In Pursuit of a New Kind of Computer

Programable Smart Machine Lab (PSML)
Jonathan Appavoo, Boston University

“And now for something completely different...”

PSML
Programmable
Smart Machines
Lab

Properties

Properties

- Simply and yet Richly Programmed

Properties

- Simply and yet Richly Programmed
- Automatically improves with its size

Properties

- Simply and yet Richly Programmed
- Automatically improves with its size
- Automatically improves/adapts with experience

Properties

- Simply and yet Richly Programmed
- Automatically improves with its size
- Automatically improves/adapts with experience
- Amenable to implementation with low power devices

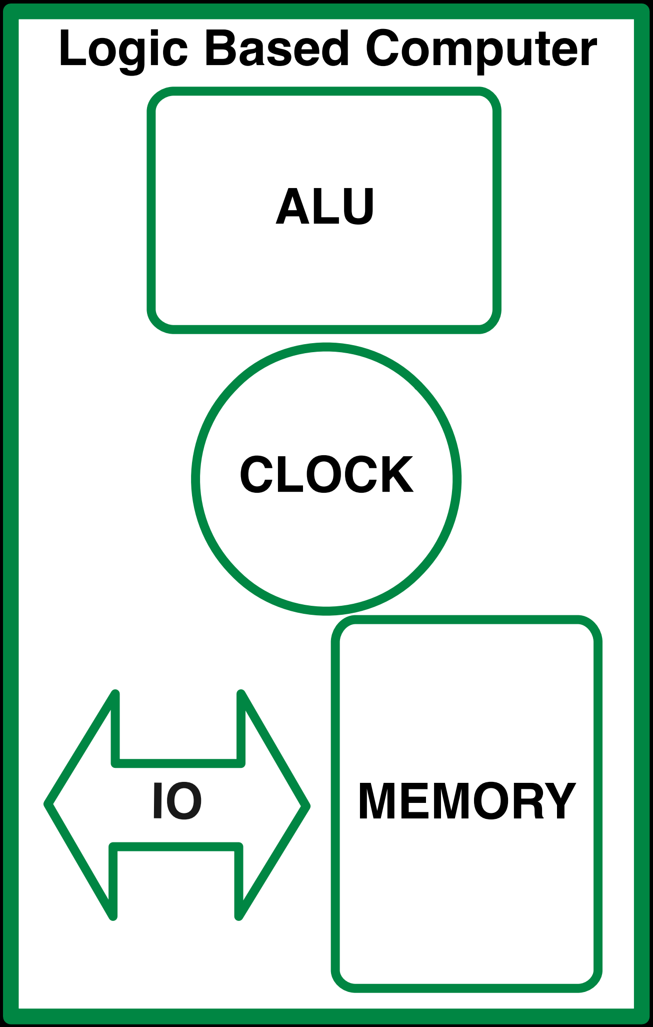
Properties

- Simply and yet Richly Programmed
- Automatically improves with its size
- Automatically improves/adapts with experience
- Amenable to implementation with low power devices

A programmable child that is pedantically obedient without the attitude :-)

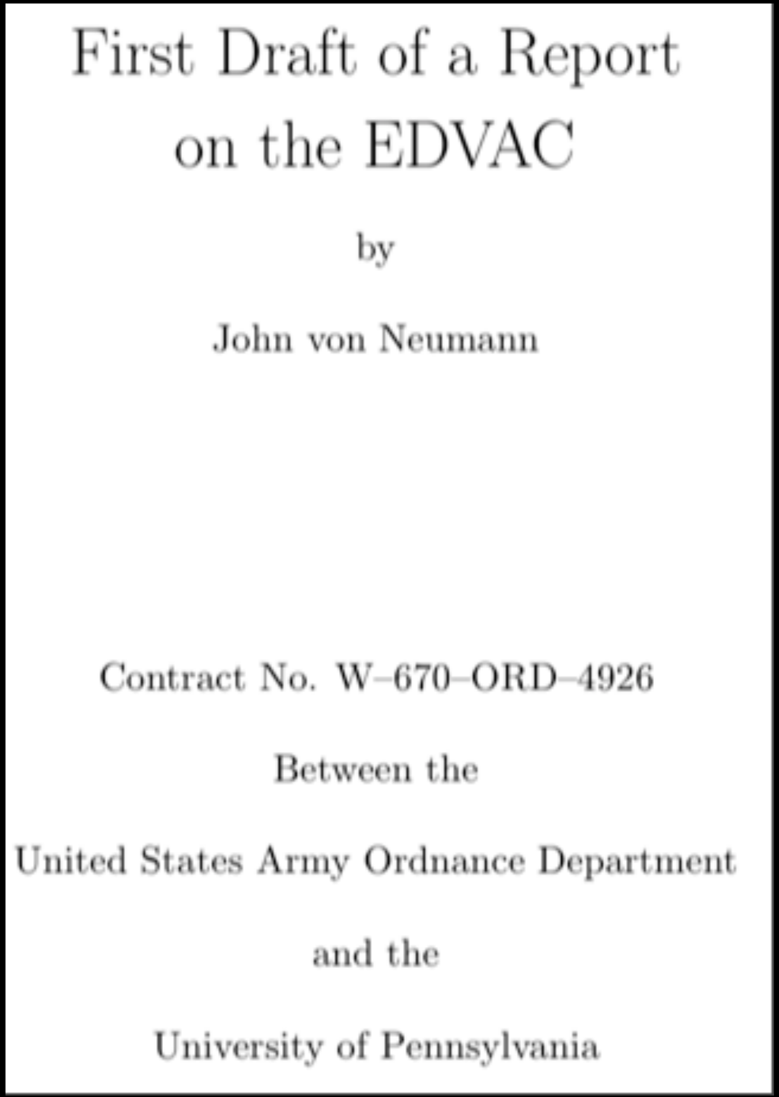
Simply and yet Richly Programmed

Let's not throw the baby out with the bath water



Simple synchronously clocked uni-processor system

- Easy to grok deterministic model
- General Purpose 'elastic'
- Easy to implement higher level SW machines



First Draft of a Report on the EDVAC

by

John von Neumann

Contract No. W–670–ORD–4926

Between the

United States Army Ordnance Department

and the

University of Pennsylvania

Hmmm Now What?

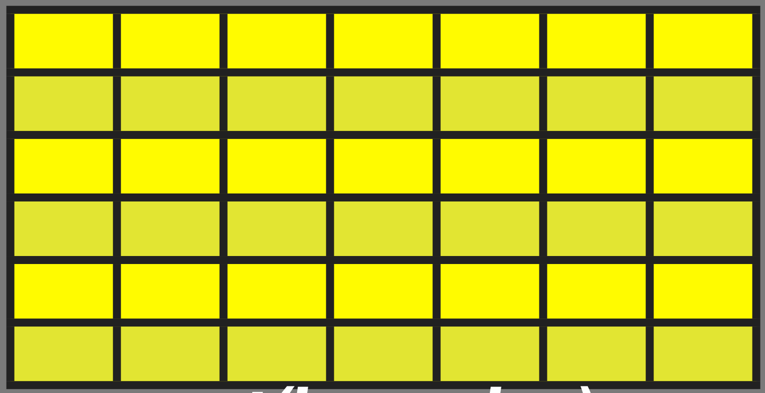
- ~~Simply and yet Richly Programmed~~
- Automatically improves with its size
- Automatically improves/adapts with experience
- Amenable to implementation with low power devices

Hmmm Now What?

- ~~Simply and yet Richly Programmed~~
- Automatically improves with its size
- Automatically improves/adapts with experience
- Amenable to implementation with low power devices

Using knowledge of runtime behavior introduce a cache based optimization:

hash table ***ht***



get(key,value)
put(key,value)

```
key = hf(s);
if (get(key,&value)==hit) {
   fast(value);
} else {
   slow(&value);
   put(key,value);
}
```

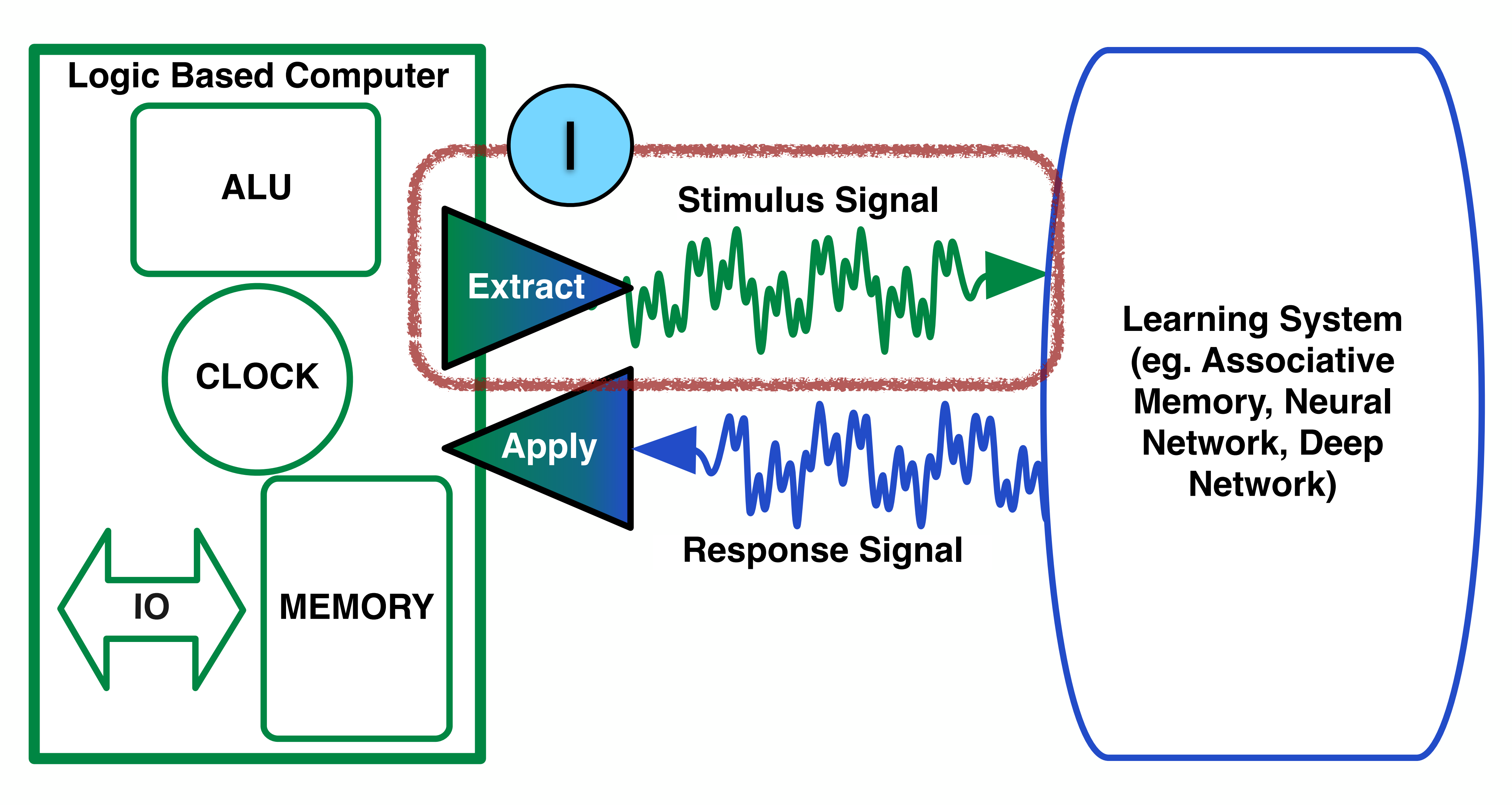
Hmmm Now What?

- ~~Simply and yet Richly Programmed~~
- Automatically improves with its size
- Automatically improves/adapts with experience
- Amenable to implementation with low power devices

Yikes...
I did warn you ;-)

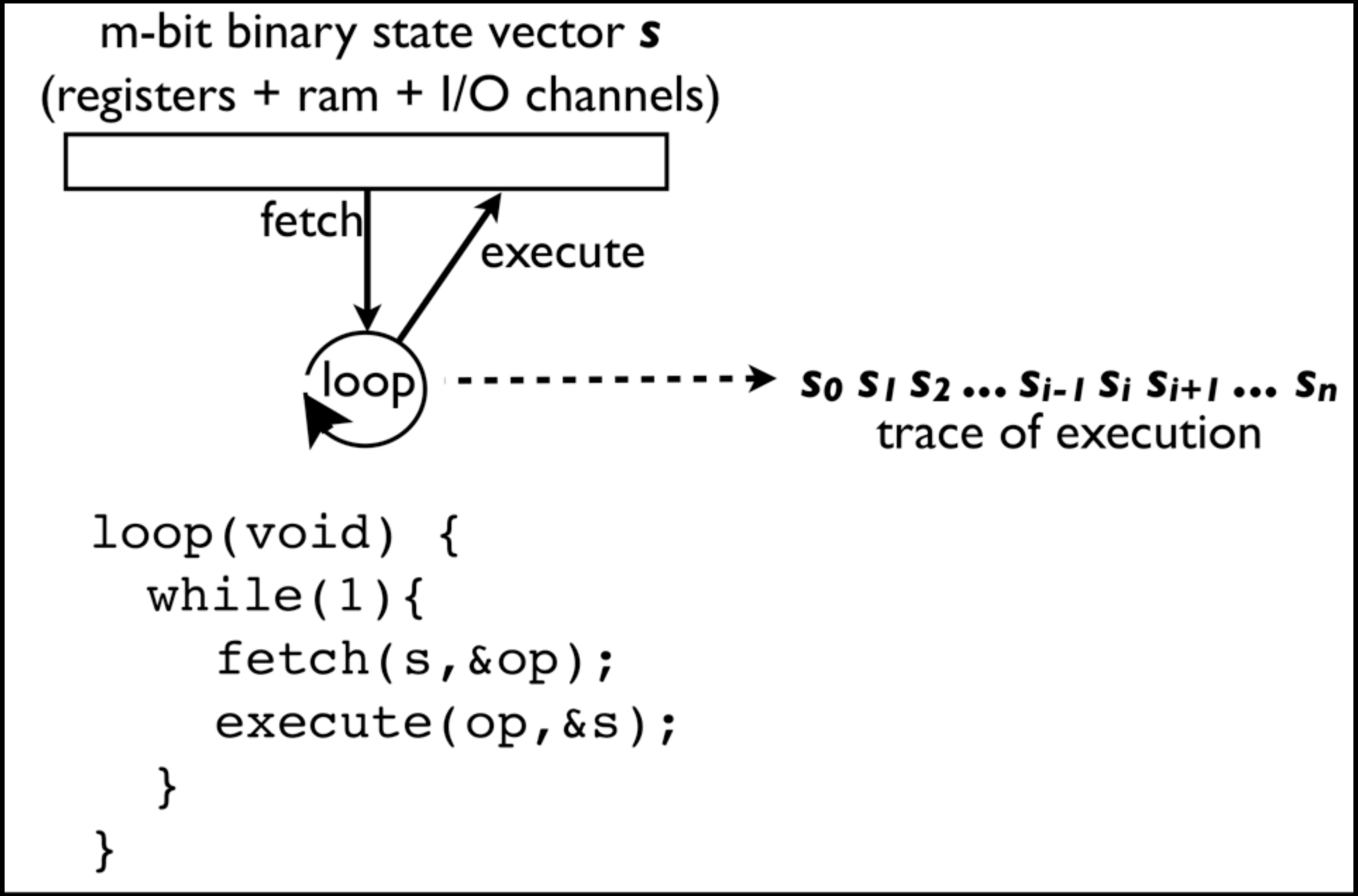
3 x 7

Need an Interface



Extract a Signal

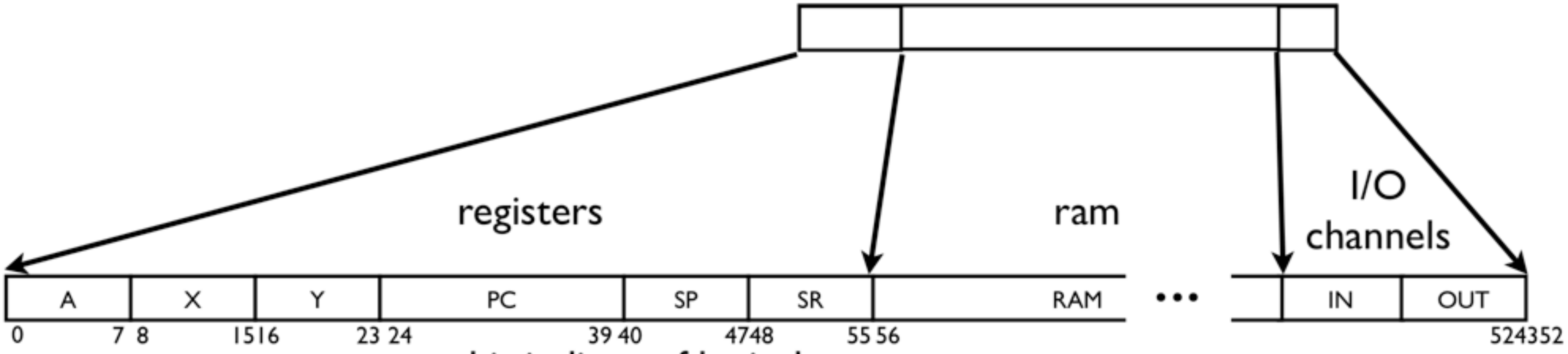
Execution Signal


m-bit binary state vector *s*
(registers + ram + I/O channels)
fetch
execute
loop
$s_0\ s_1\ s_2 \dots s_{i-1}\ s_i\ s_{i+1} \dots s_n$
trace of execution

```
loop(void) {
  while(1){
    fetch(s,&op);
    execute(op,&s);
  }
}
```

See our *HotPAR'12* Paper, "Parallelization by Simulated Tunneling", Waterland et al. For a more technical view of execution as state space traversal (Dynamical Systems Interpretation)

Simple 6502 Realization of ***s***
(registers + ram + I/O channels)



bit indices of logical system components

A 524352-bit binary vector ***s*** forms the entire 6502 system state.
Each step ***i*** of execution produces a vector $\boldsymbol{s_i}$.

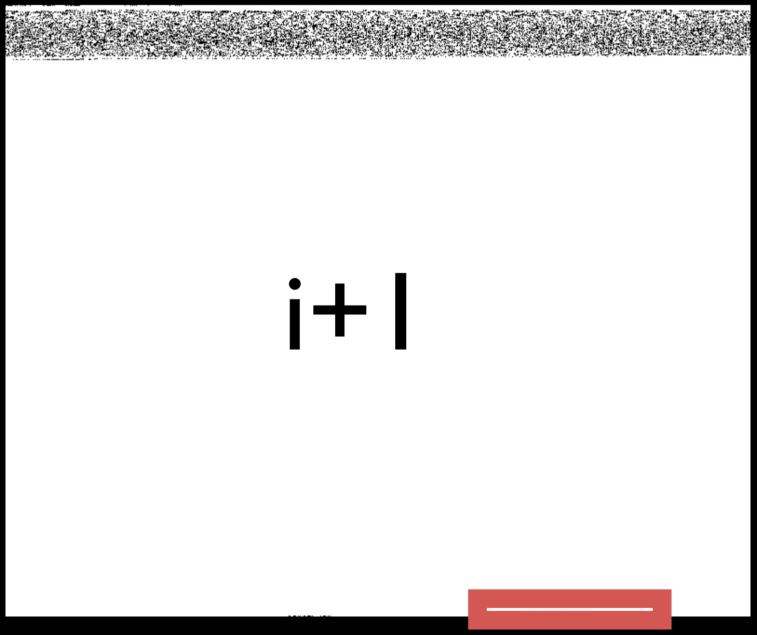
i

[illegible]

i
i+1

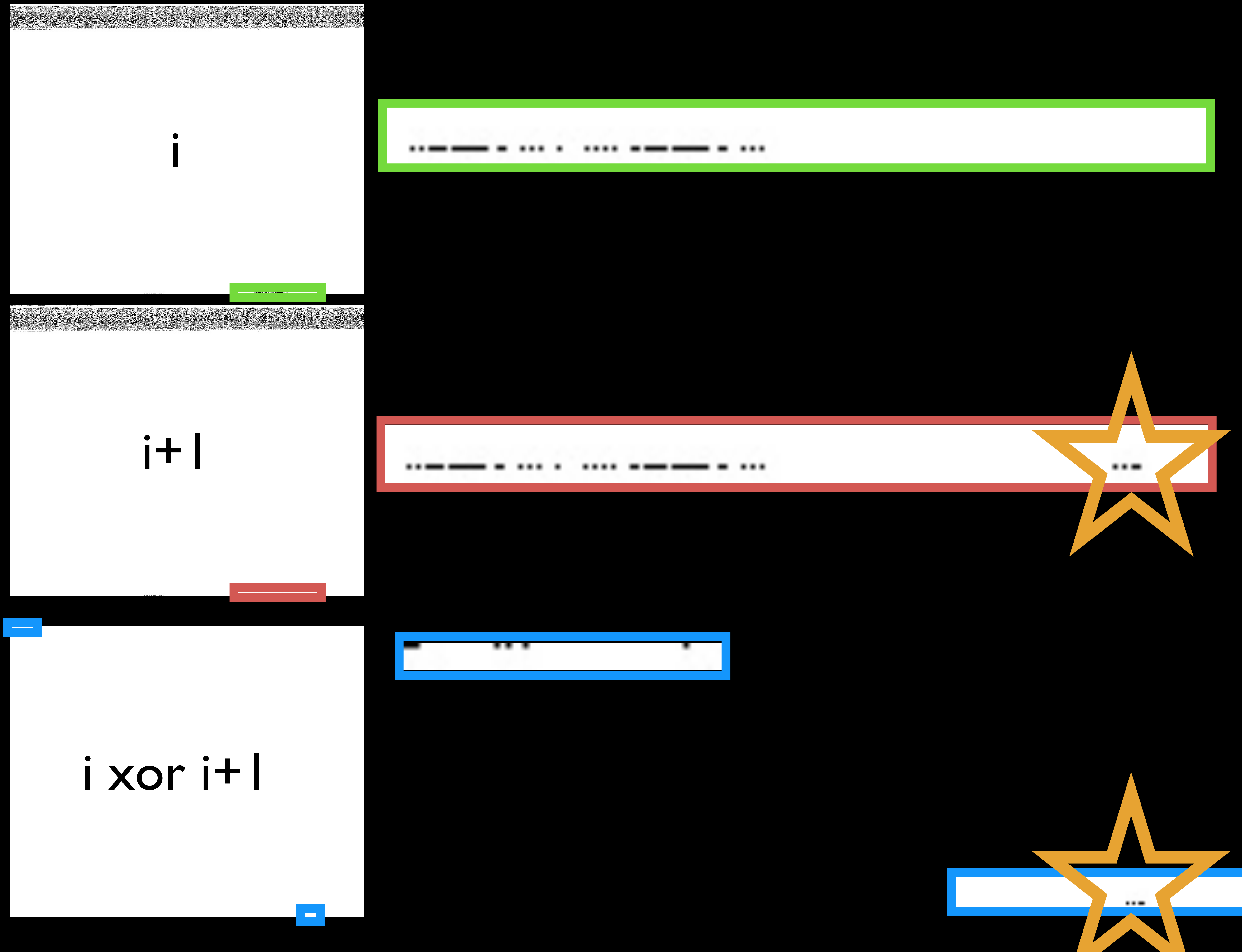
i

[illegible]



i+1

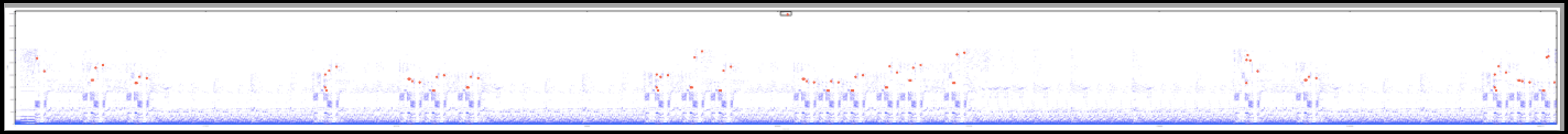
[illegible]

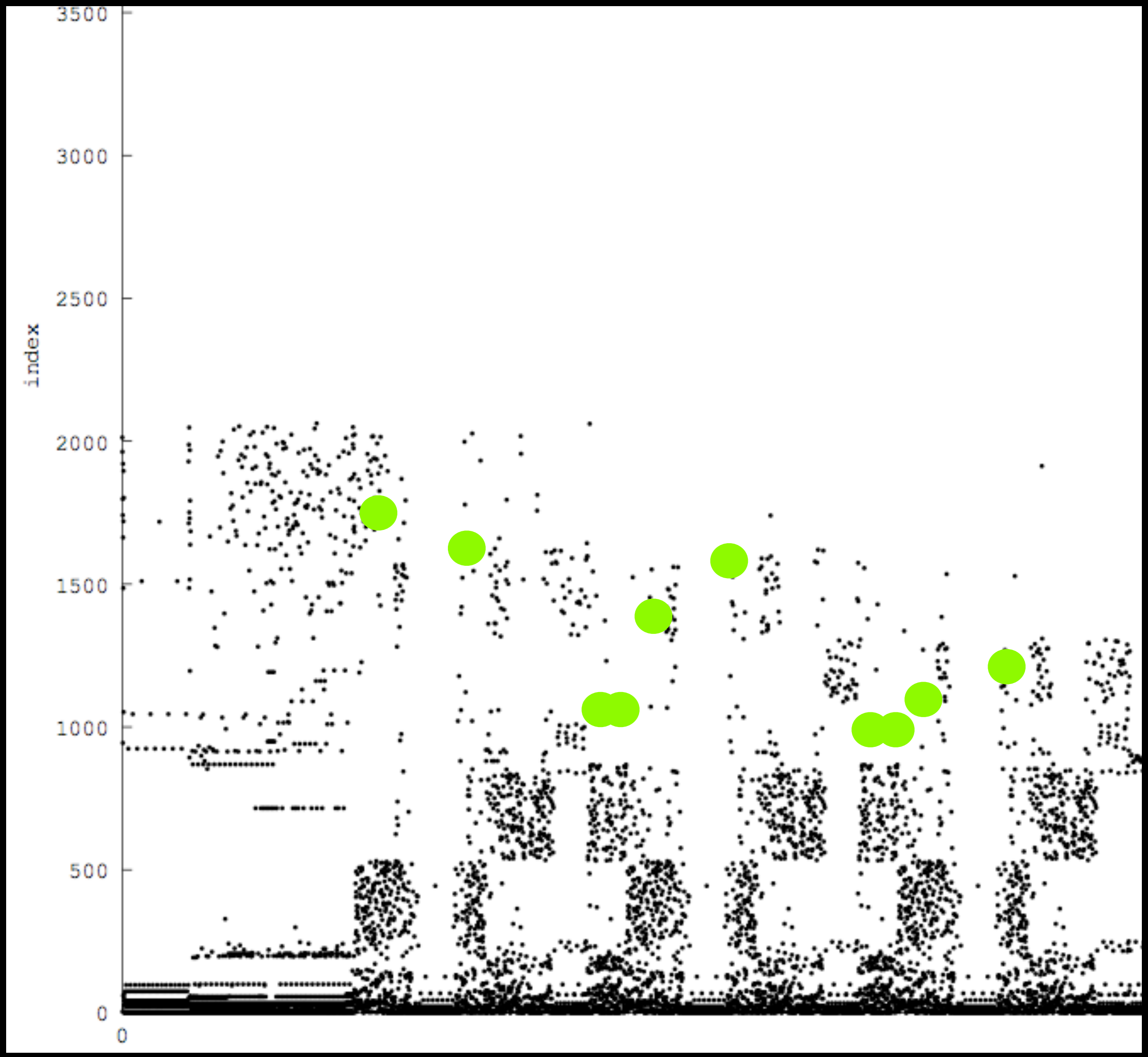


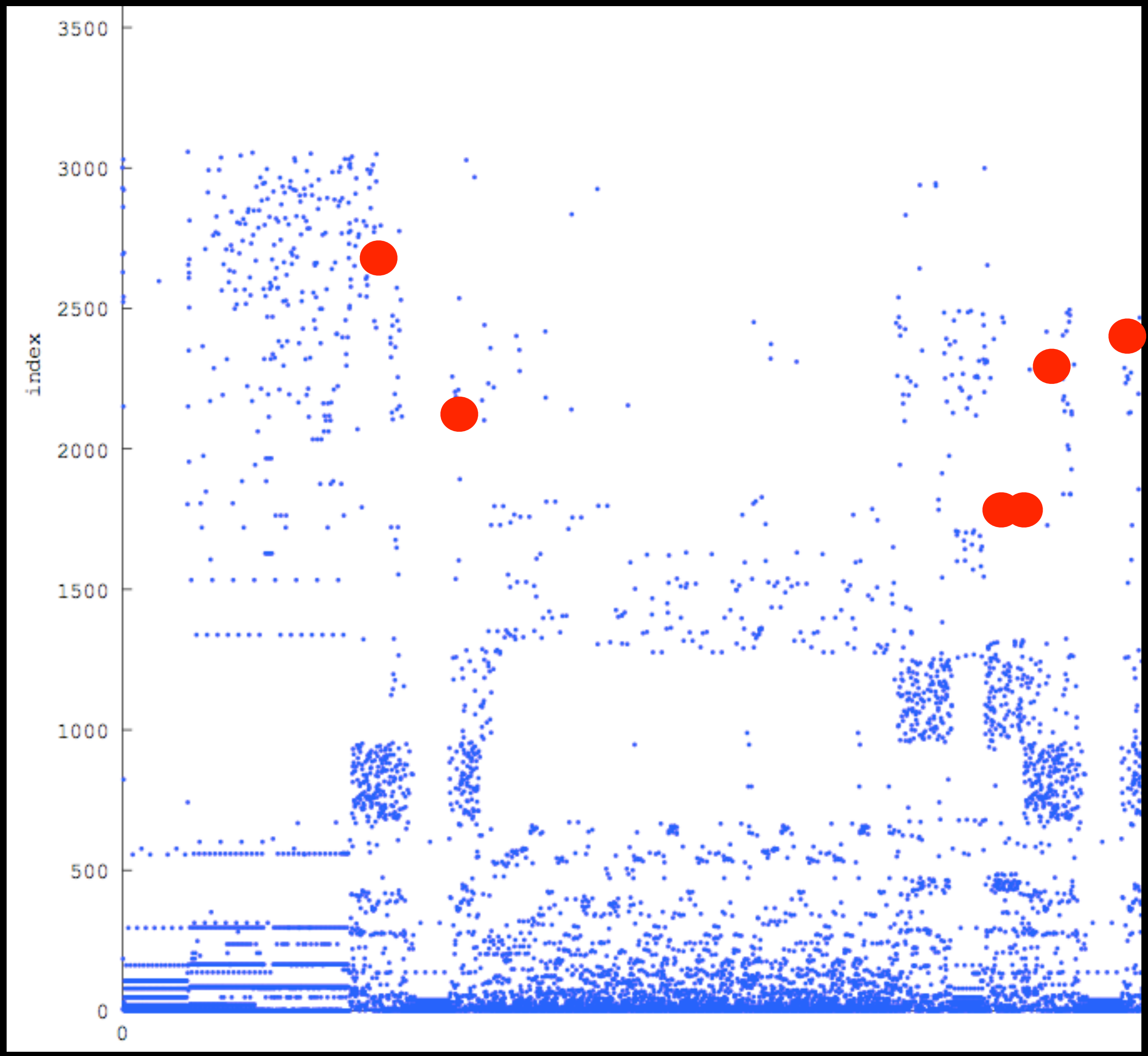
i

i+1

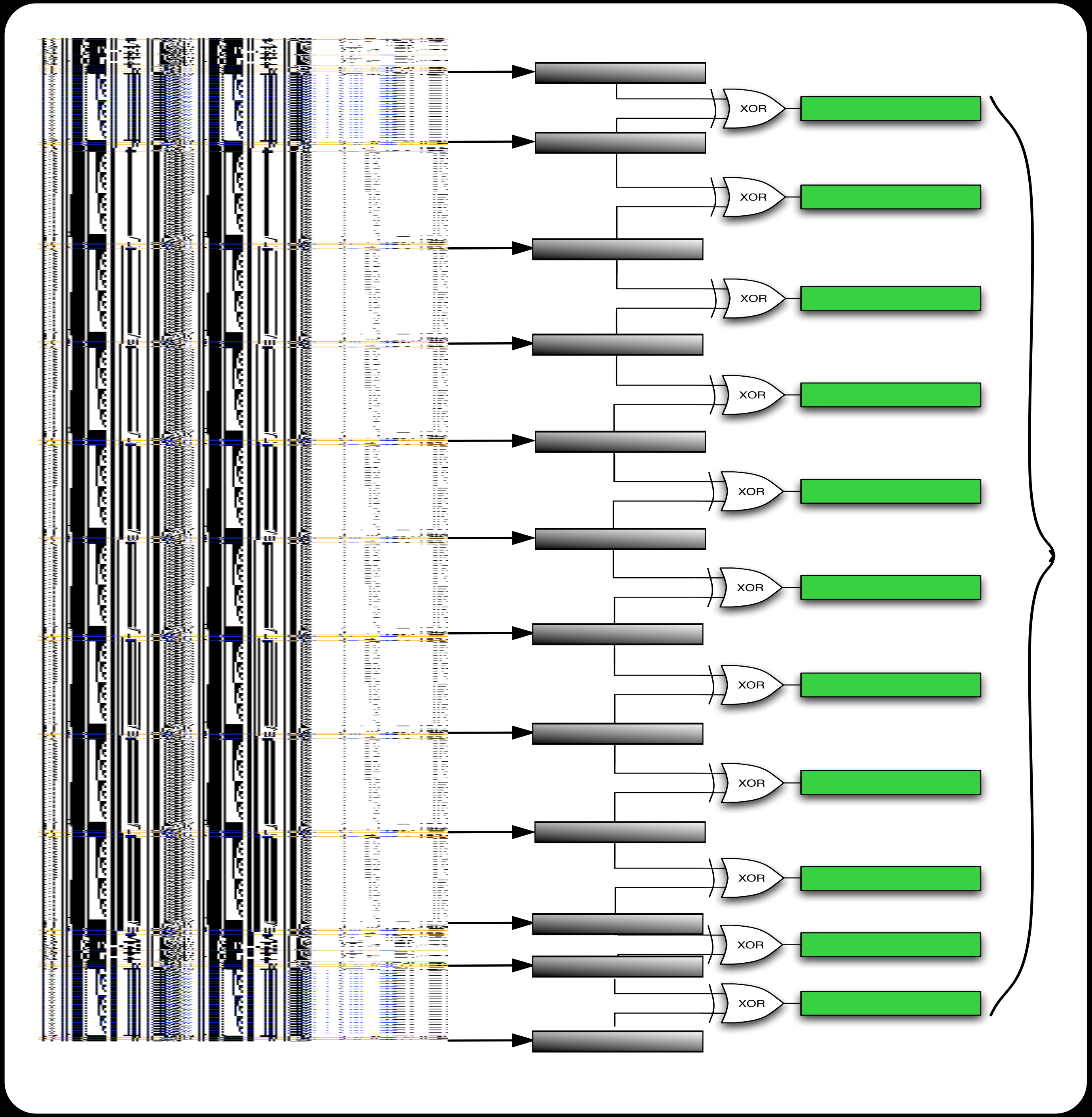
i xor i+1




3500
3000
2500
2000
1500
1000
500
0
index
0

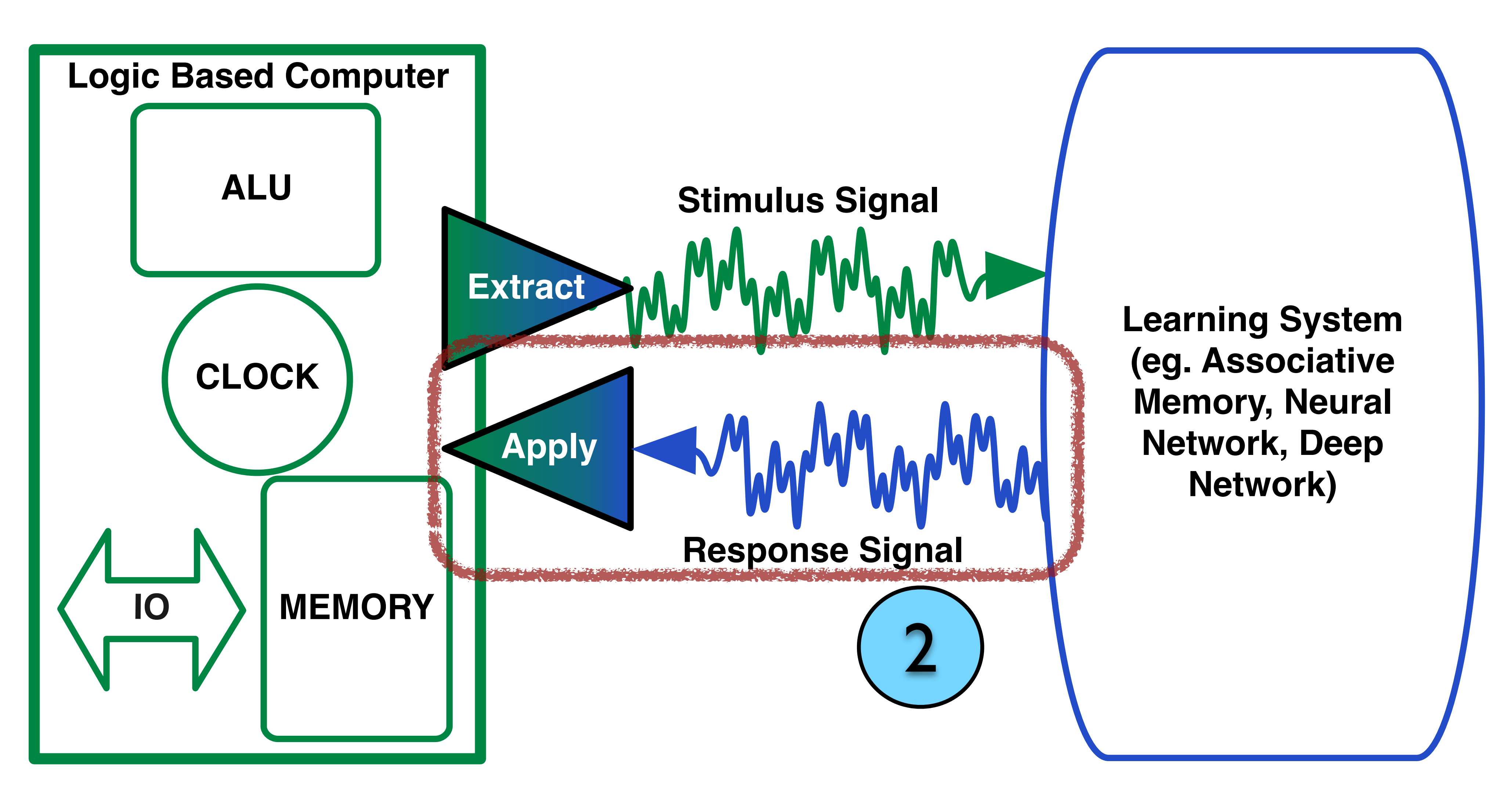

3500
3000
2500
2000
1500
1000
500
0
index
0

Intriguing Possibilities



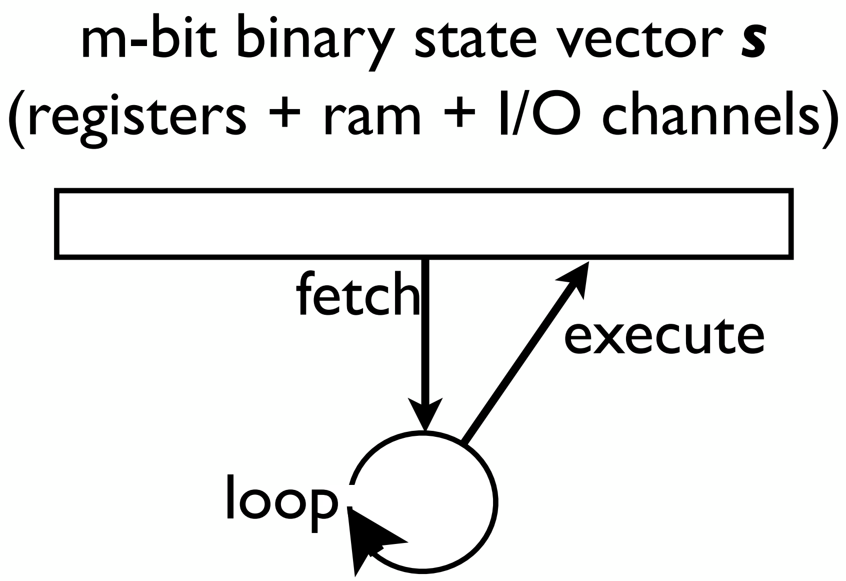
- “Derivatives” expose fascinating structure
- Simple to construct and study filters
- Unified statistical representation exposes unexpected patterns across all of system execution
- Opportunities for studies are proving to be amazingly fun and challenging our intuitions

Need a Method



How can we Apply the Response

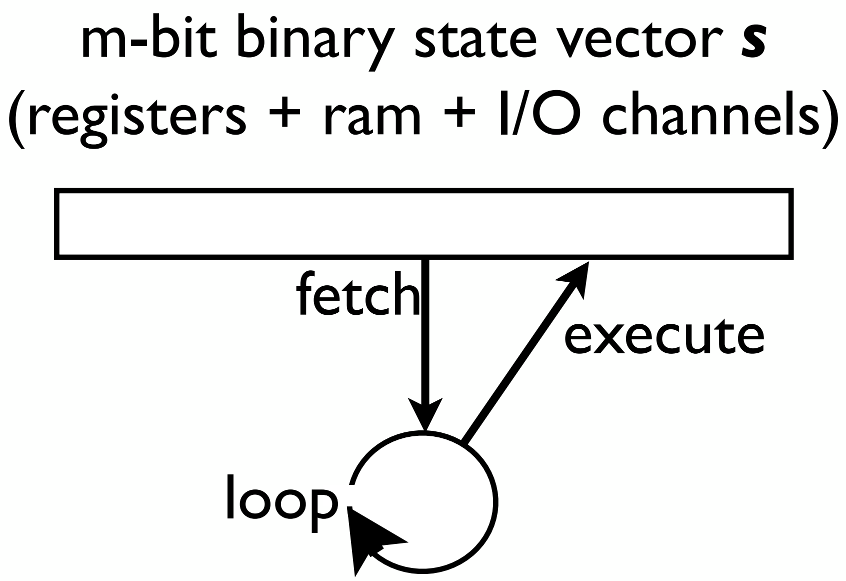
Back to the Loop


m-bit binary state vector *s*
(registers + ram + I/O channels)
fetch
execute
loop

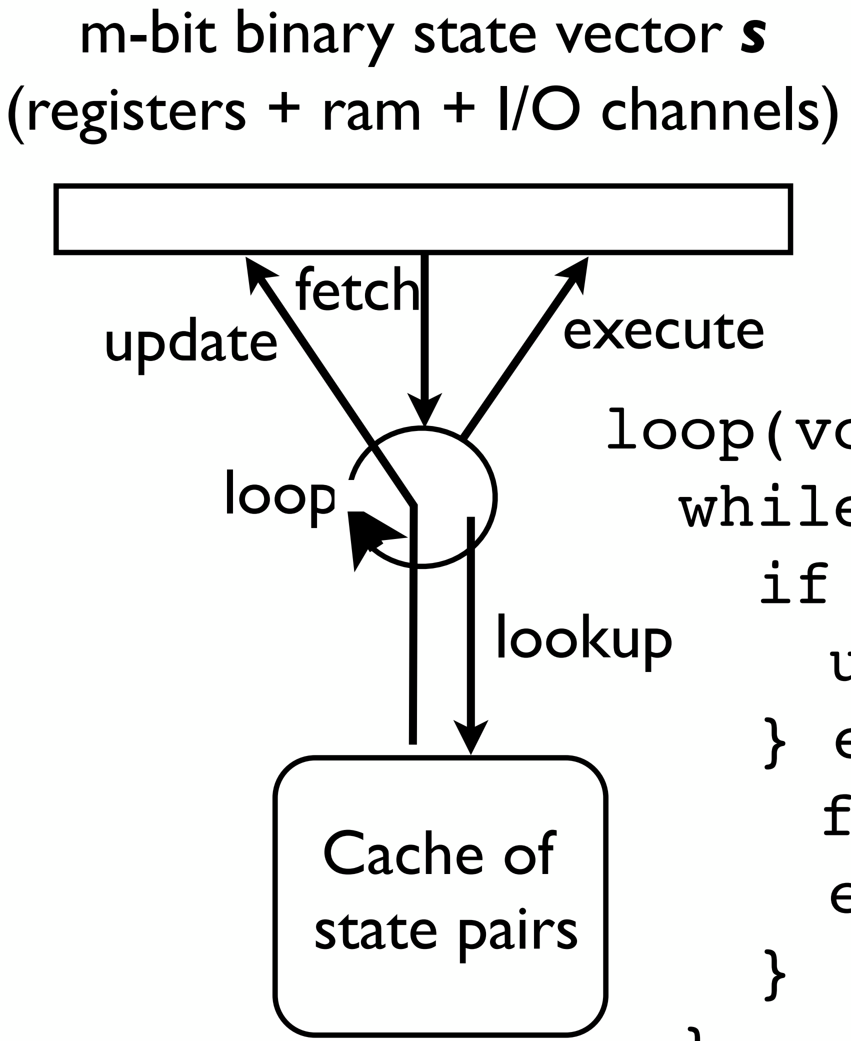
```
loop(void) {
  while(1){
    fetch(s,&op);
    execute(op,&s);
  }
}
```

State Pairs are computation (Again See our *HotPAR'12* Paper)

Back to the Loop


m-bit binary state vector *s*
(registers + ram + I/O channels)
fetch
execute
loop

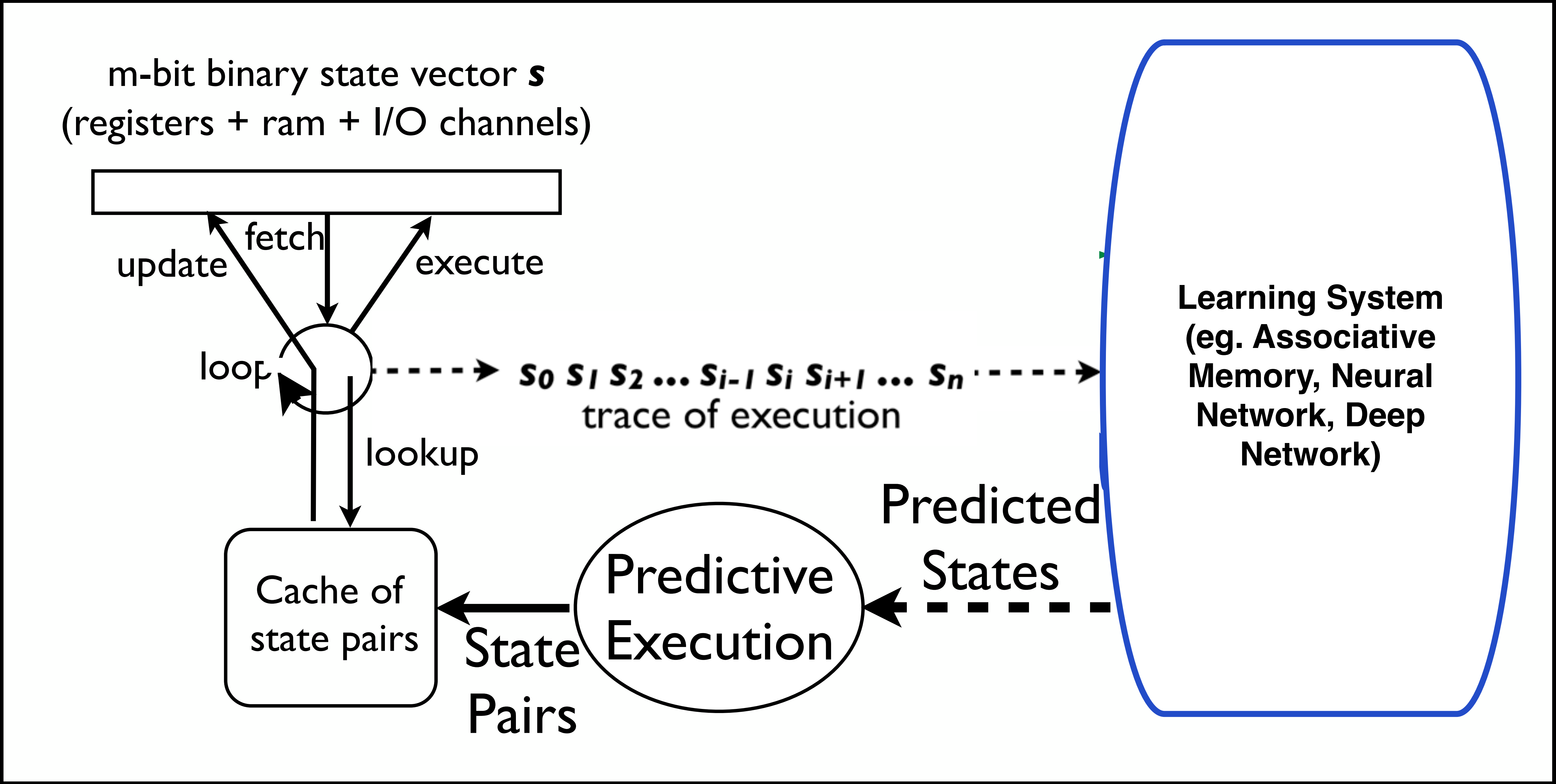
```
loop(void) {
  while(1){
     fetch(s,&op);
     execute(op,&s);
  }
}
```


m-bit binary state vector *s*
(registers + ram + I/O channels)
fetch
update
execute
loop
lookup
Cache of state pairs

```
loop(void) {
  while (1){
    if (lookup(s,&d)==hit){
      update(d,&s);
    } else {
      fetch(s,&op);
      execute(op,&s);
    }
  }
}
```

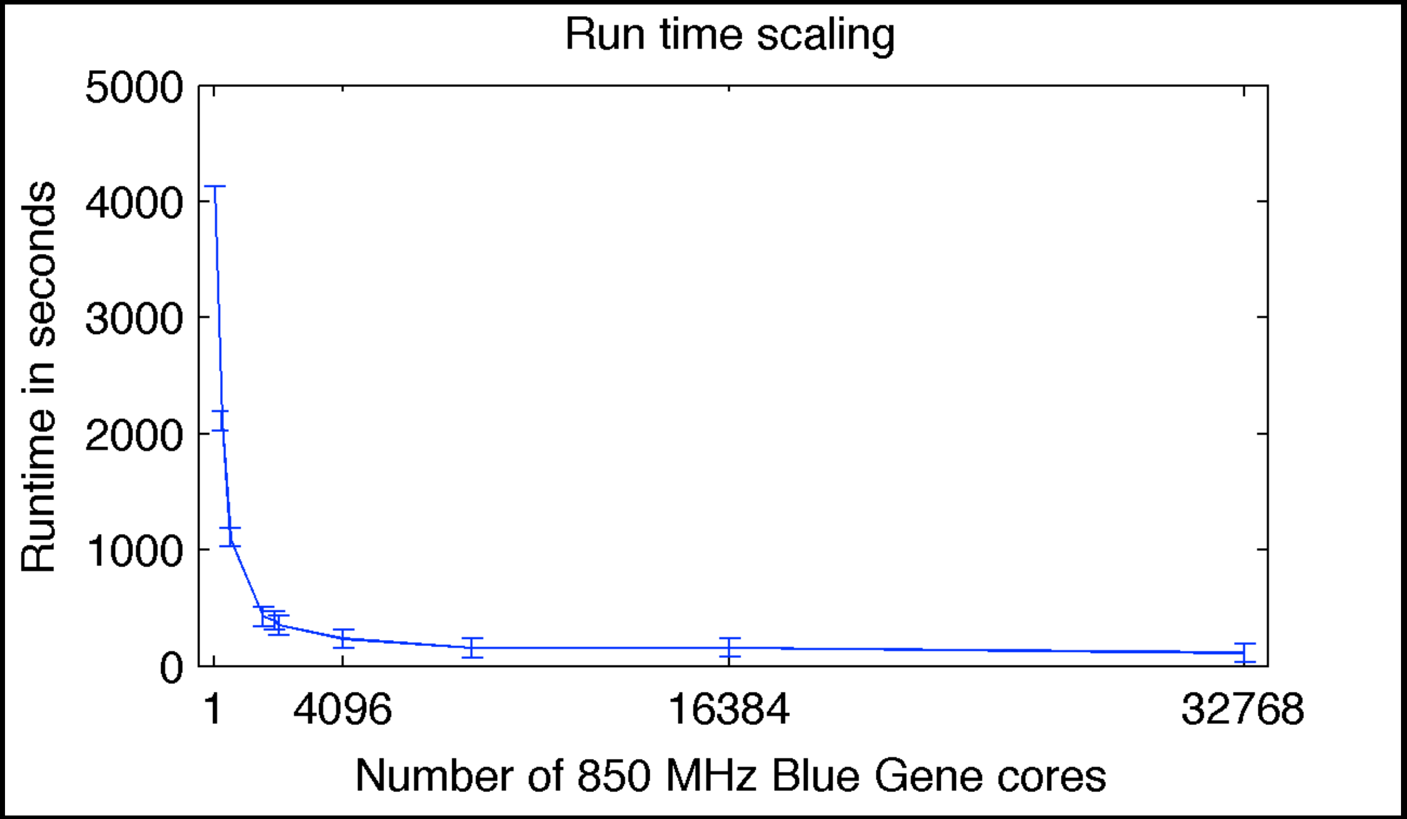
State Pairs are computation (Again See our *HotPAR'12* Paper)

Putting it together


m-bit binary state vector *s*
(registers + ram + I/O channels)
update
fetch
execute
loop
lookup
$s_0\ s_1\ s_2 \dots s_{i-1}\ s_i\ s_{i+1} \dots s_n$
trace of execution
Learning System (eg. Associative Memory, Neural Network, Deep Network)
Predicted States
Predictive Execution
State Pairs
Cache of state pairs

Sorry Animated gif broken in PDF VERSION

Hints of Smoke



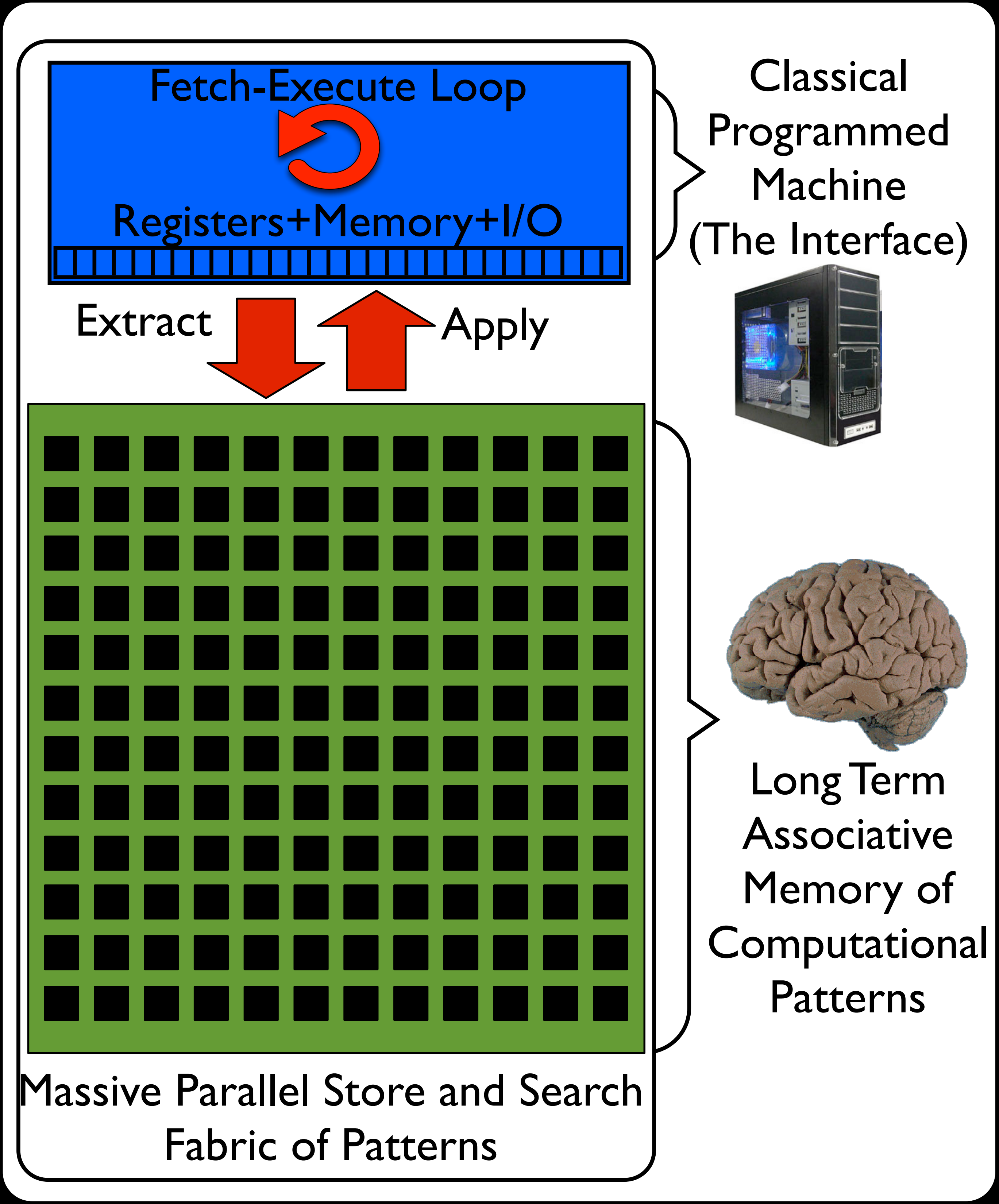
- Deterministic computation
- All I/O up front
- Restricted x86 simulator
- MPI on Blue Gene
- Simple learning hypothesis
- Simple Bayesian predictor

Initial results are documented again in our *HotPAR'12* Paper.

Computational Battery

- Split things up into two distinct phases a learning problem (charging) and later use mode
- train ML model on large scale systems and store model along with initial DB of state pairs
- VMs uses this along with local cores to accelerate computation

All Done the Talk ... Work is Just Starting


Fetch-Execute Loop
Registers+Memory+I/O
Extract
Apply
Classical Programmed Machine (The Interface)
Long Term Associative Memory of Computational Patterns
Massive Parallel Store and Search Fabric of Patterns

- These are all just first steps and all very rough
- But wow a ton of fun!
- Able to apply and explore fascinating relationships between classical logic and statical mechanisms
- Amos and I have been growing the set of crazies (from complexity, information theory, physics, mathematics, and HW)
- Thanks to them all: Margo Seltzer, Steve Homer, and all the brave and excellent students that have joined Amos: Katherine Zhao, Elaine Angelino, and others.