# Principles of Software Construction: Objects, Design, and Concurrency

## Version control with git

Christian Kästner **Bogdan Vasilescu**

(Adapted from Christopher Parnin/NCSU & Prem Devanbu/UC Davis & Kenneth Anderson/CU Boulder)

Intro to Java

Git, CI

UML

Static Analysis

Performance

GUIs

GUIs

Config management

More Git

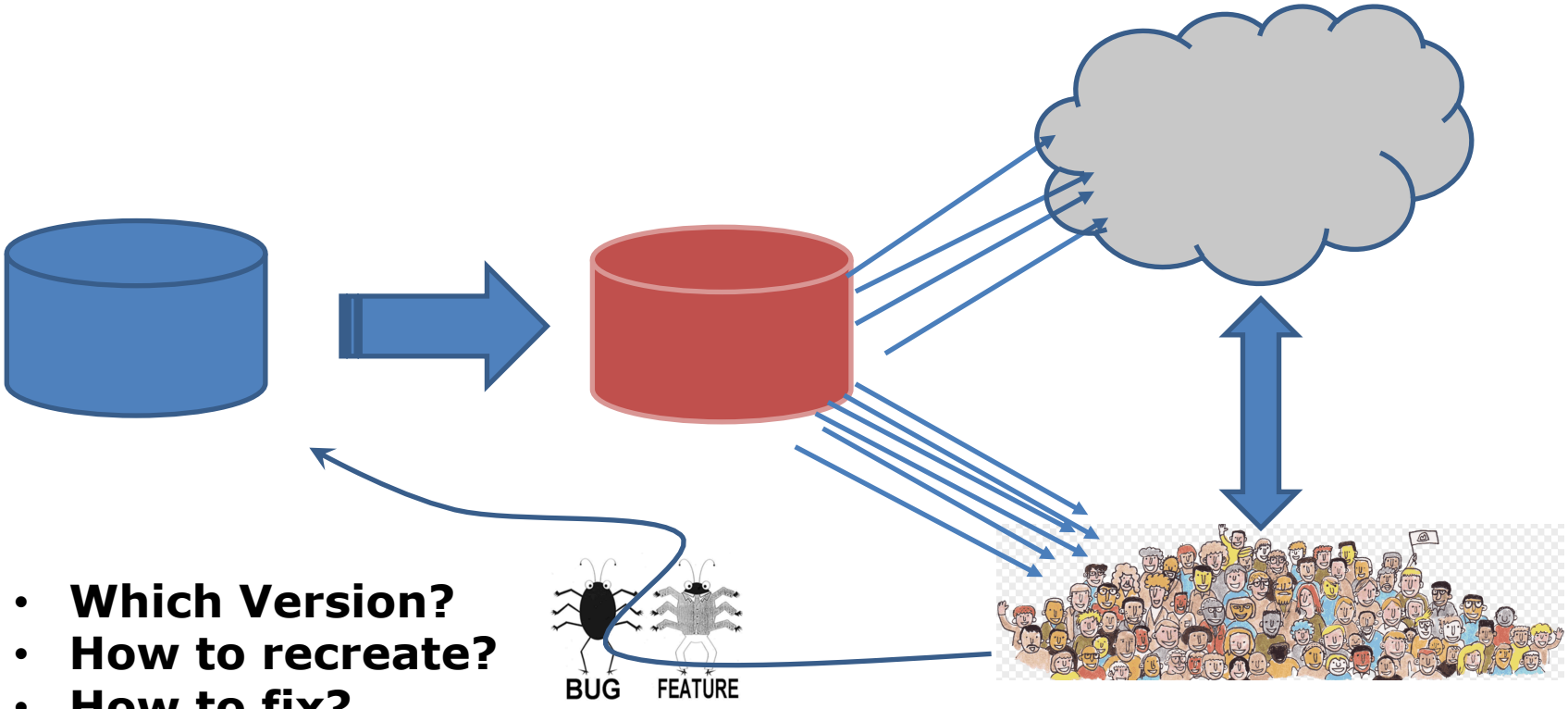
**Design**

| Part 1: Design at a Class Level | Part 2: Designing (Sub)systems | Part 3: Designing Concurrent Systems |
|---|---|---|
| Design for Change: Information Hiding, Contracts, Design Patterns, Unit Testing<br><br>Design for Reuse: Inheritance, Delegation, Immutability, LSP, Design Patterns | Understanding the Problem<br><br>Responsibility Assignment, Design Patterns, GUI vs Core, Design Case Studies<br><br>Testing Subsystems<br><br>Design for Reuse at Scale: Frameworks and APIs | Concurrency Primitives, Synchronization<br><br>Designing Abstractions for Concurrency<br><br>Distributed Systems in a Nutshell |

# Administrivia

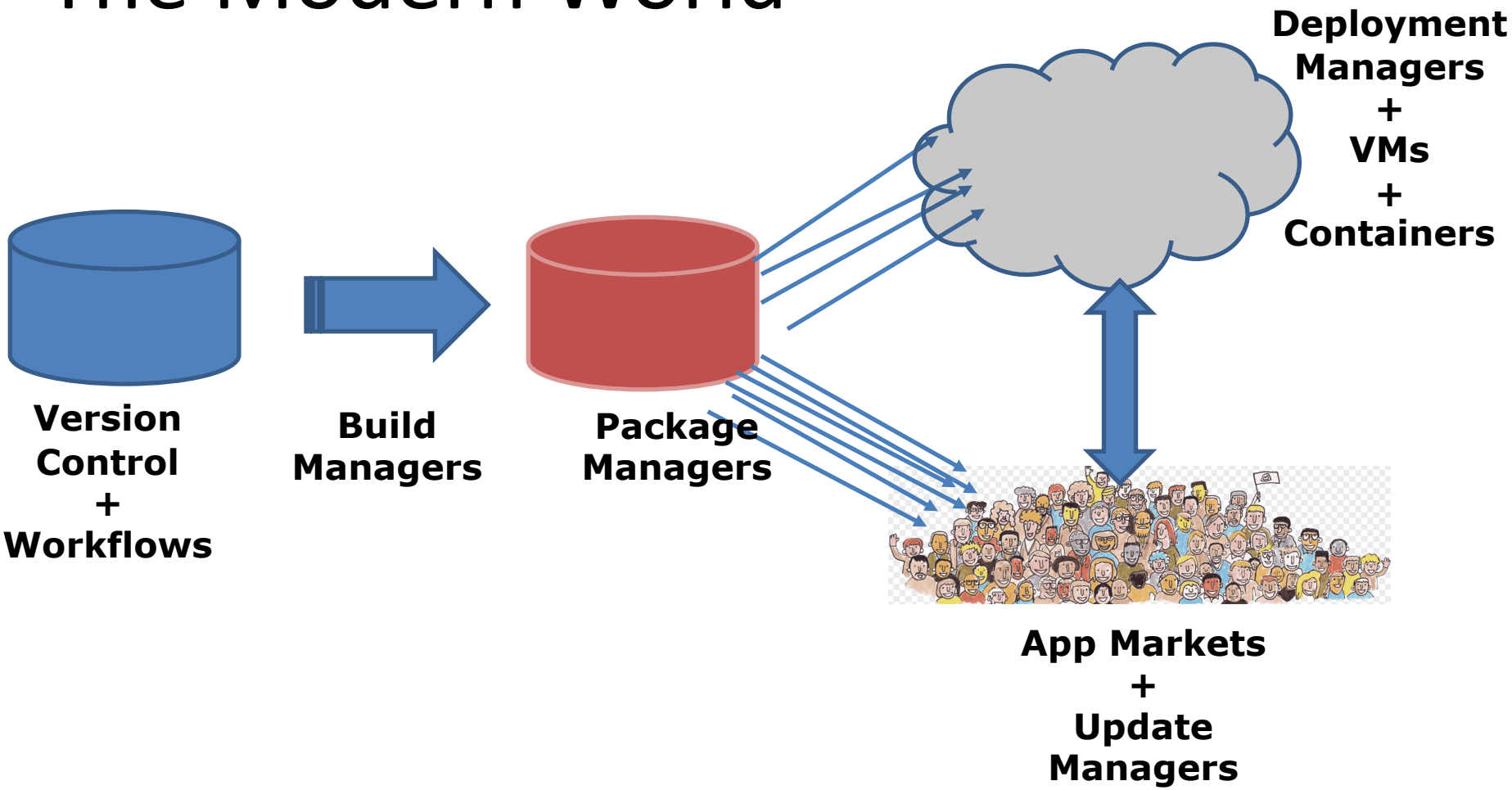
- Homework 6 due tonight, 11:59 p.m.
- Final exam Thursday, 11 May, 1pm-4pm in GHC 4401
- Review session in Wednesday, 10 May, 6pm-9pm in GHC 4401

# The Modern World




- **Which Version?**
- **How to recreate?**
- **How to fix?**
- **Where to apply the fix?**
- **How/when to Redistribute?**

# The Modern World



# Components of Modern Configuration Management

Version Control: Branches/Forks/Workflow
Task and Build managers
Build machines, virtual environments (dev stacks)
Package managers
Containers, VMs, in the Cloud
Deployment –Infrastructure as Code.
Data migration

*Other issues*: orchestration, inventory, compliance

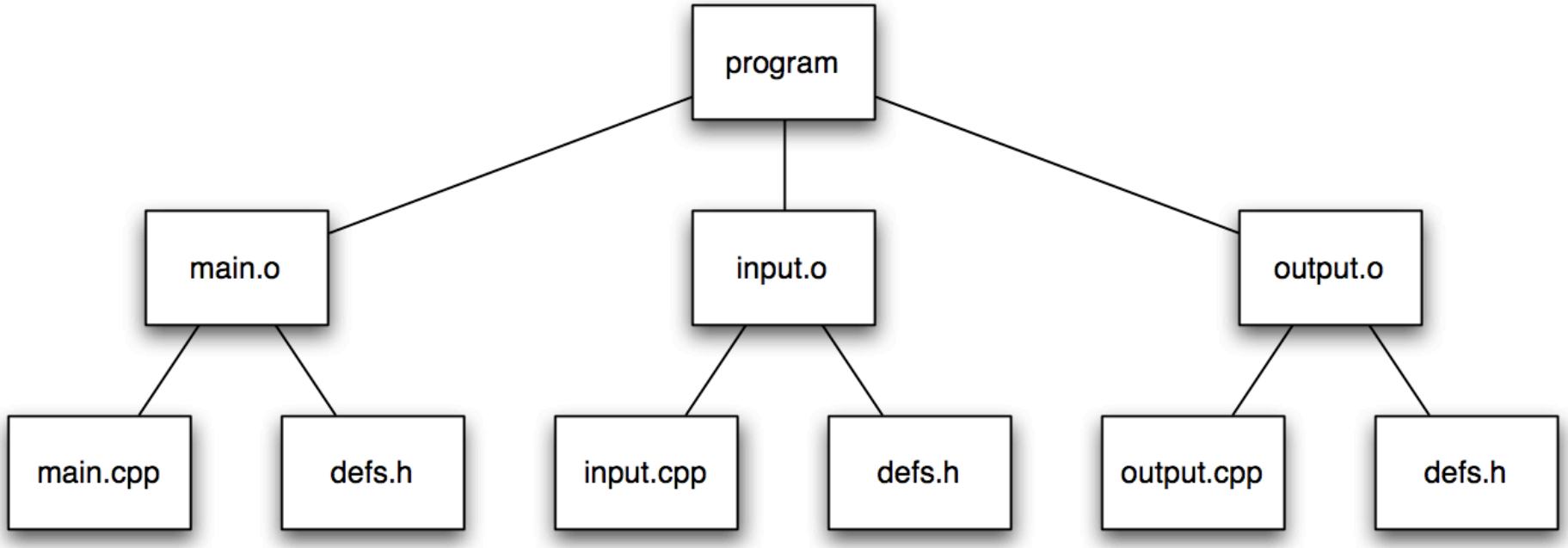
# Task and Build Managers

*Analyze dependencies, and efficiently build (only) what needs to be built or rebuilt.*

**Tools**: make, ivy, ant, maven, gradle, …

# Make dependency graph

- A makefile can be modeled as a dependency graph.
  The make algorithm performs a traversal over the graph.
  Each node is checked after all of its children, and the actions are run if any child has a timestamp greater than its parent



# Good things about make

- Available on pretty much every darn platform.
- Very fast.
- Fully featured programming language (but weird)
- First mover advantage

# Bad things about make

- **Weird syntax (indent is tab, NOT space)**
- Has only global variables.
- Where shell can be used, and where make commands? Weird.
- **No standards for anything. E.g.,: recursion, dependency analysis, file lists.**
- No “reuse” or inheritance of makefiles.
- **Not portable across OS, even across Unix flavors.**
- Debugging? Yeah, good luck with that.
- Can’t guarantee consistency/reproducibility.

# Ant

# Ant vs make

- Make file in XML.
- Replace weird indentations with weird angle brackets.
- Replace “variables” with <property />
- Replace “targets” with <target name=“jar” />
- Replace “rules” with <target name =“jar”,
  depends=“init, classes” />”
- Replace “recipes in shell” with tasks.
  “<javac />”,
  “<mkdir />”,
  “<jar />”

# Ant's model

- Everything is a Task  (sort of)
  - A task has an associated XML element in Ant build files and an associated Java class that implements the task.
  - The XML element can have various attributes and sub-elements, converted into parameters and passed to the Java class.
  - Build file called build.xml by convention
    - First task executed by invoking its associated Java class and passing it its input parameters (if any).

- What's the difference between tasks as shell commands vs tasks as Java?

# Ant Project Format

- build.xml

- Project Name
- Property Values
- Paths
- Tasks
- Targets

# Construction of Ant Build Files

- The default name for a Ant build file is build.xml
- The xml root element must be the ‘project’ element
  - The ‘default’ attribute of the project element is required and specifies the default target to use
- Targets contain zero or more AntTasks
  - The ‘name’ attribute is required
- AntTasks are the smallest units of the build process

# Ant Build File Example

```
<project default="hello">
   <target name="hello">
      <echo message="Hello, World"/>
   </target>
</project>
```

**Execution of build file:**

```
% ant
Buildfile: build.xml
hello: [echo] Hello, World
BUILD SUCCESSFUL
Total time: 2 seconds
```

# Ant Build File Example

```
<project default="hello">
   <target name="hello">
      <echo message="Hello, World"/>
   </target>
  <target name=”bye">
      <echo message=”goodbye, World"/>
   </target>
</project>
```

```
franky:xx devanbu$ ant bye
Buildfile: /private/tmp/xx/build.xml

bye:
     [echo] Bye, World

BUILD SUCCESSFUL
Total time: 0 seconds
franky:xx devanbu$ ant hello
Buildfile: /private/tmp/xx/build.xml

hello:
     [echo] Hello, World

BUILD SUCCESSFUL
Total time: 0 seconds
franky:xx devanbu$
```

# Ant Properties

- <property name="lib.dir" value="lib"/>
- From command line
- In build.xml
- From external XML
- From external property files
- From environment

# Ant Path, Ant Target/Task

```
<path id="classpath">
    <fileset dir="${lib.dir}"
             includes="**/*.jar"/>
</path>

<target name="compile">
    <mkdir dir="${classes.dir}"/>
    <javac srcdir="${src.dir}"
           destdir="${classes.dir}"
           classpathref="classpath"/>
</target>
```

# Ant Target

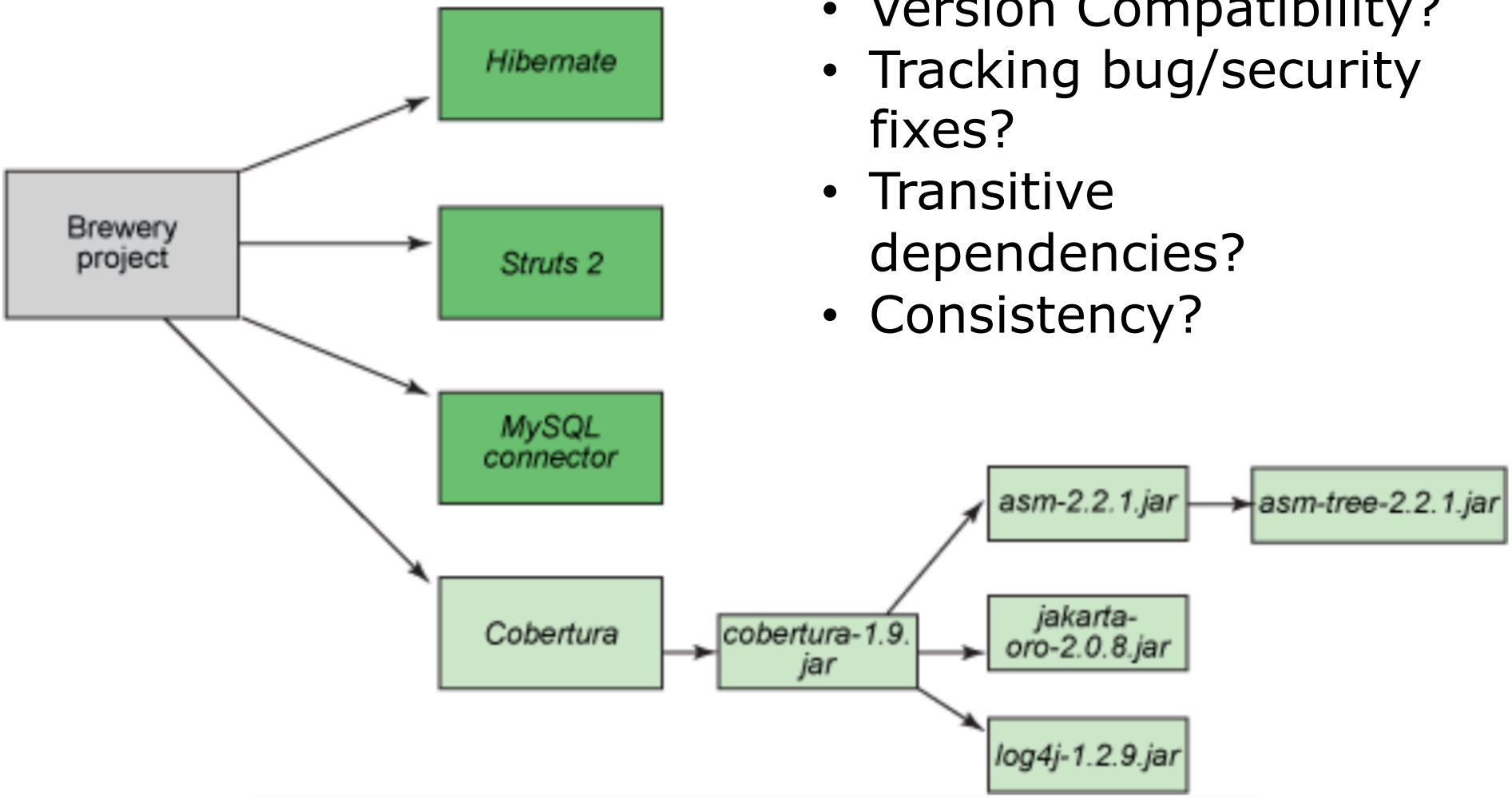
- Name
- Description
- Dependencies
- Conditionals
- <antcall> task

# Ant Tasks

- Core Tasks
- Optional Tasks
- Custom Tasks

Ant AntCall Apply Basename BuildNumber Checksum
Chmod Concat Condition Copy Cvs Delete DependSet
Dirname Ear Echo EchoXML Exec Fail Filter FixCRLF Get
Import Input Jar Java Javac Javadoc Length LoadFile
LoadProperties LoadResource MakeURL Mail MacroDef
Manifest ManifestClassPath Mkdir Move Nice Parallel
Patch PathConvert PreSetDef Property Record Replace
ResourceCount Retry Rmic Sequential SignJar Sleep Sql
Subant Sync Tar Taskdef Tempfile Touch Truncate TStamp
Typedef Unjar Untar Unwar Unzip Uptodate Waitfor
WhichResource War XmlProperty XSLT Zip

# Dependencies




- Version Compatibility?
- Tracking bug/security fixes?
- Transitive dependencies?
- Consistency?

# Imperfect techniques to manage dependencies

- Placing all dependent projects (JAR files) in a directory that's checked into the project's version-control repository.
- Allocating dependent JARs to a common file server
- Copying JAR files manually to a specific location on each developer's workstation.
- Performing an HTTP Get to download files to a developer's workstation, either manually or as part of the automated build.

# Ivy

# Defining dependencies in ivy.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet type="text/xsl" href="./config/ivy/ivy-
doc.xsl"?>
<ivy-module version="1.0">
  <info organisation="com" module="integratebutton" />
  <dependencies>
    <dependency name="hsqldb" rev="1.8.0.7" />
    <dependency name="pmd" rev="2.0" />
    <dependency name="cobertura" rev="1.9"/>
    <dependency name="checkstyle" rev="4.1" />
    <dependency name="junitperf" rev="1.9.1" />
    <dependency name="junit" rev="3.8.1" />
  </dependencies>
</ivy-module>
```

- Note: no indication of file locations or URLs
- Convention: `dependency name="cobertura" rev="1.9"` translates to cobertura-1.9.jar

# Specifying dependencies in Ivy

ivy.xml

```
<ivy-module version="2.0">
    <info organisation="org.apache" module="WebProject" />

    <dependencies>
        <dependency org="org.slf4j" name="slf4j-api" rev="1.7.6" conf="compile->default"/>
        <dependency org="jstl" name="jstl" rev="1.2" conf="compile->default"/>
        <dependency org="ch.qos.logback" name="logback-classic" rev="1.1.2" conf="compile->default"/>
        <dependency org="org.springframework" name="spring-core" rev="4.1.3.RELEASE" conf="compile->default"/>
        <dependency org="org.springframework" name="spring-beans" rev="4.1.3.RELEASE" conf="compile->default"/>
        <dependency org="org.springframework" name="spring-context" rev="4.1.3.RELEASE" conf="compile->default"/>
        <dependency org="org.springframework" name="spring-web" rev="4.1.3.RELEASE" conf="compile->default"/>
        <dependency org="org.springframework" name="spring-webmvc" rev="4.1.3.RELEASE" conf="compile->default"/>
    </dependencies>
</ivy-module>
```

- Note: no indication of file locations or URLs
- Convention: `dependency name="cobertura" rev="1.9"` translates to cobertura-1.9.jar

# Ivy settings file

```
<ivysettings>
  <settings defaultResolver="chained"/>
  <resolvers>
    <chain name="chained" returnFirst="true">
      <filesystem name="libraries">
        <artifact pattern="${ivy.conf.dir}/repository/[artifact]-[revision].[type]" />
      </filesystem>
      <url name="integratebutton">
        <artifact pattern="http://www.integratebutton.com/repo/[organisation]/[module]/
           [revision]/[artifact]-[revision].[ext]" />
      </url>
      <ibiblio name="ibiblio" />
      <url name="ibiblio-mirror">
        <artifact pattern="http://mirrors.ibiblio.org/pub/mirrors/maven2/[organisation]/
           [module]/[branch]/[revision]/[branch]-[revision].[ext]" />
      </url>
    </chain>
  </resolvers>
</ivysettings>
```

# Depending on dependencies

```
<?xml version="1.0" encoding="UTF-8"?>
<ivy-module version="2.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="http://ant.apache.org/ivy/schemas/ivy.xsd">
  <info organisation="cobertura" module="cobertura" revision="1.9"/>
  <configurations>
    <conf name="master"/>
  </configurations>

  <publications>
    <artifact name="cobertura" type="jar" conf="master" />
  </publications>

  <dependencies>
    <dependency org="objectweb" name="asm" rev="2.2.1" conf="master"/>
    <dependency org="jakarta" name="oro" rev="2.0.8" conf="master"/>
    <dependency org="apache" name="log4j" rev="1.2.9" conf="master"/>
  </dependencies>
</ivy-module>
```

Use ivy.xml within an ANT build.xml

```
<project xmlns:ivy="antlib:org.apache.ivy.ant" name="HelloProject" default="main" basedir=".">
    <description>
        Create a Spring MVC (WAR) with Ant build script
    </description>

    <!-- Project Structure -->
    <property name="jdk.version" value="1.7" />
    <property name="projectName" value="WebProject" />
    <property name="src.dir" location="src" />
    <property name="resources.dir" location="resources" />
    <property name="web.dir" value="war" />
    <property name="web.classes.dir" location="${web.dir}/WEB-INF/classes" />
    <property name="target.dir" location="target" />
    <property name="target.temp.dir" location="target/temp" />
    <property name="lib.dir" value="lib" />

    <!-- ivy start -->
    <target name="resolve" description="retrieve dependencies with ivy">
        <echo message="Getting dependencies..." />
        <ivy:retrieve />

        <ivy:cachepath pathid="compile.path" conf="compile" />
        <ivy:cachepath pathid="runtime.path" conf="runtime" />
        <ivy:cachepath pathid="test.path" conf="test" />

    </target>

    <!-- install ivy if you don't have ivyide-->
    <target name="ivy" description="Install ivy">
        <mkdir dir="${user.home}/.ant/lib" />
        <get dest="${user.home}/.ant/lib/ivy.jar"
        src="http://search.maven.org/remotecontent?filepath=org/apache/ivy/ivy/2.4.0-rc1/ivy-2.4.0-rc1.jar" />
    </target>
    <!-- ivy end -->

    <!-- Compile Java source from ${src.dir} and output it to ${web.classes.dir} -->
    <target name="compile" depends="init, resolve" description="compile source code">
        <mkdir dir="${web.classes.dir}" />
        <javac destdir="${web.classes.dir}" source="${jdk.version}" target="${jdk.version}"
            debug="true" includeantruntime="false" classpathref="compile.path">
```

# Maven

# Main Ideas of Maven

- “Convention over Configuration”
- DESCRIBE, don’t IMPLEMENT.
- Reuse build logic & standards whenever possible (mostly done as “Maven plugins”)
- Organize dependencies clearly, logically, aesthetically

# Main Benefits of Maven

- Reuse across multiple projects on the same platform.
- Smaller, more standardized, reusable, build procedures.
- Spend less time on Build, more time Coding Apps

# A simple Java app

- `mvn archetype:generate -DgroupId=edu.cmu.cs -DartifactId=hello -DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false`
  - What's in the directory structure?

- `Mvn compile`

- `Run: java -cp target/classes edu.cmu.cs.App`

- `mvn clean`

- `Mvn test` (see test results)
- `Mvn package`
- `Java –cp target/*jar edu.cmu.cs.App`

# Gradle

# Task-Based Managers: Gradle

- Combines the best of Ant and Maven
- From Ant keep:
  - Portability: Build commands described platform-independently
  - Flexibility: Describe almost any sequence of processing steps
- … but drop:
  - XML as build language, inability to express simple control flow
- From Maven keep:
  - Dependency management
  - Standard directory layouts & build conventions for common project types
- … but drop:
  - XML, inflexibility, inability to express simple control flow

# Summary

# What every build system must do:

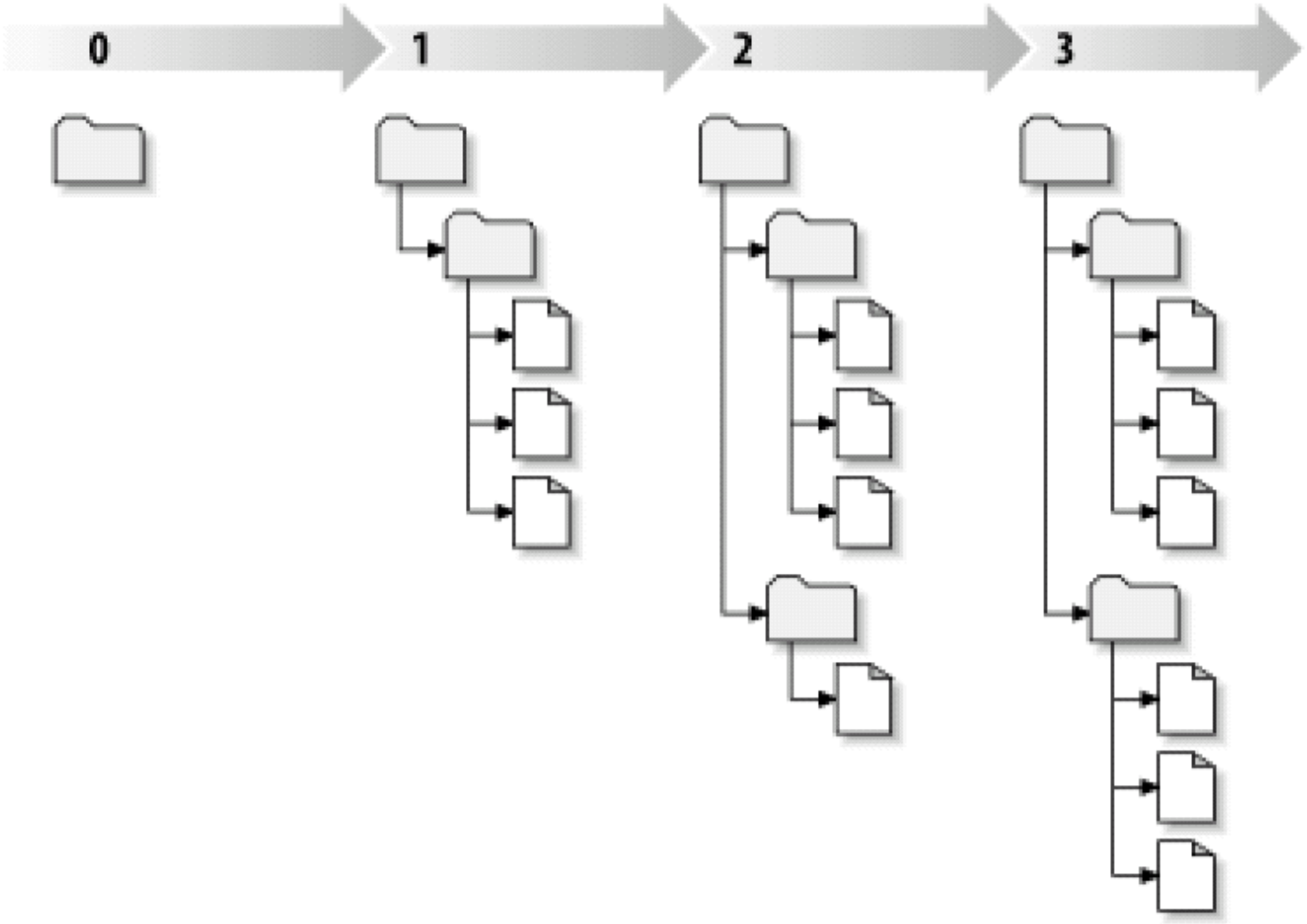
- Manage dependencies within-project.
- Manage dependencies for outside libraries.
- Maintain consistency and versioning.
- Know tasks that ”complete” the dependencies.
- Deal with complex directory structures and many types of files.
- Be as simple as possible, but no simpler.

- *How do each of the build systems we discussed do at all this?*

# Building a project should be repeatable and automated

- All but the smallest projects have a nontrivial build process
- You want to capture and automate the knowledge of how to build your system, ideally in a single command
- Build scripts are code (executable specifications) that need to be managed just like other pieces of code
- Use a build tool to script building, packaging, testing, and deploying your system
  - Most IDEs have an integrated build system

# Versioning entire projects



# Which files to manage

- All code and noncode files
  - Java code
  - Build scripts
  - Documentation
- Exclude generated files (.class, …)
- Most version control systems have a mechanism to exclude files (e.g., .gitignore)

# COLLABORATION

# Collaborating on Files

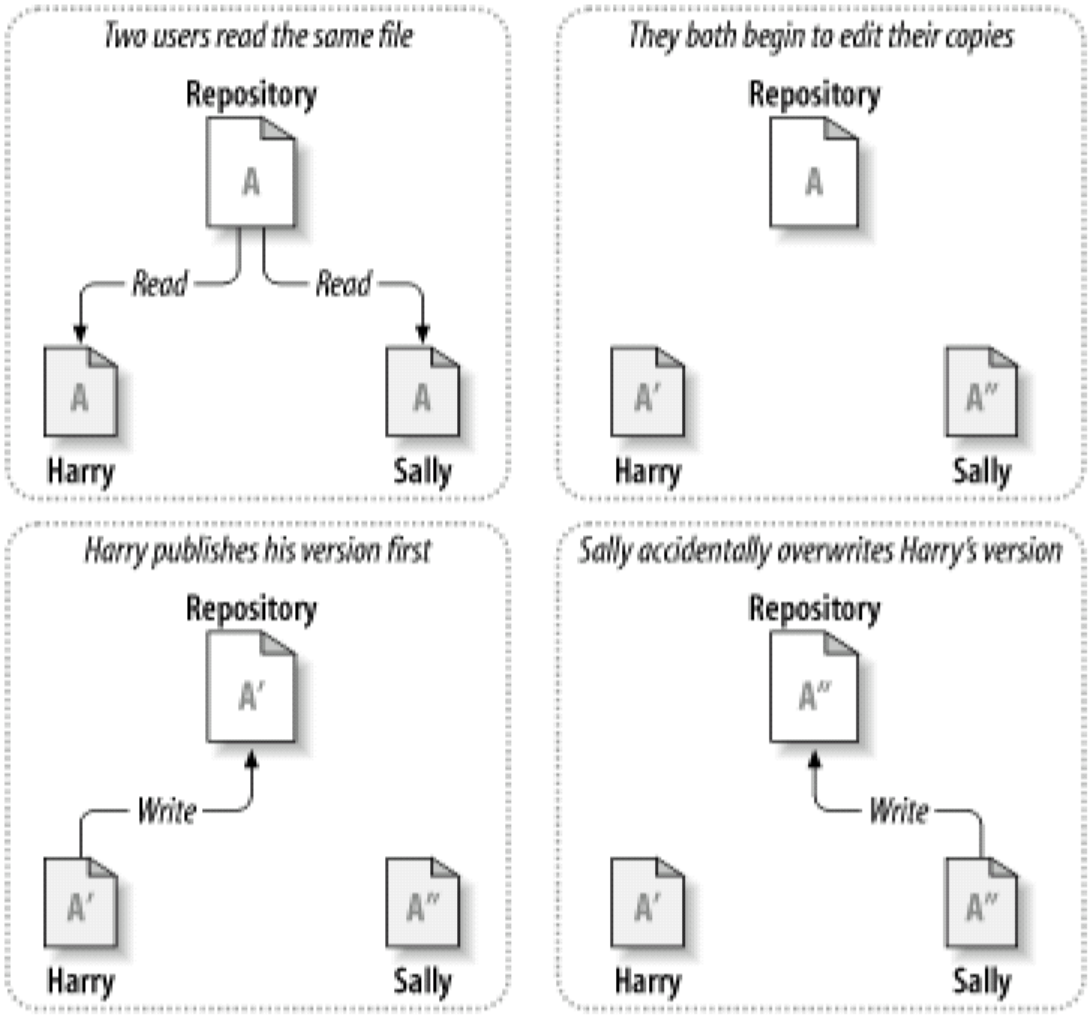
- How to exchange files
  - Send changes by email
  - Manual synchronization at project meeting
  - All files on shared network directory

- Permission models
  - Each file has an owner; only person allowed to change it
  - Everybody may change all files (collective ownership)

# Concurrent Modifications

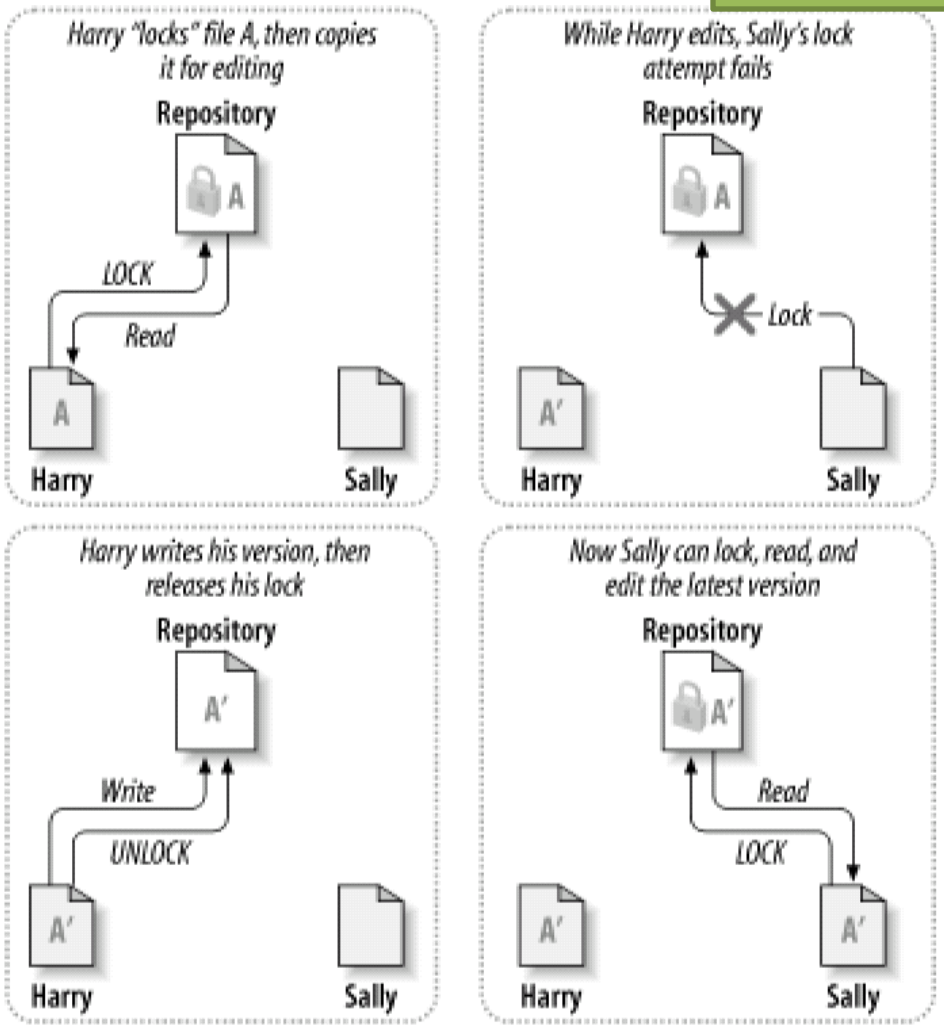
- Allowing concurrent modifications is challenging
- Conflicts (accidental overwriting) may occur
- Common strategies
  - Locking to change
  - Detecting conflicts (optimistic model)

# Change Conflicts



# Locking Files

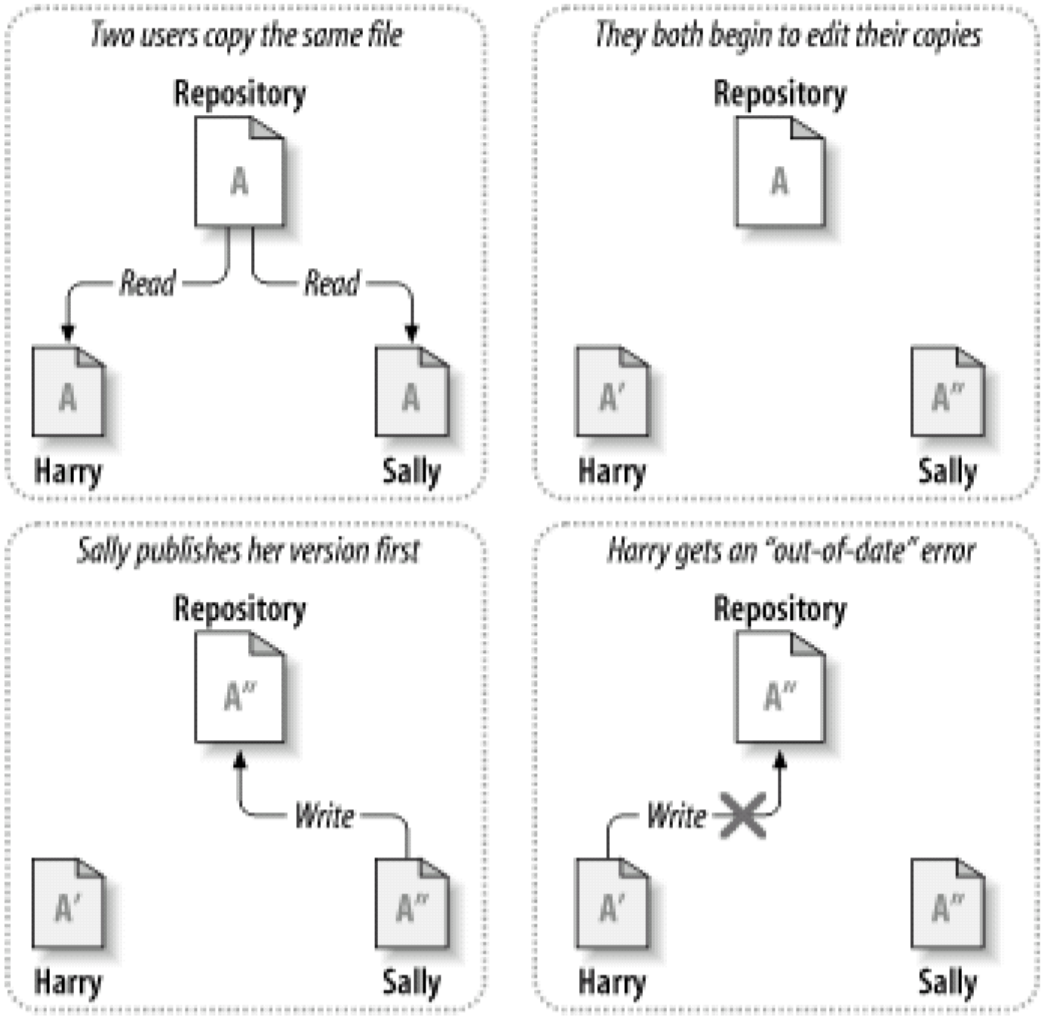
**Practical problems of locking model?**



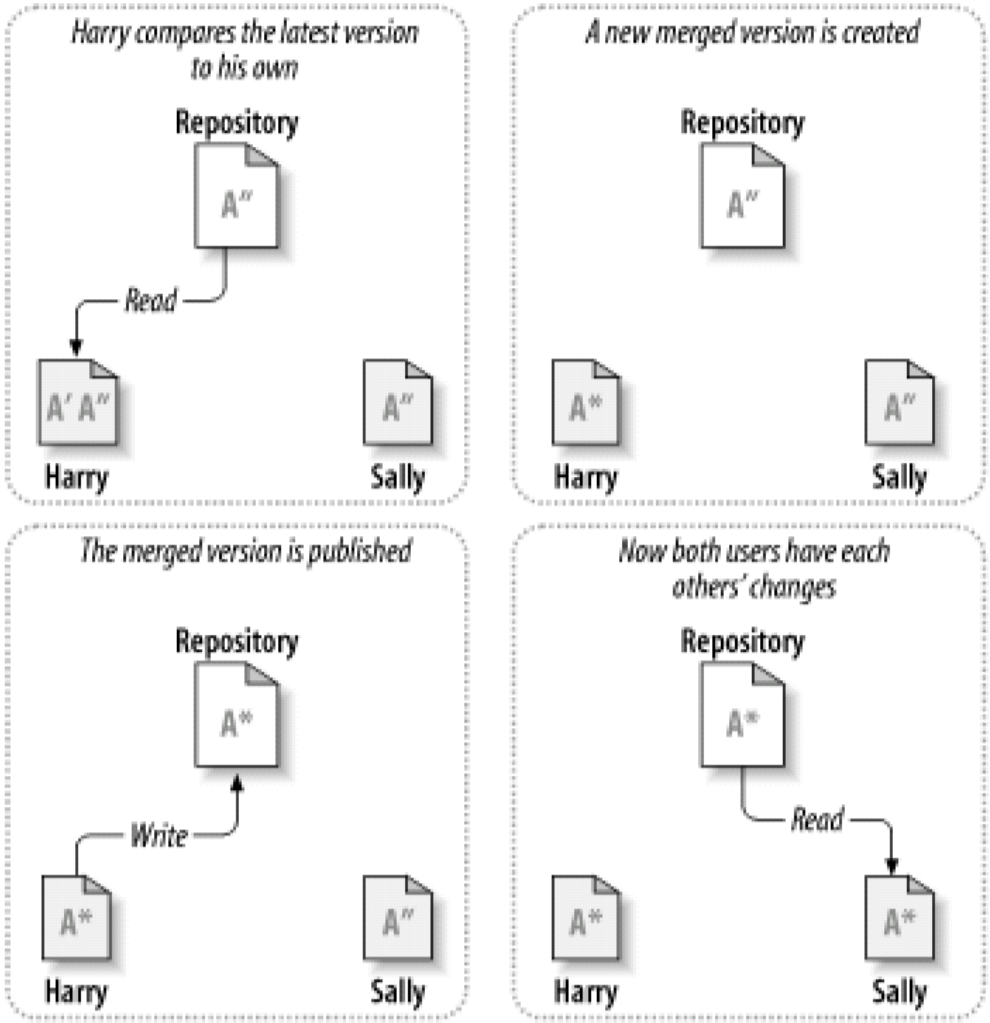
# Locking Problems

- How to lock?
  - Central system vs announcement on mailing list
  - Forgetting to unlock common
- Unnecesary sequentializing
  - Cannot work on different concepts in same file
- False sense of security
  - Changing dependant files can cause conflicts not prevented by locking

# Merging (1/2)



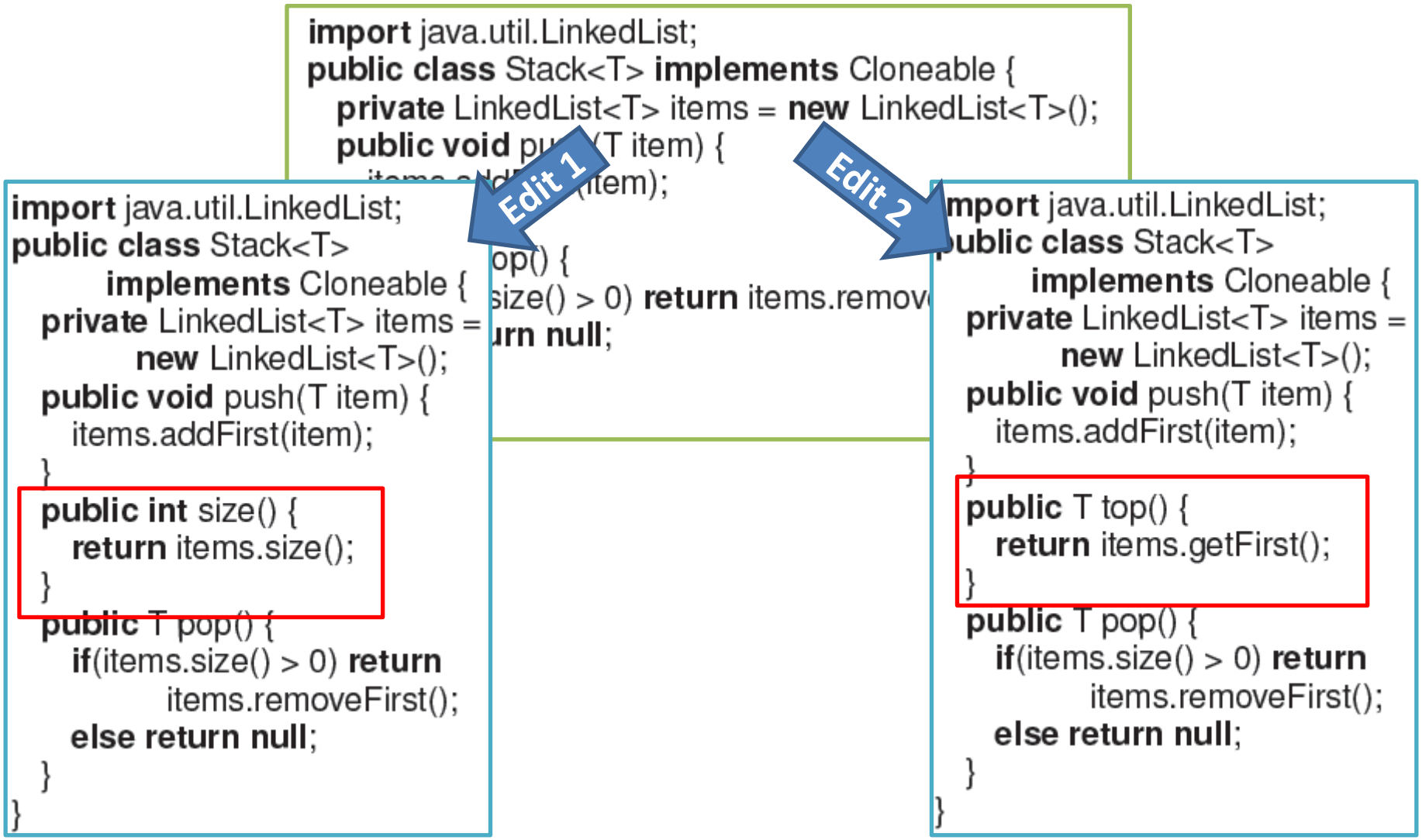
# Merging (2/2)



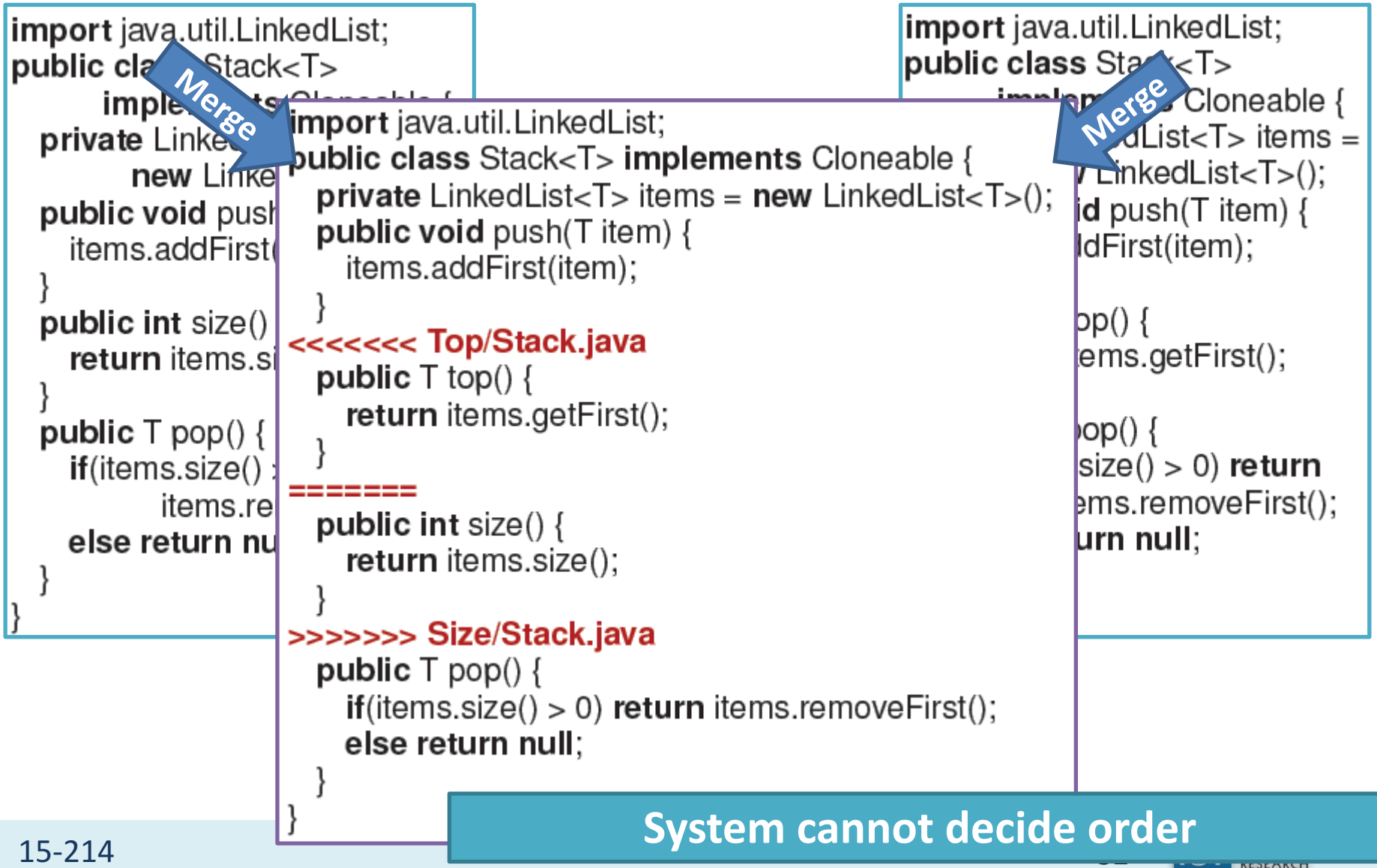
# Example

```
import java.util.LinkedList;
public class Stack<T> implements Cloneable {
   private LinkedList<T> items = new LinkedList<T>();
   public void push(T item) {
      items.addFirst(item);
   }
   public T pop() {
      if(items.size() > 0) return items.removeFirst();
      else return null;
   }
}
```

# Example



# Example


Merge
Merge
import java.util.LinkedList;
public class Stack<T> implements Cloneable {
private LinkedList<T> items = new LinkedList<T>();
public void push(T item) {
items.addFirst(item);
}
<<<<<<< Top/Stack.java
public T top() {
return items.getFirst();
}
=======
public int size() {
return items.size();
}
>>>>>>> Size/Stack.java
public T pop() {
if(items.size() > 0) return items.removeFirst();
else return null;
}
}
System cannot decide order

# 3-way merge

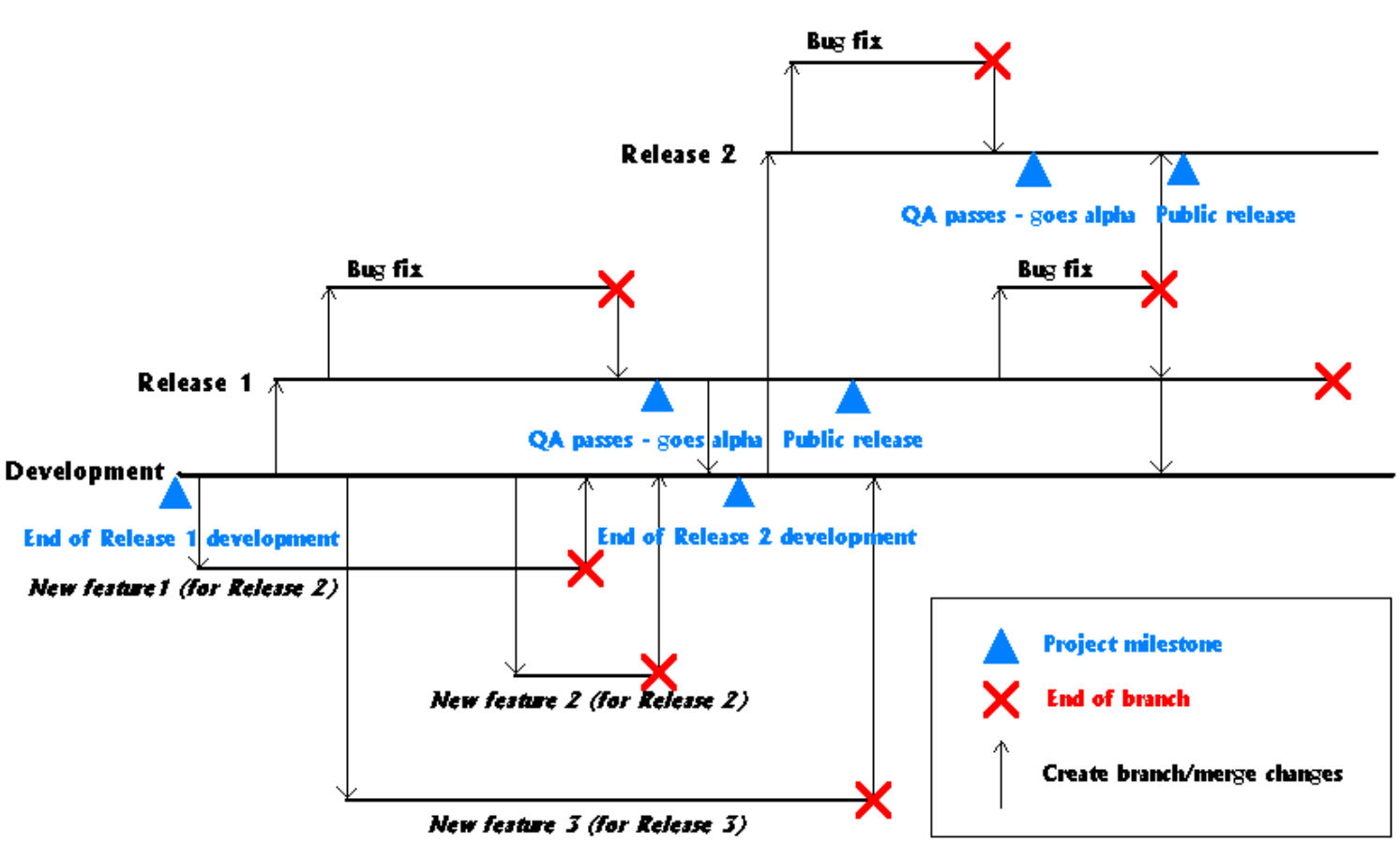
- File changed in two ways
  - Overlapping changes -> conflicts
  - Merge combines non-conflicting changes from both
- Merging not always automatic
  - diff tool to show changes
  - Manual resolution of conflicts during merge (potentially requires additional communication)
- Automatic merge potentially dangerous
  -> syntactic notion of conflicts
- Merging of binary files difficult
- In practice: most merges are conflict free

# BRANCHING

# Branching

- Parallel copies of the source tree
- Can be changed independently, versioned separately, and merged later (or left separate)
- Often used for exploratory changes or to isolate development activities
- Many usage patterns, common:
  - Main branch for maintenance OR main development
  - New branches for experimental features; merge when successful
  - New branches for nontrivial maintenance work
  - Branches for maintenance of old versions

# Release management with branches



# Variants and Revisions

- **Revision** replaces prior revision (temporal)
- **Variant** coexists with other variants
- **Version** describes both
- **Release:** Published and named version

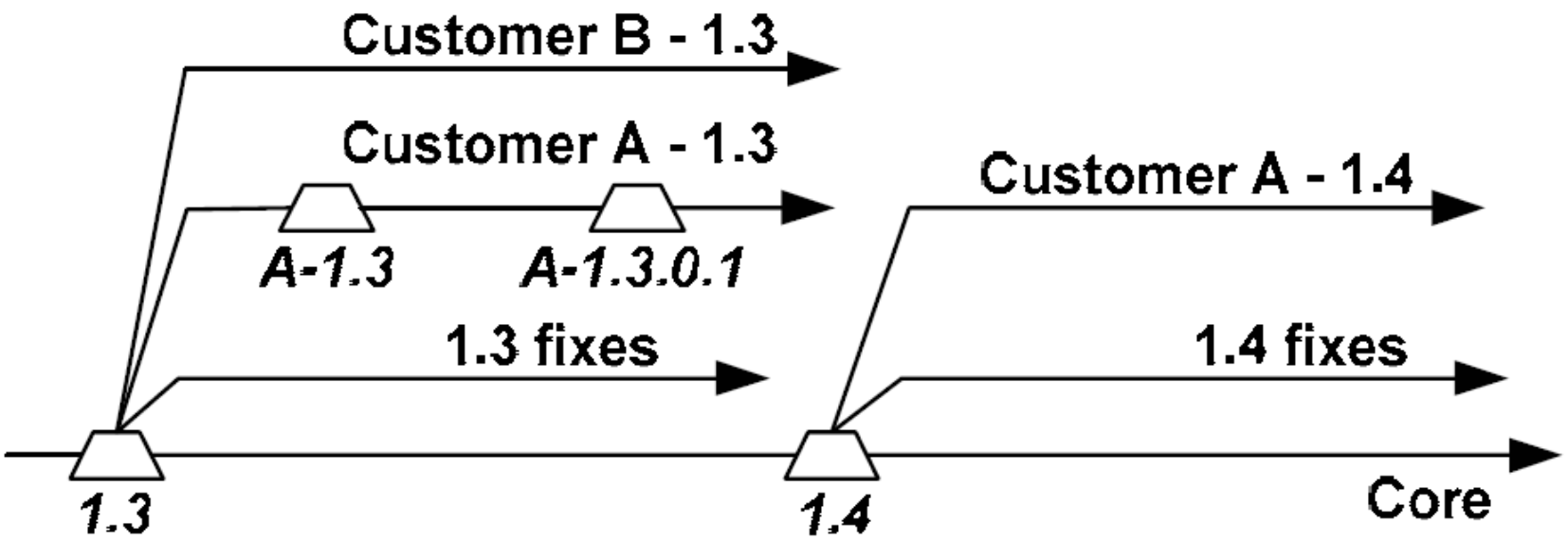
| | V1.0 | V1.1 | V2.0 | V3.0 |
|---|---|---|---|---|
| Base system (Windows) | X | X | X | X |
| Linux variant | | X | X | |
| Server variant | | | X | X |
| Extension for customer A | | X | X | X |
| Extension for customer B | | | | X |

# Semantic Versioning for Releases

- Given a version number MAJOR.MINOR.PATCH, increment the:
  - MAJOR version when you make incompatible API changes,
  - MINOR version when you add functionality in a backwards-compatible manner, and
  - PATCH version when you make backwards-compatible bug fixes.
- Additional labels for pre-release and build metadata are available as extensions to the MAJOR.MINOR.PATCH format.

**http://semver.org/**

# Variants and Revisions




[Staples&Hill, APSEC'04]

# Managing variants

- Branching for variants does not scale well
- Requires special planning or tooling

- Many solutions
  - Configuration files
  - OO polymorphism
  - Preprocessors
  - Build systems
  - DSLs
  - Software product lines
  - …

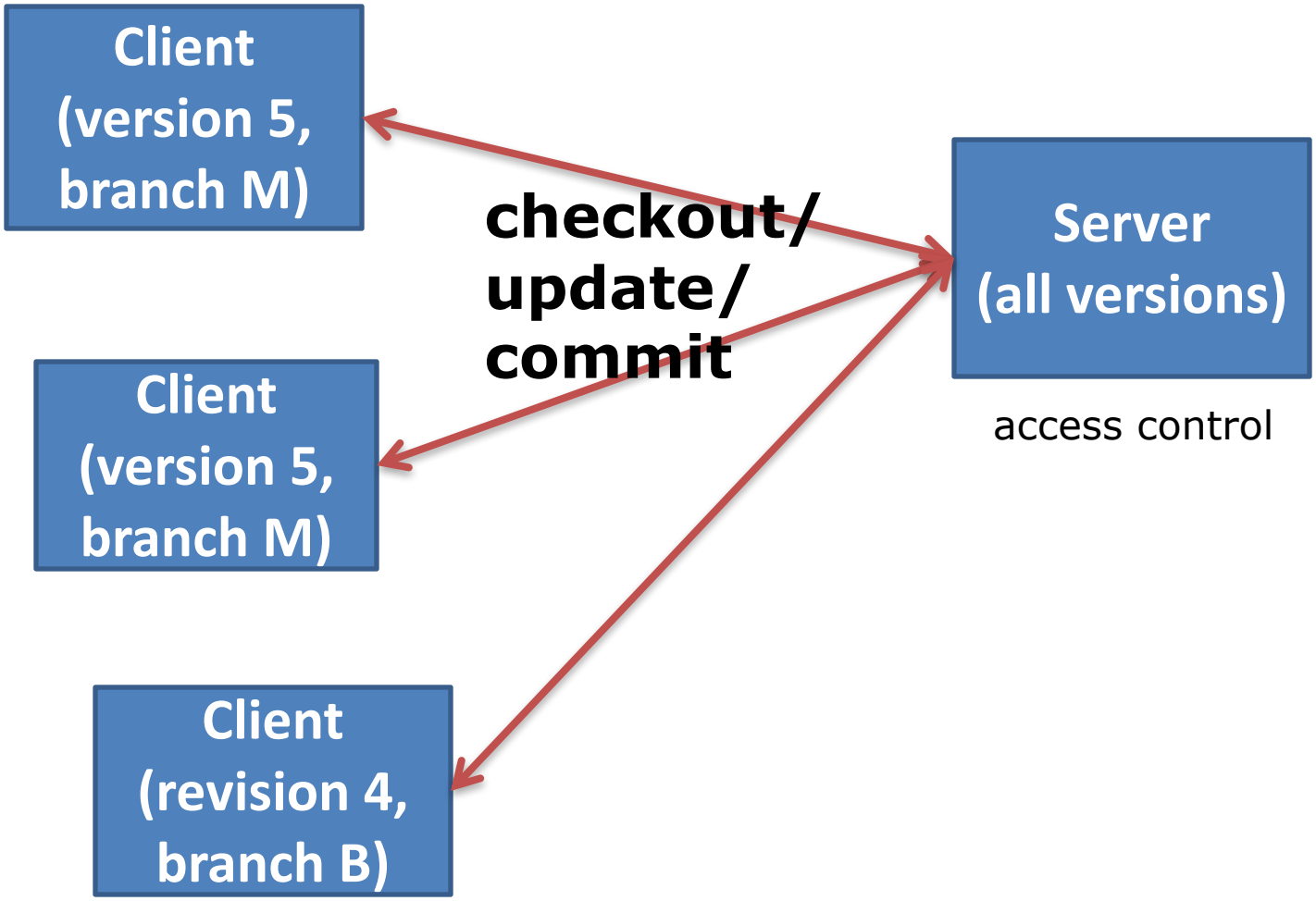
```
/* common parts */
...
/* dependent on operating system */
#if (OS == Unix)
...
#elif (OS == VMS)
...
#else
...
#endif
...
```

# CENTRALIZED VERSION CONTROL (E.G., SVN)

# Classes of version control systems

- Systems supporting merging and/or locking
- Local version control
  - Local history of files: SCCS (1970s), RCS (1982)
- Central version control
  - Versions stored on central master server
  - Clients synchronize with server (update, commit)
  - CVS (1990), SVN (2004), Perforce, Visual SourceSafe
- Distributed version control
  - Many repositories; synchronization among repositories (push, pull)
  - Git (2005), Mercurial, Bitkeeper, ClearCase

# Centralized Version Control



# Typical work cycle

- Once: Create local workspace
  - svn checkout
- Update workspace:
  - svn update
- Perform changes in workspace:
  - svn add
  - svn delete
  - svn copy
  - svn move
- Show workspace changes:
  - svn status
  - svn diff
- Revert changes in workspace:
  - svn revert
- Update and merge conflicts:
  - svn update
  - svn resolved
- Push workspace changes to server:
  - svn commit

# CVS vs. SVN

**CVS**

- Improvement over RCS in tracking entire directories
- Revision number per file
- Text files (binary files possible)

**SVN**

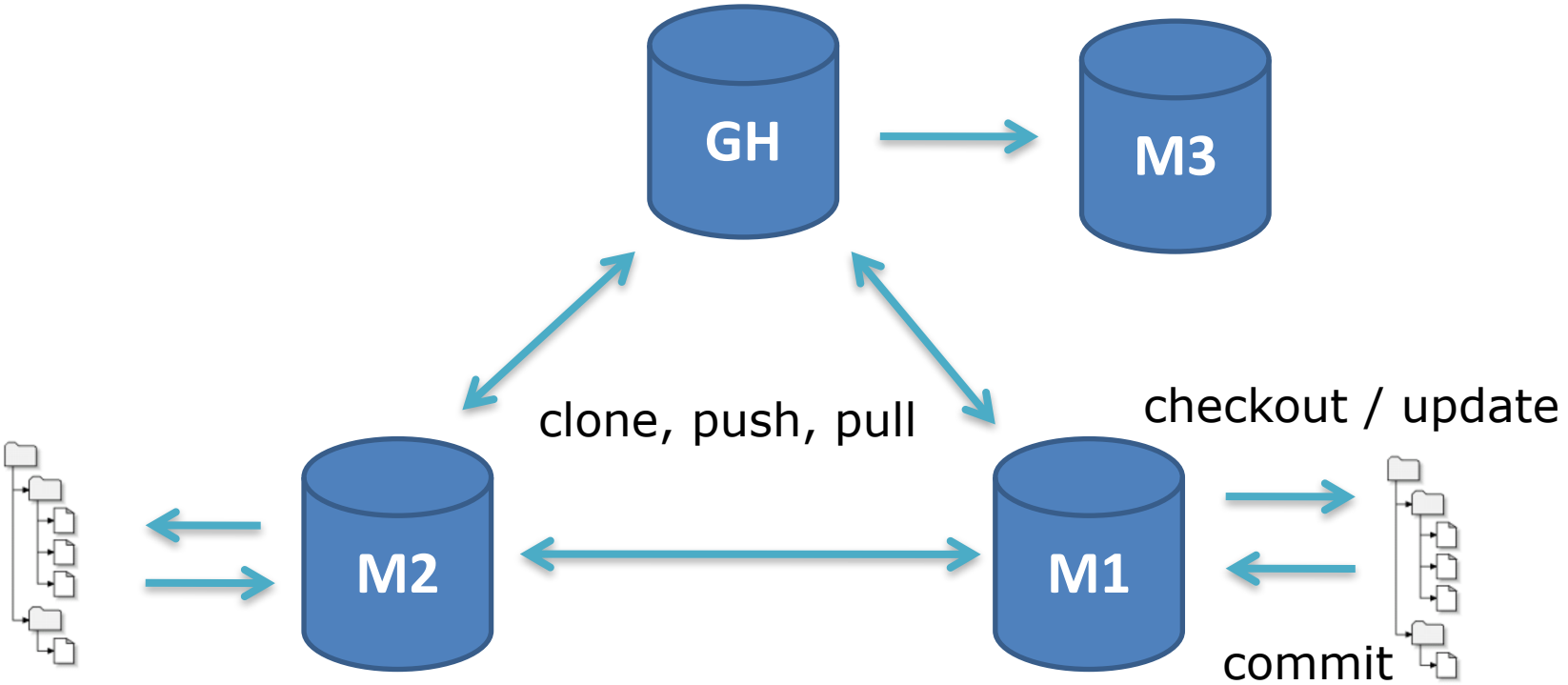
- Revision numbers for project
- Atomic commits (commiting multiple files at once)
- Tracking files and directories
- Support renaming
- Tracking of Metadata

# DISTRIBUTED VERSION CONTROL (E.G., GIT)

# Git

- Distributed version control
- No central server necessary (but possible)
- Local copies of repositories (containing all history)
  - Locally SVN like functionality: checkout, update, commit, branch, diff
- Nonlinear development: each local copy can evolve independently
- Synchronization among repositories (push/fetch/pull)
- Fast local operations (branch, commit, diff, ...)

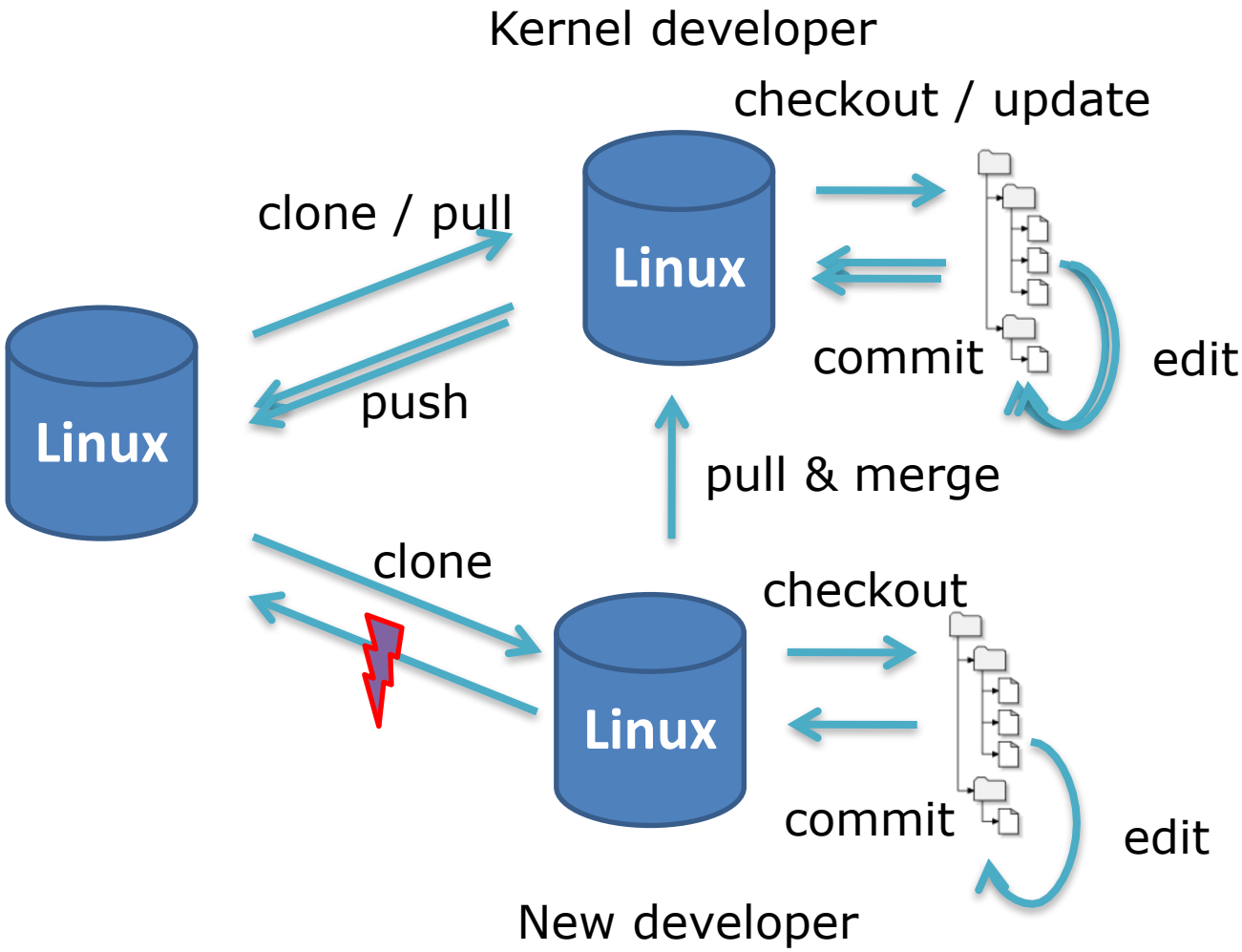
# Overview



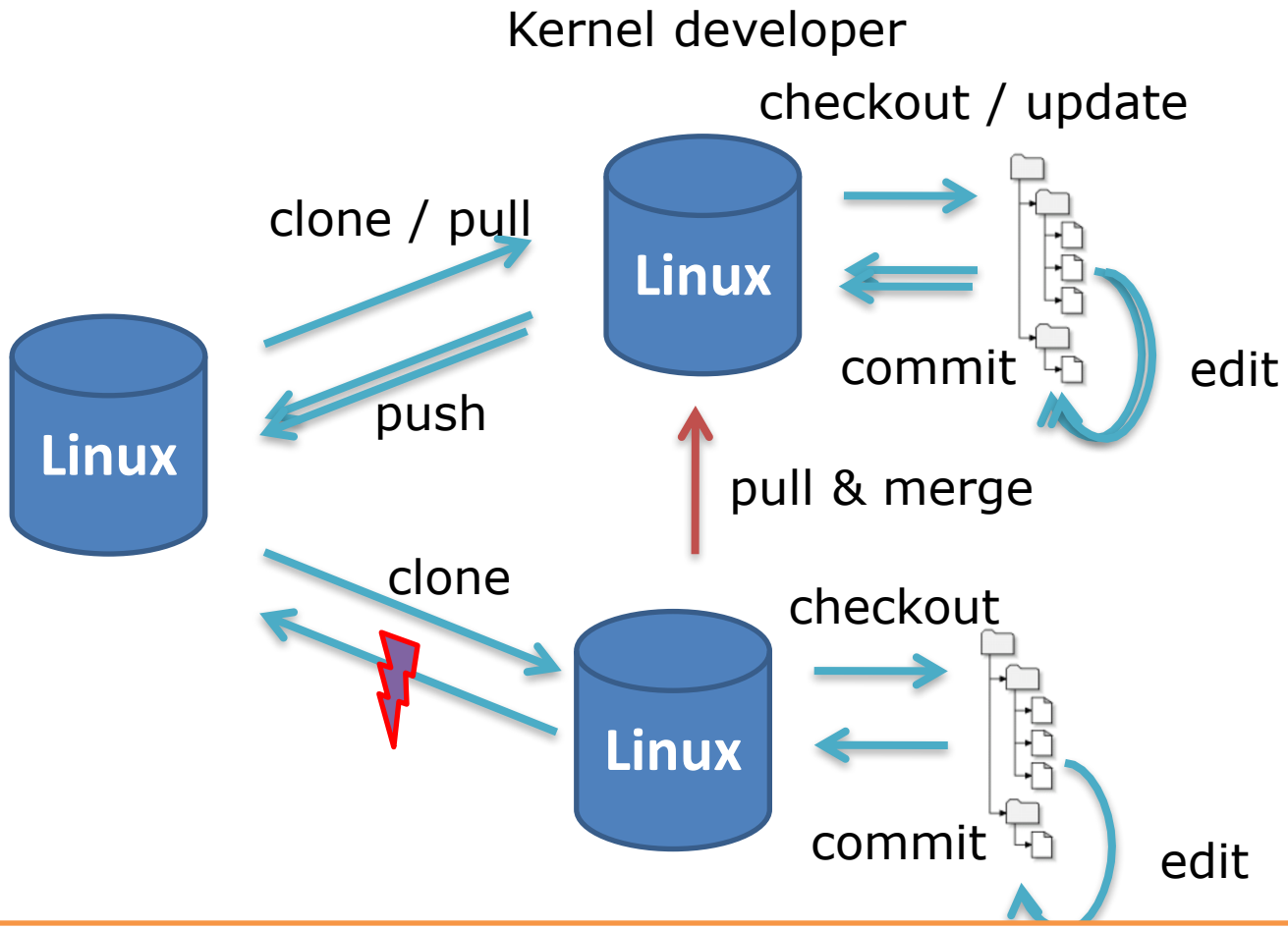
# Distributed Versions

- Versions not globally coordinated/sorted
- Unique IDs through hashes, relationships tracked in successor graph
  - e.g., 52a0ff44aba8599f43a5d821c421af316cb7305
  - Possible to merge select changes (cherry picking)
  - Possible to rewrite the history as long as not shared remotely (git rebase etc)
- Cloning creates copy of repository (including all versions)
  - Tracks latest state when cloned, relevant for updating and merging
  - Normal checkout and commit operations locally
  - Commits don't change original repository
- Fetch and pull get missing versions from remote repository (one or more)
- Push operations sends local changes to remote repository (one or more), given access rights

# Example workflow

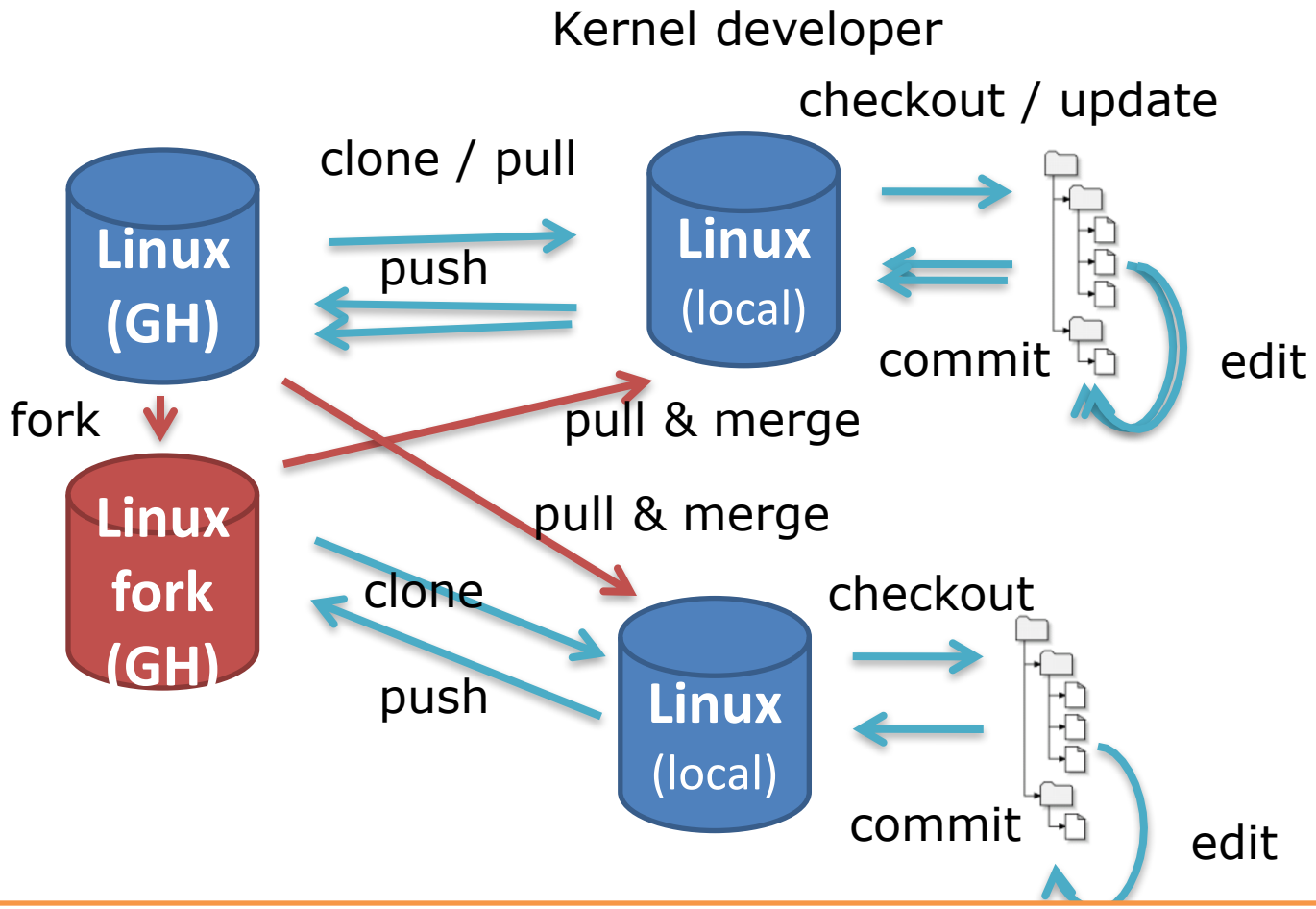


# Pull Requests



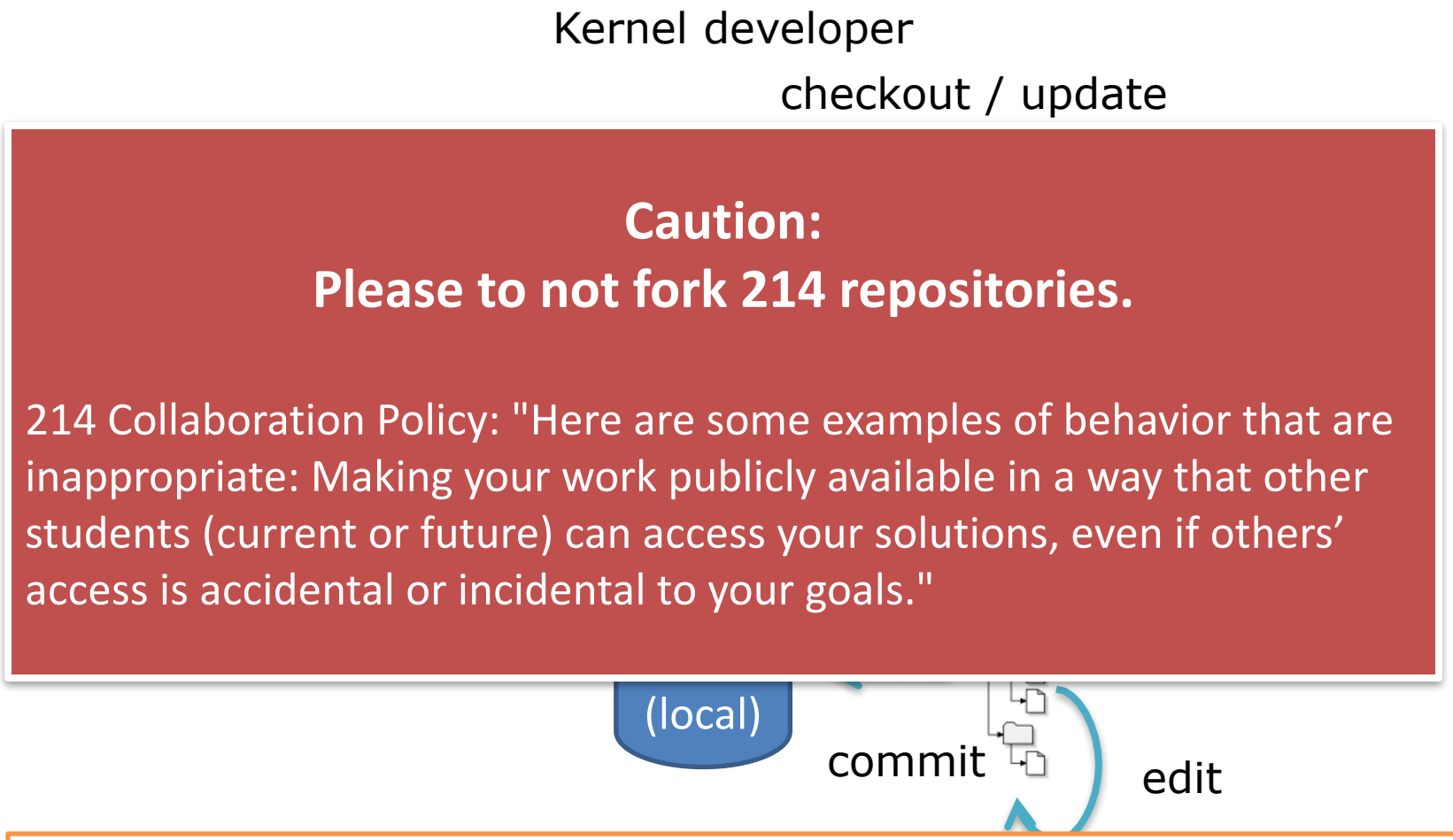

Pull request: Github feature to ask developer to pull a specific change (alternative to sending email); integration with Travis CI

# Forks



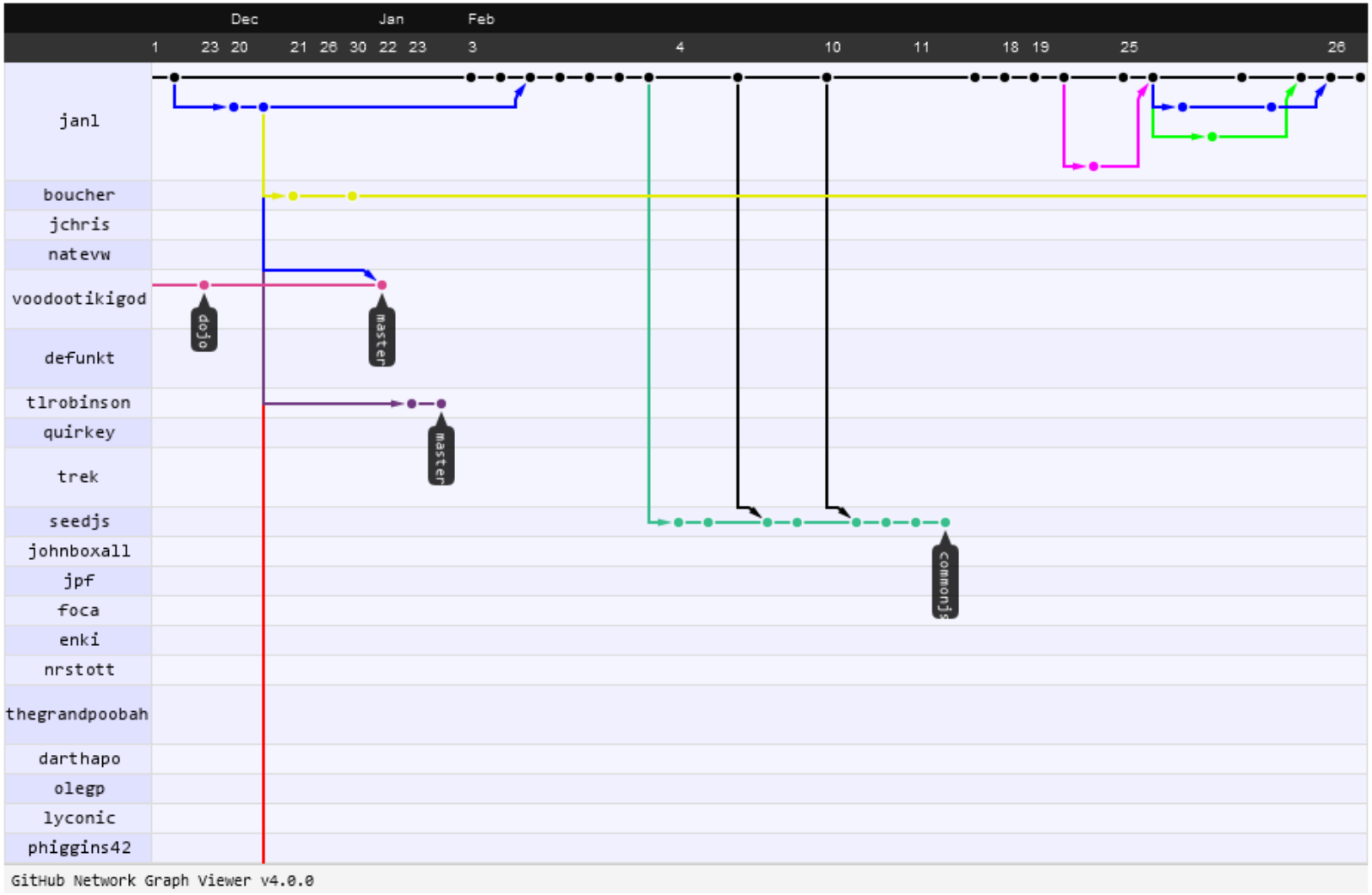

Fork: Github feature to clone repository on Github (own copy with full rights)

# Forks

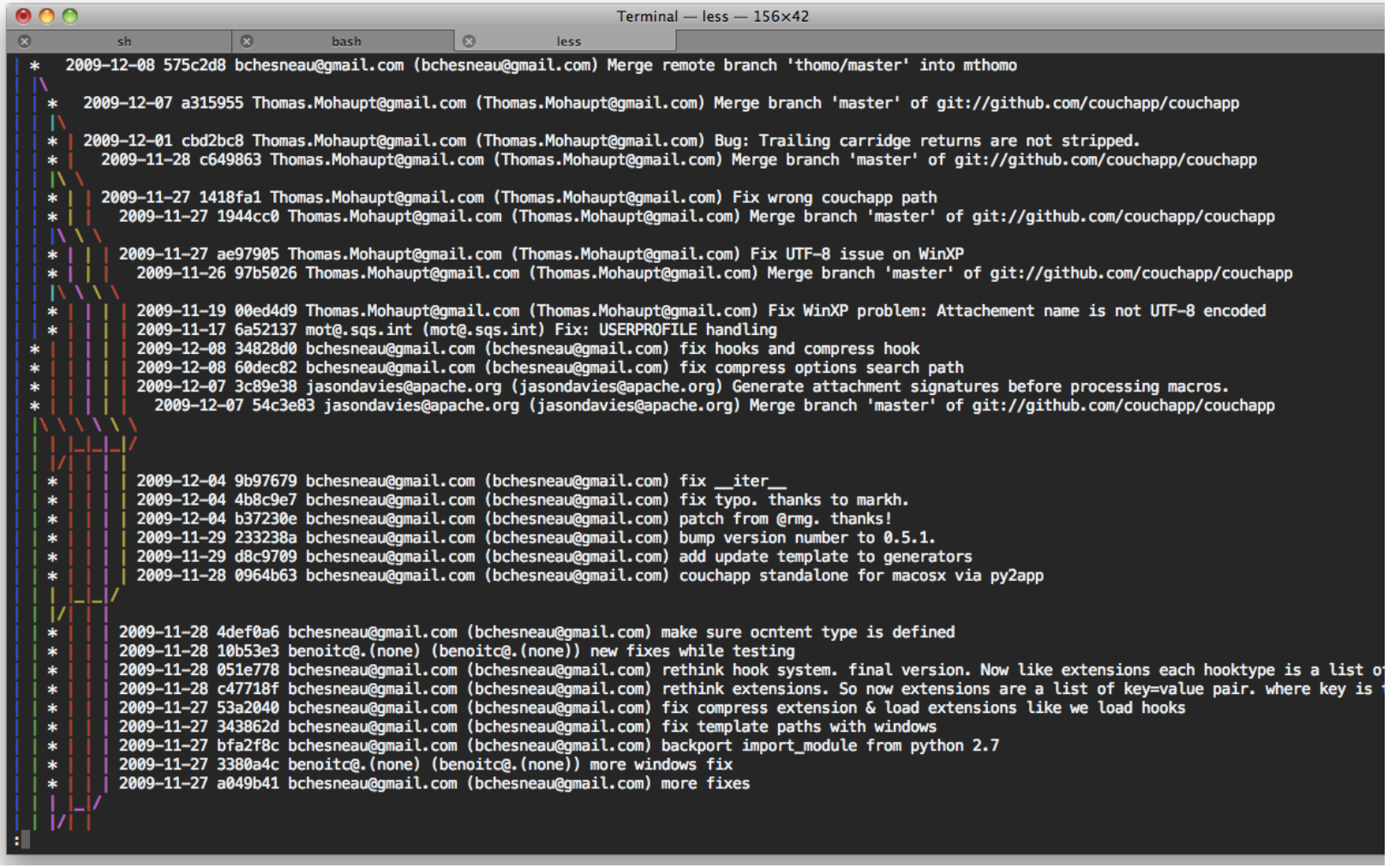



Fork: Github feature to clone repository on Github (own copy with full rights)

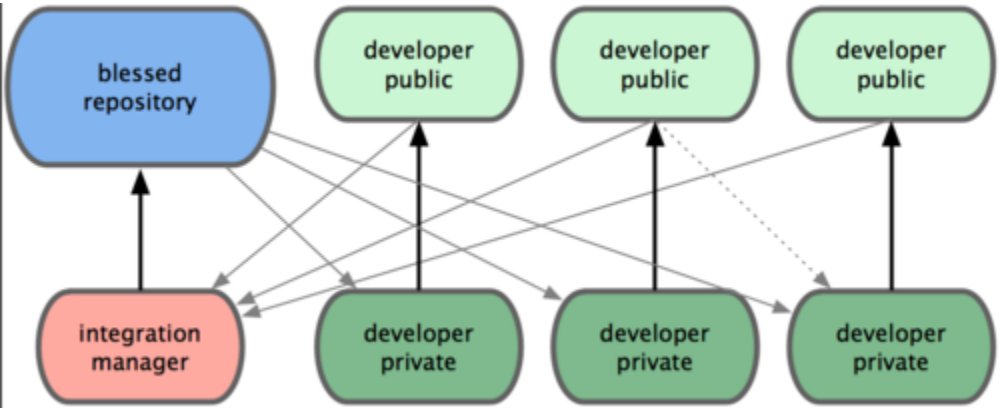
# Repositories in mustache.js

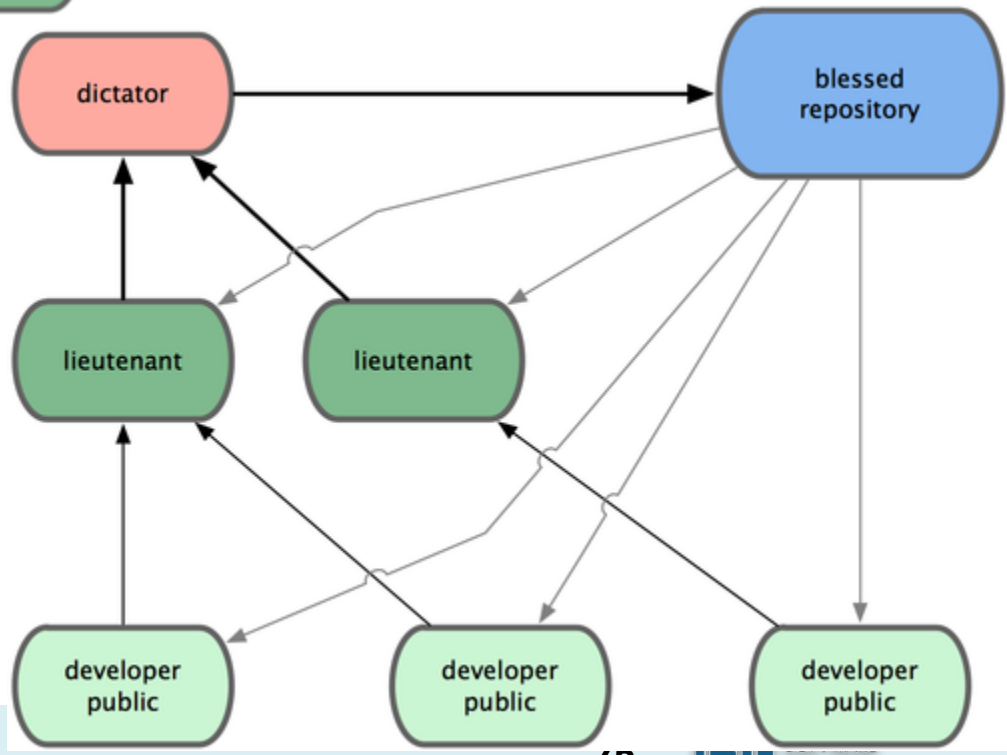


# Git History


Terminal — less — 156×42
sh
bash
less
2009-12-08 575c2d8 bchesneau@gmail.com (bchesneau@gmail.com) Merge remote branch 'thomo/master' into mthomo
2009-12-07 a315955 Thomas.Mohaupt@gmail.com (Thomas.Mohaupt@gmail.com) Merge branch 'master' of git://github.com/couchapp/couchapp
2009-12-01 cbd2bc8 Thomas.Mohaupt@gmail.com (Thomas.Mohaupt@gmail.com) Bug: Trailing carridge returns are not stripped.
2009-11-28 c649863 Thomas.Mohaupt@gmail.com (Thomas.Mohaupt@gmail.com) Merge branch 'master' of git://github.com/couchapp/couchapp
2009-11-27 1418fa1 Thomas.Mohaupt@gmail.com (Thomas.Mohaupt@gmail.com) Fix wrong couchapp path
2009-11-27 1944cc0 Thomas.Mohaupt@gmail.com (Thomas.Mohaupt@gmail.com) Merge branch 'master' of git://github.com/couchapp/couchapp
2009-11-27 ae97905 Thomas.Mohaupt@gmail.com (Thomas.Mohaupt@gmail.com) Fix UTF-8 issue on WinXP
2009-11-26 97b5026 Thomas.Mohaupt@gmail.com (Thomas.Mohaupt@gmail.com) Merge branch 'master' of git://github.com/couchapp/couchapp
2009-11-19 00ed4d9 Thomas.Mohaupt@gmail.com (Thomas.Mohaupt@gmail.com) Fix WinXP problem: Attachement name is not UTF-8 encoded
2009-11-17 6a52137 mot@.sqs.int (mot@.sqs.int) Fix: USERPROFILE handling
2009-12-08 34828d0 bchesneau@gmail.com (bchesneau@gmail.com) fix hooks and compress hook
2009-12-08 60dec82 bchesneau@gmail.com (bchesneau@gmail.com) fix compress options search path
2009-12-07 3c89e38 jasondavies@apache.org (jasondavies@apache.org) Generate attachment signatures before processing macros.
2009-12-07 54c3e83 jasondavies@apache.org (jasondavies@apache.org) Merge branch 'master' of git://github.com/couchapp/couchapp
2009-12-04 9b97679 bchesneau@gmail.com (bchesneau@gmail.com) fix __iter__
2009-12-04 4b8c9e7 bchesneau@gmail.com (bchesneau@gmail.com) fix typo. thanks to markh.
2009-12-04 b37230e bchesneau@gmail.com (bchesneau@gmail.com) patch from @rmg. thanks!
2009-11-29 233238a bchesneau@gmail.com (bchesneau@gmail.com) bump version number to 0.5.1.
2009-11-29 d8c9709 bchesneau@gmail.com (bchesneau@gmail.com) add update template to generators
2009-11-28 0964b63 bchesneau@gmail.com (bchesneau@gmail.com) couchapp standalone for macosx via py2app
2009-11-28 4def0a6 bchesneau@gmail.com (bchesneau@gmail.com) make sure ocntent type is defined
2009-11-28 10b53e3 benoitc@.(none) (benoitc@.(none)) new fixes while testing
2009-11-28 051e778 bchesneau@gmail.com (bchesneau@gmail.com) rethink hook system. final version. Now like extensions each hooktype is a list o
2009-11-28 c47718f bchesneau@gmail.com (bchesneau@gmail.com) rethink extensions. So now extensions are a list of key=value pair. where key is
2009-11-27 53a2040 bchesneau@gmail.com (bchesneau@gmail.com) fix compress extension & load extensions like we load hooks
2009-11-27 343862d bchesneau@gmail.com (bchesneau@gmail.com) fix template paths with windows
2009-11-27 bfa2f8c bchesneau@gmail.com (bchesneau@gmail.com) backport import_module from python 2.7
2009-11-27 3380a4c benoitc@.(none) (benoitc@.(none)) more windows fix
2009-11-27 a049b41 bchesneau@gmail.com (bchesneau@gmail.com) more fixes

# Git and Central Repositories




© Scott Chacon "Pro Git"

# Social Coding

**Eileen M. Uchitelle**
eileencodes

Platform Systems Team **@GitHub** & **@Rails** Core Team

Follow

Block or report user

GitHub **Staff**
New York
http://eileencodes.com

**Organizations**

Overview | Repositories 81 | Stars 199 | Followers 519 | Following 20

Pinned repositories

**rails/rails**
Ruby on Rails
Ruby ★ 35.5k 14.5k

**rack/rack**
a modular Ruby webserver interface
Ruby ★ 3.1k 1.2k

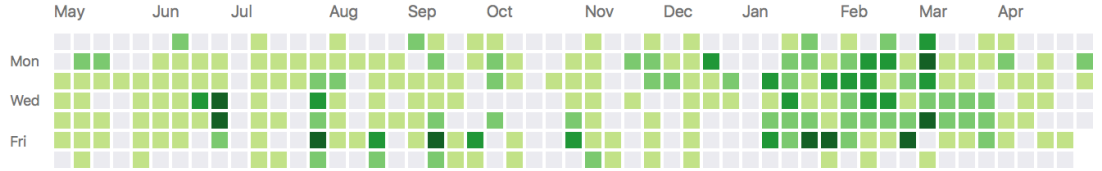
**eileencodes.github.io**
Code for eileencodes.com
CSS ★ 4 2

**security_examples**
Rails application to demo CSRF, XSS, and XXE vulnerabilities and how to avoid them
Ruby ★ 34 3

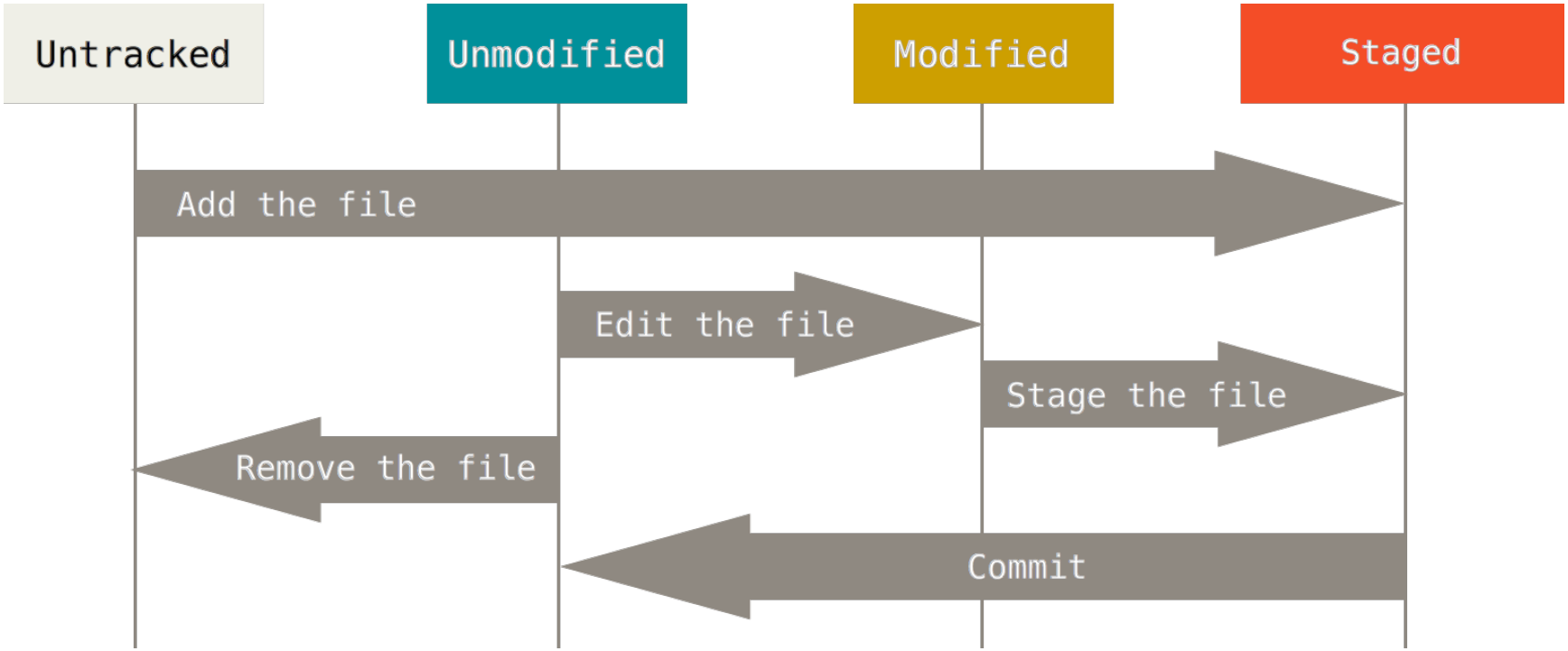
**integration_performance_test**
Application to test performance of integration versus controller tests
Ruby ★ 28 7

**basecamp/bc3-api**
API documentation for Basecamp 3
★ 152 24

1,176 contributions in the last year

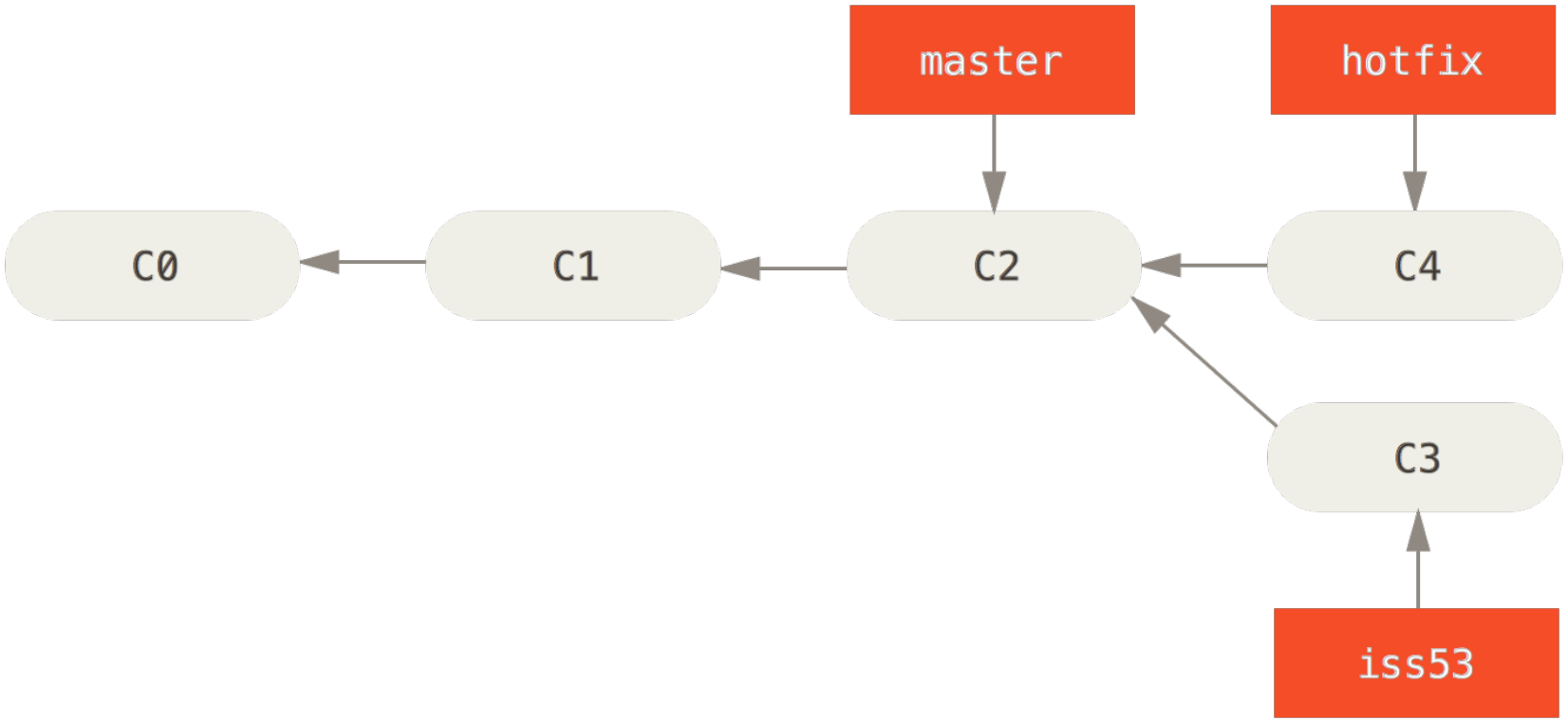


# Git Internals



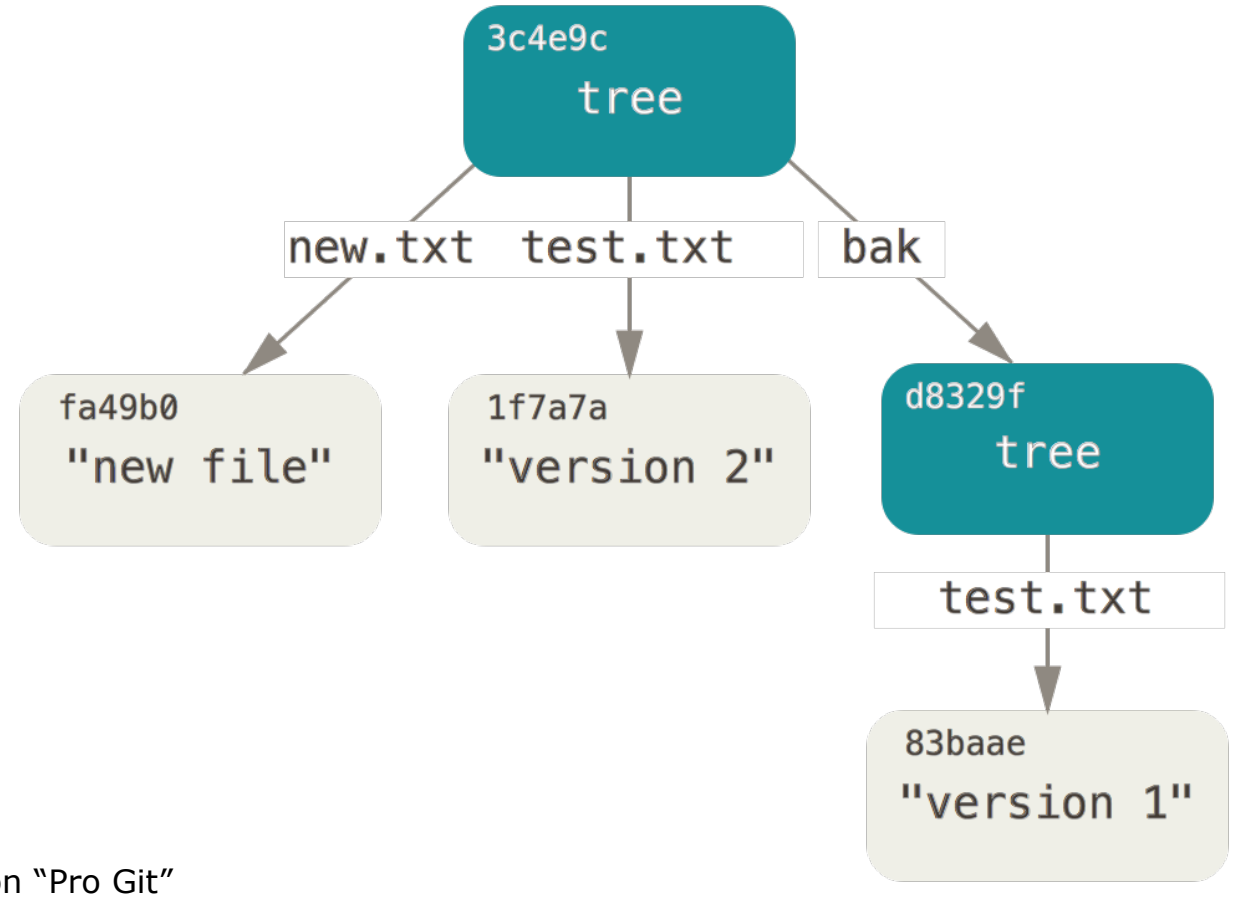

© Scott Chacon "Pro Git"

# Git Internals



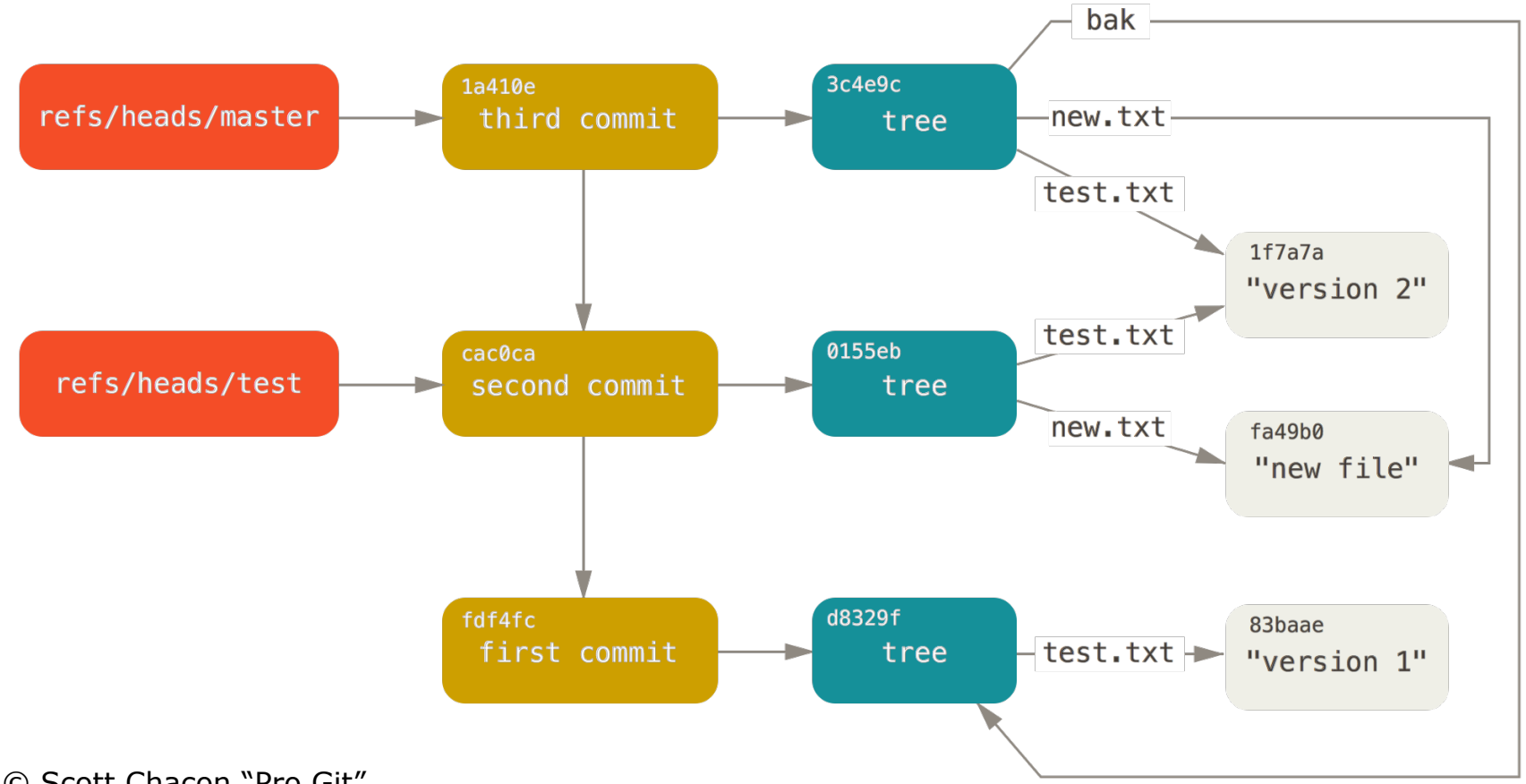

© Scott Chacon “Pro Git”

# Git Internals




© Scott Chacon "Pro Git"

# Git Internals




© Scott Chacon "Pro Git"

# Summary

- Version control has many advantages
  - History, traceability, versioning
  - Collaborative and parallel development
- Locking vs. merging and merge conflicts
- Collaboration with branches
- From local to central to distributed version control

# Lessons (reprise)

- Keep it simple
- Use all the tools you know:
  - A good IDE
  - Static analysis tools like FindBugs
  - Verification tools for critical code
  - Unit tests
  - Assert statements for known invariants
  - Code review for all code intended for other developers or users
  - Continuous integration testing for any project with multiple developers