TDP005 - Projekt: Objektorienterat system

Kursupplägg, kravspecifikation och utvecklingsmetoder

Pia Løtvedt & János Dani

Institutionen för datavetenskap

LINKÖPING UNIVERSITY

1 Make och CMake

2 Versionshantering

LINKÖPING UNIVERSITY

Make

- Problem: kompilera många filer i ett stort projekt tar tid
- Bättre om vi kompilerar om så få filer som möjligt!
- Make kan hjälpa oss!

Make

foo.h

```
#pragma once

int foo(int val);
```

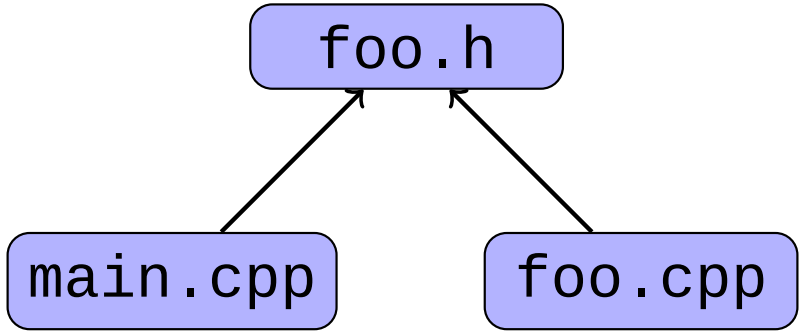
main.cpp

```
#include "foo.h"
#include <iostream>
//...
int main() {
  cout << foo(10) << endl;
  return 0;
}
```

foo.cpp

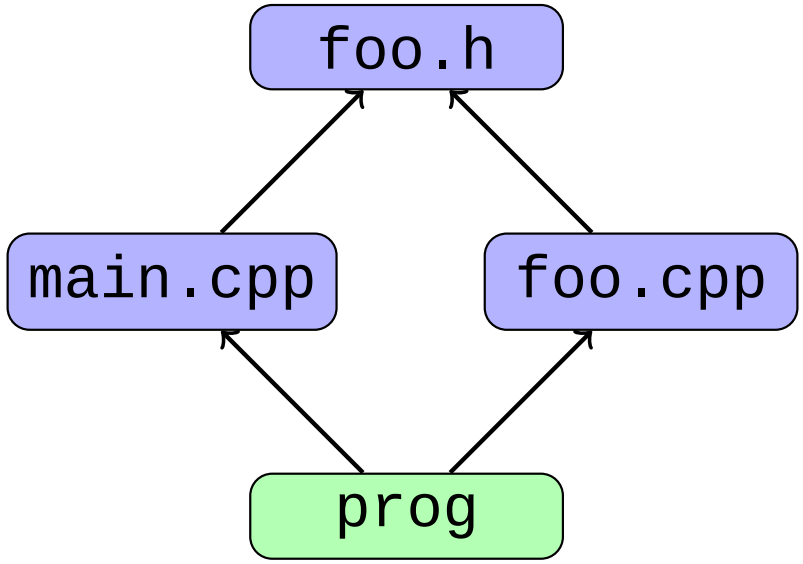
```
#include "foo.h"

int foo(int val) {
  return val + 1;
}
```

Make – Beroenden



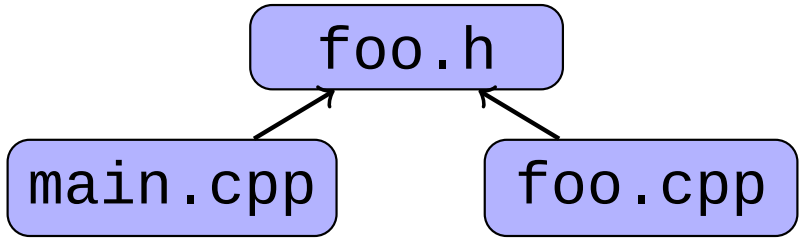

```
g++ -std=c++17 -Wall *.cpp -o prog
```

Make – Beroenden



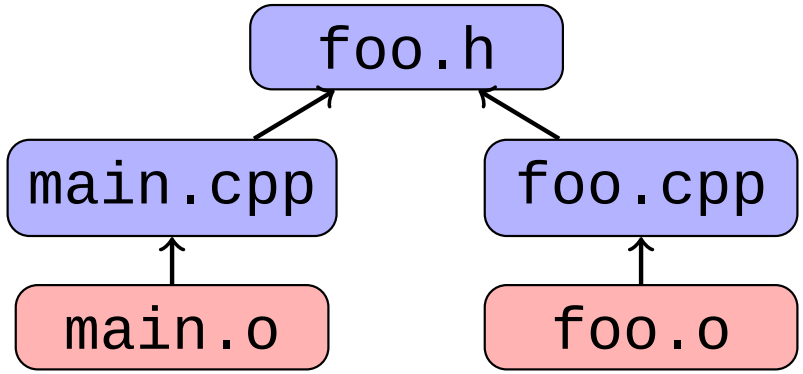

```
g++ -std=c++17 -Wall *.cpp -o prog
```

Make – Beroenden



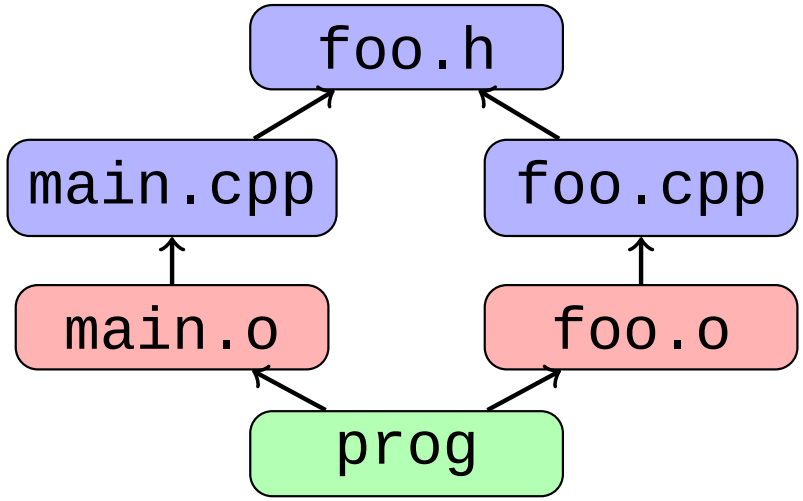

```
g++ -c -std=c++17 -Wall main.cpp -o
main.o
g++ -c -std=c++17 -Wall foo.cpp -o foo.o
g++ -std=c++17 main.o foo.o -o prog
```

Make – Beroenden




```
g++ -c -std=c++17 -Wall main.cpp -o
main.o
g++ -c -std=c++17 -Wall foo.cpp -o foo.o
g++ -std=c++17 main.o foo.o -o prog
```

Make – Beroenden




```
g++ -c -std=c++17 -Wall main.cpp -o
main.o
g++ -c -std=c++17 -Wall foo.cpp -o foo.o
g++ -std=c++17 main.o foo.o -o prog
```

Make – makefil

- Första försök:

```
prog: foo.o main.o
  g++ -std=c++17 -g -Wall foo.o main.o -o prog

foo.o: foo.cpp foo.h
  g++ -c -std=c++17 -g -Wall foo.cpp -o foo.o

main.o: main.cpp foo.h
  g++ -c -std=c++17 -g -Wall main.cpp -o main.o
```

Make – makefil

- Förbättring:

```
CXXFLAGS = -std=c++17 -g -Wall

prog: foo.o main.o
  g++ $(CXXFLAGS) foo.o main.o -o prog

foo.o: foo.cpp foo.h
  g++ -c $(CXXFLAGS) foo.cpp -o foo.o

main.o: main.cpp foo.h
  g++ -c $(CXXFLAGS) main.cpp -o main.o
```

Make – makefil med `.PHONY` och `clean`

```
CXXFLAGS = -std=c++17 -g -Wall

prog: foo.o main.o
  g++ $(CXXFLAGS) foo.o main.o -o prog

foo.o: foo.cpp foo.h
  g++ -c $(CXXFLAGS) foo.cpp -o foo.o

main.o: main.cpp foo.h
  g++ -c $(CXXFLAGS) main.cpp -o main.o

.PHONY: clean
clean:
  rm *.o prog
```

Make

- Finns mycket mer: makron, variabler, etc.
- Se make-laborationen för mer info.

CMake

CMakeLists.txt:

```
project(prog)

set(CMAKE_CXX_STANDARD 17)
#set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -std=c++17")

set(SOURCE_FILES main.cpp foo.cpp foo.h)

add_executable(prog ${SOURCE_FILES})
```

Kompilera:

```
$ cmake .
$ make
```

1 Make och CMake

2 Versionshantering

LINKÖPING UNIVERSITY

Git

- Git ska användas i kursen
- Interaktiv genomgång:
  `http://learngitbranching.js.org/`
- Hur används Git i större projekt?

GitLab – Issues

GitLab har inbyggt stöd för issues. Kan användas för att hålla koll på vad som ska göras härnäst, buggar, etc.

I ett commit-meddelande kan man skriva:

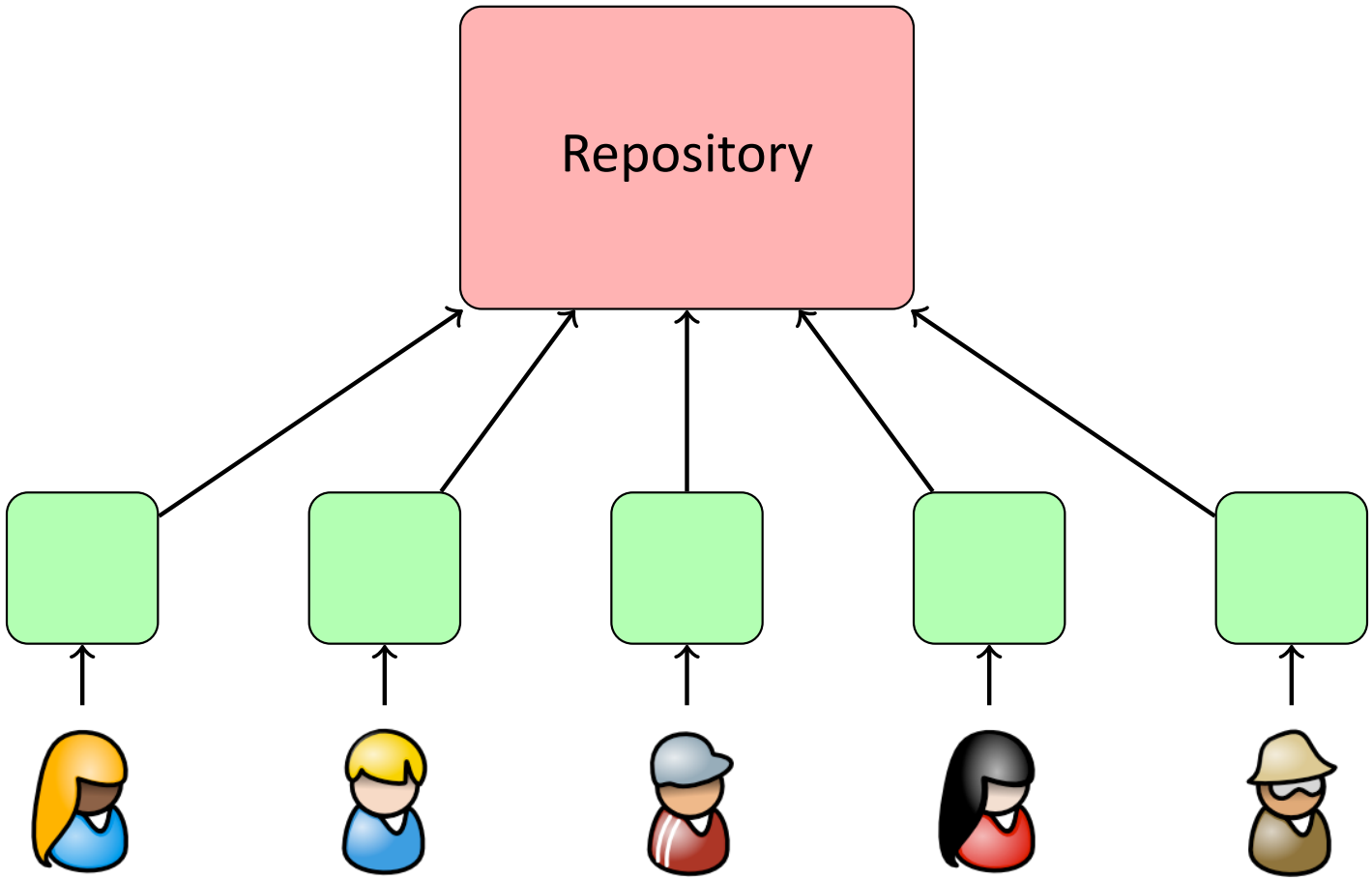
fixes #13

Då stängs automatiskt issue #13, och en referens till den commit som stängde den läggs till.

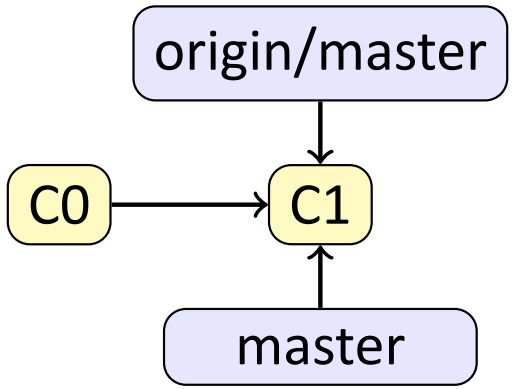
Git – Hur organiseras koden i större projekt?

- Centralt repository
  - Merge
  - Rebase
  - Feature Branch
  - Gitflow
- Flera repositories
  - Integration-Manager Workflow (Forking Workflow)
  - Dictator and Lieutenants Workflow

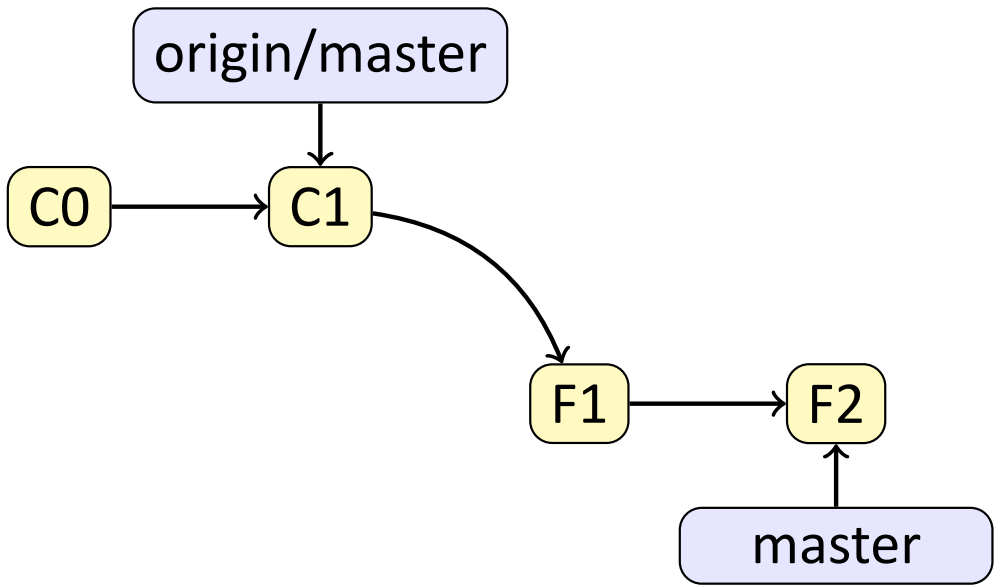
Git – Centralt repository



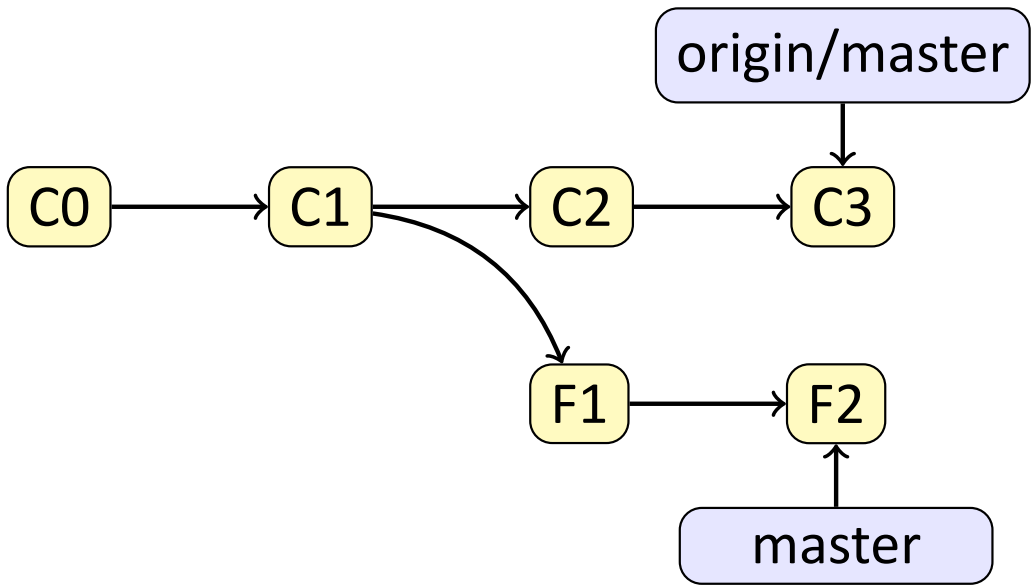
Git – Merge



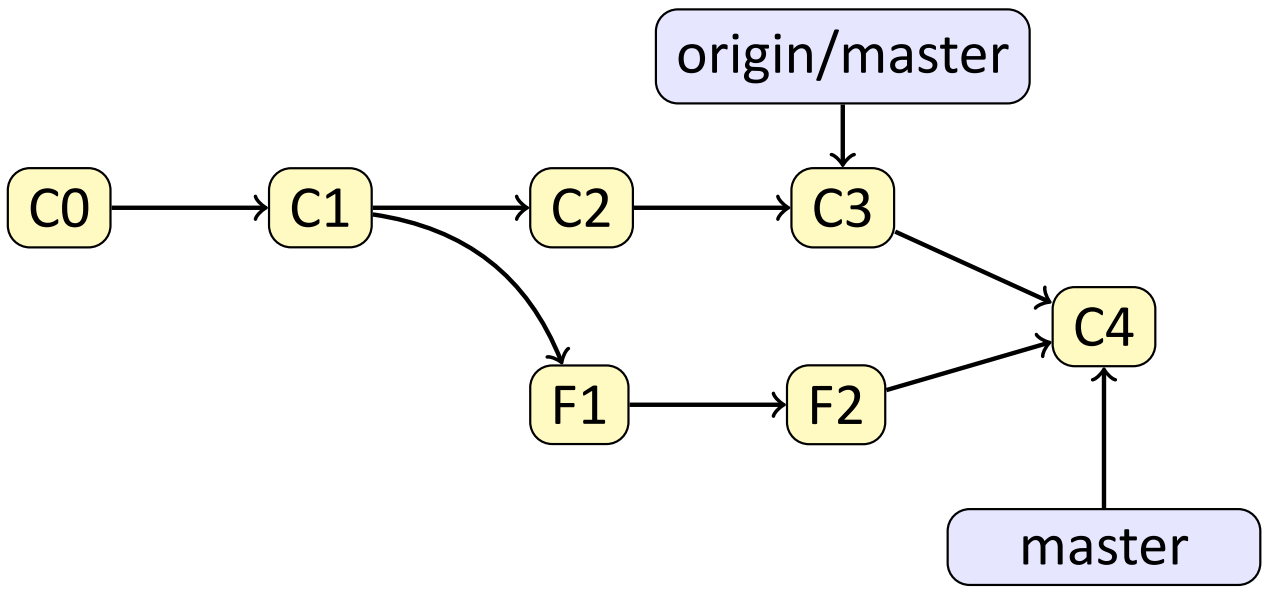
Git – Merge



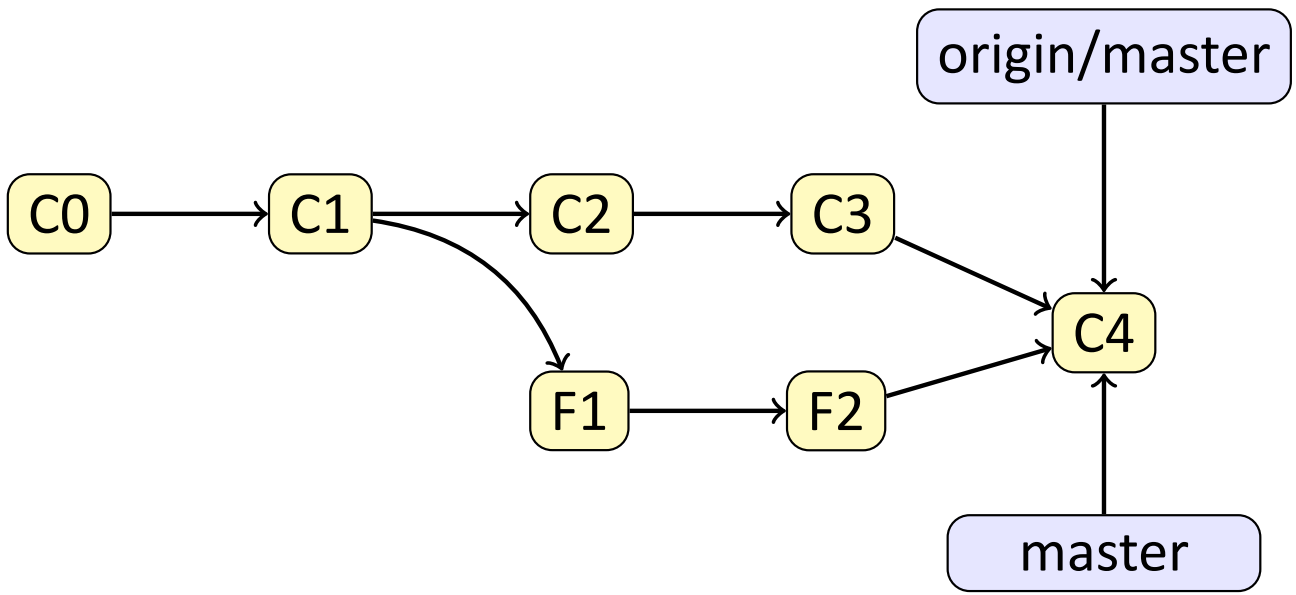
Git – Merge



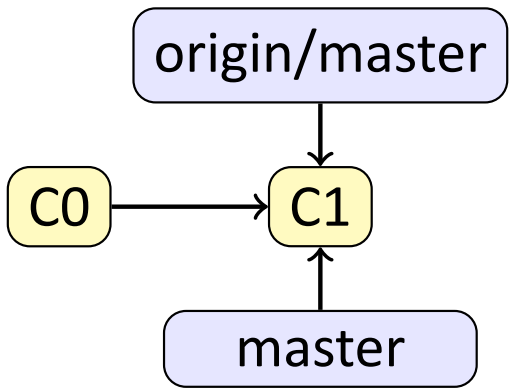
Git – Merge



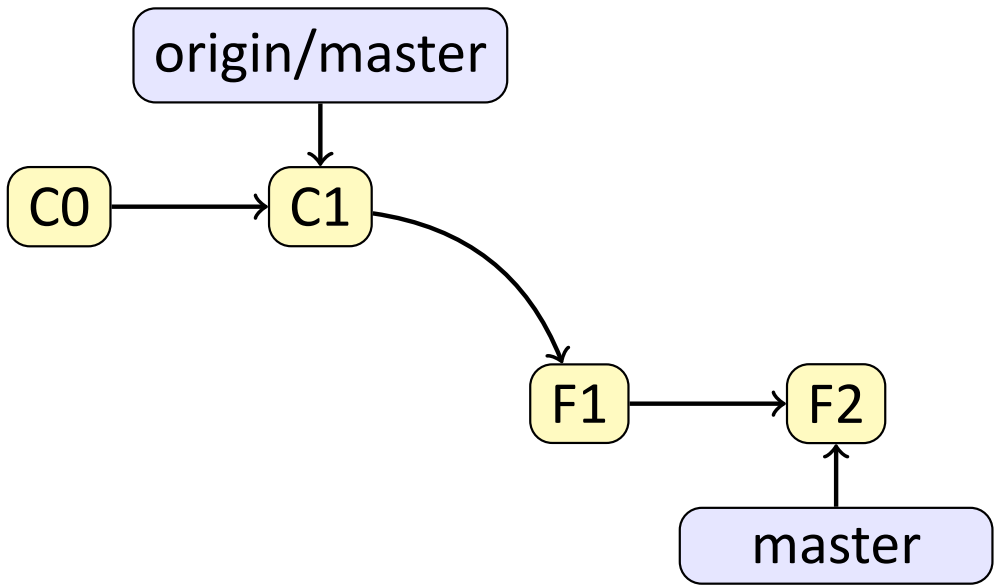
Git – Merge



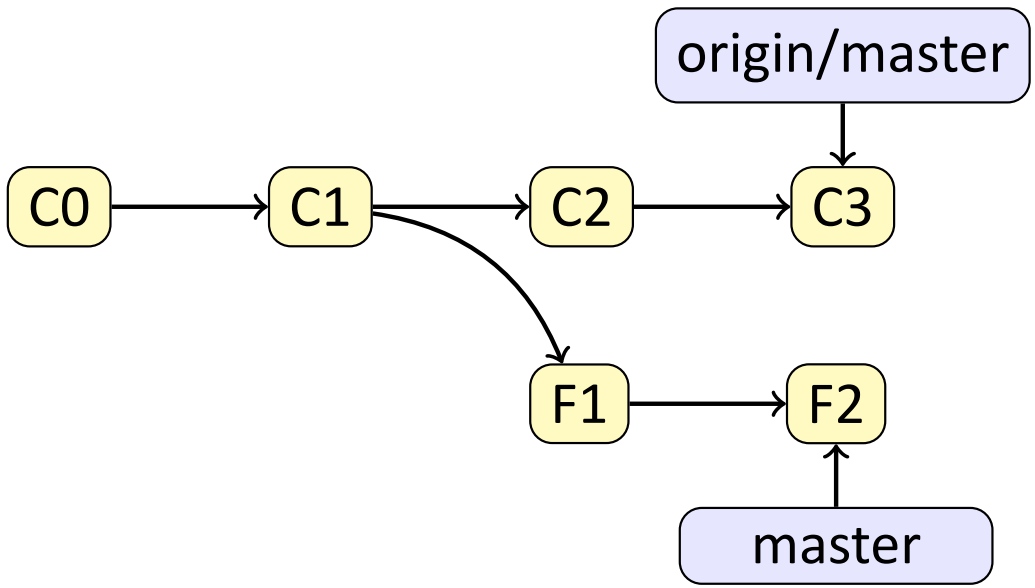
Git – Rebase



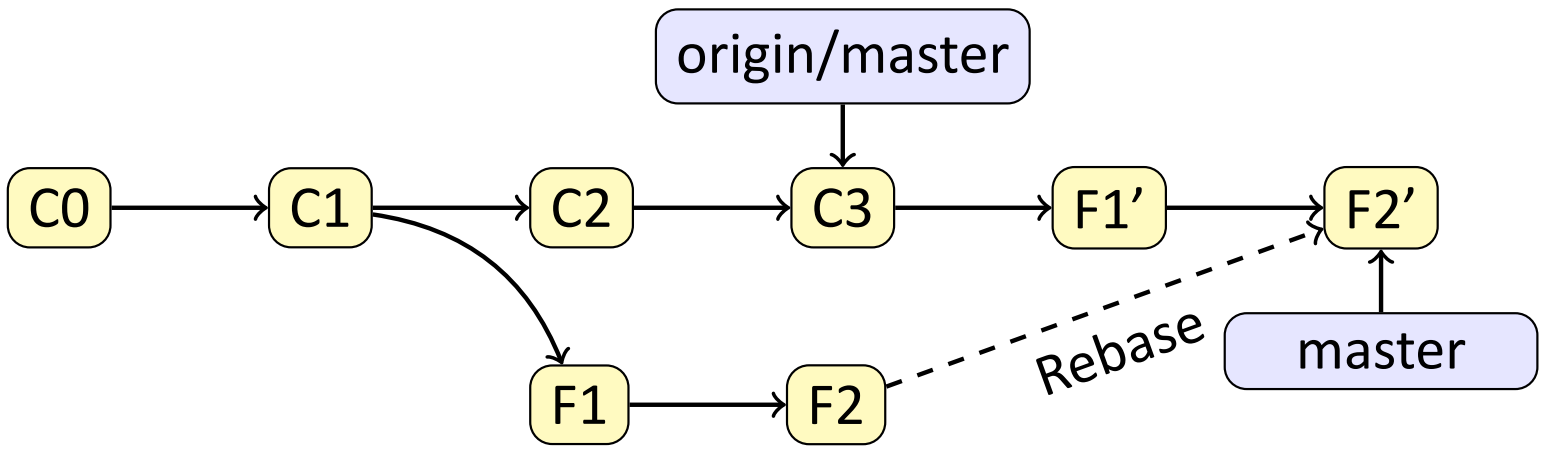
Git – Rebase



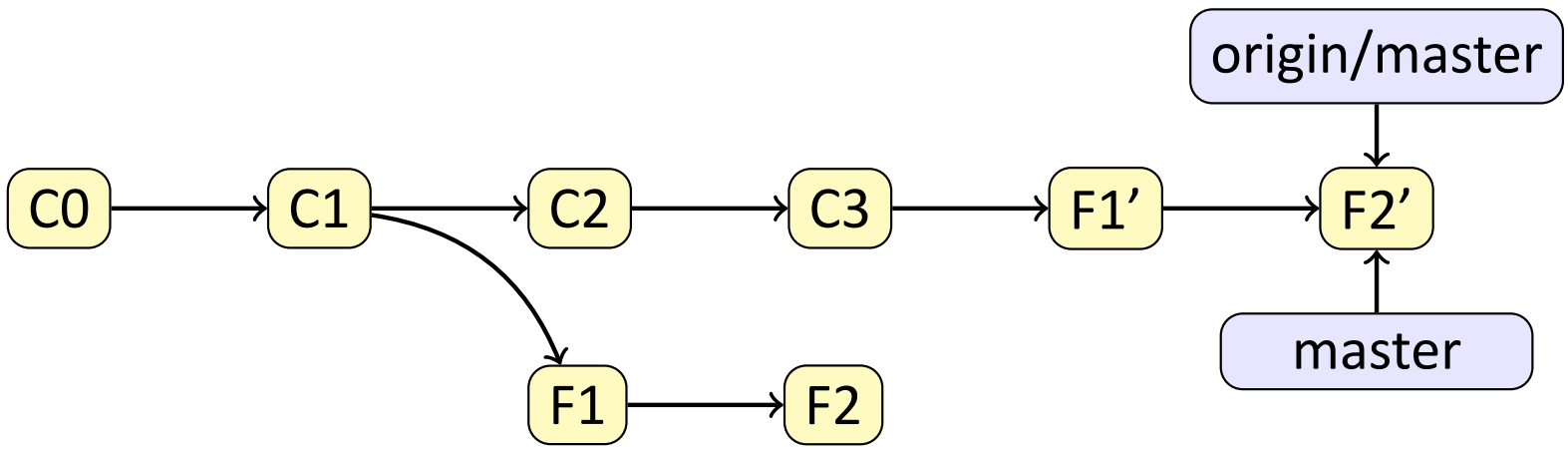
Git – Rebase



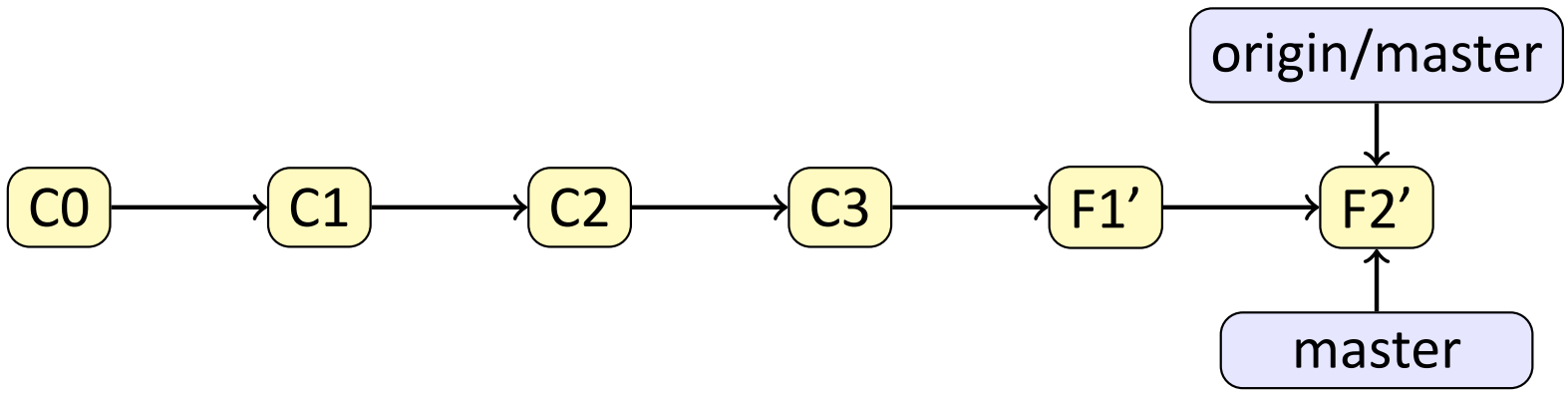
Git – Rebase



Git – Rebase



Git – Rebase

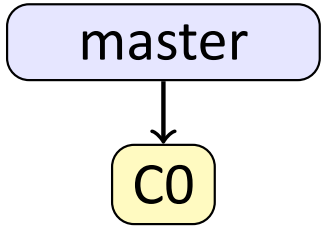


Git – Merge och Rebase

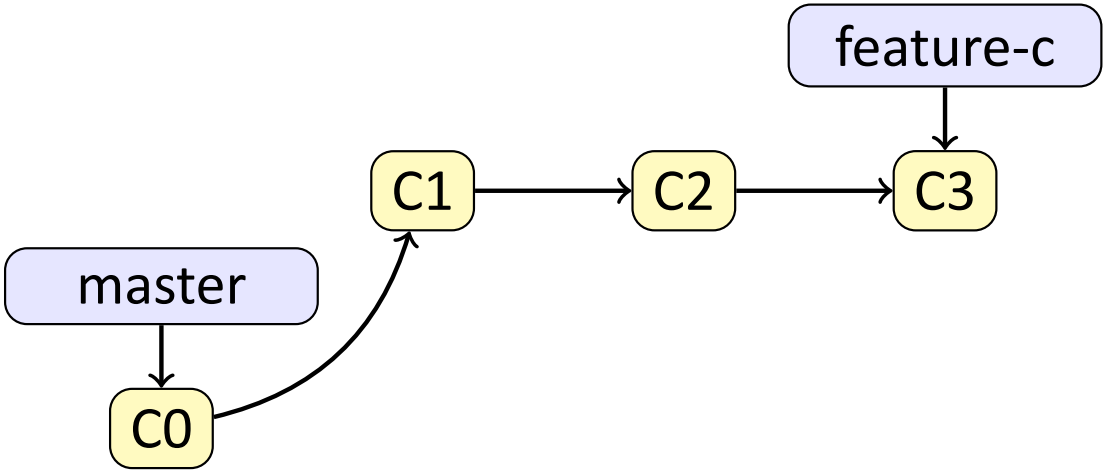
- Enklaste sättet att samarbeta
- Fungerar bra i mindre projekt

- Kan inte samarbeta på funktionalitet som ej är klar
- Risk att `master` inte fungerar, problem för andra
- Stor feature, stora merge-konflikter

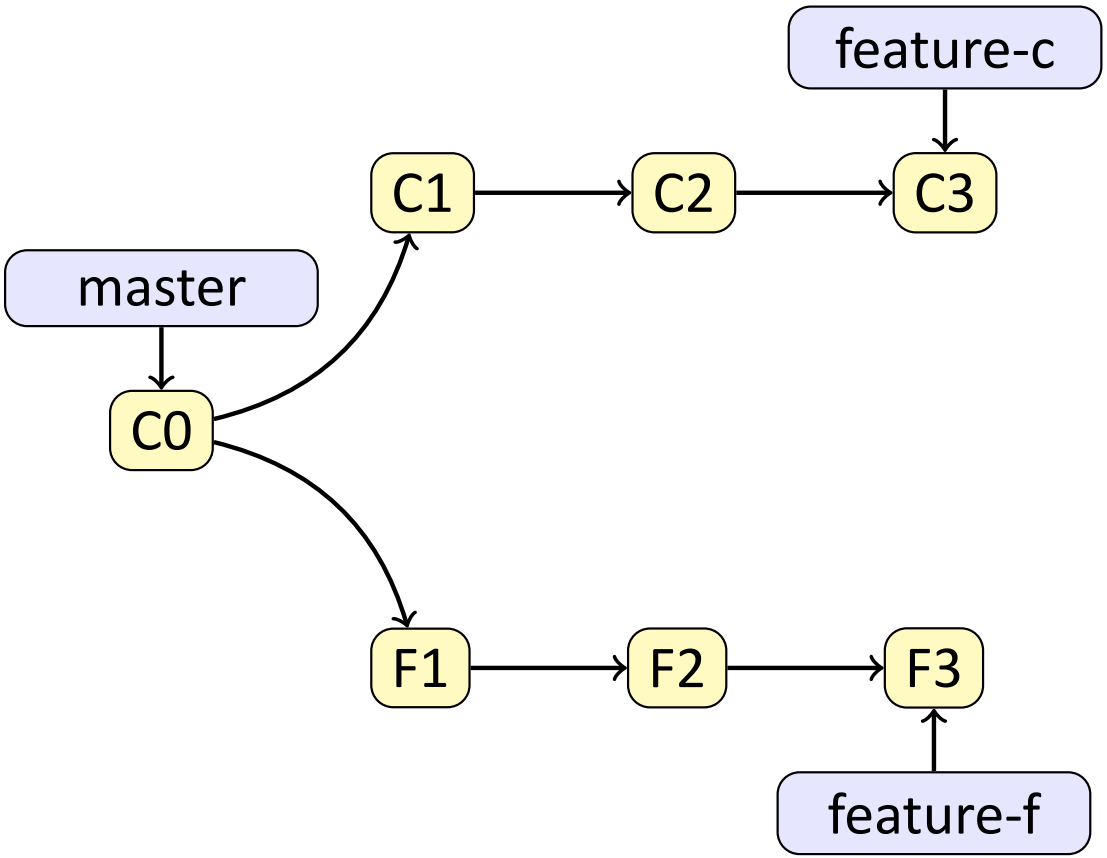
Git – Feature Branch



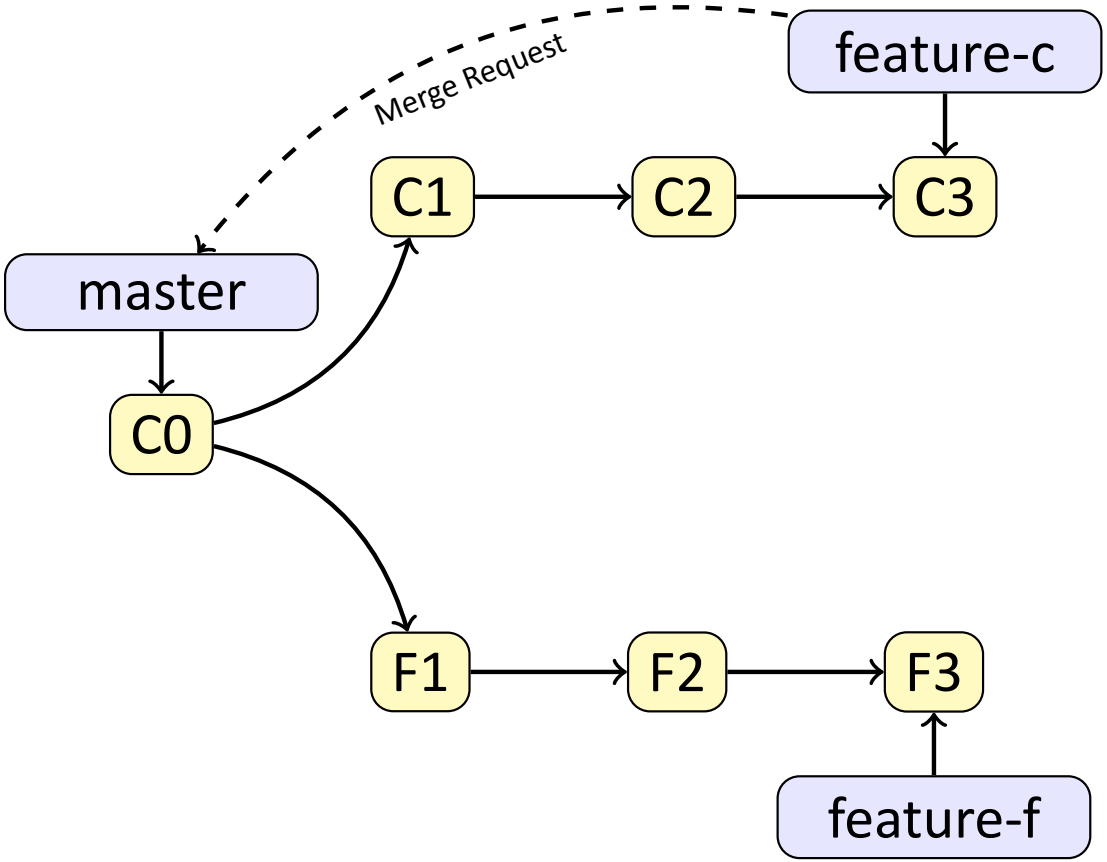
Git – Feature Branch



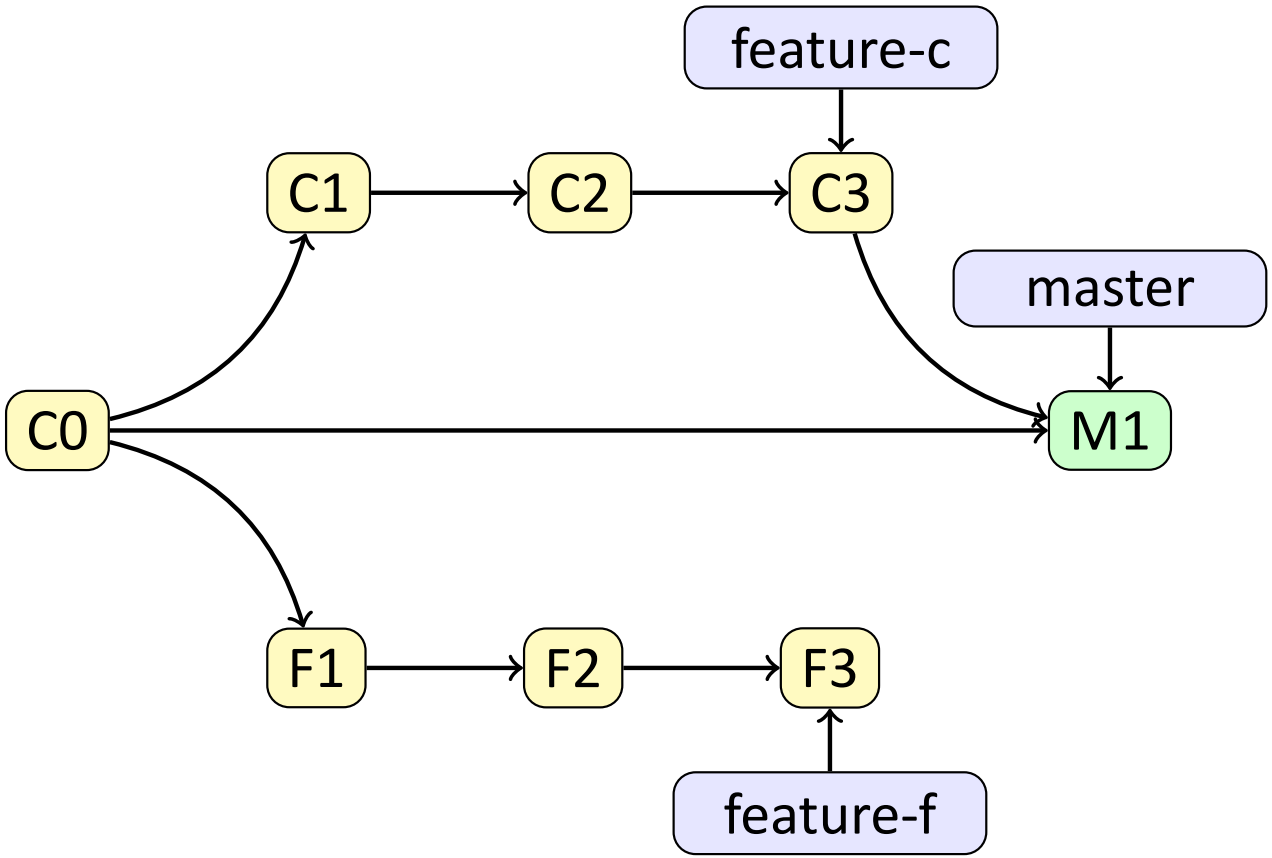
Git – Feature Branch



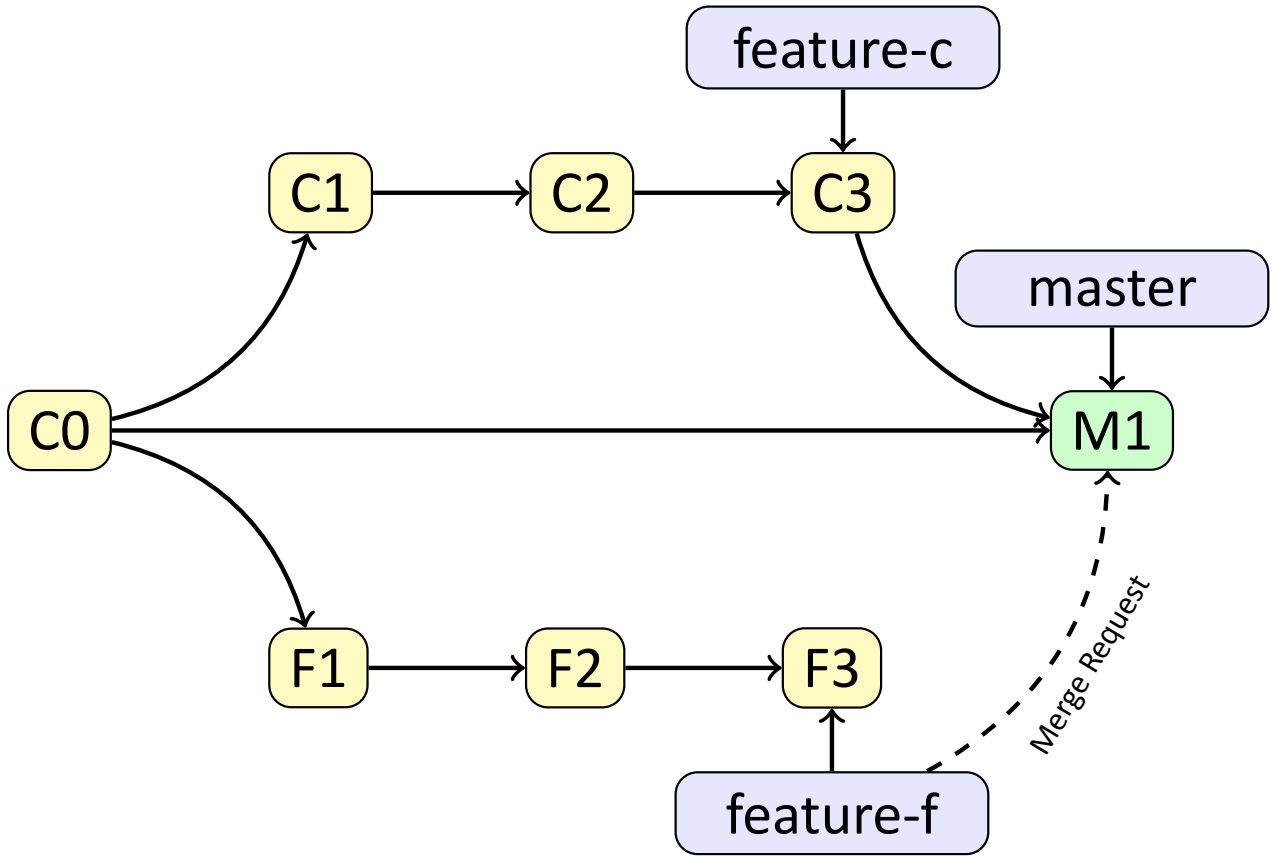
Git – Feature Branch



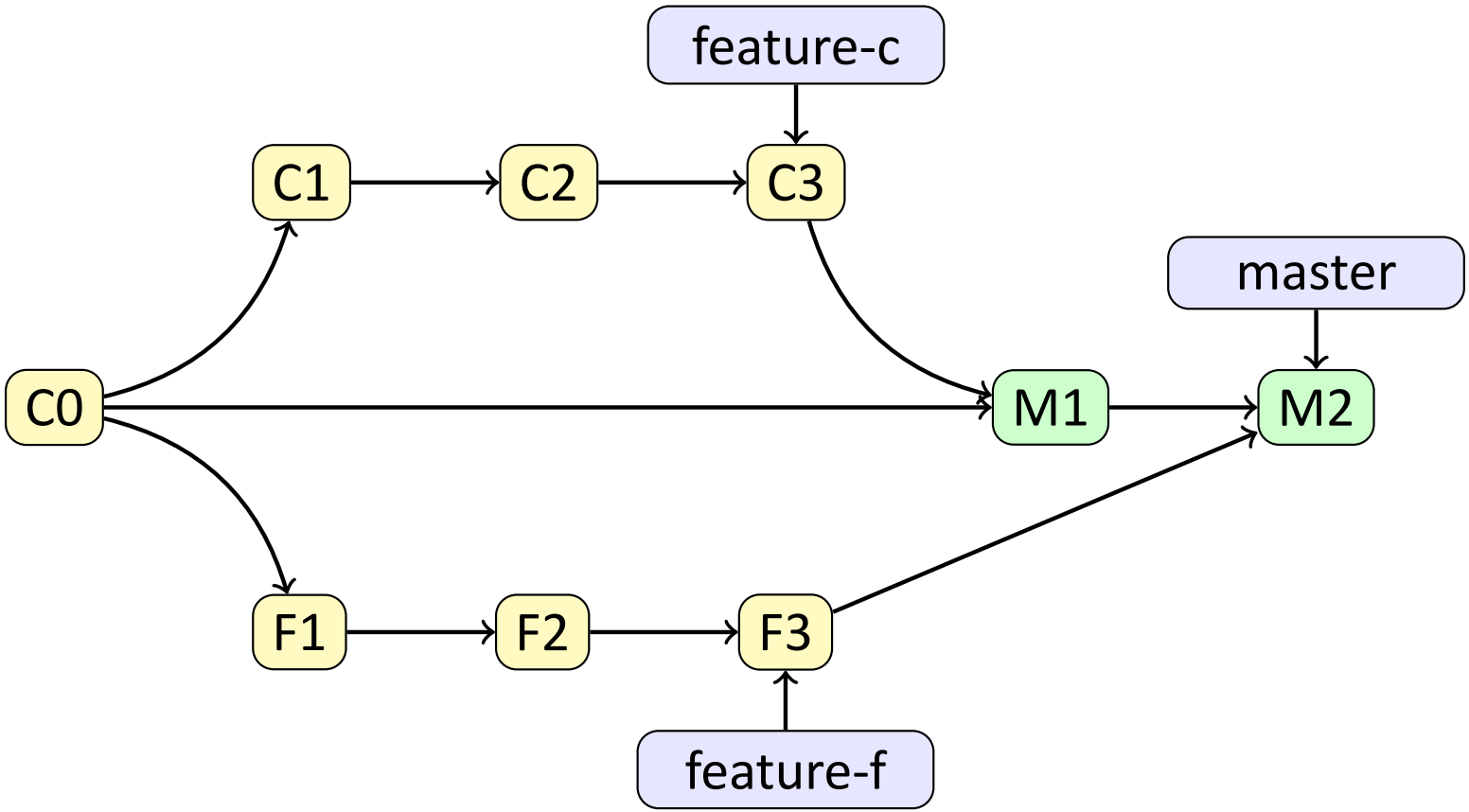
Git – Feature Branch



Git – Feature Branch



Git – Feature Branch

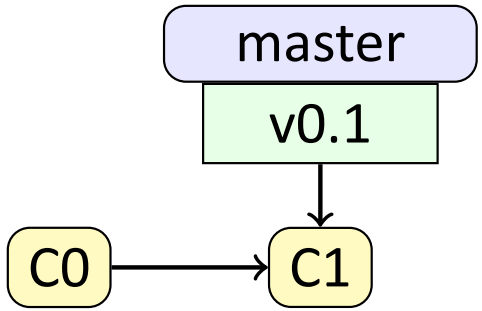


Git – Feature Branch

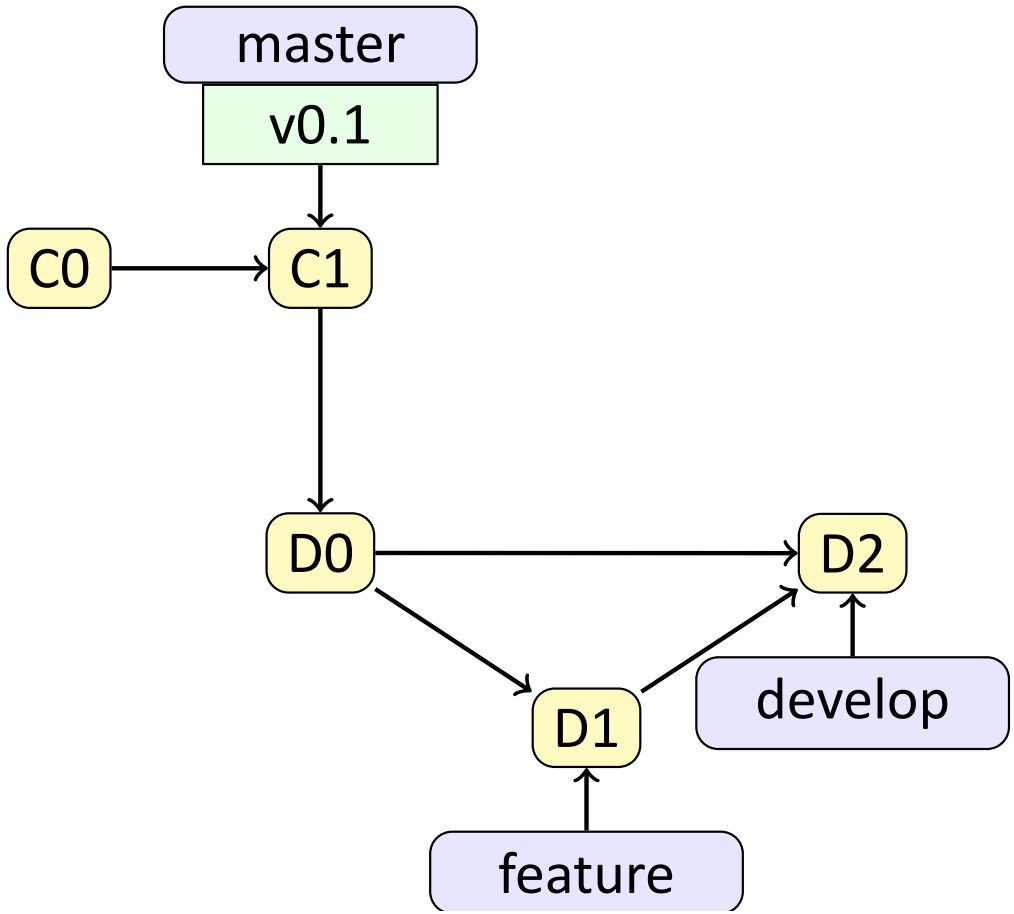
- Samarbete/backup på ej klara funktionalitet
- Tydligt var kodgranskning görs
- `master` fungerar alltid ⇒ Continuous Integration

- Lite mer arbete än tidigare
- Risk för att funktionalitet ”tappas bort”
- Risk för stora merges ⇒ lite funktionalitet i taget

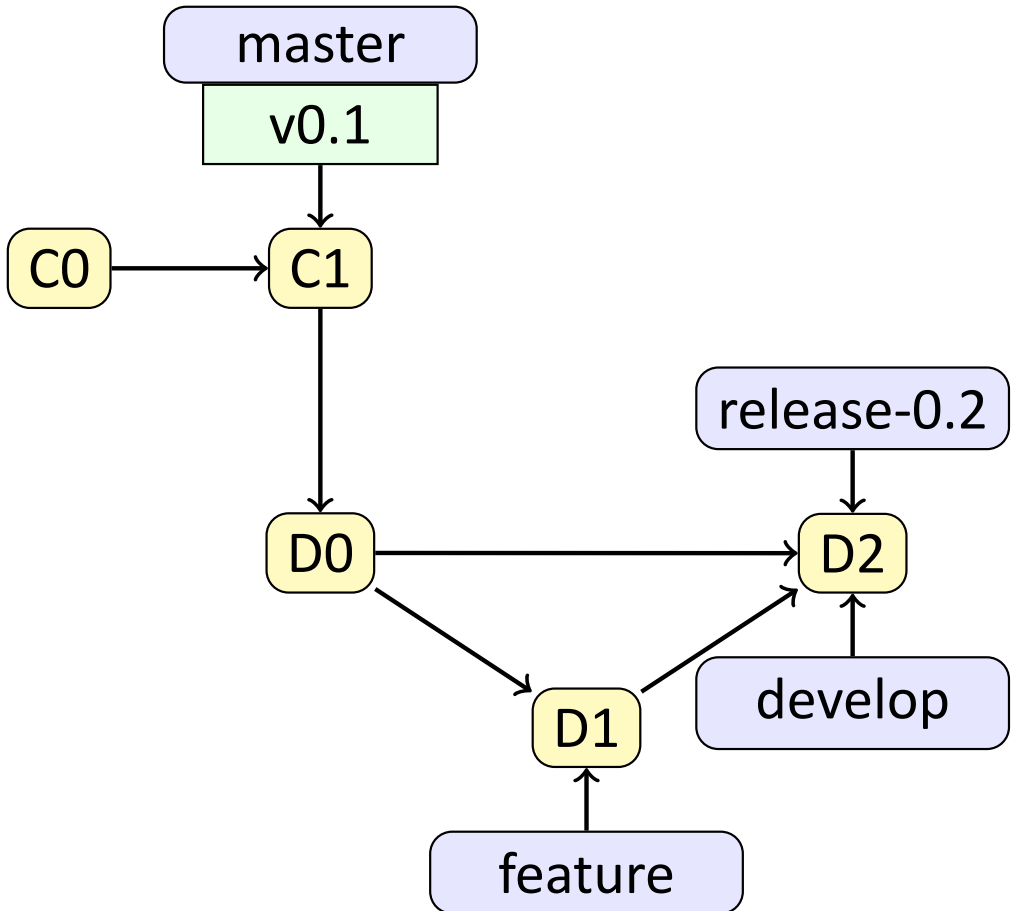
Git – Gitflow



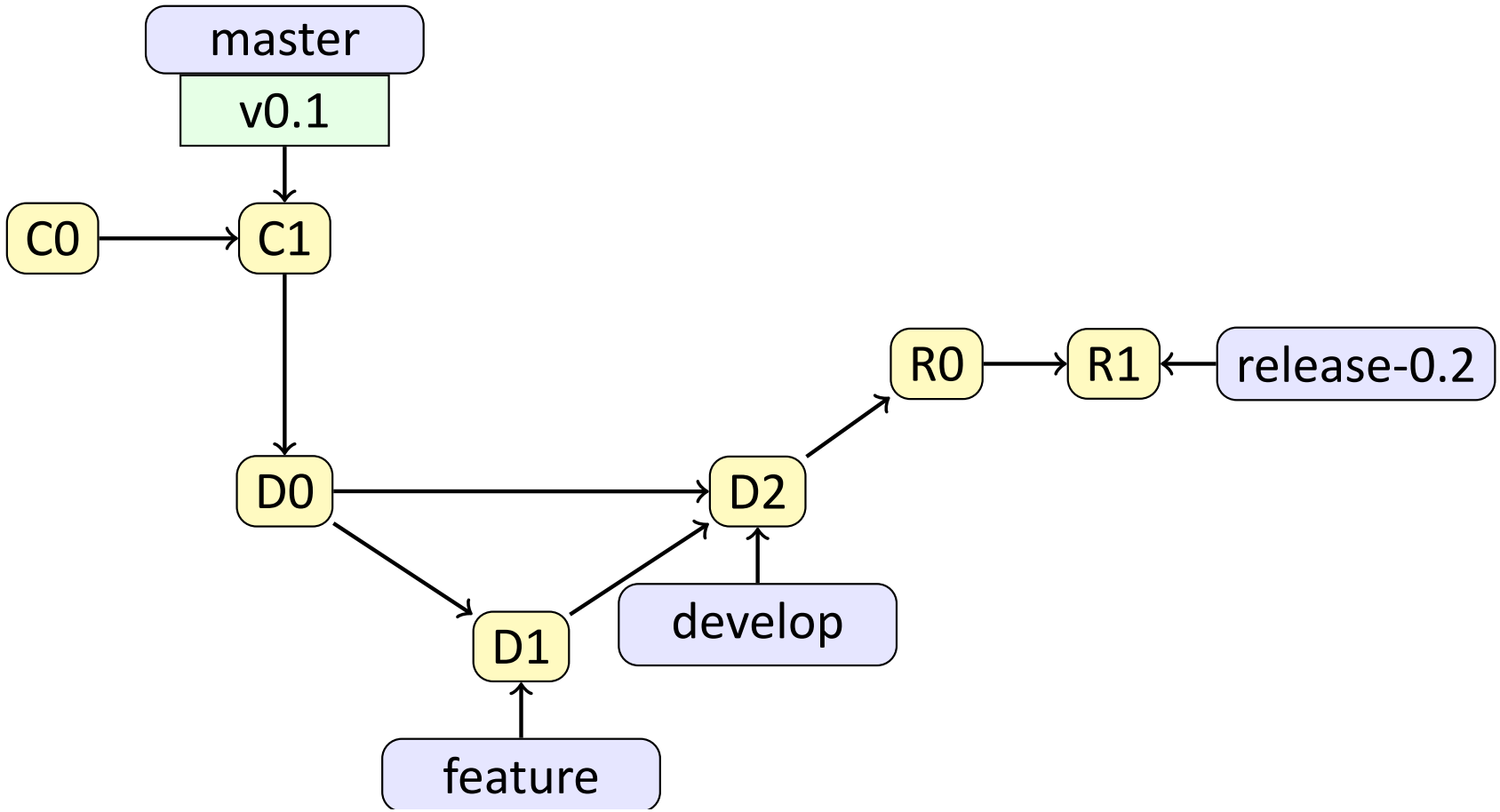
Git – Gitflow



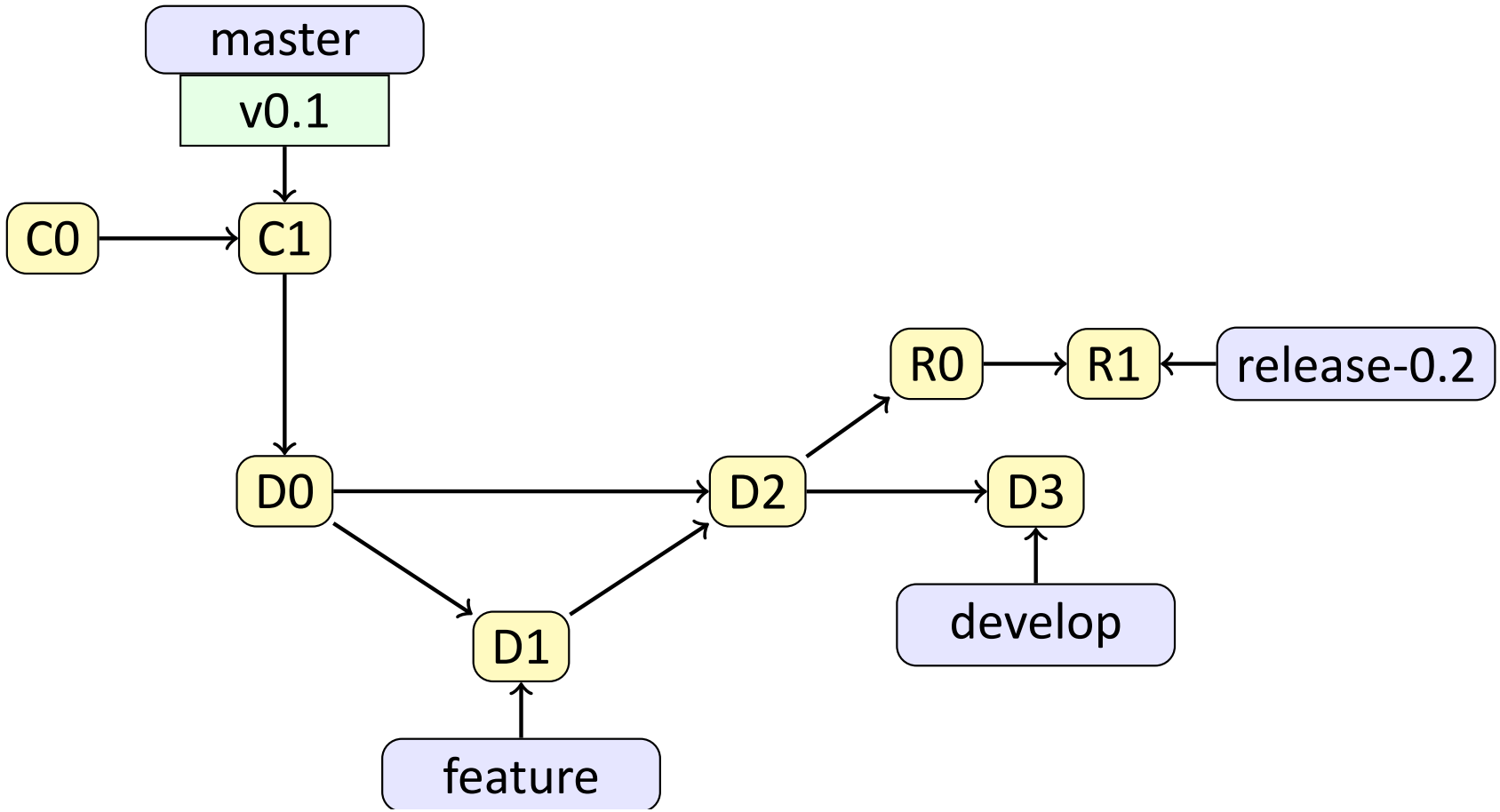
Git – Gitflow



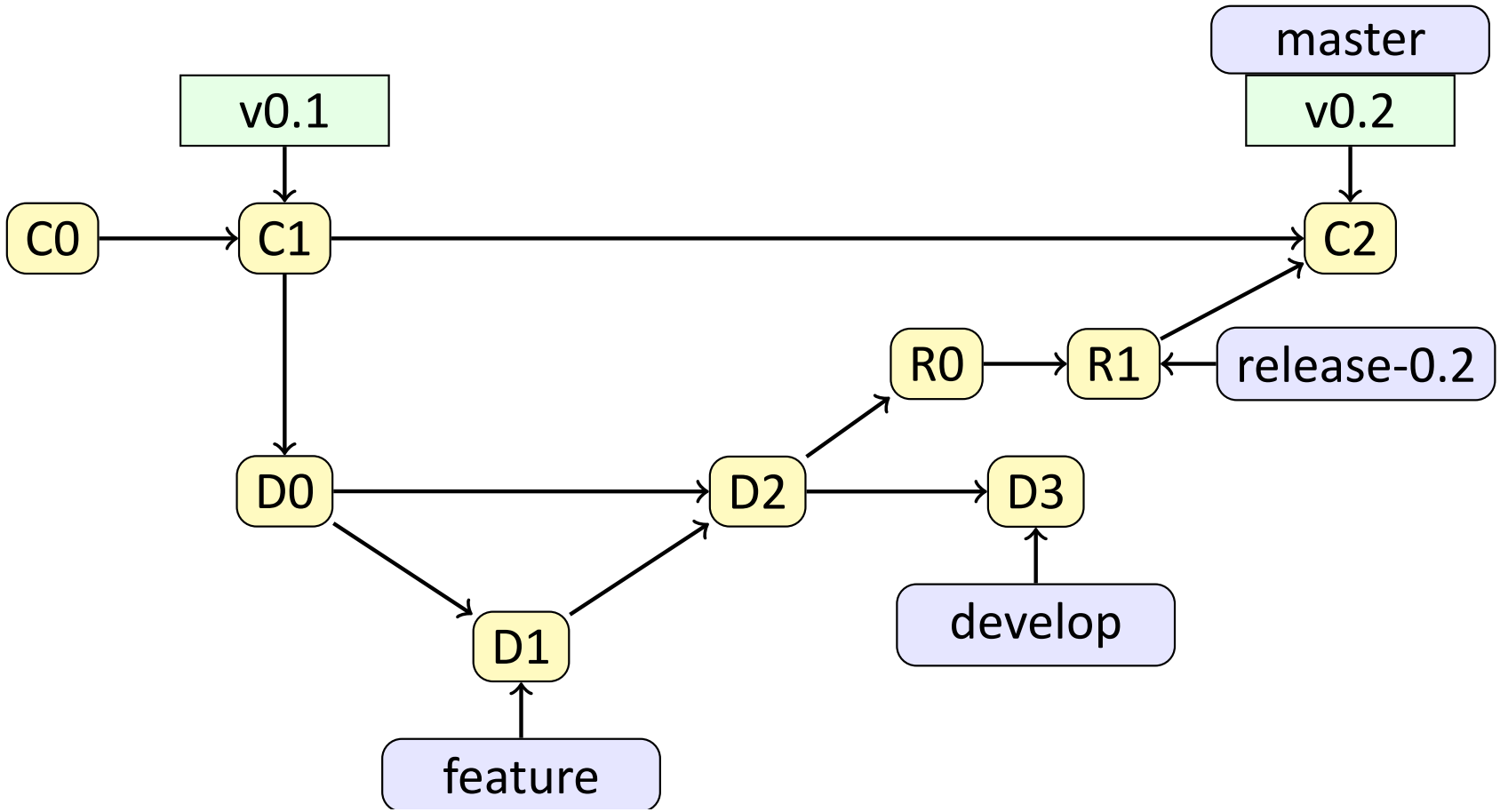
Git – Gitflow



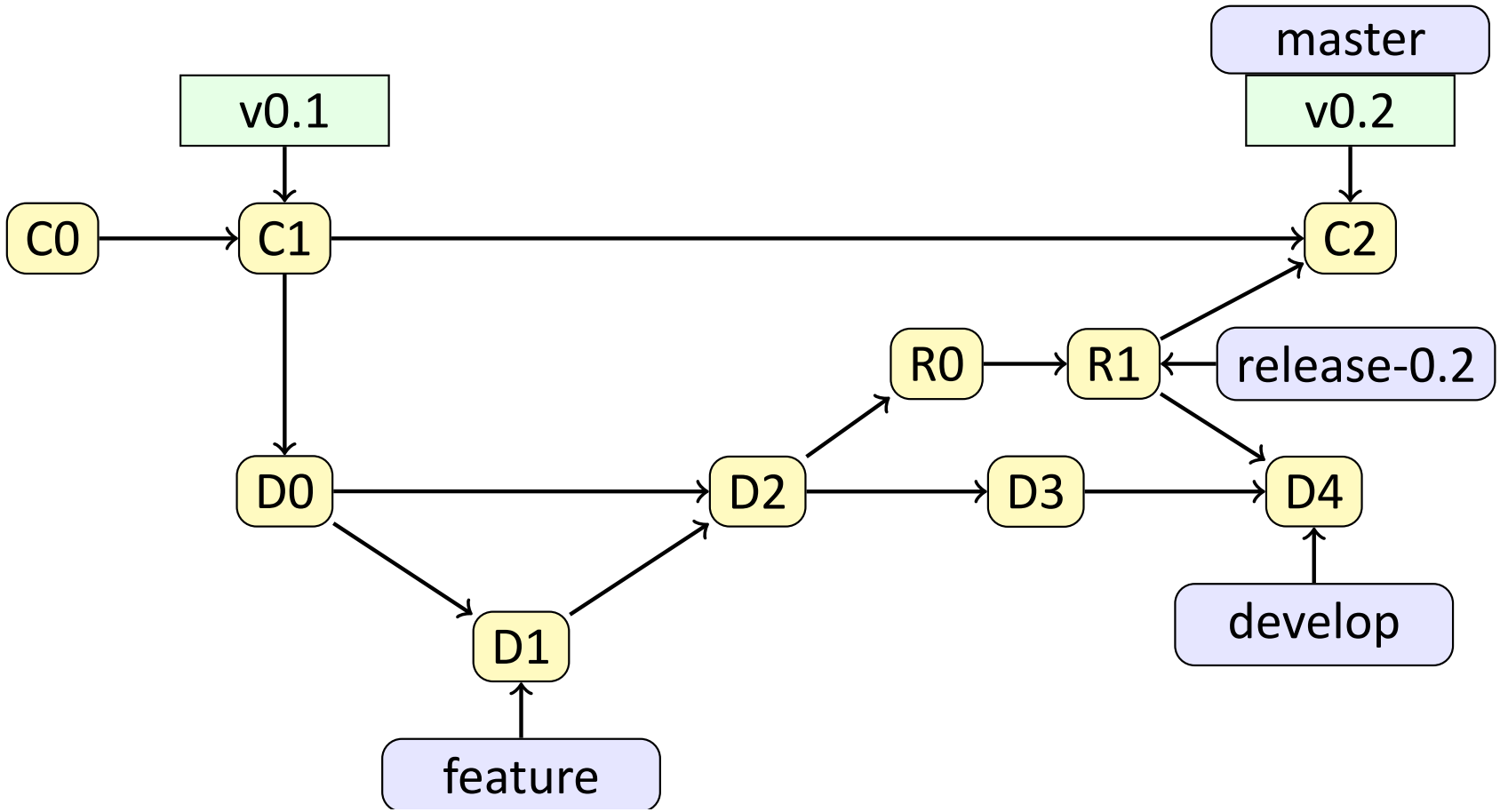
Git – Gitflow



Git – Gitflow


master
v0.1
v0.2
C0
C1
C2
R0
R1
release-0.2
D0
D2
D3
develop
D1
feature

Git – Gitflow

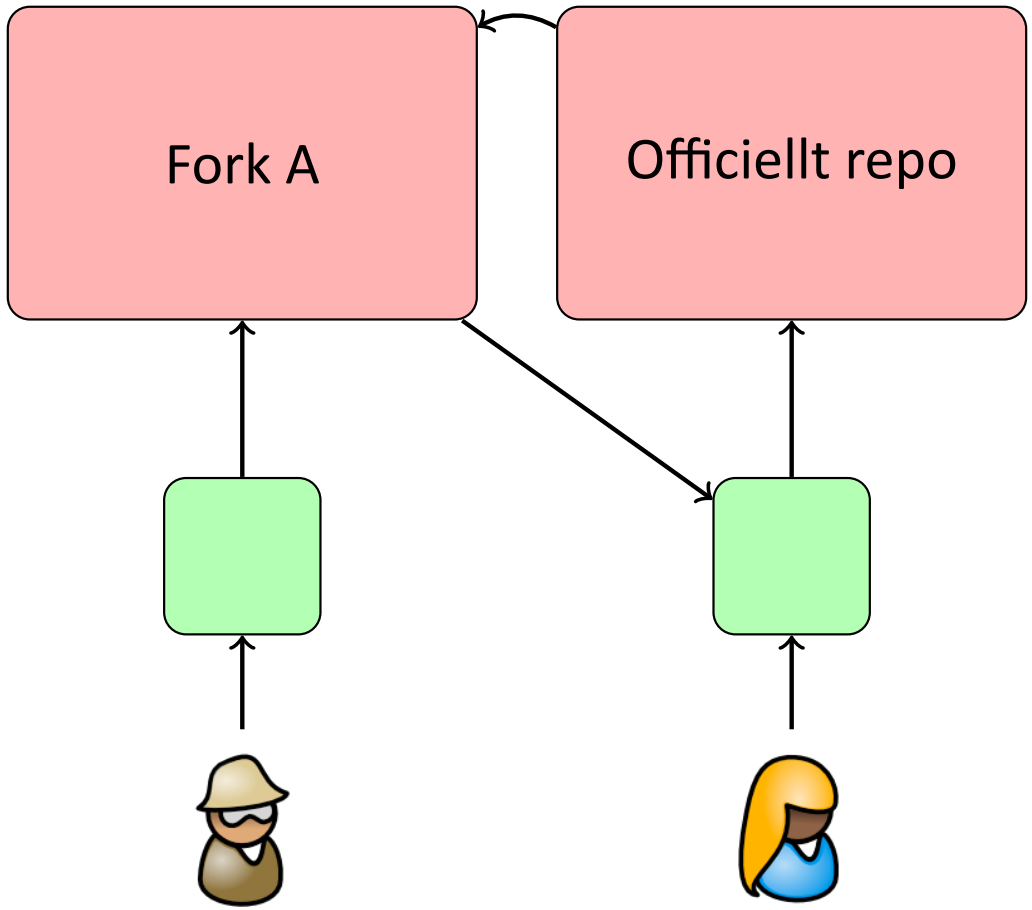

master
v0.1
v0.2
C0
C1
C2
R0
R1
release-0.2
D0
D2
D3
D4
D1
develop
feature

Git – Gitflow

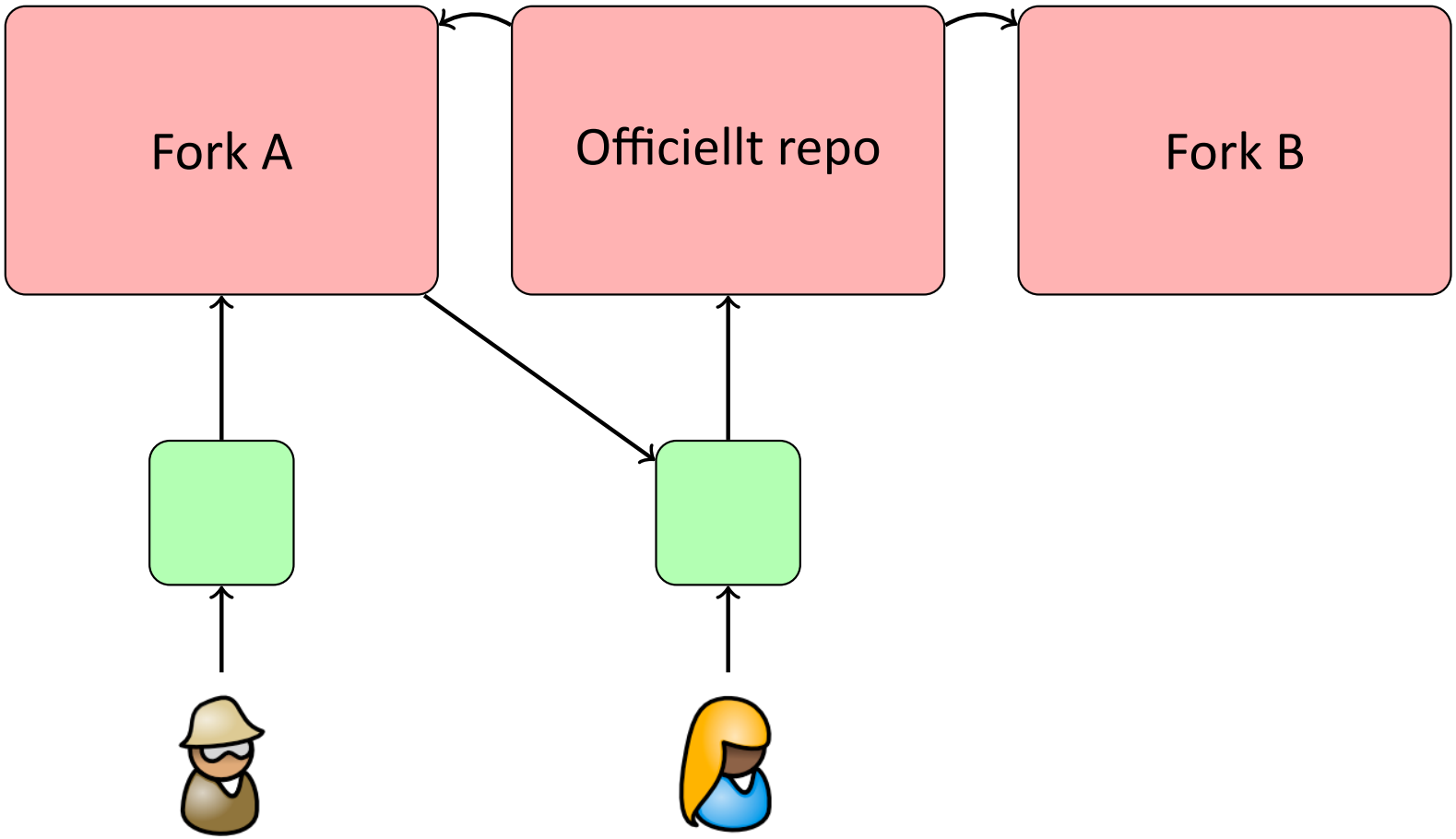
- Release kan testas och färdigställas parallellt med utveckling av ny funktionalitet
- `master` visar tydligt alla versioner
- Enkelt och tydligt att göra hotfix

- Ännu mer att hålla reda på, namngivning är viktig!

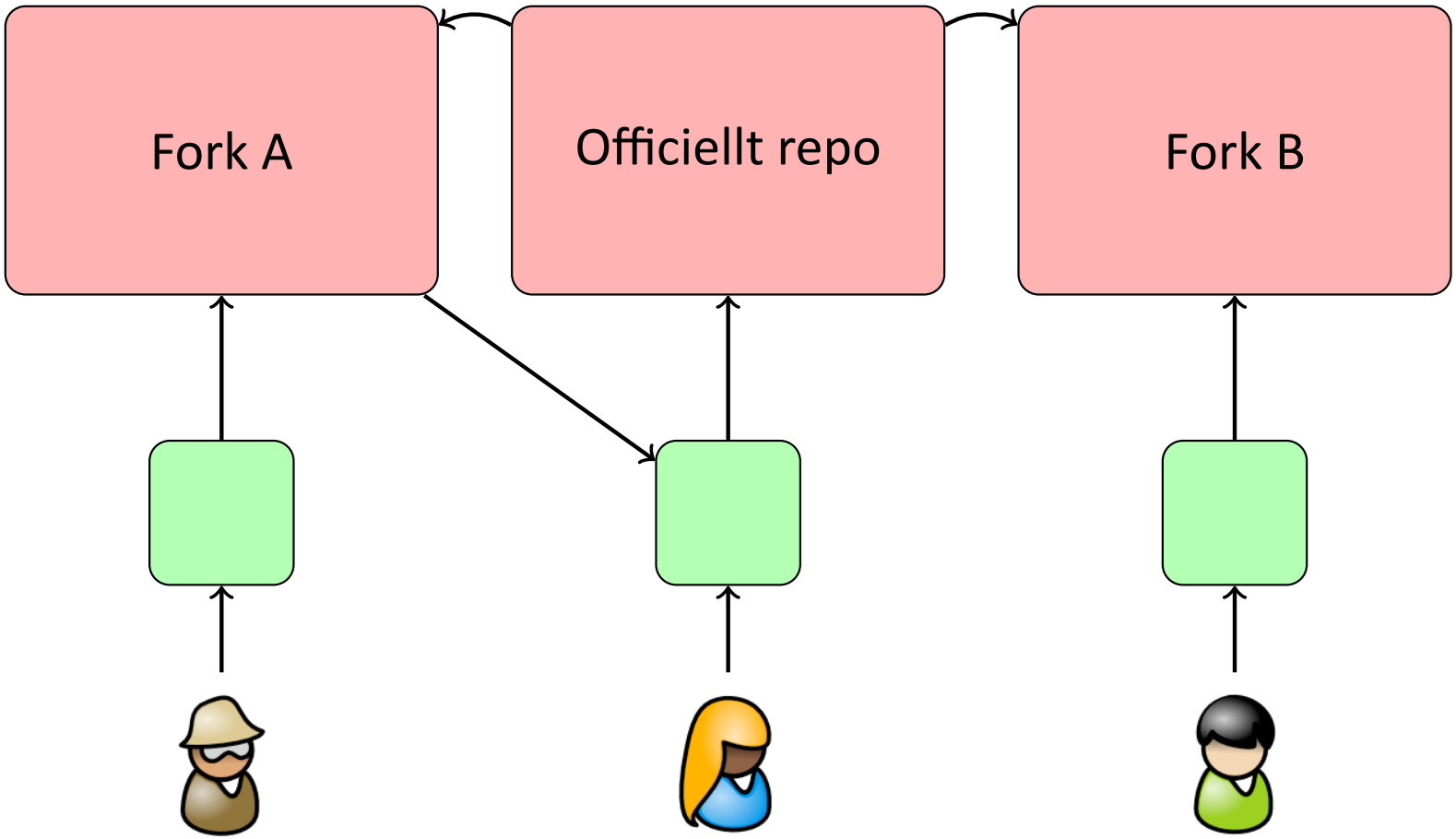
Git – Integration-Manager (Forks)



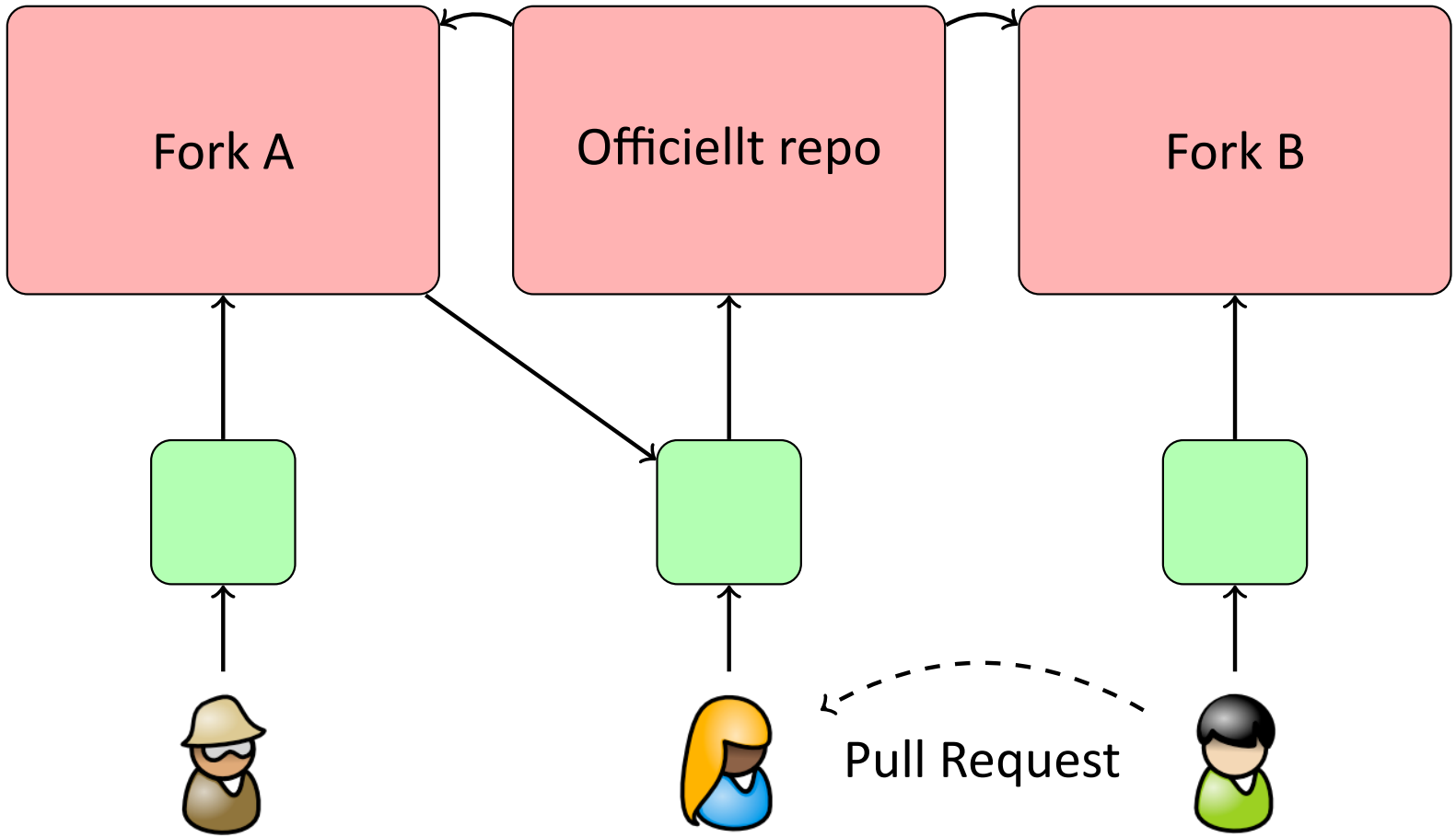
Git – Integration-Manager (Forks)



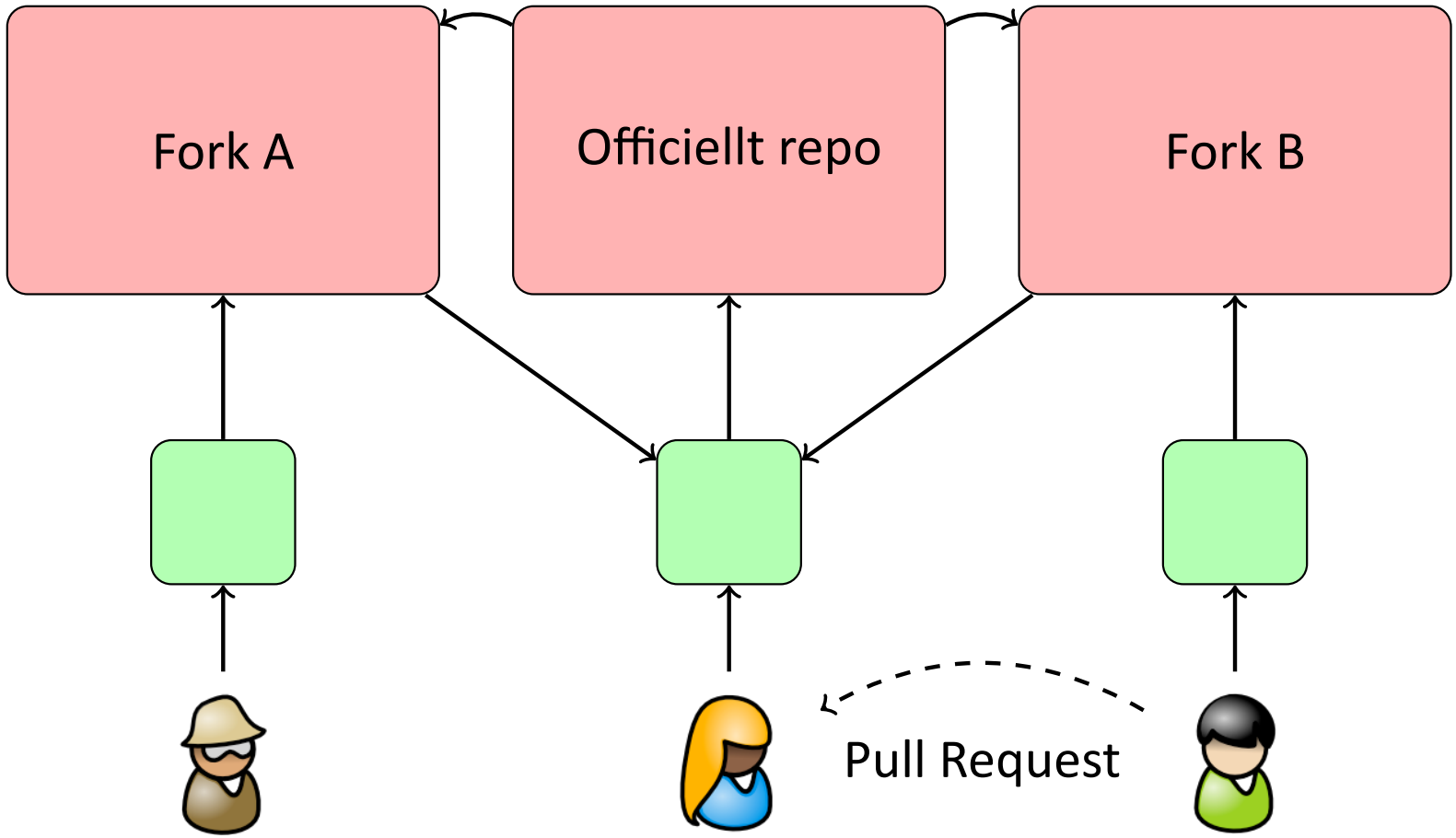
Git – Integration-Manager (Forks)



Git – Integration-Manager (Forks)



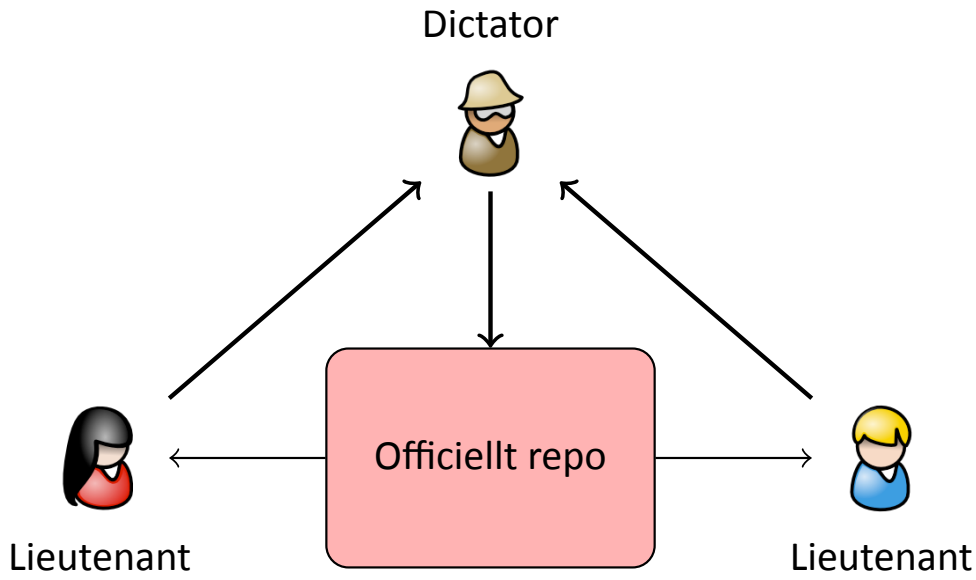
Git – Integration-Manager (Forks)



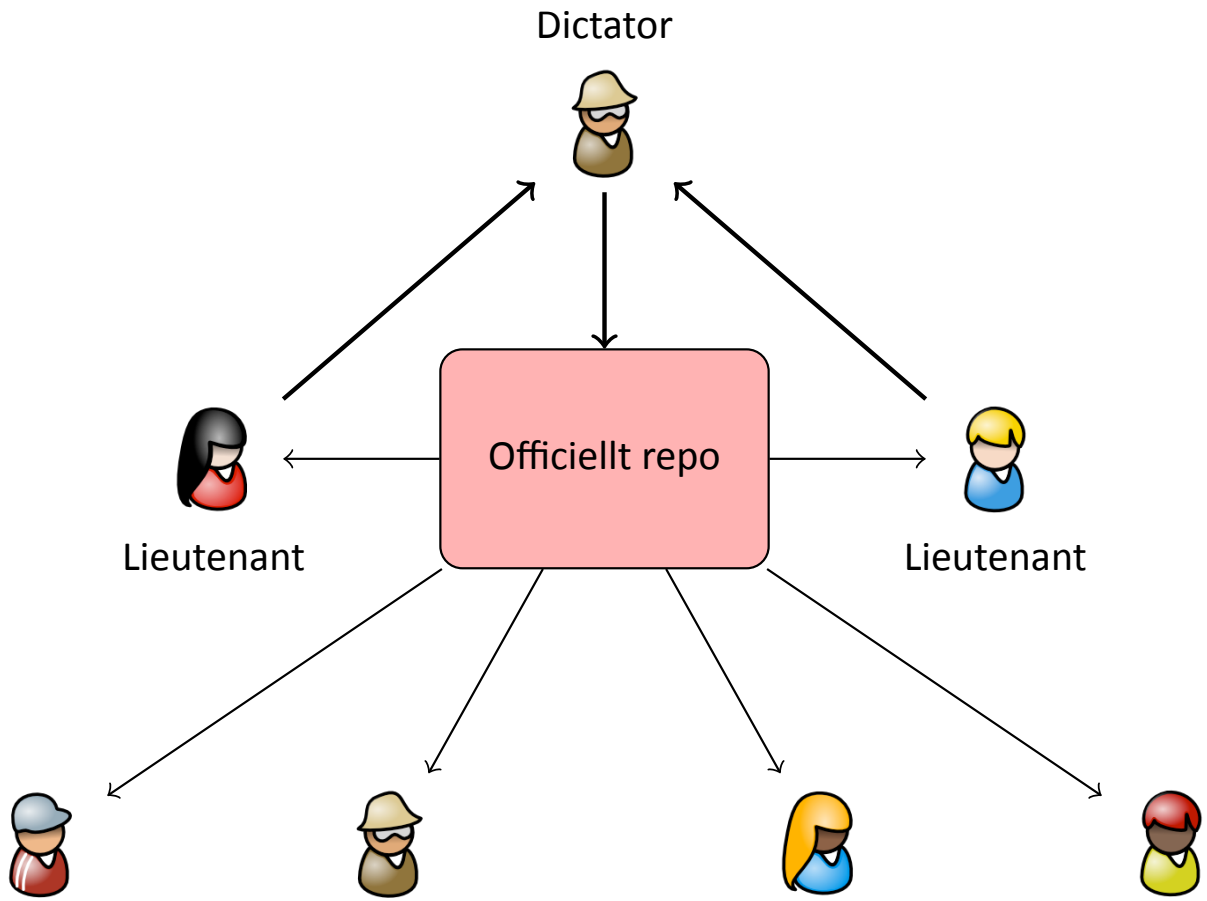
Git – Integration-Manager (Forks)

- Ägaren har full kontroll på vad som kommer med
- Pull Requests ger tydlig diskussion och genomgång av ändringar
- Officiellt repo innehåller få ”skräpgrenar”
- Alla kommer åt sina ändringar var som helst ifrån
- Enkelt att ”tappa bort” ändringar
- Enkelt att hamna efter officiellt repo
- Onödigt med flera kopior av projektet?
- Lite krångligare att komma igång

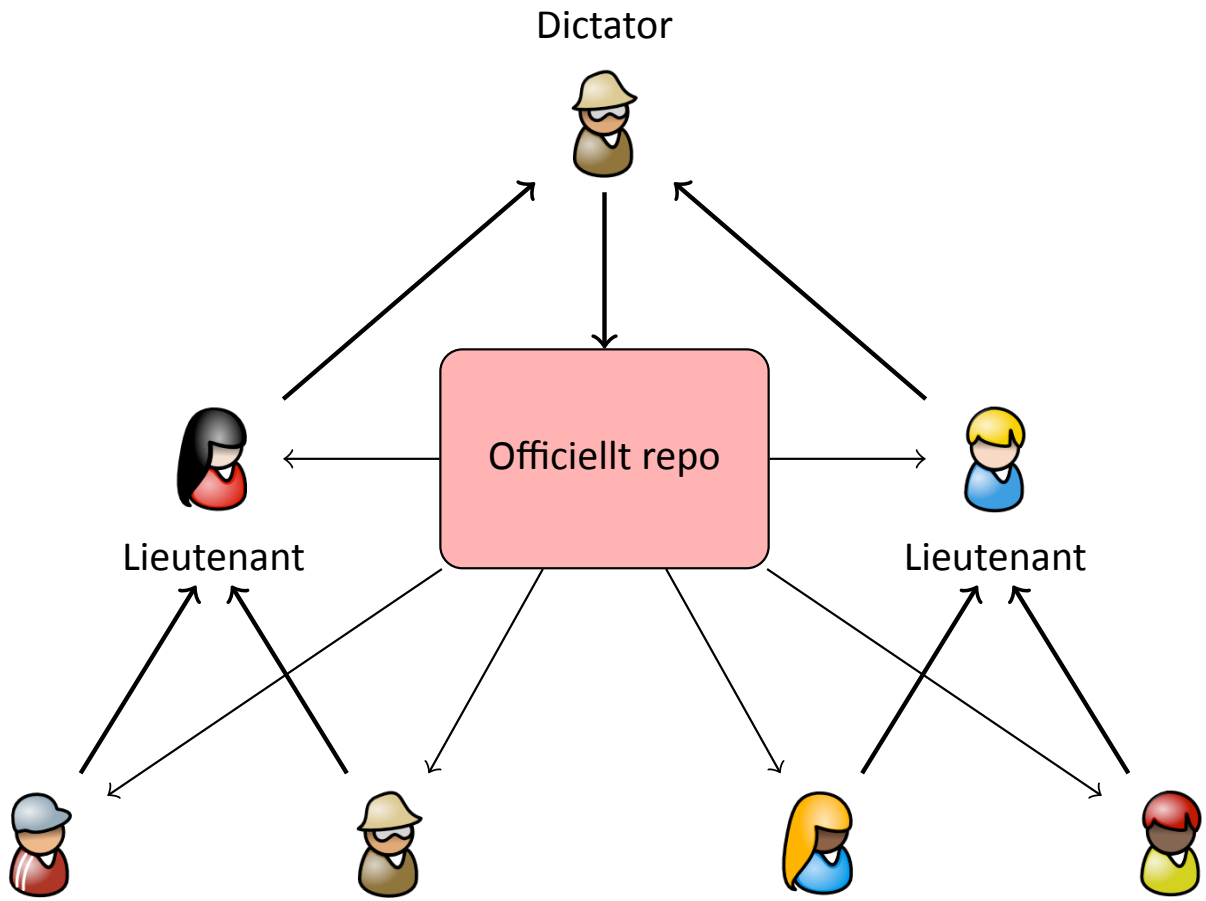
Git – Dictator and Lieutenants



Git – Dictator and Lieutenants



Git – Dictator and Lieutenants



Git – Dictator and Lieutenants

- Bra för väldigt stora projekt (Linux)
- Enskild ägare behöver inte kontrollera allt

- Lång process att få ändringar accepterade
- Många personer inblandade

www.liu.se

LINKÖPING UNIVERSITY