Applications: What are the applications that motivate future systems? What advances in algorithms and programming systems will support these applications?

Interactive High Performance Computing: Given the push towards an instrumented world—characterized by the pervasive integration of sensors and computation—the kinds of analysis, simulation, and applications associated with traditional high performance computing will likely become integrated with interactive workflows. These computations will benefit from large scales of computation applied in bursts. As such there is a need to unify and integrate HPC techniques into more general purpose event and transaction driven programming models. In such a model, a programmer can construct software in a web service or UI like event model that triggers HPC like computation on bursty streams of data. For example, consider a streaming 3D medical reconstruction application that uses bursts of thousands of cores to process a fetal MRI study as it is being conducted. Its clinical utility lies in its ability to elastically produce a high-fidelity reconstruction from the raw noisy motion blurred input images in a sub-second time frames. Such applications will blur the line between traditional large scale computation and general purpose interactive computing.

Both scale and elasticity challenge the abstraction boundaries which separate system details from algorithm and programming language design. In order to achieve the necessary performance software and associated tools become tightly bound to specific hardware. This is a serious threat leading either to hardware stagnation or constant redesign of algorithms and rewriting of software with each new hardware product.

Machine Learning: The use of machine learning and inference are likely to be critical features of future applications given the ever increasing access to data and maturing of Machine Learning techniques. Similarly, dedicated neuromorphic hardware holds the promise of highly efficient support for such computation. Attempting to exploit this type of hardware for modern applications, however, demonstrates how deep the divide is between general purpose programming and the construction of, for instance, a deep neural network for a given application. Applications require custom design and learning expertise to correctly build and utilize learning techniques. These skills are somewhat orthogonal to the training required to write and optimize applications in a traditional programming language. Tools and techniques that bridge this semantic gap will give programmers of all skill levels, access to these promising technologies. It is worth while seeking new ways to
integrate learning into our commodity infrastructure.

Scientific workloads  Scientific workloads such as physical simulations of molecular dynamics, which have major applications in drug discovery, material design, and climate modeling deserve our focus. These applications often require sequential step-wise discretization of partial differential equations. While it is true that it is possible to manually parallelize many scientific workloads, such approaches are fraught with difficulty, and parallel implementations are only available for the most important workloads. Parallelization packages require years of tuning and often need to be re-tuned for new architectures because of the need to take into account not just the parallel algorithm, but also load balancing, communication mechanisms, and data locality management. It is worth looking for approaches that can adapt such application software to new hardware systems.

It is also worth considering automatic approaches that ideally eliminate the algorithmic and software engineering efforts to create a scalable solutions for scientific computations that are known to be difficult to parallelize. For example, many statistical data analysis tasks in astronomy, neuroscience, statistical genetics, and the social sciences require difficult numerical integration via Markov chain Monte Carlo (MCMC). The intrinsically sequential nature of Markov chain simulation means there are few generally applicable parallel implementations. Tackling such problems will likely require significant foundational work.

Systems: What hardware architectures and distributed systems will support future applications? What challenges do we face in design and management?

There is a risk of moving further and further down the road of unbalanced, heterogeneous systems where there are large communication cliffs between the various scales of the system resources. For example, dense multi-core processors that support very fast internal communication can become computationally limited by I/O bandwidth to storage resources or other, physically separated multi-core processors or accelerators. Such hardware designs bring with them subtlety and complexity that can make it very difficult to write generic software.
Technologies: What emerging technologies will change fundamental assumptions in hardware and software? What constraints disappear? What challenges arise?

**Short Term:** As I/O speeds start to balance out assumptions in general purpose system stacks, a traditional organization will no longer be appropriate. For example, data centers will be able to deploy SSD’s whose Read/Write bandwidth can saturate a 10Gb Ethernet link. This will mean that there will be greater utility in locally attaching SSD’s rather than solely using storage nodes or storage racks as NAS. As is often the case, there will be a tradeoff to be made in exposing the disparity to the applications or trying to avoid exposing further complexity. Similarly, as I/O speeds increase, the computational time that system software has for processing data received from I/O connections must decrease proportionally. Interrupt processing efficiency becomes critical to ensure that applications can reap the benefits of the faster devices. Specifically, traditional OS interfaces and implementations were not designed for high speed asynchronous applications. Process and thread scheduling, and old event primitives, are thereby mismatched to the needs of applications.

**Medium Term:** Further increase in heterogeneity become commonplace as new technologies such as 3D stacking, persistent ram, and function specific accelerators—as well as intrinsic power–performance limitations like Dark Silicon—change the landscape of hardware design. Physical attributes will become more subtle, complex, and device-specific not only defeating the notion of general purpose systems but also challenging our use of abstraction to encapsulate and manage complexity. As many have observed, to reap the advantages of explicitly programming concurrent hardware means that all layers and software must carefully be matched to avoid fatal accidents. These problems are even present in existing systems; for example, one contended cache-line associated with an OS performance counter can dramatically degrade an entire software stack’s performance. As hardware complexity necessarily increases in reaction to the end of CMOS “free-lunch” power–performance gains, back pressure from this complexity can undermine potential improvements.

**Long:** As technologies such as sub-threshold and neuromorphic chips become mainstream, we have the opportunity to more radically attack the von-Neumann bottlenecks. The software challenge, however, will
only be further exacerbated. Computing has the danger of losing human comprehensibility and generality. Logical, sequential models give way to paradigms where humans must reason about statistical structure and inference in order to understand why a system behaves the way it does. While humans may be poor practitioners of such statistical reasoning, dedicated neuromorphic hardware can be designed to perform such inference extremely efficiently. Simplified programming models coupled with hardware support for autonomous acceleration seems to be a valid middle-ground between high-performance, hardware-tuned systems and generality of application description. The inclusion of aforementioned technologies in computing and system architectures is required to perform program introspection in a realizable way from a power–performance perspective.

Methodologies: How should we perform interdisciplinary research that spans applications, systems, and technologies? How should abstraction layers evolve?

Encourage small teams of experts, that can work closely over a five to six year period to construct complete prototypes. Fortunately, many of the gains of applications, systems, and technologies can be disparately realized. For example, machine learning introspection can be directly applied to executing binaries while dedicated neuromorphic accelerators can be used for explicit machine learning feedforward/prediction or feedback/learning tasks. These gains, while beneficial in their own rights, can be integrated in larger systems to improve the applicability of specific technologies and potential power–performance improvements. We envision initially distinct abstraction layers evolving by discipline, but through close informational meetings during these disparate evolutions these portions of these contributions can be directly integrated. Over the five to six year period, the technologies will be tightly integrated into one seamless architecture.

Risks: What are the risks that threaten the success of XPS research directions? How do we guard and hedge against these threats?

One risk is the possibility that the theoretical limits of parallelizable problems will be a barrier to the effectiveness of emerging technologies and that the many p-complete problems will not be amenable to these new methods. The theoretical results imply that stronger algorithmic techniques will be
needed. It will be necessary to use probabilistic techniques together with new learning methods that allow knowledge acquisition of the distribution of the problem’s input set. Artfully using this distributional information may enable us to effectively improve parallel performance on highly probable inputs even when a problem is hard in the worst case. Additionally, underlying computational structure may be due to existing language limitations that can be more easily, and generically, described with language improvements. Nevertheless, structure in data exists for many useful applications tied to real-world inputs (e.g., video processing) and software–hardware techniques focused on teasing out such structure can hedge against potential power–performance slowdowns in the coming decades.