Módulo 2:
Sistemas de gestión de versiones

Diseño y Desarrollo de Software (*1er. Cuat. 2019*)

Profesora titular de la cátedra:
Marcela Capobianco

Profesores interinos:
Sebastián Gottifredi
Gerardo I. Simari

Licenciatura en Ciencias de la Computación – UNS

Licencia

Linux Kernel

- En 1991 Linus Torvalds comenzó con el desarrollo del kernel de linux con la ayuda de la comunidad.
- 1992: Linux version 0.95 es capaz de correr el sistema X Window.
- Hasta comienzos de 1994, aparecieron 15 versiones 0.99.X.
- 1994: Se lanza Linux 1.0.0 (176,250 loc).
- 1996: Se lanza Linux 2.0.0 (777,956 loc).

Linux Kernel

- En el año 2002, 418 desarrolladores figuraban en el *credit file*, y el desarrollo se pasó a un sistema de gestión de código llamado BitKeeper.
- En 2015, Linux había alcanzado la versión 4.1, 19,5 millones de loc, y casi 14.000 programadores.
- Esta historia de éxito no es un caso aislado.
- *¿Cuál es la diferencia con los casos de proyectos fallidos que vimos*?

Motivaciones

- Como vimos anteriormente, la *ley de Brooks* establece que al agregar colaboradores a un proyecto atrasado se lo suele retrasar aún más.
- Esto no siempre es así, sobre todo en la comunidad del Software libre; pero, ¿*por qué*?

Motivaciones

- Como vimos anteriormente, la *ley de Brooks* establece que al agregar colaboradores a un proyecto atrasado se lo suele retrasar aún más.
- Esto no siempre es así, sobre todo en la comunidad del Software libre; pero, ¿*por qué*?
- Bajo algunas condiciones, podemos ordenar la interacción entre colaboradores mediante los *sistemas de gestión de versiones* (SGV).

Trabajo en grupo

- Pueden existir distintas relaciones entre los colaboradores (jerárquicas, pares, etc.)
- Hay que *integrar* los trabajos individuales.
- Coordinar para que todos puedan trabajar sobre la *misma base de código*.
- Es importante tener la capacidad de *revisar* los cambios (conjuntos de cambios).
- En general coexisten diferentes versiones simultáneamente.

Control de versiones

- En todo proyecto de desarrollo resulta conveniente tener archivada su *historia*.
- Las modificaciones pueden producir errores ocultos.
- Si colaboran varias personas, es necesario *registrar* los cambios de *cada autor*.
- A veces se mantiene una versión *estable* y una *experimental*.

Control de versiones

- El CV nace en otras disciplinas de ingeniería para llevar control de las versiones de los planos.
- Esto permitía volver atrás cuando se llegaba a ciertos puntos indeseados en el diseño.
- En forma similar, en la Ingeniería de Software se monitorean los cambios al código, documentación y archivos de configuración.

Usos

Los sistemas de CV tienen otros usos posibles:

- Versiones de los trabajos prácticos o transparencias de una materia.
- Trabajo en grupo en un informe o artículo científico.
- Archivos de configuración de un servidor.
- Versiones de los archivos CAD de un proyecto de ingeniería civil.

Gestión de fuentes

- Un sistema de gestión de fuentes registra la historia como un *conjunto de diferencias* sobre un patrón (el más reciente).
- Cada diferencia se etiqueta con los meta-datos necesarios.
- Una versión primitiva es usar *diff* y etiquetar con los conjuntos de cambios.
- Idealmente, debe permitir la colaboración y el trabajo concurrente.

Gestión de versiones

- El control puede ser *pesimista* u *optimista*.
- El control pesimista provoca un *lock* sobre los archivos (similar al concepto usado en DBMSs).
- Esto provoca retrasos.
- Un sistema optimista deja avanzar suponiendo que no va a pasar nada malo, y nos informa si se producen conflictos.

Historia

- *Source Code Control System* (SCCS) fue desarrollado por Bell Labs en 1980.
- SCCS es actualmente obsoleto, y fue superado por proyectos de software libre.
- *Revision Control System* (RCS) nació en Purdue University unos años después de SCCS .

Historia

- RCS posee una interfaz de comandos más clara.
- Fue y todavia es muy utilizado como sistema de control de versiones.
- Funciona bien para pequeños grupos de desarrolladores.
- Las fuentes de RCS son mantenidas y distribuidas por la Free Software Foundation (GNU RCS*).

(*) `https://www.gnu.org/software/rcs/`

Conceptos básicos

Veamos un poco de terminología básica:

- *Repositorio*: un conjunto compuesto por el código fuente en un punto determinado del tiempo, y la historia asociada a éste.
- La historia de un código fuente es un conjunto de *changesets*.

Conceptos básicos

- *Changeset*: conjunto ordenado de cambios.
- Un *changeset* se “aplica” a un repositorio cuando se lo introduce ordenadamente.
- A cada *changeset* le podemos asociar información adicional: nombre del autor, la fecha, etc.

Conceptos básicos

- La operación más básica es el *branch* o *check-out*, (copiar para leer o realizar cambios).
- Un *branch* produce una *working copy.*
- Hay mucho tipos de *branches*, y el concepto se ajusta según cómo lo maneje cada herramienta en particular.
- Los *branches* se realizan para poder trabajar en forma concurrente.

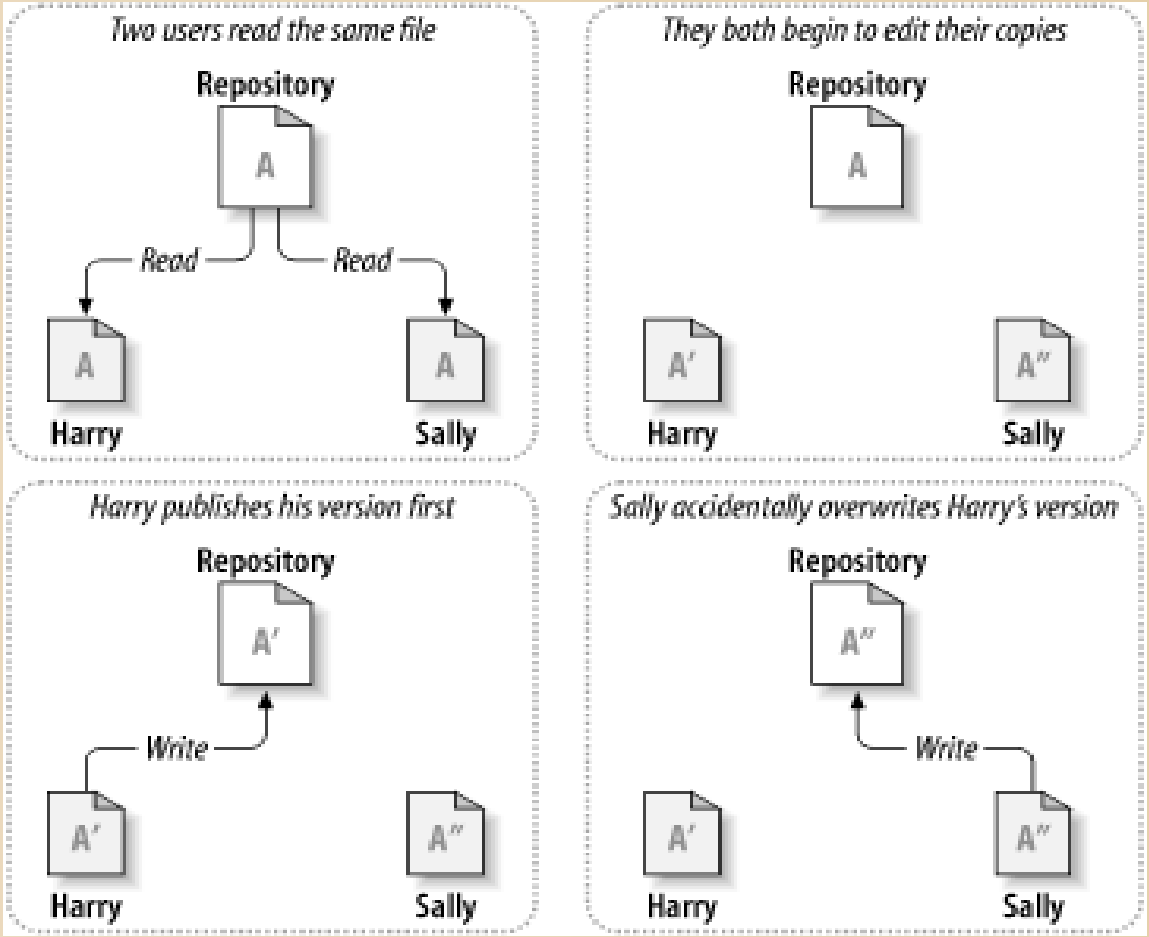
Conceptos básicos

- Luego de finalizado el trabajo, se deben combinar los cambios con un *merge*.
- Posteriormente se actualiza el repositorio con un *check-in* o *commit*.
- Cada revisión provoca un versión diferente.
- Para publicar una versión hacemos un *export*, lo que es similar al *check-out* pero produce una versión sin metadata.

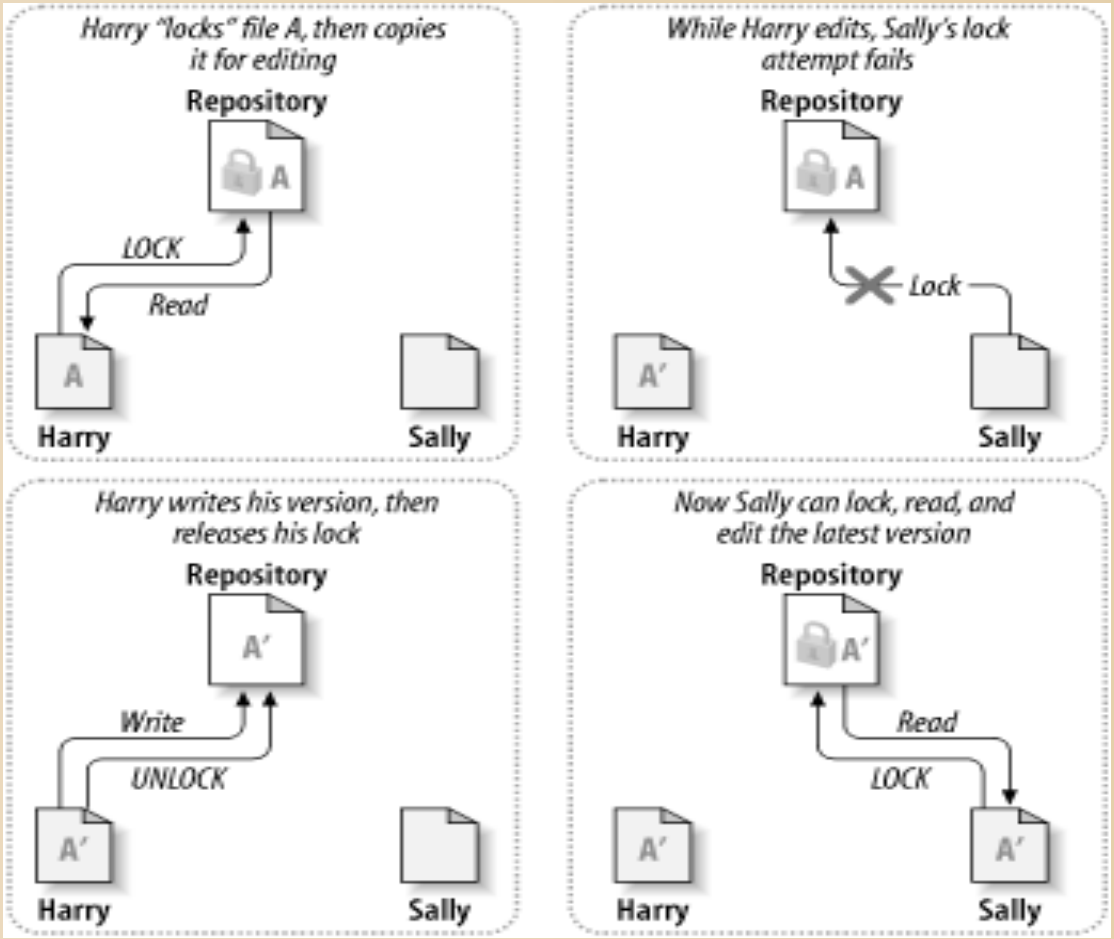
Conceptos básicos

- Una versión aprobada de un documento se denomina *baseline*.
- Cuando dos *changesets* modifican las mismas partes de un archivo, hay un *conflicto*.
- Si un SGV no detecta un conflicto, puede introducir corrupción en el código, lo cual puede ser catastrófico.

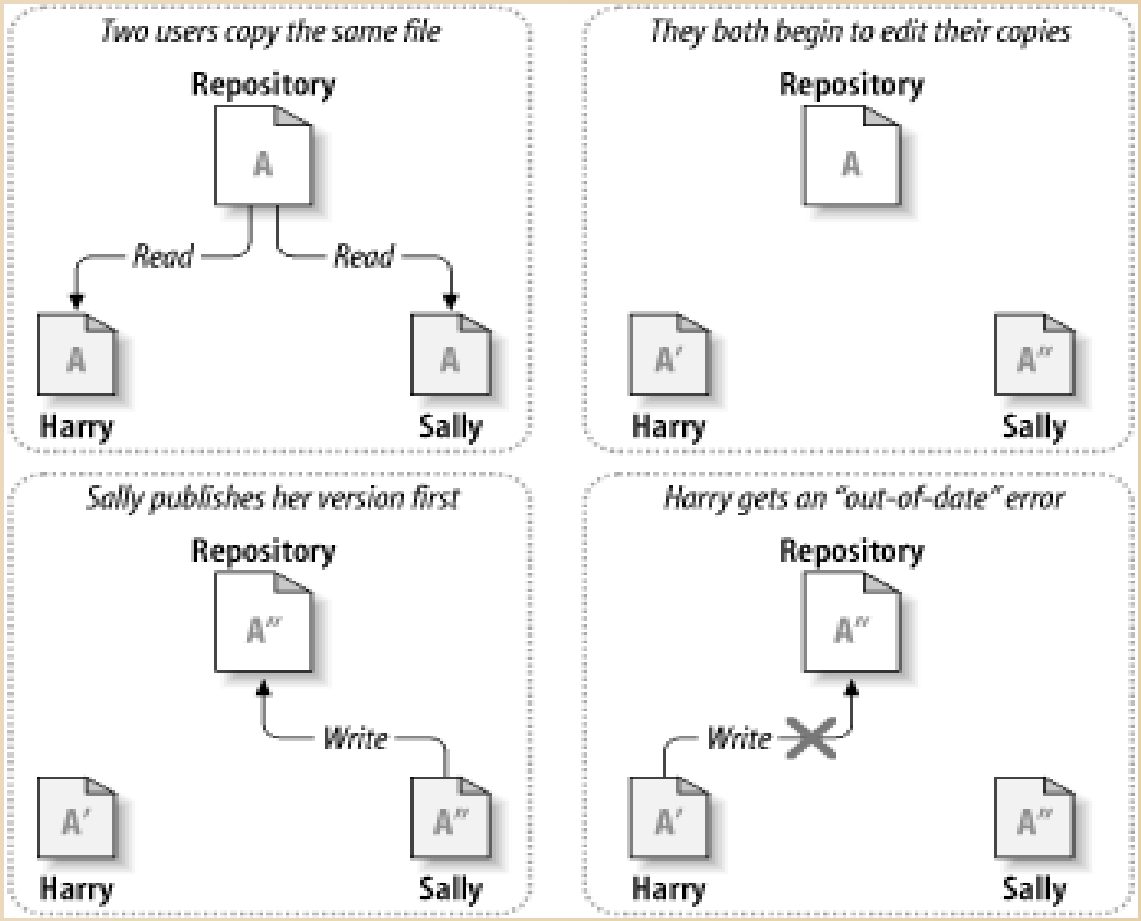
Sin SGV: *Problema*



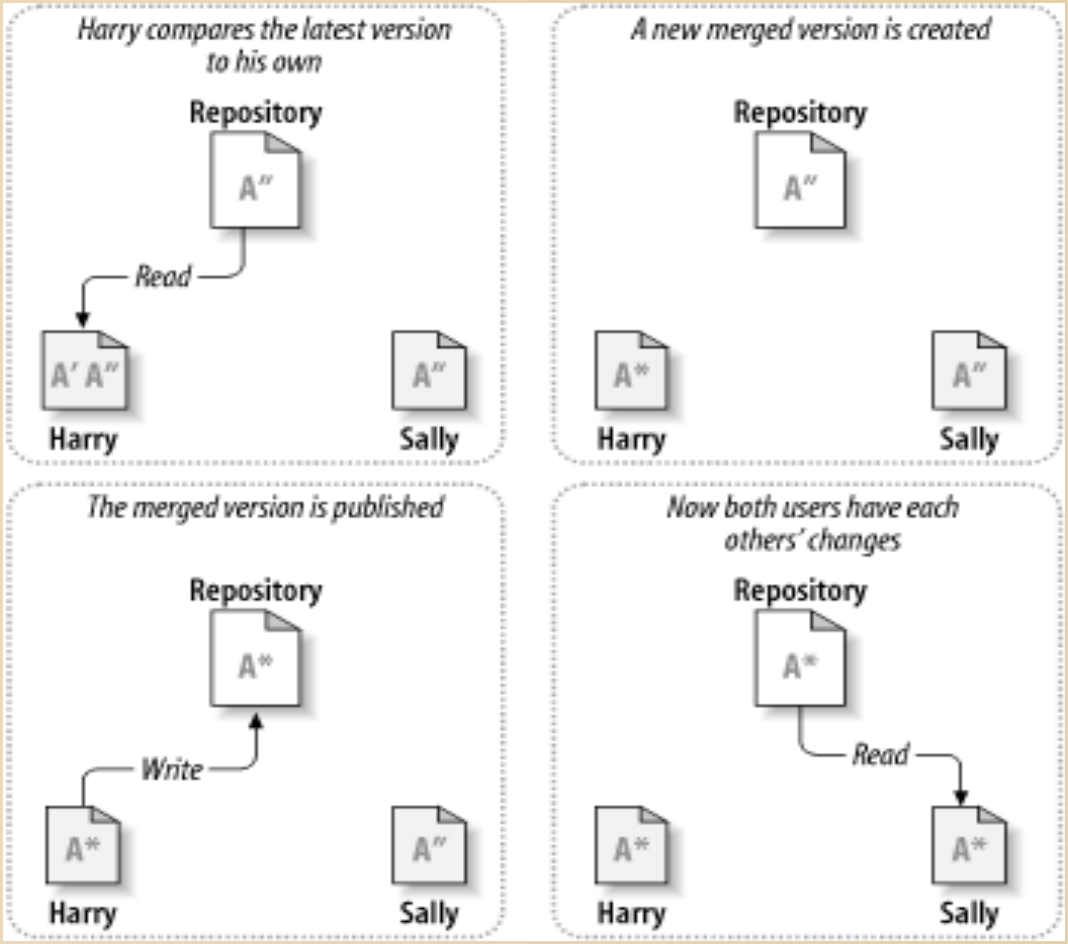
Protocolo *Lock-Modify-Unlock*



Copy-Modify-Merge



Copy-Modify-Merge (cont.)



Conflictos

- Los SGV modernos poseen muy buenos algoritmos de detección y resolución de conflictos.
- Al detectarse un conflicto que el SGV no puede resolver de forma automática, se *informa* al usuario.
- Para ejemplificar, veremos uno de los SGVs más usados: CVS

CVS

- *Concurrent Version System* (CVS) es un sistema de gestión de fuentes *optimista*, de software libre.
- Fue creado a fines de los años 80 y hoy en día es la base de los sistemas más usados en los proyectos de software libre.
- Trabaja con un repositorio central que se accede con un esquema cliente/servidor.
- La última versión data de 2008.

CVS

- El administrador decide quién tiene acceso a qué partes del repositorio.
- Puede permitirse un acceso anónimo (de sólo lectura) a cualquier persona.
- Esto es un concepto fundamental para entregar las versiones de manera rápida y frecuente.
- Los clientes pueden trabajar *al mismo tiempo*.

CVS

- Los administradores pueden ver quién está editando cada cosa.
- Se sabe quién hizo un cambio en particular (útil para el rastreo de errores).
- Para comenzar a trabajar se debe hacer un *check out* del proyecto o de una rama (*branch*).

CVS

- Cuando un cliente realiza un *check-in* de los cambios, el sistema intenta hacer un *merge*.
- Si falla, el usuario debe encargarse de resolver el conflicto.
- Si tiene éxito, se incrementa el número de versión.
- El *changeset* se etiqueta con una descripción provista por el usuario, los datos y nombre del autor.

CVS

- Los clientes también pueden mantener distintos *branches* del sistema.
- Una versión ya lanzada (*released)* puede formar un *branch* (esto se usa para arreglar bugs), mientras que una versión bajo desarrollo forma parte de otro.
- CVS usa una técnica llamada *delta encoding* (o *delta compression*) para almacenar en forma eficiente distintas versiones del mismo archivo.

Terminología

- *Module*: conjunto particular de archivos mantenido en CVS.
- *Repository*: lugar del servidor donde están los módulos.
- *Revision*: versión de un archivo.
- *Branch*: bifurcación de un módulo.

Ejemplo de uso CVS local

```
$ export CVSROOT=/cvs
$ cd /cvs
$ cvs init
```

Esto crea un modulo CVSROOT con los archivos de configuración.

Supongamos que queremos cambiar la configuración...

Ejemplo de uso CVS local

```
$ cvs checkout CVSROOT
cvs checkout: Updating CVSROOT
U CVSROOT/checkoutlist
U CVSROOT/commitinfo
U CVSROOT/config
...
$ cd CVSROOT
```

Ahora podemos editar los archivos:

```
cvs edit config
cvs commit config
cvs unedit config
```

Ejemplo de uso CVS en red

Primero configuramos el server CVS que corre desde inet.d, luego:

```
$ cvs -d :pserver:mc@frodo:/cvs login
(Logging in to mc@frodo)
CVS password:****
```

Esto se puede configurar en CVSROOT con el formato:

:pserver:<username>@<host>:/cvsdirectory

Ejemplo en red (anónimo)

```
cvs -d:pserver:anonymous@progs.org:/cvs login
cvs -d:pserver:anonymous@progs.org:/cvs checkout mod
cd mod
…
./configure
make
make install
cvs diff | mail -s “Parches” mod-maint@progs.org
```

Desarrollador estándar

- Tiene una cuenta en el servidor
- Se usa de la misma manera, cambiando "*anonymous*" por el nombre de usuario
- Por seguridad se debe usar **ssh** (que nos da un canal cifrado para una comunicación *segura*).
- A diferencia del usuario anónimo, puede hacer *commit* de los cambios en el repositorio.

Desarrollador estándar

- Ejemplo de uso:
  - modifica: cvs edit parte.c y cvs edit parte.h
  - los envía: cvs commit parte.h parte.c
  - CVS pide una explicación de qué se hizo, y esta explicación se agrega al historial
  - se incrementa el número de revisión del archivo en una unidad
  - Para notificar que ya no se está editando:

    cvs unedit parte.c y cvs unedit parte.h

Operación de *commit*

- ¿Cuándo se debe realizar una operación de *commit*?
- Durante el desarrollo de la modificación pueden alterarse otros archivos (usar el comando *update*)
- Esto es importante para realizar las pruebas correspondientes (unidad, regresión, etc.).
- En este paso pueden surgir *conflictos*.

El rol del administrador

- Al crear dos ramas, cuando la rama inestable se estabiliza hay que *mezclar ambas versiones*.
- El administrador puede realizar tareas para coordinar el equipo:
  - hacer que se envíen automáticamente correos electrónicos con notificaciones,
  - forzar la realización de pruebas,
  - restringir el acceso,
  - etc.

Inconvenientes de CVS

- No soporta renombramientos o cambios de directorios.
- Es bastante complicado el uso de ramas y mezclas (*branching/merging*).
- Para funcionar adecuadamente, necesita en todo momento *conexión* con el servidor.
- No posee *atomic commits* para grupos de archivos.

Inconvenientes de CVS

¿Qué puede pasar si no nos garantizan atomicidad?

- Si el usuario **A** necesita hacer commit de 10 archivos, alguien empieza a hacer commit al mismo tiempo y hace commit hasta el archivo 8 ...
- Los cambios a los primeros 7 archivos se aceptan y el resto no: el repositorio queda en un estado *inconsistente.*
- ¡*El commit no se puede revertir*!

Inconvenientes de CVS

- No permite usar otras herramientas para mezclar archivos que han dado lugar a conflictos.
- No posee una interfaz gráfica (aunque existen visores via Web).
- No genera (sin otras herramientas) un *changelog*.

Otros sistemas

- Existen dos paradigmas para el funcionamiento de los SGV: *centralizados* y *distribuidos.*
- El primer caso tiene una arquitectura centralizada tipo cliente-servidor.
- En el segundo no existe un punto central, por lo que puede compararse con sistemas peer-to-peer (P2P).

SGVs Centralizados

- Se basan en un repositorio único global en el que se guardan todos los *changesets*.
- Todos los desarrolladores se conectan a este repositorio para trabajar.
- Todos los *branches* están en el mismo servidor.
- Hay que trabajar online (al menos en el modelo básico).

SGVs Centralizados

- Hay dos tipos de SGV centralizados:
  - Los que usan el modelo *Lock-Modify-Unlock*
  - Los que usan el modelo *Copy-Modify-Merge*
- El primero es el más simple y limitado, no es una opción muy útil en la actualidad.
- Como vimos antes, en el segundo se hace una copia del estado actual del repositorio, se modifica y se aplica el *changeset* al repositorio con un *merge*.

SGVs Distribuidos

- No existe un punto central de desarrollo, los repositorios están distribuidos en diversas máquinas.
- Cada desarrollador tiene su repositorio propio sobre el cual trabaja de forma independiente.
- Periódicamente se consolidan los trabajos de todos en algún repositorio convenido.
- No existen distintos niveles de acceso, todos tienen control completo.

SGVs Distribuidos

- La puesta en común es una generalización del merge
- Se realiza de forma frecuente y es imprescindible para el funcionamiento del sistema (punto clave del SGV).
- Ejemplos de SGV distribuidos: Darcs, GIT y Bitkeeper.
- *El día 20/3 habrá una clase especial de GIT dada por un profesional invitado.*

Ejemplos de sistemas concretos

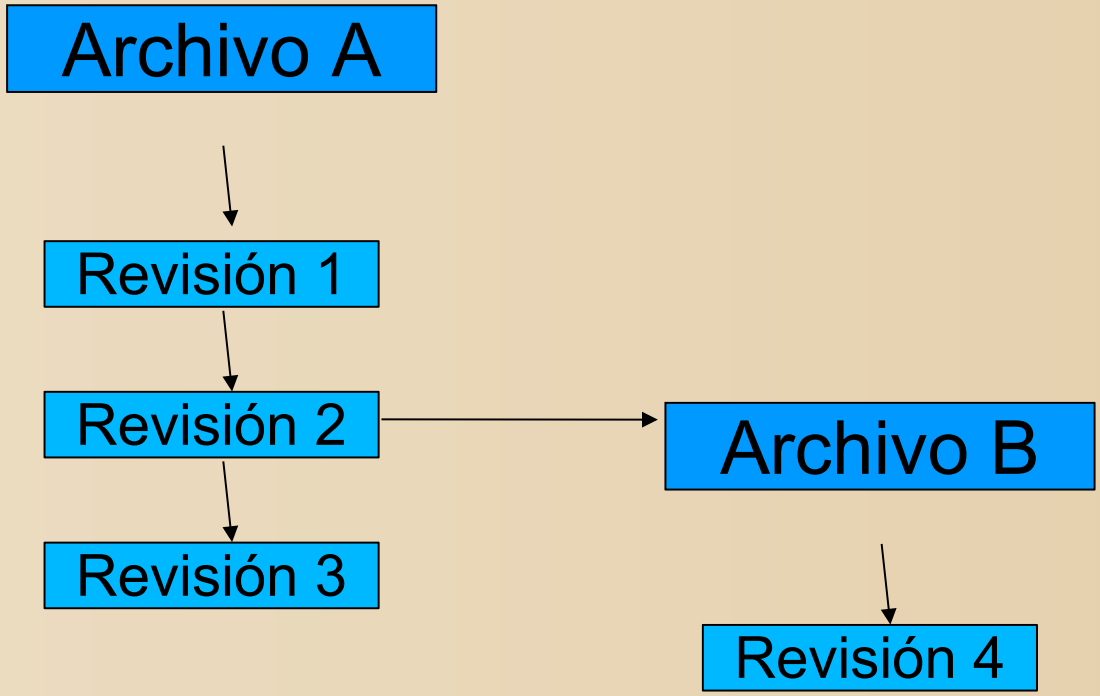
Subversion (SVN)

- Fue pensado como un reemplazo a CVS.
- Soluciona gran parte de sus problemas.
- Usa el modelo *Copy-Modify-Merge* y trata de hacer todas las operaciones posibles sin necesidad de conexión.
- Permite enviar al servidor sólo los cambios, pudiendo hacer offline la operación *diff*.
- Esto ahorra ancho de banda al no realizar transferencias innecesarias.

Subversion (SVN)

- El modelo es de un *filesystem versionado*, por lo que permite las operaciones comunes, con algunas particularidades.
- Al copiar un archivo *A* a un archivo *B*, en realidad *no se copia* el archivo sino que la copia *B* es una bifurcación en la línea de tiempo del archivo A.
- Esto soluciona el problema de CVS en donde se perdía la historia.

Filesystem con versiones



Filesystem con versiones

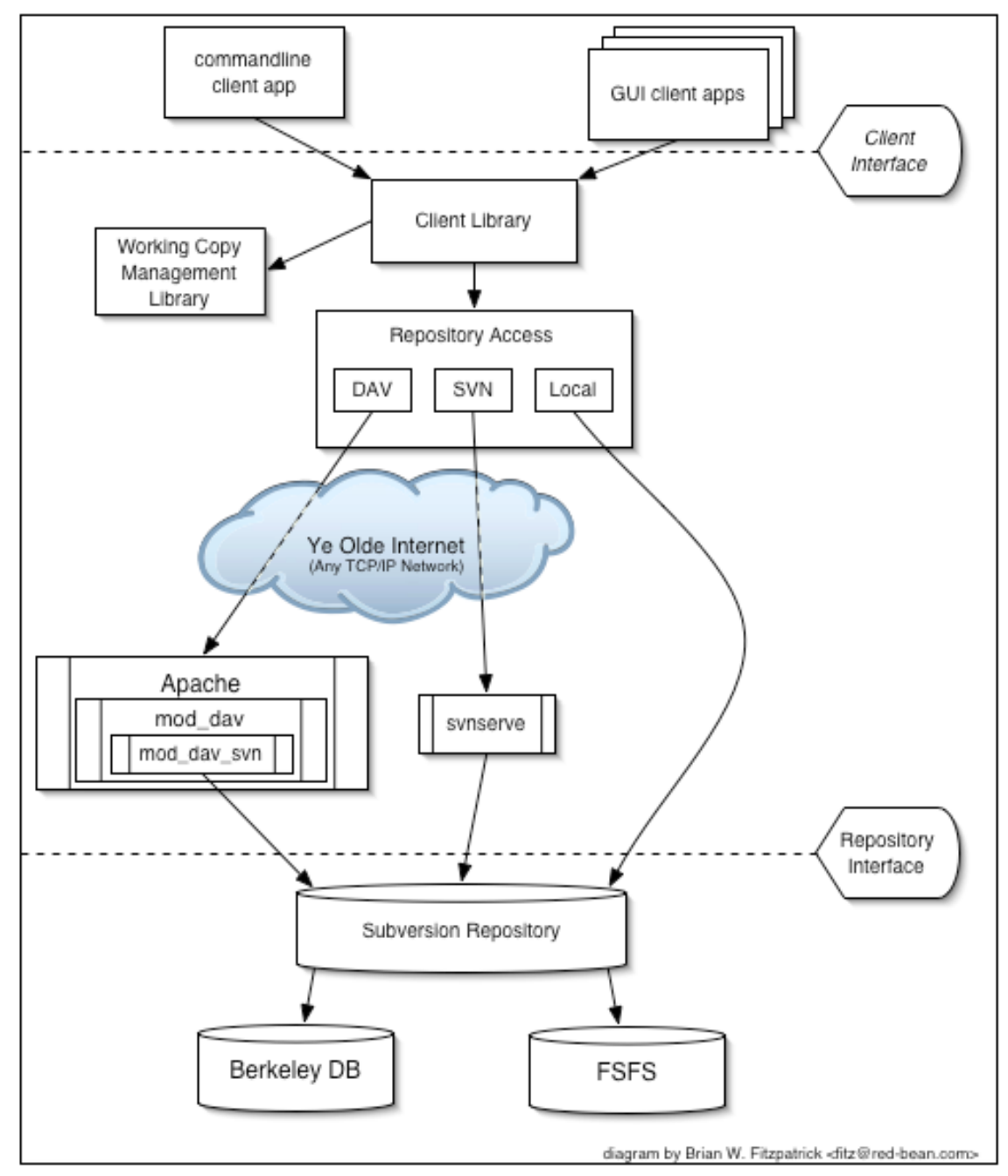
- La historia del archivo *B* será su historia más la historia del archivo *A* hasta la revisión 2.
- Las revisiones son los changesets que se fueron aplicando al repositorio.
- A esta característica se la llama *cheap copy*, y es la manera en la que subversion implementa los branches.

Subversion: *Ventajas*

- Subversion posee *atomic commit* para grupos de archivos.
- Si cualquier parte del commit falla, la transacción es vuelta atrás.
- Subversion usa *binary diffs*. En CVS el formato RCS está basado en texto (problemas para archivos binarios).
- Subversion usa *Vdelta* para lograr eficiencia en diffs binarios y de texto.

Diseño de Subversion

- Subversion maneja versiones por repositorio, no por archivo como lo hace CVS.
- Usa las extensiones *WebDAV* y *DeltaV* al protocolo HTTP; entonces, usa Apache 2 con un módulo especial para el servidor.
- Los repositorios Subversion tienen una interfaz web gratuita.
- Existe también la opción de usar un server *standalone*.


commandline client app
GUI client apps
Client Interface
Client Library
Working Copy Management Library
Repository Access
DAV
SVN
Local
Ye Olde Internet
(Any TCP/IP Network)
Apache
mod_dav
mod_dav_svn
svnserve
Repository Interface
Subversion Repository
Berkeley DB
FSFS
diagram by Brian W. Fitzpatrick <fitz@red-bean.com>

Branches

Si los branches se almacenan en el repositorio como simples directorios:

- es necesario tener una estructura que nos permita ubicarlos, o su línea de desarrollo principal.
- *Propuesta*: crear tres directorios en la raíz del repositorio...

Branches

- *trunk* (línea principal de desarrollo),
- *branches* (experimentales)
- *tags* (en general corresponden a releases).

  De esta forma, hacer un branch es tan simple como copiar 'trunk' a 'branches/mi_branch' para crear el branch 'mi_branch'.

Usando subversion

- Instalar un cliente de subversion
- Crear un repositorio:
  svnadmin create /path/to/repos
- Crear una estructura de directorios con tags y branches en el dir actual. Luego:
  svn import . file:///repos_path
  cd /home/mc/project
  svn import . file:///repos_path/trunk/project -m "Importar fuentes del proyecto"
  svn co file:///repos_path/trunk/project svn_project

Usando subversion

- Para tener un servidor en red hay que instalar Apache2 con el módulo libapache2-svn
- Otra alternativa es instalar el svnserve (más sencillo pero menos conveniente)
- También podemos usar un repositorio público para probar; ver:

  https://www.apache.org/dev/version-control.html

Usando subversion

- Creamos un directorio para albergar el proyecto:
  mkdir /home/mc/awpool
  cd /home/mc/awpool
- Bajamos el proyecto:
  svn checkout http://svn.apache.org:80/repos/asf/
- Ahora podemos modificar los archivos, puede ser que el commit esté reservado a determinados usuarios.

Comandos básicos de subversion

- co/checkout: hace una *working copy* del repositorio para poder modificarla.
- ci/commit: hace un *commit* de los cambios locales al repositorio.
- up/update: hace un *update* de la working copy para reflejar los cambios desde el último update.
- add, remove/rm: agrega o elimina archivos
- copy/cp, move/mv: copia o mueve manteniendo la historia.

Comandos básicos de subversion

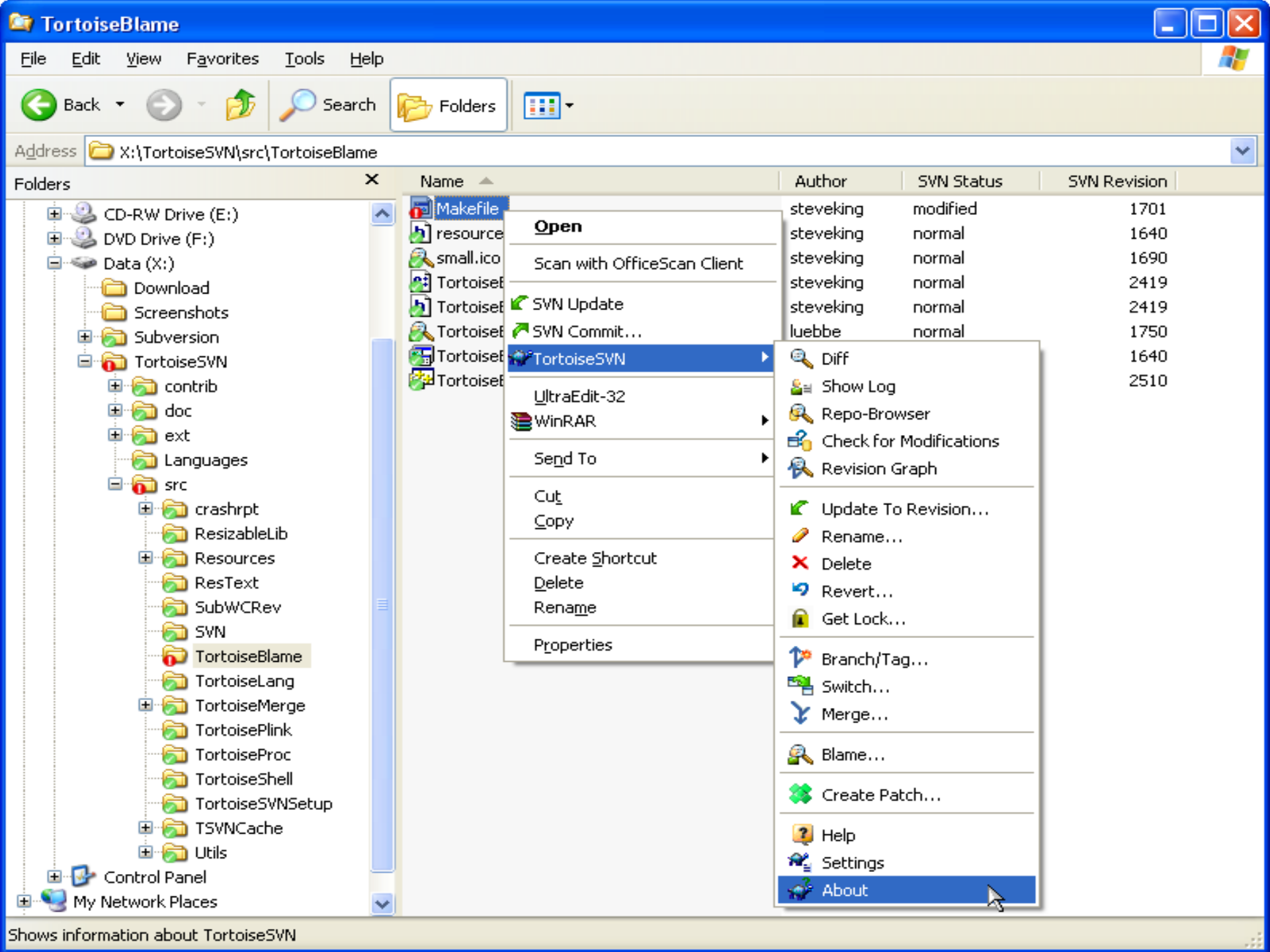
- *merge*: equivalente a cvs update -j, mezcla los cambios de otra locación en la working copy.
- *switch*: cambia el working copy a otro branch.
- *diff*: obtiene la diferencias entre la working copy y los últimos fuentes actualizados o el directorio actual.
- *log*: muestra los *log entries* para un recurso.

Ciclo básico de trabajo

- Update de la copia local: update
- Hacer cambios: add, delete, copy, move
- Examinar los cambios realizados: status, diff, revert
- Mezclar los cambios de otros: update, resolved
- Hacer un commit de los cambios: commit

GUIs

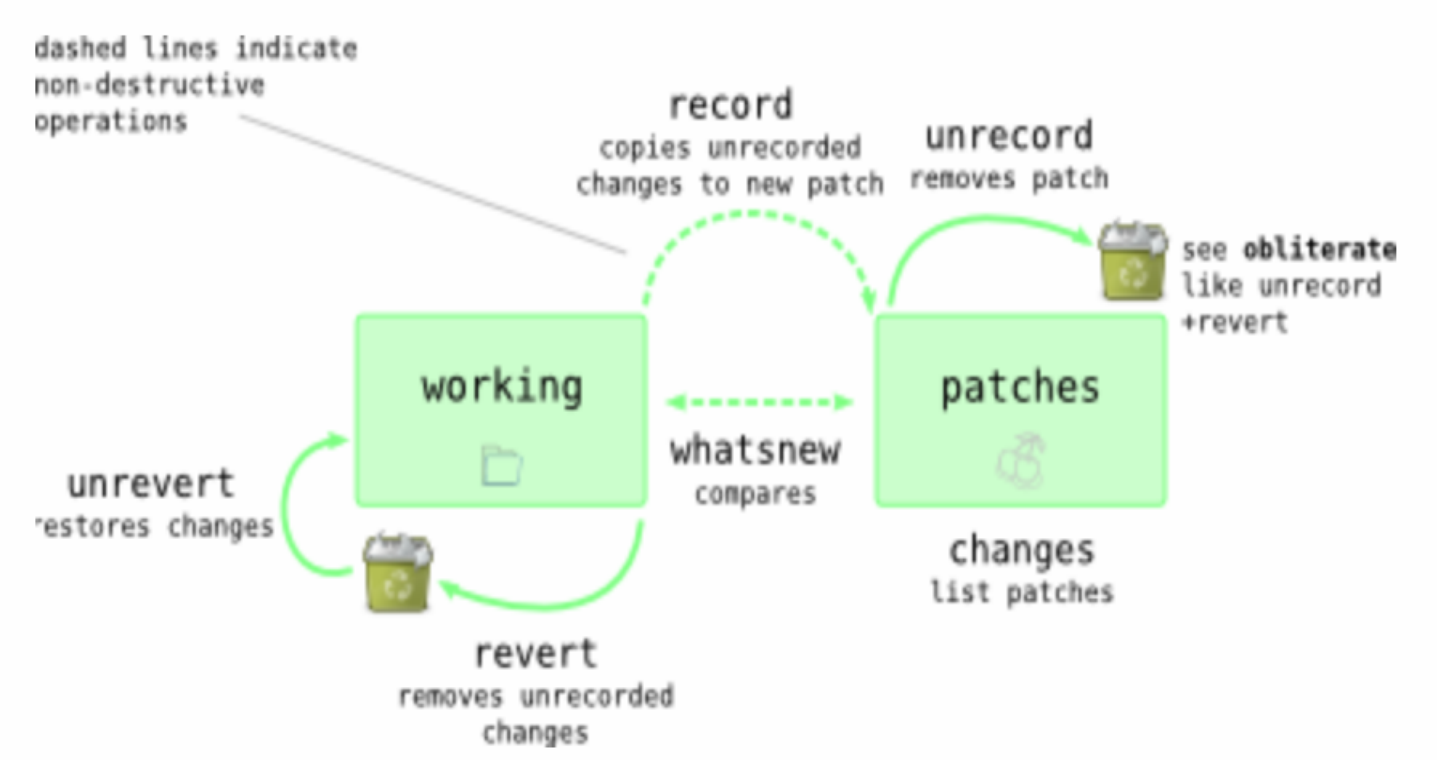
- Muy buen cliente para Windows (Gnu general public license): `http://tortoisesvn.tigris.org/`
- Cliente multiplataforma (Gnu general public license): `http://rapidsvn.tigris.org/`
- *Latest source*: The source is stored in a Subversion repository (of course!) Now you can check out the source code for RapidSVN The URL for the latest code is: http://rapidsvn.tigris.org/svn/rapidsvn/trunk You will be prompted for username and password. For read access enter "guest" with an empty password.


TortoiseBlame
File Edit View Favorites Tools Help
Back Search Folders
Address X:\TortoiseSVN\src\TortoiseBlame
Folders
CD-RW Drive (E:)
DVD Drive (F:)
Data (X:)
Download
Screenshots
Subversion
TortoiseSVN
contrib
doc
ext
Languages
src
crashrpt
ResizableLib
Resources
ResText
SubWCRev
SVN
TortoiseBlame
TortoiseLang
TortoiseMerge
TortoisePlink
TortoiseProc
TortoiseShell
TortoiseSVNSetup
TSVNCache
Utils
Control Panel
My Network Places
Name
Author
SVN Status
SVN Revision
Makefile
resource
small.ico
steveking modified 1701
steveking normal 1640
steveking normal 1690
steveking normal 2419
steveking normal 2419
luebbe normal 1750
1640
2510
Open
Scan with OfficeScan Client
SVN Update
SVN Commit...
TortoiseSVN
UltraEdit-32
WinRAR
Send To
Cut
Copy
Create Shortcut
Delete
Rename
Properties
Diff
Show Log
Repo-Browser
Check for Modifications
Revision Graph
Update To Revision...
Rename...
Delete
Revert...
Get Lock...
Branch/Tag...
Switch...
Merge...
Blame...
Create Patch...
Help
Settings
About
Shows information about TortoiseSVN

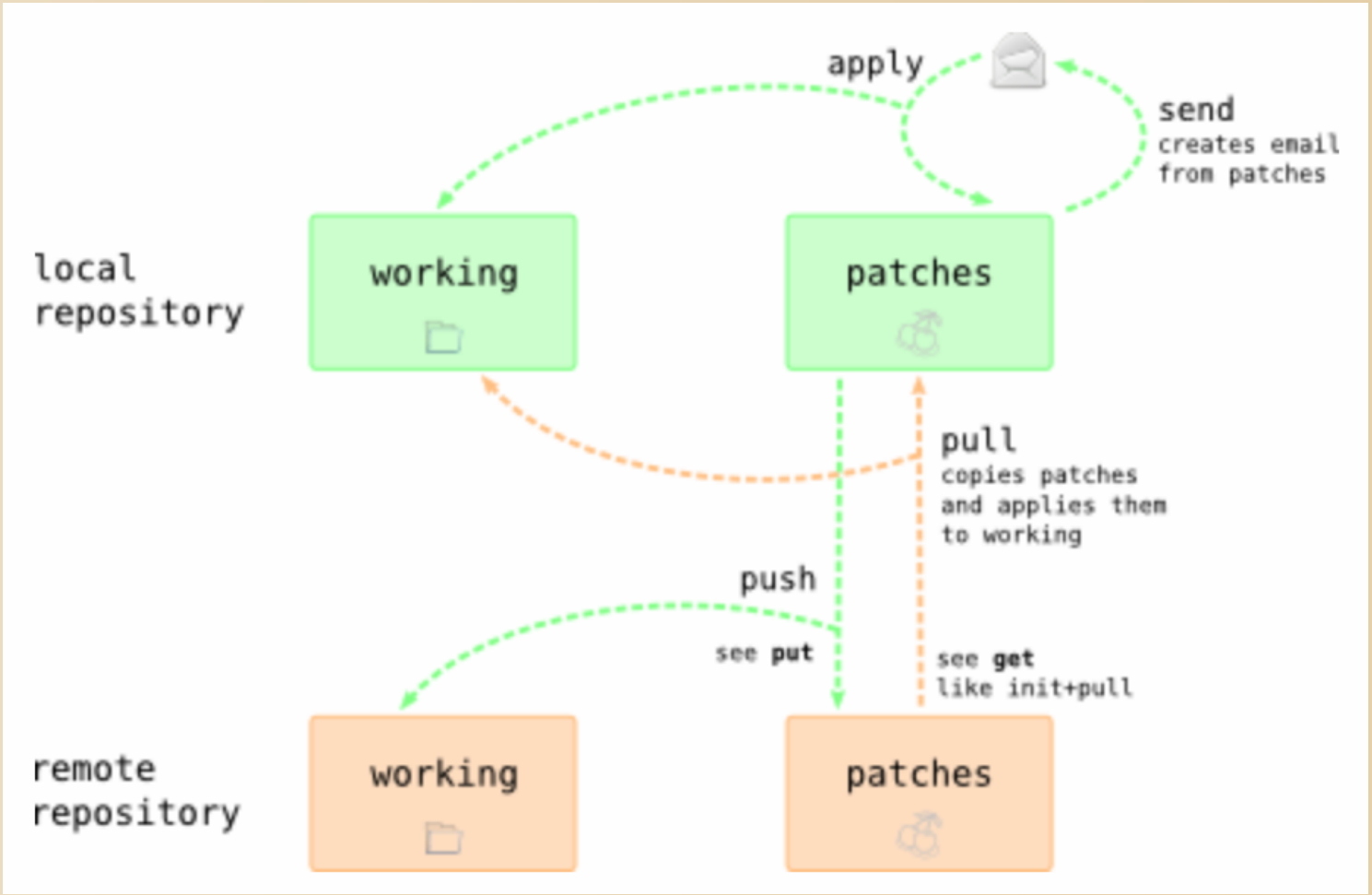
Darcs

- Es un SGV relativamente nuevo: primer lanzamiento en 2003, versión más reciente en abril de 2018.
- Fue creado por el físico David Roundy.
- El nombre es un acrónimo recursivo: *Darcs Advanced Revision Control System*.
- Muchas operaciones se pueden hacer con comandos *push* y *pull*, lo cual hace que tenga menos comandos.
- Se basa en *patches* para el manejo de cambios.
- Los patches se almacenan parcialmente ordenados.

Darcs: Modelo



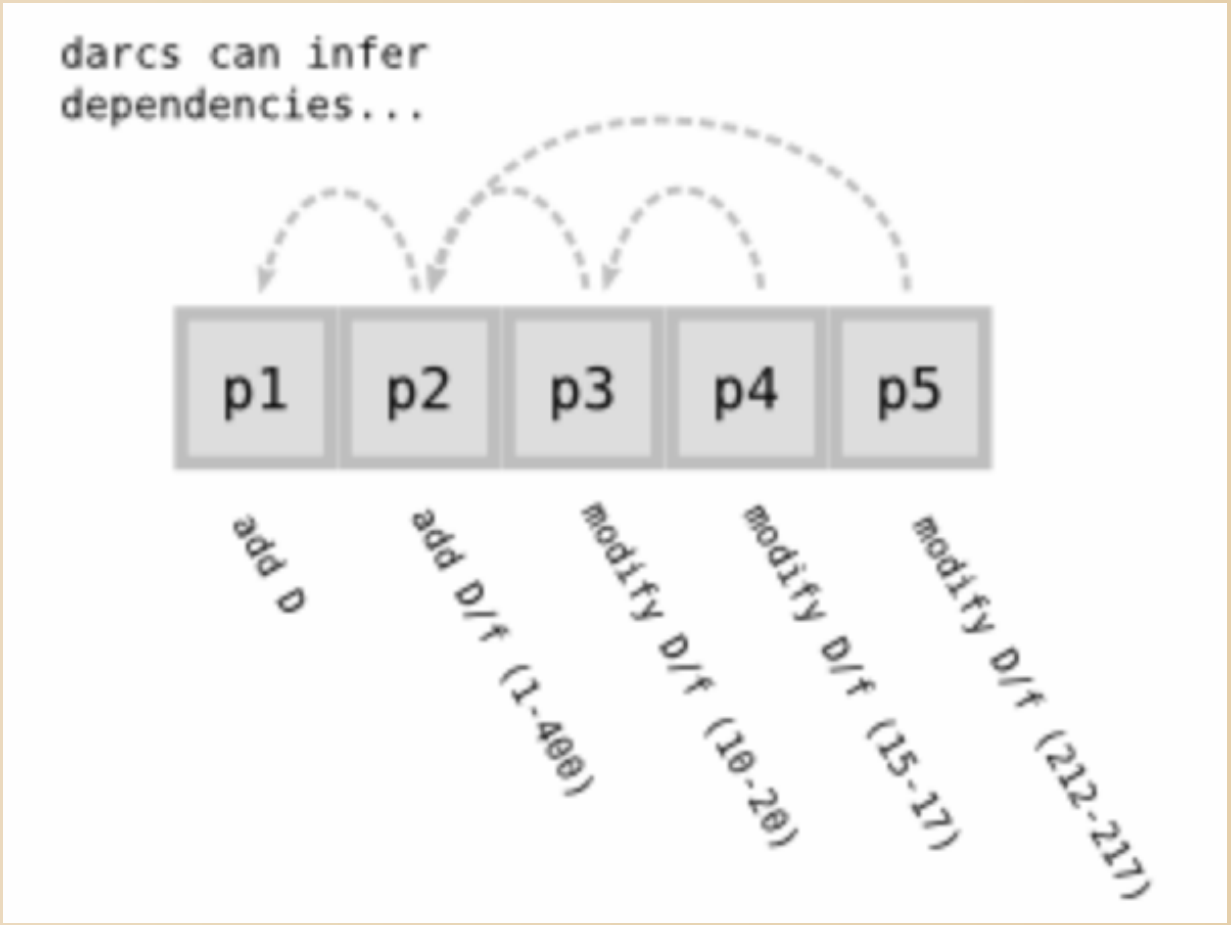
Darcs: Modelo



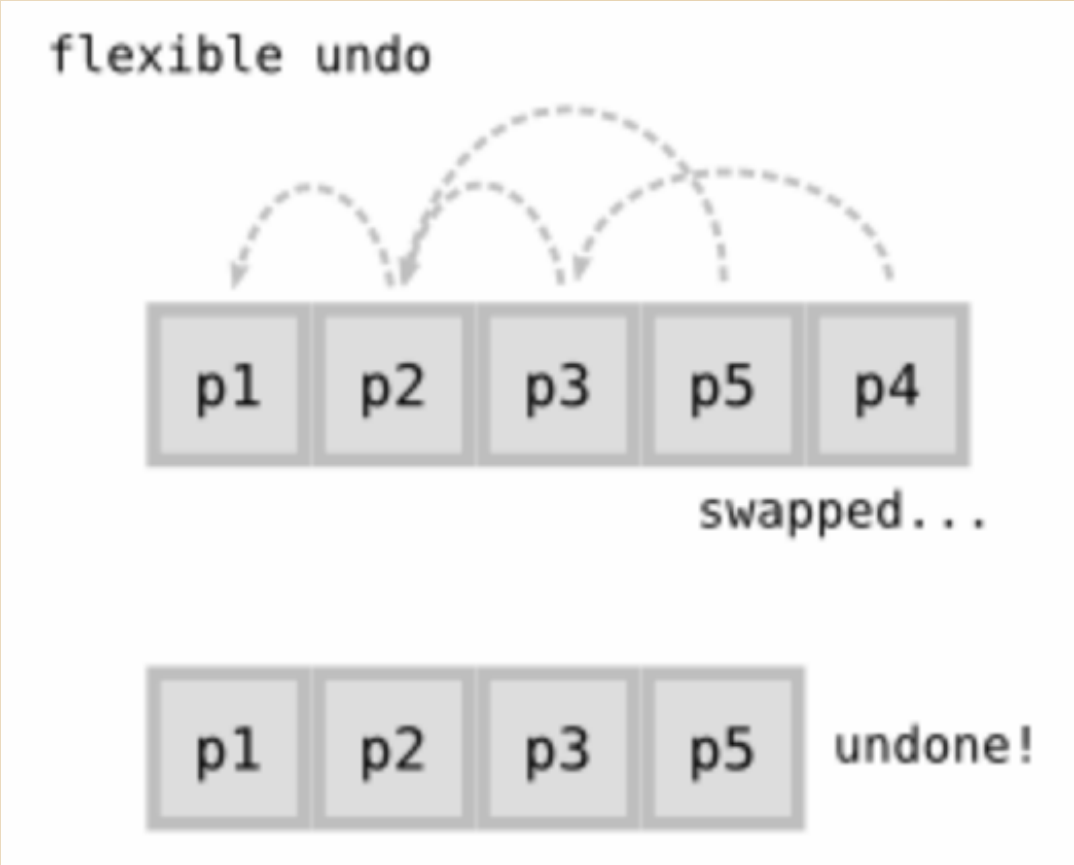
Repositorios

- Un repositorio es un conjunto de *patches*.
- Hay dos vistas:
  - *Abstracta*: conjunto parcialmente ordenado
  - *Concreta*: secuencia con el orden en que se aplicaron
- Los patches se insertan con sus respectivas dependencias, dando lugar al orden parcial.
- Este orden es útil a la hora de *deshacer* patches, o al *mezclar* repositorios.

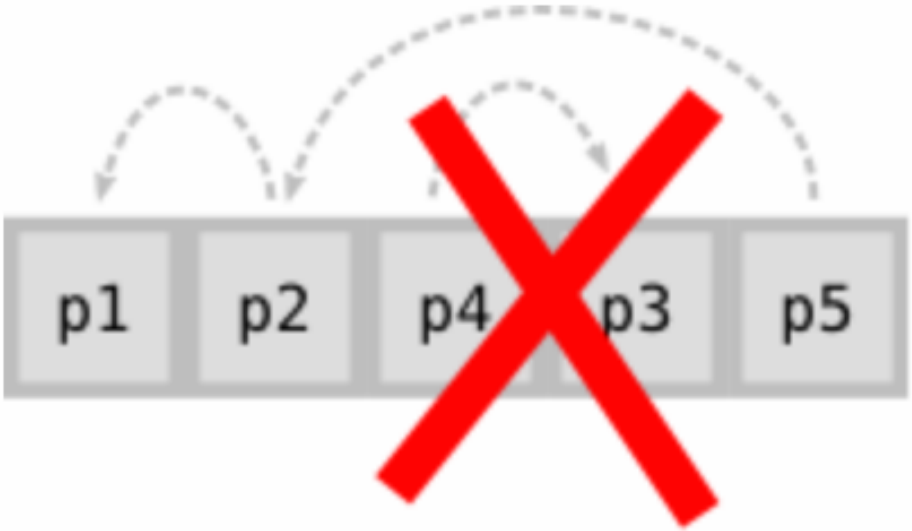
Orden parcial de patches



Deshacer p4



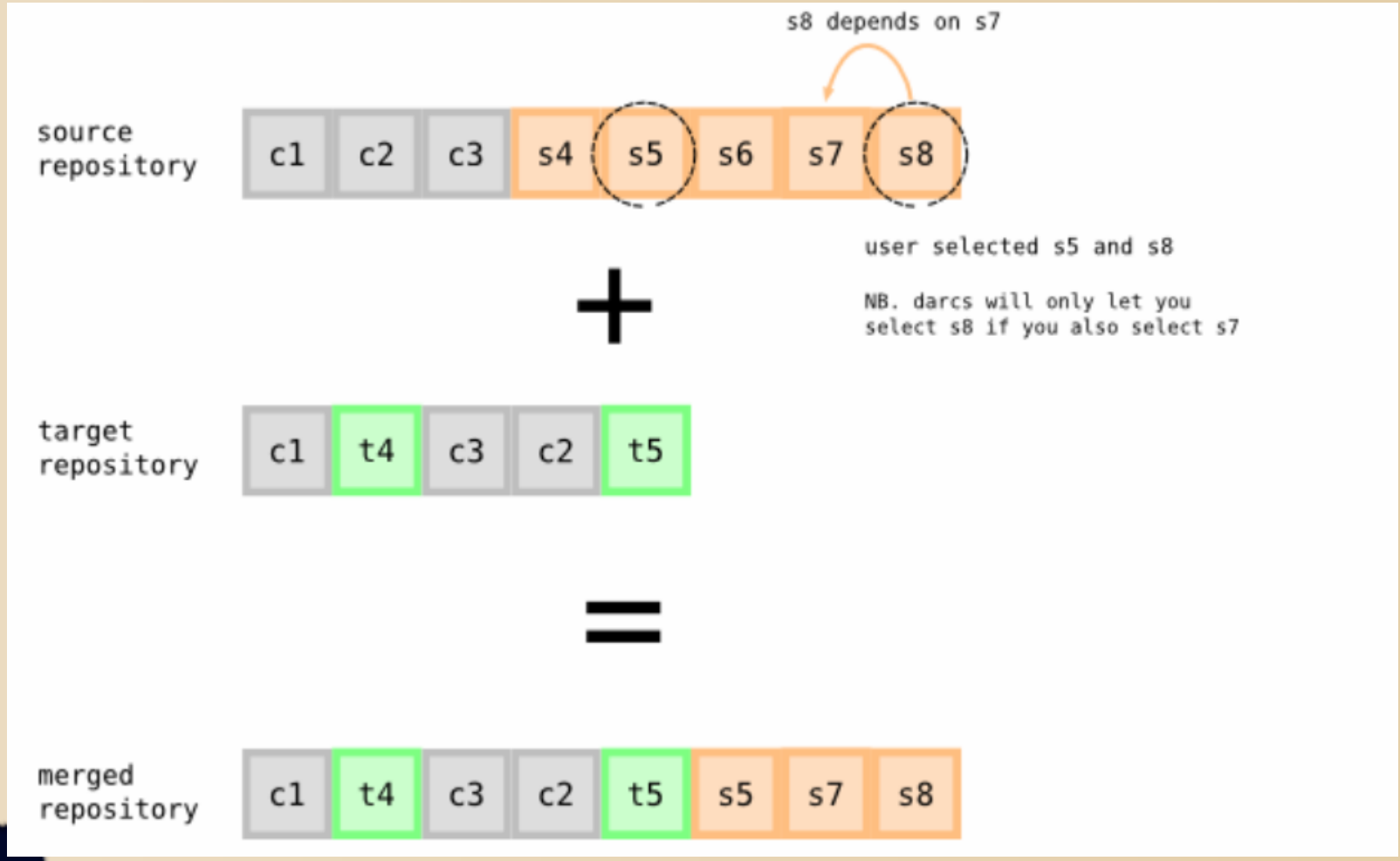
Deshacer p3?



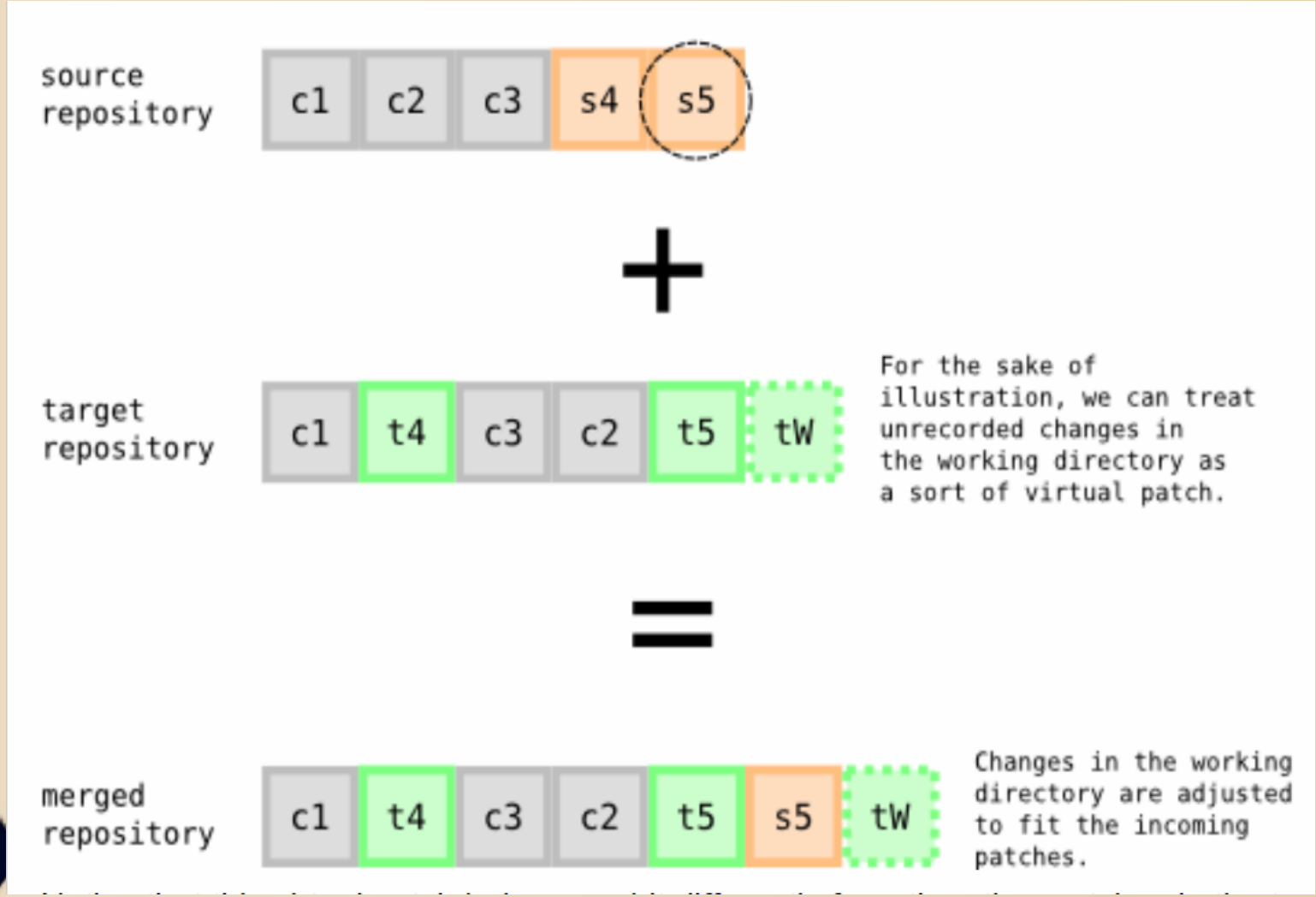
dependencies are enforced

you can only undo p3 if you undo p4 first

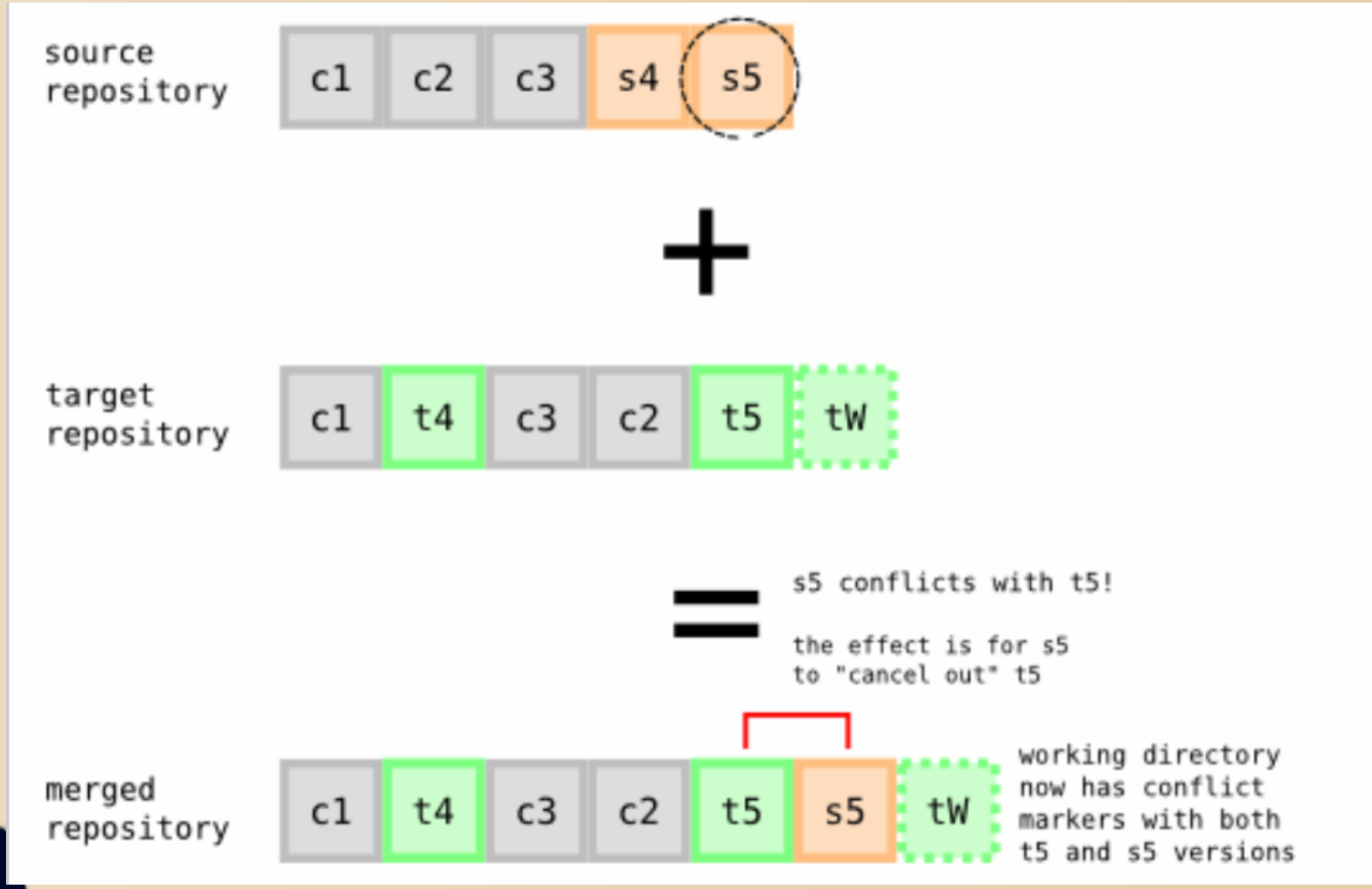
Merging de repositorios: *Cherrypicking*



Merging de repositorios: *Cambios sin guardar*



Merging de repositorios: *Conflicto*



Darcs: *Ventajas*

- Todas las copias de un repositorio son repositorios en sí mismos
- Esto facilita:
  - el trabajo con múltiples repositorios,
  - la creación de repositorios temporarios para desarrollar cosas nuevas, y
  - la publicación de éstos; se puede usar cualquier servidor Web.

Darcs: *Ventajas*

- Posee una muy buena integración con el correo electrónico, facilitando el envío de detalles de changesets por este medio.
- Está escrito en el lenguaje funcional *Haskell*, y por lo tanto es multiplataforma.
- Flexible: Para empezar un nuevo repositorio se hace un *init* en cualquier directorio.

Uso básico

Su uso es similar a CVS y SVN:

- Obtener una copia del repositorio
- Realizar los cambios
- Grabar los cambios
- Hacer un *pull* de los cambios de otros repositorios
- *Push* (enviar) cambios a los otros repositorios

Conclusiones

- Quizás la utilidad más importante de los SGV no provenga de la información misma, sino de la forma en la que se genera y almacena.
- El tener que pensar en *changesets* nos ayuda a trabajar de forma más ordenada y prolija.
- Los SGV afectan nuestra forma de crear software.

Conclusiones

- Dado esto, los SGV deben *acompañar y ajustarse* a la manera en la que concebimos y desarrollamos el software.
- Estos sistemas han revolucionado la colaboración entre los desarrolladores.
- Su utilidad va más allá del manejo de código fuente: se pueden (y en muchos casos *deben*) utilizar para administrar versiones de *todo documento relacionado con el proyecto*.

Bibliografía

- Open Source Development with CVS, 3rd Ed. M. Bar, K. Fogel. Capítulo 1, Capítulo 2 *hasta la página 25*. Disponible en:

  http://cvsbook.red-bean.com/OSDevWithCVS_3E.pdf

- *Version Control with Subversion – For Subversion 1.7 (Compiled from r5648)*. B. Collins-Sussman, B. W. Fitzpatrick, C. M. Pilato. Prefacio, Capítulos 1 y 2. Disponible en:

  http://svnbook.red-bean.com/

Otro material: Links útiles

- Sitio principal CVS: http://www.cyclic.com/
- CVS para windows: http://www.wincvs.org/
- Tutorial CVS: https://wiki.gentoo.org/wiki/CVS/Tutorial
- Free CVS hosting: https://freepository.com
- CVS server anónimo de Mozilla: anonymous@cvs-mirror.mozilla.org:/cvsroot con password “anonymous”.

Otro material: Links útiles

- Sitio principal de Subversion: http://subversion.tigris.org/
- Tutorial de subversion: https://subversion.apache.org/quick-start
- Server anónimo: http://svn.apache.org/repos
- Página de Darcs: http://www.darcs.net
- Manual de Darcs: http://darcs.net/manual/