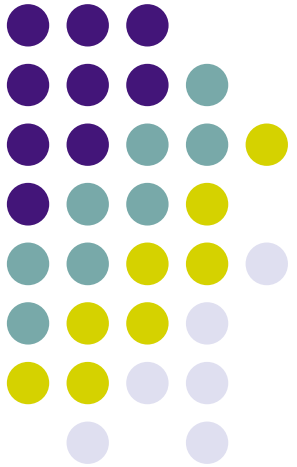
Automatic Composition of Services

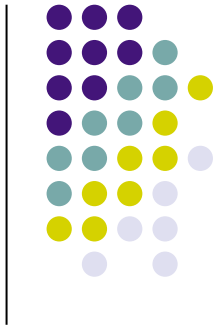
Fabio Patrizi

DIS

Sapienza - University of Rome

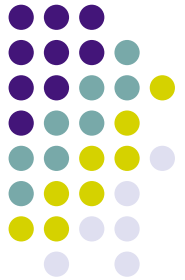


Overview



- Introduction to Services
- The Composition Problem
- Two frameworks for composition:
  - Non data-aware services
  - Data-aware services
- Conclusion & Research Direction

Services



- Given, modular, decoupled blocks
- Possibly distributed
- Interacting
- Possibility to compose!

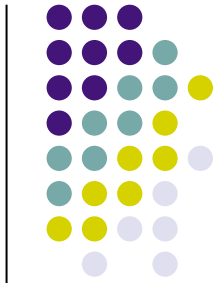
Service 1

Service 3

Service 2

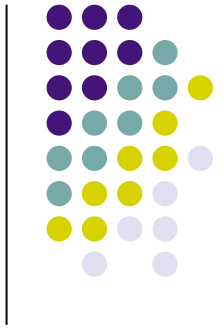
Service 4

Services (2)



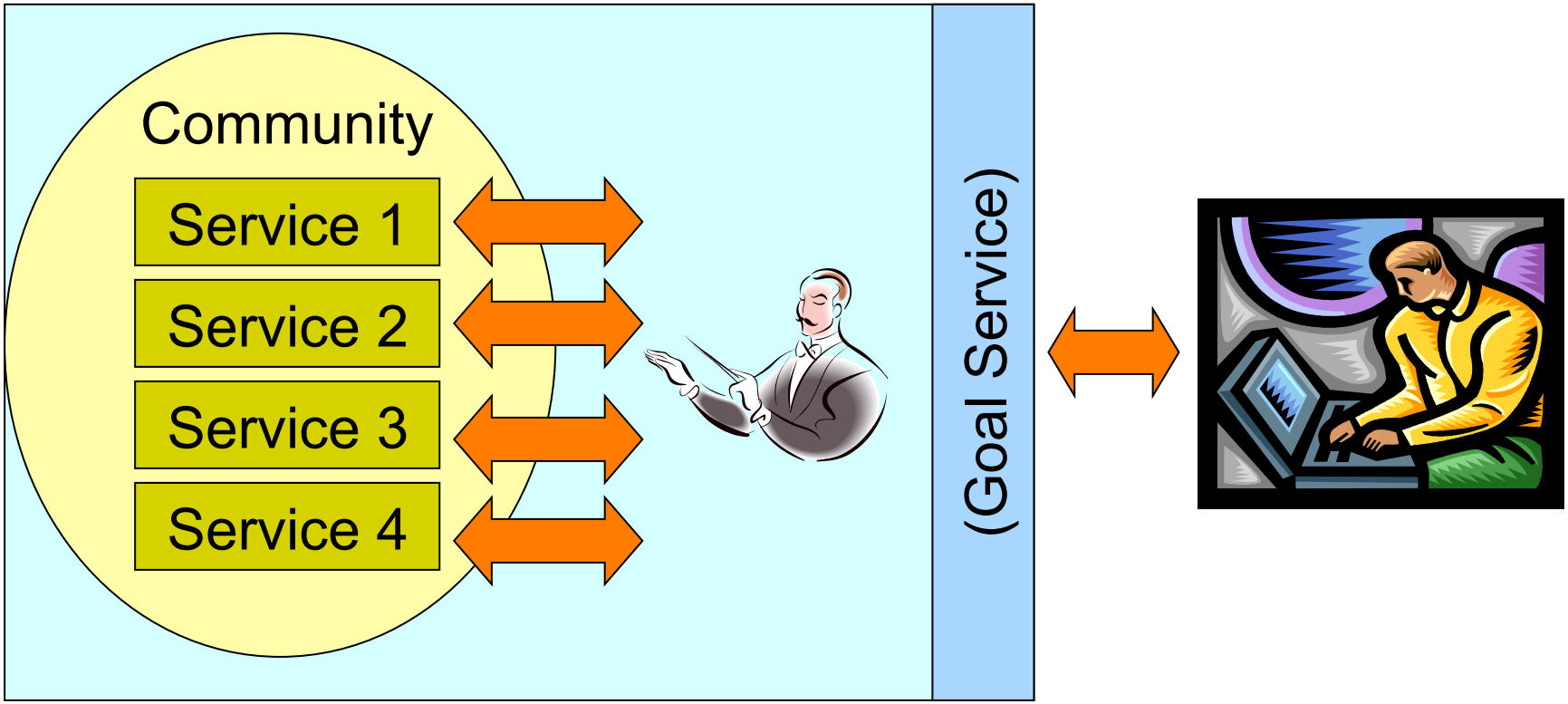
- Examples:
  - A (typical) set of web services over a network
  - A set of interacting autonomous agents

The composition Problem

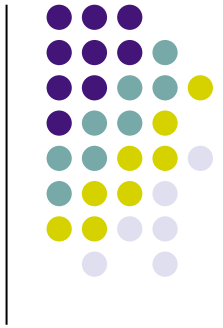


- Instance:
  - A set of available services
  - A (non available) goal service
- Solution:
  - An automaton which “mimics” the goal service, by delegating goal interactions to available services

Service composition



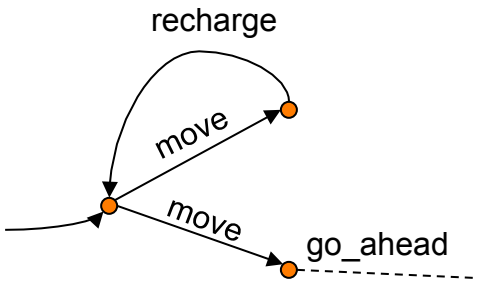
Modeling Services



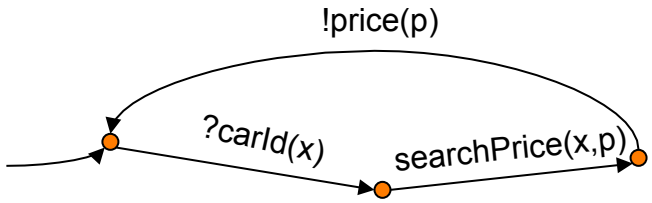
- Focus on behavior (vs in/out description)
- High-level descriptions (e.g.,WSDL, BPEL, process algebra) abstracted as
  - Finite Transition Systems (cf.[vanBreugel&Koshkina,06])
- Classification: Det, Ndet, Data, No-data

Services as TSs

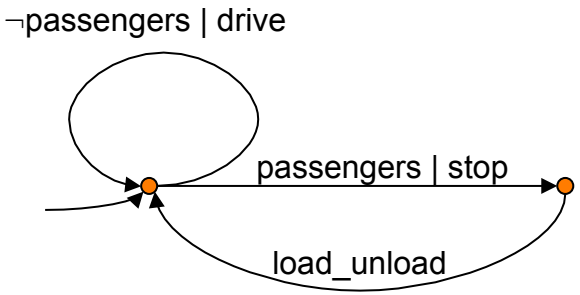
NDet



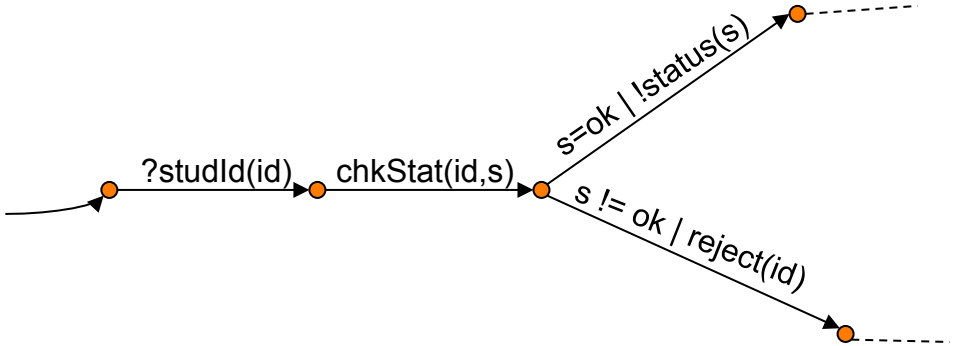
With Data/Messages

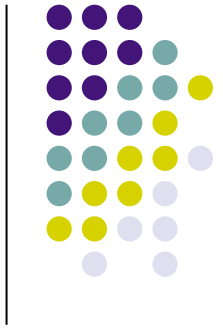


Guarded



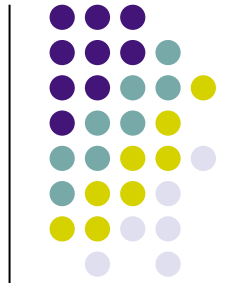
Combination





A Composition framework for Non data-aware services

The “Roman” Model [Berardi & al., ‘03, ’05]

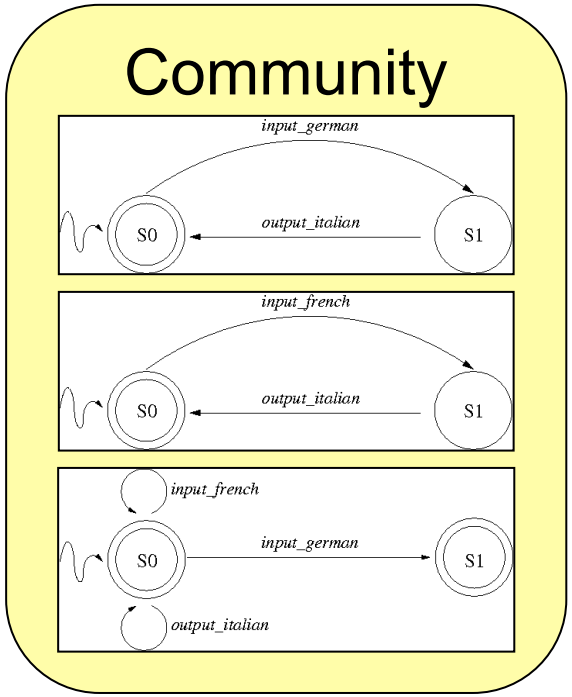


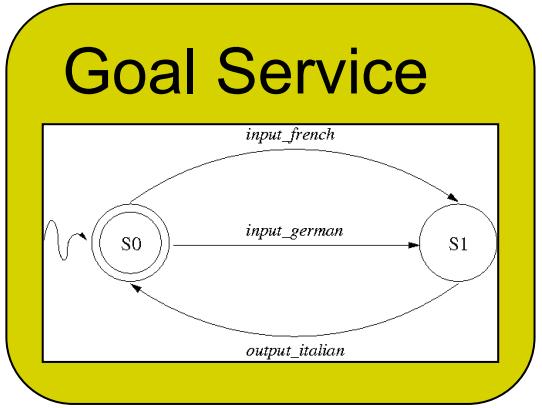
R.Hull, SIGMOD’04

- Focus on service behavior
- Atomic actions (abstract conversations)
- Asynchronous composition
- Extendible to NDet services (not here)
- Deterministic Goal service

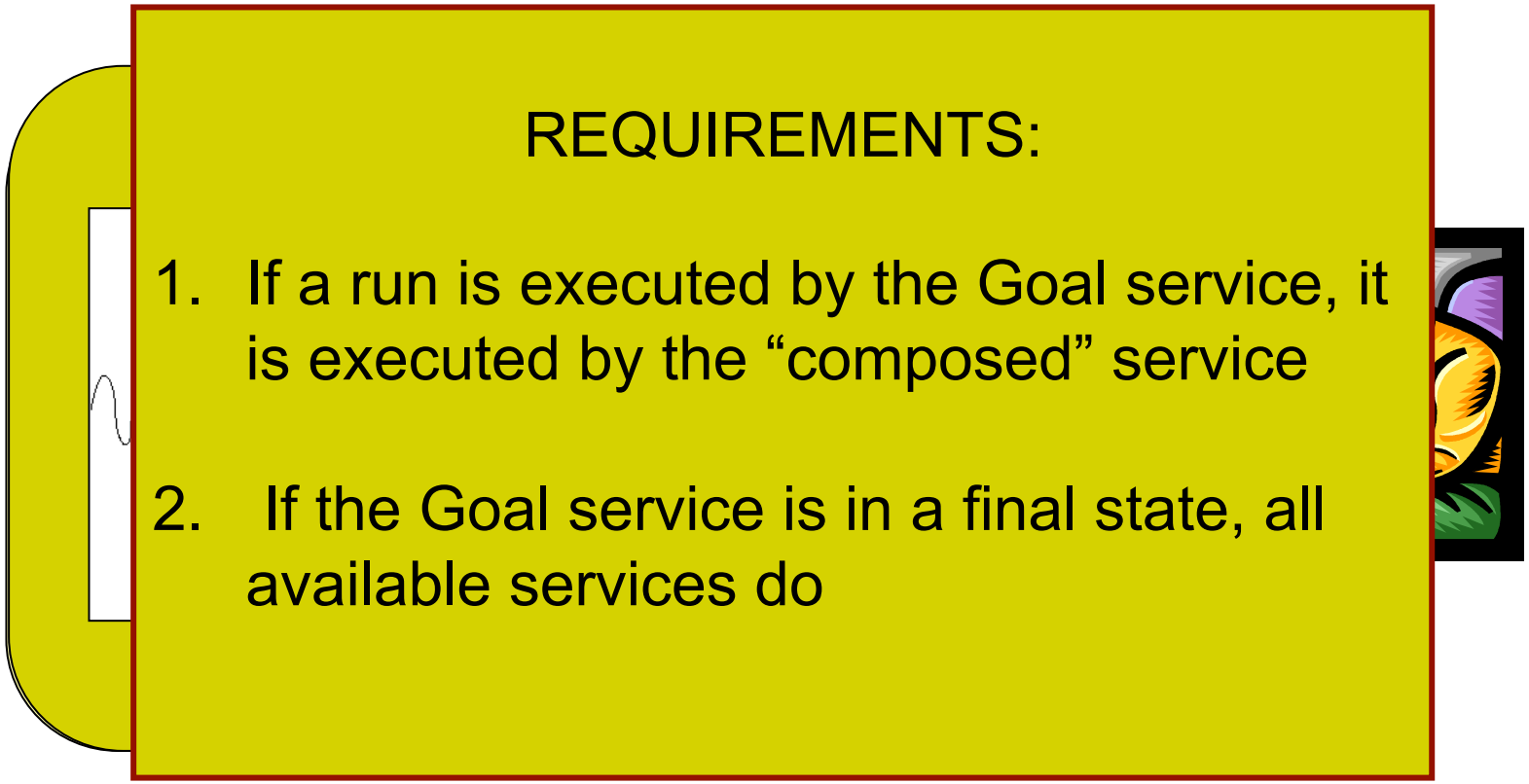
The “Roman” Model (2)

- A C*ommunity* of *services* over a shared alphabet $\mathcal{A}$
- A (Virtual) *Goal* service over $\mathcal{A}$

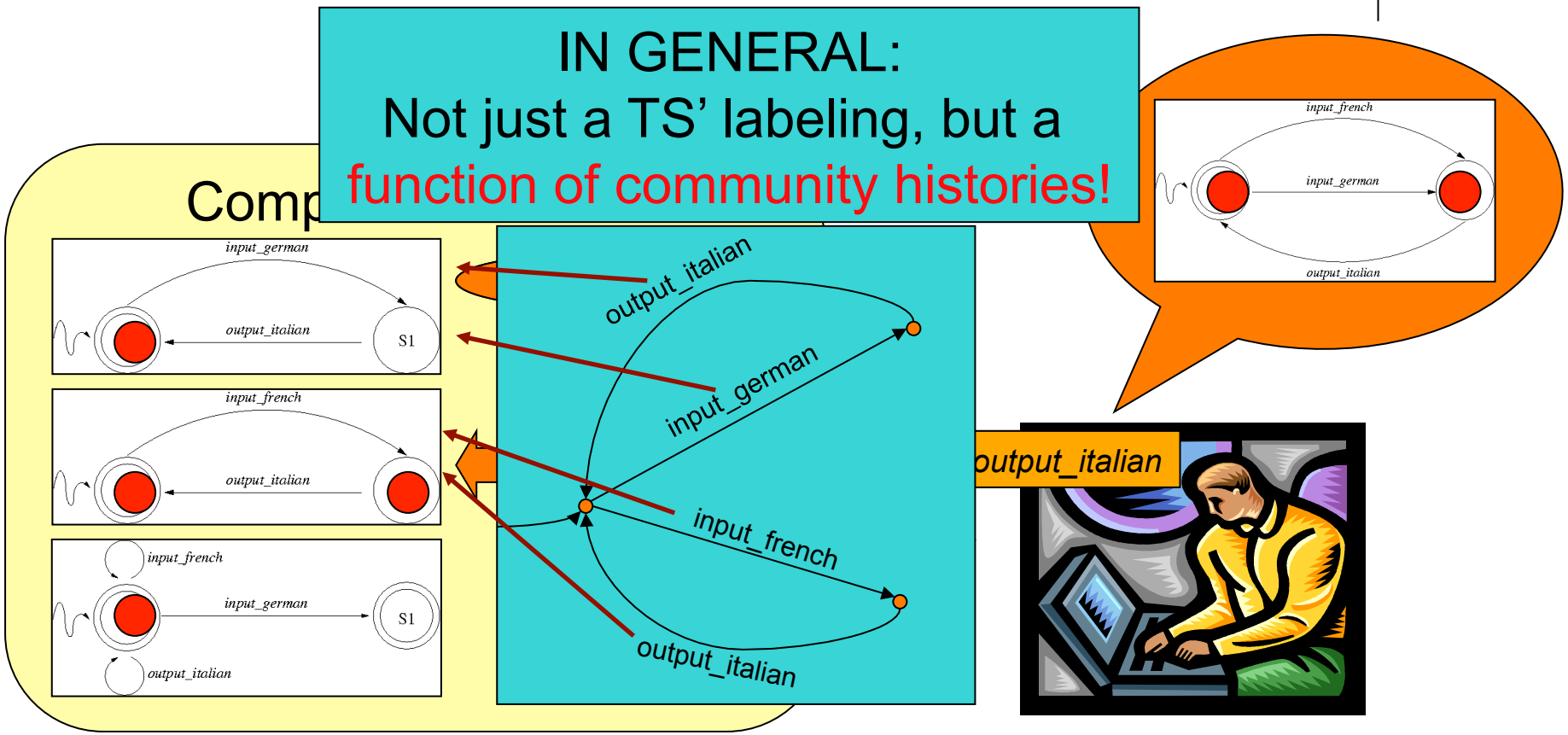




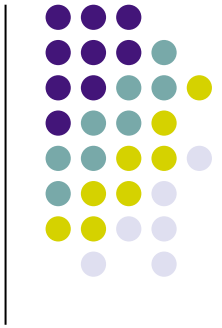
The “Roman” Model (3)



The “Roman” Model (4)



Orchestrators



Orchestrators are *functions of community histories*:

for each history and current action,

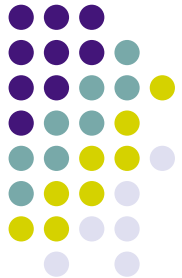
s ce

HOW TO COMPUTE ORCHESTRATORS?

Can be thought of as TSs,

possibly infinite state

Propositional Dynamic Logic



PDL[Fischer&Ladner, 79; Kozen&Tiuryn, 90;…]:

- Formu tures
- PDL-S
  - EXPTIME in the size of Φ

THEOREM[Berardi & al. '03]:

A PDL formula Φ can be built which is SAT iff an orchestrator exists

Encoding as PDL-SAT

$$\Phi = \mathbf{Init} \wedge [u](\phi_0 \wedge \bigwedge_{i=1,\ldots,n} \phi_i \wedge \phi_{aux})$$

Initial states of all services

target service

i-th available service

additional domain-independent conditions

$|\Phi|$ is polynomial in the size of services

Finding orchestrators (2)

Finding an orchestrator in the Roman Model is EXPTIME-complete

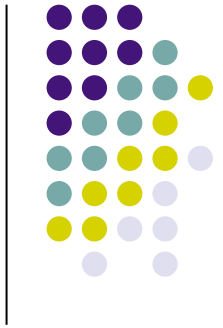
- Membership:
  - Reduction to PDL-SAT [Berardi & al. '03]
- Hardness:
  - By reducing existence of an infinite computation in LB ATM (EXPTIME-hard) [Muscholl & Walukiewicz '07]

Finding orchestrators

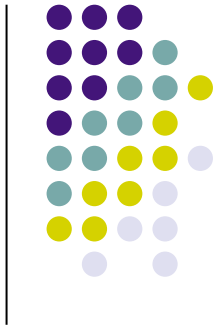
- THEOREM: If an orchestrator exists then there exists one which is finite state[Berardi et al. '03]

- Size at most exponential in the size of services $S_0,\ldots,S_n,S_g$

PDL Drawbacks

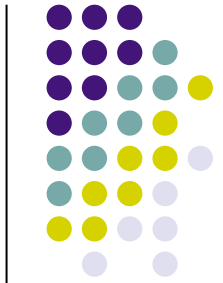


1. Only finite state orchestrators
2. Actual tools (e.g., Pellet@Univ. of Maryland) not effective:
   - Extracting models, thus orchestrators, not a trivial task: for efficiency reasons, only portions of the model are stored during tableaux construction



Service Composition Via Simulation

Simulation Relation



Given TS_1 and TS_2

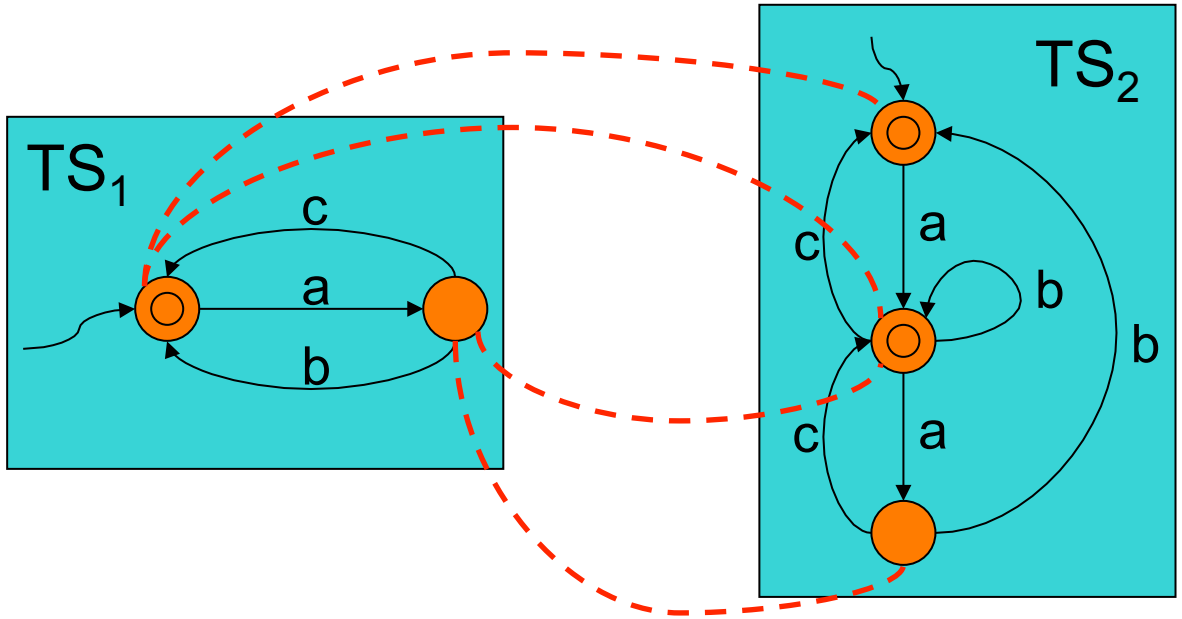
$s_1 \preccurlyeq s_2$ iff:

1. "s_1 final" implies "s_2 final"
2. For each transition $s_1 \rightarrow^a s'_1$ in TS_1, there exists a transition $s_2 \rightarrow^a s'_2$ in TS_2 s.t.

   $s'_1 \preccurlyeq s'_2$

TS_1 is *simulated* by TS_2 iff $s^0_1 \preccurlyeq s^0_2$

Simulation Relation, informally



TS_2 behaviors "include" TS_1's

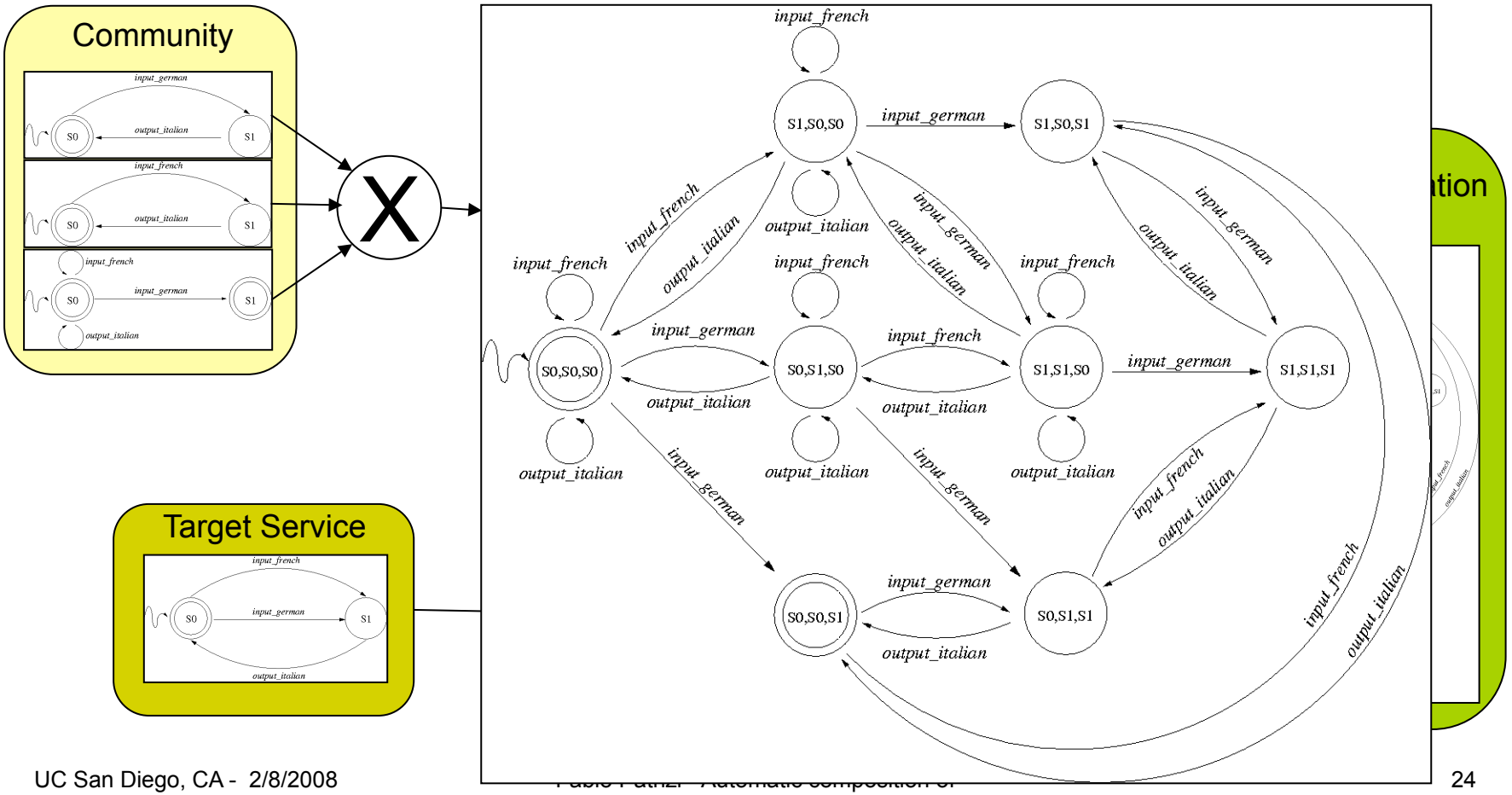
Composition via Simulation

PDL Encoding *contains* the idea of simulation.

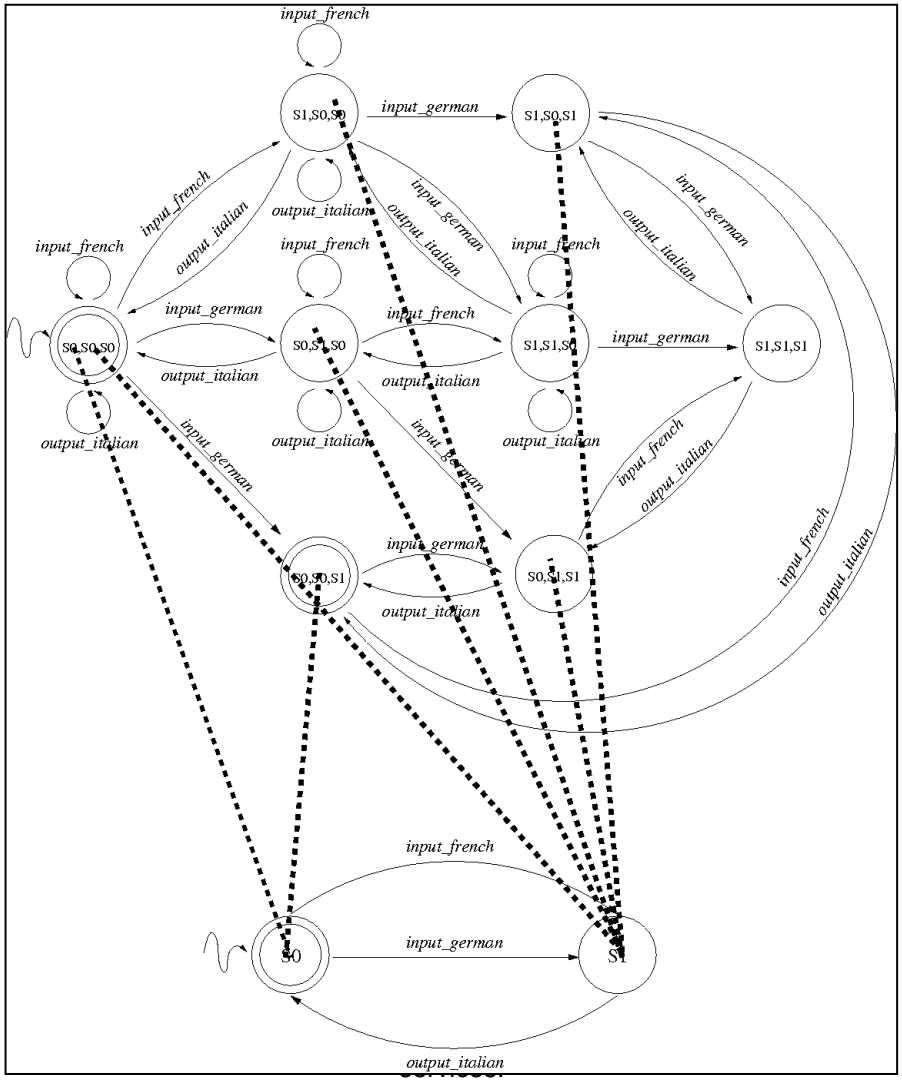
The composition problem can be reduced to search for a simulation of the target service by the available services' asynchronous product [Berardi et al., '07]

$$S_t \preccurlyeq S_1 \otimes \ldots \otimes S_n \ ?$$

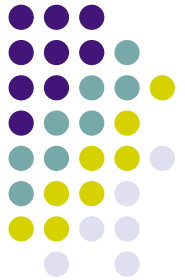
Composition via Simulation (2)



Composition via Simulation (3)



Orchestrators from Simulation



- Computing simulation is P in # of states
- # of state
  - Com

We get ALL orchestrators!

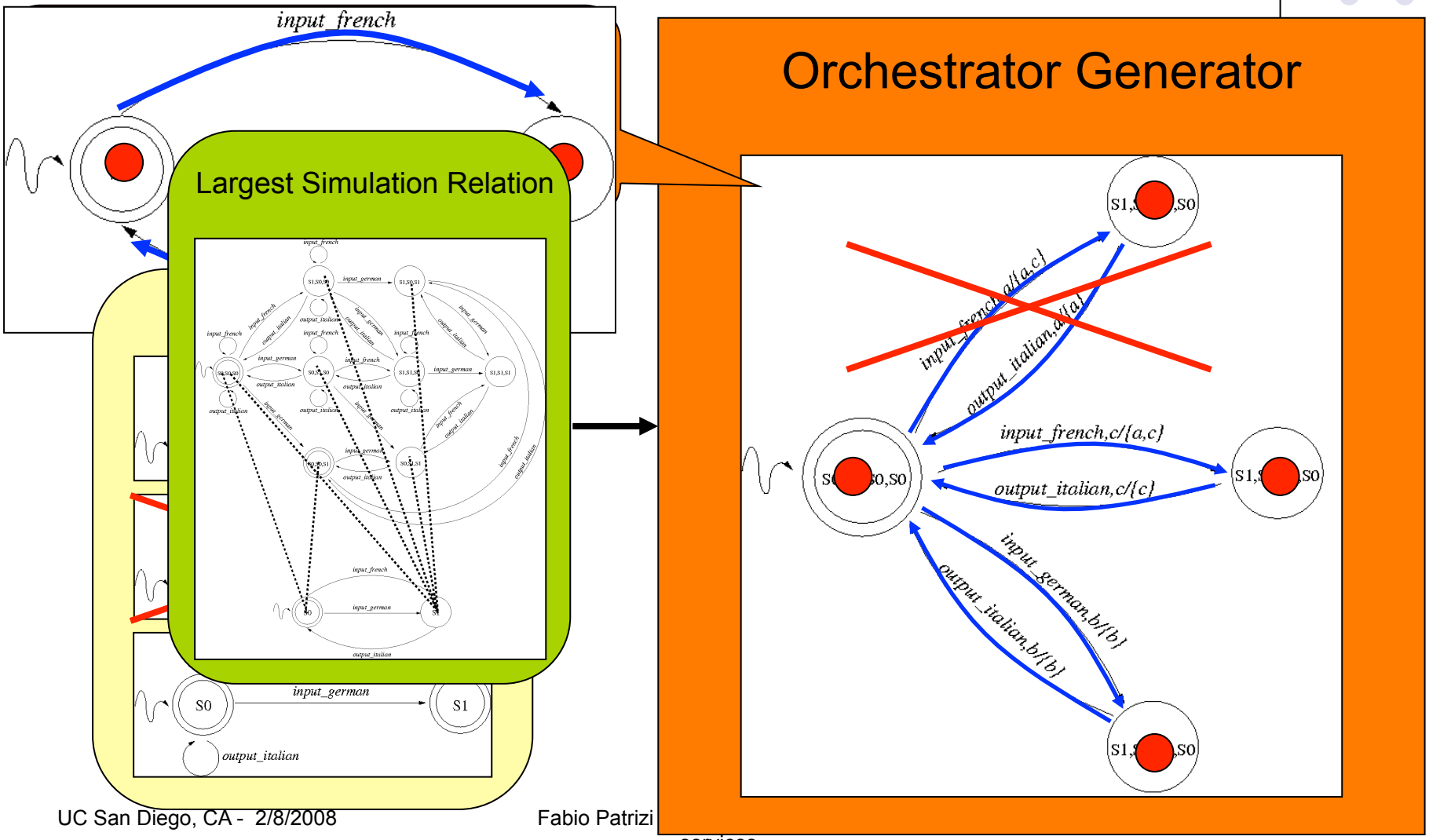
max
# of states

available
ervices

Exp

- EXPTIME, thus still optimal wrt worst-case

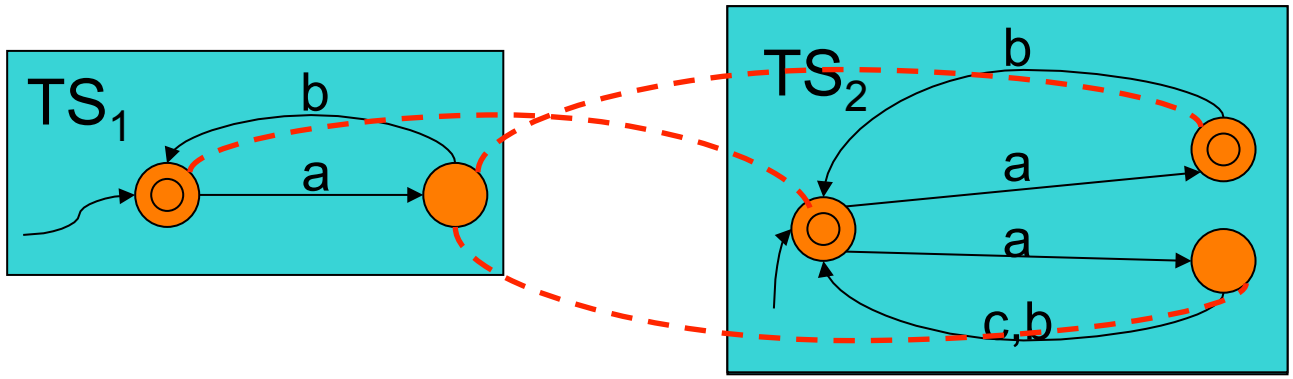
Orchestrators from Simulation (2)



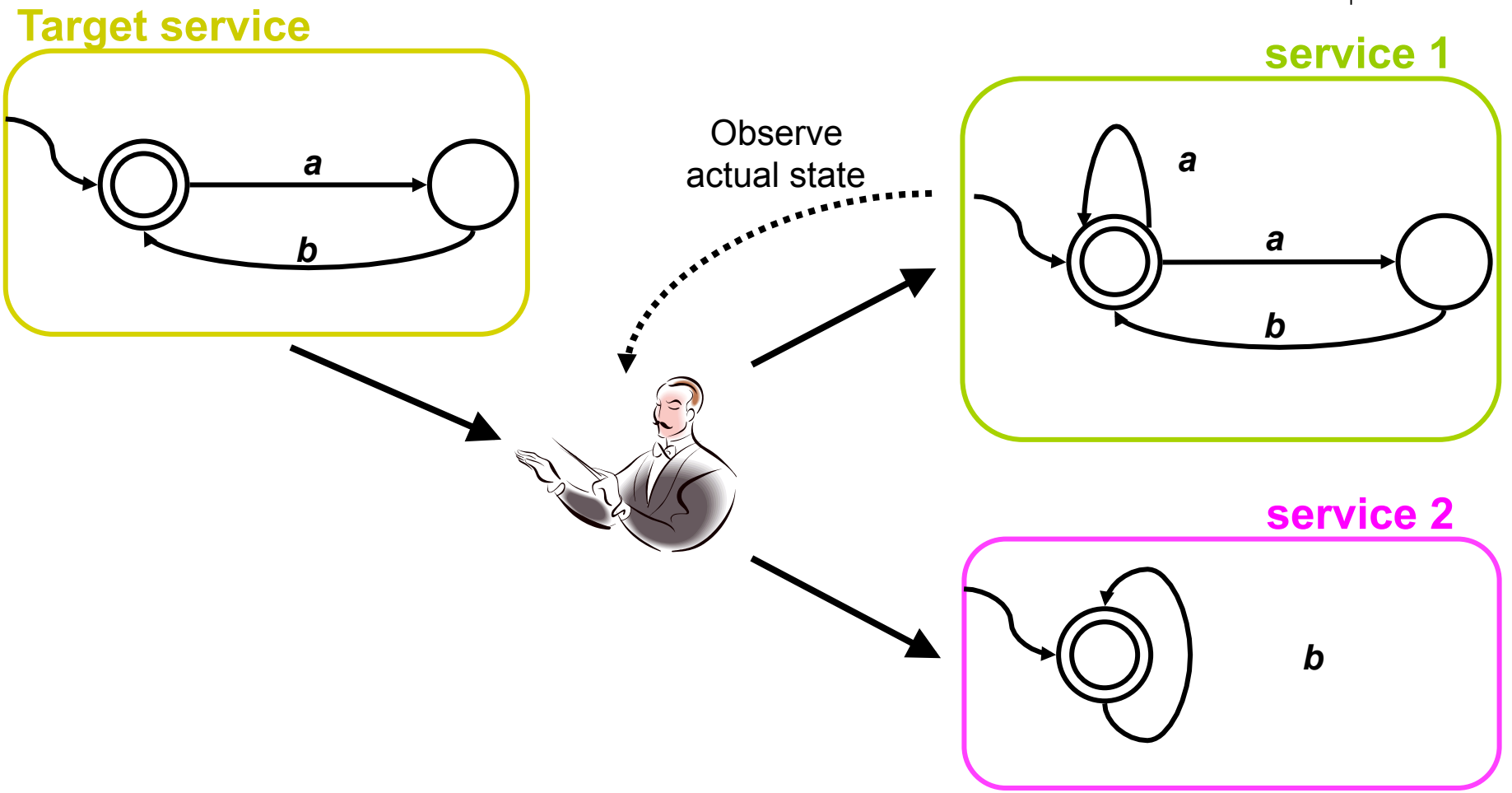
services.

Extension: ND-Simulation

- Non-det services (but det target)
  - Generalization: ND-simulation
    - Simulation preserved regardless of ND action outcomes



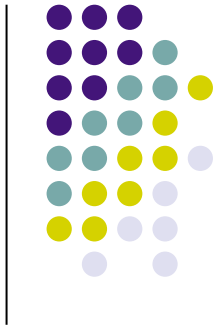
ND-Orchestrator



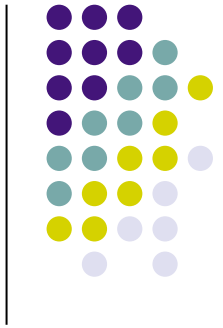
Tools for computing (ND-) orchestrators

- Effective techniques & synthesis tools developed by the verification community:
  - TLV [Pnueli & Shahar 96]
  - Based on symbolic OBDD representation
  - Conceptually based on simulation technique

Application Scenarios



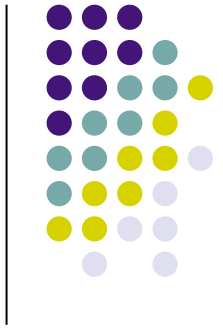
- Web service composition[Berardi et al., ‘07]
  - An implementation from BPEL specifications @ DIS
- Distributed agents in a common environment, with failures(work in preparation)



“Unfortunately”…

… many services deal with data …

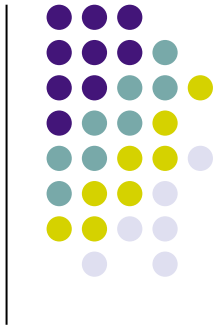
Dealing with data



Examples:

- Agents need to exchange messages (e.g., position, battery level,…)
- Web services take input messages (e.g., users subscribing a service) and return output messages (e.g., pricelist)

Dealing with data (2)



REMARK:

Infinitely many messages

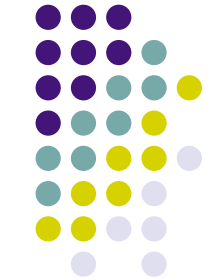
may give raise to infinitely many states

PROBLEM:

Finite-state property no longer holds

We expect to get undecidability

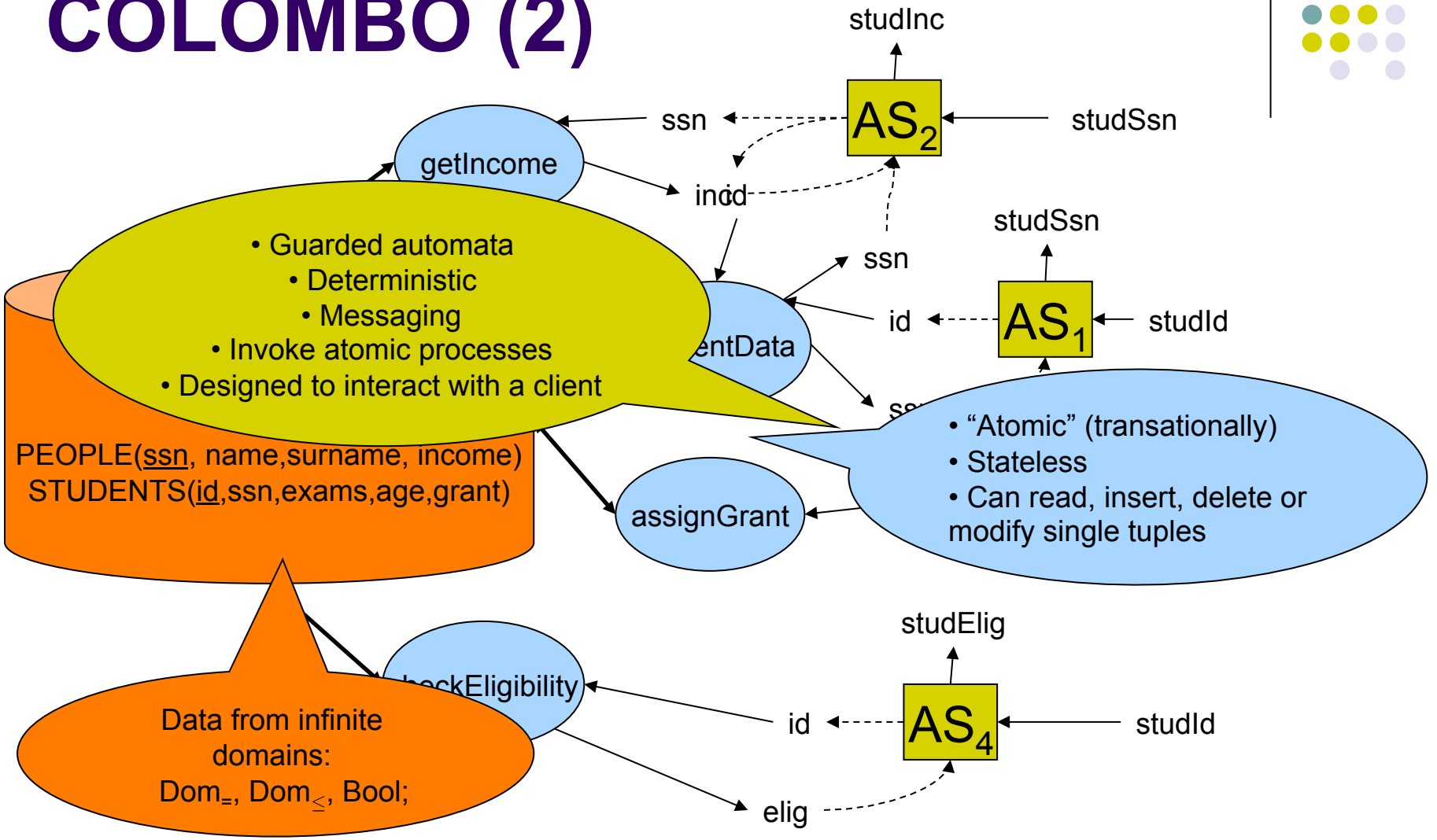
COLOMBO[Berardi & al., VLDB'05]



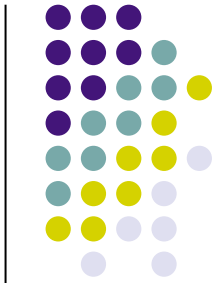
A general framework for web services with messages

- Basic results in data-aware composition
- Asynchronous, Deterministic, finite-state services with messaging
- Messages from infinite domains
- (Key-based) Access to a database through atomic processes *(i.e., parametric actions)*

COLOMBO (2)


studInc
AS2
ssn
studSsn
getIncome
incId
• Guarded automata
• Deterministic
• Messaging
• Invoke atomic processes
• Designed to interact with a client
studSsn
ssn
id
AS1
studId
entData
• "Atomic" (transationally)
• Stateless
• Can read, insert, delete or modify single tuples
PEOPLE(ssn, name,surname, income)
STUDENTS(id,ssn,exams,age,grant)
assignGrant
studElig
checkEligibility
id
AS4
studId
Data from infinite domains:
Dom=, Dom≤, Bool;
elig

Atomic processes



```
getIncome
  I:ssn; O:inc
Effects:
  inc := PEOPLE_3(ssn)
```

Interface specification

```
checkEligibility
  I:id; O:eligibility
Effects:
  if (STUDENT_4(id) == true)
    then eligibility := true
    else eligibility := false
```

Conditional effects

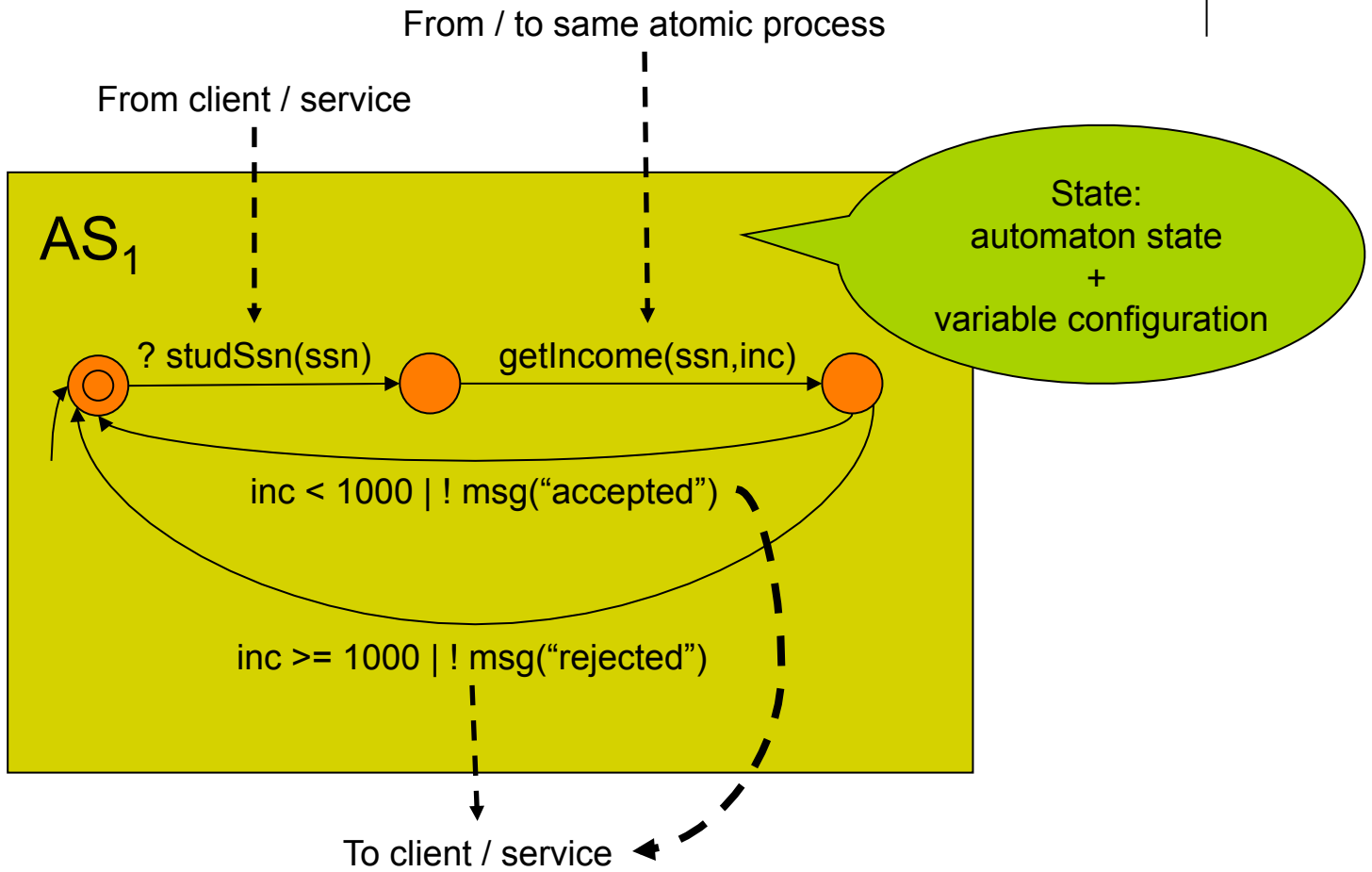
- over local variables / accessed values

```
assignGrant
  I:id
Effects:
  either
    modify STUDENT_4(id, false)
  or
    no-op
```

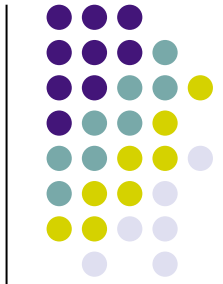
Nondeterministic effects (Finite branching)

- due to incomplete abstract model

Available services



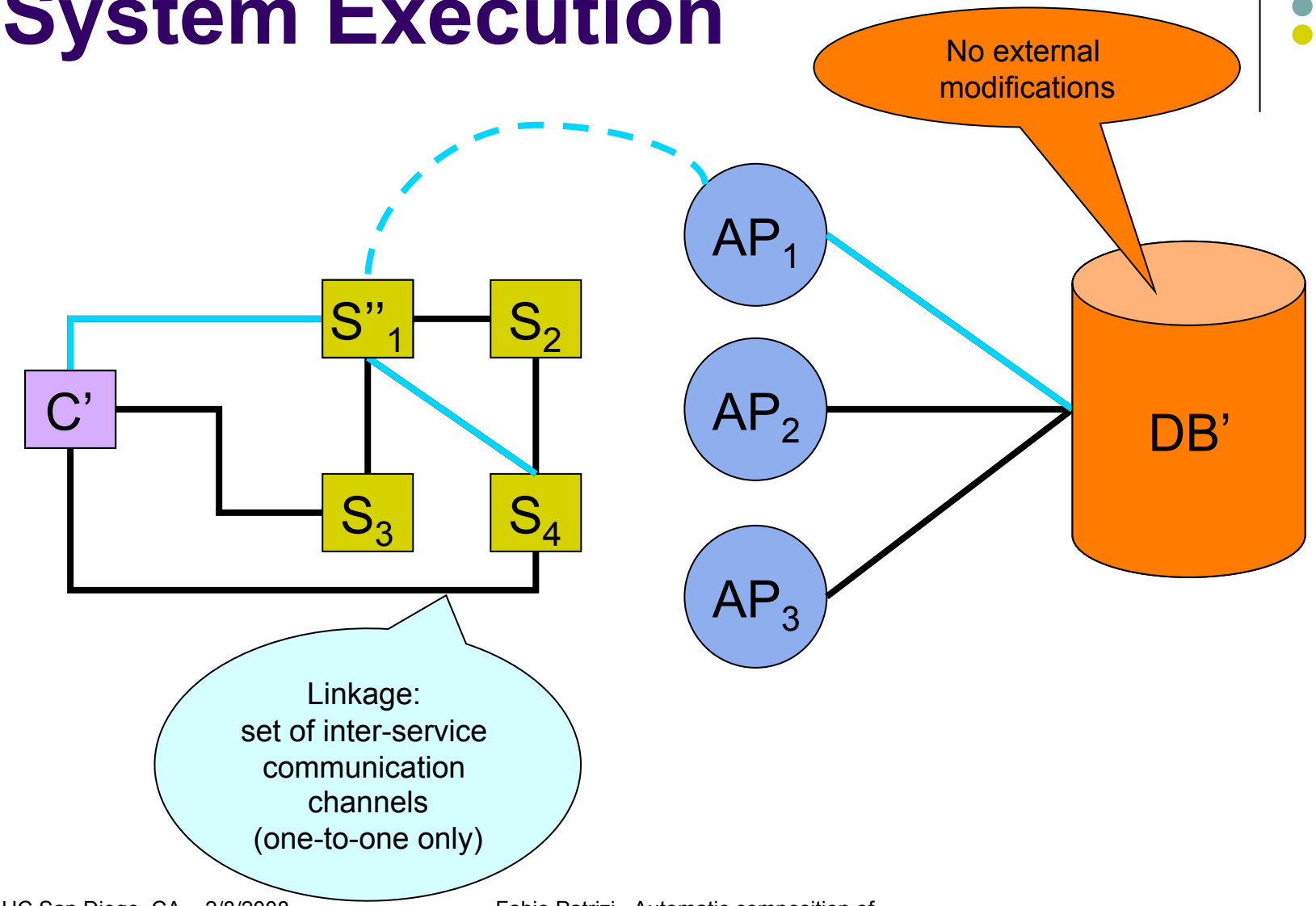
Synchronization



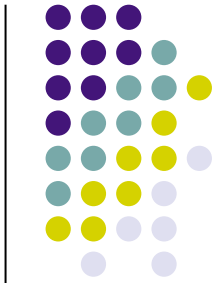
1. Wait for incoming messages (length-1 queues)
2. Execute a fragment of computation
3. After sending a message, either:
   - Terminate (in a final state) or
   - Go to 1.

- Client starts by sending a message
- Available services wait

System Execution



Execution Tree



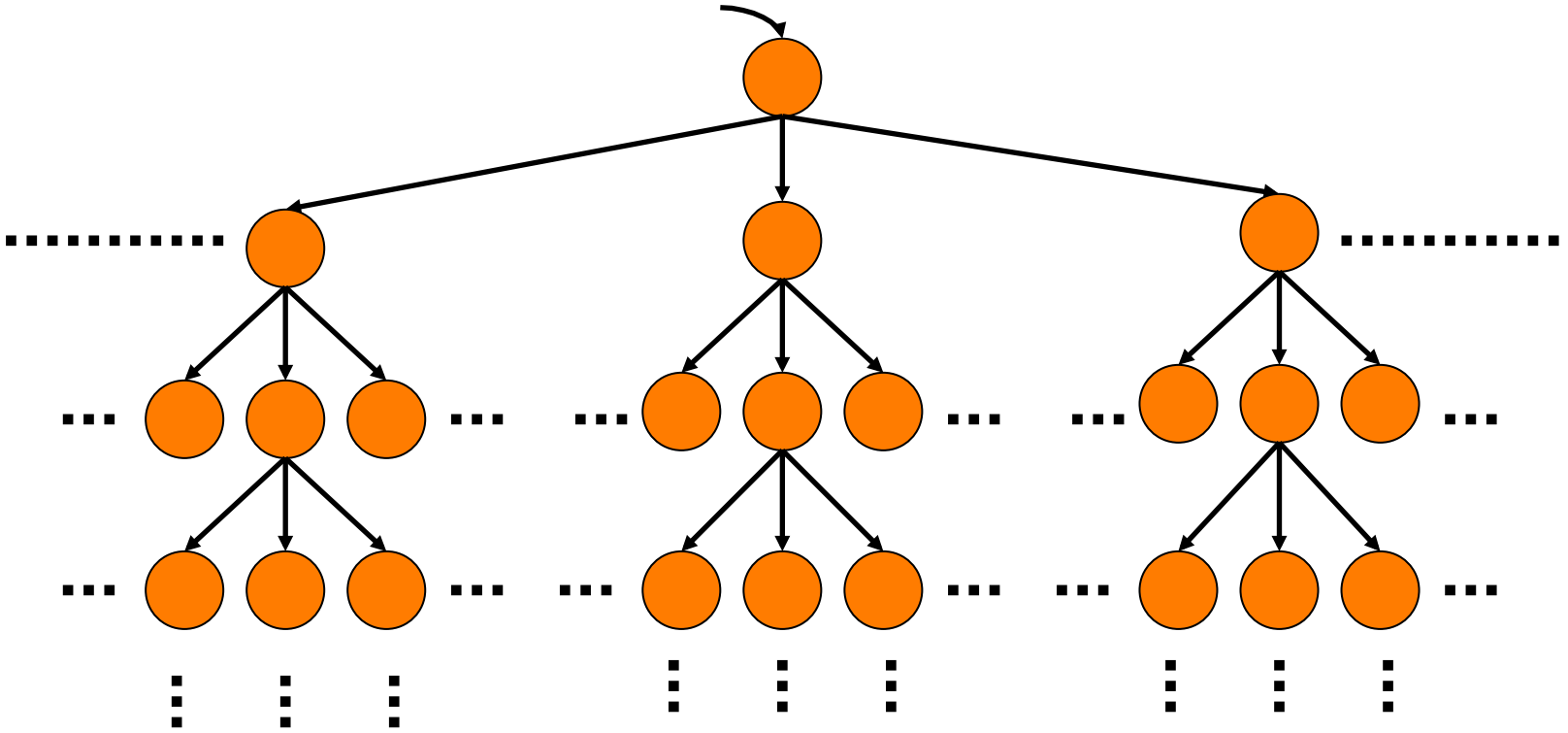
A system:

$$\mathcal{S}=\langle\, C, \{\, S_1,\ldots,S_n \,\}, \mathcal{L} \,\rangle$$

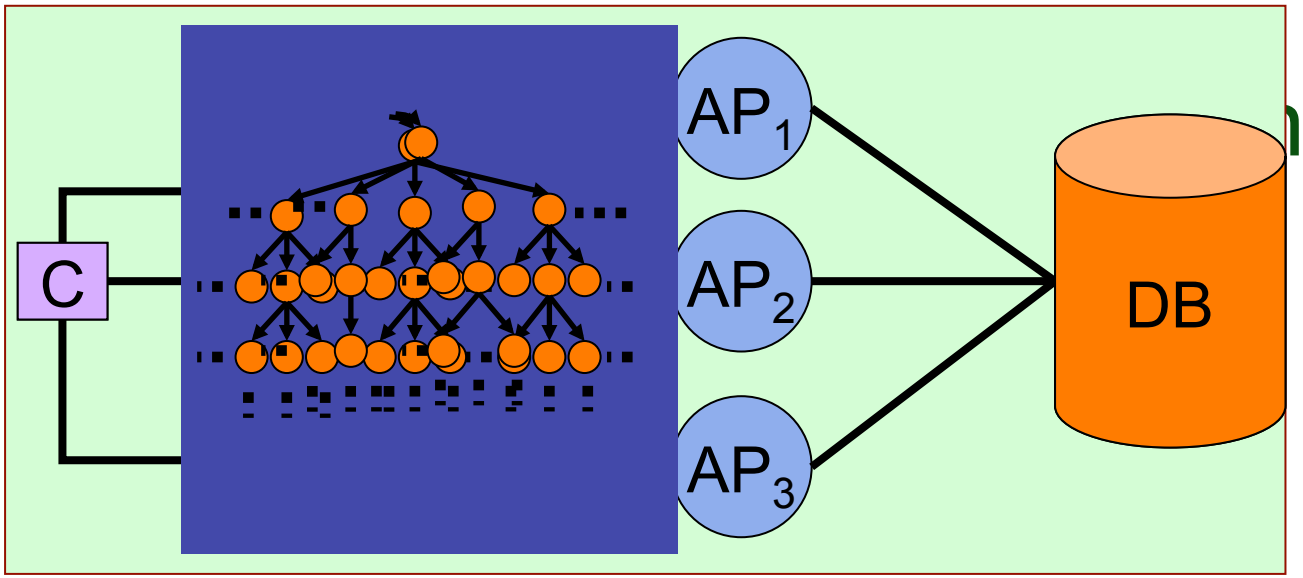
Infinite tree evolution:

- Nodes are snapshots of service + DB states
- Edges are labeled by:
  - Ground messages
  - Process invocations
  - DB states (pre / post transition)

Execution Tree (2)



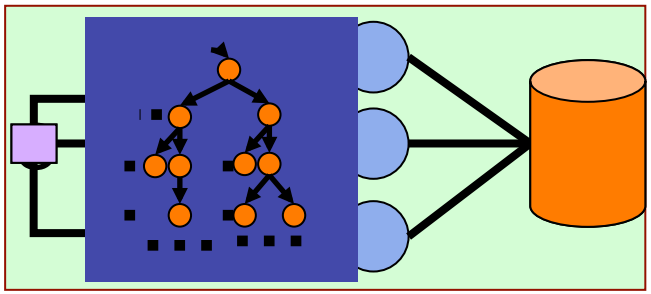
Execution Trees' "Essence"

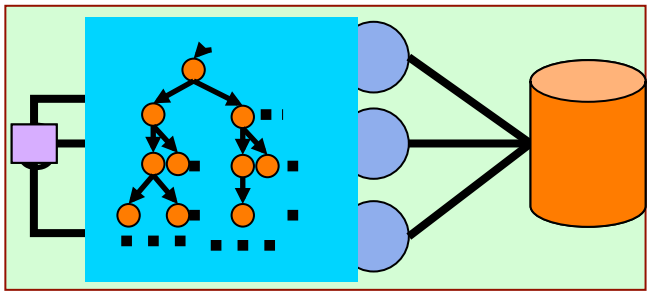


nto:

REMARK: internal messages collapse!

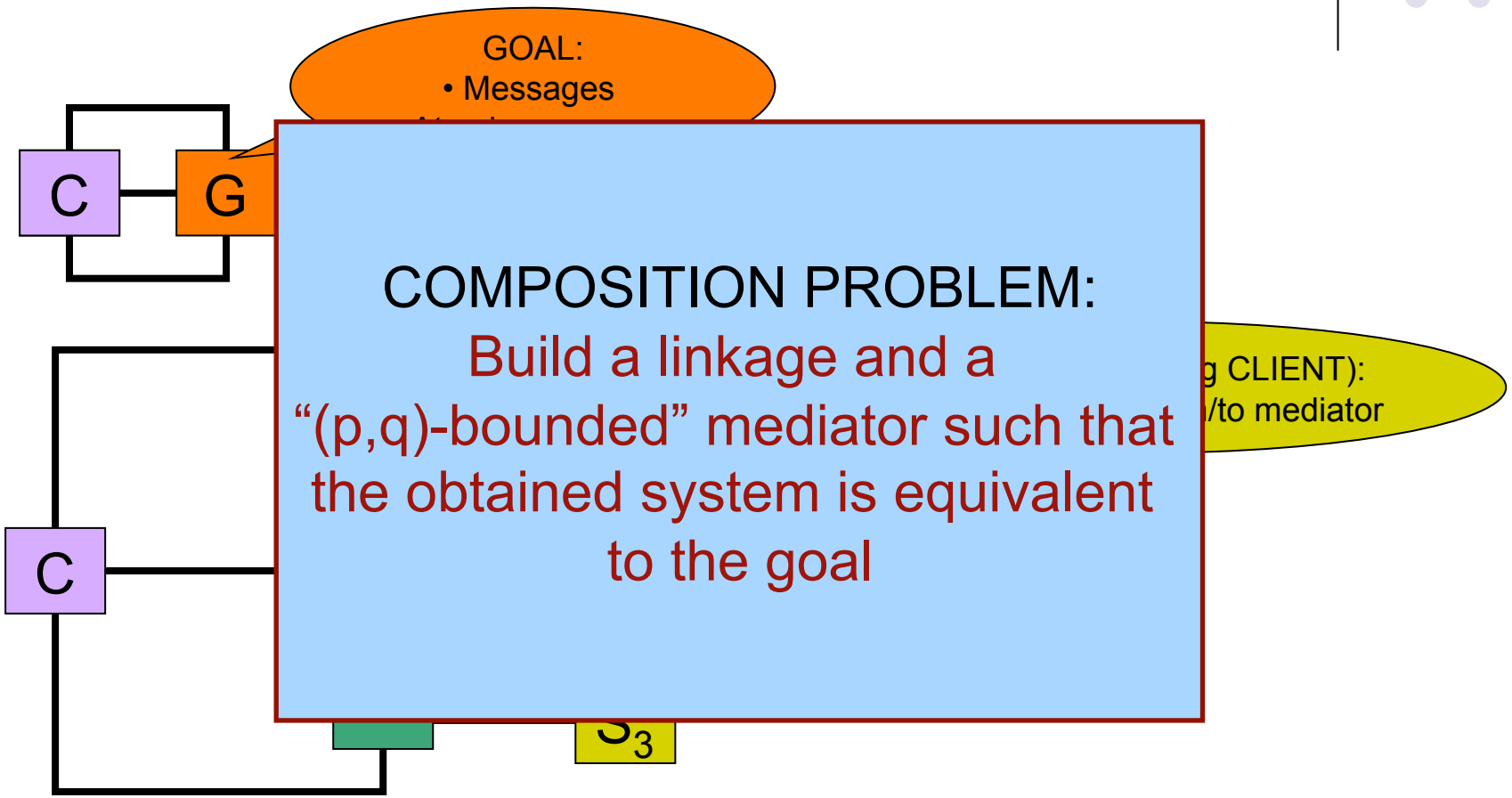
System Equivalence



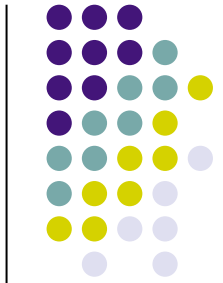


Two systems are equivalent iff they have isomorphic essences!
(Equivalent in terms of what is observable)

The Composition Problem in COLOMBO

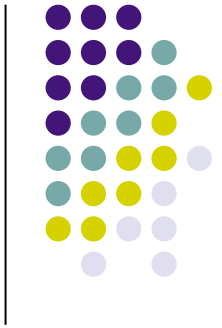


Solving the Composition Problem in COLOMBO



- IDEA
  - Reduce to the finite case
- OBSTACLES:
  - Infinite messages and initial DB yield infinite properties (e.g., send-ground-message)
- RESTRICTIONS needed

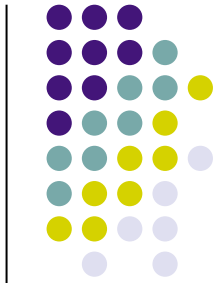
Restrictions



- Bounded # of new values introduced by the client (wrt to initial DB state)
- Bounded # of DB lookups, depending on # of new values the client introduces

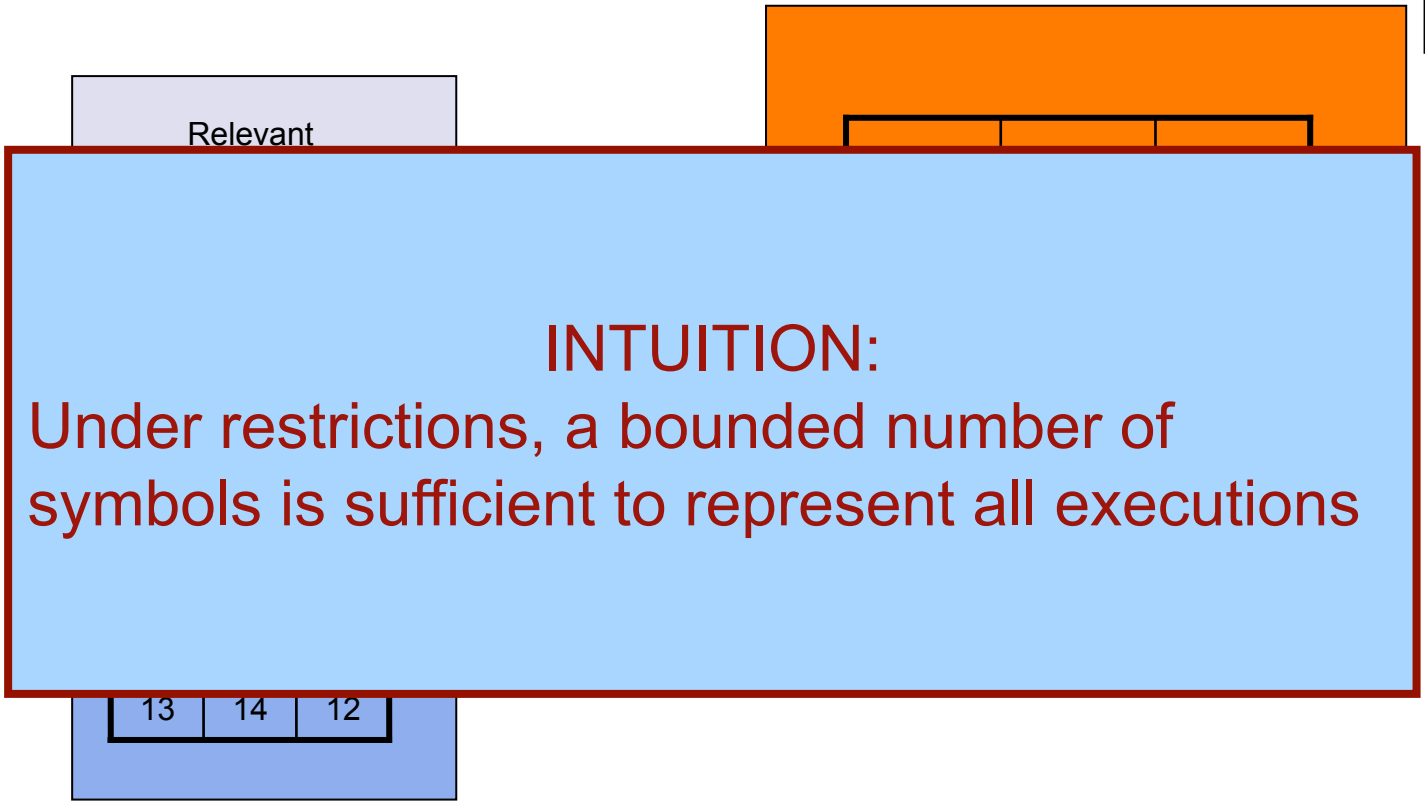
- REMARK: number of new values are finite, actual values still infinite

Symbolic representation

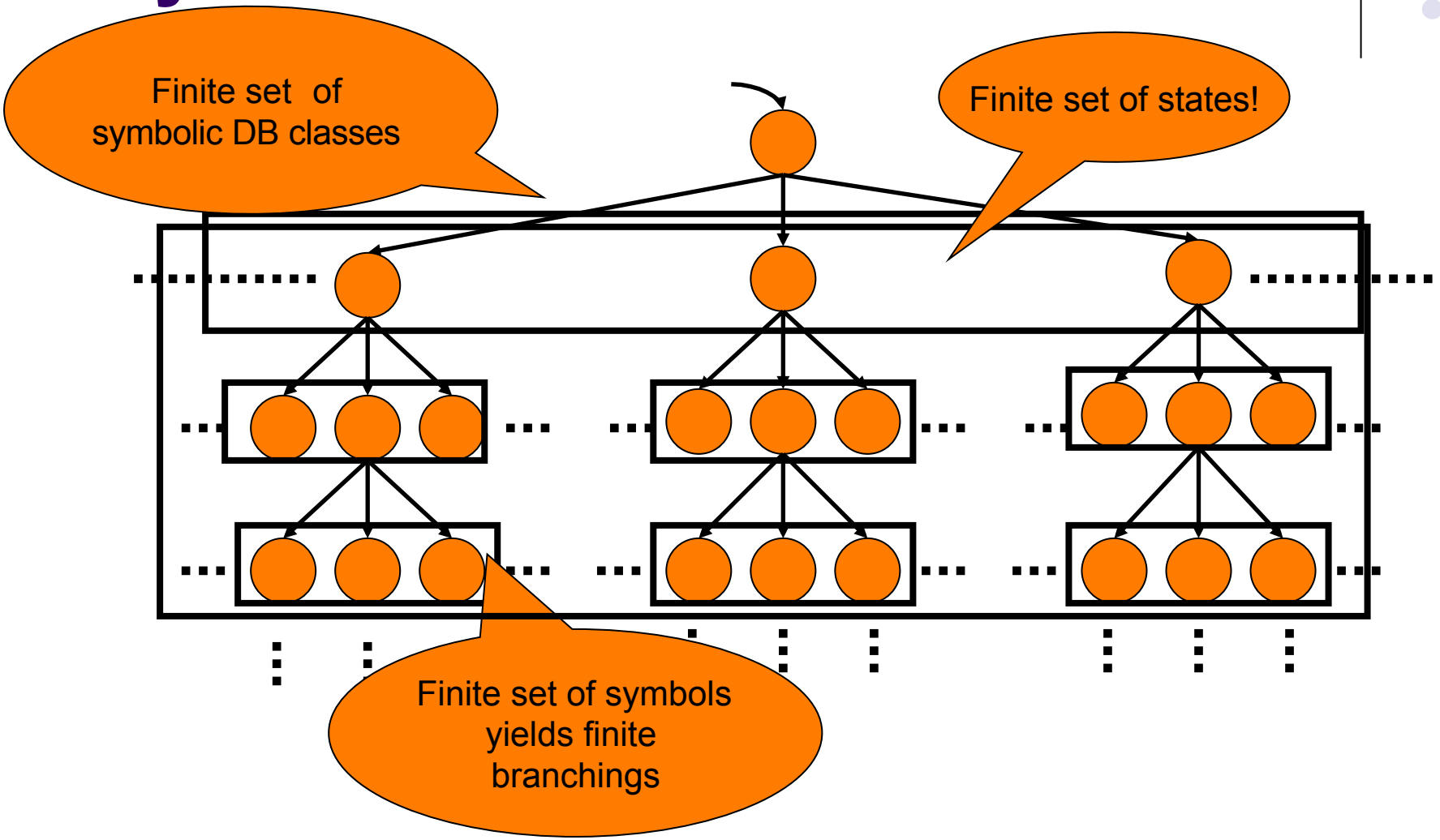


- Values are referred to by symbols
- Relevant features of symbols
  - Relationships with
    - All other symbols (wrt $\leq$, =)
    - Constants occurring in guards

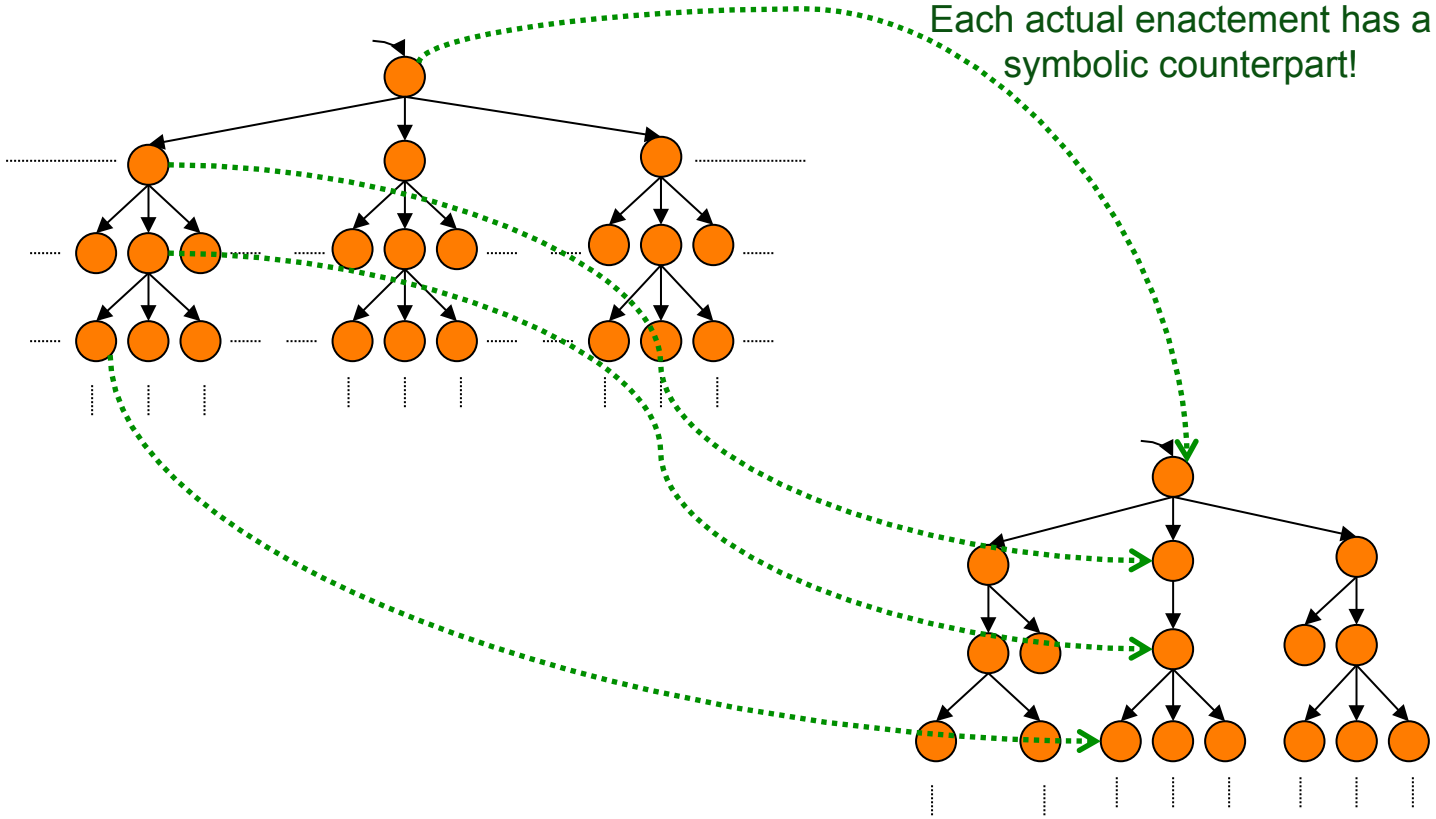
Symbolic Value Characterization



Symbolic execution tree



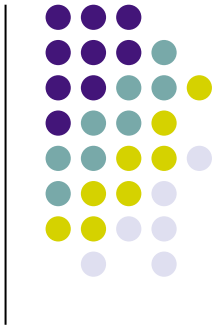
From Infinite to Finite



Solution Technique & Issues

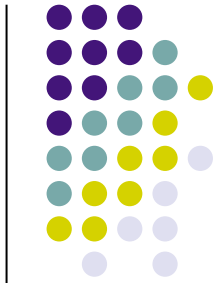
- (p,q)-bounded mediator:
  - At most p states and q variables
- Reduction to PDL-Sat, with underconstrained variables
  - To be guessed
  - Represent existence of links and mediator behavior
- Upper bound double-EXPTIME in p,q, size of target and community services:
  - Expect to get rid of p,q
    - Derivable from target and available services' structure?
  - Complexity can be refined with a more efficient encoding?

Conclusion & Future Directions



- Good understanding of “behavioral” composition:
  - Optimal technique for deterministic scenarios
  - Ongoing extension to nondeterministic contexts w/ failures
- Starting point for data-aware services:
  - General framework and first results, but severe restrictions
    - Relax key-based access assumption?
    - Remove, or derive, mediator bounds?
    - Investigate over decidability bounds
  - Flexible solutions
    - PDL technique returns only one solution, what about simulation?
  - Reasoning about infinite state systems
    - Abstraction (cf., e.g., [Pnueli & al, VMCAI 05], [Kesten&Pnueli, 00])

Thanks For Your Attention!



- Questions?