

**X  
M  
L  
M  
L**

# Formal Methods For Web Service Process Modeling

Dr. Yuhong Yan  
NRC-IIT-Fredericton  
Internet Logic

**OWL**

**RuleML**

**jDREW**

**UDDI**

**WSDL**

**SOAP**

**BPEL**



# Web Service Process Description Languages

---

- Orchestration language: BPEL4WS
- Choreograph languages: WS-CDL, WSCI
- Formal models: Process Algebras, Petri nets, Automata

# Modeling and Reasoning about Web Service Processes

---

- How to model the control flow and data flow
- What are the reasoning tasks
- What are the techniques for these tasks

# Web Service Process vs. Workflow Management

---

- Workflow management
  - Hybrid system of human and software applications
  - Collaborative working environment
  - Workitems-actors
  - Process descriptions similar to orchestration languages
- Web services
  - Automated services
  - Interoperation and service oriented architecture
  - request-response
  - Web service process engine can be similar to workflow management engine (cf. Karagiannis 2006)

# Business Process Execution Language for Web Service

---

**BPEL4WS**

**BPEL4WS**

# Business Process Execution Language for Web Service (BPEL4WS)

---

The information described by BPEL4WS:

- The **execution order** of the **activities**
- The **triggering conditions** of the **activities**
- The **partners** for the external activities
- The **composition** of Web Services
- The **binding** to WSDL (activities <-> operations)

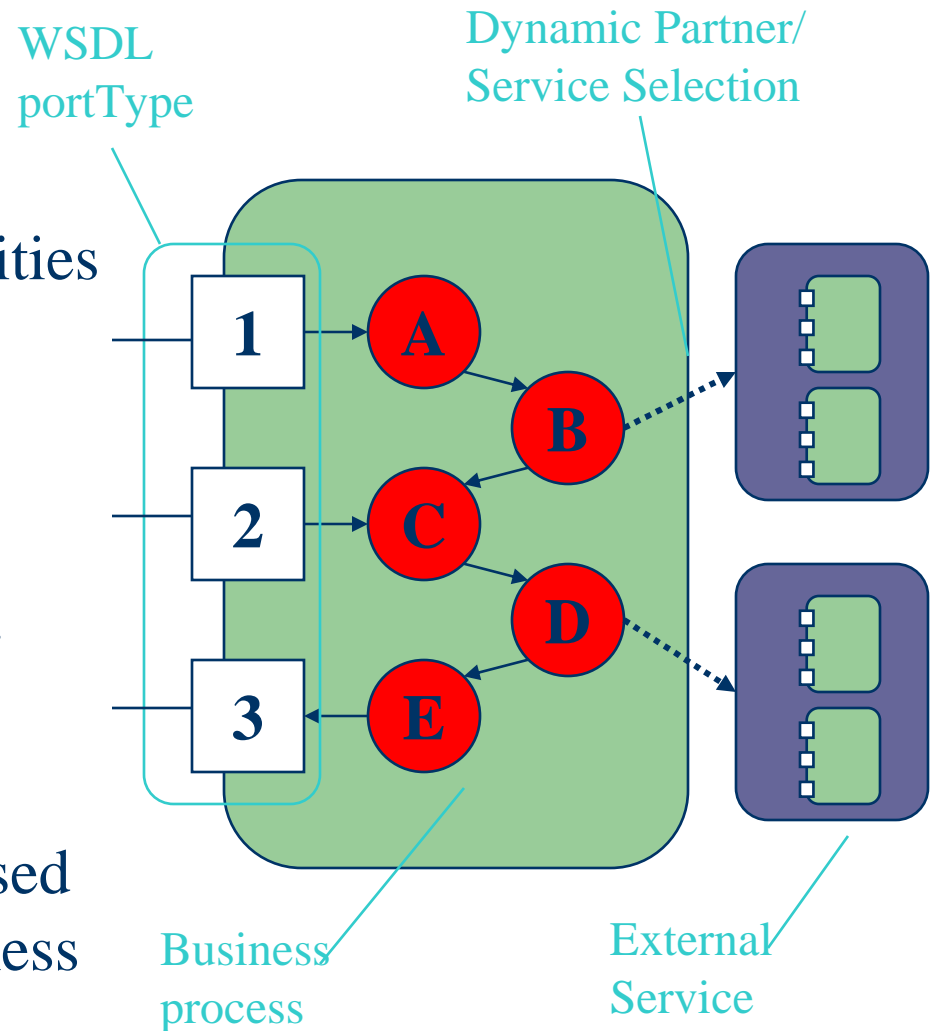
# Business Process (**what**) versus WSDL (**how**)

## Business Process: **what** to do

- Modeled as a sequence of activities
- Tools aid to define, monitor, and manage business processes

## WSDL: **how** to execute activities

- An activity can be an internal or external Web service (SOAP/WSDL)
- A business process can be exposed for consumption by another business process or a client app



# How BPEL looks like?

```
<process name="echoString"
  targetNamespace="urn:echo:echoService"
  xmlns:tns="urn:echo:echoService"
  xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/">

  <partnerLinks>
    <partnerLink name="caller"
      partnerLinkType="tns:echoPLT"
      myRole="service"/>
  </partnerLinks>

  <variables>
    <variable name="request" messageType="tns:StringMessageType"/>
  </variables>

  <sequence name="EchoSequence">
    <receive partnerLink="caller" portType="tns:echoPT"
      operation="echo" variable="request"
      createInstance="yes" name="EchoReceive"/>
    <reply partnerLink="caller" portType="tns:echoPT"
      operation="echo" variable="request" name="EchoReply"/>
  </sequence>

</process>
```

# BPEL activities

---

- Basic Activities: atomic actions
  - <receive>
  - <reply>
  - <invoke>
  - <assign>
  - <throw>
  - <terminate>
  - <wait>
  - <empty>

# BPEL Structured Activities

---

- **<sequence>**: an ordered sequence of activities
- **<flow>**: parallel activities
- **<switch>**: “case-statement” approach
- **<while>**: loop
- **<pick>**: execute one of several alternative paths

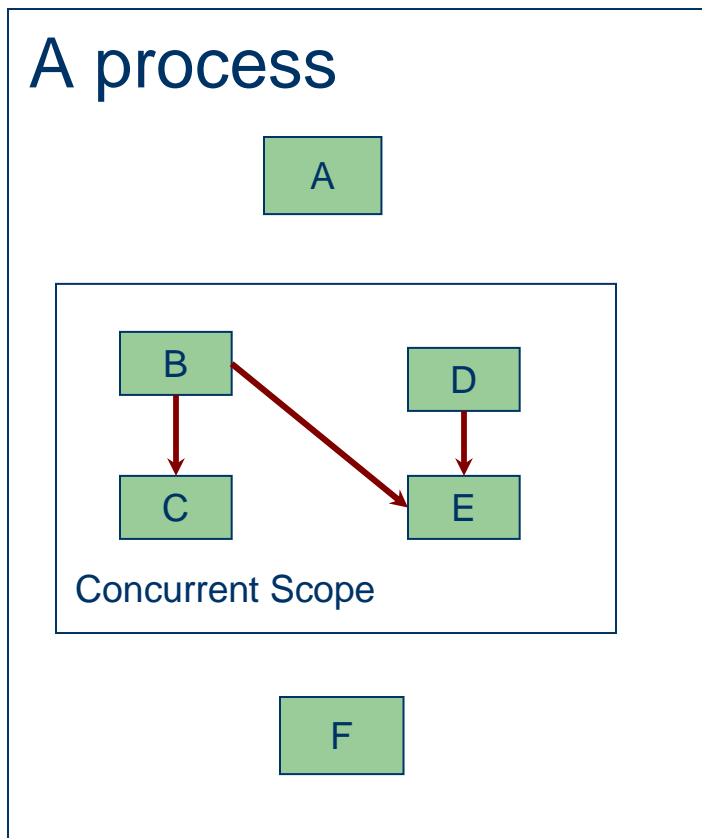
## <faultHandlers>

---

- In response to faults
- Defines the recovery actions when faults occur
- To undo the partial and unsuccessful work of a scope in which a fault has occurred
- To do other business logic

# Control the execution order using <link>

<link> defines dependencies between the activities in BPEL



Execution orders:

A-B-C-D-E-F

A-B-D-C-E-F

A-D-B-C-E-F

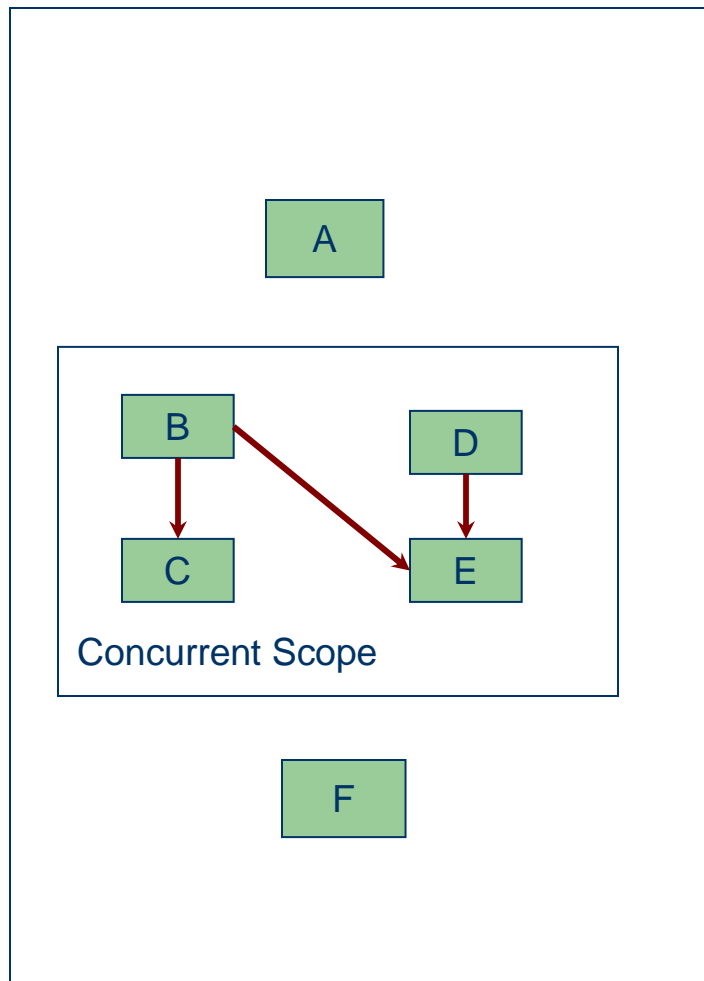
A-D-B-E-C-F

## Data flow: <variable>

---

- Data variables used by activities
  - <variable messageType=“...”>: WSDL message;
  - <variable type=“...”>: XML Schema simple type;
  - <variable element=“...”>: XML Schema element.
- Variables associated with message types can be specified as input or output variables for **invoke**, **receive** and **reply** activities.

# A Business Process and BPEL Description



```
<sequence >
  <receive a v ><corr. >p <\corr. ><\receive >
  <flow >
    <invoke b b_in b_out >
      <source linkname=b_to_c >
      <source linkname=b_to_e >
    <\invoke >
    <invoke c c_in c_out >
      <target linkname=b_to_c >
    <\invoke >
    <invoke d d_in d_out >
      <source linkname=d_to_e >
    <\invoke >
    <invoke e e_in e_out >
      <target linkname=b_to_e >
      <target linkname=d_to_e >
    <\invoke >
  <\flow >
  <reply f w><corr. >p <\corr. ><\reply >
<\sequence >
```

# Choreography Languages

---

**Choreography**

Choreography

# Web Service Choreography Interface (WSCI)

---

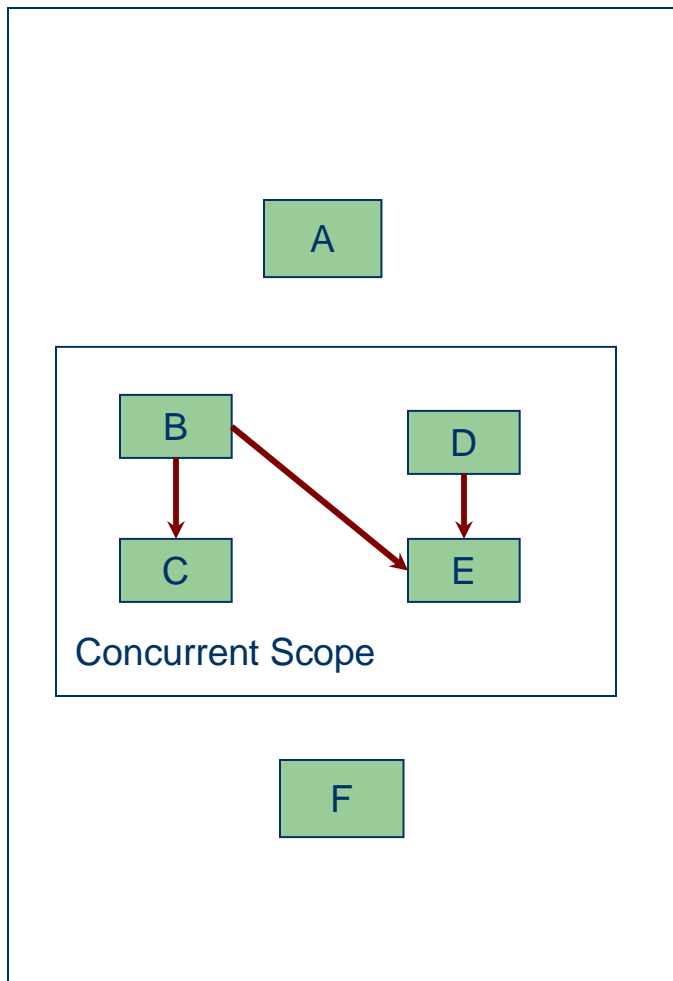
- Describes the flow of messages
- The observable behaviors of a Web service
- No internal behaviors
- Compared with WSDL
  - WSDL does not describe the execution orders of operations
  - WSDL does not describe the correlation aspects

# Web Service Choreography Interface (WSCI)

- Atomic Activities
  - `<action>` :mapping to an operation in WSDL
  - `<delay>`
  - `<empty>`
  - `<fault>`
  - `<call>`
  - `<spawn>`
  - `<join>`
- Complex Activities
  - `<process>`
  - `<all>`: parallel
  - `<choice>`
  - `<foreach>`
  - `<sequence>`
  - `<switch>`
  - `<until>`
  - `<while>`
- Variables: map to SOAP messages similar like BPEL

} For the instances of a  
<process>

# A Business Process and WSCI Description



```
<process >  
  <sequence >  
    <action a v ><corr. >p <\/corr. ><\/action >  
    <action f w ><corr. >p <\/corr. ><\/action >  
  <\/sequence >  
<\/process >
```

# Web Service Choreography Description Language (WS-CDL)

---

- A global conversation among services
  - WSCI is a projection of the entire conversation on a participant
- Observable behaviors
- Based on Pi-Calculus
- Elements
  - Channels
  - Three structures: sequence, parallel, and choice
  - Variables

# ebXML Business Process Specification Schema (ebXML BPSS)

---

- A choreography language
- Between two business partners
- Contains more information than Web service process description languages
  - Business documents
  - Business transactions  
(protocol to exchange documents)
  - Binary collaborations  
(composition of transactions)
  - Multiparty collaborations  
(2 or more binary collaborations)
  - Substitution sets

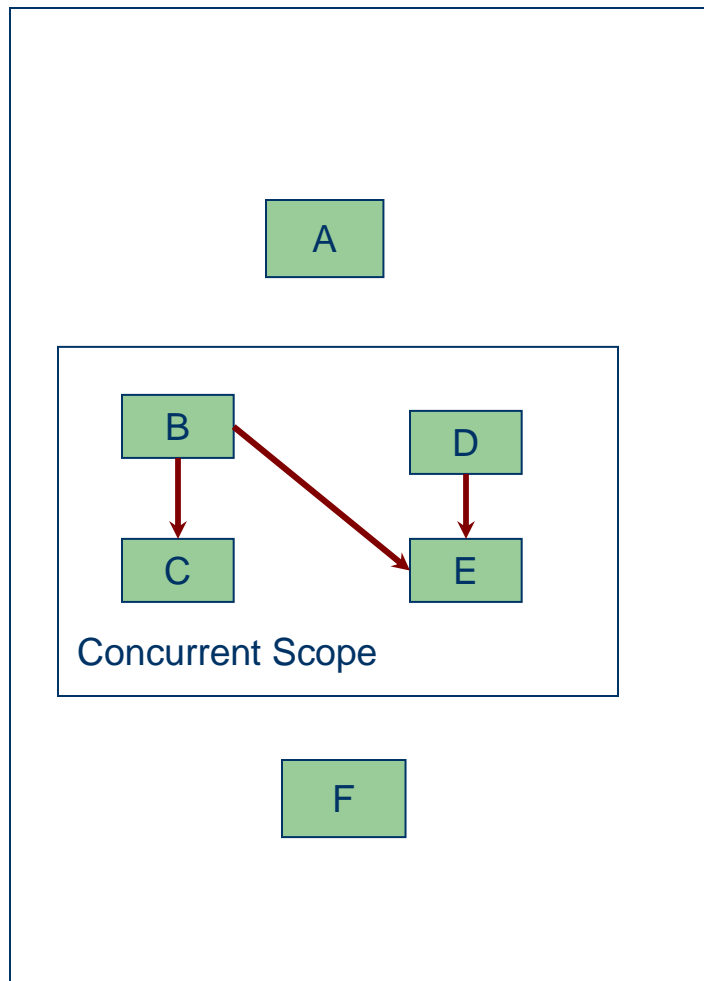
# Formal Models

---

**Formal Models**

Formal Models

# A Business Process and BPEL Description

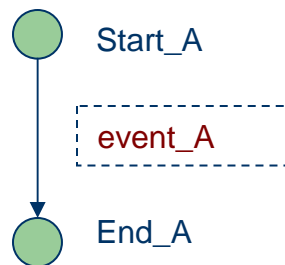


```
<sequence >
  <receive a v ><corr. >p <\corr. ><\receive >
  <flow >
    <invoke b b_in b_out >
      <source linkname=b_to_c >
      <source linkname= b_to_e >
    <\invoke >
    <invoke c c_in c_out >
      <target linkname=b_to_c >
    <\invoke >
    <invoke d d_in d_out >
      <source linkname=d_to_e >
    <\invoke >
    <invoke e e_in e_out >
      <target linkname=b_to_e >
      <target linkname= d_to_e >
    <\invoke >
  <\flow >
  <reply f w><corr. >p <\corr. ><\reply >
<\sequence >
```

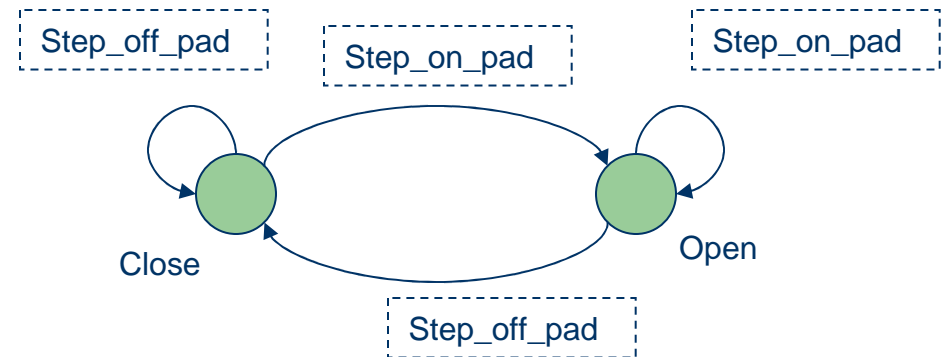
# Automata

A finite automaton  $\Gamma$  is a 5-tuple  $\Gamma = (X, \Sigma, T, I, F)$ , where:

- ★  $X$  is a finite set of states;
- ★  $\Sigma$  is a finite set of events;
- ★  $T \subseteq X \times \Sigma \times X$  is a finite set of transitions;
- ★  $I \subseteq X$  is a finite set of initial states;
- ★  $F \subseteq X$  is a finite set of final states.

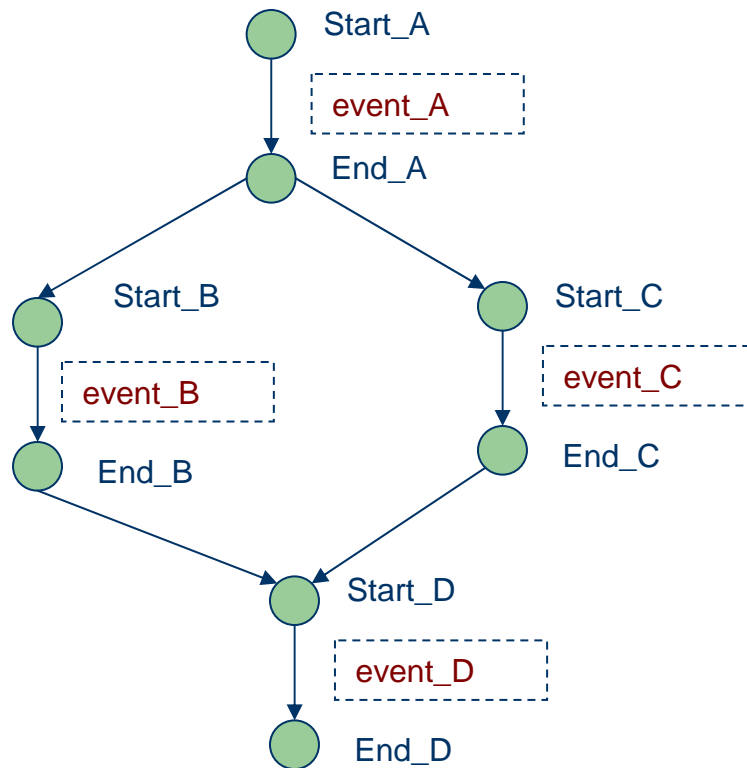


A Web Service Activity



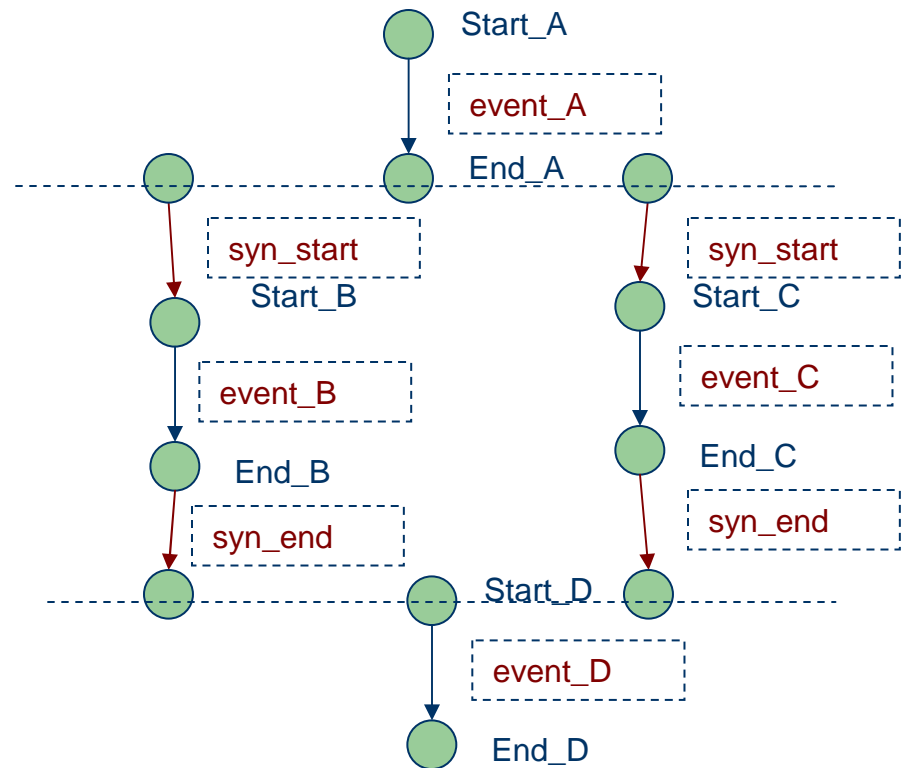
An Automatic Door

# Automata



Choose between B and C:

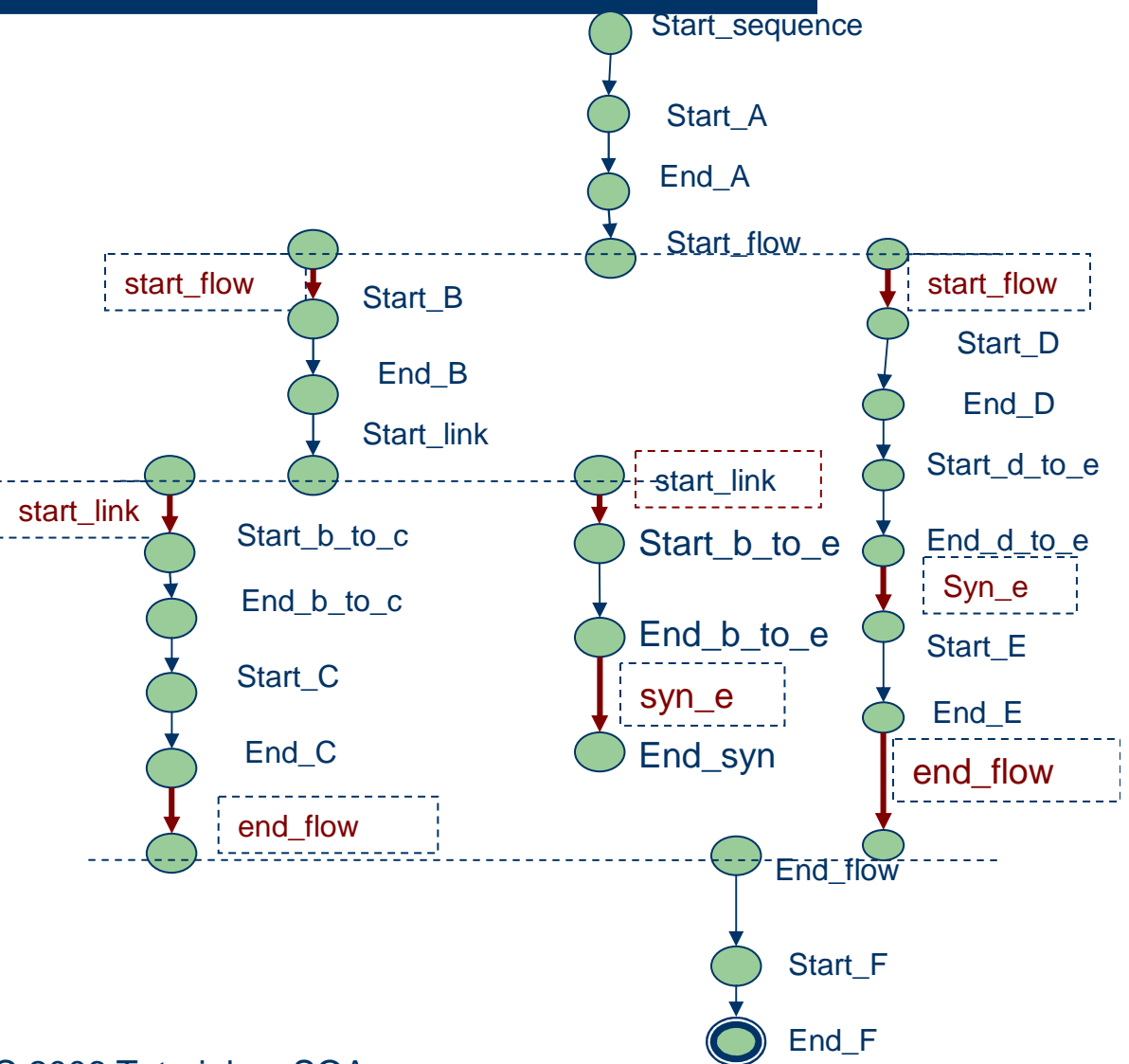
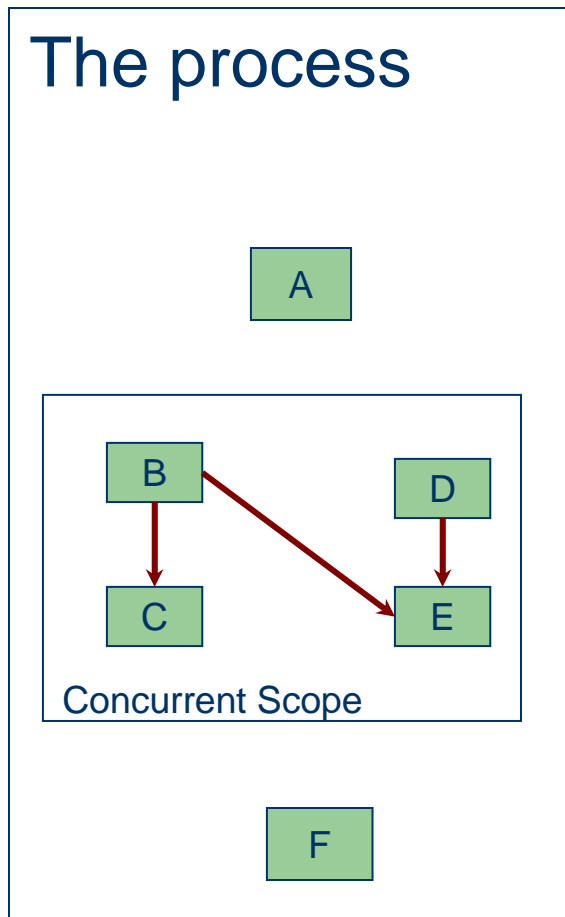
event\_A - event\_B - event\_D or  
 event\_A - event\_C - event\_D



Parallel B and C:

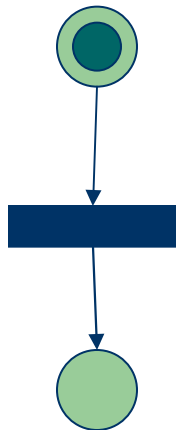
event\_A - syn\_start - event\_B - event\_C - syn\_end - event\_D or  
 event\_A - syn\_start - event\_C - event\_B - syn\_end - event\_D

# The Automata for the Example Process

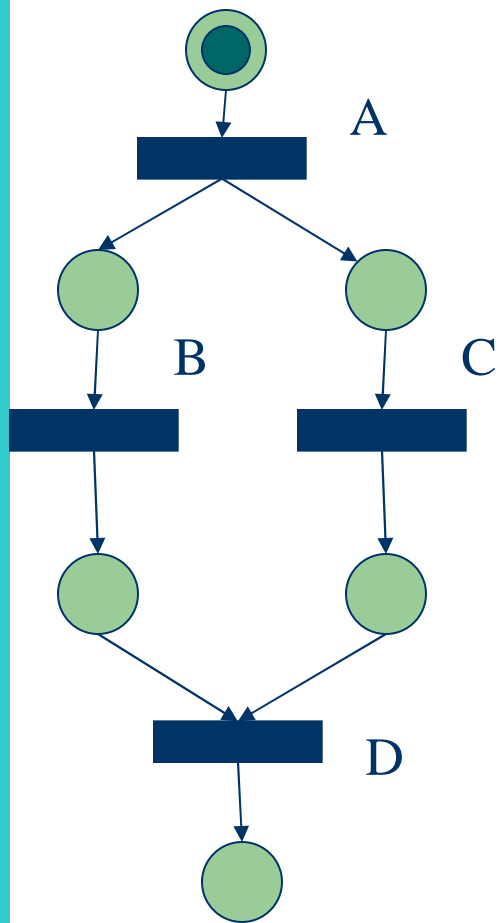


# Petri nets

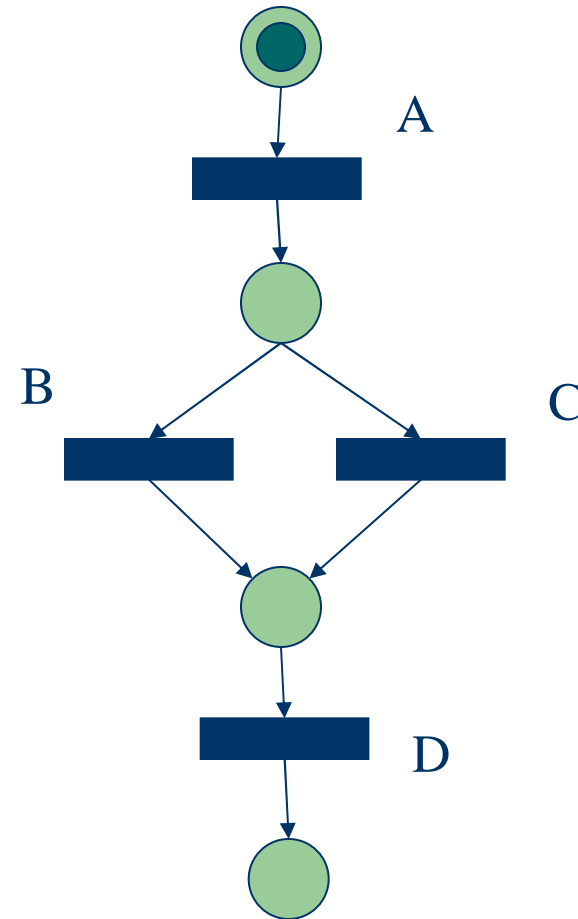
- A Petri Net (P/T Net) is a triple  $(P, T, F)$ , where
  - $P$  is a finite set of places,
  - $T$  is a finite set of transitions,  $(P \cap T = \emptyset)$
  - $F \subseteq (P \times T) \cup (T \times P)$  is a finite set of arcs, the flow relation.



# Petri Nets Blocks



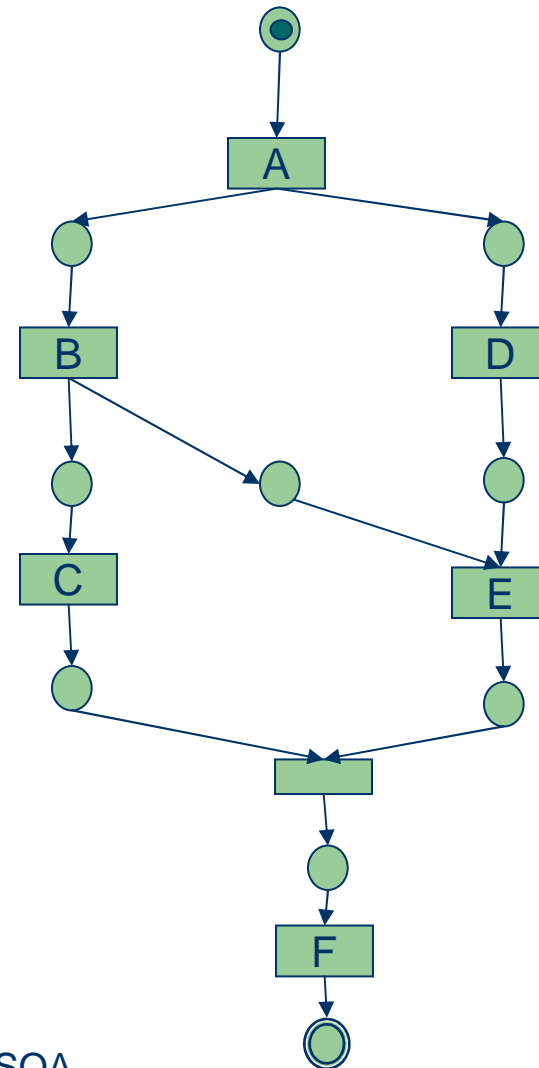
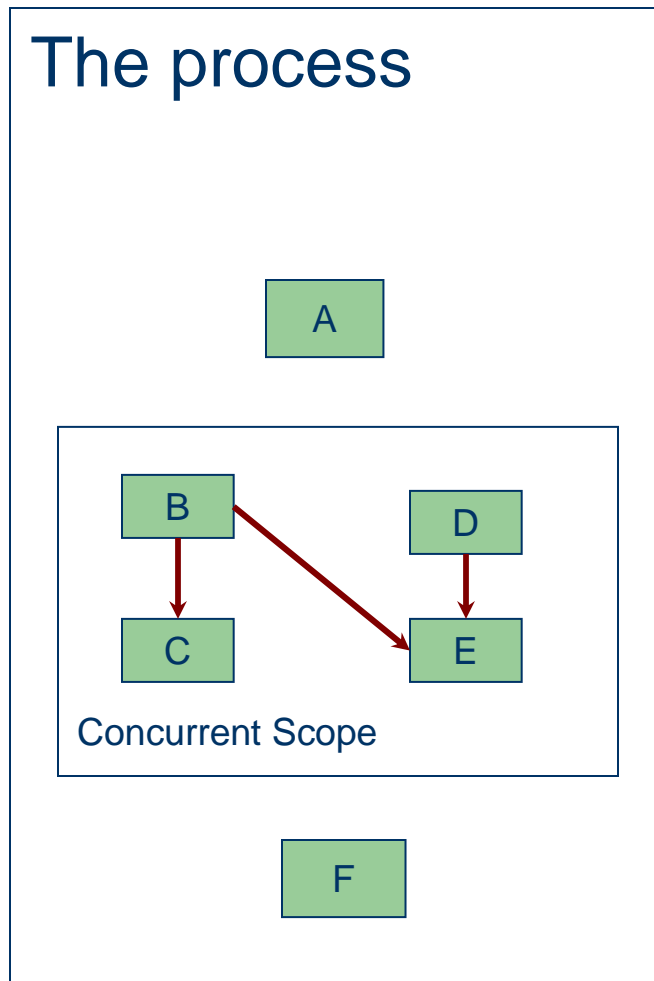
B parallels with C



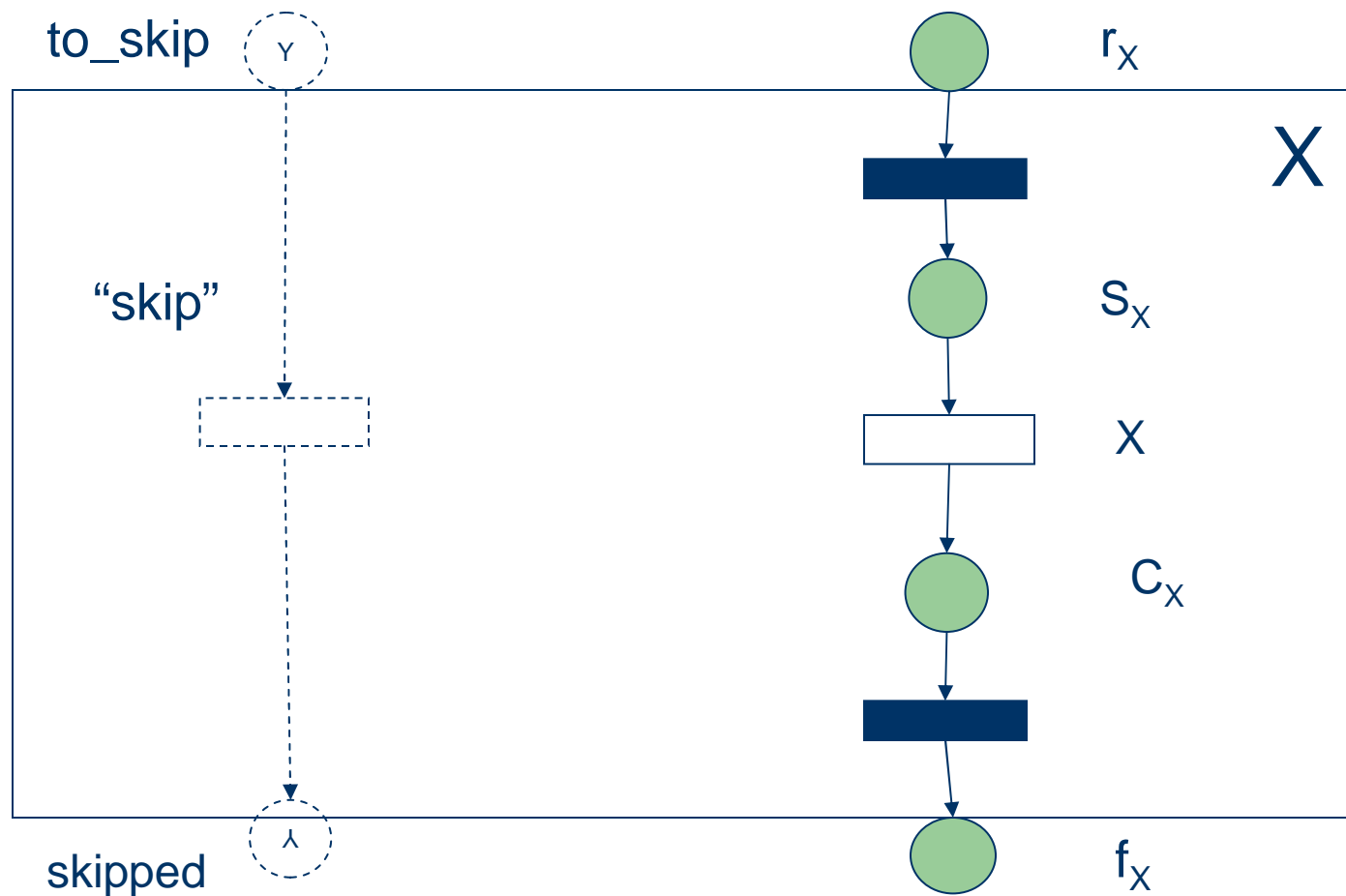
B or C

# The Petri net for the Example Process

The process



# The Petri Net for the Internal Behavior of an Activity [Aalst05]



# Process Algebra

- The processes are given by

$$P ::= a.P \mid P + P' \mid P \parallel_S P' \mid P \setminus \{a\} \mid P[f] \mid !P \mid \mathbf{0}$$

- ★ the *inaction*  $\mathbf{0}$ : a process that does nothing;
- ★ the *prefix*  $a.P$ :  $a \in \mathcal{Act} = \{a, b, c, \dots\} \cup \{\tau\}$ .  $\tau$  is non-observable action.  
 $a$ : receive a message.  $'a$ : emit a message.
- ★ *nondeterministic choice*  $P + P'$ .
- ★ *parallel composition*  $P \parallel_S P'$ .  $S \subseteq \mathcal{Act}$ : the synchronization set.
- ★ *restriction*  $P \setminus \{a\}$ : action  $a$  is within the scope of  $P$ . The external environment observes a  $\tau$  action.
- ★  $P[f]$  behaves like  $P$ , but with the actions relabeled by the function  $f : \mathcal{Act} \rightarrow \mathcal{Act}$ .
- ★ *replication*  $!P$ : an infinite composition of  $P|P|...$

# Actions in Pi-calculus

$a ::= c\langle x \rangle$	<i>send the name <math>x</math> via the channel <math>c</math></i>
$c(x)$	<i>receive any name via channel <math>c</math>, binding the name received to <math>x</math></i>
$\tau$	<i>unobservable action</i>

# Process Algebra for WS Process Modeling

- Manipulate variables

$$\bar{w} \mapsto \bar{u}$$

- Basic activities:

$$\langle \text{receive } \underline{a} \ \underline{v} \rangle \langle \text{corr. } \underline{w} \rangle \langle \backslash \text{corr. } \rangle \langle \backslash \text{receive } \rangle \mapsto \text{send}(\bar{l}, \bar{v}, \bar{w})$$

- Structured activities

$$\langle \text{sequence } \rangle \mapsto ;$$

$$\langle \text{flow } \rangle \mapsto \parallel$$

$$\langle \text{while condition} = \underline{e} \ \underline{a} \ \langle \backslash \text{while } \rangle \rangle \mapsto \text{while}(e)\{a\}$$

# Process Algebra for WS Process Modeling

- Operational Semantics

<flow >:

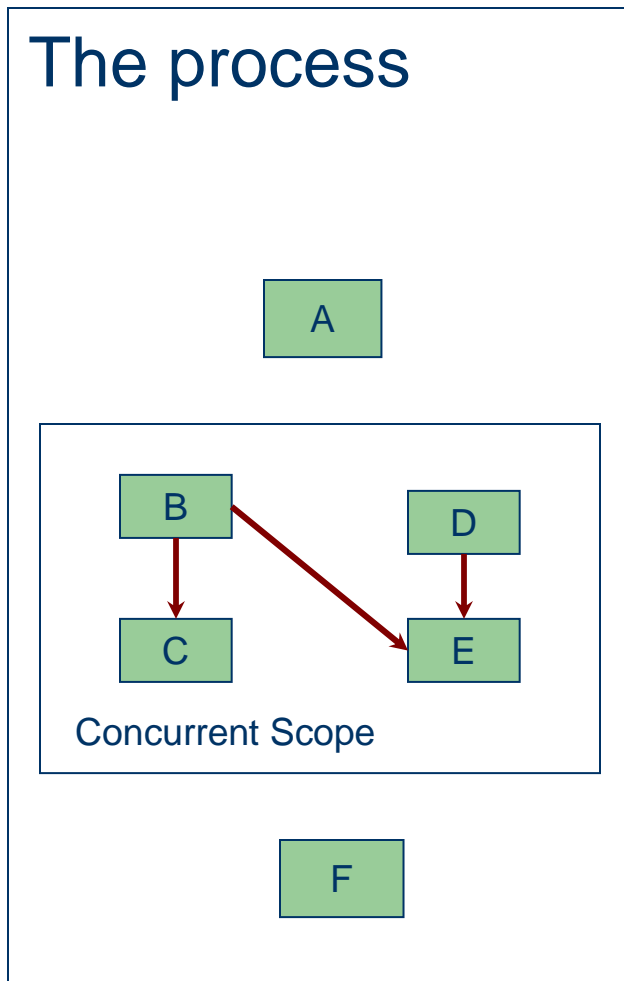
$$(\overline{w} \mapsto \overline{u})(s_0 \parallel s_1) \xrightarrow{\alpha} (\overline{w'} \mapsto \overline{u'})(s'_0 \parallel s_1) \text{ if } (\overline{w} \mapsto \overline{u})s_0 \xrightarrow{\alpha} (\overline{w'} \mapsto \overline{u'})s'_0$$

<link >

$$(\overline{w} \mapsto \overline{u})(source(\lambda); s \parallel target(\lambda); s') \xrightarrow{\tau} (\overline{w} \mapsto \overline{u})(s \parallel s')$$

# Process Algebra for the Example WS Process

## The process



```

recv( $\bar{a}, \bar{v}, \bar{p}$ );
{ { invoke( $\bar{b}, \bar{b}_{in}, \bar{b}_{out}$ ); ((source(b_to_c); target(b_to_c);
invoke( $\bar{c}, \bar{c}_{in}, \bar{c}_{out}$ )) || source(b_to_e)) } ||
{ ((invoke( $\bar{d}, \bar{d}_{in}, \bar{d}_{out}$ ); source(d_to_e);
target(d_to_e)) || target(b_to_e));
invoke( $\bar{e}, \bar{e}_{in}, \bar{e}_{out}$ )) } };
send( $\bar{f}, \bar{w}, \bar{p}$ )

```

Simplified to :

```

recv( $\bar{a}, \bar{v}, \bar{p}$ );
{ { invoke( $\bar{b}, \bar{b}_{in}, \bar{b}_{out}$ ); (
invoke( $\bar{c}, \bar{c}_{in}, \bar{c}_{out}$ )) || source(b_to_e)) } ||
{ (invoke( $\bar{d}, \bar{d}_{in}, \bar{d}_{out}$ )) || target(b_to_e));
invoke( $\bar{e}, \bar{e}_{in}, \bar{e}_{out}$ )) } };
send( $\bar{f}, \bar{w}, \bar{p}$ )

```

# Control flow verification

---

- Temporal logic model checking to prove properties of services: liveness, safety, and request-response (a request is always satisfied, also for infinite behaviors). If the property is not satisfied, a counterexample is returned.
- Bisimulation, to check whether the behaviors of two services or two versions of the same service are equivalent; or, if they are different.
- Simulation, to check whether the behavior of a service is included within the behavior of other interacting services.
- Execution traces of the service, to understand the behavior of the service.

# Data flow verification

---

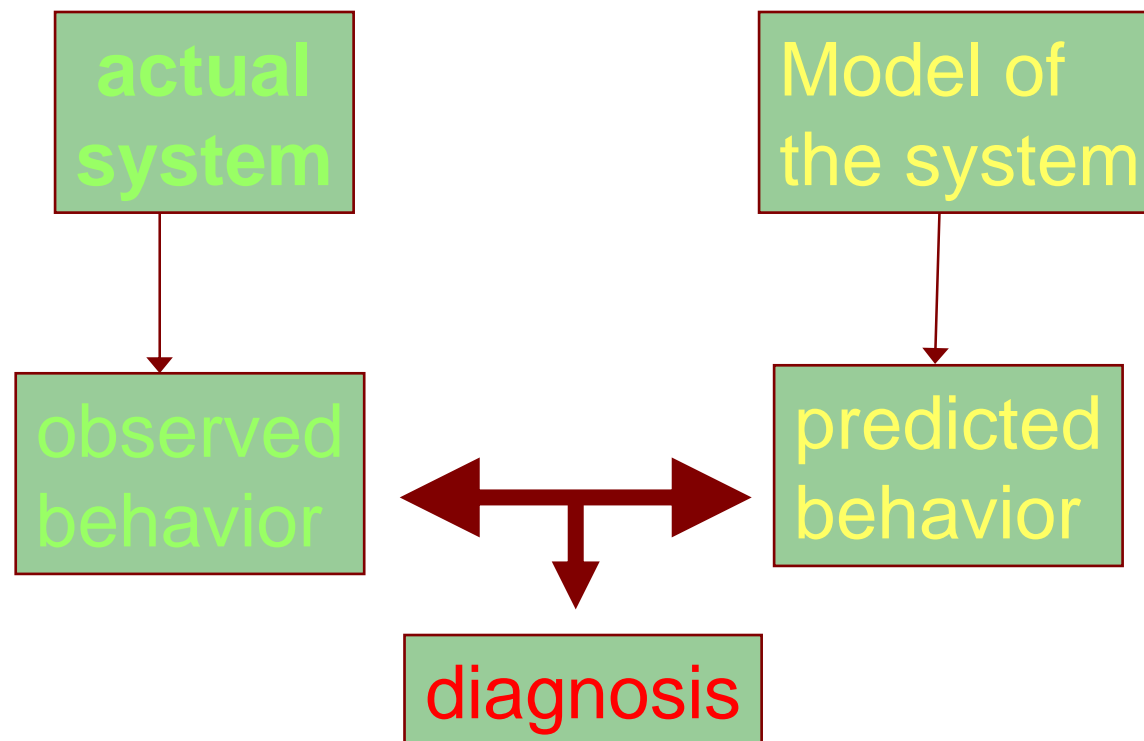
- data type checking, in the case of LOTOS and other process algebras allowing data handling.

# Model-based Diagnosis and Monitoring

---

- Find out the Web Services that are responsible for the failed business process
- Determine what kind of information to record for diagnosis
- How to recover from failures

# The Principle of Model-based Diagnosis



# Diagnosis with Discrete Event Systems

---

- Model right behavior, as well as faulty behavior
- Unfold the execution trajectory based on observations
- If a fault action is in the trajectory, this fault occurs
- Recovery is a replanning task?
- Diagnosability for discrete event system?

# Reconfiguration

---

- Change business partners or business process at run time
- Model the end points in the model
- Model the operations to forward the end points?
- BPEL has to use the forwarded end points?
- Replanning?

# Self-manageable Web Service Processes

---

Self-management = automatic composition +  
verification + fault diagnosis + monitoring +  
reconfiguration