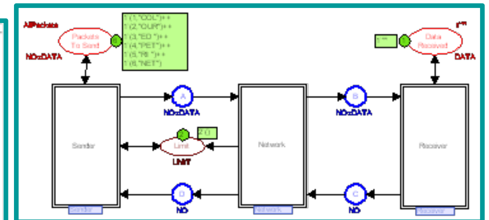
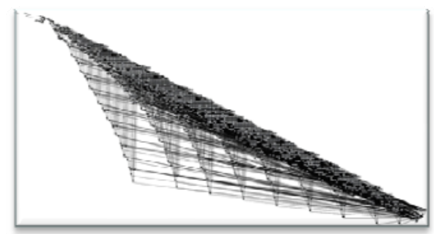
State Space Exploration of Coloured Petri Nets

Roots ← $\{s_I\}$
Nodes.Add(s_I)
while ¬ (Roots.Empty()) **do**
  Unprocessed ← Roots
  Roots ← $\emptyset$
  while ¬ (Unprocessed.Empty()) **do**
    s ← Unprocessed.GetMinElement()
    for all (t, s') such that $s \xrightarrow{t} s'$ **do**
      if ¬(Nodes.Contains(s')) **then**
        Nodes.Add(s')
        if $\psi(s) \sqsupset \psi(s')$ **then**
          Nodes.MarkPersistent(s')
          Roots.Add(s')
        else
          Unprocessed.Add(s')
        end if
      end if
    end for
    Nodes.GarbageCollect(min$\{\psi(s) \mid s \in$ Unprocessed$\}$)
  end while
end while



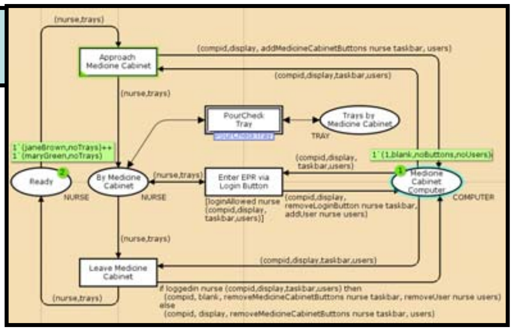


Lars M. Kristensen
Department of Computer Engineering
Bergen University College, NORWAY
Email: lmkr@hib.no /Web: www.hib.no/ansatte/lmkr

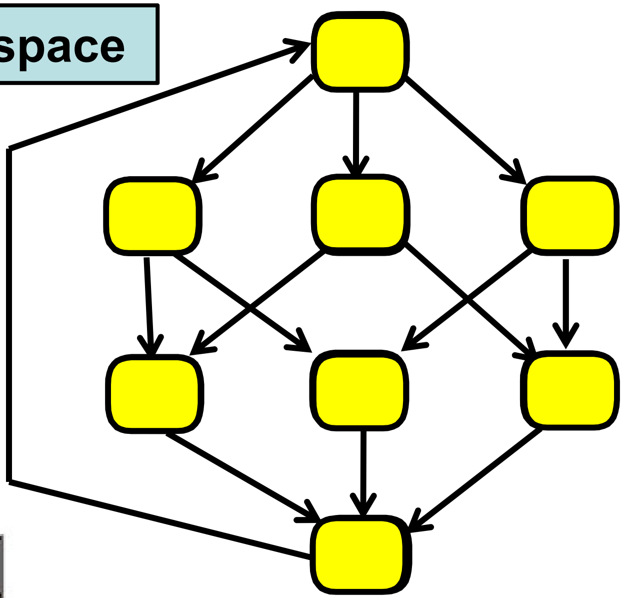
Explicit State Space Exploration

- **Explicit state space exploration is the main approach to verification of CPN models:**

Model



State space



$\textsc{Unprocessed} \leftarrow \{s_I\}$
$\textsc{Nodes.Add}(s_I)$
while $\neg\ \textsc{Unprocessed.Empty}()$ **do**
  $s \leftarrow \textsc{Unprocessed.Select}()$
  for all (e, s') such that $s \xrightarrow{e} s'$ **do**
    if $\neg(\textsc{Nodes.Contains}(s'))$ **then**
      $\textsc{Nodes.Add}(s')$
      $\textsc{Unprocessed.Add}(s')$
    end if
  end for
end while

Visited states (nodes) | **Unprocessed states**

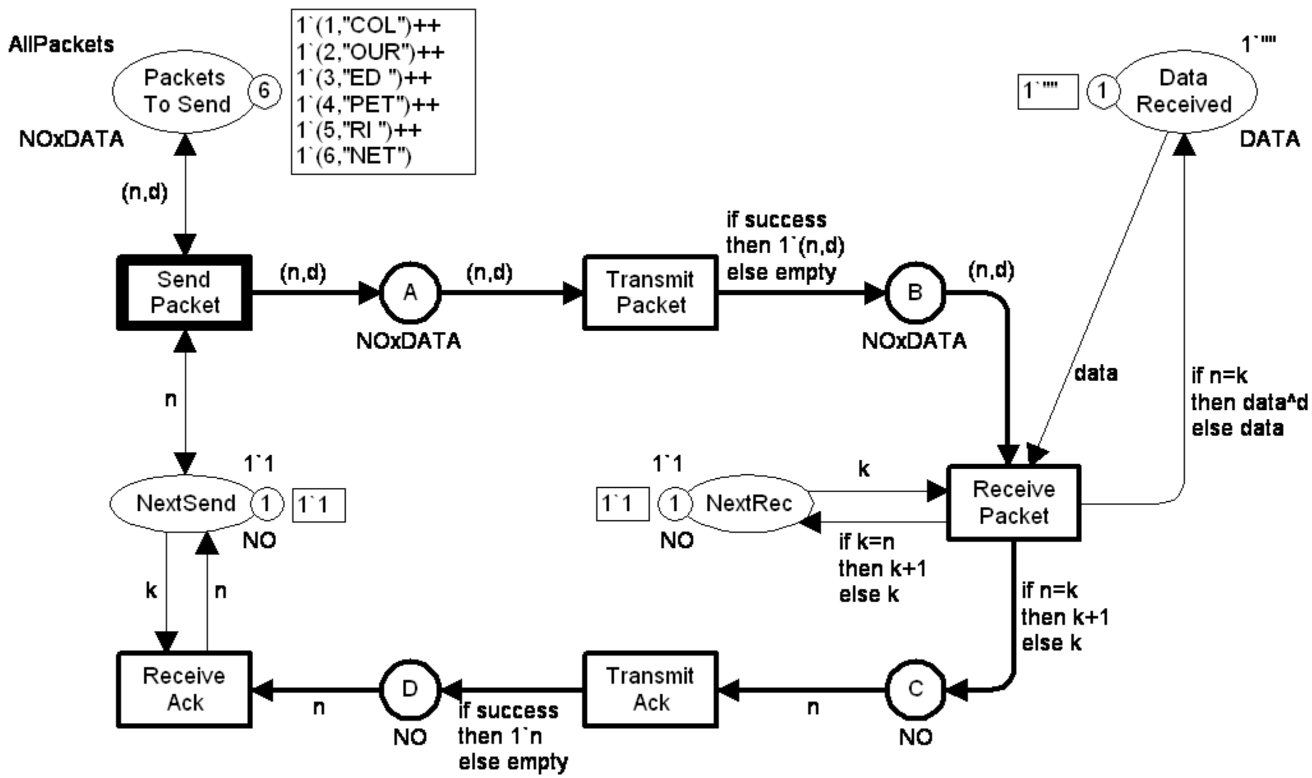
Initial state and transition relation of the model

CPNs and State Space Methods

- **A main guideline has been to support state space exploration of the full CPN modelling language:**
  - The rich data types yields state vectors (markings) of typically 100-1000 bytes.
  - The expressive inscription language make it infeasible (in general) to exploit structural properties and rely unfolding to low-level Petri Nets.
  - Calculation of enabled events (binding elements) is expensive.
- **Potentials of the CPN modelling language:**
  - The possibility of compact modelling yields smaller state spaces (model-level reduction is very important).
  - The hierarchical structure facilitates sharing of sub-states.
  - Petri net locality can be exploited to reduce time spent on calculation of enabled events.

The Simple Protocol

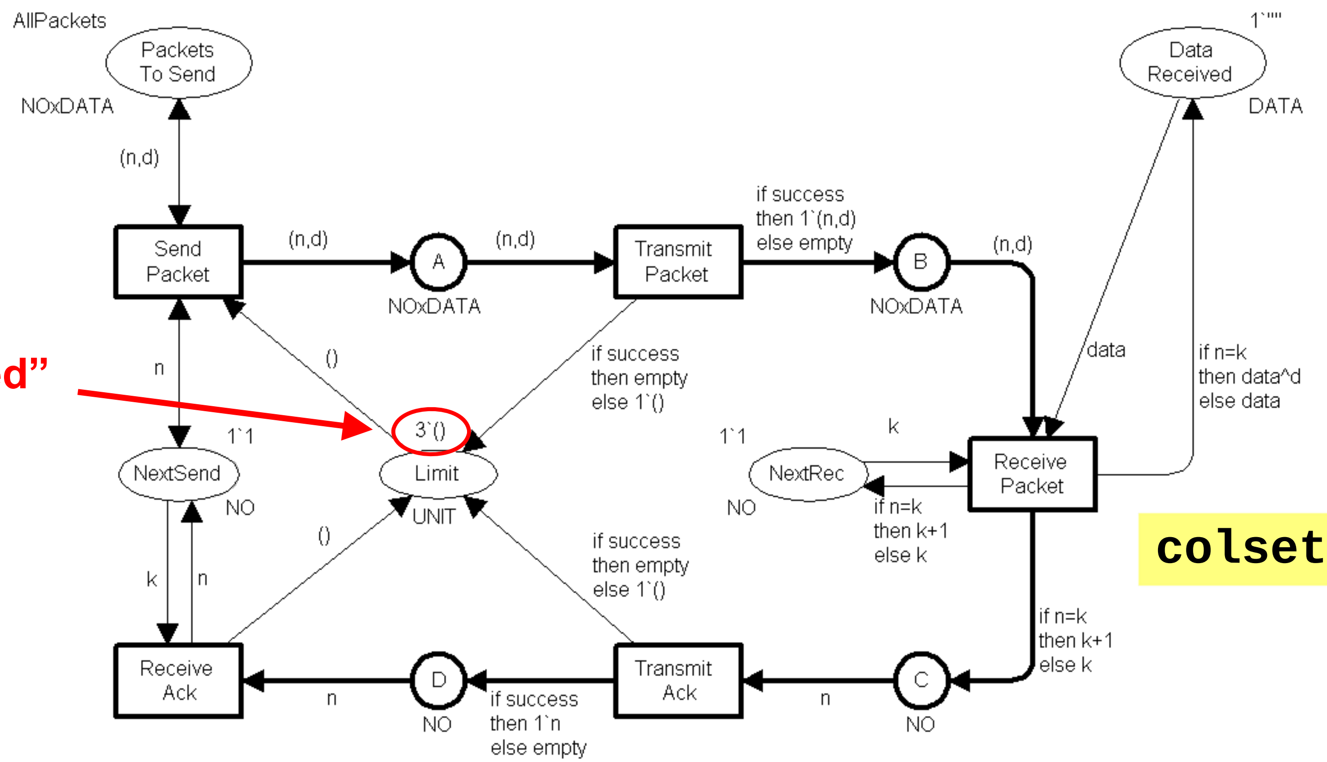
- **Transition SendPacket can produce an unlimited number of tokens on place A.**
- **This means that the state space becomes infinite:**



Revised Protocol

- **A new place Limit that limits the total number of tokens on the buffer places A, B, C, and D.**
- **This makes the state space finite.**

Three “uncoloured” tokens




```
colset UNIT = unit;
```

CPN Tools: State Space Exploration and Verification

- **Typical steps in basic application of state space methods for verification of CPNs:**
  - **Generate the full state space of the CPN model.**
  - **Generate a state space report containing answers to a set of standard CPN behavioural properties.**
  - **Investigate system specific properties using the provided standard and user-defined query functions.**
- **Supports visualisation of state space fragments interactively or automatically:**
  - **Useful for debugging purposes.**
  - **Useful for visualisation of counter examples.**
- **Implementations of several advanced state space methods for alleviating state explosion.**

State Space Report

- The state space report contains information about standard behavioural properties for CPNs.
- Statistical information:
  - Size and time used for state space generation.
- Boundedness properties:
  - Bounds for the number of tokens on each place (integer bounds)
  - Information about the possible token colours (multi-set bounds).
- Home and liveness properties:
  - List of home markings and list dead markings.
  - Dead and live transitions.
- Fairness properties for transitions.
- If the system contains errors this is very often reflected in the state space report.

Statistical Information

State Space Statistics

	State Space		Scc Graph	
	Nodes:	13,215	Nodes:	5,013
	Arcs:	52,784	Arcs:	37,312
	Secs:	53	Secs:	2
	Status:	Full		

- **State space contains more than 13.000 nodes and more than 52.000 arcs.**
- **State space was constructed in less than one minute and it is full (contains all reachable markings).**
- **The Strongly Connected Component Graph (SCC graph) is smaller (hence we have cycles).**
- **The SCC graph was constructed in 2 seconds.**

Integer bounds

- The best upper integer bound for a place is the maximal number of tokens on the place in a reachable marking.
- The best lower integer bound for a place is the minimal number of tokens on the place in a reachable marking.

Best Integers Bounds	Upper	Lower
PacketsToSend	6	6
DataReceived	1	1
NextSend, NextRec	1	1
A, B, C, D	3	0
Limit	3	0

- **PacketsToSend has exactly 6 tokens in all reachable markings.**
- **DataReceived, NextSend and NextRec have exactly one token each in all reachable markings.**
- **The remaining five places have between 0 and 3 tokens each in all reachable markings.**

Upper Multi-set Bounds

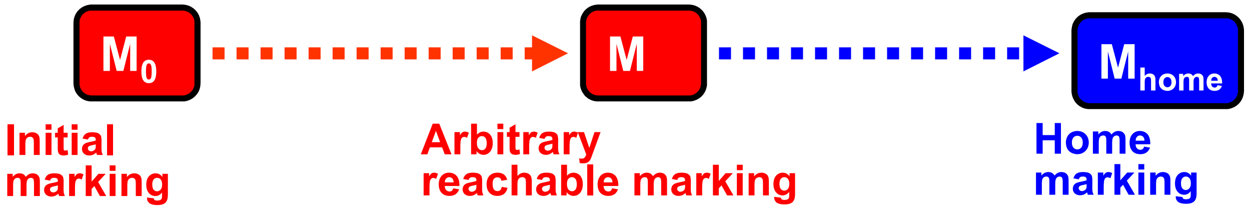
- The best upper multi-set bound specifies for each colour c the maximal number of tokens with colour c in a reachable marking.

Best Upper Multiset Bounds

PacketsToSend	1‘(1,"COL")++1‘(2,"OUR")++1‘(3,"ED ")++ 1‘(4,"PET")++1‘(5,"RI ")++1‘(6,"NET")
DataReceived	1‘""++1‘"COL"++1‘"COLOUR"++1‘"COLOURED "++ 1‘"COLOURED PET"++1‘"COLOURED PETRI "++ 1‘"COLOURED PETRI NET"
NextSend, NextRec	1‘1++1‘2++1‘3++1‘4++1‘5++1‘6++1‘7
A, B	3‘(1,"COL")++3‘(2,"OUR")++3‘(3,"ED ")++ 3‘(4,"PET")++3‘(5,"RI ")++3‘(6,"NET")
C, D	3‘2++3‘3++3‘4++3‘5++3‘6++3‘7
Limit	3‘()

Home Markings

- A **home marking** is a marking M_{home} which can be reached from any reachable marking.

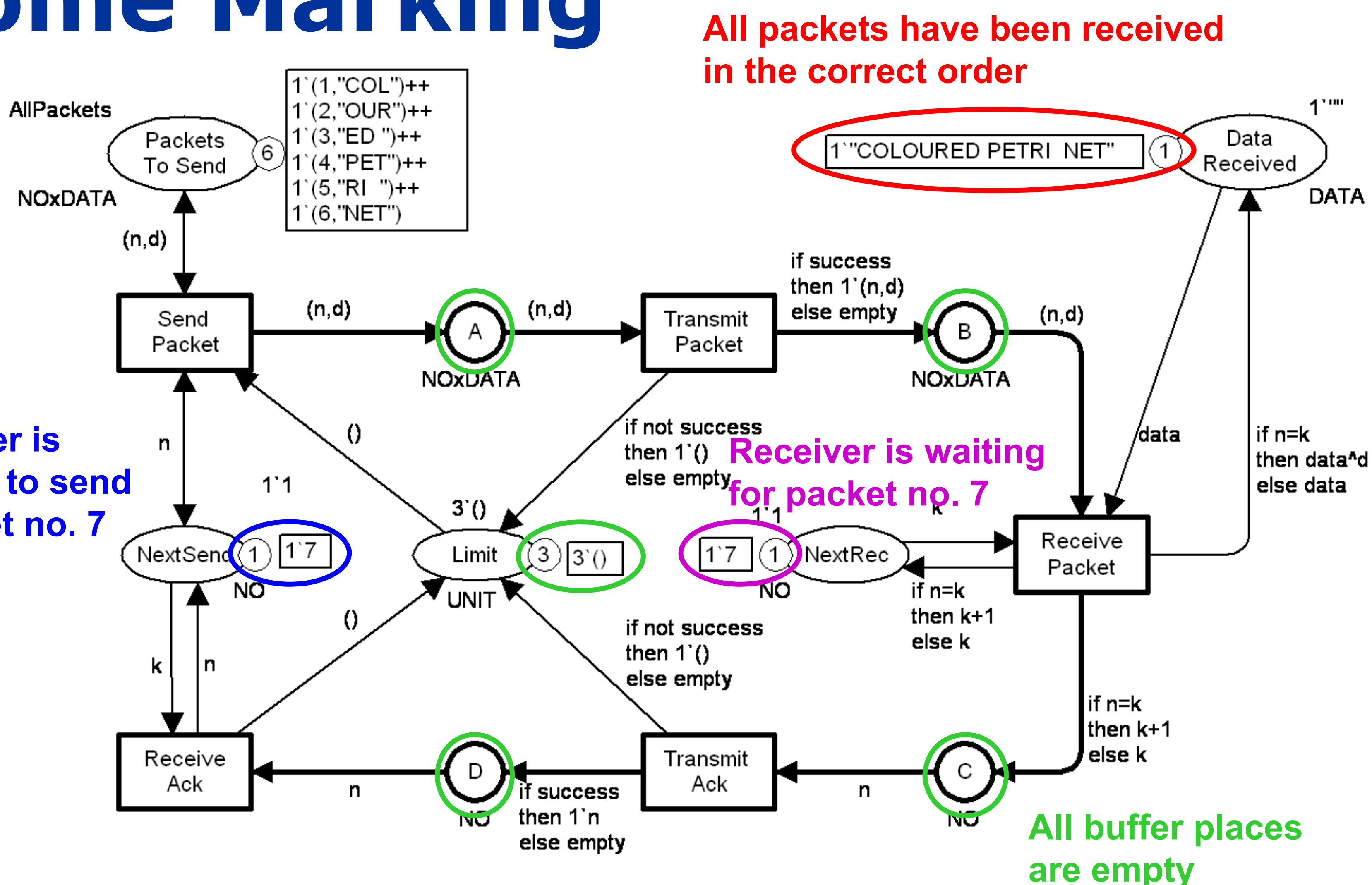


- Impossible to have an occurrence sequence which cannot be extended to reach M_{home}.
- There is a **single home marking** represented by node number 4868.

Home Properties

Home Markings: [4868]

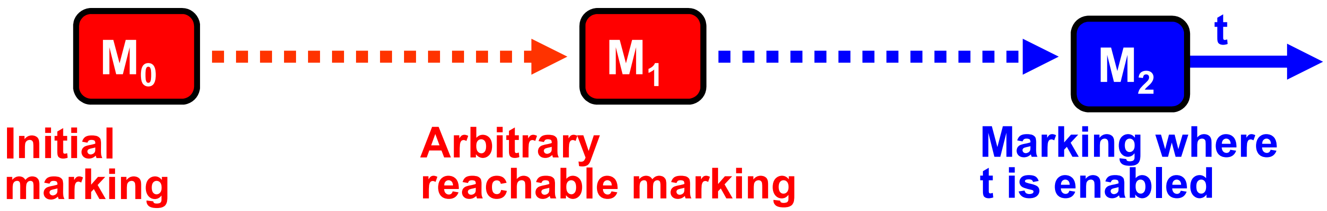
Home Marking



- **Successful completion of transmission.**

Liveness Properties

- A **marking M** is **dead** if M has **no enabled transitions.**
- A **transition t** is **dead** if it is disabled in all reachable markings.
- A **transition** is **live** if it can be made enabled from any reachable marking:



Liveness Properties

Dead Markings:	[4868]
Dead Transitions:	None
Live Transitions:	None

Marking no 4868

- We have seen that marking M_{4868} represents the state corresponding to successful completion of the transmission.
- M_{4868} is the <u>only</u> dead marking.
- If the protocol execution terminates we are in the desired terminating state.

- M_{4868} is also a home marking.
- Tells us that it always is possible to reach the desired terminating state.

Query Functions

- The state space report considers behavioural properties applicable to all CPN models.
- Model-specific properties can be investigated by means of user-defined queries.
- The queries typically consists of 5-20 lines of code written in Standard ML using:
  - A set of standard query functions.
  - A set of state space search function.
- The ASK-CTL library is also available for writing queries in a state-and-event variant of CTL.

Example: A User-defined Query Function

- Investigate whether the protocol obeys the **stop-and-wait** strategy:

```
fun StopWait n =
  let
    val NextSend = ms_to_col (Mark.Protocol'NextSend 1 n);
    val NextRec  = ms_to_col (Mark.Protocol'NextRec 1 n);
  in
    (NextSend = NextRec) orelse (NextSend = NextRec - 1)
  end;
```

Converts a multiset 1`x with one element to the colour x

```
val SWviolate = PredAllNodes (fn n => not (StopWait n));
```

Predefined search function

Negation

- The **stop-and-wait** strategy is <u>not</u> satisfied (7020 violations).

We check whether some states violate the state predicate.

Counter Example

- The **binding elements** in the **path** can be obtained by the following query:

```
List.map (ArcToBE (ArcsInPath(1,557)));
```

Maps a state space arc into its binding element

State space arcs in one of the shortest paths leading to 557.

Lowest numbered node in the list SWviolate

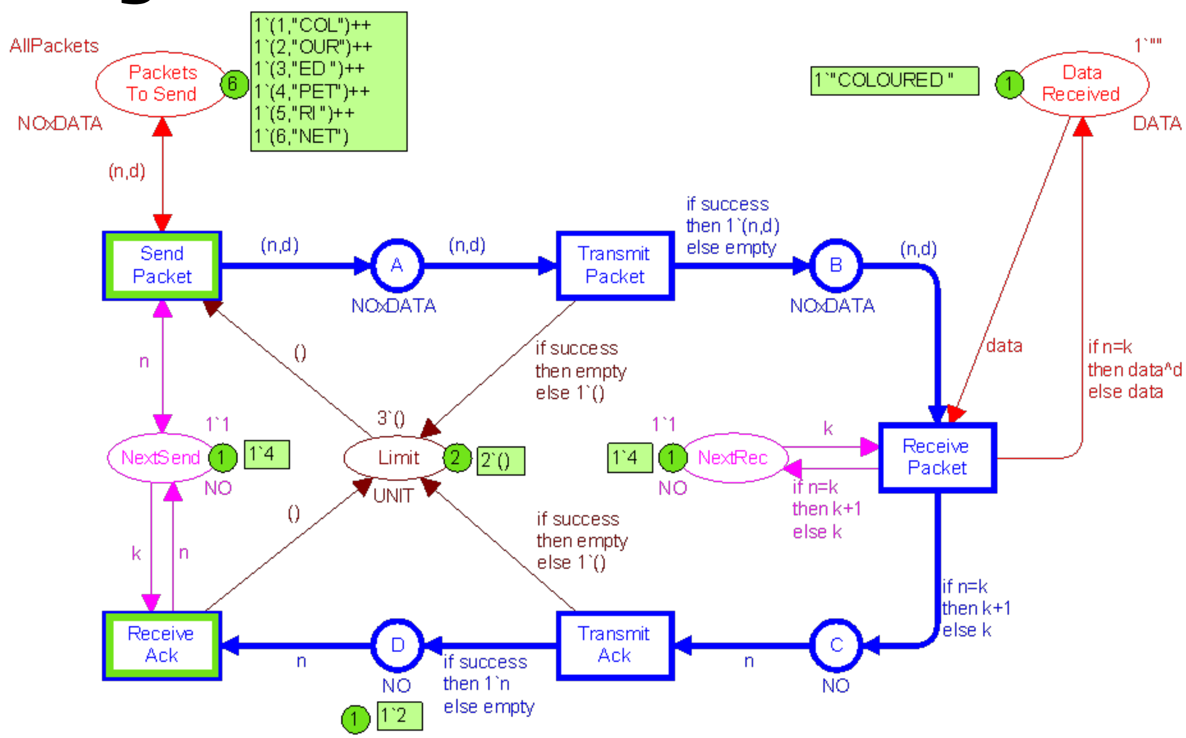
- The path can be **visualised** using the drawing facilities in the CPN Tools.

Counter Example

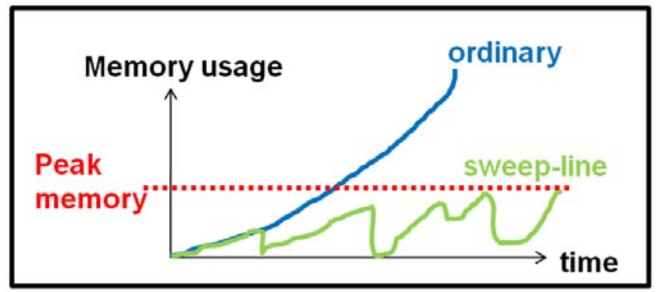
	1	(SendPacket, <d="COL",n=1>)
	2	(TransmitPacket, <n=1,d="COL",success=true>)
Packet no 1 and its ack	3	(ReceivePacket, <k=1,data="",n=1,d="COL“>)
	4	(SendPacket, <d="COL",n=1>)
	5	(TransmitAck, <n=2,success=true>)
	6	(ReceiveAck, <k=1,n=2>)
	7	(SendPacket, <d="OUR",n=2>)
	8	(TransmitPacket, <n=1,d="COL",success=true>)
Packet no 2 and its ack	9	(TransmitPacket, <n=2,d="OUR",success=true>)
	10	(ReceivePacket, <k=2,data="COL",n=1,d="COL“>)
	11	(ReceivePacket, <k=2,data="COL",n=2,d="OUR“>)
	12	(TransmitAck, <n=3,success=true>)
	13	(ReceiveAck, <k=2,n=3>)
Packet no 3	14	(SendPacket, <d="ED ",n=3>)
NextRec = 4	15	(TransmitPacket, <n=3,d="ED ",success=true>)
	16	(ReceivePacket, <k=3,data="COLOUR",n=3,d="ED “>)
Retrans- mission	17	(TransmitAck, <n=2,success=true>)
NextSend = 2	18	(ReceiveAck, <k=3,n=2>)

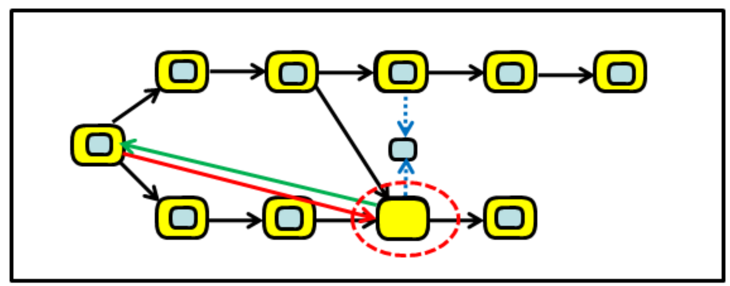
Violation of stop-and-wait strategy

- **Acknowledgements may overtake each other on C and D.**
- **It is possible for the sender to receive an old acknowledgement which decrements NextSend.**



Advanced State Space Methods for Coloured Petri Nets

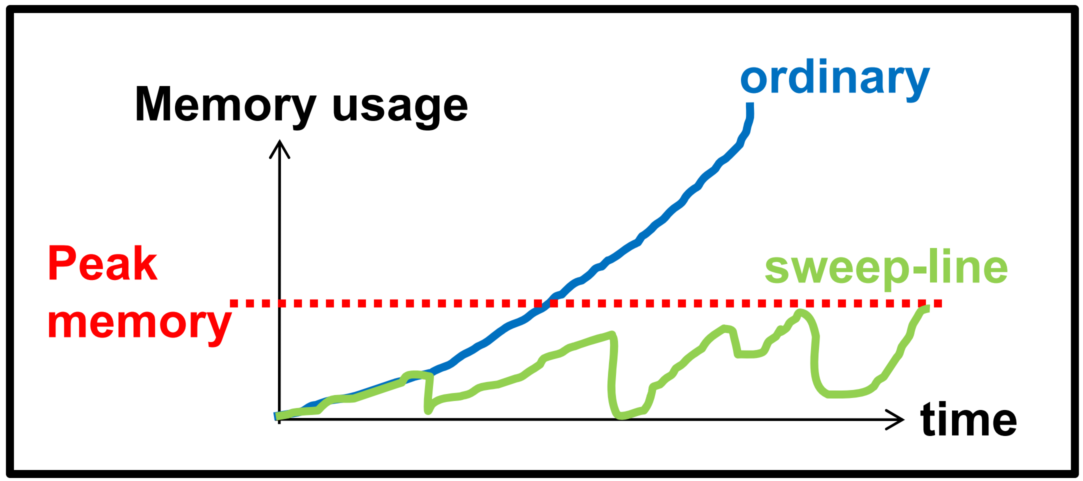




The Sweep-line Method

[Christensen, Evangelista, Kristensen, Mailund, Westergaard]

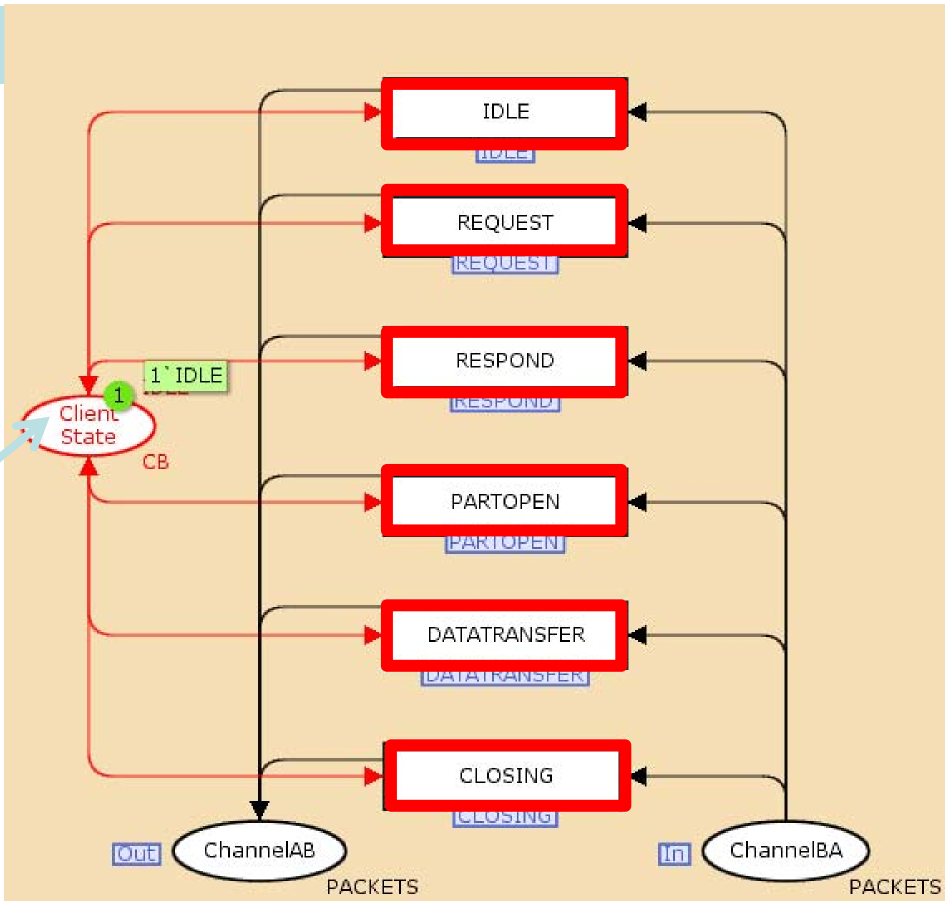
- The basic idea is to exploit a certain kind of progress exhibited by many concurrent systems:
  - Retransmission counters and sequence numbers in protocols.
  - Commit phases in transaction protocols.
  - Control flow in programs and business processes.
  - Time in timed CPN models (value of global clock).
  - Object identifiers in OO-CPNs.
  - ...
- Makes it possible to explore all the reachable states.
- Storing only small state space fragments in memory at a time.



Example: On-the-fly Verification of the Datagram Congestion Control Protocol

- **DCCP connection management proceeds in phases:**

Client

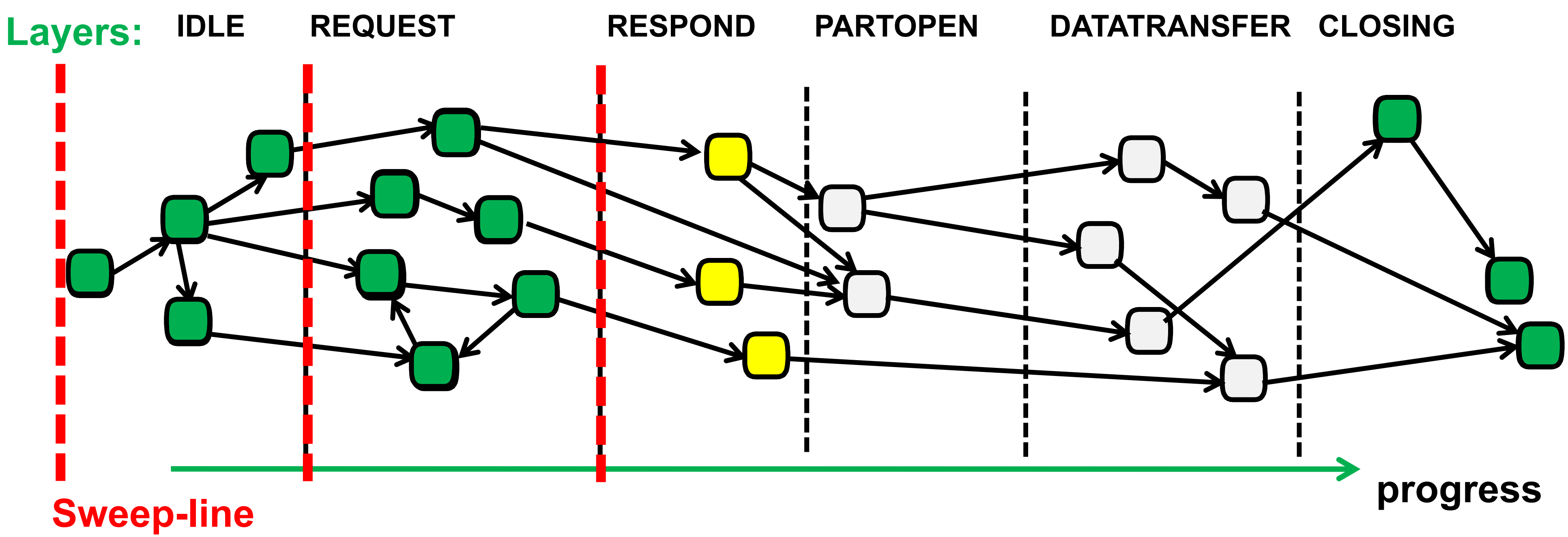


Marking of ClientState place specifies current phase

Progress

Sweep-line Exploration

- The inherent progress is reflected also in the state space of the CPN model:



- The state space is explored layer-by-layer in progress-first order.

Progress Measure

- **The progress is captured by a user-provided monotonic progress measure:**

A monotonic progress measure is a tuple $\mathcal{P} = (O, \sqsubseteq, \psi)$ *s.t:*

- O *is a set of progress values.*
- $(O, \sqsubseteq)$ *is a total order.*
- $\psi : S \rightarrow O$ *is a progress mapping satisfying:*

$$\forall s, s' \in \mathsf{reach}(s_I) : s \rightarrow^* s' \Rightarrow \psi(s) \sqsubseteq \psi(s')$$

DCCP: Client state progress mapping

$$\psi(s) = \begin{cases} 0 \text{ if } CS(s) = \text{IDLE} \\ 1 \text{ if } CS(s) = \text{REQUEST} \\ 2 \text{ if } CS(s) = \text{RESPOND} \\ 3 \text{ if } CS(s) = \text{PARTOPEN} \\ 4 \text{ if } CS(s) = \text{DATATRANSFER} \\ 5 \text{ if } CS(s) = \text{CLOSING} \end{cases}$$

- **Observation:** Monotonicity can be checked fully automatically during state space exploration.

DCCP: Experimental Results

[Vanit-Anunchai, Billington, Gallasch (2007)]

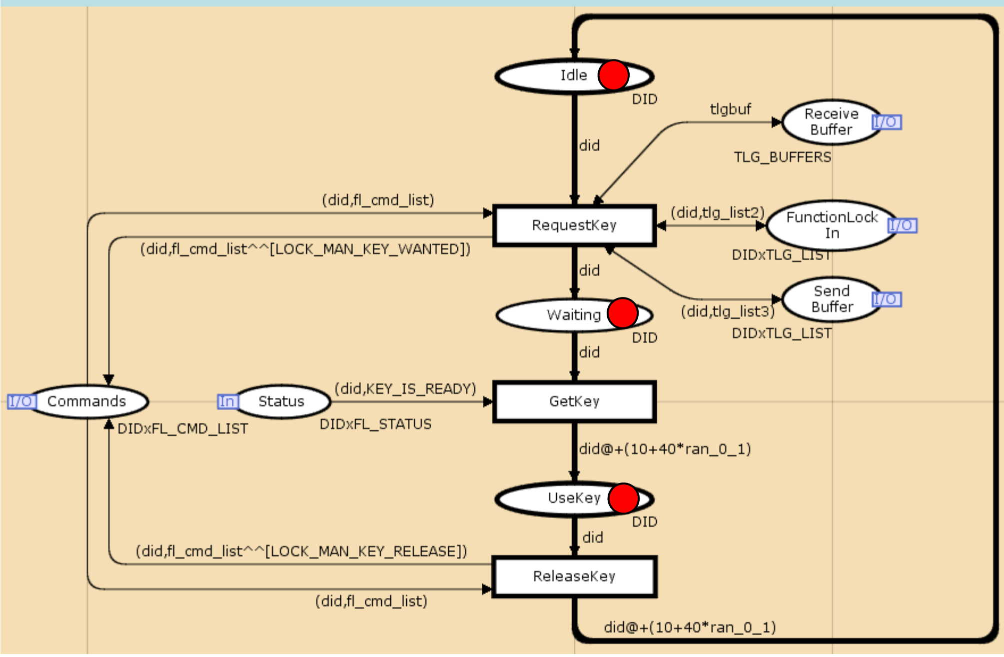
- **Refined progress measure taking into account also server state and retransmission counters:**

Sweep-line$_S$ specification model			Sweep-line$_A$ augmented model			(S/C)*100		(A/C)*100	
total nodes	peak nodes	hh:mm:ss	total nodes	peak nodes	hh:mm:ss	% space	% time	% space	% time
2,397	918	00:00:02	4,870	87	00:00:03	38.30	200	3.63	300
11,870	4,435	00:00:10	29,212	288	00:00:23	37.36	125	2.43	288
61,239	24,289	00:01:22	172,307	1,096	00:02:46	39.66	126	1.79	255
116,745	42,486	00:03:24	362,528	1,263	00:05:09	36.39	111	1.08	169
296,961	123,463	00:12:51	934,049	4,167	00:17:58	41.58	121	1.40	170
964,862	354,710	01:59:47	3,970,455	6,142	01:09:03	36.76	118	0.64	68
–	–	–	31,872,051	34,059	11:46:59	–	–	–	–
–	–	–	219,200,989	161,461	120:07:23	–	–	–	–
3,270	1,148	00:00:02	6,244	146	00:00:05	35.11	100	4.46	250
9,080	3,321	00:00:08	20,150	430	00:00:18	36.57	133	4.74	300
8,890	3,550	00:00:07	17,536	233	00:00:14	39.93	140	2.62	280
45,368	17,214	00:00:46	159,818	394	00:02:05	37.94	115	0.87	312
79,320	30,774	00:01:30	169,728	1,341	00:02:44	38.80	129	1.69	234
127,195	49,737	00:02:40	289,062	2,573	00:04:54	39.10	130	2.02	239
305,807	110,955	00:08:48	1,441,029	2,798	00:24:25	36.28	107	0.91	298
477,764	175,913	00:21:10	2,058,949	1,727	00:33:52	36.82	107	0.36	171
1,493,946	569,749	03:16:27	8,141,588	6,719	02:26:25	–	–	–	–
–	–	–	17,594,060	18,606	05:50:50	–	–	–	–

Generalised Sweep-Line Method

- **Monotonic progress measures are sufficient for systems exhibiting global progress.**
- **Many systems exhibit local progress and occasional regress (e.g., sequence number wrap, control flow loops,…):**

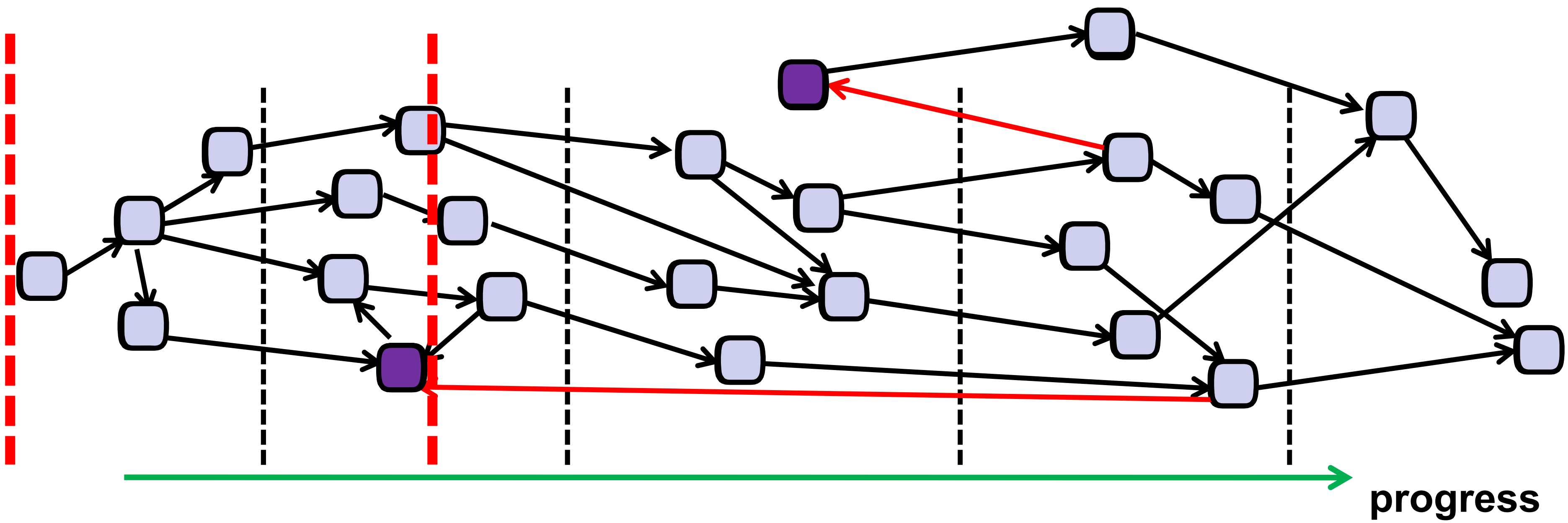
Bang and Olufsen Lock Management Protocol



- **The state space contains regress edges.**
- **Termination is no longer guaranteed.**

Generalised Sweep-line Method

- **Cannot determine whether a destination state of a regress edge has already been explored:**



- **Detect regress edges during exploration and mark destination markings as persistent.**

Algorithm and Implementation

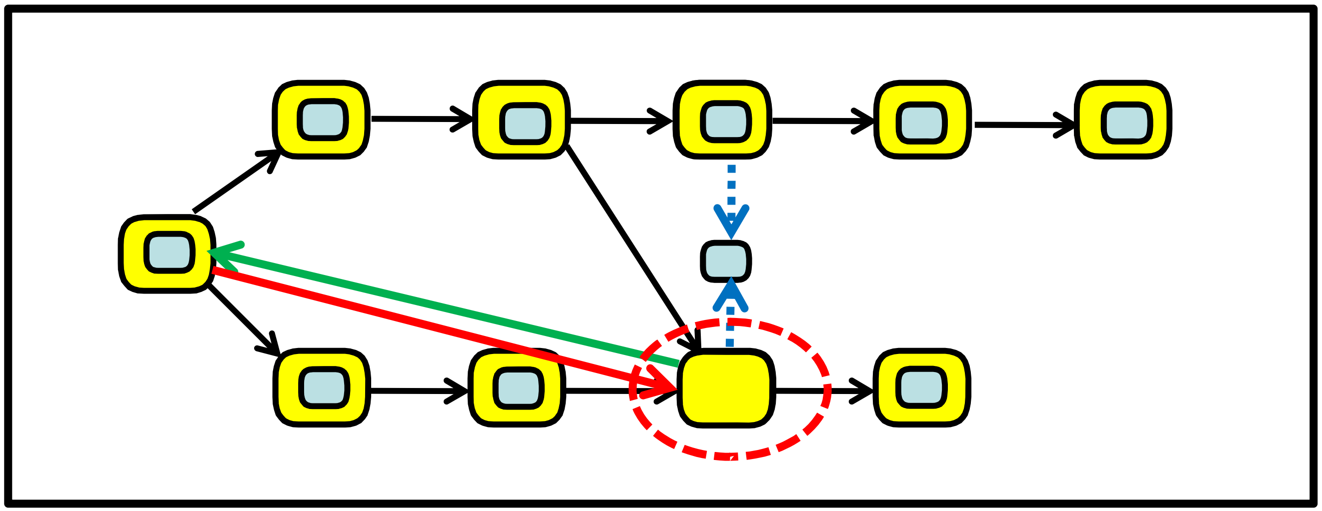
```
Roots ← {s_I}
Nodes.Add(s_I)
while ¬ (Roots.Empty()) do
   Unprocessed ← Roots
   Roots ← ∅
   while ¬ (Unprocessed.Empty()) do
      s ← Unprocessed.GetMinElement()
      for all (t, s') such that s –t→ s' do
         if ¬(Nodes.Contains(s')) then
            Nodes.Add(s')
            if ψ(s) ⊐ ψ(s') then
               Nodes.MarkPersistent(s')
               Roots.Add(s')
            else
               Unprocessed.Add(s')
            end if
         end if
      end for
      Nodes.GarbageCollect(min{ψ(s) | s ∈ Unprocessed})
   end while
end while
```

- **Unprocessed** implemented as a priority queue on progress values.
- **Deletion of states** can be implemented efficiently by detecting when the sweep-line moves.
- **Sub-state sharing** requires a reference count mechanism.

Sweep-Line Extensions

- **Counter example generation is not immediately possible due to state deletion:**
  - A inverse spanning tree can be written on external storage during state space exploration.
  - Each visited state is written to disk with an associated index pointing to its generating predecessor state.
  - Following the index pointers backwards yields the counter example (number of disk seeks proportional to path length).
- **Sweep-line exploration suited for verification of safety properties:**
  - In automata-based approaches a progress measure can be computed automatically on the property automata prior to parallel composition.
  - For CTL (LTL) model checking, state deletion can be replaced by storing only the value of atomic propositions for each marking.

The Comback Method



The Hash Compaction Method

[Wolper&Leroy'93, Stern&Dill'95]

- Relies on a hash function H for memory efficient representation of visited (explored) states:

$H : S \rightarrow \{0,1\}^w$

s ➔ 01100011000110001110000111000101

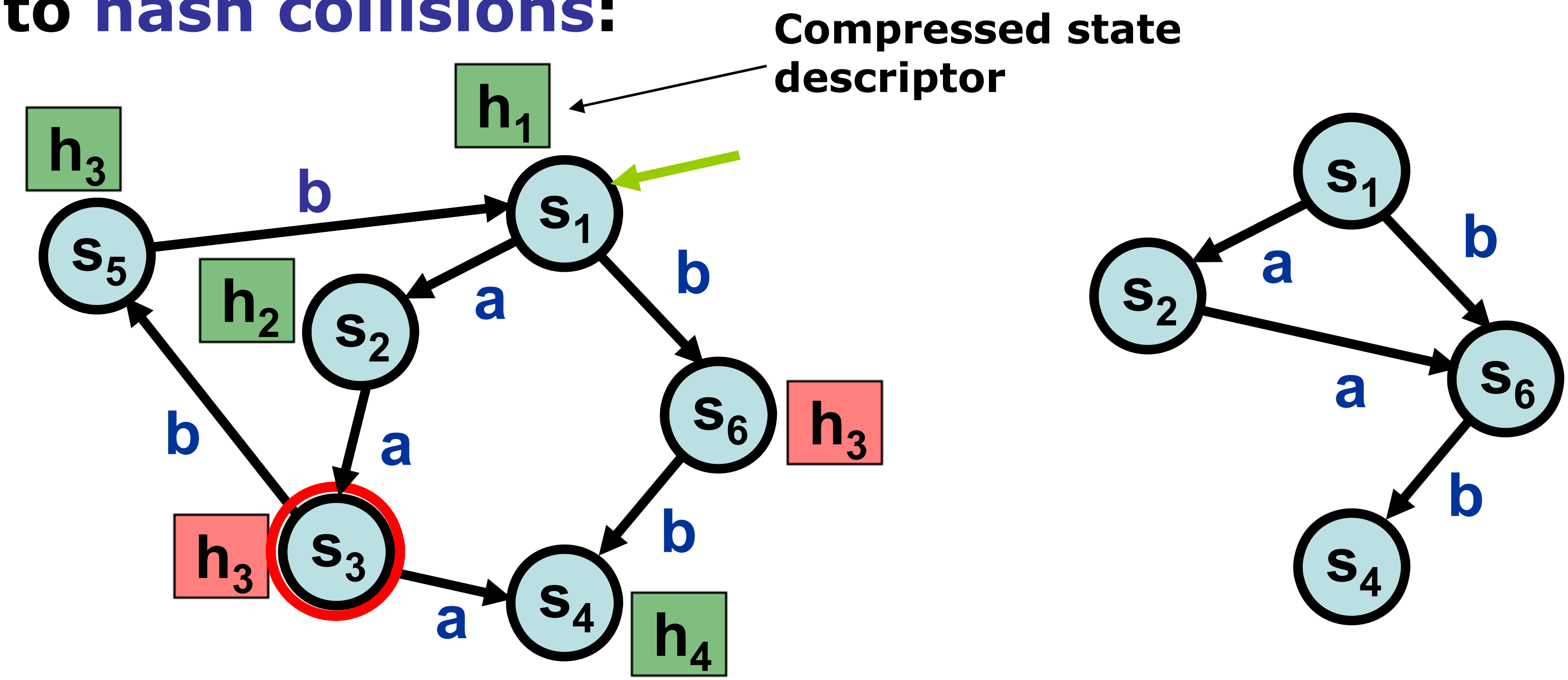
Full state descriptor (100-1000 bytes)

Compressed state descriptor (4-8 bytes)

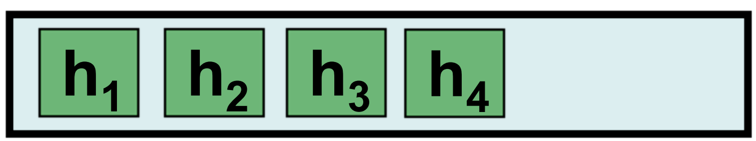
- Only the compressed state descriptor is stored in the state table of visited states.

Example: Hash Compaction

- **Cannot guarantee full state space coverage due to hash collisions:**



State table:



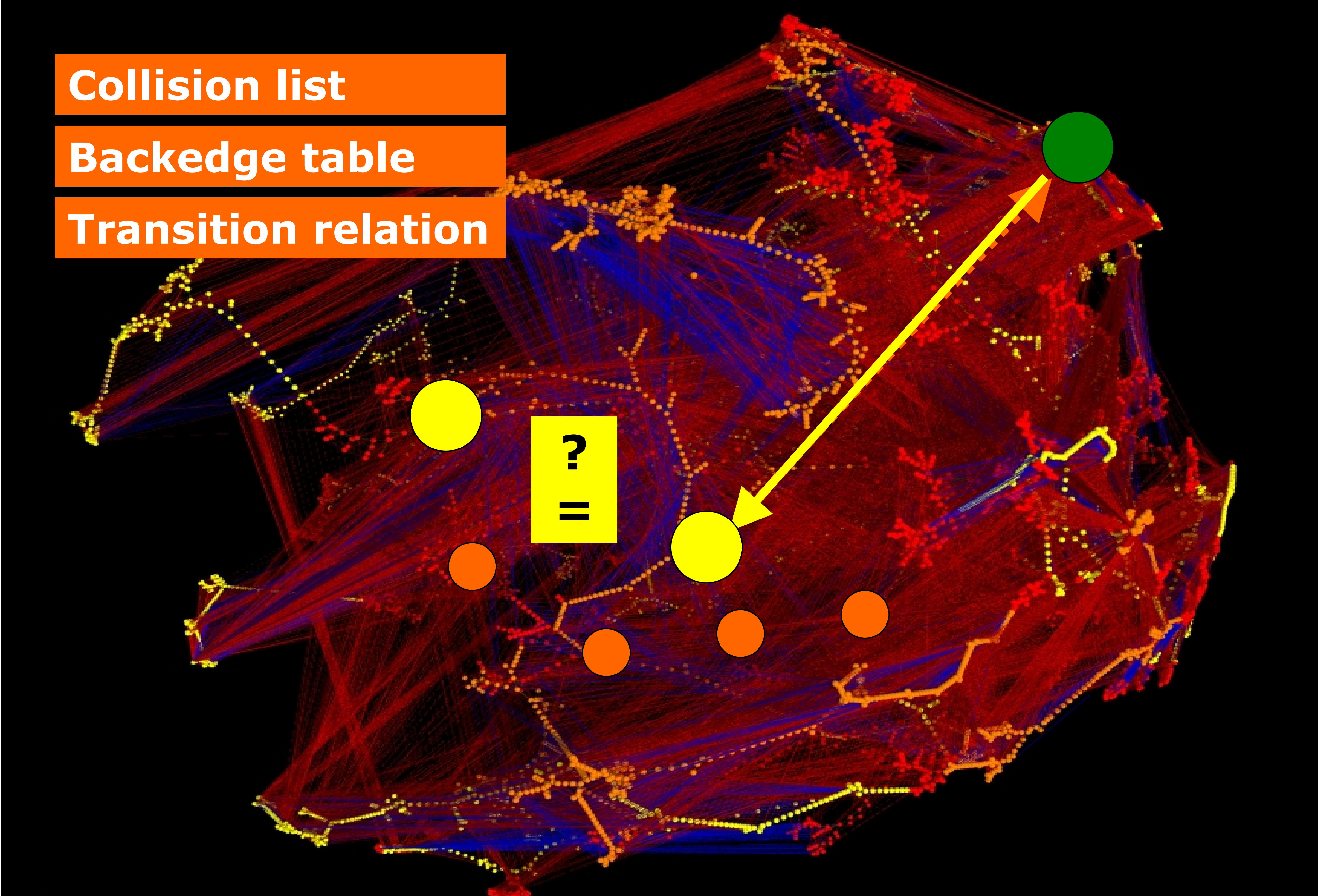
The Comback Method

[Arge, Brodal, Evangelista, Kristensen, Westergaard]

- **Uses backtracking and state reconstruction of full state descriptors to guarantee full coverage.**
- **Reconstruction is achieved by augmenting the hash compaction method:**
  - A state number is assigned to each visited state.
  - The state table stores for each compressed state descriptor a collision list of state numbers.

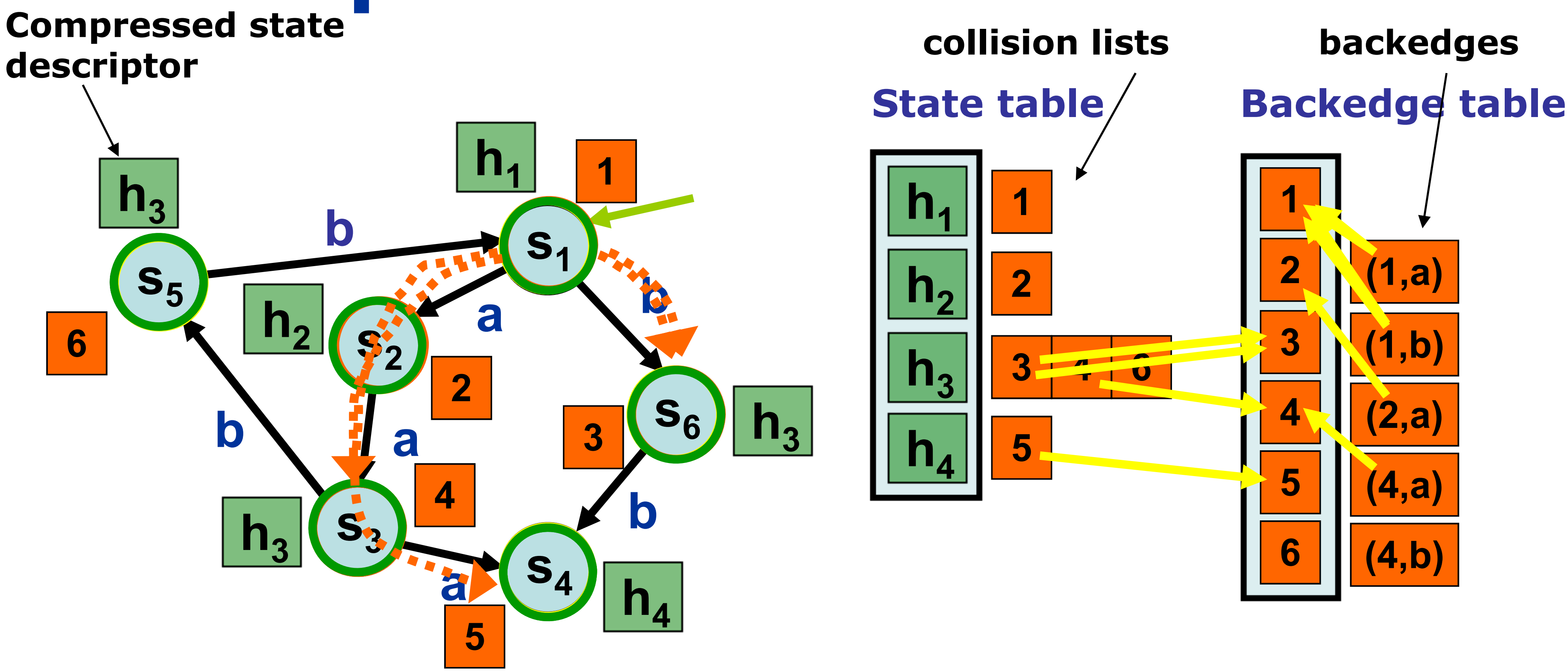
    to detect (potential) hash collisions
  - A backedge table stores a backedge for each state number of a visited state.

    to reconstruct full state descriptors



State space of the mobile1 example

Example: The ComBack Method



State Reconstruction

3 (1,b) → $S_6 \neq S_3$ 4 (2,a) (1,a) → $S_3 \neq S_5$

3 (1,b) → $S_6 \neq S_5$ 5 (4,a) (2,a) (1,a) → $S_4 = S_4$

Comback Main Theorem

- **ComBack algorithm terminates after having processed all reachable states exactly one.**
- **The elements in the state table and the backedge table can be represented using:**

$$|\mathsf{reach}(s_I)| \cdot (w_H + 3 \cdot \lceil \log_2 |\mathsf{reach}(s_I)| \rceil + \lceil \log_2 |T| \rceil) \ bits$$

Overhead compared to hash compaction

- **Number of state reconstructions bounded by:**

$$\max_{h_k \in \hat{H}} |\hat{h}_k| \cdot \sum_{s \in \mathsf{reach}(s_I)} in(s\ )$$

Experimental Results

ComBack performance relative to standard DF full state space exploration

Model	Method	Nodes	Arcs	DFS %Time	DFS %Space	BFS %Time	BFS %Space
SW	ComBack	215,196	1,242,386	178	42	258	48
	HashComp	214,569	1,238,803	92	12	103	23
	Standard	215,196	1,242,386	100	100	111	100
TS	ComBack	107,648	1,017,490	383	85	198	30
	HashComp	107,647	1,017,474	93	75	96	24
	Standard	107,648	1,017,490	100	100	106	73
ERDP	ComBack	207,003	1,199,703	180	34	353	42
	HashComp	206,921	1,199,200	93	6	100	21
	Standard	207,003	1,199,703	100	100	115	101
ERDP	ComBack	4,277,126	31,021,101	-	-	-	-
	HashComp	4,270,926	30,975,030	-	-	-	-

- **Typically reduces memory usage to 30% at the cost of doubling the state space exploration time.**

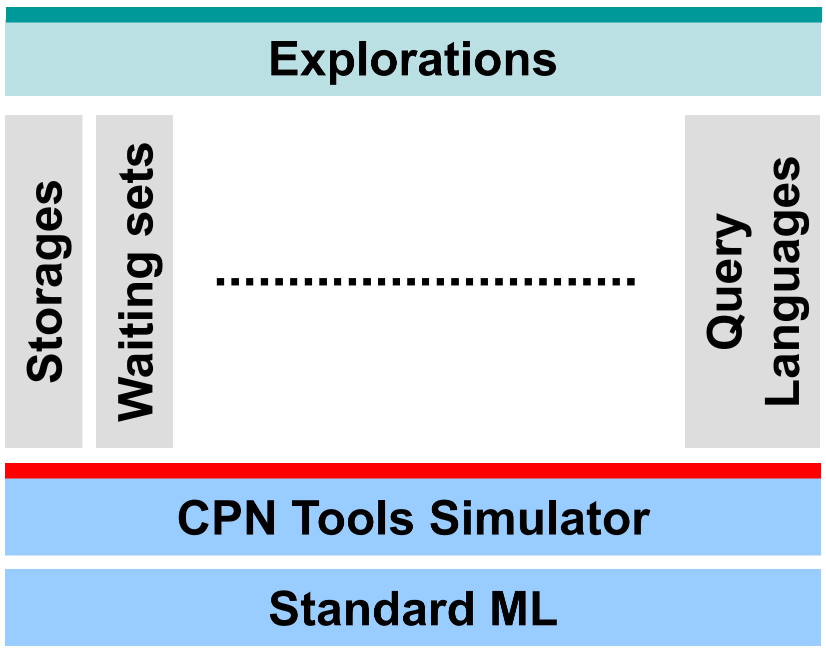
Summary

- **The sweep-line method has been successfully applied to a number of real protocols:**
  - **Internet protocols: WAP, IOTP, TCP, and DCCP.**
  - **Progress is generally easy to identify and there is no proof obligation attached.**
- **The comback method:**
  - **Search-order independent and transparent state reconstruction: compatible with most state space methods.**
  - **Experiments suggest that it represents a good time-space trade-off.**
  - **The Comback method is suited for late phases of the verification process.**

Access/CPN Framework

[http://wiki.daimi.au.dk/ascoveco/accesscpn.wiki]

- **JAVA and Standard ML interface providing access to the CPN model and the transitions relation:**



State Space Exploration Engine

M. Westergaard and L.M. Kristensen. *The Access/CPN Framework: A Tool for Interacting with the CPN Tools Simulator*. In Proc. ICATPN'09, Vol. 5606 of Springer Lectures Notes in Computer Science, pp. 313-322. Springer-Verlag, 2009.

References

- L.M. Kristensen. A Perspective on Explicit State Space Exploration of Coloured Petri Nets - Past, Present, and Future. In Proc. of ICATPN'10. Lectures Notes in Computer Science, Springer, 2010. Invited talk.
- S. Evangelista, M. Westergaard, and L.M. Kristensen The ComBack Method Revisited: Caching Strategies and Extension with Delayed Duplicate Detection. In of Transactions on Petri Nets and Other Models of Concurrency Vol. 3, pp. 189-215. Subseries of LNCS, Springer, 2009.
- M. Westergaard, S. Evangelista, and L.M. Kristensen. ASAP: An Extensible Platform for State Space Analysis. In Proc. of ICATPN'09, Vol. 5606 of Springer Lectures Notes in Computer Science, pp. 303-312. Springer-Verlag, 2009.
- M. Westergaard and L.M. Kristensen. The Access/CPN Framework: A Tool for Interacting with the CPN Tools Simulator. In Proc. of ICATPN'09, Vol. 5606 of Springer Lectures Notes in Computer Science, pp. 313-322. Springer-Verlag, 2009.
- M.Westergaard, L.M. Kristensen, G. Brodal, and L. Arge. The ComBack Method – Extending Hash Compaction with Backtracking. In Proc. of ICTAPN'07, Vol. 4546 of Lectures Notes in Computer Science, pp. 445-464. Springer-Verlag, 2007.
- L.M. Kristensen and T. Mailund. A Generalised Sweep-Line Method for Safety Properties. In Proc. of FME'02. Vol. 2391 of Lecture Notes in Computer Science, pp. 549-567. Springer-Verlag, 2002.

The CPN Simulator Interface

- **Defines how to access to the transition relation of CPN models:**

```
signature MODEL_SIMULATOR = sig

 eqtype state
 eqtype event

 (* --- get the initial state --- *)
 val getInitialState : unit -> state

 (* --- get the enabled events and successor states --- *)
 val nextStates : state -> (event * state) list

end
```

- **Makes it possible to extent CPN Tools and experiment with new state space methods.**

State Representation

- **Reflects the hierarchical structure of the CPN model:**

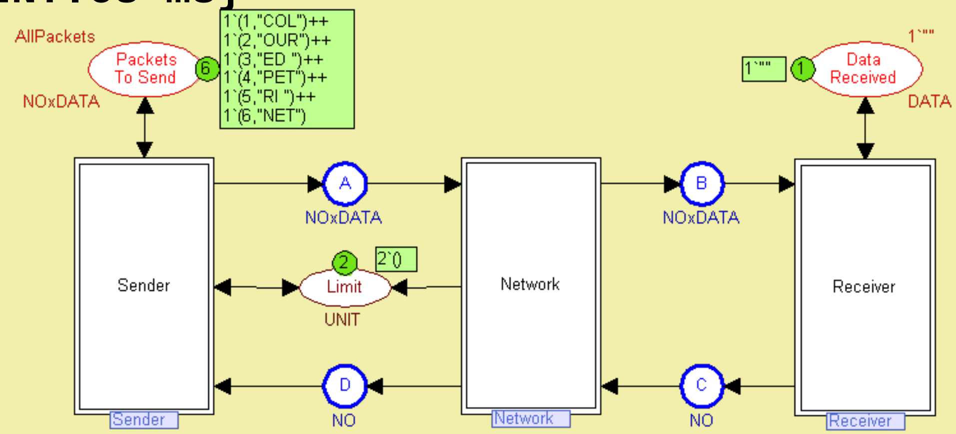
```
type Receiver = {NextRec  : INT.cs ms}
type Network  = {}
type Sender   = {NextSend : INT.cs ms}

type Protocol =
      {A : NOxDATA.cs ms,
       B : NOxDATA.cs ms,
       C : INT.cs ms,
       D : INT.cs ms,

       Limit : UNIT.cs ms,
       Packets_To_Send : NOxDATA.cs ms,
       Data_Received   : STRING.cs ms,

       Network  : Network, Receiver : Receiver, Sender : Sender}

type state = { Protocol : Protocol }
```



Event Representation

```
datatype event
   Network'Transmit_Ack of
                    int * {n : INT.cs, success : BOOL.cs}

 | Network'Transmit_Packet of
                    int * {d : DATA.cs, n : INT.cs, success : BOOL.cs}

 | Receiver'Receive_Packet of
                    int * {d : DATA.cs, data : DATA.cs,
                           k : INT.cs,  n : INT.cs}

 | Sender'Receive_Ack of int * {k : INT.cs,  n : INT.cs}

 | Sender'Send_Packet of int * {d : DATA.cs, n : CPN'ColorSets.IntCS.cs}
end
```