Putting it All Together
(the Sociology of) Using Numerical Packages In Practice

Presented to
ATPESC 2020 Participants

Ann Almgren
Deputy Director, ECP Block-Structured AMR Co-Design Center

Date 08/04/2020

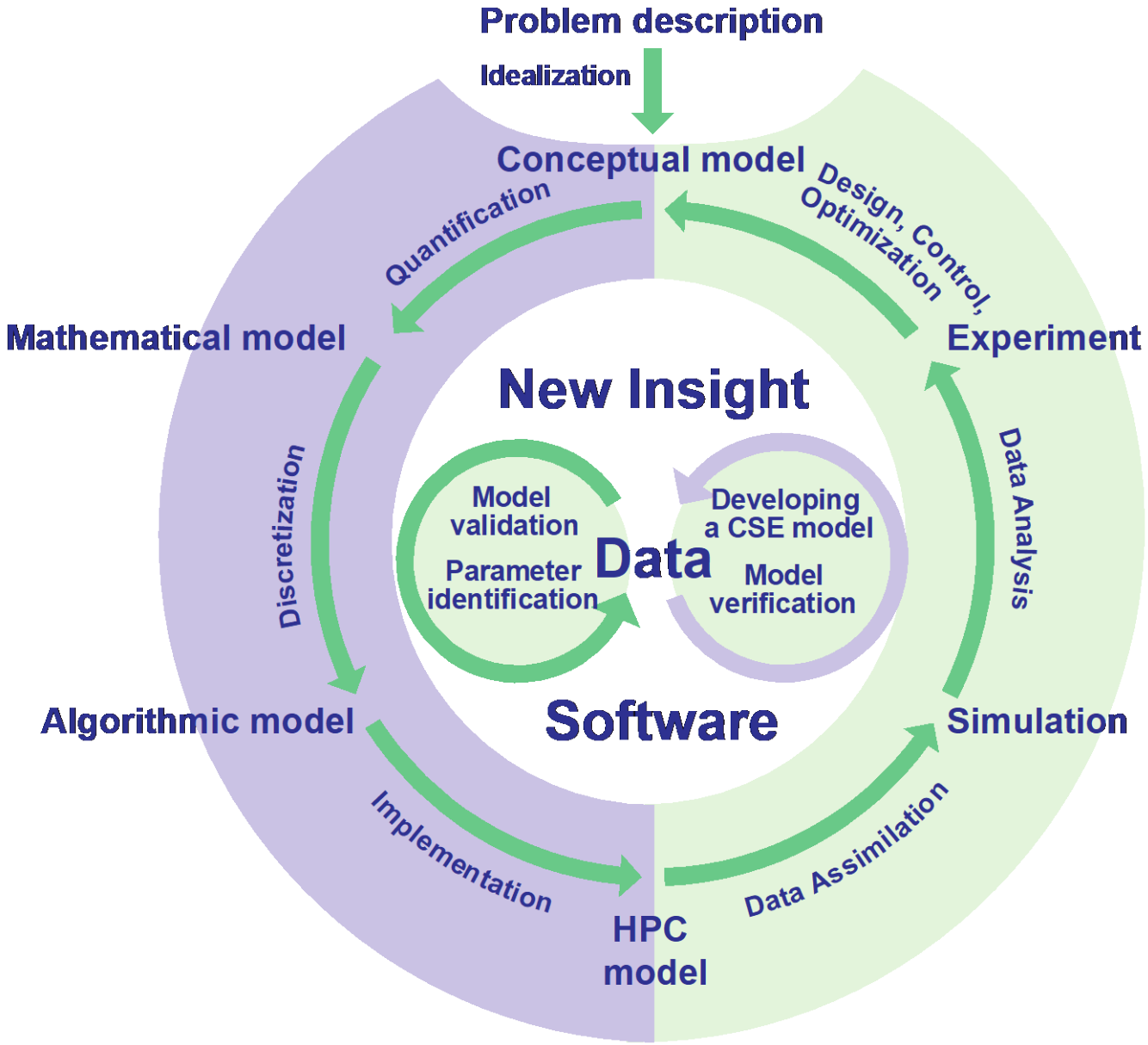
U.S. DEPARTMENT OF ENERGY | Office of Science

ATPESC Numerical Software Track

EXASCALE COMPUTING PROJECT

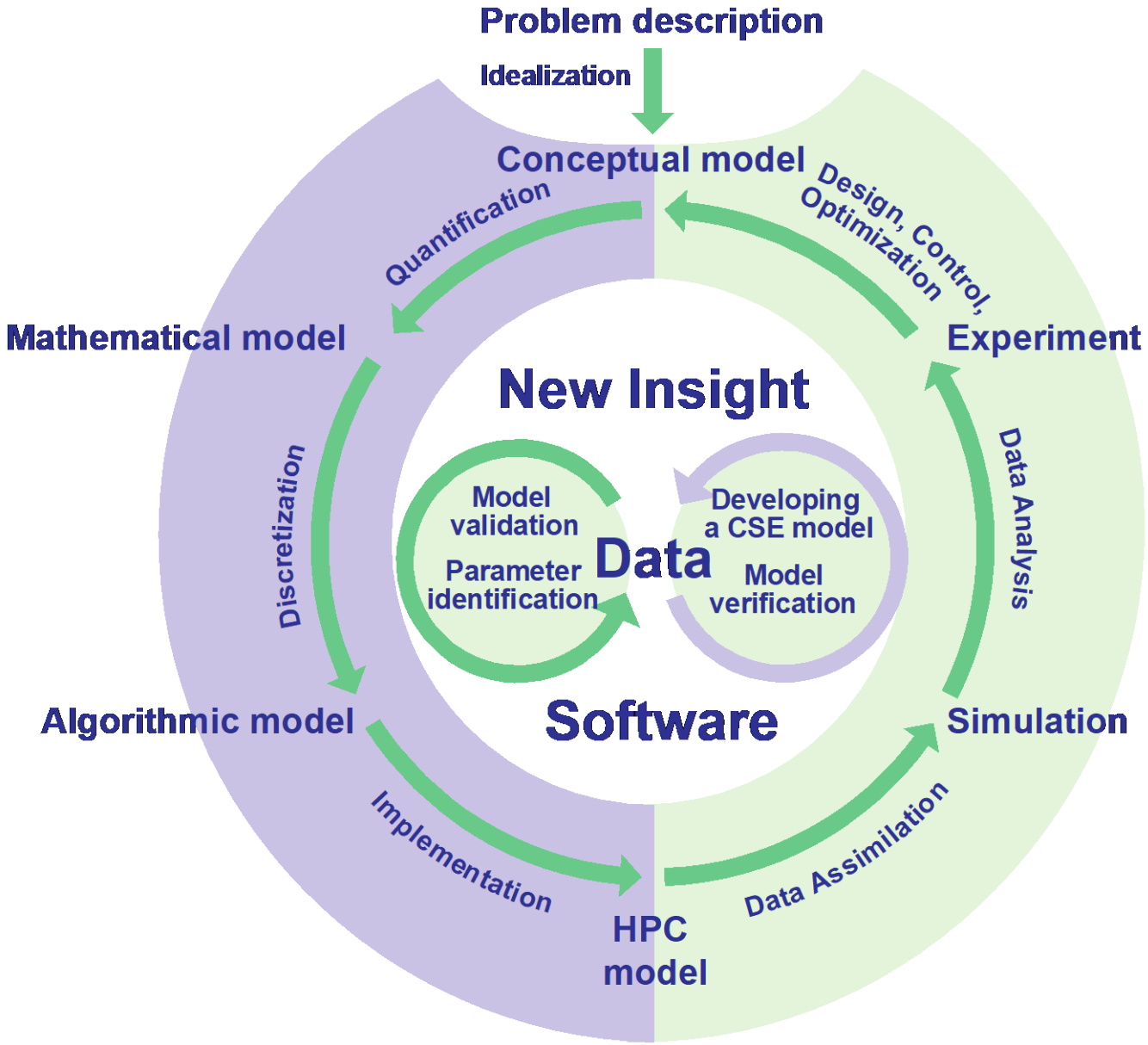
Argonne NATIONAL LABORATORY · BERKELEY LAB · Sandia National Laboratories · Rensselaer · SMU

Recall from this morning's introduction:



Recall from this morning's introduction:

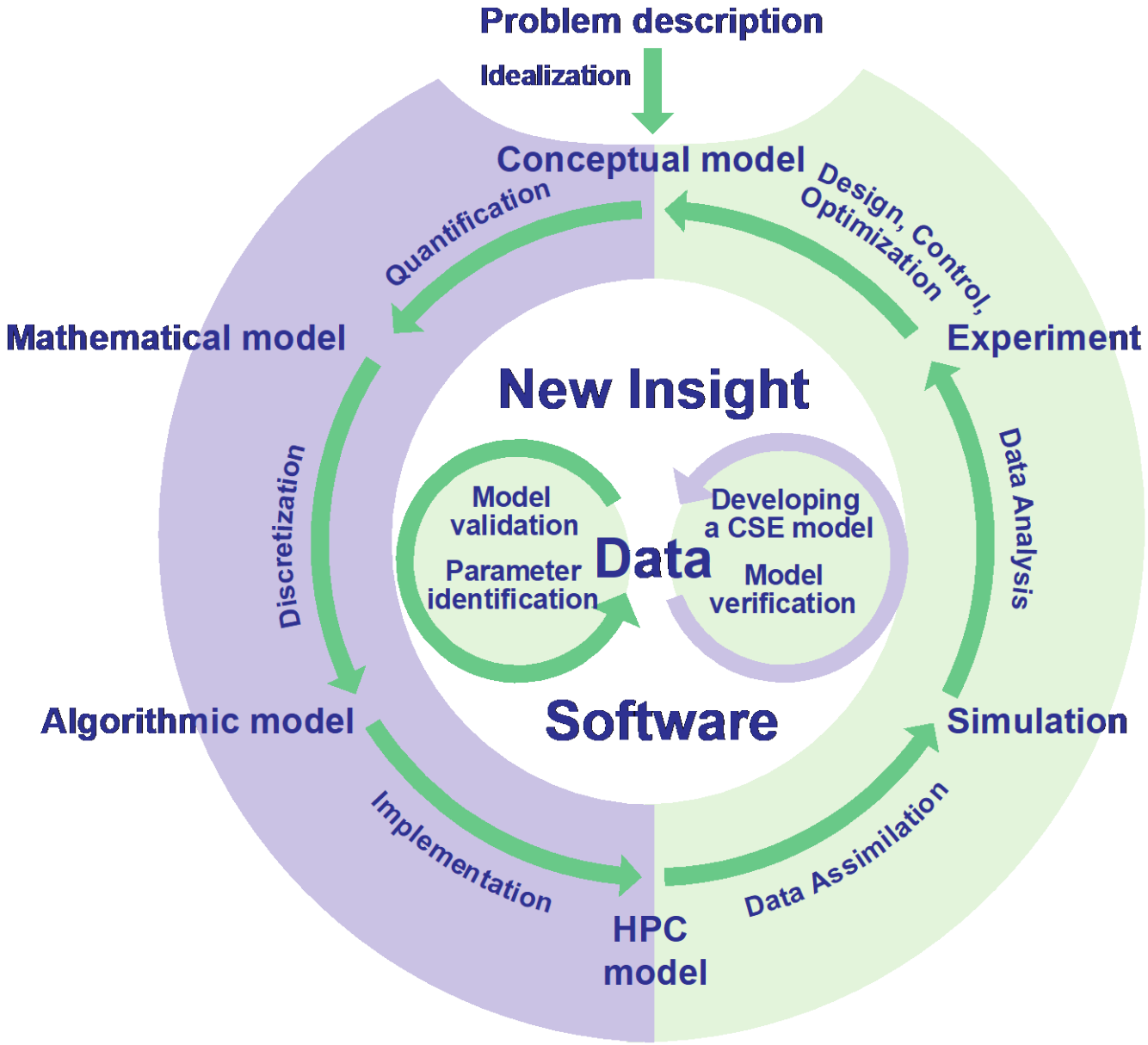
None of us can actually do it all



Recall from this morning's introduction:

None of us can actually do it all

So where do you want to spend your time?



Key steps of simulation science application development

- Physical model
  - Expertise may be very domain-specific
- Mathematical model
  - Expertise may require detailed mathematical knowledge
- Discretization and algorithm development
  - Expertise includes knowing regimes of applicability, stability, approximation, error bounds
- Parallel implementation
  - Expertise in hardware, software stack and parallel programming models

That's a lot of expertise!

Very few of us are experts in all of these areas. So how do we optimize the insight/impact of our computational science?

- Team science – in an ideal world we could work in teams that have all the relevant expertise within one team

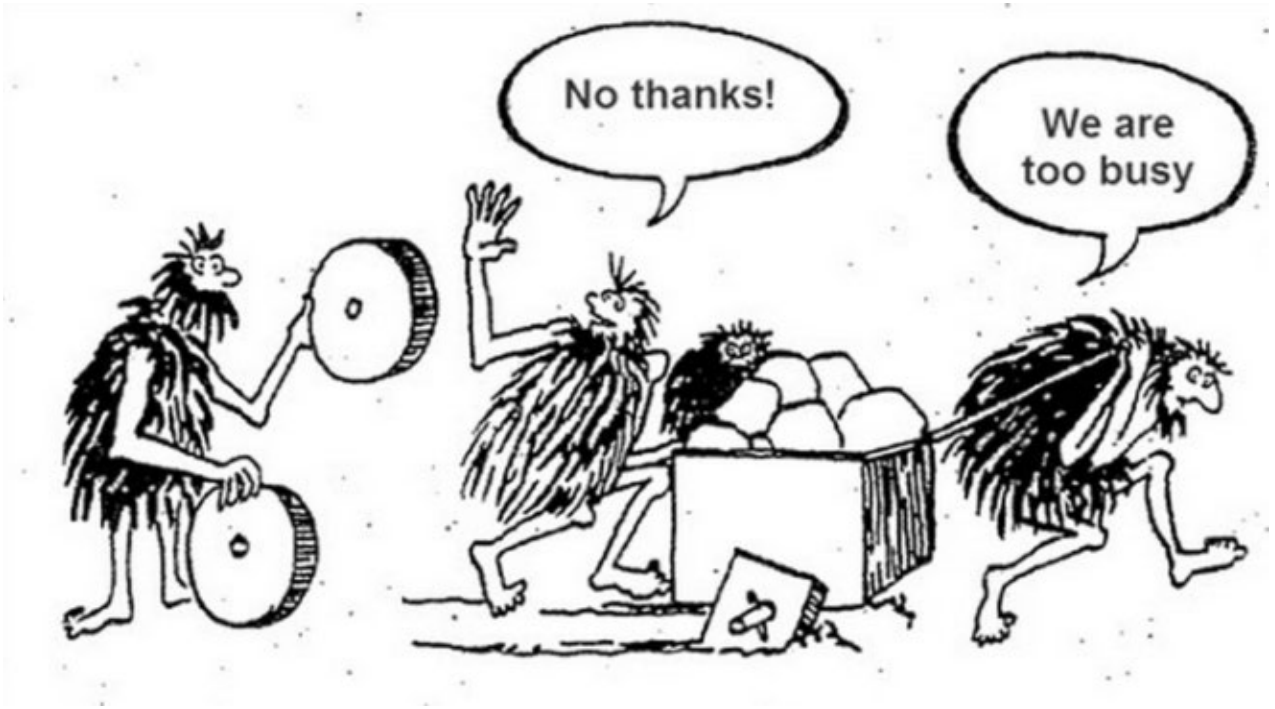
- That's not always possible –so one way to broadly share expertise is through software libraries
  - Expertise in discretization and algorithm development
  - Expertise in hardware, software stack and parallel programming models

In the short-term we often prefer to do things ourselves

$$\nabla^2 T = 0 \in \Omega$$

$$T(0) = 180^o$$

$$T(1) = 0^o$$



For the 1-D heat equation why bother learning a software package?

Sometimes simple is good

We can prototype in matlab, build simple serial implementations, and demonstrate proof-of-concept.

This can be good:

- New algorithms are often designed and validated in this mode.
- Sometimes writing your own version of a known technology (e.g. multigrid solver) is worth it -- “learning by doing”

This can be bad:

- Our own implementations are more likely to lack generality, be inefficient or even buggy.
- How much time do we spend “reinventing the wheel?”

Sometimes simple is good

We can prototype in matlab, build simple serial implementations, and demonstrate proof-of-concept.

This can be good:

- New algorithms are often designed and validated in this mode.
- Sometimes writing your own version of a known technology (e.g. multigrid solver) is worth it -- “learning by doing”

This can be bad:

- Our own implementations are more likely to lack generality, be inefficient or even buggy.
- How much time do we spend “reinventing the wheel?”
- **Do we impact anyone/anything beyond our own immediate application?**

The “supply” side of software libraries

Software libraries/frameworks/tools are made by real people.

The people aspect matters

- Software developers know a lot about their product
- But they don’t necessarily know exactly what you need

Communication/Collaboration is an important part of the process it’s good for the developer as well as the user!

The “supply” side of software libraries

Software libraries/frameworks/tools are made by real people.

The people aspect matters

- Software developers know a lot about their product
- But they don’t necessarily know exactly what you need

Communication/Collaboration is an important part of the process it’s good for the developer as well as the user!

Why don’t people “just” use software libraries

Lack of knowledge – how do you know whether the right tool even exists?

And if it exists: Where do you find it? How do you use it? Will it work with your other tools?

AKA: “package fatigue”

Why don’t people “just” use software libraries

Frustration! It can be really frustrating to not have the tool do what you want as well as you want. And how do you tell whether it’s you or the tool?

So how can you find the right tool – if it exists - and how do you learn how to use it correctly?

Ideal solution: a “toolbox” of compatible (interoperable) tools that “just work”

- This is exactly what the software developers are working towards
- But it takes time and resources
- The developer/user interaction can be a win-win

On a practical level, there are trade-offs

Advantages

- Key challenges addressed well
  - Portable, Performant, Scalable, Interoperable
- Numerics are well tested/vetted
- Functionality is often more general than you would have made yourself
- More science, more impact; less time writing/debugging software
- Become part of a community – for collaboration and help

On a practical level, there are trade-offs

Challenges

- Something new to learn
- Hard to predict show-stoppers
- Not always plug-n-play
- Trusting the work of others
- Overhead of collaborating
- Funding priorities

How do we tip the balance?

Challenge		Mitigation
Something new to learn		Many examples and documentation
Hard to predict show-stoppers		Engage package developers early
Not always plug-n-play		Submit build issues
Trusting others		Identify or develop tests
Overhead of collaborating		Builds relationships
Funding priorities		Add to the package yourself

How do we tip the balance?

Challenge		Mitigation
Something new to learn		Many examples and documentation
Hard to predict show-stoppers		Engage package developers early
Not always plug-n-play		Submit build issues
Trusting others		Identify or develop tests
Overhead of collaborating		Builds relationships
Funding priorities		Add to the package yourself

The point of open source is to encourage use

Package teams want users to make progress.

If package is missing a crucial feature, ask.

From an SC19 panel: Cutting-Edge HPC is a moving target

- ***What role will reusable libraries and tools play on future systems and how is the role changed from the past?***
  - **Increasing** role … there will be fewer and fewer opportunities to do meaningful simulations "from scratch"
  - Trade-offs between standardization and innovation
- ***What algorithmic and software ecosystem innovations are emerging and needed to enable broad usability of exascale and post-exascale platforms?***
  - We – **users and developers** -- still spend far too much time struggling with incompatible compiler versions, software package versions, gaps in interoperability, etc…
  - Finite resources for testing – we – **users and developers** -- need standardized CI / regression testing!
    - Facilities *must* be involved in this
  - Package managers, container computing, etc are a step in the right direction … but not soup yet
- ***In comparison to today, how will exascale and post-exascale software environments be?***
  - GPUs have raised the bar … easier to be 10x slower than 10x faster
  - More heterogeneous, harder to get optimal performance, increasing need for:
    - Specialization
    - Effective **communication** between specialists