
CRAY
Standard Library Improvements
Chapel Team, Cray Inc.
Chapel version 1.10
October 2nd, 2014
CRAY
CHAPEL
COMPUTE | STORE | ANALYZE

Safe Harbor Statement

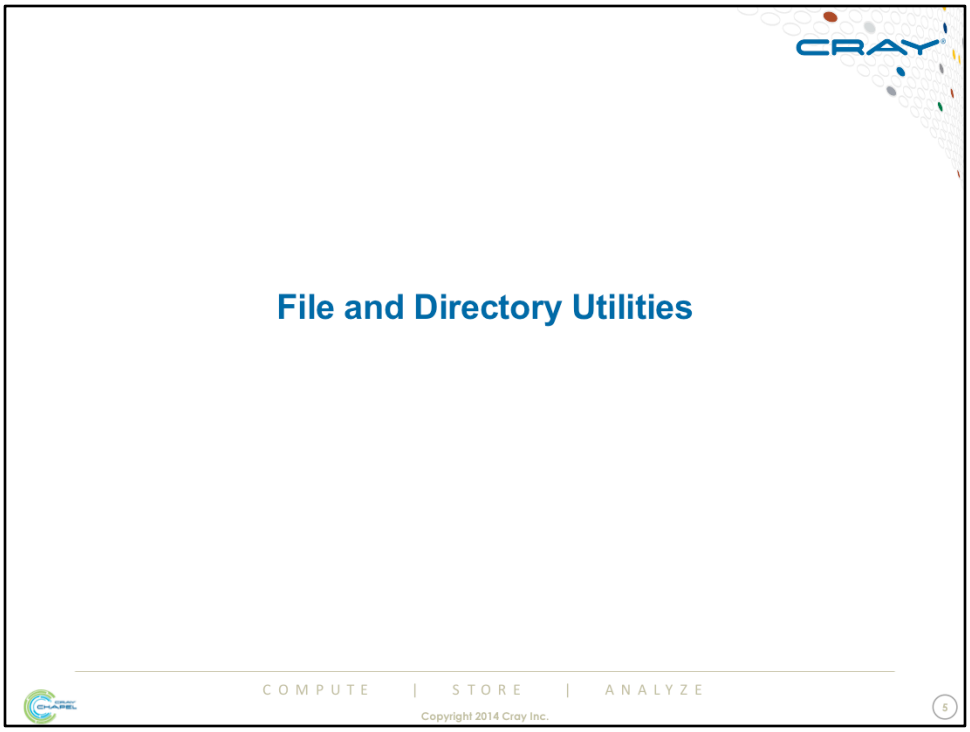
This presentation may contain forward-looking statements that are based on our current expectations. Forward looking statements may include statements about our financial guidance and expected operating results, our opportunities and future potential, our product development and new product introduction 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.

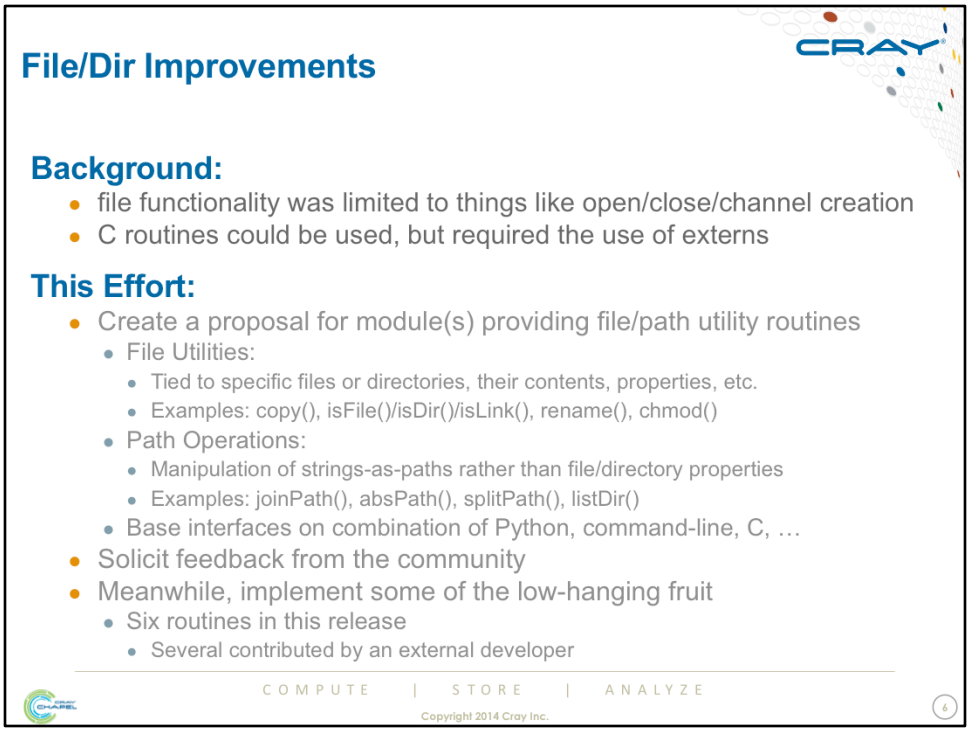
Executive Summary

- **Library support in Chapel has typically been thin**
  - Rationale: focus on language first, then library
- **Early users are now expressing desire for richer libraries**
  - Library support considered a strength of Python, Matlab, Java, C++
- **In this release cycle, we've started to focus more on libs**
  - started improving existing libraries
  - began design work for others

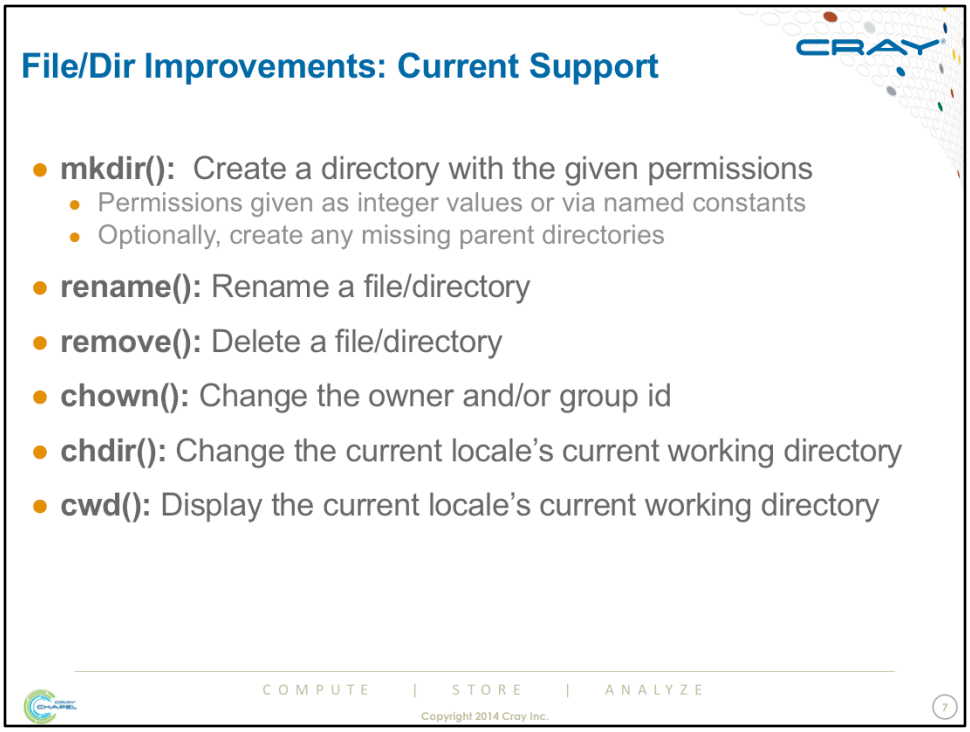
Outline

- **File and Directory Utilities**
  - Filerators: Iterators for the File System
  - Auxiliary I/O Support
- **String Library Proposal**
- **BitOps module**
- **Prefetch module**
- **Changes to other modules**
  - sorted(): a generic iterator for sorting
  - Consistent, Comprehensive Type Queries
  - readline(): read bytes into an array
- **Other Standard Library Improvements**

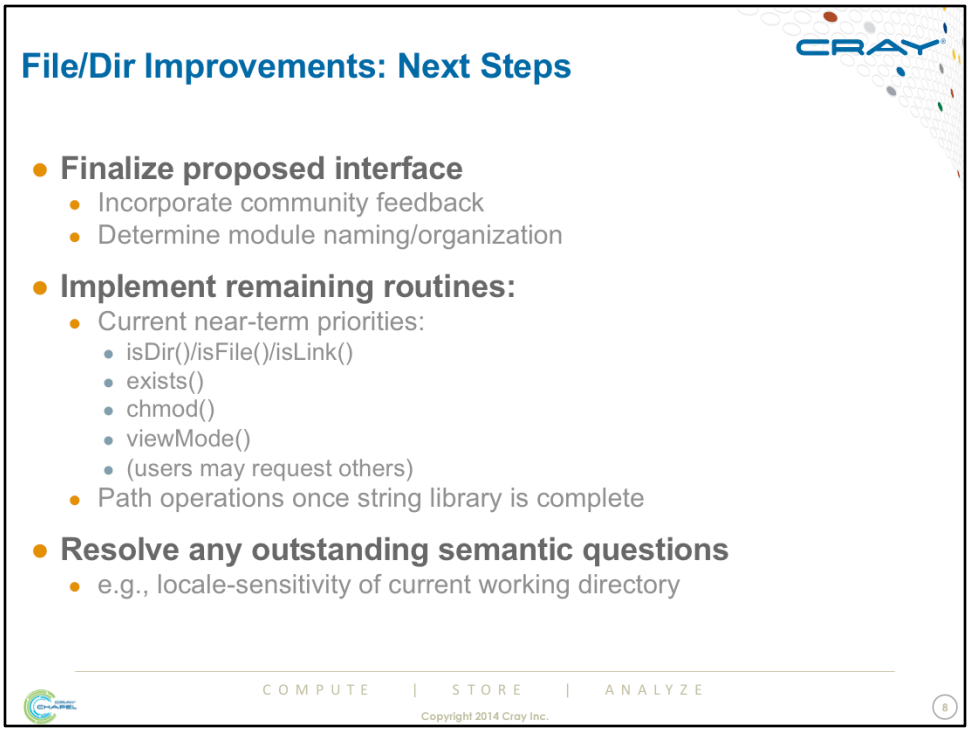

File and Directory Utilities
COMPUTE | STORE | ANALYZE
Copyright 2014 Cray Inc.
5



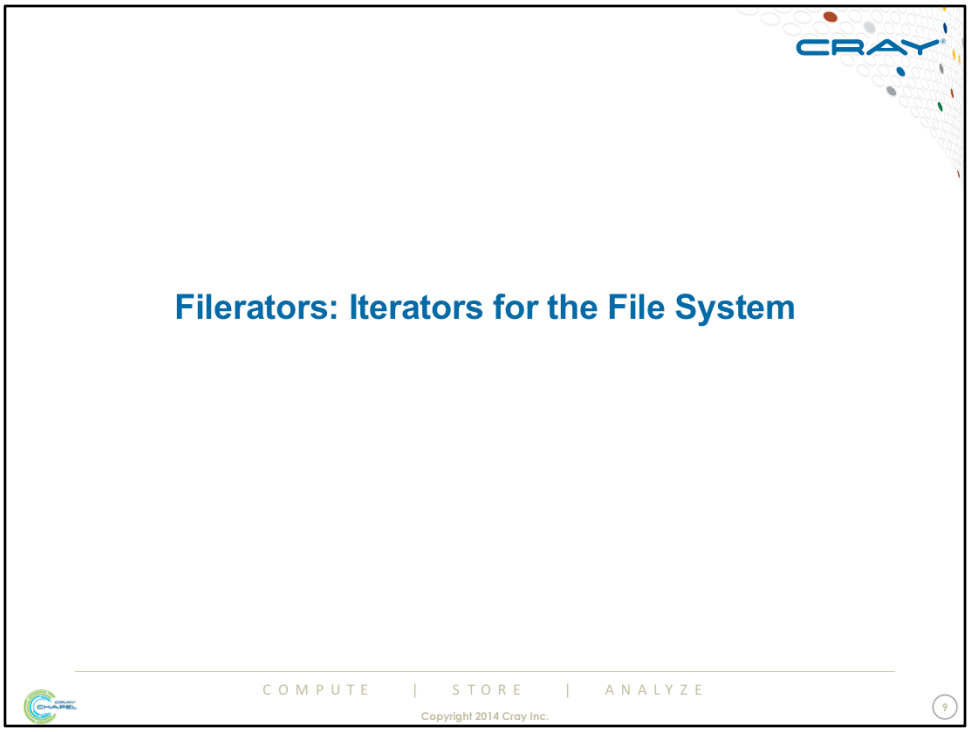
Note that, generally speaking, the file utilities can all be implemented today (modulo design questions). The path operations would benefit from waiting for Chapel's string utilities library to be completed.



Note that in the current implementation of these final two routines, each locale has its own notion of the current working directory (cwd) and any file operations that depend on it will inherit the cwd of the locale on which the task is executing. This falls out directly from taking the obvious implementation and Chapel's typical "process-per-locale" implementation approach. It's an open question whether we should do something more productive/less surprising here – for example, have the current working directory follow lexical or task scoping, or broadcast any changes to the current working directory out to all locales. Alternatively, we could more explicitly make cwd-related calls methods on locale values or abandon the notion of an implicit current working directory.



In the current release, the new routines were added to IO.chpl, an implicitly-used module. The plan for the future is to move them out into a module (or modules) of their own that would need to be explicitly used.


Filerators: Iterators for the File System
COMPUTE | STORE | ANALYZE
Copyright 2014 Cray Inc.
9

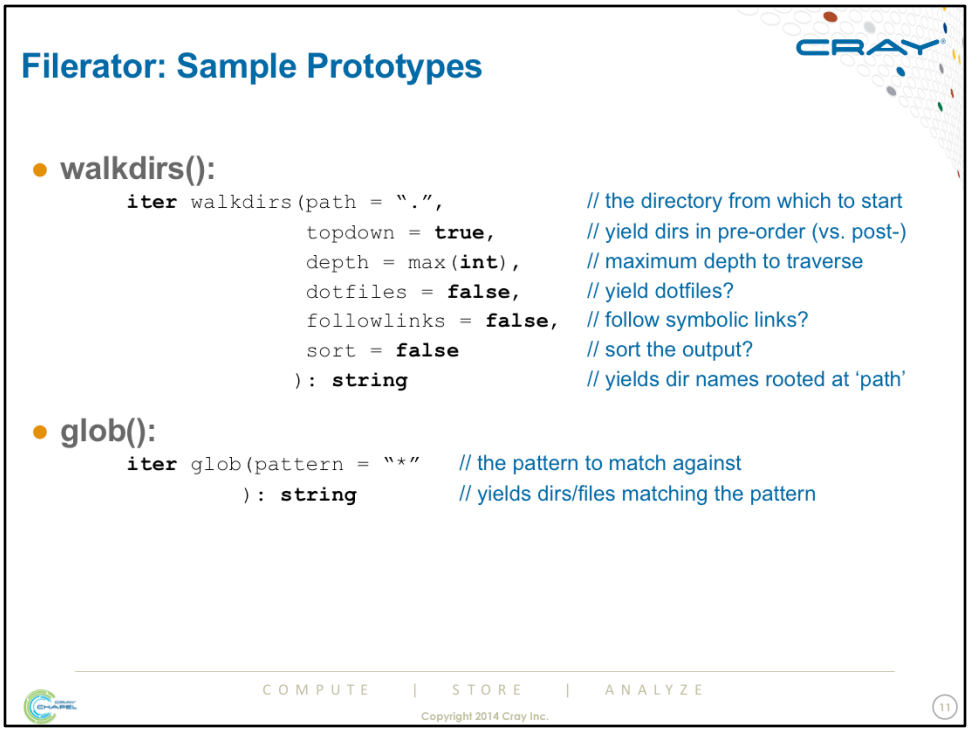
Filerators: Motivation and Description

Background:

- As noted previously, Chapel hasn't had much file/directory support

This Effort:

- Provides prototype iterators for walking directories/files
  - **walkdirs():** walks a directory tree
  - **glob()/wordexp():** provide wildcard matching capabilities
  - **listdir():** lists the files and/or directories within a given directory
  - **findfiles():** provides a simple find-like capability
- Defined in modules/standard/Filerator.chpl
  - (this name is intended as a placeholder/joke until the interface is finalized)



The interface for wordexp() is identical to glob() – in fact, only one of these routines should survive; each uses the similarly-named C routine as its implementation presently.

The interfaces for listdir() and findfiles() are not given here, primarily to save space and because they're not quite as well-baked. See the commented regions of modules/standard/Filerator.chpl for documentation.

Filerator: Sample Execution

Walk a directory structure printing out the contents:

- This code:

```
use Filerator;

for dir in walkdirs() {
  writeln("dir ", dir, " contains:");
  for file in glob(dir + "/*") do
    writeln("  ", file);
}
```

- Results in this output:

```
dir . contains:
 ./fileInSubdir.txt
 ./subdir
dir ./subdir contains:
 ./subdir/A.txt
 ./subdir/B.txt
```

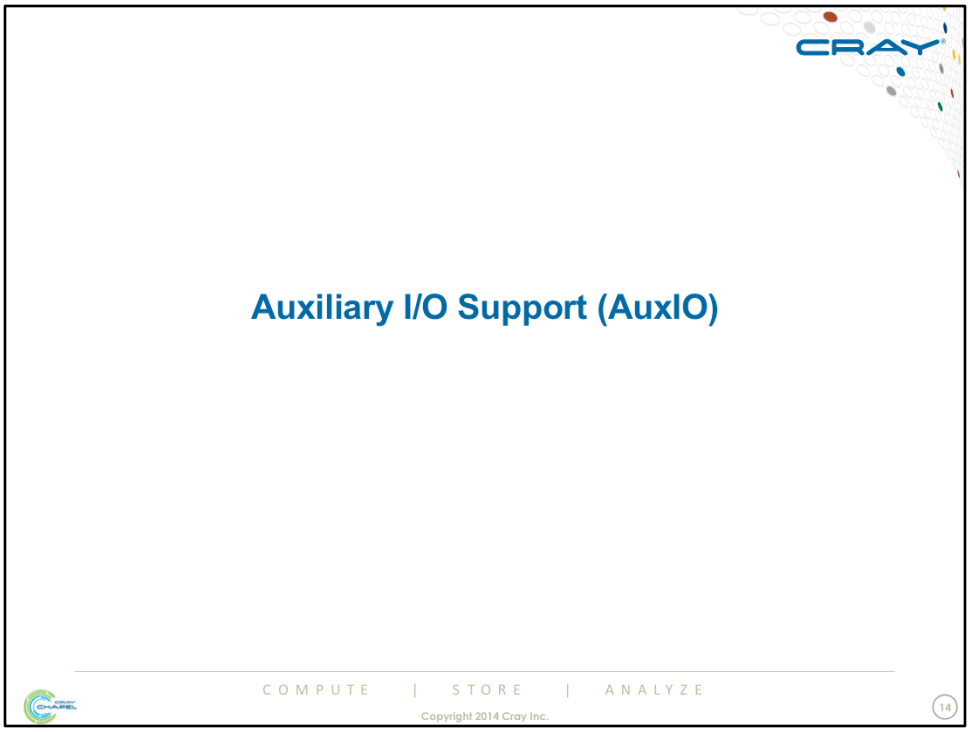
For this directory tree:

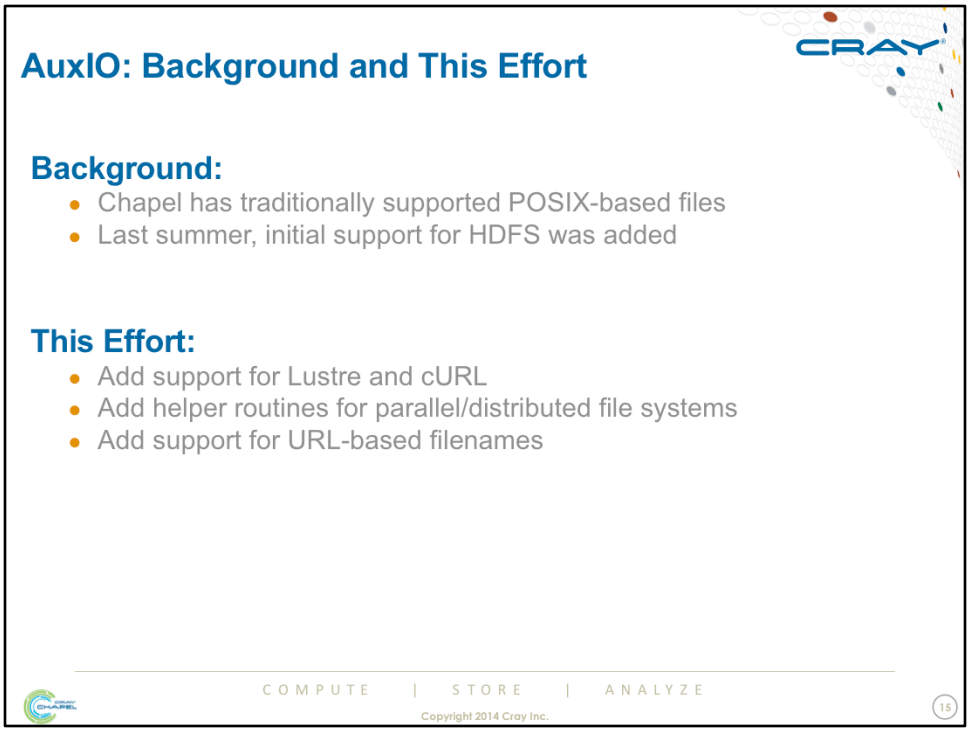
```
./
  fileInSubdir.txt
  subdir/
    A.txt
    B.txt
```

Filerator: Next Steps

Next Steps:

- wrestle with final elements of design
  - ability to filter directories in walkdirs()? (using what mechanism?)
  - support for expanding shell variables?
  - final signature for unified find[files]() iterator
- wrestle with final naming decisions
  - rename 'dotFiles' as 'hidden'?
  - rename 'walkdirs()' as 'walk()'?
  - …
- implement parallel and distributed versions
- integrate with other file/directory utility modules


Auxiliary I/O Support (AuxIO)
COMPUTE | STORE | ANALYZE
Copyright 2014 Cray Inc.
14



HDFS = Hadoop Distributed File System

AuxIO: New Routines

- **Provide low-level support for reasoning about parallelism and locality in files**

  file.getchunk(start=0, end=file.length):
  - Returns the first logical file-system block residing in [start, end]
  - Can be used to determine the natural file system block size

  file.getLocsForRegion(start, end):
  - Given a region of the file, returns the "best" locales for that region

  file.fstype()
  - Query what file or I/O system this file resides on

- **Higher-level iterators can be built from these**

AuxIO: Sample Use

```
config const path = "test.txt";
var fl = open(path, iomode.r);          // Open a file

const (start, end) = fl.getchunk();     // Query the native read size

forall i in 0..#fl.length() by start-end {
  const locs=fl.locsForRegion(i,i+len);
  on locs[0] {
    var reader = fl.reader(start=i, end=i+start-end,
                           locking=false);
    …consume data using this reader…
  }
}
```

AuxIO: Curl Support

- **Two interfaces:**
  - One that treats Curl similarly to standard Chapel file I/O
    - regex support, channels, etc.
  - A second “setopt”-based interface (a la pyCurl)
  - (the two can used together)

- **Example code:**

```
// Connect to an IMAP site, and fetch mail from the inbox
config const username = "user",
             passwd   = "xxxx",
             imapSite = "your_imap_site_here";

var handle = open(url=imapSite+"/INBOX/;UID=1", mode=iomode.cw);
handle.setopt((curlopt_username, username),(curlopt_password, passwd));
handle.perform();
handle.close();
```

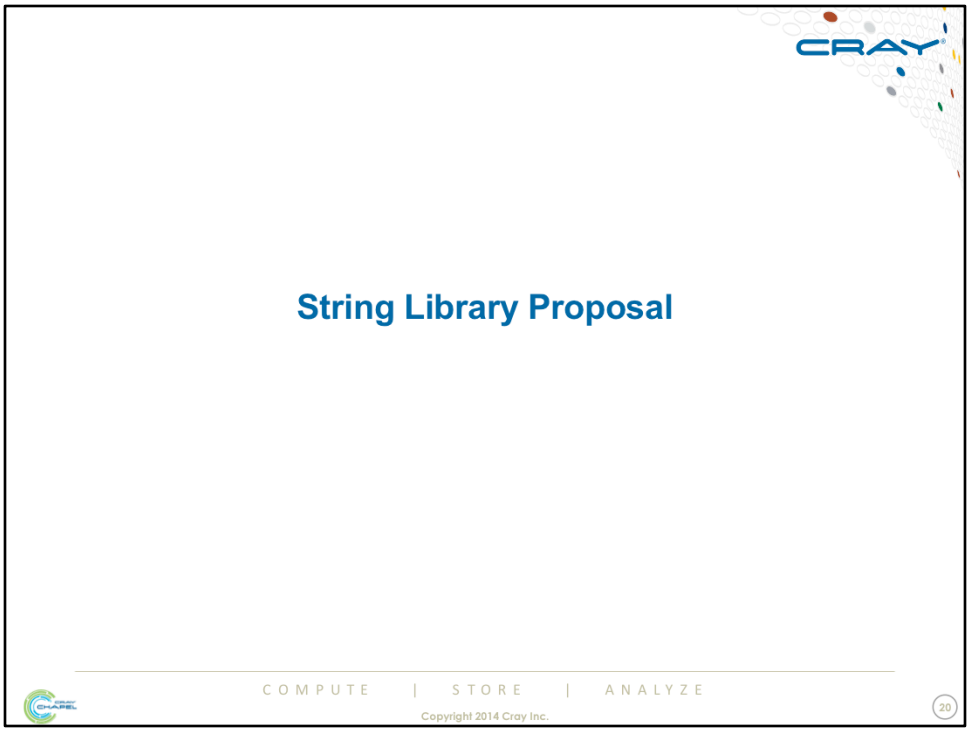
AuxIO: Docs, Next Steps

For more information:

- doc/technotes/README.auxIO
- doc/technotes/README.curl
- doc/technotes/README.hdfs

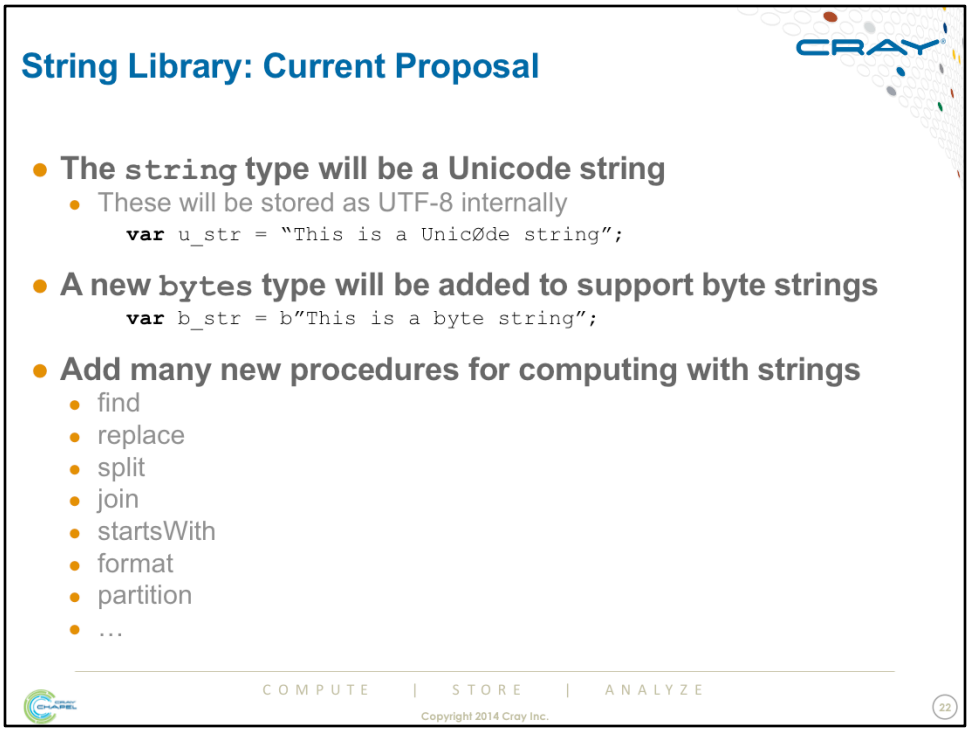
Next Steps:

- Gain more experience and user feedback from AuxIO features
- Improve features based on the above
- Better unify domain maps/locales and distributed file systems/disks (?)


String Library Proposal
COMPUTE | STORE | ANALYZE
Copyright 2014 Cray Inc.
20

String Library: Background

- Strings do not have many operations
- Currently very little to no support for Unicode
- We want to improve on both of these aspects

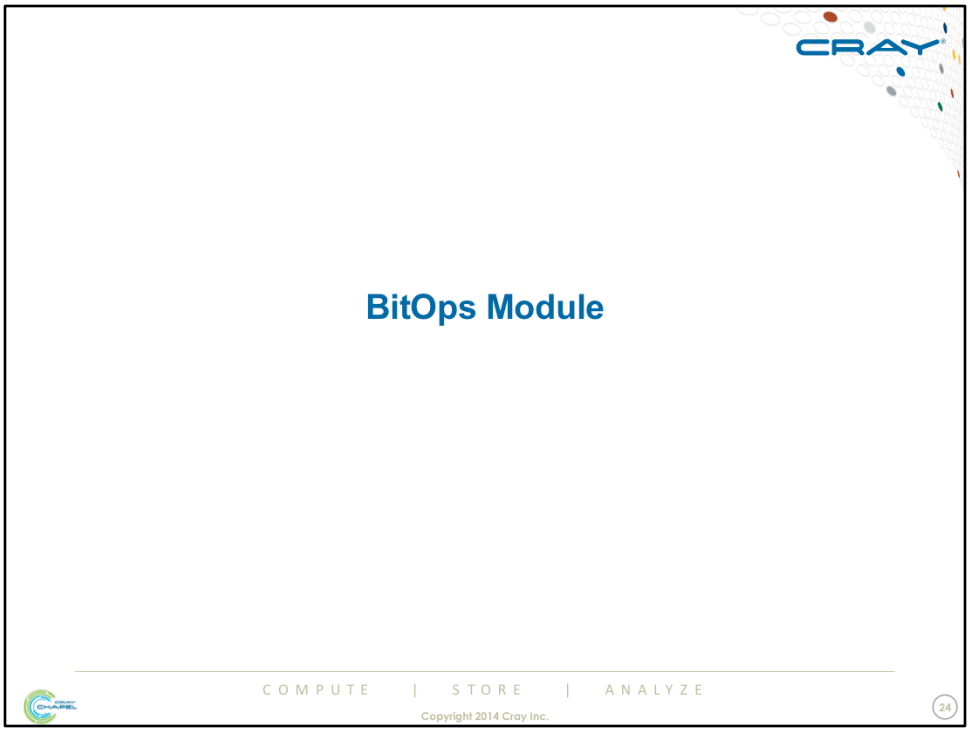


An more complete list of planned functions:

```
length          startsWith      toLower          isAlpha          normalize
size            endsWith        toTitle          isAlnum          translate
find            these           capitalize       isPrintable      format
replace         graphemes       isUpper          isDecimal        partition
split           this            isLower          isDigit          splitLines
join            contains        isTitle          isNumeric
strip           toUpper         isSpace          caseFold
```

String Library: Next Steps

- **Resolve Open Questions:**
  - Indexing Strategy – what units do `find` and `substring` work with?
    - Code points?
    - Bytes?
    - Extended grapheme clusters?
    - Current plan is code points and an opaque type to allow O(1) access
  - Should we allow Unicode identifiers?
    - Would be nice long term, not a priority right now
    - Will require the compiler to know more about Unicode
  - Interoperability with C strings
    - `string` will not be directly convertible to a `c_string`
    - `bytes` will be
- **Begin implementation**
  - Replace current `string` type with `string_rec`
  - Expand `string` to work with Unicode
  - Implement library routines


BitOps Module
COMPUTE | STORE | ANALYZE
Copyright 2014 Cray Inc.
24

BitOps: Background

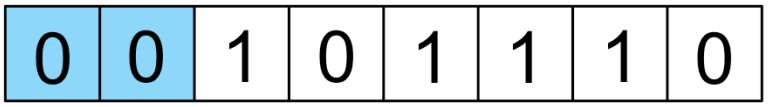
- **The BitOps module was designed to support bit-level operations**
- **A placeholder has been in-place for some time now**
  - provided as an early example of "how libraries would work in Chapel"
    - created in 2006 and not really touched since 2007
  - not many routines
  - not implemented well
    - software rather than hardware implementations
    - (and not particularly good software implementations at that)

BitOps: This Effort

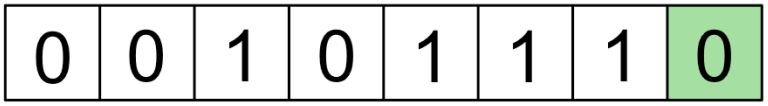
- **Added three new functions**
  - **clz():** count leading zeros
  - **ctz():** count trailing zeros
  - **popcount():** a.k.a. population count, sideways sum, hamming weight
- **Implemented using C intrinsics when possible**
  - Will generate the equivalent instructions with processor support
    - When --specialize is thrown (implied by --fast)
  - Falls back to pure C implementations otherwise

BitOps: clz and ctz

- **Count Leading Zeros**
  - Returns the number of zeros after the highest bit that is set

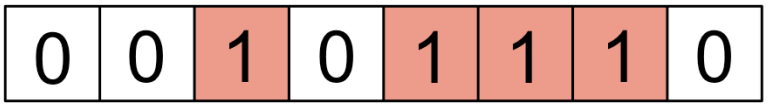


- **Count Trailing Zeros**
  - Returns the number of zeros before the lowest bit that is set



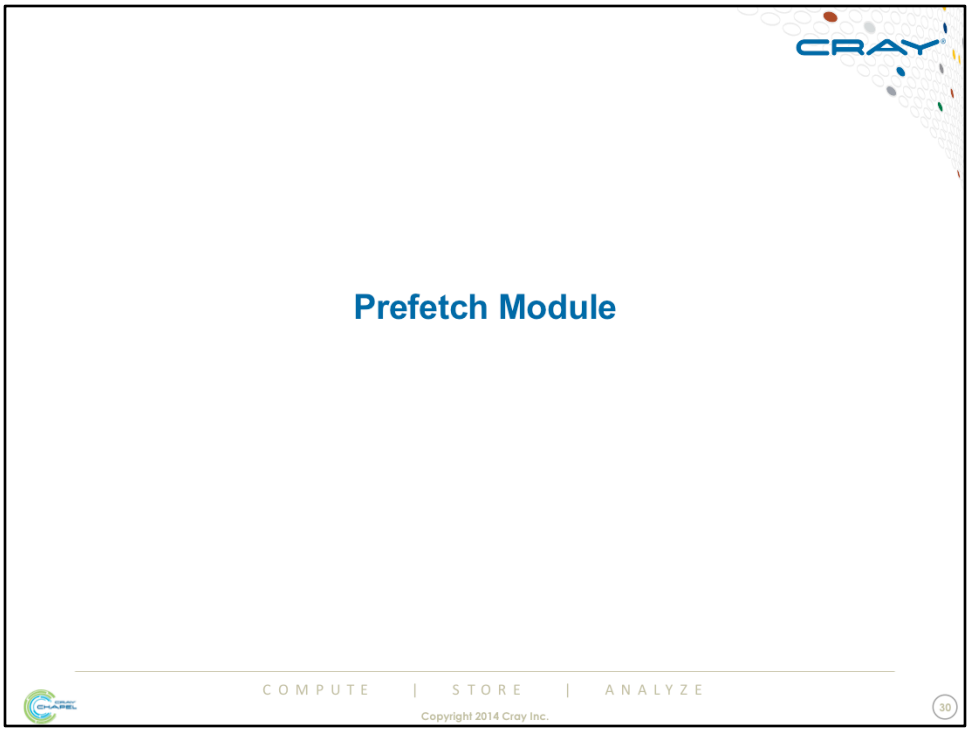
BitOps: popcount

- **Population Count**
  - Returns the number of bits that are set



BitOps: Next Steps

- **Continue to add additional functions**
  - Prioritize based on user interest and input


Prefetch Module
COMPUTE | STORE | ANALYZE
Copyright 2014 Cray Inc.
30

Prefetch Module

This Effort:

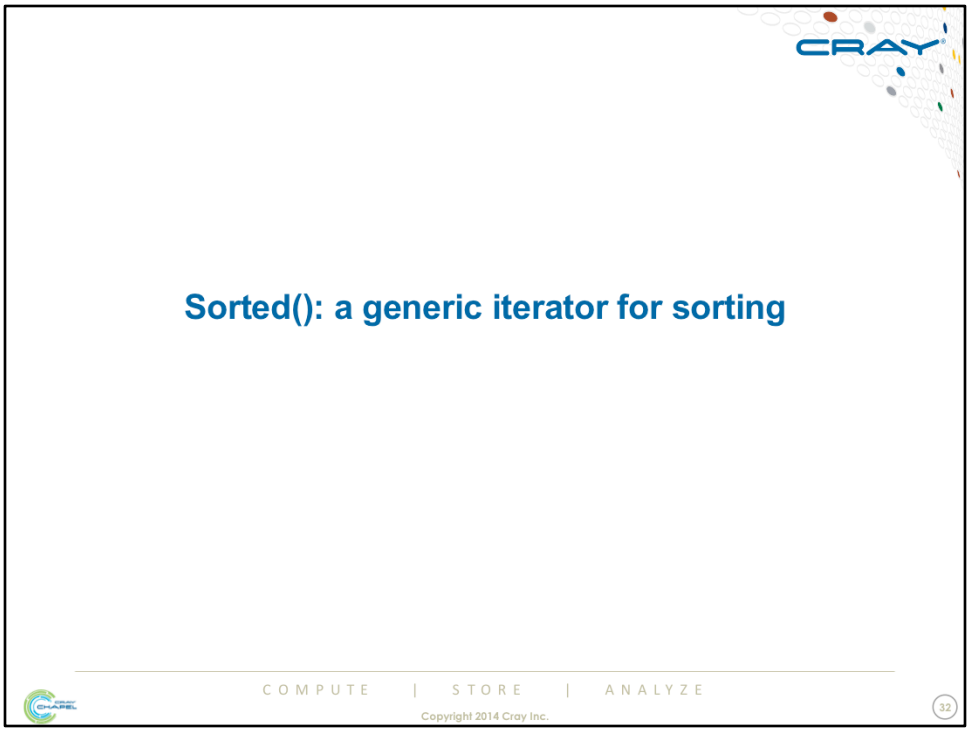
- A new module with one new function
  - **prefetch():** Reads in a cache line from the location pointed to by 'addr'

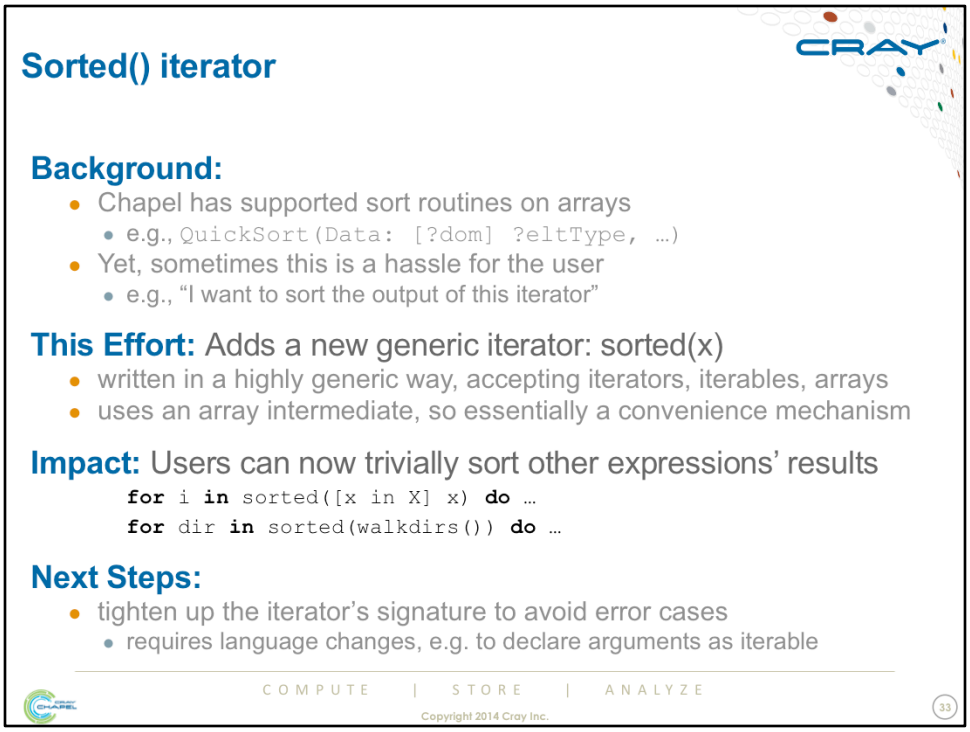
```
proc prefetch(addr: c_ptr)
proc prefetch(addr: c_void_ptr)
```

- Maps down to intrinsics or pragmas
  - Generates a prefetch instruction in the executable
  - `__builtin_prefetch` on most compilers
  - `#pragma mem prefetch` with PGI

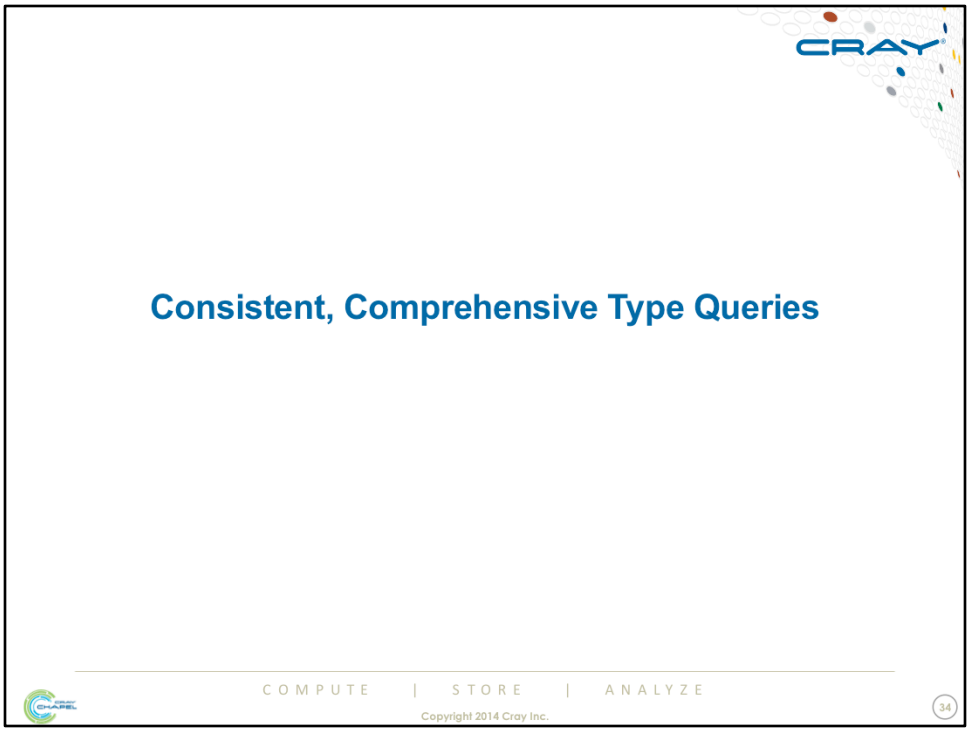
Next Steps:

- None planned at present


Sorted(): a generic iterator for sorting
COMPUTE | STORE | ANALYZE
Copyright 2014 Cray Inc.
32



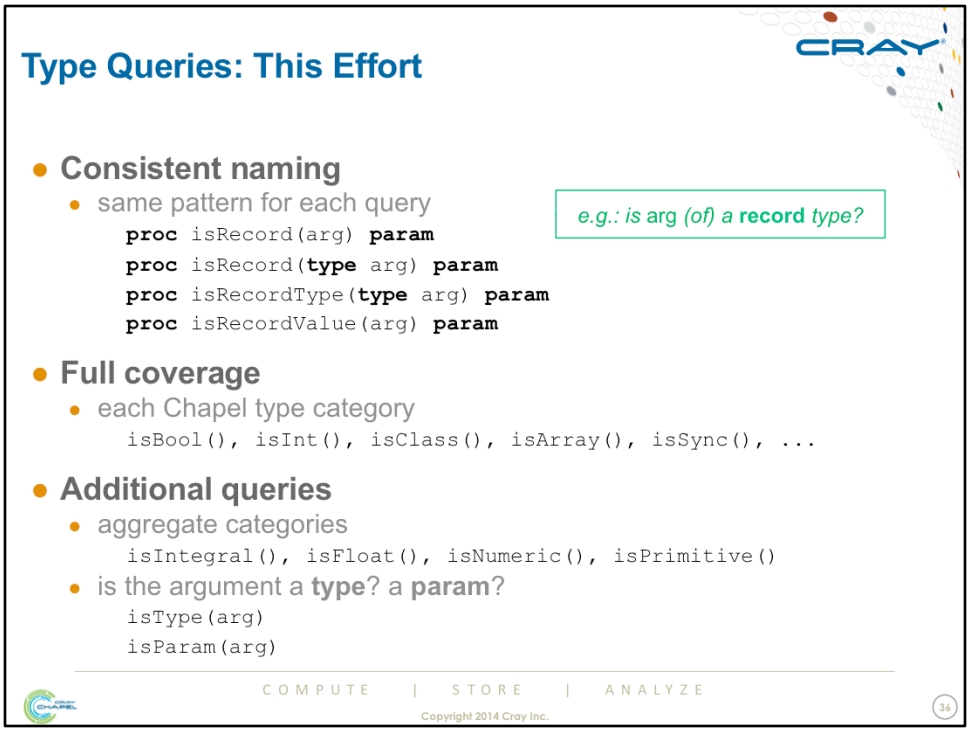
Note that sorting the output of walkdirs() will have the same result as passing 'sorted=true' to the routine itself, but will be more expensive since, when the walkdirs() iterator is sorting, it can sort a single subdirectory at a time.


Consistent, Comprehensive Type Queries
COMPUTE | STORE | ANALYZE
Copyright 2014 Cray Inc.
34

Type Queries: Background

- **A variety of functions to test types**

  - diversity of naming schemes
    - some deviations from Chapel type names
  - some Chapel types were not covered



isIntegral(): returns true for ints and uints

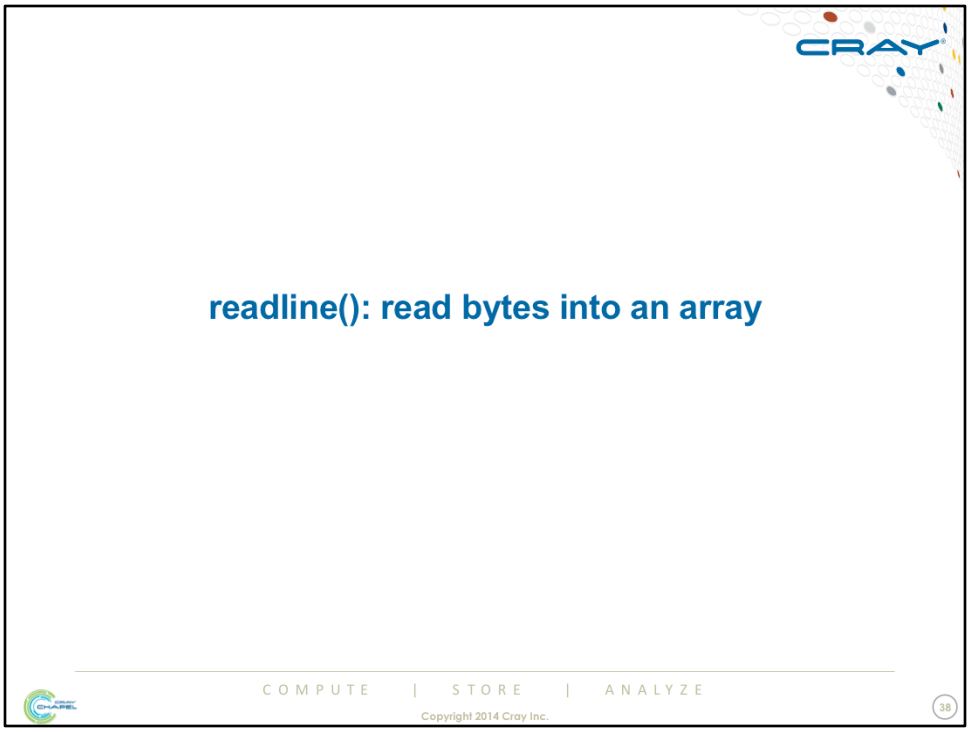
isFloat(): returns true for reals and imags

isNumeric(): returns true for integrals, floats, and complexes

isPrimitive(): returns true for void, bools, numerics, and strings

Type Queries: Next Steps

- **Should the names for is*Value() be changed?**
  - `isIntValue(1) == true`, obviously
  - `isIntValue(x) == true` for `var x: int;`
    - Is this confusing given that 'x' isn't strictly a value, but is a variable?
    - Use hasIntValue() instead?
    - Or live with the imprecision for simplicity?
- **Add additional queries? (e.g., isConst())?**


readline(): read bytes into an array
COMPUTE | STORE | ANALYZE
Copyright 2014 Cray Inc.
38

readline: Background

- In the past, readline has been supported for strings

```
proc channel.readline(ref arg:string): bool
```

- An array of bytes is useful for encoded data formats
- Working with an array of bytes was cumbersome

```
var ret: string;
stdin.readline(ret);
var data: [1..ret.length] uint(8);
data = [i in 1..ret.length] ascii(ret.substring(i)): uint(8);
```

readline: This Effort

- **Allow reading directly into a 1D rectangular array**

```
proc channel.readline(arg: [] uint(8), out numRead : int,
                      start = arg.domain.low,
                      amount = arg.domain.high - start,
                      out error: syserr) : bool
```

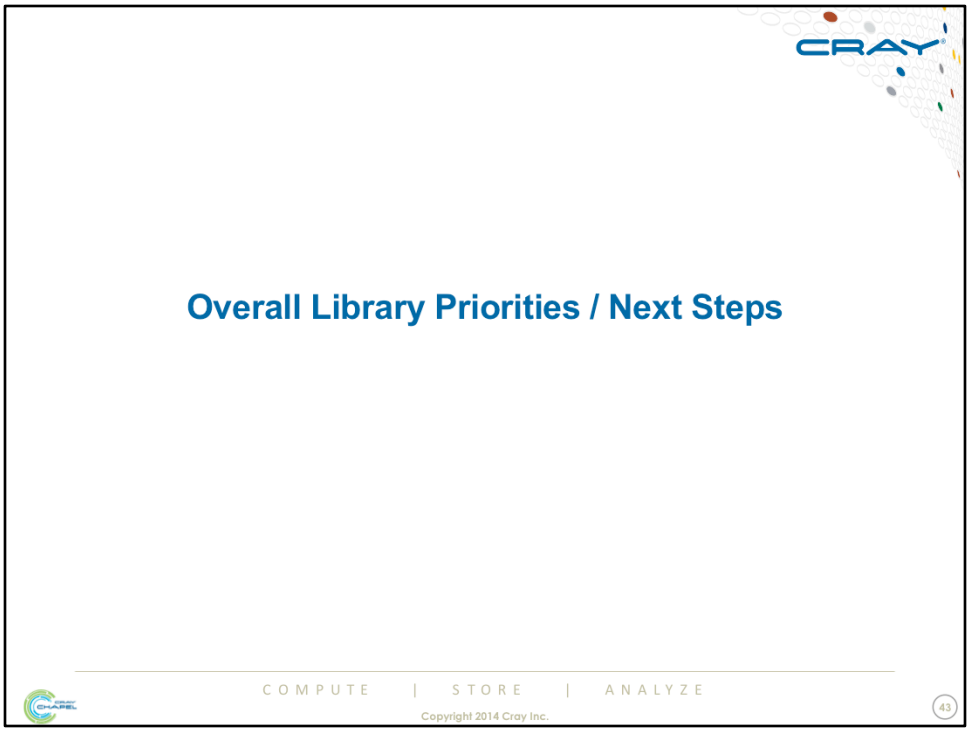
Next Steps:

- How does this fit into future string implementations?
- Is the naming sufficiently clear?


Other Standard Library Improvements
COMPUTE | STORE | ANALYZE
Copyright 2014 Cray Inc.
41

Other Standard Library Improvements

- Added the ability to sort in reverse order to Sort routines
- Advanced iterators now use atomics rather than syncs
- Added a printMemLeaks() routine to help track down leaks
- Improved the time-based seed generator in Random.chpl
- Removed stale “Profiling” module code


Overall Library Priorities / Next Steps
COMPUTE | STORE | ANALYZE
Copyright 2014 Cray Inc.
43

Overall Library Priorities / Next Steps

- **Create a path for documenting libs, from source to web**
  - **Approach:** Use Sphinx (implies moving away from 'chpldoc')
    - **effort:** write a Sphinx module for Chapel
    - **rationale:** piggyback on others' work creating pretty/useful web pages
- **File/Path Utilities**
  - complete proposal, incorporating feedback
    - resolve multi-locale issues such as cwd-per-locale
  - implement remaining file utilities
  - implement path utilities when strings are ready
- **String library**
  - complete proposal, incorporating feedback
  - implement, once strings are records and UTF-8
- **Add additional low-hanging BitOps**
- **Begin investigation of numerical libraries (e.g., BLAS)**

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, URIKA, and YARCDATA. 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.


CRAY
CHAPEL
CRAY
THE SUPERCOMPUTER COMPANY
http://chapel.cray.com
chapel_info@cray.com
http://sourceforge.net/projects/chapel/