REXX—THE FUTURE

MIKE COWLISHAW
IBM

REXX—The Future

Mike Cowlishaw

IBM UK Laboratories
Hursley

13 May 1993

The Future of REXX

- REXX usage today
- Hardware speed and REXX
- REXX Top Ten language requests
- Trends and directions
- Discussion

REXX usage today

- 18 commercial implementations, on most significant platforms
- 41 published books and manuals

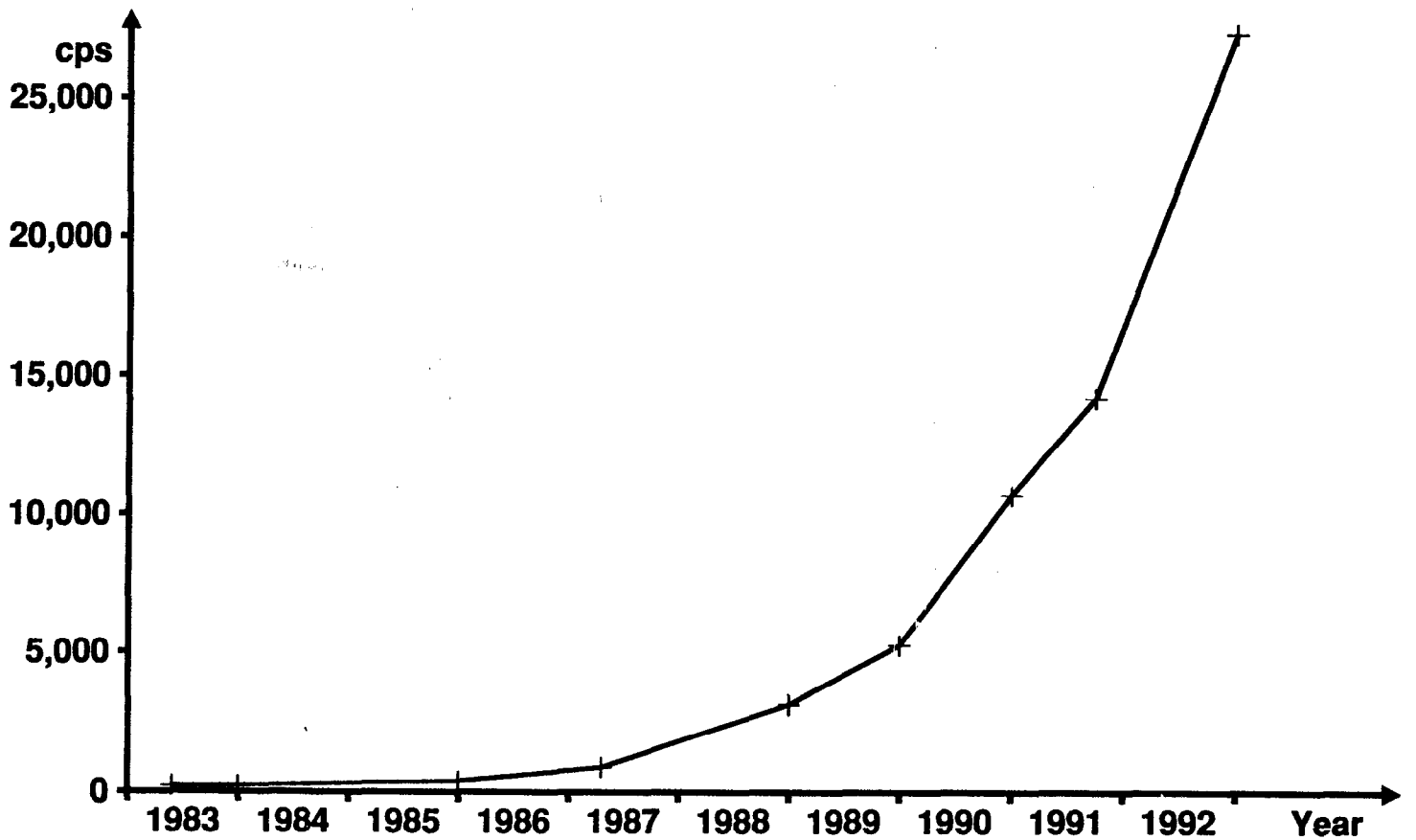
  (Nearly 60, if service guides, second editions, and translations are included.)
- Accessible to well over ten million users
- ANSI (X3J18) standard work well under way.

Hardware speed and REXX

As hardware speed increases, REXX is being used for a wider set of applications. Some informal figures:

- *x86* systems—now over 27,000 REXX clauses per second (486/66)
- RISC systems—over 42,000 REXX cps (same interpreter)
- Mainframe systems—over 90,000 REXX cps
- REXX Compiler/370—up to 465,000 REXX cps

REXX Clauses per Second—x86 platform



Top Ten language improvements

Caveats...

- This is a personal list, and does not describe any vendor's product plans (as far as I know)
- Object-Oriented extensions are not included (though these would probably be a superset of this list)
- Don't read too much into the order.

10. DIGITS condition

Allows the trapping of unexpectedly "overprecise" numeric data:

```
numeric digits 5
signal on digits
arg a
say a+1
```

... would raise the condition if the value of A had more than five digits.

9. Expressions in stem references

Today's idiom:

```
nextj=j+1
nextk=k+1
say fred.nextj.nextk
```

16

...could be written as...

```
say fred.[j+1, k+1]
```

or, perhaps

```
say fred.(j+1).(k+1)
```

7 & 8. PARSE enhancements

Two improvements:

```
parse caseless .....
```

...like `parse`, but strings will match, even if they have a different mix of uppercase and lowercase

```
parse lower .....
```

...like `parse upper`, but translates to lower case first

6. Variable CALL target

An "indirect" call:

```
where='anyname'
call (where) a, b
```

...would call the routine "anyname".

5. Change and count functions

```
needle='is'
haystack='This is the third'
new='at'

say countstr(needle, haystack)
say changestr(needle, haystack, new)
```

...would display '2' and 'That at the third'.

4. Call by Reference

Introduces aliasing to the language:

```
call fred p, q+1, r
     .
     .
fred: procedure
  use alias a, ,c
```

or

```
fred: procedure
  use arg (a), b, (c)
```

3 External Procedure Expose

Allow "externalization" and sharing of many more routines, by permitting `procedure expose...` at the start of external routines.

Inheritance of NUMERIC DIGITS and other appropriate settings must be implied by this use.

21

2. Extended DO

Iterate over the tails of a compound variable:

```
do tail over fred.
  say fred.tail
  end tail
```

22

...would display all the values held in variables whose names begin with "FRED."

Changing any FRED.xxx variable while in the loop would be an error.

1. Date and time conversions

Almost every programmer needs these at some time...

```
say date('usa', 19930827, 'standard')
```

...would display `'08/27/93'`.

23 For conversions from years with only two digits to those with four digits, the result nearest to the current year would be used, taking into account only the year and using the earlier date if a tie.

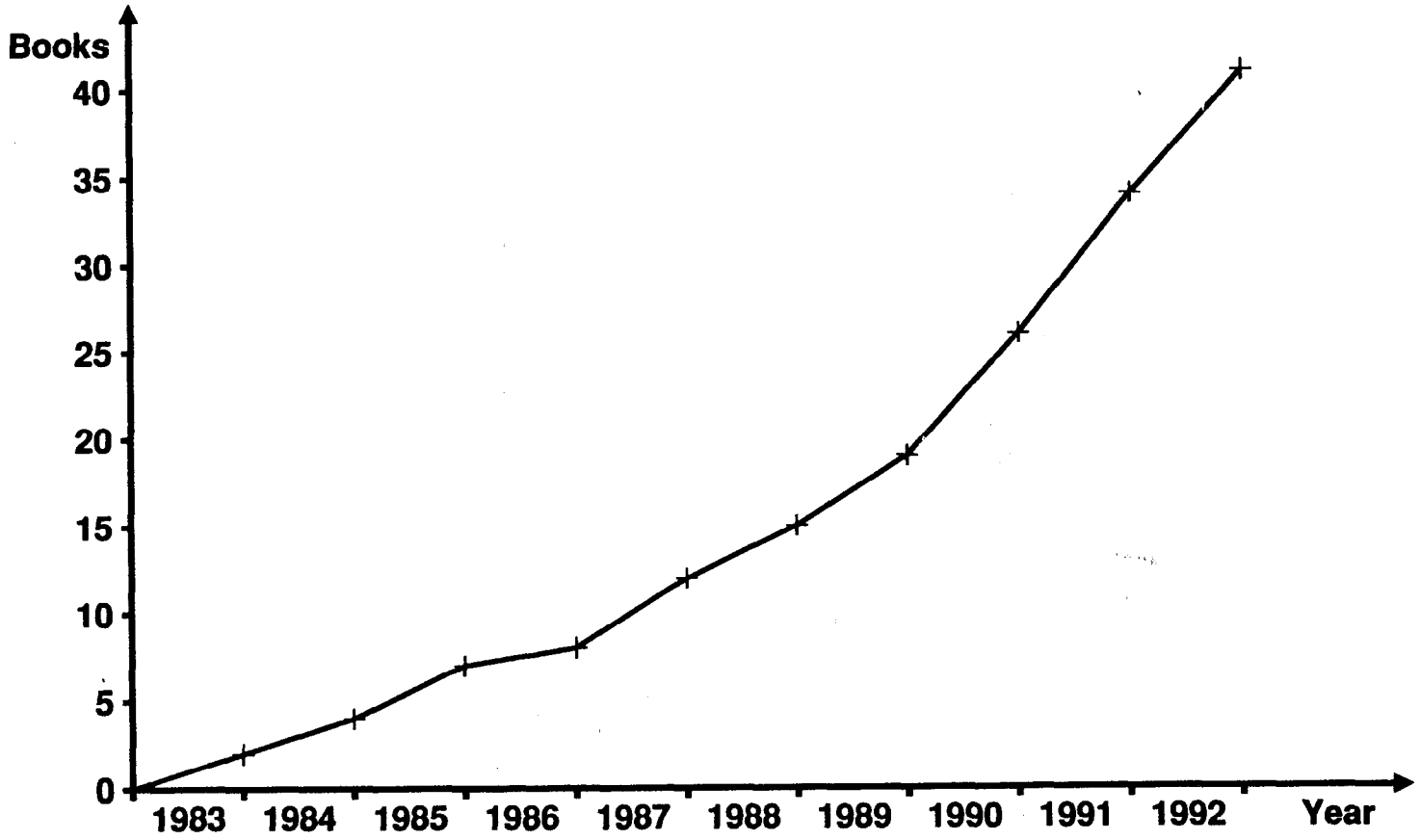
...and similarly for the TIME built-in function.

Trends and directions—Applications

- Mainframe interactive applications continue to move to the desktop
- Networking of workstations and PCs encourages standardization of applications and languages
- Increasing complexity and sophistication of applications leads users to demand extensive subsetting and customization
- Object-oriented prototype shows that REXX will be a first-class object-oriented language.

24

REXX Books and Manuals



The Future of REXX

- REXX usage today
- REXX assets
- Trends and directions
- Discussion

Which REXX assets are the most important?

- Simplicity:
  - A small, readable, language
  - Just one data type—the string
  - Decimal arithmetic
  - Few limits

More assets...

- Flexible and extendible
  - Existing and future system interfaces
  - Object-oriented extensions fit naturally

28

More assets...

- *Designed* as a multi-purpose extension language
  - Highly system and hardware independent
  - Keywords reserved only in context, so macros in source form are resistant to breakage
  - Adds value to almost all platforms and applications
- Skills reuse between platforms
  - Reduced education costs.