Slide 1
Allow me first to indulge in reflections upon the culture of hero programming, which, in my view, needs to change in order for us to achieve our vision for the future of high performance computing.

For as long as I can remember, we have all been hero programmers.

Most of you are too young to remember using the IBM series front panels to control and debug programs . I sure felt like a real super-hero when I was programming a Fortran simulation model that I ran on an IBM-1130 Model 1 with 8KB of memory, yes Kilobytes! and using its front panel to control and debug the program execution. No debugging tools: I had to read on the computer panel the location of my program in hexadecimal and try to figure out if it was doing the right thing…
It was actually lots of fun!

Maybe you remember assembler programming for the M68000? At least I believe we had 16MB of memory, but how easy it was to get things wrong… I vaguely remember that we had to worry about memory management, cache control, and even multiprocessor operations in some cases. Writing code was challenging enough, but trying to debug someone else's code? And how about debugging an assembler program that modified its own instructions as it ran?

With parallel programming, hero programmers have had to deal with many, many additional challenges, such as parallelization, synchronization, communication, placement, debugging, etc.

Here are a few of the heroes who have programmed libraries and applications for high performance computing platforms in recent history. You may not be familiar with Exact, CESAR, and ExMatEx: they are DOE co-design centers. I have shown here pictures of the lead PIs, but many many more heroes work in these teams. And of course the complete list of heroes is much larger: likely wouldn't fit on an entire presentation…

In order to program scientific applications that run and are optimized on the complex Exascale architectures and beyond, we cannot hope to educate and train armies of hero programmers, and even if we were able to, it has been demonstrated that even top hero programmers can inadvertently prevent compiler and runtime optimizations, resulting on underperforming codes.

So, something needs to change.

But can we completely replace hero programmers? enter M-5 designed to replace a well-known, respected and liked hero, Captain Kirk: **https://www.youtube.com/watch?v=XctMbIwI18I**

In this Star Trek episode, broadcasted in 1968, replacing captain Kirk by the Ultimate Computer was clearly not a good idea…

We are not planning to replace hero programmers. However, in our future programming environments we do need to introduce a lot of automation in addition to count on classes of hero programmers and domain scientists to be in a feedback loop with the automated steps of these environments. We also need to enable domain scientist to express their algorithm using the language of their science and independently from the algorithm's implementation.

Domain scientists should not feel that they need to be hero programmers before they can benefit from Exascale systems and beyond.

Slide 2
Before I discuss our vision of future programming environments, let me say a few words about the scientific applications of interest to DOE and the need for Exascale computing in order to achieve new scientific breakthroughs.

In addition to scientific computing research, the DOE Office of Science manages research in:

- Basic Energy sciences, which includes material science, chemical science, nanotechnology, etc.
- Biological and Environmental sciences
- Fusion energy sciences
- High energy physics, and
- Nuclear physics

The next generation of advancements in these areas cannot be achieved without Extreme Scale Computing.
Exascale computing won't happen if we continue doing business as usual – there are no easy fixes, no magic bullets left, no low hanging fruit.
Exascale Computing and beyond will require major major rethinking of computing technology in order to enable new scientific breakthroughs.

We also need to keep in mind that nearly all large scale applications of importance to DOE are very data intensive, so Extreme scale computing needs to enable timely and more complex processing of increasingly large Big Data sets.

Slide 3

I cannot miss the opportunity to tell you that my office, ASCR, the Advanced Scientific Computing Research office has significant role in Exascale computing, as we manage the R&D of the hardware and software needed for the computational facilities that enable researchers to analyze, model, simulate and predict complex phenomena of importance to DOE.

Slide 4

As I just mentioned when discussing the applications of interest to DOE, we need to reinvent computing because the world has changed: we have hit a power wall and without major advances, the resulting power requirements of future systems would be unacceptable.

Using current technologies, existing solutions, if scaled 1000x, would have prohibitive energy consumption, on the order of 300MWs.

To quote Bill Harrod, We certainly need to blaze new trails through new territory, and we need to partner with the US computing industry to chart the future.

Slide 5

The unparalleled concurrency and heterogeneity of computing components anticipated in Exascale platforms, particularly at the node level, offer application developers ever more powerful capabilities to simulate and analyze complex phenomena. The vision is for 1,000x capabilities of today's platforms, with a 40x energy improvement over today's systems, operating with a power envelope of about 20MW.

The anticipated exascale benefits will come with challenges never before experienced. New programming environments will be needed that facilitate the design and development of new codes and new algorithms, as well as refactoring of legacy codes to match extreme-scale capabilities.

The vision for exascale computing includes a highly resilient system, based on marketable scalable and sustainable technologies, a system of broad utility, not a one-of-a kind system, to be deployed in the early 2020's.

The vision includes new programming and execution environments.
New programming environments must be accessible to domain scientists and all classes of programmers, must make it significantly easier to develop and manage code regardless of the increased system complexities, and must achieve high performance without resorting to platform-specific methods.

New execution environments must enable the dynamic, adaptive management of all systems resources, including power, as well as the management of resulting performance execution in the presence of significant variability and faults.

Slide 6

Exascale challenges have been discussed over the last several years in a large number of conferences, workshops, meetings, etc., The DOE ASCR website has a number of reports on the Exascale challenges, the FOAS issued in the last 3-4 years attempt to summarize them, and a report on the top ten challenges was recently completed and published by the DOE Advisory committee and is available on the ASCAC website under Charges/Reports.

These challenges include billion-way concurrency, resiliency, energy efficiency, the reducing ratio of memory per FLOP, managing heterogeneity and complexity, portability, scalability, etc.

I am listing here a few of the elements that we envision will be part of new programming models:

With the different types of processors, accelerators, processing in memory and near memory, deep memory hierarchies, etc., data paths will be very heterogeneous and complex. We need to develop new programming models for processing along these data paths in an optimal way.

Data centric constructs need to become first-class citizens in the new programming models, including automated support for data movement across memory hierarchy levels and heterogeneous processors.

Today, most runtime systems move data to where the computation is. We need programming model features to express data locality, data movement, and optimization based on the data movement patterns. And we need to let compiler/runtime determine how to map work based on the location of data and available computing resources.

Key to performance portability and productivity is separating what the algorithms do from how they are implemented. Traditional imperative semantics make achieving asynchrony very challenging or impossible. Declarative programming can greatly simplify parallel programming, enabling the separation of concerns, and allowing high level programmers to specify **what is required to be achieved, not how to achieve it.**

Programming models also need to have some low level interfaces that enable expert programmers to have control over things like parallel semantics, name space, locality, and data movement.

We had a Programming Models Summit this year and are currently working on a report that will help us identify all needed abstractions, their compositions, and programming environments that support the new programming models, including interfaces to applications and to runtime systems. We are targeting SC'14 as the deadline for publishing this report, which will be available at the ASCR website.

An important element of new programming environments is the development and tool-chain support of domain-specific languages that enable scientists to talk at the level of abstraction that they are familiar with.

Mapping these abstractions to efficient implementations on a particular platform, requires automation in transformations, mappings, refinements, and optimizations, involving multiple categories of programmers in the loop.

Slide 7:

Recognized challenges and identified needs and solutions have led to our envisioned productive and performance portable programming environment. Warning to hero programmers in the room: your first reaction this this vision may be "NO WAY!"

It is true that the envisioned environment is ONLY viable if significant automation and reuse is in place. If for every change a domain scientist makes at the highest level, he would have to go through all of these steps, involving several different experts, this would not be viewed as an easy to use and productive environment.

Allowing domain scientists and programmers to use their own language and expertise, in addition to automation and reuse is what really makes this work.

Automation will mean that for every one of these refinement steps, we will rely on the experts only for the high-level insights; all the low-level reasoning required to apply those insights and make them work should be handled by a machine.

Reusability will mean that if the same insights apply to an entire class of applications, the experts should have to provide them only
once and the system should be able to apply them where necessary.
A lot of the reusability comes from the use of domain specific languages. Many of the insights will be in terms of DSLs, so they can be applied to any program written in the DSL, and a lot of the low-level reasoning about when to apply certain transformations can be handled by a combination of synthesis and autotuning.

It isn’t important what exactly we include in each these step, what is important is to realize that these steps allow for the needed separation of concerns, and that different experts are able be in the loop to provide information and optimization beyond the capabilities of the automated transformations/mappings/refinements.

In this vision slide, we give an idea of the steps that will help development teams deal with the complexity of exascale systems and beyond, and create code that optimally run on different platforms with only a few changes to the last two steps.

1. In this vision, Domain scientists are able to specify application models using the domain science language, say PDEs.
2. From that, a team of Domain scientists and computational scientists specify discretizations of models
3. Followed by a team of Computational and computer scientists that specify parallel algorithms for evaluating discretizations
4. With these high level abstractions, computer scientists and software engineers develop machine-independent code and also identify existing libraries/frameworks that can be used.

5. At the next step, teams of Software and systems engineers tune code for different hardware platforms by specifying data and computation mappings, data movement, and resilience requirements, At this level, we still should have machine-independent code!
6. The last step involves the mapping of the application onto different platforms using programmer annotations, compiler optimizations, runtime optimizations, and auto-tuning systems.
7. Refinement loops among the steps enable further tuning of both machine-independent and machine dependent code.

The end result is a runtime optimized code for a particular platform that can be easily performance ported to a different one.

Slide 8

At DOE ASCR we have created the Extreme Scale Software Stack, ES-cube program.

Part of the ES-cube is the X-Stack program, started in 2012.

The X-Stack projects have been for the last two years pursuing the research and development needed to create models, languages, compilers, mechanisms, and tools that match the vision just presented.

The projects are also pursuing research in new runtime systems and runtime optimization mechanisms in order to ensure the highest levels of performance for any application, running on any platform, scaling up or down from Exascale.

The X-Stack program also include research and development of methods and tools that enable the refactoring of existing codes (MPI, OpenMP) to new platforms.

The projects shown with the green background are called centers as they are large collaboration among Labs, Universities, and/or Industries. These four large centers have closely engaged with the co-design centers in order to understand and use apps and mini-apps for the evaluation of the software components under R&D.

The DEGAS project is focusing on … read slide
Traleika focuses on explorations of … read slide
The D-TEC team is working on a complete software stack solution (excluding OS), with a focus on DSLs and the tool-chain to support them.
And the XPRESS project focuses on … read slide

The other five projects work on complementary technologies for auto-tuning, compiler and runtime optimizations, resilience mechanisms, correctness methods, and tools.

These projects are making very impressive progress.

For a lot more information about these projects, you can check the xstack wiki, https://xstackwiki.modelado.org/Extreme_Scale_Software_Stack

Slide 9

Each one of the "X-Stack centers" will deliver by September of next year a programming environment that integrates research prototypes built as they pursue their R&D plans.

I don't have time to describe in detail any of the programming environments, but want to give you a flavor for them. This one is from the D-TEC center and focuses on supporting DSLs for Exascale. They have developed two complementary technologies that support the use of DSLs (both embedded and general DSLs):

1) Rosebud, which is used to define DSLs *by domain scientists, generating a DSL compiler, so that application teams don't need to wait years and years before a compiler for their DSLs is created.*
2) The other approach, shown in the picture in the "transformation/refinements" box, is based on sketching, machine learning, and formal methods, *involving a* Series of manual refinements steps (code rewrites), defining transformations and

performance equivalence checking between steps to verify correctness.

These technologies are built on top of the ROSE compiler infrastructure, using a single shared runtime system: the SEEC model (DARPA HPCS program, self-aware computing), which enhances the X10 runtime system. A Parameterized Abstract Machine Model informs the compiler and runtime system about details of the hardware, to be used for auto-tuning and optimizations.

Notice that this programming environment also includes a migration path for existing apps.

Slide 10

We have a large number of excellent results from this program, which pursue the programming environment vision that I presented. I can only briefly show you a few of them. The highlights that I will present were sent to me by the project PIs.

In this slide, we show the significant reduction of code size and complexity, separation of algorithms from the execution schedule, and the significant reduction in execution time for the HPGMG mini-app.

Slide 11

This slide shows the use of DSL to represent mathematical operators and automate construction of discretizations for Cartesian and Curvalinear coordinate grids. The resulting framework supports automated generation of higher order stencils for 2nd, 4th, 6th, and 8th order operators, which I understand would be insane to do manually.

The picture on the top right shows code generated using the two types of coordinates for a 4th order wave equation, 3D discretization of an electromagnetics problem. Notice that the domain scientist only had to write about 10 lines of MAPLE (DSL) code.

The picture on the bottom shows how much faster a code generated for high order stencil computations is when compared to a baseline.

Slide 12

Here we show results on auto-parallelization and locality optimization of small codes (stencils, solvers) using Reservoir Labs R-STREAM compiler, running on task oriented runtimes. The picture shows that the resulting code has greater scalability than OpenMP as number of threads increases for most of the codes.

Slide 13

Here we show results from applying new communication-avoidance algorithms in the DEGAS programming environment. The bottom graph shows the speedup of a new N-Body algorithm (called a 1.5D) relative to the old one (1D) on a large number of cores (ALCF/Intrepid and NERSC/Hopper) using this technique.

Slide 14

DEGAS PGAS language plus asynchronous communication plus adaptive scheduling enabled the scaling of a genome assembly algorithm involving irregular data-intensive computations that scale to 15K cores and allowed the assembly of wheat that simply didn't run before using DEGAS approaches, which allowed the assembly to complete in 32 seconds on the NERSC's Edison.
The work has just been accepted as an SC14.

Slide 15

The current vision of Exascale Execution Environments is captured in OS/R Report published in the DOE ASCR website. The vision was developed over a one year work by the DOE OS/R council, culminating in this report. I will not have time to go over the elements of this vision, so I encourage you to read this report.

In the OS/R program we are pursuing research and development in operating and runtime systems that coordinate to globally manage power, concurrency, and faults, integrating with X-Stack runtimes, standard runtime libraries and resource managers responsible for optimal mapping of jobs onto the system.

We expect that research prototypes to be delivered will be platform-neutral and will converge to one, vendor sustained OS/R. We are also pursuing high impact/high risk technologies that may contribute to the OS/R architecture, design, and implementation.

The ARGO project is doing R&D on new OS/R architecture and design, new Node OS/R, lightweight self-aware, goal-based, active runtime system, and a hierarchical, coordinated global optimization of system resources. It also includes a backplane for events, control, and performance.

The HOBBES project is also doing R&D on complete node, enclave, and global system design, extending previous designs in node and global OS/R. The full system virtualization approach can be used to support almost any RTS. The research includes composition of applications developed for different programming models with incompatible RTS. The Global information bus include mechanisms for sharing status information. HOBBES is working on extensions of existing power management, power control, and on tunable resilience, with cross-layer resilience coordination. It is also developing low level OS mechanisms and global privileged services for system management.

The X-ARC project is a much smaller project, based on the Tesselation OS architecture, developed by Prof. Kubiatowicz and team at UC Berkeley. The research pursued in this project focuses on Cross Nodes adaptive resource control, support for new PGAS programming models (DEGAS), enabling increased control over resources such as cores, memory, and power and user-space handling of control requests and events. The research also

includes Advanced Memory Management, Power Awareness mechanisms, and services for resilience.
Much more information about these projects is available in the xstack wiki.

Slide 16

Near Future (2015-2016): we expect that one or two programming environments will evolve from the integration of current technologies.

Future:
We are developing a software stack plan for 2016-2023. Of course, the plan is not complete or approved, so I can only speak in general terms, and examples provided here will not necessarily be included in the final plan. As it may be quite obvious, we are planning significant extensions to the research in programming and execution environments that is currently being pursued in the ES-cube program that I have briefly covered today.

We are considering open issues in a number of research areas. The goal is to achieve by 2023 a much more productive, performance portable, energy efficient, resilient, programming and execution environments and to enable much more refined optimizations and tuning of the codes being developed and deployed to future architectures.

Read a few of the bullets

Slide 17

This is my last slide. It shows the present software stack efforts and the planned future software stack program in perspective with other DOE programs, such as Fast Forward, Path Forward, Design Forward, Co-design, application development and platform acquisitions.
The ES-cube software stack program is currently under the umbrella of the DOE ASCR Extreme Scale Research Program showing in blue in the first line of this picture. The future software stack program will happen under the umbrella of the Exascale Computing Initiative, which is currently being

planned, that is to say, final plan is not yet published and funds for this initiative are not yet approved.

The future software stack will be tested on testbeds yet to be established and on hardware prototypes P0, P1, P2 shown here. These prototypes grow in size and complexity, from one node, P0, to multiple nodes, P1, to a full scale prototype, P2.