OAK RIDGE National Laboratory

Exploring the use of Novel Programming Models in Land Surface Models

***Ethan Coon*[1], Wael Elwasif[2],**

Himanshu Pillai[2], Peter Thornton[1],

Scott Painter[1]

[1]Climate Change Science Institute

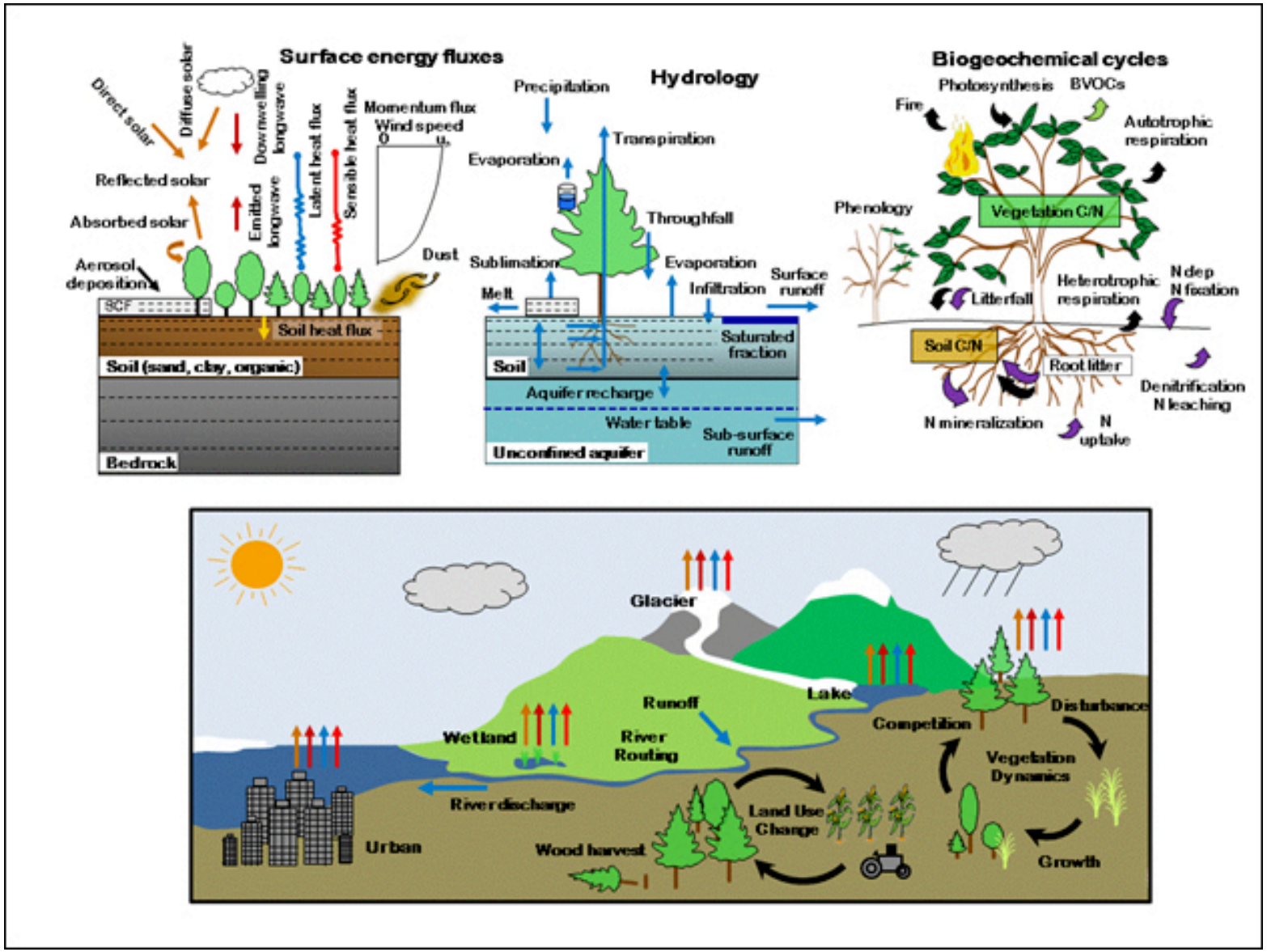
[2]Computer Science and Mathematics Division

Oak Ridge National Laboratory

ORNL is managed by UT-Battelle, LLC for the US Department of Energy

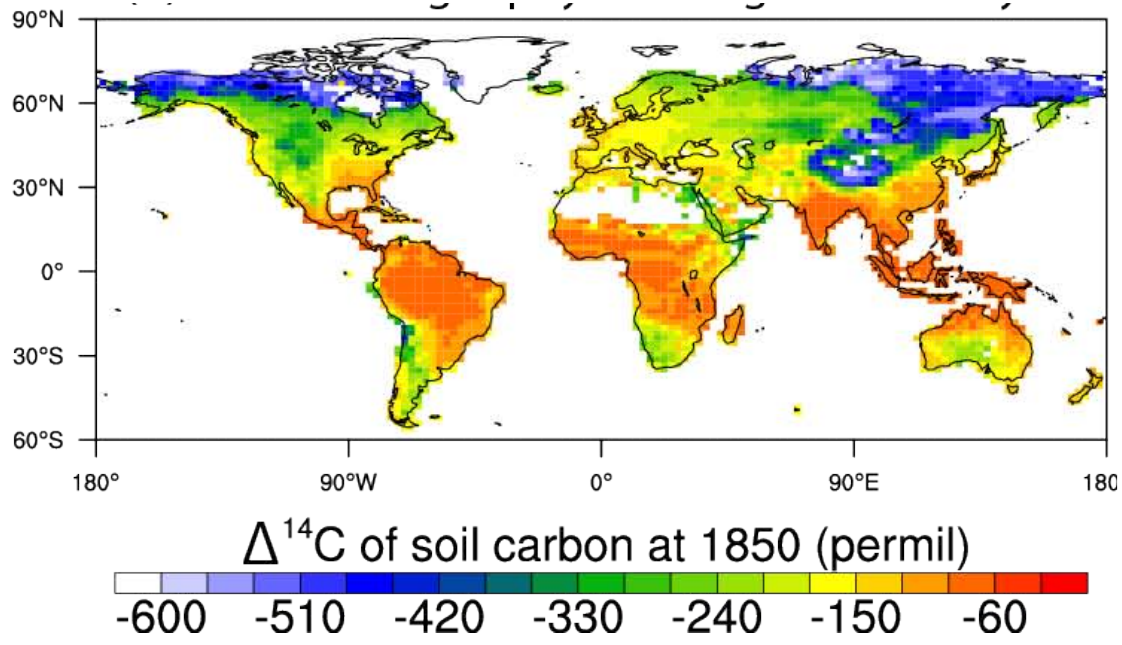
U.S. DEPARTMENT OF ENERGY

What is a Land Surface Model

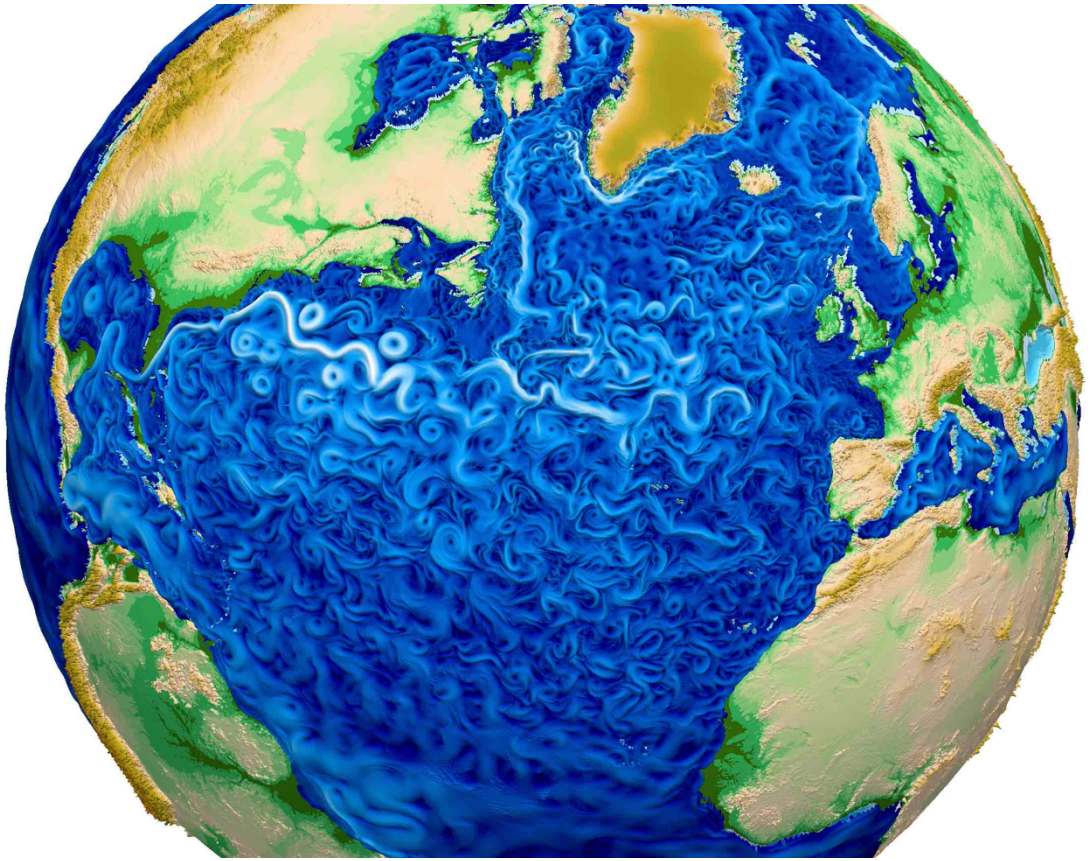


Community Land Model v4.5 Technical Note

What is a Land Surface Model



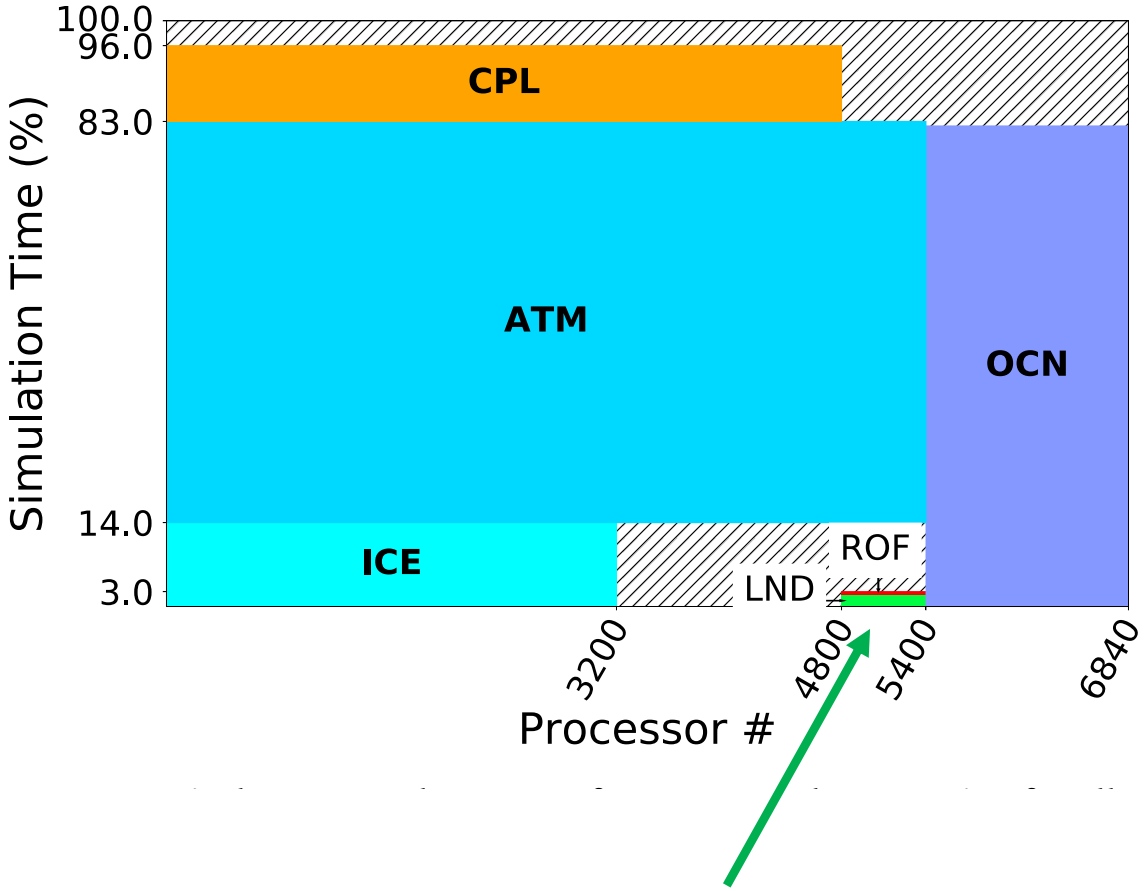
Koven et al 2013



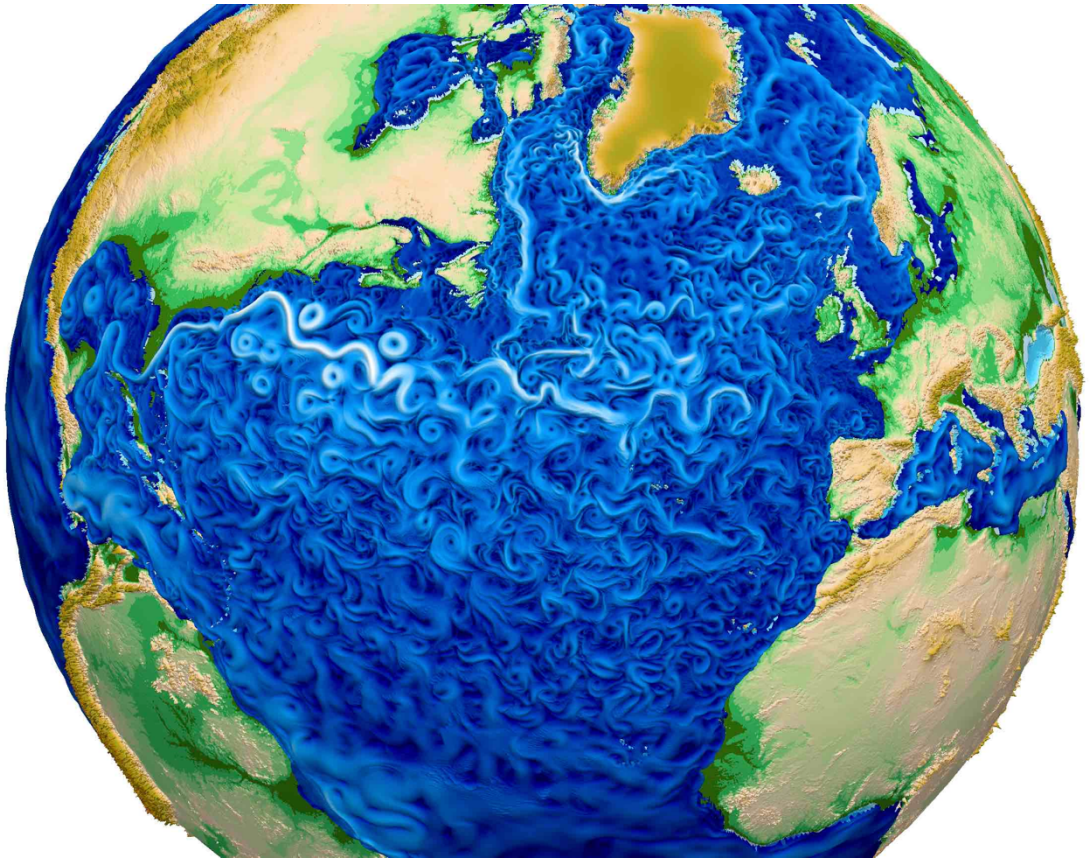
Energy Exascale Earth System Model (E3SM) (Golaz et al 2018)

OAK RIDGE National Laboratory

What is a Land Surface Model

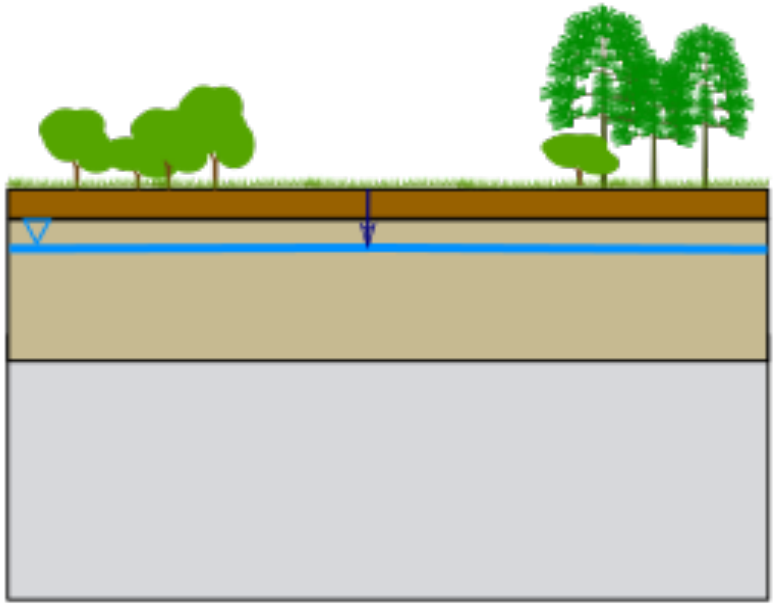


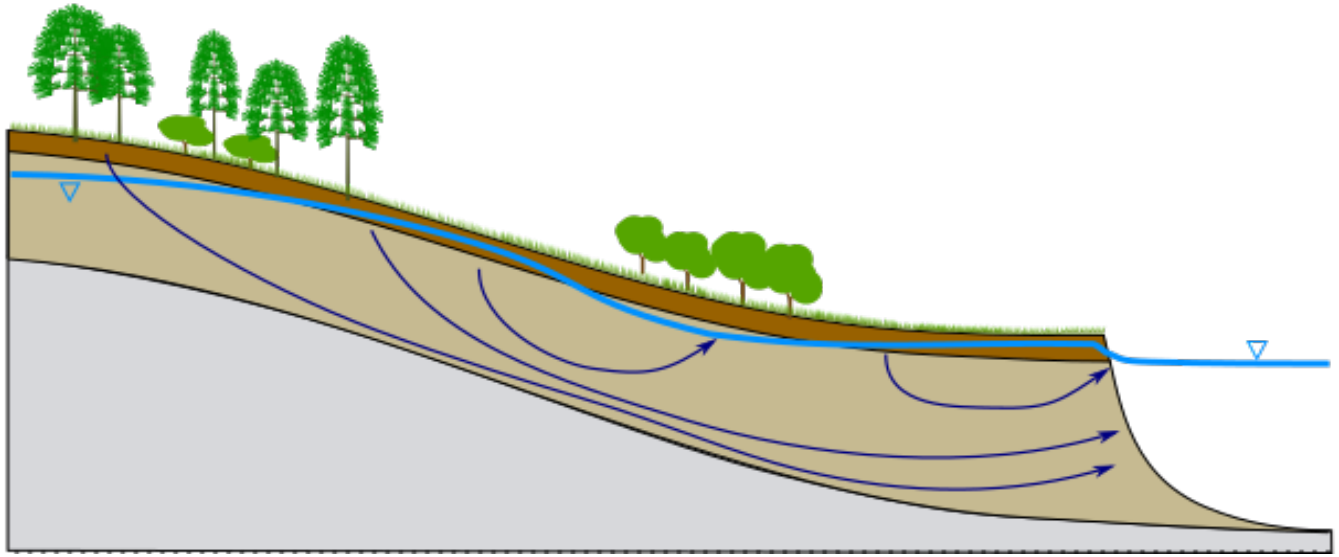
E3SM Land Model (ELM)

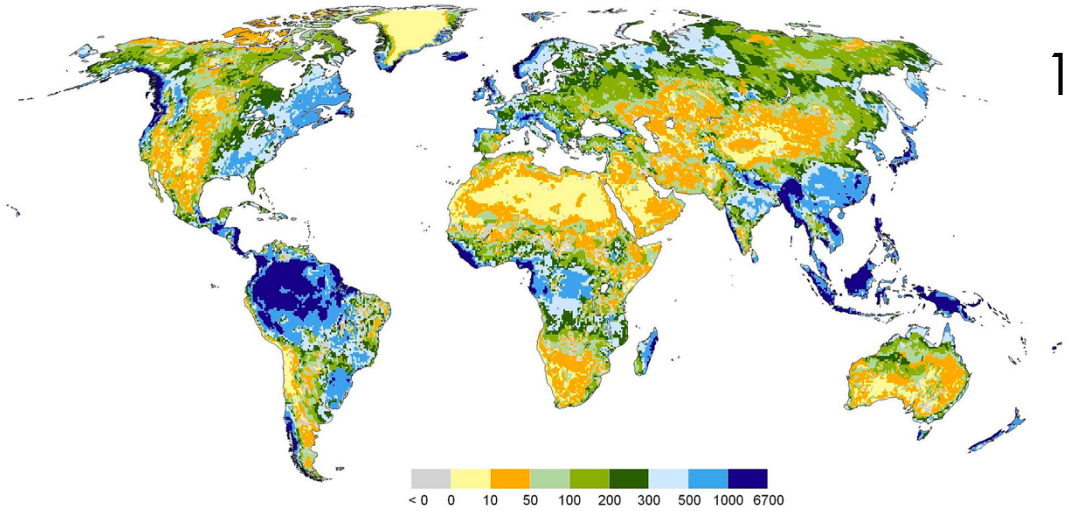


Energy Exascale Earth System Model (E3SM) (Golaz et al 2018)

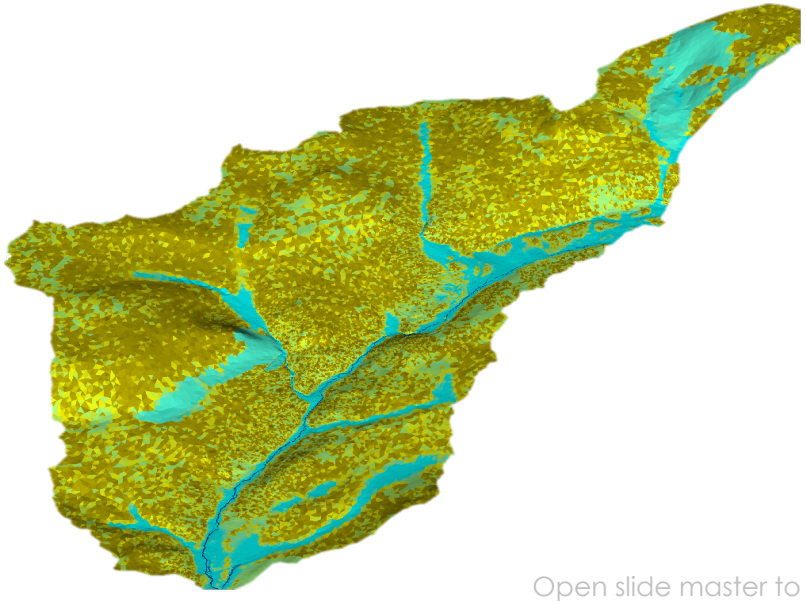
What do tomorrow's Land Surface Models look like?



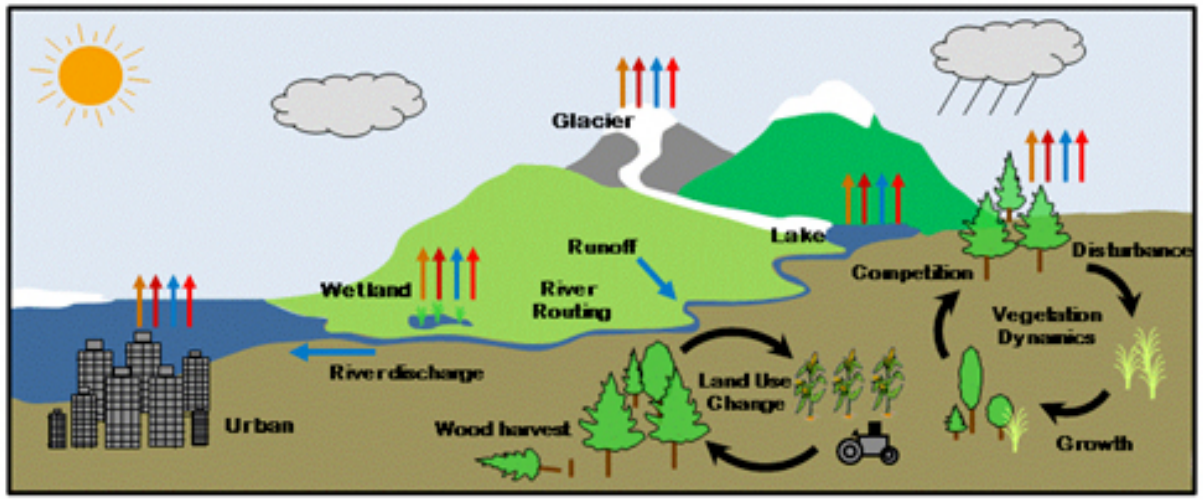


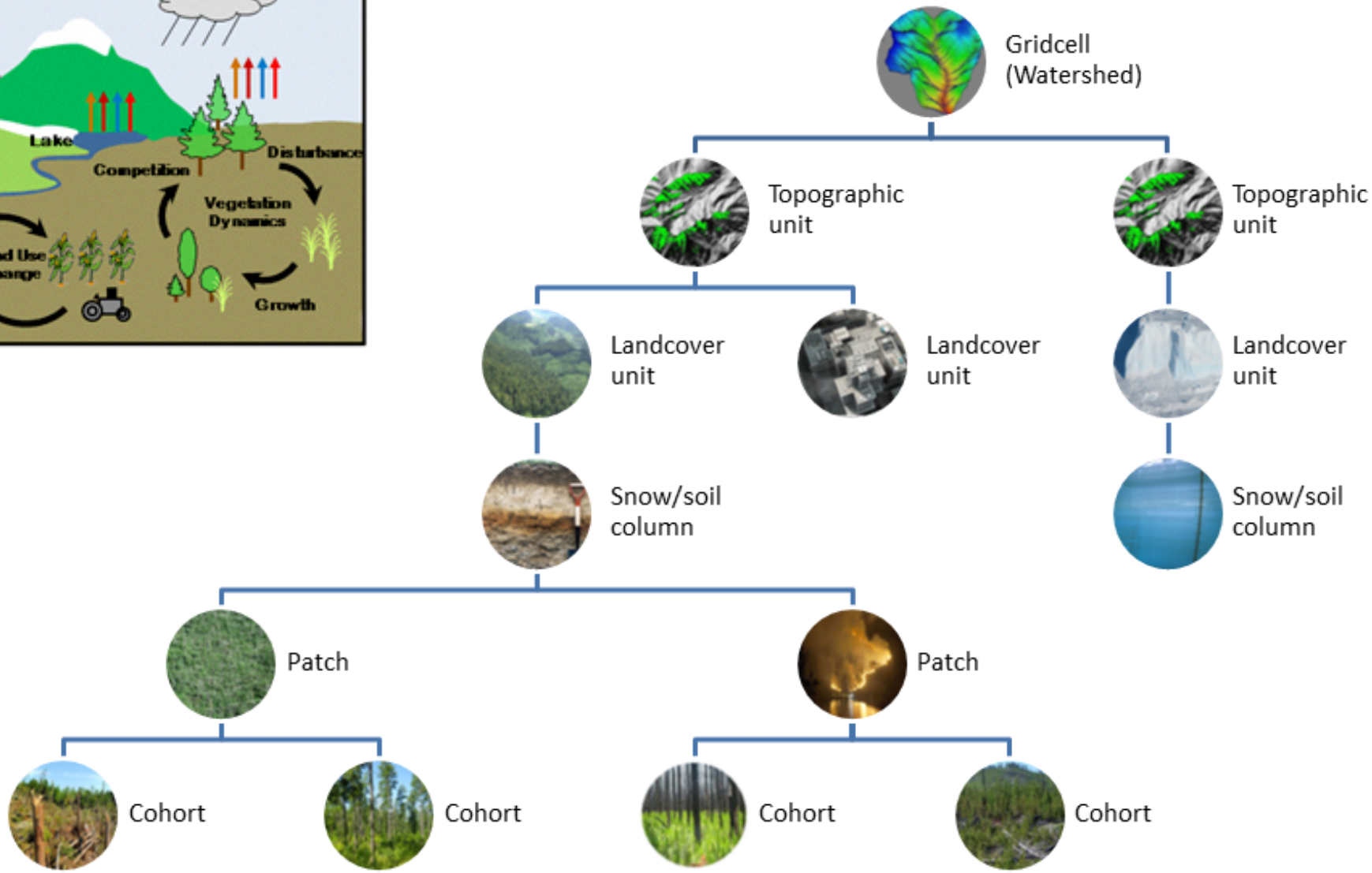


100km → 100m resolution



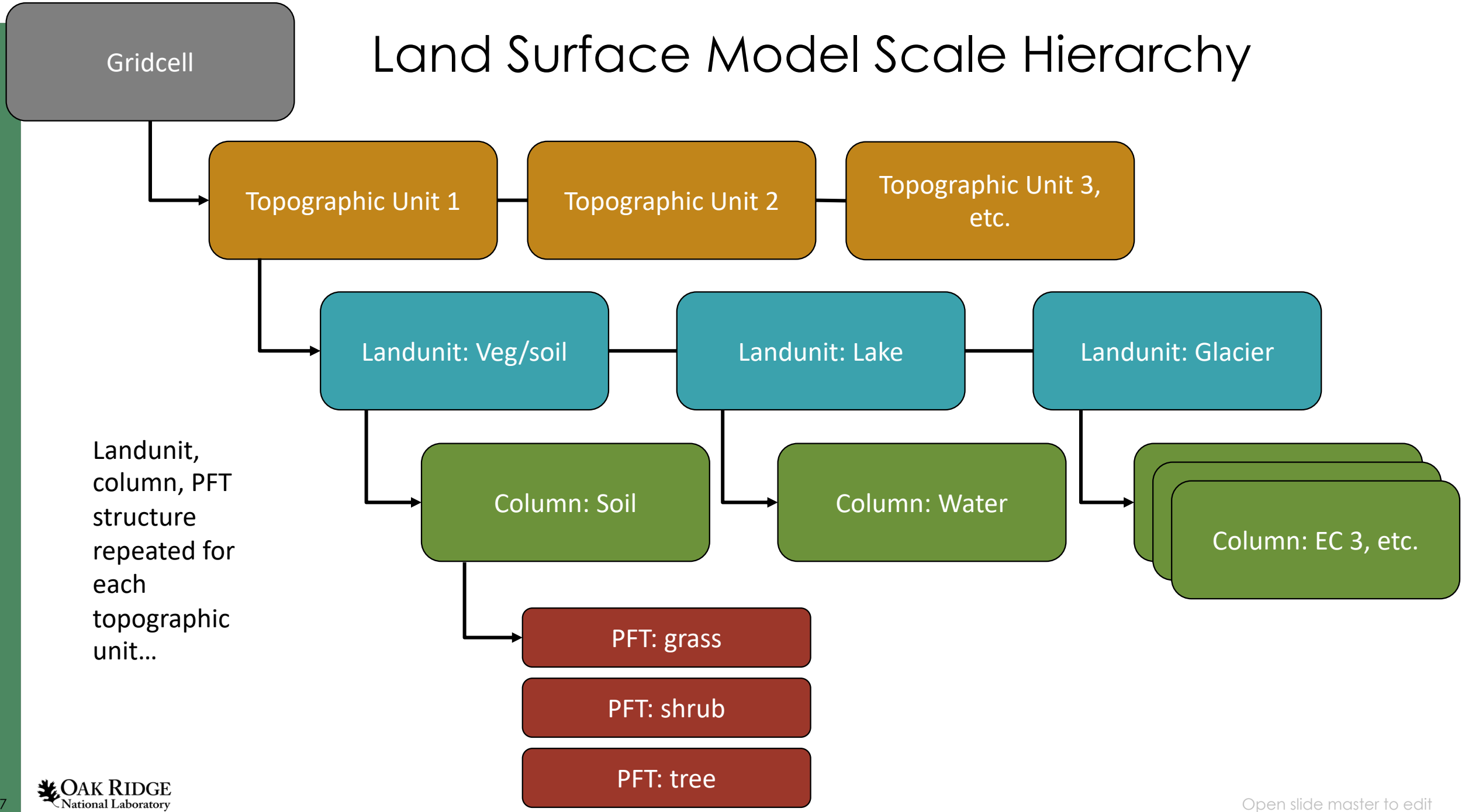
Land Surface Model Scale Hierarchy





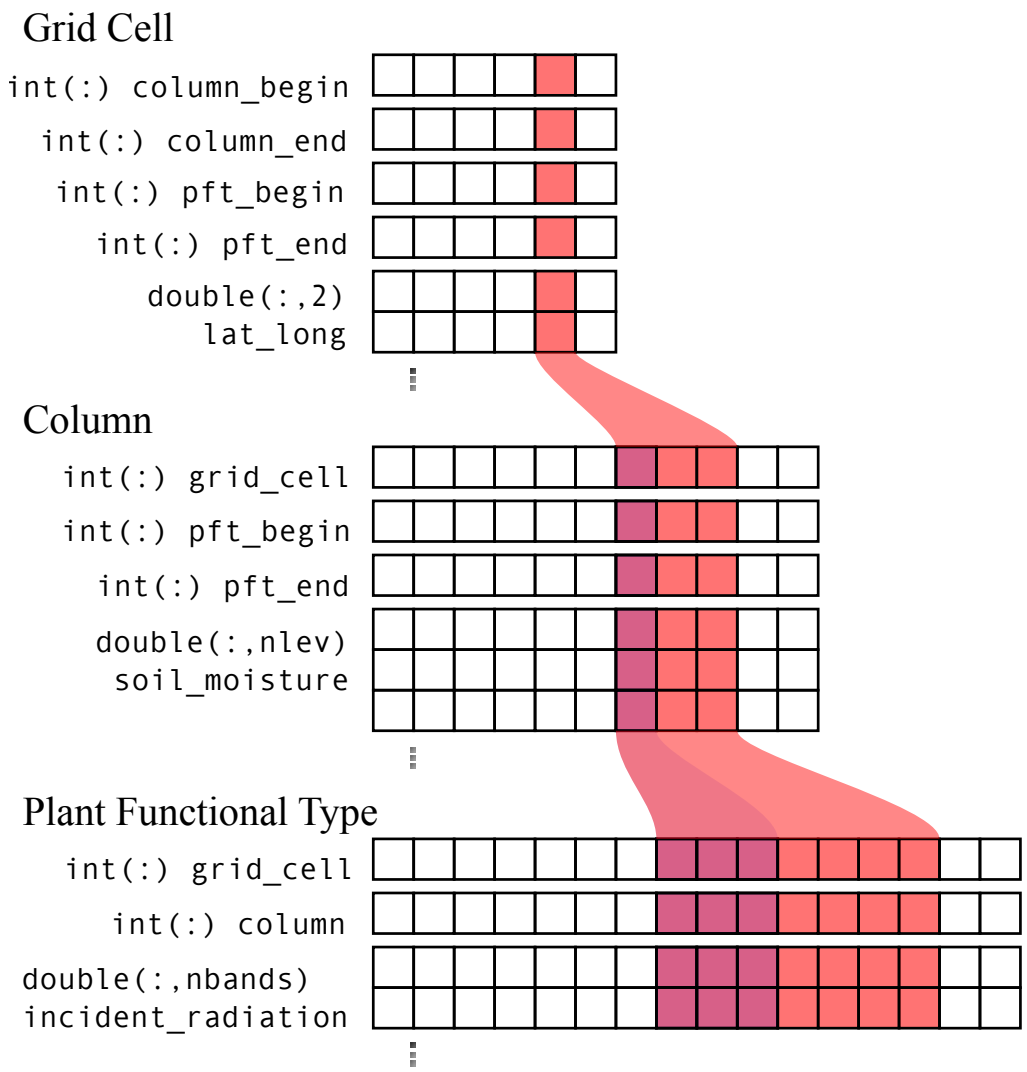
OAK RIDGE National Laboratory

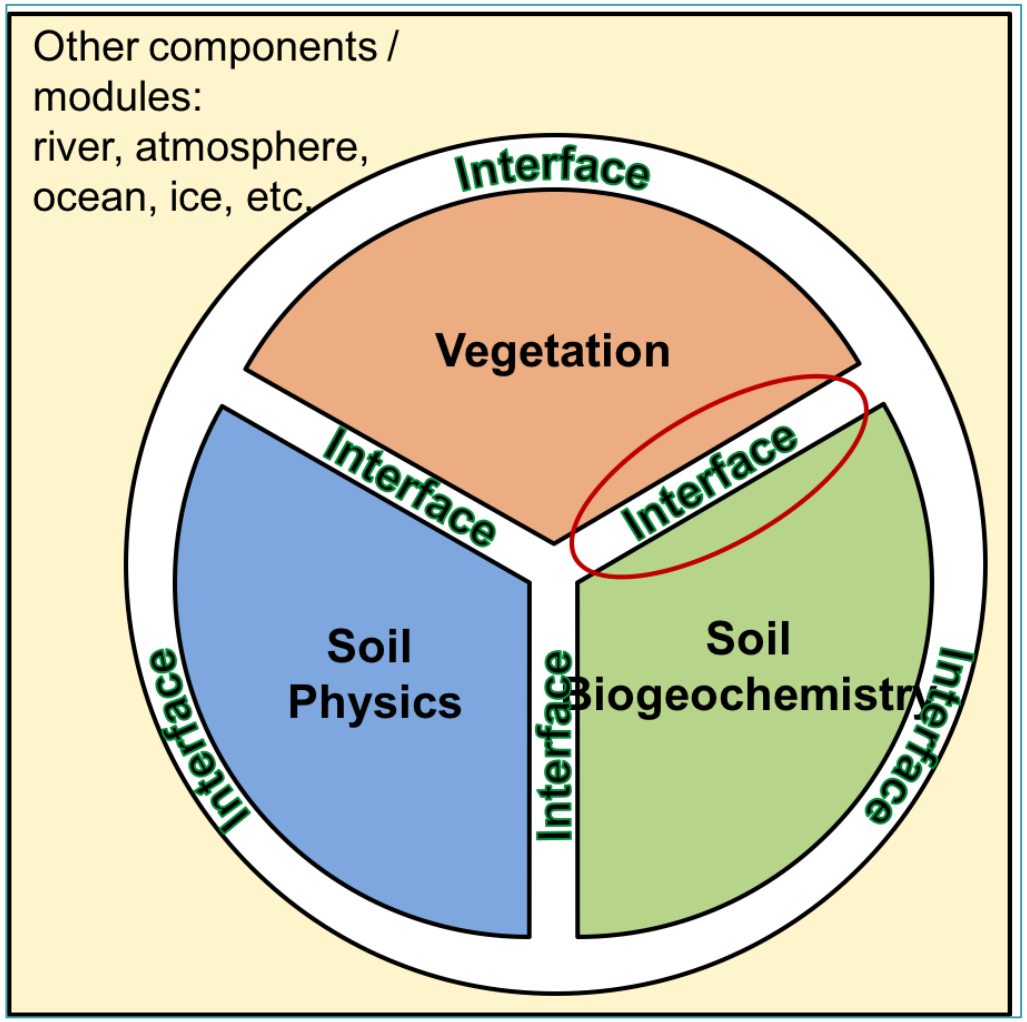
Land Surface Model Scale Hierarchy



Landunit, column, PFT structure repeated for each topographic unit…

ELM Data "Model"





OAK RIDGE National Laboratory

Land Surface Model Development

Shifting software design:

- Historically based on codes in procedural F77 with GLOBAL data
- "Object oriented" F90 with derived data types and subroutines
- Arrays of structs of arrays of structs of arrays of structs…
- Greater encapsulation, componentization, common model interfaces with types

Application programmers

- Team with maybe one or two software/computational/performance people
- Majority of code written by scientists
- Many smaller contributors, all of whom are scientists
- Limited development budget – majority of dollars are for science.

Land Surface Model Development

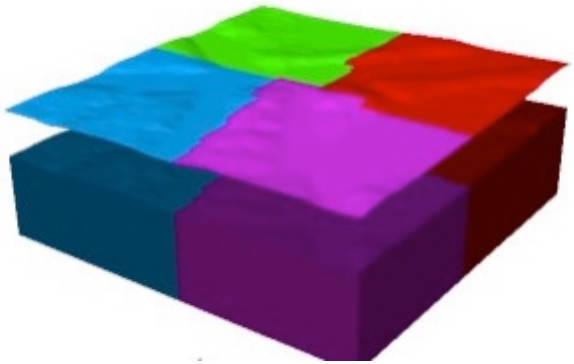
Challenges:

- Disruptive hardware changes: more and heterogeneous cores and less and heterogeneous memory per core.
- Modest software development funding: must reuse codes
- People are expensive: development time is a critical metric of success

Goal:

- Kernels that look like Fortran, or at least not too scary C++
- Some sense of kernels that are portable across a variety of architectures
- Ability to explore a variety of programming models, and combinations of programming models.
- **Prototype a collection of LSM kernels in both Kokkos and Legion**

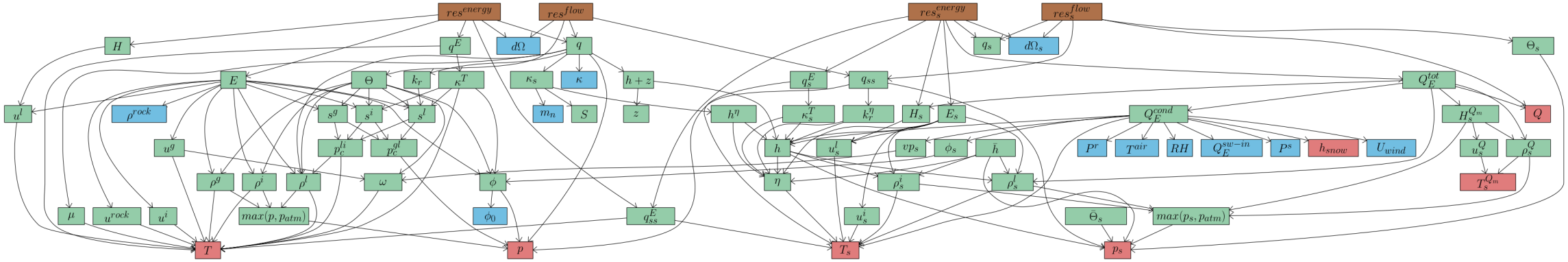
Task parallelism vs data parallelism



Task parallelism can win (relative to pure data parallism) if there is:

- sufficient heterogeneity to make data parallelism poorly load balanced
- sufficient multi-physics to expose new concurrency relative to data parallelism.
- sufficiently distinct multiphysics to reduce communication using tasks

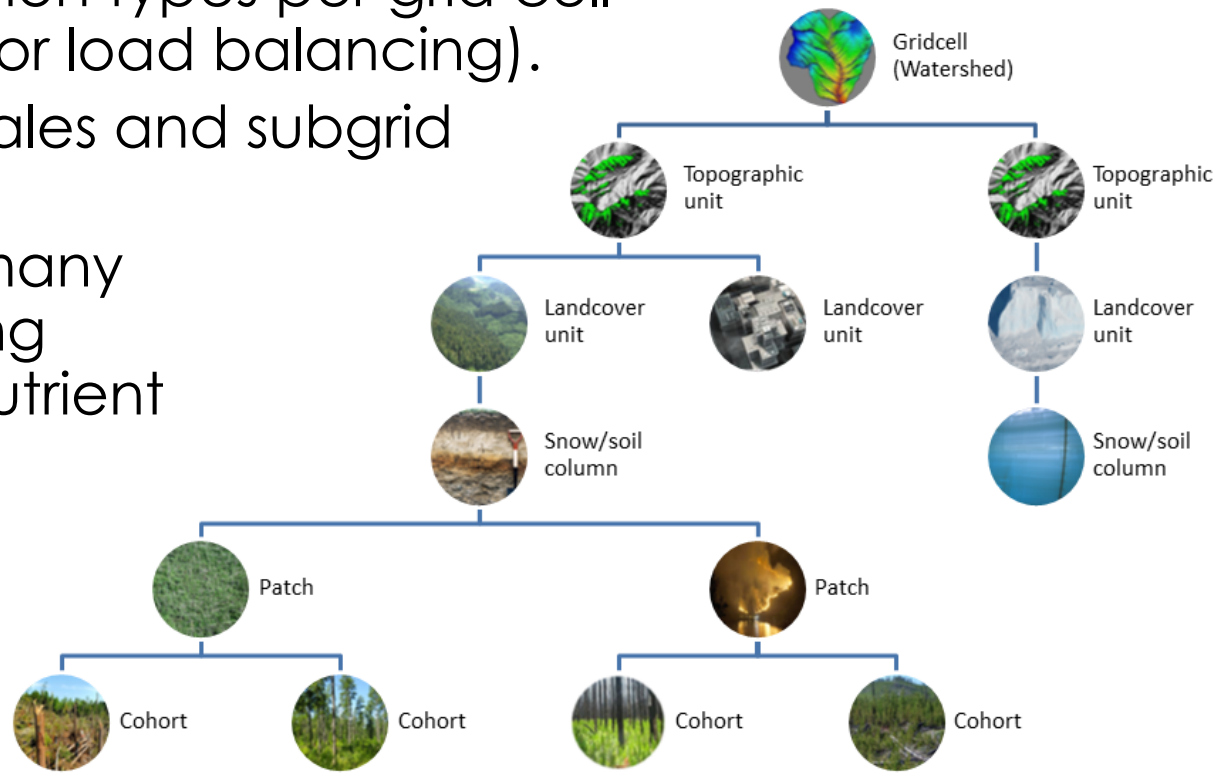
Granularity is key: too small kernels and the cost is too high, too large kernels and there is not sufficient parallelism to exploit.



Task parallel land models

ESM Land Models are an excellent potential application for Task Parallelism

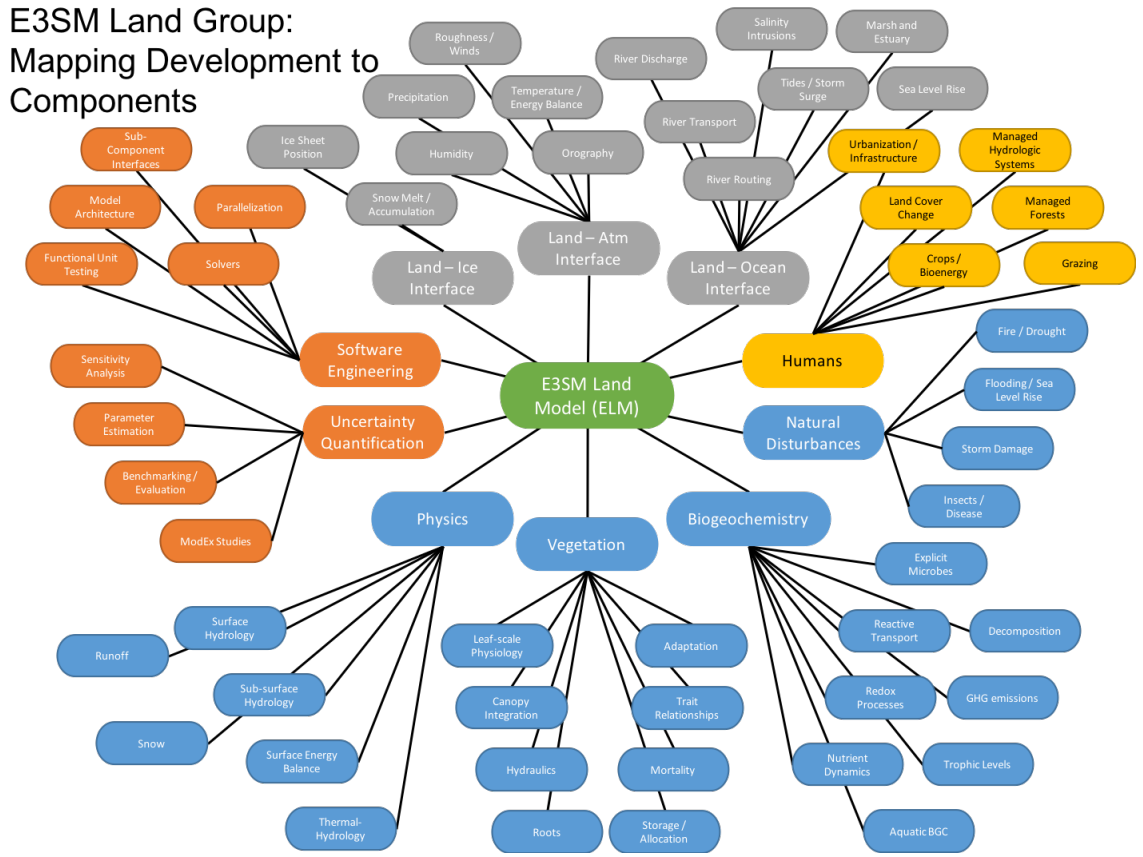
- Heterogeneous physics, e.g. snow physics, lake physics, etc, which are only relevant in certain regions (lots of tasks)
- Heterogeneous performance of the same physics, such as tropical forests with hundreds of vegetation types per grid cell vs Arctic regions with only a handful (poor load balancing).
- Physics represented at a hierarchy of scales and subgrid models.
- Continuous growth of physics, through many complex, tightly coupled processes being added, (e.g. vegetation competition, nutrient cycles, disturbance) means that this heterogeneity is changing as the model evolves.



Task parallel land models

But ELM is a difficult place to start:

- Extremely monolithic
- Tightly integrated model and data (structs-of-lists-of-structs-of-lists-of…). While there is a concept of model components, every component can (and sometimes does) write to anywhere in the hierarchy.
- Multiple ongoing efforts to generalize, abstract, and componentize, which effectively introduce a de facto, intra-model, hub-and-spoke coupler.
- Difficult to expose opportunities for concurrency within the physics



A refactoring strategy for ELM legacy code

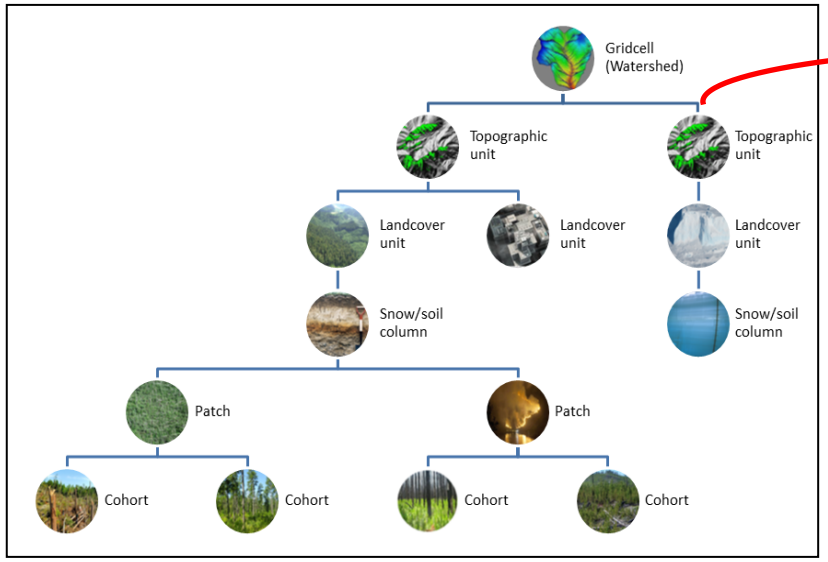
- Developed module-level unit test with test context.
- Refactored Fortran version, introducing subroutines with “tight” interfaces.
- Reimplemented Fortran kernels in C++, templating on array types

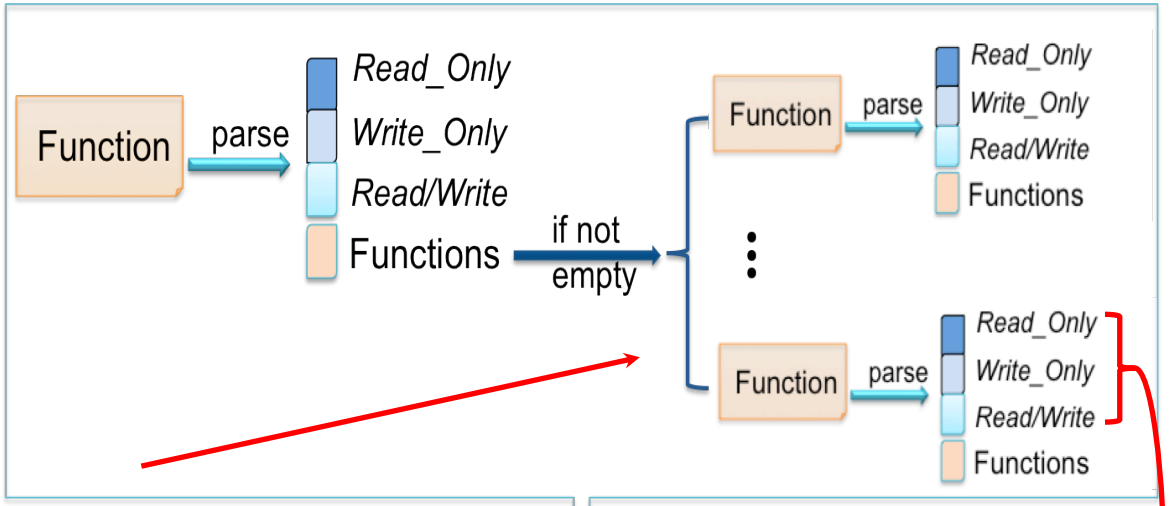
  Assuming value semantics and operator() allows general kernels that can be wrapped as Kokkos lambdas (templated on Kokkos::View) and within Legion indexed launches (templated on Legion::FieldAccessors).

- Reimplemented drivers (outer loops) in C++
- Prototyped both Kokkos and Legion variants calling kernels.

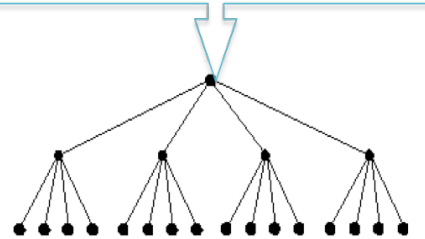
A refactoring strategy for ELM legacy code

Parser from Wang et al, 2014

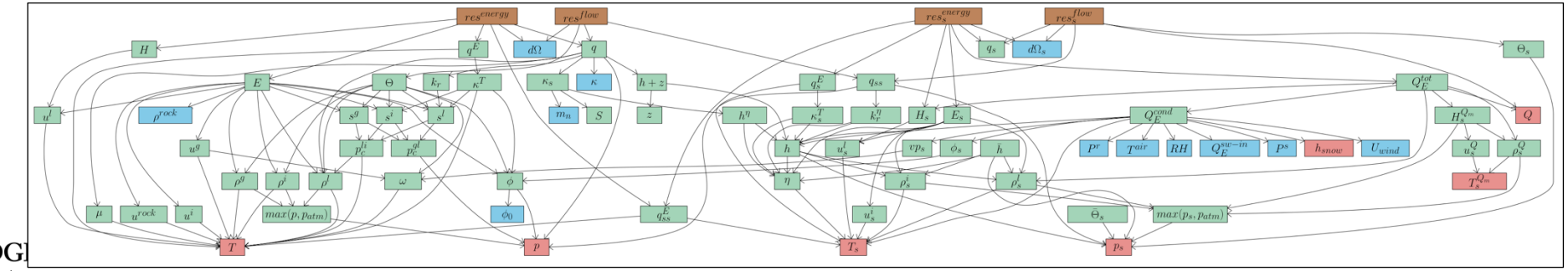




Functions become task kernels



Call Tree



Privileges form dependency graph

Legion exposes concurrency

Even in the trivial example of a single Fortran module, consisting of four kernels executed at two scale hierarchies and a reduction operator between the two scales, **Legion discovered parallelism we didn't realize existed.**

Lessons Learned for Application Developers

- Functional kernels look much more like math. Abstraction of data structure and process physics is a good thing for code readability.
- Functional kernels can be unit tested (unlike most physics code).

- The process of refactoring to kernel-based code is fairly programming model independent (as long as it is C++, but at the level of a kernel, Fortran and C++ look the same).

- New programming models are rapidly maturing, and several have mature, stable code bases with sufficient functionality, documentation, and user support to adopt now.

Lessons Learned for Programming Model Developers

- A little (lot of?) effort at outreach goes a long way. We found both Legion and Kokkos to have:
  - Strong communities with positive user support
  - A large library of tutorials, both in person and on YouTube
  - Sufficient documentation to get started (and be dangerous?)

- Kokkos feels application-developer ready, but could use more complex examples and API documentation outside of the most basic concepts.

- Legion feels incredibly impactful for the LSM community, but realistically isn't usable by application developers without a middle-ware layer (and Regent doesn't play well enough with legacy code to be the answer).

- Don't forget about the silent majority in scientific computation:
  - Trickle-down from user-facilities to institutional clusters
  - Legacy code, Fortran, and years of Object Oriented Design
  - Huge potential for growth into the "small team" development community

Lessons Learned for Collaboration

Still a huge opportunity for “middle-ware” layers and abstractions, e.g. FlecSCI, Arcos, Tpetra, many others, which look to provide abstractions between physics/mathematics and the underlying data and execution models.

Acknowledgements

CANGA

SciDAC
Scientific Discovery through Advanced Computing

U.S. DEPARTMENT OF ENERGY | Office of Science

OAK RIDGE National Laboratory | LEADERSHIP COMPUTING FACILITY