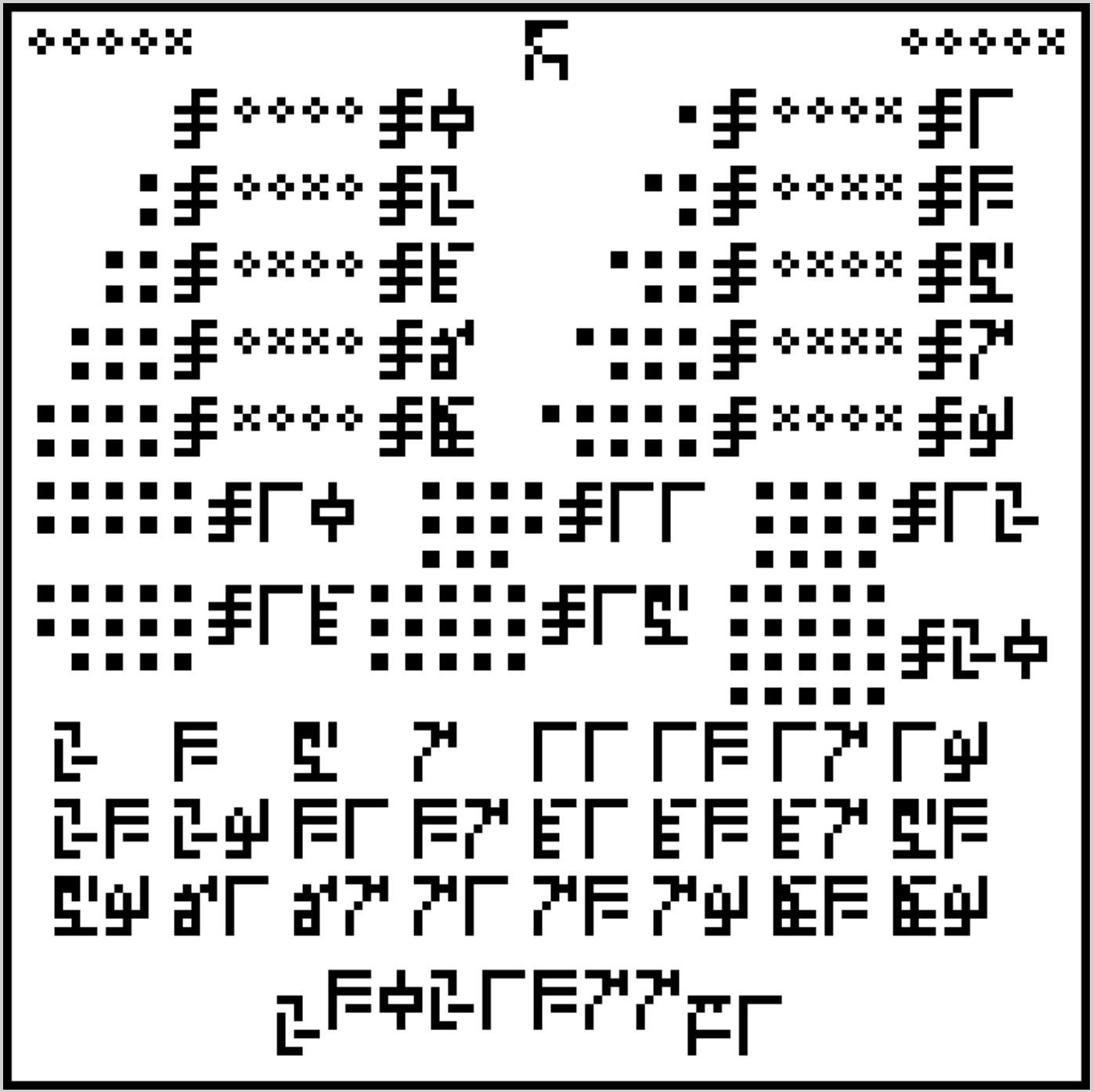
You just received an alien message from outer space. What does the first page mean?



Functional Programming in D

Andreas Zwinkau

2015-08-19

Functional Programming User Group Karlsruhe

What language **do** you work in?

Popular Programming Languages

- Java
- C/C++
- C#
- Python
- PHP
- JavaScript
- Perl
- Shell
- Assembly

Fortran? Cobol? ABAP?

What language would you **want** to work in?

Desired Programming Languages

- Haskell
- Scala
- Rust
- Next JavaScript version
- Clojure

  ...

Do you know D?

Who has heard of the D programming language?

Who wrote at least a line of D code?

Anything larger?

Walter Bright

Wrote first C++ to native code compiler

Wrote Empire on the PDP-10

Pro Compiler Writer

Creator of D (1999)

D design goals

Modern convenience.

Modeling power.

Native efficiency.

D1 had issues

Two standard libraries (Phobos vs Tango)

- Phobos feels like libc
- Tango feels like java.*

Proprietary Compiler Backend

- GDC lagging behind

Resolved with D2 in 2007

Andrei Alexandrescu

Author of „Modern C++ Design“ and „The D Programming Language“

C++ template programming Guru

Research scientist at Facebook

Co-designer of D

D in the real world?

- **Facebook** has C preprocessor „warp“ written in D
- **Sociomantic** (Berlin) does real-time ads bidding
- **Remedy Games** (Max Payne, Alan Wake) is playing with it

  more on http://wiki.dlang.org/Current_D_Use

D design goals

Modern convenience.

Modeling power.

Native efficiency.

D is not small/simple, but „comprehensive“.

D is C++ done right without the baggage.

D: Modern convenience (inference)

```
void main() {
    auto arr = [ 1, 2, 3.14, 5.1, 6 ];
    auto dictionary = [ "one" : 1,
                        "two" : 2, "three" : 3 ];
    auto x = min(arr[0], dictionary["two"]);
}

auto min(T1, T2)(T1 lhs, T2 rhs) {
    return rhs < lhs ? rhs : lhs;
}
```

D: Modern convenience (res. mgmt.)

```
import std.stdio;
class Widget { }

void main()
{
    auto w = new Widget; // GC
    scope(exit) { writeln("Exiting main."); }
    foreach (line; File("text.txt").byLine())
    {
        writeln(line);
    } // File closed deterministically at scope's end (RAII)
    writeln();
}
```

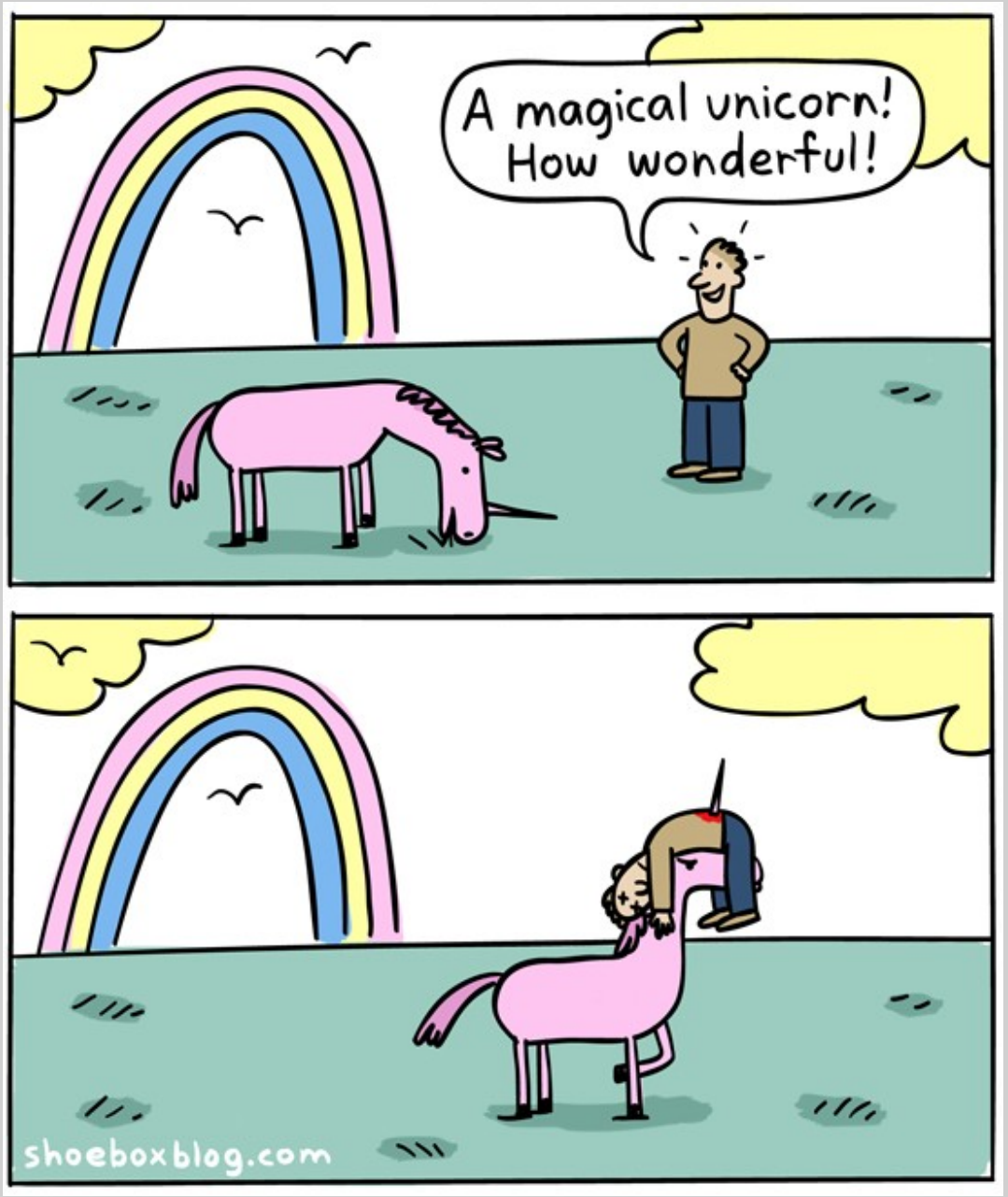
D: Modern convenience (builtin arrays)

```
import std.range, std.stdio;

void main()
{
    ulong lines = 0, sumLength = 0;
    foreach (line; stdin.byLine())
    {
        ++lines;
        sumLength += line.length;
    }
    writeln("Average line length: ",
        lines ? cast(double) sumLength / lines : 0.0);
}
```

D: Modeling power (multi-paradigm)

The best paradigm is to not impose something at the expense of others. D offers classic polymorphism, value semantics, **functional style**, generics, generative programming, contract programming, and more—all harmoniously integrated.



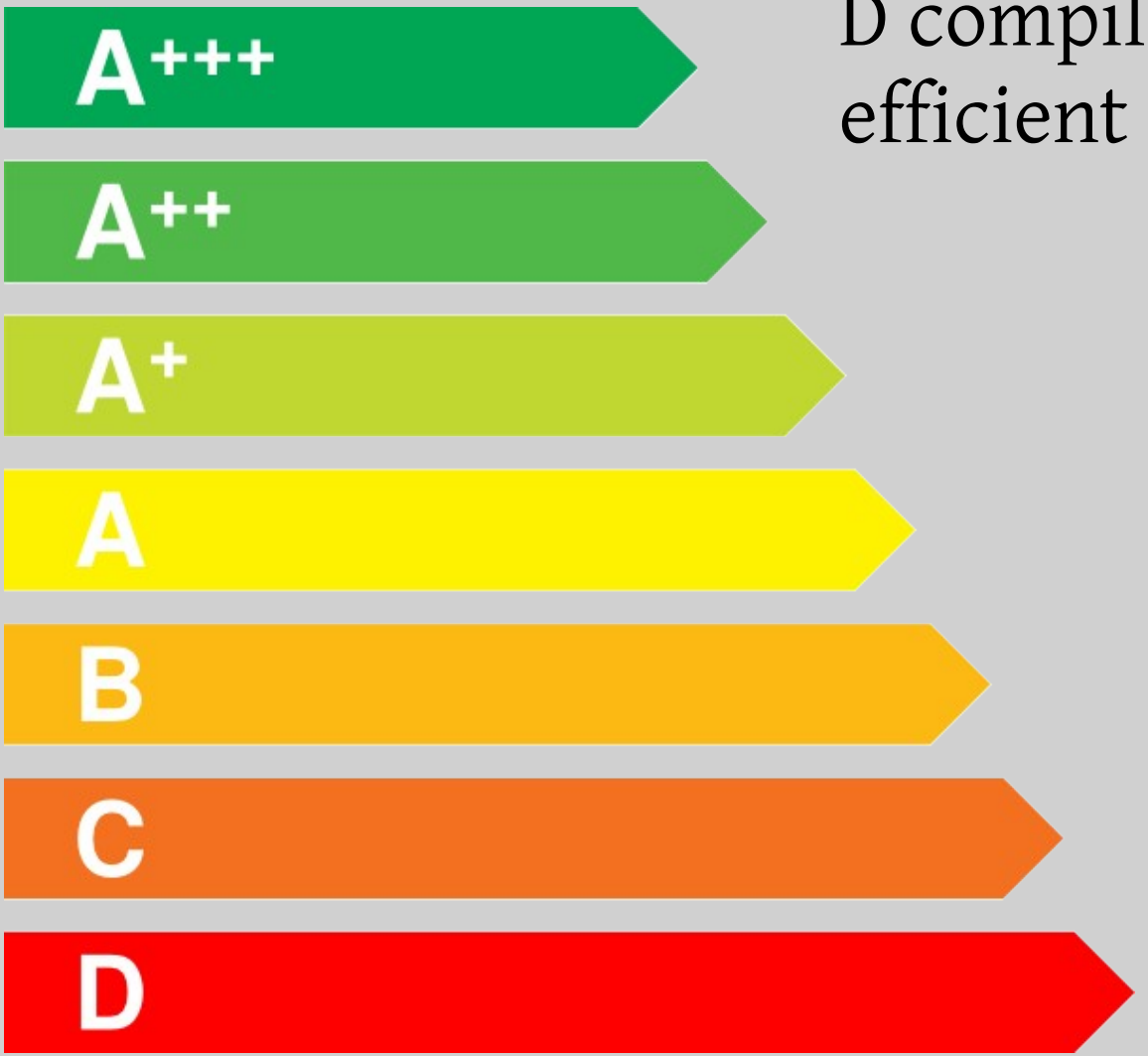
D: Modeling power (concurrency)

D offers an innovative approach to concurrency [and parallelism], featuring true immutable data, message passing, no sharing by default, and controlled mutable sharing across threads.

D: Modeling power (small and large)

From simple scripts to large projects, D has the breadth to scale with **any application**'s needs: unit testing, information hiding, refined modularity, fast compilation, precise interfaces.

D: Native efficiency.



D compiles naturally to efficient native code.

D: Native efficiency (FFI, assembly)

D is designed such that most "obvious" code is fast and safe.

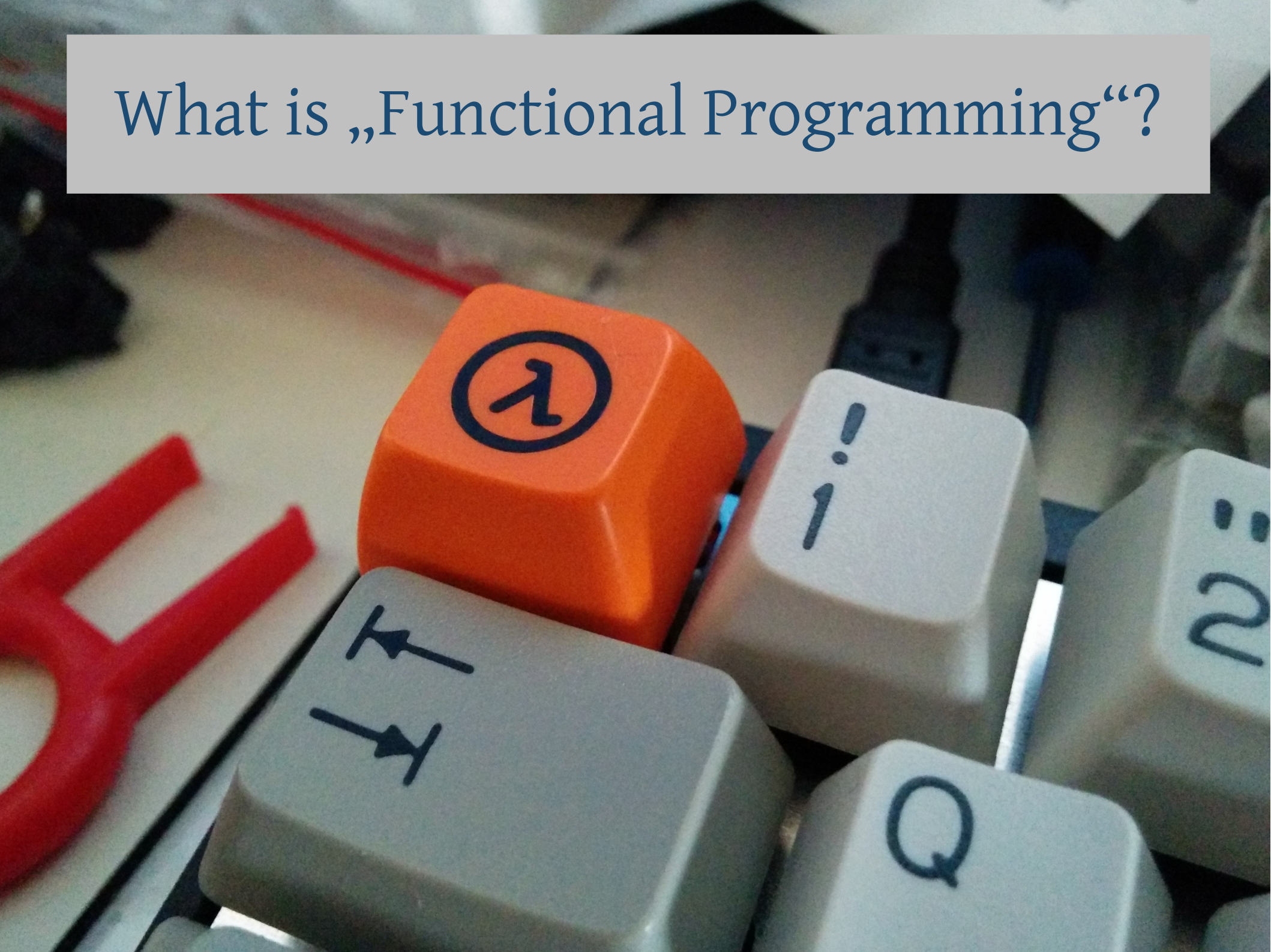
Easy to call into C. (Possible to call into some C++)

Inline assembly.

D: Native efficiency.

The @safe, @trusted, and @system function attributes allow the programmer to best decide the **safety-efficiency tradeoffs** of an application, and have the compiler check for consistency.

What is „Functional Programming“?



What is **cool** about Functional?

Anticipated „Coolness“

- If it compiles, it works
- Easy to parallelize
- Better abstractions
- Easier to reason about
- Discourages side effects
- Easier to test
- Easier reuse
- Clean and elegant

FP is Immutable Data

OO is about encapsulating and hiding state,
FP is about no mutable state.

Implies Garbage Collection

FP is Pure Functions

Functions must not change on global state.

They might depend on global state, but that state is immutable.

FP is First-Class Functions

Dynamically create new functions.

This enables higher-order functions and currying.

FP is **not** about ...

Monads
Lazyness
Static Typing
Type Inference
Recursion
Referential Transparency

Functional Programming is

- Immutable Data
- Pure Functions
- First-Class Functions

imho

What does D provide?

D has anon. functions and delegates

```
auto square = function int(int x)
    { return x * x; }

int exponent = 2;
auto square = delegate int(int x)
    { return pow(x, exponent); }

auto square = (int x) => x * x;
```

D std lib has standard FP tools

```
import std.algorithm: map, filter, reduce;
import std.functional: curry, memoize,
compose;
```

D const is transitive

```
class Foo {
  public Bar b;
}

baz(const Foo f) {
  auto b2 = f.b; // b2 const as well
}
```

const vs immutable

```
const     Foo a;
          Foo b;
immutable Foo c;

void foo(const Foo x);
foo(a);
foo(b);
foo(c);
```

D has pure functions

- cannot read or write global or static (mutable) state
- cannot call impure functions (IO,extern,etc).

Is that good enough?

Problems with purity

„Programming with pure functions will involve more copying of data, and in some cases this clearly makes it the incorrect implementation strategy due to **performance considerations**. As an extreme example, you can write a pure DrawTriangle() function that takes a framebuffer as a parameter and returns a completely new framebuffer with the triangle drawn into it as a result. Don't do that."

–John Carmack, #AltDevBlog 2012

strongly vs weakly pure

```
pure Foo bar(Foo f); // weakly pure
pure Foo bar(const Foo f); // strongly pure

class Foo {
    public TheWorld world;
    ...
}
```

Weakly pure is useful.

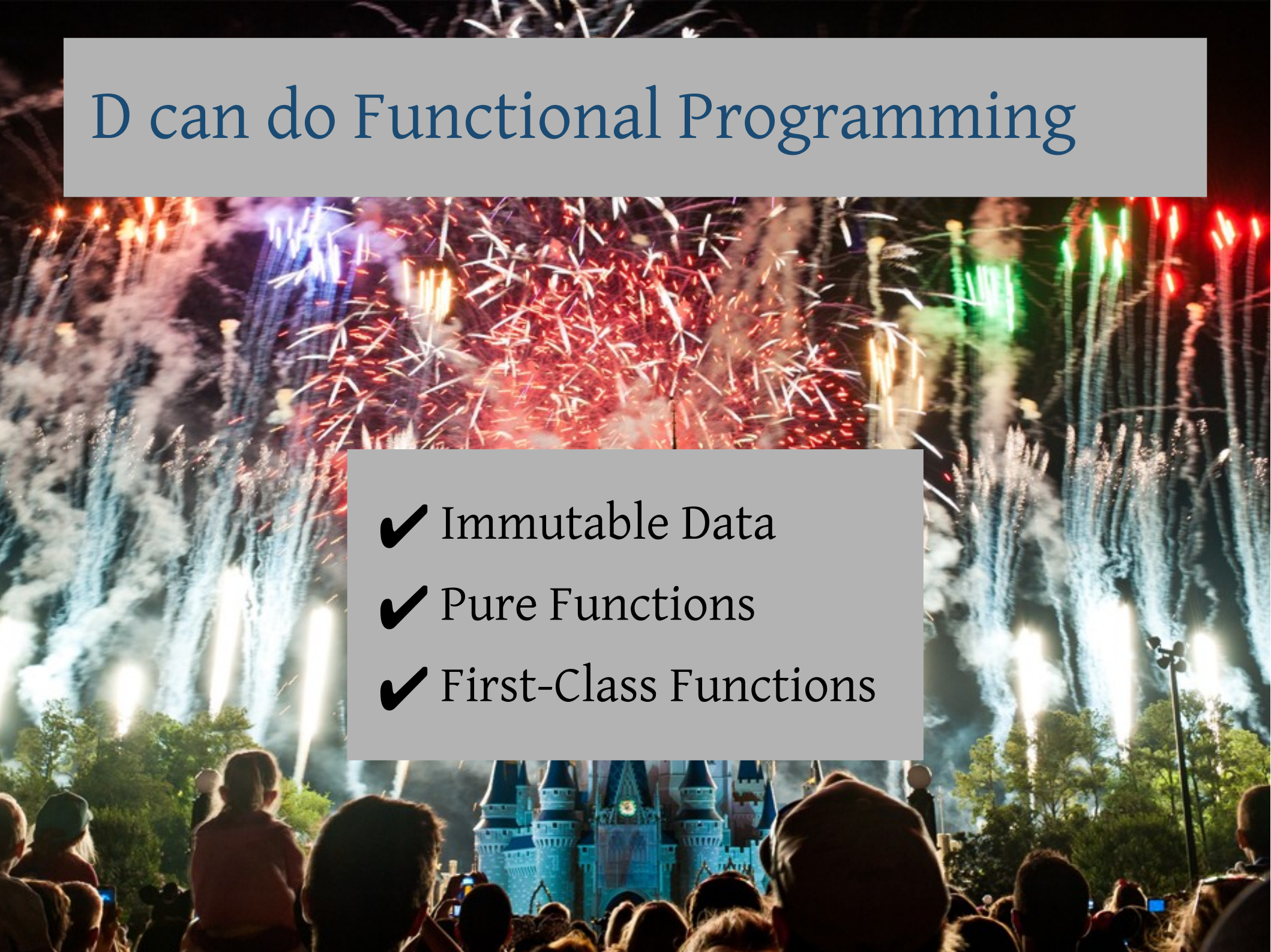
```
pure void DrawTriangle(Framebuffer fb, ...);
```

A weakly pure DrawTriangle is guaranteed to only modify the framebuffer it takes as a parameter.

pure has pragmatic loopholes

- can throw exceptions
- can terminate the program
- can allocate memory
- can do impure things in debug statements

D can do Functional Programming



- ✔ Immutable Data
- ✔ Pure Functions
- ✔ First-Class Functions

D can do lazy

```
void log(lazy string dg) {
    if (logging)
        fwritefln(logfile, dg());
}

void f(Foo x) {
    log("Enter f() with x = "~toString(x));
}
```

Haskell's lazy lists in D?

D champions „ranges“.

sort(1) in D

```
void main() {
        stdin
         .byLine(KeepTerminator.yes)
         .map!(a => a.idup)
         .array
         .sort
         .copy(stdout.lockingTextWriter());
}
```

Monads in D

See C++ http://bartoszmilewski.com/2011/07/11/monads-in-c/

Where typeclasses fail ...

... subtly changing from functional to generic programming ...

Think Collections

ArrayList, LinkedList, Queue, Set, Infinite Lists, etc

Can you
- insert at the front/back? (Not both for queues)
- iterate front/back/both? (Not all for LinkedList)
- get the length? (Not for infinite lists)
- is it thread-safe?

Lets make Interfaces

- FrontInsertable
- BackInsertable
- ForwardIterable
- BackwardIterable
- RandomAccessible
- HasLengthInterface
- ThreadSafeI

What about combinations?

Interfaces, concepts, traits, typeclasses have a problem: Names.

```
interface
FrontBackInsertableRandomAccessibleWithLength
extends
  FrontInsertable,
  BackInsertable,
  RandomAccessible,
  HasLengthInterface

class ArrayList implements
FrontBackInsertableRandomAccessibleWithLength
```

Oh and ... is it serializable? Cloneable? Comparable?

Challenge: chunk

Write a generic function chunk.
Takes a `Collection<T>` and an `int` n as input.
Outputs a `Collection<Collection<T>>`,
where every n items are grouped together.

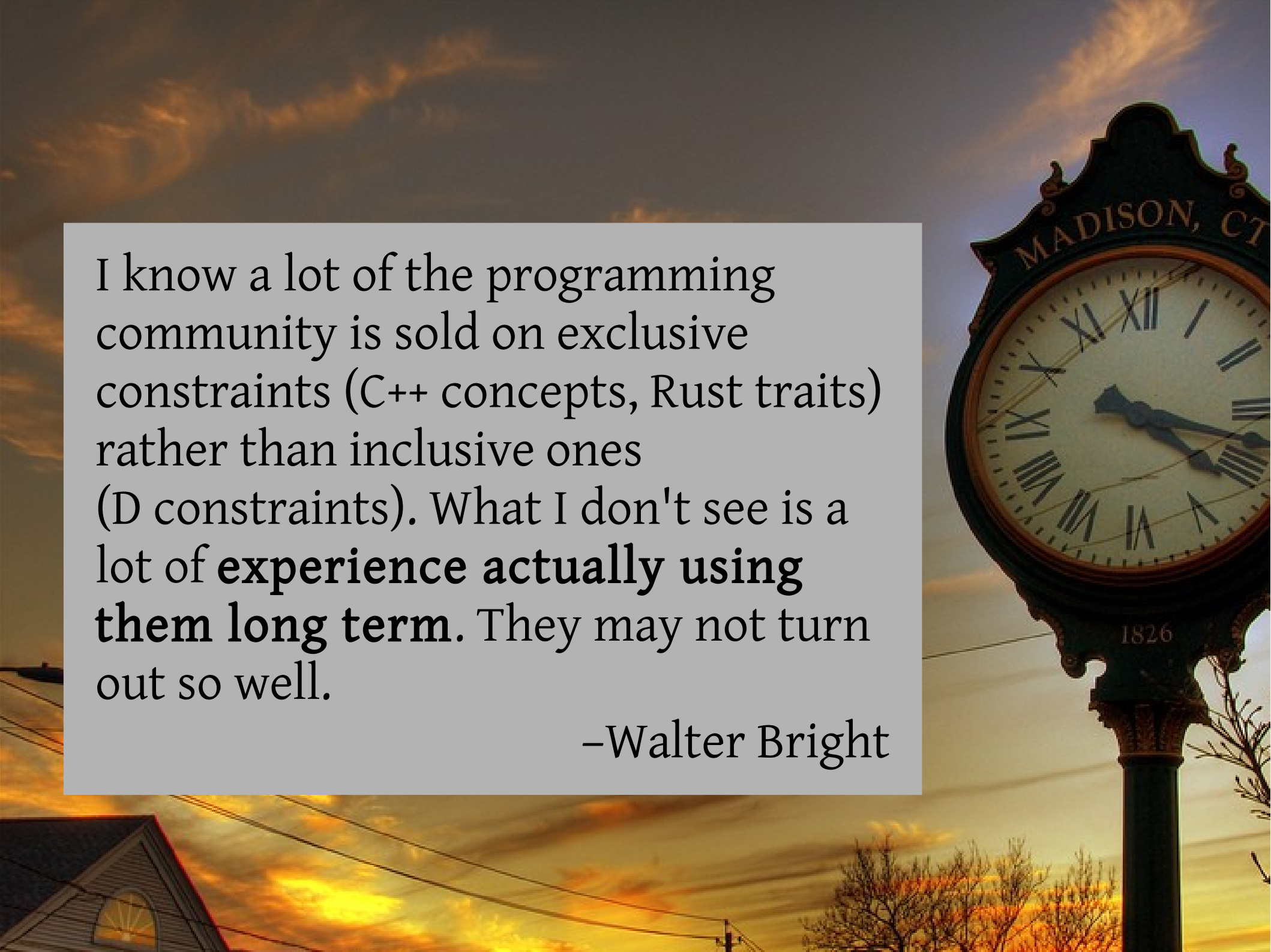
Example: [1,2,3,4,5,6] => [[1,2],[3,4],[5,6]]

Should work with ArrayList, LinkedList, Queue, etc

D has static-if to the rescue

```
C!(C!T) chunk(C,T)(C!T input,int n)
if (hasRandomAccess(C)) {
  // use slices of C => nearly no allocation
}

C!(C!T) chunk(C,T,int n)(C!T input)
if (isForwardIterable(C)) {
  // pop elements one by one
  static if (isReferenceType(T)) {
  } else {
    static assert (isCopyable(T));
  }
}
```



I know a lot of the programming community is sold on exclusive constraints (C++ concepts, Rust traits) rather than inclusive ones (D constraints). What I don't see is a lot of **experience actually using them long term**. They may not turn out so well.

–Walter Bright

D can do functional

... and all the other paradigms

D is cool.

- Easy to parallelize
- Great at (zero-cost) abstractions
- Annotations to make it easier to reason about
- Forbid side effects selectively
- Encourages to use builtin unit testing
- Generic programming for easy reuse
- Clean and elegant

Try 

Go to http://dlang.org

Downloads for Win, OS X, Ubuntu, FreeBSD, etc

For help ask at http://forum.dlang.org/


Want more? Really?

@safe: undefined behavior forbidden

- No casting from a pointer type to any type other than void*.
- No casting from any non-pointer type to a pointer type.
- No modification of pointer values.
- Cannot access unions that have pointers or references overlapping with other types.
- Calling any system functions.
- No catching of exceptions that are not derived from class Exception.
- No inline assembler.
- No explicit casting of mutable objects to immutable.
- No explicit casting of immutable objects to mutable.
- No explicit casting of thread local objects to shared.
- No explicit casting of shared objects to thread local.
- No taking the address of a local variable or function parameter.
- Cannot access __gshared variables

inline unittests

```
int half(int x) {
   return x*2;
}

unittest {
  assert (half(84) == 42, „half is broken“);
}
```

Contracts

```
int half(int x)
in              { assert (x > 42); }
out (result) { assert (result*2 == x); }
body {
   return x/2;
}
```

scope()

```
auto fh = open(foo);
scope (exit) fh.close();
fh.read();
```

Image sources in order of appearance:

https://www.flickr.com/photos/astrid/8886371211/

https://www.flickr.com/photos/randar/15036720742/

https://www.flickr.com/photos/51035610542@N01/6868746106/

https://www.flickr.com/photos/tombricker/8007545819/

https://www.flickr.com/photos/slack12/316774124/

https://www.flickr.com/photos/jabb/5582573164/