better scientific software

Refactoring Scientific Software

Presented by

PESO https://pesoproject.org

RAPIDS https://rapids.lbl.gov

Members of

CASS

Consortium for the Advancement of Scientific Software
https://cass.community

With prior support from

ECP EXASCALE COMPUTING PROJECT

IDEAS productivity

Anshu Dubey (she/her)
Argonne National Laboratory

Software Sustainability track @ Argonne Training Program on Extreme-Scale Computing summer school

Contributors: Anshu Dubey (ANL), Jared O'Neal

License, Citation and Acknowledgements

License and Citation

- **The requested citation the overall tutorial is:** Anshu Dubey, David E. Bernholdt, and Todd Gamblin, Software Sustainability track, in Argonne Training Program on Extreme-Scale Computing, St. Charles, Illinois, 2025. DOI: 10.6084/m9.figshare.29816981.
- Individual modules may be cited as *Speaker, Module Title*, in *Tutorial Title*, …

Acknowledgements

- This work was supported by the U.S. Department of Energy Office of Science, Office of Advanced Scientific Computing Research (ASCR), and by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration.
- This work was supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, Next-Generation Scientific Software Technologies (NGSST) program.
- This work was performed in part at the Argonne National Laboratory, which is managed by UChicago Argonne, LLC for the U.S. Department of Energy under Contract No. DE-AC02-06CH11357.
- This work was performed in part at the Lawrence Livermore National Laboratory, which is managed by Lawrence Livermore National Security, LLC for the U.S. Department of Energy under Contract No. DE-AC52-07NA27344.
- This work was performed in part at the Los Alamos National Laboratory, which is managed by Triad National Security, LLC for the U.S. Department of Energy under Contract No.89233218CNA000001
- This work was performed in part at the Oak Ridge National Laboratory, which is managed by UT-Battelle, LLC for the U.S. Department of Energy under Contract No. DE-AC05-00OR22725.
- This work was performed in part at Sandia National Laboratories. Sandia National Laboratories is a multi-mission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC., a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

What is Refactoring

Definition: Refactoring is a disciplined technique for restructuring an existing body of code, altering its internal structure without changing its external behavior.

- Different from development
  - You have a working code
  - You know and understand the behavior
  - You have a baseline that you can use for comparison

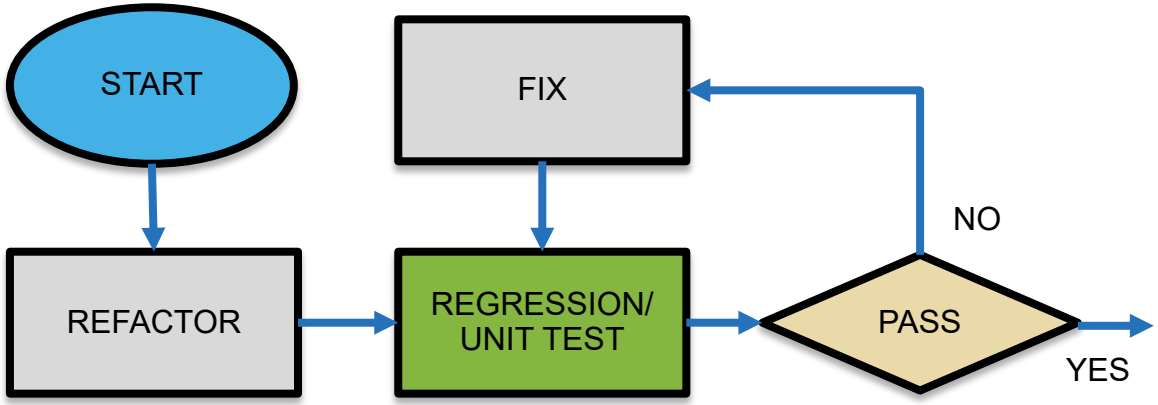
What is Refactoring

Definition: Refactoring is a disciplined technique for restructuring an existing body of code, altering its internal structure without changing its external behavior.

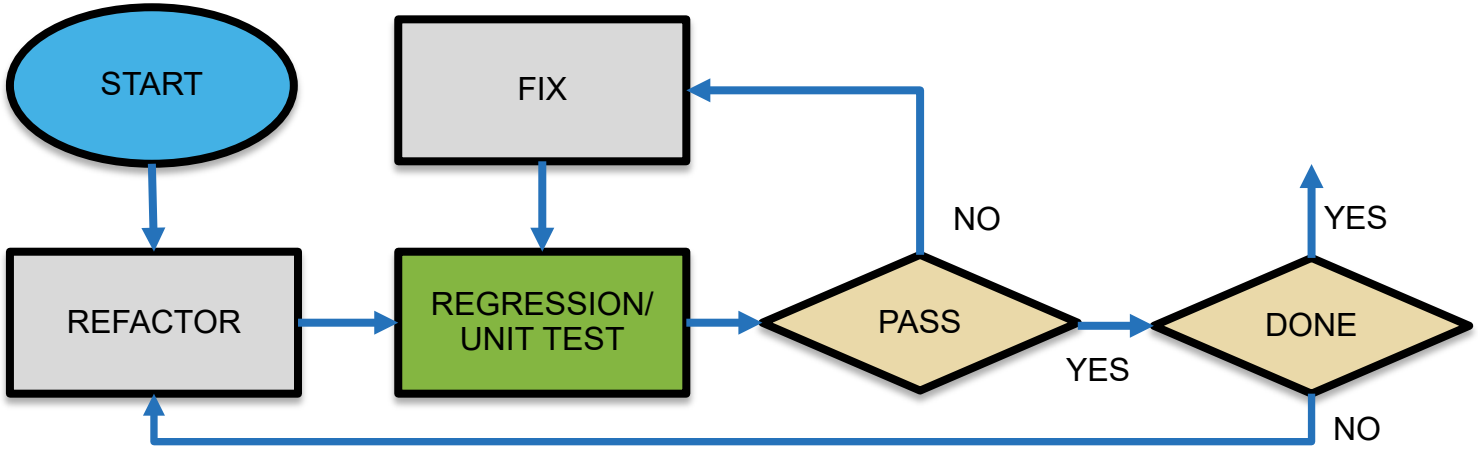
- Different from development
  - You have a working code
  - You know and understand the behavior
  - You have a baseline that you can use for comparison

- General motivations
  - Modularity enhancement
    - Improve sustainability
  - Release to outside users
    - Easier to use and understand
  - Port to new platforms
    - Performance portability
  - Expand capabilities
    - Structural flexibility

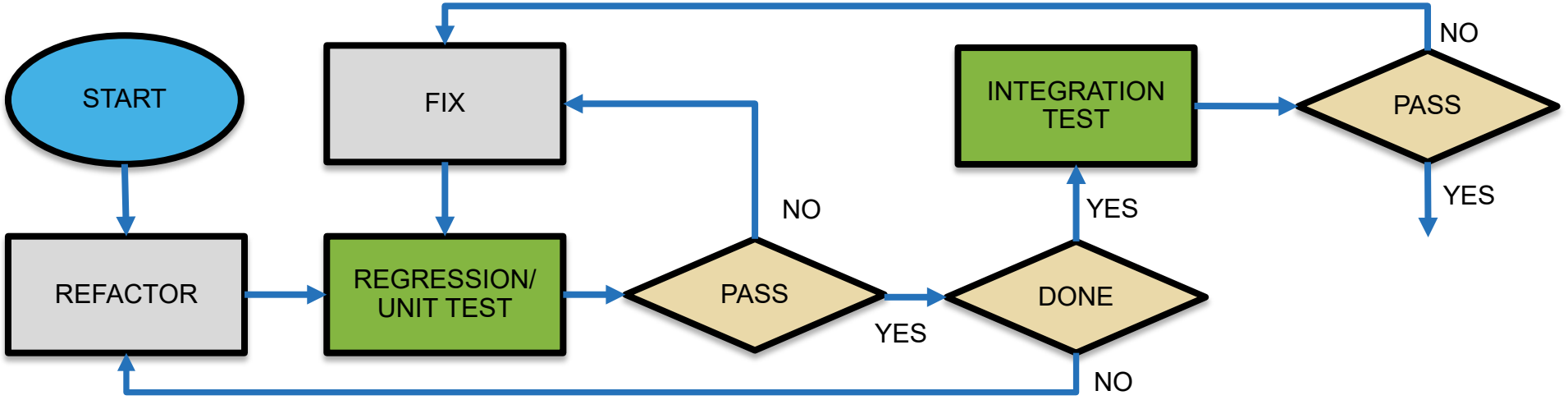
An Example Workflow



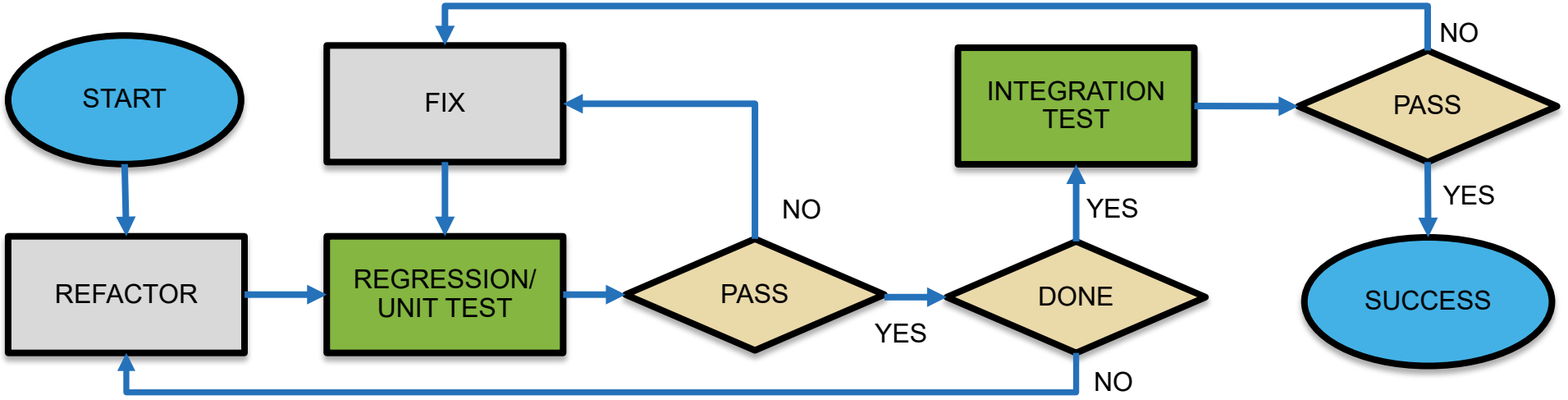
An Example Workflow



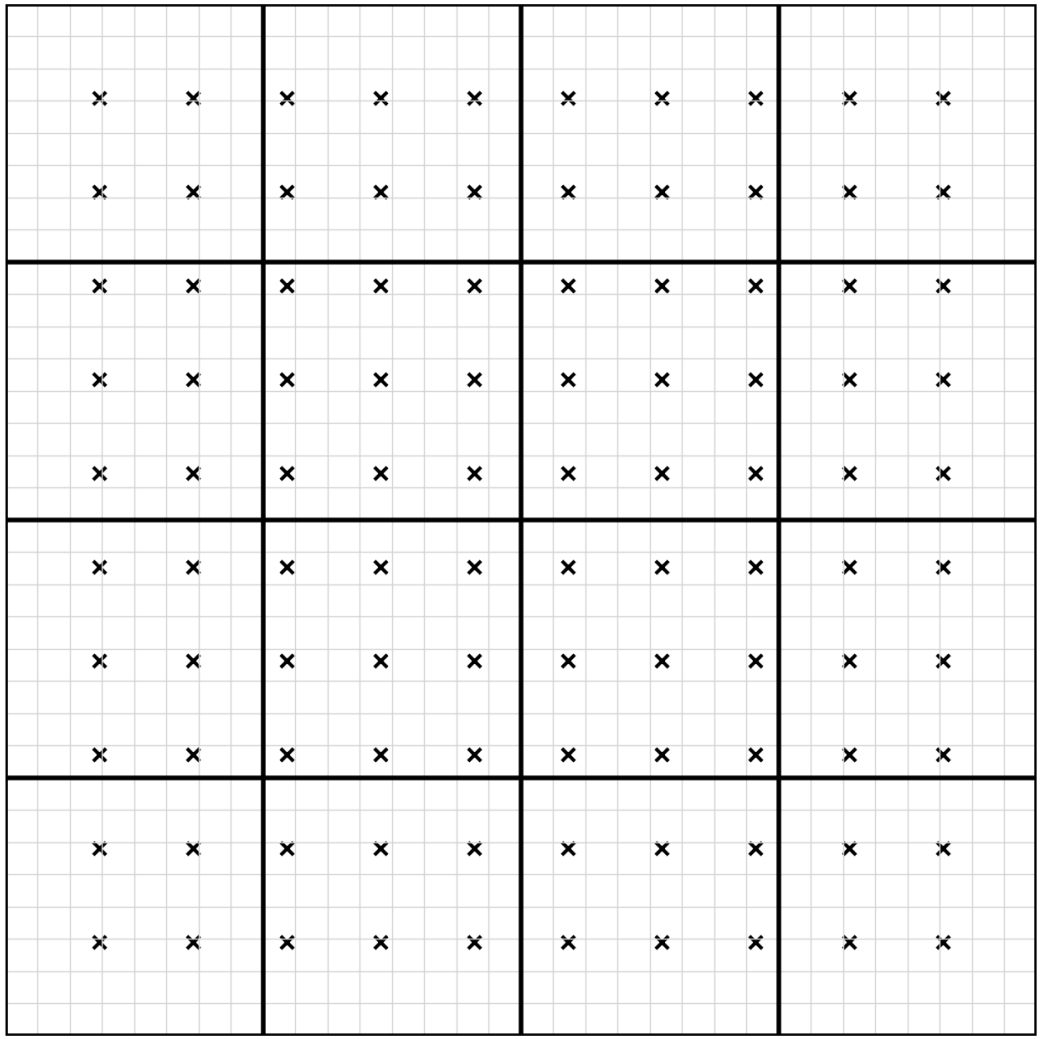
An Example Workflow



An Example Workflow



Look at the Running Example



We created a code that moves particles in a prescribed fashion and only handles periodic boundaries

- If we want to use other methods, we need to break that out into a function
- If we want to also handle outflow boundary condition, we need to allow for particles getting lost

Considerations for Refactoring

- Know why you are refactoring
  - Is it necessary?
  - Where should the code be after refactoring.
- Here we have two reasons
  - One is enhancing flexibility
  - The other is adding capability
- Know your cost estimates
- Know your bounds
  - on acceptable behavior change
  - error bounds
    - bitwise reproduction of results unlikely after transition
- Verification
  - Check for coverage provided by existing tests
  - Develop new tests where there are gaps
  - Here we need two new tests.
    - One to ensure that behavior does not change when we put the method for moving particles into a new function
    - Another one to ensure that boundary conditions are applied correctly
  - Make sure tests exist at different granularities
    - There should be demanding integration and system level tests

Considerations for Refactoring

- Know why you are refactoring
  - Is it necessary?
  - Where should the code be after refactoring.
- Here we have two reasons
  - One is enhancing flexibility
  - The other is adding capability
- Know your cost estimates
- Know your bounds
  - on acceptable behavior change
  - error bounds
    - bitwise reproduction of results unlikely after transition

- Verification
  - Check for coverage provided by existing tests
  - Develop new tests where there are gaps
  - Here we need two new tests.
    - One to ensure that behavior does not change when we put the method for moving particles into a new function
    - Another one to ensure that boundary conditions are applied correctly
  - Make sure tests exist at different granularities
    - There should be demanding integration and system level tests

Incorporate verification overheads into refactoring cost estimates

Considerations for Refactoring

- Know why you are refactoring
  - Is it necessary?
  - Where should the code be after refactoring.
- Here we have two reasons
  - One is enhancing flexibility
  - The other is adding capability
- Know your cost estimates
- Know your bounds
  - on acceptable behavior change
  - error bounds
    - bitwise reproduction of results unlikely after transition

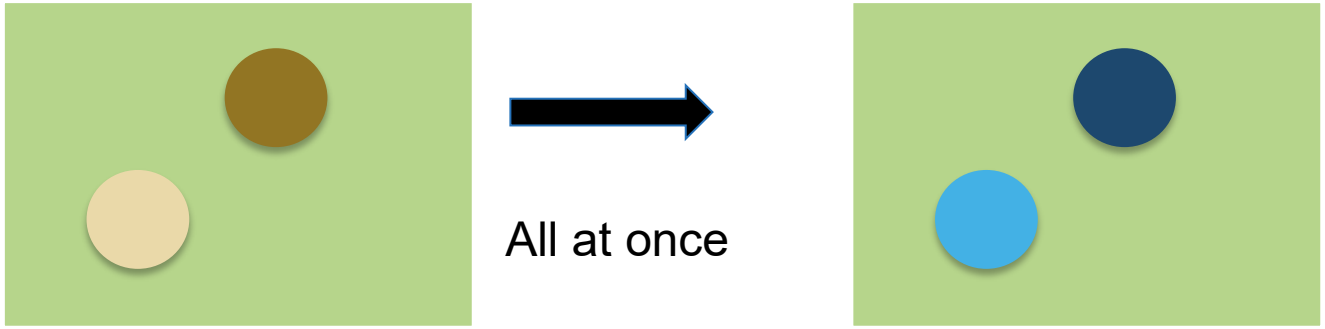
We go back to AI and look at the next set of prompts
https://tinyurl.com/yfxtf89t

- Verification
  - Check for coverage provided by existing tests
  - Develop new tests where there are gaps
  - Here we need two new tests.
    - One to ensure that behavior does not change when we put the method for moving particles into a new function
    - Another one to ensure that boundary conditions are applied correctly
  - Make sure tests exist at different granularities
    - There should be demanding integration and system level tests

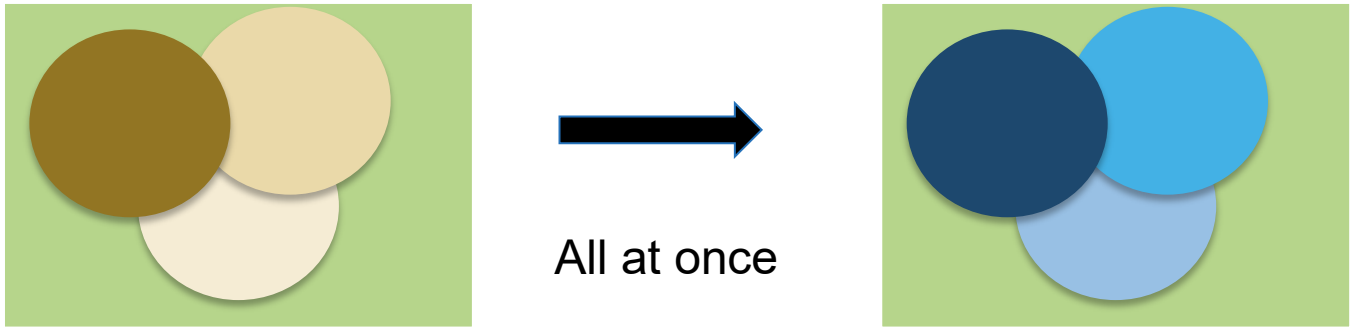
Incorporate verification overheads into refactoring cost estimates

Map from Here to There: On ramp plan

Proportionate to the scope

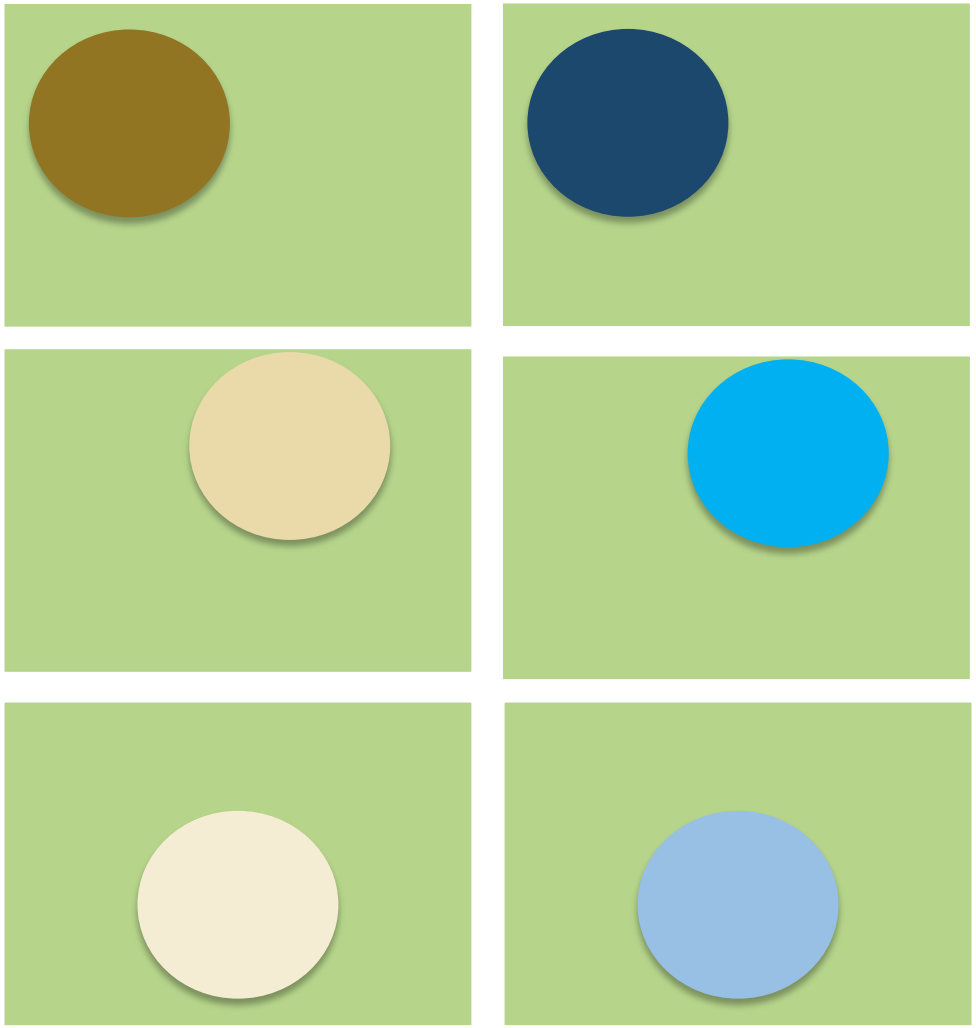


Scattered independent changes - May be OK



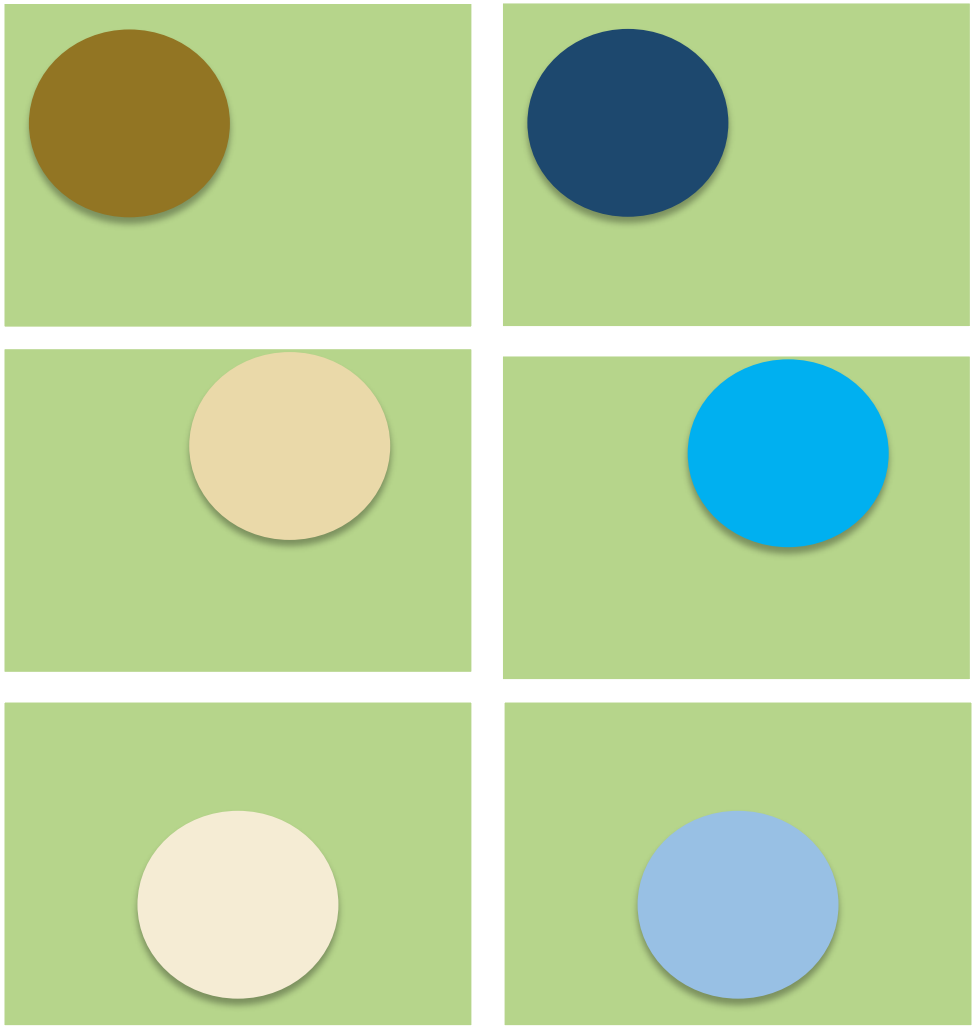
Invasive large-scale change in the code - Bad idea

Map from Here to There: On ramp plan1

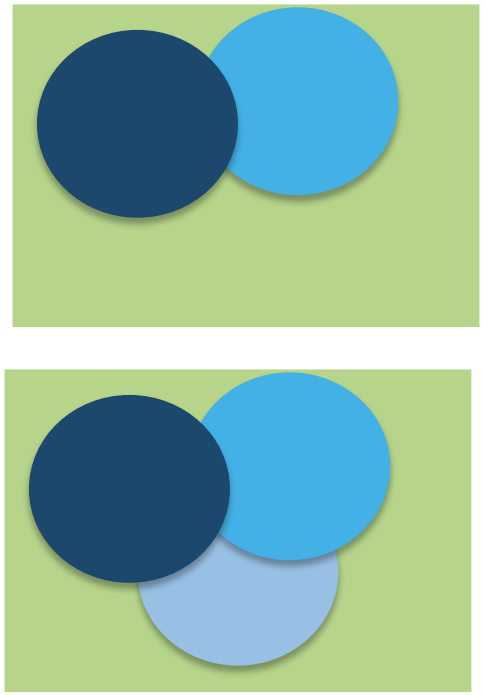


- Turn off all modules except for the one being refactored.
- Have a way of testing in intermediate stages
- Do this for all modules that need refactoring independently

Map from Here to There: On ramp plan1

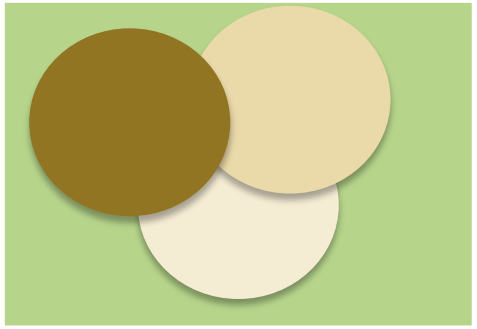


- Turn off all modules except for the one being refactored.
- Have a way of testing in intermediate stages
- Do this for all modules that need refactoring independently



- One by one turn on more than one refactored module

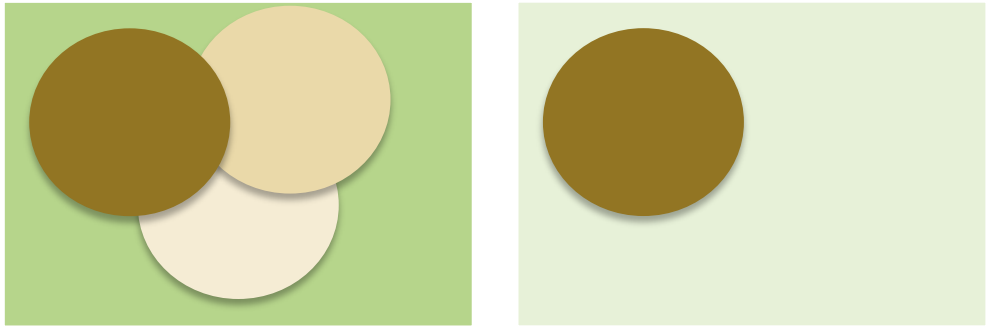
Map from Here to There: On ramp plan2



Map from Here to There: On ramp plan2

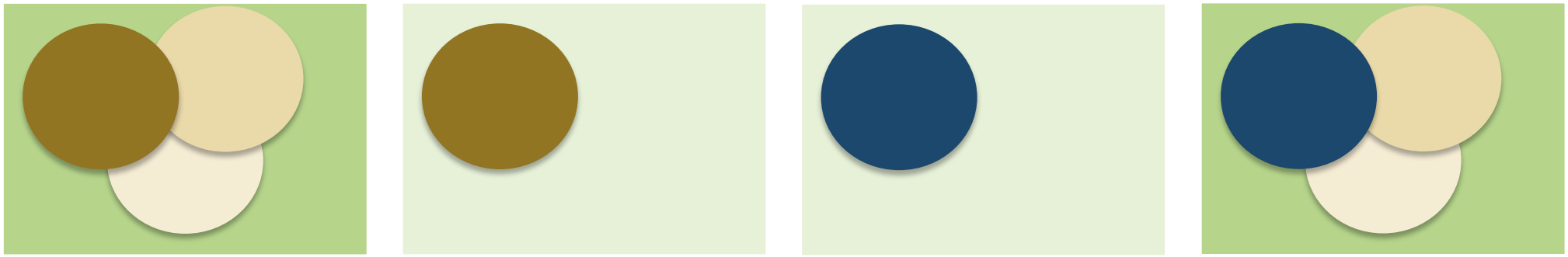
- Build a separate environment for testing refactored module

Map from Here to There: On ramp plan2



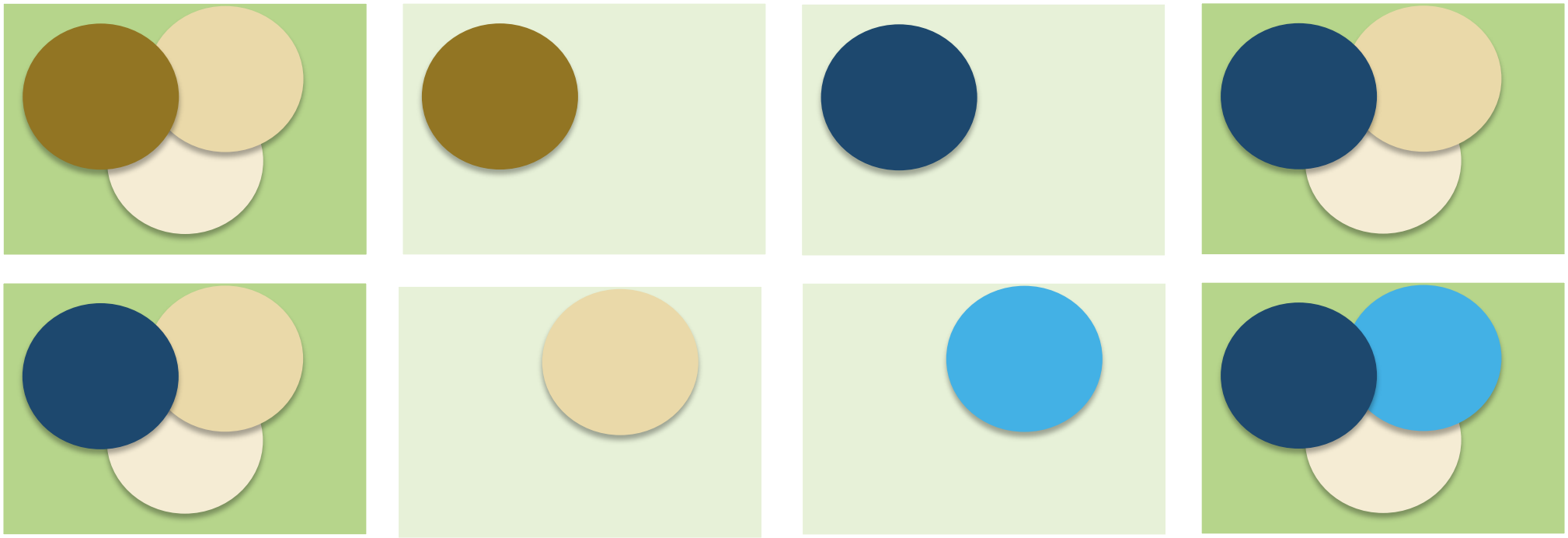
- Build a separate environment for testing refactored module
- Copy over the module in this isolated environment

Map from Here to There: On ramp plan2



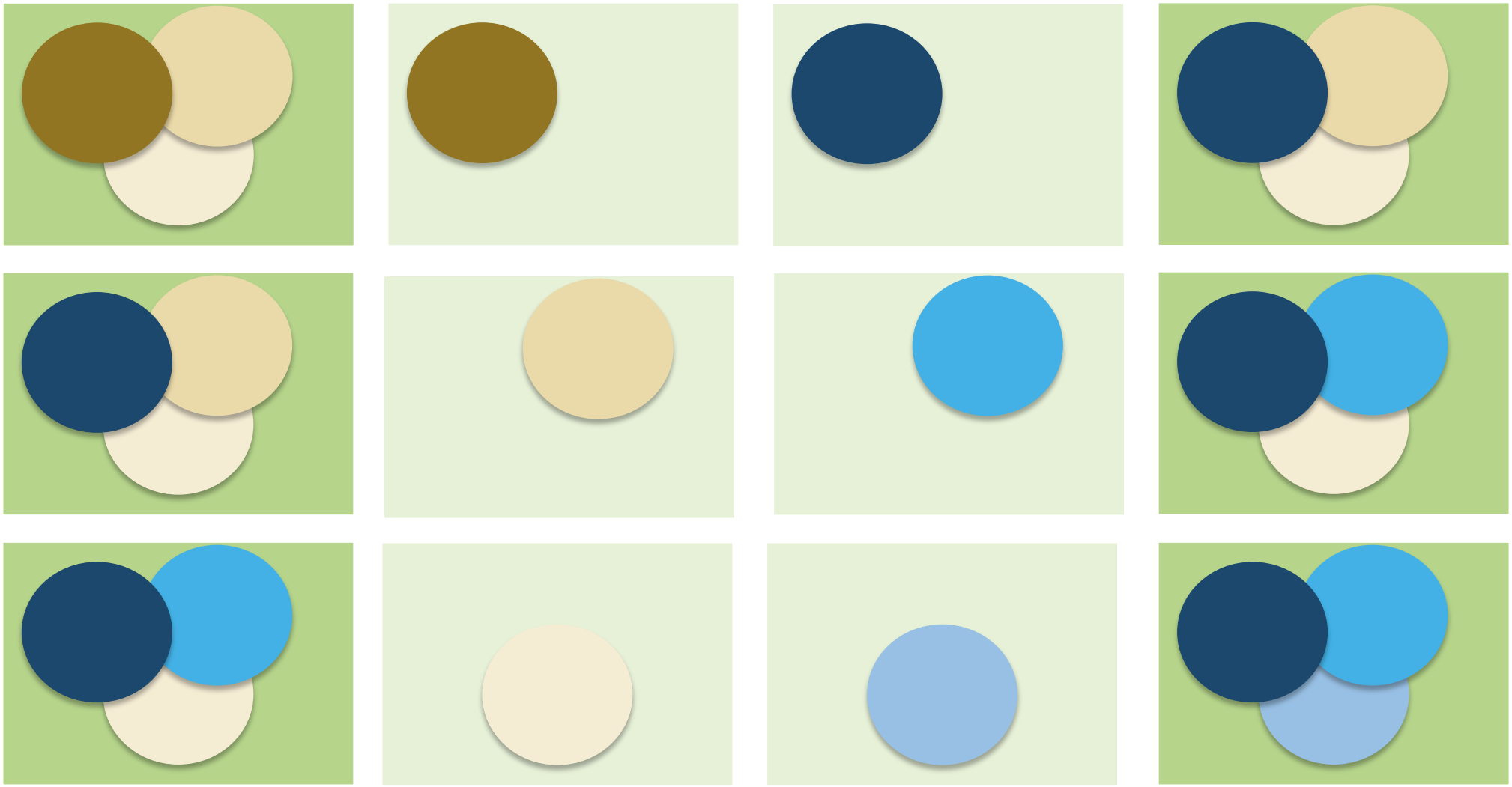
- Build a separate environment for testing refactored module
- Copy over the module in this isolated environment
- Put back refactored module

Map from Here to There: On ramp plan2



- Build a separate environment for testing refactored module
- Copy over the module in this isolated environment
- Put back refactored module

Map from Here to There: On ramp plan2



- Build a separate environment for testing refactored module
- Copy over the module in this isolated environment
- Put back refactored module

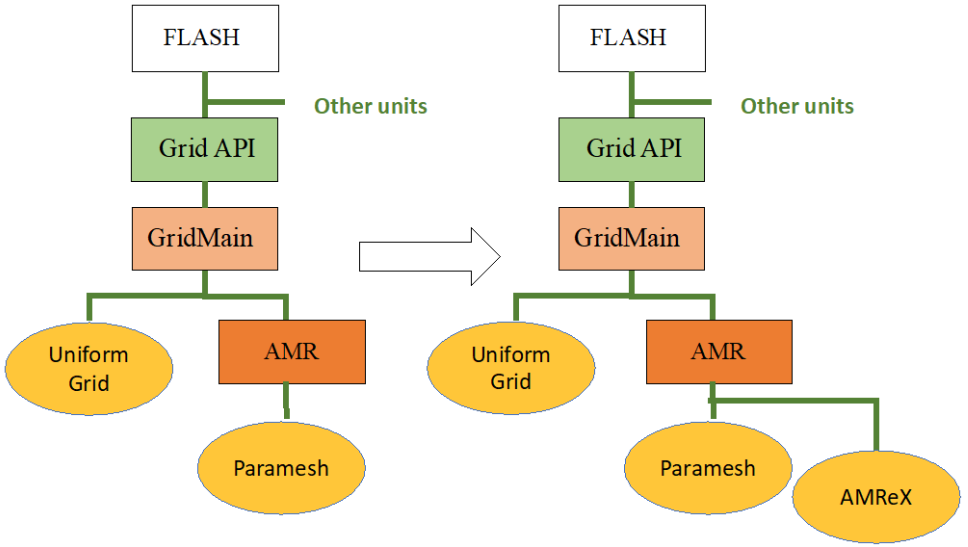
A Real-World Example: FLASH to Flash-X

Refactoring to supporting a different AMR library

Goal: Replace Paramesh with AMReX

Plan: Getting there from here

- On ramping
- Design
- Intermediate steps
- Realizing the goal



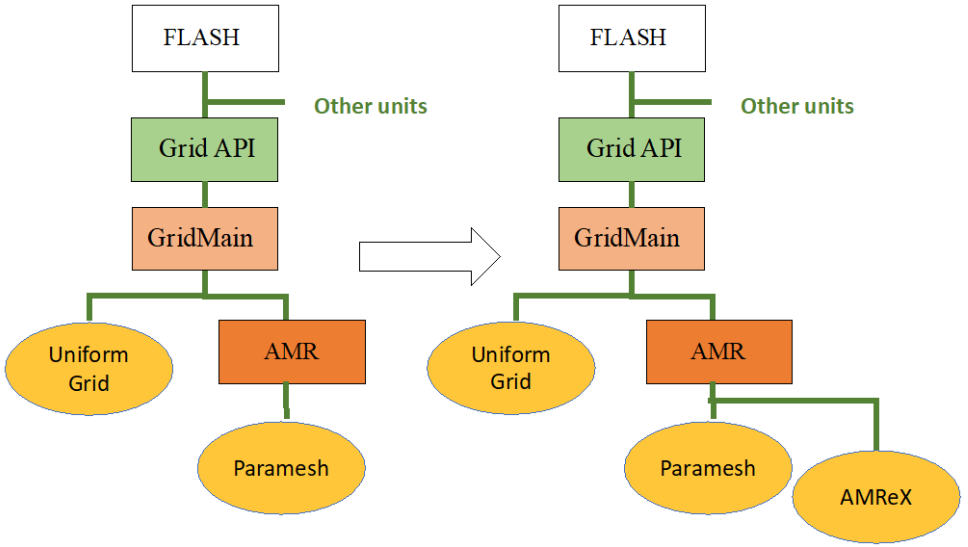
A Real-World Example: FLASH to Flash-X

Refactoring to supporting a different AMR library

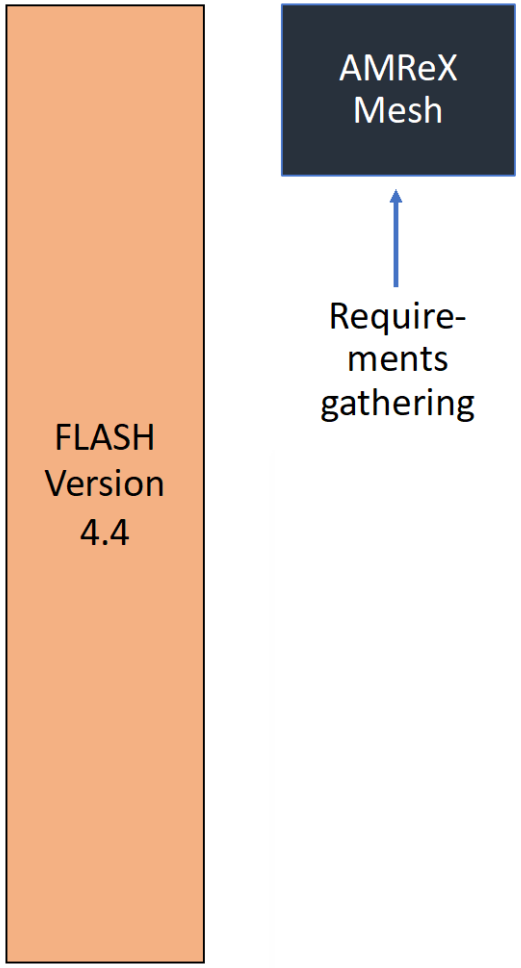
Goal: Replace Paramesh with AMReX

Plan: Getting there from here
- On ramping
- Design
- Intermediate steps
- Realizing the goal

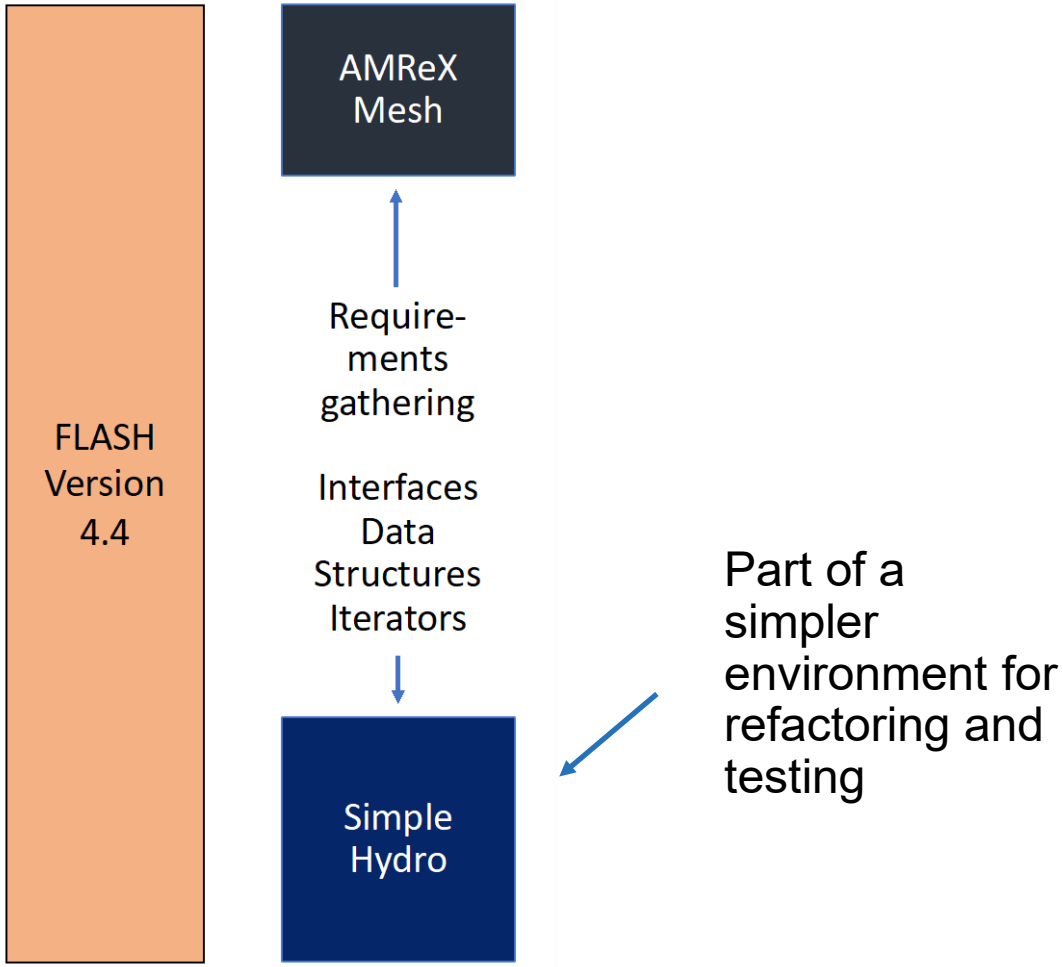
- Cost estimation
  - Expected developer time
  - Extent of disruption in production schedules
- Get a buy-in from the stakeholders
  - That includes the users
  - For both development time and disruption



Steps in the Flash-X Refactoring : a mix of strategies

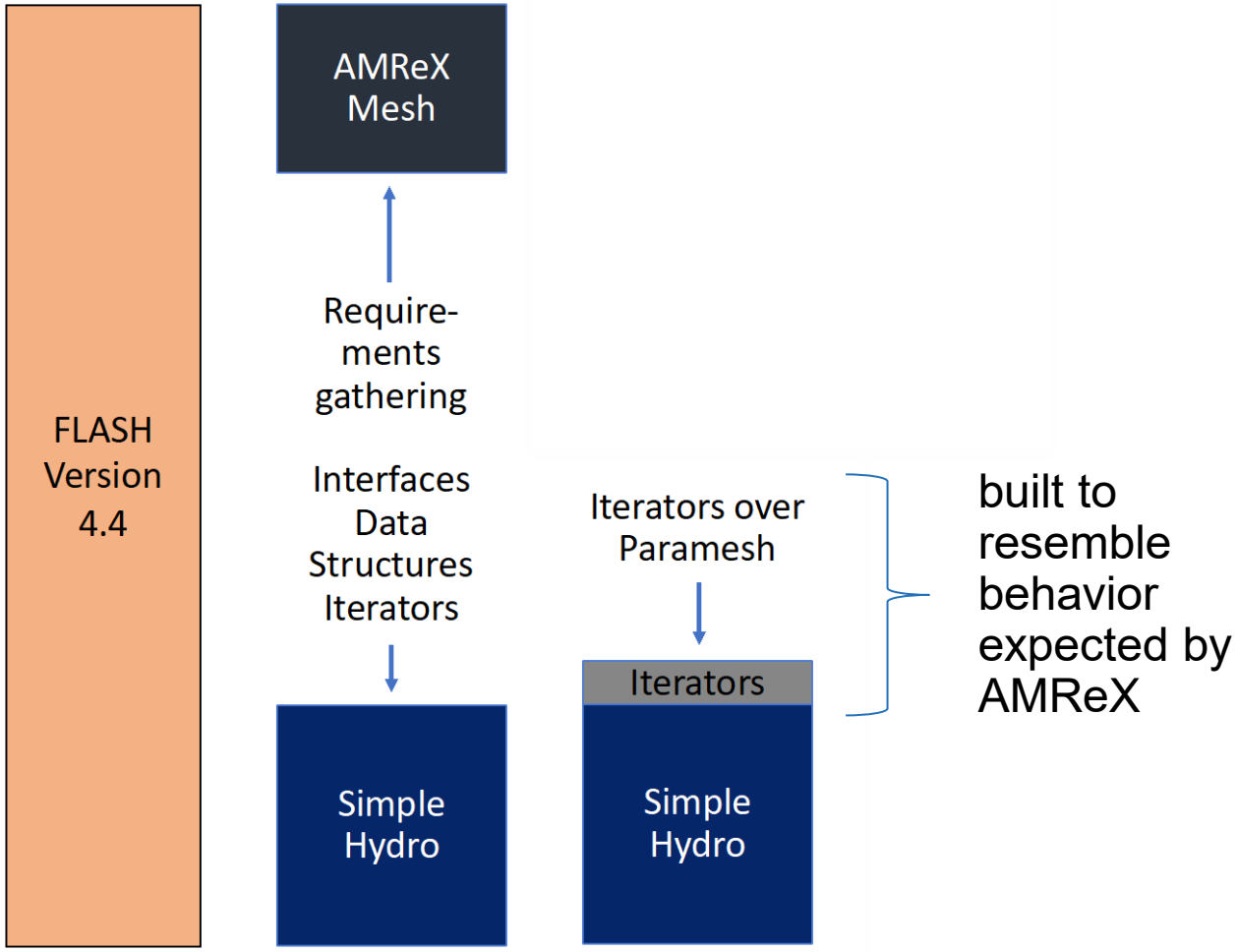


Steps in the Flash-X Refactoring : a mix of strategies

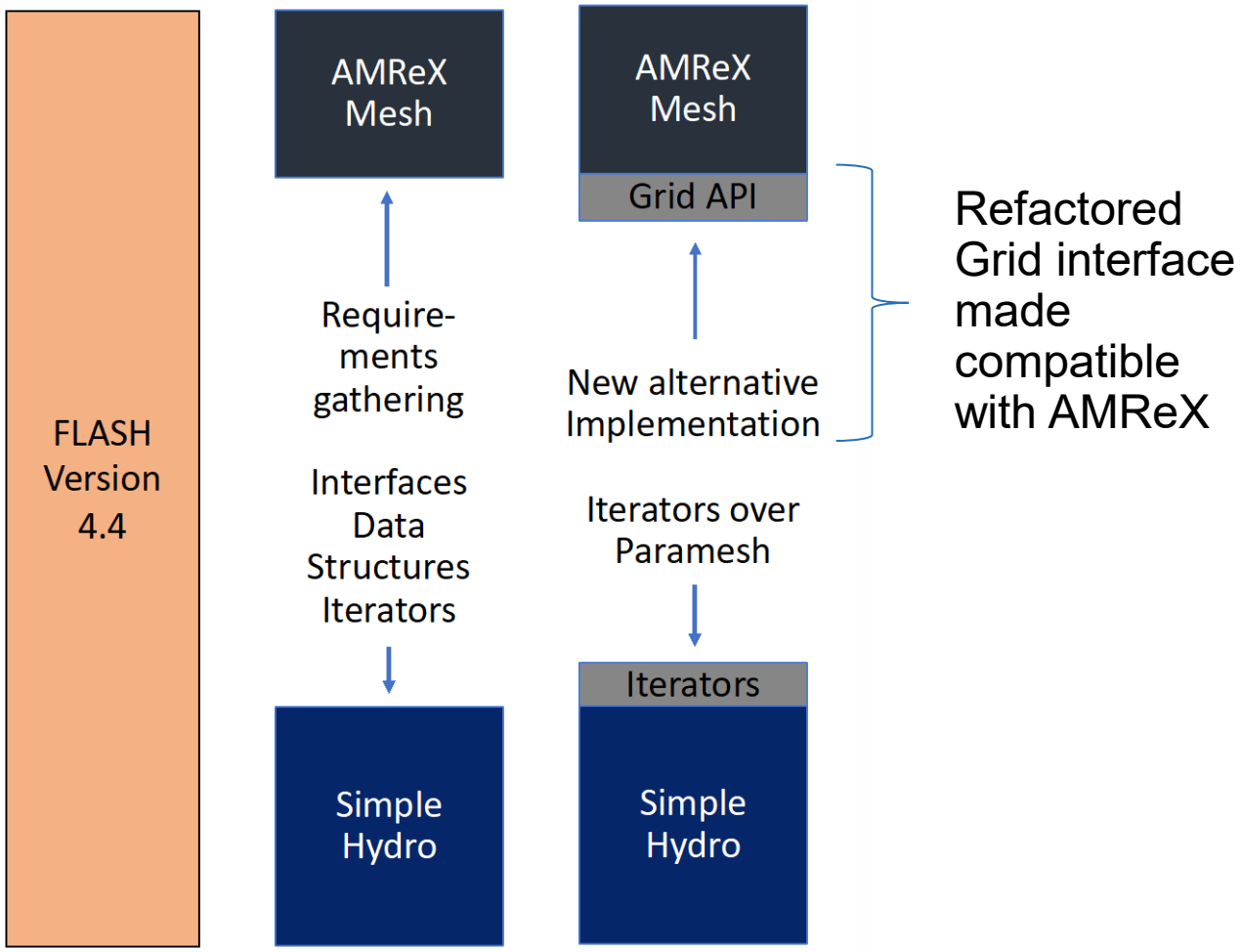


Part of a simpler environment for refactoring and testing

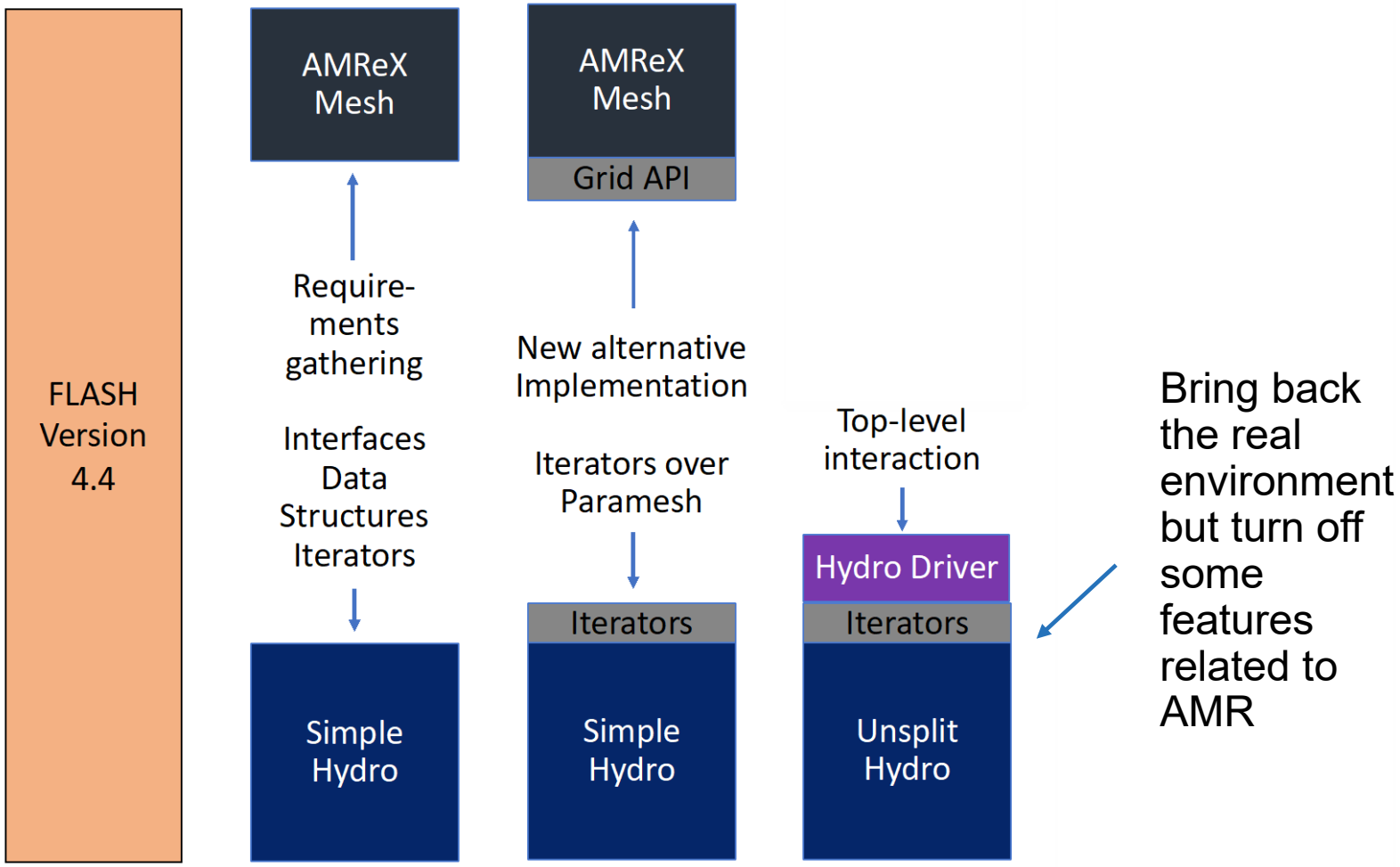
Steps in the Process



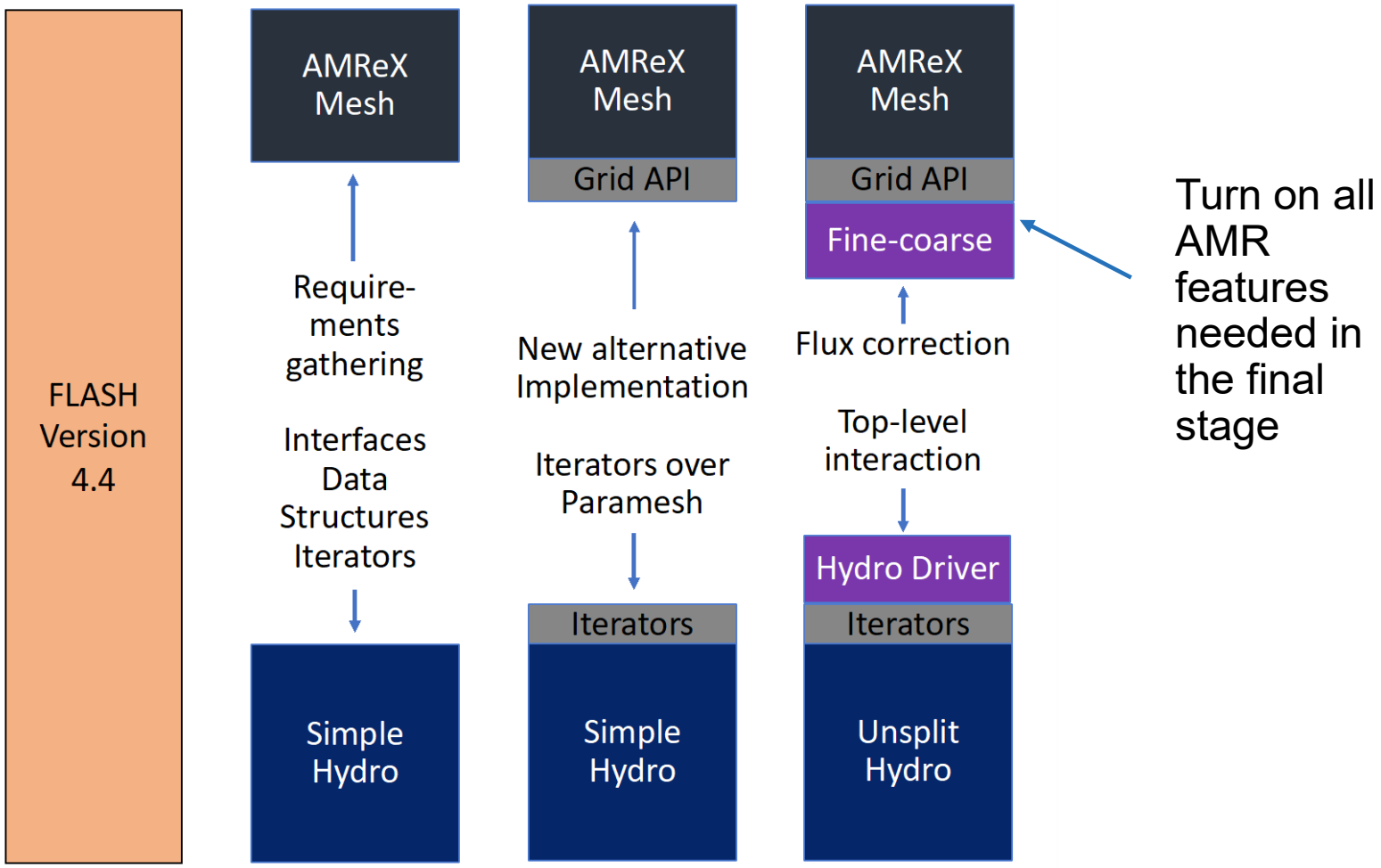
Steps in the Process



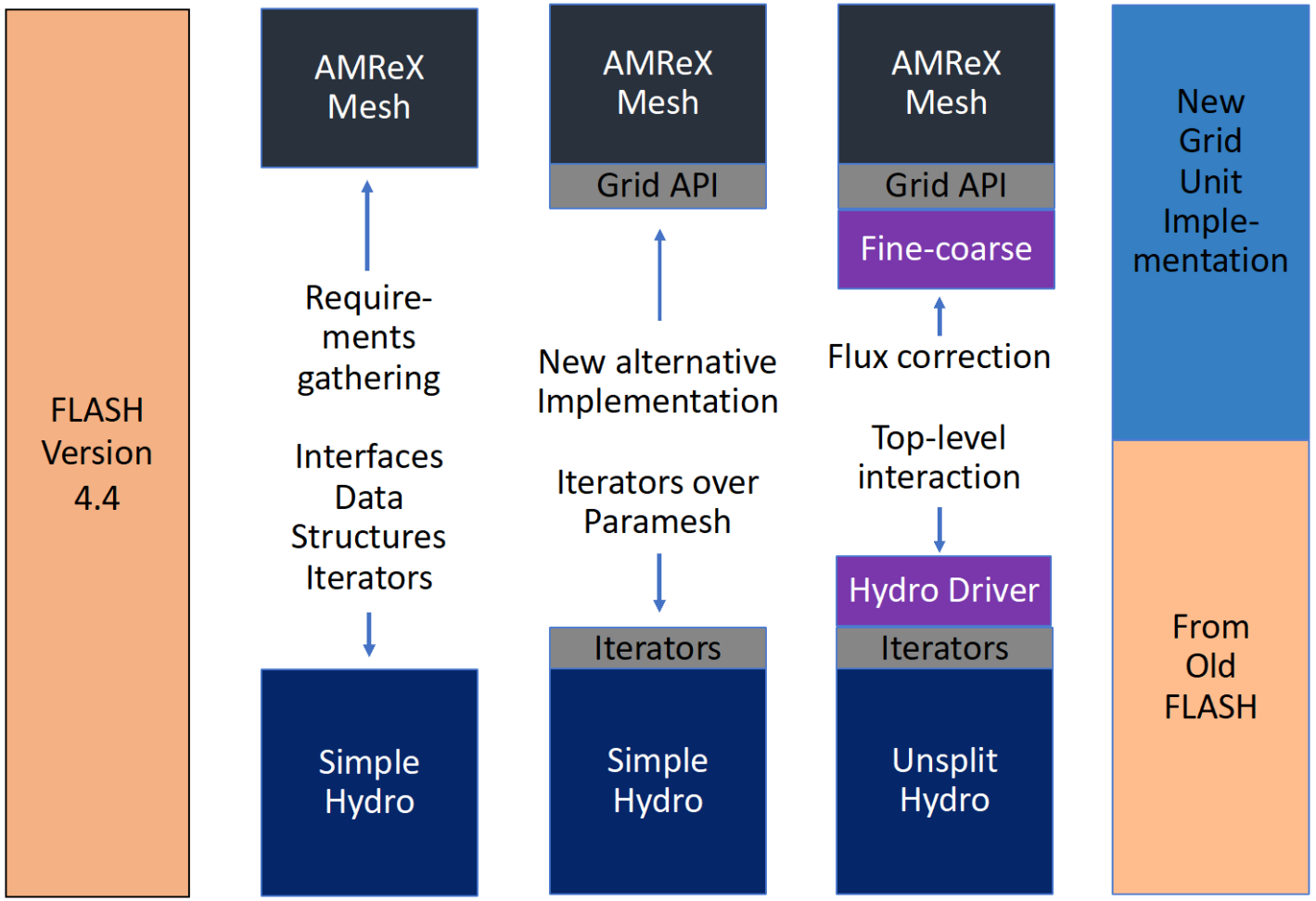
Steps in the Process



Steps in the Process



Steps in the Process



To Have a Good Outcome from Refactoring

1. Know why
2. Know how much
3. Know the cost
4. Plan
5. Have strong testing and verification
6. Get buy-in from stakeholders