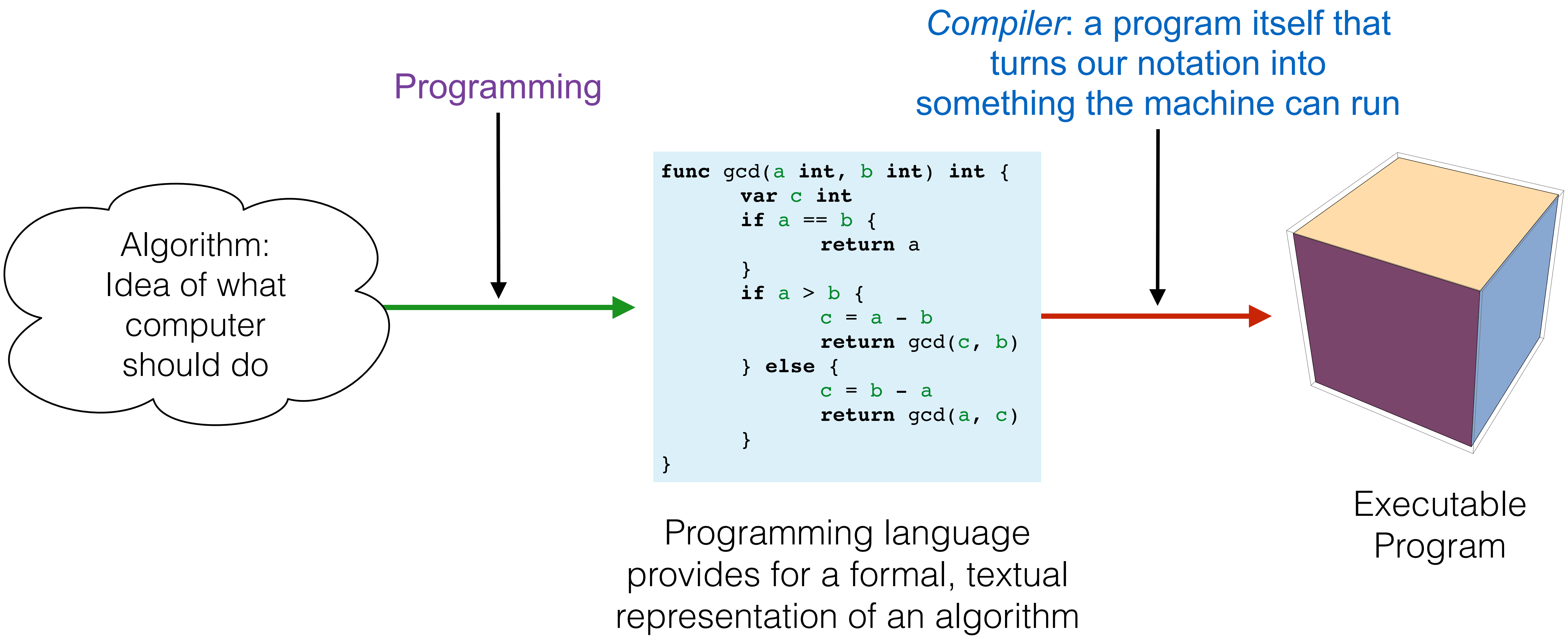
Programming for Scientists

Introduction

02-201 / 02-601

What is Programming?

Programming is clearly, correctly telling a computer what to do.



Compiler: a program itself that turns our notation into something the machine can run

```
func gcd(a int, b int) int {
        var c int
        if a == b {
                return a
        }
        if a > b {
                c = a - b
                return gcd(c, b)
        } else {
                c = b - a
                return gcd(a, c)
        }
}
```

Programming language provides for a formal, textual representation of an algorithm

Calculus is a formal notation for a branch of mathematics:

$$f(t) = \int_0^t \frac{1}{\ln(t)} dt$$

Programming languages provide an even more formal description of algorithms.

Why is Programming Important?

This case almost doesn’t need to be made nowadays:

Facebook, Google, LinkedIn, Twitter, Pinterest, DuoLingo, Flappy Bird, NetFlix, iTunes, Microsoft Word, Minecraft, …

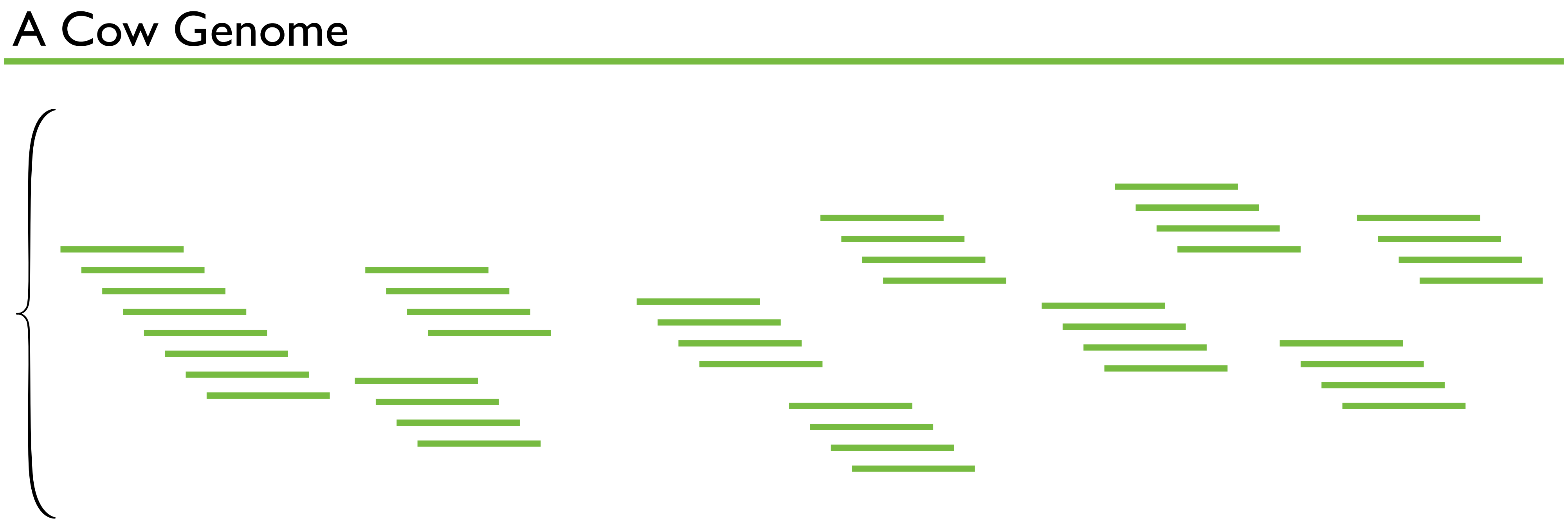
These are both “programs” and part of nearly all of our lives.

Why is programming important for scientists?

- simulation
- statistical tests
- data cleaning
- data visualization
- search
- data processing → insights
- machine learning (predicting attributes)
- modeling

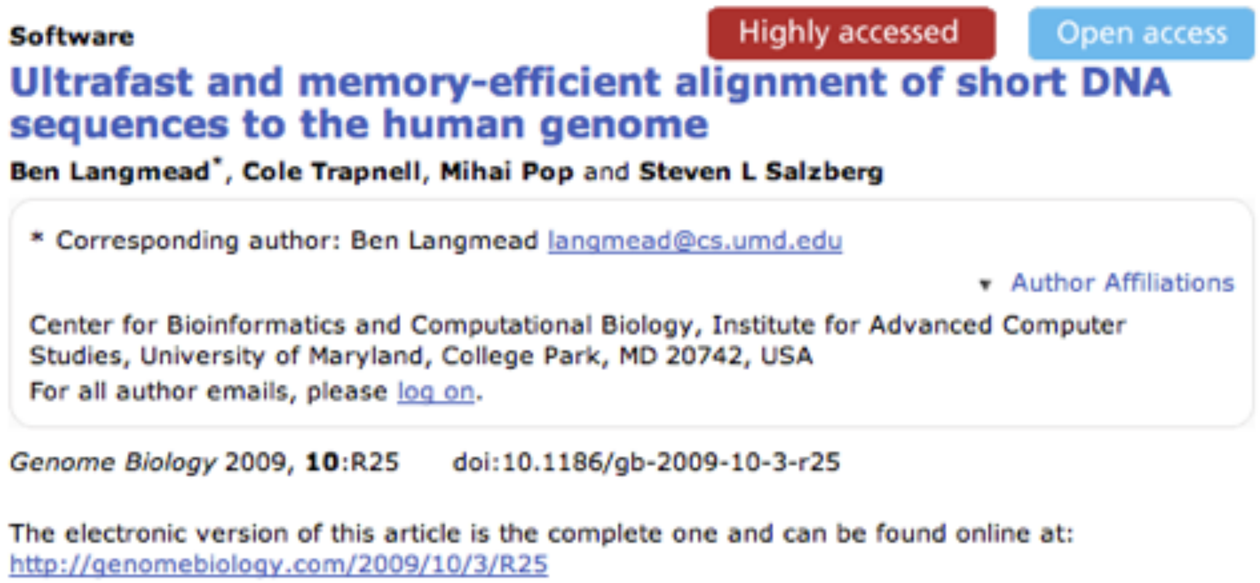
Example Programs in Science

Genome Sequencing



Sequencing technologies produce millions of "reads" = a random, short substring of the genome

If we already know the genome of one cow, we can get reads from a 2nd cow and map them onto the known cow genome — Need to do millions of string searches in a long string.


Software

Highly accessed

Open access

Ultrafast and memory-efficient alignment of short DNA sequences to the human genome

Ben Langmead*, Cole Trapnell, Mihai Pop and Steven L Salzberg

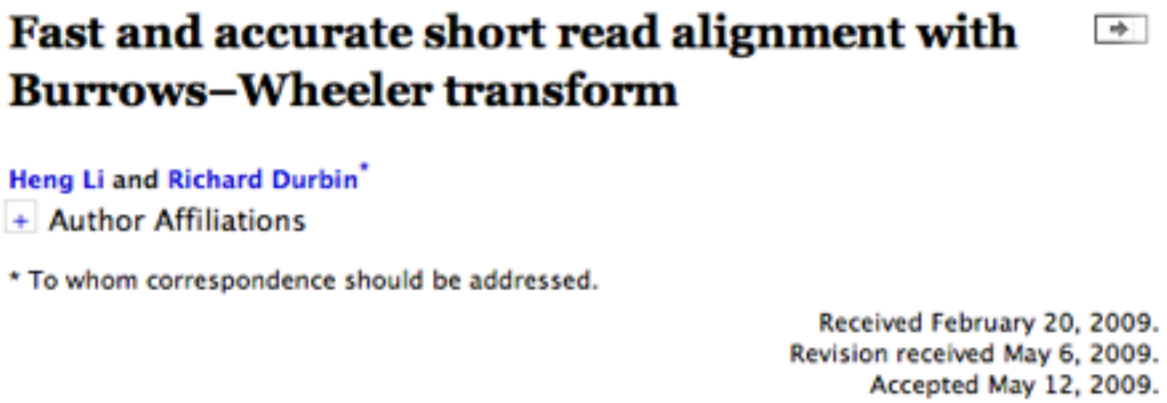
* Corresponding author: Ben Langmead langmead@cs.umd.edu

▾ Author Affiliations

Center for Bioinformatics and Computational Biology, Institute for Advanced Computer Studies, University of Maryland, College Park, MD 20742, USA
For all author emails, please log on.

Genome Biology 2009, **10**:R25 doi:10.1186/gb-2009-10-3-r25

The electronic version of this article is the complete one and can be found online at: http://genomebiology.com/2009/10/3/R25


Fast and accurate short read alignment with Burrows–Wheeler transform

Heng Li and Richard Durbin*

+ Author Affiliations

* To whom correspondence should be addressed.

Received February 20, 2009.
Revision received May 6, 2009.
Accepted May 12, 2009.

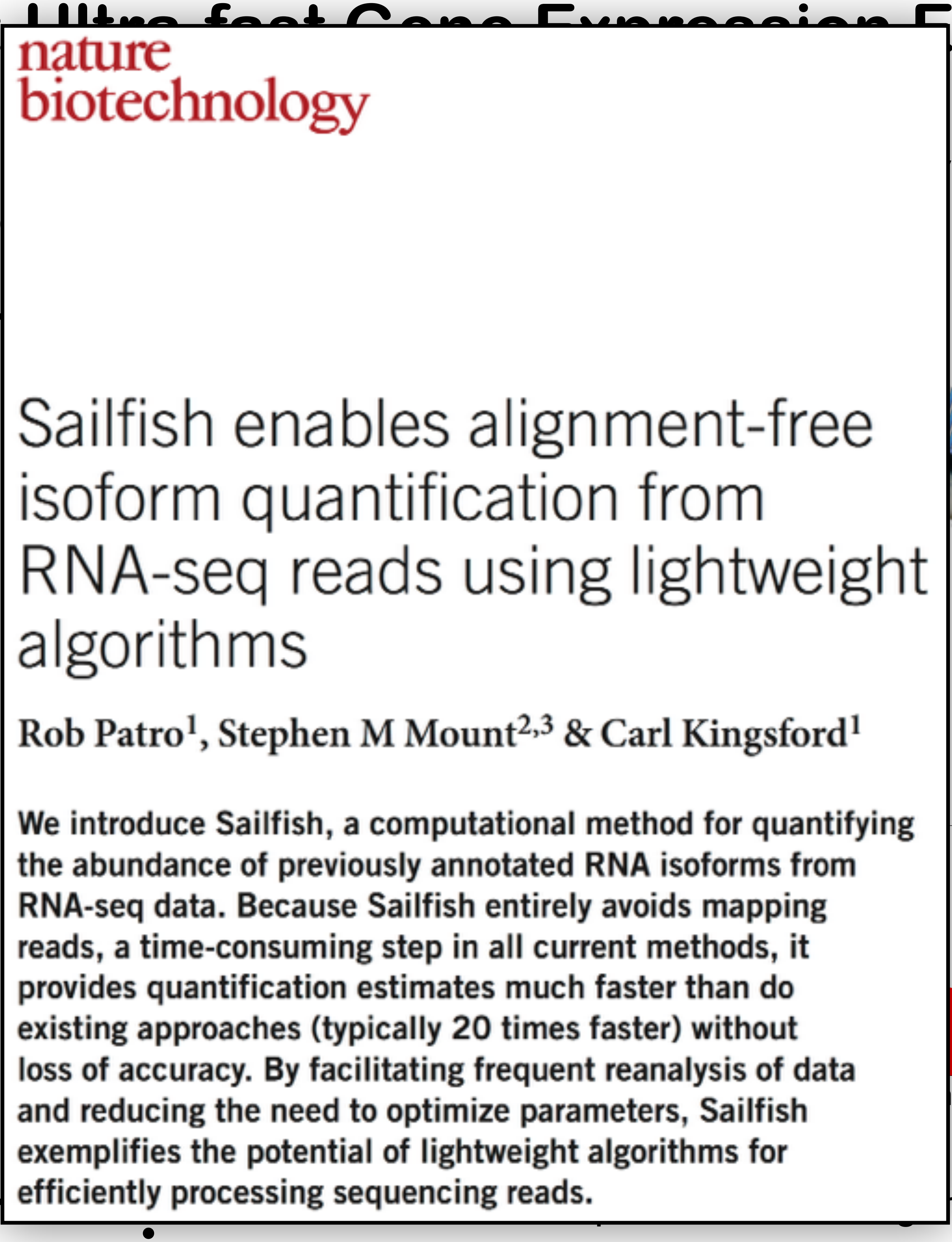
Bioinformatics (2009) 25(14):1754-1760.

Sailfish: Ultra-fast Gene Expression Estimation

- Measuring
  er organism
  response t

RNA-seq:

10m to 100m reads sampled from genes expressed during a condition

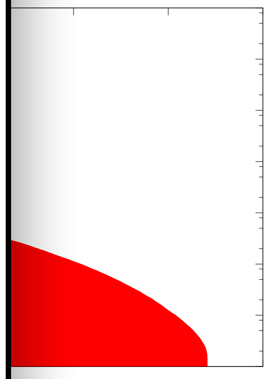


nature
biotechnology

Sailfish enables alignment-free isoform quantification from RNA-seq reads using lightweight algorithms

Rob Patro[1], Stephen M Mount[2,3] & Carl Kingsford[1]

We introduce Sailfish, a computational method for quantifying the abundance of previously annotated RNA isoforms from RNA-seq data. Because Sailfish entirely avoids mapping reads, a time-consuming step in all current methods, it provides quantification estimates much faster than do existing approaches (typically 20 times faster) without loss of accuracy. By facilitating frequent reanalysis of data and reducing the need to optimize parameters, Sailfish exemplifies the potential of lightweight algorithms for efficiently processing sequencing reads.



e

ines the relative

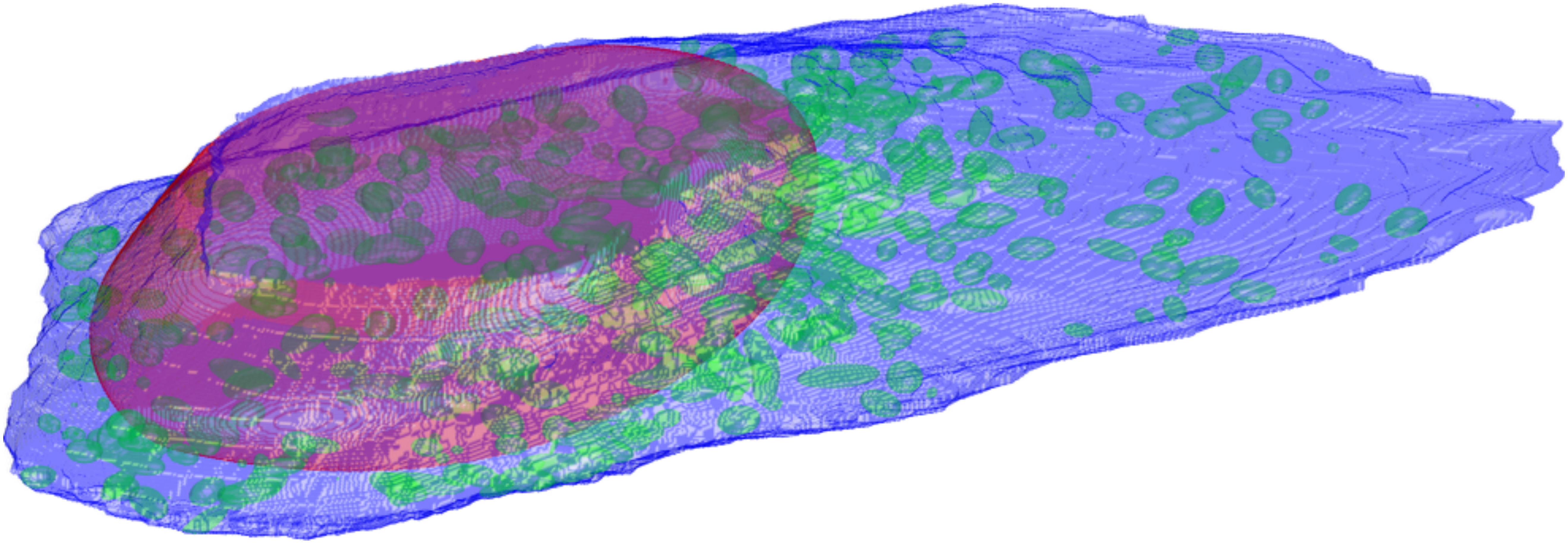
hes and their isoforms

Cell Organizer

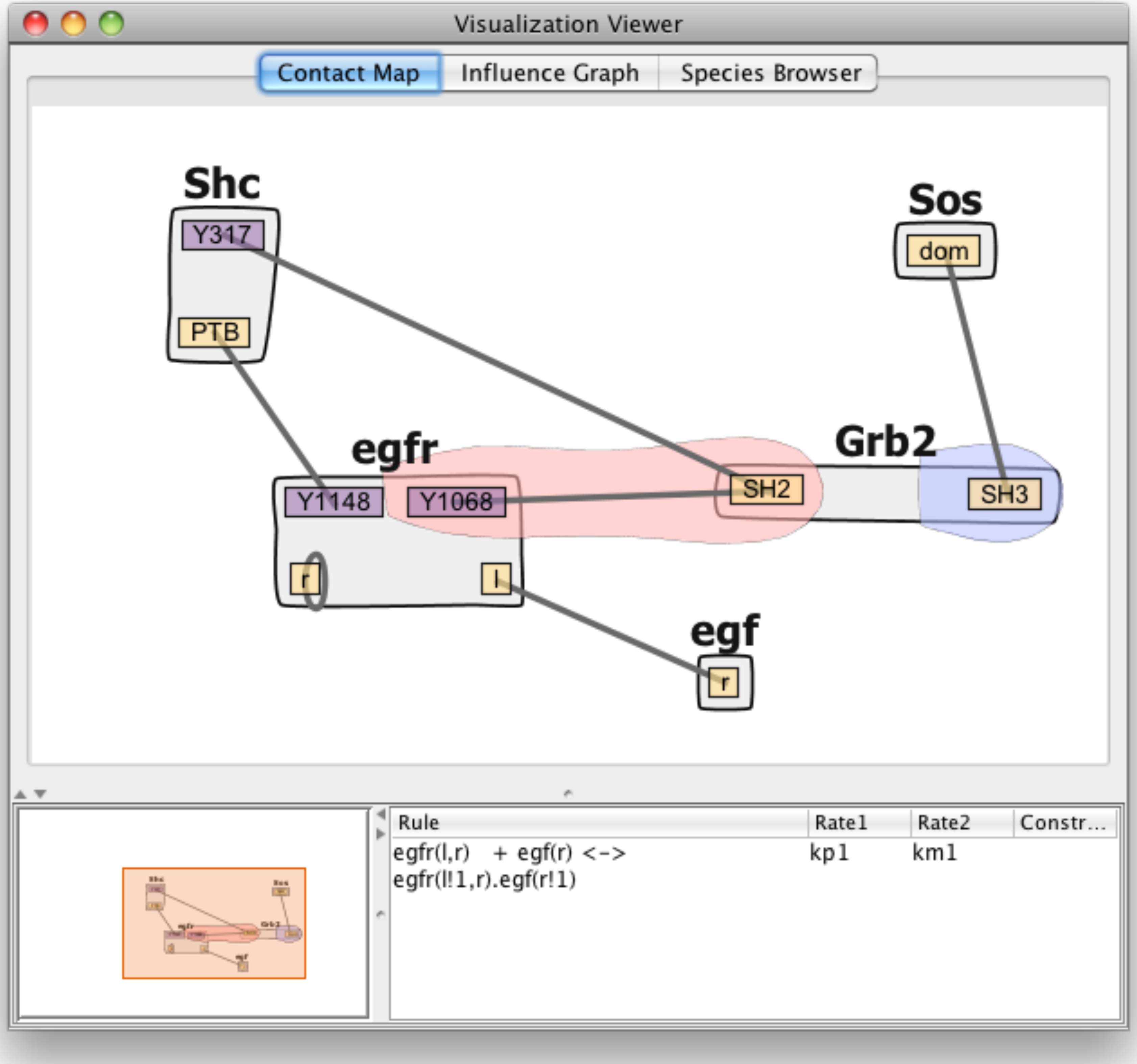
http://cellorganizer.org

CellOrganizer can learn models of

- cell shape
- nuclear shape
- chromatin texture
- vesicular organelle size, shape and position
- microtubule distribution.



Modeling Biological Reactions



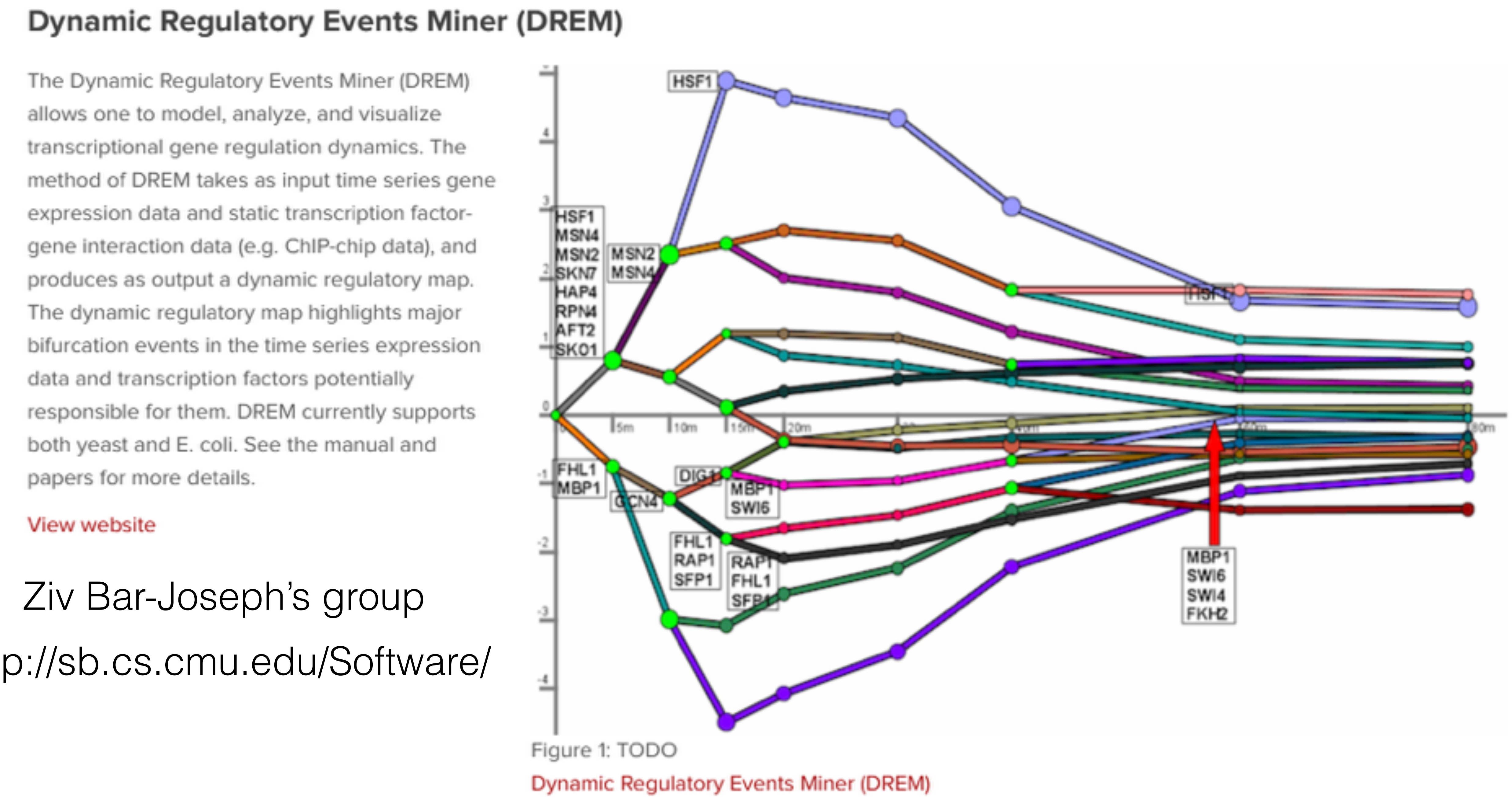
Rule	Rate1	Rate2	Constr...
egfr(l,r) + egf(r) <-> egfr(l!1,r).egf(r!1)	kp1	km1	

RuleBender & BioNetGen

Software for modeling biological pathways and chemical reactions

http://rulebender.cs.pitt.edu/wordpress/?page_id=16

Analysis of Gene Regulation



Dynamic Regulatory Events Miner (DREM)

The Dynamic Regulatory Events Miner (DREM) allows one to model, analyze, and visualize transcriptional gene regulation dynamics. The method of DREM takes as input time series gene expression data and static transcription factor-gene interaction data (e.g. ChIP-chip data), and produces as output a dynamic regulatory map. The dynamic regulatory map highlights major bifurcation events in the time series expression data and transcription factors potentially responsible for them. DREM currently supports both yeast and E. coli. See the manual and papers for more details.

View website

Figure 1: TODO

Dynamic Regulatory Events Miner (DREM)

Ziv Bar-Joseph's group

http://sb.cs.cmu.edu/Software/

One could go on and on…

Programming is now central to nearly all of science.

Programming as Carpentry

http://www.texastoolbox.com/cabinet-installation.htm

https://woodgears.ca/dovetail/dovetail_vs_boxjoint.html

There are good cabinets and bad cabinets.

As with carpentry, takes practice to develop the skill to write good programs.

Example Go Program!

Computing the constant “e”:

$$e = \sum_{n=0}^{\infty} \frac{1}{n!} = 1 + \frac{1}{1} + \frac{1}{1 \cdot 2} + \frac{1}{1 \cdot 2 \cdot 3} + \cdots$$

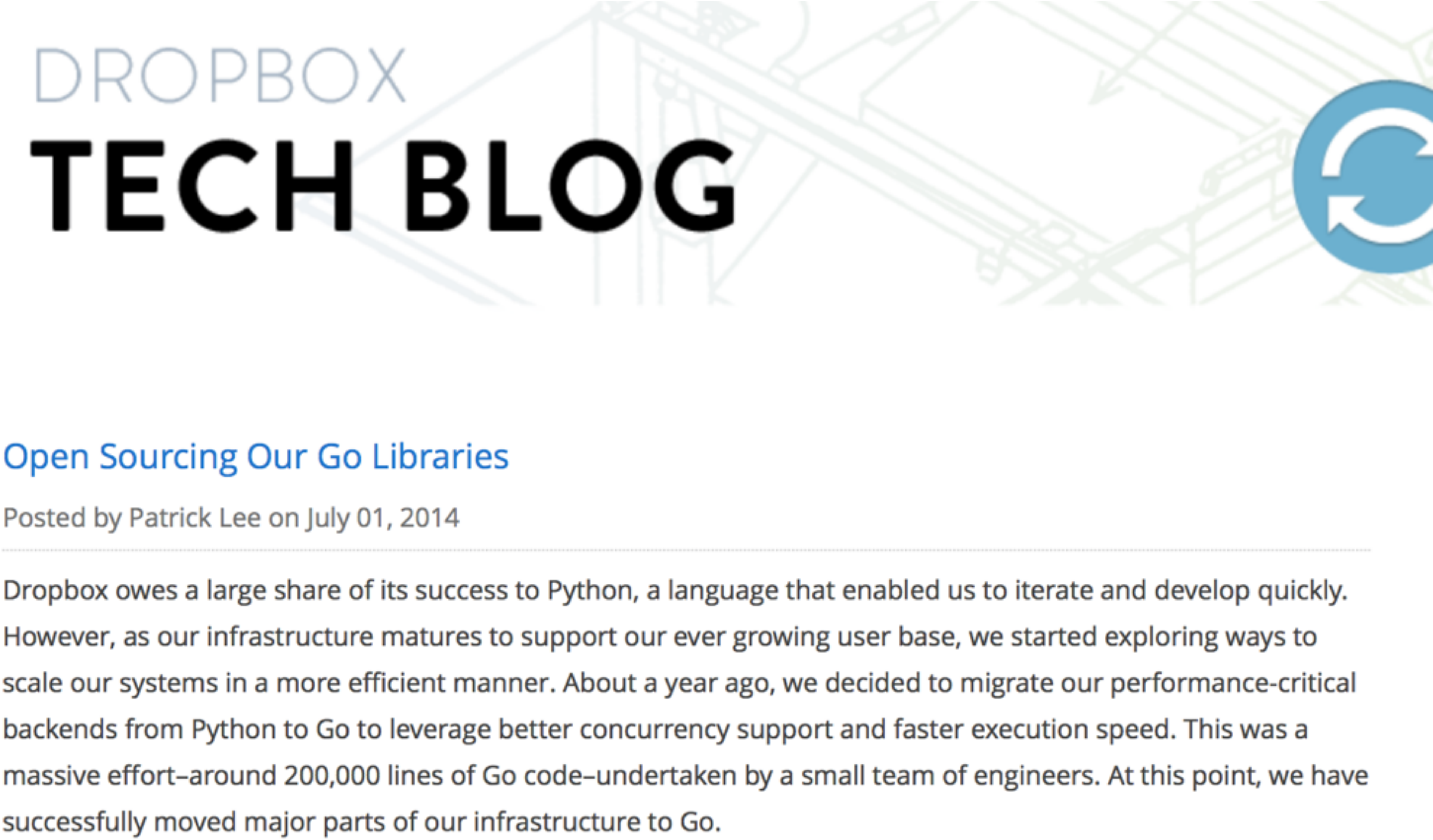
Example Go I wrote on the fly:

```go
package main
import "fmt"

func factorial(n int) int {
    var out = 1
    for i := 1; i <= n; i++ {
        out = out * i
    }
    return out
}


func approxE(k int) float64 {
    var out = 1.0
    for i := 1; i <= k; i++ {
        out = out + 1.0 / float64(factorial(i))
    }
    return out
}

func main() {
    fmt.Println(approxE(10))
}
```

Why Go?

- Modern language.
- Open source, industry supported (not a toy language like C0)
- Powerful, efficient, but relatively easy to learn.
- Far faster and more memory-efficient than Python.
- Has modern, important programming concepts like types and pointers. (Python lacks both, Java lacks pointers).
- Very easy parallel programming.
- Excellent tools and documentation.

Go is more scalable than Python



Open Sourcing Our Go Libraries

Posted by Patrick Lee on July 01, 2014

Dropbox owes a large share of its success to Python, a language that enabled us to iterate and develop quickly. However, as our infrastructure matures to support our ever growing user base, we started exploring ways to scale our systems in a more efficient manner. About a year ago, we decided to migrate our performance-critical backends from Python to Go to leverage better concurrency support and faster execution speed. This was a massive effort–around 200,000 lines of Go code–undertaken by a small team of engineers. At this point, we have successfully moved major parts of our infrastructure to Go.

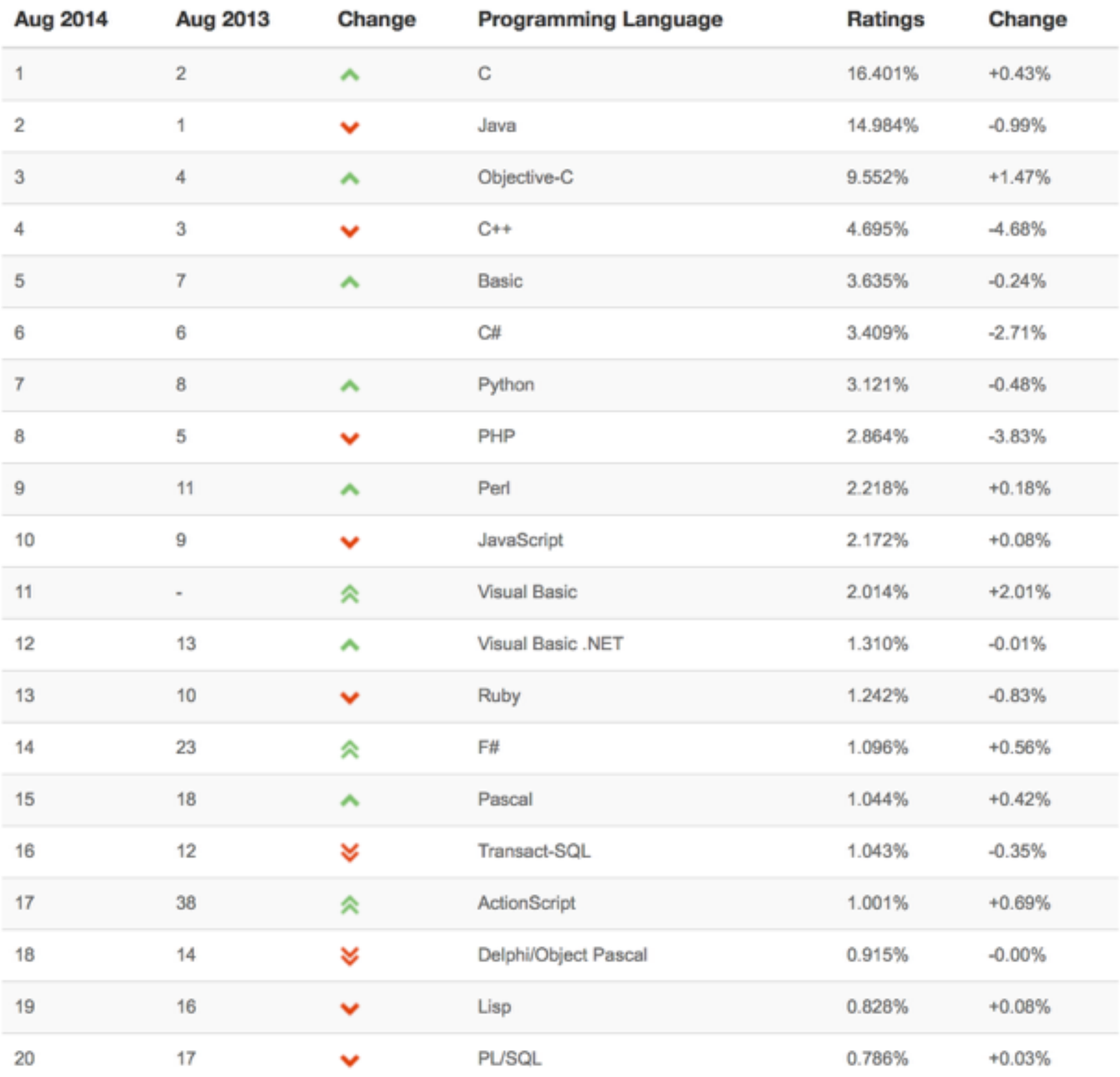
https://tech.dropbox.com/2014/07/open-sourcing-our-go-libraries/

TIOBE Programming Language Survey

There are a lot of programming languages.

Some better suited for certain jobs.

With a few exceptions, all popular languages share many similarities.



Aug 2014	Aug 2013	Change	Programming Language	Ratings	Change
1	2	▲	C	16.401%	+0.43%
2	1	▼	Java	14.984%	-0.99%
3	4	▲	Objective-C	9.552%	+1.47%
4	3	▼	C++	4.695%	-4.68%
5	7	▲	Basic	3.635%	-0.24%
6	6		C#	3.409%	-2.71%
7	8	▲	Python	3.121%	-0.48%
8	5	▼	PHP	2.864%	-3.83%
9	11	▲	Perl	2.218%	+0.18%
10	9	▼	JavaScript	2.172%	+0.08%
11	-	▲▲	Visual Basic	2.014%	+2.01%
12	13	▲	Visual Basic .NET	1.310%	-0.01%
13	10	▼	Ruby	1.242%	-0.83%
14	23	▲▲	F#	1.096%	+0.56%
15	18	▲	Pascal	1.044%	+0.42%
16	12	▼▼	Transact-SQL	1.043%	-0.35%
17	38	▲▲	ActionScript	1.001%	+0.69%
18	14	▼▼	Delphi/Object Pascal	0.915%	-0.00%
19	16	▼	Lisp	0.828%	+0.08%
20	17	▼	PL/SQL	0.786%	+0.03%

http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html

Most programming languages are not that different

Program for computing prime numbers

Go:

```go
func primeSieve(isComposite []bool) {
    var biggestPrime = 2
    for biggestPrime < len(isComposite) {
        for i := 2*biggestPrime; i < len(isComposite); i += biggestPrime {
            isComposite[i] = true
        }
        biggestPrime++
        for biggestPrime < len(isComposite) && isComposite[biggestPrime] {
           biggestPrime++
        }
    }
}
```

Python:

```python
def primeSieve(isComposite):
   biggestPrime = 2
   while biggestPrime < len(isComposite):
      for i in xrange(2*biggestPrime, len(isComposite), biggestPrime):
          isComposite[i] = True
      biggestPrime += 1
      while biggestPrime < len(isComposite) and isComposite[biggestPrime]:
          biggestPrime += 1
```

C++:

```cpp
void primeSieve(std::vector<bool> isComposite) {
    int biggestPrime = 2;
    while (biggestPrime < isComposite.size()) {
        for (int i = 2*biggestPrime; i < isComposite.size(); i += biggestPrime) {
            isComposite[i] = true;
        }
        biggestPrime++
        while (biggestPrime < isComposite.size() && isComposite[biggestPrime]) {
            biggestPrime++
        }
    }
}
```

Programming Languages Go Out of Style

You can’t get fixated on a single language —

For it’s entire history, nearly all iOS apps were developed in a language called Objective-C

In June, Apple released a new language called Swift that most new apps will be written in.

If you think of programming as being specific to a language, this is a major change.

The Goals of Programming

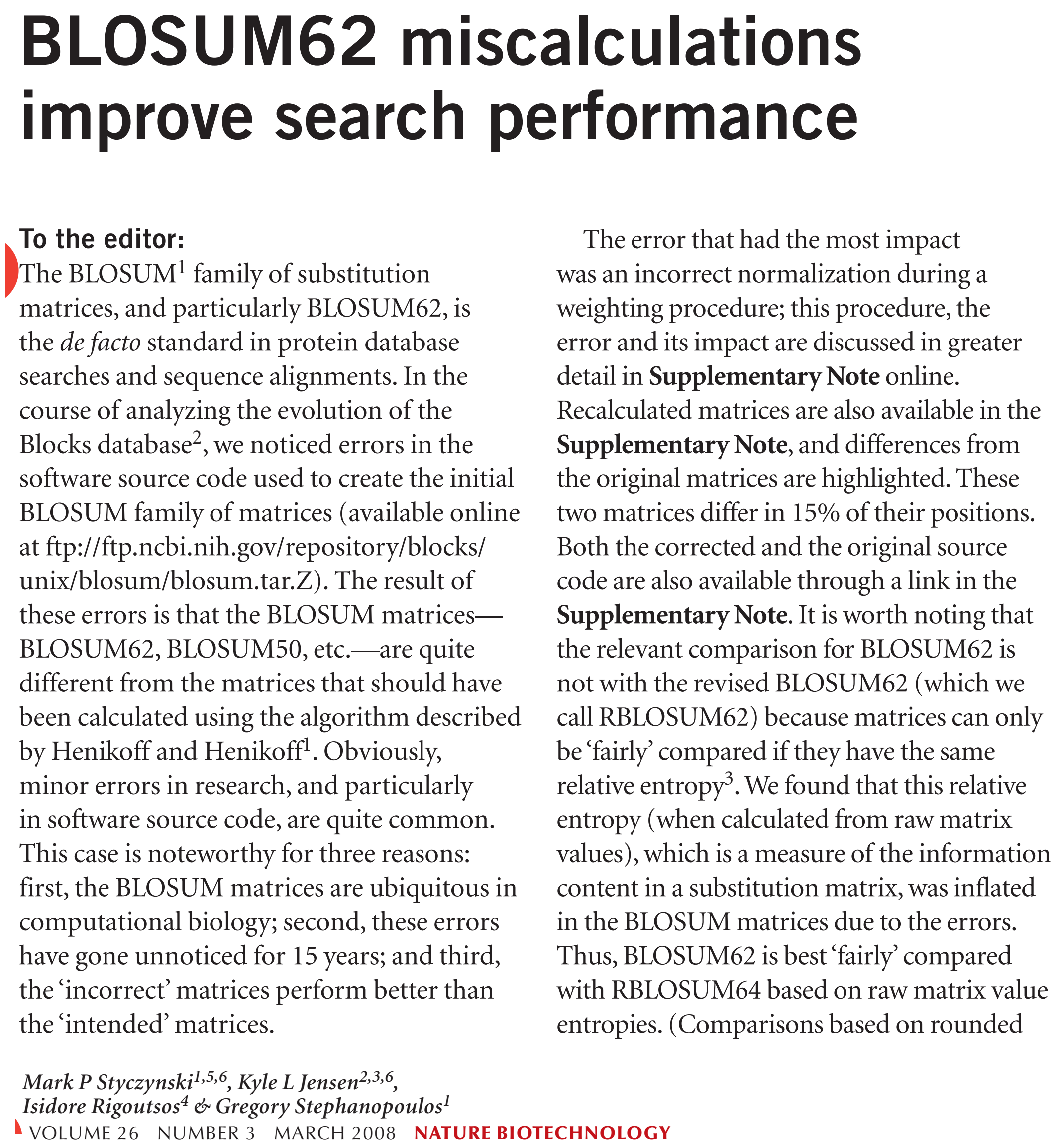
1. **Correctness** – paramount; incorrect programs aren’t worth much

2. **Maintainability** – more people will use and modify code than will write it the first time. Maintainability also leads to correctness. Function follows form.

3. **Robustness** – tolerate user errors, changes in machine configuration, changes in operating system, etc.

4. **Efficiency** – work using as few computational resources as possible.

Example Programing Mistakes (bugs)



BLOSUM62 miscalculations improve search performance

To the editor:
The BLOSUM[1] family of substitution matrices, and particularly BLOSUM62, is the *de facto* standard in protein database searches and sequence alignments. In the course of analyzing the evolution of the Blocks database[2], we noticed errors in the software source code used to create the initial BLOSUM family of matrices (available online at ftp://ftp.ncbi.nih.gov/repository/blocks/unix/blosum/blosum.tar.Z). The result of these errors is that the BLOSUM matrices—BLOSUM62, BLOSUM50, etc.—are quite different from the matrices that should have been calculated using the algorithm described by Henikoff and Henikoff[1]. Obviously, minor errors in research, and particularly in software source code, are quite common. This case is noteworthy for three reasons: first, the BLOSUM matrices are ubiquitous in computational biology; second, these errors have gone unnoticed for 15 years; and third, the 'incorrect' matrices perform better than the 'intended' matrices.

The error that had the most impact was an incorrect normalization during a weighting procedure; this procedure, the error and its impact are discussed in greater detail in **Supplementary Note** online. Recalculated matrices are also available in the **Supplementary Note**, and differences from the original matrices are highlighted. These two matrices differ in 15% of their positions. Both the corrected and the original source code are also available through a link in the **Supplementary Note.** It is worth noting that the relevant comparison for BLOSUM62 is not with the revised BLOSUM62 (which we call RBLOSUM62) because matrices can only be 'fairly' compared if they have the same relative entropy[3]. We found that this relative entropy (when calculated from raw matrix values), which is a measure of the information content in a substitution matrix, was inflated in the BLOSUM matrices due to the errors. Thus, BLOSUM62 is best 'fairly' compared with RBLOSUM64 based on raw matrix value entropies. (Comparisons based on rounded

Mark P Styczynski[1,5,6], Kyle L Jensen[2,3,6],
Isidore Rigoutsos[4] & Gregory Stephanopoulos[1]

VOLUME 26 NUMBER 3 MARCH 2008 NATURE BIOTECHNOLOGY

BLOSUM62 is a widely used algorithm to compare protein sequences, described in 1992.

In 2008, researchers reported that the code computing it contained several bugs.

Computational thinking: *testing, assuming things are wrong until you check they are right.*

A goal of the course is that you will write programs that avoid these kinds of embarrassing mistakes.

Bug: goto fail;

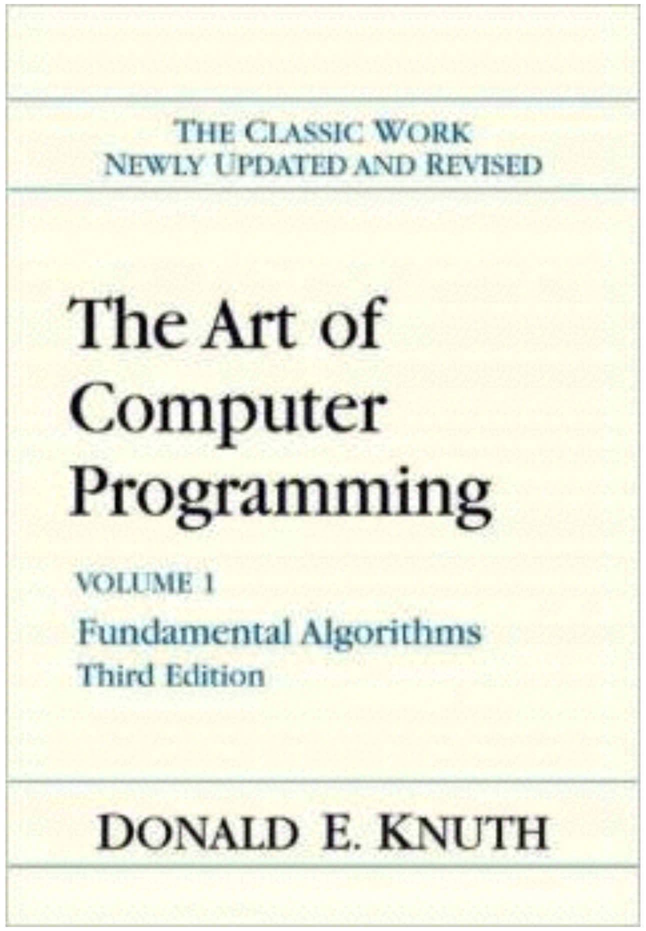
In early 2014, Apple revealed that their secure internet connection code had a huge bug that introduced a big security flaw:

```
if ((err = ReadyHash(&SSLHashSHA1, &hashCtx)) != 0)
    goto fail;
if ((err = SSLHashSHA1.update(&hashCtx, &clientRandom)) != 0)
    goto fail;
if ((err = SSLHashSHA1.update(&hashCtx, &serverRandom)) != 0)
    goto fail;
if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
    goto fail;
    goto fail; /* this code causes the SSL check to always succeed!*/
if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
    goto fail;

err = sslRawVerify(...);
```

http://opensource.apple.com/source/Security/Security-55471/libsecurity_ssl/lib/sslKeyExchange.c

Heartbleed bug in OpenSSL was also found around the same time: it allowed snooping of data in your computer’s memory.

Programming is Like Writing



THE CLASSIC WORK
NEWLY UPDATED AND REVISED

The Art of
Computer
Programming

VOLUME 1
Fundamental Algorithms
Third Edition

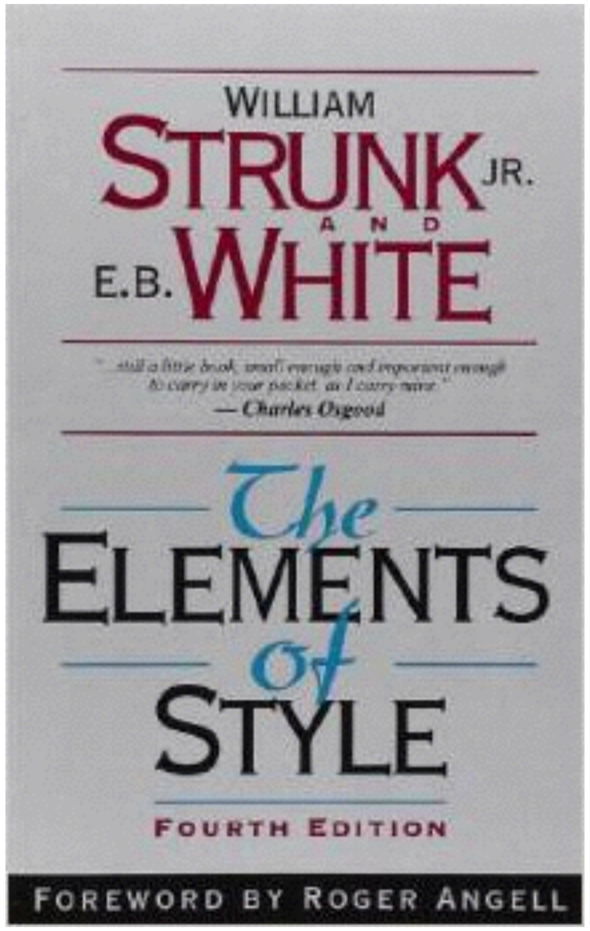
DONALD E. KNUTH

You want to express your thoughts
(in this case an algorithm) clearly.

You write, revise, edit, refine.

In a quest for:

correctness, clarity, and style



WILLIAM
STRUNK JR.
AND
E.B. WHITE

"...still a little book, small enough and important enough
to carry in your pocket, as I carry mine."
— Charles Osgood

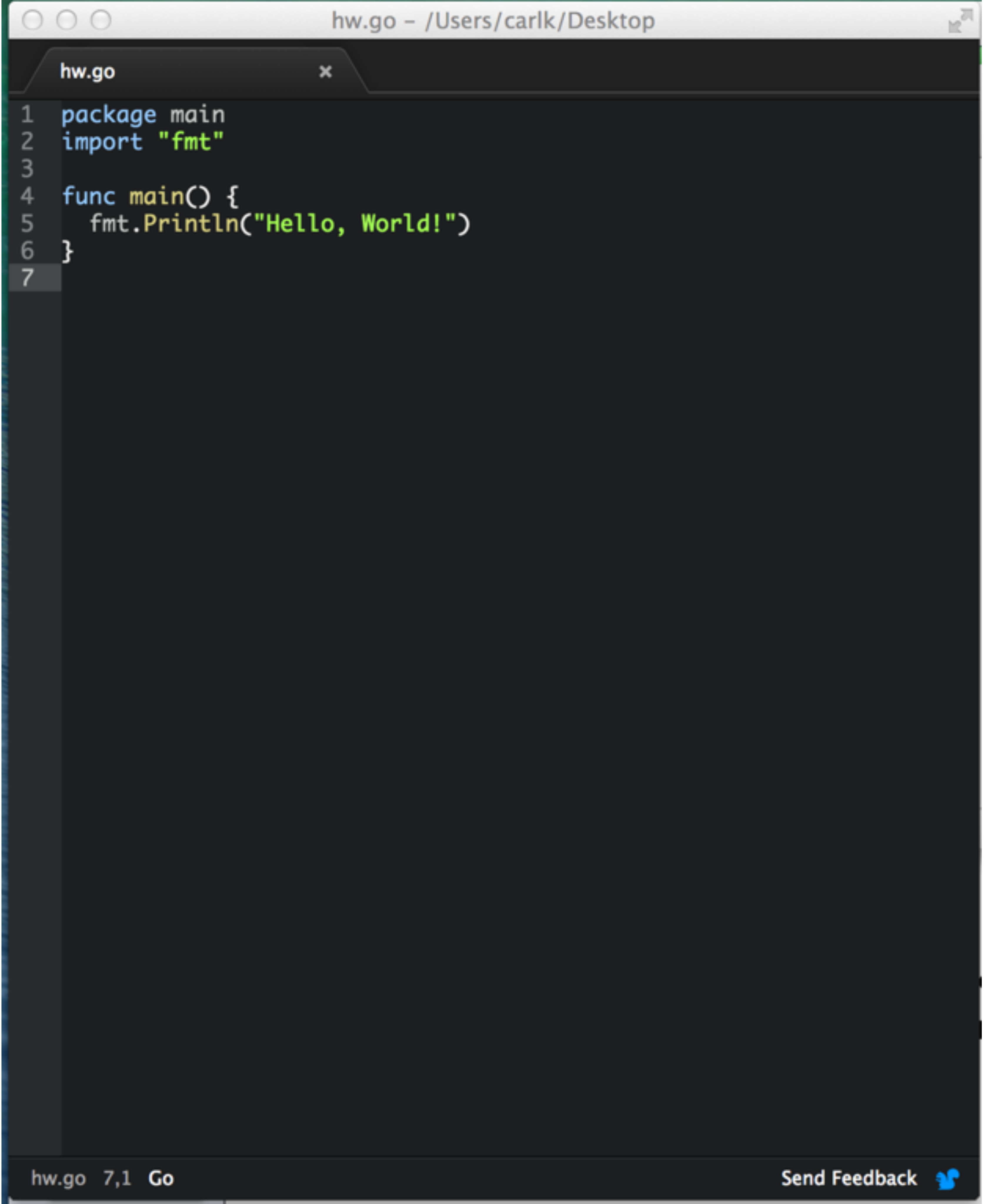
The
ELEMENTS
of
STYLE

FOURTH EDITION

FOREWORD BY ROGER ANGELL

Both your English writing and your code should be precise & concise.

Ways to Run Go



```go
package main
import "fmt"

func main() {
   fmt.Println("Hello, World!")
}

```

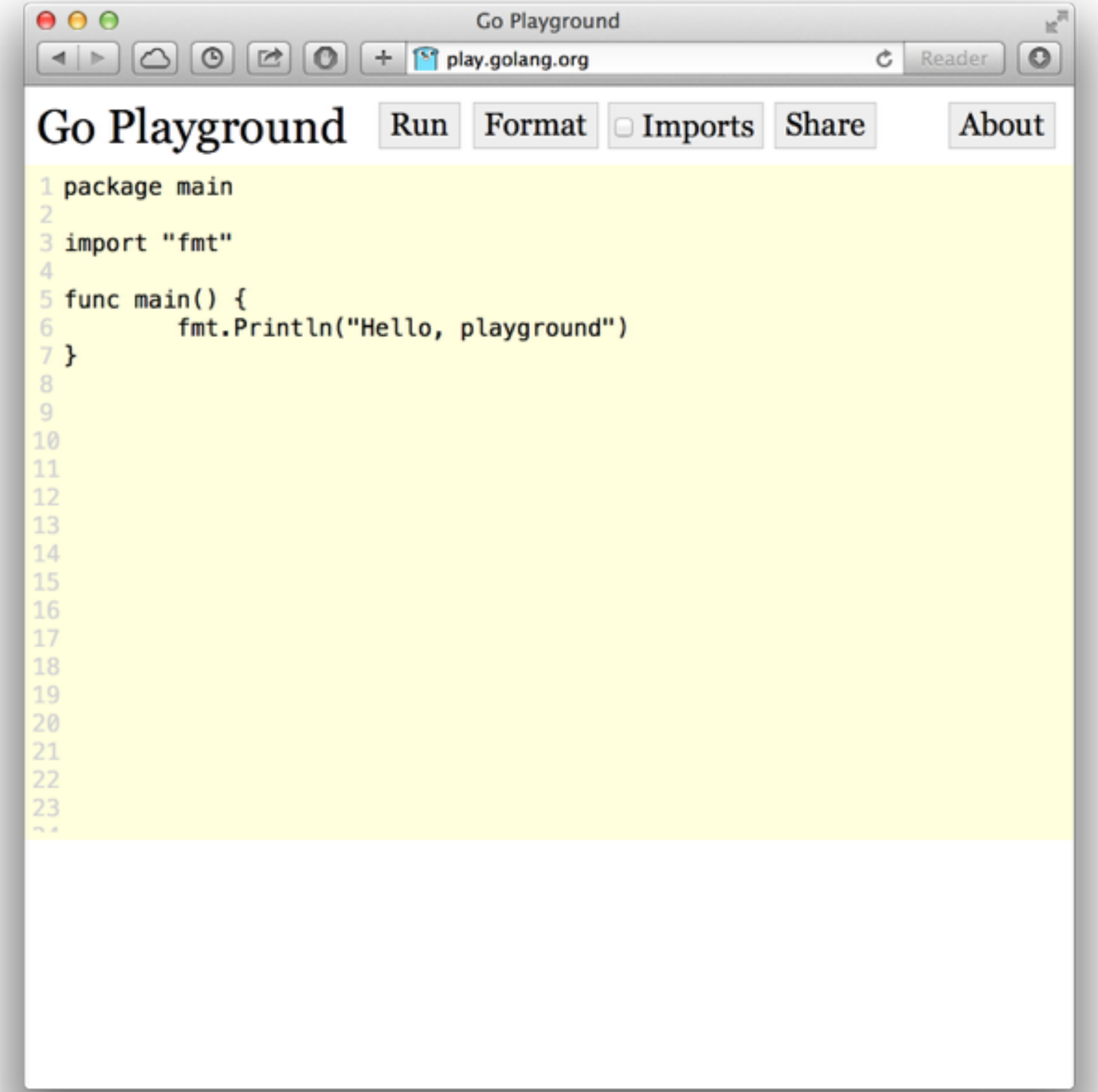
1. Type your program into a **text** file whose name ends with ".go"

2. Then compile and run your program from the command line (cmd in Windows, Terminal in OS X):

```
go run hw.go
```

playground
go run foo.go
go build
go test

Using the Go Playground



```go
package main

import "fmt"

func main() {
	fmt.Println("Hello, playground")
}
```

1. Type your program into the web form at:

   http://play.golang.org

2. Press “run”

   `go run hw.go`

Building Executables You Can Share

```
go/
  src/
    username/
      computePrimes/
         primes.go
         utils.go
      gcd/
         gcd.go
         bigint.go
         util.go
  bin/
  pkg/
```

1. Create the green directory structure.

2. Create a directory for every program you want to write (can add more later, of course)

3. Write your program, splitting into as many .go files as you want under the program directory.

4. "cd" into the program directory and run

```
go build
```

this will produce an executable with the name = to the directory name.

5. Run this program with: `./computePrimes`

Overview of the Class

- First ≈3 weeks or so: learn most of Go, write some programs in it. Learn fundamental programming concepts.

- Data structures: stacks, queues, lists, binary search trees, heaps, graphs.

- Software engineering: version control, unit testing, profiling, coverage tests, documentation, style, good interface design.

- "Object-oriented programming": a style of programming aimed at controlling complexity.

- Parallelism: Co-routines, using multiple processors at once, locks, and threads.

- C++: Another view of object-oriented programming, memory management, generics (i.e. template programming).

- Python: Comparison with Go and C++.

See syllabus for more details!

Grading

- 8 - 12 programming assignments: 70% of your grade

- You must work on program assignments **on your own** (unless otherwise noted).
  - **You cannot look at other’s code or share code.**
  - You may discuss the problems in a general way without sharing code.

- Programs will be submitted to Autolab and graded automatically for correctness.

- An additional part of your programming grade will be good style and design.

Midterm and Final

- Midterm counts for 15%
- Final counts for 15%
- Midterm will be in class.
- Final will be held at the time scheduled by the university.

Resources

Book: free online at: http://www.golang-book.com

The Go Programming Language Specification

http://golang.org/ref/spec

A Tour of Go

http://tour.golang.org/#1

Effective Go

http://golang.org/doc/effective_go.html

Office Hours / Help Sessions

- We will have 3-4 special “help sessions” per week.
- These are entirely optional.
- During the first few minutes, a TA will go over something from the topic of the week that would benefit from another example or some more discussion.
- Then the TA will take questions and provide individual help on homeworks, etc.

Homework 0: Due Thursday

1. Complete the background survey on BlackBoard.

2. Please find a way to run Go:
   - on your personal computer, (see http://golang.org/doc/install)
   - through the CMU Linux Timeshares, (see http://www.cmu.edu/computing/clusters/software/timeshares.html)
   - or in a lab (BH 140F, Wean 5207)
   - (or all 3)!

3. Work through the examples here: http://golang.org/doc/code.html

4. Read chapters 1-3 of An Introduction to Programming in Go by Caleb Doxsey: http://www.golang-book.com

Summary

- Programming is central to science.
- The “Go” language is efficient, expressive, and modern.
- Learning Go will make it easy to learn many other languages.
- Programming languages come and go, but nearly all of the work of programming has nothing at all to do with the notation.