Software and systems engineering

Paulo Borba
Informatics Center
Federal University of Pernambuco

phmb@cin.ufpe.br ◈ twitter.com/pauloborba

To do before class

- Watch video
- Read related part of chapter 10 in the textbook
- Send questions and opinions through slack

Software and systems engineering

Paulo Borba
Informatics Center
Federal University of Pernambuco

phmb@cin.ufpe.br ◈ twitter.com/pauloborba

Configuration management 1, basic concepts and operations

Software and system development is often done in teams!

Team members contribute to the same artifacts

Productivity and quality problems sharing requirements, source code, etc. via...

- email
- dropbox
- google docs

Source code (artifact) control systems

Version Control purposes (De Rosso & Jackson)

- Data management: make a set of changes persistent
- Change management: group logically related changes, record coherent points
- Collaboration: synchronise changes
- Parallel development: isolated lines of development
- Disconnected operation

Purposes, Concepts, Misfits, and a Redesign of Git

Santiago Perez De Rosso   Daniel Jackson
Computer Science and Artificial Intelligence Lab
Massachusetts Institute of Technology
Cambridge, MA, USA
{sperezde, dnj}@csail.mit.edu

Abstract

Git is a widely used version control system that is powerful but complicated. Its complexity may not be an inevitable consequence of its power but rather evidence of flaws in its design. To explore this hypothesis, we analyzed the design of Git using a theory that identifies concepts, purposes, and misfits. Some well-known difficulties with Git are described, and explained as misfits in which underlying concepts fail to meet their intended purpose. Based on this analysis, we designed a reworking of Git (called Gitless) that attempts to remedy these flaws.

To correlate misfits with issues reported by users, we conducted a study of Stack Overflow questions. And to determine whether users experienced fewer complications using Gitless in place of Git, we conducted a small user study. Results suggest our approach can be profitable in identifying,

First, we did an analysis of the product, in which we applied some new design principles [16] in an attempt to identify problematic aspects of the design, suggesting respects in which the design might be improved. Since any such analysis is likely to be influenced by subjective factors (not least our own experiences using the product, and the particular contexts in which we used it), we corroborated the analysis by examining a large number of posts in a popular Q&A forum, to determine whether the issues we identified were in fact aligned with those that troubled other users.

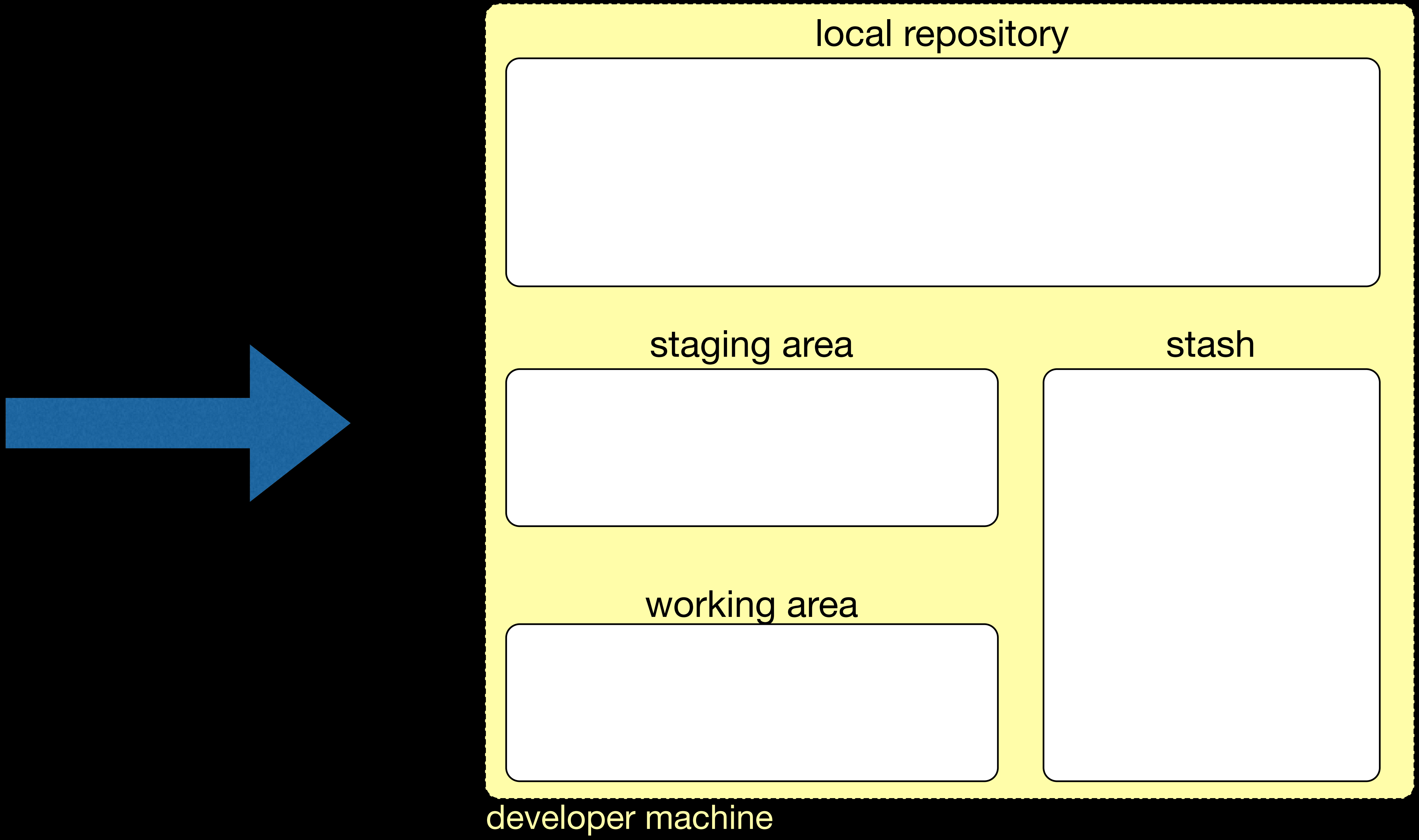
Second, we reworked the design to repair the deficiencies identified by our analysis, and implemented the new design. To evaluate the redesign, we conducted a user study in which users with a range of levels of expertise were asked to complete a variety of tasks using the existing and new product. We measured the time they took, and obtained feedback on

Overview of main concepts

Creating a local repository

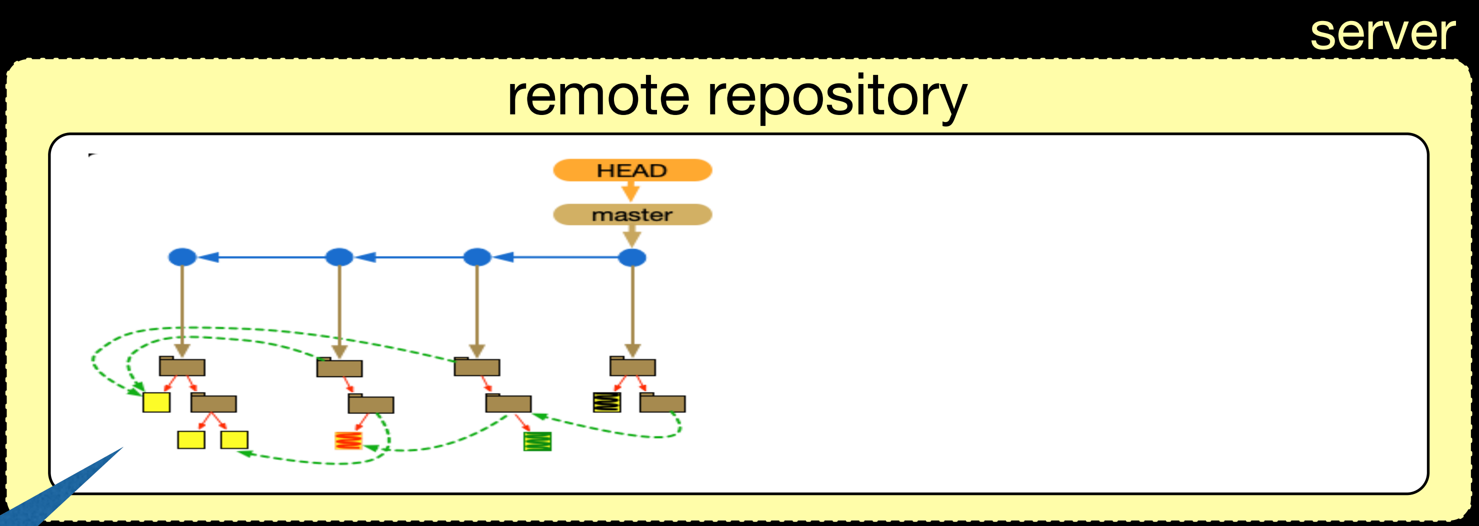
From scratch

git init

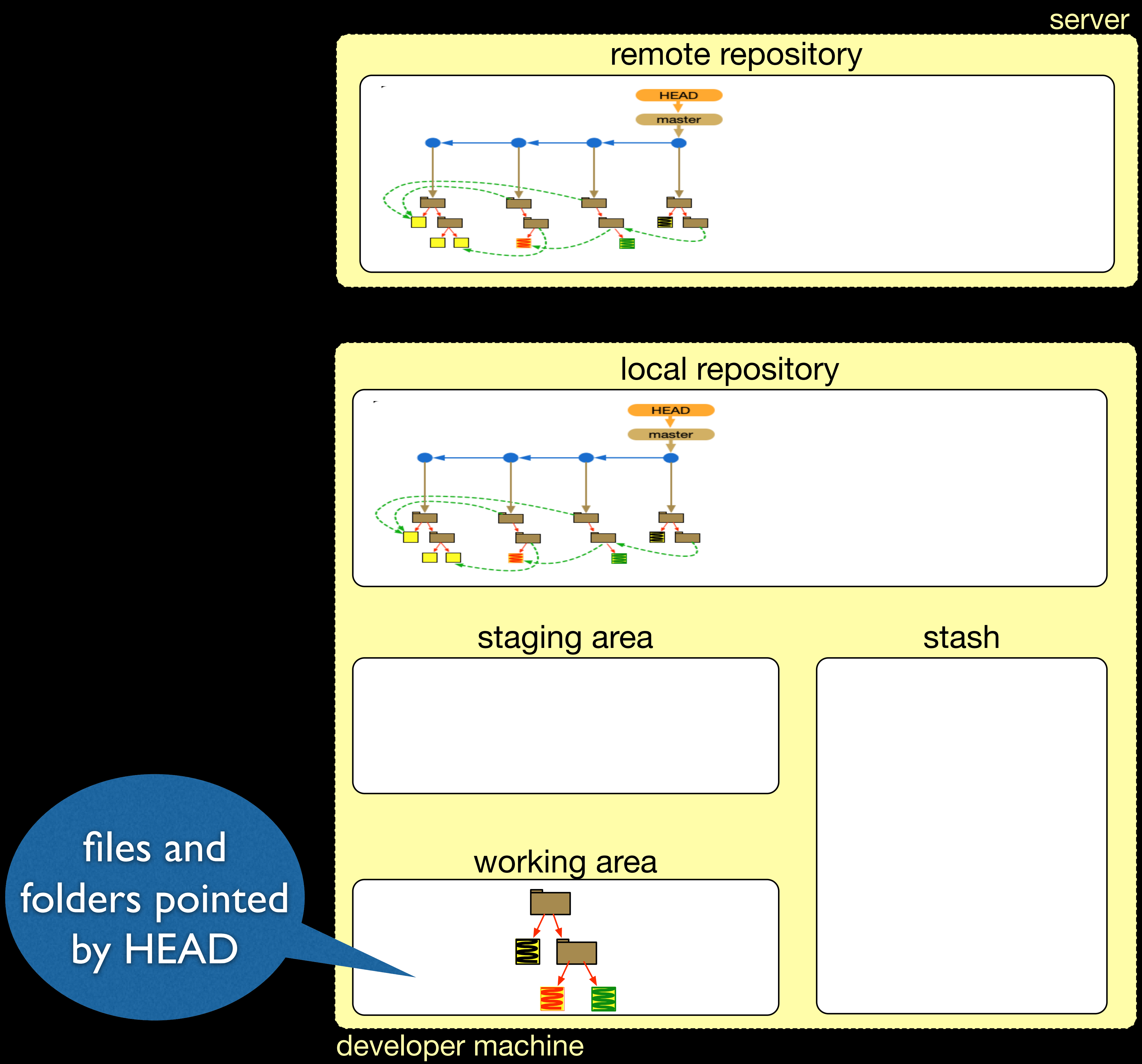

local repository
staging area
stash
working area
developer machine

From an existing repository

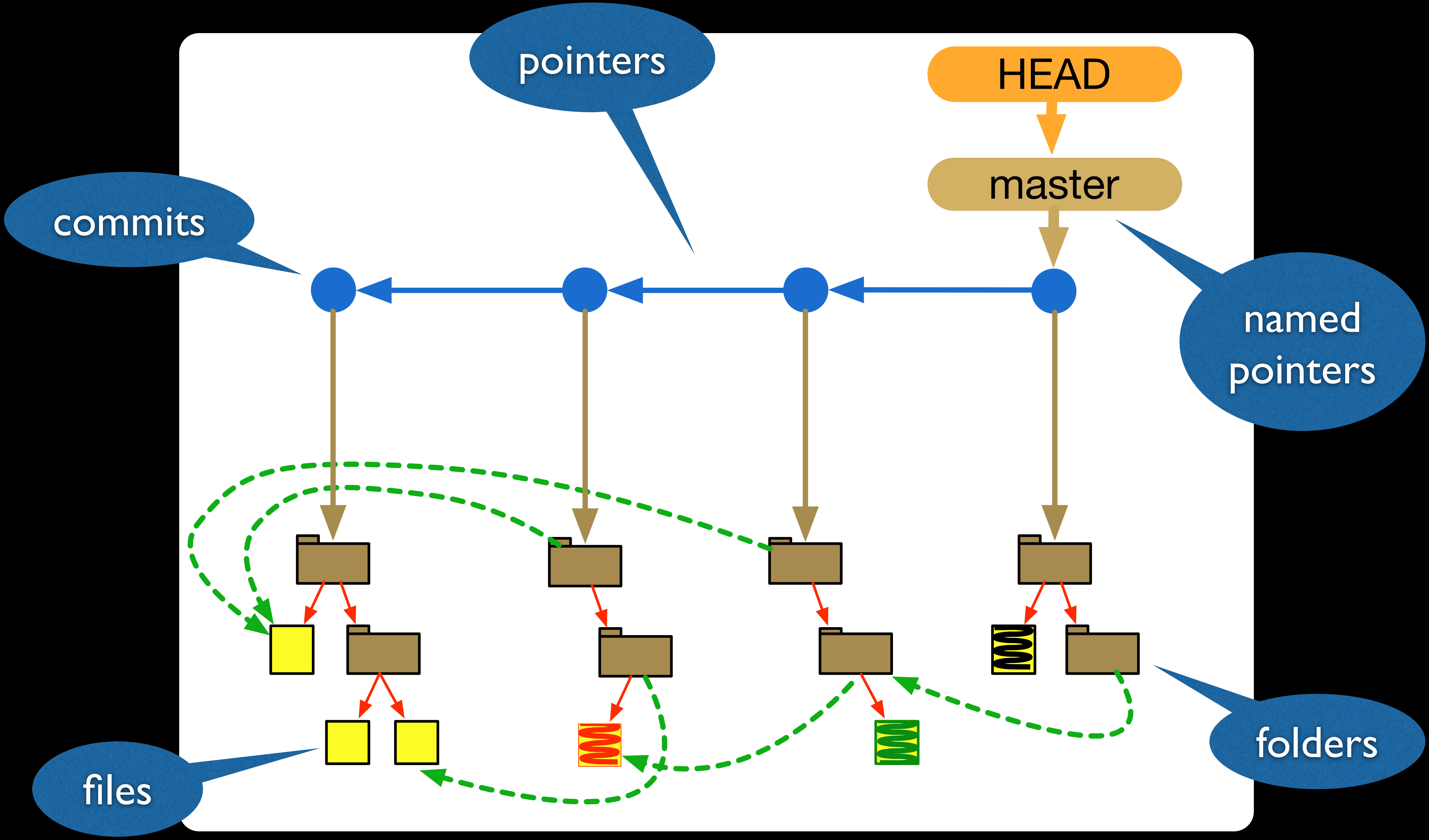
git clone



project history

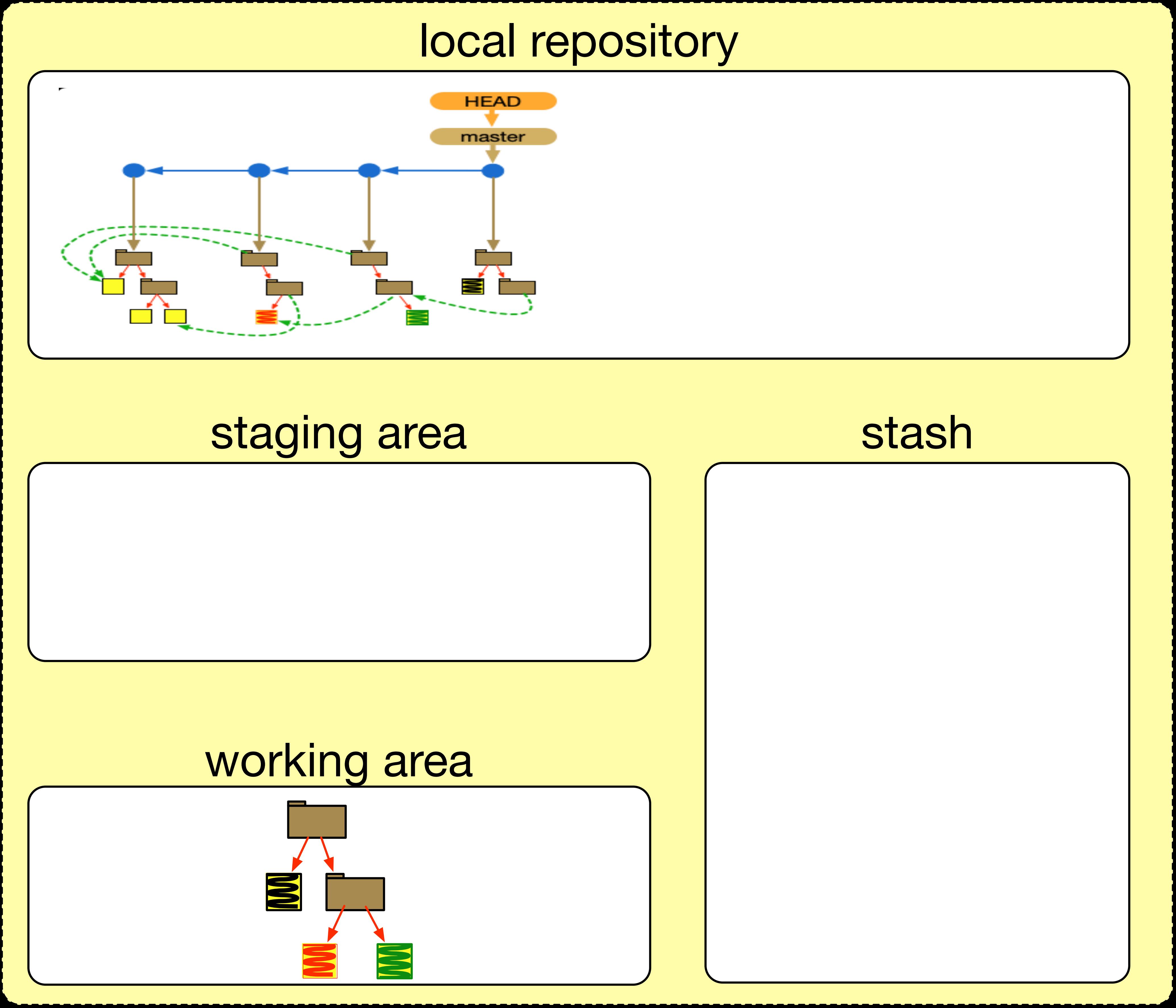

server
remote repository
HEAD
master
local repository
HEAD
master
staging area
stash
working area
files and folders pointed by HEAD
developer machine

Commit graph

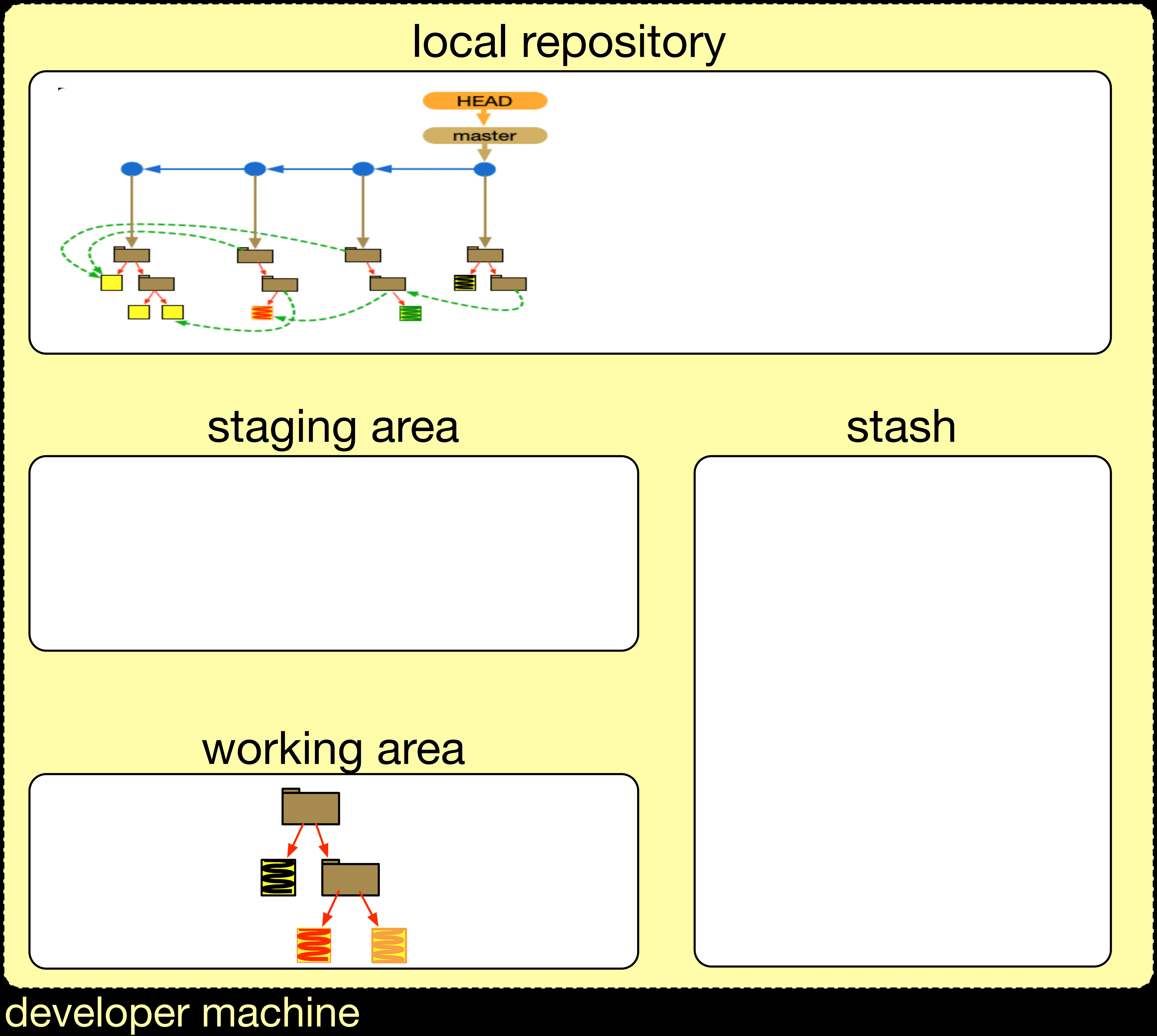


Persisting and recording changes

Locally changing files

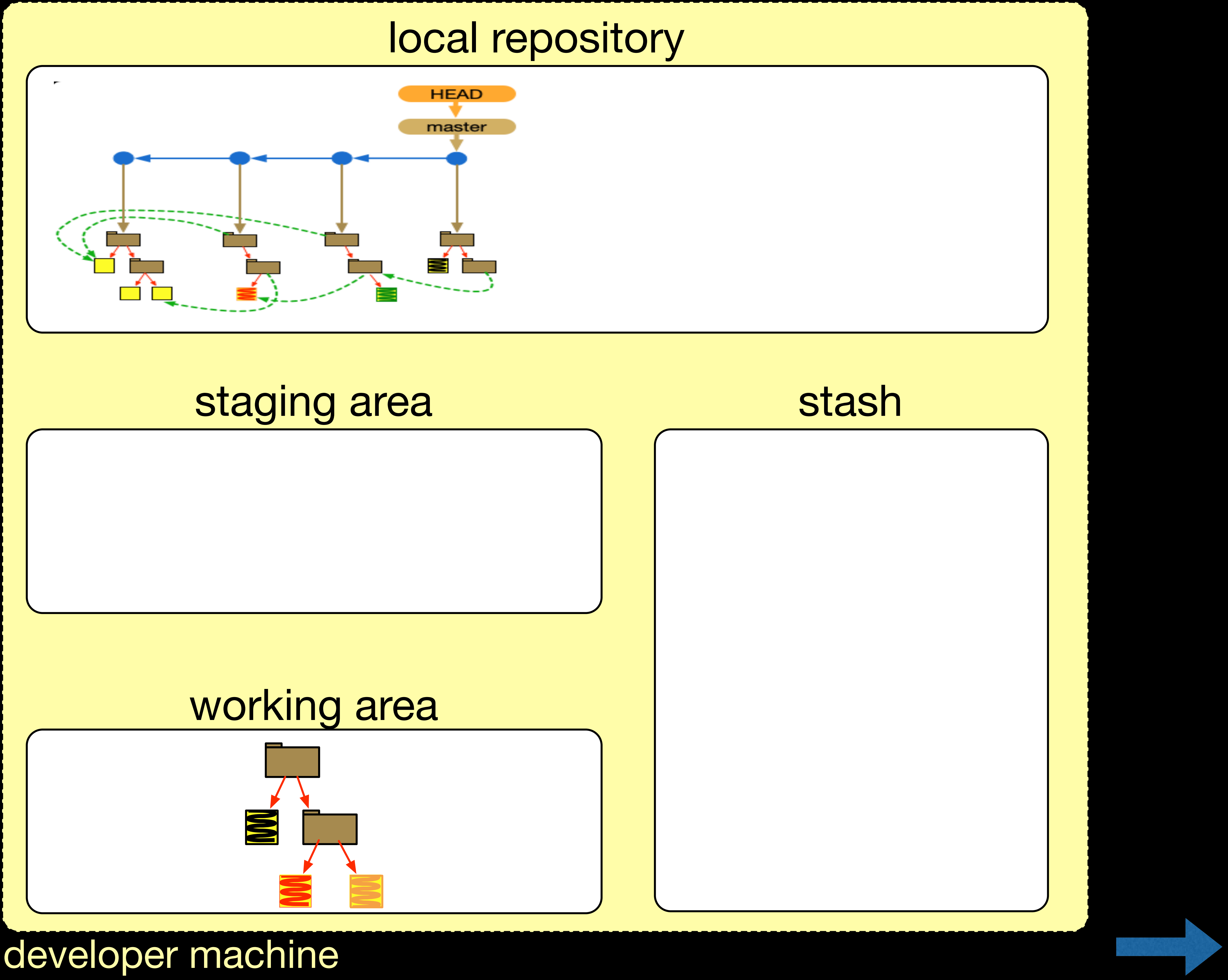


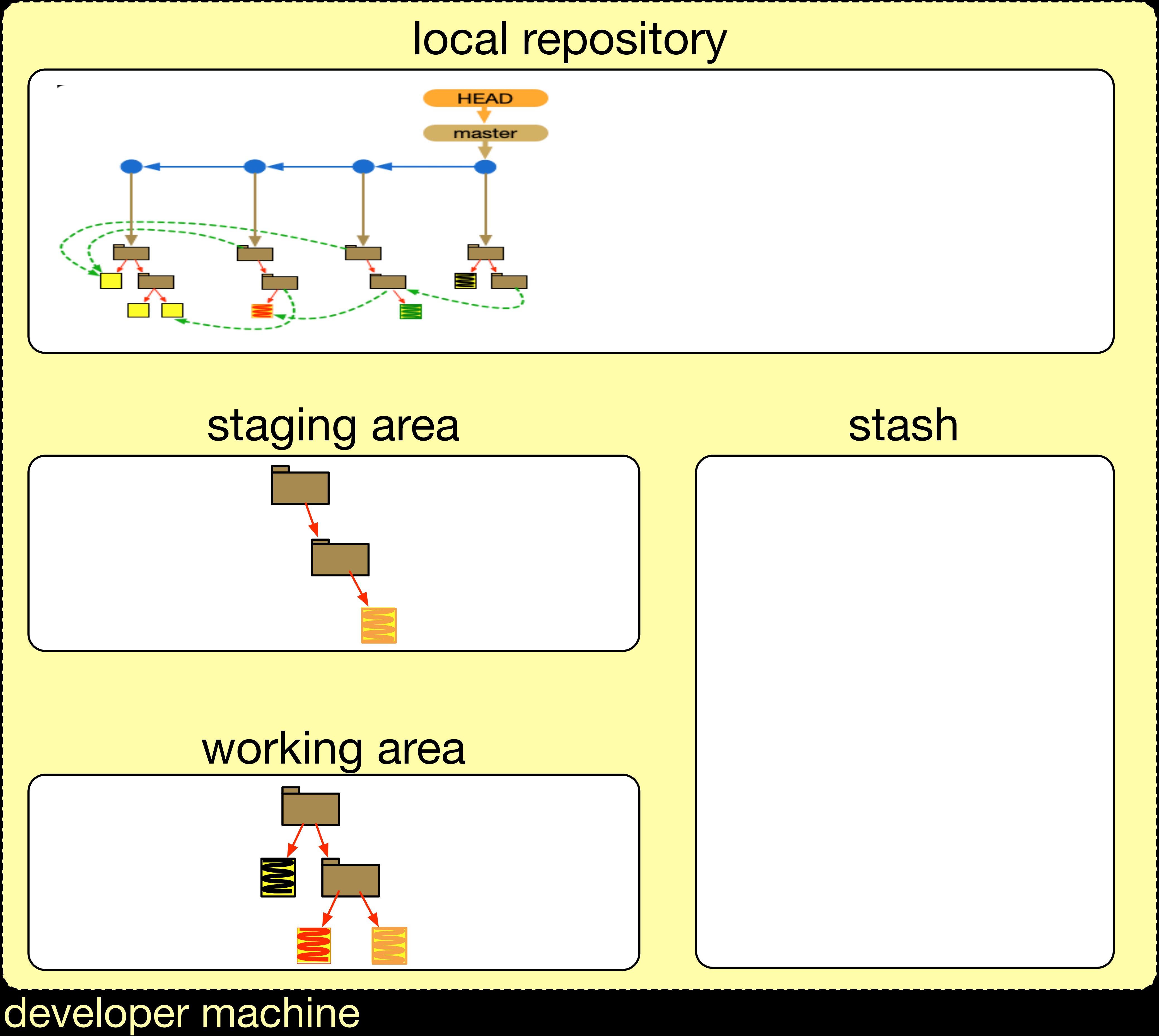
developer machine


local repository
HEAD
master
staging area
stash
working area
developer machine

git add

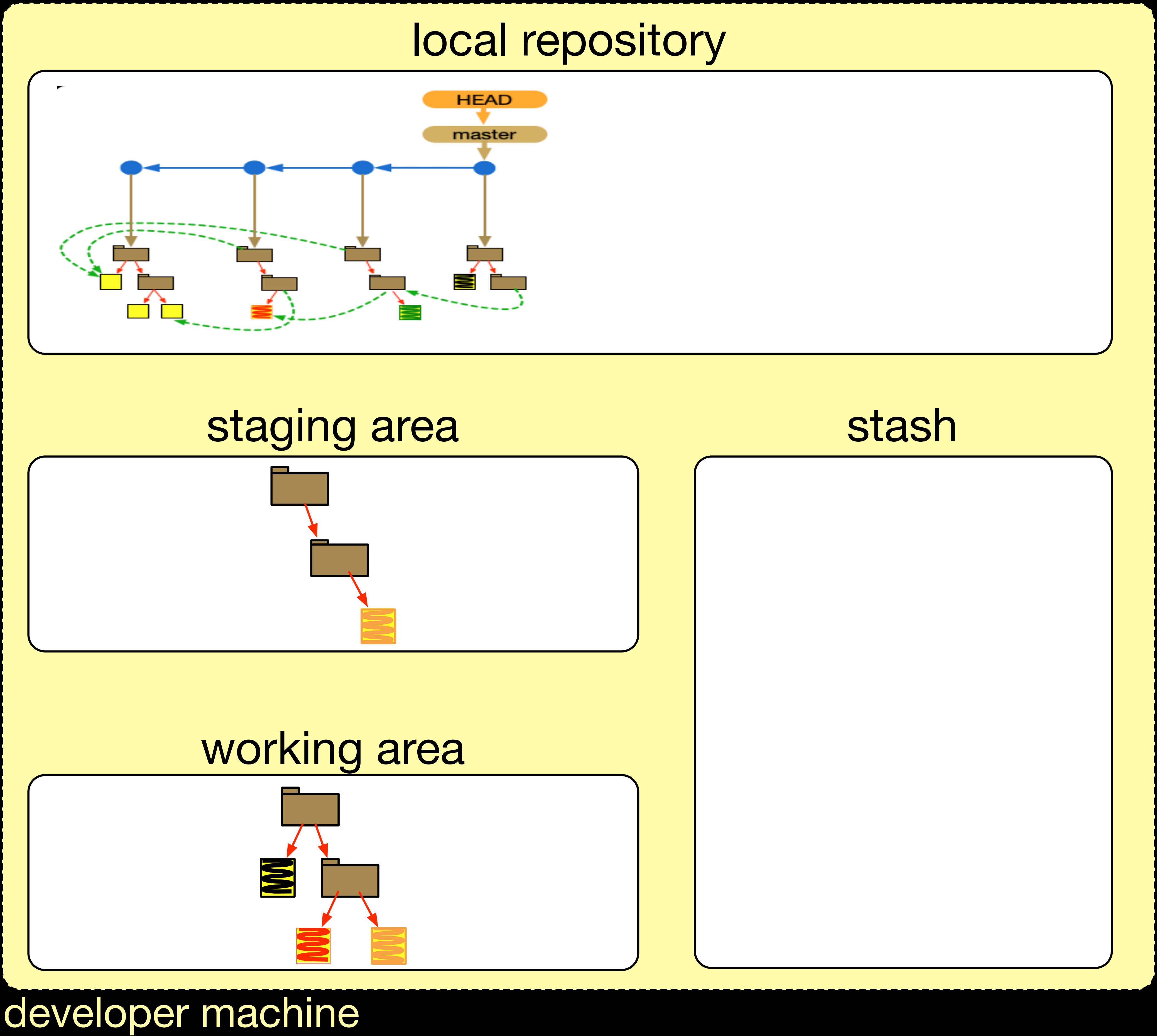
indicating that files are
of interest

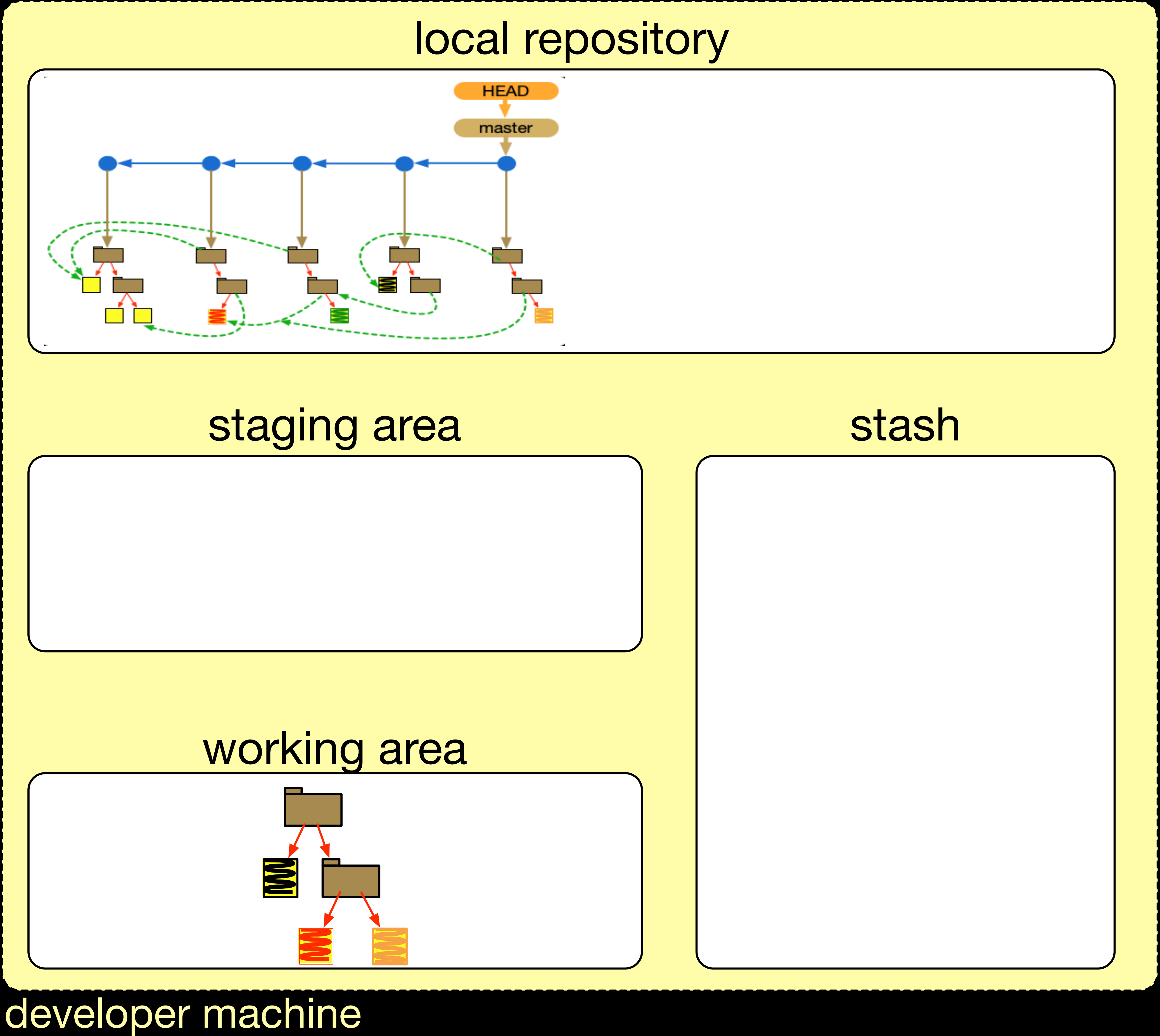

local repository
HEAD
master
staging area
stash
working area
developer machine

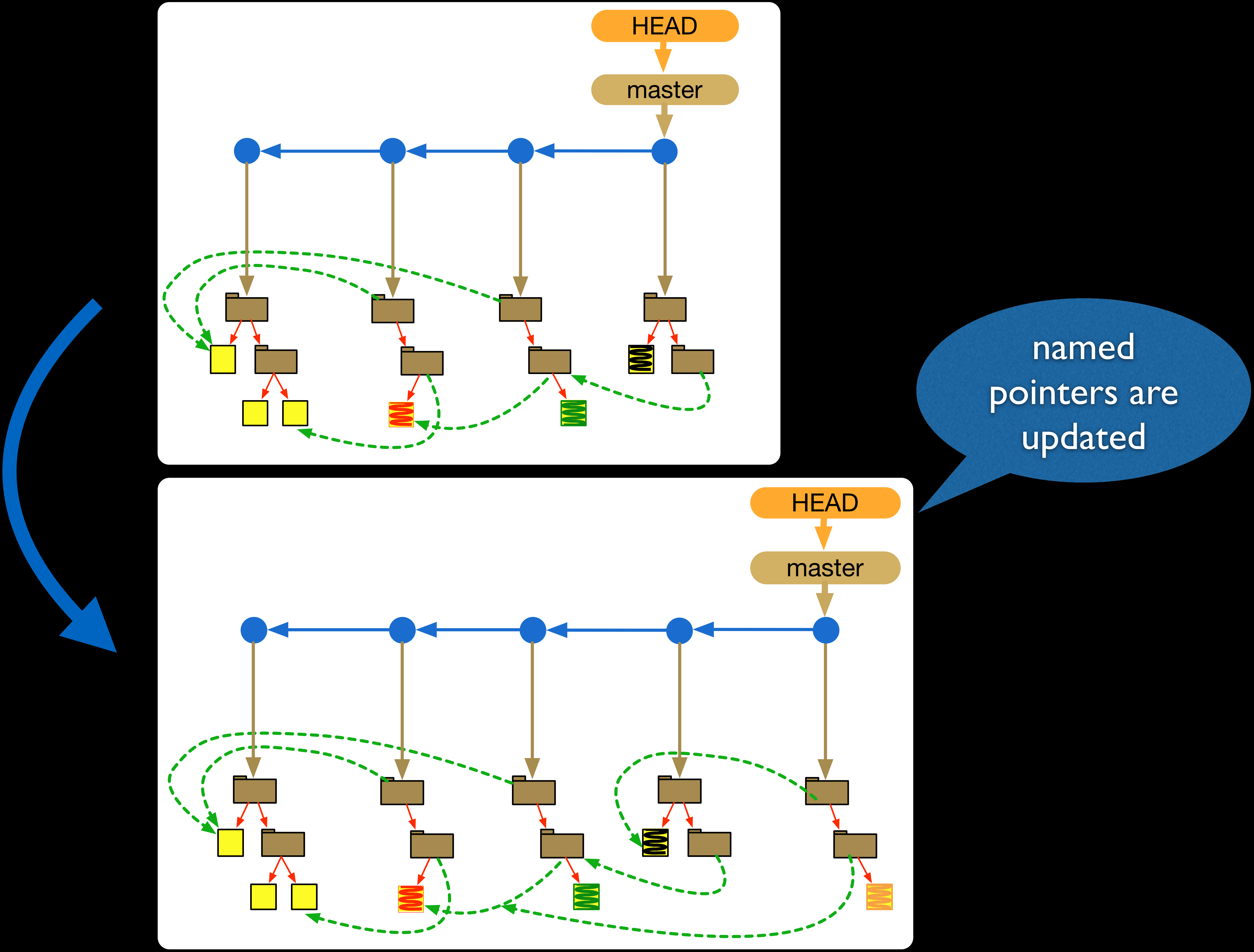

local repository
HEAD
master
staging area
stash
working area
developer machine

git commit

recording changes

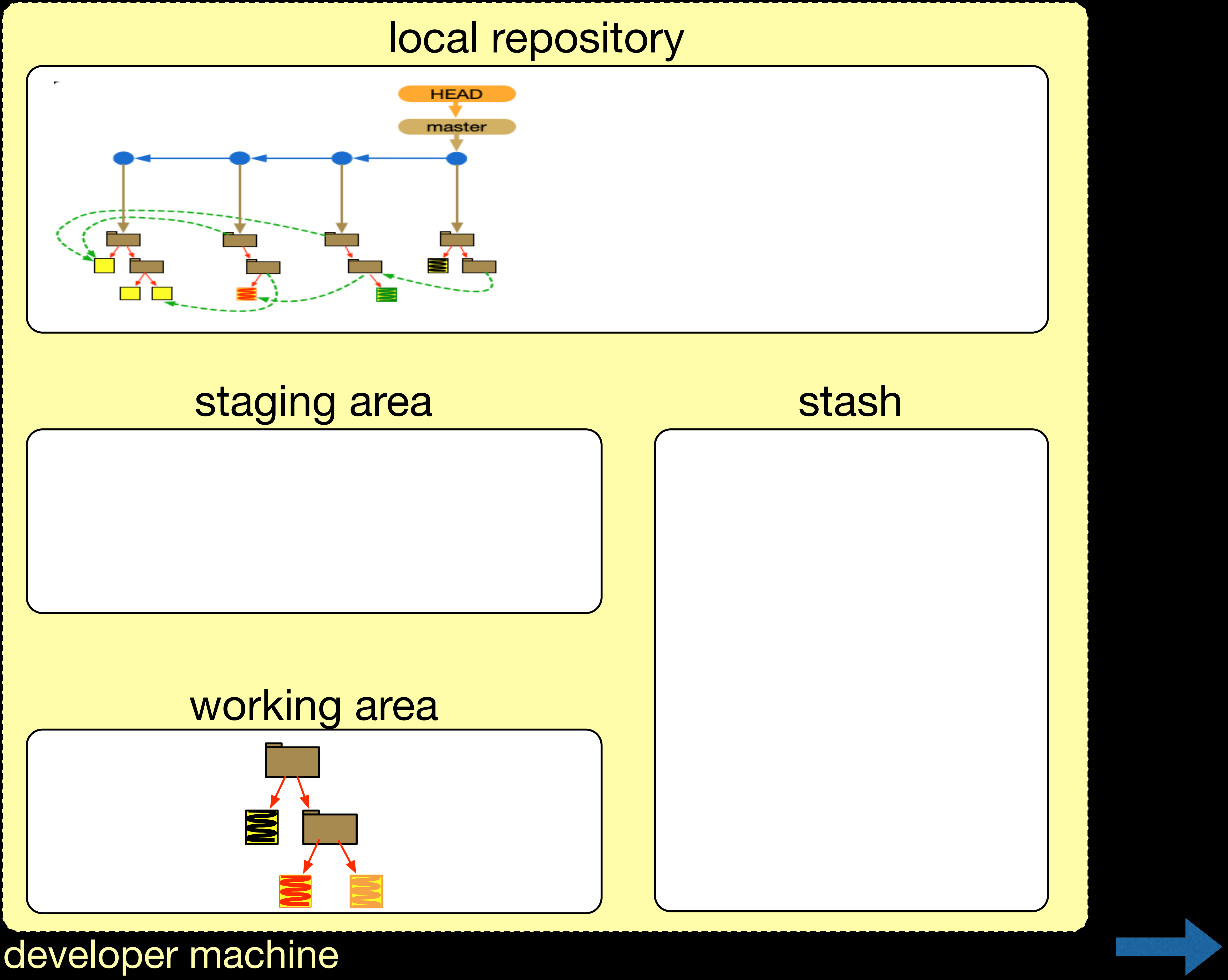

local repository
HEAD
master
staging area
stash
working area
developer machine

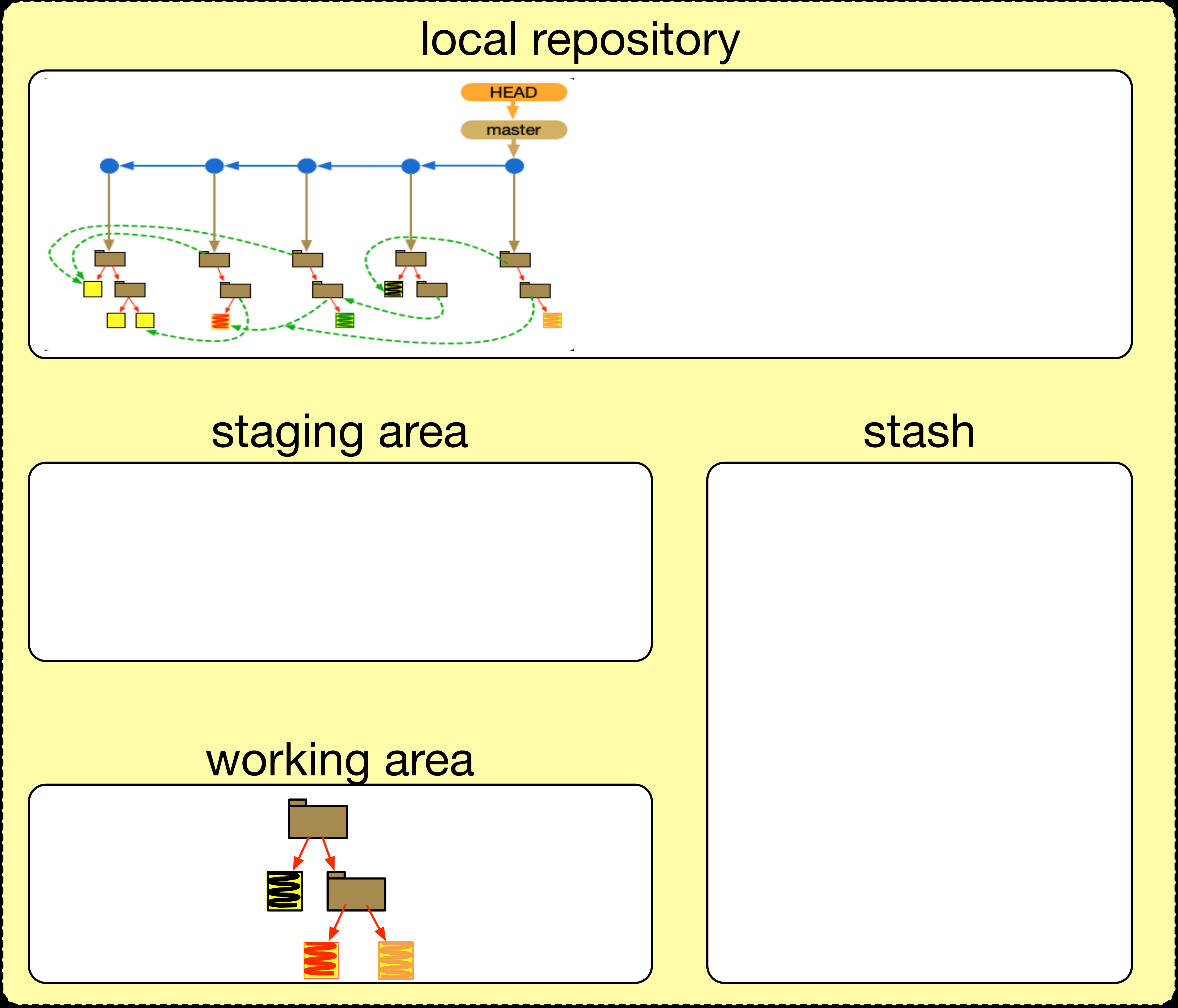

local repository
HEAD
master
staging area
stash
working area
developer machine


HEAD
master
HEAD
master
named pointers are updated

Could select subset of changes that are in the stage area git commit -a

combining commands

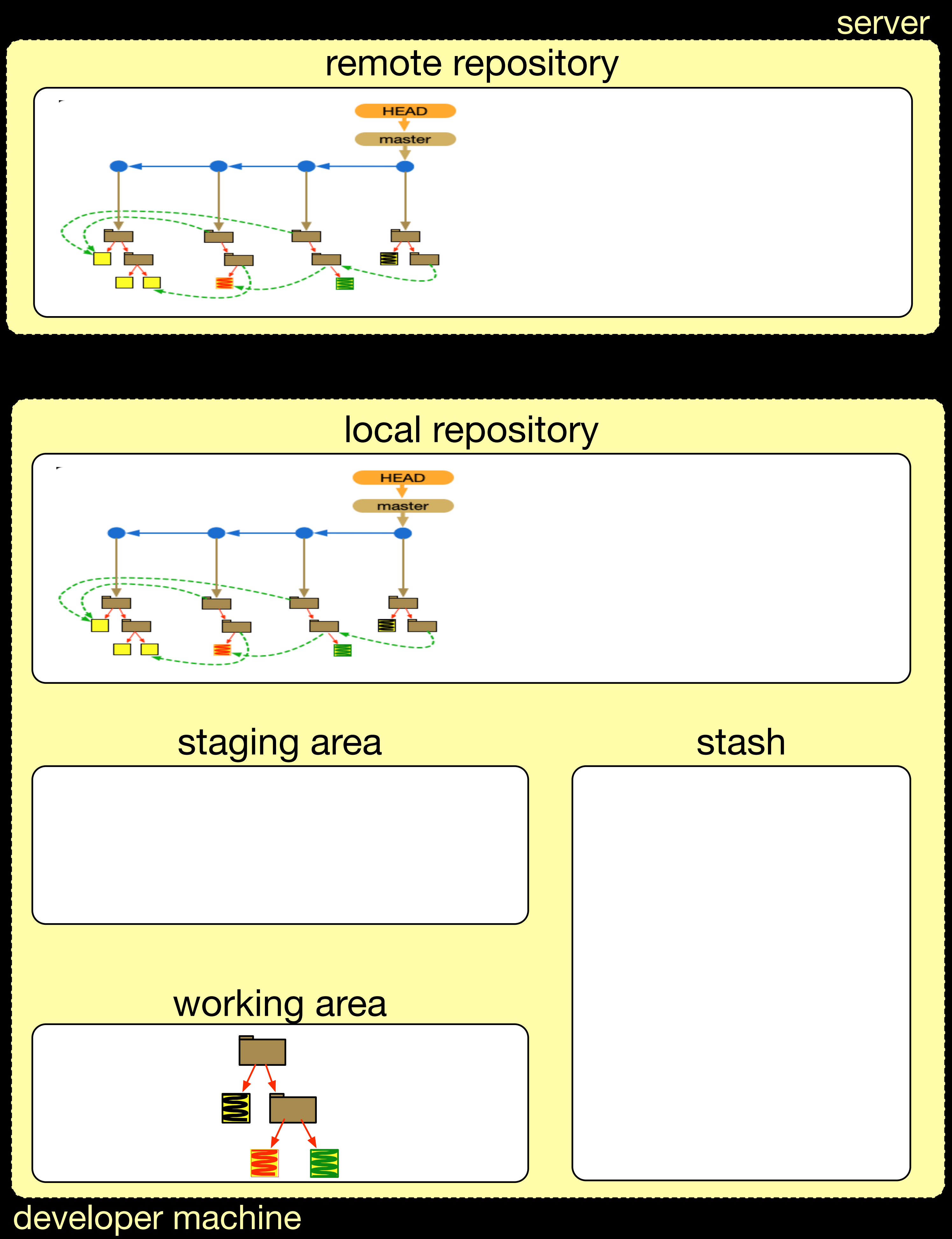

local repository
HEAD
master
staging area
stash
working area
developer machine



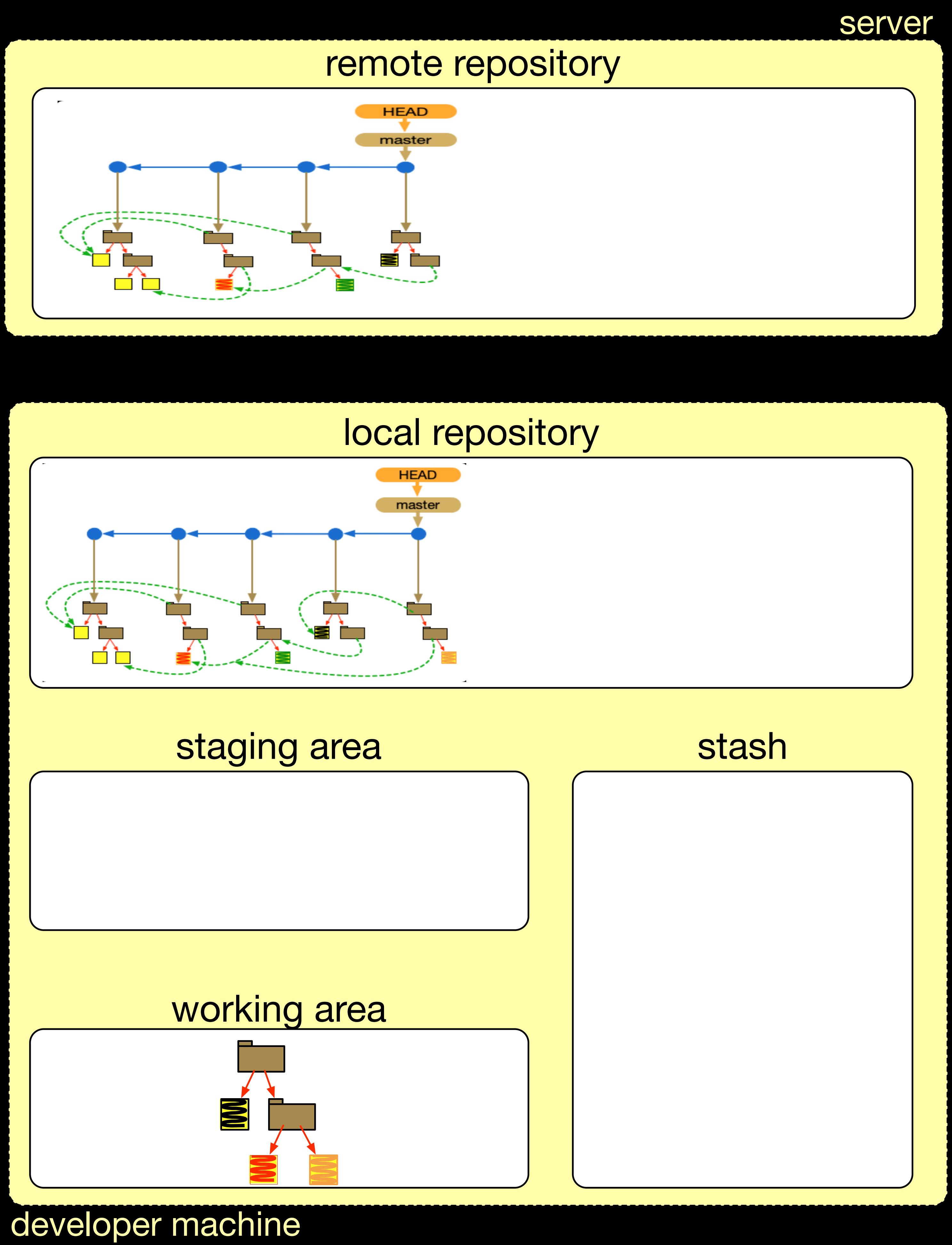
developer machine

Collaborating

At creation time, local and remote commit histories are identical

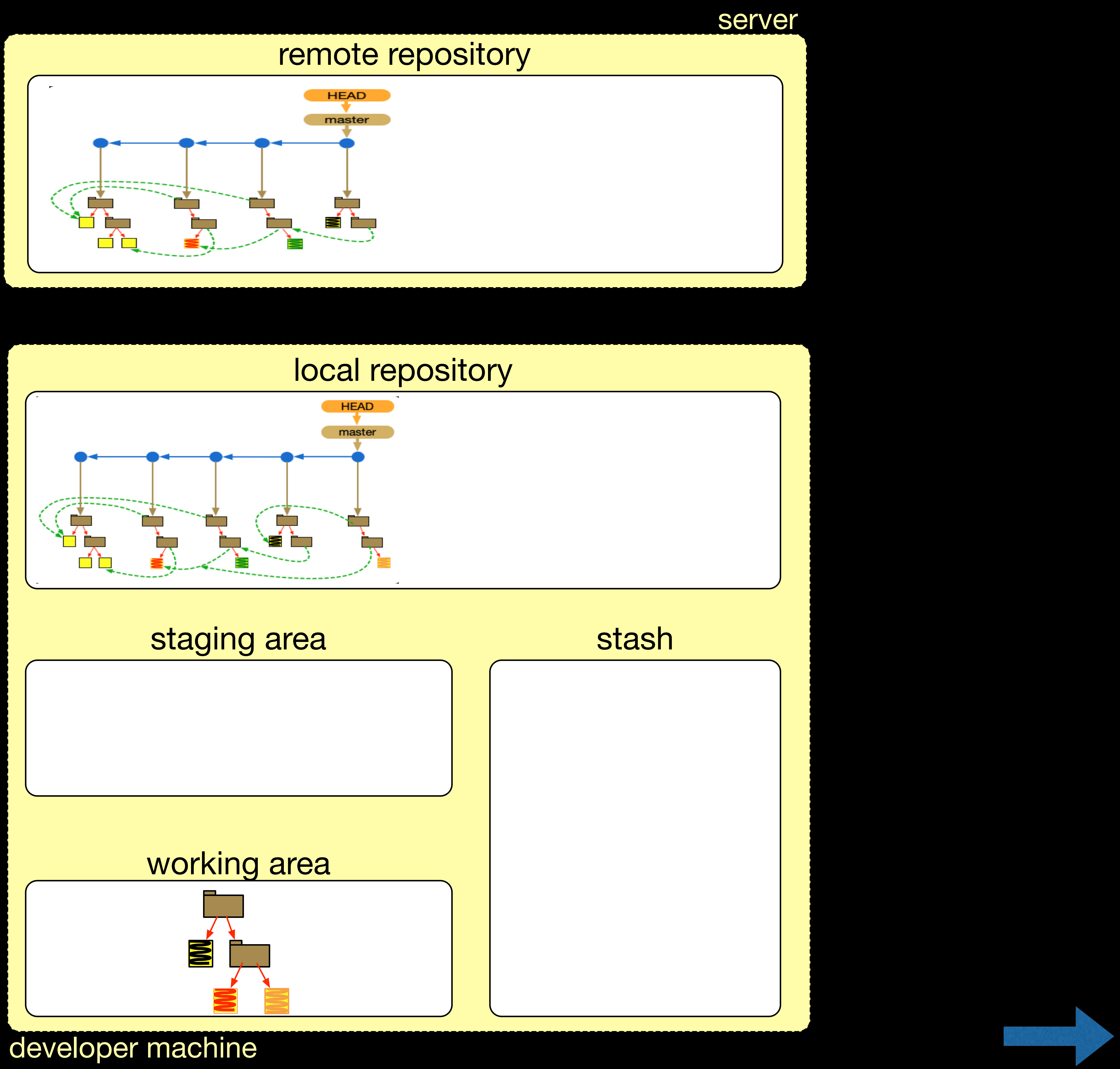

server
remote repository
HEAD
master
local repository
HEAD
master
staging area
stash
working area
developer machine

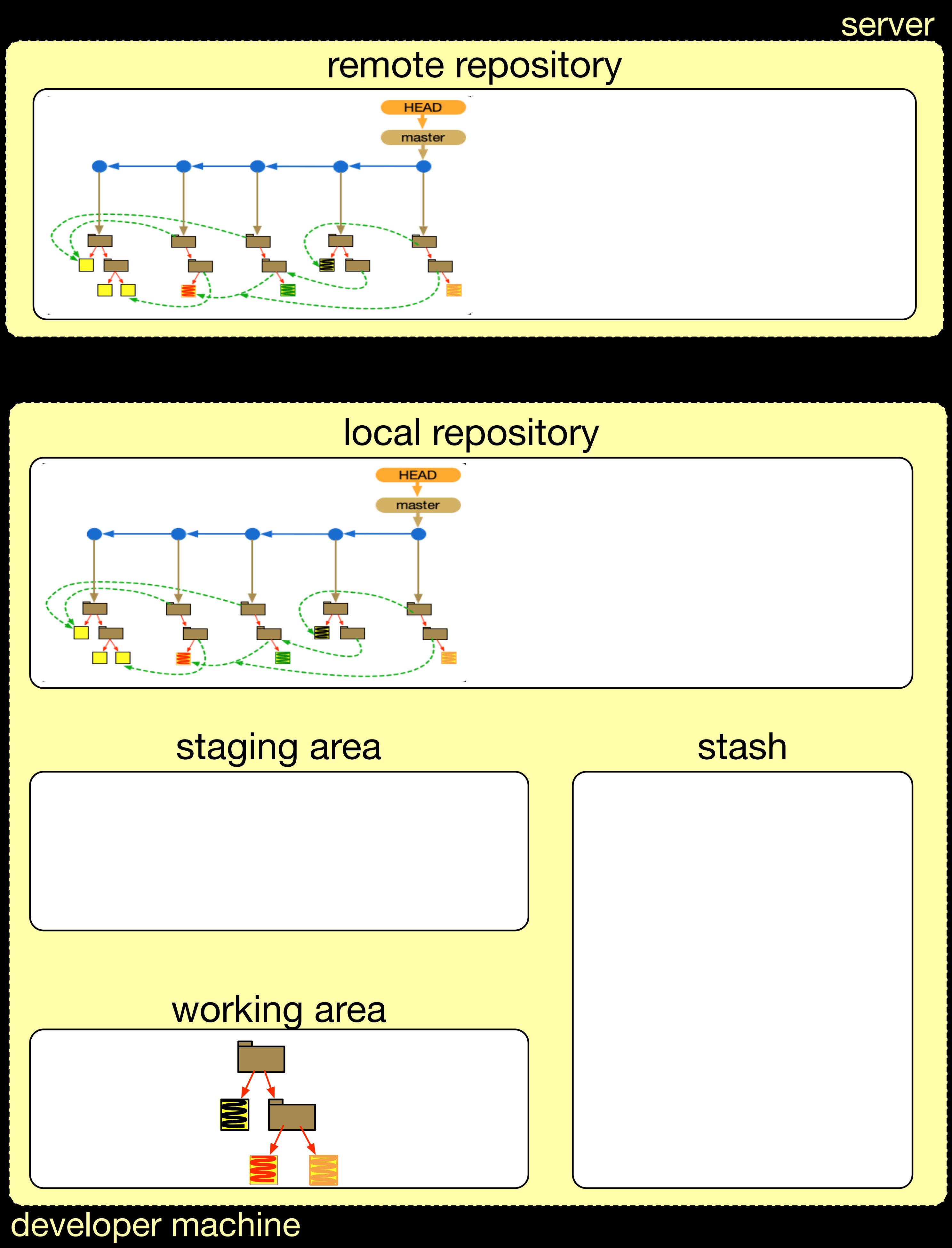
But history might have evolved in the local repository


server
remote repository
HEAD
master
local repository
HEAD
master
staging area
stash
working area
developer machine

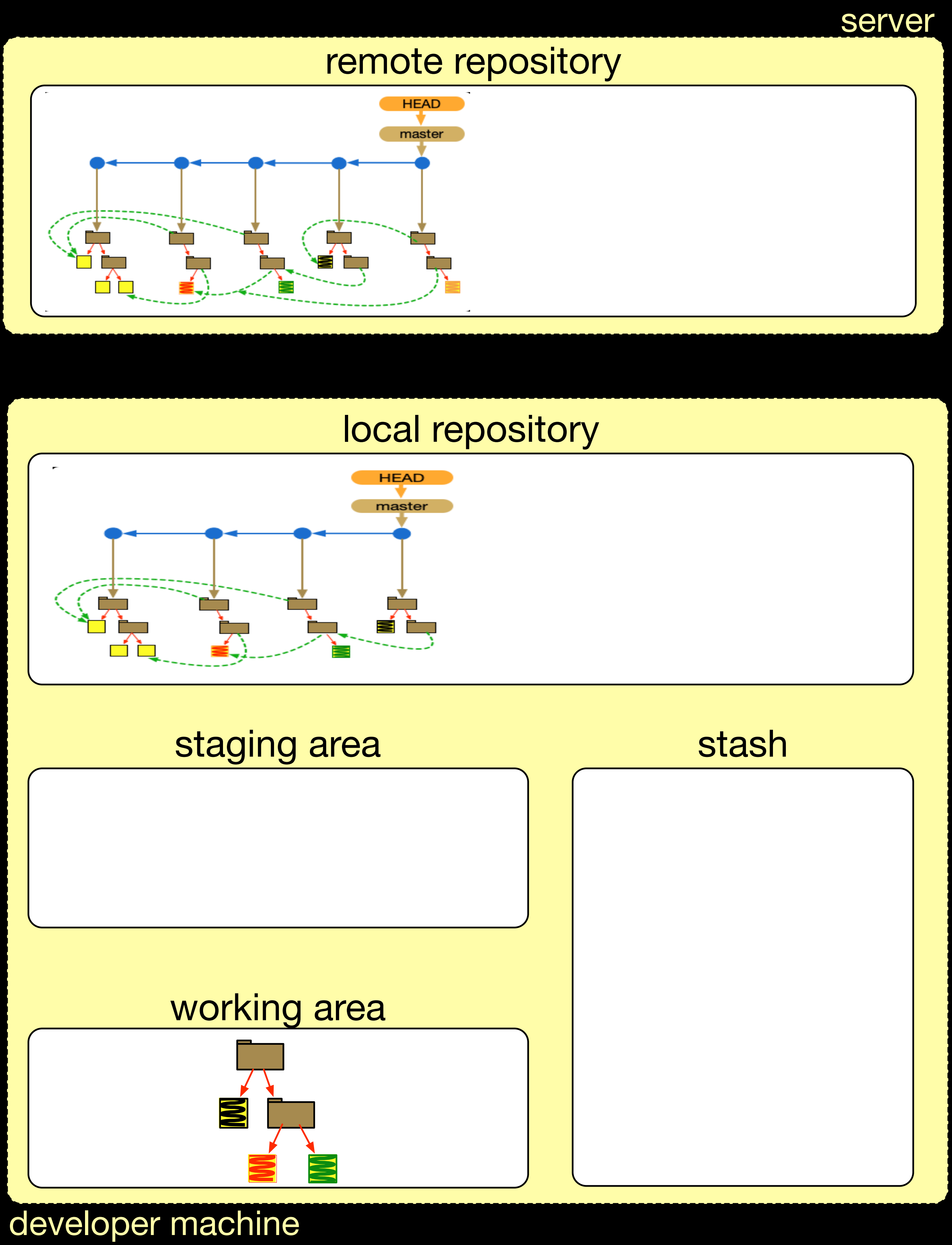
git push

sending changes to a remote repository


server
remote repository
HEAD
master
local repository
HEAD
master
staging area
stash
working area
developer machine

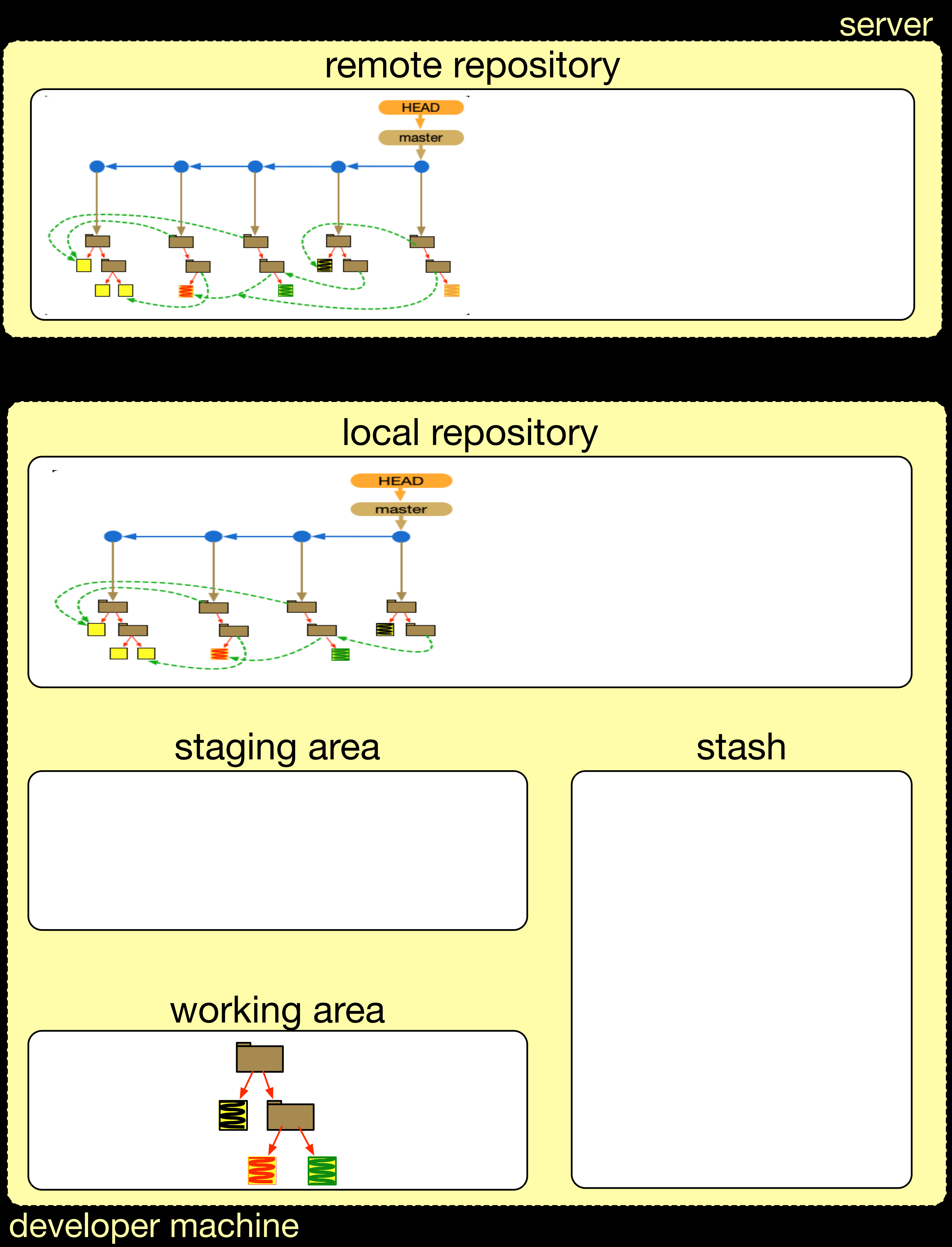

server
remote repository
HEAD
master
local repository
HEAD
master
staging area
stash
working area
developer machine

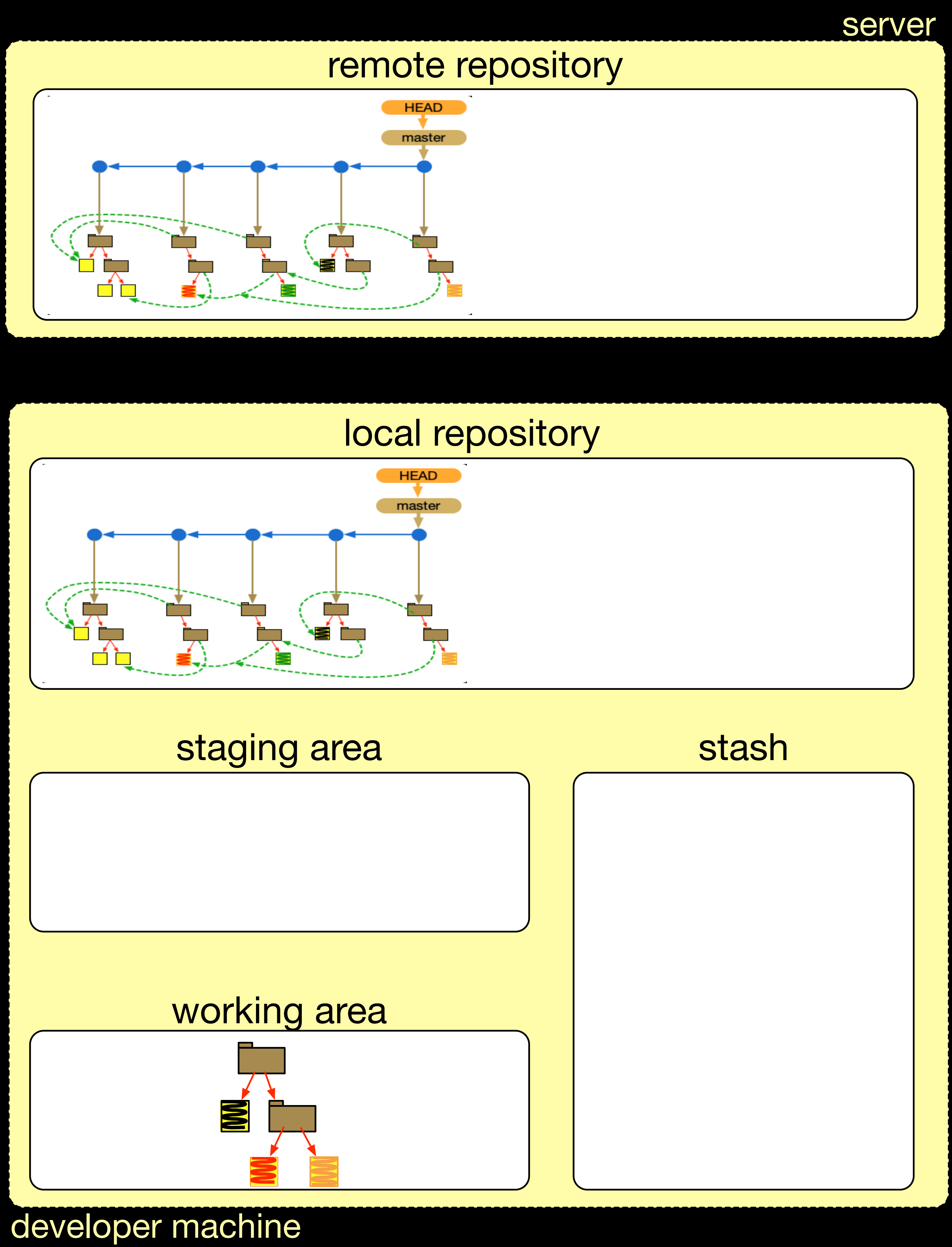
But history might have evolved in the remote repository


server
remote repository
HEAD
master
local repository
HEAD
master
staging area
stash
working area
developer machine

git pull

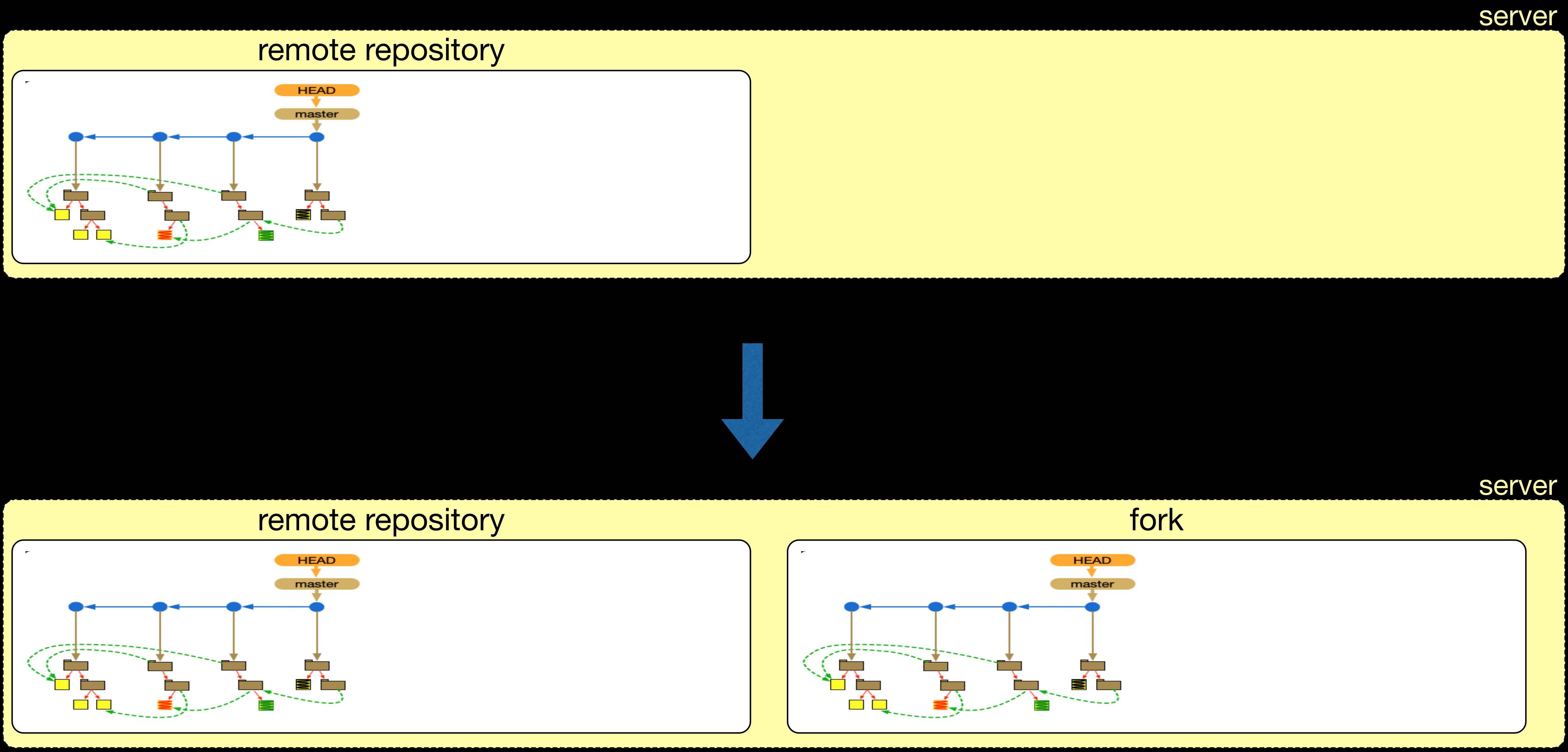
getting changes from a remote repository


server
remote repository
HEAD
master
local repository
HEAD
master
staging area
stash
working area
developer machine


server
remote repository
HEAD
master
local repository
HEAD
master
staging area
stash
working area
developer machine

When you don’t have write access to the remote repository…

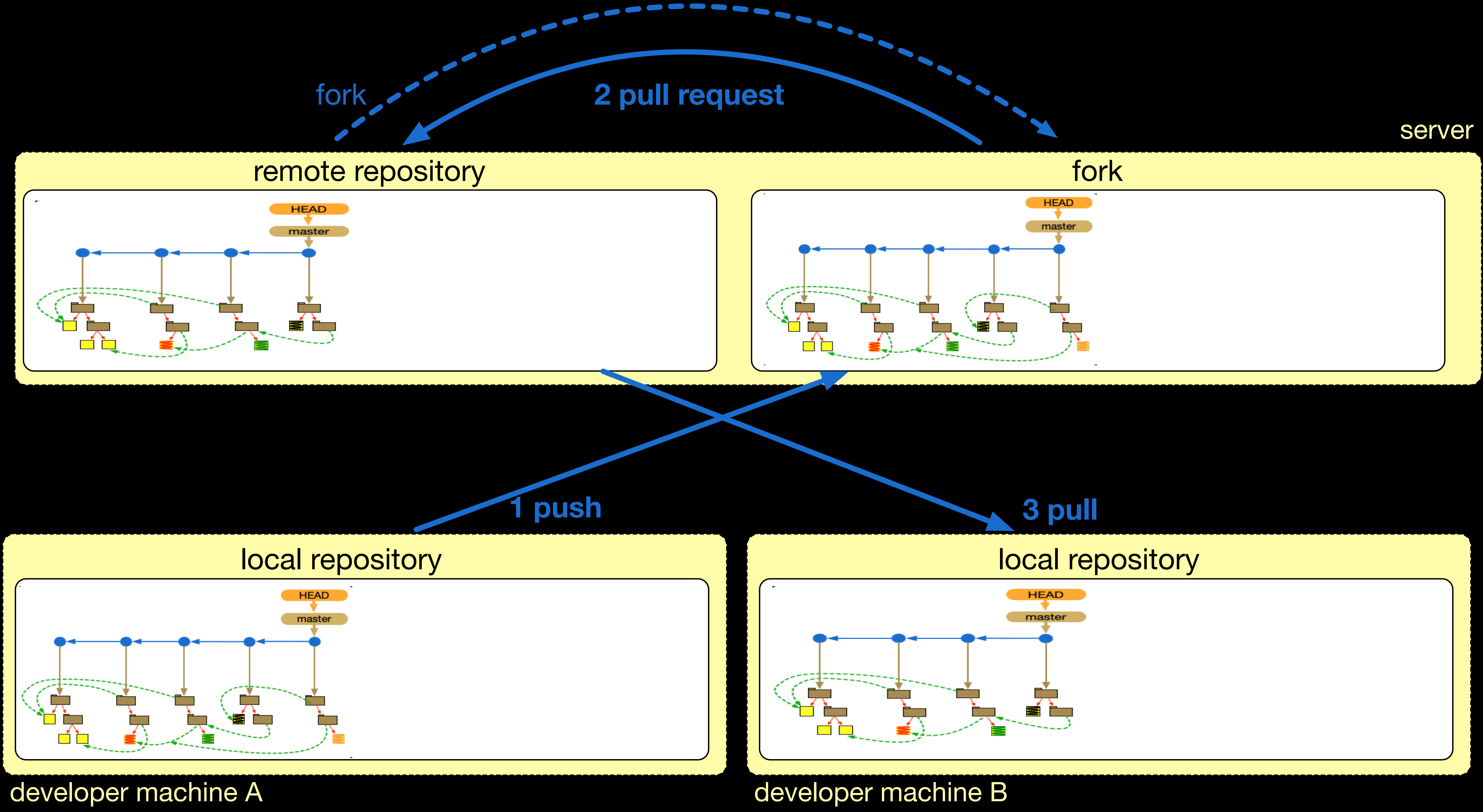
Create your own remote repository, a fork


server
remote repository
HEAD
master
server
remote repository
HEAD
master
fork
HEAD
master

Push to fork

Create and send Pull Request
(from fork to remote)

Pull request collaboration model


fork
2 pull request
server
remote repository
fork
HEAD
master
1 push
3 pull
local repository
local repository
developer machine A
developer machine B

Take notes, now!

Hands on exercises

Configuration management 1, basic concepts and operations

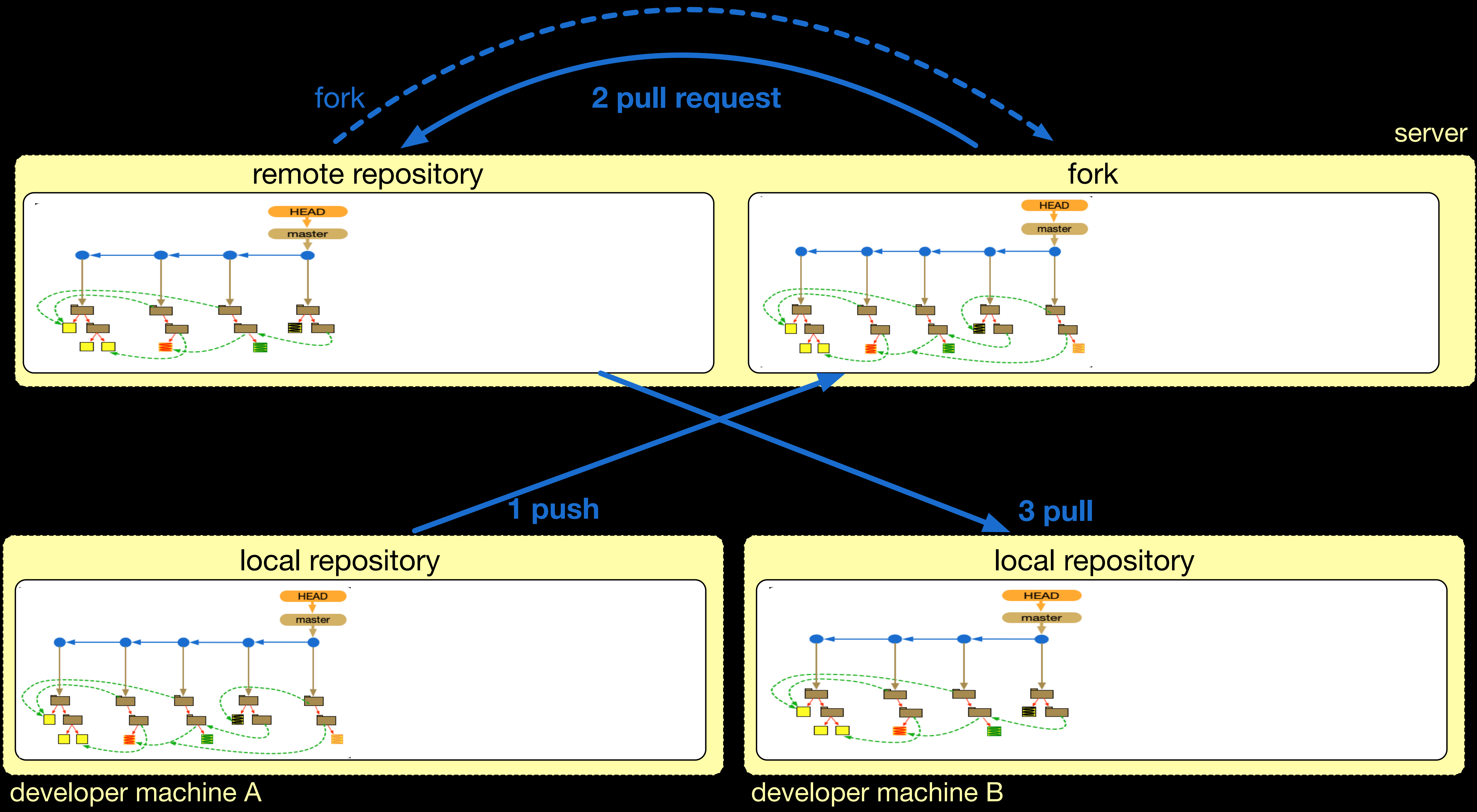
Configuration management 2, advanced concepts and operations

What if the changes you are making are tentative or risky?

And you don’t want to compromise other tasks and developers

Supporting independent (parallel) development

Naturally supported with local repositories

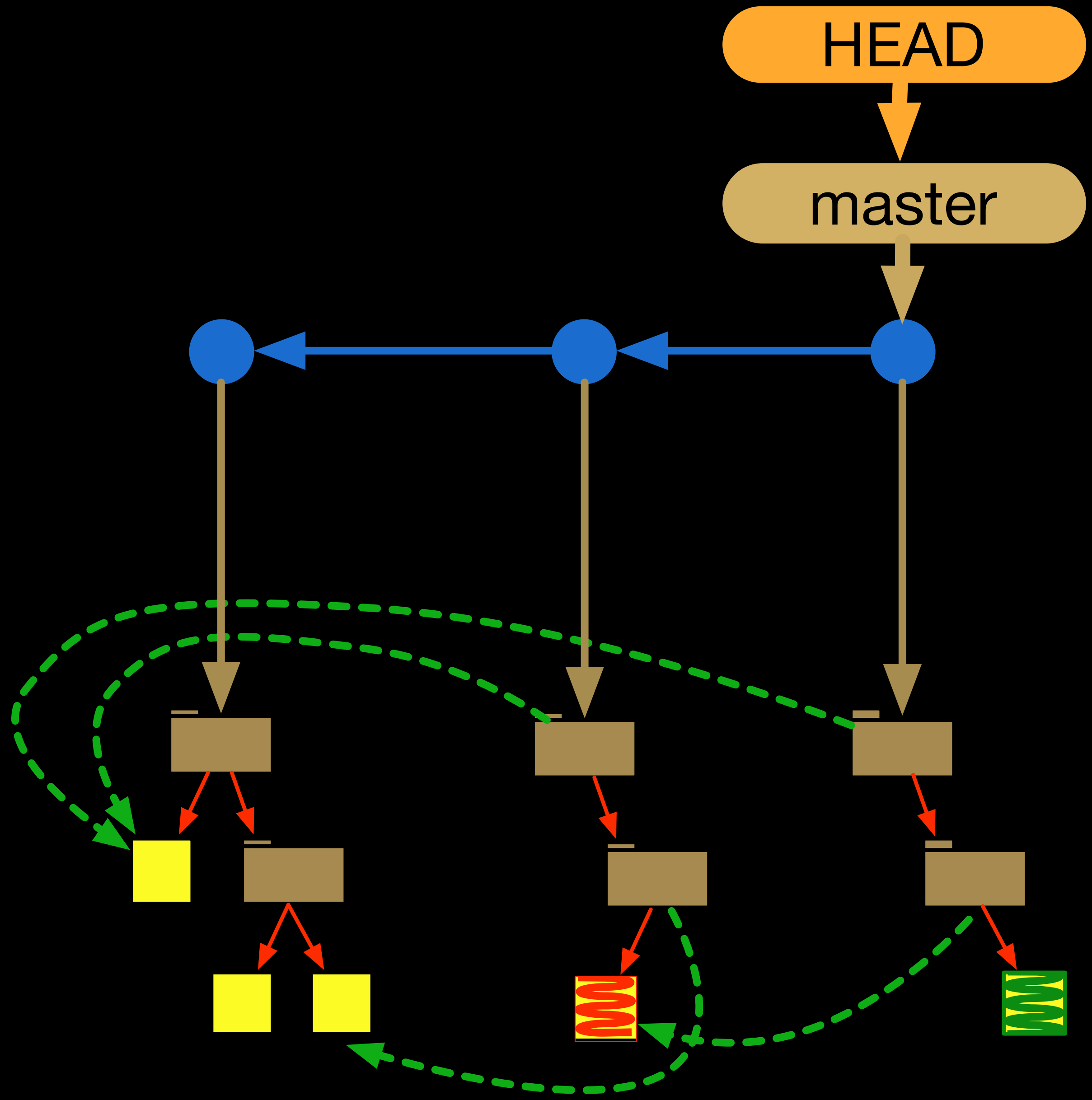

fork
2 pull request
server
remote repository
fork
HEAD
master
1 push
3 pull
local repository
local repository
developer machine A
developer machine B

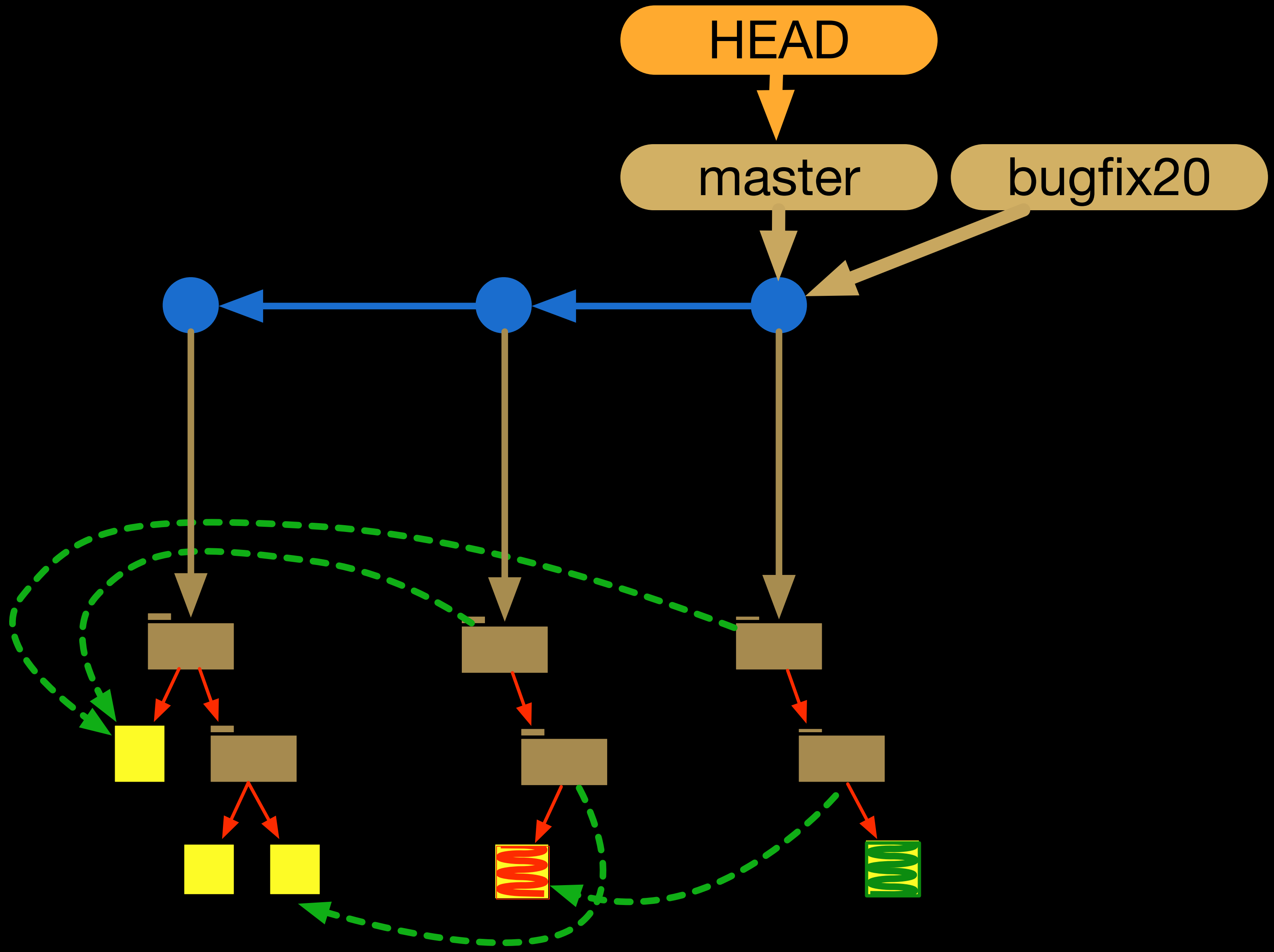
But can we also support independence locally?

git branch

branches as pointers

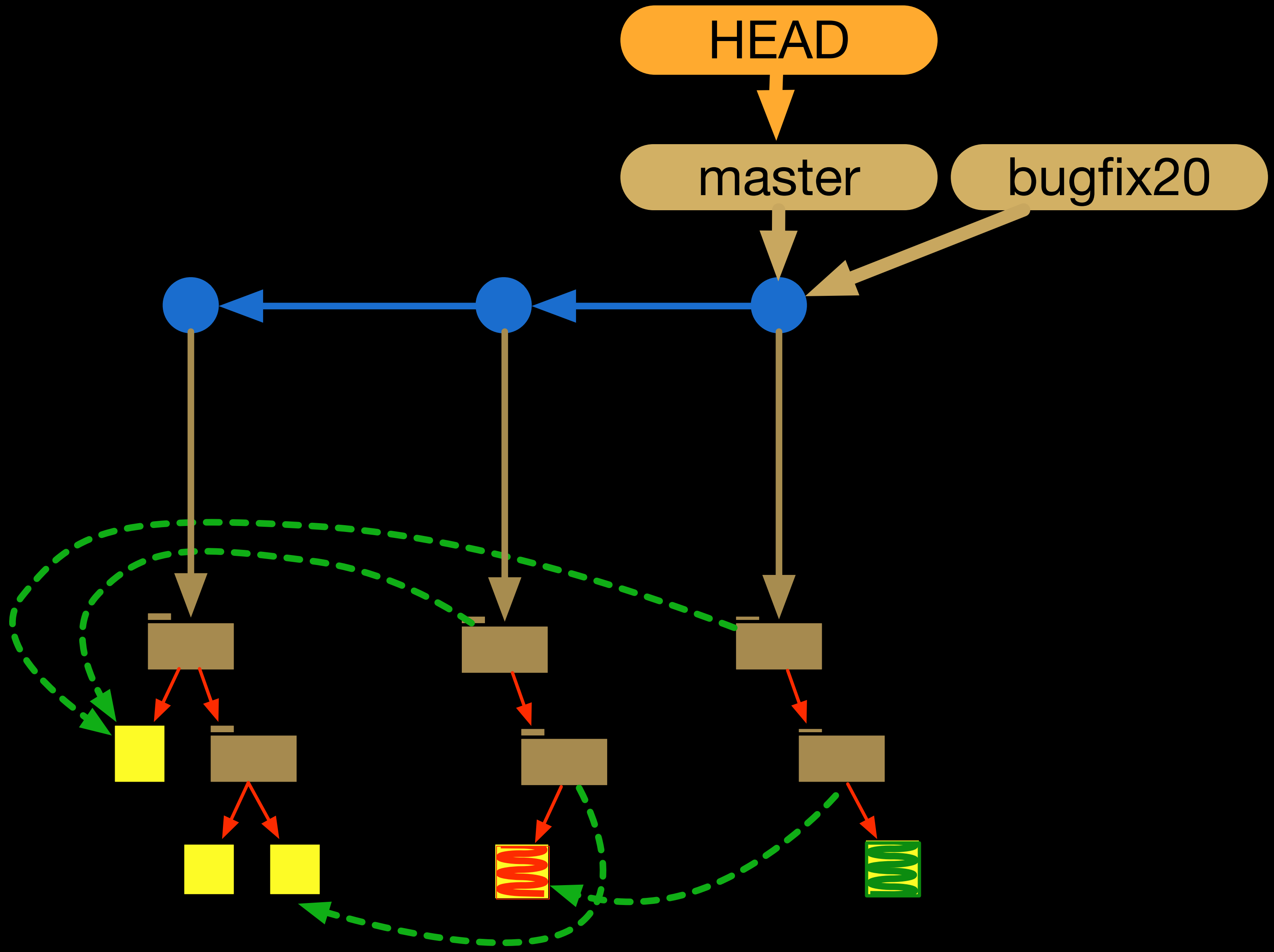
multiple development lines

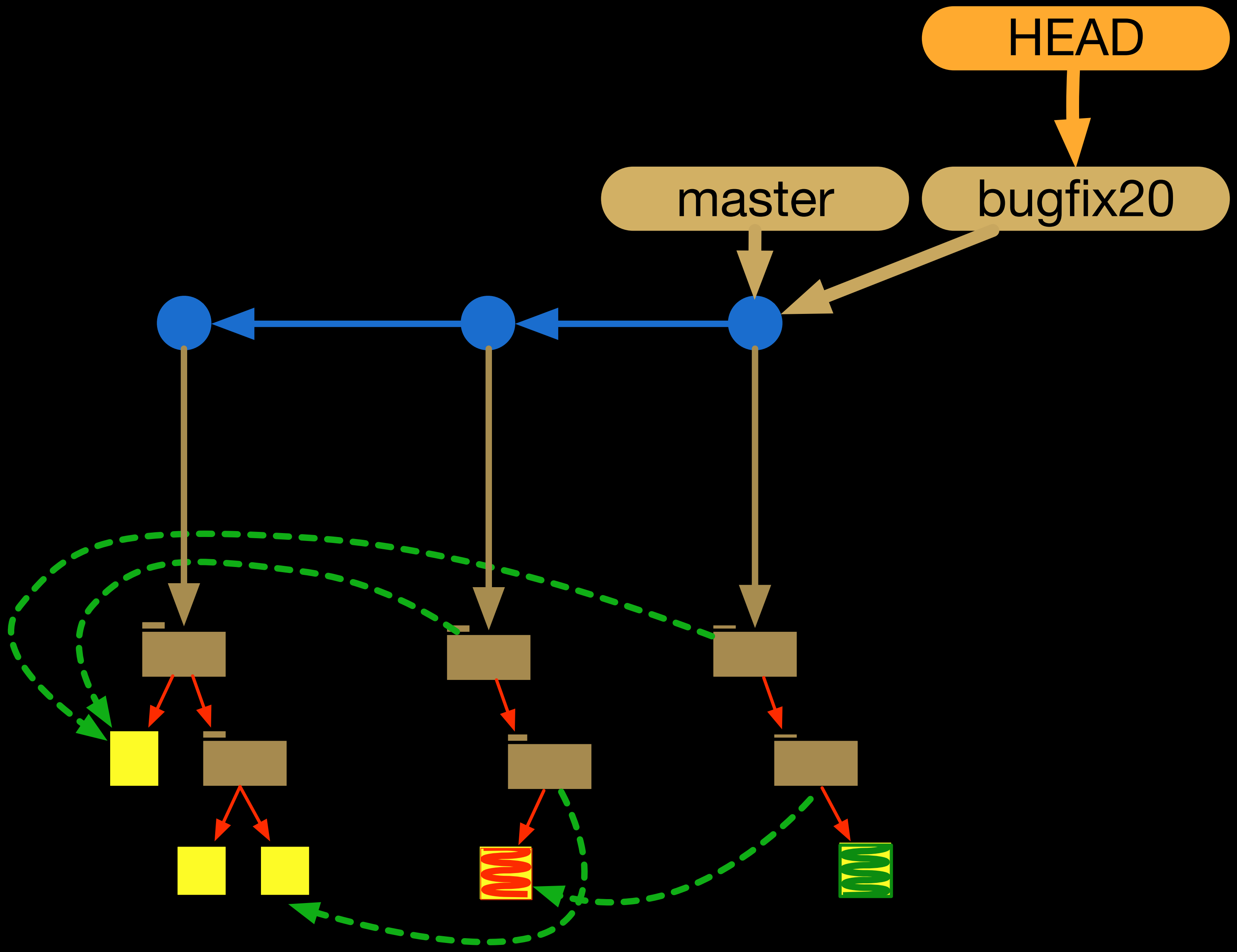

HEAD
master


HEAD
master
bugfix20

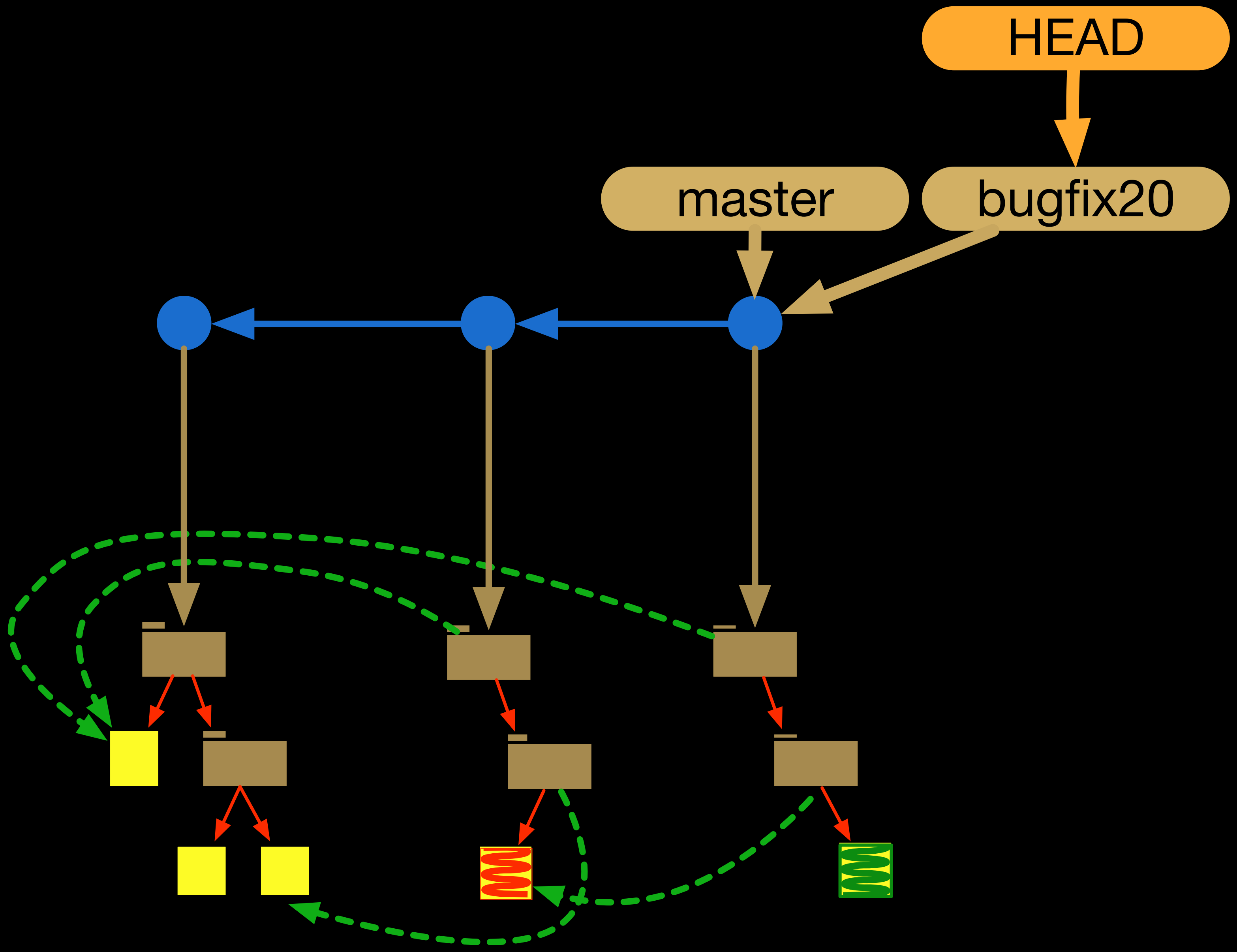
git checkout

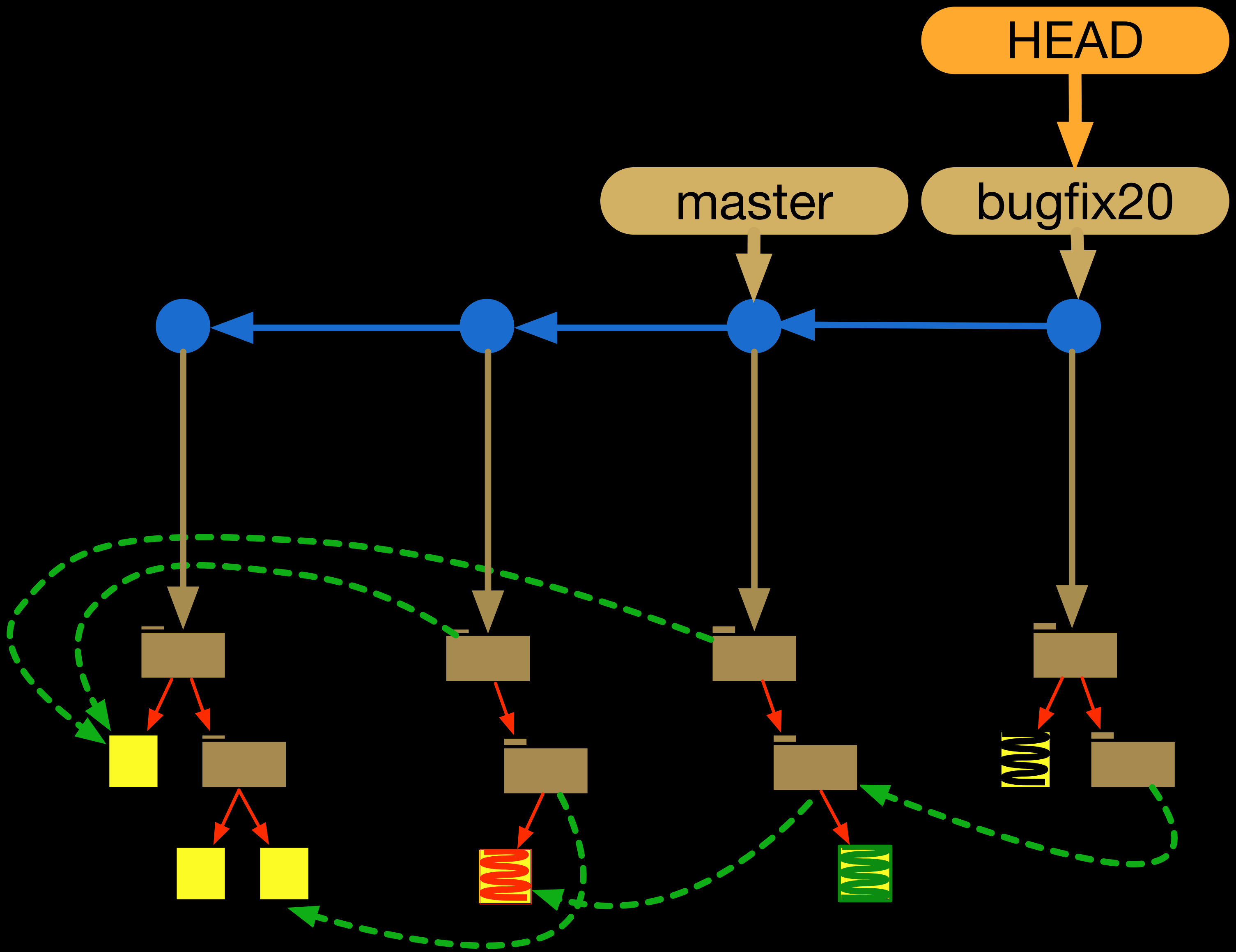
changing branches


HEAD
master
bugfix20

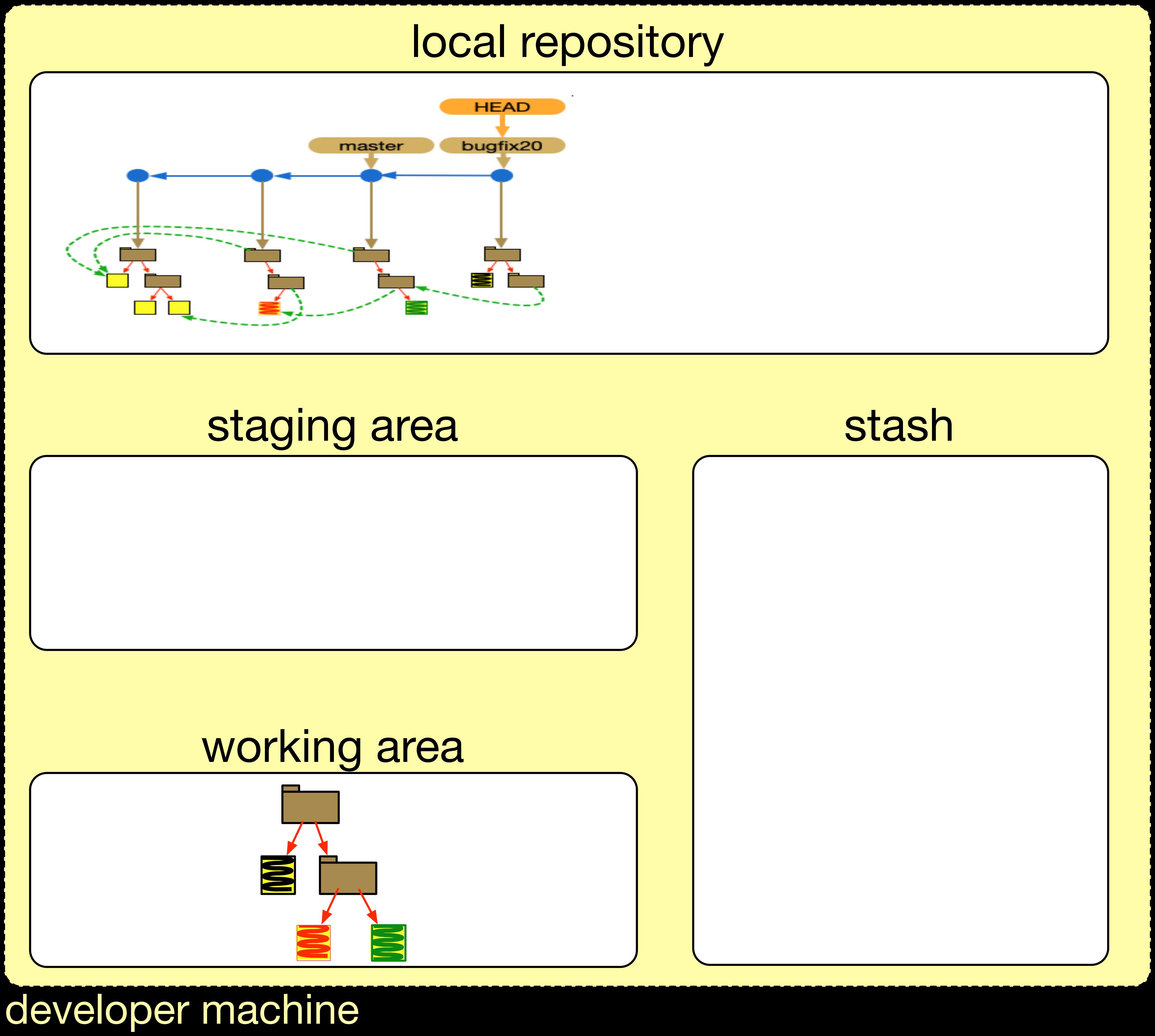

HEAD
master
bugfix20

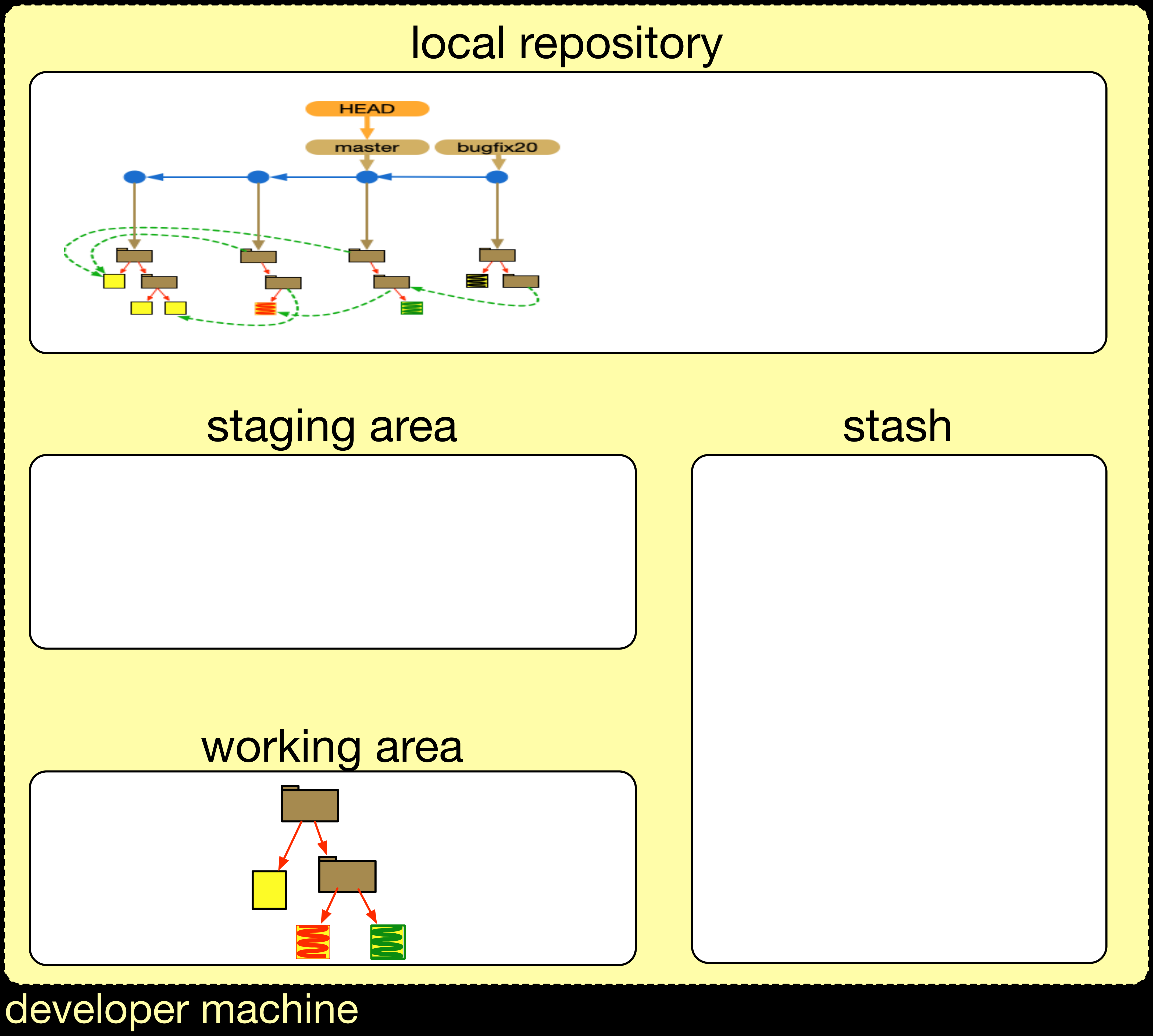
evolving a branch by creating new commits


HEAD
master
bugfix20

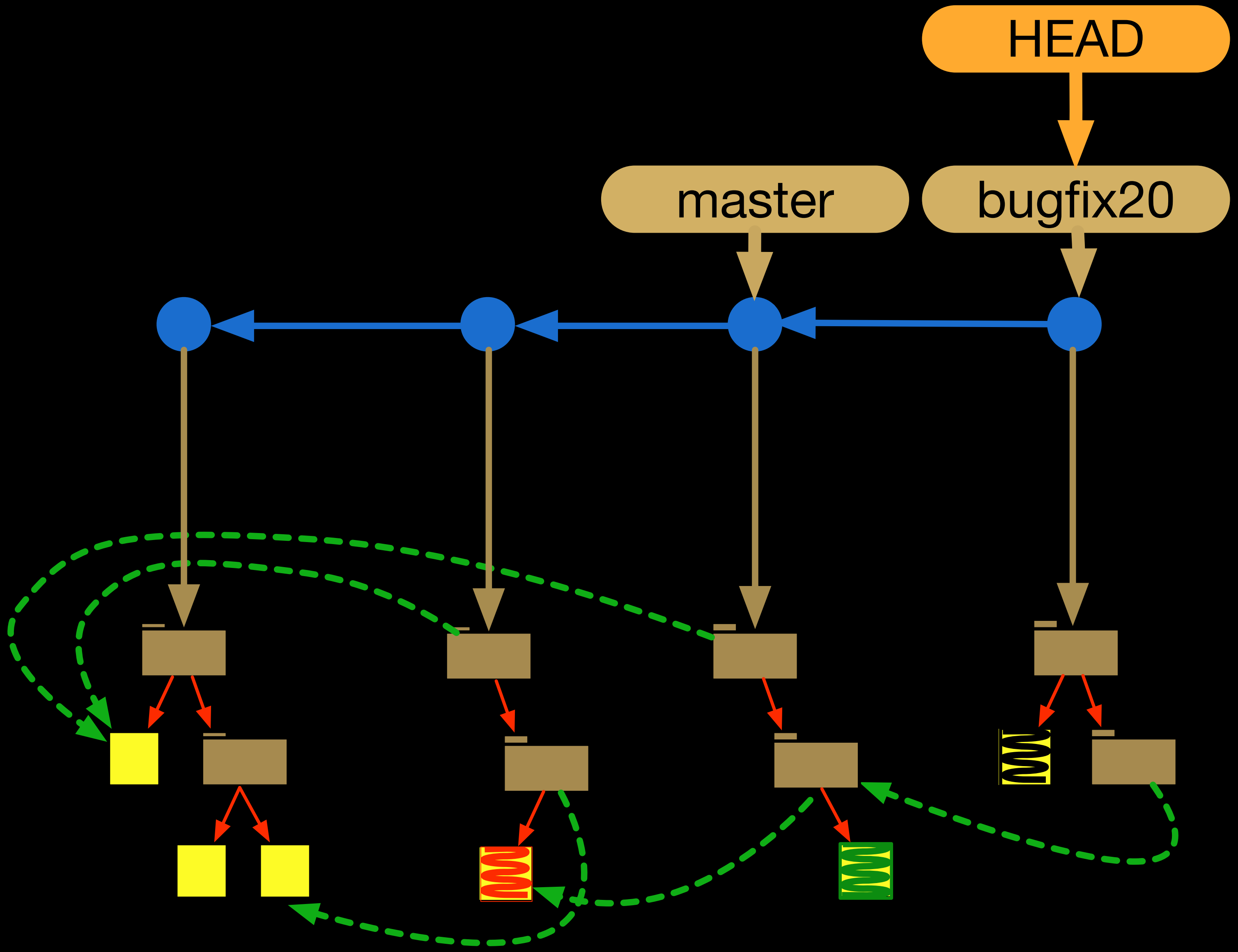

HEAD
master
bugfix20

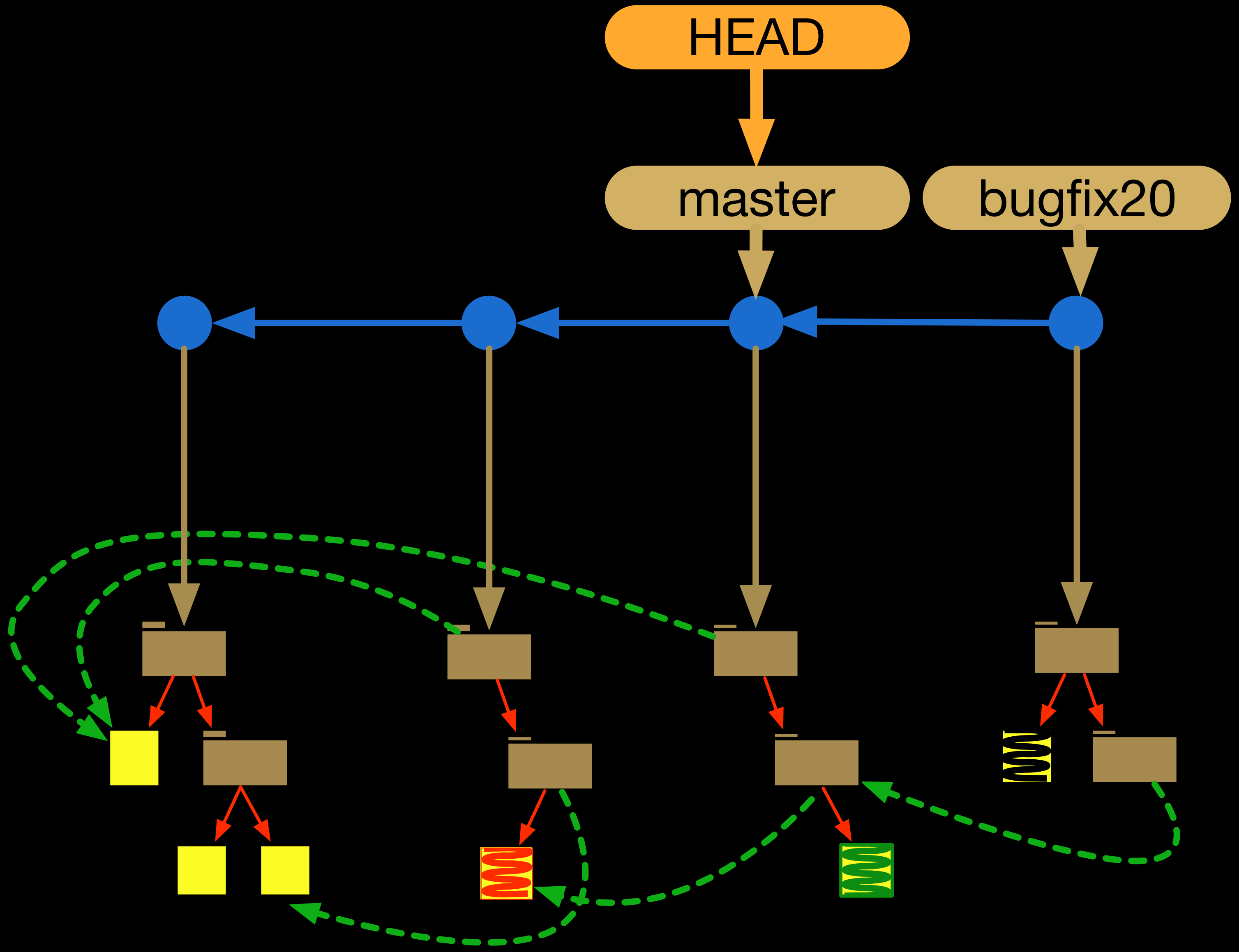
git checkout effect on working area

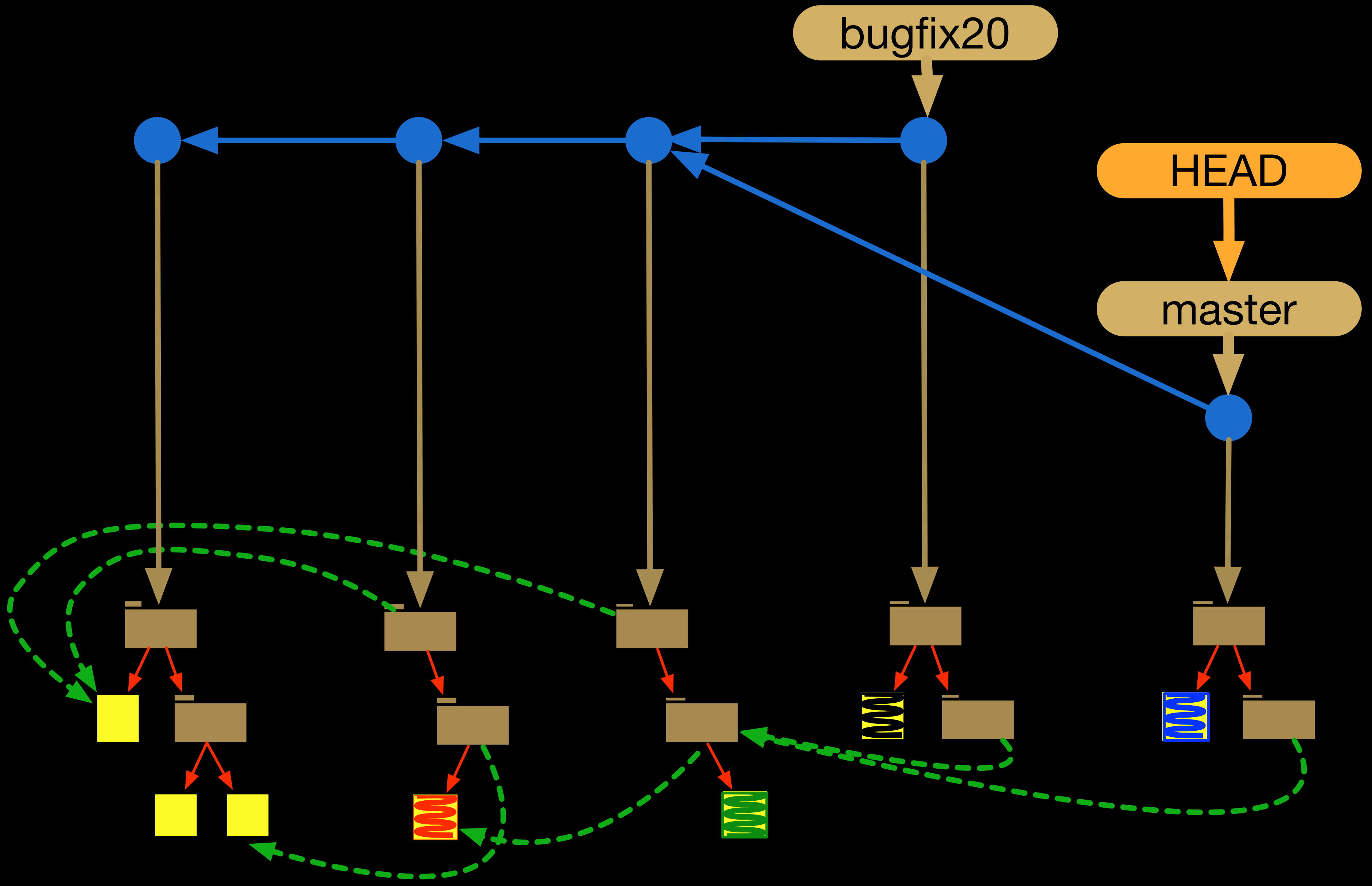

local repository
HEAD
master
bugfix20
staging area
stash
working area
developer machine


local repository
HEAD
master
bugfix20
staging area
stash
working area
developer machine

evolving another branch by creating new commits


HEAD
master
bugfix20

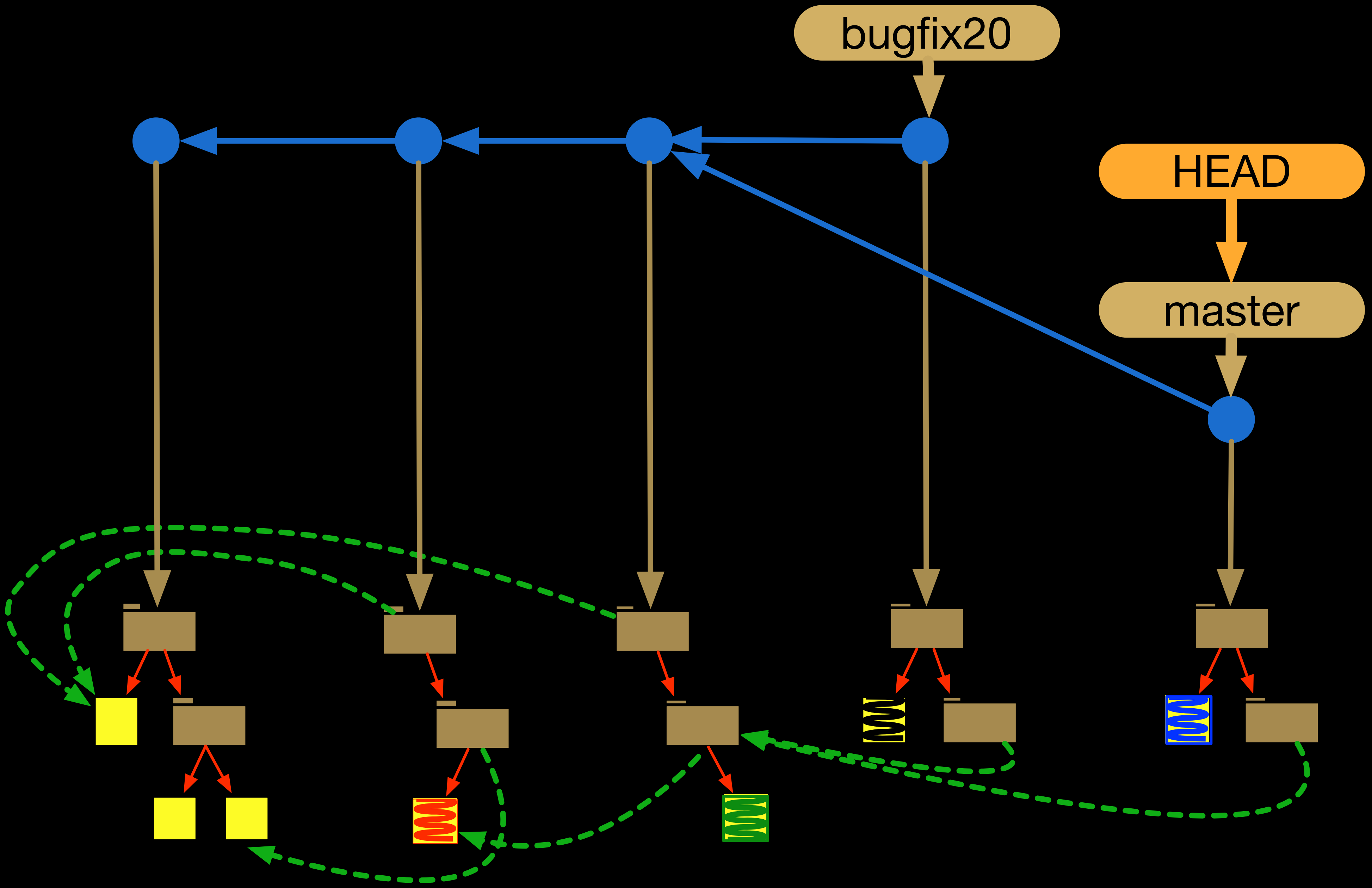

HEAD
master
bugfix20

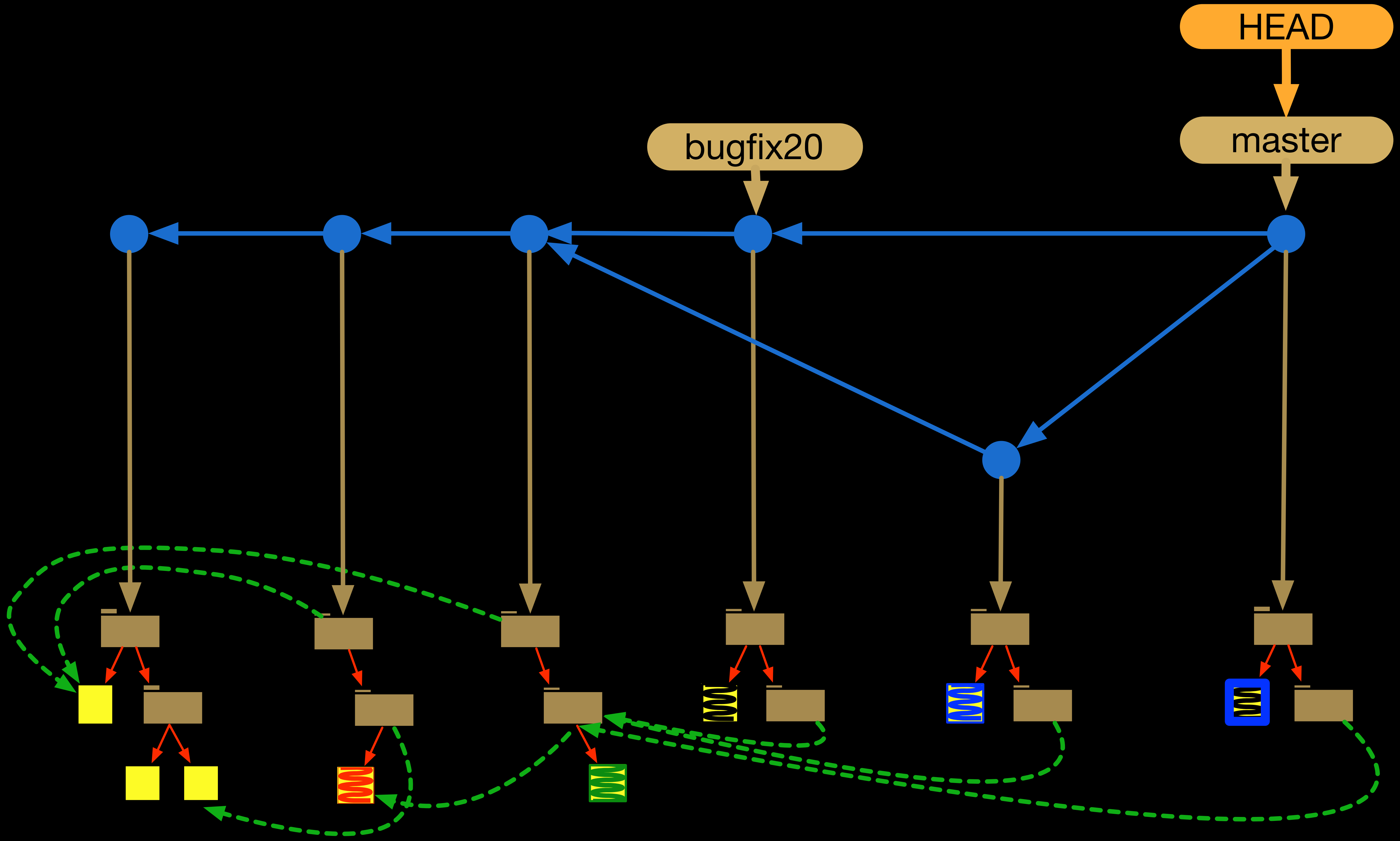

bugfix20
HEAD
master

Integrating code contributions

git merge

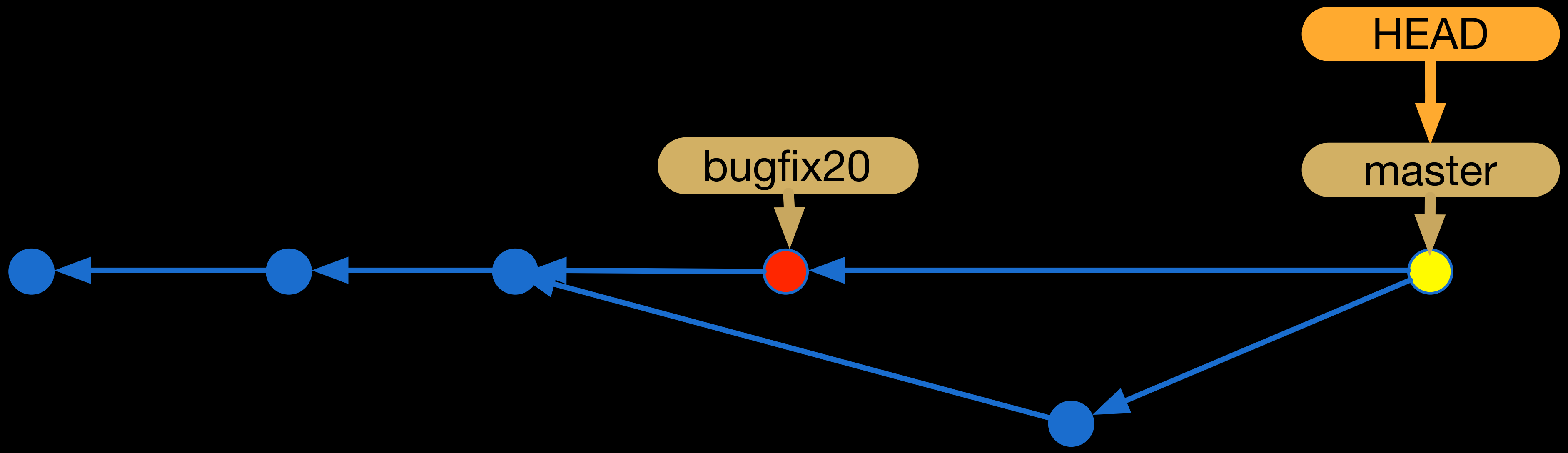
merging locally

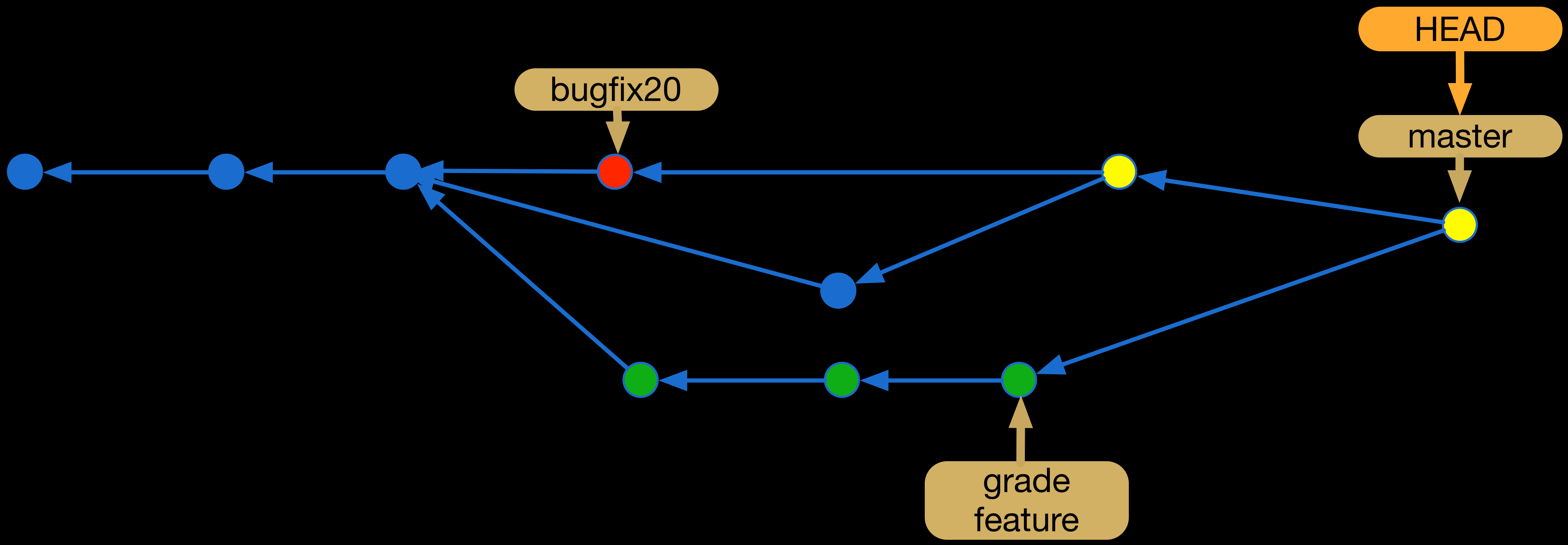

bugfix20
HEAD
master

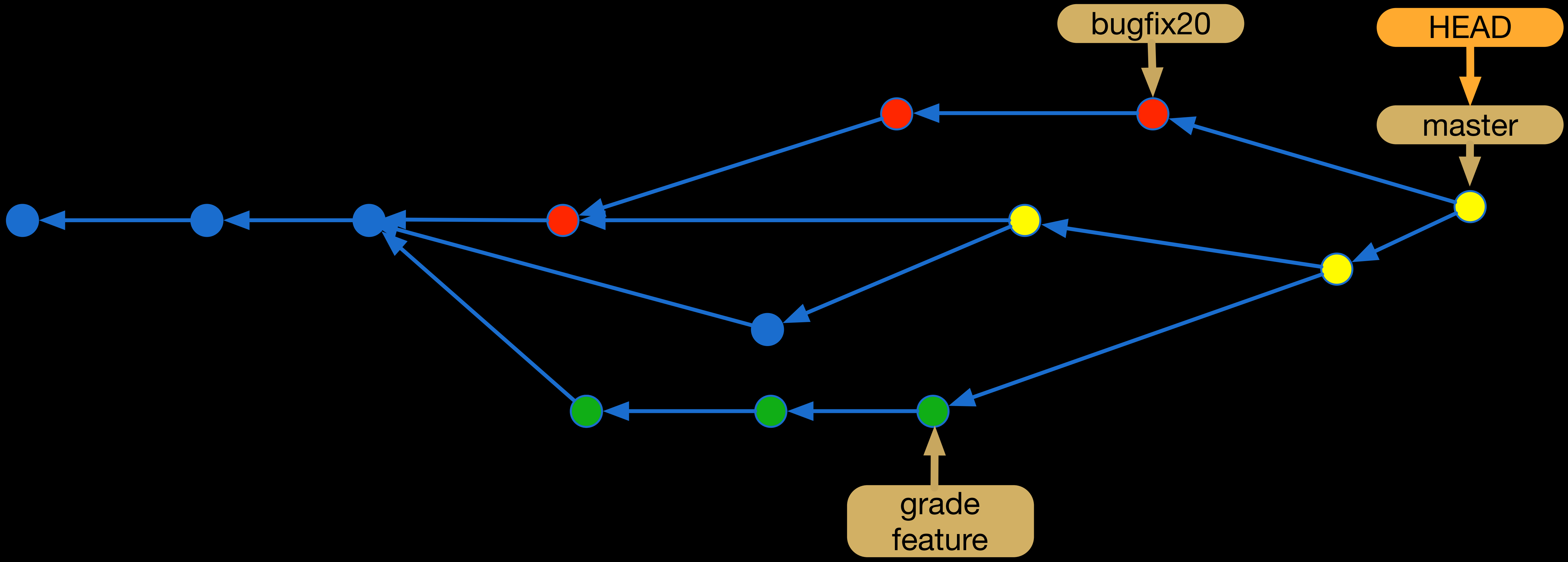

HEAD
bugfix20
master

multiple branches and lines

multiple merges


HEAD
bugfix20
master

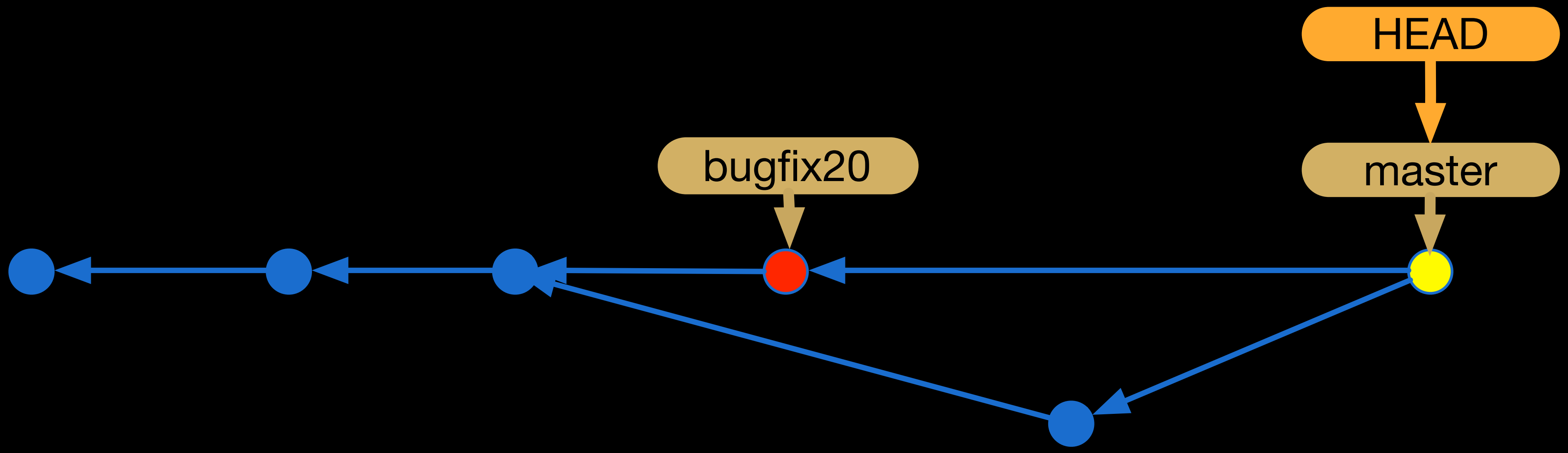

HEAD
bugfix20
master
grade feature

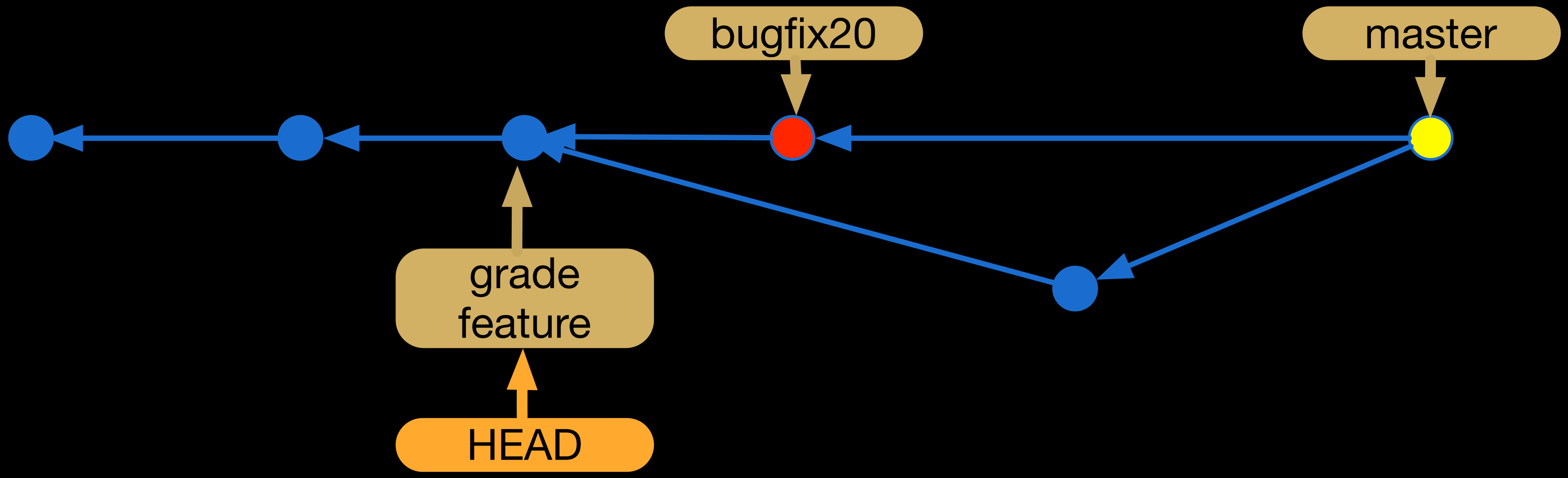

bugfix20
HEAD
master
grade feature

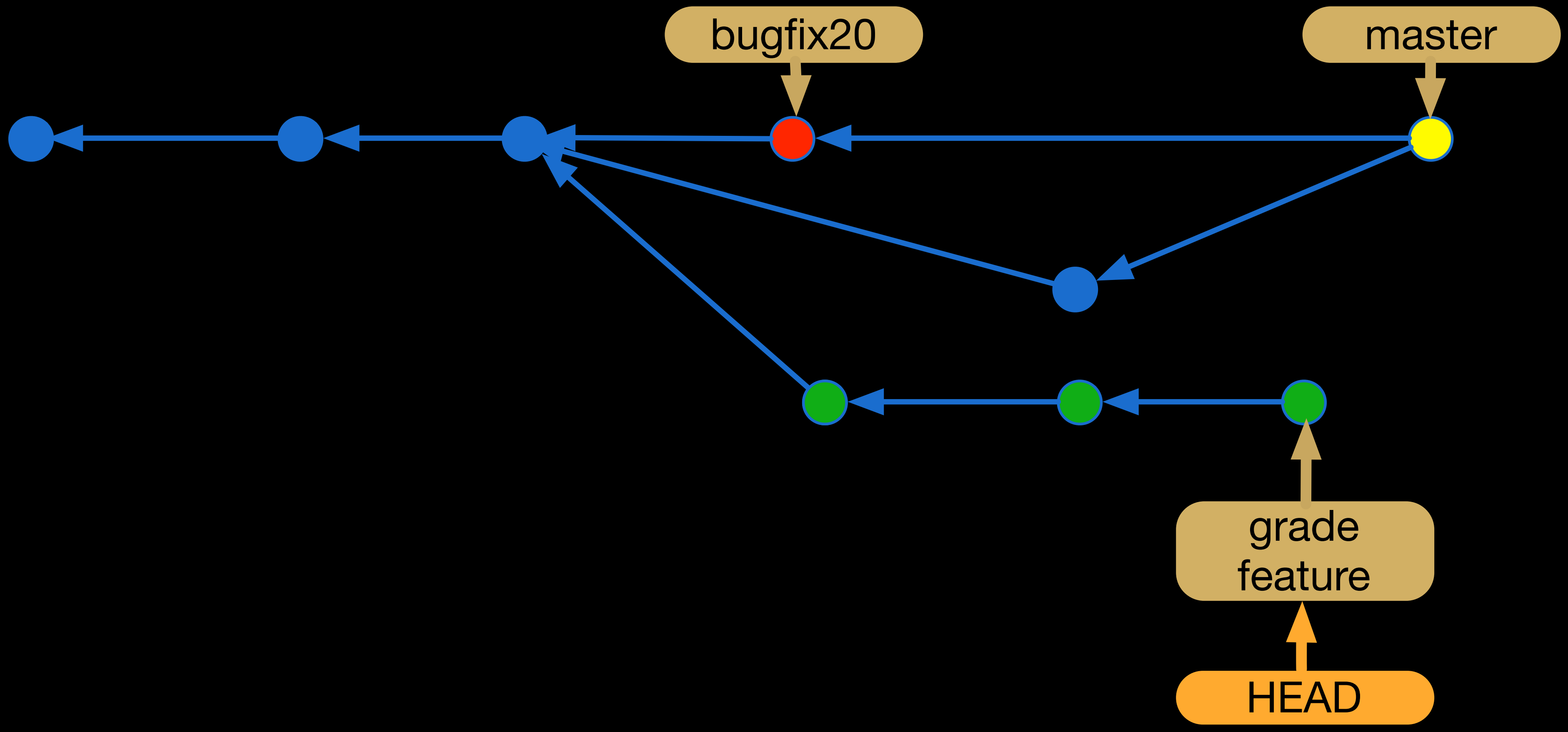
commit hashes

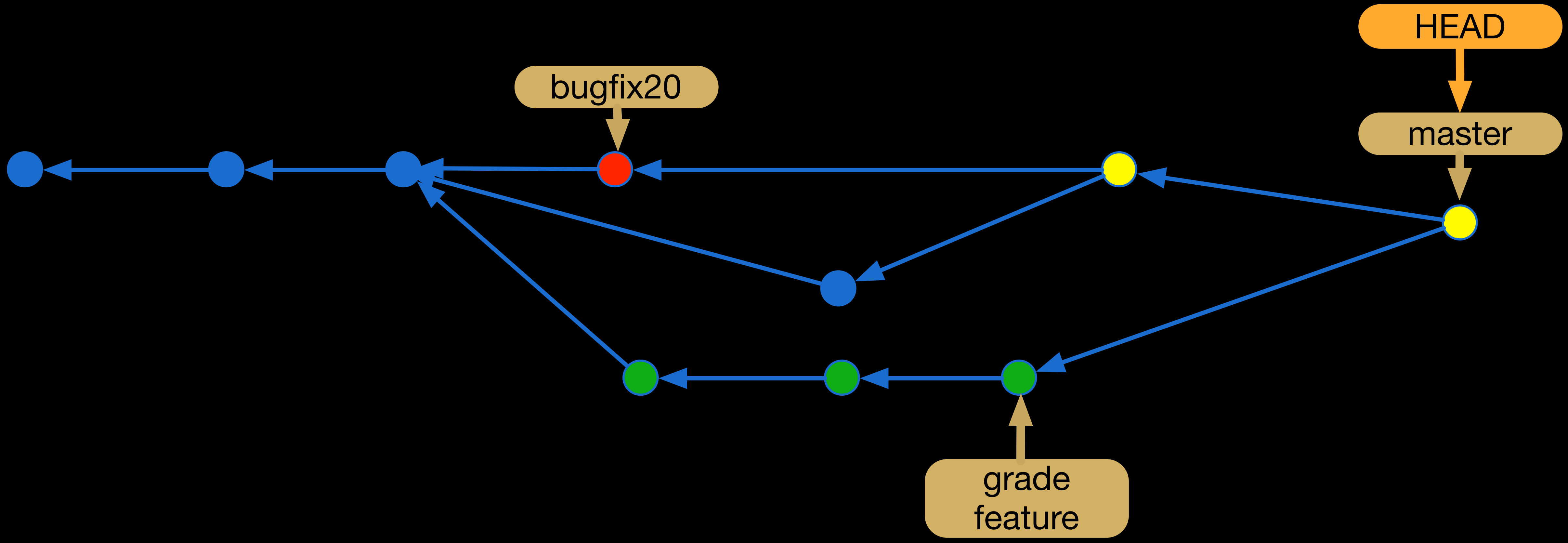
```
ed7058d9fd9ef55eed72abf29752bc50aa43ac5b

331cd13938e1487b8fbb2dd2fe7714d4934cf274

5fe7dfebaabc5c60f521568f96288f72dfb82c4c
```

git checkout for branches and hashes


HEAD
bugfix20
master

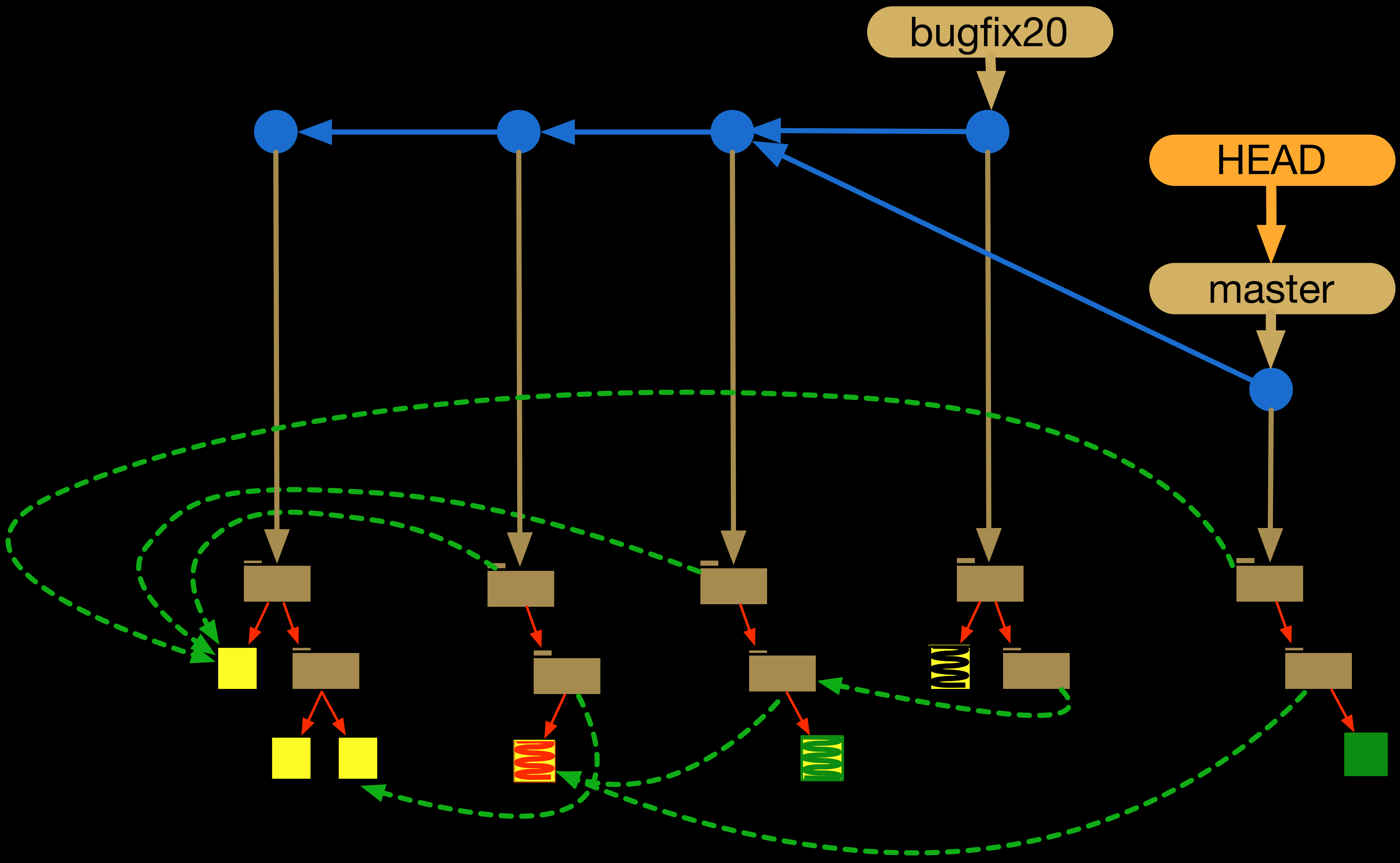

bugfix20
master
grade feature
HEAD

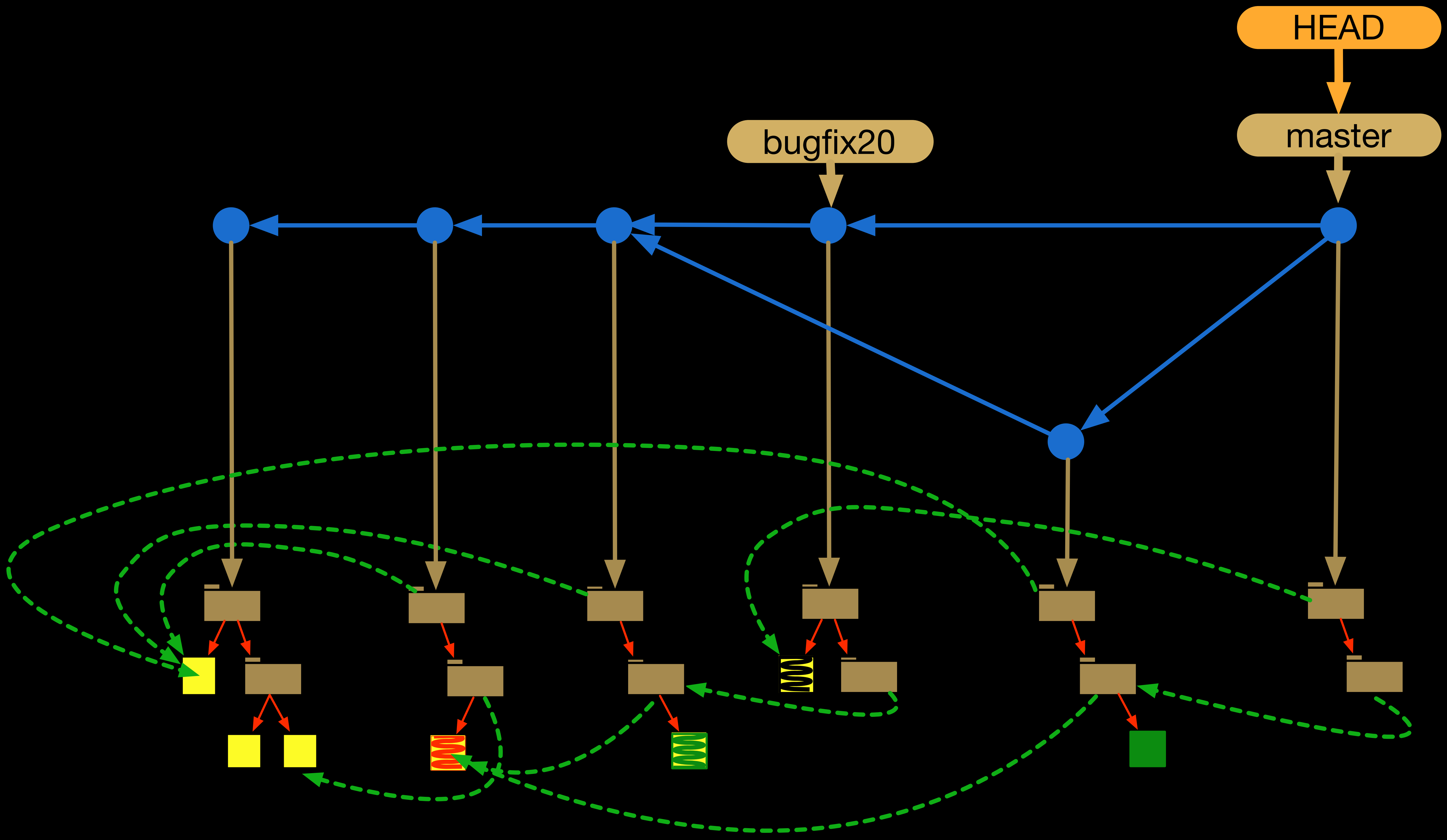

bugfix20
master
grade feature
HEAD


HEAD
master
bugfix20
grade feature

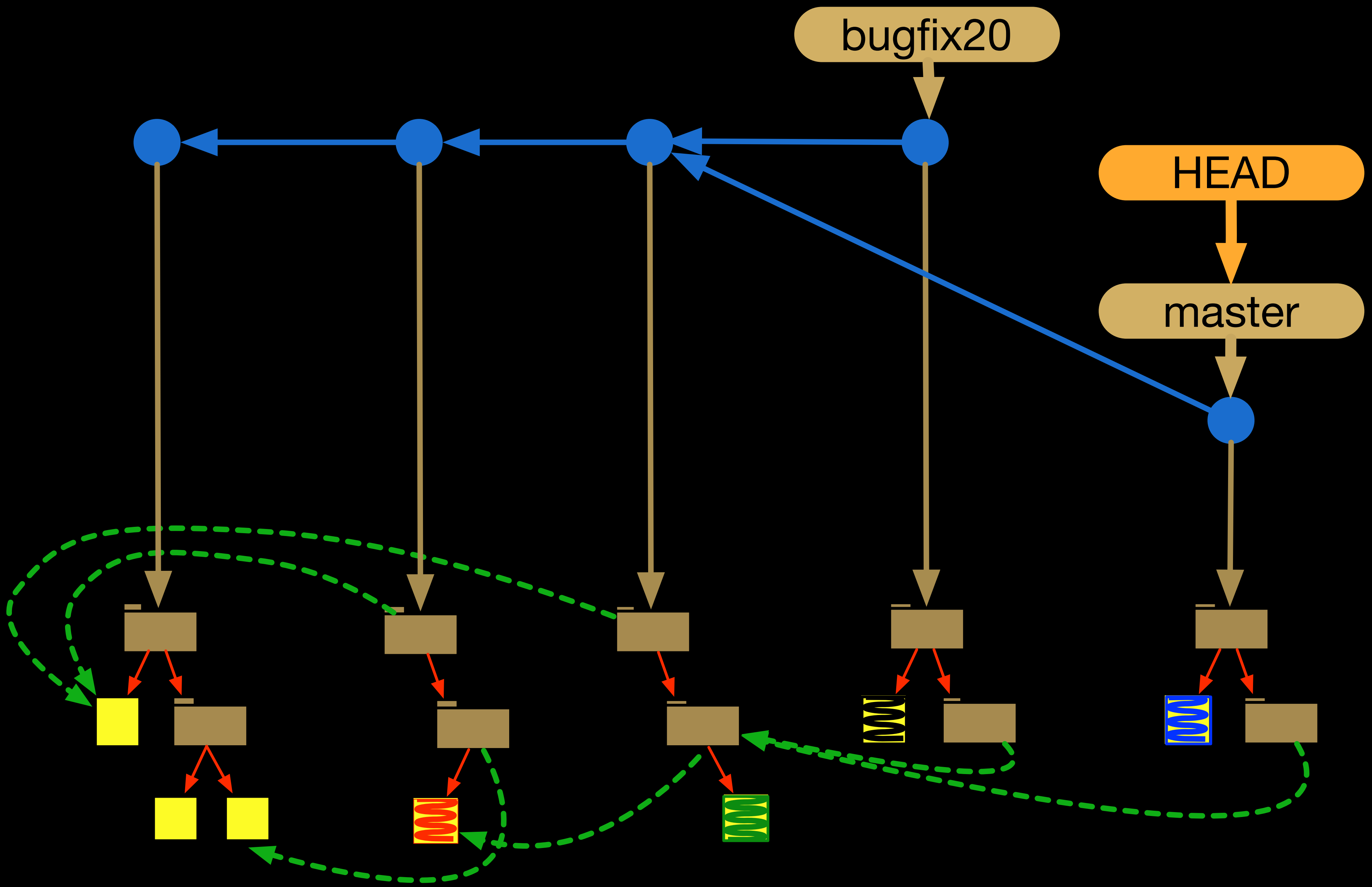
merge conflicts

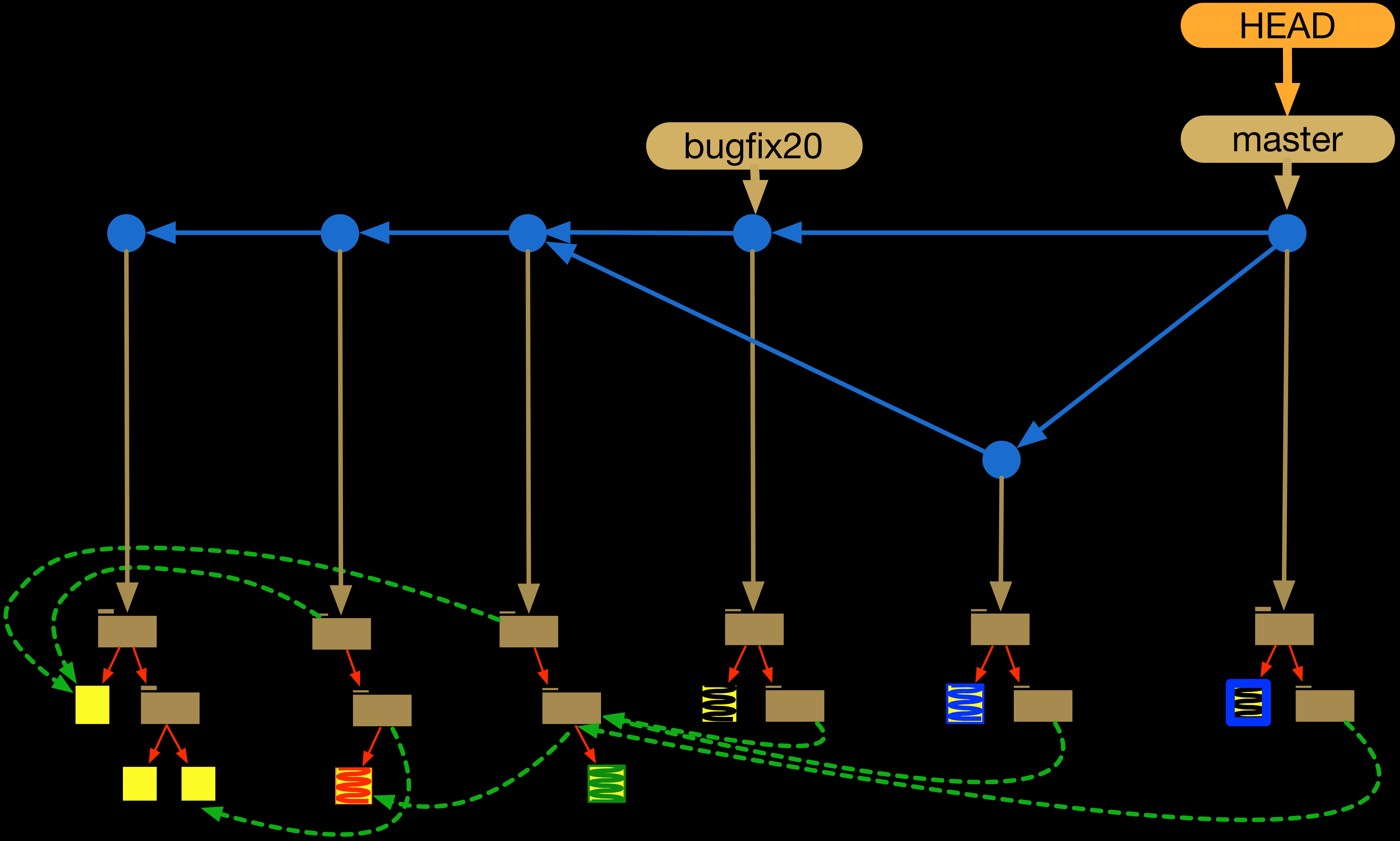
no problem when different files are changed


bugfix20
HEAD
master

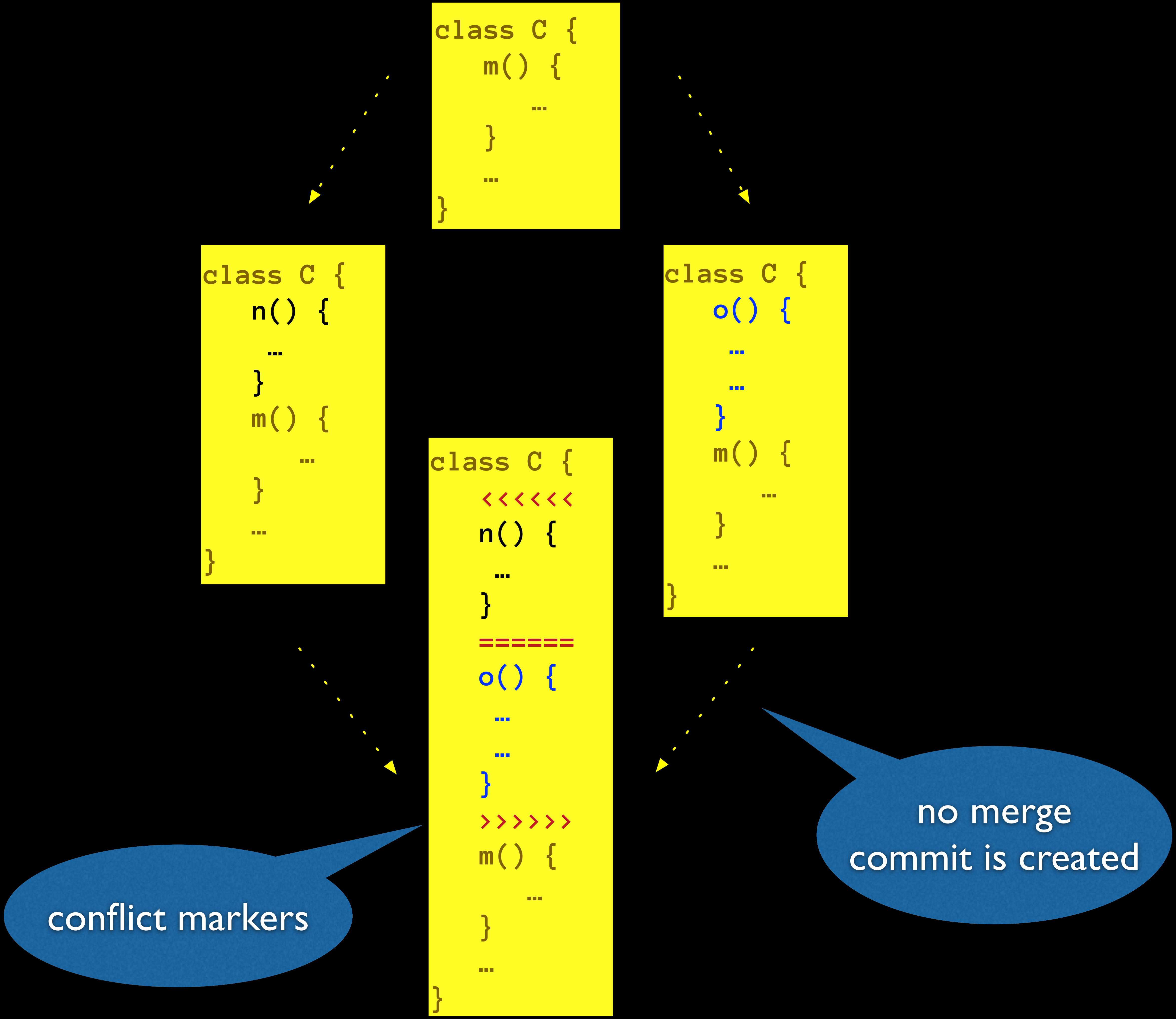

HEAD
bugfix20
master

no problem if the same files are changed in different areas


bugfix20
HEAD
master


HEAD
bugfix20
master

but conflict if the same files are changed in the same areas



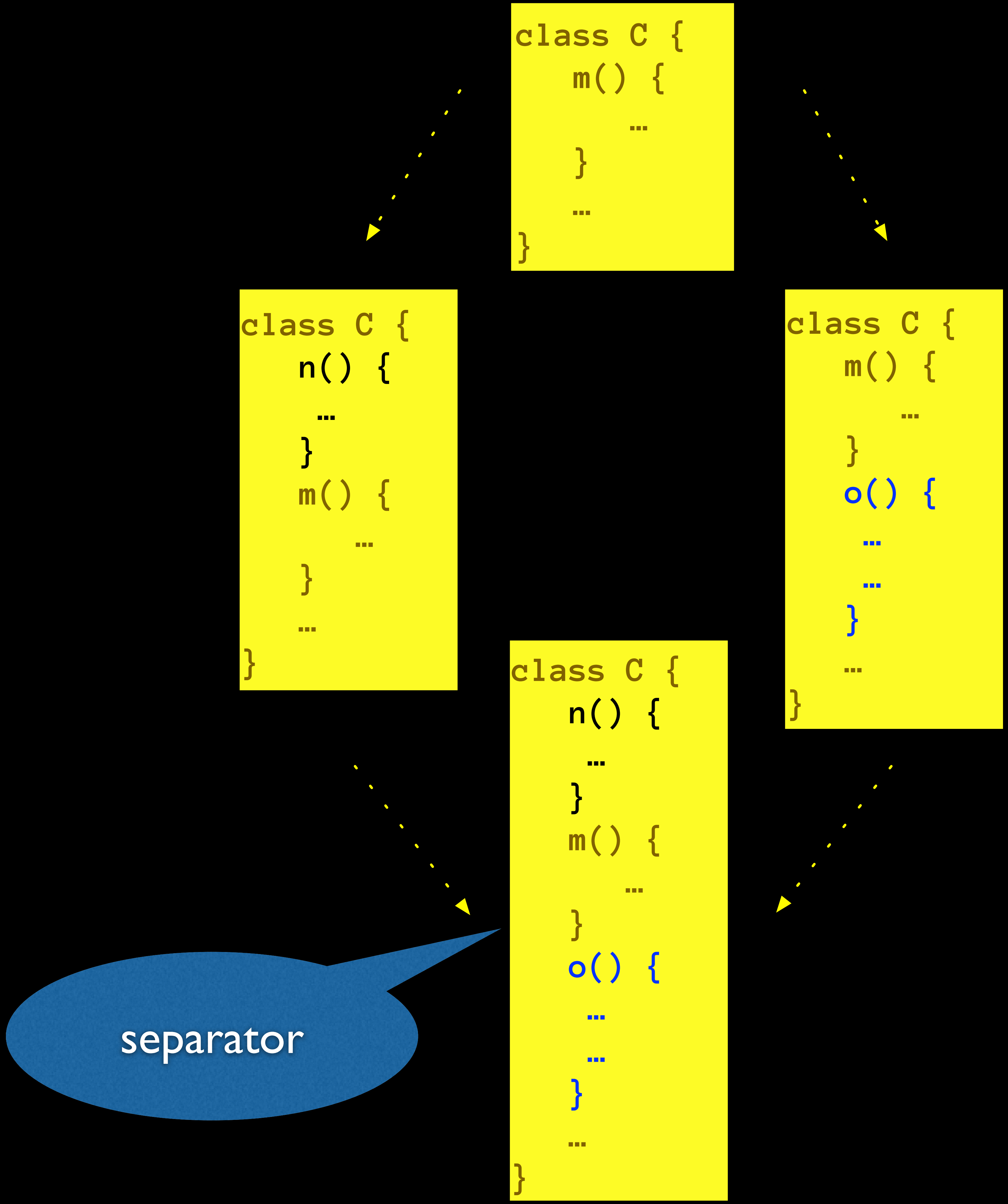
```
class C {
   m() {
      …
   }
   …
}
```

```
class C {
   n() {
    …
   }
   m() {
      …
   }
   …
}
```

```
class C {
   o() {
    …
    …
   }
   m() {
      …
   }
   …
}
```

```
class C {
   <<<<<<
   n() {
    …
   }
   ======
   o() {
    …
    …
   }
   >>>>>>
   m() {
      …
   }
   …
}
```

separators define areas



```
class C {
   m() {
      …
   }
   …
}
```

```
class C {
   n() {
    …
   }
   m() {
      …
   }
   …
}
```

```
class C {
   m() {
      …
   }
   o() {
    …
    …
   }
   …
}
```

```
class C {
   n() {
    …
   }
   m() {
      …
   }
   o() {
    …
    …
   }
   …
}
```

git merge does not detect semantic conflicts

```
m(x){
 if (x==null) return;
 y = x;
 if (x!=null) y = x.n();
}
```

```
m(x){
 if (x==null) return;
 y = x;
 y = x.n();
}
```

```
m(x){
  y = x;
  if (x!=null) y = x.n();
}
```

git merge

```
m(x){
 y = x;
 y = x.n(); ✗
}
```

Take notes, now!

Hands on exercises

Configuration management 2, advanced concepts and operations

Configuration management 3, continuous integration and deployment

Branches

Branches are used for supporting…

- independent development
  - feature or component development
  - bug fixing
- bug fixing in previous releases
- customer specific variants (not a good idea, better options available)

Branches isolate changes until they are...

- mature (do not break other functionality)
- chosen for a release (in master)
- ready to be updated with other changes (from master, without breaking development flow)
- reconciled with alternative functionality

But we should use with care

Risks of branching

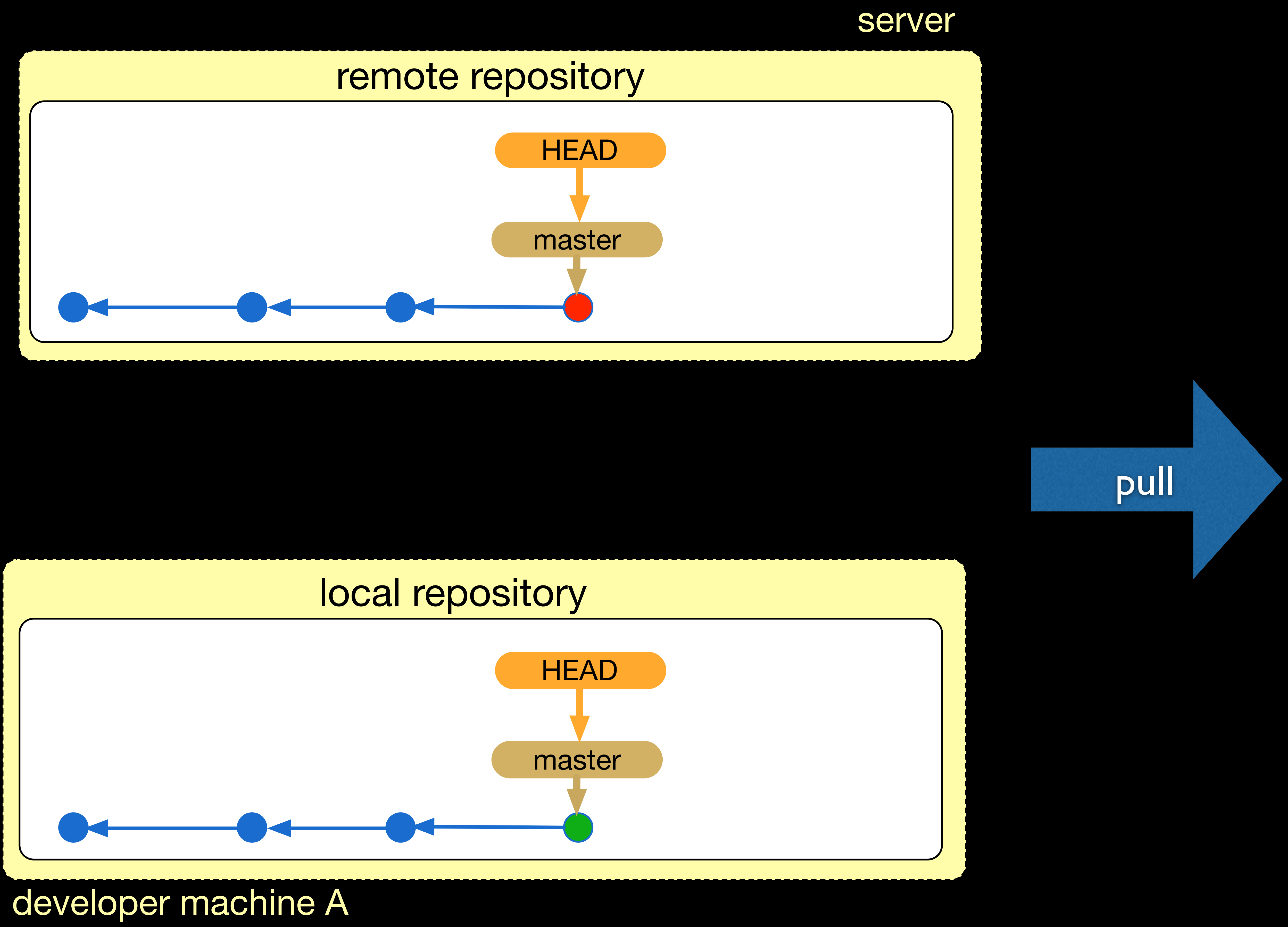
- isolation provided by branches reduce pressure to integrate changes (increase time to market)
- delayed integration
  - increase the risk of conflicts, which are harder to solve the latter they are detected
  - developers might avoid refactoring because of the risk of conflicts

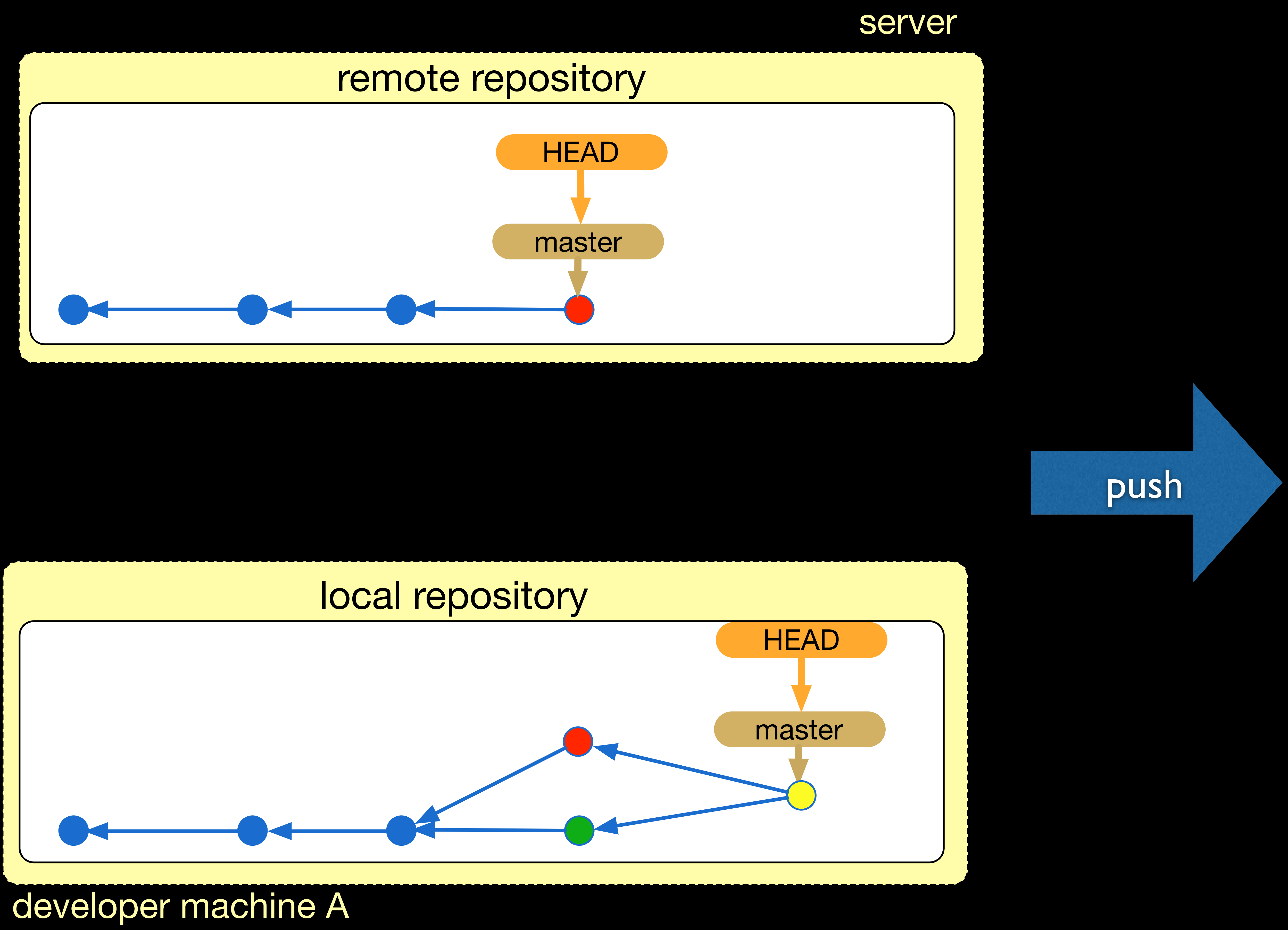
promiscuous— branch to branch — integration, or from master to branch, could reduce the risks

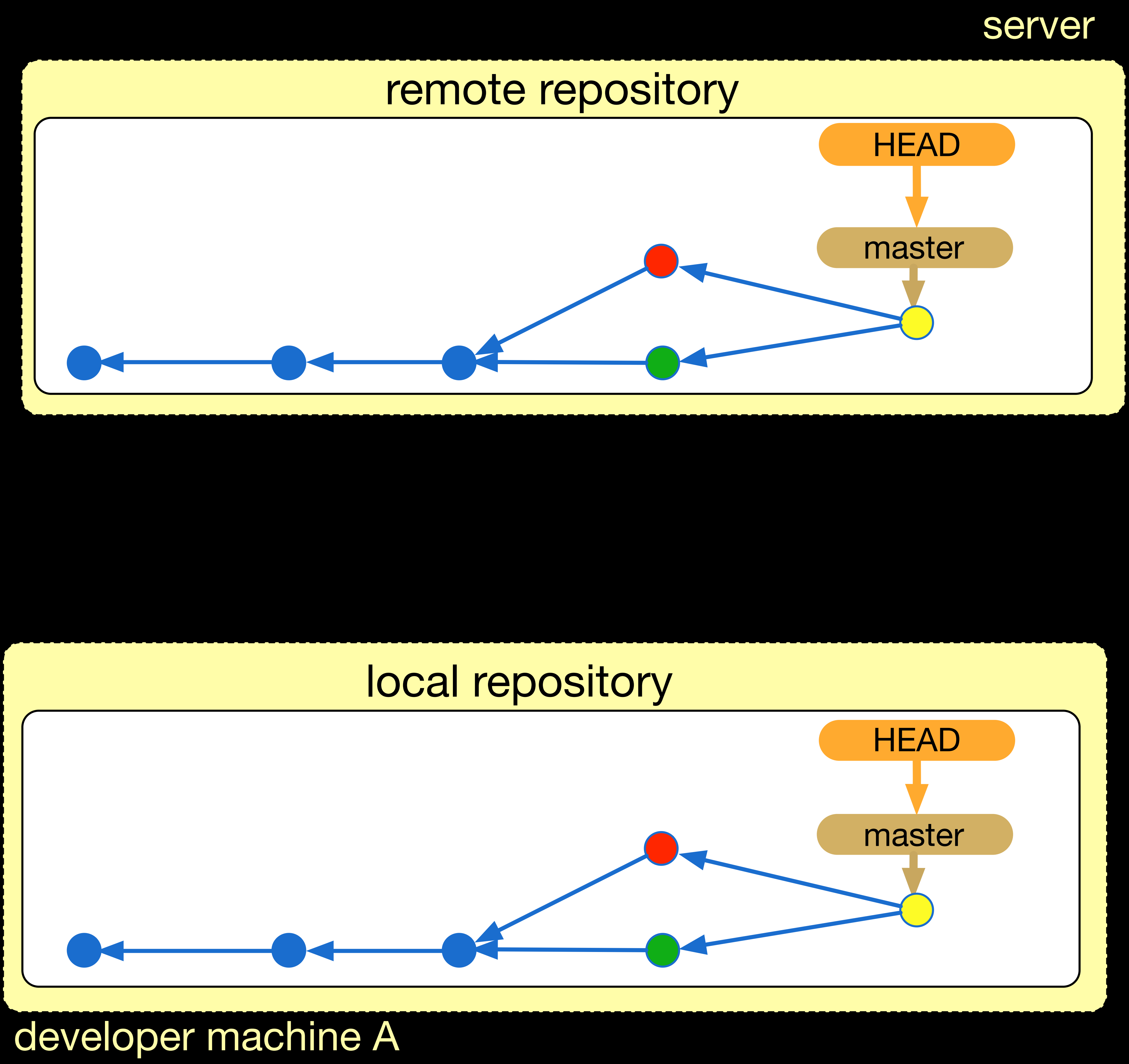
Code integration is not restricted to branches in a single repository

git pull
(fetch + merge)

merging with remote

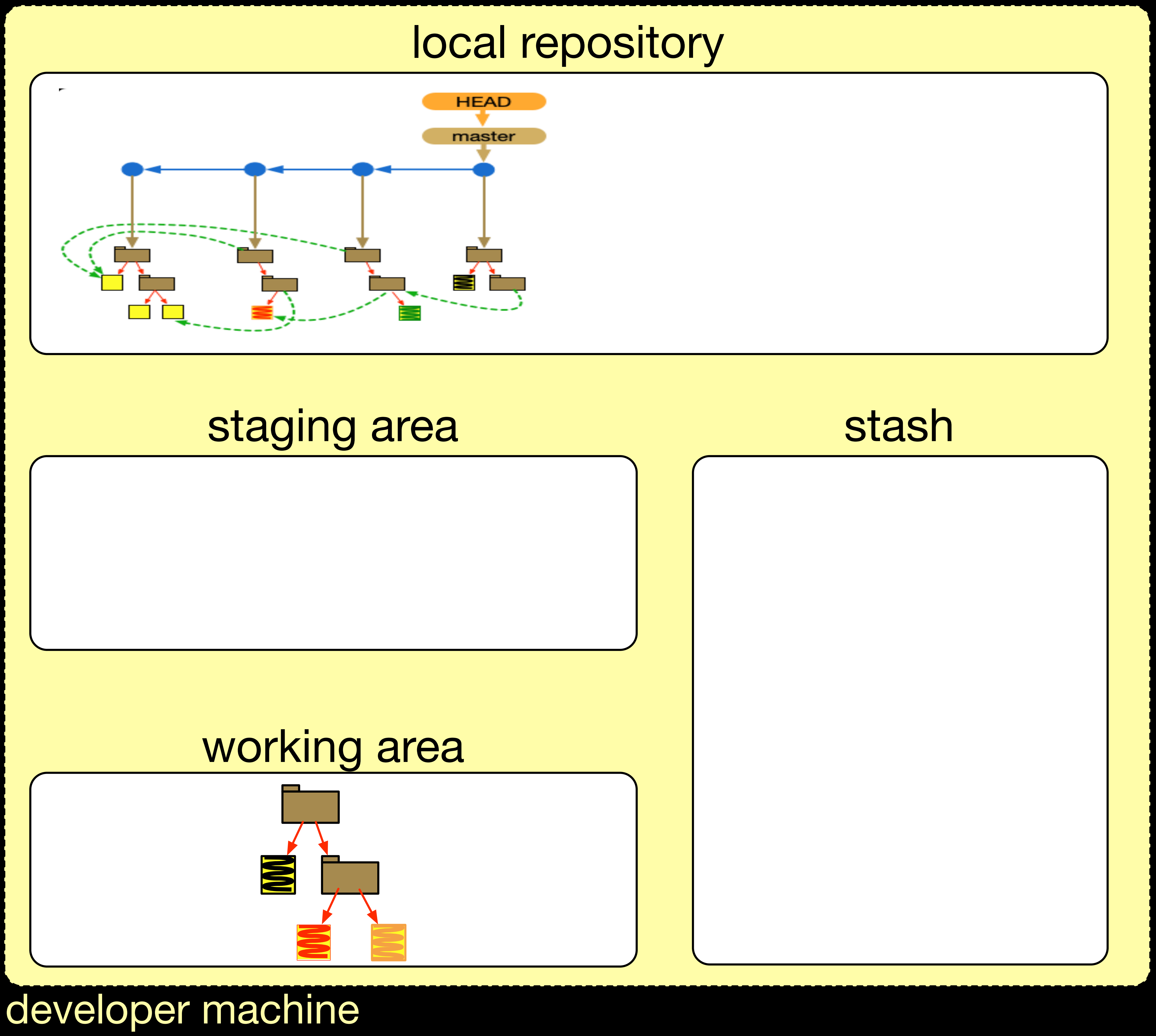

server
remote repository
HEAD
master
pull
local repository
HEAD
master
developer machine A

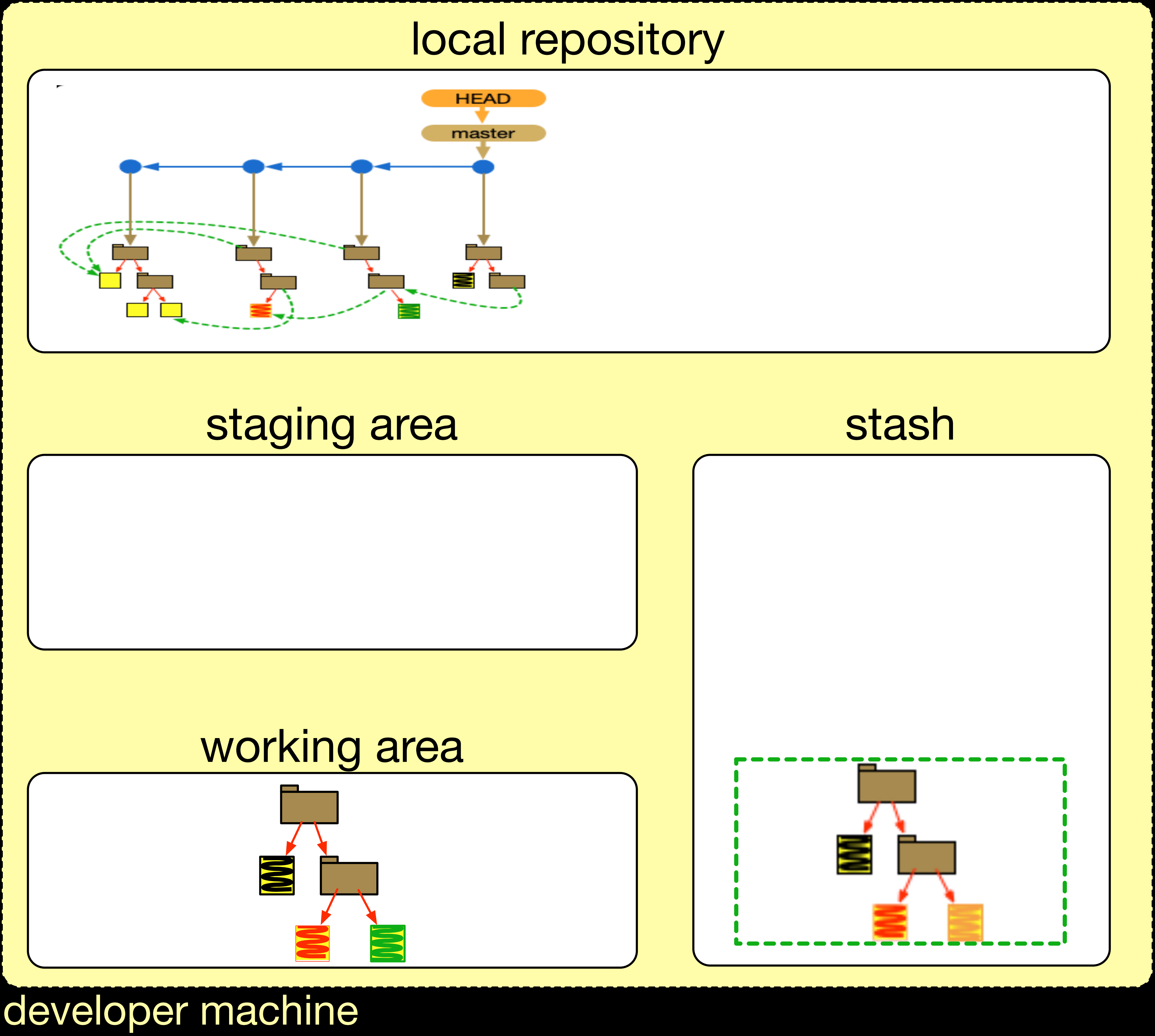

server
remote repository
HEAD
master
push
local repository
HEAD
master
developer machine A


server
remote repository
HEAD
master
local repository
HEAD
master
developer machine A

git stash

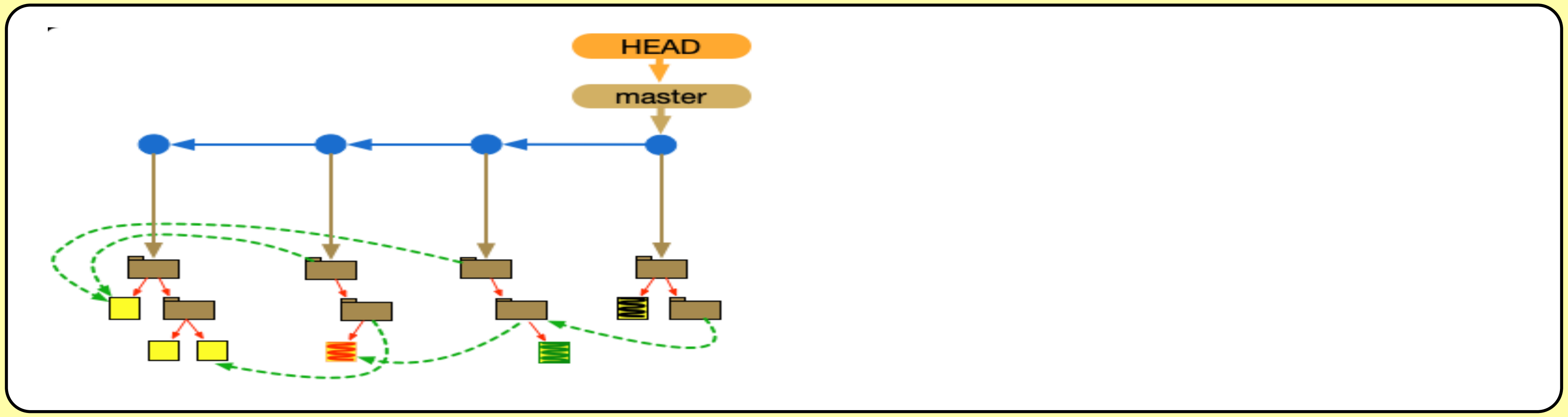
temporarily saving partial changes


local repository
HEAD
master
staging area
stash
working area
developer machine


local repository
HEAD
master
staging area
stash
working area
developer machine

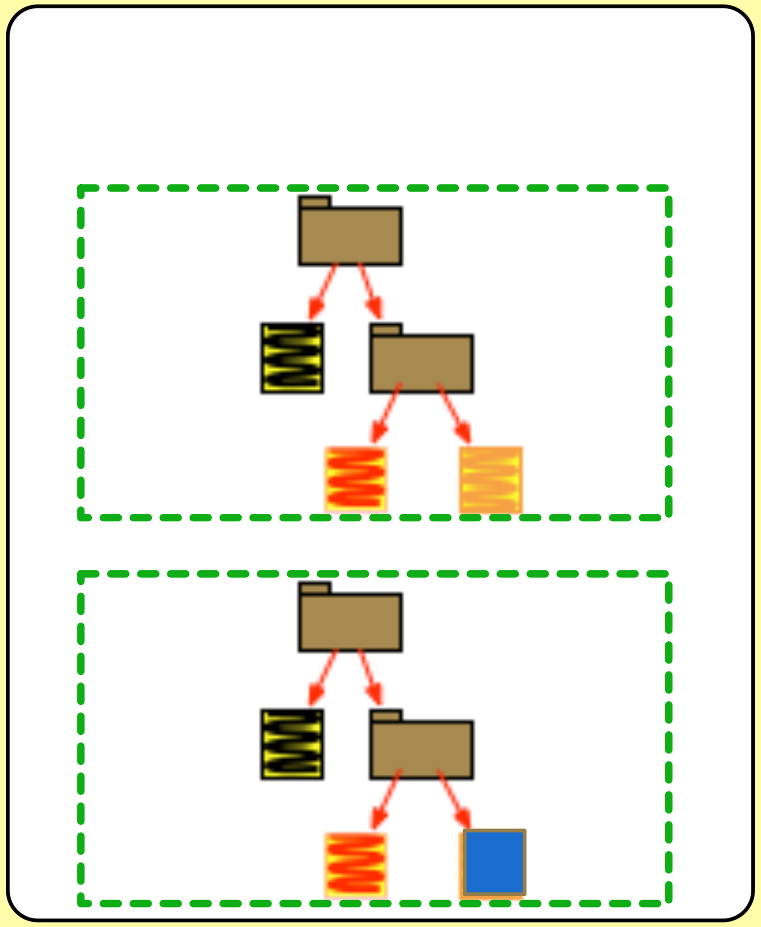
stash area is a stack

local repository

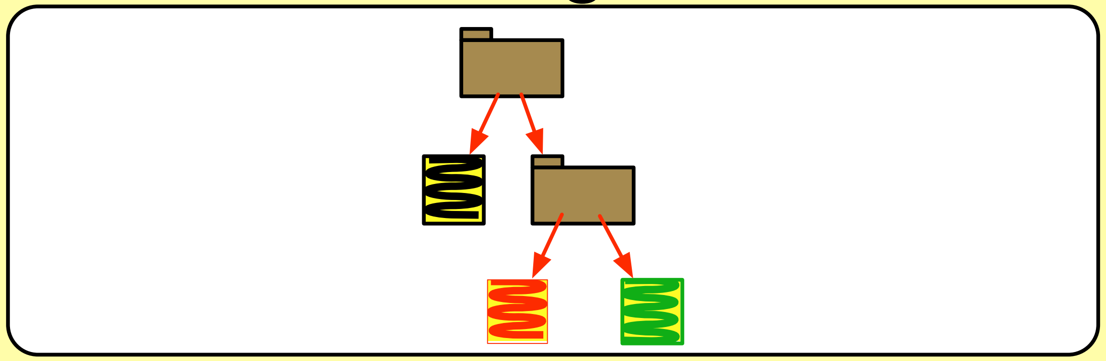


staging area

stash



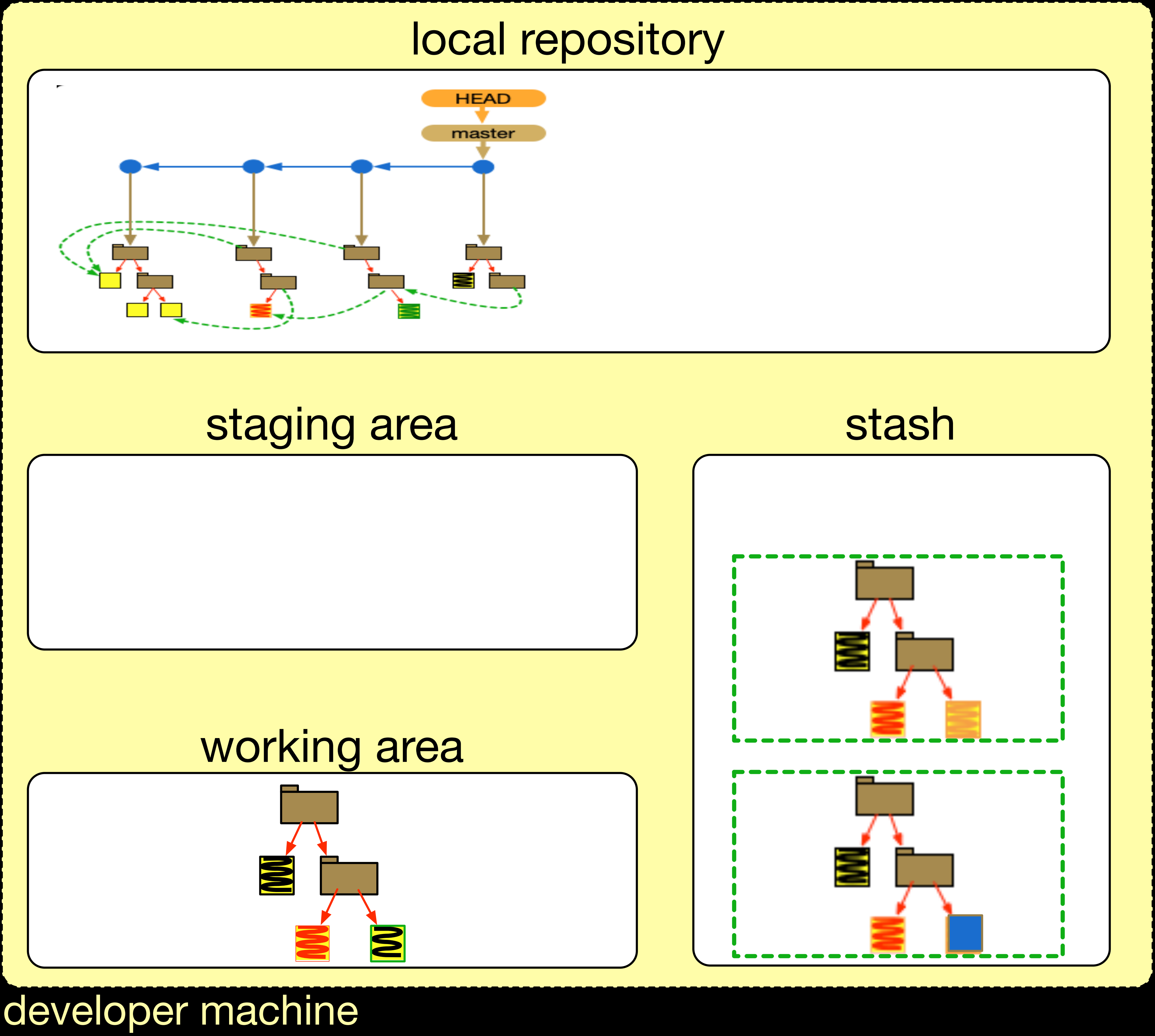
working area

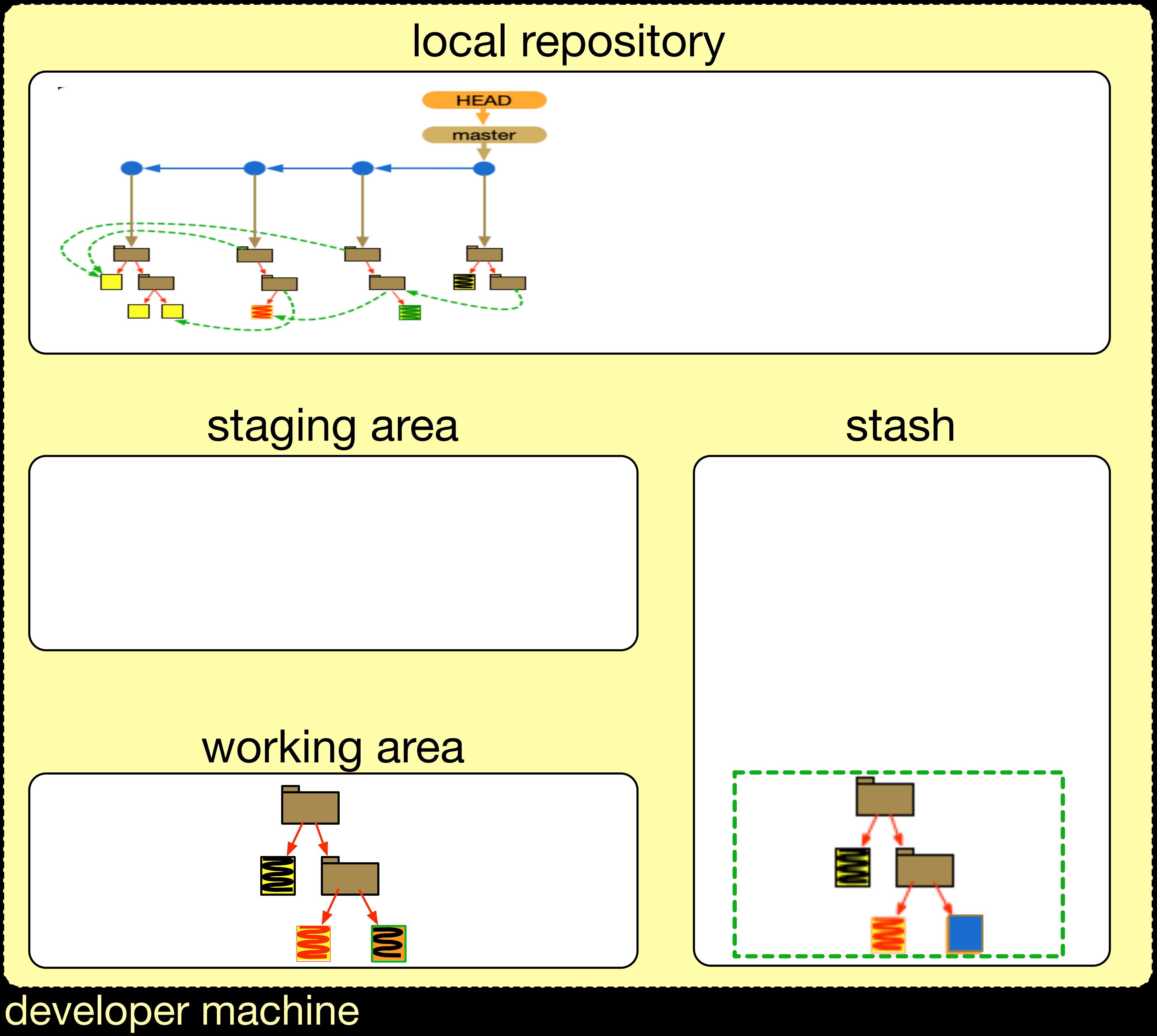


developer machine

git stash drop (pop)

git stash apply


local repository
HEAD
master
staging area
stash
working area
developer machine


local repository
HEAD
master
staging area
stash
working area
developer machine

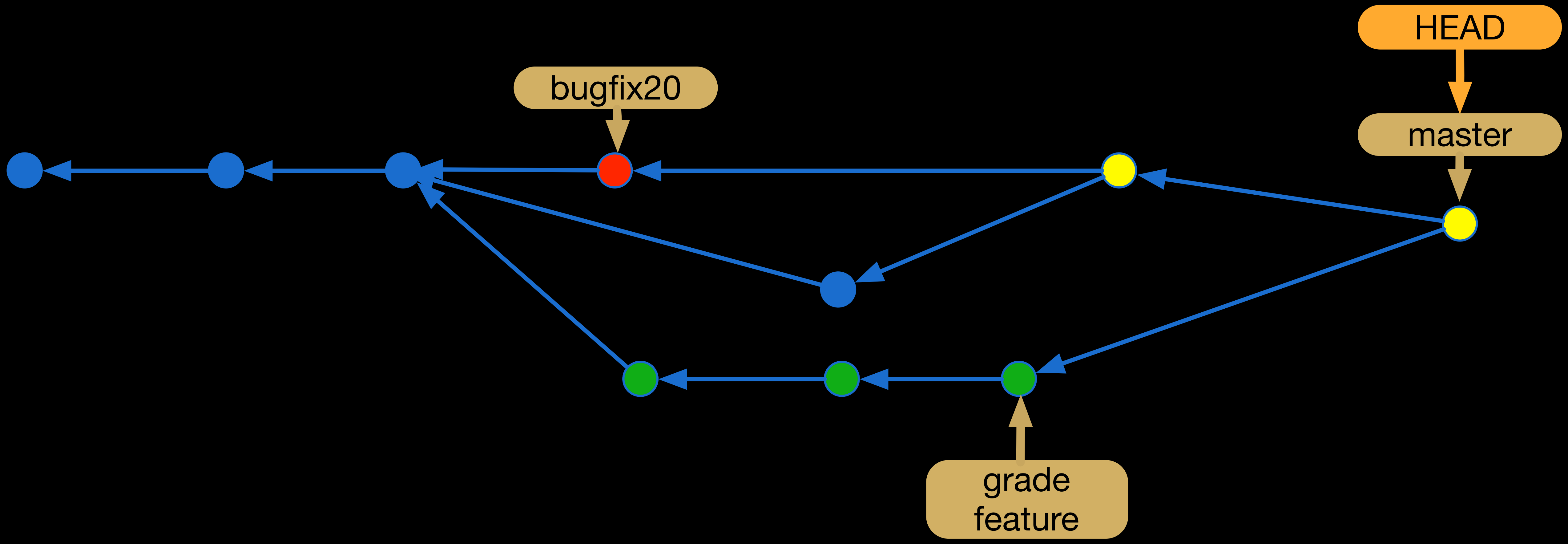
stash code is integrated to the working area

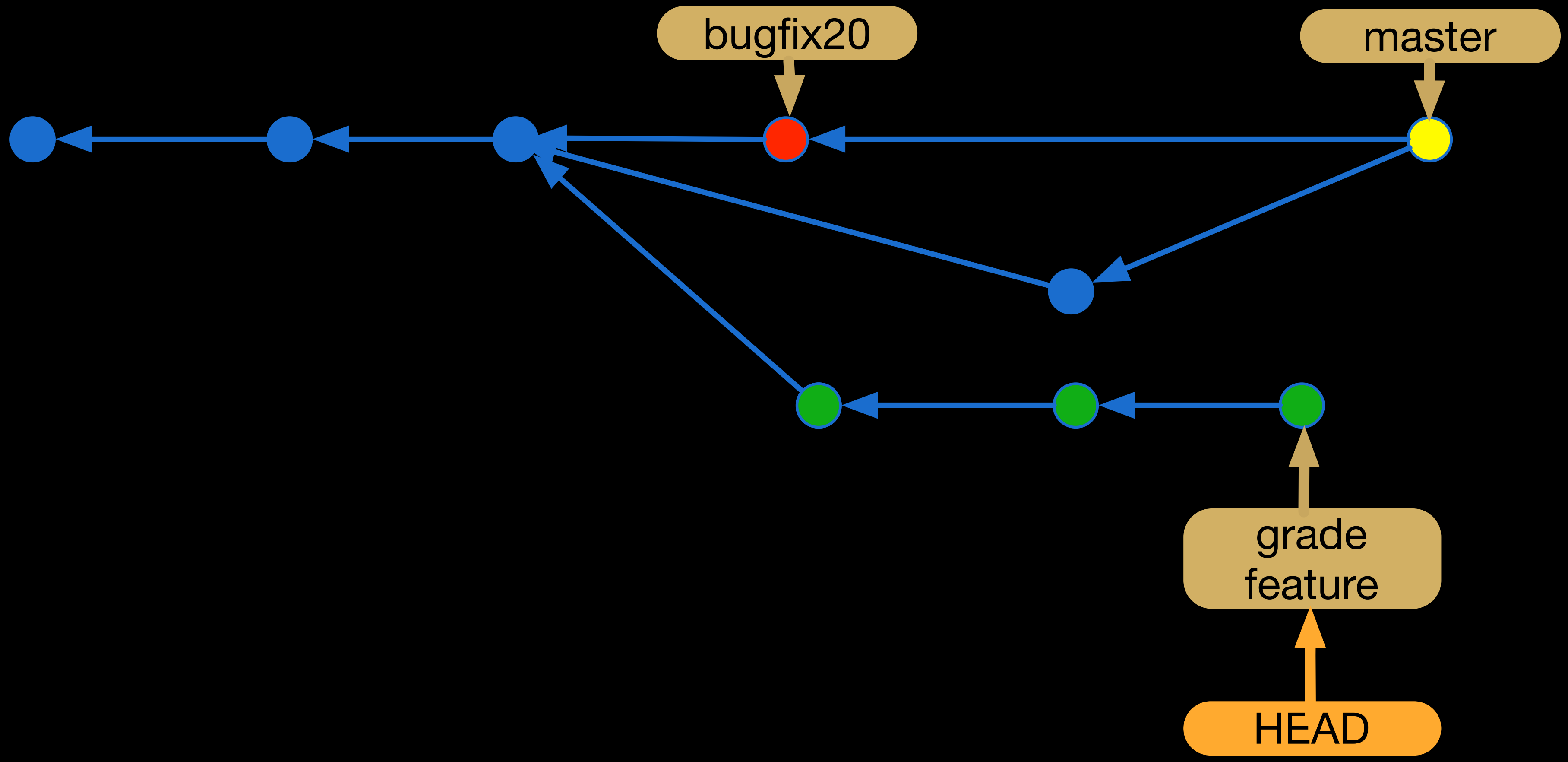
conflicts might arise

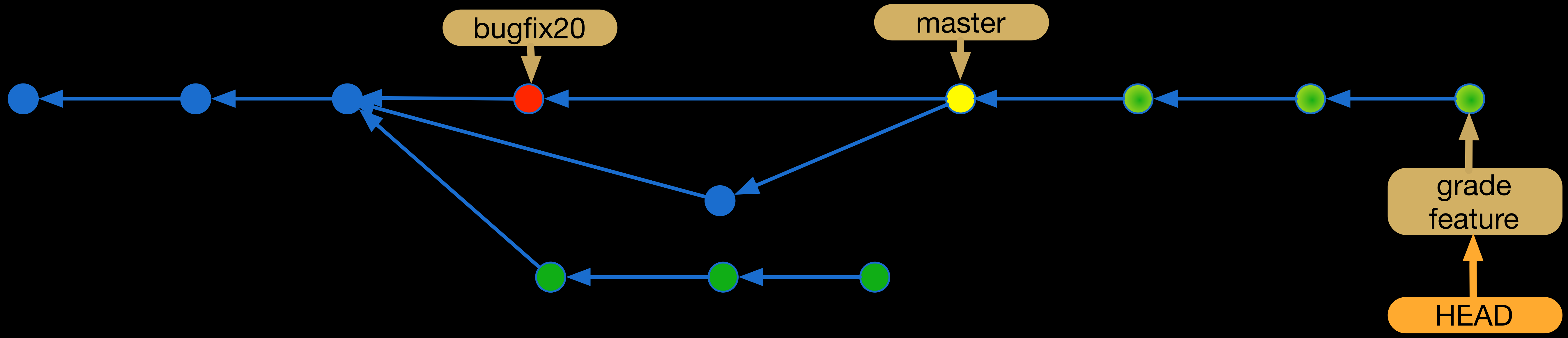
Other code integration commands

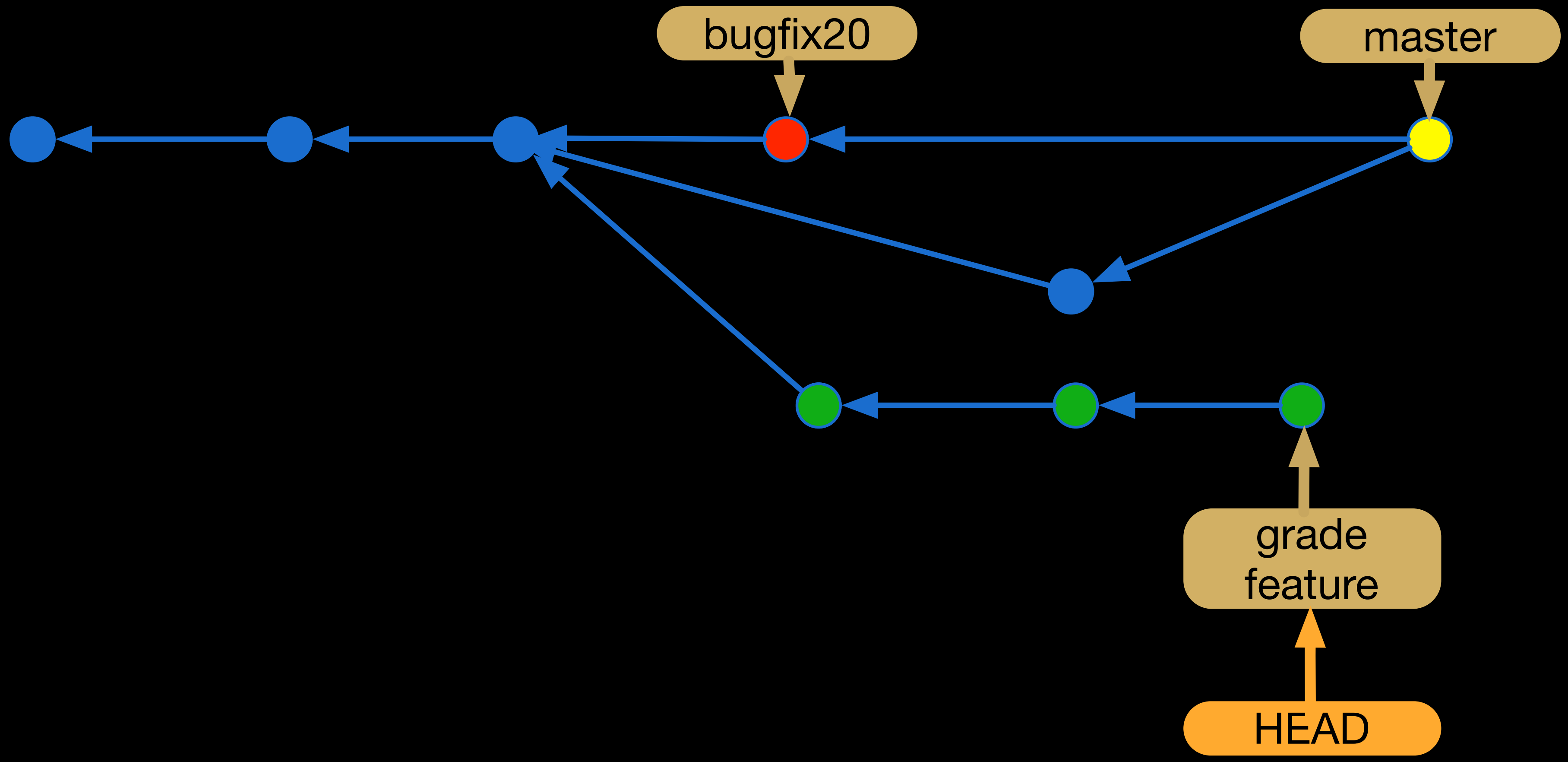
git rebase

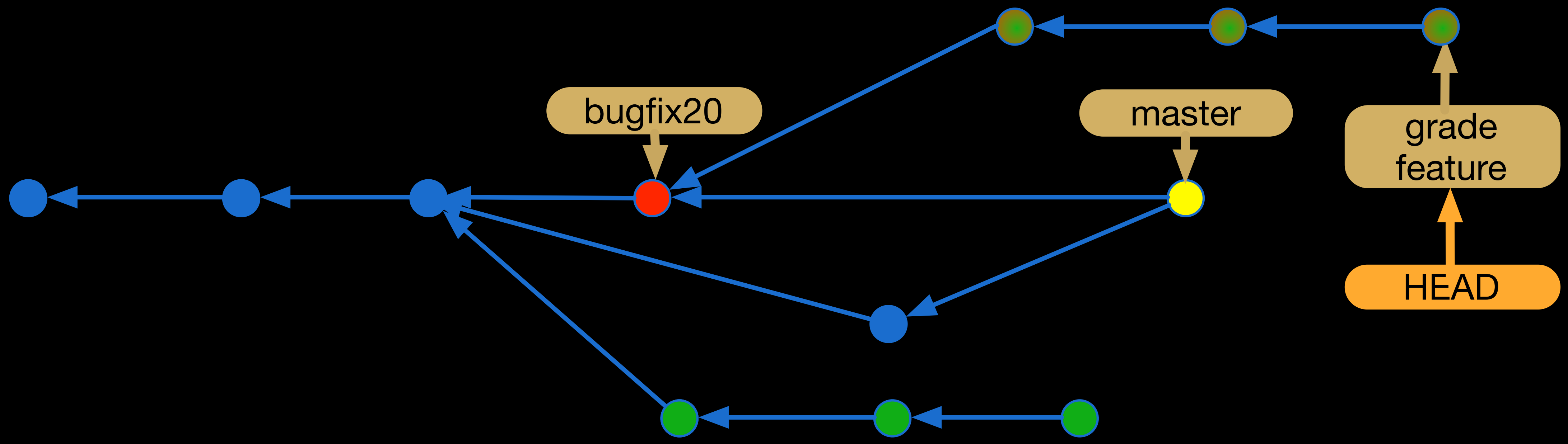
merge might complicate history




bugfix20
master
grade feature
HEAD


bugfix20
master
grade feature
HEAD

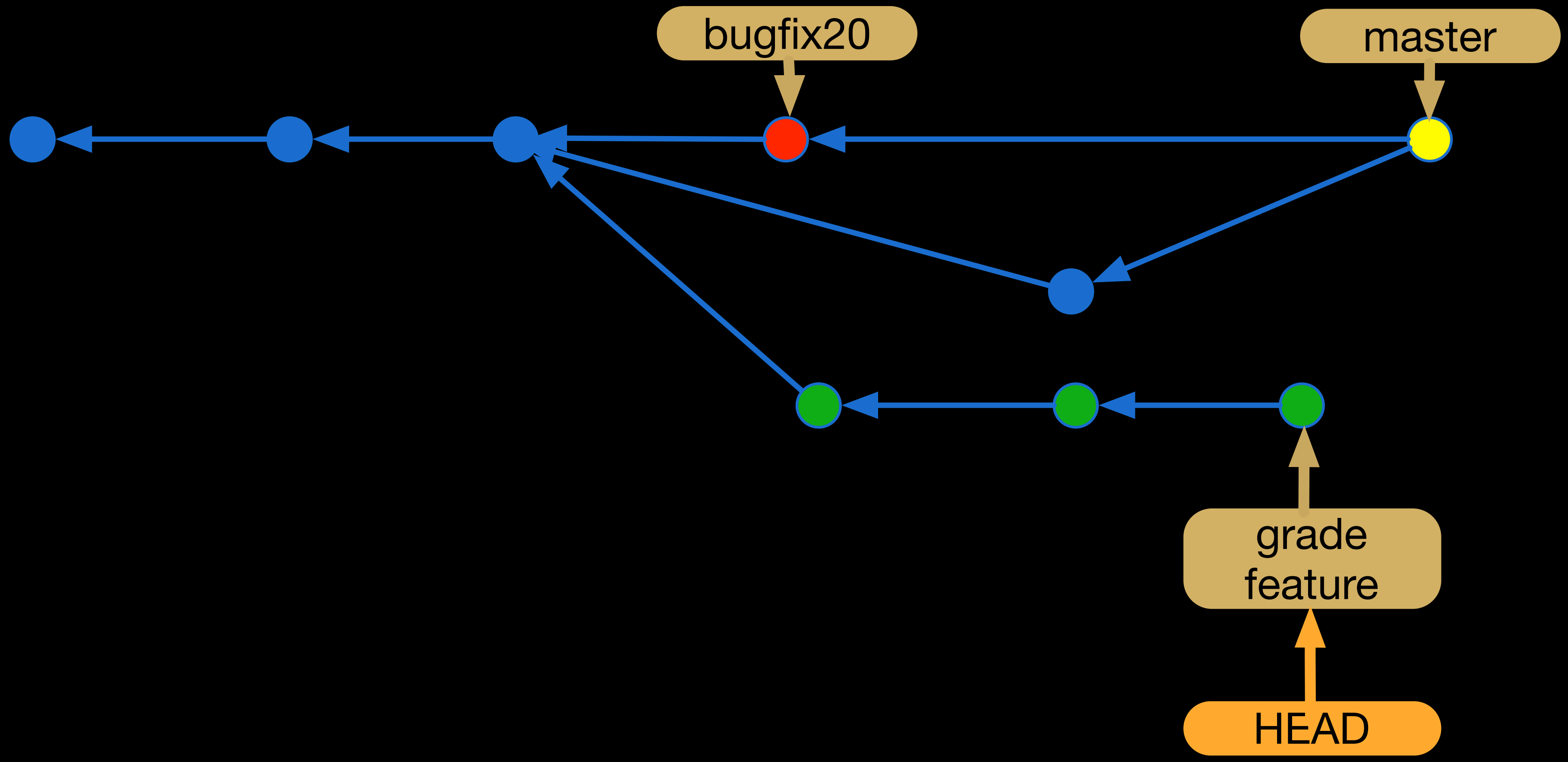

bugfix20
master
grade feature
HEAD

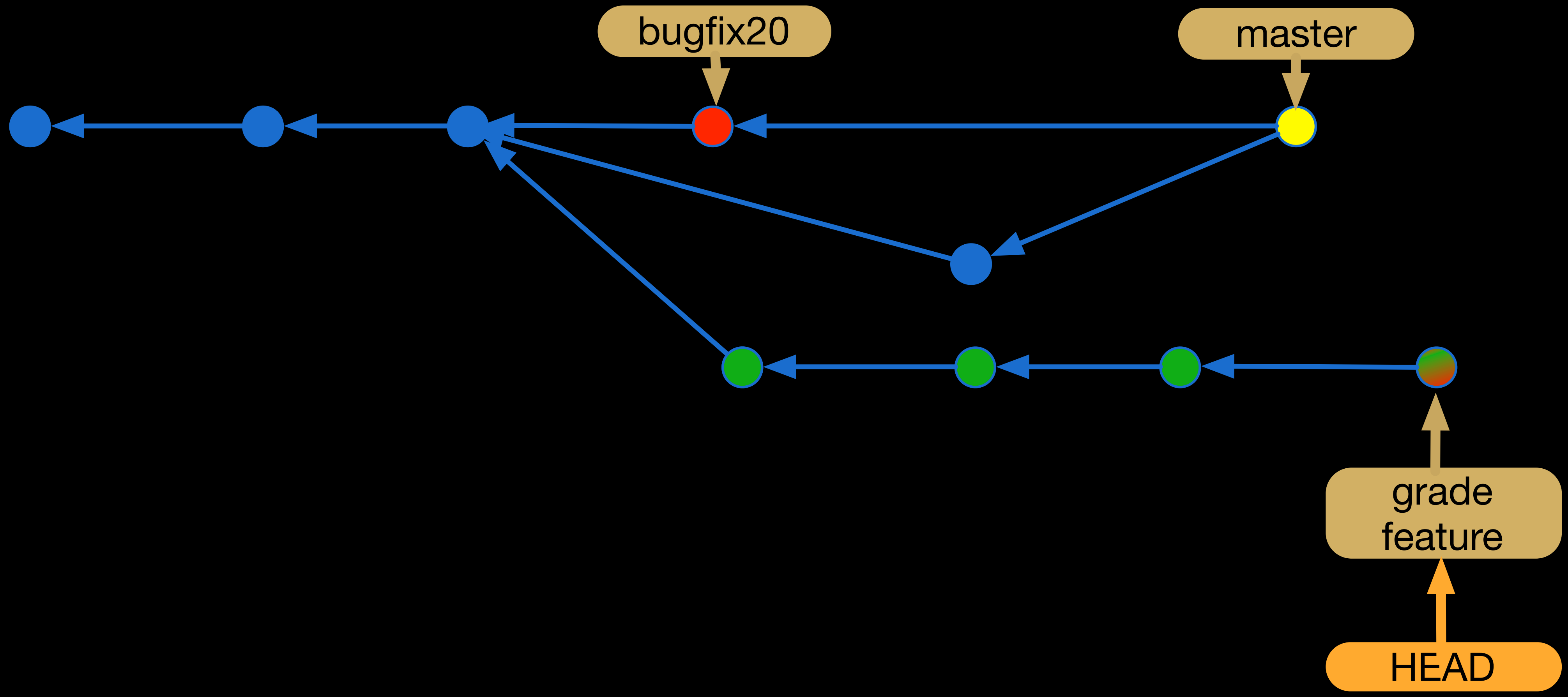

bugfix20
master
grade feature
HEAD

interactive rebase

squash

git cherry-pick

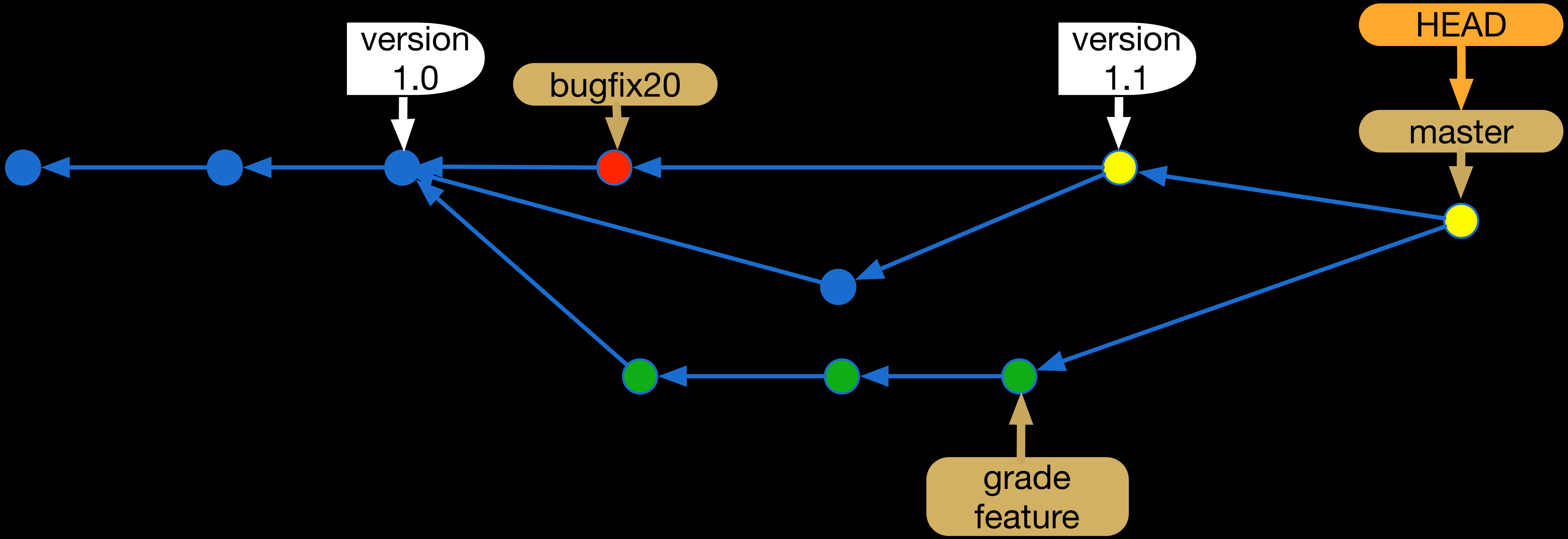

bugfix20
master
grade feature
HEAD


bugfix20
master
grade feature
HEAD

Advanced

git config

git tag


version 1.0
bugfix20
version 1.1
HEAD
master
grade feature

git reset

reset --soft X: makes HEAD and the current branch point to X (X could be HEAD~ or hash)

reset --mixed X (default): same as before plus updates stage (index) with X contents

reset --hard X: same as before plus updates working directory with X contents

Commit

- 1 hash: 40 hex characters (SHA)
- Parent commits, message, author/commiter, email, time
- Not diffs
- Root (Merkle) tree, contains names of files and folders inside (and file modes)
- Blobs (binary large objects; file contents, not names)
- git cat-file -p commit_hash (to see commit contents, all with identifiers based on SHA)

Summary of git concepts

Workflows

- Creating repositories (locals, remotes, bare, fork, clone)
- Recording changes (working area, stage, tracked, ignored, assume unchanged, commit)
- Branching (pointer, master, HEAD, stash)

Workflows

- Tagging (pointer not updated by commit)
- Integrating changes (merge, rebase, cherry-pick, conflicts, squash)
- Synchronizing repositories (push, pull, PR, remote branch, upstream branch)

Storage areas

- Non-bare Repository
- Bare Repository
- Commit
- Staging Area
- Stash

References

- Head
- Tag
- Branch
- Upstream Branch (sync default)
- Remote Branch

File classifications

- Tracked
- Untracked
- Ignored
- Assume Unchanged
- In Conflict

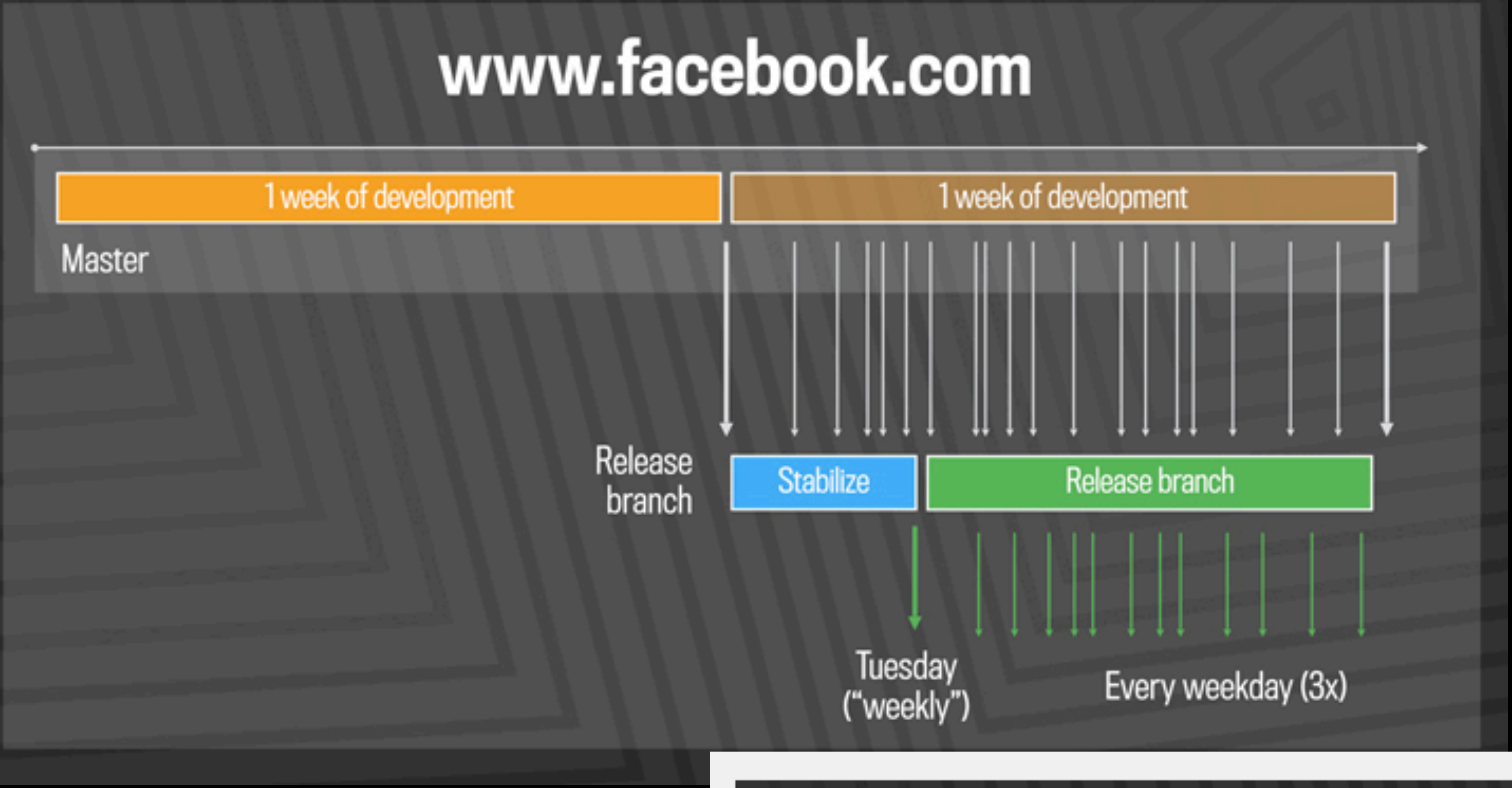
at least daily

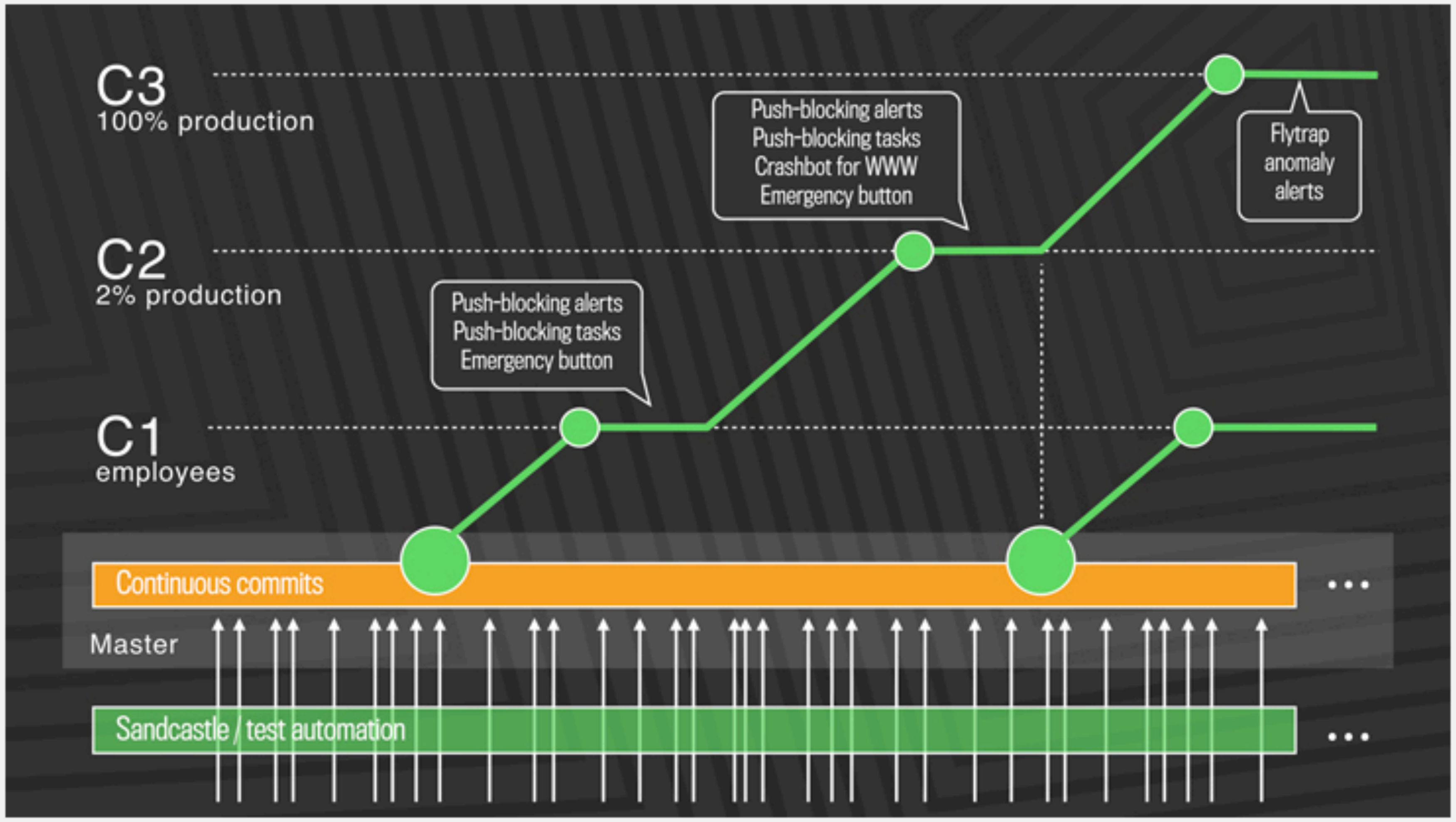
Continuous

building and testing

integration

deployment





https://code.fb.com/developer-tools/rapid-release-at-massive-scale/?utm_content=bufferb0e59&utm_medium=social&utm_source=twitter.com&utm_campaign=buffer

Advantages and disadvantages

- No need for hotfixes
- Better support for a global engineering team
- Scales (number of developers, commits, etc.)
- Conflicts are detected early
- High quality, confident developers

- Might break development flow (due to code that might not even be released)
- Need for Gatekeeper system
    - feature flags might pollute the code
    - part of the functionality might become dead code

Before integrating…

- Review scenarios and features, or write scenarios in pairs
- Make sure they follow style and quality criteria
- Organize review procedures and meetings

Checklist

Checklist

- Clear commit messages, exactly corresponding to what has changed
  - no language errors
- Commit early and often
  - avoid tangled commits
  - aim for conceptual, modular changes
- Often integrate your code (small integrations, by merging to or from master)

Checklist

- Antes de enviar um pull request, deve-se, obrigatoriamente, fazer um merge ou rebase para que suas modificações sejam aplicadas à versão mais atual do repositório central do seu projeto, não à última versão que você tinha na sua máquina e com informações desnecessárias.
- A mensagem do pull request deve descrever com precisão e clareza o seu conteúdo.
- Cada pull request deve incluir somente os arquivos que foram alterados para a atividade.

Checklist

- Pull requests não devem conter arquivos gerados ou de configuração particular de cada máquina ou IDE, como .classpath, .project, .class. O mesmo vale para arquivos de log e similares. Não deve-se dar commits nesses arquivos. Use o .gitignore para evitar esse problema.
- Evite pular linhas, adicionar espaços, etc. em partes do código que você não precisa modificar; cada mudança desss precisa ser analisada por quem vai integrar a sua contribuição ao repositório principal, e podem gerar conflitos.
- Tenha certeza de que todos os testes estejam passando na versão incluída no pull request.

Change requests and change managements

Issues or change requests

- Bug report
- Need for new feature or scenario
- Need for changing a particular feature
- Need for code improvement
- Need for performance improvement
- Chore description (provide no direct value to the customer)

Take notes, now!

Hands on exercises

Configuration management 3

CM research at CIn

- Advanced merging tools: Paulo
- Collaboration conflicts characteristics: Paulo

To do after class

- Answer questionnaire (check classroom assignment), study correct answers
- Finish exercise (check classroom assignment), study correct answers
- Read, again, part of chapter 10 in the textbook
- Evaluate classes
- Study questions from previous exams

To do after class

- Use Pro Git, and Git Cheat Sheet as references
- Read either Git concepts simplified or A Visual Git Reference
- Opcionalmente, ler artigo que descreve um workflow colaborativo, e outro artigo que adiciona mais alguns detalhes
- Opcionalmente, instale e use uma ferramenta que ajuda na resolução de conflitos

Questions from previous exams

- Explique brevemente (a) o que é um branch em um sistema de controle de versão, e (b) porque você criaria um em seu projeto. Explique também (c) o propósito e o funcionamento da operação de merge.
- Podemos tanto fazer merge de mudanças do branch principal em um branch de feature (atualizando o branch de feature) quanto fazer merge de mudanças de um branch de feature no branch principal (atualizando o branch principal)? Explique brevemente sua resposta.