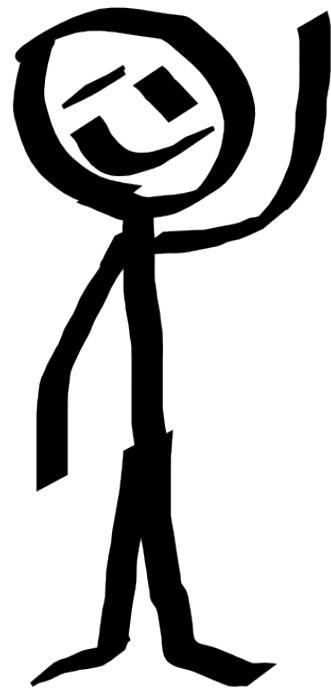


“For the first 10 years of kernel maintenance, we literally used tarballs and patches, which is a much superior source control management system than CVS is, but I did end up using CVS for 7 years at a commercial company [presumably Transmeta] and I hate it with a passion. When I say I hate CVS with a passion, I have to also say that if there are any SVN (Subversion) users in the audience, you might want to leave. Because my hatred of CVS has meant that I see Subversion as being the most pointless project ever started. The slogan of Subversion for a while was ‘CVS done right’, or something like that, and if you start with that kind of slogan, there's nowhere you can go. There is no way to do CVS right.”

--Linus Torvalds at Google Talk on git

Intended Audience

you



{BS,
MS,
PHD}

git: Design and Implementation

This really cool and interesting talk is brought to you by Richard P. Spillane of the FSL, CSGSC and:

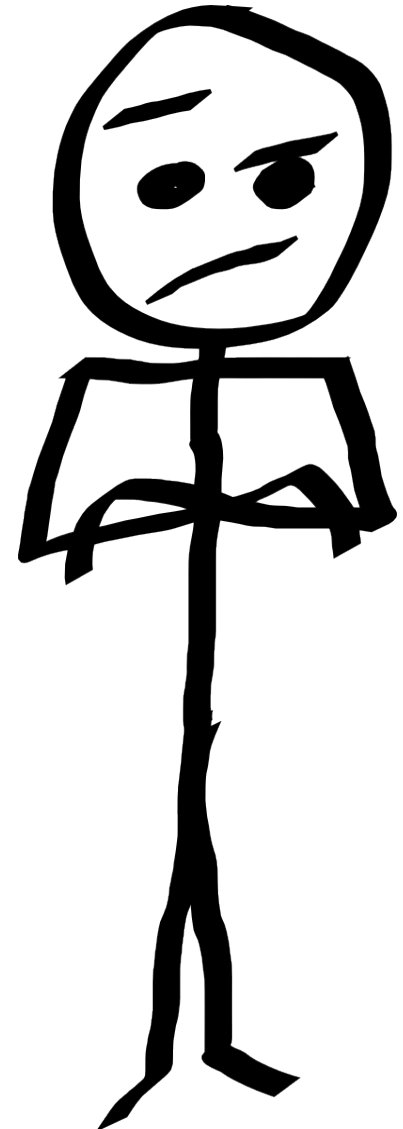
WICS

Find them at...

“blah blah blah dot **cs.sunysb.edu/~wics**”

WHY DO I CARE?

- You care about your time
- You like to use the best tools
- You want perspective
- You want to learn an awesome design pattern that will inspire you and come in handy
- Git **IS** the best SCM...



Git is the best SCM available

- Git is FASTER than all other scms
- Git is STABLE as it is the sole repo of the kernel
- Git is SCALABLE (unlike slow CVS and SVN)
- Git is SIMPLE as it was coded in 4 days

Git is FASTER than all other scms

- 3500Mhz CPU
- 1GB Ram
- Linux, ext3
- 12456 changesets
- ~30000 files from Mozilla Repository

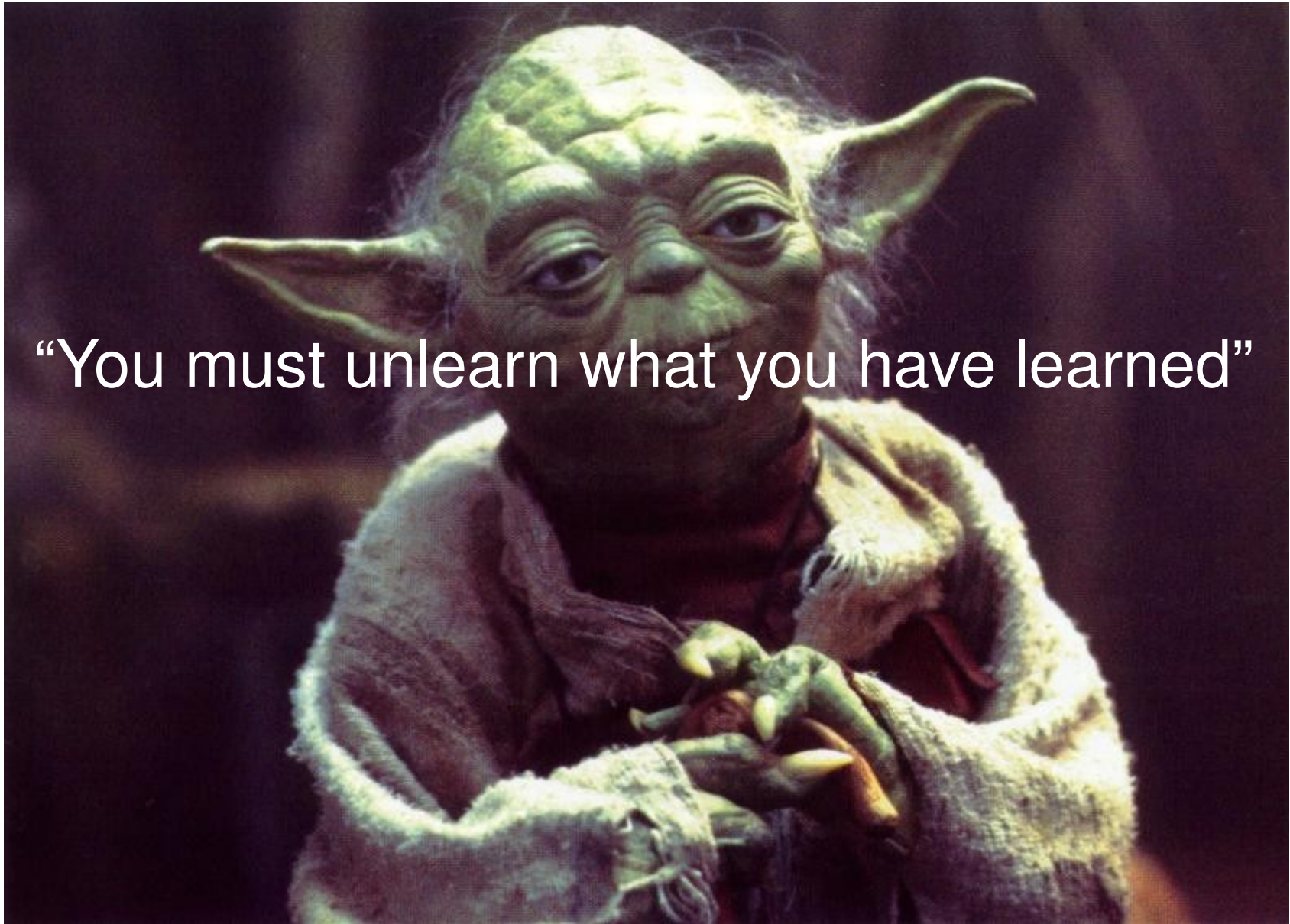


And now, the talk

(-cvs₁) +
snapshots₂ +
patches₃ +
object stores₄ = git

(1) Forget CVS

“You must unlearn what you have learned”

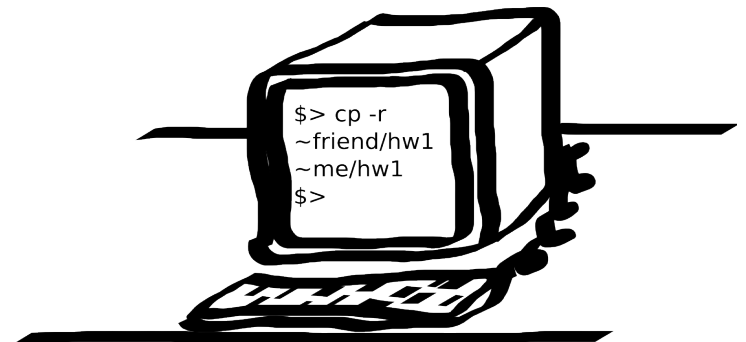


Centralized and Slow

- SVN, CVS, and RCS all use centralized source databases creating unnecessary work-flow dependencies: “the repo's on fire, can't commit”
- SVN tries to improve on CVS by providing atomic commits and better branching
- SVN branches flatten history on merge (see [svnmerge.py](#))
- SVN branches are very costly to merge for many commits

Back to basics: more like 'cp -r'

- When giving tutorials, I advise to think of 'git' as 'cp -r'++
- In CVS/SVN you only talk to the server (checkin/checkout)
- In git, you talk to whoever you want to send updates to, just like if they copied your updated directory over theirs directly

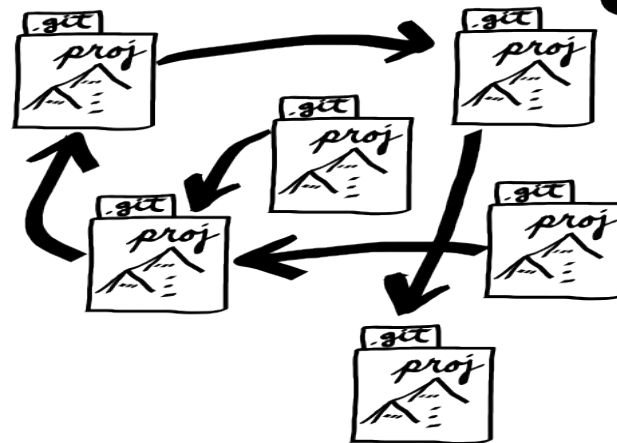


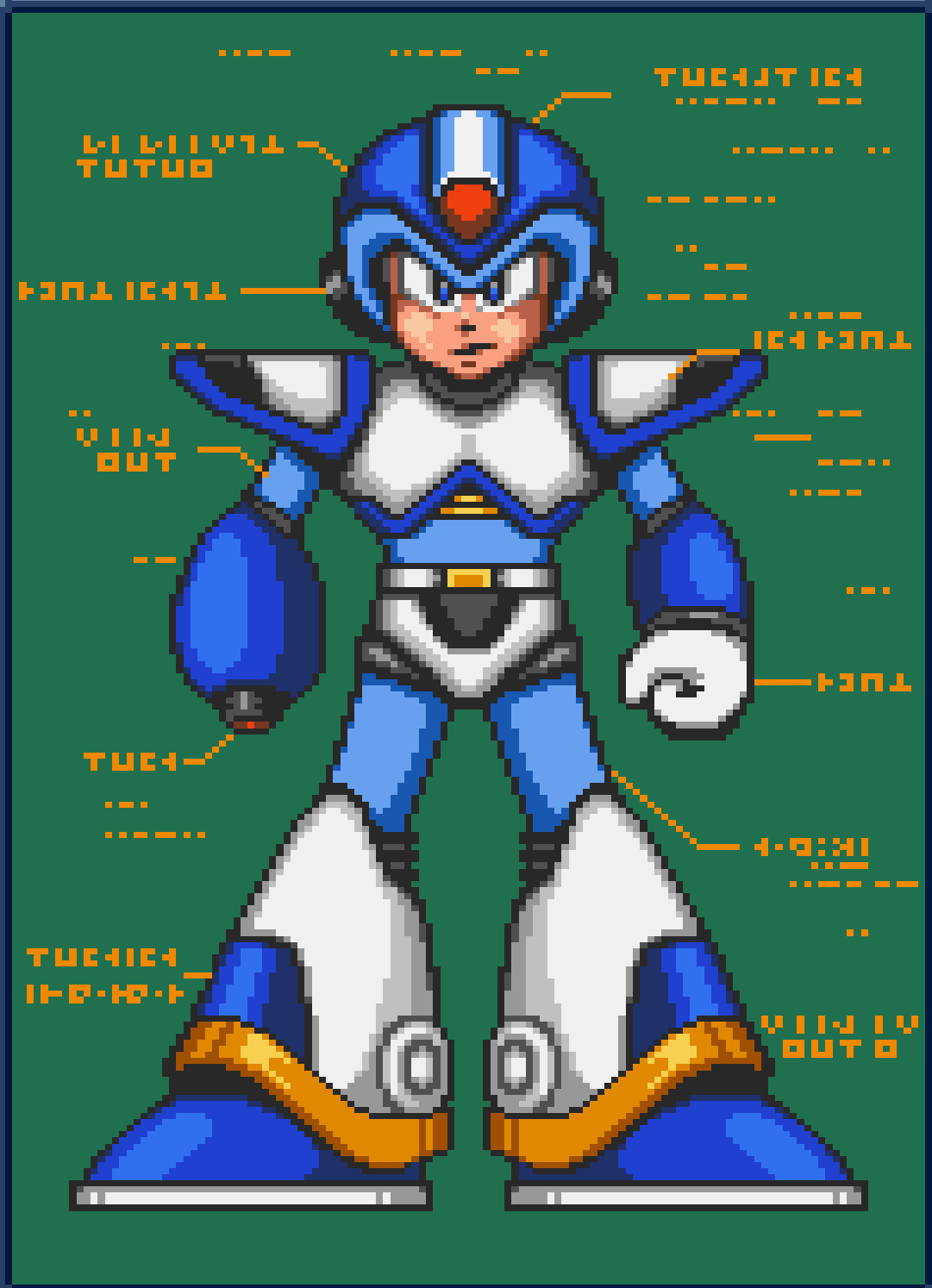
Tracking vs. Sharing

- Git was first designed to manage a single directory tree with a single user (Tracking)
- It was later extended to support 'cp -r' and later 'scp -r' (Sharing)

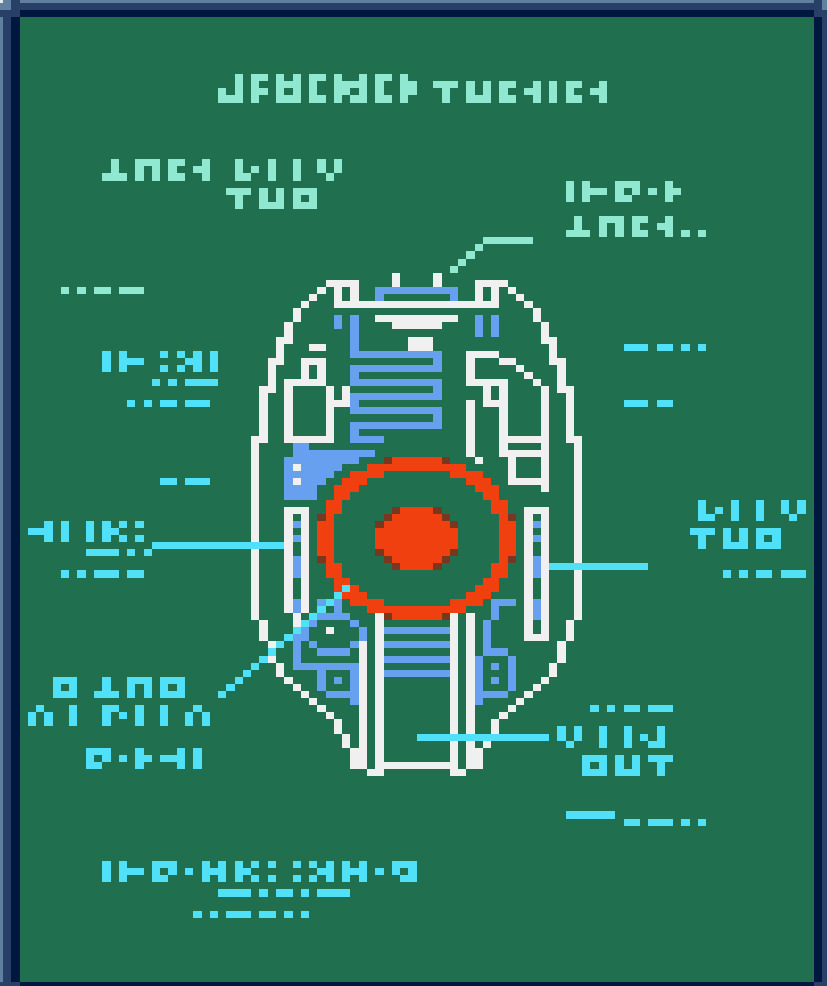
(Tracking)
"...first there
was one..."

(Sharing)
"then many..."





YOU GET
SVN SLAYER

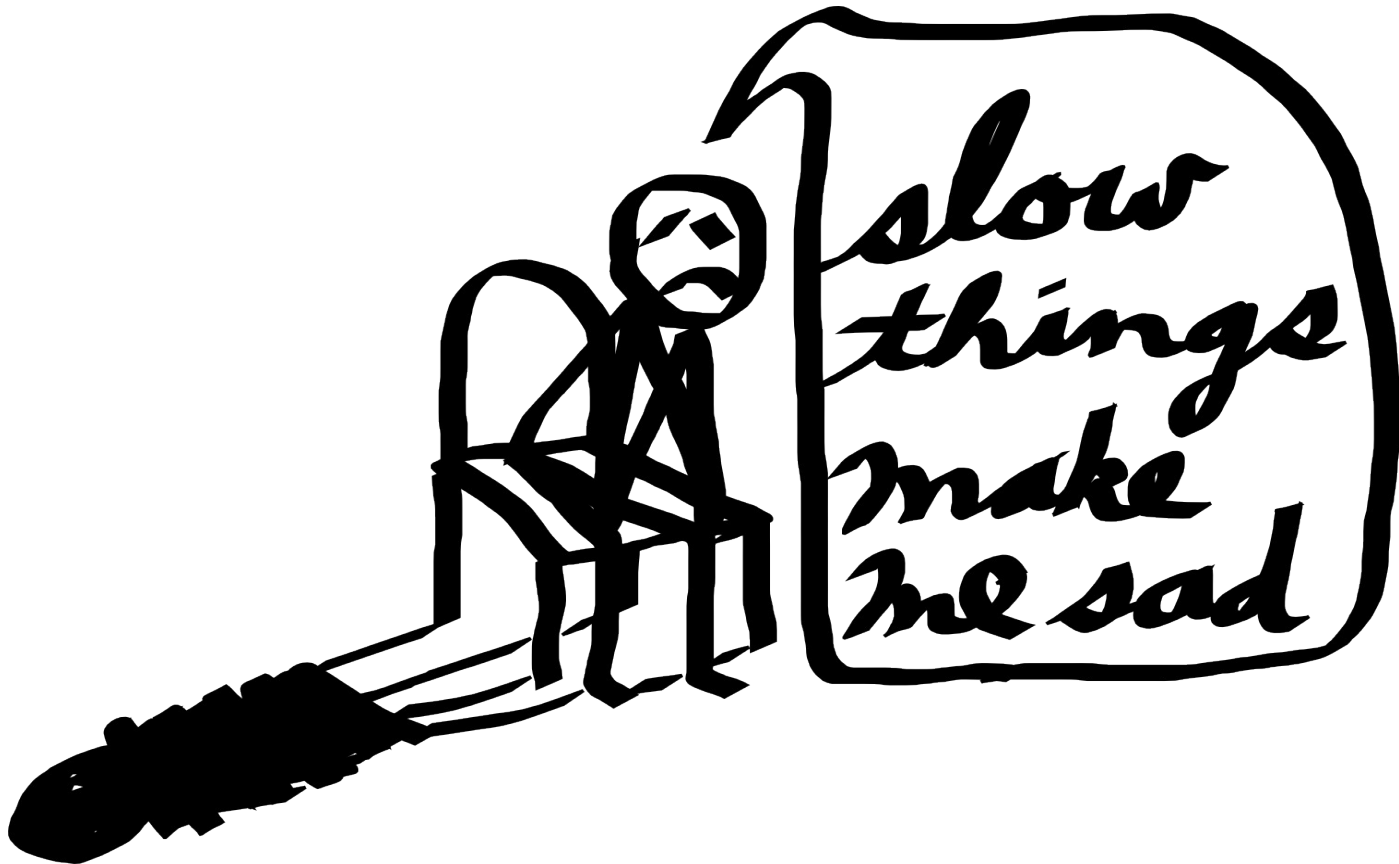


(2) Snapshots

- Backups give you two advantages:
 - Redundancy: one machine dies, you have another
 - History: you can undo mistakes
- Problem:
 - Backups are SLOW, halting user activity (BAD)
 - ...or forcing you to record non-instantaneous backups (NOT GOOD)

Backups cont.

- RAID (throw hardware at the problem)
 - Offers a fast solution to redundancy, but not history



Similar Problem

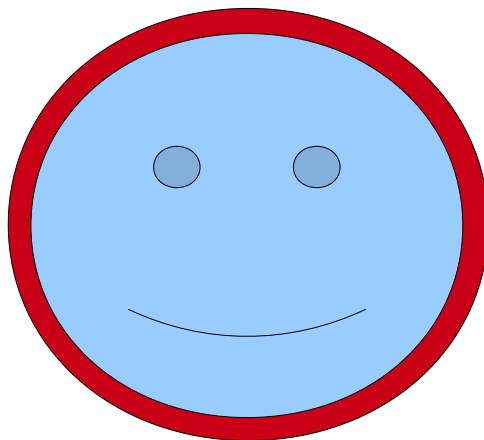
- Its sometimes useful to duplicate a current running program
- Old operating systems would copy all memory allocations from one program to the other
- Someone asked: “Why not let them share the same memory? Then we don't have to copy”

Copy-On-Write

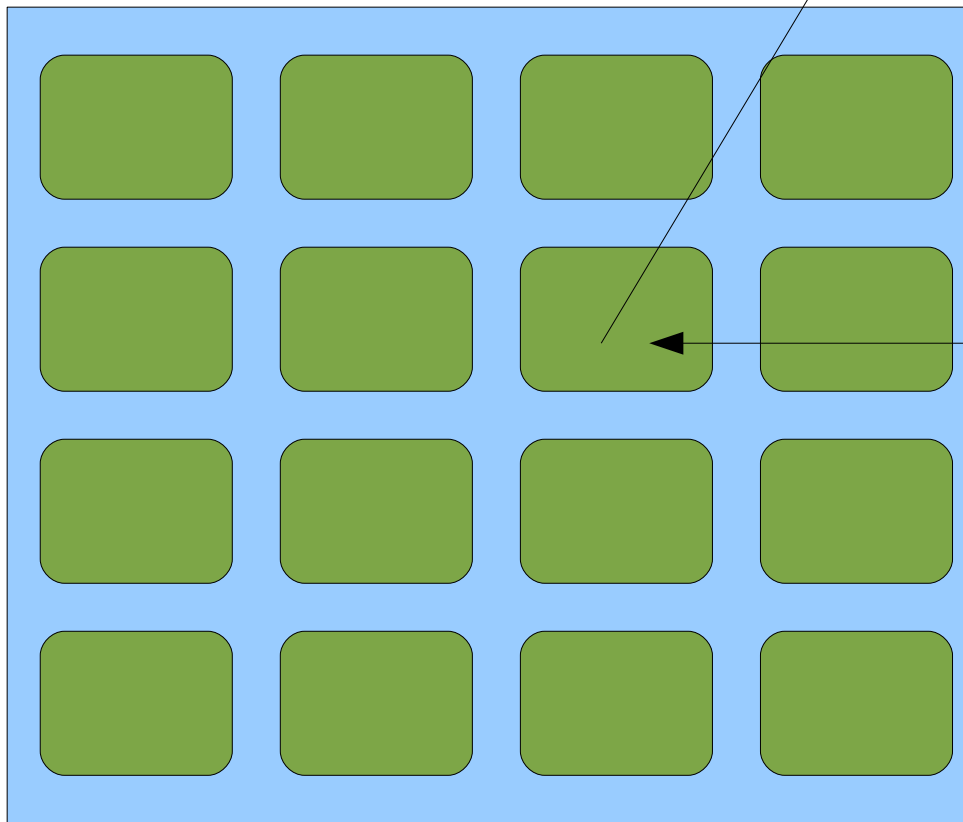
- When you want to 'duplicate' data, simply do nothing.
- Question: What if somebody modifies the duplicate?
- Answer: Have them wait a second while you copy the part they intend to modify. Keep track of which parts have been copied into the duplicate.

READ

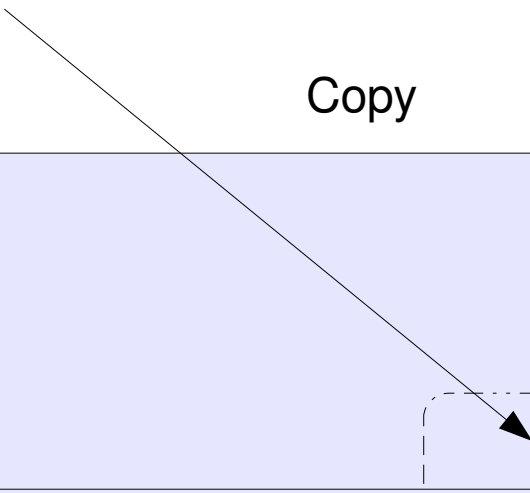
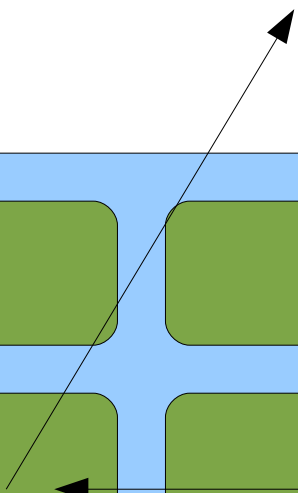
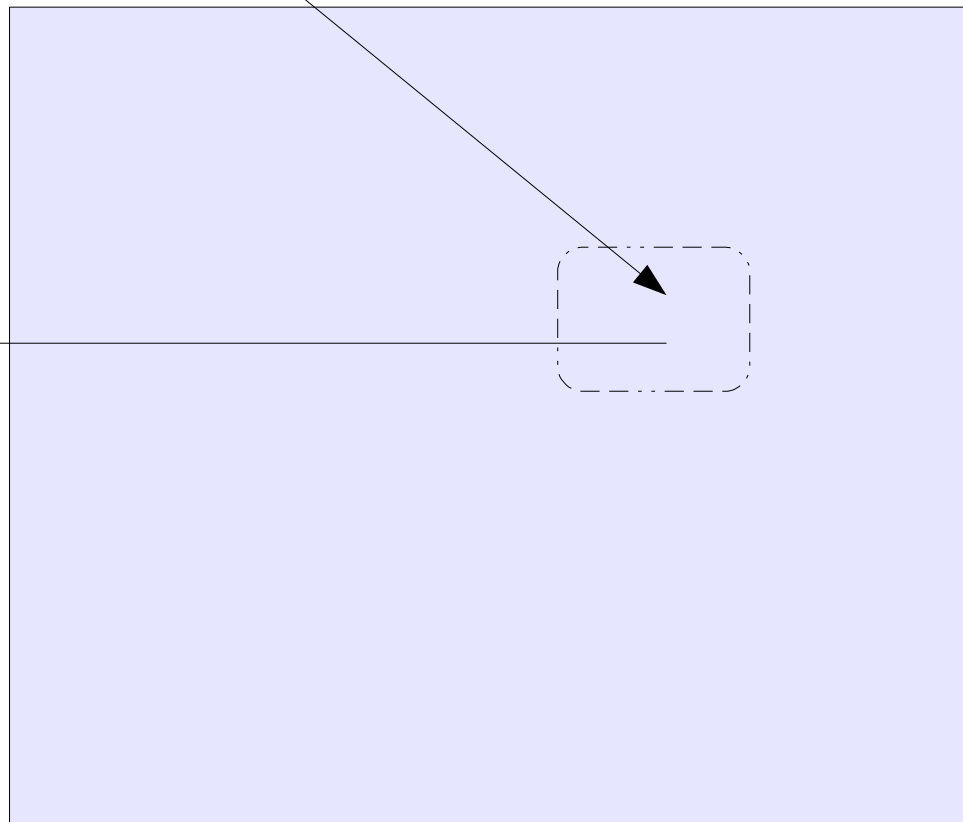
User



Original

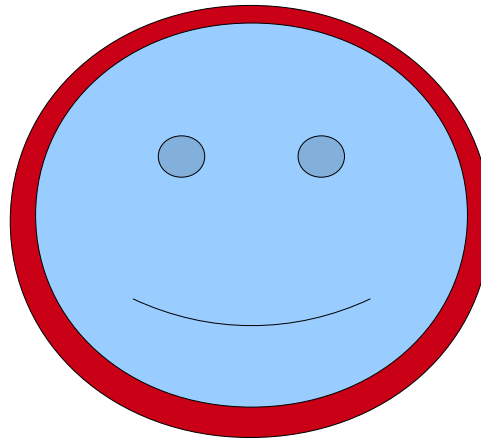


Copy

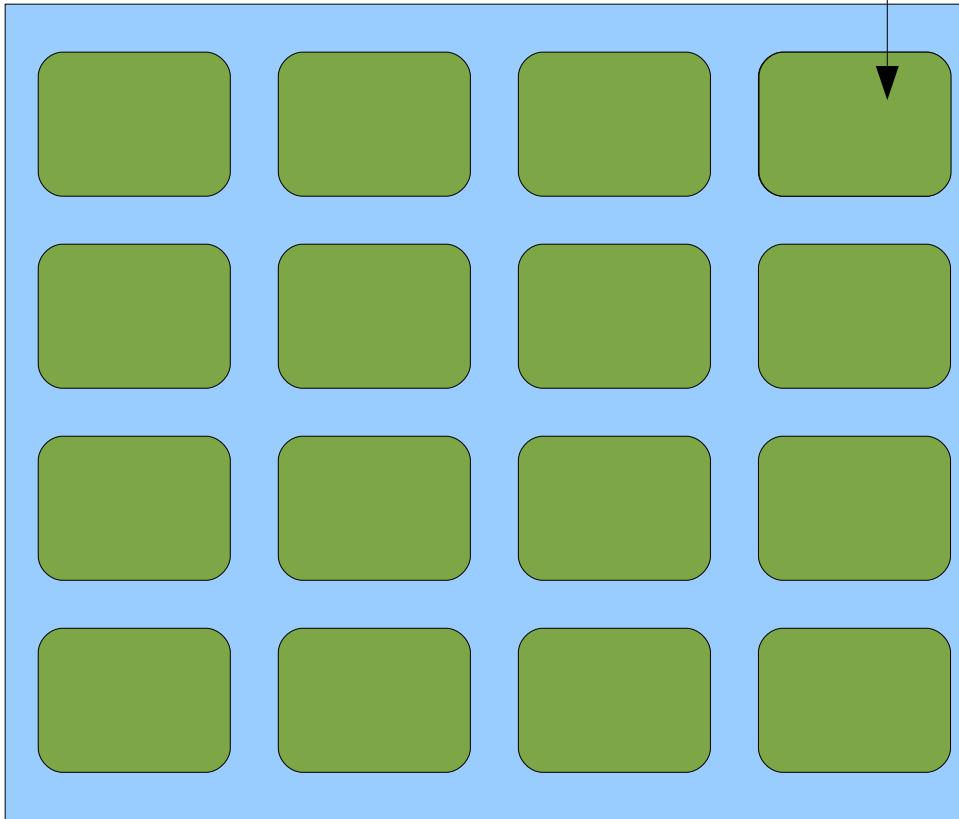


WRITE ORIGINAL

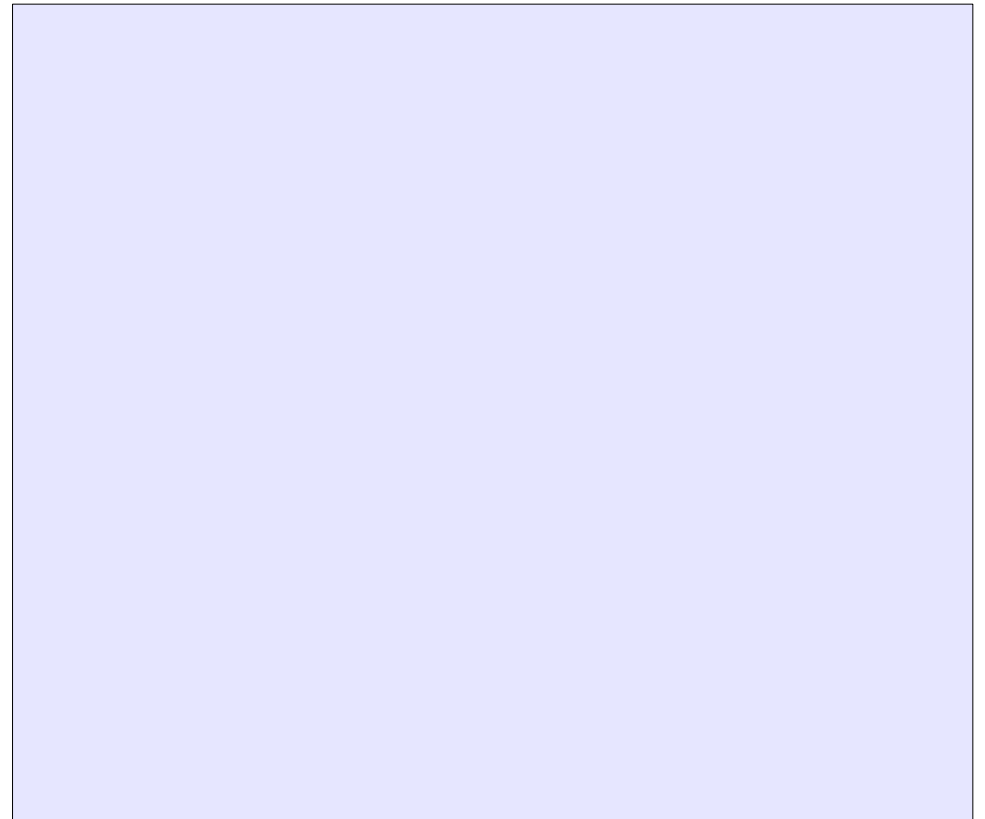
User



Original

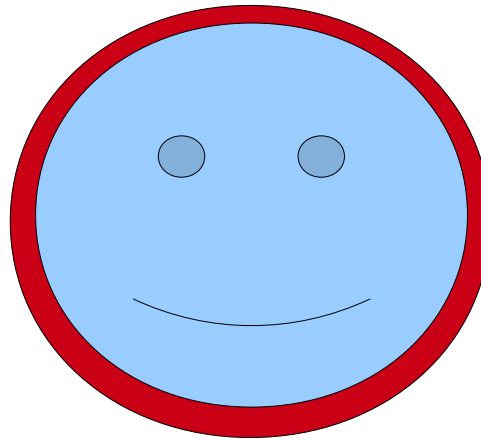


Copy

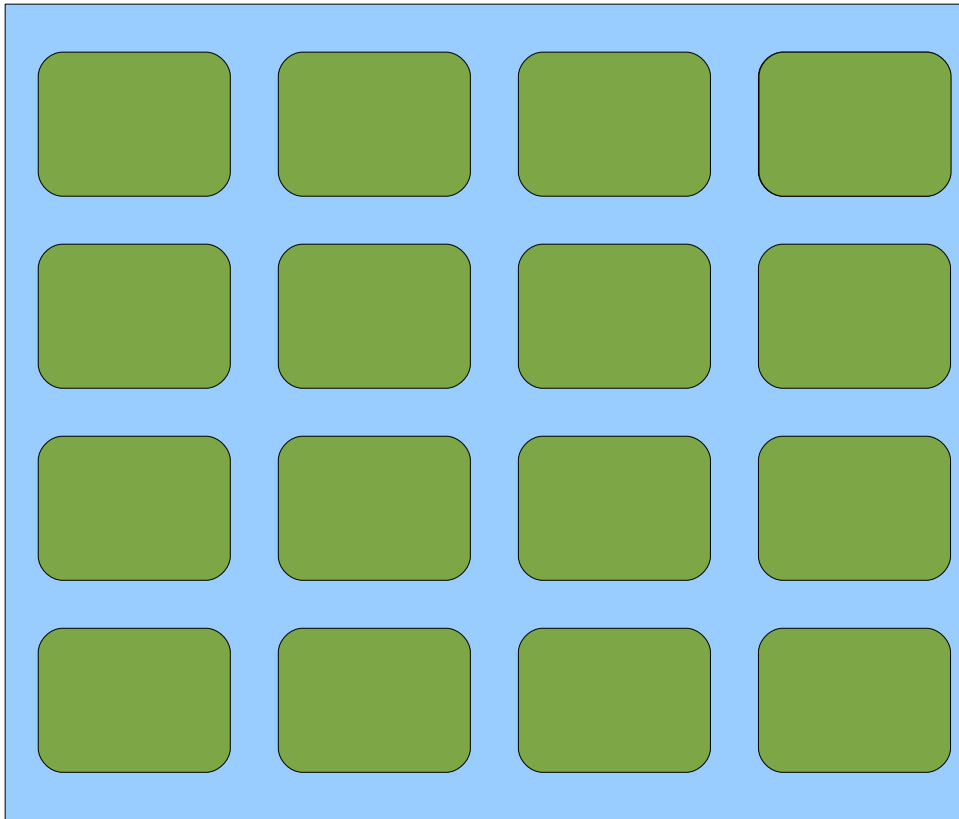


WRITE COPY

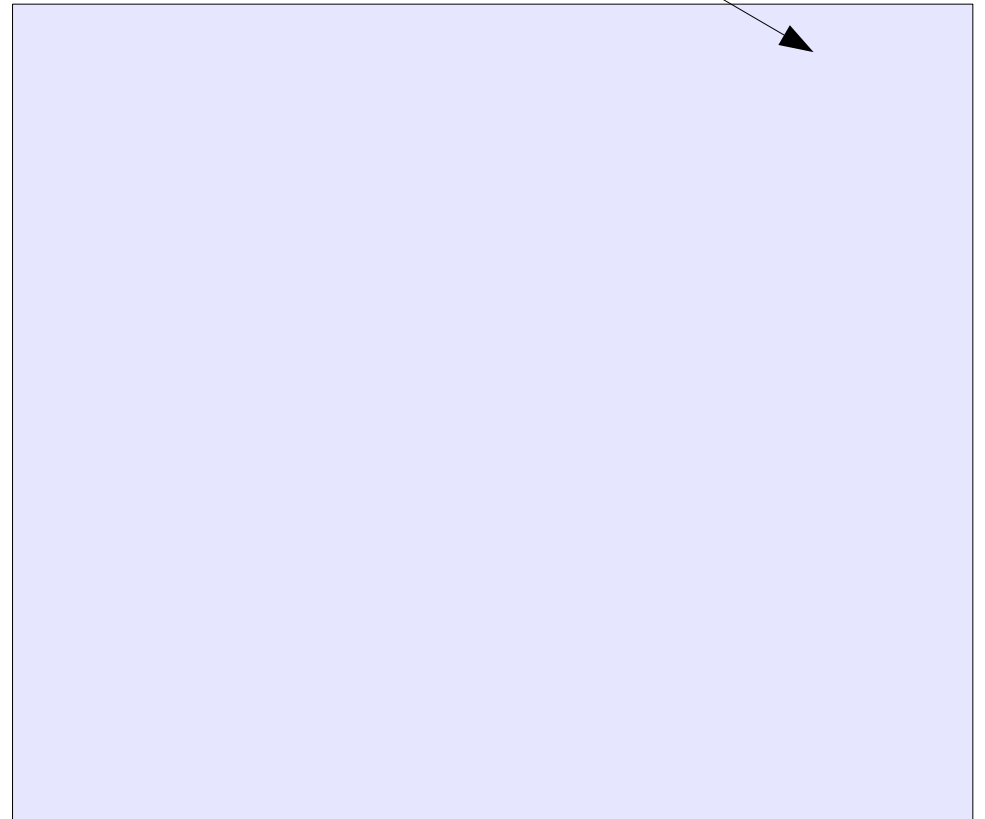
User



Original



Copy



NetApp

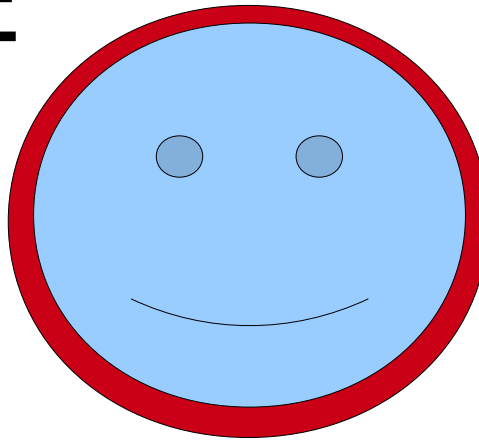
- Solve redundancy with RAID
- Solve history with copy-on-write
- RAID makes a whole FS survive one or two disk failures
- Copy-on-write can assume a mostly single-image block device

Copy-on-write in NetApp

- WAFL: Write anywhere file logging
- Simplify copy-on-write design by making copy-on-write work on a *tree*
- Think of FS as a tree where no node in the tree is ever larger than *4 kilobytes*
- YAY?

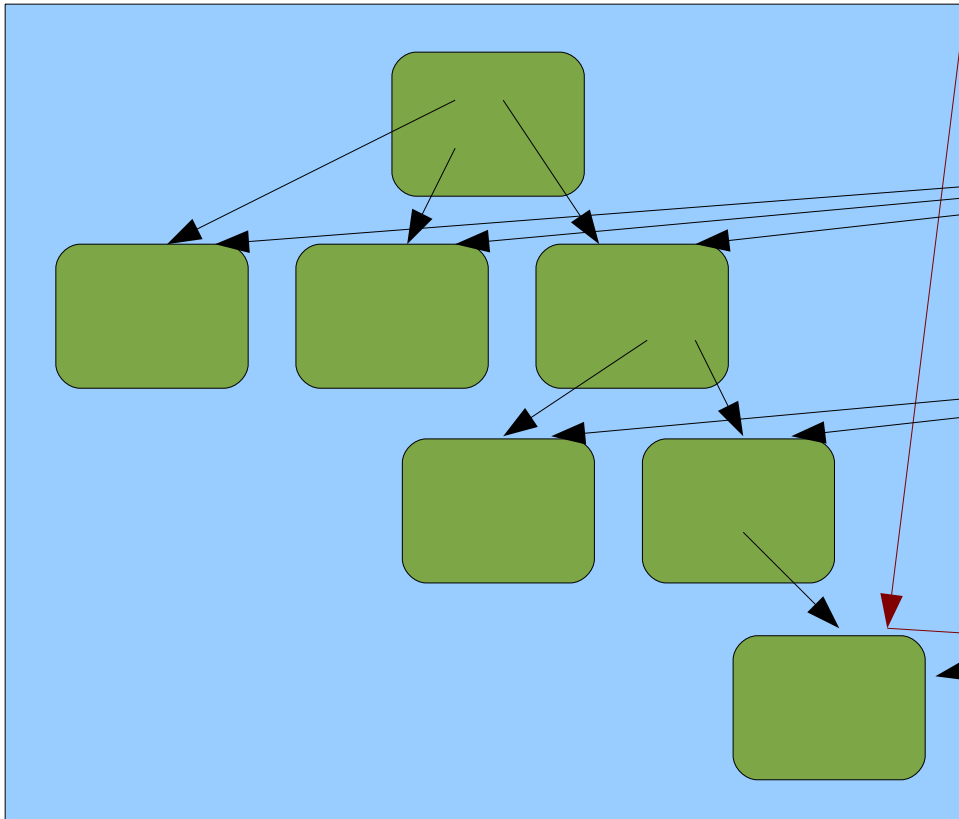
TREE WRITE COPY

User

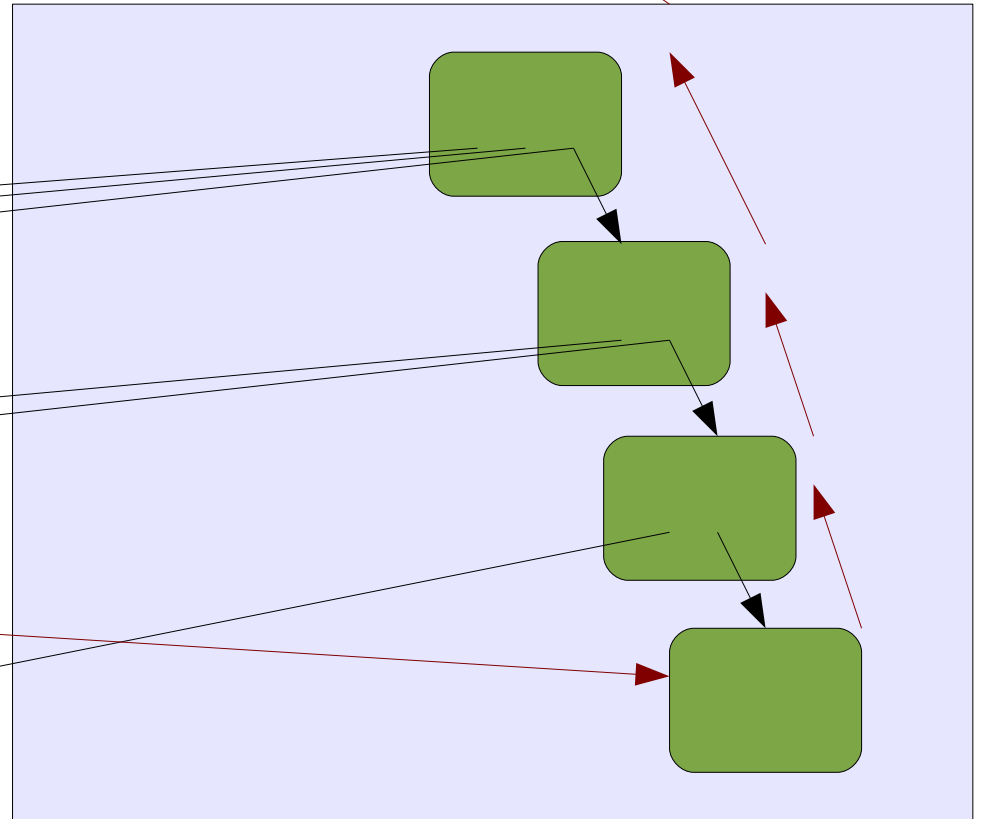


SNAPSHOT!!

Original



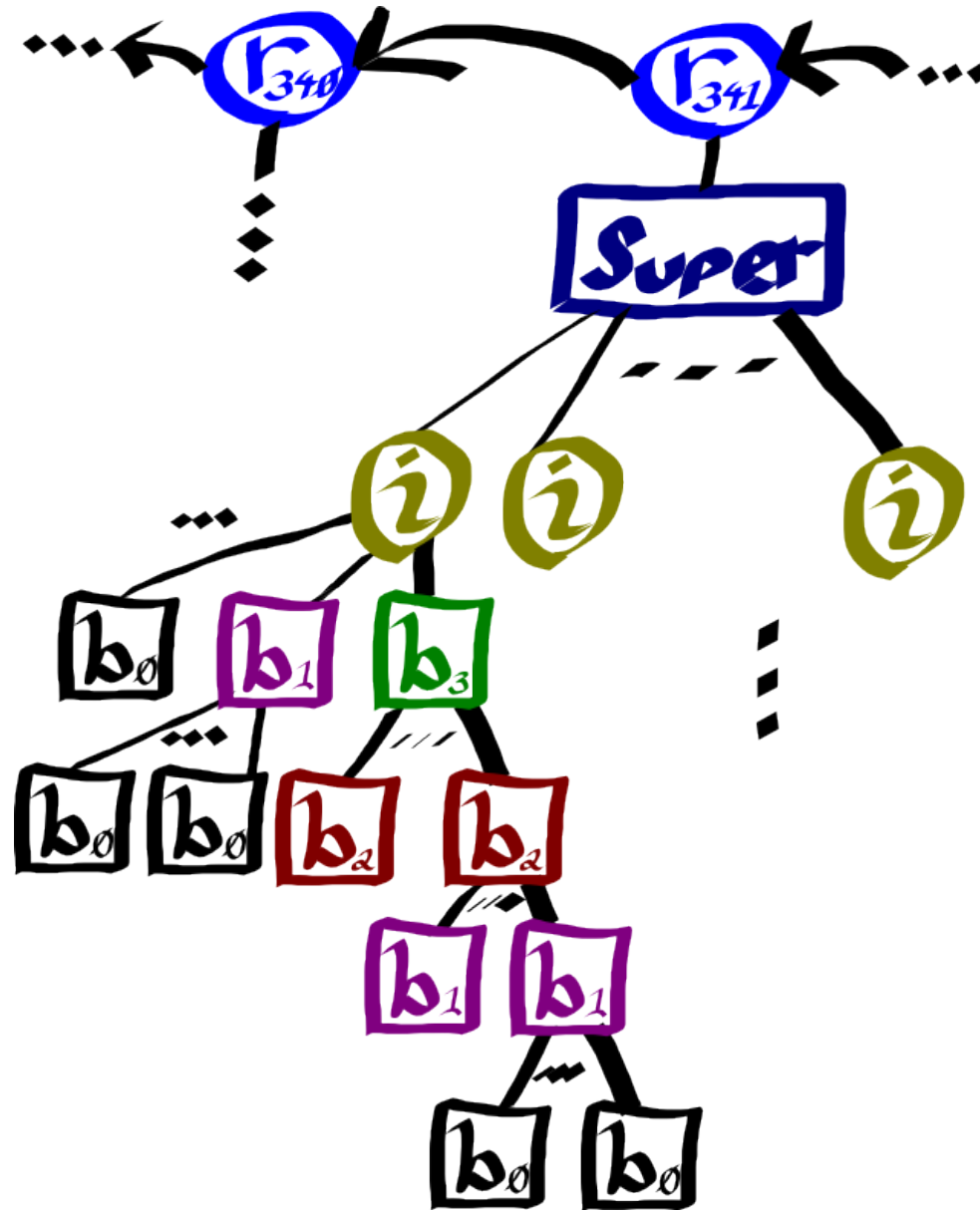
Copy



Ext3 (typical) Block Layout

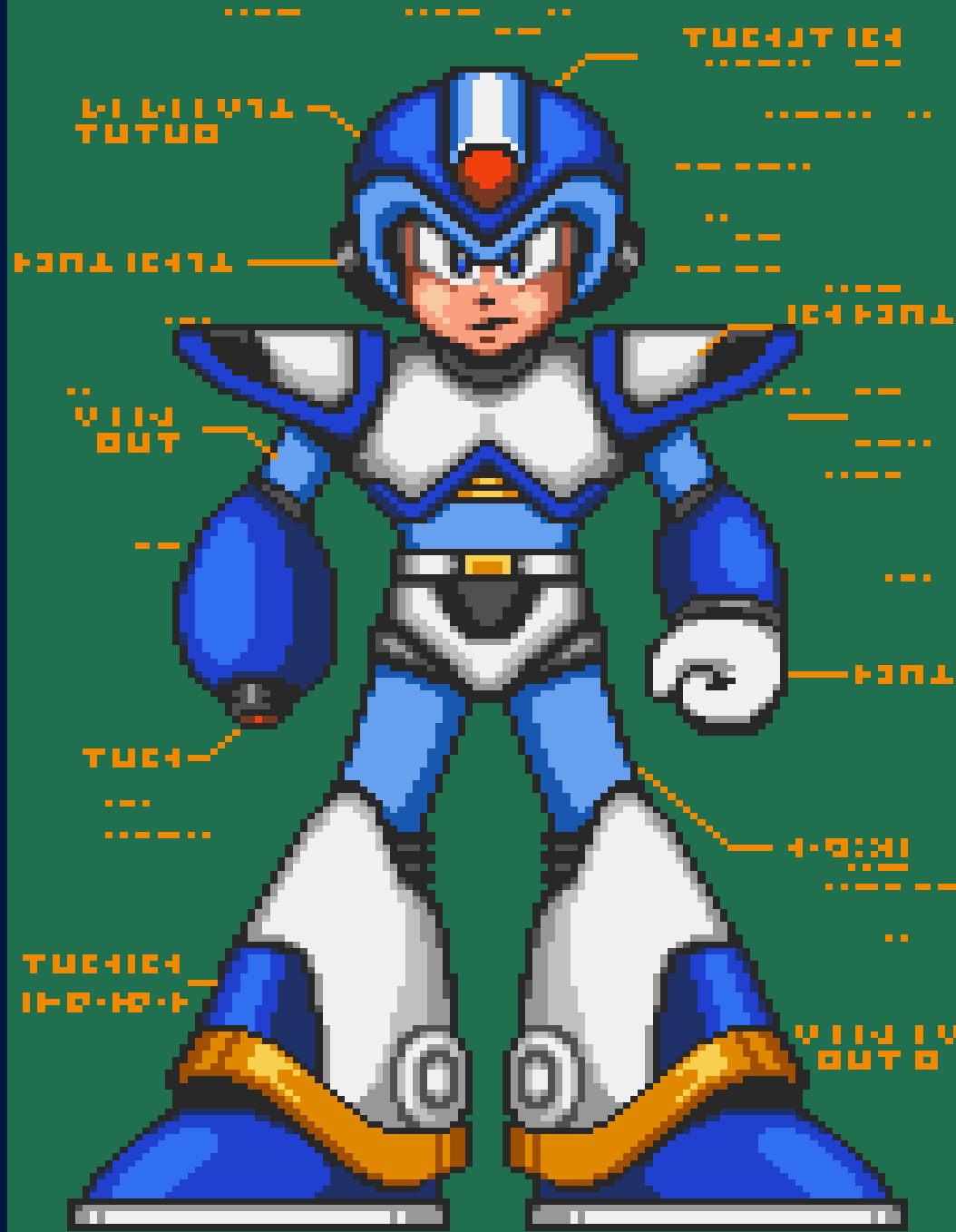
```
/*  
 * How many blocks doth make a writepage()?  
 *  
 * With N blocks per page, it may be:  
 * N data blocks  
 * 2 indirect block  
 * 2 dindirect  
 * 1 tindirect  
 * N+5 bitmap blocks (from the above)  
 * N+5 group descriptor summary blocks  
 * 1 inode block  
 * 1 superblock.  
 * 2 * EXT3_SINGLEDATA_TRANS_BLOCKS for the quote files  
 *  
 * 3 * (N + 5) + 2 + 2 * EXT3_SINGLEDATA_TRANS_BLOCKS
```

The File System *IS* a tree!



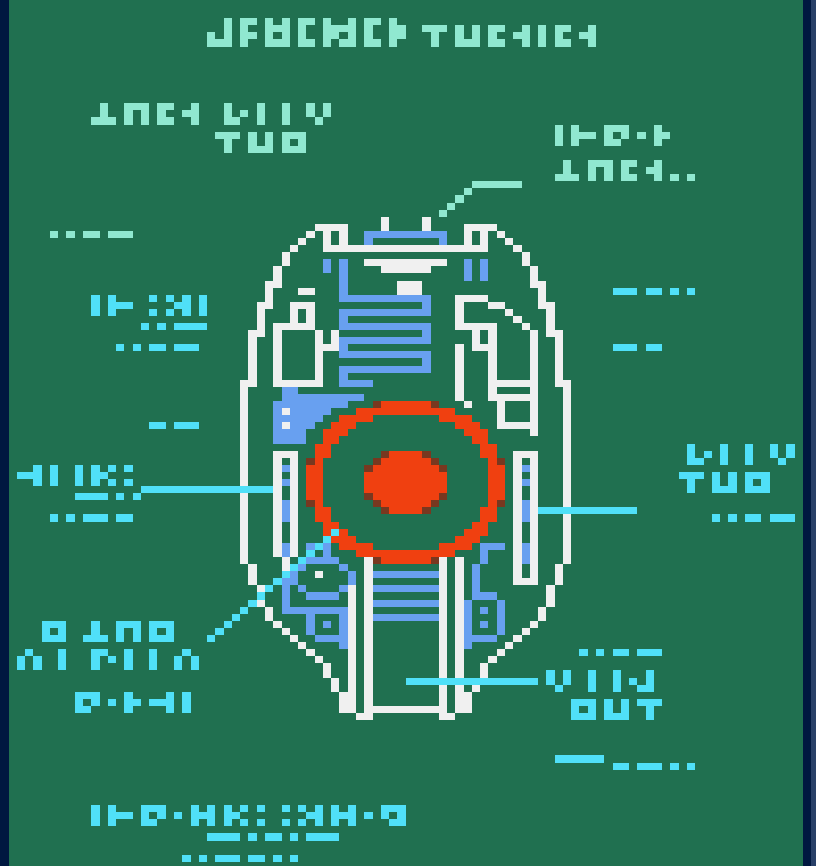
So what?

- We can now record the state of an entire file system with very little cost
- We can ensure survivability of losing a disk or two using RAID if necessary
- We have provided one reasonable solution to the backup problem



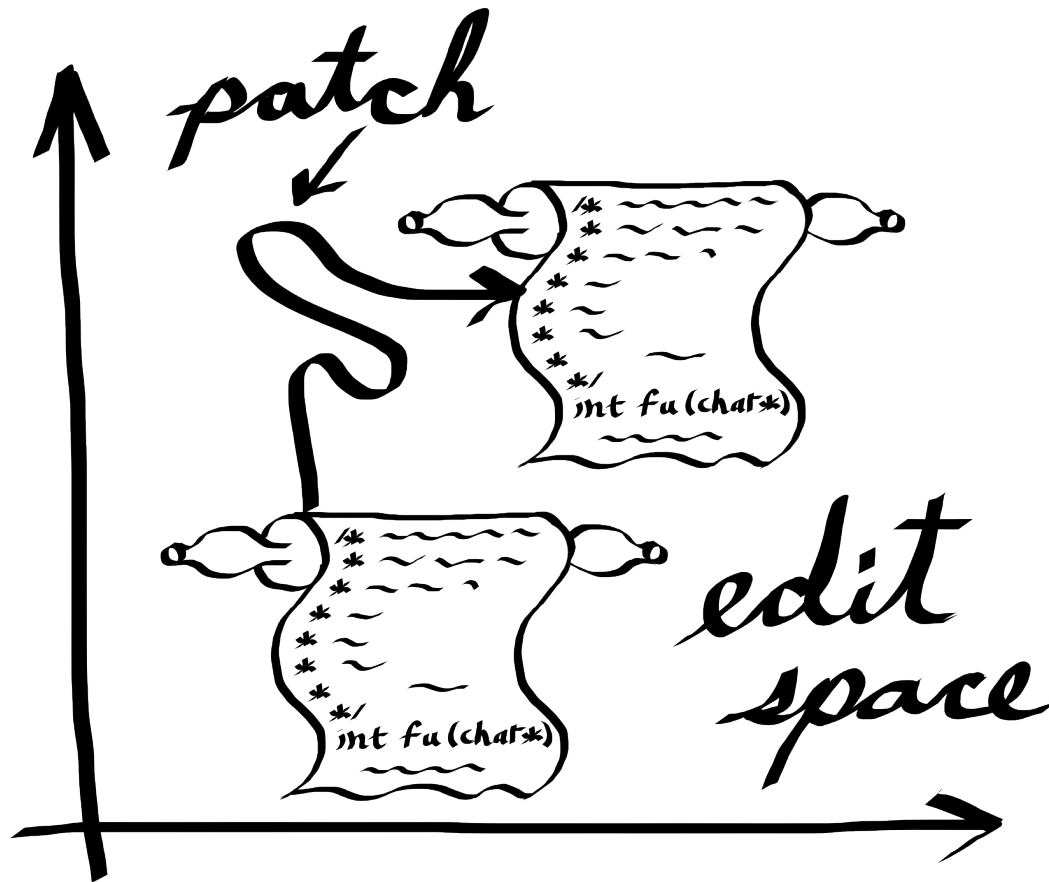
YOU GET
SNAP

ATTACK



(3) Patches

- What is the minimum amount of work needed to change one file into another?



Not Trivial

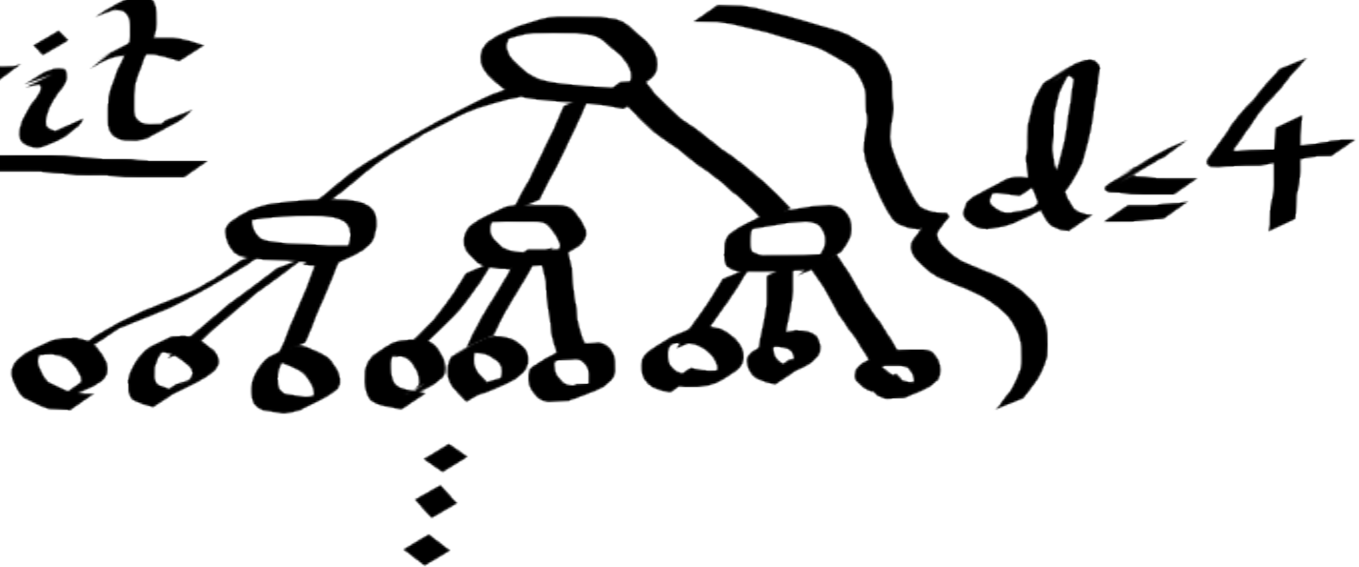
- Diffing algorithms are NP hard when diffing an arbitrary number of input files
- DP gives an $O(N^2)$ time & space solution
- Other algorithms minimize space/time by optimizing special cases (e.g., `diff`)
 - E. Meyers shows an $O(ND)$ time & space solution where D is the size of the diff, and N is the sum of the input strings

Patches and the LKML

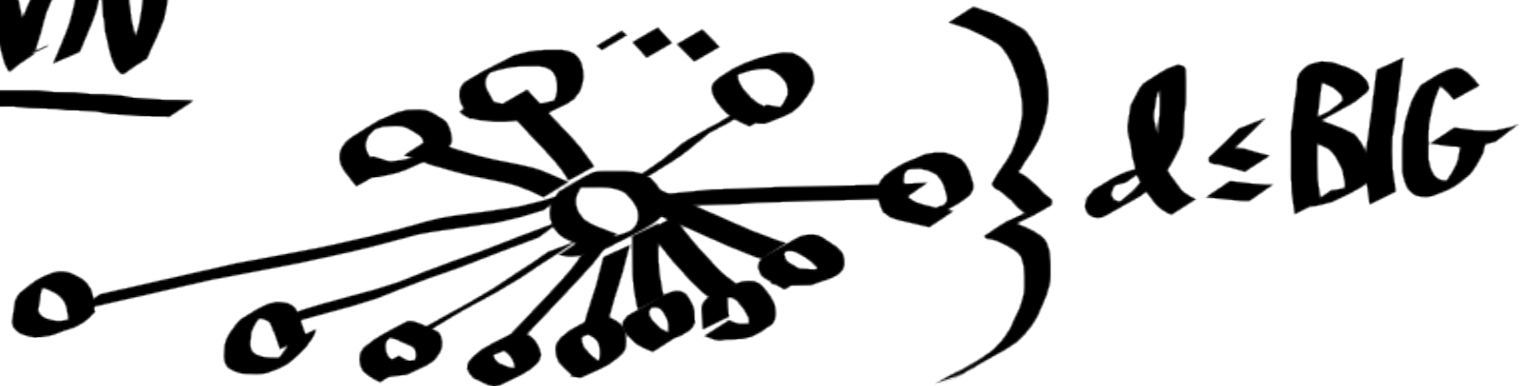
- Linux has used patches for its source control up until just two years ago (as of 2006!)
 - Survivability
 - Independence
 - Cost of hardware resources
 - Ease of use

The Patch Tree

Git



SVN

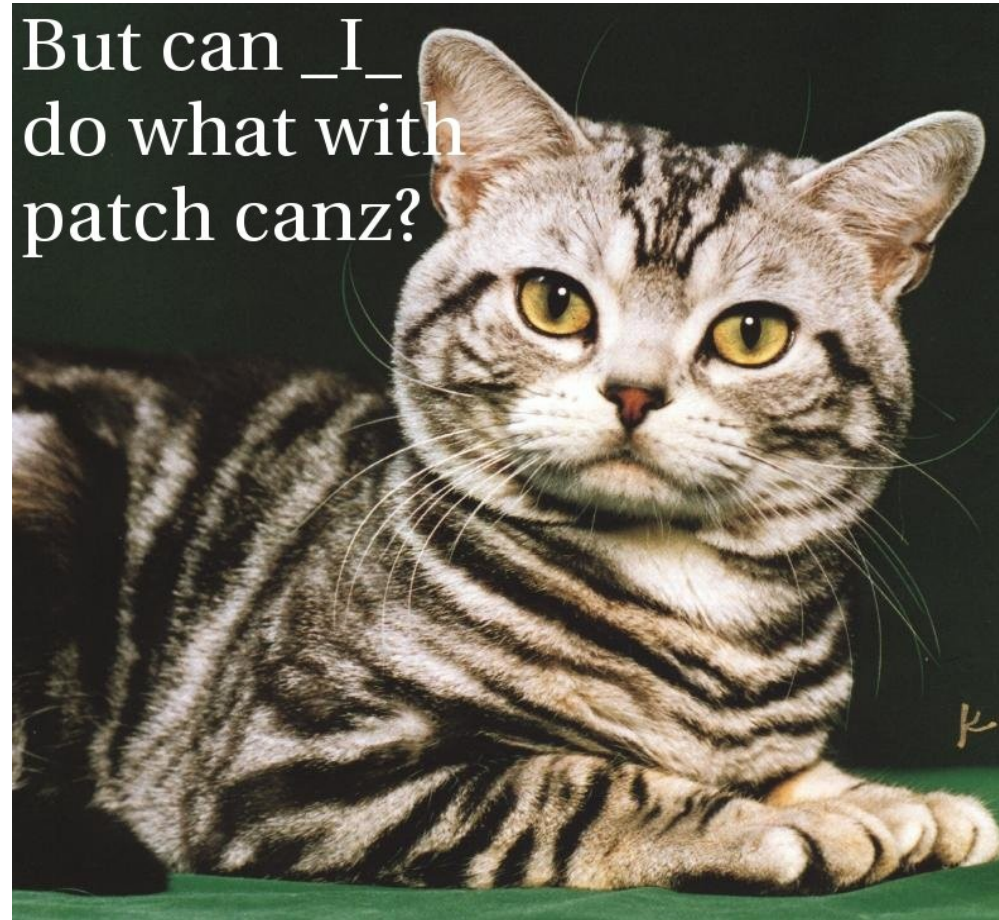


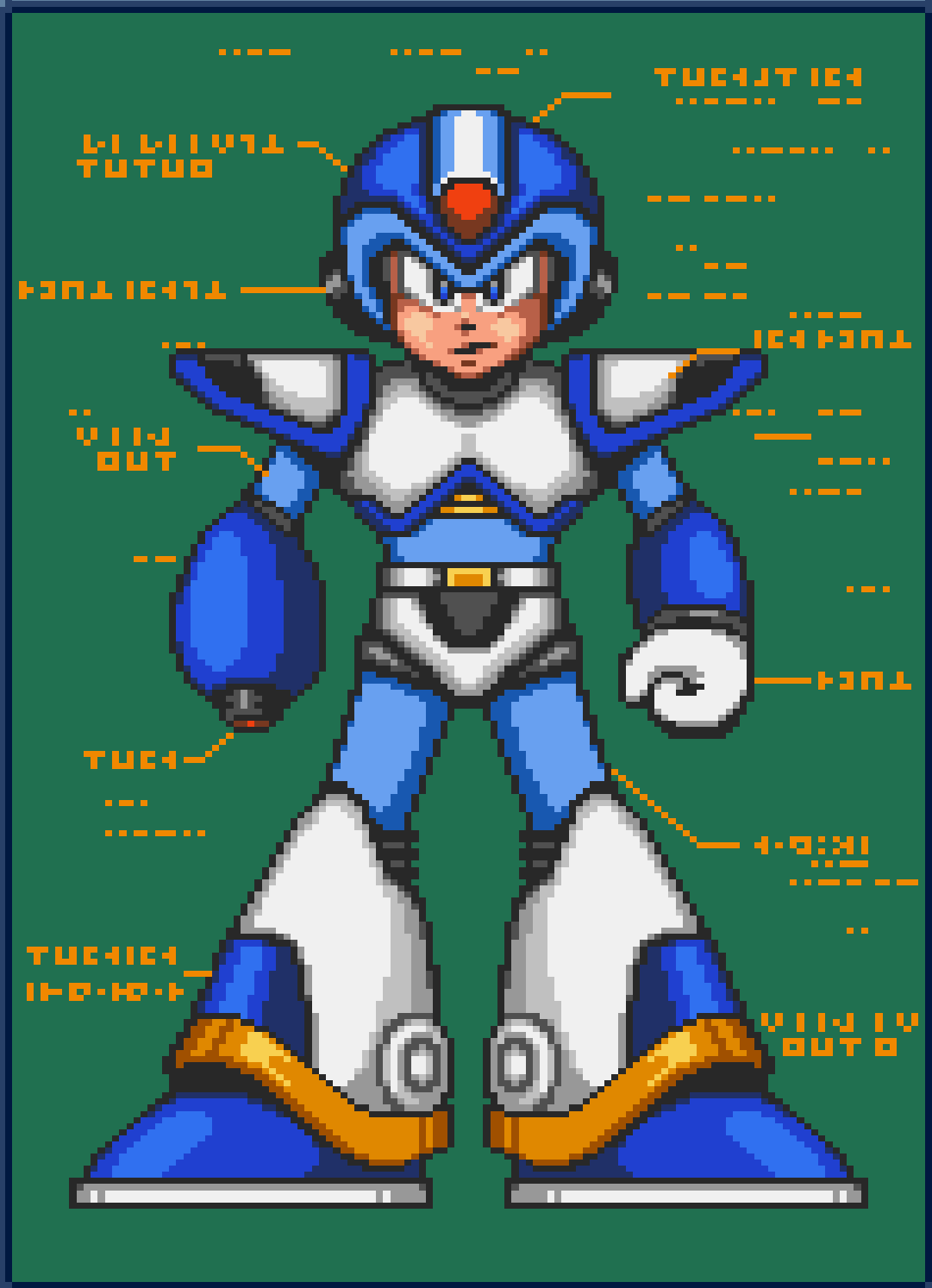
Pull vs. Push

- ***Push***: If you commit, you'll have to merge Bob's changes first. You're busy, or you don't know how to merge Bob's changes and must wait, going on a source control fast (very common)
- ***Pull***: Bob e-mails you that his changes are ready. You continue to commit to your local repo, then apply his patch when you are ready

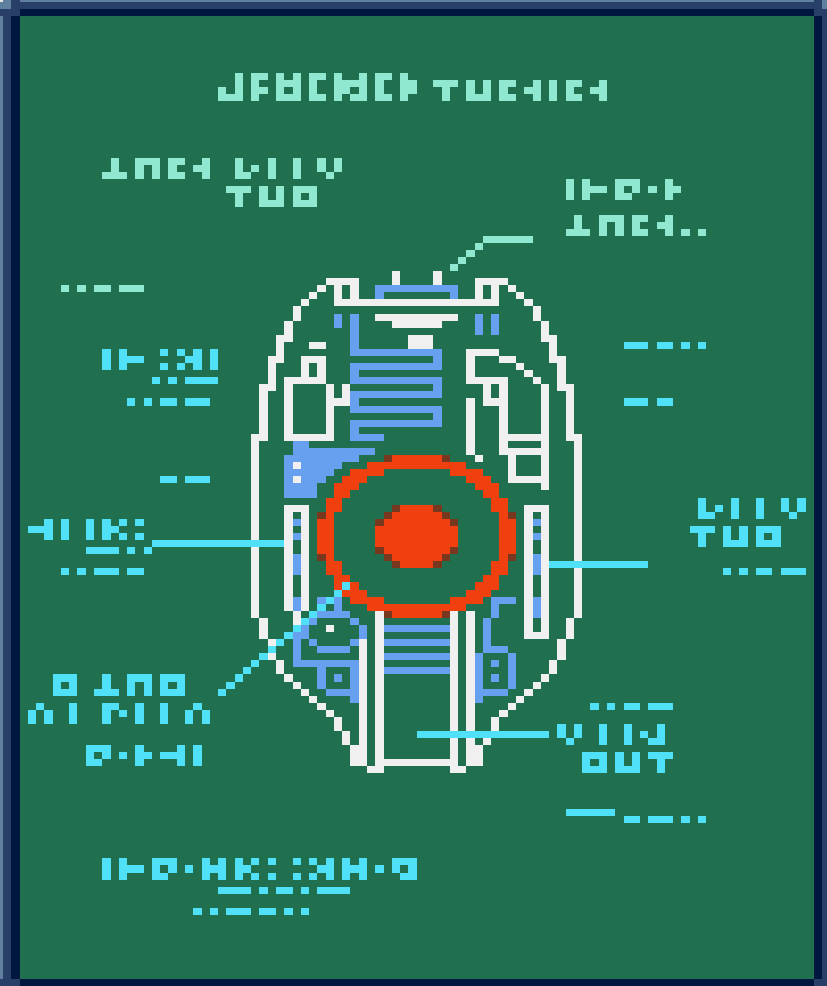
So what?

- Linux only needed to organize its current patch-based system
- Patches represent a state transition from a previous file state to a new file state





YOU GET
PATCH
CANNON



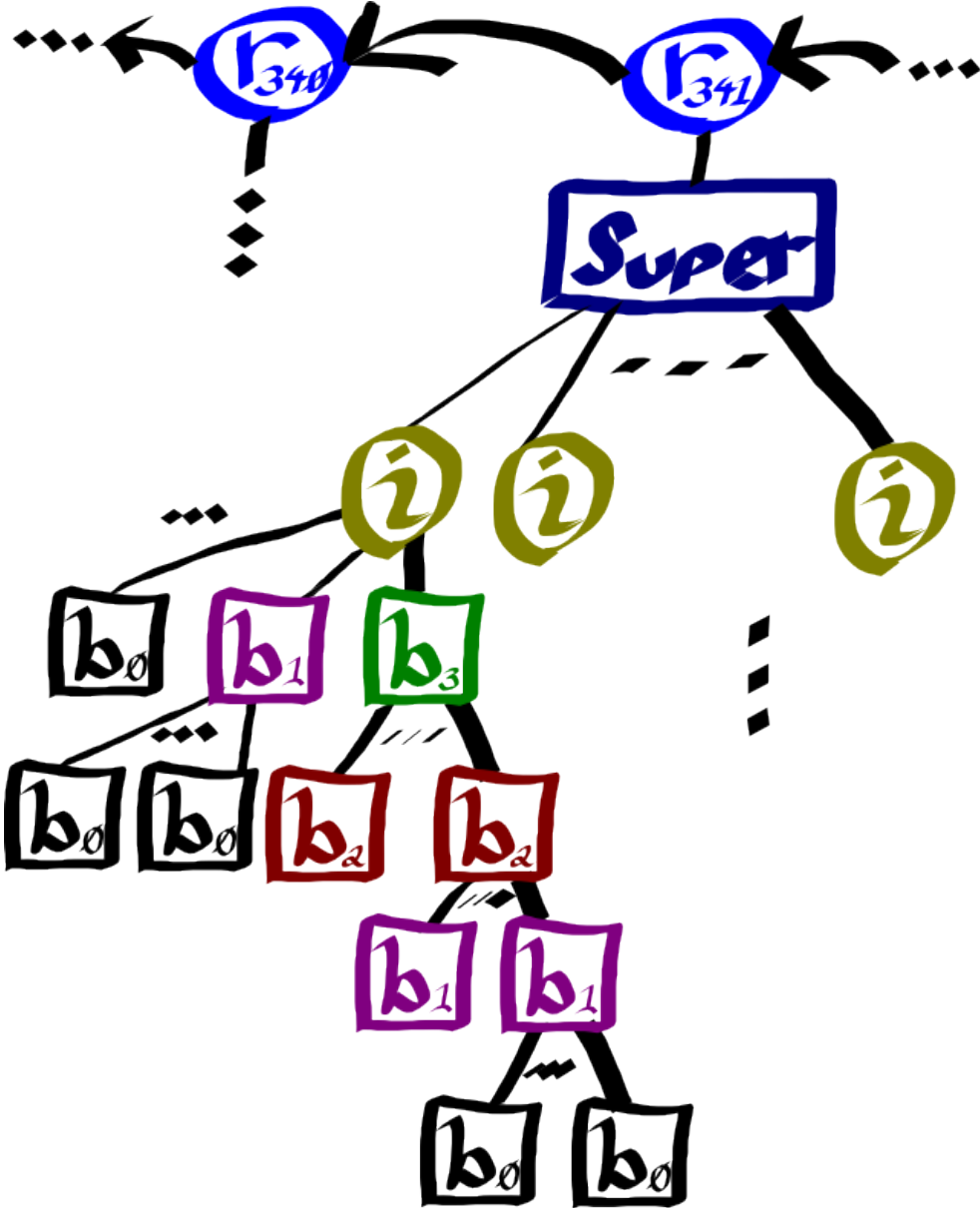
(4) Object Stores

- Object stores store objects
- Object \rightarrow {Object, Object, ...}
- Good at storing graphs
- Can we represent a revision history as a graph?

WAFL inspires

- If we want to record the state of the working tree at some point, just take a ***snapshot***
- WAFL teaches us how to use copy-on-write to construct a graph representing revision history...

WAFL doubles as a revision graph



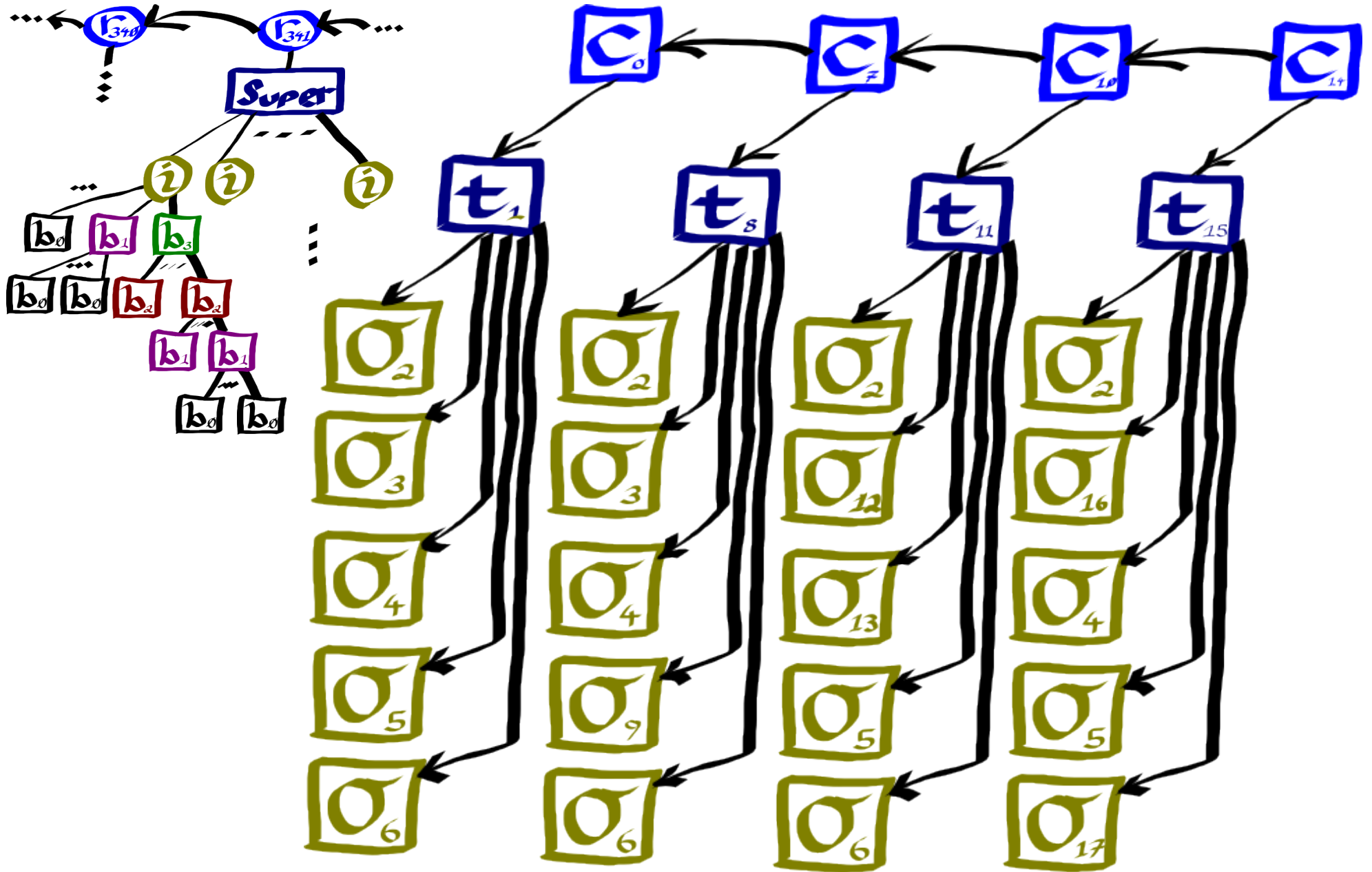
Problem...

- WAFL assumes knowledge of FS internals
- Git is a user level program, not a FS
- What to do?

...Solution

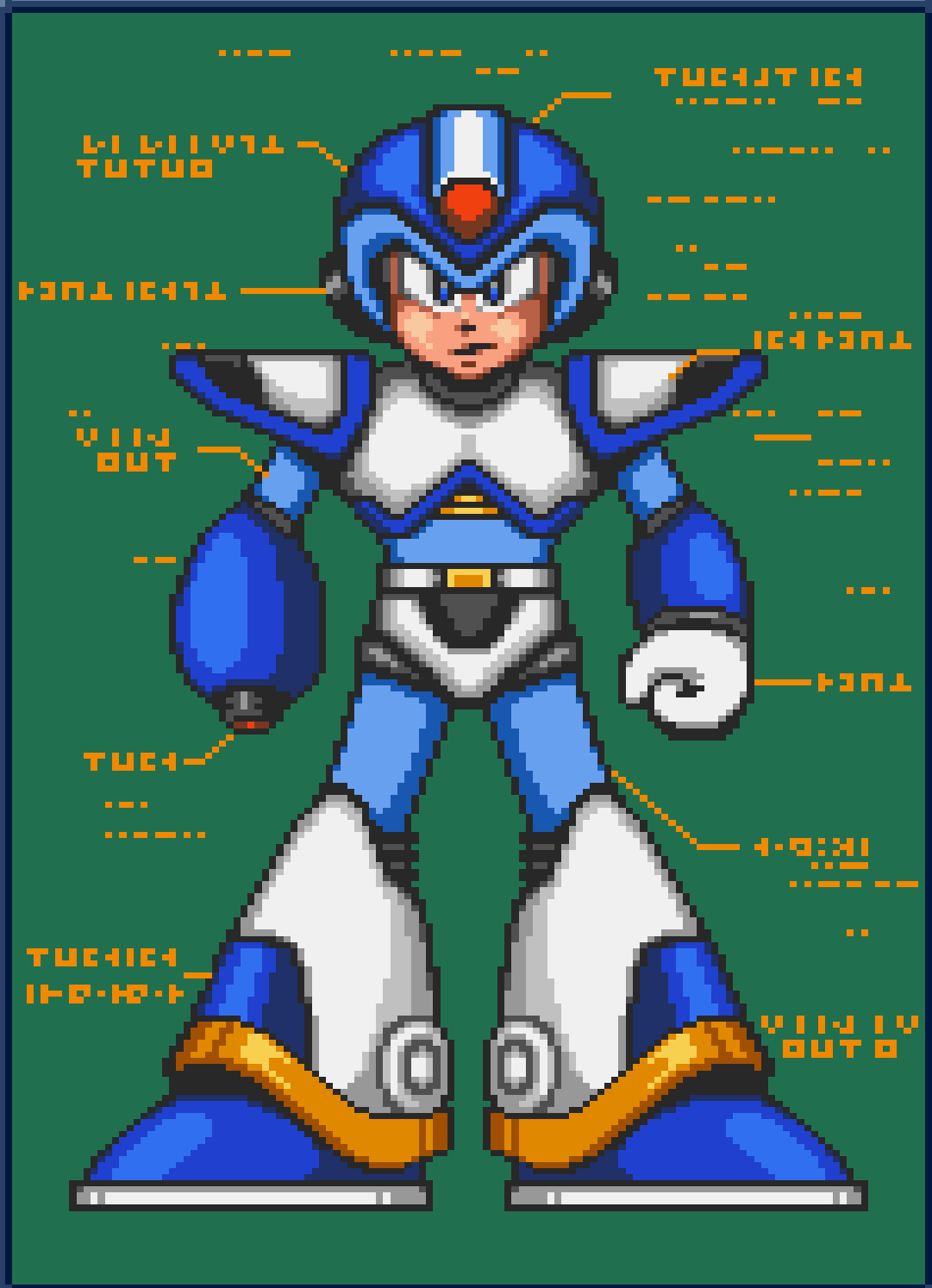
- We make three kinds of objects:
 - A ***tree*** object
 - A ***commit*** object
 - A ***file*** object
- Versions point to commits
- Commits point to trees
- Trees point to files

Sample git revision graph

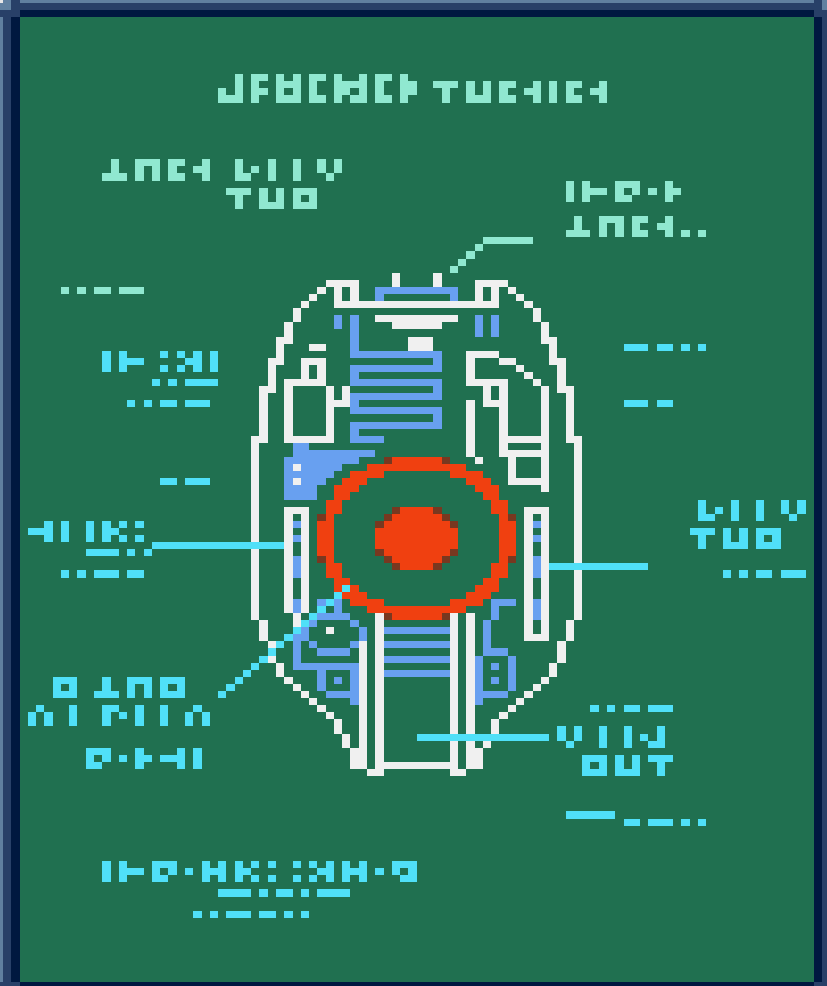


So what?

- We can encode a revision history based off of a series of snapshots in an object store
- We can encode branches and merges into a revision graph which in turn is stored in an object store



YOU GET
THE UNLINKER



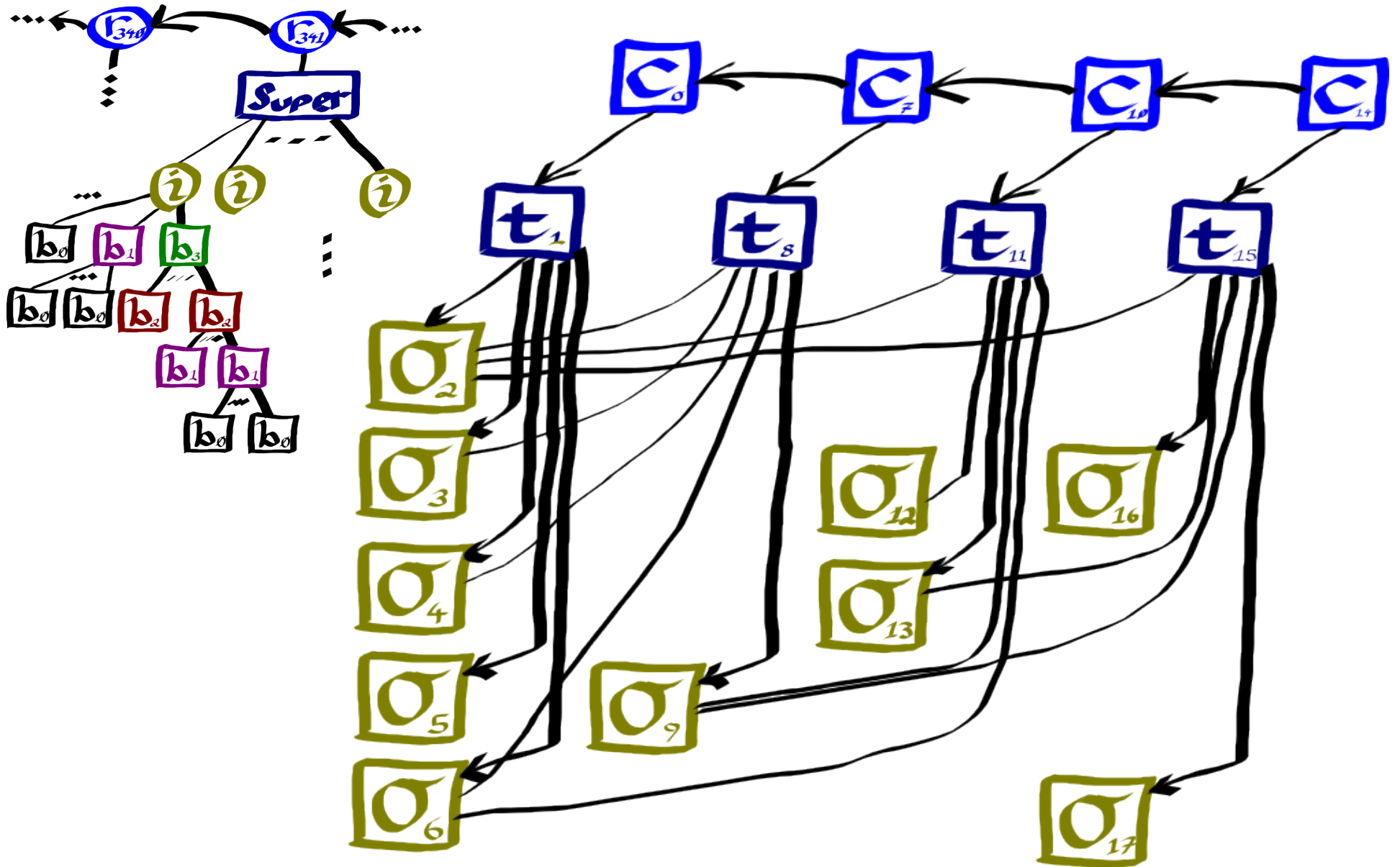
(=) Git

(-cvs)₁ +
snapshots₂ +
patches₃ +
object stores₄ = git

Supporting FS-Snapshot

- Use Case Scenario: Let user do 'snapshot'
 - Git commit -m "commit message"
- Problem: What about '*.o' files?

Sample git revision graph



Support FS-Snapshot

- Solution for '*.o' files:
 - Don't assume all child files should be in the tree
 - `git add <file>`, `git rm --cached <file>`, `git status`

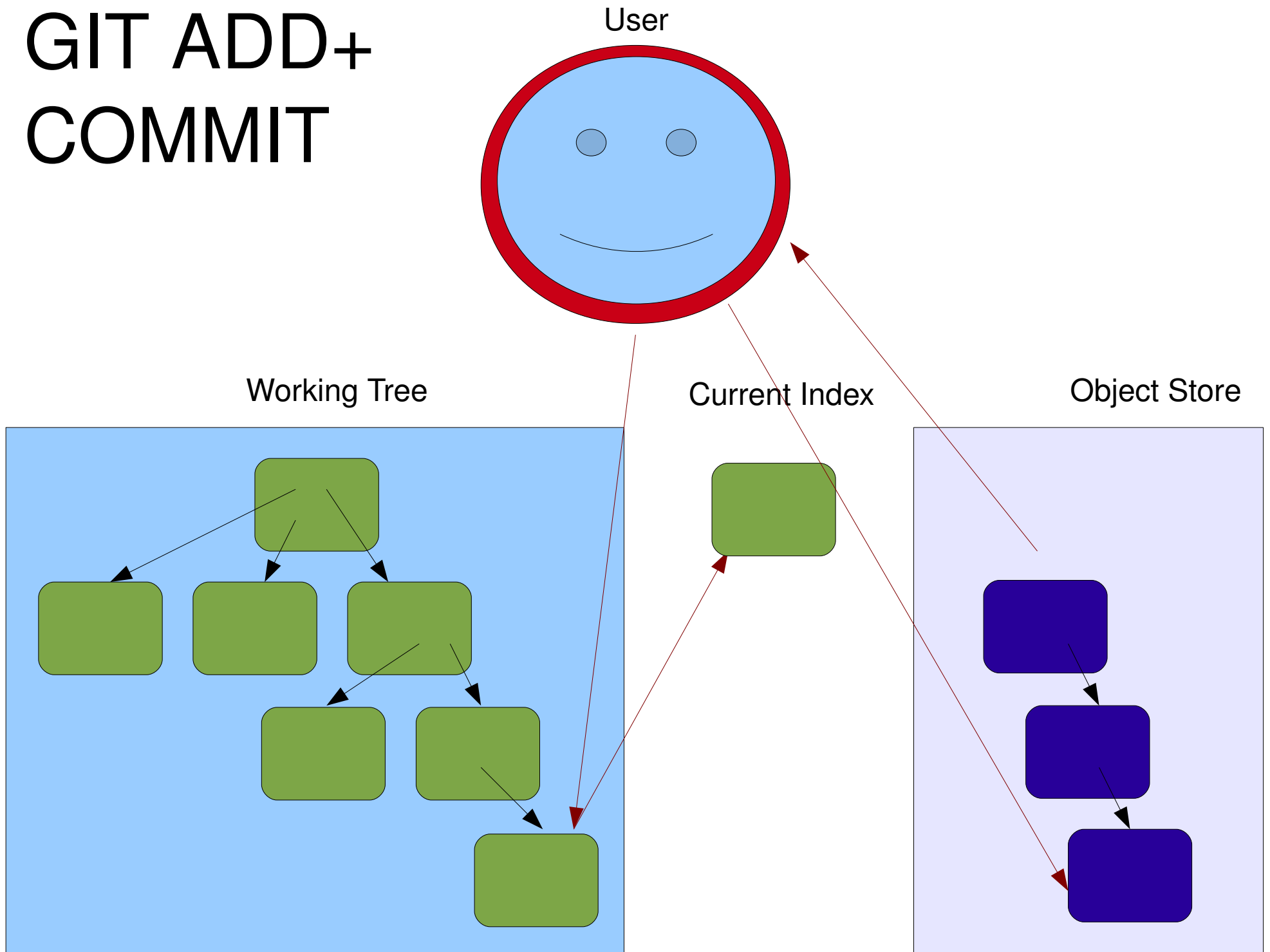
Support time-travel

- Go back in time (the point of keeping history)
 - Git checkout <object-db-id of old commit>

Designing git

- Maintain object database to store revision graph
- Maintain an index that represents the state of the FS which is modified by 'git add/rm – cached'

GIT ADD+ COMMIT



Look at the source code...

Conclusion

- git's superiority is not an accident
- Git is born from FS/systems research in
 - Snapshots
 - Patches
 - Object stores
- Practitioners (Linus) knowing how to make git represents the fruits of our labor!

