Good ideas that we forgot

Joe Armstrong

My goals

- To remind you of the important things worth knowing
- Identify the stuff worth learning
- Identify some problems worth solving

[note: this is a very biased view]

Joe Armstrong & Alan Kay - Joe Armstrong interviews Alan Kay

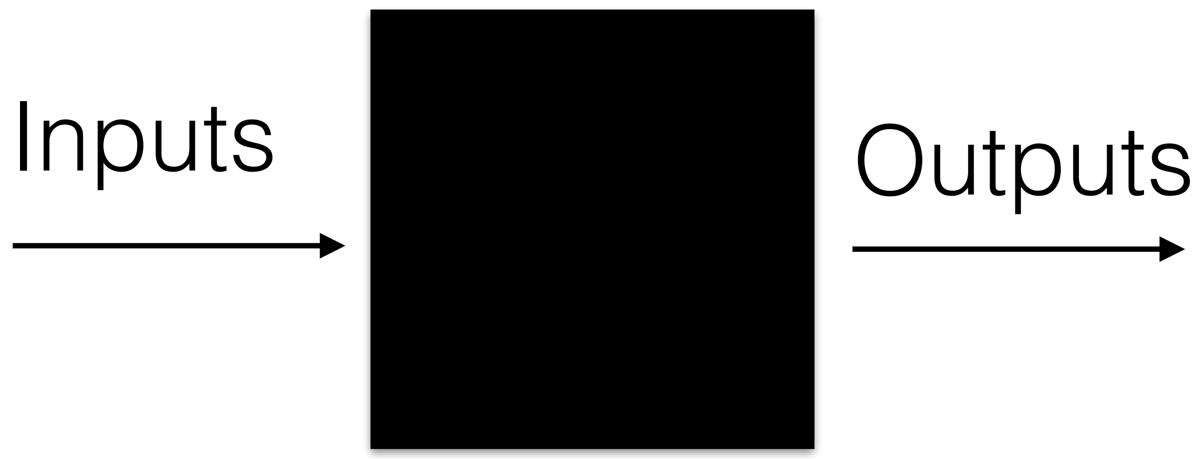
26,882 views

What ideas has we forgotten?

Computer science 101

#1 - Observational Equivalence

- Two systems are equivalent if they cannot be distinguished on the basis of their observable inputs and outputs.

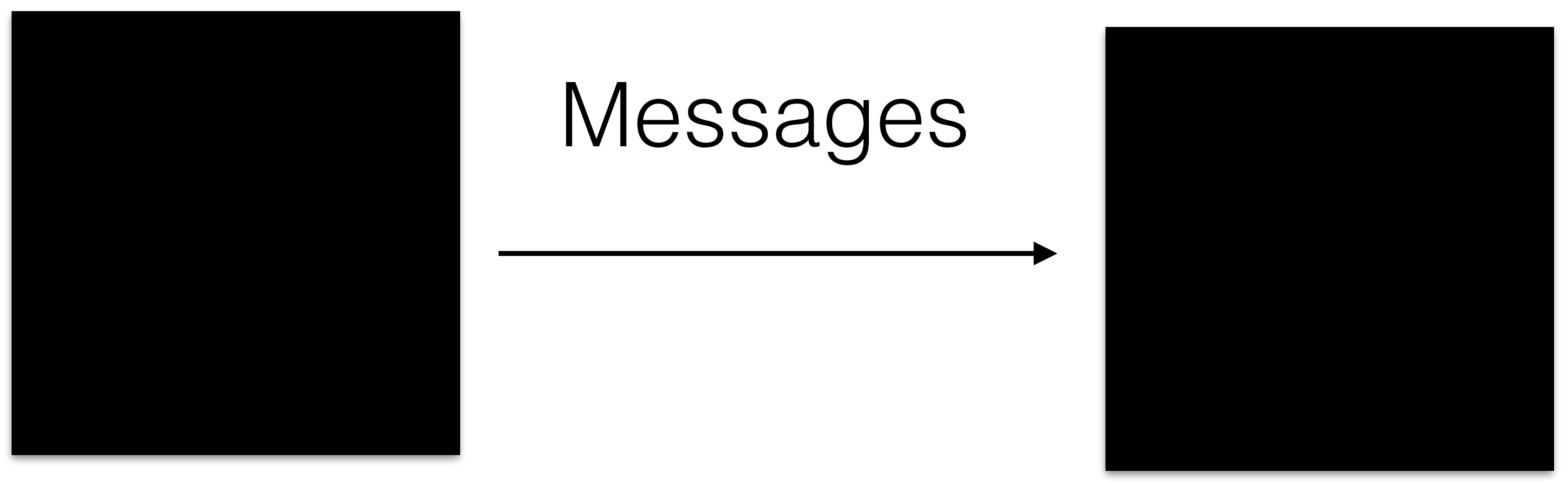


Need Several languages:

- Describe the Inputs and outputs
- Describe Computations
- Describe Connections
- Describe sequences of events

#2 - Isolation

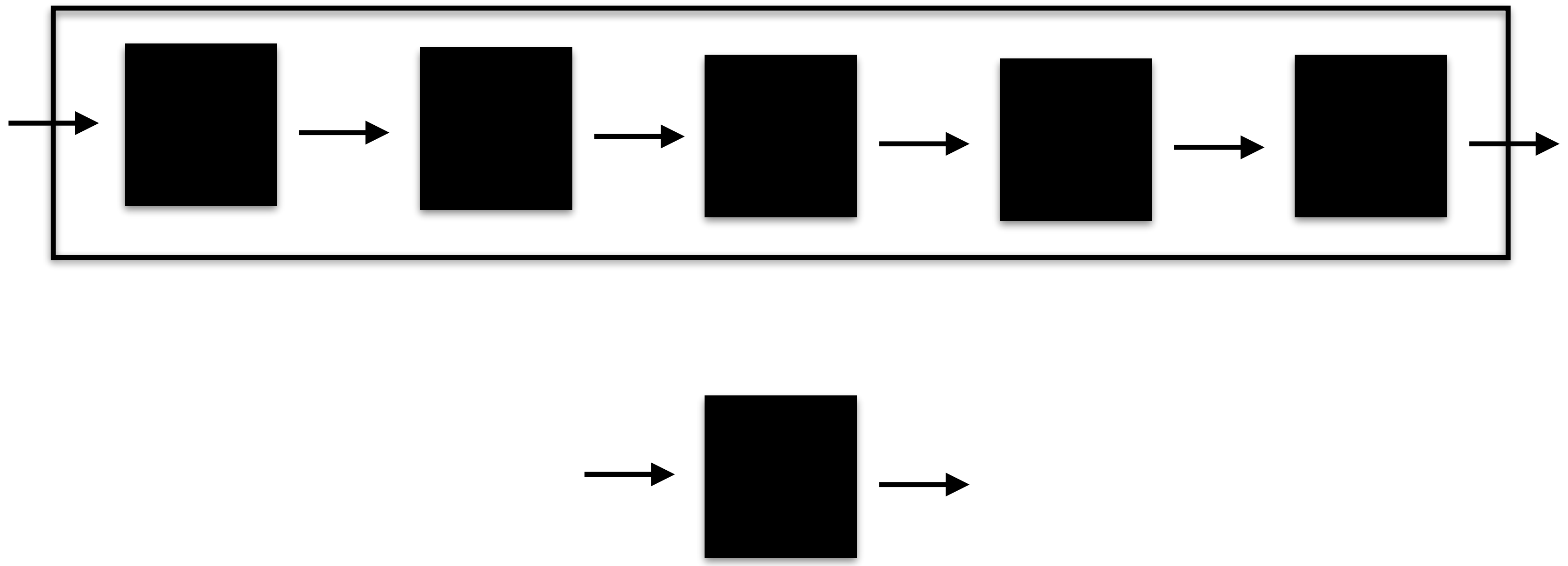
- Two systems are isolated if what happens in one system cannot influence what happens in the other system.



- Messages should never crash the system
- Messaging is inevitable
- The sender never knows if the message is received

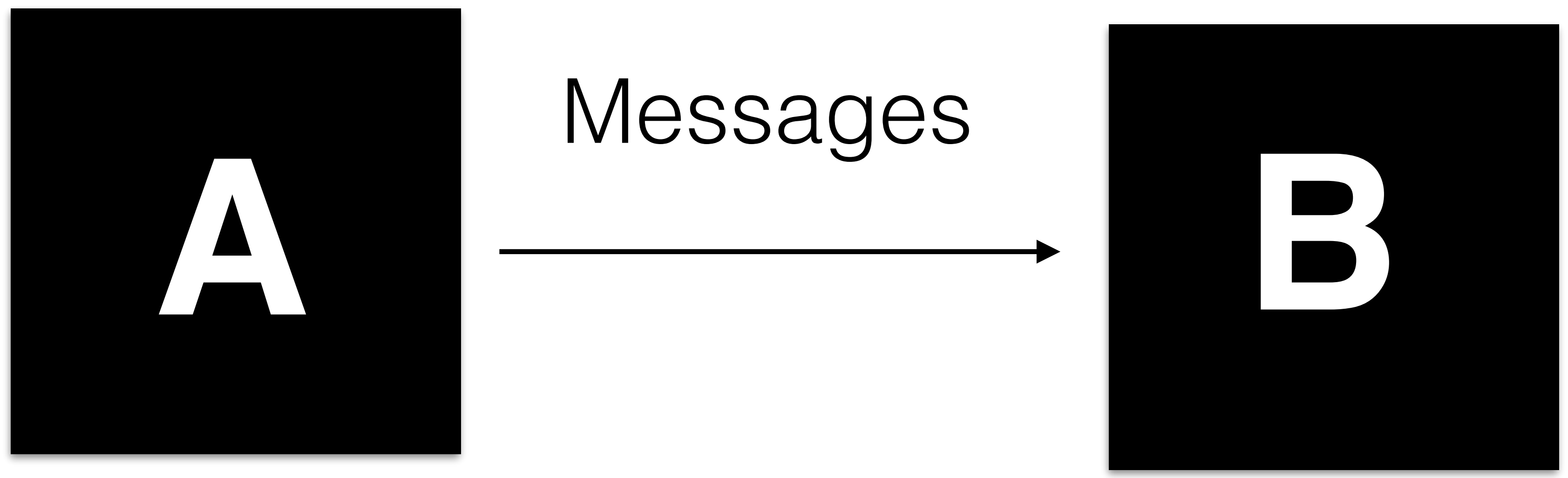
#3 - Composition

- Things are composable if they can be combined in such a way that the combination behaves in a similar manner to the individual parts.



#4 - Causality

- Effect Follows Cause



- B does not know how A **IS** only how it **WAS**
- A does not know if B received the last message it sent

#5 - Physics

- For a computation to take place all the data and the program must be at the same place in space time

So you can move the data or the program or both.

Tip: get all the data you need and the program to one place before doing a computation

Violating any one of these principles will lead to brittle software that may appear to work but will one day fail in ways that are difficult to understand

Part 2
Things to learn which you might have forgotten or not known about

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 …

- ~~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~~

80

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

[illegible]

C.A.R. Hoare

The Emperor's Old Clothes

Charles Antony Richard Hoare
Oxford University, England

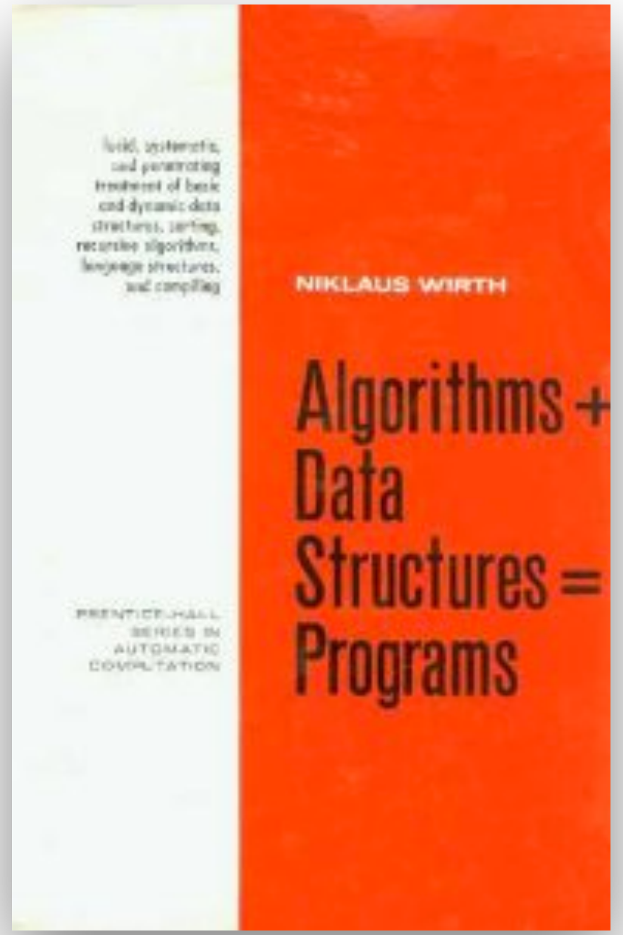
[illegible]

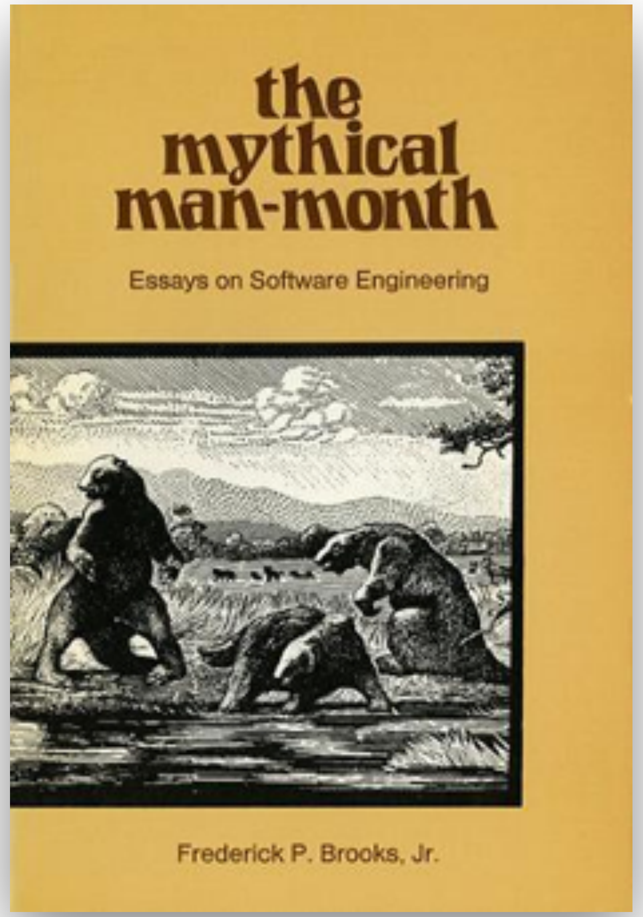
- A Plea for Lean Software - Niklaus Wirth
- The Emperor's old clothes - ACM Turing award lecture - Tony Hoare

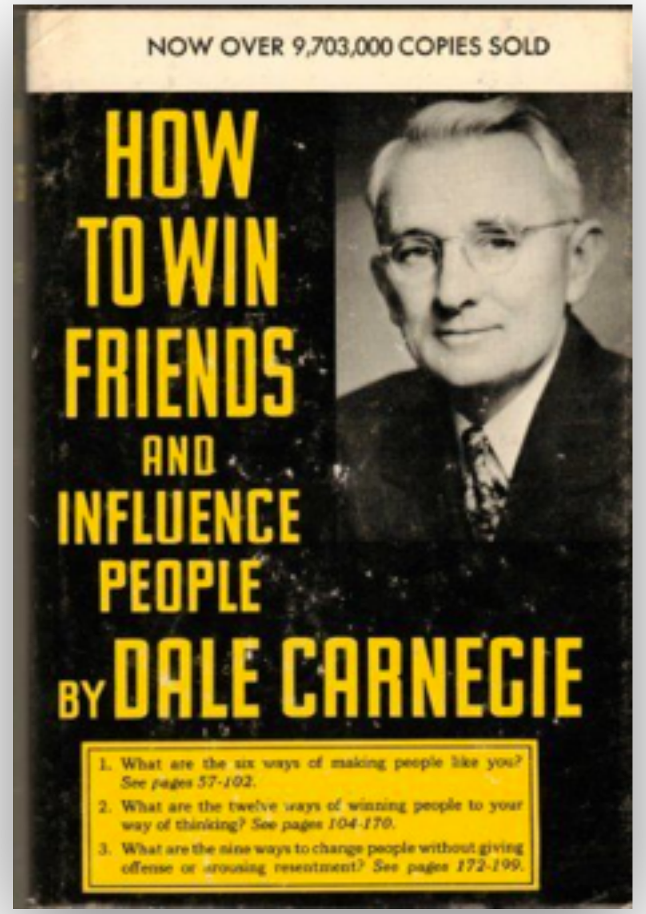
4 old tools to learn

- emacs (vi)
- bash
- make
- shell

3 great books to read

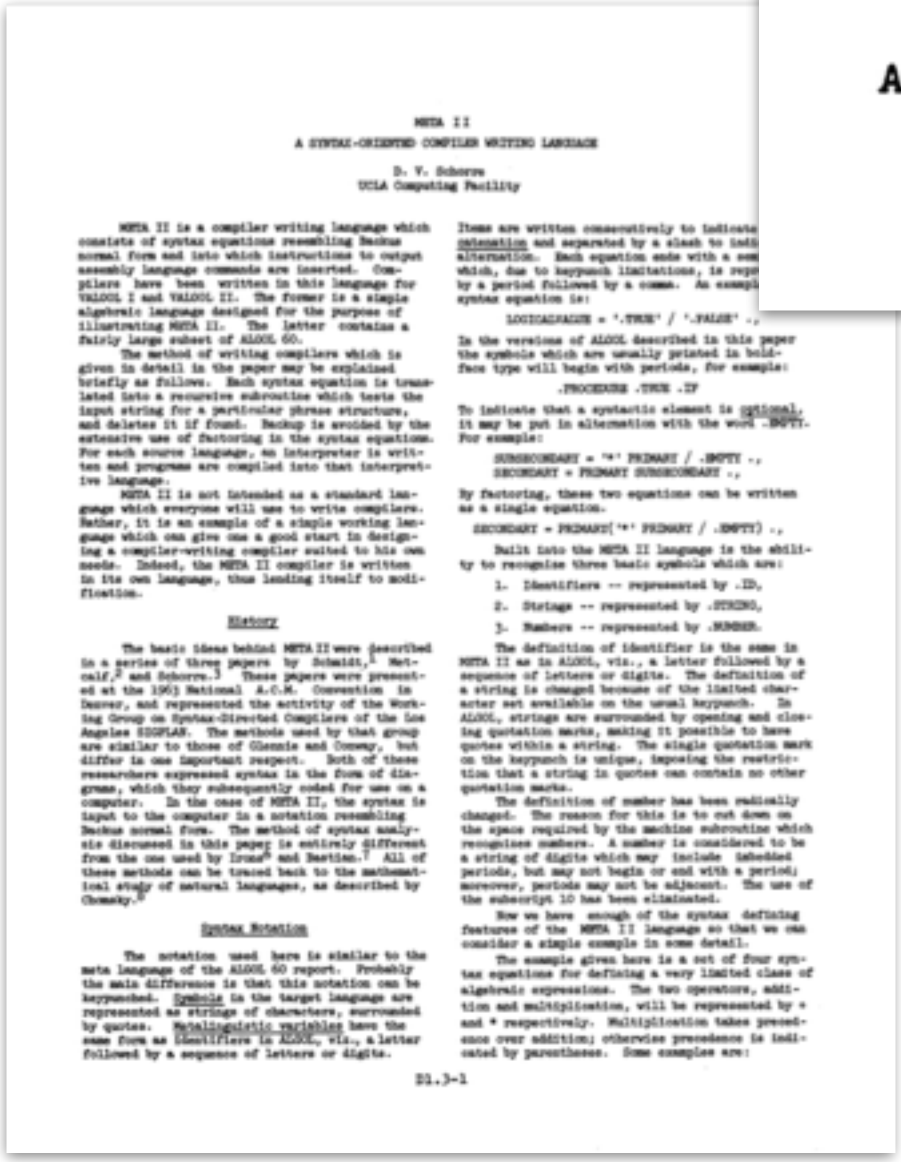

lucid, systematic, and penetrating treatment of basic and dynamic data structures, sorting, recursive algorithms, language structures, and compiling
NIKLAUS WIRTH
Algorithms + Data Structures = Programs
PRENTICE-HALL SERIES IN AUTOMATIC COMPUTATION




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 172-199.*

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.

History

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]

Syntax Notation

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. Symbols in the target language are represented as strings of characters, surrounded by quotes. Metalinguistic variables have the same form as identifiers in ALGOL, viz., a letter followed by a sequence of letters or digits.

Items are written consecutively to indicate
catenation 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 optional, 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.

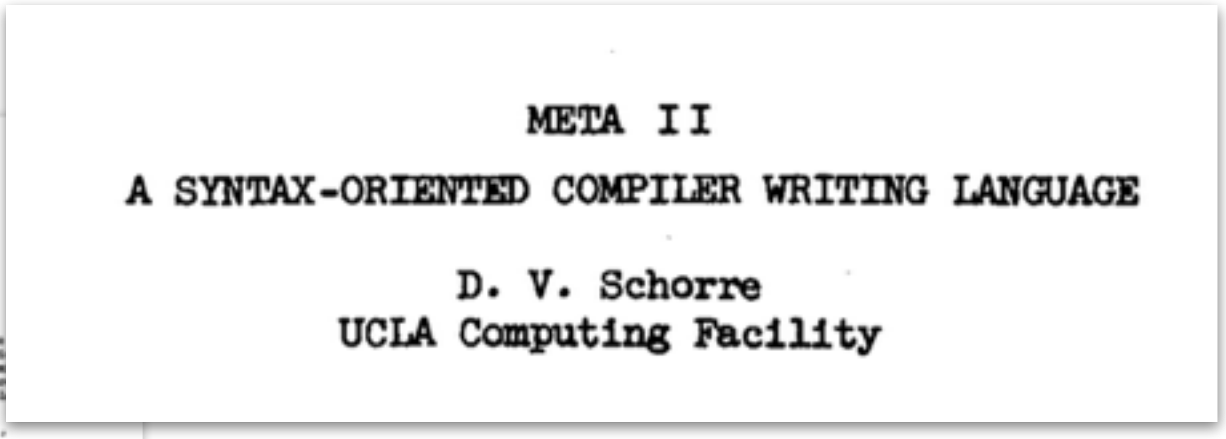
The definition of identifier is the same in META II as in ALGOL, viz., a letter followed by a sequence of letters or digits. The definition of a string is changed because of the limited character set available on the usual keypunch. In ALGOL, strings are surrounded by opening and closing quotation marks, 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

YouTube videos to watch

- The computer revolution has not happened yet
  Alan Kay

- Computers for Cynics
  Ted Nelson

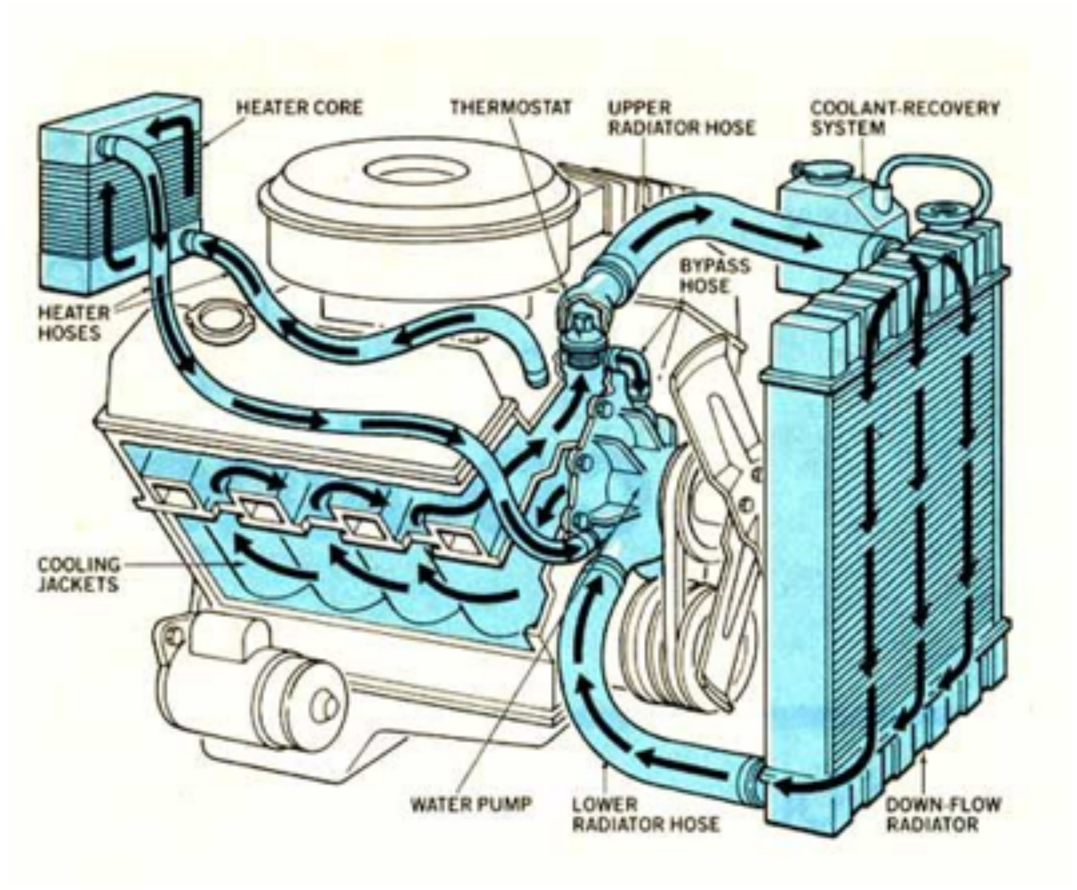
Part 4

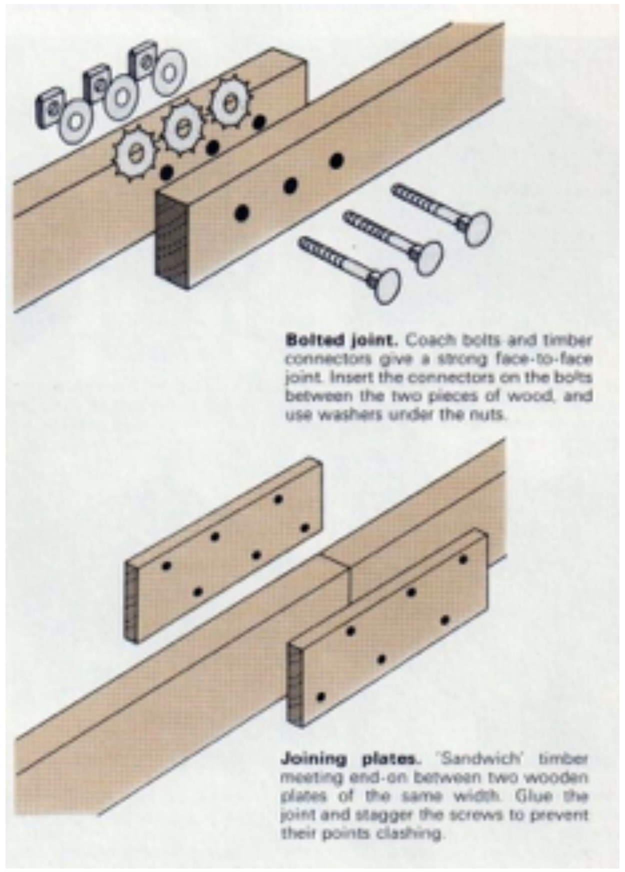
Four great forgotten ideas

Flow Based Programming

Flow Based Programming

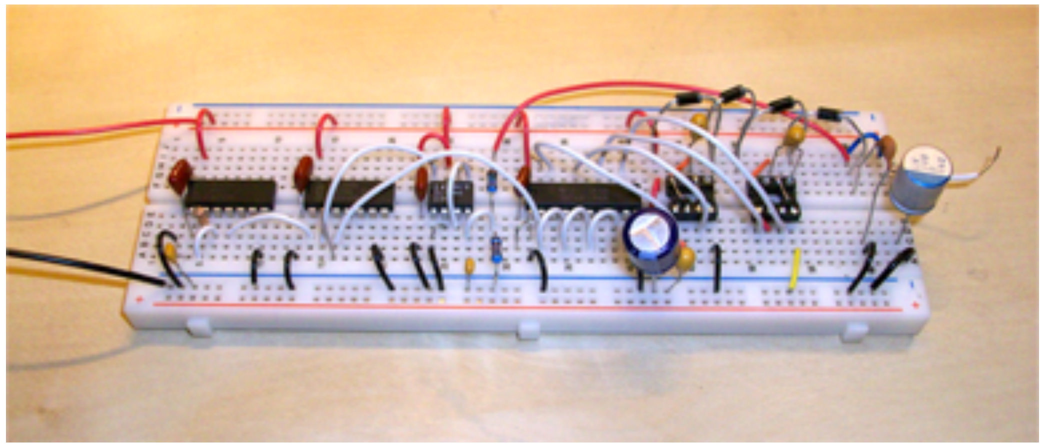
- Invented by John Paul Morrison in the early 1970's
- Programming by "placing objects next to each other"
- Binary distribution of components
- Account for all packets

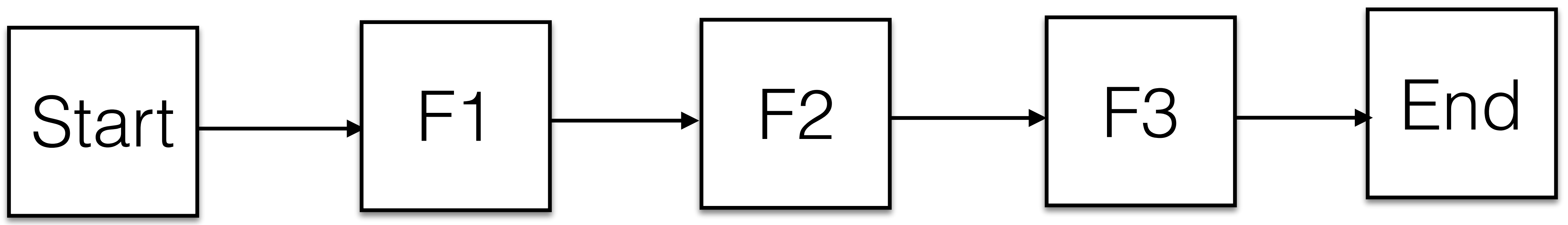


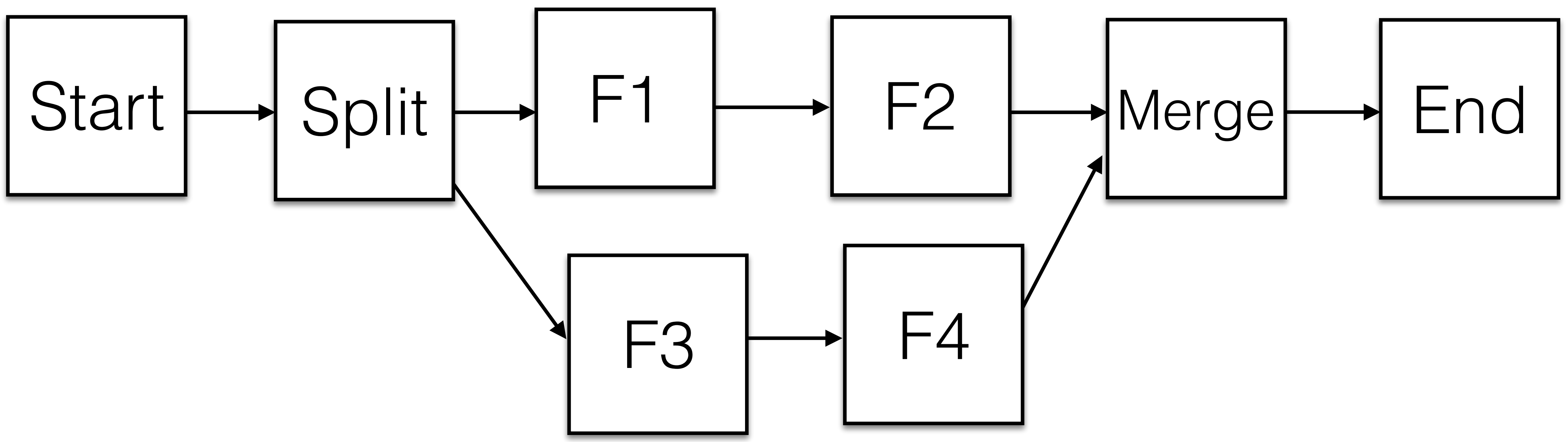


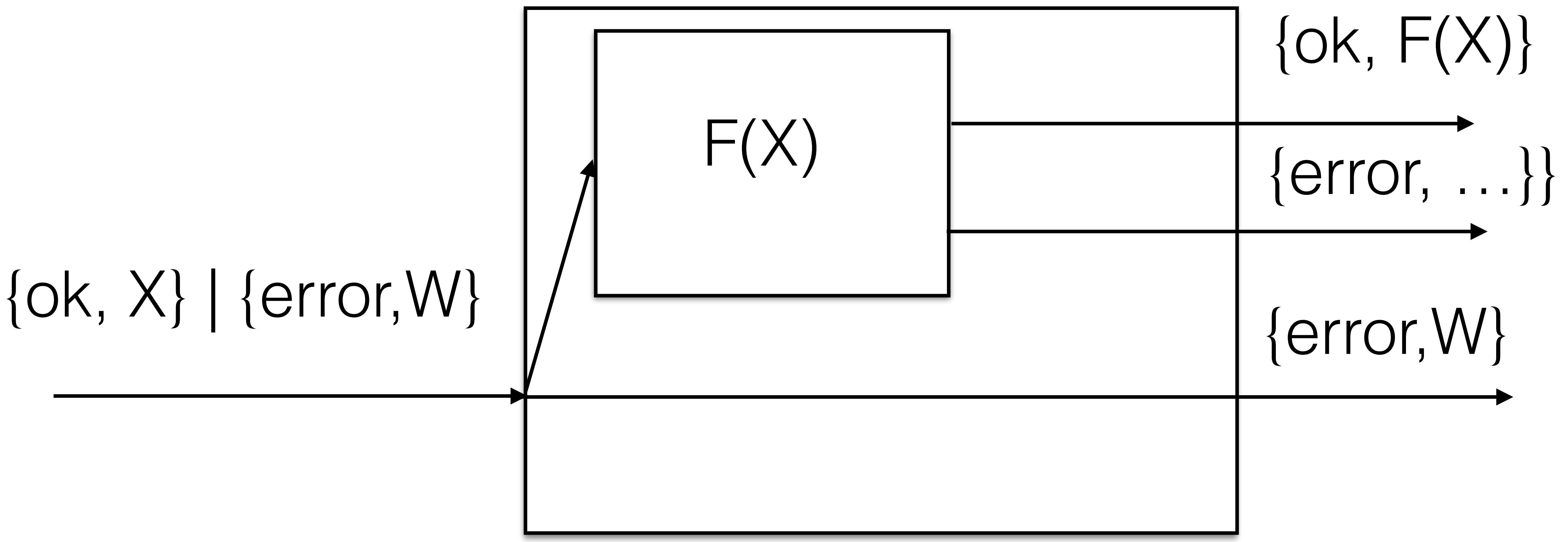
Bolted joint. Coach bolts and timber connectors give a strong face-to-face joint. Insert the connectors on the bolts between the two pieces of wood, and use washers under the nuts.

Joining plates. 'Sandwich' timber meeting end-on between two wooden plates of the same width. Glue the joint and stagger the screws to prevent their points clashing.



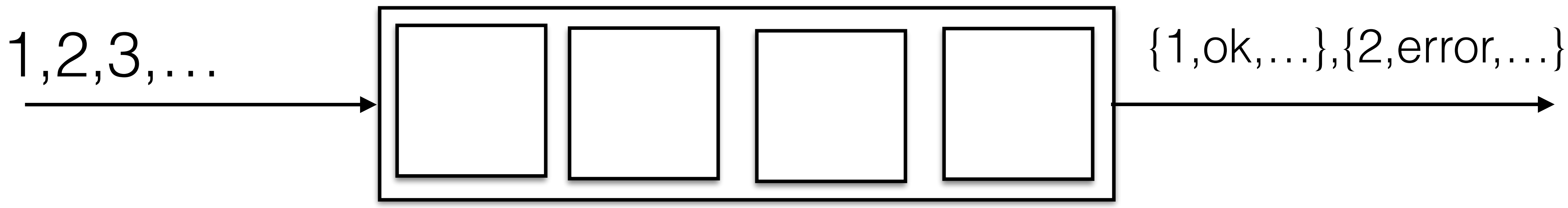

Start
F1
F2
F3
End


Start
Split
F1
F2
Merge
End
F3
F4



Errors are forwarded though the network

All “jobs” are numbered



The input is a stream of messages
1,2,3,4

The output is a stream of replies

No packets are lost

- We’re building apps and websites
- We should be building components that can be wired together

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 are not pipeable)

Summary--what's most important.

To put my strongest concerns in a nutshell:

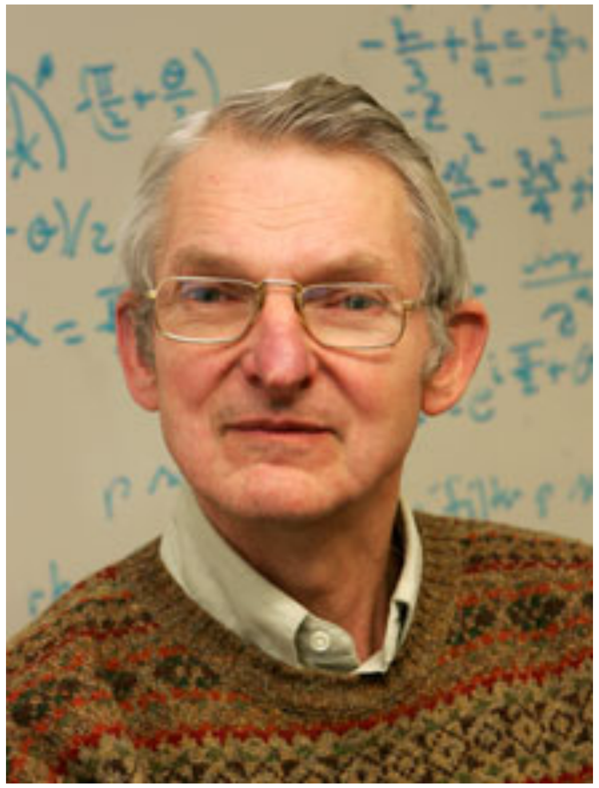
1. We should have some ways of coupling programs like garden hose--screw in another segment when it becomes when it becomes necessary to massage data in another way. This is the way of IO also.

2. Our loader should be able to do link-loading and controlled establishment.

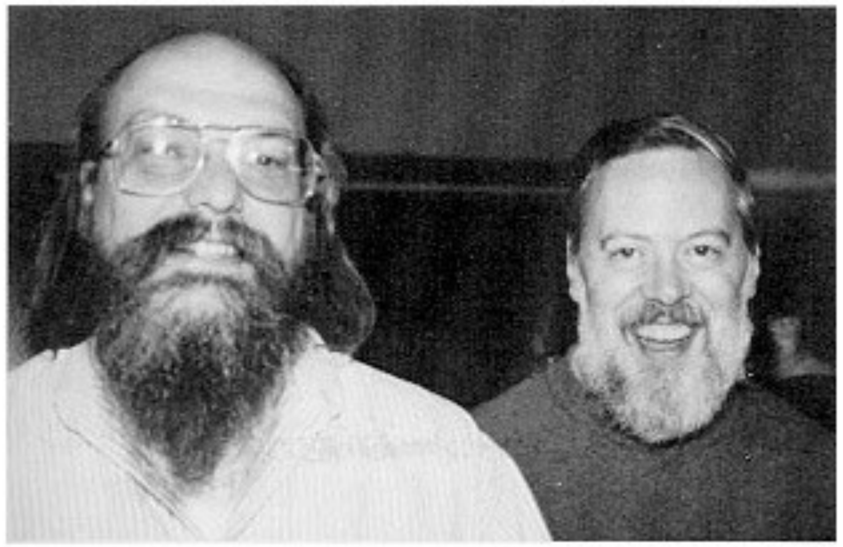
3. Our library filing scheme should allow for rather general indexing, responsibility, generations, data path switching.

4. It should be possible to get private system components (all routines are sytem components) for buggering around with.

M. D. McIlroy
Oct. 11, 1964



M.Douglas McIlroy



“Doug has been explicit in saying that he very nearly exercised managerial control to get pipes installed.”

“Point 1's garden hose connection analogy, though, is the one that ultimately whacked us on the head to best effect.”

http://cm.bell-labs.com/cm/cs/who/dmr/mdmpipe.html

Linda Tuple Spaces

Linda Tuple Spaces

- Shared Whiteboard
- Gelernter and Carriero 1986
- More declarative than message passing
- We just create jobs and don’t know who will do them

Tuple Space Operations

- out (adds a tuple to the store)
- in (reads a tuple and removes it from the store)
- rd (reads a tuple)
- eval (create a new parallel process)

Hypertext

Hypertext

- 1960’s - Ted Nelson - Xanadu (first approximation to Xanadu was 1998)
- 1962 - Douglas Engelbart - NLS (oN Line System)
- 1963 - Ted Nelson coins the word “Hypertext”
- 1980 - Tim Berners Lee makes a simple hypertext system
- 1987 - TBL Makes WWW
- 1987 - Apple makes Hypercard

- WWW is not hypertext
- HTML is not hypertext
- HTML links are not hypertext links

404

Page not found

All web pages are not writable

How to correct a typo on a web page

Correcting a typo (1)

1. Learn GIT
2. Locate the program that creates the page
3. Locate the typo in the source code
4. Correct the typo and test
5. Send a push request to the maintainer of the site

Correcting a typo (2)

1. Select the text
2. Type in the correction
3. All people observing the page see the change after a propagation delay

Xanadu

- Ted Nelson’s Hypertext system
- https://en.wikipedia.org/wiki/Project_Xanadu#Original_17_rules
    - No data is ever lost - no 404’s
    - All data is secure
    - Every user can read write and store data
    - …

Two fun
hobby projects
to try at home
and change the world

Project 1

Link to a content hash not a name

<a href="http://anysite/sha256/af34bc..3da45f2">name</a>

- Go to ANY website
- Request a content by SHA256 (or MD5)
- Immune to people-in-the-middle
- Return data or "a nearer website"

Theory

- Kademlia
- Chord

Projects

- https://datproject.org/
- https://ipfs.io/

Project 2

Elastic Links

Hypertext links
should be bi-directional

Links should not break if you
move an endpoint

*How do you implement this
at planetary scale?*

The end

Finally

Programs that are not secure and cannot be remotely controlled should not be written

All Apps should be scriptable

All Apps should be composable