Runtime and Third-Party Improvements

Chapel Team, Cray Inc.

Chapel version 1.13

April 7, 2016

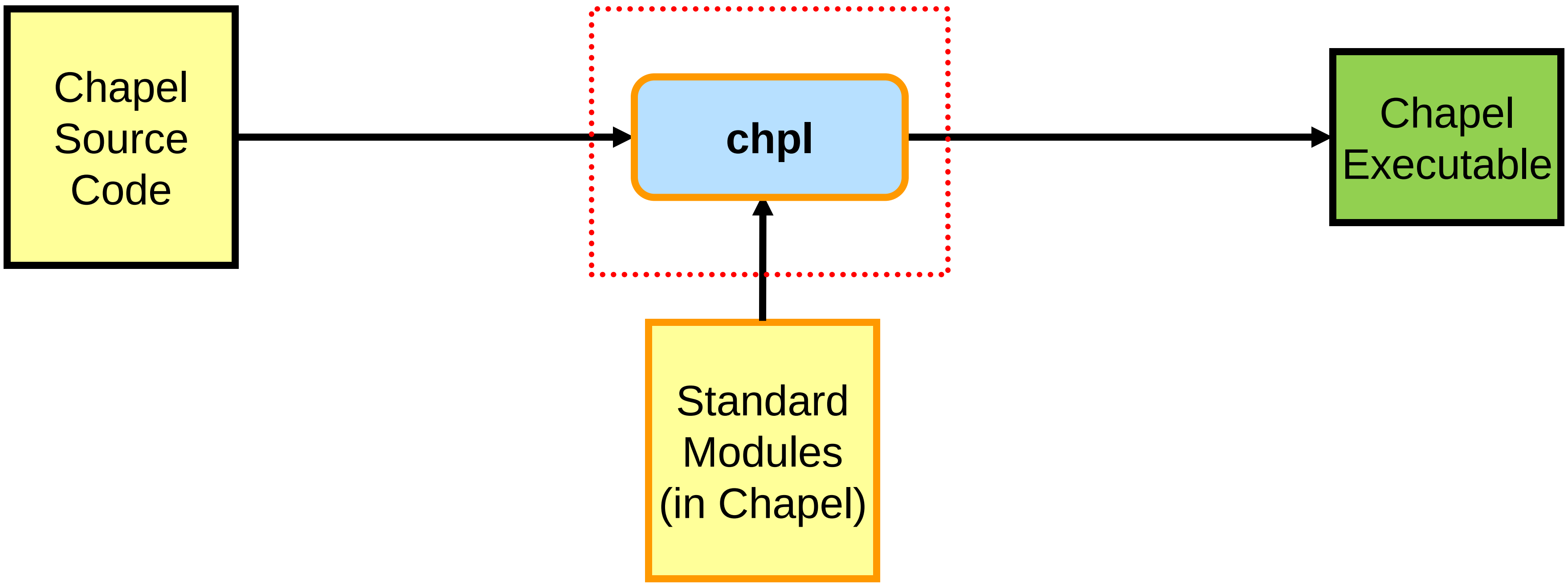
Safe Harbor Statement

This presentation may contain forward-looking statements that are based on our current expectations. Forward looking statements may include statements about our financial guidance and expected operating results, our opportunities and future potential, our product development and new product introduction plans, our ability to expand and penetrate our addressable markets and other statements that are not historical facts. These statements are only predictions and actual results may materially vary from those projected. Please refer to Cray's documents filed with the SEC from time to time concerning factors that could affect the Company and these forward-looking statements.

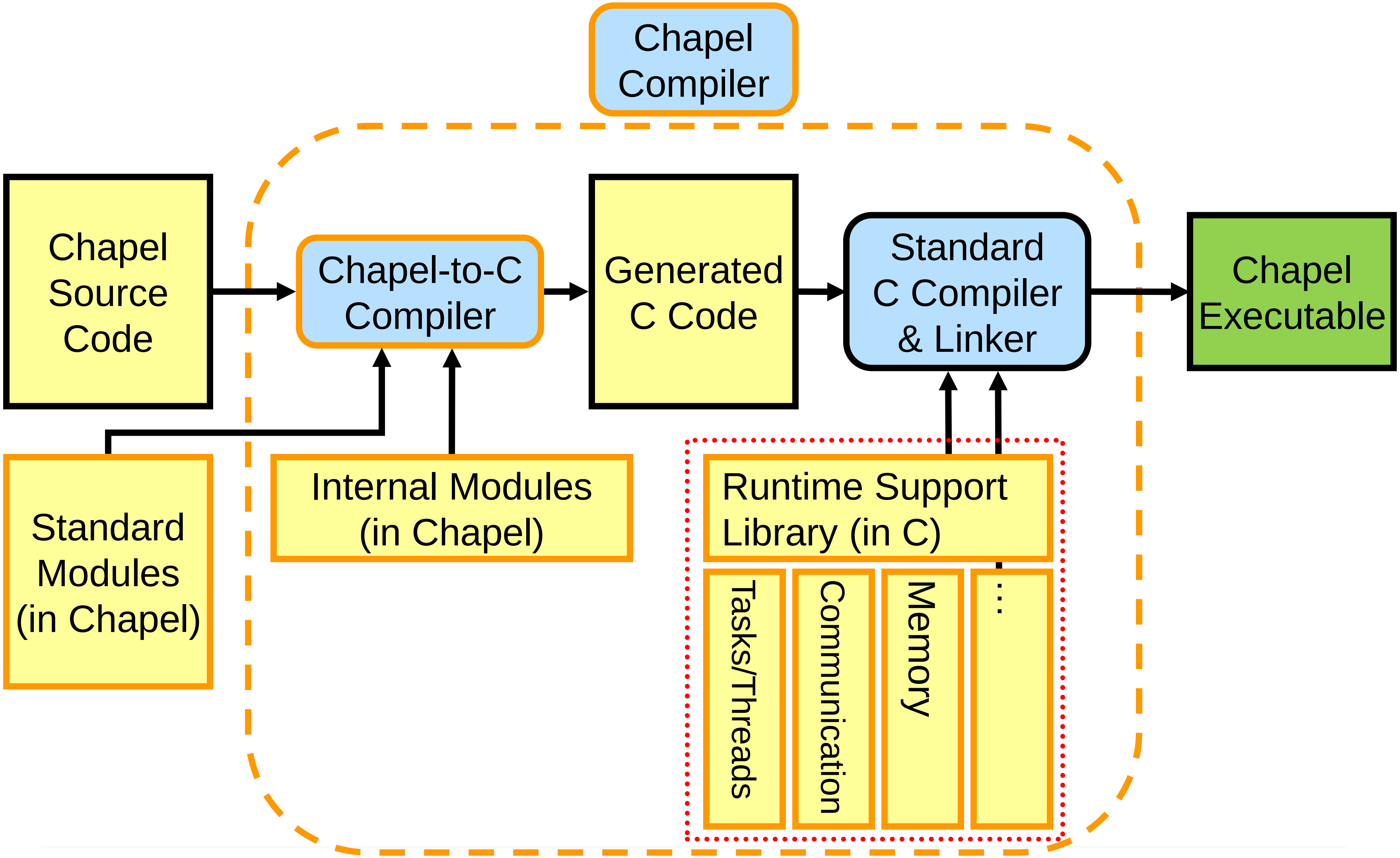
Outline

Runtime and Memory Layer Overview

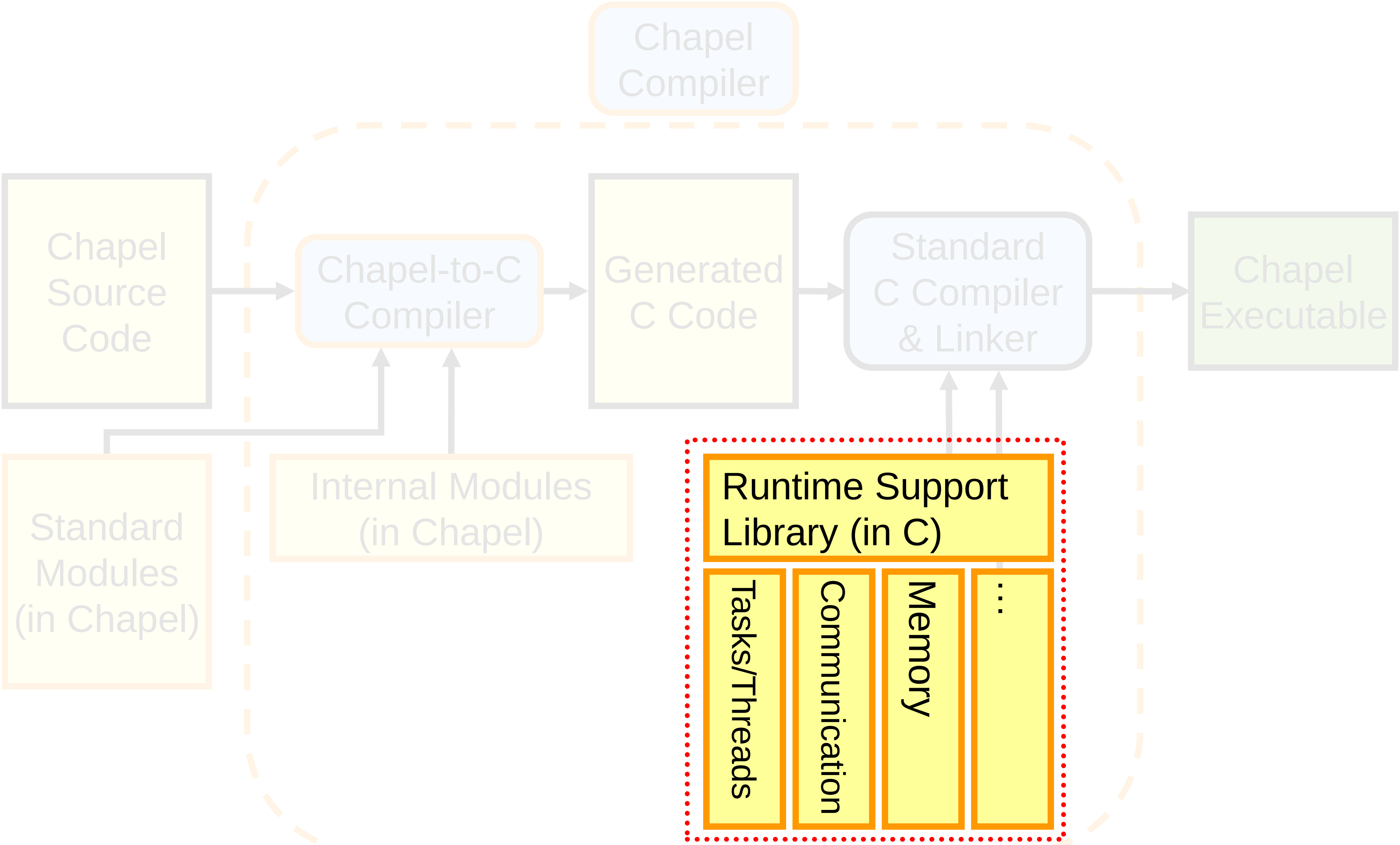
Compiling Chapel



Chapel Compilation Architecture



Chapel Compilation Architecture



Chapel Runtime

- **Lowest level of Chapel software stack**
- **Supports language concepts and program activities**
- **Relies on system and third-party services**
- **Written in C**
- **Composed of *layers***
  - A misnomer – these are not layers in the sense of being stacked
  - More like *posts*, in that they work together to support a shared load
  - Standardized interfaces
  - Interchangeable implementations
- **Env. vars. select configuration when building the runtime**
  - Also select between configurations when compiling a Chapel program
    - (must have already been built ahead of time)

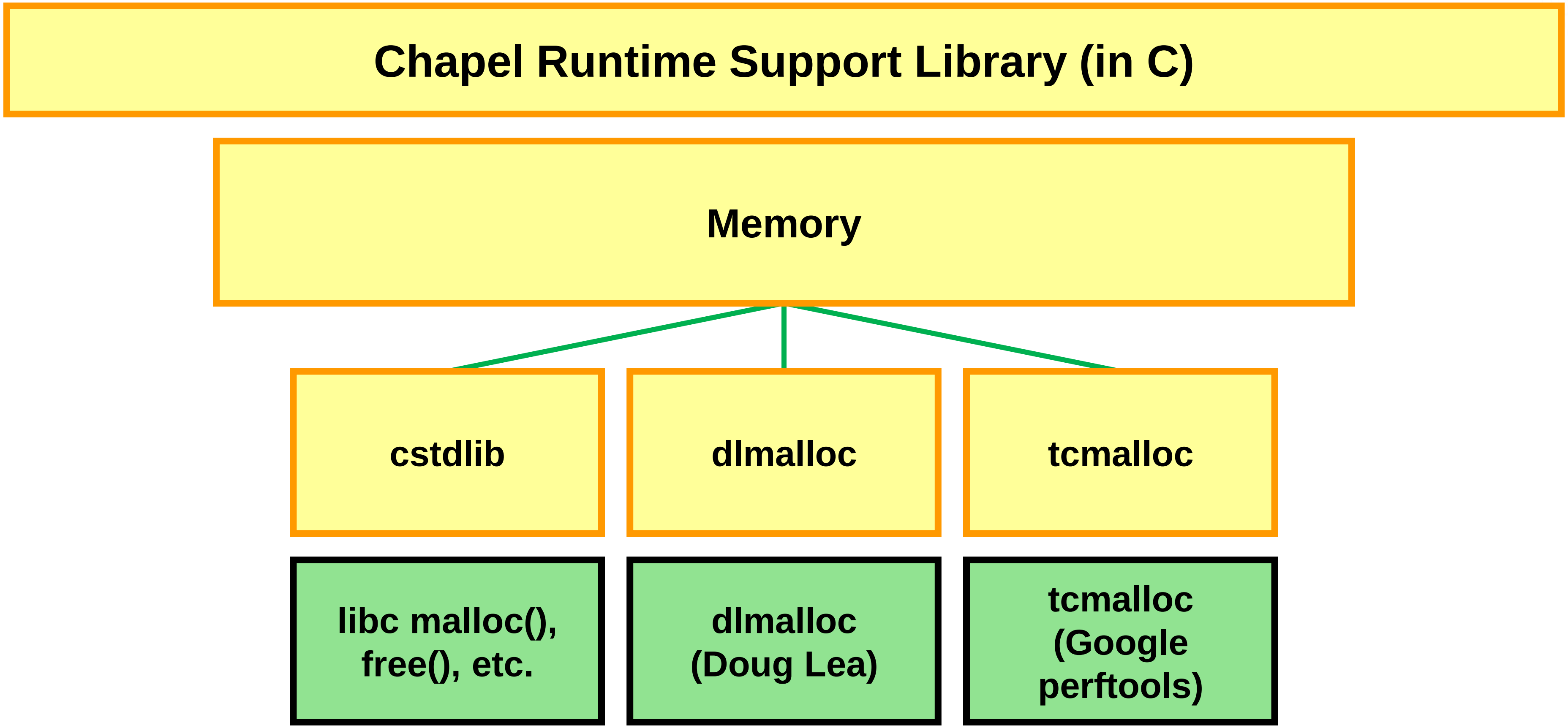
Chapel Runtime Organization

Chapel Runtime Support Library (in C)

Commu-nication	Tasking	Memory	Launch-ers	QIO	Timers	Standard

Standard and third-party libraries

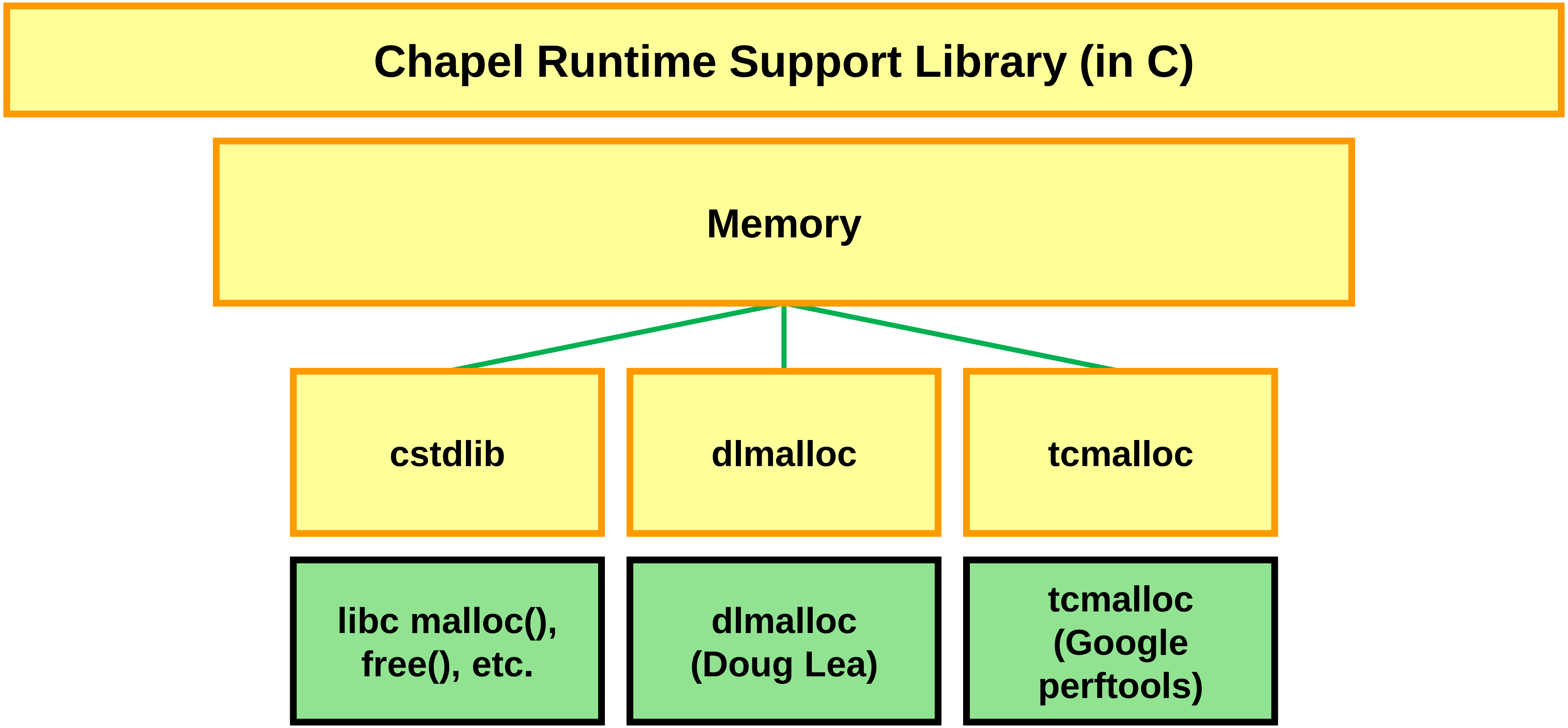
Runtime Memory Layer



Runtime Memory Layer

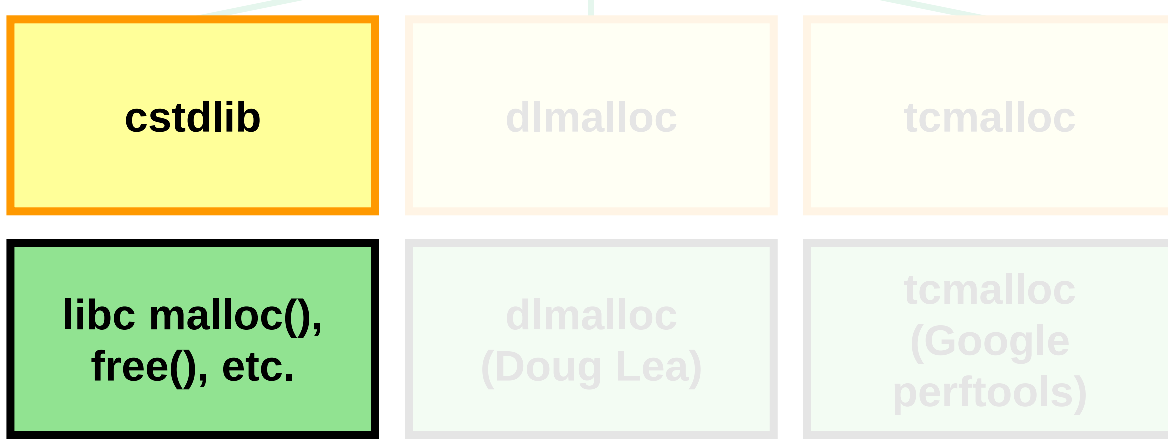
- **Supports dynamic memory management**
  - memory allocation, deallocation, and alignment
- **Ideally supports allocating out of a given memory region**
  - ugni and gasnet sometimes require a shared heap
  - in such cases, the communication layer:
    - grabs memory from the system
    - registers it with the NIC
    - hands it over to the memory layer
  - and the memory layer:
    - must fulfill allocations from that memory segment (instead of grabbing memory from the system)

Runtime Memory Layer



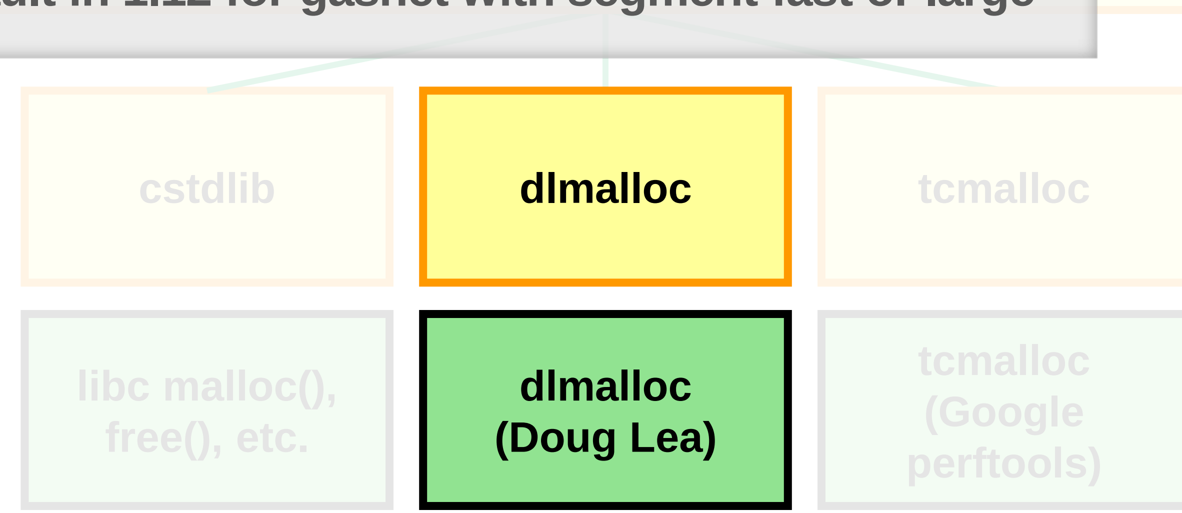
Runtime Memory Layer: cstdlib

- **System allocator:**
  - portable
  - relatively poor multi-threaded performance
  - no shared-heap support
- **the default in 1.12 when a shared heap was not required**



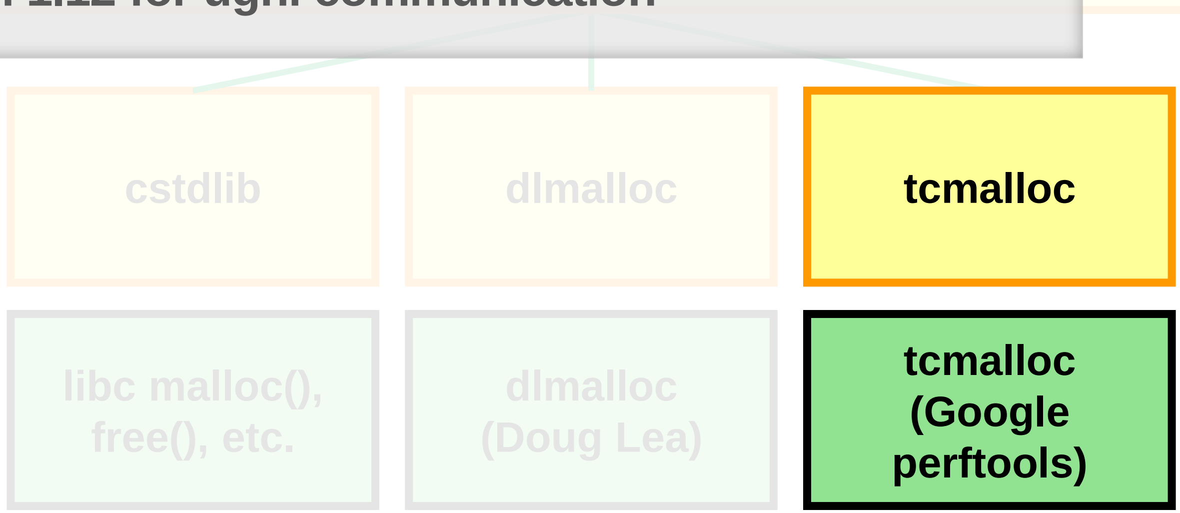
Runtime Memory Layer: dlmalloc

- **Doug Lea's dlmalloc allocator:**
  - relatively portable
  - poor multi-threaded performance
  - shared-heap support
- **the default in 1.12 for gasnet with segment fast or large**



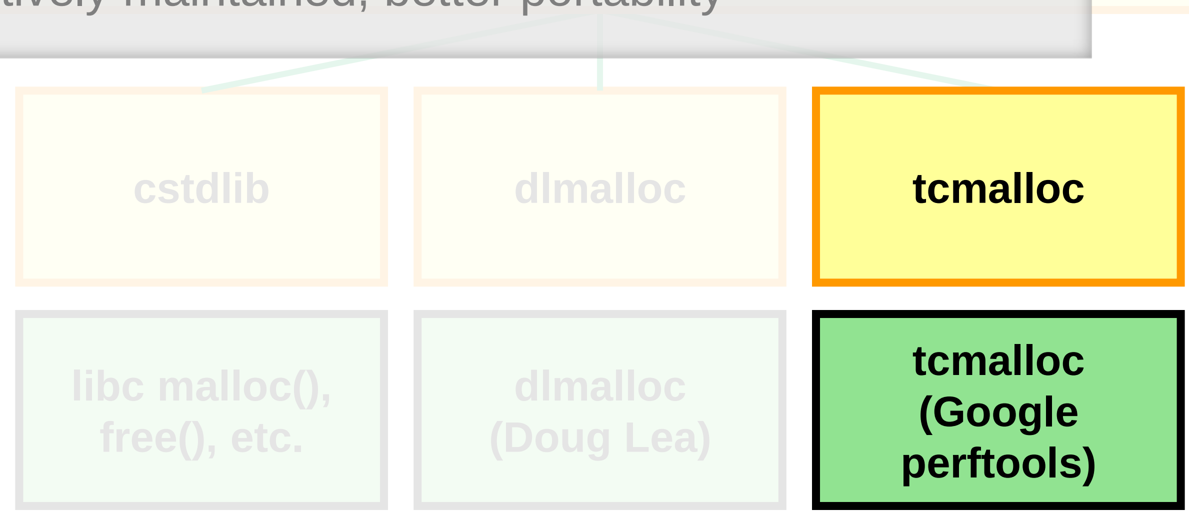
Runtime Memory Layer: tcmalloc

- **Google Perftool's tcmalloc allocator:**
  - not as portable as we'd like (gnu, clang, intel only)
  - excellent multi-threaded performance
  - shared-heap support
- **default in 1.12 for ugni communication**



Runtime Memory Layer: tcmalloc

- **Tried to make tcmalloc our preferred allocator for 1.10**
  - experienced sporadic segfaults with Qthreads
- **Decided to pursue jemalloc instead**
  - reputed to have good multi-threaded performance
  - actively maintained, better portability



Jemalloc Memory Layer

Jemalloc: Overview

- **General purpose malloc implementation**
  - “scalable concurrency support”
  - “emphasizes fragmentation avoidance”
  - also supports an extended API
    - good alloc size, sized deallocation, etc.
- **Actively maintained on GitHub**
  - https://github.com/jemalloc/jemalloc
- **Large number of notable users**
  - FreeBSD and NetBSD
  - Mozilla Firefox
  - Facebook
  - Android
  - Rust

Jemalloc: This Effort

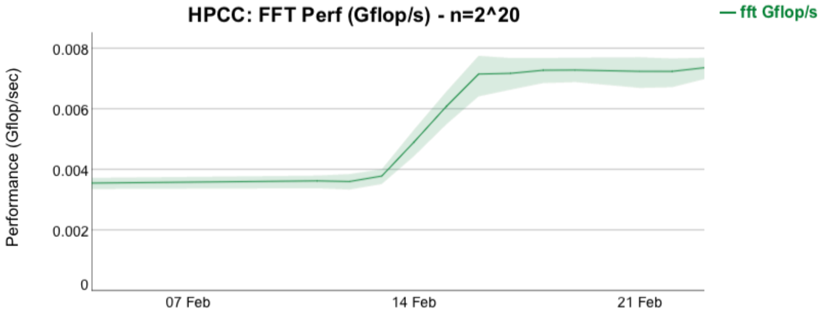
- **Added support for jemalloc**
  - initially just for configurations that did not require a shared heap
  - saw good performance, decided to add shared-heap support
    - using jemalloc's custom chunk allocator interface
  - implemented good_alloc_size() using extended API

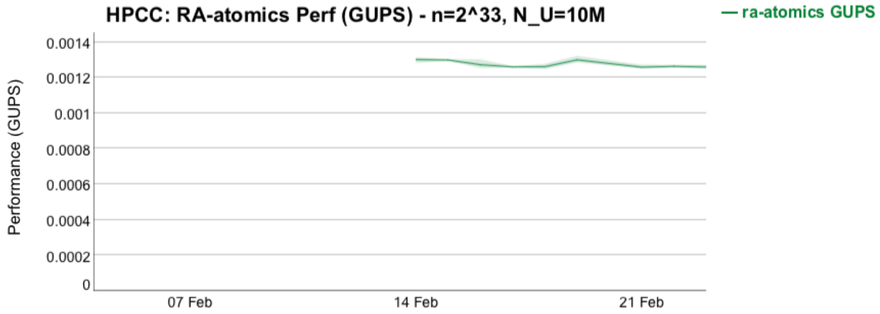
- **Switched our default allocator to jemalloc**
  - except for a few configurations:
    - gnu+darwin (build issues)
    - cygwin (build issues)
    - pgi (segfaults at execution time)

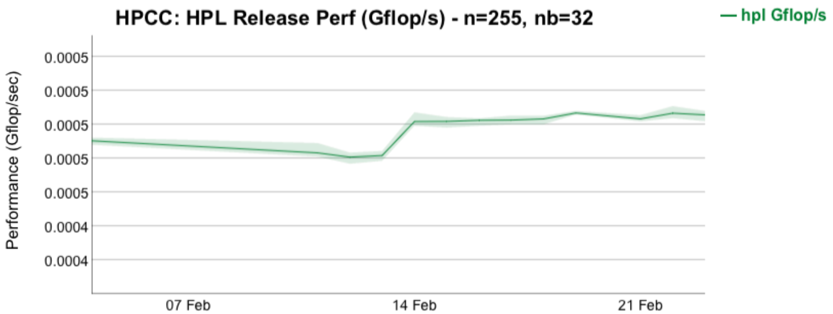
- **Removed dlmalloc and tcmalloc support**
  - jemalloc performs as well as tcmalloc and is as portable as dlmalloc

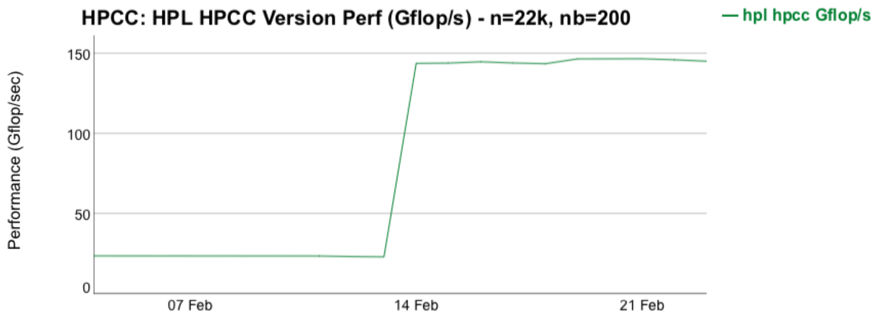
Jemalloc: Impact

- **Excellent multi-locale performance**
  - No performance regressions from tcmalloc to jemalloc switch
    - this is great news, as tcmalloc provided excellent performance
  - Substantial performance improvements from dlmalloc to jemalloc switch:



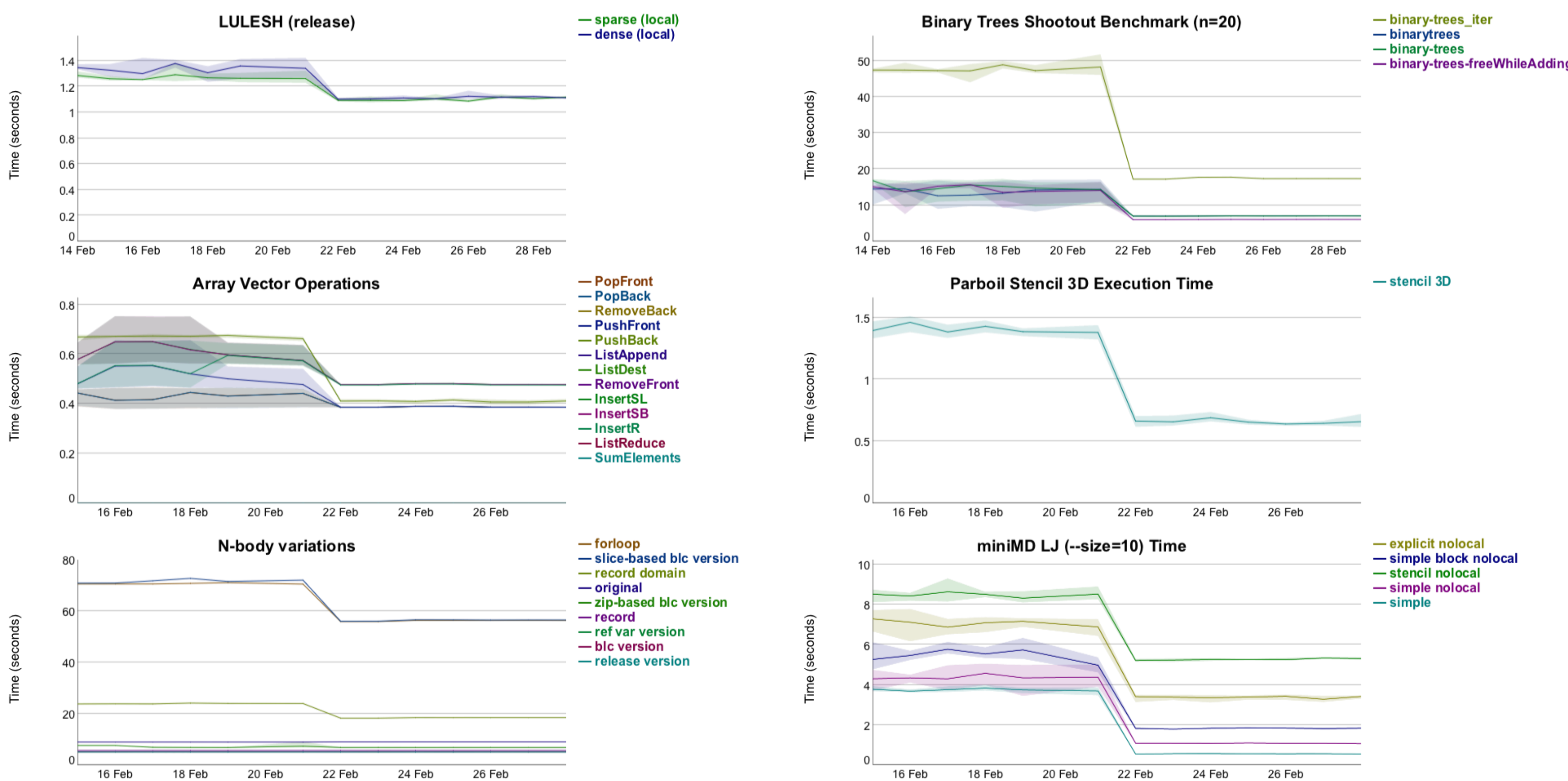






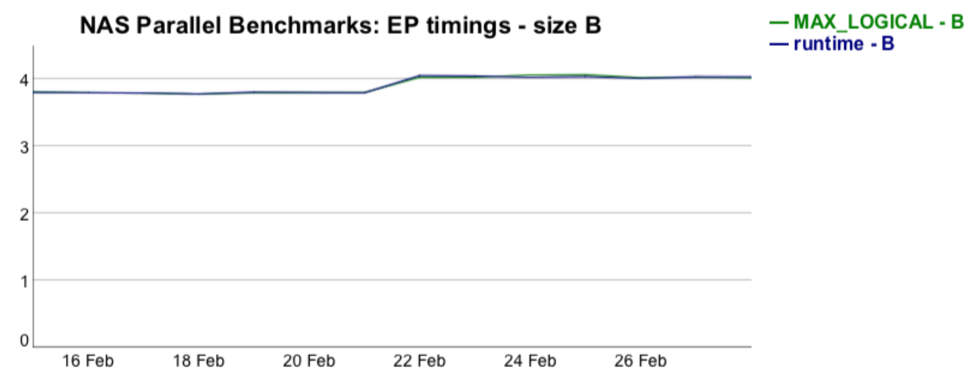
Jemalloc: Impact (continued)

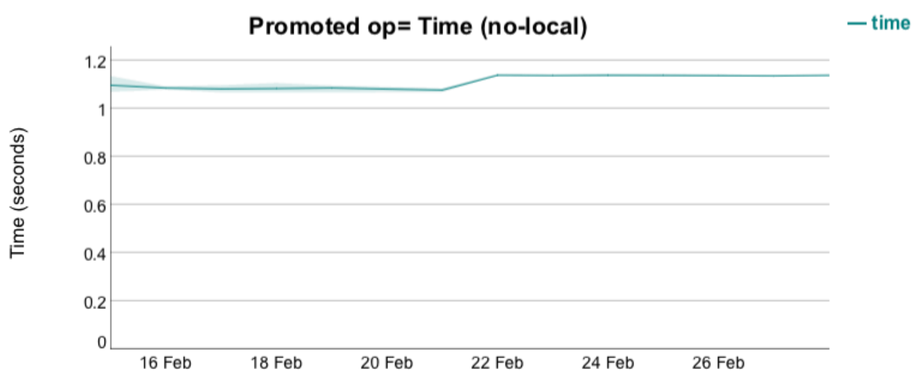
- **Excellent single-locale performance**
  - Substantial speedups from cstdlib to jemalloc switch:



Jemalloc: Impact (continued)

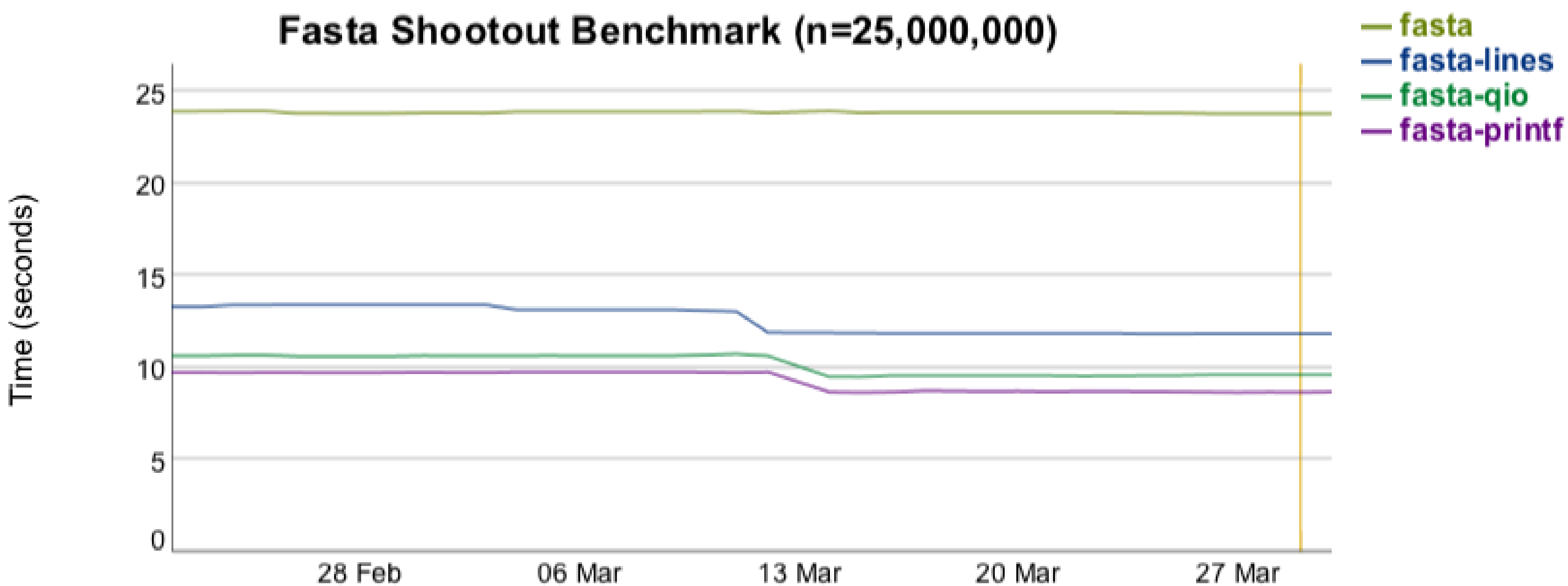
- **A few minor single-locale performance regressions**
  - only impacted tests that we aren't very invested in
    - i.e., benchmarks we haven't reviewed/optimized in-depth





Jemalloc: Impact (continued)

- **Implementing good_alloc_size() also improved perf.**
  - good_alloc_size() lets you reduce slack for strings, vectors, etc.
    - see Facebook's vector documentation for more information
    - we currently use good_alloc_size() for strings



Jemalloc: Summary and Next Steps

Summary:

- Added support for jemalloc
- Switched to jemalloc as our default allocator for most configurations
  - overall result was a dramatic performance improvement
  - likely resulted in much less memory fragmentation as well

Next Steps:

- Look into fixing support for darwin+gnu, cygwin, and pgi
- Use more of the extended API (sized_deallocation(), etc.)
- Use good_alloc_size() for array-as-vector
- Consider overriding malloc to call into jemalloc
  - possible performance benefits for third-party code that calls malloc()

ugni Communication Layer Improvements

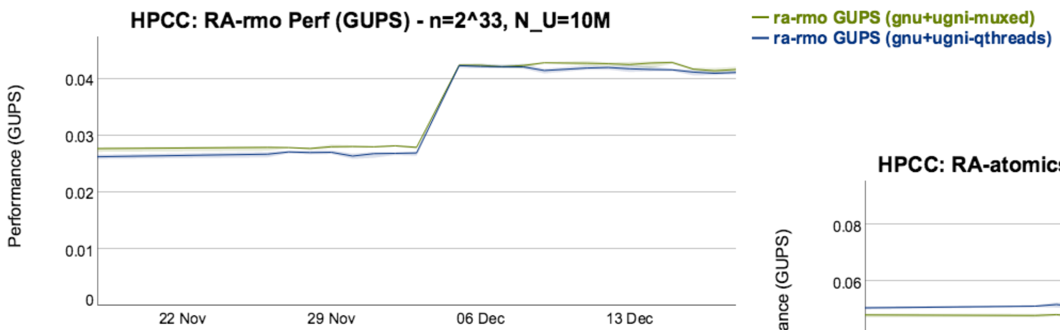
ugni Improvements: Background

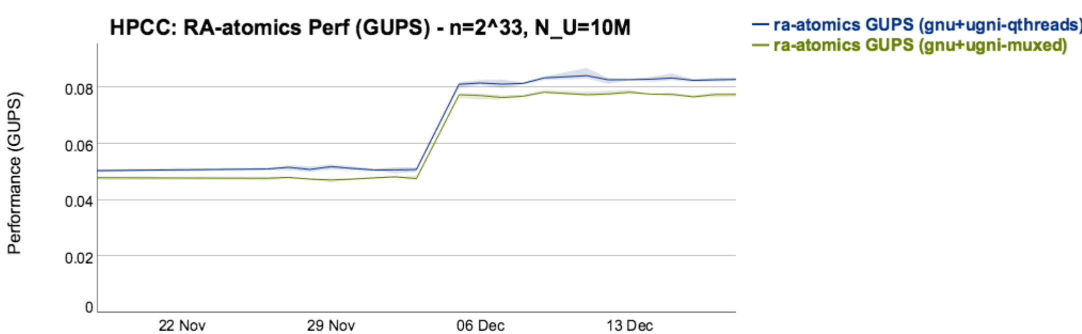
- **GNI memory registration wrecks NUMA affinity**
  - NIC needs virtual/physical mapping, so registration pins memory
    - NIC accepts virtual addrs from user code but references physical mem
  - GNI prefers pinning to NUMA domain 0 because it's closer to the NIC
  - Thus all or most memory ends up there; user first-touch has no effect
  - Touch-before-registering forces locality, but we can't leverage that yet
- **Workaround: don't register user memory**
  - Permit user first-touch to have its usual effect
- **Register minimal internal memory**
  - GNI RDMA communication requires registration
  - Includes management data, trampoline buffers, etc.
  - Implies: reworking many internal data structures
    - these were previously allocated from the (registered) heap
- ***This is a stopgap, not a principled solution***

ugni Improvements: Status and Impact (1)

- **Reworked ugni communication layer internals**
  - Streamlined resource management
    - GNI RDMA comm. domains (represent Gemini/Aries FMA descriptors)
  - Reorganized internal data structures and procedures
    - request buffers and completion handling for remote on-stmts
    - completion handling for nonblocking GET/PUT

- **Result: improved remote memory access performance**





ugni Improvements: Status and Impact (2)

- **Implemented minimal registered memory**
  - Register communicable internal data structures
  - Register trampoline buffers for communicating user data
  - Don’t register anything else
  - Add logic to bounce transfers through trampolines, etc.

- **Results without hugepages:**
  - Much-improved NUMA-related performance
    - Cray XC multi-node: stream-ep ~2X faster than with hugepages
    - on par with HPCC MPI+OpenMP reference version
  - Hurt comm-intensive benchmark performance in some cases
    - sometimes dramatic: have seen ~50X slower for RA-rmo
  - Beneficial side effect: without hugepages, ‘Spawn’ works with ugni
    - GNI kernel driver has issues duplicating registered memory in fork()

ugni Improvements: Summary and Next Steps

Summary:

- Support for minimal registered memory is complete
- Related work greatly improved remote memory access performance
- Users can optionally run without hugepages
  - remember: this is only a stopgap, or for specific use cases
- Principled solution for performance is to use the numa locale model
  - first-touch currently depends on luck: how tasks are assigned to threads
  - numa locale model won't depend on first-touch
  - numa locale model improvements didn't make it into 1.13 release
    - (see "Ongoing Efforts" deck for more information)

Next Steps:

- Investigate performance surprises due to minimal registered memory

Soft-threading Patent Granted

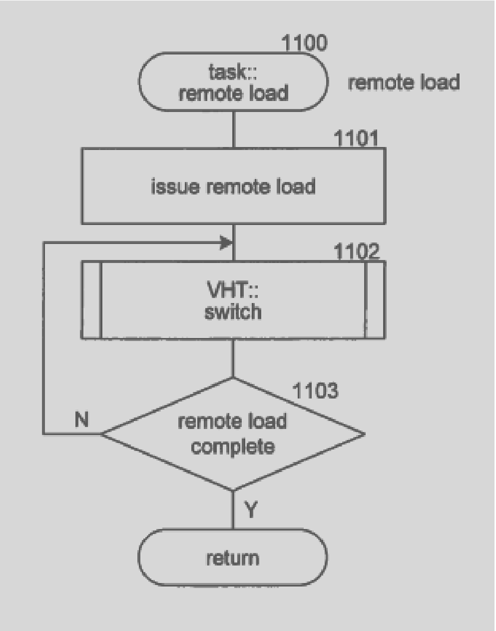
The Soft-threading Patent Has Been Granted

Background: Intellectual property in Cray runtime layers

- Enables fine-grained compute/communicate overlap in ugni+muxed

This Effort: [Patent 9,201,689 at uspto.gov]

Software emulation of massive hardware threading for tolerating remote memory references



Impact: Cray runtime layer IP no longer secret

Next Steps: Lightweight comp & comm overlap in other layers?

- Need to evaluate effectiveness of lightweight switching in Qthreads
- If effective, may be able to retire muxed tasking

Other Third-Party Changes

Other Third-Party Changes

- **Turned on optimizations when compiling 're2' optimized**
- **Enabled 're2' for Cray and PGI compilers**
- **Upgraded GASNet to version 1.26.0**
- **Upgraded hwloc to version 1.11.2**
  - also cherry-picked a /proc/mounts buffer overflow fix from their master
- **Upgraded LLVM to version 3.7**
- **Made several third-party directories more Git-friendly**
  - stopped embedding version numbers in subdirectory names

Legal Disclaimer

Information in this document is provided in connection with Cray Inc. products. No license, express or implied, to any intellectual property rights is granted by this document.

Cray Inc. may make changes to specifications and product descriptions at any time, without notice.

All products, dates and figures specified are preliminary based on current expectations, and are subject to change without notice.

Cray hardware and software products may contain design defects or errors known as errata, which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Cray uses codenames internally to identify products that are in development and not yet publically announced for release. Customers and other third parties are not authorized by Cray Inc. to use codenames in advertising, promotion or marketing and any use of Cray Inc. internal codenames is at the sole risk of the user.

Performance tests and ratings are measured using specific systems and/or components and reflect the approximate performance of Cray Inc. products as measured by those tests. Any difference in system hardware or software design or configuration may affect actual performance.

The following are trademarks of Cray Inc. and are registered in the United States and other countries: CRAY and design, SONEXION, and URIKA. The following are trademarks of Cray Inc.: ACE, APPRENTICE2, CHAPEL, CLUSTER CONNECT, CRAYPAT, CRAYPORT, ECOPHLEX, LIBSCI, NODEKARE, THREADSTORM. The following system family marks, and associated model number marks, are trademarks of Cray Inc.: CS, CX, XC, XE, XK, XMT, and XT. The registered trademark LINUX is used pursuant to a sublicense from LMI, the exclusive licensee of Linus Torvalds, owner of the mark on a worldwide basis. Other trademarks used in this document are the property of their respective owners.

THE SUPERCOMPUTER COMPANY