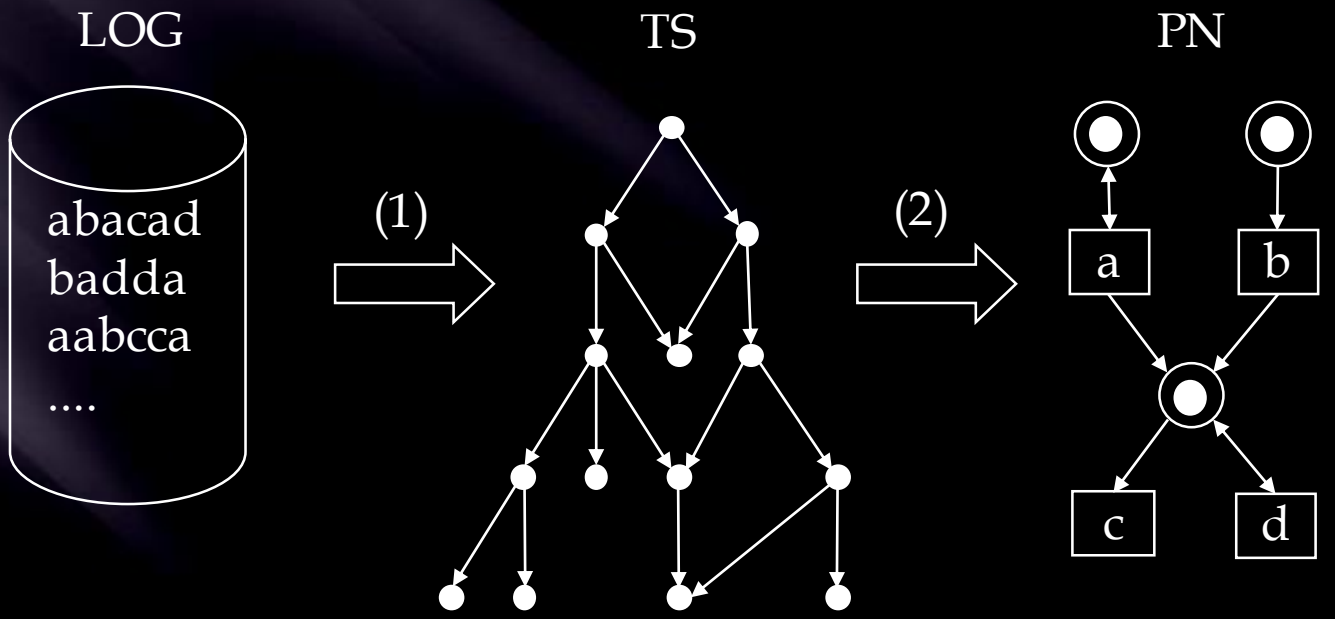
Outline

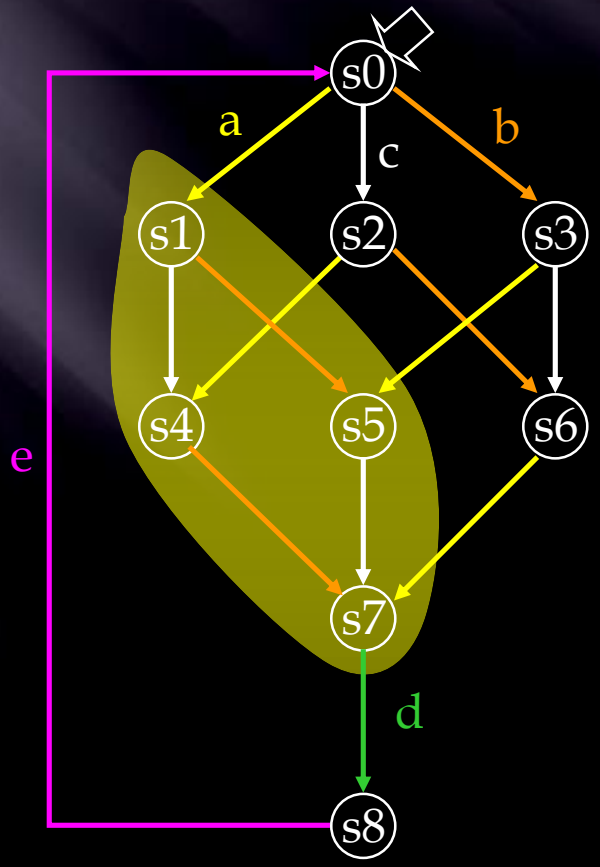
- **Region-based approach for Process Mining**
- Decompositional strategy: sequential views
- Divide-and-Conquer strategy

From logs to TS



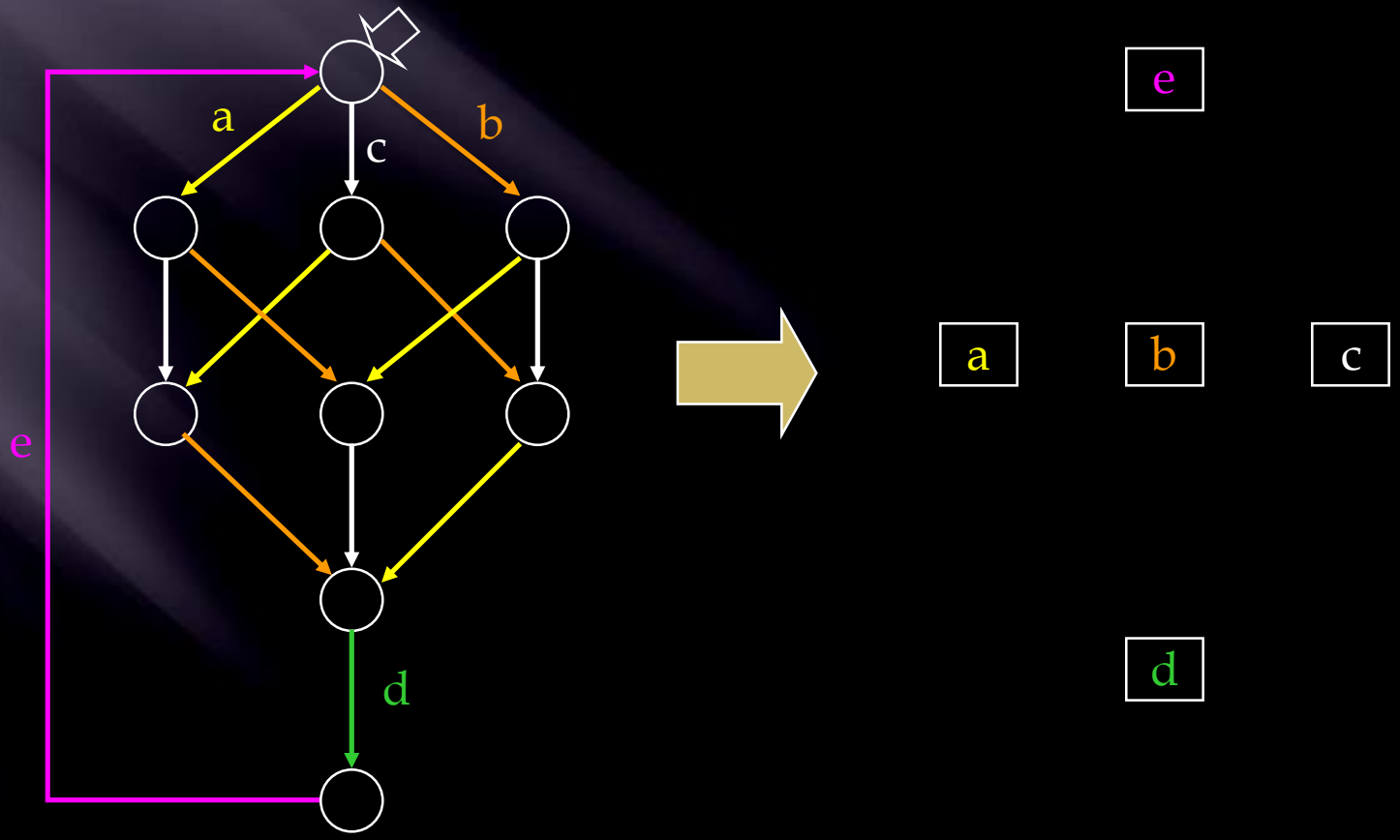
Region [Ehrenfeucht & Rozenberg]

A set of states is a region if every event has an homogeneous relation with the region



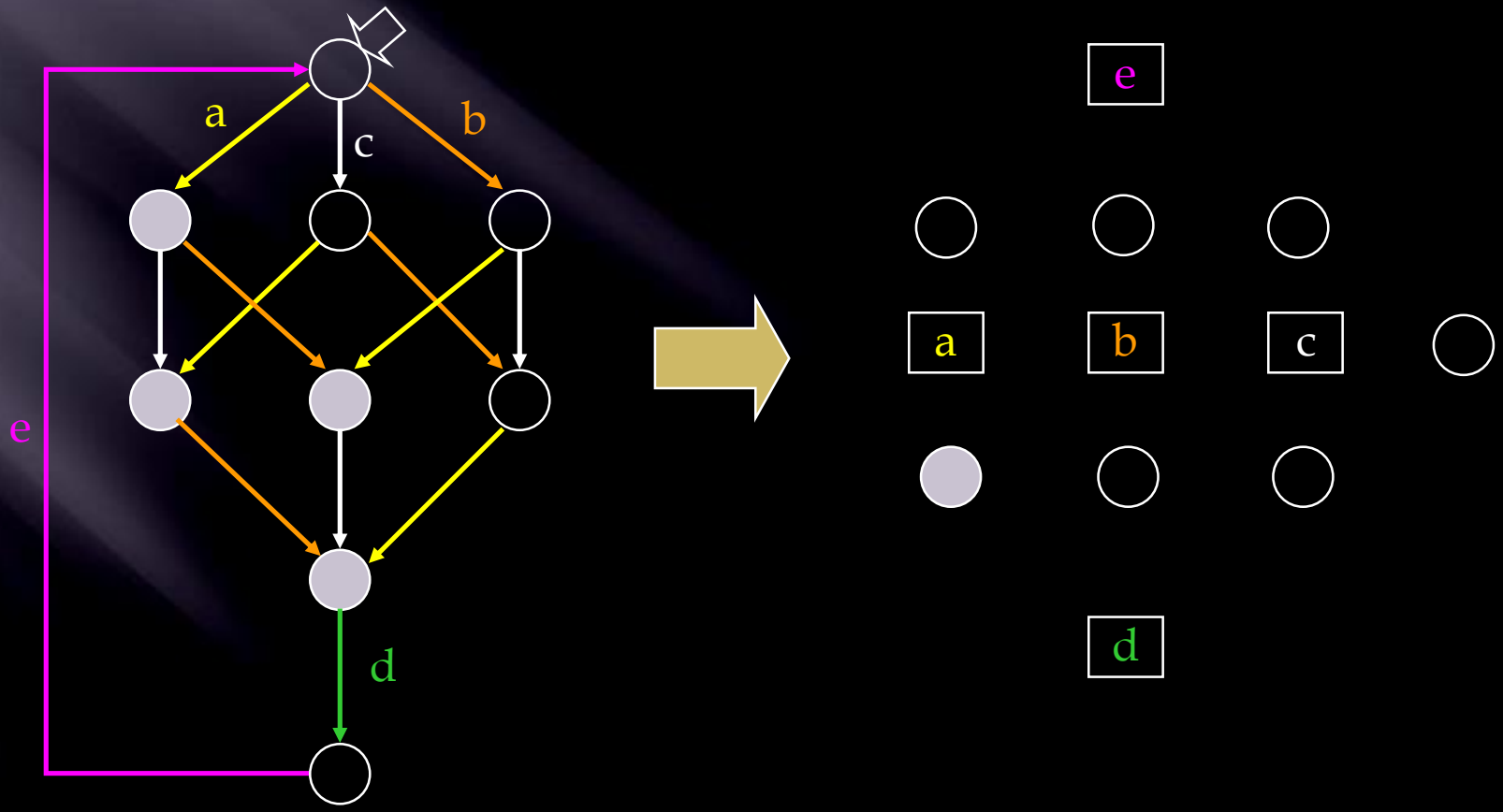
- S = {s1,s4,s5,s7}
  - a *enters* S
  - d *exits* S
  - c, b and e do not cross
- Regions correspond to PN places

PN synthesis [Ehrenfeucht & Rozenberg]



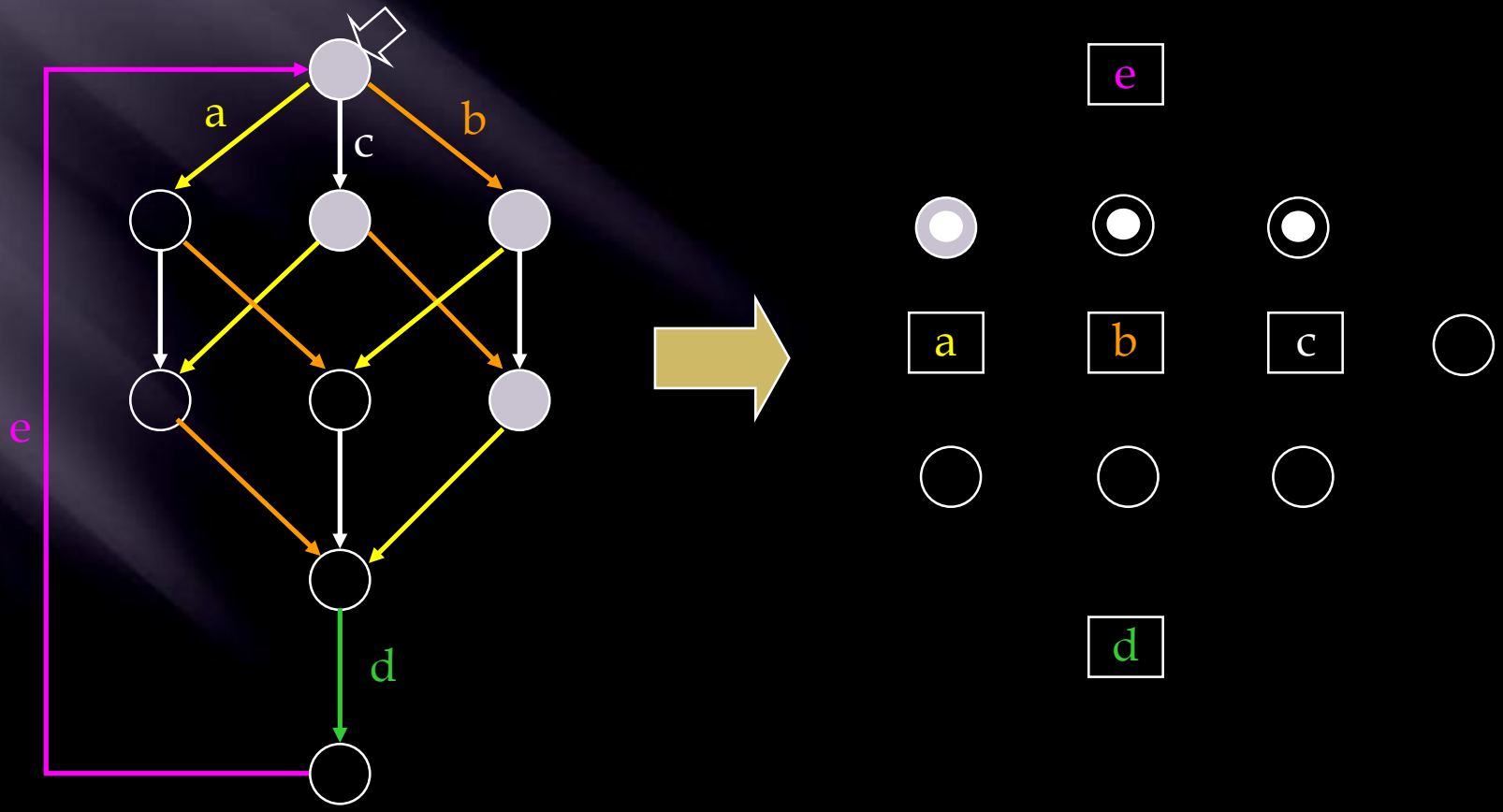
For every event generate a transition labeled with this event

PN synthesis algorithm



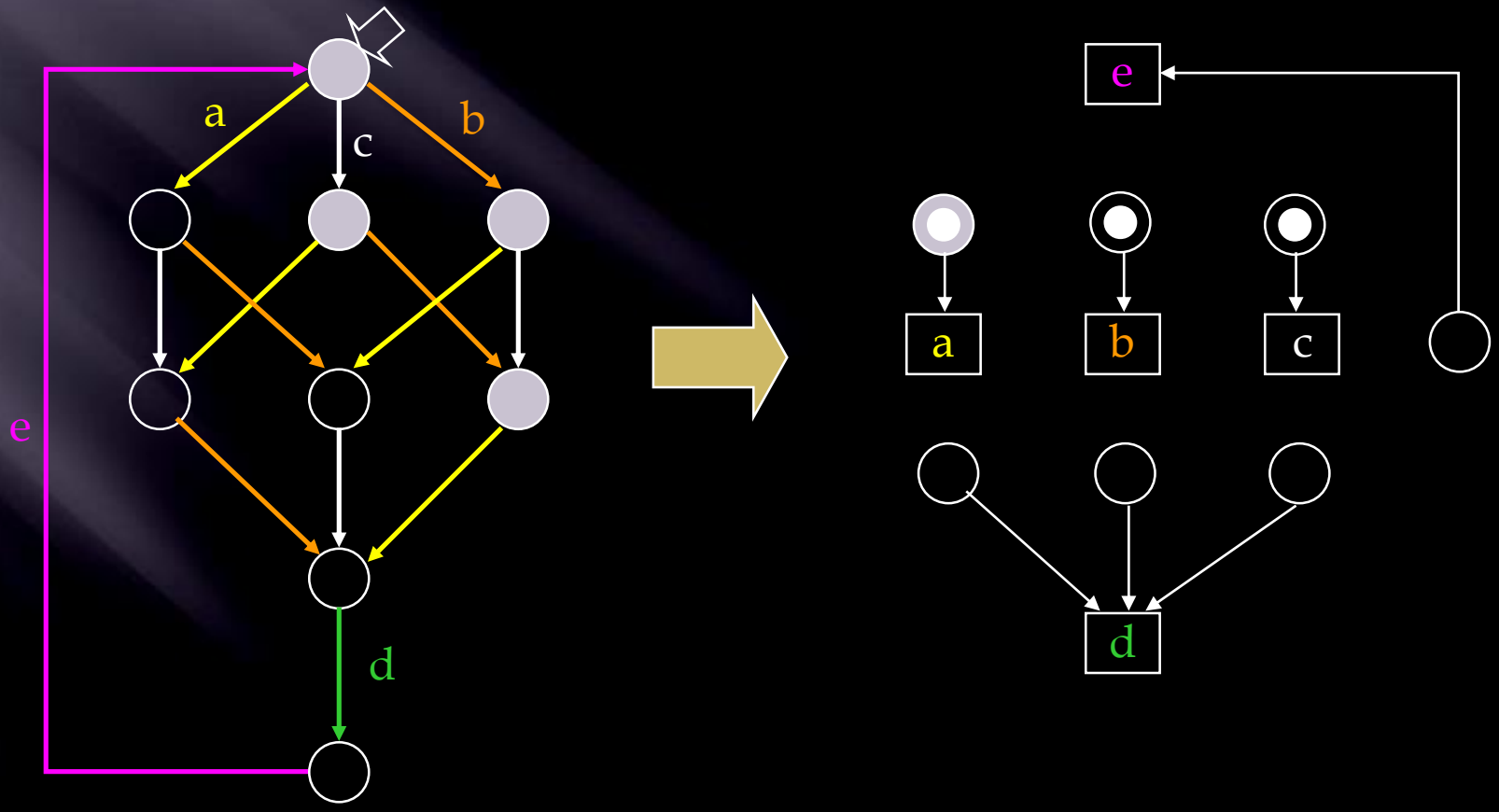
For every (minimal) region generate a place

PN synthesis algorithm



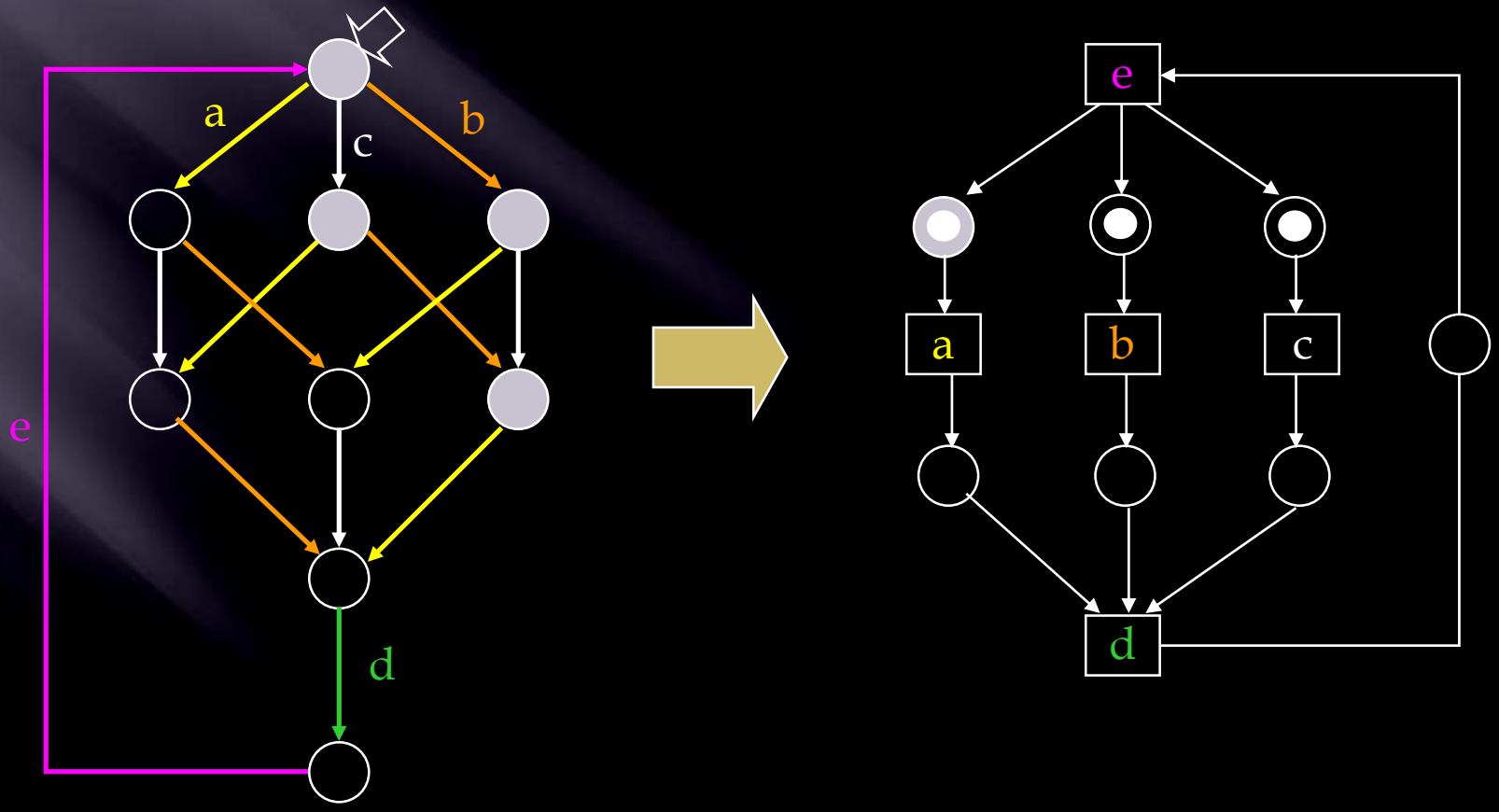
If region includes the initial state then mark the corresponding place

PN synthesis algorithm



If event exits the region add an arc between
the corresponding place and the corresponding transition

PN synthesis algorithm



If event enters the region add and arc between
the corresponding transition and the corresponding place

Limitations

- Large TSs cannot be handled by this approach
- The use of efficient data structures and algorithms may help, but the problem still remains
  - Implicit representation (Binary Decision Diagrams)
  - Dynamic Programming
  - Exploration heuristics
- Language-based regions [Bergenthum *et al.*, Van der Werf *et al.*] suffer from the same problem

In the rest of the talk …

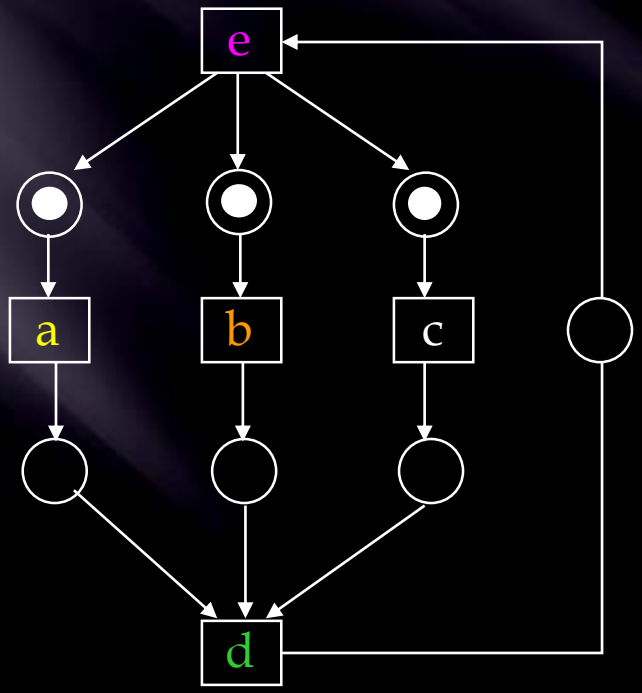
- Two strategies to overcome these limitations:

1) *Decompositional approach*: search for a set of **sequential** views of the log that jointly explain it

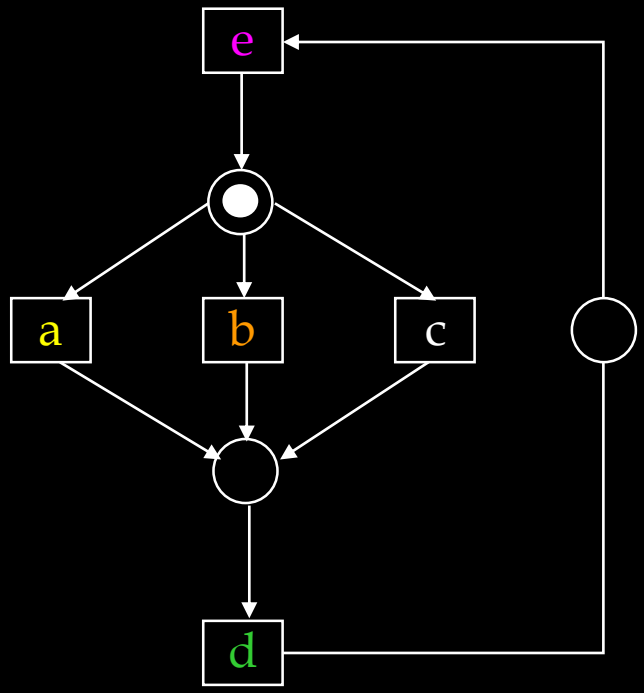
2) *Divide-and-Conquer approach:* iteratively **project** the log into smaller logs that can be handled afterwards

Decompositional Strategy

- Sequential views of the log ⇔ State Machines (SM)



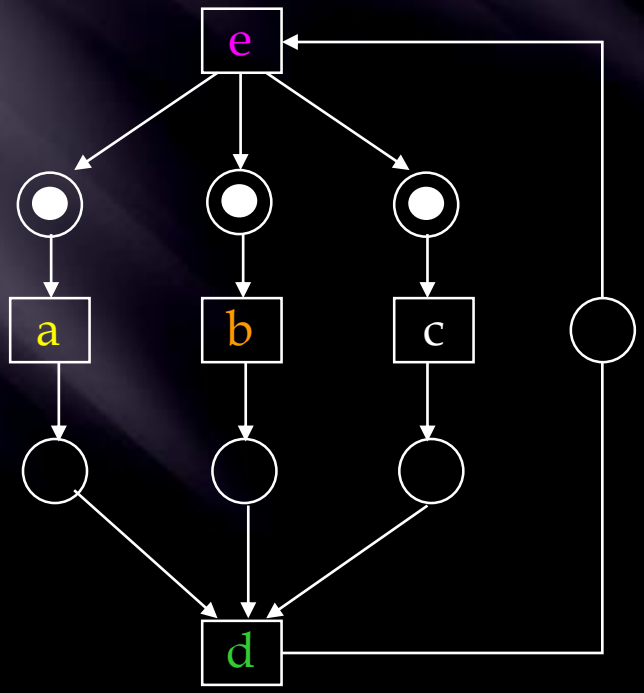
Not an SM: a,b,c concurrent

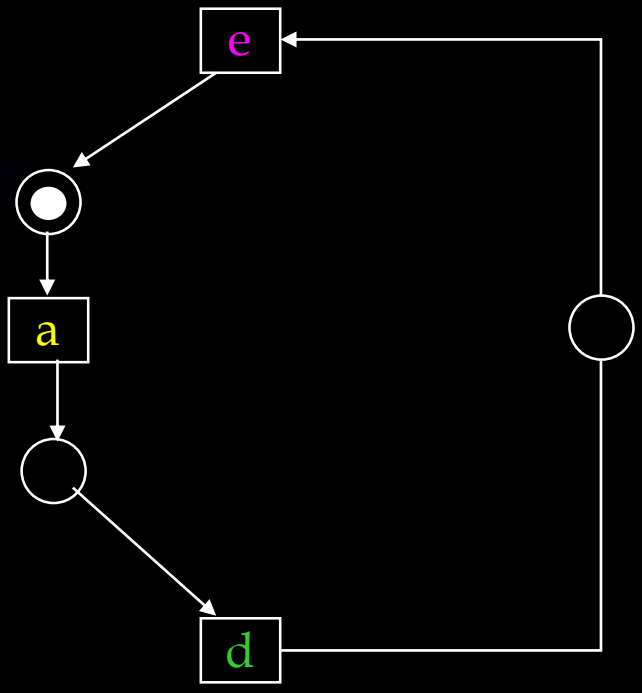


SM

Decompositional Strategy

- *State-machine component* of a PN:

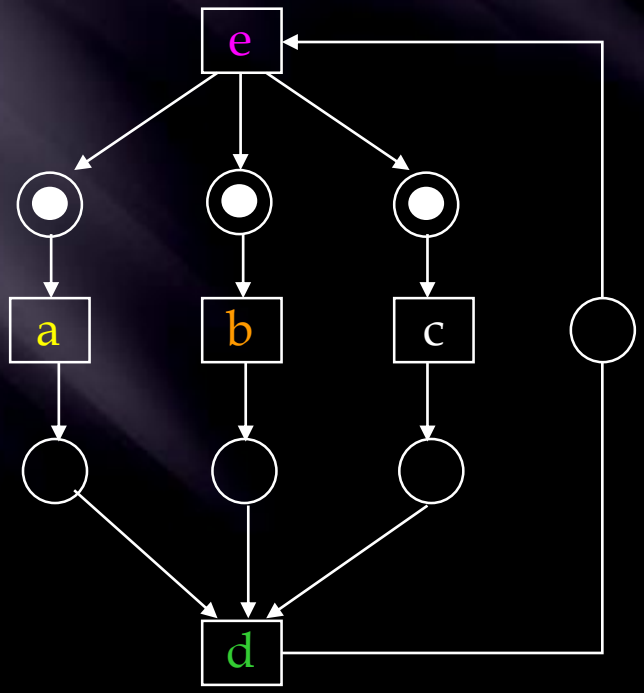


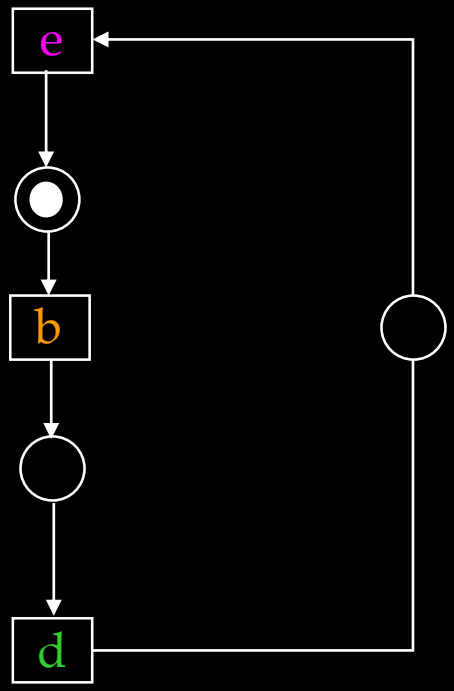


SM1

Decompositional Strategy

- *State-machine component* of a PN:

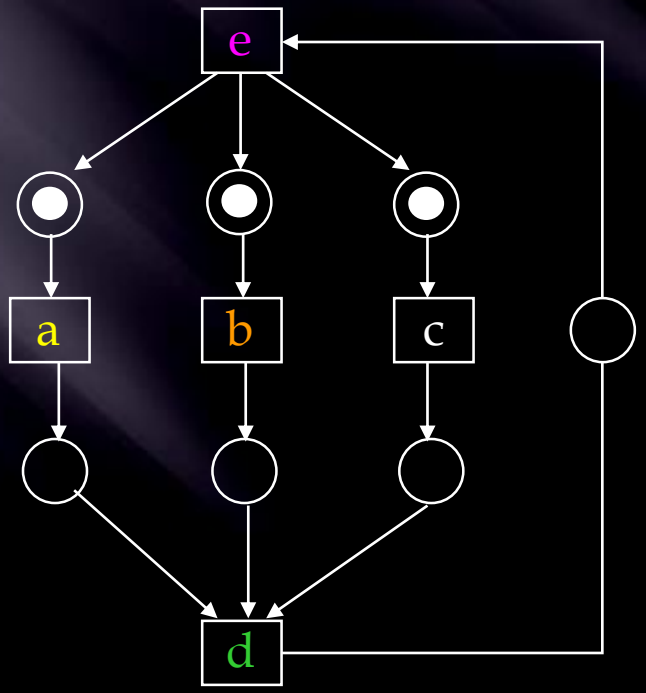


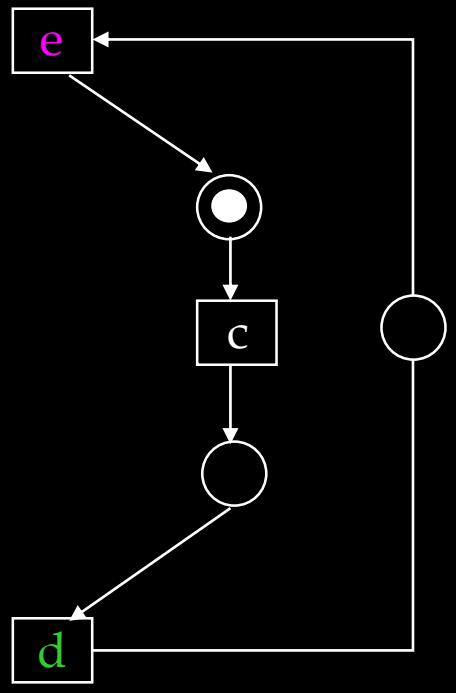


SM2

Decompositional Strategy

- *State-machine component* of a PN:

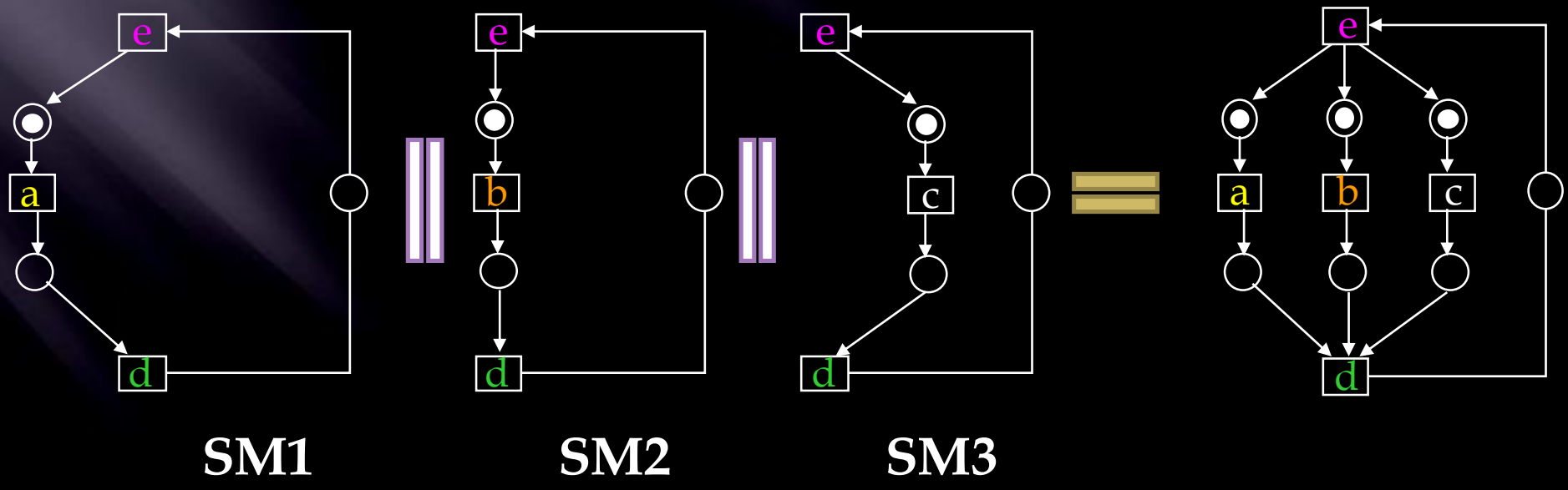




SM3

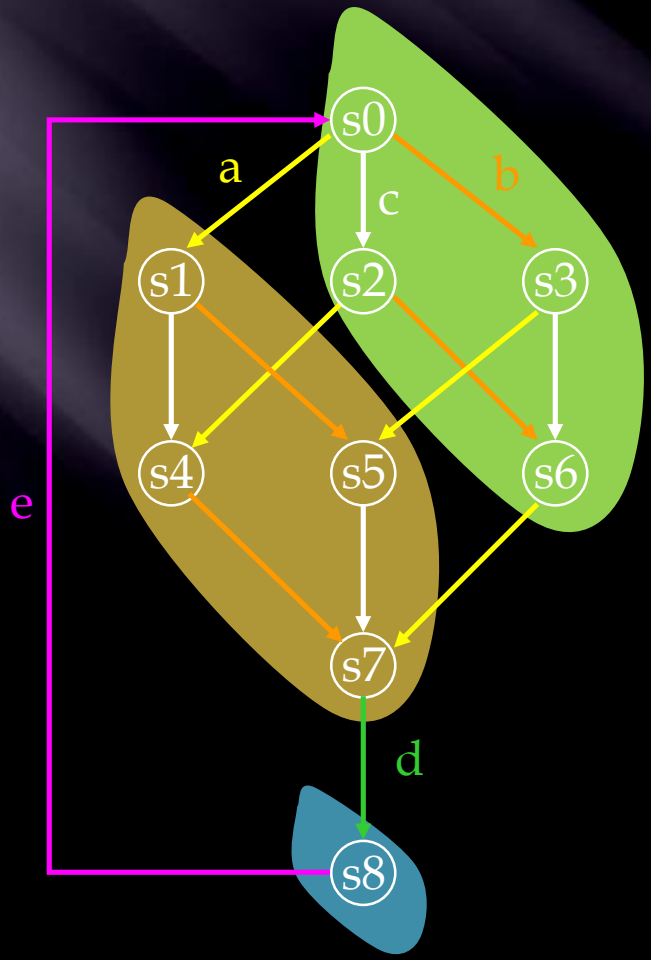
Decompositional Strategy

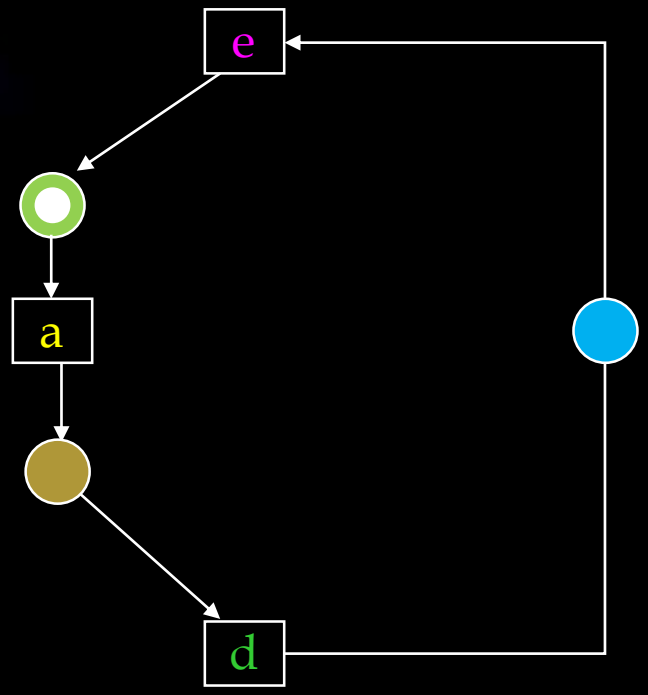
- *Parallel composition*:



Decompositional Strategy

Theorem: a **partition** of a TS by regions is a SM

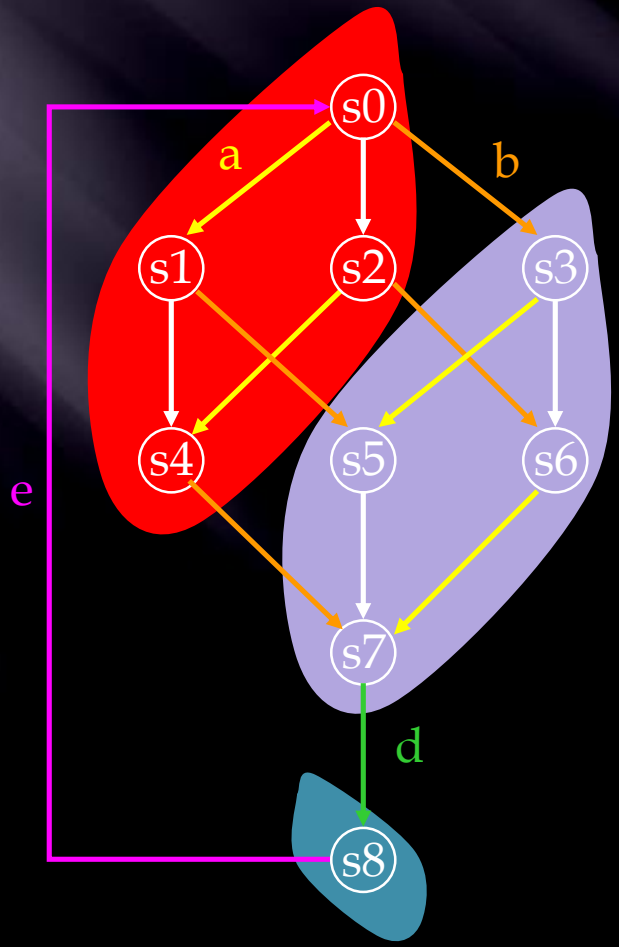


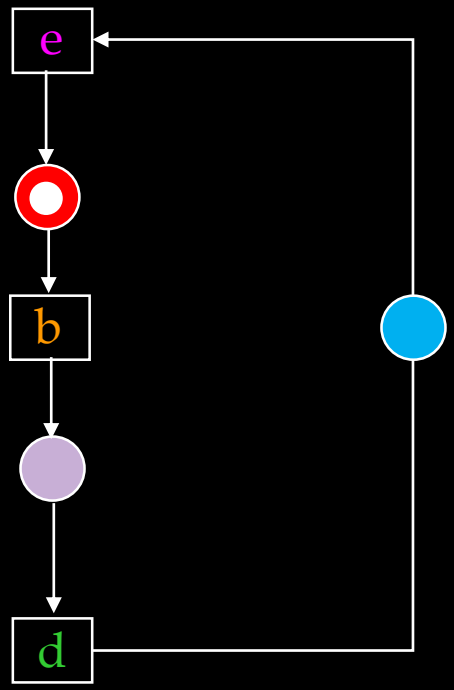


SM1

Decompositional Strategy

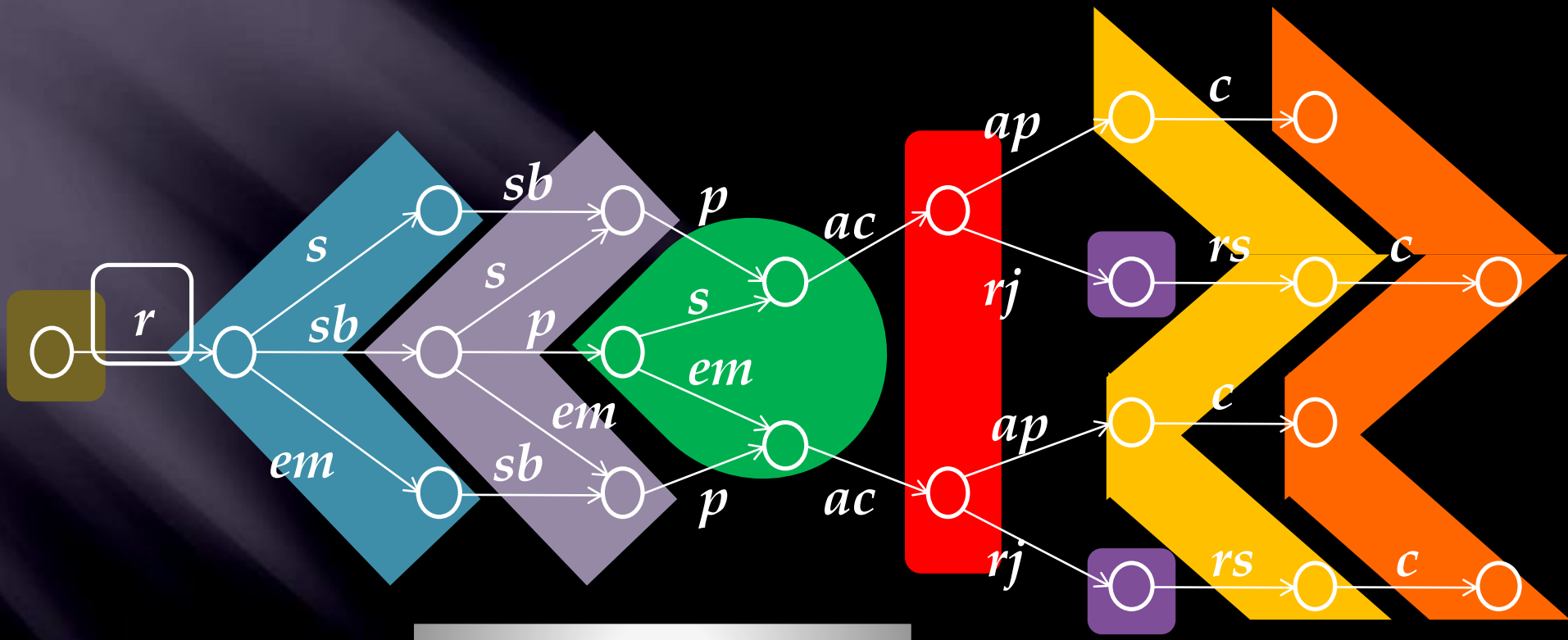
Theorem: a **partition** of a TS by regions is a SM



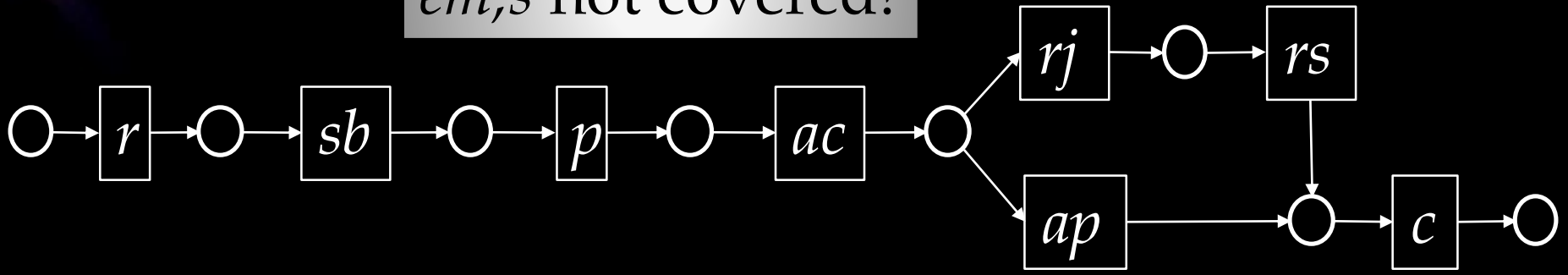


SM2

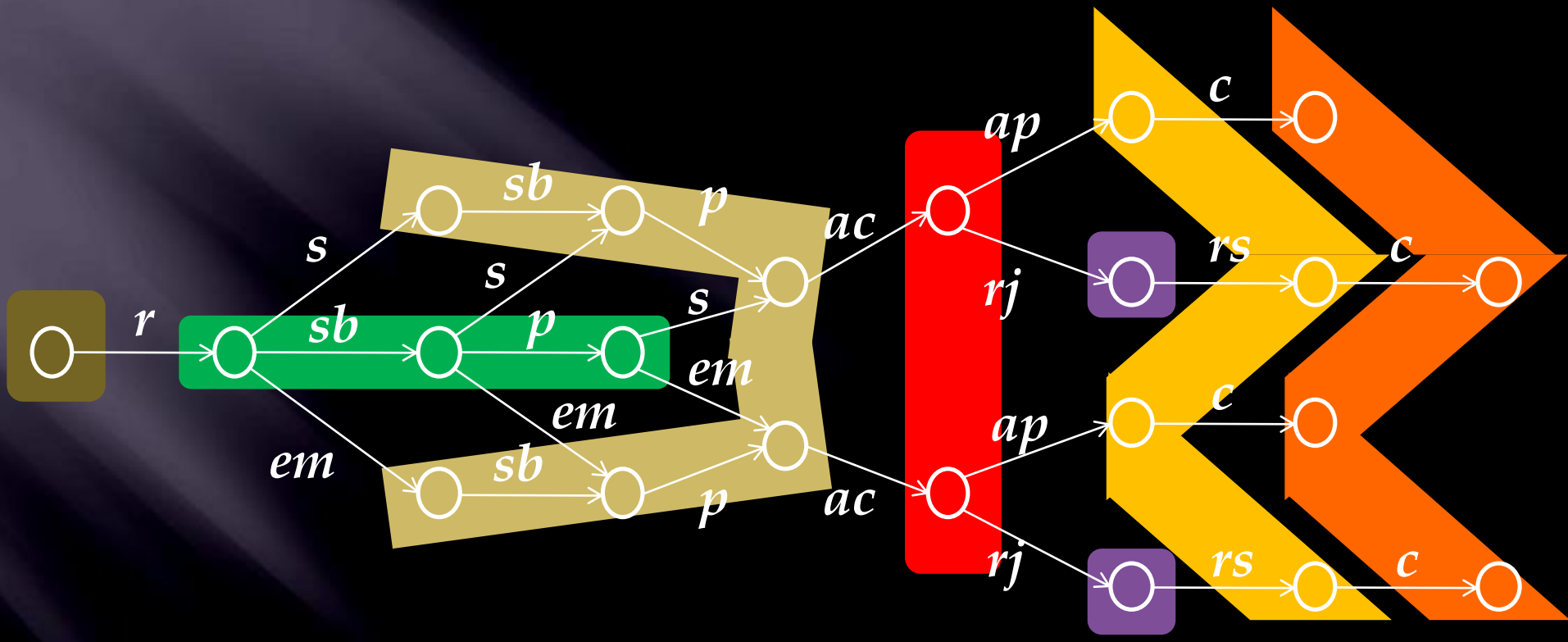
Decompositional Strategy

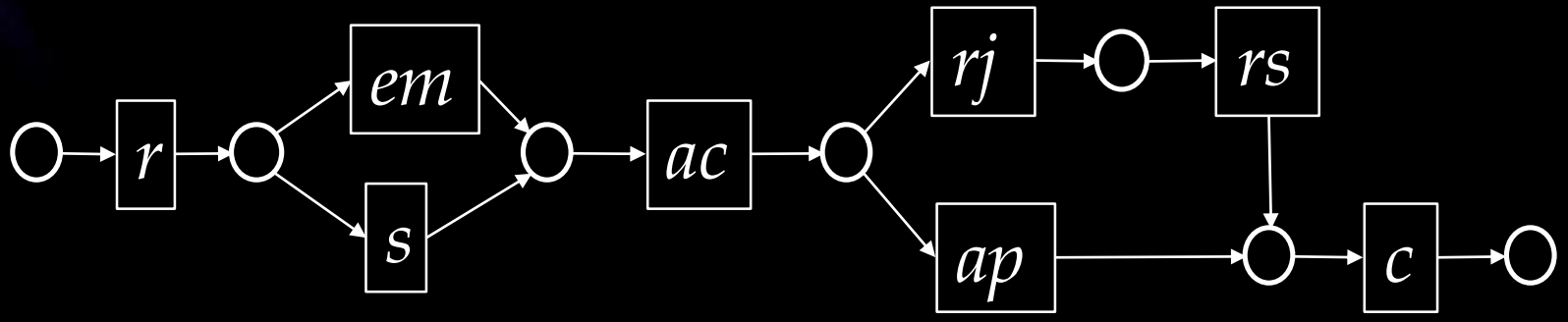


em,s not covered!

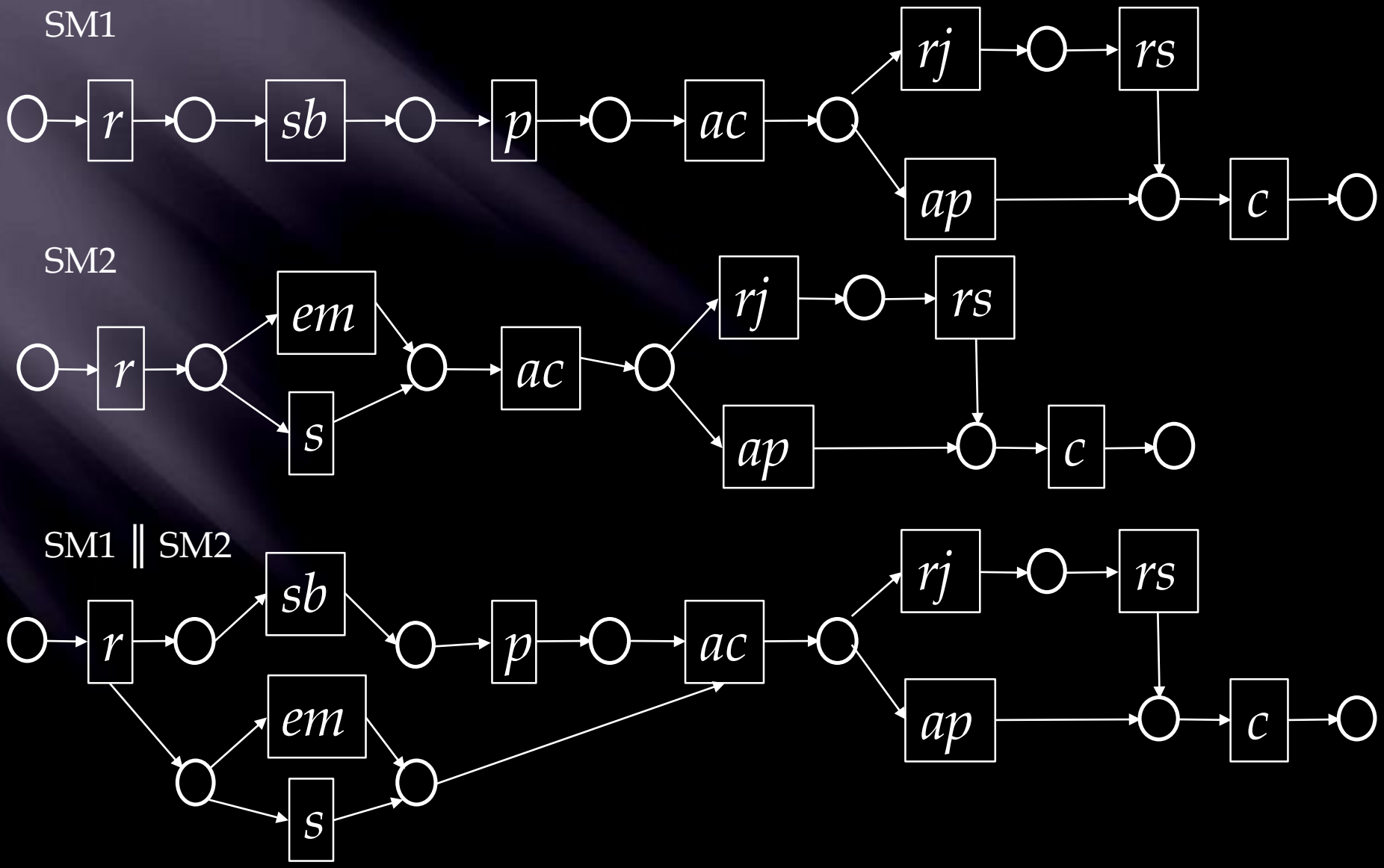


Decompositional Strategy





Decompositional Strategy



Decompositional Strategy

General Algorithm:

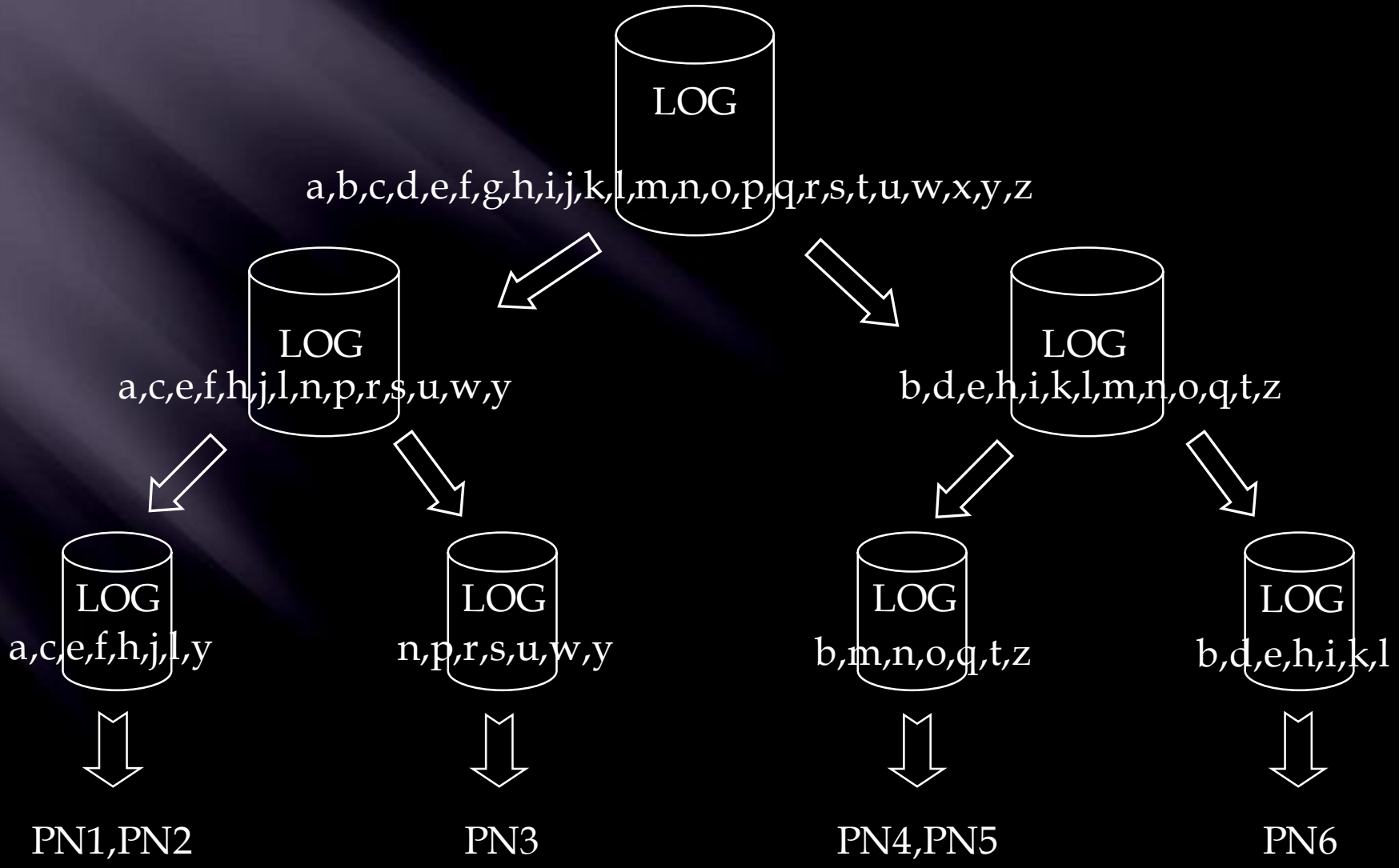
while (there is still event *ev* uncovered) **do**
find SM_i that covers *ev*;
i = i + 1;
update covering table
endwhile

Theorem: $L(TS) \subseteq L(SMC_1 \parallel \ldots \parallel SMC_n)$

Outline

- Region-based approach for Process Mining
- Decompositional strategy: sequential views
- **Divide-and-Conquer strategy**

Divide-and-Conquer Strategy

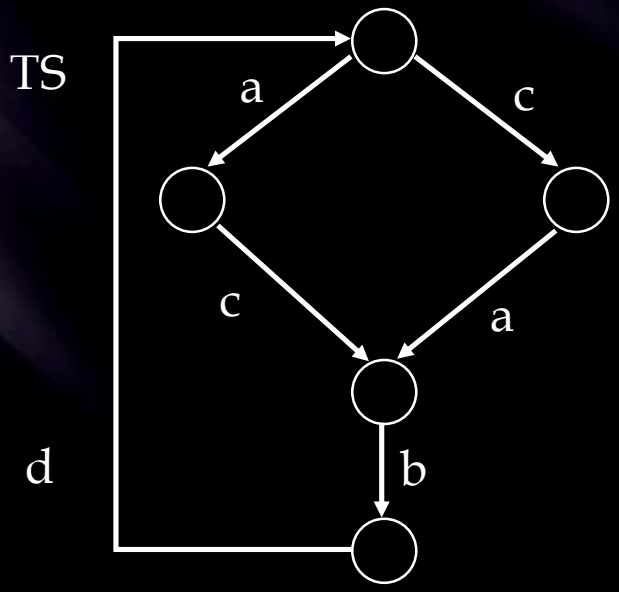


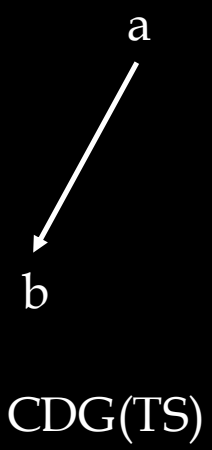
Event partition

- Events in the same class if:
  - Related by the log. Examples:
    - *Events performed by the same person*
    - *Events within a particular time period*
  - Causality relations: too strong notion to compute
  - Triggering relations: defined on the TS, easy to compute
- **How to obtain a meaningful partition ?**

Causal Dependency Graph (CDG)

- Graph having a node per each event of the log
- An arc between two nodes x, y iff x triggers y:

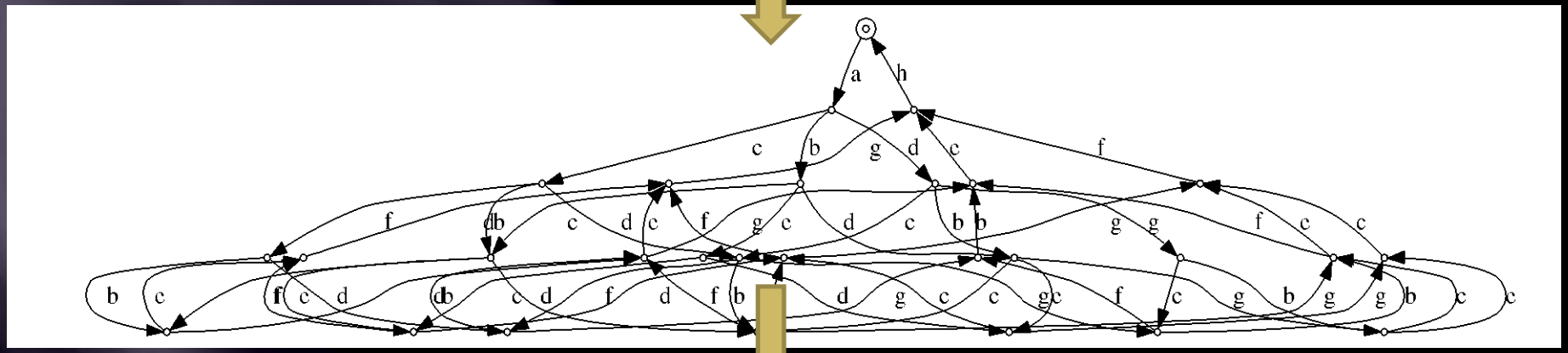




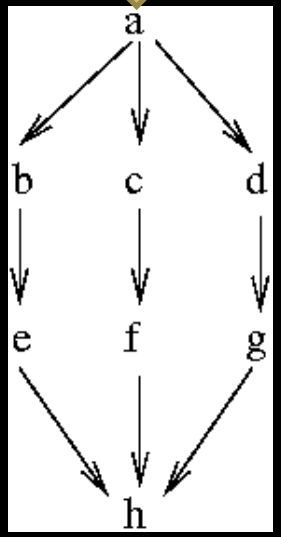
CDG(TS)

Causal Dependency Graph (CDG)

EVENT LOG



TS

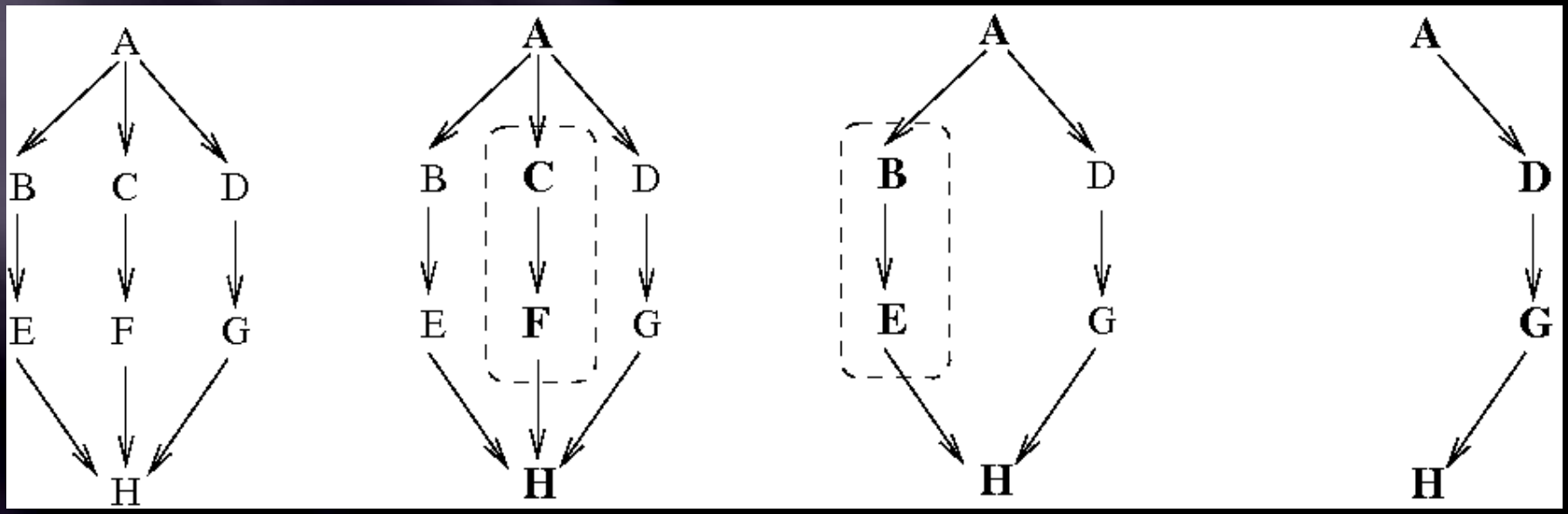


CDG(TS)

Spectral Approach

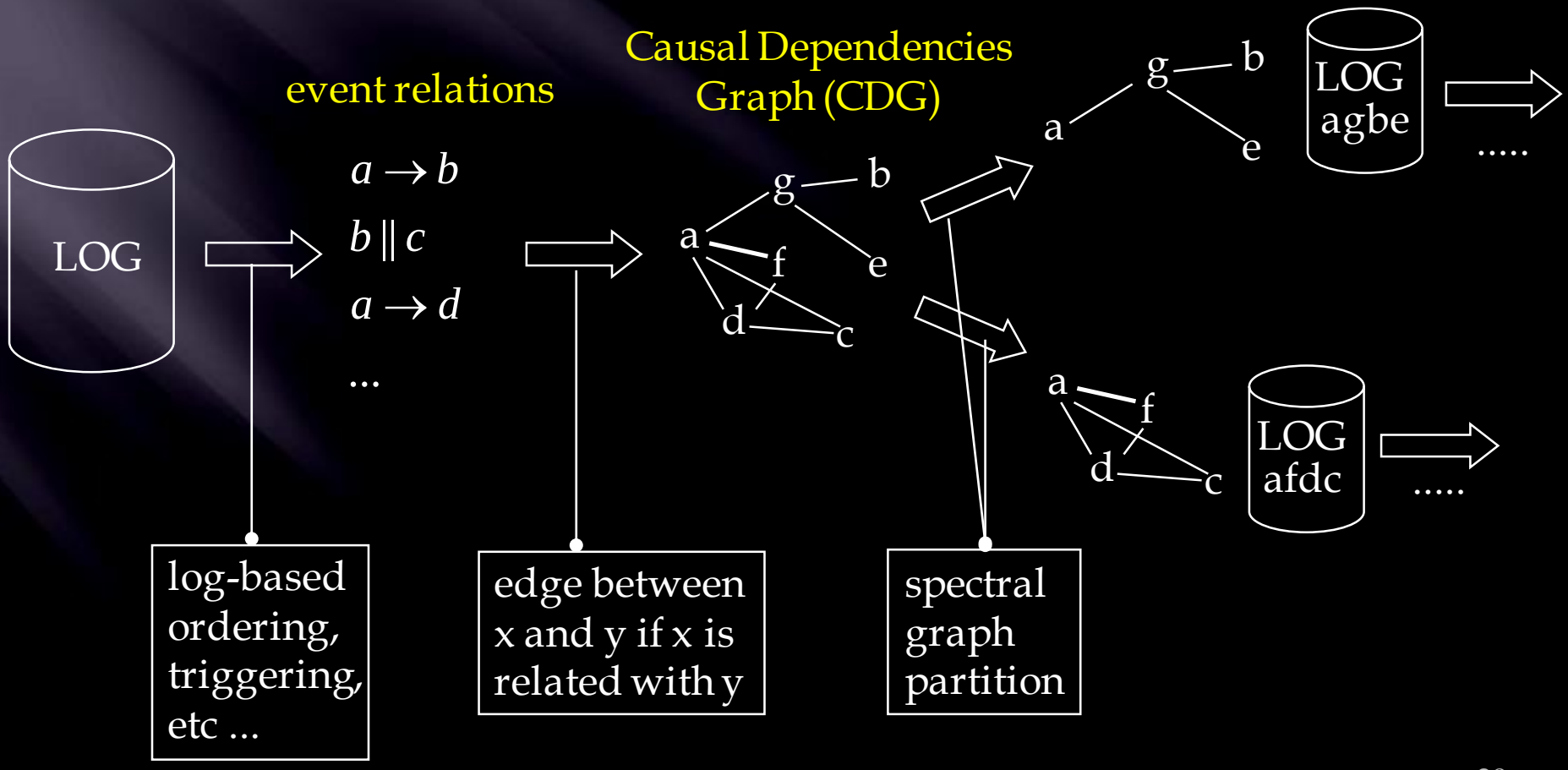
- Main idea: find *minimal cuts* in the CDG, i.e. sets of nodes tightly related between them and with few connections with the rest of the graph
- Short survey on Spectral Graph theory:
  - Graph G with *adjacency matrix* A and *degree matrix D*
  - $L = D - A$ is called the *Laplacian matrix* of G
  - In equation $L \cdot x = \lambda \cdot x$, x is called *eigenvector*, λ is *eigenvalue*
  - The *eigenvector* corresponding to the second *eigenvalue* of L is a good bi-partition (cut) of G [Cvetkovic *et al.*]
  - Efficient algorithms exist to compute this vector

Spectral Approach: example

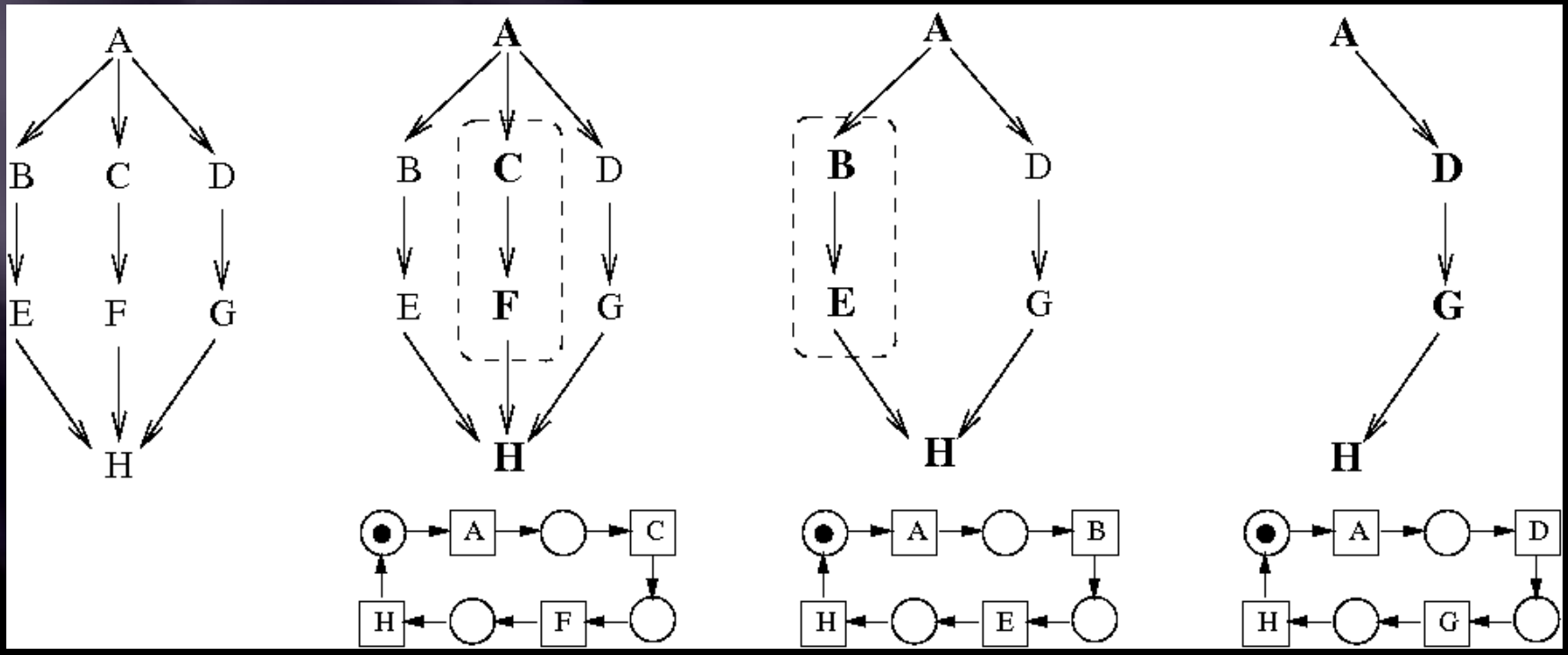


INITIAL CDG(TS) FIRST CUT SECOND CUT

Divide-and-Conquer Strategy



Spectral Approach: example



SM1 SM2 SM3

Divide-and-Conquer Strategy

General Algorithm:

compute CDG(TS);

(E_1, ... , E_n) = FindSpectralPartition(CDG(TS), MaxSize);

forall E_i **do**

E_i = E_i + AddBorderEvents(CDG(TS), E_i);

SMDecomposition(TS$|_{E_i}$)

endwhile

Theorem: $L(TS) \subseteq L(SMC_1 \parallel \ldots \parallel SMC_n)$

Thanks!