Structured Occurrence Nets

Brian Randell and Maciej Koutny
Newcastle University, UK

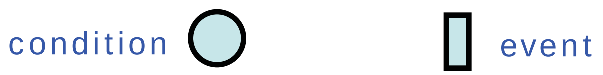
Summary

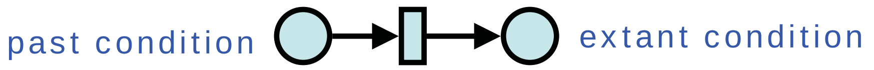
- We introduce the concept of a Structured Occurrence Net (SON), based on that of an ‘occurrence net’ (ON) - a well-established formalism for an abstract record that represents causality and concurrency information concerning a single execution of a system.
- SONs consist of multiple related ONs, and are intended for recording either actual system behaviour, or evidence concerning alleged past behaviour.
- We show how SONs can enable better understanding of complex fault-error-failure chains (i) among co-existing interacting and evolving systems, (ii) between systems and their sub-systems, and (iii) involving systems that are controlling, supporting, creating or modifying other systems.
- We discuss how, perhaps using extended versions of existing tools, SONs could form a basis for improved techniques of system failure prevention and analysis.

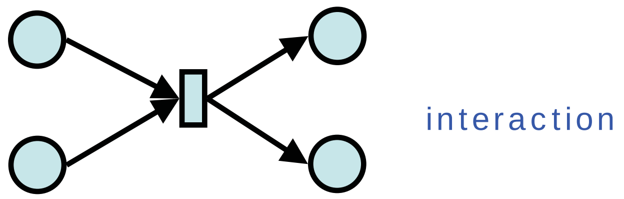
Occurrence Nets

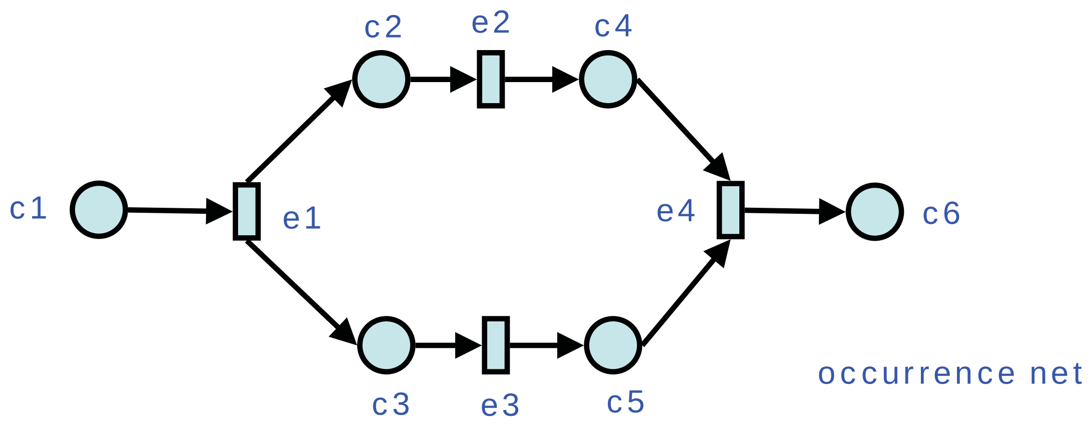
- ONs are directed acyclic graphs that portray the (alleged) past and present state of affairs, in terms of places (i.e. conditions, represented by circles), transitions (i.e. events, represented by rectangles) and arrows (each from a place to a transition, or from a transition to a place, representing (alleged) causality).
- Occurrence nets look like “unwound” Petri Nets, but have no necessary link to PNs.
- For simple nets, an actual graphical representation suffices. (In the case of complex nets, these are better represented in some linguistic or tabular form.)
- What we have realised is that ‘system’ and ‘state’ are not separate concepts, but just a question of abstraction, so that (different related) occurrence nets can represent both systems and their states using the same symbol - a ‘place’. (We make use of this to deal with evolving systems.)
- In fact in this talk we introduce and define, and discuss the utility of, several types of relation between occurrence nets, since - in the interests of complexity reduction - we also employ related sets of relatively simple occurrence nets to substitute for very large single occurrence nets.

(Graphical) Representation of ONs









Occurrence Nets and System Failures

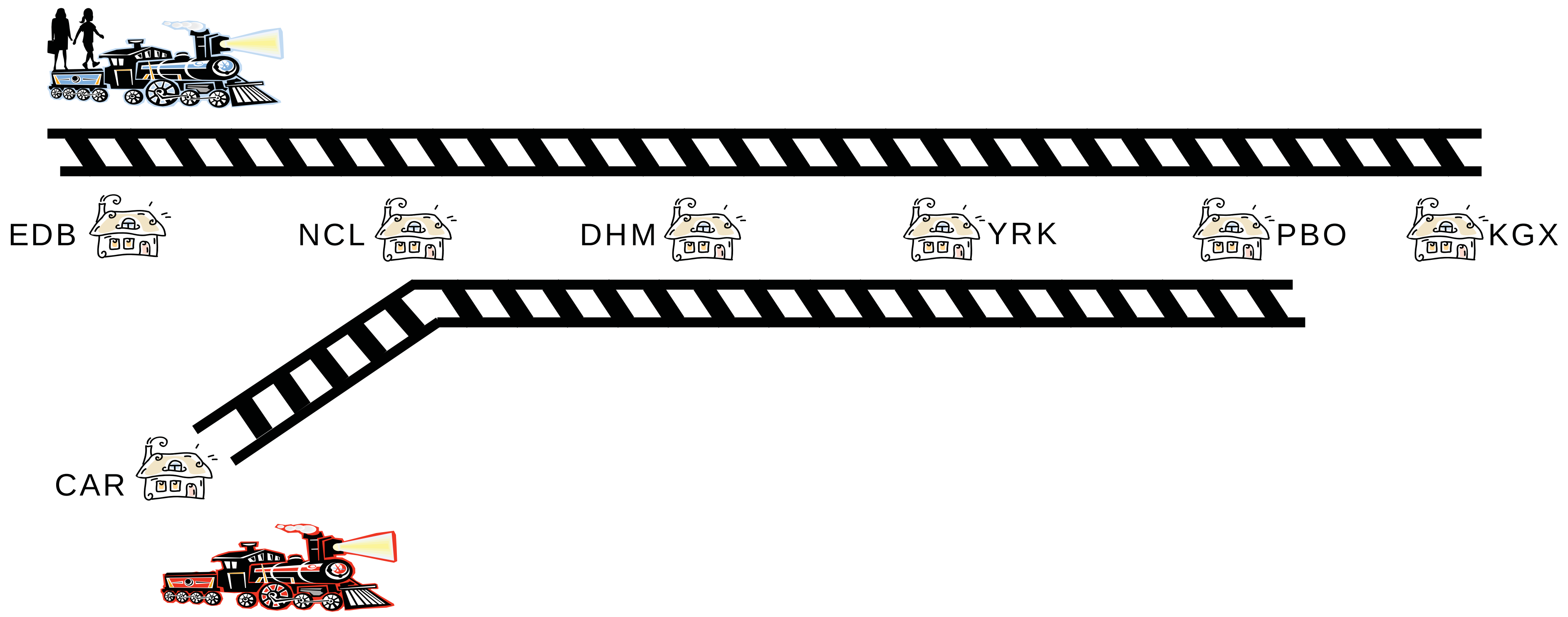
- Occurrence Nets are a convenient way of recording or visualising the activity of a system, in particular the flow of errors within a system caused by a fault.
- We say a failure occurs when an error passes through the system-user interface and affects the service delivered by the (compound) system
- This failure may constitute a fault to the enclosing system
- The manifestation of failures/faults/errors thus follows a chain:

  . . . -> failure -> fault -> error -> failure -> fault ->. . .
- This chain can flow from one system to:
  - another system that it is interacting with
  - the system which it is part of
  - a system which it creates, modifies, or sustains
- A single huge occurrence net *could* be used to record or visualize the combined and perhaps concurrent activities of a whole set of error-prone systems, and hence to investigate such fault/error/failure chains among such systems.
- Occurrence nets can readily portray situations in which there are multiple faults - (a failure may be judged to be due to multiple co-incident faults, e.g. the activity of a hacker exploiting a bug left by a programmer)
- But the problems of modelling evolving systems, and the complexity of situations involving multiple interacting and evolving systems, have motivated our introduction of structured occurrence nets.

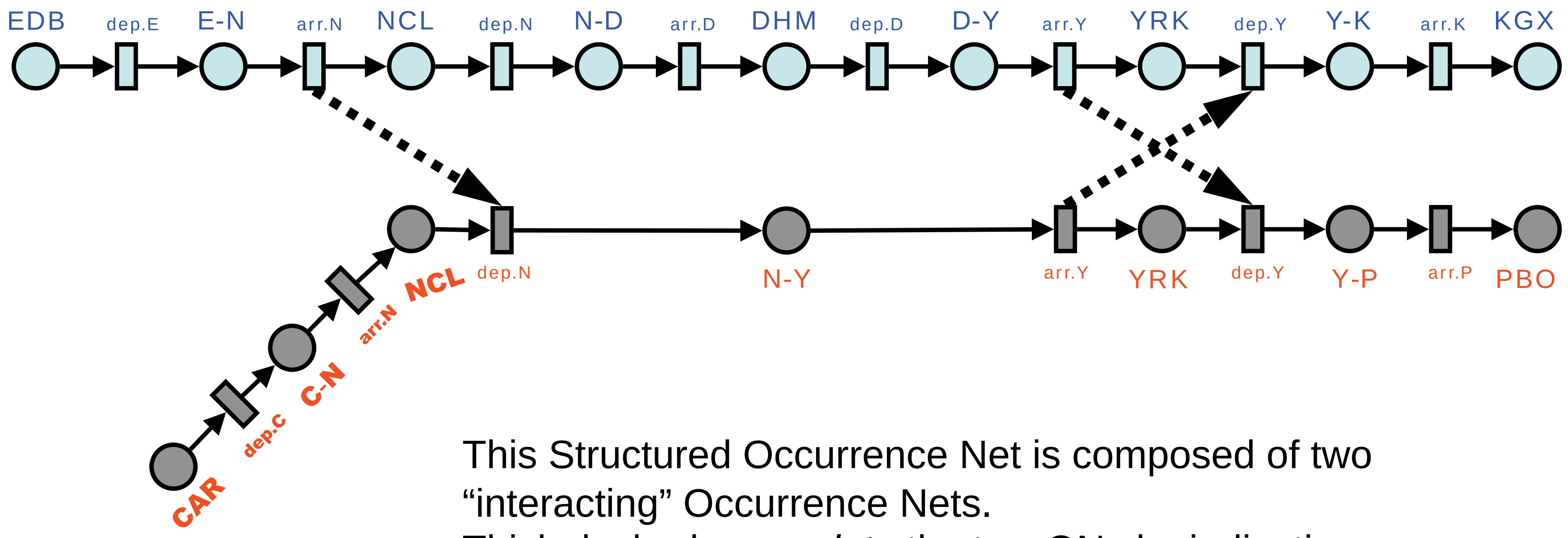
Structured Occurrence Nets

- A Structured Occurrence Net is the term we use for a *related* set of Occurrence Nets (using several specific forms of relation)
- The Occurrence Nets used in a SON are in fact coloured directed acyclic graphs, the colouring being used in order that the states of different systems can be distinguished.
- The various relations we have defined are all such that SON's, like ONs, are acyclic - and so respect the causality rules.
- The significance of SONs is that (i) their structuring reduces their complexity, compared to that of an equivalent ON, and (ii) they provide a direct means of modelling evolving systems.
- These advantages can we believe facilitate such tasks as system evaluation (via model-checking), system synthesis, and system failure analysis. In this presentation we concentrate just on system failure analysis.
- And here the SON idea is illustrated not just with computer systems, but also railway systems!

Portraying (the activity of) a train system

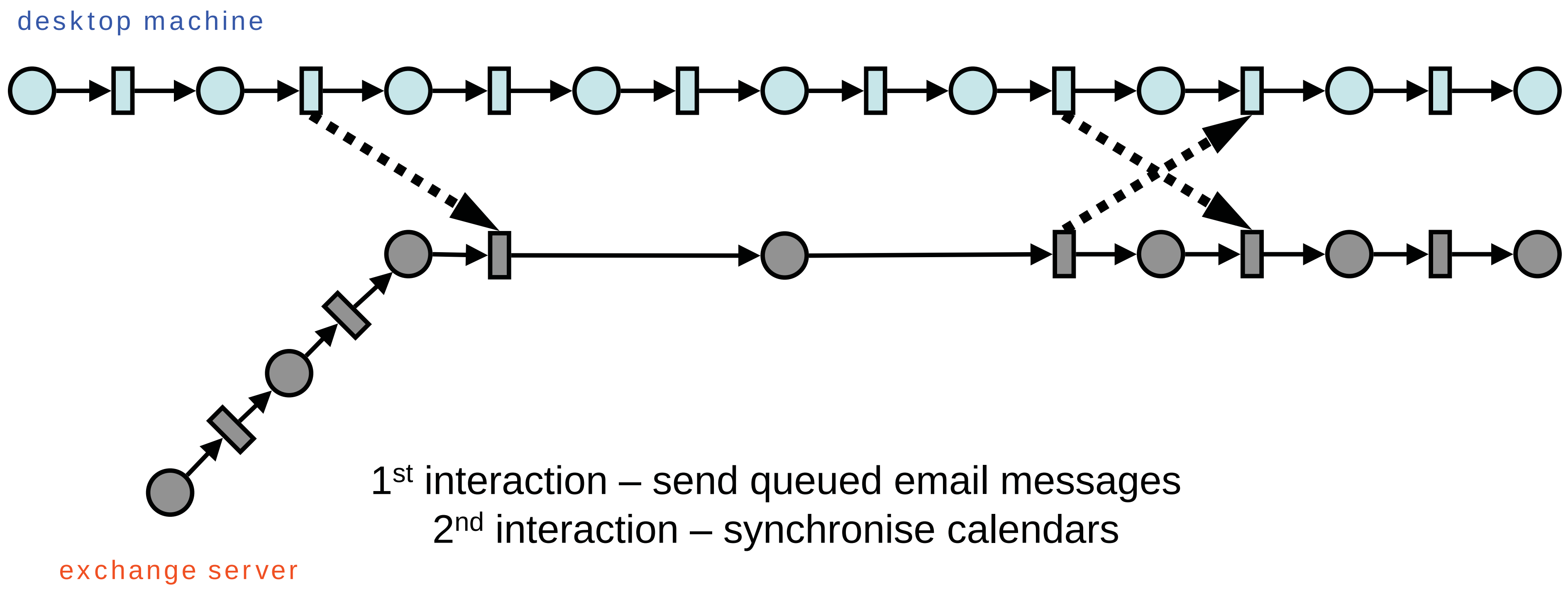


A SON Portrayal of these Train Journeys



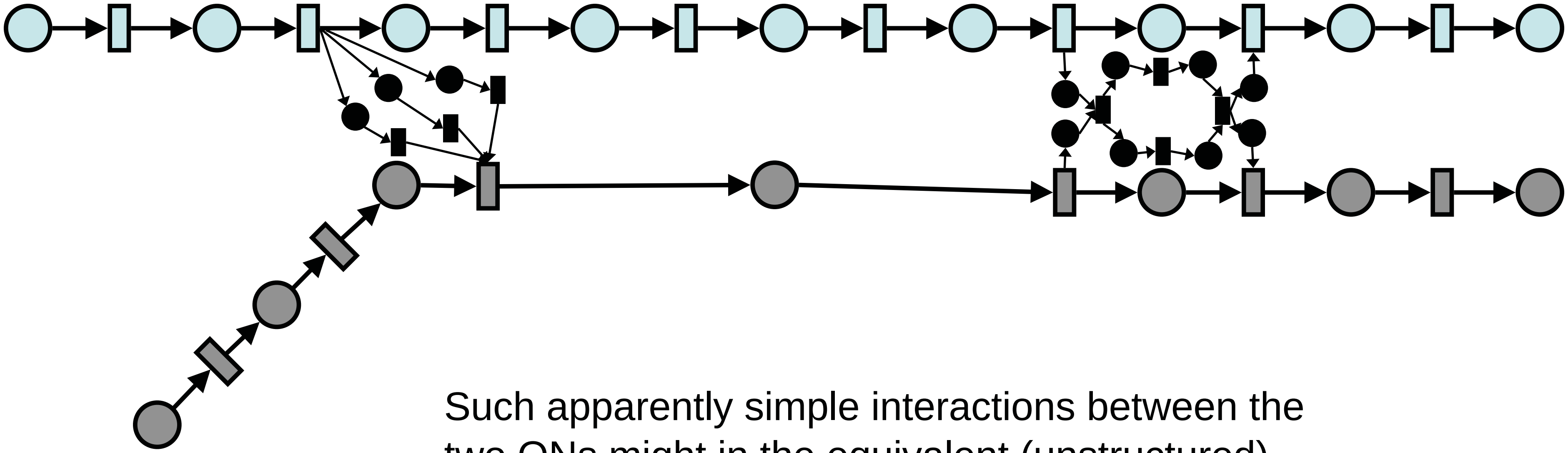
This Structured Occurrence Net is composed of two "interacting" Occurrence Nets.
Thick dashed arcs *relate* the two ONs by indicating events in one ON that are causal predecessors of events in the other ON. (Thus they indicate that there was unidirectional information flow - of passengers!)

But the portrayal could be of interacting computing systems



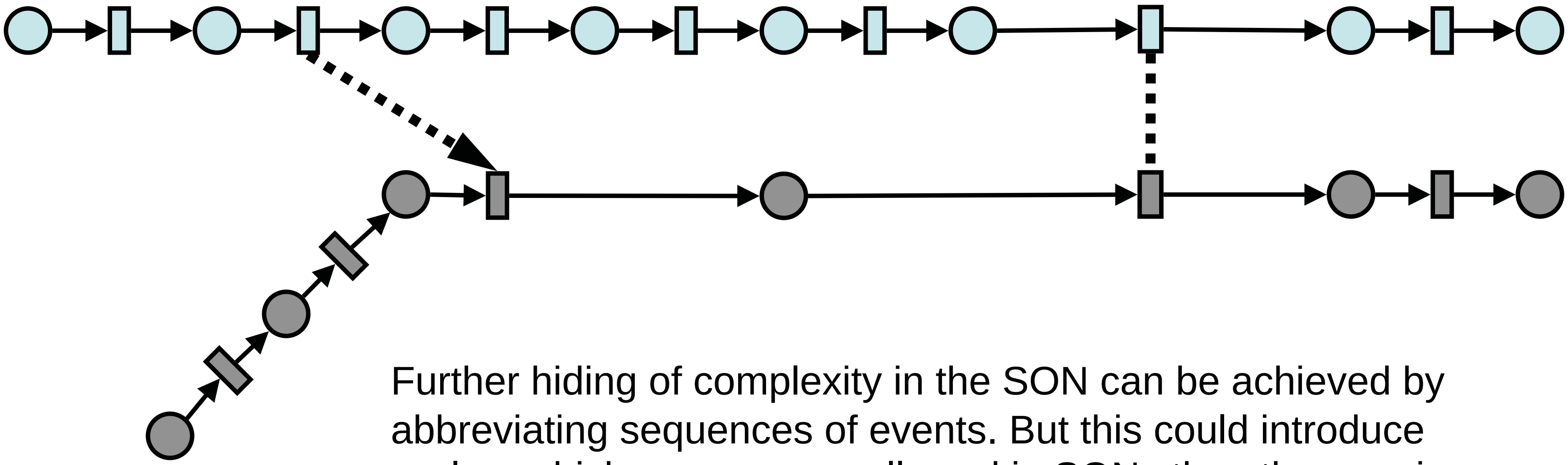
1st interaction – send queued email messages
2nd interaction – synchronise calendars

Detailed Interactions in a single ON



Such apparently simple interactions between the two ONs might in the equivalent (unstructured) single large ON involve extensive sequences of interactions, whose complexity is merely hinted at above.

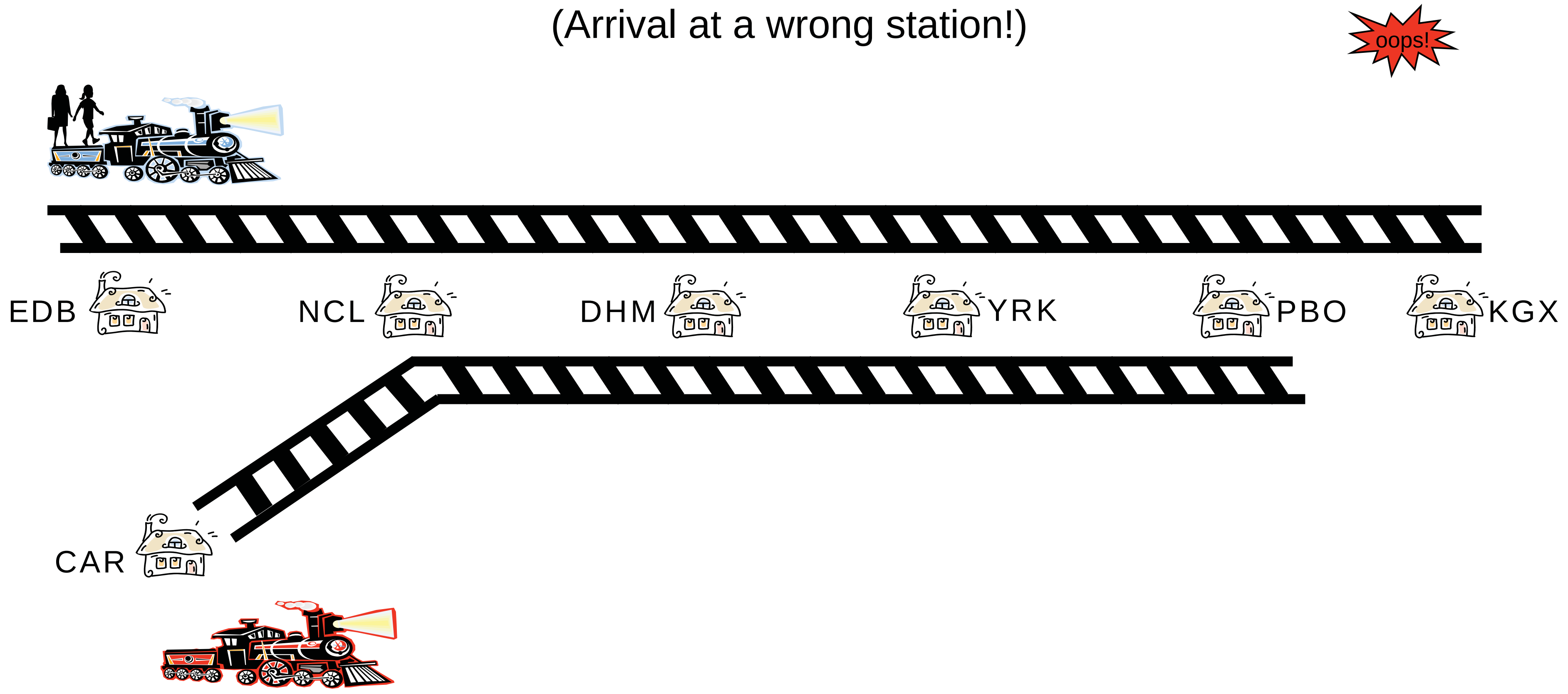
Synchronous interactions between ONs



Further hiding of complexity in the SON can be achieved by abbreviating sequences of events. But this could introduce cycles, which are no more allowed in SONs than they are in ONs - this motivates the introduction of the concept of undirected (i.e. *synchronous*) interactions between ONs. (Asynchronous interactions between ONs are shown by thick dashed *arcs*, synchronous ones by thick dashed *edges*.)

Portraying failure of a (train) system

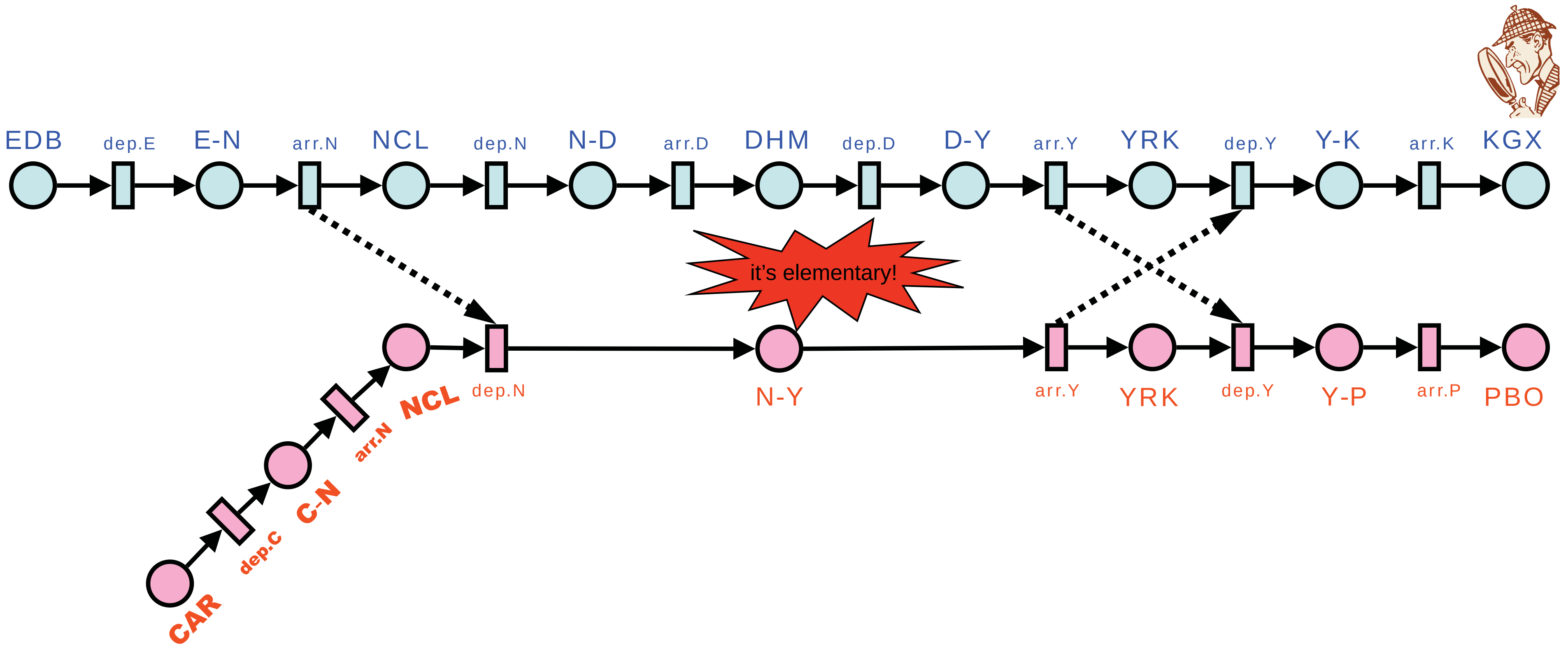
(Arrival at a wrong station!)



Failure Analysis 1

- Failure analysis can involved following links in ONs backwards in order to identify causes (faults), and then forwards to identify further errors.
- Arcs and edges (asynchronous and synchronous interactions) between ONs in a SON can similarly be followed in each direction, to trace fault/error/failure chains between systems, e.g. between interacting systems.
- Other types of relations between ONs (defined later) can also be involved in such analysis
- However, the actual identification of errors and faults requires additional information, e.g. obtained from system specifications.

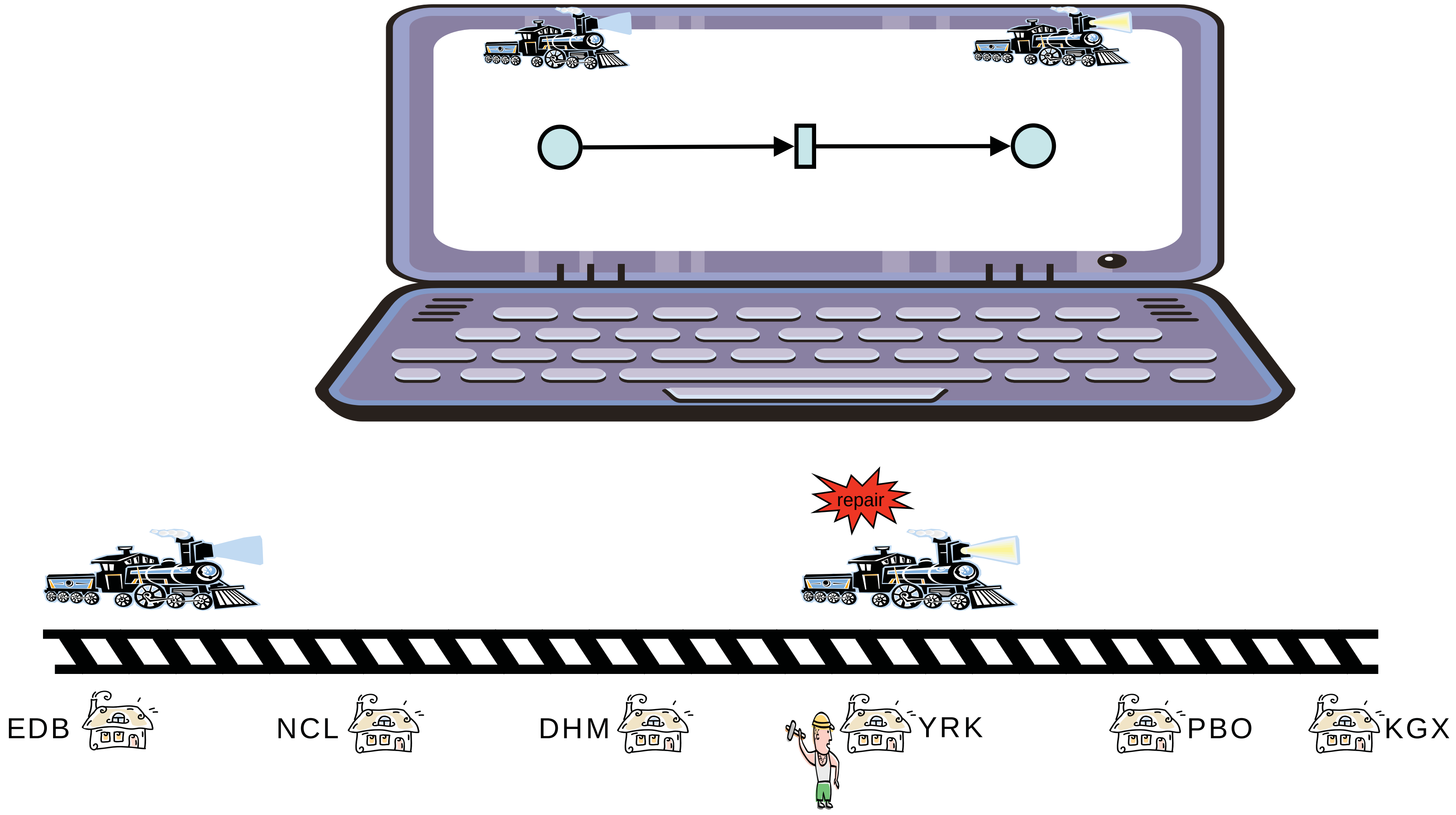
Investigating the causes of failure



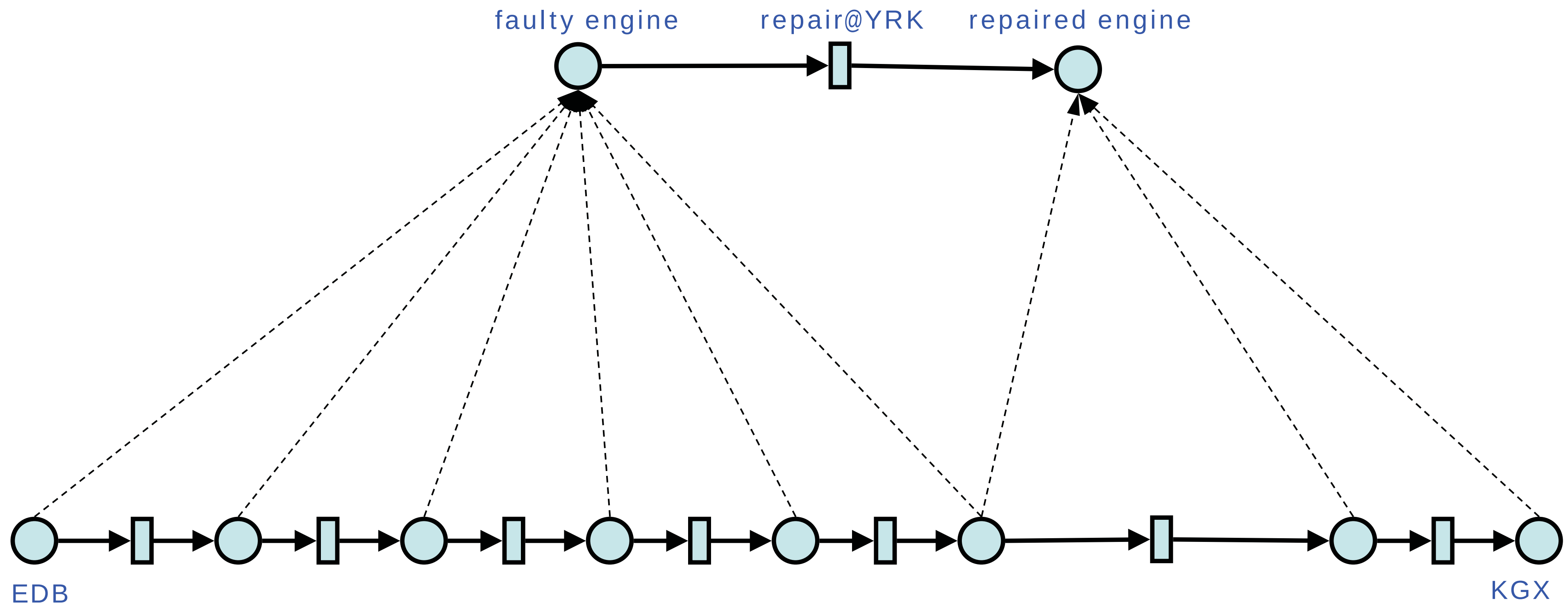
Behavioral Abstraction

- Any condition can be viewed either as a state (of some system), or as a system itself (that presumably has its own states) - which is just a matter of viewpoint.
- Thus one could have two related ONs, one showing how a system is evolving (e.g. being modified), the other showing the behaviours of each version of this system. In fact the first ON can be viewed as the **behavioural abstraction** of the second ON.
- Such behavioural abstraction is a further type of relation that can be used to construct a SON out of multiple ONs.
- However, unlike the other relations, such as interaction, it enables us to portray histories that could not be shown using just a single very large ON.

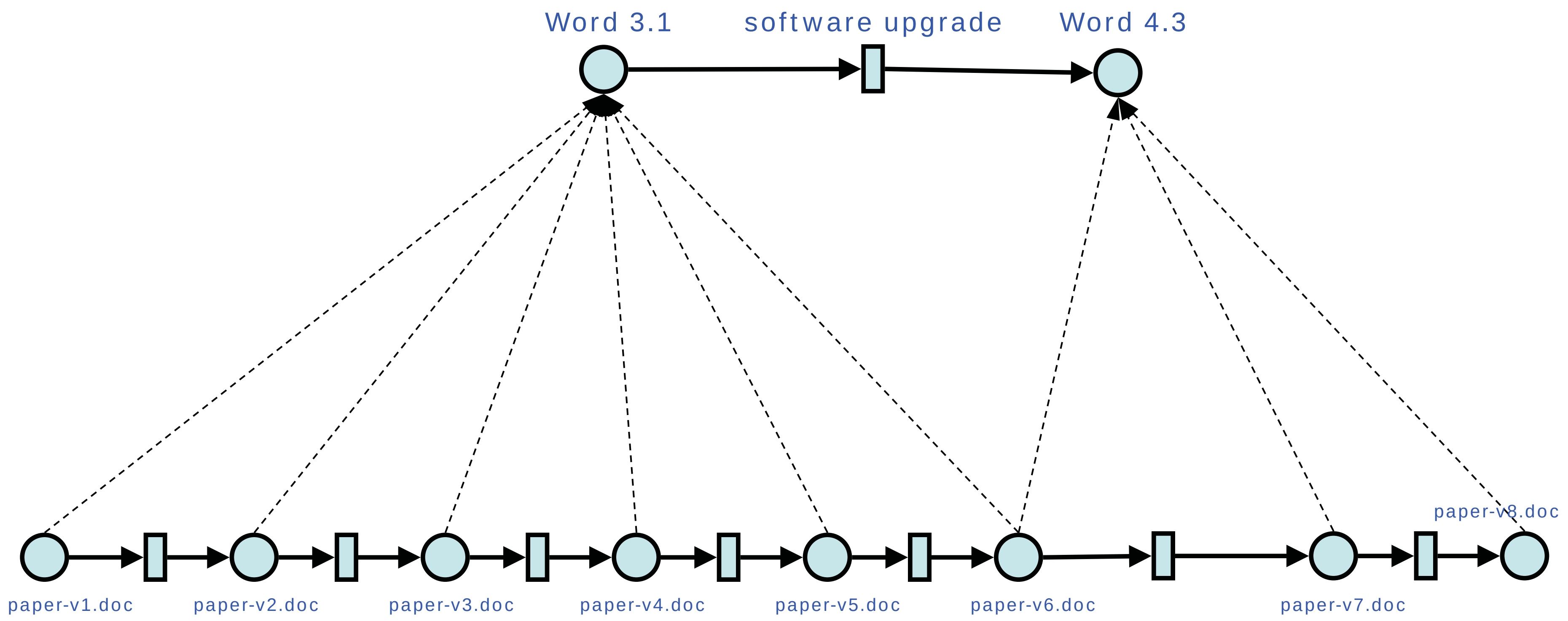
A Two-Level view of a Train (and its Repair)



A SON Portraying such Train Repair

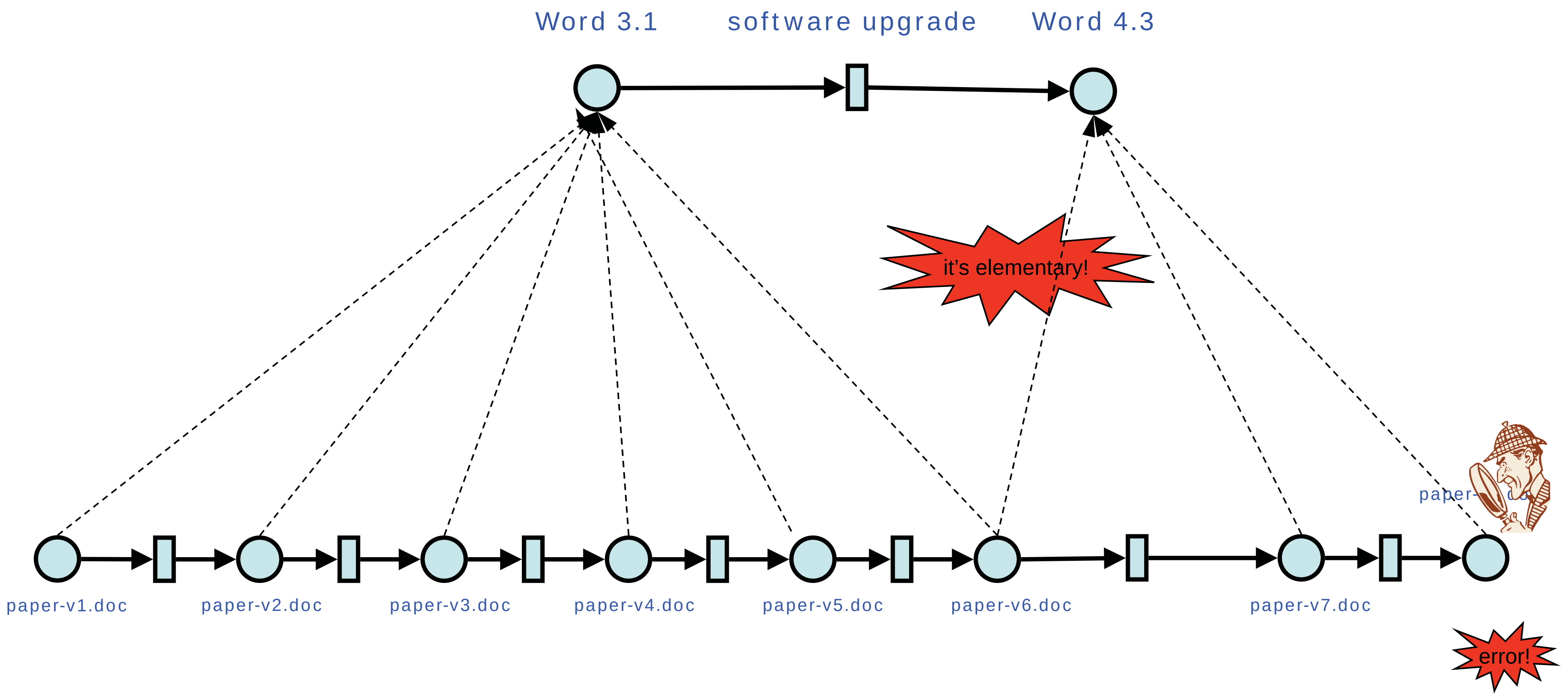


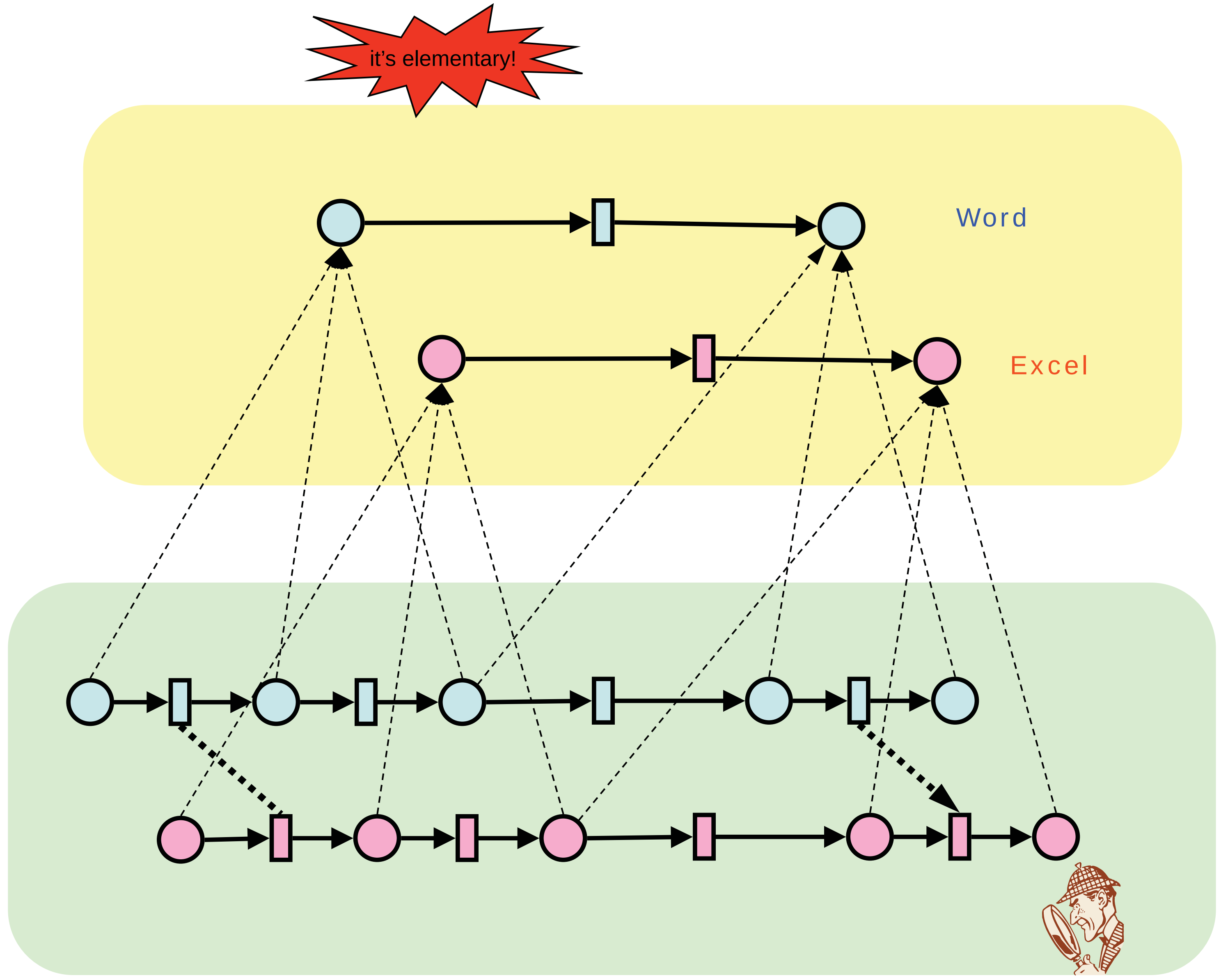
Software Modification



Note - A SON that involves behavioural abstraction cannot be represented in ordinary occurrence nets

Investigating failure of an evolving system



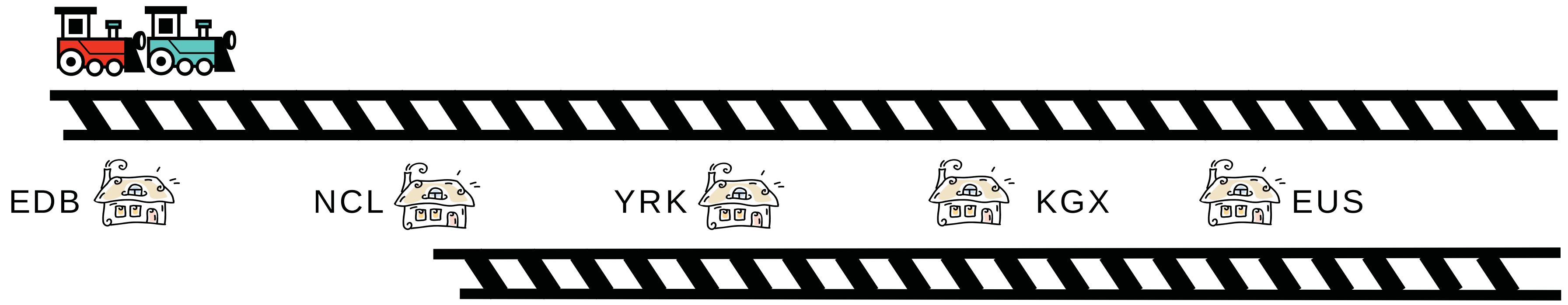


A combined investigation: intra- and inter-level error in a spreadsheet has been caused by a Word version change

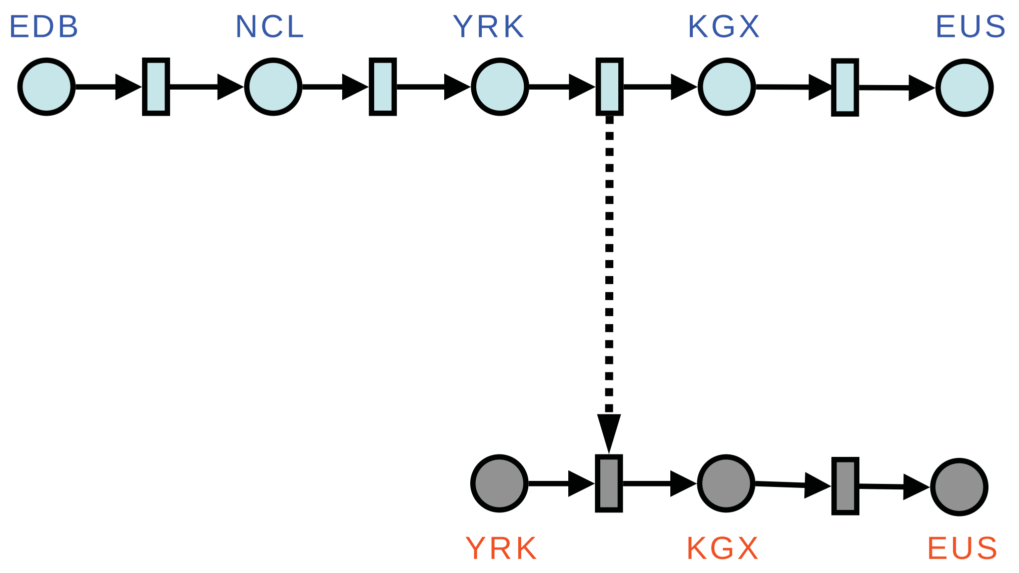
And system A begat system B . . .

The next example shows that one system has spawned another system, and after that both systems went through some independent further evolutions - and indicates how the latest versions of these systems have interacted (first as an animation, then using a SON).

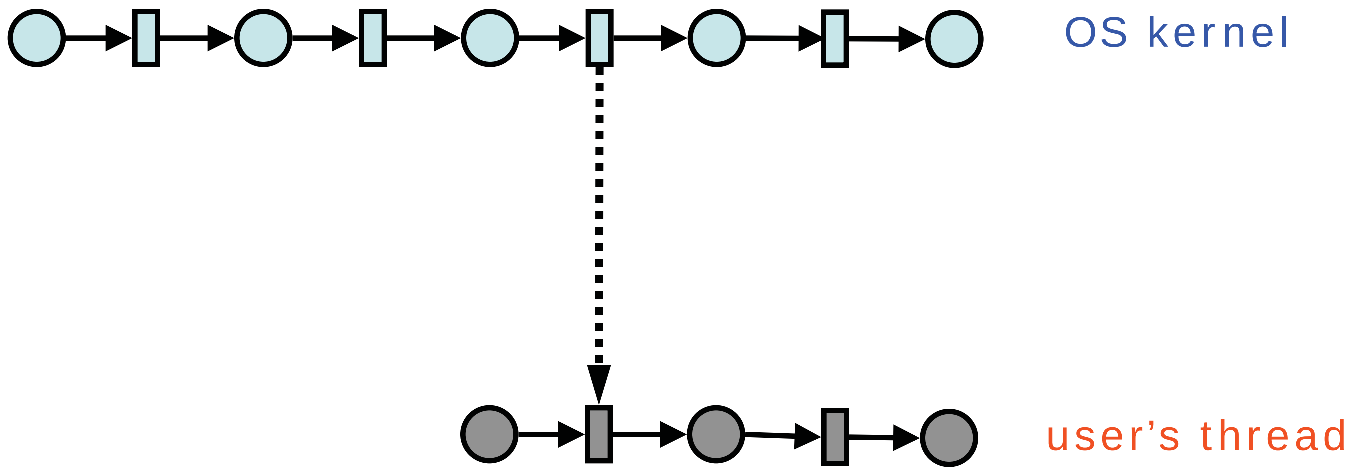
“Creating” Trains



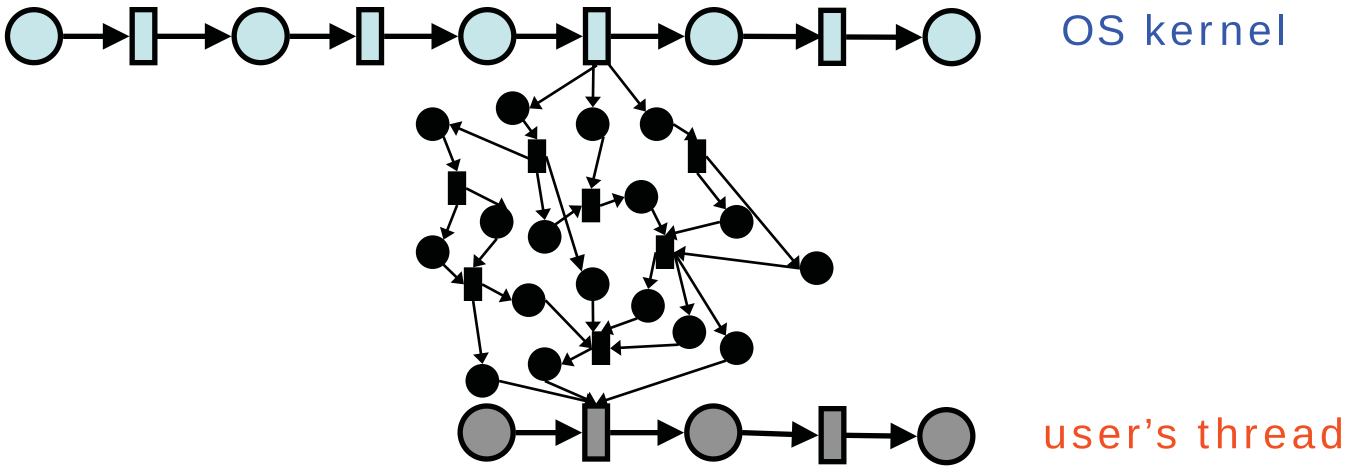
Portraying System Creation, using a SON



And showing a software example . . .



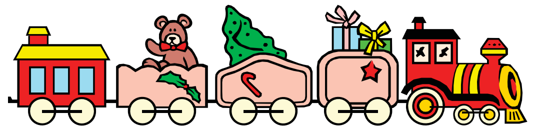
. . . though hiding the hideous details

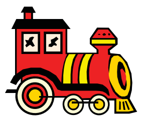


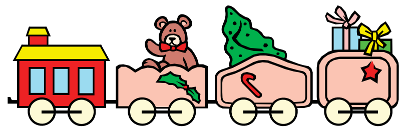
Compositional (Spatial) Abstraction

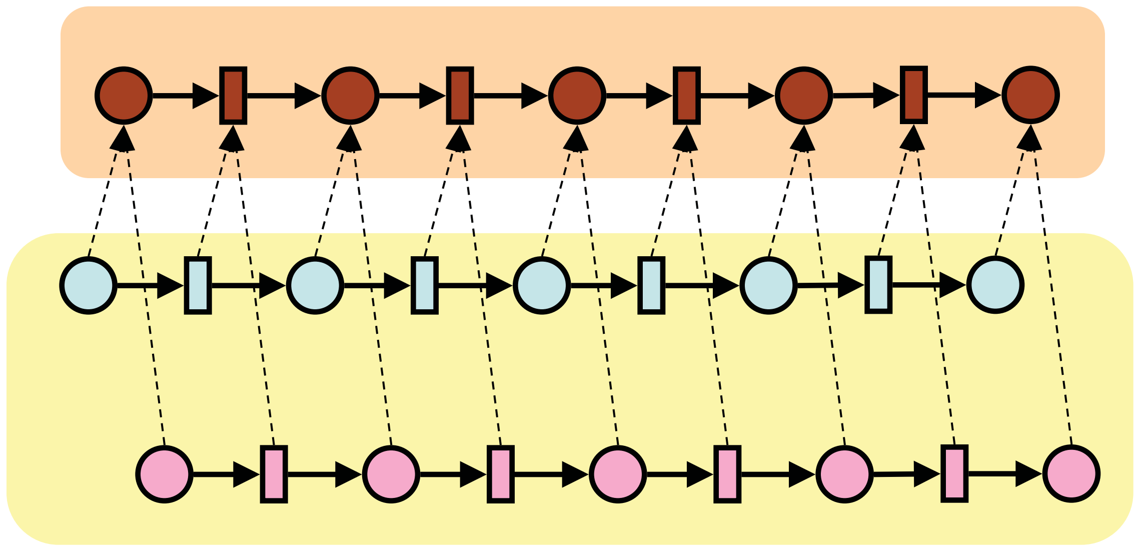
This shows the behaviour of a system and of its component systems, and how its behaviour is related to that of its components. (It does not represent the matter of how, or indeed whether, the component systems are enabled to interact, i.e., what design is used, or what connectors are involved.) Each component system has the other(s) as its *environment.*

“Composing” a train







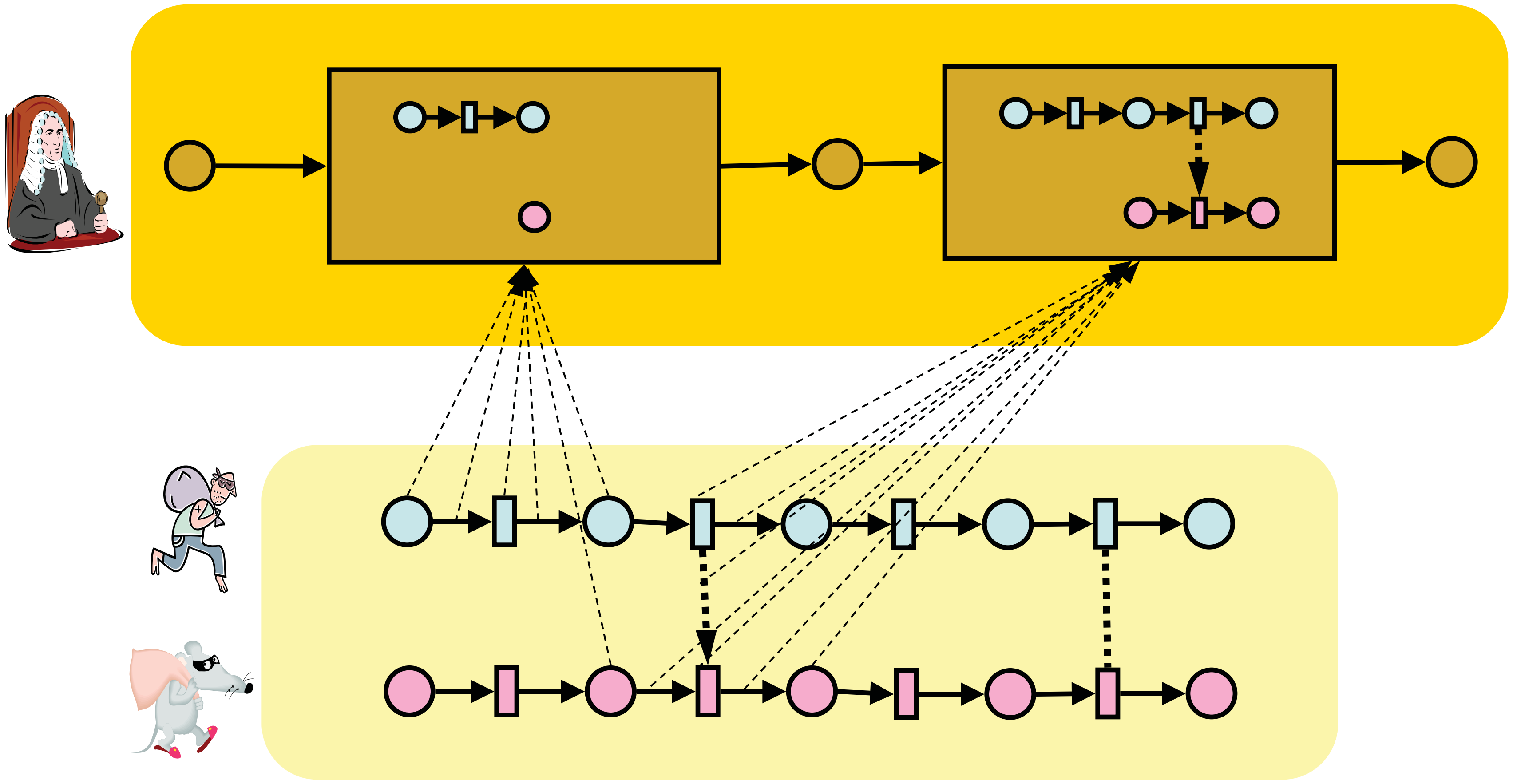


The composition relation links all the events in the components to events in the composed system, and all the states in the components to ones in the composed system

Recovery Points

To allow for the possibility of failure a system might, e.g., make use of ‘recovery points’. Such recovery points can be recorded in retained states that take no further (direct) part in the system’s ongoing (normal) behaviour.

Information retention

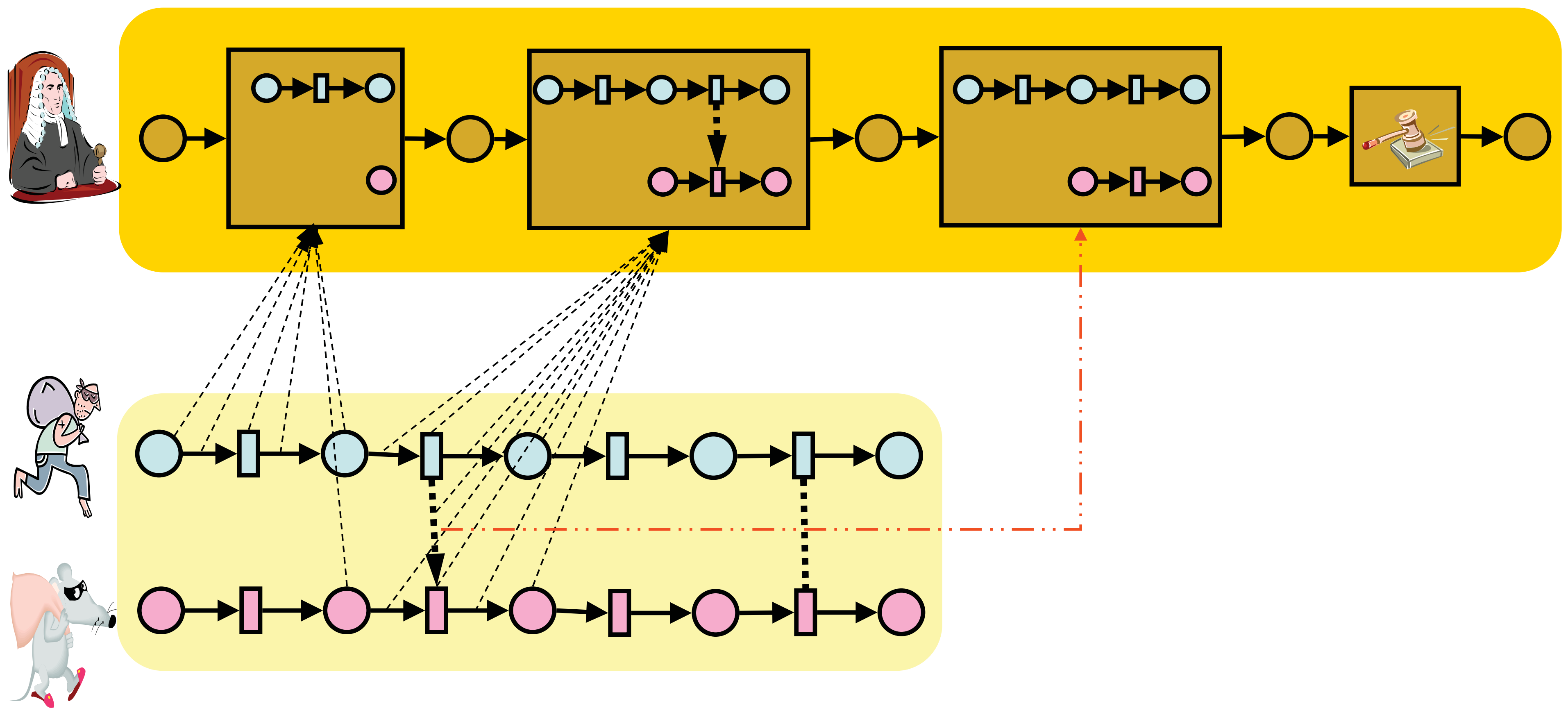


Judgemental Systems

- The notion of a ‘failure’ event involves, in principle, three systems — the given (possibly failing) system, its environment, and a judgemental system.
- The judgemental system may interact directly and immediately with the given system, in which case it is part of the system’s environment, e.g., a built-in checking circuit, or in a very different world, a football referee!
- Alternatively the judgemental system may be deployed after the fact using an occurrence net that represents how the failing event (is thought to have) occurred.
- Such an occurrence net can be recorded in a retained state, e.g., that of the judgment system.

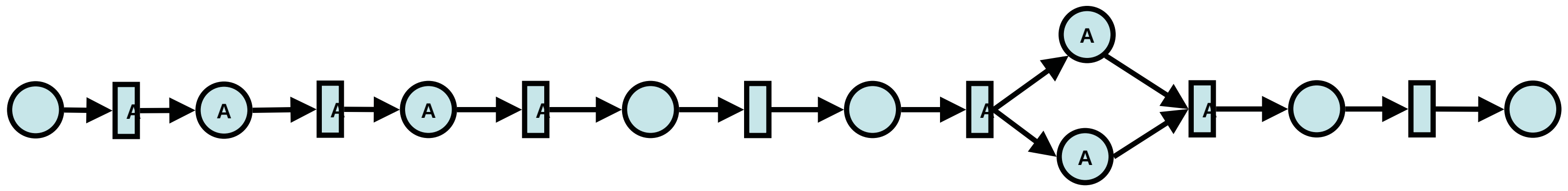
Post-hoc Judgement

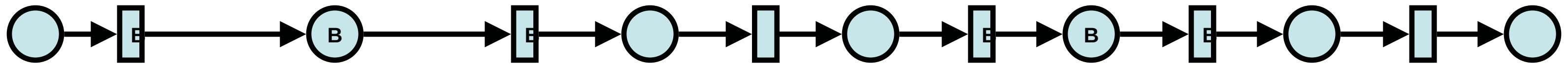
Interpretation: the witness to the alibi withdrew his testimony



The judgement system has obtained only incomplete evidence of the systems’ states and events and even the causal relationships between conditions and events.

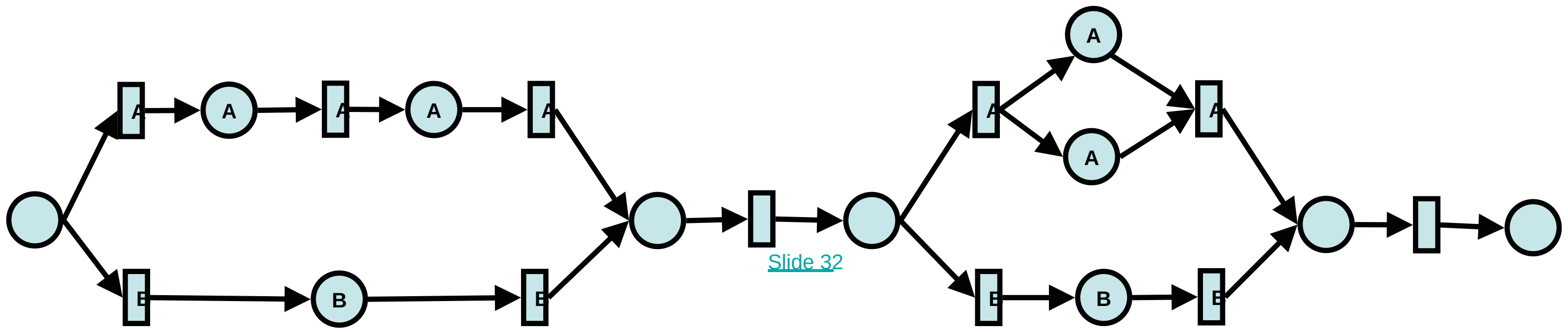
Alternative (Assumed) Scenarios





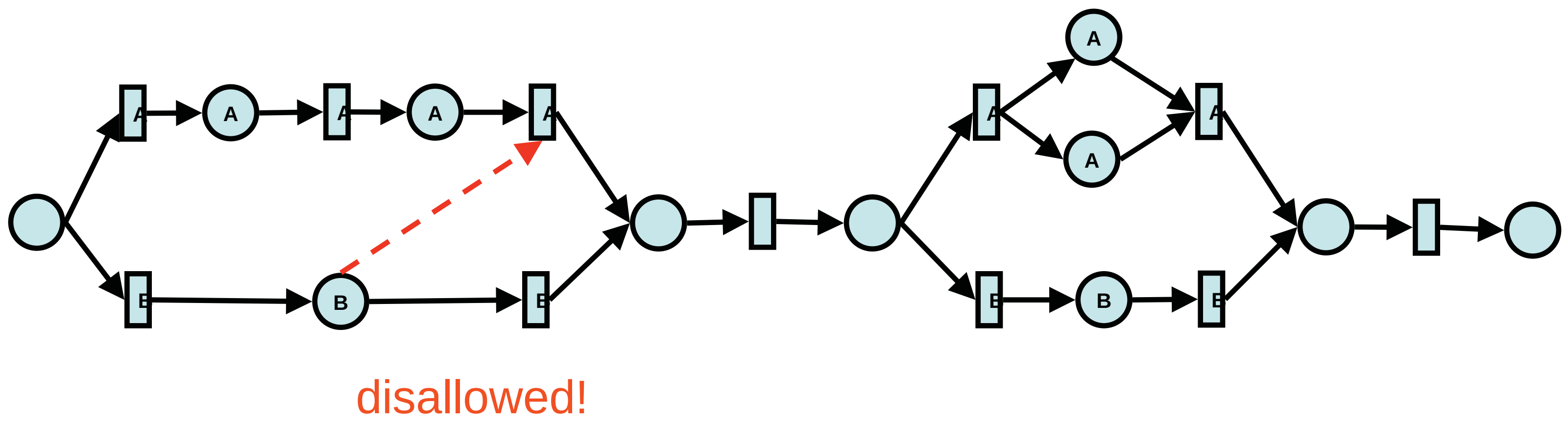
- Differing evidence or interpretations can lead to differing portrayals of a given system’s activity
- In the two ONs shown above the letters ‘A’ and ‘B’ distinguish between two alternative views of some of the states and events that feature in these portrayals of a given system’s activity.

Incorporating Alternatives in a single SON



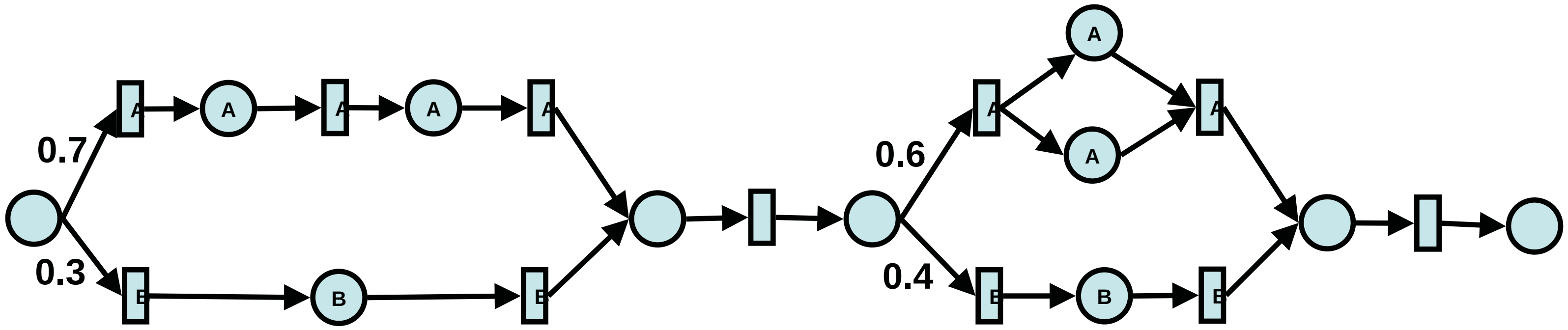
- We allow the simultaneous modelling of multiple alternative scenarios, within a single ON, when there is insufficient evidence to indicate which particular scenario actually occurred.
- Again using the letters ‘A’ and ‘B’ distinguish between two alternative possible system activities, the above ON shows two ways in which the given system’s final state might have come about have been envisaged and are being represented in a single ON.
- This ON can be regarded as a combination of the two alternative ONs, but is more likely to have been developed by adding alternatives to an initial simpler ON.

An Assumed Scenario Incorporating Alternative Possibilities - but . . .



Alternative chains of activity within a single ON represent happenings in what are in effect different “worlds”, and it would not make sense to have any interactions, however indirect, between such worlds

Uncertain Evidence



It could be useful to be able to annotate particular events, conditions, links and relations with some form of probability estimate – indicating the current degree of certainty a judge has about the accuracy of their representation.

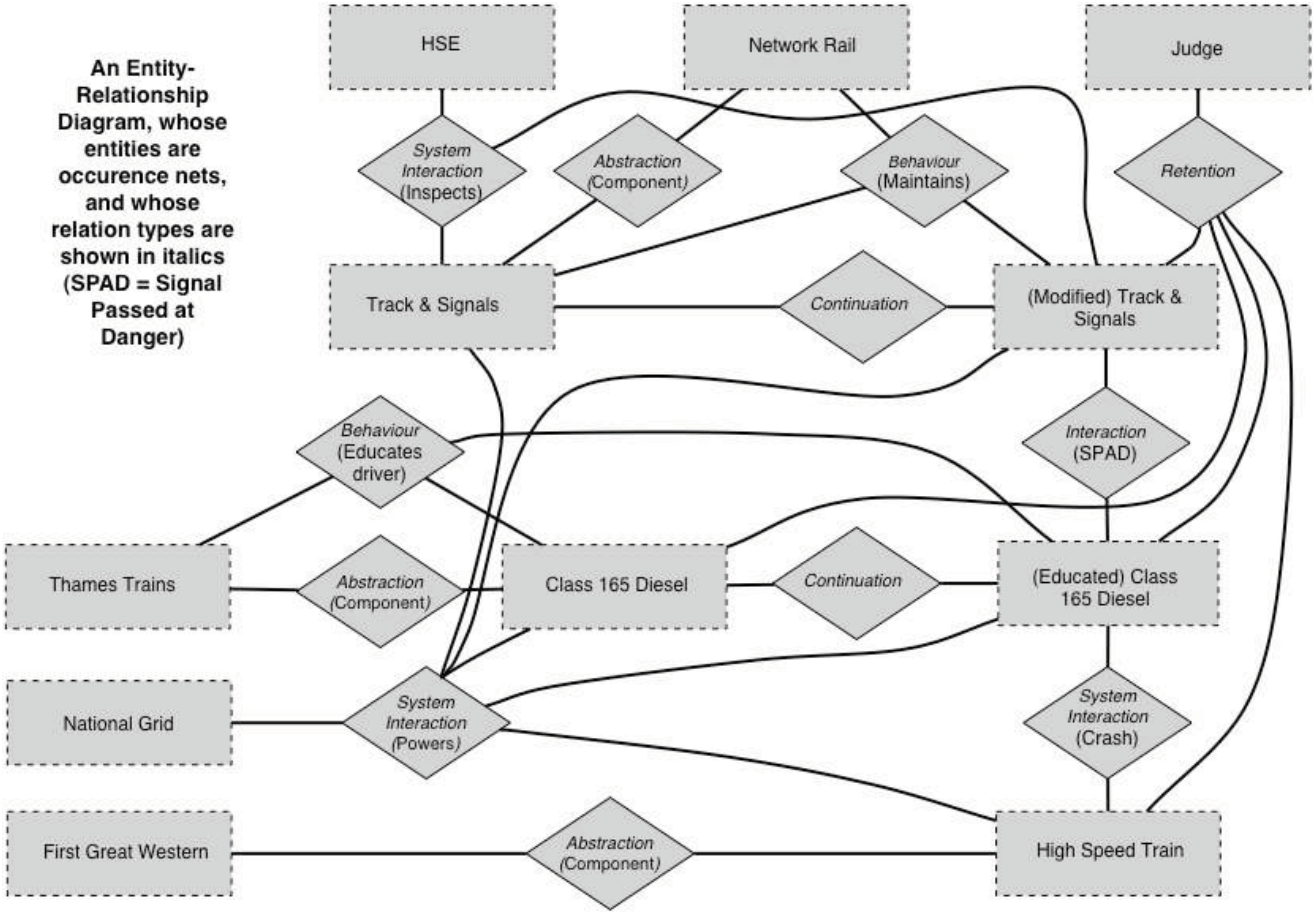
Failure Analysis 2

- SONs could be used to represent actual or assumed past behaviour, or possible future behaviour, and to record F-E-F chains between systems.
- They could be generated and recorded (semi?) automatically – alternatively they might need to be generated retrospectively, from whatever evidence and testimony is available.
- Analysis of a SON typically involves following (possibly in both directions) causal arrows *within* ONs, and the various different sorts of relations *between* ONs.
- Such analysis is of course limited by the accuracy and the completeness of the SON – and might be interspersed with efforts at validating and enhancing the SON.

Our one “experiment” to date

- Ladbrooke Grove was the scene of a bad railway accident in October 1999, when a three-car Class 165 diesel train operated by Thames Trains collided with a First Great Western High Speed Train
- The immediate cause of the disaster - the diesel train passed a particular signal when red.
- A lengthy enquiry identified many more issues, and many systems (rail companies, government organizations, people, trains, etc.) were implicated.
- As a (thought) experiment we have considered how the huge mass of evidence considered by the enquiry could be represented and analyzed.
- We have used the conventional Entity-Relationship graphical notation, the entities in fact being individual (un-detailed) occurrence nets, representing information about the activities of each of the systems involved, the whole being a *very* large SON.
- Our belief is that, with the right tool support, such a SON could greatly aid the documentation and analysis of such a complex failure situation.

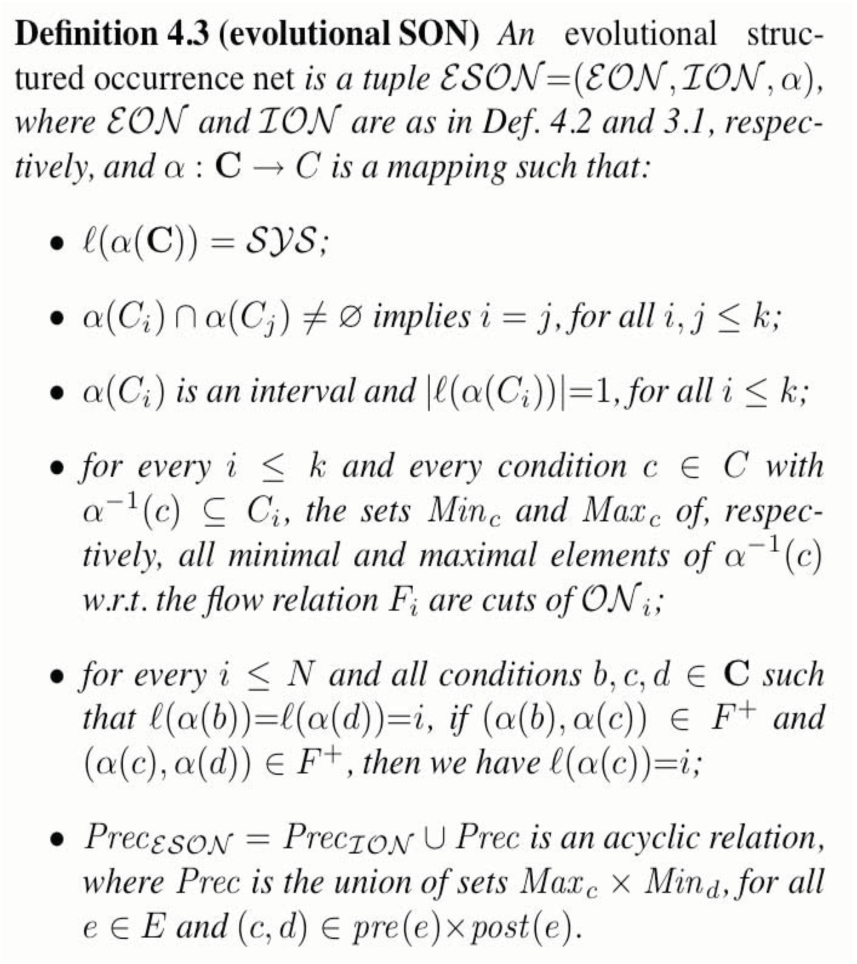
An Entity-Relationship Diagram, whose entities are occurence nets, and whose relation types are shown in italics (SPAD = Signal Passed at Danger)


HSE
Network Rail
Judge
System Interaction (Inspects)
Abstraction (Component)
Behaviour (Maintains)
Retention
Track & Signals
Continuation
(Modified) Track & Signals
Behaviour (Educates driver)
Interaction (SPAD)
Thames Trains
Abstraction (Component)
Class 165 Diesel
Continuation
(Educated) Class 165 Diesel
National Grid
System Interaction (Powers)
System Interaction (Crash)
First Great Western
Abstraction (Component)
High Speed Train

Concluding Remarks 1

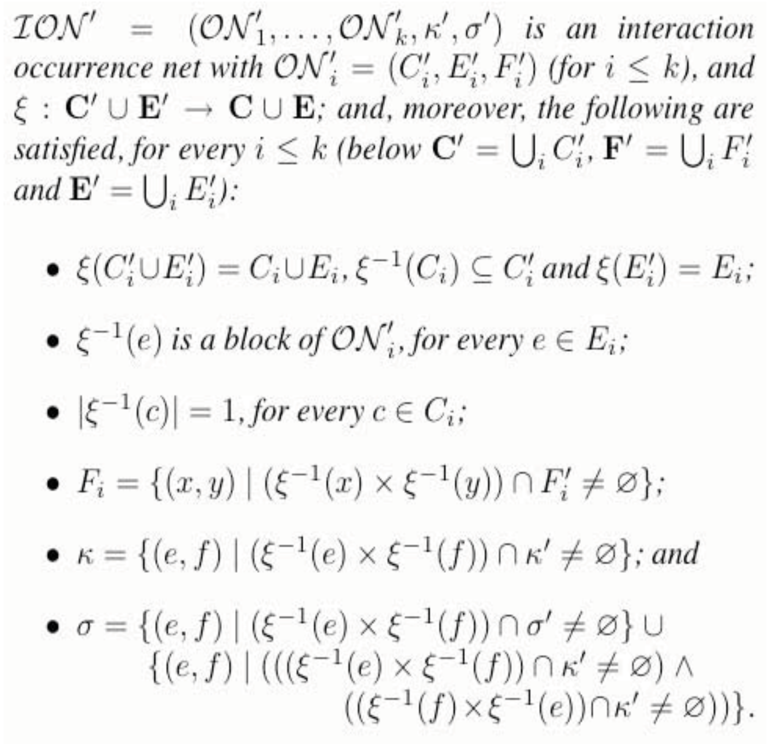
- Our various types of abstractions are all ones that could facilitate the task of understanding complex systems and their failures, and analyzing the cause(s) of such failures.
- They would in most cases be a natural consequence of the way the systems have been conceived and perceived. Thus they can be viewed as providing a means of naturally structuring what would otherwise be an impossibly large and complex occurrence net.
- Alternatively, they can be viewed as a way of reducing the combinatorial complexity of the information accumulated and the analyses performed in following fault-error-failure chains after the fact.
- In either case, computer assistance is needed, something we plan to investigate, building on existing work at Newcastle and elsewhere.
- We provide the formalizations of the various types of abstraction that are needed as a starting point for this investigation. (It's not just a set of pretty pictures!)

Examples of What You've Been Spared



Definition 4.3 (evolutional SON) *An* evolutional structured occurrence net *is a tuple* $\mathcal{ESON}=(\mathcal{EON},\mathcal{ION},\alpha)$, *where* $\mathcal{EON}$ *and* $\mathcal{ION}$ *are as in Def. 4.2 and 3.1, respectively, and* $\alpha : \mathbf{C} \to C$ *is a mapping such that:*

- $\ell(\alpha(\mathbf{C})) = \mathcal{SYS}$;
- $\alpha(C_i) \cap \alpha(C_j) \neq \varnothing$ *implies* $i = j$, *for all* $i, j \leq k$;
- $\alpha(C_i)$ *is an interval and* $|\ell(\alpha(C_i))|=1$, *for all* $i \leq k$;
- *for every* $i \leq k$ *and every condition* $c \in C$ *with* $\alpha^{-1}(c) \subseteq C_i$, *the sets* Min_c *and* Max_c *of, respectively, all minimal and maximal elements of* $\alpha^{-1}(c)$ *w.r.t. the flow relation* F_i *are cuts of* $\mathcal{ON}_i$;
- *for every* $i \leq N$ *and all conditions* $b, c, d \in \mathbf{C}$ *such that* $\ell(\alpha(b))=\ell(\alpha(d))=i$, *if* $(\alpha(b),\alpha(c)) \in F^+$ *and* $(\alpha(c),\alpha(d)) \in F^+$, *then we have* $\ell(\alpha(c))=i$;
- $Prec_{\mathcal{ESON}} = Prec_{\mathcal{ION}} \cup Prec$ *is an acyclic relation, where* $Prec$ *is the union of sets* $Max_c \times Min_d$, *for all* $e \in E$ *and* $(c,d) \in pre(e) \times post(e)$.



$\mathcal{ION}' = (\mathcal{ON}'_1,\ldots,\mathcal{ON}'_k,\kappa',\sigma')$ *is an interaction occurrence net with* $\mathcal{ON}'_i = (C'_i, E'_i, F'_i)$ *(for* $i \leq k$*), and* $\xi : \mathbf{C}' \cup \mathbf{E}' \to \mathbf{C} \cup \mathbf{E}$; *and, moreover, the following are satisfied, for every* $i \leq k$ *(below* $\mathbf{C}' = \bigcup_i C'_i$, $\mathbf{F}' = \bigcup_i F'_i$ *and* $\mathbf{E}' = \bigcup_i E'_i$*):*

- $\xi(C'_i \cup E'_i) = C_i \cup E_i$, $\xi^{-1}(C_i) \subseteq C'_i$ *and* $\xi(E'_i) = E_i$;
- $\xi^{-1}(e)$ *is a block of* $\mathcal{ON}'_i$, *for every* $e \in E_i$;
- $|\xi^{-1}(c)| = 1$, *for every* $c \in C_i$;
- $F_i = \{(x,y) \mid (\xi^{-1}(x) \times \xi^{-1}(y)) \cap F'_i \neq \varnothing\}$;
- $\kappa = \{(e,f) \mid (\xi^{-1}(e) \times \xi^{-1}(f)) \cap \kappa' \neq \varnothing\}$; *and*
- $\sigma = \{(e,f) \mid (\xi^{-1}(e) \times \xi^{-1}(f)) \cap \sigma' \neq \varnothing\} \cup \{(e,f) \mid (((\xi^{-1}(e) \times \xi^{-1}(f)) \cap \kappa' \neq \varnothing) \wedge ((\xi^{-1}(f) \times \xi^{-1}(e)) \cap \kappa' \neq \varnothing))\}$.

Concluding Remarks 2

- The messages we’re trying to put over are that we have a formal notation for portraying failure-prone activities of sets of evolving systems, a notation that is intended to be able to deal with very complex situations;
- The purposes for such notations are not just system failure analysis but also system validation and system synthesis
- Our notation isn’t yet fully sufficient for any for these three things (... look for the text ...) but it can provide a powerful infrastructure /basis for analysis algorithms, etc.
- We have tried one interesting thought experiment - in examining how we might use a SON to aid analysis of the Ladbroke Grove Rail Crash, in fact by identifying a possible set of ONs and their relationships.
- But we also argue that SONs could be of utility, through their role in complexity reduction, in tools for system validation (through model-checking), and system synthesis - but that’s another talk!

Some of our References

- Best, E. and Randell, B. (1981). A Formal Model of Atomicity in Asynchronous Systems, *Acta Informatica*, Vol. 16 (1981), pp 93-124. Springer-Verlag Germany. http://www.cs.ncl.ac.uk/research/pubs/articles/papers/397.pdf
- Chatain, T. and Jard, C. (2004). Symbolic Diagnosis of Partially Observable Concurrent Systems. Proc. of FORTE'04, LNCS 3235, 326–342.
- Grahlmann, P and Best, E: PEP - More than a Petri net tool. Proc. of TACAS'96, LNCS 1055, 1996, pp.397-401 [PEP]
- Holt, A.W., Shapiro, R.M., Saint, H., and Marshall, S., "Information System Theory Project", Appl. Data Research ADR 6606 (US Air Force, Rome Air Development Center RADC-TR-68-305), 1968.
- Khomenko, V. and Koutny, M.: Branching Processes of High-Level Petri Nets, Proc. of TACAS'03, LNCS 2619, 2003 pp.458-472, http://www.cs.ncl.ac.uk/research/pubs/articles/papers/425.pdf
- Merlin, P.M. and Randell, B. State Restoration in Distributed Systems, In *Proc FTCS-8*, Toulouse, France, 21-23 June 1978 pp. 129-134. IEEE Computer Society Press 1978 http://www.cs.ncl.ac.uk/research/pubs/articles/papers/347.pdf
- Randell, B. and Koutny, M. Failures: Their Definition, Modelling and Analysis In *Theoretical Aspects of Computing - ICTAC 2007*. Macao, China, September 26-28, 2007 Jones, C.B., Liu, Z. and Woodcock, J. (eds.) LNCS, 4711, pp 260-274 Springer-Verlag, 2007 http://www.cs.ncl.ac.uk/research/pubs/trs/papers/994.pdf