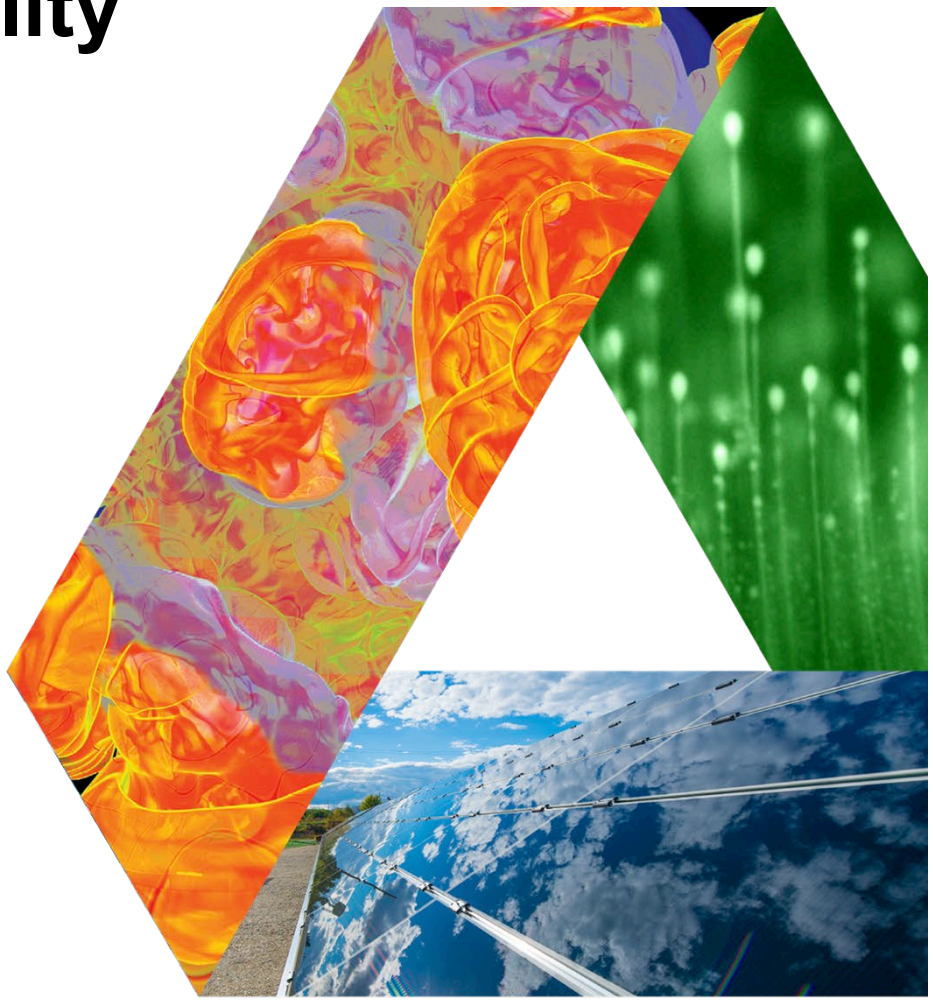
Software Design for Longevity with Performance Portability

Anshu Dubey
Argonne National Laboratory

HPC-BP Webinar

December 9, 2020



License, Citation and Acknowledgements

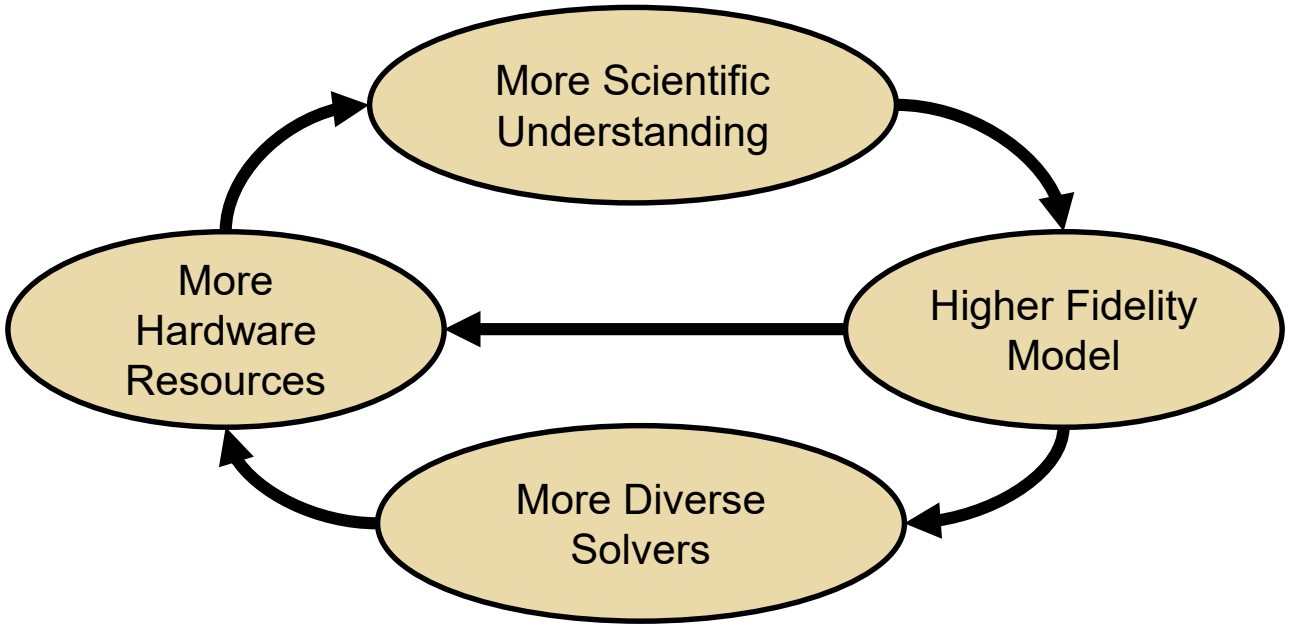
License and Citation

- This work is licensed under a Creative Commons Attribution 4.0 International License (CC BY 4.0).
- **The requested citation is: Anshu Dubey, Software Design for Longevity with Performance Portability, HPC-BP webinar, December 9, 2020 : DOI:10.6084/m9.figshare.13342265**

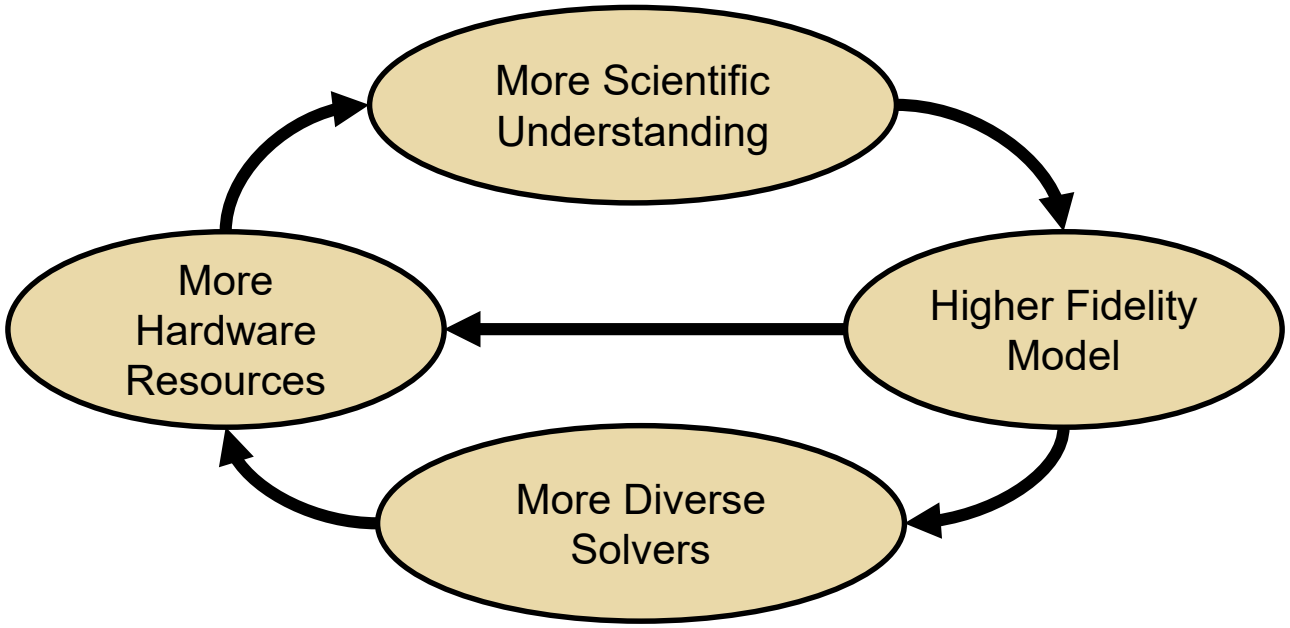
Acknowledgements

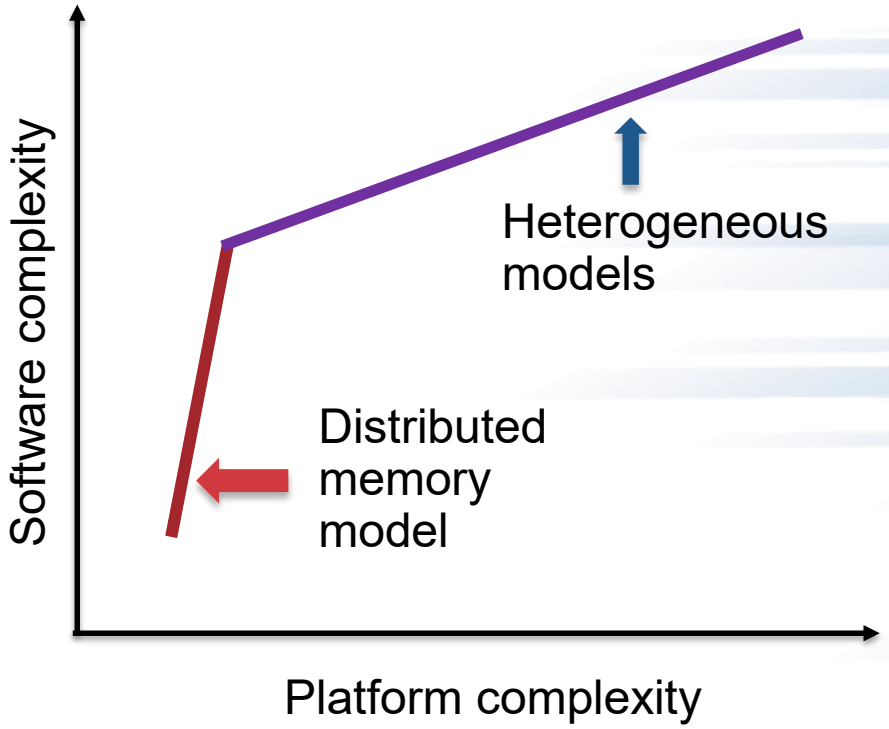
- This work was supported by the U.S. Department of Energy Office of Science, Office of Advanced Scientific Computing Research (ASCR), and by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration.
- This work was 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.

HPC Computational Science Use-case



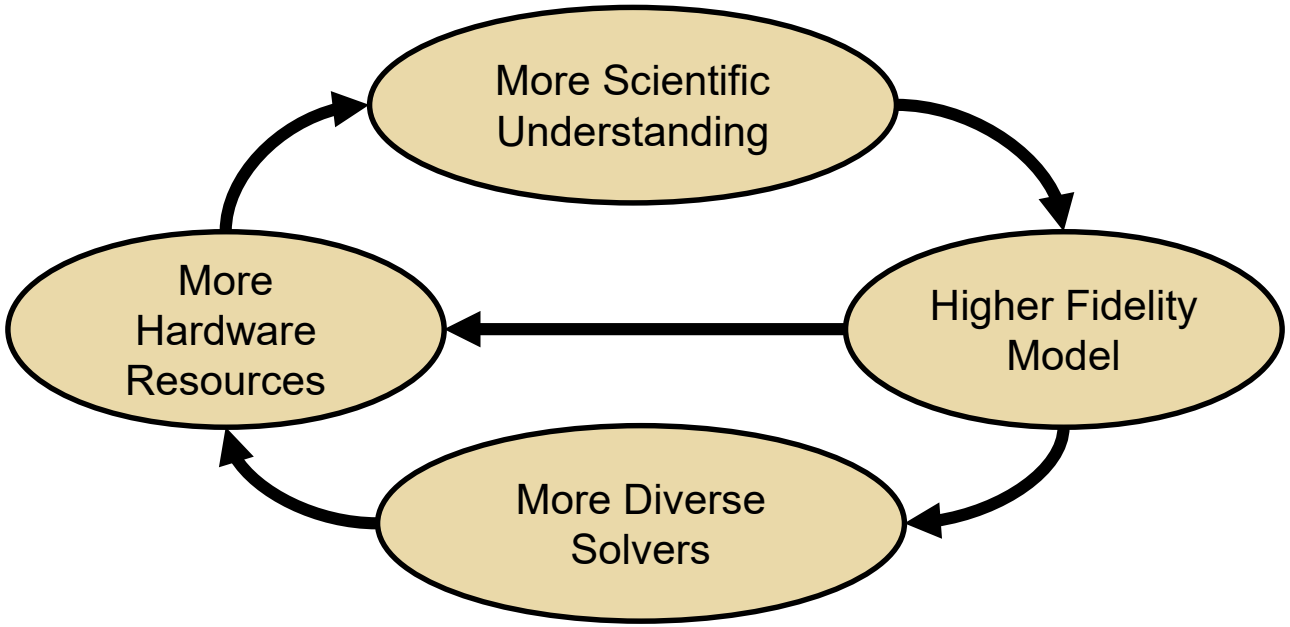
HPC Computational Science Use-case

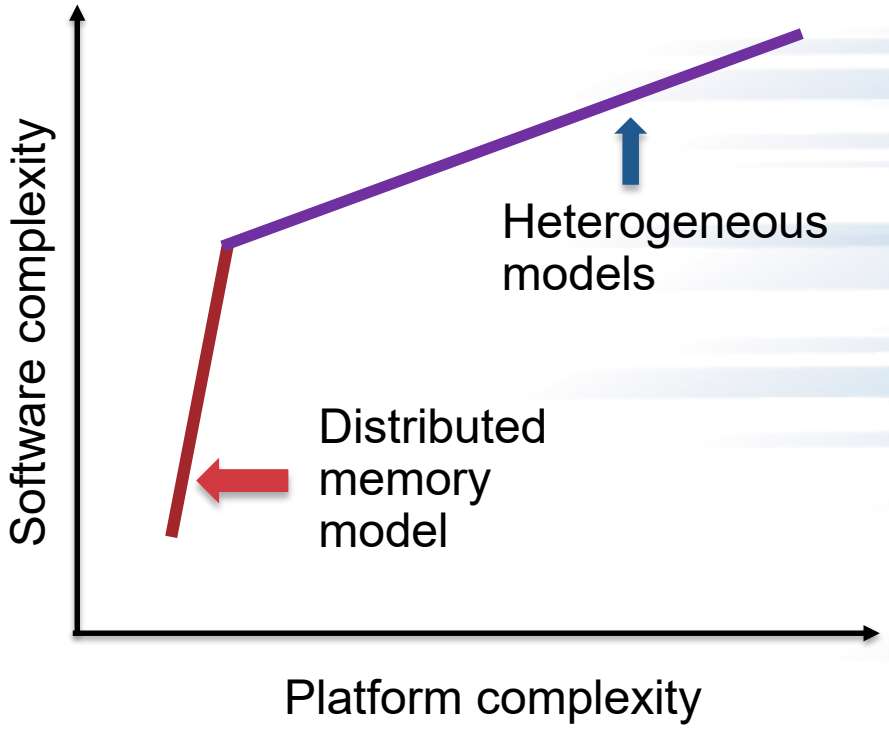




4

HPC Computational Science Use-case





- Many components may be under research
- Software continuously evolves
- All use cases are different and unique
- The US Exascale Computing Project (ECP) is at the forefront of these challenges

5

The ECP Performance Portability Series

The objective of ECP is to have participating applications and software technologies needed for their science be ready for the exascale platforms
For details about ECP please visit www.exascaleproject.org

- Motivation for the series
  - Platforms differ
    - What works well on one platform may not work *equally* well on others
    - ECP community has experiences in a variety of approaches; there is acquired wisdom
      - This wisdom should be shared as widely as possible
  - Need was felt for in-depth discussions
    - We had been considering focused in-person workshops
    - Panel series became the best available alternative during time of social distancing
- Outcomes
  - Share lessons learned, identify gaps, discover opportunities for partnerships
  - **Some basic design principles for performance portability also emerged**

For more information about the panel series please view
https://doi.org/10.6084/m9.figshare.13283714.v1

IDEAS productivity

ECP EXASCALE COMPUTING PROJECT

General Design Principles for HPC Scientific Software

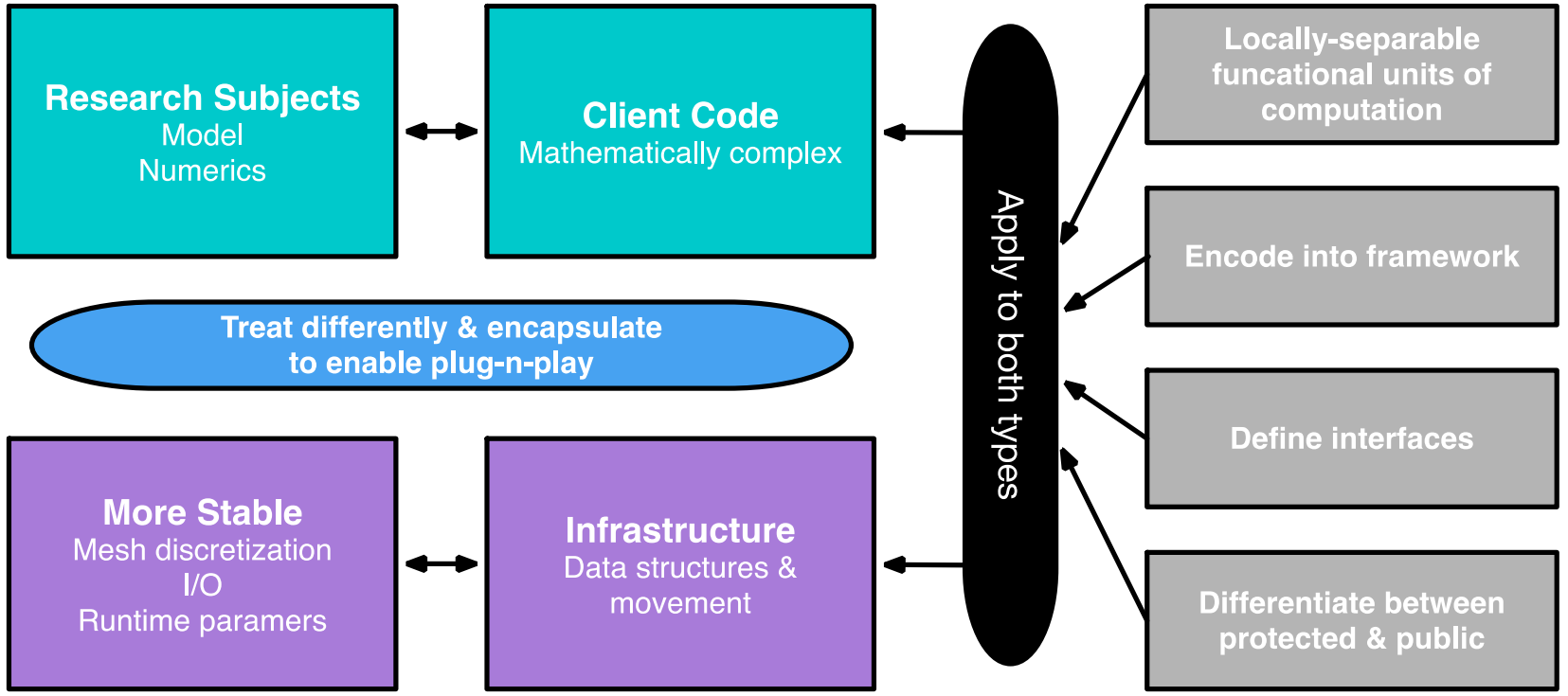
Considerations

- Multidisciplinary teams
  - Many facets of knowledge
  - To know everything is not feasible
- Two types of code components
  - Infrastructure (mesh/IO/runtime …)
  - Science models (numerical methods)
- Codes grow
  - New ideas => new features
  - Code reuse by others

Design Implications

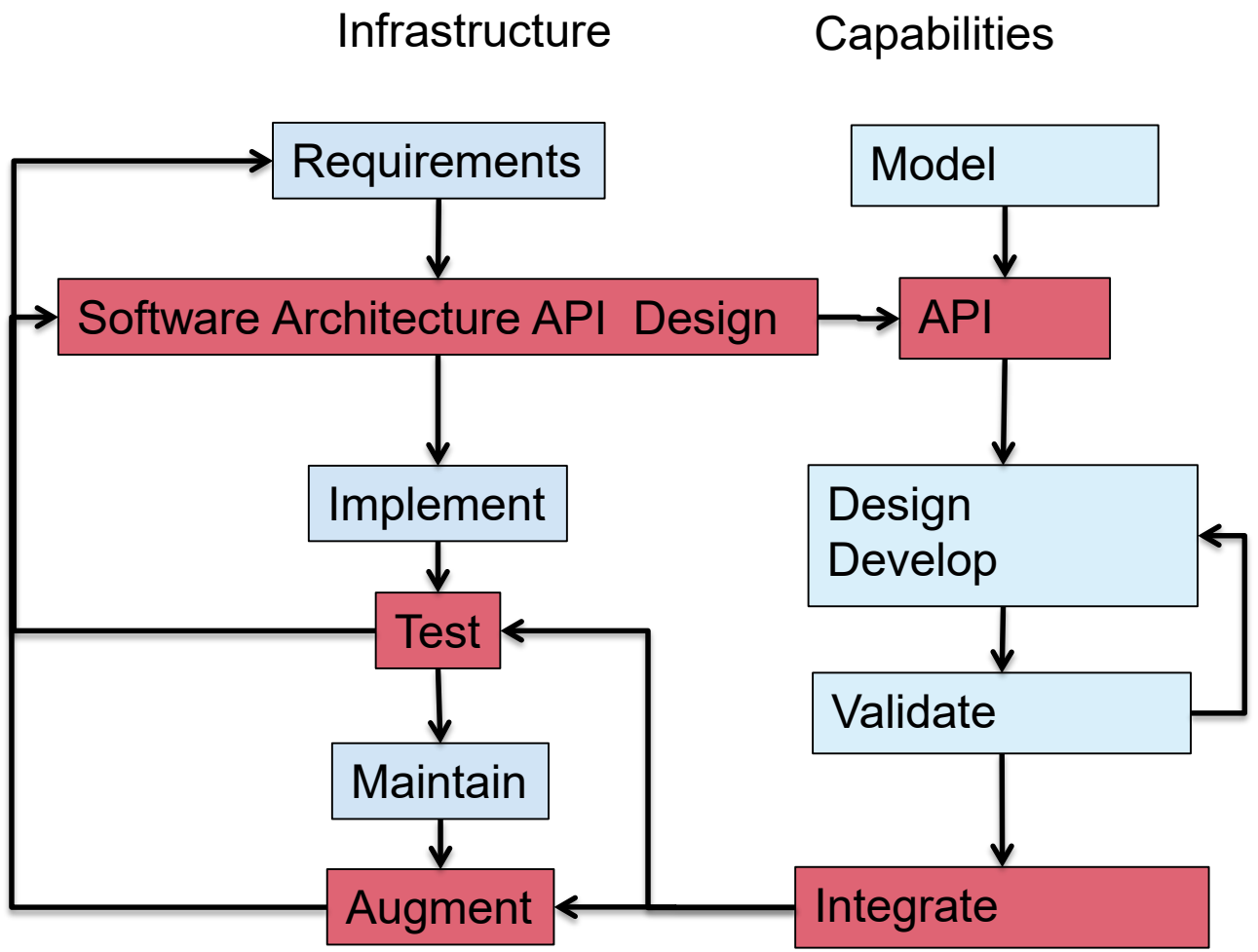
- Separation of Concerns
  - Shield developers from unnecessary complexities
- Work with different lifecycles
  - Long-lasting vs quick changing
  - Logically vs mathematically complex
- Extensibility built in
  - Ease of adding new capabilities
  - Customizing existing capabilities

General Design Principles for HPC Scientific Software



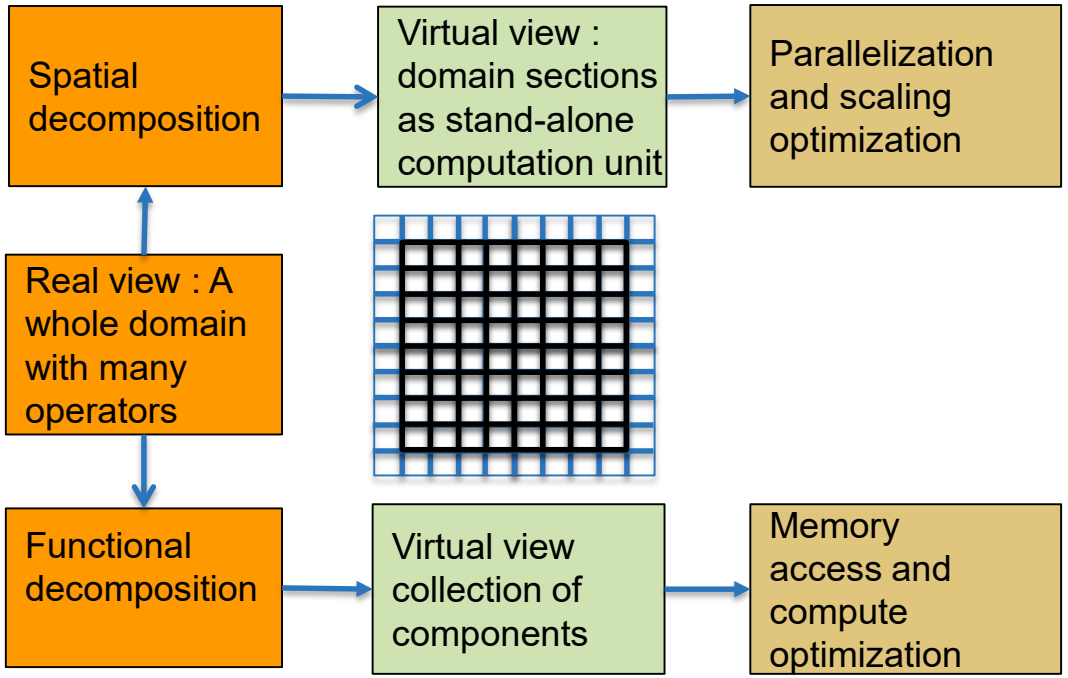
Design first, then apply programming model to the design instead of taking a programming model and fitting your design to it.

A Design Model for Separation of Concerns



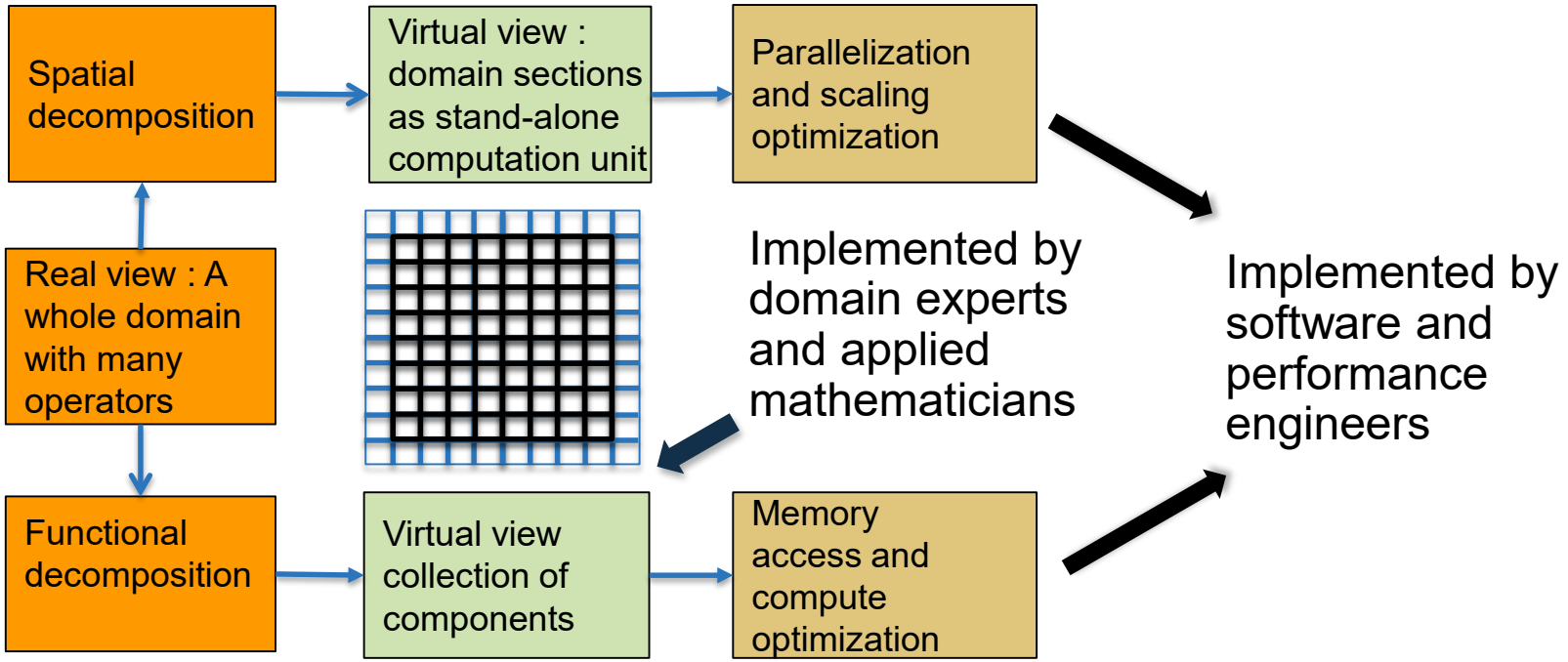
Example: Multiphysics PDEs for Distributed Memory Parallelism

- Virtual view of domain and functionalities
- Decomposition into components and definition of interfaces



Example: Multiphysics PDEs for Distributed Memory Parallelism

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



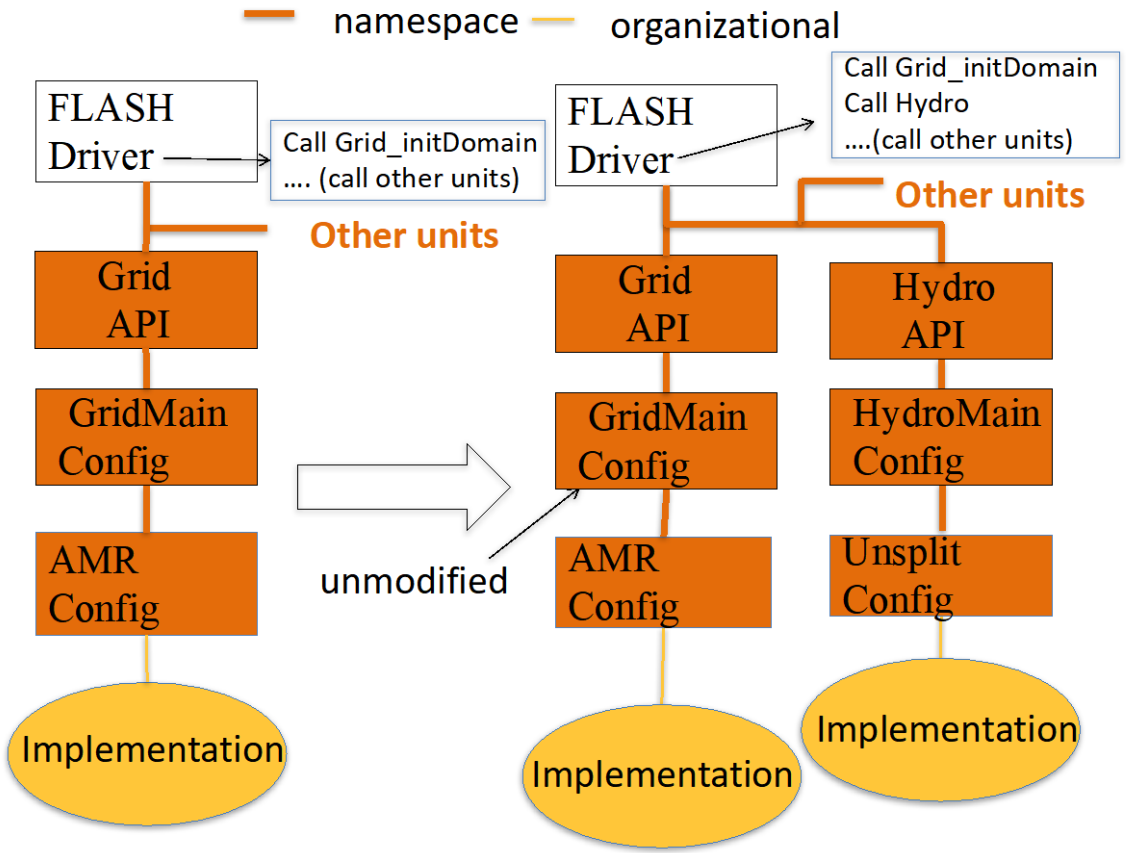
Example: Design for Extensibility from FLASH

Assumed that capabilities will be added for better models

- Assembly from components
- Decentralized maintenance of metadata
- Python tool to parse and configure
- OOP implemented through Unix directory structure and configuration tool

Key idea is distributed intelligence

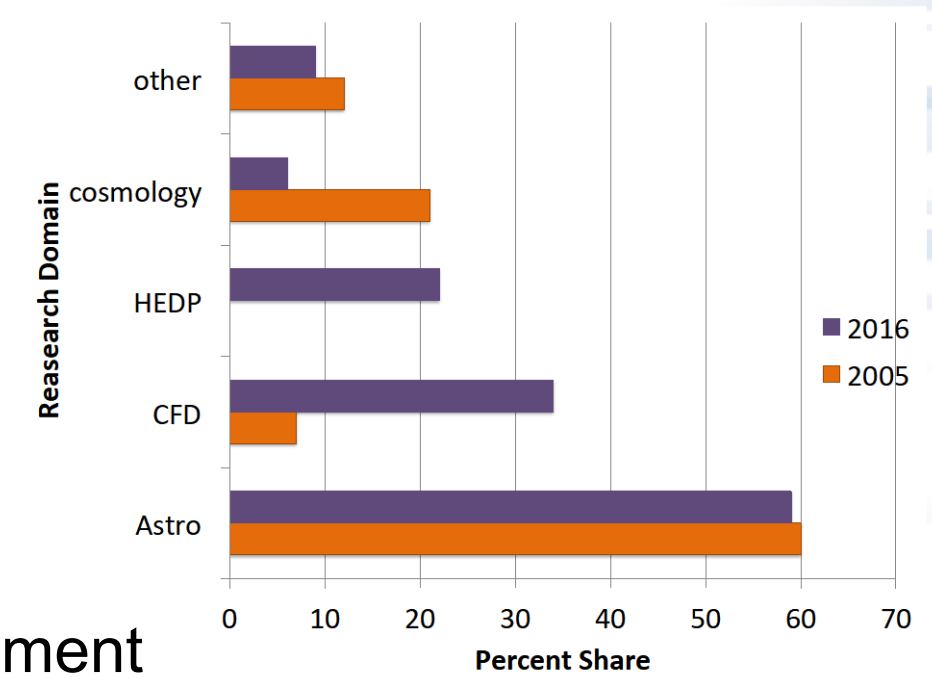
```
REQUIRES Driver
DEFAULT unsplit
EXCLUSIVE split unsplit Spark
VARIABLE dens TYPE: PER_VOLUME
.
.
VARIABLE temperature
PARAMETER small        REAL    1.E-10
.
.
PARAMETER smlrho       REAL    1.E-10
```



Dubey et al 2009: Extensible component-based architecture for FLASH, a massively parallel, multiphysics simulation code
https://doi.org/10.1016/j.parco.2009.08.001

Dividends from Investing in Design

	astro-physics	cosmo-logy	CFD/ FSI	HEDP	solar physics	recon-nection	star fo-rmation	combus-tion
compress-ible hydro	1998	*		*	*			*
burn	1999							*
MHD	2002	*		*	*	*	*	
elliptic solver	*	2001 2001	*				*	
particles	*	2002	*	*		*	*	*
bittree	*	*	2012	*				
HYPRE interface			*	2011				
radiation	*	*		2011				



52 Person years for infrastructure development

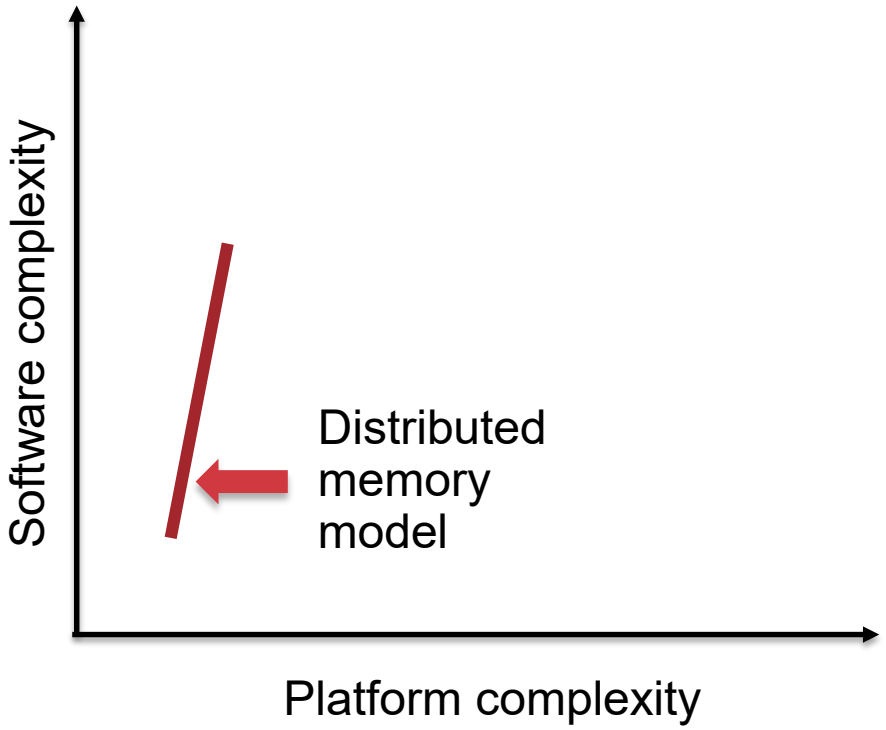
- Assume other communities reuse 75% of the infrastructure
- Saving of ~40 person years per new domain

Dubey et al 2017: The dividends of investing in computational software design: A case study
https://doi.org/10.1177/1094342017747692

IDEAS productivity

ECP EXASCALE COMPUTING PROJECT

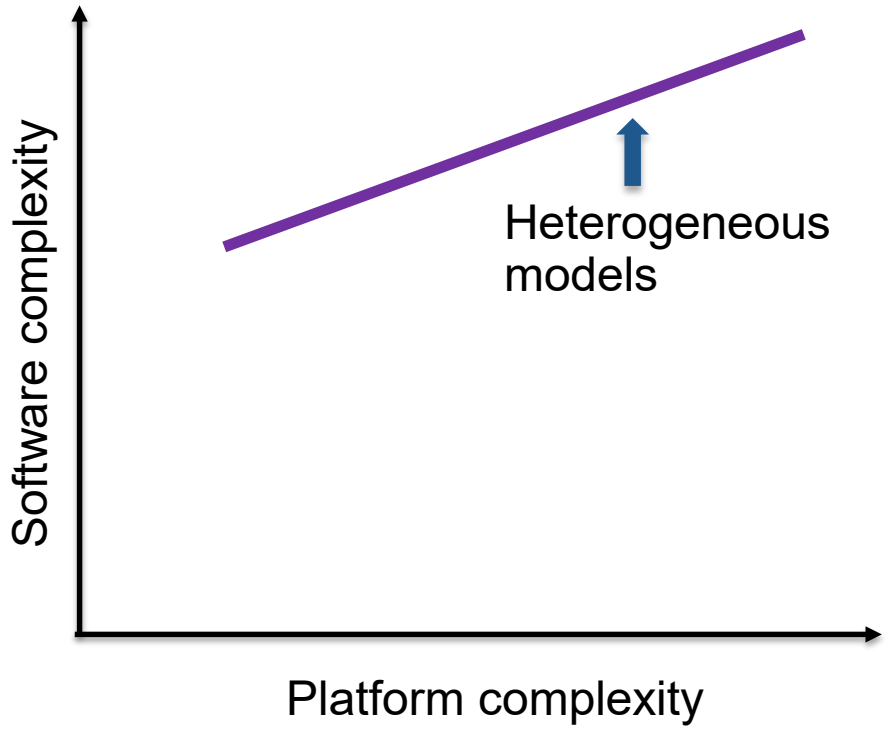
Takeaways Until Now



- Differentiate between slow changing and fast changing components of your code
- Understand the requirements of your infrastructure
- Implement separation of concerns
- Design with portability, extensibility, reproducibility and maintainability in mind
- Do not design with a specific programming model in mind

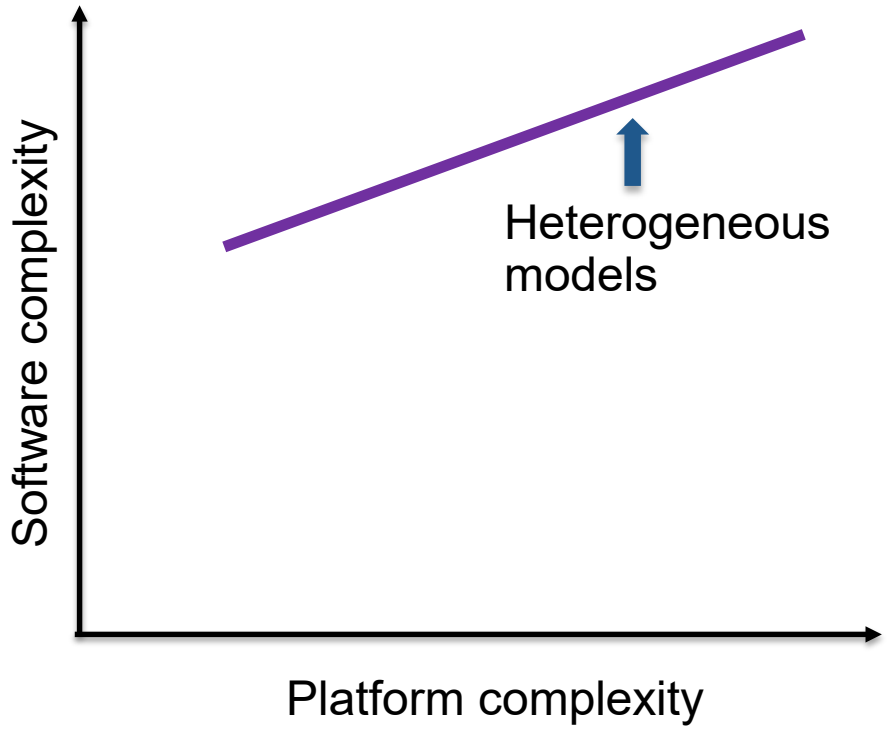
ANY QUESTIONS SO FAR?

A New Paradigm Because of Platform Heterogeneity



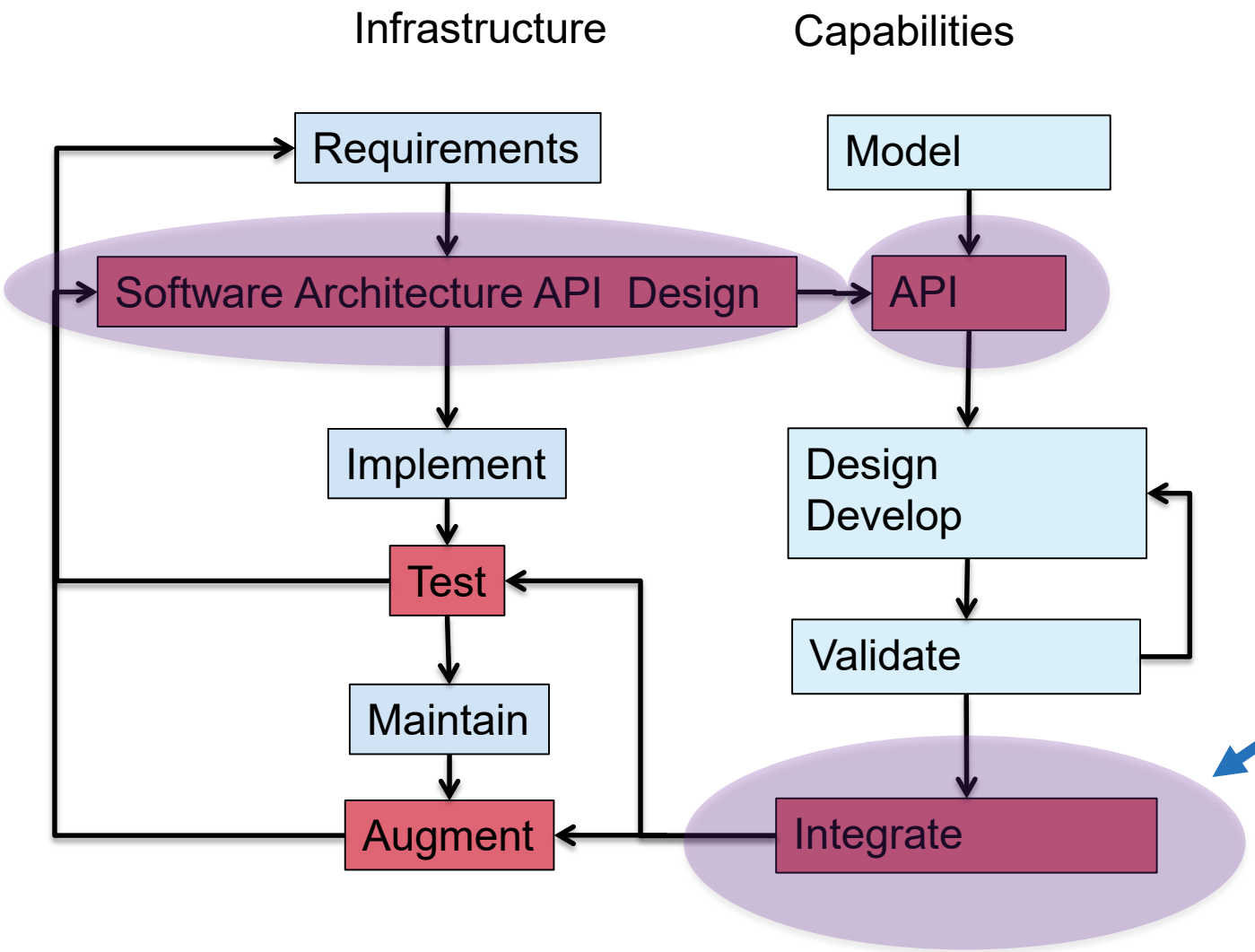
- Question - do the design principles change?

A New Paradigm Because of Platform Heterogeneity



- Question - do the design principles change?
- The answer is – not really
- The details get more involved

A Design Model for Separation of Concerns

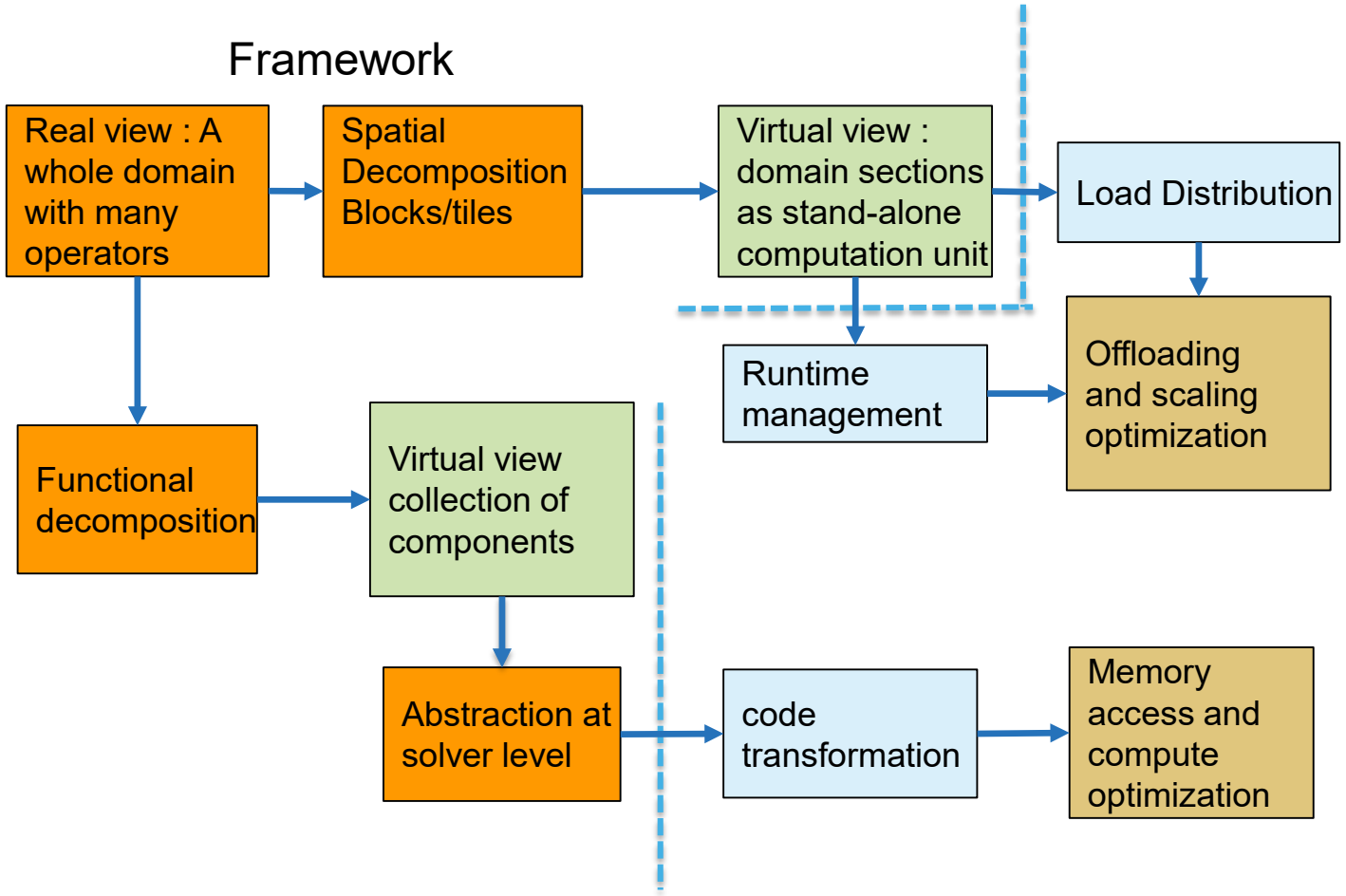


This is where maximum change is likely

Design Guidance Articulated in the Panel Series

- Design for Hierarchical parallelism
- Design towards several thousand threads
- Design for a hierarchical memory space
- Design patterns that count, allocate, and reuse memory
- Avoid exposing/using non-portable vendor-specific options

Features and Abstractions that must Come in



Approaches to Portability

Historically

- Hand-tune the code for the target
- Some teams are still doing it

Approaches to Portability

Historically

- Hand-tune the code for the target
- Some teams are still doing it

Current Trend

- Have multiple implementations
- Use third party abstraction tools

Approaches to Portability

Historically

- Hand-tune the code for the target
- Some teams are still doing it

Current Trend

- Have multiple implementations
- Use third party abstraction tools

Intermediate Option

- Refactor the code exposing opportunities for use of abstractions
- Figure out the parameters for plugging in abstractions
- Design composability into infrastructure
- Make tools, or leverage community tools that let you hand tune without all the pain

Underlying Ideas

Make the same code work on different devices

- A way to let compiler know that ”this” expression can be specialized in many ways
- Definition of specializations

Template meta-programming in abstraction layers

Underlying Ideas

Make the same code work on different devices

- A way to let compiler know that ”this” expression can be specialized in many ways
- Definition of specializations

Template meta-programming in abstraction layers

Assigning work within the node

- “Parallel For” or directives with unified memory
- Directives or specific programming model for explicit data movement

More complex data orchestration system for asynchronous computation

Underlying Ideas

Make the same code work on different devices

- A way to let compiler know that ”this” expression can be specialized in many ways
- Definition of specializations

Template meta-programming in abstraction layers

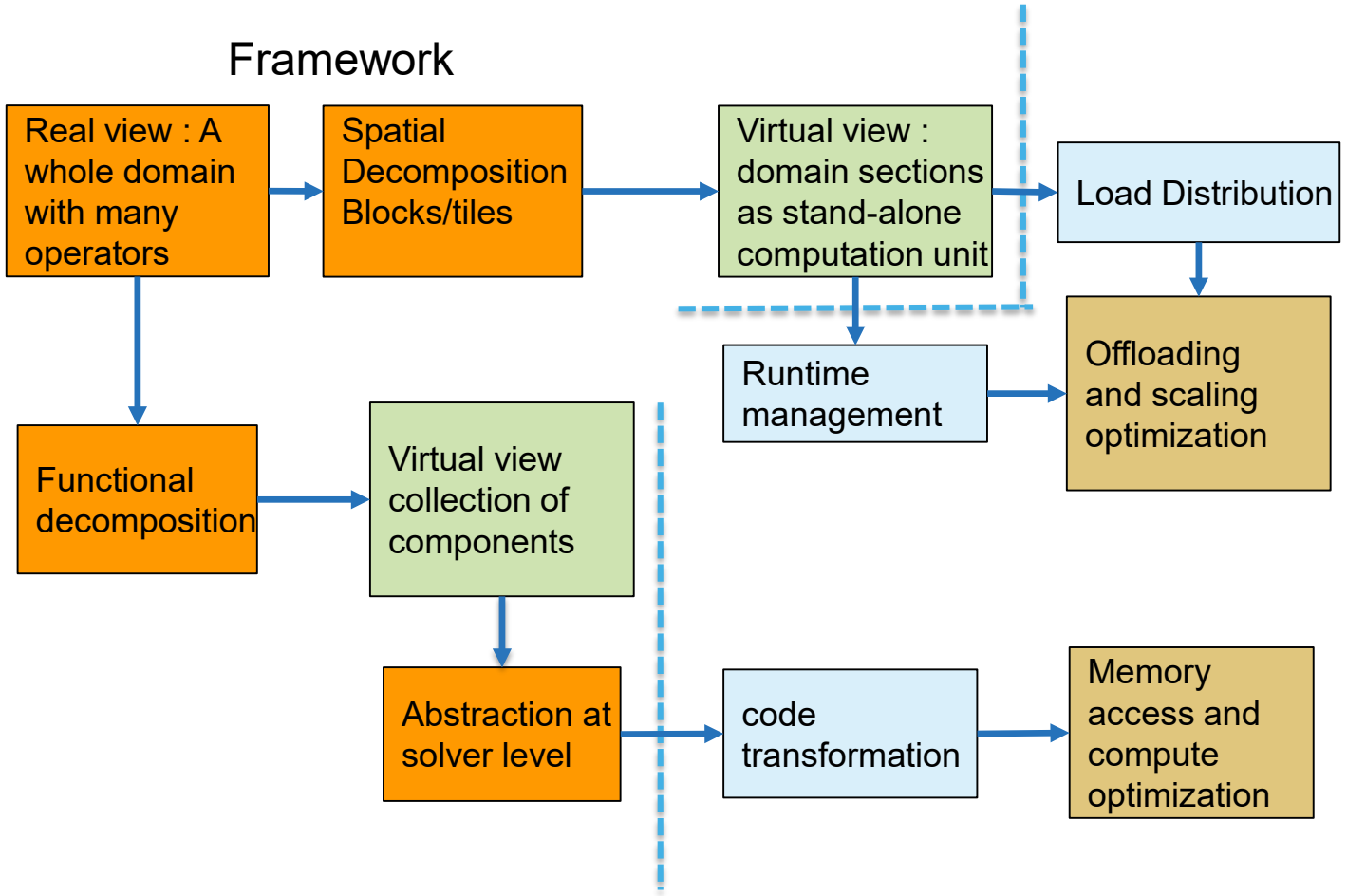
Look at what is needed, design for commonalities, encode them

Assigning work within the node

- “Parallel For” or directives with unified memory
- Directives or specific programming model for explicit data movement

More complex data orchestration system for asynchronous computation

Features and Abstractions that must Come in



How do abstraction layers work

- ❑ Infer the structure of the code
- ❑ Infer the map between algorithms and devices
- ❑ Infer the data movements
- ❑ Map computations to devices
- ❑ These are specified either through constructs or pragmas

Performance depends upon how well the mapping is done.

Design for Performance Portability

Example from Fortran with key-dictionary

- A computation on a 4D array
  - 1 dimension for state variables
- Copied into temporaries: uPlus, uMinus and flux

Code for CPU

```
subroutine recon(uPlus,uMinus,flux)
    real, pointer, dimension(:) :: uPlus,uMinus,flux
     if (flux(HY_MASS ) > 0.) then
        flux(HY_NUM_FLUX+1:NFLUXES ) = &
          uPlus(HY_NUM_VARS+1:NRECON )* &
          flux(HY_MASS)
     else
        flux(HY_NUM_FLUX+1:NFLUXES ) =  &
          uMinus(HY_NUM_VARS+1:NRECON )* &
          flux(HY_MASS)
     end if
```

Code for GPU

```
Subroutine recon(uPlus,uMinus,flux,iLow,iHigh,jLow,jHigh,lLow,kHigh)
    real, pointer, dimension(:,:,:,:) :: uPlus,uMinus,flux
    integer, iLow,iHigh,jLow,jHigh,kLow,kHigh
    integer :: i1,i2,i3
    do i3 = kLow,kHigh
        do i2 = jLow,jHigh
            do i1 = iLow, iHigh
                if (flux(HY_MASS ,i1,i2,i3) > 0.) then
                   flux(HY_NUM_FLUX+1:NFLUXES ,i1,i2,i3) = &
                       uPlus(HY_NUM_VARS+1:NRECON ,i1,i2,i3)* &
                       flux(HY_MASS ,i1,i2,i3)
                else
                   flux(HY_NUM_FLUX+1:NFLUXES ,i1,i2,i3 ) =  &
                       uMinus(HY_NUM_VARS+1:NRECON ,i1,i2,i3)* &
                      flux(HY_MASS ,i1,i2,i3)
                end if
            enddo
        enddo
    enddo
```

Design for Performance Portability

Example from Fortran with key-dictionary

- A computation on a 4D array
  - 1 dimension for state variables
- Copied into temporaries: uPlus, uMinus and flux

Code for CPU

```
subroutine recon(uPlus,uMinus,flux)
    real, pointer, dimension(:) :: uPlus,uMinus,flux
     if (flux(HY_MASS ) > 0.) then
        flux(HY_NUM_FLUX+1:NFLUXES ) = &
          uPlus(HY_NUM_VARS+1:NRECON )* &
          flux(HY_MASS)
     else
        flux(HY_NUM_FLUX+1:NFLUXES ) =  &
          uMinus(HY_NUM_VARS+1:NRECON )* &
          flux(HY_MASS)
     end if
```

Code for GPU

```
Subroutine recon(uPlus,uMinus,flux,iLow,iHigh,jLow,jHigh,lLow,kHigh)
    real, pointer, dimension(:,:,:,:) :: uPlus,uMinus,flux
    integer, iLow,iHigh,jLow,jHigh,kLow,kHigh
    integer :: i1,i2,i3
    do i3 = kLow,kHigh
        do i2 = jLow,jHigh
            do i1 = iLow, iHigh
                if (flux(HY_MASS ,i1,i2,i3) > 0.) then
                   flux(HY_NUM_FLUX+1:NFLUXES ,i1,i2,i3) = &
                       uPlus(HY_NUM_VARS+1:NRECON ,i1,i2,i3)* &
                       flux(HY_MASS ,i1,i2,i3)
                else
                   flux(HY_NUM_FLUX+1:NFLUXES ,i1,i2,i3 ) =  &
                       uMinus(HY_NUM_VARS+1:NRECON ,i1,i2,i3)* &
                      flux(HY_MASS ,i1,i2,i3)
                end if
            enddo
        enddo
    enddo
```

- Different dimensionalities for the temporaries
- No do loop vs explicit do loop in the kernel

Design for Performance Portability

Step 1: temporaries and arguments

Key Definitions for CPU

```
[hy_recon_args]
  uPlus, uMinus, flux

[hy_recon_declare]
real, pointer, dimension(:) :: uPlus, uMinus,flux
```

Key Definitions for GPU

```
[hy_recon_args]
uPlus, uMinus,
flux,iLow,iHigh,jLow,jHigh,kLow,kHigh

[hy_recon_declare]
 real, pointer, dimension(:,:,:,:) :: uPlus, uMinus, flux
 integer :: iLow,iHigh,jLow,jHigh,kLow,kHigh
```

Step 2: constructs

Key definitions for CPU kernels (null)

```
[hy_ind3spec]    [hy_inline_loop]

 [hy_inline_loop_end]
```

Key definitions for GPU kernels

```
[hy_inline_loop]
 do i3 = kLow,kHigh
          do i2 = jLow,jHigh
               do i1 = iLow, iHigh

[hy_inline_loop_end]          [hy_ind3spec]
         enddo                  ,i1,i2,i3
      enddo
   enddo
```

Design for Performance Portability

Subroutine Definition

```
subroutine recon(@hy_recon_args)
@hy_recon_declare

   @hy_inline_loop
   if (flux(HY_MASS @hy_ind3spec) > 0.) then
     flux(HY_NUM_FLUX+1:NFLUXES @hy_ind3spec) = &
        uPlus(HY_NUM_VARS+1:NRECON @hy_ind3spec)*  &
        flux(HY_MASS @hy_ind3spec)
    else
     flux(HY_NUM_FLUX+1:NFLUXES @hy_ind3spec) = &
        uMinus(HY_NUM_VARS+1:NRECON @hy_ind3spec)*  &
       flux(HY_MASS @hy_ind3spec)
    end if
    @hy_inline_loop_end
```

Design for Performance Portability

Subroutine Definition

```
subroutine recon(@hy_recon_args)
@hy_recon_declare

   @hy_inline_loop
   if (flux(HY_MASS @hy_ind3spec) > 0.) then
     flux(HY_NUM_FLUX+1:NFLUXES @hy_ind3spec) = &
        uPlus(HY_NUM_VARS+1:NRECON @hy_ind3spec)*  &
        flux(HY_MASS @hy_ind3spec)
    else
     flux(HY_NUM_FLUX+1:NFLUXES @hy_ind3spec) = &
        uMinus(HY_NUM_VARS+1:NRECON @hy_ind3spec)*  &
       flux(HY_MASS @hy_ind3spec)
    end if
    @hy_inline_loop_end
```

Ideally one would go through a similar exercise of locating good use of abstractions to obtain good results from using third-party abstraction tools

Approaches to Portability

Historically

- Hand-tune the code for the target
- Some teams are still doing it

Current Trend

- Have multiple implementations
- Use third party abstraction tools

Intermediate Option

- Refactor the code exposing opportunities for use of abstractions
- Figure out the parameters for plugging in abstractions
- Design composability into infrastructure
- Make tools, or leverage community tools that let you hand tune without all the pain

A highlight from the panel series is that users of Kokkos and Raja derived greater benefit if they understood their code's structure and needs

In other words, thought about design

FINAL TAKEAWAYS

- The key to both performance portability and longevity is careful software design
- Extensibility should be built into the design
- Design should be independent of any specific programming model
- Composability and flexibility help with performance portability

RESOURCES:

https://www.exascaleproject.org/

https://doi.org/10.6084/m9.figshare.13283714.v1

https://figshare.com/articles/presentation/SC20_Tutorial_Better_Scientific_Software/12994376?file=25219346

https://bssw.io/blog_posts/performance-portability-and-the-exascale-computing-project

https://www.exascaleproject.org/event/kokkos-class-series