Erlang: The Unintentional Neural Network Programming Language

Erlang Factory
San Francisco – March 2011

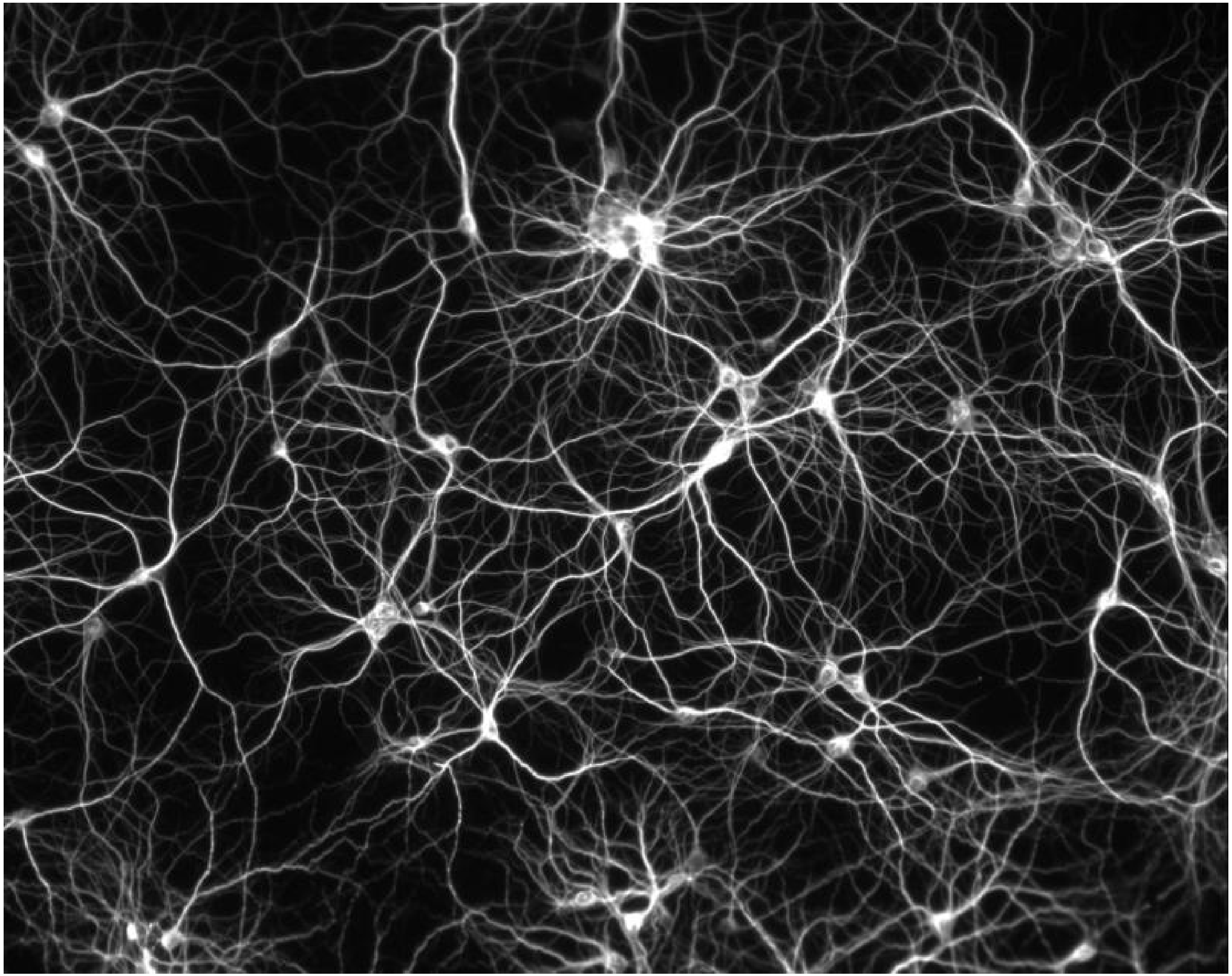
Gene Sher

Introduction

- The Long Standing Goals of Computational Intelligence
- Brain is just an organic substrate based NN
  - Blue Brain Project - http://bluebrain.epfl.ch/
- Computer hardware is advancing steadily
- A programming language is needed with the right features
- Erlang: As the NN Programming Language

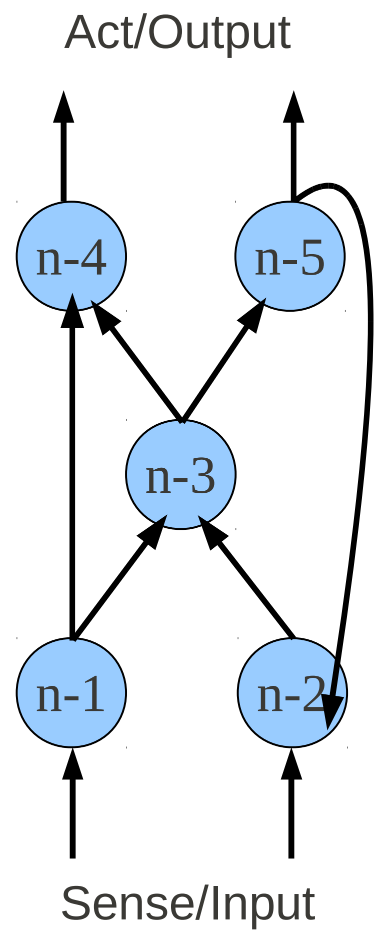
Computational Intelligence Through Genetic Algorithms and Neural Networks

Biological Neural Network

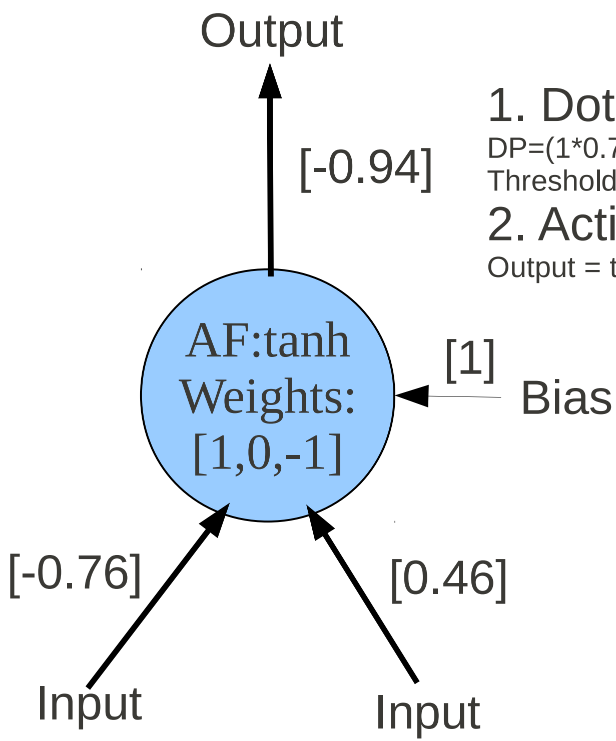


Artificial Neural Networks

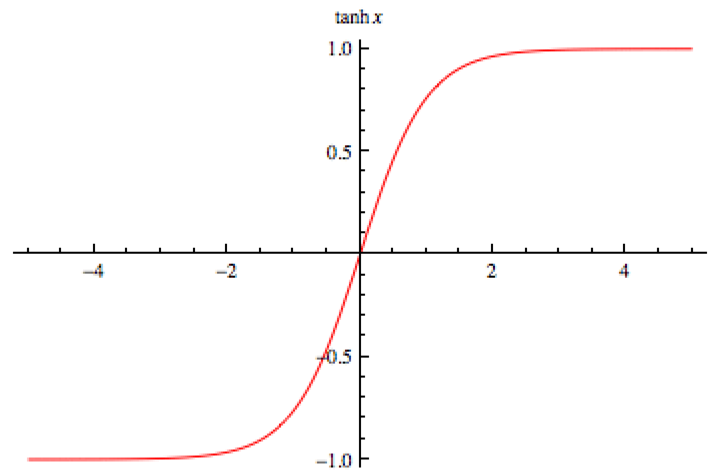
- Simulate biological NNs to various degrees of precision
- Directed graphs
- Parallel
- Learn, adapt, and generalize



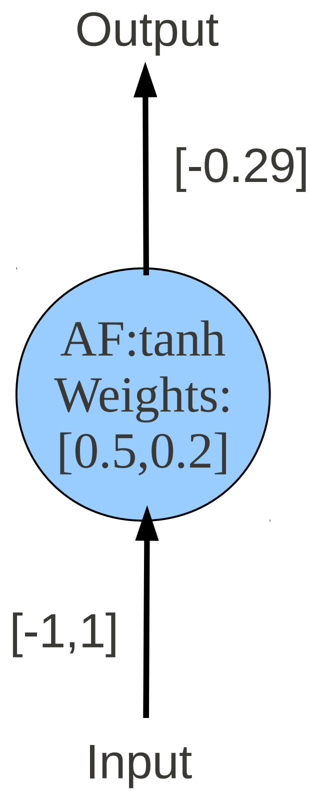
An Artificial Neuron



1. Dot product:
DP=(1*0.76) + (0*0.46)
Threshold = (-1*1)
2. Activation strength:
Output = tanh(DP+Threshold)

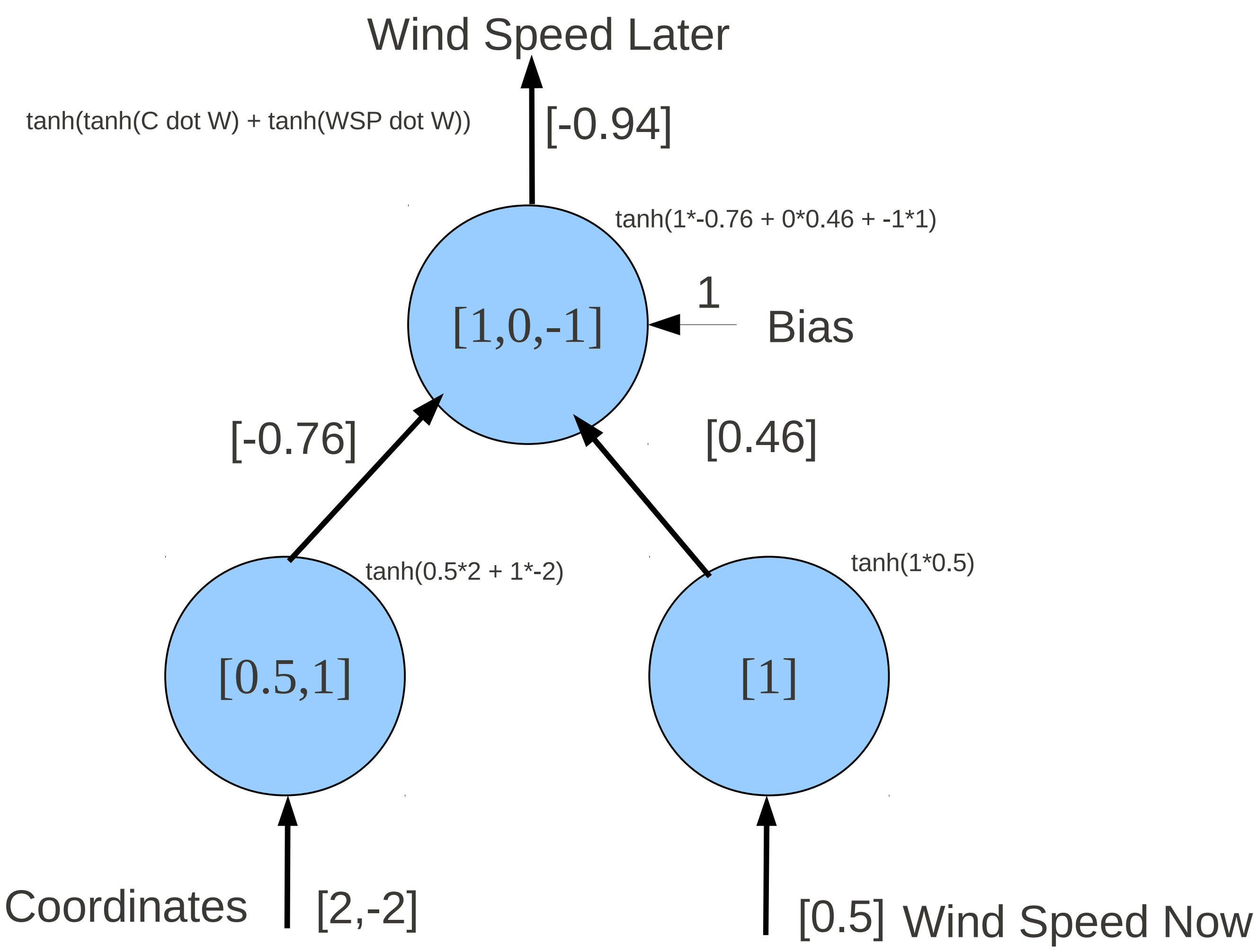


The Input is Just a Vector



1. Dot product:
DP=(0.5*-1) + (0.2*1)
Threshold = (0*1)
2. Activation strength:
Output = tanh(DP+Threshold)

A Neural Network

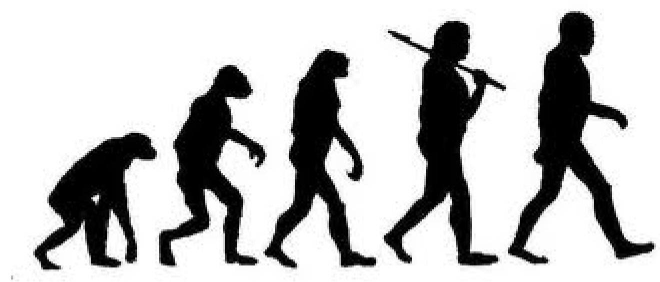


NN Learning & Plasticity Algorithms

- Supervised
  - Backpropagation
  - ...
- Unsupervised
  - Kohonan (Self-organizing) map
  - Adaptive Resonance Theory
  - Hebbian
    - "The general idea is an old one, that any two cells or systems of cells that are repeatedly active at the same time will tend to become 'associated', so that activity in one facilitates activity in the other." (Hebb 1949, p. 70)
  - Modulated
  - Evolutionary
  - ….

Evolutionary Computation

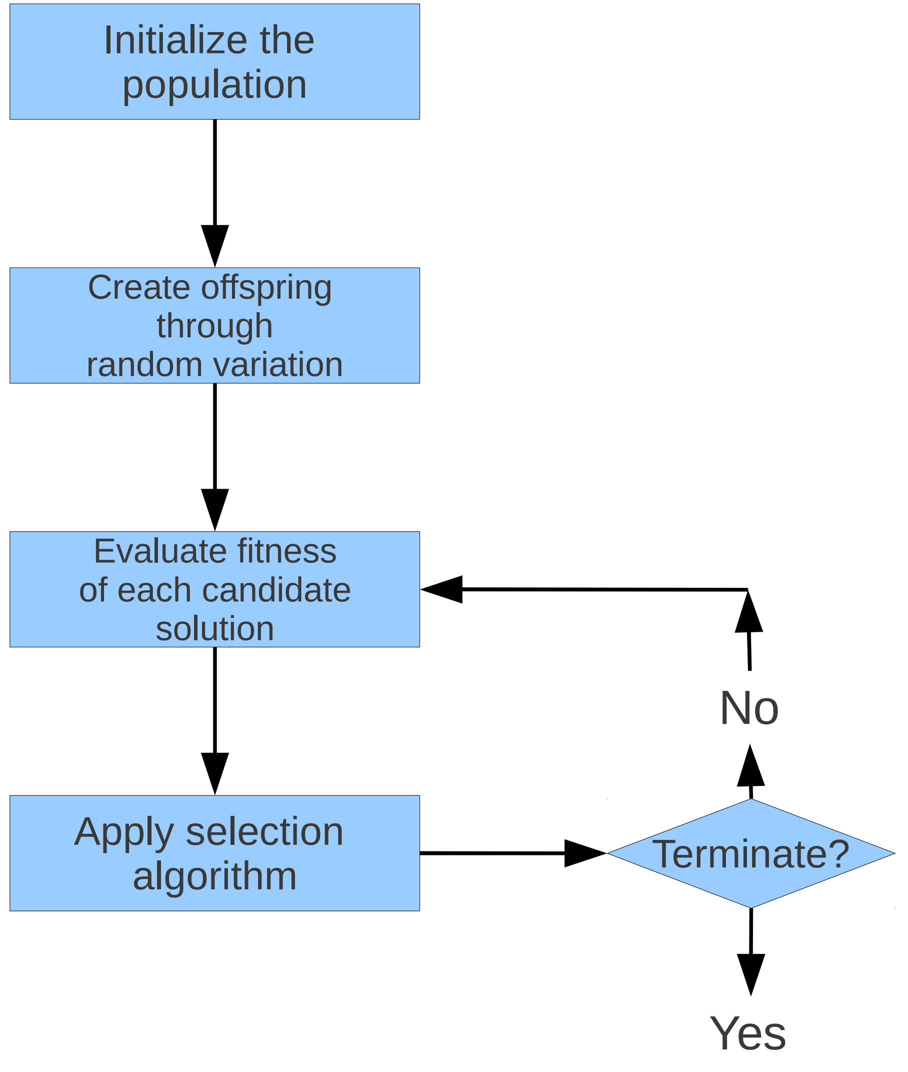
- Based on evolutionary principles
- Stochastic search with a purpose
    - Survival of the fittest
- Genotype to Phenotype
- Mutation and crossover



George E. P. Box

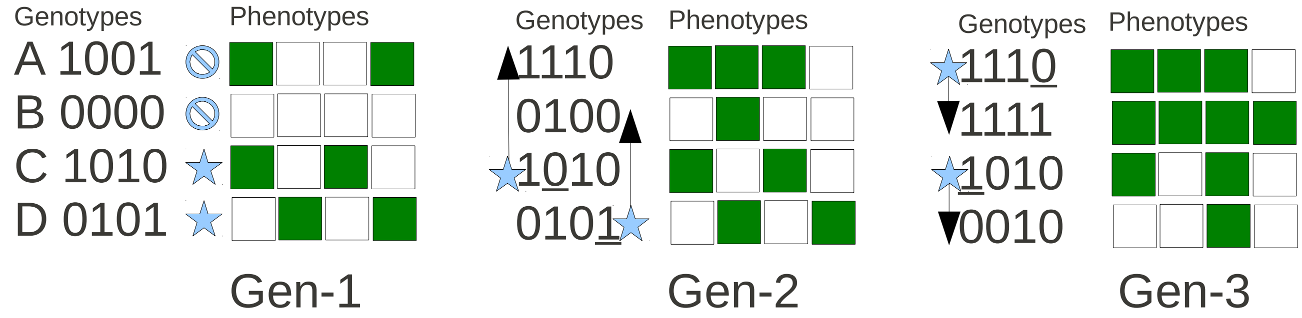
- Improving productivity in a chemical process plant at Imperial Chemical Industries Ltd.
  1. Vary a setting
  2. See what happens
  3. ???
  4. Profit!!!

Evolutionary Computation Flowchart

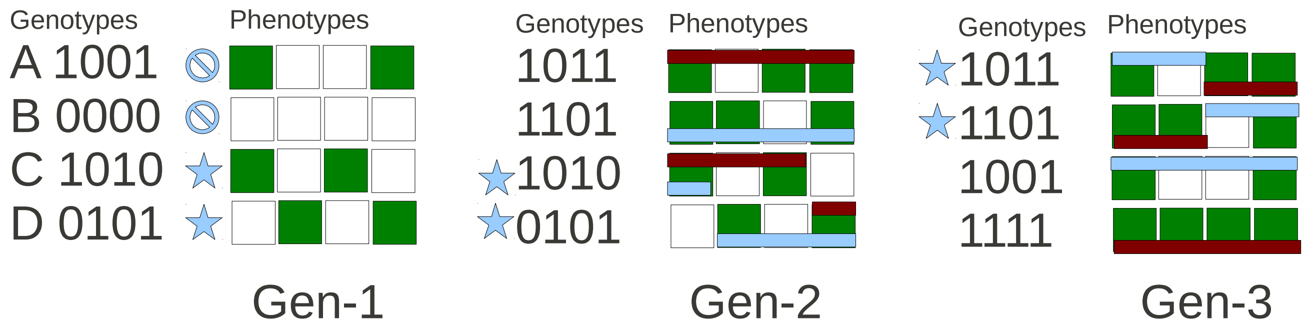


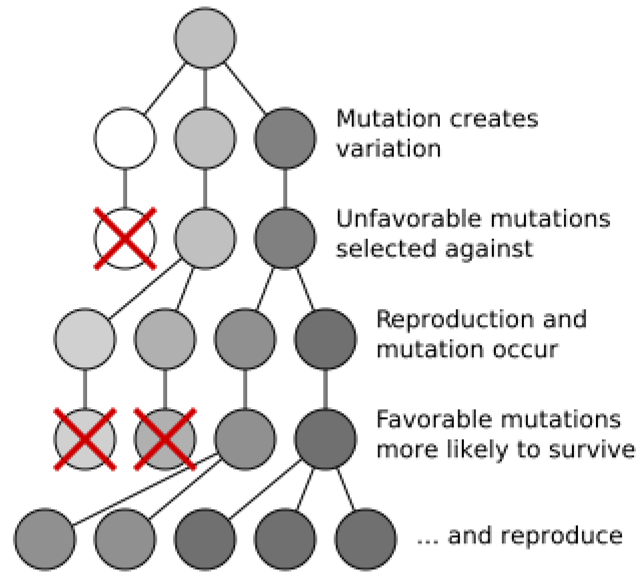
Simple Genetic Algorithm Example

Simple Mutations

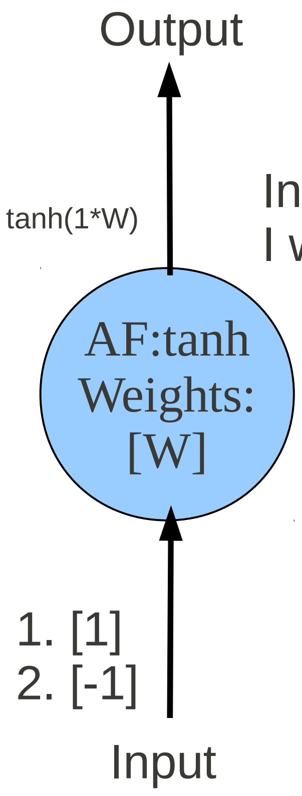


Crossover





Simple Hill Climber



Initial W = 1
I want: Output == 0

1. Output1 = tanh(1*1) = 0.76
   Output2 = tanh(1*-1) = -0.76

2. Perturbation power!!!
Perturbation = -0.5
Try W = 0.5 = 1 - 0.5
Output1 = tanh(0.5*1) = 0.46
Output2 = tanh(0.5*-1) = -0.46
That's closer! New W = 0.5

3. Perturbation power!!!
Perturbation = +0.2
Try W = 0.7 = 0.5 + 0.2
Output1 = tanh(0.7*1) = 0.60
Output2 = tanh(0.7*-1) = -0.60
Not as good as before, New W = 0.5

4. Perturbation power!!!
Perturbation = -0.5
Try W = 0 = 0.5 - 0.5
Output1 = tanh(0*1) = 0 !!!
Output2 = tanh(0*-1) = 0 !!!

The right weight is 0.

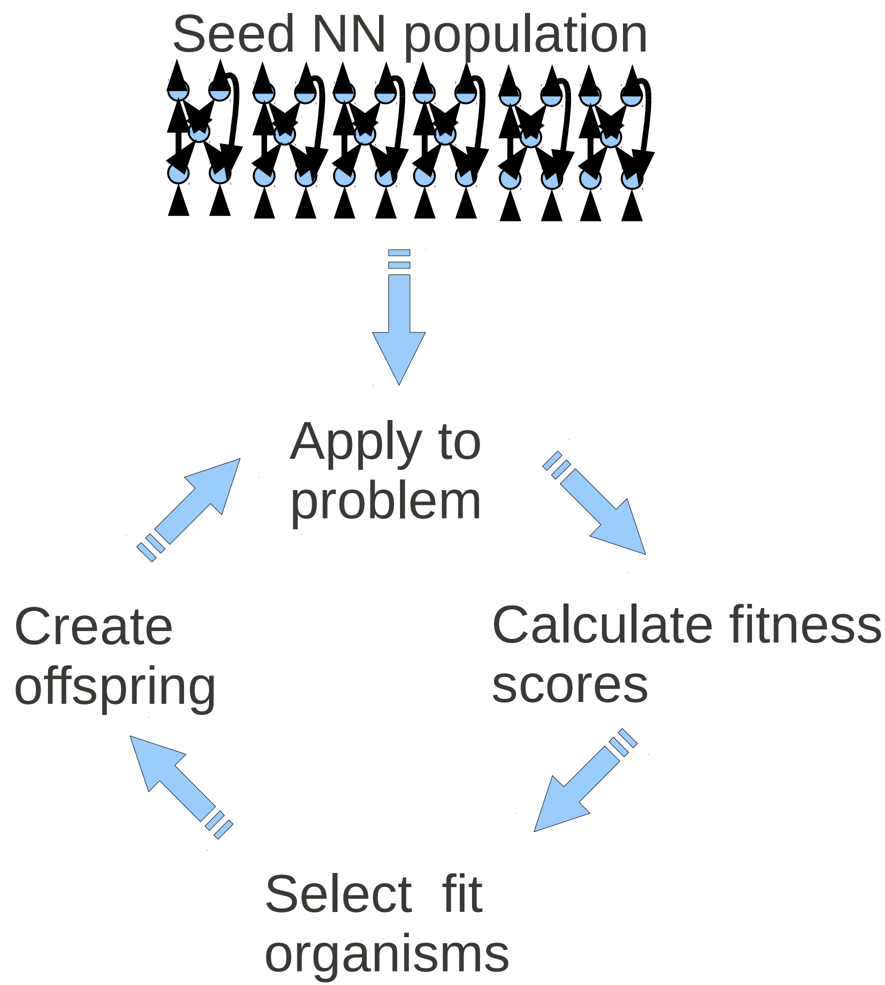
Evolutionary Computation Approaches

- Genetic Algorithms (John Holland, 73-75)
  - Population of fixed length genotypes, bit strings, evolved through perturbation/crossing
- Genetic Programming (John Koza, 92)
  - Variable sized chromosome based programs represented as treelike structures, with specially crafted genetic operators
- Evolutionary Strategies (Ingo Rechenberg, 73)
  - Normal distribution based, adaptive perturbations (self-adaptation)
- Evolutionary Programming (L. & D. Fogel, 63)
  - Like ES, but for evolution of state transition tables for finite-state machines (FSMs)

Topology and Weight Evolving Artificial Neural Networks

- Populations and Fitness Functions
- Parametric mutation operators
- Topological mutation operators
- Other mutation operators
  - Learning Algorithms, Activation Functions
  - Submodules

TWEANN Cycle



NN Programming Language

Necessary Features for a NN-PL

(These will sound very familiar)

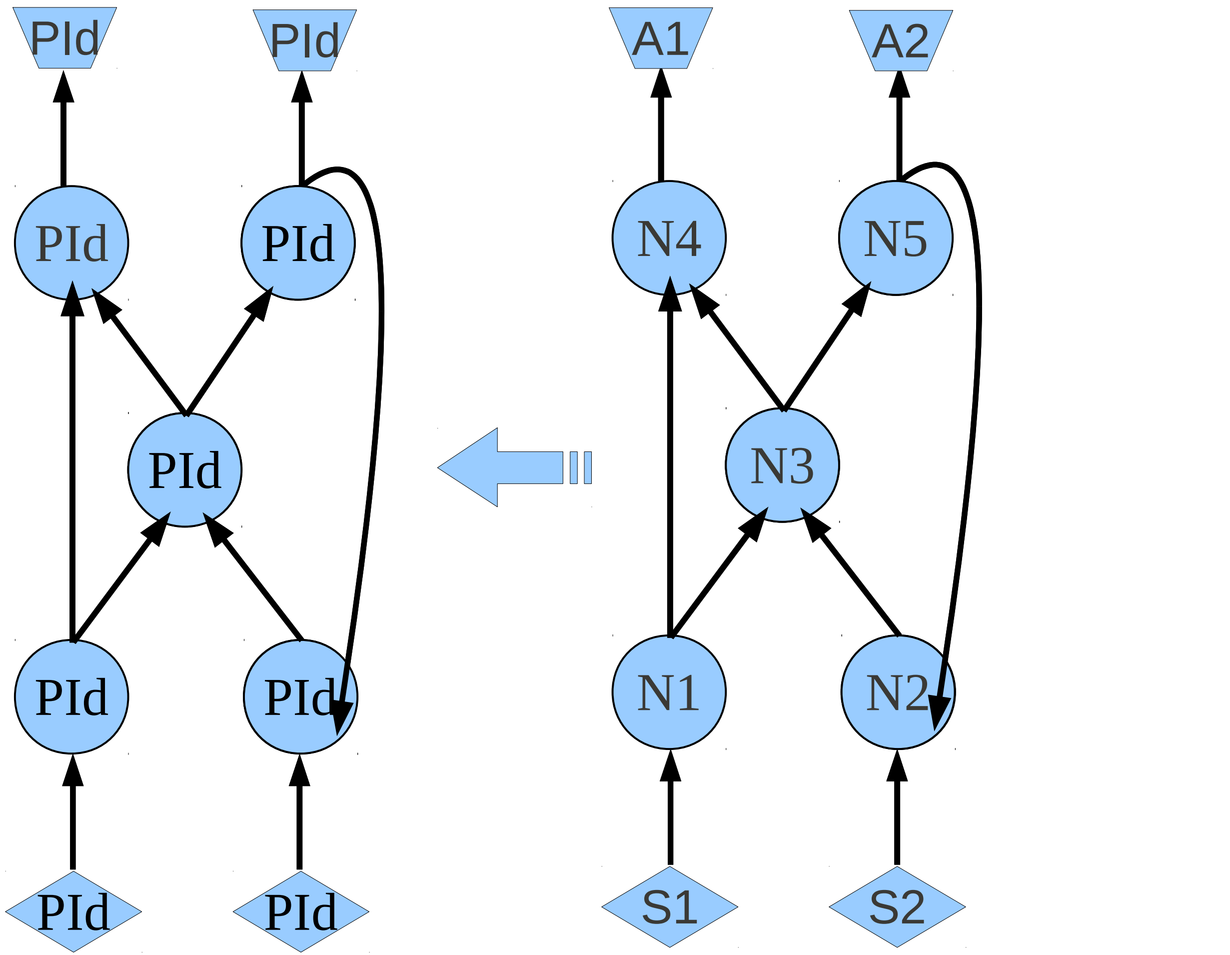
- Encapsulation
- Concurrency through Neuron primitives
- Fault detection primitives
- Location transparency
- Dynamic code upgrade

Erlang's Features

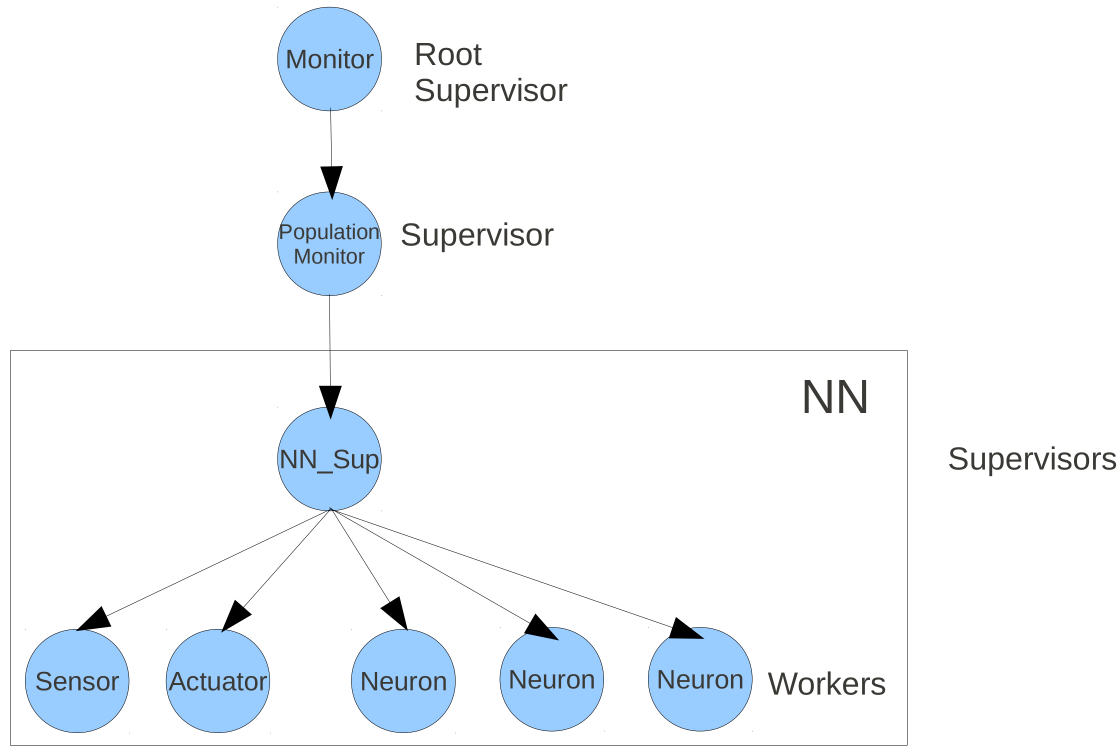
- Encapsulation primitives
- Concurrency
- Fault detection primitives
- Location transparency
- Dynamic code upgrade

Neural Networks Through Erlang

The Topological 1:1 Mapping



Monolithic NN Supervision Tree



States & Functions

- Neuron
  - {InputPIds,OutputPIds,Ws,AF,PF}
  - {U_Ws,Output} = update_weights(Input,Ws,AF,PF)
  - fanout(OutputPIds,Output)
- Sensor
  - {Function,OutputPIds,Parameters}
  - Sensory_Signal = sensor:Function(Parameters)
  - fanout(NPIds,Sensory_Signal)
- Actuator
  - {Function,InputPIds,Parameters}
  - actuator:Function(Actuator_Signal,Parameters)

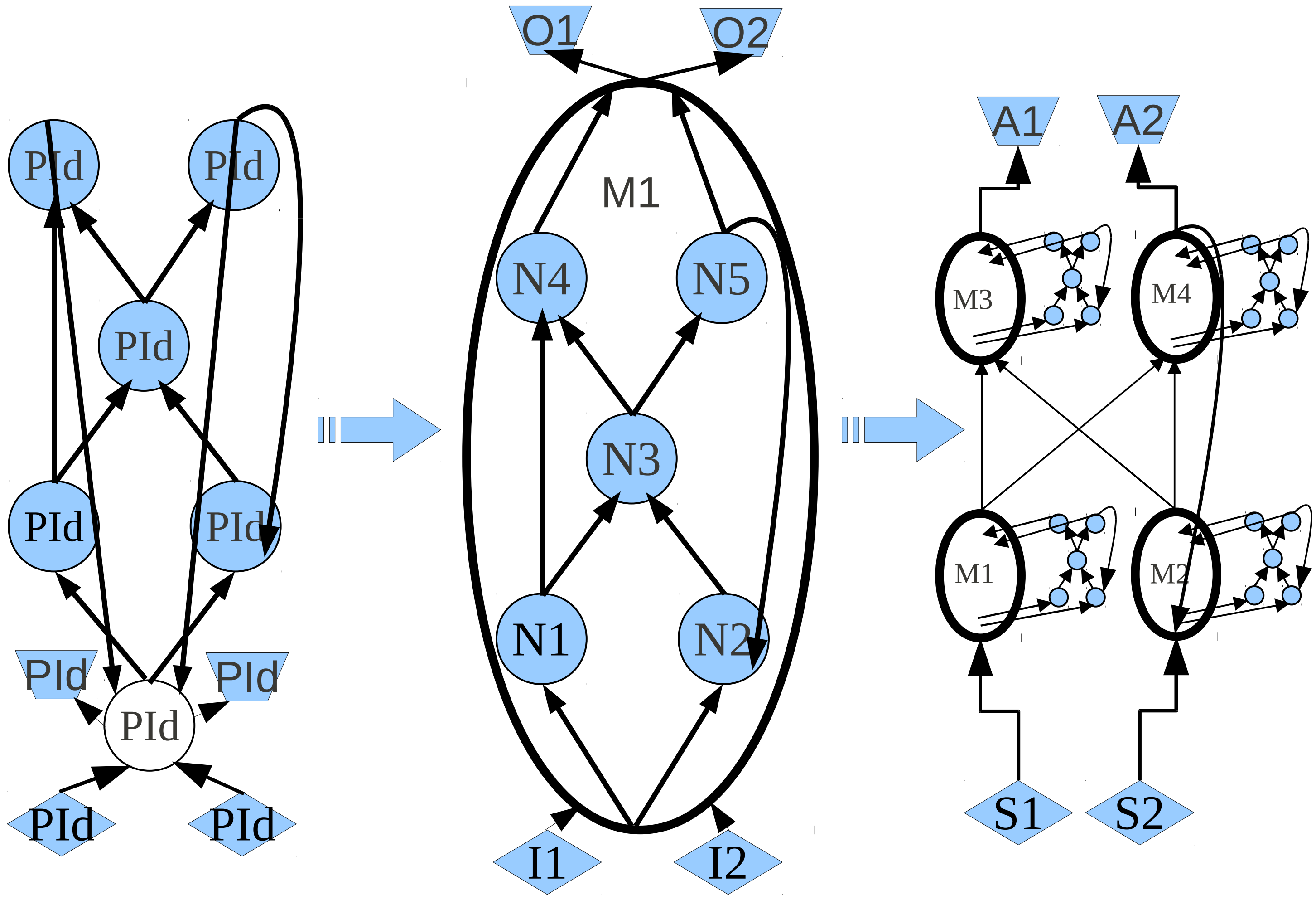
Genotype Storing and Encoding

- Tuple encoded
  - {neuron,InputPIds,OutputPIds,Ws,AF,PF}
- Relational database friendly
- Human readable
- Standard directed graph vertex and edge based
- Mutation operators are standard digraph opreators
  - Add/remove Neuron/Vertex
  - Add/remove Synapse/Edge

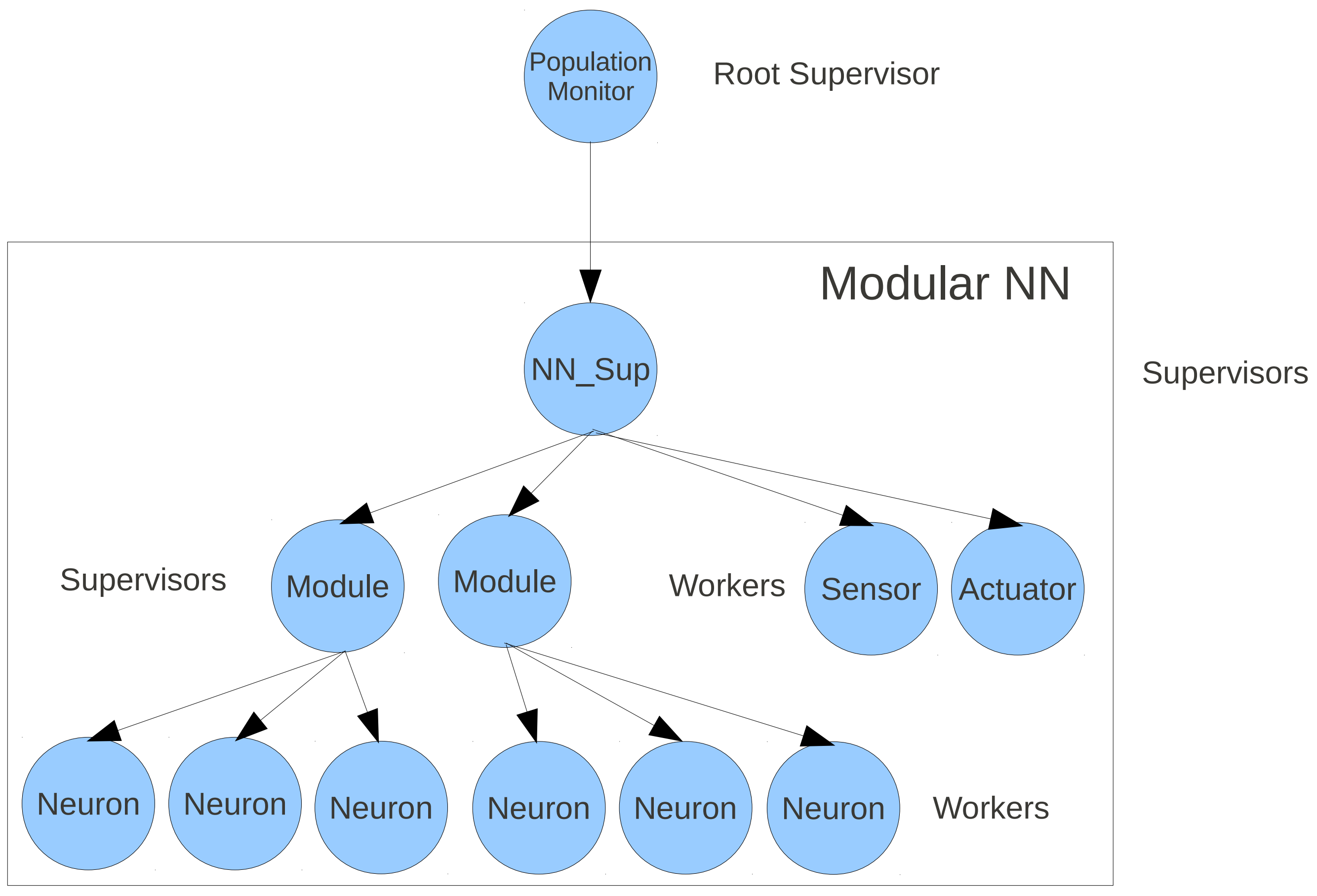
Mnesia as Storage for Genotypes

- Robust and safe
- Tuple friendly
- Easy atomic mutations
  - If any part of the mutation fails, the whole mutation is just retracted automatically

Modular NN Topology Mapping



Modular NN Supervision Tree

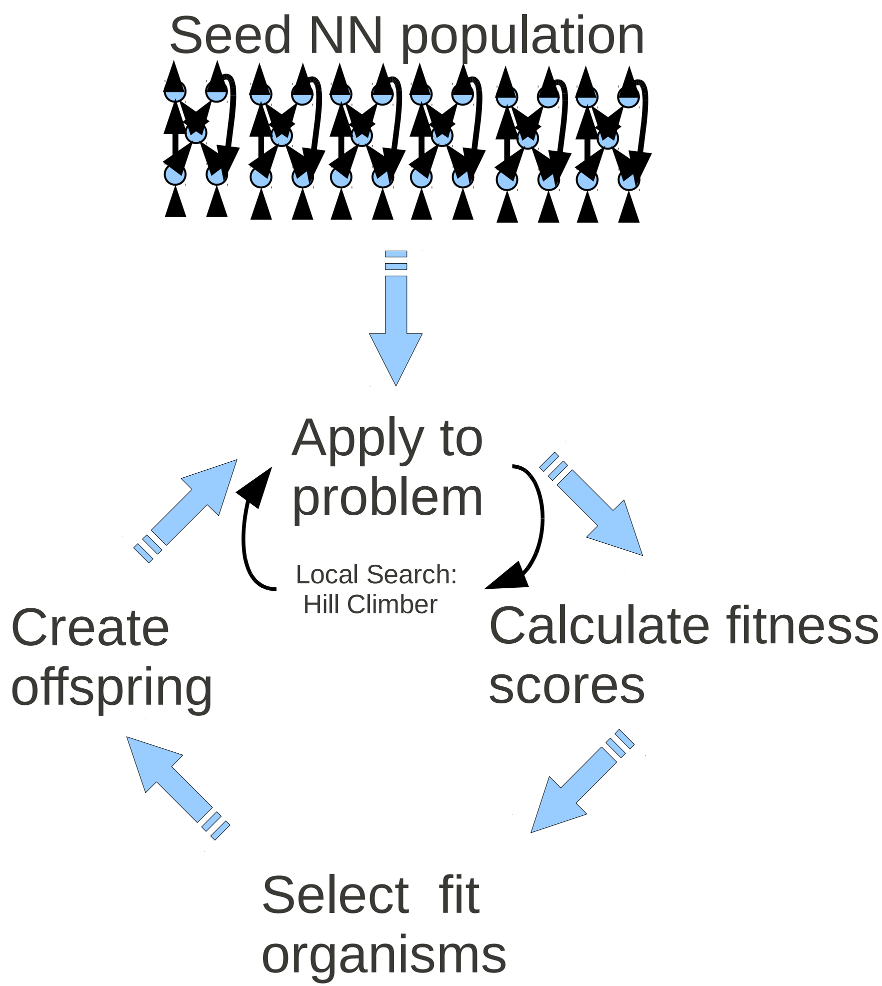


What Erlang Offers to the Field of NN Research

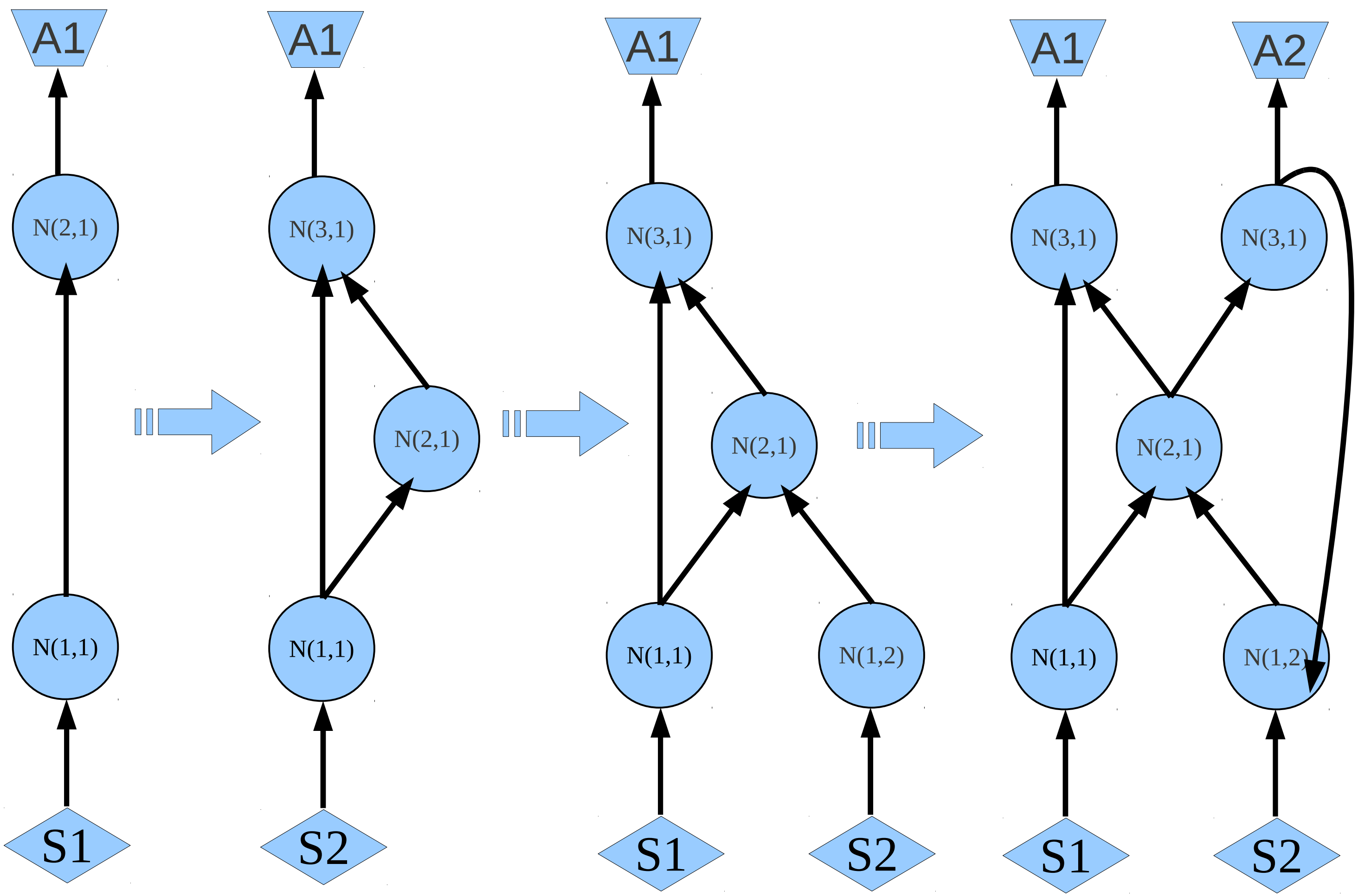
- Augmenting topologies live
- Full distribution and utilization of hardware
- Fault tolerance; ”stroke recovery”
- 1:1 mapping, from ideas to prototype systems
  - No need to overcome linguistic determinism
- Switching and adding new modules, no matter what they do, requires no rewrite thanks to message passing
- Flexibility... in everything!

DXNN: A Case Study

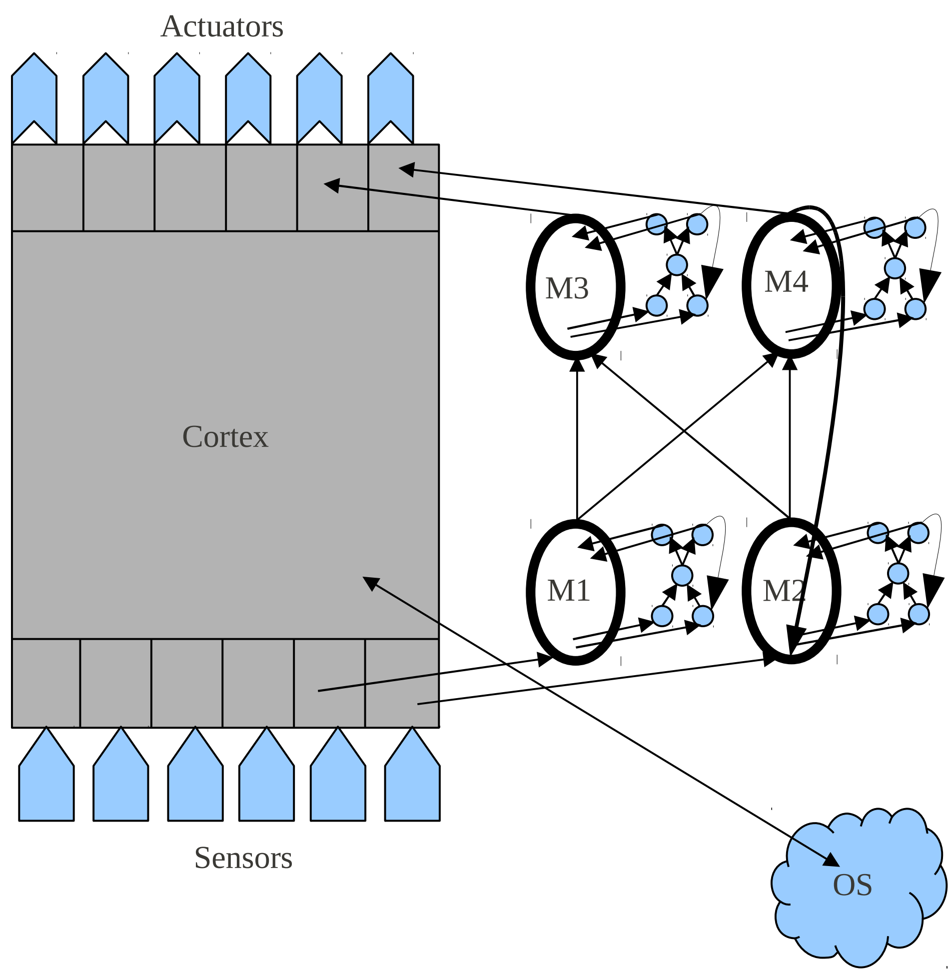
Memetic Algorithm Based TWEANN



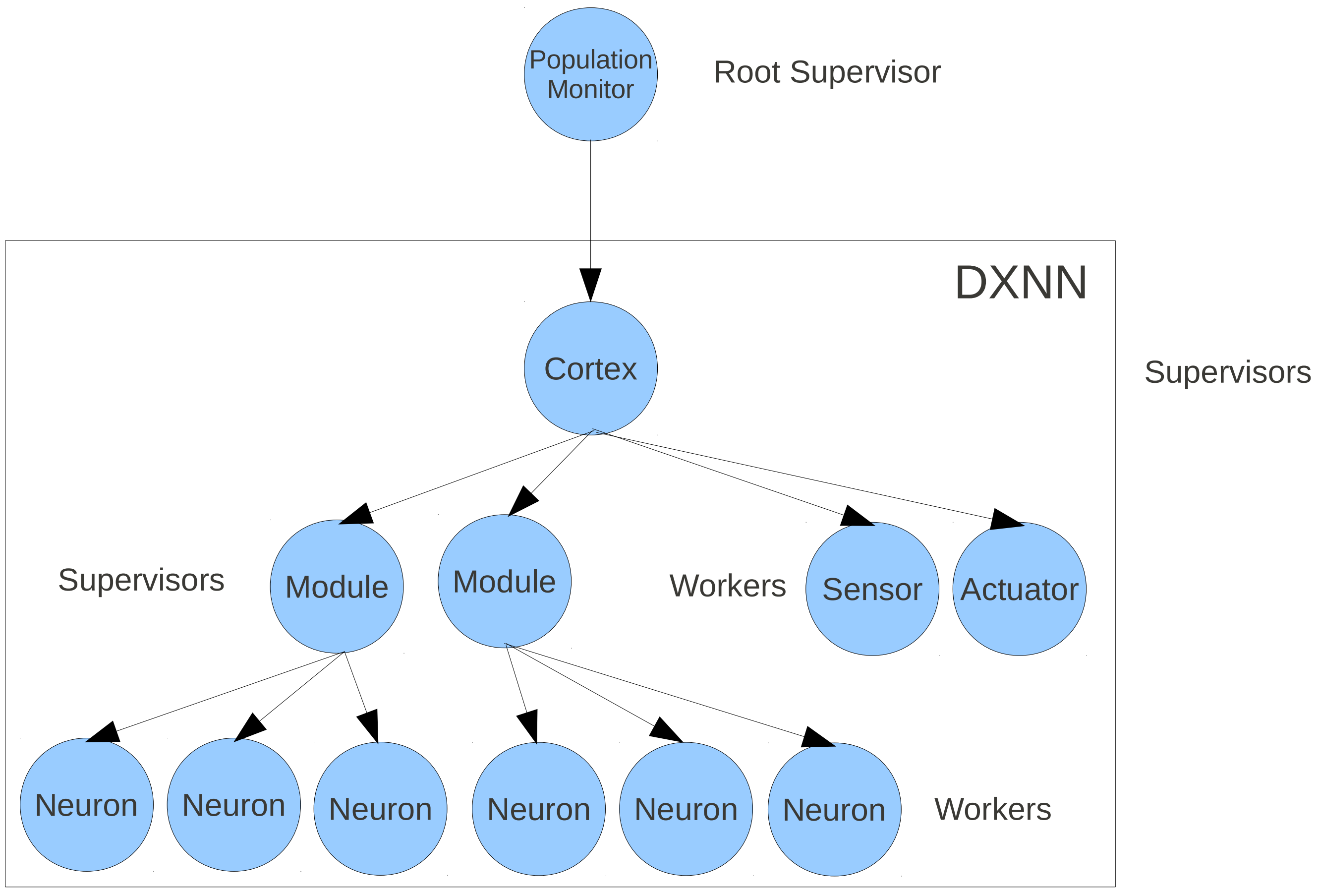
Evolving Topologies



Modular DXNN Architecture



Modular DXNN Supervision Tree

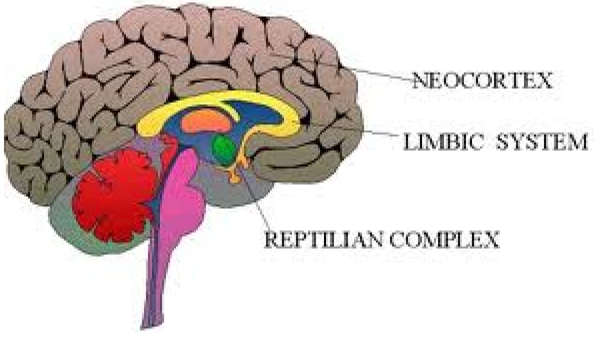


Seed Population

- Total first layer neurons = total # of sensors
- Total last layer neurons = output vector length
  - This is usually the sum of actuator vector lengths
- Choose AF, PF... randomly from the available lists

Complexification and Elaboration

- Start with a simple initial topology
- Add to and elaborate on the topology during mutation phases
- Apply parametric mutations only to the newly created Neurons
- Scale the fitness scores based on NN size



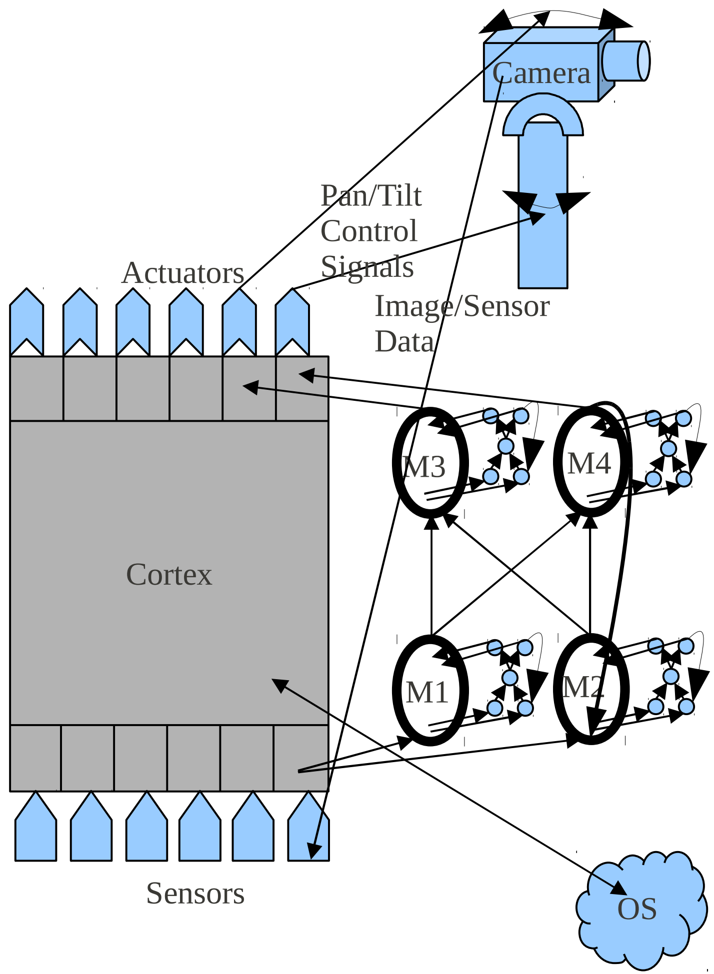
DXNN Genotype

- {neuron,Id,InputPIdPs,OutputPIds,AF,PF}
  - InputPIdPs: [{PId1,W1}...{PIdn,Wn}]
- {module,Id,InputPIds,FanOut,FanIn,OutputPIds}
- {cortex,Id,Sensors,FanOut,FanIn,Actuators}
- {dxnn,Id,CortexId,ModuleIds,NeuronIds}
  - {available_AFs,[AF1...AFn]}
  - {available_PFs,[PF1...PFn]}
  - {available_SensorTypes,[S1...Sn]}
  - {available_ActuatorTypes,[A1...An]}
  - {available_MutationOperators,[MO1...MO2]}

DXNN mutation operators

- Local search mutation operators
  - perturb_weights(Weights)
- Global search mutation operators
  - add_subcore(Genotype)/remove_subcore(Genotype)
  - add_sclink(Genotype)/remove_sclink(Genotype)
  - add_neuron(Genotype)/remove_neuron(Genotype)
  - add_nlink(Genotype)/remove_nlink(Genotype)
  - add_bias(Genotype)/remove_bias(Genotype)
  - change_af(Genotype)/change_plasticity(Genotype)

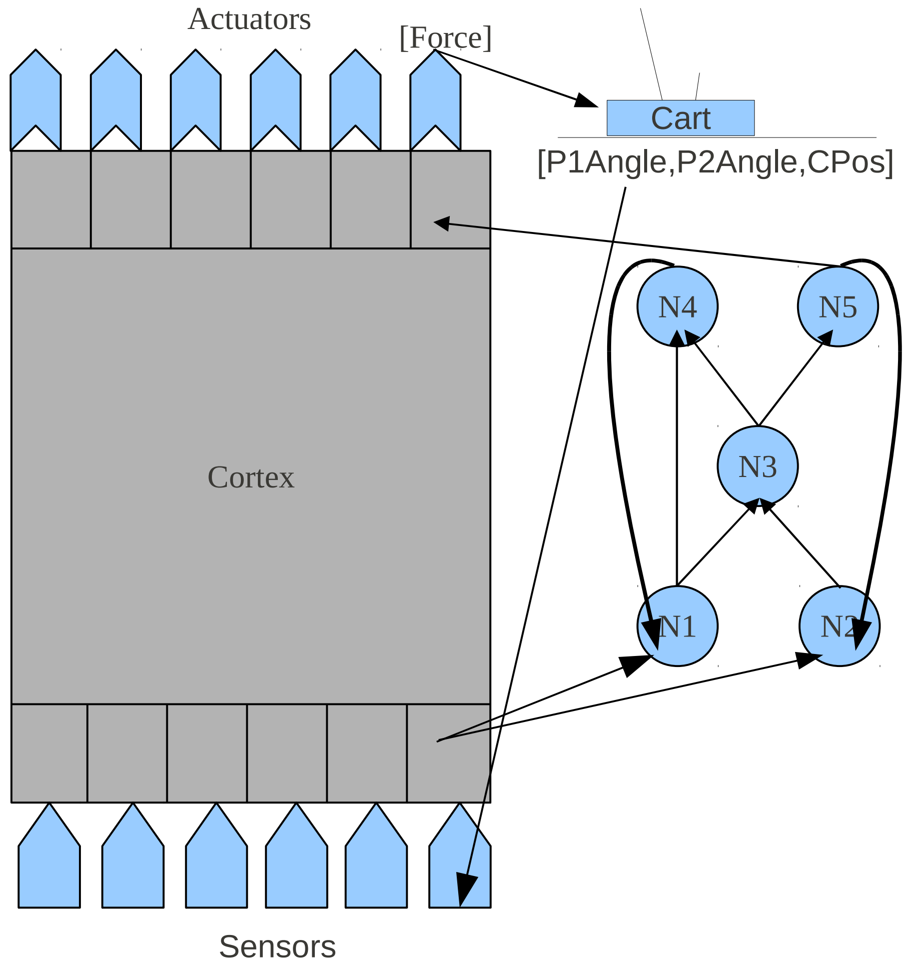
DXNN Platform



DXNN Benchmarks & Application Areas

- Double pole balancing
- Simple food gathering
- Dangerous food gathering
- Predator vs prey simulations
- Room navigation and new sensor acquisition
- Circuit design
- Time series analysis
- ...

Double Pole Balancing Setup



Double Pole Benchmark

Non DXNN data taken from Table-3, Table-4 of: Faustino Gomez, Jurgen Schmidhuber, Risto Miikkulainen,: Accelerated Neural Evolution through Cooperatively Coevolved Synapses. Journal of Machine Learning Research 9 (2008) 937-965

Double pole balancing with velocities

Method	Evaluations
RWG	474329
EP	307200
CNE	22100
SANE	12600
Q-MLP	10582
NEAT	3600
ESP	3800
CoSyNE	954
CMA-EX	895
DXNN	725

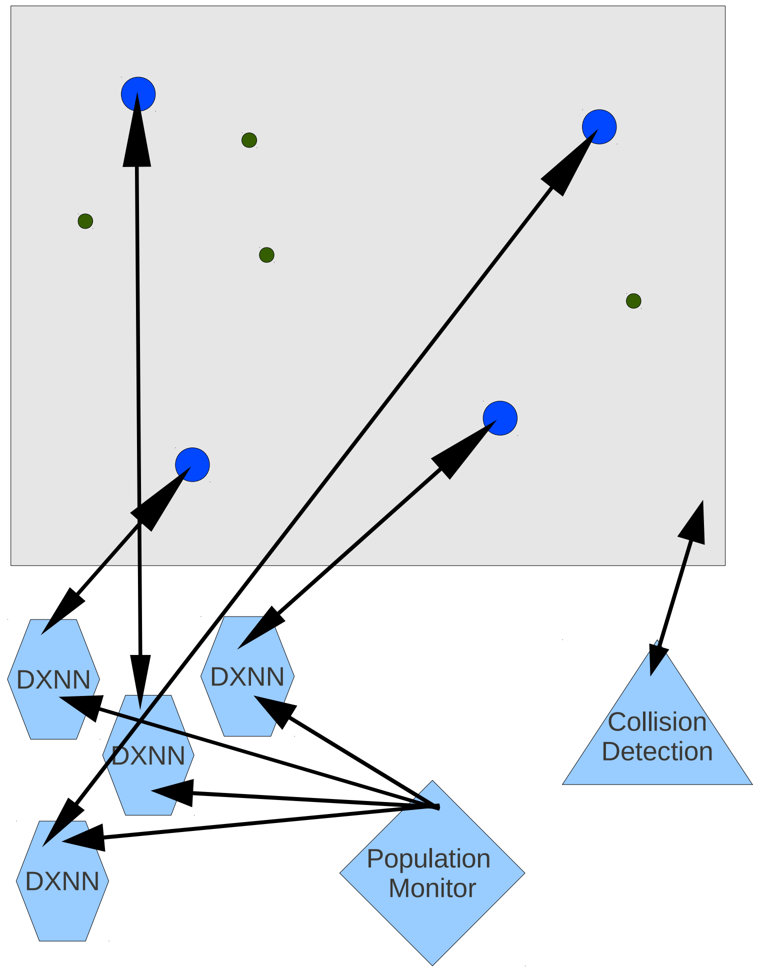
Without velocities, without damping

Method	Evaluations
RWG	415209
EP	
CNE	76906
SANE	262700
Q-MLP	
NEAT	
ESP	7374
CoSyNE	1249
CMA-EX	3521
DXNN	2359

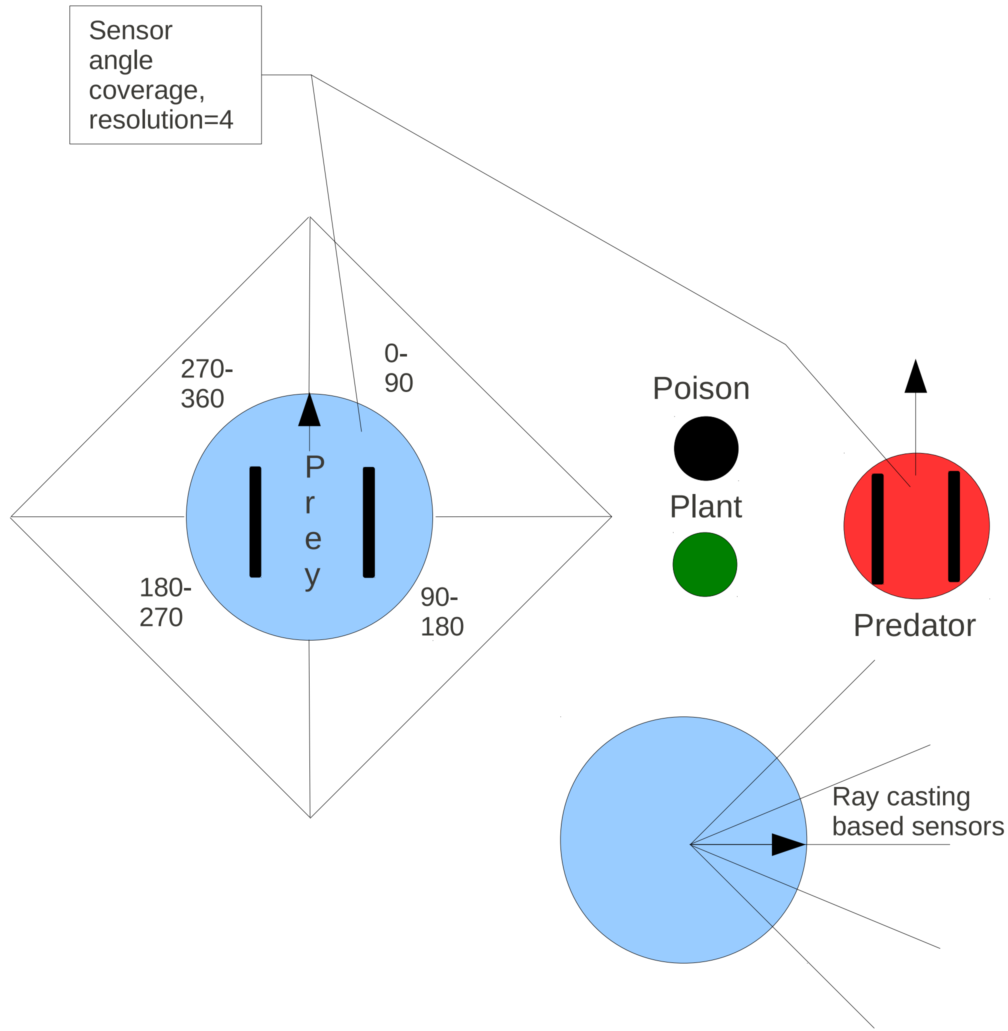
Without velocities, with damping

Method	Evaluations
RWG	1232296
EP	
CNE	87623
SANE	451612
Q-MLP	
NEAT	6929
ESP	26342
CoSyNE	3416
CMA-EX	6061
DXNN	2313

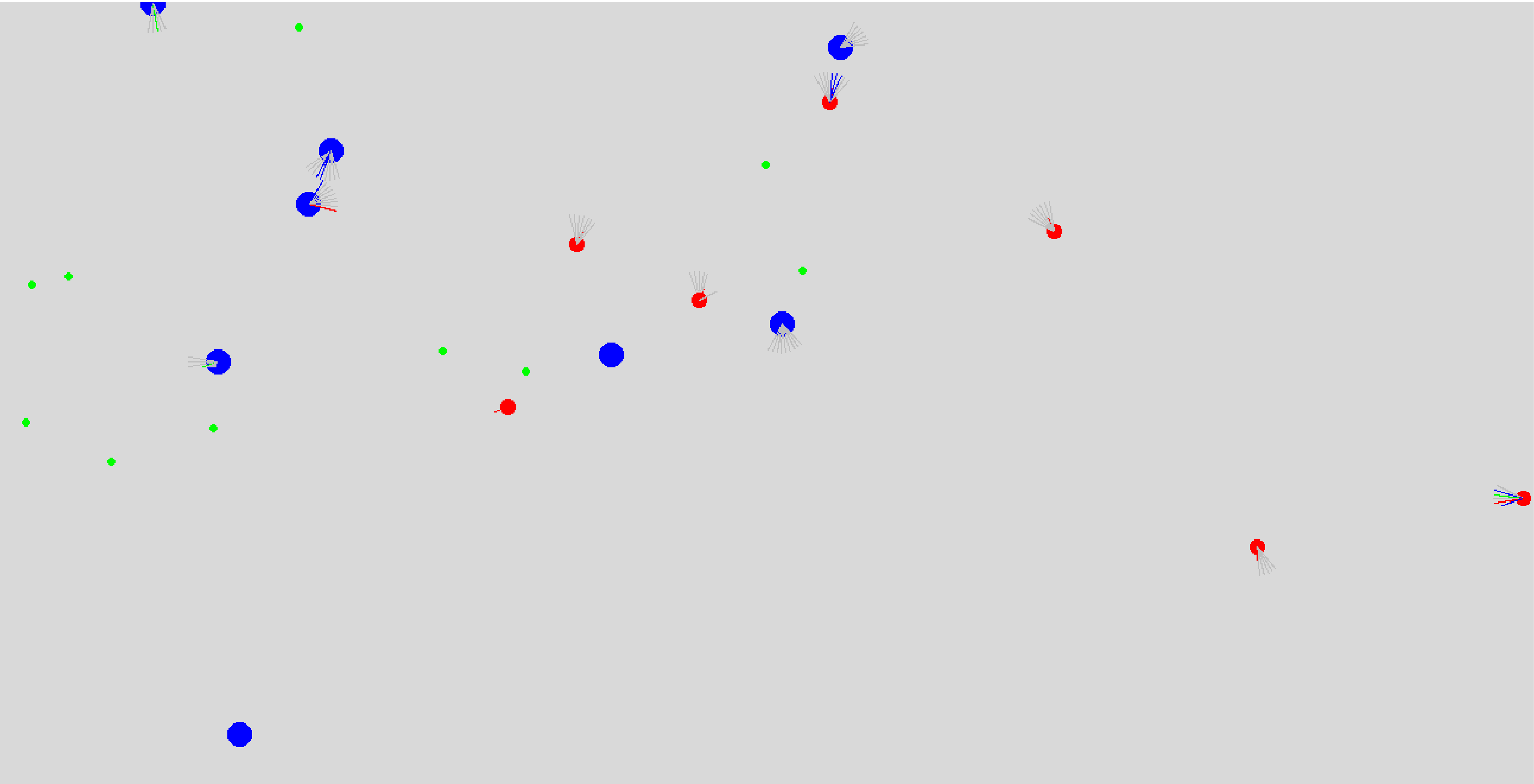
Artificial Life Setup



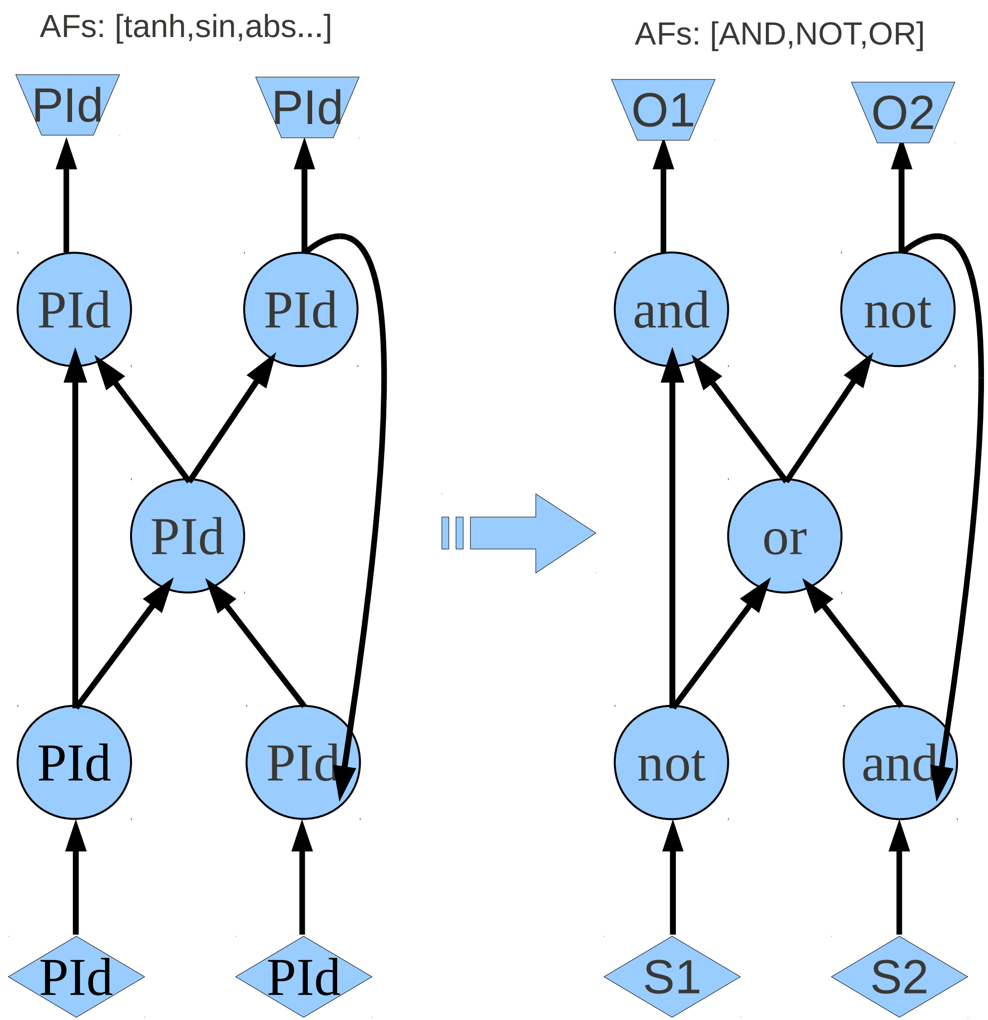
2d Robots and Sensor Types



Predator vs. Prey

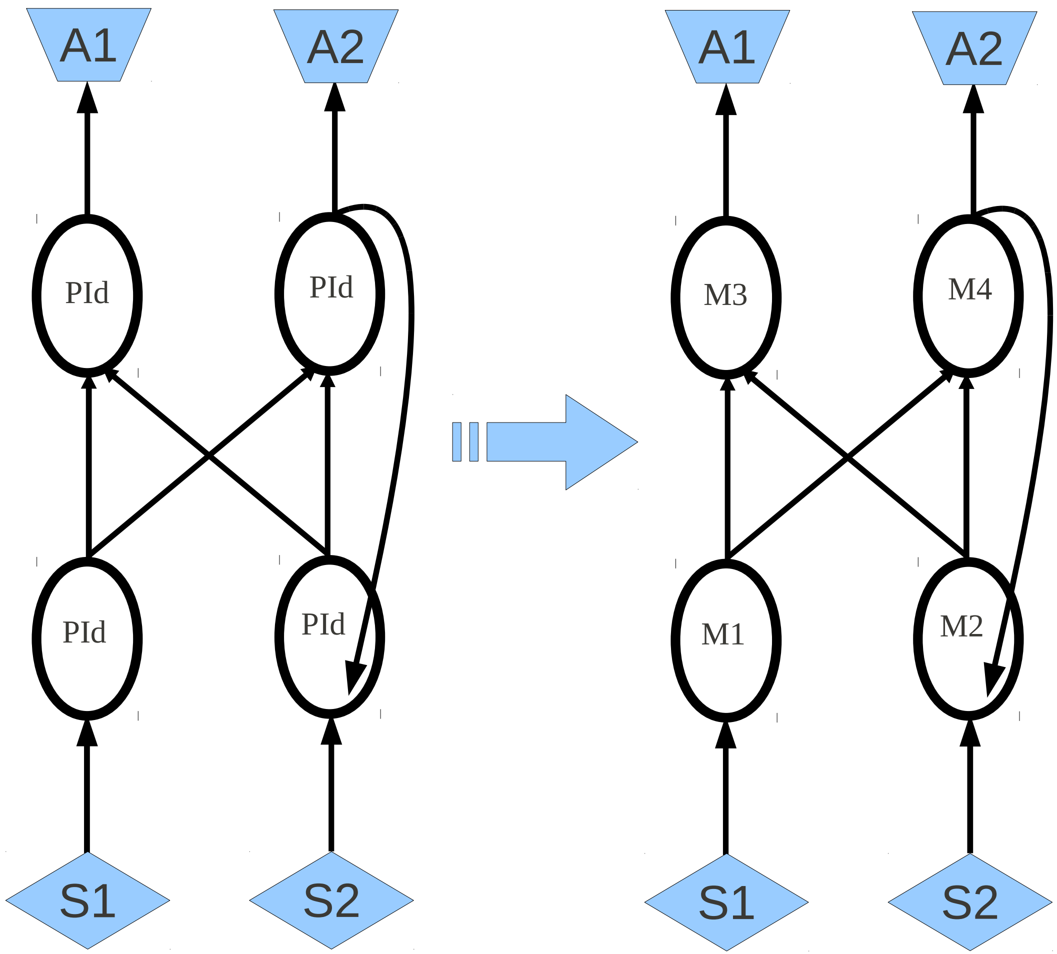


Evolving Hardware

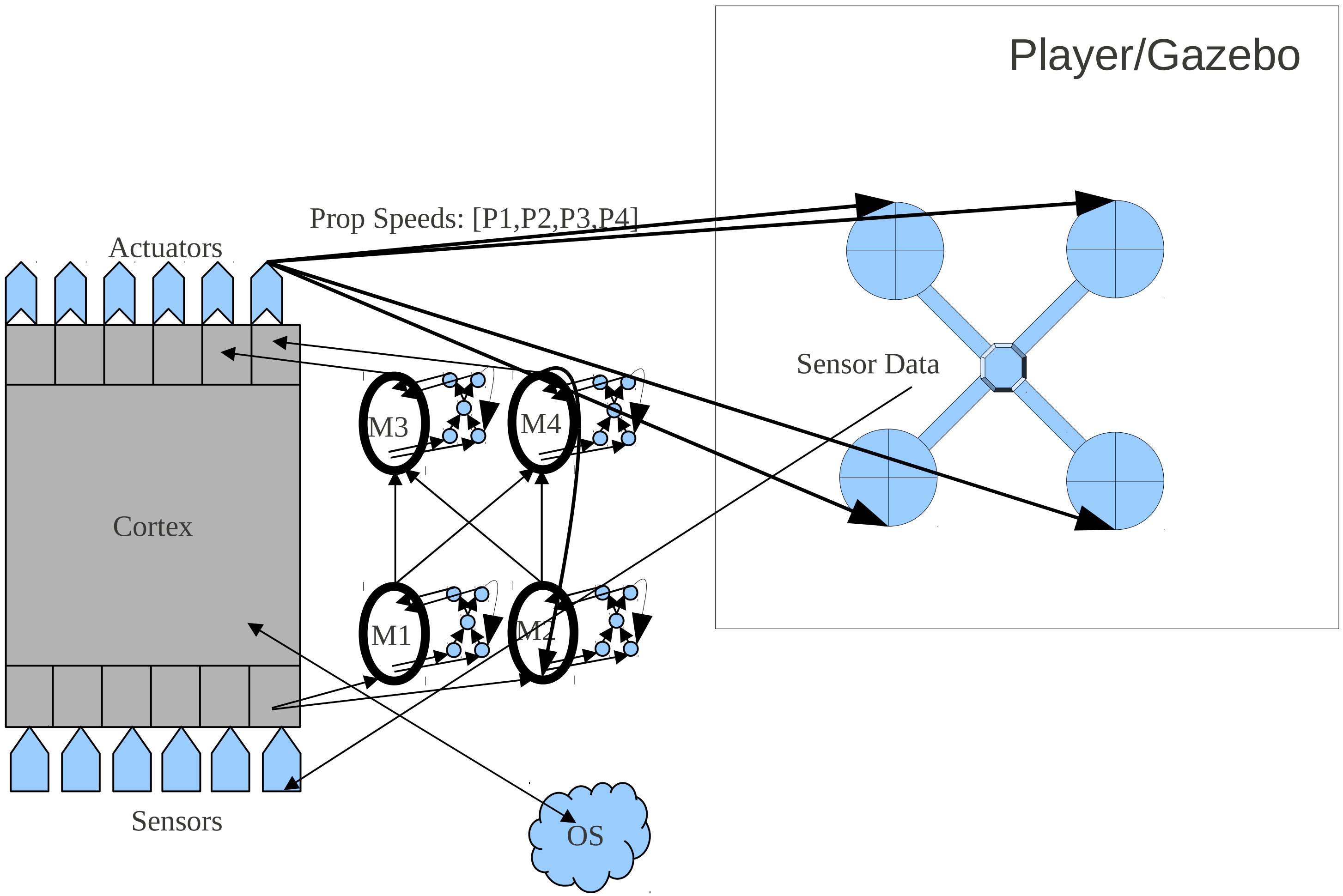


Modules as general programs

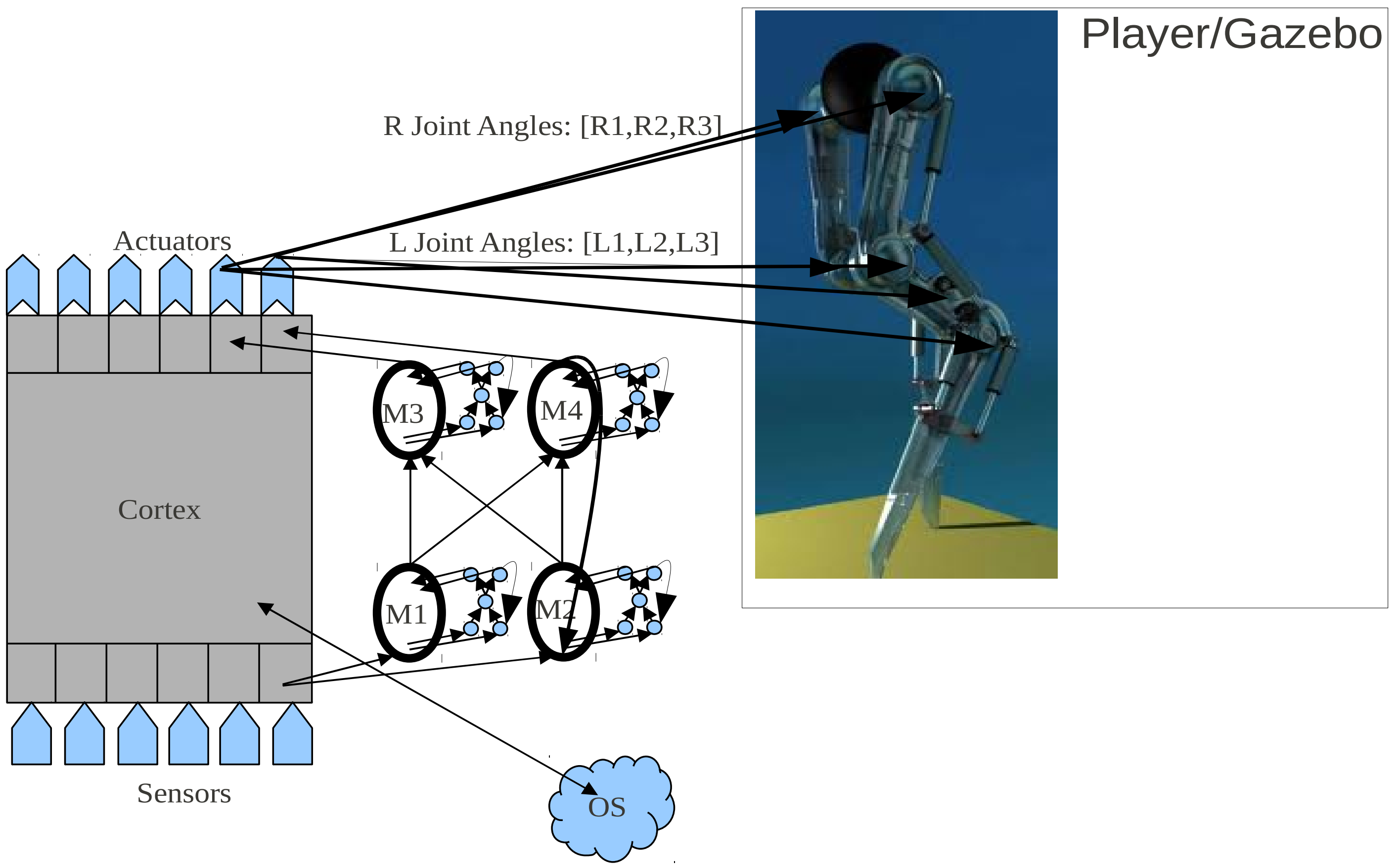
Where each Module can be externally evolved NN, a general program, a circuit, a substrate encoded NN...



QuadCopter Stabilization

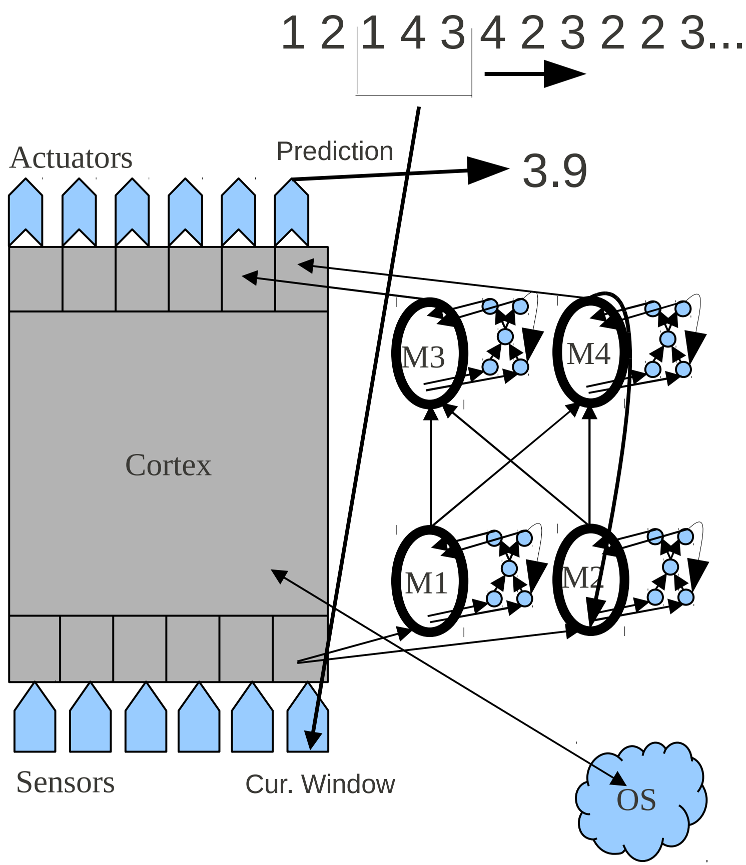


Bipedal Gait



Time Series Analysis

(Through a sliding window)

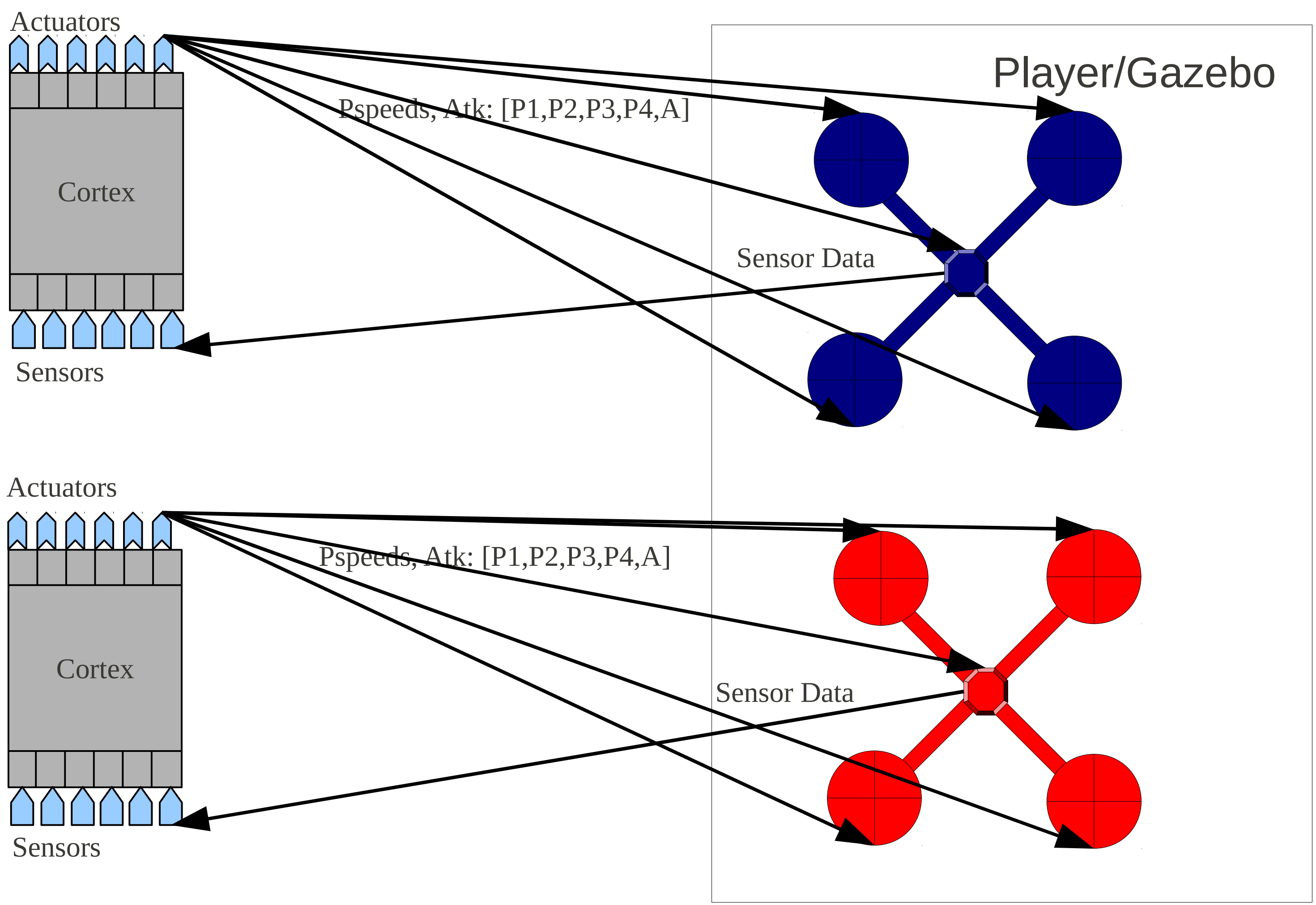


Coevolution

- Environment and fitness landscape is created by the interaction of competing species
- Arms race

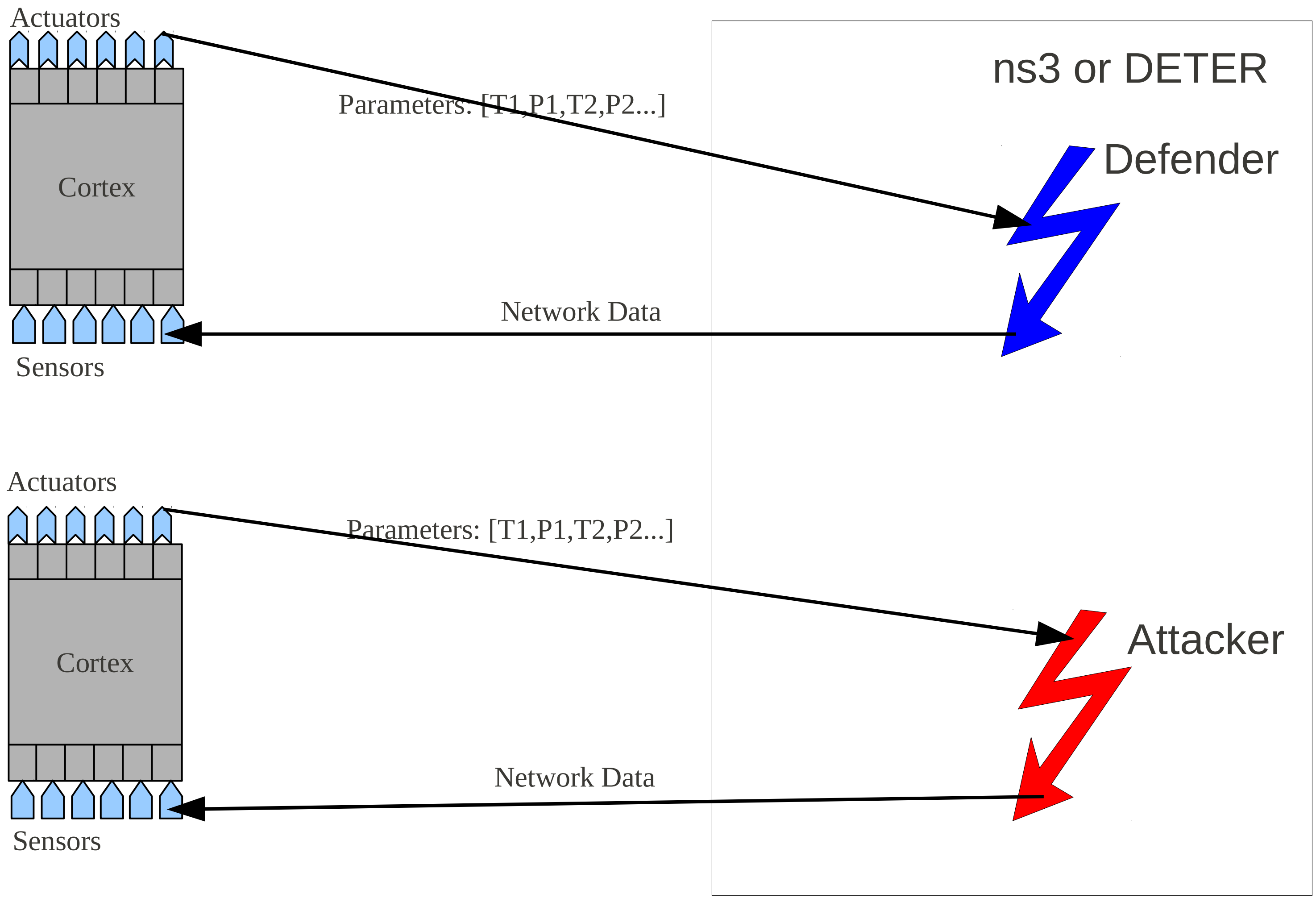
UAVs & Aerial Combat

Bently – Creative Evolutiony Systems, Ch-19, Discovering Novel Fighter Combat Maneuvers: Simulating Test Pilot Creativity



Cyberwarfare

(Something like metasploit, but NN provides the parameters and chooses the tools)
DETER (NetSecTestbed) can be found at http://www.isi.deterlab.net/



Conclusion & Future Work

- Memetic vs Genetic
- Spiking neural networks and fault tolerance
- Over-the-net distribution and sensing
- Artificial life, 3d systems, distributed processing...
- Towards more realistic simulators and evolving greater complexity...

Thanks! Questions?

- Learn More
  - Preprints available on arxiv.org
- Get the code
  - Will be available on github soon
- Get in touch
  - dxnn.research@gmail.com

END