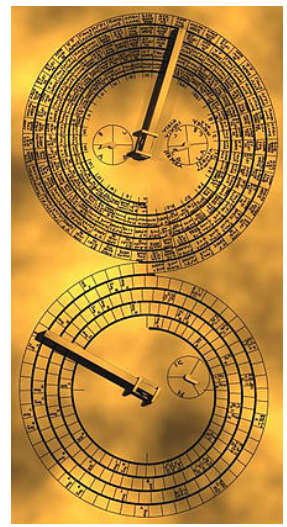
Compilers and Formal Languages (1)



Antikythera automaton, 100 BC (Archimedes?)

Email: christian.urban at kcl.ac.uk

Office: S1.27 (1st floor Strand Building)

Slides: KEATS

The Goal of this Course

Write A Compiler

The Goal of this Course

lexer input: a string

```
"read(n);"
```

lexer output: a sequence of tokens

```
key(read); lpar; id(n); rpar; semi
```

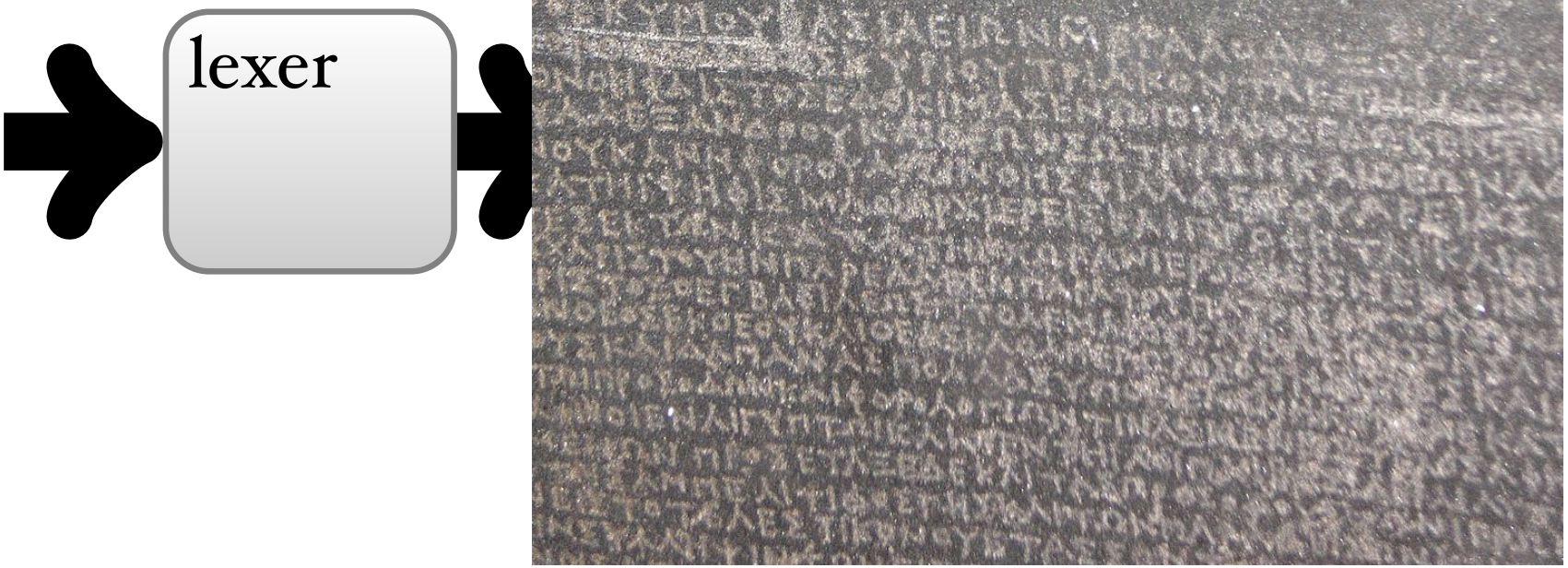
The Goal of this Course

lexer input: a string

```
"read(n);"
```

lexer output: a sequence of tokens

```
key(read); lpar; id(n); rpar; semi
```

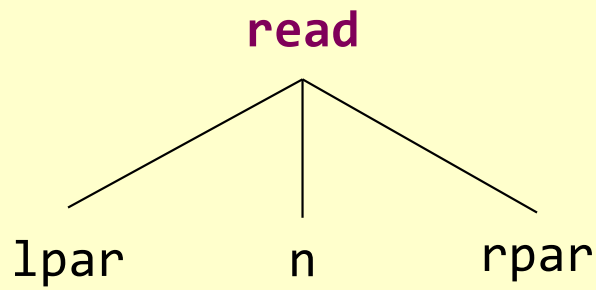


lexing ⇒ recognising words (Stone of Rosetta)

The Goal of this Course

parser input: a sequence of token

parser output: an abstract syntax tree

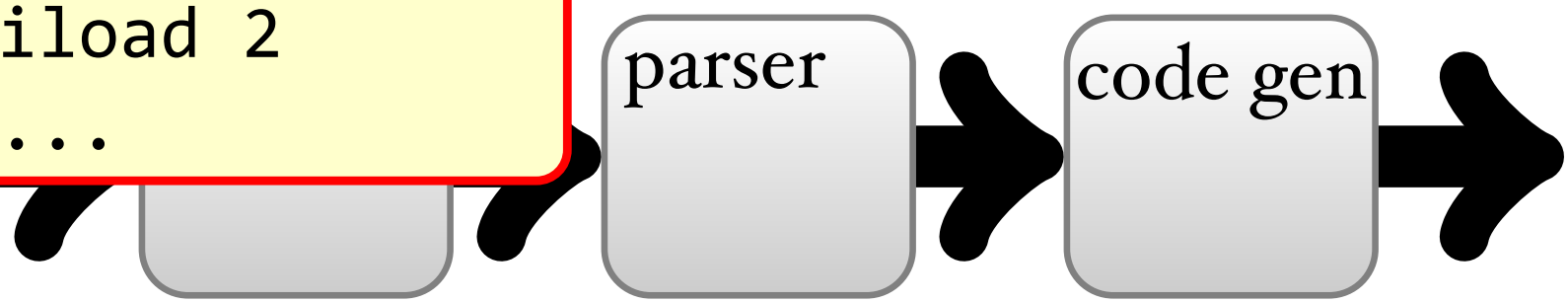


The Goal of this Course

code generator:

```
istore 2
iload 2
ldc 10
isub
ifeq Label2
iload 2
...
```

e A Compiler



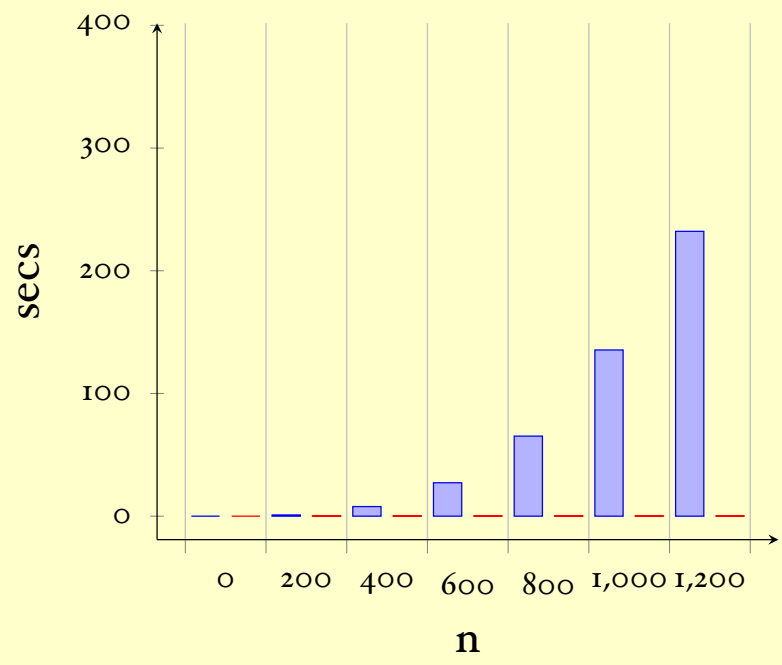
The Goal of this Course

code generator:

```
istore 2
iload 2
ldc 10
isub
ifeq Label2
iload 2
...
```

e A Compiler

parse



The subject is quite old

- Turing Machines, 1936
- Regular Expressions, 1956
- The first compiler for COBOL, 1957 (Grace Hopper)
- But surprisingly research papers are still published nowadays

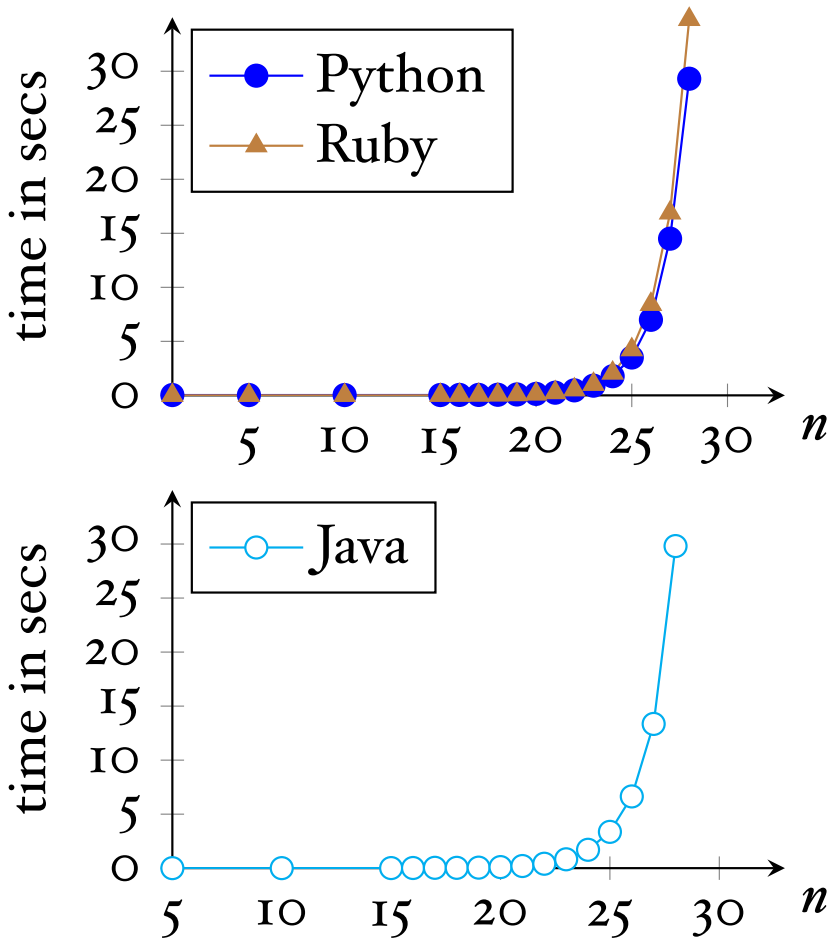
Grace Hopper

(she made it to David Letterman's Tonight Show,

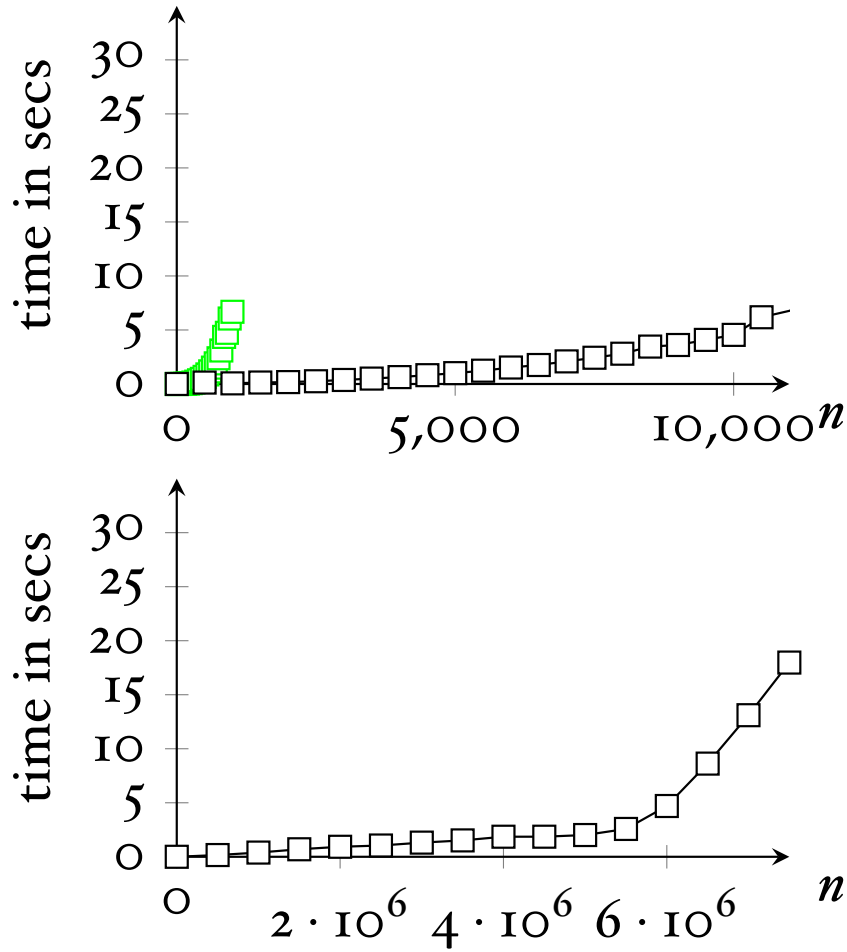
`http://www.youtube.com/watch?v=aZOxtURhfEU`)

Why Bother?

Ruby, Python, Java



Us (after next lecture)



matching `[a?]{n}[a]{n}` and `[a*]*b` against $\underbrace{\texttt{a...a}}_{n}$

Lectures 1 - 5

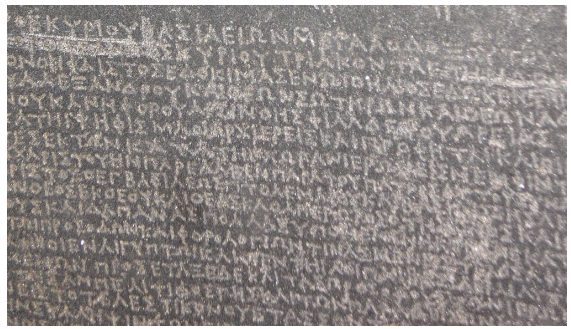
transforming strings into structured data

Lexing

(recognising "words")

Parsing

(recognising "sentences")



Stone of Rosetta

Lectures 1 - 5

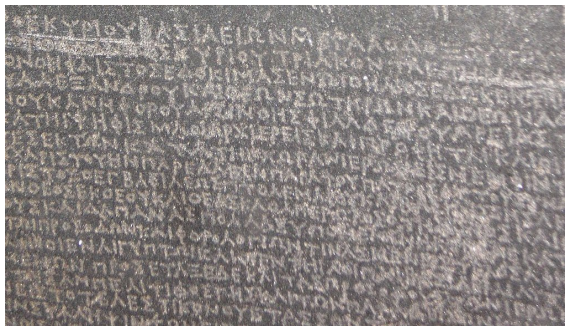
transforming strings into structured data

Lexing based on regular expressions

(recognising "words")

Parsing

(recognising "sentences")



Stone of Rosetta

Familiar Regular Expr.

```
[a-z0-9_.-]+ @ [a-z0-9.-]+ . [a-z.]{2,6}
```

`re*`	matches 0 or more times
`re+`	matches 1 or more times
`re?`	matches 0 or 1 times
`re{n}`	matches exactly `n` number of times
`re{n,m}`	matches at least `n` and at most `m` times
`[...]`	matches any single character inside the brackets
`[^...]`	matches any single character not inside the brackets
`a-zA-Z`	character ranges
`\d`	matches digits; equivalent to `[0-9]`
`.`	matches every character except newline
`(re)`	groups regular expressions and remembers the matched text

Today

- While the ultimate goal is to implement a small compiler (a really small one for the JVM)...

Let's start with:

- a web-crawler
- an email harvester
- (a web-scraper)

A Web-Crawler

1. given an URL, read the corresponding webpage
2. extract all links from it
3. call the web-crawler again for all these links

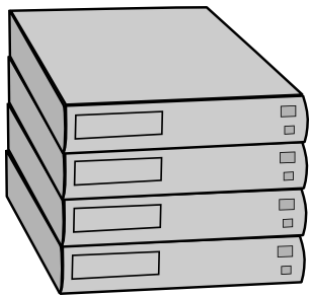
A Web-Crawler

1. given an URL, read the corresponding webpage
2. if not possible print, out a problem
3. if possible, extract all links from it
4. call the web-crawler again for all these links

A Web-Crawler

1. given an URL, read the corresponding webpage
2. if not possible print, out a problem
3. if possible, extract all links from it
4. call the web-crawler again for all these links

(we need a bound for the number of recursive calls)
(the purpose is to check all links on my own webpage)



Server

GET request

webpage

POST data

Browser

Scala

A simple Scala function for reading webpages:

```
import io.Source

def get_page(url: String) : String = {
  Source.fromURL(url).take(10000).mkString
}
```

Scala

A simple Scala function for reading webpages:

```
import io.Source

def get_page(url: String) : String = {
  Source.fromURL(url).take(10000).mkString
}

get_page("""http://www.inf.kcl.ac.uk/staff/urbanc/""")
```

Scala

A simple Scala function for reading webpages:

```
import io.Source

def get_page(url: String) : String = {
  Source.fromURL(url).take(10000).mkString
}

get_page("""http://www.inf.kcl.ac.uk/staff/urbanc/""")
```

A slightly more complicated version for handling errors:

```
def get_page(url: String) : String = {
  Try(Source.fromURL(url).take(10000).mkString).
     getOrElse { println(s"  Problem with: $url"); ""}
}
```