Building a Nature-Inspired Computer

Peter J Bentley

p.bentley@cs.ucl.ac.uk

http://www.cs.ucl.ac.uk/staff/p.bentley/

UCL

1951

This is a **maze-solving machine** that is capable of **solving a maze** by trial-and-error means, of **remembering** the solution, and also of **forgetting** it in case the situation changes and the solution is no longer applicable.

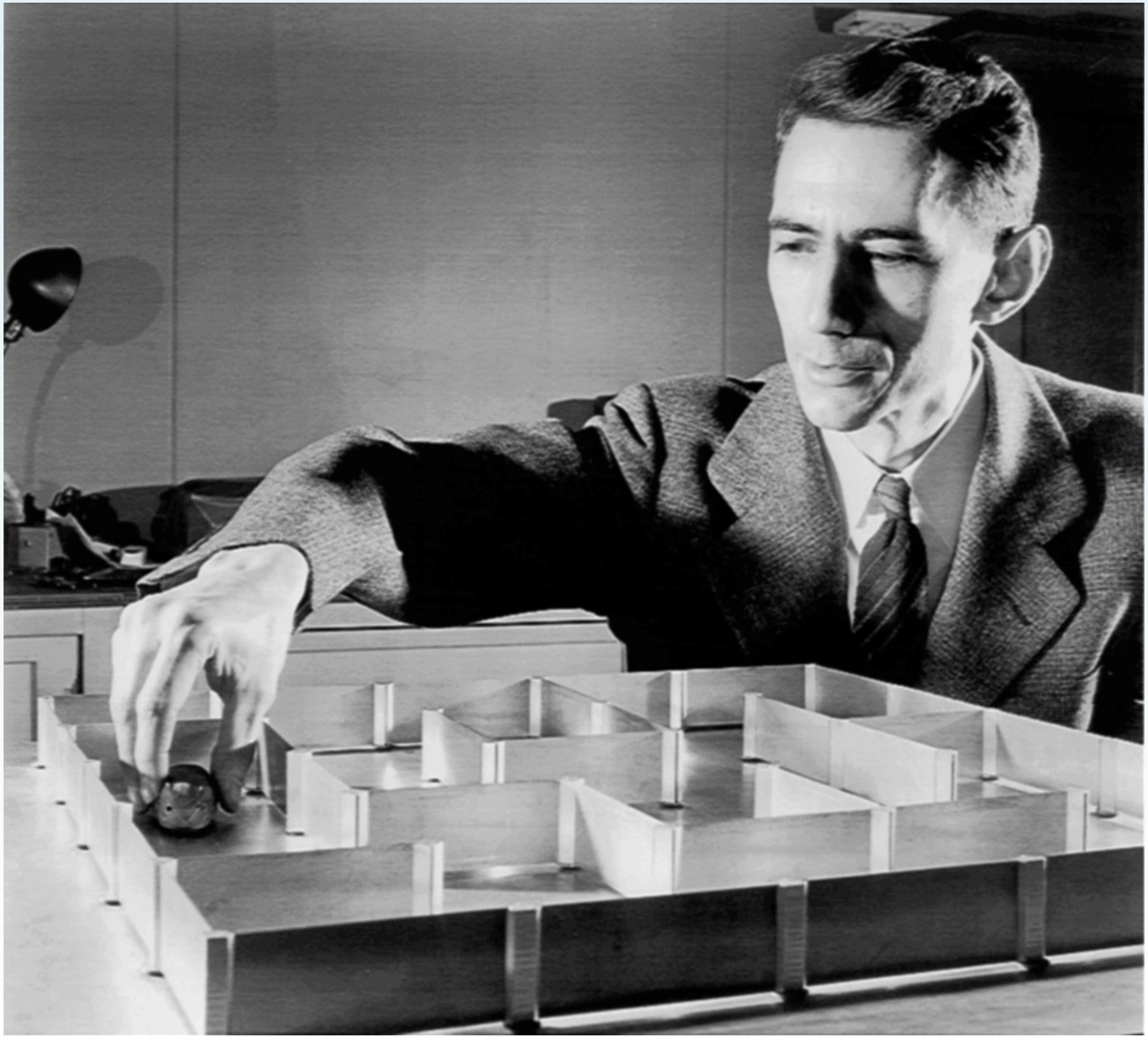
…Now I would like to show you one further feature of the machine. I will change the maze so that the solution the machine found no longer works. By moving the partitions in a suitable way, I can obtain a rather interesting effect. In the previous maze the proper solution starting from Square A led to Square B, then to C, and on to the goal. By changing the partitions I have forced the machine at Square C to go to a new square, Square D, and from there back to the original square, A. When it arrives at A, it remembers that the old solution said to go to B and so it goes around the circle A, B, C, D, A, B, C, D …. It has established a vicious circle, or a singing condition.

- A **neurosis**
- Yes
- It can't do that when its mind is blank, but can do it after it has been conditioned?
- Yes, only after it has been conditioned. However, the machine has an **antineurotic circuit** built in to proven just this sort of solution

-It doesn't have any way to recognise that it is **"psycho"** it just recognizes that it has been going too long?
-Yes. As you see, it has now gone back to the exploring strategy.

Presentation of a Maze-Solving Machine

Claude Shannon 1951



1950

I believe that at the end of the century the use of words and general educated opinion will have altered so much that **one will be able to speak of machines thinking without expecting to be contradicted.**

I believe further that no useful purpose is served by concealing these beliefs.

The popular view that scientists proceed inexorably from well- established fact to well-established fact, never being influenced by any unproved conjecture, is quite mistaken.

Provided it is made clear which are proved facts and which are conjectures, no harm can result. Conjectures are of great importance since they suggest useful lines of research.

Computing Machinery and Intelligence

A. M. Turing

1950

1945

In analyzing the functioning of the contemplated device, certain classificatory distinctions suggest themselves immediately.

First: Since the device is primarily a computer, it will have to perform the elementary operations of arithmetics most frequently. These are addition, multiplication and division.

It is therefore reasonable that it should contain **specialized organs** for just these operations... a central arithmetic part of the device will probably have to exist and this constitutes the first specific part: CA.

Second: The logical control of the device, that is the proper sequencing of its operations can be most efficiently carried out by a **central control organ**... this constitutes the second specific part: CC.

Third: Any device which is to carry out long and complicated sequences of operations (specifically of calculations) must have a considerable **memory**... this constitutes the third specific part: M.

...The three specific parts CA, CC and M correspond to the **associative neurons in the human nervous system**.

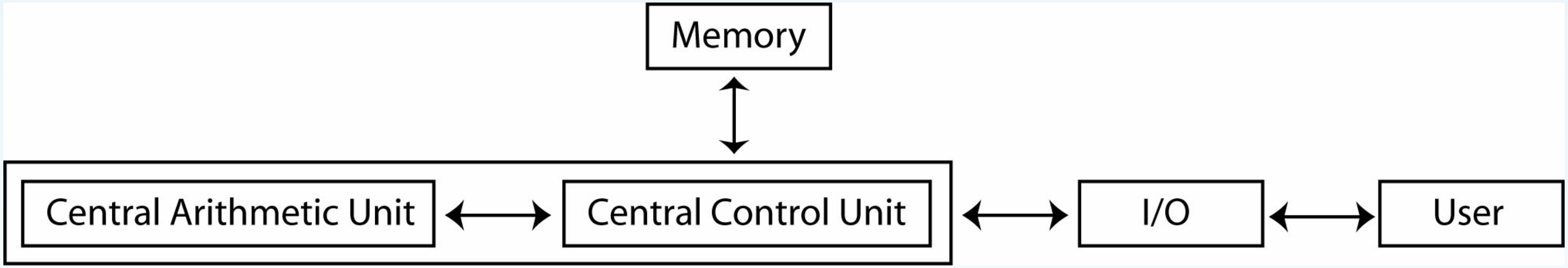
It remains to discuss the equivalents of the **sensory or afferent and the motor or efferent neurons**.

These are the **input and the output organs** of the device.

First Draft of
a Report on
the EDVAC

John von Neumann

1945

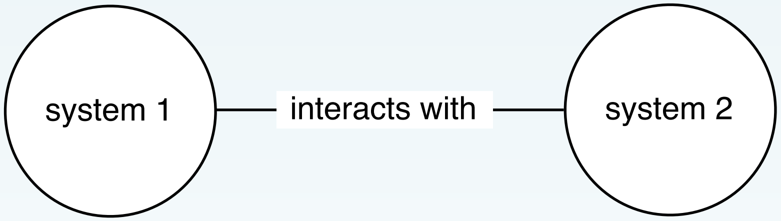

Memory
Central Arithmetic Unit
Central Control Unit
I/O
User

Conventional vs Natural Computation

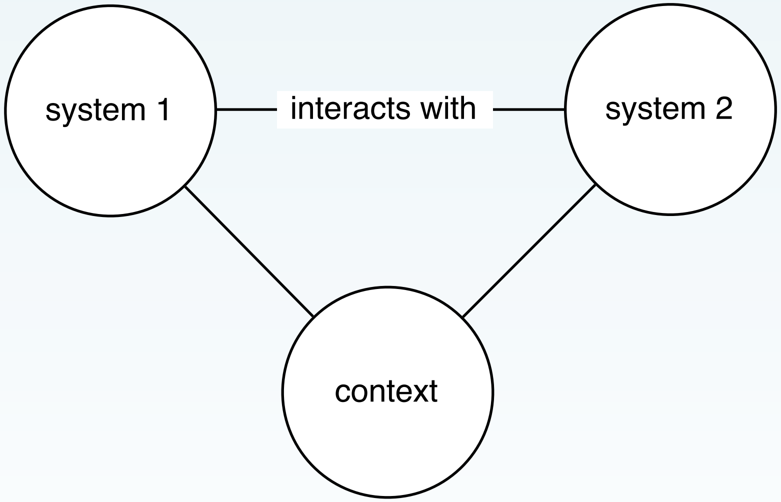
Conventional	Natural
Deterministic	Stochastic
Synchronous	Asynchronous
Serial	Parallel
Heterostatic	Homeostatic
Batch	Continuous
Brittle	Robust
Fault intolerant	Fault tolerant
Human-reliant	Autonomous
Limited	Open-ended
Centralised	Distributed
Precise	Approximate
Isolated	Embodied
Linear Causality	Circular Causality

How to build a Nature-Inspired Computer?

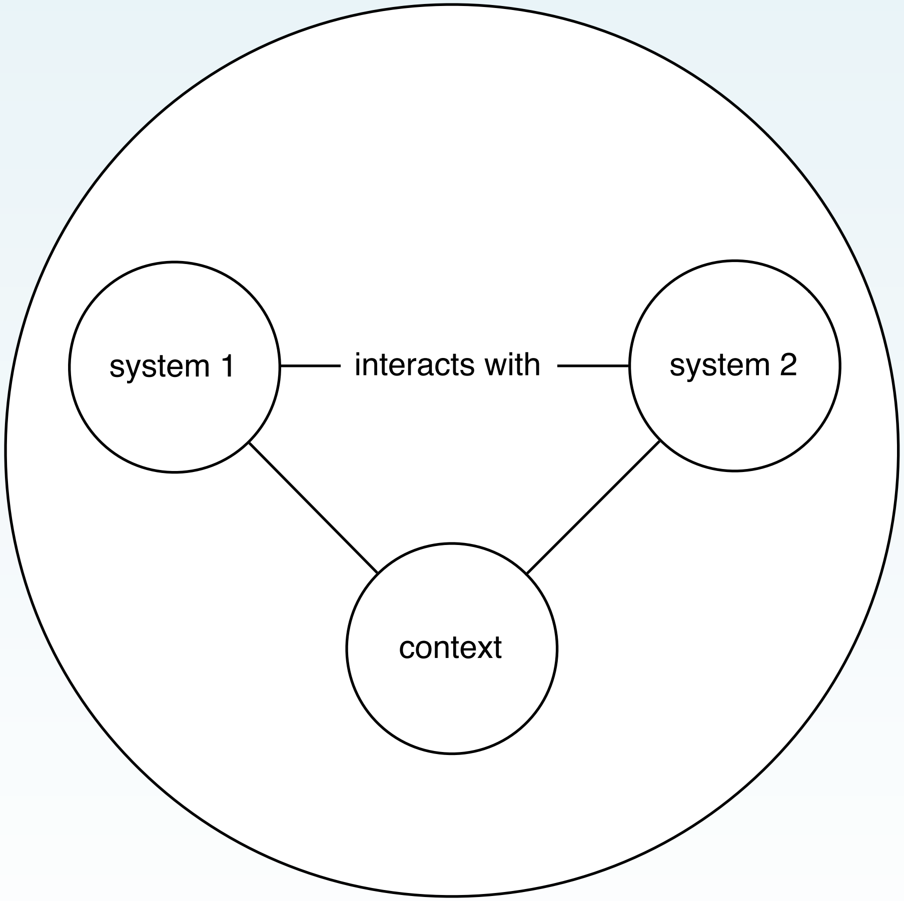
How to represent reality?



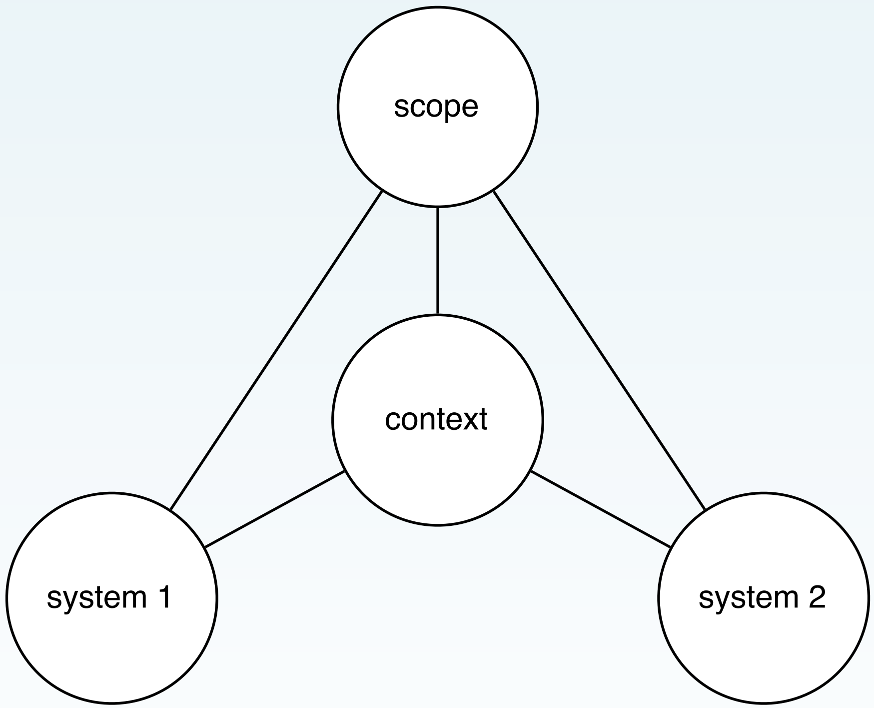
How to represent reality?



How to represent reality?

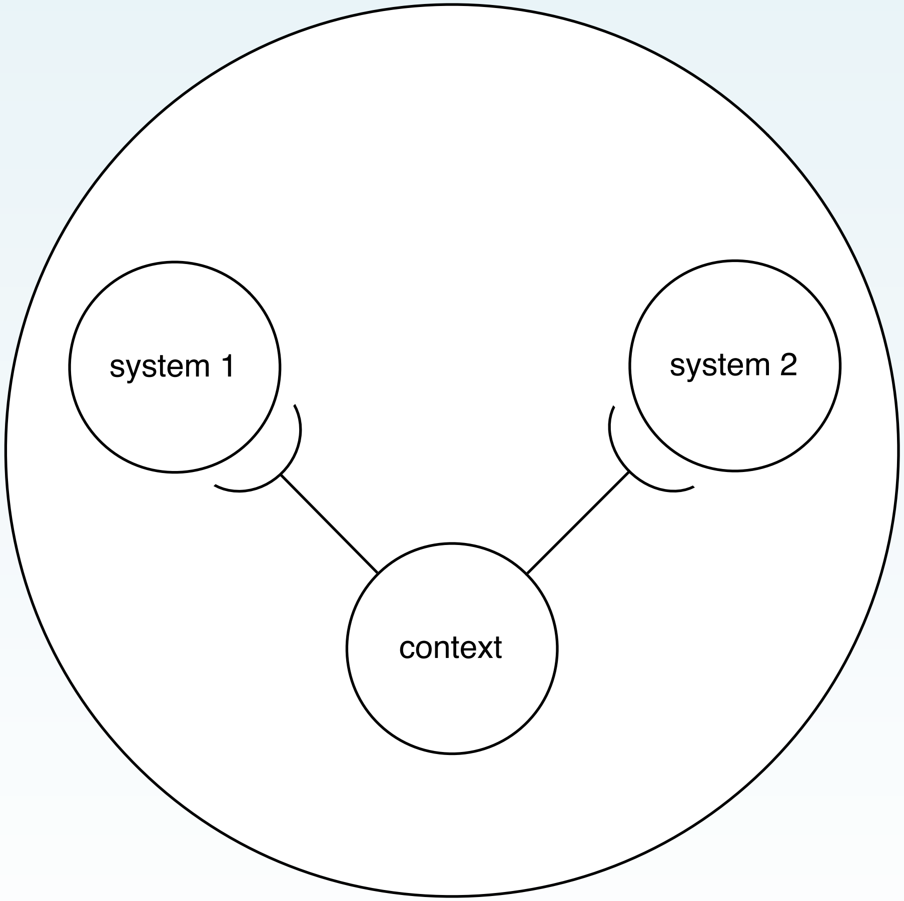


How to represent reality?



Graph, with index & pointer to represent potential future relationships?

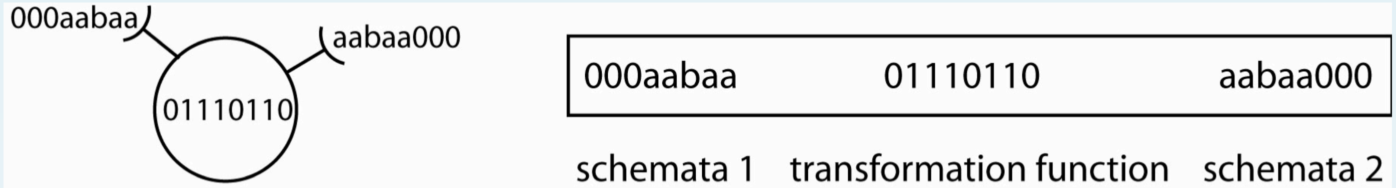
How to represent reality?

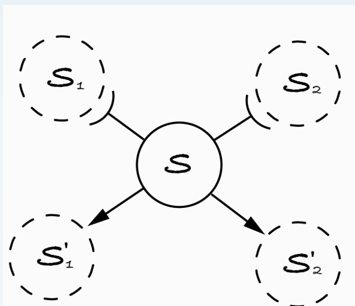


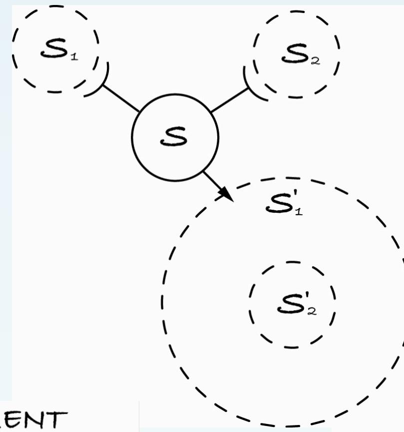
Pattern matching to discover new relationships?

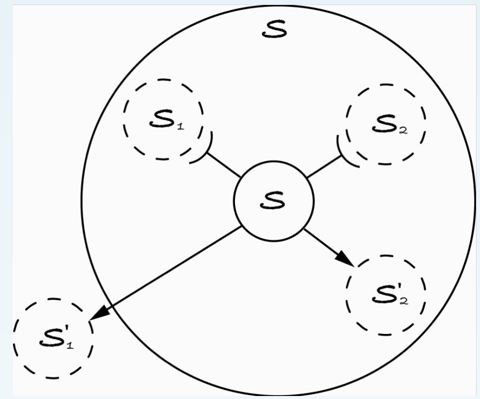
Systemic Computation

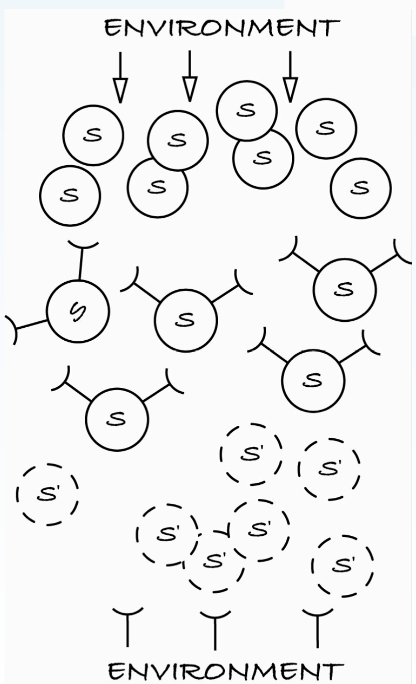
- Everything is a *system*
- Systems may comprise or share other nested systems.
- Systems can be transformed but never destroyed or created from nothing
- Interaction between systems may cause transformation of those systems, where the nature of that transformation is determined by a contextual system.
- All systems can potentially act as context and affect the interactions of other systems, and all systems can potentially interact in some context.
- The transformation of systems is constrained by the scope of systems.
- Computation is transformation

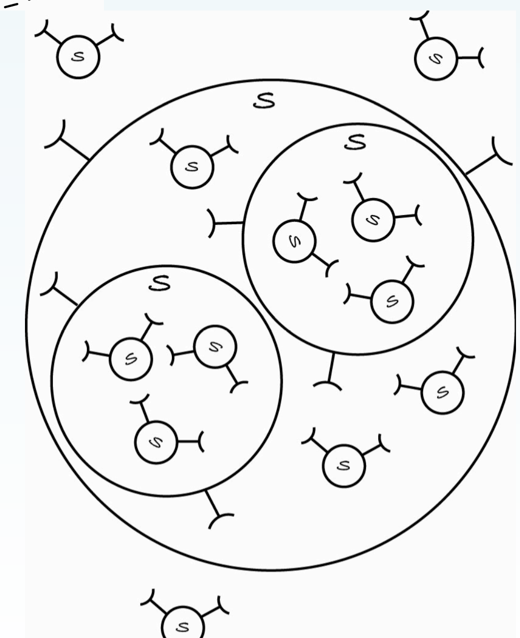

000aabaa
aabaa000
01110110
000aabaa
01110110
aabaa000
schemata 1
transformation function
schemata 2

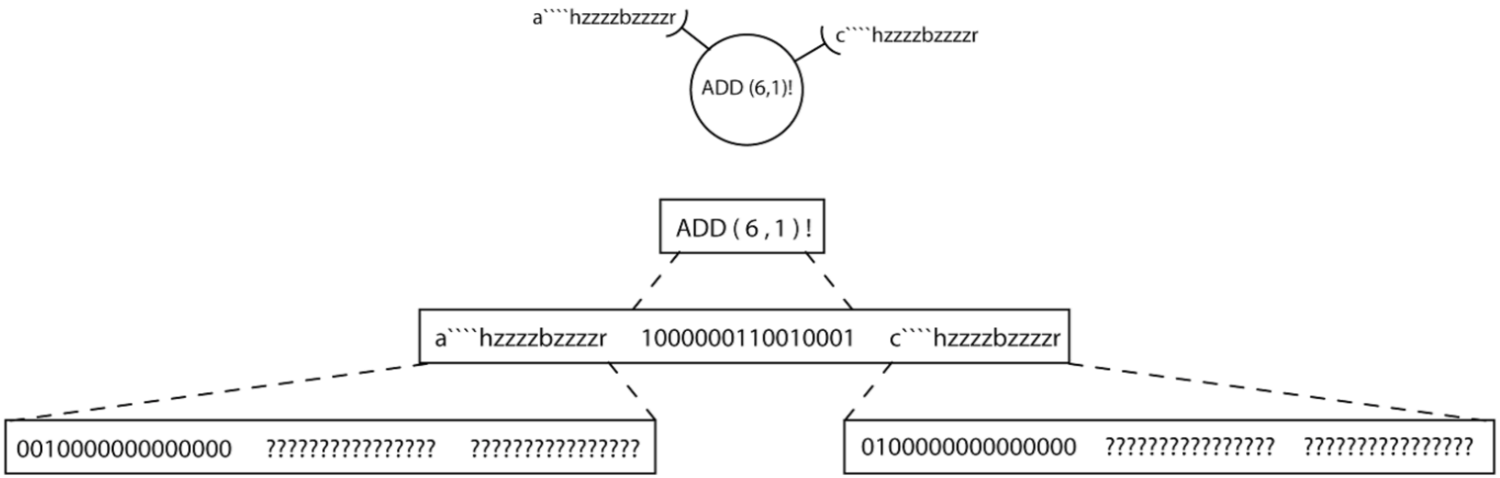







ENVIRONMENT
ENVIRONMENT





schemata code table

Code	value	code	value
0	000	n	11?
'	000	o	1?0
a	001	p	1?1
b	00?	q	1??
c	010	r	?00
d	011	s	?01
e	01?	t	?0?
f	0?0	u	?10
g	0?1	v	?11
h	0??	w	?1?
i	100	x	??0
j	101	y	??1
k	10?	z	???
l	110	1	111
m	111		

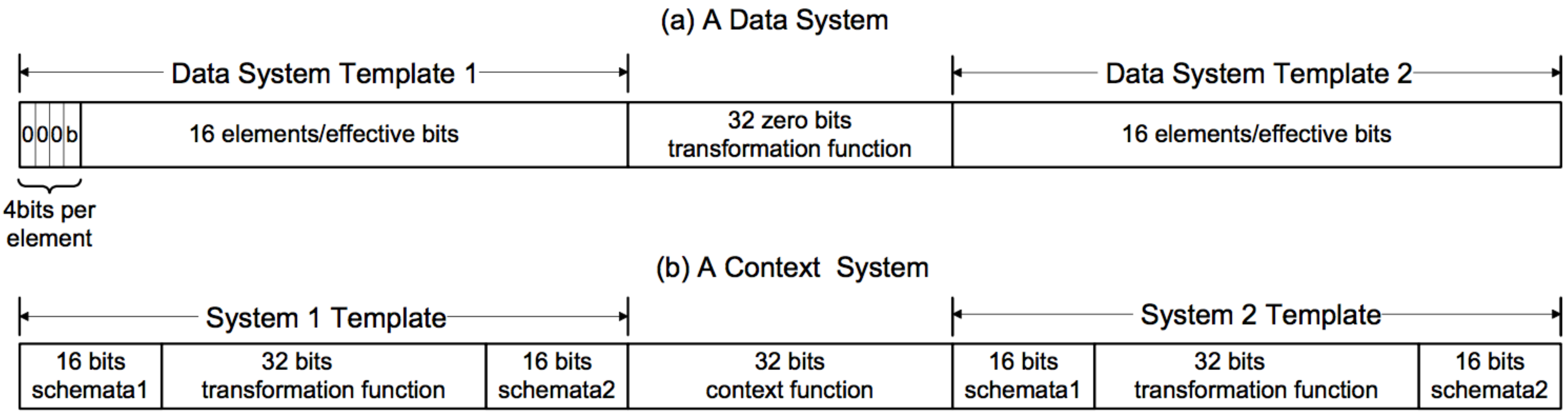
transformation function table

bits	Meaning
0..6	function identifier
7..10	schemata 1 matching threshold
11..14	schemata 2 matching threshold
15	NOT

scope table

System	1	2	3	4
1	0	0	0	0
2	0	0	0.5	0
3	1	0	0	0
4	1	0	0	0

Where ? means “don’t care”.


(a) A Data System
Data System Template 1
Data System Template 2
0 0 0 b
16 elements/effective bits
32 zero bits
transformation function
16 elements/effective bits
4bits per
element
(b) A Context System
System 1 Template
System 2 Template
16 bits
schemata1
32 bits
transformation function
16 bits
schemata2
32 bits
context function
16 bits
schemata1
32 bits
transformation function
16 bits
schemata2

Val	C	Val	C	Val	C	Val	C	Val	C	Val	C	Val	C	Val	C	Val	C
0000	!	0100	/	0?00	:	1000	@	1100	I	1?00	R	?000	`	?100	i	??00	r
0001	#	0101	2	0?01	;	1001	A	1101	J	1?01	S	?001	a	?101	j	??01	s
000?	%	010?	3	0?0?	<	100?	B	110?	K	1?0?	T	?00?	b	?10?	k	??0?	t
0010	&	0110	4	0?10	=	1010	C	1110	L	1?10	U	?010	c	?110	l	??10	u
0011	*	0111	5	0?11	>	1011	D	1111	M	1?11	V	?011	d	?111	m	??11	v
001?	+	011?	6	0?1?	[	101?	E	111?	N	1?1?	W	?01?	e	?11?	n	??1?	w
00?0	,	01?0	7	0??0	]	10?0	F	11?0	O	1??0	X	?0?0	f	?1?0	o	???0	x
00?1	-	01?1	8	0??1	^	10?1	G	11?1	P	1??1	Y	?0?1	g	?1?1	p	???1	y
00??	.	01??	9	0???	_	10??	H	11??	Q	1???	Z	?0??	h	?1??	q	????	z

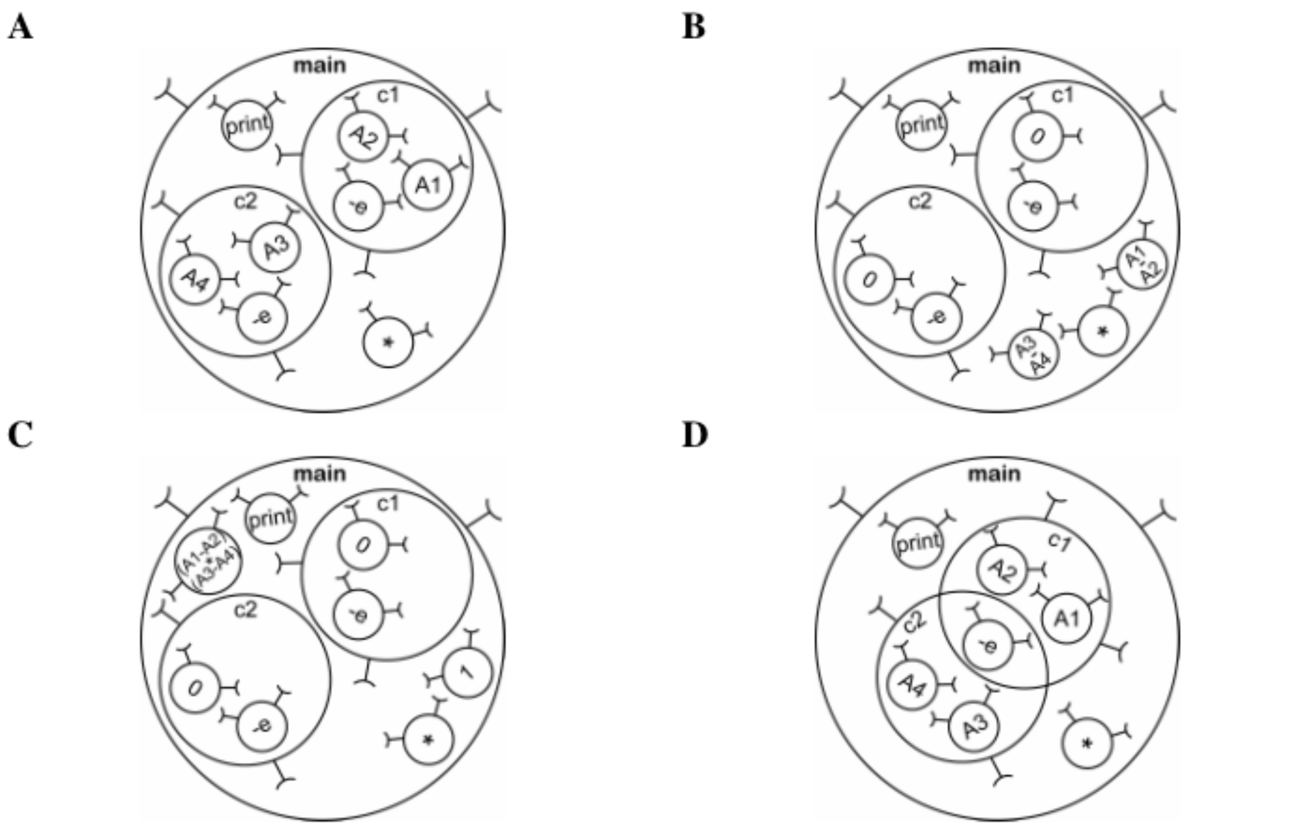
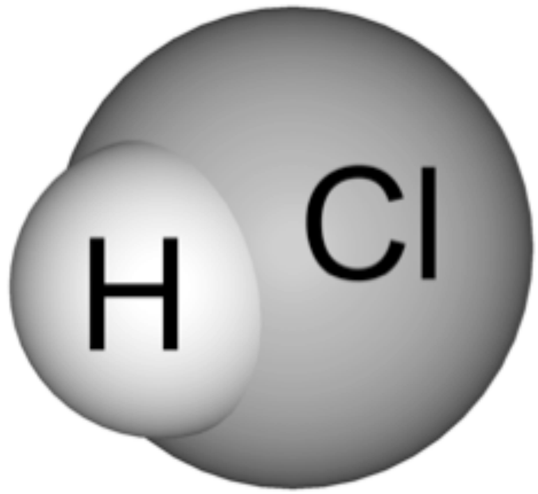


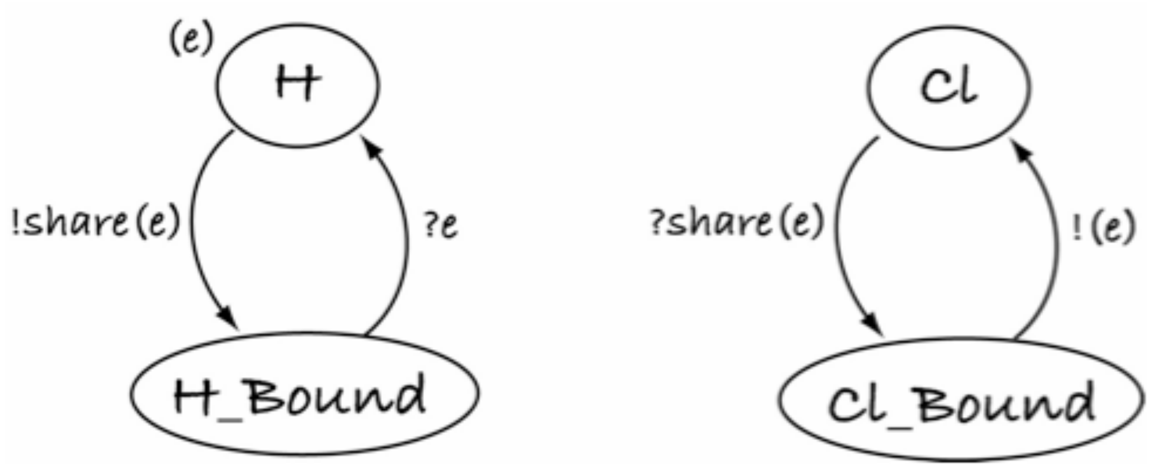
Fig. 5. Systemic computation calculation of PRINT((A1-A2)*(A3-A4)). The initial systems prior to calculation (**A**). Systems transformed by subtract-escape function '-e' (executed fragments shown in bold): PRINT((**A1-A2**)*(**A3-A4**)) (**B**). Systems transformed by multiply function, prior to activation of PRINT function: PRINT**((A1-A2)*(A3-A4))** (**C**). The same calculation can b§e performed in different ways, for example, a more compact, functionally equivalent arrangement of systems, sharing the subtract-escape function (**D**).

binding model *H* + *Cl* <--> *HCl*



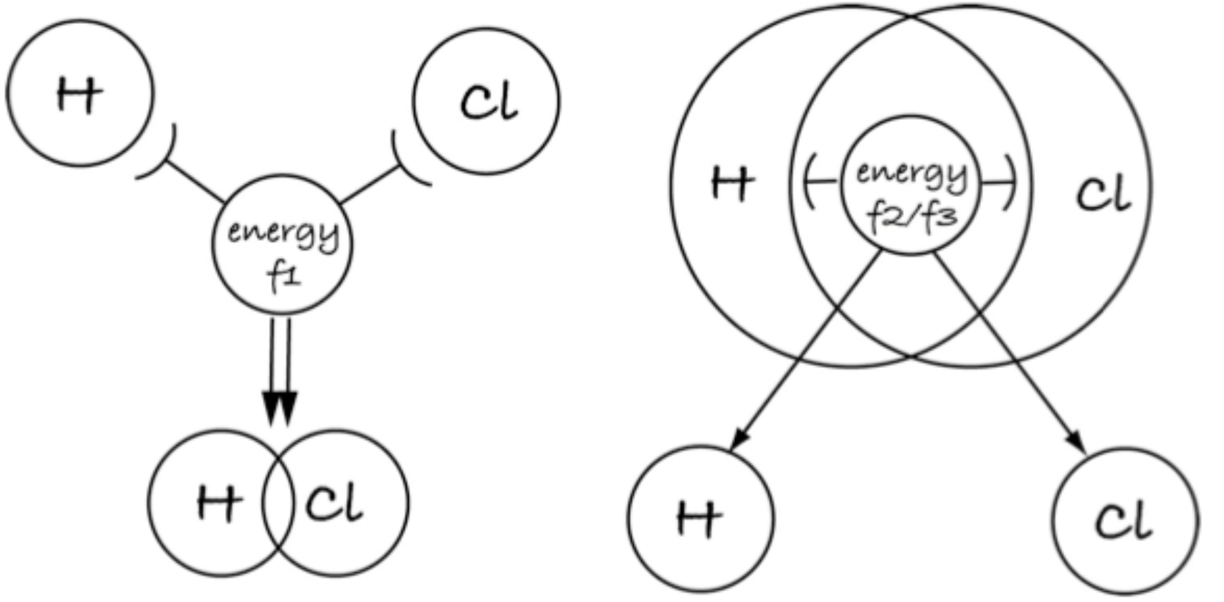
Molecular model of HCl (Hydrogen Chloride)

binding model *H* + *Cl* <--> *HCl*



H() = new e@10.0 (!share(e); H_Bound(e))
H_Bound(e) = !e; H()
Cl() = ?share(e); Cl_Bound(e)
Cl_Bound(e) = ?e; Cl()

binding model *H + Cl <--> HCl*

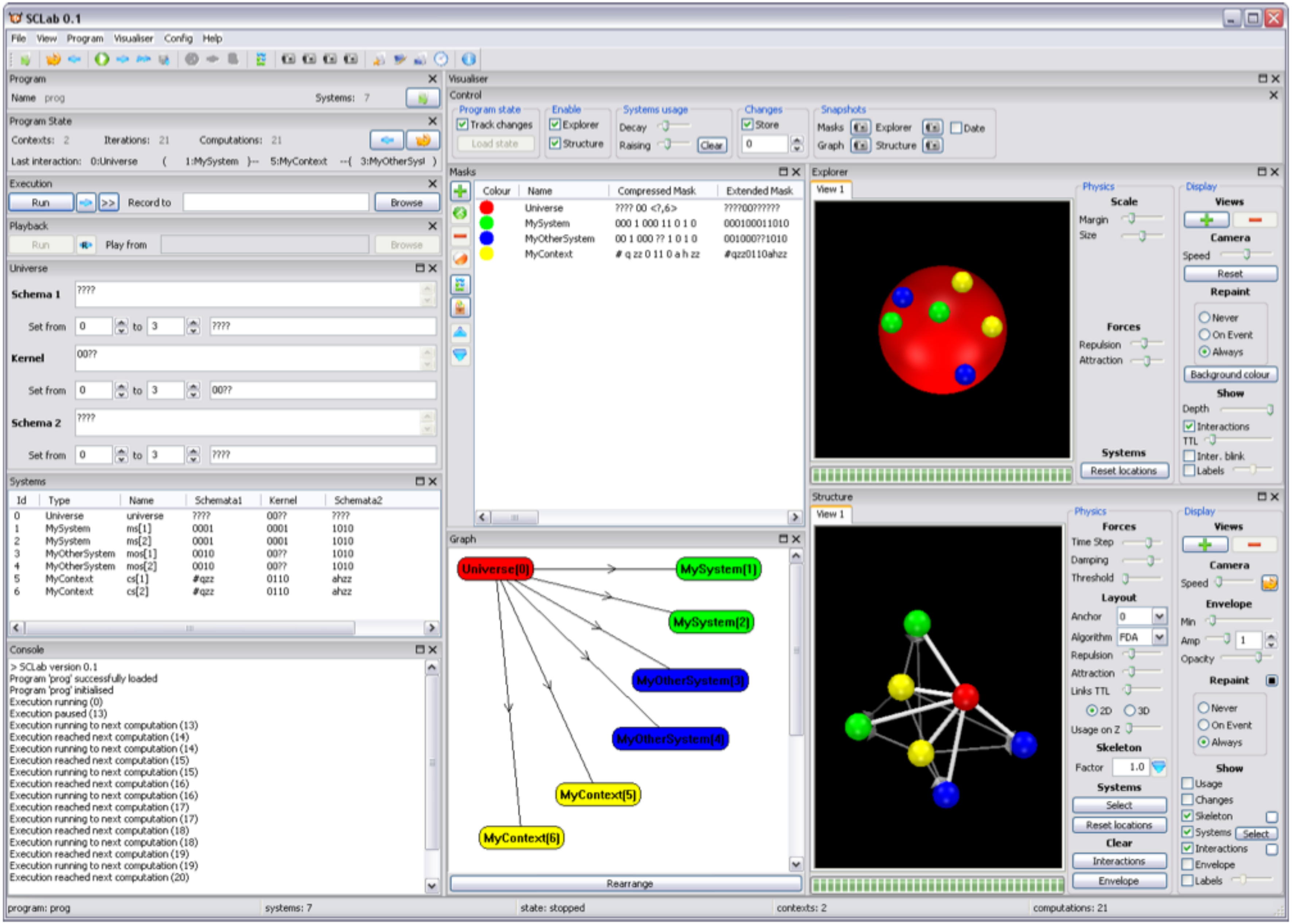


H }-energy$_{f1}$-{ Cl --> (H()Cl)
H(Cl) }}-energy$_{f2}$ --> H Cl
Cl(H) }}-energy$_{f3}$ --> H Cl

How to build a parallel, stochastic, distributed computer that runs quickly?

Solution 1: Simulation.

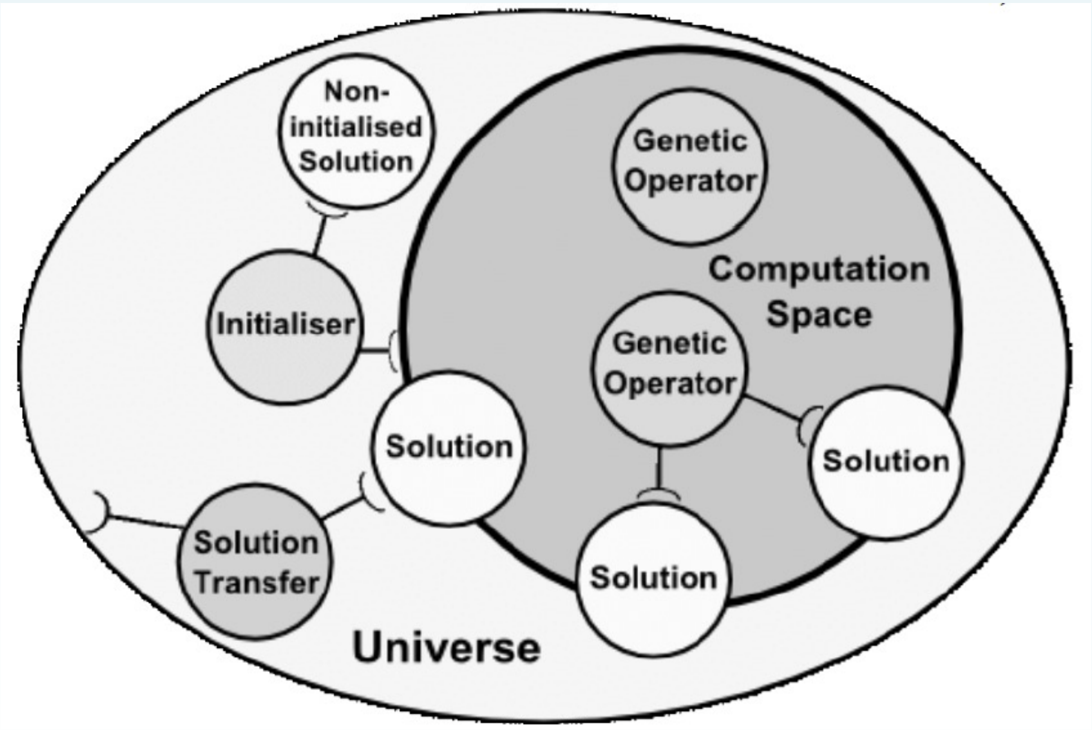
UCL

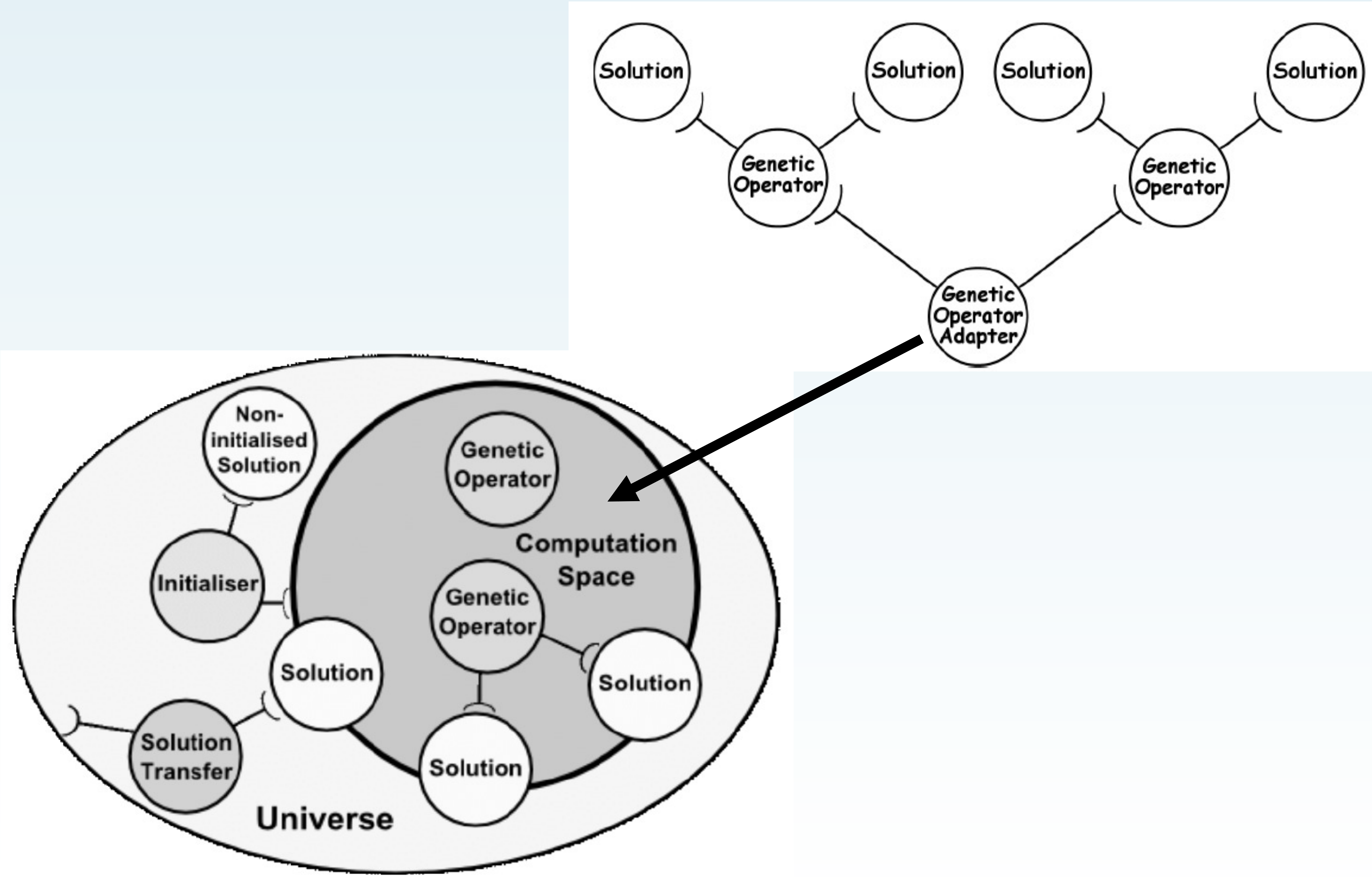

SCLab 0.1
File View Program Visualiser Config Help
Program
Name prog
Systems: 7
Program State
Contexts: 2 Iterations: 21 Computations: 21
Last interaction: 0:Universe (1:MySystem)-- 5:MyContext --(3:MyOtherSyst)
Execution
Run
Record to
Browse
Playback
Run
Play from
Browse
Universe
Schema 1
????
Set from 0 to 3 ????
Kernel
00??
Set from 0 to 3 00??
Schema 2
????
Set from 0 to 3 ????
Systems
Id Type Name Schemata1 Kernel Schemata2
0 Universe universe ???? 00?? ????
1 MySystem ms[1] 0001 0001 1010
2 MySystem ms[2] 0001 0001 1010
3 MyOtherSystem mos[1] 0010 00?? 1010
4 MyOtherSystem mos[2] 0010 00?? 1010
5 MyContext cs[1] #qzz 0110 ahzz
6 MyContext cs[2] #qzz 0110 ahzz
Console
> SCLab version 0.1
Program 'prog' successfully loaded
Program 'prog' initialised
Execution running (0)
Execution paused (13)
Execution running to next computation (13)
Execution reached next computation (14)
Execution running to next computation (14)
Execution reached next computation (15)
Execution running to next computation (15)
Execution reached next computation (16)
Execution running to next computation (16)
Execution reached next computation (17)
Execution running to next computation (17)
Execution reached next computation (18)
Execution running to next computation (18)
Execution reached next computation (19)
Execution running to next computation (19)
Execution reached next computation (20)
Visualiser
Control
Program state
Track changes
Load state
Enable
Explorer
Structure
Systems usage
Decay
Raising
Clear
Changes
Store
0
Snapshots
Masks Explorer Date
Graph Structure
Masks
Colour Name Compressed Mask Extended Mask
Universe ???? 00 <?,6> ????00??????
MySystem 000 1 000 11 0 1 0 000100011010
MyOtherSystem 00 1 000 ?? 1 0 1 0 001000??1010
MyContext # q zz 0 11 0 a h zz #qzz0110ahzz
Graph
Universe[0]
MySystem[1]
MySystem[2]
MyOtherSystem[3]
MyOtherSystem[4]
MyContext[5]
MyContext[6]
Rearrange
Explorer
View 1
Physics
Scale
Margin
Size
Forces
Repulsion
Attraction
Systems
Reset locations
Display
Views
Camera
Speed
Reset
Repaint
Never
On Event
Always
Background colour
Show
Depth
Interactions
TTL
Inter. blink
Labels
Structure
View 1
Physics
Forces
Time Step
Damping
Threshold
Layout
Anchor 0
Algorithm FDA
Repulsion
Attraction
Links TTL
2D 3D
Usage on Z
Skeleton
Factor 1.0
Systems
Select
Reset locations
Clear
Interactions
Envelope
Display
Views
Camera
Speed
Envelope
Min
Amp 1
Opacity
Repaint
Never
On Event
Always
Show
Usage
Changes
Skeleton
Systems Select
Interactions
Envelope
Labels
program: prog
systems: 7
state: stopped
contexts: 2
computations: 21

```
#systemic start

  // define the functions
#function NOP         %b0000000000000000
#function ADD         %b1000000000000000
#function ADDe        %b0100000000000000
#function SUBTRACT    %b1100000000000000
#function SUBTRACTe   %b0010000000000000
#function MULT        %b1010000000000000
#function MULTe       %b0110000000000000
#function DIV         %b1110000000000000
#function DIVe        %b0001000000000000
#function AND         %b1101000000000000
#function OR          %b0011000000000000
#function EOR         %b1011000000000000
#function ZERO        %b0111000000000000
#function ESCAPE      %b1111000000000000
#function CAPTURE     %b0000100000000000
#function PRINT       %b1000100000000000
#function COPY        %b0100100000000000
#function ISZERO      %b1100100000000000

// define some useful labels
#label zero        %b0000000000000000
#label dontcare    %b????????????????
#label num1        %b1000000000000000
#label num2        %b0100000000000000
#label num3        %b1100000000000000
#label num4        %b0010000000000000

// the program begins here:
main {%d0 %d0 %d0}
c1 {%d255 %d255 %d255}
data1 {num1 %d0 %d10}
data2 {num2 %d0 %d5}
minus    {[ num1 zero dontcare]
SUBTRACTe(0,0) [ num2 zero dontcare] }

#scope c1
{
      data1
      data2
      minus
}
c2 {%d255 %d255 %d255}
data3 {num1 %d0 %d16}
data4 {num2 %d0 %d4}

#scope c2
{
      data3
      data4
      minus
}
times {[dontcare zero dontcare] MULT(0,0)
[ dontcare zero dontcare] }
output  {[dontcare dontcare dontcare]
PRINT(0,0) [ dontcare dontcare dontcare] }

#scope main
{
       c1
       c2
       times
       output
}
#systemic end
```

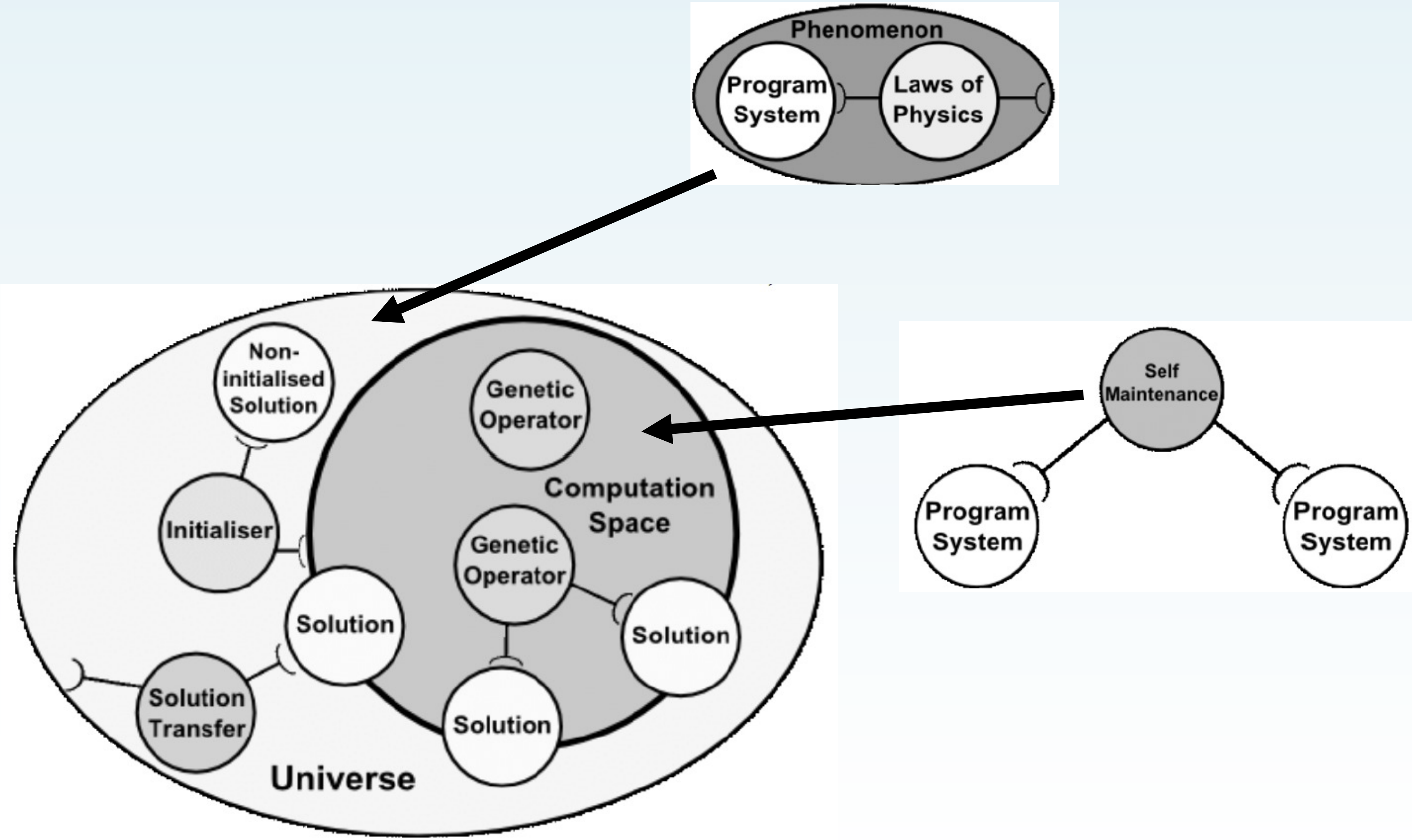
```
// systemic computation code autogenerated by
assembler from /Users/Peter/My
programs/compilesc2/build/calculation1.sc

// number of functions
20

// number of systems
10

// scope table
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

// function definitions
NOP 0000000000000000
ADD 1000000000000000
ADDe 0100000000000000
SUBTRACT 1100000000000000
SUBTRACTe 0010000000000000
MULT 1010000000000000
MULTe 0110000000000000
DIV 1110000000000000
DIVe 0001000000000000
AND 1101000000000000
OR 0011000000000000
EOR 1011000000000000
ZERO 0111000000000000
ESCAPE 1111000000000000
CAPTURE 0000100000000000
PRINT 1000100000000000
COPY 0100100000000000
ISZERO 1100100000000000

// system definitions
0000000000000000 0000000000000000 0000000000000000
1111111100000000 1111111100000000 1111111100000000
1000000000000000 0000000000000000 0101000000000000
0100000000000000 0000000000000000 1010000000000000
i`````````bzzzzz 0010000000000000 c`````````bzzzzz
1111111100000000 1111111100000000 1111111100000000
1000000000000000 0000000000000000 0000100000000000
0100000000000000 0000000000000000 0010000000000000
zzzzzr````bzzzzz 1010000000000000 zzzzzr````bzzzzz
zzzzzzzzzzzzzzzz 1000100000000000 zzzzzzzzzzzzzzzz
```

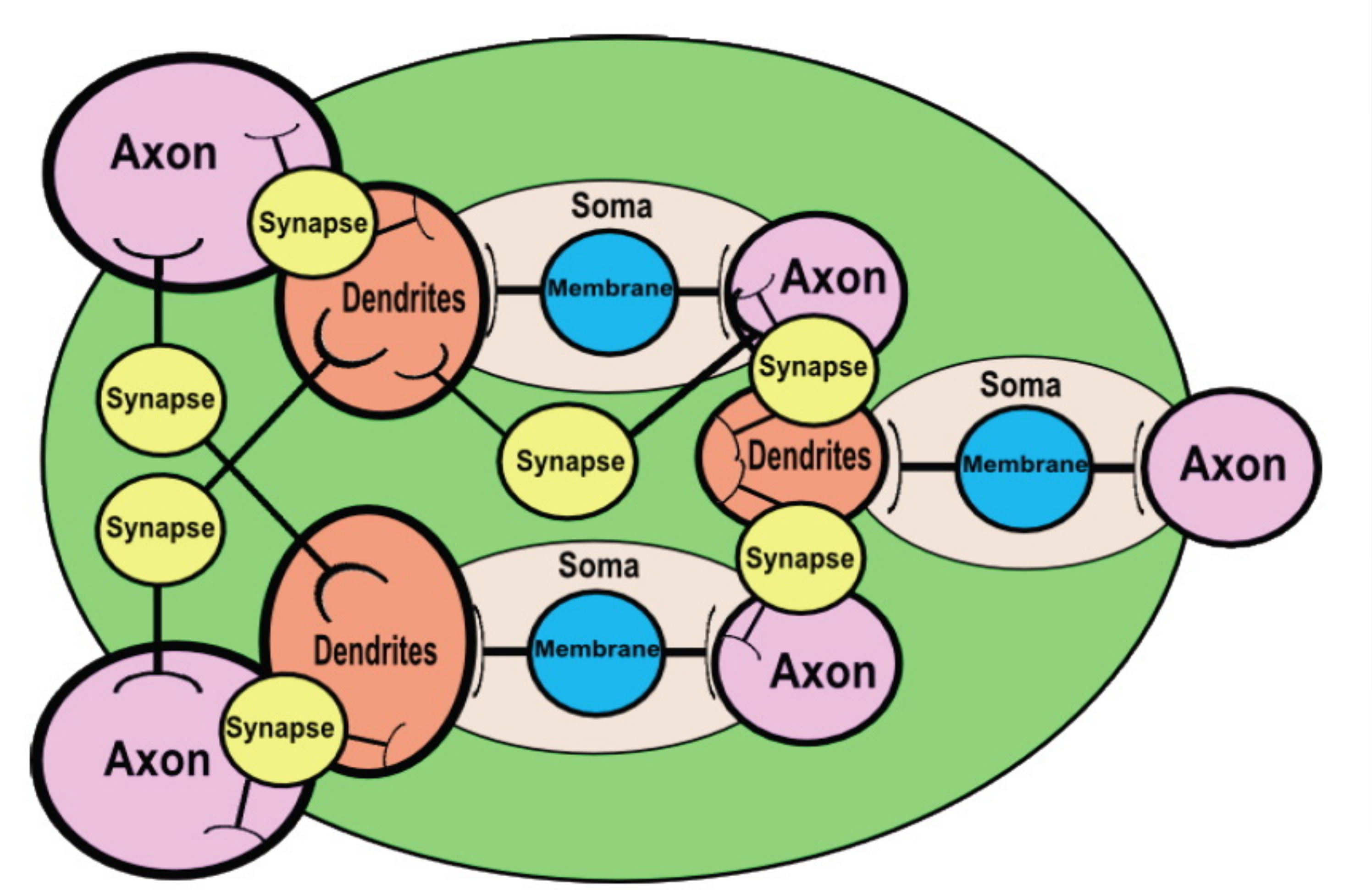
Fig. 6. Assembly language for systemic computer; system definitions take the form: "`textual-identifier ([ schemata 1] transformation-function [ schemata 2] )`" (left). Corresponding compiled machine code (right)

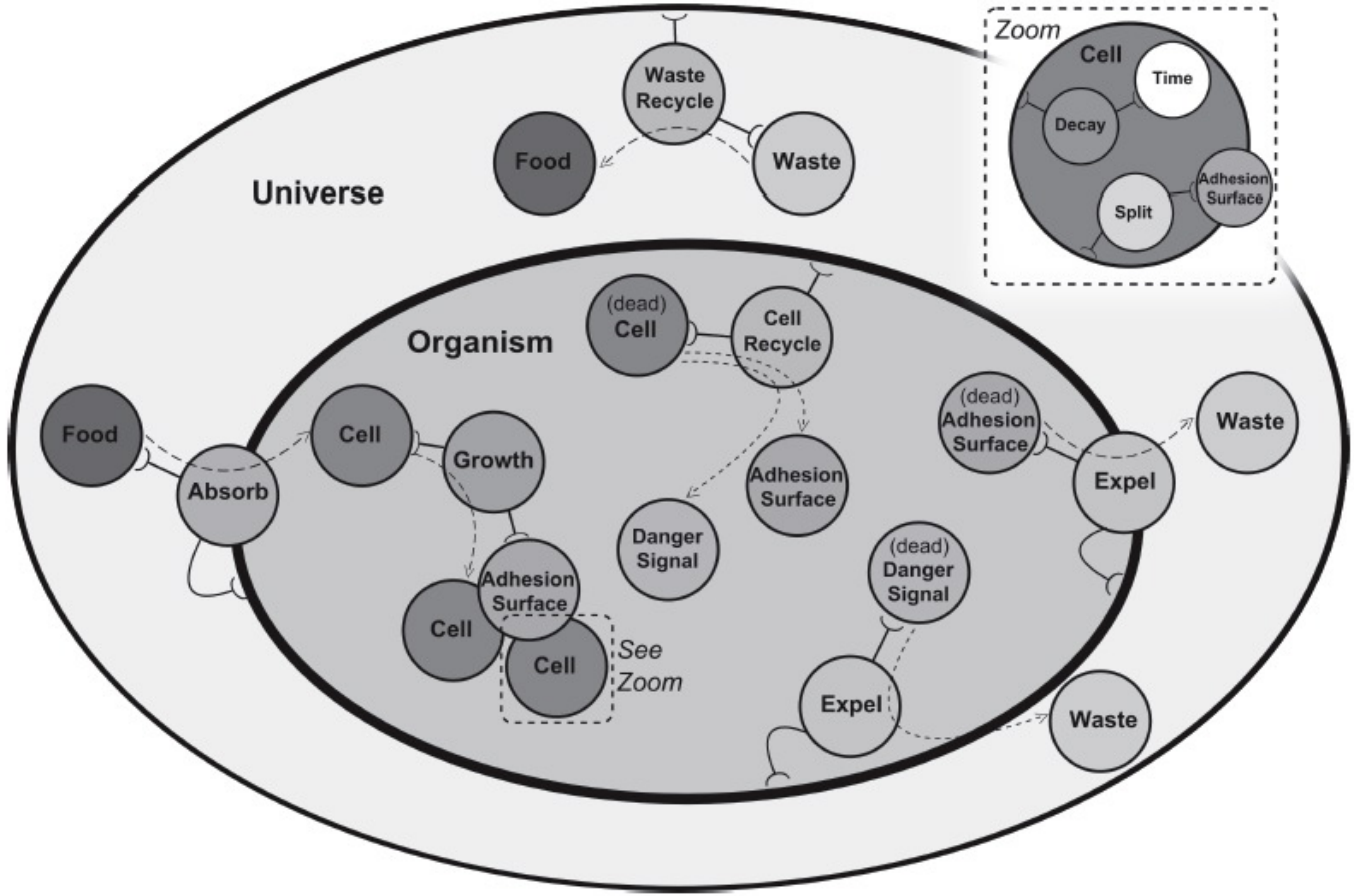

Non-
initialised
Solution
Genetic
Operator
Computation
Space
Initialiser
Genetic
Operator
Solution
Solution
Solution
Transfer
Solution
Solution
Universe



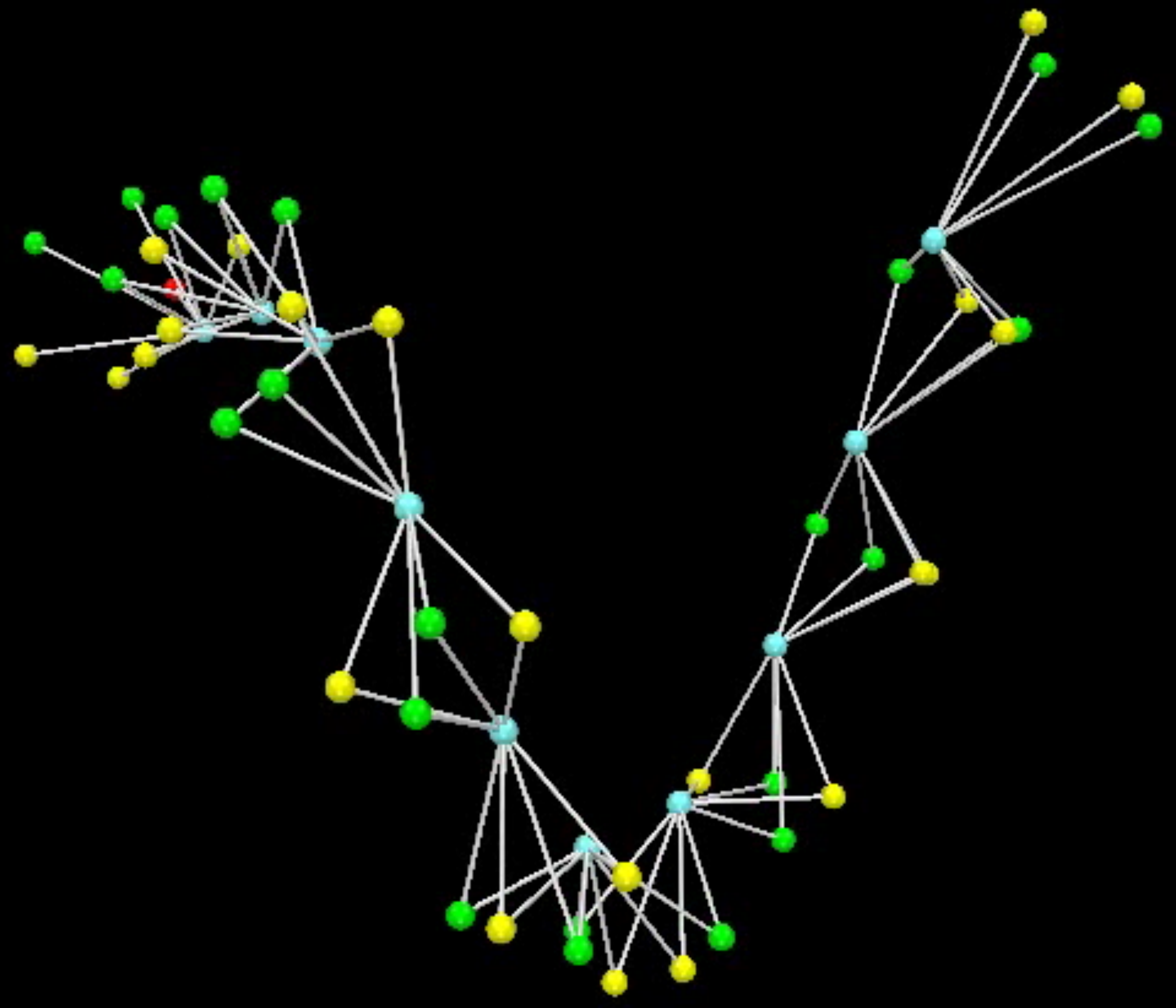
Solution
Solution
Solution
Solution
Genetic Operator
Genetic Operator
Genetic Operator Adapter
Non-initialised Solution
Initialiser
Solution
Solution Transfer
Genetic Operator
Computation Space
Genetic Operator
Solution
Solution
Universe


Phenomenon
Program System
Laws of Physics
Non-initialised Solution
Initialiser
Solution
Solution Transfer
Genetic Operator
Computation Space
Genetic Operator
Solution
Solution
Universe
Self Maintenance
Program System
Program System

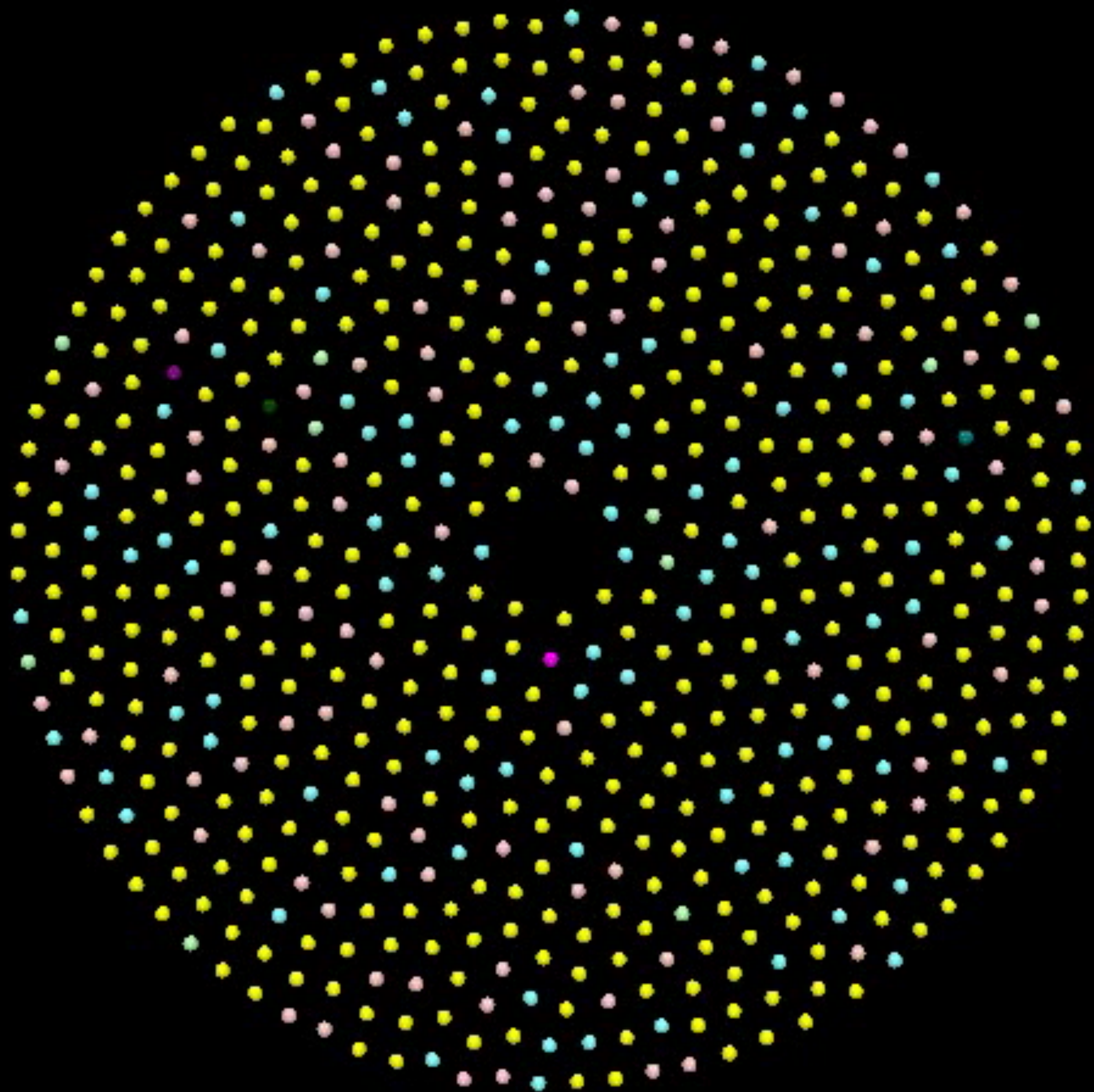

Axon
Synapse
Soma
Dendrites
Membrane
Axon
Synapse
Synapse
Synapse
Soma
Dendrites
Membrane
Axon
Synapse
Synapse
Soma
Dendrites
Membrane
Axon
Synapse
Axon


Universe
Food
Waste Recycle
Waste
Zoom
Cell
Time
Decay
Split
Adhesion Surface
Organism
(dead) Cell
Cell Recycle
Food
Absorb
Cell
Growth
Adhesion Surface
Danger Signal
(dead) Adhesion Surface
Expel
Waste
Adhesion Surface
Cell
Cell
See Zoom
(dead) Danger Signal
Expel
Waste

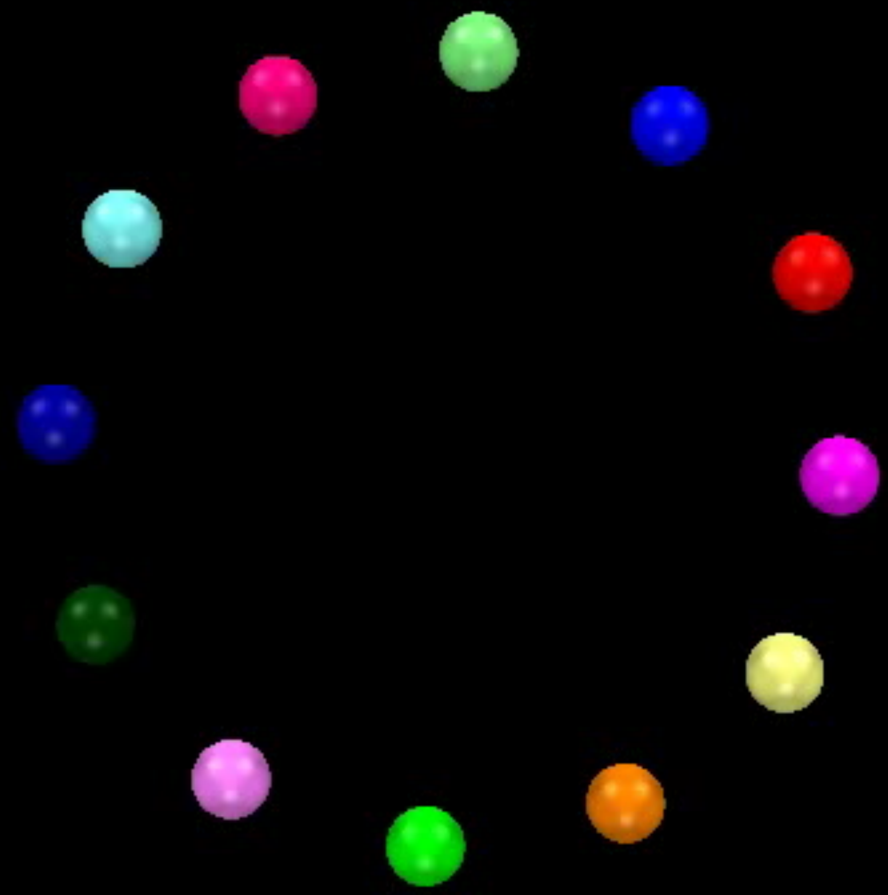
Visualising fire



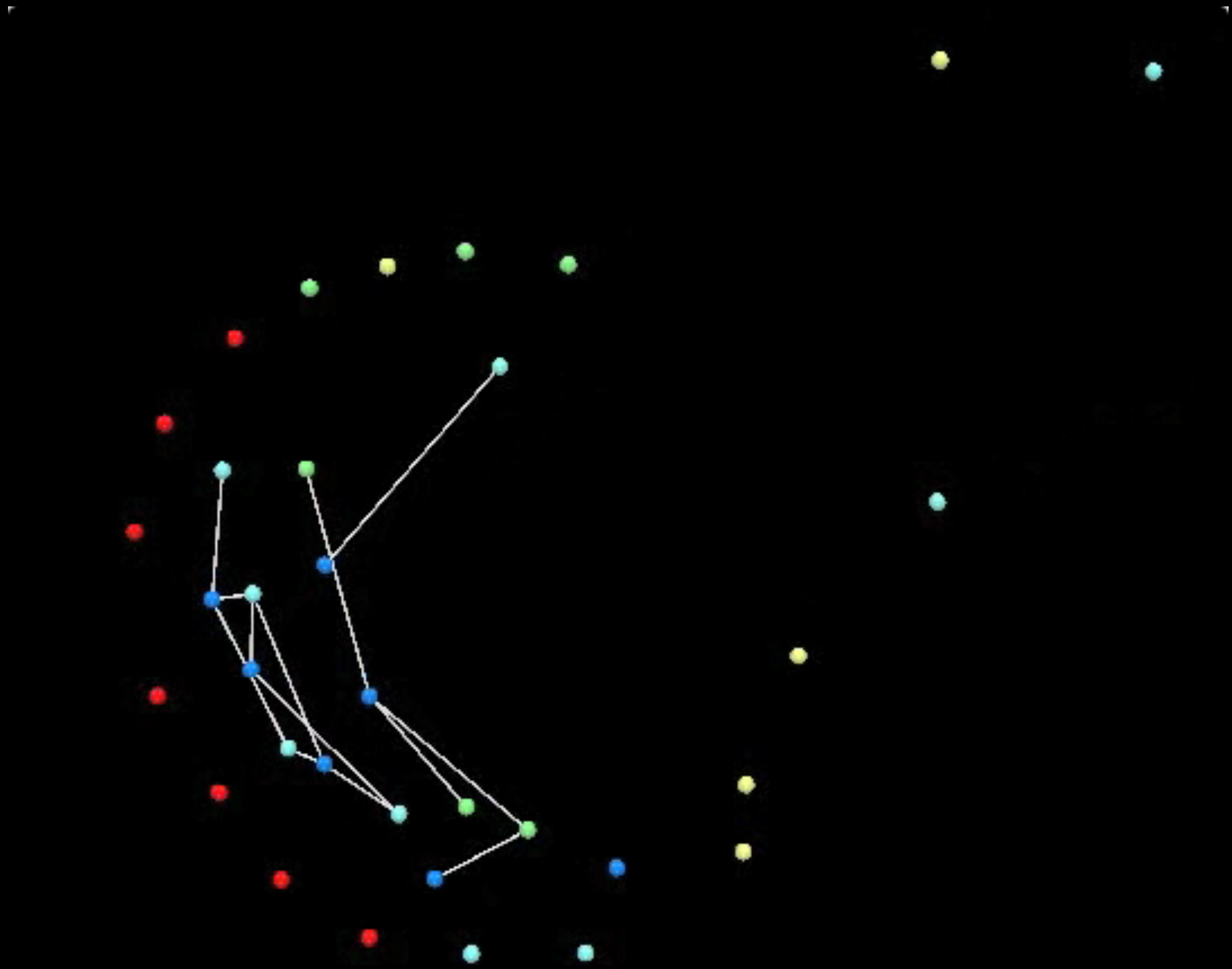
MAPK phosphorylation model

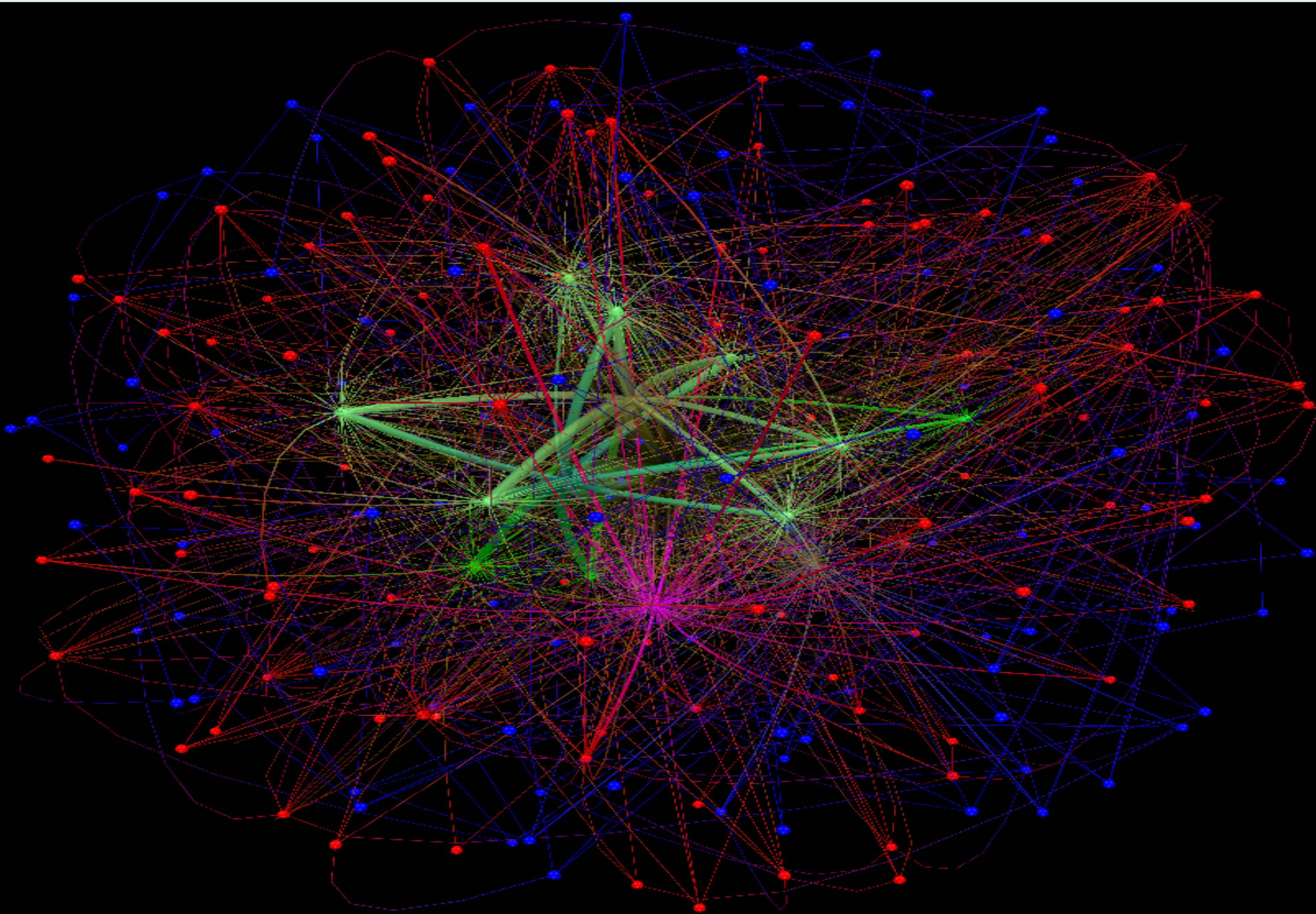


Bi stable gene network



Model of Organism



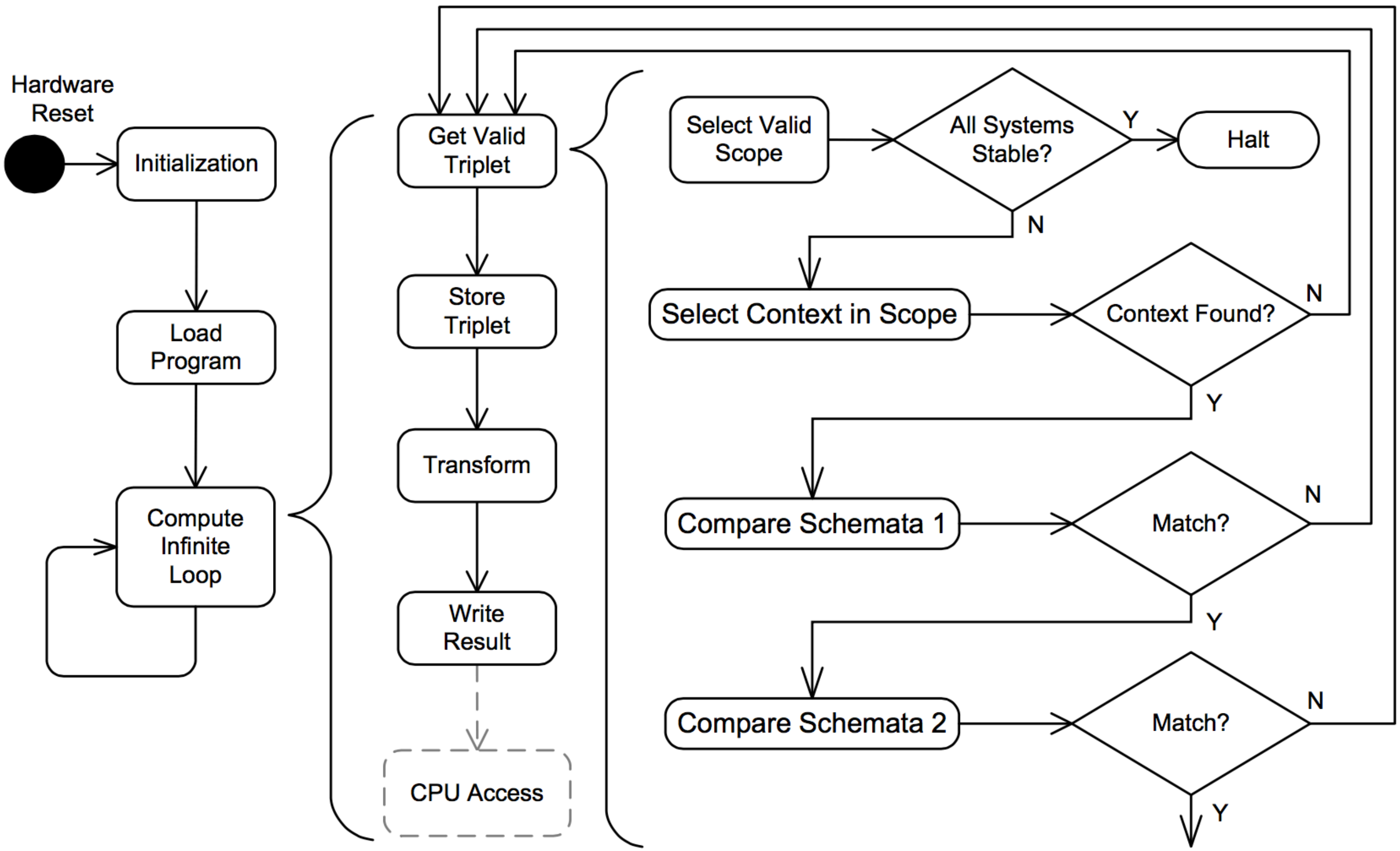


How to build a parallel, stochastic, distributed computer that runs quickly?

Solution 1: Simulation.
Good proof of concept, but slow!

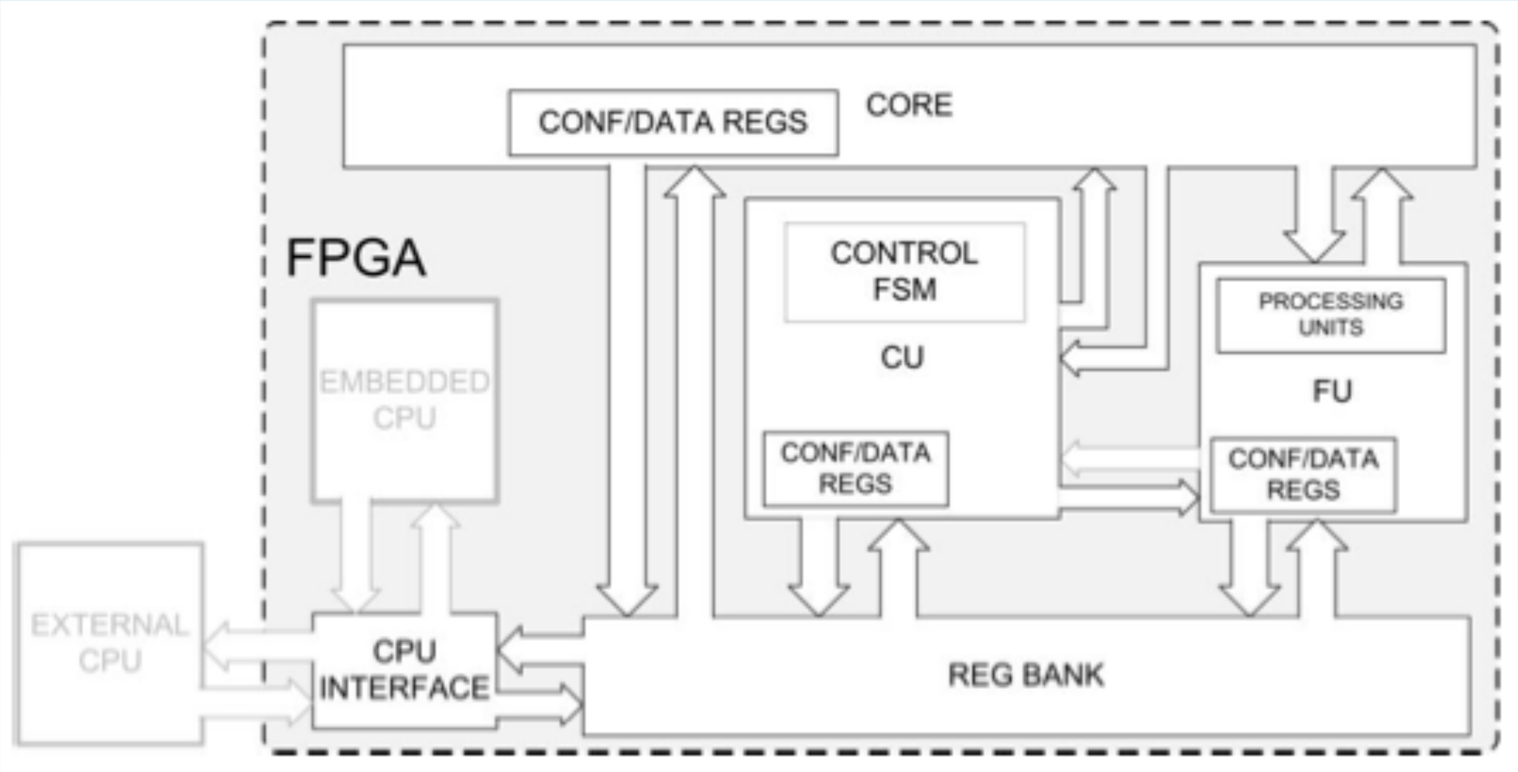
Solution 2: Novel hardware.

FPGA SC –speed up key bottlenecks



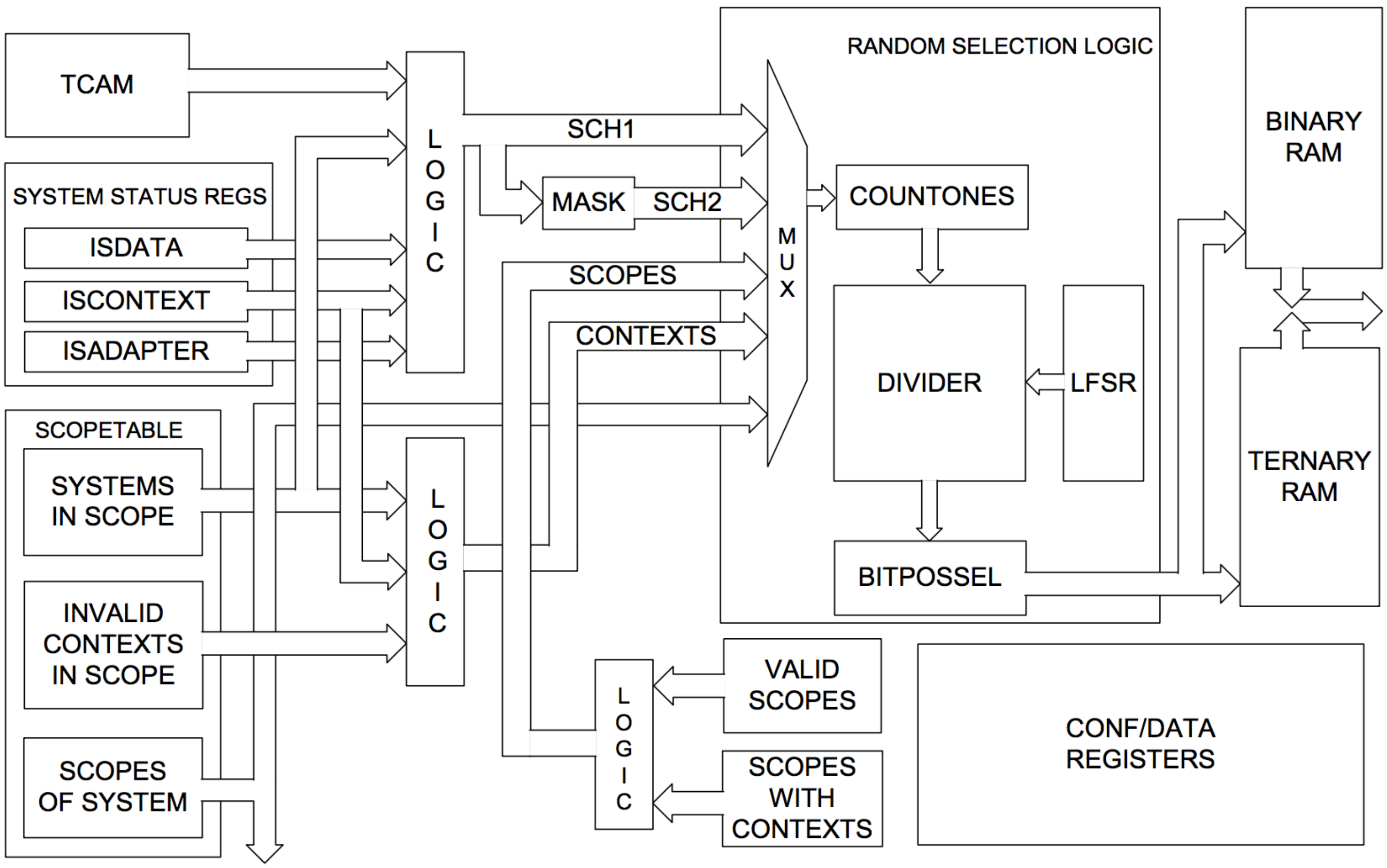
HAoS Program Control Flow : HAoS enters an infinite computation loop after the SC program is loaded, which involves finding valid triplets and transforming the selected systems

FPGA SC



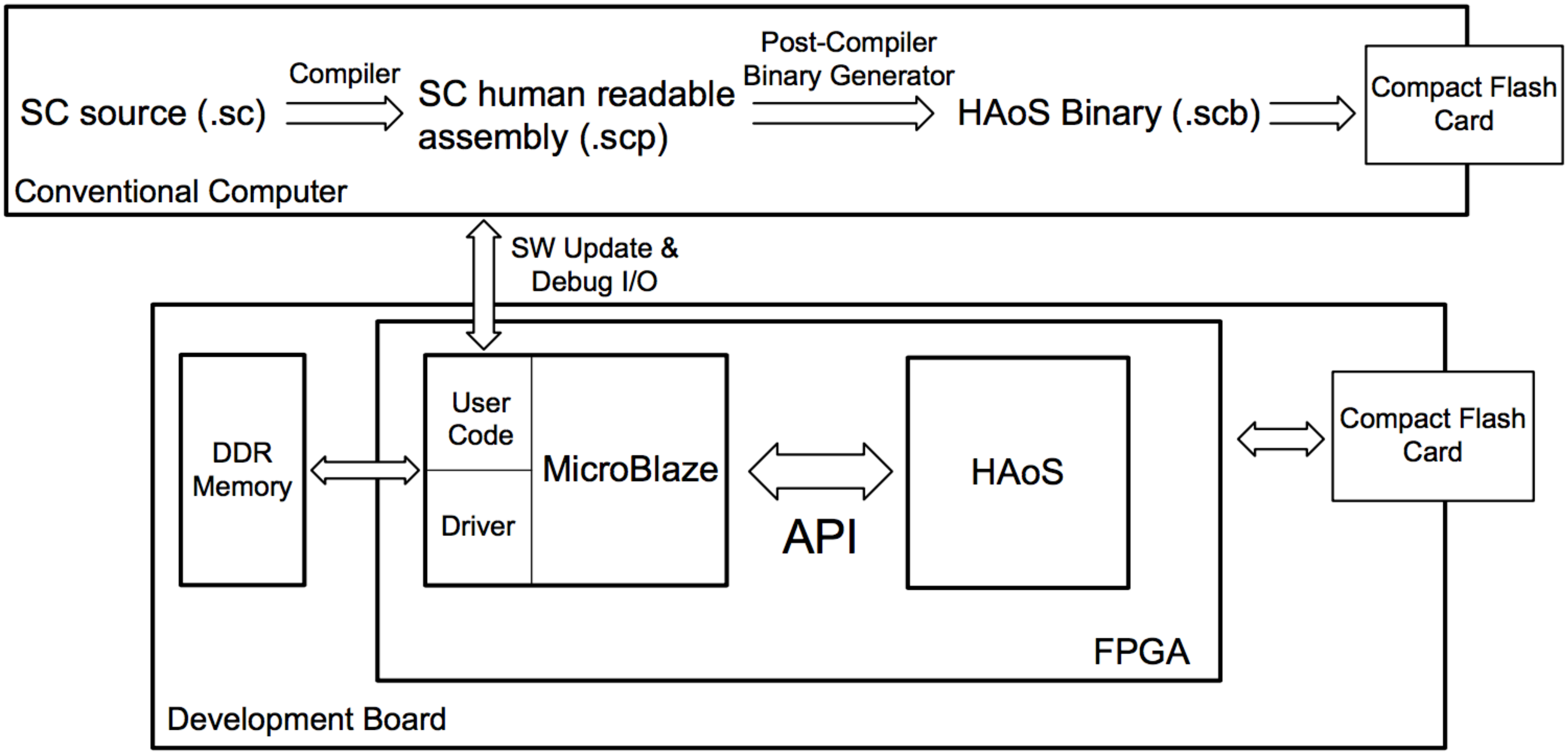
The SC FPGA Hardware Architecture.
CORE contains optimized logic for parallel schemata matching and memory elements.
CU handles execution sequence of SC program and communication with optional CPU.
REG BANK provides control and debug interface between CPU and local registers of SC sub-modules.
FU provides basic local processing functionality. A set of simple instructions is supported to avoid expensive data transfers between the REG BANK and the CPU.

FPGA SC



HAoS Core basic building blocks

FPGA SC

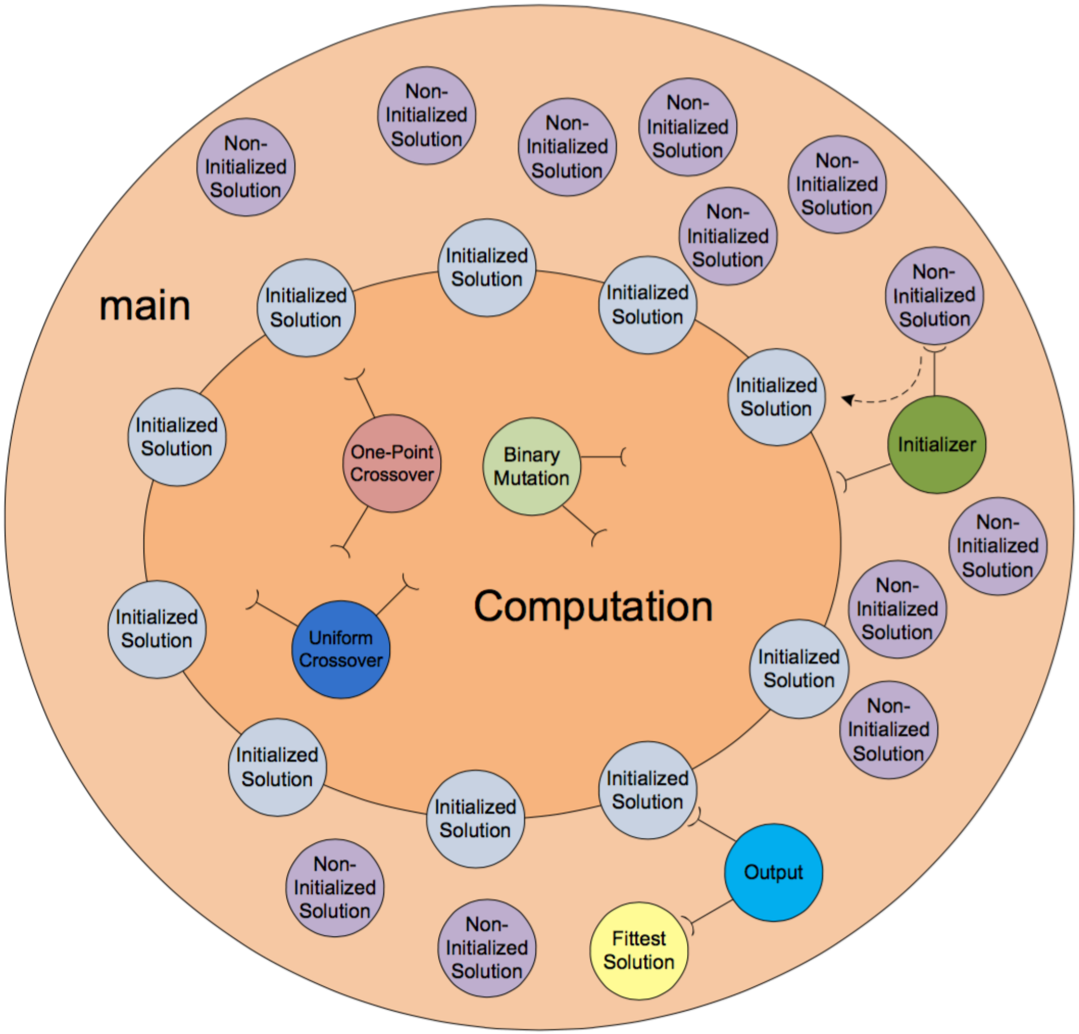


. **HAoS programming toolchain and software framework illustrating the complete suggested programming platform**

FPGA SC

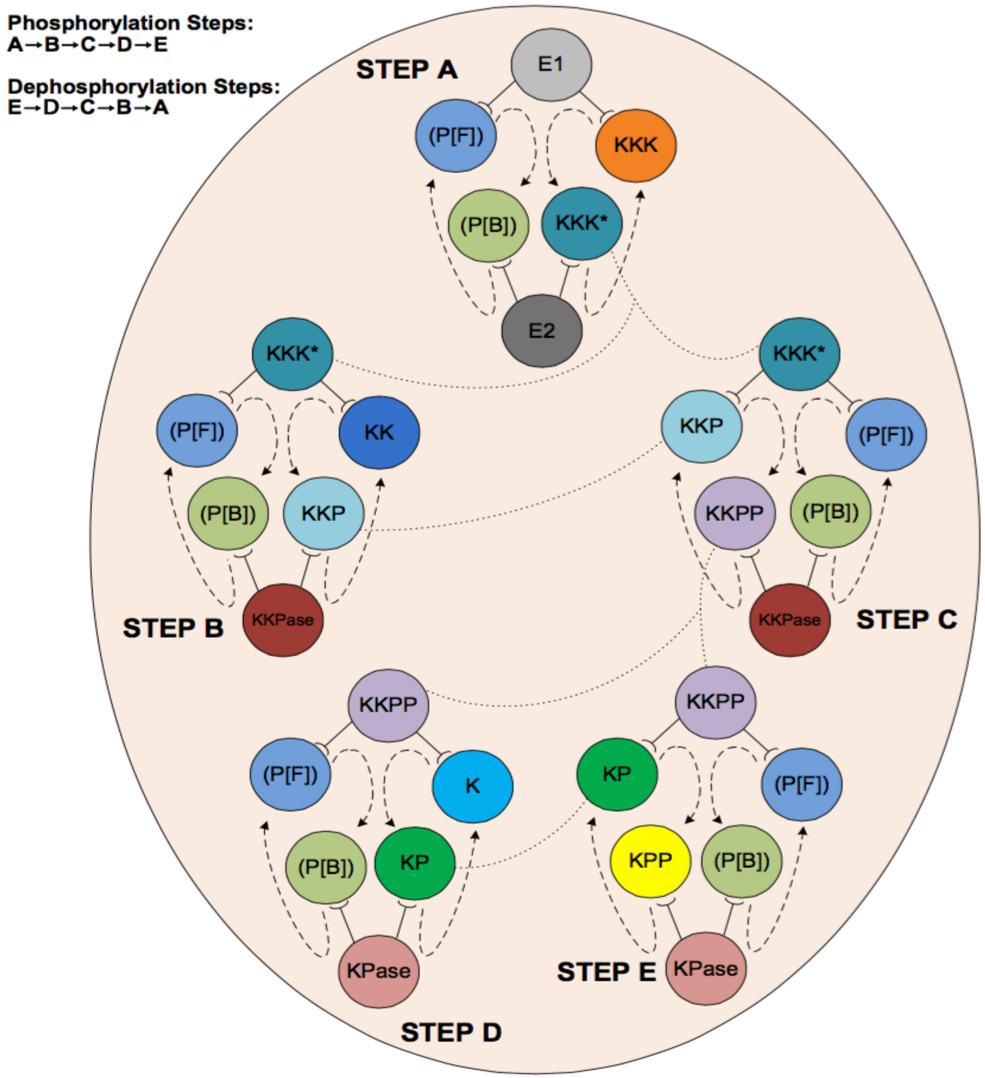

Welcome to HAoS
Knapsack Demo!!!
Transcend
CompactFlash
2GB
Industrial

Knapsack problem using genetic algorithm



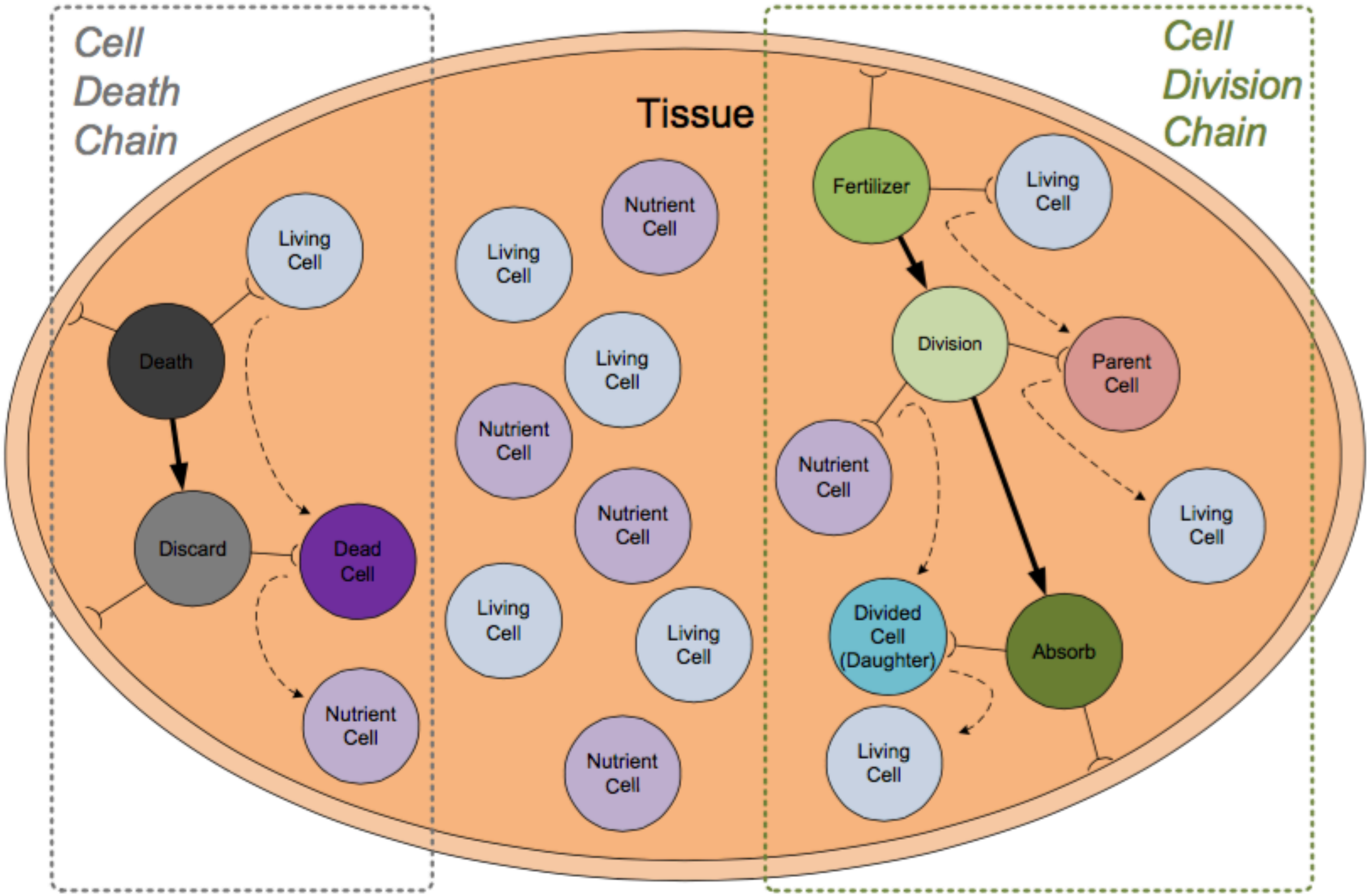
The binary knapsack SC model. Non-initialized solutions are initialized by the initializer context and added into the computation scope where they are transformed through genetic operations. The output context updates, if necessary, the fittest solution.

MAPK phosphorylation model



The HAoS MAPK model in SC graphical notation. During phosphorylation, E1 mitogens activate KKKs, which become KKK*s and phosphorylate KKs, which, when double phosphorylated, become KKPPs and phosphorylate Ks. This process is reversed during dephosphorylation with KPase and KKPase phosphatases and E2 mitogens bringing the cascade to its initial state.

Cancer development model



Optimized SC cancer model. The surgery functionality is embedded in the death context while the contexts implementing the two main genetic operations, death and division, are chained

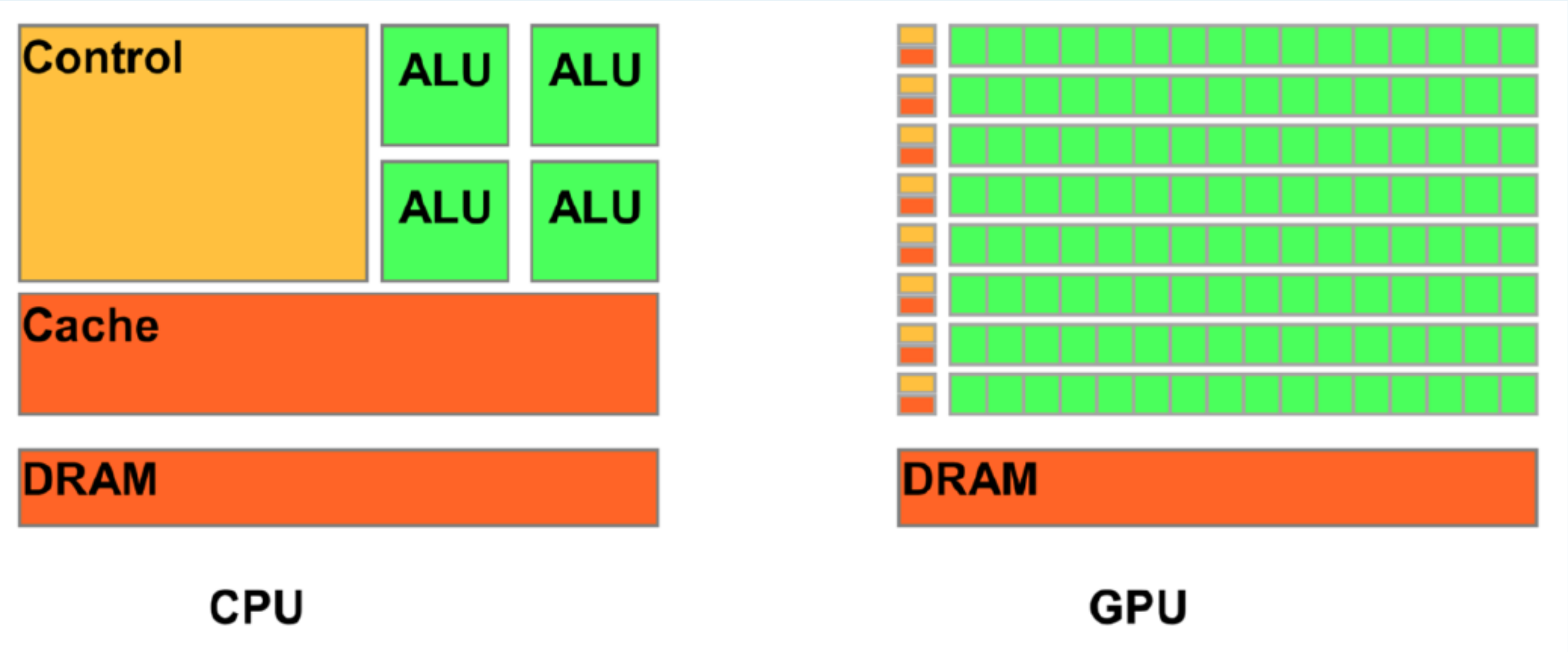
How to build a parallel, stochastic, distributed computer that runs quickly?

Solution 1: Simulation.
Good proof of concept, but slow!

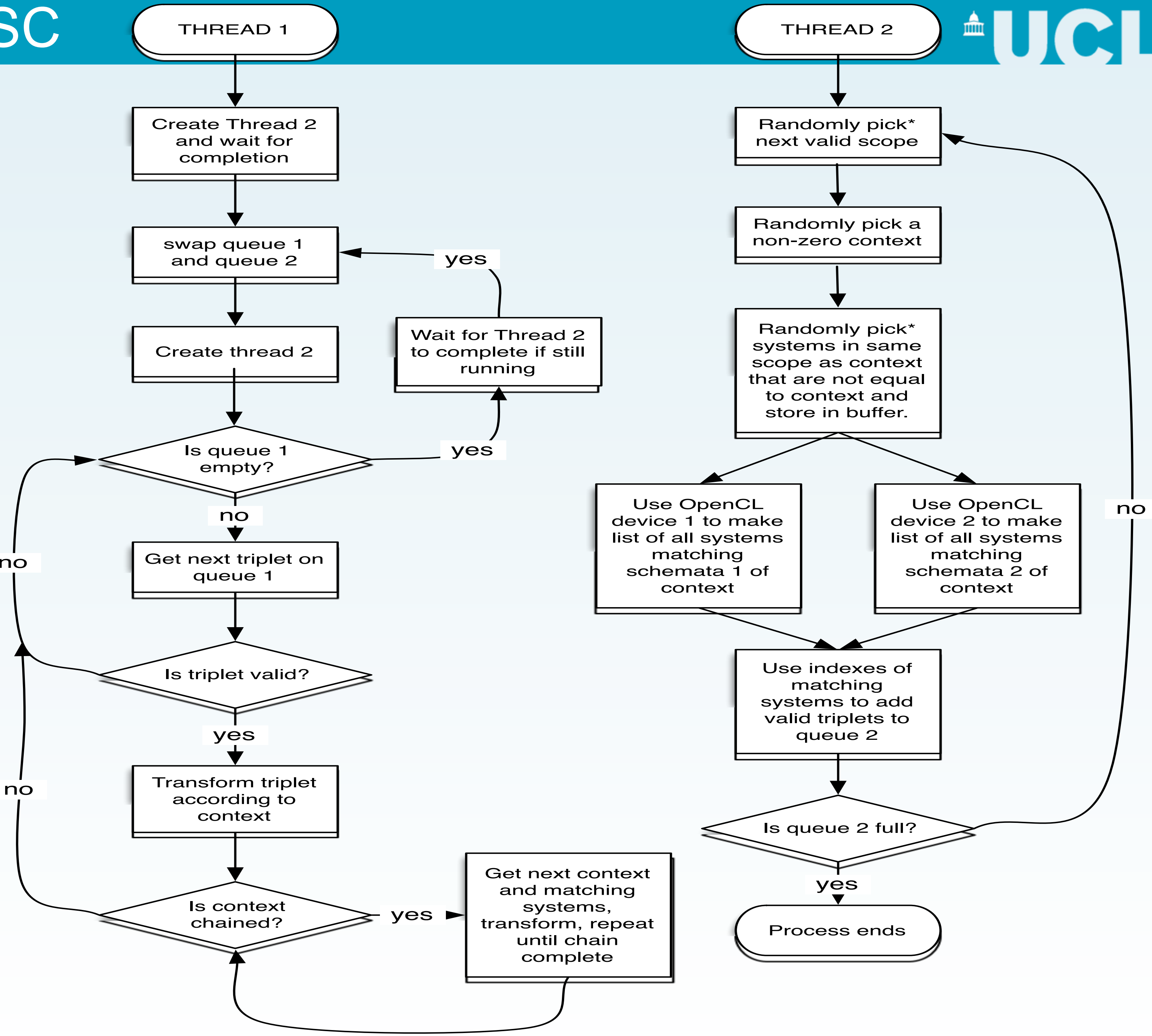
Solution 2: Novel hardware.
Much faster, but limited!

Solution 3: Use existing parallel architectures.

Multicore CPU and GPU architectures



GPU SC


THREAD 1
Create Thread 2 and wait for completion
swap queue 1 and queue 2
Create thread 2
Wait for Thread 2 to complete if still running
yes
yes
Is queue 1 empty?
no
Get next triplet on queue 1
no
Is triplet valid?
yes
Transform triplet according to context
no
Is context chained?
yes
Get next context and matching systems, transform, repeat until chain complete
THREAD 2
Randomly pick* next valid scope
Randomly pick a non-zero context
Randomly pick* systems in same scope as context that are not equal to context and store in buffer.
Use OpenCL device 1 to make list of all systems matching schemata 1 of context
Use OpenCL device 2 to make list of all systems matching schemata 2 of context
Use indexes of matching systems to add valid triplets to queue 2
Is queue 2 full?
no
yes
Process ends

* Random cycle algorithm

```
12   32   12   34   44   12   44   01   11   06
```

Randomly pick a value from the list and repeat without choosing the same value, until all have been chosen.

```
randomcycle(a, range)

a = (a + BIGPRIME) % (range)
```

Performance comparison

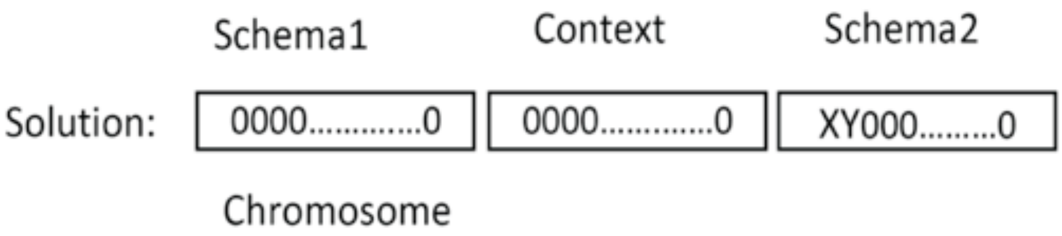
Genetic Algorithm Optimization of Binary Knapsack

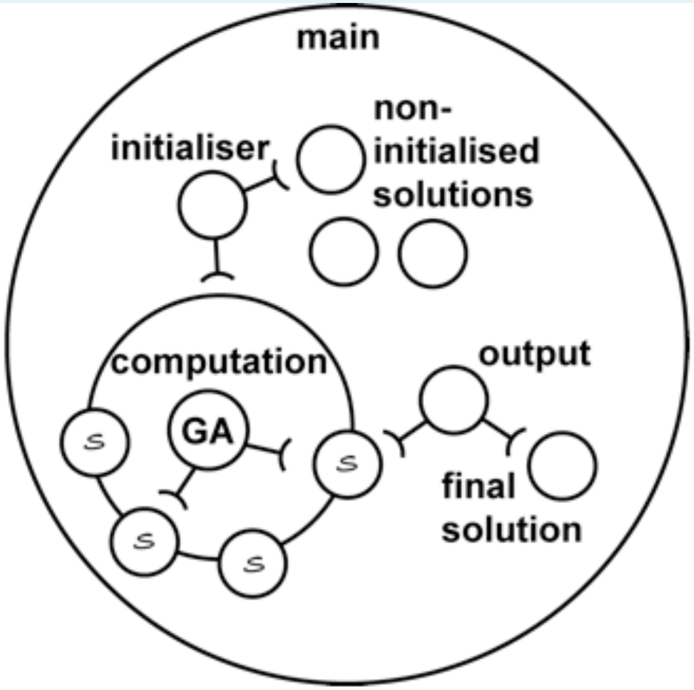
In the knapsack problem there are *n* objects with value $v_i > 0$ and weight $w_i > 0$. We want to find a set of objects with maximum total weight that fits into a knapsack with capacity *C*. Thus, we wish to maximize:

$$\sum_{i=1}^{n} v_i x_i \text{ where } \sum_{i=1}^{n} w_i x_i \leq C \text{ and } x_i \in \{0,1\}$$

Here, we use a Genetic Algorithm in SC. There are three different solutions: uninitialized solutions, initialized solutions, and final solutions. The chromosome size equals the schema size (16 in this implementation). So, this program supports a knapsack with 16 objects.

SC GA knapsack program





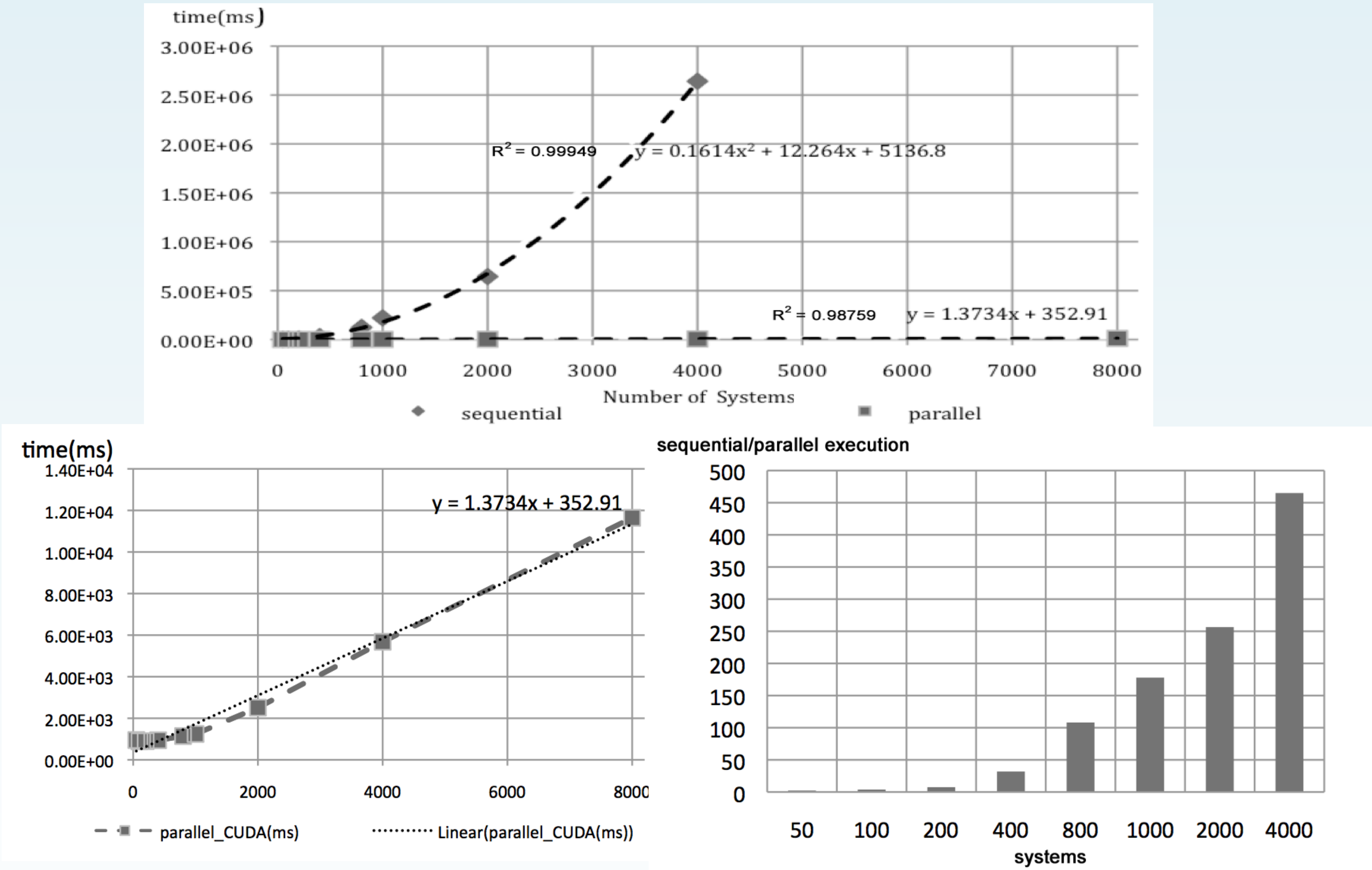
Left: The Solution system *S*. Schema1 stores the chromosome. **XY** in Schema2 specifies solution type (00: non initialized, 11: final solution). **Right:** the systemic program (not all non-initialised solutions an GA systems shown).

Experiments

CPU	Intel® dual core™, 2.40 GHz
RAM	2 GB
OS	Microsoft Windows XP professional 2002 SP1
GPU	Name: GeForce 9800 GT CUDA: 1.1 Size of Global memory: 1 GB Multiprocessors 14 Number of cores: 112 Clock Rate: 1.62 GHz

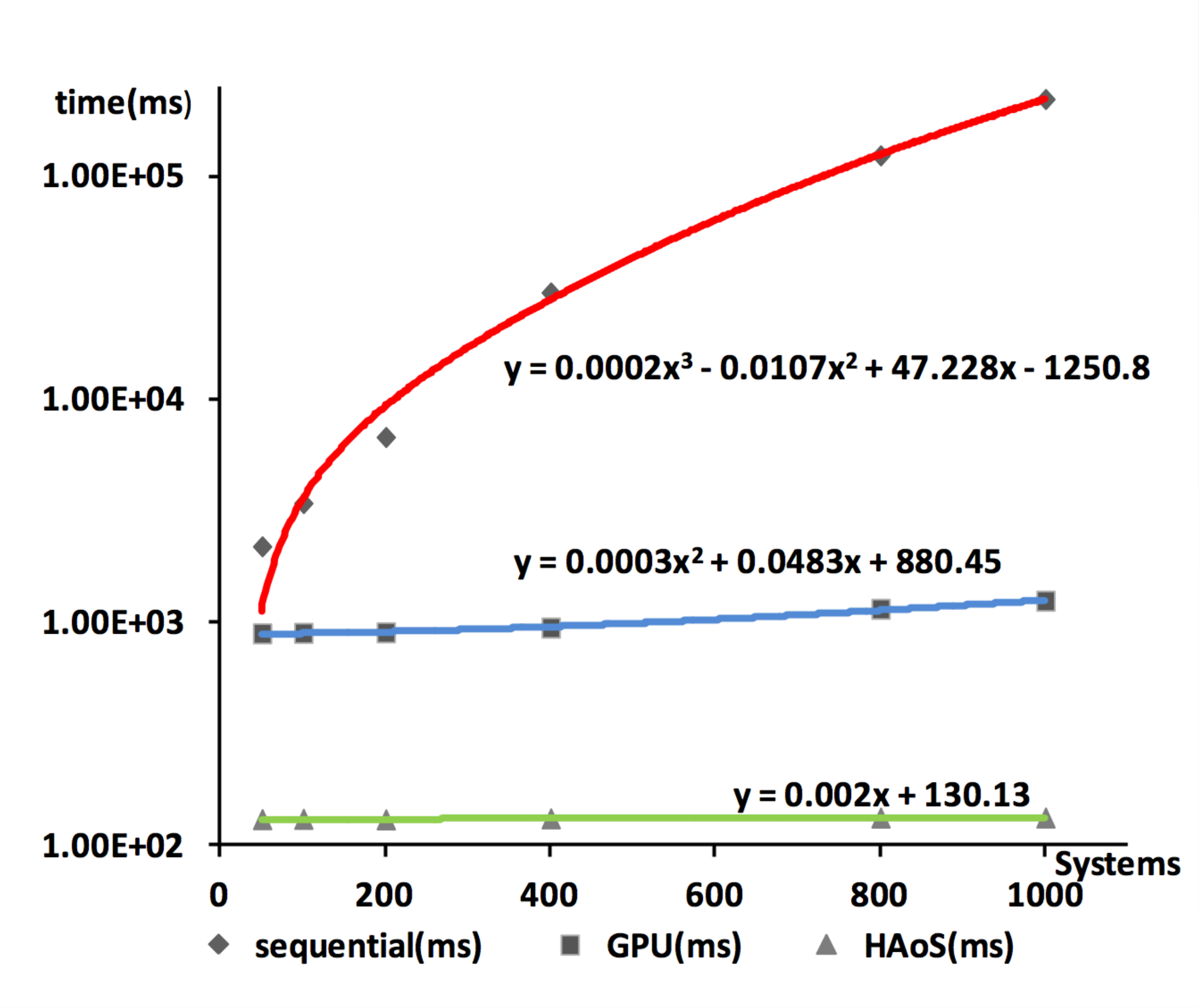
- Number of knapsack objects is 16; the maximum knapsack's weight is 80.0 kg.
- Context systems: 3 GA systems and 1 output system
- Solution systems: 50 to 4000 systems for sequential implementation and 50 to 8000 systems for parallel implementation (each increment is double the previous increment except 800 to 1000 with an increment of 200)
- Final Solution system: 1 system
- Scope: 1 main scope and 1 computation scope

Performance comparison (speed)



Top: Execution time of knapsack problem on both sequential and parallel implementation with increasing number of systems. **Bottom left:** execution time of parallel implementation alone. **Bottom right:** improvement as shown by sequential divided by parallel execution times for different numbers of systems in the program.

Performance comparison (speed)

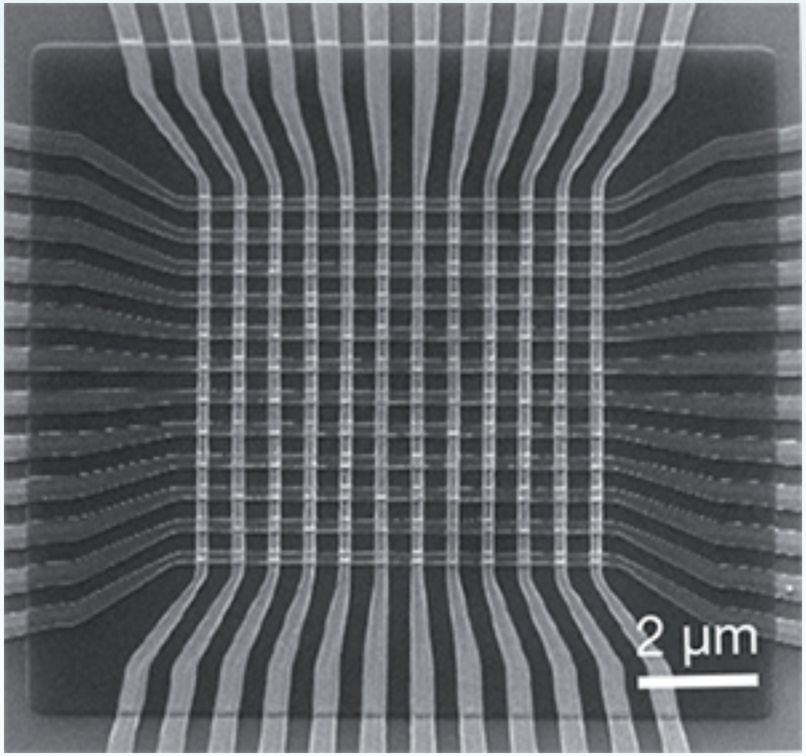


What else could we use?

CYPHER MATCH p = (:Category)<--(:Place)-[*]->(:Terminal {name:'A'}) RETURN p

Graph languages? (akin to Semantic Web)

What else could we use?



Memristor or neuromorphic chips?

What else could we use?

?

Something new?

Conclusions

- The future of computing, as predicted by von Neumann, will be parallel and distributed.
- Perhaps by learning lessons from nature we will be able to achieve this future with the efficiency and reliability of a living system.
- Biology and conventional technology are designed differently and may work very differently to each other.
- If we could learn to combine the advantages of existing technologies with those of natural systems, our capabilities would be transformed.

Conclusions

- Systemic computation is our solution. It is:
  - A model of computation
  - A common language for computer science and natural systems
  - A computer architecture
- It's possible to simulate the systemic computer, develop custom hardware, or use the latest parallel hardware solutions.
- The most practical and scalable solution today is to use the latest parallel hardware.
- We anticipate that the hardware of tomorrow will be even more suitable.

Thank You.

http://www.cs.ucl.ac.uk/staff/p.bentley/

http://www.peterjbentley.com/