Datorer och datoranvändning
Versionshantering och Git

Mattias Nordahl

mattias.nordahl@cs.lth.se

Vad är versionshantering?

- Har du någonsin...
  - Mejlat filer fram och tillbaka med ändringar?
  - Förlorat en viktig fil eller version av ditt arbete?
  - Skrivit över en fil av misstag?
  - Behövt hålla koll på olika versioner av samma dokument?
- Versionshantering hjälper dig att lösa dessa problem!

Versionshanteringssystem

Vad är ett versionshanteringssystem?

Ett verktyg för att göra versionshantering enklare, genom att spåra och hantera ändringar i projektfiler över tid.

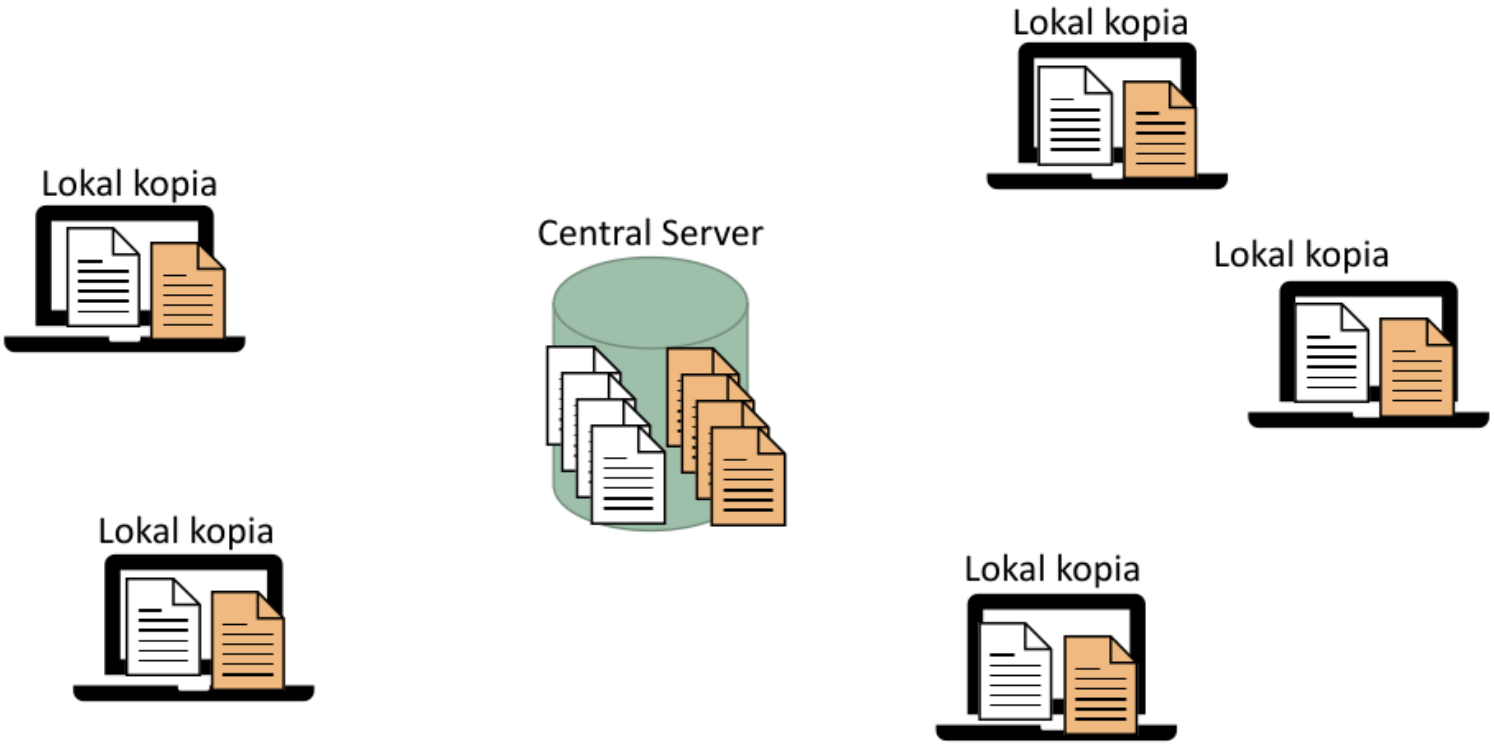
- Spårbarhet – vem som gjort vad och när. T.ex.:
  - “Vad har ändrats sedan förra veckan?”
  - “Vem la till den här raden?”
- Möjlighet att återgå till äldre versioner av filer vid behov.
- Möjliggör parallellt arbete och konflikthantering.

Typer av versionshanteringssystem (1/2)

Centraliserade system

- En central server lagrar alla filer och versioner. (*repository*)
- Användare hämtar **en** version, ändrar, och sparar tillbaka den.
- Kommandon: *checkout* (hämta filer), *commit* (spara till servern).
- Exempel: Subversion (SVN), Perforce

Centraliserad versionshantering

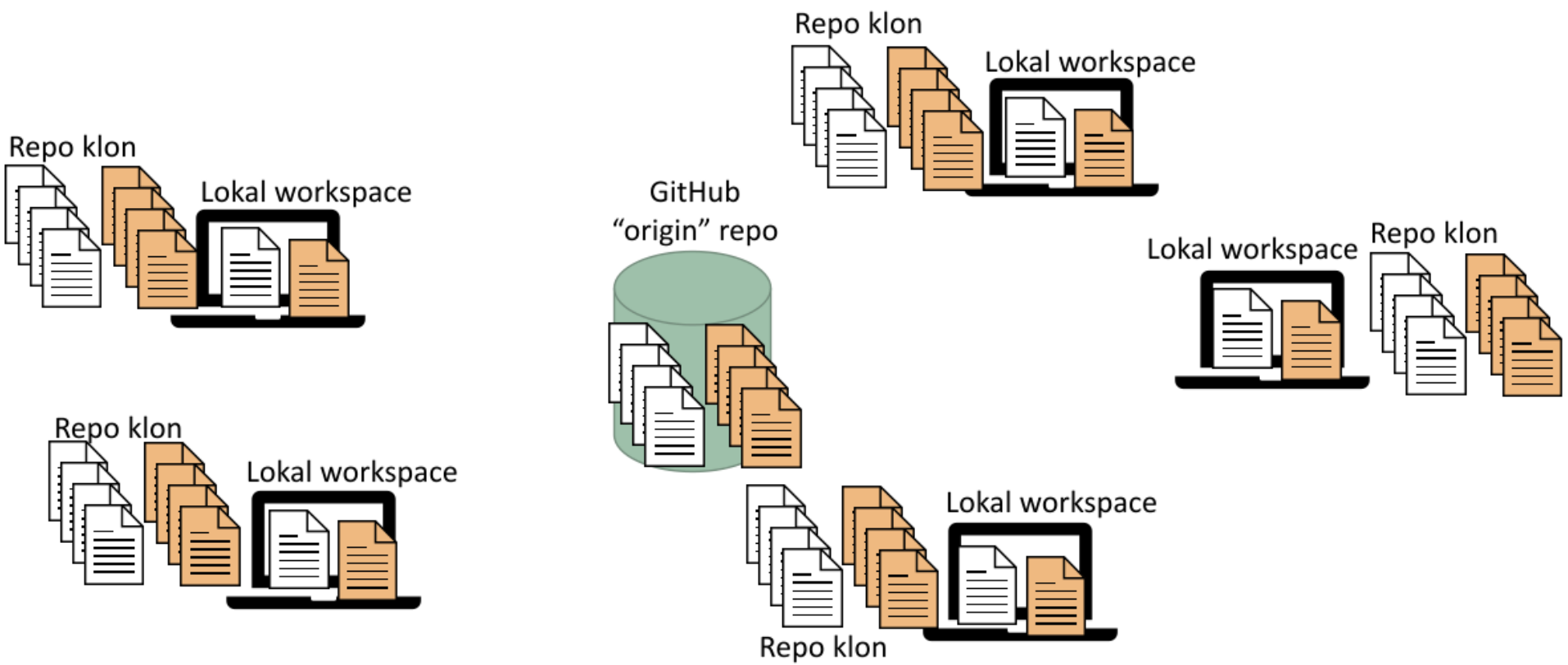


Typer av versionshanteringssystem (2/2)

Distribuerade system

- Ingen central server – varje användare är en “nod” i systemet.
- Varje användare har en lokal kopia av hela projektets historik.
- Kommandon: *clone* (kopiera hela repository), *checkout* och *commit* (men annorlunda), och *push/pull* (synkronisera ändringar).
- Exempel: Git, Mercurial

Distrubuerad versionshantering



Introduktion till Git

- *Git* är ett distribuerat versionshanteringssystem.
- Utvecklat av Linus Torvalds 2005. `https://github.com/git/git`
- Designat för snabb hantering av projekt med många bidragsgivare.
- Används av de flesta moderna mjukvaruprojekt och open source-projekt.
- Ger full kontroll över historik, branchning, och sammanslagning av ändringar (*merging*).

Testa: `git rev-list --max-parents=0 HEAD`

Git vs. GitHub: Vad är skillnaden?

Git

- Ett *verktyg* för versionshantering som körs lokalt på din dator.

GitHub

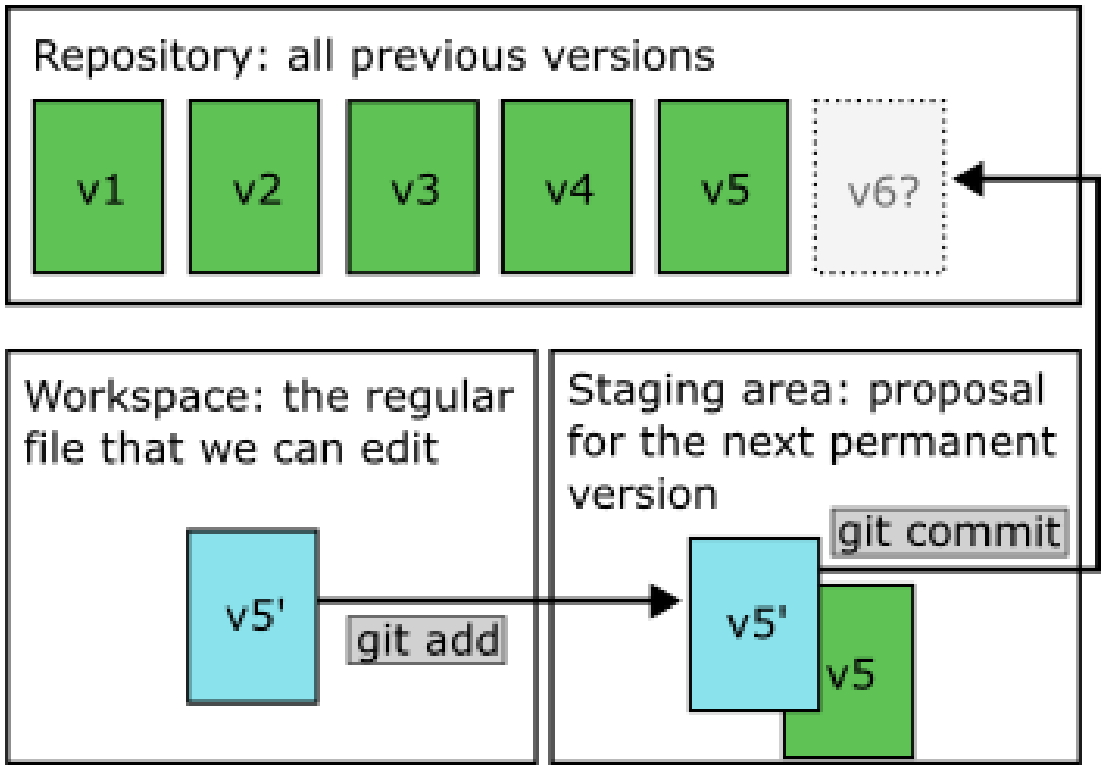
- En *tjänst* för att hosta Git-repositories online.
- Underlättar samarbete med andra utvecklare genom att erbjuda funktioner som pull requests, issues och projektöversikt.
- Använder Git internt för att hantera repositories, precis som lokala installationer av Git.

Grundläggande Git-kommandon

git init	Initierar ett nytt Git-repository lokalt.
git clone	Kopierar ett existerande repository (t.ex. från GitHub) till din dator.
git add	Lägger till ändringar i staging-området inför nästa commit.
git commit	Sparar ändringar lokalt med en beskrivande kommentar.
git push	Laddar upp dina lokala ändringar till ett fjärrrepository (t.ex. GitHub).
git pull	Hämtar och integrerar ändringar från ett fjärrrepository.

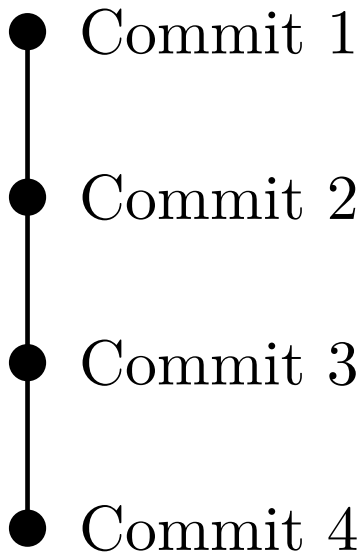
Git-modellen

- *Workspace*: En vanlig mapp på datorn där du jobbar med filer.
- *Staging Area*: En ”kopia” av workspace där du förbereder ändringar inför nästa sparning.
- *Repository*: Ett arkiv som lagrar alla tidigare versioner av projektet.



Vad är en Commit?

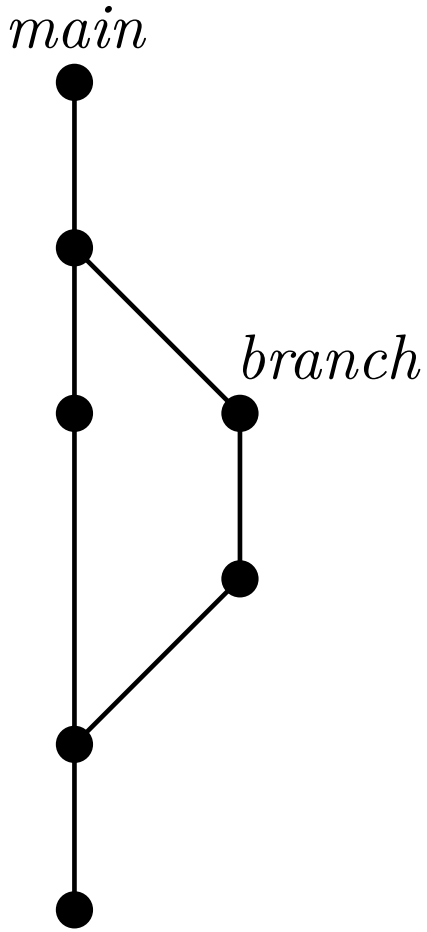
- En *commit* är en ”ögonblicksbild” (*snapshot*) av projektet vid ett specifikt tillfälle.
- Innehåller alla ändringar som gjorts sedan den senaste commiten.
- Varje commit refererar till sin(a) föregångare och skapar en historik.
- Bildar en graf av versioner som visar projektets utveckling över tid.



Grenar (*Branches*)

Man kan skapa parallella utvecklingsgrenar, som sedan kan slås samman (*merge*) med huvudgrenen igen.

- Varför vill man göra det?
- Vilka problem kan uppstå?



Varför grenar (*branches*)?

- Grenar = parallella vägar; sidospår utan att störa huvudvägen.
- Arbeta på olika delar av projektet samtidigt, utan att störa varandra.
- Testa nya funktioner, experimentera säkert.
- Skapa snabb fix för fel utan att stoppa ny utveckling.
- Branches i Git är lätta att skapa och använda!

Vilka problem kan uppstå?

- *Merge*: Git kombinerar ändringar från olika grenar automatiskt.
- *Merge-konflikt*: Uppstår när Git inte kan avgöra hur vissa ändringar ska kombineras.
- Konflikter sker ofta när samma del av en fil ändrats på flera grenar.
- Exempel: Flera redigerar samma rad, eller ändrar/byter namn på samma filer.
- Konflikter måste lösas manuellt för att slutföra en *merge*.

Exempel: Merge-konflikt i källkoden

Kod före merge

```
def greet(): String = {
    "Hello, world!"
}
```

Kod efter merge (med konflikt)

```
def greet(): String = {
<<<<<<< HEAD
    "Hello, world!"
=======
    "Hi, everyone!"
>>>>>>> branch
}
```

- Konflikten visas med `<<<<<<<`, `=======`, och `>>>>>>>`.
- Utvecklaren måste välja vilken version som ska behållas eller kombinera dem manuellt.

Ett större exempel

```
object Main {
    def main(args: Array[String]): Unit = {
        println("Starting calculation...")
        println("Result: " + calculateSum(5, 10))
    }

    def calculateSum(a: Int, b: Int): Int = {
<<<<<<< HEAD
        val sum = a + b
        println("Sum calculated: " + sum)
        sum
=======
        val sum = a + b
        println("Calculating sum for: " + a + " and " + b)
        println("Sum is: " + sum)
        sum * 2 // Doubles the result
>>>>>>> branch
    }
}
```

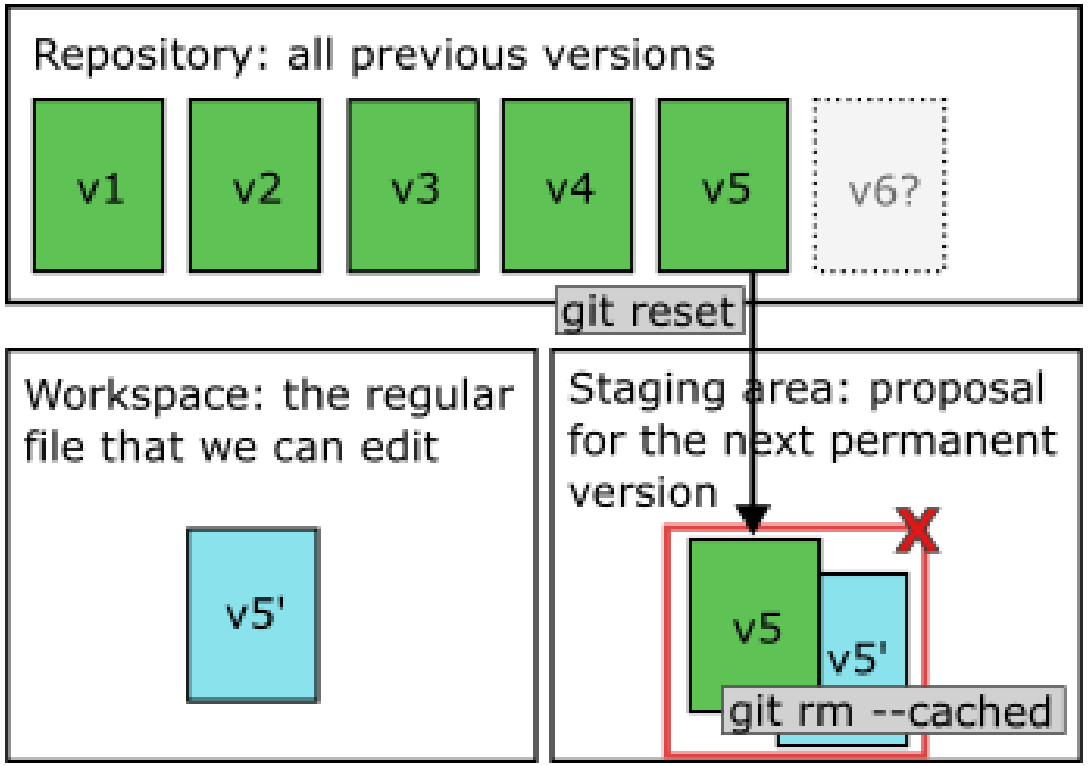
Arbeta distribuerat med Git

- *Klona*: Kopiera ett repository från internet till din dator.
- *Pull*: Hämta ändringar från ett fjärrrepository.
- *Push*: Skicka dina ändringar till ett fjärrrepository.
- *Pull request*: Föreslå ändringar till ett repository.
  - Mycket vanligt i *open source*-projekt.
  - Kallas ibland *merge request*.

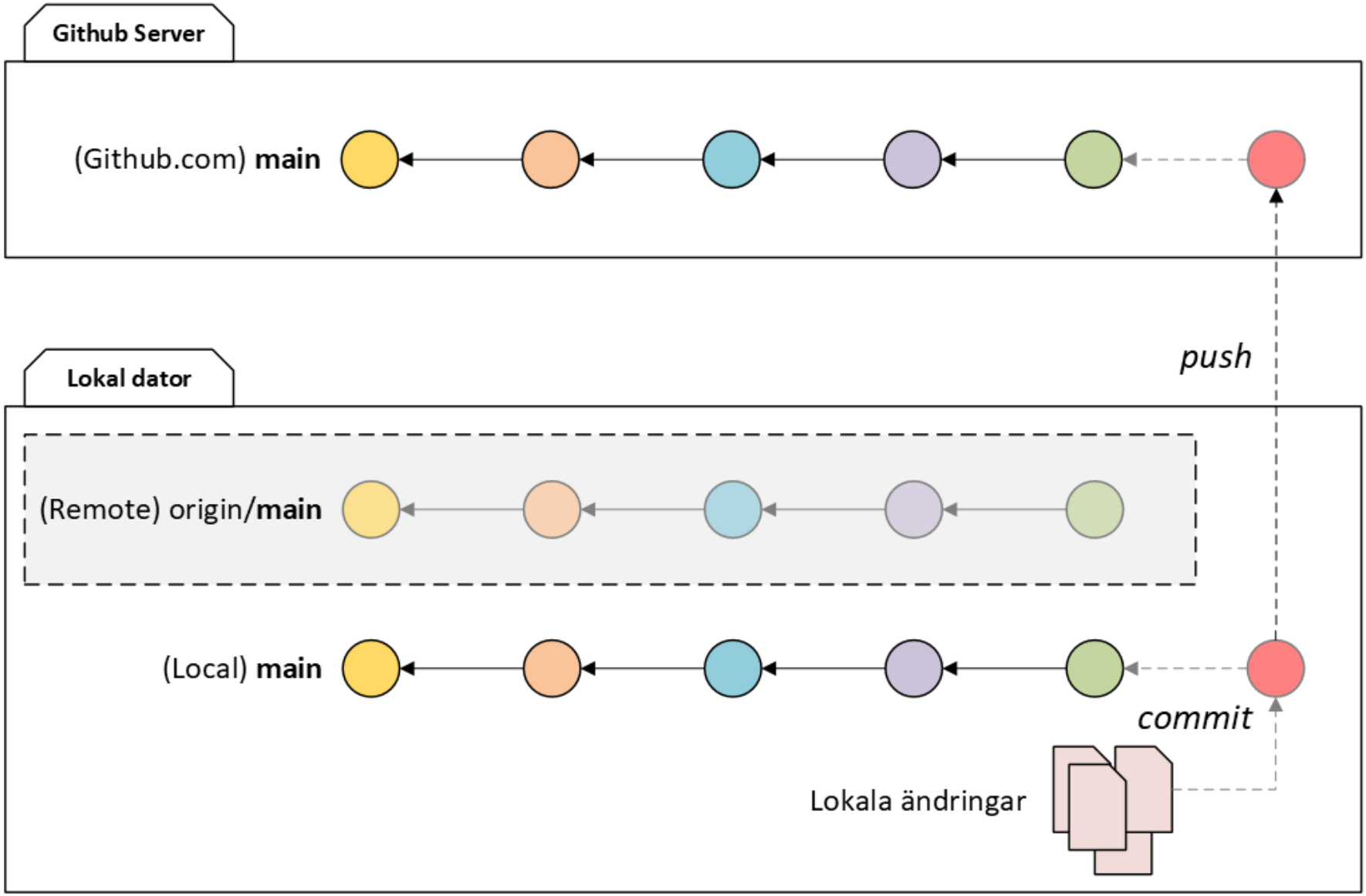
Remote

- Git använder termen *remote*. Lokal "spegling" av fjärrrepository.
- Standardnamn för remote är `origin`.
- Möjligt att ha flera remotes för olika kopior av projektet.
- Kolla med `git remote [-a, -v, -vv]`

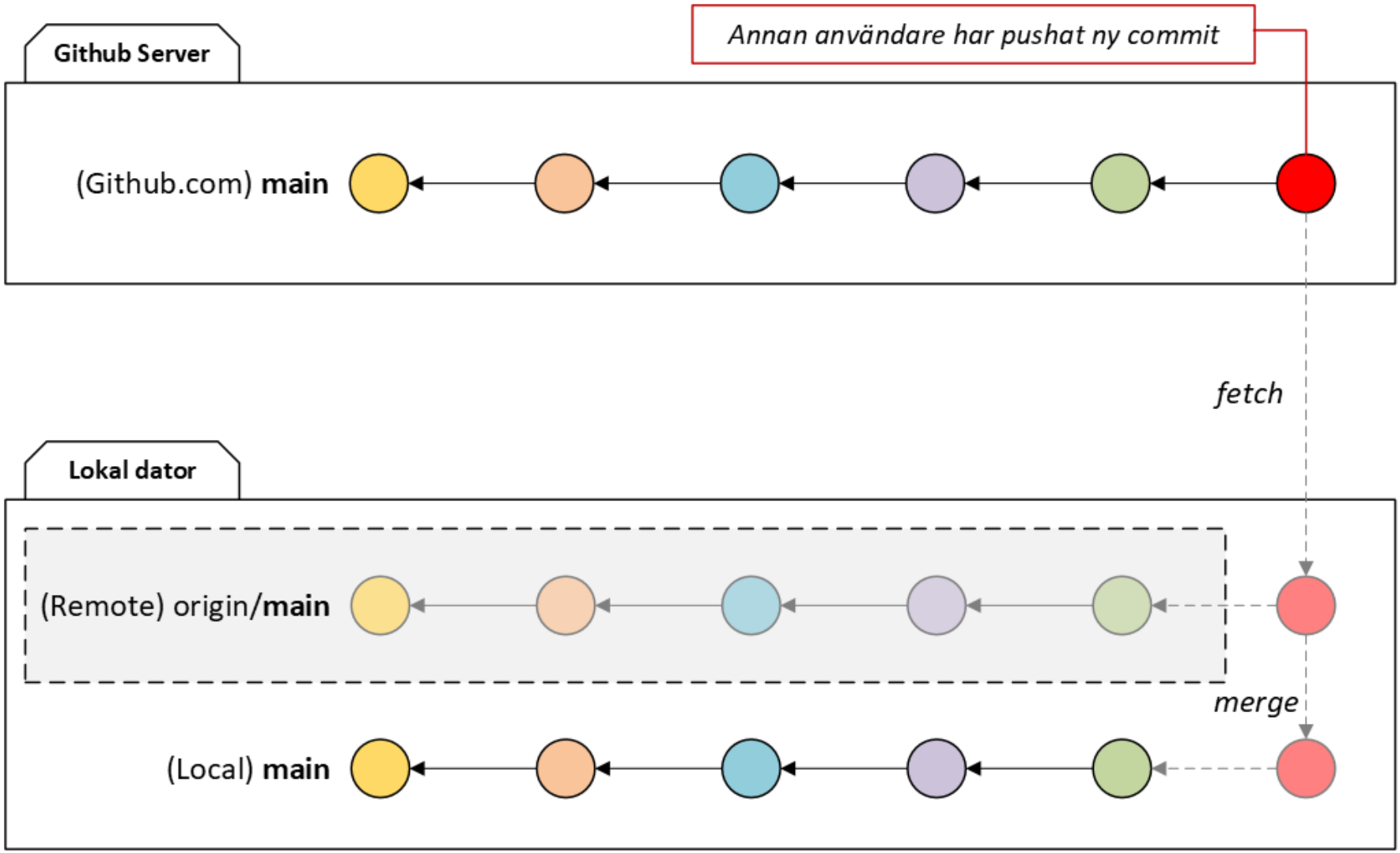
Påminnelse - Context



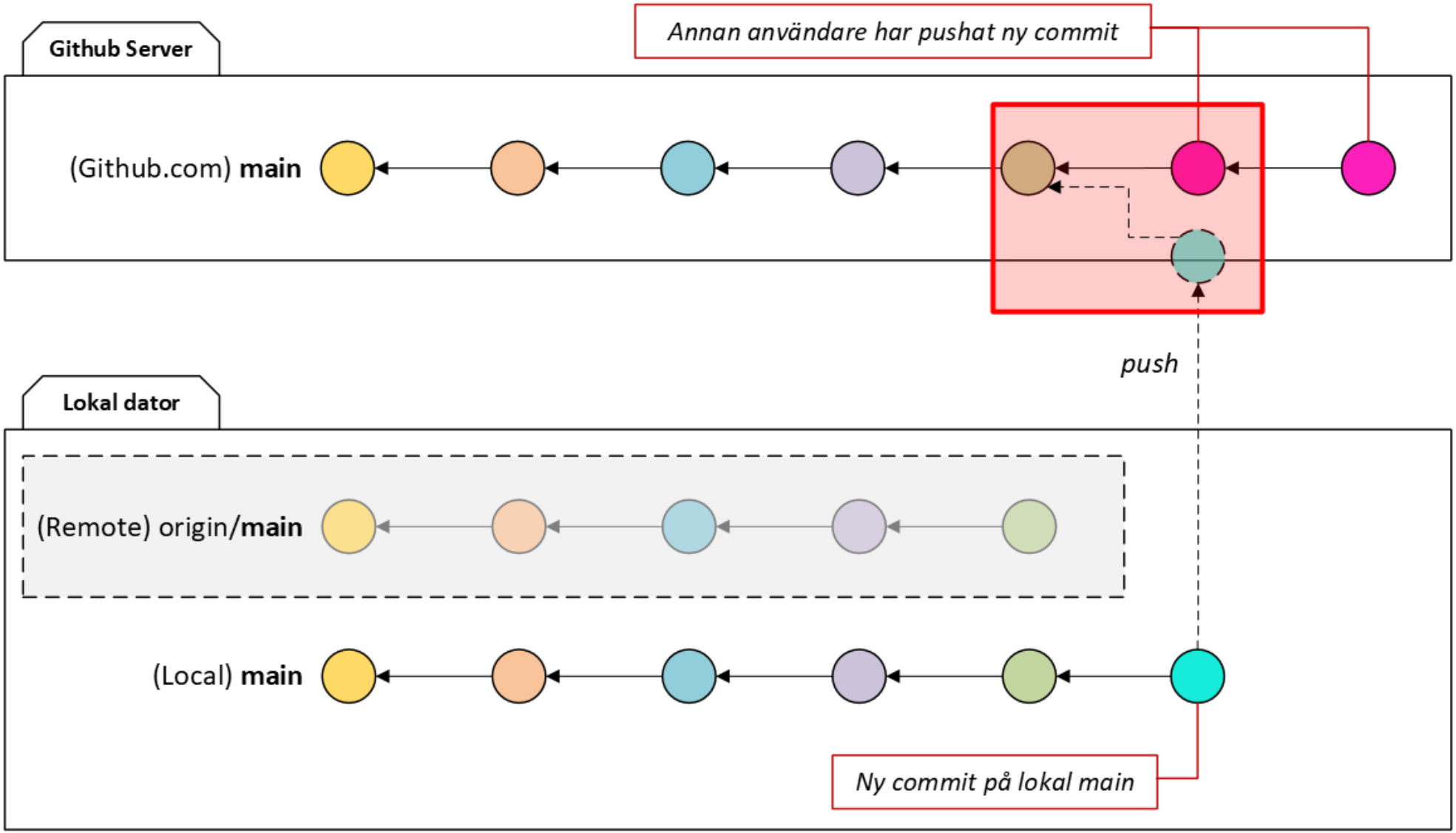
Local vs remote: commit och push



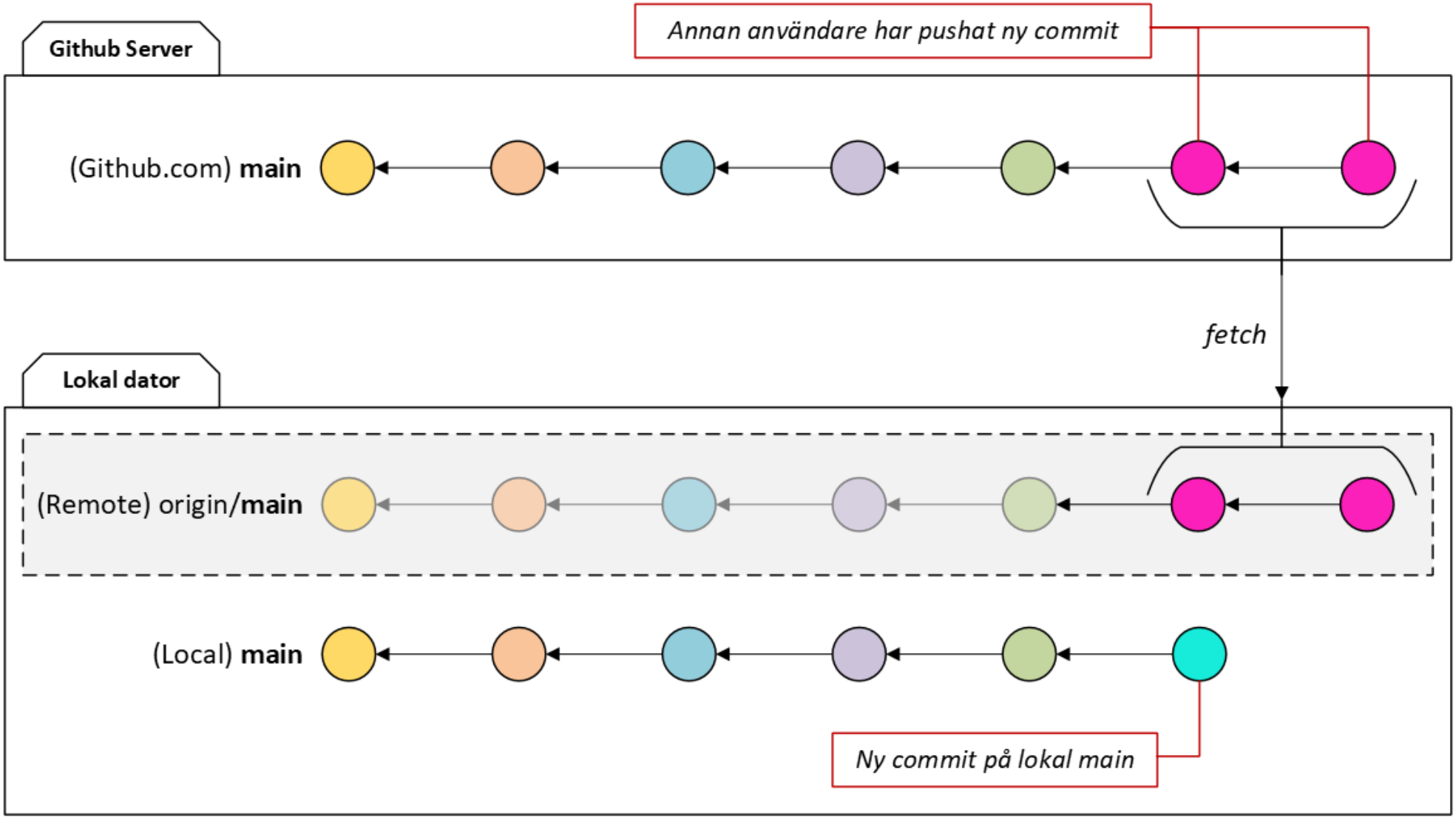
Local vs remote: fetch och merge



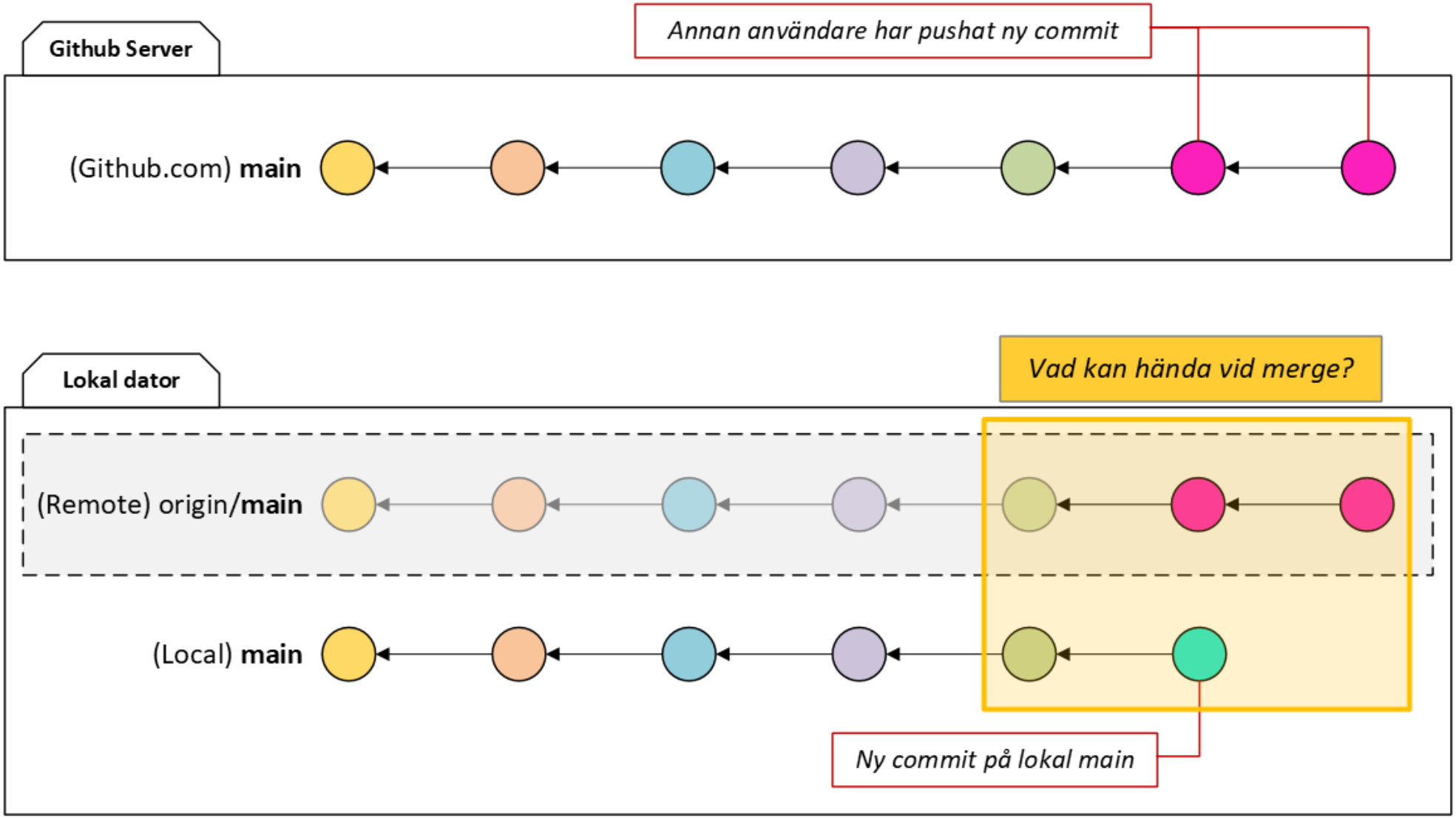
Båda samtidigt!? – push



Båda samtidigt!? – fetch



Båda samtidigt!? – merge efter fetch

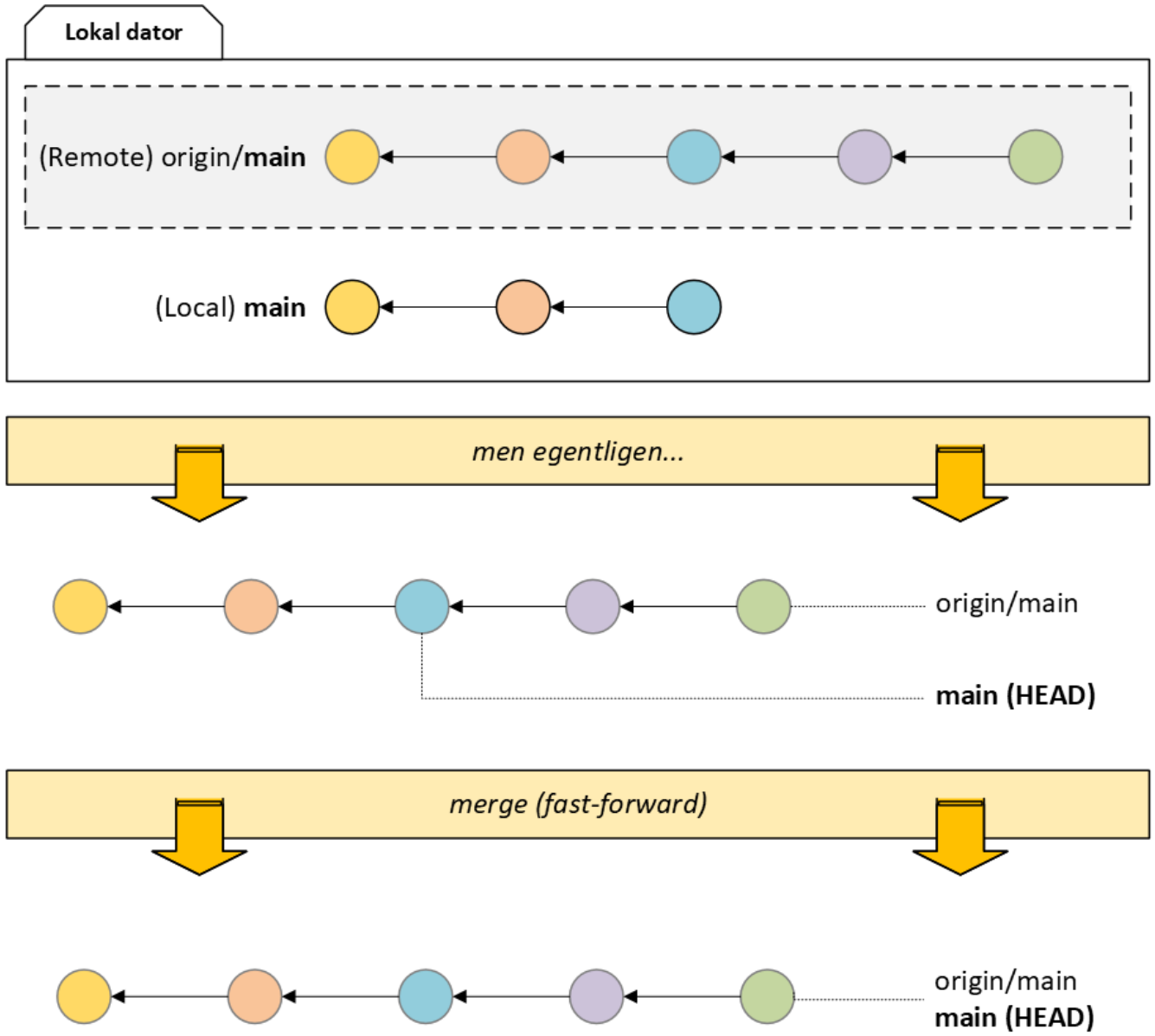


Merge-situationer

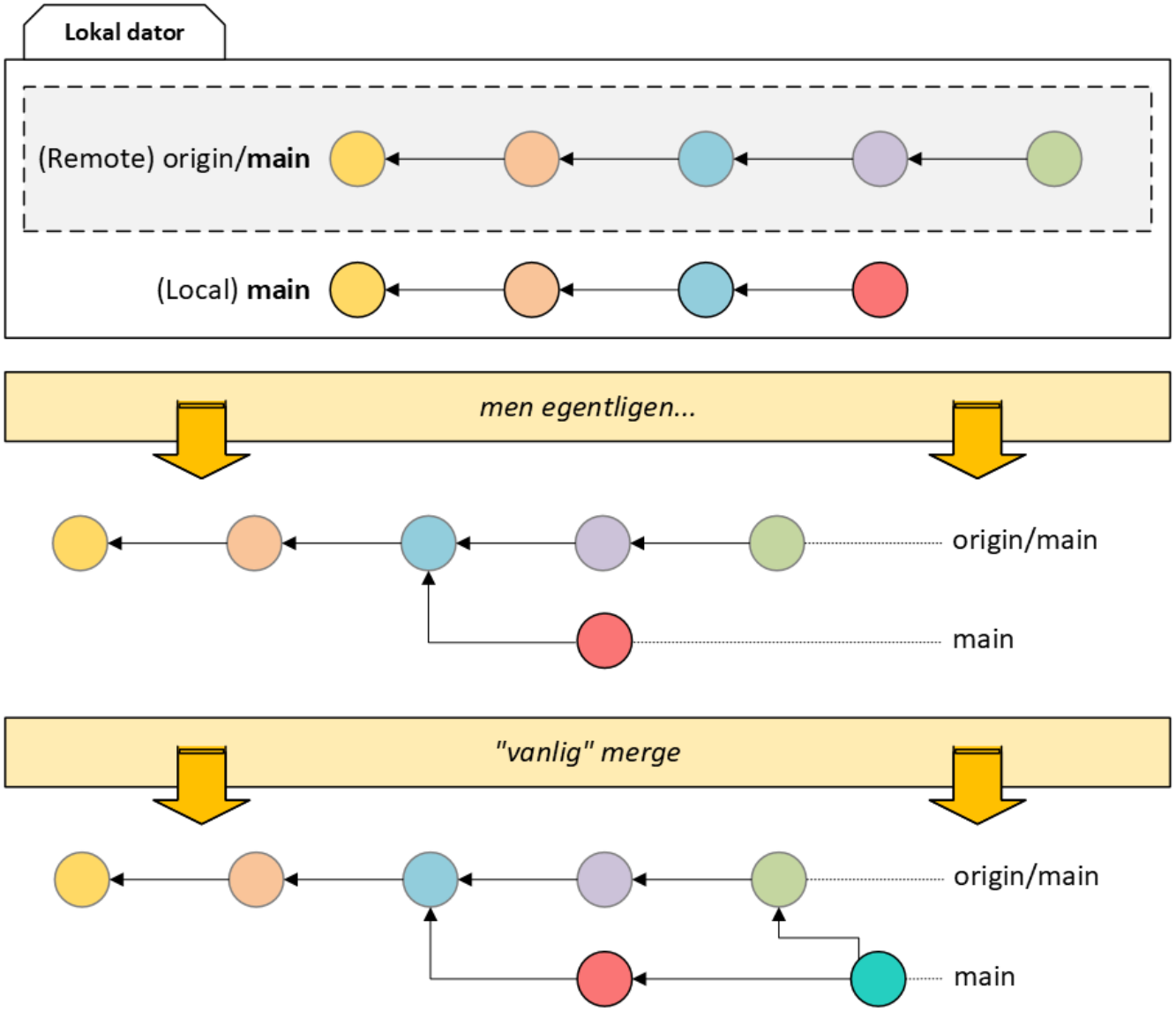
Vad kan vid merge?

- fast-forward merge
- merge commit
- konflikt
- rebase

Fast-forward merge



Vanlig merge – skapar en “merge commit”

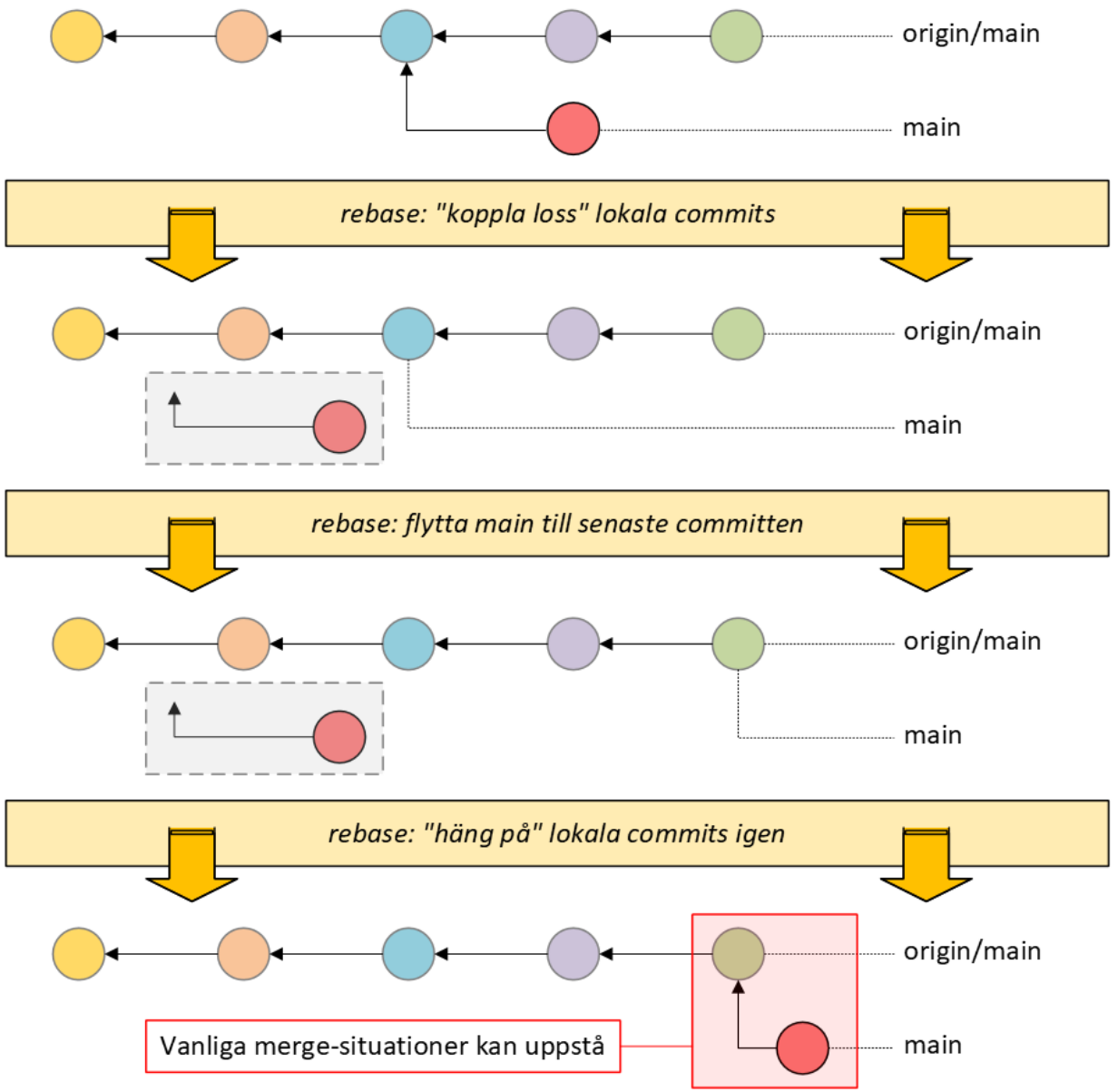


Mergekonflikt

Vid konflikt, se föregående figur.

- Om ändringarna i de två grenarna inte kan kombineras automatiskt, uppstår en konflikt.
- Git markerar konflikter i filerna och pausar merge-processen.
- Efter konflikten lösts, finns alltså ny ändringar i worspace...
- ...och du måste göra en ny commit för att slutföra merge.

Rebase


origin/main
main
rebase: "koppla loss" lokala commits
origin/main
main
rebase: flytta main till senaste committen
origin/main
main
rebase: "häng på" lokala commits igen
origin/main
main
Vanliga merge-situationer kan uppstå

Bra att veta

- `HEAD` är en *symbolisk referens* till den branch som är utcheckad i ditt workspace.
- "Detatched HEAD" – om du checkar ut en specifik commit, snarare än en branch.

Veckans laboration

På veckans laboration kommer ni:

- Arbeta er igenom ett enkelt scenario som visar hur ni kan använda Git och Github för ett litet projekt.
- Tänkt att kunna genomföras individuellt, men OK och uppmuntras till att samarbeta, speciellt i den sista delen av laborationen.

Glöm inte att:

- Installera Git på din egen dator.

Vad händer nu?

- Läs avsnittet om Git i Scalakompendiet (Björn Regnell).
- Se avsnitt 1.1, 1.2, 1.3, 1.6 samt 1.7 av youtube-serien “Git and GitHub for Poets”.
- Läs kapitel 1, 2 och 3.1–3.2 i Pro Git-boken (Chacon & Straub).
- Länkar till materialet finns på kurshemsidan:

  `https://cs.lth.se/dod/datorlaborationer/`