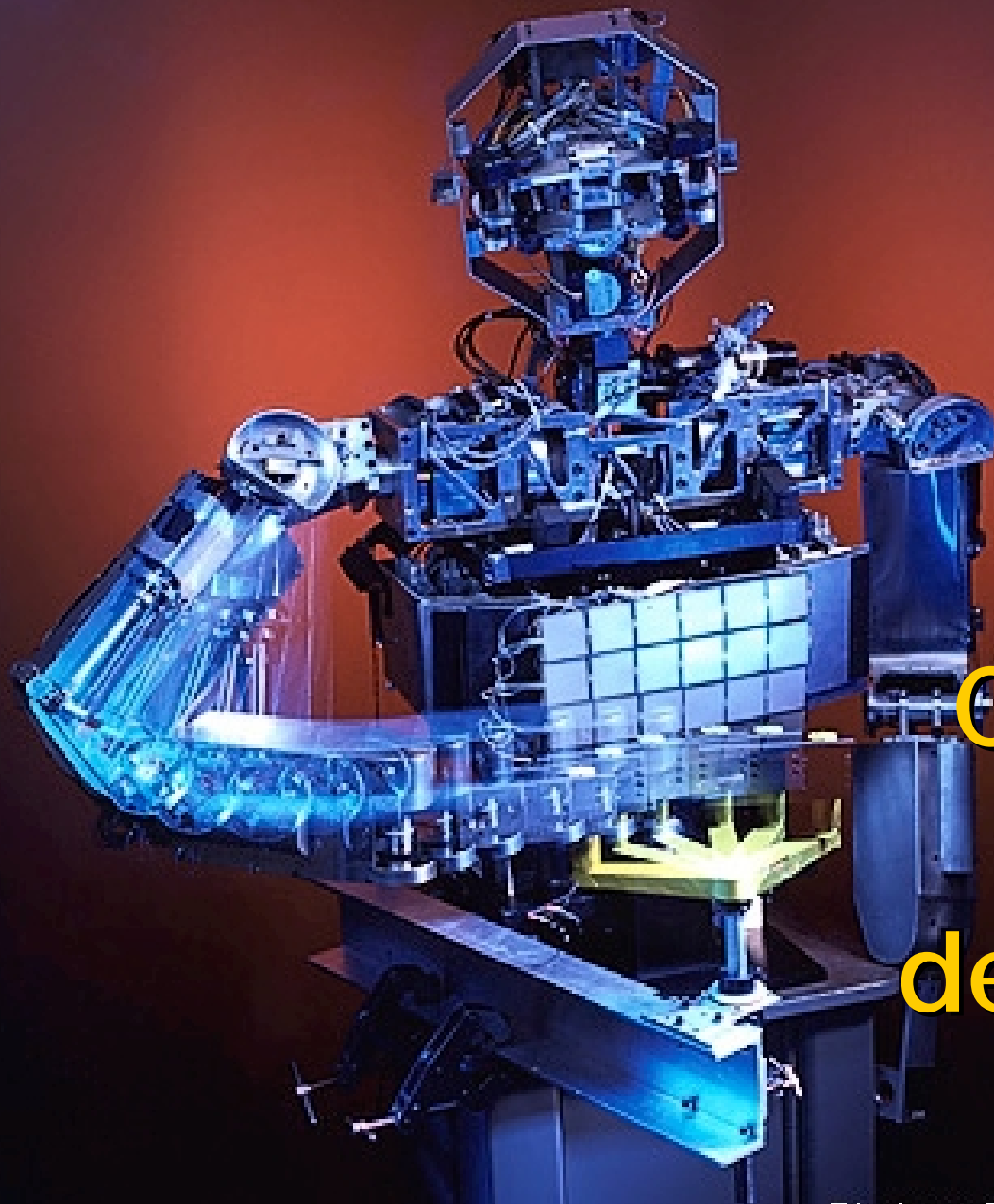


# Intrinsic Evolvability Hardware

A photograph of a man with a full brown beard and long hair, wearing a grey sweater, holding a light-colored dog. The man is looking at the dog with a smile. The background shows a residential street with trees and a house.

Can this  
system be  
constructed?

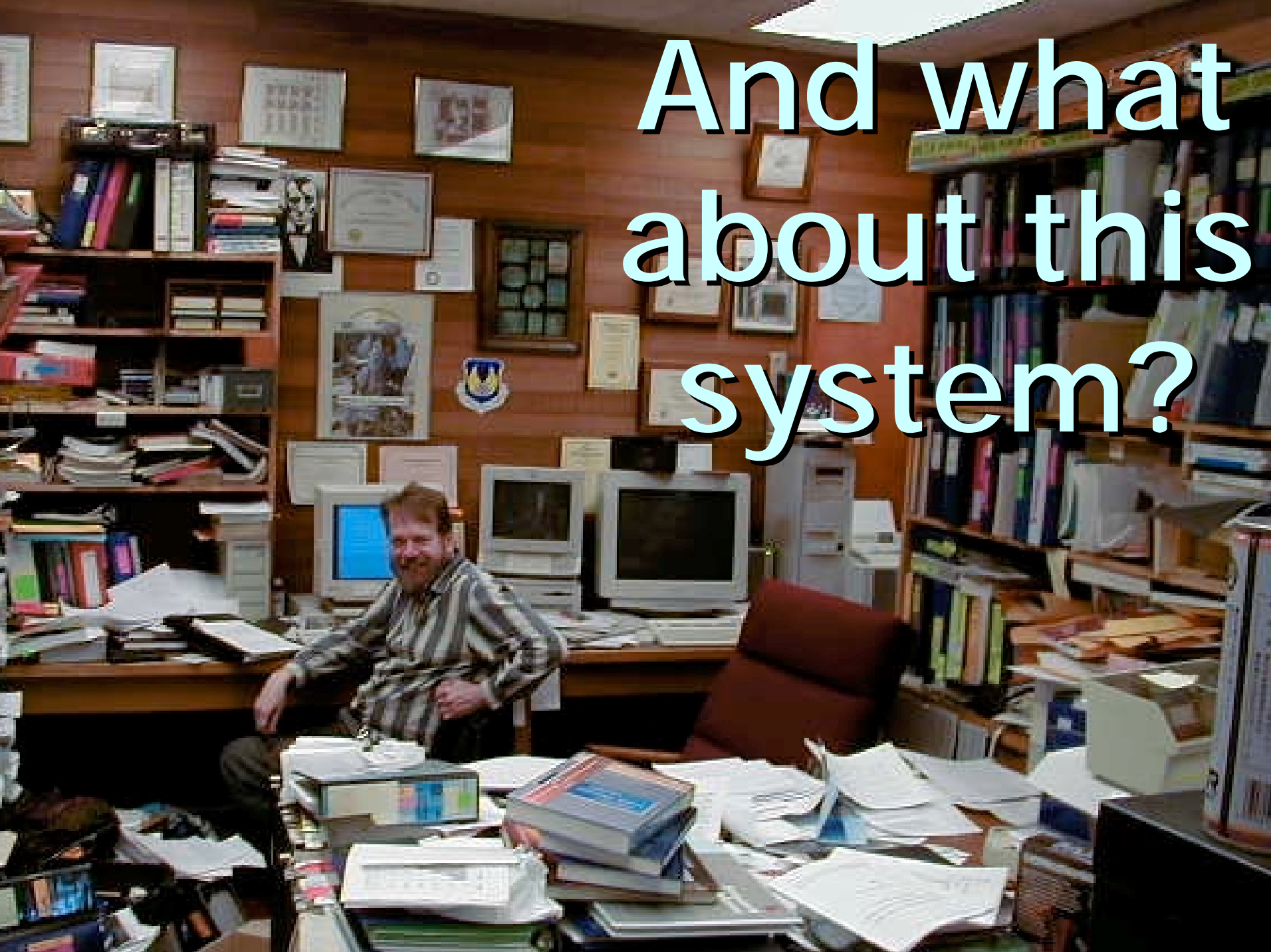


**COG system  
has been  
demonstrated**

# MIT, AI LAB



And what  
about this  
system?





# *Constructive Induction*

# I. MACHINE LEARNING FOR ROBOTICS

- It has long been the dream of many people to design a robot that learns from its own mistakes in a real environment.
- From Asimov's "I, ROBOT" to Furby, people have been enthralled with the concept of a thinking machine.
- However, getting a robot to learn from its mistakes is not all that futuristic.
- The advances in Artificial Intelligence, Neural Networks, Genetic Algorithms, Data Mining and Functional Decomposition have opened the doorway to a multitude of learning strategies.
- Today, these technologies have even found practical applications in areas as robotics and large data-base mining such as in the health care field for diagnosing diseases or other medical conditions.

- The aforementioned learning techniques are all very similar in some way.
- They update their parameters by an error or approximation method from sampled data and/or from some type of cost function.
- This is the general method of any learning system, but a distinction is made between *weak* and *strong* criteria.
- The aforementioned techniques utilize *weak criterion* learning.



- **Constructive Induction** (or Logic Synthesis in design automation) uses a similar method of evaluating its cost or fitness but uses a *strong criterion* which means that the system communicates concepts learned in symbolic form.
- This is difficult to ask a ANN or Fuzzy Network or Genetic Program for a symbolic reason as to what concepts it learned.
  - This method is used for **Learning Hardware**, and that is why it finds its applications in Robotics.

- Don't assume that this is always a humanoid or an android of Star Trek
- The Robot will
  - function in his/her/its environment,
  - learning from its mistakes.
- The learning procedure is fairly basic and consists of two stages:
  - **1) Learning Phase**
  - **2) Acting Phase**

- **Phase 1** is the phase where we examine our state and determine how we should construct and tune our network.
- **Phase 2** involves using our newly tuned network to interact with the environment. That is, running the network on a data set input.
- This particular 2-phase process is common to probably all the intelligence engines.
- Some perform the two phases only once and others continuously iterate through the two-phases.



- **Learning in Hardware** is not all that dissimilar to how human intelligence works.
- Logic algorithms derived from human knowledge tend to be highly accurate, very sophisticated, and very high quality.
- However, implementing these concepts into software becomes very inefficient.
- Learning Hardware uses the concept that human thought consists of **abstract use of symbols**.
- A symbolic representation of learning lends itself to hardware realization where symbolic mathematical optimization in hardware is performed with techniques such as graph coloring, satisfiability, and decomposition.
- Methods which **support Occam's Razor**.

This symbolic approach shouldn't lead us to the conclusion that a system (such as a robot) can only learn what data is fed to it.

That is, the power of generalizing is lost.

This is not true.

The concept of generalizing in symbolic logic is dealt with by relations.

Relations and multi-valued logic allow for intelligent generalizations.



Even in the case of a single bit data, we still have the concept of a *Don't Care (X)*.

During the logic synthesis stages from learnt data, logic is created in hardware.

This logic must provide an *output* for all possible *inputs*, even those it did not have access to during learning (i.e., data it was not trained on).

That output is now a *generalization*, and logically this generalization will hopefully show some intelligent reasoning.

Also, since the hardware is synthesized via minimization techniques, the generalization might also satisfy **Occam's Razor**.

(It is noted a powerful feature of ANN is their ability to generalize, in which generalization is more accurate for smaller highly efficient networks.)

A very simple example of learning from given data and generalizing.

# Approach of ATR in Japan

# Hugo De Garis announced the BrainBuilding and Robokoneko



# CAM-BRAIN MACHINE (CBM)

- The CAM-Brain Machine (CBM) is a piece of specialized "evolvable hardware" which grows and evolves cellular automata based neural network circuit modules in about a second.
- This is so fast that from now on it will be practical to build artificial brains by assembling tens of thousands of these quickly evolved modules into humanly architected artificial brains.
- The modules (evolved one at a time by the CBM) are downloaded into a gigabyte of RAM.



- The CBM can then update the 3D cellular automata cells in the RAM at a rate of 150 Billion a second, updating an artificial brain consisting of 32000 modules (40 million neurons) at a rate of 300 times a second, fast enough for real time control of our life sized kitten robot "Robokoneko".
- The shape and colors of the machine are symbolic.
  - The shape is supposed to represent a slice of cortex, and its colors (grey and white) represent the outer "grey matter" of the cortex (i.e. the neurons) and its inner "white matter" (the interconnecting axons).



- The electronic boards are placed in the grey section. The power supply is placed in the white section, as shown in the photos below.
- If you want to buy one of these machines, contact Dr. Michael Korkin of Genobyte Inc, Boulder, Colorado, USA. We have only enough programmable (and evolvable) chips (Xilinx XC6264) for about 8 machines.
- So if you want one, you should get your order in quickly, because 3 have already been signed for.

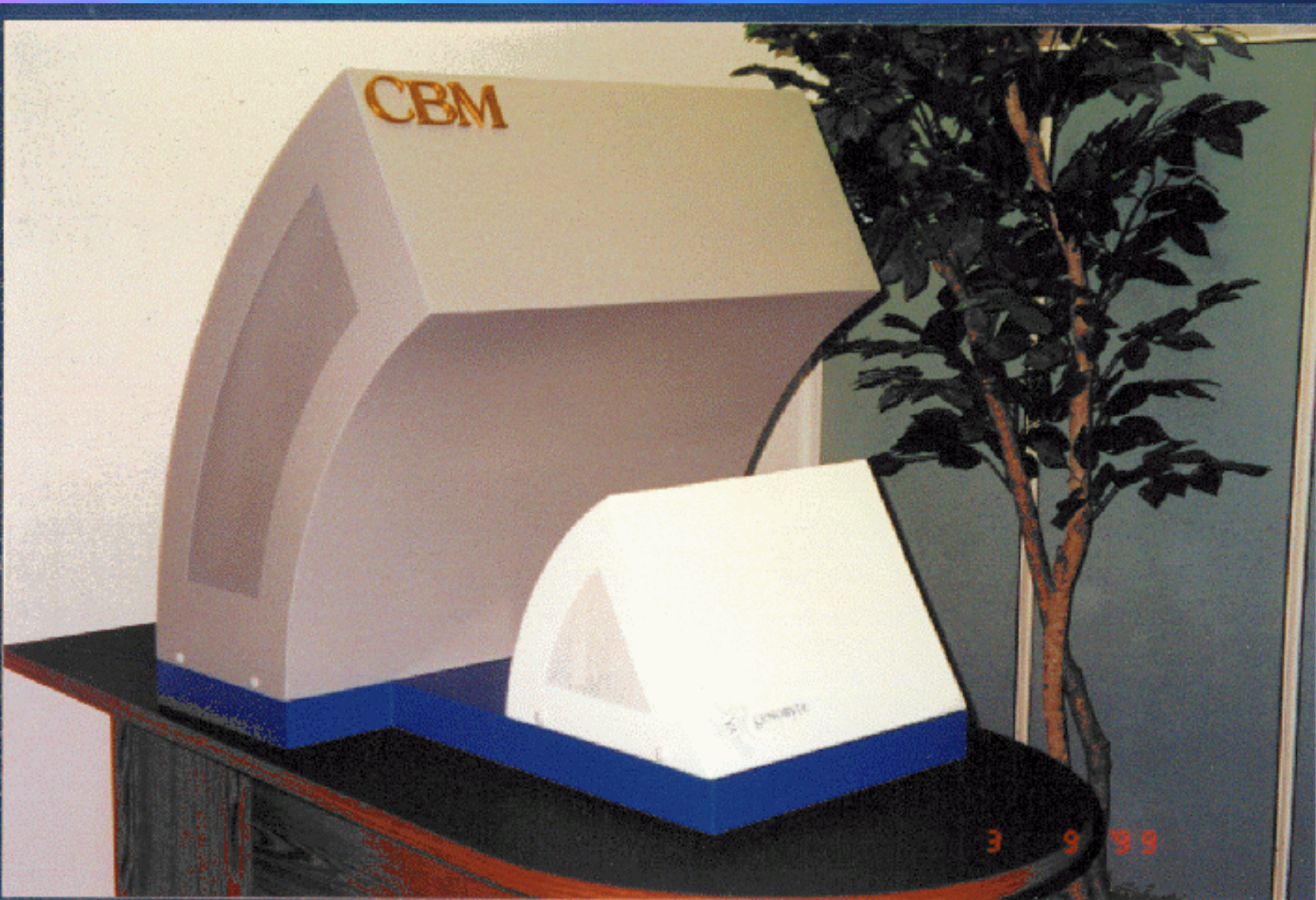
## ■ Summary of CBM Technical Specifications:

- Cellular Automata Update Rate (max.): 152 billion cells/second.
- Cellular Automata Update Rate (min.): 114 billion cells/second.
- Number of Supported Cellular Automata Cells (max.): 453 million
- Number of Supported Neurons (max., per module): 1,152.
- Number of Supported Neural Modules: 32,768.
- Number of Supported Neurons (max., per brain): 37,748,736.
- Neural Module Chromosome Length: 91,008 bits.

- Information Flow Rate, Neuronal Level (max.): 12 Gbytes/s.
- Information Flow Rate, Dendrite Level (estimated average): 36 Gbytes/s.
- Information Flow Rate, Intermodular Level (max.): 400 Mbytes/s.
- Number of FPGAs: 72 (Xilinx XC6264BG560).
- Number of FPGA Reconfigurable Function Units: 1,179,648.
- Phenotype/Genotype Memory: 1.18 Gbytes.
- Power Consumption: 1.5 KWatt (5 V, 300 A).
- Computational power (estimated): 10,000 Pentium II 400 MHz computers.



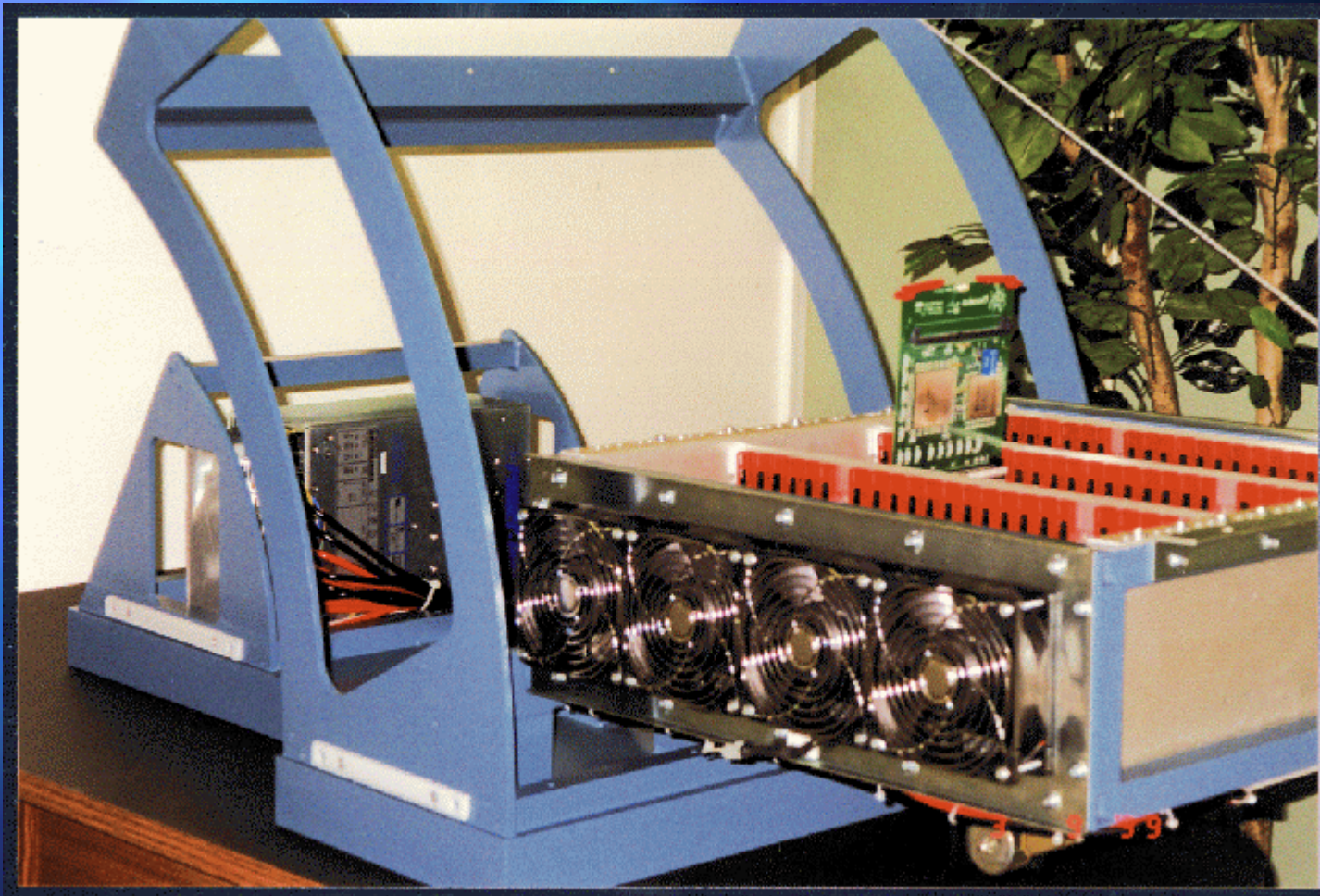
# Evolvable Neural Net Supercomputer of De Garis



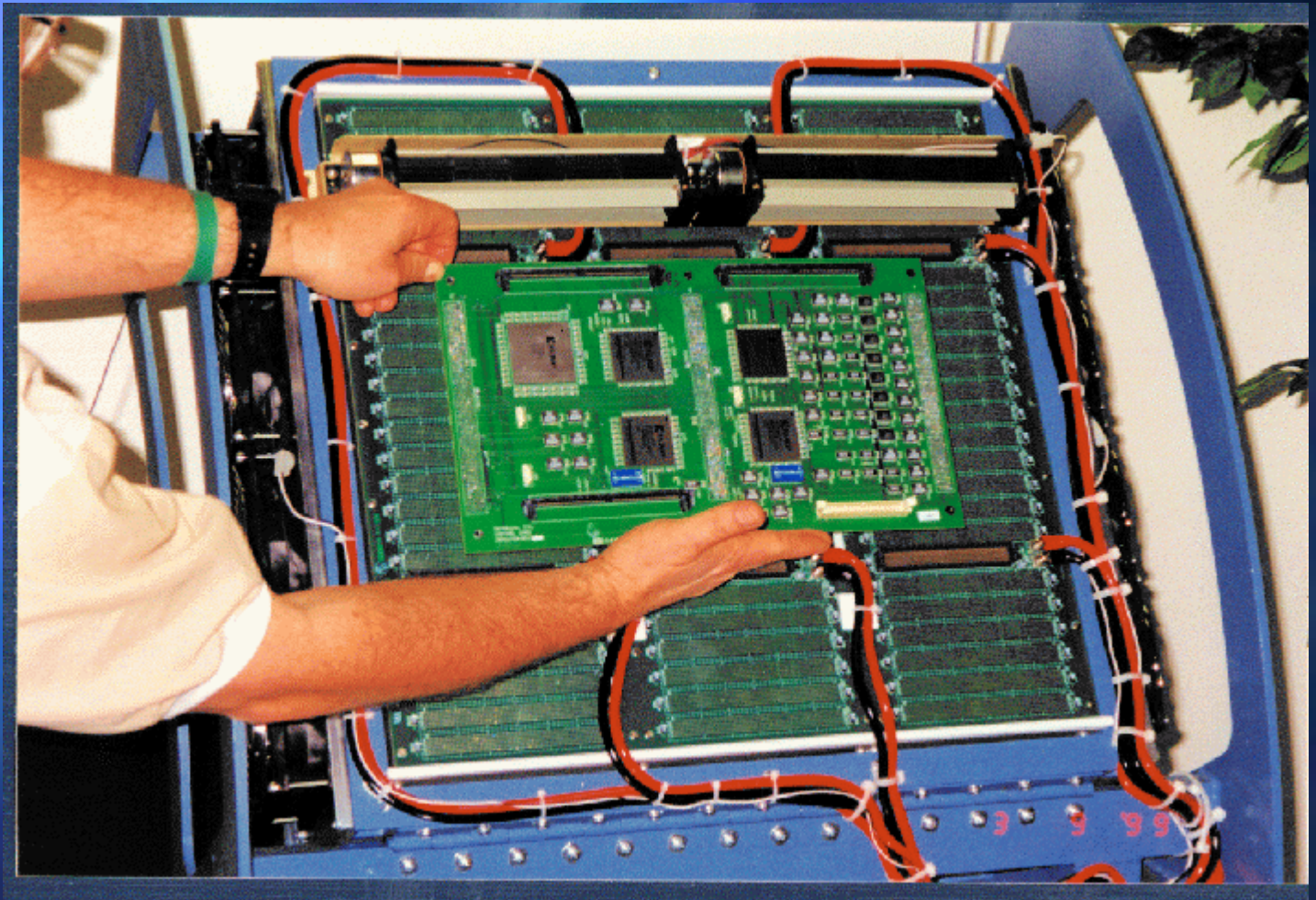












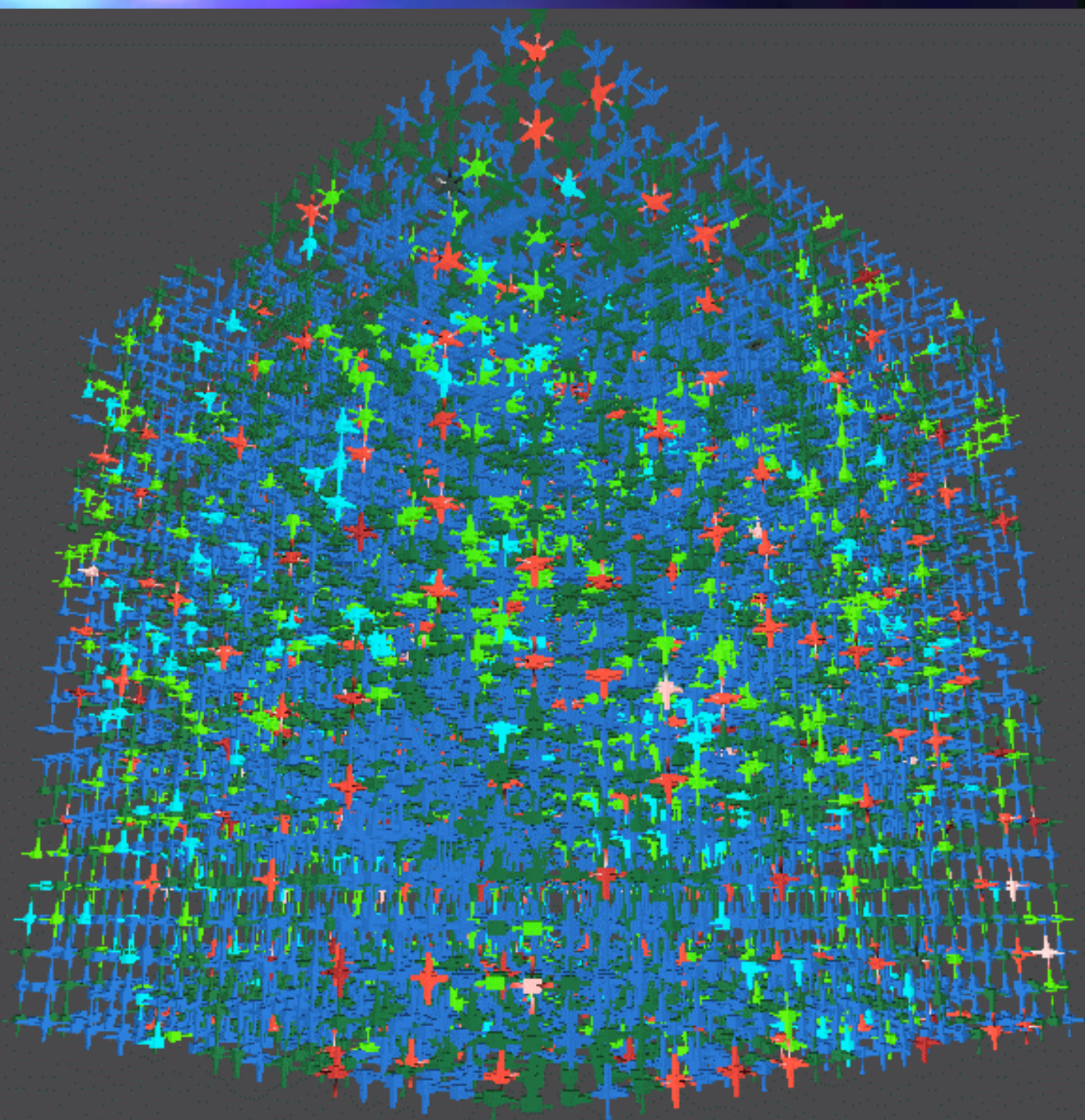
# CAM-Brain 3D Cube Circuit Module Images

- These three images show a CAM-Brain Module at different scales.
- A module is a  $24*24*24$  cube of 3D CA (cellular automata) cells, which contain up to about 1000 neurons (red).
- From these neurons grow axons (blue) and dendrites (green).
- Lighter blue and green cells contain a signal (1 bit).
- Whitened tapered connections to a neuron are excitatory neural inputs, i.e. the signal adds to the neurons' binary counter.

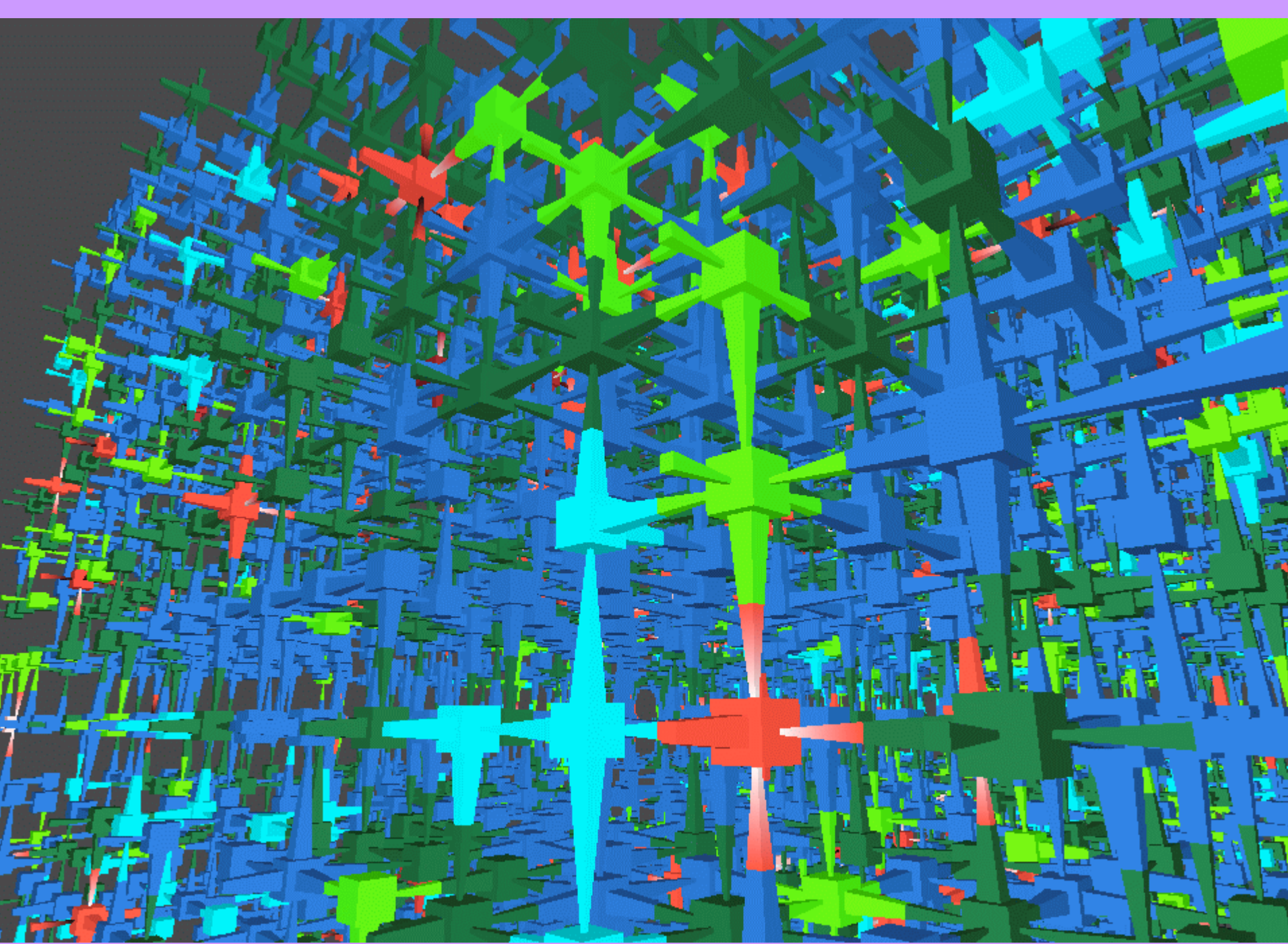


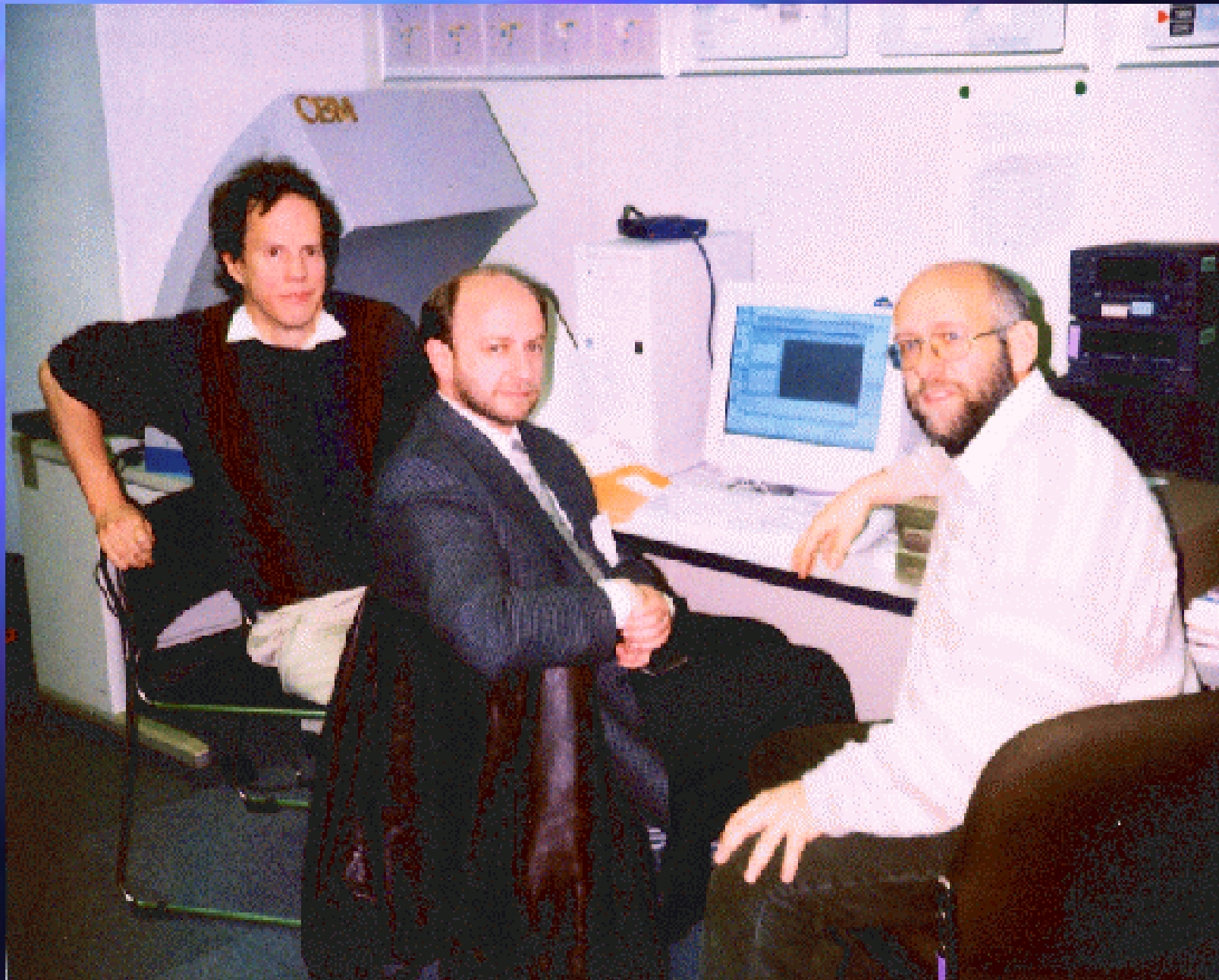
# CAM-Brain 3D Cube Circuit Module Images

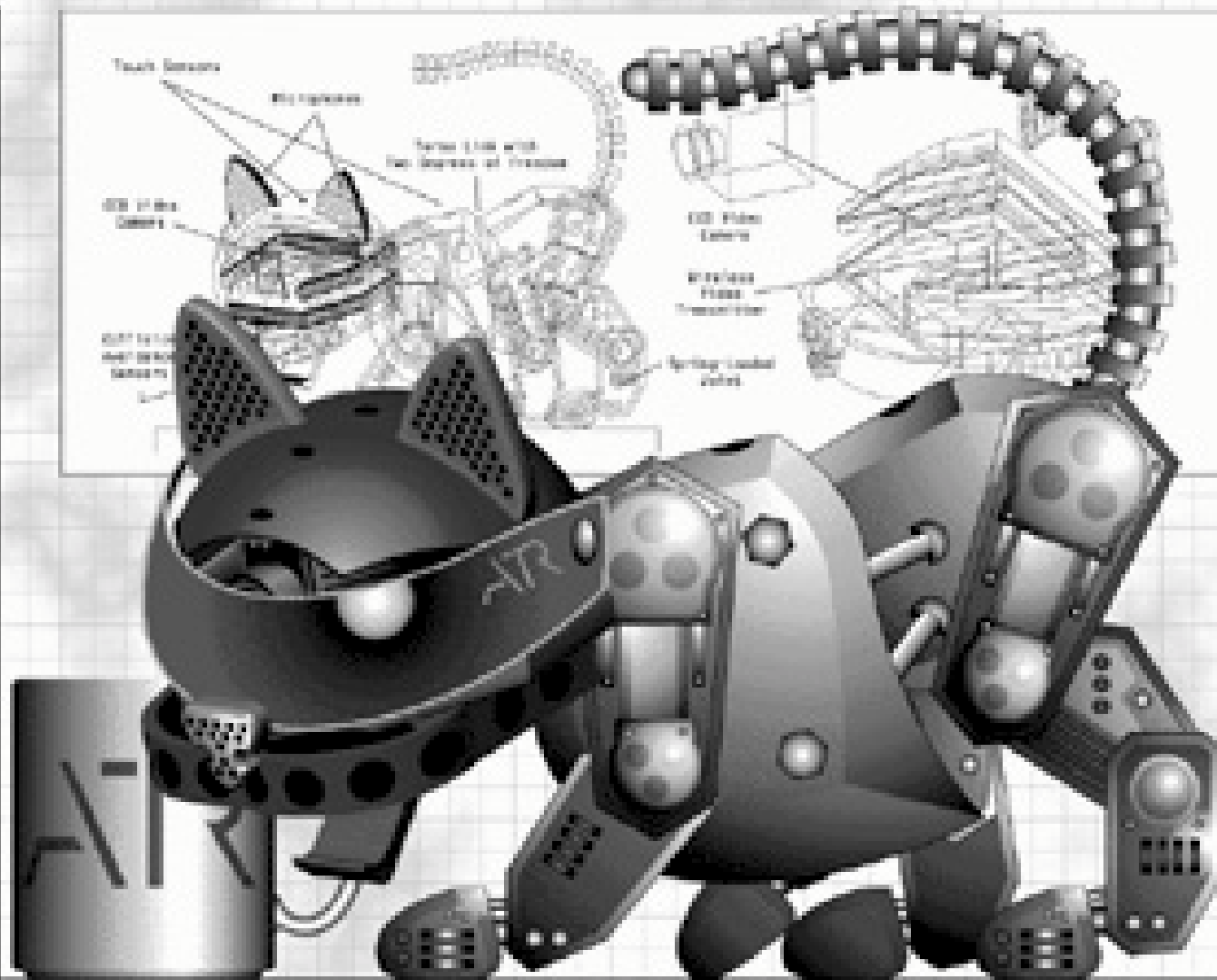
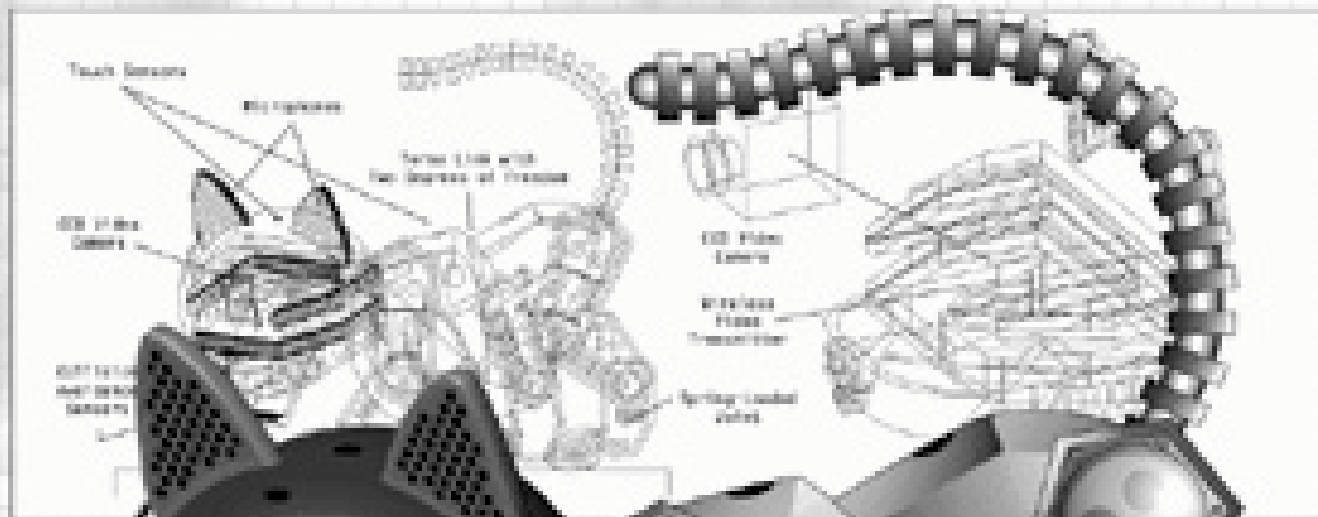
- Darkened tapered connections to a neuron are inhibitory neural inputs, i.e. the signal subtracts from the neuron's binary counter.
- If the counter value goes over a threshold value (usually 2), the neuron fires a binary signal into its axons.
  - Cube1 24\*24\*24 3D CA cells in a CAM-Brain circuit module
  - Cube2 24\*24\*24 3D CA cells in a CAM-Brain circuit module, zoomed
  - Cube3 24\*24\*24 3D CA cells in a CAM-Brain circuit module, zoomed again












ROBOKONEKO

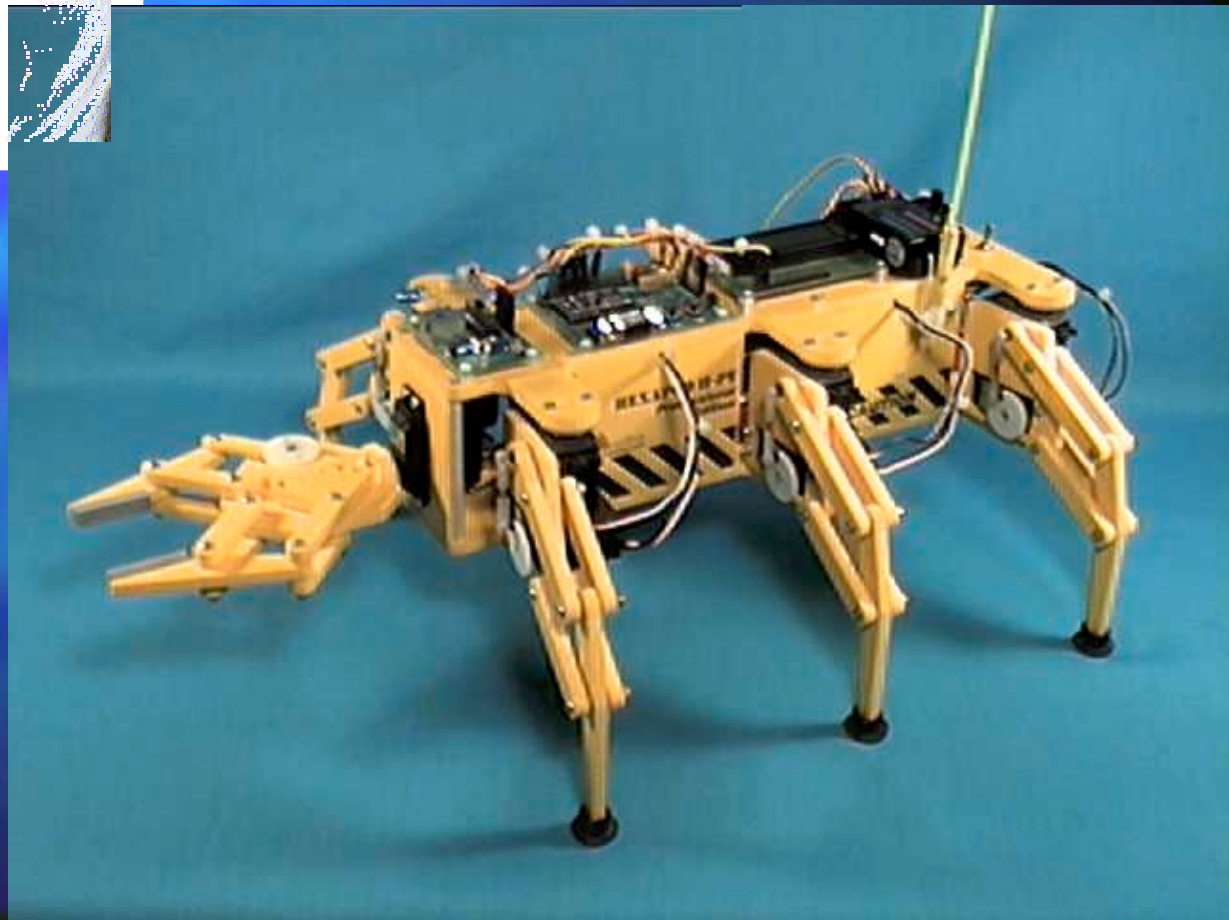
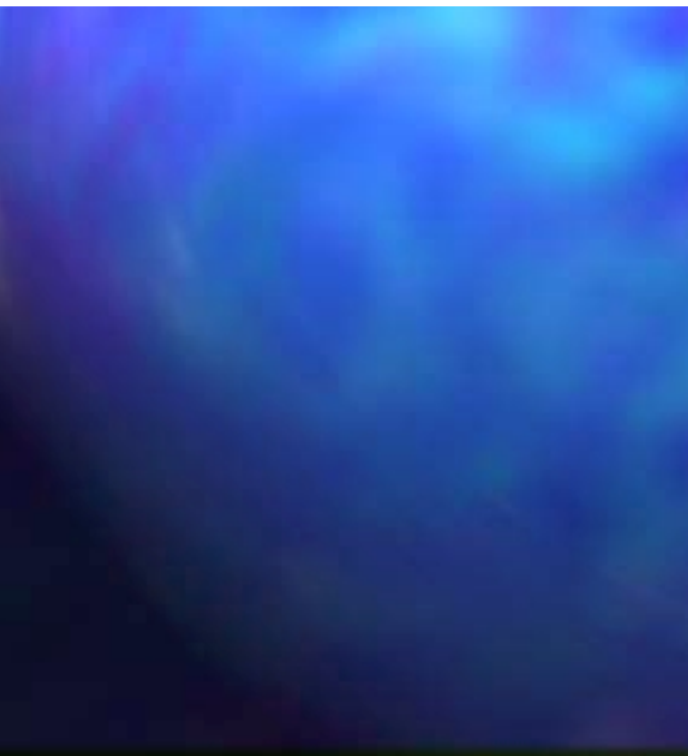




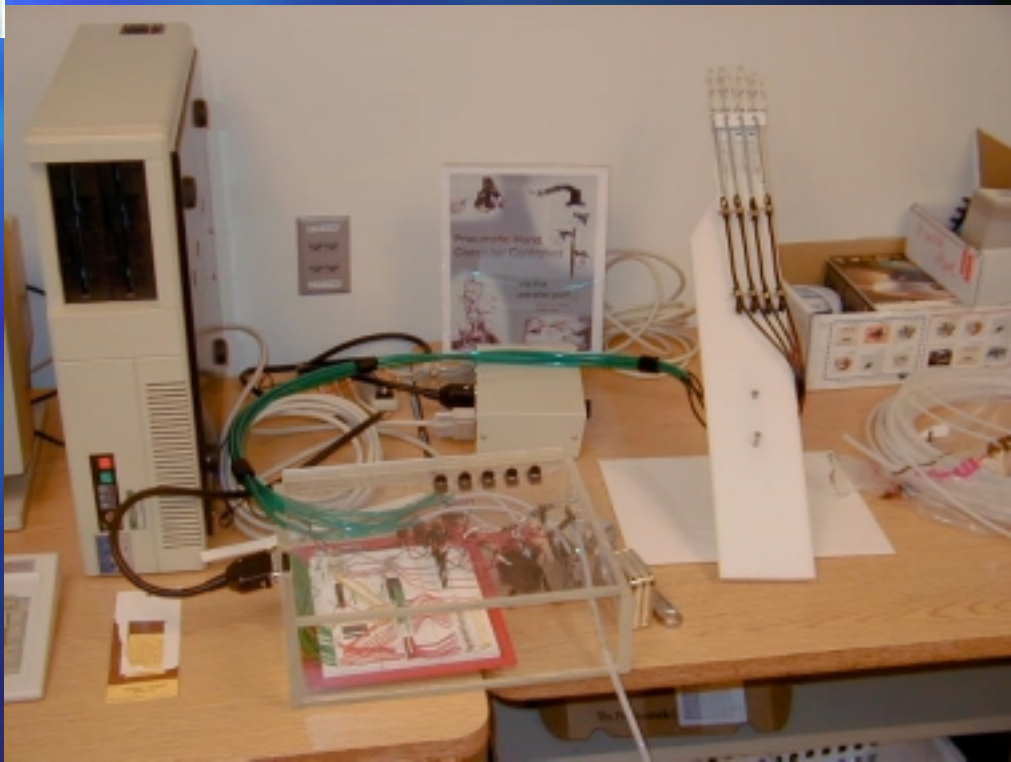
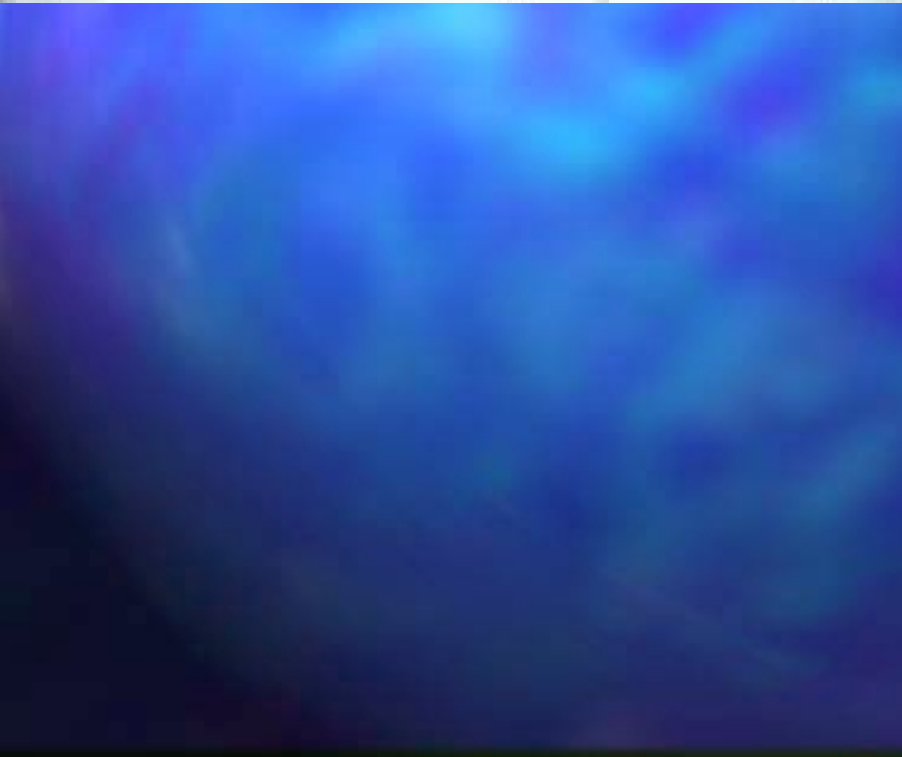
# **Approach of PSU**

... and what  
about PSU?









# Evolving or Learning in Hardware?

Machine Learning becomes a new and most general system design paradigm

It starts to become a new hardware construction paradigm as well

Evolvable Hardware is Genetic Algorithm PLUS reconfigurable hardware

We propose **Learning Hardware** as any learning algorithm PLUS reconfigurable hardware

Learning algorithm can be realized in software or in hardware.

# Universal Logic Machine

- Synthesis and Decision problems reduced to NP-hard combinational problems
- Combinational problems reduced to simple combinational problems such as graph coloring, set covering, bipartite covering, clique partitioning, satisfiability or multi-valued relation/function manipulation
- Cube Calculus Machine (CCM) operates on multiple-valued cubes (terms of MV literals).
- First variant uses two FPGA 3090 chips and second the DEC-PERLE-1 board with 23 chips
- General Special-Purpose computer for Cube Calculus



# Universal Logic Machine

- Phase of learning (construction, synthesis)
- Phase of acting (function evaluation, state machine operation)
- You cannot redesign standard computer hardware when it cannot solve the problem correctly.
- The Learning Hardware redesigns itself using new learning examples given to it
- Michie makes distinction between black-box and knowledge-oriented learning systems
- Concepts of "weak" and "strong" criteria
- "The system satisfies a weak criterium if it uses data to generate an updated basis for improved performance on subsequent data" (Neural, Genetic)

# What is most important in robotics?

- Speed of processing data in real time
- Understanding what is going on, rather than using black boxes
- Building model of the world around

*Our algorithms proved to be useful and give very good quality solutions in*

*--- FPGA synthesis*

*--- Data Mining*

*So now let us try the challenge of **ROBOTICS***

# Why we do not like the Genetic Algorithm?

- ➔ Our criticism is about using GA for digital circuits
- ➔ No explanation is given by a GA
- ➔ It is very slow comparing to any EDA tool
- ➔ It makes no use of human knowledge and years of accumulated engineering/research experience

Practically, it is not convergent

It is difficult to control

We do not know of any “real” success story of using GA in digital design



# Universal Logic Machine

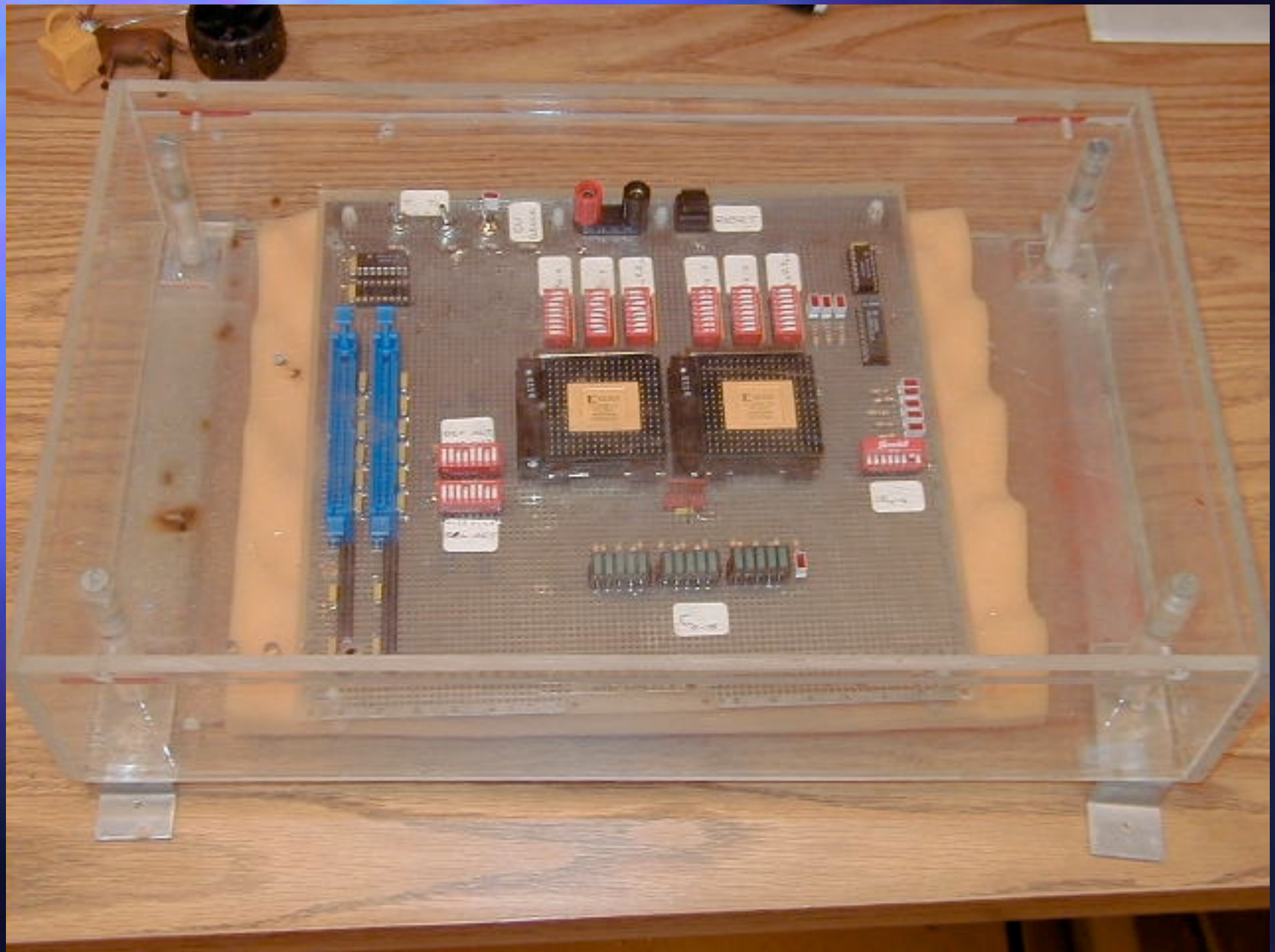
## **Synthesis and Decision problems reduced to NP-hard combinational problems**

→ Combinational problems reduced to simple combinational problems such as graph coloring, set covering, bipartite covering, clique partitioning, satisfiability or multi-valued relation/function manipulation

Cube Calculus Machine (CCM) operates on multiple-valued cubes (terms of MV literals).

**First variant** uses two FPGA 3090 chips and **second** the DEC-PERLE-1 board with 23 chips

General Special-Purpose computer for Cube Calculus







**DECstation  
and the  
DEC PERLE  
1 board**



# Universal Logic Machine

→ The high quality of decompositional techniques in Machine Learning, Data Mining and Knowledge Discovery areas was demonstrated by several authors; Ross (Wright Labs), Bohanec, Bratko/Zupan, Perkowski/Grygiel, Perkowski/Luba/Sadowska, Jozwiak, Luba, Goldman, Axtel.

→ Small learning errors. Natural problem representation

We compared the same problems using several methods: decomposition, decision trees, neural nets, and genetic algorithms

Decomposition is clearly the winner but it is slow because the NP-complete problem of graph creation and coloring is repeated very many times.

# *Example of Constructive Induction*

Decomposition of Relations

# Who are the good guys?



John



Mark



Dave



Jim

Good guys



Alan



Mate



Nick



Robert

Bad guys

Logic Synthesis for Machine Learning



# Who are the good guys?



John



Mark



Dave



Jim

Good guys



Alan



Mate



Nick



Robert

Bad guys

A - size of hair

B - size of nose

C - size of beard

D - color of eyes



John



Mark



Dave



Jim

Good guys

$A' BCD$

$A' BCD'$

$A' B'CD$

$A' B'CD$

		CD			
		00	01	11	10
AB	00	-	-	1	-
	01	-	-	1	1
	11	-	-	-	-
	10	-	-	-	-

A - size of hair

B - size of nose

C - size of beard

D - color of eyes



Alan



Mate



Nick



Robert

Bad guys

$A' BC'D'$

$AB'C'D$

$ABCD$

$A' B'C'D$

A - size of hair

B - size of nose

C - size of beard

D - color of eyes

	CD			
	00	01	11	10
AB 00	-	-	1	-
01	0	0	1	1
11	-	-	0	-
10	-	0	-	-



$A'C$



# Generalization 1:

Bald guys with beards are good

# Generalization 2:

All other guys are no good

	CD			
	00	01	11	10
AB				
00	-	-	1	-
01	0	0	1	1
11	-	-	0	-
10	-	0	-	-

A - size of hair

B - size of nose

C - size of beard

D - color of eyes

A'C



# **Induction of Logic**

**Formulas from  
Examples**

# Induction of Logic Formulas from Examples

- Multi-Valued multi-output (combinational) relation in tabular form

Record from data base

	X1	X2	Y1	Y2
a	0,2	1	-	2
b	0,1	0	0,2	1
c	2	0	1,2	0
d	1	1	1,2	2

Input 1      input2      output1      output2



# Induction of State Machines from Temporal Logic Constraints

- From the previous table Y1 denotes a relation output. X1 and X2 together with output Y1 specify an (oriented) relation. Relation can be used to express facts as :- this color is red or white but not yellow or black. Symbol Y2 denotes a function output;  $Y2(X1, X2)$ . Rows c and d have only one value for each attribute, so they are minterms.
- Rows a and b have more than one value for attributes, so they are cubes. Each row can be thought of as a record from a data base, or their set, or a collection of image features after image preprocessing.
- Observe, however that although such language is quite powerful for ML, DM, and KD from data bases applications, it cannot specify time, it is thus too poor to describe state machines, regular expressions, petri nets, path expressions, sequential netlists, grammars and other models, that are used in speech

# **Induction of State Machines from Temporal Logic Constraints**

# Example:

- Now, we have a data-base of good girlfriends and bad girlfriends that can be described by their four attributes.
- We should see how learning in Hardware is much like logic-design (using K-maps) and generalization is similar to taking advantage of “Don’t Cares”.
- Now let’s look at our example data base....



# The Characteristic Key:

Before we introduce our data-base of good and bad girlfriends, we first need to encode their characteristics as states before. Each girl is given a **4 bit code**. The first bit is her **Eye Color** (0=dark, 1=light), the second is **Hair Length** (0=short, 1=long), the third is **Height** (0=short, 1=tall), and the fourth is **Weight** (0=skinny, 1=healthy). So, for example:

1111

Would represent a girl with light eyes, long hair, tall, and of a healthy weight. Now we can finally go to our data base example:

# Girlfriend Database:

First, the sample “GOOD” girlfriends:



0000



1011



0001



0010

And lastly, our data-base of “BAD” girlfriends



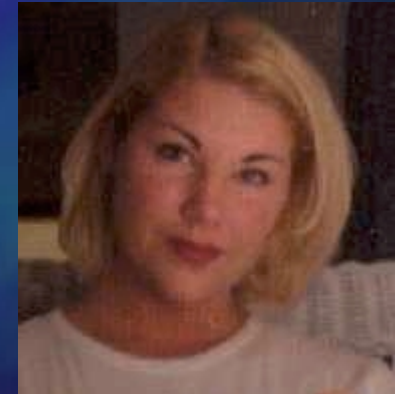
0100



1100



0110



1100

Of course, the pictures have absolutely no correlation to my previous girlfriends that were “bad”.

Next, we want to learn from this data. So, let’s construct a K-Map with output being “GOOD=1” and “BAD=0”. And the inputs are the four input bits that describe a prospective girlfriend.





So, in our example we obtained an expression for the GOOD girls as though we were performing logic minimization with a K-Map.

At the same time, we took advantage of the Don't Cares to generalize and simplify our logic.

So what did our data-base reveal? The output function or “Rule” for a GOOD girlfriend was:

$$\text{GOOD} = \text{not}(b), \quad \text{Where } b = \text{Hair Length}$$

So, good girlfriends are ones with short hair. And therefore, BAD girlfriends can be generalized to **not(GOOD)** or **BAD = b**, or girls with long hair.

Now your robot can tell you if your date tonight is good or bad for you :).

- This was a very simple example, but the steps were somewhat similar.
- We see that symbolic representation and optimization are natural and we have a solid foundation in symbolic theory from many years of human invention. -->

- This is another powerful point of Learning Hardware using Logic Synthesis (or Constructive Induction).
- We have at our disposal years of engineering experience to benefit from.
- This differs greatly with evolvable techniques in which a decent solution is found but the resulting network can be chaotic, complex, redundant, and perplexing.
- It is this random behavior that interests some people, but for Hardware Learning, we are not interested in evolving a circuit that uses 100 gates to perform the simple operation of  $A \text{ EXOR } B$ .



# Relation



# Function



```
.type mv          0 1 0 0 - 1 1 - - 1
.i 9              1 0 0 0 - 1 1 - - 1
.o 1              0 0 - 0 1 1 1 - - 1
.ilb i0 i1 i2 i3 i4 i5 i6 i7
    i8            0 1 - 0 1 0 1 - - 1
    i8            1 1 0 - 0 1 0 - - 1
.ob o0            1 0 1 - 1 0 0 - - 1
.imv 2 2 2 2 2 2 2 2 2 2 0 1 - 1 1 0 0 - - 1
.omv 2            0 1 0 1 - 0 - 1 - 1
.p 156           - 0 1 1 0 0 - 1 - 1
0 0 0 1 1 - 1 - - 1
0 1 1 0 0 - 1 - - 1
1 0 1 0 0 - 1 - - 1
0 1 0 0 - 1 1 - - 1
1 0 0 0 - 1 1 - - 1
```

```
0 1 0 0 - 1 1 - - 1
1 0 0 0 - 1 1 - - 1
0 0 - 0 1 1 1 - - 1
0 1 - 0 1 0 1 - - 1
1 1 0 - 0 1 0 - - 1
1 0 1 - 1 0 0 - - 1
0 1 0 1 - 0 - 1 - 1
- 0 1 1 0 0 - 1 - 1
0 0 - - 0 1 1 1 - 1
- - 0 0 1 0 1 1 - 1
0 1 - - 0 0 1 1 - 1
.end
```

```
.type mv
.i 3
.o 1
.ilb i0 i1 i2
.ob s2.0
.imv 2 2 2
.omv 4
.p 8
0 1 0 1
0 1 1 0
1 1 0 0
1 1 1 2
1 0 0 1
1 0 0 1
1 0 1 0
.end
```

```
.type mv
.i 3
.o 1
.ilb i3 i4 i5
.ob s2.1
.imv 2 2 2
.omv 4
.p 8
0 1 0 1
1 1 0 0
1 1 1 2
1 0 0 1
1 0 1 0
.end
```

```
.type mv
.i 3
.o 1
.ilb i3 i4 i5
.ob s2.1
.imv 2 2 2
.omv 4
.p 8
0 1 0 1
1 1 0 0
1 1 1 2
1 0 0 1
1 0 1 0
.end
```

# Lattice diagrams realize regular layout of logic gates

LOTUS - [Lotus]

File Edit View Examples Window Help

Variable order: a(S), b(S), c(S), d(S), e(S).

```

.i 5
.o 1
.p 32
00001 1
00011 1
00101 1
00111 1
01001 1
01010 -
01011 1
01100 1
    
```

$F1 = abce + abc'd'e$

$F0 = abd'e' + abc'd + ab'cd$

c	0	0	0	0	1	1	1	1
d	0	0	1	1	1	1	0	0
e	0	1	1	0	0	1	1	0
ab								
00	0	1	1	0	0	1	1	0
01	0	1	1	-	-	1	0	1
11	0	1	0	0	-	1	1	0
10	0	1	-	0	0	0	1	0

ON Set

9 nodes

OFF Set

11 nodes

Read Espresso file

# Lotus: Layout-Driven Logic Synthesis for submicron technologies

- Lattice Diagrams - Generalization of Binary Decision Diagrams and Kronecker Diagrams
- New data structure for layout minimization
- Optimization of area, speed and power
- Predicts results of many levels of synthesis
- Ideal for future technologies





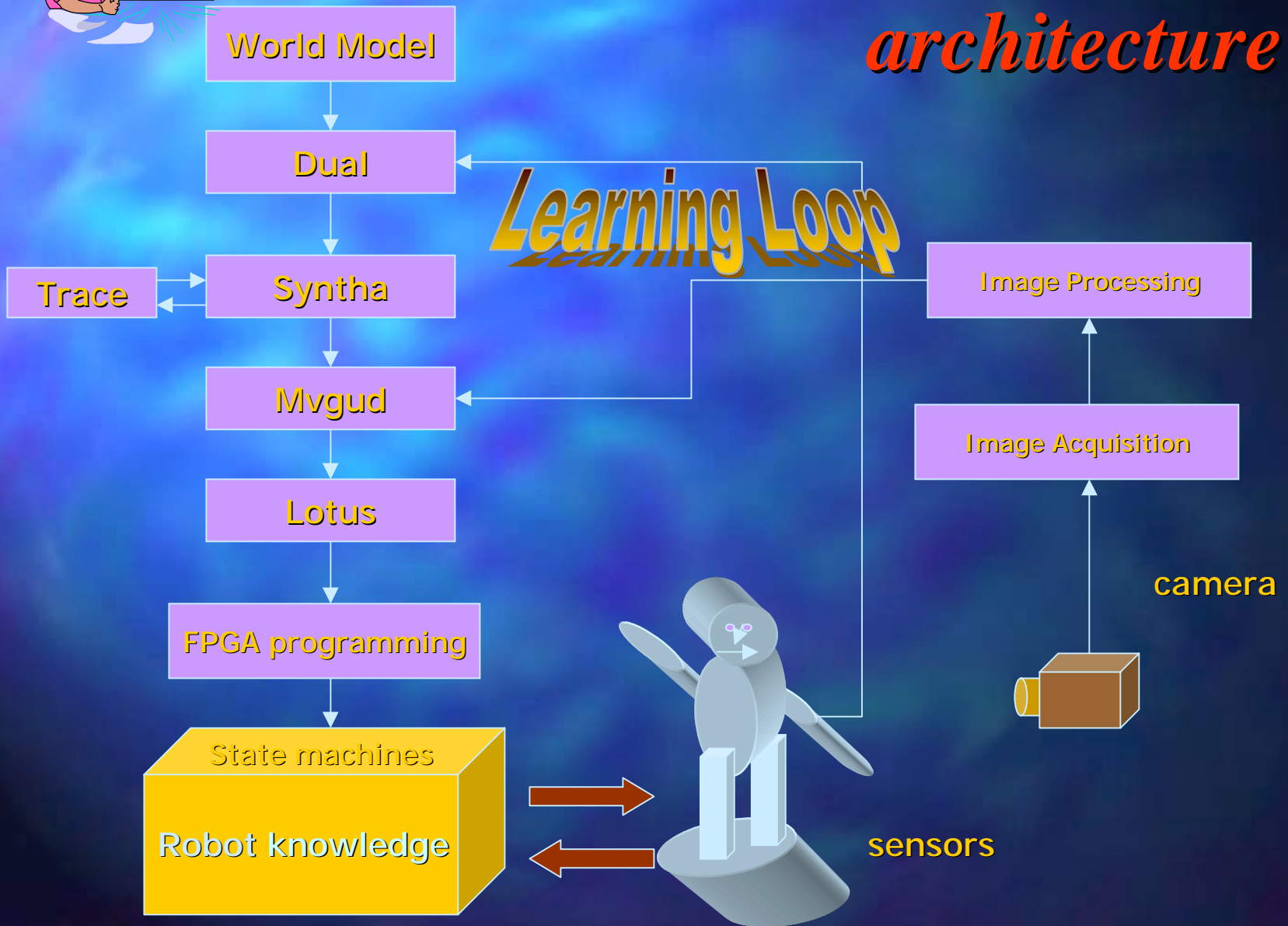
*Our  
Hardware  
System*

Faster!!

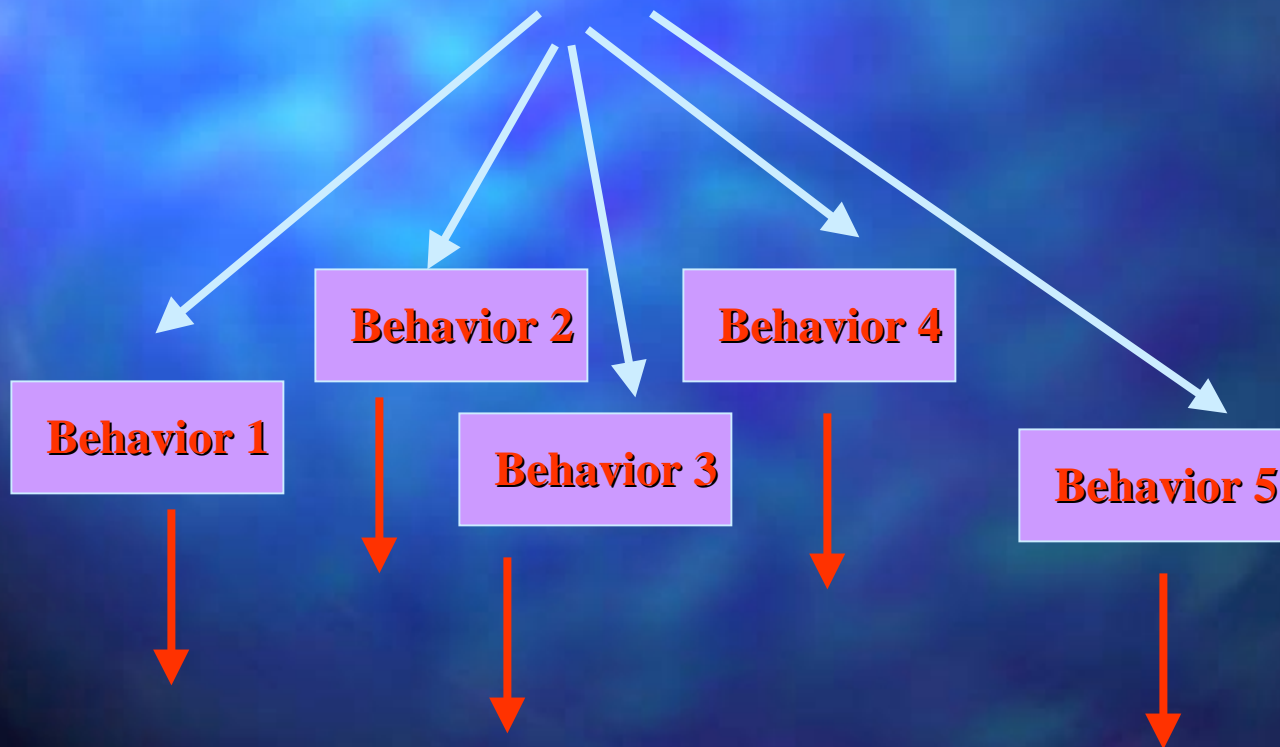


Human teacher

# MUVAL architecture

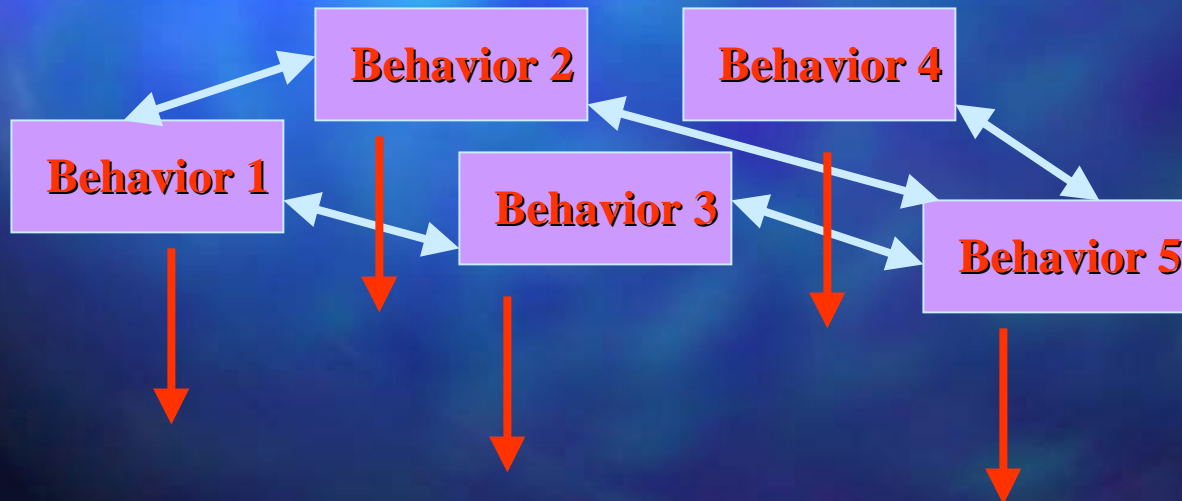


# Software Architecture 1

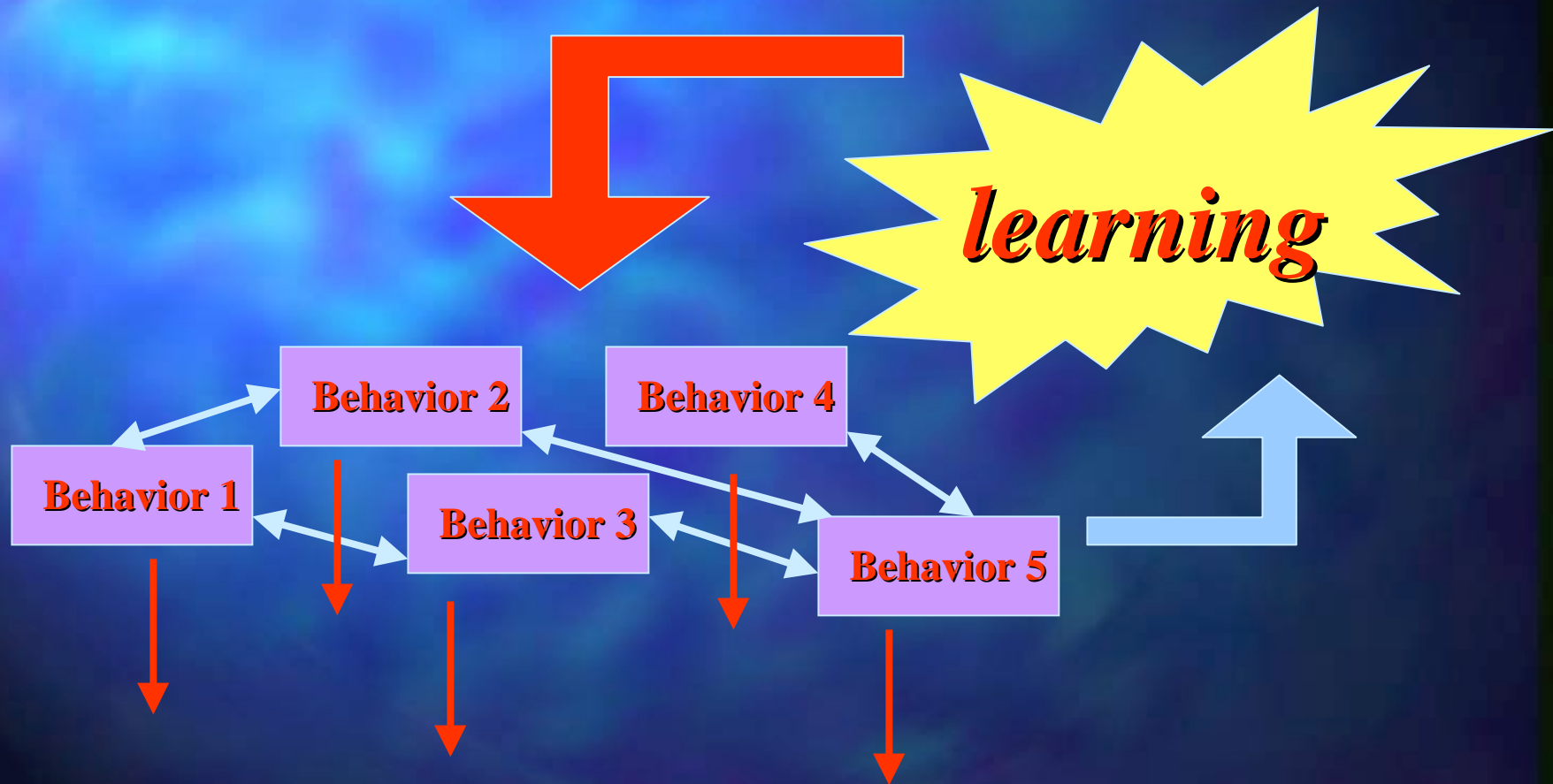




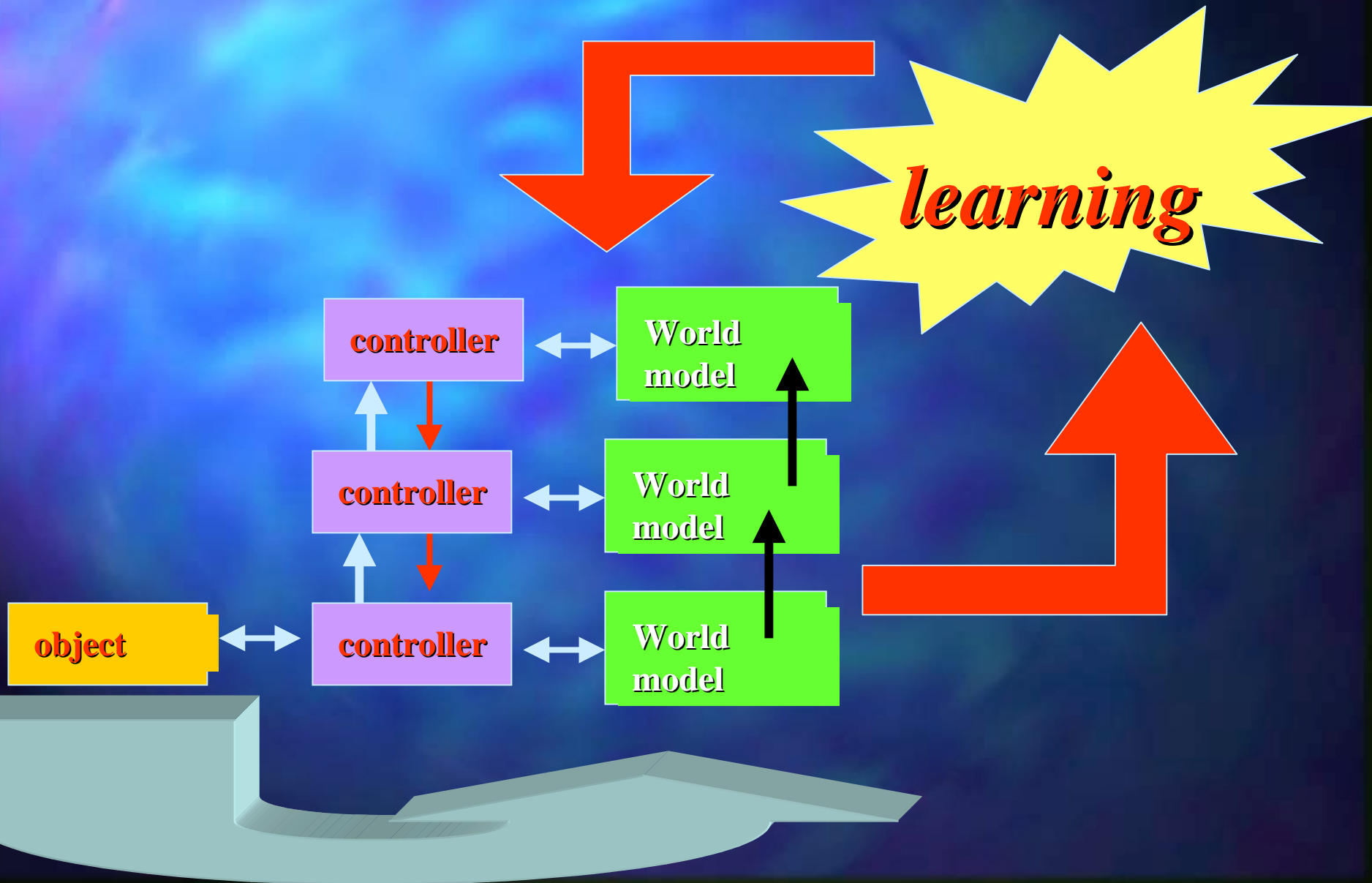
# Software Architecture 2



# Software Architecture 3



# Software Architecture 4





# Experimental Results

- Speedup on 3 variables is 0.72, 4 variables - 0.72, 5 variables - 0.8

Frequency of FPGA 3090 was 4MHz

Frequency of Sun Ultra was 270MHz

If we map the entire CCM into one chip delay would be reduced

New chips are faster and denser.

The delay of CLB of 3090 is 4.5 nS, the delay for CLB of 4085XL is 1.2 nS.

4085 has array  $56 * 56$  and 448 user I/O pins.

# Experimental Results

- We can map entire CCM into **one** 4085

Clock of 4085 is **20** MHz

Clock of CCM is five times slower than Sun

CCM will run **4 times faster** than software approach

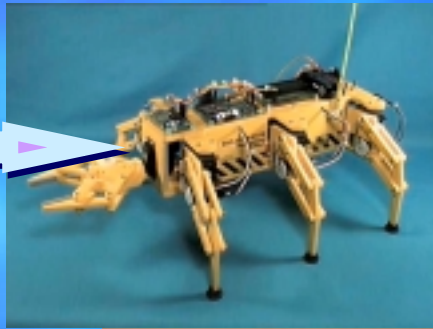
# Undergraduate Projects

**XILINX**

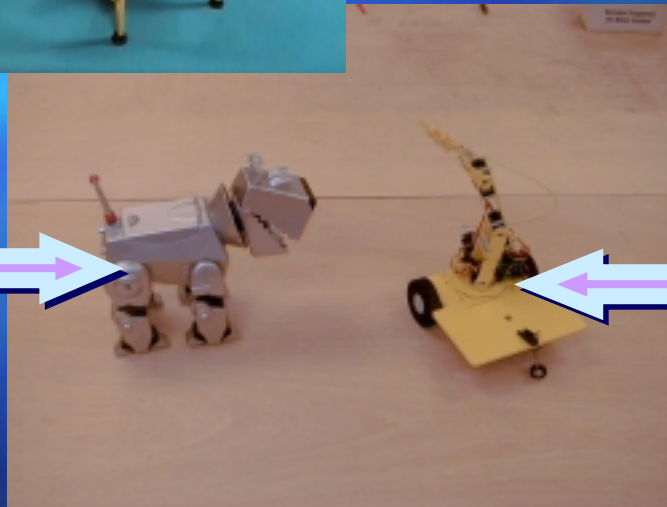
**ALTERA**

**CYPRESS**

EPLD



EPLD



FPGA

Group Learning behaviors



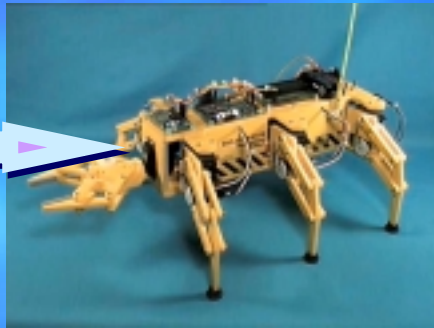
# Undergraduate Projects

**XILINX**

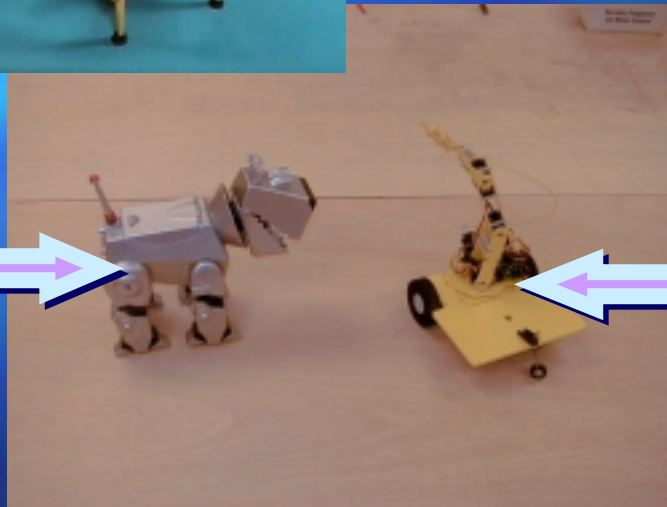
**ALTERA**

**CYPRESS**

**EPLD**

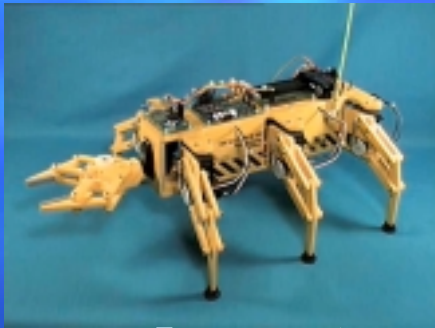


**EPLD**



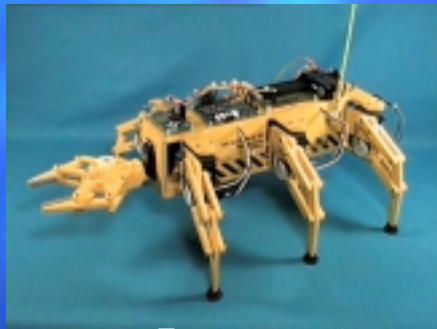
**FPGA**

# Spider I control - phase one



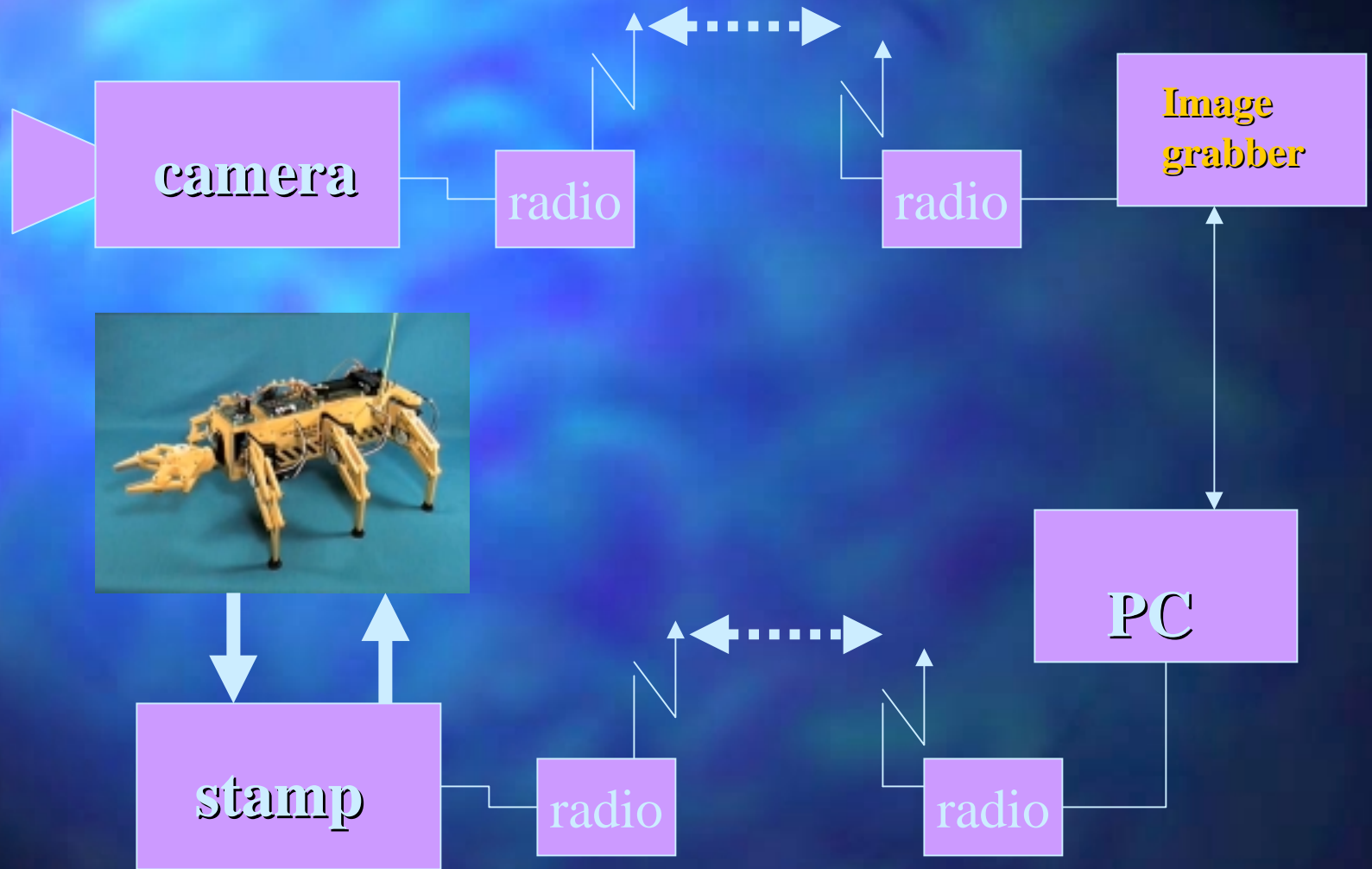
stamp

# Spider I control - phase two





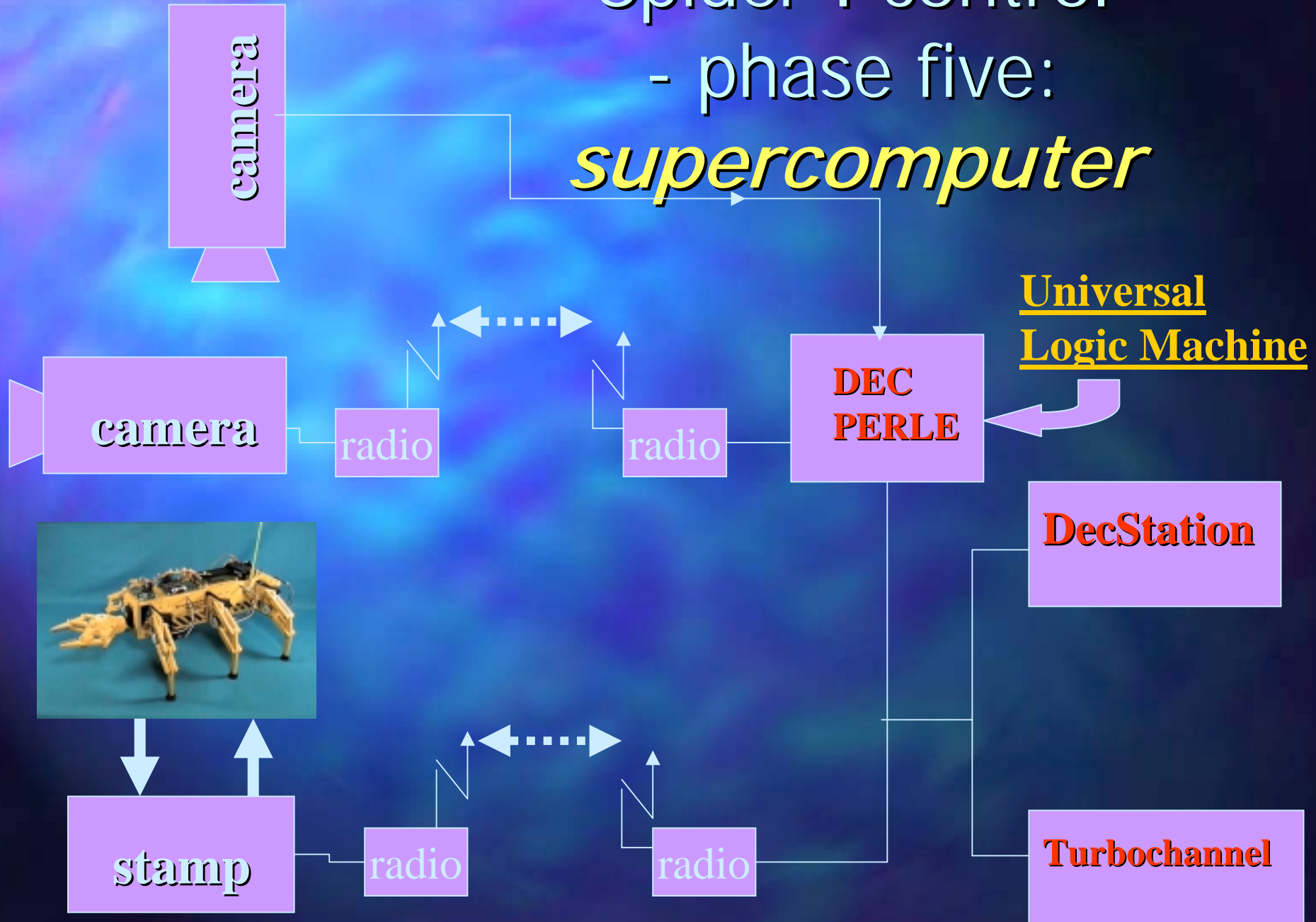
# Spider I control - phase three



# Spider I control

- phase five:

*supercomputer*



# Modes of Operation

- Programmed (theater, demos)
- Learning
- Interactive (interactive theater, society of robots, human-robot interaction)