DA Evolution and Expansion: JEDI
Joint Effort for Data assimilation Integration

Yannick Trémolet

JCSDA

Joint DTC-EMC-JCSDA GSI/EnKF Tutorial – 13 July 2017

Data Assimilation Evolution

Data assimilation applications are expanding:

- Earth-system components
  - Atmosphere, ocean, chemistry, land surface, sea-ice, space weather…
- More observations (exponential growth in numbers)
- New types of observing systems

Data assimilation science is evolving:

- Hybrid methods
- Coupled Earth-system DA (different scales, observations…)
- Adaptation to new computer architectures

Data assimilation will only become more complex and more costly (in computational and human efforts)

The JEDI Project

Increasing complexity is everywhere around us

- Cars, video games, telecommunication systems, understanding of the human brain, ...

It is daunting but it means there are tools and techniques to manage it.

Using appropriate tools, the JEDI project aims at developing a unified DA system for:

- Research and operational use
- All components of the Earth-system
- Efficient use of human and computational resources

Modern Software Engineering

We want a very flexible, reliable, efficient, generic, readable and modular code.

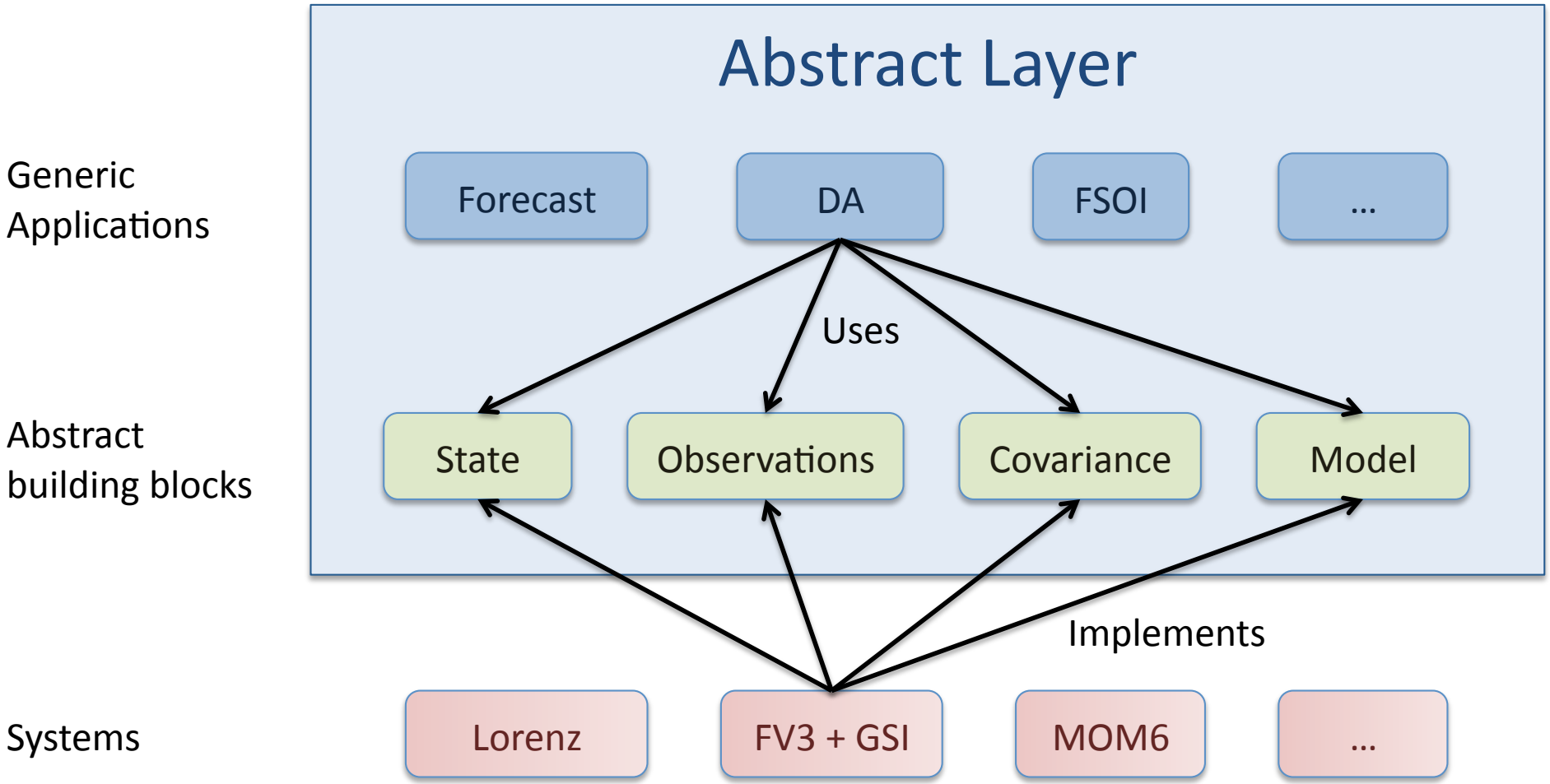
This is not specific to NWP: the software industry has moved to generic and object-oriented programming 20 years ago.

The key idea is **separation of concerns**:

- All aspects exist but scientists focus on one aspect at a time.
- Different concepts are treated in different parts of the code.
- Nobody can know it all!

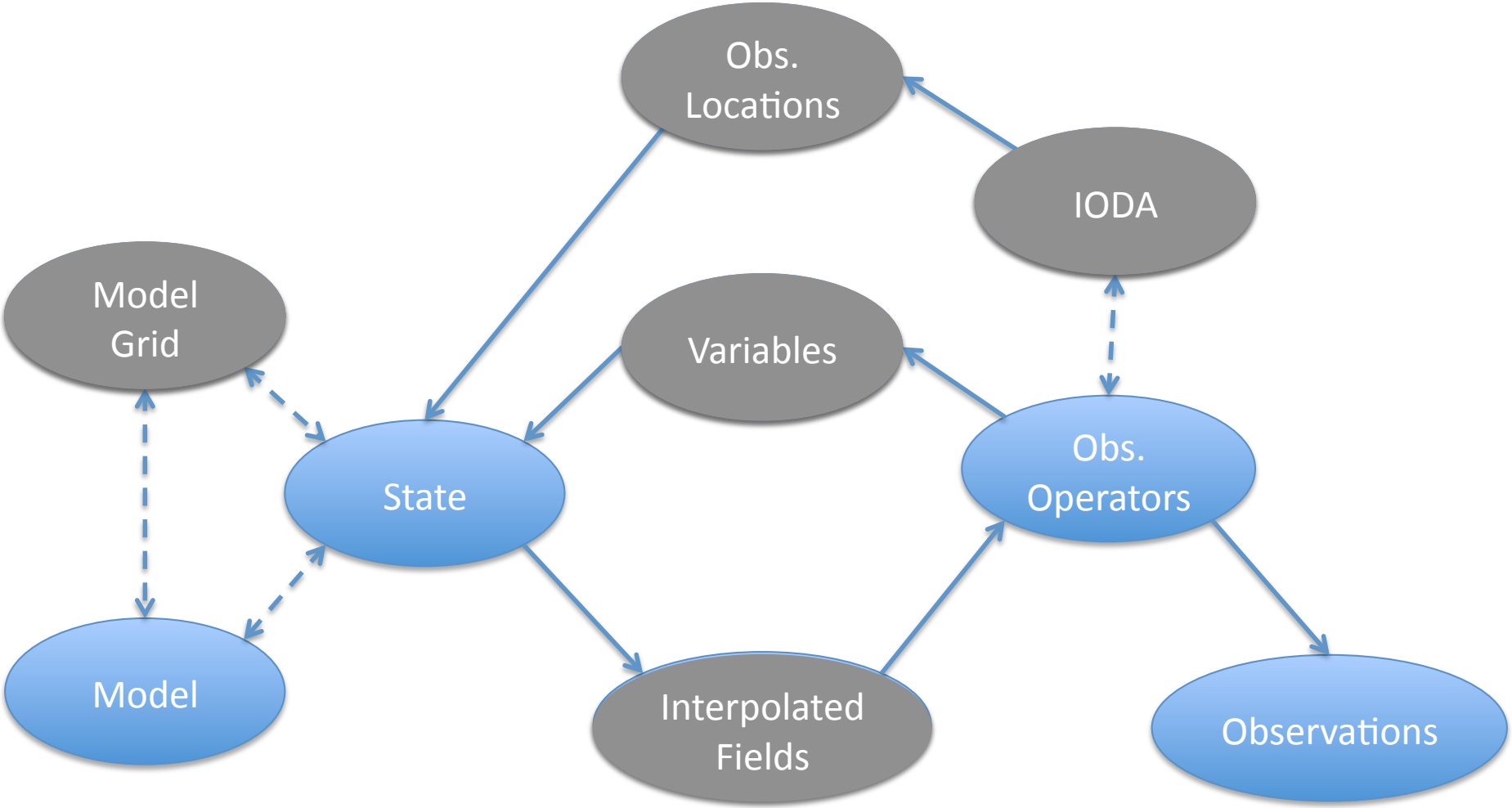
Good software engineering does not solve scientific problems: it is a tool to express and manage computations more efficiently.

JEDI Abstract Design

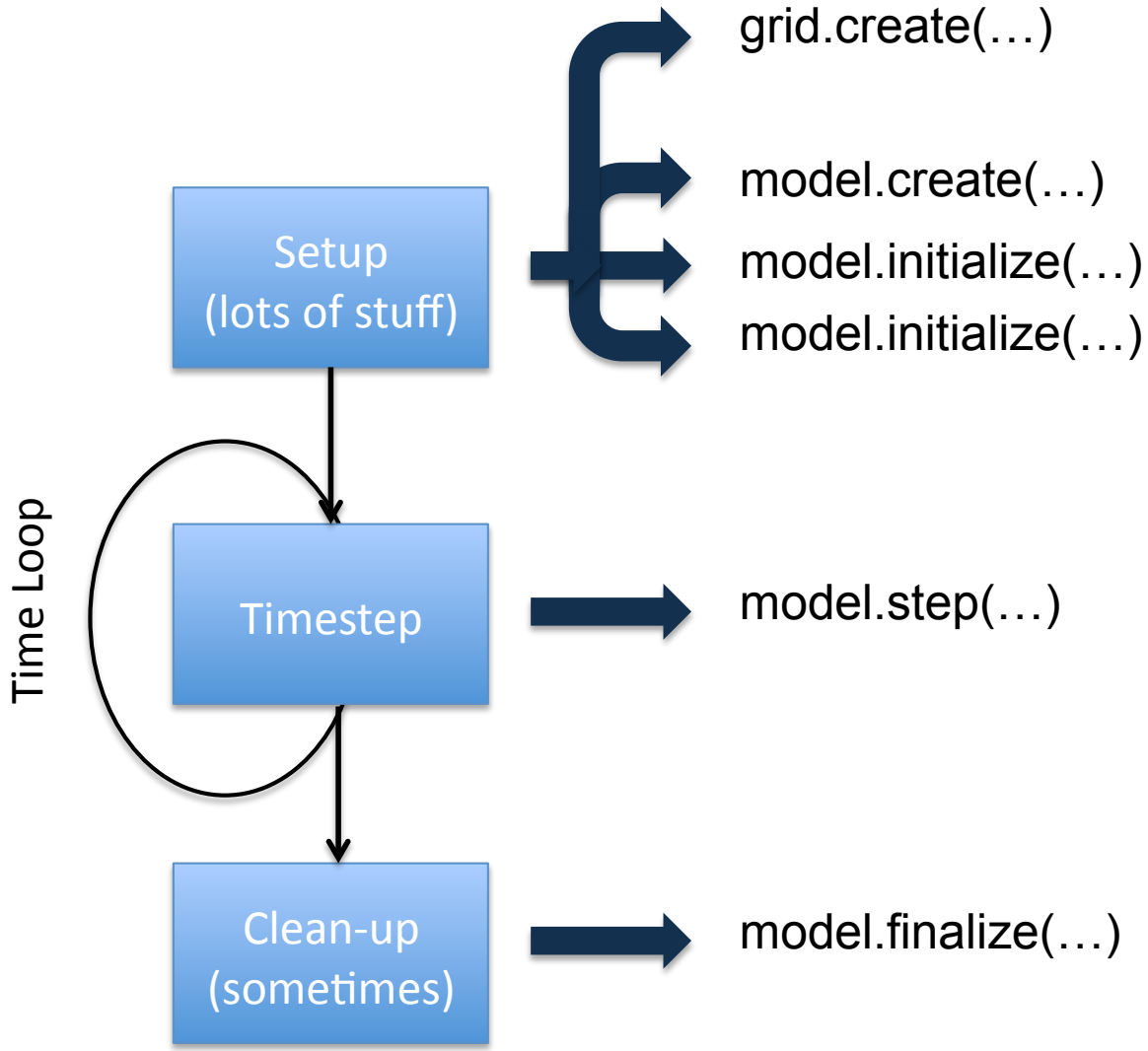


Abstract interfaces are the most important aspect of the design

Design: Unified Forward Operator [y=H(x)]



Design: What about the Model? [x_t=M(x_0)]



Works for any model, including coupled models

Interface to Observations (IODA)

Isolate scientific code from data storage: separation of concerns

Three levels:
- Long term (possibly shared) historic archive
- Storage on disk during an “experiment”
- In memory handling of observations (hardware specific?)

Two environments:
- Plotting, analyzing, verifying on workstation
- DA and other HPC applications (MPI, threads, GPUs…)

Interface for Observation Data Access (IODA) goals:
- Interface definition
- Implementation (can it be unique?)

Design Comments

Interfaces, interfaces, interfaces!

The concrete implementation classes do not know about the high level algorithm currently being run:

- **All** actions are driven by the top layer,
- **All** data, input and output, is passed by arguments.

Interfaces must be general enough to cater for all cases, and detailed enough to be able to perform the required actions.

JEDI stops at the level of the calls to the model time-step and observation operators but the same principle could be applied at any level.

Where is the interface between suites, scripts and executable?

Top-Down and Bottom-Up

Keep the computational parts of the existing code and reuse them in a re-designed flexible structure.

This can be achieved by a top-down and bottom-up approach:
– From the top: develop a new, modern, flexible structure
– From the bottom: create self-contained units of code by refactoring legacy code
– Put the two together

Interfaces are the key.

From a Fortran point of view, this implies:
– Control via interfaces (derived types passed by arguments)
– No global variables

Code performance

Experience shows that refactoring generally improves code performance

Moving the entire DA algorithm into one executable will reduce I/O and improve efficiency (even more in the future)

New structure will make scalability investigations possible

Code efficiency is an important aspect of the project

Performance and readability are not incompatible

Modern Software Project Management

Collaborative environment

- Access to up-to-date source code (Bitbucket, github)
- Exchange of information (Confluence, JIRA)
  - About the code
  - About the project

Continuous Integration

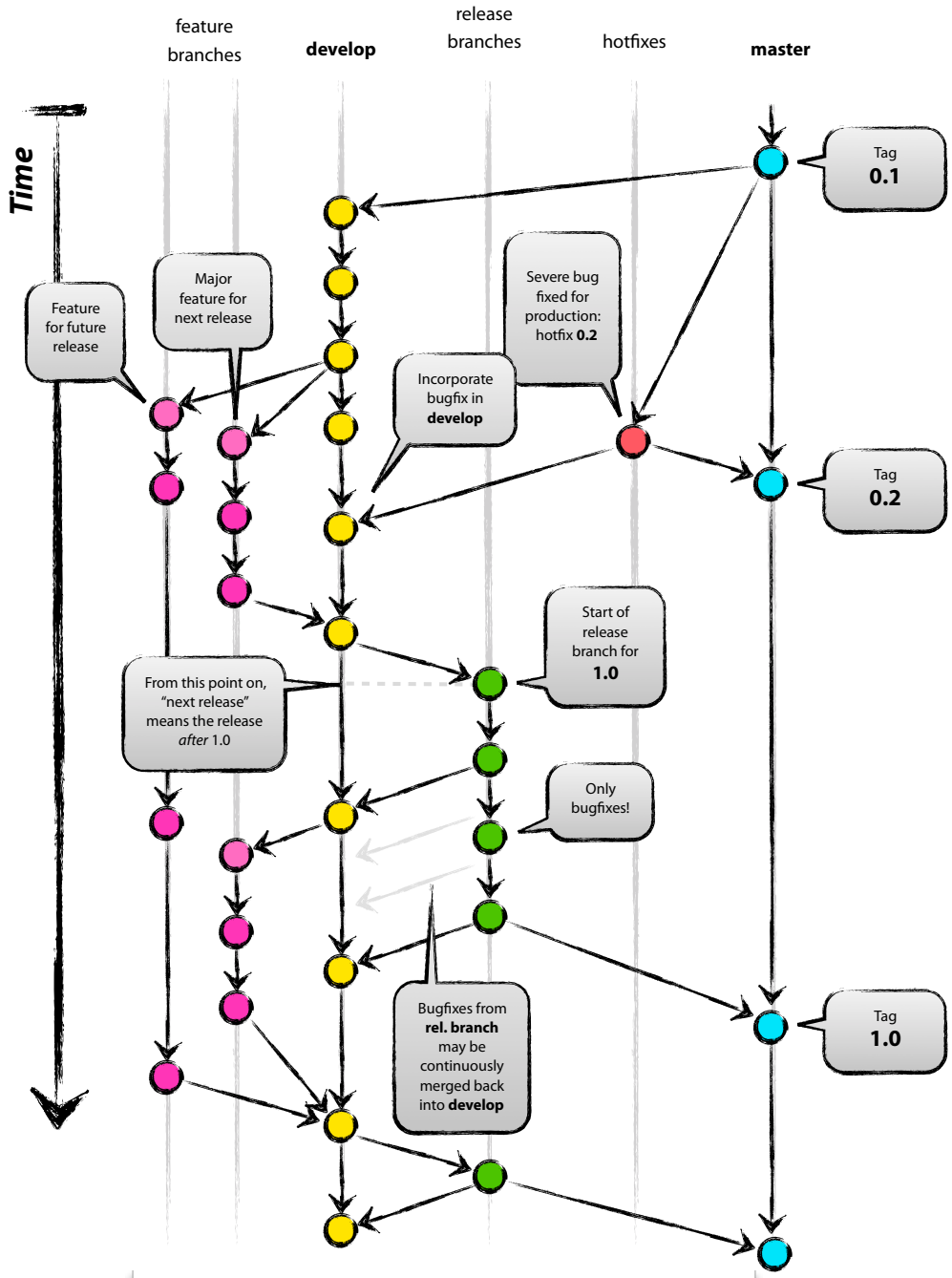
- Automated testing
- Enforce software quality (correctness, coding norms, efficiency)
- Preliminary step for code reviews

Project methodology inspired by Agile/SCRUM

Collaborating: Branching and Merging

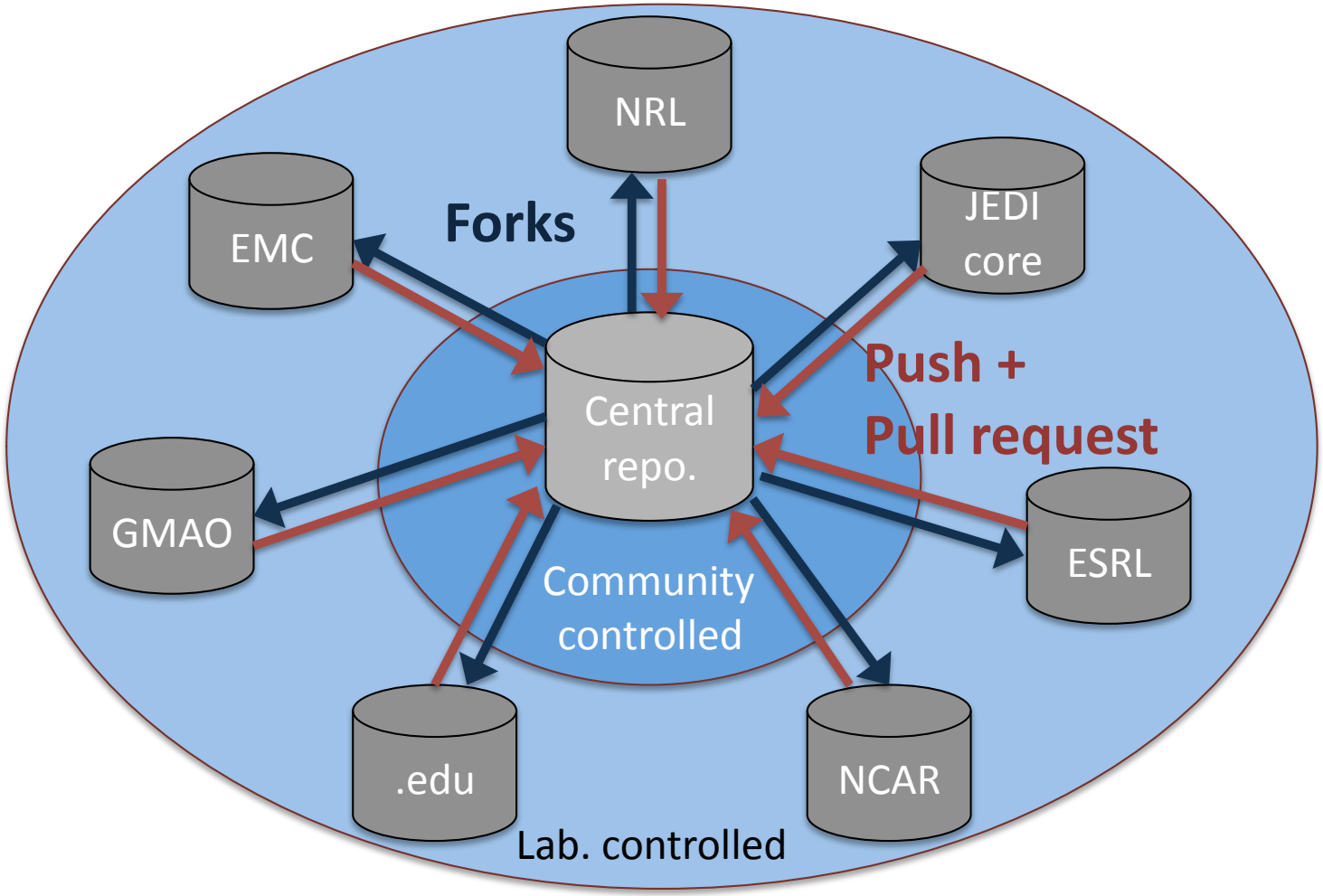
Git-flow model

Merges require code reviews



Author: Vincent Driessen
Original blog post: http://nvie.com/posts/a-succesful-git-branching-model
License: Creative Commons BY-SA

Collaborating: Repositories

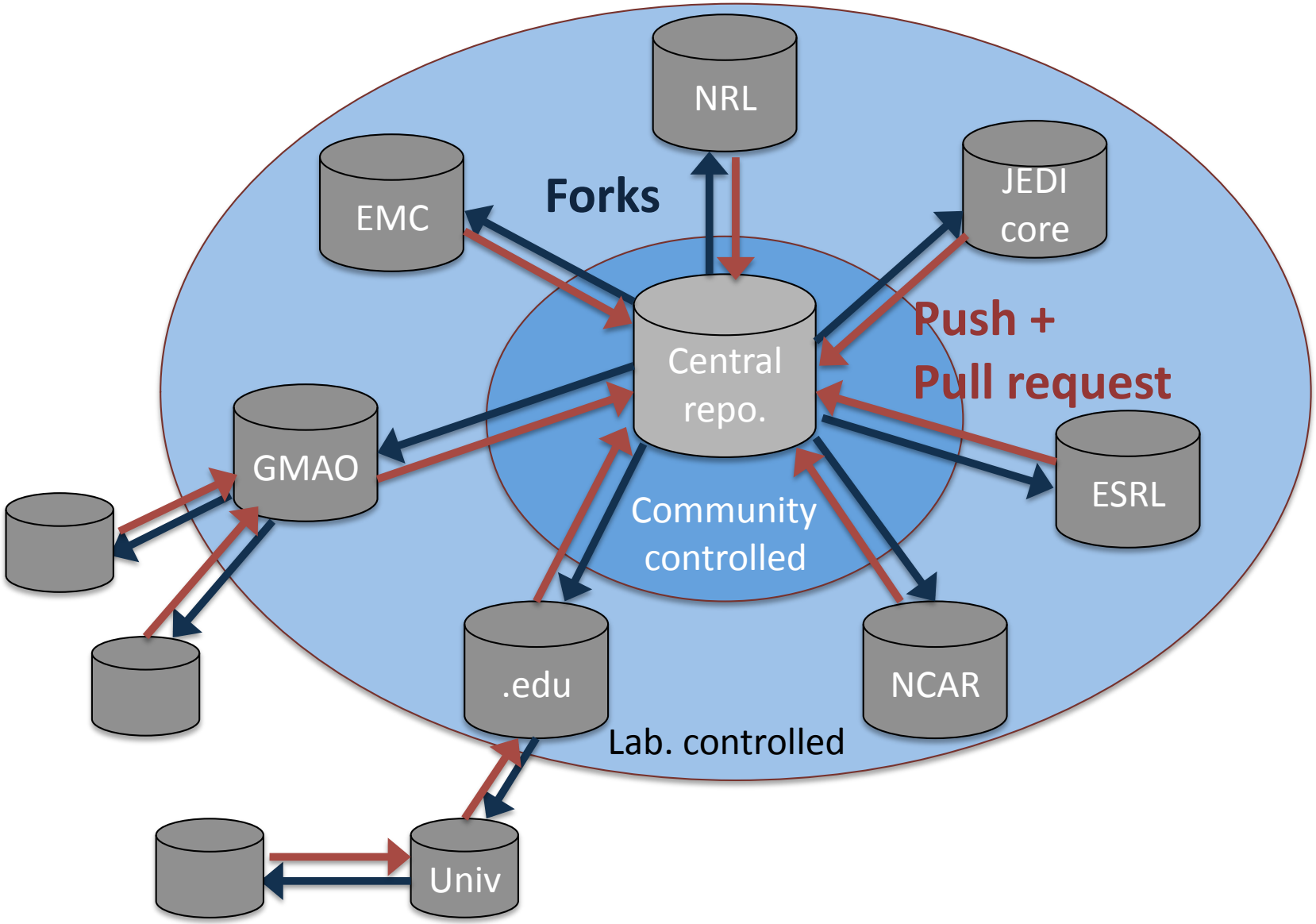


Permission to fork repository are very easy to obtain

Contributing code is very controlled:

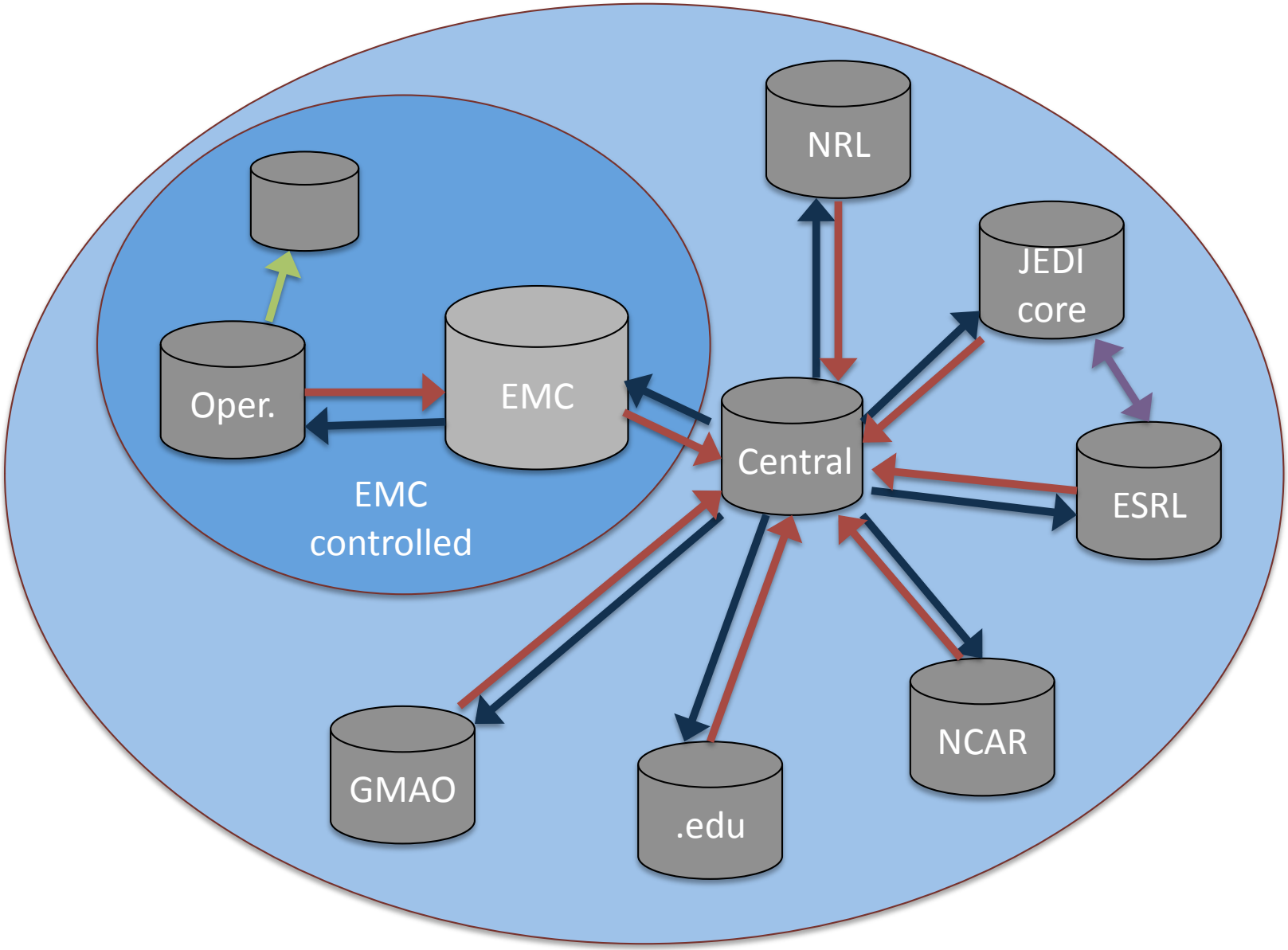
- Pushing a branch requires write permission on central repository
- Pull request triggers code review and approval for merging to higher level branch

Collaborating: Repositories



The process is recursive with similar controls at all levels

Collaborations and Operations



Full control on operational codes lie within EMC/NCO, or NRL, or …

Governance and code reviews

Governance is about management keeping in control and deciding what features should be in the system

Code reviews are about quality of the code

Two different levels of control

- Good code can stay outside of central repository (stability of interfaces is important)
- A desired feature that does not satisfy quality requirements cannot be accepted as is

Testing is a pre-requirement for code reviewing

Different people and different pace: **Separation of concerns**…

Project Roadmap

We are at **Stage 0**:

General plan, detailed plan for stage 1

Stage 1:

UFO (interpolation from model grid, observation operators), IODA

"*Observer*" in GSI jargon

Stage 2:

Covariance matrices, IODA, linearized UFO, 3D solvers, bias correction

Stage 3:

Optimized UFO, UFO from model, 4D solvers

Stage 4:

Multi-scale DA, coupled DA and more!

This plan will evolve

Final comments

All the technologies are proven
– Nobody in the software industry would try to develop software with using objects or better (functional programming…)

"Our problem is too complex..."
– Object Oriented Programming was invented to manage complexity in code more efficiently!
– What about the software you (or your kids) use everyday?

The difficulties are not technical, they are human
– Most scientists are not trained in software engineering
– Can we afford to be so inefficient in the future?

JEDI is for **operational forecasting** and **scientific exploration!**