Questions for X-Stack kickoff meeting (from SLEEC group)

1. Exascale systems will undoubtedly require that much code be rewritten to deal with the different challenges of exascale (lightweight nodes, etc.) Will this rewriting require a fundamental re-think of how we write large-scale programs? For example, even if we still program with MPI, any sort of global communication will be virtually impossible, and even kinds of "local" communication that can be used effectively might change.
   (a) How will these kinds of changes affect the structure of our applications?
   (b) Are these constraints the kinds of things that could be "hidden away" in high level abstractions, leaving it to compilers and runtime systems to map non-exascale-ready idioms to more efficient counterparts? Or should the programming model directly expose the low-level constraints?
   (c) Should these constraints be directly enforced in a programming model? Or should they be observed by convention?
   (d) Will these constraints bleed over into the design of algorithms and methods that we can effectively run at exascale?
2. Many of us (our project included) think that domain-specific approaches are the best way forward to allow programmers to efficiently write exascale code. But history has shown that programmers are loath to give up the languages and libraries they are used to.
   (a) What will be different this time? Will domain-specific approaches become more prevalent?
   (b) In modern applications, there may be multiple domains, at multiple levels of the application stack, interacting. How will domain-specific approaches compose with one another?
   (c) Should we be focused on domain-specific approaches as an aid to productivity (e.g., machine-generated code)? Or as an aid to optimization (e.g., semantics-driven optimization)? Or both? Do we need different approaches to the two? How do they interact with each other?
3. Effective exascale programs will require coordination between applications and runtime systems.
   (a) What kinds of capabilities should we expect from runtimes, freeing languages and compilers from having to provide them? What kinds of capabilities *can* be provided by a runtime? What kinds of things must be provided by other layers?
   (b) Can there be some sort of standardized runtime interface layer, providing a single target for languages and compilers, rather than having to specialize for different systems? What would such an interface look like?