Bayesian Networks of Dynamic Systems

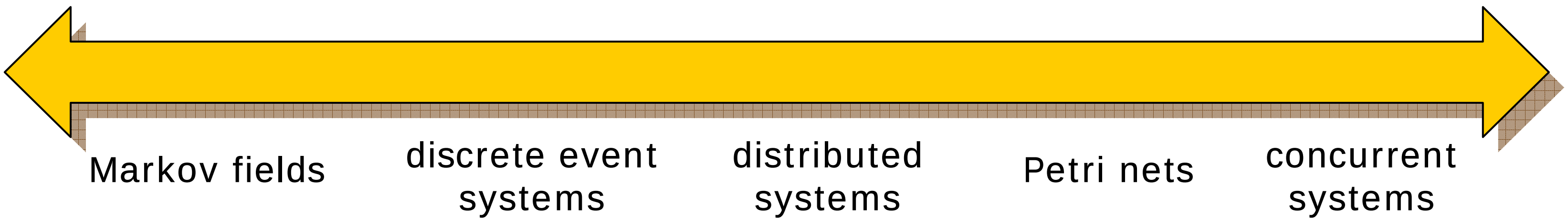
Eric Fabre
DistribCom team
IRISA/INRIA

HDR Defense – June 14, 2007

Bayesian Networks

of Dynamic Systems

Bayesian Networks

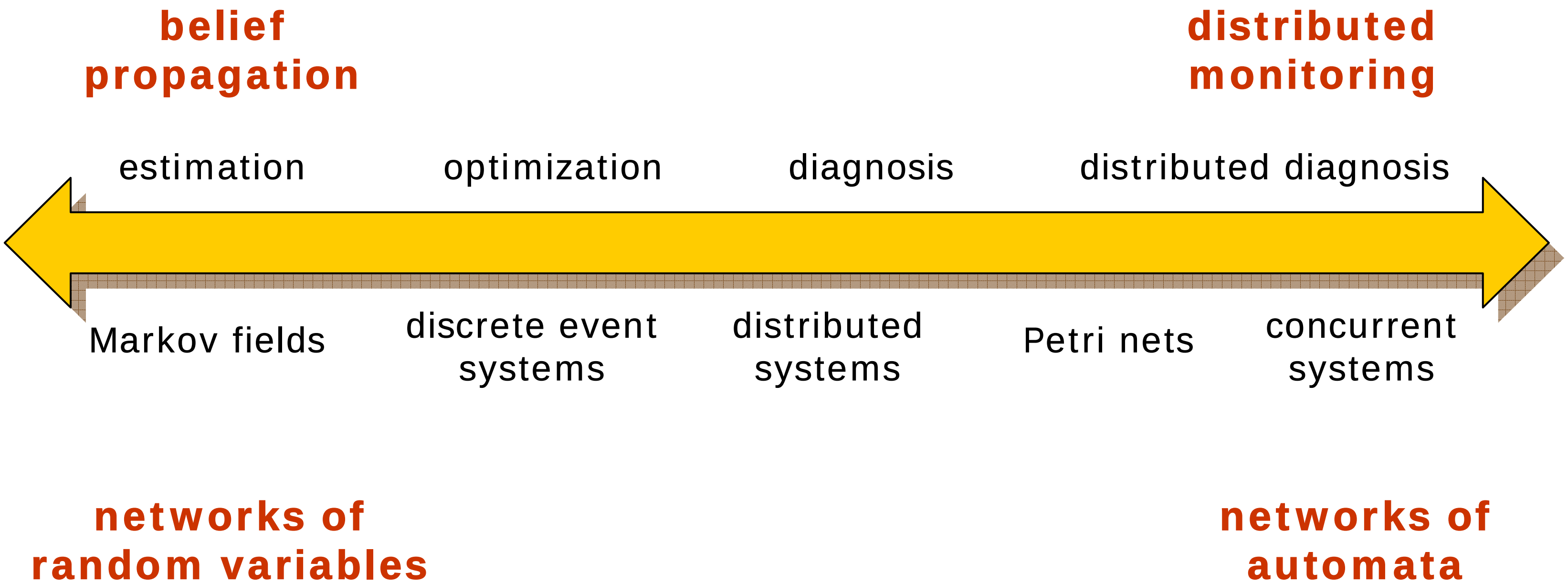


networks of random variables

networks of automata

# of Dynamic Systems

Bayesian Networks



# of Dynamic Systems

Bayesian Networks

belief propagation

distributed monitoring

estimation　optimization　diagnosis　distributed diagnosis

category theory + distributed programming

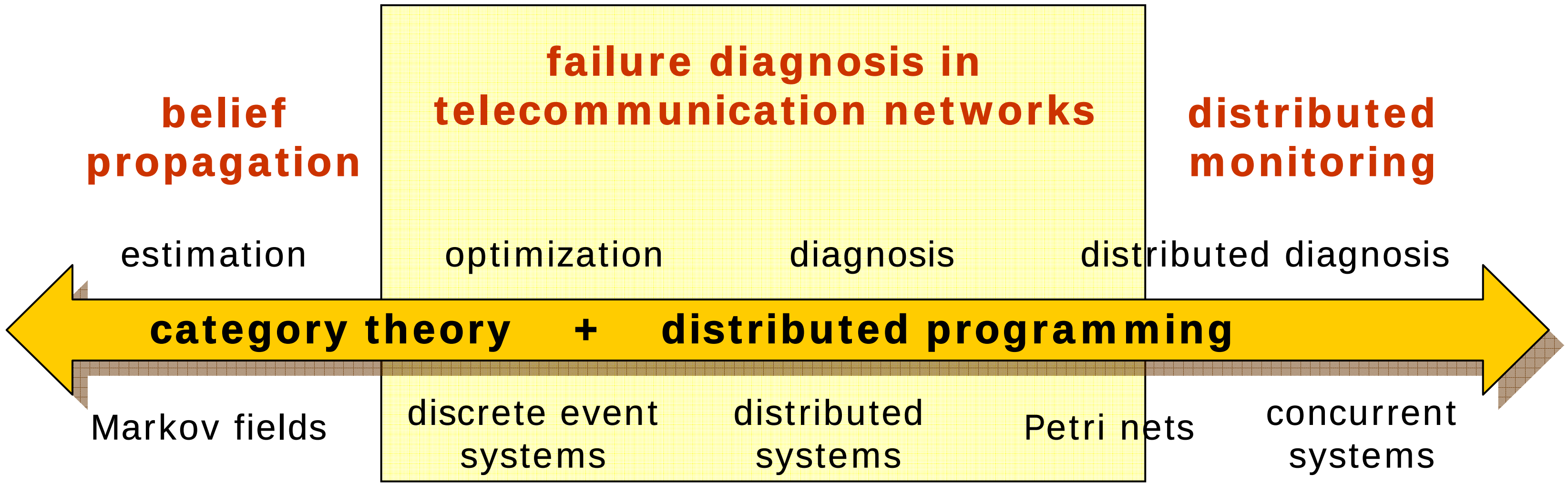
Markov fields　discrete event systems　distributed systems　Petri nets　concurrent systems

networks of random variables

networks of automata

# of Dynamic Systems

Bayesian Networks

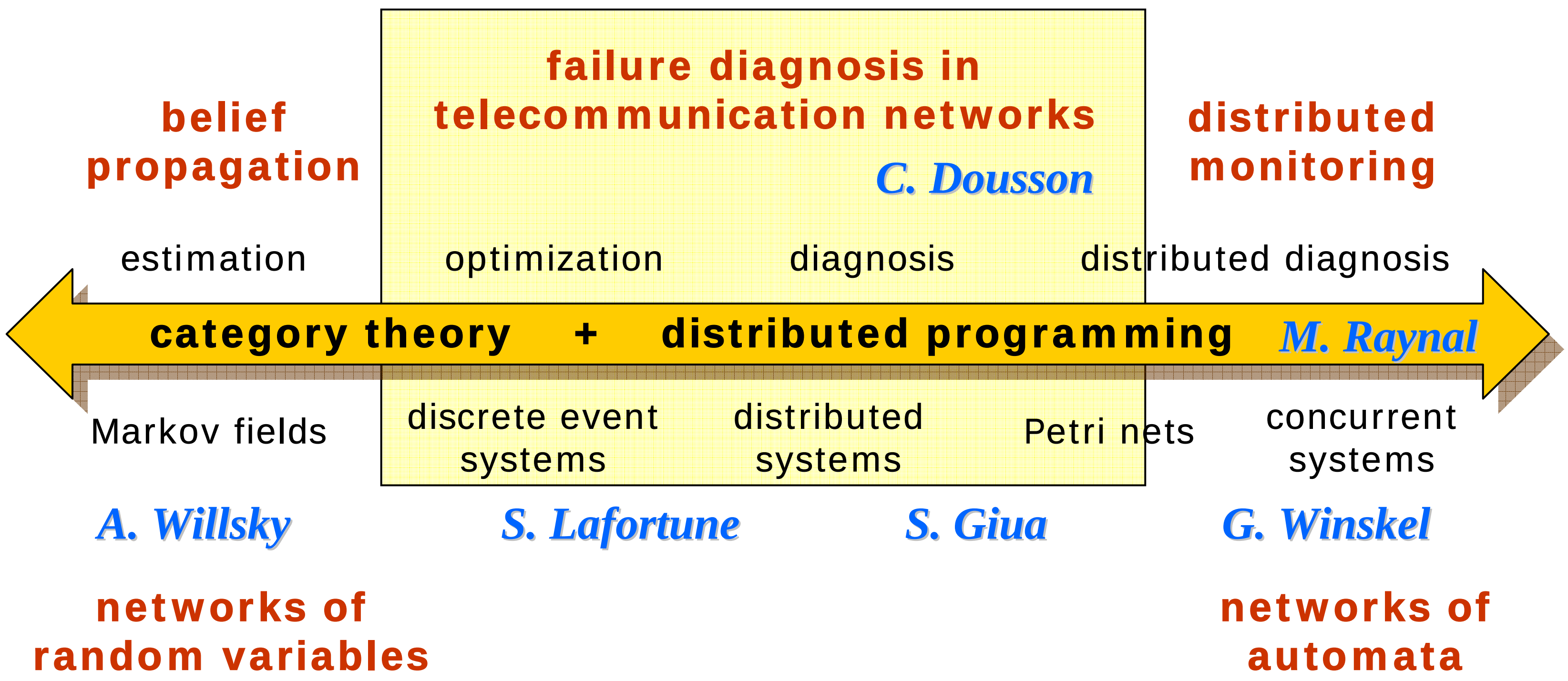


**networks of
random variables**

**networks of
automata**

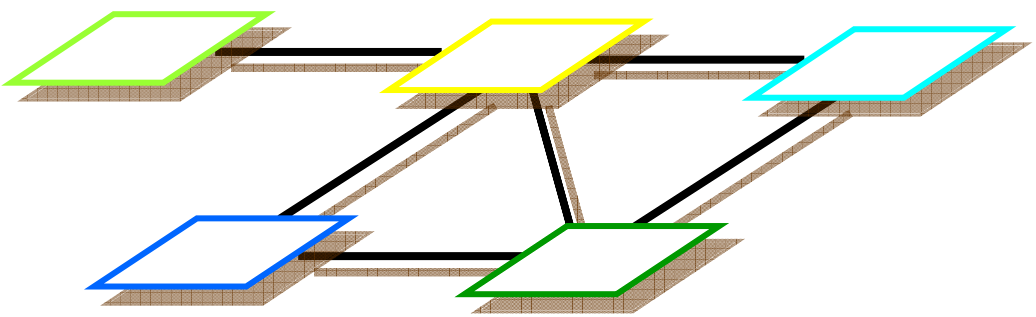
# of Dynamic Systems

Bayesian Networks

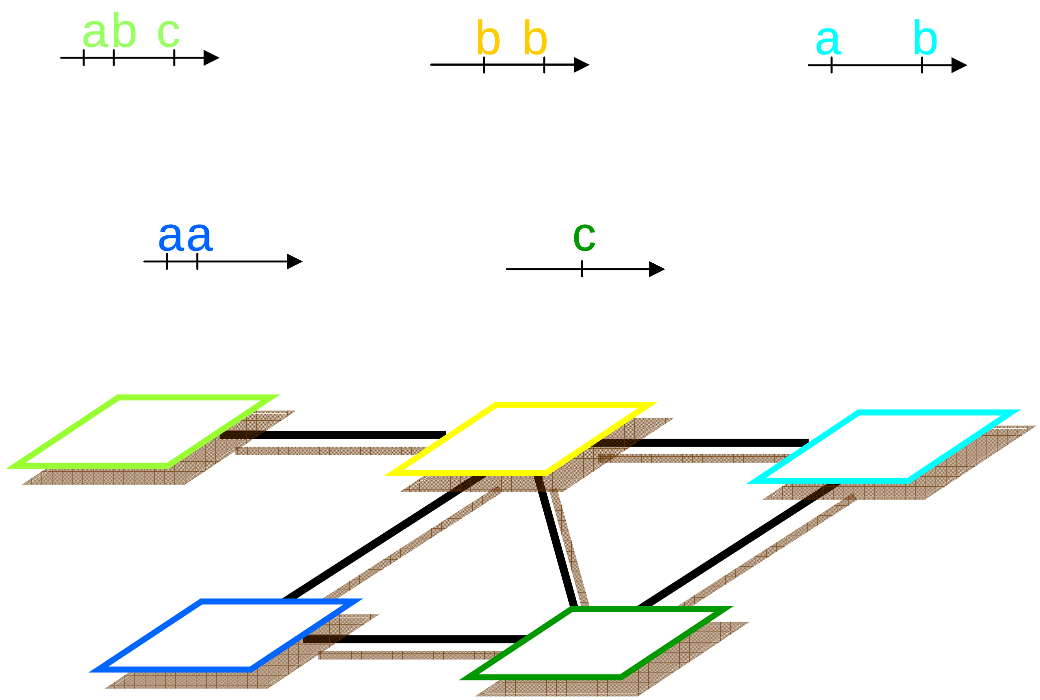


of Dynamic Systems

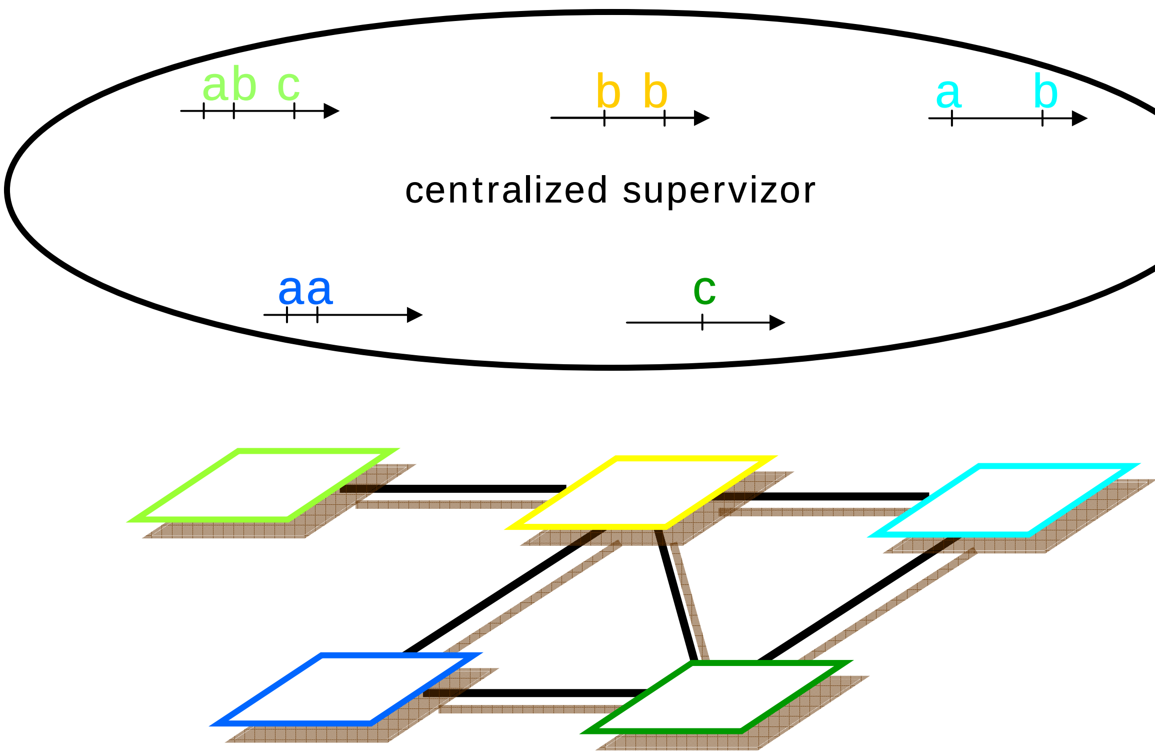
Distributed system monitoring…



Distributed system monitoring…

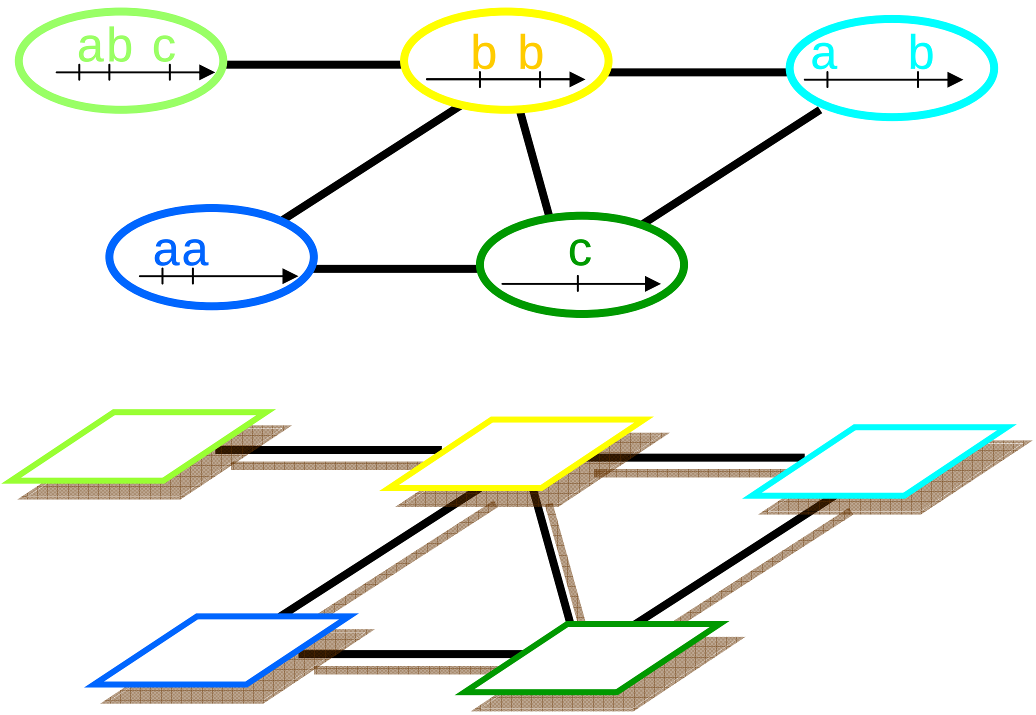


Distributed system monitoring…



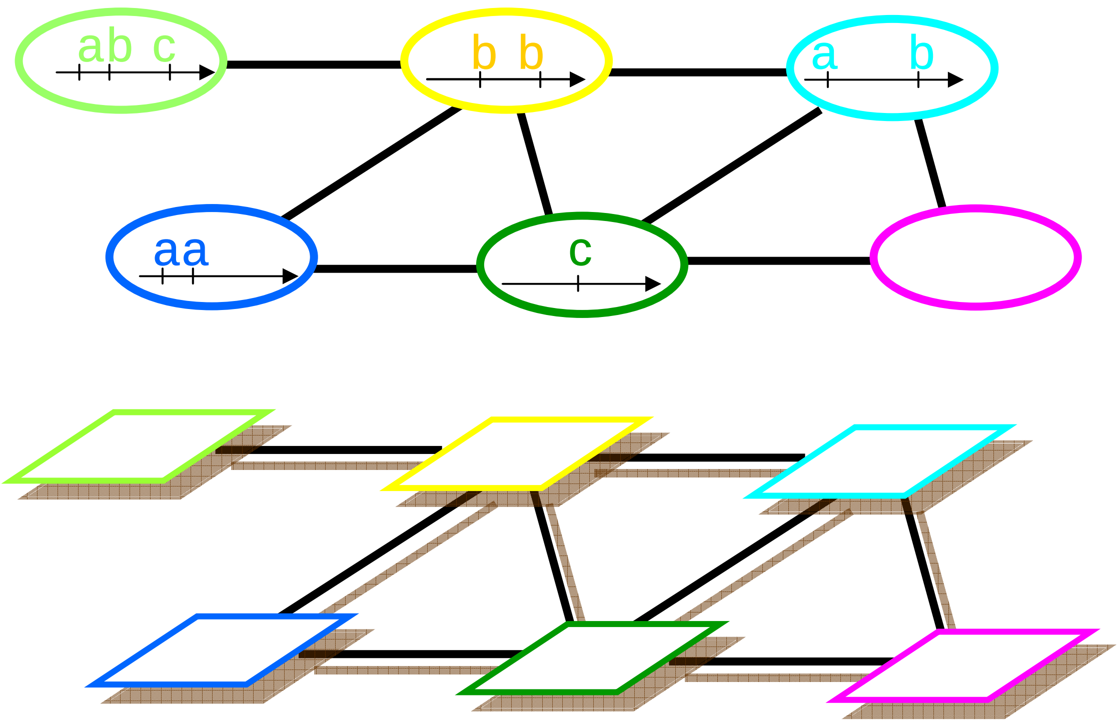
Distributed system monitoring…

distributed supervision



Distributed system monitoring…

distributed supervision



Outline

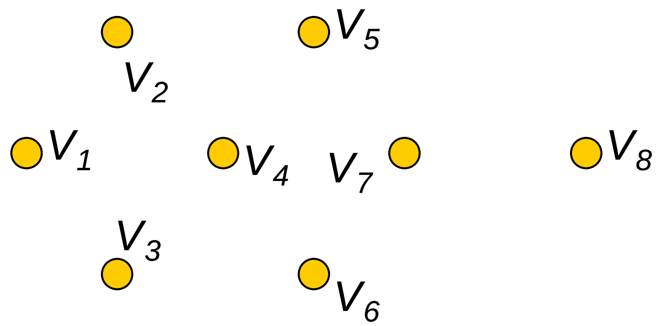
- Static distributed systems
  from Markov fields to abstract distributed systems
- Networks of automata
  introduction of the time dimension
- Networks of concurrent systems
  a partially ordered notion of time
- Applications
  distributed diagnosis in telecommunication networks
- Perspectives

Outline

- Static distributed systems
  from Markov fields to abstract distributed systems
- Networks of automata
  introduction of the time dimension
- Networks of concurrent systems
  a partially ordered notion of time
- Applications
  distributed diagnosis in telecommunication networks
- Perspectives

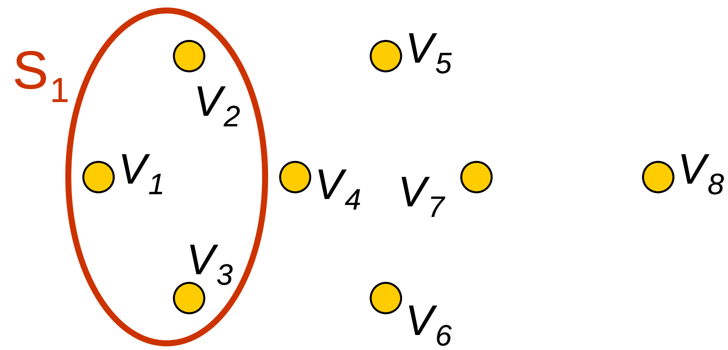
How to build a Markov field ?

- A collection of variables (sites) : $V_1,\ldots,V_n$



How to build a Markov field ?

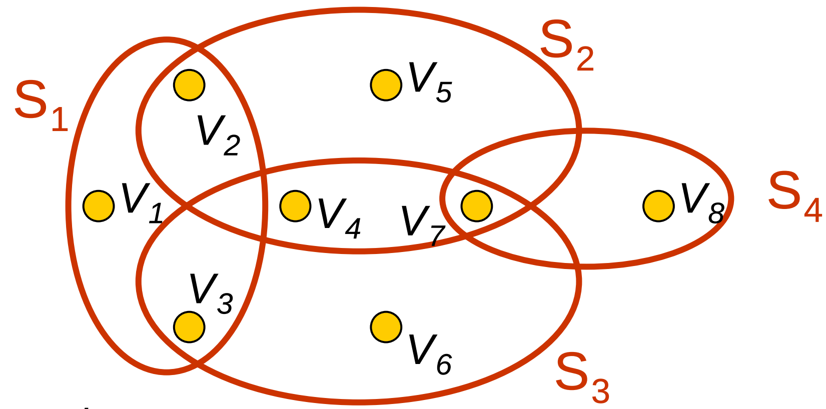
- A collection of variables (sites) : $V_1, \ldots, V_n$



- Component :
  - defines local interactions in a *clique* $S_i \subseteq \{V_1, \ldots, V_n\}$
  - by constraints : legal tuples $s_1 = (v_1, v_2, v_3)$
  - and/or by "soft" constraints : $\phi(s_1) = \phi(v_1, v_2, v_3)$

How to build a Markov field ?

- A collection of variables (sites) : $V_1, \ldots, V_n$



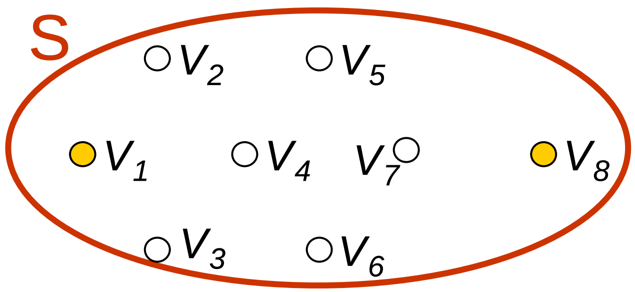
- Component :
  - defines local interactions in a *clique* $S_i \subseteq \{V_1, \ldots, V_n\}$
  - by constraints : legal tuples $s_1 = (v_1, v_2, v_3)$
  - and/or by "soft" constraints : $\phi(s_1) = \phi(v_1, v_2, v_3)$

$$P(s) \propto \Pi_i \, \phi(s_i)$$

- Composition :
  - by shared variables
  - conjunction of constraints, product of potentials

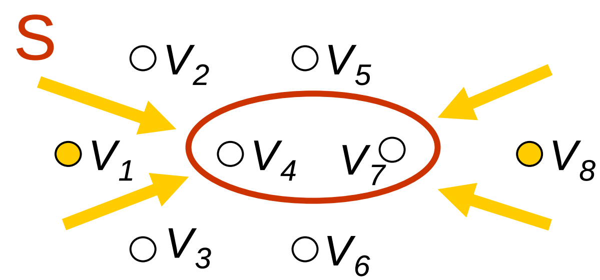
Inference : a reduction problem

- A typical problem : (Bayesian) inference
  - Some variables are known/fixed by constraints.
  - What is the most likely value / conditional law of the others ?



Inference : a reduction problem

- A typical problem : (Bayesian) inference
  - Some variables are known/fixed by constraints.
  - What is the most likely value / conditional law of the others ?

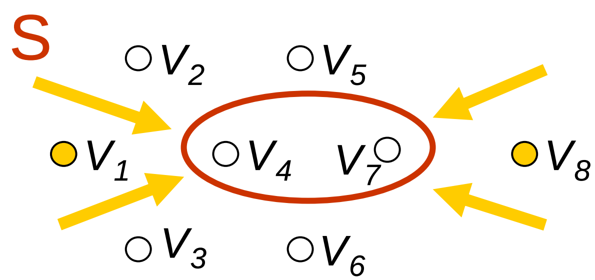


- Amounts to **reducing** the global “system” S
  - to variables of interest,
  - by maximizing / summing over all the rejected variables

$$P^*(v_i) \propto \max_{v_j,\ j \neq i} P(v)$$

Inference : a reduction problem

- A typical problem : (Bayesian) inference
  - Some variables are known/fixed by constraints.
  - What is the most likely value / conditional law of the others ?



- Amounts to **reducing** the global “system” S
  - to variables of interest,
  - by maximizing / summing over all the rejected variables

$$P^*(v_i) \propto \max_{v_j,\ j \neq i} P(v)$$

- There exist fast algorithms to do that jointly

  Kalman, Viterbi, MPM, BCJR, Sum-Product, Belief propagation, …

A more abstract viewpoint

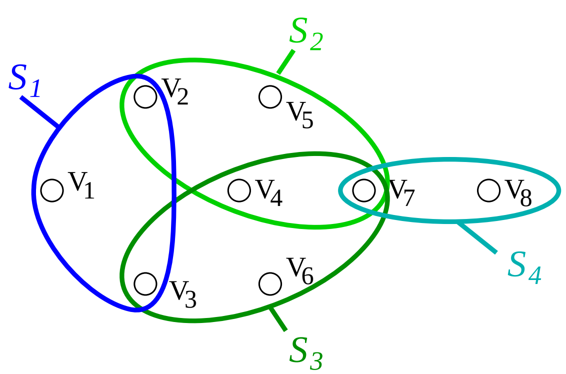
- Ingredients :
  - variables $V_{max} = \{V_1, V_2, \ldots\}$
  - "systems" or "components" S_i on these variables
  - a composition operator $S = S_1 \wedge S_2$
  - a reduction operator $\Pi_V(S)$ for $V \subseteq V_{max}$

A more abstract viewpoint

- Ingredients :
  - variables $V_{max} = \{V_1, V_2, \ldots\}$
  - "systems" or "components" S_i on these variables
  - a composition operator $S = S_1 \wedge S_2$
  - a reduction operator $\Pi_V(S)$ for $V \subseteq V_{max}$
- Axioms :
  - reductions are projections $\Pi_{V_1} \circ \Pi_{V_2} = \Pi_{V_1 \cap V_2}$
  - systems have a *local effect* $\exists\, V \subseteq V_{max},\ \Pi_V(S) = S$

A more abstract viewpoint

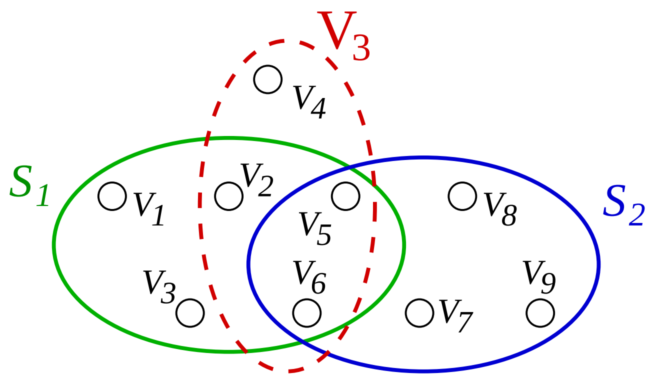
- Ingredients :
  - variables $V_{max} = \{V_1, V_2, \ldots\}$
  - "systems" or "components" S_i on these variables
  - a composition operator $S = S_1 \wedge S_2$
  - a reduction operator $\Pi_V(S)$ for $V \subseteq V_{max}$
- <u>Axioms</u> :
  - reductions are projections $\Pi_{V_1} \circ \Pi_{V_2} = \Pi_{V_1 \cap V_2}$
  - systems have a *local effect* $\exists\, V \subseteq V_{max},\ \Pi_V(S) = S$
- Graph of a composite system : $S = S_1 \wedge \ldots \wedge S_n$



Central axiom

- S_1 operates on V_1, S_2 operates on V_2

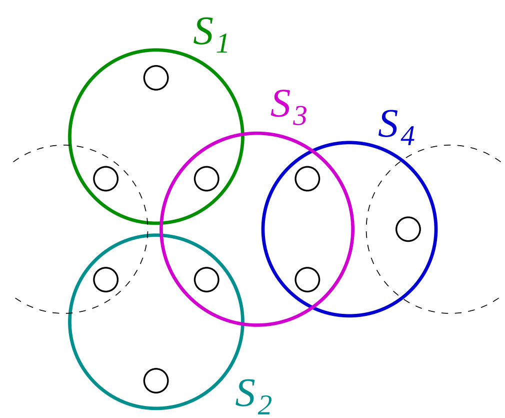
  let $V_3 \supseteq V_1 \cap V_2$ then $\Pi_{V_3}(S_1 \wedge S_2) = \Pi_{V_3}(S_1) \wedge \Pi_{V_3}(S_2)$



- A form of *conditional independence* :
  no interaction of S_1 and S_2 outside their shared variables.
- The key to fast estimation (reduction) algorithms for Bayesian networks.

Modular/distributed reduction algorithms

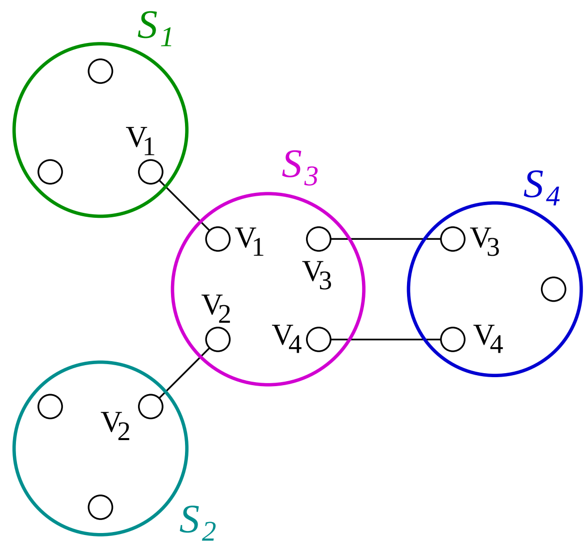
- Problem :
  - Given $S = S_1 \wedge \ldots \wedge S_n$ where S_i operates on V_i
  - compute the reduced components $S'_i = \Pi_{V_i}(S)$
  - i.e. *how does S_i change once inserted into the global S ?*
- This can be solved by *Message Passing Algorithms* (MPA)



  - only involves local computations
  - exact if the graph of S is a (hyper-) tree
  - can be seen as an *asynchronous distributed algorithm*

Modular/distributed reduction algorithms

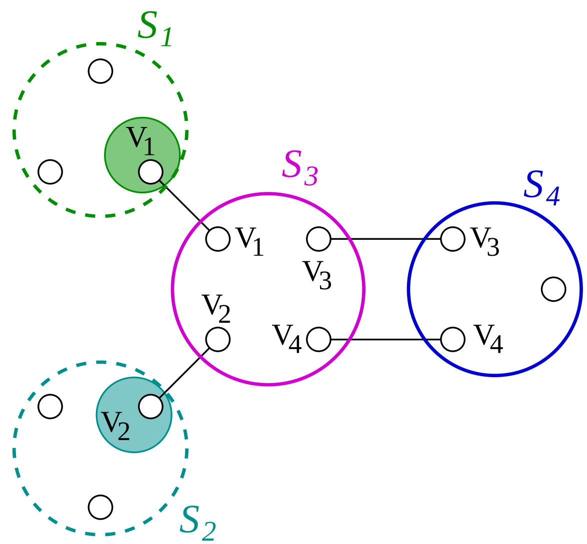
- Problem :
  - Given $S = S_1 \wedge \ldots \wedge S_n$ where S_i operates on V_i
  - compute the reduced components $S'_i = \Pi_{V_i}(S)$
  - i.e. *how does S_i change once inserted into the global S ?*
- This can be solved by *Message Passing Algorithms* (MPA)



  - only involves local computations
  - exact if the graph of S is a (hyper-) tree
  - can be seen as an *asynchronous distributed algorithm*

Modular/distributed reduction algorithms

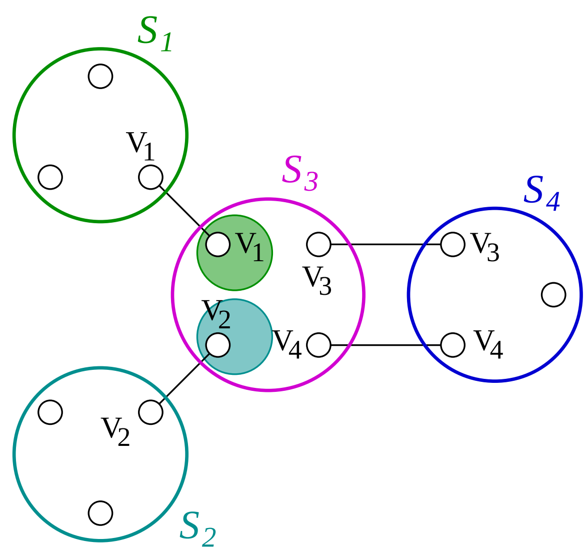
- Problem :
  - Given $S = S_1 \wedge \ldots \wedge S_n$ where S_i operates on V_i
  - compute the reduced components $S'_i = \Pi_{V_i}(S)$
  - i.e. *how does S_i change once inserted into the global S ?*
- This can be solved by *Message Passing Algorithms* (MPA)



  - only involves local computations
  - exact if the graph of S is a (hyper-) tree
  - can be seen as an *asynchronous distributed algorithm*

Modular/distributed reduction algorithms

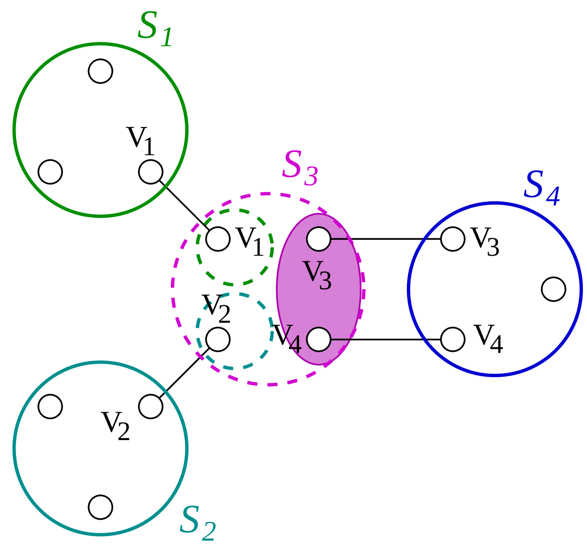
- Problem :
  - Given $S = S_1 \wedge \ldots \wedge S_n$ where S_i operates on V_i
  - compute the reduced components $S'_i = \Pi_{V_i}(S)$
  - i.e. *how does S_i change once inserted into the global S ?*
- This can be solved by *Message Passing Algorithms* (MPA)



  - only involves local computations
  - exact if the graph of S is a (hyper-) tree
  - can be seen as an *asynchronous distributed algorithm*

Modular/distributed reduction algorithms

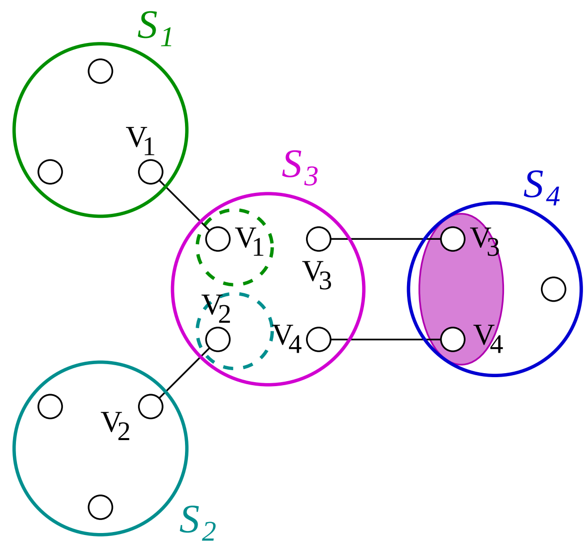
- Problem :
  - Given $S = S_1 \wedge \ldots \wedge S_n$ where S_i operates on V_i
  - compute the reduced components $S'_i = \Pi_{V_i}(S)$
  - i.e. *how does S_i change once inserted into the global S ?*
- This can be solved by *Message Passing Algorithms* (MPA)



  - only involves local computations
  - exact if the graph of S is a (hyper-) tree
  - can be seen as an *asynchronous distributed algorithm*

Modular/distributed reduction algorithms

- Problem :
  - Given $S = S_1 \wedge \ldots \wedge S_n$ where S_i operates on V_i
  - compute the reduced components $S'_i = \Pi_{V_i}(S)$
  - i.e. *how does S_i change once inserted into the global S ?*
- This can be solved by *Message Passing Algorithms* (MPA)

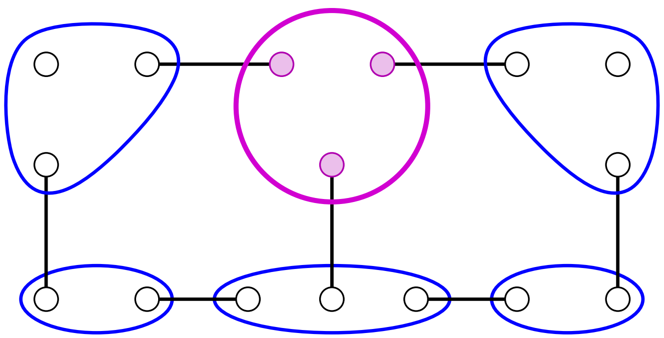


  - only involves local computations
  - exact if the graph of S is a (hyper-) tree
  - can be seen as an *asynchronous distributed algorithm*

What about systems with loops ?

- MPA can still be applied…
  - but they are sub-optimal.
  - They correspond to *turbo-algorithms :*
    good convergence properties in practice

- How good are their results ?
  - Local extendibility to a tree around each component.

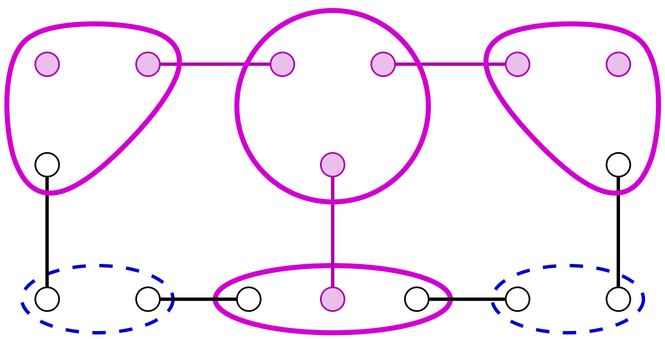


  - Local optimality of the max likelyhood estimates (Weiss ’01)

What about systems with loops ?

- MPA can still be applied…
  - but they are sub-optimal.
  - They correspond to *turbo-algorithms :*
    good convergence properties in practice

- How good are their results ?
  - Local extendibility to a tree around each component.



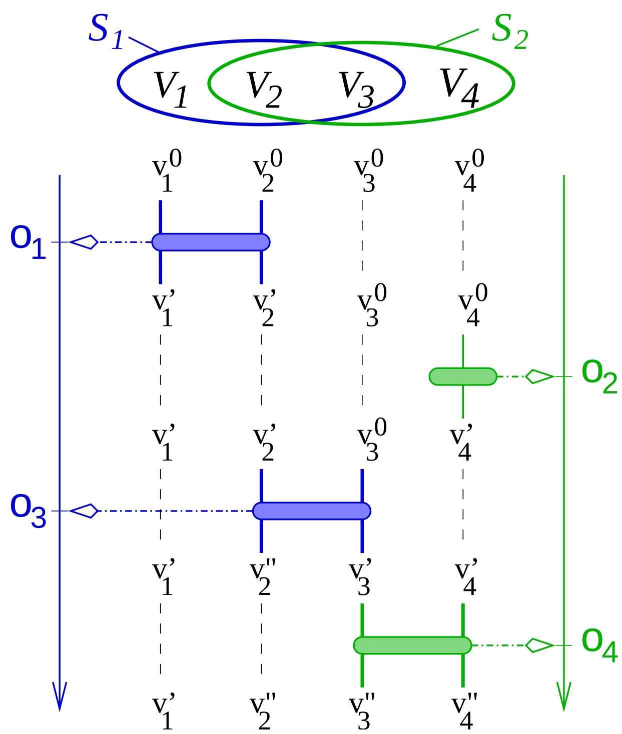
  - Local optimality of the max likelyhood estimates (Weiss ’01)

Outline

- Static distributed systems
  from Markov fields to abstract distributed systems
- Networks of automata
  introduction of the time dimension
- Networks of concurrent systems
  a partially ordered notion of time
- Applications
  distributed diagnosis in telecommunication networks
- Perspectives

Components as dynamic systems

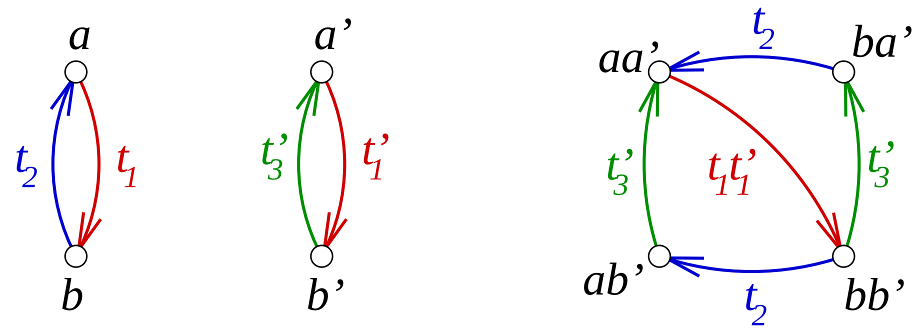
- <u>Objective</u> : change "cliques" into dynamic systems
  - allow components to change the value of their variables



  - This is hard to do with "3-D" Markov models (interactions in space + time) : so we take another path

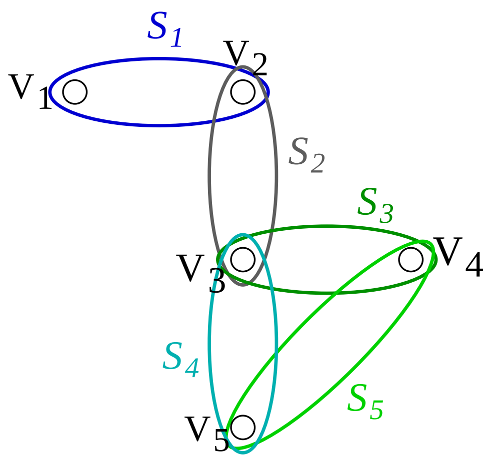
Components

- Variables are (labeled) automata
  - $V = (S, T, s^0, \rightarrow, \lambda, \Lambda)$
  - labeling on transitions $\lambda : T \rightarrow \Lambda$
- Interactions : defined by parallel product $V_1 \times V_2$
  - product of state sets $S_1 \times S_2$
  - transitions with identical labels are synchronized
  - transitions with private labels remain private



Components (2)

- Interaction graph of a system $S = V_1 \times \ldots \times V_n$
  - shared labels define the local interactions...



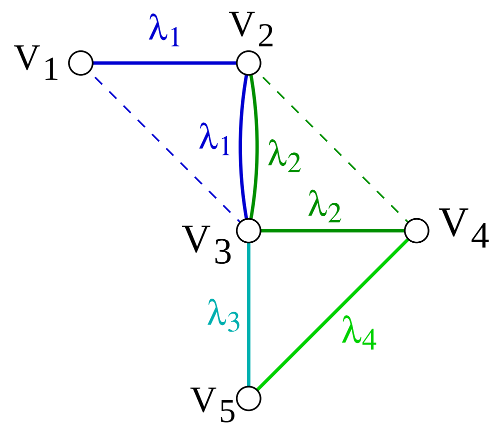
  - ... but they are not expressed by shared variables.

- Synchronization by *pullback*
  - takes into account the presence of common variables

$$V_1 \times V_2 \times V_3 = (V_1 \times V_2) \wedge (V_2 \times V_3)$$
$$= \quad S_1 \quad \wedge \quad S_2$$

Components (2)

- Interaction graph of a system $S = V_1 \times \ldots \times V_n$
  - shared labels define the local interactions...



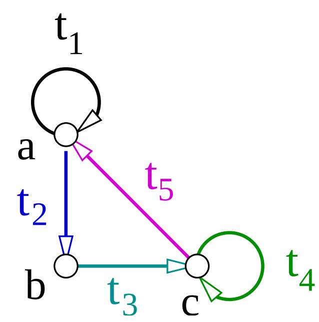
  - ... but they are not expressed by shared variables.

- Synchronization by *pullback*
  - takes into account the presence of common variables

$$V_1 \times V_2 \times V_3 = (V_1 \times V_2) \wedge (V_2 \times V_3)$$
$$= \quad S_1 \quad \wedge \quad S_2$$

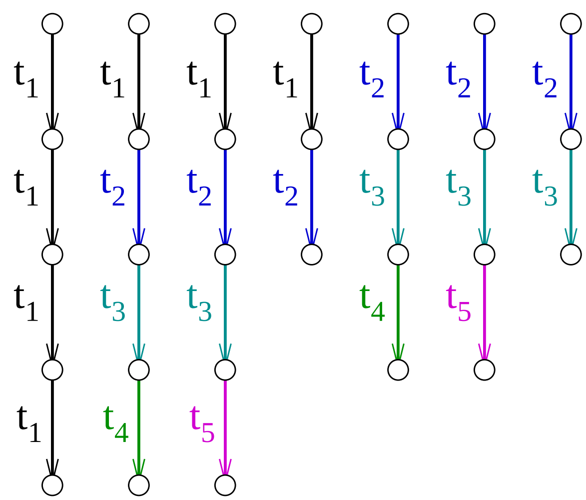
Trajectory sets

- Runs of $S = V_1 \times \ldots \times V_n$ are sequences of events



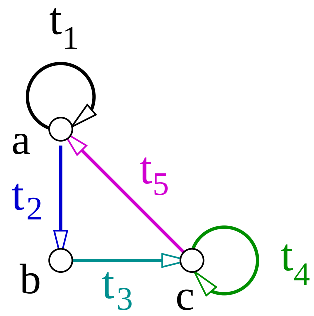
- Different encodings for trajectory sets :

(sub-) language



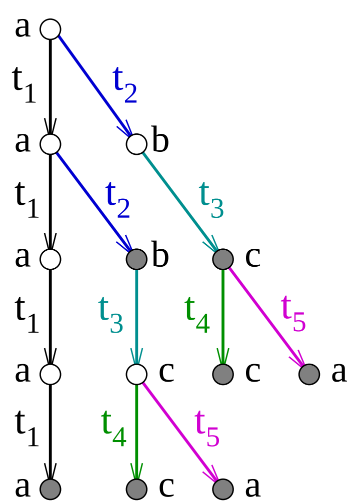
Trajectory sets

- Runs of $S = V_1 \times \ldots \times V_n$ are sequences of events



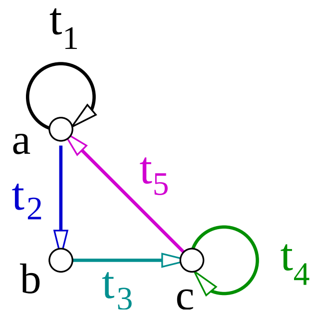
- Different encodings for trajectory sets :

branching process
(unfolding)



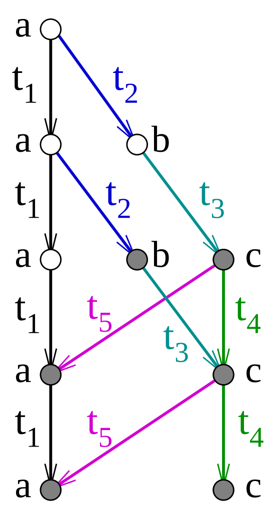
Trajectory sets

- Runs of $S = V_1 \times \ldots \times V_n$ are sequences of events



- Different encodings for trajectory sets :

trellis process
(time-unfolding)



time must be counted independently in each V_i for $S = V_1 \times \ldots \times V_n$

Where category theory helps…

- Moving to trajectory systems
  - replace each component S by its trellis $\mathcal{T}(S)$
  - ***Thm*** *: this functor has a left adjoint, which entails*

$$S = V_1 \times \ldots \times V_n \quad \Rightarrow \quad \mathcal{T}(S) = \mathcal{T}(V_1) \times^{\mathbf{T}} \ldots \times^{\mathbf{T}} \mathcal{T}(V_n)$$

$$S = S_1 \wedge \ldots \wedge S_m \quad \Rightarrow \quad \mathcal{T}(S) = \mathcal{T}(S_1) \wedge^{\mathbf{T}} \ldots \wedge^{\mathbf{T}} \mathcal{T}(S_m)$$

Where category theory helps…

- Moving to trajectory systems
  - replace each component S by its trellis $\mathcal{T}(S)$
  - ***Thm*** *: this functor has a left adjoint, which entails*

$$S = V_1 \times \ldots \times V_n \quad \Rightarrow \quad \mathcal{T}(S) = \mathcal{T}(V_1) \times^{\mathbf{T}} \ldots \times^{\mathbf{T}} \mathcal{T}(V_n)$$

$$S = S_1 \wedge \ldots \wedge S_m \quad \Rightarrow \quad \mathcal{T}(S) = \mathcal{T}(S_1) \wedge^{\mathbf{T}} \ldots \wedge^{\mathbf{T}} \mathcal{T}(S_m)$$

- Interest:
  - we are back to *static systems*, in *factorized form*,
  - we get procedures to compute products/pullbacks of trellis processes,
  - products/pullbacks automatically come with a natural notion of *projection* !

Where category theory helps…

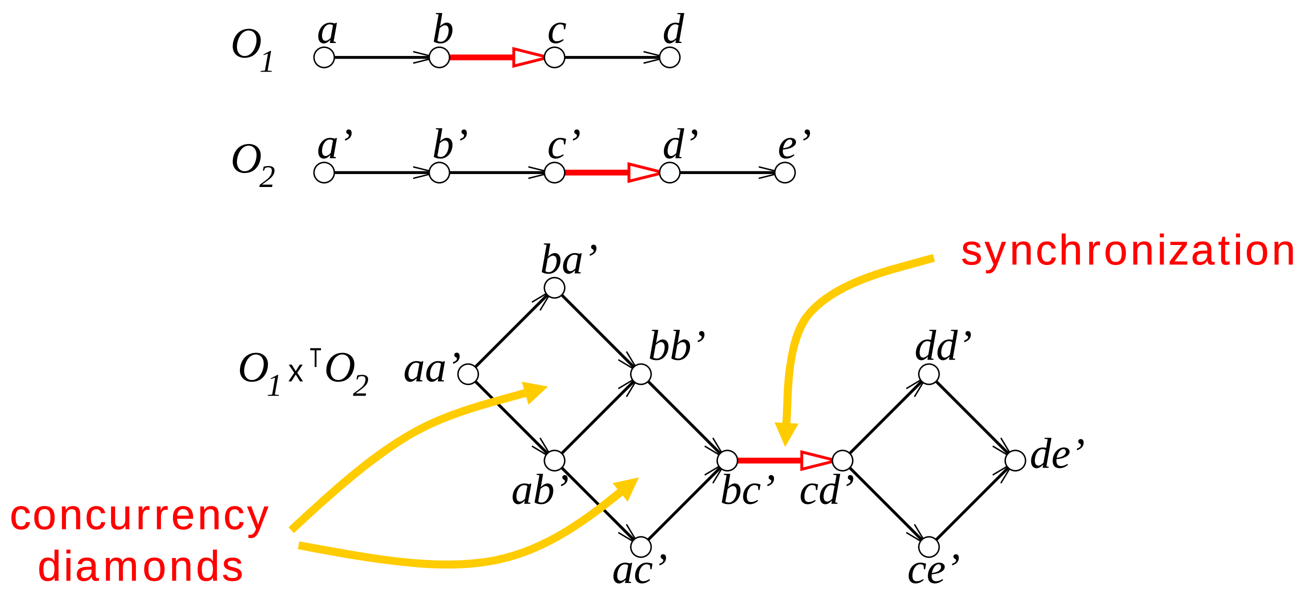
- Moving to trajectory systems
  - replace each component S by its trellis $\mathcal{T}$(S)
  - ***Thm*** *: this functor has a left adjoint, which entails*

$$S = V_1 \times \ldots \times V_n \quad \Rightarrow \quad \mathcal{T}(S) = \mathcal{T}(V_1) \times^{\mathbf{T}} \ldots \times^{\mathbf{T}} \mathcal{T}(V_n)$$

$$S = S_1 \wedge \ldots \wedge S_m \quad \Rightarrow \quad \mathcal{T}(S) = \mathcal{T}(S_1) \wedge^{\mathbf{T}} \ldots \wedge^{\mathbf{T}} \mathcal{T}(S_m)$$

- Interest:
  - we are back to *static systems*, in *factorized form*,
  - we get procedures to compute products/pullbacks of trellis processes,
  - products/pullbacks automatically come with a natural notion of *projection* !
  - ***Thm*** *: the central axiom holds on pullbacks and projections of trajectory sets.*

Example of a product



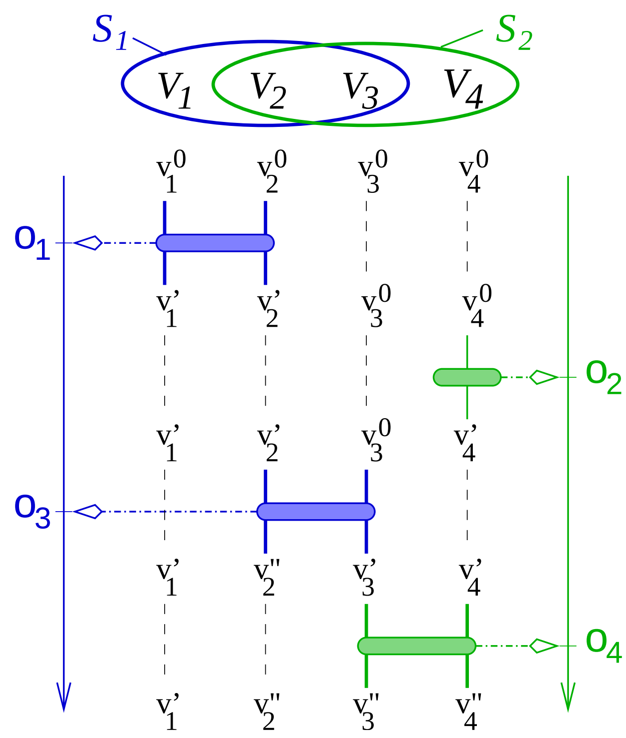
- Drawbacks:
  - computes all possible interleavings of runs
  - <u>concurrency is against us</u>...
  - Not suitable to distributed concurrent systems.

Outline

- Static distributed systems
  from Markov fields to abstract distributed systems
- Networks of automata
  introduction of the time dimension
- Networks of concurrent systems
  a partially ordered notion of time
- Applications
  distributed diagnosis in telecommunication networks
- Perspectives

Components as Petri nets

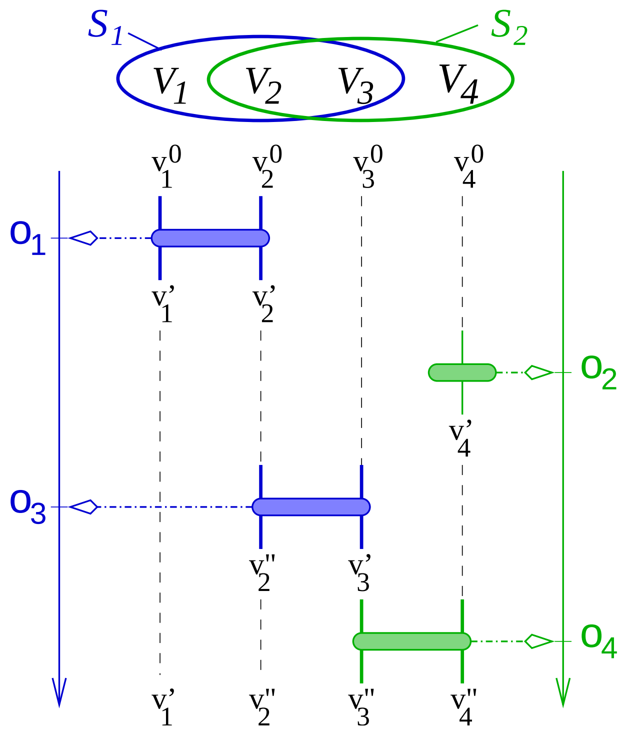
- Objective : represent the concurrency of events



  - replace the global clock by local clocks, one for each variable,
  - preserve only causality links between events:
  - *time is now partially ordered*

Components as Petri nets

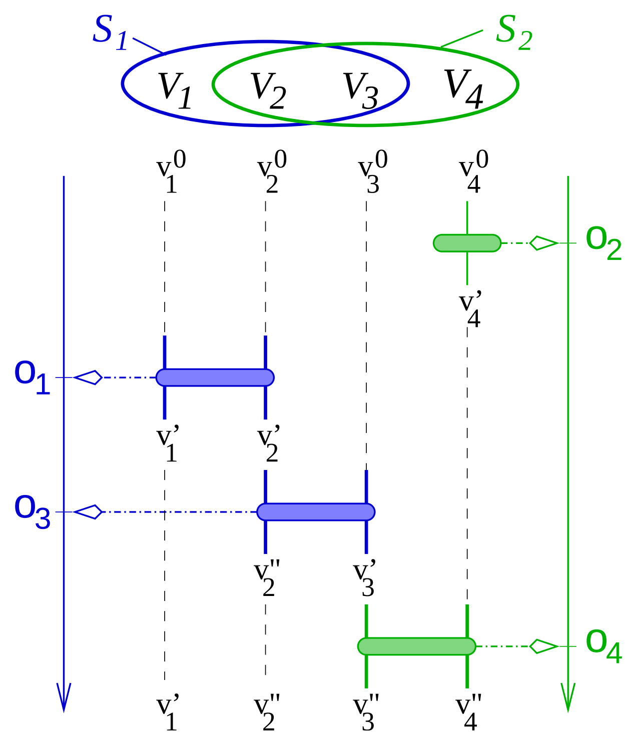
- <u>Objective</u> : represent the concurrency of events



  - replace the global clock by local clocks, one for each variable
  - preserve only causality links between events
  - *time is now partially ordered*

Components as Petri nets

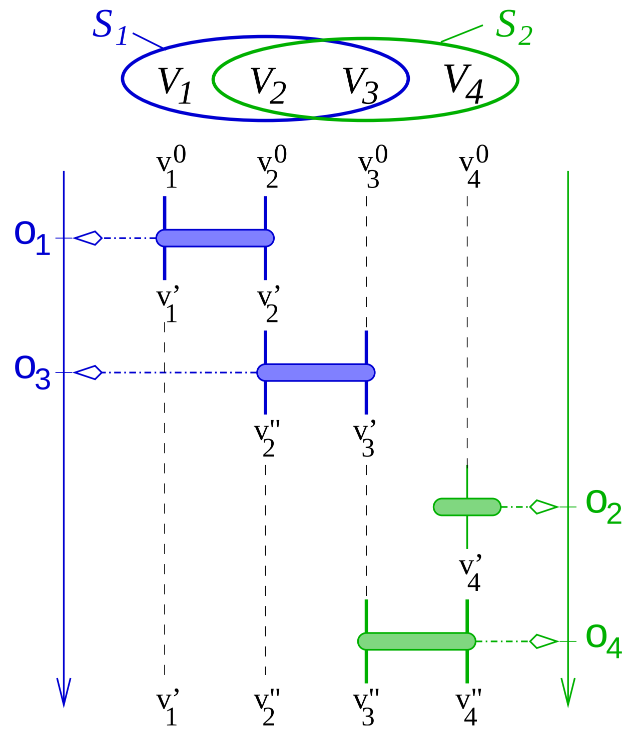
- Objective : represent the concurrency of events



  - replace the global clock by local clocks, one for each variable
  - preserve only causality links between events
  - *time is now partially ordered*

Components as Petri nets

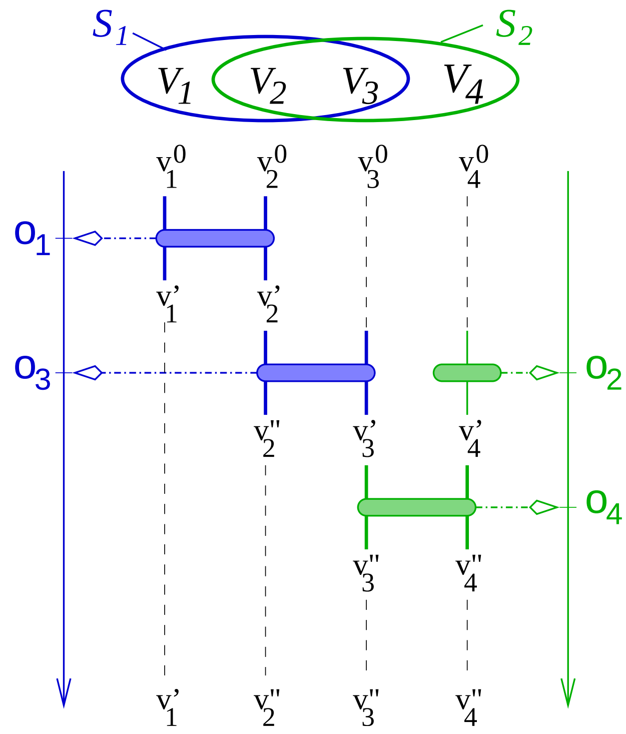
- <u>Objective</u> : represent the concurrency of events



  - replace the global clock by local clocks, one for each variable
  - preserve only causality links between events
  - *time is now partially ordered*

Components as Petri nets

- <u>Objective</u> : represent the concurrency of events

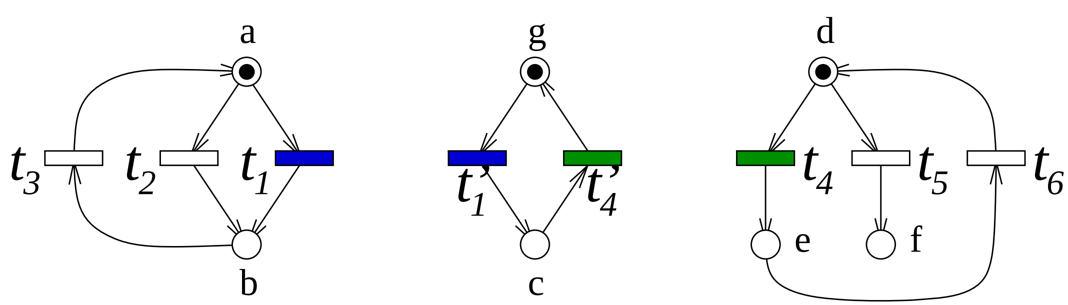


  - replace the global clock by local clocks, one for each variable
  - preserve only causality links between events
  - *time is now partially ordered*

Components as Petri nets (2)

- Composition of automata by “concurrent” product :
  - *disjoint union* of state sets (instead of product)
  - transitions with shared labels are “glued”
  - transitions with private labels don’t change

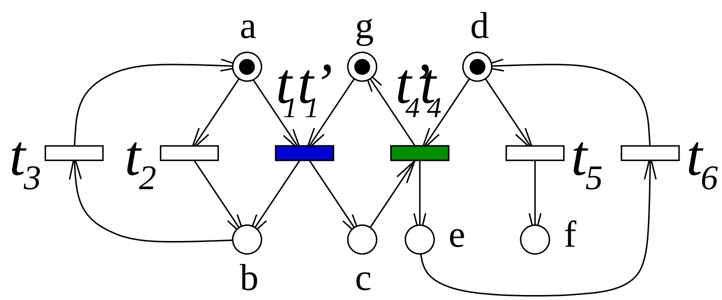
$$S = V_1 \times V_2 \times V_3$$



Components as Petri nets (2)

- Composition of automata by “concurrent” product :
  - *disjoint union* of state sets (instead of product)
  - transitions with shared labels are “glued”
  - transitions with private labels don’t change

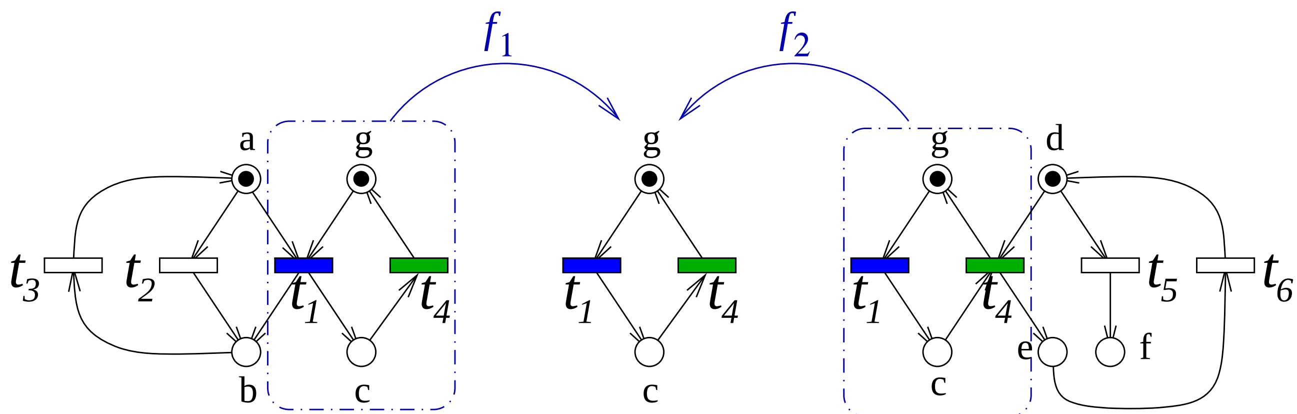
$$S = V_1 \times V_2 \times V_3$$



Components as Petri nets (3)

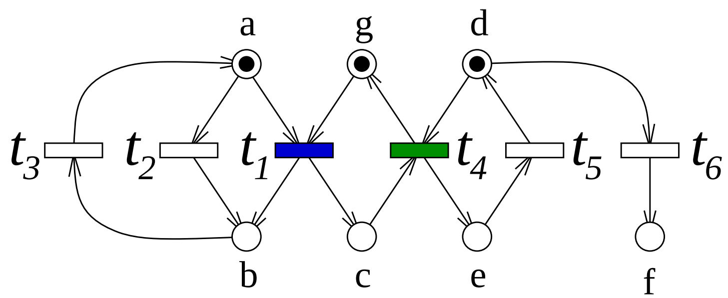
- Composition by "concurrent" pullback is also possible

$$S = V_1 \times V_2 \times V_3 = (V_1 \times V_2) \wedge (V_2 \times V_3)$$

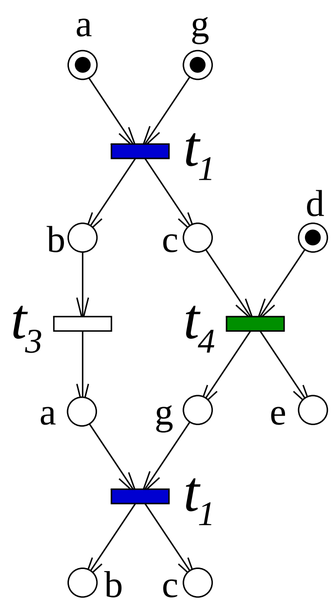


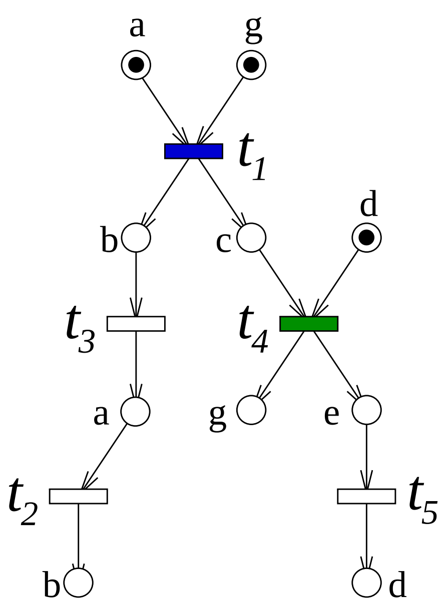
- Graph of a distributed system $S = V_1 \times \ldots \times V_n = S_1 \wedge \ldots \wedge S_m$
  - exactly as before...

Trajectory sets

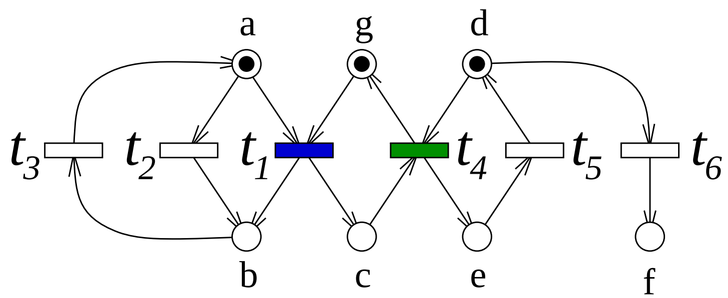


- Runs of $S = V_1 \times \ldots \times V_n$ are *partial orders* of events (also called *configurations*)



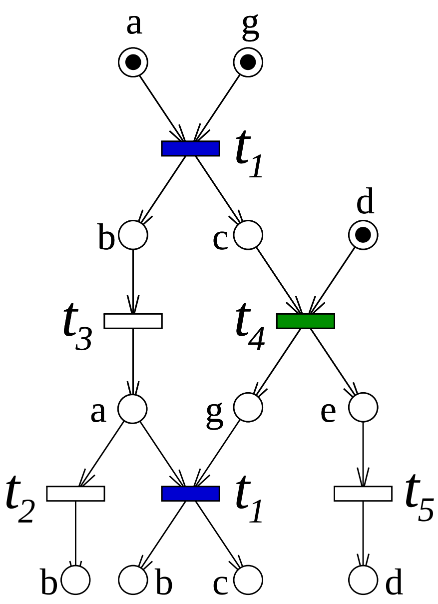


Trajectory sets

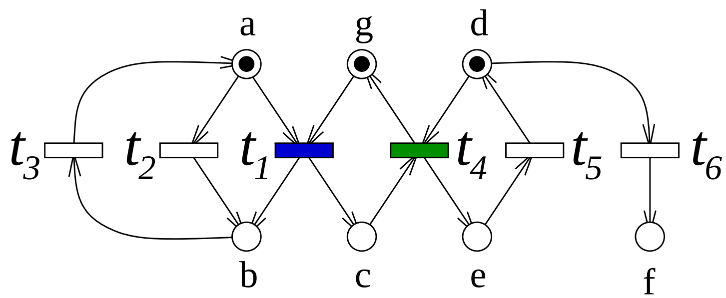


- Runs of $S = V_1 \times \ldots \times V_n$ are *partial orders* of events (also called *configurations*)

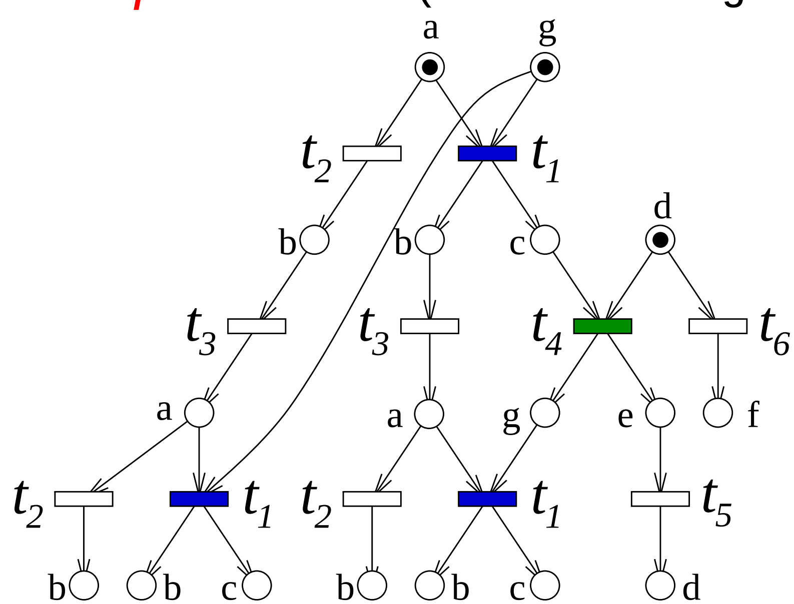
branching process (unfolding)



Trajectory sets (2)

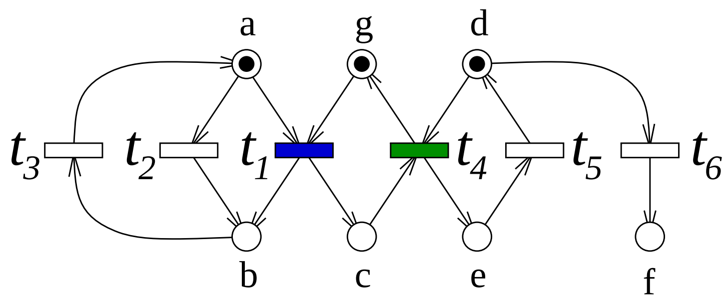


- *Branching processes* can be further compressed into *trellis processes* (cousin of Merged processes, Khomenko '05)

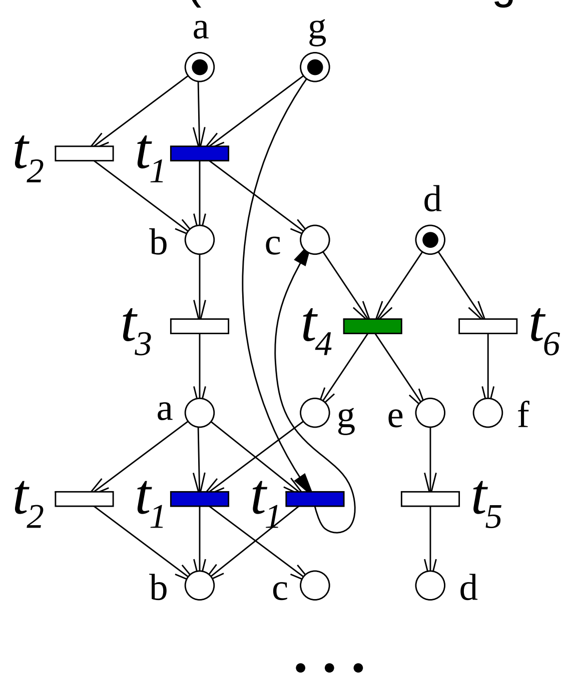


. . .

Trajectory sets (2)



- *Branching processes* can be further compressed into *trellis processes* (cousin of Merged processes, Khomenko ’05)



time is counted independently in each V_i for $S = V_1 \times \ldots \times V_n$

Category theory again…

- Moving to trajectory systems
  - replace each component S by its unfolding $\mathcal{U}(S)$ or by its time-unfolding $\mathcal{T}(S)$
  - ***Thm :*** *these functors both have a left adjoint*

    $S = V_1 \times \ldots \times V_n \quad \Rightarrow \quad \mathcal{T}(S) = \mathcal{T}(V_1) \times^{\mathbf{T}} \ldots \times^{\mathbf{T}} \mathcal{T}(V_n)$

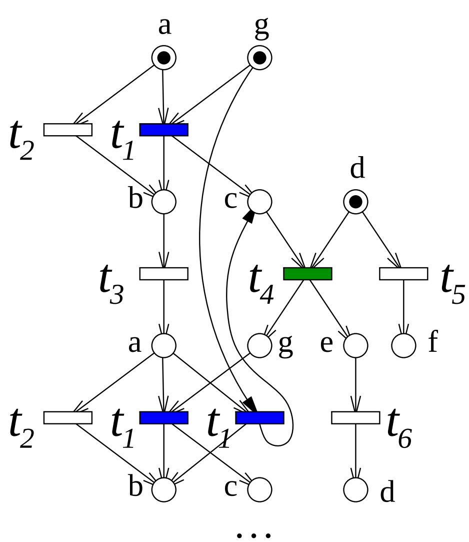
    $S = S_1 \wedge \ldots \wedge S_m \quad \Rightarrow \quad \mathcal{T}(S) = \mathcal{T}(S_1) \wedge^{\mathbf{T}} \ldots \wedge^{\mathbf{T}} \mathcal{T}(S_m)$

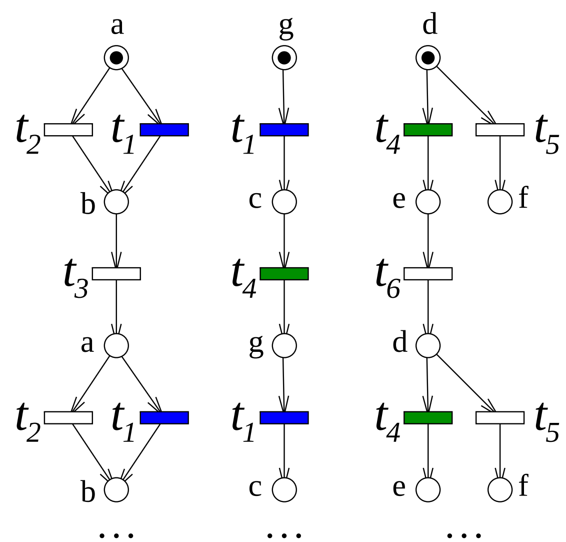
Category theory again…

- Moving to trajectory systems
  - replace each component S by its unfolding $\mathcal{U}(S)$ or by its time-unfolding $\mathcal{T}(S)$
  - ***Thm :*** *these functors both have a left adjoint*

    $S = V_1 \times \ldots \times V_n \quad \Rightarrow \quad \mathcal{T}(S) = \mathcal{T}(V_1) \times^{\mathbf{T}} \ldots \times^{\mathbf{T}} \mathcal{T}(V_n)$

    $S = S_1 \wedge \ldots \wedge S_m \quad \Rightarrow \quad \mathcal{T}(S) = \mathcal{T}(S_1) \wedge^{\mathbf{T}} \ldots \wedge^{\mathbf{T}} \mathcal{T}(S_m)$





Category theory again…

- Moving to trajectory systems
  - replace each component S by its unfolding $\mathcal{U}(S)$ or by its time-unfolding $\mathcal{T}(S)$
  - ***Thm*** *: these functors both have a left adjoint*

    $S = V_1 \times \ldots \times V_n \quad \Rightarrow \quad \mathcal{T}(S) = \mathcal{T}(V_1) \times^{\mathbf{T}} \ldots \times^{\mathbf{T}} \mathcal{T}(V_n)$

    $S = S_1 \wedge \ldots \wedge S_m \quad \Rightarrow \quad \mathcal{T}(S) = \mathcal{T}(S_1) \wedge^{\mathbf{T}} \ldots \wedge^{\mathbf{T}} \mathcal{T}(S_m)$

- Interest:
  - We are back to *static systems,*
  - we get *computation procedures* for products and pullbacks,
  - we get a natural notion of *projection,*

Category theory again…

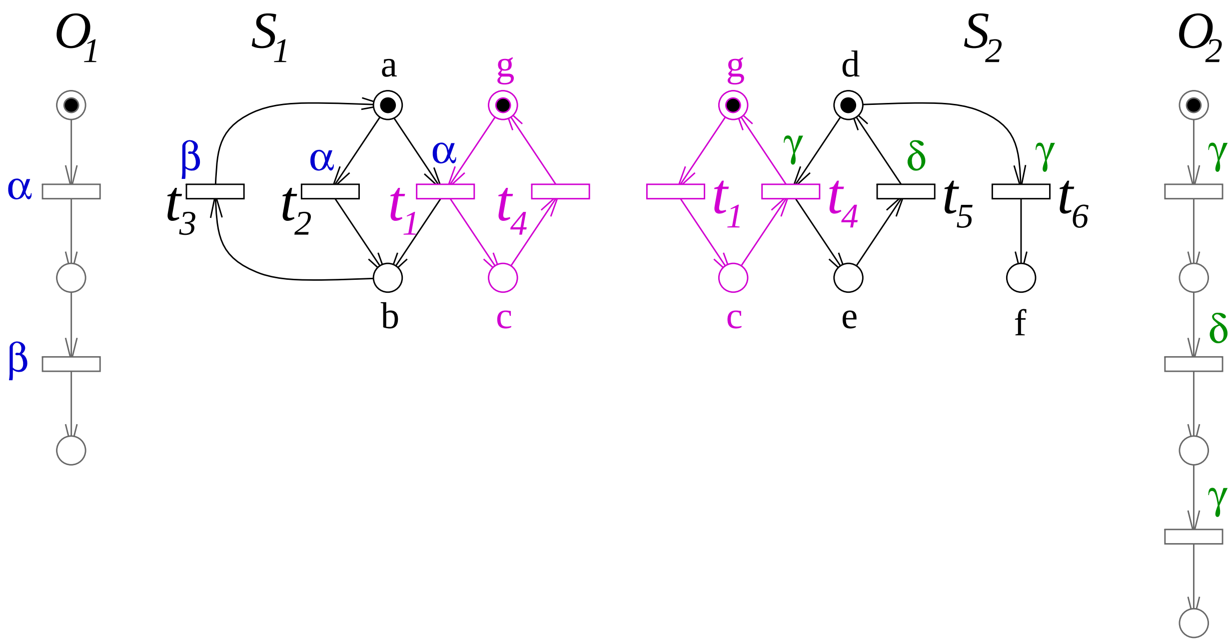
- Moving to trajectory systems
  - replace each component S by its unfolding $\mathcal{U}$(S) or by its time-unfolding $\mathcal{T}$(S)
  - ***Thm*** *: these functors both have a left adjoint*

    $S = V_1 \times \ldots \times V_n \quad \Rightarrow \quad \mathcal{T}(S) = \mathcal{T}(V_1) \times^{\mathbf{T}} \ldots \times^{\mathbf{T}} \mathcal{T}(V_n)$

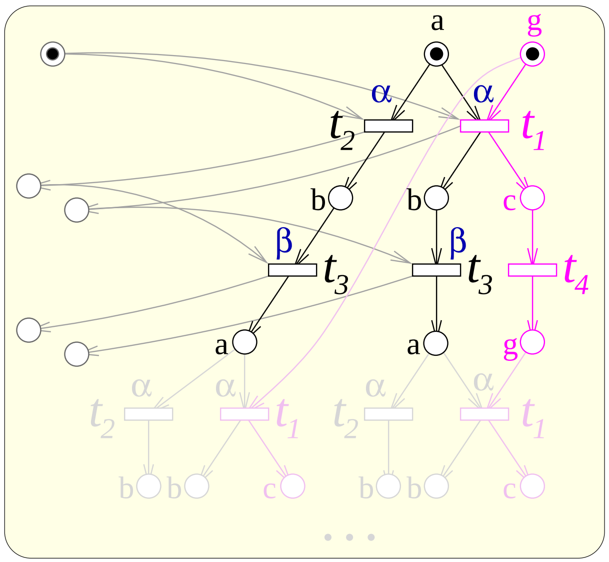
    $S = S_1 \wedge \ldots \wedge S_m \quad \Rightarrow \quad \mathcal{T}(S) = \mathcal{T}(S_1) \wedge^{\mathbf{T}} \ldots \wedge^{\mathbf{T}} \mathcal{T}(S_m)$

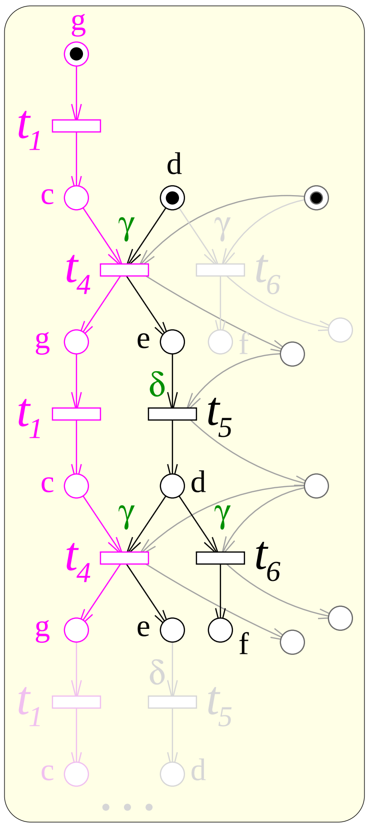
- Interest:
  - We are back to *static systems,*
  - we get *computation procedures* for products and pullbacks,
  - we get a natural notion of *projection,*
  - ***Thm*** *: the central axiom is valid, but only in limited cases.*

A computation example

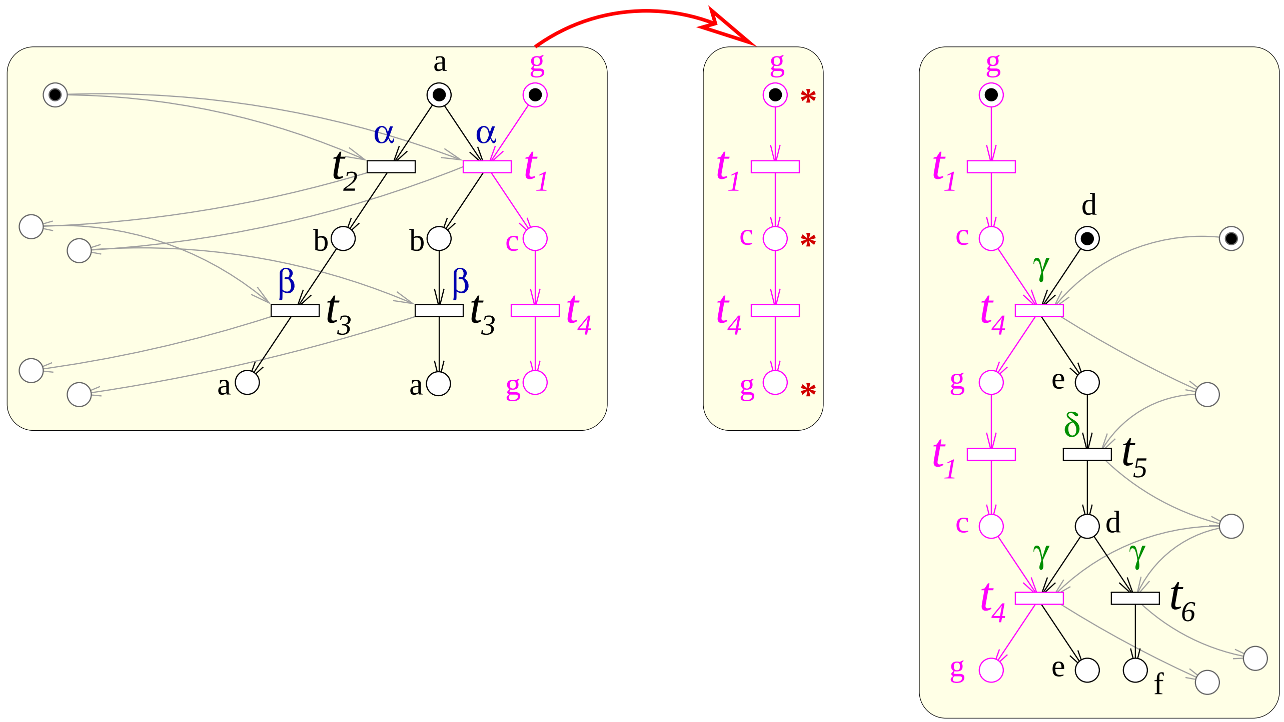


A computation example

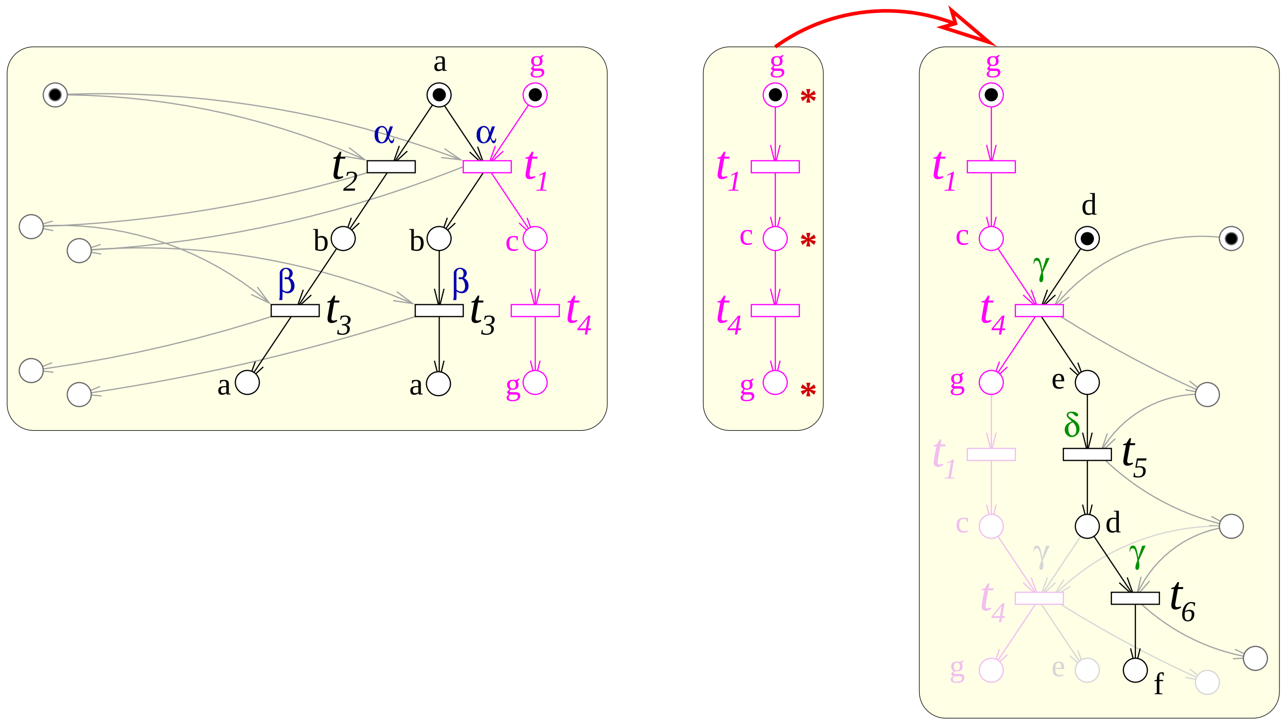




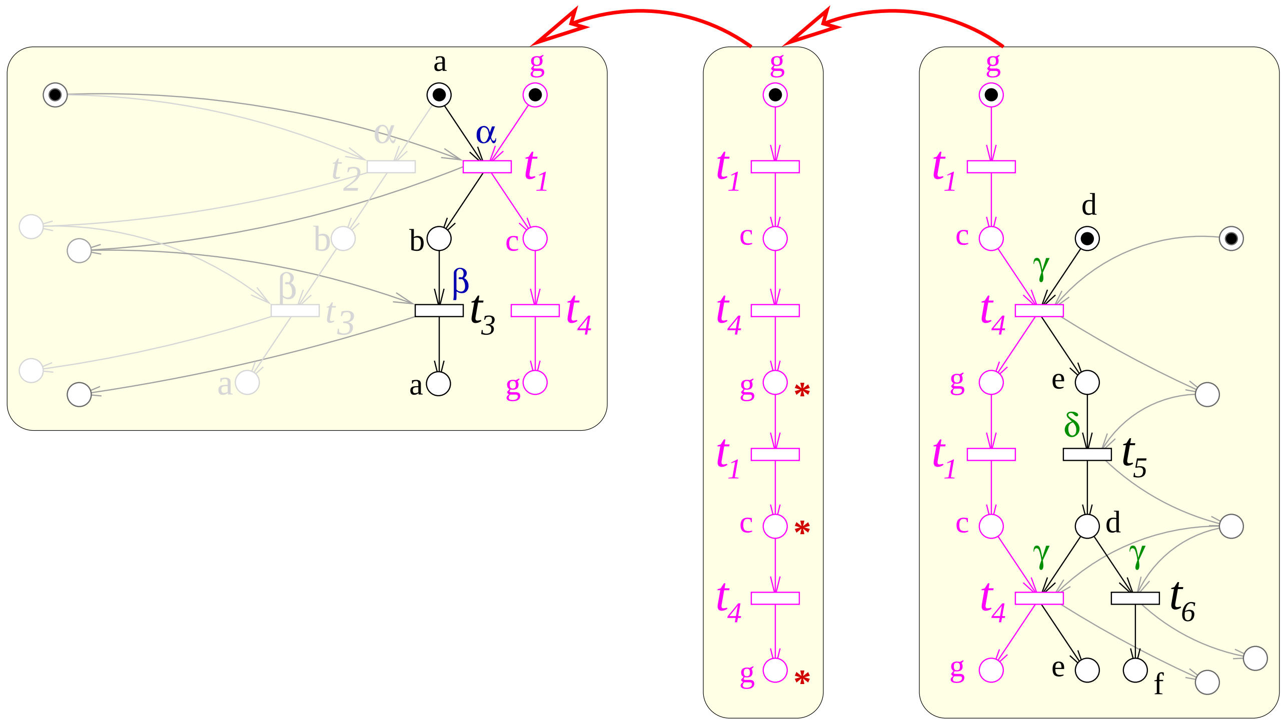
A computation example



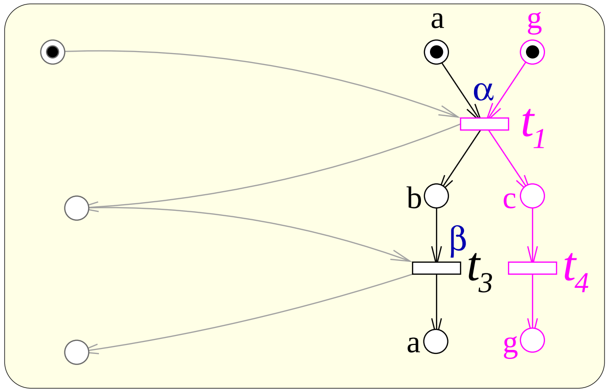
A computation example

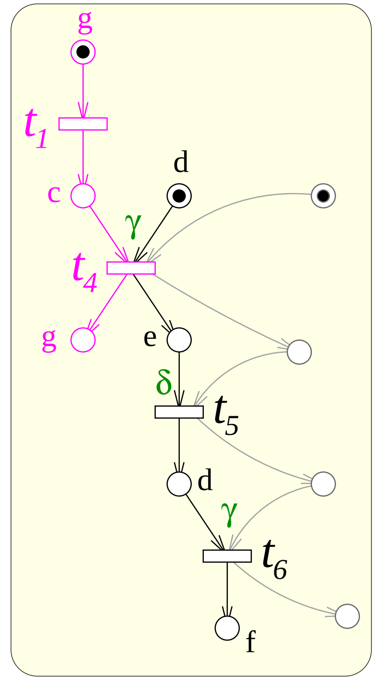


A computation example



A computation example



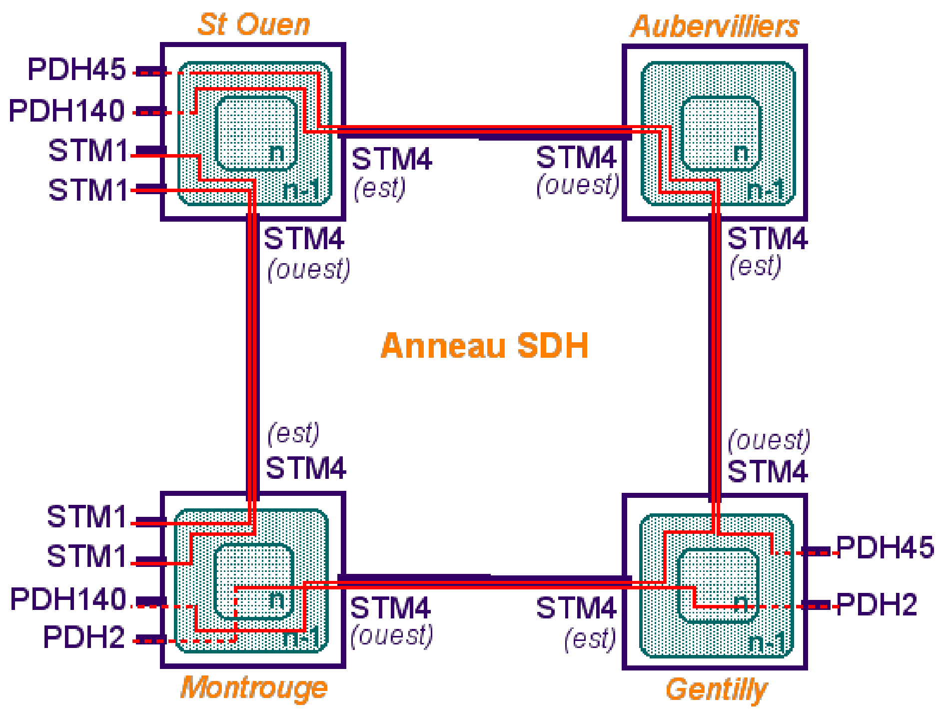


Outline

- Static distributed systems
  from Markov fields to abstract distributed systems
- Networks of automata
  introduction of the time dimension
- Networks of concurrent systems
  a partially ordered notion of time
- Applications
  distributed diagnosis in telecommunication networks
- Perspectives

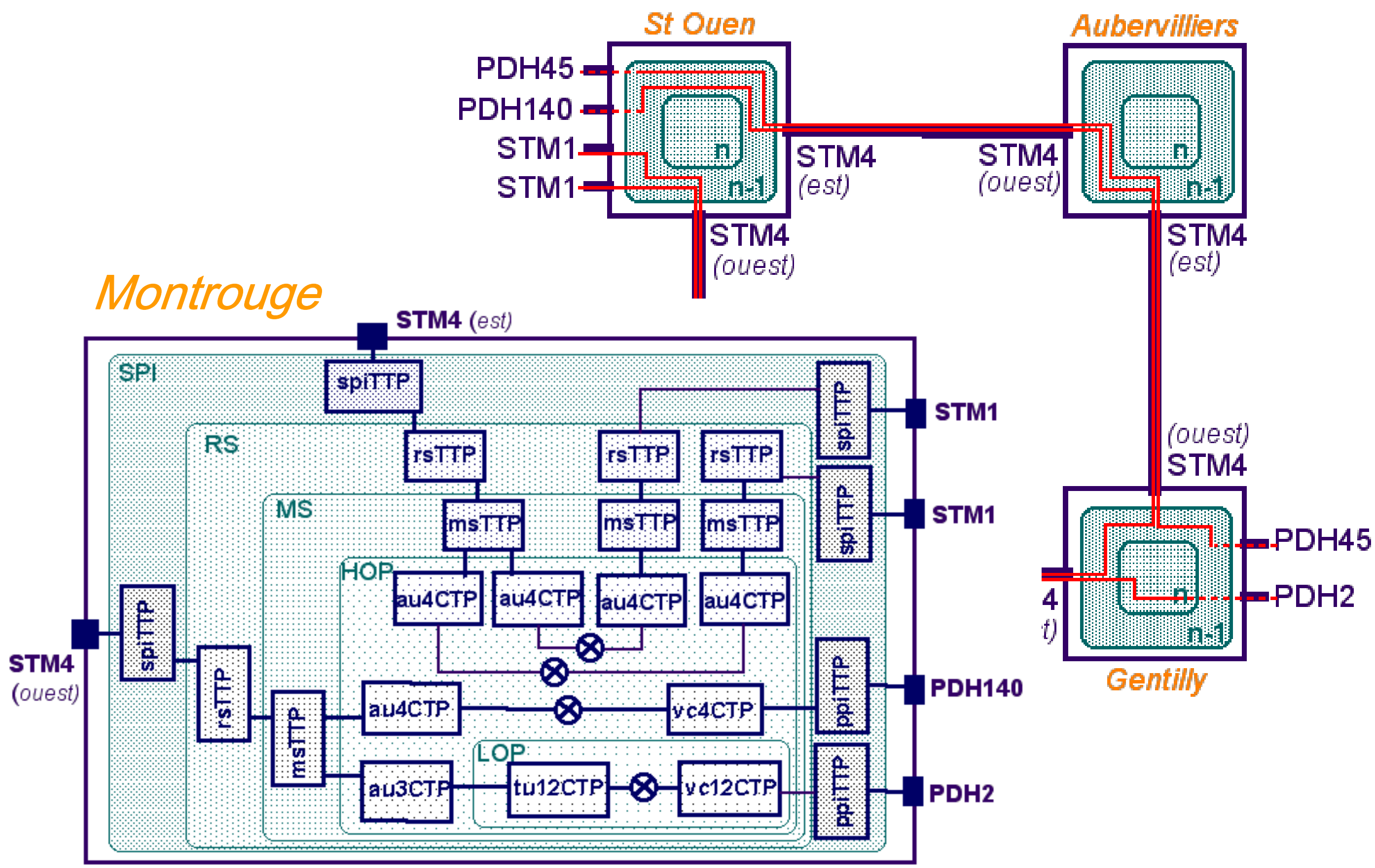
Magda + Magda 2

- Two RNRT projects *(6 years overall)*
  - Partners : Alcatel, France Telecom R&D, Ilog, LIPN
  - Distributed alarm correlation and failure diagnosis,
  - implemented above a rule engine.



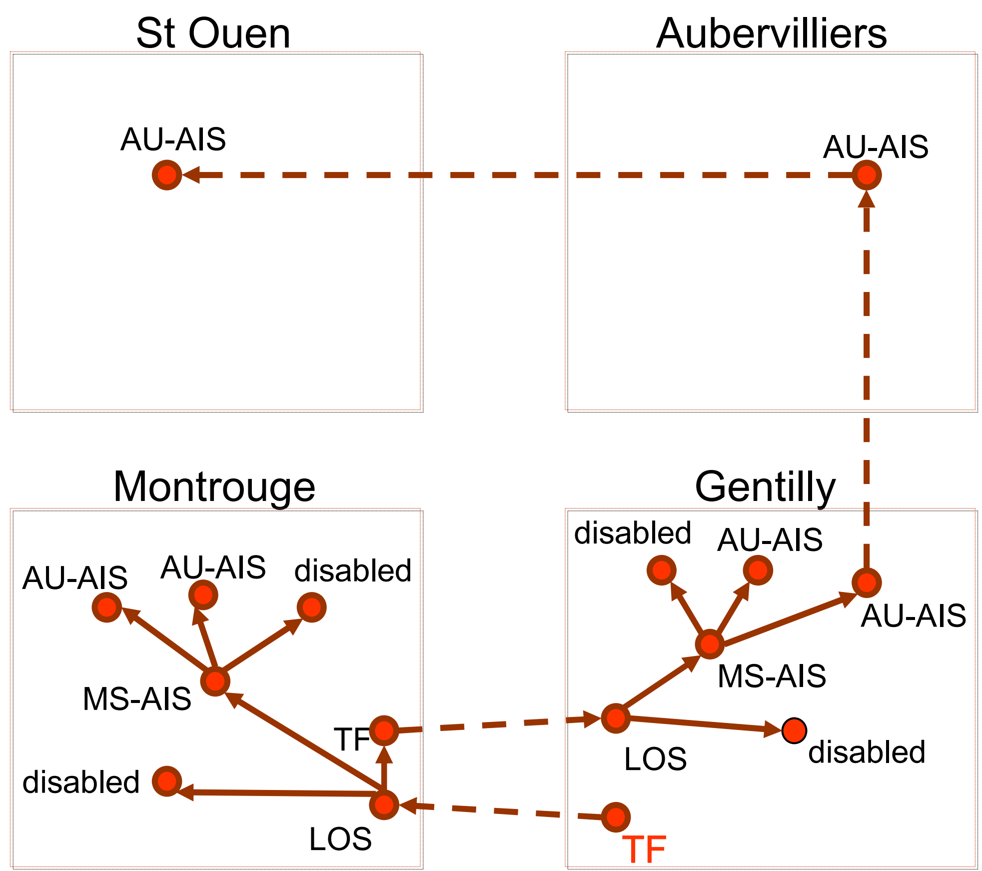
Magda + Magda 2

- Two RNRT projects *(6 years overall)*
  - Partners : Alcatel, France Telecom R&D, Ilog, LIPN
  - Distributed alarm correlation and failure diagnosis,
  - implemented above a rule engine.

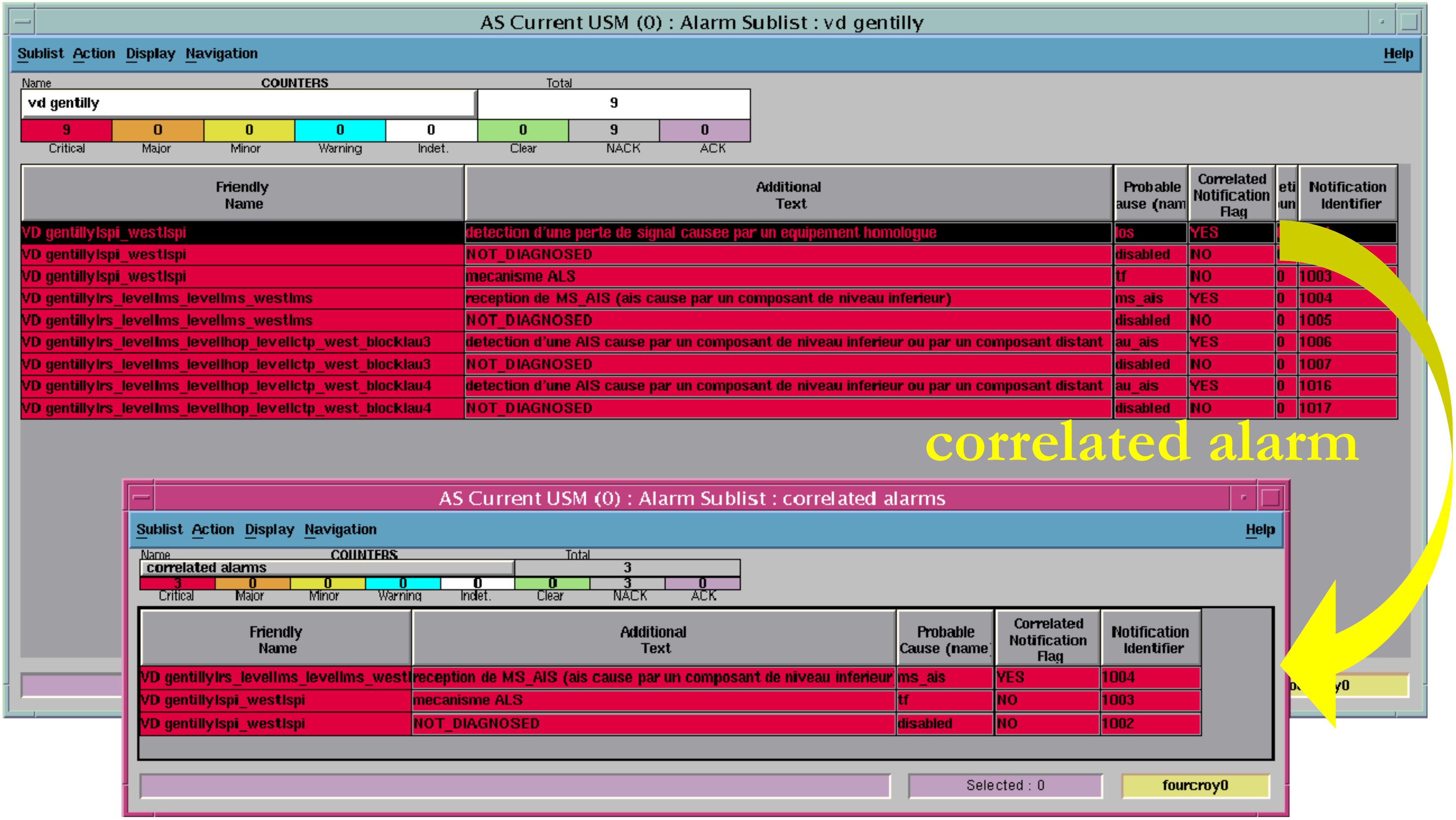


Magda + Magda 2

- A typical alarm correlation pattern, reconstructed with distributed supervisors

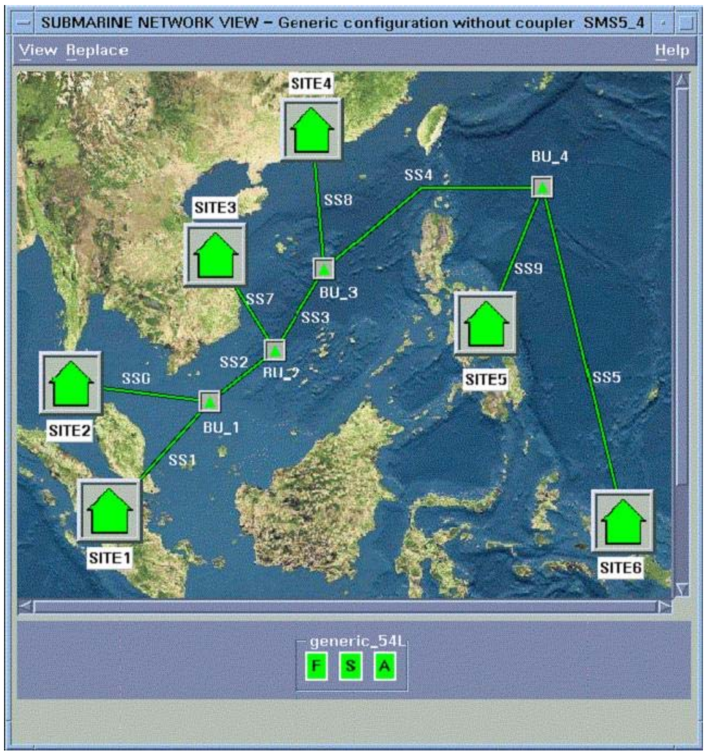


Magda + Magda 2



VDT contract

- Partner : Alcatel R&I + Optical Networks Division *(1 year)*
  - alarm correlation for a submarine-line terminal equipment
  - centralized, but unfolding-based correlation



Outline

- Static distributed systems
  from Markov fields to abstract distributed systems
- Networks of automata
  introduction of the time dimension
- Networks of concurrent systems
  a partially ordered notion of time
- Applications
  distributed diagnosis in telecommunication networks
- Perspectives

Holes in the theory…

- Complexity issues…
- Distributed optimization
- Robustness issues:

  alarm selection/rejection, as in *chronicles*
- On-line collection of (partial) results

  introduction of true *distributed programming* aspects
- What for systems with changing architecture ?

  e.g. web services, mesh networks, …

Holes in the theory…

- Complexity issues…
- Distributed optimization
- Robustness issues:
  alarm selection/rejection, as in *chronicles*
- On-line collection of (partial) results
  introduction of true *distributed programming* aspects
- What for systems with changing architecture ?
  e.g. web services, mesh networks, …

New Applications

- Finite complete prefixes in factorized form,
  already started with Agnes Madalinski.
- Distributed optimal planning
  cooperation project with Univ. of Canberra
- Distributed control ?
  Probably in connection with game theory aspects.

Special thanks to

Armen Aghasaryan
Albert Benveniste
Thomas Chatain
Didier Devaurs
Christophe Dousson
Claude Jard
Christine Guillemot
Arnaud Guyader
Stefan Haar
Christoforos Hadjicostis
Guy-Bertrand Kamga
Mutlu Koca
Bruno Marquie
Agnes Madalinski
Patrick Perez
Vincent Pigourier
Helia Pouyllau
Laurie Ricker
Aline Roumy
Mark Smith
Alexandre Skrzypczak
Julien Thomas
Franck Wielgus
...