Towards Stability in the Chapel Language

Michael Ferguson

CHIUW 2020

May 22, 2020

michael.ferguson@hpe.com

chapel-lang.org

@ChapelLanguage

CRAY®
a Hewlett Packard Enterprise company

CRAY CHAPEL

Language Stability

- *Language stability* is a feature of a programming language
  - guarantee that valid programs will continue to function

- Without language stability, programmers need to update code for each release
  - all this added work can reduce any productivity benefit the language offers

- Languages in wide use have two strategies for language stability:
  1. Don't change the language in a way that breaks existing programs
  2. Provide versions of the language (e.g. C99 or Python 3)

Language Versioning

- Providing versions of the language doesn't entirely solve the problem
    - programmers still need to update when migrating to the new version
    - the old version might eventually become unsupported
        - e.g. Python 2 is reaching end of life

- Language versioning does give programmers more control over when to update

- Some compilers can even apply newer optimization to older standards
    - e.g. C compilers with flags like --std=c99

Some History

- The Chapel language started development as a research prototype
  - Started in 2003 - first public release was 0.8 in 2008

- Initial focus was to demonstrate key differentiating features
  - productive parallel and distributed programming
  - user-defined distributions

- Over time Chapel has become significantly more usable and performant
  - and the user community has grown

- Language stability is now important to Chapel users

Language Stability for Chapel

- In the past several years, we have been working towards language stability
  - Sometimes refer to the stable language version as "Chapel 2.0"

- Once the language is stable, the project will
  - commit to not breaking a set of core language features
  - adopt semantic versioning

Why is Language Stability Challenging?

- Could the project have just declared Chapel 1.14 as stable?
  - and committed not to change core features past that point?

- Doing so would have prevented addressing changes requested by users

- Stabilizing a language to soon leads to obvious problems never being fixed
  - e.g. Makefiles and tab characters

- There is a balance between addressing requests and stabilizing

Changes Requested by Users

- Language support for error handling
- Leak-free use of classes without needing to call 'delete'
- Classes that cannot store nil by default
- Robust class and record initializer support
- Language design that minimizes unnecessary copies and memory errors
- A package manager enabling the community to share libraries
- Support for Unicode strings
- Make the built-ins either 1-based or 0-based, not a mix of the two

Known Language Problems Over Time

	1.14
error handling	🟥
delete-free programming	🟥
class types and nilability	
initializers	🟥
initialization, deinitialization & intents	🟫
package manager	🟥
modules and visibility	🟨
Unicode strings	🟥
arrays and loop expressions	🟫
function overload resolution	🟧
override and overload set checking	
	2016

🟫 serious defects 🟥 unsolved problems 🟧 partially solved 🟨 mostly stable 🟩 stable

Known Language Problems Over Time

	1.14	1.15
error handling	🟥	
delete-free programming	🟥	🟧
class types and nilability		
initializers	🟥	
initialization, deinitialization & intents	🟫	
package manager	🟥	
modules and visibility	🟨	
Unicode strings	🟥	
arrays and loop expressions	🟫	
function overload resolution	🟧	
override and overload set checking		
	2016	2017

- New types Owned and Shared types lead to realization that class types should have nilable and non-nilable variants
- Partly due to issues of ownership transfer

🟫 serious defects 🟥 unsolved problems 🟧 partially solved 🟨 mostly stable 🟩 stable

Known Language Problems Over Time

	1.14	1.15
error handling	🟥	
delete-free programming	🟥	🟧
class types and nilability		🟥
initializers	🟥	
initialization, deinitialization & intents	🟫	
package manager	🟥	
modules and visibility	🟨	
Unicode strings	🟥	
arrays and loop expressions	🟫	
function overload resolution	🟧	
override and overload set checking		
	2016	2017

- New types Owned and Shared types lead to realization that class types should have nilable and non-nilable variants
- Partly due to issues of ownership transfer

🟫 serious defects 🟥 unsolved problems 🟧 partially solved 🟨 mostly stable 🟩 stable

Owned in 1.15

- Owned example from 1.15:

```
var myOwned      = new Owned(new MyClass());

var otherOwned = myOwned;
// anotherOwned now stores nil
// both assignment and copy-initialization transfer ownership
```

- Ownership transfer adds a new way for variables to become 'nil'
  - increasing the chance of 'nil' dereference errors
  - Can these be caught at compile-time?

Known Language Problems Over Time

	1.14	1.15
error handling	🟥	
delete-free programming	🟥	🟧
class types and nilability		🟥
initializers	🟥	
initialization, deinitialization & intents	🟫	
package manager	🟥	
modules and visibility	🟨	
Unicode strings	🟥	
arrays and loop expressions	🟫	
function overload resolution	🟧	
override and overload set checking		
	2016	2017

🟫 serious defects 🟥 unsolved problems 🟧 partially solved 🟨 mostly stable 🟩 stable

Known Language Problems Over Time

	1.14	1.15
error handling	🟥	🟧
delete-free programming	🟥	🟧
class types and nilability		🟥
initializers	🟥	🟧
initialization, deinitialization & intents	🟫	🟥
package manager	🟥	🟥
modules and visibility	🟨	🟨
Unicode strings	🟥	🟥
arrays and loop expressions	🟫	🟥
function overload resolution	🟧	🟧
override and overload set checking		
	2016	2017

🟫 serious defects 🟥 unsolved problems 🟧 partially solved 🟨 mostly stable 🟩 stable

Known Language Problems Over Time

	1.14	1.15	1.16
error handling	🟥	🟧	🟨
delete-free programming	🟥	🟧	🟧
class types and nilability		🟥	🟥
initializers	🟥	🟧	🟧
initialization, deinitialization & intents	🟫	🟥	🟥
package manager	🟥	🟥	🟧
modules and visibility	🟨	🟨	🟨
Unicode strings	🟥	🟥	🟥
arrays and loop expressions	🟫	🟥	🟥
function overload resolution	🟧	🟧	🟧
override and overload set checking			🟥
	2016	2017	2017

- CHIP 20 described "hijacking" scenarios

🟫 serious defects	🟥 unsolved problems	🟧 partially solved	🟨 mostly stable	🟩 stable

Known Language Problems Over Time

	1.14	1.15	1.16
error handling	🟥	🟧	🟨
delete-free programming	🟥	🟧	🟧
class types and nilability		🟥	🟥
initializers	🟥	🟧	🟧
initialization, deinitialization & intents	🟫	🟥	🟥
package manager	🟥	🟥	🟧
modules and visibility	🟨	🟨	🟨
Unicode strings	🟥	🟥	🟥
arrays and loop expressions	🟫	🟥	🟥
function overload resolution	🟧	🟧	🟧
override and overload set checking			🟥
	2016	2017	2017

- Users of new 'mason' manager in 1.16 demonstrated problems with module system

🟫 serious defects 🟥 unsolved problems 🟧 partially solved 🟨 mostly stable 🟩 stable

Known Language Problems Over Time

	1.14	1.15	1.16	
error handling	🟥	🟧	🟨	
delete-free programming	🟥	🟧	🟧	
class types and nilability		🟥	🟥	
initializers	🟥	🟧	🟧	
initialization, deinitialization & intents	🟫	🟥	🟥	
package manager	🟥	🟥	🟧	
modules and visibility	🟨	🟨	🟨	🟥
Unicode strings	🟥	🟥	🟥	
arrays and loop expressions	🟫	🟥	🟥	
function overload resolution	🟧	🟧	🟧	
override and overload set checking			🟥	
	2016	2017	2017	

- Users of new 'mason' manager in 1.16 demonstrated problems with module system

🟫 serious defects 🟥 unsolved problems 🟧 partially solved 🟨 mostly stable 🟩 stable

Module Issue Revealed by Mason

file hierarchy:

```
.
├── AB.chpl
├── A/
│   ├── A.chpl
│   └── Help.chpl
└── B/
    ├── B.chpl
    └── Help.chpl
```

```
// AB.chpl
use A;
use B;
```

```
> chpl AB.chpl A/A.chpl B/B.chpl
warning: Ambiguous module source file
-- using A/C.chpl over B/C.chpl
```

```
// A.chpl
use Help; // expecting A/Help.chpl
```

```
// B.chpl
use Help; // expecting B/Help.chpl
```

- Module system needs to help distinguish local and global modules, somehow

Known Language Problems Over Time

	1.14	1.15	1.16	
error handling	🟥	🟧	🟨	
delete-free programming	🟥	🟧	🟧	
class types and nilability		🟥	🟥	
initializers	🟥	🟧	🟧	
initialization, deinitialization & intents	🟫	🟥	🟥	
package manager	🟥	🟥	🟧	
modules and visibility	🟨	🟨	🟨	🟥
Unicode strings	🟥	🟥	🟥	
arrays and loop expressions	🟫	🟥	🟥	
function overload resolution	🟧	🟧	🟧	
override and overload set checking			🟥	
	2016	2017	2017	

🟫 serious defects 🟥 unsolved problems 🟧 partially solved 🟨 mostly stable 🟩 stable

Known Language Problems Over Time

	1.14	1.15	1.16	1.17
error handling	🟥	🟧	🟨	🟨
delete-free programming	🟥	🟧	🟧	🟧
class types and nilability		🟥	🟥	🟥
initializers	🟥	🟧	🟧	🟧
initialization, deinitialization & intents	🟫	🟥	🟥	🟧
package manager	🟥	🟥	🟧	🟧
modules and visibility	🟨	🟨	🟨	🟥
Unicode strings	🟥	🟥	🟥	🟥
arrays and loop expressions	🟫	🟥	🟥	🟥
function overload resolution	🟧	🟧	🟧	🟧
override and overload set checking			🟥	🟥
	2016	2017	2017	2018

🟫 serious defects 🟥 unsolved problems 🟧 partially solved 🟨 mostly stable 🟩 stable

Known Language Problems Over Time

CRAY
a Hewlett Packard Enterprise company

	1.14	1.15	1.16	1.17	1.18
error handling	🟥	🟧	🟨	🟨	
delete-free programming	🟥	🟧	🟧	🟧	🟨 x
class types and nilability		🟥	🟥	🟥	
initializers	🟥	🟧	🟧	🟧	🟨 x
initialization, deinitialization & intents	🟫	🟥	🟥	🟧	
package manager	🟥	🟥	🟧	🟧	
modules and visibility	🟨	🟨	🟨	🟥	
Unicode strings	🟥	🟥	🟥	🟥	
arrays and loop expressions	🟫	🟥	🟥	🟥	
function overload resolution	🟧	🟧	🟧	🟧	
override and overload set checking			🟥	🟥	
	2016	2017	2017	2018	2018

- Big breaking changes
  - initializers replace constructors
  - managed class types

🟫	🟥	🟧	🟨	🟩
serious defects	unsolved problems	partially solved	mostly stable	stable

Initializers Replaced Constructors in 1.18

```
record Point {
  var x, y: real;
  proc Point(x: real, y: real) {
    this.x = x;
    this.y = y;
  }
}
var p = new Point(1.0, 2.0);
```

```
record Point {
  var x, y: real;
  proc init(x: real, y: real) {
    this.x = x;
    this.y = y;
  }
}
var p = new Point(1.0, 2.0);
```

- Initializers significantly more robust and capable than constructors

Managed Class Types in 1.18

```
class C {
  var x: int;
}
proc main() {
  var instance = new C(1);
  var tmp: C = instance;
  delete instance;
}
```

```
class C {
  var x: int;
}
proc main() {
  var instance = new owned C(1);
  var tmp: borrowed C = instance;
  // instance automatically deleted here
}
```

- Generally removed need for 'delete'
- Some memory errors are now caught at compile-time

Known Language Problems Over Time

	1.14	1.15	1.16	1.17	1.18
error handling	🟥	🟧	🟨	🟨	
delete-free programming	🟥	🟧	🟧	🟧	🟨 x
class types and nilability		🟥	🟥	🟥	
initializers	🟥	🟧	🟧	🟧	🟨 x
initialization, deinitialization & intents	🟫	🟥	🟥	🟧	
package manager	🟥	🟥	🟧	🟧	
modules and visibility	🟨	🟨	🟨	🟥	
Unicode strings	🟥	🟥	🟥	🟥	
arrays and loop expressions	🟫	🟥	🟥	🟥	
function overload resolution	🟧	🟧	🟧	🟧	
override and overload set checking			🟥	🟥	
	2016	2017	2017	2018	2018

🟫 serious defects	🟥 unsolved problems	🟧 partially solved	🟨 mostly stable	🟩 stable

Known Language Problems Over Time

	1.14	1.15	1.16	1.17	1.18
error handling	🟥	🟧	🟨	🟨	
delete-free programming	🟥	🟧	🟧	🟧	🟨 x
class types and nilability		🟥	🟥	🟥	
initializers	🟥	🟧	🟧	🟧	🟨 x
initialization, deinitialization & intents	🟫	🟥	🟥	🟧	
package manager	🟥	🟥	🟧	🟧	
modules and visibility	🟨	🟨	🟨	🟥	
Unicode strings	🟥	🟥	🟥	🟥	
arrays and loop expressions	🟫	🟥	🟥	🟥	
function overload resolution	🟧	🟧	🟧	🟧	
override and overload set checking			🟥	🟥	
	2016	2017	2017	2018	2018

- Errors are classes
  - So class changes caused problems for error handling

🟫 serious defects 🟥 unsolved problems 🟧 partially solved 🟨 mostly stable 🟩 stable

Known Language Problems Over Time

	1.14	1.15	1.16	1.17	1.18
error handling	🟥	🟧	🟨	🟨	🟥
delete-free programming	🟥	🟧	🟧	🟧	🟨 x
class types and nilability		🟥	🟥	🟥	
initializers	🟥	🟧	🟧	🟧	🟨 x
initialization, deinitialization & intents	🟫	🟥	🟥	🟧	
package manager	🟥	🟥	🟧	🟧	
modules and visibility	🟨	🟨	🟨	🟥	
Unicode strings	🟥	🟥	🟥	🟥	
arrays and loop expressions	🟫	🟥	🟥	🟥	
function overload resolution	🟧	🟧	🟧	🟧	
override and overload set checking			🟥	🟥	
	2016	2017	2017	2018	2018

- Errors are classes
  - So class changes caused problems for error handling

🟫 serious defects 🟥 unsolved problems 🟧 partially solved 🟨 mostly stable 🟩 stable

Error Handling Problem in 1.18

```
proc f() throws {
  throw new InvalidArgumentError();
}

try {
  f();
} catch e: InvalidArgumentError {
  throw new WrappedError(e); // led to double-free
}
```

undecorated new is
'new borrowed'
and can't be returned?

if 'e' is a borrowed Error,
how can I transfer ownership?

Known Language Problems Over Time

	1.14	1.15	1.16	1.17	1.18
error handling	🟥	🟧	🟨	🟨	🟥
delete-free programming	🟥	🟧	🟧	🟧	🟨 x
class types and nilability		🟥	🟥	🟥	
initializers	🟥	🟧	🟧	🟧	🟨 x
initialization, deinitialization & intents	🟫	🟥	🟥	🟧	
package manager	🟥	🟥	🟧	🟧	
modules and visibility	🟨	🟨	🟨	🟥	
Unicode strings	🟥	🟥	🟥	🟥	
arrays and loop expressions	🟫	🟥	🟥	🟥	
function overload resolution	🟧	🟧	🟧	🟧	
override and overload set checking			🟥	🟥	
	2016	2017	2017	2018	2018

🟫 serious defects 🟥 unsolved problems 🟧 partially solved 🟨 mostly stable 🟩 stable

Known Language Problems Over Time

	1.14	1.15	1.16	1.17	1.18
error handling	🟥	🟧	🟨	🟨	🟥
delete-free programming	🟥	🟧	🟧	🟧	🟨 x
class types and nilability		🟥	🟥	🟥	🟥
initializers	🟥	🟧	🟧	🟧	🟨 x
initialization, deinitialization & intents	🟫	🟥	🟥	🟧	🟧
package manager	🟥	🟥	🟧	🟧	🟧
modules and visibility	🟨	🟨	🟨	🟥	🟥
Unicode strings	🟥	🟥	🟥	🟥	🟧
arrays and loop expressions	🟫	🟥	🟥	🟥	🟧
function overload resolution	🟧	🟧	🟧	🟧	🟧
override and overload set checking			🟥	🟥	🟨
	2016	2017	2017	2018	2018

🟫 **serious defects** 🟥 **unsolved problems** 🟧 **partially solved** 🟨 **mostly stable** 🟩 **stable**

Known Language Problems Over Time

a Hewlett Packard Enterprise company

	1.14	1.15	1.16	1.17	1.18	1.19
error handling	🟥	🟧	🟨	🟨	🟥	🟩
delete-free programming	🟥	🟧	🟧	🟧	🟨 x	🟨
class types and nilability		🟥	🟥	🟥	🟥	🟥
initializers	🟥	🟧	🟧	🟧	🟨 x	🟩
initialization, deinitialization & intents	🟫	🟥	🟥	🟧	🟧	🟧
package manager	🟥	🟥	🟧	🟧	🟧	🟩
modules and visibility	🟨	🟨	🟨	🟥	🟥	🟥
Unicode strings	🟥	🟥	🟥	🟥	🟧	🟧
arrays and loop expressions	🟫	🟥	🟥	🟥	🟧	🟧
function overload resolution	🟧	🟧	🟧	🟧	🟧	🟧
override and overload set checking			🟥	🟥	🟨	🟨
	2016	2017	2017	2018	2018	2019

🟫 serious defects 🟥 unsolved problems 🟧 partially solved 🟨 mostly stable 🟩 stable

Known Language Problems Over Time

	1.14	1.15	1.16	1.17	1.18	1.19	1.20
error handling	🟥	🟧	🟨	🟨	🟥	🟩	
delete-free programming	🟥	🟧	🟧	🟧	🟨 x	🟨	
class types and nilability		🟥	🟥	🟥	🟥	🟥	🟨 x
initializers	🟥	🟧	🟧	🟧	🟨	🟩	
initialization, deinitialization & intents	🟫	🟥				🟧	
package manager	🟥	🟥				🟩	
modules and visibility	🟨	🟨				🟥	
Unicode strings	🟥	🟥				🟧	
arrays and loop expressions	🟫	🟥				🟧	
function overload resolution	🟧	🟧				🟧	
override and overload set checking			🟥	🟥	🟨	🟨	
	2016	2017	2017	2018	2018	2019	2019

- Big breaking change:
  - Class types cannot store 'nil' by default

🟫 serious defects 🟥 unsolved problems 🟧 partially solved 🟨 mostly stable 🟩 stable

Non-Nilable Class Types in 1.20

```
class C {
  var x: int;
}
var a: borrowed C = ...;

var b: borrowed C = nil; // now an error in 1.20
var c: borrowed C; // now an error in 1.20
a = nil; // now an error in 1.20

var bb: borrowed C? = nil; // OK in 1.20, C? is a nilable class type
```

- Helps discover more errors at compile time and make safer code the default

Known Language Problems Over Time

CRAY
a Hewlett Packard Enterprise company

	1.14	1.15	1.16	1.17	1.18	1.19	1.20
error handling	🟥	🟧	🟨	🟨	🟥	🟩	
delete-free programming	🟥	🟧	🟧	🟧	🟨 x	🟨	
class types and nilability		🟥	🟥	🟥	🟥	🟥	🟨 x
initializers	🟥	🟧	🟧	🟧	🟨 x	🟩	
initialization, deinitialization & intents	🟫	🟥	🟥	🟧	🟧	🟧	
package manager	🟥	🟥	🟧	🟧	🟧	🟩	
modules and visibility	🟨	🟨	🟨	🟥	🟥	🟥	
Unicode strings	🟥	🟥	🟥	🟥	🟧	🟧	
arrays and loop expressions	🟫	🟥	🟥	🟥	🟧	🟧	
function overload resolution	🟧	🟧	🟧	🟧	🟧	🟧	
override and overload set checking			🟥	🟥	🟨	🟨	
	2016	2017	2017	2018	2018	2019	2019

🟫 serious defects | 🟥 unsolved problems | 🟧 partially solved | 🟨 mostly stable | 🟩 stable

Known Language Problems Over Time

	1.14	1.15	1.16	1.17	1.18	1.19	1.20
error handling	🟥	🟧	🟨	🟨	🟥	🟩	
delete-free programming	🟥	🟧	🟧	🟧	🟨 x	🟨	
class types and nilability		🟥	🟥	🟥	🟥	🟥	🟨 x
initializers	🟥	🟧	🟧	🟧	🟨 x	🟩	
initialization, deinitialization & intents	🟫	🟥	🟥	🟧	🟧	🟧	
package manager	🟥	🟥	🟧	🟧	🟧	🟩	
modules and visibility	🟨	🟨	🟨			🟥	
Unicode strings	🟥	🟥	🟥			🟧	
arrays and loop expressions	🟫	🟥	🟥			🟧	
function overload resolution	🟧	🟧	🟧			🟧	
override and overload set checking			🟥			🟨	
	2016	2017	20			9	2019

- Non-nilable class types cannot be default initialized
- Non-nilable owned cannot be copy-initialized

🟫 serious defects 🟥 unsolved problems 🟧 partially solved 🟨 mostly stable 🟩 stable

Known Language Problems Over Time

CRAY
a Hewlett Packard Enterprise company

	1.14	1.15	1.16	1.17	1.18	1.19	1.20
error handling	🟥	🟧	🟨	🟨	🟥	🟩	
delete-free programming	🟥	🟧	🟧	🟧	🟨 x	🟨	
class types and nilability		🟥	🟥	🟥	🟥	🟥	🟨 x
initializers	🟥	🟧	🟧	🟧	🟨 x	🟩	
initialization, deinitialization & intents	🟫	🟥	🟥	🟧	🟧	🟧	🟥
package manager	🟥	🟥	🟧	🟧	🟧	🟩	
modules and visibility	🟨	🟨	🟨			🟥	
Unicode strings	🟥	🟥	🟥			🟧	
arrays and loop expressions	🟫	🟥	🟥			🟧	
function overload resolution	🟧	🟧	🟧			🟧	
override and overload set checking			🟥			🟨	
	2016	2017	20			9	2019

- Non-nilable class types cannot be default initialized
- Non-nilable owned cannot be copy-initialized

🟫 serious defects 🟥 unsolved problems 🟧 partially solved 🟨 mostly stable 🟩 stable

Non-Nilable Initialization in 1.20

```
var x: owned MyClass;
try {
  var arg = ...;
  ... lots of code setting arg ...
  x = new MyClass(arg);
}
```

- Needed a way to write such patterns without needing nilable variables

Non-Nilable Ownership Transfer in 1.20

```
var x: owned MyClass = new MyClass(1);
var y = x; // ownership transfer... now x stores 'nil'
           // but x's type is not a nilable class type!
```

- Needed a way to address this type system gap
  - preferably, without completely prohibiting such patterns

Known Language Problems Over Time

	1.14	1.15	1.16	1.17	1.18	1.19	1.20
error handling	🟥	🟧	🟨	🟨	🟥	🟩	
delete-free programming	🟥	🟧	🟧	🟧	🟨 x	🟨	
class types and nilability		🟥	🟥	🟥	🟥	🟥	🟨 x
initializers	🟥	🟧	🟧	🟧	🟨 x	🟩	
initialization, deinitialization & intents	🟫	🟥	🟥	🟧	🟧	🟧	🟥
package manager	🟥	🟥	🟧	🟧	🟧	🟩	
modules and visibility	🟨	🟨	🟨	🟥	🟥	🟥	
Unicode strings	🟥	🟥	🟥	🟥	🟧	🟧	
arrays and loop expressions	🟫	🟥	🟥	🟥	🟧	🟧	
function overload resolution	🟧	🟧	🟧	🟧	🟧	🟧	
override and overload set checking			🟥	🟥	🟨	🟨	
	2016	2017	2017	2018	2018	2019	2019

🟫 serious defects 🟥 unsolved problems 🟧 partially solved 🟨 mostly stable 🟩 stable

Known Language Problems Over Time

	1.14	1.15	1.16	1.17	1.18	1.19	1.20
error handling	🟥	🟧	🟨	🟨	🟥	🟩	🟩
delete-free programming	🟥	🟧	🟧	🟧	🟨 X	🟨	🟩
class types and nilability		🟥	🟥	🟥	🟥	🟥	🟨 X
initializers	🟥	🟧	🟧	🟧	🟨 X	🟩	🟩
initialization, deinitialization & intents	🟫	🟥	🟥	🟧	🟧	🟧	🟥
package manager	🟥	🟥	🟧	🟧	🟧	🟩	🟩
modules and visibility	🟨	🟨	🟨	🟥	🟥	🟥	🟧
Unicode strings	🟥	🟥	🟥	🟥	🟧	🟧	🟨
arrays and loop expressions	🟫	🟥	🟥	🟥	🟧	🟧	🟨
function overload resolution	🟧	🟧	🟧	🟧	🟧	🟧	🟨
override and overload set checking			🟥	🟥	🟨	🟨	🟩
	2016	2017	2017	2018	2018	2019	2019

🟫 serious defects 🟥 unsolved problems 🟧 partially solved 🟨 mostly stable 🟩 stable

Known Language Problems Over Time

	1.14	1.15	1.16	1.17	1.18	1.19	1.20	1.21
error handling	🟥	🟧	🟨	🟨	🟥	🟩	🟩	
delete-free programming	🟥	🟧	🟧	🟧	🟨 x	🟨	🟩	
class types and nilability		🟥	🟥	🟥	🟥	🟥	🟨 x	
initializers	🟥	🟧	🟧	🟧	🟨 x	🟩	🟩	
initialization, deinitialization & intents	🟫	🟥	🟥	🟧	🟧	🟧	🟥	
package manager	🟥	🟥	🟧	🟧	🟧	🟩	🟩	
modules and visibility	🟨	🟨	🟨	🟥	🟥	🟥	🟧	🟩 x
Unicode strings	🟥	🟥	🟥	🟥	🟧	🟧	🟨	
arrays and loop expressions	🟫	🟥	🟥				🟨	
function overload resolution	🟧	🟧	🟧				🟨	
override and overload set checking			🟥	🟥	🟨	🟨	🟩	
	2016	2017	2017	2018	2018	2019	2019	2020

- Breaking change:
  - Module visibility is much tighter

🟫 serious defects 🟥 unsolved problems 🟧 partially solved 🟨 mostly stable 🟩 stable

Module System Improvements

- Import statements are new and support a more resilient coding style:

```
import MyModule;
writeln(MyModule.sym1);  // Enabled by the 'import'
writeln(sym1);           // Not enabled, won’t work
```

- 'use' statements are private by default
- 'use' and 'import' statements can request relative module paths

```
use this.Submodule;         // Uses module defined within current module
use super.SiblingModule;    // Uses module defined in parent module
```

- These improvements support mason packages
  - by considering the possibility of module name collisions across packages

Known Language Problems Over Time

	1.14	1.15	1.16	1.17	1.18	1.19	1.20	1.21
error handling	🟥	🟧	🟨	🟨	🟥	🟩	🟩	
delete-free programming	🟥	🟧	🟧	🟧	🟨 x	🟨	🟩	
class types and nilability		🟥	🟥	🟥	🟥	🟥	🟨 x	
initializers	🟥	🟧	🟧	🟧	🟨 x	🟩	🟩	
initialization, deinitialization & intents	🟫	🟥	🟥	🟧	🟧	🟧	🟥	
package manager	🟥	🟥	🟧	🟧	🟧	🟩	🟩	
modules and visibility	🟨	🟨	🟨	🟥	🟥	🟥	🟧	🟩 x
Unicode strings	🟥	🟥	🟥	🟥	🟧	🟧	🟨	
arrays and loop expressions	🟫	🟥	🟥	🟥	🟧	🟧	🟨	
function overload resolution	🟧	🟧	🟧	🟧	🟧	🟧	🟨	
override and overload set checking			🟥	🟥	🟨	🟨	🟩	
	2016	2017	2017	2018	2018	2019	2019	2020

🟫 serious defects 🟥 unsolved problems 🟧 partially solved 🟨 mostly stable 🟩 stable

Known Language Problems Over Time

	1.14	1.15	1.16	1.17	1.18	1.19	1.20	1.21
error handling	🟥	🟧	🟨	🟨	🟥	🟩	🟩	🟩
delete-free programming	🟥	🟧	🟧	🟧	🟨 x	🟨	🟩	🟩
class types and nilability		🟥	🟥	🟥	🟥	🟥	🟨 x	🟩
initializers	🟥	🟧	🟧	🟧	🟨 x	🟩	🟩	🟩
initialization, deinitialization & intents	🟫	🟥	🟥	🟧	🟧	🟧	🟥	🟩
package manager	🟥	🟥	🟧	🟧	🟧	🟩	🟩	🟩
modules and visibility	🟨	🟨	🟨	🟥	🟥	🟥	🟧	🟩 x
Unicode strings	🟥	🟥	🟥	🟥	🟧	🟧	🟨	🟩
arrays and loop expressions	🟫	🟥	🟥	🟥	🟧	🟧	🟨	🟨
function overload resolution	🟧	🟧	🟧	🟧	🟧	🟧	🟨	🟨
override and overload set checking			🟥	🟥	🟨	🟨	🟩	🟩
	2016	2017	2017	2018	2018	2019	2019	2020

🟫 serious defects 🟥 unsolved problems 🟧 partially solved 🟨 mostly stable 🟩 stable

0-based indexing

- We polled Chapel users about switching to 0-based indexing
  - Most said they would prefer it, if we were designing the language from scratch
  - Most were not terribly concerned about updating their existing Chapel code
  - Most expressed concern about the expected impact to other users
- After studying the impact on key codes, we decided to switch to 0-based
- Did so in a separate release - 1.22 - to make migrating code easier

- Change impacts many types: tuples, strings, bytes, array literals, lists, ...

```
var t = (1.2, 3.4);   // t(1) was 1.2; it's now 3.4, and t(0) is 1.2
```

Changes Requested by Users

- Language support for error handling
- Leak-free use of classes without needing to call 'delete'
- Classes that cannot store nil by default
- Robust class and record initializer support
- Language design that minimizes unnecessary copies and memory errors
- A package manager enabling the community to share libraries
- Support for Unicode strings
- Make the built-ins either 1-based or 0-based, not a mix of the two

Changes Requested by Users

- Language support for error handling
- Leak-free use of classes without needing to call 'delete'
- Classes that cannot store nil by default
- Robust class and record initializer support
- Language design that minimizes unnecessary copies and memory errors
- A package manager enabling the community to share libraries
- Support for Unicode strings
- Make the built-ins either 1-based or 0-based, not a mix of the two
- Have not needed major changes to the unique features of Chapel:
  - parallelism and distributed programming

Stabilization Next Steps

- Adding constrained generics
- Addressing problems with point-of-instantiation in function resolution
- Improving array initialization
  - currently does default initialization + assignment instead of copy initialization

- Stabilizing the standard libraries

- The 1.21/1.22 release notes have much more detail
  - about problems addressed in 1.21 to support language stability
  - about areas not yet included in stabilization

Questions?

a Hewlett Packard Enterprise company

	1.14	1.15	1.16	1.17	1.18	1.19	1.20	1.21
error handling	🟥	🟧	🟨	🟨	🟥	🟩	🟩	🟩
delete-free programming	🟥	🟧	🟧	🟧	🟨 x	🟨	🟩	🟩
class types and nilability		🟥	🟥	🟥	🟥	🟥	🟨 x	🟩
initializers	🟥	🟧	🟧	🟧	🟨 x	🟩	🟩	🟩
initialization, deinitialization & intents	🟫	🟥	🟥	🟧	🟧	🟧	🟥	🟩
package manager	🟥	🟥	🟧	🟧	🟧	🟩	🟩	🟩
modules and visibility	🟨	🟨	🟨	🟥	🟥	🟥	🟧	🟩 x
Unicode strings	🟥	🟥	🟥	🟥	🟧	🟧	🟨	🟩
arrays and loop expressions	🟫	🟥	🟥	🟥	🟧	🟧	🟨	🟨
function overload resolution	🟧	🟧	🟧	🟧	🟧	🟧	🟨	🟨
override and overload set checking			🟥	🟥	🟨	🟨	🟩	🟩
	2016	2017	2017	2018	2018	2019	2019	2020

🟫 serious defects 🟥 unsolved problems 🟧 partially solved 🟨 mostly stable 🟩 stable

FORWARD LOOKING STATEMENTS

This presentation may contain forward-looking statements that involve risks, uncertainties and assumptions. If the risks or uncertainties ever materialize or the assumptions prove incorrect, the results of Hewlett Packard Enterprise Company and its consolidated subsidiaries ("Hewlett Packard Enterprise") may differ materially from those expressed or implied by such forward-looking statements and assumptions. All statements other than statements of historical fact are statements that could be deemed forward-looking statements, including but not limited to any statements regarding the expected benefits and costs of the transaction contemplated by this presentation; the expected timing of the completion of the transaction; the ability of HPE, its subsidiaries and Cray to complete the transaction considering the various conditions to the transaction, some of which are outside the parties' control, including those conditions related to regulatory approvals; projections of revenue, margins, expenses, net earnings, net earnings per share, cash flows, or other financial items; any statements concerning the expected development, performance, market share or competitive performance relating to products or services; any statements regarding current or future macroeconomic trends or events and the impact of those trends and events on Hewlett Packard Enterprise and its financial performance; any statements of expectation or belief; and any statements of assumptions underlying any of the foregoing. Risks, uncertainties and assumptions include the possibility that expected benefits of the transaction described in this presentation may not materialize as expected; that the transaction may not be timely completed, if at all; that, prior to the completion of the transaction, Cray's business may not perform as expected due to transaction-related uncertainty or other factors; that the parties are unable to successfully implement integration strategies; the need to address the many challenges facing Hewlett Packard Enterprise's businesses; the competitive pressures faced by Hewlett Packard Enterprise's businesses; risks associated with executing Hewlett Packard Enterprise's strategy; the impact of macroeconomic and geopolitical trends and events; the development and transition of new products and services and the enhancement of existing products and services to meet customer needs and respond to emerging technological trends; and other risks that are described in our Fiscal Year 2018 Annual Report on Form 10-K, and that are otherwise described or updated from time to time in Hewlett Packard Enterprise's other filings with the Securities and Exchange Commission, including but not limited to our subsequent Quarterly Reports on Form 10-Q. Hewlett Packard Enterprise assumes no obligation and does not intend to update these forward-looking statements.

THANK YOU

QUESTIONS?

✉ chapel_info@cray.com

@ChapelLanguage

chapel-lang.org

CRAY®

a Hewlett Packard Enterprise company

cray.com

@cray_inc

linkedin.com/company/cray-inc-/