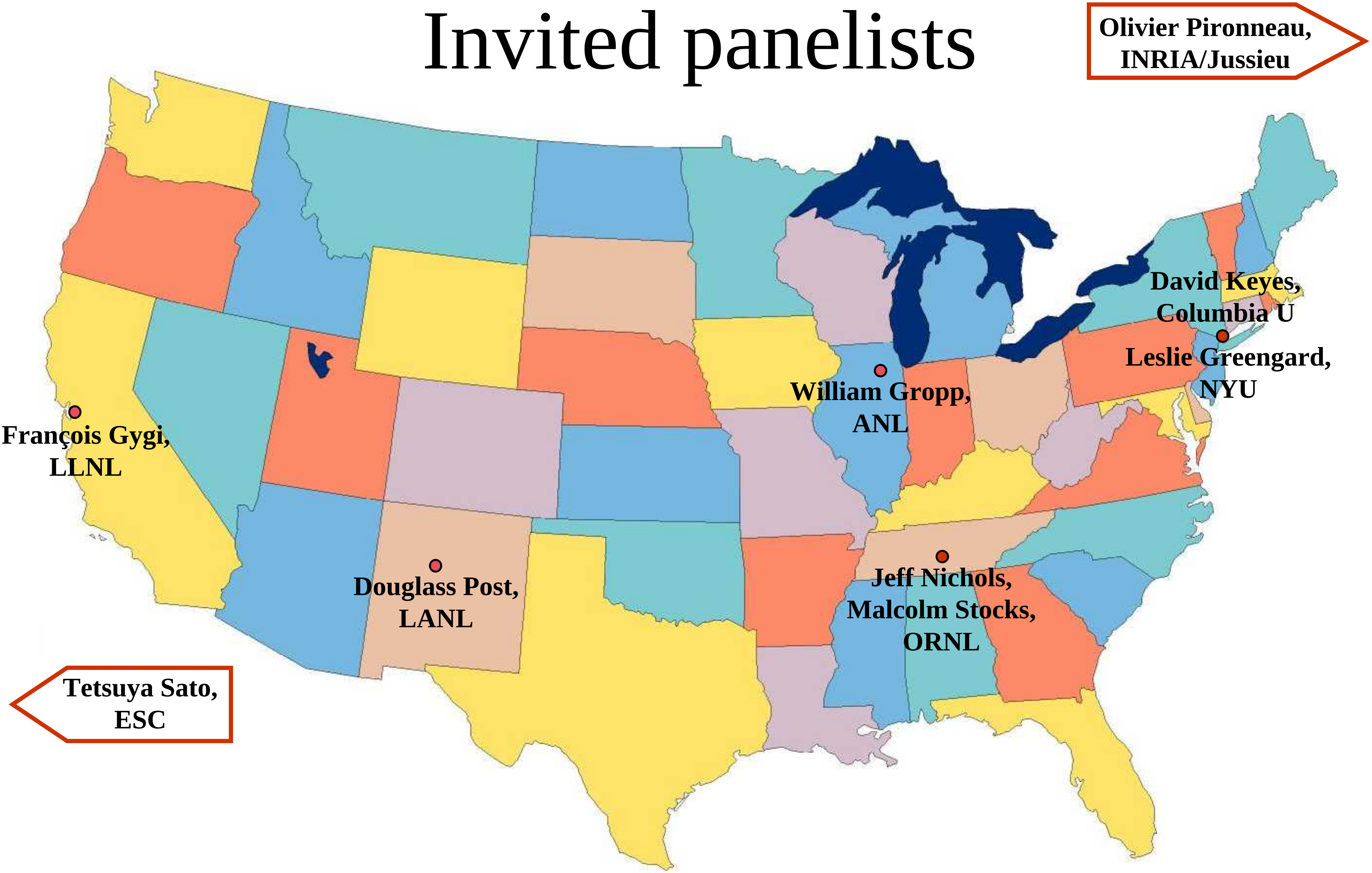
Computational Science Infrastructure
(Charge “c”)

host: D. E. Keyes

Invited panelists



Invited Panelists

- **Leslie Greengard, Courant Institute, NYU**
- **William Gropp, Argonne National Lab**
- **François Gygi, Lawrence Livermore National Lab**
- **David Keyes, Columbia University**
- **Jeff Nichols, Oak Ridge National Lab**
- **Douglass Post, Los Alamos National Lab**
- **Malcom Stocks, Oak Ridge National Lab**

Session IV

- David Keyes (5 min)

  Reflection on what we've heard so far
- Bill Gropp (30 min)

  Hardware and software environments for high-end simulation
- Doug Post (20 min)

  Lessons learned from ASCI software projects
- François Gygi (20 min)

  Current limits of first principles simulations
- Leslie Greengard (20 min)

  Fast algorithms, potential theory, and computational engineering

<Break (20 min)>

- David Keyes (30 min)

  Lessons learned from SciDAC and software from the SciDAC ISICs
- Malcolm Stocks (5 min)

  Computational "end stations" for reactor wall material simulations
- Jeff Nichols (30 min)

  Karaoke

  Open microphone

Personal remarks

- Thanks! We “spies” have enjoyed the free “professional short course” in materials simulation and reactor environments
- *We’re still missing some vital information that we need to write our chapter!!*
- Our presentations will (we hope) draw some of what we’re missing out from you, while also communicating some useful experience (and URLs) back to you
- *Last 30 minutes of open discussion is intentional and sacrosanct*

Personal remarks, cont.

- *We've all done science, too, before going over to the dark side*
- We don't know what all your "nails" look like, but we have hammers ☺
- *We, and many of our colleagues, approach collaborations with materials scientists with great confidence and zeal*

Personal observations

- You seem to need:
  - Stiff integrators
  - Implicit solvers (mainly for potentials?)
  - Force summations (for DD)
  - Sensitivity analysis, uncertainty quantification
  - Large-scale data bases, visualization, data mining
  - Remote data, platform, and instrument access
- *You have some highly relevant experience in programs like PERFECT*
  - *Mixture of simulation, experimental validation, and community training*

Personal observations

- In some ways, you're like everyone else: "better" means
  - Bigger
    - Avogadro's is a big number
    - BCs need to be less intrusive
    - Diluent factors need to be smaller
    - Interactions between multiple cascades important (?))
  - Faster
  - Cheaper

Personal questions

- *Do you guys have important community codes? If so…*
  - *On what do you run them?*
  - *What is their parallel programming model?*
  - *In what are they written?*
  - *Are they open source? Version controlled?*
  - *What are their storage requirements?*
  - *What are their complexity bottlenecks?*
  - *What are their performance bottlenecks?*

Personal questions

- Parallelizing a code involves:
  - Decomposition into (generally balanced) concurrent tasks
  - Assignment of tasks to processes
  - Orchestration of processes (communication, synchronization, replication)
  - Mapping processes to processors
- How are your workhorse codes doing this?
- With PDE-based codes, the first question leads immediately to answering all of the others