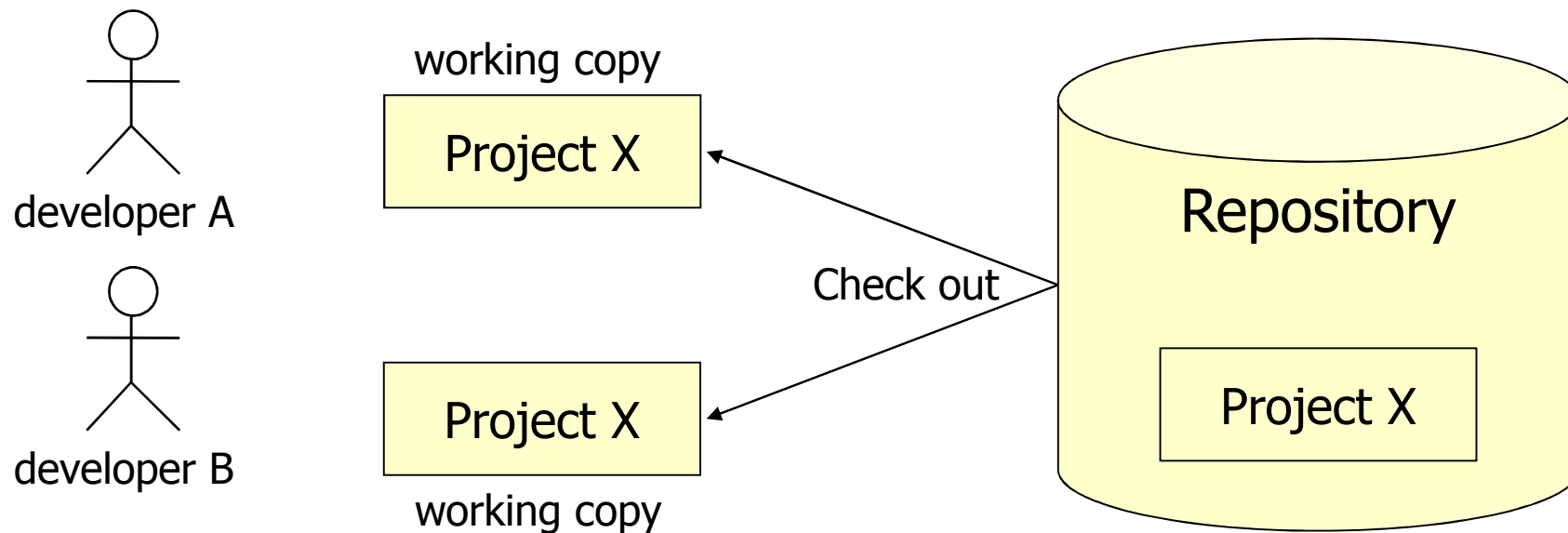
# Sharing / Checkout: Put Project Under Version Control

- ◆ A repository exists
- ◆ Your project exists but you didn't share it.
- ◆ **Sharing:** The project is added to the repository
  - It is now available for all authorized developers



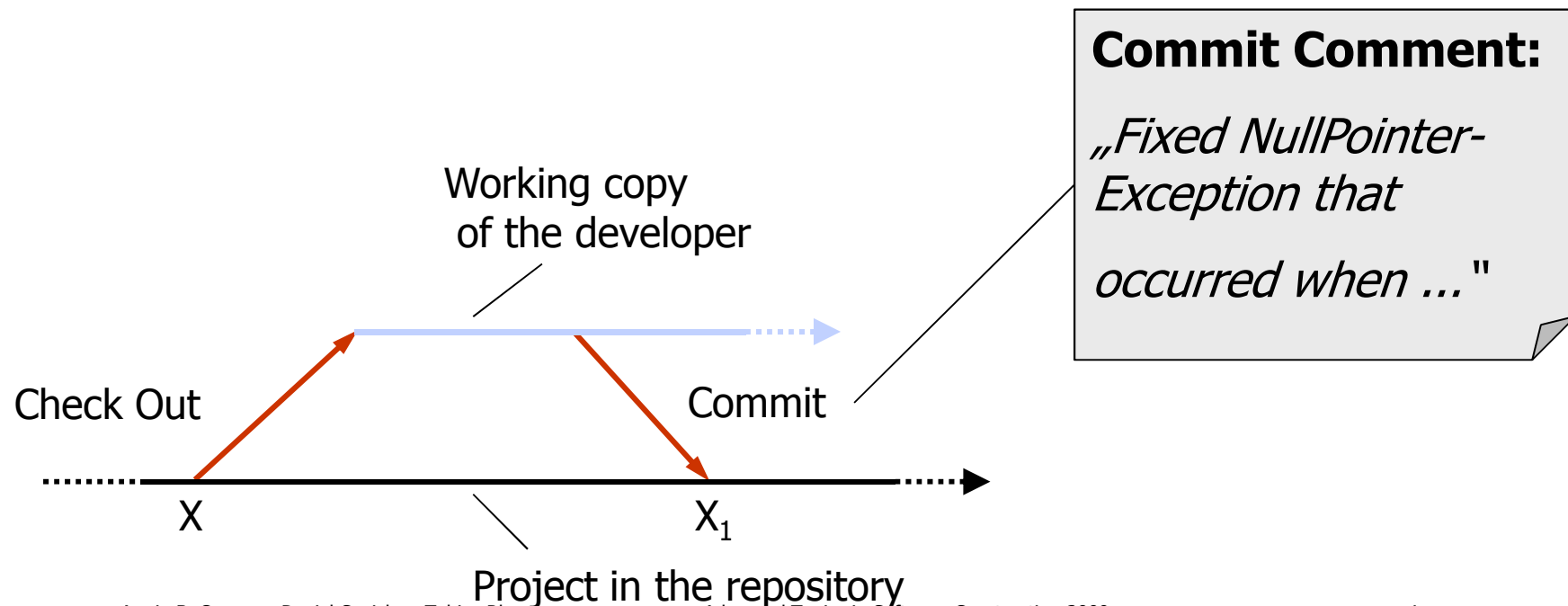
# Check Out: Initial Project Download

- ◆ Developers check-out the project
  - ◆ get a local **working copy**
- ◆ From now on they can work on the project
- ◆ Check-out is only done once!
  - ◆ Get new versions → **Update**



# Commit: Writing Changes To The Repository

- ◆ The developer changes his working copy
- ◆ He adds his changes to the repository (Commit)
- ◆ A Commit Comment describes what has changed and why



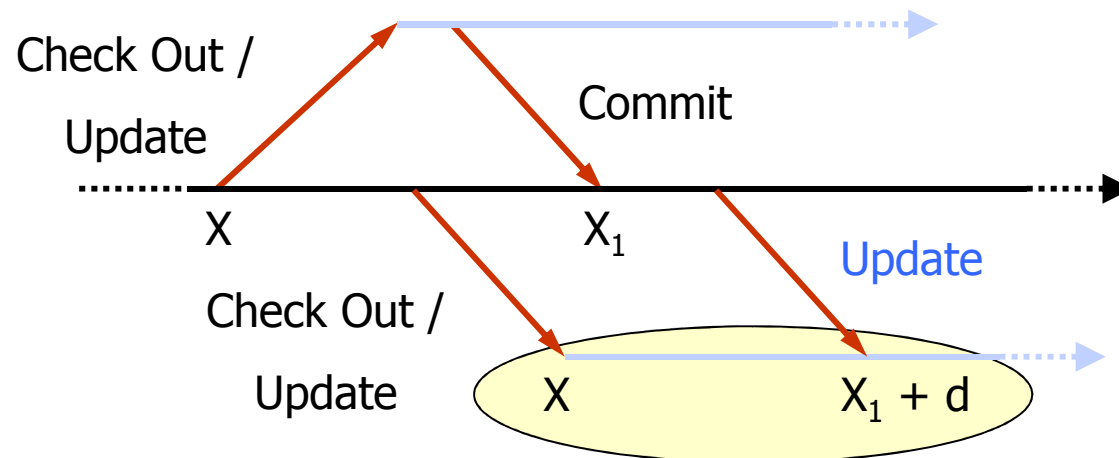
# Update: Get Latest Changes From the Repository

- ◆ Updates the working copy with the **current state** of the repository
- ◆ Problem: Changes are not verified
- ◆ Better: → Use „**Synchronize**“

Working copy  
of developer A

Project in the repository

Working copy  
of developer B



# Synchronize: Synchronize The Project

- ◆ Motivation

- ◆ Compare the local version and the repository version
  - ◆ automated overview comparison
  - ◆ automated detailed comparison
- ◆ Decide yourself what to copy
- ◆ Perform selective update or commit

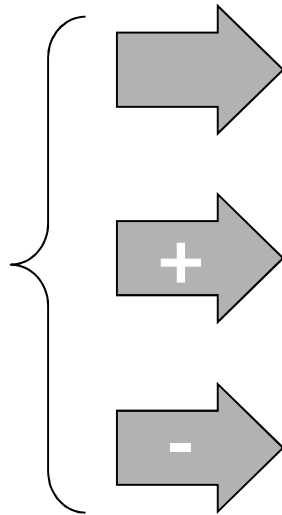
- ◆ Automated **overview** comparison

- ◆ Comparison at project level



# Synchronize: Meaning of Symbols in „Synchronize View“

## Outgoing changes



File has been changed locally and exists in the repository

→ Copy local version into the repository

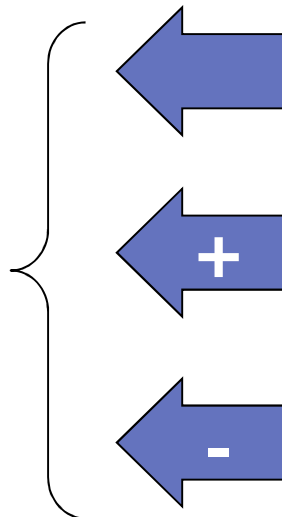
File has been added locally and doesn't exist in the repository

→ Add the file to the repository

File has been deleted locally and exists in the repository

→ Delete the file from the repository

## Incoming changes



File has been changed in the repository and exists locally

← Copy repository version into the the working copy

File has been added to the repository and doesn't exist locally

← Add the file to the local workingcopy

File has been deleted from the repository and exists locally

← Delete the file from the local workingcopy

# Synchronize: Automated Detailed Comparison

- ◆ ... opens the **Compare** window
  - ◆ The local workingcopy and the repository versions are compared
  - ◆ Selective updates are possible now
    - ◆ fine-grained: individual change

The screenshot shows a side-by-side comparison of two code files. The left pane is titled 'Local File' and the right pane is titled 'Remote File (335 [mark.schmatz])'. Both panes show the same code structure: a package declaration 'package org.foo;', a multi-line comment block, and a class declaration 'public class Bar {'.

```
Local File
```

```
package org.foo;
```

```
/**  
 *  
 * @author schmatz  
 */  
public class Bar {  
  
}
```

```
Remote File (335 [mark.schmatz])
```

```
package org.foo;
```

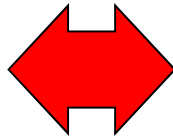
```
public class Bar {  
  
}
```

A vertical line in the center separates the two panes. A blue highlight is visible in the local file's comment block, and a thin line connects it to the corresponding comment block in the remote file, indicating a comparison or synchronization point.



# Synchronize: Synchronize The Project

Conflict



File has been changed in the local working copy **and** in the repository

↔ Manual conflict resolution of the necessary

- ◆ Conflict resolution needs detailed comparison at file level
  - ◆ see next 2 slides

# Solving Conflicts

- ◆ SVN / CVS helps comparing versions
- ◆ Programmer decides what to do
  - ◆ „**Overwrite and Update**“: overwrite local changes with repository version
  - ◆ „**Overwrite and Commit**“: overwrite repository changes with local version
  - ◆ Merge the files „by Hand“ and convert the conflict into an „outgoing change“ by selecting „mark as merged“ in the context menu

Local File	Remote File (394 [mark.schmatz])
<pre>public void myTestMethod() {     log("Test method entered.");      boolean keepOnRunning = true;     int i=0;      while( keepOnRunning )     {         int r1 = doSomething1();         int r2 = doSomething3();          if( r1+r2 &gt; MAX_THRESHOLD )         {             log("Threshold exceeded.");             log("Iterations: " + i);             keepOnRunning = false;         }     } }</pre>	<pre>public void myTestMethod() {     log("Test method entered.");      boolean keepOnRunning = true;     int i=0;      while( keepOnRunning )     {         int r1 = doSomething1();         int r2 = doSomethingElse();          if( r1+r2 &gt; MAX_THRESHOLD )         {             log("Threshold exceeded.");             log("Iterations: " + i);             keepOnRunning = false;         }     } }</pre>