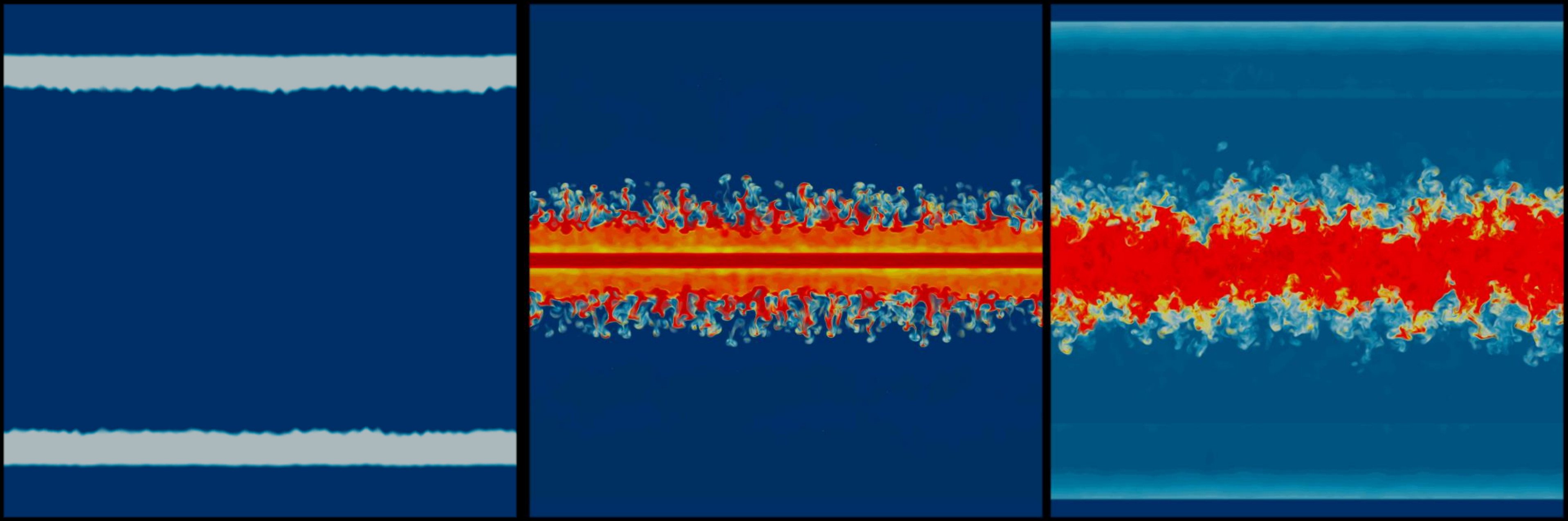
Test Problem with mPPM on 704^3 Grid
Density in X-Y Slice through Center in Z



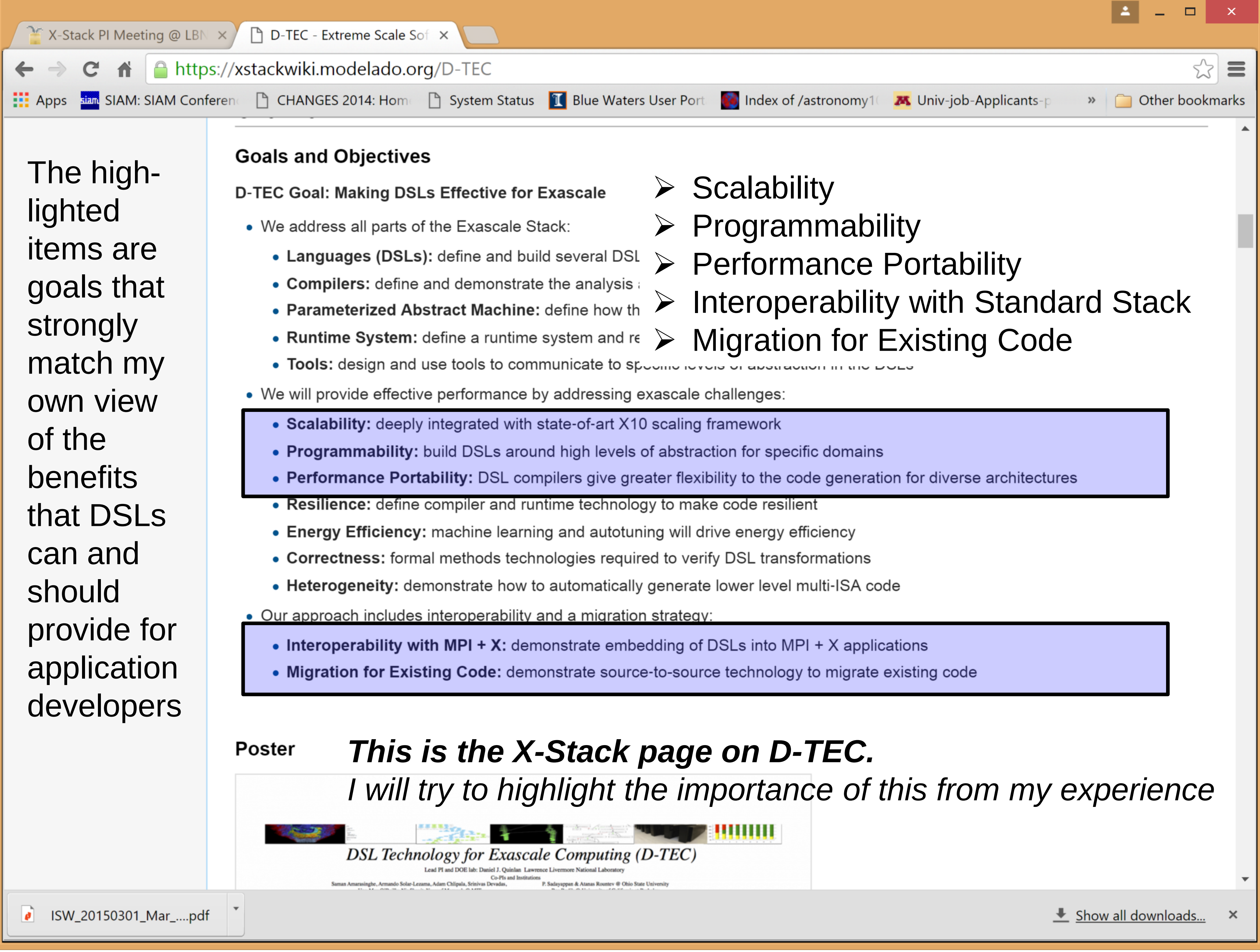
Tools for Exascale Software Development from the Application Perspective: The Importance of DSL Technology

Paul Woodward
Laboratory for Computational Science & Engineering
University of Minnesota

<u>**Whatever the Future Holds, DSLs are the Answer:**</u>

I will try to make this case by means of historical examples.

1. A DSL allows time travel, you can write code for whatever the future brings right now, and it will also run right now.
   a. The vendors tell you what they plan to do to you.
   b. You figure out the best way to exploit new advantages.
   c. You figure out the best way to avoid new pitfalls.
   d. You determine appropriate new meanings for old code expressions (example: the vectorizable loop).
   e. If this is not possible, you determine new code expressions that can easily be translated to the old ones and also to the needed new ones. (ex: Fortran-W).
   f. You:
      1) Hire 2 CS Ph.D. students from sensitive countries.
      2) Call up the ROSE team and plead for help.
      3) Write your own pre-compiler in Fortran or ______.
      4) All of the above (I am one who did all of these).
   g. You enjoy great performance on all platforms.



The high-lighted items are goals that strongly match my own view of the benefits that DSLs can and should provide for application developers

Goals and Objectives

D-TEC Goal: Making DSLs Effective for Exascale

- ➢ Scalability
- ➢ Programmability
- ➢ Performance Portability
- ➢ Interoperability with Standard Stack
- ➢ Migration for Existing Code

- We address all parts of the Exascale Stack:
  - **Languages (DSLs):** define and build several DSL
  - **Compilers:** define and demonstrate the analysis
  - **Parameterized Abstract Machine:** define how th
  - **Runtime System:** define a runtime system and re
  - **Tools:** design and use tools to communicate to sp
- We will provide effective performance by addressing exascale challenges:
  - **Scalability:** deeply integrated with state-of-art X10 scaling framework
  - **Programmability:** build DSLs around high levels of abstraction for specific domains
  - **Performance Portability:** DSL compilers give greater flexibility to the code generation for diverse architectures
  - **Resilience:** define compiler and runtime technology to make code resilient
  - **Energy Efficiency:** machine learning and autotuning will drive energy efficiency
  - **Correctness:** formal methods technologies required to verify DSL transformations
  - **Heterogeneity:** demonstrate how to automatically generate lower level multi-ISA code
- Our approach includes interoperability and a migration strategy:
  - **Interoperability with MPI + X:** demonstrate embedding of DSLs into MPI + X applications
  - **Migration for Existing Code:** demonstrate source-to-source technology to migrate existing code

Poster

This is the X-Stack page on D-TEC.

I will try to highlight the importance of this from my experience

DSL Technology for Exascale Computing (D-TEC)

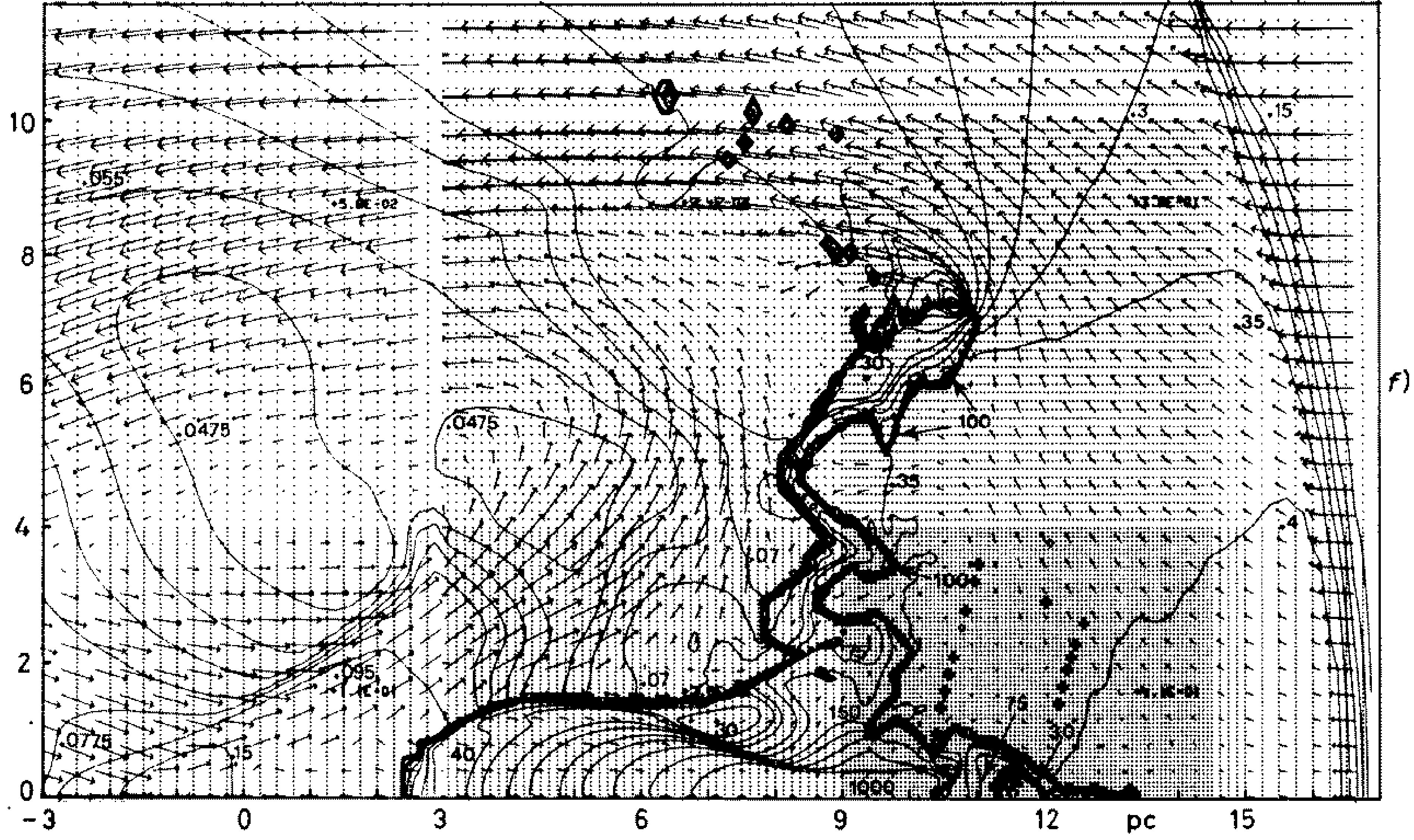
Lead PI and DOE lab: Daniel J. Quinlan Lawrence Livermore National Laboratory

Co-PIs and Institutions

<u>Why DSLs are the Answer: Some History</u>

Just after receiving my Ph.D., in 1973, I worked on the world's first vectorized hydrocode, BBC, at Livermore.

1. A DSL was implemented in elaborate macros.
    a. Performance Portability on CDC 7600 & CDC Star 100.
    b. Stacklib runtime turned 7600 into vector machine (4x)
    c. Memory management to keep 36000-long vectors under control: could have only ws1,ws2,ws3,ws4,ws5.
    d. <u>Fastest machine on earth at the time</u>.
    e. Civic compiler for Startran discarded in 1978, when Cray compiler emerged and offered to port legacy code with its own embedded DSL (example: cDIR$ IVDEP)
    f. Legacy code conversion took 10 years, and some codes simply never converted. <u>You have to want to</u>.
    g. "cost is no object" approach, with whole compiler team of about 3 people, as well as complete control of the OS
    h. Today, the vendor would provide a huge list of intrinsic functions, and a compiler could be source-to-source.



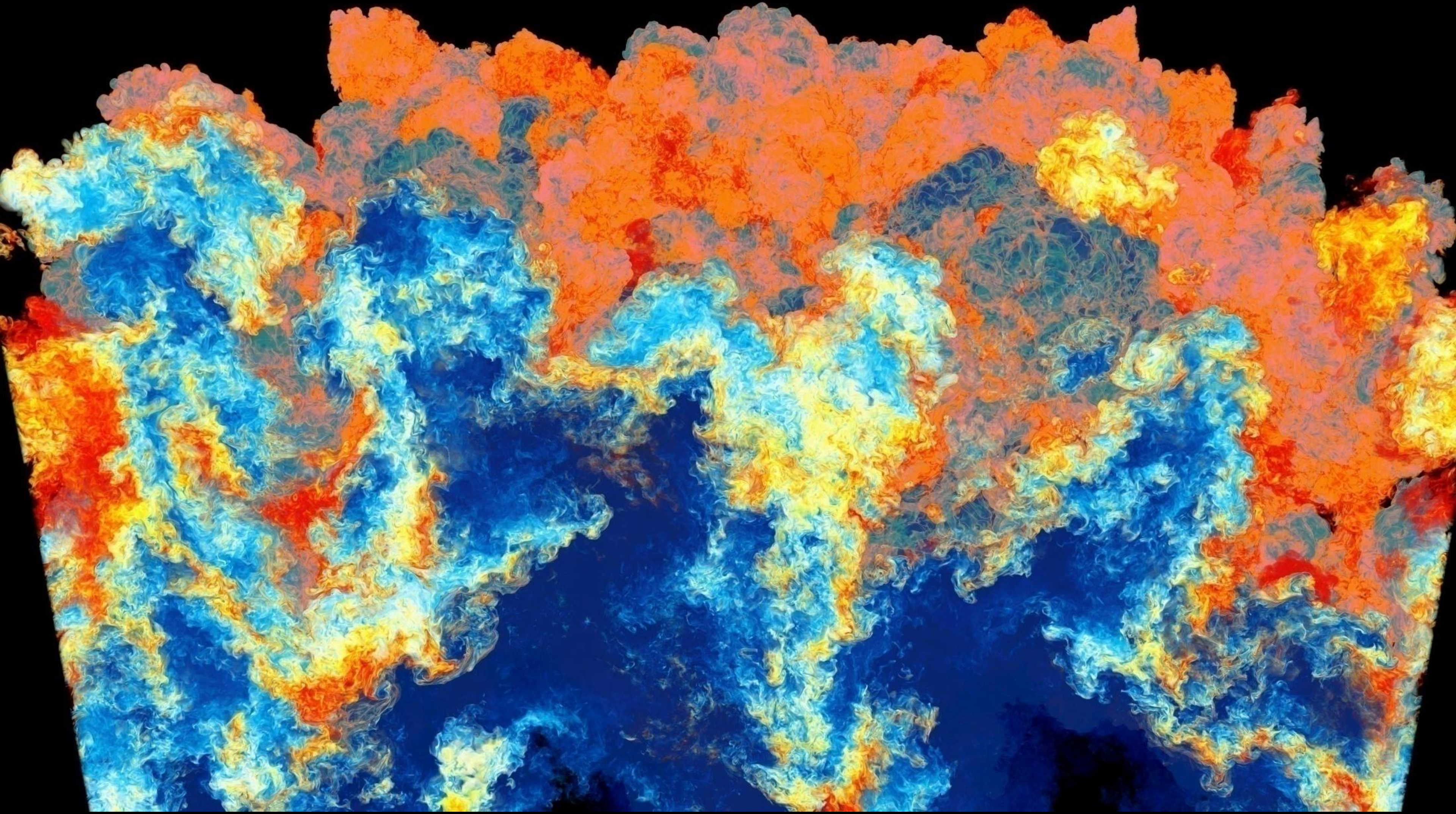
An early computation of an interstellar gas cloud being imploded by a shock wave, compressing it so that its self gravity can take over and cause it to collapse and form a star. (Woodward 1976)

Why DSLs are the Answer: Some More History

Skipping over the CM5 Connection Machine, we come to the Los Alamos Roadrunner machine, in 2008:

1. Jagan Jayaraj & Pei-Hung Lin implemented a DSL.
   a. Performance Portability on everything & Roadrunner.
   b. Converted Fortran-77 into C with calls to vendor provided intrinsic functions for the SPU SIMD engine.
   c. PowerPC intrinsics had only different prefix.
   d. Intel intrinsics nearly the same, with other names.
   e. Fastest machine on earth at the time.
   f. Backend to CELL intrinsics discarded with Roadrunner.
   g. Legacy code conversion NOT SIMPLE.
   h. Need to rewrite performance critical code modules, but DSL (CFDbuilder) developed to make that easy.
   i. Need to restructure data in main memory. Damn.
   j. Also, still need to want to. (This is even harder!)
   k. We still have 2 more years before we reach the 10-year time scale historically required for legacy conversion.

PPM simulation of Rayleigh-Taylor mixing on grid of $4096^2 \times 2048$ cells. We view the entire mixing layer at dump 140.



Began on a grid of $8192^2 \times 1024$ cells, now using $4096^2 \times 2048$ cells.

Works Quite Generally: Gas dynamics with PPM at extreme scale on Blue Waters (not with mPPM mini-app, but close)

Focusing on mixing of fluids at multifluid interfaces, both “stable” & unstable, and its consequences in various contexts.

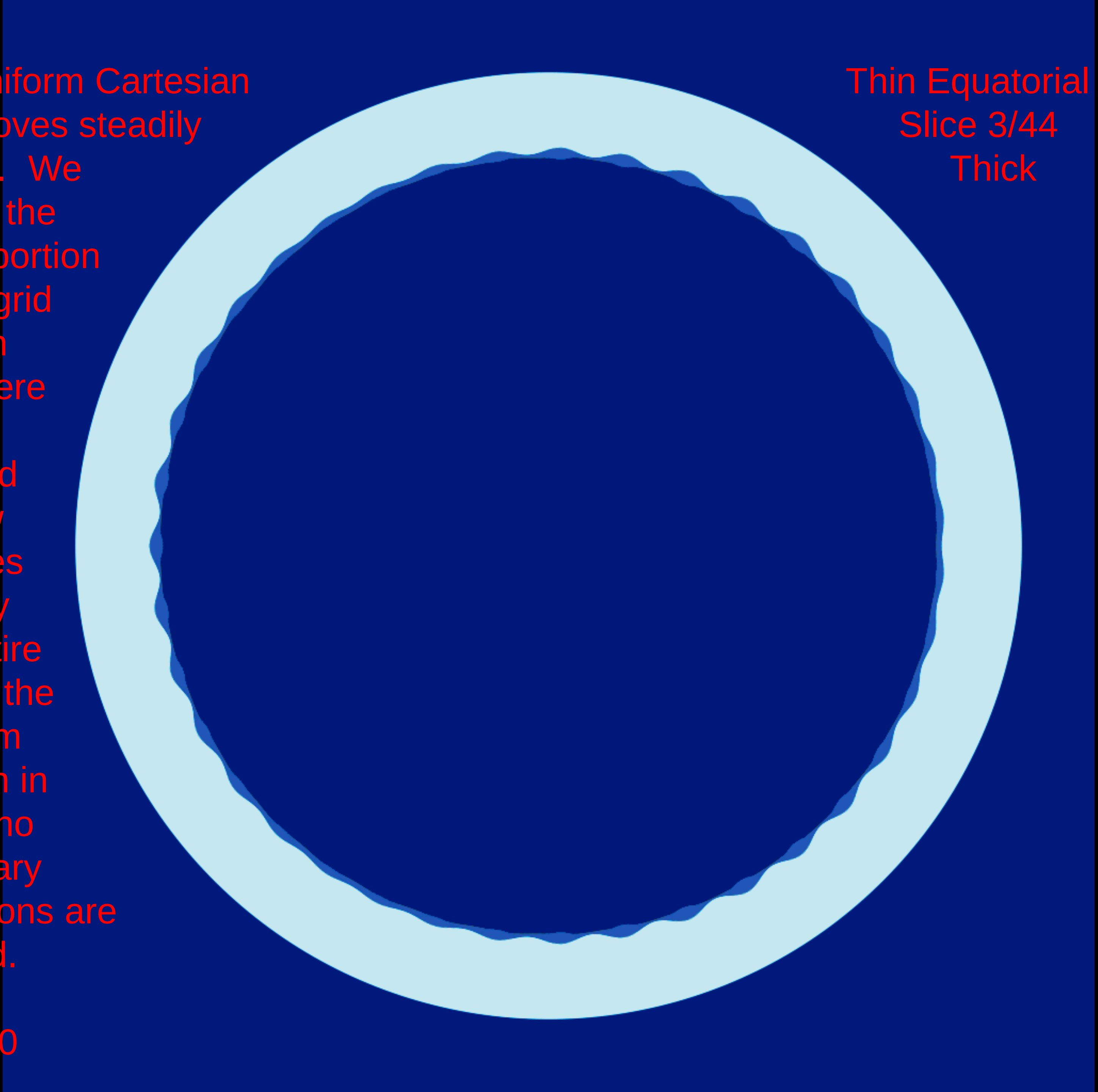
1. Simulated the hydrodynamic compression of an initially spherical, but very simplified, inertial confinement fusion (ICF) capsule (during friendly user access period).
   a. Perturbed with 2 spherical harmonic modes that have a mode 3 beat frequency, plus 1 extremely high mode.
   b. Demonstrated ability to preserve those symmetries that must be preserved, in a statistical sense.
   c. 1.18 trillion cells, about 2 days on 702,000 cores.
   d. 1.5 Pflop/s (32-bit) sustained.
2. Mixing of hydrogen into convection zone above helium shell flash in an evolved giant star (AGB star).
   a. Very long approach to violent, unstable global mode of ingested hydrogen burning (GOSH).
   b. Implications for nucleosynthesis of heavy elements.

t = 0

The uniform Cartesian grid moves steadily inward. We render the same portion of the grid in each view here so that the field of view includes roughly the entire part of the problem domain in which no boundary conditions are applied.

Dump 0

Thin Equatorial Slice 3/44 Thick



ρ

10560^3 grid

22nd 44th in Y

rholut 7

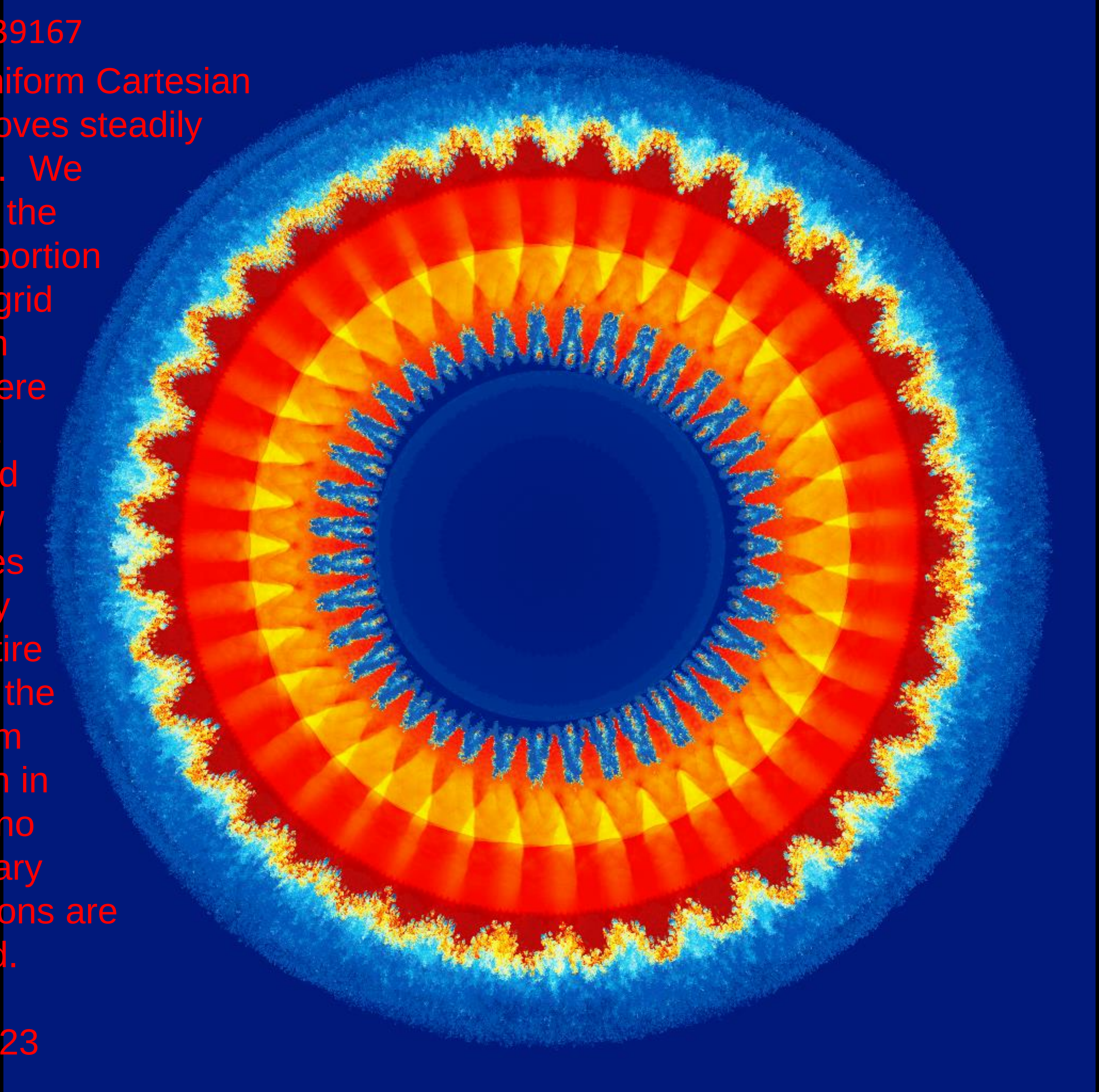
opacity 48

distance from midplane 1.45

t = 0.039167

The uniform Cartesian grid moves steadily inward. We render the same portion of the grid in each view here so that the field of view includes roughly the entire part of the problem domain in which no boundary conditions are applied.

Dump 23



ρ

10560^3 grid

22nd 44th in Y

rholut 7

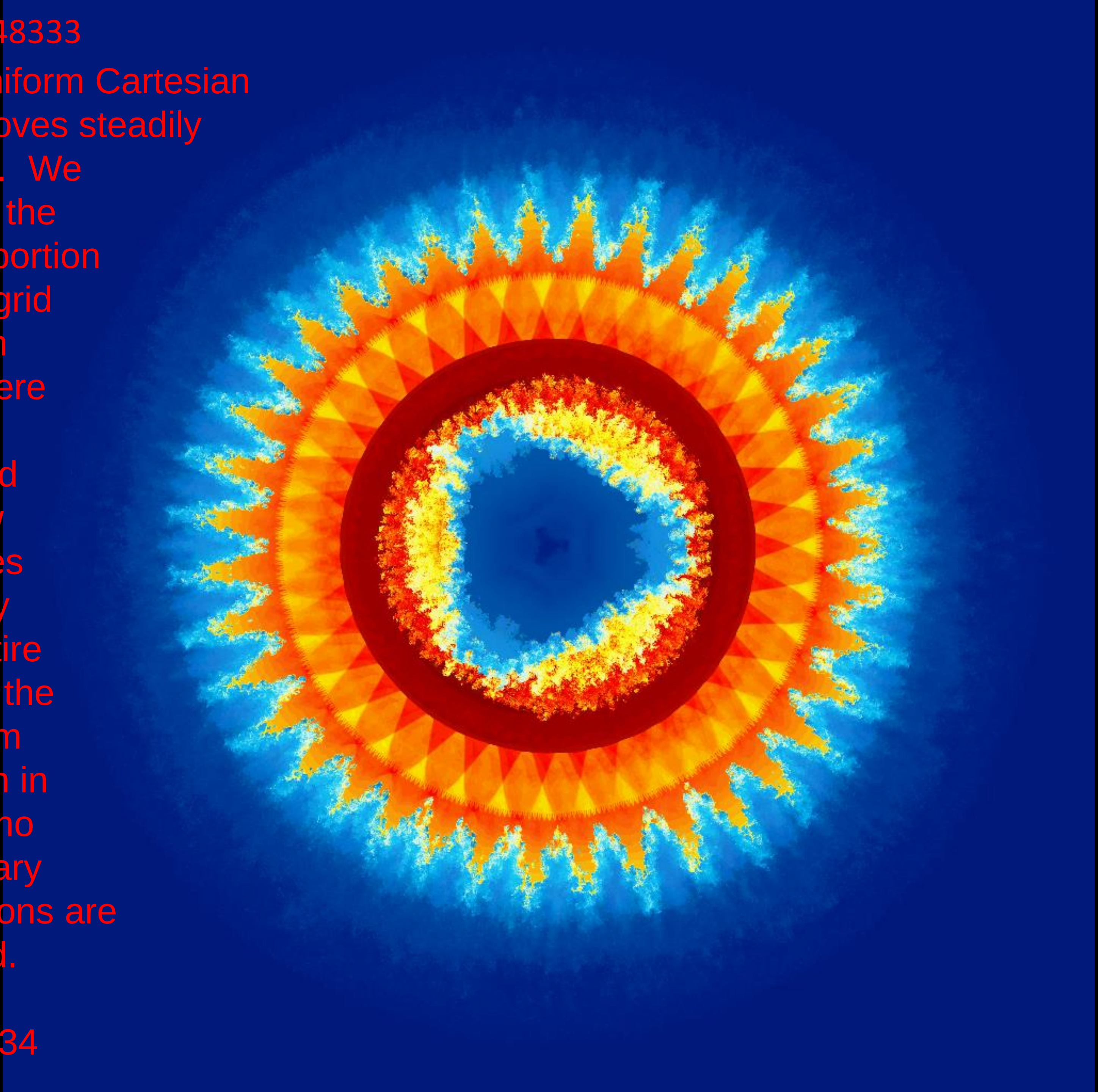
opacity 48

distance from midplane 1.45

t = 0.048333

The uniform Cartesian grid moves steadily inward. We render the same portion of the grid in each view here so that the field of view includes roughly the entire part of the problem domain in which no boundary conditions are applied.

Dump 34



ρ

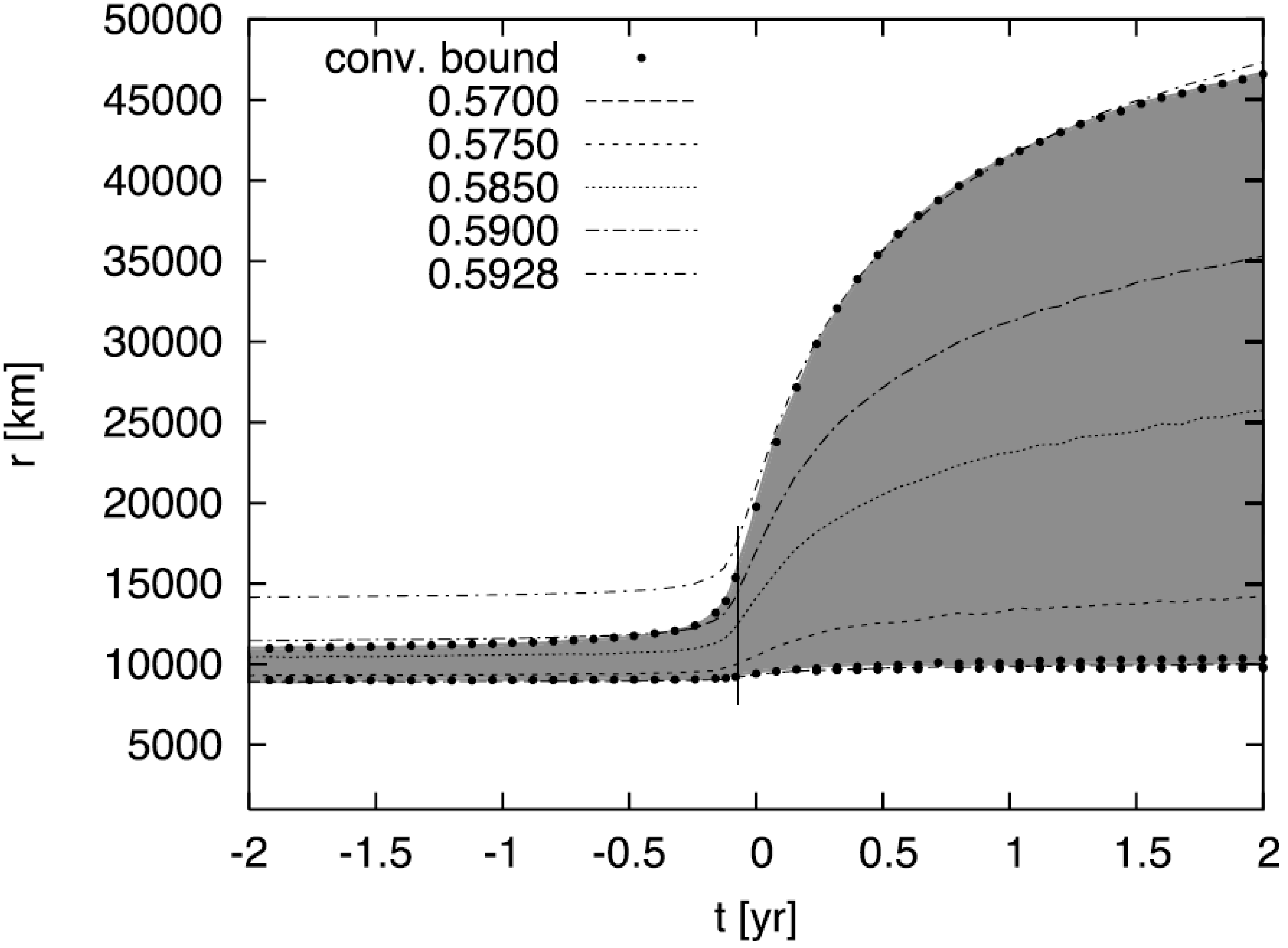
10560^3 grid

22nd 44th in Y

rholut 12

opacity 48

distance from midplane 1.45



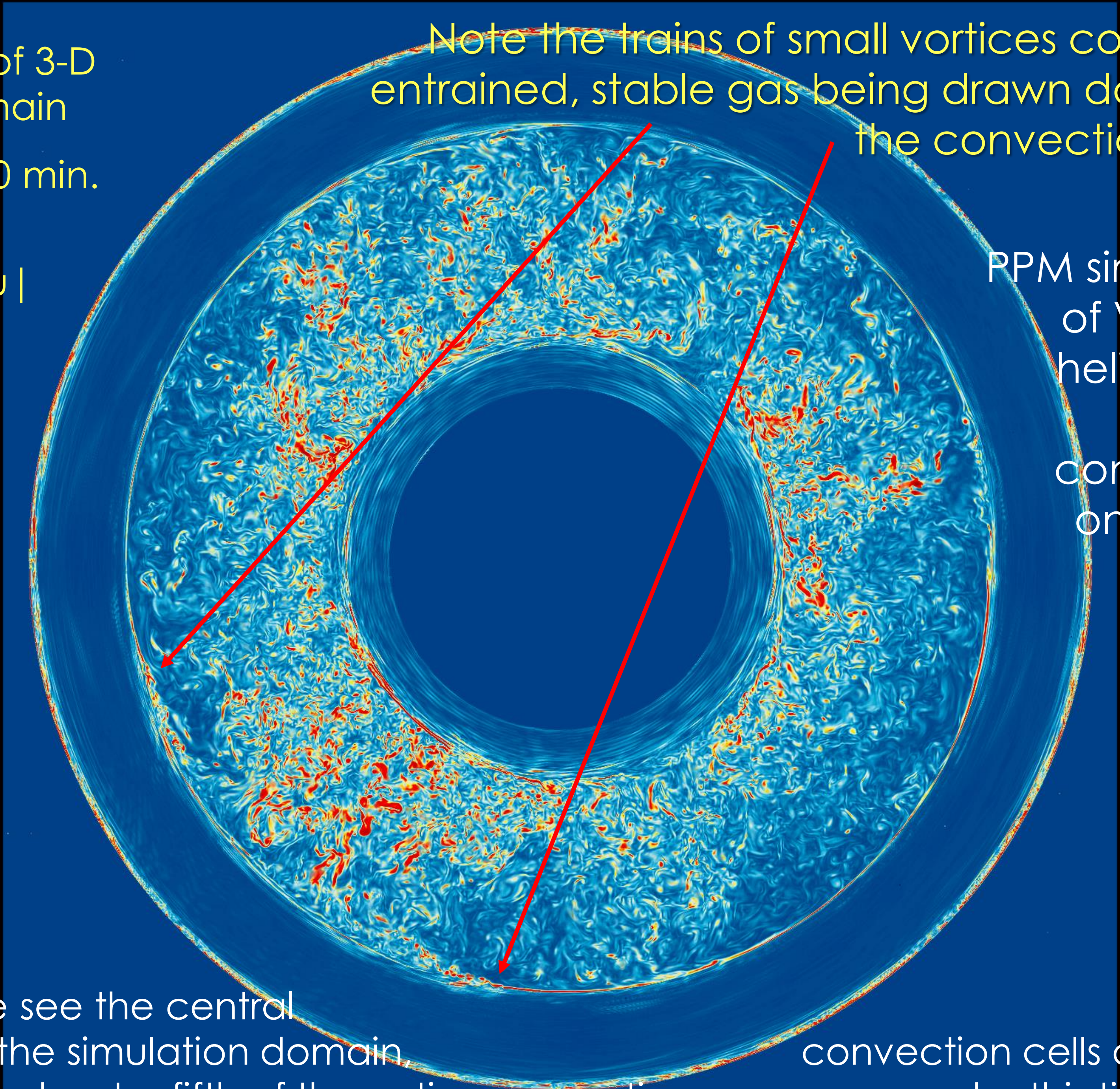
Time evolution of the radial location of the He-shell flash convection zone based on the 1-D stellar evolution model of Herwig. Time is set to 0 at the peak of the He-burning luminosity. Dots represent individual time steps. Lagrangian lines at different mass fractions are shown. The convection zone grows both in radius and in mass fraction over the 2-year interval shown. Our simulation is performed at about time 0.2 yr on this slide.

Slice of 3-D Domain

t = 400 min.

$|\nabla \times u|$

Note the trains of small vortices containing entrained, stable gas being drawn down into the convection zone.



PPM simulation of VLTP star helium shell flash convection on a 1536^3 grid.

Here we see the central 0.2% of the simulation domain, convection cells as large as about a fifth of the entire convection zone are seen by this time.

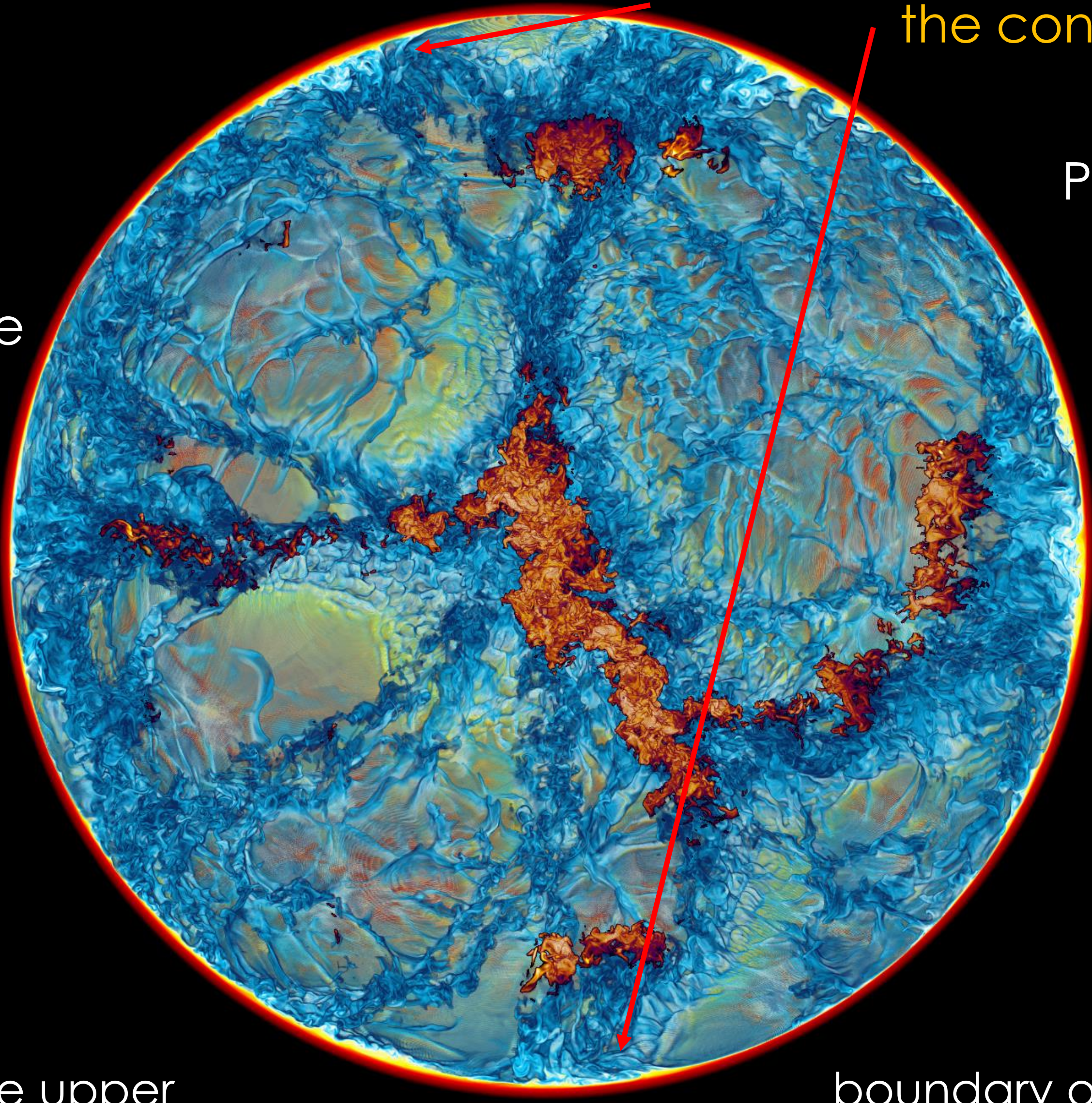
Half of 3-D Domain

t = 400 min.

FV_{H+He}

Energy release from burning ingested hydrogen is shown as the dark purple and yellow/red flame.

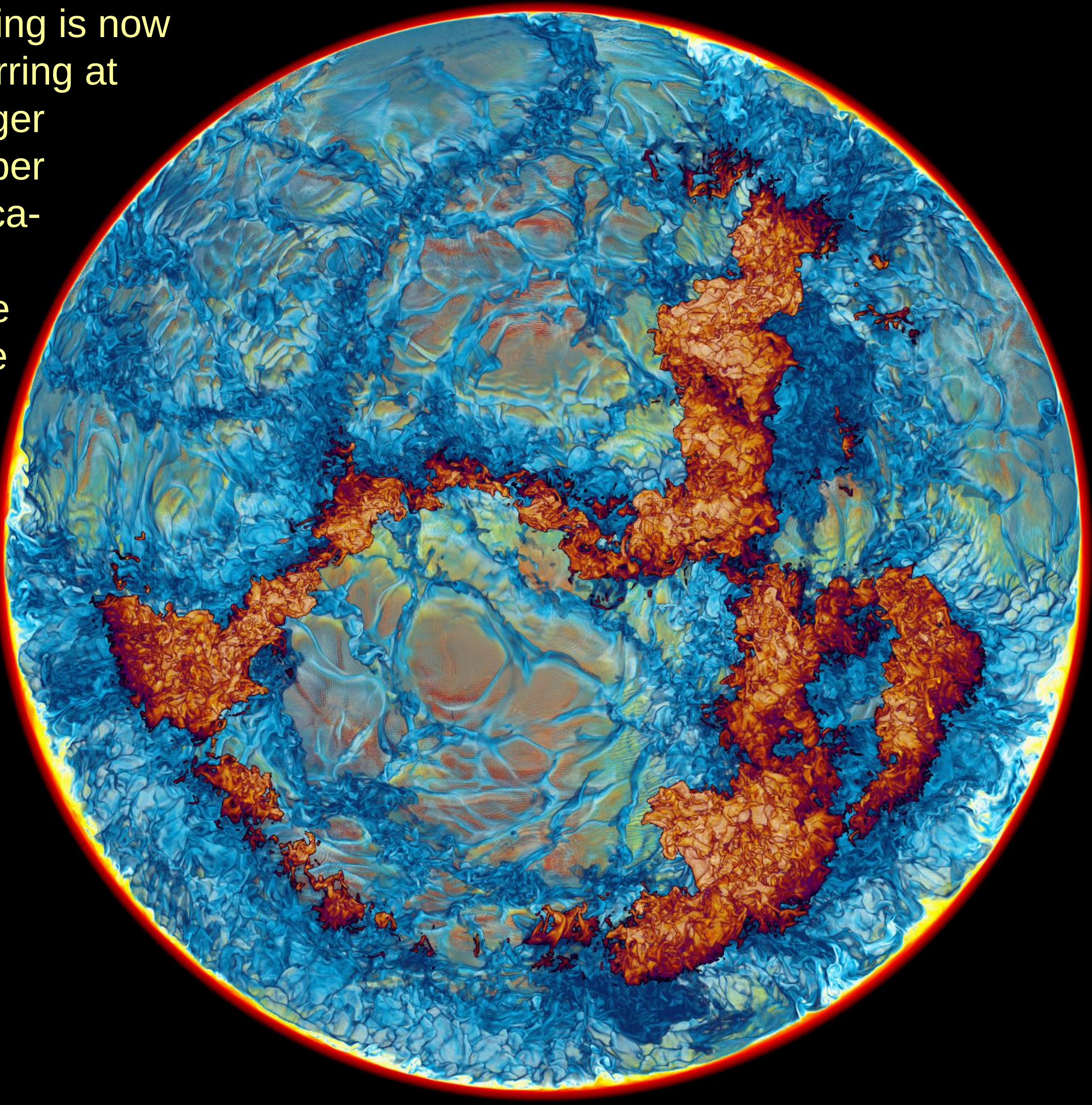
Note the trains of small vortices containing entrained, stable gas being drawn down into the convection zone.



PPM simulation of VLTP star helium shell flash convection on a 1536^3 grid.

Here we see the upper boundary of the convection zone above the helium burning shell, looking from the center of the star outward. The blue descending plumes trace out the convection cells

Burning is now occurring at a larger number of loca-tions at the same time.



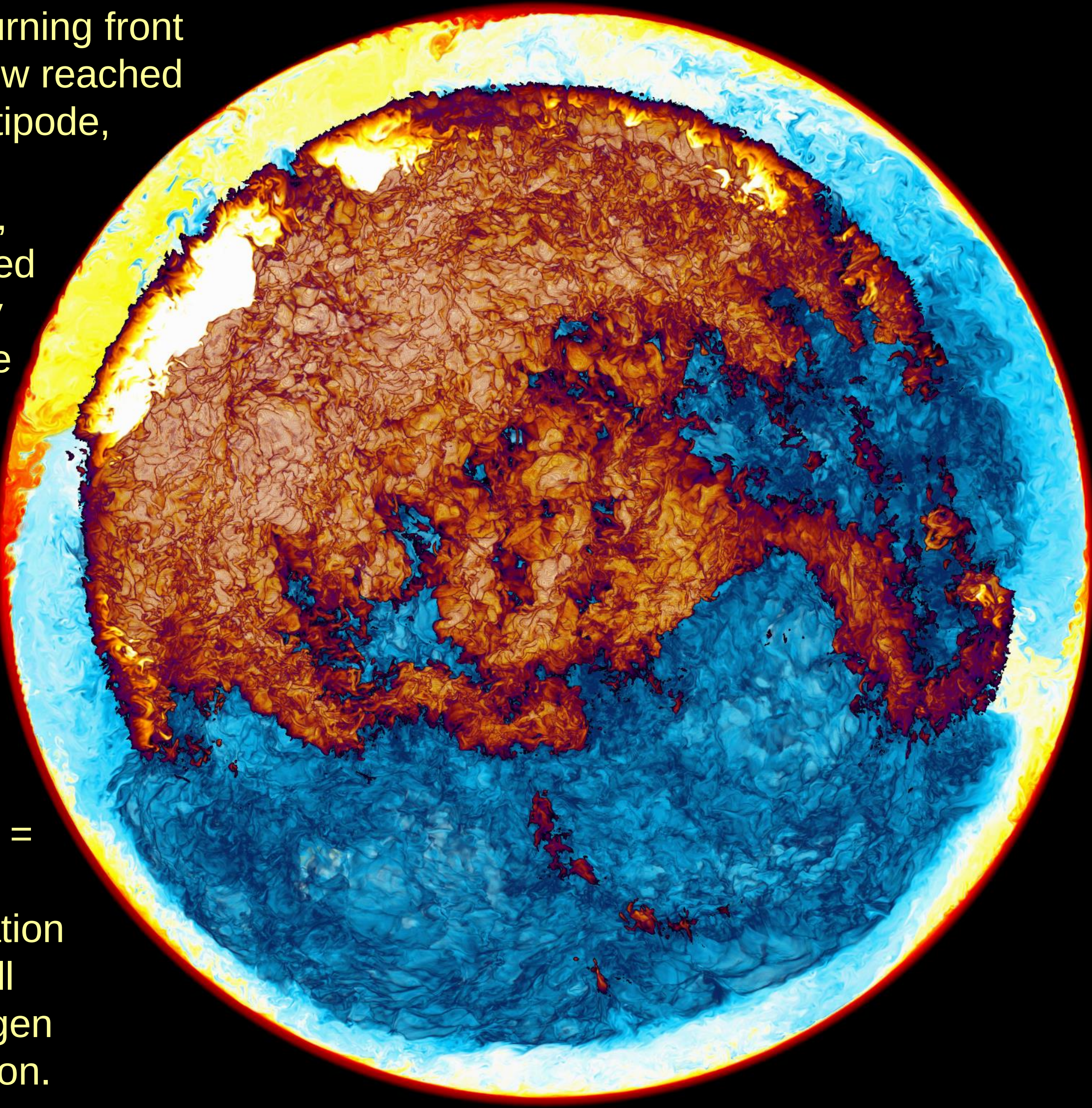
Sakurai's Object H-ingestion simulation on Blue Waters machine in Jan., 2014, on a grid of 1536^3 cells.

We see a hemisphere and make only mixtures of entrained hydrogen-rich gas with gas of the helium shell flash convection zone visible. The energy release rate from burning ingested H is shown in very dark blue, yellow, and white.

t = 650 min.

The burning front has now reached the antipode, where violent, localized energy release drives the oscill-ation back to its origin-al site.

GOSH = Global Oscillation of Shell Hydrogen ingestion.

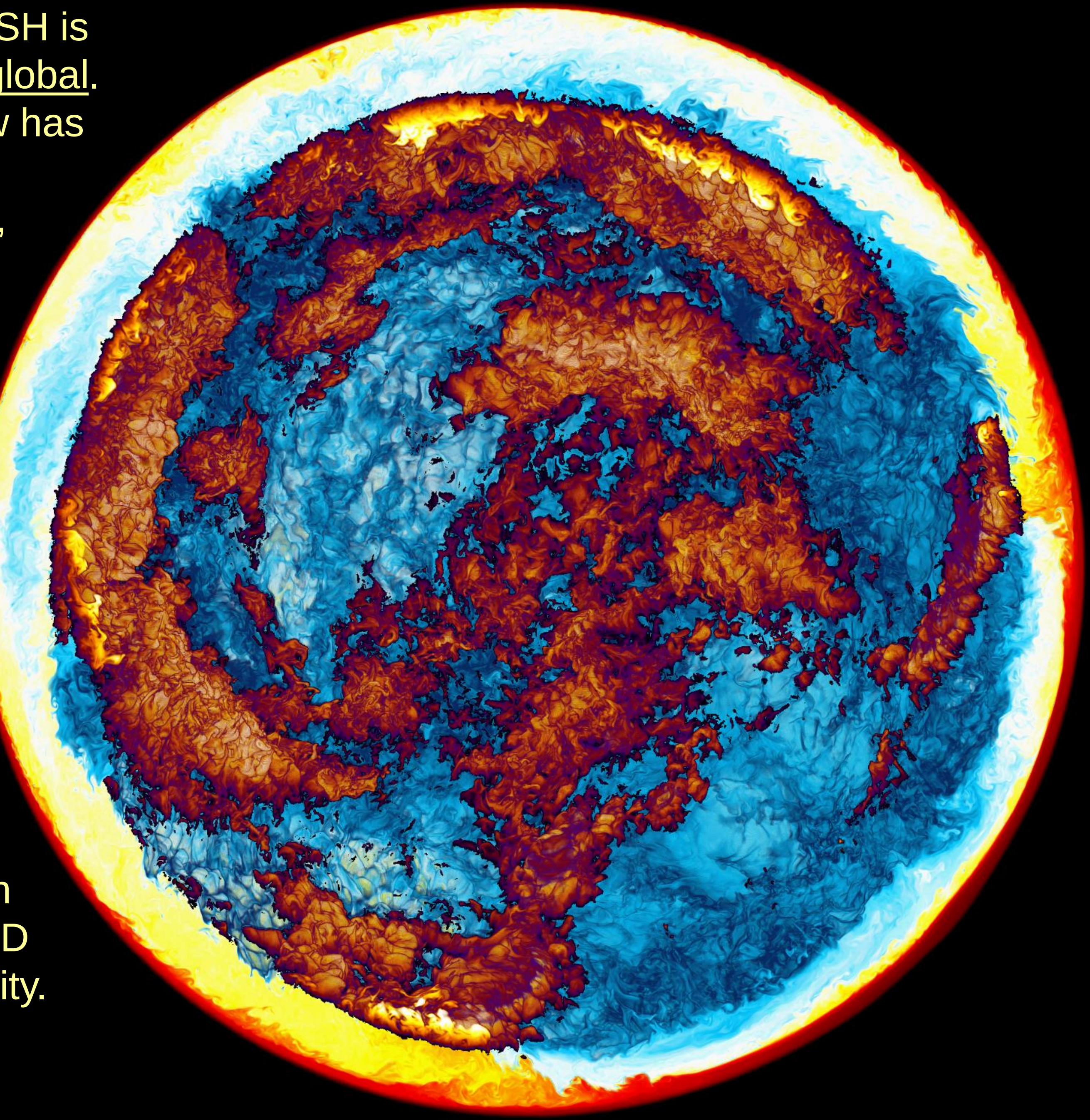


Sakurai's Object H-ingestion simulation on Blue Waters machine in Jan., 2014, on a grid of 1536^3 cells.

We see a hemisphere and make only mixtures of entrained hydrogen-rich gas with gas of the helium shell flash convection zone visible. The energy release rate from burning ingested H is shown in very dark blue, yellow, and white.

t = 1188 min.

The GOSH is indeed global. This flow has a 1-D average, but it is by no means a 1-D phen-omen-on. Blue Waters makes it possi-ble to see the GOSH in its full 3-D complexity.

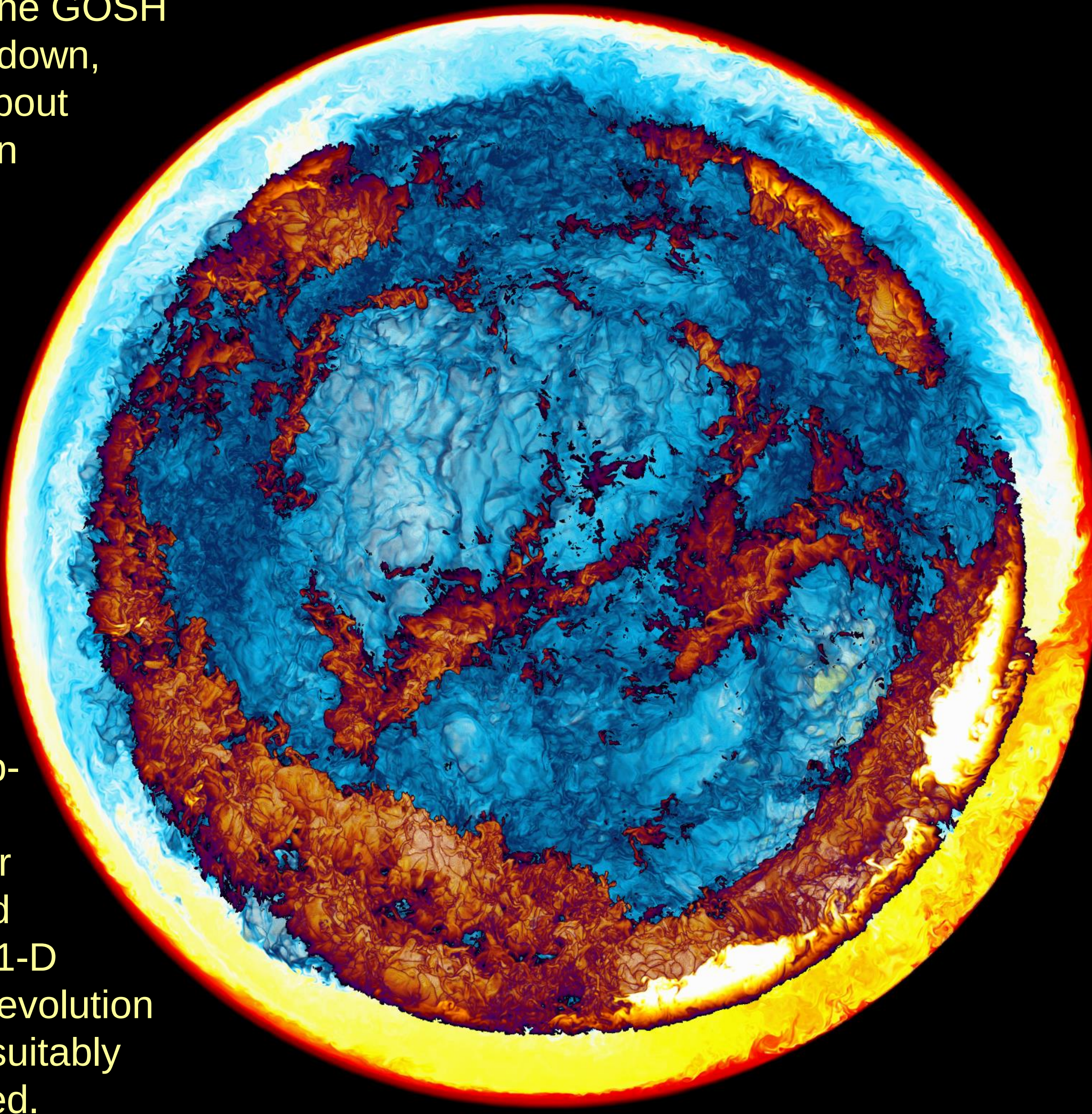


Sakurai's Object H-ingestion simulation on Blue Waters machine in Jan., 2014, on a grid of 1536^3 cells.

We see a hemisphere and make only mixtures of entrained hydrogen-rich gas with gas of the helium shell flash convection zone visible. The energy release rate from burning ingested H is shown in very dark blue, yellow, and white.

t = 1200 min.

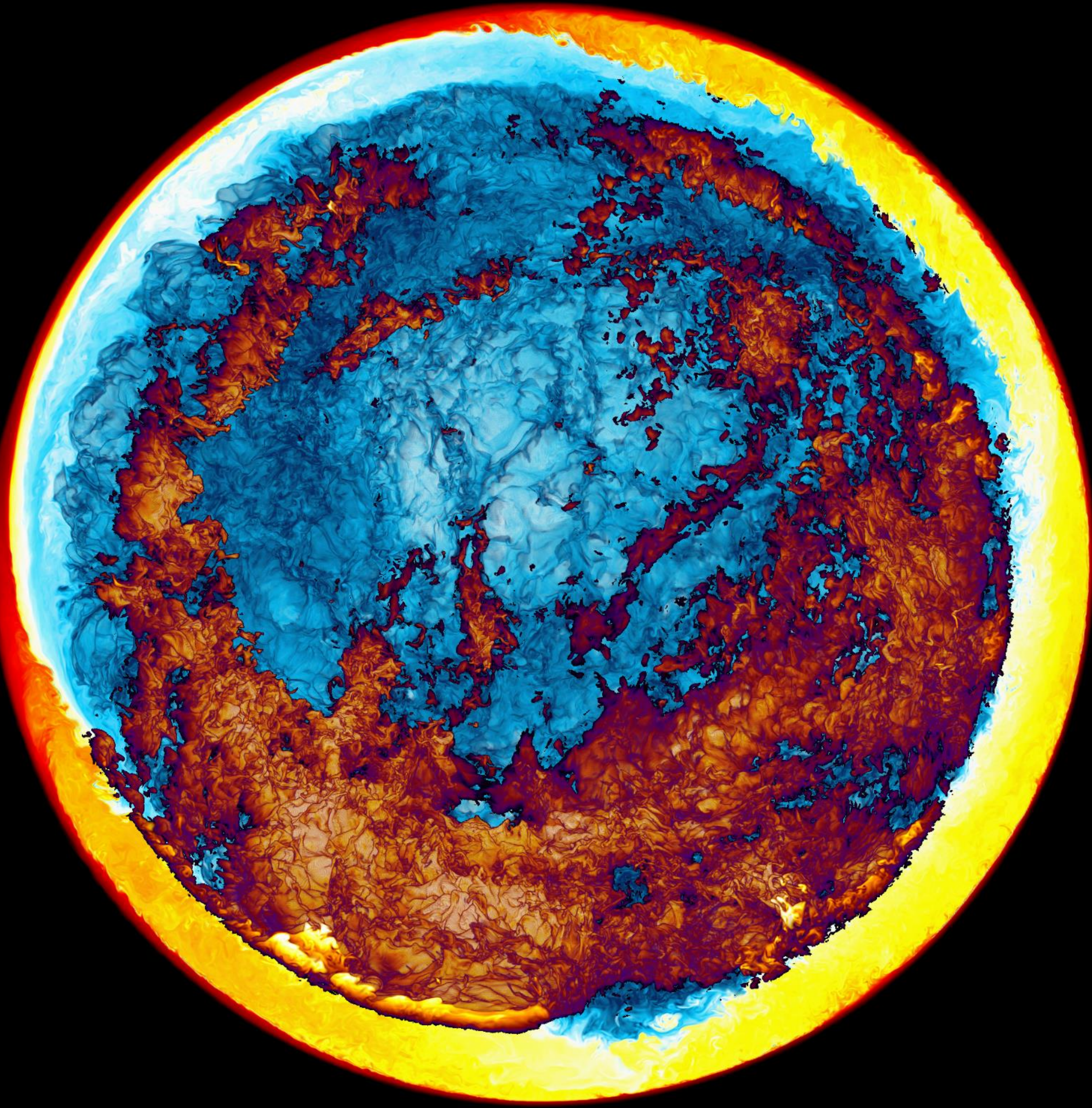
Once the GOSH quiets down, after about a day in the life of this star, we can be well justi-fied in carry-ing our descrip-tion of the star forward with a 1-D stellar evolution code, suitably modified.



Sakurai's Object H-ingestion simulation on Blue Waters machine in Jan., 2014, on a grid of 1536^3 cells.

We see a hemisphere and make only mixtures of entrained hydrogen-rich gas with gas of the helium shell flash convection zone visible. The energy release rate from burning ingested H is shown in very dark blue, yellow, and white.

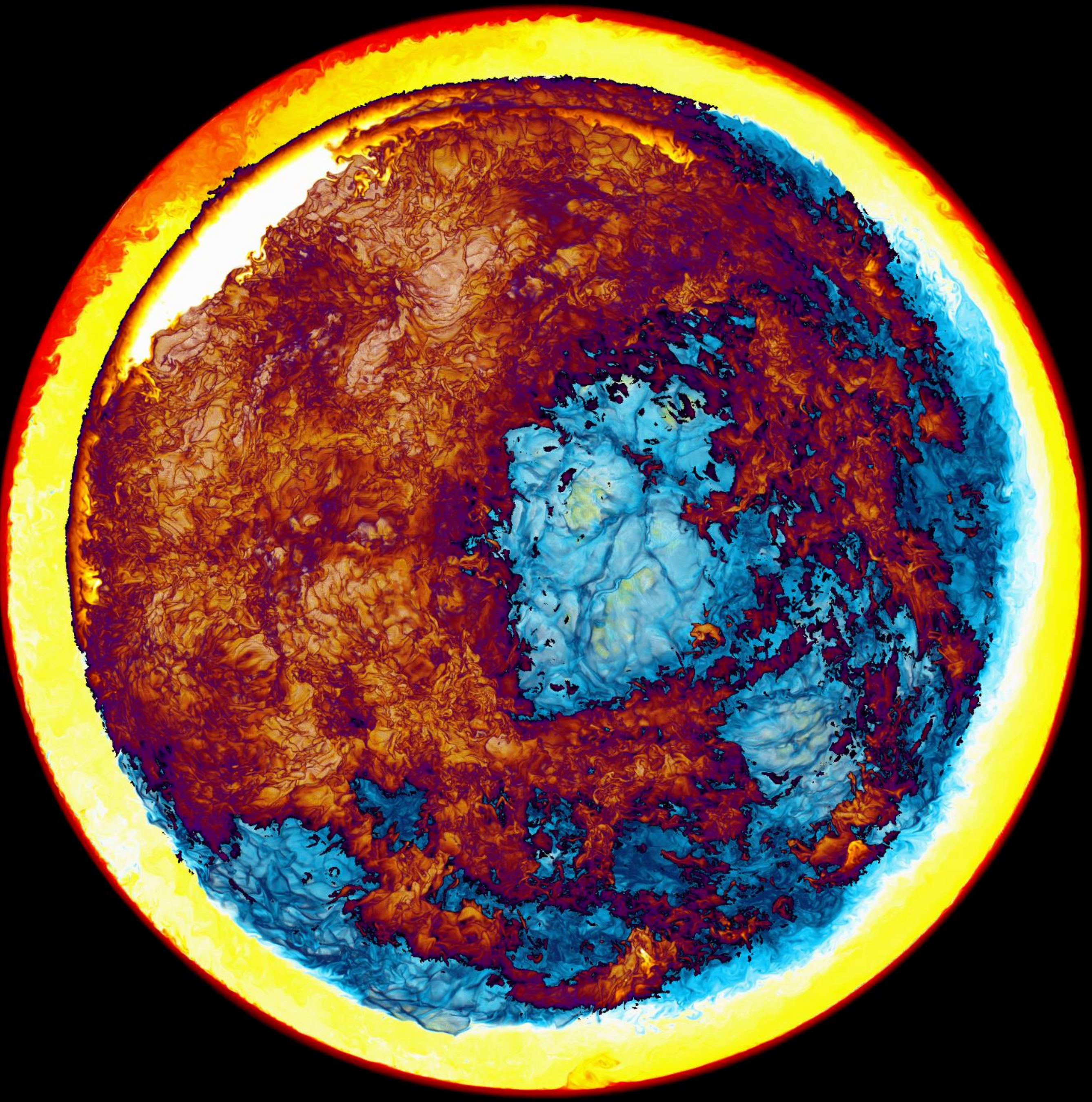
t = 1212 min.



Sakurai's Object H-ingestion simulation on Blue Waters machine in Jan., 2014, on a grid of 1536^3 cells.

We see a hemisphere and make only mixtures of entrained hydrogen-rich gas with gas of the helium shell flash convection zone visible. The energy release rate from burning ingested H is shown in very dark blue, yellow, and white.

t = 1225 min.



Sakurai's Object H-ingestion simulation on Blue Waters machine in Jan., 2014, on a grid of 1536^3 cells.

We see a hemisphere and make only mixtures of entrained hydrogen-rich gas with gas of the helium shell flash convection zone visible. The energy release rate from burning ingested H is shown in very dark blue, yellow, and white.

t = 1238 min.

Why we need a DSL and not just subroutines or macros:

1. Briquette data structure.
   a. D(4,4,4,16,nbqs), ***not*** D(4*nbqx,4*nbqy,4*nbqz,16)
   b. Indxbq(4,0:nbqx+1,0:nbqy+1,0:nbqz+1,8)
   c. AMR version makes everything much harder.
   d. D is bunch of briquette records, 4^3cells, 16 variables.
   e. Indxbq is a look-up table – indirect addressing of bqs.
2. Annotated Fortran-W code expression – ***easy to write***.
   a. Simple program for a uniform grid for a single briquette
3. CFDbuilder automatic code translator (moving to ROSE).
   a. Takes code for sequence of single briquette updates.
   b. In-lines everything, fuses ALL loops, compresses memory footprint to fit into cache. ***Doing this manually is insane***.
4. Benefits in (adjustably) adapting code to future hardware:
   a. Increases flops/word, adjustably, to extreme levels.
   b. Decreases relevance, adjustably, of off-chip data bandwidth.
   c. Produces massive sequence of short vector (SIMD) ops that ALL have perfectly aligned 16-word operands.

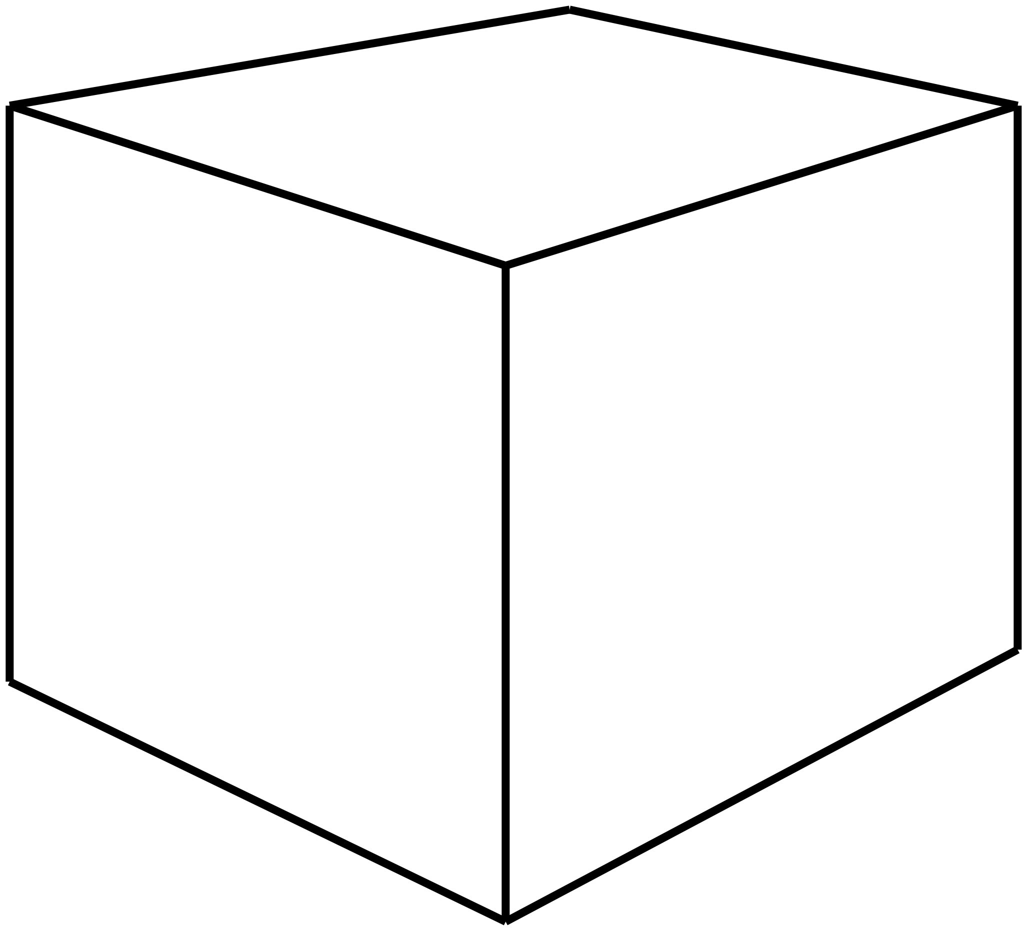
<u>**Why we need a DSL and not just subroutines or macros**</u>**:**

1. Further benefits of the DSL approach
   a. The compiler does not have to take responsibility for the code transformations.
   b. It does not have to prove that they are "safe."
   c. It does not need to become convinced that the transformed code is "correct."
   d. By using the DSL, you assert that the transformations are safe and that the transformed code is correct.
   e. The use of a DSL forces the programmer to take responsibility for his or her code.
   f. This is no different than the default situation, the bugs are always your responsibility.
   g. These facts make the DSL translator "easy" to write.
   h. Similar example from the deep past: cDIR$ IVDEP

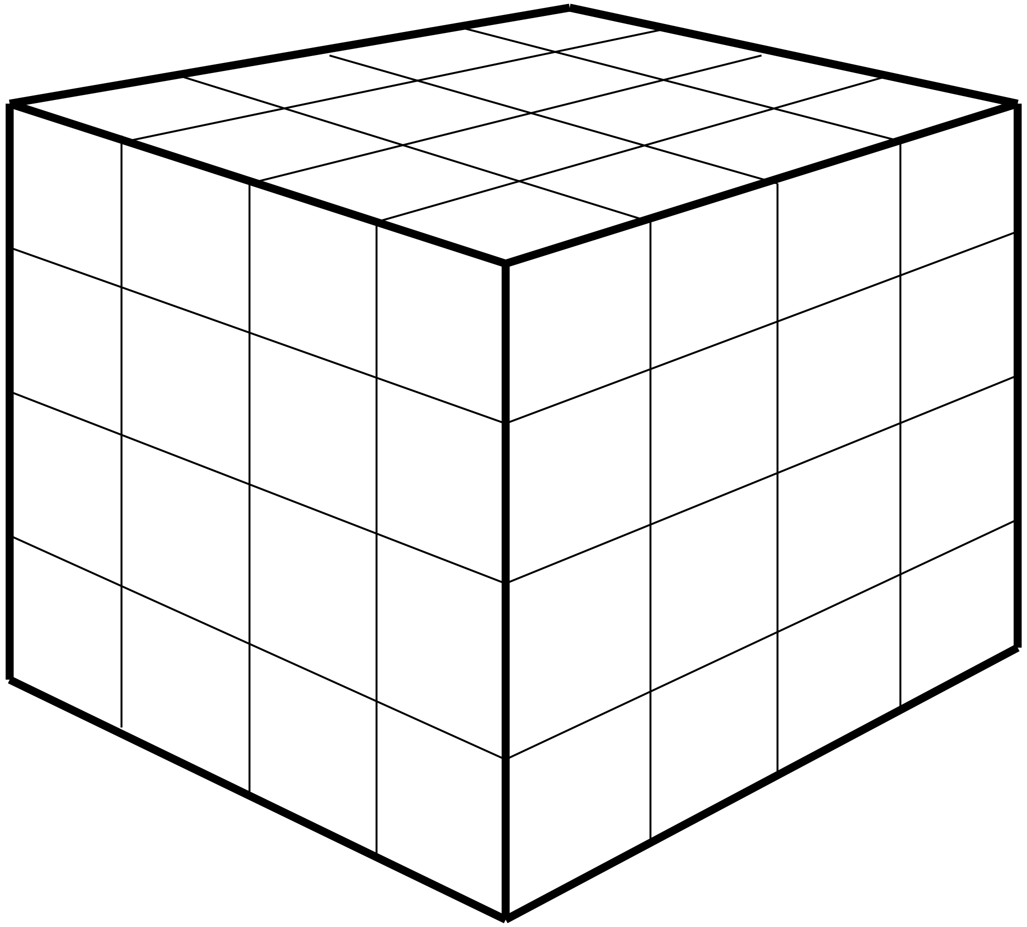
<u>A Strategy for a Legacy Code, as an Example of Potential of DSL</u>:

1. Briquette data structure.
   a. DSL can understand new intrinsic data type, a bq-array.
   b. Ease programming in computationally intensive routines.
   c. Automatically apply new rule for computing memory location from array indices in other parts of code, I/O etc.
   d. Could dramatically reduce barriers to adoption.
2. Assume a code that uses as fundamental data structures lists of grid cells, lists of their neighbors, and lists of their grid cell faces, with lists of cells on either side of the faces.
   a. Such codes do exist. You might know one.
3. Imagine that you could wave a wand and magically turn each grid cell into a grid briquette of 4^3 cells.
   a. Stalls on indirect addressing would simply disappear.
   b. Essentially every line of code would now vectorize.
4. <u>Why would you not do that</u>?
   a. You might justifiably be afraid.
   b. <u>An automatic DSL tool that is flawless is the answer to fear</u>.

Let’s start small and work outward. This is a grid cell.



We will subdivide it evenly into a grid briquette.

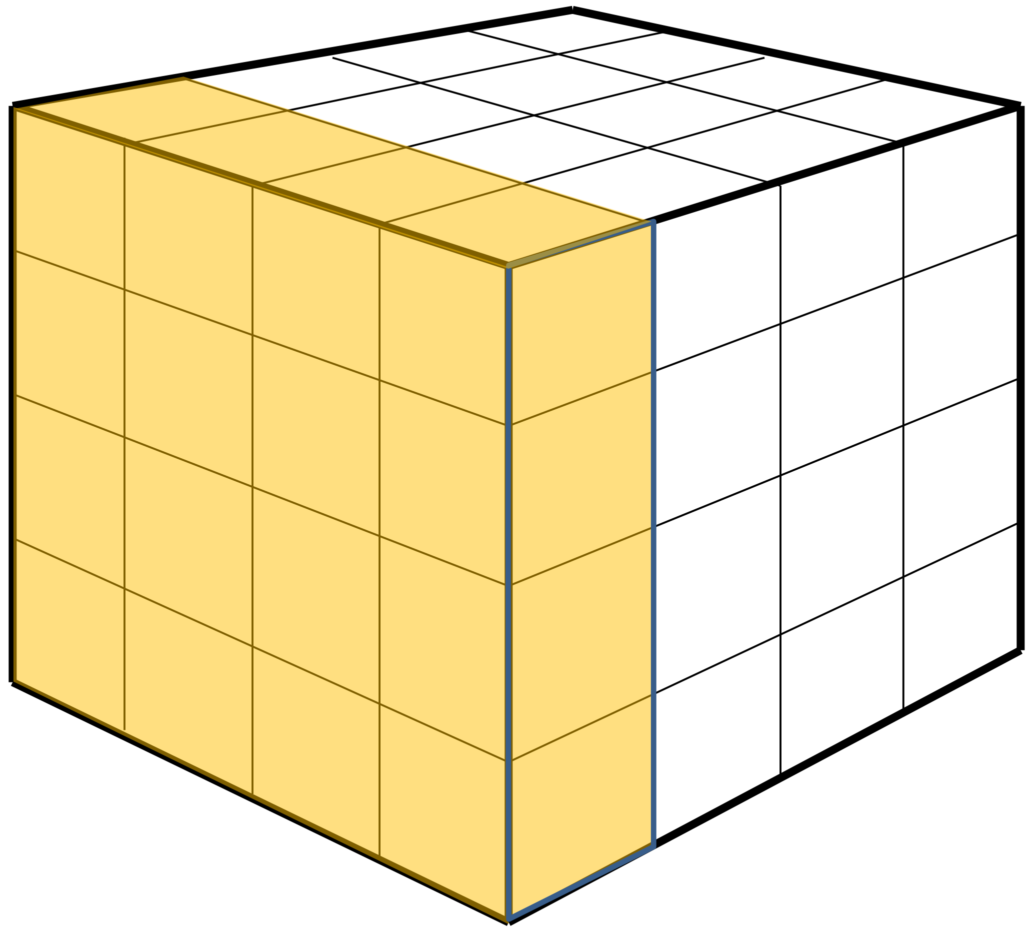


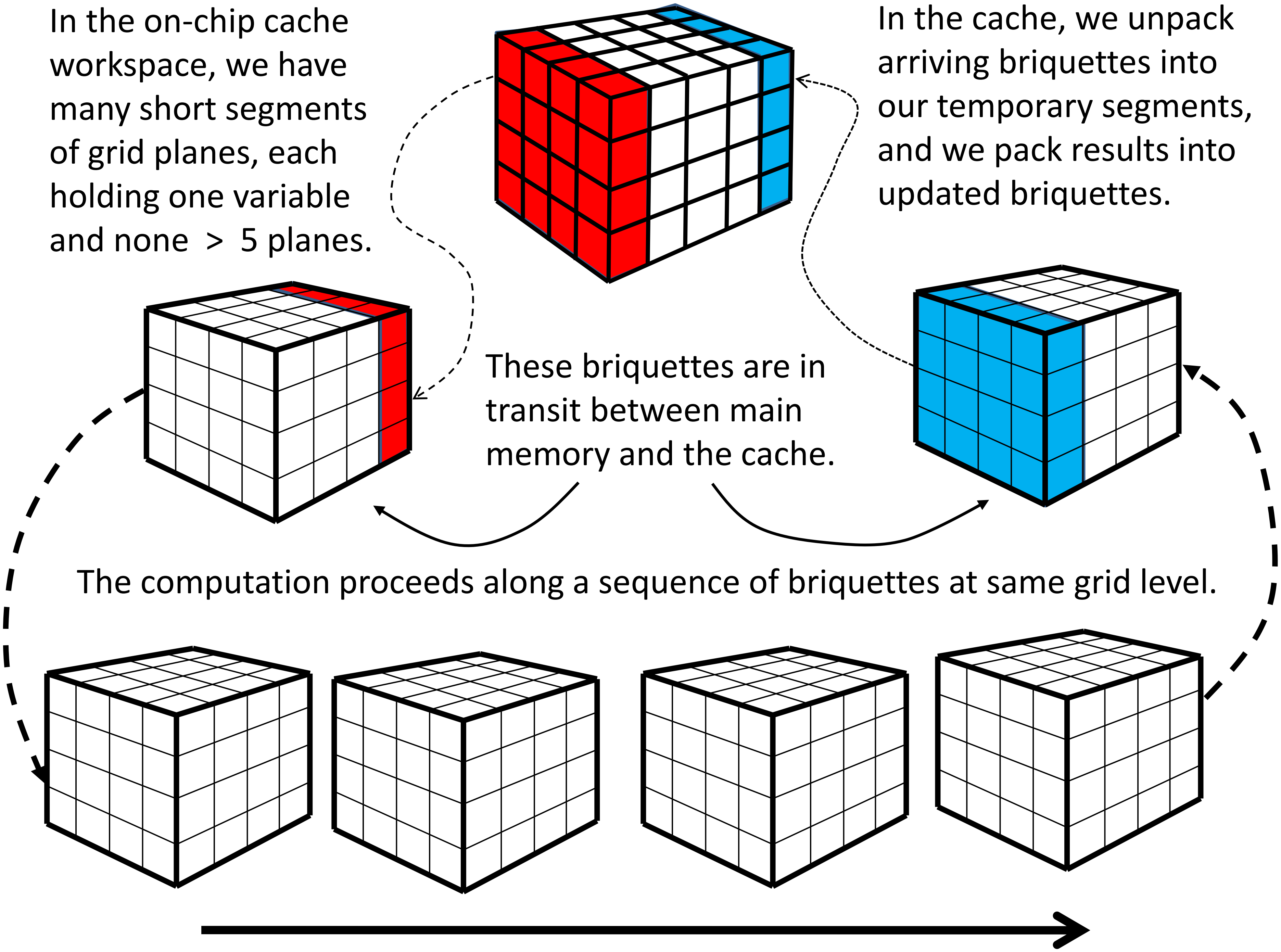
We force a minimal degree of uniformity at the microscale to accommodate needs of SIMD engine.

The highlighted “grid plane” will consist of either:

- 4 quadwords (Cell, Power7, Opteron, Nehalem),
- 2 octowords (Intel Sandy Bridge), or
- 1 hexadecaword (Intel MIC, Nvidia Fermi).

Process 2 grid planes at once for 32-wide SIMD of Nvidia Kepler.





In the on-chip cache workspace, we have many short segments of grid planes, each holding one variable and none > 5 planes.

In the cache, we unpack arriving briquettes into our temporary segments, and we pack results into updated briquettes.

These briquettes are in transit between main memory and the cache.

The computation proceeds along a sequence of briquettes at same grid level.

Briquettes & Pipelining-for-reuse

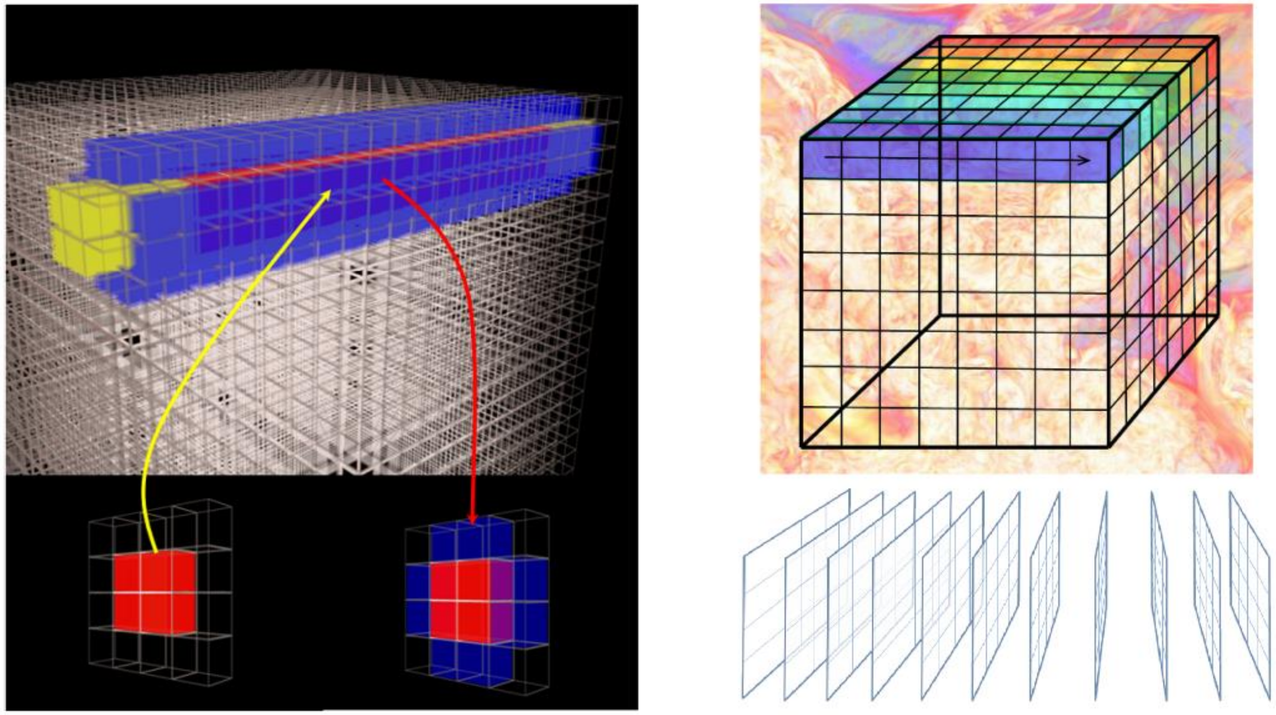


Figure 1. At the left, a grid pencil is indicated within its larger grid brick data structure. It consists of a core of $2^2\times16$ grid briquettes, shown in red, surrounded by transverse "ghost briquettes" shown in blue and with longitudinal ghost briquettes shown in yellow. A CPU core works its way down this grid pencil (from left to right in the figure) pulling into its cache memory groups of 4 core and 8 transverse ghost briquettes as shown. These are unpacked to produce vectors representing the 16 cells of single transverse core grid planes as indicated at the bottom right. These form the working data in the L1 cache for 16-long, aligned vector operations. Differences to obtain derivatives in this 1-D pass are formed by subtracting neighboring grid plane vectors along the longitudinal direction of the pass. The transverse ghost briquettes are used to construct, in the cache, 16-long vectors representing grid planes offset by one or two cells in a transverse direction. At the top right in the Figure, we illustrate how 8 processor cores may simultaneously update neighboring grid pencils. The ghost briquettes of one such grid pencil may be core briquettes of a neighboring one, so that they need be fetched from main memory only once into a shared on-chip data cache. This is a streaming paradigm of vector computation that works exceptionally well on modern multicore CPUs.

Woodward, P. R., J. Jayaraj, P.-H. Lin, P.-C. Yew, M. Knox, J. Greensky, A. Nowatzki, and K. Stoffels, "Boosting the performance of computational fluid dynamics codes for interactive super-computing," Proc. Intntl. Conf. on Comput. Sci., ICCS 2010, Amsterdam, Netherlands, May, 2010

Overcoming main memory bandwidth limitation

- We need about **220** temporary arrays per thread to update the problem state
- Through our optimizations, we reduced the workspace containing all the 220 temporaries to just **45.09 KB** per thread (*good for CPU or Xeon Phi, but not good enough for a GPU*).
- Text segment for computation region is **91 KB**.
- Text segment and workspaces of 2 to 4 threads can easily fit in the 256 KB or 512 KB L2 cache

Performance gains

Redundancy in computation eliminated

	Workspace / thread (KB)	flop/cell		
		Fortran-W	Pipelined	% redundancy
RK-adv	16.59	379.89	162.92	133.16
PPM-adv	19.2	454.61	273.31	66.34
tp3	208.28	5195.77	3218.67	61.43

Performance gains for PPM-adv

	Speed-up from		
	briquettes	pipelining-for-reuse & memory reduction	both
Nehalem	2x	3.33x	6.69x
Sandy Bridge	3.78x	1.66x	6.28x

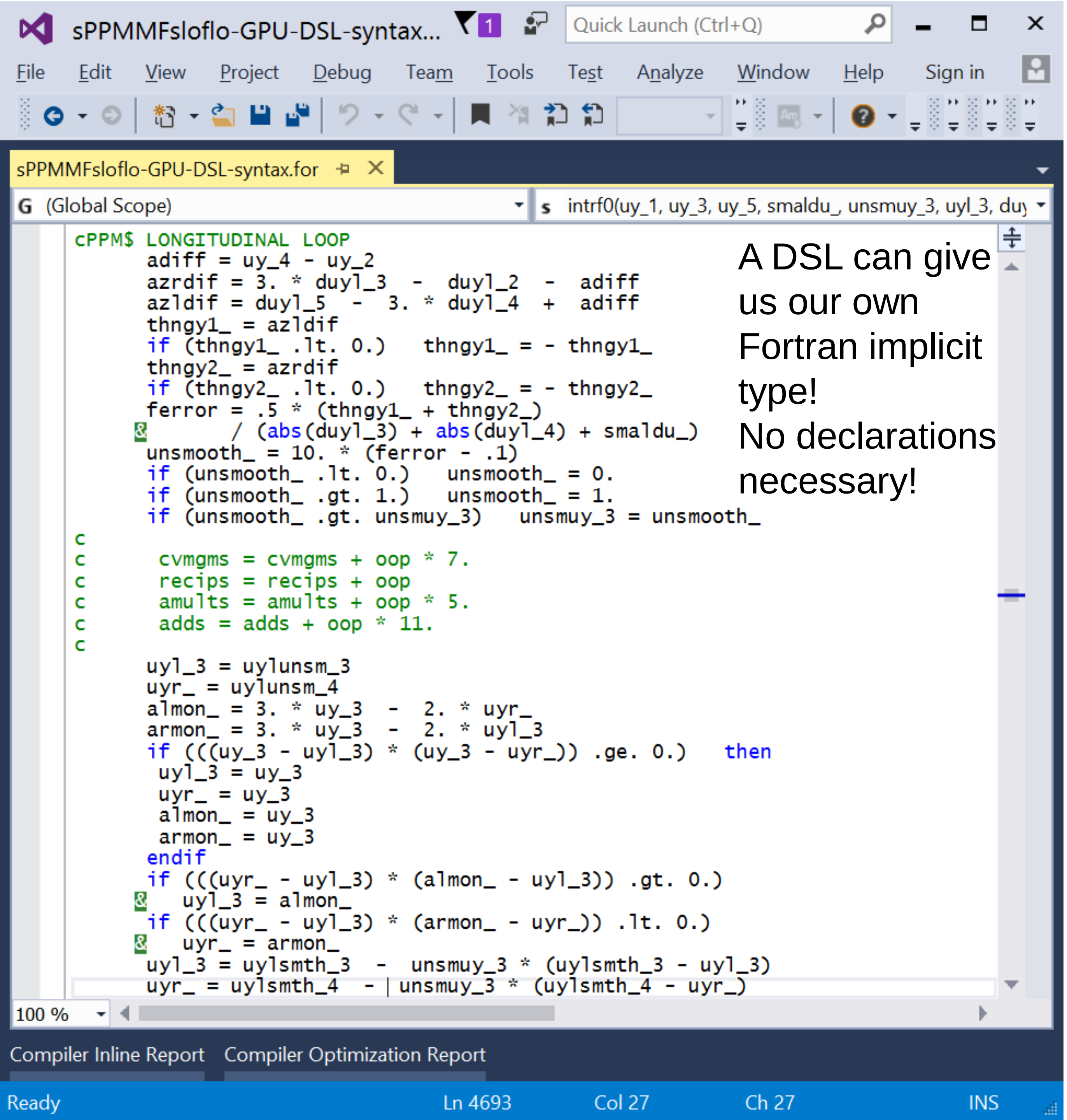
Nehalem : Xeon 5570 ; Intel 9 Fortran Compiler; 16 OpenMP threads running on two sockets
Dual-socket, 4-core @ 2.93GHz, SSE-4.2 (128-bit)
Sandy Bridge : Xeon ES-2670; Intel 13 Fortran Compiler; 32 OpenMP threads running on two sockets
Dual-socket, 8-core @ 2.6GHz, AVX (256-bit)
Expect performance to double by number of cores and double by increased vector widths (for vectorized sections), and decrease by 11 % for clock-frequency of Nehalem is higher (3.54x in total)

A Strategy for a Legacy Code: Potential for Incremental Adoption

1. Assume a code that uses as fundamental data structures lists of grid cells, lists of their neighbors, and lists of their grid cell faces, with lists of cells on either side of the faces.
2. With assistance from a DSL, rewrite the small portion of the code that produces the lists of cells and faces.
    a. This can be done independently of the routines that use these lists to do physics.
    b. Turn each cell into a briquette, and produce a 64-item list fragment for each briquette.
    c. Produce an equivalent, and shorter, list of briquettes.
3. What happens to the routines that use the lists?
    a. EXACTLY NOTHING.
    b. There is no benefit, but also no cost.
4. What happens to the routines that you rewrite, with assistance of the DSL, to use the new lists of briquettes?
    a. The indirect addressing performance hit vanishes.
    b. Everything vectorizes effortlessly.

Key Role of Automated Code Transformation – the DSL

1. Embedded DSL allows original code to be annotated, but otherwise not changed.
   a. Only minor revisions, and code still compiles and runs in original way, with all its pluses and minuses.
   b. Transformation of code is reversible – just discard the translated code, and even the translator, & you are back!
2. When you find a bug, don't panic. Just debug the translator.
   a. Automatically fix ALL bugs of this sort effortlessly, without having to find them all.
   b. Just like compiler bugs, you can code around them if you know which they are and the bugs are slow to be fixed.
3. Do not exploit your DSL developers.
   a. Do not ask them to do things that you can do easily.
   b. Do ask them to do things that they can do easily.
   c. Negotiate what is translated and what is not.
4. If the DSL is built easily, a goal of ROSE team, then you do not need to feel any guilt for all the benefits the DSL gives to you.


sPPMMFsloflo-GPU-DSL-syntax...
Quick Launch (Ctrl+Q)
File Edit View Project Debug Team Tools Test Analyze Window Help Sign in
sPPMMFsloflo-GPU-DSL-syntax.for
(Global Scope)
intrf0(uy_1, uy_3, uy_5, smaldu_, unsmuy_3, uyl_3, du
100 %
Compiler Inline Report
Compiler Optimization Report
Ready
Ln 4693
Col 27
Ch 27
INS

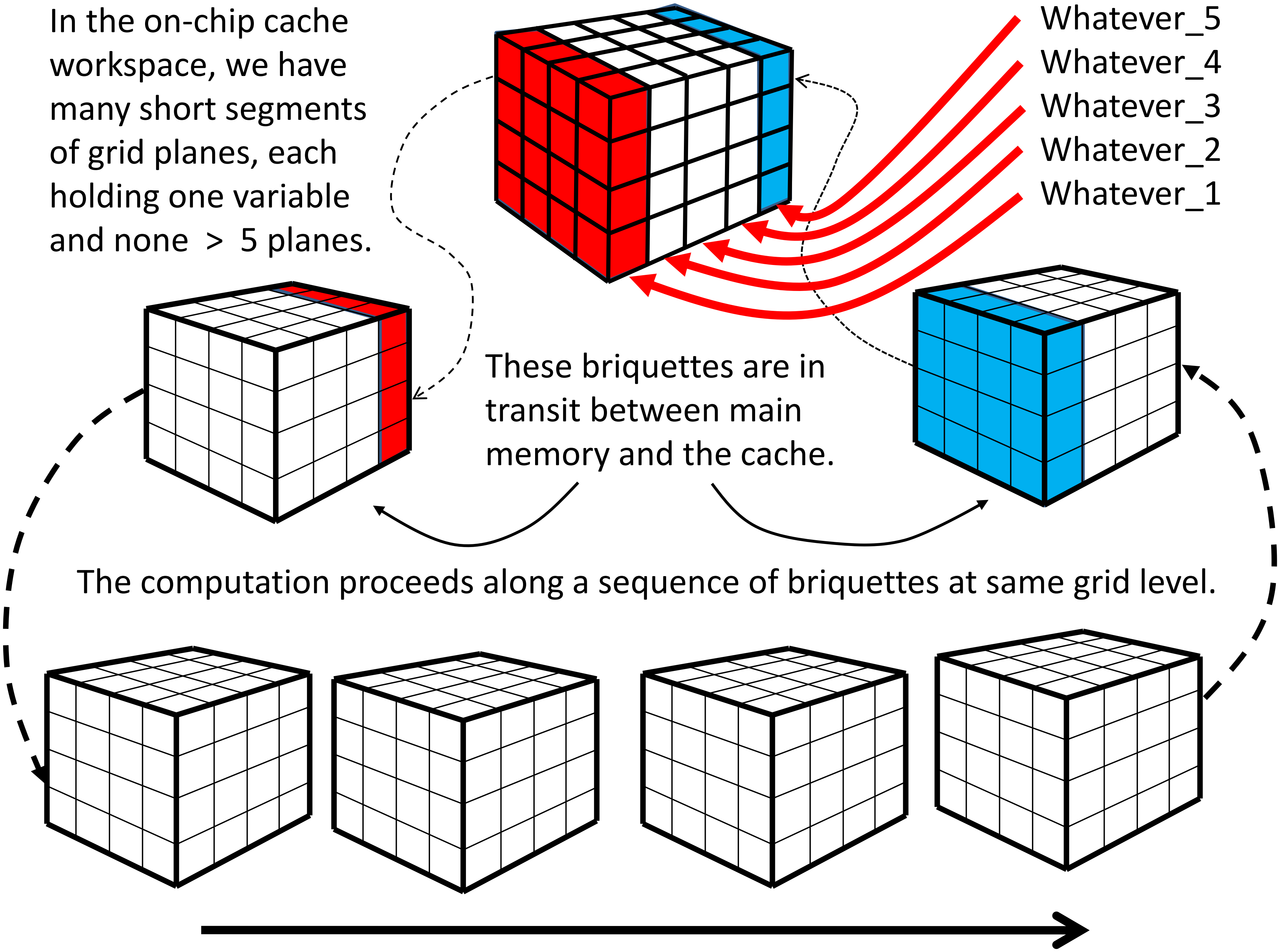
```
cPPM$ LONGITUDINAL LOOP
      adiff = uy_4 - uy_2
      azrdif = 3. * duyl_3  -  duyl_2  -  adiff
      azldif = duyl_5  -  3. * duyl_4  +  adiff
      thngy1_ = azldif
      if (thngy1_ .lt. 0.)   thngy1_ = - thngy1_
      thngy2_ = azrdif
      if (thngy2_ .lt. 0.)   thngy2_ = - thngy2_
      ferror = .5 * (thngy1_ + thngy2_)
     &       / (abs(duyl_3) + abs(duyl_4) + smaldu_)
      unsmooth_ = 10. * (ferror - .1)
      if (unsmooth_ .lt. 0.)   unsmooth_ = 0.
      if (unsmooth_ .gt. 1.)   unsmooth_ = 1.
      if (unsmooth_ .gt. unsmuy_3)   unsmuy_3 = unsmooth_
c
c       cvmgms = cvmgms + oop * 7.
c       recips = recips + oop
c       amults = amults + oop * 5.
c       adds = adds + oop * 11.
c
      uyl_3 = uylunsm_3
      uyr_ = uylunsm_4
      almon_ = 3. * uy_3  -  2. * uyr_
      armon_ = 3. * uy_3  -  2. * uyl_3
      if (((uy_3 - uyl_3) * (uy_3 - uyr_)) .ge. 0.)   then
       uyl_3 = uy_3
       uyr_ = uy_3
       almon_ = uy_3
       armon_ = uy_3
      endif
      if (((uyr_ - uyl_3) * (almon_ - uyl_3)) .gt. 0.)
     &   uyl_3 = almon_
      if (((uyr_ - uyl_3) * (armon_ - uyr_)) .lt. 0.)
     &   uyr_ = armon_
      uyl_3 = uylsmth_3  -  unsmuy_3 * (uylsmth_3 - uyl_3)
      uyr_ = uylsmth_4  -  unsmuy_3 * (uylsmth_4 - uyr_)
```

A DSL can give us our own Fortran implicit type!
No declarations necessary!

Variable_3 denotes a 32-word array consisting of the 3rd and 4th grid planes of the 6-plane working buffer for all variables in the transformed section.

duyl_2, duyl_3, duyl_4
Need not all be defined, because duyl_3 "obviously" Is just the second half of duyl_2 followed by the first half of duyl_4.

thngy1_
denotes a 32-word array of 2 grid planes that matches any position in the sequence _1, _2, …



In the on-chip cache workspace, we have many short segments of grid planes, each holding one variable and none > 5 planes.

These briquettes are in transit between main memory and the cache.

The computation proceeds along a sequence of briquettes at same grid level.

Future Challenges:

- ***As Moore's Law stalls, vendors might think about improving their designs to deliver more than 5% to 10% of peak***.
- **WE** would need to stop publishing LinPack performance.
- **WE** would need to stop writing codes that force data to make frequent, needless round trips between chip & main memory.
- **WE** would need to learn to love the cache.
- **THEY** would need to stop giving us pointless cache coherency.
- **THEY** would need to start giving us fast and cost-free register spilling to on-chip cache.
- **THEY** would need to give us fast semaphores requiring no electrons to leave the chip, so all those cores can COOPERATE.
- **WE ALL** would really like very fast chip interconnects. Bandwidth is key, and latency is not (we can plan).
- **WE ALL SHOULD** simply throw away all that expensive and power hungry DRAM. If it scales, it doesn't need DRAM.
- ***I, personally, want to write programs that treat the entire machine memory as shared. I promise not to abuse this***.

Co-Design Issues for Extreme Performance per Node:

1. Challenges for Many-Core Devices:
    a. Many cores require even many more threads.
        1) 2 threads per core is a minimum today.
        2) 4 threads per core keeps being threatened.
    b. NUMA concerns on the chip itself.
        1) Tasks performed by parallel threads are best today if they are independent.
        2) This model becomes increasingly wasteful of memory bandwidth as core counts increase.
        3) Cores can cooperate through a shared L3 cache.
        4) Cooperation through what amounts to message passing on the chip between L2 caches is looking increasingly promising.
        5) This adds a whole new layer to the memory and processing hierarchy.
        6) Knights Corner failed at this, but Knights Landing might succeed. GPUs cannot now do this at all.

Co-Design Issues for Extreme Performance per Node:

1. Challenges for GPU Devices:
   a. Need for truly tiny on-chip memory footprint.
      1) Forced by requiring minimum of 8 threads/core.
      2) Have essentially only L1 “caches” (actually local stores) on chip, 8 per core with 32 KB each.
      3) Can run up to 8 threads per L1, for 64 threads/core.
      4) On-chip storage is small, but computation there is very fast. Really is not a cache, but a local store.
      5) No false sharing, because no sharing, except for main memory & read-only items in “texture cache.”
      6) 8 threads/core produces only ¼ memory bandwidth

   b. Consequences of forcing at least 8 threads per core:
      1) Data must be read and written in a very special order that minimizes its on-chip residence time.
      2) Tasks making up computation must be performed in special order that produces intermediate data that can be almost immediately consumed.

<u>Co-Design Issues for Extreme Performance per Node</u>:

1. Challenges arising from CPU and GPU differences:
   a. ***GPUs require redundant computation*** to minimize on-chip memory footprint and avoid off-chip data access, but CPUs do not require this.
      1) ***Forces good GPU program to be fundamentally different from good CPU program***.
      2) Have found a way to accommodate this by variable degrees of pipelining, but this is not simple.

   b. ***GPU's 8 threads per core minimum cannot share on-chip data, but CPU's threads can do this***.
      1) ***Forces good programs for the two devices to be fundamentally different***, unless restrict self to programs that can run on both, and which are suboptimal on both.

   c. Preferred languages for two devices are different.
      1) Forces good programs to be syntactically different.
      2) Code translator can address this.

I am Very Hopeful Despite these Issues & Challenges:

1. ***Acceleratorsseem to be devices that the scientific community can substantially influence*.**
   a. ***It could be easier to convince a vendor to make a change if that vendor is used to serving a narrow range of uses and making changes for them*.**

Some History:

1. ***The strategy of the code design used in multifluid PPM has been set out in:***
   Woodward, P. R., Jagan Jayaraj, and Pei-Hung Lin, "Transforming Scientific Codes to Execute Efficiently on the IBM Cell Processor as well as on Other Multicore Microprocessor CPUs." LCSE Report, Nov. 9, 2006, available at www.lcse.umn.edu/F77-for-CELL.
   Woodward, P. R., J. Jayaraj, P.-H. Lin, and P.-C. Yew, 2008, "Moving Scientific Codes to Multicore Microprocessor CPUs," *Computing in Science & Engineering*, special issue on novel architectures, Nov., 2008, p. 16-25. Preprint available at www.lcse.umn.edu/CiSE.
   Woodward, P. R., J. Jayaraj, P.-H. Lin, and W. Dai 2009, "First Experience of Compressible Gas Dynamics Simulation on the Los Alamos Roadrunner Machine," *Concurrency and Computation Practice and Experience*, **21,** 2160-2175 (2009), preprint available at www.lcse.umn.edu/RR-experience.
   Woodward, P. R., J. Jayaraj, P.-H. Lin, P.-C. Yew, M. Knox, J. Greensky, A. Nowatzki, and K. Stoffels, "Boosting the performance of computational fluid dynamics codes for interactive supercomputing," Proc. Intntl. Conf. on Comput. Sci., ICCS 2010, Amsterdam, Netherlands, May, 2010. Preprint available at www.lcse.umn.edu/ICCS2010.
2. ***Co-design issues exposed by mPPM mini-app are discussed in:***
   Woodward, P. R., J. Jayaraj, and R. Barrett, "mPPM, Viewed as a Co-Design Effort," Proc. Co-HPC workshop, Supercomputing 2014, New Orleans, LA.

<u>Some History:</u>

3. ***The motivation for and results of building automated tools to execute this code design strategy has been set out in:***
   Lin, P.-H., J. Jayaraj, and P. R. Woodward, "A Study of the Performance of Multifluid PPM Gas Dynamics on CPUs and GPUs," Proc. SAAHPC Conference, Knoxville, Tennessee, July, 2011. Preprint available at www.lcse.umn.edu/SAAHPC and presentation available at http://www-users.cs.umn.edu/~phlin/pub/SAAHPC2011.pdf ;
   P.-H. Lin, J. Jayaraj, P. R. Woodward, and P.-C. Yew, 2011, "A Code Transformation Framework for Scientific Applications on Structured Grids," Technical Report 11-021, UMN Computer Science and Engineering Technical Report, Sept., 2011, available at https://wwws.cs.umn.edu/tech_reports_upload/tr2011/old_files/11-021.pdf ;
   Jayaraj, J., P.-H. Lin, P. R. Woodward. And P.-C. Yew, "CFD Builder: A Library Builder for Computational Fluid Dynamics," to appear in Proceedings of 28th IEEE International Parallel & Distributed Processing Symposium (IPDPS): Programming Models, Languages and Compilers Workshop for Manycore and Heterogeneous Architectures (PLC2014), Phoenix, AZ, 2014, preprint available at www.lcse.umn.edu/IPDPS2014.
   <u>Also the theses of Jagan Jayaraj and Pei-Hung Lin in 2013.</u>

Some History:

4. ***The simulations quoted in these slides have been published in:***
   Woodward, P. R., J. Jayaraj, P.-H. Lin, M. Knox, D. H. Porter, C. L. Fryer, G. Dimonte, C. C. Joggerst, G. M. Rockefeller, W. W. Dai, R. J. Kares, and V. A. Thomas, "Simulating Turbulent Mixing from Richtmyer-Meshkov and Rayleigh-Taylor Instabilities in Converging Geometries using Moving Cartesian Grids," Proc. NECDC2012, Oct., 2012, Livermore, Ca., LA-UR-13-20949; also available at www.lcse.umn.edu/NECDC2012;
   Woodward, P. R., Herwig, F., and Lin, P.-H., "Hydrodynamic Simulations of H Entrainment at the Top of He-Shell Flash Convection," *Astrophysical Journal* **798**, 49 (2015). arXiv:1307.3821, (2013).
   Herwig, F., P. R. Woodward, P.-H. Lin, Mike Knox, and C. L. Fryer, "Global Non-Spherical Oscillations in 3-D 4π Simulations of the H-Ingestion Flash," *Astrophysical Journal Letters* **792**, L3, preprint available at arXiv:1310.4584, (2014).
5. ***Acknowledgement:***
   The mPPM mini-app was developed under a contract with Sandia's Mantevo project; CFDbuilder development was supported through contracts with LANL and SNL; work with extreme-scale simulation of multi-fluid mixing in ICF and other contexts using pipelined briquette processing was supported through contracts with LANL; support for stellar hydro and interactive supercomputing was provided by NSF, also Blue Waters access.