High Productivity and the Cray Cascade System

Keith Shields
Director, Cray HPCS Program Office
keith@cray.com

Los Alamos Computer Science Symposium
October 14th , 2008

Adaptive Supercomputing Vision

Combines multiple processing architectures into a single, scalable system

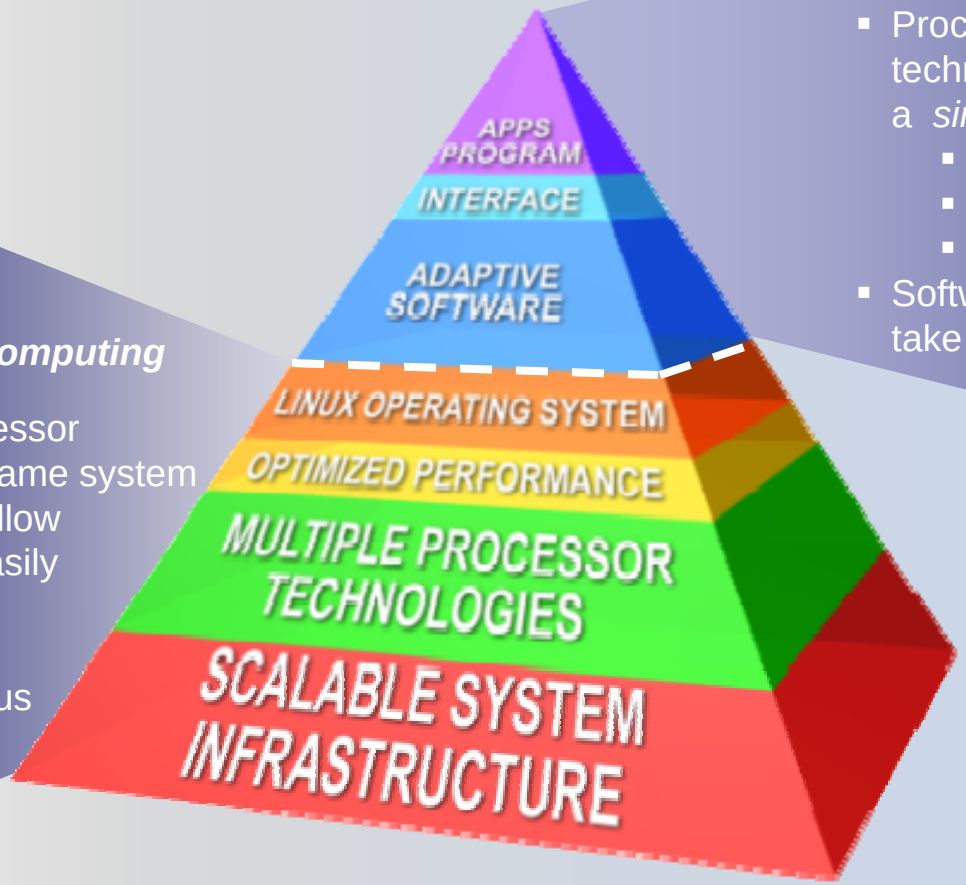
Tomorrow – Adaptive Supercomputing

- Processor that can leverage multiple technologies while operating within a *single code*
  - *Scalar*
  - *Vector*
  - *Multi-threading*
- Software (compilers & languages) to take advantage of these features

Today:

Hybrid Supercomputing

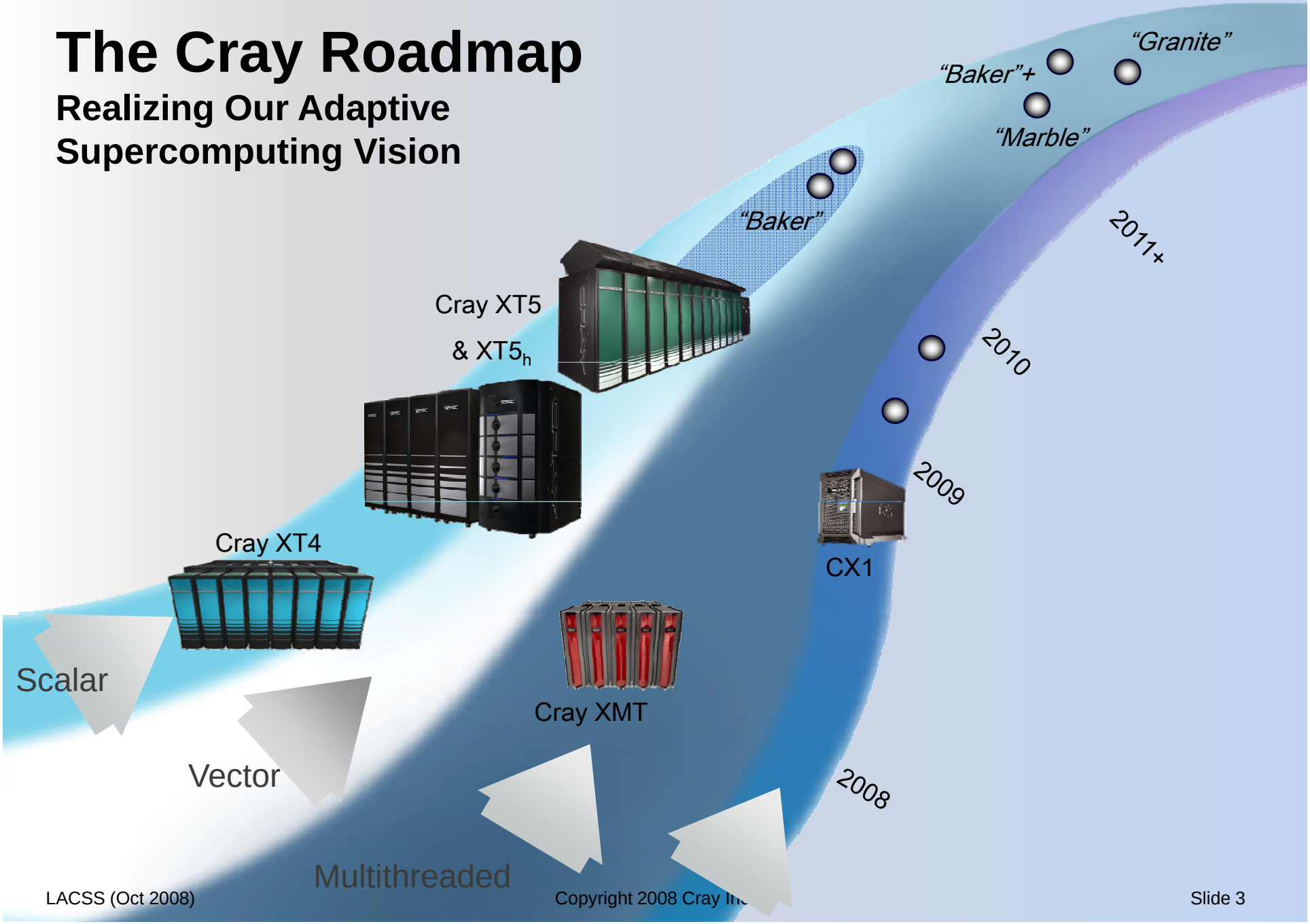
- Multiple processor types in the same system
- Software to allow them to be easily used and administered
- Heterogeneous workflows



Adapt the system to the application – not the application to the system

The Cray Roadmap

Realizing Our Adaptive Supercomputing Vision



High Productivity Computing Systems

Critical to National Security

- Develop a **new generation** of **economically viable** high productivity computing systems for national security and industrial user communities (2011)
- **Ensure U.S. lead, dominance, and control** in this critical technology

Phase III Vendors: CRAY IBM

Mission Partners: NSA Office of Science NNSA

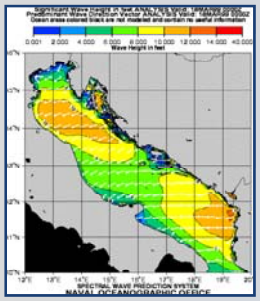
DARPA

Impact:

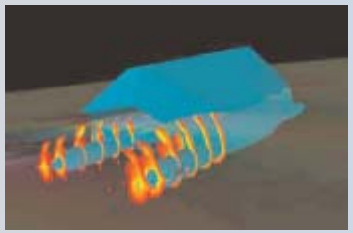
- **Performance** (time-to-solution): speedup critical apps by **10X to 40X**
- **Programmability** (idea-to-first-solution): dramatically reduce cost and time for developing apps
- **Portability** (transparency): insulate software from system specifics
- **Robustness** (reliability): continue operating in the presence of failures

Applications:

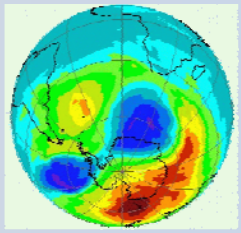
Weather Prediction



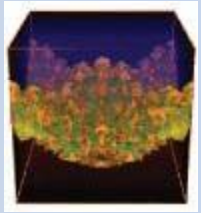
Ocean/wave Forecasting



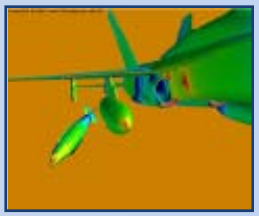
Ship Design



Climate Modeling



Nuclear Stockpile Stewardship



Weapons Integration

Fill the Critical DoD Need for:

Operational weather and ocean forecasting, weapons design and analysis, airborne contaminant modeling, intelligence / surveillance / reconnaissance, cryptanalysis, etc.

Productivity Begins with the Architecture

- Global shared address space with one-sided data transfers
  - *So that code can easily reference and access objects held in remote nodes without involvement of code running on those nodes*
- High bandwidth, fine granularity network
  - *So that programs can be written with far less concern about how and when communication takes place*
- Latency-tolerant processors
  - *So that compute capabilities do not go idle waiting for data, and programmers do not have to stage data and computation*
- Plentiful threading with efficient, lightweight synchronization
  - *So that parallelism can be dynamically exploited at multiple levels in the code, and programmers need to worry less about load balancing and synchronization*
- Adaptive processing capabilities
  - *So that idioms that would benefit from vectorization, streaming, fine-grain multithreading, or fast sequential processing can execute efficiently, and the programmer does not have to change the code to fit the paradigm*

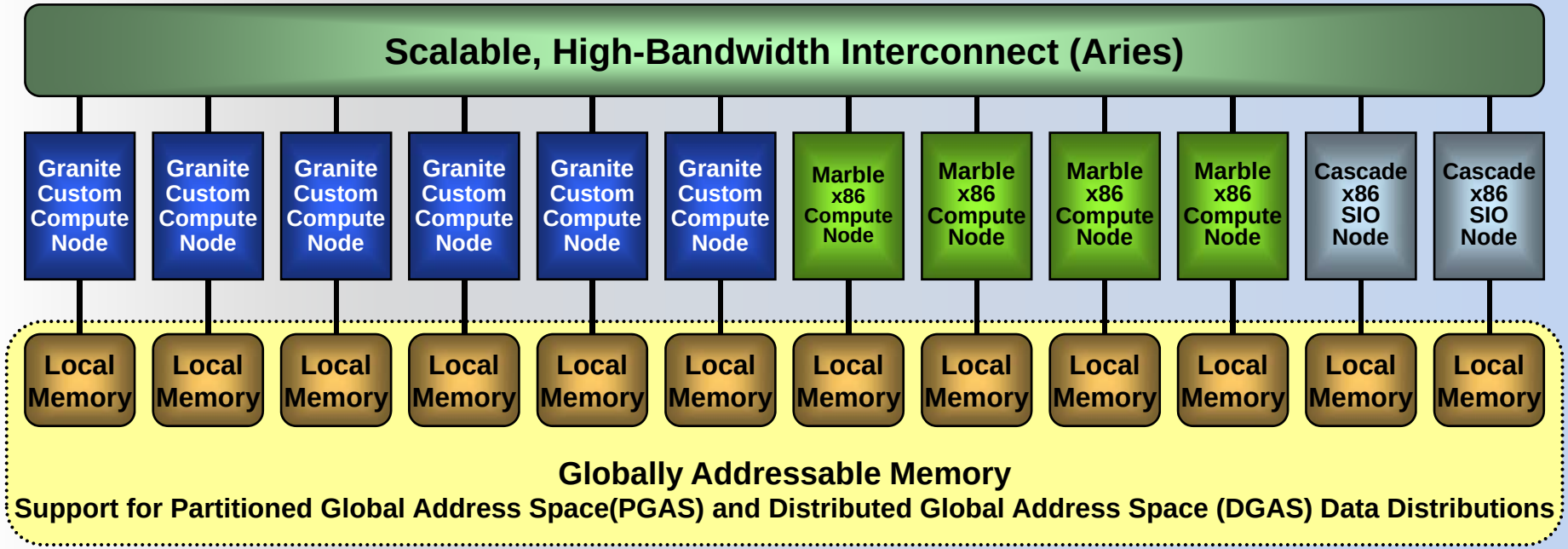
A New Partnership

- Collaborative agreement to pursue high-end HPC technologies and opportunities
  - Intel's world-class IC process and processor architecture
  - Cray's strength in HPC architecture & large systems
  - Cray working with Intel on a new high-end processor

- Our strong partnership with AMD continues
- Cray will continue to develop and upgrade our existing XT product line past the end of the decade

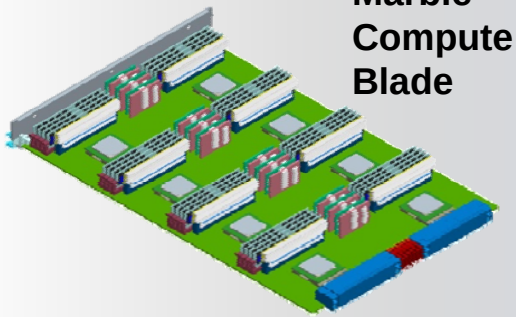
Cray Cascade System

DARPA High Productivity Computing Systems

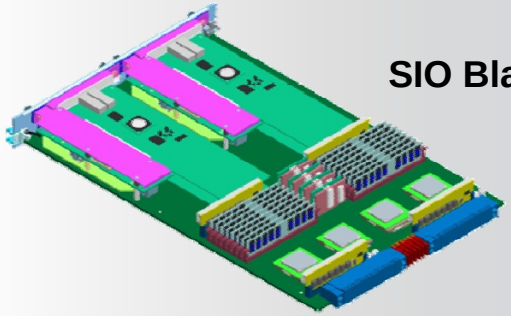


- Tightly integrated hybrid computing
- Configurable network, memory, processing and I/O
- Globally addressable memory
- Very high performance communication and synchronization

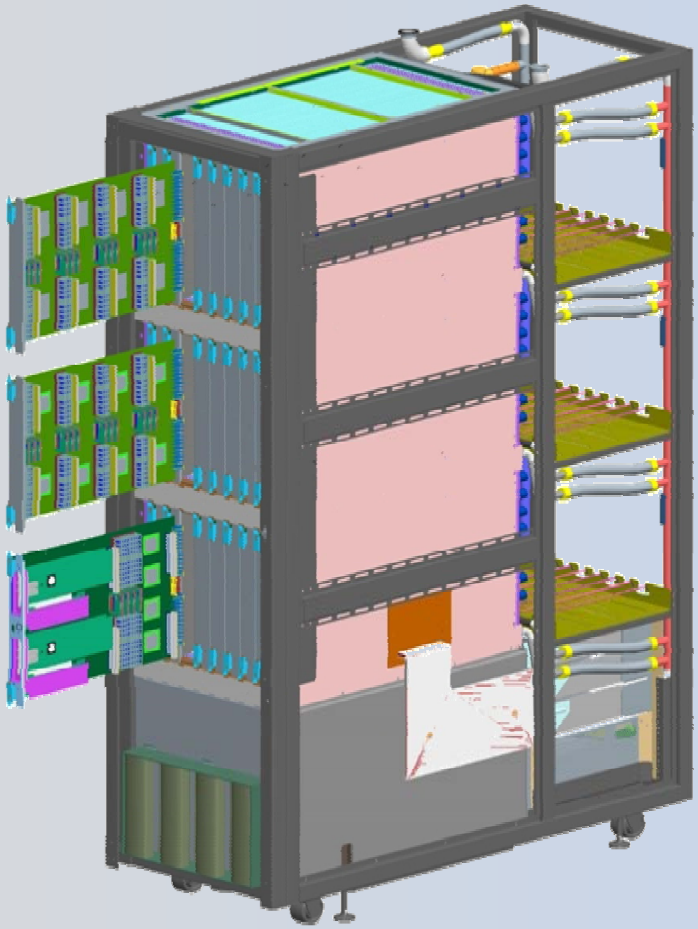
Cascade Packaging Overview



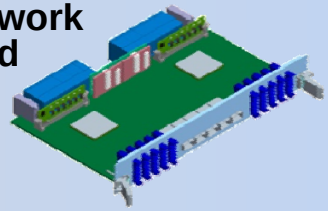
Granite or Marble Compute Blade



SIO Blade



Aries network card



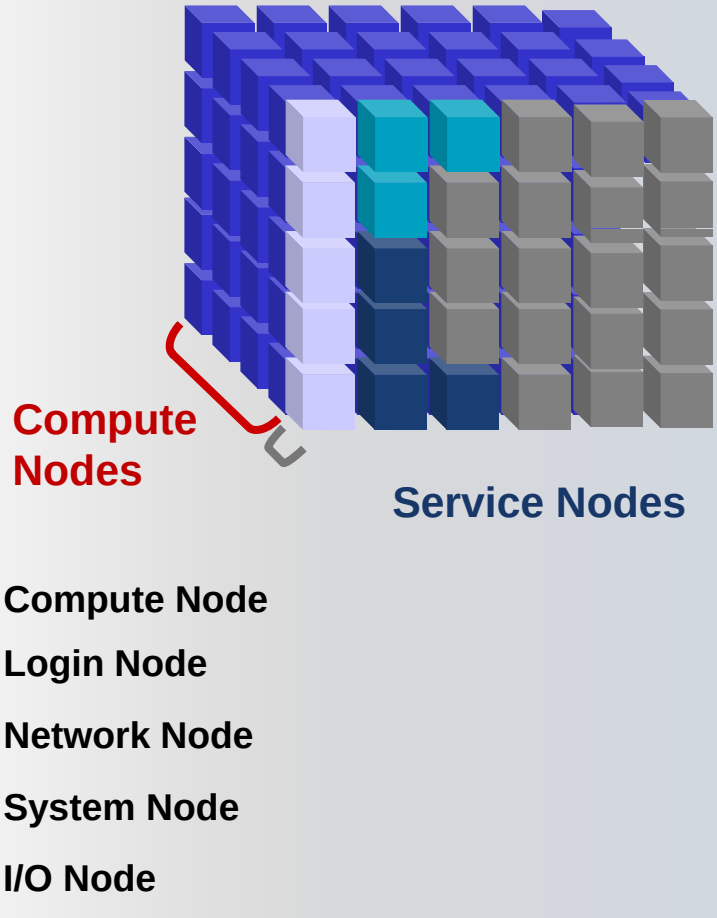
Cascade Cabinet

- Improved density and cooling over XT4
- Common network with multiple blade types
- Extensible over multiple years

Cray Linux Environment

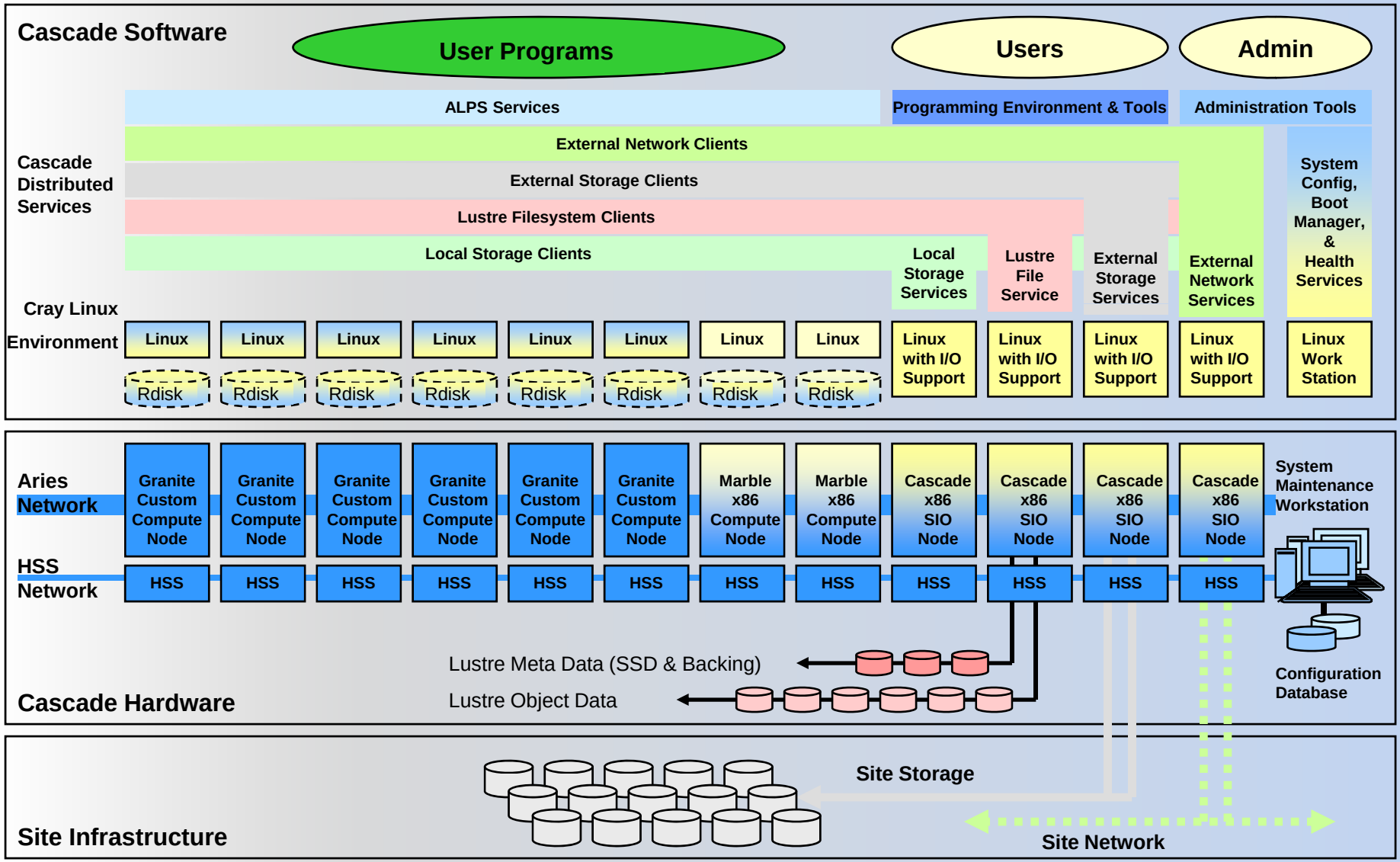
World's Most Scalable Software System

"*Primum non nocere*" (First, do no harm)



- Microkernel on Compute Nodes, full featured Linux on Service Nodes
- Service Nodes specialized by function
- Software Architecture
  - Minimizes OS "Jitter"
  - Enables reproducible runtimes
- Large machines boot in 15 to 30 minutes, including filesystem
- Fast job launch for hundreds of thousands of nodes
- High bandwidth Lustre filesystem

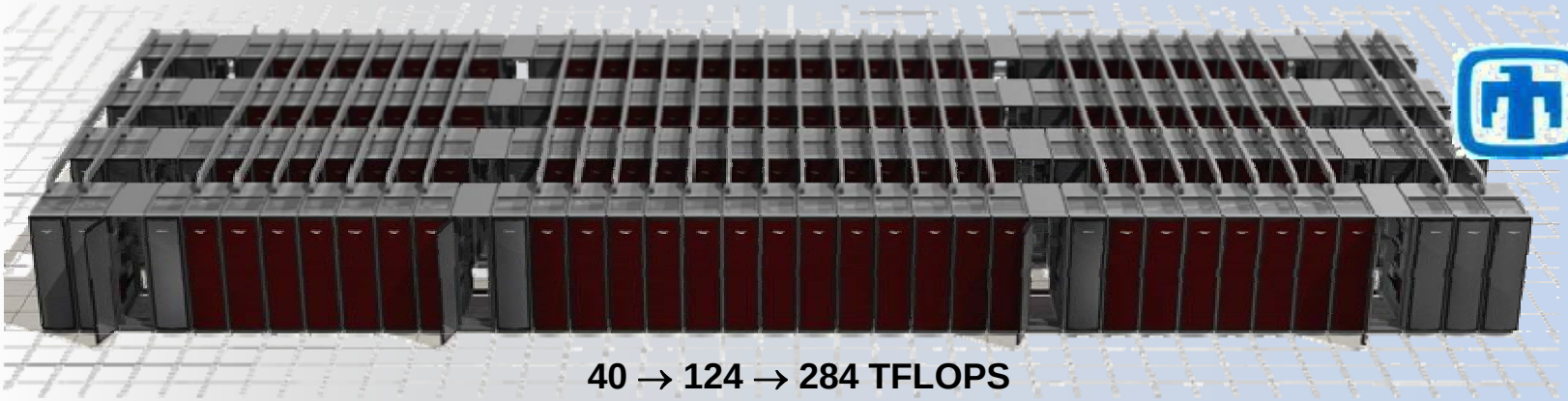
Cascade OS Architecture



Some Very Large Systems from Cray

100TFLOPS → 350TF

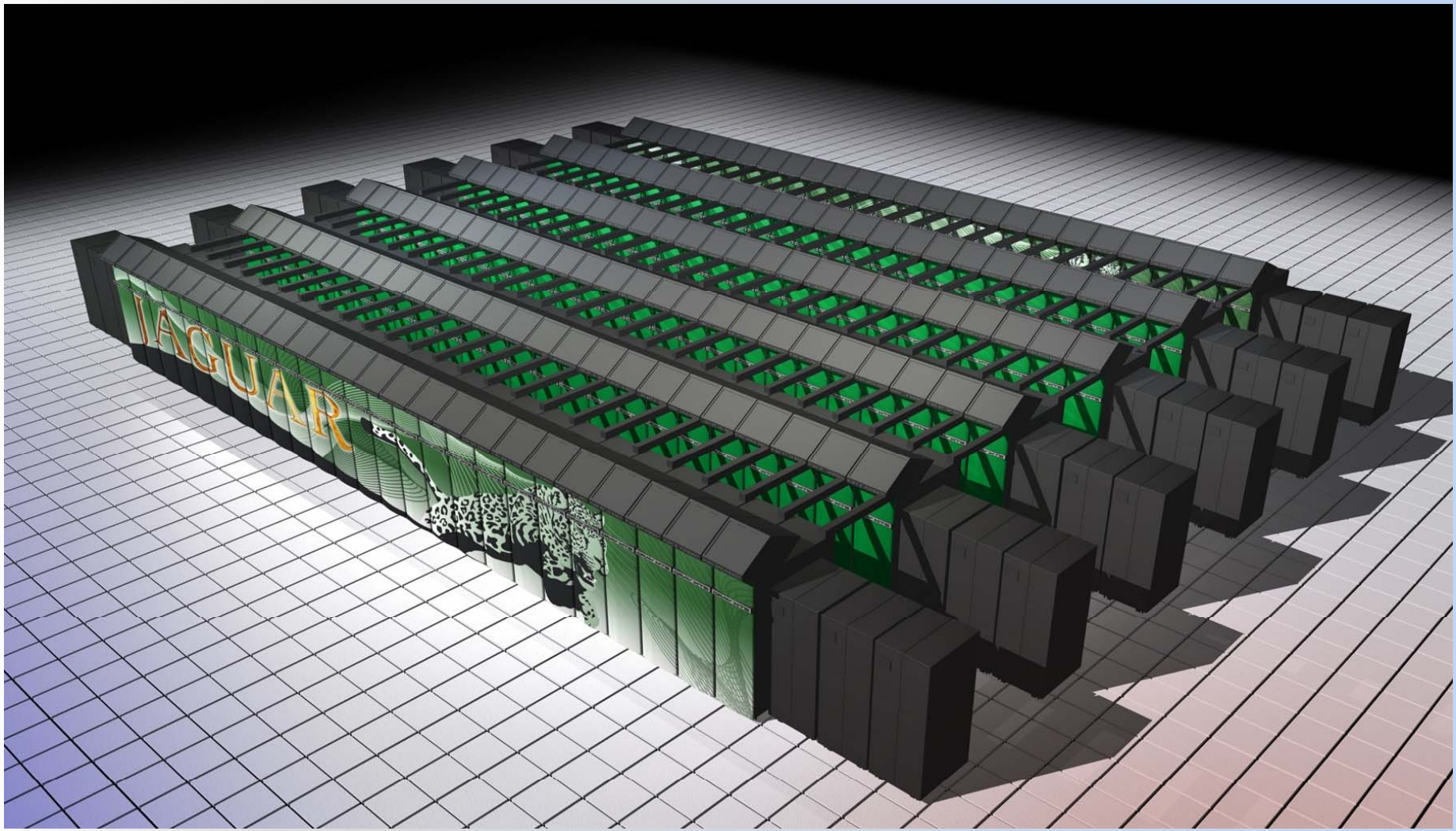
20 → 50 → 119 → 263 TF → ~1 PF



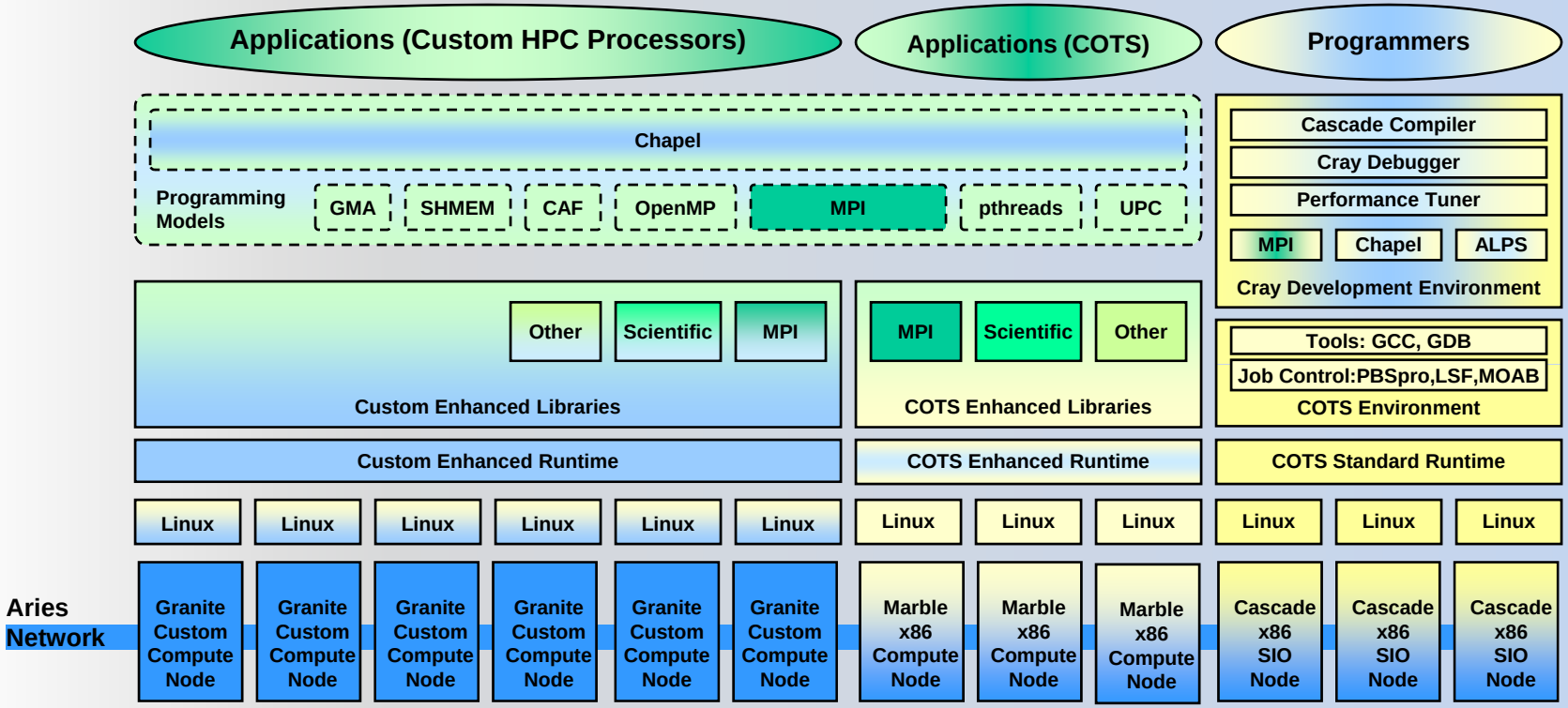
Sandia National Laboratories

40 → 124 → 284 TFLOPS

ORNL Petaflop System, 2H 2008



Cascade Programming Environment



- COTS and Customized libraries and runtimes
- Common and emerging programming models supported across processor types
- COTS and Cray Custom development environments

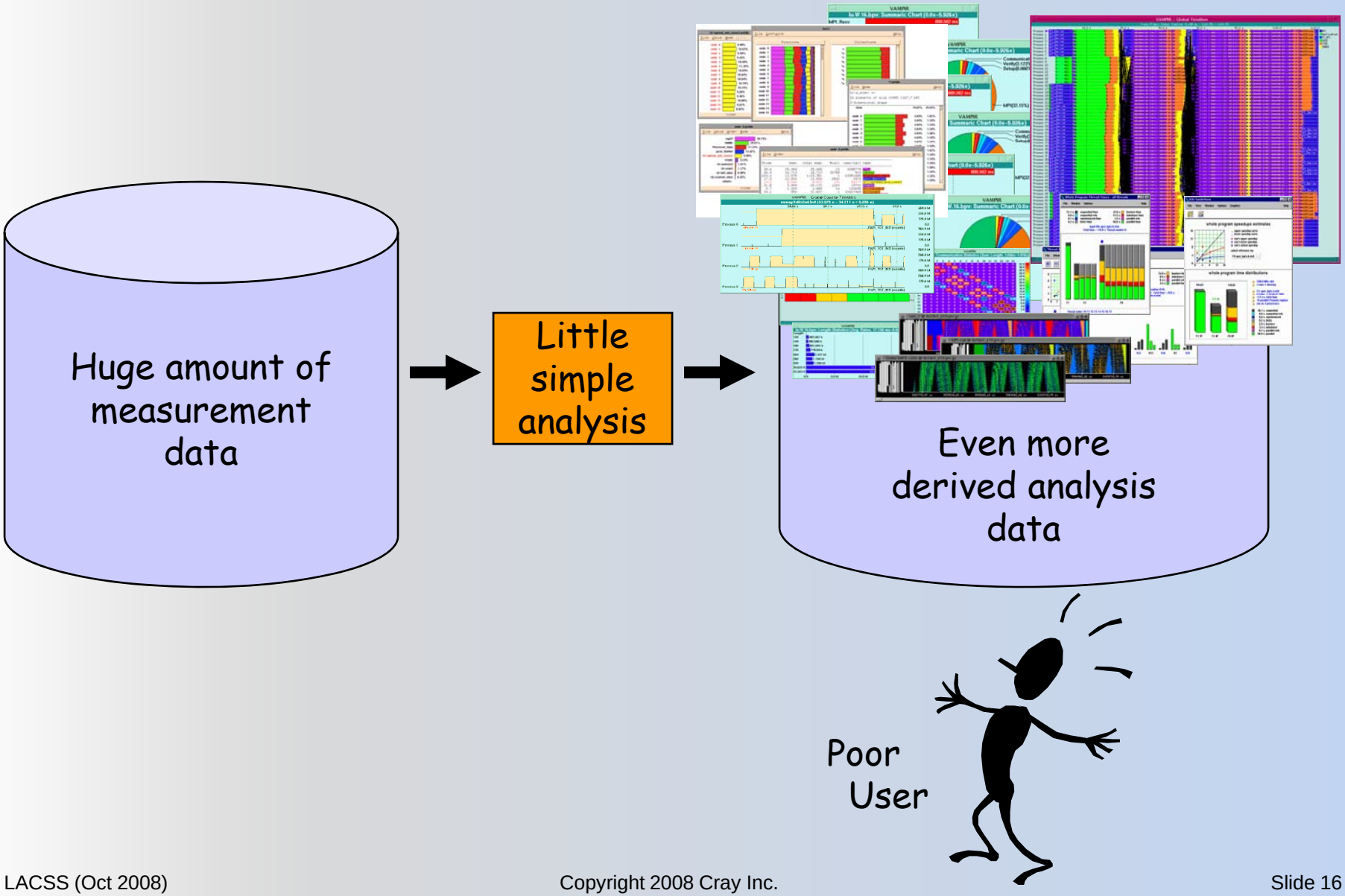
Key Areas of PE Productivity Enhancement

- Compilers
  - Incremental compilation, runtime profiling, improved user feedback
  - Fully automatic multi-level parallelism (shared memory, multi-threading, vectorization)
- Programming Tools
  - Environment setup
  - Debugging
  - Performance Analysis
- Scientific Libraries
- Programming Languages
  - Support for traditional languages
  - Integrated support for UPC, CAF, OpenMP
  - Chapel

Debugging in Cascade

- Support for traditional debugging mechanisms
  - TotalView, DDT, and gdb
- Comparative debugging
  - A data-centric paradigm rather than control-centric paradigm
  - Compare working application to failing application
    - Simultaneous execution of both
    - Comparison of computed results at key moments
    - Focus on data – not state and internal operations
    - Narrow down problem without massive thread study
  - Comparative debugging scenarios
    - Serial converted to parallel
    - Small scaling versus large scaling
    - One optimization level versus another
    - One programming language converted to another

Optimization with Traditional Performance Tools



Automatic Performance Analysis in Cascade

- Basic approach:
  - Intelligently collect and filter data
  - Distinguish between “similar” and “different” application behavior
  - Search data for inefficient execution patterns using performance models
- Automatically identify and expose performance anomalies
  - Load imbalance (MPI and OpenMP)
  - Communication / synchronization / I/O problems
  - Environment variables
  - Etc.
- Support includes:
  - Automatic profiling analysis
    - Automatically detects the most time consuming functions in the application
    - Feeds information back to the tool for further (focused) data collection
  - Recomendation infrastructure in CrayPat
    - E.g.: MPI rank placement suggestions (how ranks are mapped to cores)
  - Scalable performance visualizer

Math Software Productivity Enhancements

- **Auto-tuning:** use code generator and automatic tester to develop codes
  - Cray Adaptive Sparse Kernels (CASK)

- **Adaptivity:** make offline and online decisions to choose best kernel/library/routine
  - Cray Adaptive FFT (CRAFFT)
  - Cray Adaptive Sparse Kernels (CASK)

- **Performance Enhancements:**
  - Iterative Solver Performance
  - FFT performance
  - Fast libm

Chapel

A new parallel language developed by Cray for HPCS

Themes

- Raise level of abstraction, generality compared to SPMD approaches
- Support prototyping of parallel codes + evolution to production-grade
- Narrow gap between parallel and mainstream languages

Chapel's Productivity Goals

- Vastly improve **programmability** over current languages/models
- Support **performance** that matches or beats MPI
- Improve **portability** over current languages/models (actually better than MPI)
- Improve code **robustness** via better abstractions and semantics

```
...
coforall block in UpdateSpace.subBlocks do
  for r in RAStream( block ) do
    T( r & indexMask ) ^= r;
```

Status

- Draft language specification available
- Portable prototype implementation underway
- Most effort has been focused on functionality and feature evaluation
- Early releases to ~40 users at ~20 sites (academic, government, industry)
- Intend to do public release in late 2008

Fortran+MPI 3D 27-point stencil (NAS MG *rprj3*)

```
      subroutine comm3(u,n1,n2,n3,kk)
      use caf_intrinsics

      implicit none

      include 'cafnpb.h'
      include 'globals.h'

      integer n1, n2, n3, kk
      double precision u(n1,n2,n3)
      integer axis

      if( .not. dead(kk) )then
         do  axis = 1, 3
            if( nprocs .ne. 1) then
               call sync_all()
               call give3( axis, +1, u, n1, n2, n3, kk )
               call give3( axis, -1, u, n1, n2, n3, kk )
               call sync_all()
               call take3( axis, -1, u, n1, n2, n3 )
               call take3( axis, +1, u, n1, n2, n3 )
            else
               call comm1p( axis, u, n1, n2, n3, kk )
            endif
         enddo
      else
         do  axis = 1, 3
            call sync_all()
            call sync_all()
         enddo
         call zero3(u,n1,n2,n3)
      endif
      return
      end


      subroutine give3( axis, dir, u, n1, n2, n3, k )
      use caf_intrinsics


      implicit none

      include 'cafnpb.h'
      include 'globals.h'

      integer axis, dir, n1, n2, n3, k, ierr
      double precision u( n1, n2, n3 )

      integer i3, i2, i1, buff_len,buff_id

      buff_id = 2 + dir
      buff_len = 0

      if( axis .eq.  1 )then
         if( dir .eq. -1 )then

            do  i3=2,n3-1
               do  i2=2,n2-1
                  buff_len = buff_len + 1
                  buff(buff_len,buff_id ) = u( 2,  i2,i3)
               enddo
            enddo

            buff(1:buff_len,buff_id+1)[nbr(axis,dir,k)] =
>           buff(1:buff_len,buff_id)

         else if( dir .eq. +1 ) then

            do  i3=2,n3-1
               do  i2=2,n2-1
                  buff_len = buff_len + 1
                  buff(buff_len, buff_id ) = u( n1-1, i2,i3)
               enddo
            enddo

            buff(1:buff_len,buff_id+1)[nbr(axis,dir,k)] =
>           buff(1:buff_len,buff_id)

         endif
      endif

      if( axis .eq.  2 )then
         if( dir .eq. -1 )then
            do  i3=2,n3-1
               do  i1=1,n1
                  buff_len = buff_len + 1
                  buff(buff_len, buff_id ) = u( i1,  2,i3)
               enddo
            enddo

        buff(1:buff_len,buff_id+1)[nbr(axis,dir,k)] =
>           buff(1:buff_len,buff_id)

      else if( dir .eq. +1 ) then

            do  i3=2,n3-1
               do  i1=1,n1
                  buff_len = buff_len + 1
                  buff(buff_len,  buff_id )= u( i1,n2-1,i3)
               enddo
            enddo

            buff(1:buff_len,buff_id+1)[nbr(axis,dir,k)] =
>           buff(1:buff_len,buff_id)

         endif
      endif

      if( axis .eq.  3 )then
         if( dir .eq. -1 )then

            do  i2=1,n2
               do  i1=1,n1
                  buff_len = buff_len + 1
                  buff(buff_len, buff_id ) = u( i1,i2,2)
               enddo
            enddo

            buff(1:buff_len,buff_id+1)[nbr(axis,dir,k)] =
>           buff(1:buff_len,buff_id)

         else if( dir .eq. +1 ) then

            do  i2=1,n2
               do  i1=1,n1
                  buff_len = buff_len + 1
                  buff(buff_len, buff_id ) = u( i1,i2,n3-1)
               enddo
            enddo

            buff(1:buff_len,buff_id+1)[nbr(axis,dir,k)] =
>           buff(1:buff_len,buff_id)

         endif
      endif

      return
      end


      subroutine take3( axis, dir, u, n1, n2, n3 )
      use caf_intrinsics


      implicit none

      include 'cafnpb.h'
      include 'globals.h'

      integer axis, dir, n1, n2, n3
      double precision u( n1, n2, n3 )

      integer buff_id, indx

      integer i3, i2, i1

      buff_id = 3 + dir
      indx = 0

      if( axis .eq.  1 )then
         if( dir .eq. -1 )then

            do  i3=2,n3-1
               do  i2=2,n2-1
                  indx = indx + 1
                  u(n1,i2,i3) = buff(indx, buff_id )
               enddo
            enddo

         else if( dir .eq. +1 ) then

            do  i3=2,n3-1
               do  i2=2,n2-1
                  indx = indx + 1
                  u(1,i2,i3) = buff(indx, buff_id )
               enddo
            enddo

         endif
      endif

      if( axis .eq.  2 )then
         if( dir .eq. -1 )then

            do  i3=2,n3-1
               do  i1=1,n1
                  indx = indx + 1
                  u(i1,n2,i3) = buff(indx, buff_id )
               enddo
            enddo

      else if( dir .eq. +1 ) then

            do  i3=2,n3-1
               do  i1=1,n1
                  indx = indx + 1
                  u(i1,1,i3) = buff(indx, buff_id )
               enddo
            enddo

         endif
      endif

      if( axis .eq.  3 )then
         if( dir .eq. -1 )then

            do  i2=1,n2
               do  i1=1,n1
                  indx = indx + 1
                  u(i1,i2,n3) = buff(indx, buff_id )
               enddo
            enddo

         else if( dir .eq. +1 ) then

            do  i2=1,n2
               do  i1=1,n1
                  indx = indx + 1
                  u(i1,i2,1) = buff(indx, buff_id )
               enddo
            enddo

         endif
      endif

      return
      end


      subroutine comm1p( axis, u, n1, n2, n3, kk )
      use caf_intrinsics


      implicit none

      include 'cafnpb.h'
      include 'globals.h'

      integer axis, dir, n1, n2, n3
      double precision u( n1, n2, n3 )

      integer i3, i2, i1, buff_len,buff_id
      integer i, kk, indx

      dir = -1

      buff_id = 3 + dir
      buff_len = nm2

      do  i=1,nm2
         buff(i,buff_id) = 0.0D0
      enddo


      dir = +1

      buff_id = 3 + dir
      buff_len = nm2

      do  i=1,nm2
         buff(i,buff_id) = 0.0D0
      enddo

      dir = +1

      buff_id = 2 + dir
      buff_len = 0

      if( axis .eq.  1 )then
         do  i3=2,n3-1
            do  i2=2,n2-1
               buff_len = buff_len + 1
               buff(buff_len, buff_id ) = u( n1-1,
     i2,i3)
            enddo
         enddo
      endif

      if( axis .eq.  2 )then
         do  i3=2,n3-1
            do  i1=1,n1
               buff_len = buff_len + 1
               buff(buff_len,  buff_id )= u( i1,n2-
     1,i3)
            enddo
         enddo
      endif

      if( axis .eq.  3 )then
         do  i2=1,n2
            do  i1=1,n1
               buff_len = buff_len + 1
               buff(buff_len, buff_id ) = u( i1,i2,n3-
     1)
            enddo
         enddo
      endif

      dir = -1

      buff_id = 2 + dir
      buff_len = 0

      if( axis .eq.  1 )then
         do  i3=2,n3-1
            do  i2=2,n2-1
               buff_len = buff_len + 1
               buff(buff_len,buff_id ) = u( 2,  i2,i3)
            enddo
         enddo
      endif

      if( axis .eq.  2 )then
         do  i3=2,n3-1
            do  i1=1,n1
               buff_len = buff_len + 1
               buff(buff_len, buff_id ) = u( i1,
     2,i3)
            enddo
         enddo
      endif

      if( axis .eq.  3 )then
         do  i2=1,n2
            do  i1=1,n1
               buff_len = buff_len + 1
               buff(buff_len, buff_id ) = u( i1,i2,2)
            enddo
         enddo
      endif

      do  i=1,nm2
         buff(i,4) = buff(i,3)
         buff(i,2) = buff(i,1)
      enddo

      dir = -1

      buff_id = 3 + dir
      indx = 0

      if( axis .eq.  1 )then
         do  i3=2,n3-1
            do  i2=2,n2-1
               indx = indx + 1
               u(n1,i2,i3) = buff(indx, buff_id )
            enddo
         enddo
      endif

      if( axis .eq.  2 )then
         do  i3=2,n3-1
            do  i1=1,n1
               indx = indx + 1
               u(i1,n2,i3) = buff(indx, buff_id )
            enddo
         enddo
      endif

      if( axis .eq.  3 )then
         do  i2=1,n2
            do  i1=1,n1
               indx = indx + 1
               u(i1,i2,n3) = buff(indx, buff_id )
            enddo
         enddo
      endif


      dir = +1

      buff_id = 3 + dir
      indx = 0

      if( axis .eq.  1 )then
         do  i3=2,n3-1
            do  i2=2,n2-1
               indx = indx + 1
               u(1,i2,i3) = buff(indx, buff_id )
            enddo
         enddo
      endif

      if( axis .eq.  2 )then
         do  i3=2,n3-1
            do  i1=1,n1
               indx = indx + 1
               u(i1,1,i3) = buff(indx, buff_id )
            enddo
         enddo
      endif

      if( axis .eq.  3 )then
         do  i2=1,n2
            do  i1=1,n1
               indx = indx + 1
               u(i1,i2,1) = buff(indx, buff_id )
            enddo
         enddo
      endif

      return
      end

      subroutine rprj3(r,m1k,m2k,m3k,s,m1j,m2j,m3j,k)
      implicit none
      include 'cafnpb.h'
      include 'globals.h'

      integer m1k, m2k, m3k, m1j, m2j, m3j,k

      double precision r(m1k,m2k,m3k), s(m1j,m2j,m3j)
      integer j3, j2, j1, i3, i2, i1, d1, d2, d3, j
      double precision x1(m), y1(m), x2,y2

      if(m1k.eq.3)then
        d1 = 2
      else
        d1 = 1
      endif

      if(m2k.eq.3)then
        d2 = 2
      else
        d2 = 1
      endif

      if(m3k.eq.3)then
        d3 = 2
      else
        d3 = 1
      endif

      do  j3=2,m3j-1
        i3 = 2*j3-d3
        do  j2=2,m2j-1
          i2 = 2*j2-d2
          do j1=2,m1j
            i1 = 2*j1-d1
            x1(i1-1) = r(i1-1,i2-1,i3  ) + r(i1-1,i2+1,i3  )
>                    + r(i1-1,i2,  i3-1) + r(i1-1,i2,  i3+1)
            y1(i1-1) = r(i1-1,i2-1,i3-1) + r(i1-1,i2-1,i3+1)
>                    + r(i1-1,i2+1,i3-1) + r(i1-1,i2+1,i3+1)
          enddo
          do  j1=2,m1j-1
            i1 = 2*j1-d1
            y2 = r(i1,  i2-1,i3-1) + r(i1,  i2-1,i3+1)
>                + r(i1,  i2+1,i3-1) + r(i1,  i2+1,i3+1)
            x2 = r(i1,  i2-1,i3  ) + r(i1,  i2+1,i3  )
>                + r(i1,  i2,  i3-1) + r(i1,  i2,  i3+1)
            s(j1,j2,j3) =
>                0.5D0 * r(i1,i2,i3)
>              + 0.25D0 * (r(i1-1,i2,i3) + r(i1+1,i2,i3) + x2)
>              + 0.125D0 * ( x1(i1-1) + x1(i1+1) + y2)
>              + 0.0625D0 * ( y1(i1-1) + y1(i1+1) )
          enddo
        enddo
        j = k-1
        call comm3(s,m1j,m2j,m3j,j)
        return
        end
```

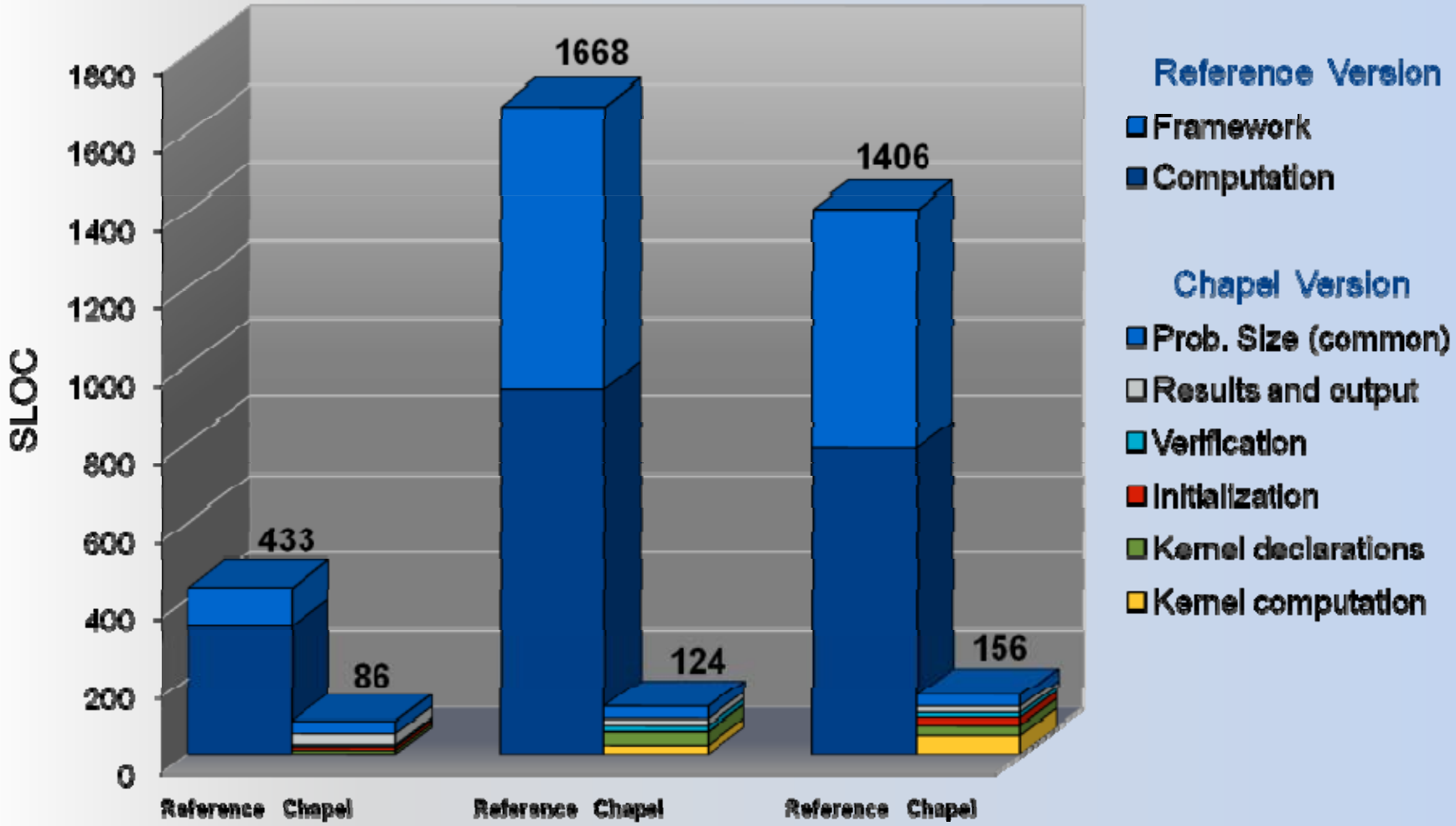
NAS MG *rprj3* stencil in Chapel

```
def rprj3(S, R) {
  param Stencil = [-1..1, -1..1, -1..1],
        w: [0..3] real = (0.5, 0.25, 0.125, 0.0625),
        w3d = [(i,j,k) in Stencil] w((i!=0) + (j!=0) + (k!=0));

  forall ijk in S.domain do
    S(ijk) = + reduce [offset in Stencil]
                        (w3d(offset) * R(ijk + R.stride*offset));
}
```

Chapel Code Size Comparison

For HPC Challenge Benchmarks



STREAM Triad

Random Access

FFT

More Chapel Information

- http://chapel.cs.washington.edu
- chapel_info@cray.com

- Initial public open source release prior to SC '08
- Chapel tutorial at SC '08 (Sunday, 11/16)
- Part of PGAS tutorial at SC '08 (Monday, 11/17)
- Talk at Multicore Programmability Workshop at SC '08 (Monday, 11/17)
- HPCC competition at SC '08

Acknowledgements

- This material is based upon work supported by the Defense Advanced Research Projects Agency under its Agreement No. HR0011-07-9-0001.

Thank you! Questions?

keith@cray.com