# State of the Chapel Union:

## HPCS Reflections and Musings about the Future

Brad Chamberlain

PGAS 2012: October 12, 2012

# What is Chapel?

- An emerging parallel programming language
  - Design and development led by Cray Inc.
    - in collaboration with academia, labs, industry
  - Initiated under the DARPA HPCS program

- **Overall goal:** Improve programmer productivity
  - Improve the programmability of parallel computers
  - Match or beat the performance of current programming models
  - Support better portability than current programming models
  - Improve the robustness of parallel codes

- A work-in-progress

# Chapel's Implementation

- Being developed as open source at SourceForge
- Licensed as BSD software
- **Target Architectures:**
  - Cray architectures
  - multicore desktops and laptops
  - commodity clusters
  - systems from other vendors
  - *in-progress:* CPU+accelerator hybrids, manycore, …

# Chapel's Setting: HPCS (slide circa 2009)

**HPCS:** High *Productivity* Computing Systems (DARPA *et al.*)

- Goal: Raise productivity of high-end computing users by 10×
- Productivity = Performance
  + Programmability
  + Portability
  + Robustness

- **Phase II**: Cray, IBM, Sun (July 2003 – June 2006)
  - Evaluated the entire system architecture's impact on productivity…
    - processors, memory, network, I/O, OS, runtime, compilers, tools, …
    - …and new languages:
      Cray: Chapel  IBM: X10  Sun: Fortress

- **Phase III**: Cray, IBM (July 2006 – )
  - Implement the systems and technologies resulting from phase II
  - (Sun also continues work on Fortress, without HPCS funding)
    (in fact, the Fortress Team pulled the plug as recently as July 2012)

# Outline

- ✓ Context
- ➢ Chapel Under HPCS
- Chapel Today
- Chapel's Future

CRAY
THE SUPERCOMPUTER COMPANY

# Where I came from

# ZPL

***ZPL:*** a contemporary of HPF

- similar goals, but a very different approach

**Developed by:** University of Washington

**Timeframe:** 1991 – 2003 (can still download today)

**Target machines:** 1990's HPC parallel platforms

- clusters of commodity processors
- clusters of SMPs
- custom parallel architectures
  - Cray T3E, KSR, SGI Origin, IBM SP2, Sun Enterprise, …

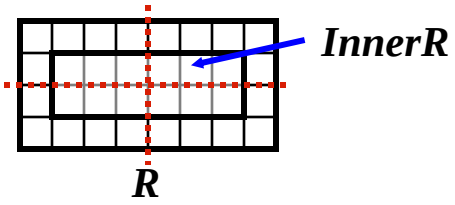
**Main concepts:**

- abstract machine model: CTA
- regions: first-class index sets
- WYSIWYG performance model

# ZPL Concepts: Regions

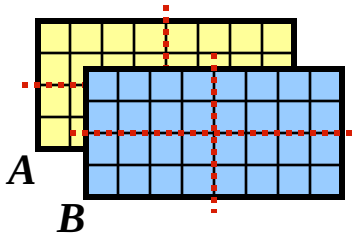
*regions:* distributed index sets…

```
region R      = [1..m, 1..n];
       InnerR = [2..m-1, 2..n-1];
```



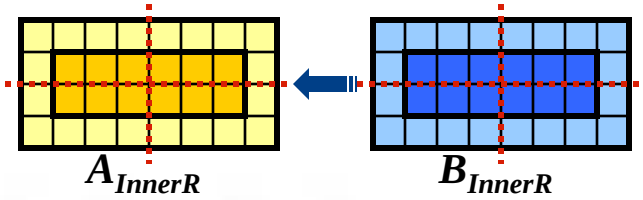

…used to declare distributed arrays…

```
var A, B: [R] real;
```

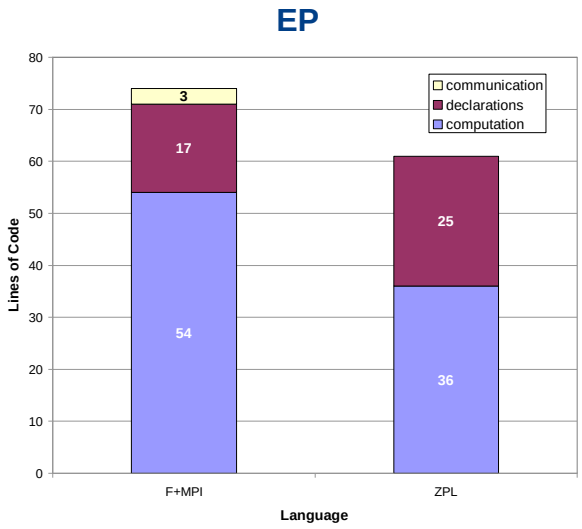


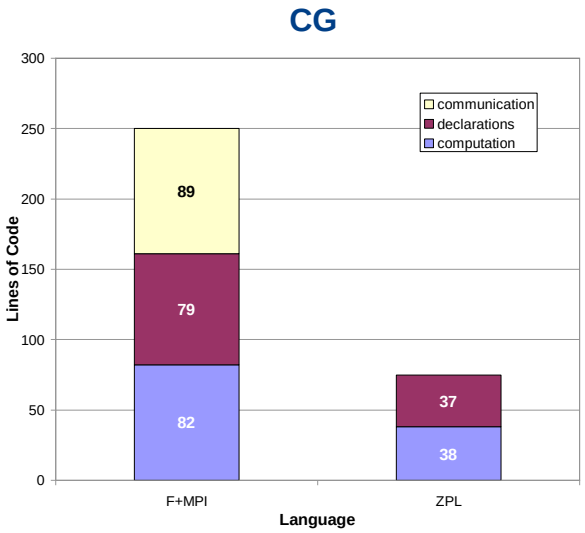

…and computation over distributed arrays

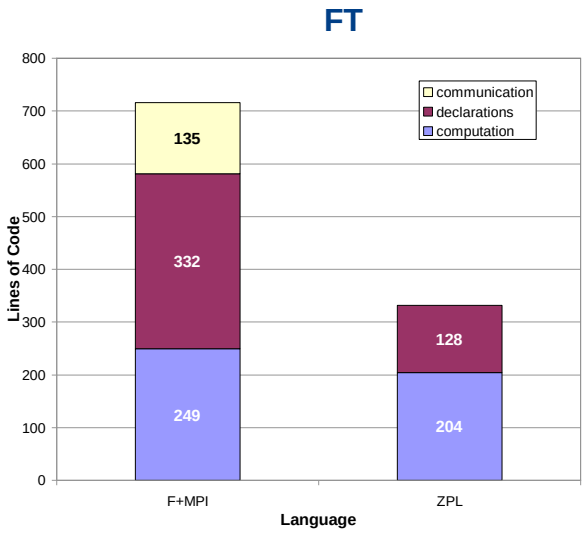
```
[InnerR] A = B;
```

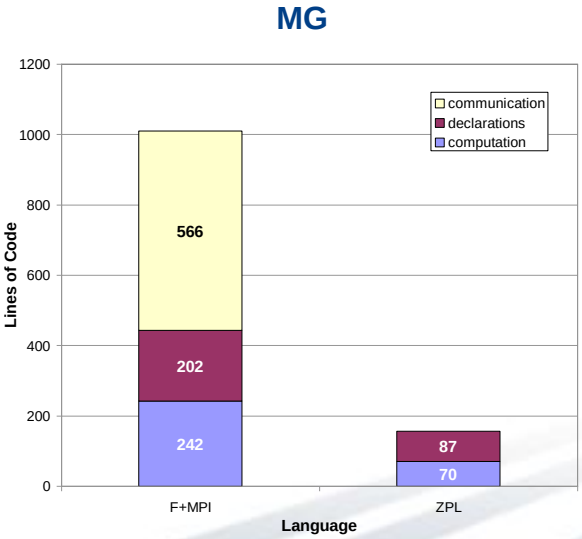


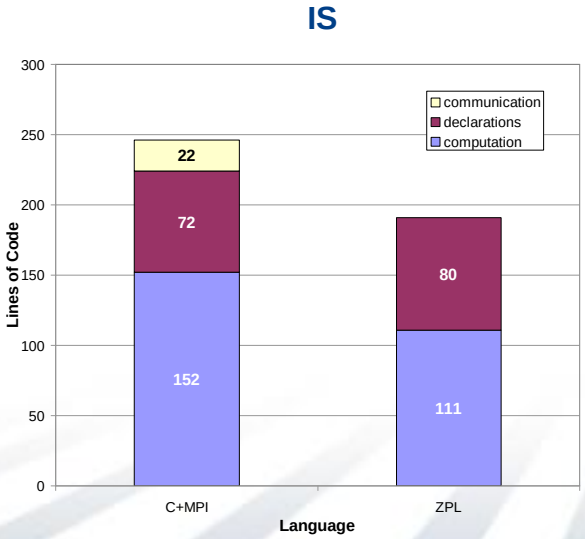
# ZPL's Lesson: Compact High-Level Code…

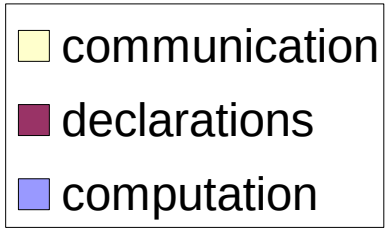


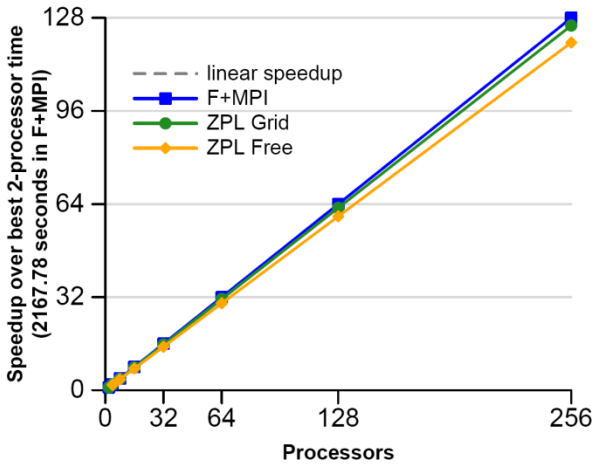



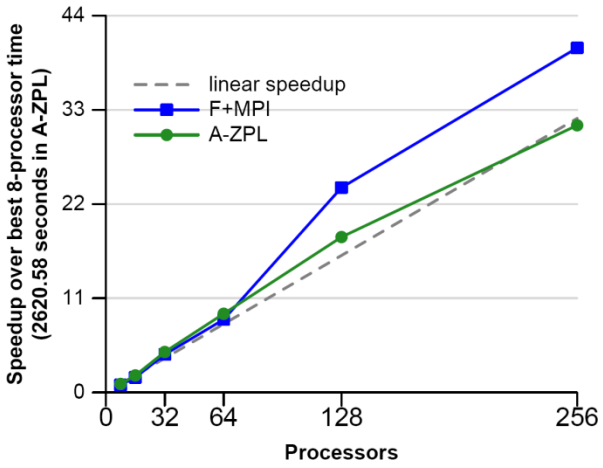



# …need not perform poorly

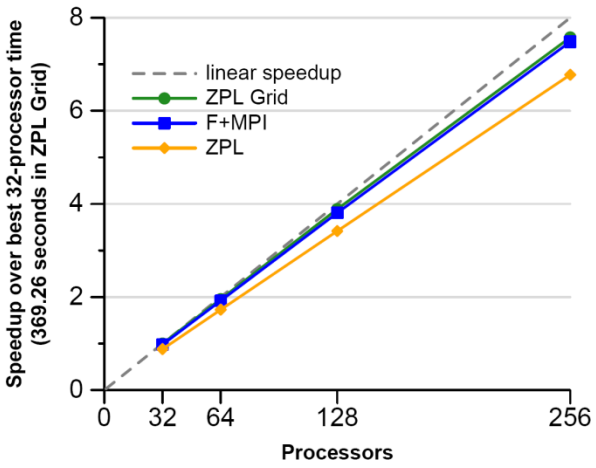
EP



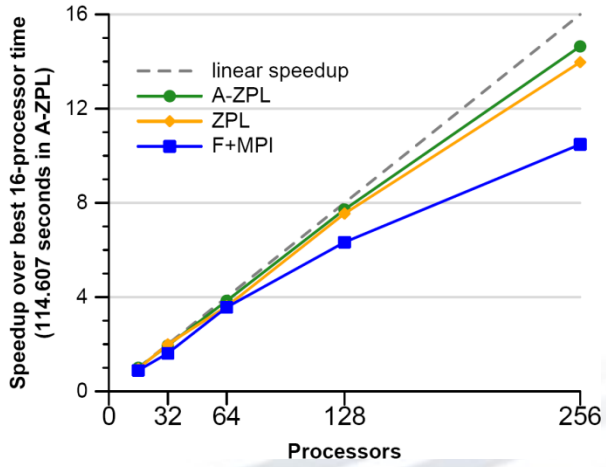

CG



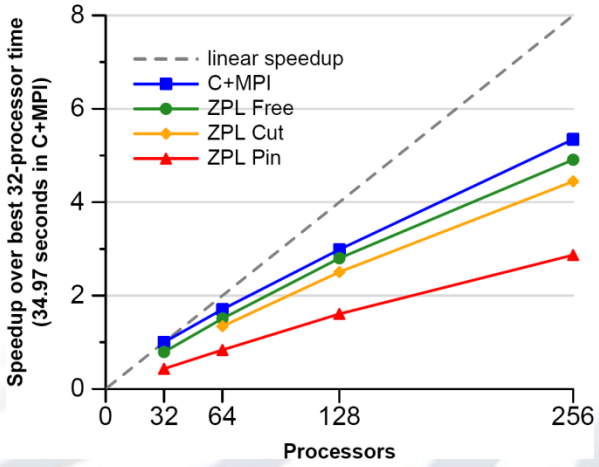

FT

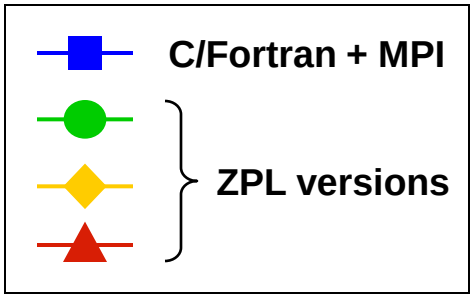



MG



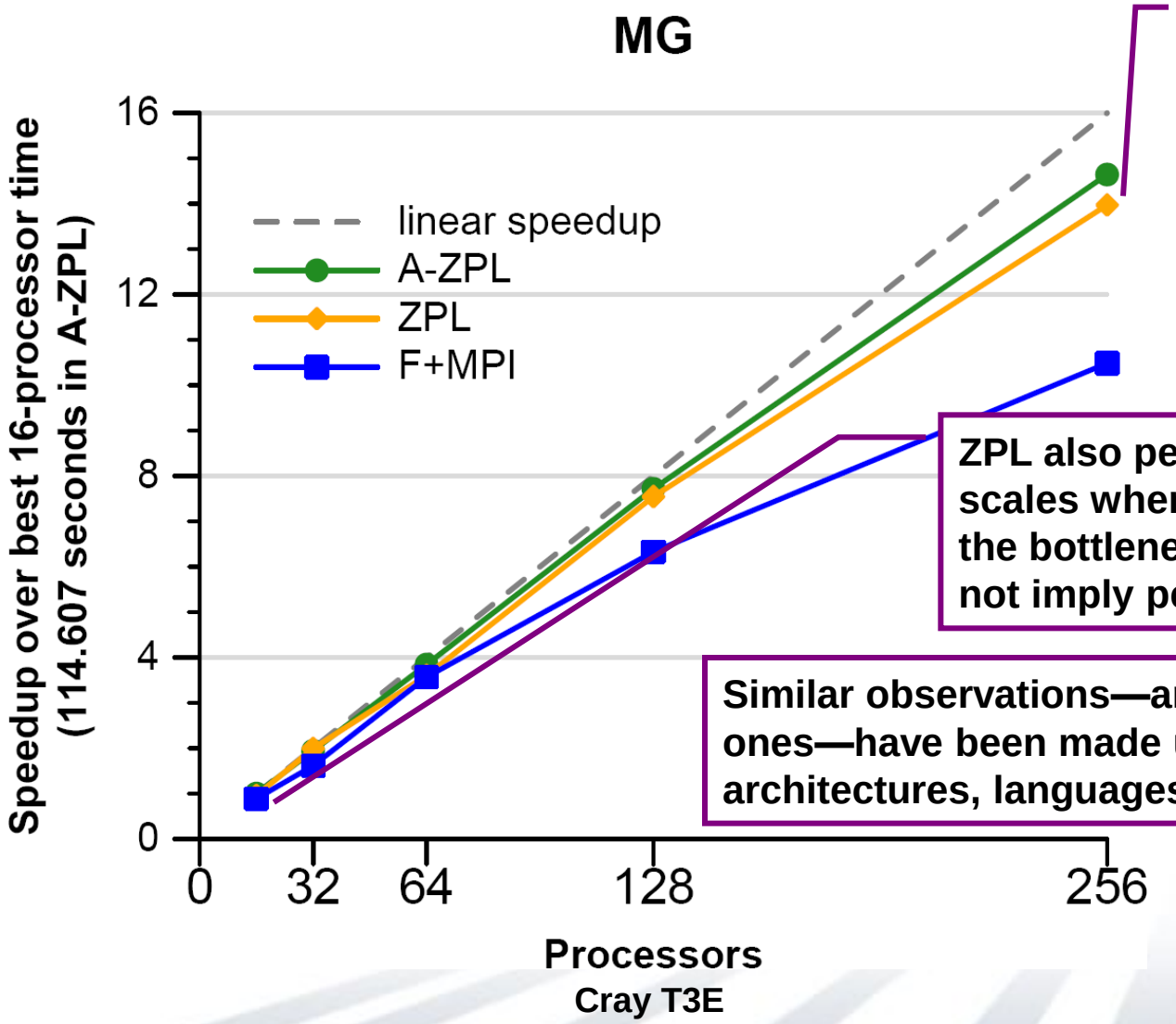

IS




**See also Rice University's recent D-HPF work…**

# NAS MG Speedup (MPI vs. ZPL)

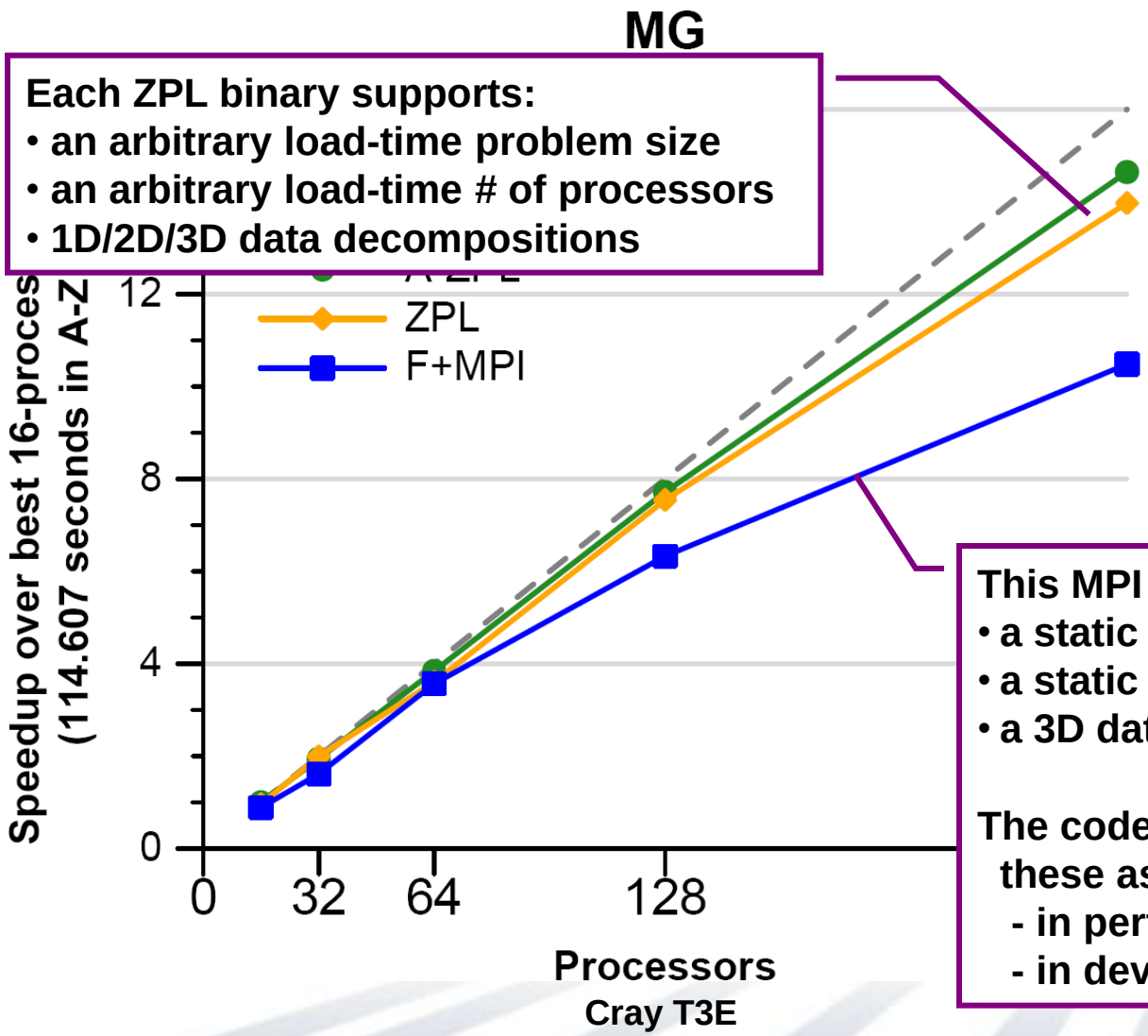



**ZPL scales better than MPI since its communication is expressed in an implementation-neutral way; this permits the compiler to use SHMEM on this Cray T3E but MPI on a commodity cluster**

**ZPL also performs better at smaller scales where communication is not the bottleneck ⇒ new languages need not imply performance sacrifices**

**Similar observations—and more dramatic ones—have been made using more recent architectures, languages, and benchmarks**

# NAS MG Speedup (MPI vs. ZPL)

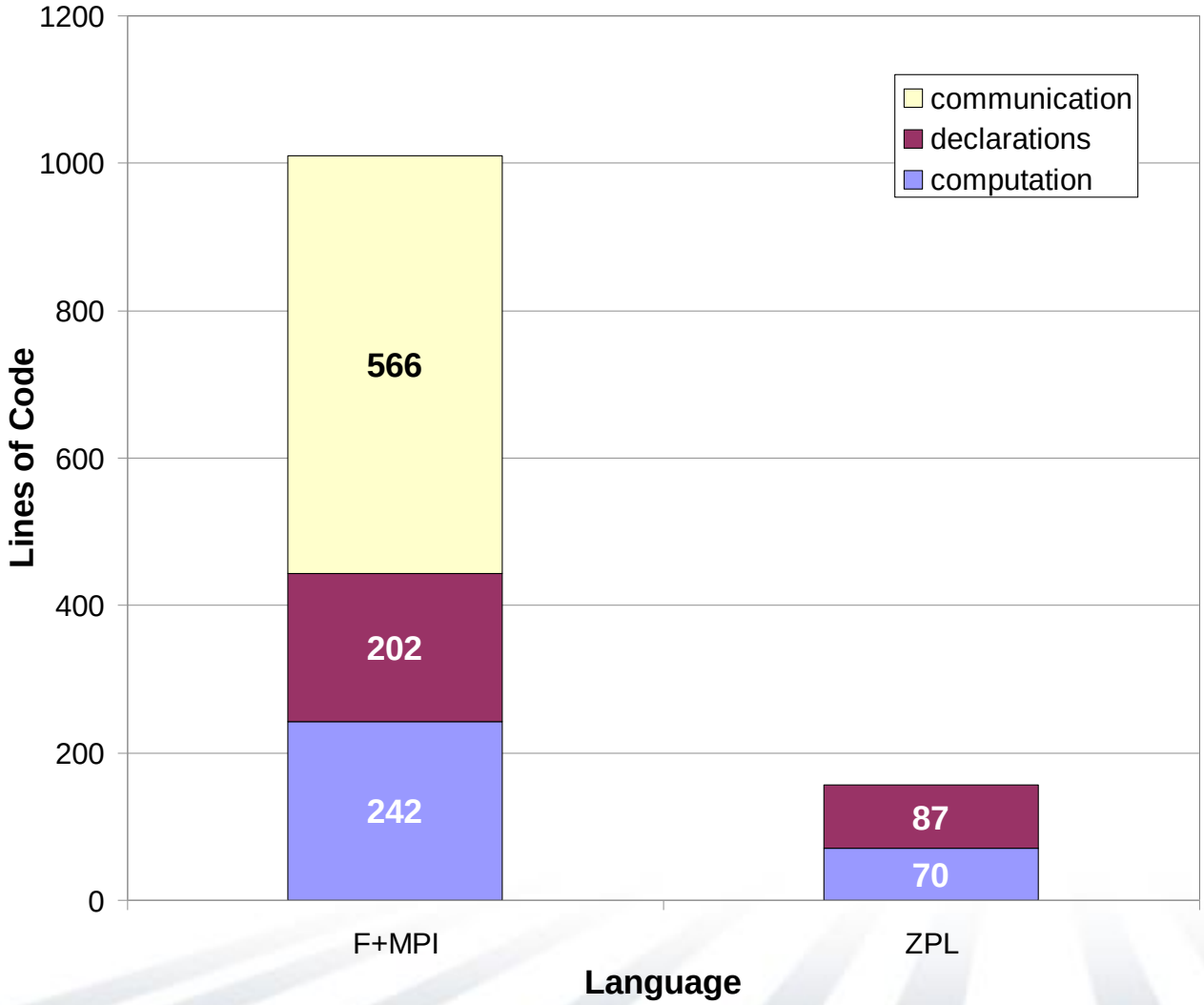



**Each ZPL binary supports:**
- **an arbitrary load-time problem size**
- **an arbitrary load-time # of processors**
- **1D/2D/3D data decompositions**

**This MPI binary only supports:**
- **a static 2**_k_ problem size**
- **a static 2**_j_ # of processors**
- **a 3D data decomposition**

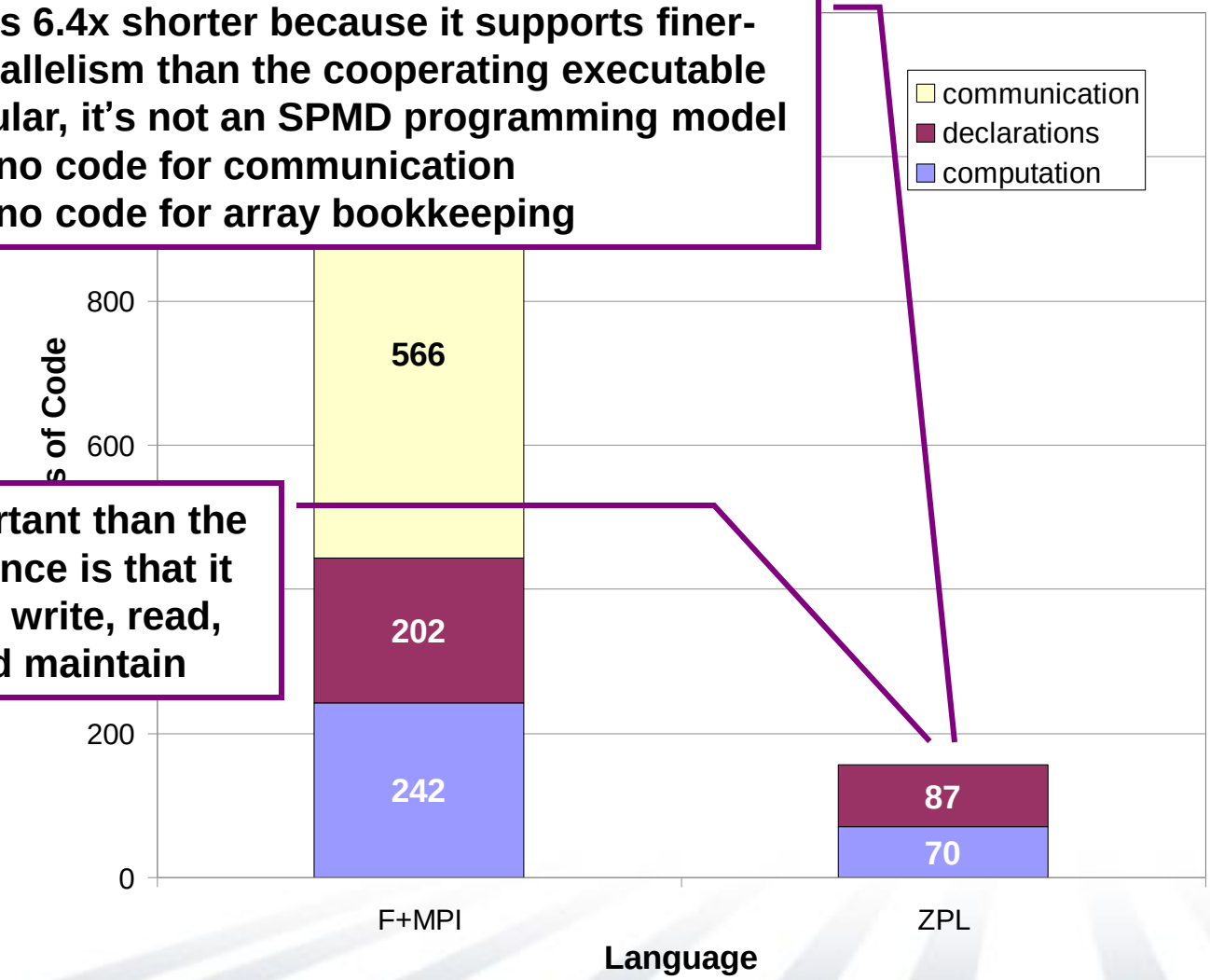
**The code could be rewritten to relax these assumptions, but at what cost?**
- **in performance?**
- **in development effort?**

# NAS MG code size (MPI vs. ZPL)



# NAS MG code size (MPI vs. ZPL)

- the ZPL is 6.4x shorter because it supports finer-grain parallelism than the cooperating executable
- in particular, it's not an SPMD programming model
  - ⇒ little/no code for communication
  - ⇒ little/no code for array bookkeeping




More important than the size difference is that it is easier to write, read, modify, and maintain

# Why Aren't We Done? (ZPL's Limitations)

- Only supports a single level of data parallelism
  - imposed by execution model: single-threaded SPMD
  - not well-suited for task parallelism, dynamic parallelism
  - no support for nested parallelism
- Distinct types & operators for distributed and local arrays
  - supports ZPL's WYSIWYG syntactic model
  - impedes code reuse (and has potential for bad cross-products)
  - annoying
- Only supports a small set of built-in distributions for arrays
  - e.g., Block, Cut (irregular block), …
  - if you need something else, you're stuck

# ZPL's Successes

- First-class concept for representing index sets
  - ⇒ makes clouds of scalars in array declarations and loops concrete
  - ⇒ supports global-view of data and control; improved productivity
  - ⇒ useful abstraction for user and compiler

  ***The Design and Implementation of a Region-Based Parallel Language*.** Bradford L. Chamberlain. PhD thesis, University of Washington, November 2001
- Semantics constraining alignment of interacting arrays
  - ⇒ communication requirements visible to user and compiler in syntax

  **ZPL's WYSIWYG performance model.** Bradford L. Chamberlain, Sung-Eun Choi, E Christopher Lewis, Calvin Lin, Lawrence Snyder, and W. Derrick Weathersby. In *Proceedings of the IEEE Workshop on High-Level Parallel Programming Models and Supportive Environments*, 1998.
- Implementation-neutral expression of communication
  - ⇒ supports implementation on each architecture using best paradigm

  **A compiler abstraction for machine independent parallel communication generation.** Bradford L. Chamberlain, Sung-Eun Choi, and Lawrence Snyder. In *Proceedings of the Workshop on Languages and Compilers for Parallel Computing*, 1997.
- A good start on supporting distributions, task parallelism

  Steven J. Deitz. ***High-Level Programming Language Abstractions for Advanced and Dynamic Parallel Computations*.** PhD thesis, University of Washington, February 2005.

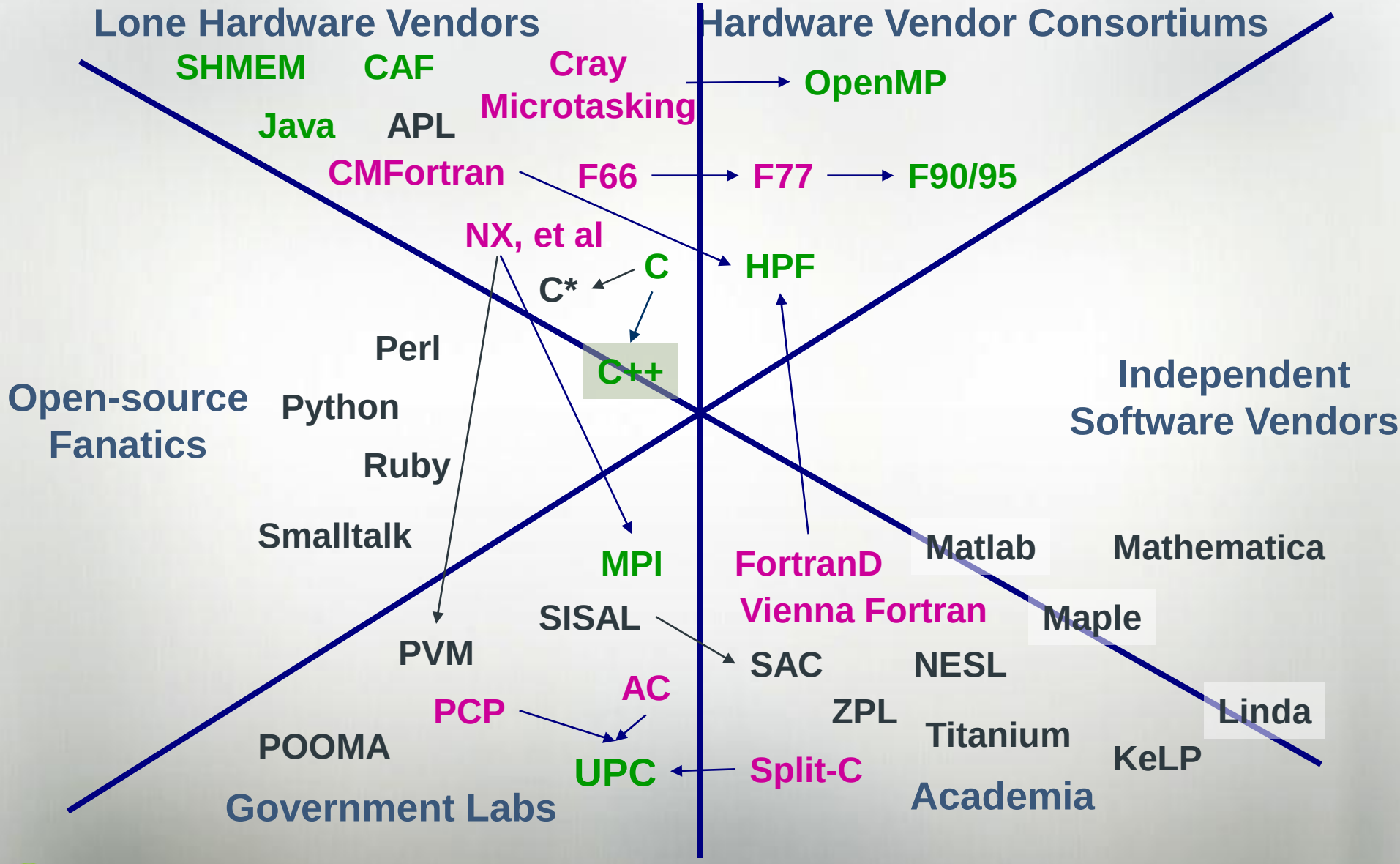
CRAY
THE SUPERCOMPUTER COMPANY

# Chapel's Genesis

# October 2002

- After a year out of school at a startup, I went to Cray
- This was an exciting time
  - Red Storm was just starting (precursor to XT, XE, XK lines)
  - HPCS was just starting
  - I was tasked with helping develop our SW Productivity story
    - How should we measure productivity?
    - What technologies should we pursue?
    - “Burton, can we do a language?” “No.”

# Where do Languages Come From?



CRAY
THE SUPERCOMPUTER COMPANY

# Chapel, phase I: Molten State (2003-2006)

# Chapel Timeline: Ramping Up

**Nov-Dec 2002:** Burton warms up to the concept of doing a language

**Jan 2003:** Cray first states interest in developing new languages to HPCS team

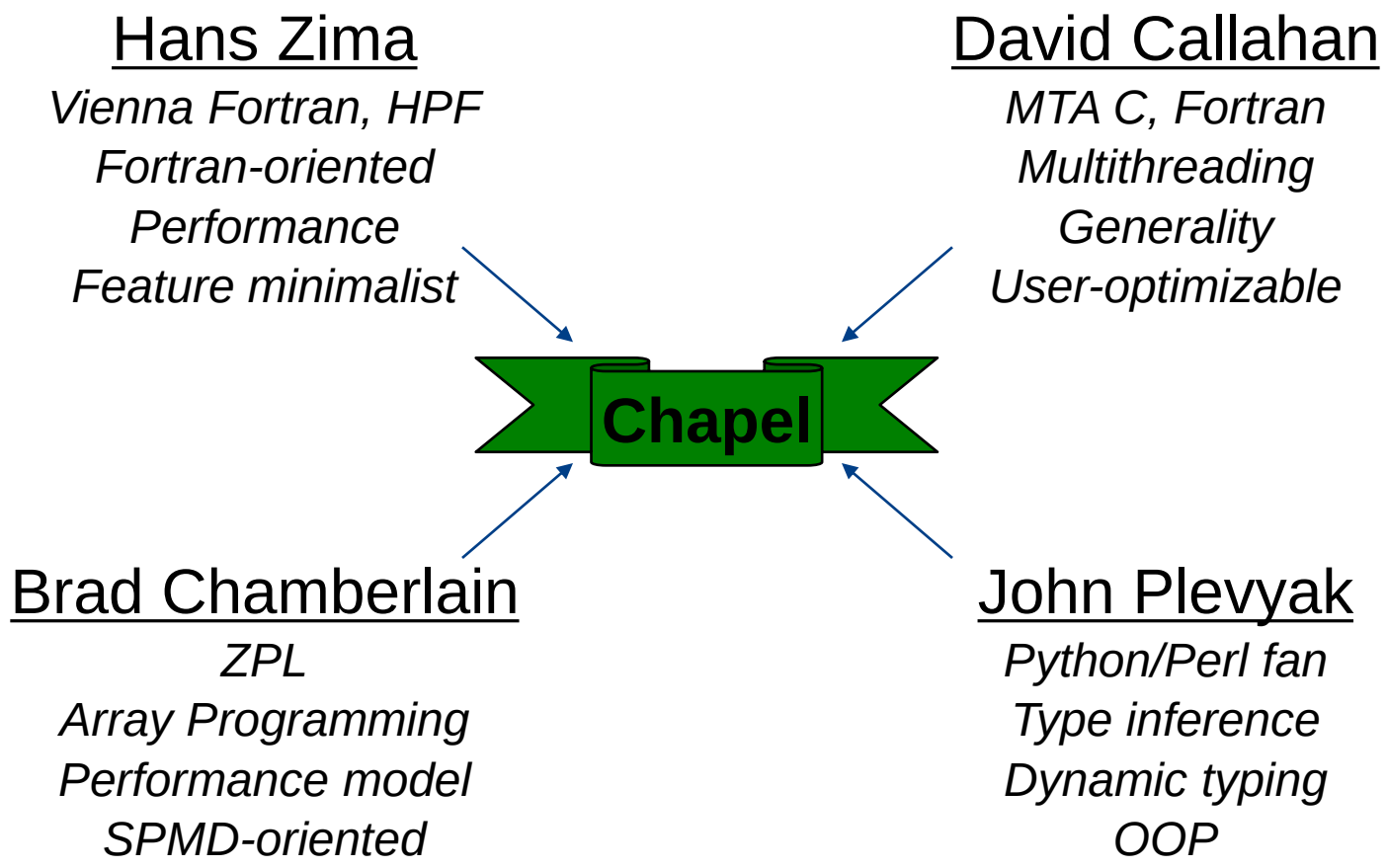
**Jul 2003:** HPCS phase II starts

Chapel name coined (Cascade High Productivity Language)

David Callahan, Hans Zima, and I form the initial Chapel team

**Sept 2003:** John Plevyak contracted to help with implementation

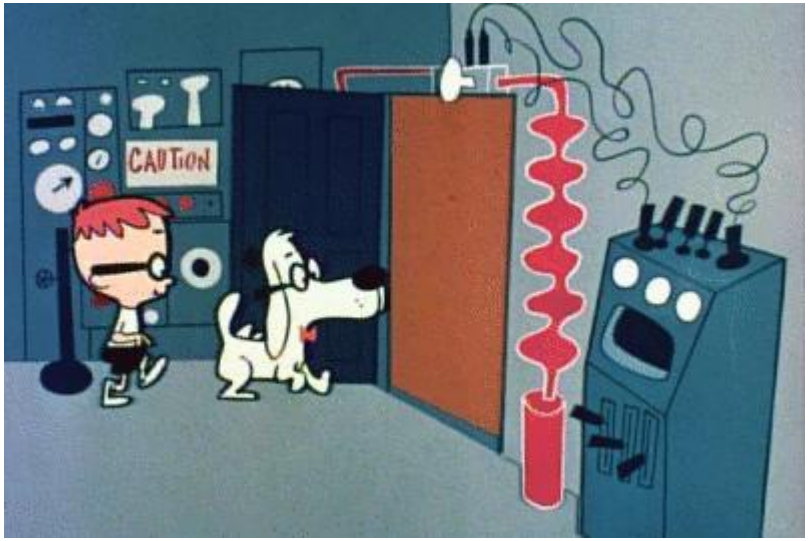
# Language Design Approach



# Chapel Timeline: Setting the Stage

**Oct 2003:** First public talk (I can find) mentioning Chapel (by Burton Smith)

# “Sherman, set the WABAC machine for 2003...”



# Concrete Chapel

- A modern base language
  - Strongly typed
  - Objected-oriented (classes, single inheritance)
  - Fortran-like array features
  - Module structure for name space management
  - Optional automatic storage management
- HPC Features
  - Aggregate operations
  - Explicit fine-grain parallelism
  - Locality management tools

*Support distributed data but don't require fragmented control* (simpler machine models)


(from "*Cascade: Toward Sustained Petaflops Computing*", Burton Smith, Oct 2003)

# Concrete Chapel

- A modern base language
  - Strongly typed
  - Objected-oriented (classes, single inheritance)
  - Fortran-like array features
  - Module structure for name space management
  - Optional automatic storage management
- HPC Features
  - Aggregate operations
  - Explicit fine-grain parallelism
  - Locality management tools

*Support distributed data but don't require fragmented control* (simpler machine models)


(from "*Cascade: Toward Sustained Petaflops Computing*", Burton Smith, Oct 2003)

# Abstract Chapel

- Extend Concrete Chapel with tools for generic programming (and more productive use)
  - Type inference
  - Function/class parameterization by type
  - Resolution of overloading via type constraints
- Also explore rapid prototyping tools
  - First-class functions
  - Aggregate temporaries with inferred implementations
  - Data structure extensions
- Concepts can be applied to other input languages but Chapel syntax will be motivated by this goal.

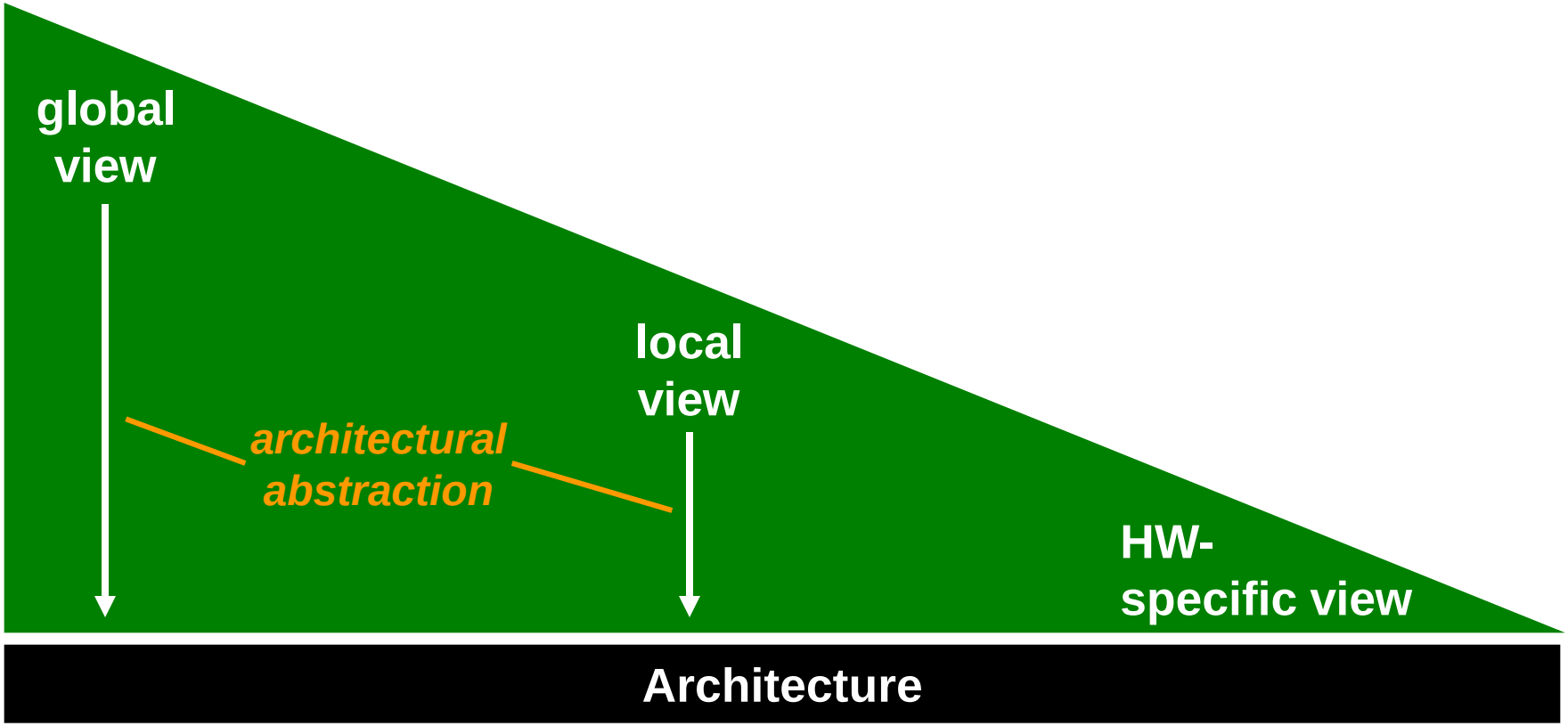

(from "*Cascade: Toward Sustained Petaflops Computing*", Burton Smith, Oct 2003)

# Abstract Chapel

- Extend Concrete Chapel with tools for generic programming (and more productive use)
  - Type inference
  - Function/class parameterization by type
  - Resolution of overloading via type constraints
- Also explore rapid prototyping tools
  - First-class functions
  - Aggregate temporaries with inferred implementations
  - Data structure extensions
- Concepts can be applied to other input languages but Chapel syntax will be motivated by this goal.


(from "*Cascade: Toward Sustained Petaflops Computing*", Burton Smith, Oct 2003)

# "The Language Ramp"




- provides easy entry point for novices
- allows experts to write 90% code quickly & easily

- supports lower-level parallel tuning, yet abstractly

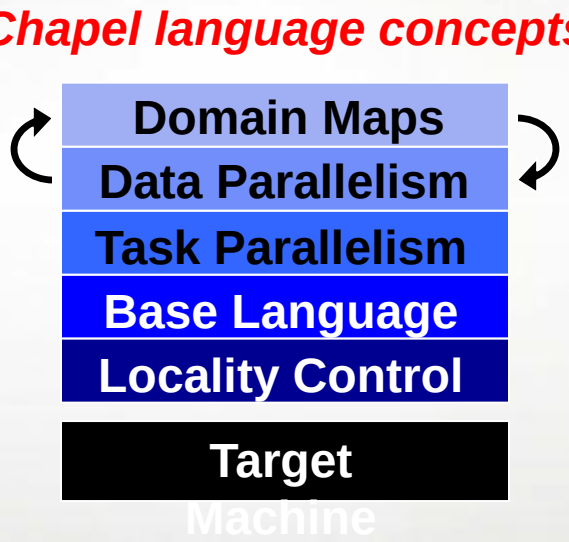
- supports tuning for architectural features
- should be avoided, left to compiler

(from my first(?) presentation at an HPCS review, Feb 2003)

# Multiresolution Design

***Multiresolution Design:*** Support multiple tiers of features

- higher levels for programmability, productivity
- lower levels for greater degrees of control




- build the higher-level concepts in terms of the lower
  - examples: array distributions and layouts; forall loop implementations
- permit the user to intermix layers arbitrarily

# Chapel Timeline: Baby Steps

**Jan 2004:** Chapel source repository created

**May 2004:** nightly regression testing starts

**Sept-Nov 2004:** Callahan writes draft of original language spec for discussion

**Oct 2004:** first Chapel paper published (HIPS 2004)

**Nov 2004:** Steve Deitz hires into Chapel team

…lots of wrestling over language concepts, choices, implementation…

# Example 3: Fast Multipole Method (FMM)

```
var OSgfn, ISgfn: [lvl in Levels] [SpsCubes[lvl]] [Sgfns[lvl]] [1..3] complex;
```

**1D array over levels of the hierarchy**

# Example 3: Fast Multipole Method (FMM)

```
var OSgfn, ISgfn: [lvl in Levels] [SpsCubes[lvl]] [Sgfns[lvl]] [1..3] complex;
```

1D array over levels of the hierarchy

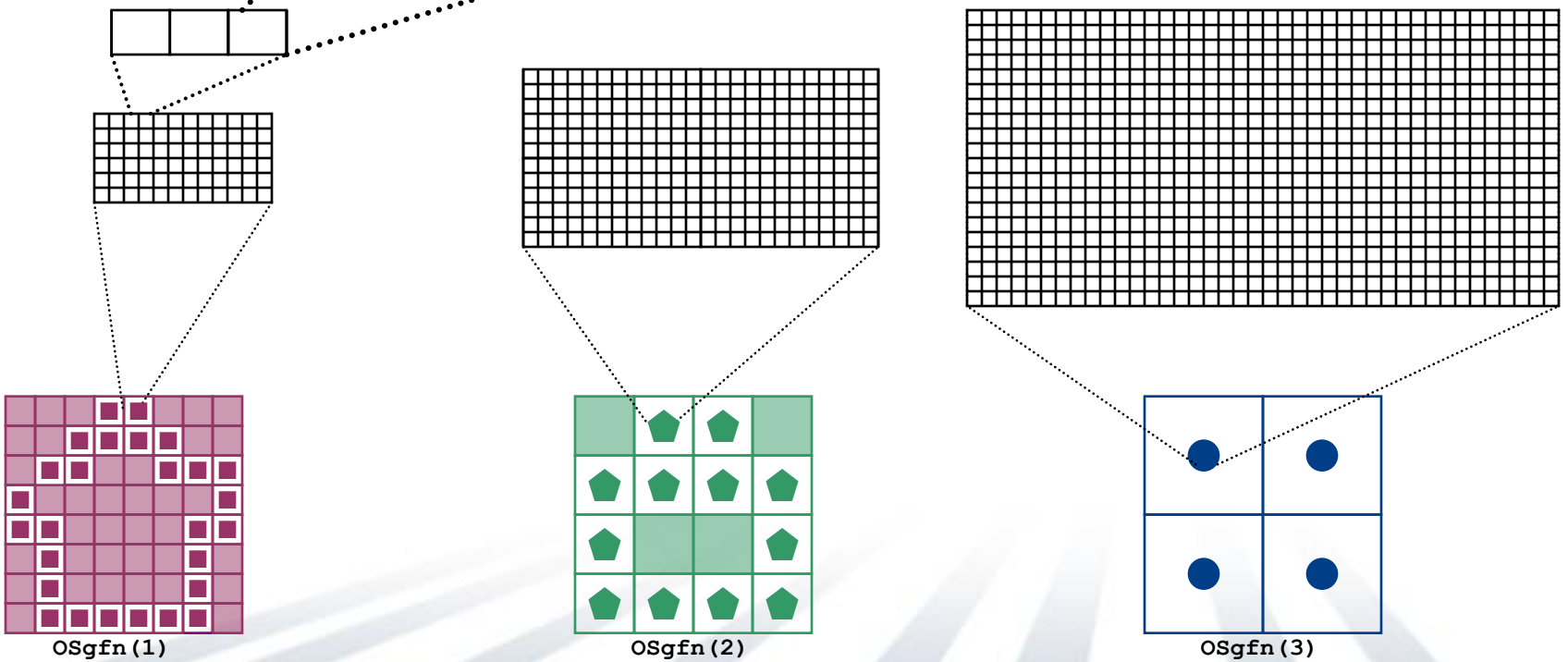
…of 3D sparse arrays of cubes (per level)

…of 1D vectors

…of 2D discretizations of spherical functions, (sized by level)

…of complex values

$x + y \cdot i$

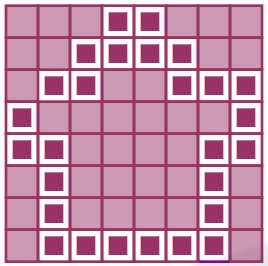


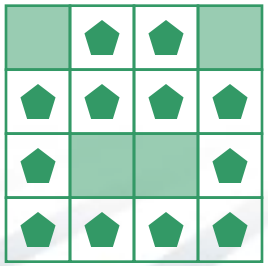
# FMM: Supporting Declarations

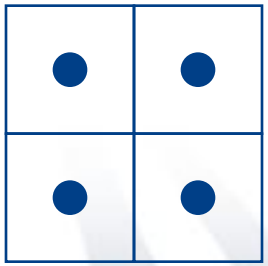
```
var OSgfn, ISgfn: [lvl in Levels] [SpsCubes(lvl)] [Sgfns(lvl)] [1..3] complex;
```

*previous definitions:*

```
var n: int = …;
var numLevels: int = …;

var Levels: domain(1) = [1..numLevels];

var scale: [lvl in Levels] int = 2**(lvl-1);
var SgFnSize: [lvl in Levels] int = computeSgFnSize(lvl);

var LevelBox: [lvl in Levels] domain(3) = [(1,1,1)..(n,n,n)] by scale(lvl);
var SpsCubes: [lvl in Levels] sparse subdomain(LevelBox) = …;

var Sgfns: [lvl in Levels] domain(2) = [1..SgFnSize(lvl), 1..2*SgFnSize(lvl)];
```



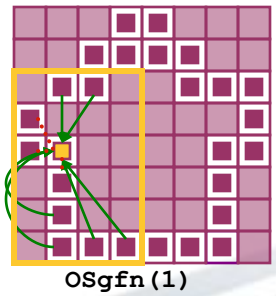



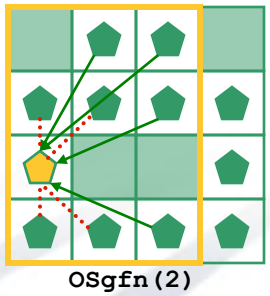
# FMM: Computation

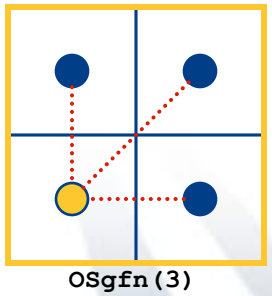
```
var OSgfn, ISgfn: [lvl in Levels] [SpsCubes(lvl)] [Sgfns(lvl)] [1..3] complex;
```

## *outer-to-inner translation:*

```
for lvl in [1..numLevels) by -1 {
  …
  forall cube in SpsCubes(lvl) {
    forall sib in out2inSiblings(lvl, cube) {
      const Trans = lookupXlateTab(cube, sib);

      atomic Isgfn[lvl][cube] += Osgfn[lvl][sib] * Trans;
    }
  }
  …
}
```

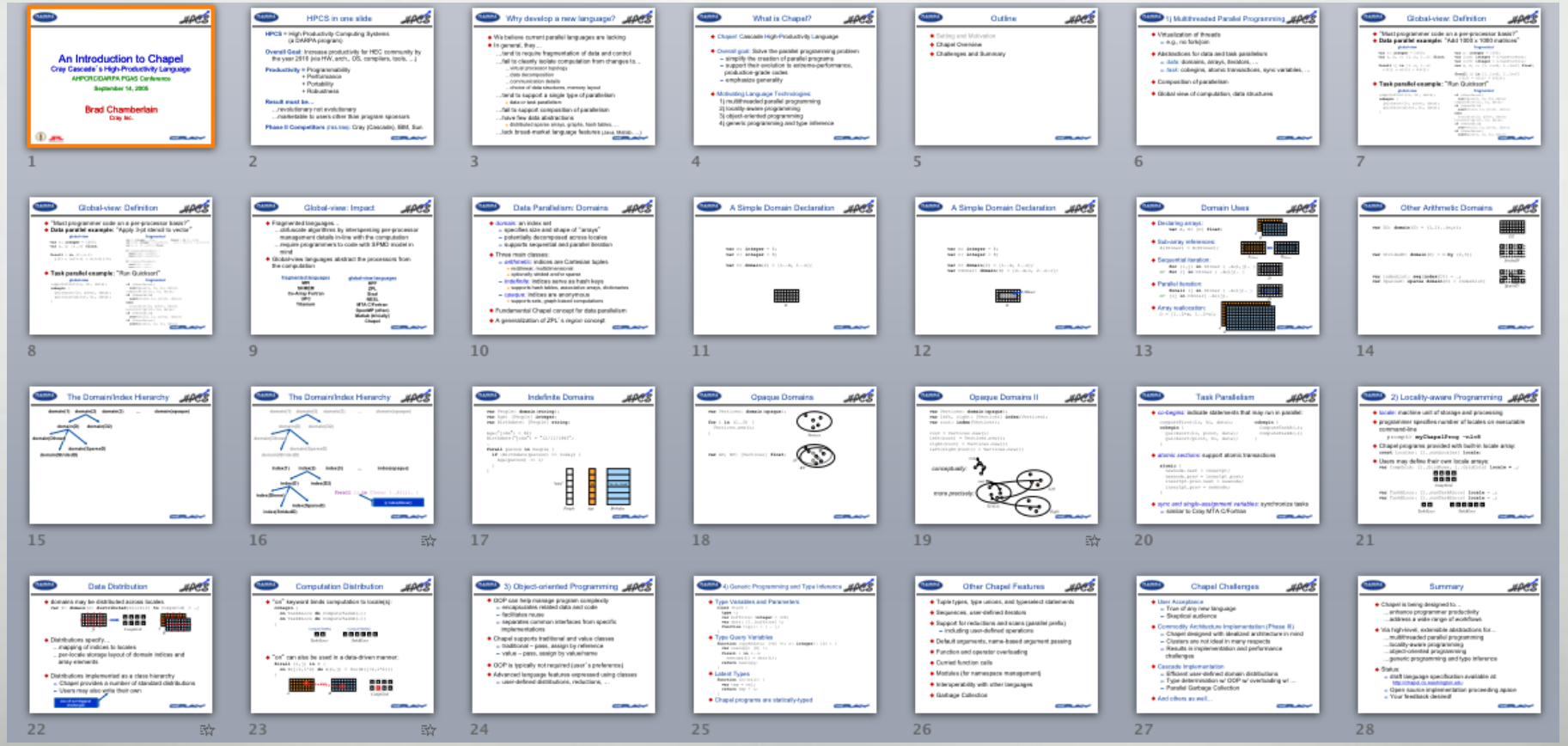


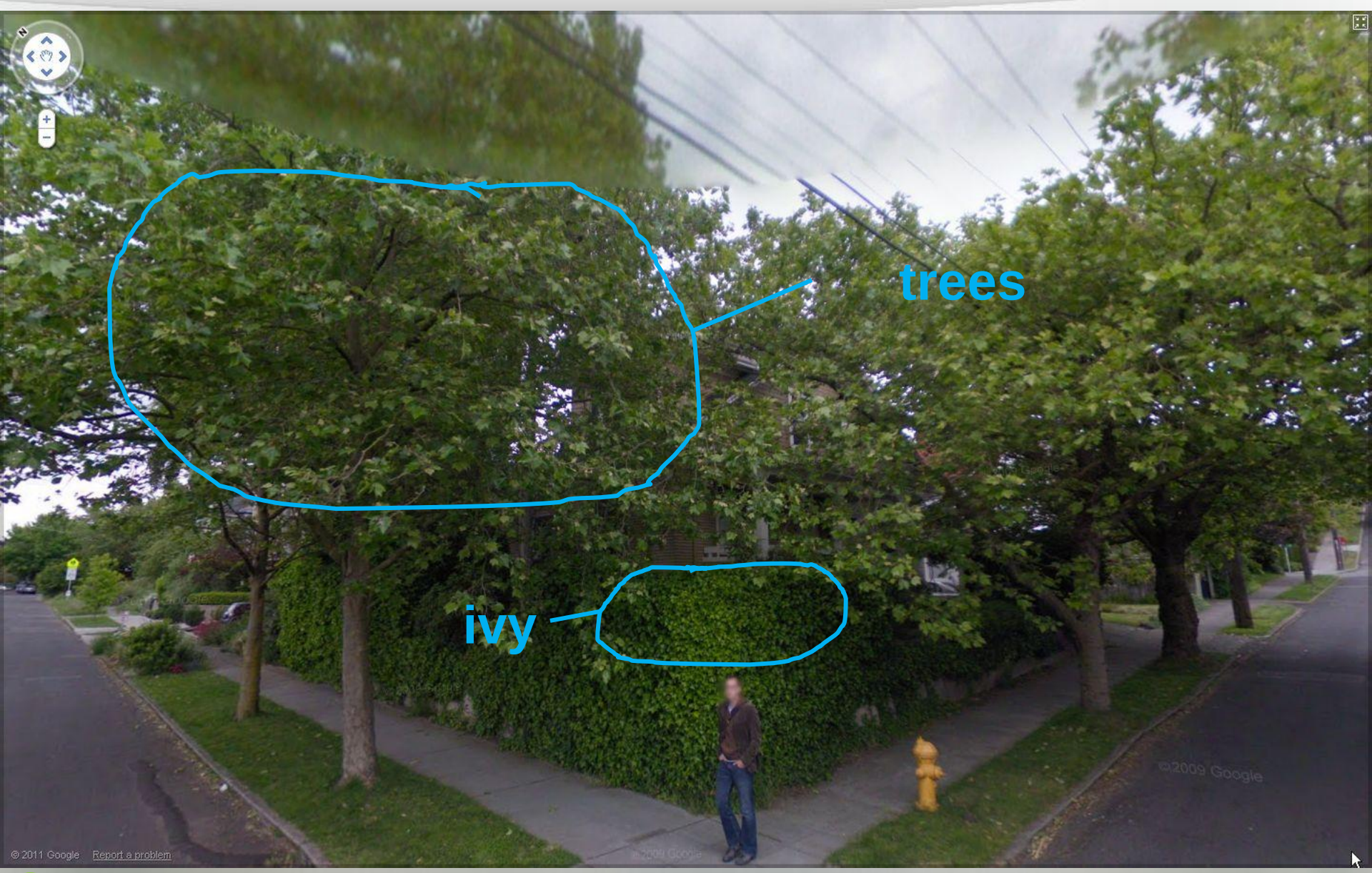



# Timeline: Spreading the Word

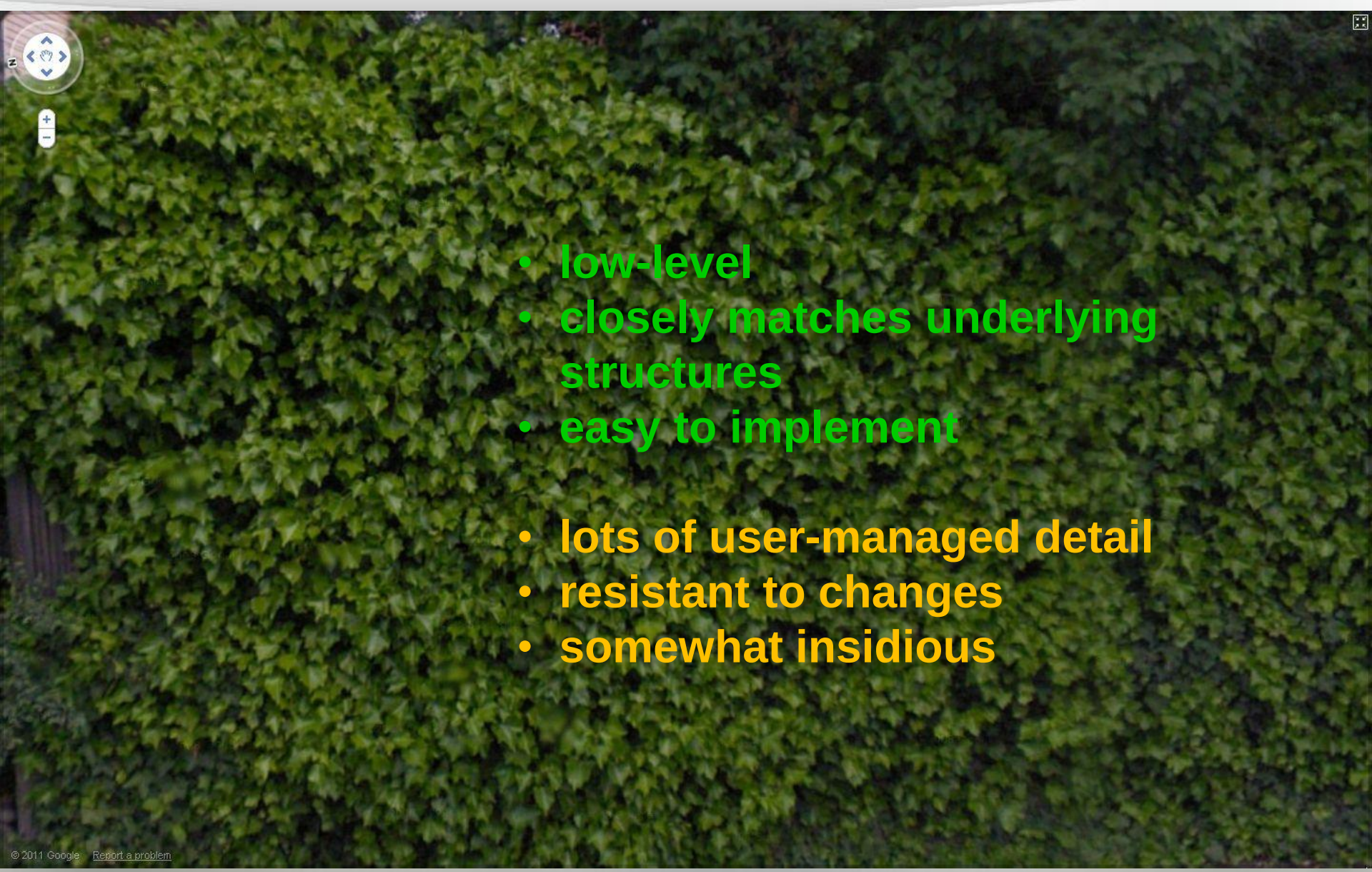
## First PGAS workshop (2005): Chapel overview talk



# A Seattle Corner



# Ivy




- low-level
- closely matches underlying structures
- easy to implement

- lots of user-managed detail
- resistant to changes
- somewhat insidious

# Trees


- higher-level
- more elegant, structured

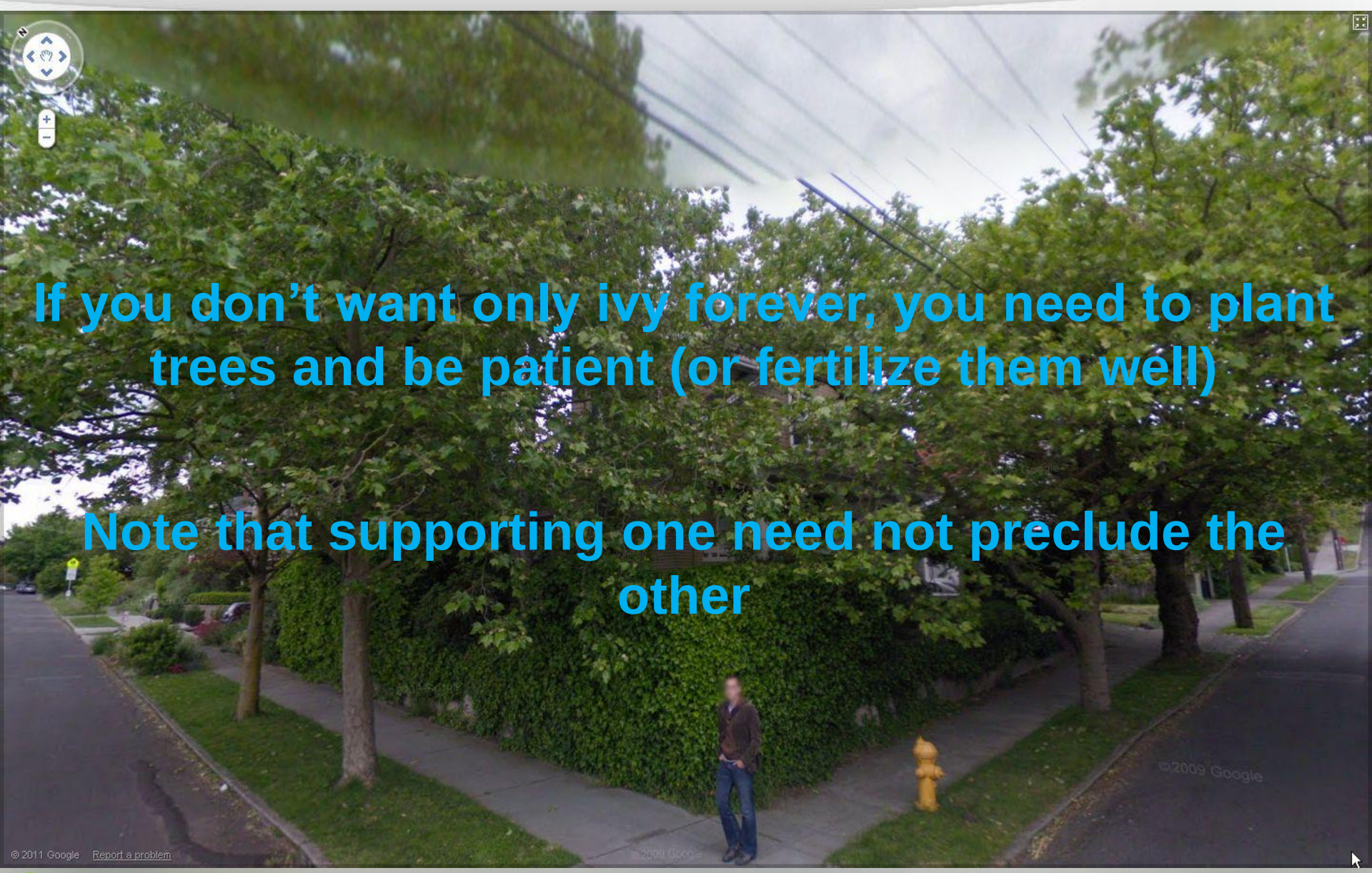
- requires a certain investment of time and force of will to establish

# Landscaping Quotes from the HPC community

## Early HPCS years:

- "The HPC community tried to plant a tree once. It didn't survive. Nobody should ever bother planting one again."
- "Why plant a tree when you can't be assured it will grow?"
- "Why would anyone ever want anything other than ivy?"
- "We're in the business of building treehouses that last 40 years; we can't afford to build one in the branches of your sapling."
- "This sapling looks promising. I'd like to climb it now!"

# A Corner in Seattle: Takeaways



# Chapel Timeline: The Breaking of the Fellowship

**Oct 2005:** David Callahan and Burton Smith leave Cray

**Mar 2006:** academics leave the project as Cascade becomes a Cray-only effort

**June 2006:** John Plevyak's contract runs out

CRAY
THE SUPERCOMPUTER COMPANY

# Chapel, phase II: Traction (2006-2008)

# Chapel Timeline: Excerpts

**Apr 2006:** first task-parallel, shared-memory codes running

**June-July 2006:** made three key design changes:

moved from deep to shallow type inference

dropped multiple-dispatch OOP in favor of single-dispatch

switched to current compiler architecture

== TRACTION!!!

**Fall 2006:** first draft of current language spec written

**Oct 2006:** second PGAS workshop (see next slide)

**Nov 2006:** first HPCC entry: elegant, correct, but tons of memory leaks

**Dec 2006:** first release (request-only basis)

# PGAS: What's in a Name?

| | | *memory model* | *programming model* | *execution model* | *data structures* | *communication* |
|---|---|---|---|---|---|---|
| | **MPI** | distributed memory | cooperating executables (often SPMD in practice) | | manually fragmented | APIs |
| | **OpenMP** | shared memory | global-view parallelism | shared memory multithreaded | shared memory arrays | N/A |
| PGAS Languages | **CAF** | PGAS | Single Program, Multiple Data (SPMD) | | co-arrays | co-array refs |
| PGAS Languages | **UPC** | PGAS | Single Program, Multiple Data (SPMD) | | 1D block-cyc arrays/ distributed pointers | implicit |
| PGAS Languages | **Titanium** | PGAS | Single Program, Multiple Data (SPMD) | | class-based arrays/ distributed pointers | method-based |
| | **Chapel** | PGAS | global-view parallelism | distributed memory multithreaded | global-view distributed arrays | implicit |

(second PGAS workshop 2006)

# Chapel Timeline: The Milestone Years

**January 2007:** HPLS (HPCS language evaluation team) meeting at Rice

**Mar-May 2007:** first competitive serial 1-locale runs of HPCC Stream and RA

**July 2007:** first distributed-memory task parallel programs start working

**March 2008:** first executions on a Cray XT

**Sept-Oct 2008:** first data-parallel codes running (shared- and dist- memory)

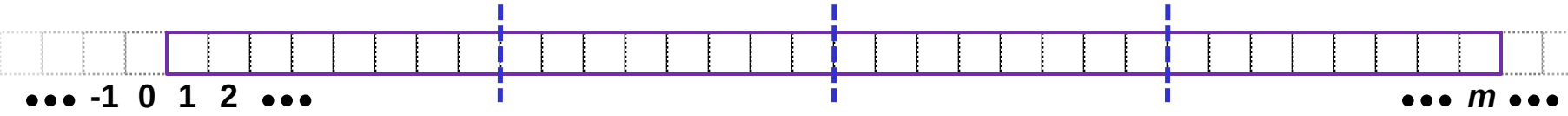
**November 2008:** first SC tutorials (one standalone, one with UPC/X10)

three-way tie for “most elegant” in HPCC

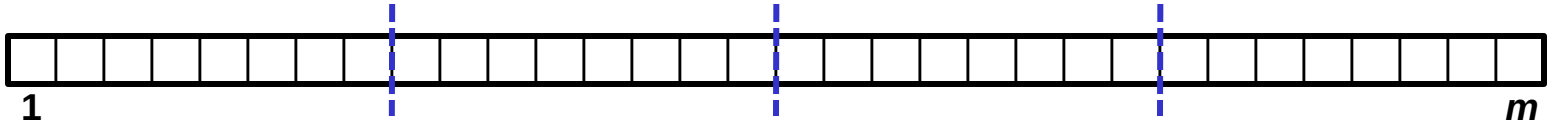
first public release of Chapel

# STREAM Triad in Chapel

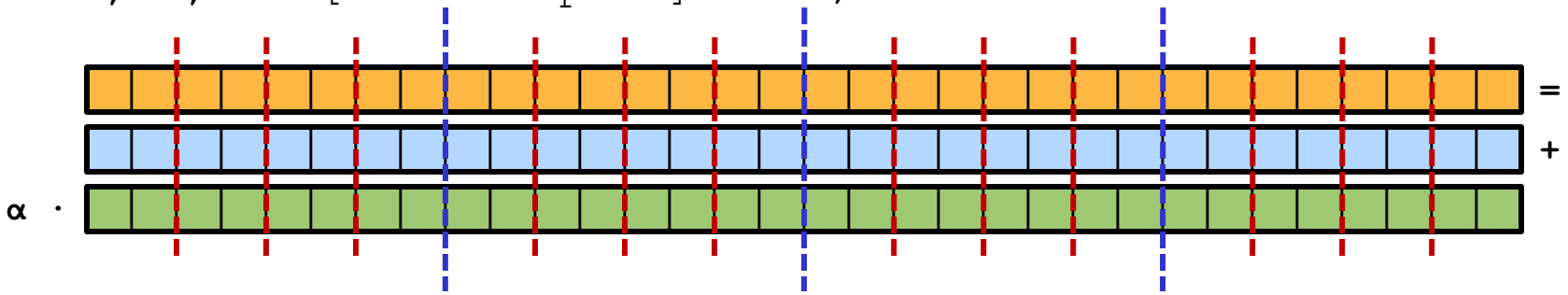
```
const BlockDist = new Block1D(bbox=[1..m], tasksPerLocale=…);
```




```
const ProblemSpace: domain(1, int(64)) distributed BlockDist
                    = [1..m];
```




```
var A, B, C: [ProblemSpace] real;
```

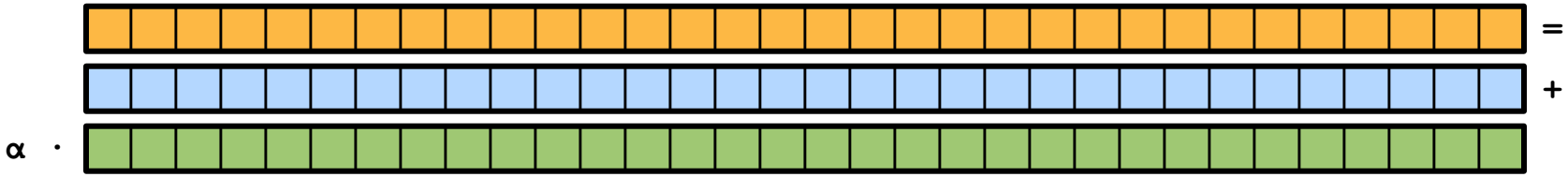



```
forall (a, b, c) in (A, B, C) do
  a = b + alpha * c;
```

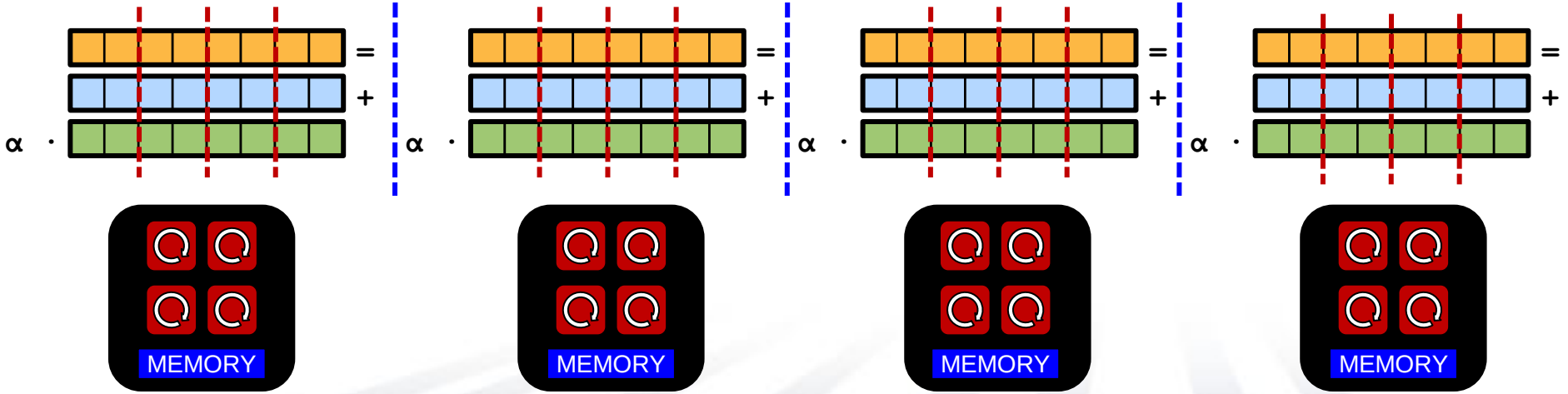
# Chapel Distributions

***Distributions:*** "Recipes for parallel, distributed arrays"

- help the compiler map from the computation's global view…




…down to the *fragmented*, per-processor implementation



# Chapel Distributions

- (Advanced) Programmers can write distributions in Chapel
- Chapel will support a standard library of distributions
  - *research goal:* using the same mechanism that users would
- Block1D is our first such distribution
  - *our compiler has no semantic knowledge of block distributions*
  - only of a distribution's interface--how to…
    - …create domains and arrays using that distribution
    - …map indices to locales
    - …access array elements
    - …iterate over indices/array elements
      - sequentially
      - in parallel
      - in parallel and zippered with other parallel iteratable types
    - …and so forth…

CRAY
THE SUPERCOMPUTER COMPANY

# Chapel, phase III: Emergence (2009-2012)

# Chapel Timeline: Going Public

**February 2009:** First Chapel executions on an IBM system

**April 2009:** Chapel source hosting moves from UW (internal) to SourceForge

**May 2009:** Michael Ferguson, user extraordinaire, arrives on the scene

**June 2009:** Sung-Eun Choi joins to the Chapel team

**July 2009:** Chapel website moved from UW to http://chapel.cray.com

**November 2009:** Won "most productive" entry at HPC Challenge competition

…lots of capabilities and improvements; collaborations; user requests…

# Chapel Timeline: Reaching Out

**November 2010:** first Chapel Users Group (CHUG) happy hour

**August 2011:** Chapel logo unveiled

**September 2011:** start planning for post-HPCS

**November 2011:** first Chapel SWAG (USB with Chapel logo)

first Chapel lightning talks BoF at SC12

**December 2011, March 2012:** LULESH work with Jeff Keasler

**October 2012:** final Chapel presentation & demo under HPCS

# Example Pair-Programming Study: LLNL/LULESH

**Apr 2011:** LLNL expresses interest in Chapel at Salishan
- made us aware of the LULESH benchmark from DARPA UHPC

**Summer 2011:** Cray intern ports LULESH to Chapel
- *caveat:* used structured mesh to represent data arrays

**Nov 2011:** Chapel team tunes LULESH for single-node performance

**Dec 2011:** Chapel team visits LLNL (talk, tutorial, 1-on-1 sessions)

**Mar 2012:** Jeff Keasler (LLNL) visits Cray to pair-program
- in one afternoon, converted from structured to unstructured mesh
- impact on code minimal (mostly in declarations) due to:
  - domains/arrays/iterators
  - rank-independent features

**Apr 2012:** LLNL reports on collaboration at Salishan

**Apr 2012:** Chapel 1.5.0 release includes current version of LULESH

**Next steps:** distributed sparse domains, improved scalability

# Outline

- ✓ Context
- ✓ Chapel Under HPCS
- ➢ Chapel Today
- ● Chapel's Future

# The Elephant in the Room: Chapel Performance

- Chapel performance is continually improving
- Yet it's still not where it would need to be to supplant MPI or UPC
- For many potential users, this is their major (only?) barrier to using Chapel

# Slow but steady progress

Chapel performance and scalability *are* improving:

- e.g., over the past 5 months, SSCA#2 in Chapel:
  - is running on a graph that has 65,536x more vertices
  - is running on 23.5x more compute nodes
  - results in a 2333x faster TEPs rate
  - moved from Cray XE6™ to Cray Cascade™
- we're not done, but not stuck in the mud either

# In defense of Chapel's performance

- Performance has a multiplicative characteristic
  - it doesn't take many 0's to kill things…
- An aggressive parallel language is going to take at least as much effort as a more minimal one
- What would a revolutionary new language look like as it approached?

  (hint: it would not spring, fully formed, from the forehead of Zeus)
- Judge Chapel not based on what you get today, but on what a compiler will be able to produce

# Chapel: By the Numbers

**20,683:** commits against the repository

**4552:** downloads of public Chapel releases

**930:** emails sent to chapel-users

**174:** unique mailing list subscribers (non-Cray Chapel team)

**~144:** Chapel talks given during the HPCS program
(72 workshops/conferences, 32 milestone reviews, 16 academic, 16 government, 8 industry)

**~24:** notable collaborations
(10 lab, 10 academic, 4 int'l)

**15:** Chapel tutorials
(5 SC, 4 European, 3 gov't, 3 CUG)

**12:** major releases
(4 request-only, 8 public)

**0:** language changes due to Cascade architectural mods

# Chapel's Greatest Hits

- Multiresolution Language Philosophy
- User-Defined Parallel Iterators, Layouts, and Distributions
- Distinct Concepts for Parallelism and Locality
- Multithreaded Execution Model
- Unification of Data- and Task-Parallelism
- Productive Base Language Features
  - type inference, iterators, tuples, ranges
- Portable Design, Open-Source Implementation
  - Yet, able to take advantage of Cascade-specific features
- Revitalization of Community Interest in Parallel Languages

# Chapel Achievements Unlocked

**"Disordered Behavior":** Change LULESH's grid from regular to irregular in a few hours of work and a few dozen lines of code

**"Dynamic Duo":** Implement OpenMP-style dynamic loops with no compiler changes

**"Mmmm… dogfood":** Implement core features within Chapel

- (ranges, locales, tuples, sync/single variables, operators, …)

**"DIY Data":** Define all arrays using the same mechanism a user would

- a C-style array used in just one base case

**"Plug-and-play":** Trivially support novel runtime implementations and switching between them

**"Never met a computer I didn't like":** Run on any UNIX-like system

(with much credit due to C99, POSIX threads, GASNet)

# Chapel: Lessons Learned

- If we, as language developers want to do something, eventually an end-user will want to as well
- Uniform, undifferentiated tasks are overly simplistic
- Multicore NUMA nodes: bigger impact than anticipated
- Good research prototypes should anticipate productization
- Don’t put off performance/scalability work too long
- Don’t underestimate effort required to support early users

# Chapel Testimonials

*"Chapel is a maintainable future-proof language. With additional back-end performance enhancements, we would be using it to develop science codes, with an eye towards multiphysics production codes."*

**- Jeff Keasler, ASC code developer, LLNL**

*"After 8 years of observing the Chapel project from the outside while working with X10, I've now had the opportunity to start working with Chapel this year. The experience has been very positive for my research group. We have been able to come to up to speed with the Chapel compiler and runtime infrastructure very quickly. On the language front, we've found Chapel's data-parallel constructs to be extremely elegant. However, we see opportunities to broaden the task-parallel constructs, which is the focus of our current research related to Chapel."*

**- Vivek Sarkar, Professor, Rice University**

*"Chapel taught me to use hash tables before I had the slightest idea what they were. It convinced me to think of a multidimensional iteration as a single operation rather than a slew of micromanaged details. It suggested that when I'm stuck on a problem, I should think carefully about my index sets, a lesson that has served me well many times over. Chapel improved my productivity not only when writing Chapel, but also upon returning to traditional languages, by showing me how to see through the details.*

*It's a beautiful language full of excellent programming advice, perfect when I was a mathematician with little formal training, and something I miss every time I tear into gory C++ now as a professional software engineer."*

**- Jonathan Claridge, Google (PhD, UW AMath)**

# Chapel Testimonials

*"Chapel's well-thought-out language design and its modular implementation made it an ideal target for plugging in our task parallel library work. Chapel turned out to be not only an excellent standalone effort, but also a valuable platform for world-wide collaboration."*

**- Kenjiro Taura, Associate Professor, University of Tokyo**

*"In association with the department of Pre-College Programs at the University of Notre Dame, the Center for Research Computing (CRC) runs a two week summer school in high-performance computing for high-school students from across the nation. For the last two years we have introduced the high-school students to parallel programming (and for a lot of students an introduction to programming in general) via the Chapel programming language from Cray Inc. After just 5 hours of instruction, the majority of students had not only understood the basics of Chapel but also the important concepts of task and data parallelism to the point were they are able to implement their own parallel solutions to a set of challenge questions. Furthermore, the Chapel session ends with a benchmarking competition were teams are to parallelize a vector-vector addition problem from a serial template solution. In less than one hour all teams were able to submit two sets of solutions and timings corresponding to single node parallelization using the Chapel data-parallel 'forall' construct as well as multi-node parallelization using Chapel's data-distribution features. I believe it would be impossible for a group of inexperienced high-school students to achieve this rapid progress and understanding without the benefit of Chapel language's expressibility and productivity strengths. Having taught programming at college level for over 10 years, Chapel is at the top of my list as a language for introducing students to the art of both sequential and parallel programming. In all seriousness, it should be the first language used to introduce programming on every computing curriculum!"*

**- Tim Stitt, Research Assistant Professor, University of Notre Dame**

*"I remain consistently optimistic about Chapel. Over time, the quality of the language and the implementation has been steadily increasing. […] In my opinion, the current prototype for Chapel is at the forefront of parallel programming languages. At the same time, Chapel is an ambitious effort that will require more work."*

**- Michael Ferguson, Researcher, LTS**

# Heartfelt thanks!

- To the HPCS program for providing the very unique opportunity to work on an incredibly rewarding project
- To the members of the Chapel team, past and present, within Cray and externally
- To the broader Chapel community for their support and enthusiasm
- To the PGAS community for including us

# Outline

- ✓ Context
- ✓ Chapel Under HPCS
- ✓ Chapel Today
- ➢ Chapel's Future

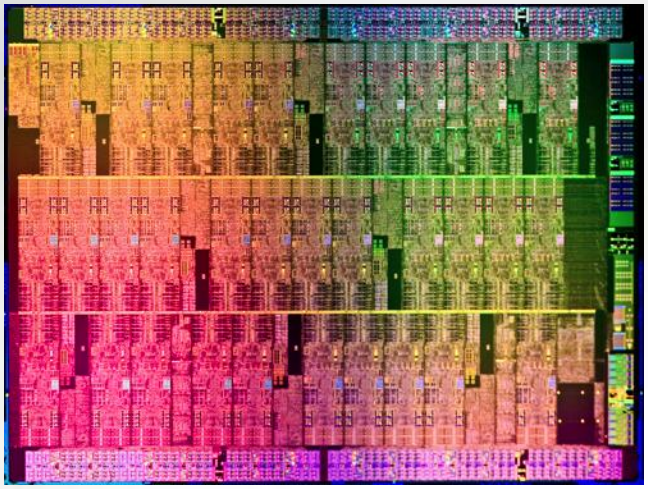
# What's Next? (immediate future)

- **Cut 1.6.0 release** (Oct 18th)
- **Prepare for SC12** (Nov)
  - HPC Challenge entry
  - Lightning Talks 2012 BoF
  - Chapel Tutorial
  - Chapel Overview in HPC Educators Forum (non-Cray)
- **Develop user-requested Improvements**
  - scalar performance improvements
  - reference counting
  - string improvements
- **Define next-stage project model**

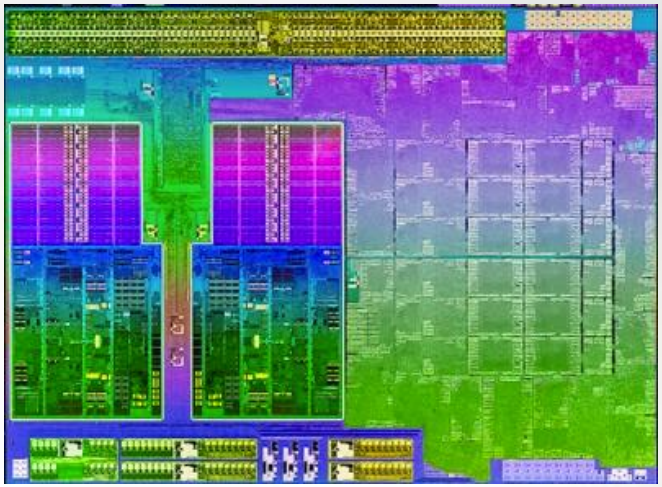
# Possible Future Directions

- **Continue to improve performance**
  - scalar performance
  - communication aggregation (stencils, remaps, reductions, …)
  - focus on codes with fewer degrees of freedom than SSCA#2
- **Fill in language gaps**
  - task teams, collectives, eurekas, task-private variables
  - hierarchical locales, support for next-generation compute nodes
  - exceptions, resilience features
  - additional distributions, partial reductions, …
- **Identify strategic partners to aid with adoption**
- **Exascale/Next-Generation Architectures**

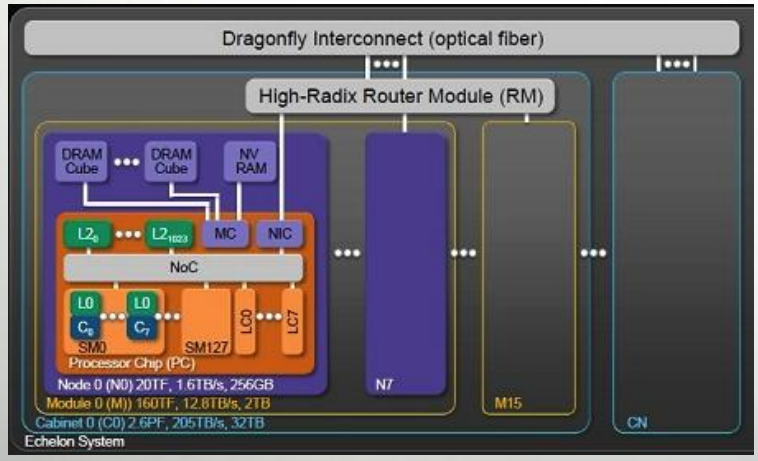
# Prototypical Next-Gen Processor Technologies



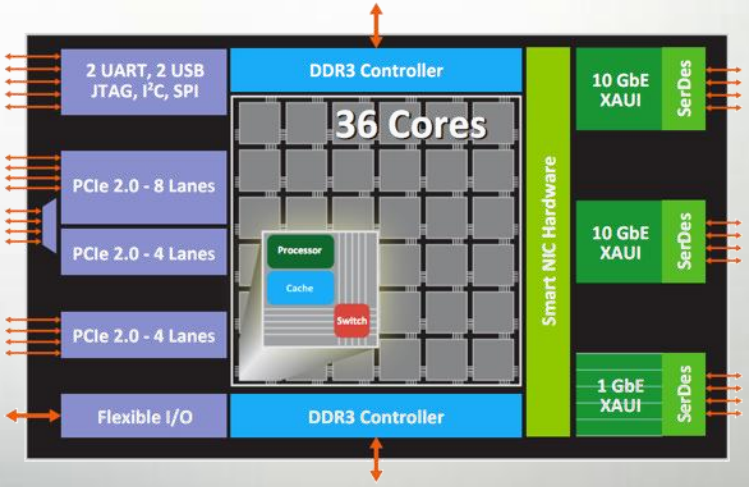
Intel MIC



AMD Trinity



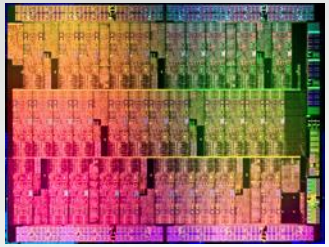

Nvidia Echelon

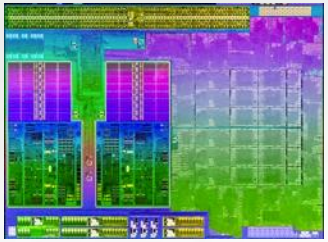


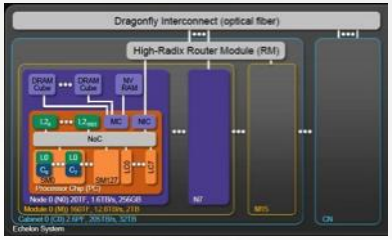

Tilera Tile-Gx

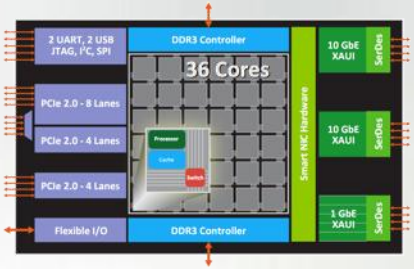
Sources: http://download.intel.com/pressroom/images/Aubrey_Isle_die.jpg, http://www.zdnet.com/amds-trinity-processors-take-on-intels-ivy-bridge-3040155225/, http://insidehpc.com/2010/11/26/nvidia-reveals-details-of-echelon-gpu-designs-for-exascale/, http://tilera.com/sites/default/files/productbriefs/Tile-

# General Characteristics of These Architectures







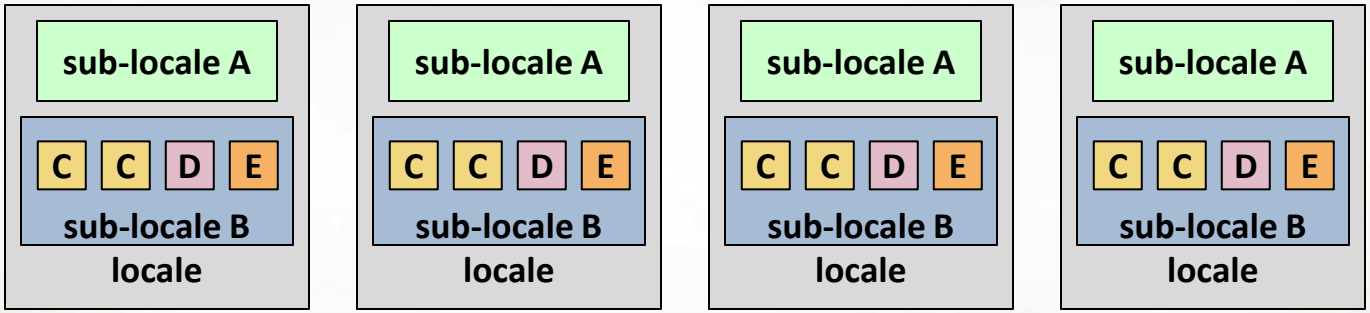

- Increased hierarchy and/or sensitivity to locality
- Potentially heterogeneous processor/memory types

**⇒ Next-gen programmers will have a lot more to think about at the node level than in the past**

# Current Work: Hierarchical Locales (our PGAS-X 2012 talk)

## Concept:

- Extend locales to support locales within locales to describe architectural sub-structures within a node




- As with traditional locales, *on-clauses* and *domain maps* should be used to map tasks and variables to a sub-locale's memory and processors
- Locale structure is defined as Chapel code
  - permits implementation policies to be specified in-language
  - introduces a new Chapel role: *architectural modeler*

# By Analogy: Let's Cross the United States!

# By Analogy: Let's Cross the United States!


…Hey, what's that sound?

# The Chapel Team (Summer 2012)

# For More Information

**Chapel project page:** http://chapel.cray.com

- overview, papers, presentations, language spec, …

**Chapel SourceForge page:** https://sourceforge.net/projects/chapel/

- release downloads, public mailing lists, code repository, …

**Blog Series:**

Myths About Scalable Programming Languages:

https://www.ieeetcsc.org/activities/blog/

**Upcoming Events:**

**SC12:** tutorial, BoFs (November 12th-16th)

**Mailing Lists:**

- chapel_info@cray.com: contact the team
- chapel-users@lists.sourceforge.net: user-oriented discussion list
- …


CRAY
THE SUPERCOMPUTER COMPANY

http://chapel.cray.com chapel_info@cray.com http://sourceforge.net/projects/chapel/