

High-level Supervision of Program Execution Based on Formal Specification

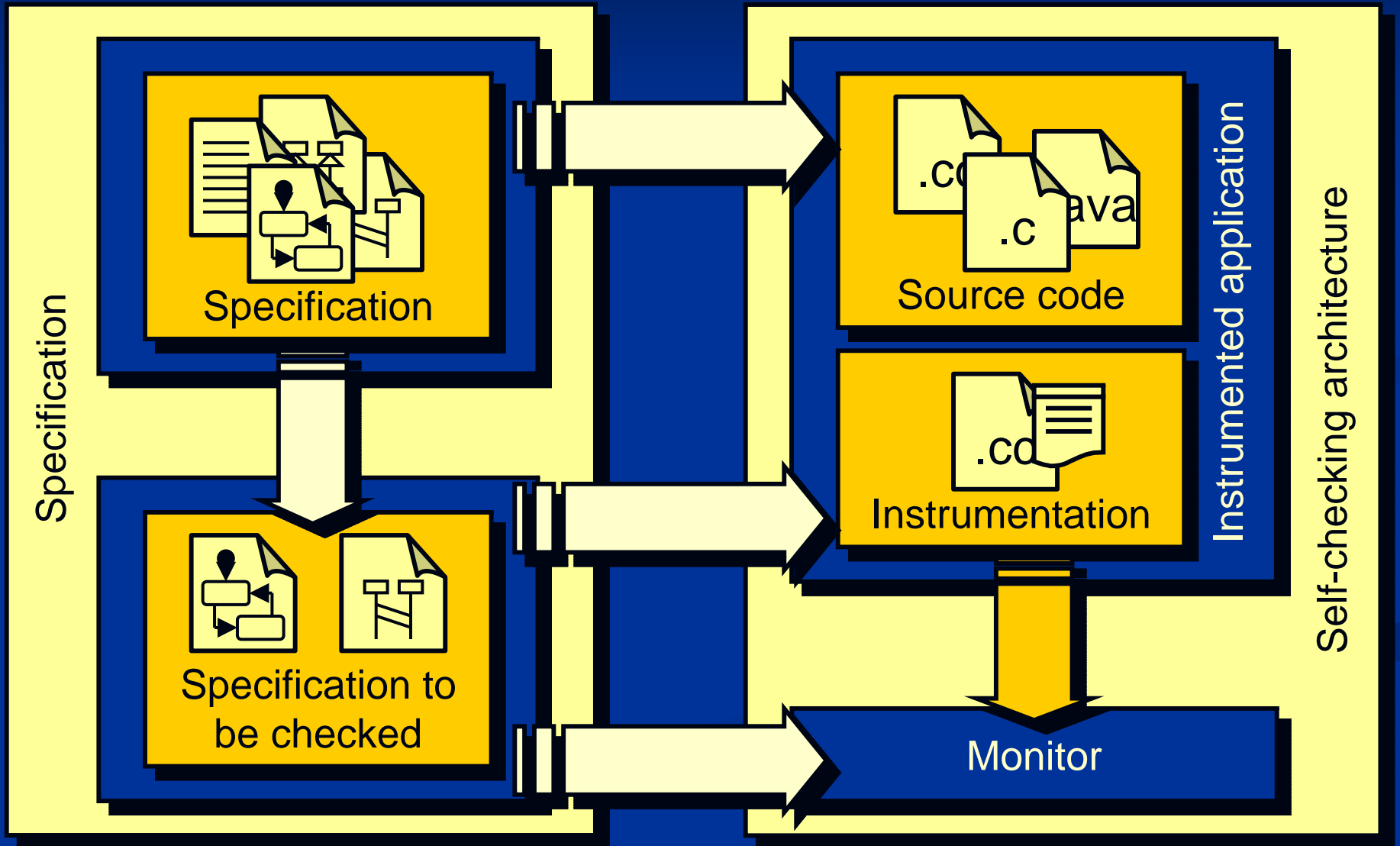
- Gergely PINTÉR
- István MAJZIK

Budapest University of Technology and Economics
Department of Measurement and Information Systems

Research goals

- Run-time fault-detection architecture based on the abstract specification
 - Behavioral models (e.g. **statecharts**)
 - Communication protocols (e.g. **live sequence charts, sequence diagrams**)
- Configurable granularity of observation
 - Selection of key aspects of the specification (e.g. by Temporal Logic Formulae)
- Supporting safety-critical SW architectures (e.g. EN-50128)

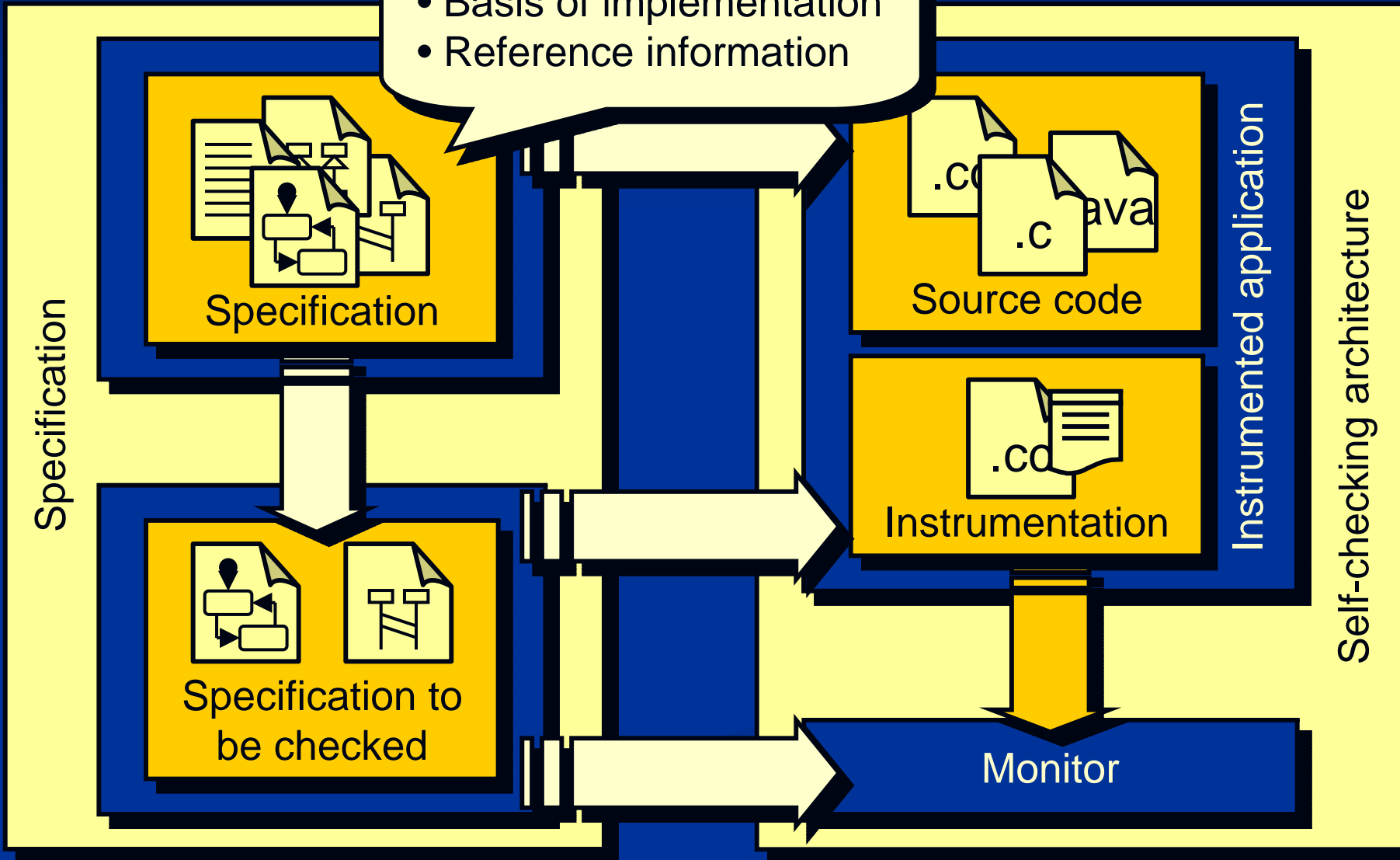
Run-time verification against formal models



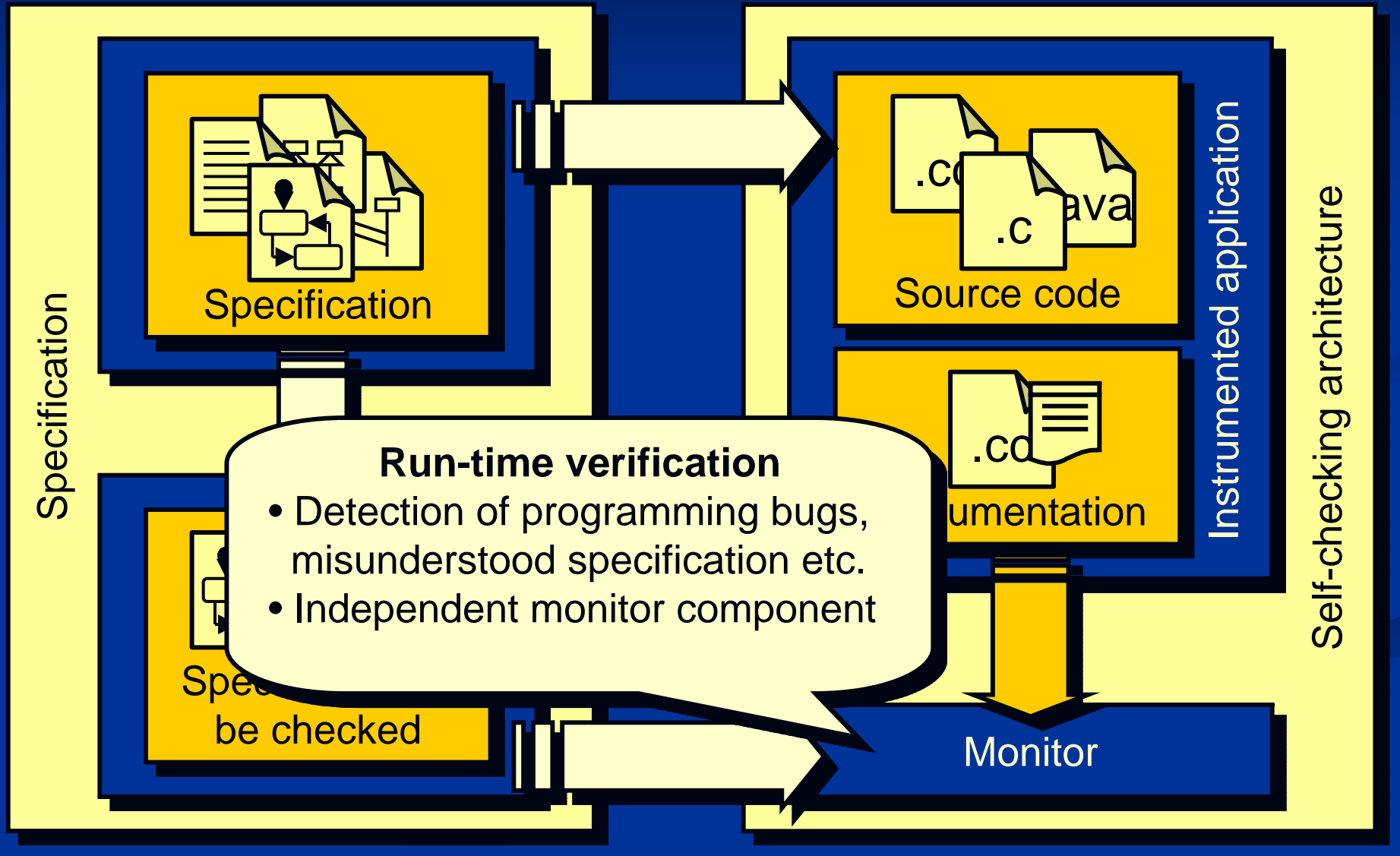
Run-time verification against

Specification

- Basis of implementation
- Reference information



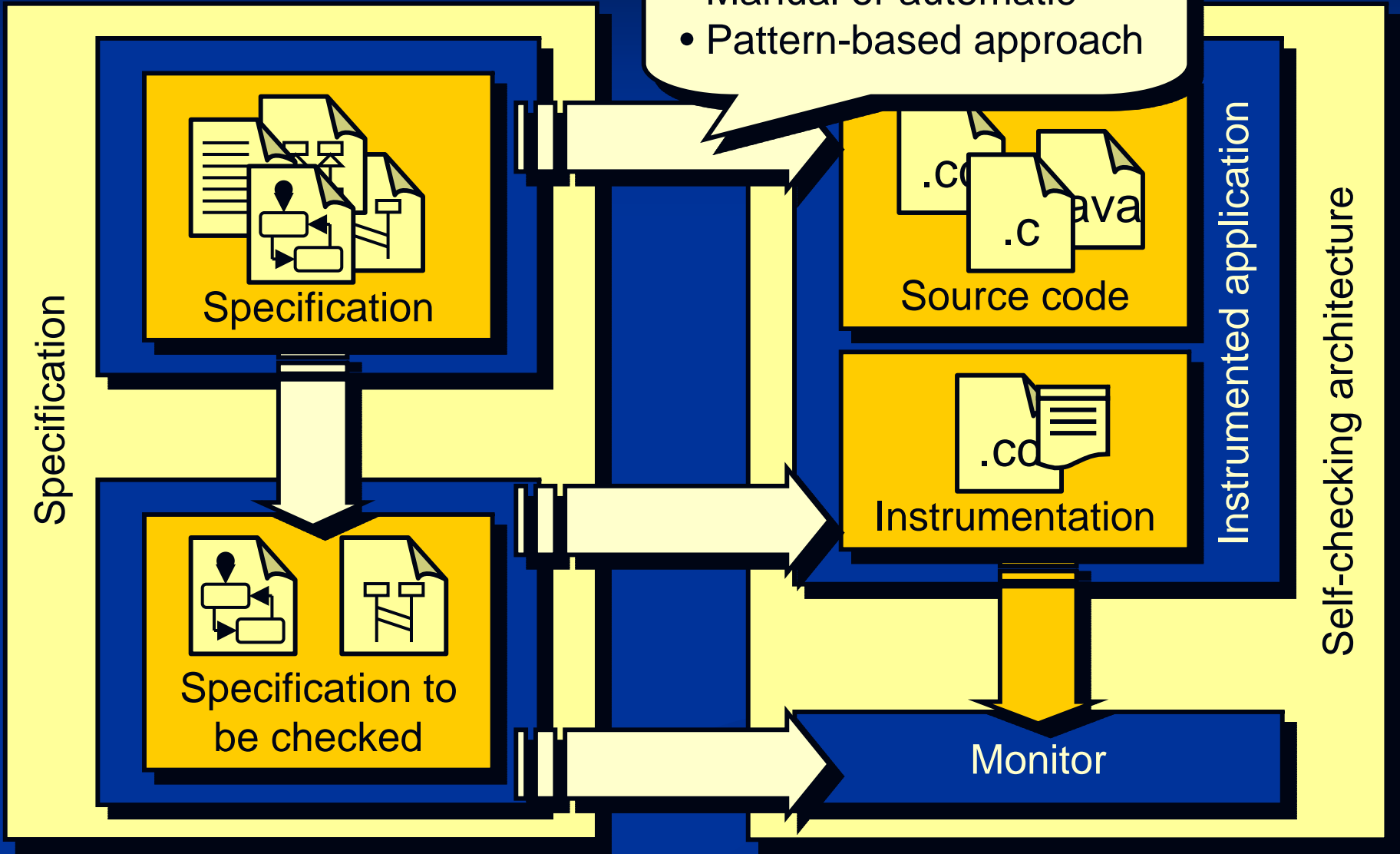
Run-time verification against formal models



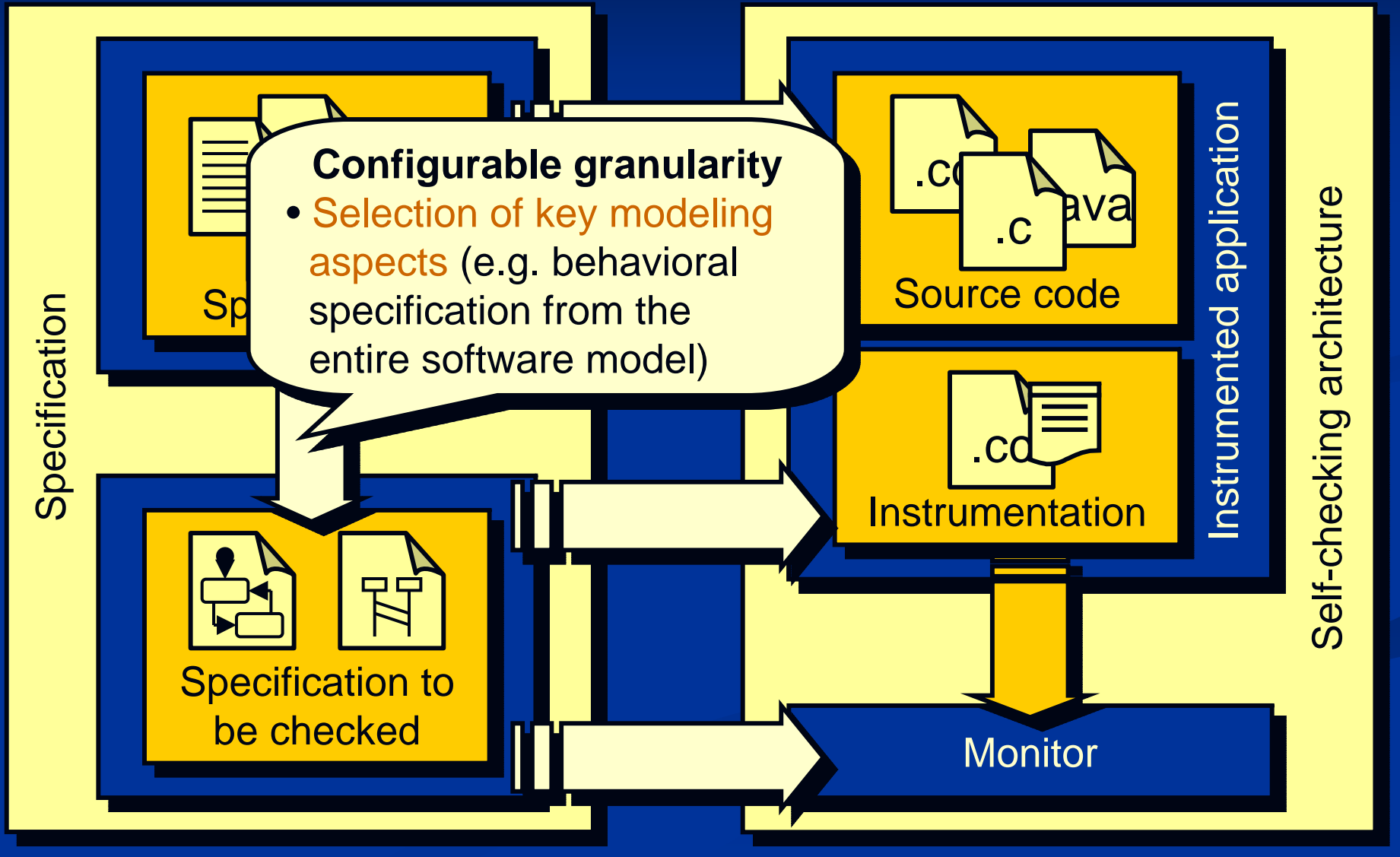
Run-time verification against formal

Implementation

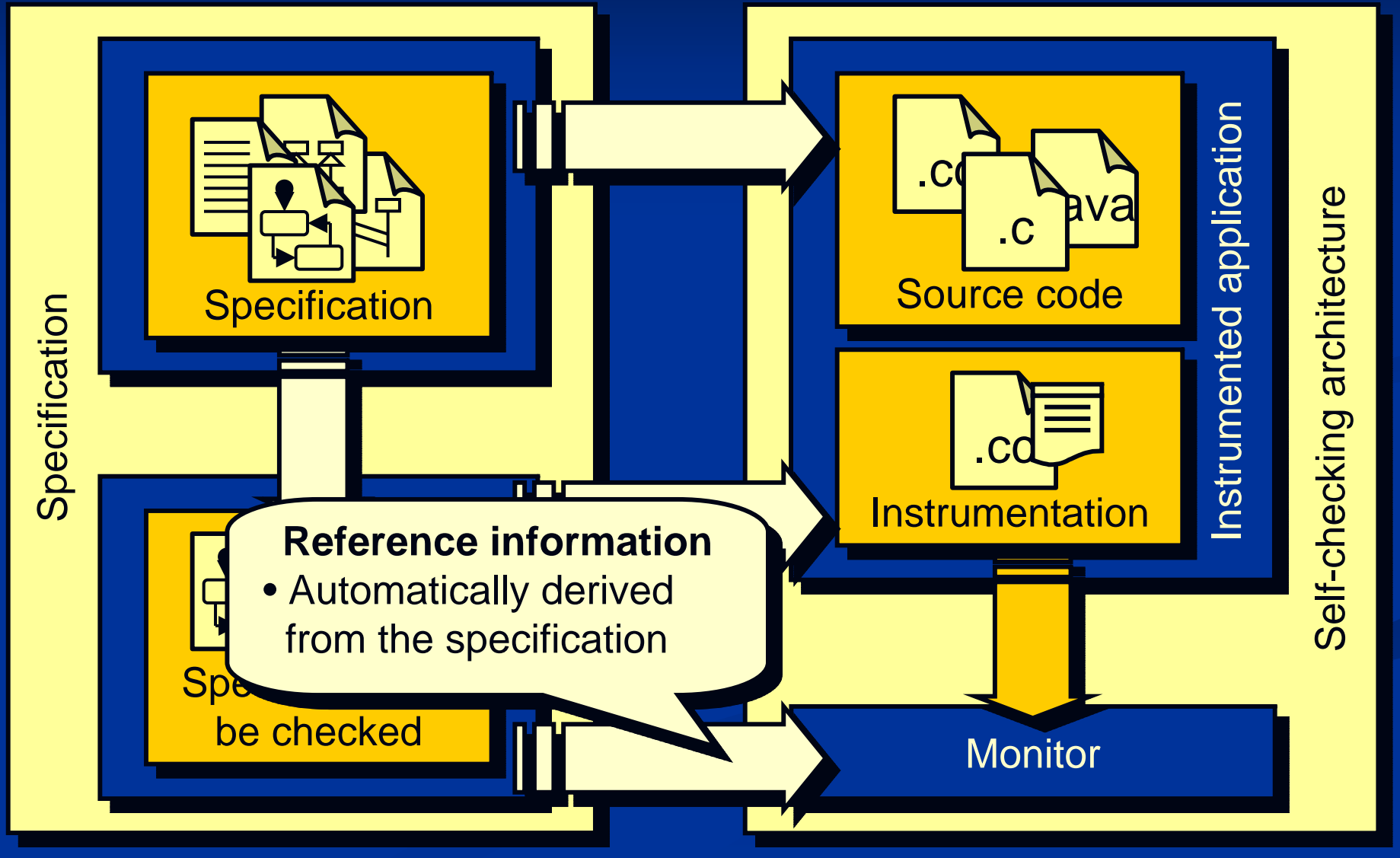
- Manual or automatic
- Pattern-based approach



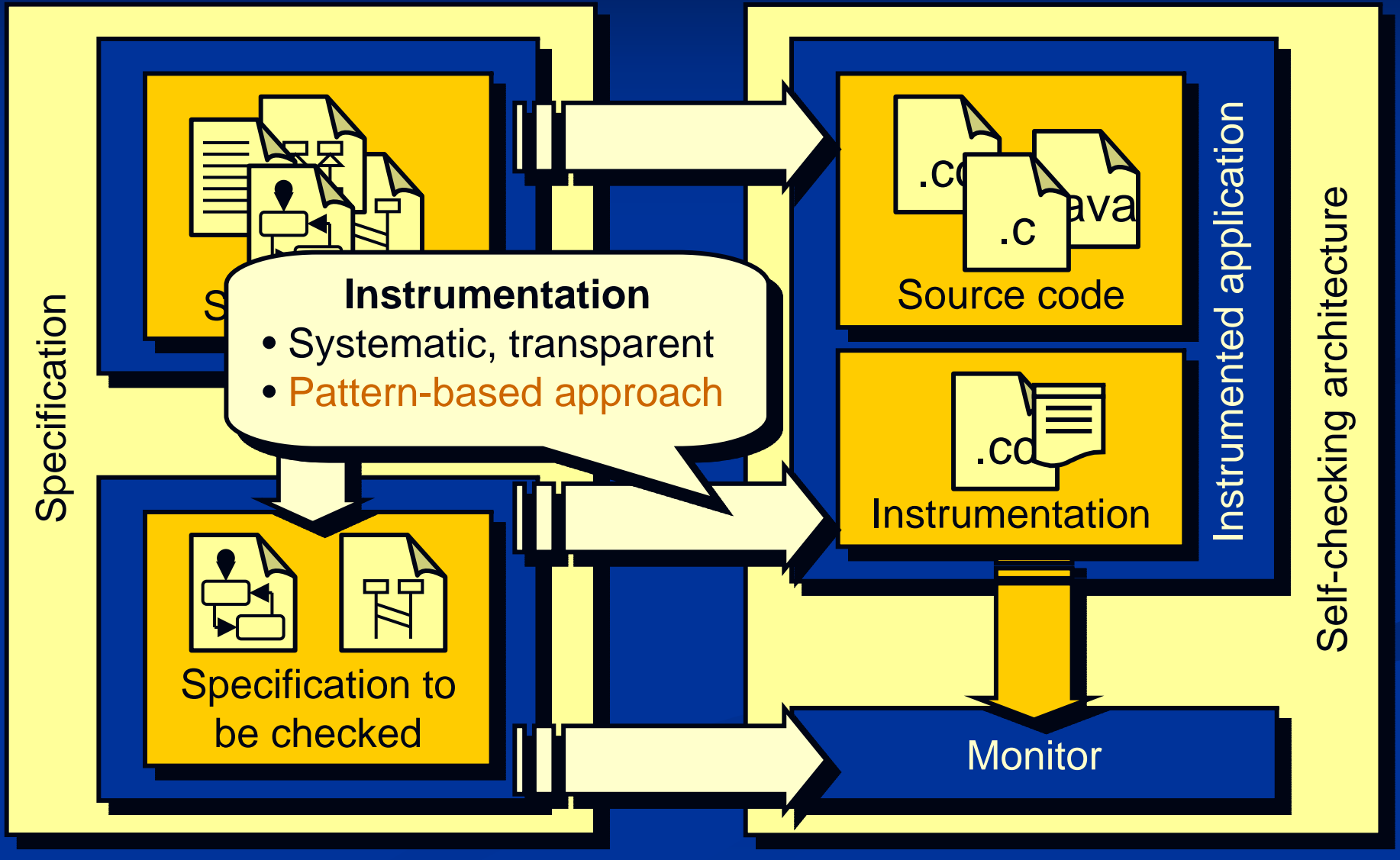
Run-time verification against formal models



Run-time verification against formal models

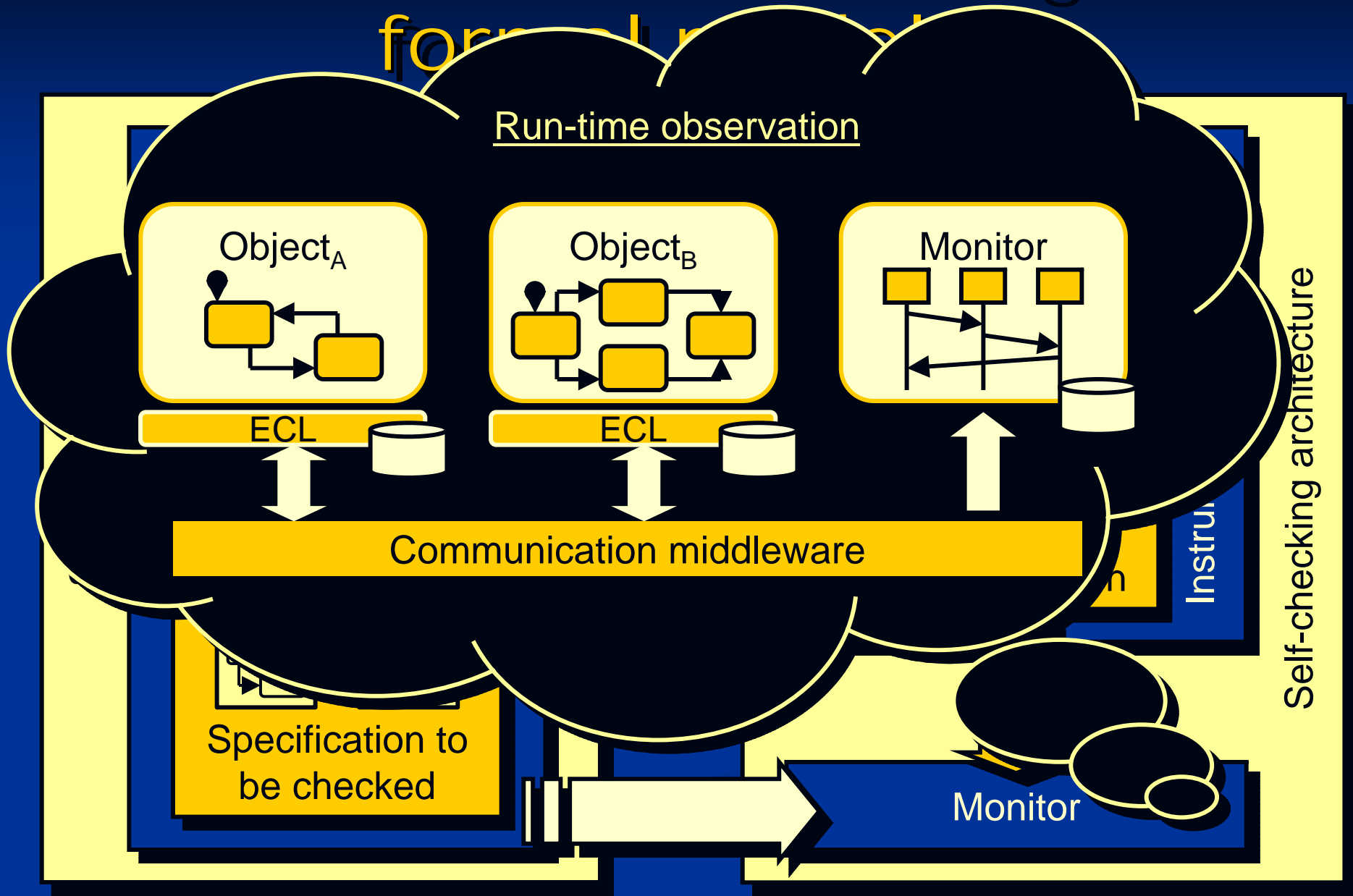


Run-time verification against formal models



Run-time verification against

formal requirements



Self-checking architecture

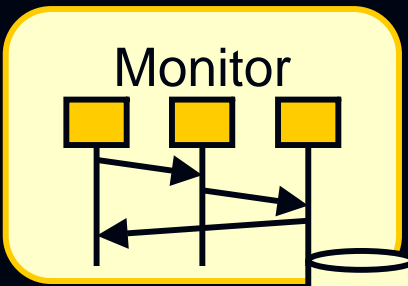
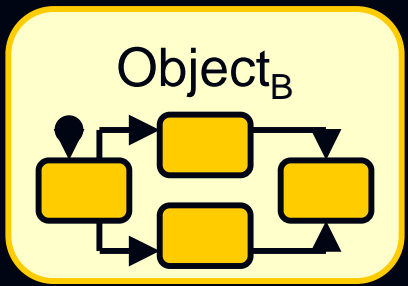
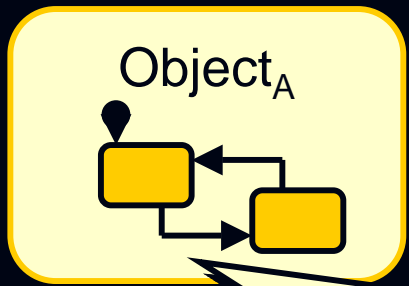
Instru

Monitor

Run-time verification against

formal requirements

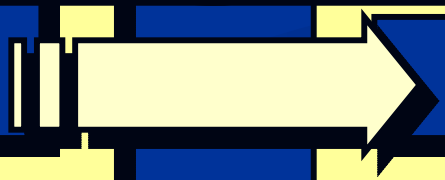
Run-time observation



Intra-object behavior

- Internal event-driven behavior should correspond to the behavioral specification
- Fault detection by embedded component
- Reference information:
UML statecharts

Specifications to be checked



Monitor

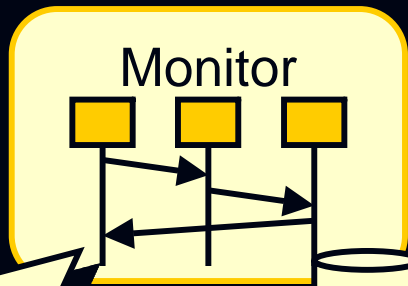
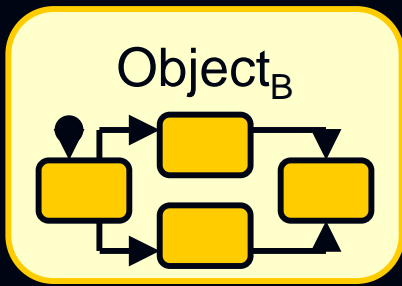
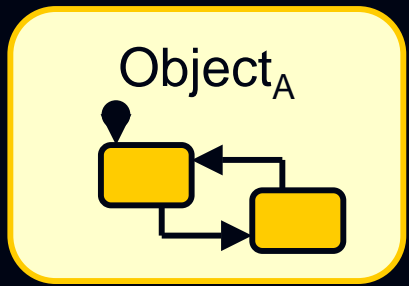
Instru

Self-checking architecture

Run-time verification against

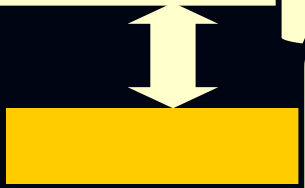
formal models

Run-time observation



ECL

ECL



Specificati
be check

- Inter-object communication**
- Monitor component on the communication bus
 - Detection of communication protocol violations
 - Reference information:
Sequence diagrams,
Life sequence charts



Monitor

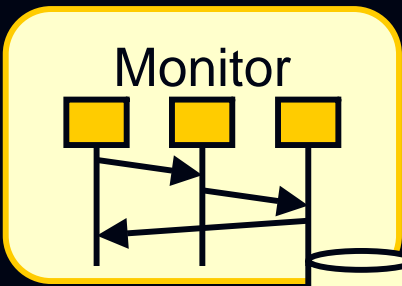
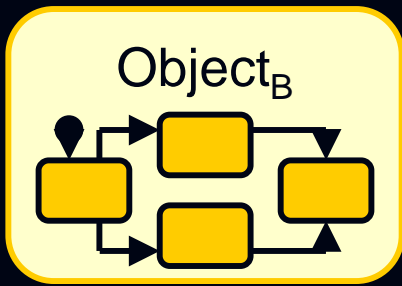
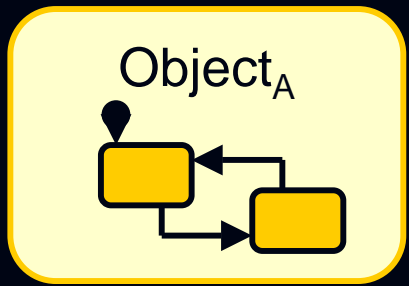
Instru

Self-checking architecture

Run-time verification against

formal requirements

Run-time observation



Self-checking architecture

Error Confinement Layer

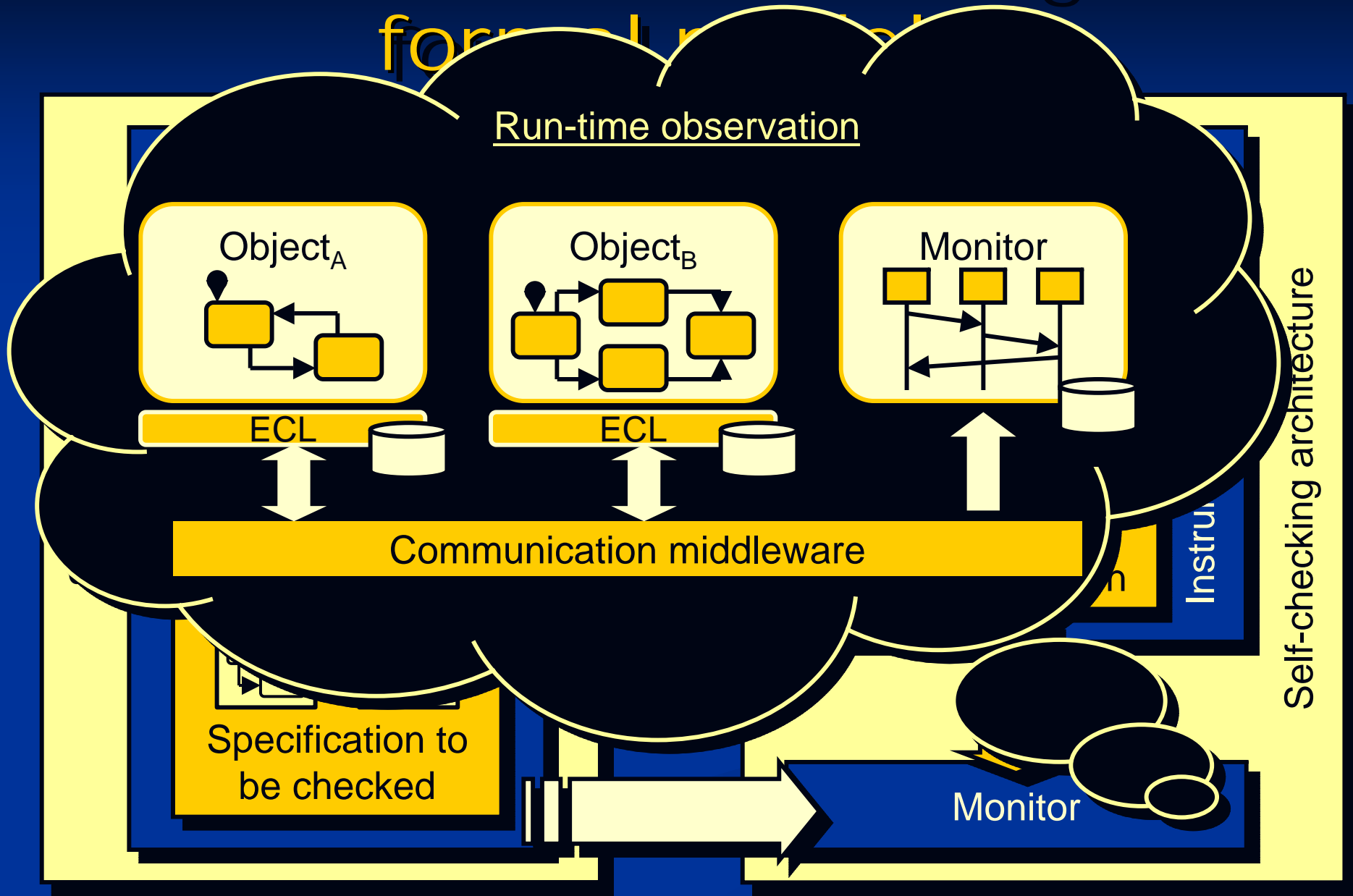
- Detach the faulty object after fault detection
- Fault silent behavior or more advanced schemes

be checked

Monitor

Run-time verification against

formal requirements



Self-checking architecture

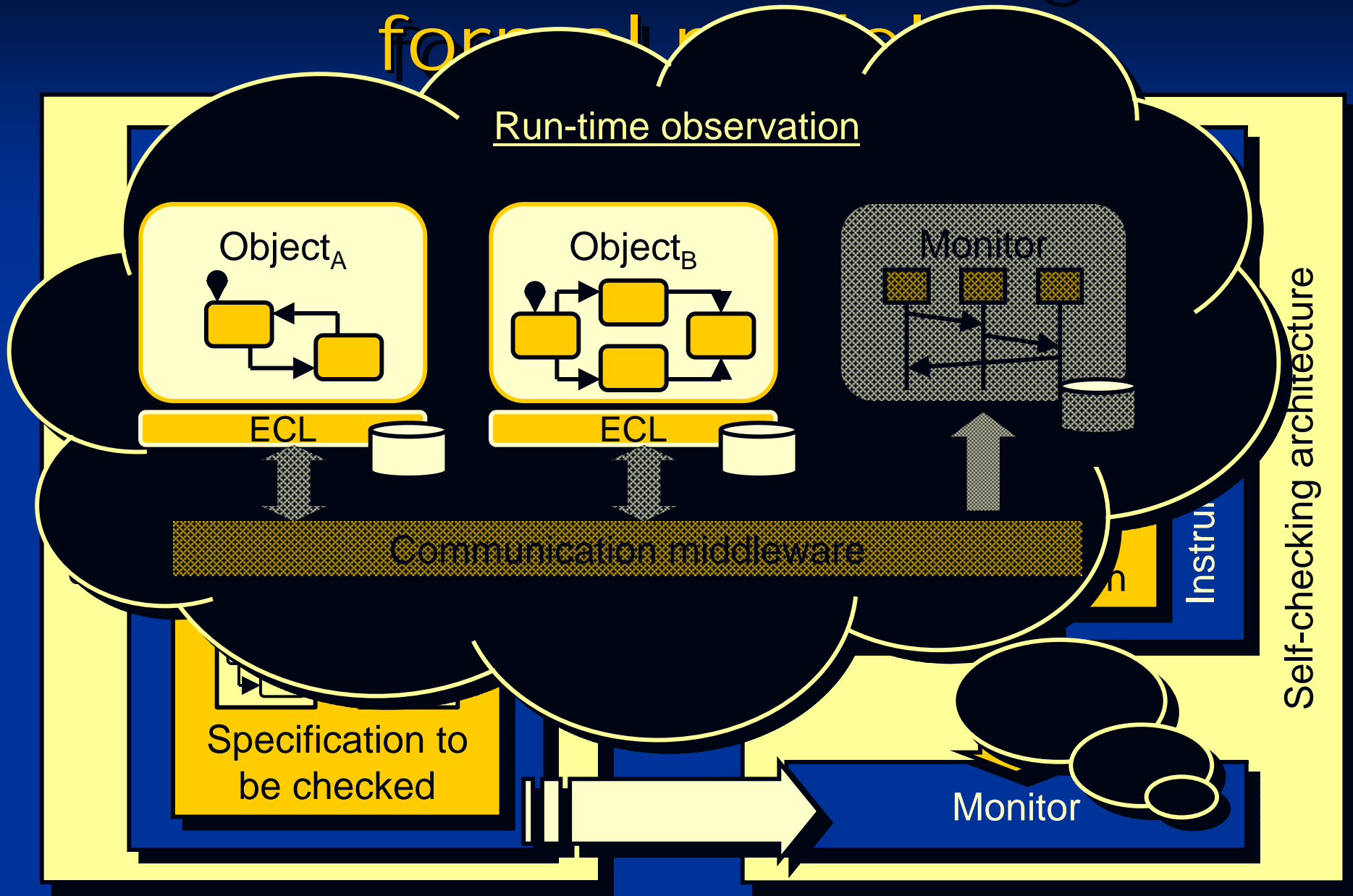
Specification to be checked

Monitor

Instru

Run-time verification against

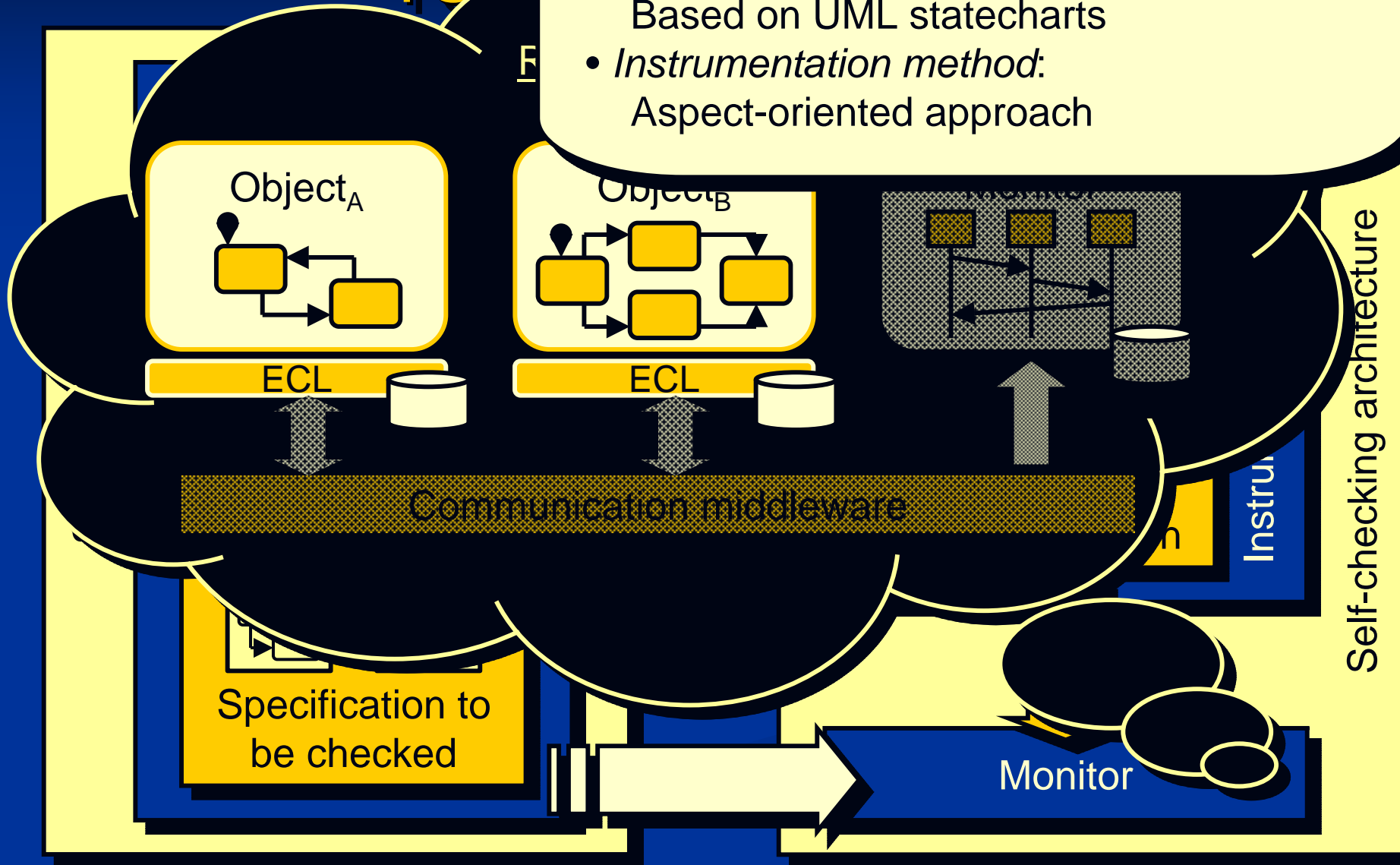
formal requirements



Run-time verification

Focus and Contribution

- Run-time observation of *internal behavior*
- Advanced *monitoring mechanism*:
Based on UML statecharts
- *Instrumentation method*:
Aspect-oriented approach



Self-checking architecture

Instrumentation

Monitor

Specification to be checked

ECL

ECL

Object_A

Object_B

Abstract, high-level control-flow fault detection

Reference information:

- Automatically derived from the behavioral specification
- Capable of expressing state hierarchies, concurrent operation, etc.

Implementation of the monitor:

- Based on the operational semantics of the behavioral model
- Run-time checking of the behavior on the basis of the abstract reference model

Implementation of the instrumentation:

- Providing information to the monitor about the internal behavior
- Configurable, transparent and automatically applied

Abstract, high-level control-flow fault detection

Reference information:

- Automatically derived from the behavioral specification
- Capable of expressing state hierarchies, concurrent operation, etc.

Implementation of the monitor:

- Based on the operational semantics of the behavioral model
- Run-time checking of the behavior on the basis of the abstract reference model

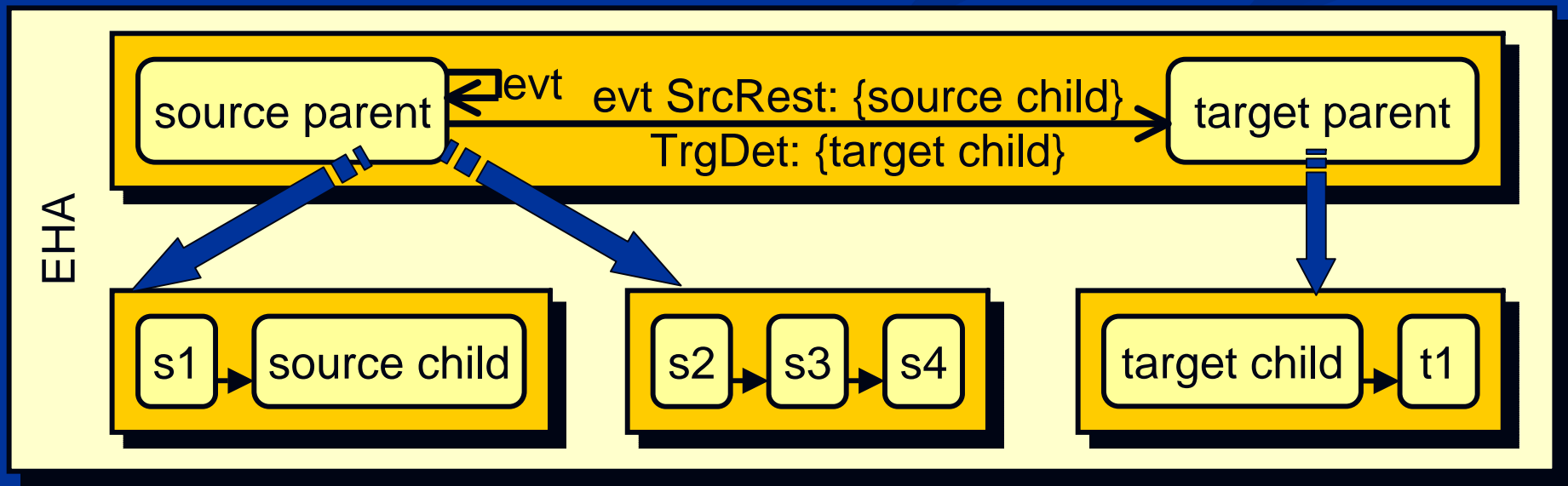
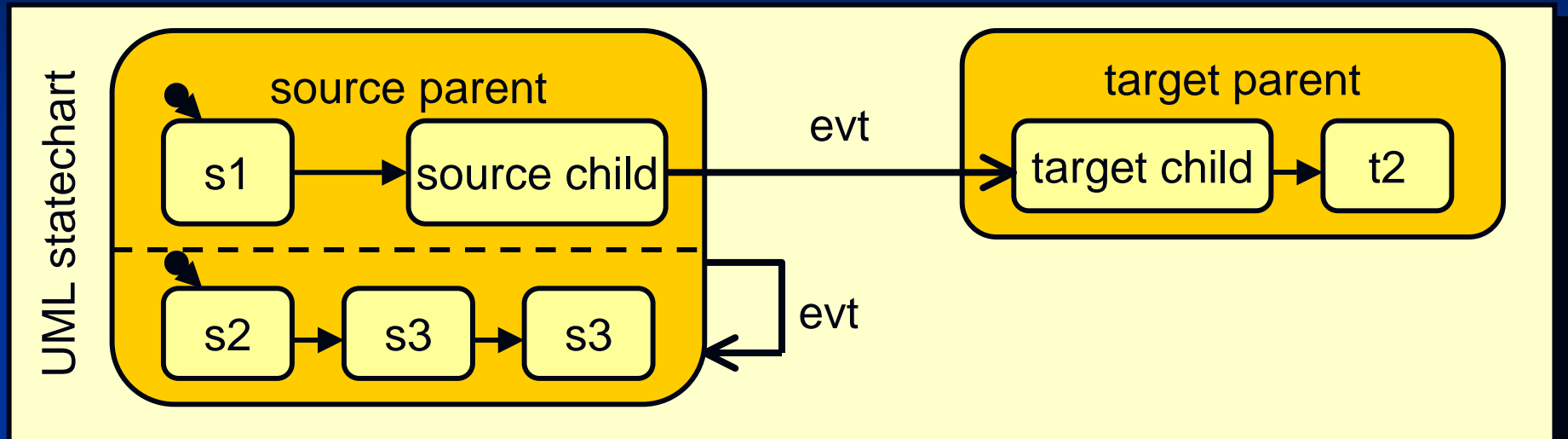
Implementation of the instrumentation:

- Providing information to the monitor about the internal behavior
- Configurable, transparent and automatically applied

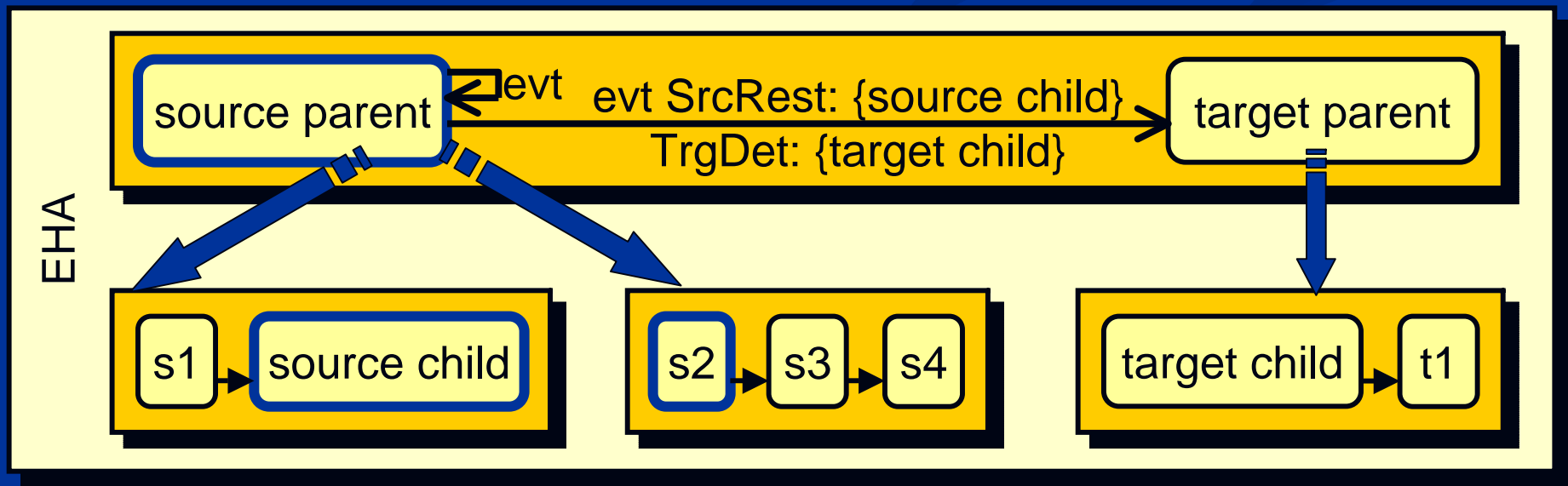
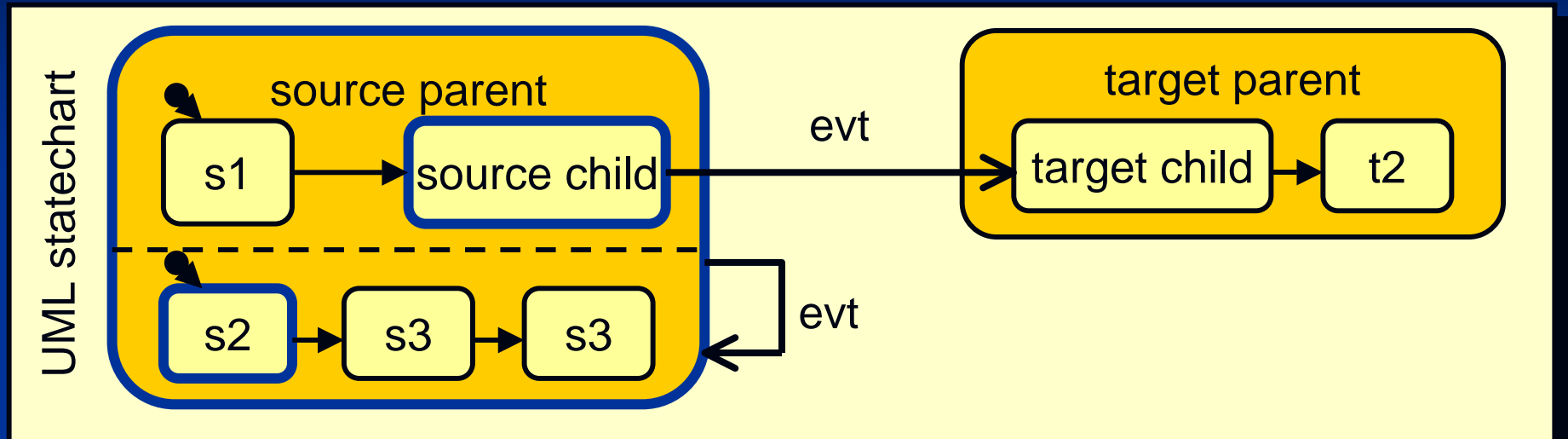
Reference information of the internal behavior

- Extended Hierarchical Automata (EHA)
 - Clear structure:
 - Sequential automata:
Containing any number of states
 - Non-composite states:
Refined to any number of sequential automata
 - Non-interlevel transitions:
Source restriction and target determination sets
 - Well elaborated formal semantics
 - Automatically derived from UML statecharts

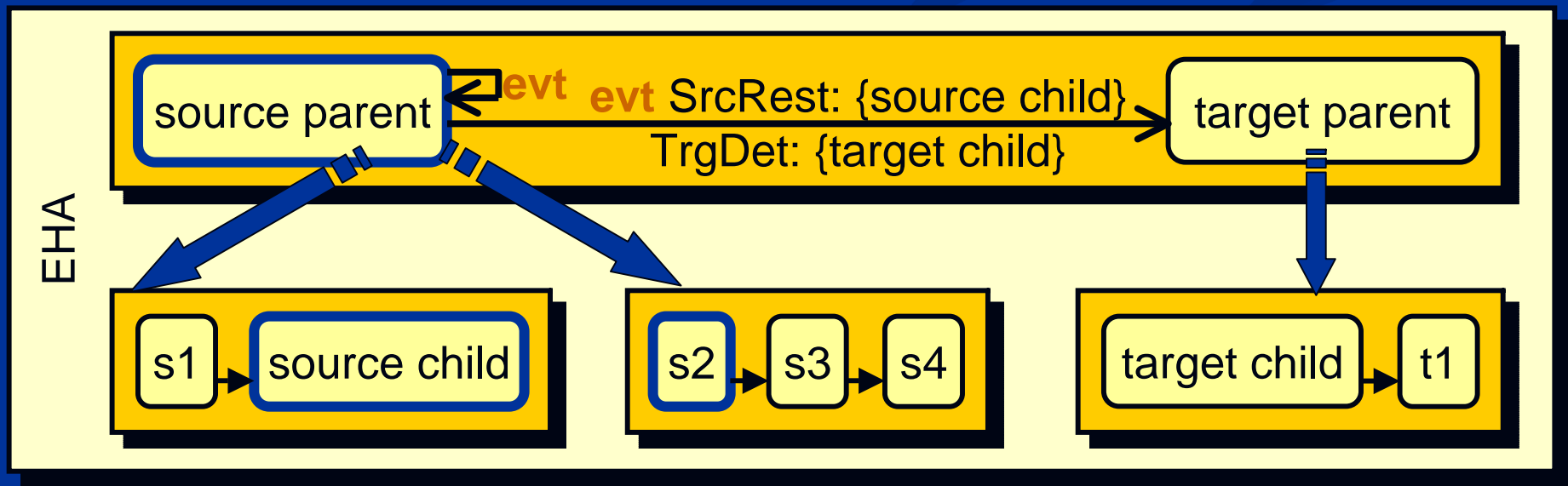
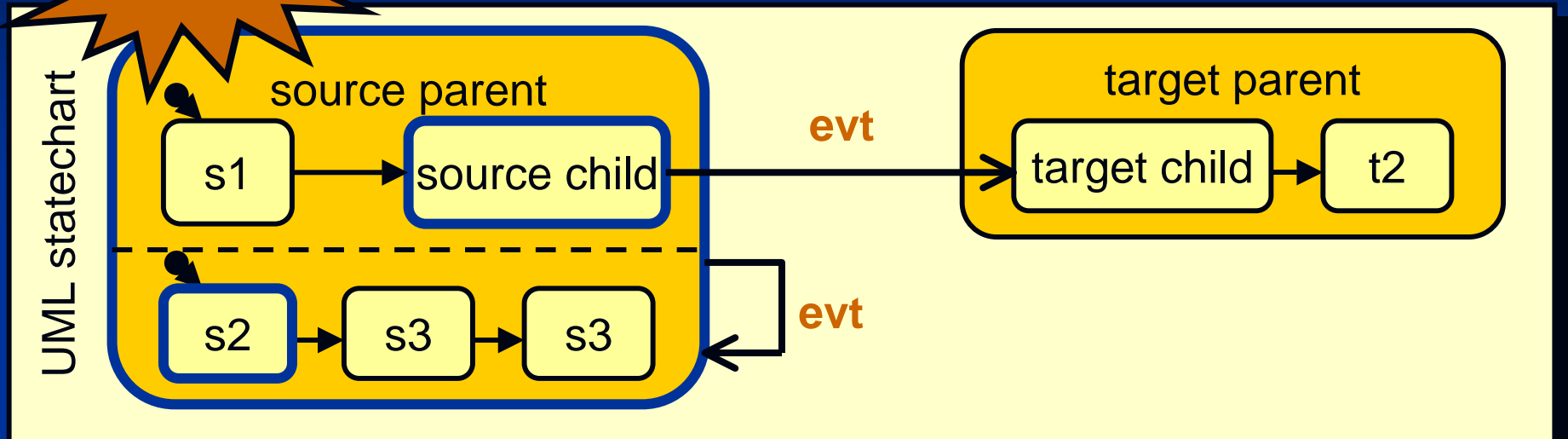
Reference information of the internal behavior



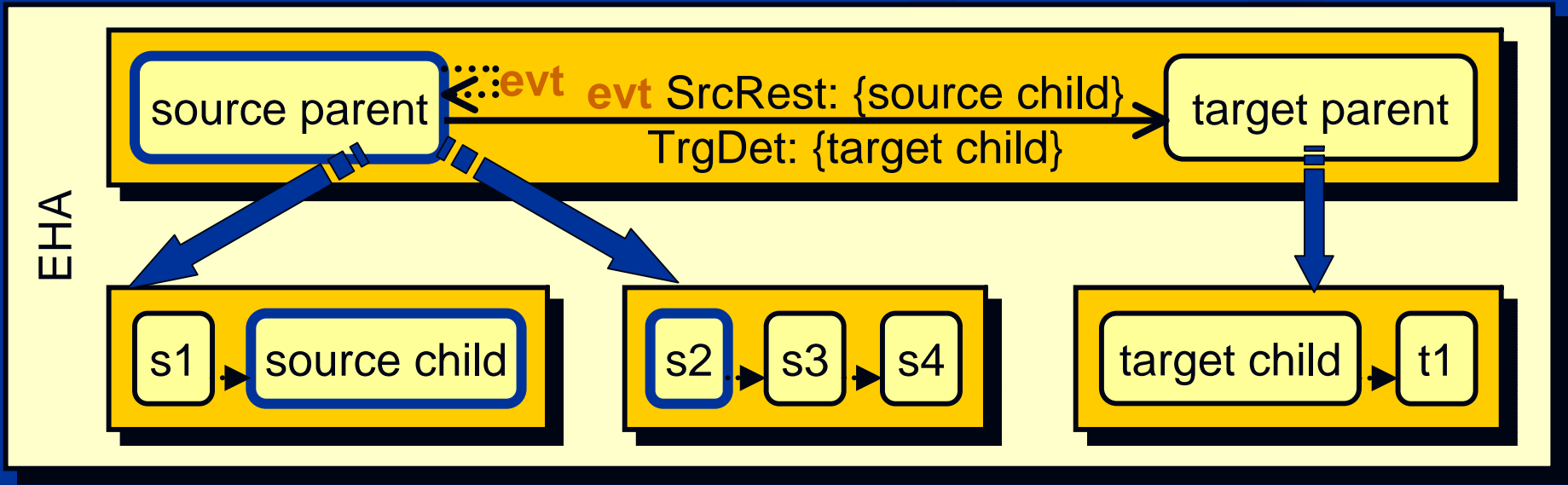
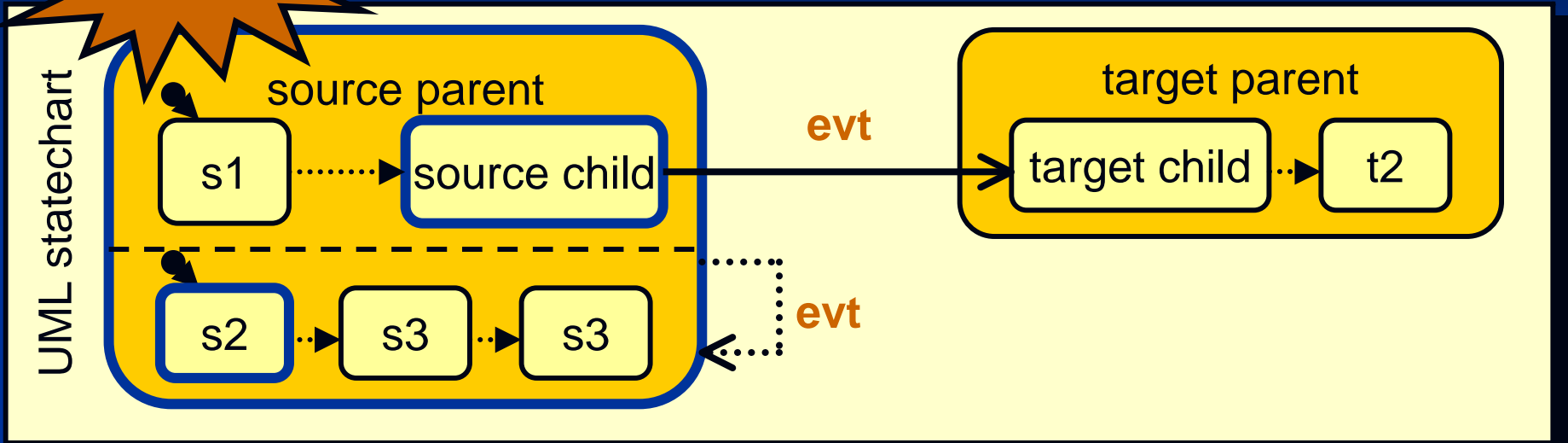
Reference information of the internal behavior



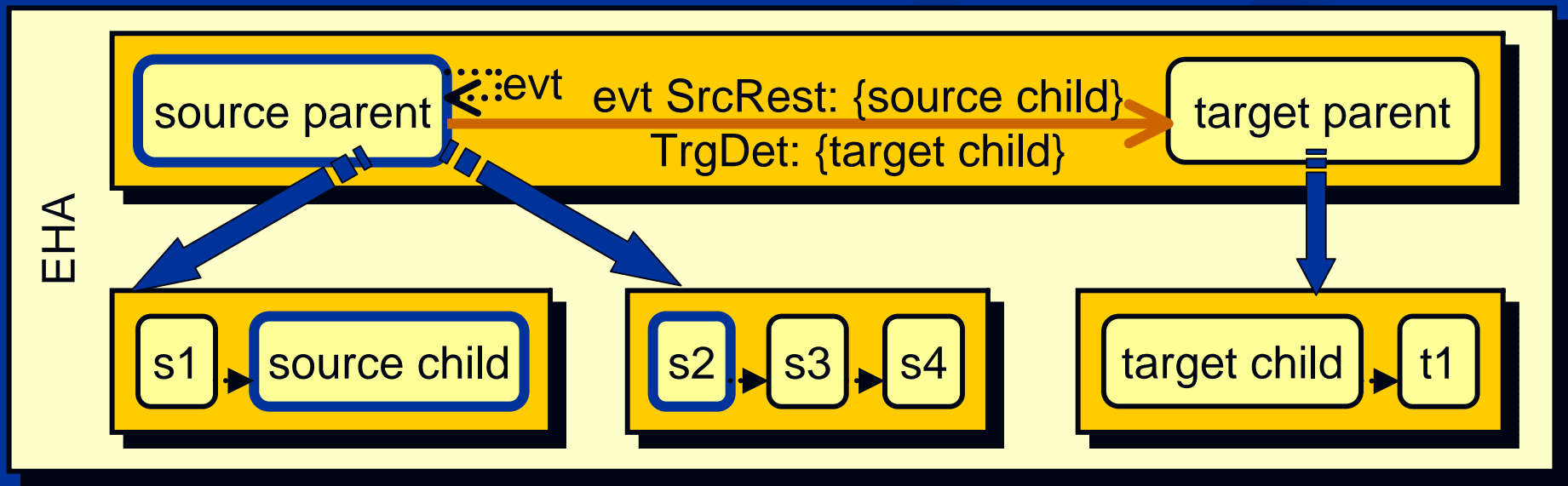
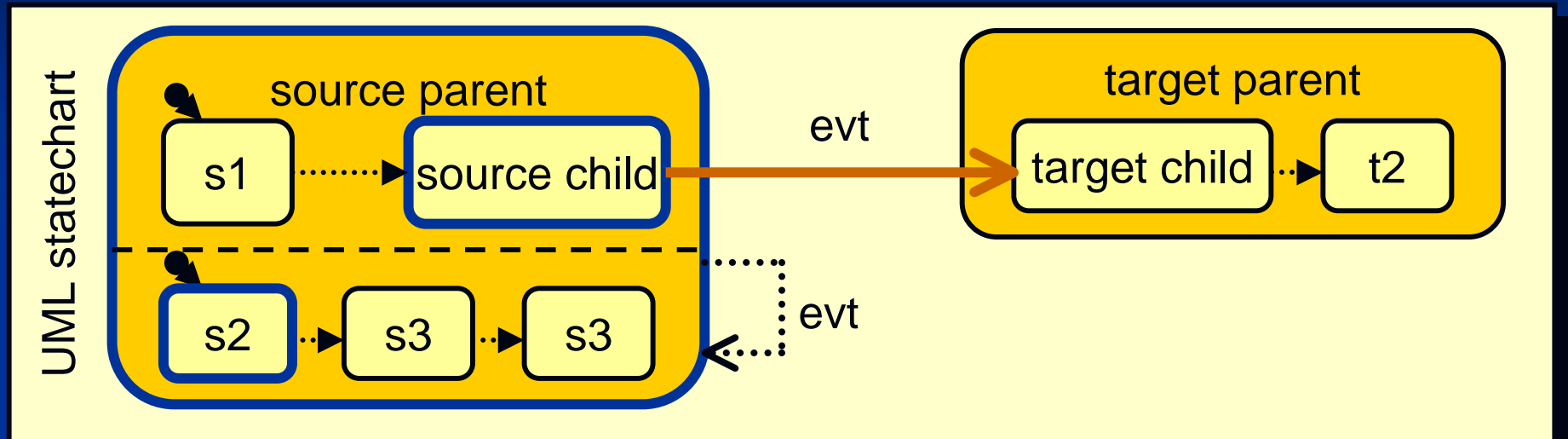
Reference information of the internal behavior



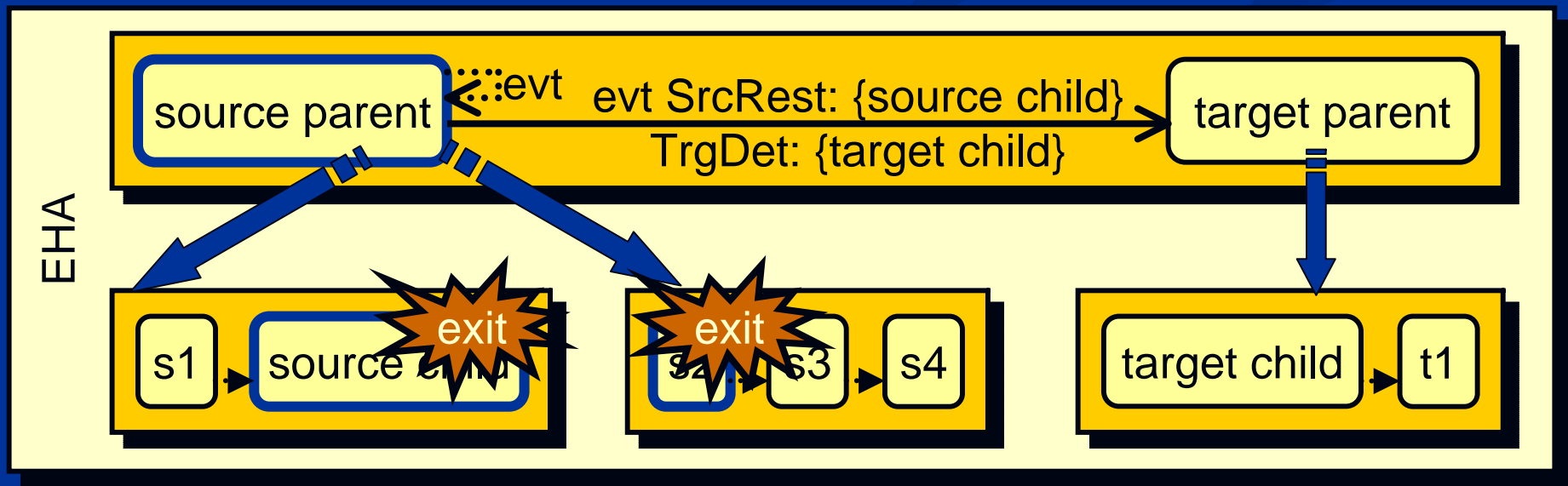
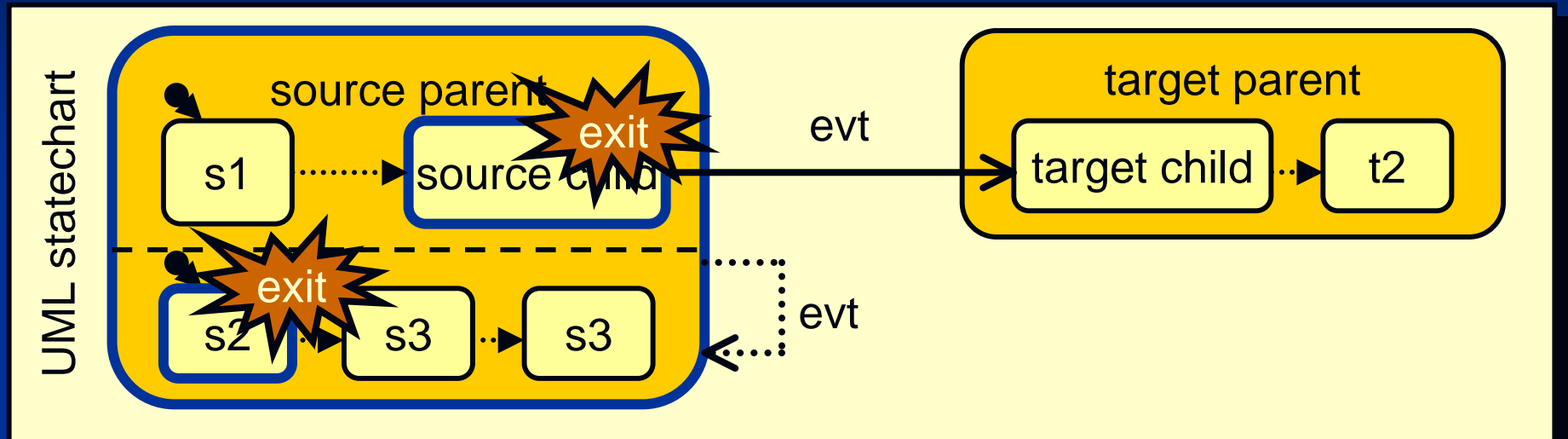
Reference information of the internal behavior



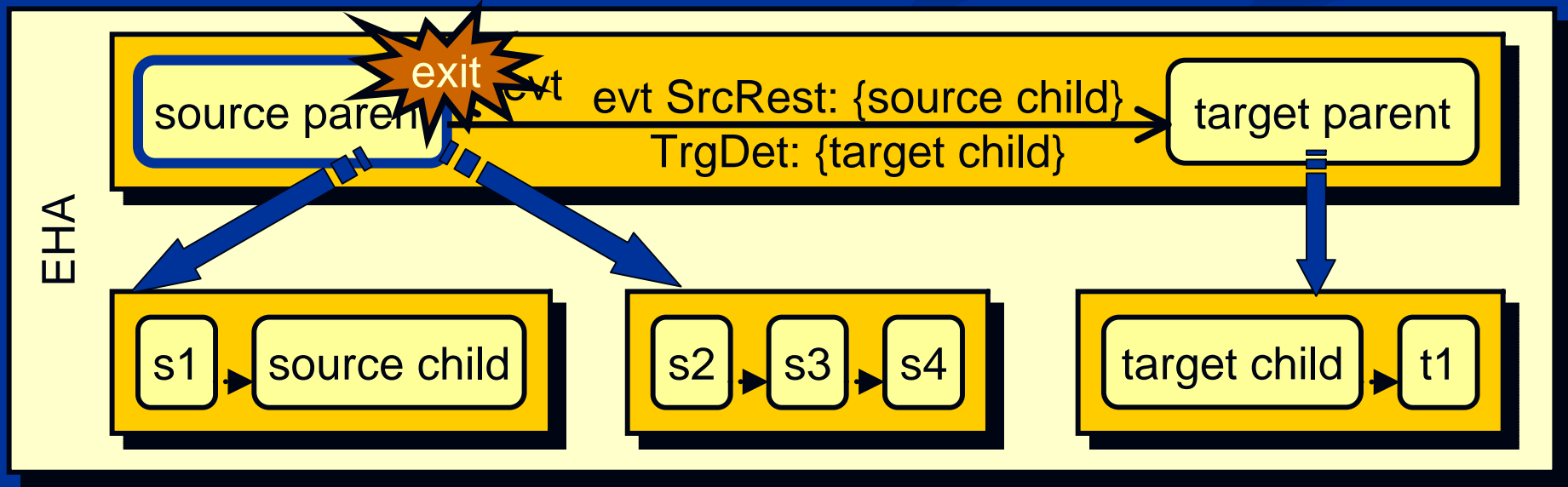
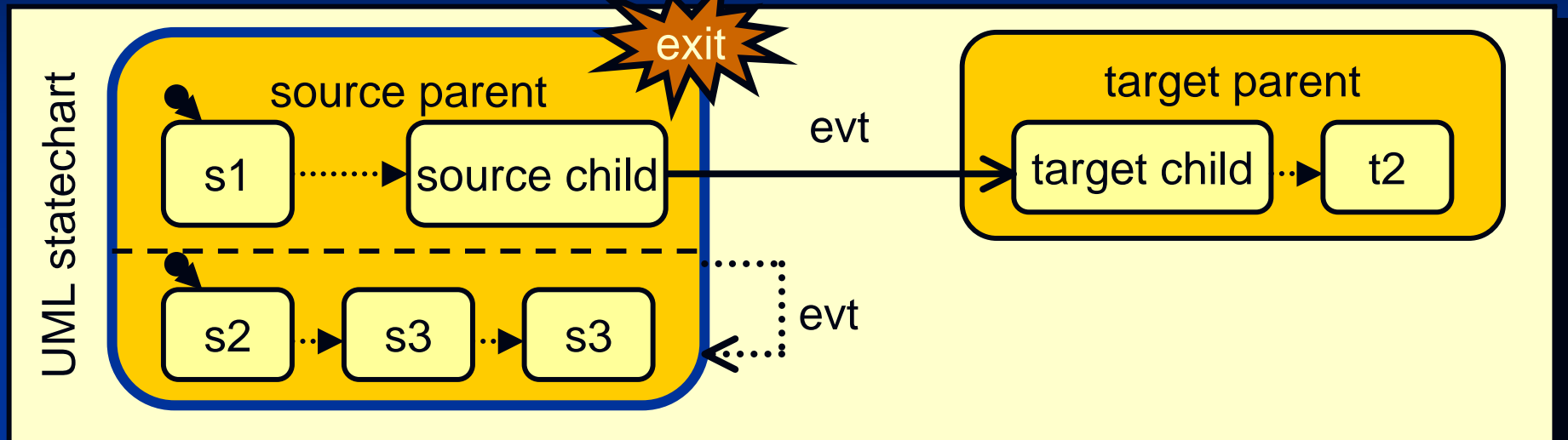
Reference information of the internal behavior



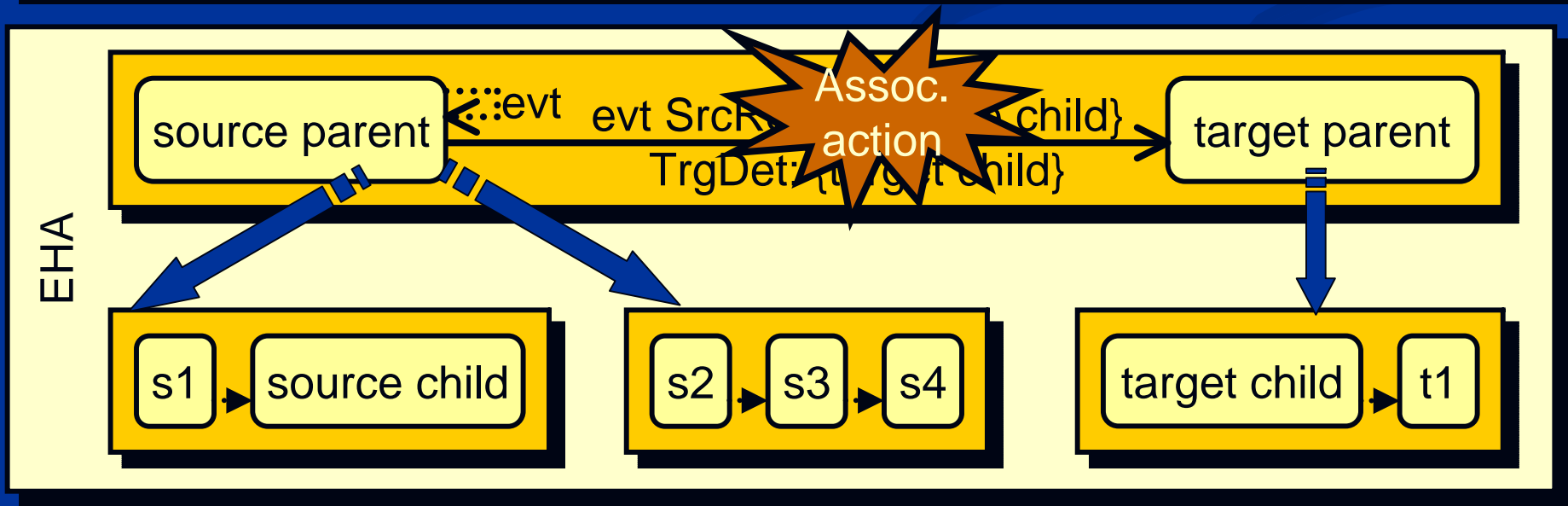
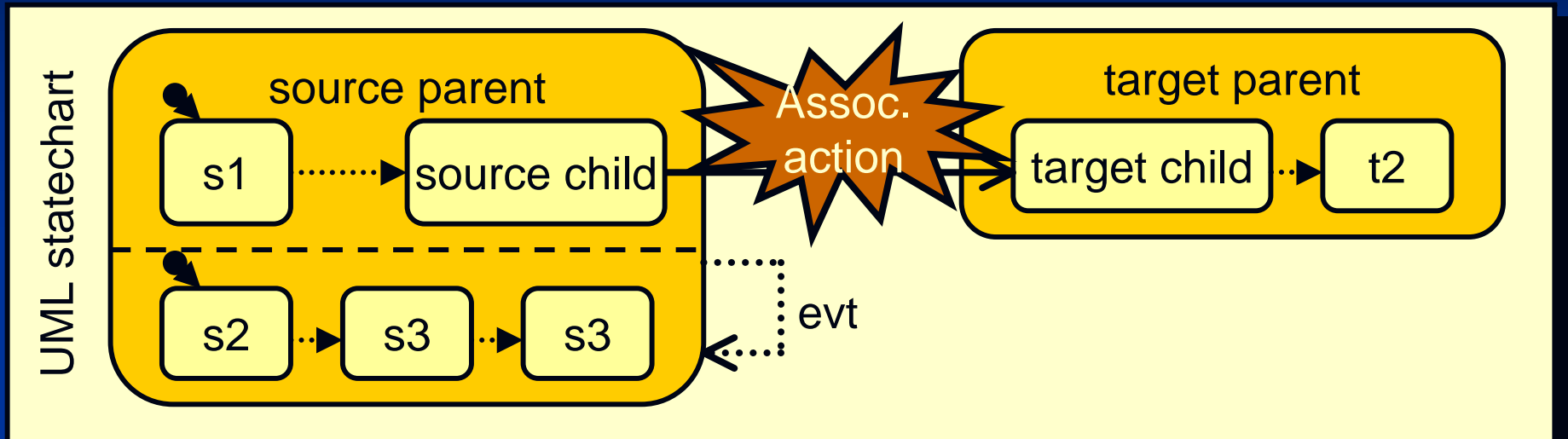
Reference information of the internal behavior



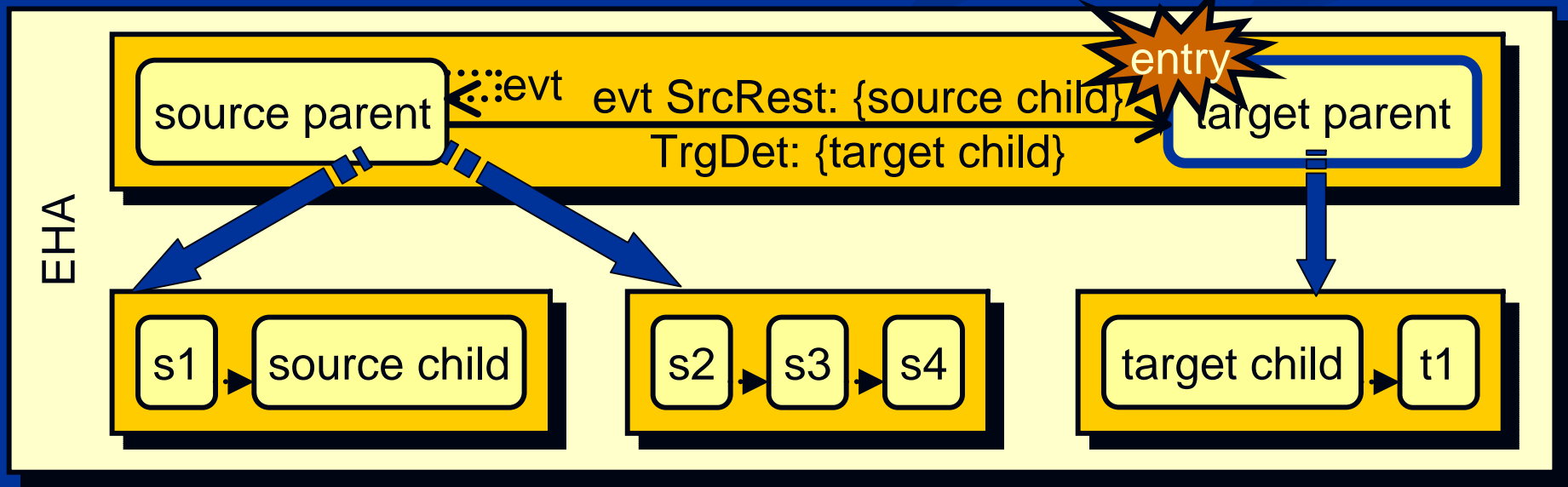
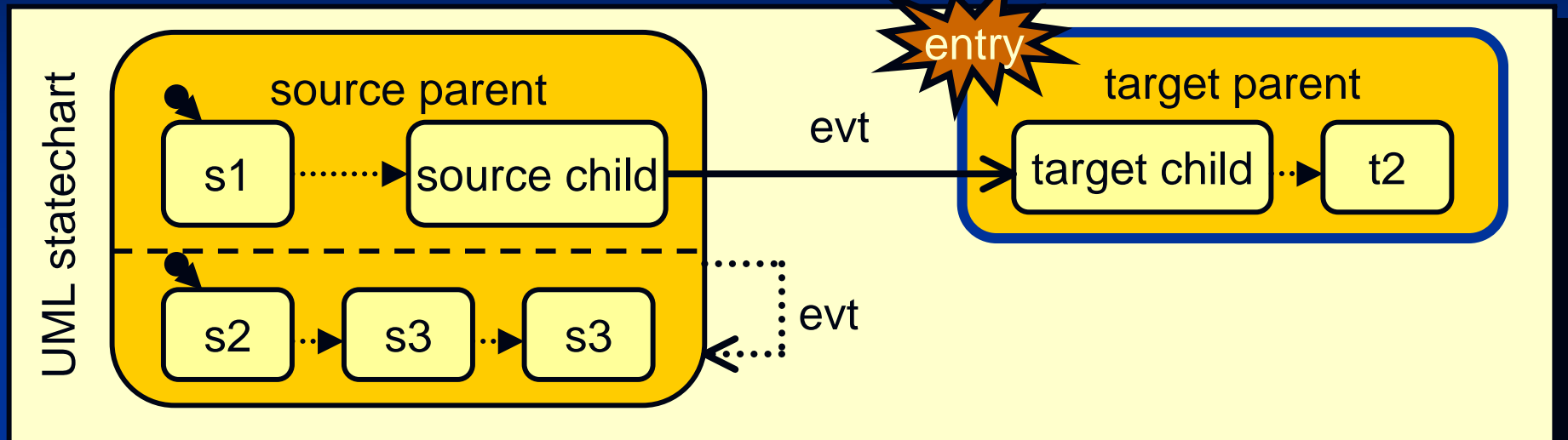
Reference information of the internal behavior



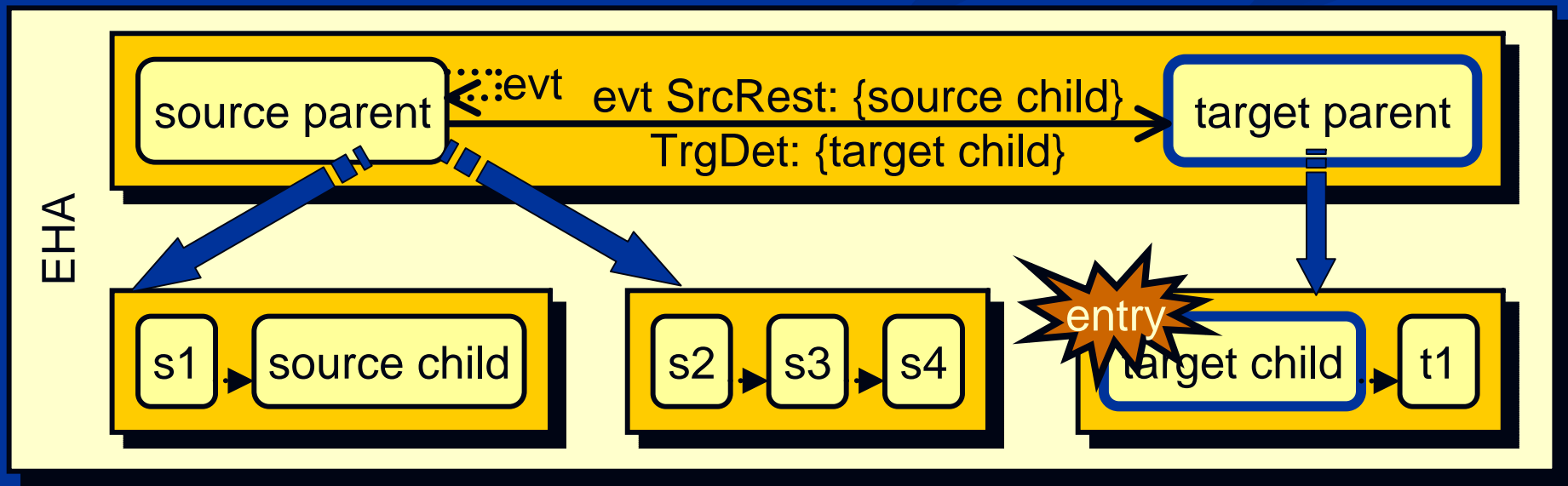
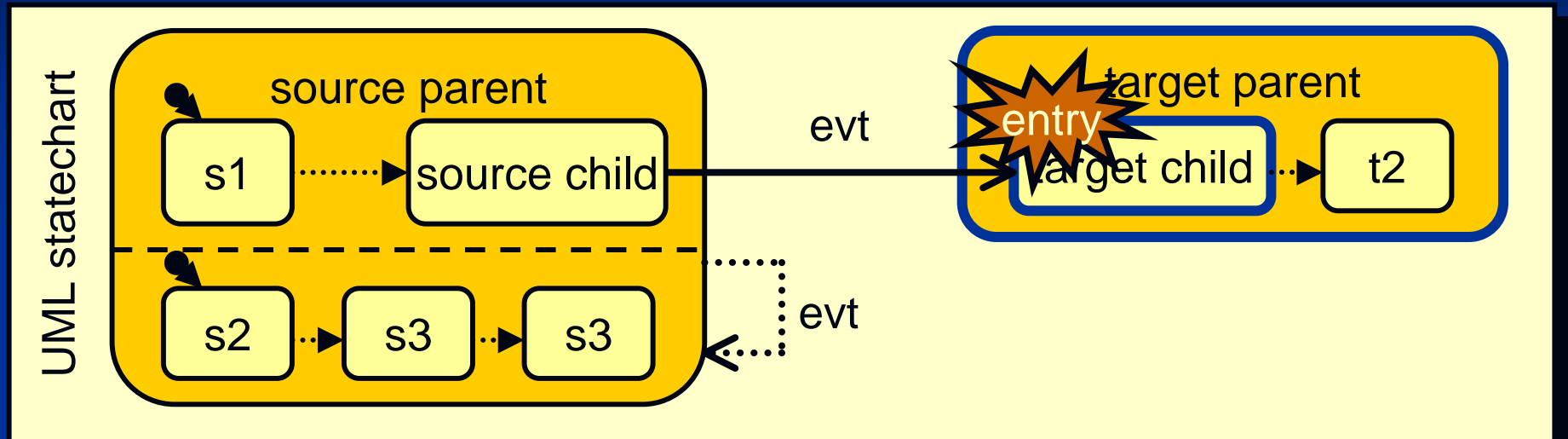
Reference information of the internal behavior



Reference information of the internal behavior



Reference information of the internal behavior



Abstract, high-level control-flow fault detection

Reference information:

- Automatically derived from the behavioral specification
- Capable of expressing state hierarchies, concurrent operation, etc.

Implementation of the monitor:

- Based on the operational semantics of the behavioral model
- Run-time checking of the behavior on the basis of the abstract reference model

Implementation of the instrumentation:

- Providing information to the monitor about the internal behavior
- Configurable, transparent and automatically applied

Abstract, high-level control-flow fault detection

Reference information:

- Automatically derived from the behavioral specification
- Capable of expressing state hierarchies, concurrent operation, etc.

Implementation of the monitor:

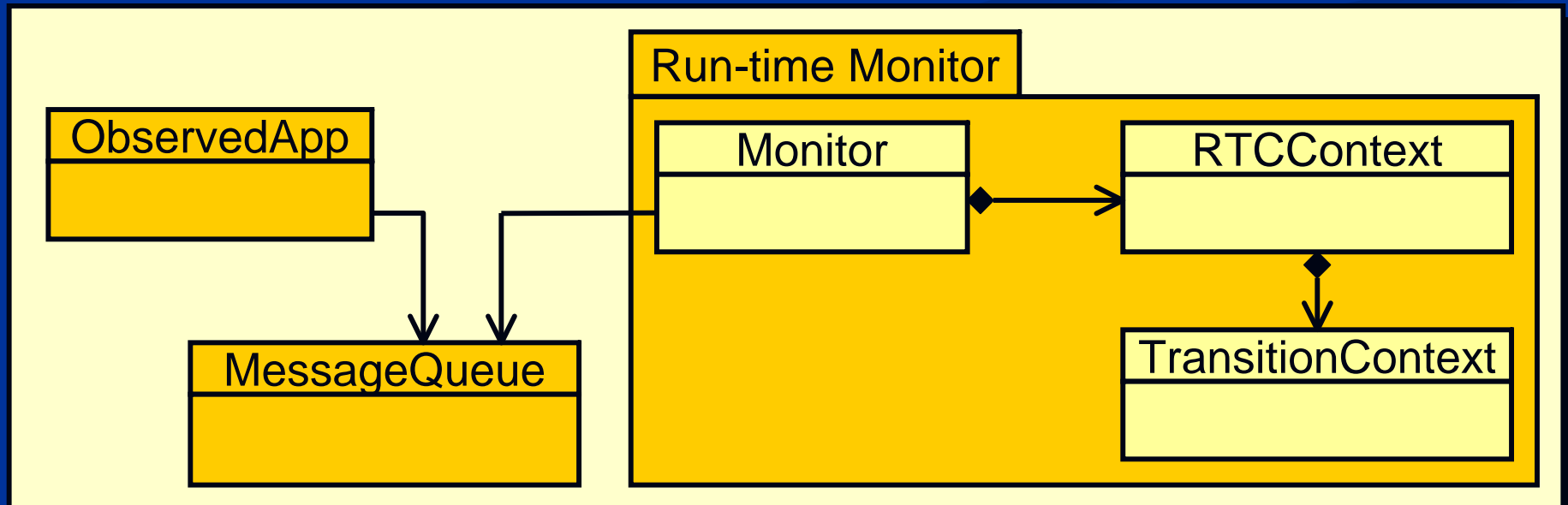
- Based on the operational semantics of the behavioral model
- Run-time checking of the behavior on the basis of the abstract reference model

Implementation of the instrumentation:

- Providing information to the monitor about the internal behavior
- Configurable, transparent and automatically applied

Checking the internal behavior

- Structural decomposition:
 - *Run-to-completion and transition contexts*
- Specification of contexts:
 - Protocol state machines (statecharts)

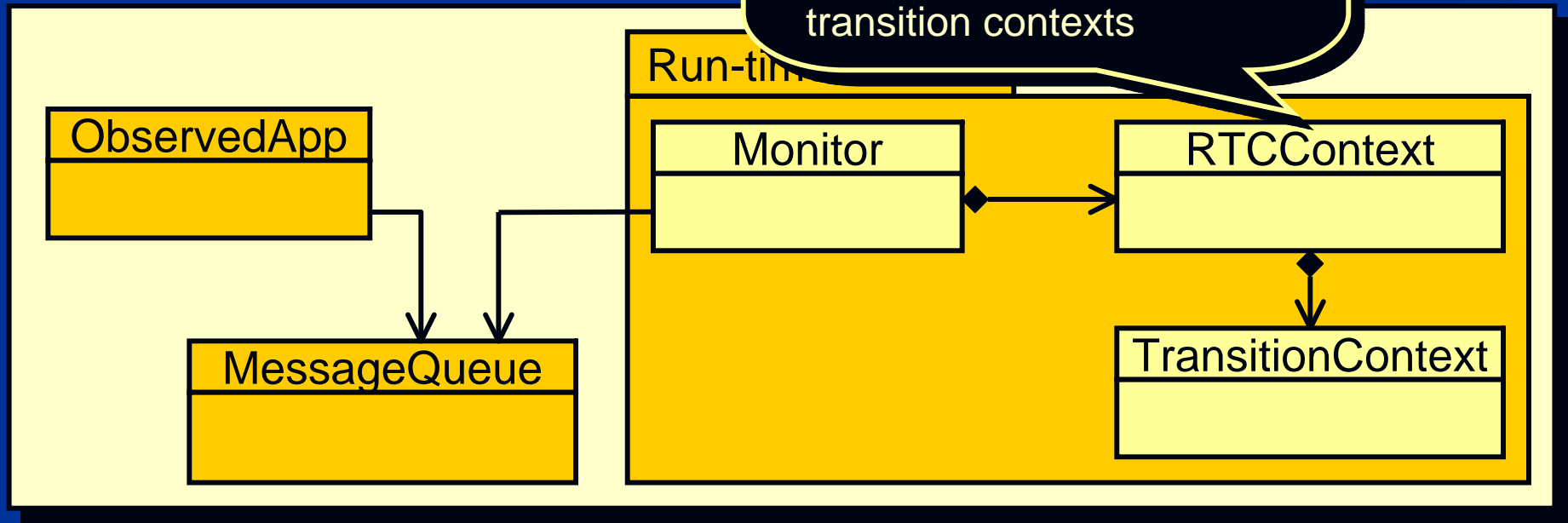


Checking the internal behavior

- Structural decomposition:
 - *Run-to-completion and transition contexts*
- Specification of components
 - Protocol state machine

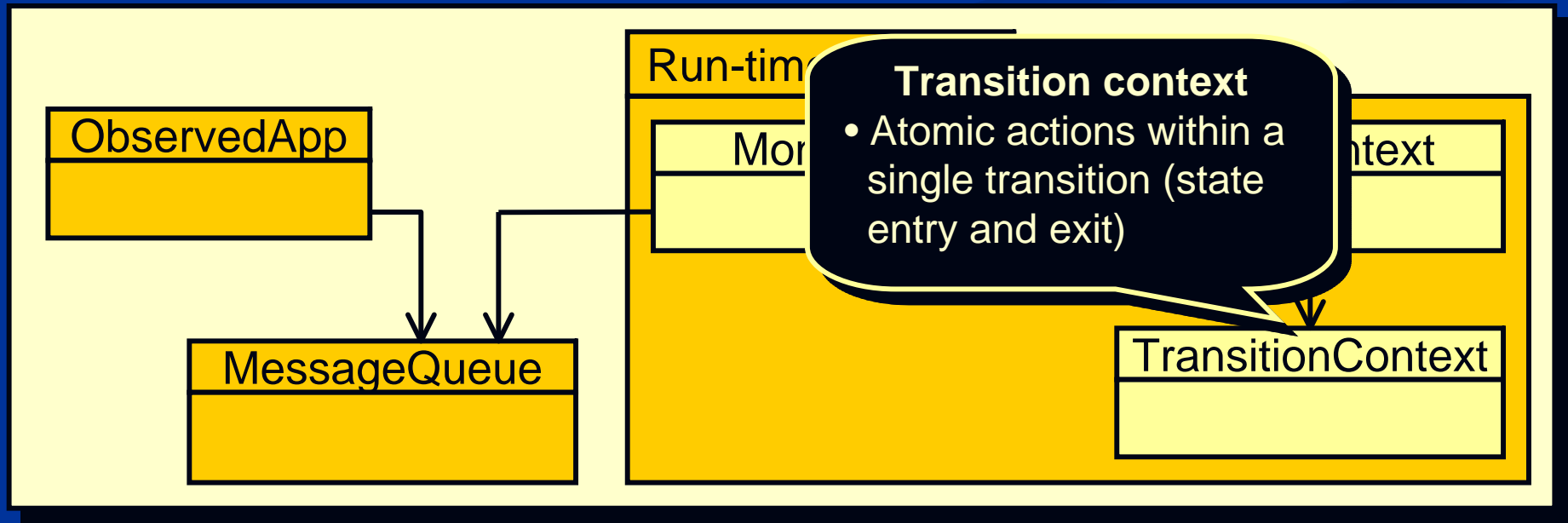
Run-to-completion context

- Initialization
- Start and finish of event processing
- Dispatching messages to transition contexts



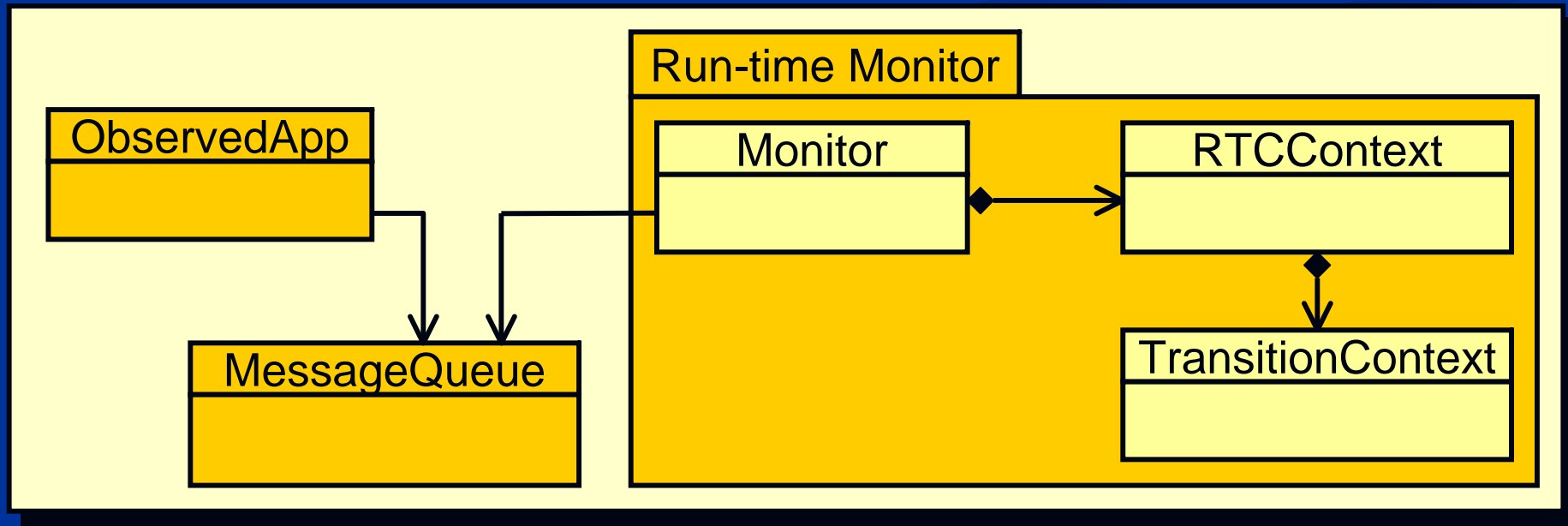
Checking the internal behavior

- Structural decomposition:
 - *Run-to-completion and transition contexts*
- Specification of contexts:
 - Protocol state machines (statecharts)



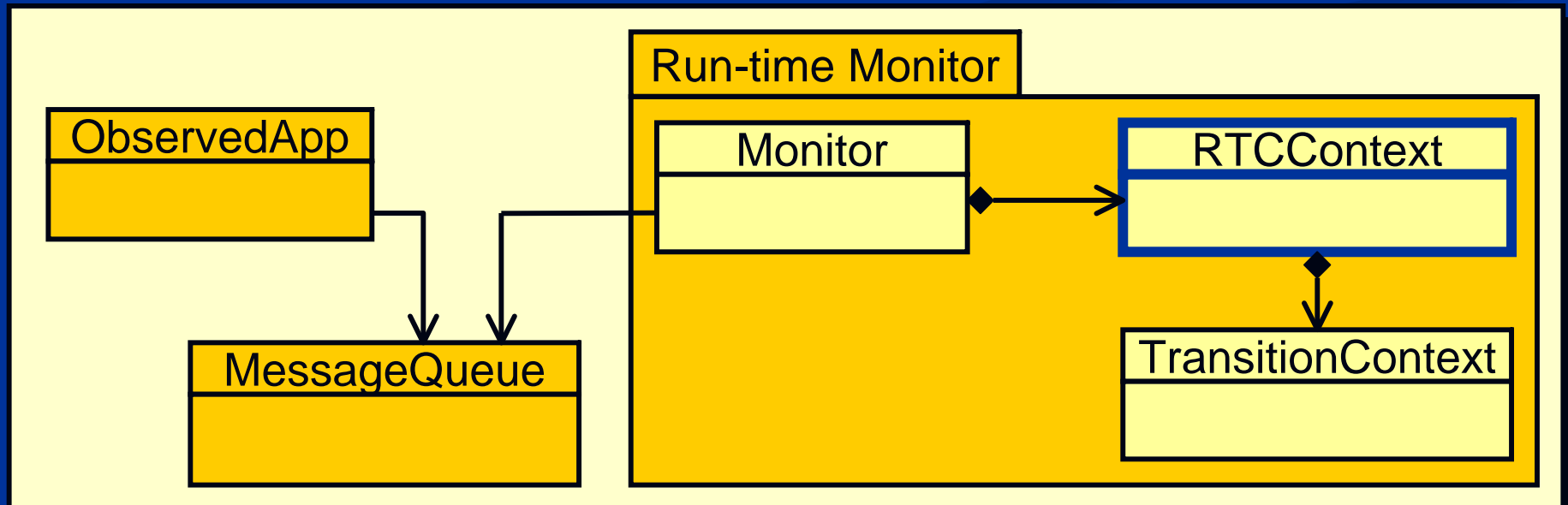
Checking the internal behavior

- Structural decomposition:
 - *Run-to-completion and transition contexts*
- Specification of contexts:
 - Protocol state machines (statecharts)

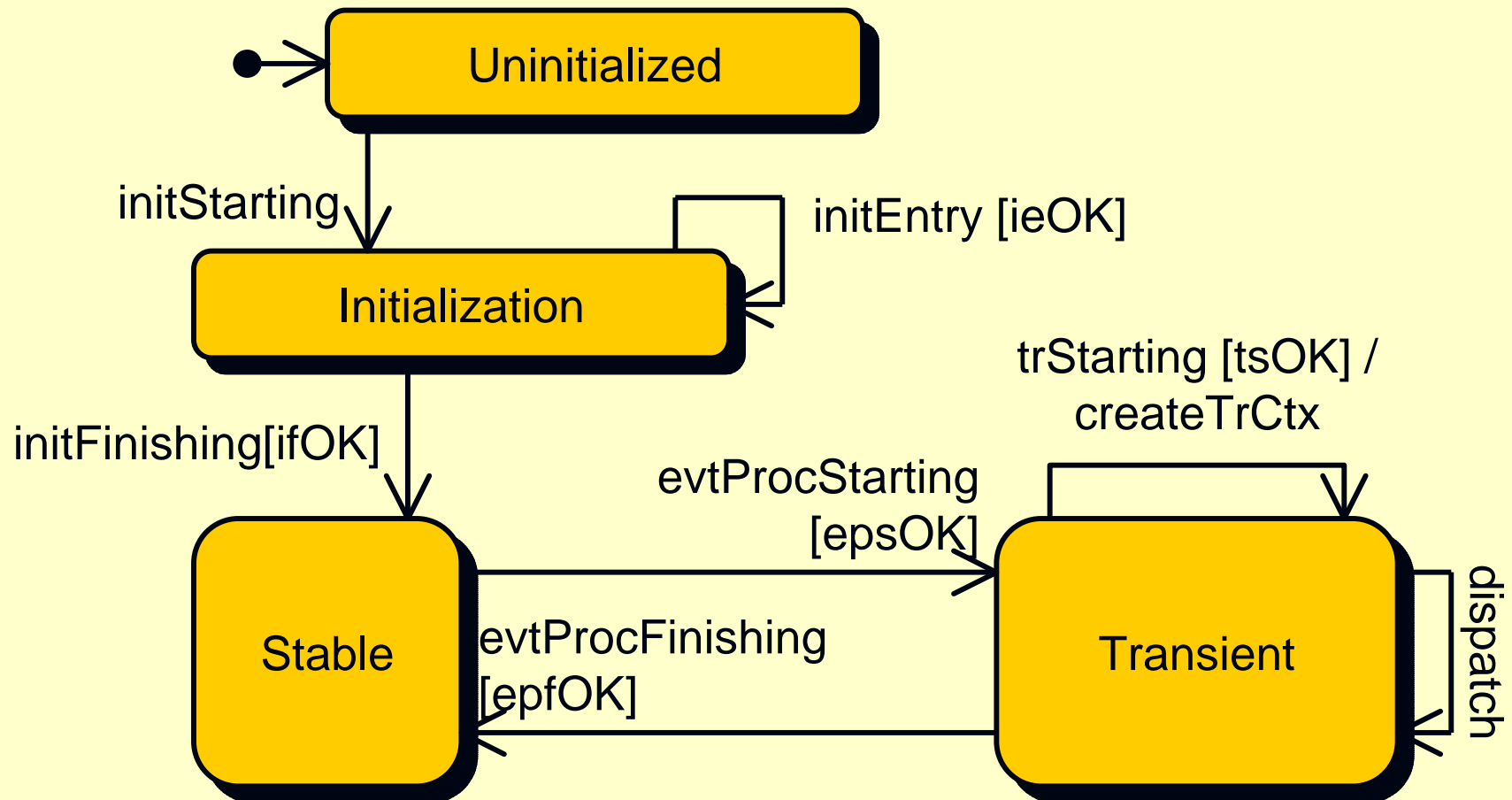


Checking the internal behavior

- Structural decomposition:
 - *Run-to-completion and transition contexts*
- Specification of contexts:
 - Protocol state machines (statecharts)



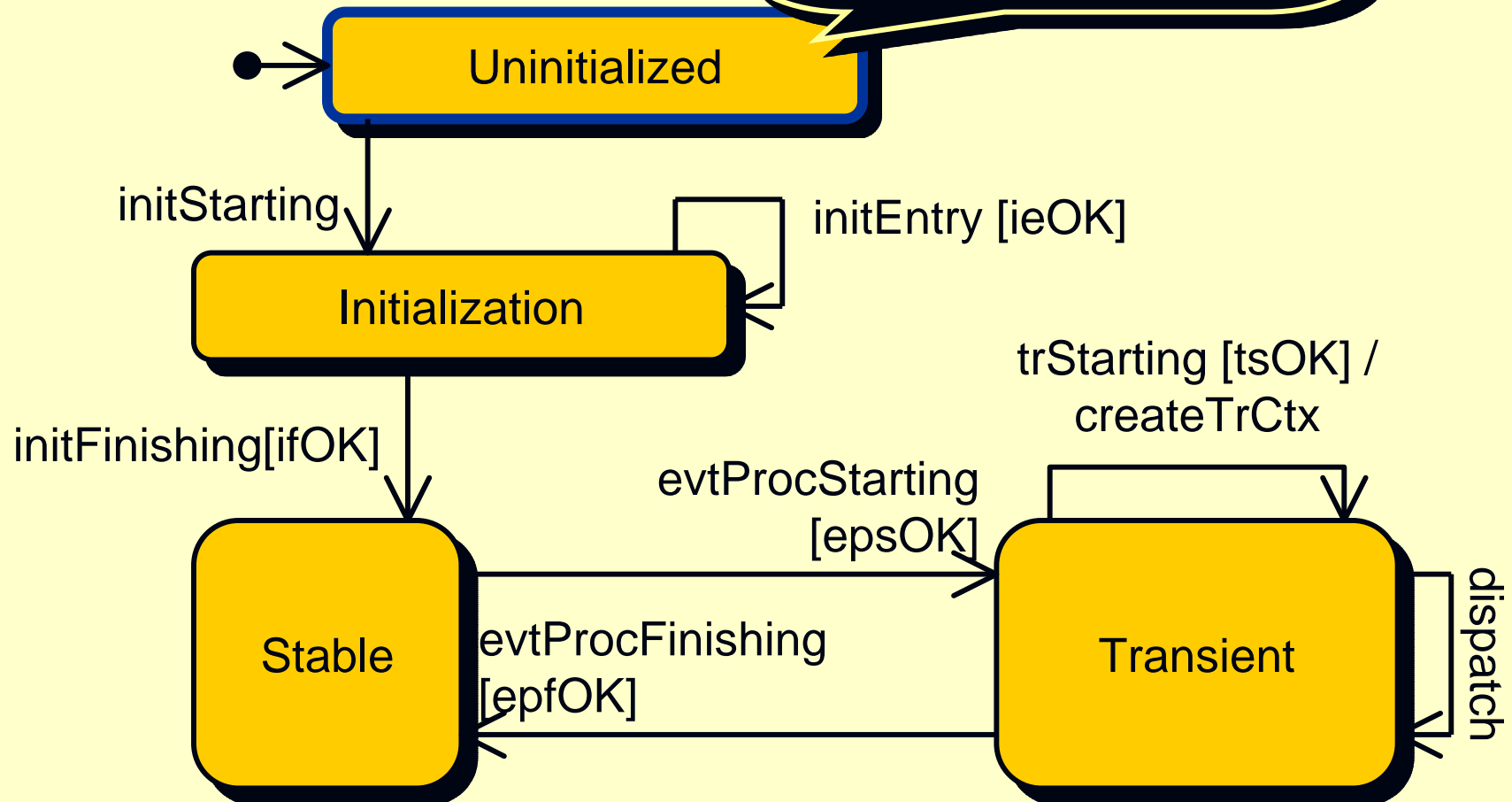
Run-to-completion context



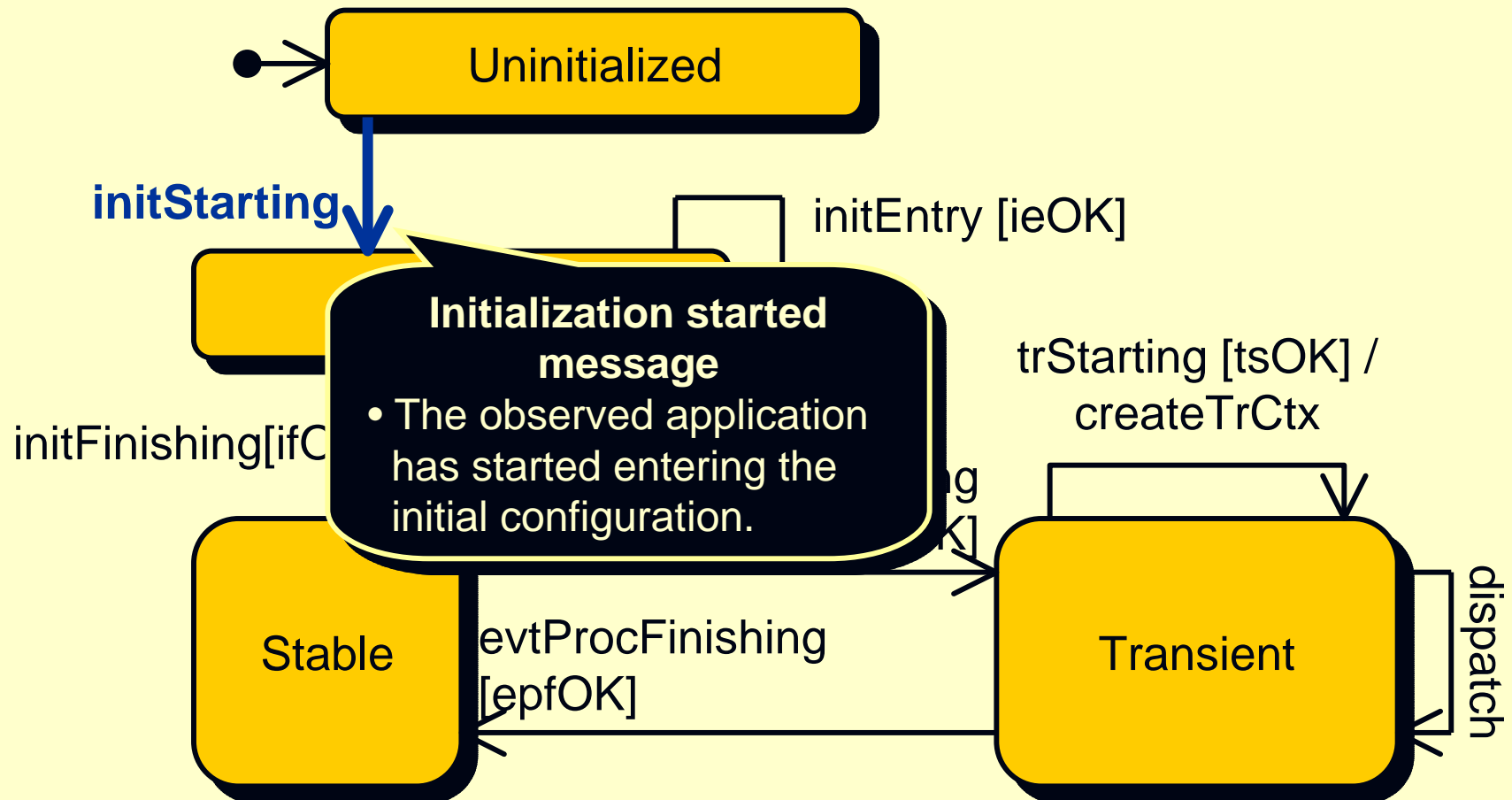
Run-to-compl

Before initialization

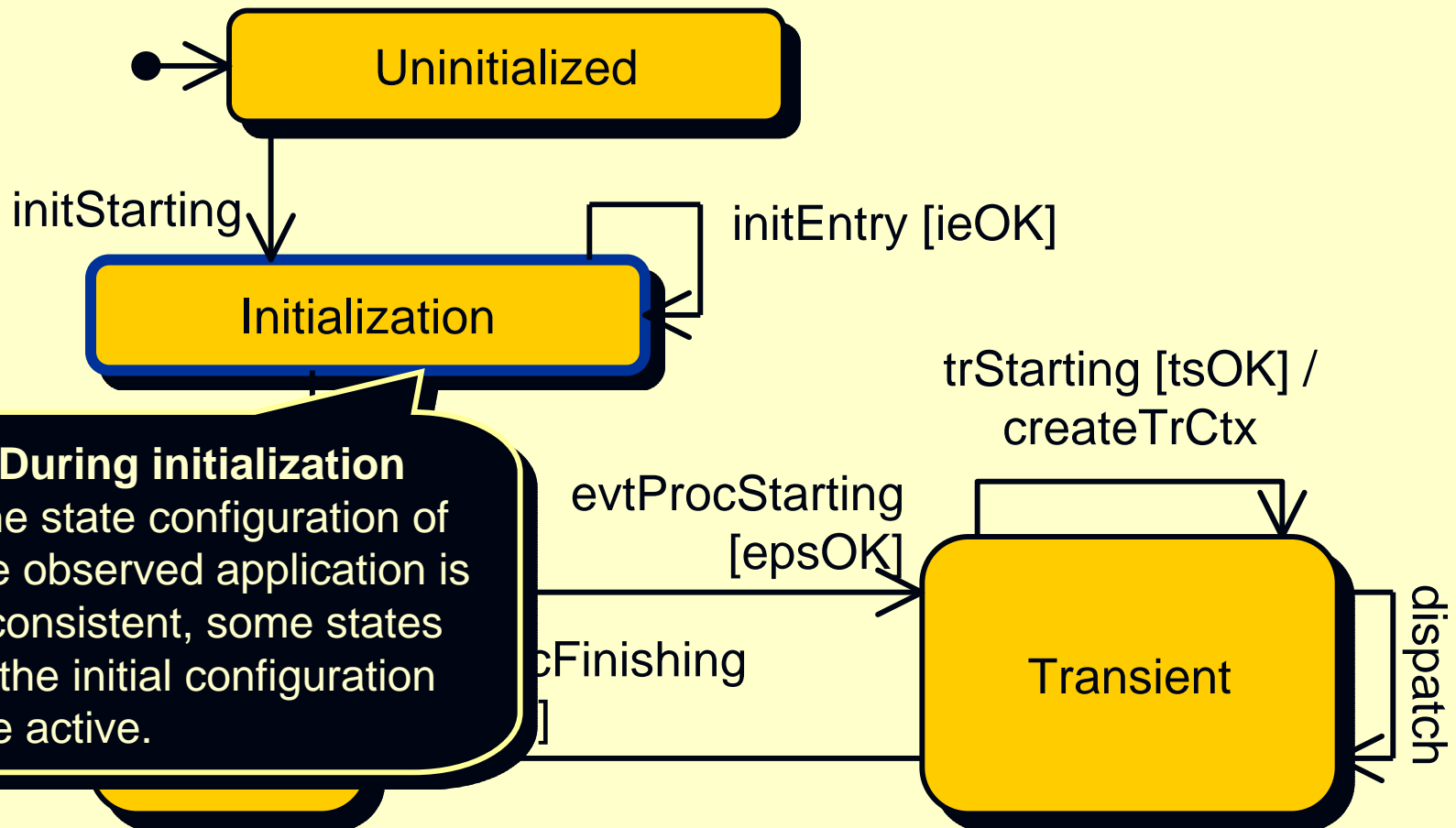
- The state configuration of the observed application is inconsistent, none of the states is active.



Run-to-completion context



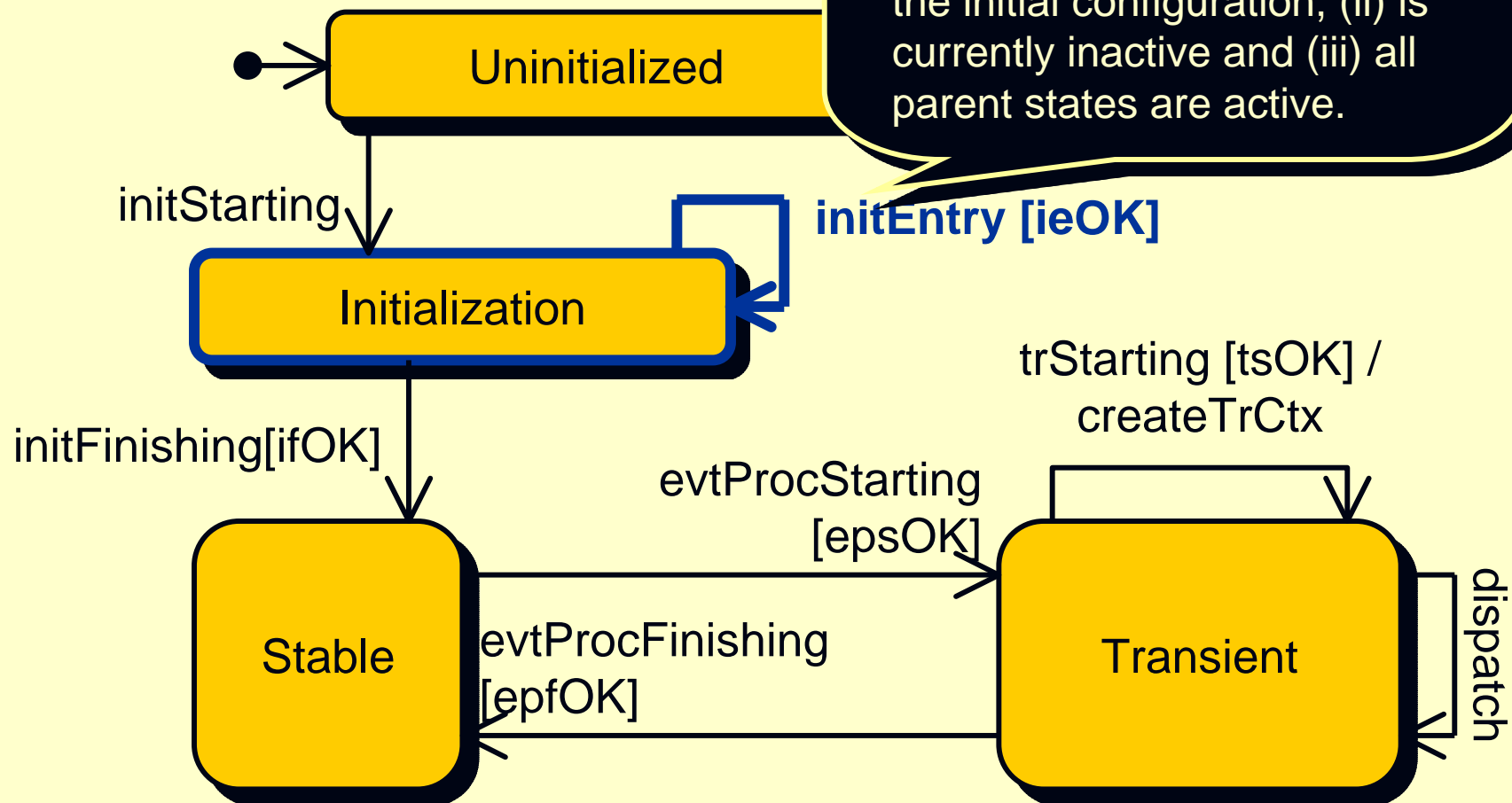
Run-to-completion context



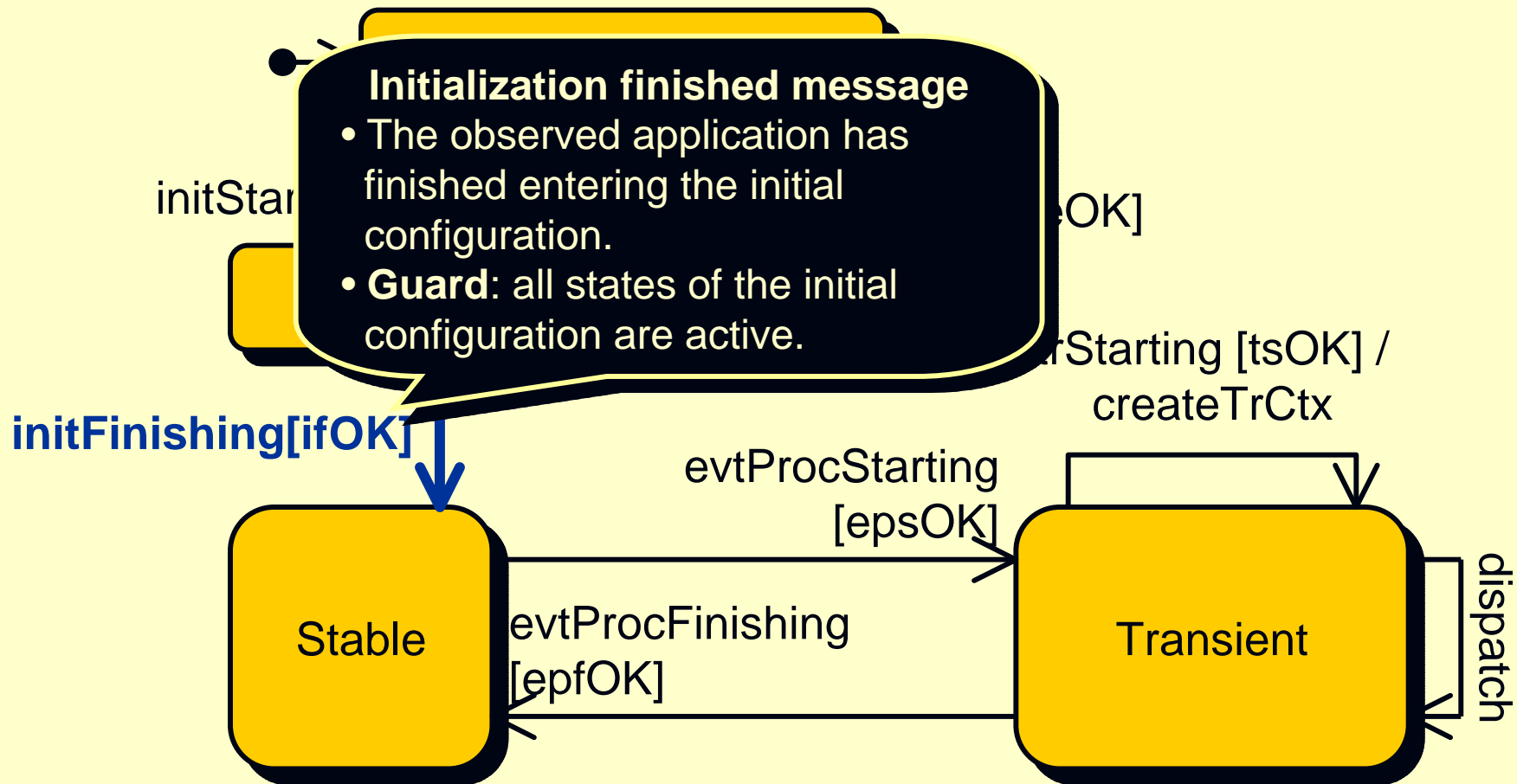
Run-to-complet

Entry during initialization

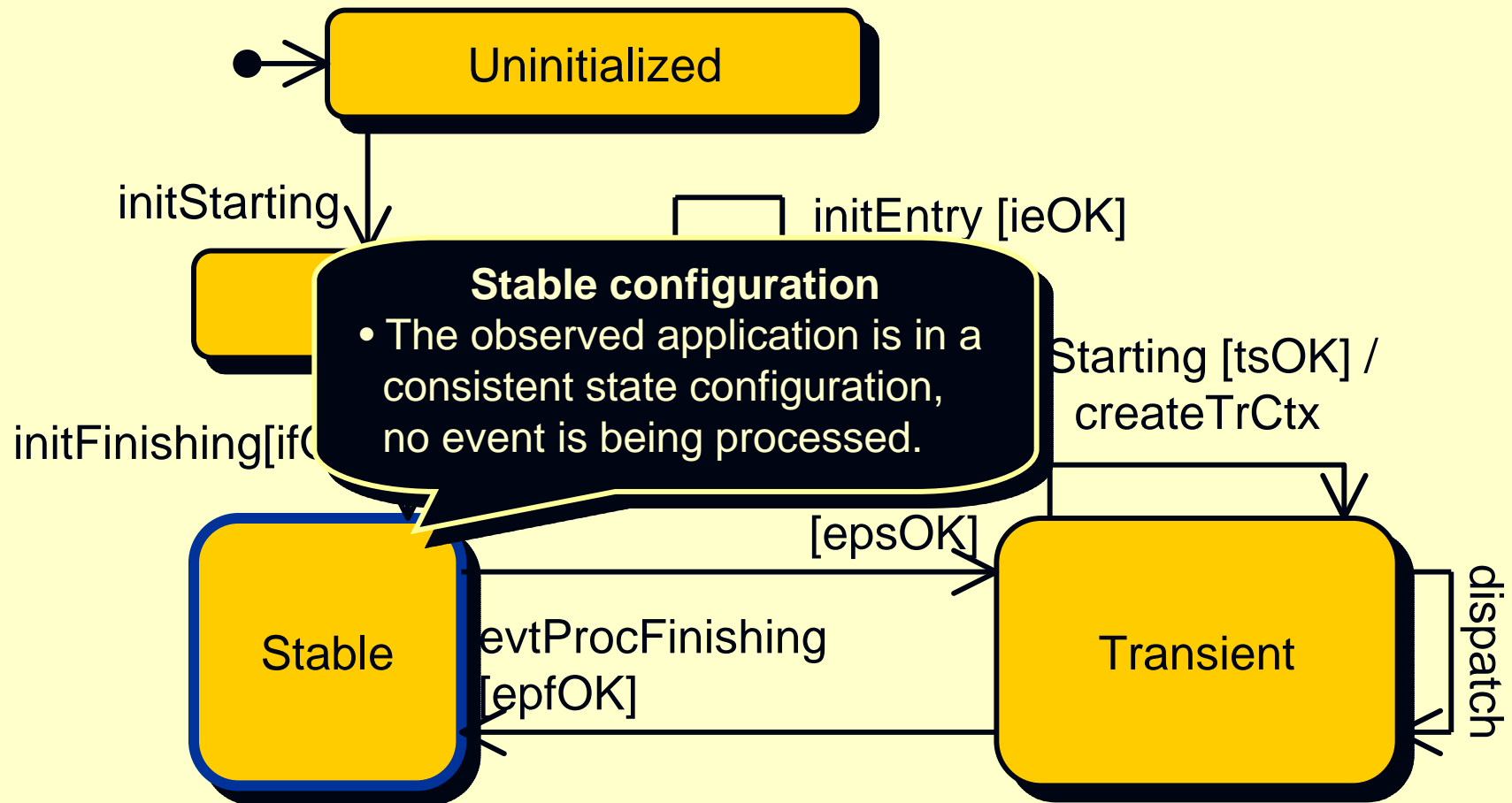
- The observed application has entered a state.
- **Guard:** (i) the state belongs to the initial configuration, (ii) is currently inactive and (iii) all parent states are active.



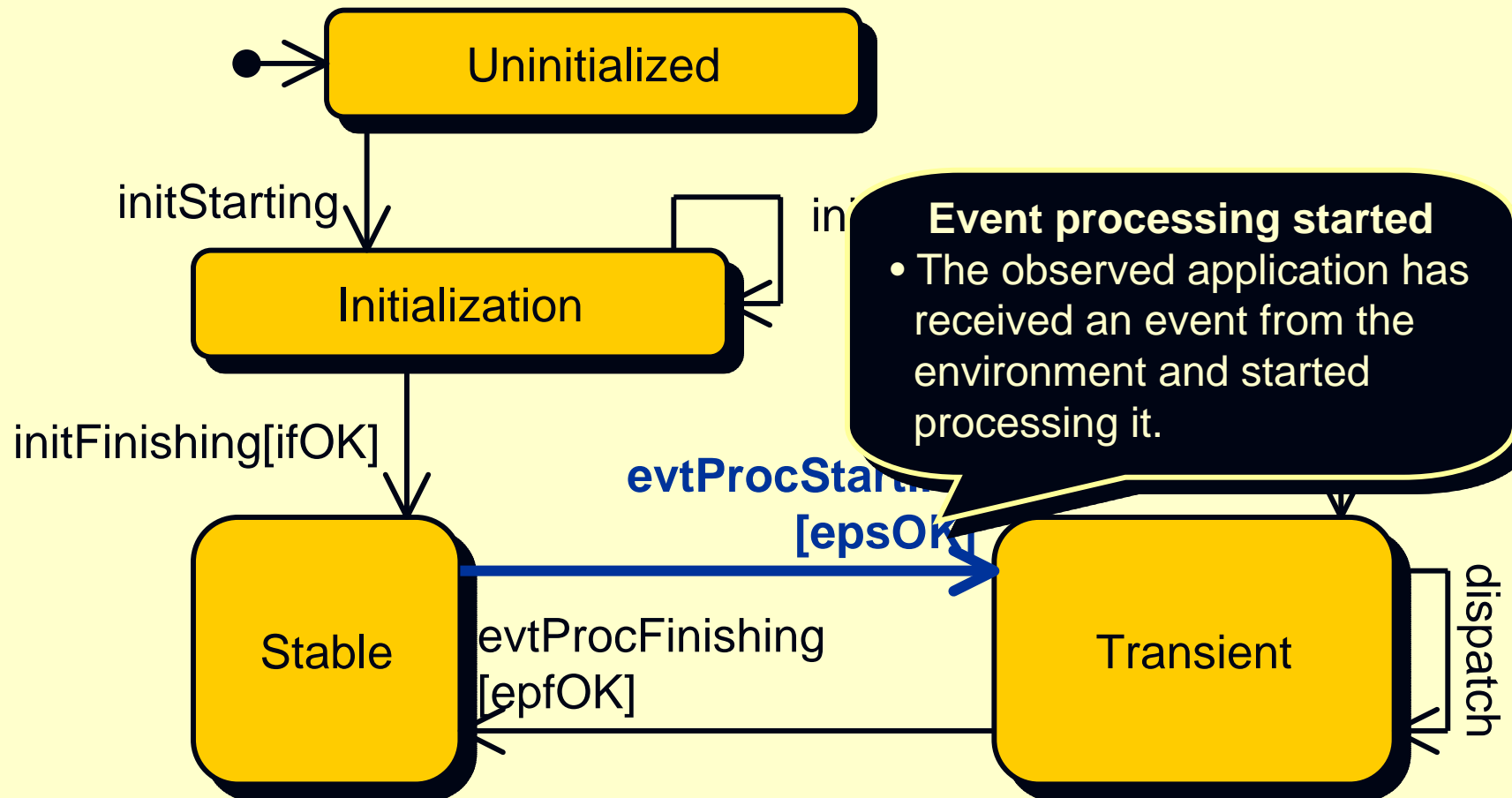
Run-to-completion context



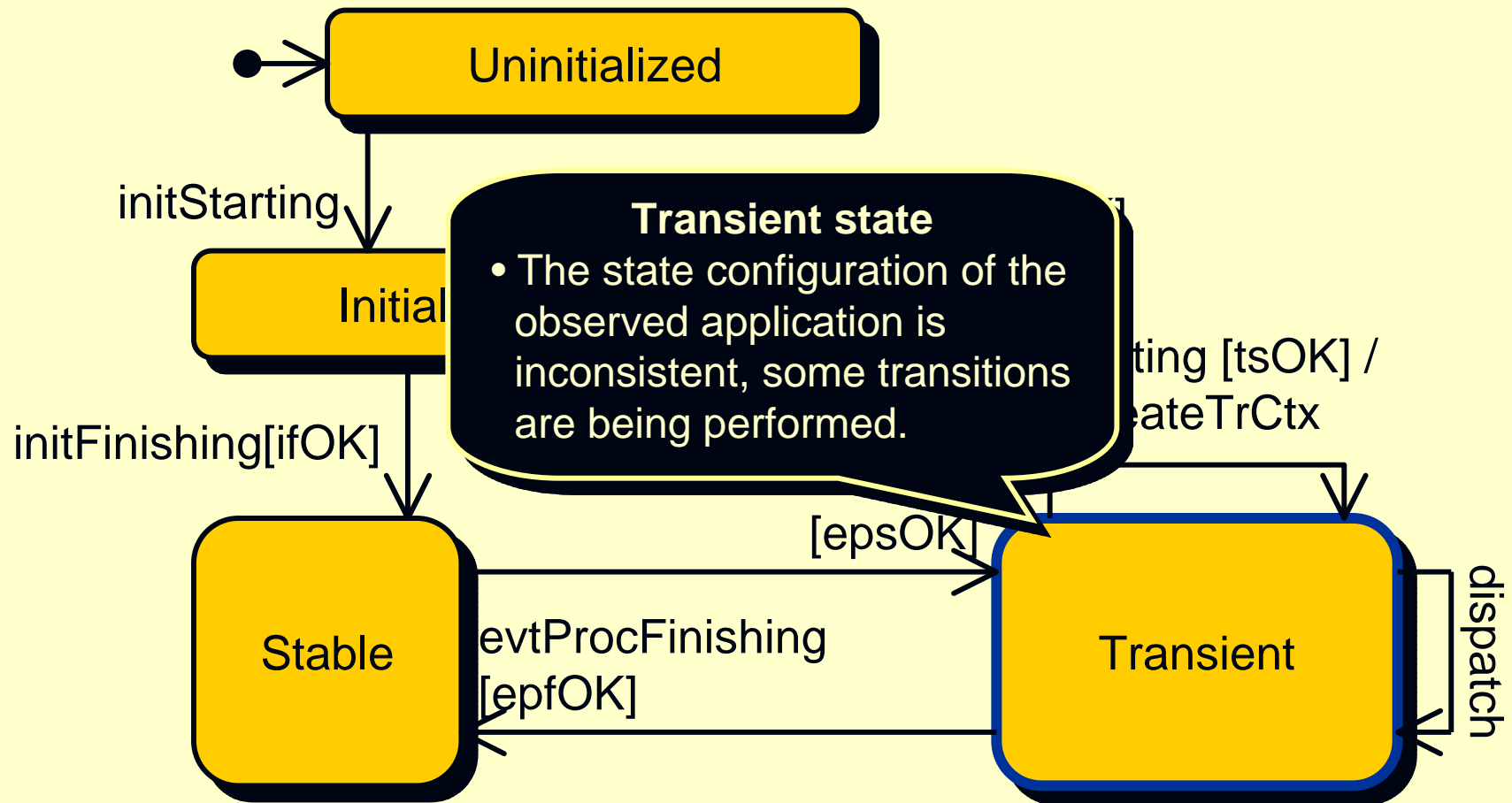
Run-to-completion context



Run-to-completion context



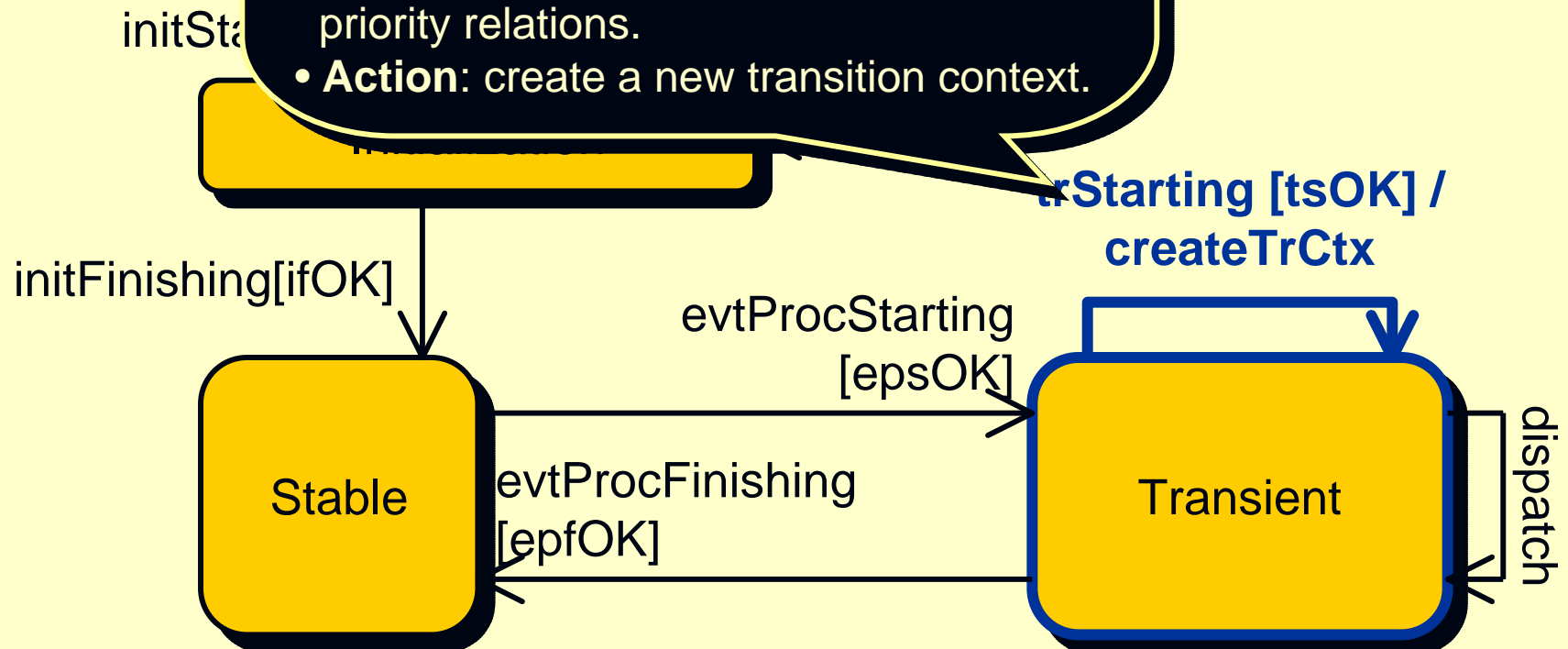
Run-to-completion context



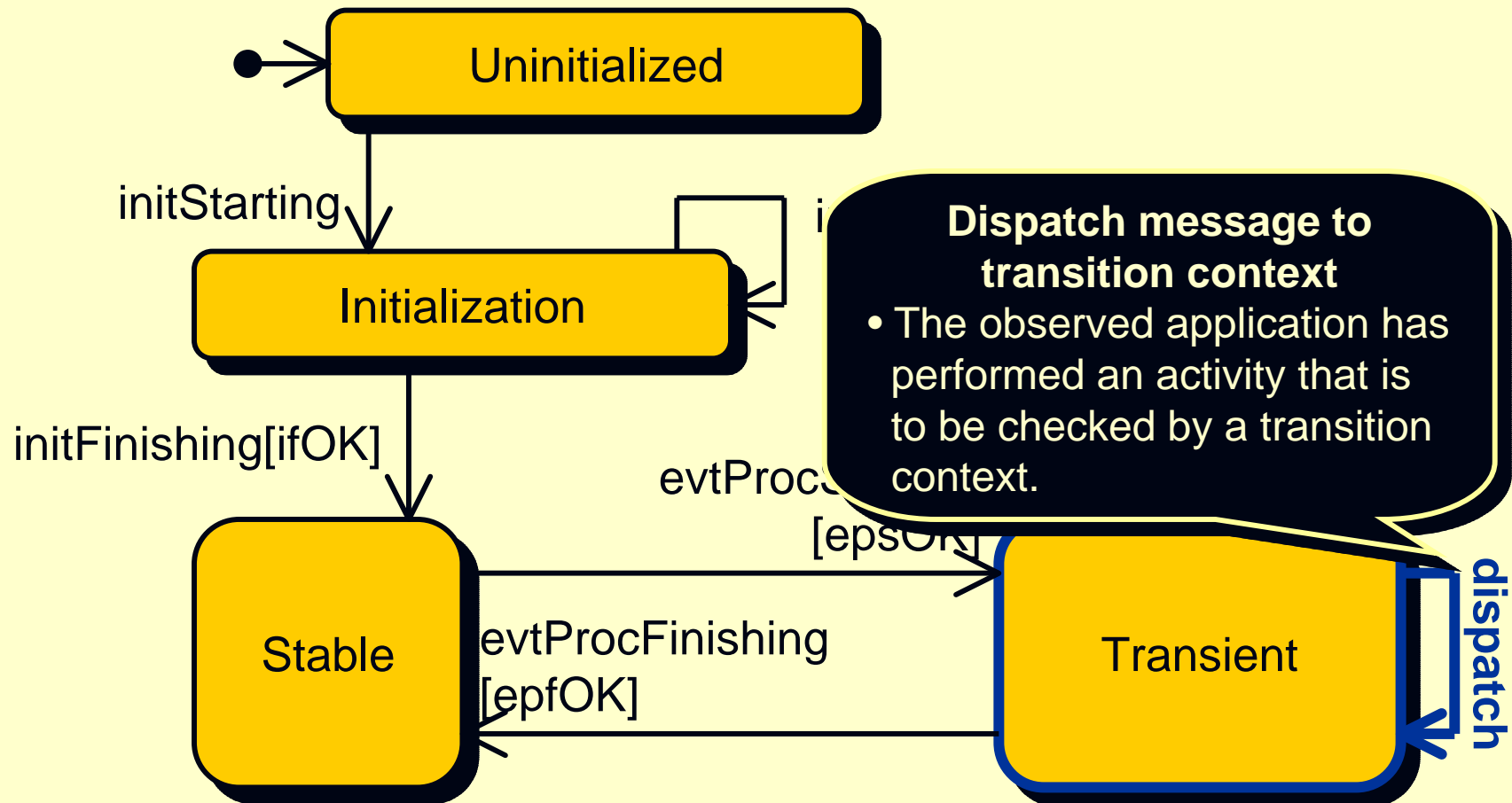
Run context

Transition selected for firing

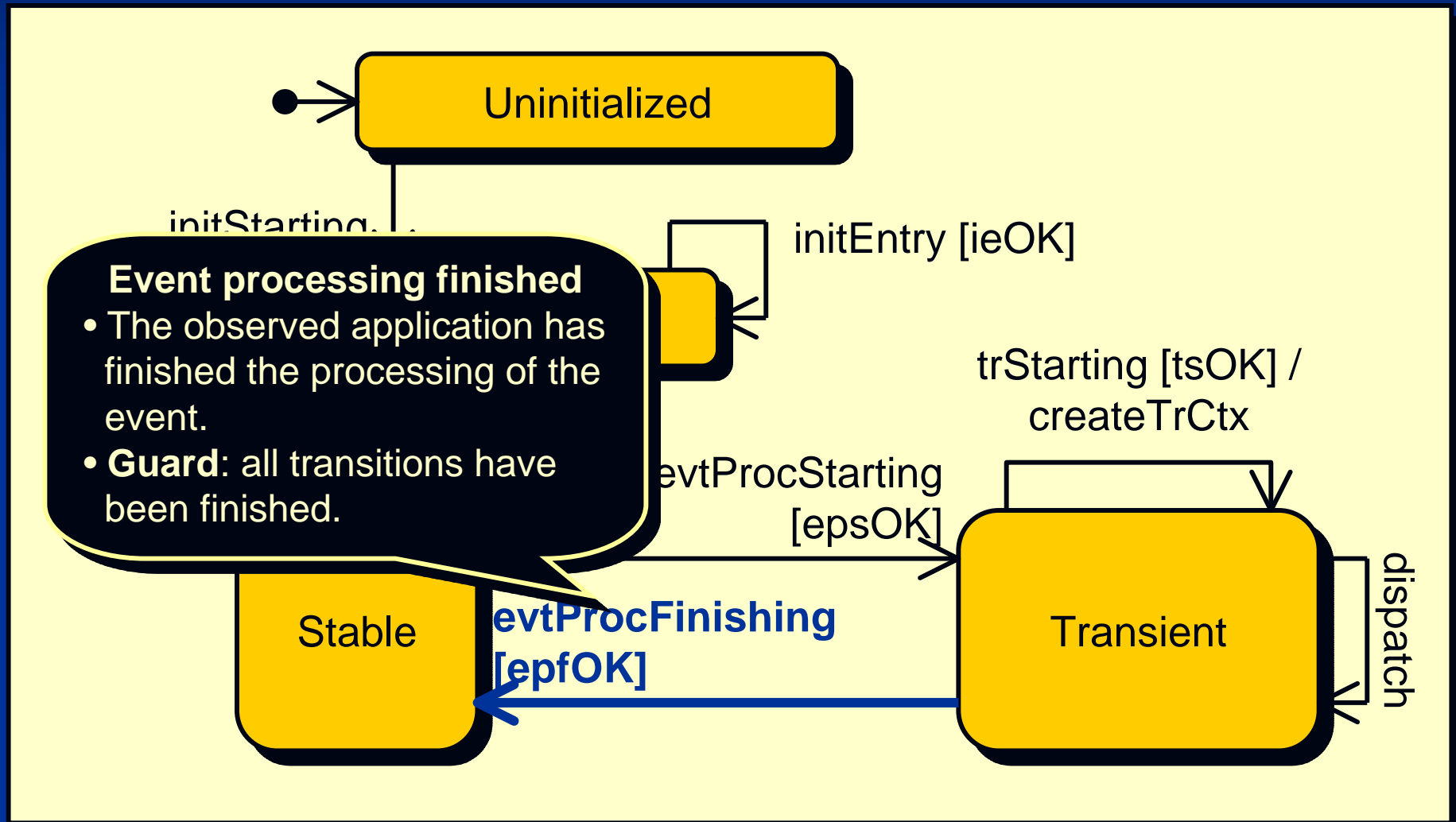
- The observed application has selected a transition to be fired during the event processing.
- **Guard:** (i) triggered by the currently processed event, (ii) source state and ones in the source restriction set are active and (iii) does not violate the priority relations.
- **Action:** create a new transition context.



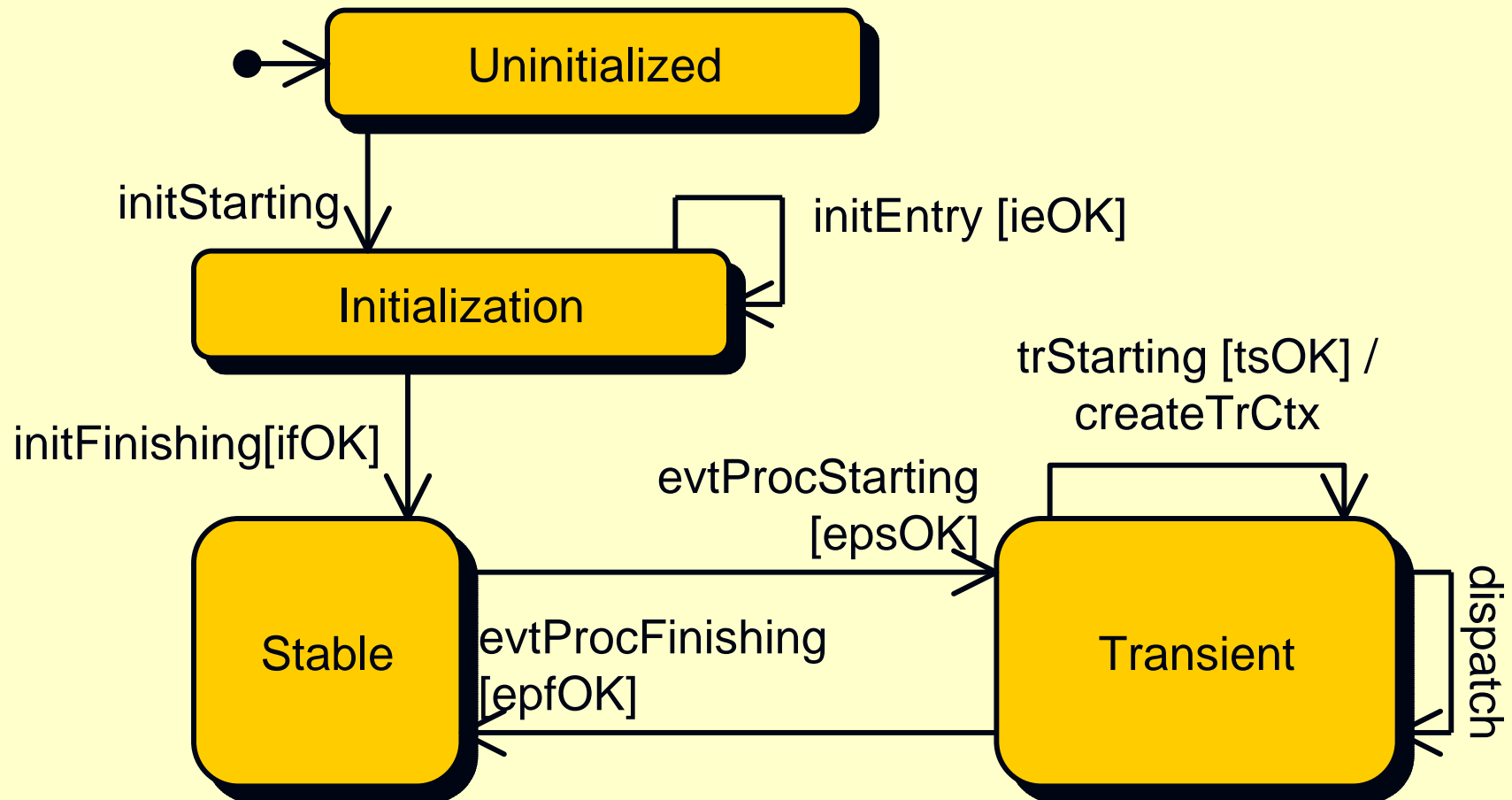
Run-to-completion context



Run-to-completion context



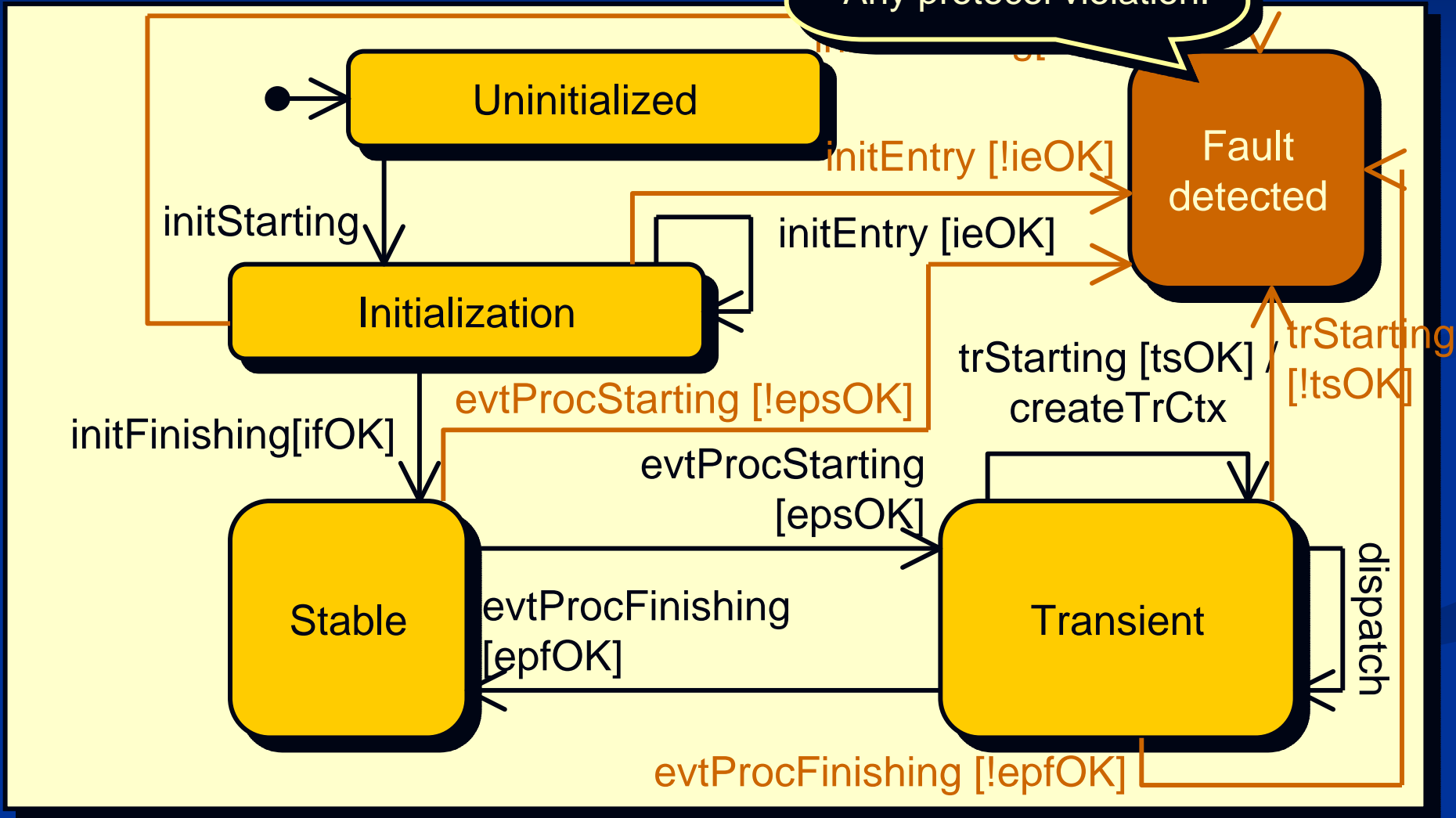
Run-to-completion context



Run-to-completeness

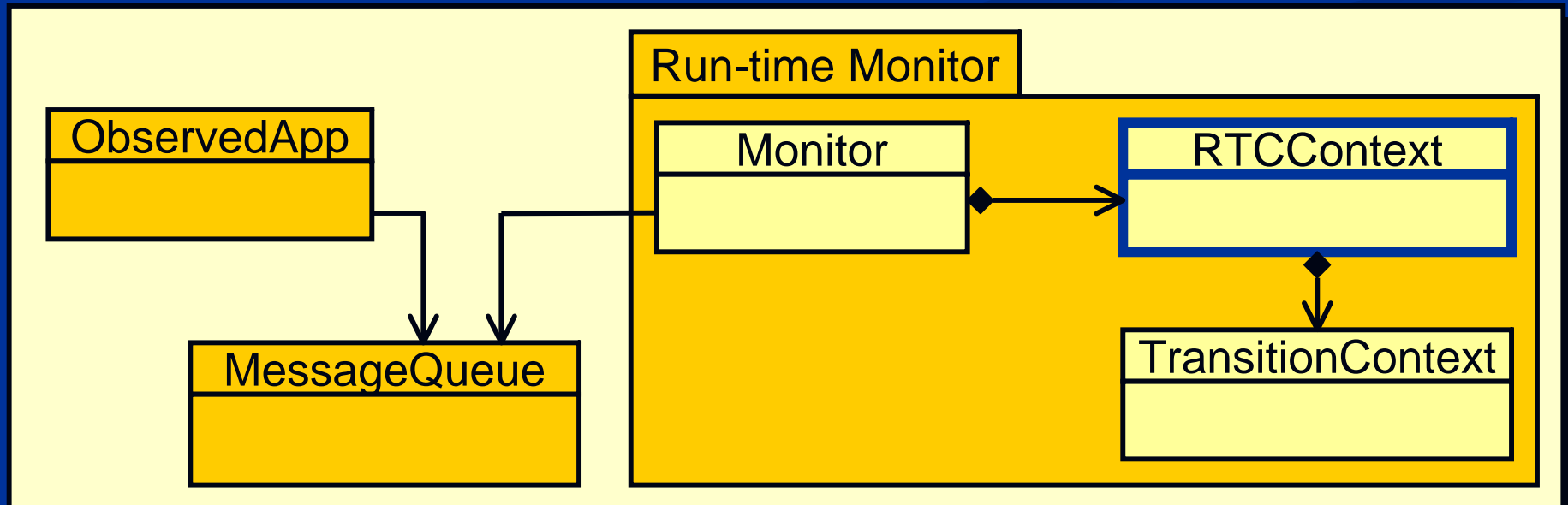
Fault detected

- Any guard evaluated to false.
- Any protocol violation.



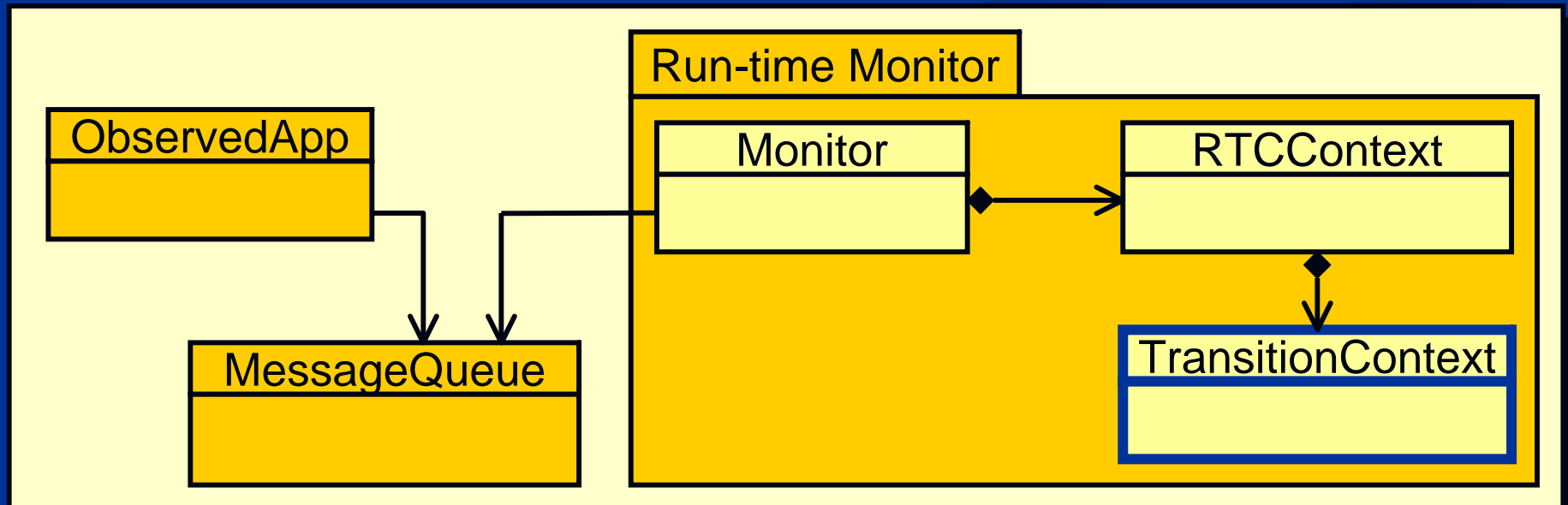
Checking the internal behavior

- Structural decomposition:
 - *Run-to-completion and transition contexts*
- Specification of contexts:
 - Protocol state machines (statecharts)

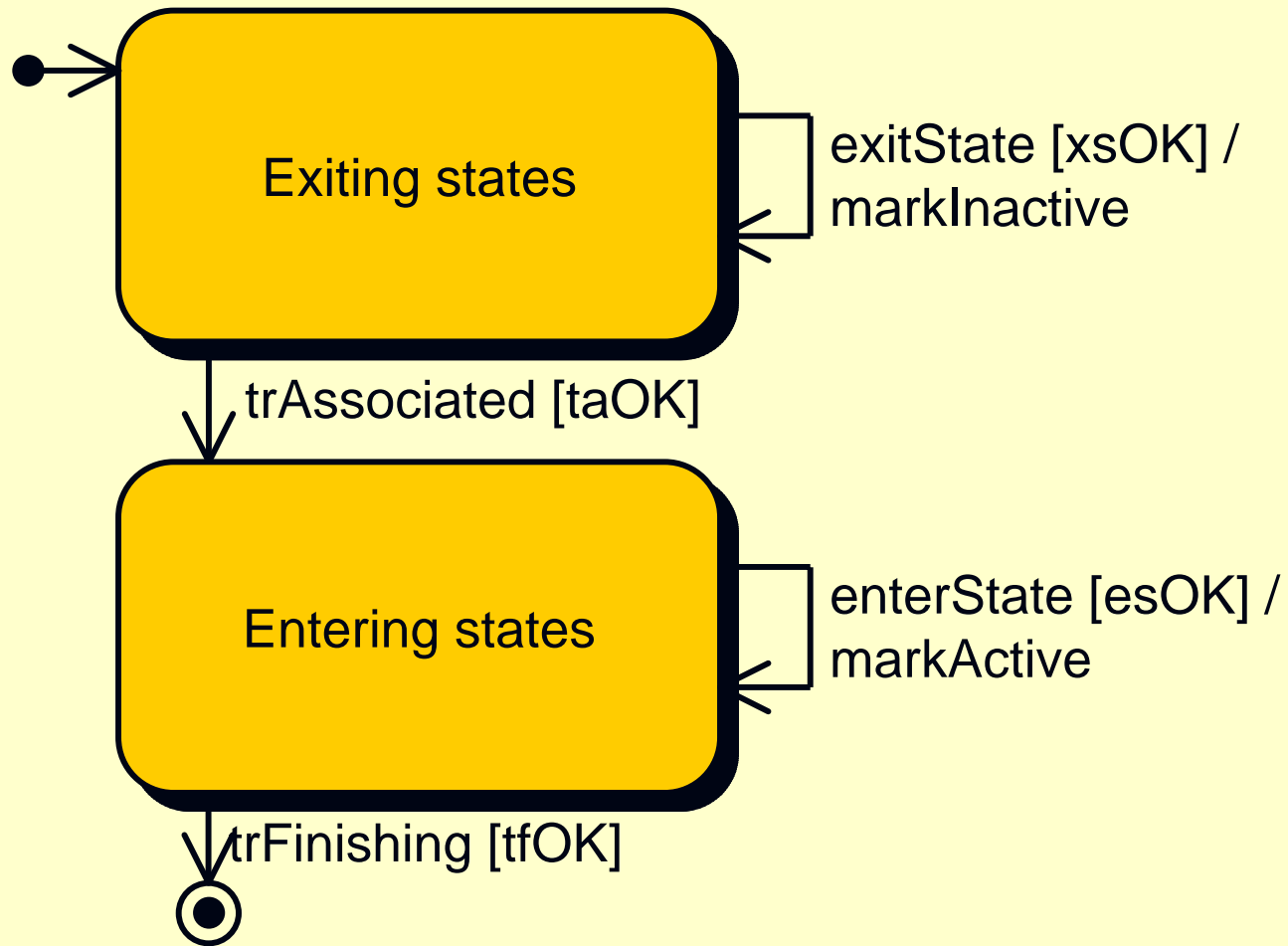


Checking the internal behavior

- Structural decomposition:
 - *Run-to-completion and transition contexts*
- Specification of contexts:
 - Protocol state machines (statecharts)



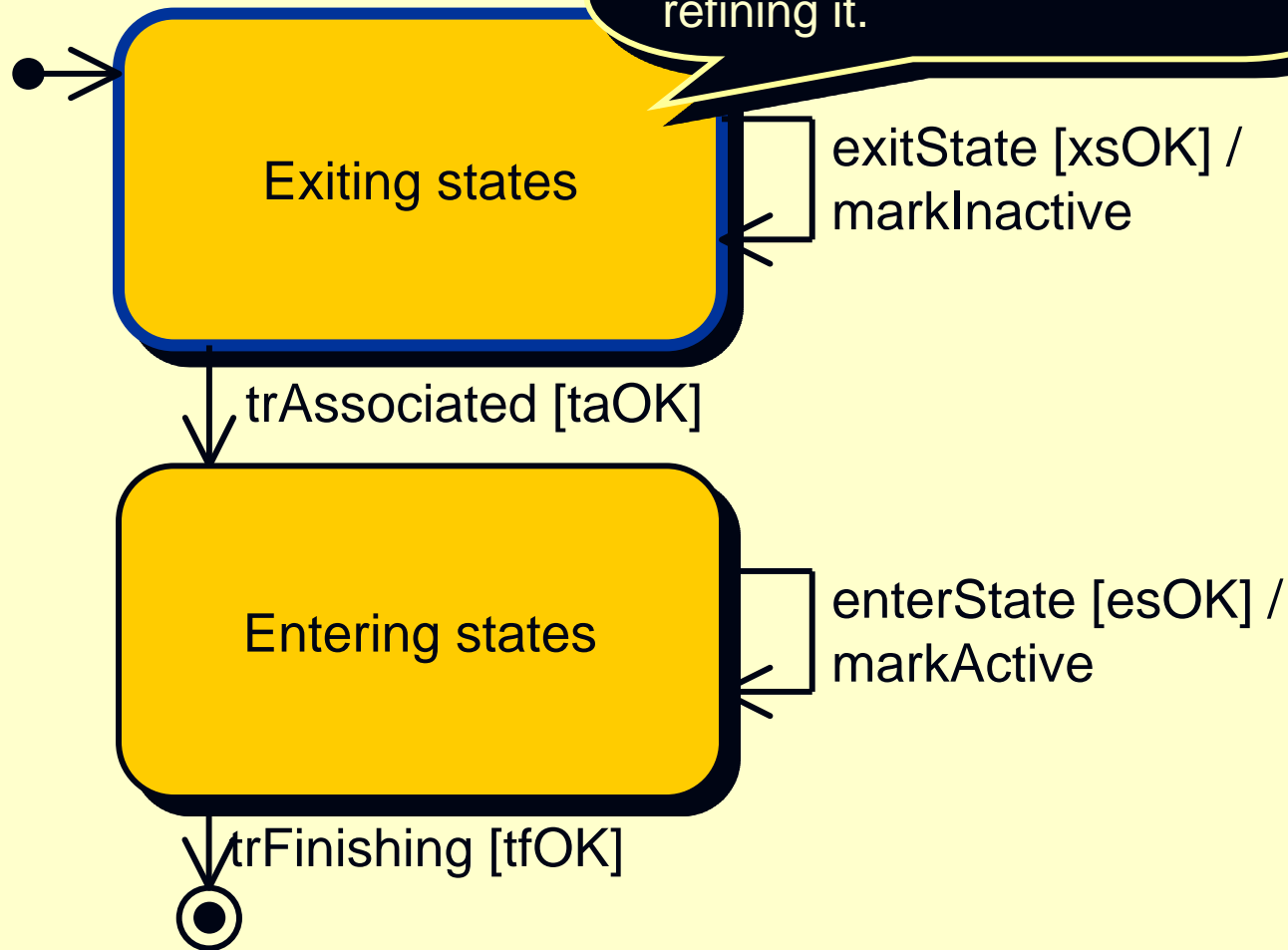
Transition context



Transit

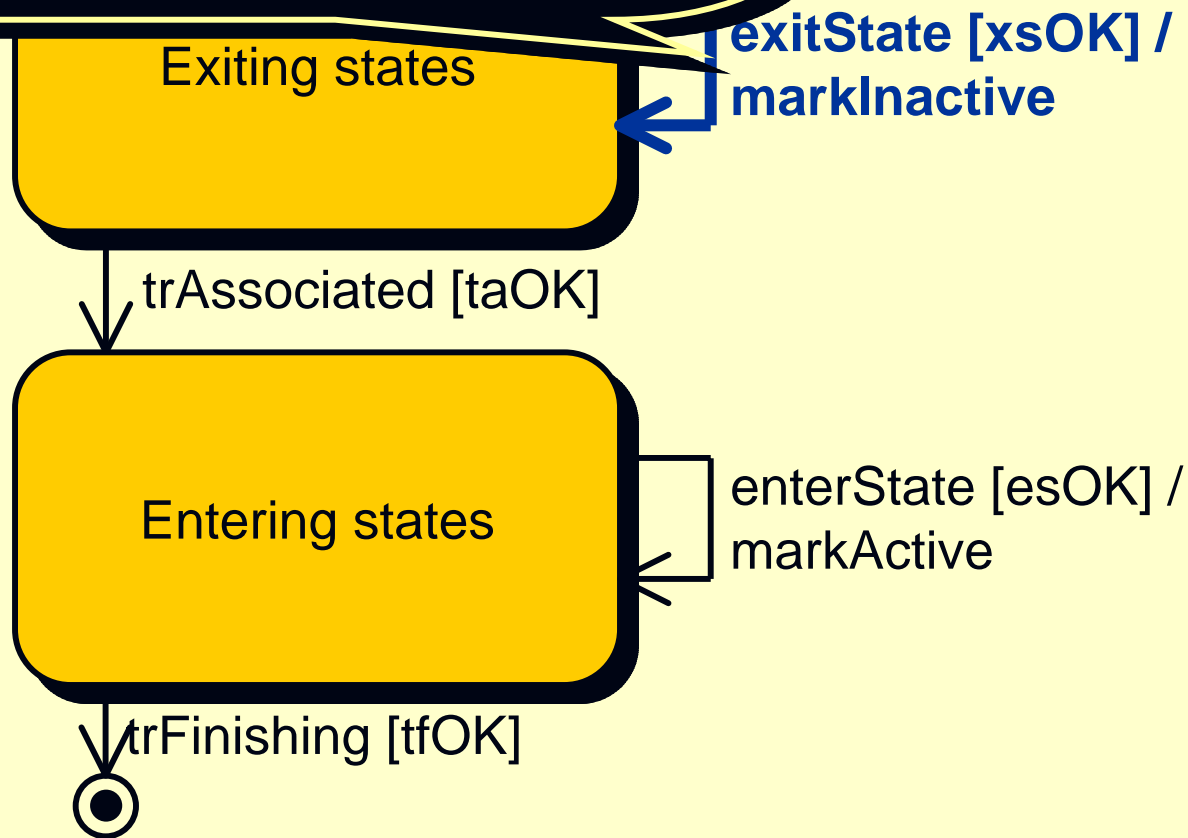
Exiting states

- The observed application is in the first phase of performing a transition: leaving the source state and all active states refining it.

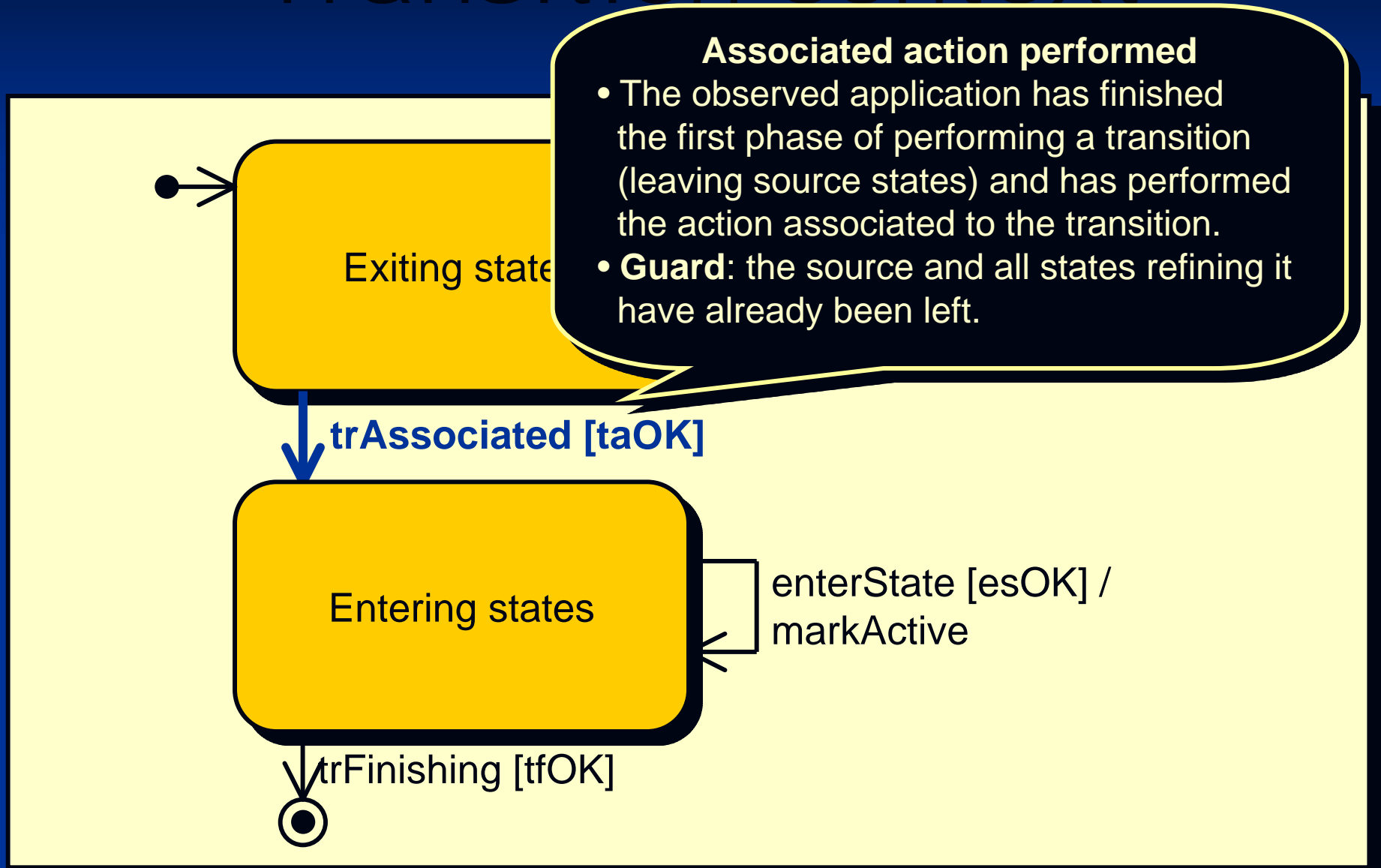


State left

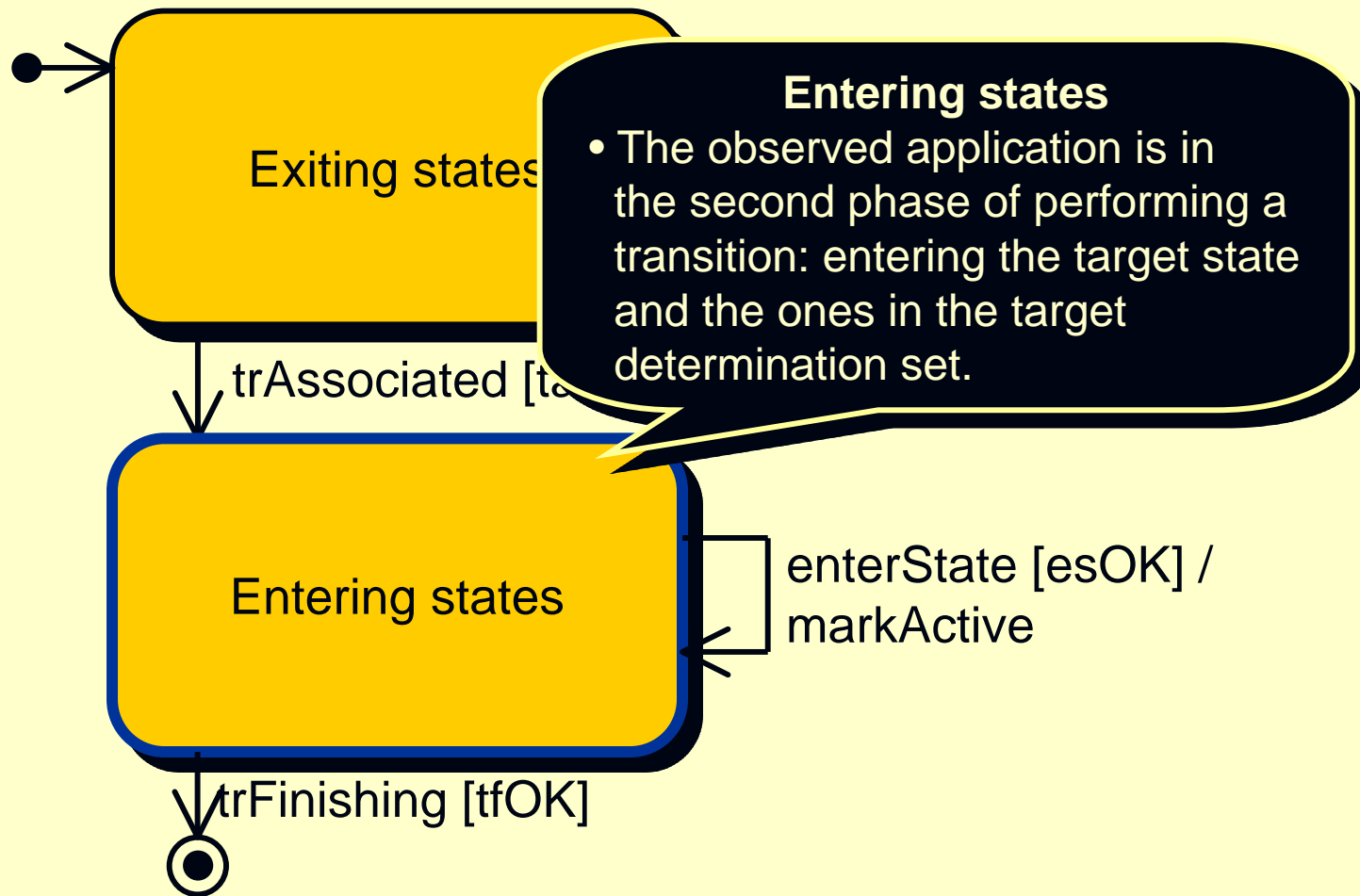
- The observed application has left a state during a transition.
- **Guard:** (i) the state is the source of the transition or a refinement of it, (ii) it is active and (iii) none of its refinements are active.
- **Action:** update the configuration observation.



Transition context



Transition context



Transition context

State entered

- The observed application has entered a state during a transition.
- **Guard:** (i) the state is the target of the transition or member of the target determination set, (ii) it is inactive and (iii) all of its parents are active.
- **Action:** update the configuration observation.

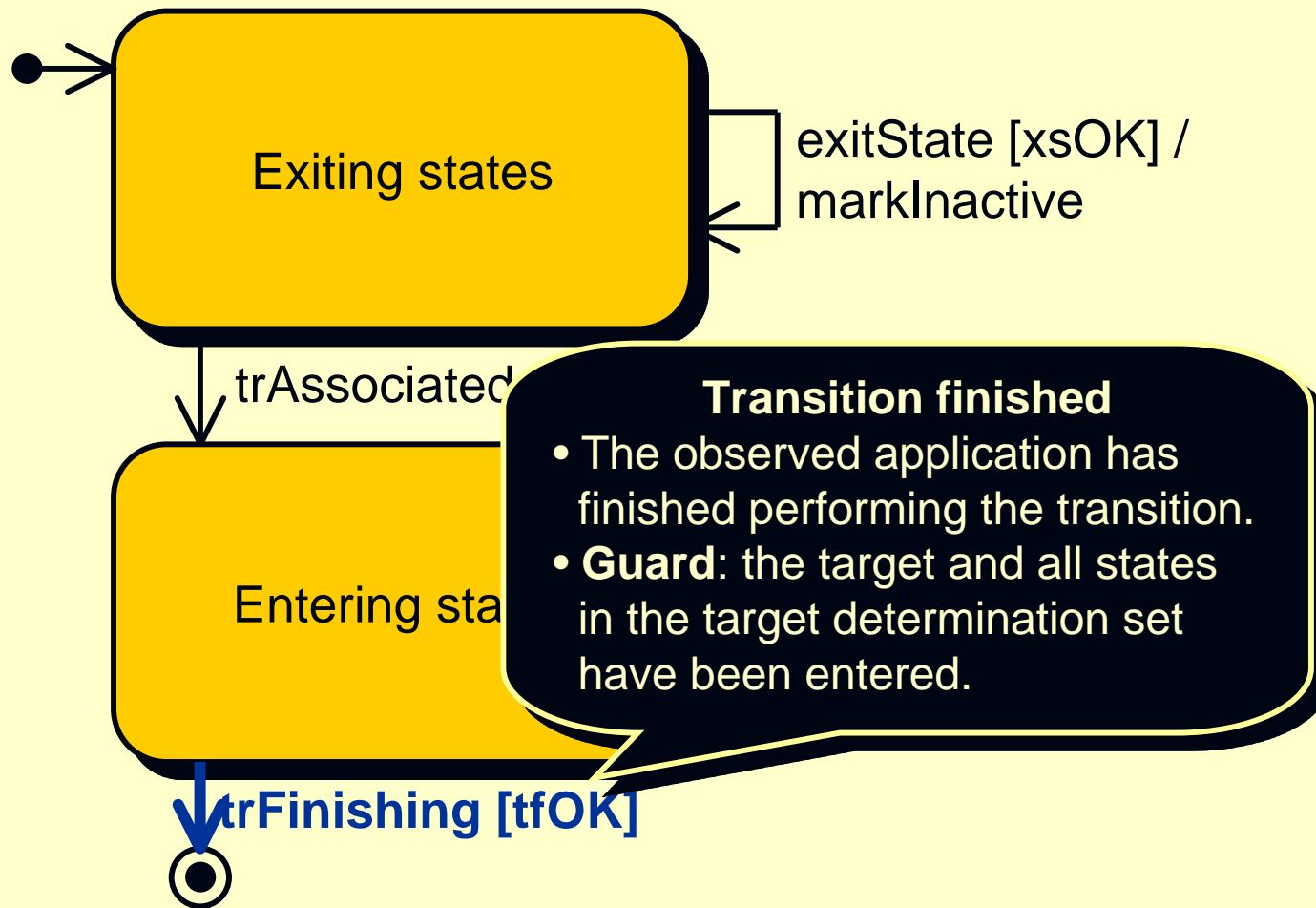
Entering states

enterState [esOK] /
markActive

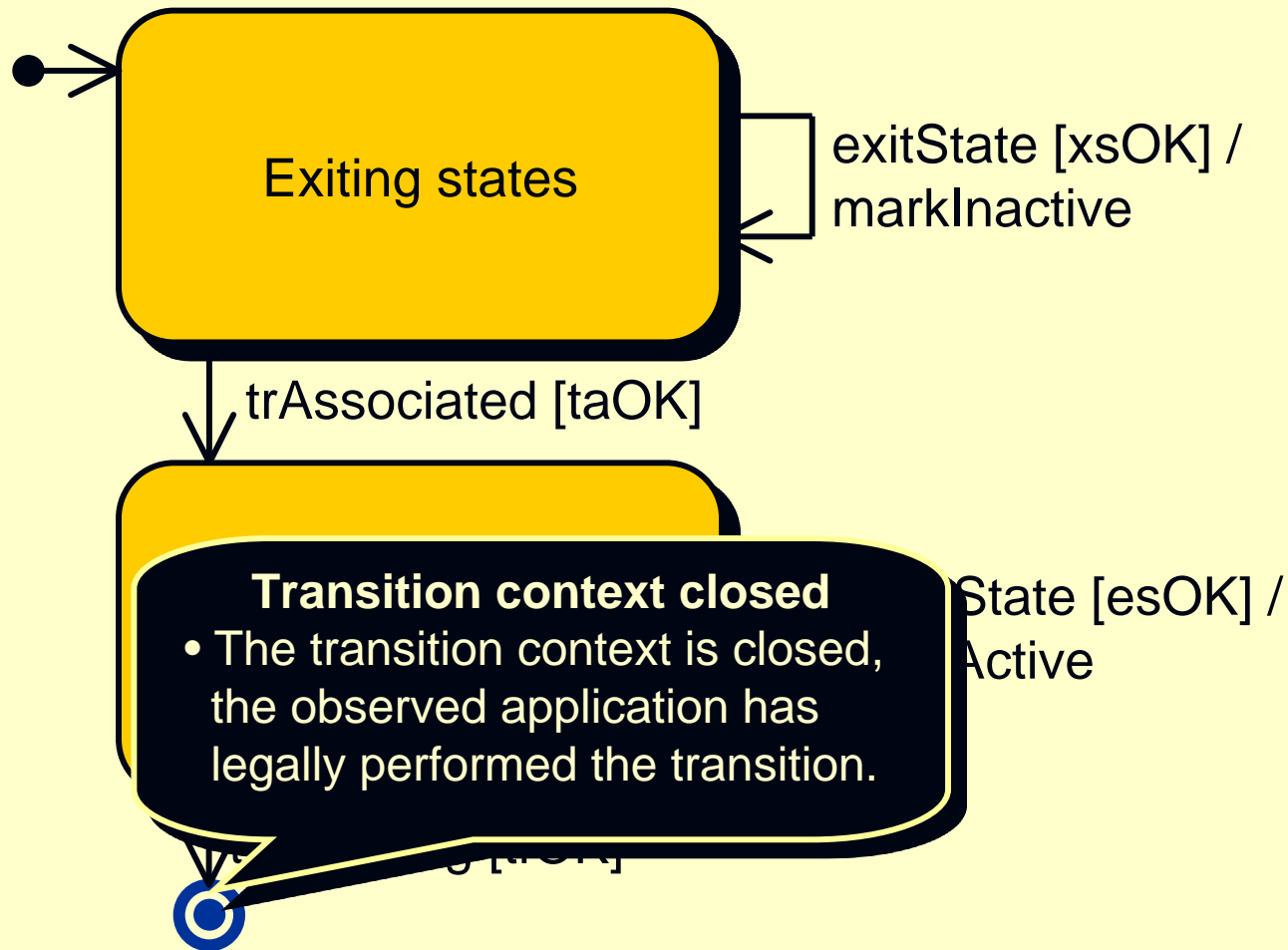
trFinishing [tfOK]



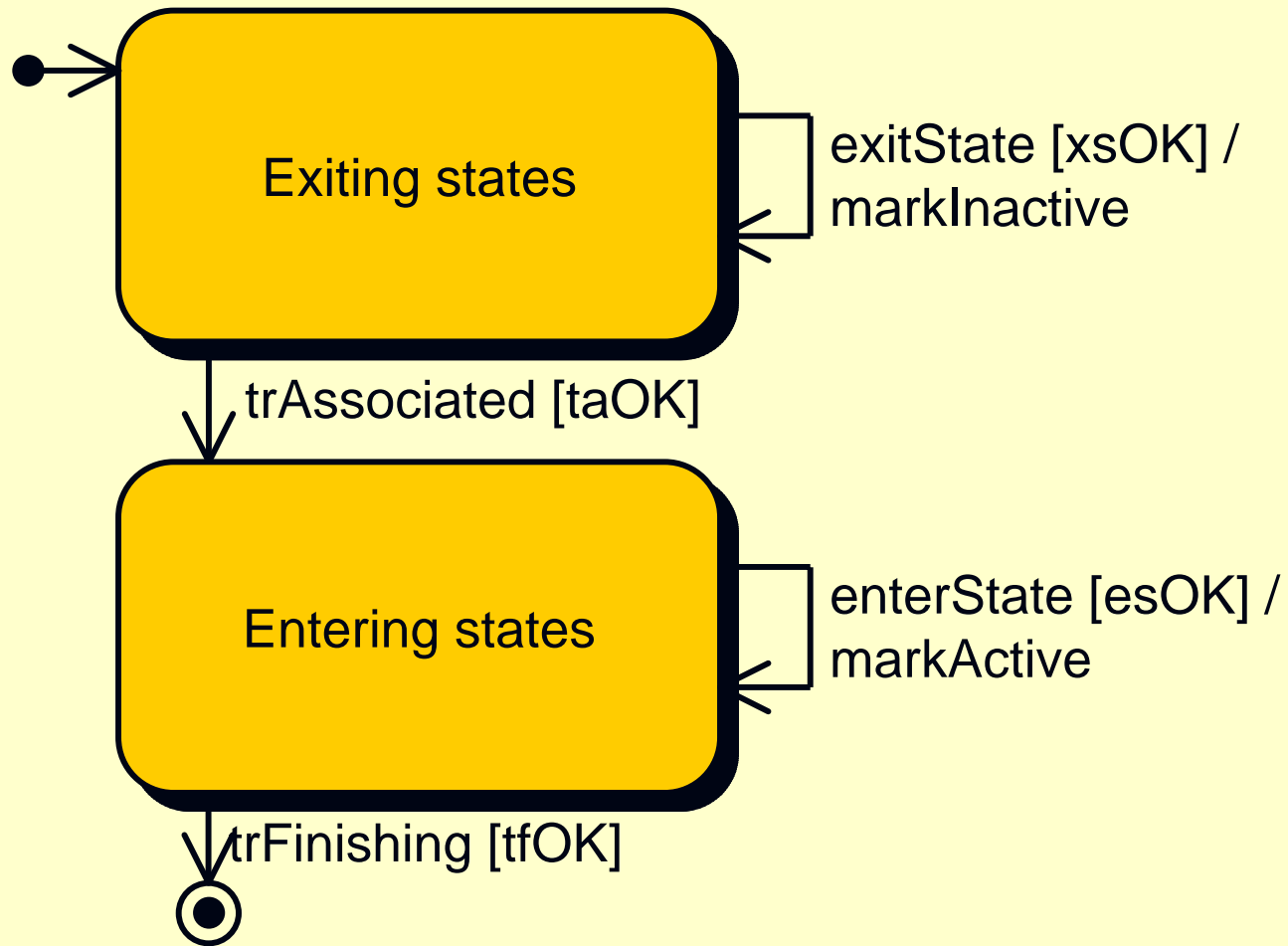
Transition context



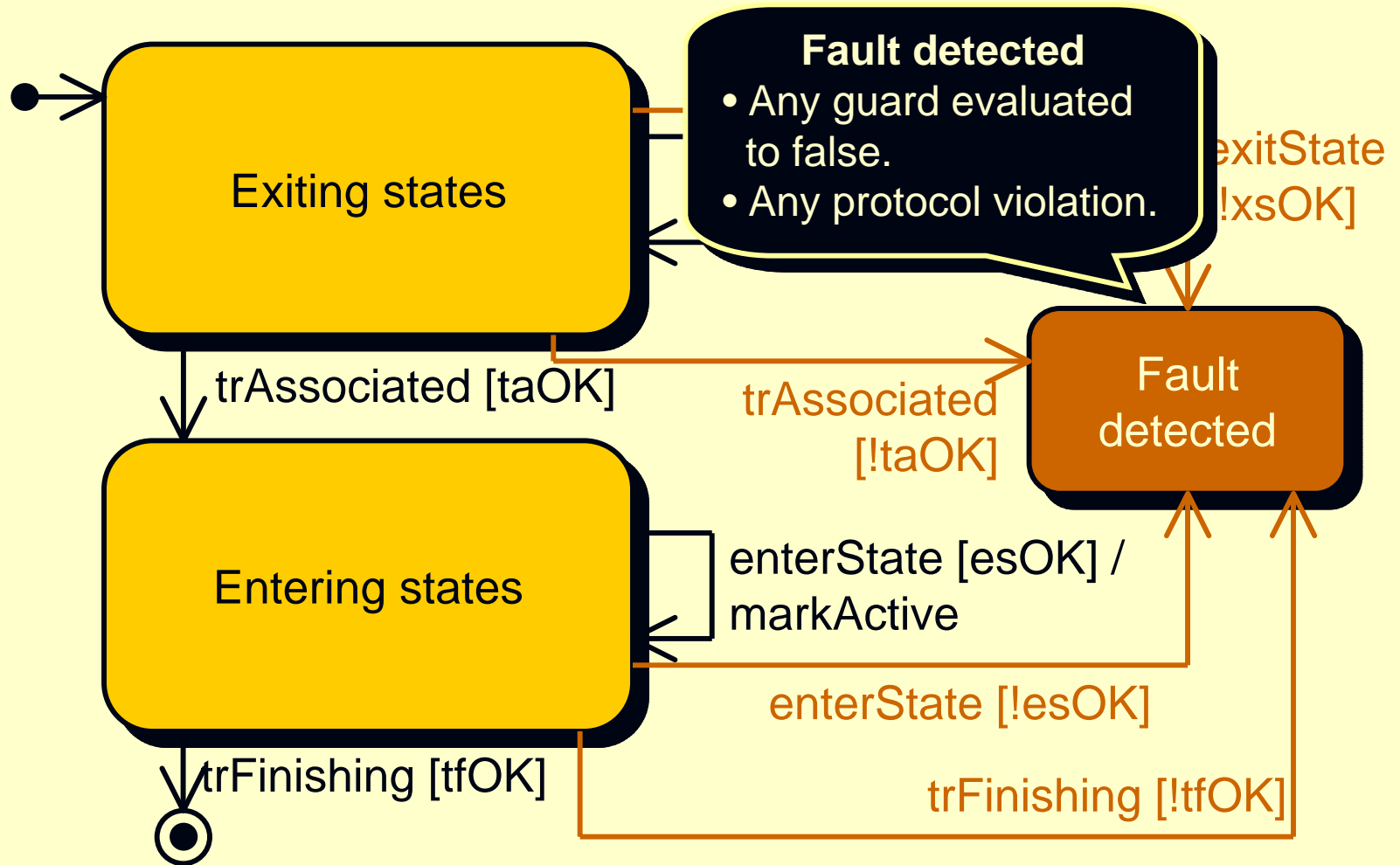
Transition context



Transition context

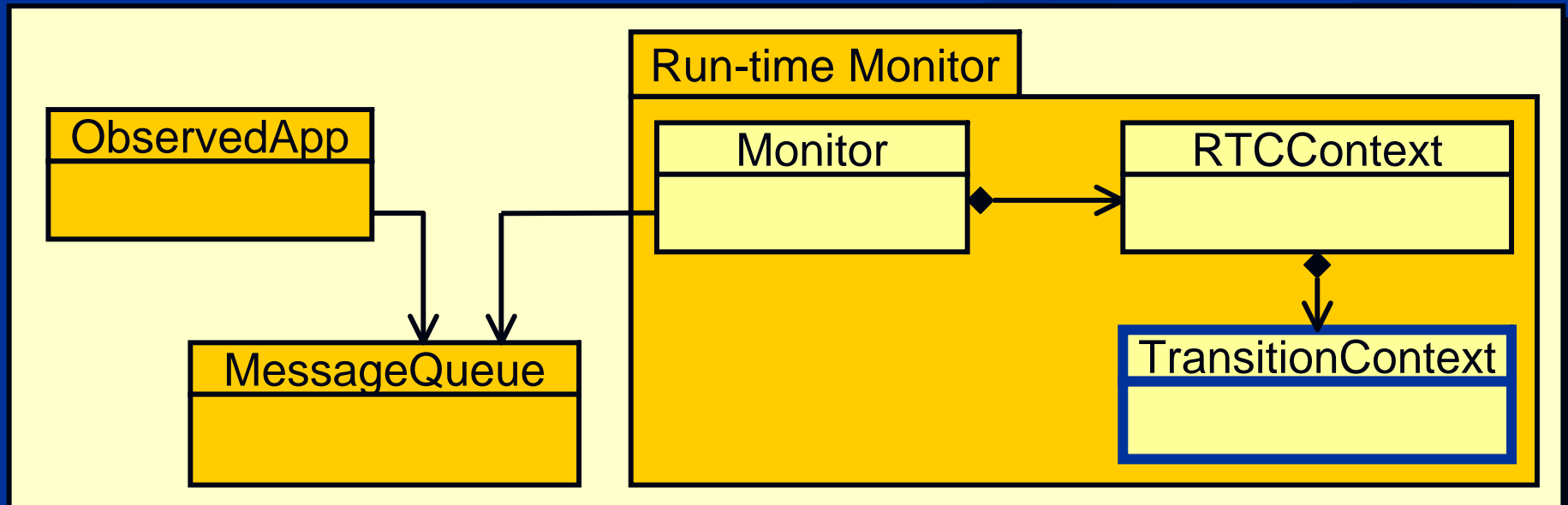


Transition context



Checking the internal behavior

- Structural decomposition:
 - *Run-to-completion and transition contexts*
- Specification of contexts:
 - Protocol state machines (statecharts)



Abstract, high-level control-flow fault detection

Reference information:

- Automatically derived from the behavioral specification
- Capable of expressing state hierarchies, concurrent operation, etc.

Implementation of the monitor:

- Based on the operational semantics of the behavioral model
- Run-time checking of the behavior on the basis of the abstract reference model

Implementation of the instrumentation:

- Providing information to the monitor about the internal behavior
- Configurable, transparent and automatically applied

Abstract, high-level control-flow fault detection

Reference information:

- Automatically derived from the behavioral specification
- Capable of expressing state hierarchies, concurrent operation, etc.

Implementation of the monitor:

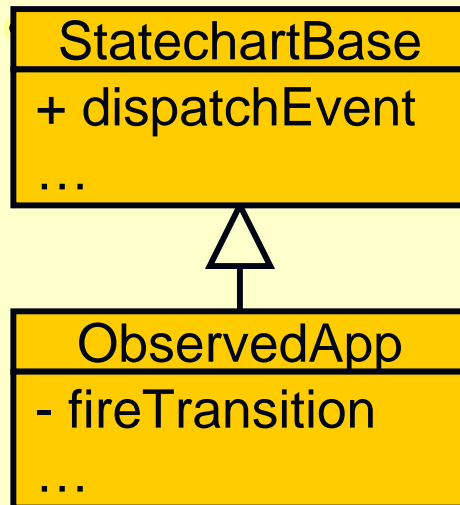
- Based on the operational semantics of the behavioral model
- Run-time checking of the behavior on the basis of the abstract reference model

Implementation of the instrumentation:

- Providing information to the monitor about the internal behavior
- Configurable, transparent and automatically applied

Instrumentation

- Systematic transparent instrumentation:
 - Explicit message transfer to the monitor
 - Modification of the data model and the behavior
 - Case study: Aspect-Oriented Programming

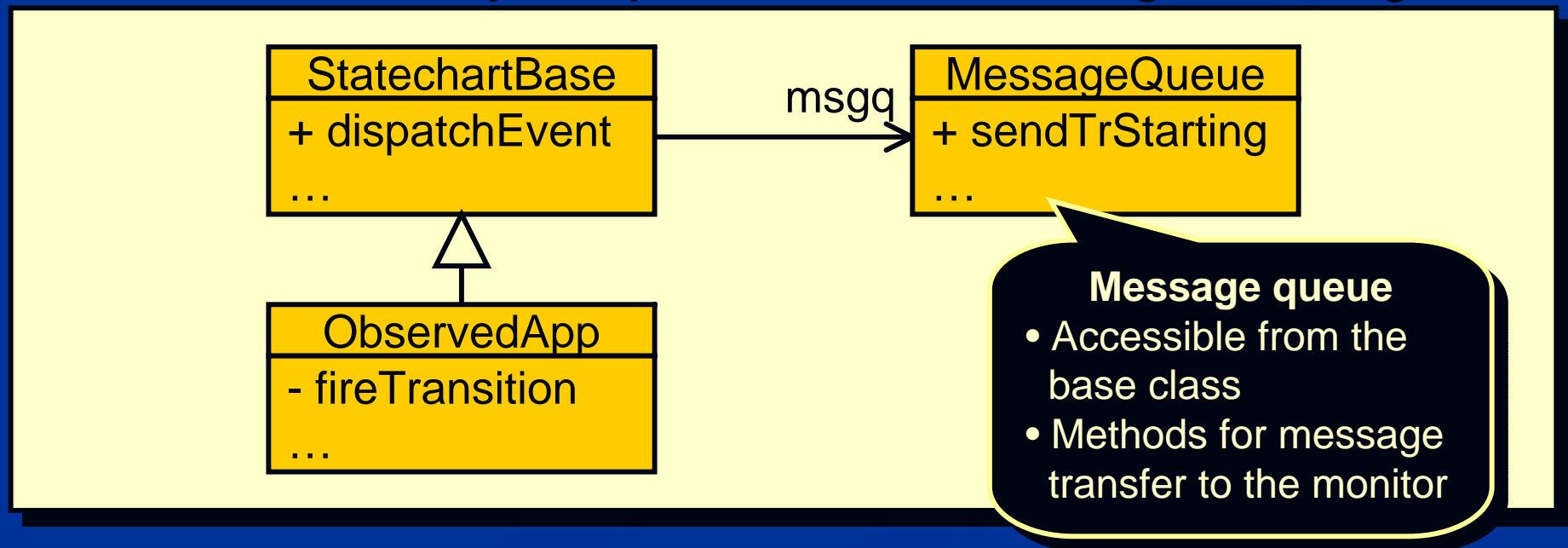


Implementation pattern

- Abstract base class:
Fundamental facilities
- Derived class:
Implements the behavior

Instrumentation

- Systematic transparent instrumentation:
 - Explicit message transfer to the monitor
 - Modification of the data model and the behavior
 - Case study: Aspect-Oriented Programming

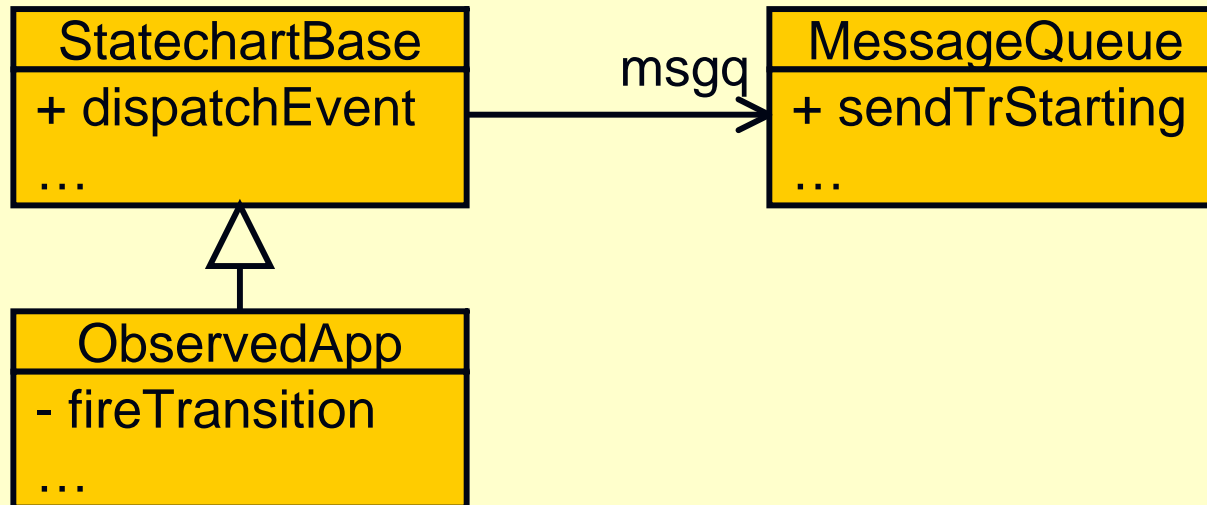


Instrumentation

Adding a member variable (Java AOP)

```
public aspect BehavioralMonitoring {  
    // Add a member variable to the base class  
    protected MessageQueue StatechartBase.msgq;  
    ...  
}
```

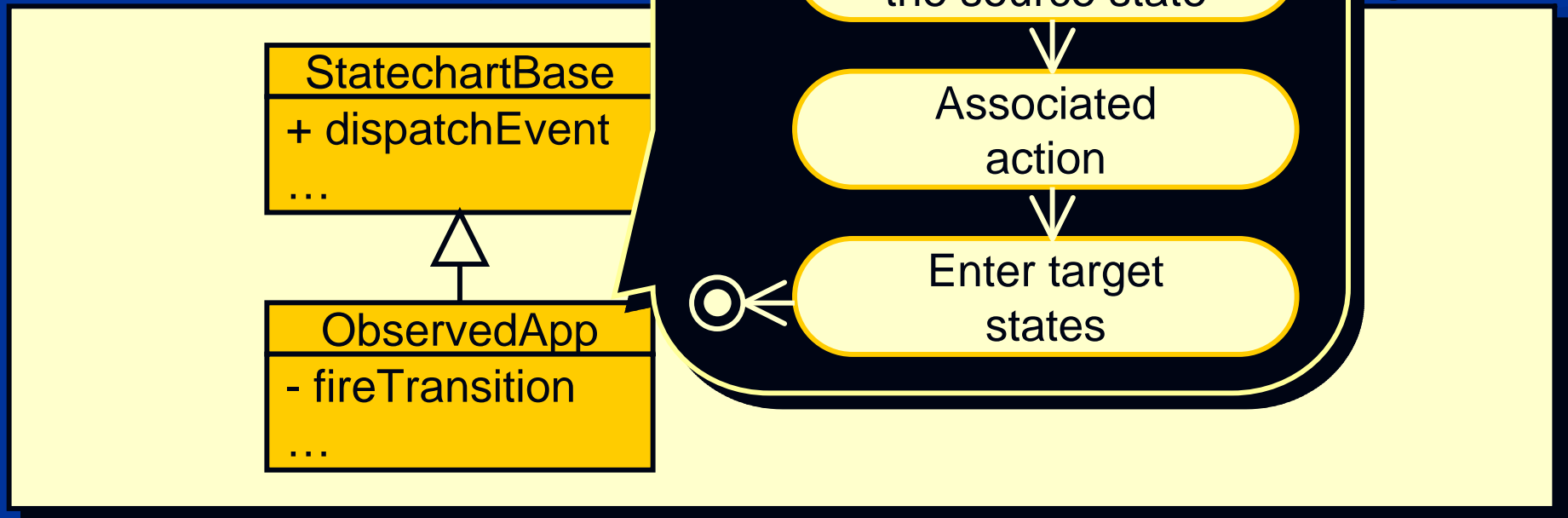
- Systematic
 - Explicit requirements
 - Modification
 - Case study: Aspect-Oriented Programming



Instrumentation

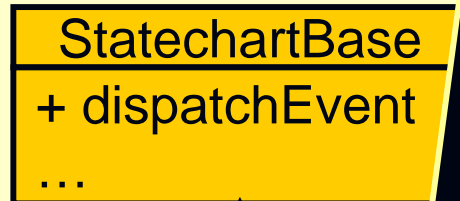
- Systematic transparent instrumentation:

- Explicit message to the instrumentation layer
- Modification of the original behavior
- Case study: ASP

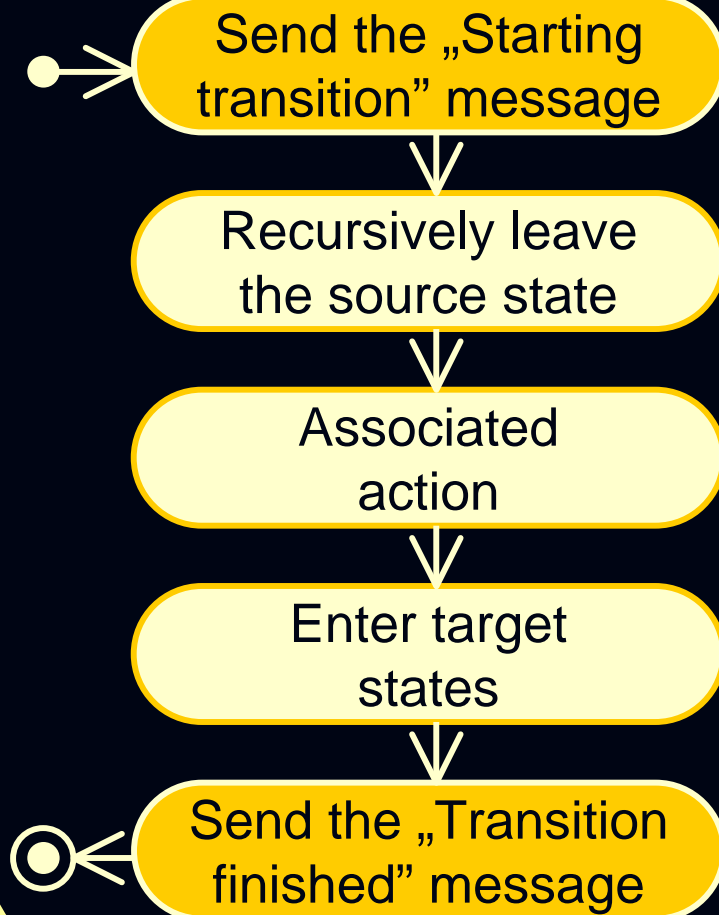


Instru

- Systematic trans
- Explicit message
- Modification of t
- Case study: Asp



Instrumented behavior (Firing a transition)



n:

behavior

g

trStarting [tsOK] /
createTrCtx

Transient

dispatch

Instrumented behavior
(Firing a transition)

Send the „Starting
transition” message

Recursively leave
the source state

Associated
action

Enter target
states

Send the „Transition
finished” message

Case Study: Asp

StatechartBase
+ dispatchEvent
...

ObservedApp
- fireTransition
...

n:

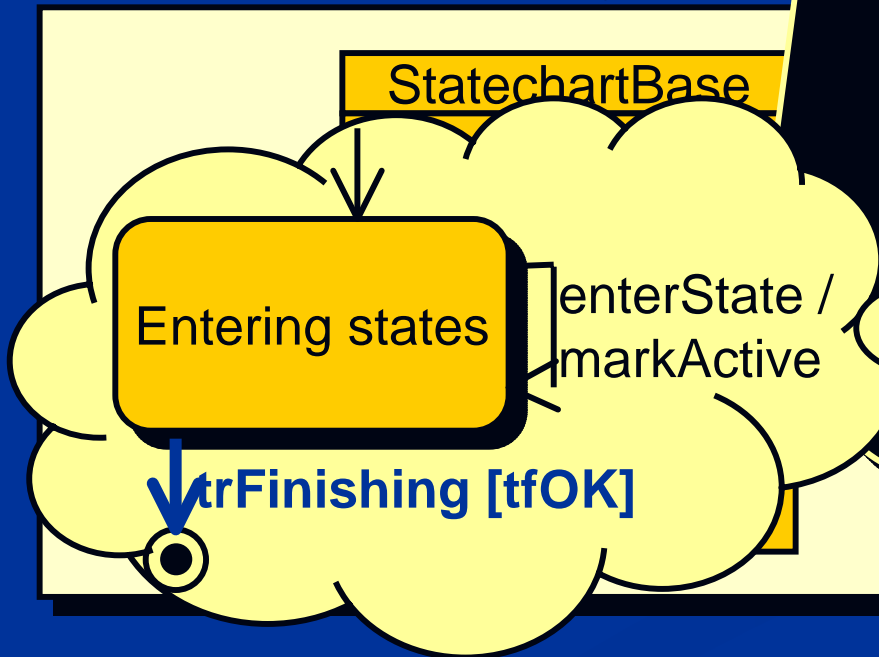
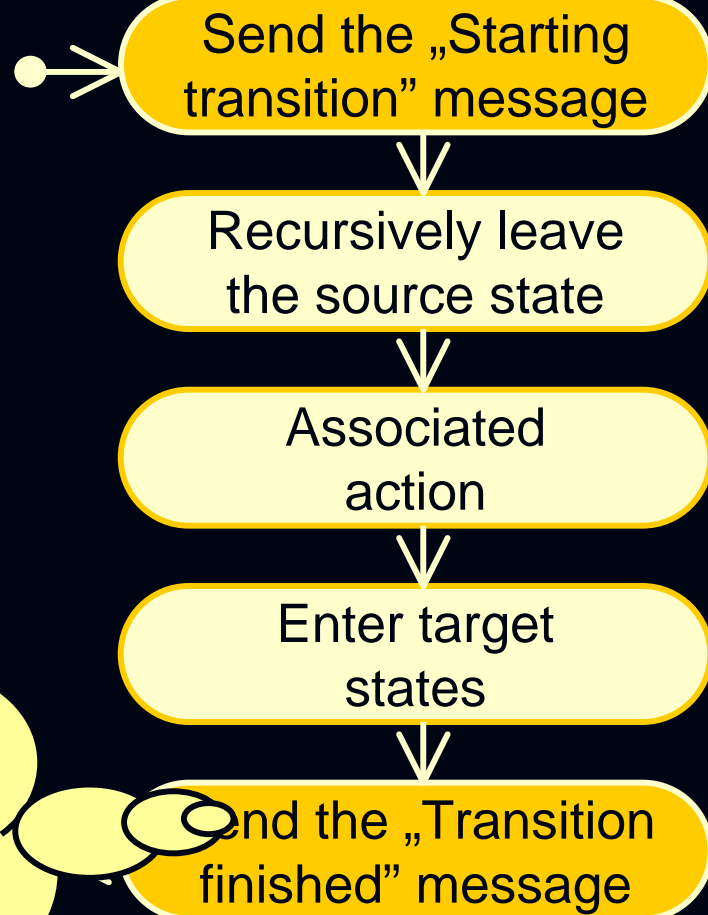
behavior

g

Instru

- Systematic trans
 - Explicit message
 - Modification of t
 - Case study: Asp

Instrumented behavior (Firing a transition)



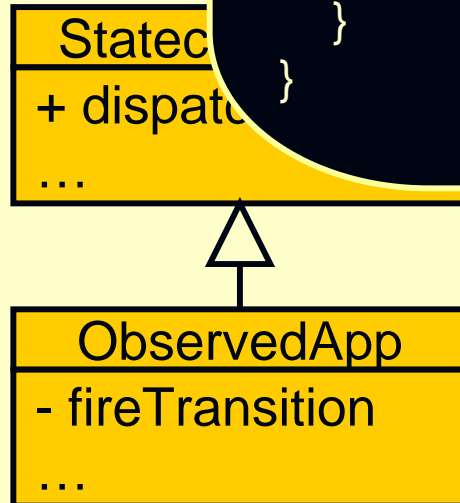
h:
behavior
g

In

- Systemat
 - Explicit r
 - Modifica
 - Case stu

Add code around function call (Java AOP)

```
public aspect BehavioralMonitoring {  
    // Define pattern matching calls to fireTransition  
    pointcut firingTransitionPattern  
        call (StatechartBase+.fireTransition(Transition t));  
  
    // Define instrumentation to be applied  
    around(): firingTransitionPattern() {  
        msgq.sendTrStarting();  
        proceed();  
        msgq.sendTrFinishing();  
    }  
}
```



Abstract, high-level control-flow fault detection

Reference information:

- Automatically derived from the behavioral specification
- Capable of expressing state hierarchies, concurrent operation, etc.

Implementation of the monitor:

- Based on the operational semantics of the behavioral model
- Run-time checking of the behavior on the basis of the abstract reference model

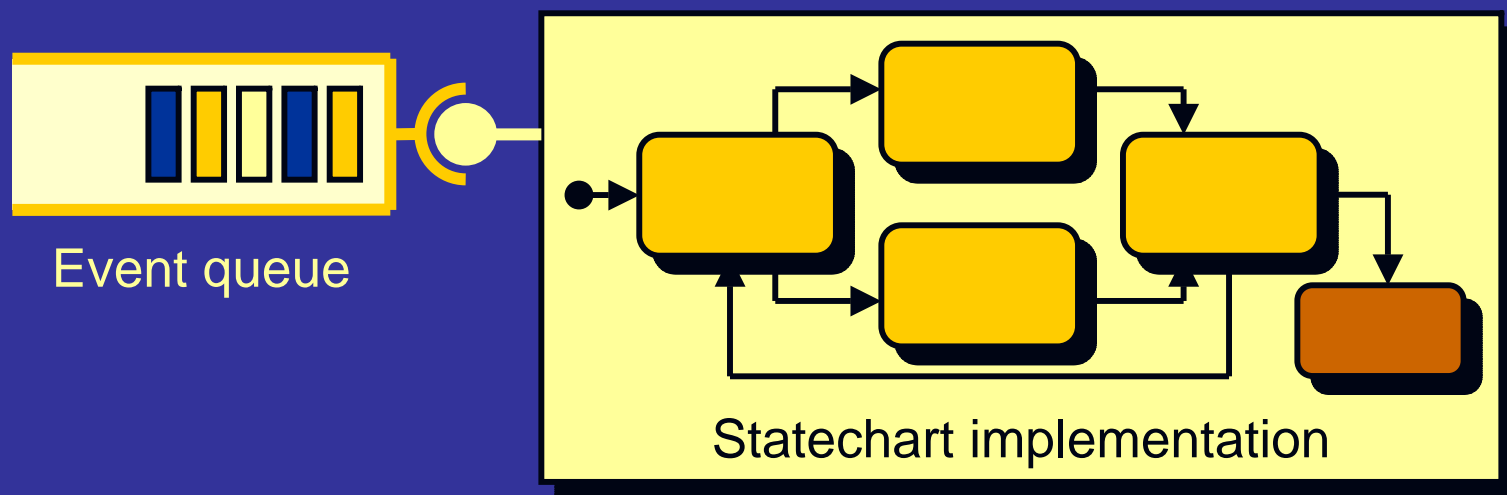
Implementation of the instrumentation:

- Providing information to the monitor about the internal behavior
- Configurable, transparent and automatically applied

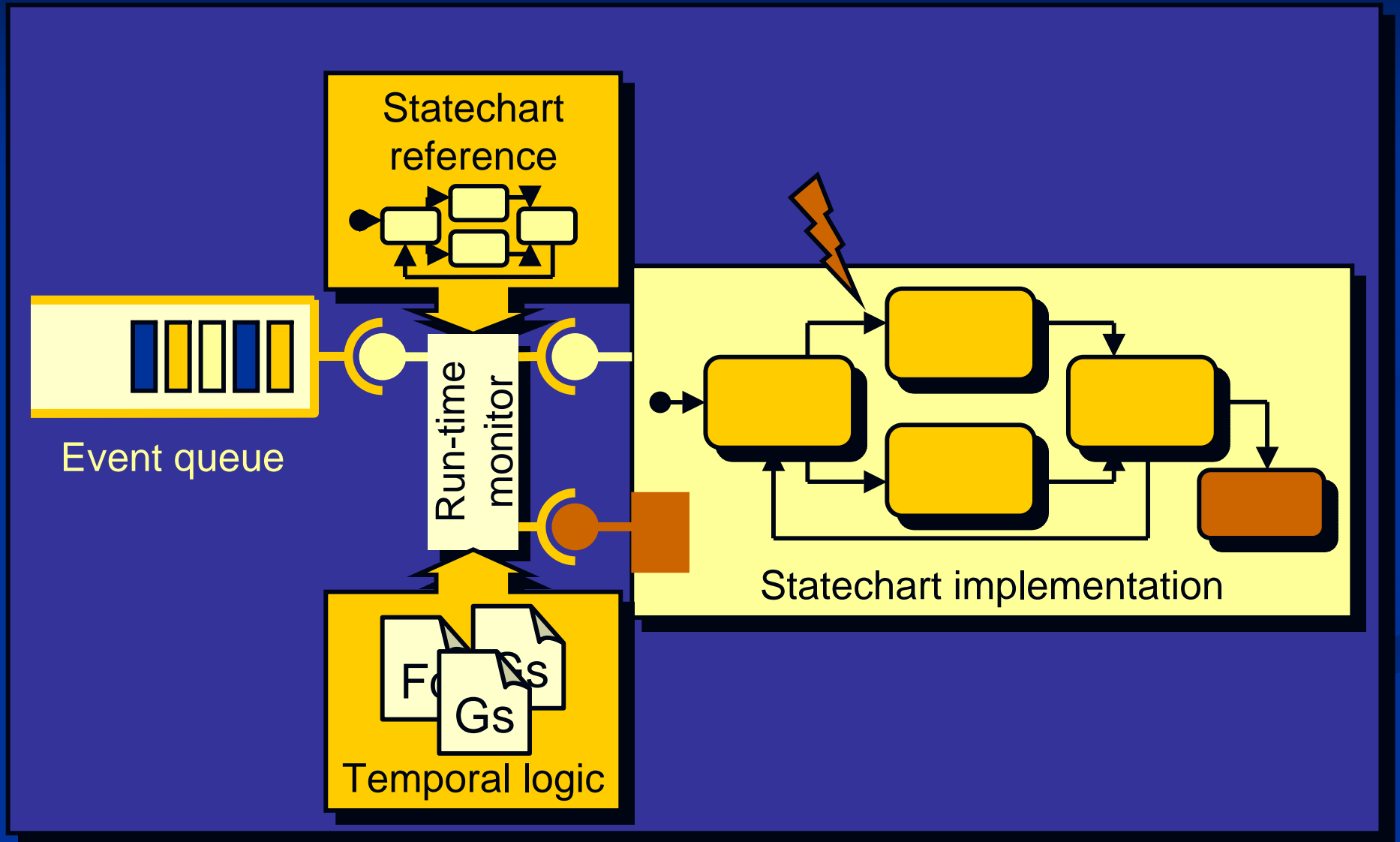
Summary

- Monitoring of the dynamic behavior
 - Verification against abstract specification
 - Pattern-based instrumentation scheme
- Prototype implementation
 - Benchmark experiment: bit-inversion faults in the statechart implementation (C++ version)
 - HW: 40%, monitor: 21.5%, SW: 18.5%
 - Instrumentation case study (Java, AspectJ):
 - Run-time overhead: 10.9%

Future work



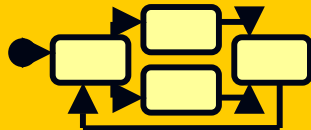
Future work



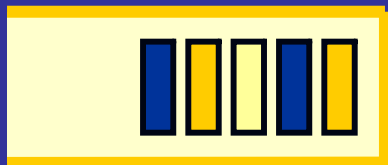
Future work

Run-time verification of behavior

Statechart reference

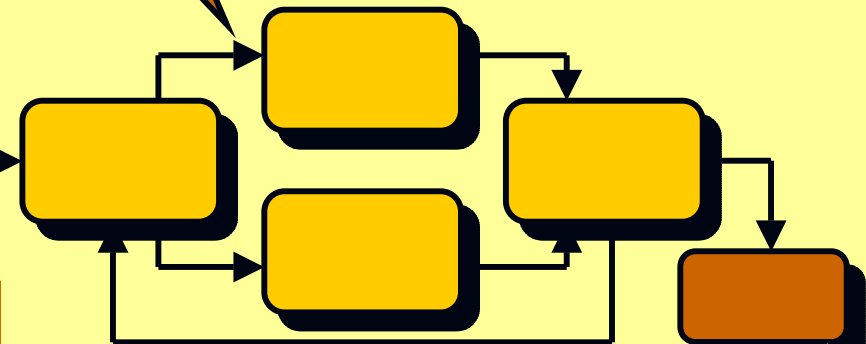


Assessment by fault injection
EUROMICRO 04



Event queue

Run-time monitor



Statechart implementation

Verification against SC-LTL formulae



Temporal logic

Instrumentation by AOP

Exception handling in statecharts

Futi

Run-time fault detection of statechart implementations

- Formal reference
- Temporal logic fitted to statecharts
- Exception handling
- No manual modification on the implementation (AOP)
- Original event dispatcher interface

