Hewlett Packard Enterprise

CHAPEL RELEASE NOTES, 1.25.1 / 1.26.0: LIBRARY IMPROVEMENTS

Chapel Team

December 9, 2021 / March 31, 2022

OUTLINE

- New Package Modules
- Argument Parser and '--help'
- Standard library Stabilization
- Other Library Improvements

NEW PACKAGE MODULES

- Copy Aggregation Library
- Socket Library
- Go-Style Channels
- Concurrent Map Module


COPY AGGREGATION LIBRARY

COPY AGGREGATION LIBRARY

Background and This Effort

Background: Copy aggregation can significantly speed up fine-grained copies

- Initially added to Arkouda in the Chapel 1.22 timeframe
- Chapel 1.24 added an automatic copy aggregation optimization to the compiler
- A generally useful feature that we ultimately want in Chapel's standard library

This Effort: Expose existing copy aggregation to users

- Current implementation requires that one side of the copy is always local
  - 'DstAggregator' when source is local and destination may be remote
  - 'SrcAggregator' when destination is local and source may be remote

COPY AGGREGATION LIBRARY

Example

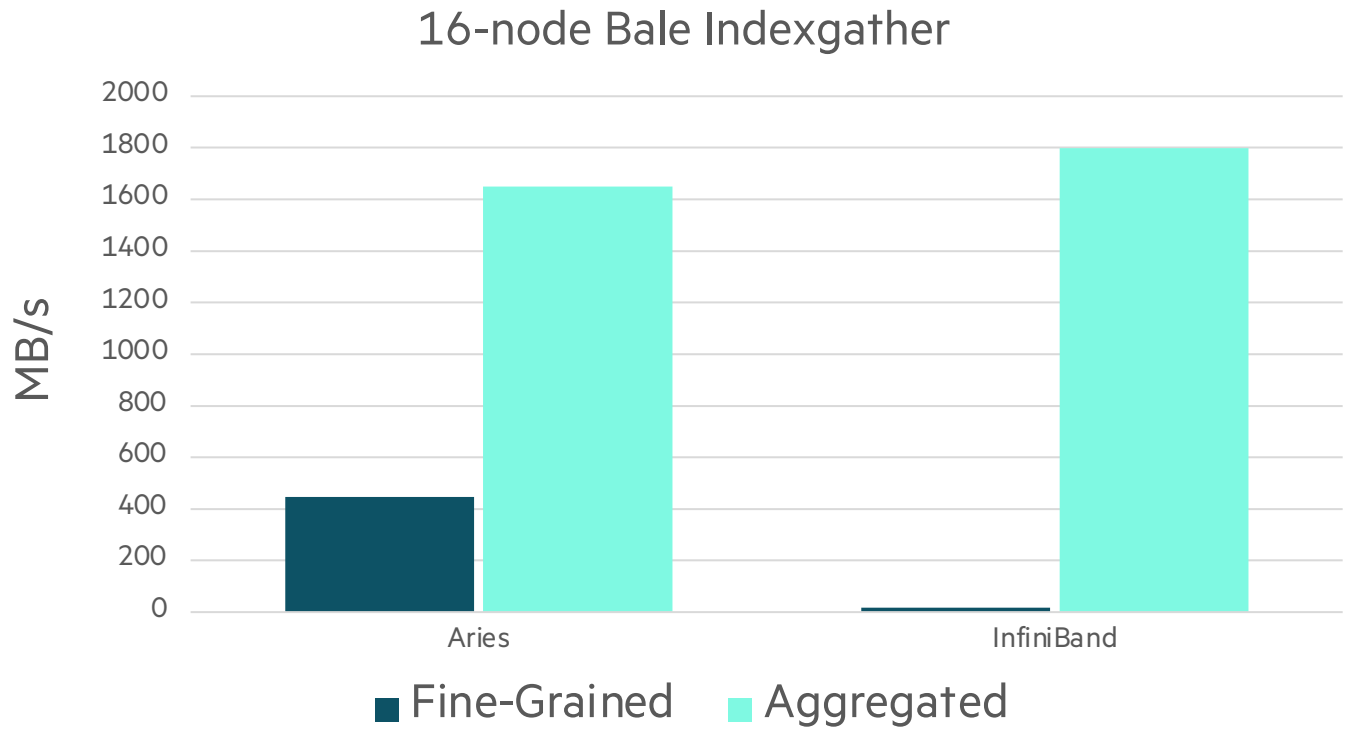
- Reverse array using aggregation:

```
use BlockDist, CopyAggregation;

const size = 10000;
const D = newBlockDom(0..size);
var A, reversedA: [D] int = D;

forall (rA, i) in zip(reversedA, D) with (var agg = new SrcAggregator(int)) do
  agg.copy(rA, A[size-i]);
```

COPY AGGREGATION LIBRARY

Impact

- Copy aggregation is available to users, provides a large speedup for fine-grained copies
  - Particularly on networks where Chapel has poor small-message rates



COPY AGGREGATION LIBRARY

Next Steps

- Support aggregating more operations
  - Add copy aggregator where both source and destination can be remote
  - Add atomic aggregators
  - Add support for arbitrary user-defined aggregation

- Further improve performance and reduce memory footprint
  - Including optimizing local aggregations

SOCKET LIBRARY

SOCKET LIBRARY

Background and This Effort

Background: TCP and UDP socket programming have not been supported in Chapel

- Could only be done through C interoperability
- Blocking socket calls are a mismatch for Qthreads-based user-level tasking

This Effort: Implemented a 'Socket' module in Chapel

- Implemented as a Google Summer of Code Project
  - Student: Lakshya Singh
  - Mentors: Ankush Bhardwaj (Chapel GSoC 2020 Alum), Krishna Kumar Dey (Chapel GSoC 2019 Alum), Michael Ferguson

SOCKET LIBRARY
Example

```
use Socket;

var port: uint(16) = 8812;
var host = "127.0.0.1";
var addr = ipAddr.ipv4(IPv4Localhost, port);

proc server(srvSock: udpSocket) throws {
  // receive 5 bytes from the connected client
  var got = srvSock.recv(5);
  // do something with 'got'
}
```

```
proc client() throws {
  var clientSock = new udpSocket();
  // send "hello" to the server
  var n = clientSock.send(b"hello", addr);
}

proc main() throws {
  // create a new server
  var srvSock = new udpSocket();
  bind(srvSock, addr);

  // start a server and a client in different tasks
  cobegin {
    server(srvSock);
    client();
  }
}
```

SOCKET LIBRARY

Status and Next Steps

Status:

- Included in 1.26 as a package module
- Uses 'libevent' to allow useful work in other Chapel tasks while waiting on network activity
- Implementation has some caveats at present:
  - only works with C back-end (e.g., 'CHPL_TARGET_COMPILER=gnu')
  - only works with 'CHPL_TASKS=qthreads'

Next Steps:

- Address caveats listed above—especially the problems when building with the LLVM back-end
- Use the I/O plugin facility to arrange socket I/O calls to work with libevent
- Study the performance of servers written in Chapel
- Add a helper class to make it easier to implement a server and demonstrate a simple HTTP server

GO-STYLE CHANNELS

GO-STYLE CHANNELS

Background and This Effort

Background: Chapel intends to support general parallel programming

- One missing idiom was a message queue like in Go and Rust (known as a ‘channel’ there)

This Effort: Implement Go-style channels in Chapel

- Implemented as a Google Summer of Code Project
  - Student: Divye Nayyar
  - Mentors: Michael Ferguson, Aniket Mathur (Chapel GSoC 2020 Alum)

Status: Included in 1.26 as a package module

Next Steps:

- Add compiler support for blocking ‘select’ statements (see example on next slide)
- Investigate and improve performance
- Enable channels to communicate across locales

GO-STYLE CHANNELS
Examples

```
// simple send/recv
use Channel;

var channel1 = new channel(int, 5);

begin {
 channel1.send(4);
}
var recv1: int;
channel1.recv(recv1);
writeln("Received ", recv1);
```

```
// current approach for writing a select
use selectOperation;
var sel1: SelectBaseClass = new shared
      SelectCase(x1, channel1, recv, 0),
    sel2: SelectBaseClass = new shared
      SelectCase(x2, channel2, send, 0);
var arr = [sel1, sel2];

const option = selectProcess(arr);

if option == 0 {
  writeln("Received: ", x1);
} else {
  writeln("Sent: ", x2);
}
```

GO-STYLE CHANNELS

Examples

```
// simple send/recv
use Channel;

var channel1 = new channel(int, 5);

begin {
 channel1.send(4);
}
var recv1: int;
channel1.recv(recv1);
writeln("Received ", recv1);
```

```
// proposed select statement syntax
select {
  when var x1 = channel1.recv() {
    writeln("Received: ", x1);
  }
  when channel2.send(x2) {
    writeln("Sent: ", x2);
  }
}

// Syntax above requires compiler integration
```

CONCURRENT MAP MODULE

CONCURRENT MAP MODULE

Background:

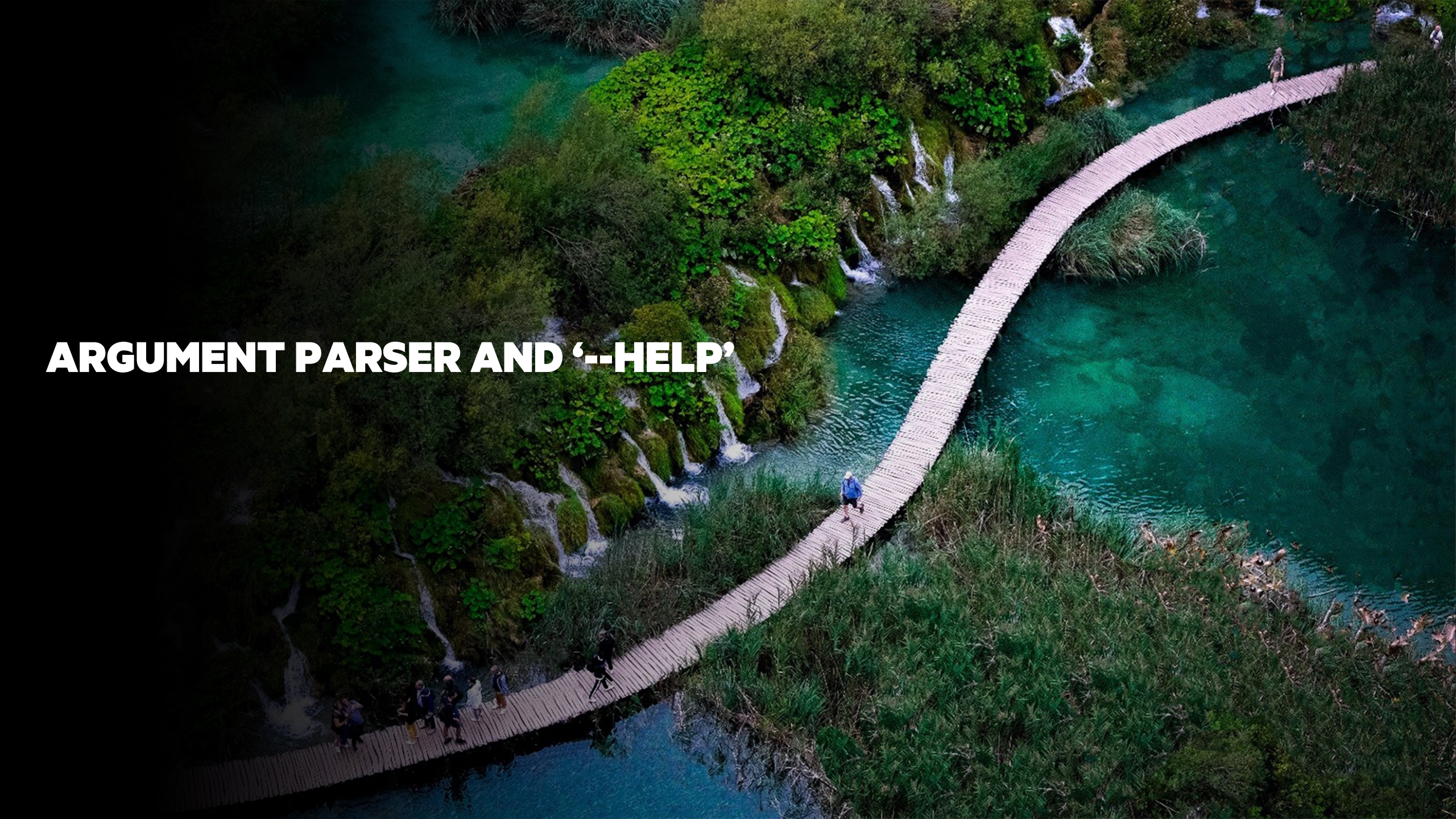
- A high-performance, concurrent map
- Offers an API like the standard 'Map', e.g., 'add( )', 'set( )', and 'remove( )'
- Uses the 'EpochManager' package module for epoch-based memory management
- Only supported on x86_64 with GCC or Clang
- Implemented as a Google Summer of Code Project
  - Student: Garvit Dewan
  - Mentor: Louis Jenkins (Chapel GSoC 2017 Alum)

This Effort:

- Reviewed and merged 'ConcurrentMap' with help from the author
- Offered as a package module

Next Steps:

- Get 'ConcurrentMap' to work with managed classes ('owned', 'shared')
- Consider supporting more platforms



ARGUMENT PARSER AND '--HELP'

ARGUMENT PARSER AND '--HELP'

Background:

- Argument parsing library released in 1.25 for handling arguments passed to 'main( )'
  - updated in 1.25.1 to handle '--help' requests

This Effort:

- Add customizable handling of the '--help' flag and optional help text generated from the defined arguments
  - Handles help requests in the form of '-h' and '--help'
  - Builds a help message and a usage message
  - Prints the combined help and usage messages and exits when help requested or bad input

Open Discussions:

- How should argument parser's help and usage messages be formatted? (#18687)
- How should help message and metadata be defined for ArgumentParser options/flags? (#18646)

ARGUMENT PARSER AND '--HELP'

Example help output:

- No additional Chapel code needed to access this functionality

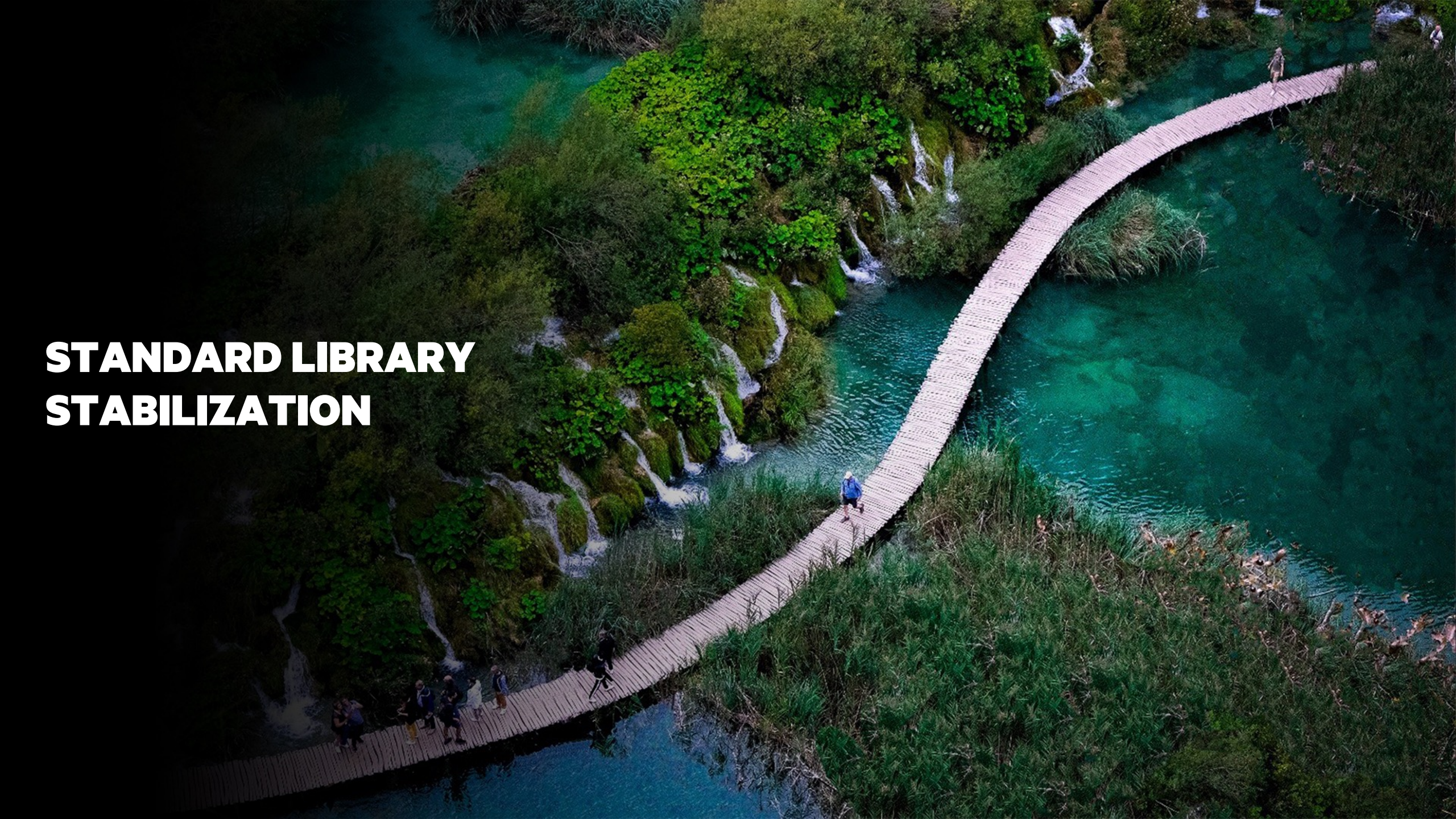
```
$ quickStart -h
USAGE: quickStart <POSITIONAL> [-h, --help] [--debug] [--optional <OPTIONAL>]

ARGUMENTS:
        POSITIONAL

OPTIONS:
        -h, --help              Display this message and exit
        --optional <OPTIONAL>
```

See source in the ArgumentParser documentation

Next Steps: Continue to add new features, improve message formatting

- Constrain option values
- Conditionally require/exclude other arguments

STANDARD LIBRARY STABILIZATION



STANDARD LIBRARY STABILIZATION

Background

- The effort to release Chapel 2.0 is currently focused primarily on standard library stabilization
  - *Stabilization:* The interface should not change in ways that break existing programs

- We have been reviewing standard libraries
  - On even weeks, we review a new module, scrutinizing
    - the name of the module itself
    - names of public types, enums, global variables, constants, …
    - names of public procedures, arguments
    - behaviors / definitions of all public symbols

  - On odd weeks we follow up on a previously reviewed module

  - Also created a sub-team to review the IO module
    - IO sub-team members meet regularly and call full-team meetings when part of the interface is ready for discussion

STANDARD LIBRARY STABILIZATION

This Effort

- This release we continued that cadence
- As of the Chapel 1.25 release, we had:
  - Reviewed 23 standard libraries
  - Stabilized 2 standard libraries

- During this release cycle we:
  - Reviewed 7 more standard libraries
  - Re-reviewed 7 standard libraries
  - Implemented many changes based on reviews

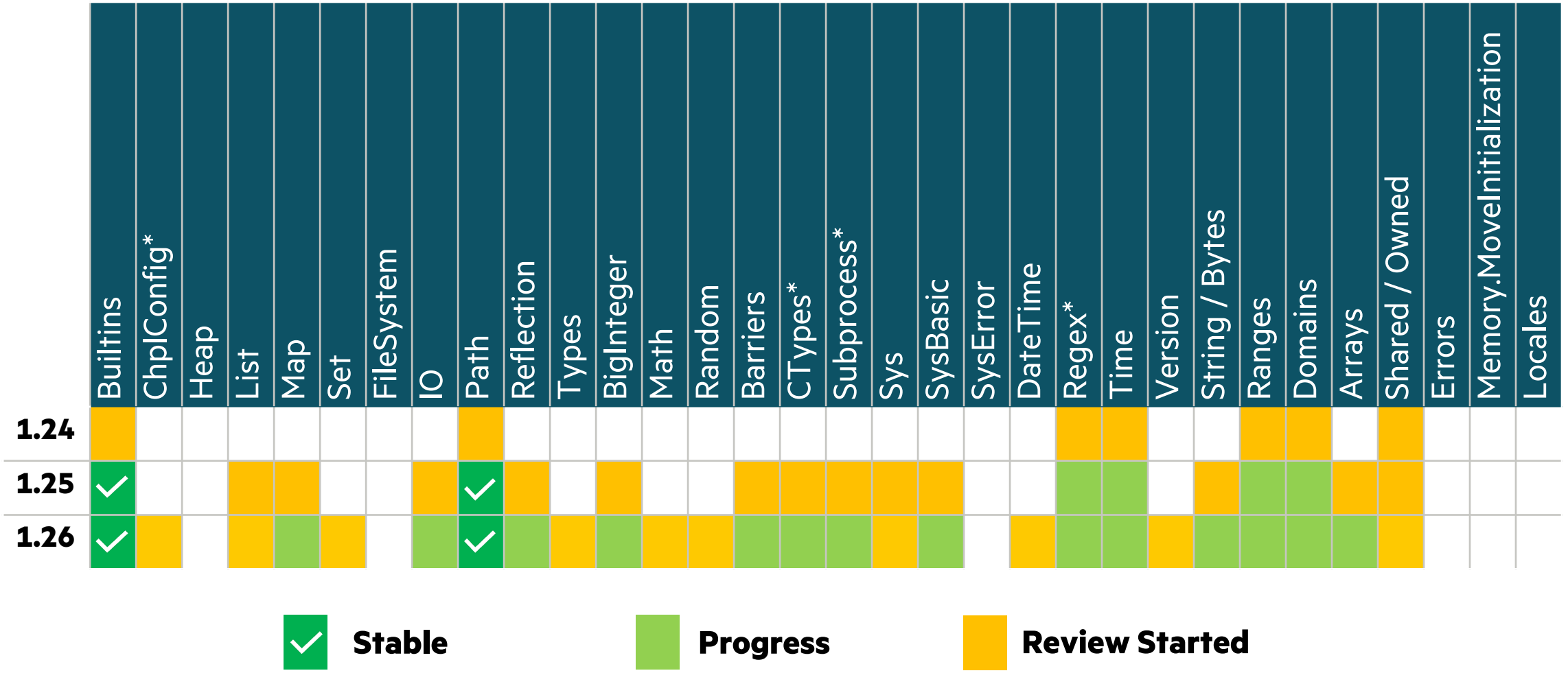
STANDARD LIBRARY STABILIZATION

Status: In Numbers

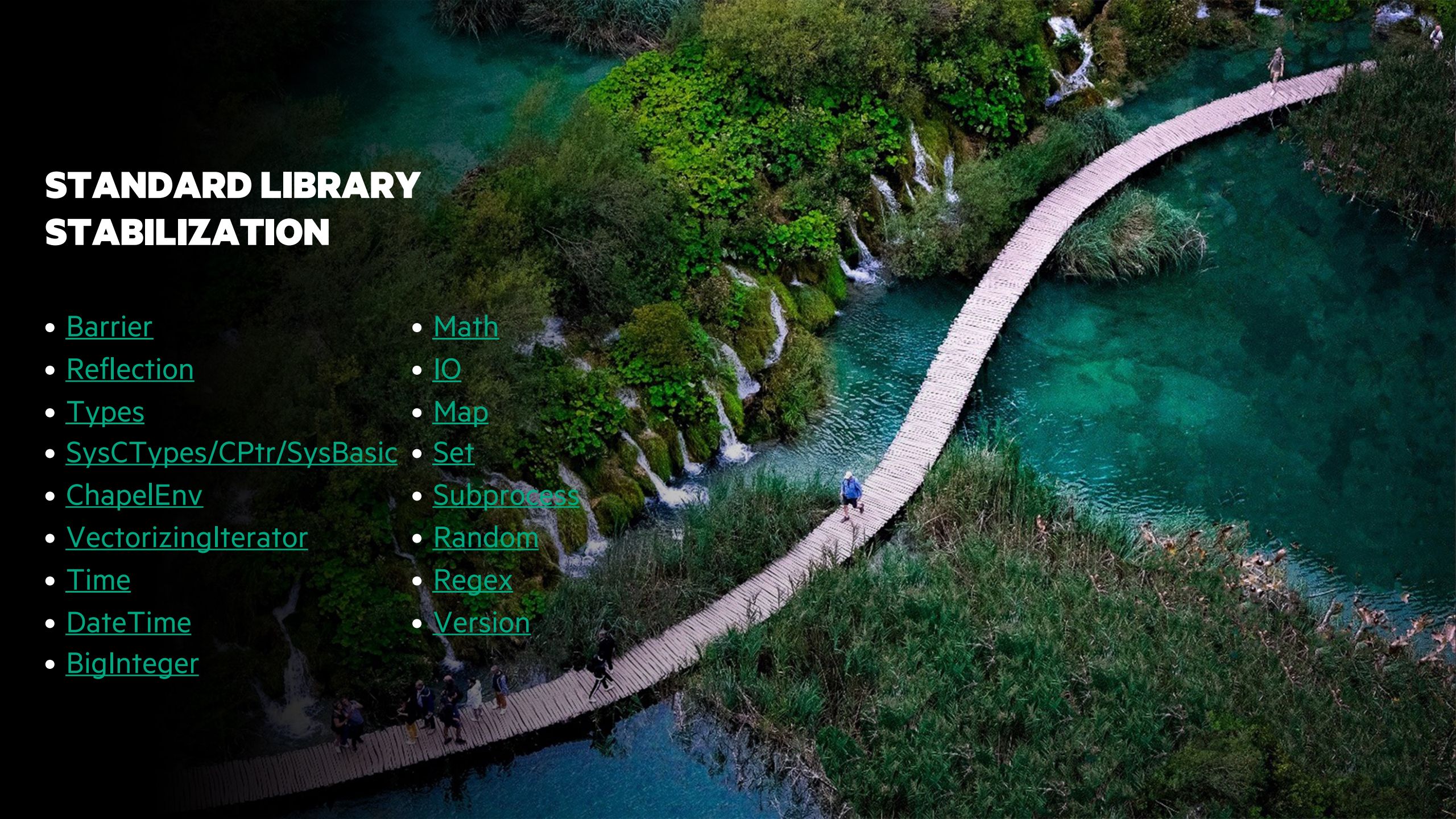
- 30 modules reviewed
- 2 modules stabilized:
  - Path, Builtins
- 6 modules that are close to being stabilized:
  - CTypes, Sys, Regex, Time, Version, Subprocess
- 7 modules that we've decided not to stabilize before Chapel 2.0:
  - CommDiagnostics, Memory[.Diagnostics], BitOps, GMP, DynamicIters, VectorizingIterator, Help
- 6 modules that still need review:
  - SysError, Errors, FileSystem, Heap, Memory.MoveInitialization, Locales
    - See the Ongoing Work deck for more on Locale model design

STANDARD LIBRARY STABILIZATION

This Effort: Overview



* - ChapelEnv was renamed to ChplConfig, CPtr / SysCTypes were combined and renamed to CTypes, Spawn was renamed to Subprocess, and Regexp was renamed to Regex



STANDARD LIBRARY STABILIZATION

- Barrier
- Reflection
- Types
- SysCTypes/CPtr/SysBasic
- ChapelEnv
- VectorizingIterator
- Time
- DateTime
- BigInteger
- Math
- IO
- Map
- Set
- Subprocess
- Random
- Regex
- Version

BARRIER MODULES

Background:

- 'Barriers' module provides a general-purpose barrier
  - Initializer only accepts the number of tasks participating with no notion of locality, which limits scalability
- 'AllLocalesBarriers' module provides a global singleton barrier between all locales
  - Has good scalability, but only suitable for SPMD-style codes

Open Discussions:

- Should we move all barriers to a new 'Collectives' module and create a single 'barrier' type? (#18861 / #18863)
- Should we keep 'check( )' and 'reset( )' methods? (#18862)
- Should we continue to support a waiting policy? (#18864)
- Should reusability be selectable? (#18865)

REFLECTION MODULE

Background: The Reflection module allows users to query properties about composite types

- e.g., fetch a field by ordinal position or name, or get the number of fields in a type

Actions Taken / Decisions Made:

- Deprecated formals for 'getField( )' family of functions in favor of a standard naming scheme (#18958)
  - Now uses 'idx', 'name', 'obj' instead of one-letter formal names
- Want to implement a new general-purpose function to resolve any expression
  - This function will replace the 'canResolve( )' family of functions
- Want to unify the 'getField( )' and 'getFieldRef( )' functions into a single function

Open Discussions:

- Should most Reflection functions be methods instead? (#17984)
  - Worry about polluting the global method namespace
- Should we drop the 'get' from most Reflection function names? (#18006)
- How should inherited fields be reported by 'numFields( )'? (#8736)

TYPES MODULE

Background:

- This module contains functions to query and modify types

Decisions Made:

- Move all ‘isXType’ and ‘isXValue’ query functions into ‘Types’
  - e.g., ‘isArrayType’, ‘isDmapValue’, ‘isRangeType’, ‘isSingleValue’, etc.
  - Since these are in internal modules, from a user’s perspective, this primarily affects where things are documented
- Remove type/subtype comparison operators (in favor of named functions)
  - For example: we don’t need the '<' operator on types because we already have ‘isProperSubtype’

Open Discussions:

- Should we have: ‘isXType’, ‘isXValue’, and ‘isX’ functions for each type ‘X’ or just one of these three? (#19361)
- Should we rename ‘isFloatType’/ ‘isFloatValue’/’isFloat’ to something else? (#19362)
  - There is no type named ‘float’ in Chapel (as there is in C), people may confuse this with ‘isReal’
- Should we have non-param ‘numBits’/‘numBytes’ functions? (#19364)

SYSCTYPES, CPTR, SYSBASIC MODULES

Background:

- Chapel has supported C type aliases for convenience, but spread across multiple modules:
  - 'SysCTypes': types whose representations are likely to vary between C compilers (e.g., 'c_int', 'size_t')
  - 'CPtr': types representing C pointers and fixed-size arrays (as well as related procedures)
  - 'SysBasic': types corresponding to C 'float' and 'double' as well as some system-oriented types ('off_t', 'mode_t', 'socklen_t')
- In practice, these features felt scattered, and therefore challenging to remember what lived where

Actions Taken:

- Created a new 'CTypes' module to serve as a central place for major C type aliases and related routines
  - replaces 'SysCTypes' and 'CPtr' while also including 'c_float' and 'c_double' from 'SysBasic'
  - started using 'c_' prefix more uniformly on such types (e.g., 'size_t' -> 'c_size_t')
- Moved other C types from 'SysBasic' to 'Sys'

Open Discussions:

- Should we make other changes to pointer-related features within 'CTypes'? (#18010, #18011, #18014, #18015, #18016, #18017)
- What should happen to the remaining features in 'SysBasic', which are mostly error-related?
  - e.g., can we merge with 'SysError' or 'Errors'?

CHAPELENV MODULE

Background:

- 'ChapelEnv' is an auto-'use'd module that exposes the 'CHPL_*' settings as 'param' values
  - e.g., CHPL_HOME, CHPL_COMM, CHPL_TASKS, CHPL_MEM, etc.
- Its name raised some concerns:
  - By convention, modules starting with 'Chapel' are typically not intended for end-users
  - "Env[ironment]" seemed misleading since the settings could be inferred or specified on the 'chpl' command-line
- It also seemed difficult to stabilize given that the 'CHPL_*' settings have grown and evolved frequently
- Auto-'use' seemed like overkill given the number of symbols it defines and how rarely they are used

Actions Taken:

- Deprecated the contents of the 'ChapelEnv' module, replacing it with a new non-auto-'use'd module 'ChplConfig'
  - new name reflects that it supports reasoning about the configuration of the 'chpl' compiler, whether set or inferred
- Characterized uses of the 'CHPL_*' variables in Chapel code
  - e.g., 'CHPL_COMM != "none"' is a common idiom used to determine whether we're compiling for multi-locale execution

Open Discussions:

- What user-facing queries could we support to replace raw 'CHPL_*' string comparisons? (#19188)

VECTORIZING ITERATOR MODULE

Background:

- 'VectorizingIterator' provided iterators that served to indicate a parallel loop should not introduce new tasks
  - flagged a loop as a candidate for vectorization, GPU-style parallelization, etc.

```
forall i in vectorizeOnly(1..n) do
```

  - have had lingering concerns about its definition, syntax, implementation, etc.
  - an auto-'use'd module
- Chapel 1.25.0 added a 'foreach' loop form as a language-based way of expressing similar information

Actions Taken / Decisions Made:

- Decided to deprecate 'VectorizingIterator'
- Deprecated all of its iterators in Chapel 1.26.0

Next Steps:

- deprecate the 'VectorizingIterator' module itself (left as a separate step due to its auto-'use')
- implement 'with-clauses' and shadow variables for 'foreach' loops (#18500)
- add support for 'foreach' expressions (#19336)

TIME MODULE

Background:

- One of our older modules, defining a ‘Timer’ type for timing things, a ‘sleep’ call, queries for current date, time
- Has some overlap with the newer, better-designed ‘DateTime’ module

Actions Taken / Decisions Made:

- Plan to rename ‘Timer’ to ‘stopwatch’ as a more accurate name and to extend its methods (#16393)
  - but where to store it? ‘Time’? ‘DateTime’? Combine them into one module? Add a new module?
- Conversations seem to be trending toward keeping two distinct modules:
  - ‘DateTime’ for reasoning about real-world dates and times
  - ‘???’ for taking timings, measuring the passage of time

Open Discussions:

- What should the name of this second module be?
  - ‘Time’: has the advantage of matching what Python, Rust, Go, Java call this; yet difficult to distinguish from ‘DateTime’
    - or should we rename ‘DateTime’ to something else, like simply ‘Dates’?
  - ‘Timers’: suggests things that measure time (more awkwardly: ’Timepieces’, ‘Chronometers’, ‘Chrono’)
- What should we call our monotonic clock query? (e.g., ‘now( )’, ‘clock( )’, ‘tic( )’, ‘tick( )’?)

DATETIME MODULE

Background:

- Support for representing dates, times, date + times, and timedeltas
- Heavily influenced by the Python module ‘datetime’
- Exposes some C relics such as 'struct tm' from <time.h>

Decisions Made:

- Hide/deprecate/"no doc" C procs like 'timetuple', 'strptime', and 'ctime' (#18833)
- Remove some ambiguous functions like 'operator datetime.-(dt: datetime, d: date): timedelta' (#18834)
- Chapel-ify names to fit the standard module style guide (#18846)
- Track whether a time is timezone-aware at the type level (#18941)

Open Discussions:

- Cleanly deprecating 'TZInfo' for timezones is a challenge
  - 'TZInfo' requires defining all timezones at compile time
    - loading from tzdata is impractical and won’t get updated until recompiled
  - Would like a replacement in hand to let users upgrade, but low priority at this point

BIGINTEGER MODULE

Background:

- Provides ‘bigint’ type for storing very large integers, and many methods and functions that use them

Actions Taken / Decisions Made:

- renamed 13 additional methods, for a total of 20 renamed methods and one renamed enum
- renamed arguments of 6 additional methods, for a total of 11 methods with renamed arguments
- updated the return value of 2 methods
- hid an additional implementation detail, for a total of 2
- added documentation to renamed symbols
- fixed some bugs and inconsistencies

Open Discussions:

- new 'round' enum name should be revisited—conflicts with 'Math.round( )'
- There are 17 other small library stabilization issues remaining that are likely uncontentious
  - See the list of issues
  - And 9 non-breaking changes

MATH MODULE

Background:

- Provides mathematical constants and functions, e.g., 'e', 'sqrt( )', 'gcd( )'
  - Names are usually based on C's interface, which was influenced by ISO standards
- Included in all programs by default

Actions Taken / Decisions Made:

- Decided to split into two modules, one that will still be auto-included and one that will need a 'use'/'import'

Open Discussions:

- What names should be used when splitting the module in two? (#18989)
- Which symbols should still be included by default? (#18990)
- How closely should the interface match C/the ISO standards? Lean seems to be "fairly closely", e.g.
  - How to name 'log' functions and related module-level constants? (#18995)
  - 'cproj' (#19011) and 'erf' (#19013) aren't self-explanatory, but have established meaning
- Rounding support is incomplete, should it be extended for 2.0? (#19024)

IO MODULE

Background and Actions Taken

Background:

- The IO module handles reading and writing to files, as well as formatted IO
  - 'write( )', 'writeln( )' and 'writef( )' are provided by default, all other IO functions are defined in the IO module
- Implements 'file' and 'channel' types
- This module is very large, ~7300 lines
- The IO module has several known API design issues (#7954)

Actions Taken:

- IO subteam performed initial review on most of the IO module and made proposals for Chapel 2.0
- Presented proposals to entire Chapel team for feedback and approval
- Began implementing some of the proposals (see next slide)

IO MODULE
Status

Completed:

- Deprecated the I/O style feature (#18501)
- Deprecated binary format strings, including endianness specifiers (#18503)

Pending:

- Rename I/O ‘channel’ type to ‘reader’ and ‘writer’ (#18112)
- Add an extensible Encoder/Decoder mechanism (#18499)
- Deprecate ‘j’ and ‘h’ format string specifiers in favor of Encoders/Decoders

IO MODULE

Open Discussion and Next Steps

Open Discussions:

- What should be done with the ‘iokind’ field on channels? (#19314)
- Resolve ‘readline’ vs ‘readln’ vs ‘read*line’ functionality (#19495)
- Clean up ‘read’ functionality (#19498)
- Replace ‘readstring’ and ‘readbytes’, mimic Python’s behavior (#18496)
- Deprecate ‘readwrite’, ‘readWriteLiteral’, and ‘readWriteNewline’ (#19500)
- Should ‘assertEOF’ be replaced with ‘atEOF’? (#19316)

Next Steps:

- Reach decisions on the open discussion items above
- Implement the Encoder/Decoder design
- Rename ‘channel’ to ‘reader’ and ‘writer’
- Review 'file' interface

MAP MODULE

Background:

- The Map module contains only the ‘map’ type
- A map is an unordered collection of key/value pairs

Actions Taken / Decisions Made:

- Made 'map.getValue( )' throw instead of halting when key is not present (#18786)
  - Additionally, added overload with sentinel value to return instead

- Decided to deprecate operators (#18493)
  - Removing old operator methods (=, ==, !=, +, +=, |, |=, &, &=, -, -=, ^, ^=)
  - Added by default when the module was created
  - Unneeded and unused

Open Discussions:

- Should parallel-safe and/or distributed collections be distinct types? (#18494)

SET MODULE

Background:

- The Set module contains only the ‘set’ type
- A set is a collection of unique, unordered, and unindexed elements

Actions Taken / Decisions Made:

- Deprecated 'set.isIntersecting( )' (#18796)
- Added arguments to initializers for more control over resizing (#18810)
- Updated some function argument names to use ‘element’ in favor of ‘x’ (#18797)
- Documented that the first argument takes precedence in set operations (#18842)
  - Overriding the ‘==‘ operator on a record can result in two elements that are not identical being considered equivalent
  - e.g., when intersecting two sets that contain '==‘-equivalent elements, the one from the LHS will be chosen

Open Discussions:

- Should parallel-safe and/or distributed collections be distinct types? (#18494)

SUBPROCESS MODULE

Background:

- provides a 'subprocess' type and methods for launching and communicating with subprocesses

Actions Taken / Decisions Made:

- renamed a method to better match naming conventions
  `subprocess.sendPosixSignal()`
- moved module-scope constants that name pipe styles into an enum
  `enum pipeStyle {…}`
- added two methods for sending specific signals to subprocesses
  `subprocess.abort(), subprocess.alarm()`

Other Comments:

- plan to deprecate POSIX signal names from 'Subprocess' and move them to ‘Sys.POSIX’
  – e.g. 'SIGALRM', 'SIGINT', 'SIGKILL'

RANDOM MODULE

Background:

- Provides two random number generators: NPB and PCG
- Provides a pseudo-interface named 'RandomStreamInterface'

Decisions Made:

- Keep only the PCG generator, move NPB generator to a package or test module
- Remove the pseudo-interface

Open Discussions:

- Is it possible to replace ‘iterate’ methods with ‘these’ iterators? (#19603)
- Name for the generator, e.g., just ‘Random’, and for the potential future random-generator interface (#19601)
- Semantic questions about the random number generator types (#19602)
- Should the ‘Random’ type be generic over the element type? (#19604)
  - Should they produce elements of other types? This is possible with the PCG generator
- Should we keep ‘getNth( )’ and ‘skipToNth( )’? if so, what are good names for them? (#19606)

REGEX MODULE

Background:

- The Regex module provides support for regular expressions based on Google's RE2 library

Actions Taken / Decisions Made:

- Renamed 'RegexMatch' fields 'size'→'numBytes' and 'offset'→'byteOffset' (#19076)
- Renamed arguments 'needle'→'pattern' and 'region'→'indices' (#18264)
  - Also updated methods on strings and bytes to use new names
- Should the Regex module define tertiary methods on string/bytes? (#18960, #17226)
  - Removed 'string.search' and 'bytes.search' methods with 'ignorecase' argument
  - Decided to replace 'search' with 'find', which returns a 'byteIndex' where a regular expression and string/byte match
  - Decided to remove 'matches' method
  - Decided to replace 'match' with 'startsWith', which returns true if a string/byte starts with a given regular expression
- Decided to use 'new regex("/a/")' instead of 'compile("/a/")', for compiling regular expressions (#17187)

Open Discussions:

- Deprecate and replace regex.sub and regex.subn (#19079)

VERSION MODULE

Background:

- Supports compile-time reasoning about version numbers for Chapel and Chapel programs
  - Introduced in 1.23

Actions Taken:

- Decided to add support for reasoning about versions at run-time
  - use case: Mason needs to evaluate versions for package dependencies it discovers at run-time

Open Discussions:

- How to provide a non-param type like 'sourceVersion' (#19201)
  - draft implementation (PR #19300)

STANDARD LIBRARY STABILIZATION

Next Steps

- Continue with our current process
  - Start reviewing remaining modules
  - Revisit modules that were first examined in previous releases
  - Continue resolving issues discussed in reviews, e.g.
    - Finalize design for serial, parallel, and distributed collections
    - How closely should the Math module interface match C/the ISO standards?
    - Finalize the Encoder/Decoder design
    - Finalize the division and naming of the DateTime and Time modules
- Develop a means of documenting the stability of a module (or language feature)

OTHER LIBRARY IMPROVEMENTS

OTHER LIBRARY IMPROVEMENTS

For a more complete list of library changes and improvements in the 1.25.1 and 1.26.0 releases, refer to the following sections in the CHANGES.md file:

- 'Name Changes in Libraries'
- 'Deprecated / Removed Library Features'
- 'Standard Library Modules'
- 'Package Modules'
- 'Documentation'
- 'Error Messages / Semantic Checks'
- 'Bug Fixes for Libraries'


THANK YOU
https://chapel-lang.org
@ChapelLanguage