Scientific Software Design

Anshu Dubey
Argonne National Laboratory

Better Scientific Software Tutorial, SC20, November 2020

License, Citation and Acknowledgements

License and Citation

- **The requested citation the overall tutorial is: David E. Bernholdt, Anshu Dubey, Patricia A. Grubel, Rinku K. Gupta, Better Scientific Software tutorial, in SC '20: International Conference for High Performance Computing, Networking, Storage and Analysis, online, 2020. DOI: 10.6084/m9.figshare.12994376**
- Individual modules may be cited as *Speaker, Module Title*, in Better Scientific Software tutorial…

Acknowledgements

- Additional contributors include: Mike Heroux, Alicia Klinvex, Mark Miller, Jared O'Neal, Katherine Riley, David Rogers, Deborah Stevens, James Willenbring
- 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 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 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 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 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.

Architecting scientific codes

Desirable Characteristics and Why They are Challenging

Extensibility

Well defined structure and modules
Encapsulation of functionalities

Architecting scientific codes

Desirable Characteristics and Why They are Challenging

Extensibility

Well defined structure and modules
Encapsulation of functionalities

Same data layout not good for all solvers. Many corner cases. Necessary lateral interactions

Architecting scientific codes

Desirable Characteristics and Why They are Challenging

Extensibility

Well defined structure and modules
Encapsulation of functionalities

Same data layout not good for all solvers. Many corner cases. Necessary lateral interactions

Performance

Spatial and temporal locality of data
Minimizing data movement
Maximizing scalability

Architecting scientific codes

Desirable Characteristics and Why They are Challenging

Extensibility	Performance
Well defined structure and modules Encapsulation of functionalities	Spatial and temporal locality of data Minimizing data movement Maximizing scalability
Same data layout not good for all solvers. Many corner cases. Necessary lateral interactions	Low arithmetic intensity solvers with hard dependencies. Proximity and work distribution at cross purposes

Architecting scientific codes

Desirable Characteristics and Why They are Challenging

Portability

General solutions that work without significant manual intervention across platforms

Architecting scientific codes

Desirable Characteristics and Why They are Challenging

Portability

General solutions that work without significant manual intervention across platforms

Tremendous platform heterogeneity
A version for each class of device => combinatorial explosion

Architecting scientific codes

Desirable Characteristics and Why They are Challenging

Portability

General solutions that work without significant manual intervention across platforms

Tremendous platform heterogeneity
A version for each class of device => combinatorial explosion

Verifiability and Maintainability

Clean code
Documentation
Comprehensive testing

Architecting scientific codes

Desirable Characteristics and Why They are Challenging

Portability	Verifiability and Maintainability
General solutions that work without significant manual intervention across platforms	Clean code Documentation Comprehensive testing
Tremendous platform heterogeneity A version for each class of device => combinatorial explosion	Wrong incentives Designing good tests is hard

Architecting scientific codes

Taming the Complexity: Separation of Concerns

Subject of research
Model
Numerics

More Stable
Discretization
I/O
Parameters

Architecting scientific codes

Taming the Complexity: Separation of Concerns

Subject of research
Model
Numerics

Treat differently

More Stable
Discretization
I/O
Parameters

Architecting scientific codes

Taming the Complexity: Separation of Concerns

Subject of research
Model
Numerics

Client Code
Mathematically complex

Treat differently

More Stable
Discretization
I/O
Parameters

Infrastructure
Data structures and movement

Architecting scientific codes

Taming the Complexity: Separation of Concerns

Subject of research
Model
Numerics

Client Code
Mathematically complex

Treat differently

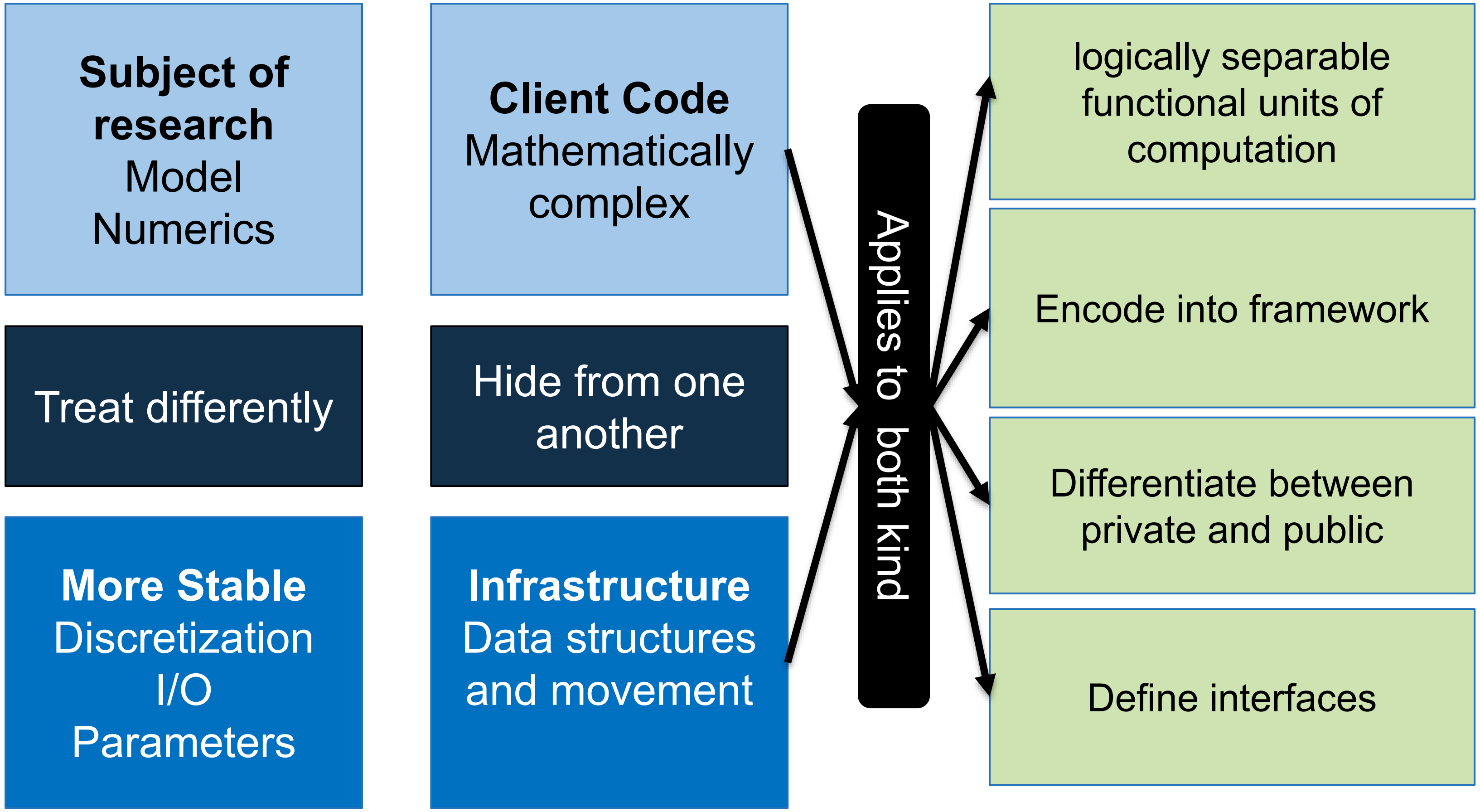
Hide from one another

More Stable
Discretization
I/O
Parameters

Infrastructure
Data structures and movement

Architecting scientific codes

Taming the Complexity: Separation of Concerns

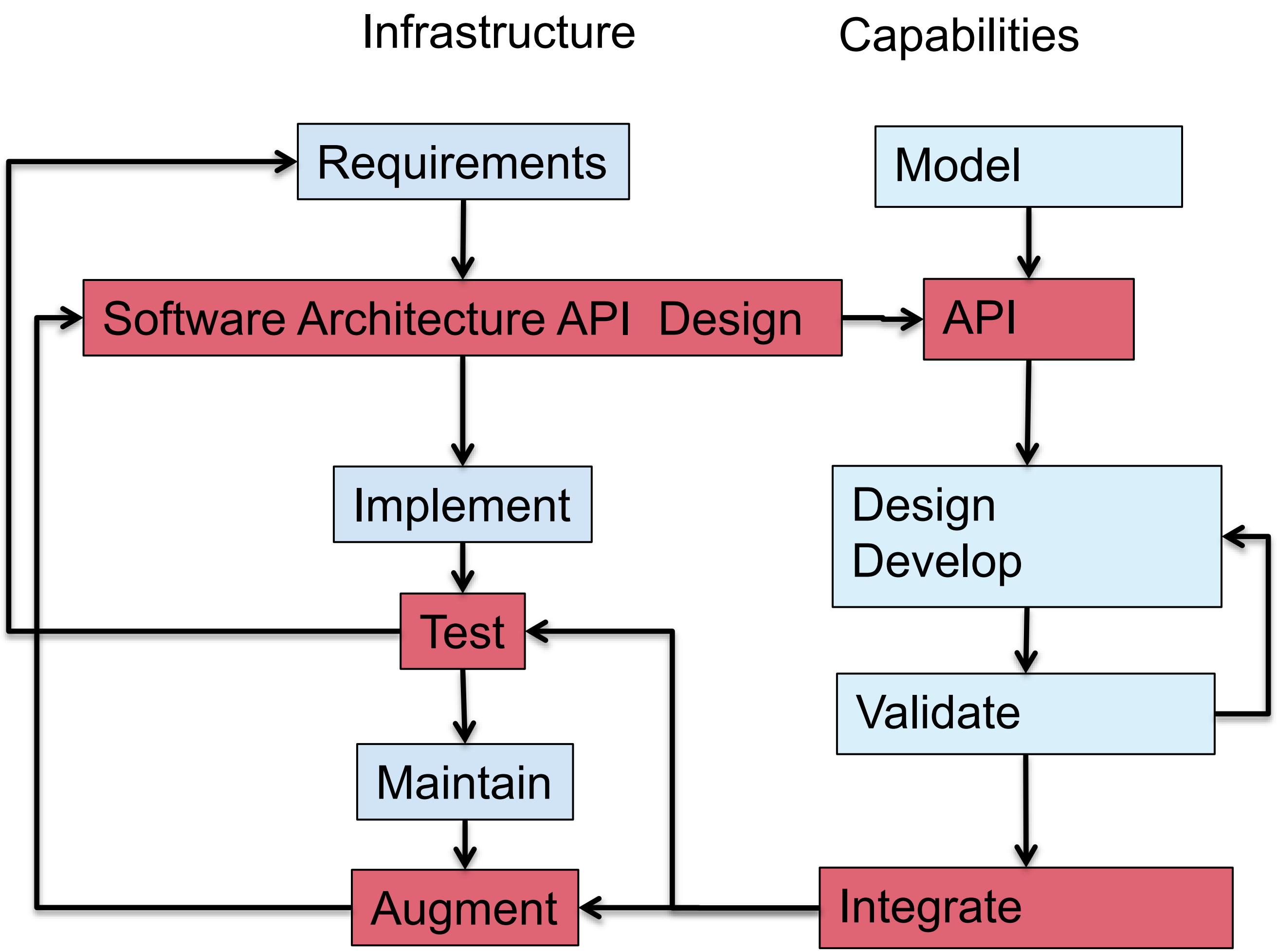


Other Considerations

- Leverage existing software
  - Libraries may have better solvers
    - Off-load expertise and maintenance
  - Examine the interoperability constraints
    - Many times the cost is justified even if there is more data movement
- More available packages are attempting to achieve interoperability
  - See if a combination meets your requirements
- May be worthwhile to let the library dictate data layout if the corresponding operations dominate

Institute a rigorous verification regime at the outset

A Design Model for Separation of Concerns



Design Considerations

- Infrastructure design
  - Take time to discuss, iterate over requirements and specification
  - Keep end users involved
    - Not doing so leaves possible options on the table
- Simple is better
  - Flexibility Vs transparent to the user
    - Flexibility wins

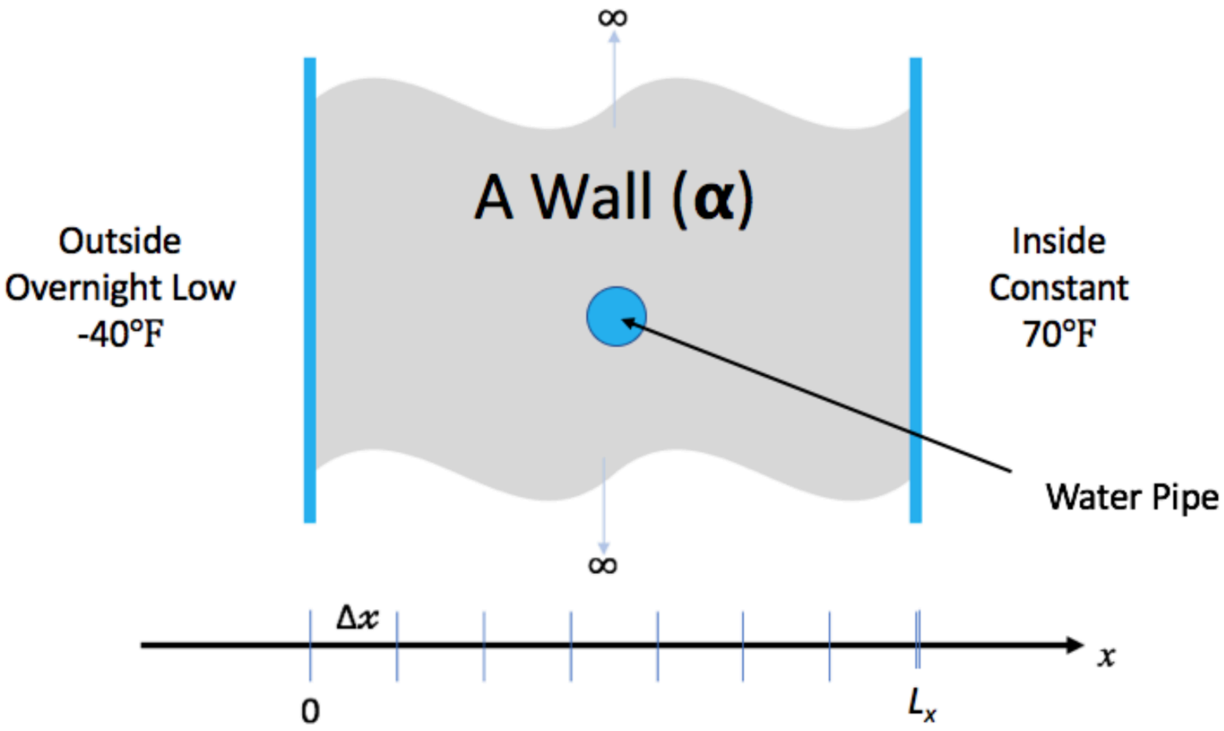
Design Considerations

- Infrastructure design
  - Take time to discuss, iterate over requirements and specification
  - Keep end users involved
    - Not doing so leaves possible options on the table
  - Keep API independent of numerics
- Simple is better
  - Flexibility Vs transparent to the user
    - Flexibility wins

- Model/numerics design
  - Abstract away the infrastructure knowledge as much as possible
  - Encapsulate
  - Let model needs guide API
  - Design flexible API to accommodate quick upgrades to methods
- Simple is better
  - Flexibility Vs transparent to the user
    - Flexibility wins

The Running Example

Lets say you live in a house with exterior walls made of a single material of thickness, $$L_x$$. Inside the walls are some water pipes as pictured below.



You keep the inside temperature of the house always at 70 degrees F. But, there is an overnight storm coming. The outside temperature is expected to drop to -40 degrees F for 15.5 hours. Will your pipes freeze before the storm is over?

Problem Specification - Design Considerations

- Specification
  - Solve heat equation with some initial and boundary conditions
  - Apply different integration methods

- What is infrastructure here?
  - Discretization/ State
  - Verification
  - I/O
  - Application of initial conditions
  - Runtime parameters
  - Comparison

- What is model here?
  - Initial conditions
  - Boundary conditions
  - Integration

Infrastructure API

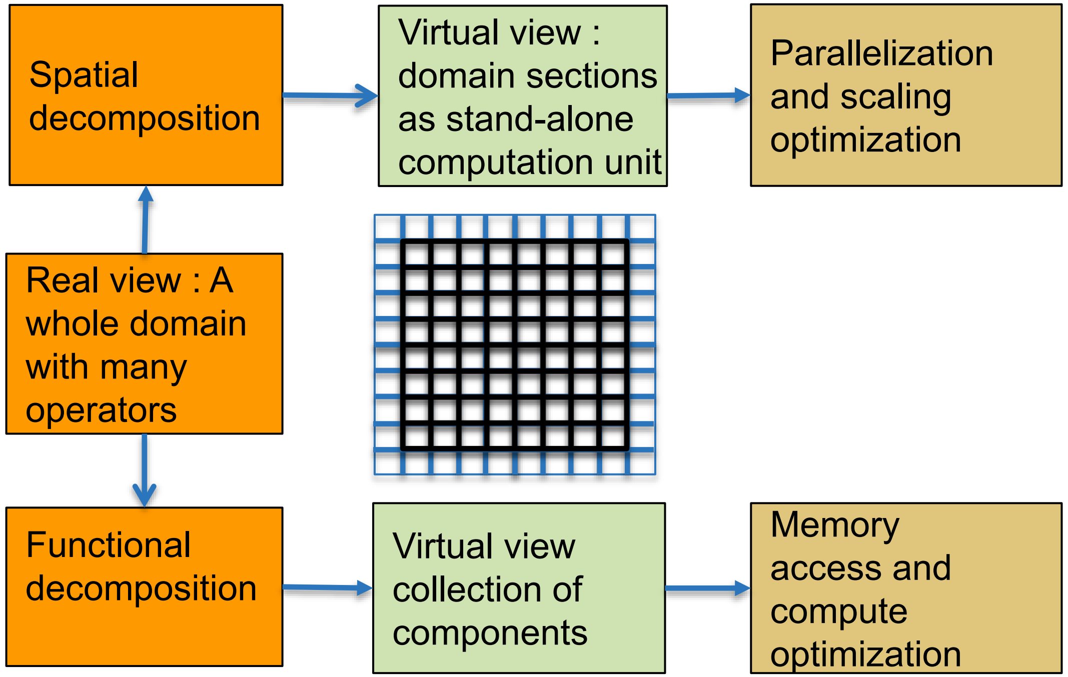
- process_args(int argc, char **argv)
- static void initialize(void)
- void copy(int n, double *dst, double const *src)
- void write_array(int t, int n, double dx, double const *a)
- void set_initial_condition(int n, double *a, double dx, char const *ic)

Numerics API

- double l2_norm(int n, double const *a, double const *b)
- static void r83_np_fa(int n, double *a)
- static void r83_np_sl (int n, double const *a_lu, double const *b, double *x)
- bool update_solution_crankn(int n, double *curr, double const *last, double const *cn_Amat, double bc_0, double bc_1)
- bool update_solution_upwind15(int n, double *curr, double const *last, double alpha, double dx, double dt, double bc_0, double bc_1)
- void compute_exact_solution(int n, double *a, double dx, char const *ic, double alpha, double t, double bc0, double bc1)
- bool update_solution_ftcs(int n, double *uk1, double const *uk0, double alpha, double dx, double dt, double bc0, double bc1)

Example: Architecting Multiphysics PDEs

- Virtual view of functionalities
- Decomposition into units and definition of interfaces



TAKEAWAYS

- DIFFERENTIATE BETWEEN SLOW CHANGING AND FAST CHANGING COMPONENTS OF YOUR CODE
- TAKE YOUR TIME TO UNDERSTAND THE REQUIREMENTS OF YOUR INFRASTRUCTURE
- IMPLEMENT SEPARATION OF CONCERNS
- DESIGN WITH PORTABILITY, EXTENSIBILITY, REPRODUCIBILITY AND MAINTAINABILITY IN MIND
- LEVERAGE EXISTING CAPABILITIES WHERE POSSIBLE

…….QUESTIONS ?