Change Management (CM)*

Source code control

Versioning

Configuration Management

*Some of this material adapted from Ian Sommerville's course notes, Ch. 25, and Wigerd, L. and Seiwald, 1998

What did you do…

…when you collaborated on your first team paper or project at ASU?

Gmail? Dropbox? Google site?

Have you used “track changes”?

How did you manage experimentation?

Did you ever just want to “undo” something?

Why does software change?

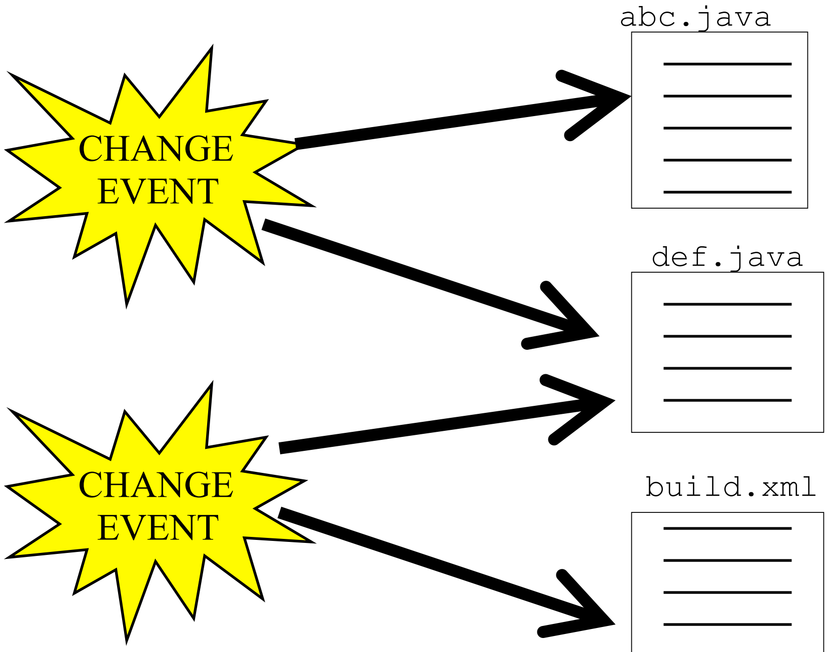
Name all the reasons why software changes

Now, with that list, consider:

1. Does your change happen before or after a software has been released (is in a customer's hands)?
2. What is the root nature of your change (a defect, a new feature, a change in requirements)?
3. Could the change have been avoided? If so, how? If not, why not?
4. What is the best way to make the change?
5. Who has to know about the change?

Change Management

What happens when change happens?



Change management

Keeping track of *requests for changes* to the software from customers and developers, working out the *costs and impact of changes*, and *deciding changes* to be implemented.

Configuration management (CM)

Keeping track of how software components and artifacts are assembled, including what versions, how they are configured, and associated metadata to inform a release

Version management (Source Code Control)

Keeping track of the *multiple versions of system components* and ensuring that changes made to components by different developers do not interfere with each other.

Change Management

Change happens!

- Every unit of work requires changing some system artifact

Many reasons for change:

- Business opportunity presents itself
- Incomplete and ambiguous requirements
- New technology
- *…and a zillion other reasons*

Change Management processes identify

- What system artifacts changed (which new artifact version)
- Why it needed to be changed (which task caused the artifact change)
- Who made the change and when it occurred (audit-ability)

Change Management processes

- Traditionally requires traceability and a management tool
- Must inform stakeholders (often there is a CCB)
- *Agile says to embrace it*; that empirical process control thing

IEEE828-2005

What are the 7 SCM activities?

1. *Configuration Identification* – identifying what needs to be controlled (the spec lists everything under the sun), and how it is named and versioned
2. *Configuration Control* – “request, evaluate, approve or disapprove, and implement changes to baselined CIs.”
3. *Configuration Status Accounting* – “status” – really how you track CIs
4. *Configuration evaluation and reviews* – “a management mechanism to evaluate a baseline”
5. *Interface control* – Coordinate changes w/ changes external to the project
6. *Subcontractor/vendor control* – how you plan and monitor change that happens with your subcontractors
7. *Release management and delivery* – how build, release, and delivery of software products and documentation will be formally controlled

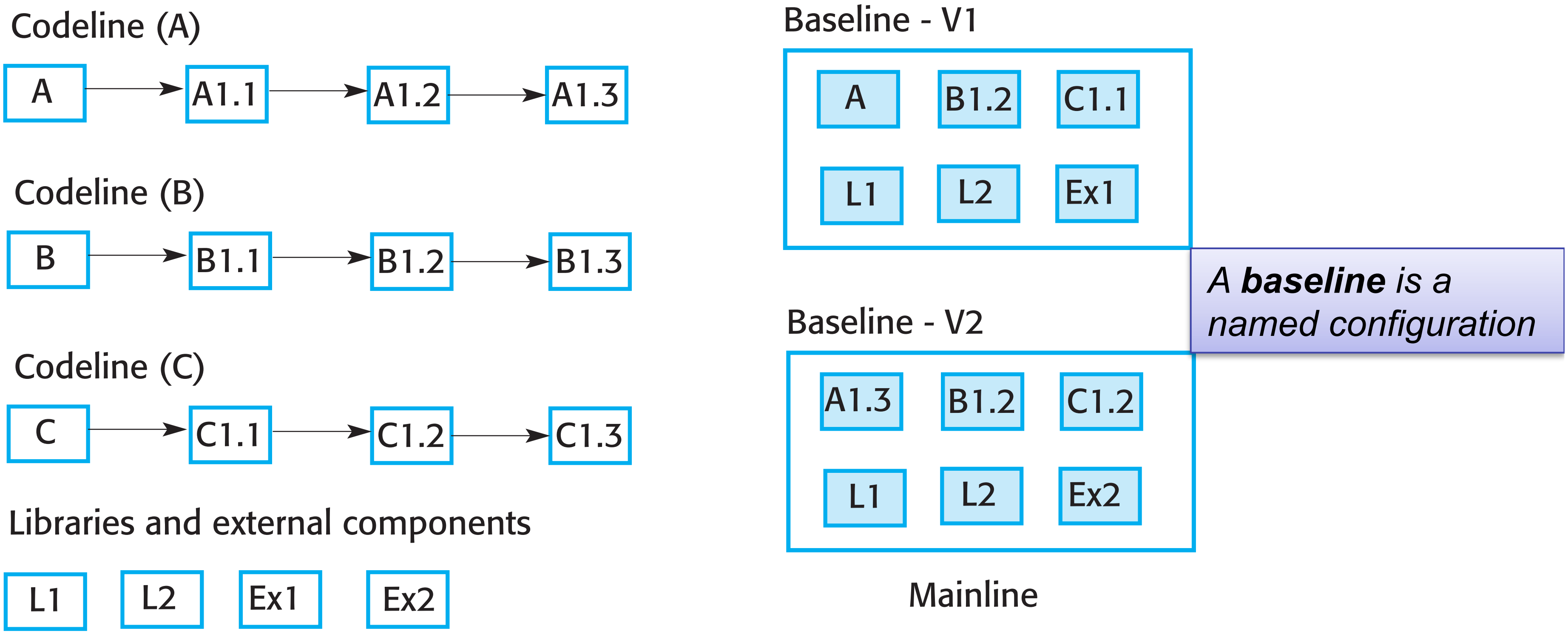
Read section 3.3 – count how many times you read the word (or hear it implied) “controlled” – not “embrace”

Configuration Mgmt Concepts:

Configuration Management is a management of software artifact (component) assembly and configuration

Codelines define a *trajectory* for [source code] artifacts

- You have a history
- You have a notion of where it is going
- You have a *set of policies governing participation*

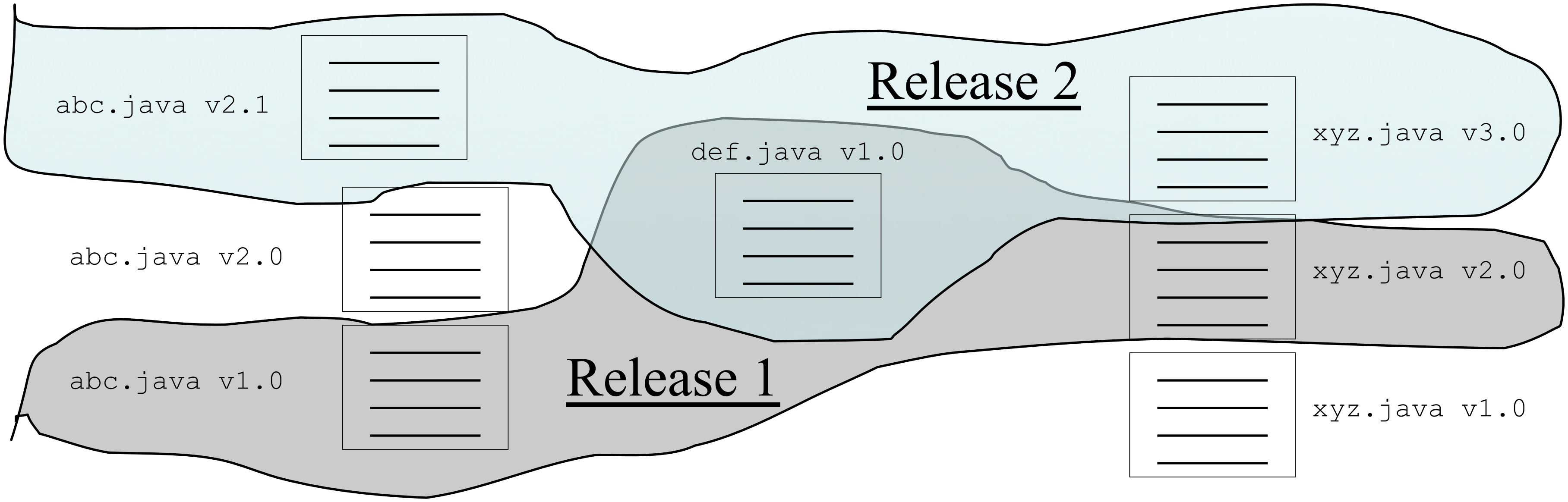


*A **baseline** is a named configuration*

CM Concepts

Configuration: An instance of a system composed of specific versions of its artifacts

- Includes expectations of the target environment(s) & config files!



Release: An instance of a system distributed to users outside of the development team

- Releases may be targeted for (in)external communities

CM Concepts

Version management (VM)

- *keeping track of versions of software components* or configuration items (CIs) and the systems in which these components are used.
  - involves ensuring that changes made by different developers to these versions do not interfere with each other.
- VM is what we usually think of as *source code control*.

A **source code control (SCC) repository**

- A shared file system (?) of software artifacts *(…well)*
- Typically supported with client/server tools *(…well)*
- Often provides some mechanism for assigning *jobs* to *change control* on software artifacts.

Content-Addressable Filesystems

- Support Distributed Version Control Systems (DVCS, *stay tuned…*)
- These store whole copies of repository objects locally (p2p)
- This is what Git, BitBucket, etc. basically are

CM Concepts

Workspaces are local (client) repositories

- Where developers build, test, and debug.
- Developers must periodically synch with SCC repository

Branch - A branch is a named variant of a codeline. That is

- A collection of software artifacts
- Assigned a logical identifier
- Whose purpose is to be either folded back into the main codeline, or maintained as a release.

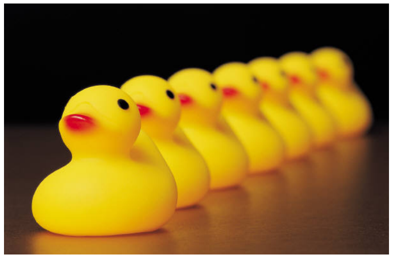
Label – tag identifying a specific version of an artifact

- Identifies a group of artifact variants – for a change, for a build, for a release, etc.

Optimistic vs. Pessimistic - Version Management "mood"

- If you are *optimistic* you allow changes to artifacts checked out by collaborators, and trust them to not clobber each other's commits.
- If you are *pessimistic* you lock resources to allow sole editing

CM Best Practices



Codelines:

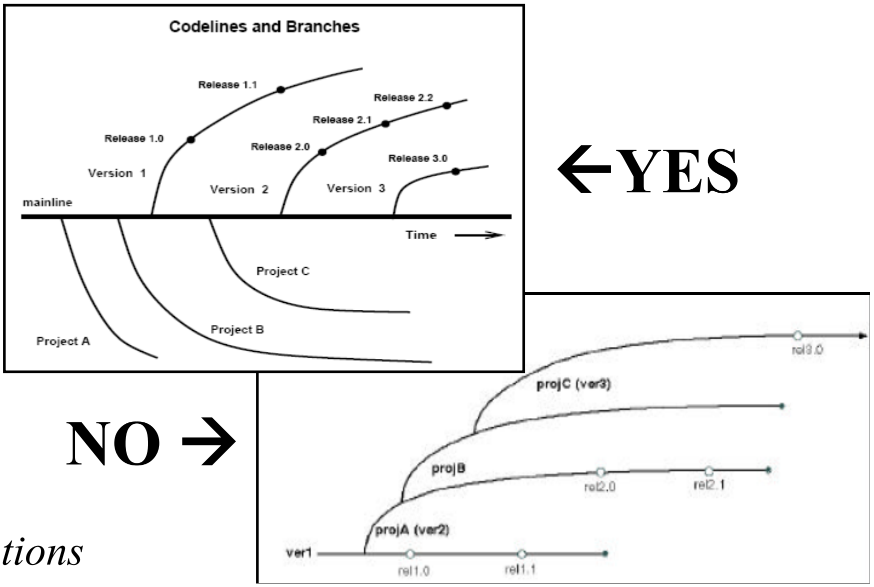
- Give each codeline a policy.
- Give each codeline an owner.
- Have a mainline.

Change propagation: from one codeline to another.

- Make original changes in the branch that has evolved the least.
- Get the right person to do the merge.
- Propagate early and often?
  - The thinking of this one has changed with DVCM (*stay tuned…)*

Branches**: <u>*variants*</u> of code

- Don't copy when you mean to branch.
- Branch only when necessary.
- Branch on incompatible policy.
- Branch late, merge early.
- Branch, instead of freeze.



***stay tuned, in DVCM (Git) we will change these assumptions*

Branching/Merging** (per artifact)

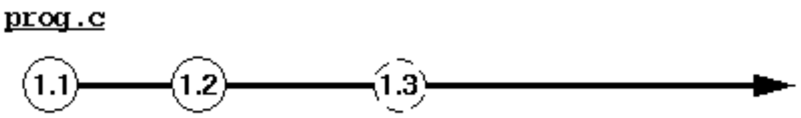


Figure 2: Serial development line for file `prog.c`

Exclusive locking – one codeline

Changes visible immediately after check-in

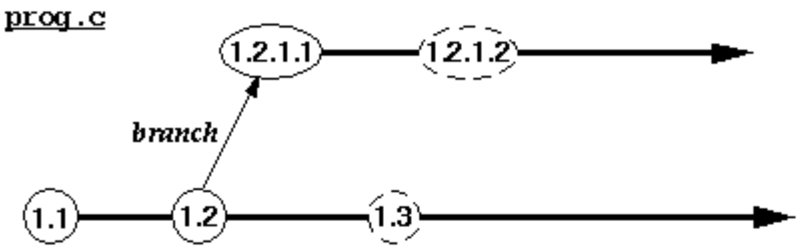


Figure 3: Branching off a new development line for file `prog.c`

Concurrent development using branching – multiple codelines

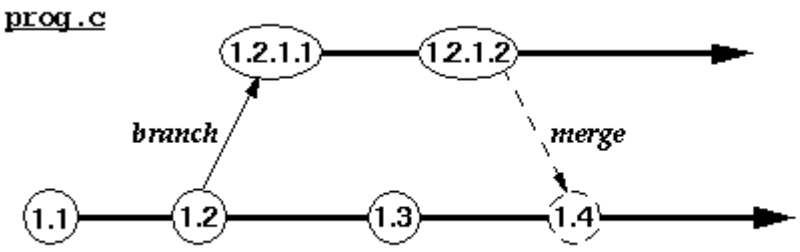


Figure 4: Merging back to the parent branch for file `prog.c`

From http://www.cmcrossroads.com/bradapp/acme/branching/

Synchronize concurrent development using merges – merge between codelines

Changes private and not part of mainline until merge – better supports large, long changes

***stay tuned, in DVCM (Git) we will change these assumptions*

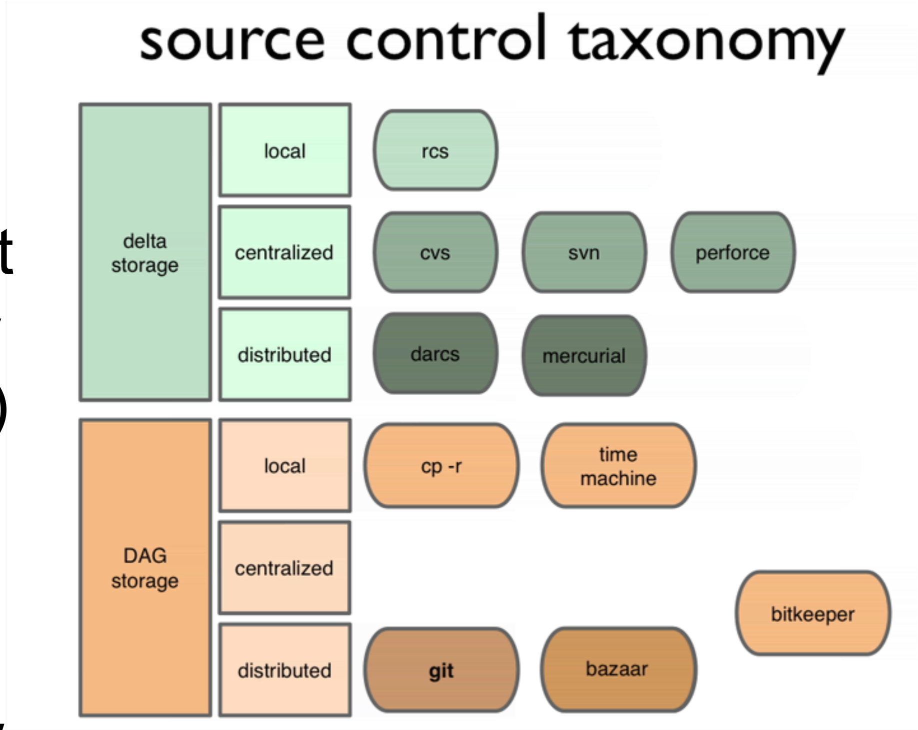
DVCs an CAFs

DVCS – Distributed Version Control Systems

CAFs – Content Addressable Filesystems

Why are they important?

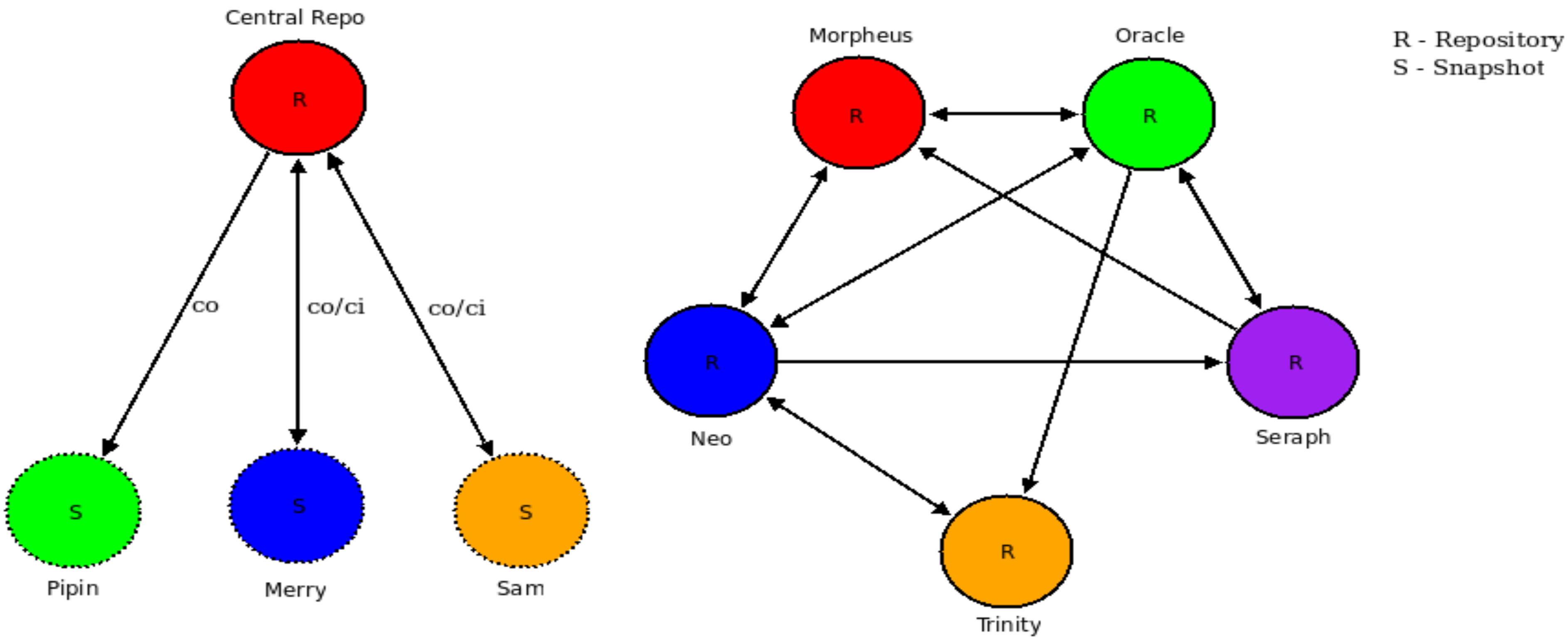
- DVCS support distributed repository storage
- CAFs support complete object storage where objects identify themselves (self-addressable)
- These enable a powerful new approach to SCC
  - “Repos” are peer-to-peer
  - Single “branch & merge” workflow evolves to support for flexible workflows



What is Git?

Distributed version control

Content-addressable filesystem (Torvalds)



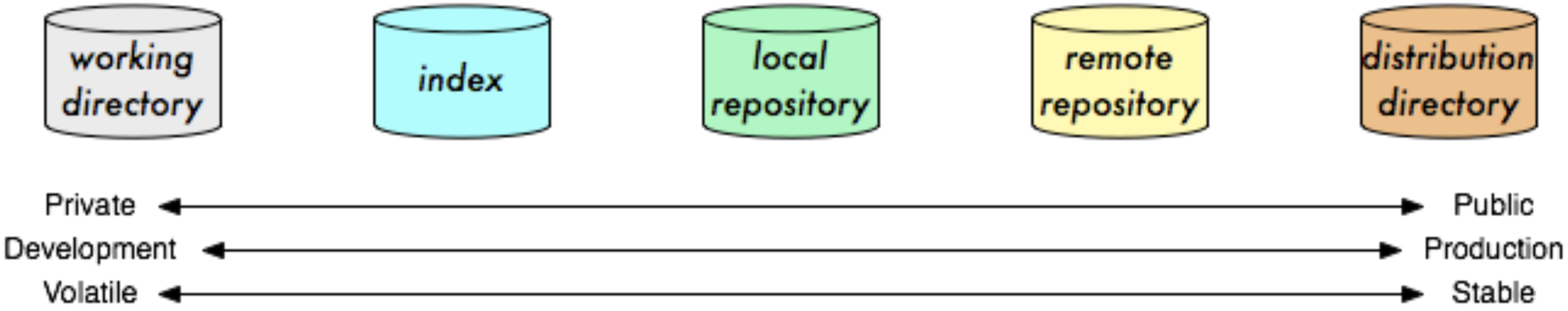
Traditional (Centralized)

Eg: Subversion (svn), CVS (cvs), etc.

Distributed

Eg: Git (git), Mercurial (hg), Monotone (mnt), etc.

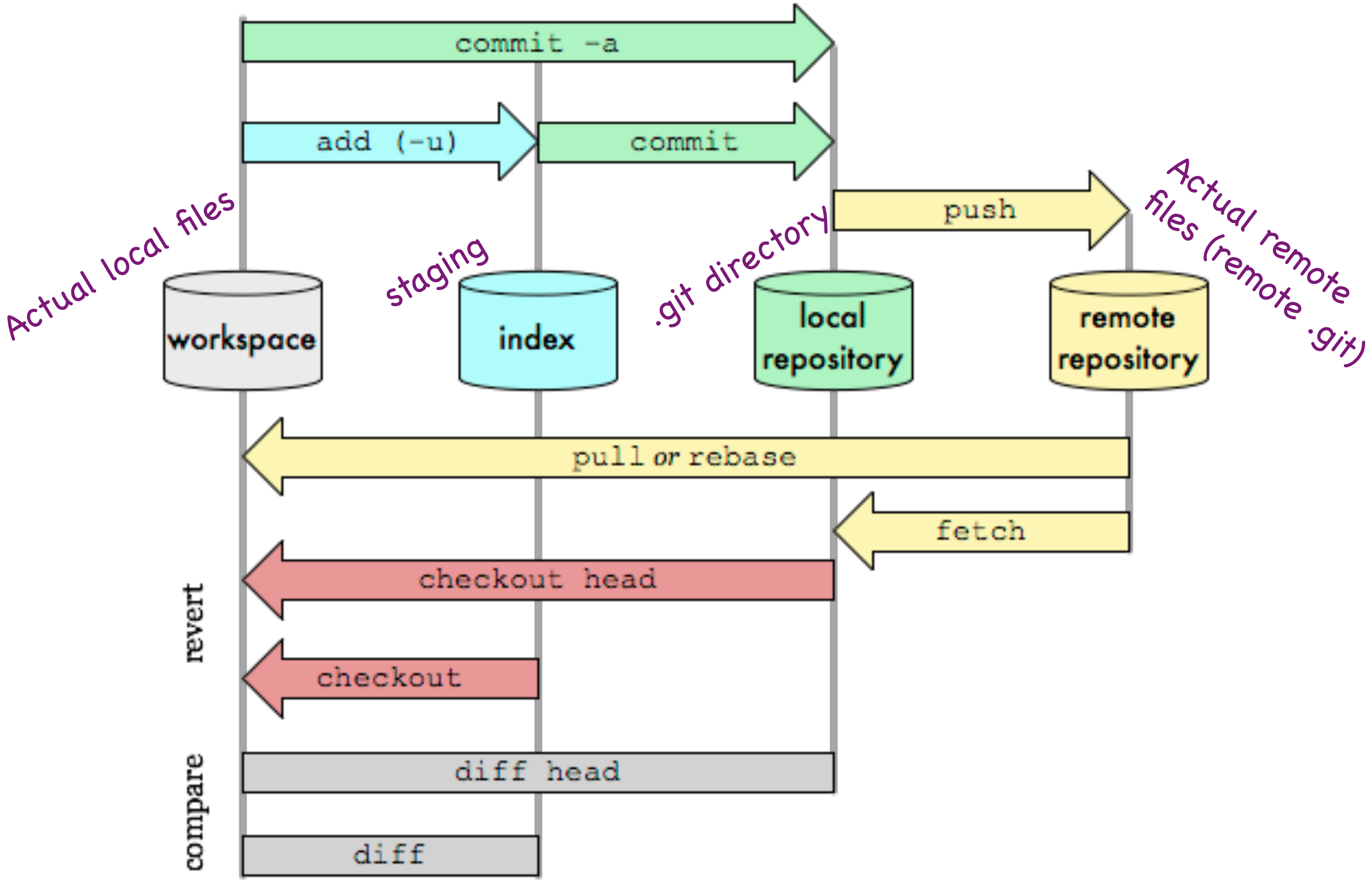
Working with Git



- Git is as effective locally as remotely, perhaps even moreso
- Do not confuse “Git” and “Github”
  - Github is a remote repository in the above diagram, with some value-added tools and services
- Keeping in mind that Git stores complete copies of objects, it is important to understand the *index* as a staging mechanism

Git Data Transport Commands

http://osteele.com

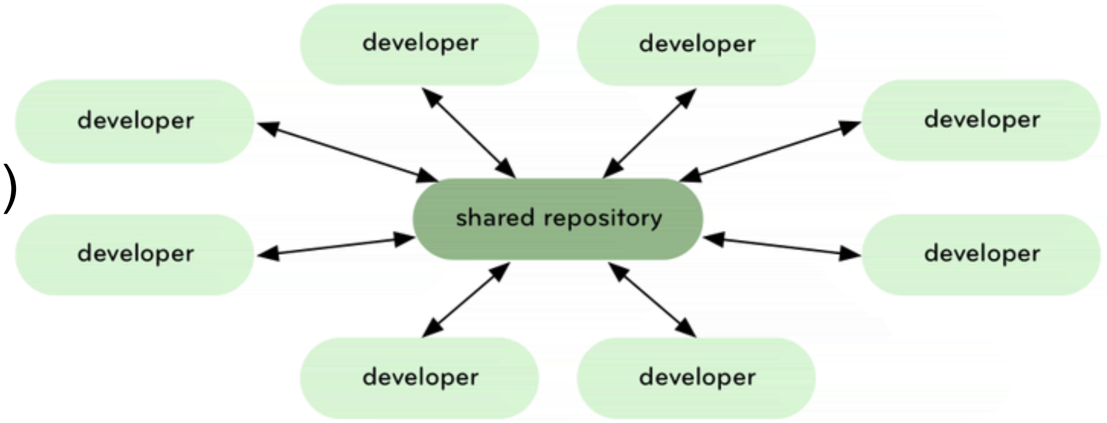


Git Workflows

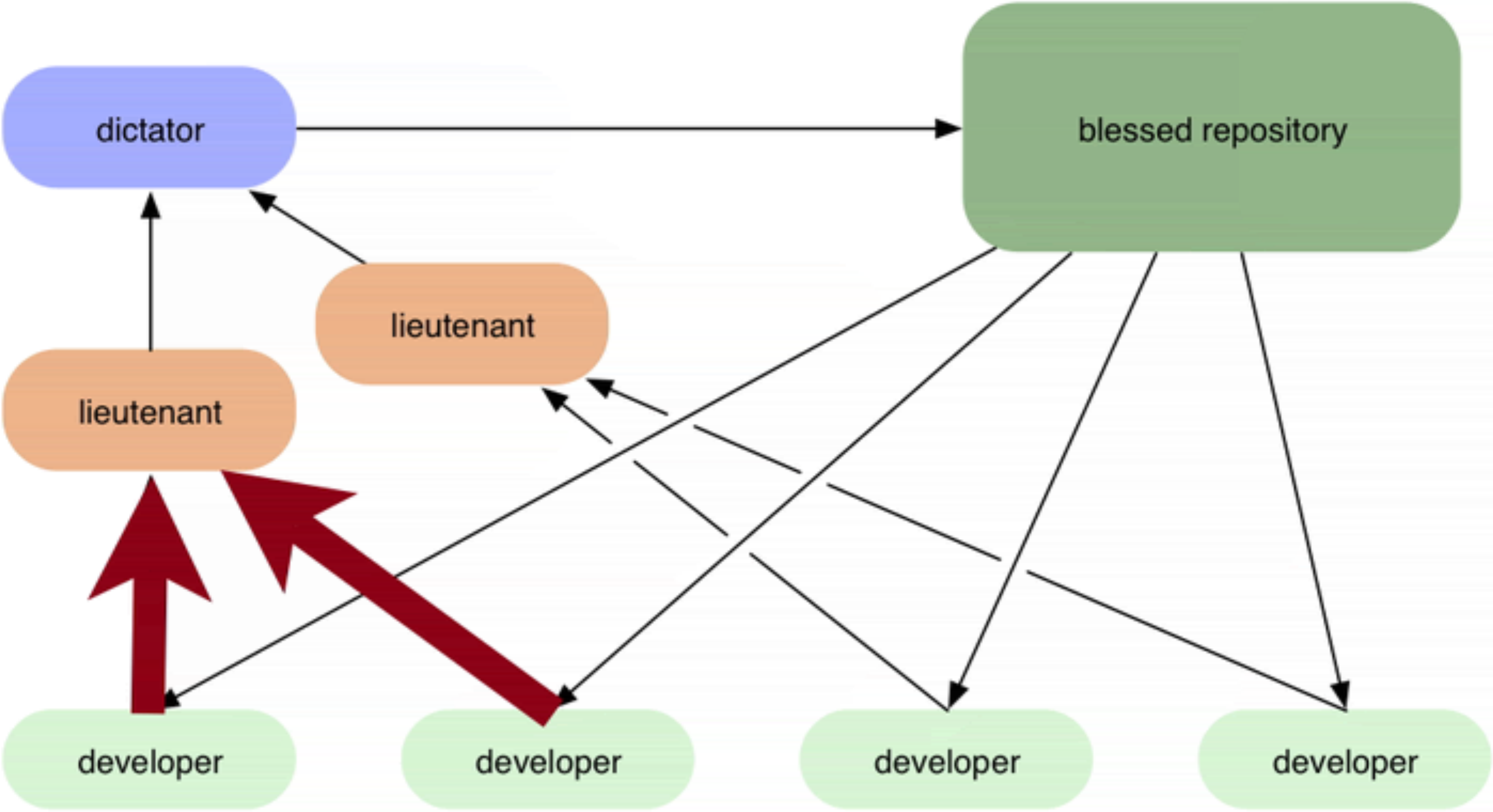
- As a content-addressable filesystem, Git doesn't prescribe the way in which you manage objects
- While others SCMs allow you to vary the workflow, the predominant is branch-&-merge
- *Nothing in git enforces a workflow, but agreeing on a workflow is a best practice, otherwise chaos ensues!*

Some Example Git Workflows

- Pretty easy to keep doing what you are used to doing
- Have to get used to re-interpreting some of the commands (checkout, commit)
- You will clone-push-fetch-merge (lather-rinse-repeat)



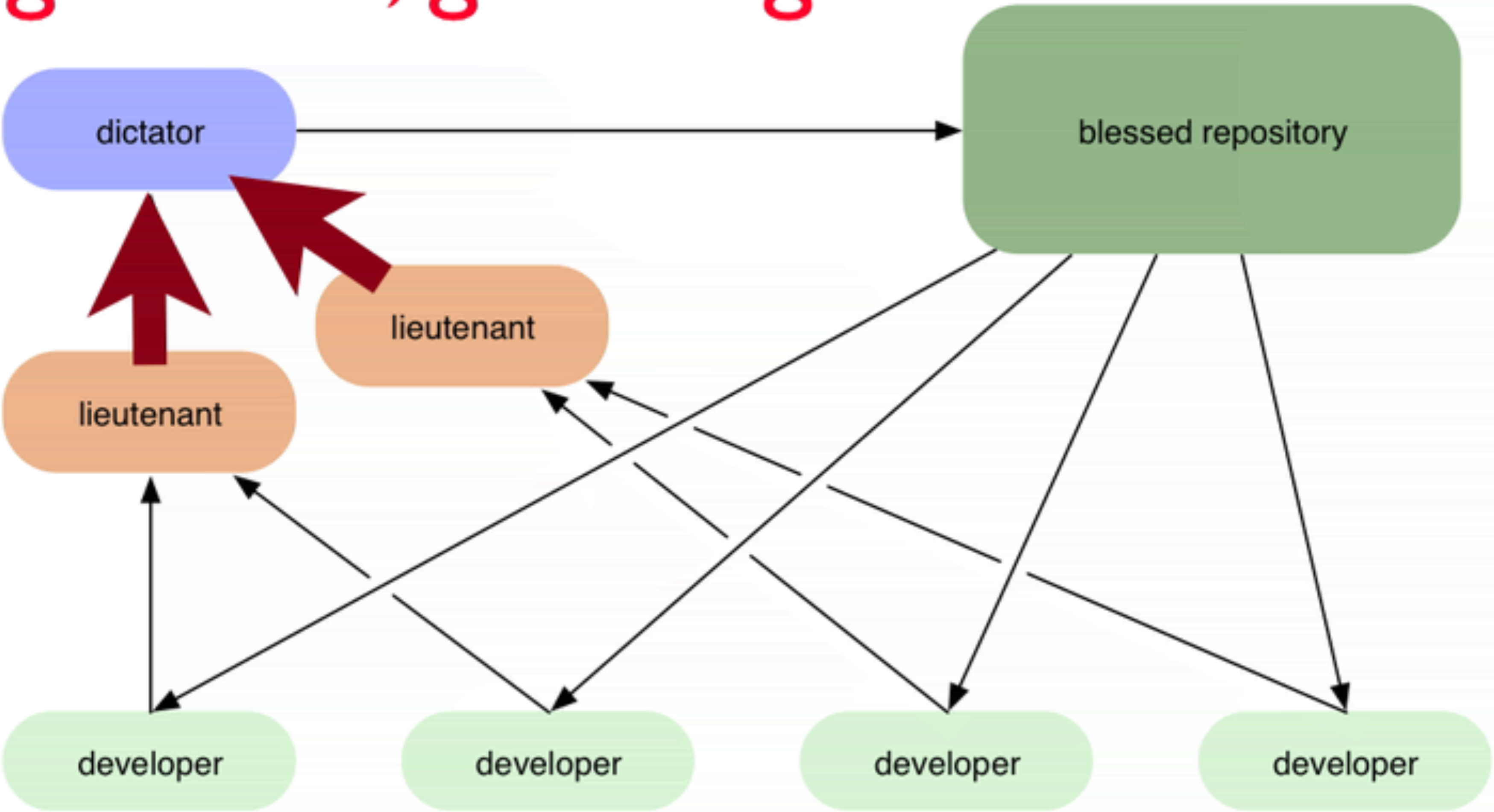
(Benevolent) Dictator Workflow



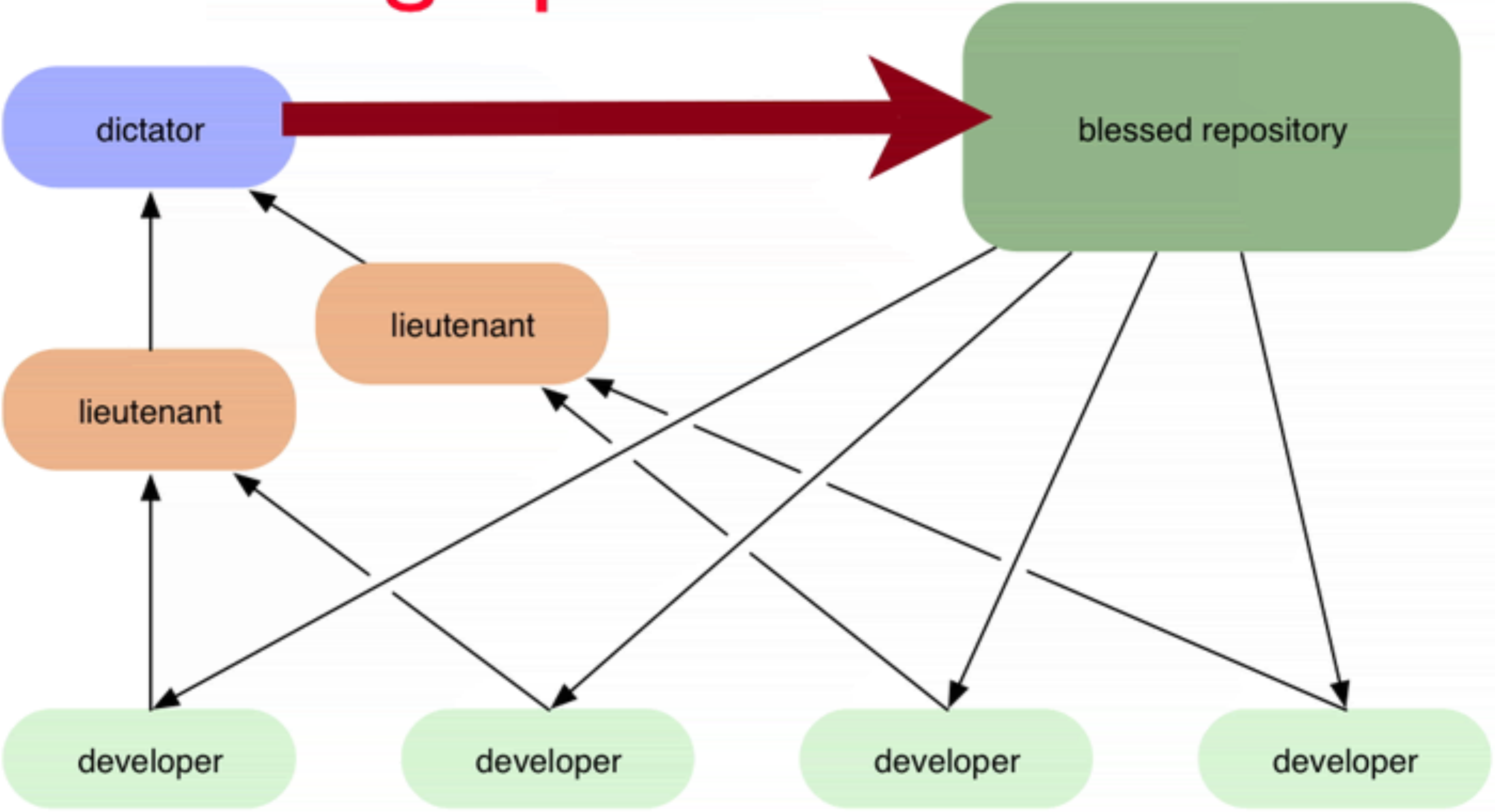
git fetch

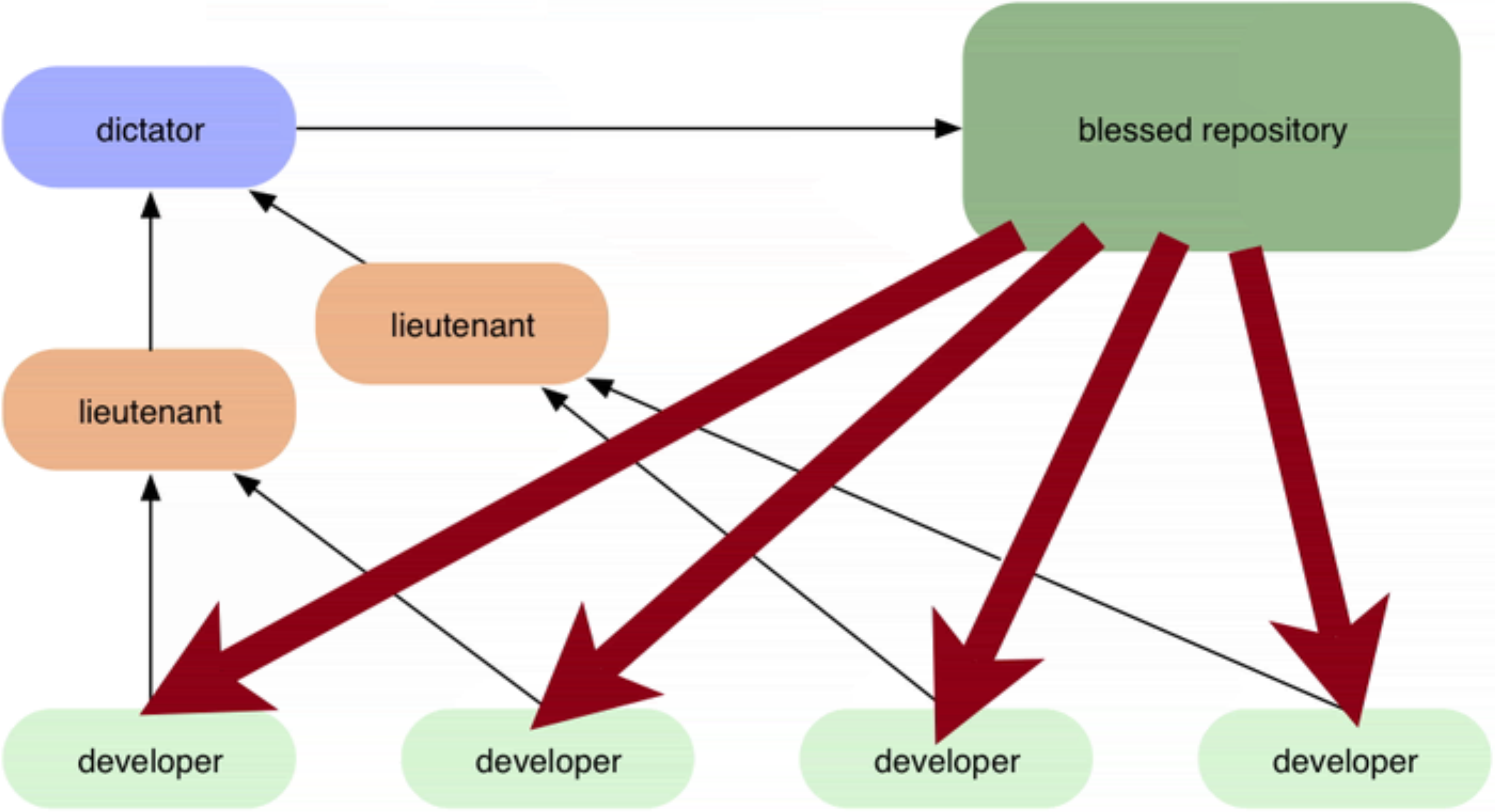
git merge

git fetch; git merge



git push

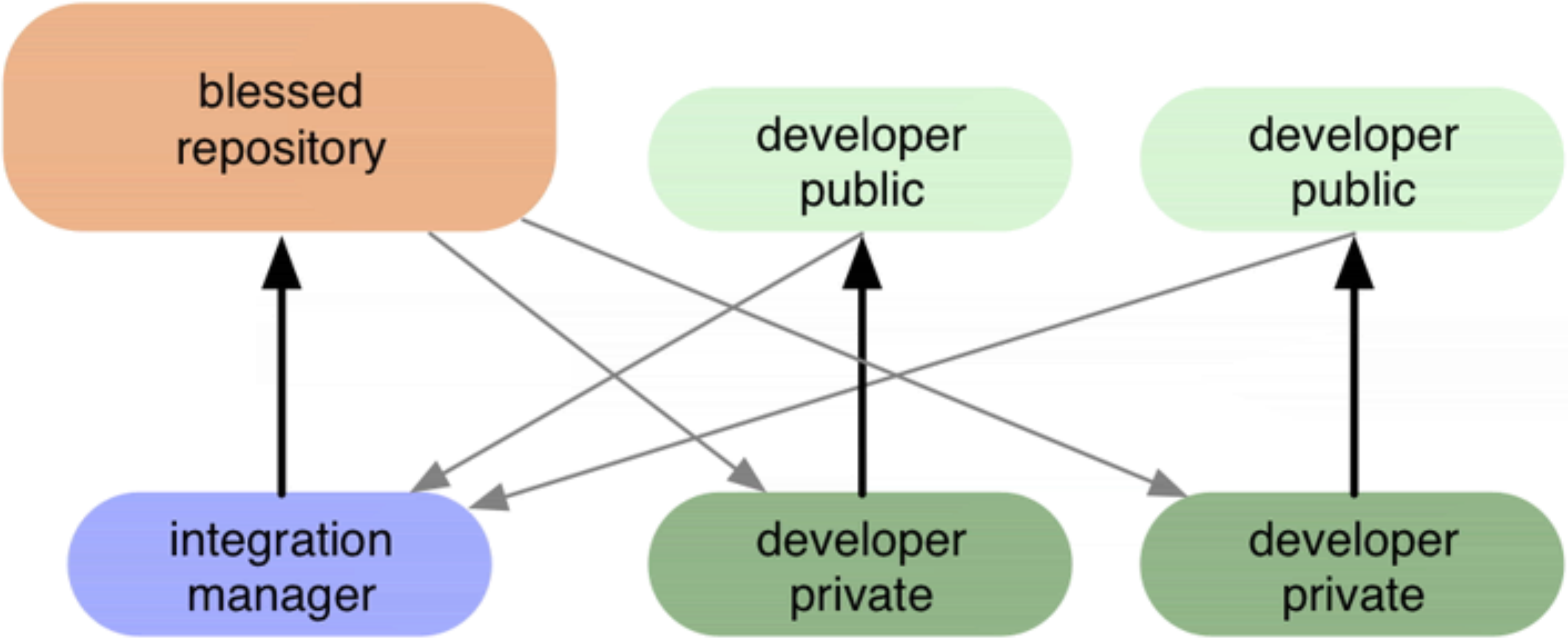

dictator
blessed repository
lieutenant
lieutenant
developer
developer
developer
developer


dictator
blessed repository
lieutenant
lieutenant
developer
developer
developer
developer

git fetch

Integration Manager Workflow

Useful for managing periodic integrations of components or public forks



Summary

Change Management defines how your project deals with … change (duh).

- *How you deal with, or embrace, change is one of the main things that distinguishes Agile from traditional SDLC*

Configuration Management is the assembly and (duh) configuration of artifacts into releasable software

Source Code Control is version management for code

- Git is not an incremental evolution in SCC, it is a CAF
  - To use Git effectively you have to understand what it *is*
  - People like Git because it takes the pain out of merging
    - "Unlearning" centralized SCM is probably the main reason why people do not like it
    - Some folks like its advanced capabilities, like defining your workflow

Recommended Reading

- Sommerville, Ian. *Software Engineering, 9th ed.*, Chapter 25. Addison-Wesley, 20011.
- Wingerd, L. and Seiwald, C. “High-level Best Practices in Software Configuration Management”, ECOOP 98, SCM-8, Springer-Verlag LNCS 1439, 1998.
- Chacon, Scott. Pro Git. (free online at http://git-scm.com/book), 2009.
- IEEE Standard 828-2005, IEEE Standard for Software Configuration Management Plans
- Software Engineering Body of Knowledge (SWEBOK), Chapter 7, Software Configuration Management, 2004