Git: *Distributed* Version Control

Lecture 3

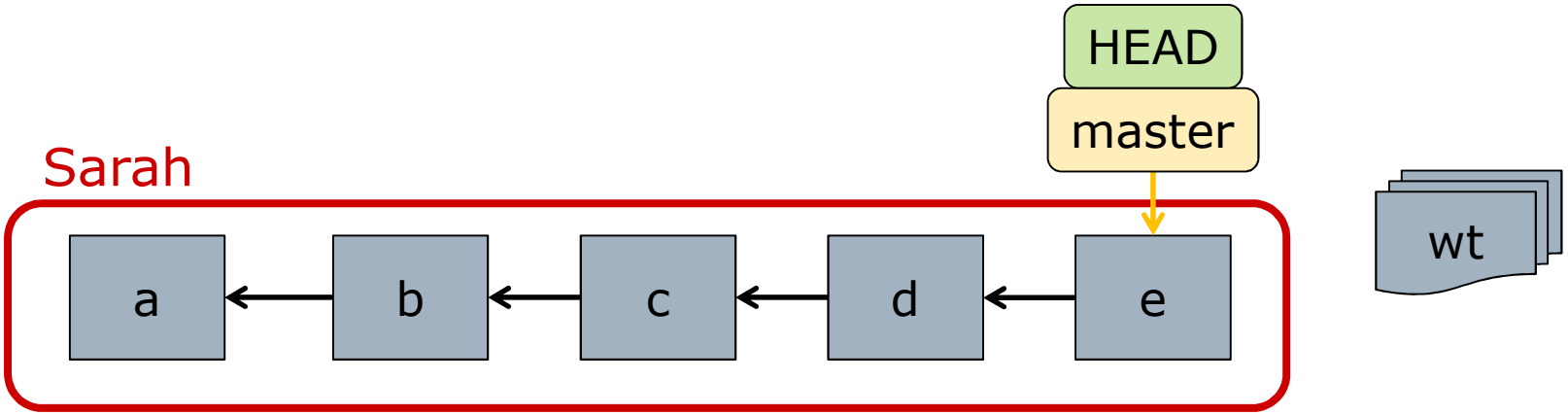
Demo

- Prep: Empty (but initialized) repo
- Linear development:
  - Create, edit, rename, ls -la files
  - Git: add, status, commit, log
- Checkout (time travel, detach HEAD)
- Branch (re-attach HEAD)
- More commits, see split in history
- Merge
  - No conflict
  - Fast-forward

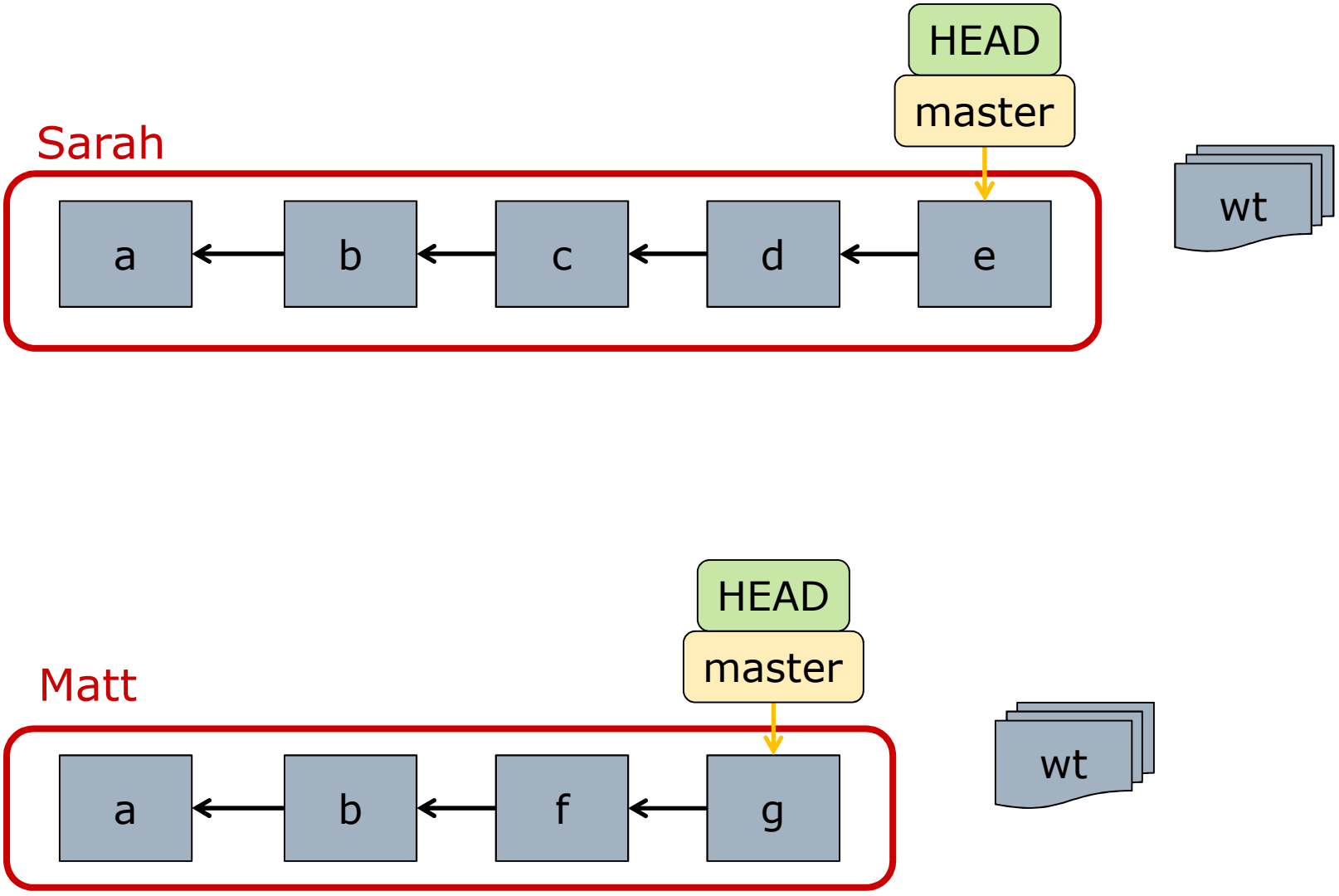
What Does "D" Stand For?

- *Distributed* version control
  - Multiple people, distributed across network
- Each person has their own repository!
  - Everyone has their own store (history)!
  - Big difference with older VCS (eg SVN)
- Units of data movement: changeset
  - Communication between teammates is to bring *stores* in sync
  - Basic operators: fetch and push

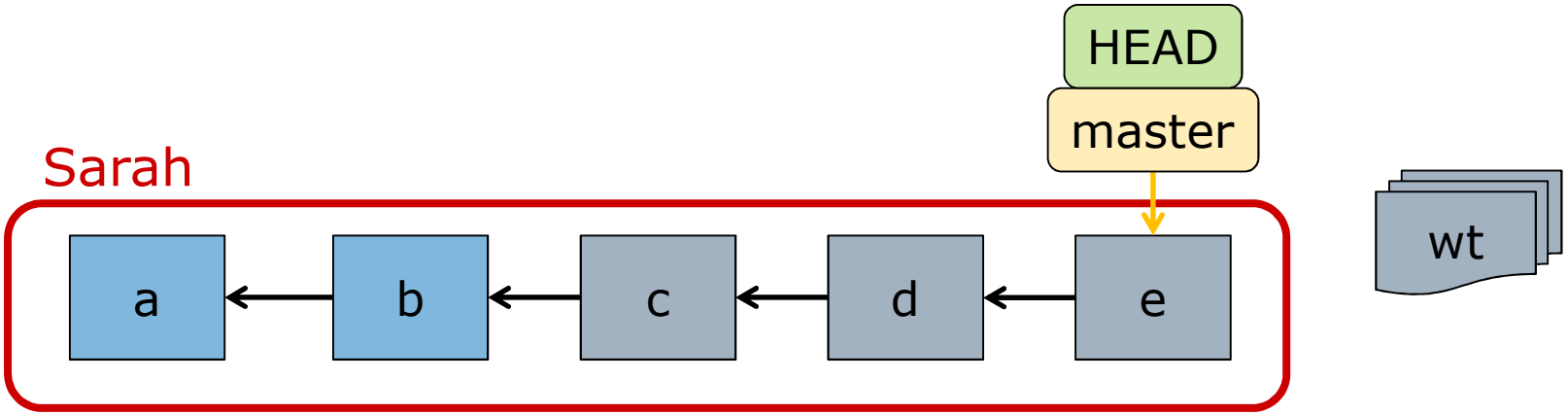
Sarah's Repository

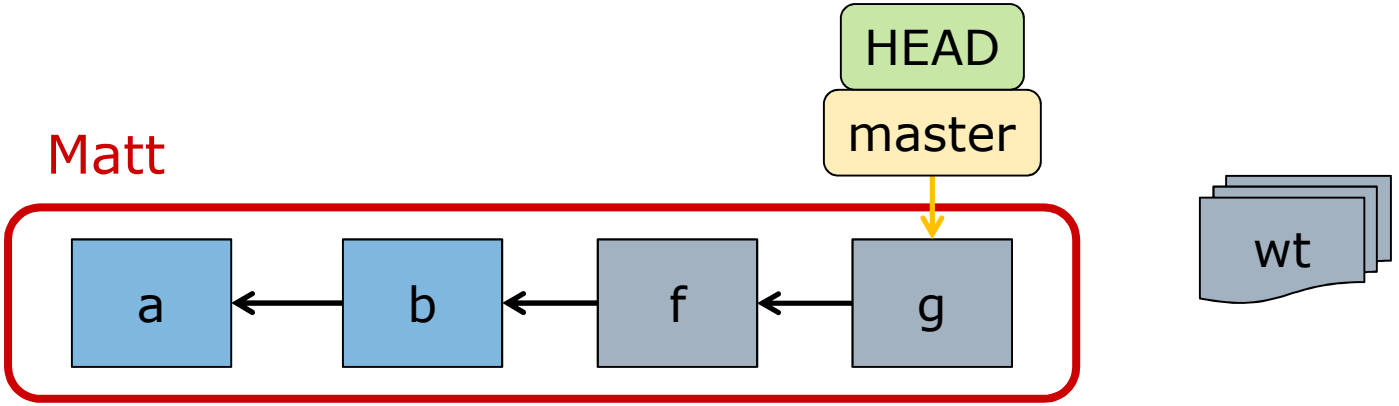


And Matt's Repository

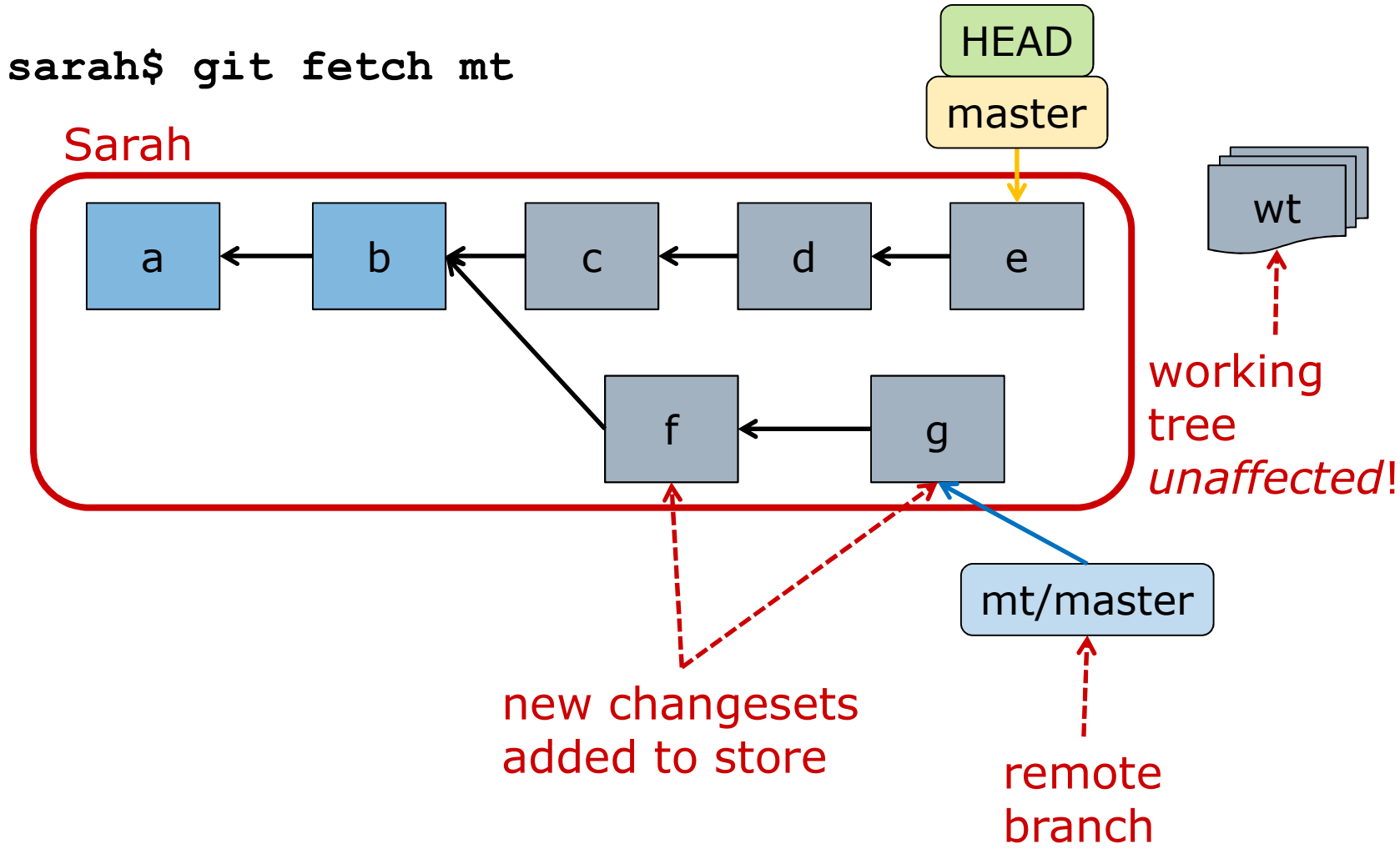


Some Shared History

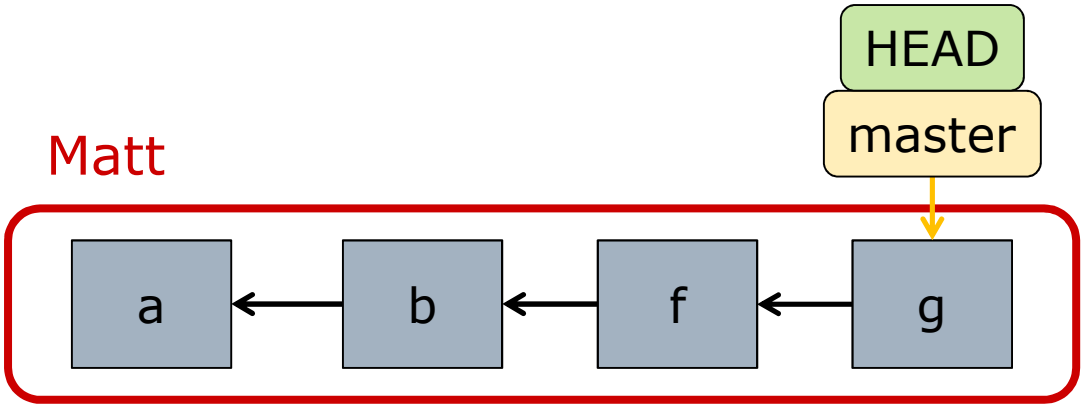




Fetch: Remote Store → Local

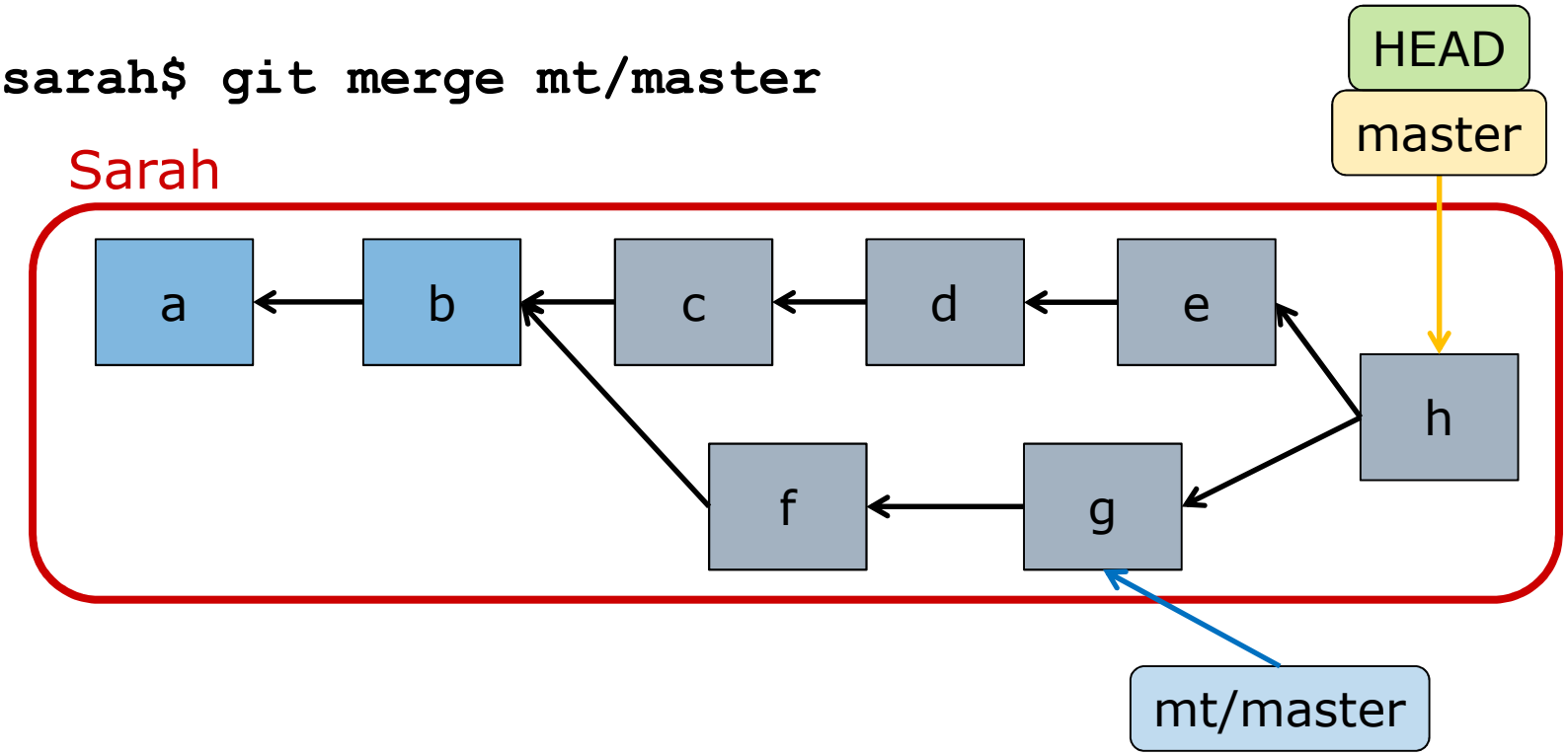


Remote Repository Unchanged

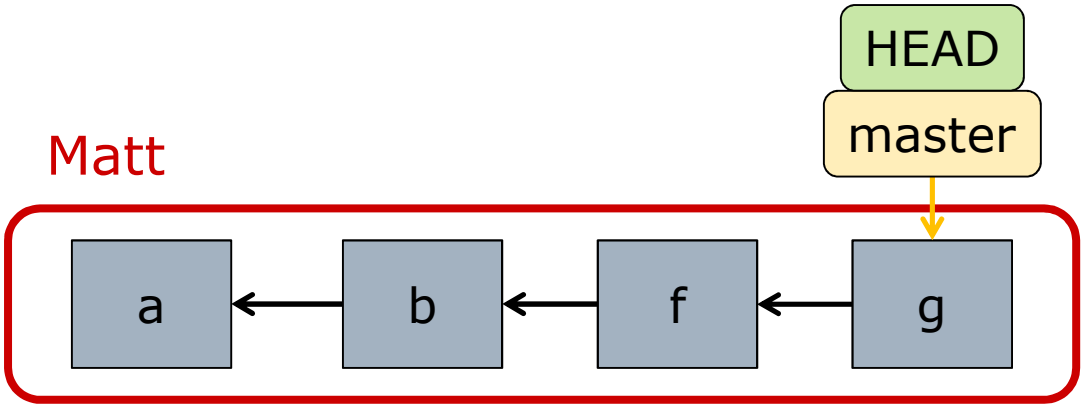


Workflow: Merge After Fetch


```
sarah$ git merge mt/master
```



Remote Repository Unchanged



View of DAG with All Branches

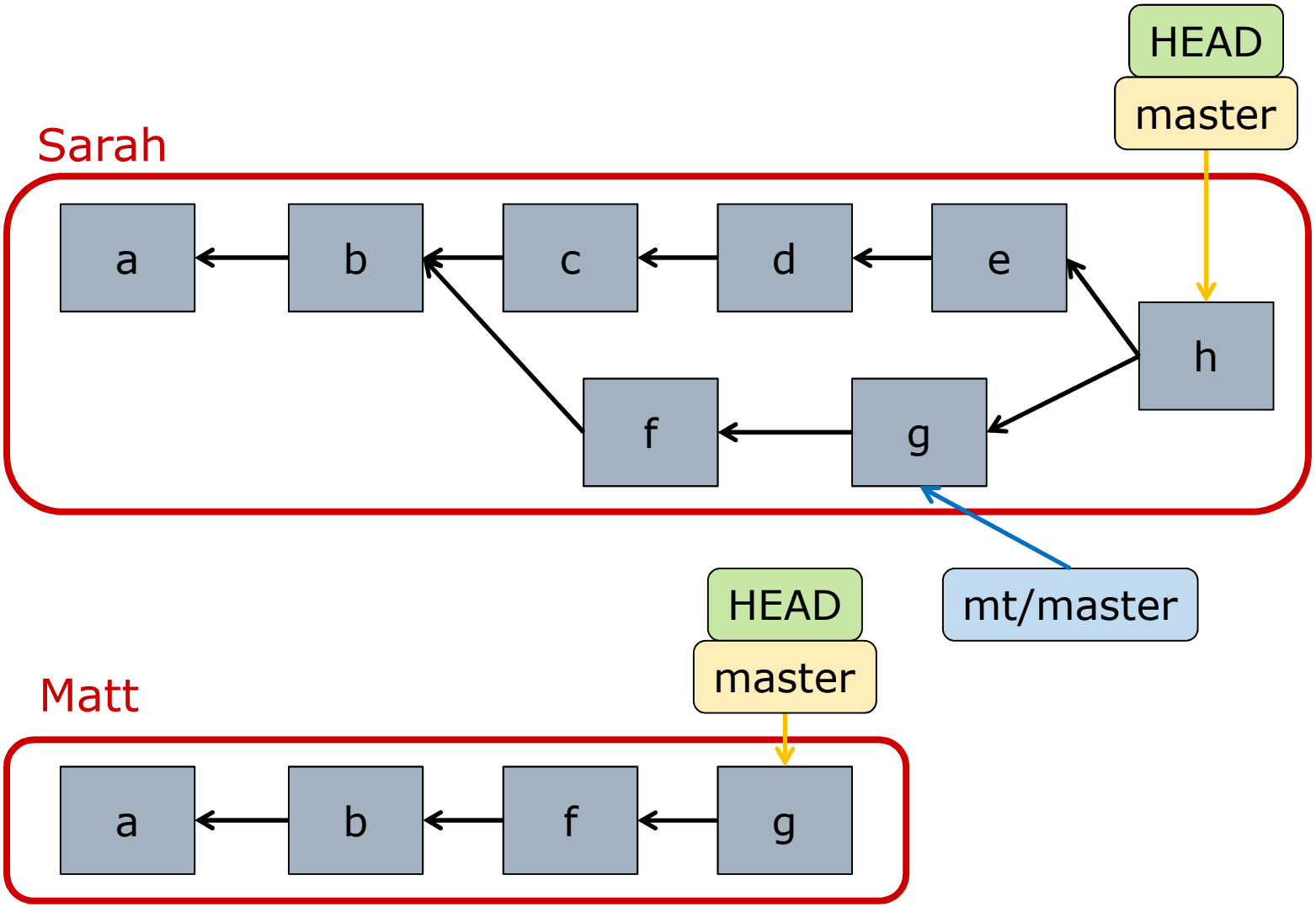

```
$ git log --oneline --graph --decorate --all

* 1618849 (HEAD-> master, origin/master) clean up css
*   d579fa2 (alert) merge in improvements from master
|\
| * 0f10869 replace image-url helper in css
* | b595b10 (origin/alert) add buckeye alert notes
* | a6e8eb3 add raw buckeye alert download
|/
* b4e201c wrap osu layout around content
* e9d3686 add Rakefile and refactor schedule loop
* 515aaa3 create README.md
* eb26605 initial commit
```

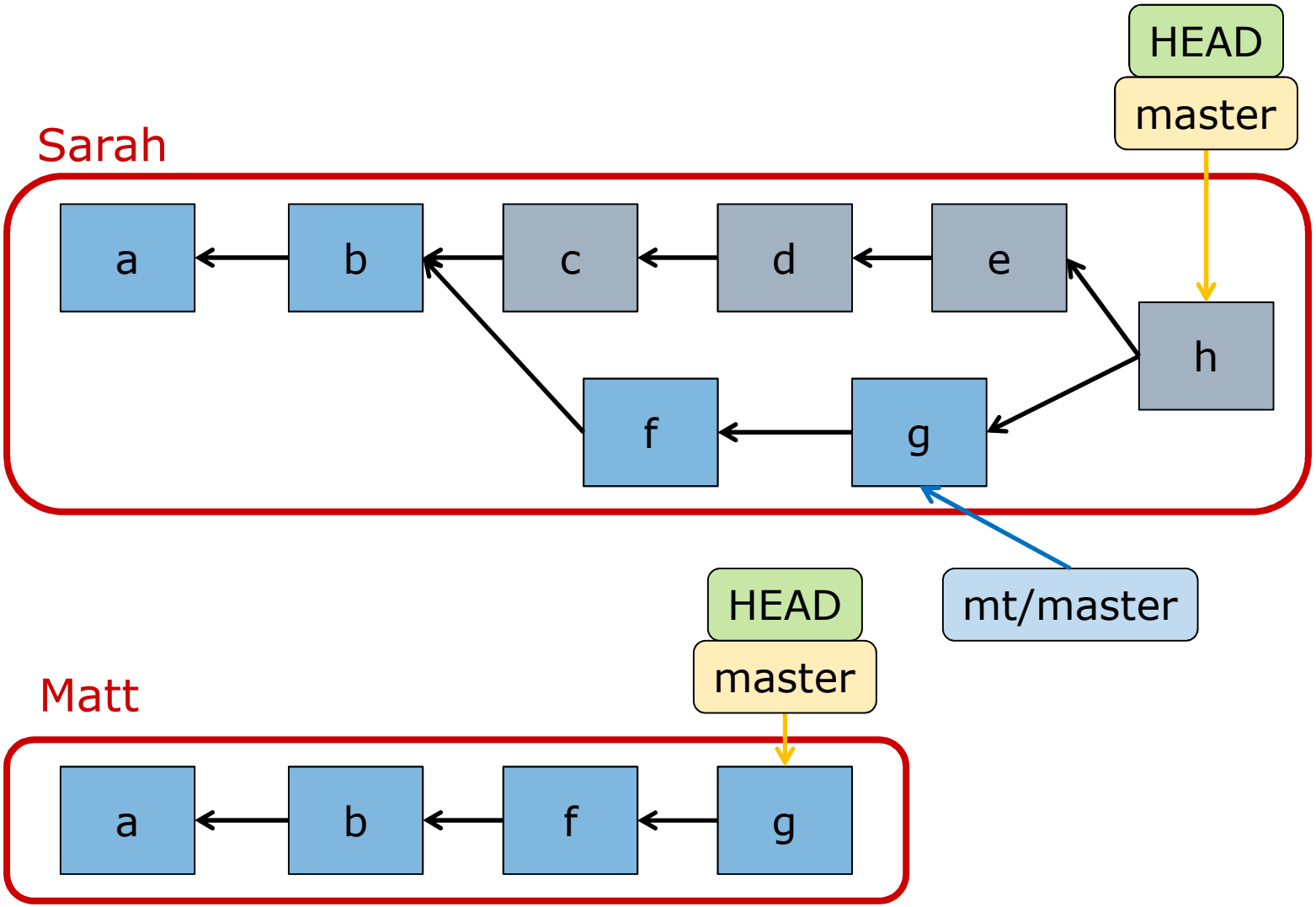
Your Turn

- Show the state of Matt's repository after each of the following steps
  - Fetch (from Sarah)
  - Merge

Sarah and Matt's Repositories



Some Shared History



Your Turn: Fetch


```
matt$ git fetch sr
```

Your Turn: Merge


```
matt$ git merge sr/master
```

Pull: Fetch then Merge

- A "pull" combines both fetch & merge

```
matt$ git pull sr
```

- Advice: Prefer explicit fetch, merge
  - After fetch, examine new work

```
$ git log --all #see commit messages
$ git checkout #see work
$ git diff #compare
```

  - Then merge
  - Easier to adopt more complex workflows (*e.g.*, rebasing instead of merging)

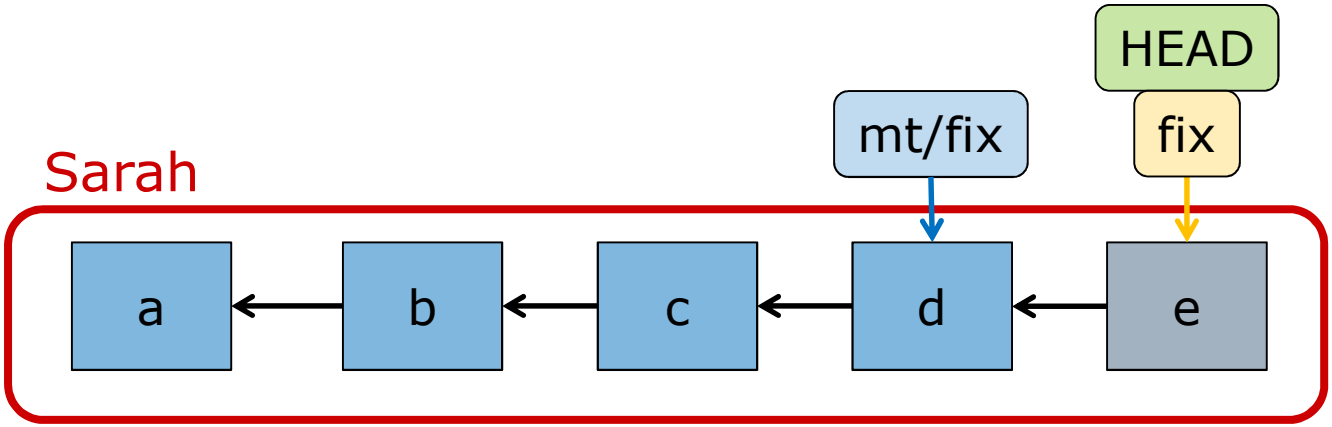
Push: Local Store → Remote

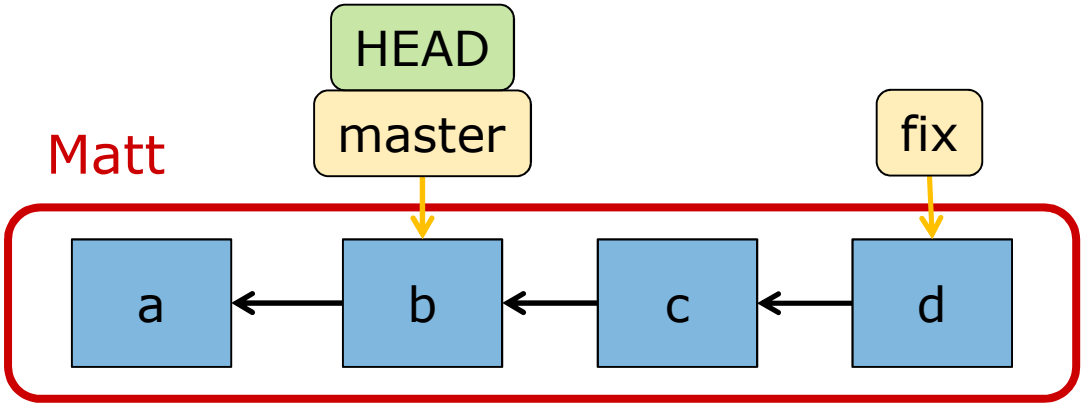
- Push sends local commits to remote store
- Usually push one branch (at a time)

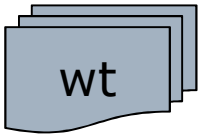
```
sarah$ git push mt fix
```

  - Advances Matt's fix branch
  - Advances Sarah's mt/fix remote branch
- Requires:
  1. Matt's fix branch *must not* be his HEAD
  2. Matt's fix branch *must be* ancestor of Sarah's
- Common practices:
  1. Only push to *bare* repositories (bare means no working tree, ie no HEAD)
  2. Get remote store's branch into local DAG (ie fetch, merge, commit) *before* pushing

Remote's Branch is Ancestor

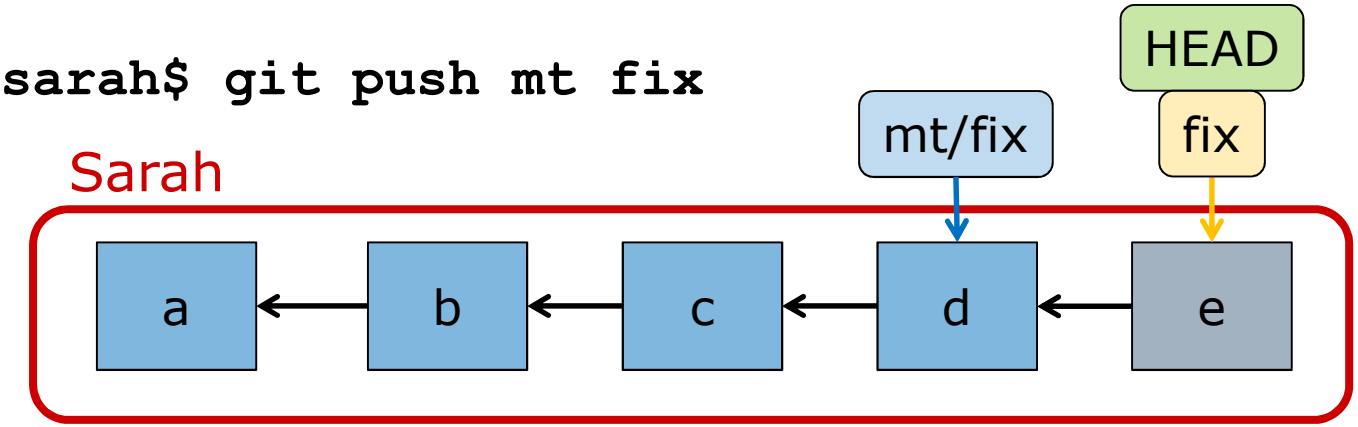


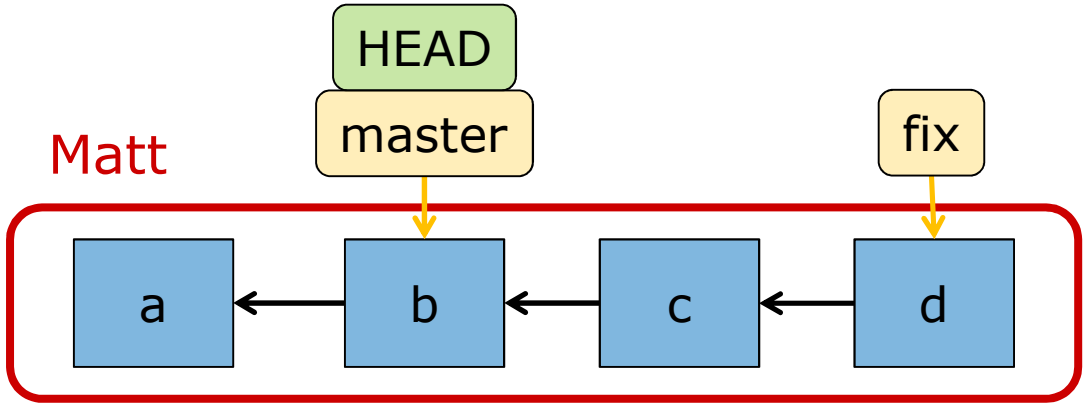


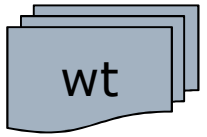


Push: Local Store → Remote


```
sarah$ git push mt fix
```

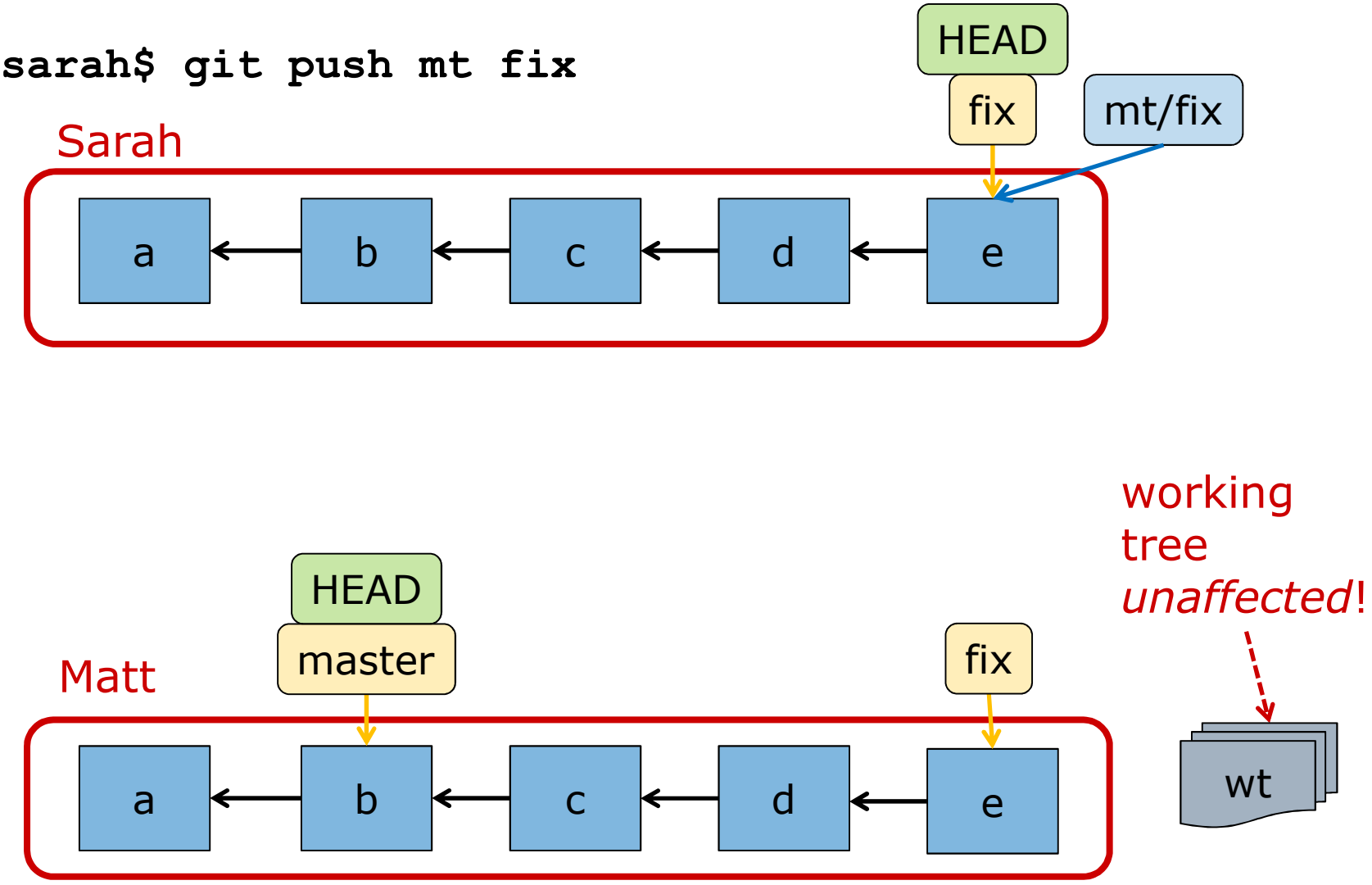




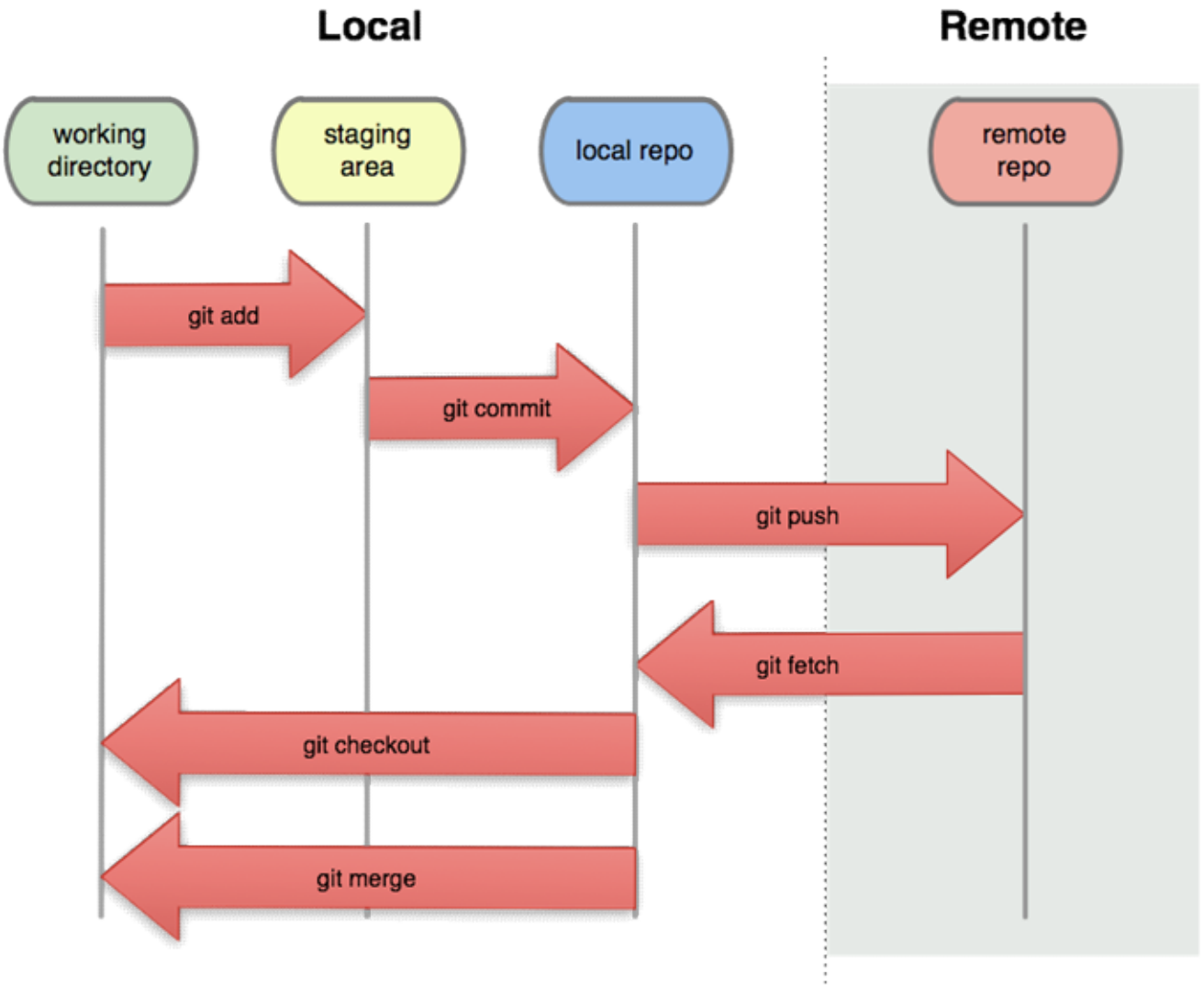


Push: After


```
sarah$ git push mt fix
```



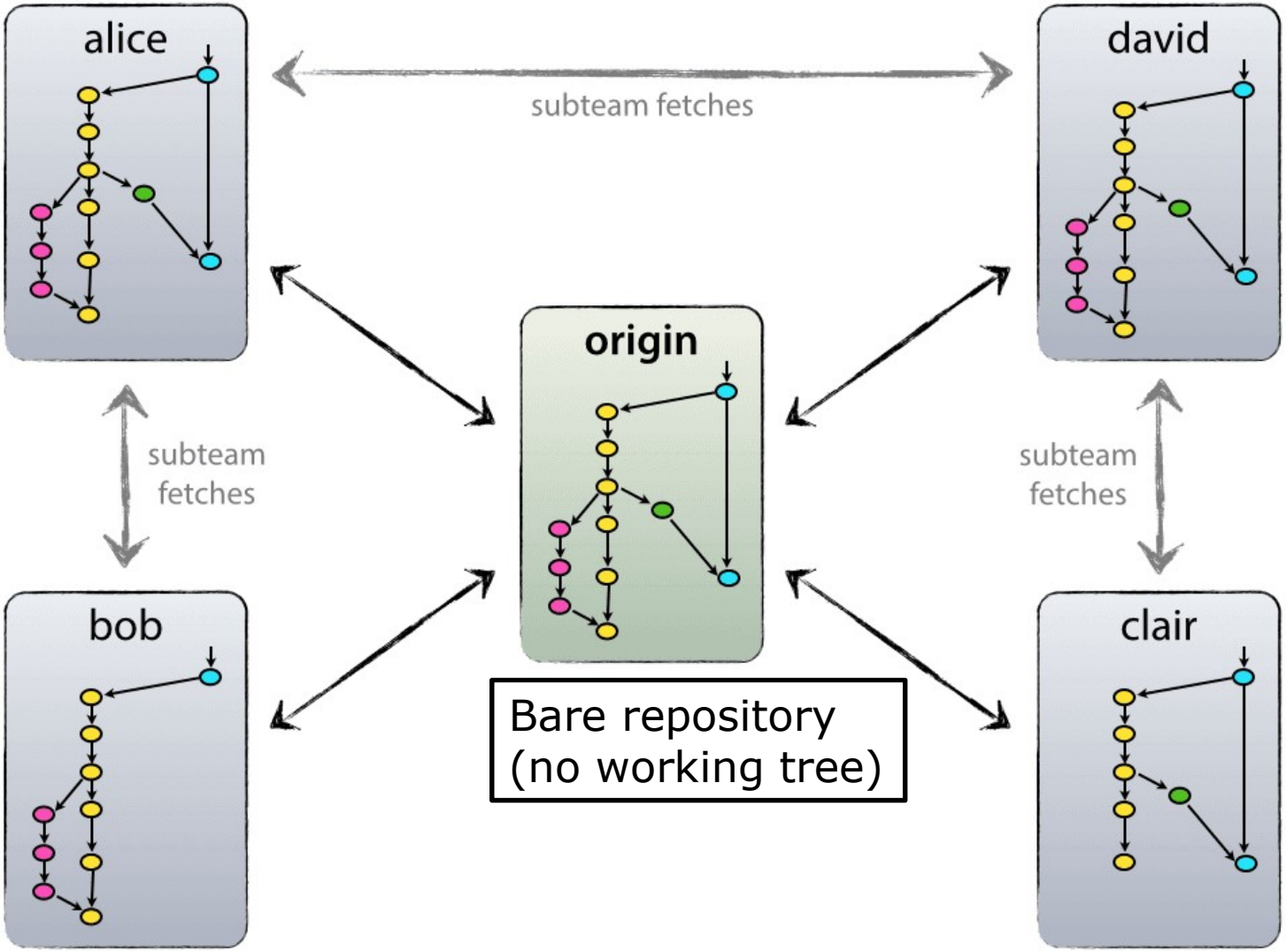
Commit/Checkout vs Push/Fetch



Common Topology: Star

- *n*-person team has *n+1* repositories
  - 1 shared central repository (bare!)
  - 1 local repository / developer
- Each developer *clones* central repository
  - Cloning creates a remote called "origin"
  - Default source/destination for fetch/push
- Variations for central repository:
  - Everyone can read and write (ie push)
  - Everyone can read, but only 1 person can write (responsible for pulling and merging)

Common Topology: Star



Source: http://nvie.com/posts/a-successful-git-branching-model/

Summary

- Push/fetch to share your store with remote repositories
  - Neither working tree is affected
- Branches in history are easy to form
  - Committing when HEAD is not a leaf
  - Fetching work based on earlier commit
- Team coordination
  - One single, central repo
  - Every developer pushes/fetches from their (local) repo to this central (remote) repo