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

Many future, and increasingly current, applications are irregular in control flow, data-flow, data-layout, and communication. Physics simulations are increasingly based on meshes and multi-resolution data-structures with per-element time steps and periodic retriangulation. Machine learning, graph analytics and many life-sciences algorithms are likewise based on sparse matrices, graphs, sets, and trees.

Programming systems are not well suited to such algorithms. Classic systems are based around loop analysis and block communication. Irregular algorithms are not analyzable at this level. Programming systems must, and increasingly are, providing more irregular and data-driven abstractions: e.g., tasks, fork-join, futures, etc. Programming systems which successfully support these algorithms will allow specification in a high enough and semantically rich enough level that the compiler can generate drastically different implementations tailored to the various heterogeneous compute and communication resources available. Data-driven, data-structure aware programming models for irregular algorithms show promise in achieving multi-device performance from a single source.

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

Systems will support increasing levels of communication and runtime adaptation to allow applications to maintain performance and reliability in the face of dynamic resource availability; changing power, thermal, and bandwidth envelopes of the compute system; and dynamic, data-dependent behavior of algorithms and sub-systems.

As the availability of transistors outstrips power, increasingly heterogeneous systems will be standard. The design and management of future systems must be significantly more holistic than at present. Designs must be based on abstractions that support the application programmer. The challenge of future system design is to support uniform management of the components of a system and to provide at the programmer level a coherent view of computation. The systems of the future must allow a programmer to write a program - not a collection of components written in different languages with different restrictions with different communication layers and visibility.

(c) Technologies: What emerging technologies will change fundamental assumptions in hardware and software? What constraints disappear? What challenges arise?

Dense and fast non-volatile memories will change the fundamental notions of program state. This will impact everything from programming languages, runtimes, recovery and reliability, storage abstractions, and communication abstraction.

High-speed local interconnects, from silicon photonics for example, and stacked memories change the bandwidth, coherence, and latency trade-offs when building a system, especially heterogeneous systems.
Communications must become an first-class citizen in such systems. Emerging applications are poorly suited to using either bulk-transfer APIs or coherent memory as their primary communication mechanism.

Approximate hardware will allow a new dimension in reasoning about the sufficient utility or goodness of a computation. Approximation already exists at the algorithm level, say by modifying convergence criteria, and at the representation level, using shorter types, for example. Introducing approximation at the basic computation level will increase the avenues a programmer may achieve a result within a given computation or power envelope.

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

Evolving an abstraction layer, rather than just replacing one immature one with another, requires significant time and effort. From a systems and runtime-time perspective, many researchers make "new" abstractions and systems which are similar, but incompatible, with other systems to incorporate their one new (analysis, compilation, runtime, memory layout, communication, etc) trick. The community mush overcome the not-invented-here syndrome and work on systems they didn't write from scratch. No evolution will happen if all the offspring are killed before any of them can reach maturity.

Besides time, to evolve systems we need pressure. Application-level research is well posed to provide that. If systems can mature to the point of being usable for applications, applications can provide new and unexpected challenges. Likewise, mature systems will evolve to meet a demand for scaling, support for heterogeneous systems, performance, and programmability which immature systems won't face.

By way of analogy, compiler research happens mostly on a handful of widely developed, mature, industry-strength code bases. It is hard to even evaluate work performed on toy compilers, it is unclear if the results are significant or just a by-product of less robust optimization and analysis.

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

Opaqueness of the causes of emergent behaviors in large parallel systems will greatly impact the usability and management of large parallel systems. For example, application programmers must be able to debug performance and do so at a layer of abstraction close to that which they use. Abstractions guide thinking. To require an application programmer to bypass too many layers of abstraction to understand a system sufficiently to solve a problem is a failure to support the whole lifecycle of application development. Support must be provided to allow more integration of layers in the software stack and more testing of complete stacks or sets of abstractions by application-level programmers targeting production scale problems.

Lack of open, experimental systems limits the speed of discovery and validation of new ideas. For example, It is non-trivial to have sufficient documentation, cryptographic signing keys, etc to replace firmware on a GPU or a NIC. Significant efforts should be made to encourage vendors to supply open, expandable systems. XPS is
targeting "new and visionary approaches to re-evaluate and possibly re-design the traditional computer hardware and software stack". Being able to validate ideas and abstractions at scale on real problems requires more than simulation. A cost-reasonable approach for many levels of the stack is to use existing hardware in new and unexpected and unsupported ways. This requires more open hardware than is currently common.