Vector Symbolic Architectures: Analogue computation on discrete data structures

Ross W. Gayler
Honorary Associate
La Trobe University
r.gayler@gmail.com

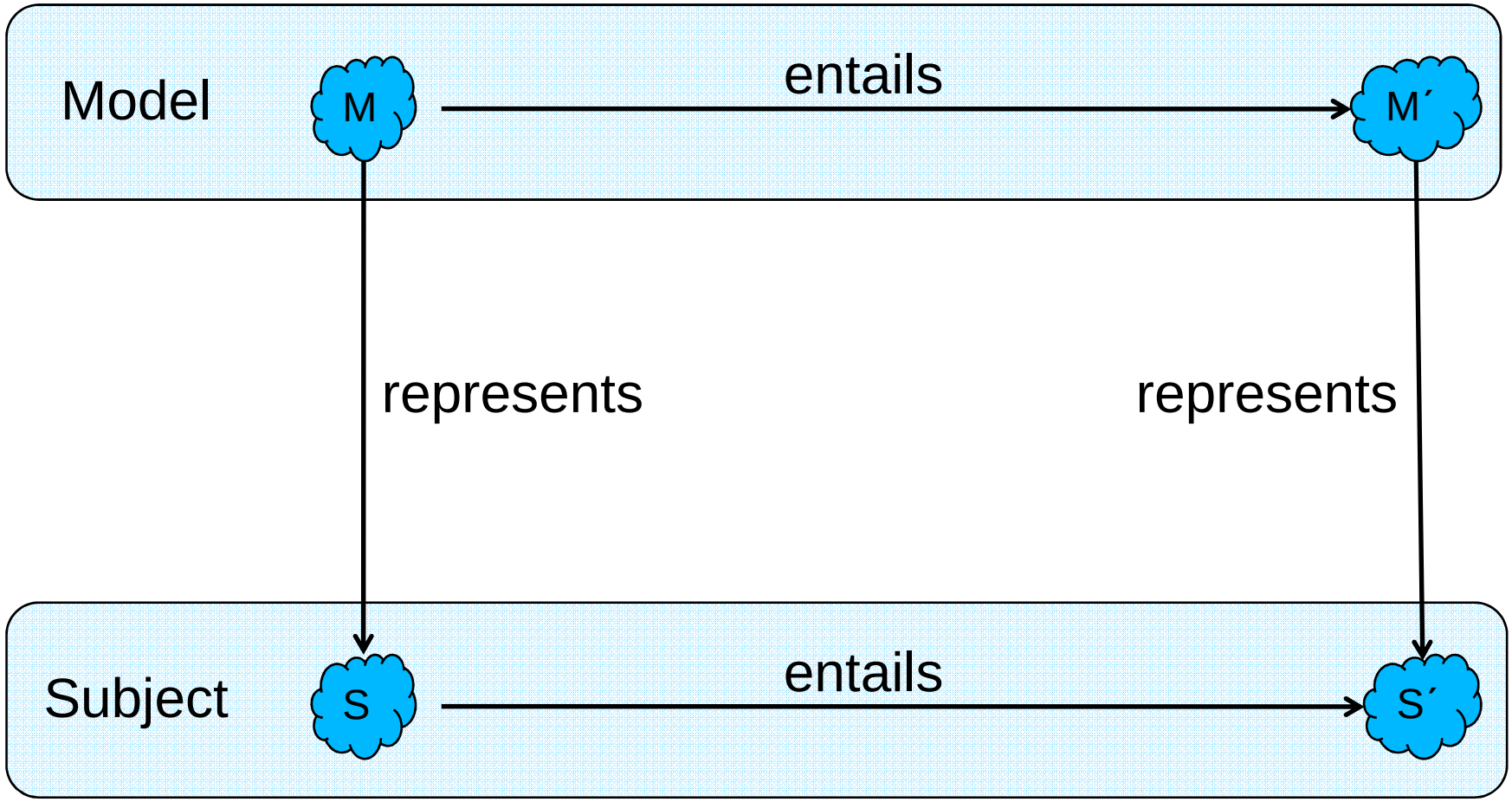
Outline

- What is an analogue computer?
- Analogue approaches to discrete data structures
- Replicator equations for graph isomorphism
- Introduction to Vector Symbolic Architectures
- Replicator equations via VSA

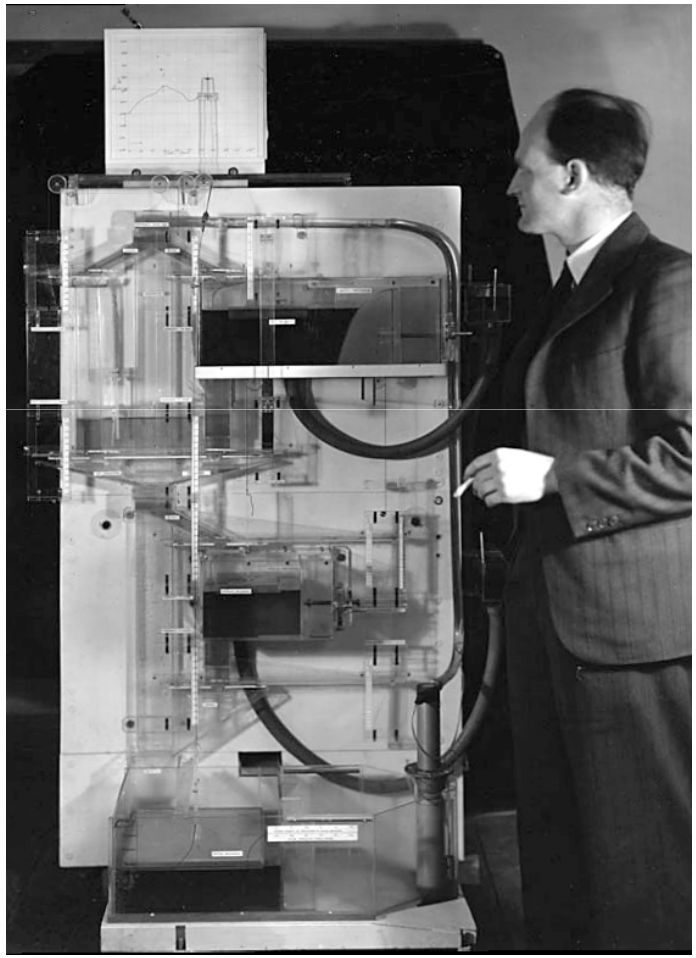
Modelling as isomorphism

- Computing is modelling – look at modelling first
- Subject system has structure
  - Components & relations
- Model system has structure
- Isomorphism between subject and model structures allows the model to stand for subject
- Modelling consists of finding or constructing a model system with the needed structure to adequately match the subject system

Modelling as isomorphism

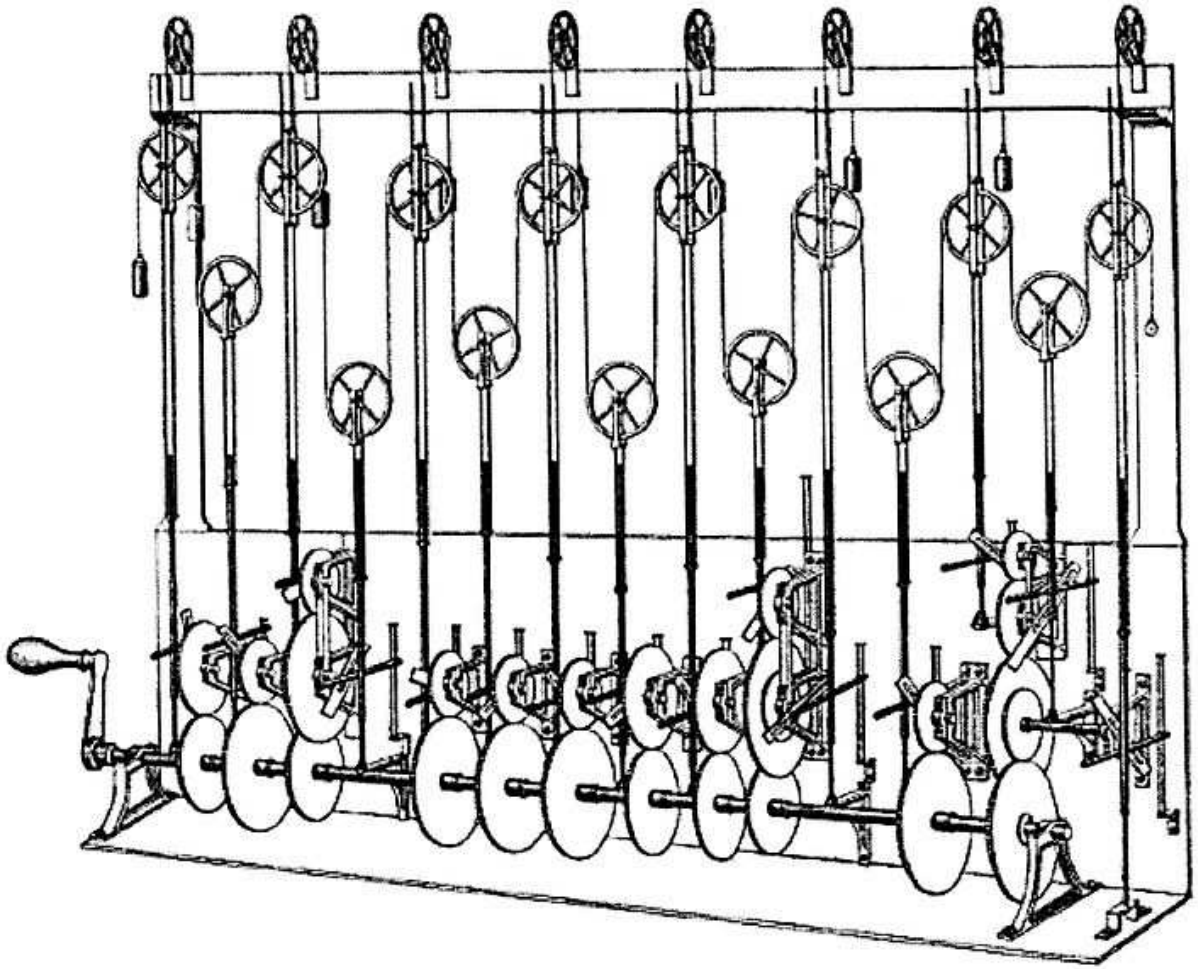


MONIAC



http://upload.wikimedia.org/wikipedia/commons/d/dc/Phillips_and_MONIAC_LSE.jpg

Tide Predictor Machine



http://upload.wikimedia.org/wikipedia/commons/thumb/a/aa/099-tpm3-sk.jpg/794px-099-tpm3-sk.jpg

Computer vs Model

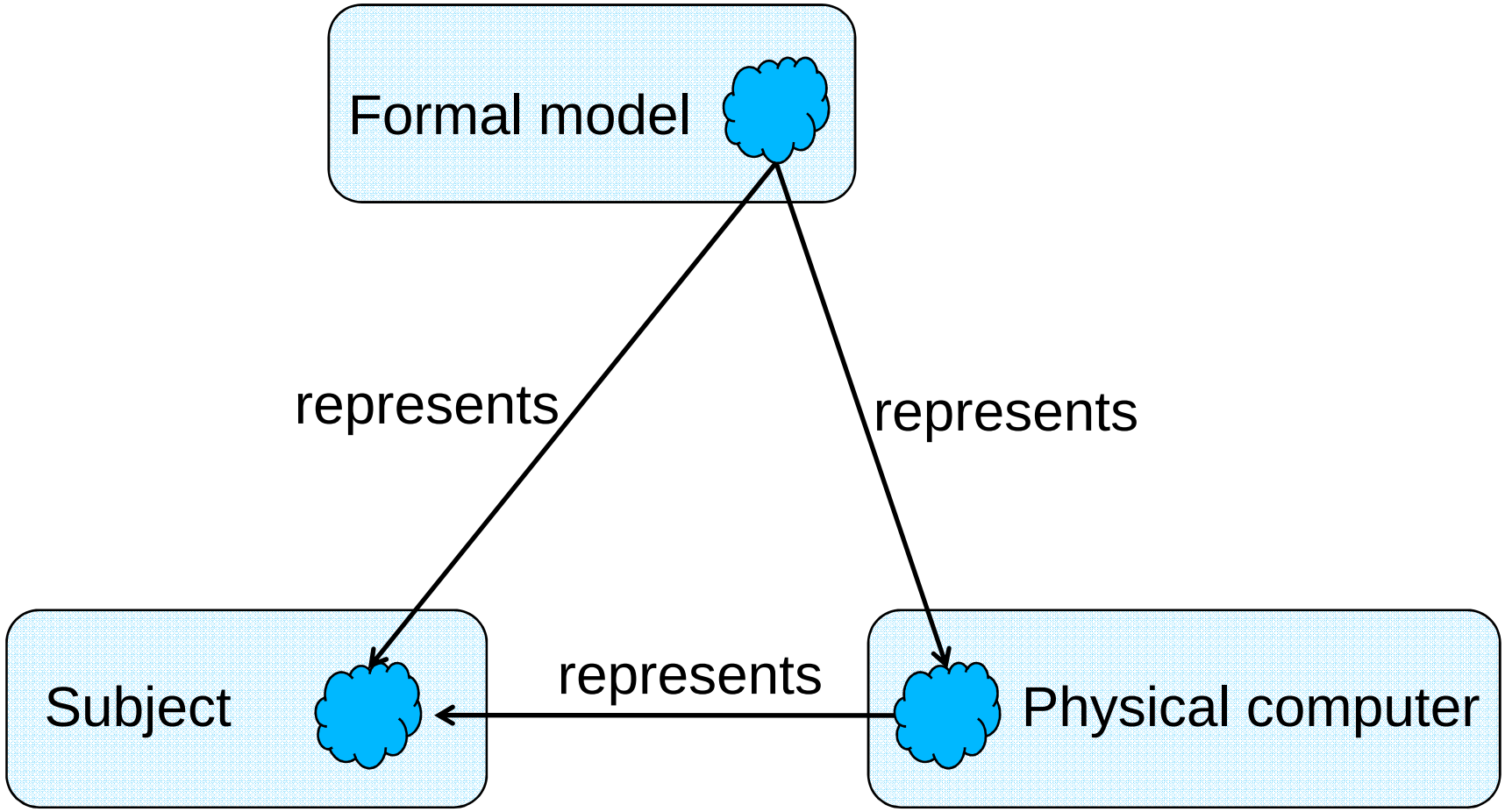
"A computer is any physical object that can be reconfigured to solve multiple problems, that is, to answer many different questions." Jonathan Mills http://www.cs.indiana.edu/hyplan/jwmills/EAC.ppt.htm

- Emphasis on flexibility and reconfigurability
  - e.g. architectural model vs modelling program
- Computers are physical systems
  - Always carry extra, unwanted properties
- Reconfigurable physical systems are rare/special

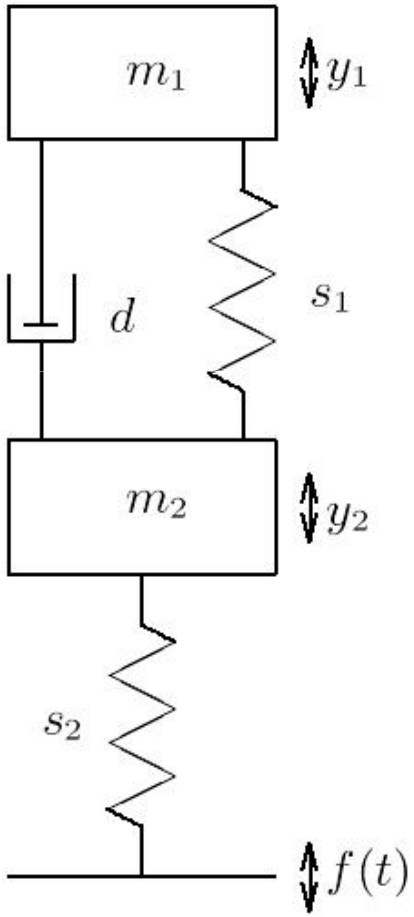
Emphasis on the formal model

- Physical system issues complicate design for reconfigurability
- Introduce a formal model framework
  - Choose formal model that facilitates reconfigurability
  - Formal model must allow modelling of subject
  - Formal model must be adequately approximated by physical computer
    - Logic/discrete maths for digital computers
    - Differential equations for analogue computers
  - Build computer to approximate the formal model
    - E.g. Electronic analogue computers

Emphasis on the formal model



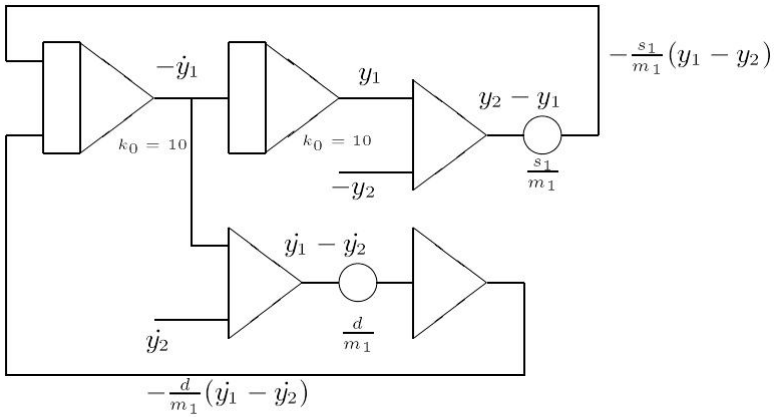
Electronic analogue computers:
Car suspension simulator

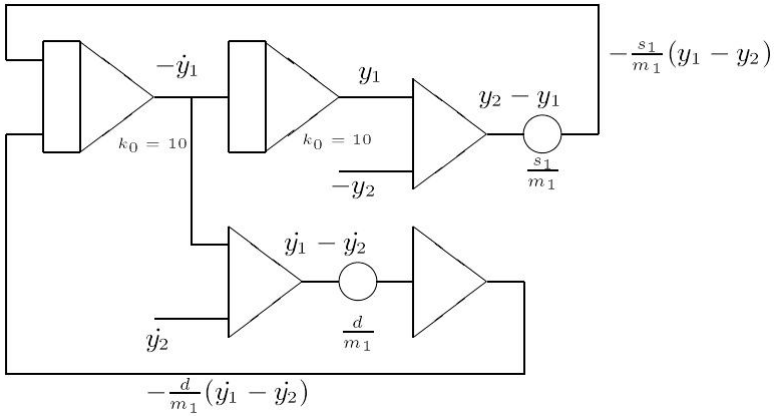


$$m_1\ddot{y}_1 + d(\dot{y}_1 - \dot{y}_2) + s_1(y_1 - y_2) = 0$$
$$m_2\ddot{y}_2 + d(\dot{y}_2 - \dot{y}_1) + s_1(y_2 - y_1) + s_2(y_2 - y_3) = 0$$

http://www.analogmuseum.org/english/examples/vehicle_simulation/

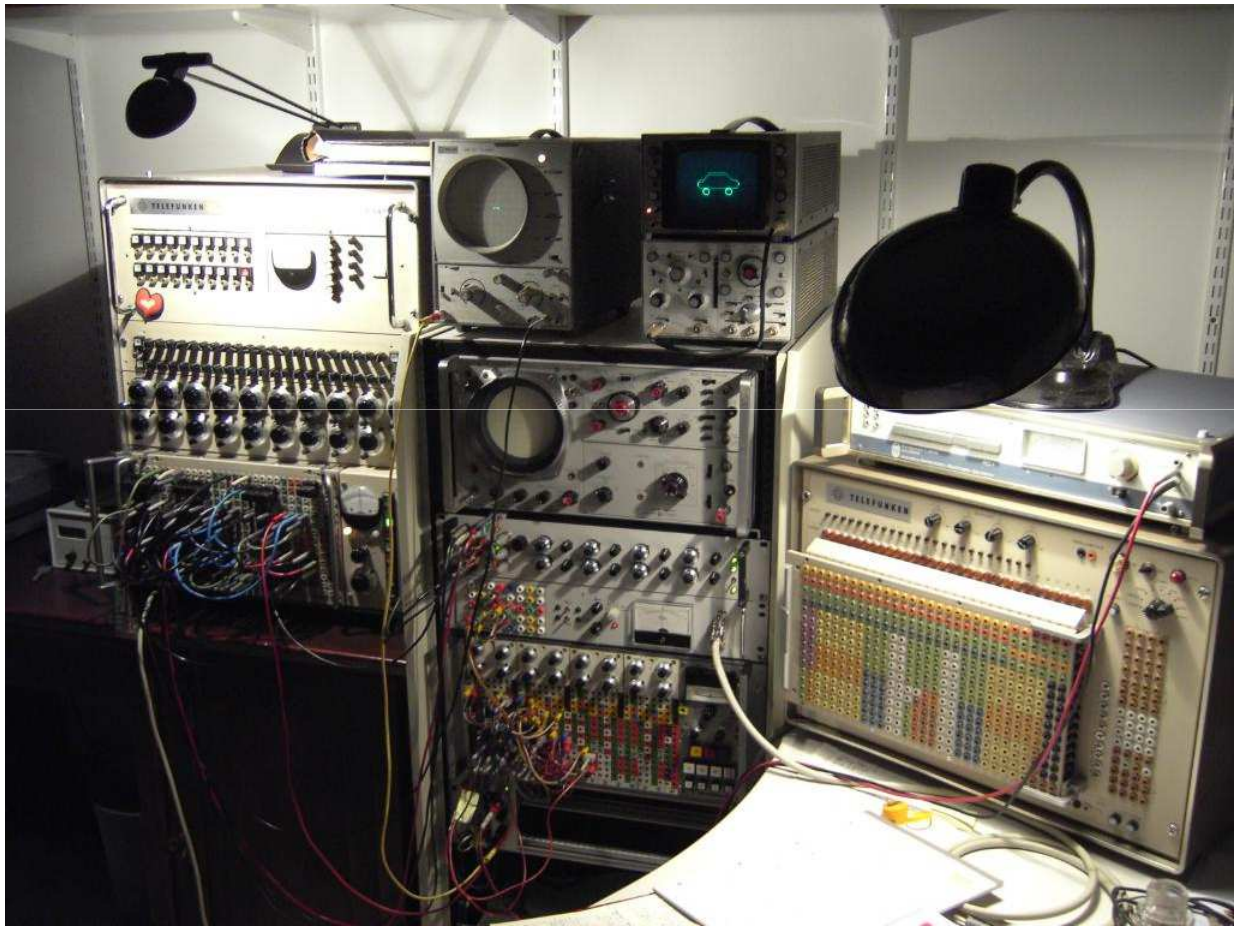
Electronic analogue computers: Car suspension simulator





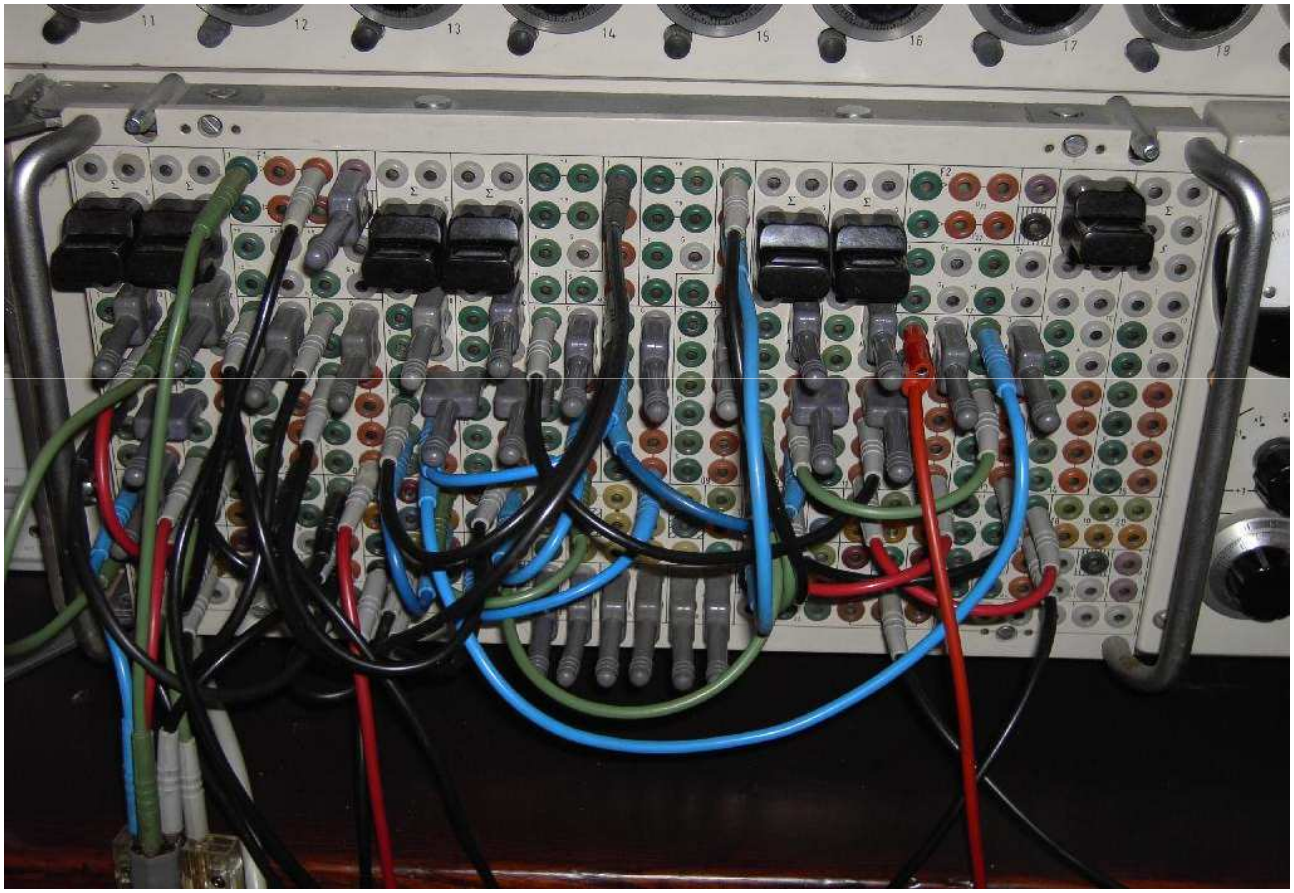
http://www.analogmuseum.org/english/examples/vehicle_simulation/

Electronic analogue computers: Car suspension simulator



http://www.analogmuseum.org/english/examples/vehicle_simulation/

Electronic analogue computers:
Car suspension simulator



http://www.analogmuseum.org/english/examples/vehicle_simulation/

Electronic analogue computers: Wires & functions

- Wires + functions = computation
  - Functions: arithmetic, function look-up, integration
- Wires = functional composition
- Wire = label + scalar magnitude
  - Scalar value exists in system
  - Label exists outside of system (attached by user)
- Similarity to neural networks, but
  - In real neural networks some labels are known fixed
  - Localist neural networks assume all labels fixed

Representing discrete structures: Localist approach

- Dynamics depends on structural components
- Represent (with a wire) every dynamically relevant literal component (e.g. edge(v_1,v_2))
  - Label of wire is identity of structure
  - Scalar magnitude is “support” for structure
- Issues
  - Hardware cost of enumerating all structures
  - Labels not accessible to the computer

Issues not insuperable for small enough problems

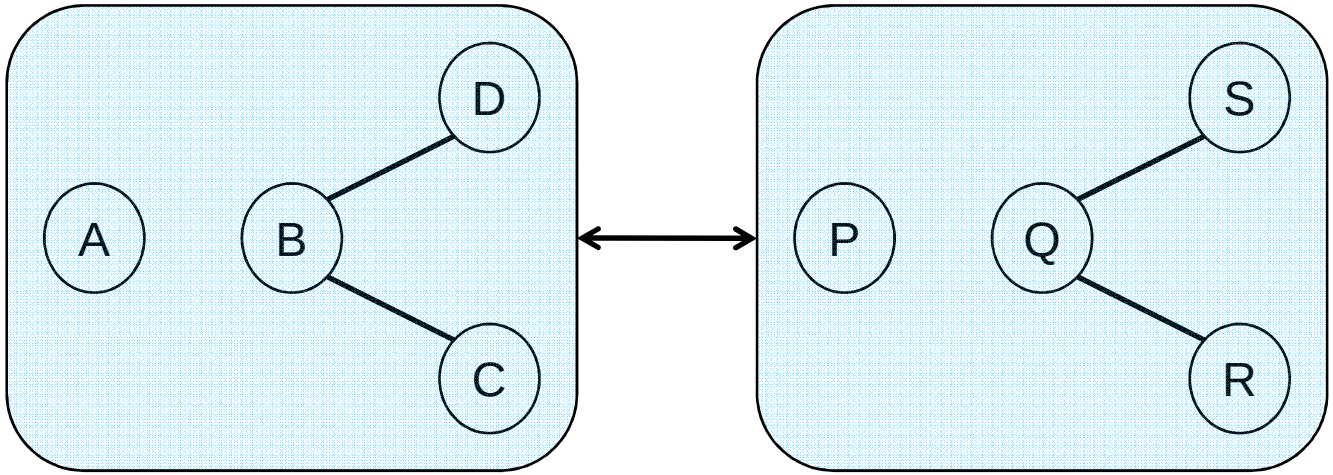
Example: Subgraph isomorphism via replicator dynamics

- Replicator dynamics arise in evolutionary game theory (extensively studied mathematically)
- Applied to graph isomorphism by Pelillo (1999)
- Can represent graphs by numerical matrices
- He mapped graph isomorphism to maximization of a continuous function of those matrices
  - Embedded a discrete problem in continuous
  - Heuristic solution (may not be maximal isomorphism)

Subgraph isomorphism via replicator dynamics

- Vector representing support for all vertex mappings (accumulates the solution)
- Matrix representing compatibility of vertex mappings (constructed from edge information in the graphs to be matched)
- Vector-matrix multiplication is propagation of support between vertex mappings via compatibility information
- Iterative update of the vertex mapping vector

Take two base graphs

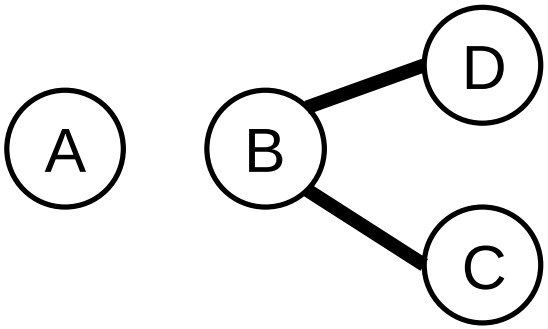


Possible solutions:

{A↔P, B↔Q, C↔R, D↔S}

{A↔P, B↔Q, C↔S, D↔R}

Encode each base graph



	A	**B**	**C**	**D**
A	0	0	0	0
B	0	0	1	1
C	0	1	0	0
D	0	1	0	0

Association graph

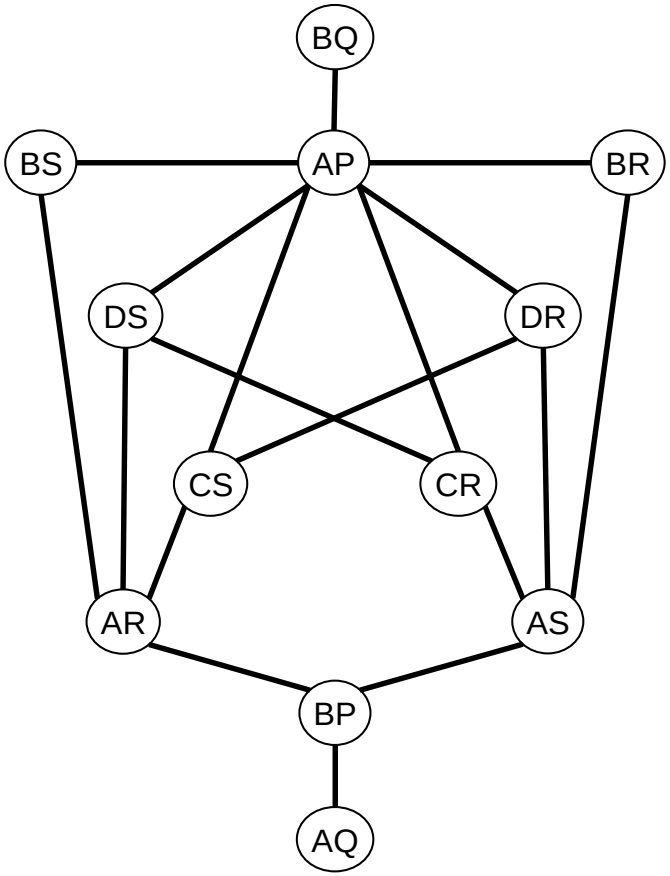
- Represents a product of the two base graphs
- Association graph vertices represent mappings between vertices of the two base graphs
- Edges show local structural consistency

  g1:A—B & g2:P—Q → assoc:AP—BQ
- Edges interpretable as inference rules

  Given edge AP—BQ

  Support for A↔P implies support for B↔Q

Association graph & matrix



	AP	AQ	AR	AS	BP	BQ	BR	BS	CP	CQ	CR	CS	DP	DQ	DR	DS
AP	0	0	0	0	0	1	1	1	0	1	1	1	0	1	1	1
AQ	0	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0
AR	0	0	0	0	1	0	0	1	1	0	0	1	1	0	0	1
AS	0	0	0	0	1	0	1	0	1	0	1	0	1	0	1	0
BP	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0
BQ	1	0	0	0	0	0	0	0	0	0	1	1	0	0	1	1
BR	1	0	0	1	0	0	0	0	0	1	0	0	0	1	0	0
BS	1	0	1	0	0	0	0	0	0	1	0	0	0	1	0	0
CP	0	1	1	1	0	0	0	0	0	0	0	0	0	1	1	1
CQ	1	0	0	0	0	0	1	1	0	0	0	0	1	0	0	0
CR	1	0	0	1	0	1	0	0	0	0	0	0	1	0	0	1
CS	1	0	1	0	0	1	0	0	0	0	0	0	1	0	1	0
DP	0	1	1	1	0	0	0	0	0	1	1	1	0	0	0	0
DQ	1	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0
DR	1	0	0	1	0	1	0	0	1	0	0	1	0	0	0	0
DS	1	0	1	0	0	1	0	0	1	0	1	0	0	0	0	0

Replicator equations

$$\pi_i(t) = \sum_{j=1}^{N} w_{ij} x_j(t)$$

$$x_i(t+1) = \frac{x_i(t)\pi_i(t)}{\sum_{j=1}^{N} x_j(t)\pi_j(t)}$$

π_i Evidence for vertex mapping *i*

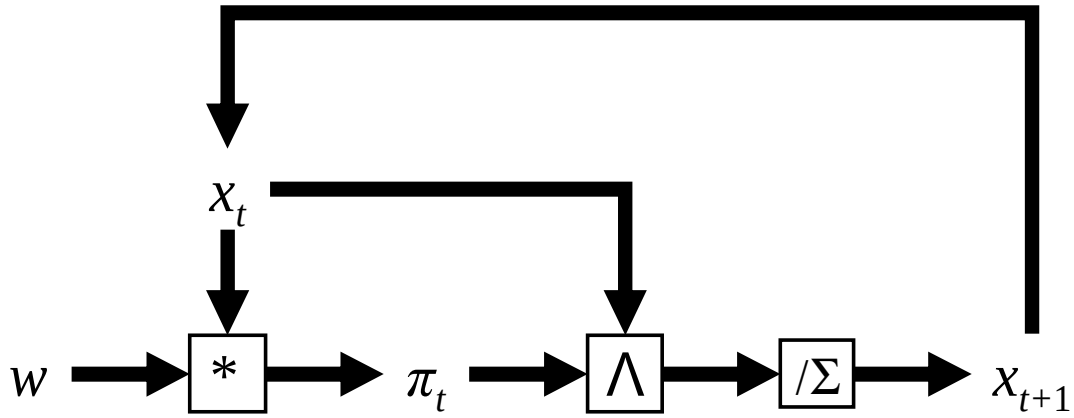
x(t) Prior support for vertex mappings

x(t+1) Posterior support for vertex mappings

W Inference rules (Association matrix)

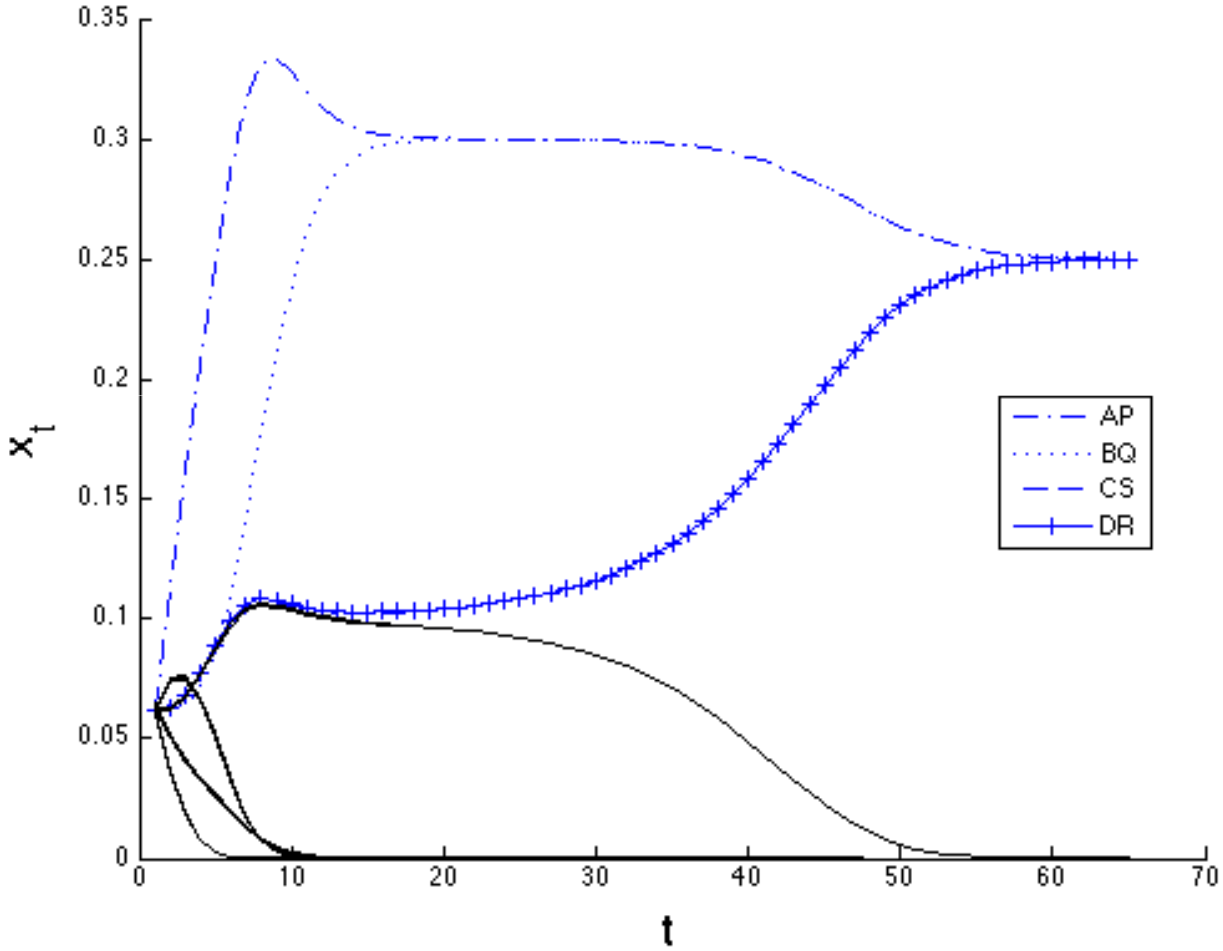
- Direct connection with Bayesian inference (Harper, 2010)
- There is a continuous time version

Analogue architecture



- Each element of a vector or matrix can be implemented as a wire (label + magnitude)
- Functions are simple arithmetic
- Localist representation: each entity maps to wire
  - Can be implemented as a localist neural network

Settling of localist system



Performance of replicator dynamics

- Pelillo tested on graphs with > 65,000 vertices
- Competitive with state of the art search
- Typically settle in < 100 iterations
- Fast parallel implementation possible
- Settling time ~ independent of graph size
- Settling time depends strongly on how constraining the graphs are
- Generalized to weighted, attributed graphs

Problems with localist graph isomorphism implementation

- Localist implementation identifies functional elements with physical resources (wires)
  - Scaling: k^2 vertex mappings, k^4 edge mappings
- Labels represent one specific mapping problem
  - Labels not available to localist analogue computer
    - An issue for autonomous (cognitive) systems
  - Reconfigurability: How to solve different problems?
  - Possible solutions:
    - External process to dynamically relabel wires
    - Fixed set of all possible labelled wires
    - Not attractive solutions in neural networks

Distributed analogue representation

- Make labels values that can be represented
  - Wires become label:scalar bindings
  - Labels can be manipulated by the computer
  - Starting to look like a digital computer – but need to do it in a way that respects analogue dynamics
- Use a representation that allows superposition
  - Represent multiple wires simultaneously on the same fixed hardware (a weighted sum of labels)
  - Scaling problem becomes a signal to noise problem
  - This does not look like a digital system

What are Vector Symbolic Architectures (VSA)?

- Family of connectionist architectures well suited to “symbolic” processing
- Representations are high-dimensional vectors (~10,000) of low resolution values
  - Vector values are equivalent to systematically constructed labels with scalar magnitudes
    - Represent complex data structures in a fixed vector space
    - Vectors represent, not individual vector elements
- Small, fixed set of vector operators (M,A,P)

Analogue computing on weighted, systematic labels

The blessing of dimensionality

- In a high-dimensional vector space:
  - # orthogonal directions = dimension k
  - # approximately orthogonal directions = exponential k
- Two randomly chosen vectors are approximately orthogonal with very high probability
  - Random vectors can be used as symbols (unique labels)
- Vectors can be systematically, arithmetically composed to represent composite structures

Why are VSA good?

- Useful properties of high-dimensional vector spaces (Kanerva, 2009)
- Implementable as realistic connectionist systems (Eliasmith & Anderson, 2003)
- Robust to noise (30% corruption tolerable)
- Graceful degradation
- Operators not learned
- Operators blind to interpretation of vectors

How are VSA used?

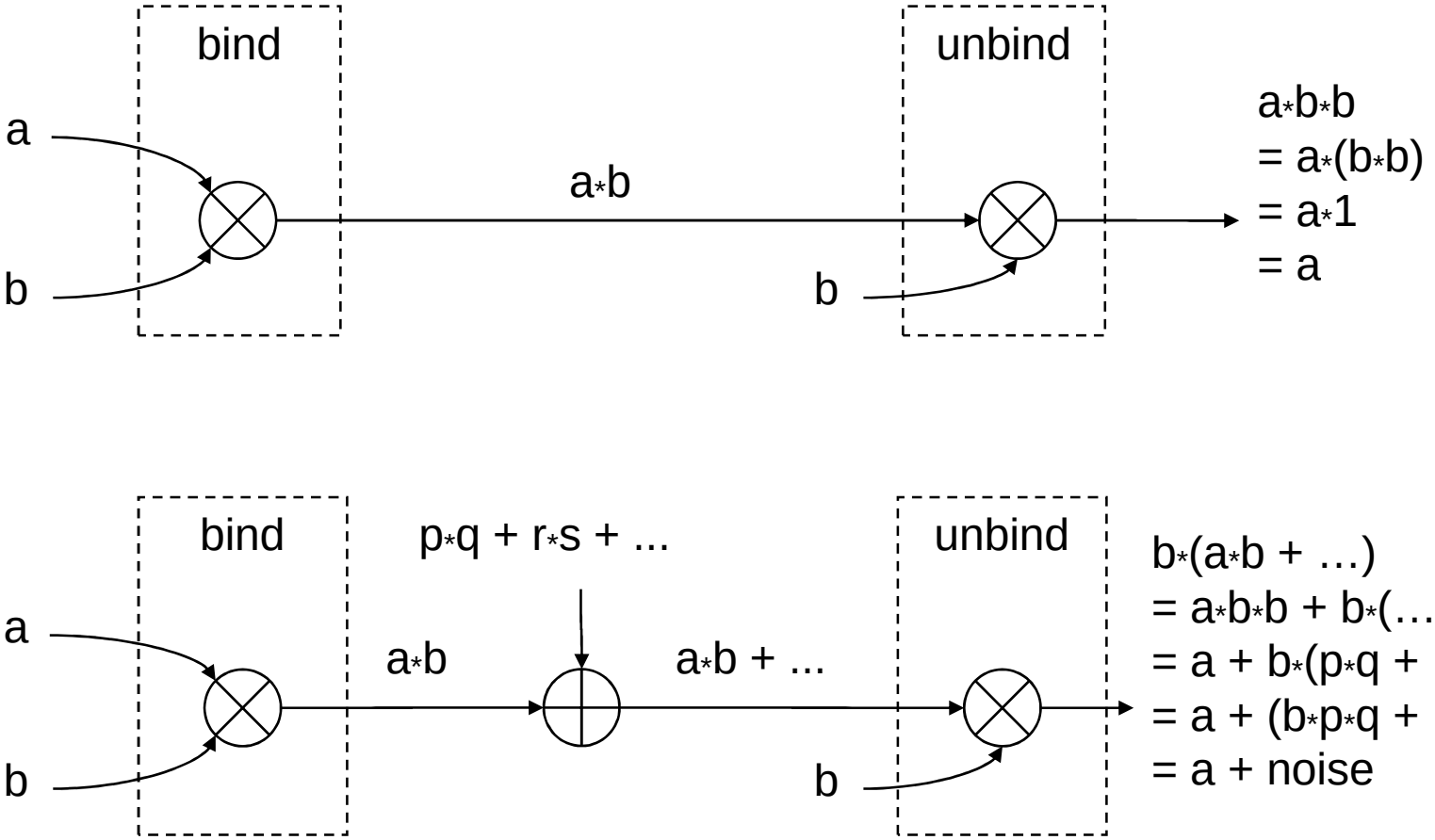
VSA vectors and operators as the bricks and mortar of computational circuits

- Multiply ***** : bind, query, apply mapping
- Add **+** : superpose, add to set
- Permute $\mathbf{P_i( )}$: quote

Design a fixed circuit that calculates the desired result by virtue of it's structure

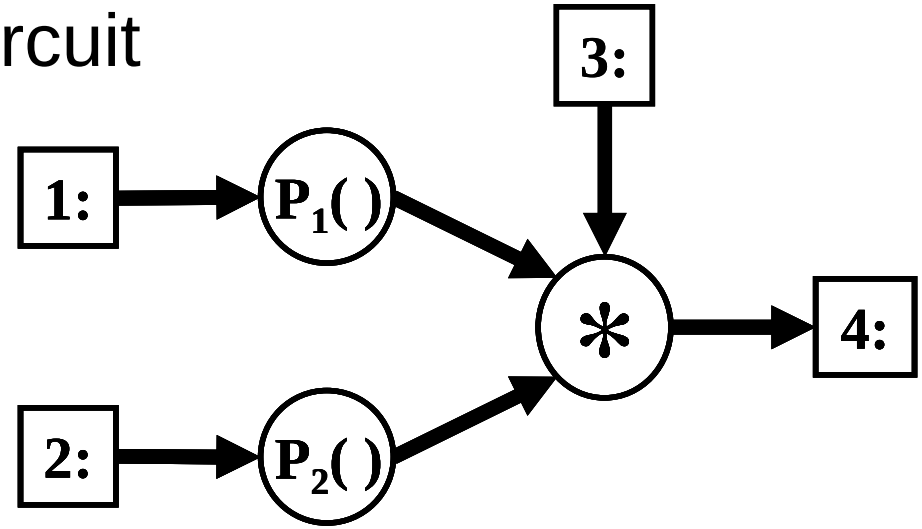
How are VSA used?

Example of binding, unbinding, superposition



How are VSA used?

Multiset intersection circuit



1: $k_1A + k_2B + k_3C$

2: $k_4A + k_5B + k_6D$

3: a vector that is a specific function of the “basis”

4: $k_1k_4A + k_2k_5B +$ *noise*

Substitution via VSA

Substitution is effectively a primitive operator

- Substitution is central to symbolic computing

The product of two vectors can be applied as a substitution operator:

(A*B) (substitutes **A** for **B** and vice versa)

(A*B) * (A*X) = A*B*A*X = (A*A)*B*X = (B*X)

Subgraph isomorphism via VSA

Translate the replicator equation algorithm to a VSA implementation (Gayler & Levy, 2009)

- How to represent the data structures?
- How to operate on data structures?
- Preserve the replicator equation dynamics!
- Embody the algorithm as a fixed circuit

Represent graphs as VSA vectors

- vertices: randomly chosen vectors *A*, *B*, …
- vertex sets : (*A* + *B* + *C* + *D*).
- edges: *B***C*, B*D, …
- edge sets: (*B***C* + B*D)

- vertex mappings: A*P, B*Q, ...
- state vector: k_1A*P + k_2A*Q …
- compatibility: A*P*B*Q + A*P*B*R + …

Construct the initial values

Dynamic construction of structures

- initial state vector

$$x=(A+B+C+D)*(P+Q+R+S)$$
$$=A*P+A*Q+\ldots+B*P+B*Q+\ldots+D*R+D*S$$

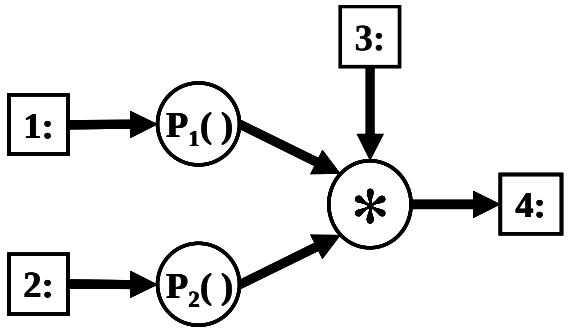
- compatibility vector

$$w=(B*C+B*D)*(Q*R+Q*S)+$$
$$(A*B+A*C+A*D+C*D)*(P*Q+P*R+P*S+R*S)$$
$$=B*C*Q*R+B*C*Q*S+B*D*Q*R+B*D*Q*S$$
$$+A*B*P*Q+A*B*P*R+\ldots+A*B*R*S$$
$$+A*C*P*Q+A*C*P*R+\ldots+A*C*R*S+\ldots+C*D*R*S$$

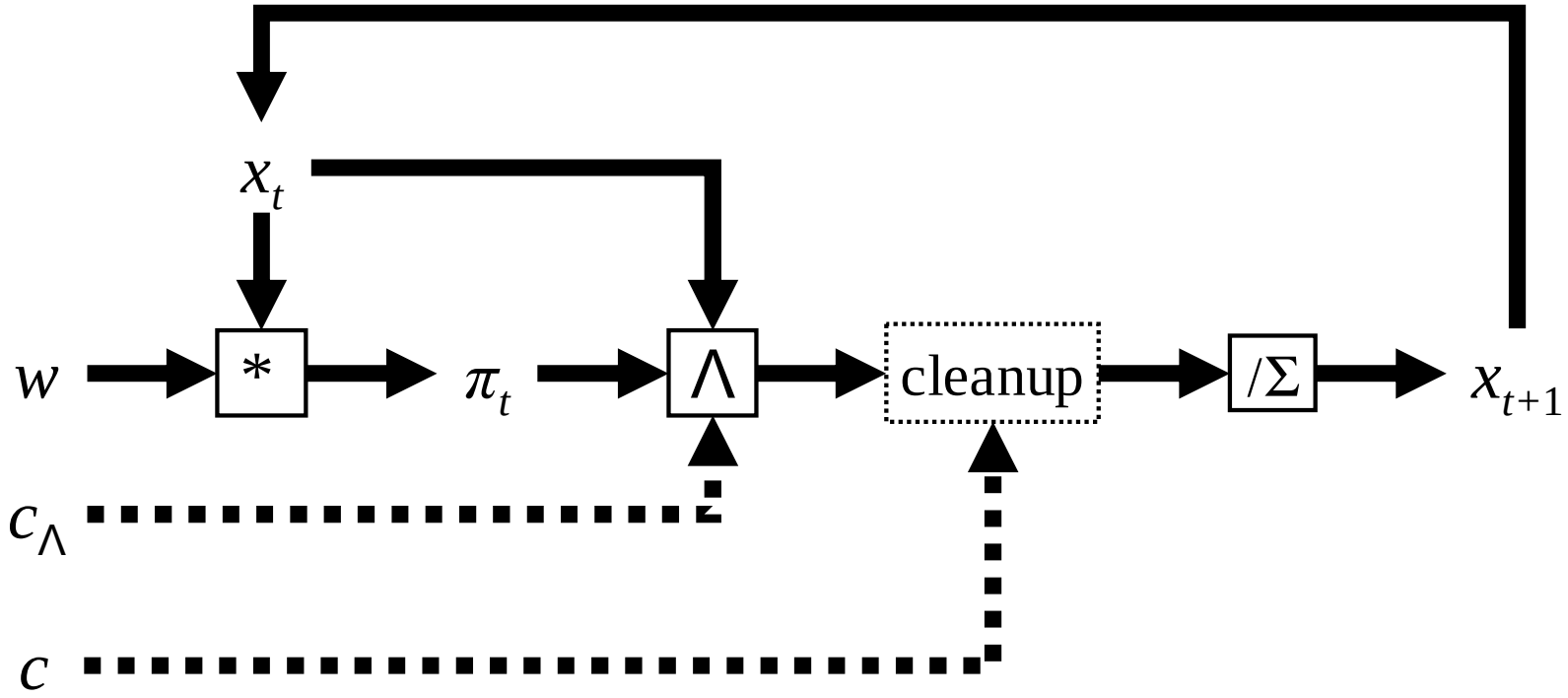
Each requires only a constant time operation (algebraic parallelism)

Evidence propagation & update

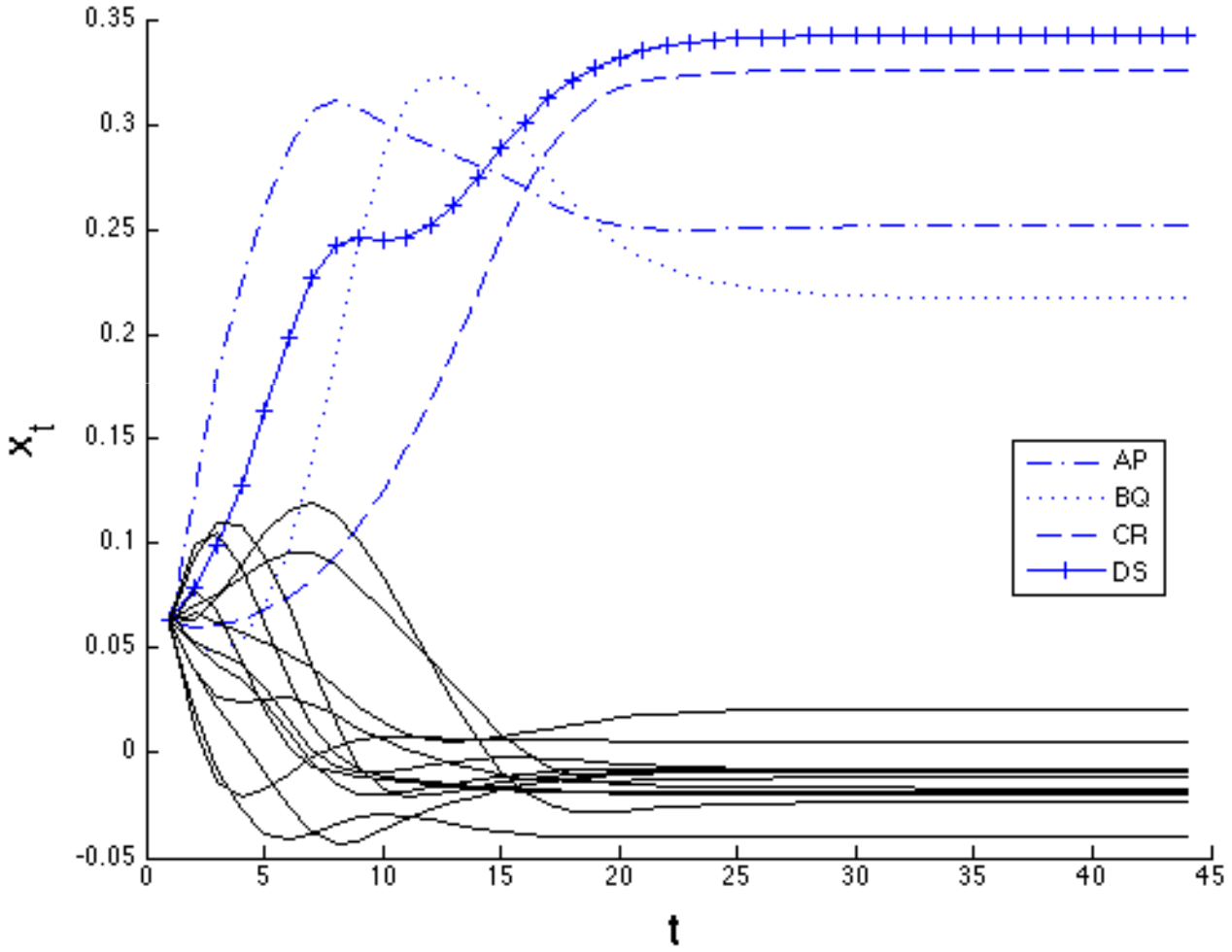
- evidence propagation
  - Product of state and compatibility vectors (constant-time operation)
- update operation
  - Apply previously introduced multiset intersection circuit (constant-time operation)



Fully distributed architecture



Settling of distributed system



Salient points

- It works! (Only tested on a few graphs)
- Hardware is fixed
- Problem is loaded as patterns of activation
- Those patterns are calculated holistically from the graph vector representations
- VSA
  - Analogue dynamics over weighted sums of labels
  - Labels are systematically constructed
  - Labels are able to b decomposed