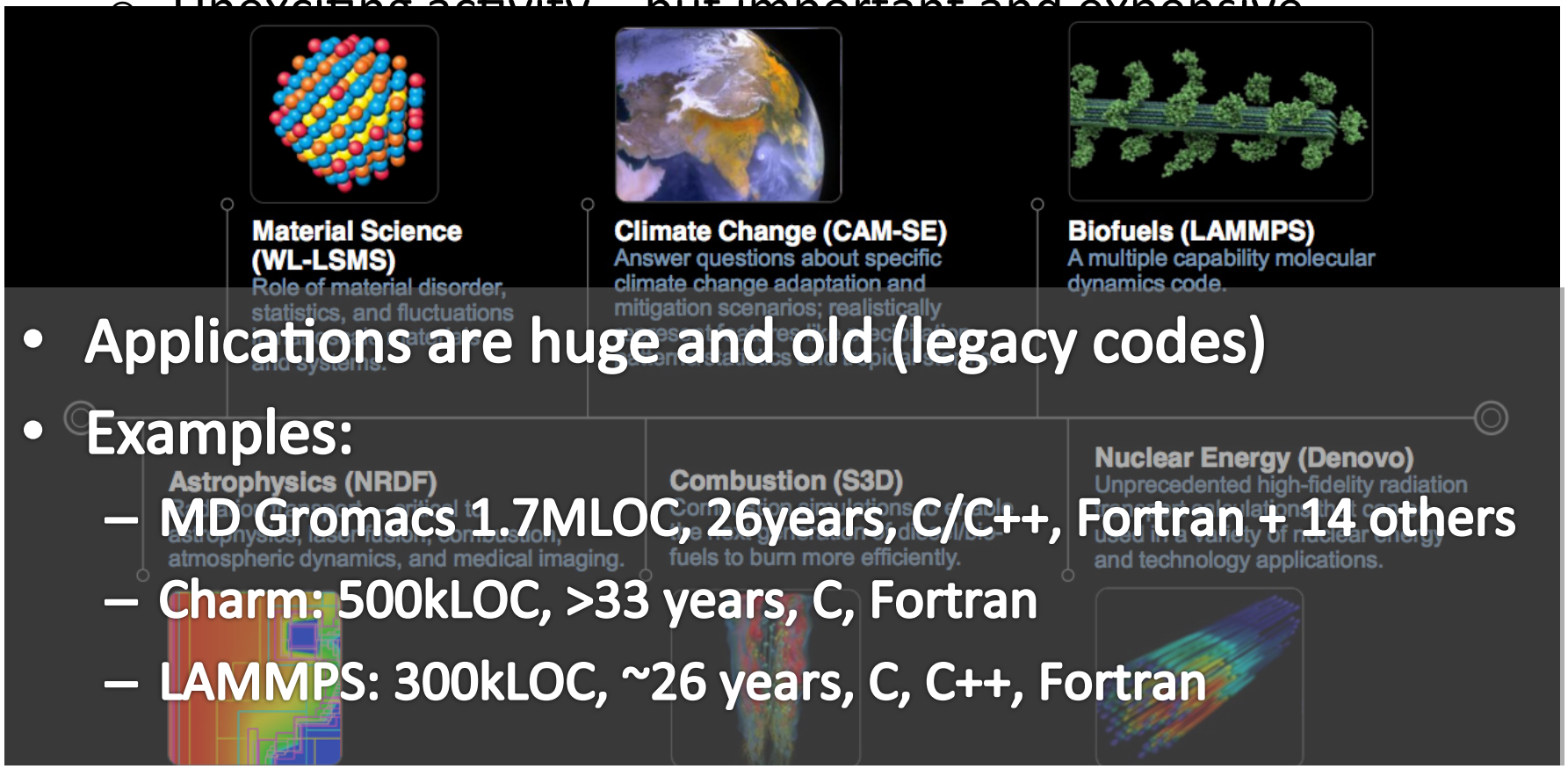
Workshop on Sustainable Software for Science: Practice and Experiences
SC 2013

Development, Support, and Maintenance of Existing Software

Software for Science

- Development, Support and Maintenance – Dimensions:
  - New target architectures
  - Constantly changing middleware (compilers, rt systems, libs)
  - Unexciting activity – but important and expensive



- Applications are huge and old (legacy codes)
- Examples:
  - MD Gromacs 1.7MLOC, 26years, C/C++, Fortran + 14 others
  - Charm: 500kLOC, >33 years, C, Fortran
  - LAMMPS: 300kLOC, ~26 years, C, C++, Fortran

SimGrid: a Sustained Effort for the Versatile Simulation of Large Scale Distributed Systems

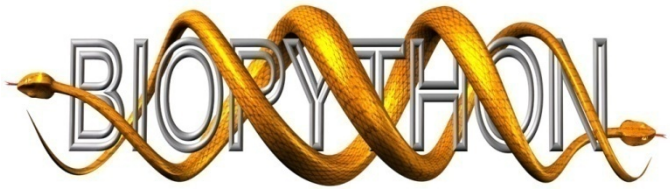
- Experimental software → Versatile Scientific Instrument
  - Comprehensive: for multiple types of distributed systems
  - 3 major versions over 15 years
- Success factors (most research software discarded):
  - Follow needs: Grid, P2P, volunteer computing, HPC, Clouds
  - Recurring funds and support: 15yrs, 20 authors, international
  - Code maintenance: stability/quality, trustworthy results, dev tools: bug tracker, online repository, unit tests, examples
  - Foster user community: mailing list, tutorials, doc, user meetings

Simplifying the Development, Use and Sustainability of HPC Software

- More heterogeneous resources, how to run efficiently?
  - Domain scientists must cooperate with computer scientists and resource providers
  - Clouds are attractive for “unlimited” comp power
- Solution:
  - libhpc: component-based approach + patterns of control, abstract component → concrete components
  - Metadata: software components annotated: capabilities, features, and hardware requirements
  - Situation-specific program: automatic selection of the most suitable concrete components for a given resource
- Abstraction of deployment layer, separation of roles: end-user, method developer (domain expert), application developer, framework developer, infrastructure provider

The Big Effects of Short-term Efforts: A Catalyst for Community Engagement in Scientific Software

- Biopython: bioinformatics software library, 14yrs



- Success story:
  - Open-source code, GitHub, Build developers community, react on user needs
  - External funds: Google Summer of Code from 2009: review students' proposals, interviews, mentoring
  - Short-term effort became long-term effect: GSoC projects become core, grow team members, students benefit: experience, CV, firming research interest
  - In our reputation economy, shift focus from paper to enabling software

Towards Semi-Automatic Adaptation/Deployment of Scientific and Engineering Applications

- Domain scientists cannot handle all porting/deployment details when switch between platforms
- Solution: unmodified applications, autonomous software deployment, abstract metadeployment scripts, situation-specific deployment recipes for required dependencies
- Keep the application unmodified while increasing the range of resources
  - Software "adapters" or shims to match application requirements to platforms
- Deployment knowledge from software and hardware vendors encapsulated in recipes

Scientific software success factors

- Long-term sustainability and usability
  - People factors
    - Community around the code: Open-source code with devel infrastructure (repository, bug trackers, mailing lists, www, etc.)
  - Funding, support and credit
    - Software is important as enabler of research
    - Greater consciousness of software role in papers/results
  - Software approaches
    - Automated mapping of old software to new hardware/paradigms
    - Self-evolving software
  - All of the above dependent on value of software!
    - Measure RoI of scientific software?

Discussion

- ?? && || ##