It’s not my fault!
Finding errors in parallel codes

查找並行程序的錯誤

David Abramson

Research Computing Centre,
University of Queensland,
Brisbane
Australia

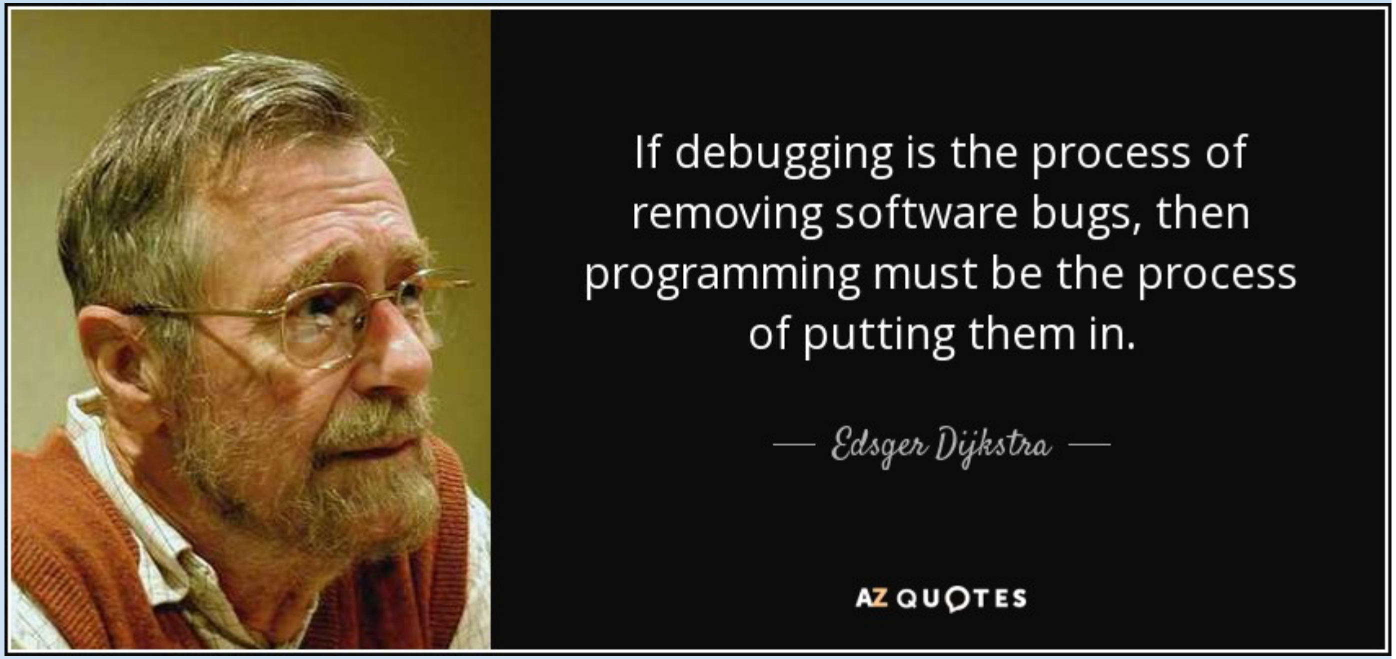
Minh Dinh (UQ)
Chao Jin (UQ)

Luiz DeRose (Cray)
Bob Moench (Cray)
Andrew Gontarek (Cray)

State of the art in debugging?

printf(“%f %f %f\n”, a[i], b[i], c[i])

a.out

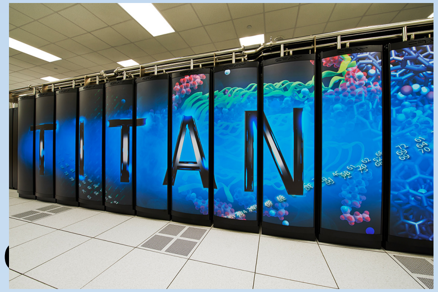


a.out > dumpfile; b.out > dumpfile1

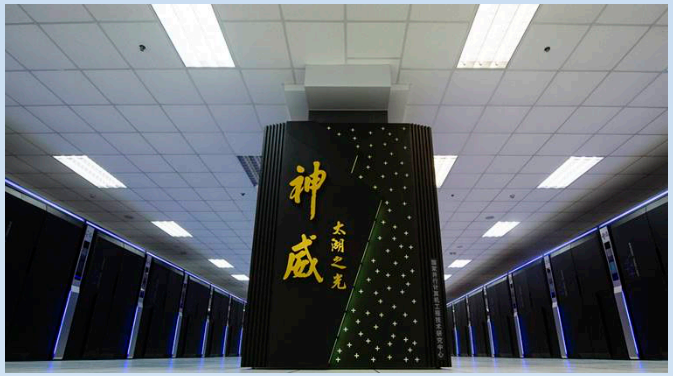
diff dumpfile1 dumpfile2

Supercomputing requires extreme debugging

- Titan
  - 299,008 AMD Opteron cores
  - 18,688 Nvidia Tesla K20 GPU Accelerators
  - 710 TB system memory, 32 GB + 6 GB per node (w/accelerator)
  - 18,688 compute nodes

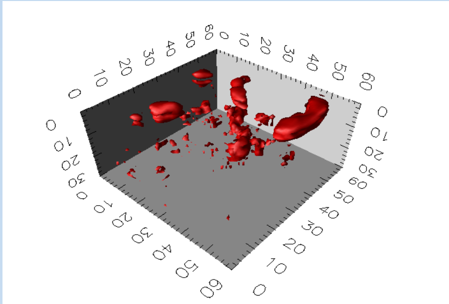


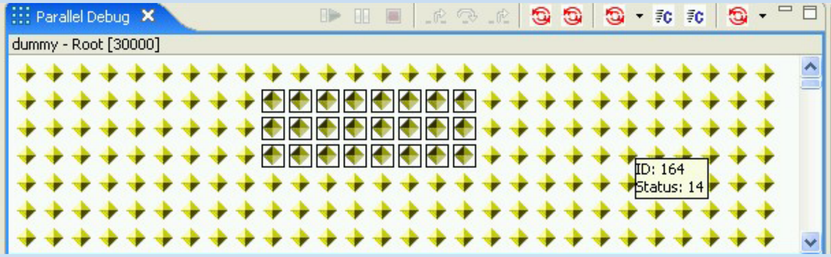
- Sunway TaihuLight
  - 40,960 SW26010 manycore 64-bit RISC processors
  - Each processor chip contains
    - 256 processing cores,
    - 4 auxiliary
  - 10,649,600 CPU cores

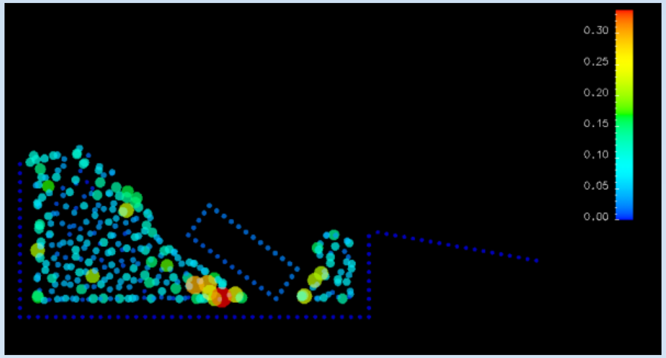


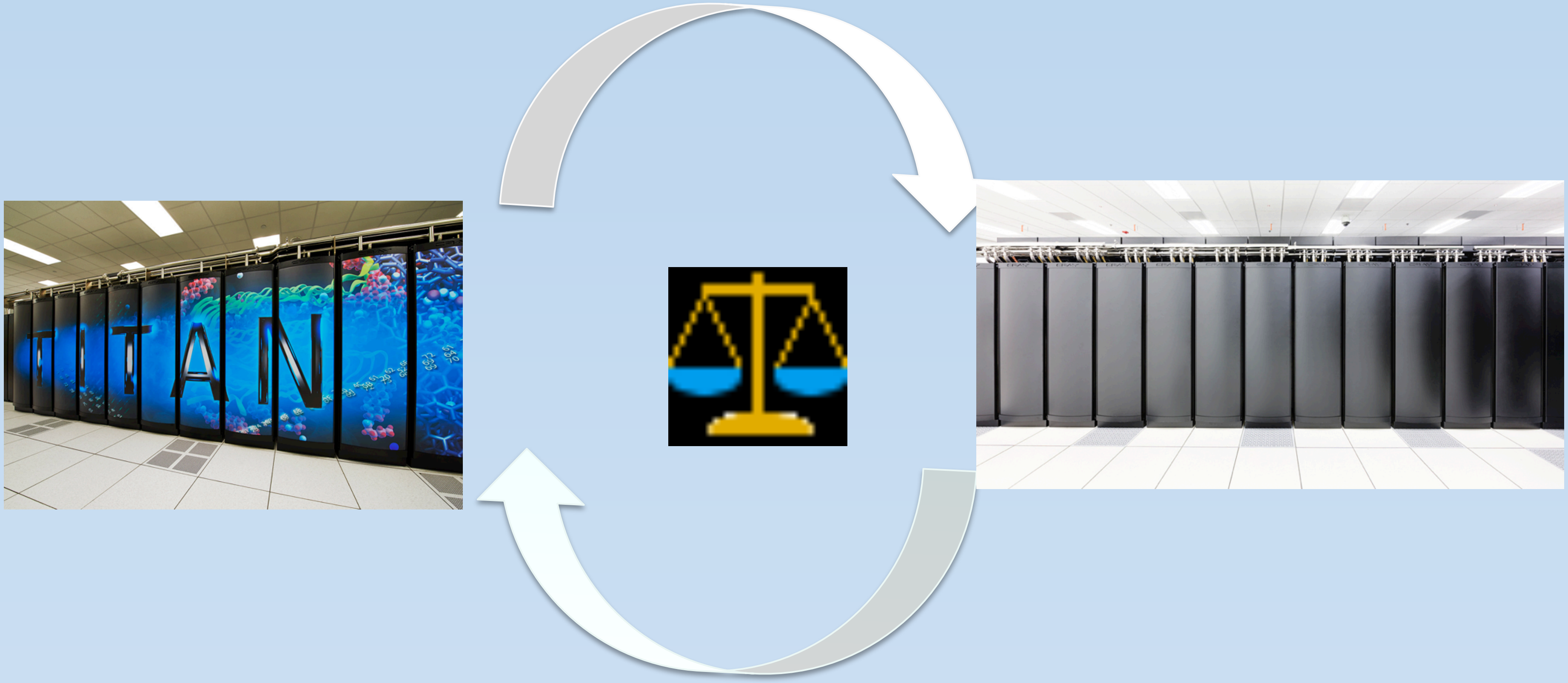
Debugging large codes

- *Cognitive* challenge
  - Large number of processes
    - Particular problems for UI
  - Large data structures
    - Infeasible to examine individual cells of multi-dimensional, floating point, structures.
  - Heterogeneity
    - A great source of errors
    - Hard to debug when do fail
- *Performance Challenge*
  - High level debugging is expensive
  - Debuggers generally don't use underlying parallel platform
- In the Exascale this just gets worse!









COMPARATIVE DEBUGGING

Debugging Evolved Applications

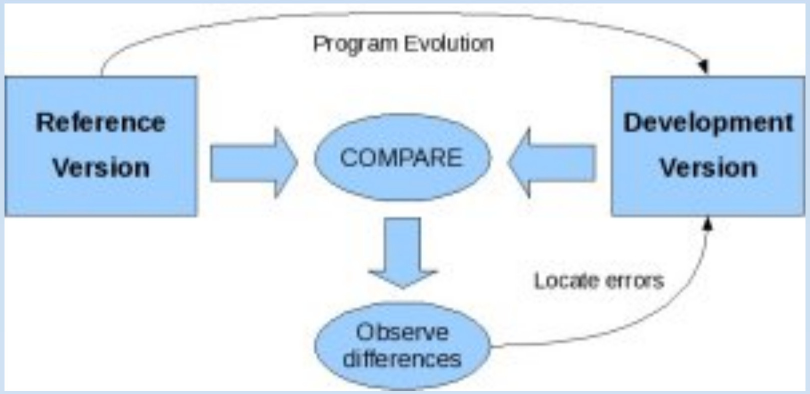
- Large codes are constantly evolving
  - User requirements
  - Underlying algorithms
  - New architectures

- Subtle errors occur often
  - Programmers spend lots of time debugging
  - Identify the source of a discrepancy
  - Follow it back to original source of deviation

Comparative Debugging

- What is comparative debugging?
  - Data centric approach
  - Two applications, same data
  - Key idea: The data should match
  - Quickly isolate deviating variables
  - Focus is on where deviations occur

- How does this help me?
  - Algorithm re-writes
  - Language ports
  - Different libraries/compilers
  - New architectures



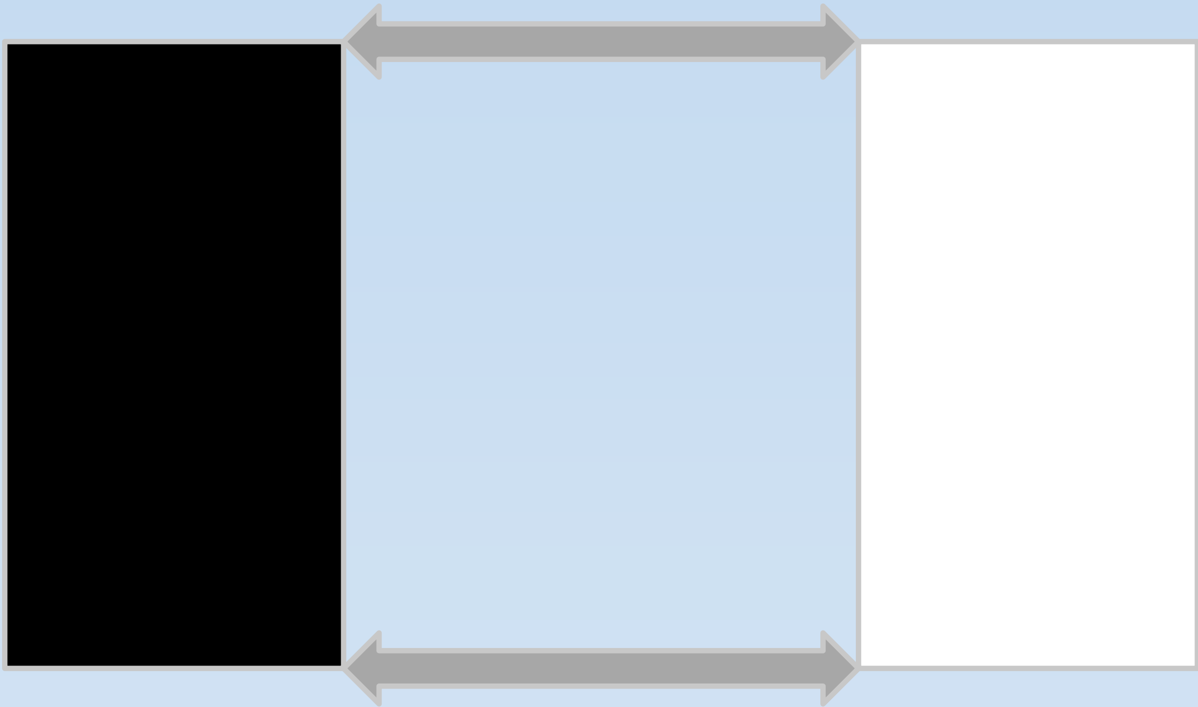
Comparative Debugging

- Specify conditions for correct behavior prior to execution
- Debugger:
  - keeps track of breakpoints
  - performs comparison automatically
- Control returned to user:
  - examination of state
  - continuation of execution

```
assert P1::big[100..199]@"file.c":240 = P2::small@"prog.f":300
```

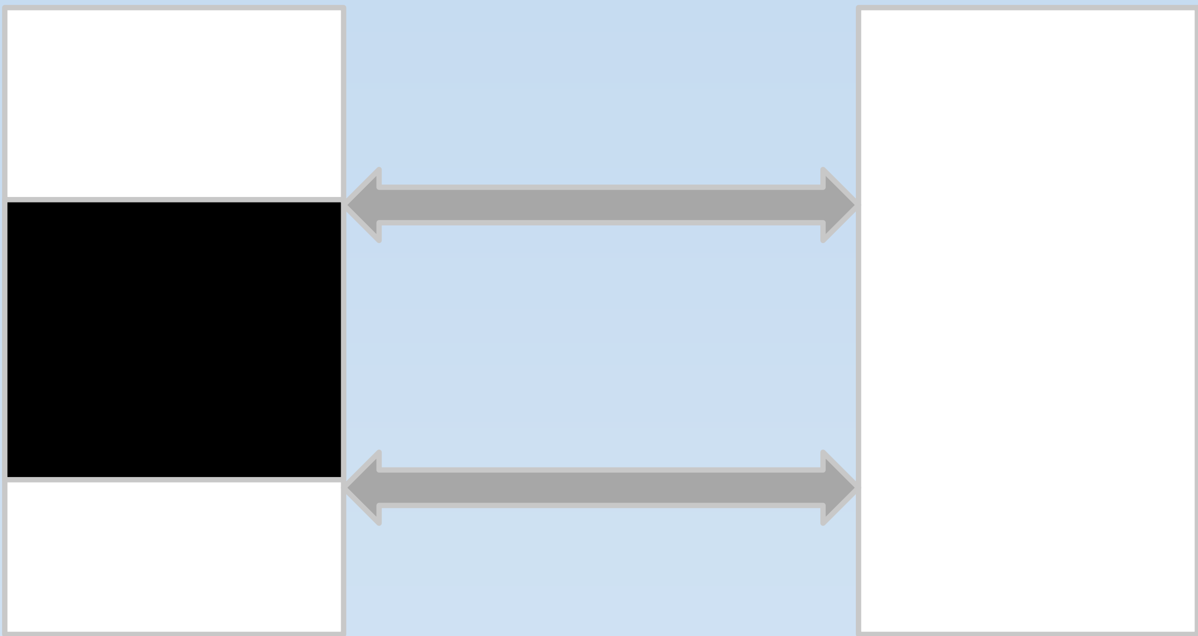
Why this works?

- Iterative refinement of problem area



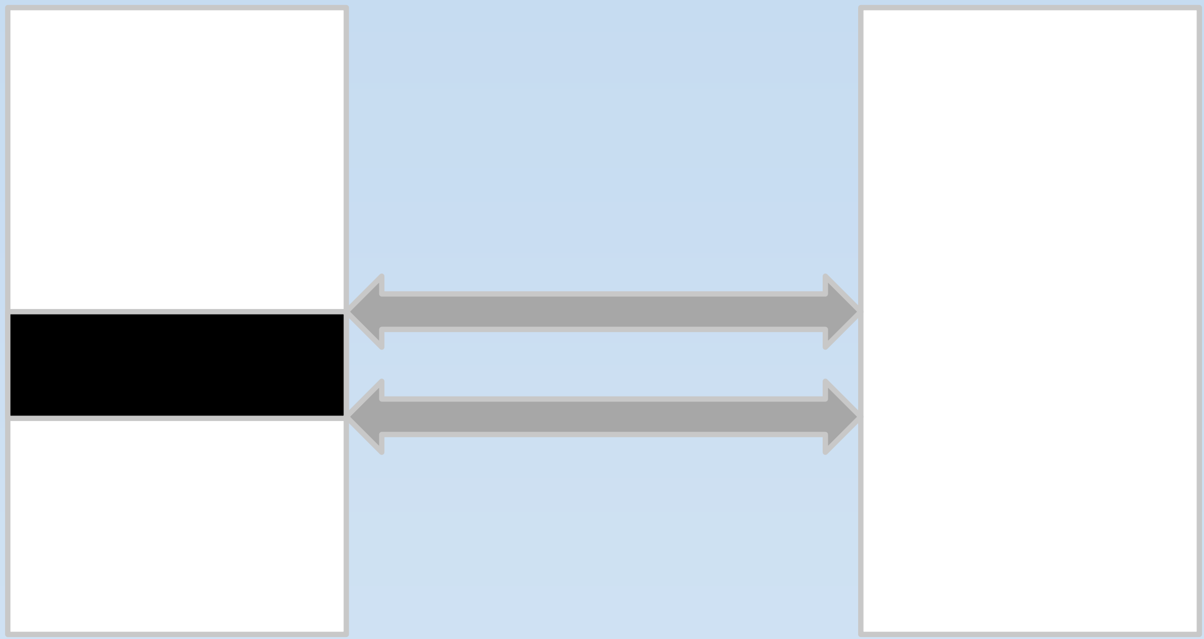
Why this works?

- Iterative refinement of problem area



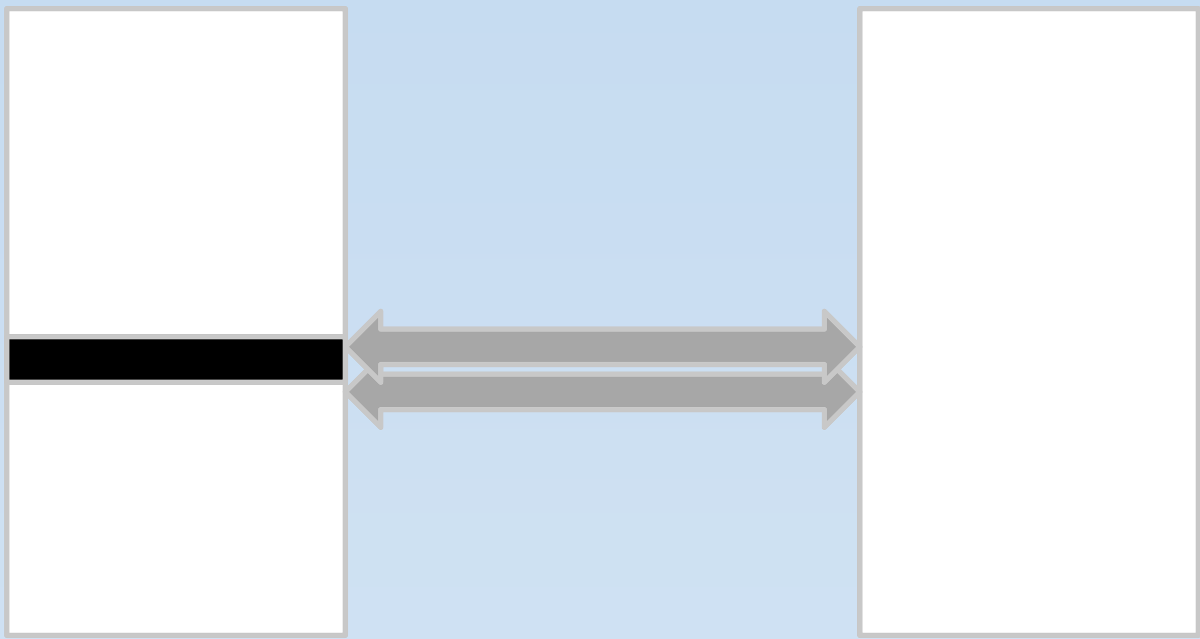
Why this works?

- Iterative refinement of problem area



Why this works?

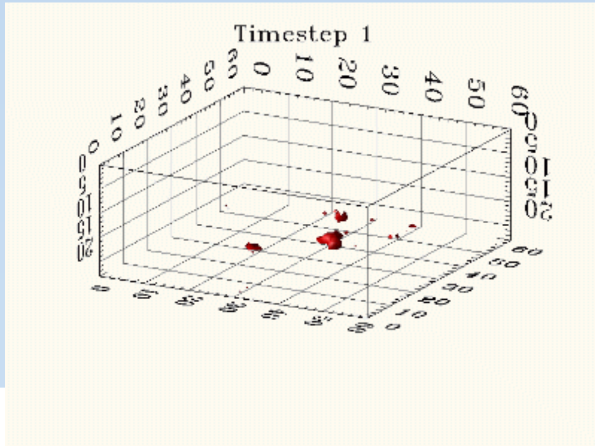
- Iterative refinement of problem area



VISUALIZATION

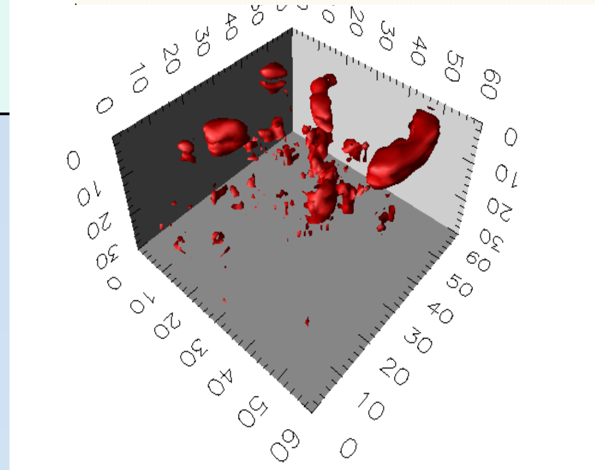
Reporting Differences

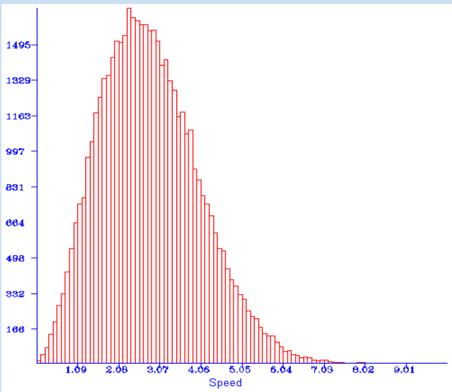
Movies



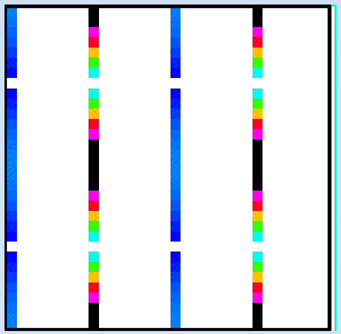
Values of scalars, small arrays

```
Starting execution of processes
Comparing c and c.
Maximum difference between values: 1.15442e-23
Total difference between values: 4.37116e-23
Number of differences detected = 823
First 10 errors are:
At Index : ( 30) = (Diff, Value 1, Value2) 0.000488
At Index : ( 32) ( 32) = (Diff, Value 1, Value2) 0.
```



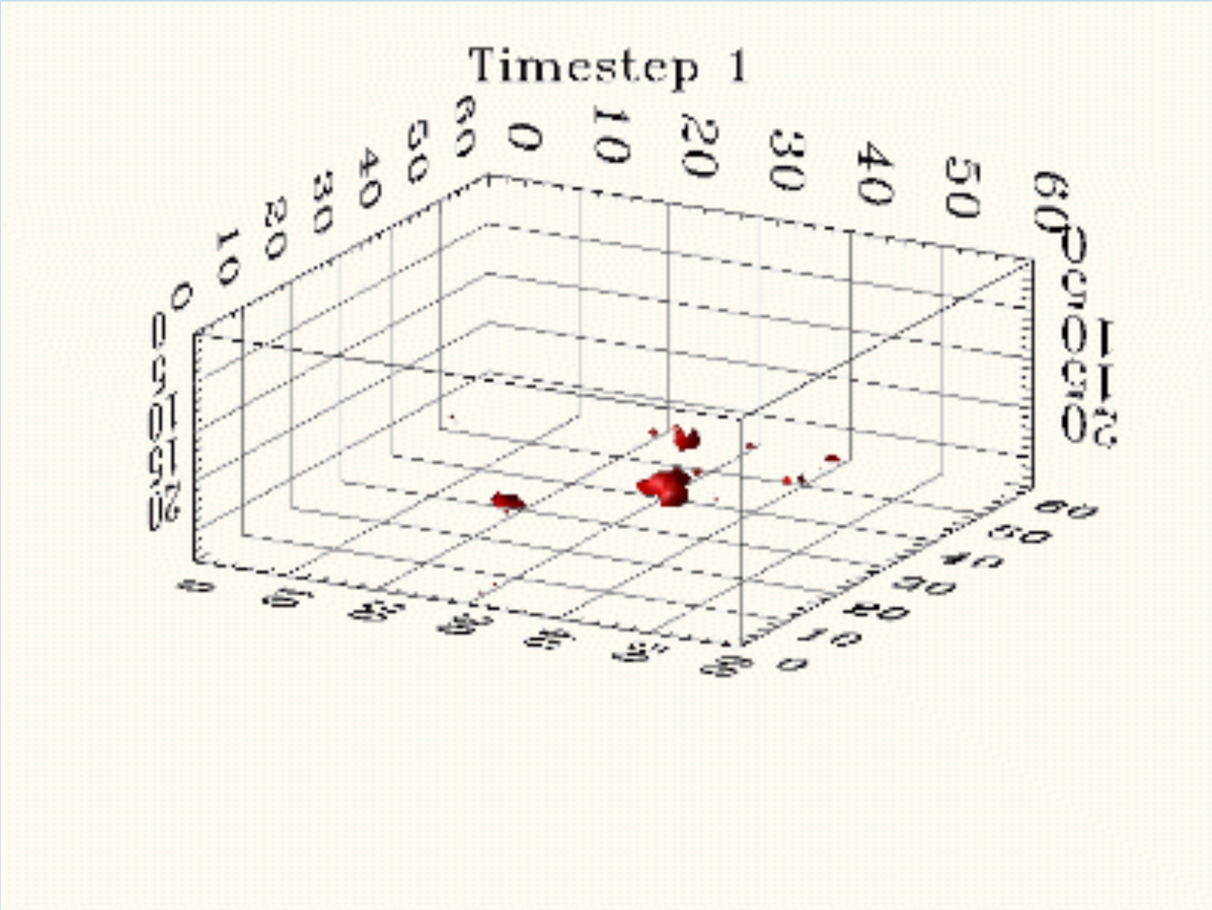


2-D pixel maps



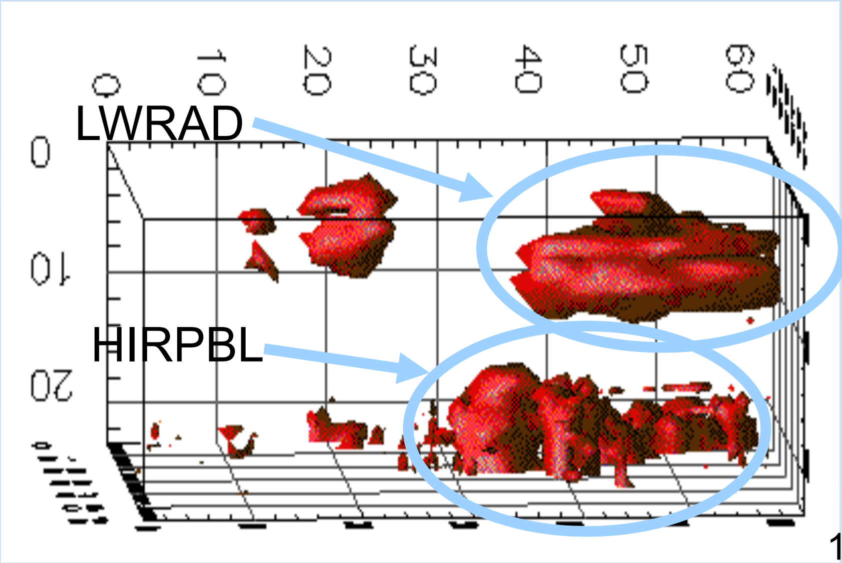
Multi-dimensional visualisation

The power of visualization

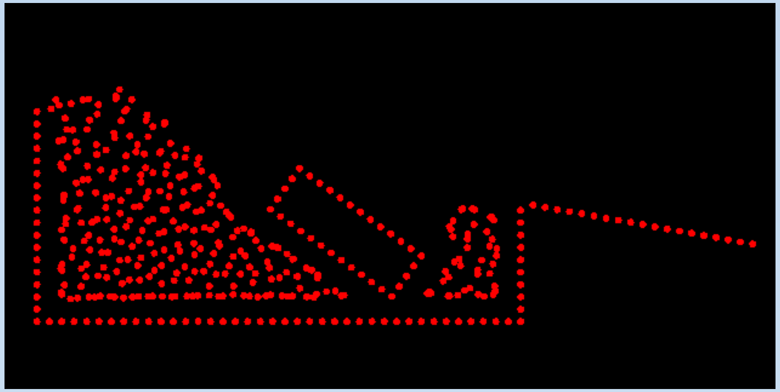


The power of visualization

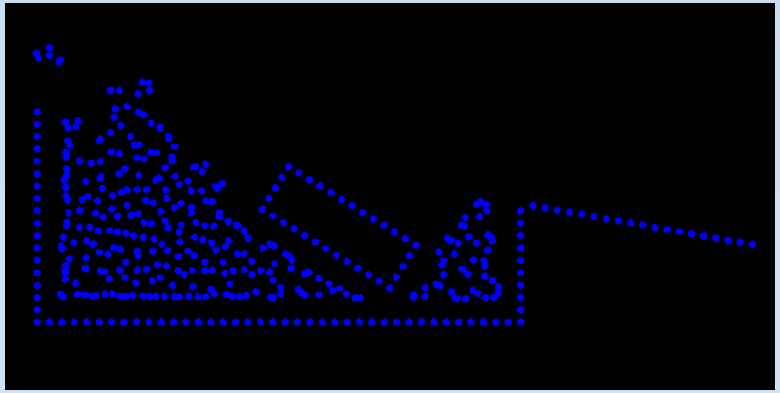
- Difference in physics of planetary boundary layer
  - Computation of #steps suited to parallel execution
  - Evident in 3 dimensional visualisation
- Error in radiation time step computation
- More complete physics in long wave radiation



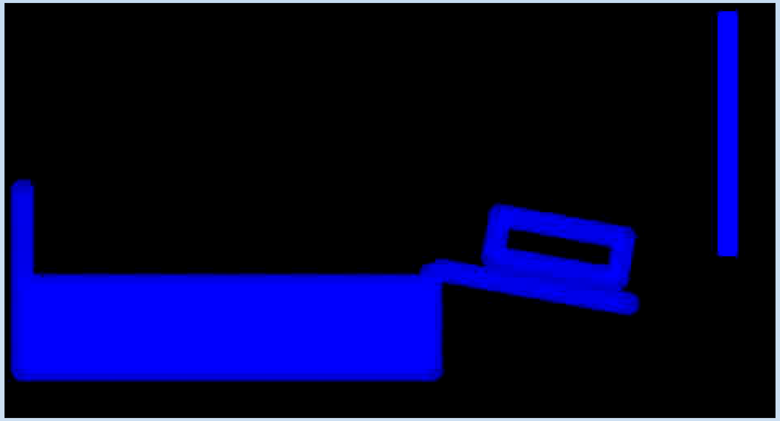
The power of visualization



-

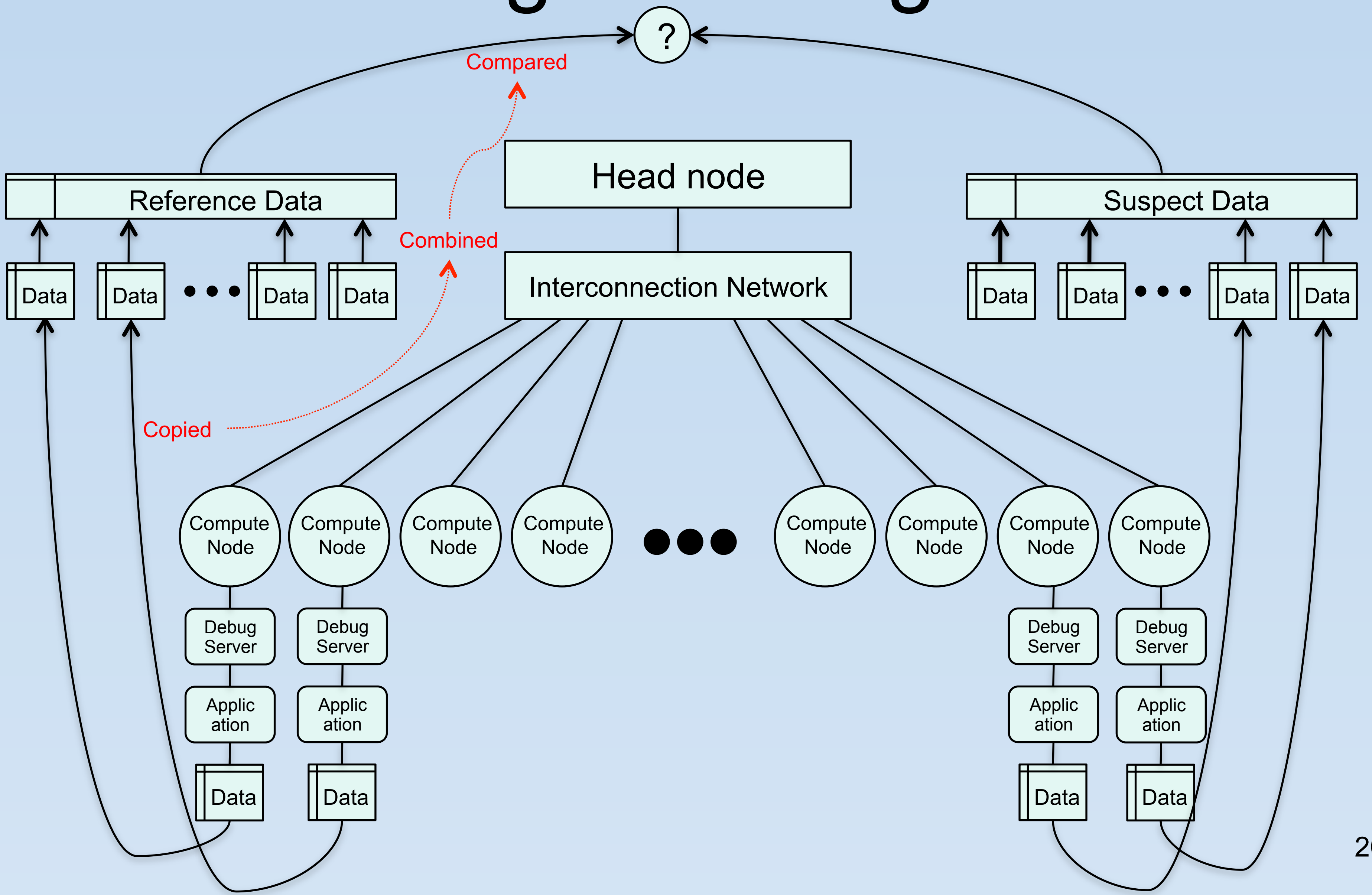


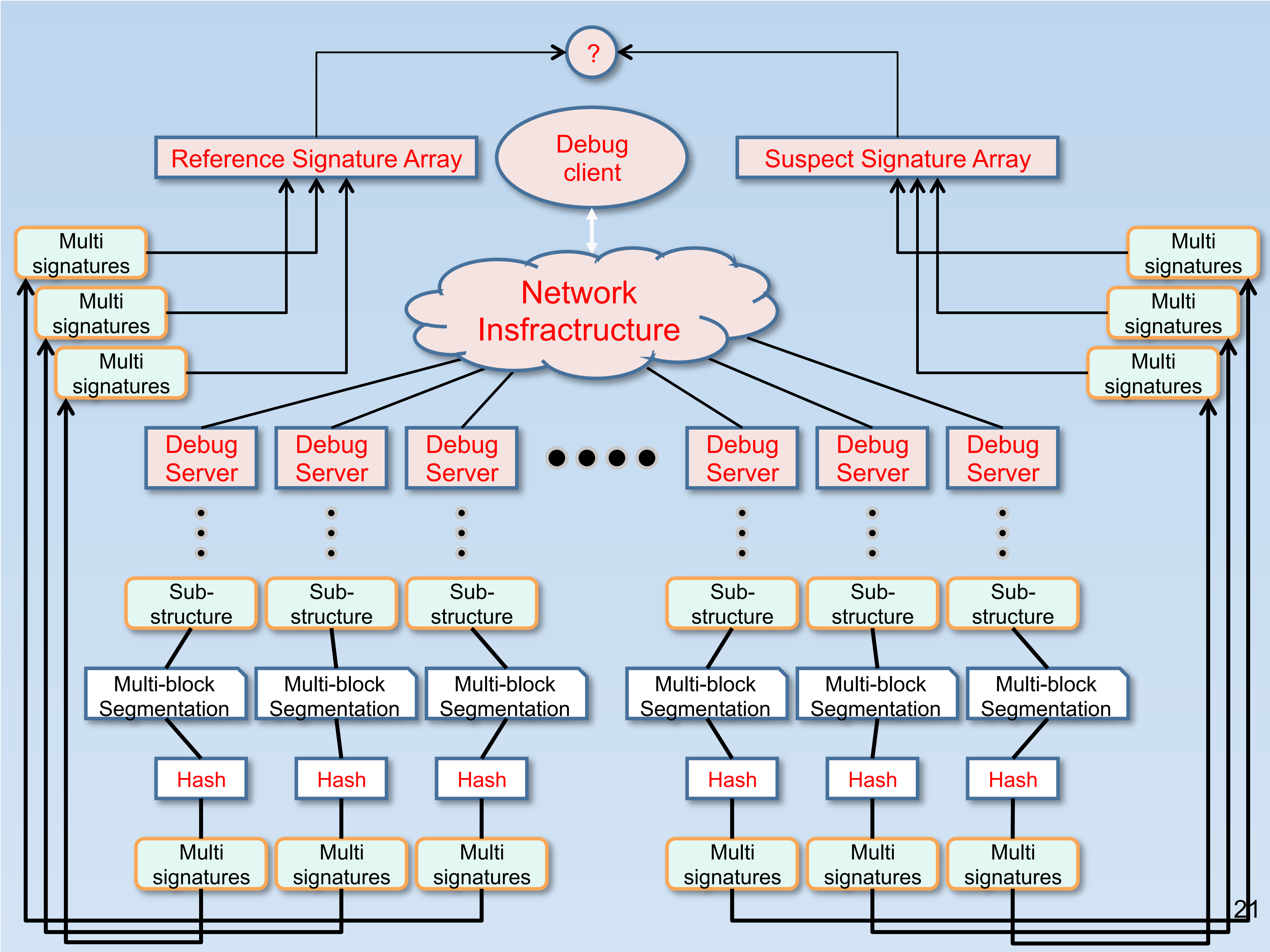
=



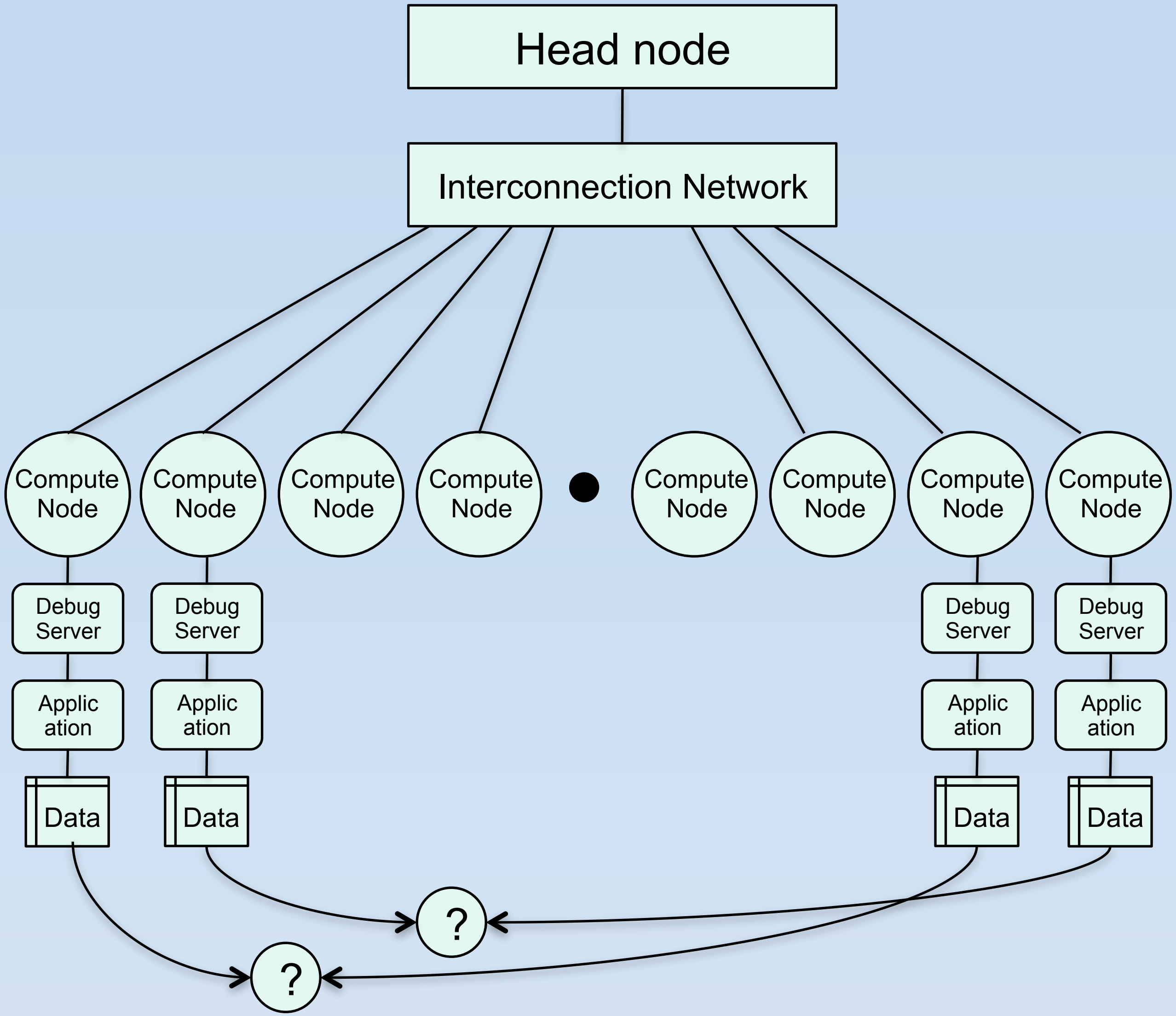
SCALABILITY

Original Design




?
Reference Signature Array
Debug client
Suspect Signature Array
Multi signatures
Multi signatures
Multi signatures
Network Insfractructure
Multi signatures
Multi signatures
Multi signatures
Debug Server
Debug Server
Debug Server
Debug Server
Debug Server
Debug Server
Sub-structure
Sub-structure
Sub-structure
Sub-structure
Sub-structure
Sub-structure
Multi-block Segmentation
Multi-block Segmentation
Multi-block Segmentation
Multi-block Segmentation
Multi-block Segmentation
Multi-block Segmentation
Hash
Hash
Hash
Hash
Hash
Hash
Multi signatures
Multi signatures
Multi signatures
Multi signatures
Multi signatures
Multi signatures

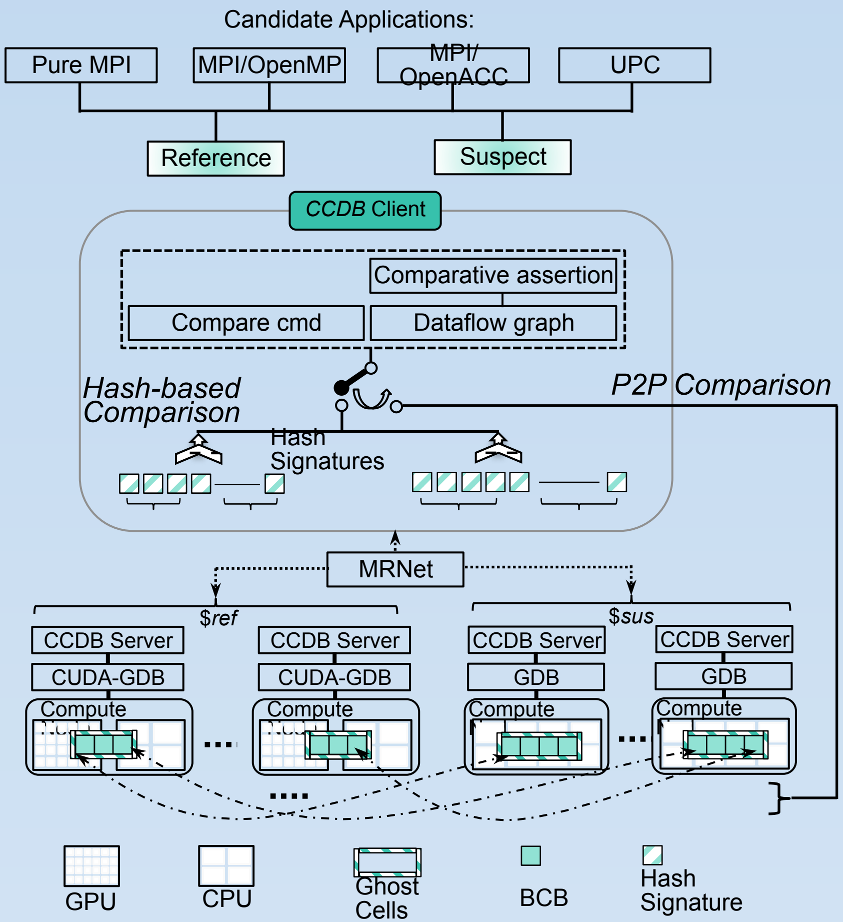
Point to Point protocol



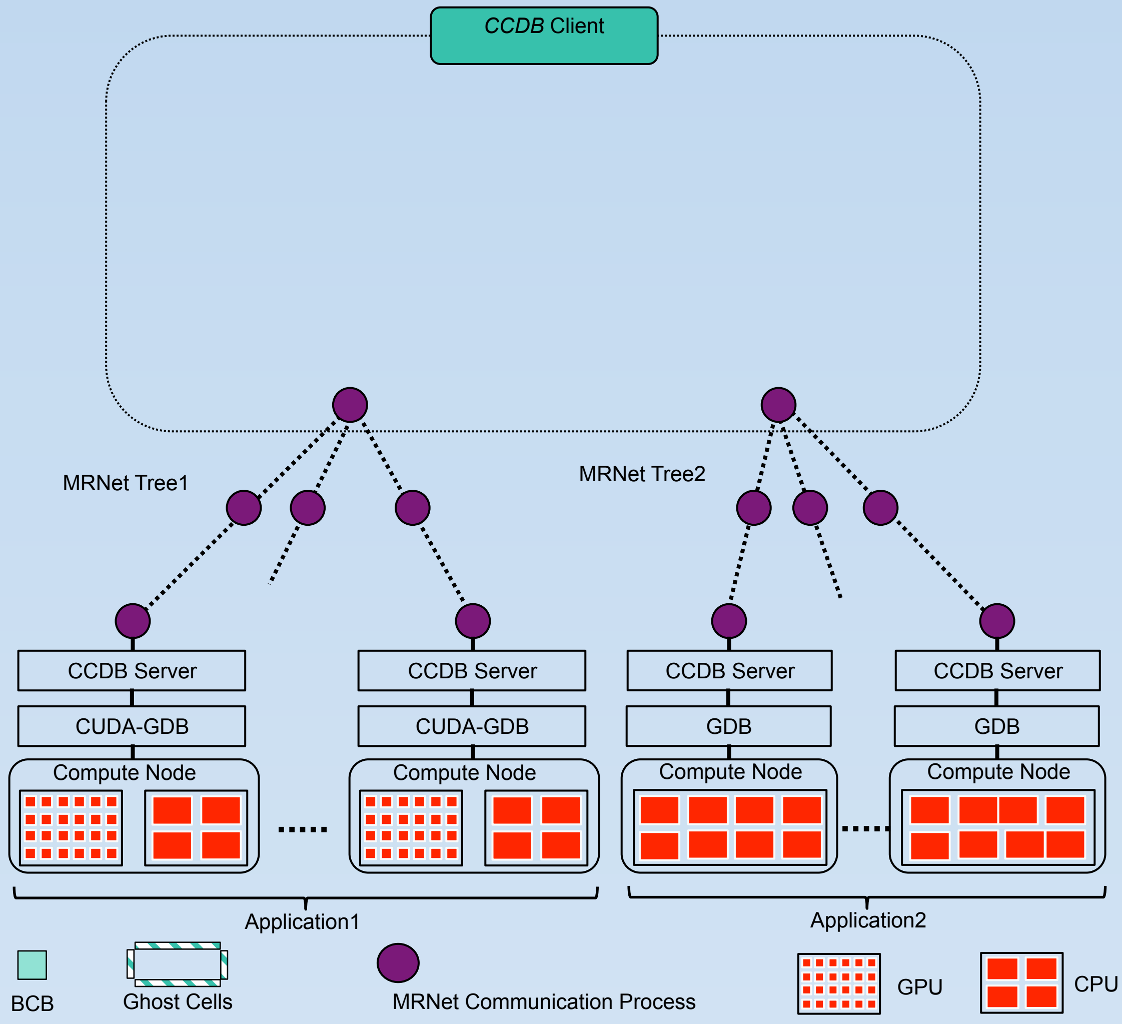
CCDB ARCHITECTURE

Overall architecture

- Scalable broadcasts and reductions
- Switchable backends
- Result aggregation

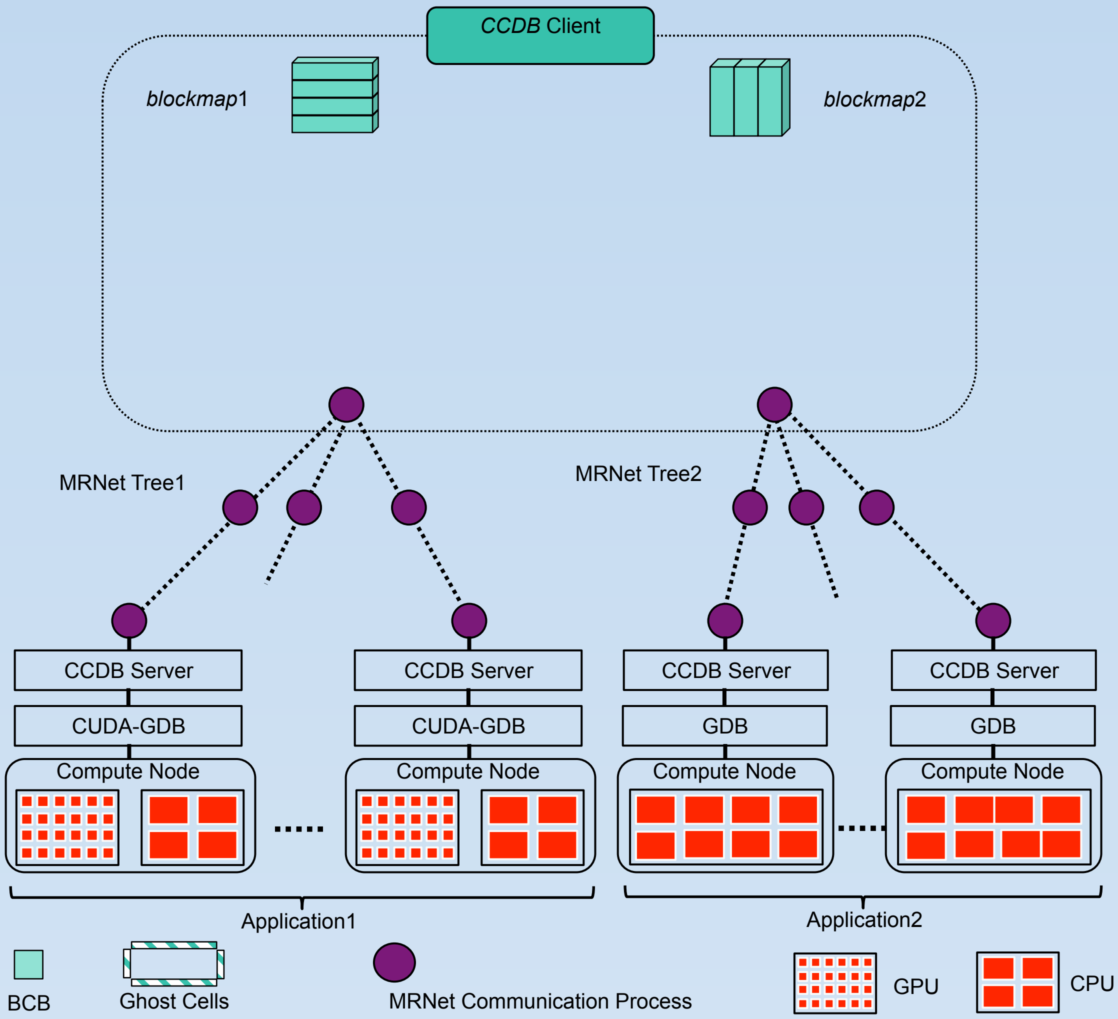


CCDB Architecture



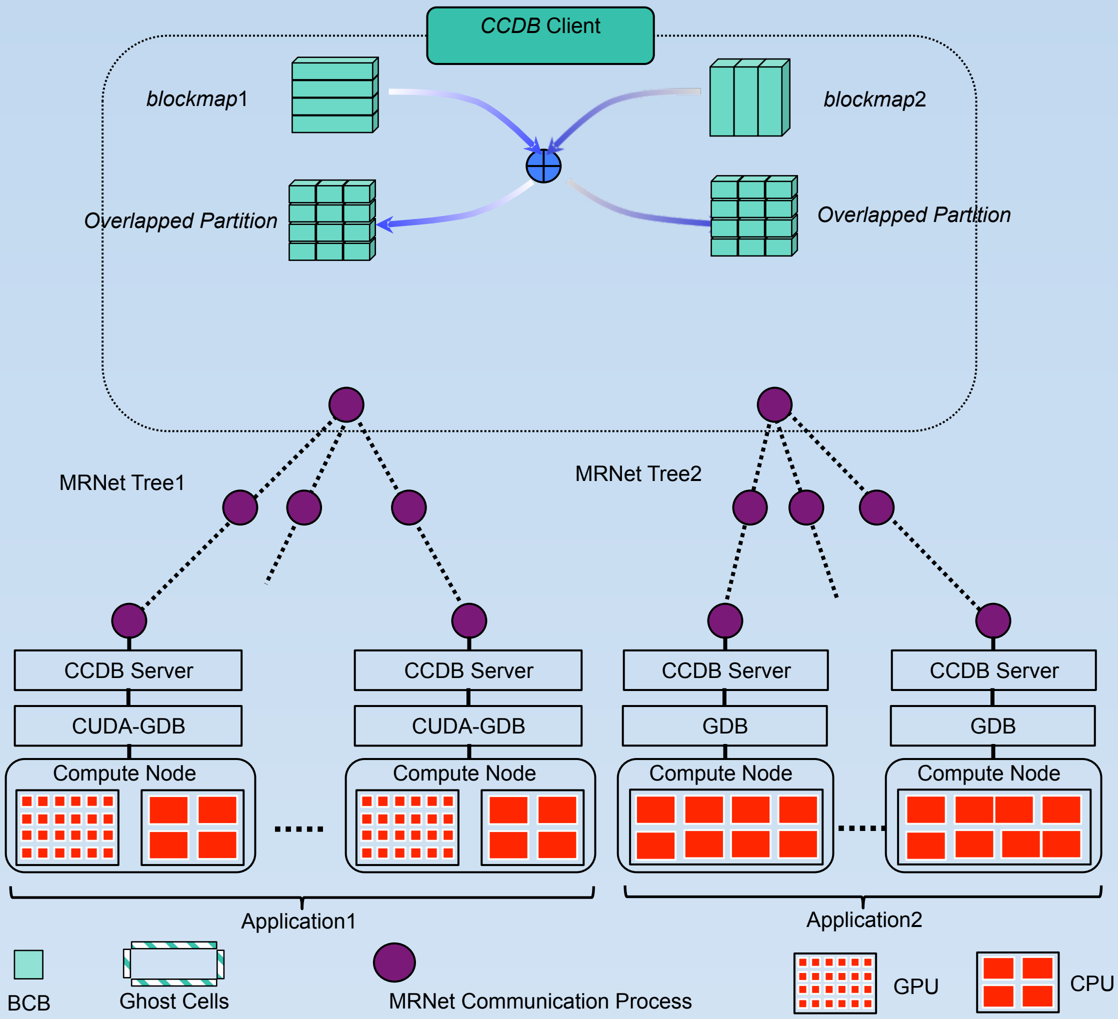
The architecture of CCDB is illustrated using the Hash-based comparison.[*]

CCDB Architecture



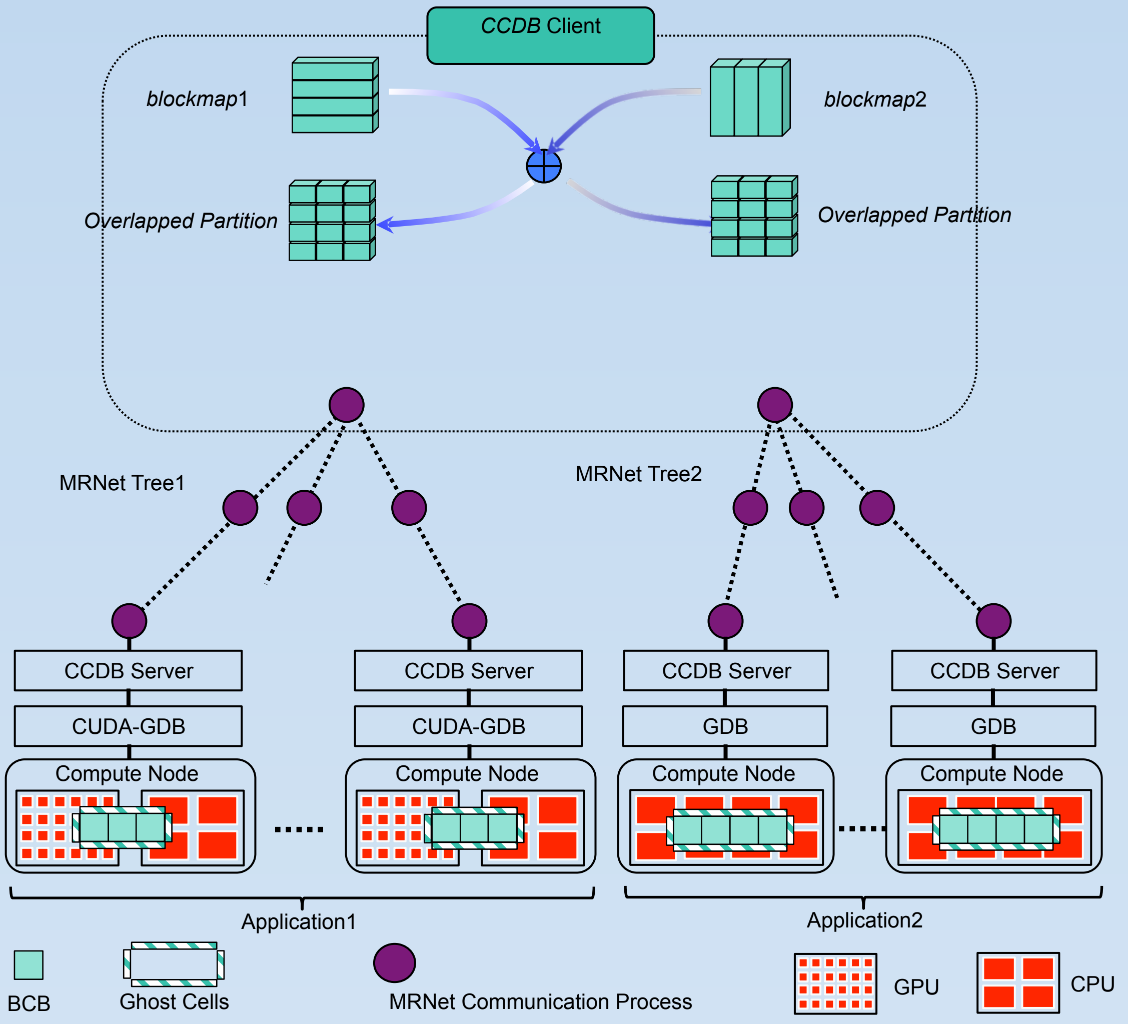
**The architecture of CCDB is illustrated using the Hash-based comparison.*

CCDB Architecture



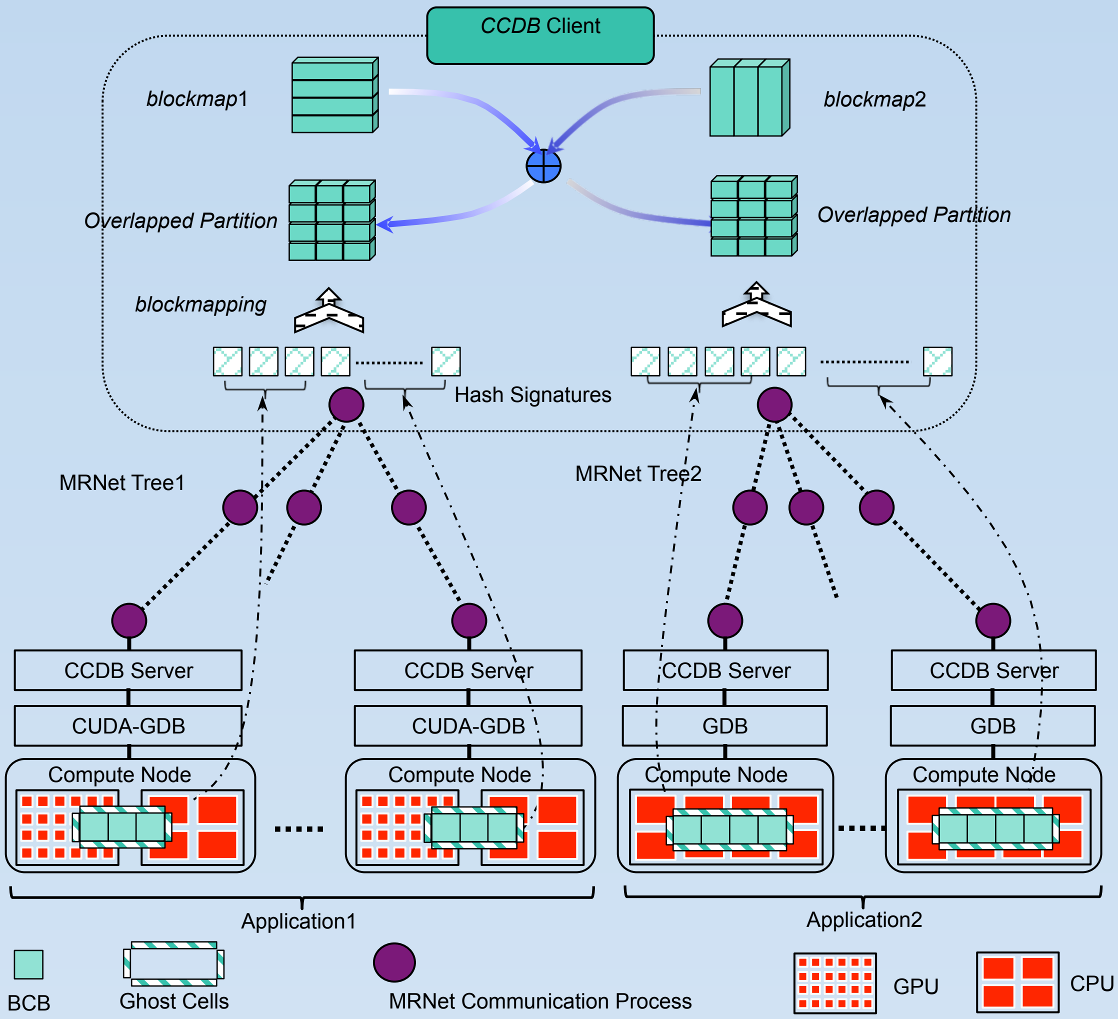
*The architecture of CCDB is illustrated using the Hash-based comparison.

CCDB Architecture



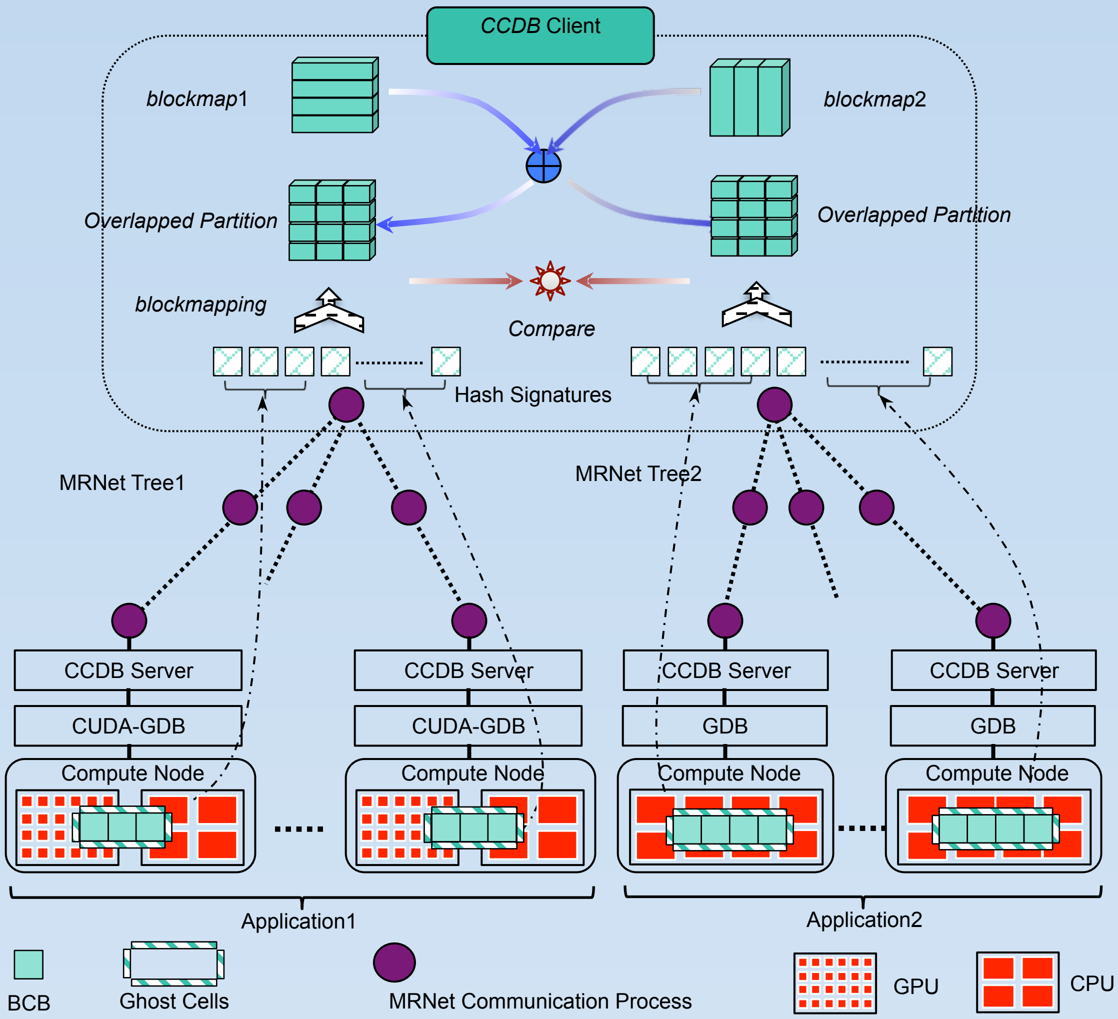
The architecture of CCDB is illustrated using the Hash-based comparison.

CCDB Architecture



**The architecture of CCDB is illustrated using the Hash-based comparison.*

CCDB Architecture



The architecture of CCDB is illustrated using the Hash-based comparison.

HETEROGENEITY AND IMPLEMENTATION TECHNIQUES

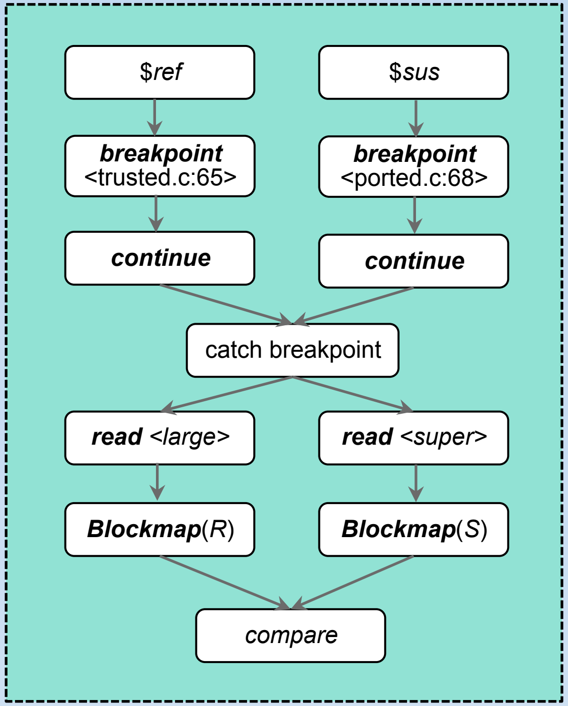
Assertion Engine

- Application processes run asynchronously
- Multiple assertions, can share same line numbers or variables
- Assertions specify breakpoint locations in processes
  - multiple breakpoints reached at any time
  - need to read data from process at breakpoint
- Comparison process is automated
- Stop execution when threshold reached

⇒DATAFLOW

Dataflow Engine

- Supports asynchronous behavior in debugged processes
- Flexible assertion structure
  - Single program assertions
  - Cross coupled assertions
  - Multi-process parallel programs



assert R($ref::large)*@trusted.c:65 = *S($sus::super)@ported.c:68

Architecture Independent Format

- Ability to represent data from different architectures in an architecture neutral way
- Need to perform numerical operations on data in this format
- Need to be able to convert to/from native formats

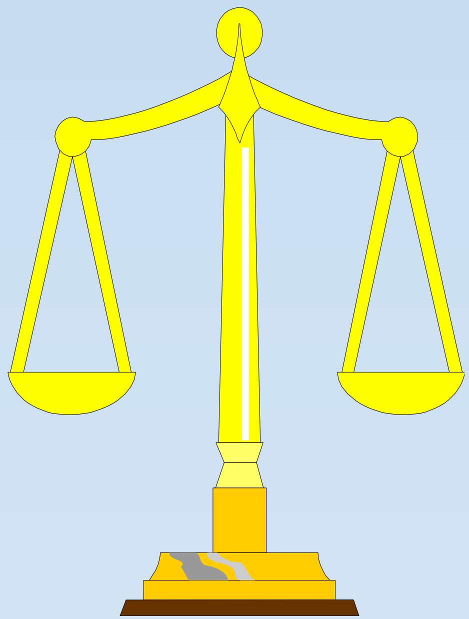
```
struct {
  int a;
  float b[3];
};
```

```
{a:is4,b:[r0..2is4]f4}
```

byte 1	byte 2	byte 3	byte 4
s	exponent	mantissa	
s	exponent	mantissa	
s	exponent	mantissa	

Flexibility in Comparisons

- Tolerances used for inexact equality
- Data structures should be:
  - type conformant (with conversion)
  - same size, but can be differing shapes
- Arrays
  - Differences allowed are:
    - offset ranges in arrays
    - ordering of indexes
    - Number of indexes
    - Language
- Dynamic data
  - Linked lists
  - Objects



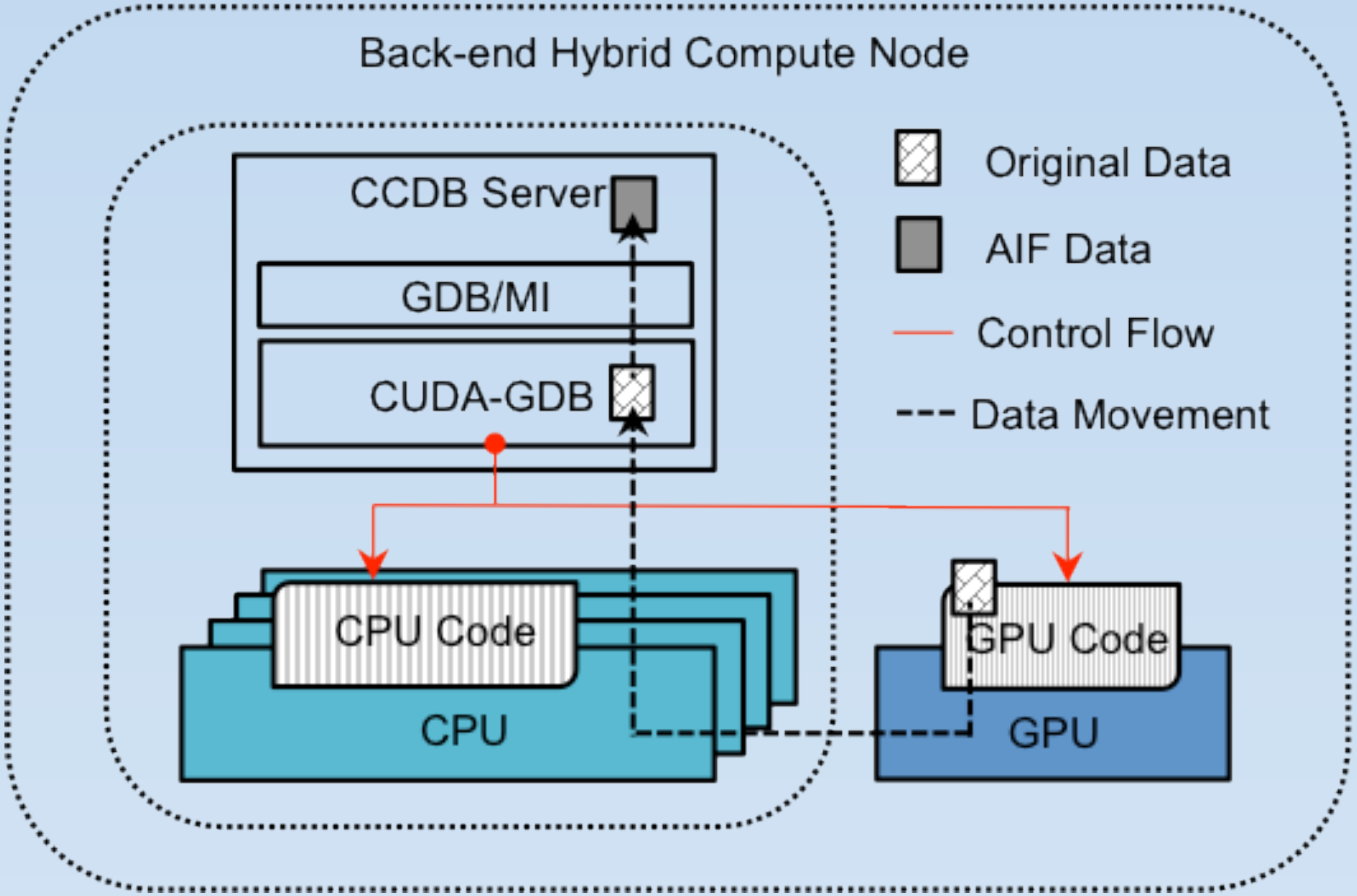
Programming Languages other than C/F

- OpenACC/OpenMP
  - Sequential regions executed on CPU
  - Parallel regions offloaded to GPU
  - Data dynamically moves between CPU and GPU
  - Separated address spaces for CPU and GPU codes
  - Inconsistent precision of floating numbers across CPU and GPU
- UPC: a virtual global memory space
  - Automatically decomposing the global data across a number of SPMD threads
  - Exchanging data between threads is managed by the UPC runtime system

Implementation

- Modification of CUDA-GDB
  - Automatically identify the variables residing on the GPU device attached to a Cray system
  - Move the data of a targeted GPU variable into the memory space of CUDA-GDB (in the memory of the host)
    - This enhancement is implemented using the debugger API provided by NVIDIA for GDB
- Tolerance threshold for comparing floating numbers
  - Truncate floating numbers to the same precision before they are converted into AIF.

The CCDB server on a hybrid node



Supporting UPC

- Affinity in UPC
  - Describes different domain decompositions
  - A user can provide a blocking-factor to achieve different decomposition schemas
- Implementation
  - Retrieve affinity metadata
  - Automatically generating a blockmap function, called auto-blockmap
  - Reconstruct a UPC global-shared array using the auto-blockmap function

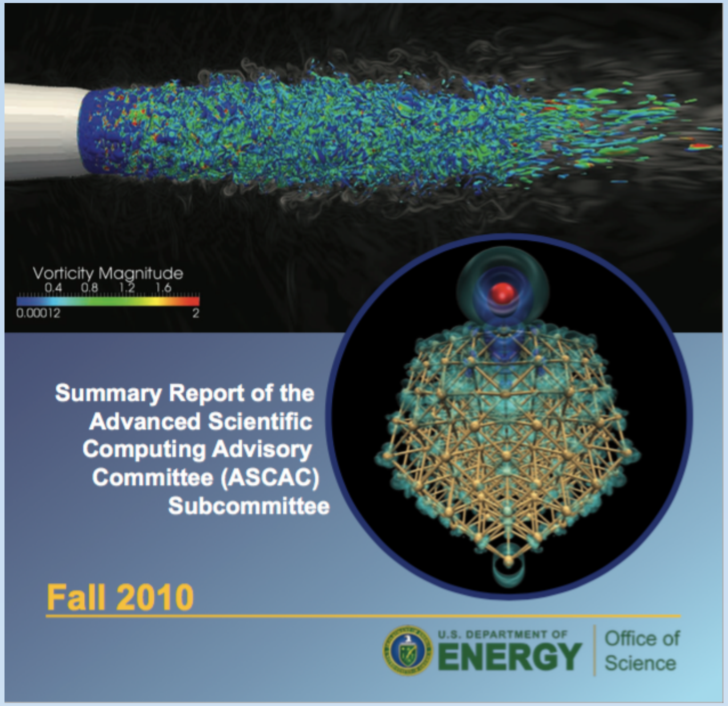
CCDB on Cray supercomputers

- Supporting Cray XE, XK, and XC supercomputers
- CCDB client: a comparative debugging interface
  - Launching parallel applications onto the back-end
  - Controlling the execution of the programs remotely
  - Compare key data structures between different applications
- CCDB server: a pluggable architecture
  - GDB: C, Fortran, and UPC programs
  - CUDA-GDB: OpenACC, OpenMP
  - MRNet
  - Scalable communication between the CCDB client and servers
  - AIF(Architecture Independent Format)
  - ‘Normalizing’ the data across platforms and languages

TO INFINITY AND BEYOND?

Exascale

- Probably big!
- Heterogeneous
- Mixed precision
- Hierarchical memories
- Algorithms
  - Loose synchronization
  - Fault tolerant



Debugging and Correctness

Scaling Debugging Techniques

Debugging Hybrid and Heterogeneous Architectures

Specialized Memory Systems

Domain Specific Languages

Mixed Precision Arithmetic

Adaptive Systems

Correctness Tools

Debugging and Correctness

Scaling Debugging Techniques	✔
Debugging Hybrid and Heterogeneous Architectures	✔
Specialized Memory Systems	✔
Domain Specific Languages	✔
Mixed Precision Arithmetic	✔
Adaptive Systems	?
Correctness Tools	?

✔ means some progress

Statistical Assertions

- Asserting descriptive statistics of a given dataset
  - Mean, standard deviation …
- Asserting statistical hypotheses
  - Distribution functions
  - Statistical tests
- Adjacent time steps show high data correlation
  - Can help identifying potential errors and outliers
- Asserting program states across time steps

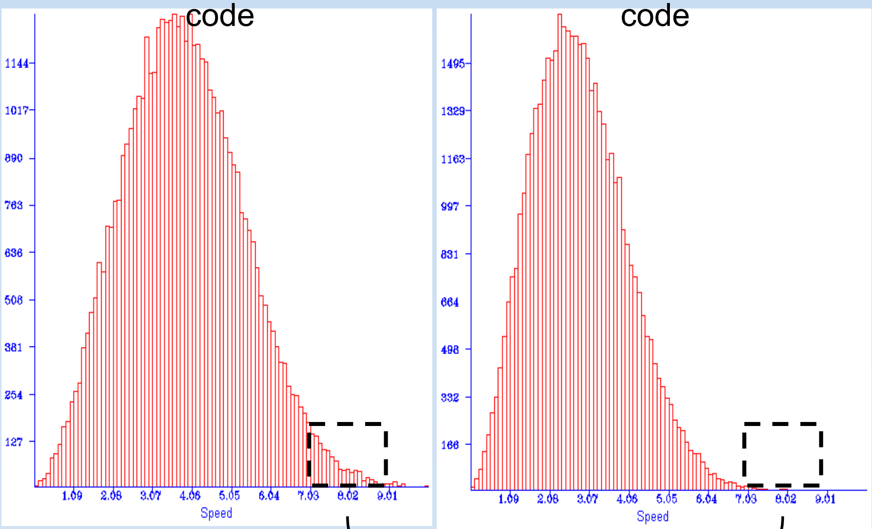
history etot $a::dvalue@“thermo.cpp”:1521 10 100

set reduce stdev; compare etot < 0.1

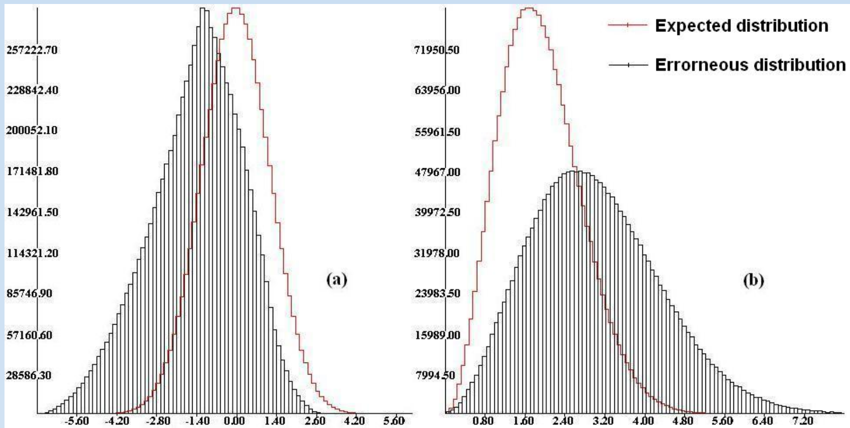
Statistical Assertions

- *Statistical parameters (mean, SD, etc)*
- *Statistical tests (T, χ^2, etc)*
- *Distributions*

Speed histogram for incorrect code   Speed histogram for correct code



Many high speed particles



Conclusion

- Comparative Debugging
  - Focuses on errors during code and platform evolution
    - Very rapid convergence
  - Large machines and programs are challenging
  - Techniques that scale to hundreds of thousands of cores
  - Commercial release from Cray