git
an introduction

How Communication Channels Shape and Challenge a Participatory Culture in Software Development

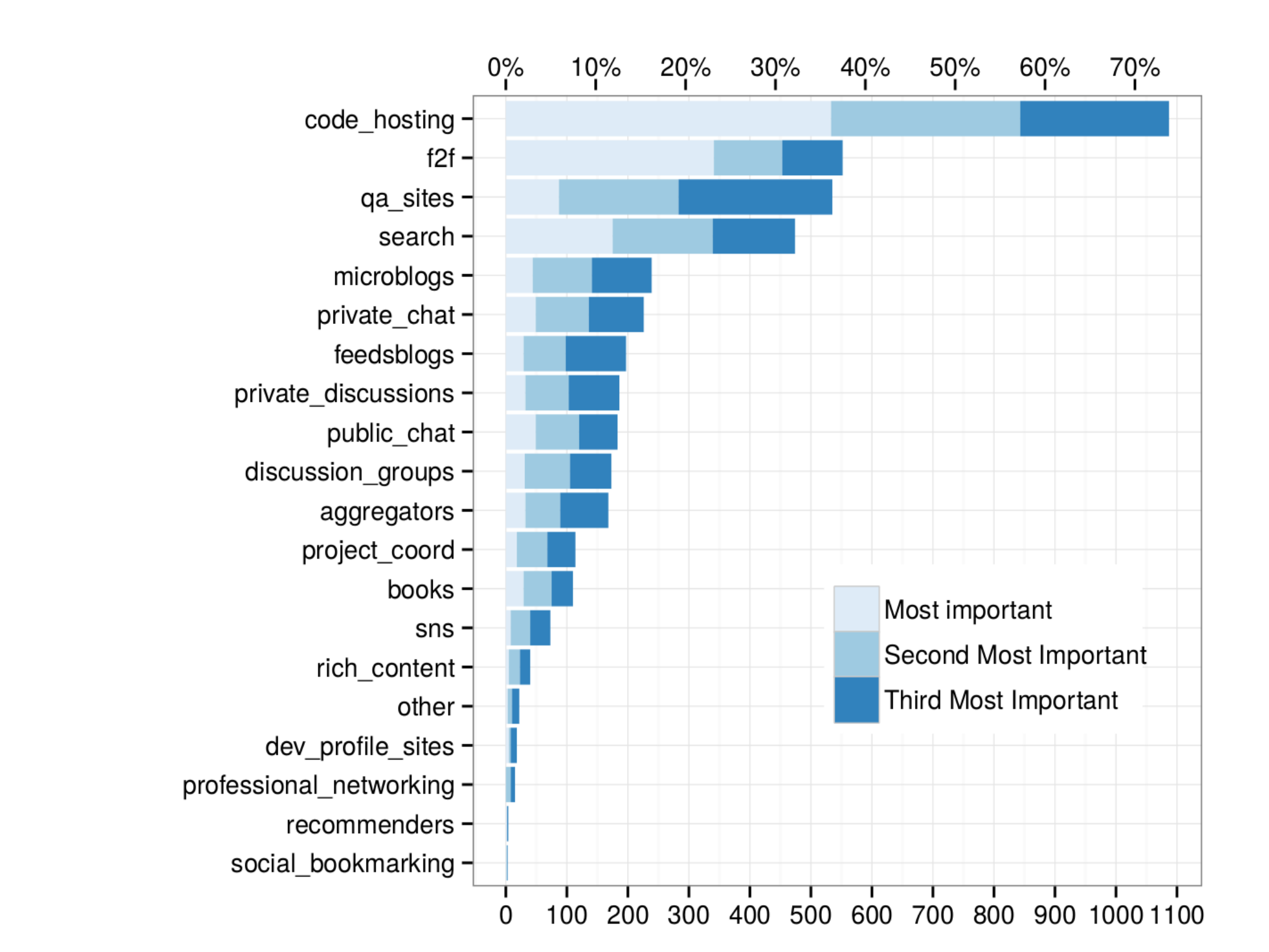
Margaret-Anne Storey, Leif Singer, Fernando Figueira Filho, Alexey Zagalsky, and Daniel M. German

Abstract—Software developers use many different communication tools and channels in their work. The diversity of these tools has dramatically increased over the past decade, giving rise to a wide range of socially-enabled communication channels and social media that developers use to support their activities. A participatory culture of software development is emerging in which developers want to engage with, learn from, and co-create software with other developers. However, the interplay of these channels, as well as the opportunities and challenges they may create when used together, are not yet well understood.

In this paper we report on a large-scale survey conducted with 1,448 GitHub users. We describe which channels these developers find essential to their work, and gain an understanding of the challenges they face using them. Our findings lay the empirical foundation for providing recommendations to developers and tool designers on how to use and improve tools for developers.

Index Terms—Participatory Culture, Communication, Social Media, CSCW, Software Engineering.

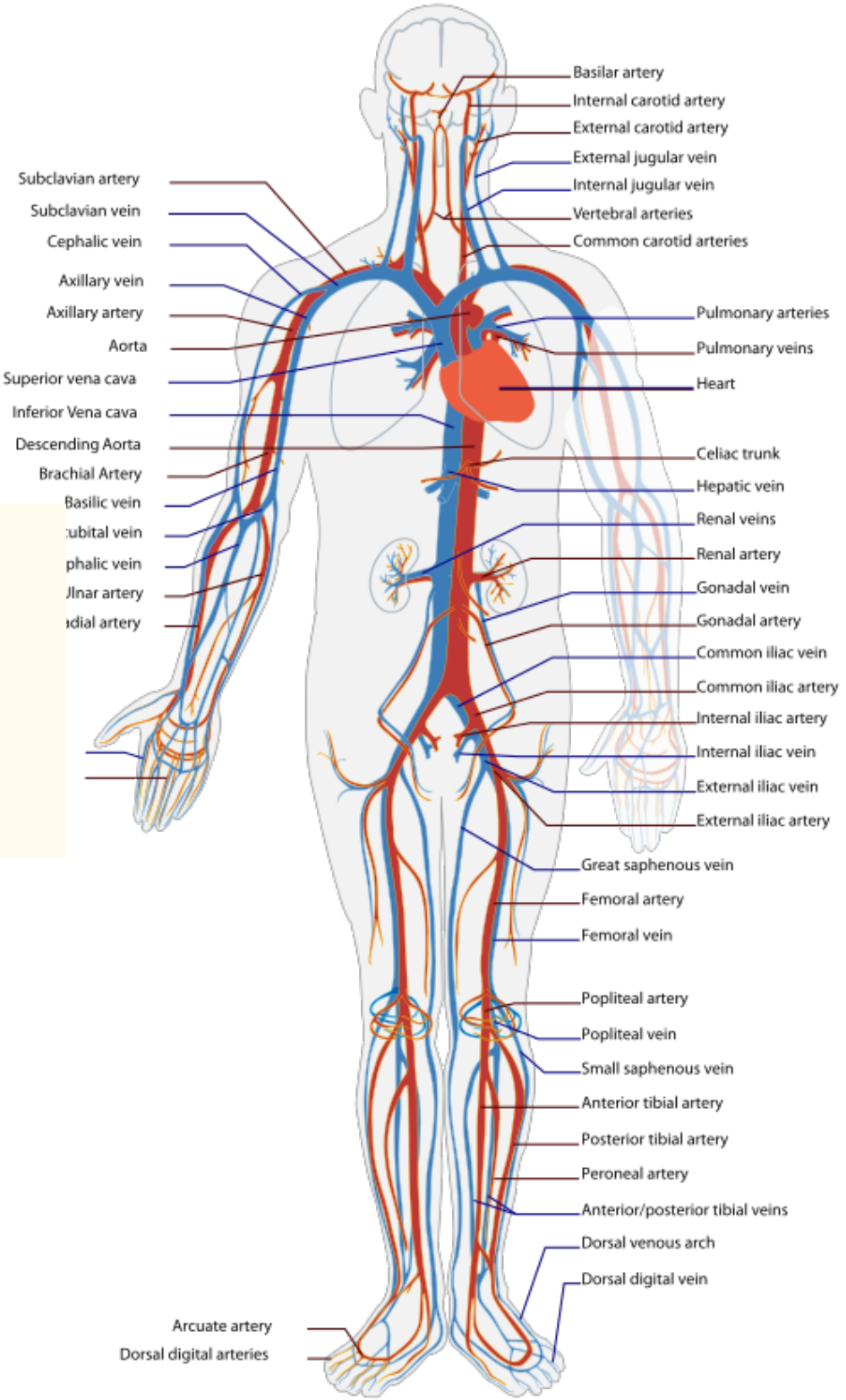




0%
10%
20%
30%
40%
50%
60%
70%
code_hosting
f2f
qa_sites
search
microblogs
private_chat
feedsblogs
private_discussions
public_chat
discussion_groups
aggregators
project_coord
books
sns
rich_content
other
dev_profile_sites
professional_networking
recommenders
social_bookmarking
Most important
Second Most Important
Third Most Important
0
100
200
300
400
500
600
700
800
900
1000
1100

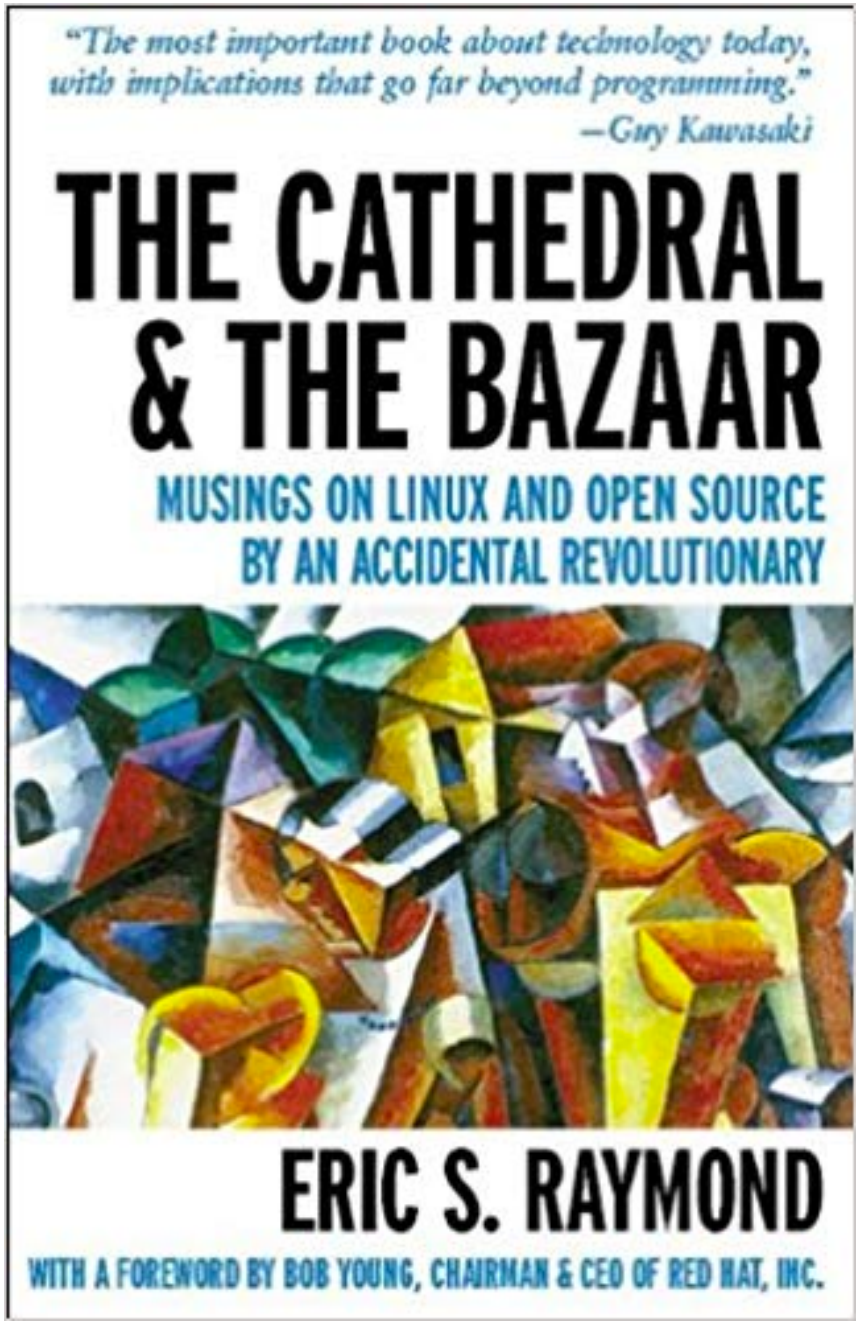
How is distributed version control affecting software development?

VERSION CONTROL

a developer's best friend



1997


"The most important book about technology today, with implications that go far beyond programming."
—Guy Kawasaki
THE CATHEDRAL & THE BAZAAR
MUSINGS ON LINUX AND OPEN SOURCE BY AN ACCIDENTAL REVOLUTIONARY
ERIC S. RAYMOND
WITH A FOREWORD BY BOB YOUNG, CHAIRMAN & CEO OF RED HAT, INC.

The *Bazaar* model

Eric S Raymond credits Linus Torvalds as the inventor of the Bazaar model:

> “the Linux community seemed to resemble a great babbling bazaar of differing agendas and approaches out of which a coherent and stable system could seemingly emerge"

Surprisingly, Linux **was not using version control** software at the time

Linus/Linux has an interesting relationship with version control

“Tarballs and patches is a much superior version control system than CVS is”

“I see Subversion as the most pointless project ever started”

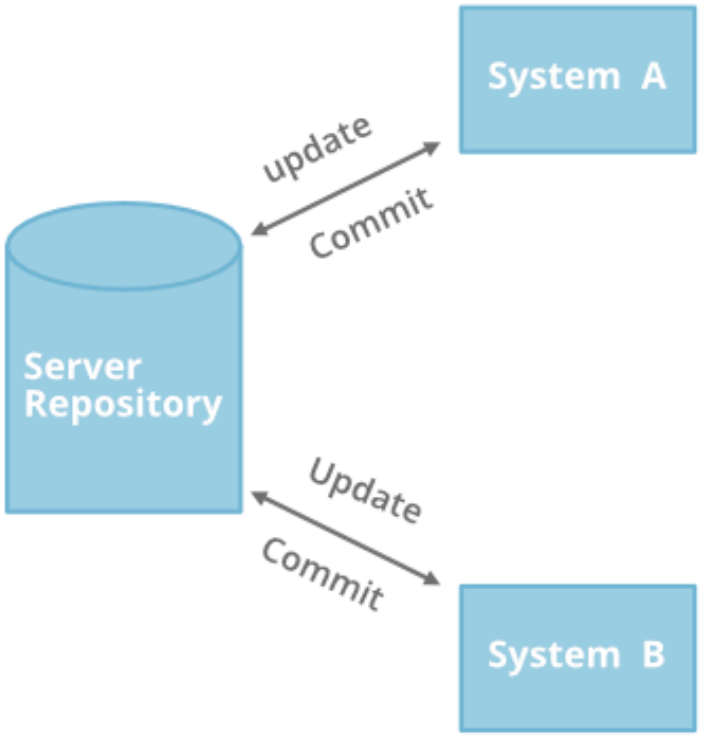
Centralized Version Control

- Centralized version control relies on a central, publicly accessible repository
  - Any developer who wants to participate, needs to have a user account

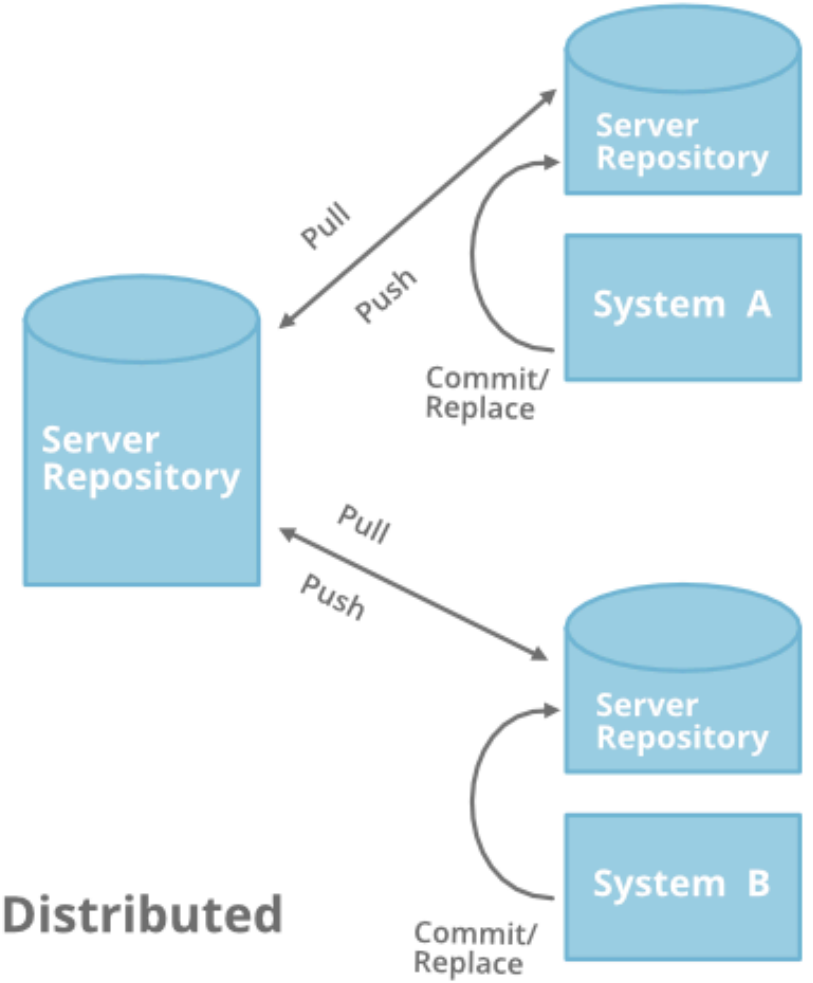
It means that before you start contributing, somebody needs to trust you

Distributed Version Control

- There are multiple repositories, all equally important
  - No need to “**register**” to contribute
  - Easy to start creating a feature
    - Relies on effective merging mechanisms



Centralized



Distributed

Implicit vs Explicit Branching

- In C-VCs
  - One must create a branch explicitly to work independently
  - Needs continuous access to the central repo for operations
- In D-VCs
  - Any repo is a branch of another repo
  - All operations are local

“[source control management] was uninteresting and yet profoundly important.”

“I hated all SCMs with a passion”

‘The **big thing** about **distributed** source control is that it makes one of the main issues with SCM’s go away – the politics around “**who can make changes.**”’

"So git was basically designed and written for **my** requirements, and it shows.”

“I’m just happy that it made it so **easy** to **start a new project.** Project hosting used to be painful, and with git and GitHub it’s just so trivial to do a random small project.”

Bitkeeper

- Linus adopted Bitkeeper as its first version control system in 2002
  - It was not open source
  - Used until 2005, when the license is revoked
  - This triggered the development of **git**
  - Bitkeeper influenced the development of git

Git

- Linus design goals:
  - “What would CVS would never do?"
  - Support distributed bitkeeper-style workflow
  - Protect against corruption and data loss
  - Be fast

Collaboration

- Isolation
  - Every developer commits without disturbing others
  - You do not have to trust anybody else
  - **Everybody** has their **own branch**
  - **Everybody has commit access**

Concurrent development!

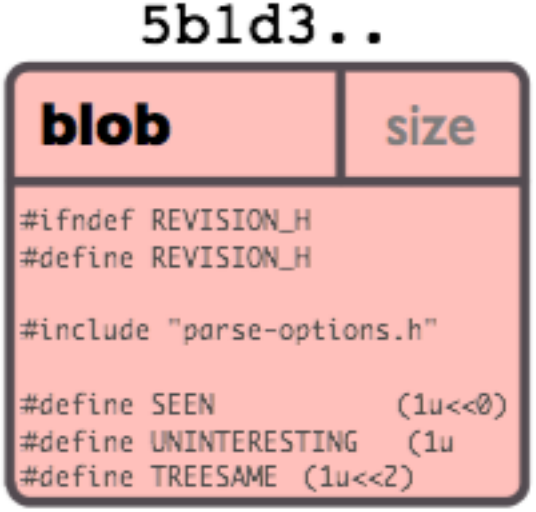
But git need at least one public repo

- In order to collaborate
- In 2008, GitHub is created to fill this gap

git Internals and data model

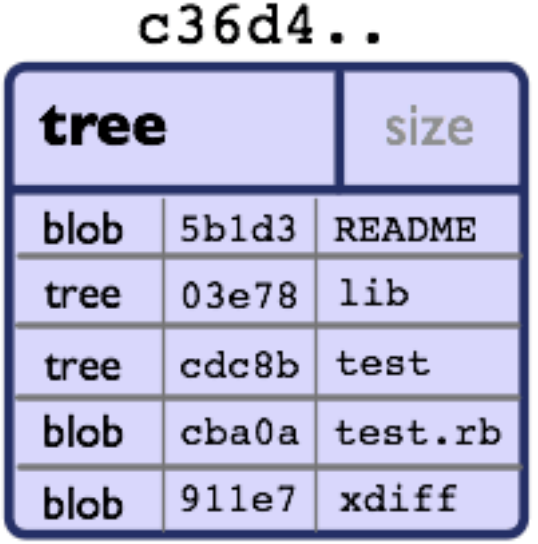
The blob

- At its core, git is a storage system
- The blob is its basic unit of storage
- Every blob is addressable by its id (a hash)
- The simplest blob is a file



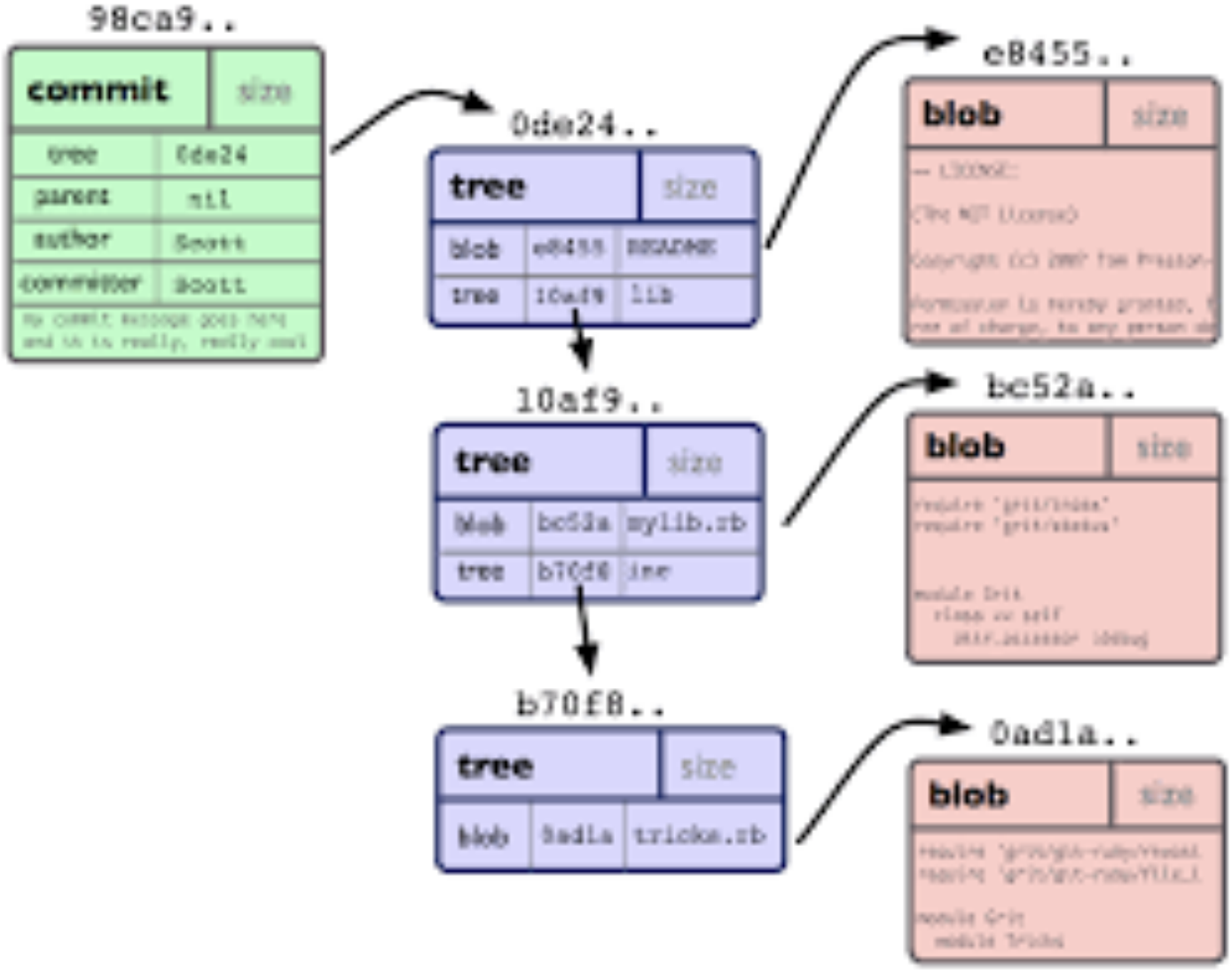
The tree

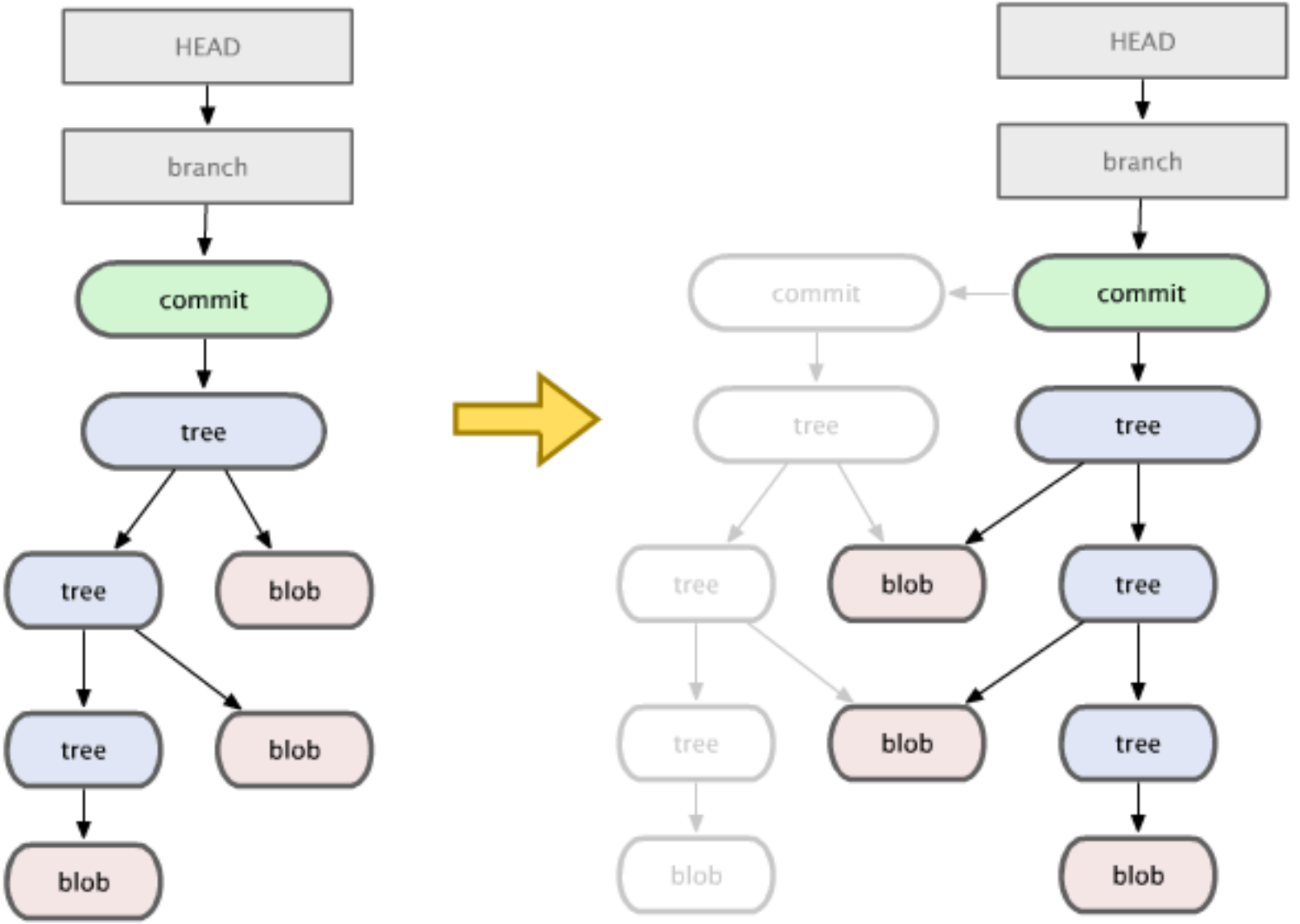
- A directory/folder is stored as a list of blobs
  - Which can be other directories



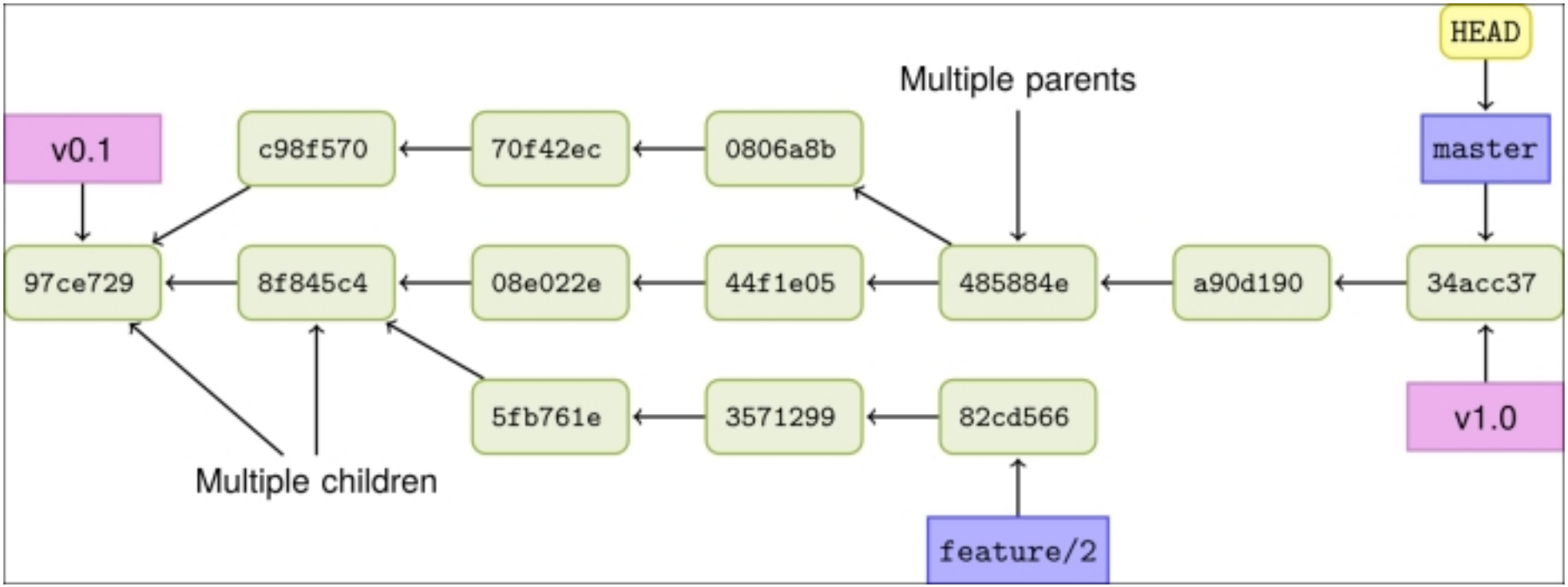
The commit

- Store:
  - the current tree (and its blobs)
  - The parent(s) commit(s)
- Along with some metadata
  - Author/Date
  - Committer/Date
  - Message


98ca9..
commit
size
tree
0de24
parent
nil
author
Scott
committer
scott
0de24..
tree
size
blob
e8455
README
tree
10af9
lib
e8455..
blob
size
10af9..
tree
size
blob
bc52a
mylib.rb
tree
b70f8
inc
bc52a..
blob
size
b70f8..
tree
size
blob
0ad1a
tricks.rb
0ad1a..
blob
size


HEAD
branch
commit
tree
tree
blob
tree
blob
blob
HEAD
branch
commit
commit
tree
tree
tree
blob
tree
tree
blob
tree
blob
blob

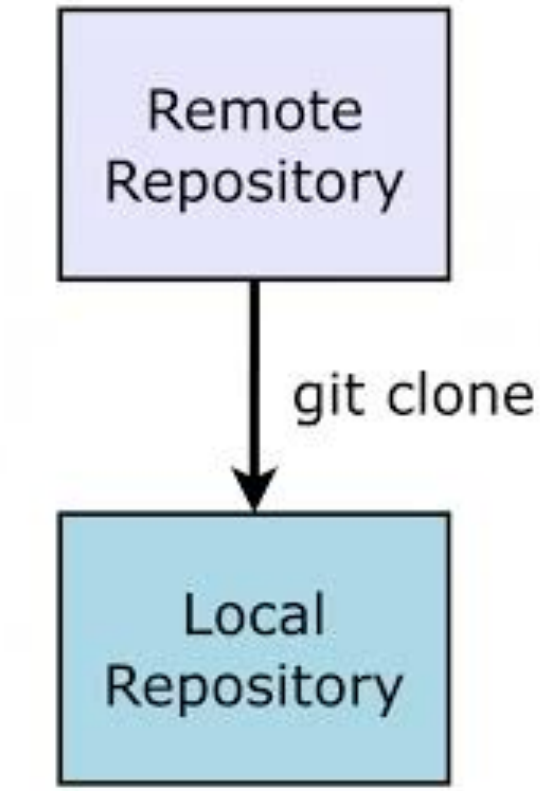
The DAG

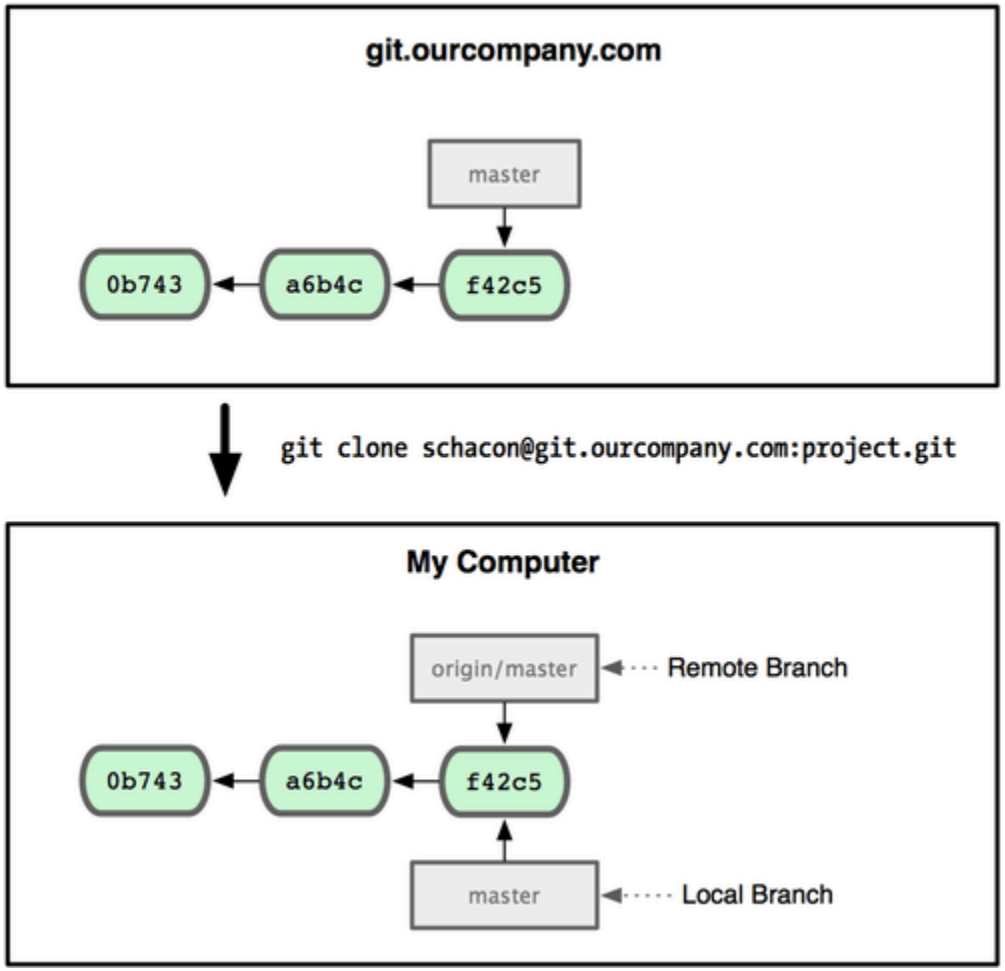


Tags identify branches and specific commits

The operations

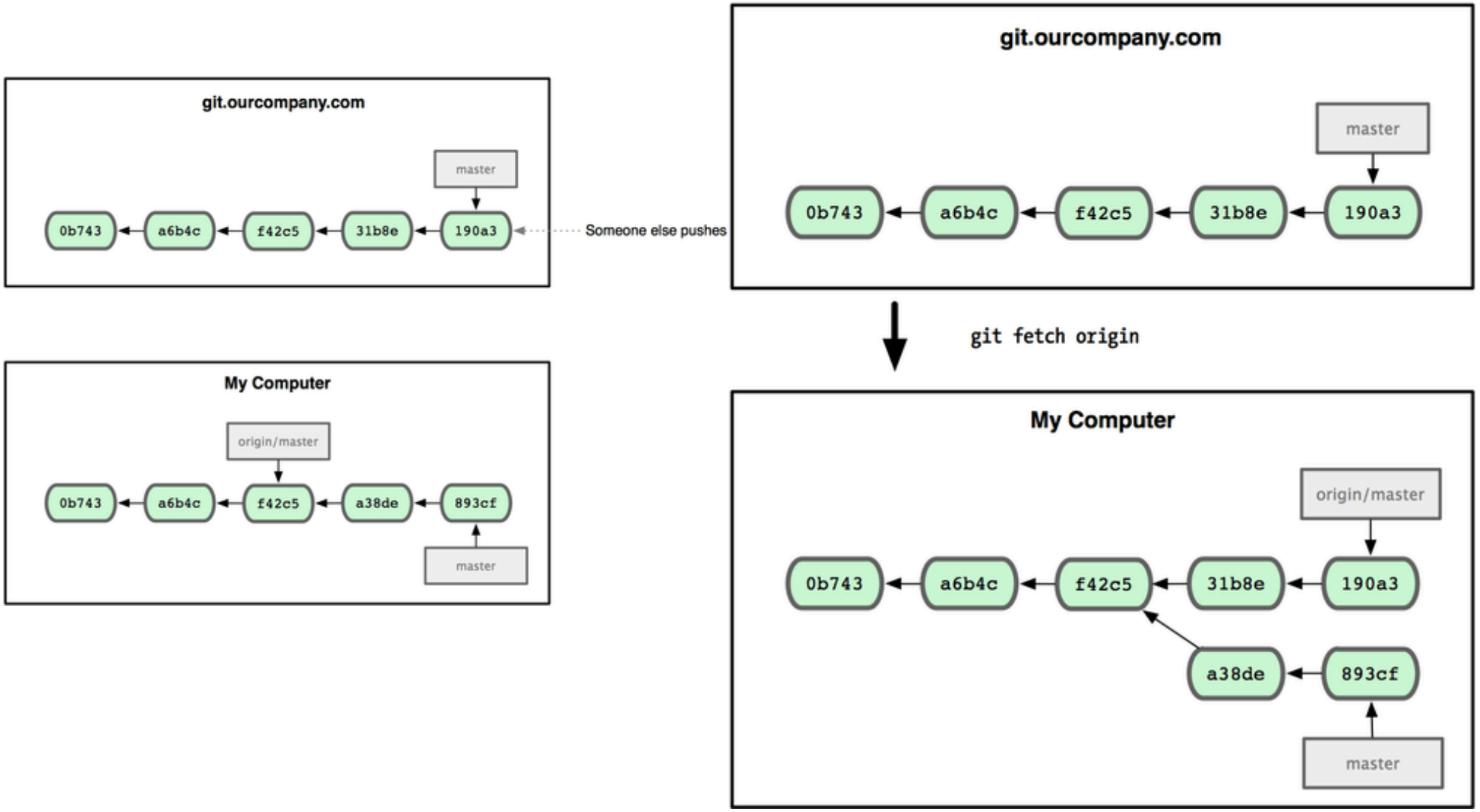
clone



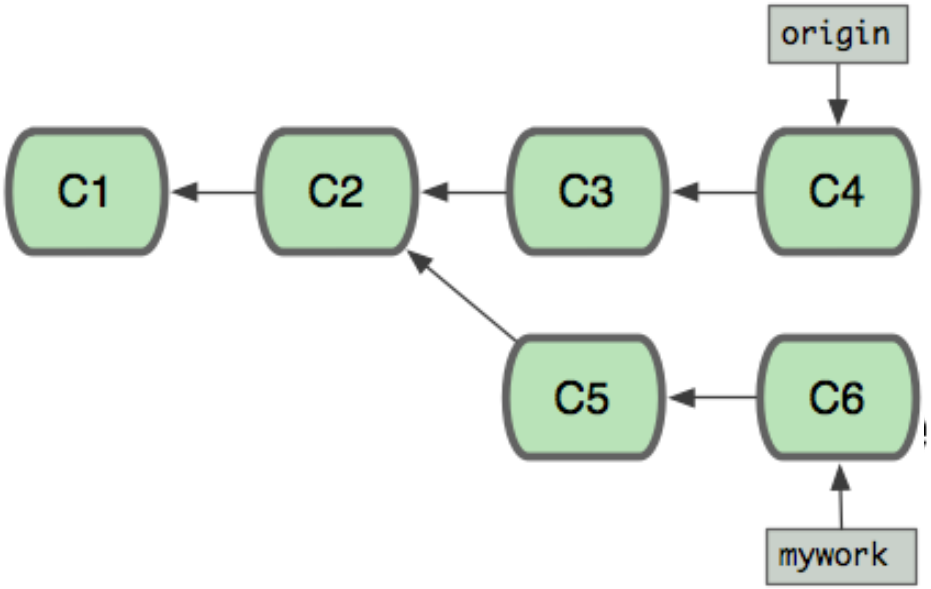


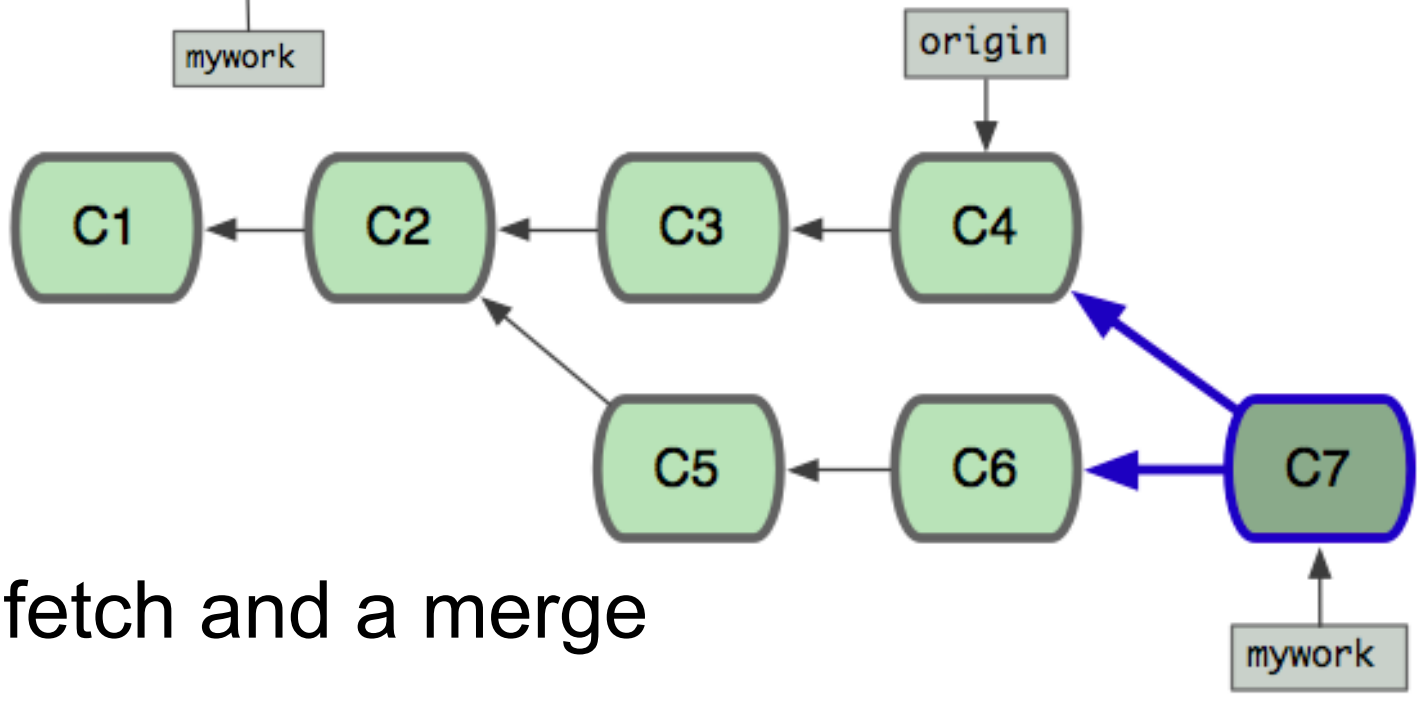
(git) clone is different from a (github) fork

Fetch


git.ourcompany.com
master
0b743
a6b4c
f42c5
31b8e
190a3
Someone else pushes
git.ourcompany.com
master
0b743
a6b4c
f42c5
31b8e
190a3
git fetch origin
My Computer
origin/master
0b743
a6b4c
f42c5
a38de
893cf
master
My Computer
origin/master
0b743
a6b4c
f42c5
31b8e
190a3
a38de
893cf
master

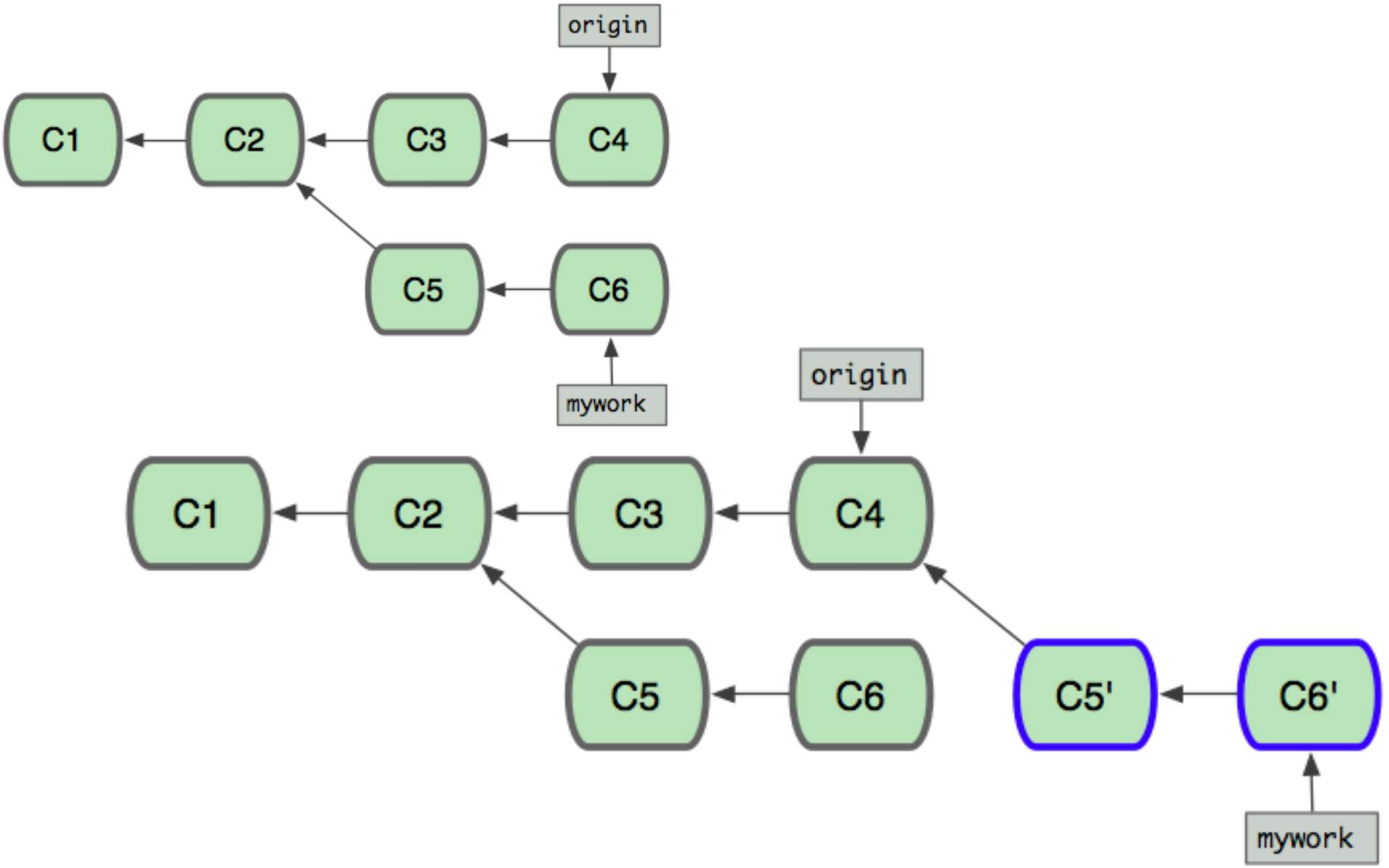
Merge



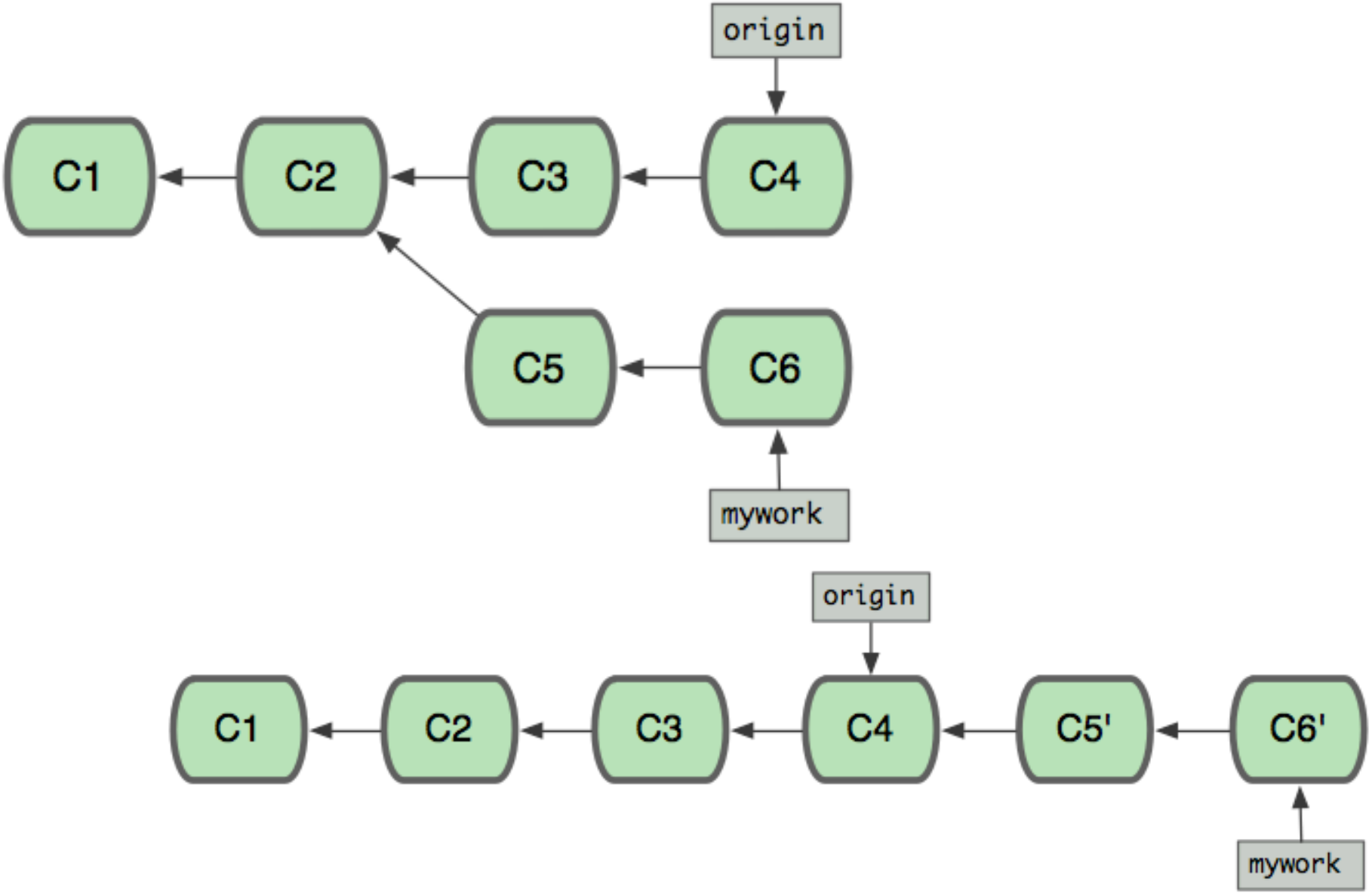


A **pull** is simply a fetch and a merge

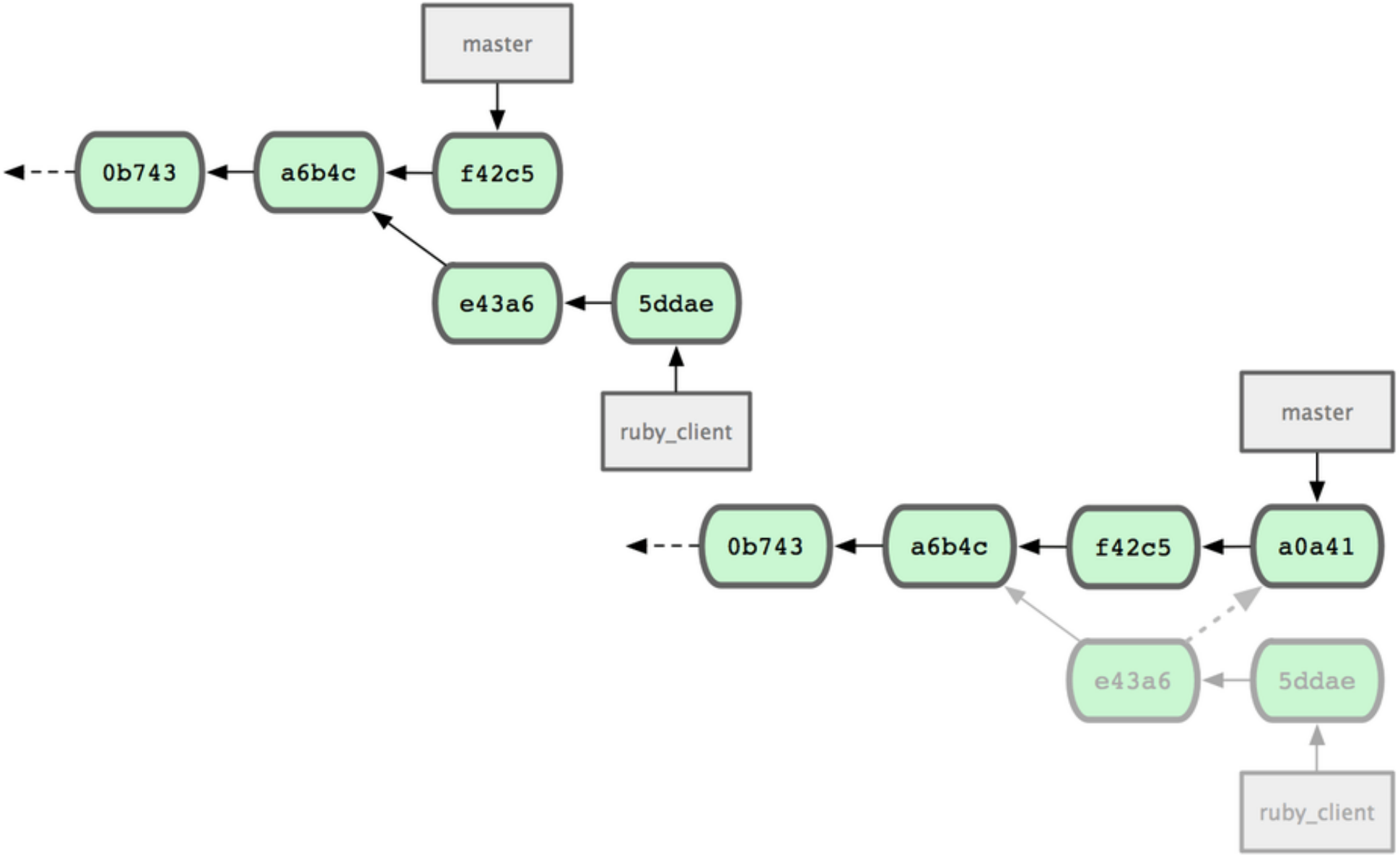
The rebase


origin
C1
C2
C3
C4
C5
C6
mywork
origin
C1
C2
C3
C4
C5
C6
C5'
C6'
mywork

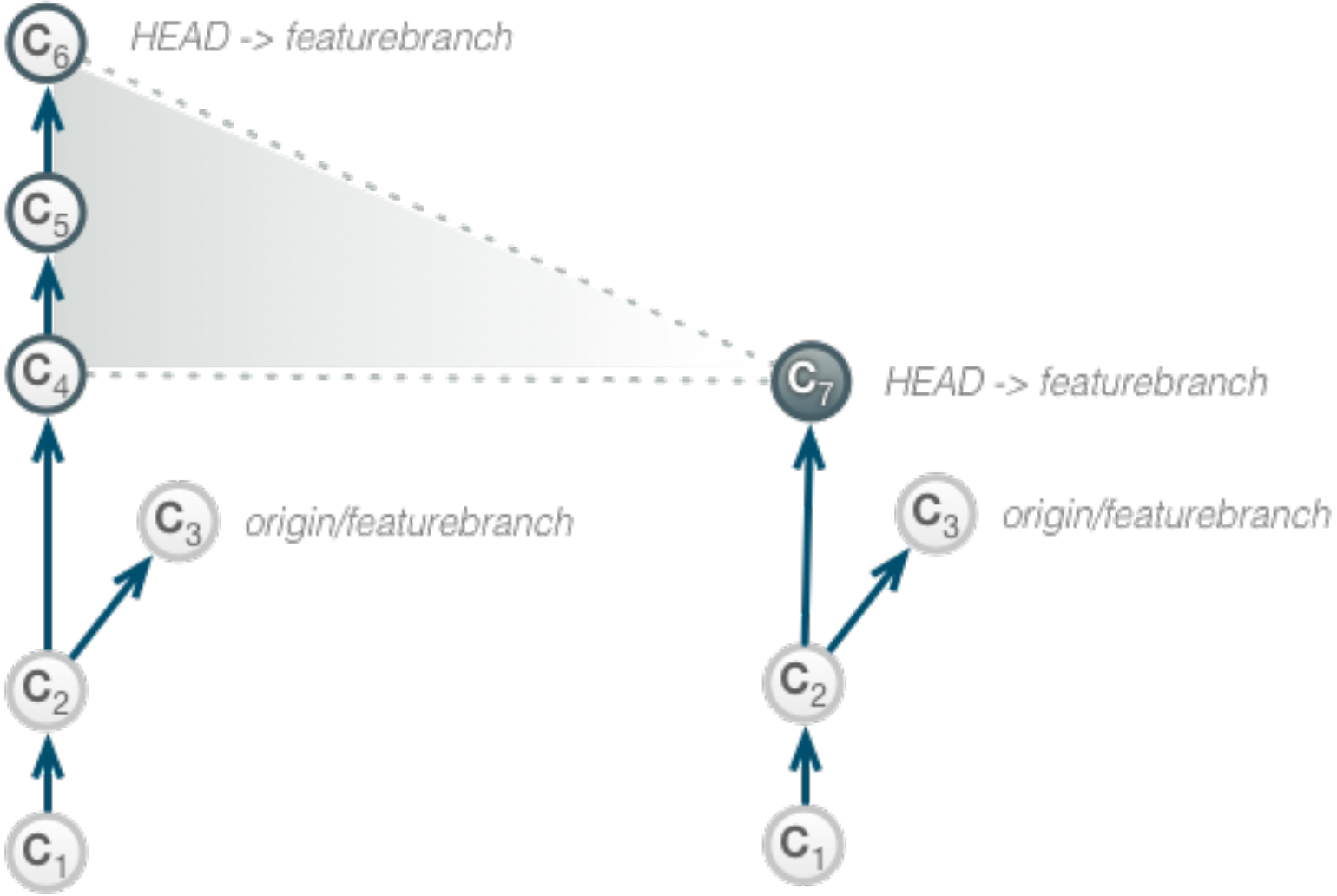
A fast-forward merge



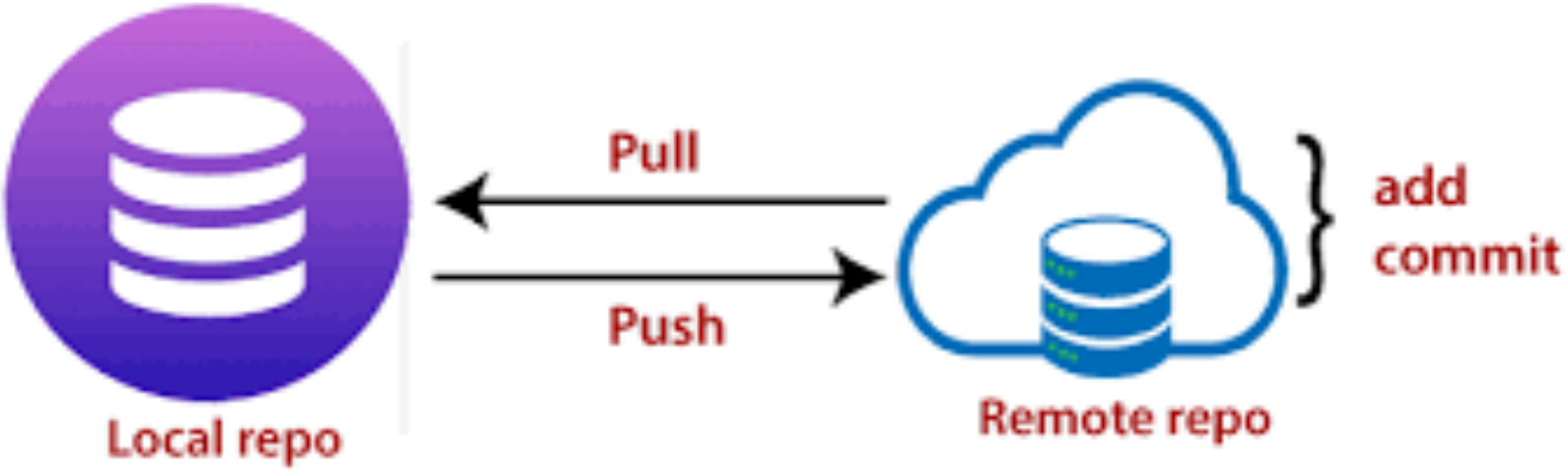
The cherry-pick



The squash


C6
HEAD -> featurebranch
C5
C4
C7
HEAD -> featurebranch
C3
origin/featurebranch
C3
origin/featurebranch
C2
C2
C1
C1

Push



How to propagate commits

- Using email
  - Create one or more emails with the patches
  - Recipient can take those emails and add their commits to their repo
- Push
  - Requires write access to the destination repo (usually public)
- Pull
  - Requires read access to the source repo (usually public)

Pull request

- A developers pushes changes to a public repo
- Then asks another developer to pull those changes to his/her local repository
- If accepted, the changes are then pushed to the public repository of the recipient
  - GitHub removes the need to do the pull request locally
  - It can be done from one public repo to another

Committer/Author?

- The author is the one that “writes” the patch
- The committer is the one that “commits” it to the current repo
- Most of the time, it is the
- Both author and committer
  - Name + email

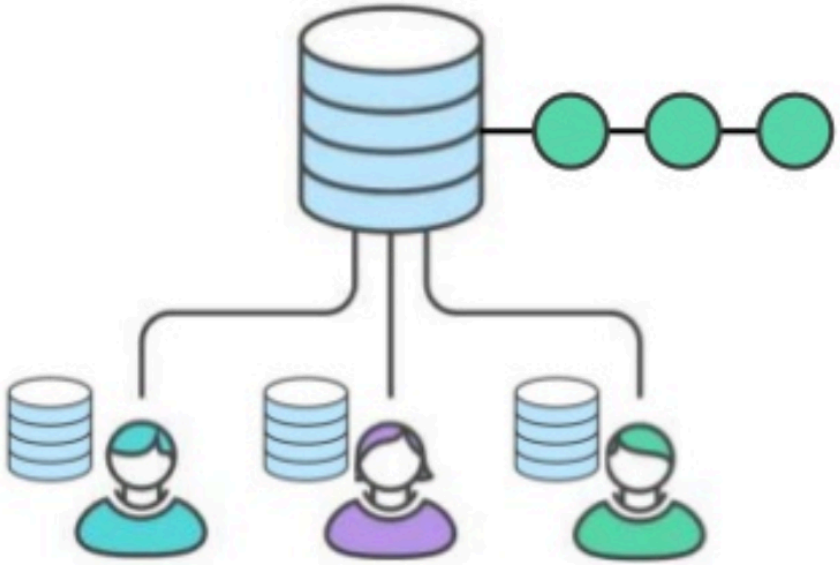
Several operations “recommit” a change

- Rebase, Squash and cherry-pick
  - The resulting commit keeps its author, but the committer is the one that does the change
  - If the commit has the same date for author and commit
    - Then the commit has never been rebased/ squashed/cherry picked

The workflows

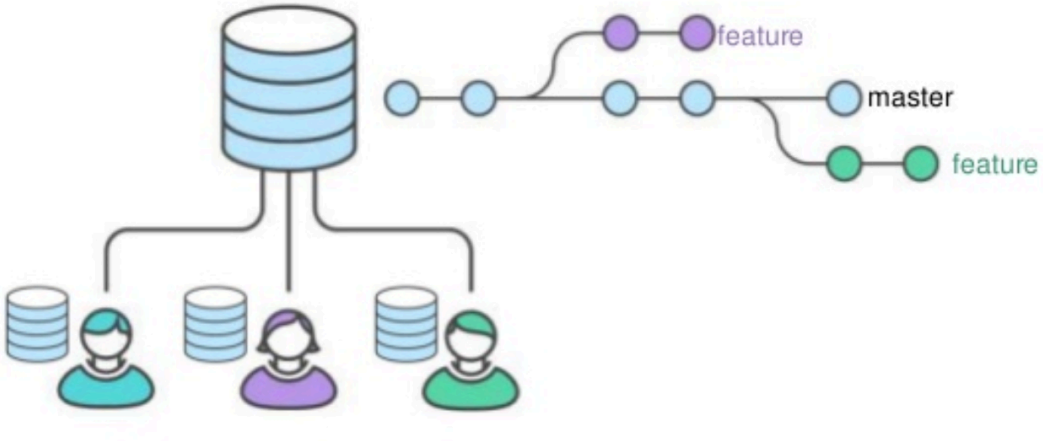
Centralized

- Each developer has a local repository
- But there is only one public repository
  - All developers fetch/pull/push to the same repo
  - No branch
  - No code review



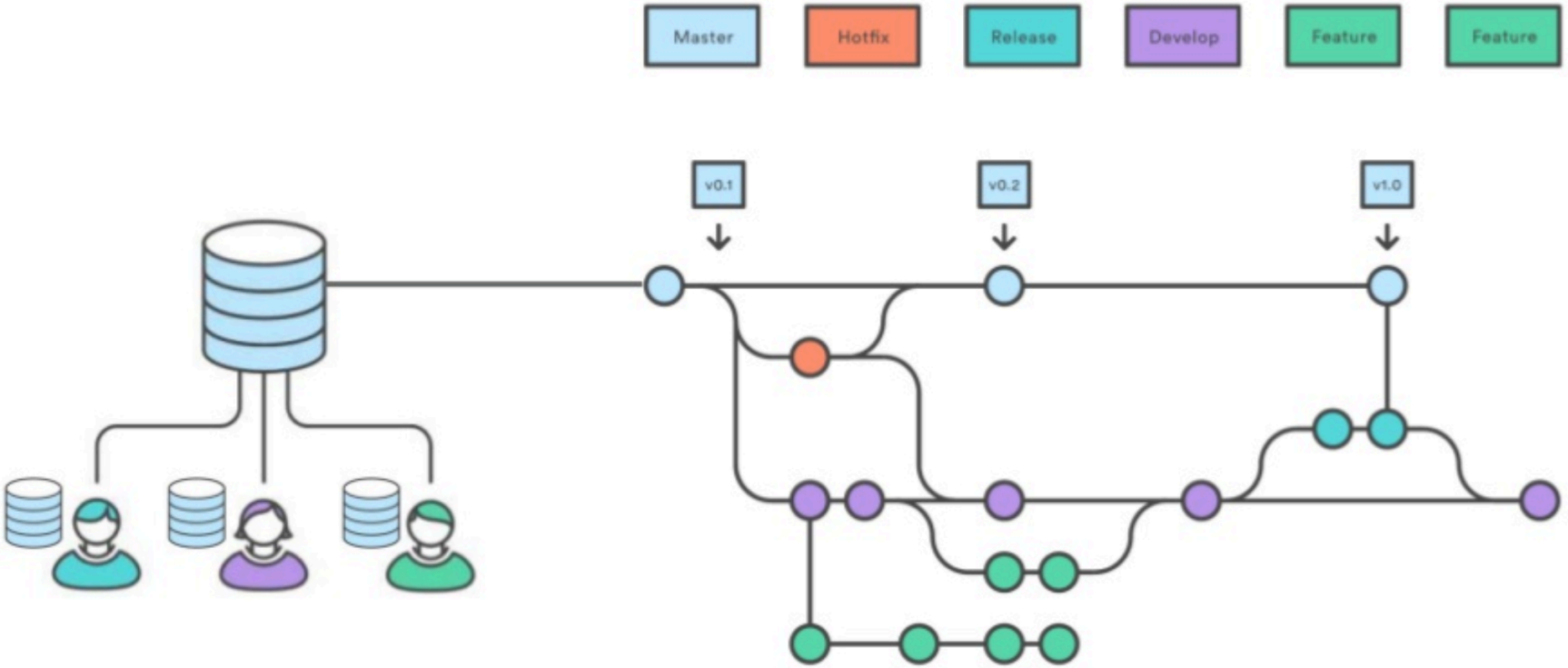
Feature branch

- Variant of centralized (one public repository)
  - Development is done in different branches
  - A branch is merged when the feature is completed
    - At that point, it disappears



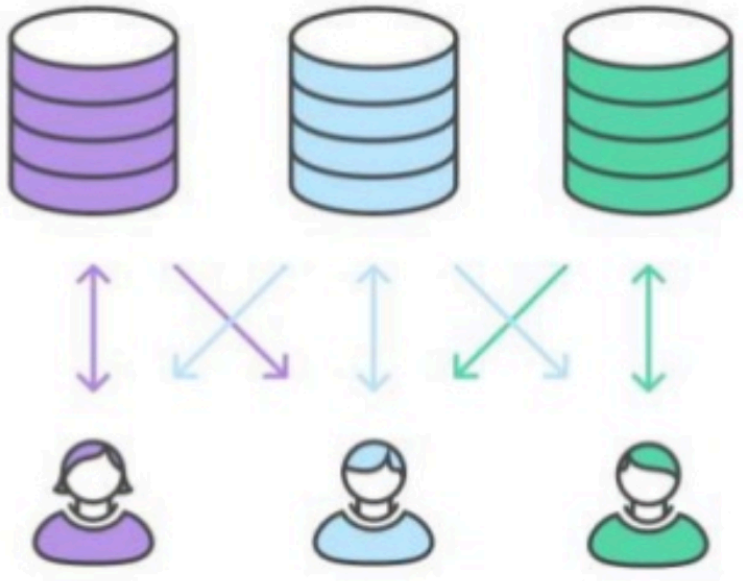
Git-flow

- Variant of centralized (one public repository)
  - More complex branching/cherry picking
  - Branches used for versions



Fork

- Multiple public repos
  - One repo is designed as the official
  - Code flows across repos



The Promises and Perils of Mining Git

Christian Bird*, Peter C. Rigby†, Earl T. Barr*, David J. Hamilton*, Daniel M. German†, Prem Devanbu*

*University of California, Davis, USA
†University of Victoria, Canada

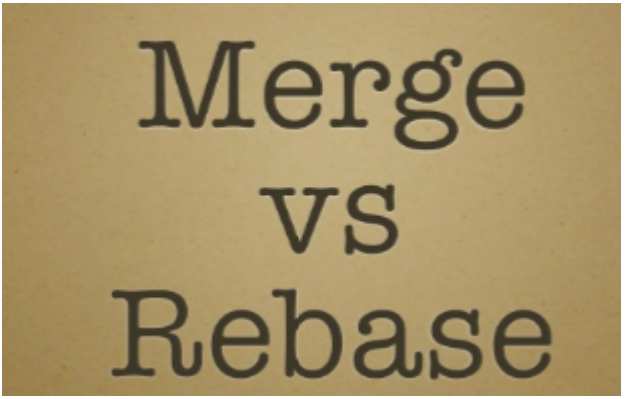
{bird,barr,hamiltod,devanbu}@cs.ucdavis.edu {pcr,dmg}@cs.uvic.ca

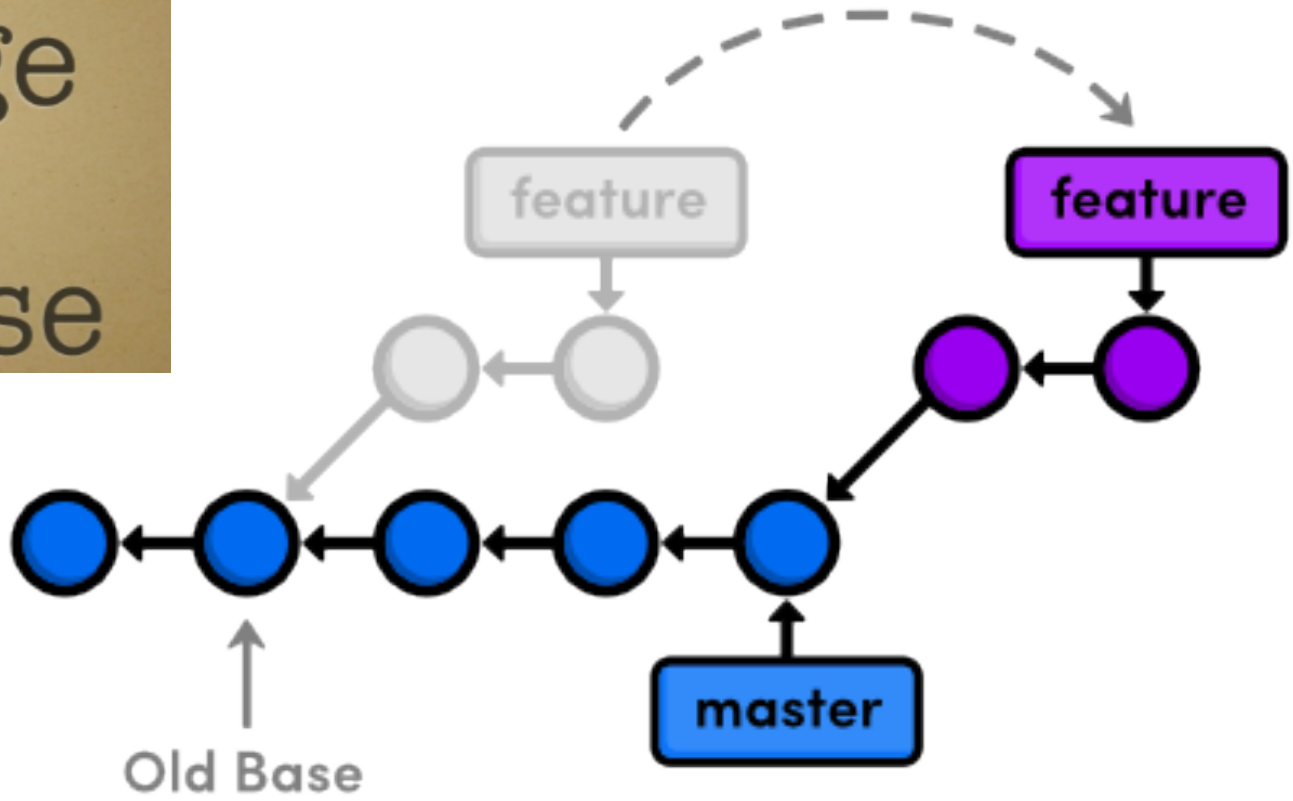
Abstract

e are now witnessing the rapid growth of decentralized rce code management (DSCM) systems, in which every eloper has her own repository. DSCMs facilitate a style collaboration in which work output can flow sideways l privately) between collaborators, rather than always nd down (and publicly) via a central repository. Decenization comes with both the **promise** *of new data and the* **l** *of its misinterpretation. We focus on git, a very popular DSCM used in high-profile projects. Decentralization, and other features of git, such as automatically recorded con-*

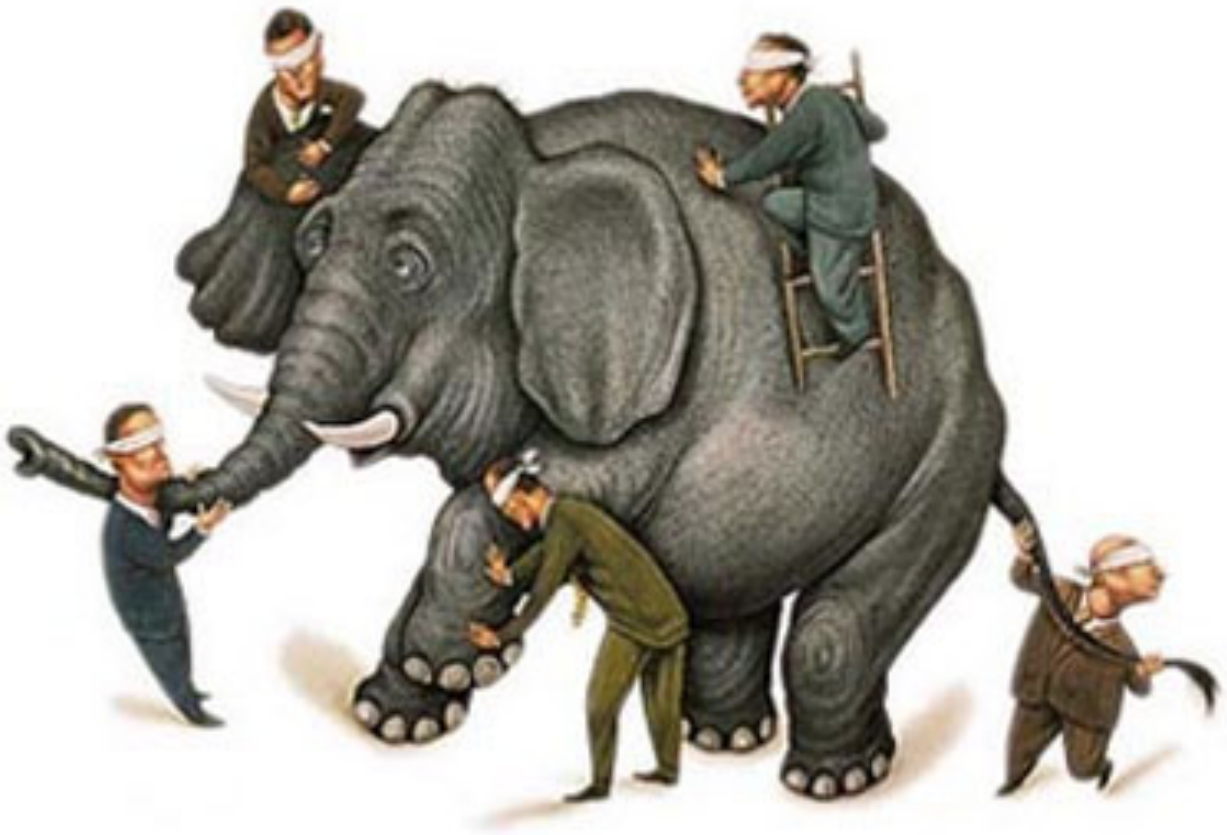
Git's “history”

The Rebase

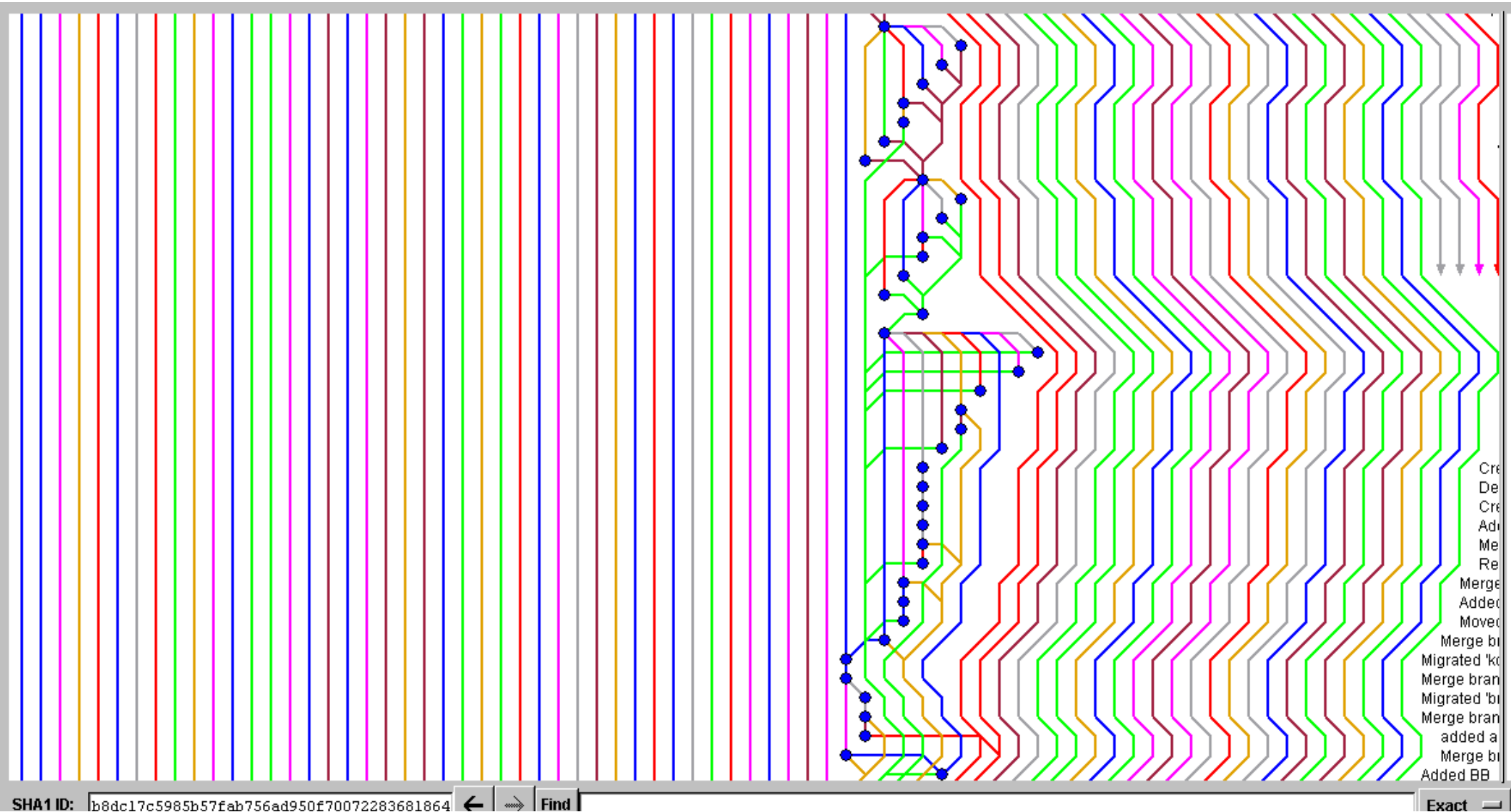




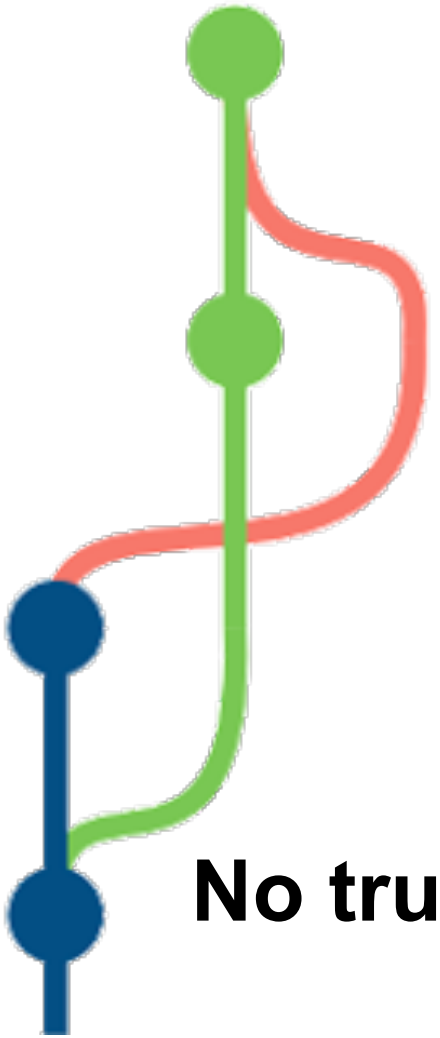
Where is the trunk?



There is **no** trunk

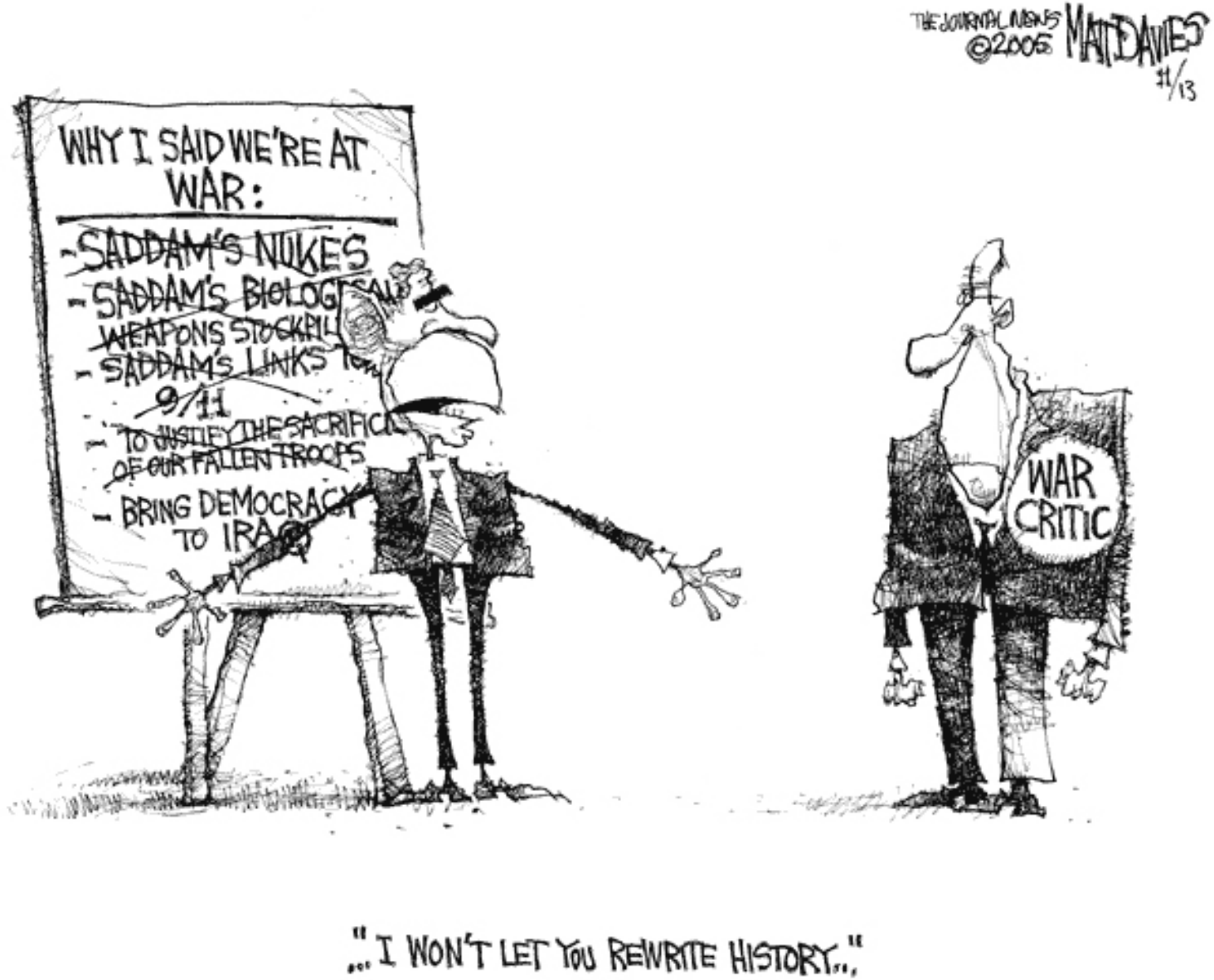


The Foxtrot

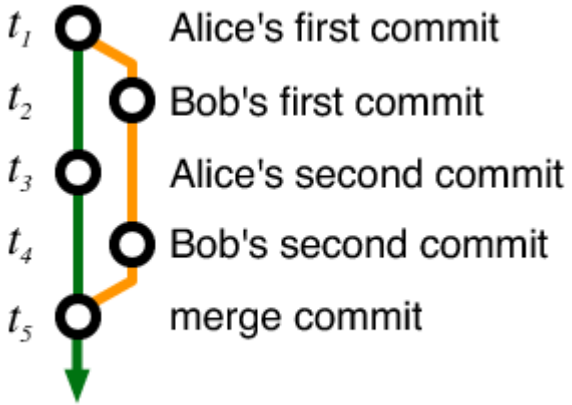


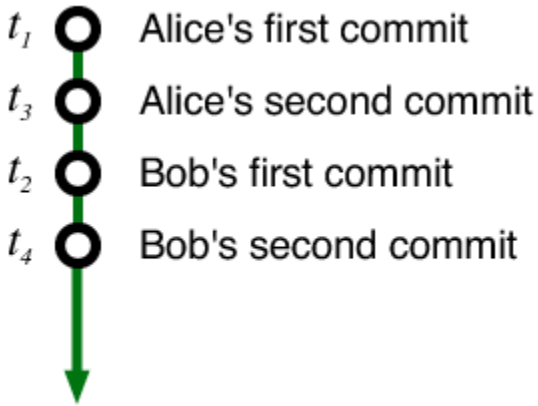
No true concept of “master” branch

History? What is history?



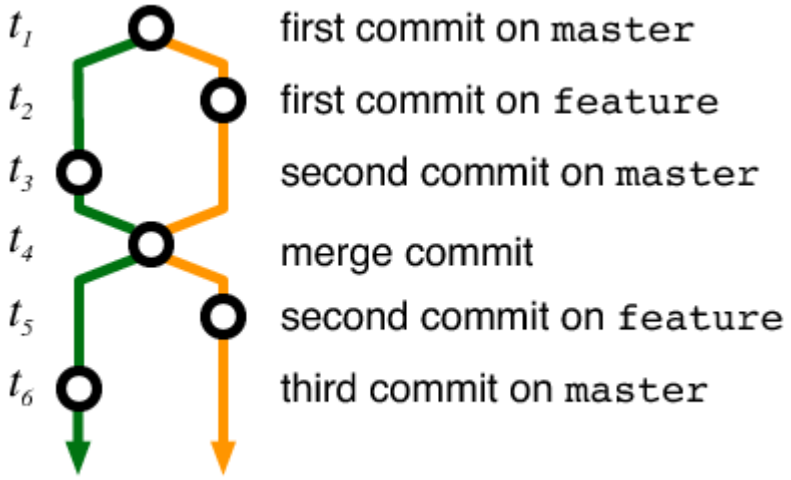
Rebasing





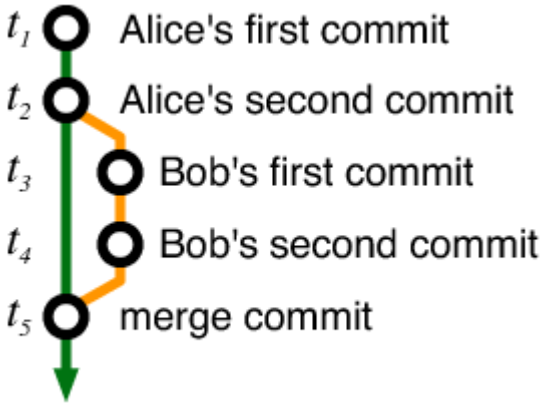
Makes for cleaner history!

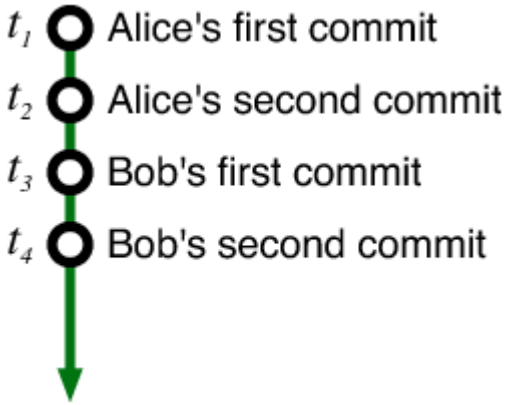
In which repo?



Nobody knows!

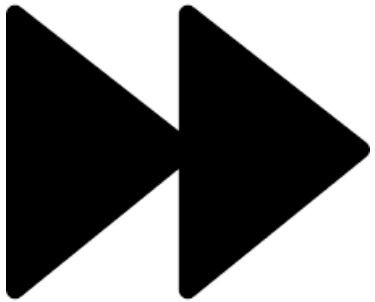
Was there a merge? I didn't see it!

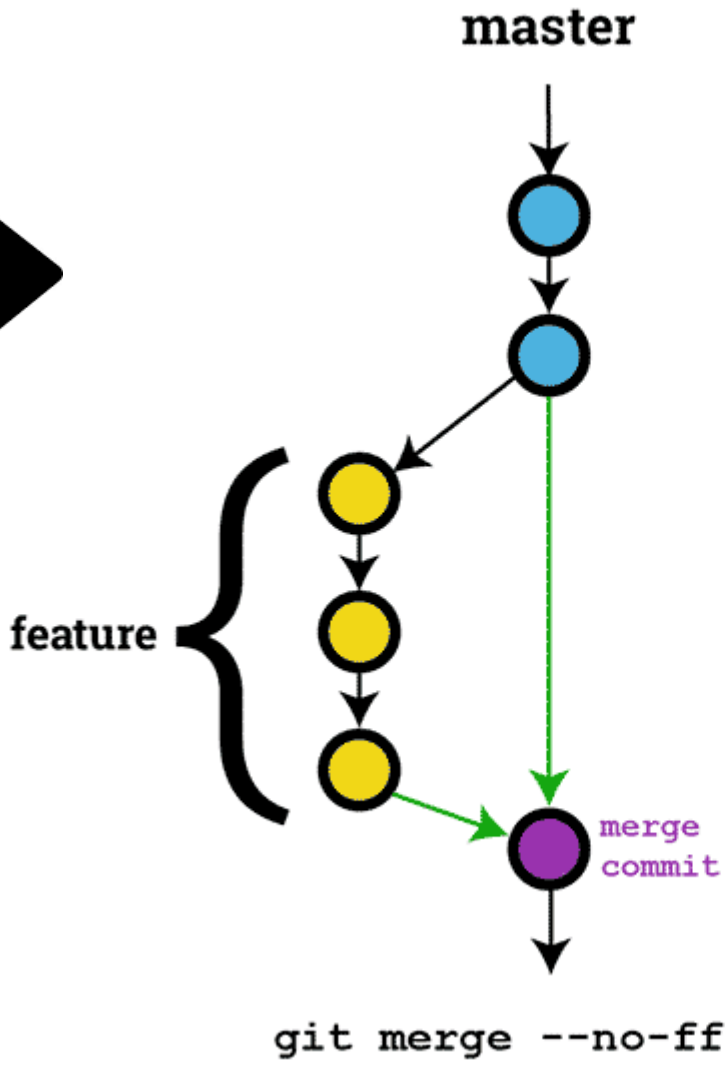


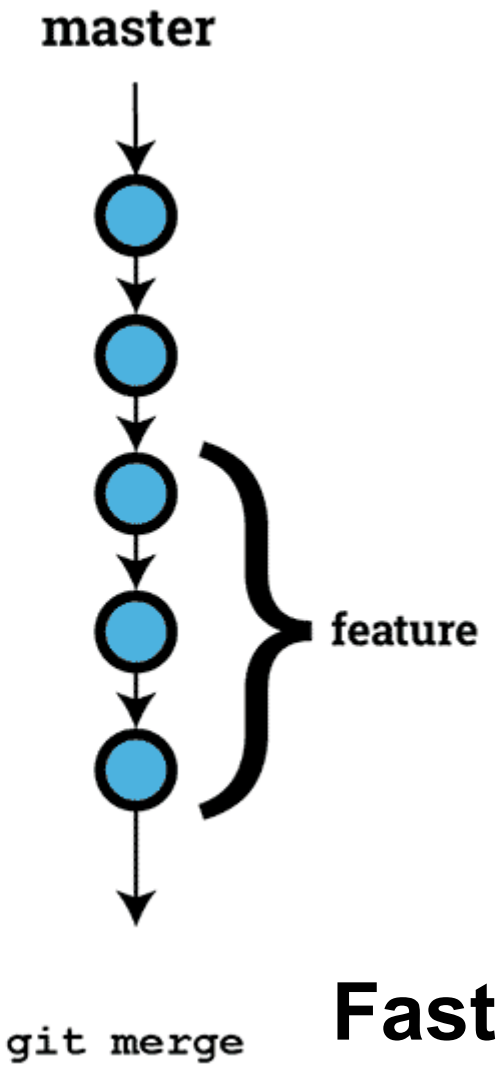


Fast-forward commits

Was there a merge? I didn't see it



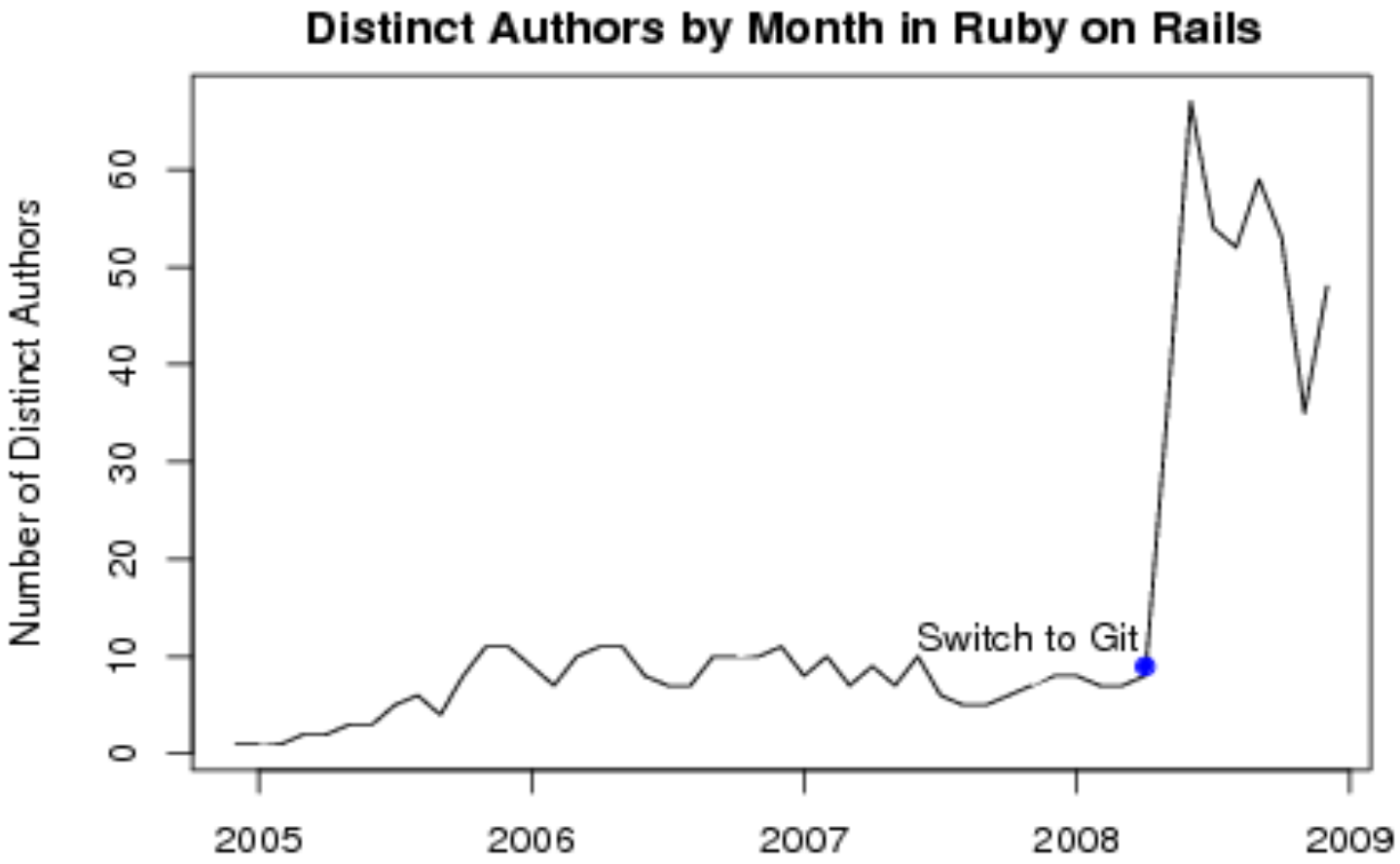




Fast forward commits

Metadata

- Committer is not the same as the author



No centralized logging

Example: postgres

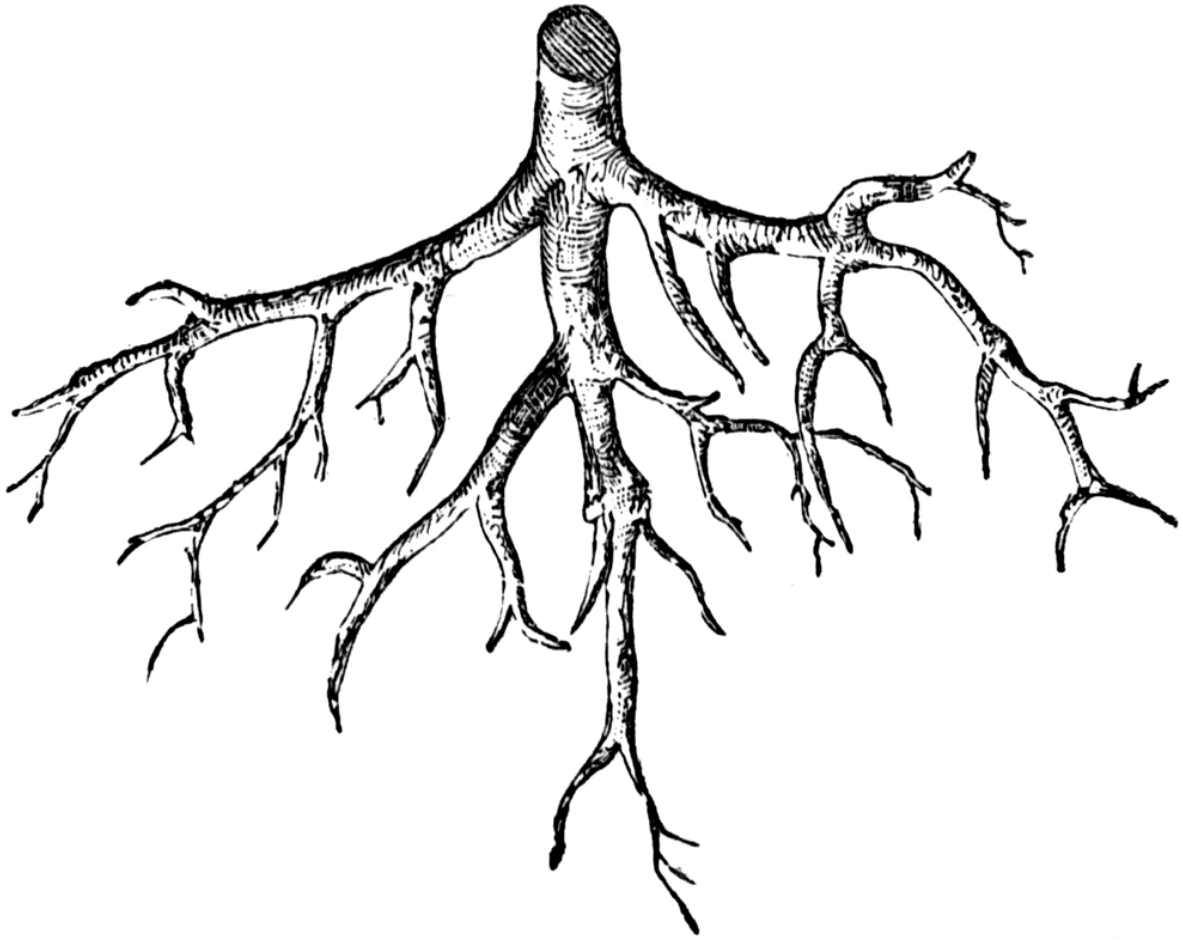
- It does **not have a single merge**
  - It is used as CVS

Linux and git

Can we learn from Linux?

Study Linux main repo

Not a single (open) branch in Linus repo



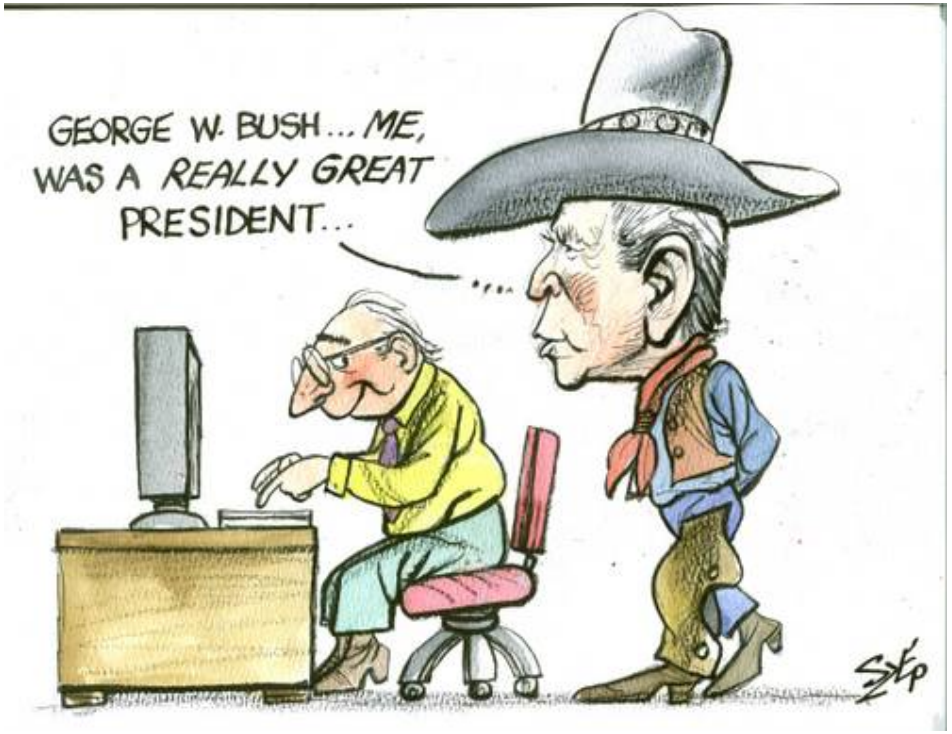
Challenge 1

- Personal repos are beyond reach
- Local commits might never be observable

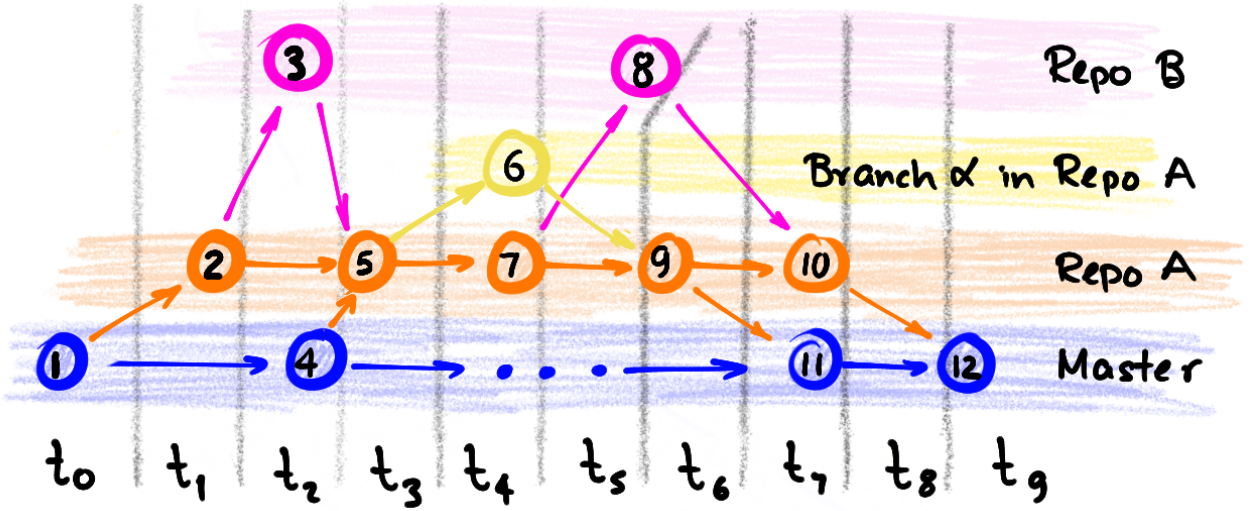
Challenge 2: History

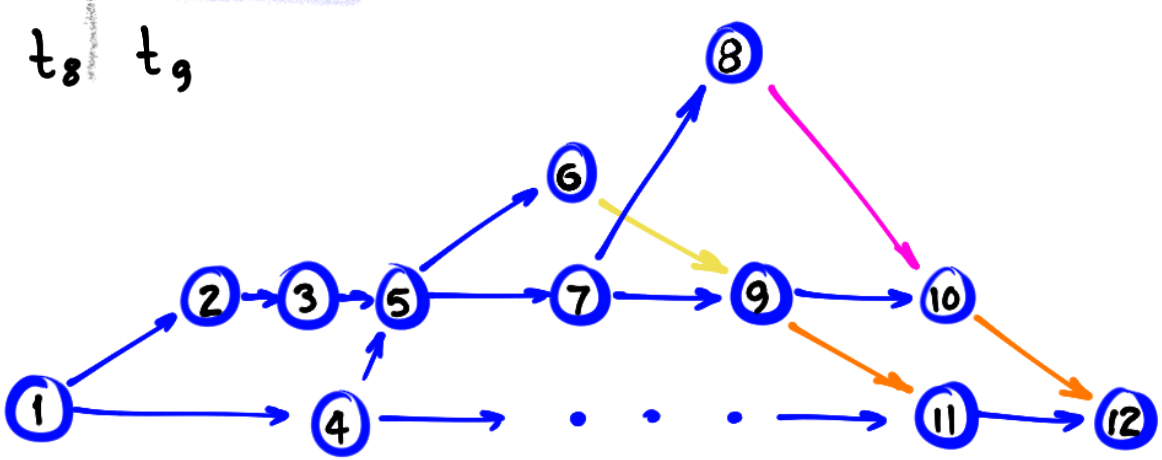
"History is written by the victors"

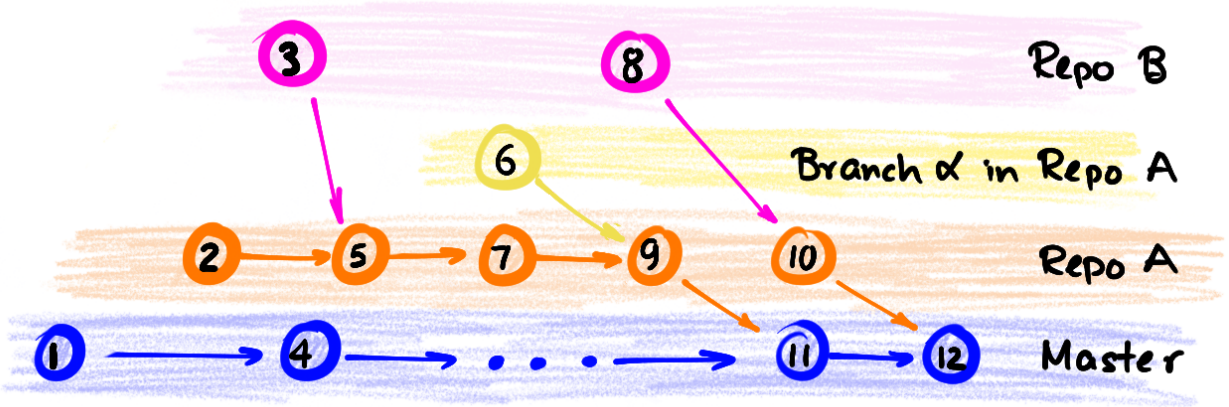
- Niccolò Machiavelli



Save history before it is lost!

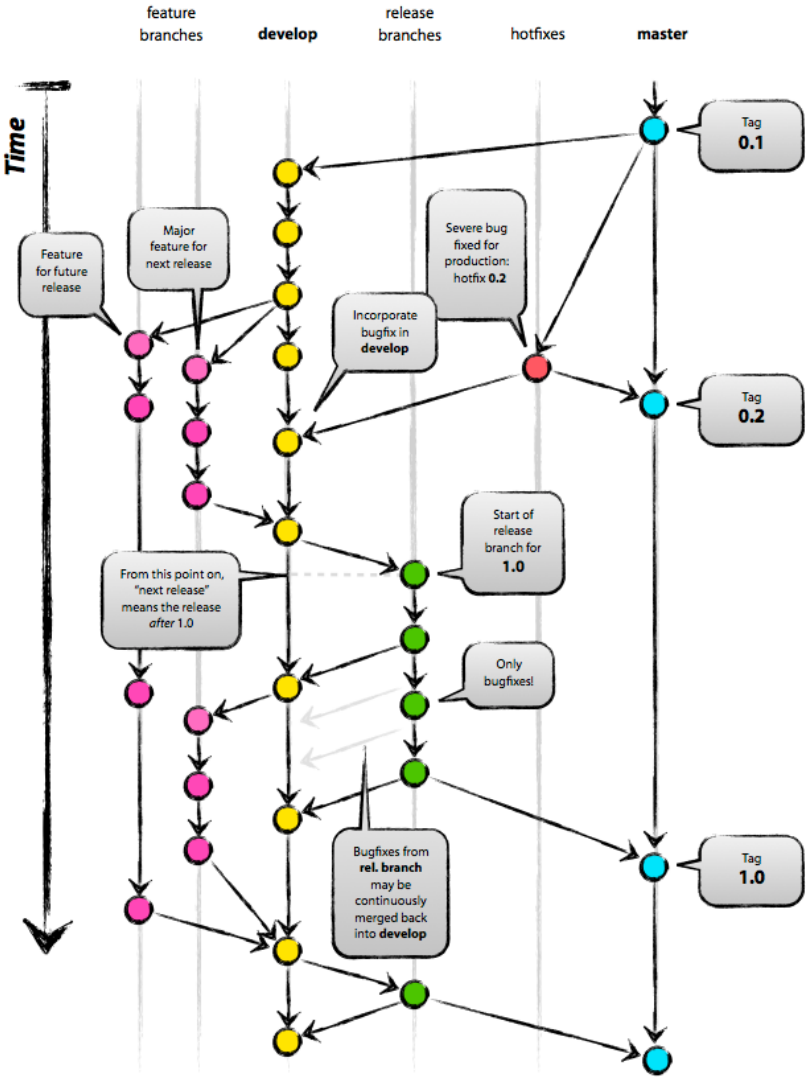






Super-repository

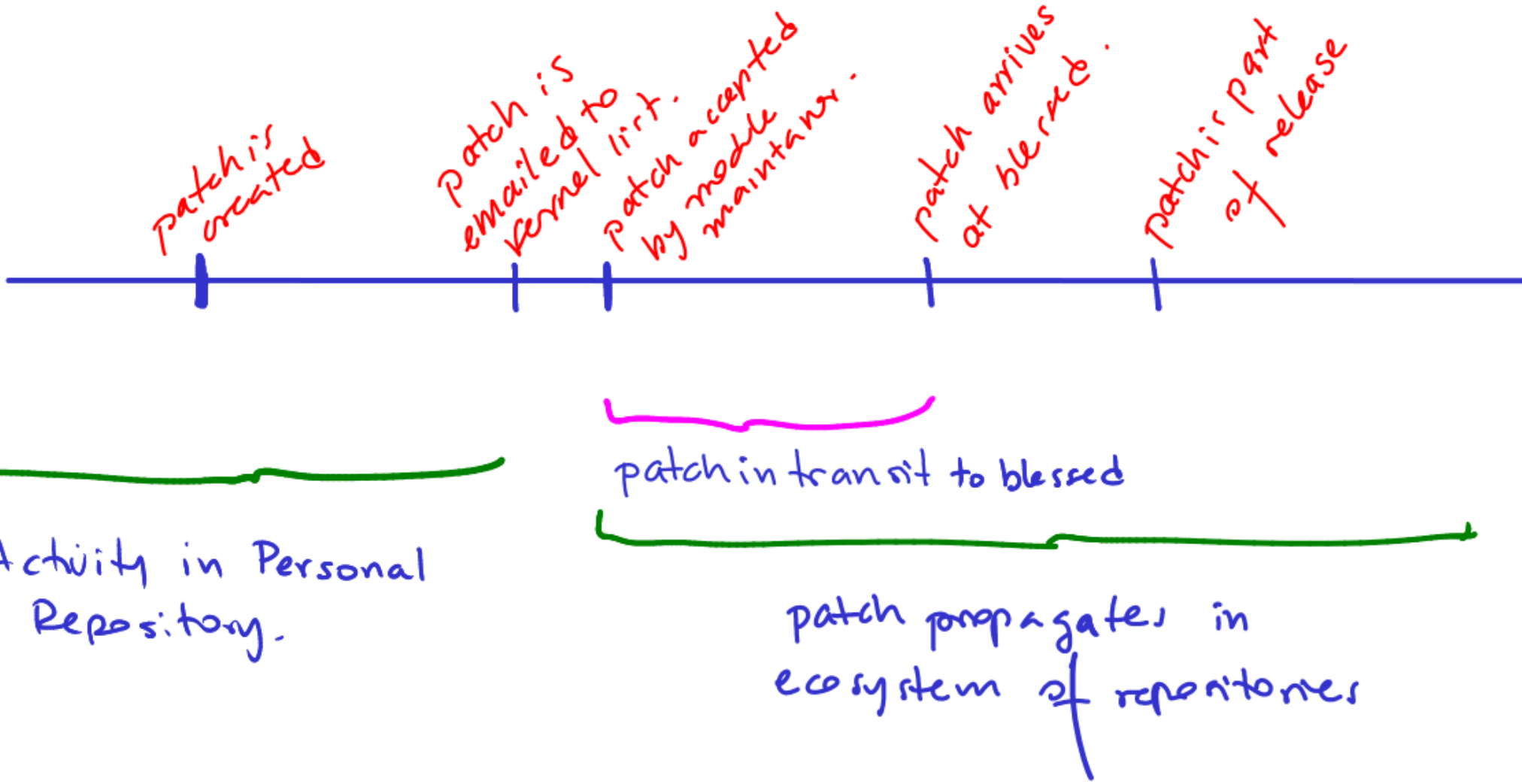
- Collection of repositories cloned (recursively) from the same repo
  - At least one per developer
    - In their personal compute
  - At least one public repositor
    - The **blessed**
  - In git, no way to trace them



Moving commits across the superRepo

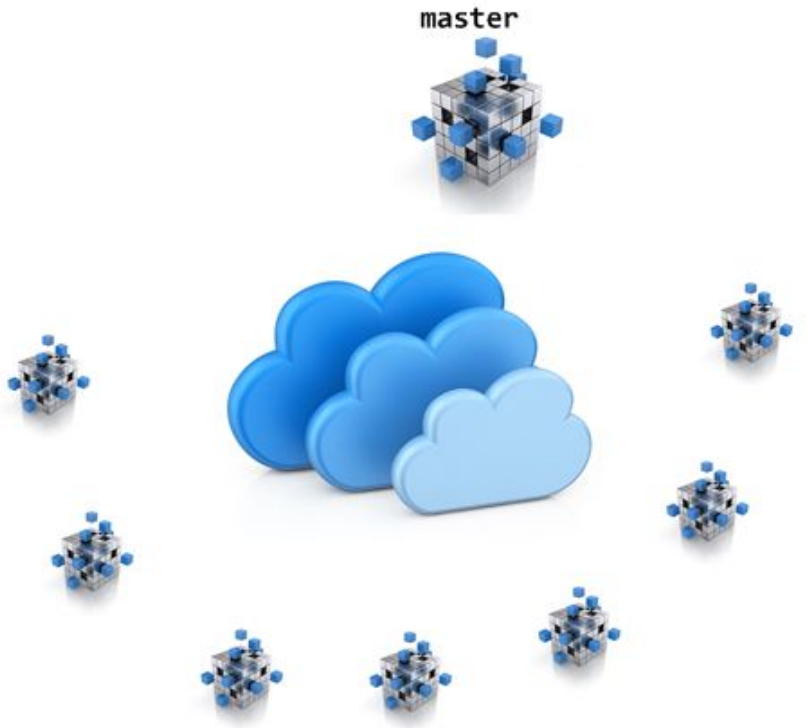
Method	
Push	Done at source, needs write access to source
Pull	Done at destination, needs read access to source
Email	Source creates patch, recipient applies it

Life of a Patch in Linux

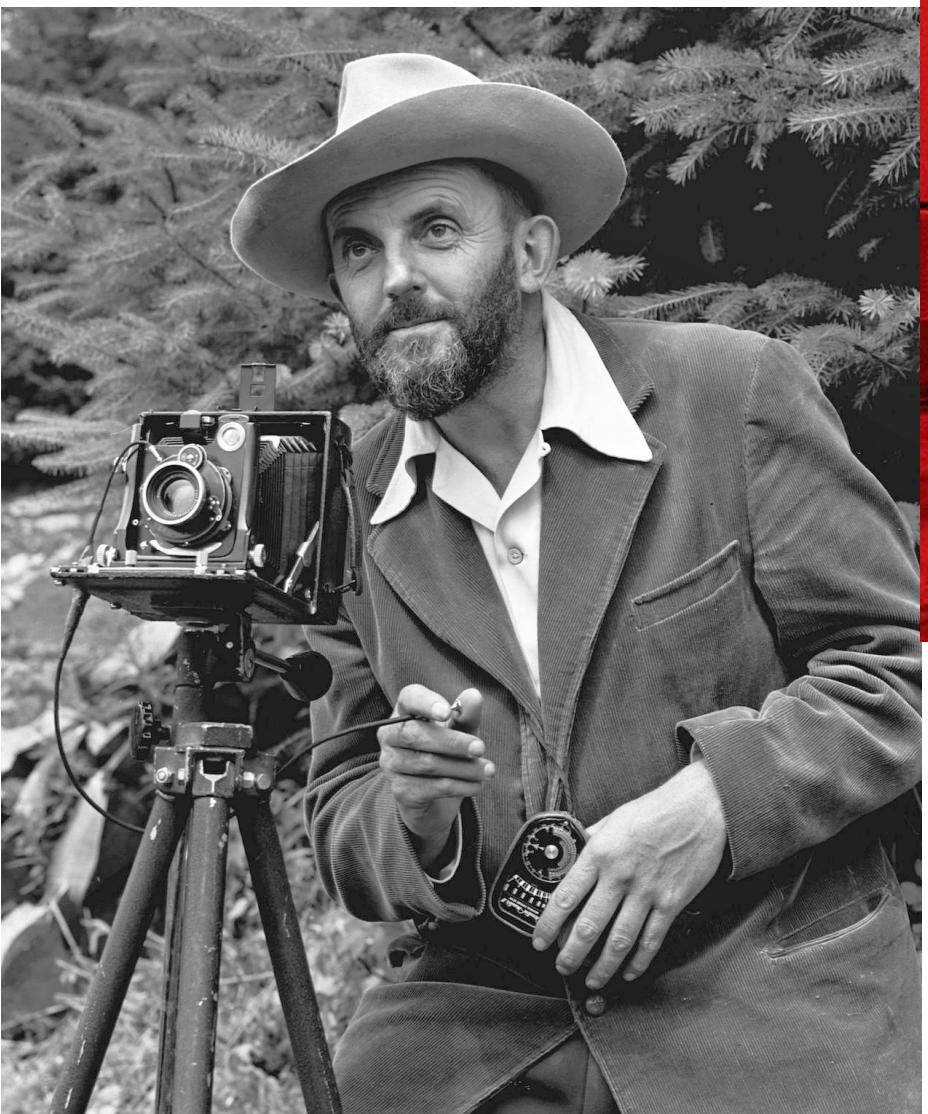


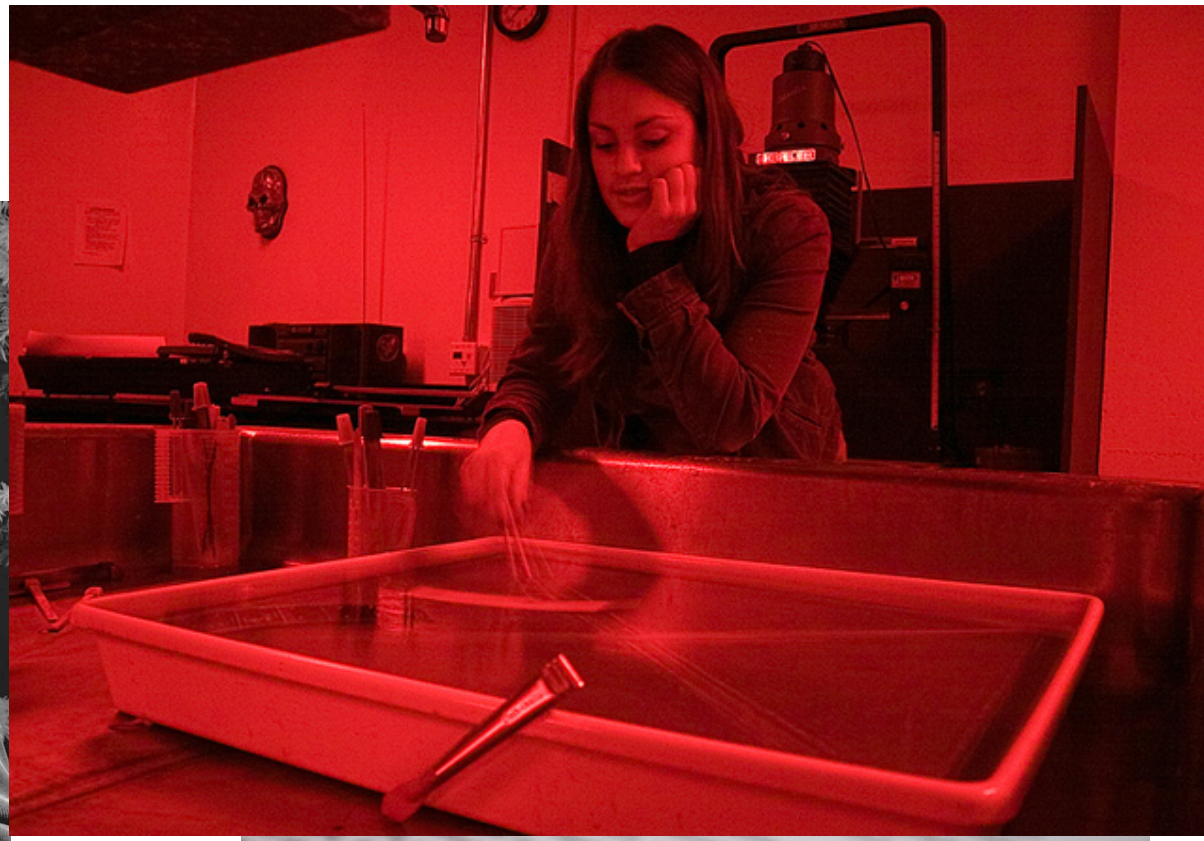
How can we observe them?

We have to find them!

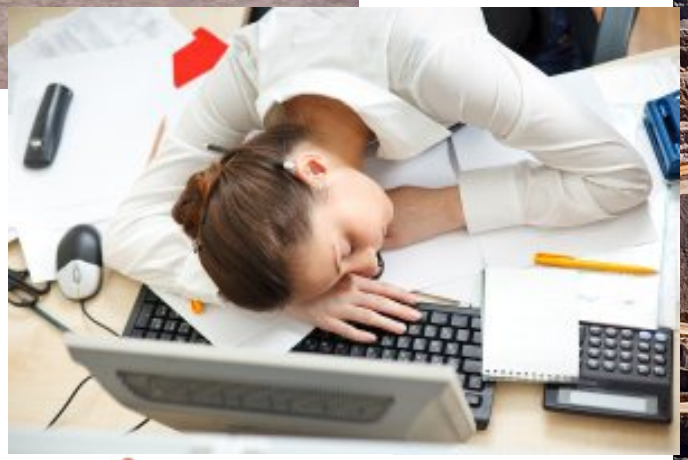


Snapshot Mining

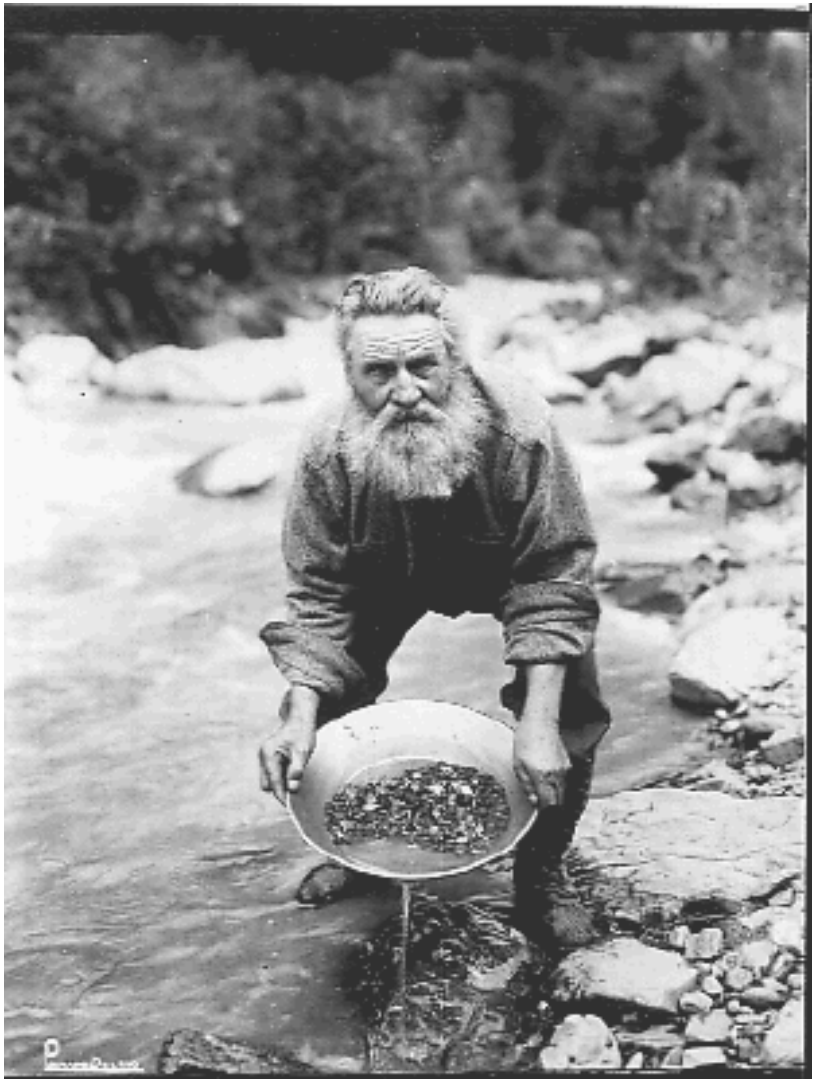




Continuous Mining



Continuous Mining



- Every interval
  - Mine as many repos as known
    - What is new?
    - What has been deleted?

Continuous Mining

- Challenges:
  1) Finding the repositories
  2) Logging their commits

1) Proactive vs Reactive implementation
   - Logging vs discovery

ContinuousMining of Linux

- Linux has no centralized logging
  - Nobody really knows what the superRepo is
  - Commits flow without any event broadcasting mechanism
- Where do we find the activity?
  - Repos
  - Commits
  - Mailing lists

Repos and Committers

- Most repos will have a known set of persons committing to them
  - Simplest case: its owner is the only committer
  - Extreme case: repo is used as centralized version control system: everybody commits to it

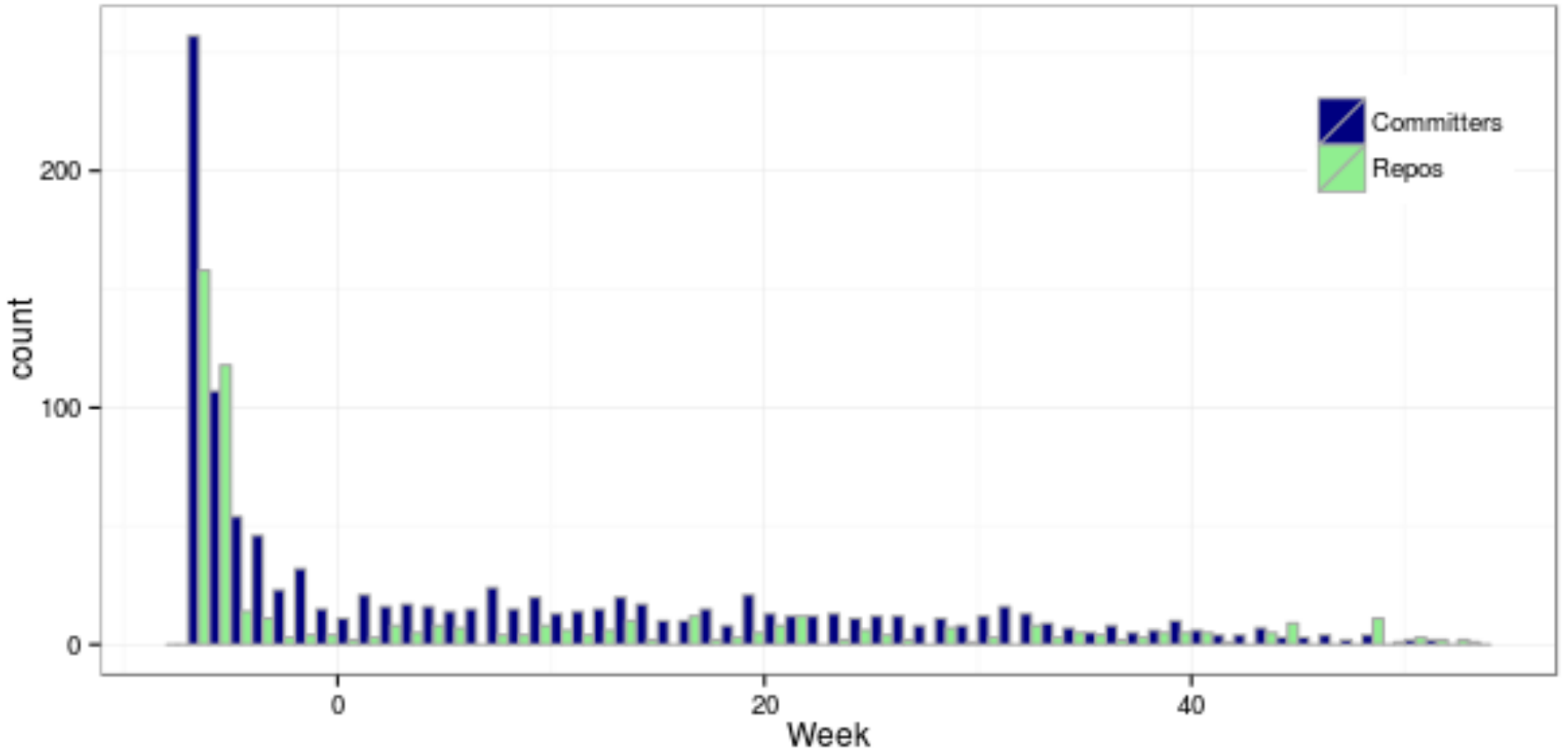
Semiautomatic Process

- **Every 3 hrs**, ask every repo:
  - What new commits do you have?
  - What commits did you delete?
  - Automatically resolve propagations
    - Commits might propagate before we scan
- Daily:
  - Are commits in repo by unknown committers?
    - Answer:
      - **is there a new repo?** or **is committer new to repo?**

Implementation

- Run from Nov 2011 to April 2015
- Retrieved:
  - 3.8 M commits (compared to .55M in Linus repo)
  - 670 repositories
  - 160 million propagation records:
    <cid, added|deleted,repo,when>

Discovery of new repos



Is one better than the other?

- RQ1
  - Does continuousMining uncover a larger development ecosystem than snapMining?
- RQ2
  - Does continuousMining expose any missing information, or bias in the recorded history of the project recovered using snapMining?

	Snapshot (Linus)	Continuous
No Repos	1	479
Commits	64k	533k
Non-merge Commits	59k	485k
Unique Non-merges	58k	135k
%unique non-merges	98.9%	27.9%
Non-merges that reached Blessed		43.1%
Different authors emails	3434	5646
Different authors	2883	4575
Different committers emails	283	1185
Different committers	245	1058

- RQ2
  - Does continuousMining **expose any missing information**, or bias in the recorded history of the project recovered using snapMining?

Commit vs Patches

	Blessed	Super-repository
Non-merge commits	58,953	485,027
Patches	58,356	135,532
Ratio of patches to commits	98.9%	27.9%
Ratio of commits committed in 2012 that reached *blessed*		12.1%
Ratio of patches created in 2012 that reached *blessed*		43.1%

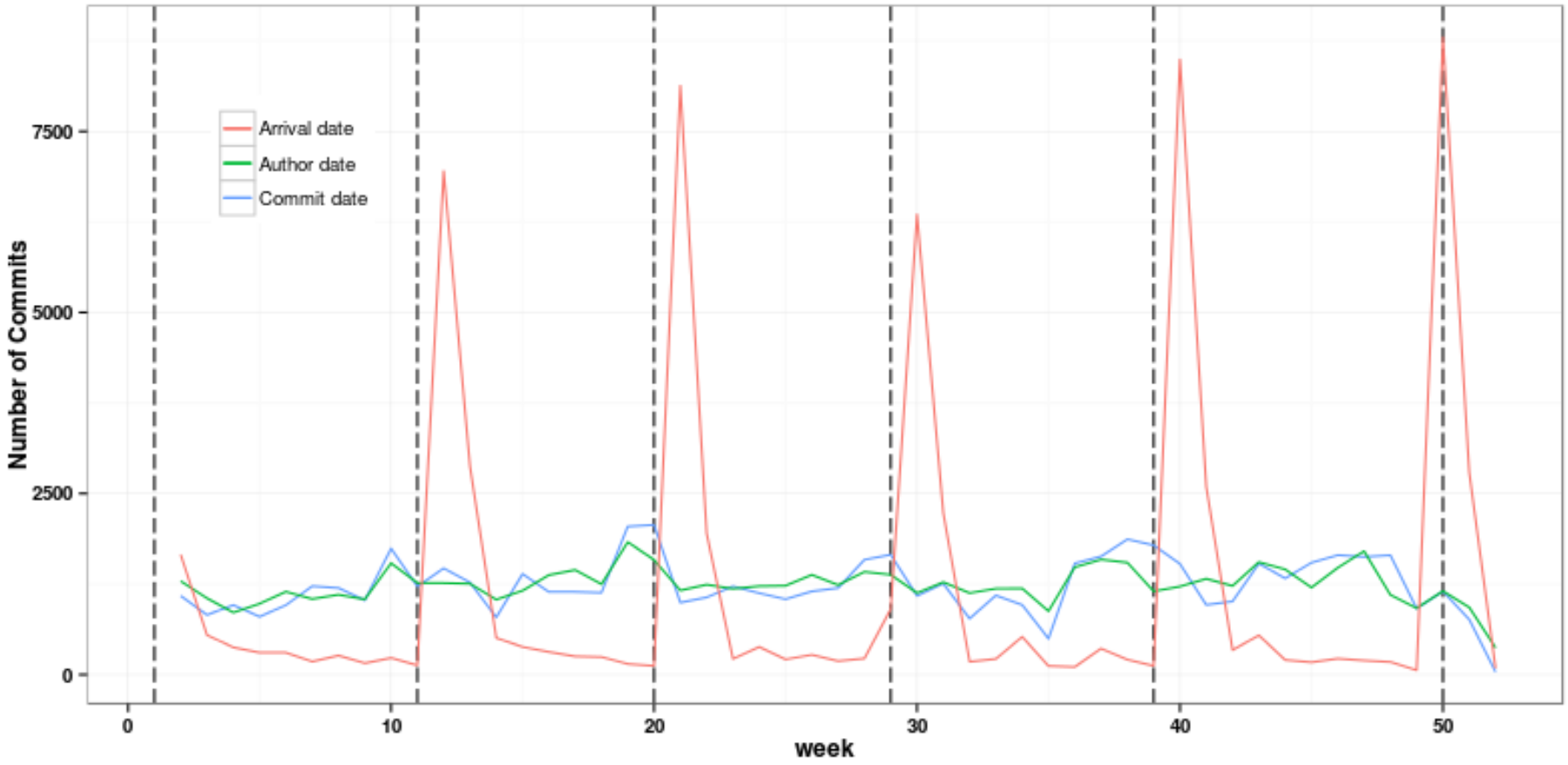
- Due to rebasing, commit **ids are insufficient** to tracks patches
- Large amount of work not reaching *blessed*

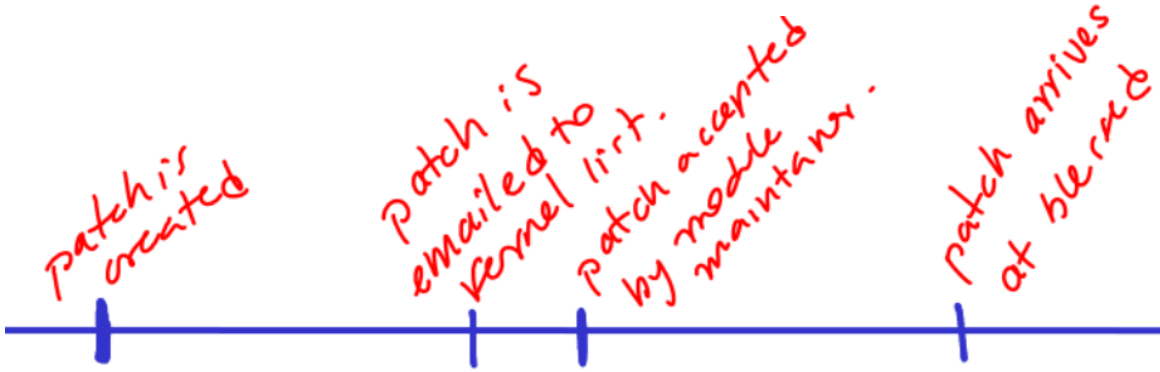
Committer data in *blessed* is biased

- 37.9% of patches arriving at ***blessed*** did not arrived in its original commit

	Count	Prop
Changed date of commit	22,326	37.9%
Changed committer	7,330	12.4%
Changed date of authorship	6,725	11.4%
Changed author	203	00.3%
Changed log	13,323	22.6%
Changed log summary (subset of change log)	2,634	04.5%

Arrival of Commits at Blessed





Arrival of Commits at Blessed...

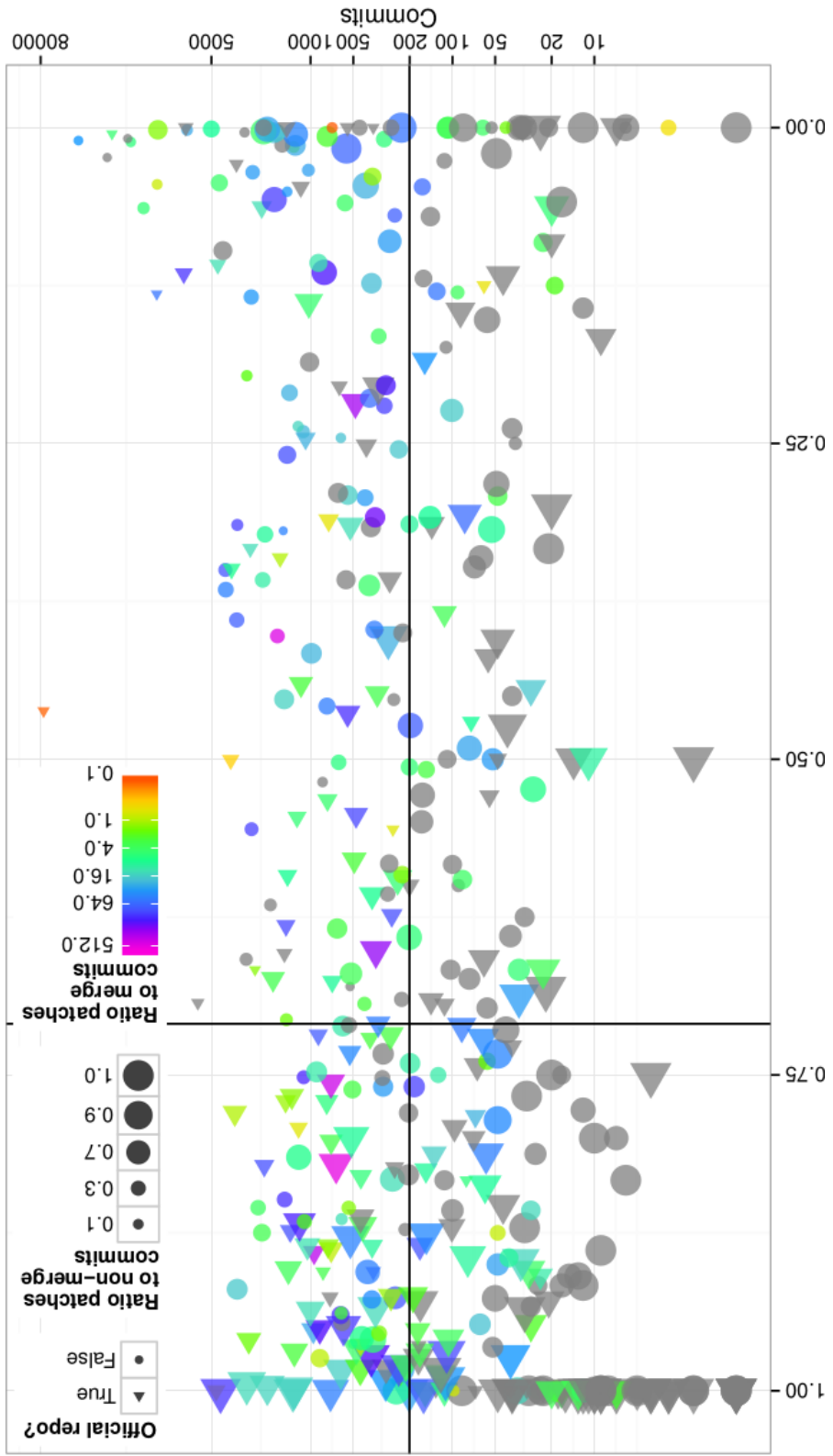
- We can classify patches as a new feature or bug-fix

Release	Merge Window	Ratio Identified Bugs	Bug-Fixing Window	Ratio Identified Bugs	Total
3.2	9,459	15.8%	1,955	42.9%	11,415
3.3	9,858	17.5%	1,969	41.2%	11,828
3.4	10,068	13.3%	1,702	42.1%	11,771
3.5	9,160	15.1%	1,968	41.9%	11,129
3.6	11,083	15.4%	1,920	42.4%	13,004
3.7	11,737	15.4%	(incomplete)		

Empirical study

- RQ. What are the characteristics of the repos in the Linux Super-repository

	Min	1st Q	Median	Mean	3rd Q	Max	*blessed*	Total	*Snap.*
Total Commits	1	34	198	1,162.3	717.5	76,279	4,633	533,513	64,029
Contributed to *blessed*	0	2	23	139.5	117.5	4633	4633	64,029	
Ratio contributed to total	0	1%	29%	38%	67%	100%	100%		
Merges	0	0	2	105.6	19.5	23,078	2,103	48,486	5,076
Non-merges	0	33	169	1056.7	617.5	53,201	2,530	485,027	58,953
Patches	0	18	84	269.1	286.5	5,911	1,116	125,535	58,356
Committers	1	1	1	3.2	1	339	1	1,058	245
Authors	1	3.5	12	44.51	38.5	1,061	454	4,575	2,883
Ratio Authors/ Committers	1	4	10	25.5	28.3	454	454	4.32	11.76



The Repos

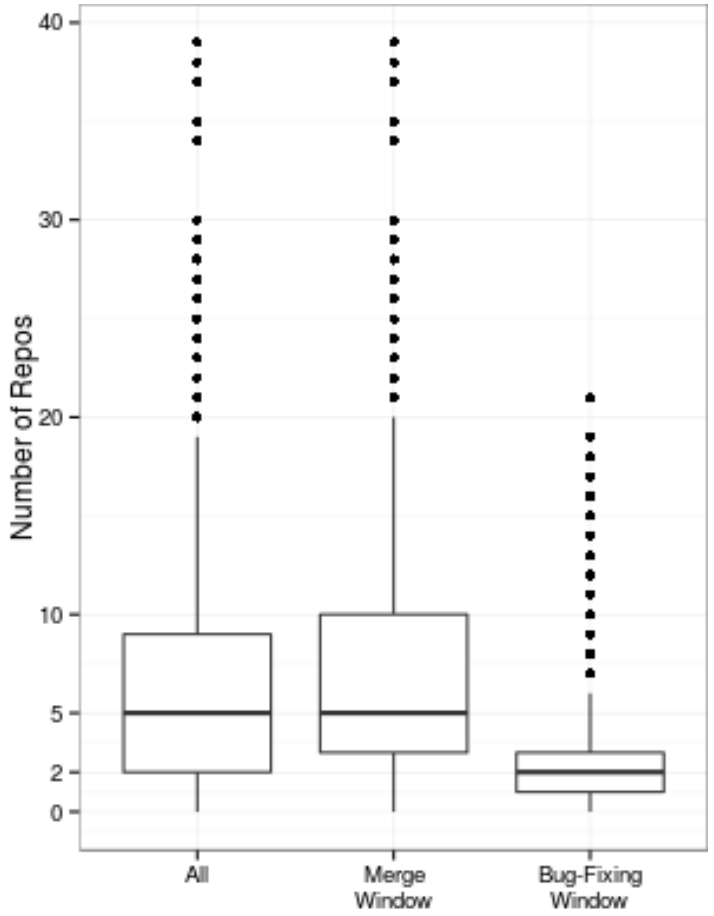
- X: activity (in commits)
- Y: ratio of commits accepted by Linus to total commits
- Shape:
  - Triangles: official repos
  - Circles: non-official repos
- Size:
  - Smaller: consume commits
  - Larger: produce commits
- Color: merge/commit ratio
  - Grey: never merge
  - “Cooler”: high ratio
  - “Warmer”: lower ratio

Consumers are very active

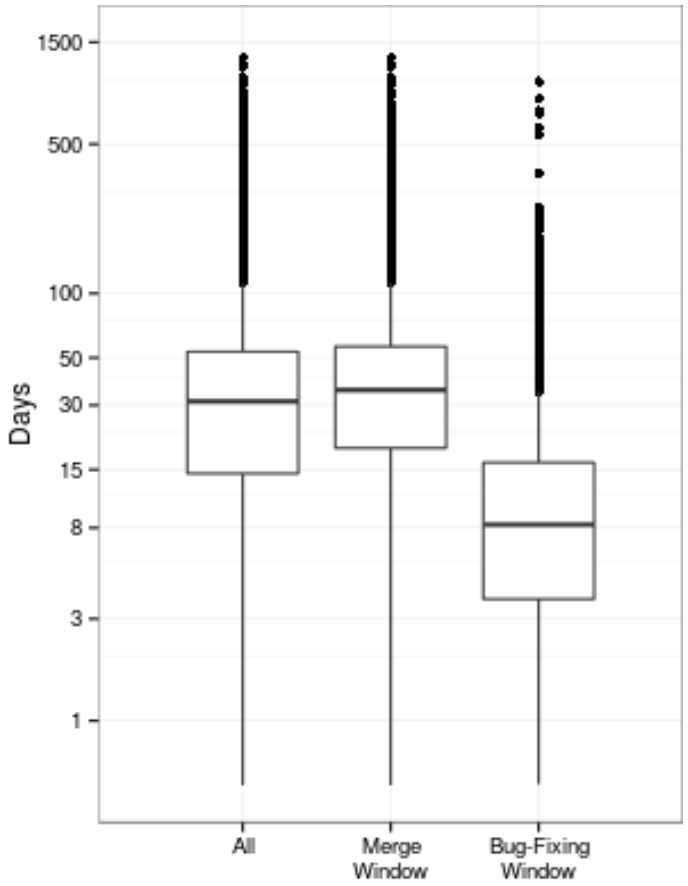
Address	New Commits	Committers	New Patch.	Contr. Patch.	Prop. Patches Contr.
linaro.org/.../andygreen/kernel-tilt	43,290	21	2743	28	1.0%
ubuntu.com/.../**ubuntu-precise**	27,141	6	1184	28	2.3%
kernel.org/.../rt/**linux-stable-rt**	25,129	1	407	2	0.4%
ubuntu.com/.../**ubuntu-quantal**	19,459	6	809	7	0.8%
linaro.org/.../linux-tb-ux500	18,508	21	1765	20	1.1%
linaro.org/../linux-linaro-tracking	12,058	28	1291	58	4.4%
android.googlesource.com/.../**msm**	11,922	339	5911	10	0.1%
kernel.org/.../stable/**linux-stable**	7,591	3	1116	0	0.0%
github.com/vstehle/linux	7,421	14	1079	2	0.1%
denx.de/linux-denx	5,005	69	1915	2	0.1%
opensuse.org/kernel	4,399	41	1716	75	4.3%

Propagation

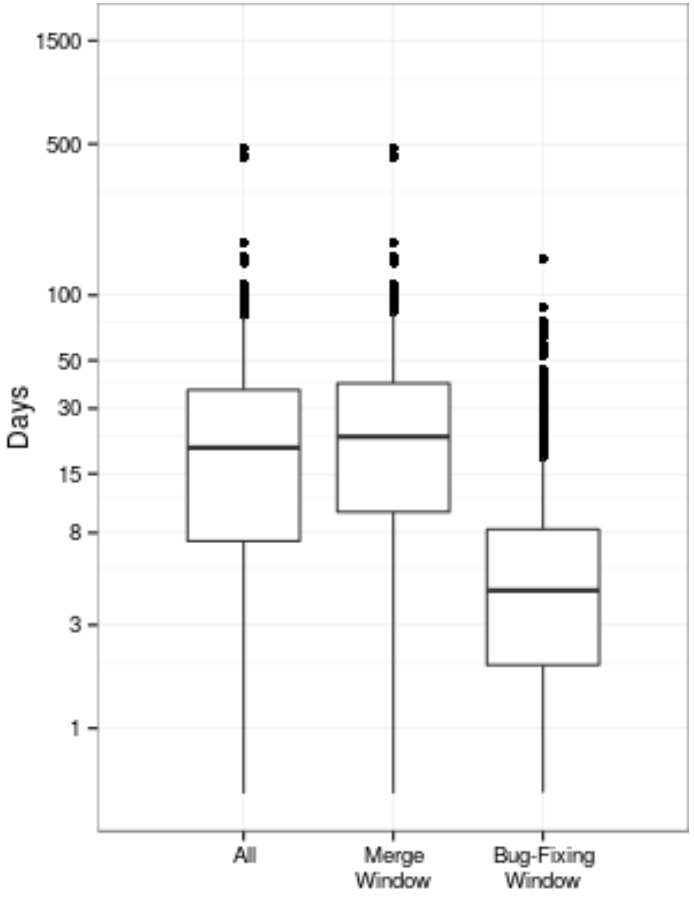
- RQ: How do repositories interact, and how do commits propagate across repositories?



The Latency

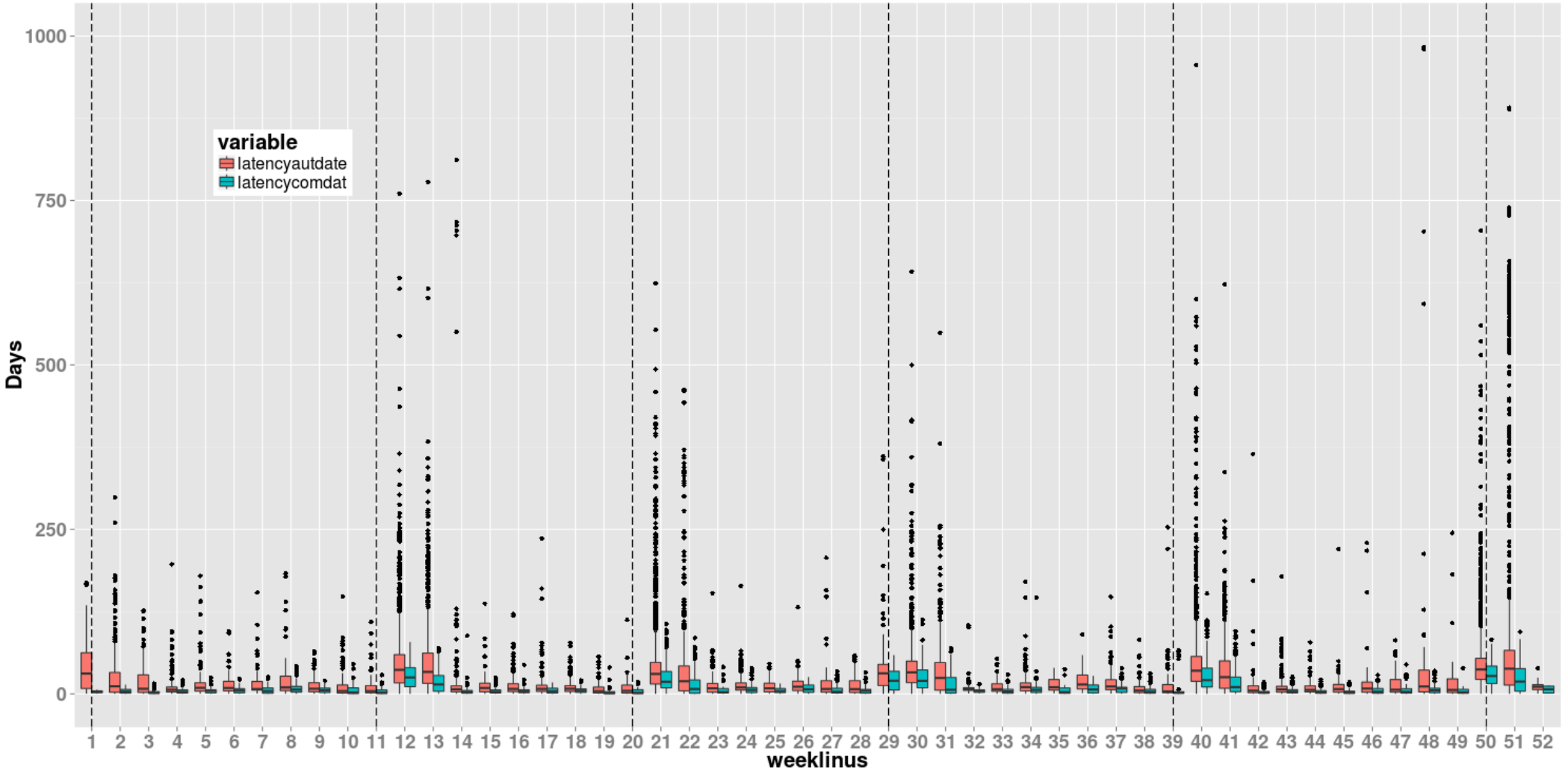


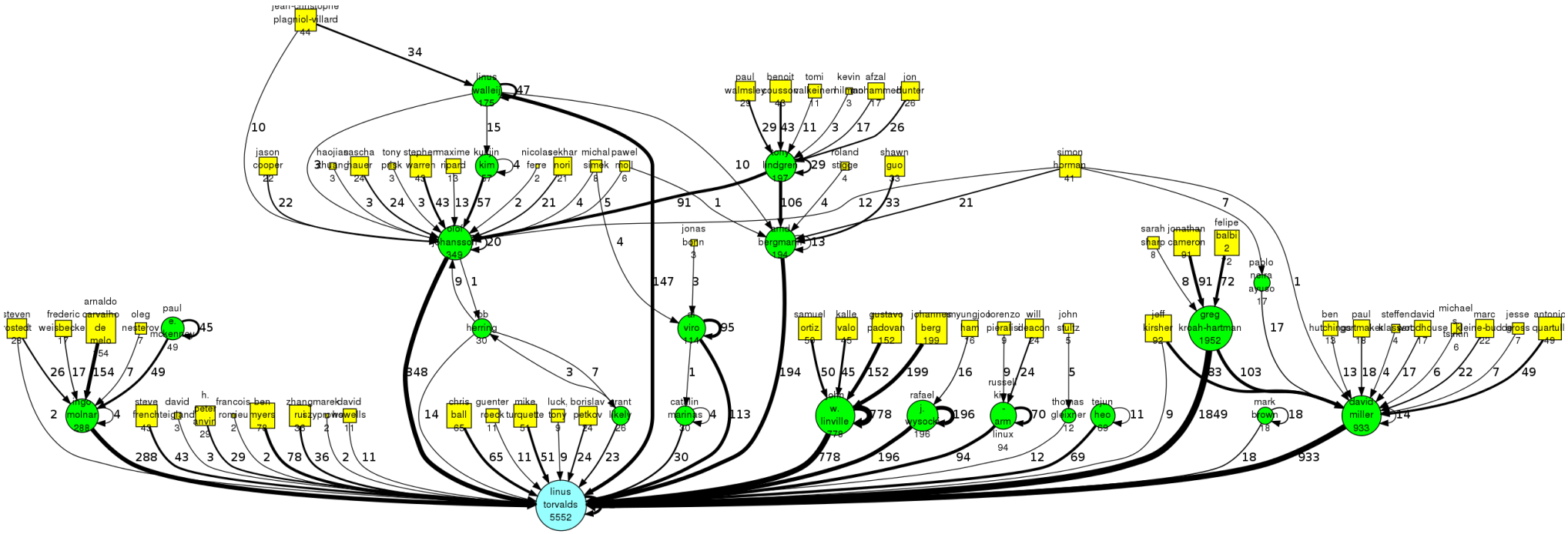
Time of Authorship



Time of Commit

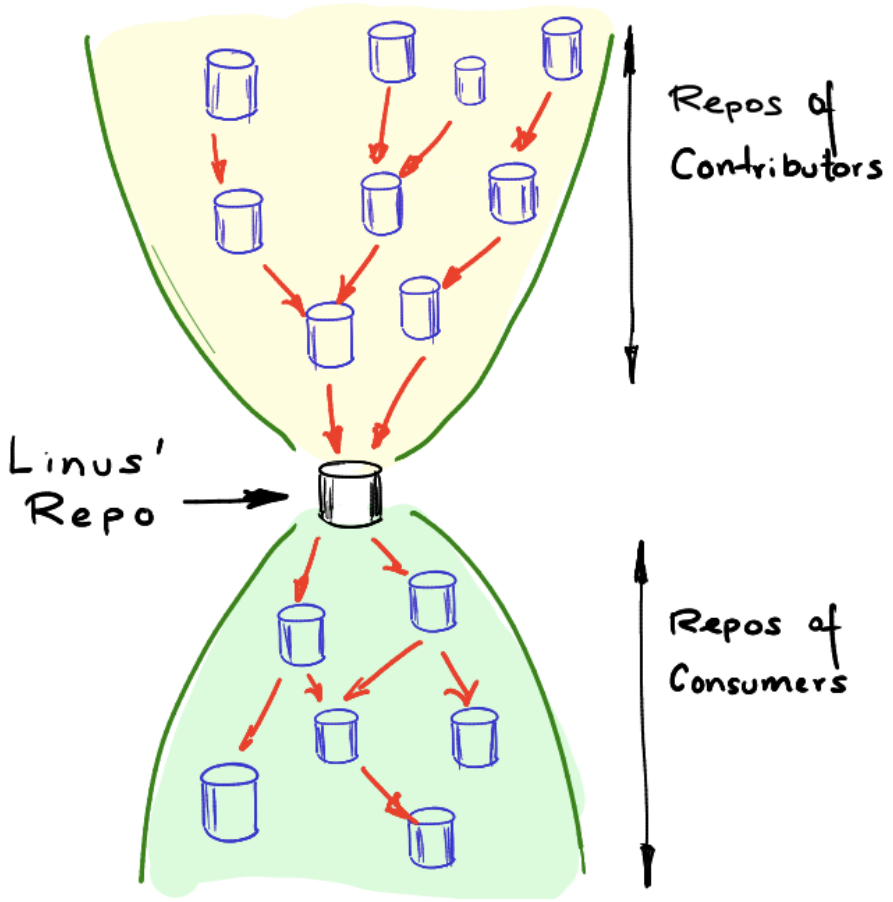
Latency

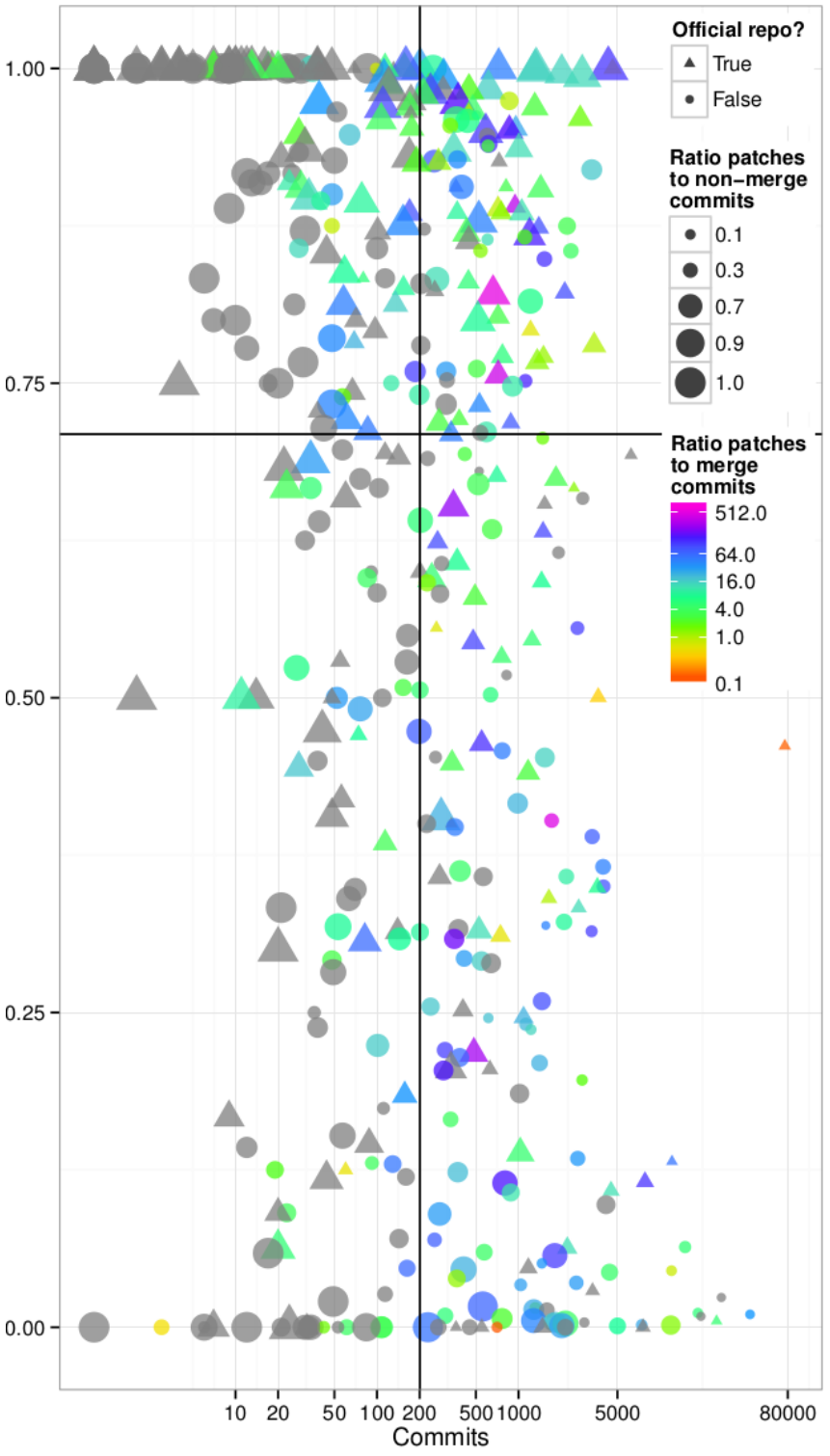

variable
latencyautdate
latencycomdat
Days
1000
750
500
250
0
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52
weeklinus


jean-christophe plagniol-villard 44
34
linus wallen 175
47
10
15
paul walmsley 23
benoit cousson 48
tomi walkeinen 11
kevin hilman 3
afzal mohammed 17
jon hunter 26
29 43 11 3 17 26
jason cooper 22
haojian zhuang 3
ascha hauer 24
tony prodyz 3
stephen warren 43
maxime ripard 13
kukjin kim 57
4
nicolas ferre 2
sekhar nori 21
michal simek 8
pawel moll 6
roland stige 4
shawn guo 33
simon horman 41
10
29
22
3
24
3
43
13
57
2
21
4
5
91
1
106
4
12
33
21
7
arnd bergmann 194
13
20
jonas bon 3
sarah sharp 8
jonathan cameron 91
felipe balbi 2
pablo neira ayuso 17
8
91
72
1
4
147
3
9
1
348
194
50
45
152
199
16
9
24
5
8
91
72
17
26
17
154
7
49
2
4
288
43
3
29
2
78
36
2
11
14
65
11
51
9
24
23
30
778
196
94
12
69
18
933
1849
83
103
13
18
4
17
6
22
7
49
steven rostedt 28
frederic weisbecker 17
arnaldo carvalho de melo 154
oleg nesterov 7
paul e. mckenney 49
45
ingo molnar 288
4
steve french 43
david teigland 3
h. peter anvin 25
francois romieu 2
ben myers 73
zhang rusty russell 36
marek szyprowski 2
david wells 11
chris ball 65
guenter roeck 11
mike turquette 51
luck, tony 9
boris petkov 24
grant likely 26
catalin marinas 30
viro 141
95
113
samuel ortiz 50
kalle valo 45
gustavo padovan 152
johannes berg 199
myungjoo ham 76
lorenzo pieralisi 9
will deacon 24
john stultz 5
w. linville 778
j. wysocki 196
rafael j. wysocki 196
russell king 70
arm linux 94
thomas gleixner 12
heo 11
jeff kirsher 92
greg kroah-hartman 1952
mark brown 18
18
ben hutchings 15
paul mackerras 18
steffen klassert 7
david woodhouse 17
michael s. tsirkin 6
marc kleine-budde 22
jesse gross 7
antonio quartulli 49
davem miller 933
14
linus torvalds 5552

The Linux Super-repository

- Producers
- Consumers

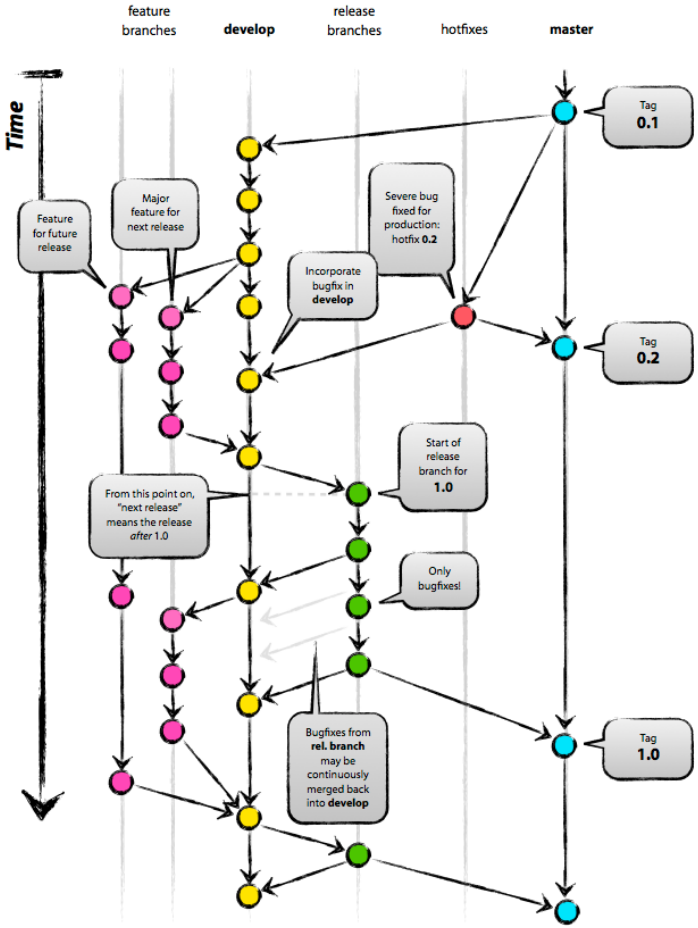




Final observations

Super-repository

- Collection of **repositories cloned** (recursively) from the same repo



The paradox

Distributed Version control allows anybody to participate…

but it centralizes control on who contributes

Linux is not only software...

it is also software engineering

Git promotes Linux practices:

Git promotes Linux practices:

- code reviews

Git promotes Linux practices:

- code reviews
- pull requests

Git promotes Linux practices:

- code reviews
- pull requests
- isolation

Git promotes Linux practices:

- code reviews
- pull requests
- isolation

and github popularizes them

GitHub for non-technical people

GitHub

GitHub for the rest of us

contributions welcome

Credit: Flickr/Nick Quaranto

ROBERT MCMILLAN BUSINESS 09.02.13 6:30 AM

FROM COLLABORATIVE CODING TO WEDDING INVITATIONS: GITHUB IS GOING MAINSTREAM

Git made it possible for programmers to coordinate distributed work across teams -- now GitHub makes it possible for everyone else