The forgotten ideas in computer science

Joe Armstrong

The
ideas for might have forgotten
or never knew
or
needed reminding
or
should have forgotten
in
computer science

Joe Armstrong

Work in progress
with a lot of
personal bias

Part 1
motivation

Problems (1980’s) ?

- How to find things
- How to store things
- How to program things

My questions early 1980’s

Plan

- Learn emacs
- Learn unix
- Learn a programming language

What happened?

- I didn’t learn emacs
- I didn’t learn unix
- I created a programming language

Some Progress
(after 30 years)

- Finding things
  Google and friends (but we find the wrong stuff)

- Saving things
  Dropbox and friends (but it not forever, only as long as your credit card keeps up the payments)

- Programming things
  Some small improvements - nothing dramatic
  The last new thing was Prolog - no major improvements since then.

What problems should we solve now?

Problems (2016) ?

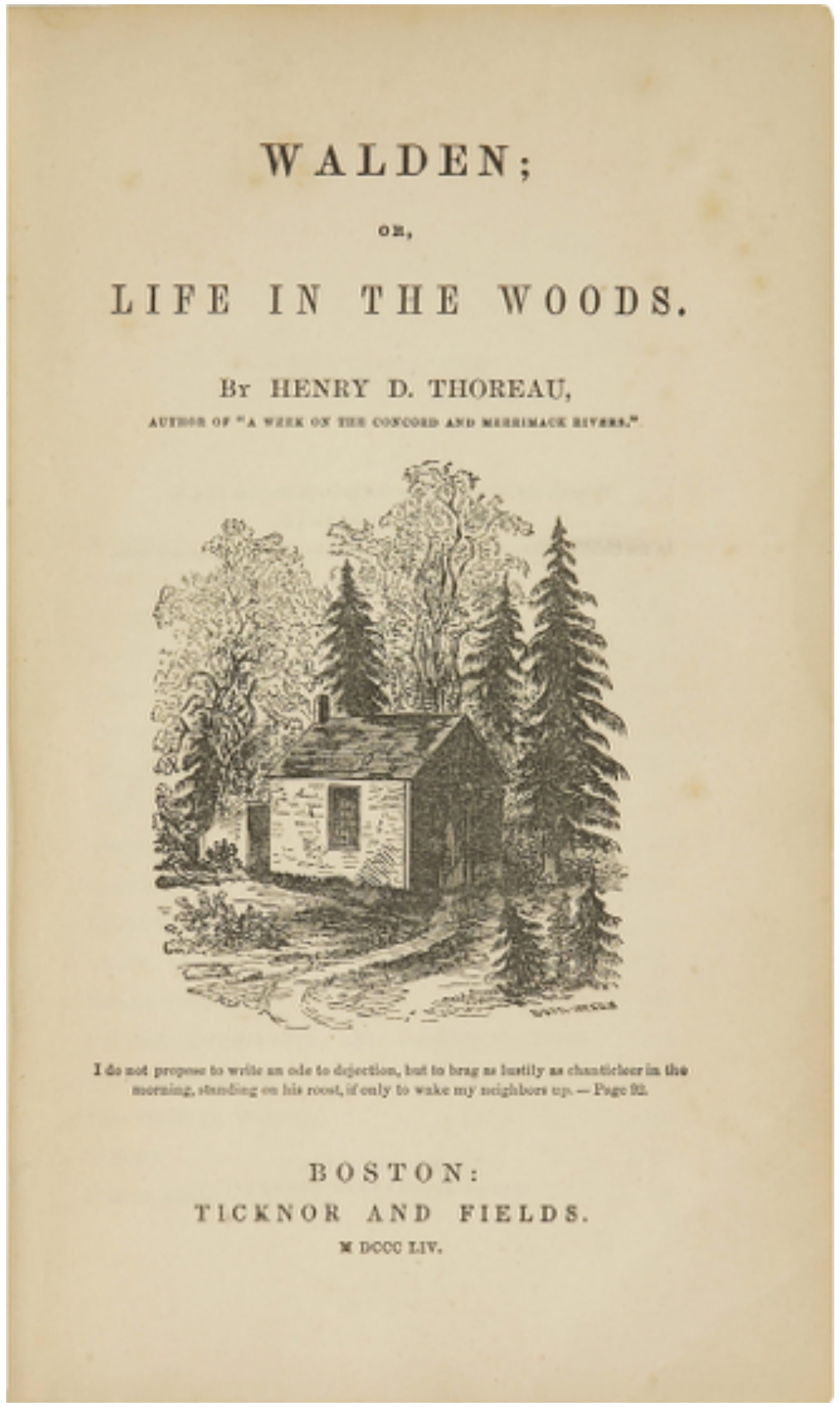
- There is too much stuff
  How can we get rid of most of it

- We’ve invented all this stuff what the heck are we going to do with it all.
  Do we really need IoT? Is it a good idea to **hijack our attention systems** every 30 seconds?

Trump Sneezed

IMPORTANT BREAKING NEWS

This is not a new problem



WALDEN;

OR,

LIFE IN THE WOODS.

By HENRY D. THOREAU,

AUTHOR OF "A WEEK ON THE CONCORD AND MERRIMACK RIVERS."

I do not propose to write an ode to dejection, but to brag as lustily as chanticleer in the morning, standing on his roost, if only to wake my neighbors up. — Page 92.

BOSTON:

TICKNOR AND FIELDS.

M DCCC LIV.

As with our colleges, so with a hundred "modern improvements"; there is an illusion about them; there is not always a positive advance … Our inventions are wont to be pretty toys, which distract our attention from serious things.

Henry David Thoreau
Walden (1854)

“To a philosopher all news, as it is called, is gossip, and they who edit and read it are old women over their tea …”

“… and as for England, almost the last significant scrap of news from that quarter was the revolution of 1649.”

Henry David Thoreau
Walden (1854)

Methodology

- Ask some questions
- Get some replies
- Organise the result
- Choose the best things to do

Questions

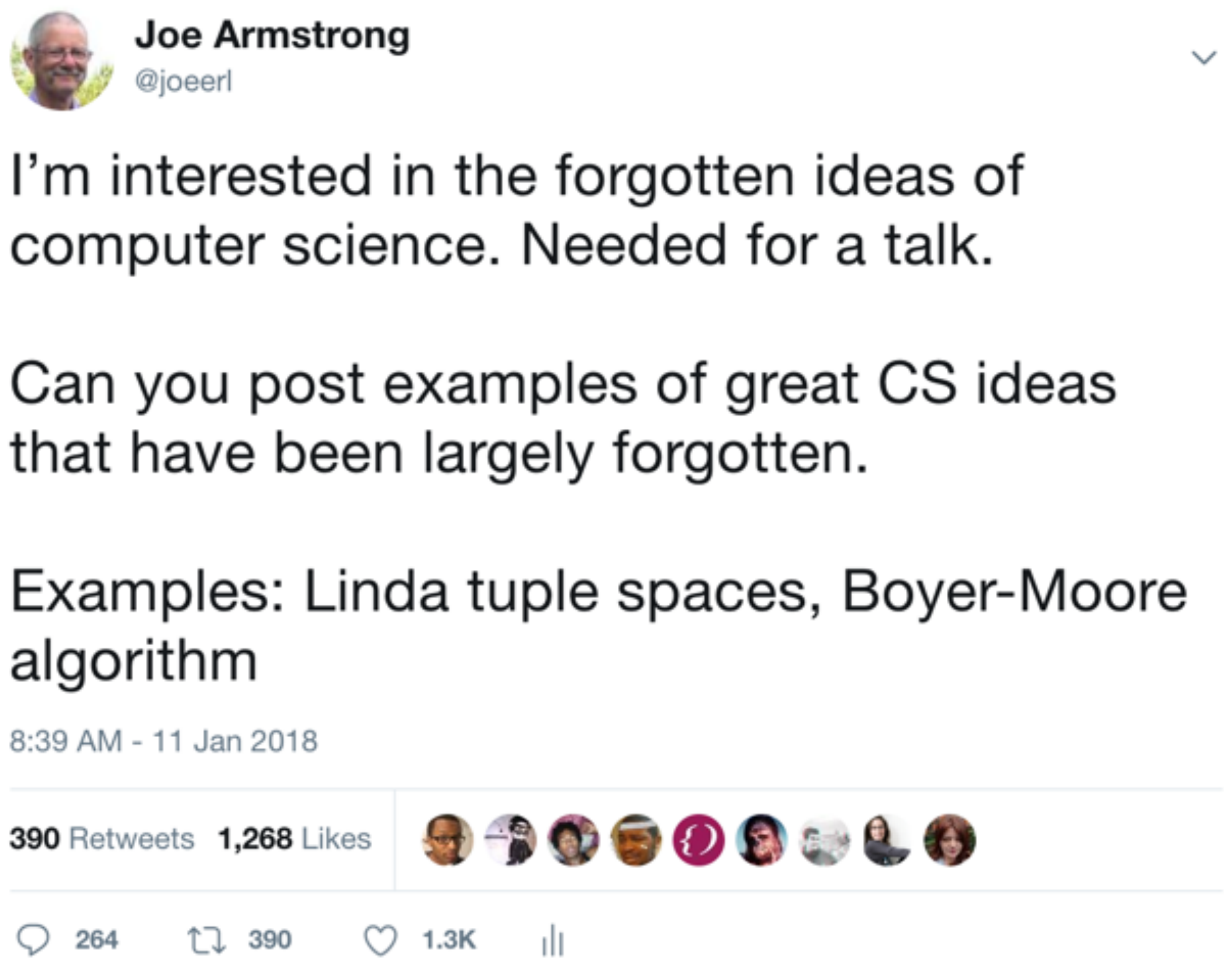
Joe Armstrong & Alan Kay - Joe Armstrong interviews Alan Kay

26,882 views

What ideas has we forgotten?

The forgotten ideas of computer science

- Ask some well-known computer scientists
- Ask all the Professors of CS that I know
- Ask all my friends who are old and programmers
- Think a lot about what I’ve forgotten



Joe Armstrong
@joeerl

I'm interested in the forgotten ideas of computer science. Needed for a talk.

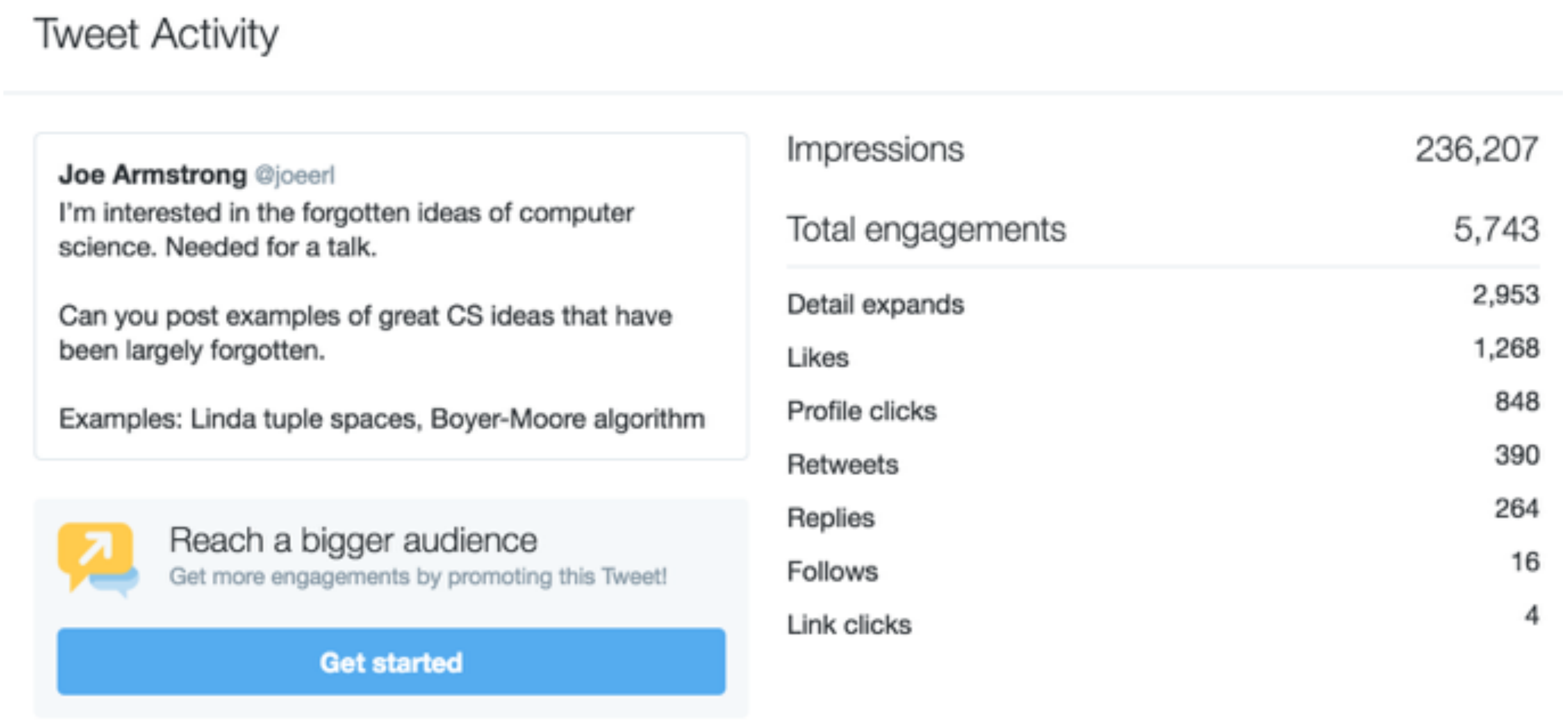
Can you post examples of great CS ideas that have been largely forgotten.

Examples: Linda tuple spaces, Boyer-Moore algorithm

8:39 AM - 11 Jan 2018

390 Retweets 1,268 Likes

264 390 1.3K



Tweet Activity

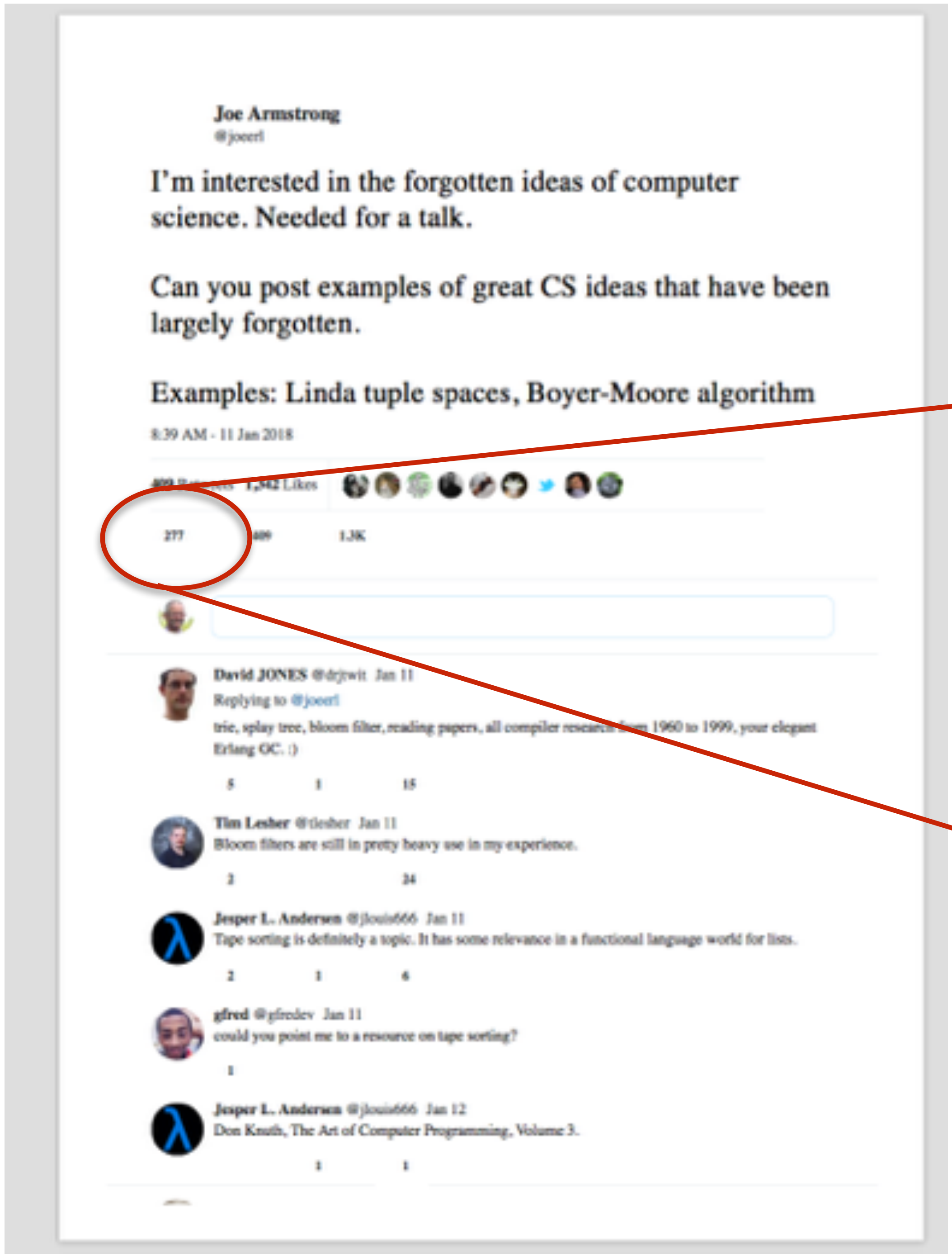
Joe Armstrong @joeerl
I'm interested in the forgotten ideas of computer science. Needed for a talk.

Can you post examples of great CS ideas that have been largely forgotten.

Examples: Linda tuple spaces, Boyer-Moore algorithm

Impressions	236,207
Total engagements	5,743
Detail expands	2,953
Likes	1,268
Profile clicks	848
Retweets	390
Replies	264
Follows	16
Link clicks	4

And on the next day



Joe Armstrong
@joeerl

I'm interested in the forgotten ideas of computer science. Needed for a talk.

Can you post examples of great CS ideas that have been largely forgotten.

Examples: Linda tuple spaces, Boyer-Moore algorithm

8:39 AM - 11 Jan 2018

409 Retweets **1,342** Likes

277 409 1.3K

David JONES @drjtwit Jan 11
Replying to @joeerl
trie, splay tree, bloom filter, reading papers, all compiler research from 1960 to 1999, your elegant Erlang GC. :)
5 1 15

Tim Lesher @tlesher Jan 11
Bloom filters are still in pretty heavy use in my experience.
2 24

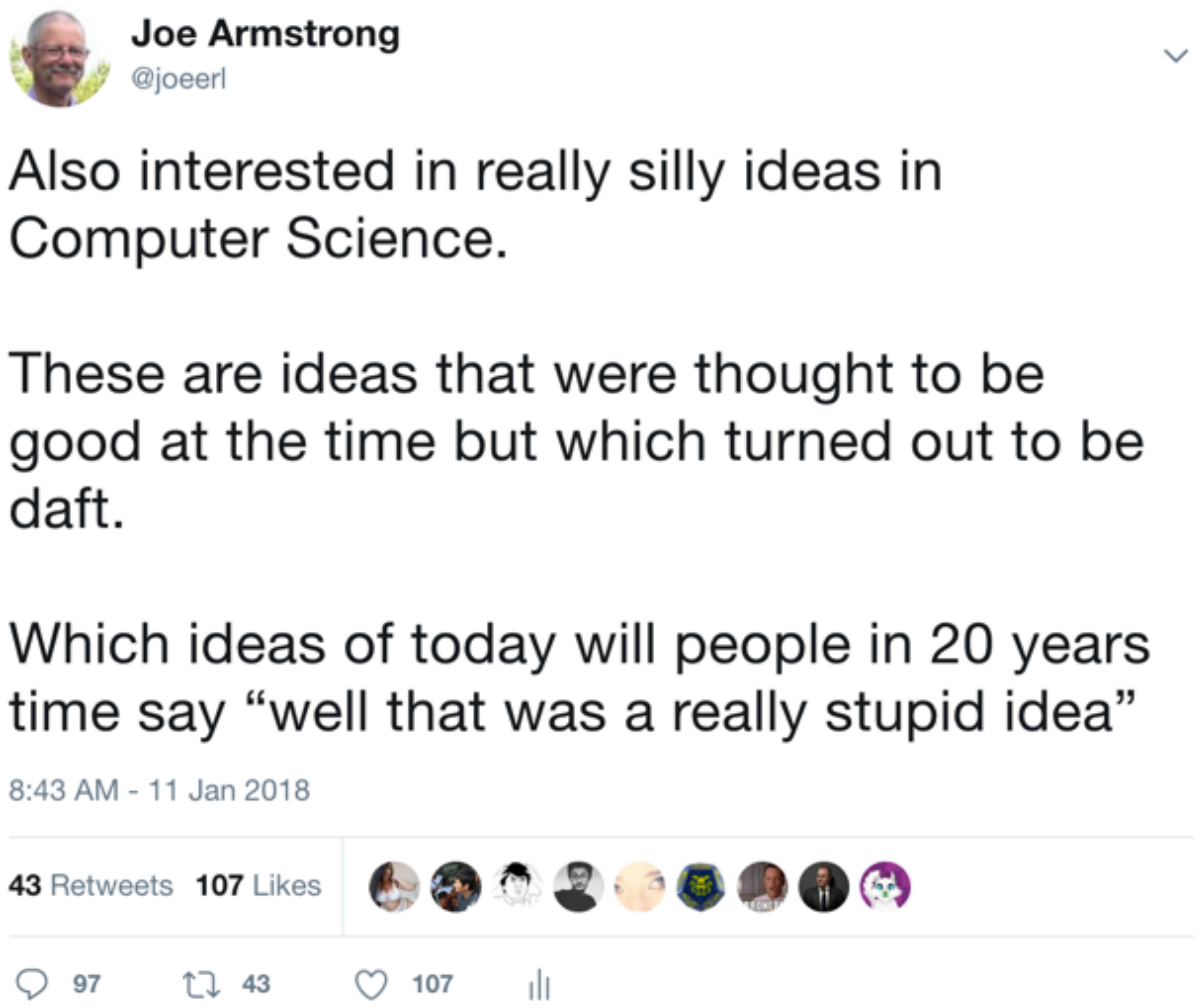
Jesper L. Andersen @jlouis666 Jan 11
Tape sorting is definitely a topic. It has some relevance in a functional language world for lists.
2 1 6

gfred @gfredev Jan 11
could you point me to a resource on tape sorting?
1

Jesper L. Andersen @jlouis666 Jan 12
Don Knuth, The Art of Computer Programming, Volume 3.
1 1

277

So I asked more questions



Joe Armstrong
@joeerl

Also interested in really silly ideas in Computer Science.

These are ideas that were thought to be good at the time but which turned out to be daft.

Which ideas of today will people in 20 years time say “well that was a really stupid idea”

8:43 AM - 11 Jan 2018

43 Retweets 107 Likes

97 43 107

What started as
“Forgotten ideas”
became

- Forgotten ideas
- Silly ideas
- Hot research topics
- Problems that non-programmers have
- Money making ideas
- Bad ideas
- Socially good projects
- Voluntary projects
- Fun ideas
- Crazy Ideas

So what started
as forgotten ideas

became a

Lists of topics

or

The essential guide
to computer science
(what you need to learn)
or
A guide for the confused

How to make a list

- Collect lots of items easy
- Assign to lists difficult
- Shorten the lists to N items (N is small) very difficult Throwing things away is much more difficult than collecting things - but what's left is better.

Part 2
Things to learn

Essential Guide to CS

- 80 things in 18 categories
  (some old, some new, some forgotten)
- Pix and Mix
- Not all equally important

  I'll talk about the most important ones
  later

80 things to do

- 2 great papers to read
- 4 old tools to learn
- 4 really bad things
- 3 great books to read
- 7 reasons why software is difficult now
- 10 reasons why software was easier back in the day
- 1 fun programming exercise
- 8 great machines from the past

… and …

. . . more

- 3 performance improvements
- 5+ YouTube videos to watch
- 6 things not to do
- 5 sins
- 4 languages to learn
- 4 great forgotten ideas
- 6 areas to research
- 2 dangers
- 4 ideas that are obvious now but strange at first
- 2 fantastic programs to try

2 great papers to read

Cybersquare

A Plea for Lean Software

Niklaus Wirth
ETH Zürich

[illegible]

Software's girth has surpassed its functionality, largely because hardware advances make this possible. The way to streamline software lies in disciplined methodologies and a return to the essentials.

The 1980 ACM Turing Award Lecture

Delivered at ACM '80, Nashville, Tennessee, October 27, 1980

C.A.R. Hoare

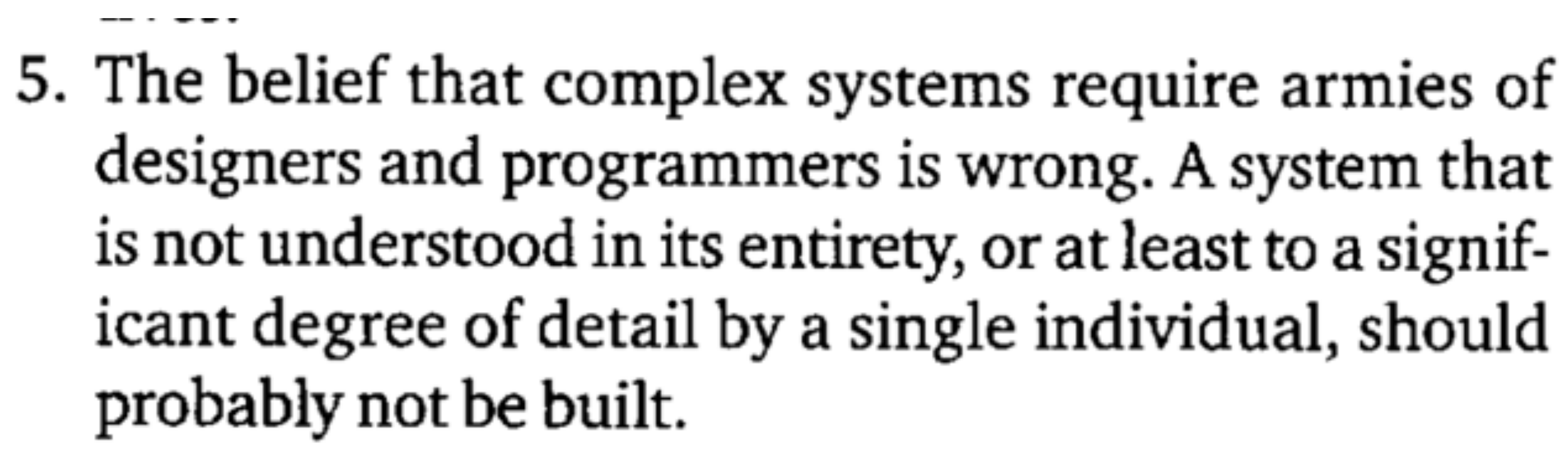
[illegible]

The Emperor's Old Clothes

Charles Antony Richard Hoare
Oxford University, England

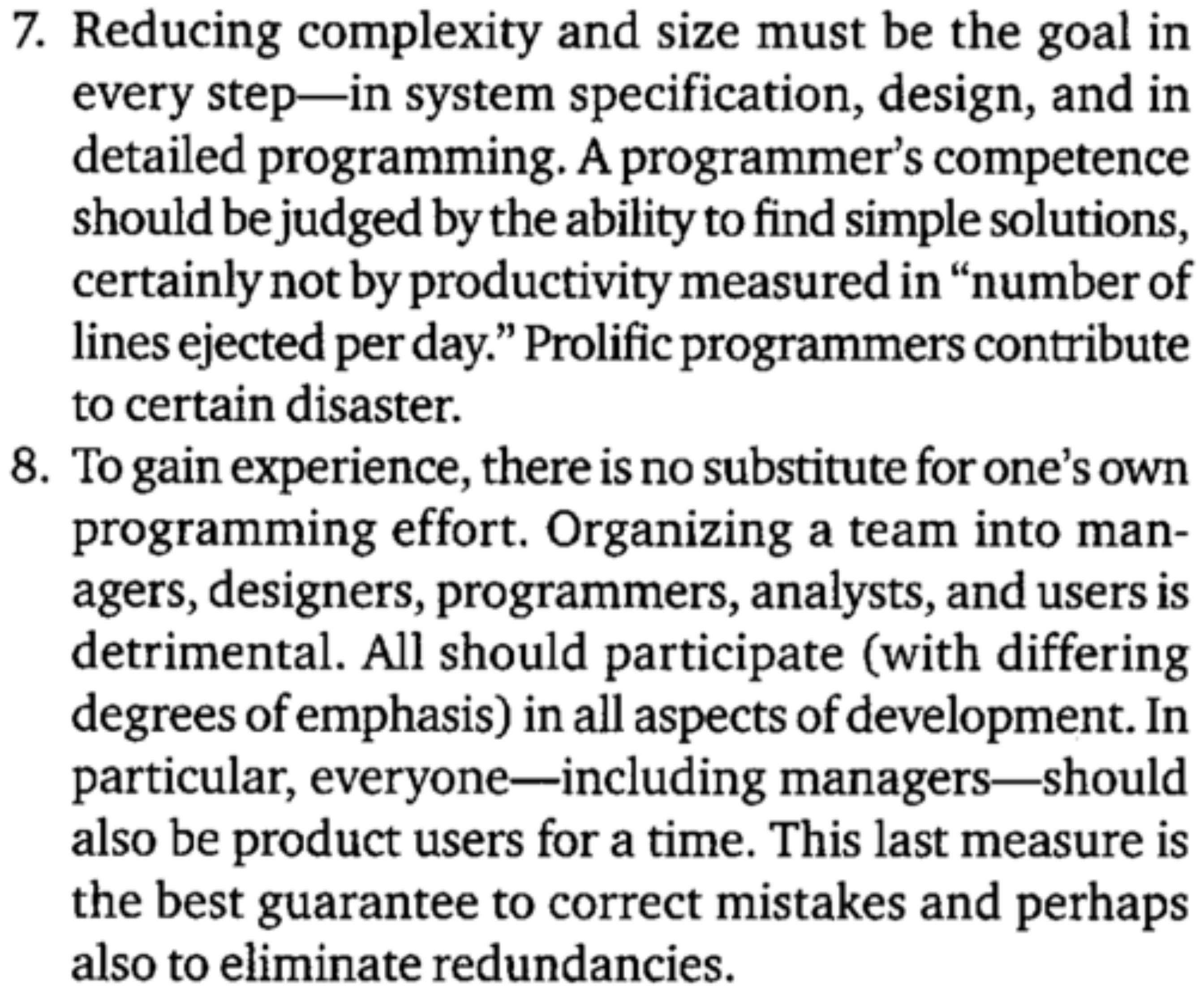
[illegible]

- A Plea for Lean Software - Niklaus Wirth
- The Emperor's old clothes - ACM Turing award lecture - Tony Hoare

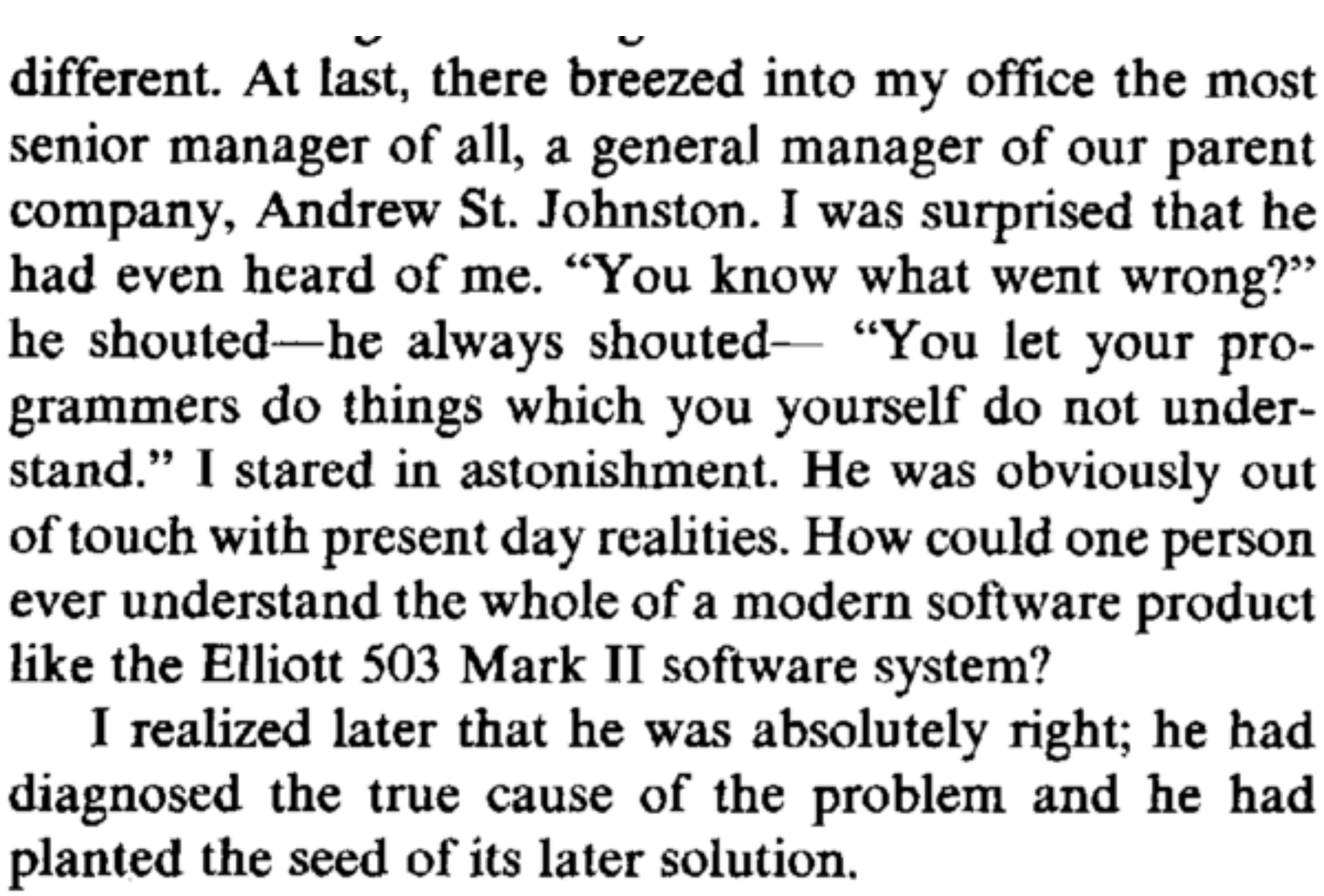


5. The belief that complex systems require armies of designers and programmers is wrong. A system that is not understood in its entirety, or at least to a significant degree of detail by a single individual, should probably not be built.

Wirth

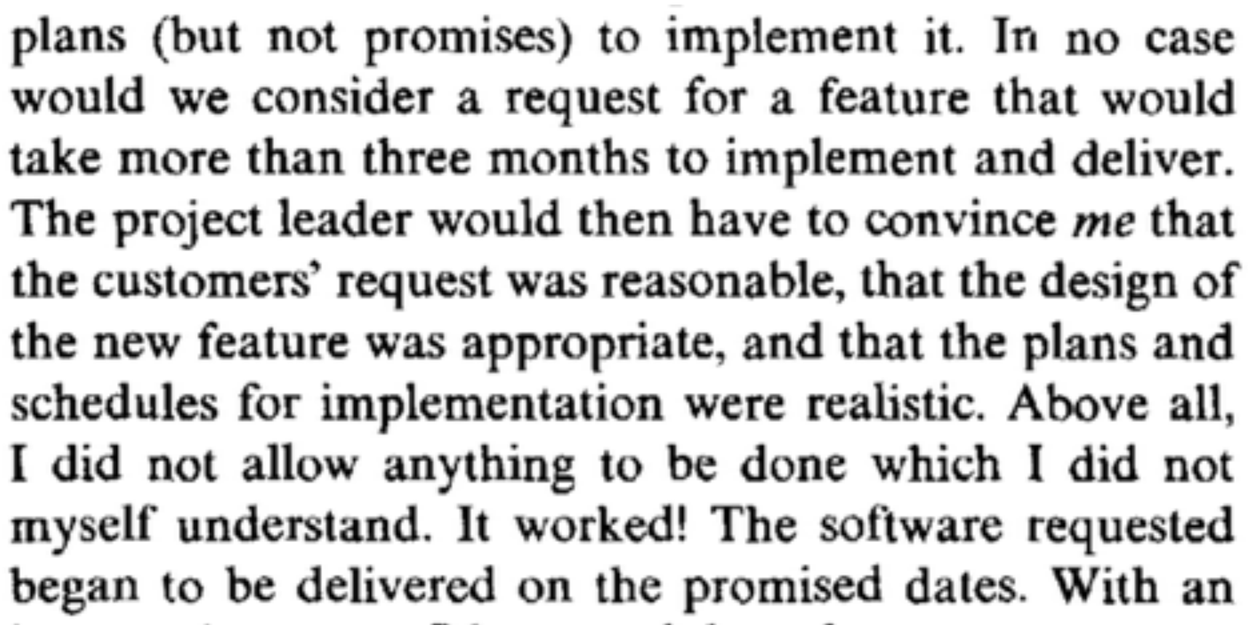


7. Reducing complexity and size must be the goal in every step—in system specification, design, and in detailed programming. A programmer's competence should be judged by the ability to find simple solutions, certainly not by productivity measured in "number of lines ejected per day." Prolific programmers contribute to certain disaster.
8. To gain experience, there is no substitute for one's own programming effort. Organizing a team into managers, designers, programmers, analysts, and users is detrimental. All should participate (with differing degrees of emphasis) in all aspects of development. In particular, everyone—including managers—should also be product users for a time. This last measure is the best guarantee to correct mistakes and perhaps also to eliminate redundancies.


different. At last, there breezed into my office the most senior manager of all, a general manager of our parent company, Andrew St. Johnston. I was surprised that he had even heard of me. "You know what went wrong?" he shouted—he always shouted— "You let your programmers do things which you yourself do not understand." I stared in astonishment. He was obviously out of touch with present day realities. How could one person ever understand the whole of a modern software product like the Elliott 503 Mark II software system?

I realized later that he was absolutely right; he had diagnosed the true cause of the problem and he had planted the seed of its later solution.

Hoare


plans (but not promises) to implement it. In no case would we consider a request for a feature that would take more than three months to implement and deliver. The project leader would then have to convince *me* that the customers' request was reasonable, that the design of the new feature was appropriate, and that the plans and schedules for implementation were realistic. Above all, I did not allow anything to be done which I did not myself understand. It worked! The software requested began to be delivered on the promised dates. With an

4 old tools to learn

- emacs (vi)
- bash
- make
- shell

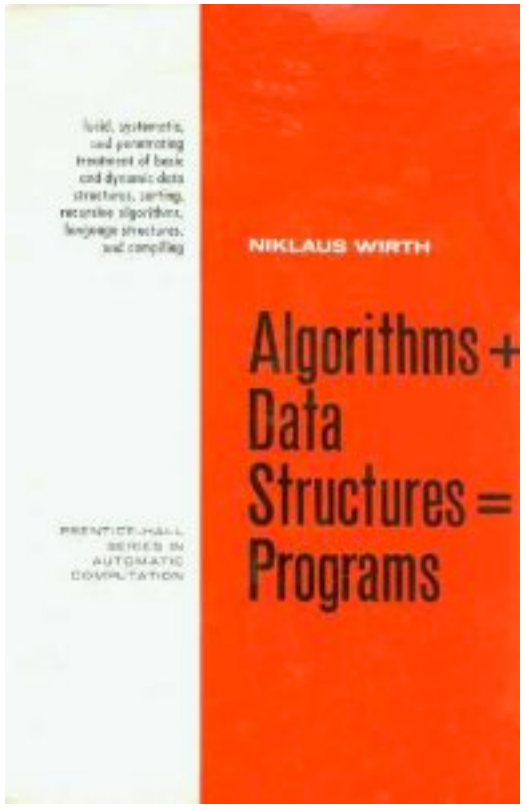
4 really bad things

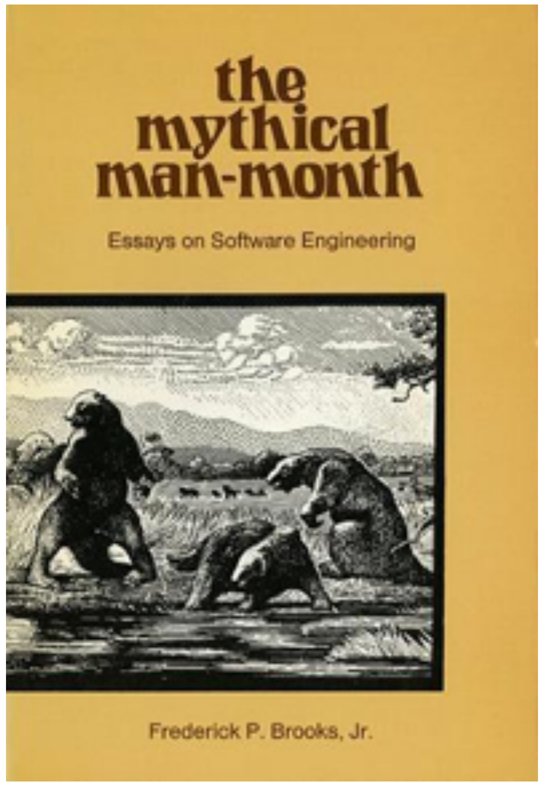
- Lack of Privacy
- Attempts to manipulate us through social media
- Vendor Lock in
- Terms and Conditions

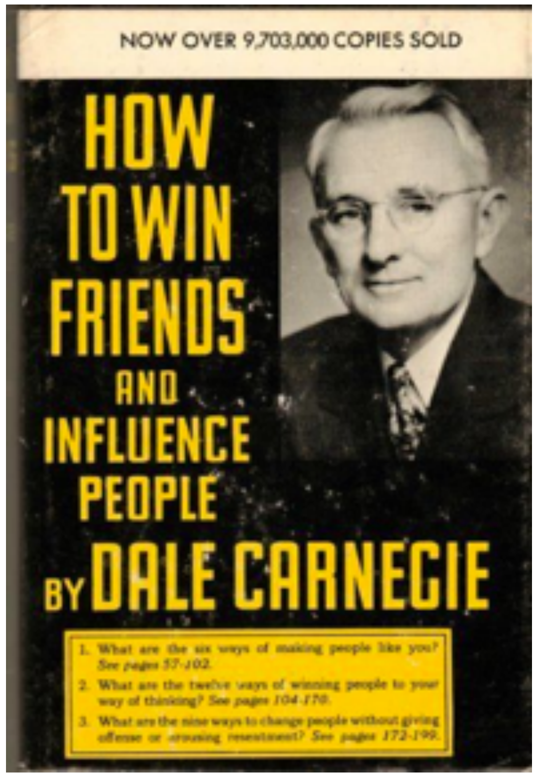
Show of hands

- I’ve read all the terms and conditions and understood them
- I’ve read the terms and conditions and didn’t understand them
- I just clicked on accept

3 great books to read


lucid, systematic, and penetrating treatment of basic and dynamic data structures, sorting, recursive algorithms, language structures, and compiling
NIKLAUS WIRTH
Algorithms + Data Structures = Programs
PRENTICE-HALL SERIES IN AUTOMATIC COMPUTATION




NOW OVER 9,703,000 COPIES SOLD
HOW TO WIN FRIENDS AND INFLUENCE PEOPLE
BY DALE CARNEGIE
1. What are the six ways of making people like you? See pages 57-102.
2. What are the twelve ways of winning people to your way of thinking? See pages 104-170.
3. What are the nine ways to change people without giving offense or arousing resentment? See pages 172-190.

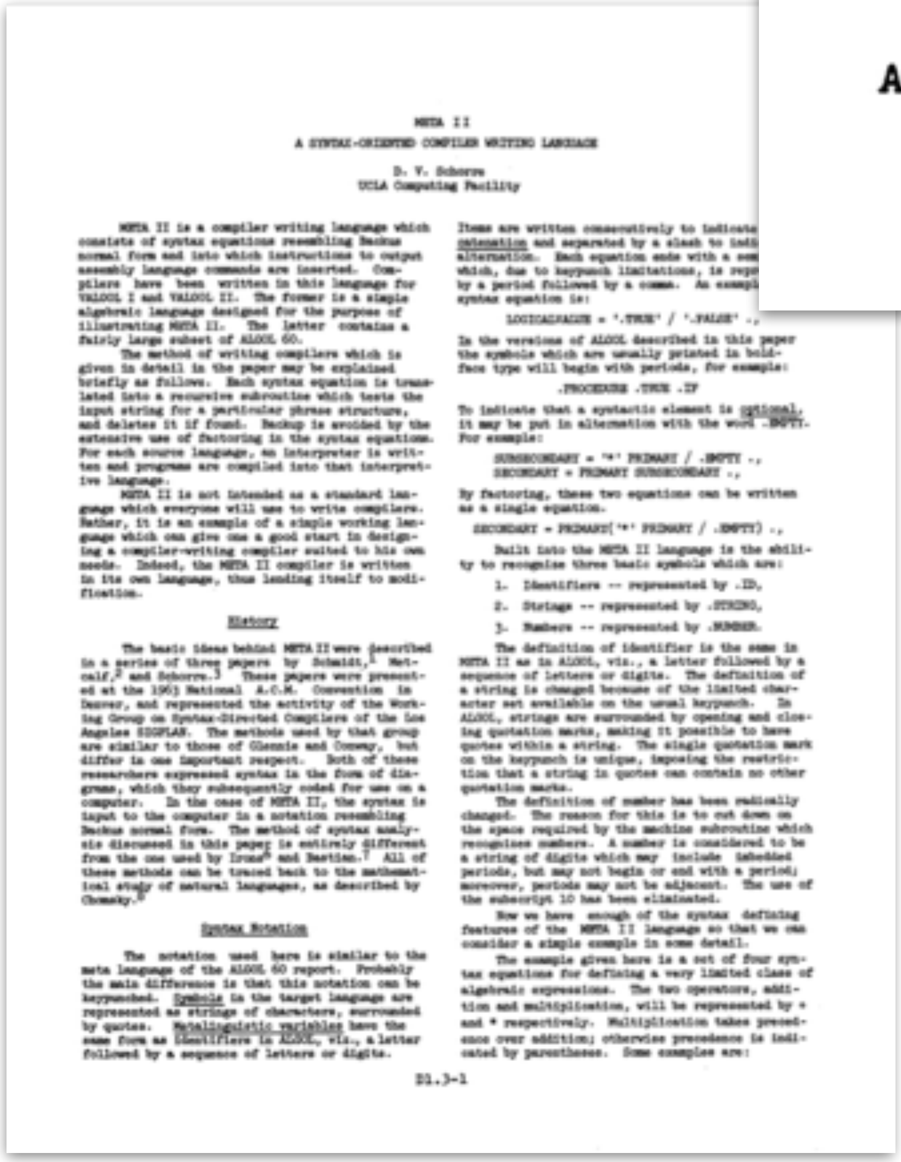
7 reasons why software is difficult now

- Fast machines
- Huge memory
- Hundreds of PLs
- Distributed
- Huge programs
- No specifications
- Reuse

10 reasons why software was easier back in the day

- Small machines
- Small memory
- Few languages
- Not distributed
- No reuse of code
- No Xcode etc
- No GIT.
- Complete control
- Did not communicate
- Understandable in it’s entirety

1 fun programming exercise



META II

A SYNTAX-ORIENTED COMPILER WRITING LANGUAGE

D. V. Schorre
UCLA Computing Facility

META II is a compiler writing language which consists of syntax equations resembling Backus normal form and into which instructions to output assembly language commands are inserted. Compilers have been written in this language for VALGOL I and VALGOL II. The former is a simple algebraic language designed for the purpose of illustrating META II. The latter contains a fairly large subset of ALGOL 60.

The method of writing compilers which is given in detail in the paper may be explained briefly as follows. Each syntax equation is translated into a recursive subroutine which tests the input string for a particular phrase structure, and deletes it if found. Backup is avoided by the extensive use of factoring in the syntax equations. For each source language, an interpreter is written and programs are compiled into that interpretive language.

META II is not intended as a standard language which everyone will use to write compilers. Rather, it is an example of a simple working language which can give one a good start in designing a compiler-writing compiler suited to his own needs. Indeed, the META II compiler is written in its own language, thus lending itself to modification.

<u>History</u>

The basic ideas behind META II were described in a series of three papers by Schmidt,[1] Metcalf,[2] and Schorre.[3] These papers were presented at the 1963 National A.C.M. Convention in Denver, and represented the activity of the Working Group on Syntax-Directed Compilers of the Los Angeles SIGPLAN. The methods used by that group are similar to those of Glennie and Conway, but differ in one important respect. Both of these researchers expressed syntax in the form of diagrams, which they subsequently coded for use on a computer. In the case of META II, the syntax is input to the computer in a notation resembling Backus normal form. The method of syntax analysis discussed in this paper is entirely different from the one used by Irons[6] and Bastian.[7] All of these methods can be traced back to the mathematical study of natural languages, as described by Chomsky.[8]

<u>Syntax Notation</u>

The notation used here is similar to the meta language of the ALGOL 60 report. Probably the main difference is that this notation can be keypunched. <u>Symbols</u> in the target language are represented as strings of characters, surrounded by quotes. <u>Metalinguistic variables</u> have the same form as <u>identifiers in ALGOL</u>, viz., a letter followed by a sequence of letters or digits.

Items are written consecutively to indicate <u>catenation</u> and separated by a slash to indi alternation. Each equation ends with a sem which, due to keypunch limitations, is repr by a period followed by a comma. An exampl syntax equation is:

```
LOGICALVALUE = '.TRUE' / '.FALSE' .,
```

In the versions of ALGOL described in this paper the symbols which are usually printed in boldface type will begin with periods, for example:

```
.PROCEDURE .TRUE .IF
```

To indicate that a syntactic element is <u>optional</u>, it may be put in alternation with the word .EMPTY. For example:

```
SUBSECONDARY = '*' PRIMARY / .EMPTY .,
SECONDARY = PRIMARY SUBSECONDARY .,
```

By factoring, these two equations can be written as a single equation.

```
SECONDARY = PRIMARY('*' PRIMARY / .EMPTY) .,
```

Built into the META II language is the ability to recognize three basic symbols which are:

1. Identifiers -- represented by .ID,
2. Strings -- represented by .STRING,
3. Numbers -- represented by .NUMBER.

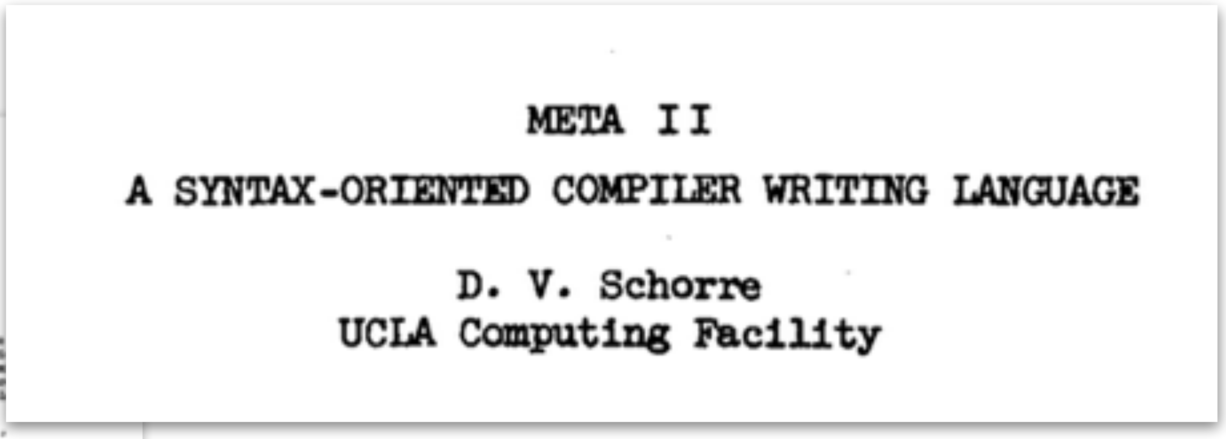
The definition of identifier is the same in META II as in ALGOL, viz., a letter followed by a sequence of letters or digits. The definition of a string is changed because of the limited character set available on the usual keypunch. In ALGOL, strings are surrounded by opening and closing quotation marks, making it possible to have quotes within a string. The single quotation mark on the keypunch is unique, imposing the restriction that a string in quotes can contain no other quotation marks.

The definition of number has been radically changed. The reason for this is to cut down on the space required by the machine subroutine which recognizes numbers. A number is considered to be a string of digits which may include imbedded periods, but may not begin or end with a period; moreover, periods may not be adjacent. The use of the subscript 10 has been eliminated.

Now we have enough of the syntax defining features of the META II language so that we can consider a simple example in some detail.

The example given here is a set of four syntax equations for defining a very limited class of algebraic expressions. The two operators, addition and multiplication, will be represented by + and * respectively. Multiplication takes precedence over addition; otherwise precedence is indicated by parentheses. Some examples are:

D1.3-1



META II

A SYNTAX-ORIENTED COMPILER WRITING LANGUAGE

D. V. Schorre
UCLA Computing Facility

Serious fun - might cause your brain to melt

8 great machines from the past

- Baby SSEM
- PDP11
- Vax 11/750
- Cray 1
- IBM PC
- Raspberry PI
- iPhone/iPad
- Nvidia Tesla P100

3 performance improvements

- Better algorithms (x 6) (Interpreter -> Compiler)
- Better Programming language (x50) (Prolog -> C)
- Better Hardware (x1000 per 10 years)

5+ YouTube videos to watch

- The computer revolution has not happened yet
  Alan Kay

- Computers for Cynics
  Ted Nelson

- Free is a lie (Aaron Balkan)

- How a handful of tech companies control billions of minds every day Tristan Harris (TED-Talk)

- Matt Might - Winning the War on Error: Solving Halting Problem, Curing Cancer - Code Mesh 2017

6 things not to do

- Backdoors
- Violate privacy
- Put microphones in everybody's houses
- Hijack our attention system
- Hijack our social systems
- Sell crap that we don’t want or need

5 sins

- Crap documentation
- Crap website
- Crap dependencies
- Crap build instructions
- Group think

4 languages to learn

- C
- Prolog
- Erlang
- Javascript

4 great forgotten ideas

- Linda Tuple Spaces - David Gelernter and Nicholas Carriero.
- Flow based programming - John Paul Morrison.
- **Xanadu - Ted Nelson**
- Unix pipes

Pipes

- **The output of my program should be the input to your program**
- A | B | C
- Text-flows across the boundary
- **Killed by GUIs** and Apps

Apps

- Pads - Tablets - Phones
- Human can interact with Apps
- Apps can’t interact with each other
- You are locked inside your Apps. They all do different things with a varying degree of success.

6 areas to research

- Robotics
- AI
- Progammer productivity
- Energy efficiency
- Precision Medicin
- Security

2 dangers

- Group think
- Bubble think

4 ideas that are obvious now but strange at first

- Indentation
- Versioning
- Hypertext across machine boundaries
- Pipes

2 fantastic programs to try

- TiddlyWiki (more later)
- SonicPI

Part 3
Important non computer science things

learn to write

- A program with excellent documentation is not going to go anywhere
- …

3 rules at work

- If you get a bad boss move immediately
  do not try to change your boss

- The relationship comes first (Jane Walerud)

- Engage with management
  just because they do not understand what you are saying is no reason not to talk to them - and whose fault is it anyway (that they don’t understand you)

7 distractions

- Open plan offices
- The latest stuff
- Twitter/Facebook (social media)
- Notifications (turn ‘em off)
- Links (don’t click on them)
- Ban Scrum etc.
- We can only do one thing at a time
  Our brains are terribly bad at context switching

6 ways to get your boss to <program in Erlang>

- Do things that gain trust
- Tell success stories
- Reduce fear of failure
- Introduce on a small scale - for a part of the problem
- Network with Erlang folks
- Make a prototype at home

1 thing to look for when applying for a new job

- Look at their balance sheet
  a company with a positive cash flow and increasing profits is good to work for - a company that makes a loss is not good to work for

3 general laws

- Software complexity grows with time (because we build on old stuff)
- Bad code crowds out good (Gresham’s law)
  bad money drives out good (clipping)
- Bad code contaminates good code

Laws of Physics and maths

3 laws of physics

- A computation can only take place when the data and the program are at the same point in space time => get all the data + program to the same place (can be client OR server or someplace in-between) (problem - easy to move data - difficult to move programs) **This is why PHP is good :-)**

- Causality - Effect follows cause. We don't know how stuff **is** we know how it **was** (the last time it told us)

- 2'nd law thermo dynamics - Entropy (disorder) always increases

Entropy

- Early Unix (1970) had a very small disk so programs that were not used were thrown away (decreases entropy - natural selection)
- Git keeps all old versions (increases entropy - cancer)
- https://en.wikipedia.org/wiki/Unix_philosophy

It's all about Trust and Responsibility

Trust is transitive

- I trust the SW written by Robert
- Robert wrote X
- => I trust X

Can I trust X?

- I need a program to do X
- I find X in github
- I do not know who wrote X
- Can I trust X?

Responsibility

- I reuse X in program P
- I ship program P to custom A
- A reports an error in P
- I am responsable
- => I must trust that P is correct

User’s Problems

6 common problems

- Does not know how to delete files - when the system runs out of space they buy a new computer
- No idea of what MBytes, Mbits, Bbits/sec quad cores etc means
- If the app doesn’t work immediately gives up
- Does not Google for fixes - or does and does not understand the answers
- Does not want to try the latest things
- Uses a method that works (not the best) - ie to copy
  a file open word - read the file in then writes it out with a new name

5 more Problems

- The UI changes
- Passwords
- Stuff doesn’t work
- Terms & Conditions
- … non reproducible errors

Helping your non-technical neighbour

- Tell them “it’s not your fault”
- Tell them “it’s crap software”
- Tell them “I don’t understand this crap either”
- Tell them “computers can’t do everything”

Part 4
Important
half forgotten BIG
ideas

Things can be small

- Forth OS 24 KB
- Forth compiler 12KB
- IBM PC DOS < 640KB
- USCD Pascal
- Turbo Pascal
- Turbo C

The old truths

- Keep it simple
- Make it small
- Make it correct
- Fight complexity

Learning

- Kids can learn computing
- OAPs can learn computing
- Everybody can learn computing
  It was easy to learn BASIC back in the 80’s so why is it more difficult now?

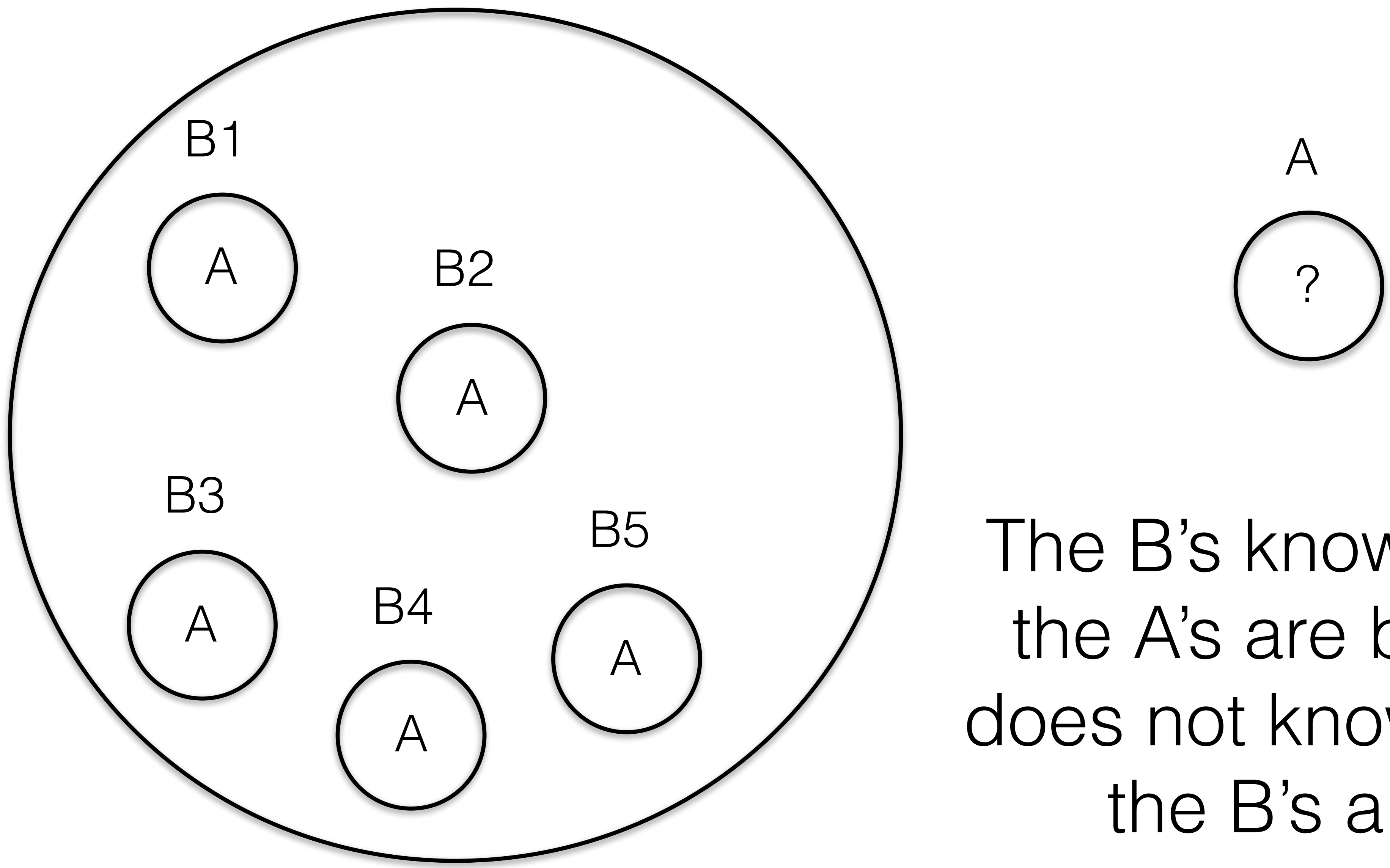
Web is broken

- It’s not symmetric
  Users read data but write very little
- Can every page be changed?
- Can I make new data by combining fragments from other data in a flexible manner? - no
- The Web is dominated by a small number of companies (Amazon, Apple, Goole, Facebook) using huge data centers, it should be controlled from the edge network.
- The original vision was a Web controlled by “citizen programmers” (Google Ted Nelson talks)

HTML and HTTP have several problems

- Non symmetric
- Easy to read/difficult to write
- Pages get lost (disappear)
- Links are wrong (404-problem)
- Re-use, attribution, IP rights, payments is a mess
- Controlled by a very small number of companies

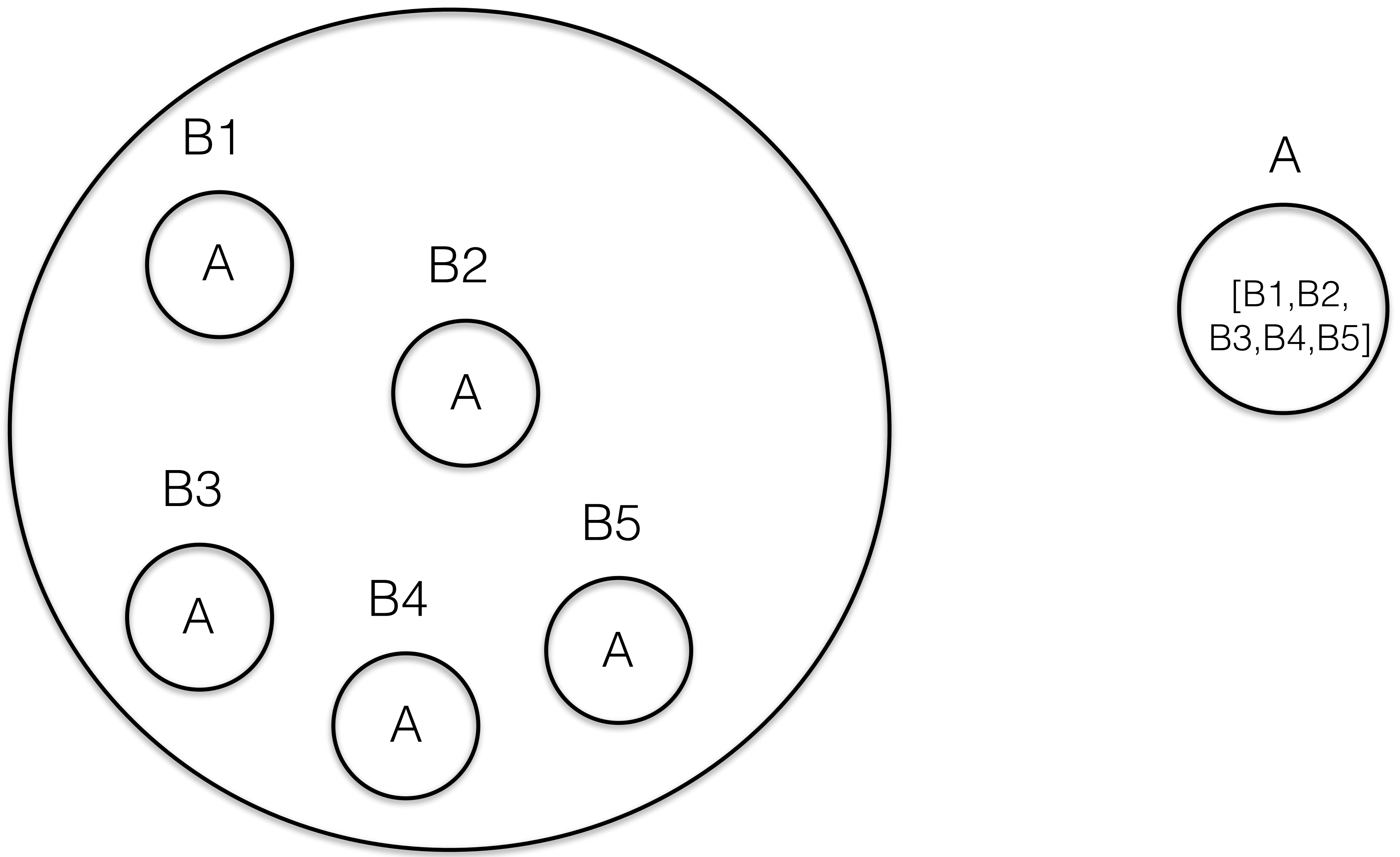
Lack of symmetry



The B's know who the A's are but A does not know who the B's are

<a href="A">link</a>

Recovering symmetry



<a href="A">link</a>

Wiki

- Links cannot get lost
- Much better integration - entities are tightly intertwined (less entropy)
- All in one place

Xanadu

- Like the web but better
- No broken links
- No difference between readers and writers
- Never loose any data
- All copyright and attribution correct
- Complete knowledge of parents and children

Problems

- 404 - Not found
- A might move to a new server
- Server where A is might be down
- A cannot be renamed

Part 5
What we can do

- Unbreak the web
  Make it read/write symmetric

- Bring computation to the edge network

- Ensure that all personal data is owned by the individual and not by large corporations

- Make computing easy again

- Build Apps so they can communicate with each other

A program that is not secure
and cannot be remotely
controlled should not be written

We’ve given
millions of people
supercomputers -
so let them use them
and …

It’s your turn next