

Scala – From Zero to Testing

<https://github.com/ericssmith/neb15>

Eric Smith

Goals

Understand library
APIs

Write principled code

Organize larger
programs

Morning Agenda



Intro

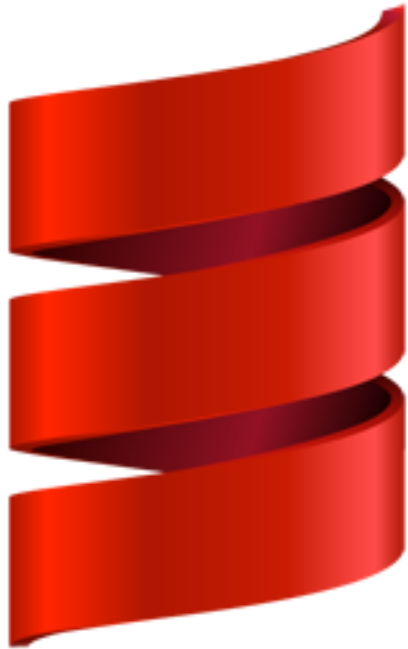
IntelliJ

Essentials

Collections

Sequencing

Afternoon Agenda



Option

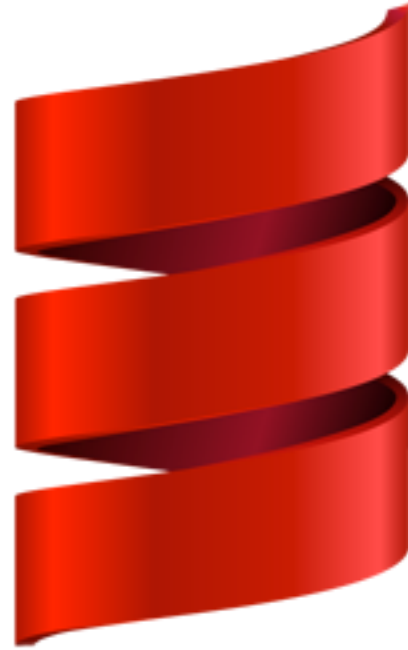
I/O

Java

Testing

Libraries

Scalable language



Scalable language



Scaling

Features

Users

Servers

Developers

Concepts

Expressions

Functions as values

Subtyping

Polymorphism

Algebraic types


Pattern matching

Abstract types

Modules

Use the worksheet (REPL)

Code with expressions



... the thing an expression denotes, i.e., its
"value", depends only on the values of its sub-
expressions

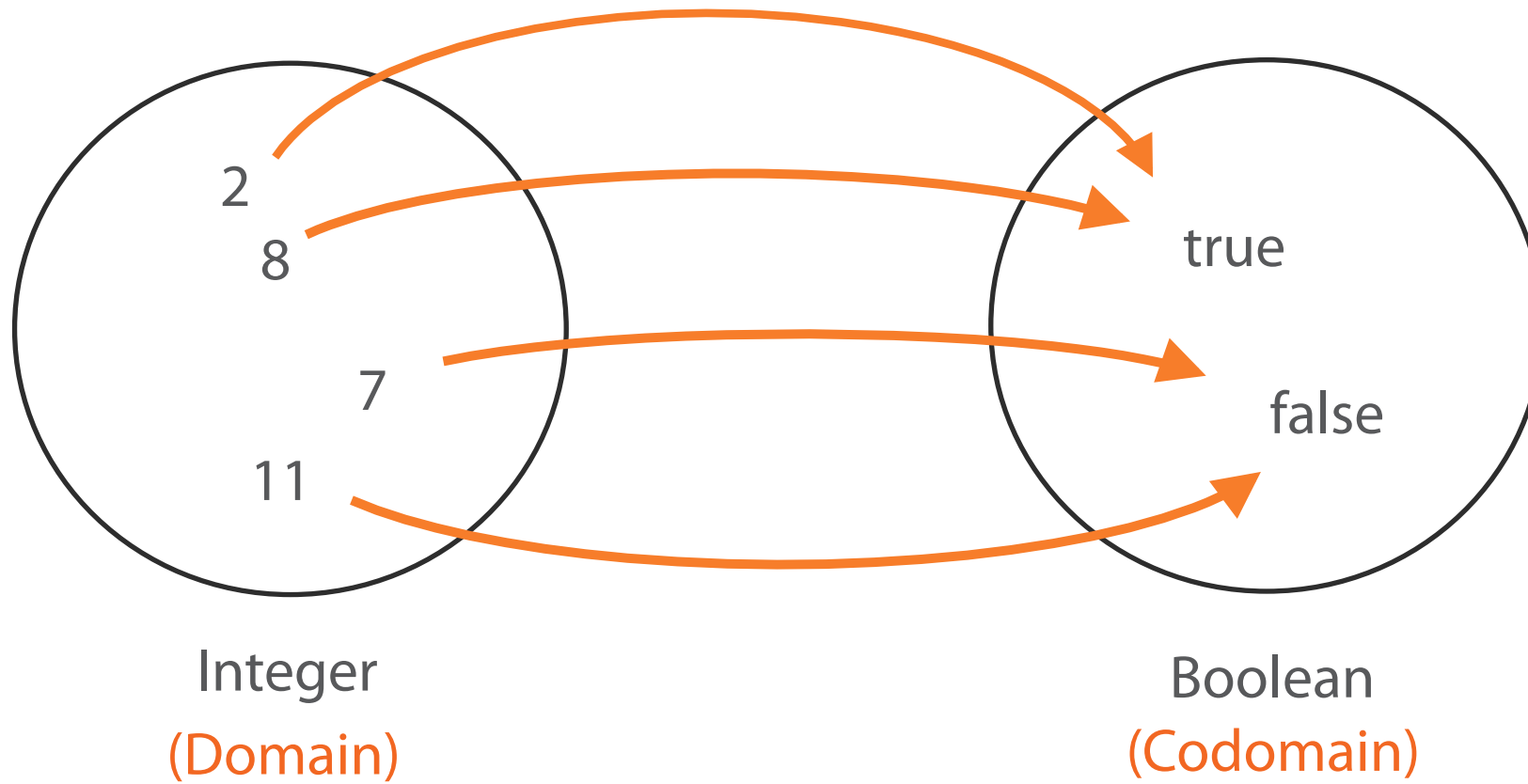
— Peter Landin



Think in functions

What is the relation?

"Even"



Domain	Image
....	???
-2	true
-1	false
0	true
1	false
2	true
3	false
4	true
....	???

Scala has modules

Modules

object keyword — no parameters to module

class keyword — parameterized module

Compile faster

Avoid name
collisions

Hide things

Reasons we're interested in modules

Use algebraic types

Making types from other types: compound types

Algebra

Elements of one or
more sets

Operations on
elements of sets

Rules for relating
the operations

Cartesian Product

Combining elements from multiple sets

.

“Product type”

.

Struct, record, tuples — these are supported in most programming languages.

Disjoint Union

Union of two or more sets without overlap

.

“Sum type”

.

These are not present in most programming languages.

```
data Shape = Circle Float | Rectangle Float Float
```

```
getShapeArea s = case s of
```

```
    Circle r -> 3.14 * r * r
```

```
    Rectangle l w -> l * w
```

Haskell

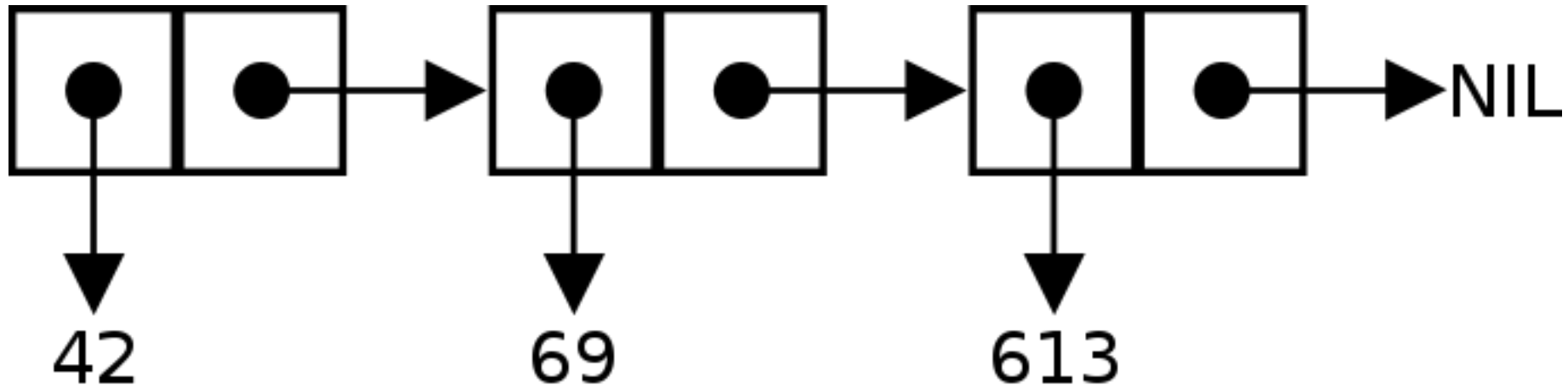
Eric likes Haskell. Haskell has nice motivating examples of many things.

```
type shape = Circle of float | Rectangle of (float * float)
```

```
let get_shape_area s = match s with  
  | Circle r -> 3.14 *. r *. r  
  | Rectangle (x, y) -> x *. y
```

OCaml

List from *cons* cells



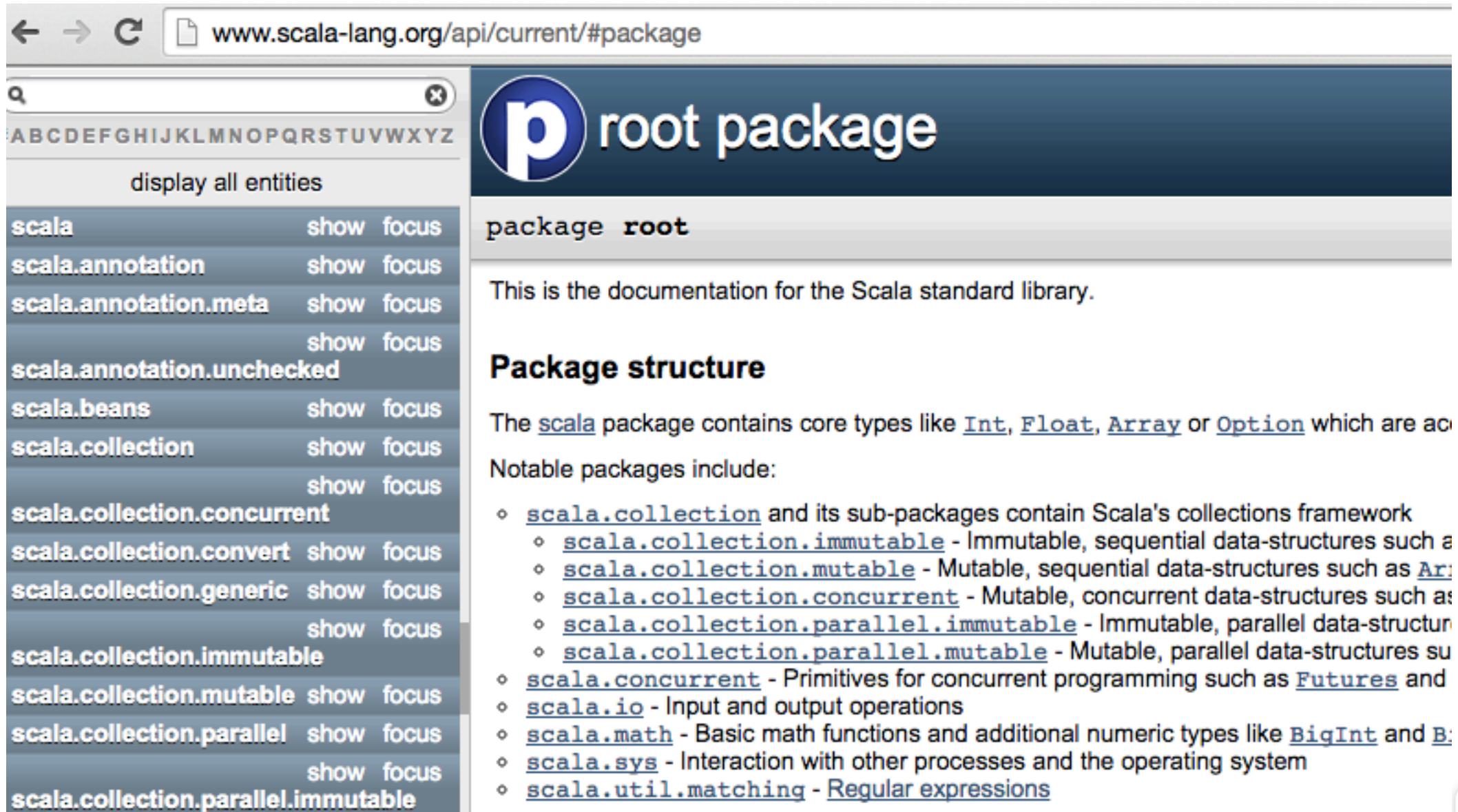
List(42, 69, 613)

42 :: 69 :: 613 :: NIL

Cons(42, Cons(69, Cons(613, NIL)))

Know the Collections

Scala Libraries



The screenshot shows the Scala API documentation website. The browser address bar displays `www.scala-lang.org/api/current/#package`. On the left, there is a search bar and an alphabetical index (A-Z). Below the index is a table listing various Scala packages, each with 'show' and 'focus' links. The main content area features a large 'p root package' header. Below this, it states 'This is the documentation for the Scala standard library.' and 'Package structure'. A paragraph explains that the `scala` package contains core types like `Int`, `Float`, `Array`, or `Option`. It then lists notable packages including `scala.collection` (with its sub-packages), `scala.concurrent`, `scala.io`, `scala.math`, `scala.sys`, and `scala.util.matching`.

← → ↻ `www.scala-lang.org/api/current/#package`

q ×

ABCDEFGHIJKLMNOPQRSTUVWXYZ

display all entities

scala	show	focus
scala.annotation	show	focus
scala.annotation.meta	show	focus
scala.annotation.unchecked	show	focus
scala.beans	show	focus
scala.collection	show	focus
scala.collection.concurrent	show	focus
scala.collection.convert	show	focus
scala.collection.generic	show	focus
scala.collection.immutable	show	focus
scala.collection.mutable	show	focus
scala.collection.parallel	show	focus
scala.collection.parallel.immutable	show	focus

p root package

package **root**

This is the documentation for the Scala standard library.

Package structure

The [scala](#) package contains core types like [Int](#), [Float](#), [Array](#) or [Option](#) which are accessible from any Scala program.

Notable packages include:

- [scala.collection](#) and its sub-packages contain Scala's collections framework
 - [scala.collection.immutable](#) - Immutable, sequential data-structures such as [List](#)
 - [scala.collection.mutable](#) - Mutable, sequential data-structures such as [Array](#)
 - [scala.collection.concurrent](#) - Mutable, concurrent data-structures such as [ConcurrentHashMap](#)
 - [scala.collection.parallel.immutable](#) - Immutable, parallel data-structures such as [ParList](#)
 - [scala.collection.parallel.mutable](#) - Mutable, parallel data-structures such as [ParArray](#)
- [scala.concurrent](#) - Primitives for concurrent programming such as [Futures](#) and [Promises](#)
- [scala.io](#) - Input and output operations
- [scala.math](#) - Basic math functions and additional numeric types like [BigInt](#) and [BigDecimal](#)
- [scala.sys](#) - Interaction with other processes and the operating system
- [scala.util.matching](#) - [Regular expressions](#)

Collections – Performance

	head	tail	apply	update	prepend	append	insert
immutable							
List	C	C	L	L	C	L	-
Stream	C	C	L	L	C	L	-
Vector	eC	eC	eC	eC	eC	eC	-
Stack	C	C	L	L	C	C	L
Queue	aC	aC	L	L	L	C	-
Range	C	C	C	-	-	-	-
String	C	L	C	L	L	L	-

C	The operation takes (fast) constant time.
eC	The operation takes effectively constant time, but this might depend on some assumptions such as maximum length of a vector or distribution of hash keys.
aC	The operation takes amortized constant time. Some invocations of the operation might take longer, but if many operations are performed on average only constant time per operation is taken.
Log	The operation takes time proportional to the logarithm of the collection size.
L	The operation is linear, that is it takes time proportional to the collection size.
-	The operation is not supported.

Use Option


Prefer Option functions
over pattern matching

Use Scalaz Disjunction

\vee

Keep side-effecting code
on the outside of the program

Avoid statements and
variable assignment

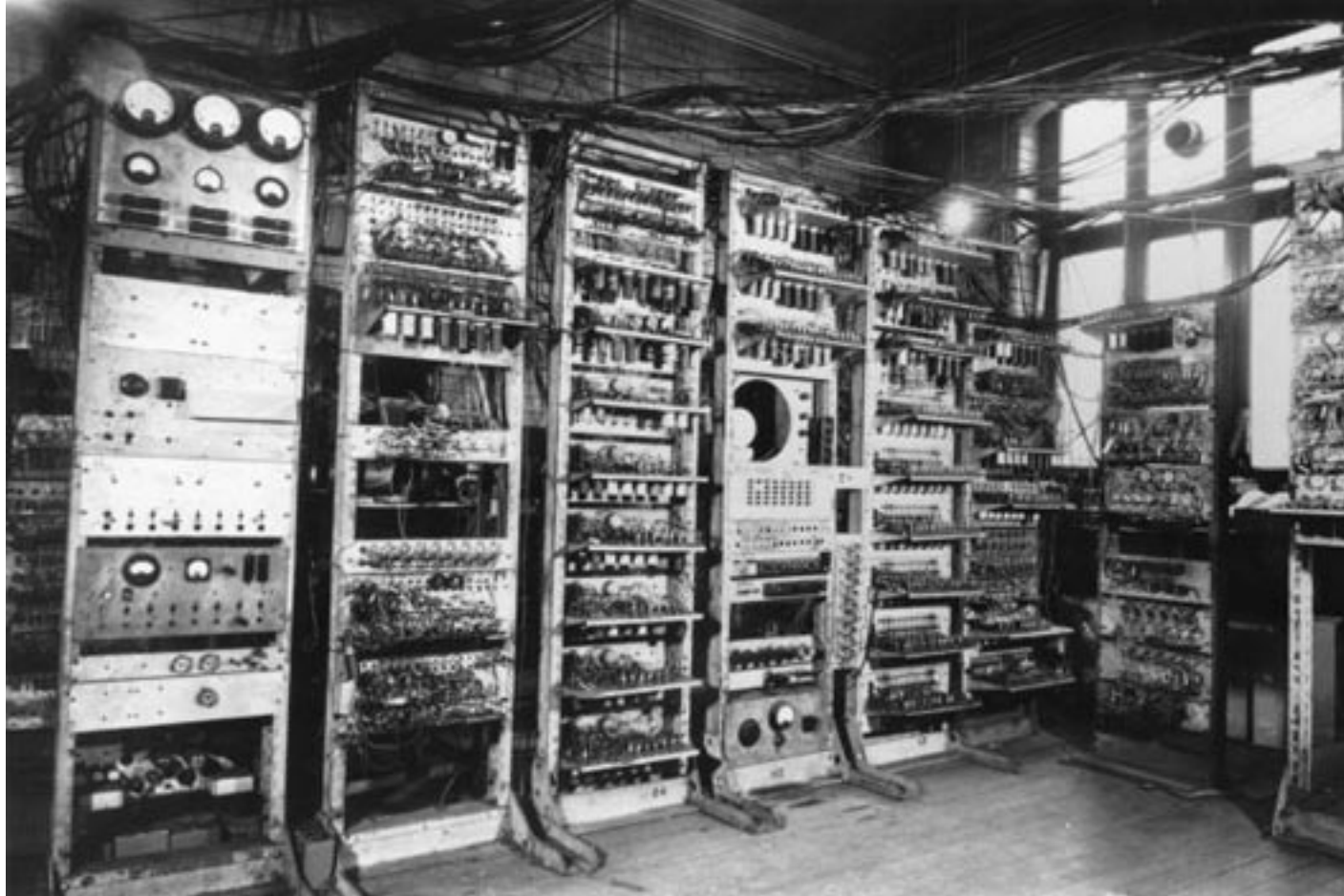



Von Neumann programming languages use variables to imitate the computer's storage cells; control statements elaborate its jump and test instructions; and assignment statements imitate its fetching, storing, and arithmetic.

— John Backus



Manchester "Baby"






But one day it stopped, and there, shining brightly in the expected place, was the expected answer. It was a moment to remember. This was in June 1948, and nothing was ever the same again.

— Frank Williams





... the realization came over me with full force
that a good part of the remainder of my life
was going to be spent in finding errors in my
own programs.

— Maurice Wilkes

