Modernizing Scientific Software Development

Christopher Harrop & Mark Govett

18th Workshop on High Performance Computing in Meteorology, Sept 27, 2018

Contents

Challenges

To increase forecast skill we need:

- Improved representation of physical processes
- More accurate numerical methods
- Improved initial conditions
- Higher resolution

Challenges

To increase forecast skill we need:

- Improved representation of physical processes
- More accurate numerical methods
- Improved initial conditions
- Higher resolution

And something else...

Challenges

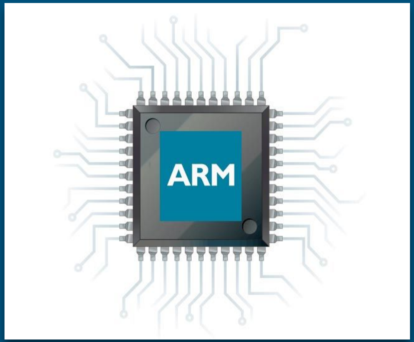
To advance modeling capabilities we need

HIGH QUALITY SOFTWARE

Software Challenges

Rapidly evolving hardware

- Performance portability
- Optimal code structures vary
    - IJK vs KIJ vs ?
- Single source not feasible?
- Flexible design vs optimal performance
- Legacy code modification restrictions



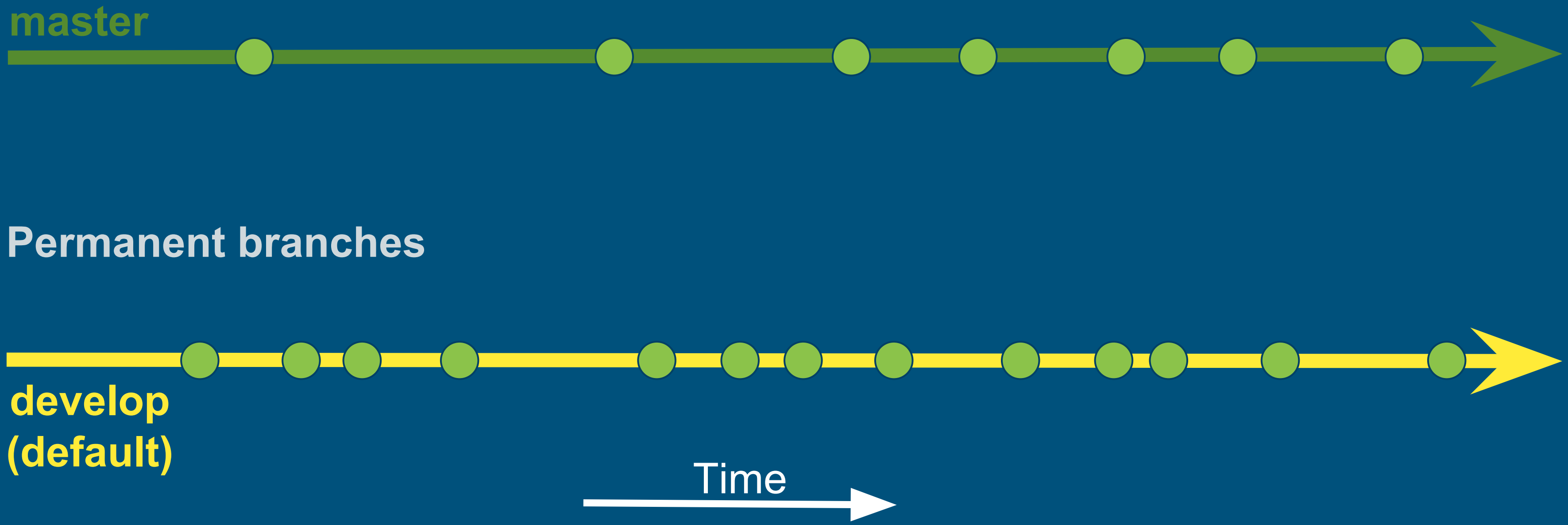
More Fundamental Software Challenges

- Lack of investment in software development
  - Tools, people, expertise, rigorous processes
- Having tools is not sufficient
  - You also have to know how to use them
- Sloppy code management
  - Multiple mirrors, unclear policies, stifling of collaboration
- Conflation of science with software
  - Inadequate testing of software correctness
- Leveraging previous success
  - Avoiding previous failures
- Cultural inertia

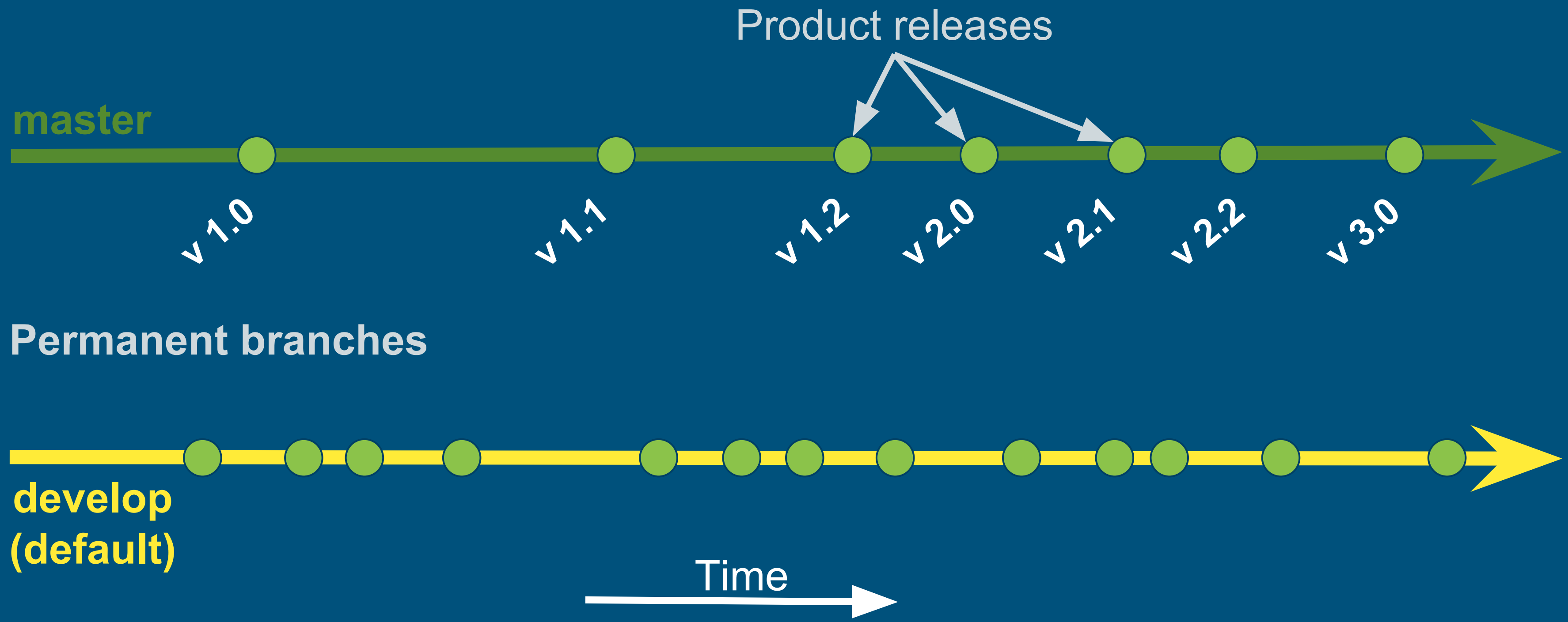
Repository Branching Run Amok

- No discernable repository branching methodology
- Free-for-all branching
- No authoritative “stable” development branch
- Unbounded scope/purpose
- Infinite lifespan
- Branches not merged back to main development
- Branches do not keep up with main development

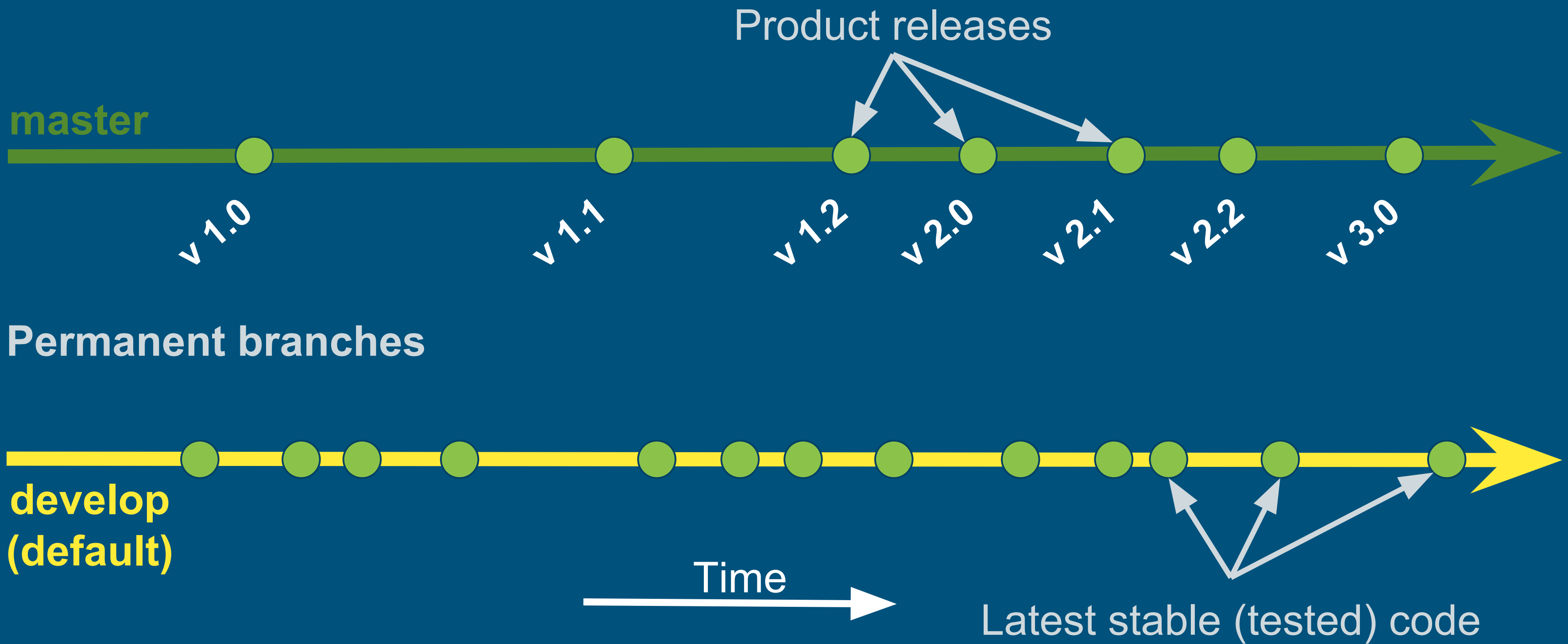
Branch Management With Git-Flow



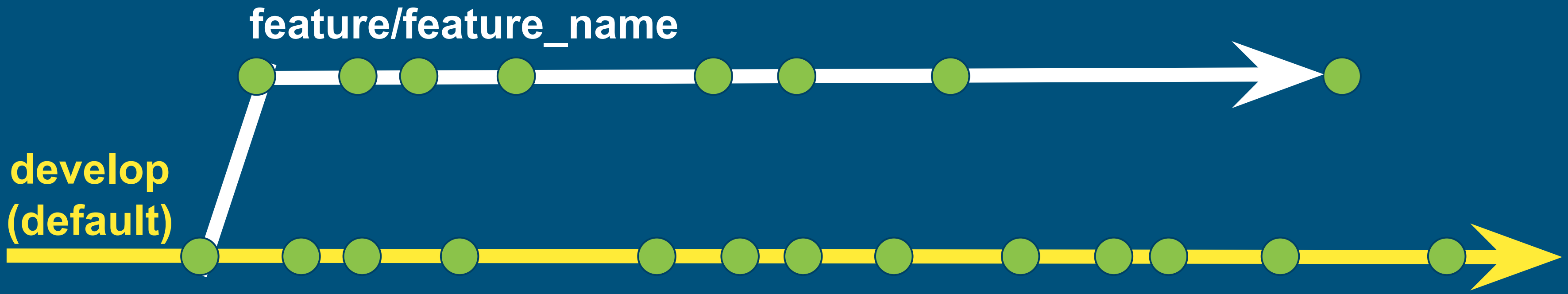
Branch Management With Git-Flow



Branch Management With Git-Flow



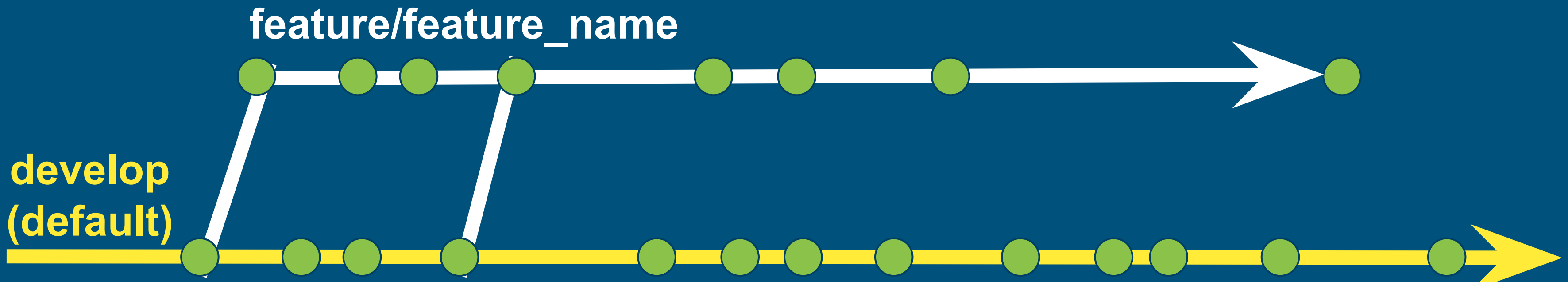
Branch Management With Git-Flow



Feature branches

Singular purpose

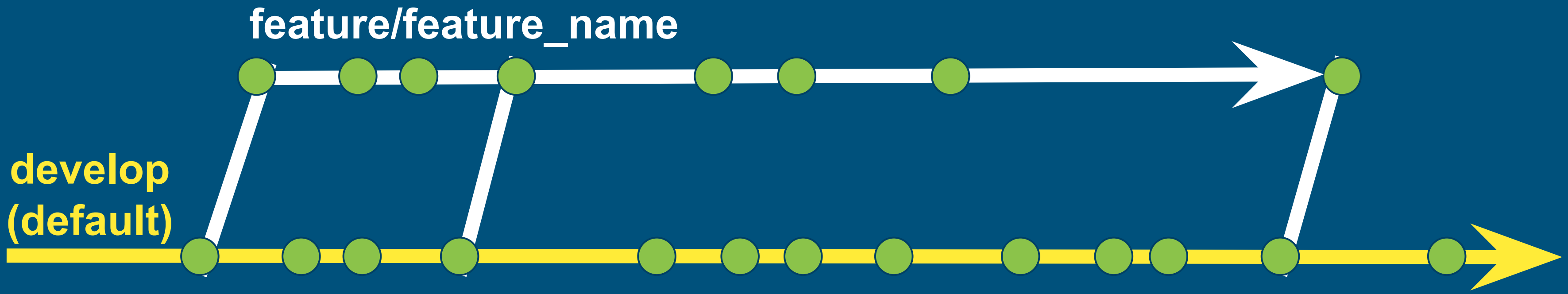
Branch Management With Git-Flow



Feature branches

Singular purpose

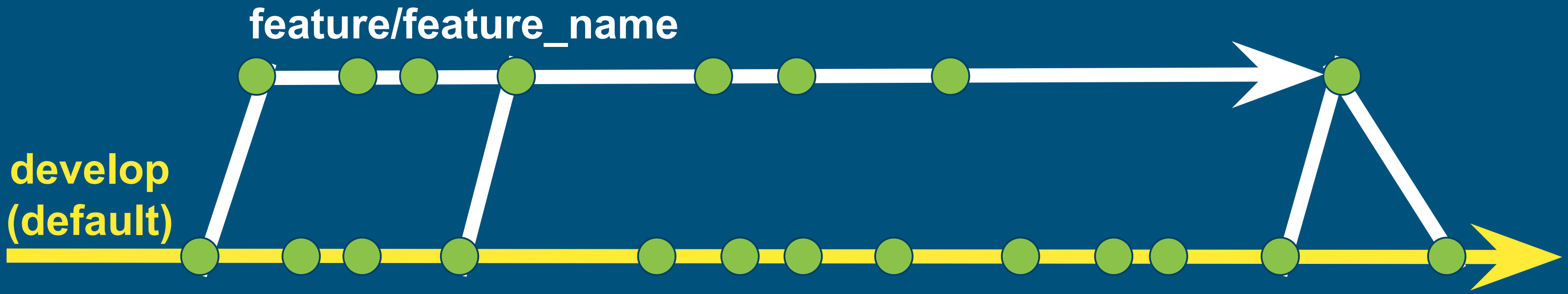
Branch Management With Git-Flow



Feature branches

Singular purpose

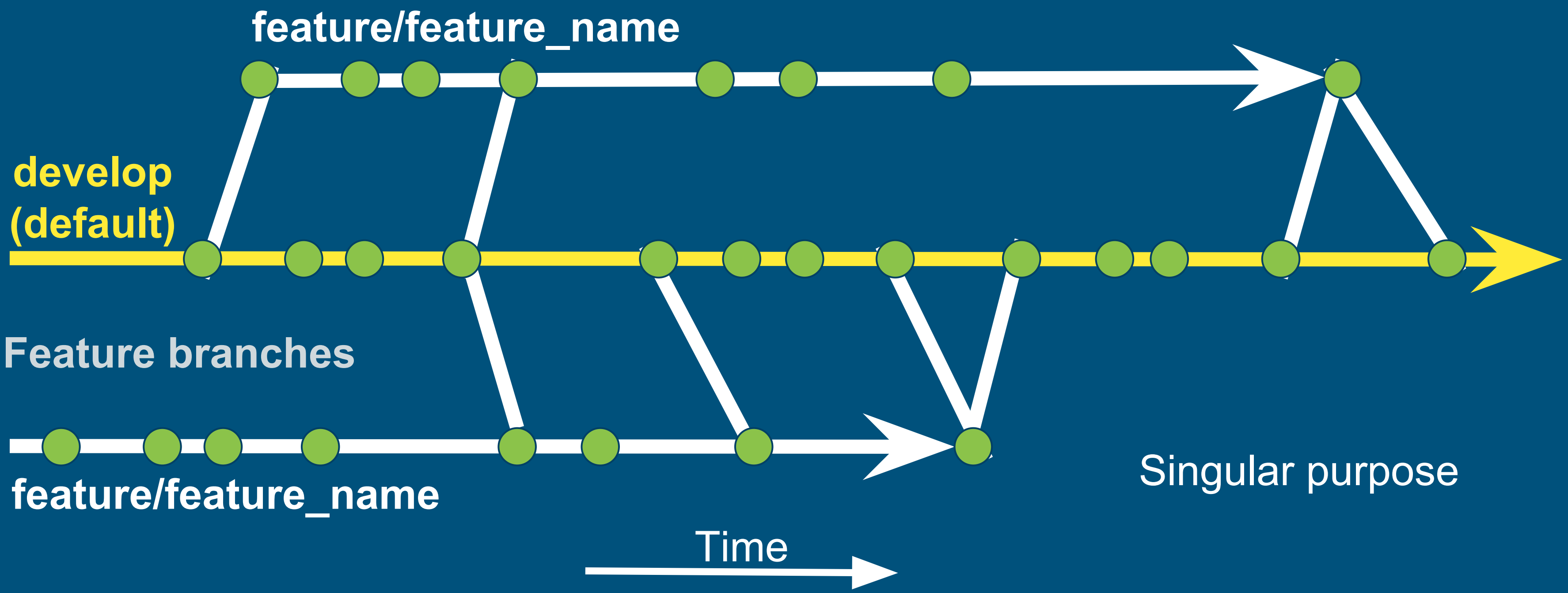
Branch Management With Git-Flow



Feature branches

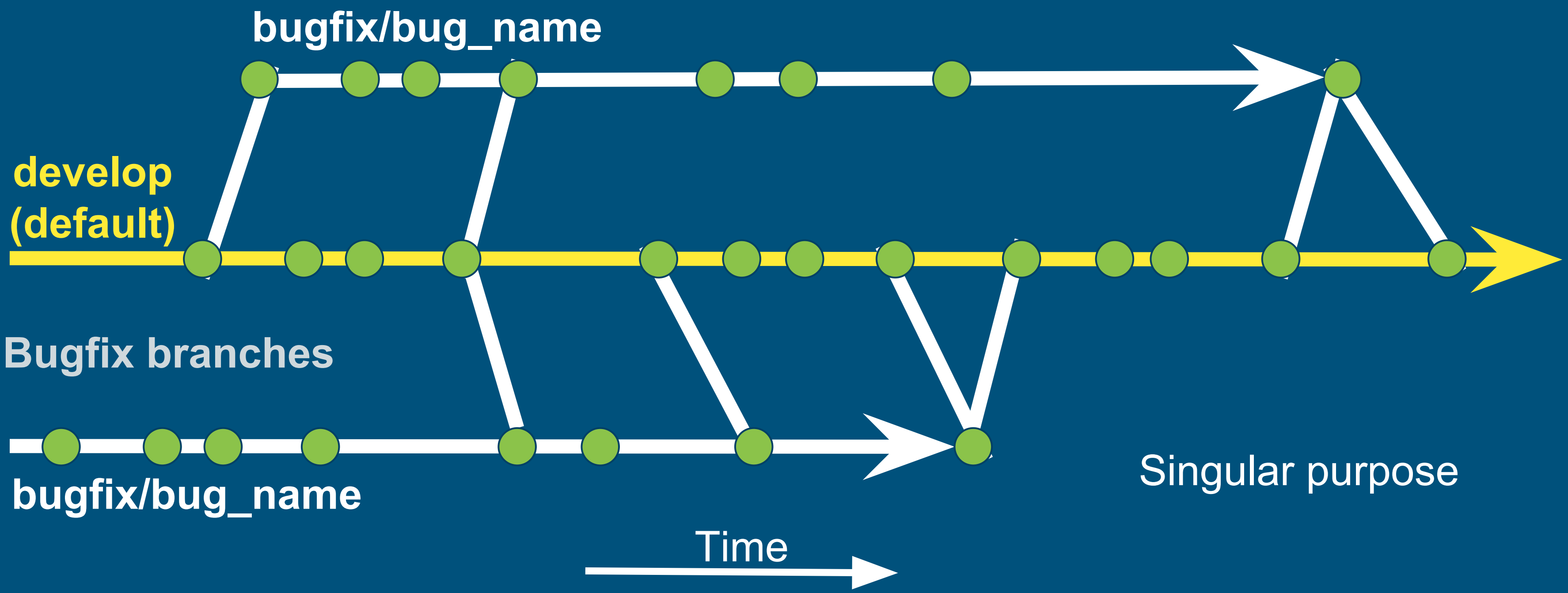
Singular purpose

Branch Management With Git-Flow



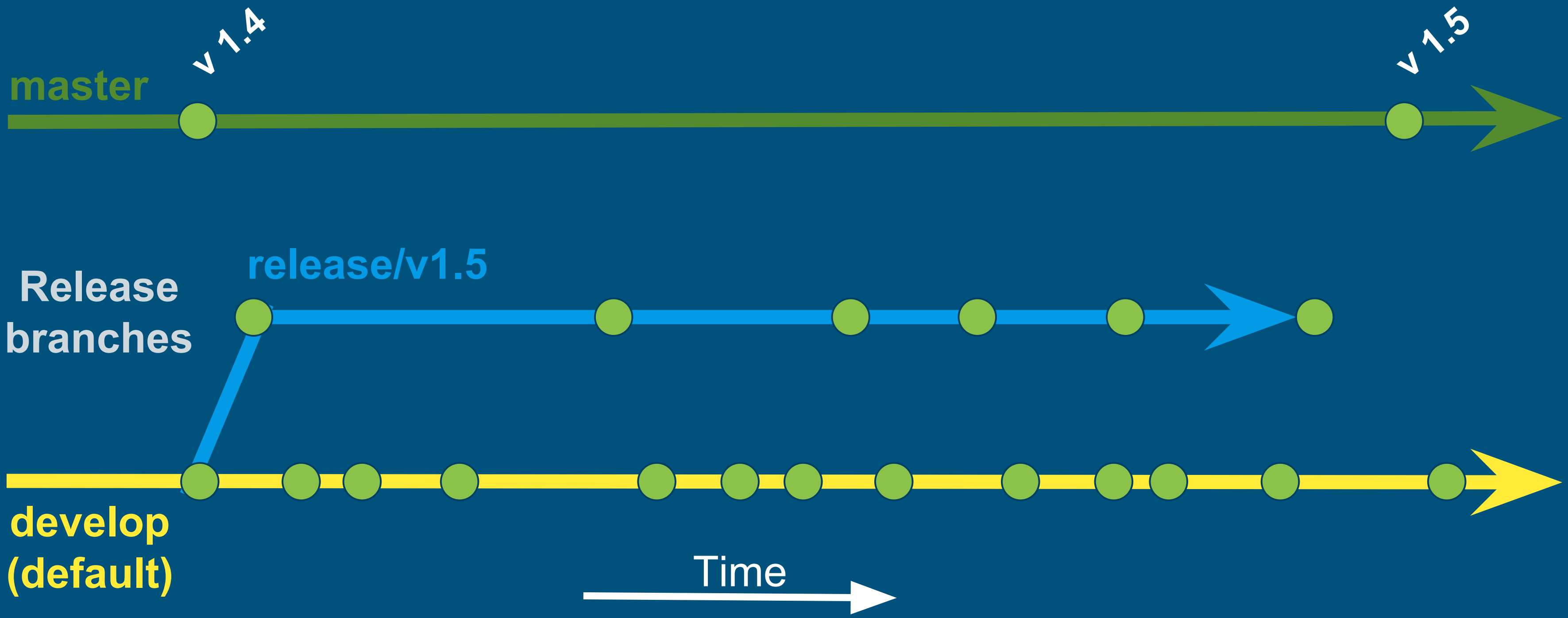
Singular purpose

Branch Management With Git-Flow

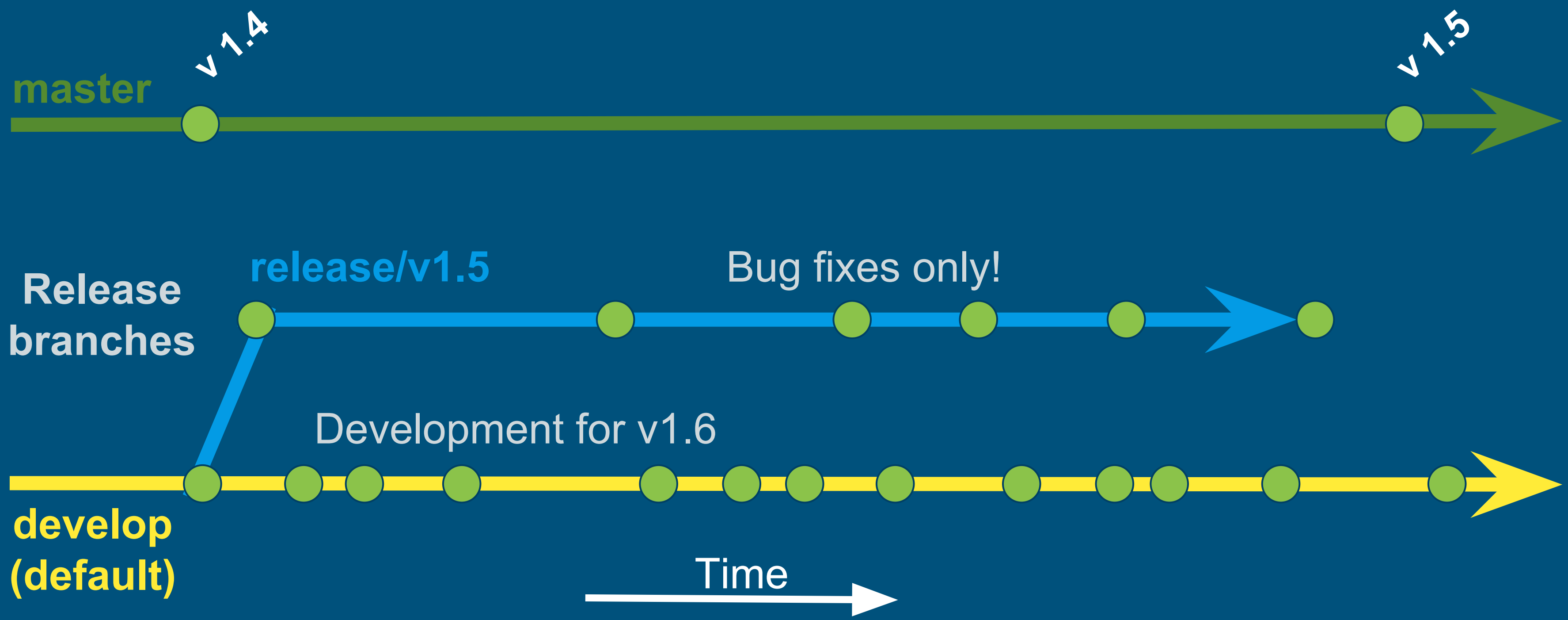


Singular purpose

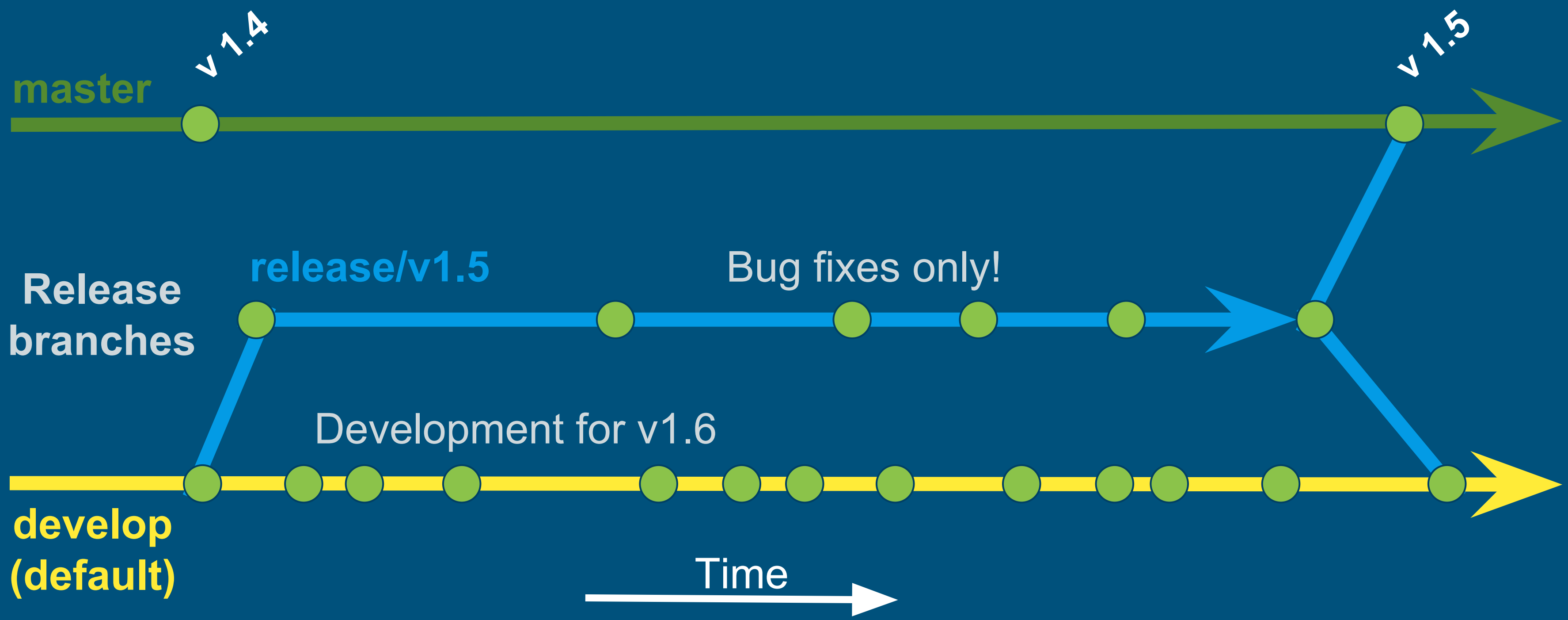
Branch Management With Git-Flow



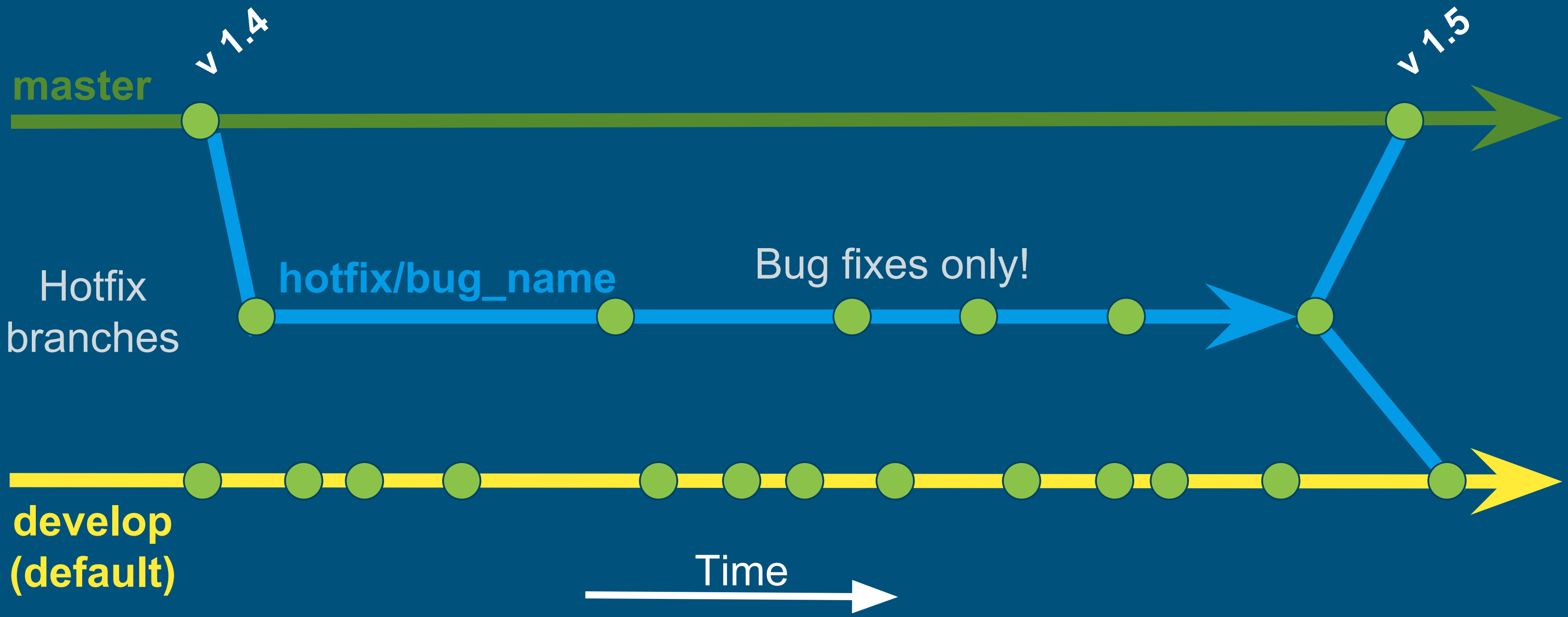
Branch Management With Git-Flow



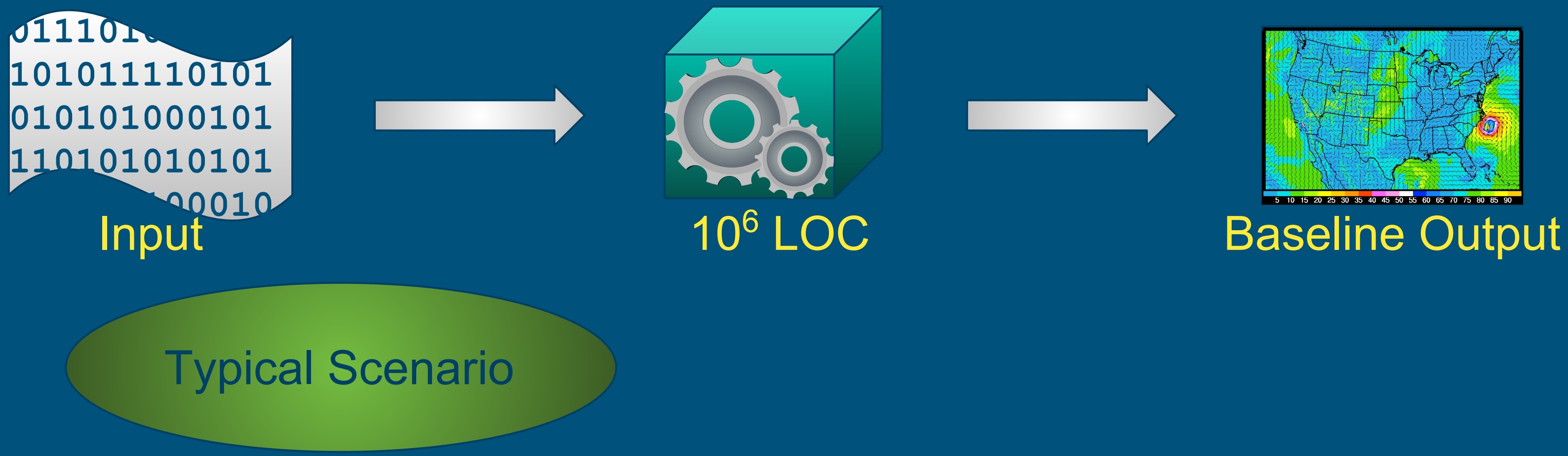
Branch Management With Git-Flow



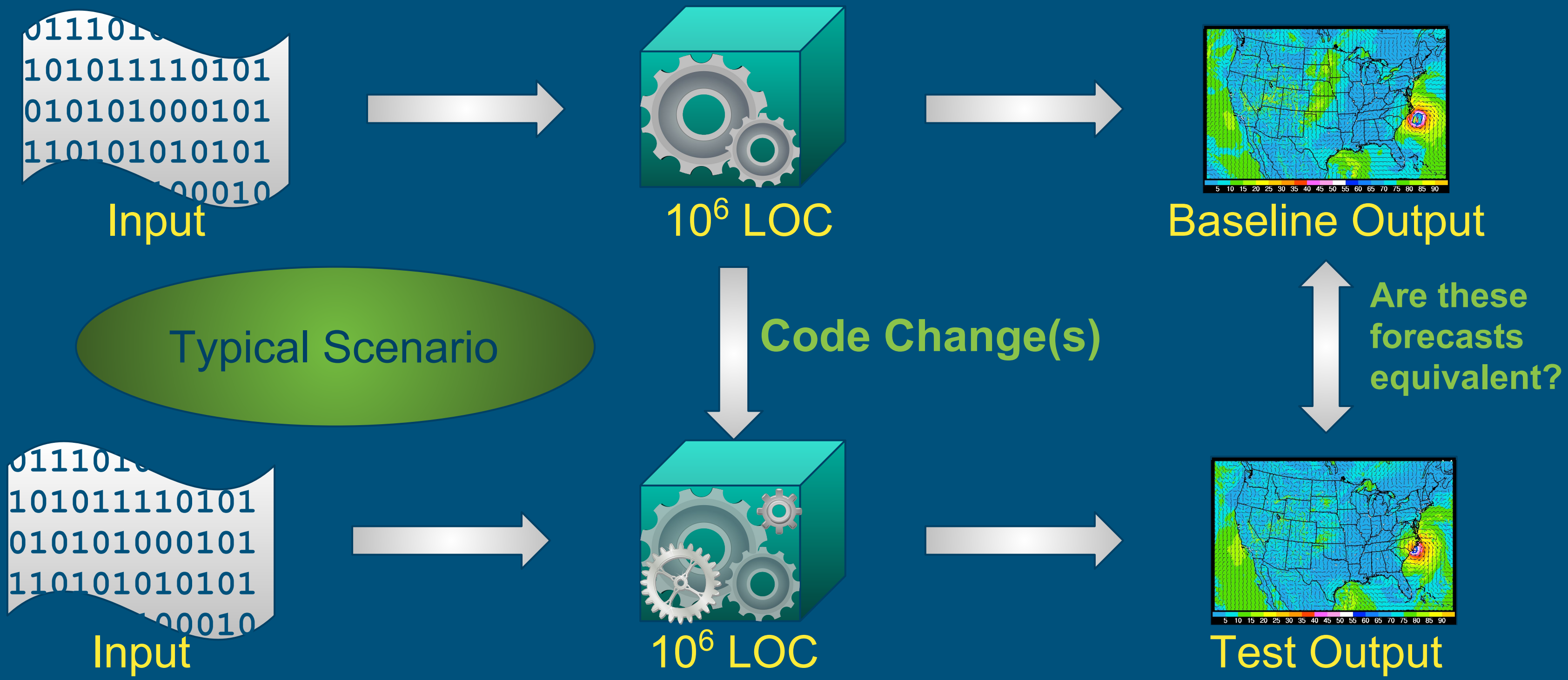
Branch Management With Git-Flow



Test Driven Development



Test Driven Development



Test Driven Development

Several problems with reliance on system level tests

- Focus is on testing the “model” instead of the “software”
- Does not provide error localization when failures are detected
- Trillions of operations performed exacerbate comparison of results
- High levels of test coverage are difficult to achieve
- Often masks serious errors
- Undetected bugs are allowed into the “stable” repository branches

Test Driven Development

A better way....

- Test the science AND the software
  - Theoretical system, computational system, software implementation
- Test multiple quality factors
  - Performance, reliability, correctness, portability
- Test at all granularities
  - Unit tests, integration tests, system tests
- Write new code → Write new tests

Test Driven Development

Rules of engagement

- Automate tests / continuous integration
- Require pull requests for merges
- Require reviews for pull requests
- No pull requests merged unless all tests pass
- Pull requests must supply tests for all new code

```
Test project /scratch4/BMC/gsd-hpcs/Christopher.W.Harrop/Exascale-DA/build_theia_intel
      Start  1: shallow_water_config_arglist
 1/16 Test  #1: shallow_water_config_arglist ...........   Passed    0.01 sec
      Start  2: shallow_water_config_nlfile
 2/16 Test  #2: shallow_water_config_nlfile ............   Passed    0.01 sec
      Start  3: shallow_water_config_nlunit
 3/16 Test  #3: shallow_water_config_nlunit ............   Passed    0.01 sec
      Start  4: shallow_water_model_matlab_regression
 4/16 Test  #4: shallow_water_model_matlab_regression ...   Passed   22.94 sec
      Start  5: shallow_water_model_init_default
 5/16 Test  #5: shallow_water_model_init_default ........   Passed    0.01 sec
      Start  6: shallow_water_model_init_optional
 6/16 Test  #6: shallow_water_model_init_optional .......   Passed    0.01 sec
      Start  7: shallow_water_model_adv_nsteps
 7/16 Test  #7: shallow_water_model_adv_nsteps .........   Passed    0.01 sec
      Start  8: shallow_water_model_regression
 8/16 Test  #8: shallow_water_model_regression .........   Passed    0.02 sec
      Start  9: shallow_water_reader
 9/16 Test  #9: shallow_water_reader ...................   Passed    0.01 sec
      Start 10: shallow_water_writer
10/16 Test #10: shallow_water_writer ...................   Passed    0.02 sec
      Start 11: shallow_water_tl_init_default
11/16 Test #11: shallow_water_tl_init_default ..........   Passed    0.01 sec
      Start 12: shallow_water_tl_init_optional
12/16 Test #12: shallow_water_tl_init_optional .........   Passed    0.01 sec
      Start 13: shallow_water_tl_adv_nsteps
13/16 Test #13: shallow_water_tl_adv_nsteps ............   Passed    0.19 sec
      Start 14: shallow_water_adj_init_default
14/16 Test #14: shallow_water_adj_init_default .........   Passed    0.01 sec
      Start 15: shallow_water_adj_init_optional
15/16 Test #15: shallow_water_adj_init_optional ........   Passed    0.01 sec
      Start 16: shallow_water_adj_adv_nsteps
16/16 Test #16: shallow_water_adj_adv_nsteps ...........   Passed    0.20 sec

100% tests passed, 0 tests failed out of 16

Total Test time (real) =  23.55 sec
[Christopher.W.Harrop@Theia:tfe03 build_theia_intel]$
```

Scientific software design challenges

- Poor software design quality throttles scientific progress
- Requirements are often poorly defined up front
- Requirements driven by scientific discovery process
- Evolving requirements make extensibility and reproducibility difficult
- Maintainability needs to be prioritized in design

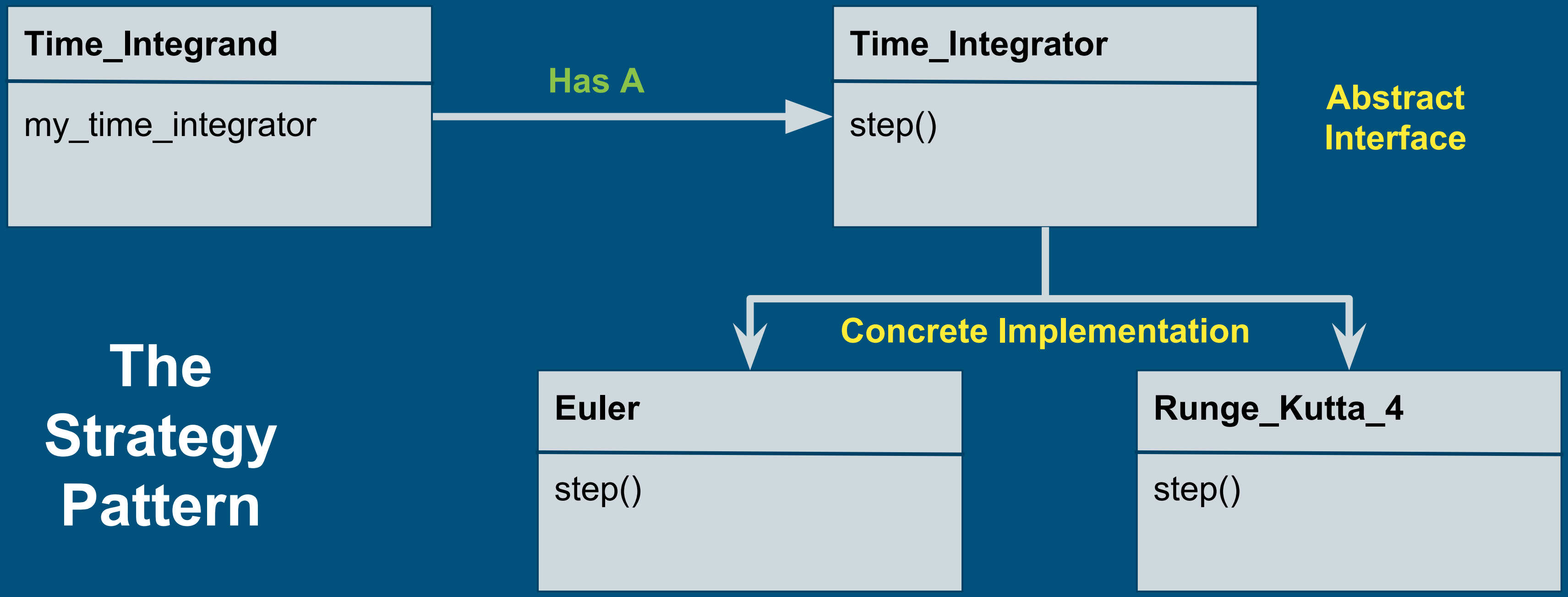
A case for scientific software design patterns

- Reusable code → Reusable designs
- Robust recipes for solutions to common design problems
- Innocculate code against future changes
- Provide lexicon for discussing design properties

A case for scientific software design patterns

- Adoption of classic patterns to scientific software
- Identify new patterns specific to scientific problems
- Build a common repository of robust design elements for the community
    - Requires community collaboration
- Anti-patterns → Repository of how NOT to design is also useful

A case for scientific software design patterns



The Strategy Pattern

Conclusions

- Investment in software quality is required for improvements in science
    - process/design/maintainability
- We can learn from commercial software engineering industry
    - Git-Flow branching model
    - Test-driven development
    - Design patterns
- Automation should be maximized to minimize human error