allinea

Leaders in parallel software development tools

Petascale Debugging via Allinea DDT for IBM Blue Gene /P and IBM Blue Gene /Q

Ian Lumb <ilumb@allinea.com>

Senior Systems Engineer, Allinea Software Inc.

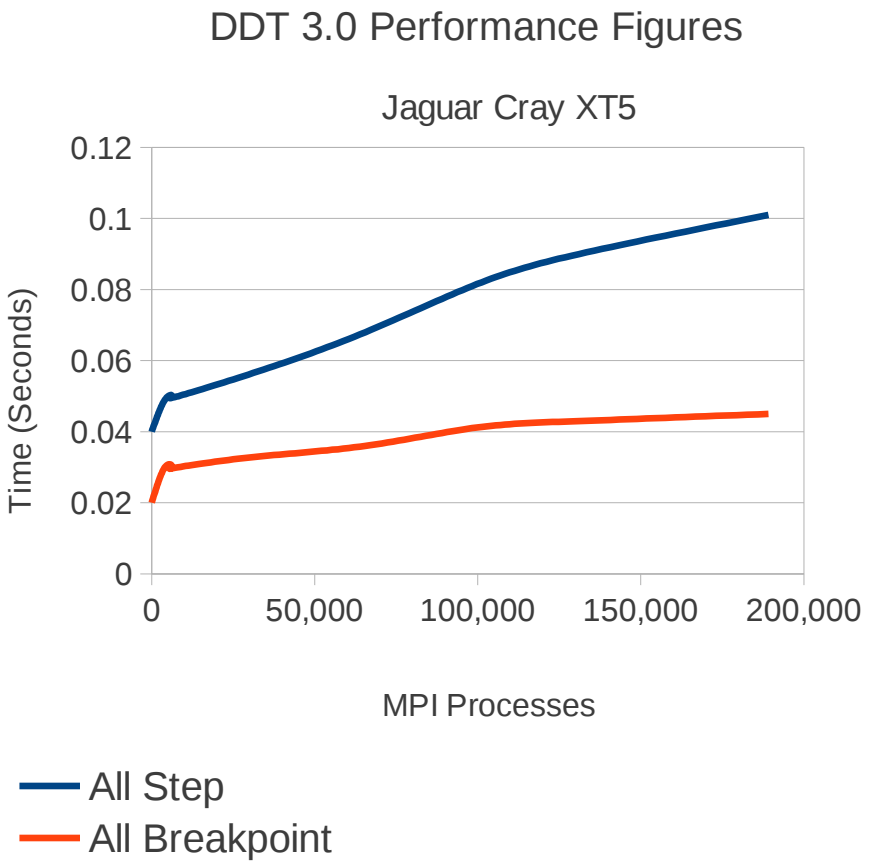
ALCF L2P Workshop, May 23, 2012

Outline

- Experience petascaling Allinea DDT
- Petascaling Allinea DDT for IBM Blue Gene */x*
- Getting Started with Allinea DDT

Petascaled Infrastructure

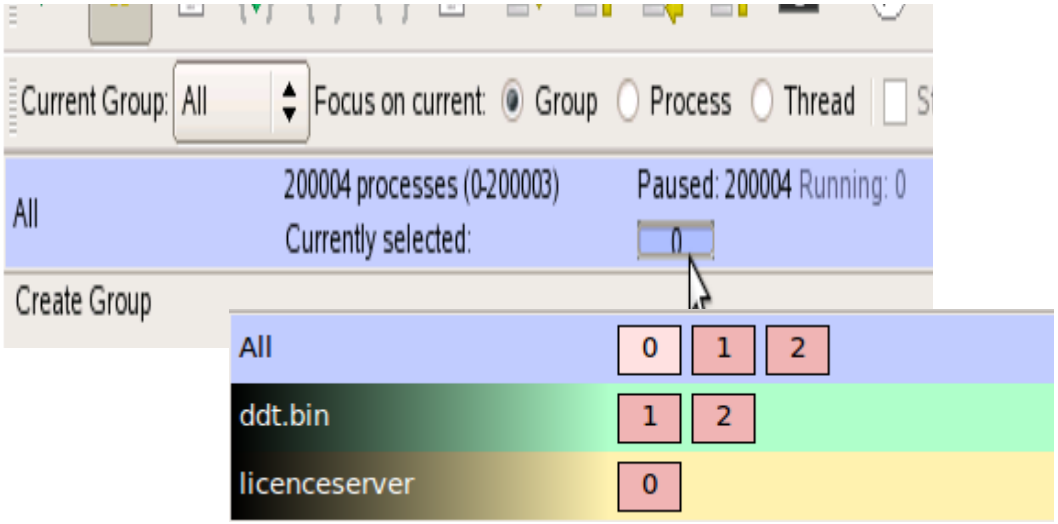
Allinea DDT 3.0 (April 2011)

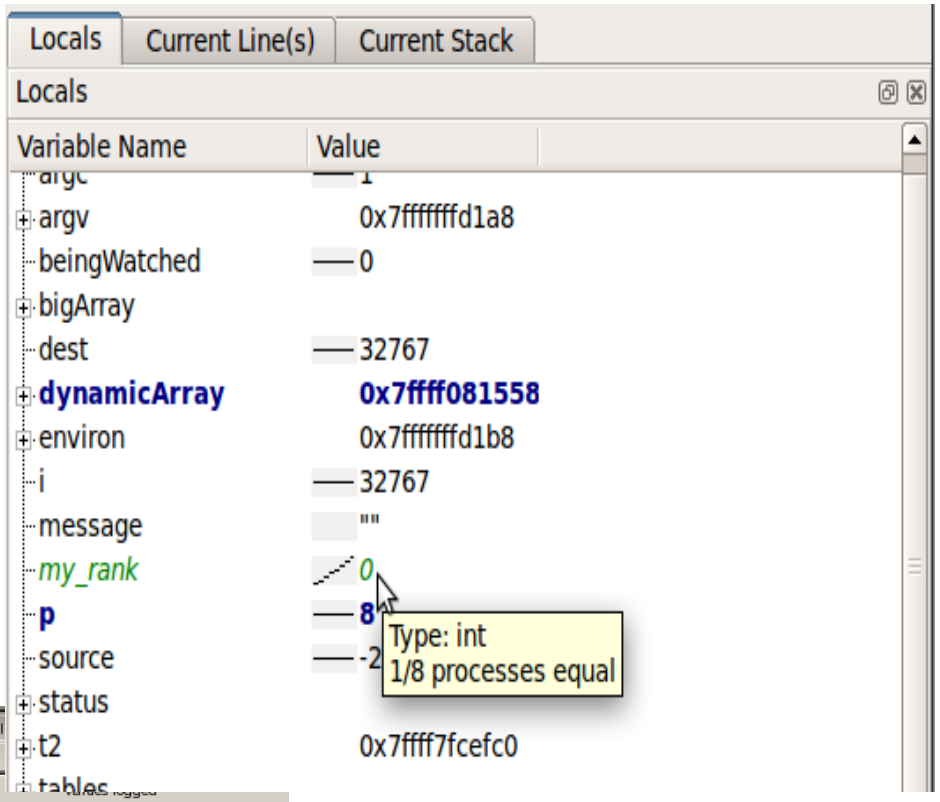


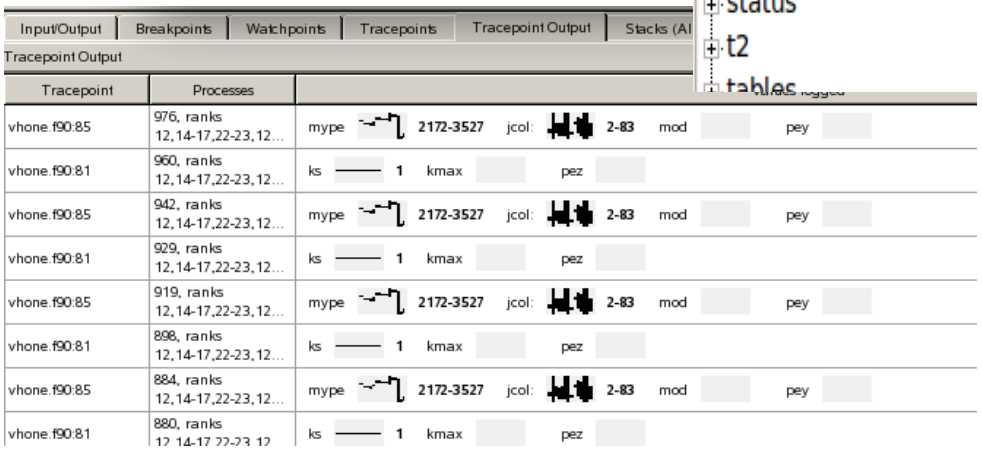
- Allinea DDT scales – logarithmically
- Tree network for communication
- Partnership with largest users
  - US DoE Oak Ridge National Laboratories
    - OLCF applications
    - Open MPI development
      - Routine debugging at 80K processes
- High performance petascale debugging
  - Even at 220,000 cores
- Step all and display stacks in ~1/10 second
- Scalable interface and features

Petascaled UI

Allinea DDT 3.1 (November 2011)







allinea
www.allinea.com

Debugging the IBM Blue Gene /P

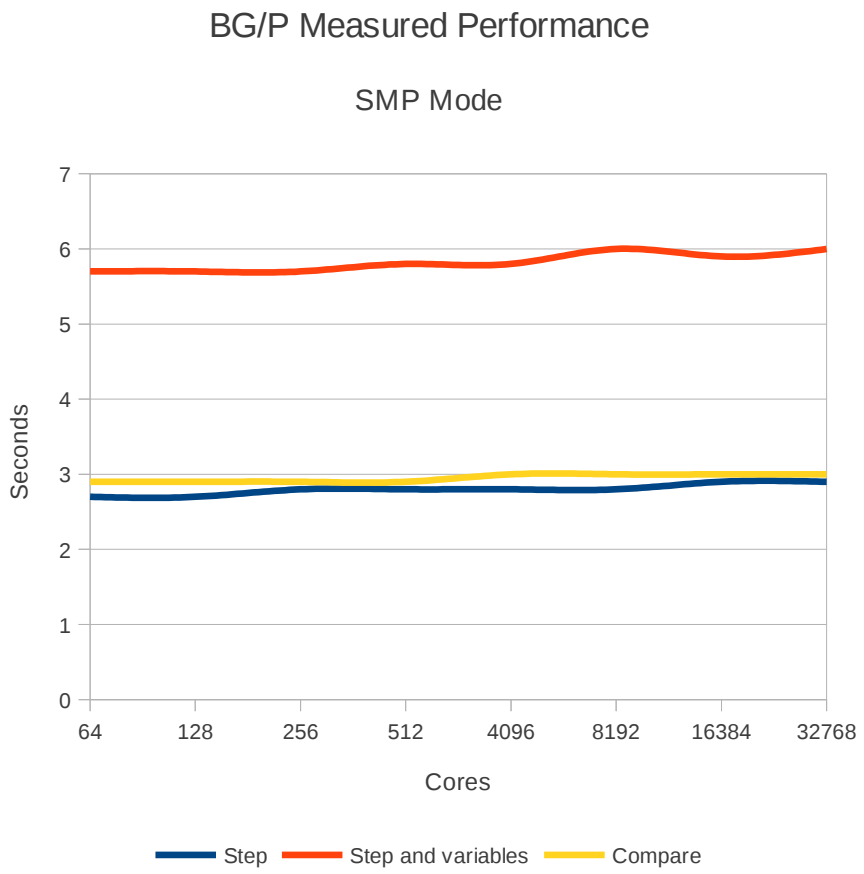
- The challenge
  - Lightweight OS on compute nodes
    - Only allows debug daemons at I/O nodes
    - Ratio of compute nodes to I/O cores: 64-512
  - Each I/O node is busy ...
    - Handles compute node debug work for each core
    - No tree to help here: not fast within one I/O node!
    - The bottleneck of IBM Blue Gene /P
- Multiple I/O nodes scale logarithmically

A Path to Petascale on IBM BG /P

- Phase 1 [2010]
  - Cut memory usage per compute process at I/O node
  - Debuggers share common internal tables
    - Memory mapping of symbol tables
    - Raises limit to ~128 processes
- Delivered!

The memory mapped result

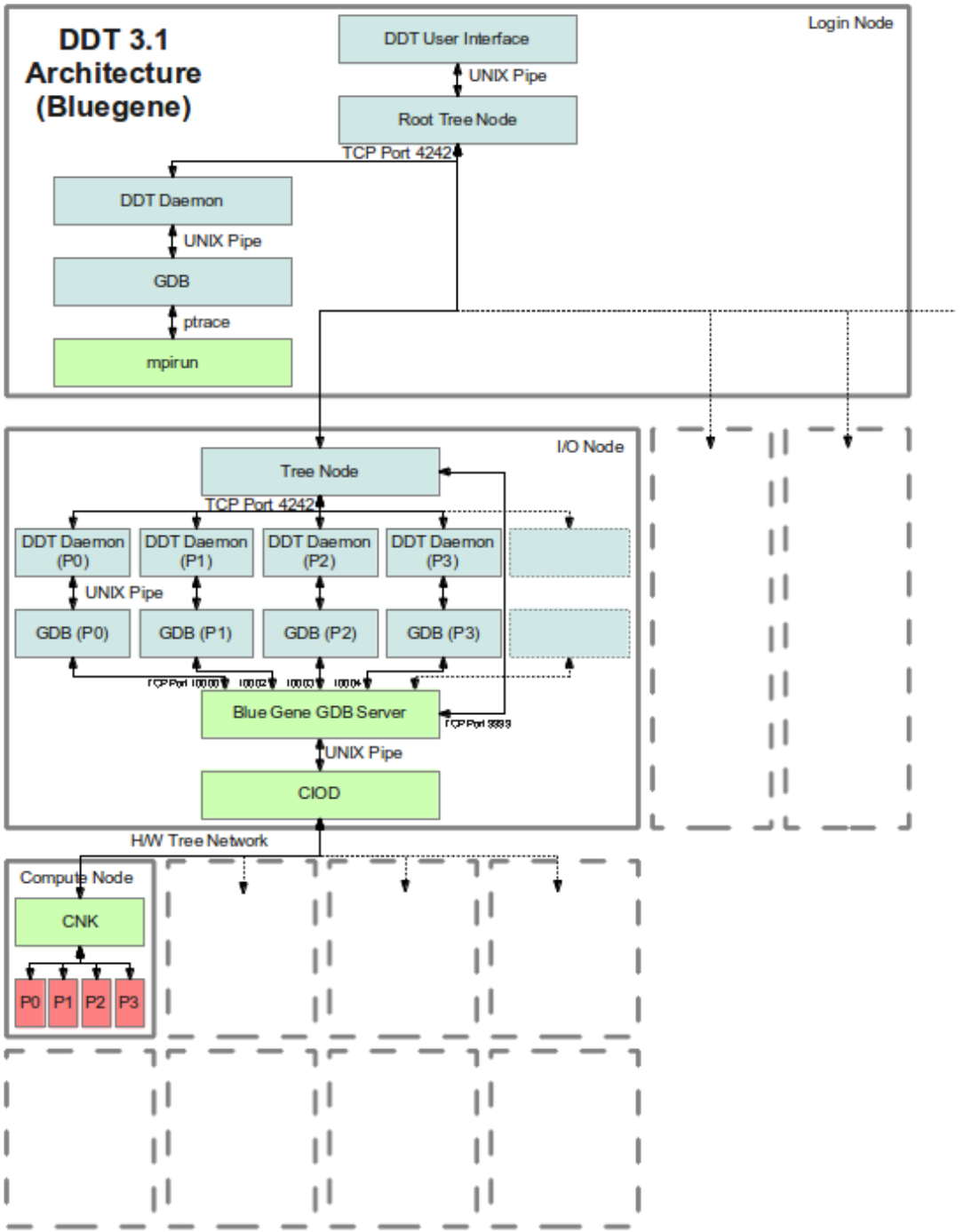
- Simplest to achieve – with benefits to multicore systems
  - Boosted max cores per I/O node to 256
- Reached 32K cores
  - 32,000 cores as quick as 64 cores
  - … flat – but not instantaneous
  - Most operations ~ 3 seconds
  - Close work with ANL – ran at scale on Intrepid



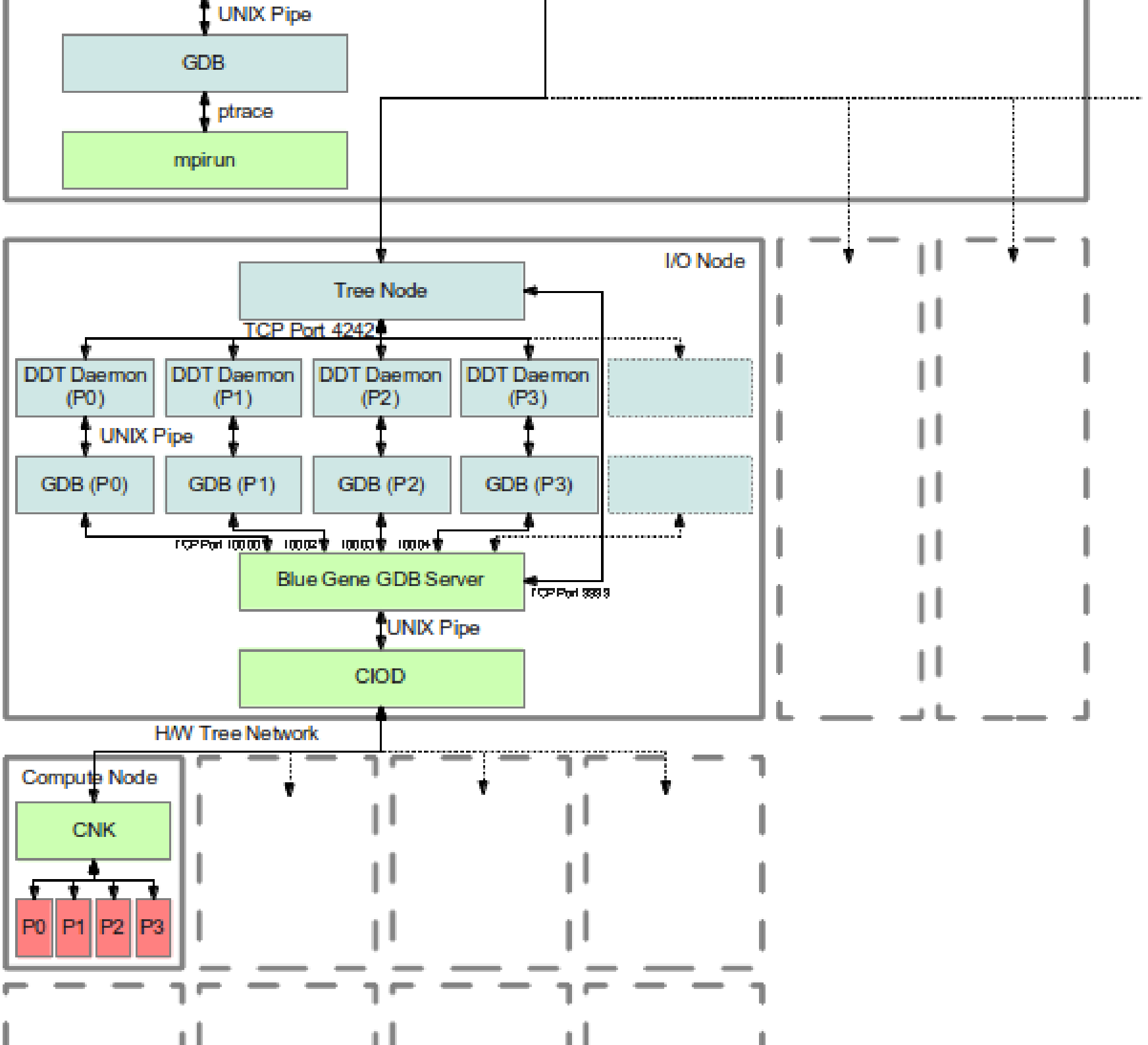
Petascale IBM Blue Gene /P Debugging

- Phase 2 [2011]
  - Reduce per-I/O-node daemon count
  - Reduces context thrashing: faster!
  - Each daemon handles multiple compute processes
    - Multiplexing commands and responses via CIOD
    - Multiplexing within the debugger
    - Cuts memory usage and improves speed
  - Limit 256-512 processes per I/O node
- Delivery: July 2012

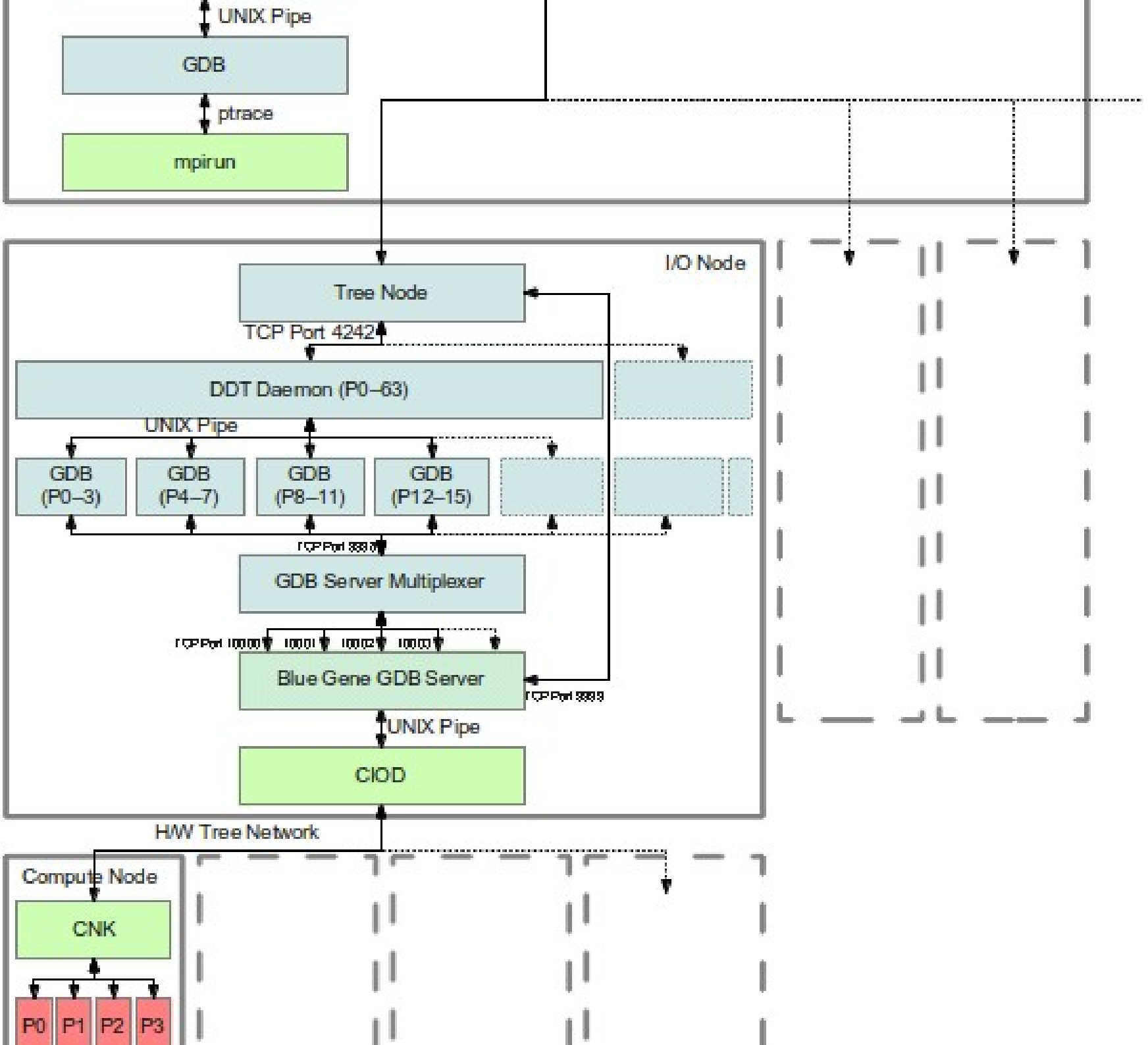
Original Architecture



Original Architecture



Multiplexed Architecture



Current Status

- IBM Blue Gene /P
  - Acceptance testing at ALCF (Allinea DDT 3.1)
    - Scale for Intrepid
      - Memory-mapped debugger data
      - Multiplexed debugger daemons
- IBM Blue Gene /Q
  - Under development (Allinea DDT 3.2)
    - Early access for IBM Blue Gene /Q expected July 2012
  - ALCF requirement
    - Scale for Mira

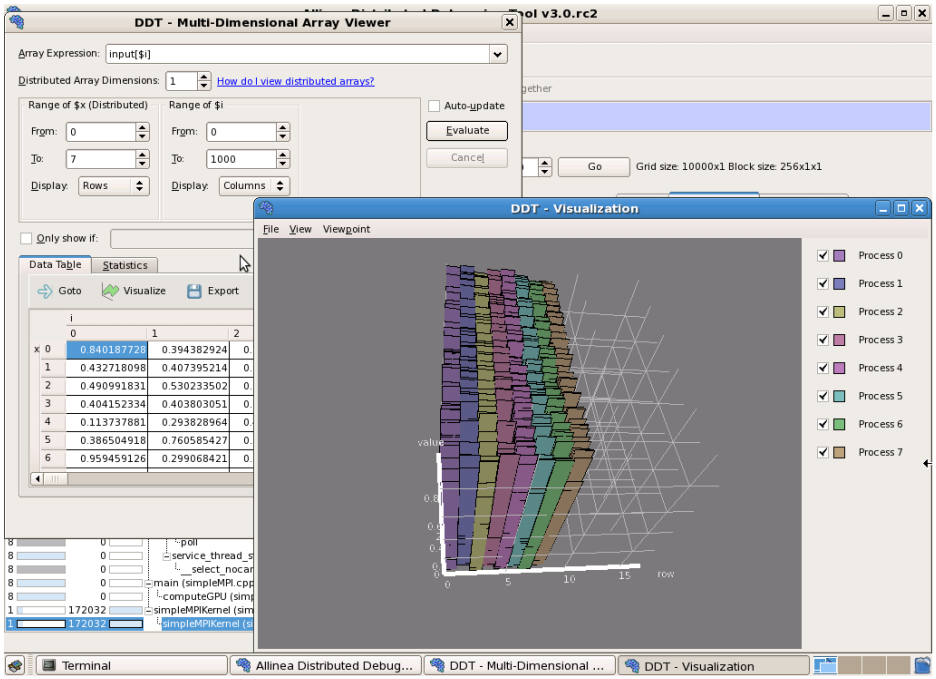
Allinea DDT

- BG/P licensing
  - 64-process permanent license
  - Full machine development license available (contact support)
- Startup overview
  - Compile –g –O0
    - OMP code compile -qsmp=omp:noauto:noopt
  - Softenv key "+ddt"
  - Need X11 server and ssh –X forwarding
  - Start interactive job with *isub*
  - Run ddt from isub
- More details:
  - http://www.alcf.anl.gov/resource-guides/debugging ("Using Allinea DDT")

http://www.alcf.anl.gov/resource-guides/allinea-ddt

Self-Paced Debugging Workshop

- Debugging use cases
  - Straightforward crashes
  - Memory errors and leaks
  - Deadlocks
  - Incorrect results
- Workshop-style approach
  - Detailed examples via annotated code



http://www.allinea.com/downloads/ddt_training.tar.gz

http://www.allinea.com/products/ddt-trial

allinea
www.allinea.com

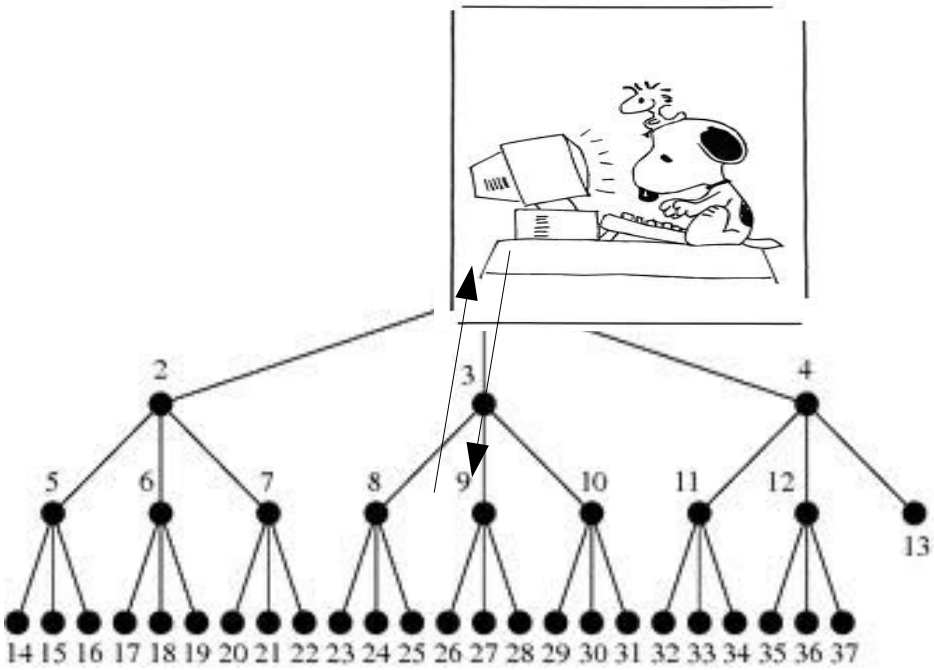
Summary

- Petascaling for > 1 year
  - Petascaled infrastructure and UI
- Scaling for IBM Blue Gene /P
  - Acceptance testing at ALCF
- Scaling for IBM Blue Gene /Q
  - Addressing ALCF requirements
    - Early access for IBM Blue Gene /Q expected July 2012
- Architecture applicable elsewhere
  - Multicore/GPU??? architectures
- Exascaling ...

Additional Slides ...

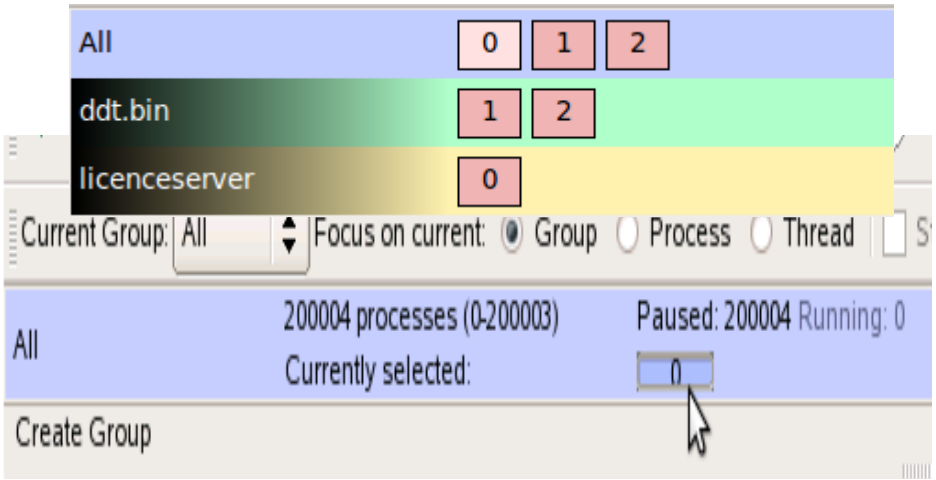
Petascaling Allinea DDT

- A control tree gives scalability
- Ability to send bulk commands and merge responses
  - 100,000 processes in a depth 3 tree
- Compact data type to represent sets of processes
  - eg. For message envelopes
  - An ordered tree of intervals, or a bitmap?
- Develop aggregations
  - Merge operations are key: not everything can/should merge losslessly
  - Maintain the essence of the information: eg. min, max, distribution



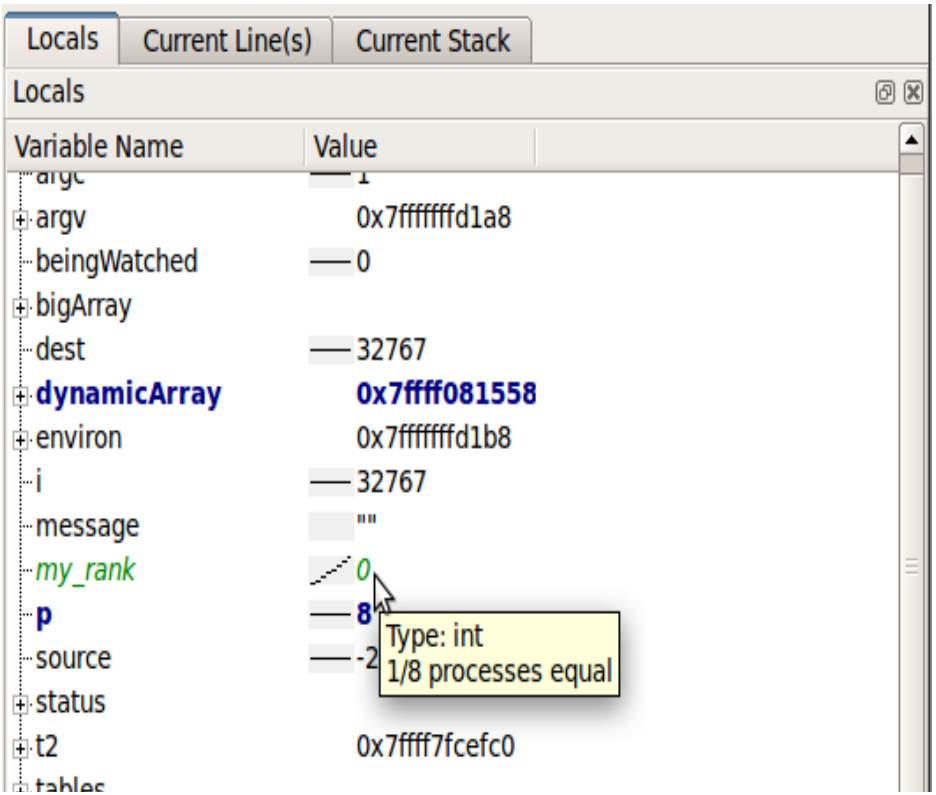
Process Control

- Interacting with application progress is easy with Allinea DDT
  - Step, breakpoint, play, or set data watchpoints based on groups
  - Change interleaving order by stepping/playing selectively
- Group creation is easy
  - Integrated throughout Allinea DDT - eg. stack and data views
- Common issues easily visible by seeing the outlier
  - Divergence of processes is clear in the Parallel Stack View

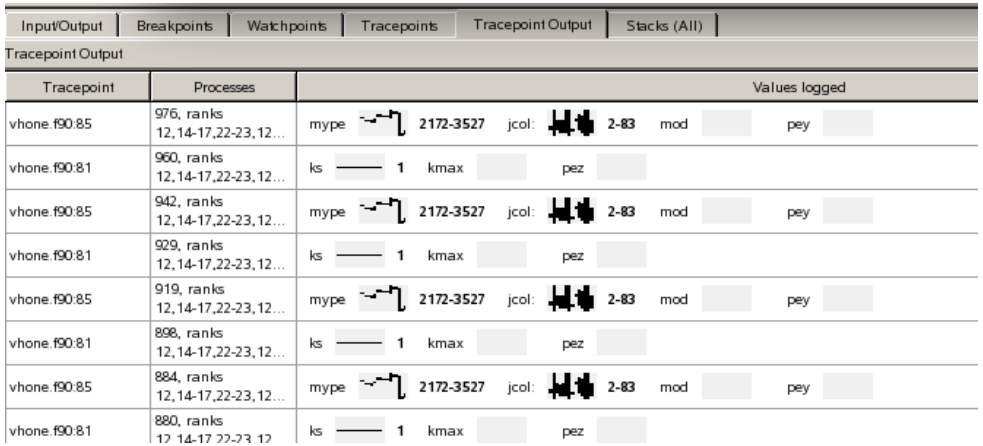


Sparklines (DDT 3.1, 11/2011)

- Clear need to see data
  - Too many variables to trawl manually
  - Allinea DDT compares data automatically
- Smart highlighting
  - Subtle hints for differences and changes
  - Colour and sparklines!
- More detailed analysis
  - Full cross process comparison
  - Historical values via tracepoints



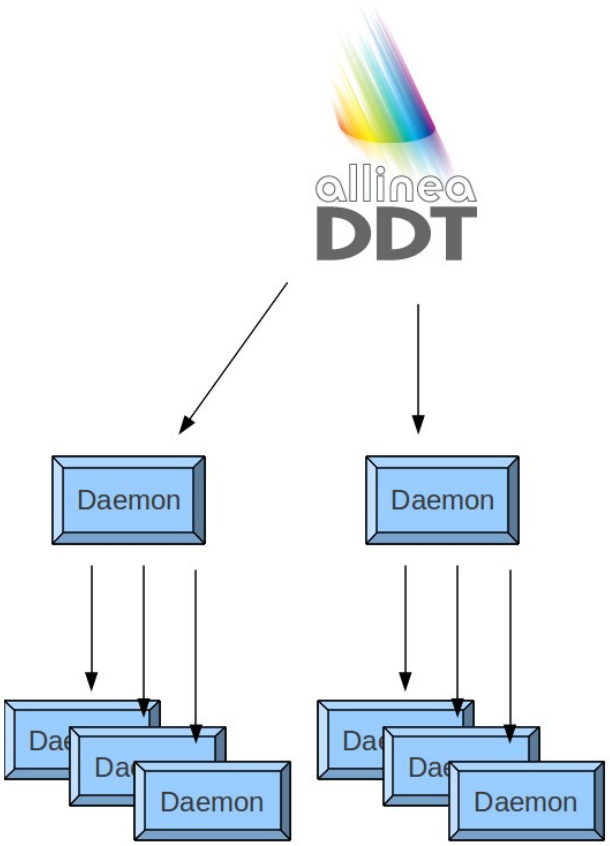
Tracepoints (DDT 3.1, 11/2011)



- A scalable print alternative
  - Merged print – with a sparkline graph showing distribution
  - Change at runtime – no recompilation required

Scaling for IBM Blue Gene /P ...

- Allinea DDT's architecture
  - Two daemons per MPI process: controller and a single process debugger
  - As close to process as possible: on the I/O node
  - Ideal for a full O/S
- But on the I/O node..
  - RAM per core low
  - Debugger cores per compute core low
- ***Must do less work, and do it for less memory***



How to use less memory?

- Debugging needs memory
  - Complex C++ generates biggest symbol tables
  - … but with 256 cores even 20MB per process is too much
  - Target is debugging multi-thousand core jobs on ANL ALCF facilities
- Ideas ...
  - *Use one debugger and `multiplex' the process*
    - *A good answer, but more work than necessary*
  - *Load symbol table once and fork other debuggers from it …*
    - *Wouldn't work for many cases – particularly shared libraries*
  - **... memory mapped read-only internal debugger data file**
    - Sounded plausible!
    - Idea used before in GDB but suffered bit-rot

Where next?

- Future ratios need more work
  - IBM Blue Gene /Q is a big step
  - Compute-core to I/O node memory ratio shooting up
  - A real hardware bottleneck – just when we cured the software one
- What technologies are right?
  - Multiplexing all daemons
  - Good – but still lot of CPU load at the IO node
- Do more at compute node
  - Real O/S (please!) or in-process debugging/off-load
  - More opportunity if we had more speed: Potential to do *anything*