Looking Good, Behaving Well

PhD Defence

Michael Westergaard

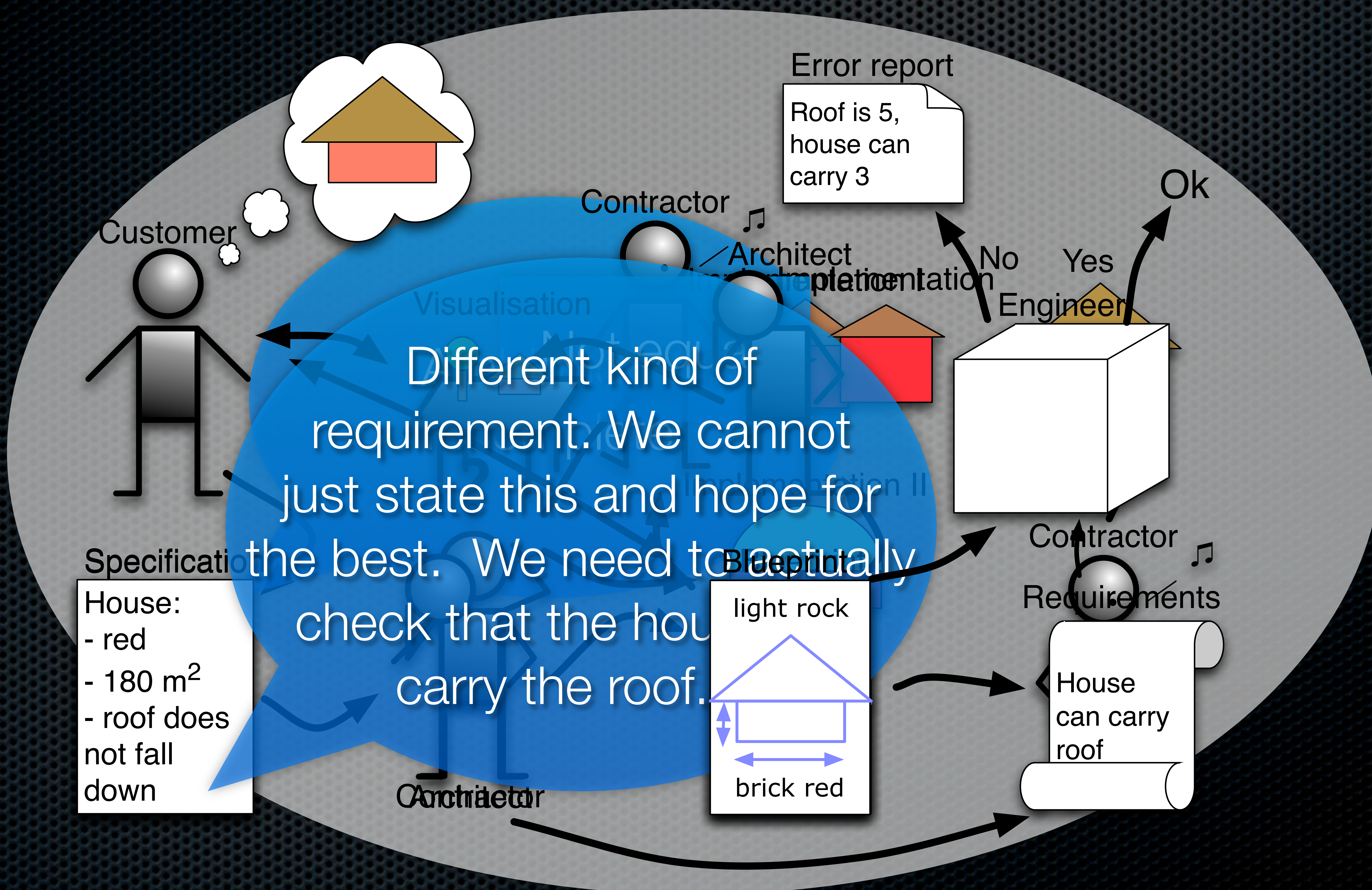
Department of Computer Science

University of Aarhus

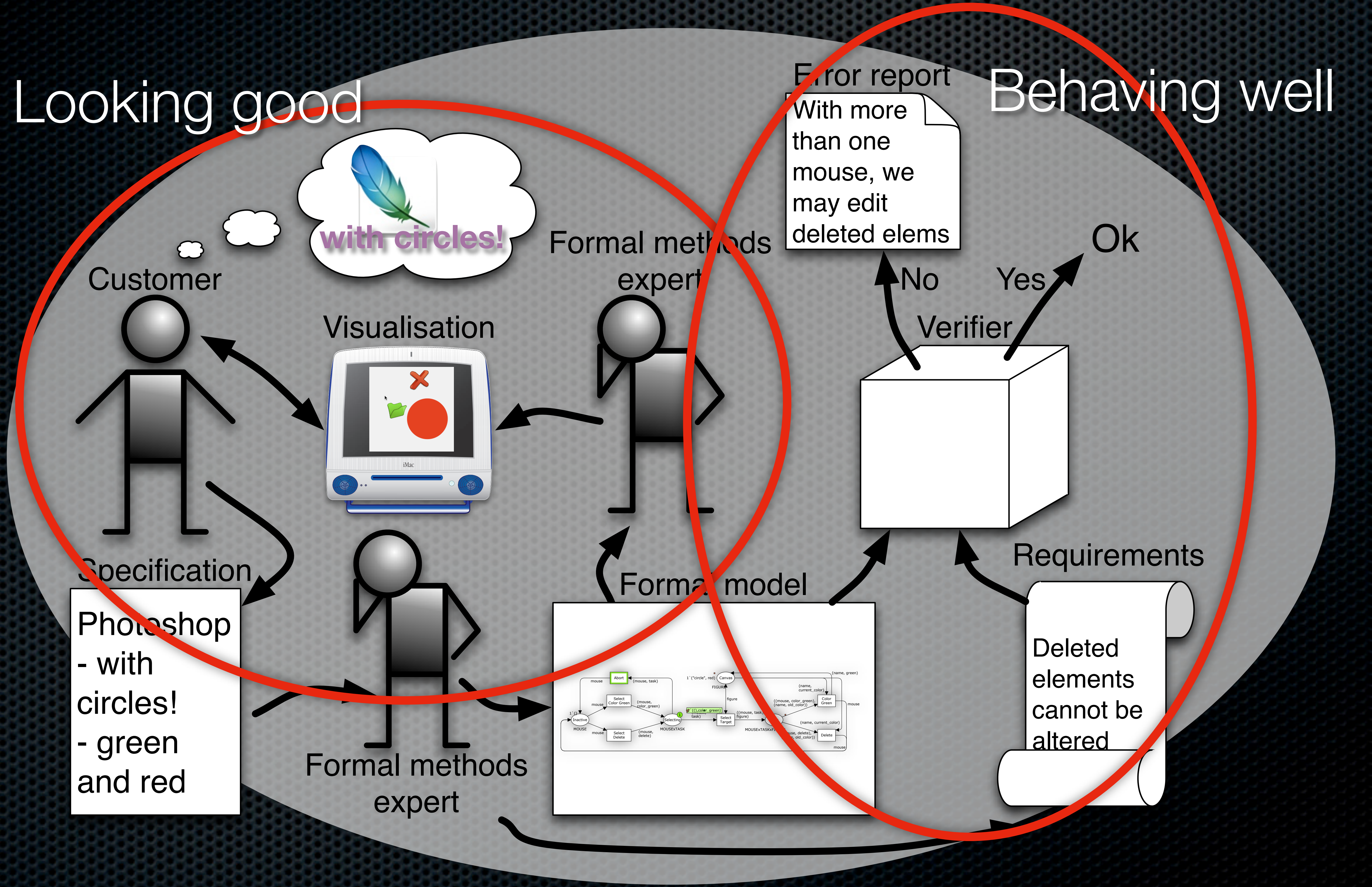
Motivation – Horror Stories

- Disasters caused by failure of computer software:
  - Ariane 5 lifting rocket (economical loss)
  - Therac-25 radiation machine (loss of human lives)
  - Ignored hole in ozone layer (worsening global warming)
- Can we prevent such disasters or at least reduce the probability of such disasters?

Motivation – Inspiration



Motivation – The Real One

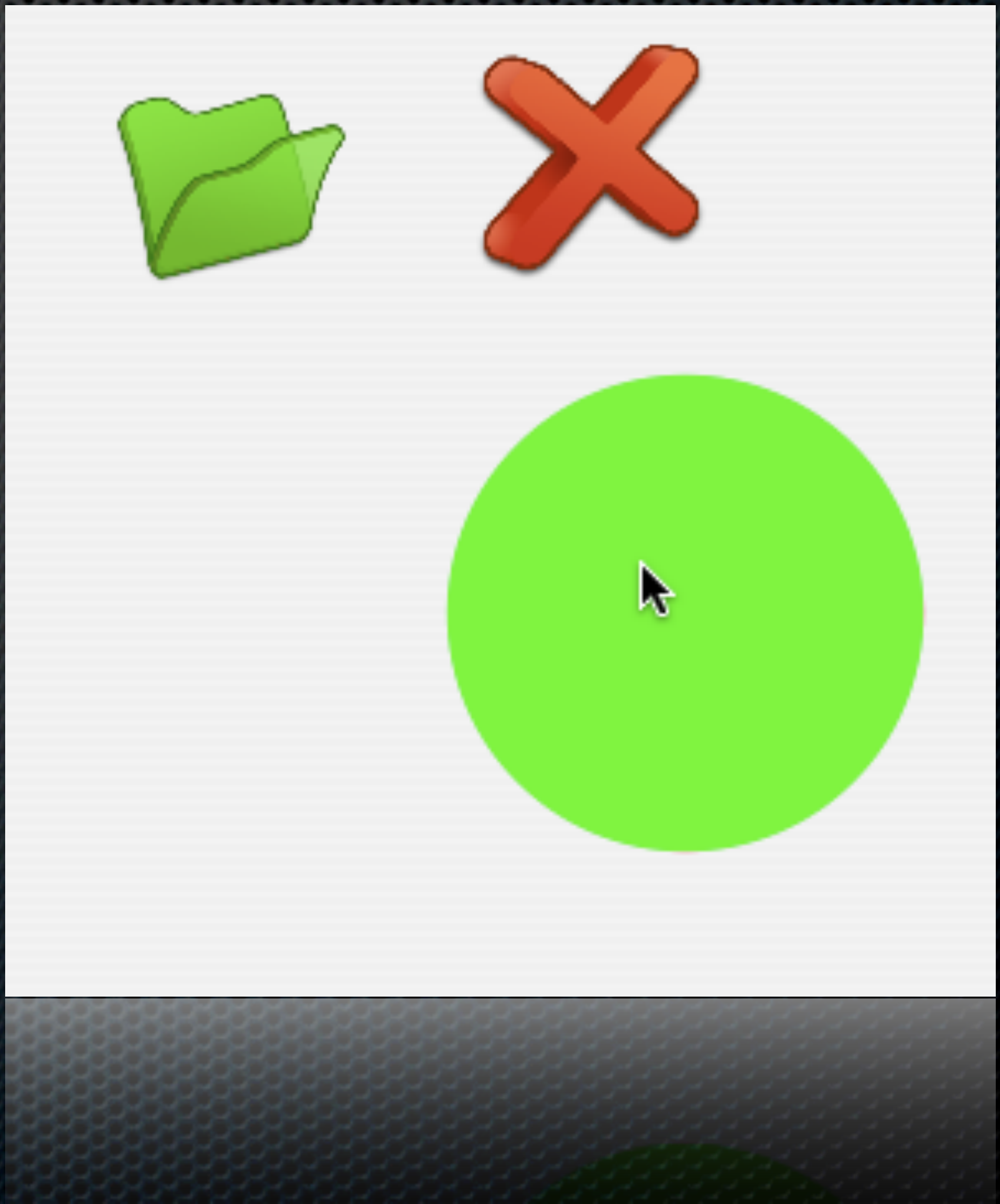


Overview

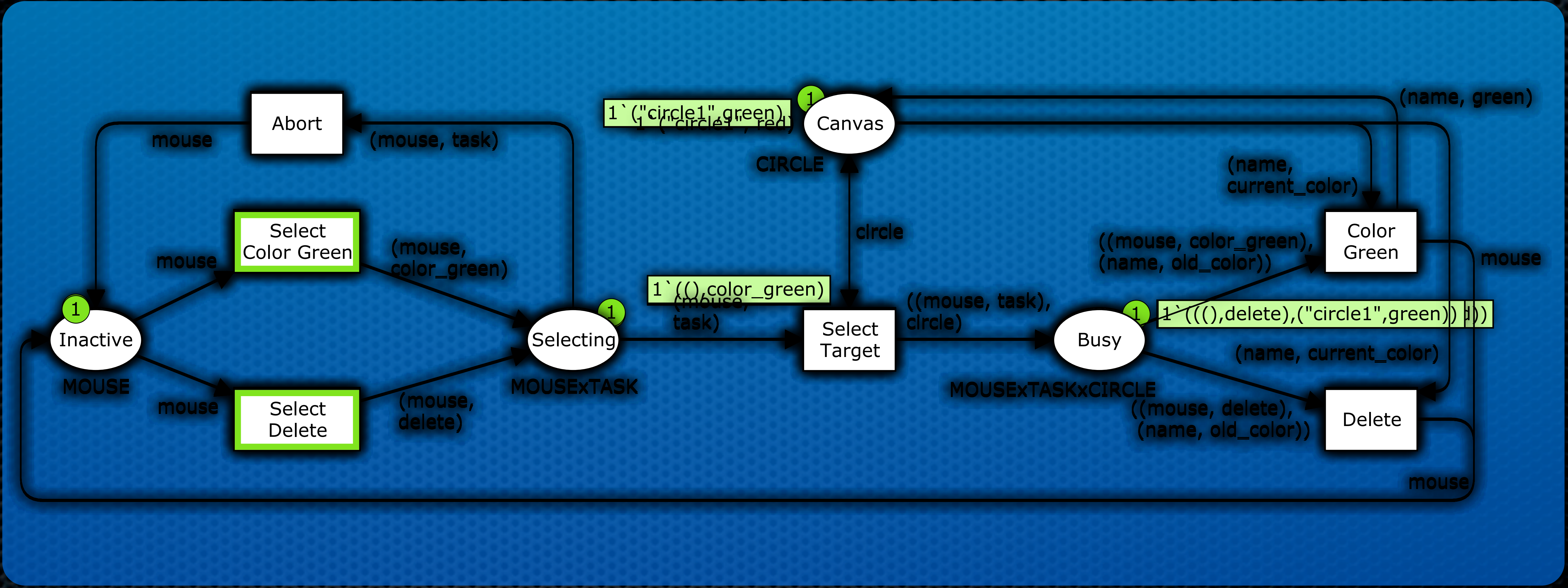
- Example
- Visualisation – Looking Good
- Verification – Behaving Well
- Impact and Future Work

Example – Specification

- Drawing program: Photoshop – with circles!
- We want to support
  - Colouring circles green
  - Deleting circles



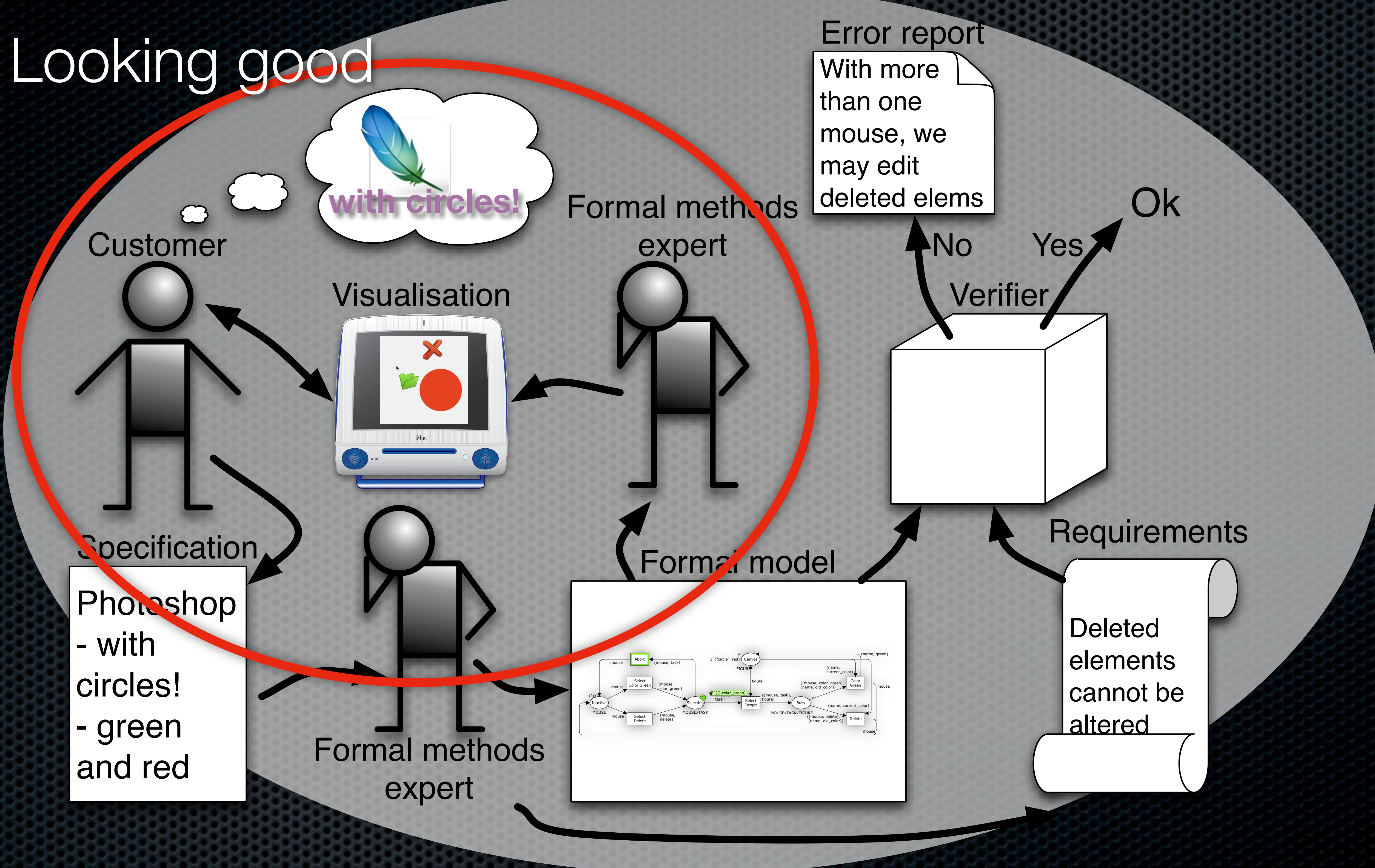
Example – Formal Model



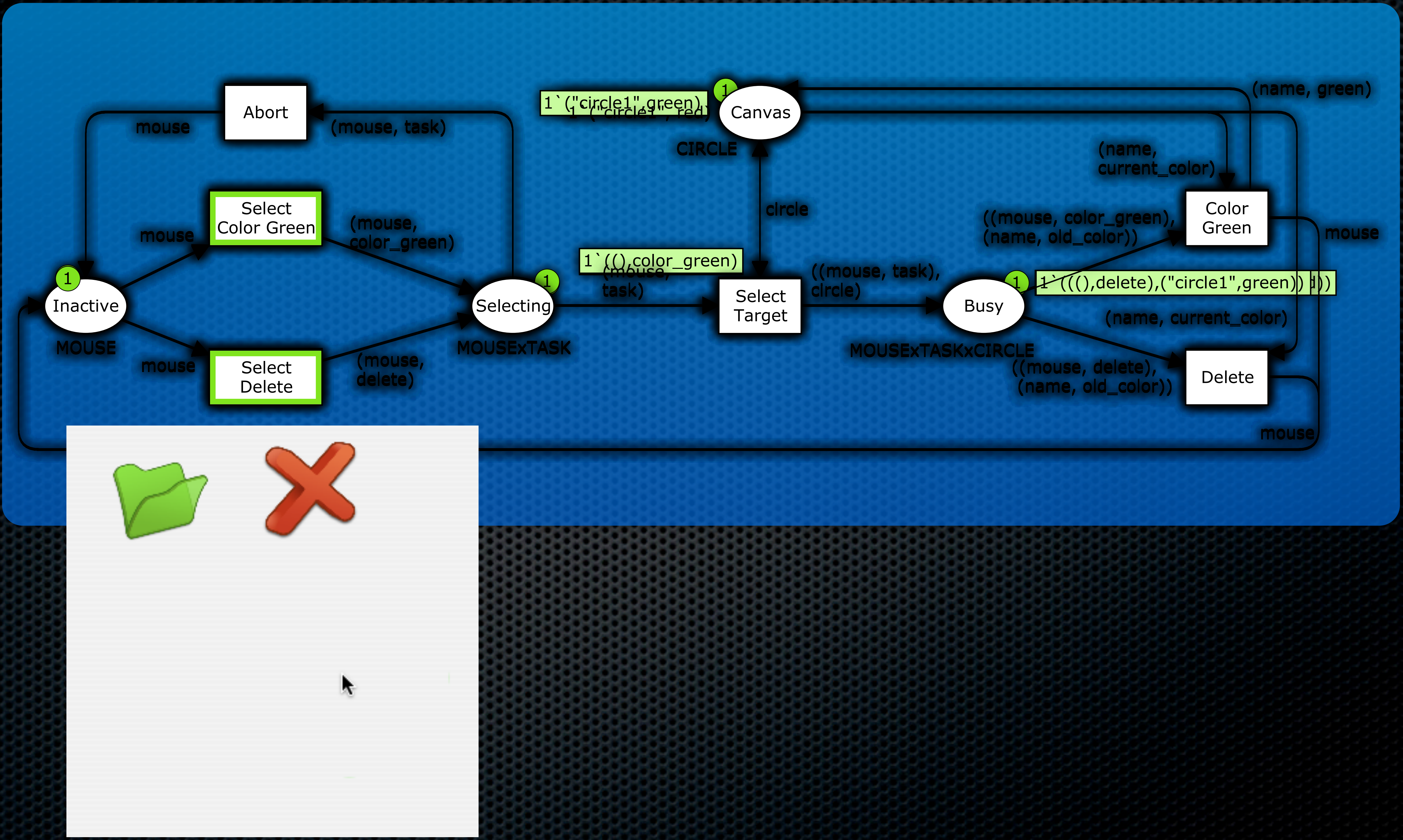
Visualisation Looking Good

- While the model is graphical, it may not be easy to explain it to a user
- The task does not get easier if the model is more complex, e.g., if we allow colouring circles red, creating new circles, or moving circles around

Looking good



Example – Visualisation

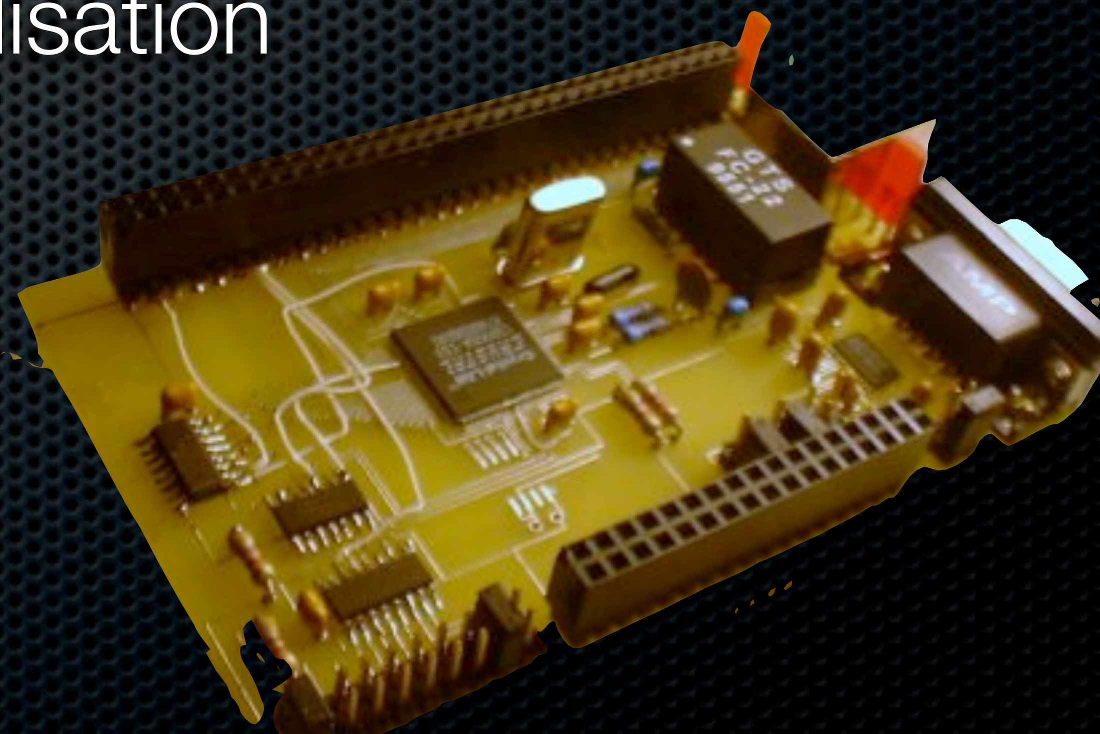


The BRITNeY Suite

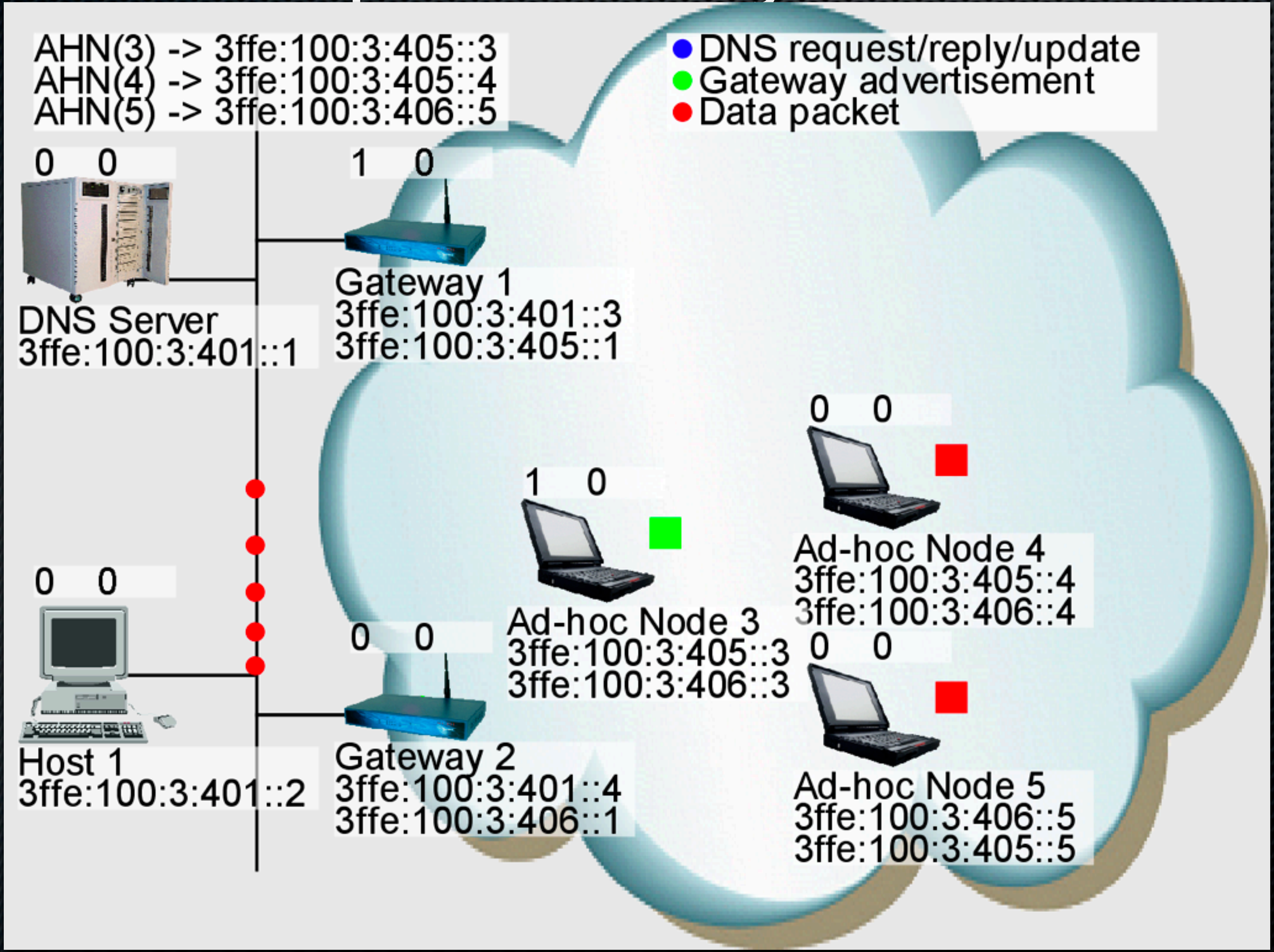
- M. Westergaard and K.B. Lassen. The BRITNeY Suite Animation Tool. In *Proc. of ATPN'06*, volume 4024 of *LNCS* pages 431–440. Springer-Verlag, 2006.
  - The BRITNeY Suite supports visualisation of formal models

Model-based Prototyping of an Interoperability Protocol

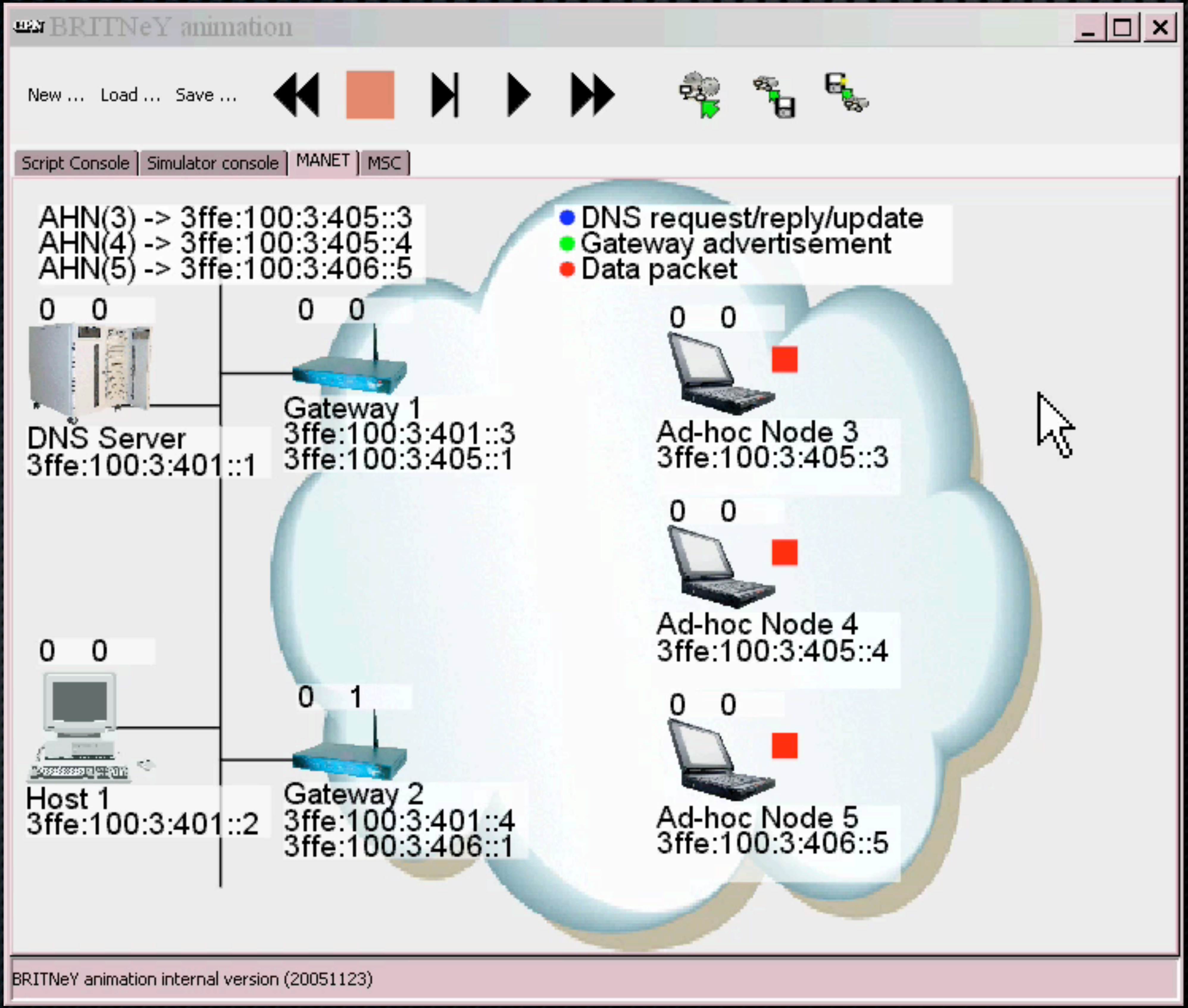
- L.M. Kristensen, M. Westergaard, and P.C. Nørgaard. Model-based Prototyping of an Interoperability Protocol for Mobile Ad-hoc Networks. In *Proc. of IFM'05*, volume 3771 of *LNCS*, pages 266–286, Springer-Verlag, 2005.
  - Using formal models and visualisation to rapidly develop a prototype implementation of a real-life network protocol



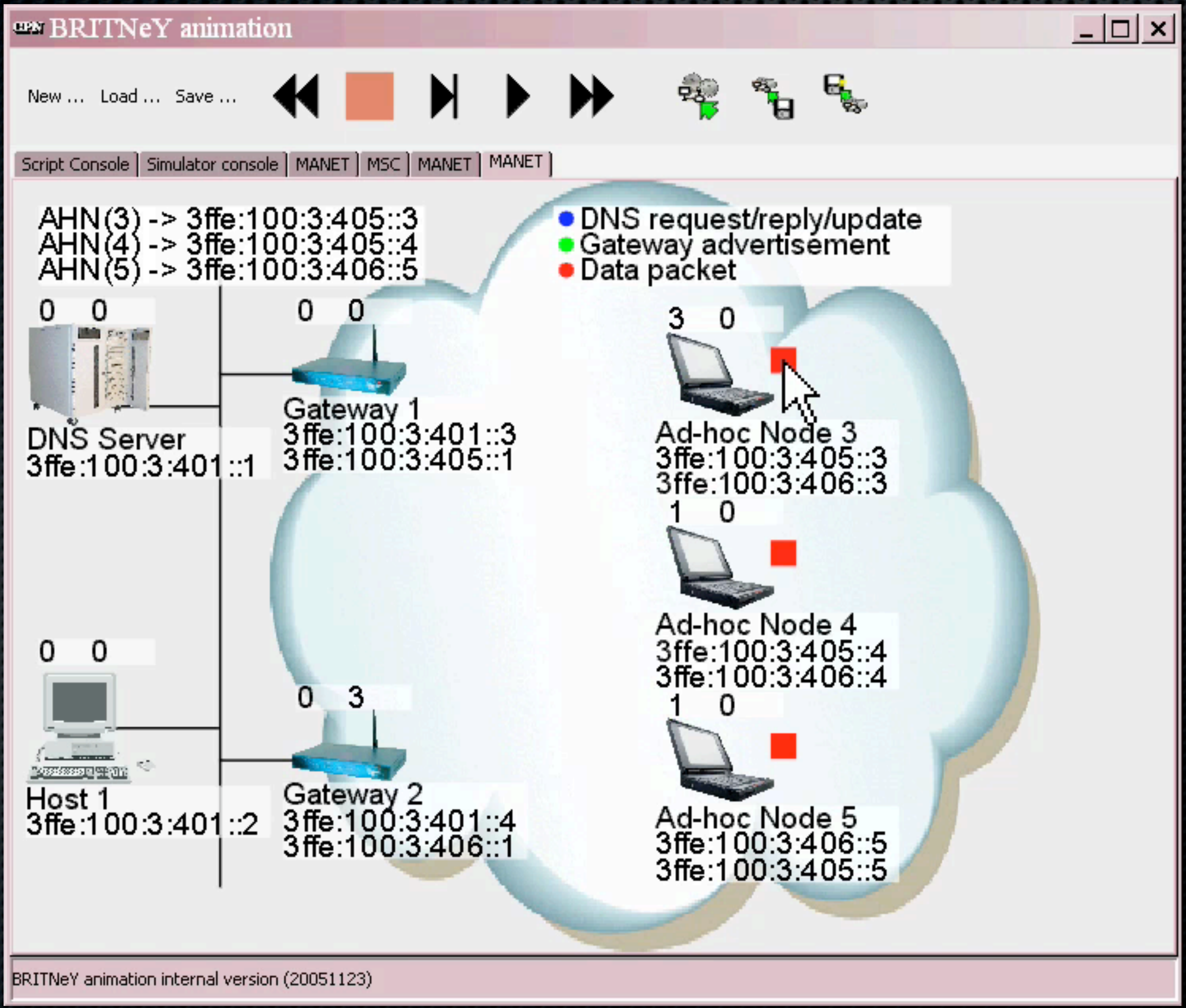
Model-based Prototyping of an Interoperability Protocol



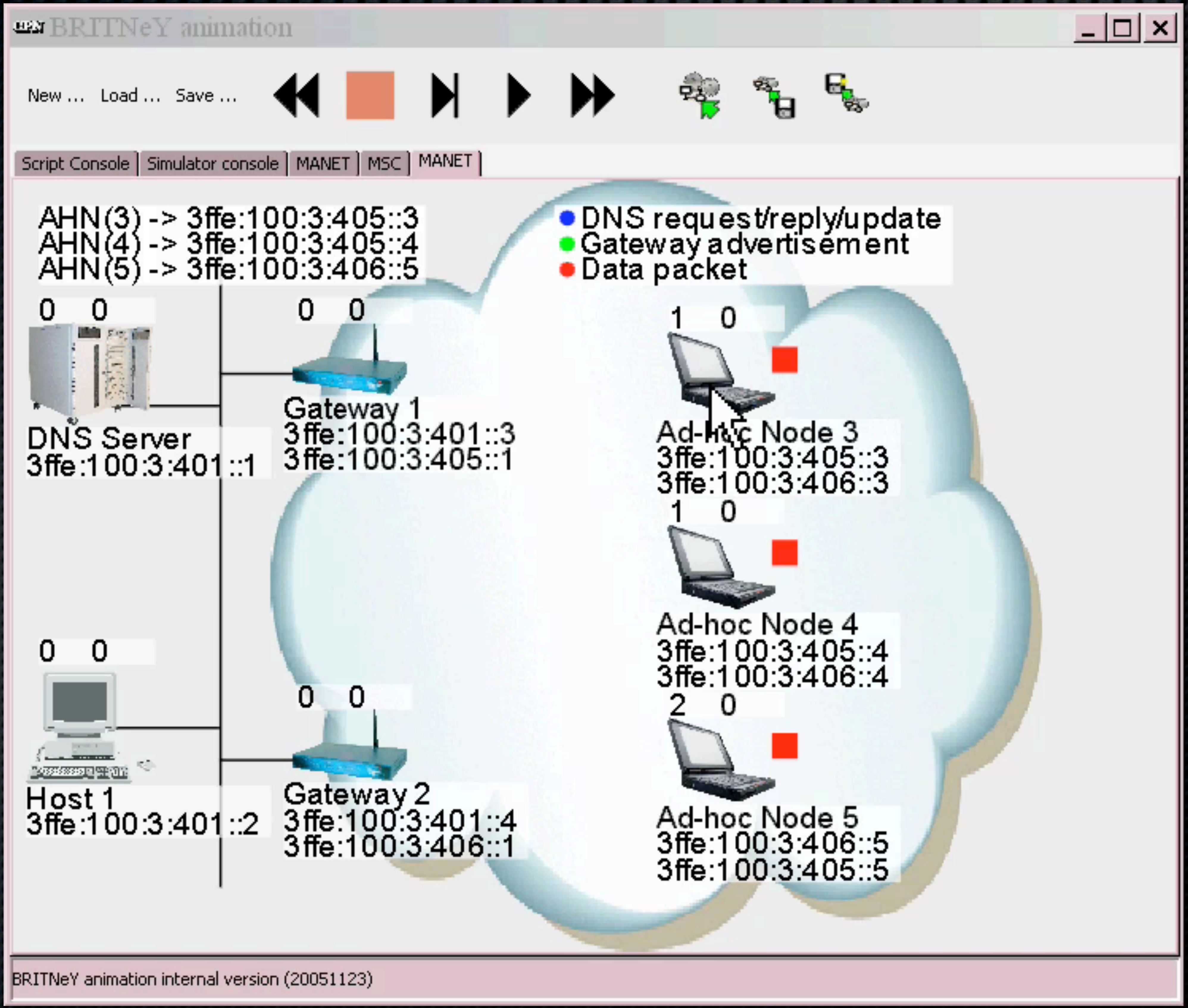
Gateway Advertisements



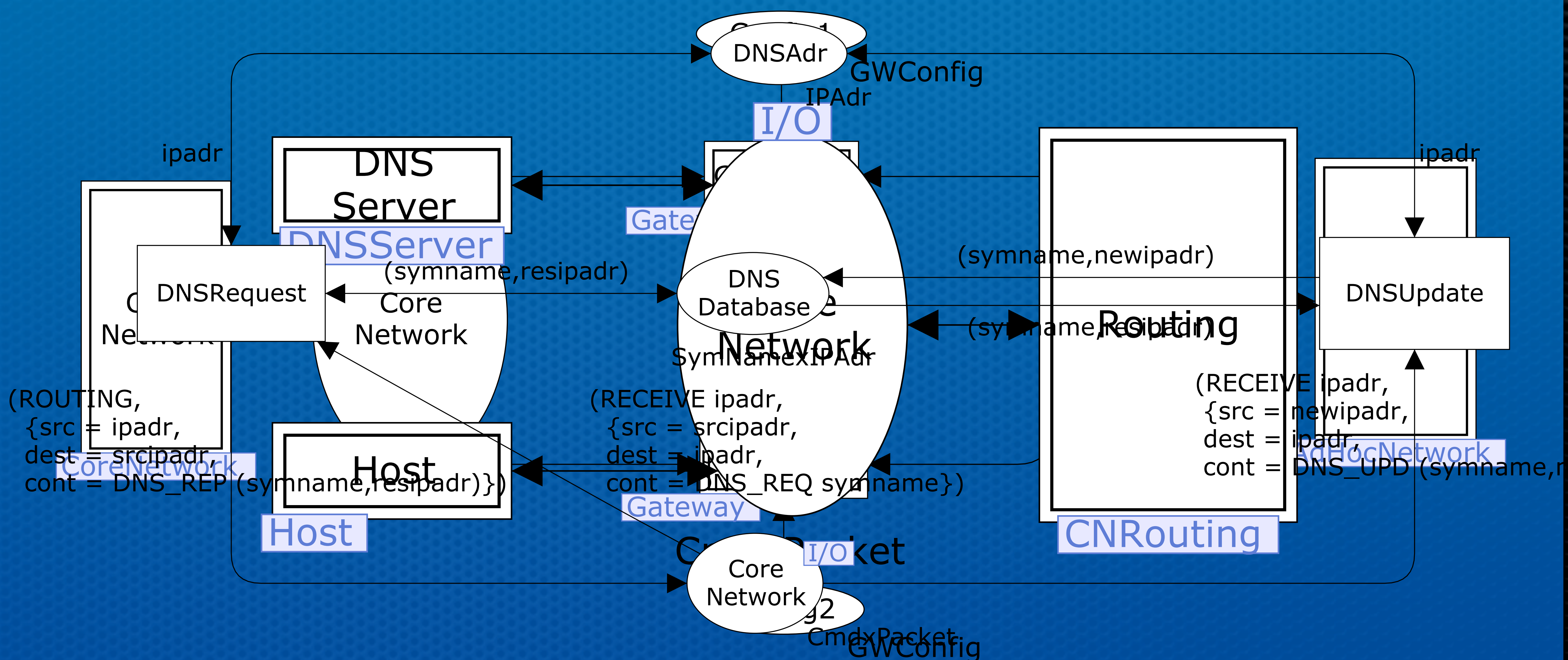
Sending Data



Mobility and DNS Update



Interoperability Protocol Formal Model



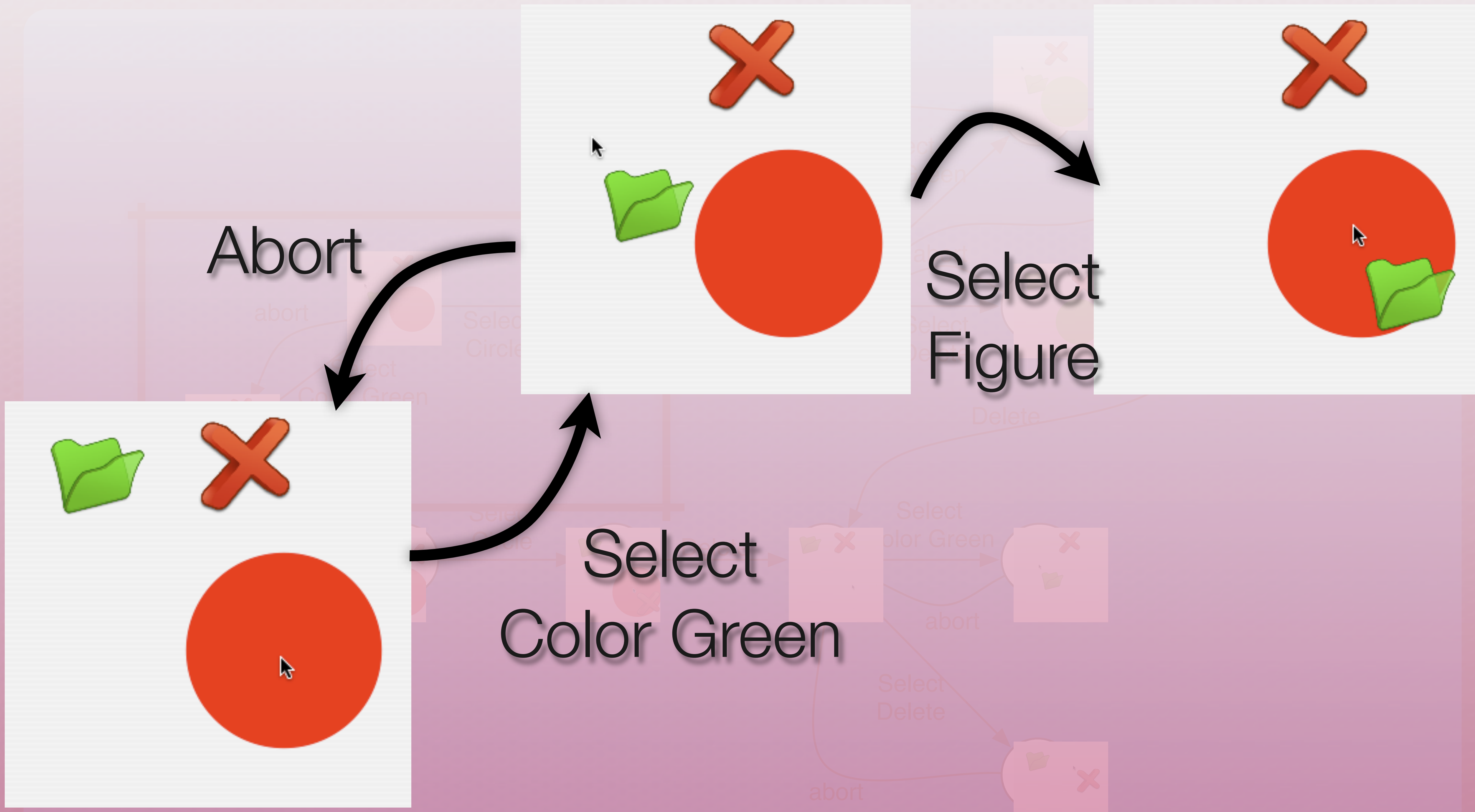
Model-based Prototyping of an Interoperability Protocol

- It is possible to create a prototype using formal models and visualisation
- The behaviour of the prototype is defined by a formal model which can be used for further refinement and/or analysis
- Using the visualisation it proved possible to demonstrate the model to management

Problems of Visualisation Tools

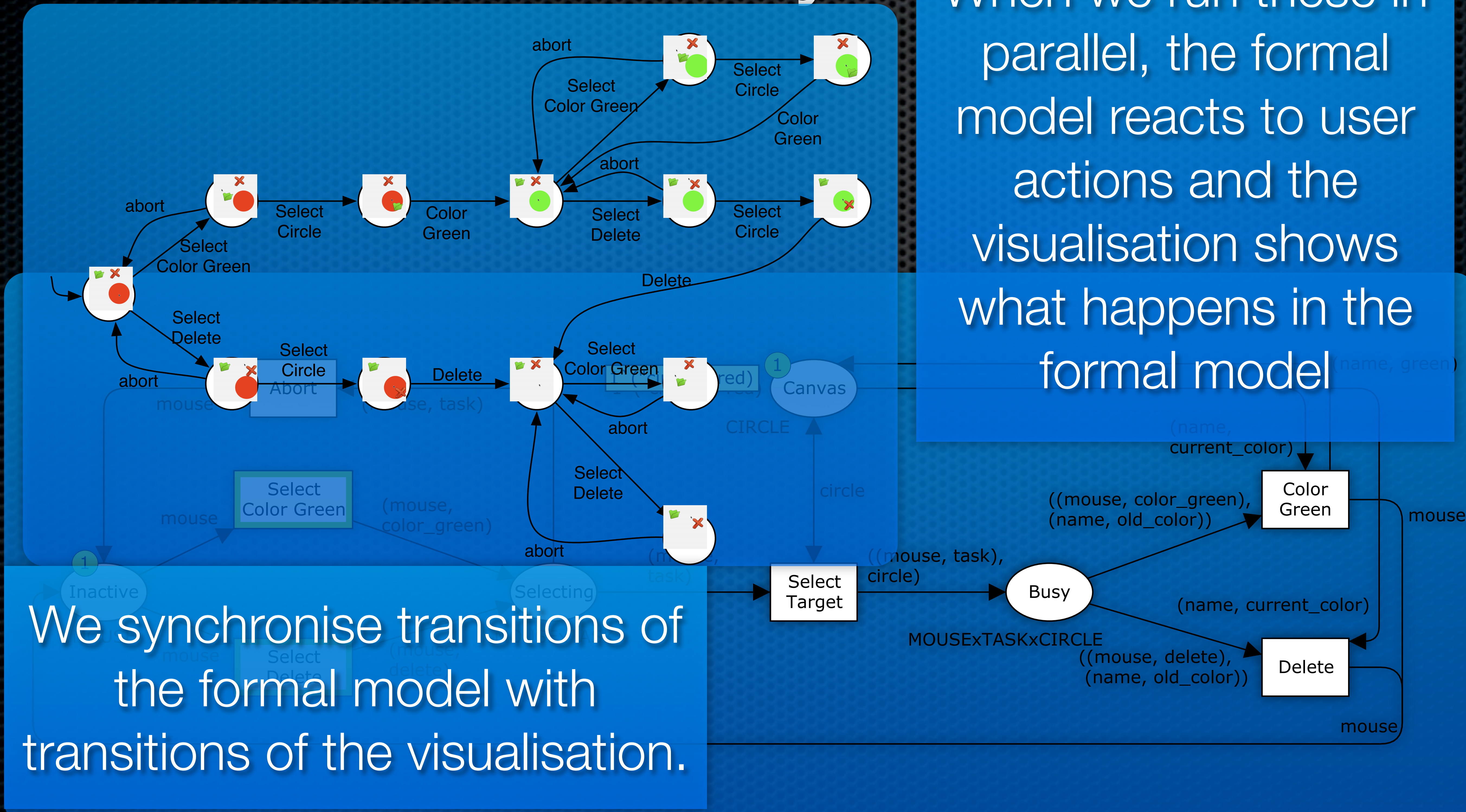
- Visualisation of formal models is usually added to tools for formal modelling in an ad-hoc manner
  - Visualisation is tied to a specific formalism or tool
  - It is not possible to extend/modify the functionality of visualisations
- M. Westergaard. A Game-theoretic Approach to Behavioural Visualisation. In *Pre-proc. of FMIS'07*, Queen Mary, University of London, Dept. of Computer Science, Technical Report number RR-07-08, 2007.
  - Introduces a formal framework for visualisations

Viewing a Visualisation as a Transition System



Running Visualisation and Formal Model Synchronously

When we run these in parallel, the formal model reacts to user actions and the visualisation shows what happens in the formal model



We synchronise transitions of the formal model with transitions of the visualisation.

Gist of Definition of a Visualisation

- Given two transition systems, *V* and *M*, *V* can be used as visualisation for *M* if the states and transitions of *V* and *M* can be related in a way such that whenever two states are related
  - if it is possible to take a step in *V* it is possible to take a related step in *M* and the resulting states are related
  - if it is possible to take a step in *M* it is possible to take a related step in *V* and the resulting states are related
- We do not need to represent *V* and *M* before simulation; we only need to be able to generate them on-the-fly

…or…

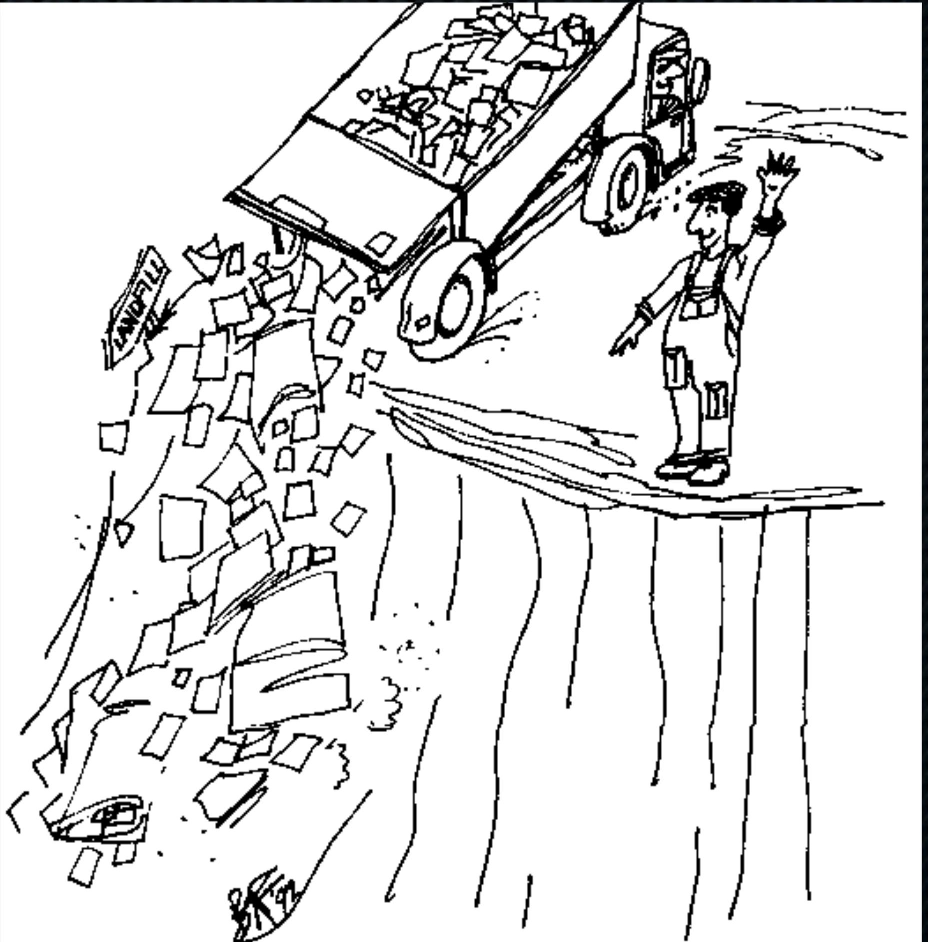
Whenever we do something in the visualisation, it must be possible to reflect it in the model

Whenever the model does something, it must be possible to show it in the visualisation

In the report we give the complete definition, and use a separation of user actions from system actions to impose a clear flow of information

Contributions

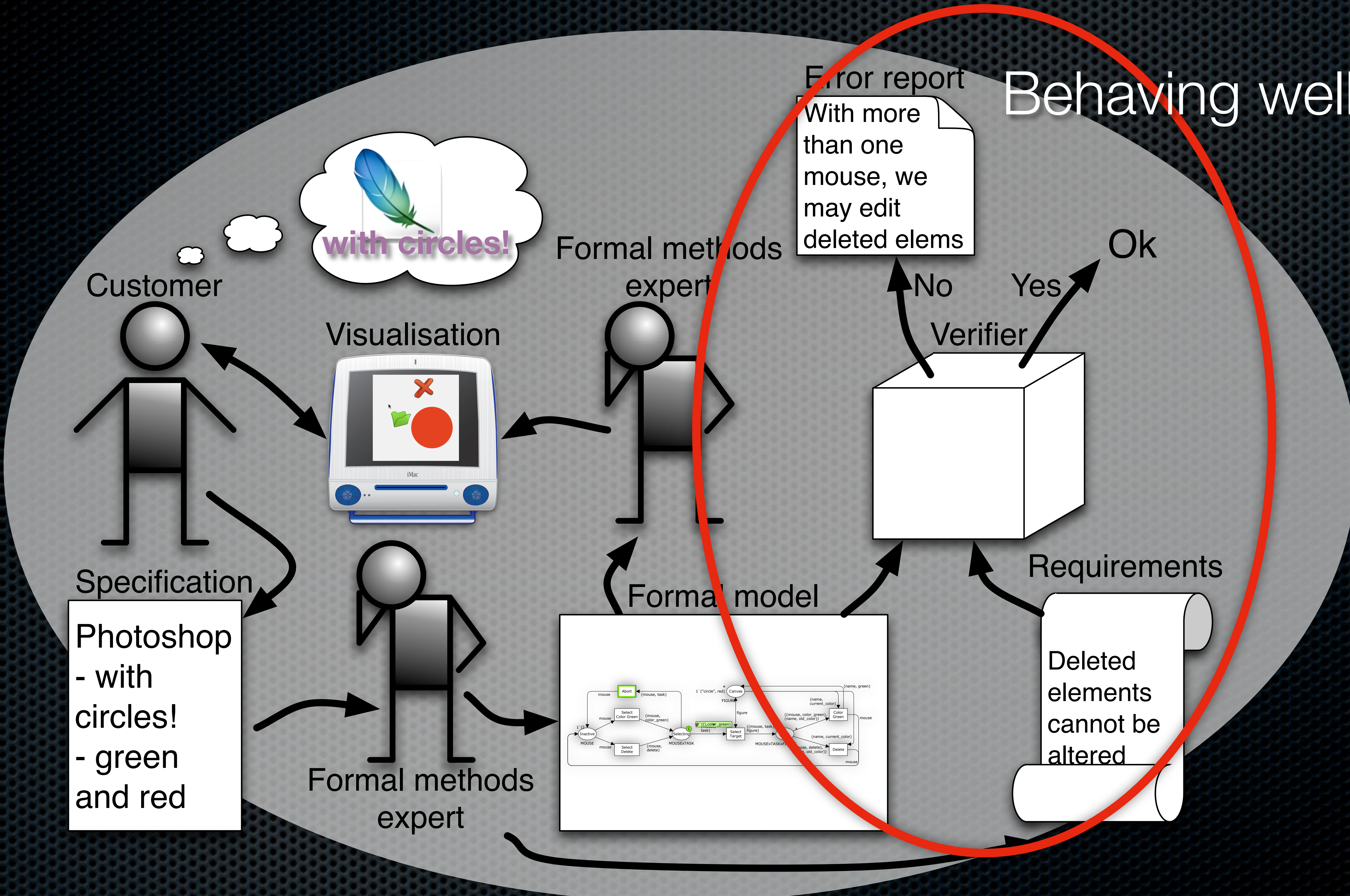
- Development of the BRITNeY Suite for visualisation
  - Extended to provide generic platform for experiments with the coloured Petri net formalism
- Use of the BRITNeY Suite in a real-life case study
- Development of formal framework for visualisation
  - Allows us to detach visualisations from the formal model and supporting tools

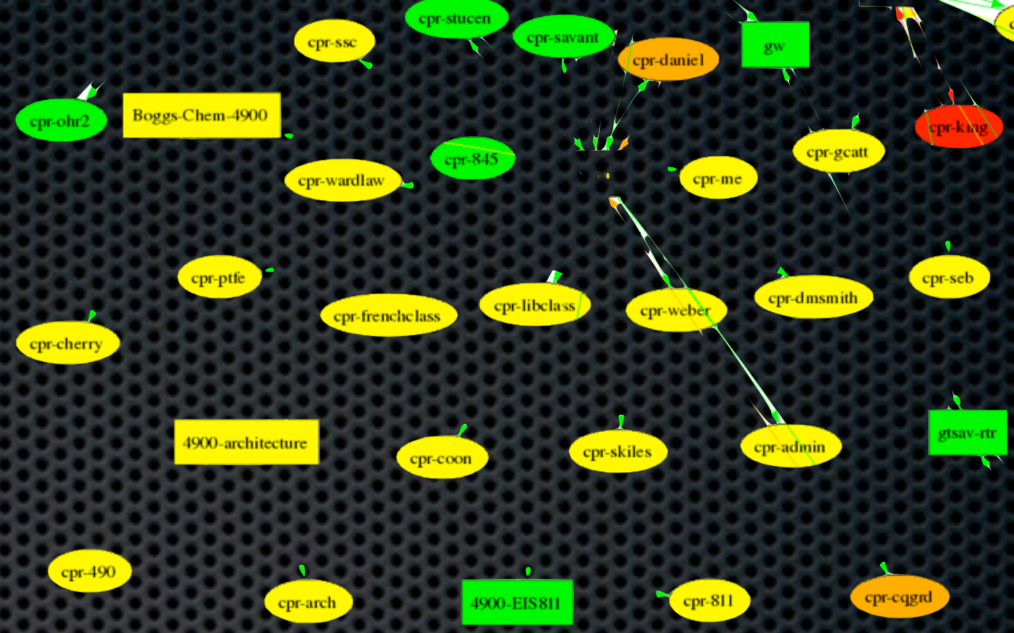


A contribution to the field

Verification – Behaving Well

- Reconsider the drawing program
- Is it possible to alter a circle that has been deleted?
- Sporadic testing suggests that the answer is no
  - ...but how can we be *sure*?

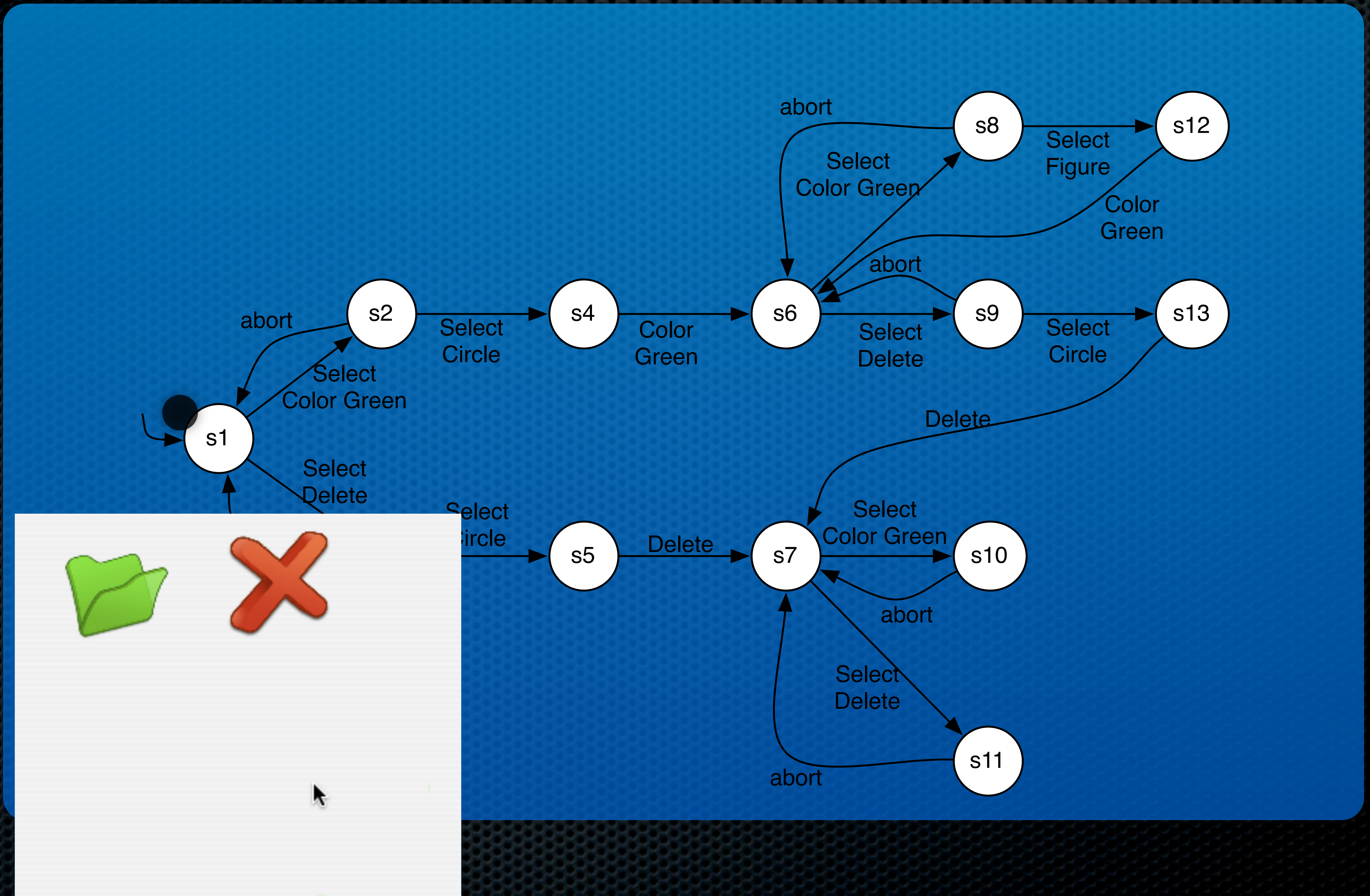

Behaving well
Error report
With more than one mouse, we may edit deleted elems
Formal methods expert
with circles!
Customer
No
Yes
Ok
Visualisation
Verifier
Specification
Photoshop - with circles! - green and red
Formal model
Requirements
Deleted elements cannot be altered
Formal methods expert



Approach

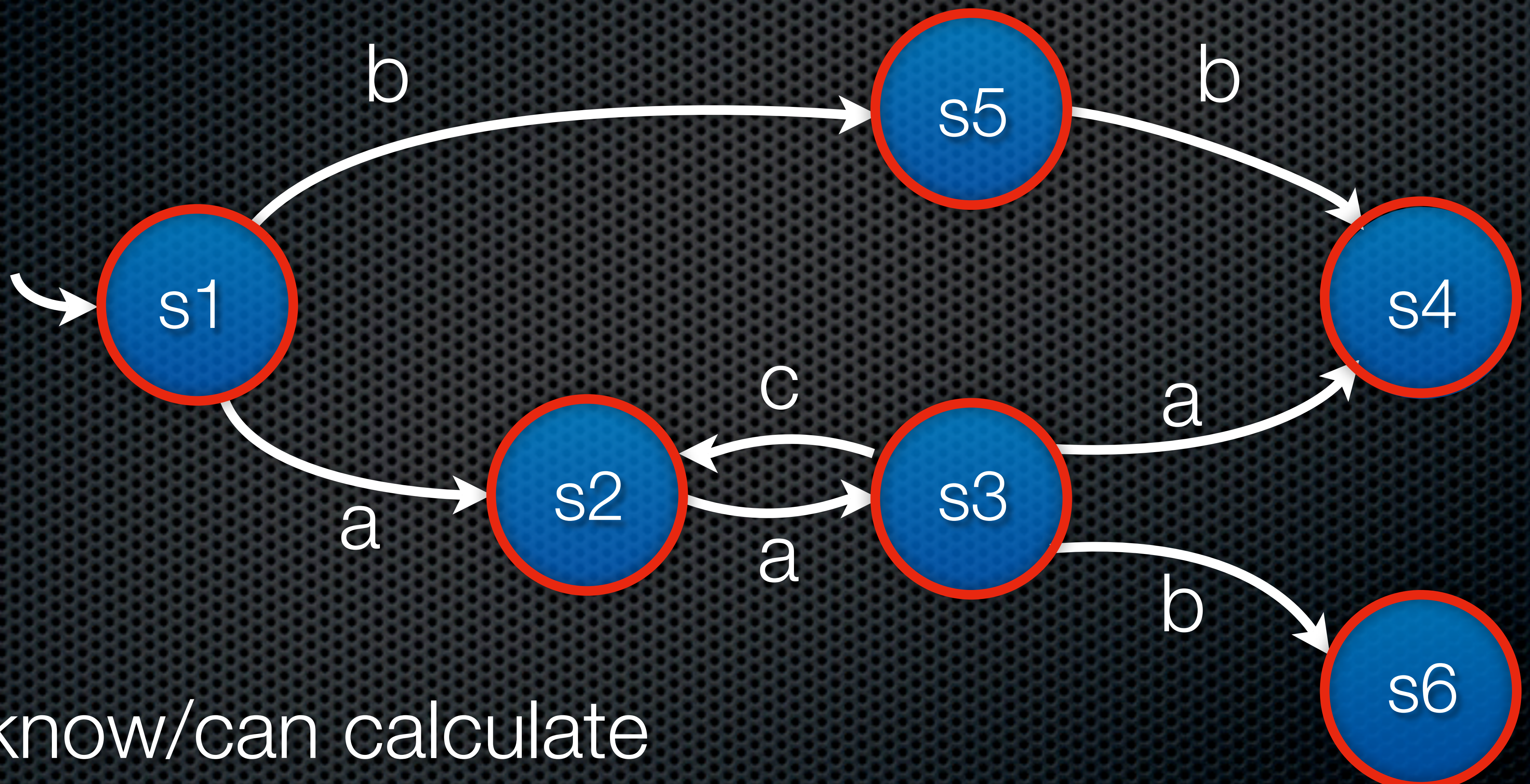
- Represent the entire behaviour of the system as a graph (a reachability graph)
  - Each node in the graph represents a possible state of the system
  - Each labelled edge in the graph represents that it is possible to go from the source state to the destination state using the transitions represented by the label
- Traces in the reachability graph correspond to executions of the formal model

Example



abort
s8
Select Figure
s12
Select Color Green
Color Green
abort
abort
s2
Select Circle
s4
Color Green
s6
Select Delete
s9
Select Circle
s13
Select Color Green
s1
Delete
Select Delete
Select
s5
Delete
s7
Select Color Green
s10
abort
Select Delete
s11
abort

Calculating Reachability Graphs



We know/can calculate

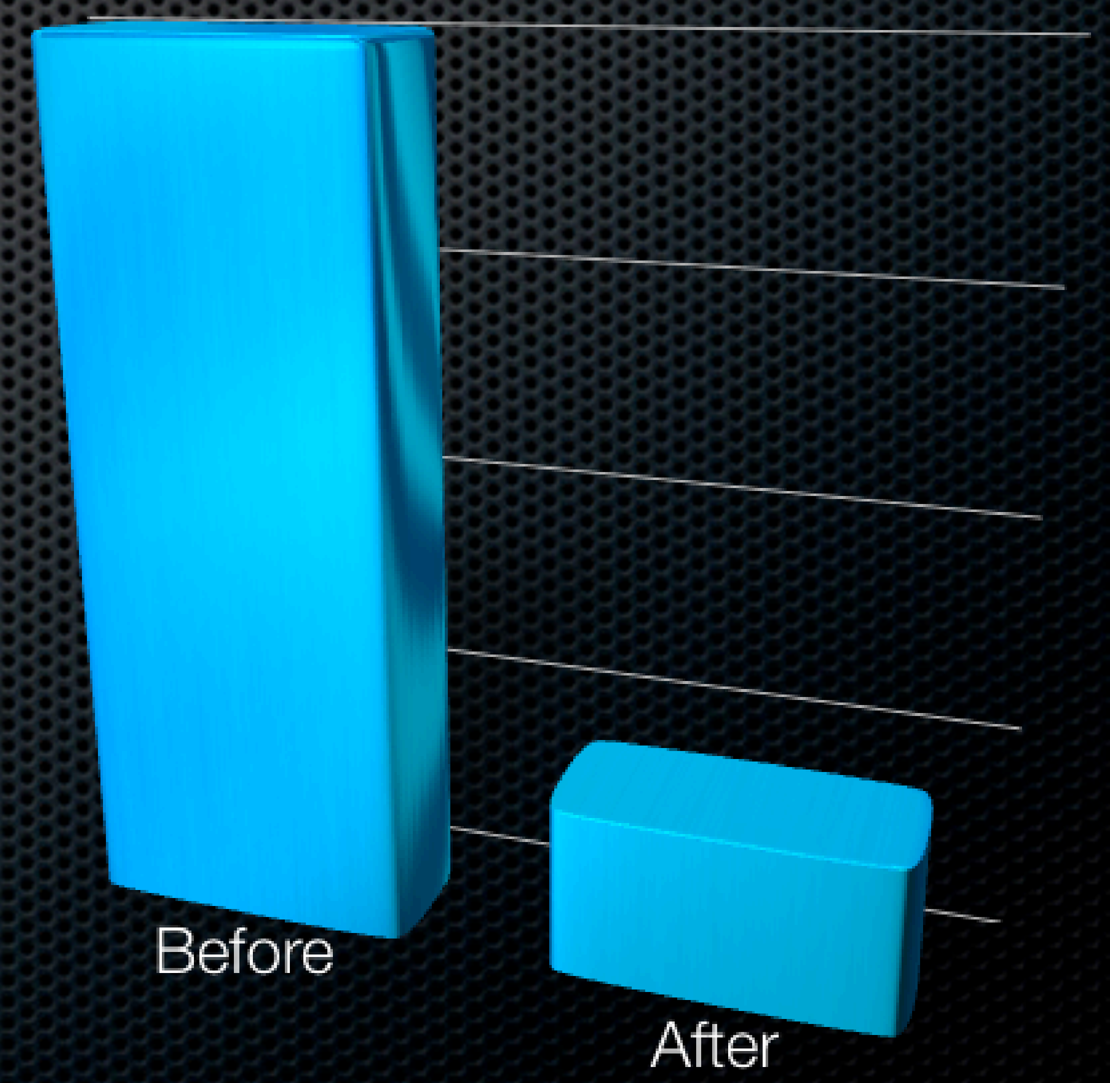
- The initial state
- Successors of any given state

Size of Reachability Graphs

Circles	Nodes	Time	Ratio(Nodes)	Ratio(Time)
0	3	0.00	-	-
1	13	0.00	4.33	-
2	51	0.00	3.92	-
3	189	0.01	3.71	5.00
4	675	0.02	3.57	3.60
5	2,349	0.07	3.48	3.89
6	8,019	0.27	3.41	3.86
7	26,973	1.07	3.36	3.94
8	89,667	3.69	3.32	3.46
9	295,245	13.15	3.29	3.56
10	964,467	129.69	3.27	9.86
11	3,129,598	2338.22	3.24	18.03

Reduction Techniques

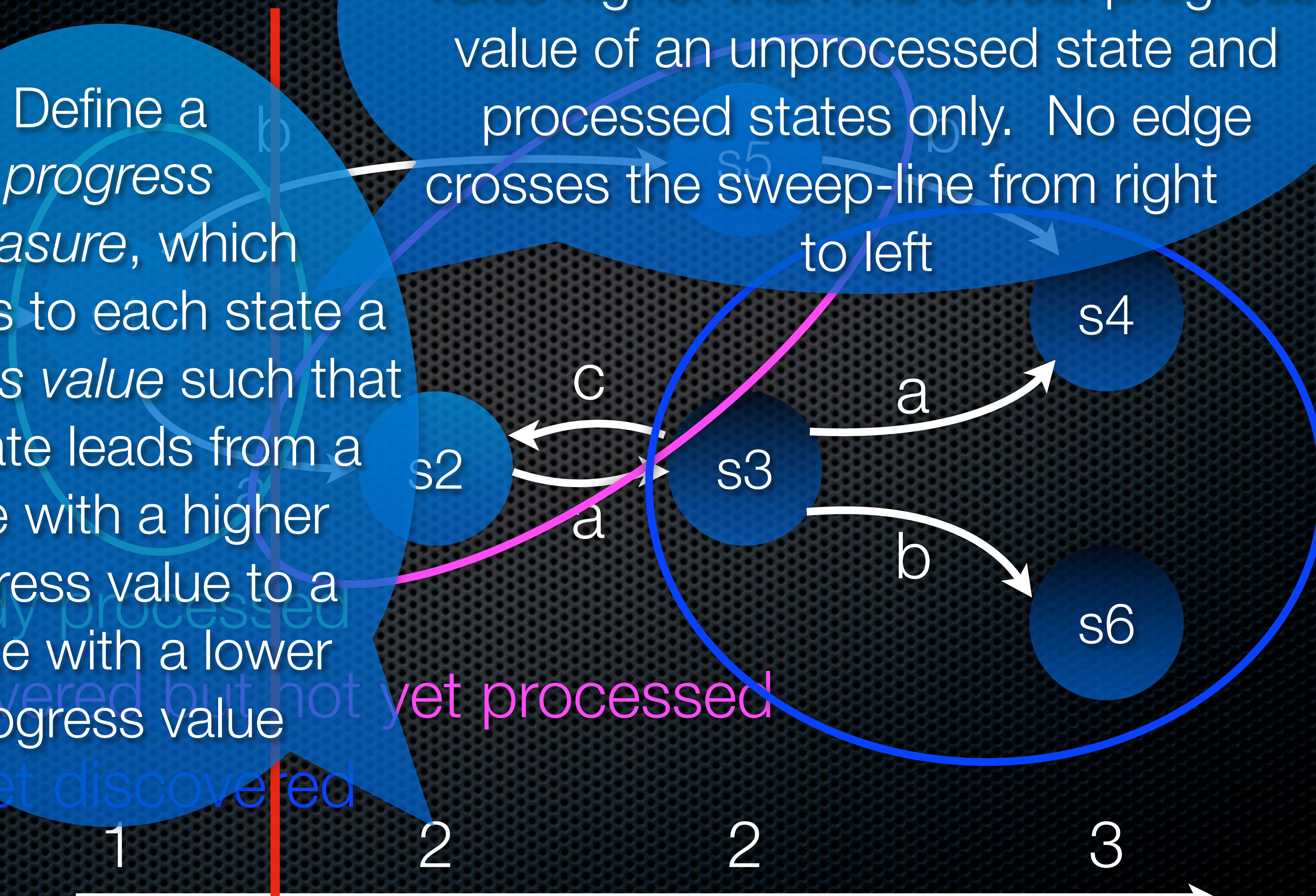
- (Symbolic reachability graph analysis)
- Explicit reachability graph analysis
  - (Use external memory)
  - (Explore only some states)
  - Store states more efficiently
  - Delete states during exploration



The Sweep-Line Method

This defines a *sweep-line* between the states with progress value higher than the lowest progress value of an unprocessed state and processed states only. No edge crosses the sweep-line from right to left

Define a *progress measure*, which assigns to each state a *progress value* such that no state leads from a state with a higher progress value to a state with a lower progress value



Already processed

Discovered but not yet processed

Not yet discovered

Invariant vs. Liveness

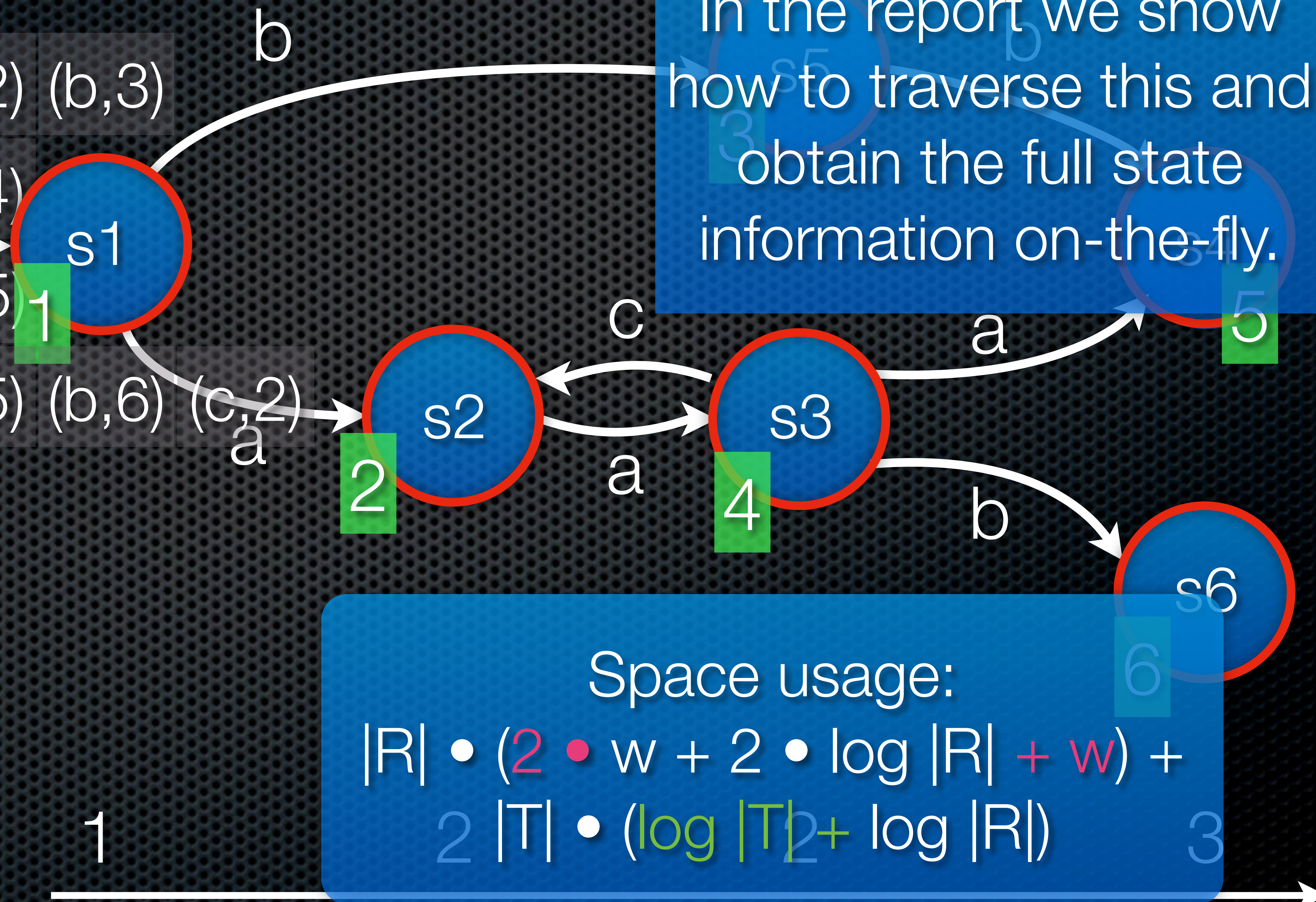
- This algorithm is fine for checking simple invariant properties
- The algorithm (in a more general version) is not very suitable for checking more advanced properties
  - Cycles may not be preserved
  - The algorithm imposes a certain traversal order
- T. Mailund and M. Westergaard. Obtaining Memory-Efficient Reachability Graph Representations Using the Sweep-Line Method. In *Proc. of TACAS'04*, volume 2988 of *LNCS*, pages 177–191. Springer-Verlag, 2004.
  - Store a compact version of the reachability graph with enough information that we can later reconstruct it

Extended Sweep-Line Method

Number of successors

Bits per successor

1	2	2	(a,2)	(b,3)	
2	1	3	(a,4)		
3	1	3	(b,5)		
4	3	3	(a,5)	(b,6)	(c,2)
5	0				
6	0				



In the report we show how to traverse this and obtain the full state information on-the-fly.

Space usage:

$|R| \cdot (2 \cdot w + 2 \cdot \log |R| + w) + |T| \cdot (\log |T| + \log |R|)$

Results – Drawing Example

	Full			Sweep-line based algorithm					
	States	Mem	Time	States	Peak	Memory		Time	
1	14	0.0	0	19	7	0.0	61%	0	-
2	52	0.0	0	59	20	0.0	45%	0	100%
3	190	0.0	0	199	59	0.0	36%	0	111%
4	676	0.1	0	687	172	0.0	30%	0	118%
5	2,350	0.2	1	2,363	486	0.1	24%	1	100%
6	8,020	0.9	9	8,035	1,469	0.2	22%	8	90%
7	26,974	3.5	112	26,991	4,425	0.7	19%	88	79%
8	89,668	12.7	1,522	89,687	12,513	2.1	17%	942	62%
9	295,246	45.6	22,430	295,267	38,083	7.0	15%	11,569	52%
10	964,468	161.9	268,275	964,491	115,920	23.1	14%	131,531	49%

Results – Drawing Example

	Sweep-line method			Sweep-line based algorithm					
	States	Mem	Time	States	Peak	Memory		Time	
1	19	0.0	0	19	7	0.0	121%	0	-
2	59	0.0	0	59	20	0.0	117%	0	100%
3	199	0.0	0	199	59	0.0	116%	0	111%
4	687	0.0	0	687	172	0.0	116%	0	141%
5	2,363	0.1	1	2,363	486	0.1	117%	1	145%
6	8,035	0.2	6	8,035	1,469	0.2	118%	8	147%
7	26,991	0.6	61	26,991	4,425	0.7	118%	88	145%
8	89,687	1.8	661	89,687	12,513	2.1	119%	942	142%
9	295,267	5.9	7,185	295,267	38,083	7.0	119%	11,569	161%
10	964,491	19.5	81,772	964,491	115,920	23.1	118%	131,531	161%

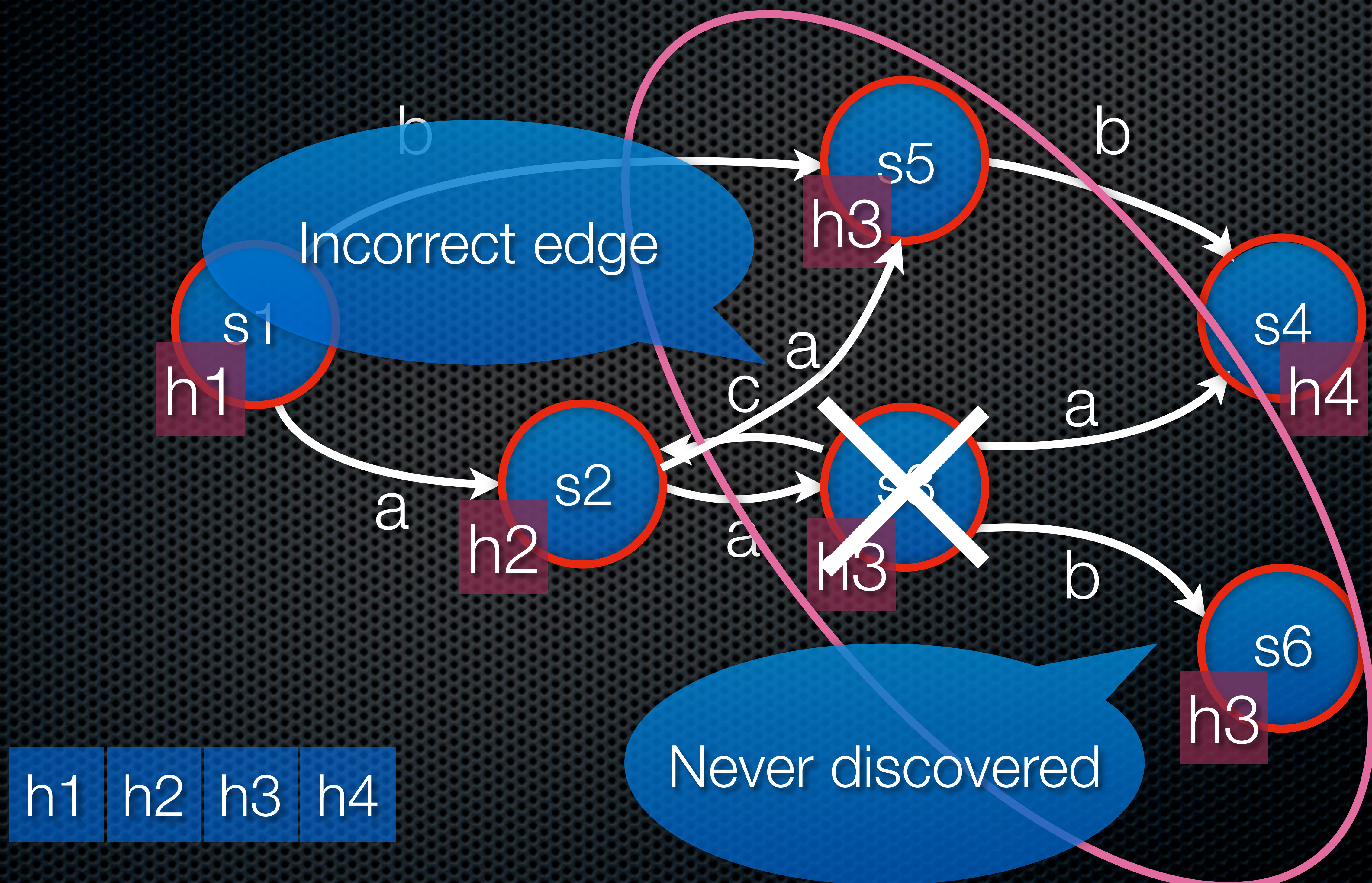
Points about the Algorithm

- Performs well when the sweep-line method does
- The method, like the standard sweep-line method, can be extended to handle *regress edges*
- The constructed representation of the reachability graph uses little more memory than an optimal representation
- The (extended) method is only implemented in DESIGN/CPN, which is no longer maintained

Hash-Compaction

- Instead of storing the full representation of a state, use a hash function to generate a compressed state descriptor
- Hash functions need not be injective, so if two states have the same compressed state descriptor, we may not realise they are different

Hash-Compaction



The ComBack Method

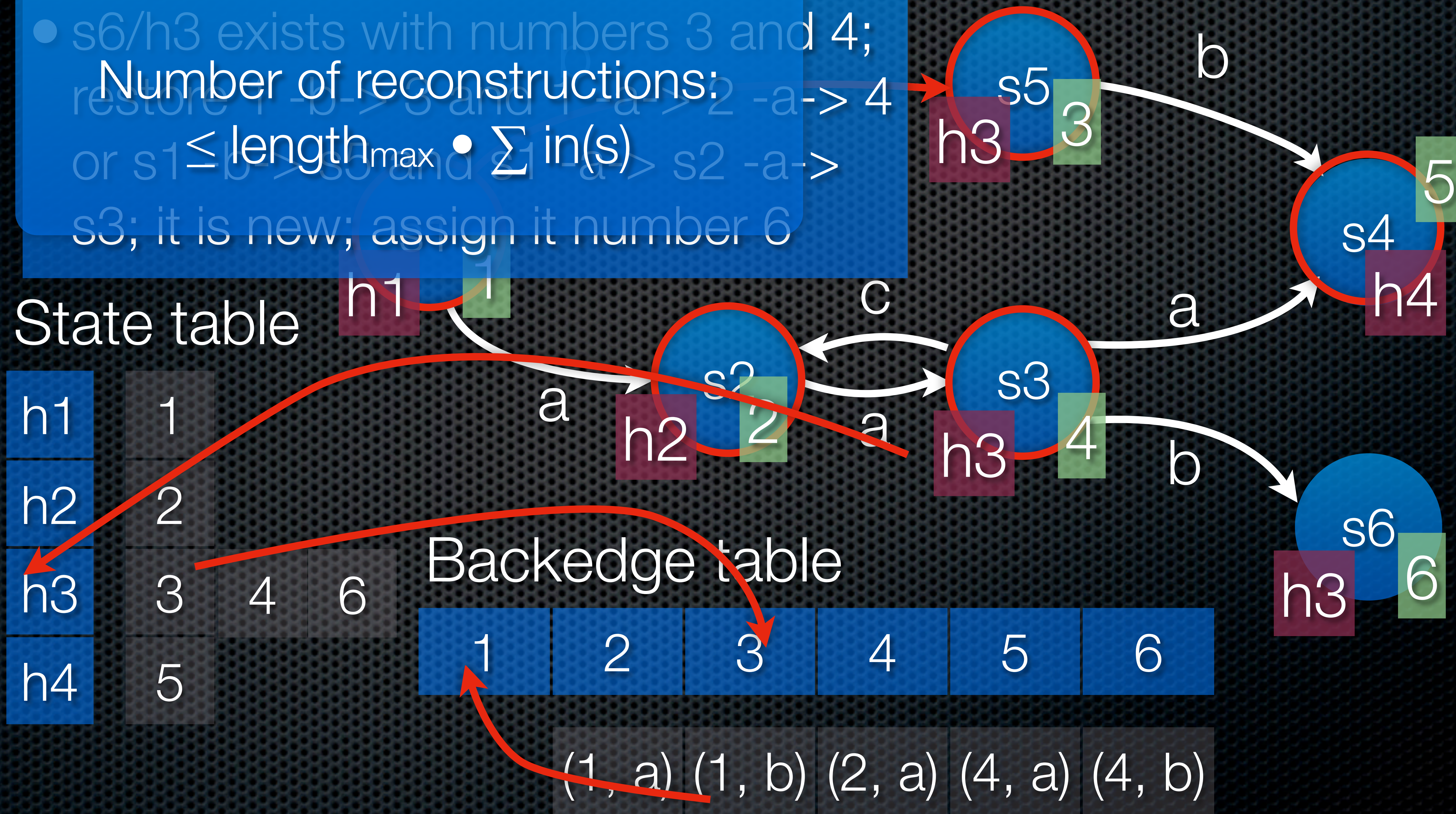
- M. Westergaard, L.M. Kristensen, G.S. Brodal, and L. Arge. The ComBack Method—Extending Hash Compaction with Backtracking. In *Proc. of ATPN'07*, volume 4546 of *MNCS*, pages 446–464. Springer-Verlag, 2007.
  - Store a spanning tree rooted in the initial state, which allows us to reconstruct full state descriptors from compressed state descriptors

Space usage:

$|R| \bullet (w_H + 3 \bullet \log |R| + \log |T|)$

Number of reconstructions:

$\leq \text{length}_{max} \bullet \sum in(s)$



Results – Drawing Example

Circles = 9	States	Memory	%	Time	%
Hash-compaction	295,237	7.5	20%	12.06	92%
Full	295,245	37.6	100%	13.10	100%
ComBack	295,245	26.1	69%	29.46	225%
ComBack Cache	295,245	26.2	70%	22.02	168%

Circles = 11	States	Memory	%	Time	%
Hash-compaction	3,124,294	75.6	18%	168.66	7%
Full	3,129,598	427.0	100%	2338.22	100%
ComBack	3,129,598	280.6	66%	1547.39	66%
ComBack Cache	3,129,598	280.6	66%	1447.90	62%

Results – Real Life Example

ERDP6,2	States	Memory	%	Time	%
Hash-compaction	206,921	5.1	6%	106	93%
Full	207,003	87.4	100%	114	100%
ComBack	207,003	29.1	33%	865	759%
ComBack Cache	207,003	29.0	33%	227	199%

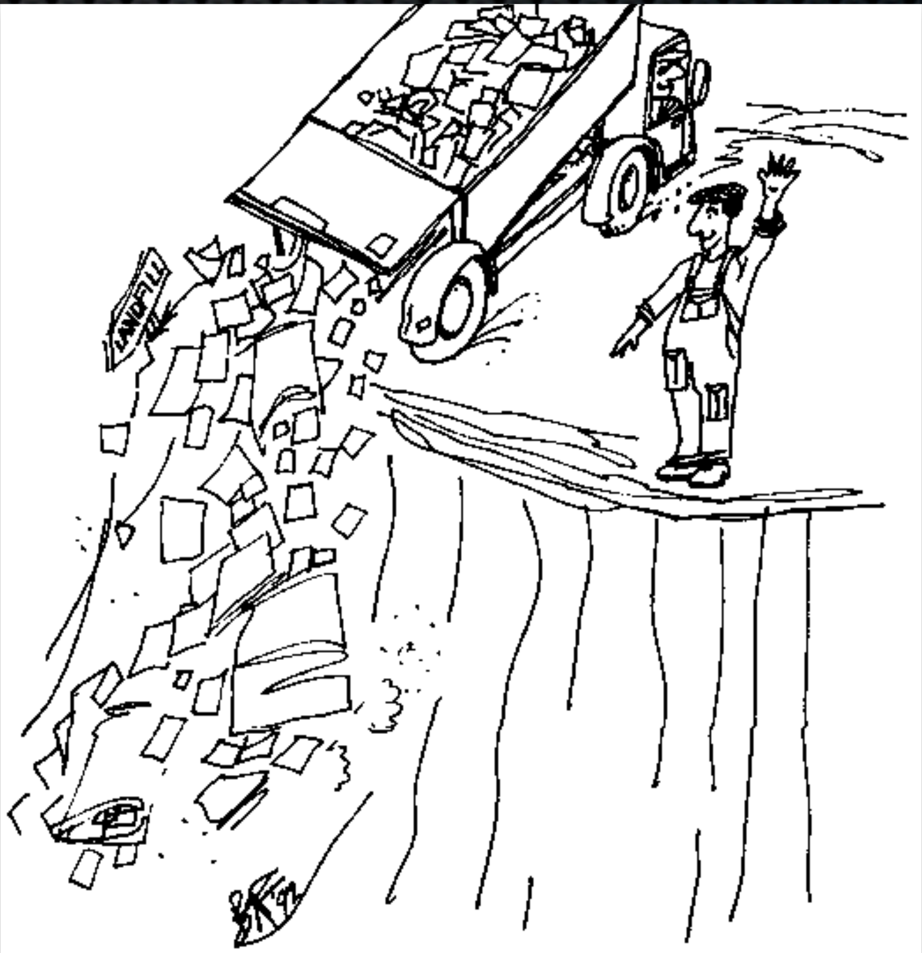
ERDP6,3	States	Memory	%	Time	%
Hash-compaction	4,270,926	113.5	-	3,341	-
Full	-	-	-	-	-
ComBack	4,277,126	572.3	-	42,711	-
ComBack Cache	4,277,126	571.2	-	18,043	-

Points about the Algorithm

- Performs relatively poorly when a lot of states need reconstruction
  - This is not only caused by hash-collisions, but also because we need a reconstruction each time we re-encounter a state
- A good caching strategy minimises the number of reconstructions and significantly improves performance!
- The algorithm is traversal agnostic (and thus easy to combine with other algorithms)
- Depth-first traversal often yields long backtraces (= takes longer) but saves more space that breadth first traversal

Contributions

- The extended sweep-line method
  - Facilitates verification of liveness properties in main memory using the sweep-line method
- The ComBack method
  - Makes hash-compaction complete



A contribution to the field

Impact – Verification

- The presented methods for verification have been used little
- Lack of user-friendly implementation
  - Difficult to make real-life case studies
  - Thus difficult to identify problems
  - Difficult for others to experiment with and improve the algorithm

Impact – Visualisation

- The BRITNeY Suite has been used extensively for
  - Visualisation (like our own real-life case)
  - Meta-visualisation (building newer, better visualisations, e.g., a work-flow system)
  - Other things (calling Java algorithms from CPN models, integrating CPN models into multi-formalism tool)
- The tool is fairly mature and according to e-mail correspondence used for several ongoing projects spanning all of the above categories

Future Work – Visualisation

- The BRITNeY Suite is fairly mature and primarily needs better documentation, bug-fixes, cleanup, etc.

Future Work – Verification

- Combine extended Sweep-Line and ComBack methods (ongoing work in ASCoVeCo project)
- Make new methods more accessible (providing means of using them for non-experts, done in ASCoVeCo project)
- Combine visualisation framework efficiently with analysis to be able to provide counter examples