A hitchhikers guide to MXCuBE development

-

a discussion starter

Rasmus Fogh,
Global Phasing

MXCuBE meeting. Diamond, 1 February 2018

GΦL

Contents

- **Introduction**
- Part 1: Code
  - Repositories
  - Installation and dependencies
  - Modularity
- Part 2: Collaboration
  - Versions and code flow
  - Refactoring
  - Testing

GΦL

Why this talk?

- Starting point for discussion
  - Where are we?
  - Where do we want to be?
  - What should we do to get there?
  - What can we do and what should we let drop?
- GΦL has resources to contribute
  - How best to employ them?

MXCuBE collaboration

- Many dispersed groups
  - Who are both users and developers
  - No release to external parties
- Everybody work to their own beamline
  - You need a beamline for proper testing
- Different hardware, different projects – different versions
- Intense time pressure

GΦL

At the end of a shut-down:

Does Sir want his tea now, or does Sir prefer to wait till it is ready?

GΦL

Contents

- Introduction
- **Part 1: Code**
  - **Repositories**
  - **Installation and dependencies**
  - **Modularity**
- Part 2: Collaboration
  - Versions and code flow
  - Refactoring
  - Testing

GΦL

Repositories

- Two active user interfaces (web and Qt4)
- Application combined from several repositories:
  UI, HardwareRepository, …
  - **NOT** a problem,
  - as long as it is clear what goes where
- We need one more repository:
  - Developing with submodules is clumsy
  - Separate submodules-only ‘Release’ repository.
  - Use to combine different repositories into a release

GΦL

Installation system

- No single-step installer
- No up-to-date dependency list
- It works on the development machine, but …
- Sort out your own dependencies

- Single-step installer for easy setup
  - With docker to test on a standard OS version

- Matias has a proposal (?)

GΦL

Modules with clear interfaces

- Good practice
  - *necessary* for multi-branch/site coding
- We need functions that
  - Do all you need
  - Are clearly defined
  - Have the same effect in all branches/sites
  - A lot already done: Abstract and Mockup classes
- *Respect* the interface
  - Implementation details in private functions
- Matias has a proposal (?)

GΦL

Example: moving motors

- Some classes have `moveMotors()`
- Others have `move_motors()`
- Some have both – *what you gonna call?*

- `move_motors(motor_positions)` does NOT always move the motors to the input positions
  - Moving kappa may change alignment motors after the fact
  - Kappa or phi may not be moved at all (!)

GΦL

Example: moving more motors

- AbstractMotor:
```
def move(self, position,
         wait=False, timeout=None):
```
- MD2Motor:
```
def move(self, absolutePosition,
         timeout=None, wait=False):
```
- SardanaMotor:
```
def move(self, absolutePosition)
```
- Resolution:
```
def move(self, pos, wait=False)
```
- *What you gonna call?*

GΦL

Contents

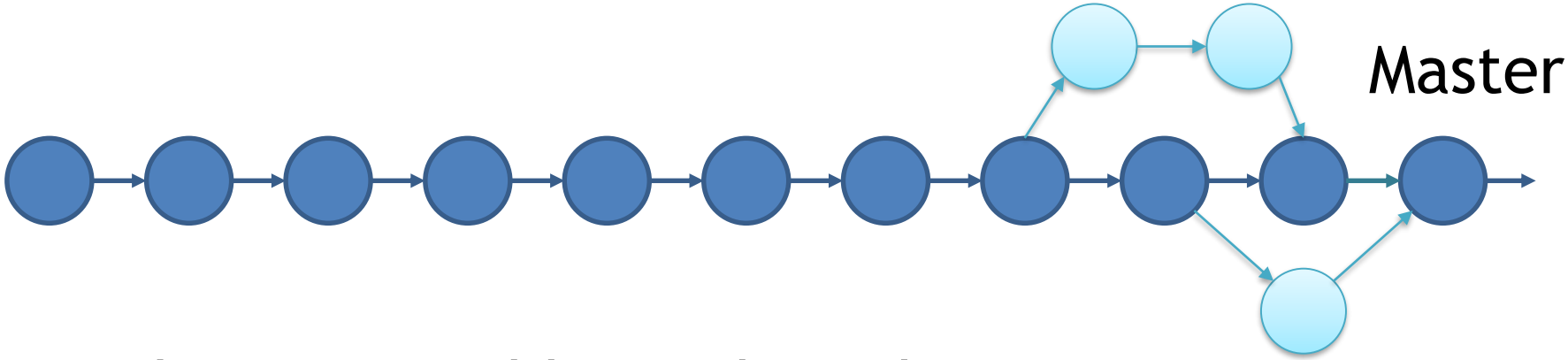
- Introduction
- Part 1: Code
  - Repositories
  - Installation and dependencies
  - Modularity
- **Part 2: Collaboration**
  - **Versions and code flow**
  - Refactoring
  - Testing

GΦL

Current versions

Lab	UI	HwObj	Future UI	future HwObj	Change date
MaxIV	web/master	2.2	web/master	master?	TBD
EMBL-HH	Qt4/master	master	Qt4/master	master	n/a
ESRF	Qt3/2.1	2.1	web/master	master?	Q1 2018
SOLEIL PX1	Qt3/2.1	2.1	Qt4	master?	Q1 2018
SOLEIL PX2	Qt3/2.1	2.1	web	master?	Q1 2018
BESSY	Qt4/2.2	2.2	Qt5??	master?	Q1 2018
DESY P11	Qt4/master	master	Qt5??		??
ALBA	Qt4/master	master			n/a
Elettra	web/master	2.2 (Submodule)			
LNLS-Brasil	Qt4/2.2	2.2	Qt4/master		ongoing

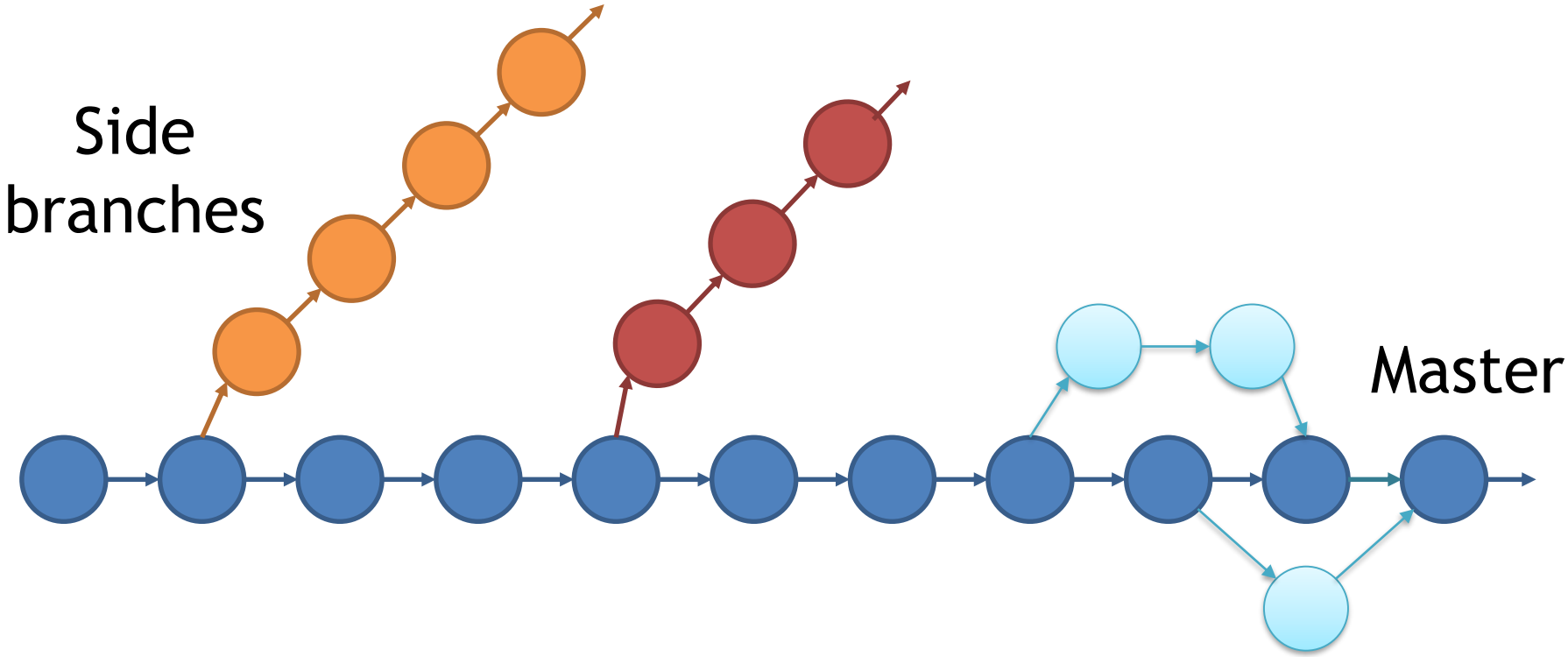
Current status - master



Hamburg, Grenoble, and Lund
work closely together,
test continuously,
and upgrade to newest version

GΦL

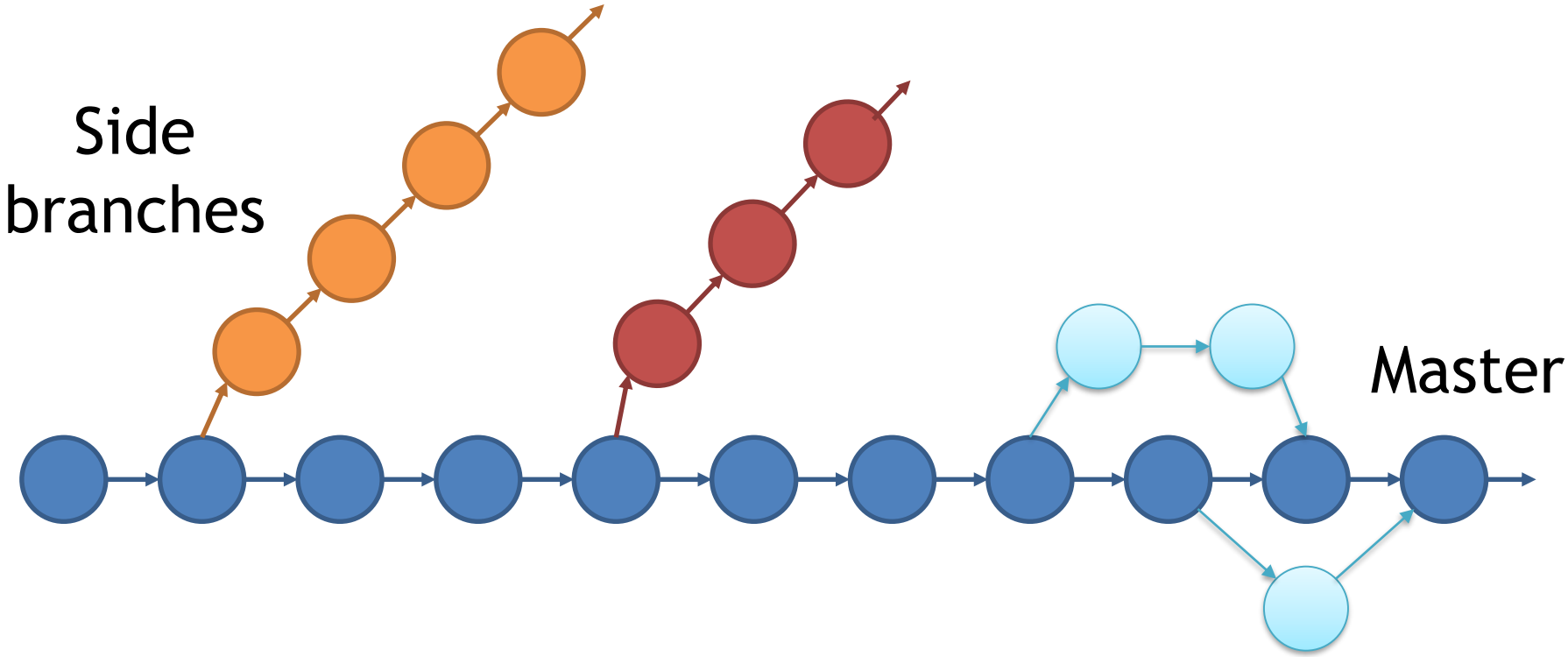
Current status - overall



Each site programs for
 – and tests on –
their own beamline

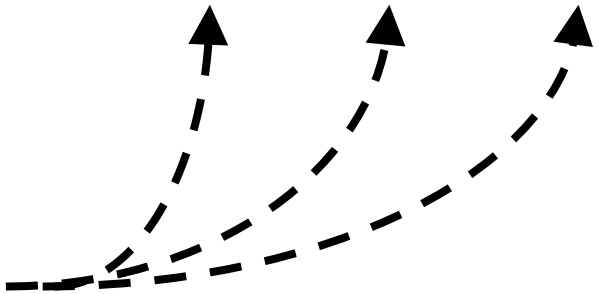
GΦL

Hard to upgrade



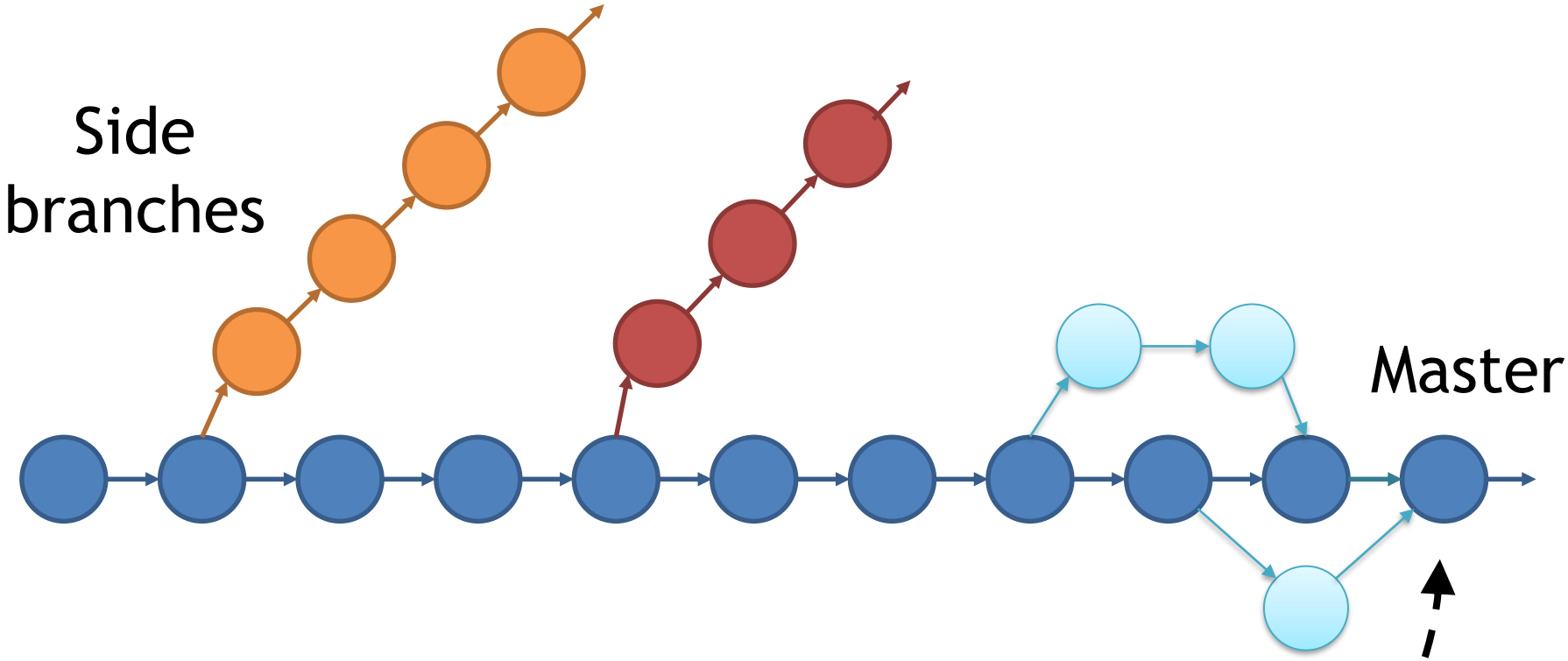
Which commit is
- Consistent?
- Tested?
- Going to be supported?

How much adaption is needed?



GΦL

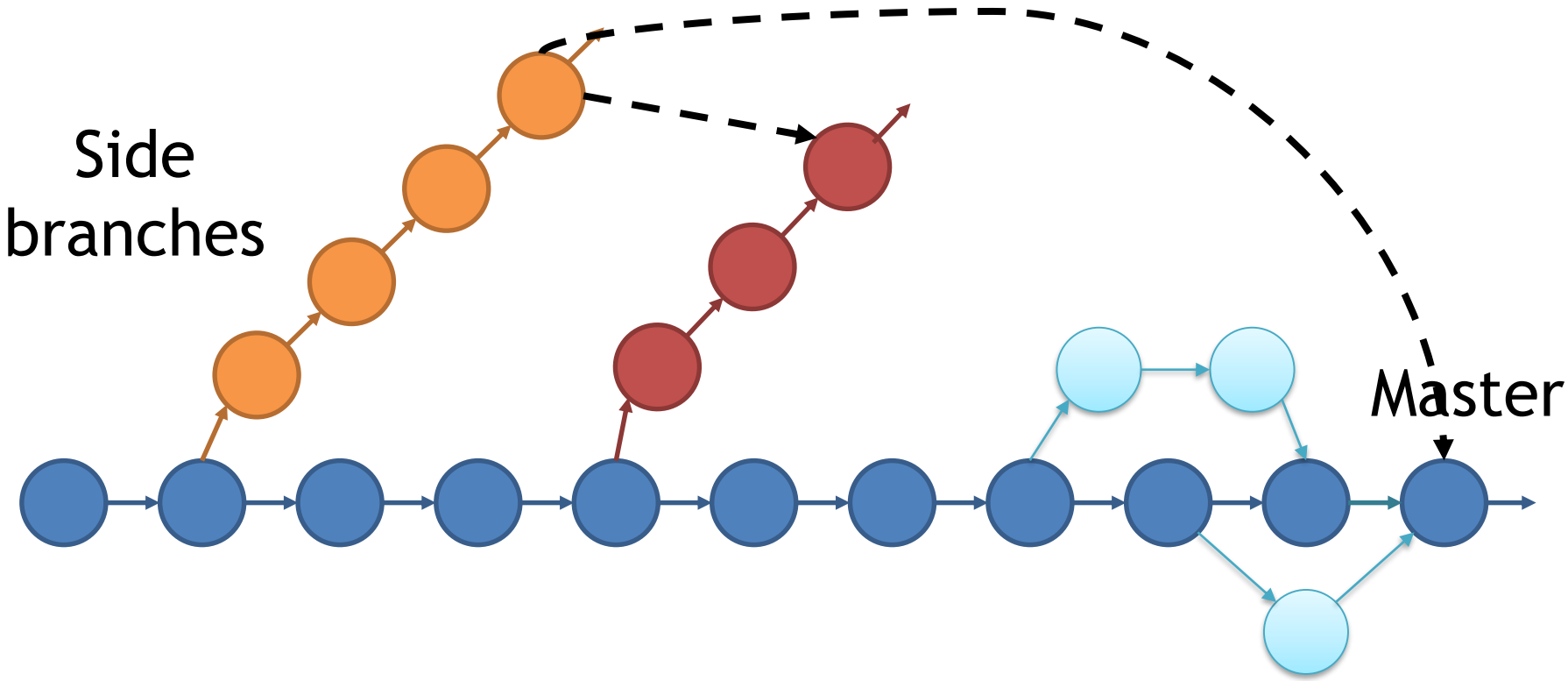
Hard to contribute



- You cannot test without a beamline
- Additions – and bugs –
  go right into production
- Other, busy, people have to test for you

GΦL

Hard to synchronise



- Your changes need to go to several places that you do not have running locally
- If not synchronised you will lose them on upgrade

GΦL

Multiple, divergent branches

- A single shared release would be ideal
  - But would require all sites to agree and move in tandem
  - Is that realistic?
- The greater the divergences, the harder any collaboration
- All the nifty Git workflows assume a *single* active, released branch
- All solutions require extra work

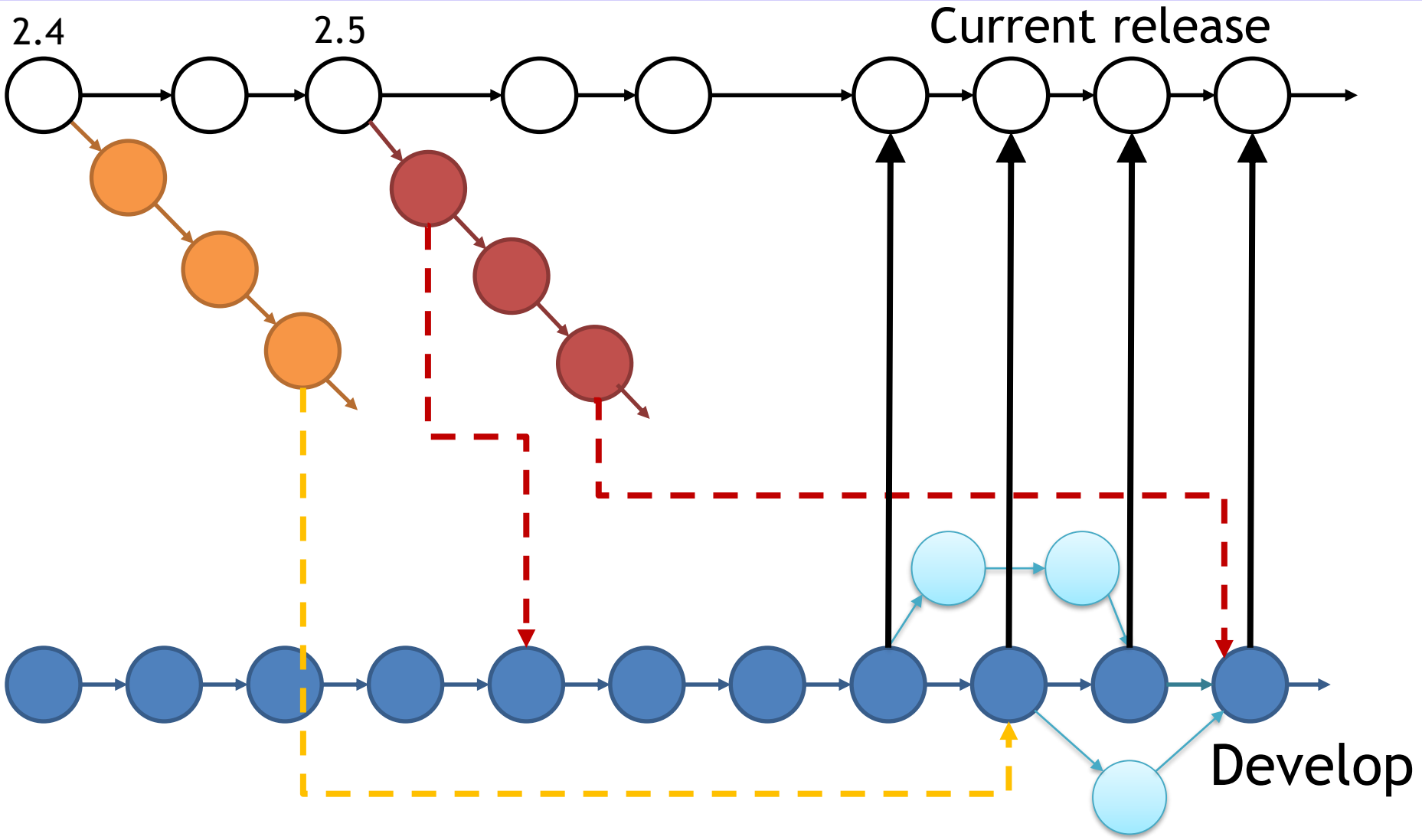
GΦL

Solution: Releases

- All production code comes from a release
  - No development branch in production
  - Releases are supported
- Code starts in development branch
  - is tested (again) on move to master release branch
- Older releases become side branches, if supported
- **Each release is managed by the beamline teams that use the release**

GΦL

Workflow proposal



GΦL

Leading edge release

- Current release can be gradually updated in master branch
- All changes are synced with development branch
- Releases are tagged when other sites upgrade
- Responsible sites help with merging and testing of contributions

GΦL

Side branch releases

- Changes and additions must be synced to development branch
  - otherwise you lose them next time you update
- All code (including config) must be checked into git for people to see
- If you keep your branch ready, updates are easier (and will get communal support)

- Too old releases lose MXCuBE support (!)

GΦL

Contents

- Introduction
- Part 1: Code
  - Repositories
  - Installation and dependencies
  - Modularity
- **Part 2: Collaboration**
  - Versions and code flow
  - **Refactoring**
  - Testing

GΦL

Refactoring

- snake_case v. CamelCase
- Python 3 support
  - Python 2 comes off support in 2020
- Code duplication
- Standard naming conventions
  - It is Obj.name? Obj.name()? Obj.get_name()? Obj.getName()?
- Functions changing content of input collections
  - E.g. {string:value} → {MotorObj:value}
- …

GΦL

Practical example

- I tried to fix some problems
  - Stop objects from exposing internal collections
  - Remove mixed tab/space indentation from code
  - Both count as bugs (!)
- The combined commit changed 115 files
- Pull request rejected

So, what is the problem?

- Any change, however small, might break working code
  - Yes. Serious problem. Needs agreed procedures.
- Too much work to read through and accept
  - Well, OK.
- Hard to compare old and new files by diff when there are so many changes
  - Well, OK.
- ‘Breaks code history’
  - How so? Git preserves the history perfectly!

GΦL

Procedures for changes

- If we want clean, standardised code
  - There must be a way to make it happen!
  - Commit enough resources to integrate changes
  - How should one do it?
    - Make this kind of changes just before a release, to simplify comparisons?
- If we do not have the resources
  - We should drop the standards
  - Decide up front which things will get done - and which things will not

GΦL

Contents

- Introduction
- Part 1: Code
  - Repositories
  - Installation and dependencies
  - Modularity
- **Part 2: Collaboration**
  - Versions and code flow
  - Refactoring
  - **Testing**

GΦL

Unit tests

- Test each module function by function
- Easy to run, useful to catch errors as you go
- It is not *that* bad to set up
- Serves to define specification / interface
  - This is *not* a neutral activity!
- Fit well with modules and abstract classes
  - After code has been modularised?

GΦL

System tests

- Slow – to be done when making a release
- Start the program and run each major functionality
- Use both mock and beamline (if you have one)
  - Collect emulation can maybe help?
- Make an agreed list of tests, so other people can do it too.
- If you see an error, *FIX IT!*

GΦL

Acknowledgements

All of you in MXCuBE

who patiently answered my questions

GΦL

END

GΦL