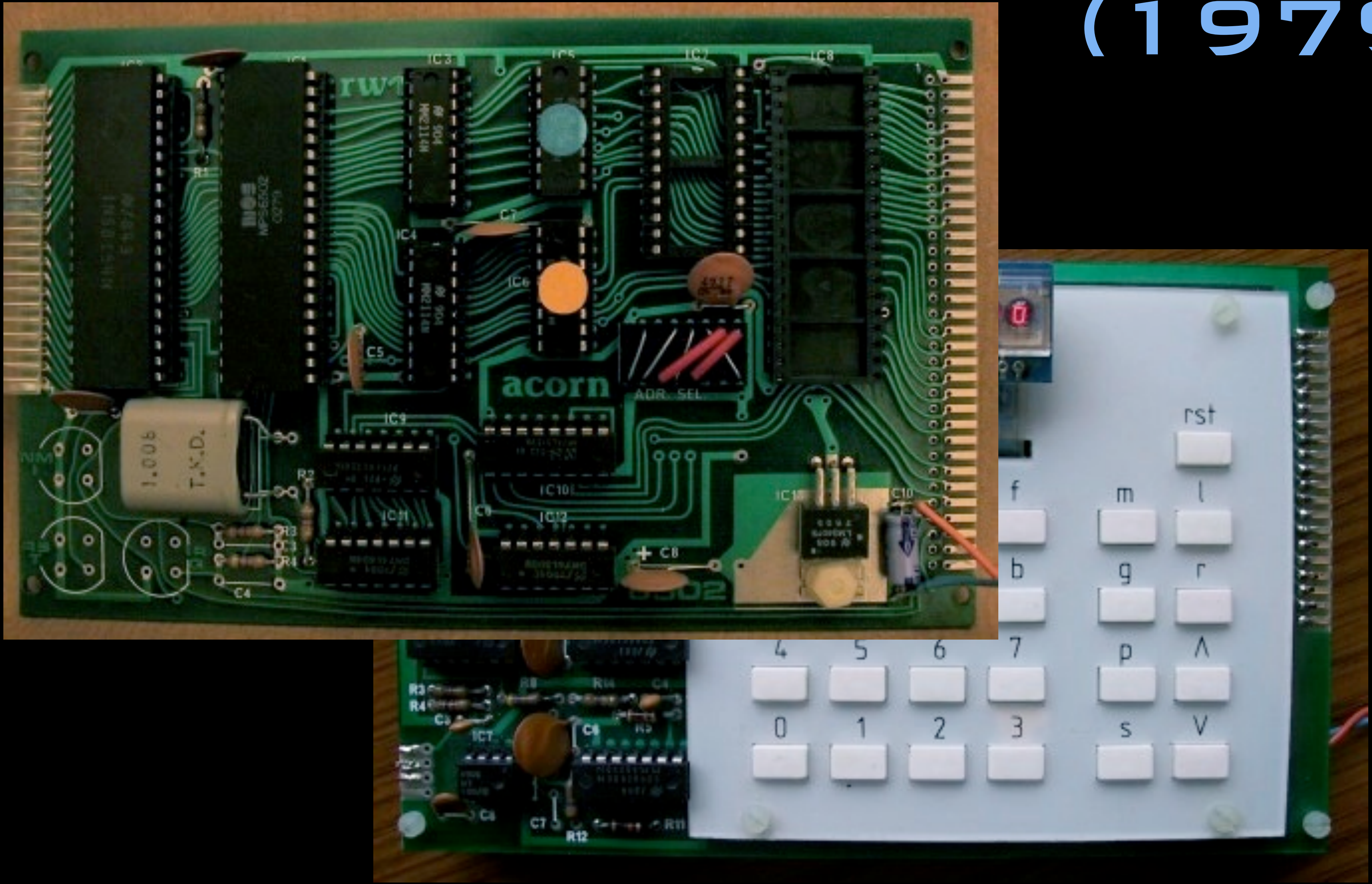
Programming the Virtual Infrastructure

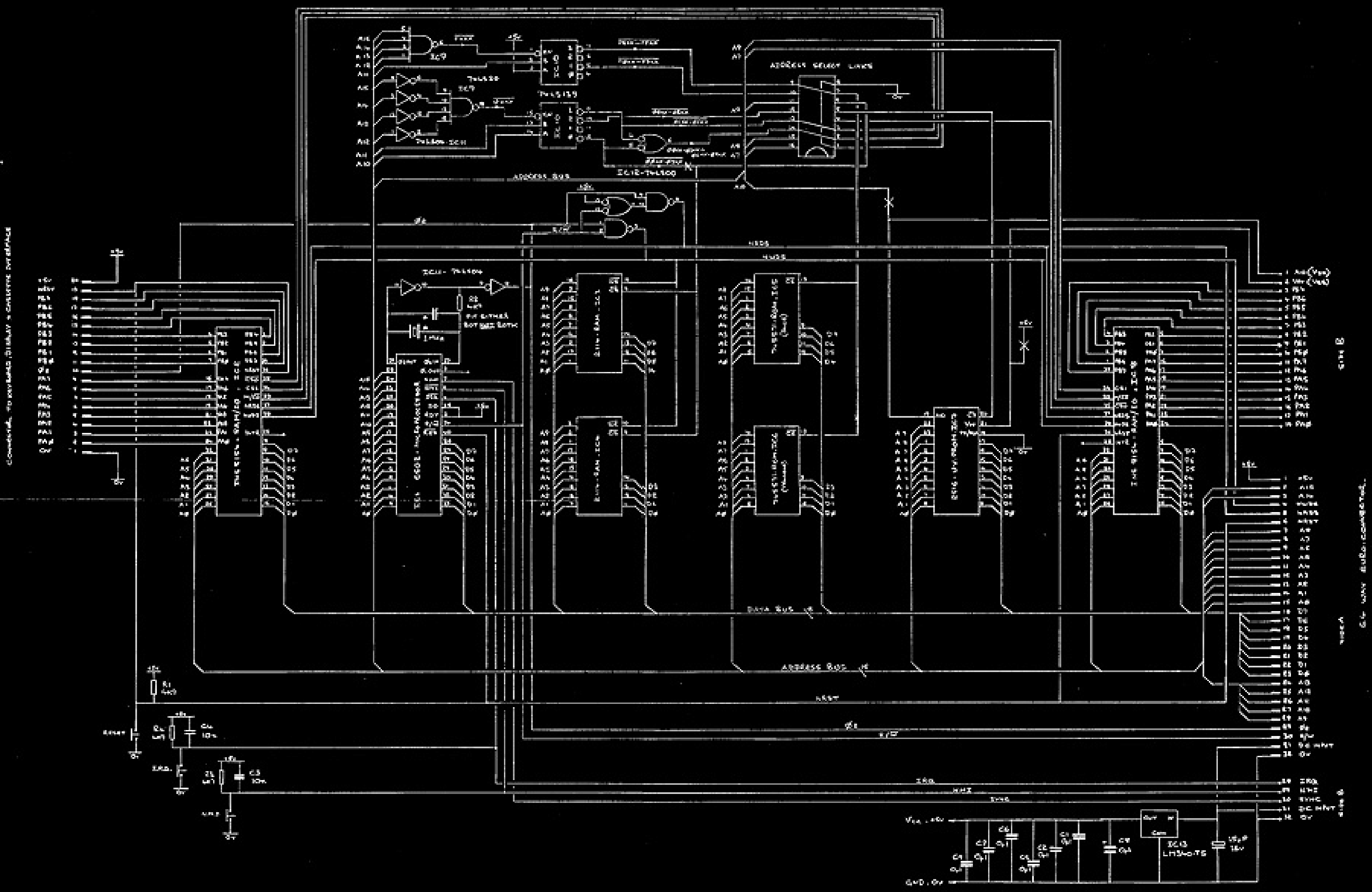
Paul Anderson <dcspaul@ed.ac.uk>

http://homepages.inf.ed.ac.uk/dcspaul/publications/lisa-2008-talk.pdf

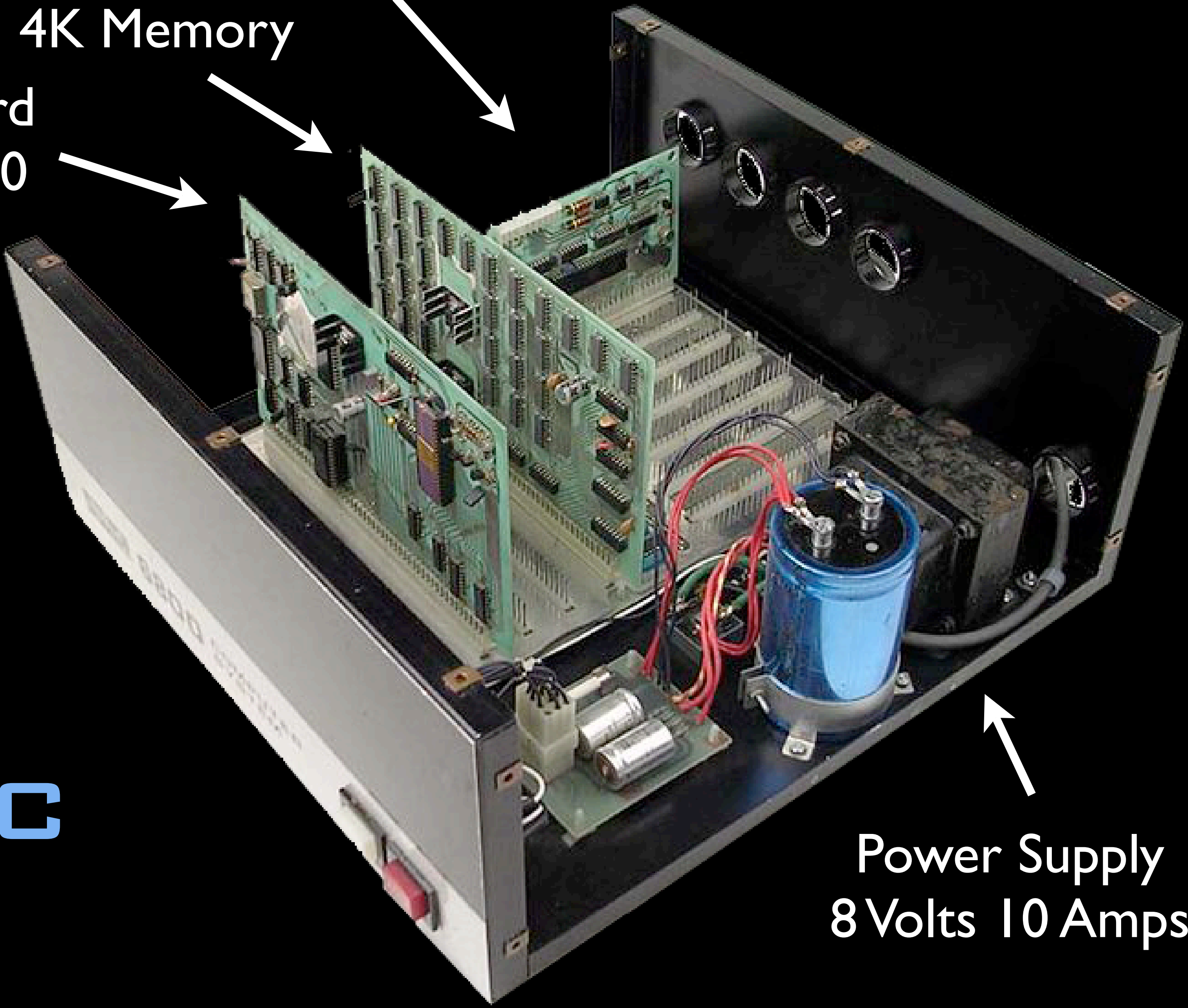
Acorn System I

(1979)





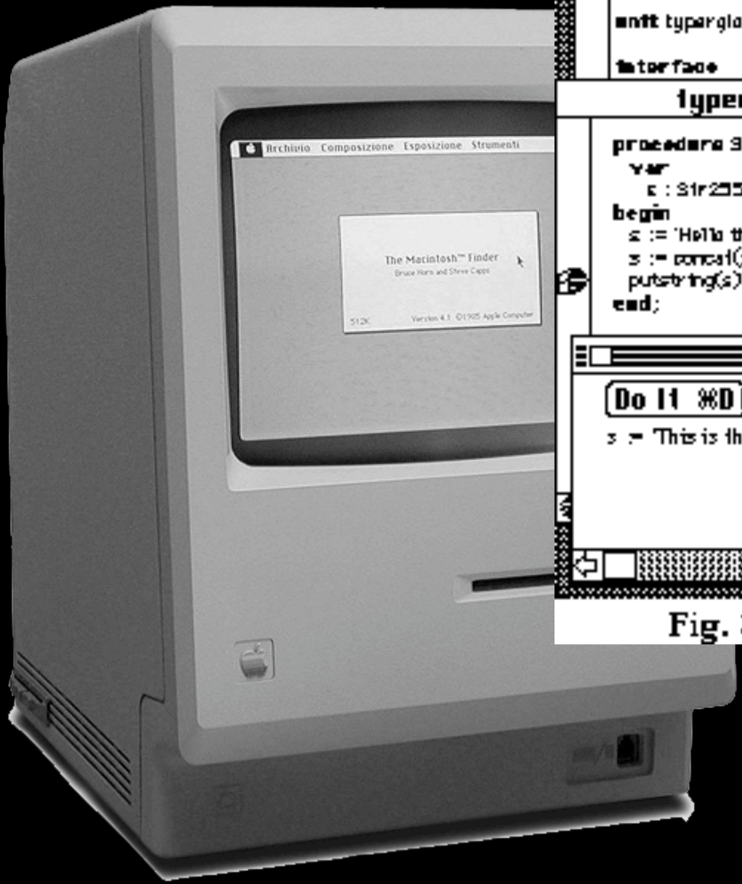
ADDRESS SELECT LINKS
ADDRESS BUS
DATA BUS
ADDRESS BUS
RESET
IRQ
NMI
SYNC
IC13 LM340-T5
GND, 0V

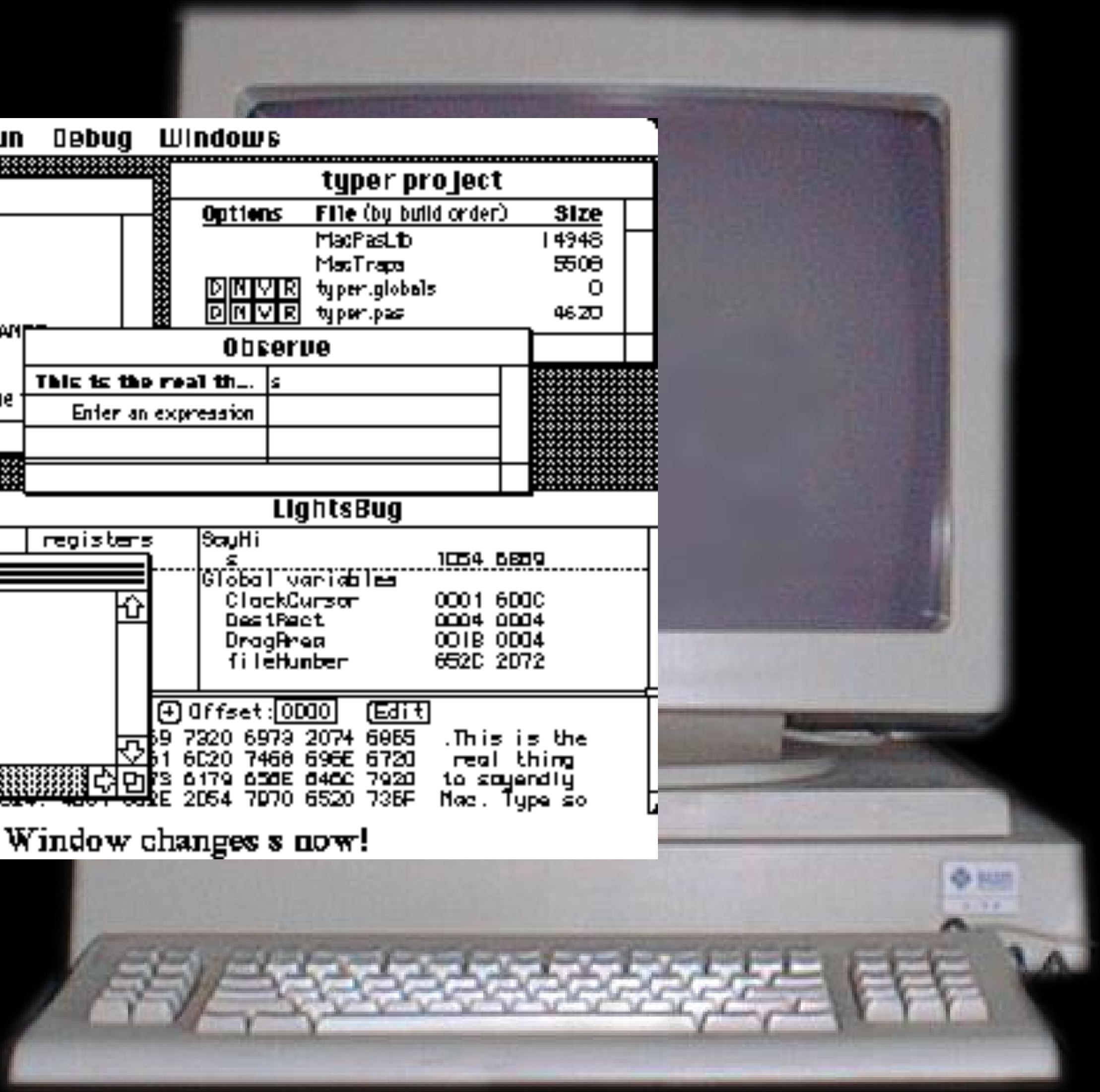

MP-C Serial Interface
MP-M 4K Memory
MP-A CPU Board
0.9MHz MC6800
Power Supply
8 Volts 10 Amps

SWTPC 6800

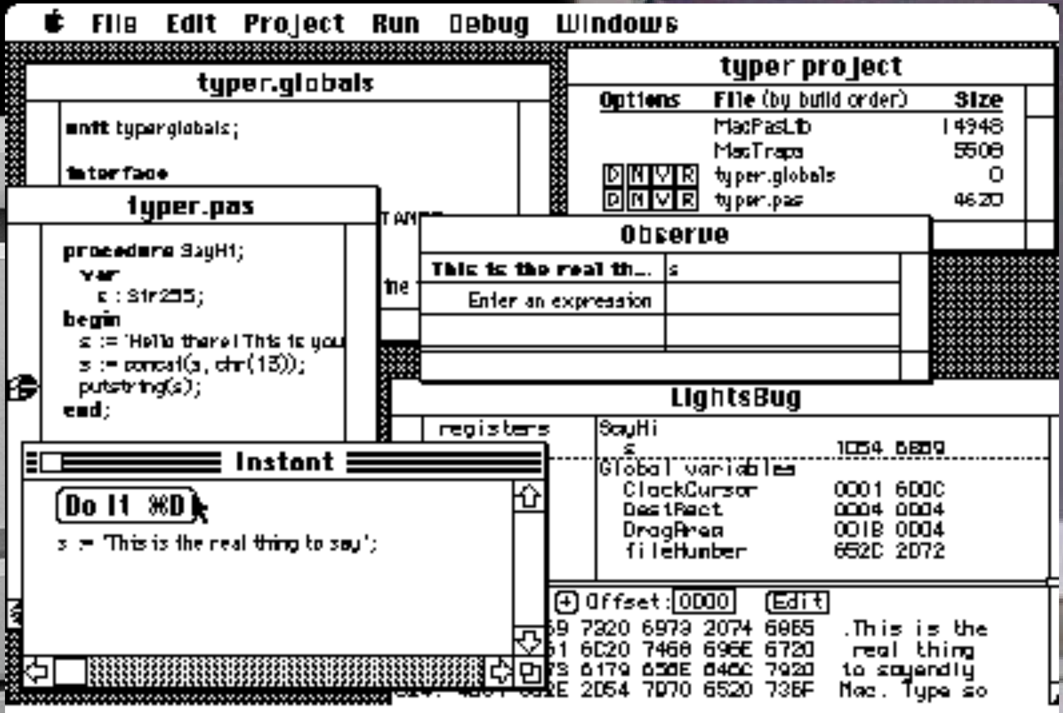
128K Mac

(1984)





Sun 3/50



```
unit typerglobals;

interface
```

```
procedure SayHi;
  var
    s : Str255;
begin
  s := 'Hello there! This is you
  s := concat(s, chr(13));
  putstring(s);
end;
```

```
s := 'This is the real thing to say';
```

Options	File (by build order)	Size
	MacPasLib	14948
	MacTraps	5508
D N V R	typer.globals	0
D N V R	typer.pas	4620

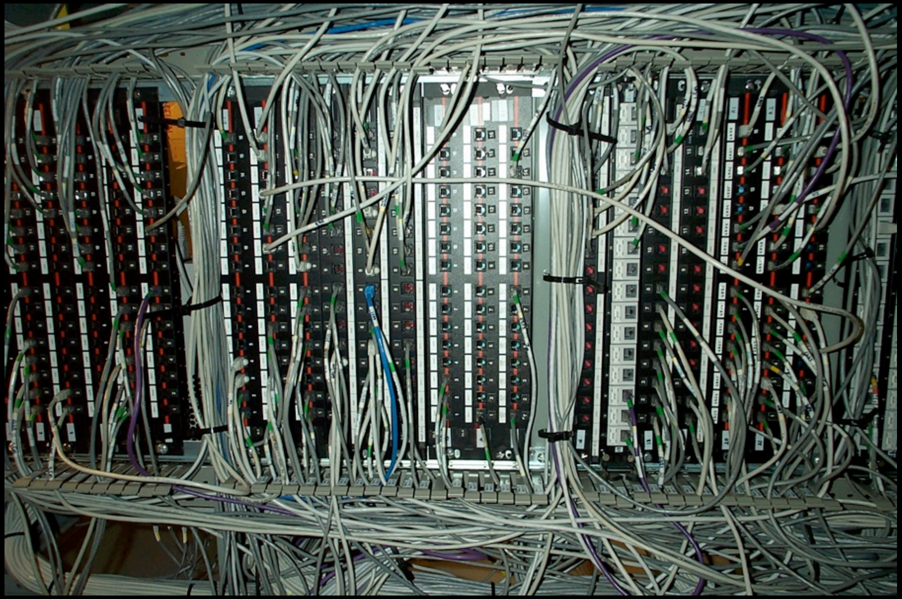
This is the real th...	s
Enter an expression	

```
SayHi
  s                       1054 6889
Global variables
  ClockCursor             0001 600C
  DestRect                0004 0004
  DragArea                001B 0004
  fileNumber              652C 2072
```

```
59 7320 6973 2074 6865   .This is the
61 6C20 7468 696E 6720    real thing
73 6179 656E 646C 7920    to sayendly
2E 2054 7970 6520 736F   Mac. Type so
```

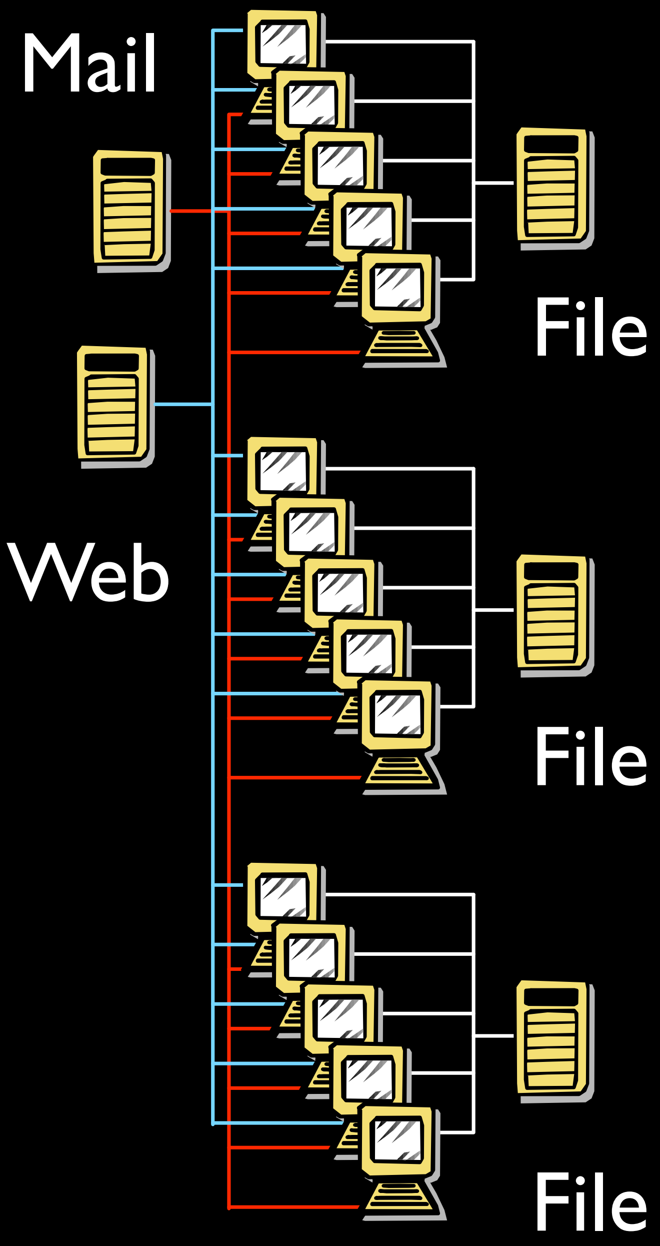
Fig. 3 The Instant Window changes s now!

- Which machine is connected to which network?
- Which disk is connected to which machine?



- What software goes on each machine?
- Someone else creates the software and the hardware ...

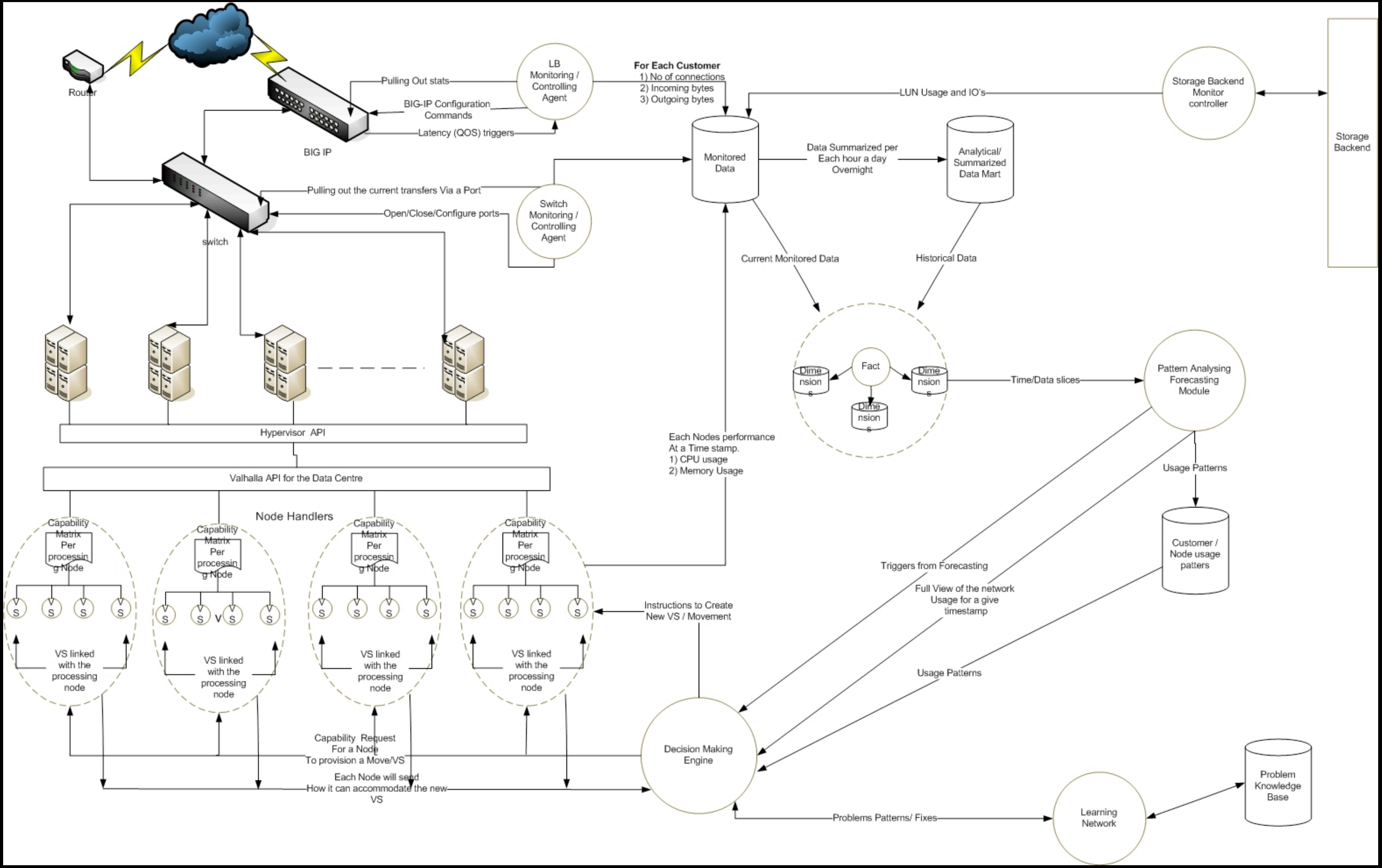
System Configuration



Virtualisation

- We can now virtualise ...
  - the processor
  - the network
  - the storage
- So we no longer need physical intervention to do most reconfiguration tasks
  - replacing failed machines (we can migrate off)
  - transferring to a machine with more resources
  - adding more storage
  - reconfiguring network topology
  - etc ....

Flexiscale

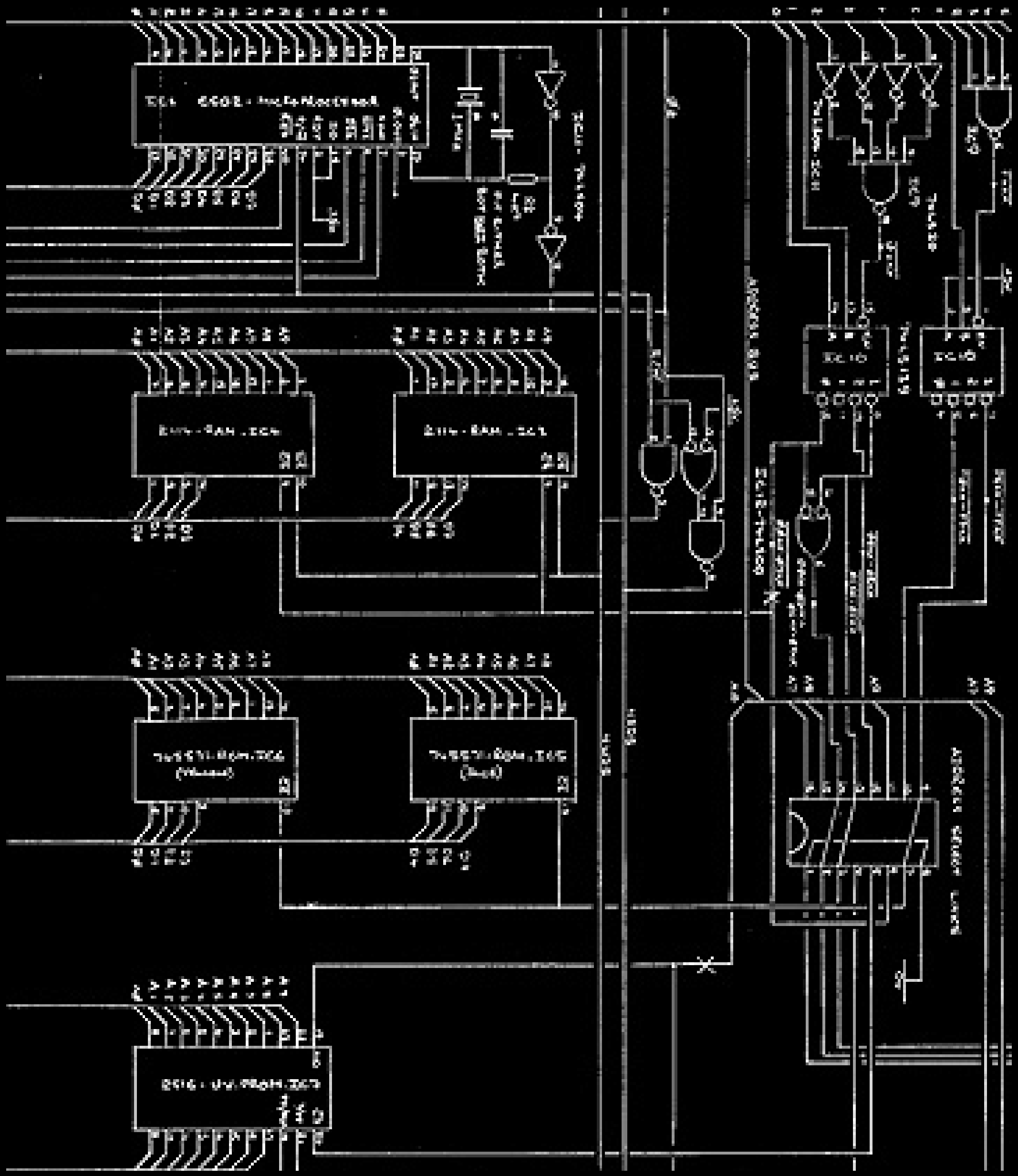


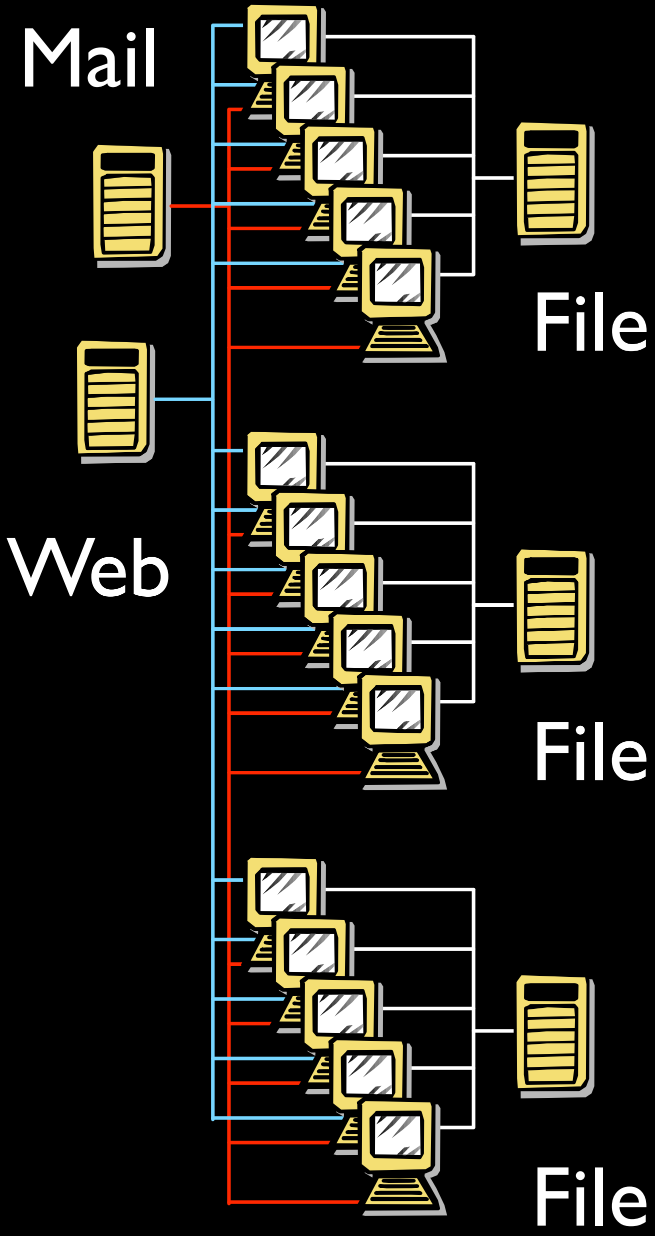
Router
BIG IP
Pulling Out stats
BIG-IP Configuration Commands
Latency (QOS) triggers
LB Monitoring / Controlling Agent
For Each Customer
1) No of connections
2) Incoming bytes
3) Outgoing bytes
LUN Usage and IO's
Storage Backend Monitor controller
Storage Backend
switch
Pulling out the current transfers Via a Port
Open/Close/Configure ports
Switch Monitoring / Controlling Agent
Monitored Data
Data Summarized per Each hour a day Overnight
Analytical/ Summarized Data Mart
Current Monitored Data
Historical Data
Fact
Dimensions
Dimensions
Dimensions
Time/Data slices
Pattern Analysing Forecasting Module
Hypervisor API
Valhalla API for the Data Centre
Node Handlers
Capability Matrix Per processing Node
S
V
VS linked with the processing node
Each Nodes performance At a Time stamp.
1) CPU usage
2) Memory Usage
Usage Patterns
Customer / Node usage patters
Triggers from Forecasting
Full View of the network Usage for a give timestamp
Instructions to Create New VS / Movement
Usage Patterns
Capability Request For a Node To provision a Move/VS
Each Node will send How it can accommodate the new VS
Decision Making Engine
Problems Patterns/ Fixes
Learning Network
Problem Knowledge Base

The charm of history and its enigmatic lesson consist in the fact that, from age to age, nothing changes and yet everything is completely different.

ALDOUS HUXLEY

An Analogy ?





Programming the Virtual Infrastructure

- In some sense, we can think of configuration as "programming the virtual infrastructure"
  - the function of the infrastructure (hardware) depends on the configuration (program)
  - (but, of course, the problem is not identical)
- The "virtual" nature of the infrastructure does not fundamentally change the nature of the configuration problem
  - but it does significantly increase the complexity
  - the configuration of the inside and the outside of the virtual machines are intimately related

Learning from History?

- When something becomes sufficiently complex, a new layer of abstraction often develops to enable things to move forward
- Each stage comes with new techniques, theories and specialists
- Can we learn anything from the way in which programming languages and software engineering have developed?
  - maybe there are some specific analogies?
  - or perhaps just lessons in the process?
- Lets look at some history ...

In the Beginning ...

- The hardware engineers wrote their own programs
- Writing machine code was slow and error-prone
- “Automatic Programming” was proposed as the solution
- Fortran was one of the first high-level languages

Efficiency

The programmer attended a one-day course on Fortran and spent some more time referring to the manual.

He then programmed the job in four hours, using 47 Fortran statements. These were compiled by the 704 in six minutes, producing about 1000 instructions.

He estimated that it might have taken three days to code this job by hand.

From a 1957 paper on Fortran

Correctness

He studied the output (no tracing or memory dumps were used) and was able to localise his error in a Fortran statement he had written.

He rewrote the offending statement, recompiled and found that the resulting program was correct.

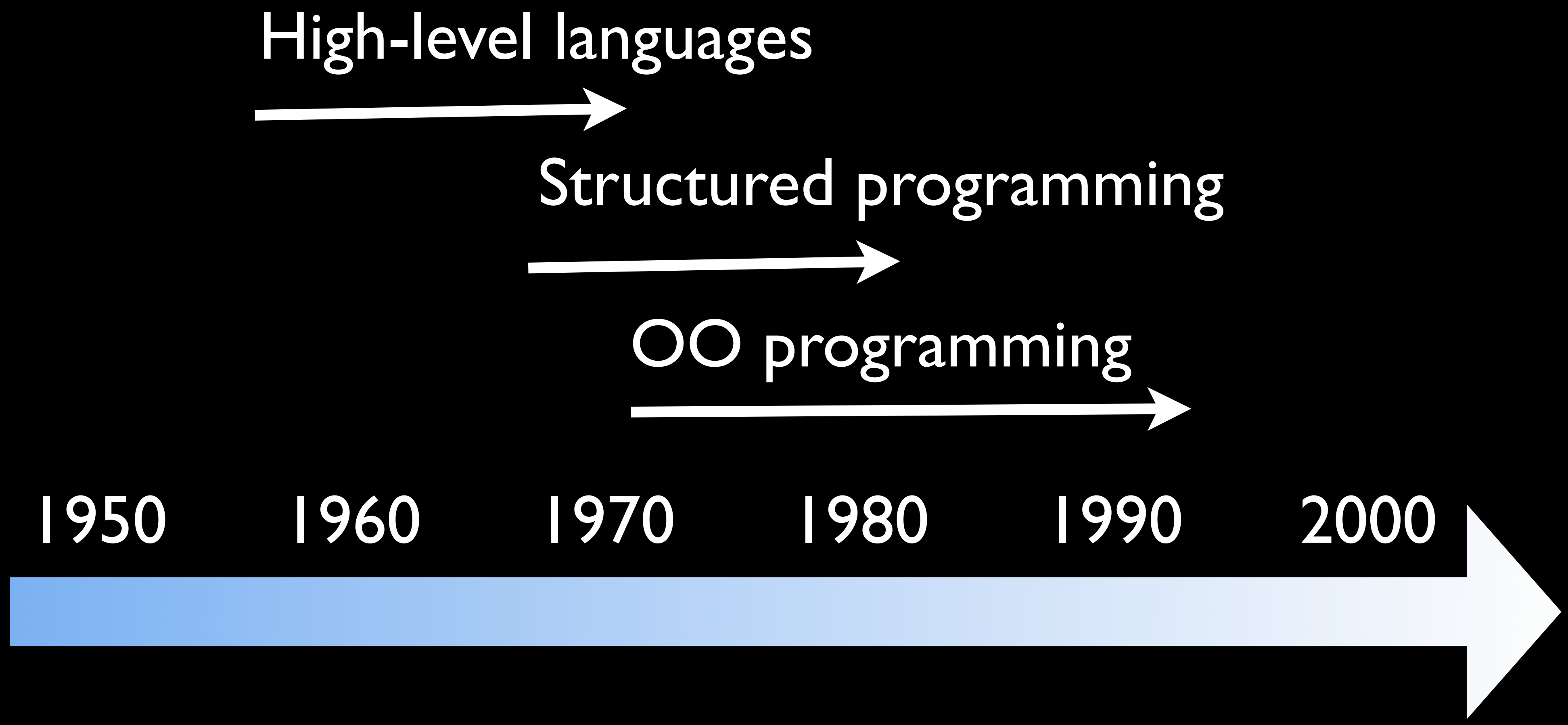
He estimated that it might have taken three days to code this job by hand, plus an unknown time to debug it.

From a 1957 paper on Fortran

Some Other benefits

- Programs were now portable between different types of hardware
  - and programmers did not have to learn multiple machine codes
- Control of the machines was opened up to those who wanted to make use of them
- We would like to have the same properties for programming our virtual infrastructure:
  - efficiency
  - correctness
  - portability
  - usability ...

Programming Languages



Fortran 1954

Algol 1958, C 1972

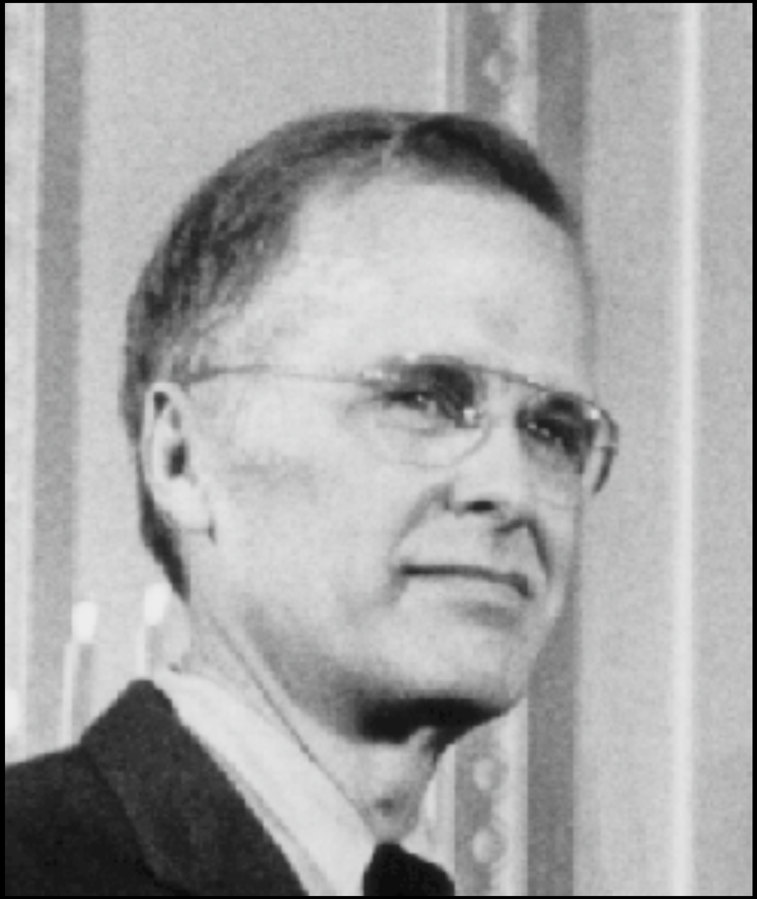
Simula 1967, Smalltalk 1980, C++ 1989, Java 1995

Complexity?

- "Essential Complexity" is the complexity inherent in a problem
  - virtual machines are significantly increasing the essential complexity of the configuration problem
- "Accidental complexity" is the complexity created as part of a (bad) solution
- Fred Brooks claims that modern programming has eliminated most of the accidental complexity
  - *No Silver Bullet - Essence and Accidents of Software Engineering*
- This does not feel true for configuration ...

Language Development

- New approaches can take 15 years to become accepted practice
- Language design has become more formal and academic
  - but features only survive when they prove themselves in practice
- There is a general trend towards higher levels of abstraction
  - but different levels are appropriate for different applications
- Languages can be mixed relatively easily



"We simply made up the language as we went along. We did not regard language design as a difficult problem, merely a simple prelude to the real problem: designing a compiler which could produce efficient programs."

John Backus

Developer of Fortran & inventor of BNF

Configuration Languages

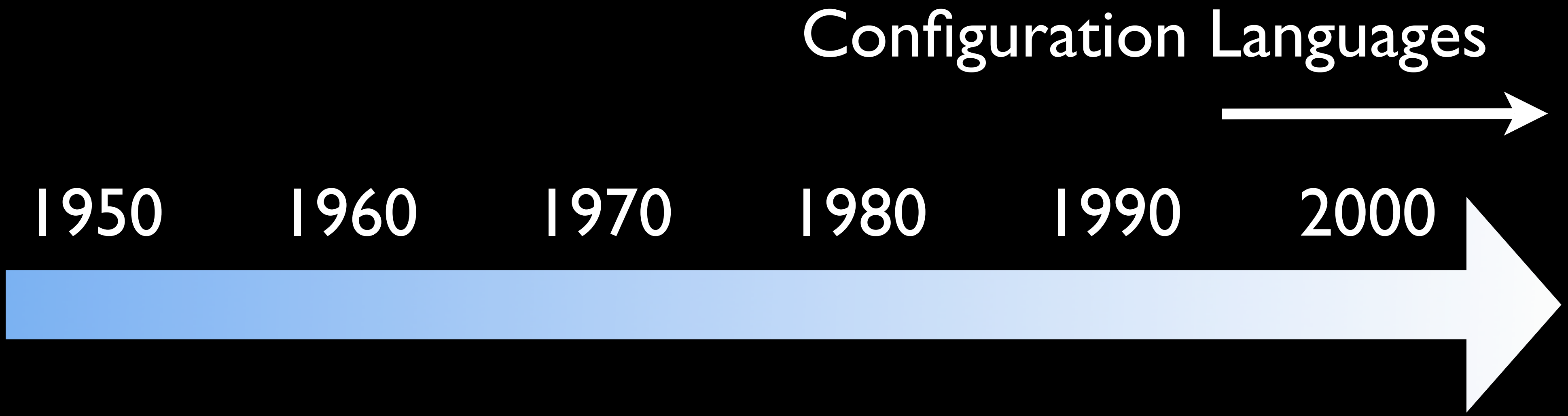
- Several features of programming languages have been exploited in the design of configuration languages
  - modules and objects
  - aspects, prototype-based languages
- But configuration languages usually describe some "desired configuration state"
  - this is different from a conventional programming language, which usually describes a computation
- So some analogies are more interesting that others
  - agile programming
  - declarative programming languages ...
- How do these apply to virtual infrastructures ?

Agile Configuration ?

- Why is (for system configuration) ...
  - Perl more popular than Java ?
  - Cfengine more popular than CIM ?
- Perhaps because these are more agile ?
  - system configurations change rapidly
  - new components need to be incorporated quickly
  - “agile” development is traditional
- What about the virtual infrastructure ?
  - perhaps the “outside” management of the virtual machines is a more stable problem ?
  - or not ?!

Configuration Languages

How suitable are these for virtual infrastructures?



LCFG 1994 (?)

Cfengine 1997 (?)

BCFG 2003 (?)

Puppet 2005 (?)

Approximate dates of first publication!

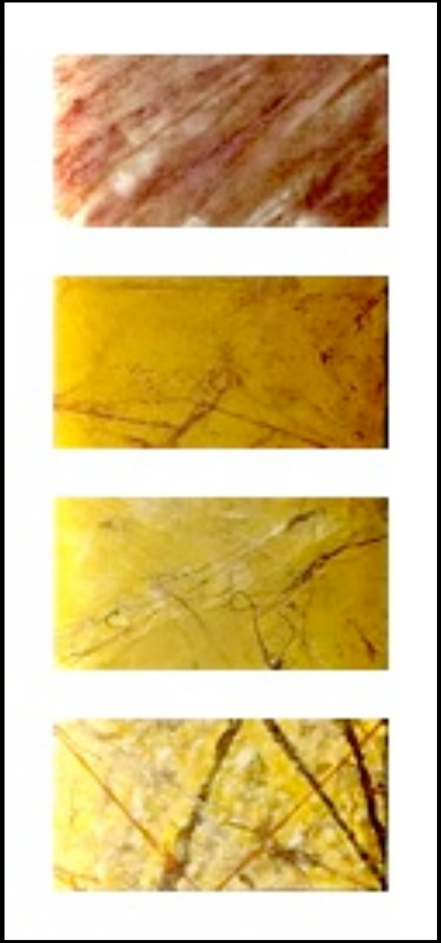
Declarative Descriptions

- "Declarative" descriptions say
  "what you want" rather than "how to get there"
- They have some important advantages over "imperative" descriptions:
  - the sequence of statements is not important
  - we don't have to worry about "idempotence"
    (multiple executions having a bad effect)
  - the statements describe the desired state
    (it is easier to be confident that they are correct)
  - It is easy to combine requirements from different people (aspects)

Declarative Configuration

- Declarative specifications have become the norm for system configuration tools:
  - Cfengine specifies that a file should contain a line
  - LCFG specifies the value of "mailrelay" resource
  - MLN specifies the virtual machine configurations
- BUT - the tool needs to compute and implement the changes necessary to bring the system, into the desired state
  - this is easy when specifying the content of a file
  - it is a serious planning problem when specifying the state of a virtual infrastructure
  - the intermediate states are important

MLN



```
switch lan {}
host one {
  network eth0 {
    switch lan
    address 10.0.0.1
    netmask 255.255.255.0
  }
}
host two {
  network eth0 {
    switch lan
    address 10.0.0.2
  netmask 255.255.255.0
  }
}
```

Declarative Programming

- Prolog is the original declarative programming language (1970s)
  - this is widely used, but only in restricted applications
- In theory, this can support both
  - the specification of the configuration itself
  - and the planning of the transitions
- Such fully-automated reasoning is not appropriate for many tasks
  - the outcome may be unexpected
  - computation times may be unpredictable
  - it may be not clear why a particular decision is made

Constraints

- Constraint Logic Programming (CLP) is a technique for solving declarative constraints
- This has proven useful in generating network and system configurations from policies
  - Every network segment must have at least two DHCP servers
  - No component of network should be a single point of failure
- It seems appropriate for virtual infrastructures
  - VM X must not be on the same physical machine as any VM owned by company Y
  - VMs A and B must be on the same network

Constraint Properties

- Specifications from different sources can be combined without worrying about their interaction
  - VM X must be on a machine owned by Y
  - All VMs on machines owned by Y must be connected to network Z
- Specifications are “loose” enough so that autonomic systems have room to make choices
  - VM X can be hosted anywhere where the filesystem Y is available
- Constraints are also useful in planning
  - There must always be at least one active VM hosting database X

Constraint Problems

- Specifying things in terms of constraints is not always natural ...
- It is easy to underspecify
  - if you don’t specify that something is ***not*** valid, then the system might well attempt it!
- It is easy to overspecify
  - this leaves no “room” for autonomic adjustment
- Solving general constraints is computationally hard
  - progress is being made with the technology
  - restrictions can be used to simplify the problem
  - more loosely specified problems are harder

Automatic Programming

- So ... automatic programming of Virtual Infrastructures is hard
    - it is not easy to specify correctly what is required
    - translating "high-level" requirements into implementable specifications is hard
    - the languages are immature and contain a lot of accidental complexity
    - the solutions can be difficult to compute
    - automatic solutions may be difficult to understand and trust
    - the planning of the change implementation is important (and difficult) as well as the final configuration state

Operating Systems

- Operating systems provide a framework for integrating the programming of a machine
- Configuration tools (such as LCFG) provide a framework for system configuration
  - these have different requirements
  - configuration frameworks are much less advanced
- Virtual infrastructures require a different kind of framework again
  - existing attempts at frameworks for virtual machine management are very simple
  - they are not well-suited to more complex automation

Frameworks

- One promising approach is to use a framework which allows humans and automatic processes to collaborate smoothly ...

In the context of modern distributed, virtual organisations, when attempting any sort of collaborative synthesis task, it is likely to require the capabilities of both human and computer agents.

The I-X Project

The I-X Framework

- The I-X Framework has been used for planning, for example, emergency response
- Decisions are made by a combination of human and automatic processes -
  - the system may present alternatives for the user to select
  - decisions may be passed to other (remote) users, or delegated to automatic processes
  - “canned” solutions may be stored for configurations or plans
  - the user may make explicit choices to constrain automatic solutions

Distributed Configuration

- The underlying model of an infrastructure is also different from a programmable machine.
  For example:
  - the target "machine" is distributed (and unreliable)
  - the source of the configuration is distributed - different parts of the infrastructure are under the control of different people
- These features often create difficulties for configuration tools
  - although tools such as cfengine emphasise the autonomy of the individual machines
  - there is still a tension between this and the need for central "control"

VMs as Agents

- Virtual infrastructures emphasise these difficulties
  - the machines migrate around the infrastructure (and across infrastructures)
  - the "inside" of the virtual machine may be under the control of someone different
  - it may have different "goals" from the physical machine
- This makes it tempting to think of virtual machines in terms of "reactive agents"
  - this is one example of a radically different approach to the problem ...

Reactive Systems

Reactive systems are systems that cannot adequately be described by the relational or functional view.

The relational view regards programs as functions from an initial state to a terminal state.

Typically, the main role of reactive systems is to maintain an interaction with their environment, and therefore must be described (and specified) in terms of their on-going behaviour

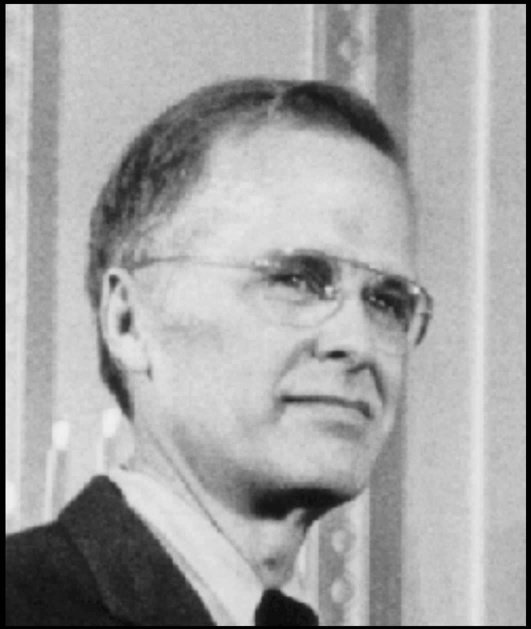
Amir Pnueli (1986)

VMs as Agents

- Of course, thinking in terms of "agents" does not "solve the problem" ...
- There is no general method for co-ordinating the agents towards some common goal
  - Eg. "Build me a web service"
- The individual agents must have implementable specifications for their behaviour
  - probably a declarative specification
- There is much less certainty in the resulting configuration
  - it is harder to prove certain properties

Some Final Thoughts

- Fully automatic “programming” of virtual infrastructures is very hard and unlikely to be practical in the near future
- Existing configuration techniques are inadequate
- Perhaps the demands of virtual infrastructures will prompt a radical rethink of these
- But language design takes time to evolve and eliminate accidental complexity
- And new models take time to gain trust and acceptance
- Smooth collaboration between manual and automatic procedures is looks like an important approach



The team was heavy with math training because so much of computing at the time was numerical analysis and mathematics, but it was an eclectic group.

A crystallographer, a cryptographer, a chess wizard, an employee lent from an aircraft manufacturer, a researcher borrowed from M.I.T. and a young woman who joined the project straight out of Vassar College.

John Backus

"And in the background was the scepticism, the entrenchment of many of the people who did programming in this way at that time; what was called "hand-to-hand combat" with the machine"

IRVINE ZILLER