High-level Supervision of Program Execution Based on Formal Specification

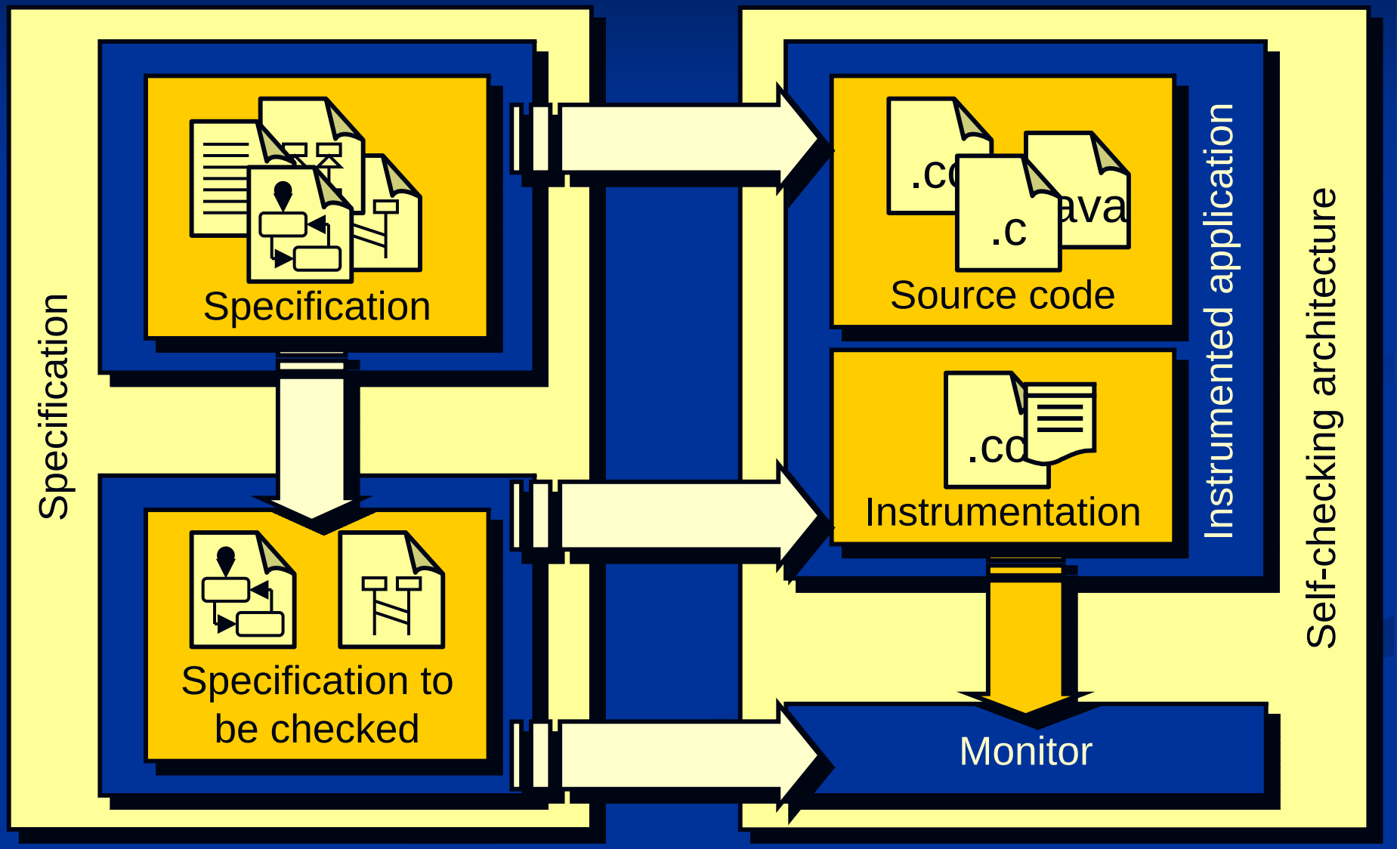
- **Gergely PINTÉR**
- István MAJZIK

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

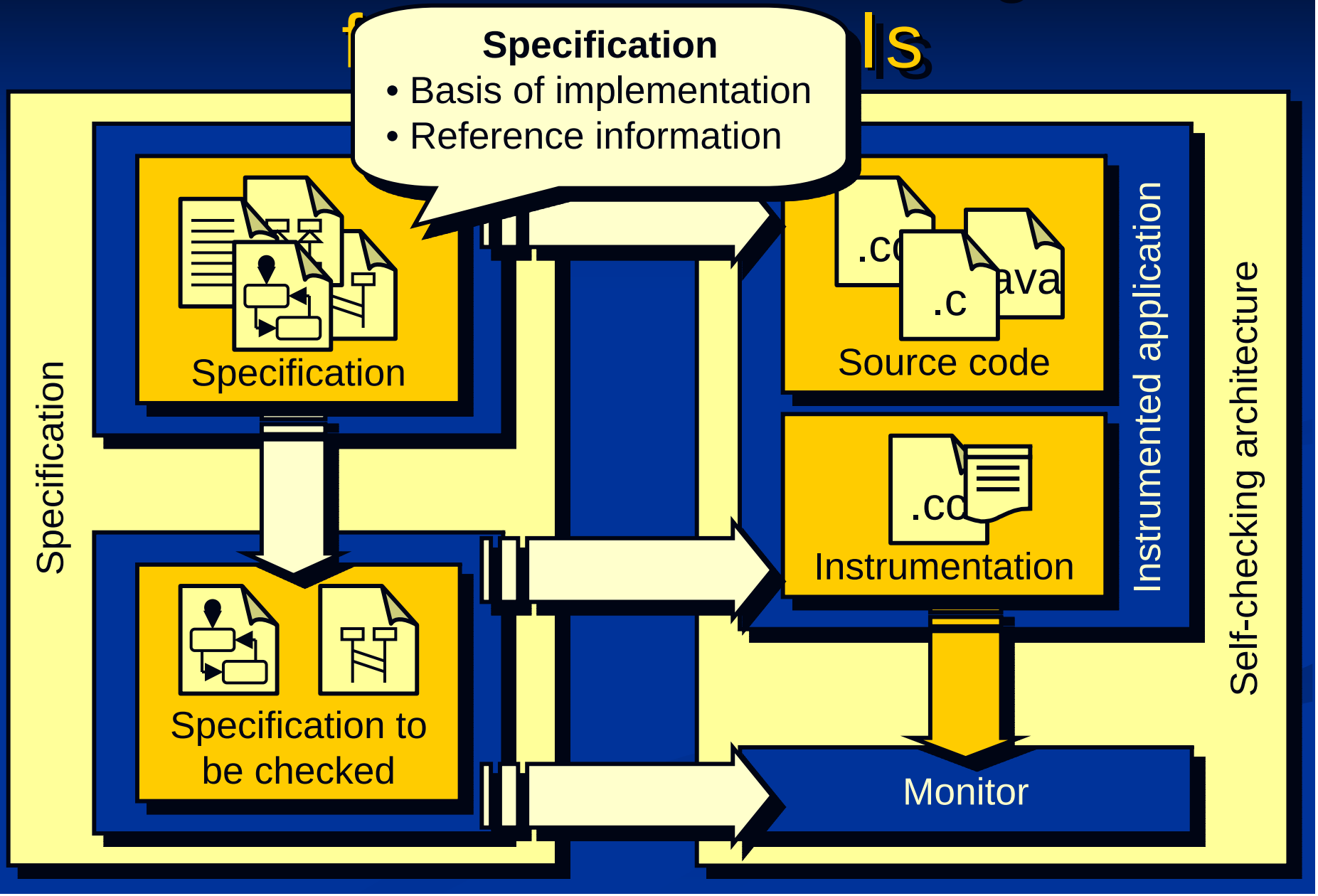
Research goals

- 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)

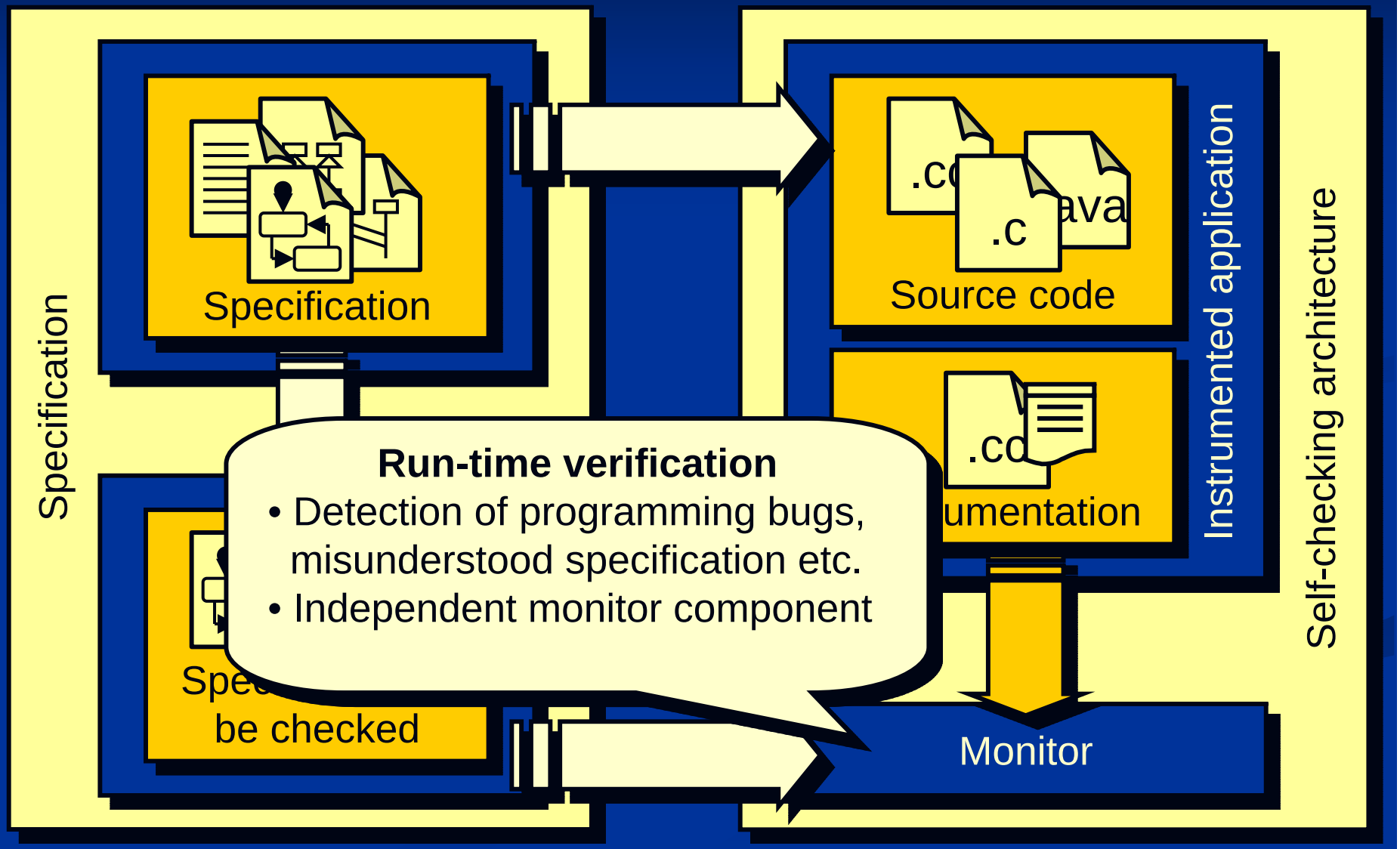
Run-time verification against formal models



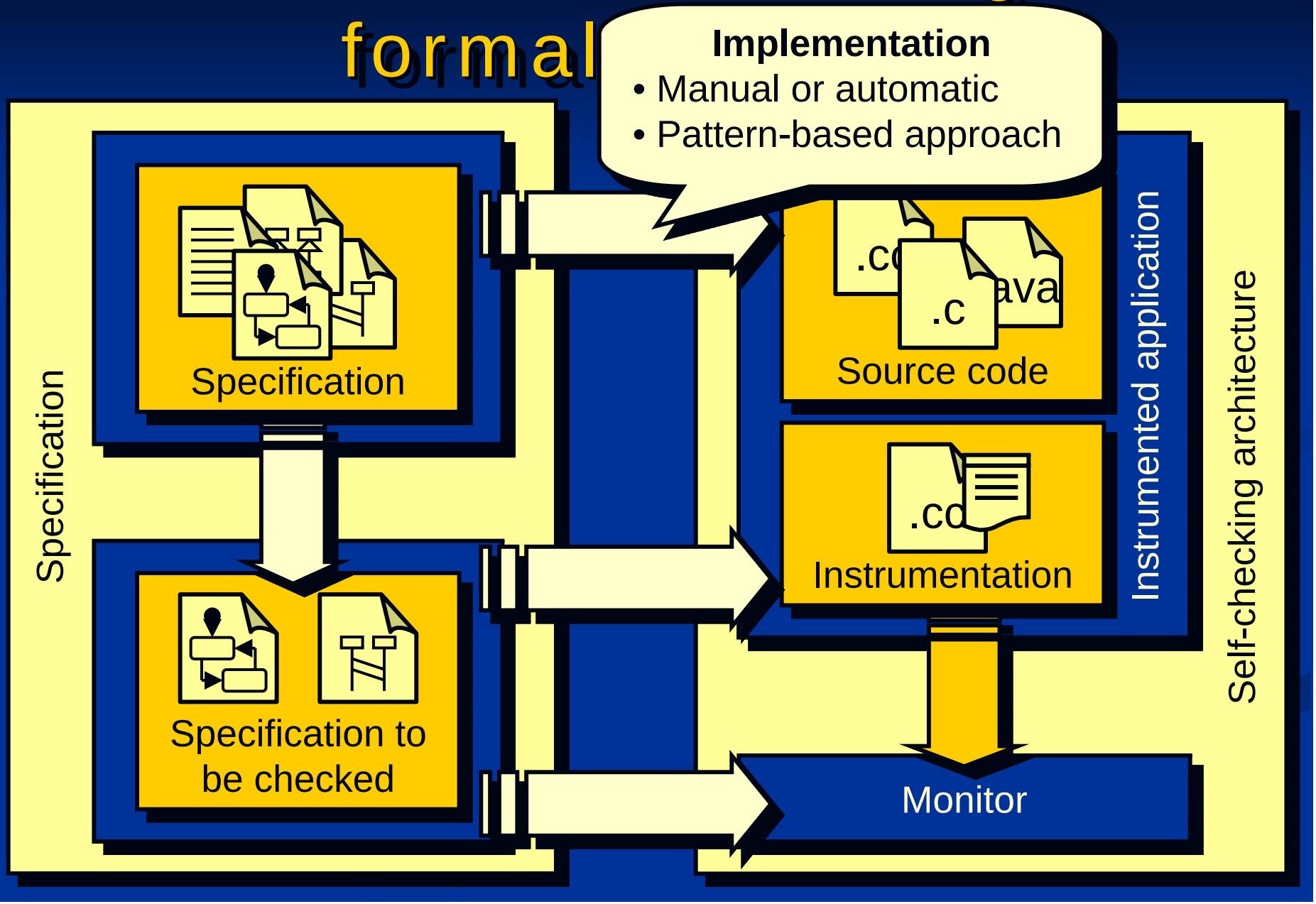
Run-time verification against f ls



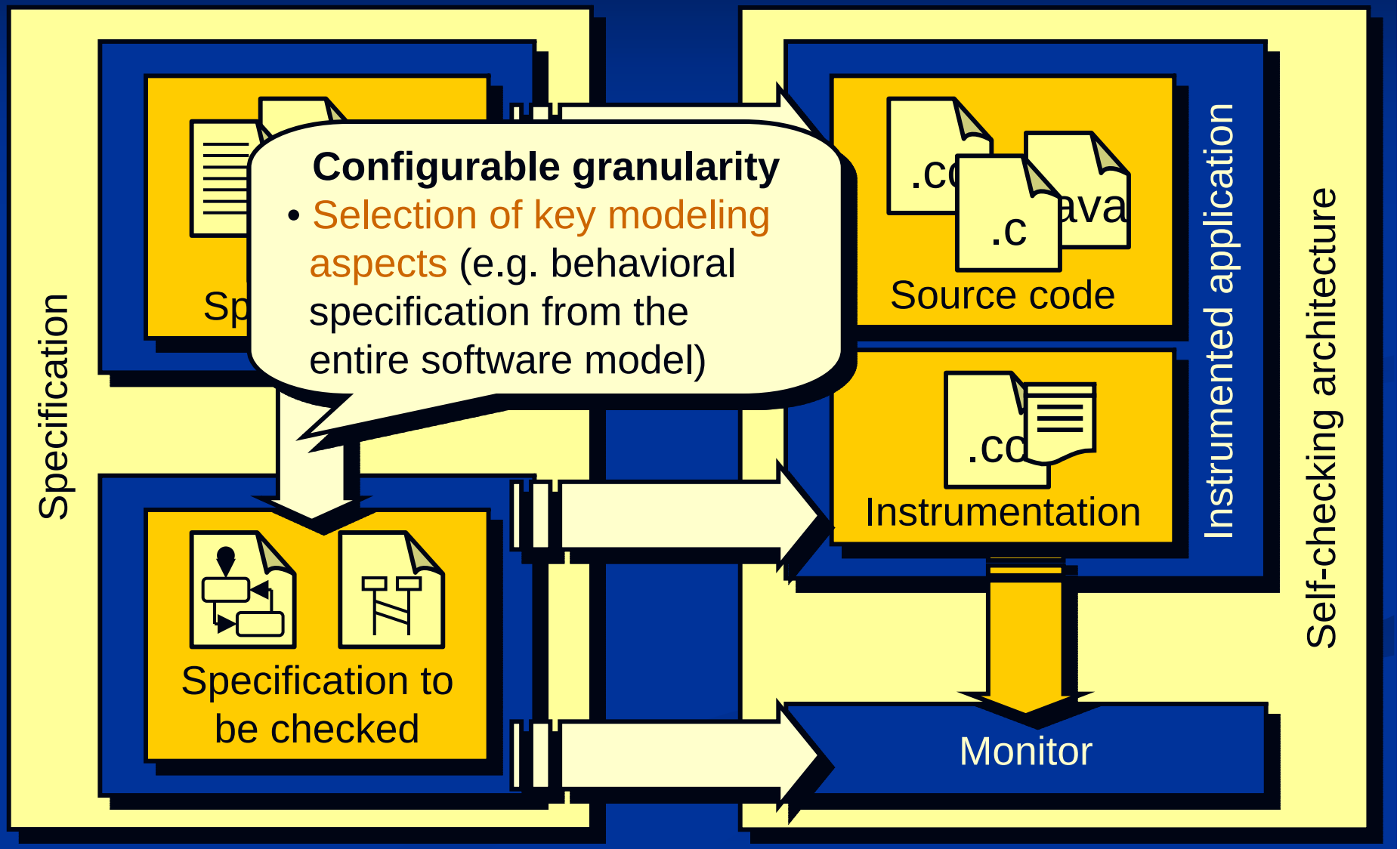
Run-time verification against formal models



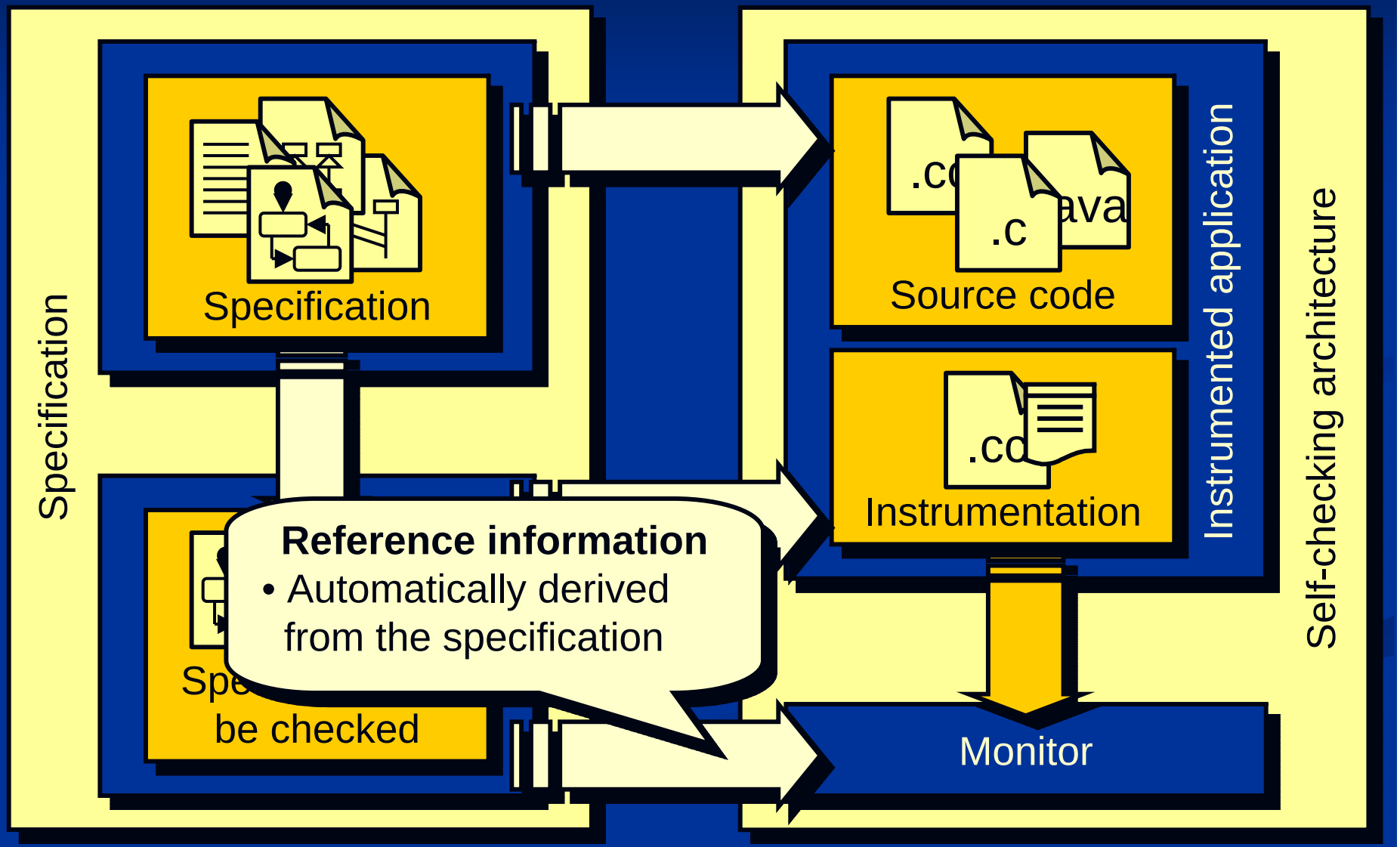
Run-time verification against formal



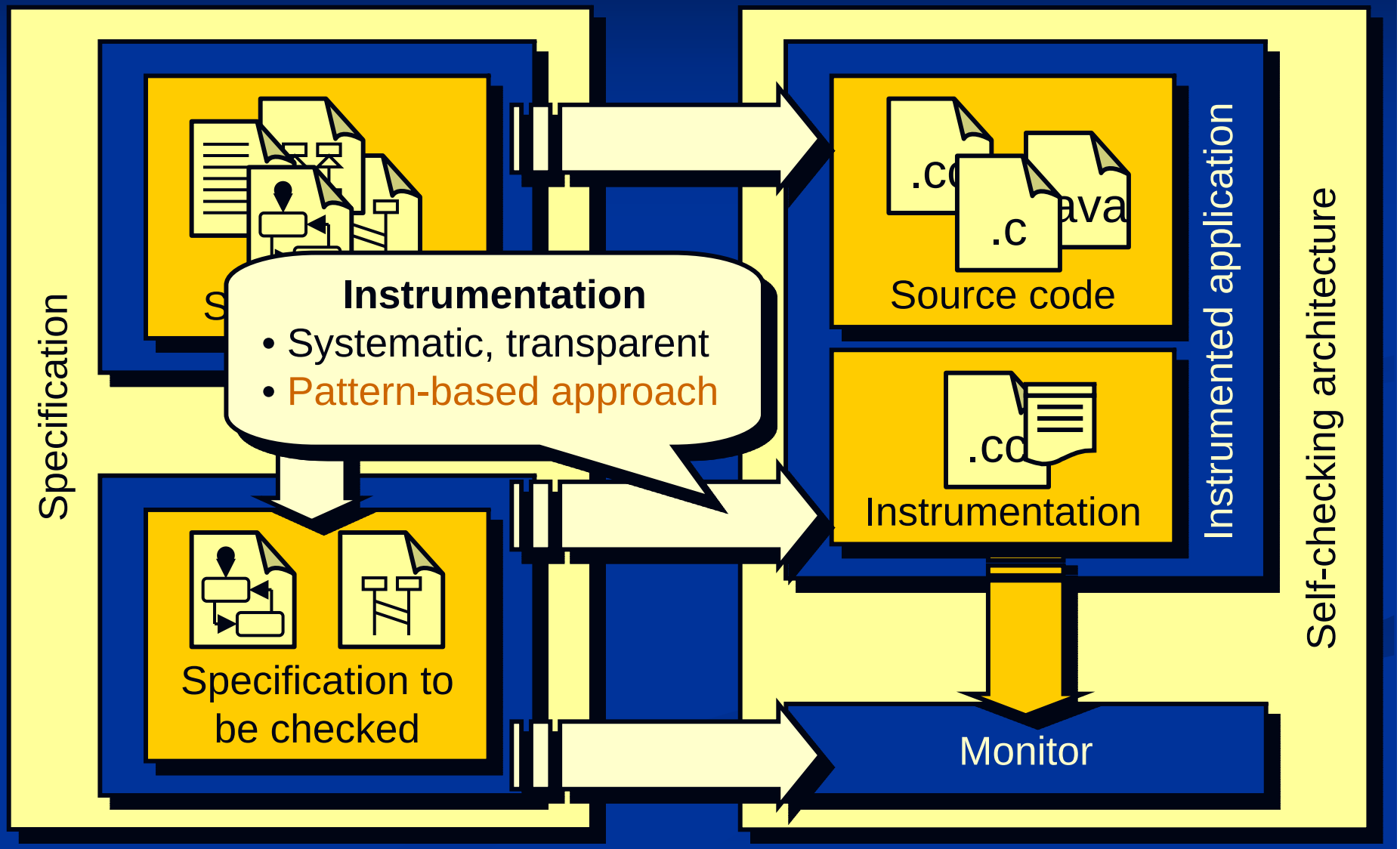
Run-time verification against formal models



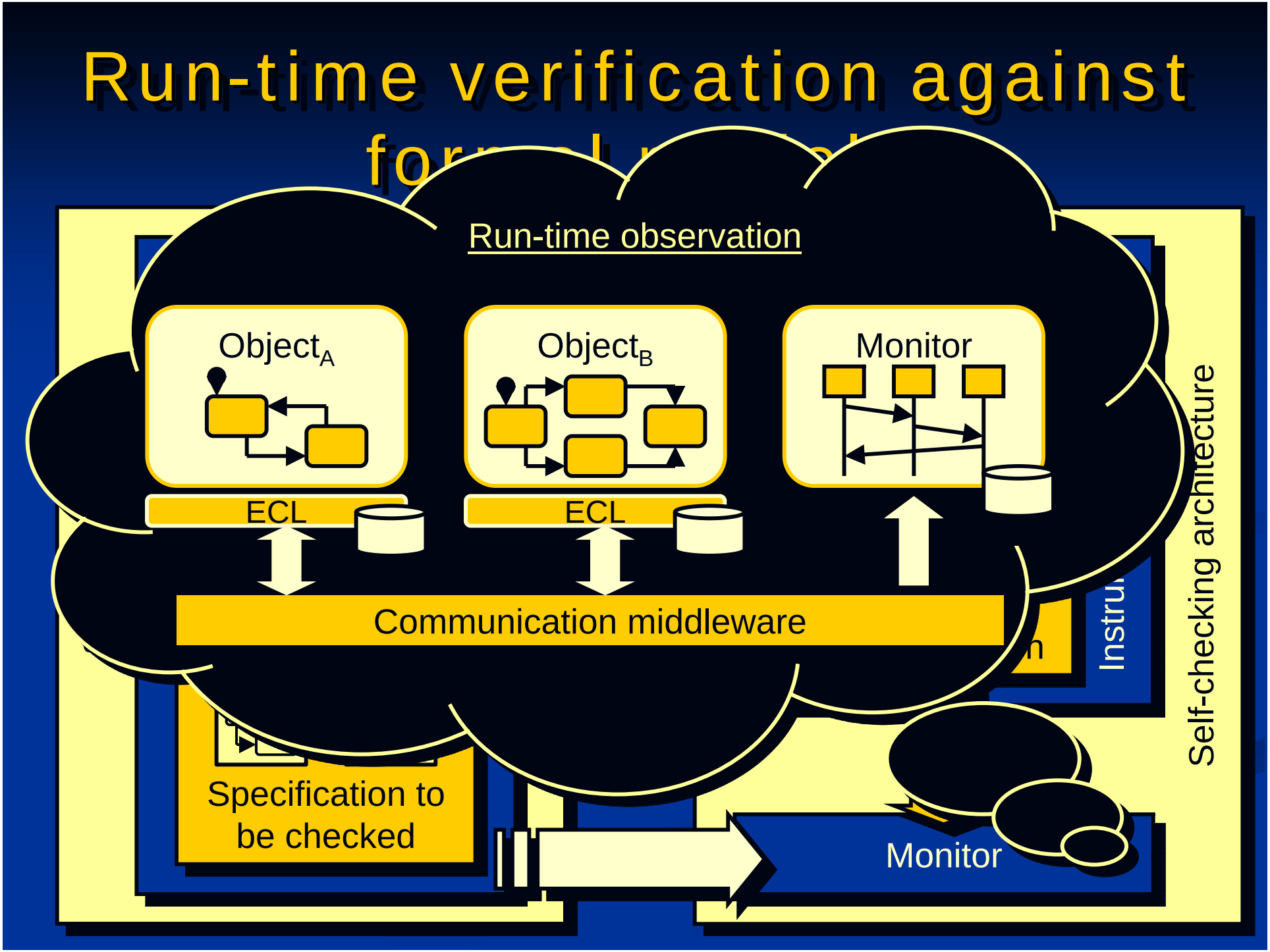
Run-time verification against formal models



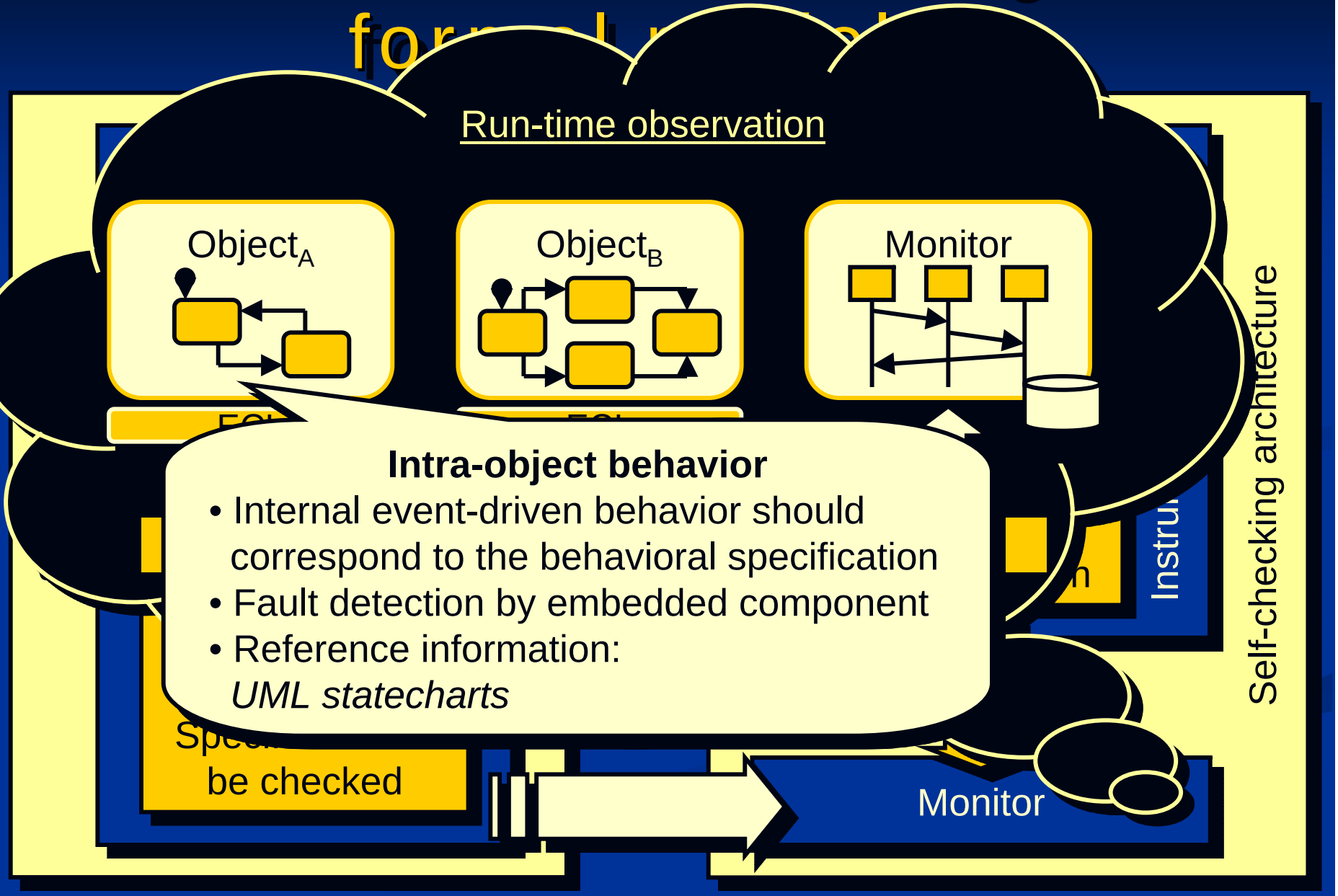
Run-time verification against formal models



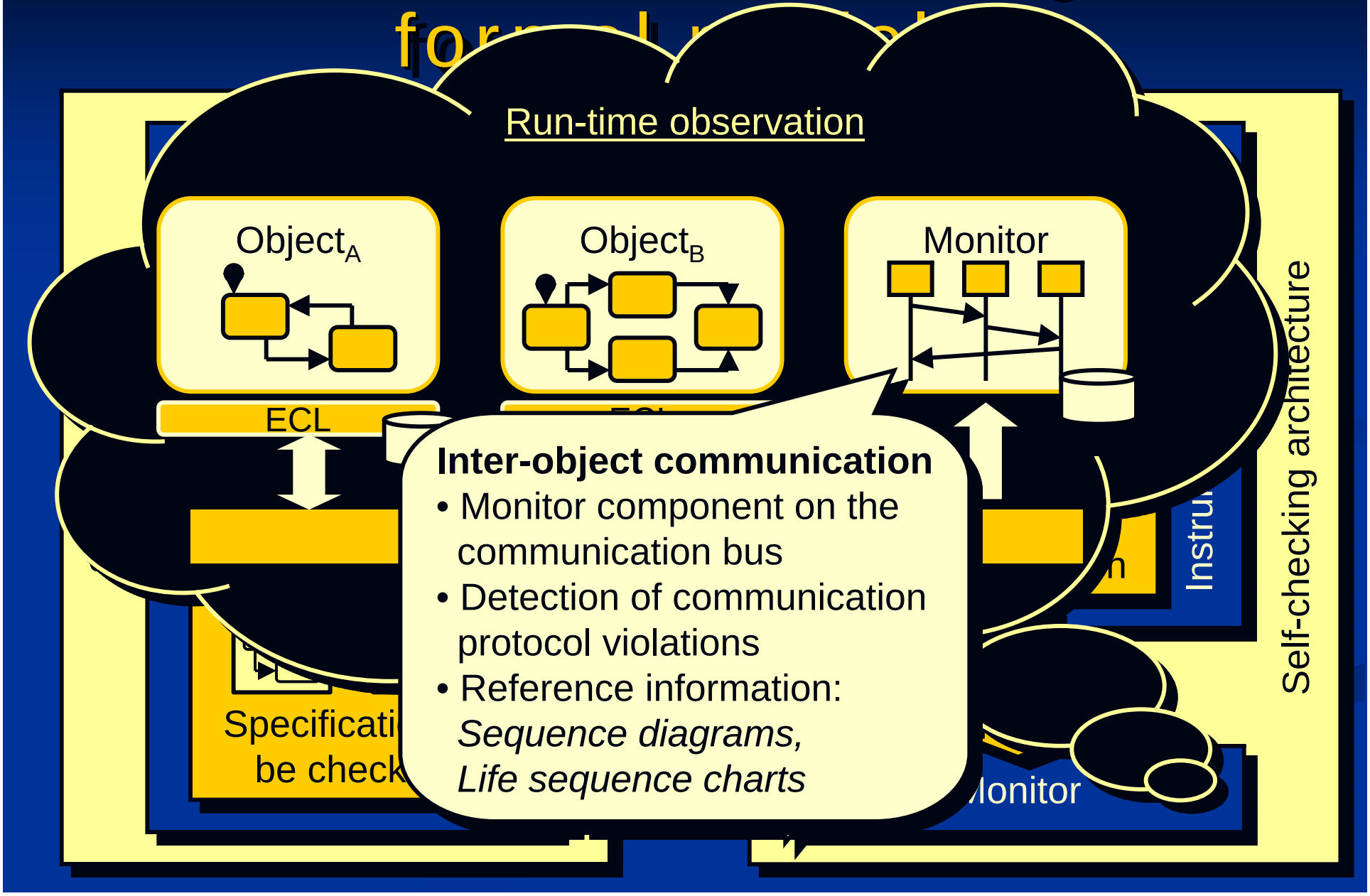
Run-time verification against formal



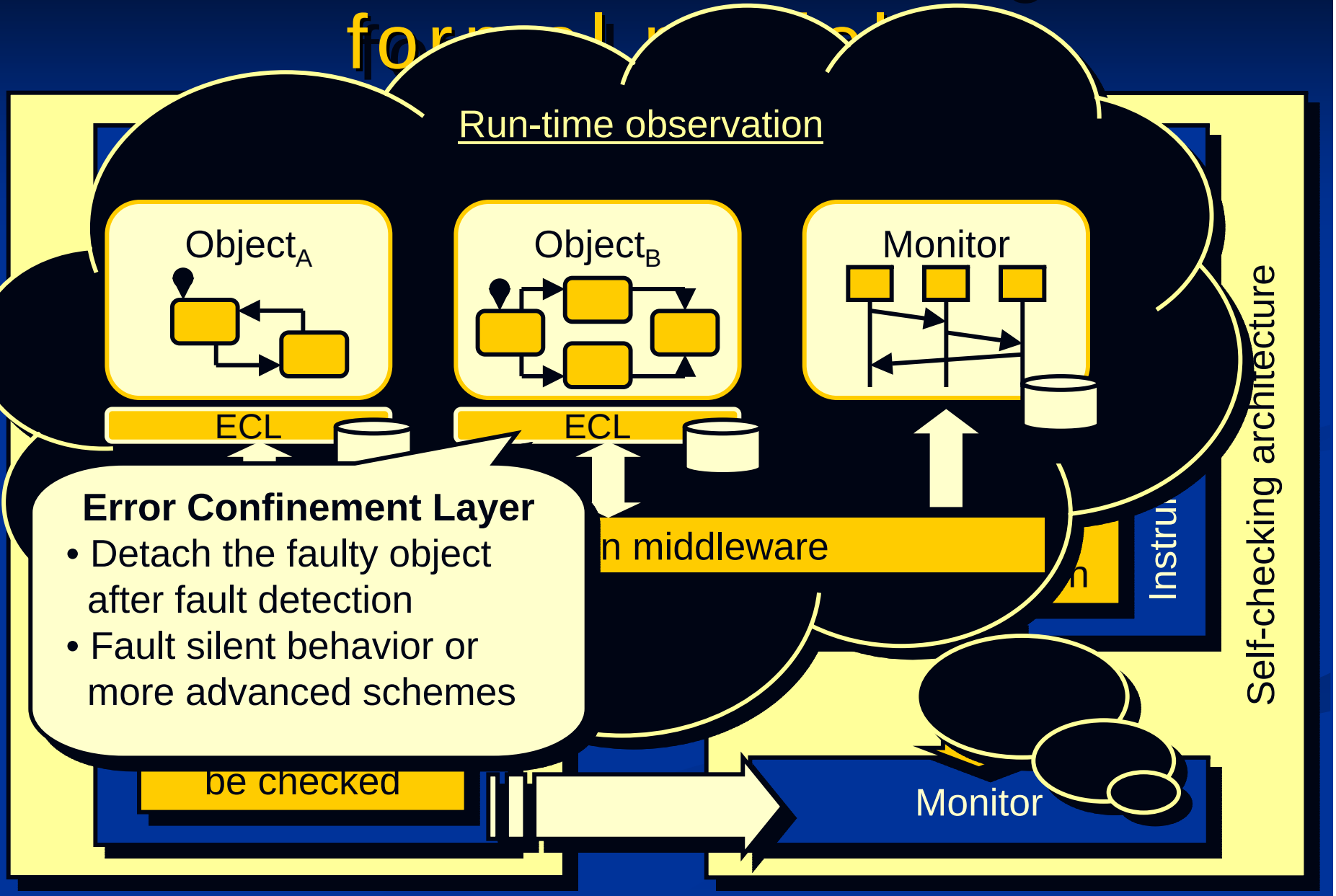
Run-time verification against for



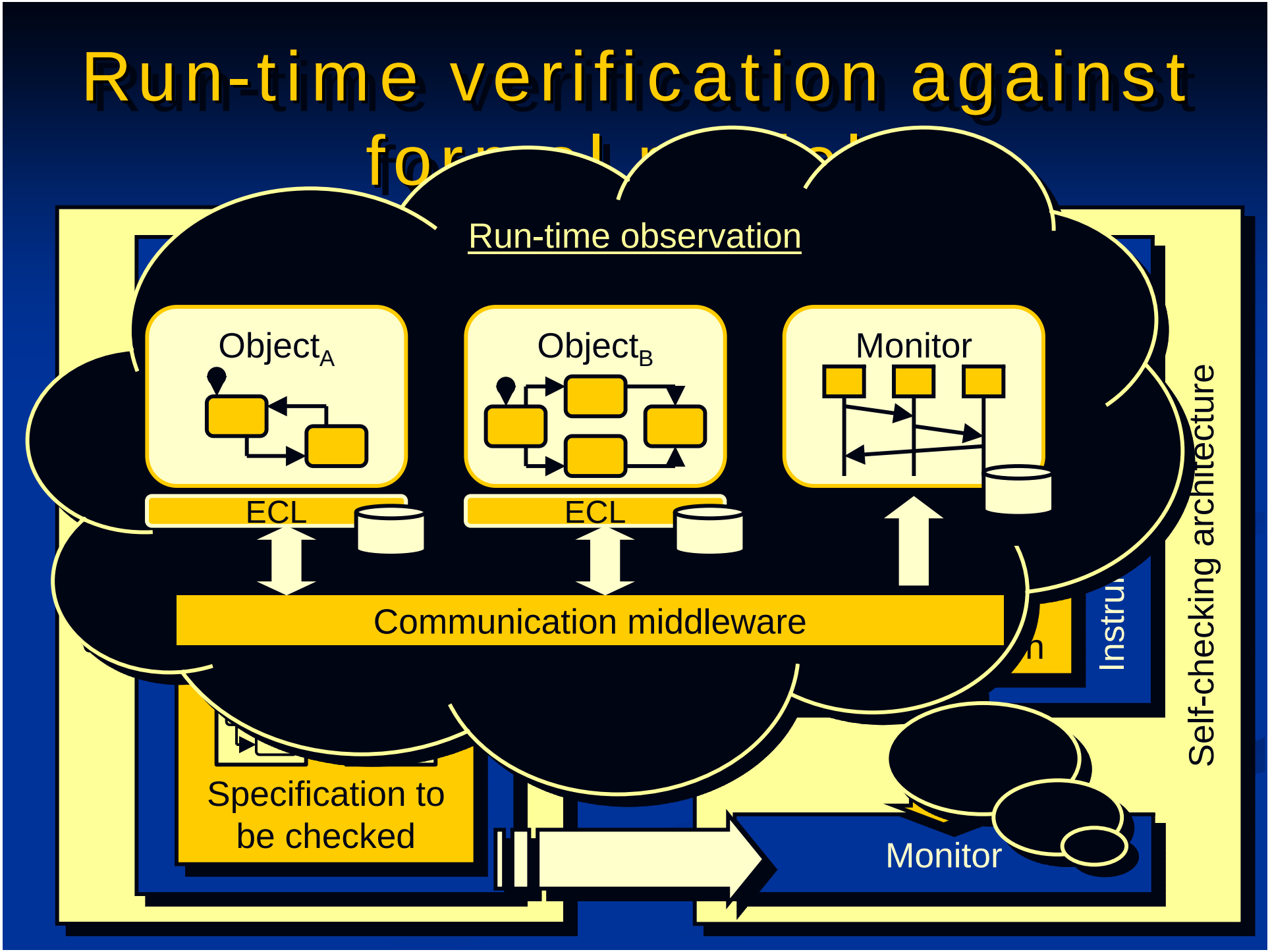
Run-time verification against formal



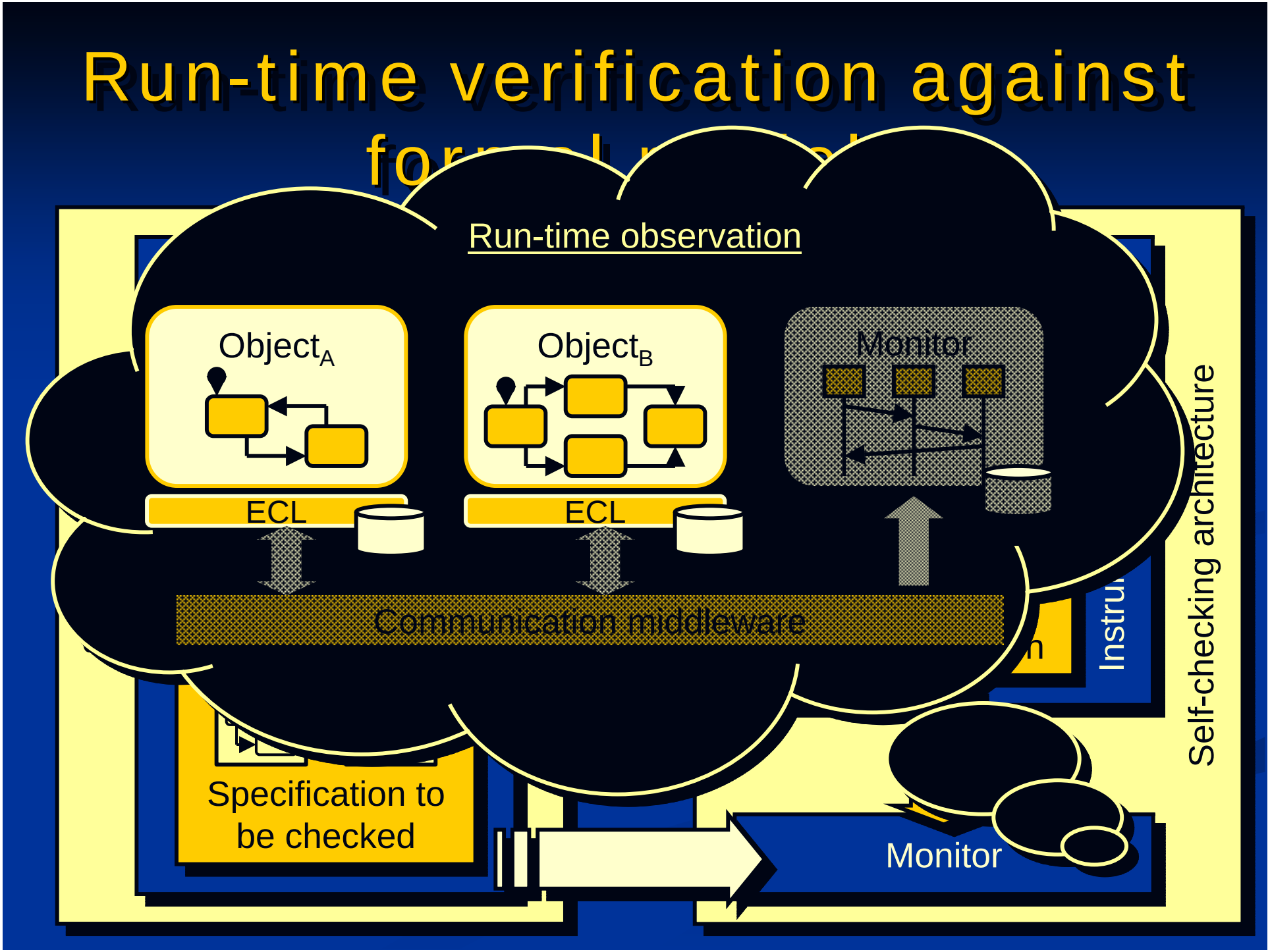
Run-time verification against for



Run-time verification against formal



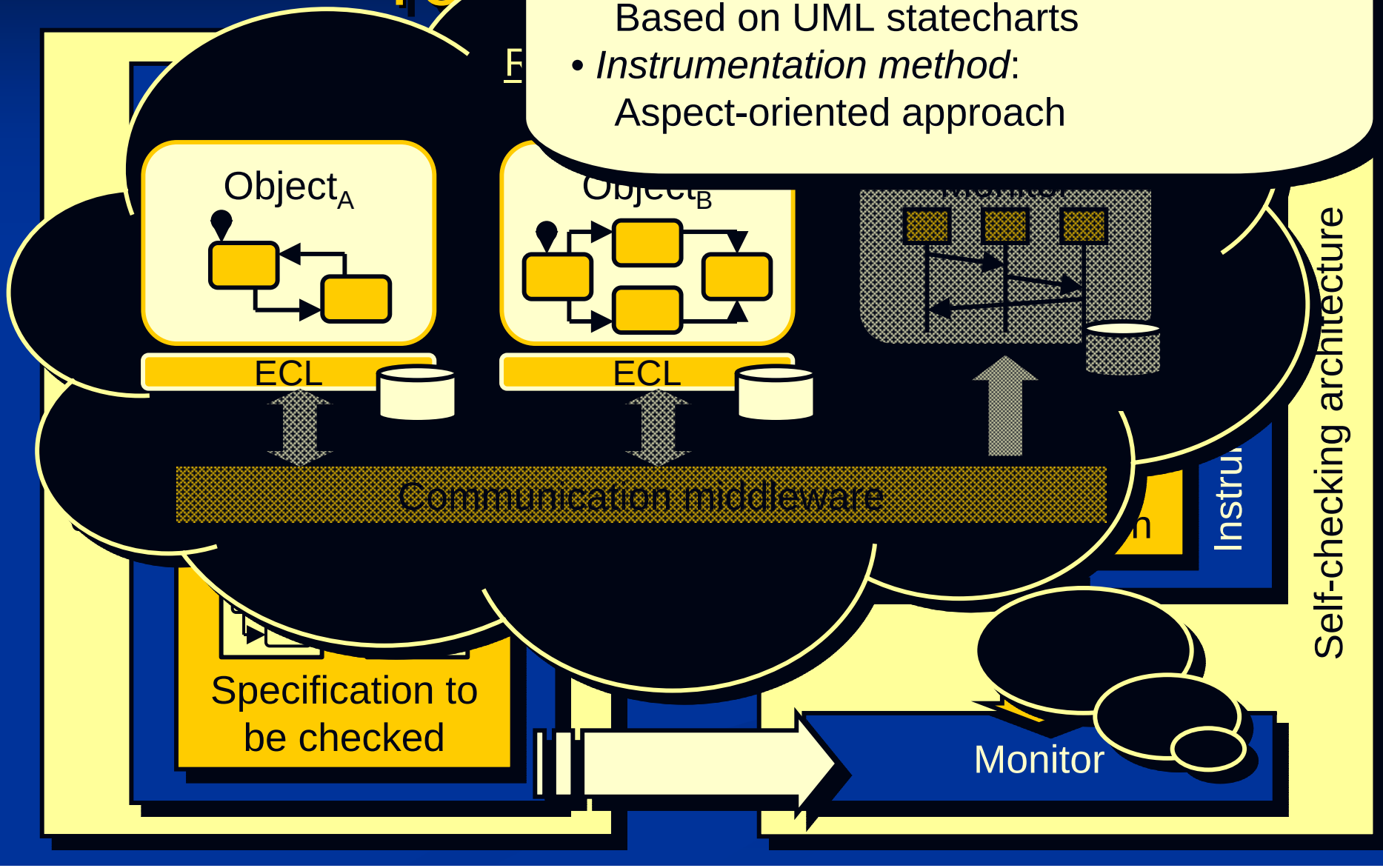
Run-time verification against formal



Run-time v
for

Focus and Contribution

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



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.

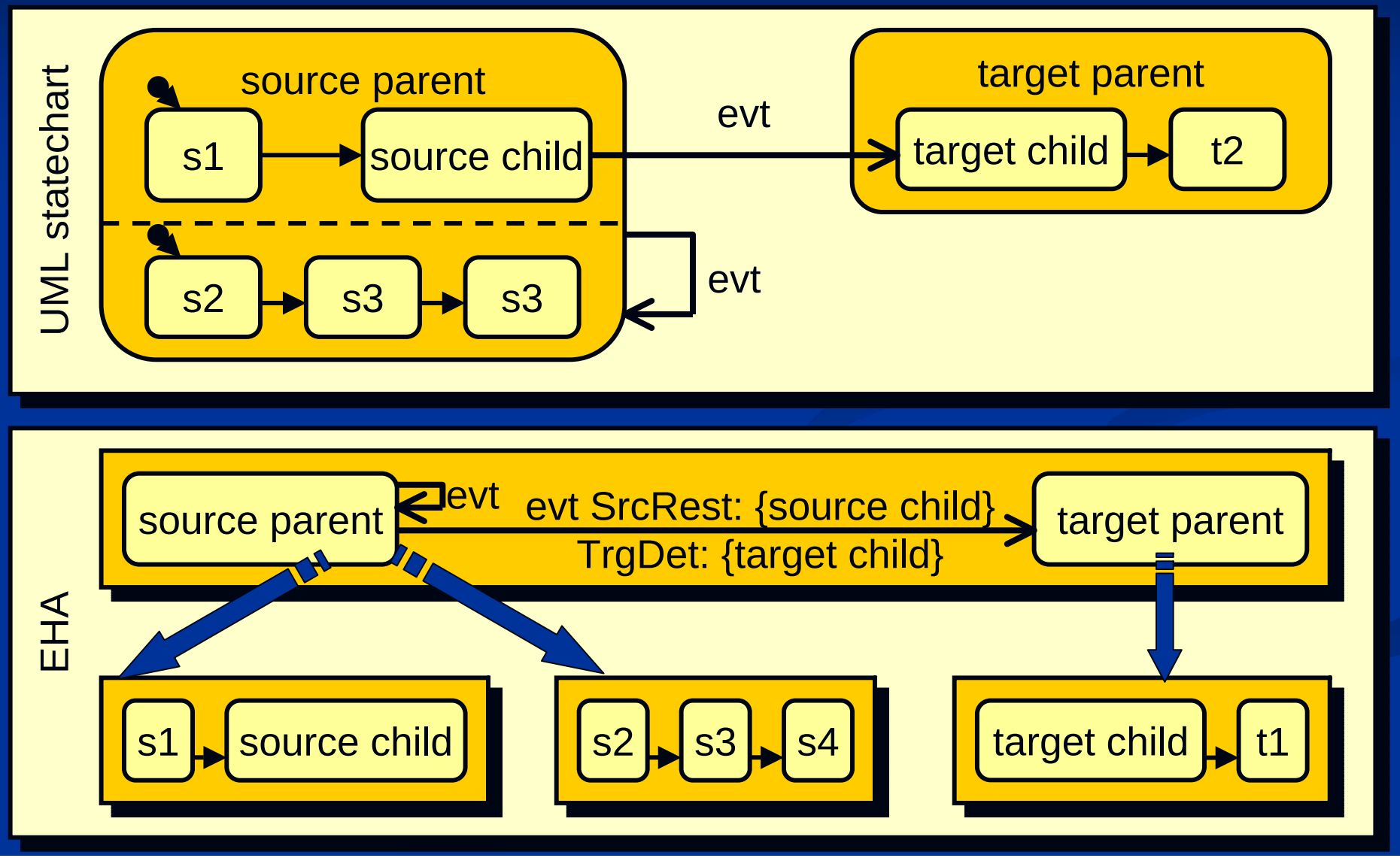
Implementation of the monitor:
- Based on the operational semantics of the behavioral model
- Run-time checking of the behavior on the basis of the abstract reference model

Implementation of the instrumentation:
- Providing information to the monitor about the internal behavior
- Configurable, transparent and automatically applied

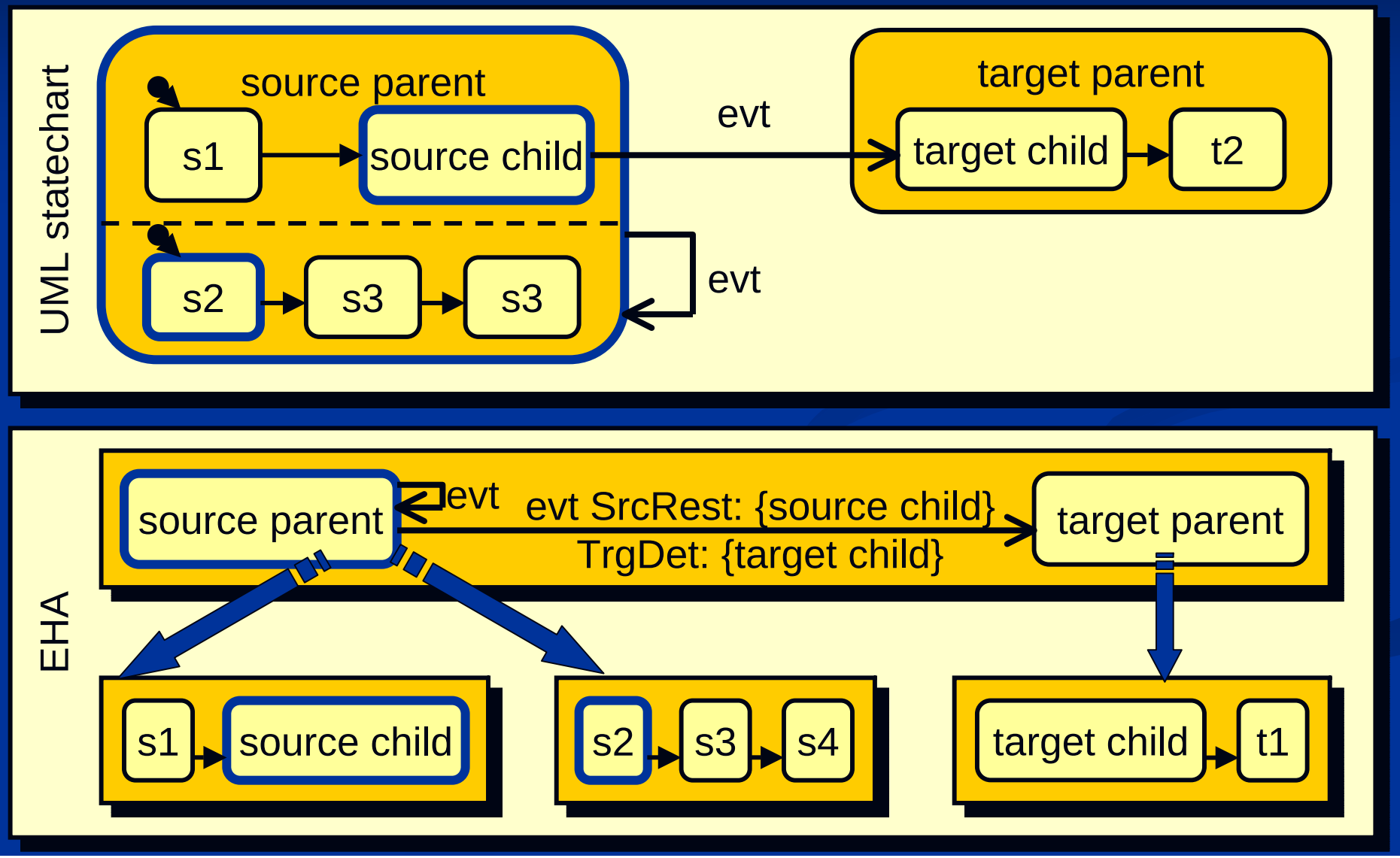
Reference information of the internal behavior

- 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

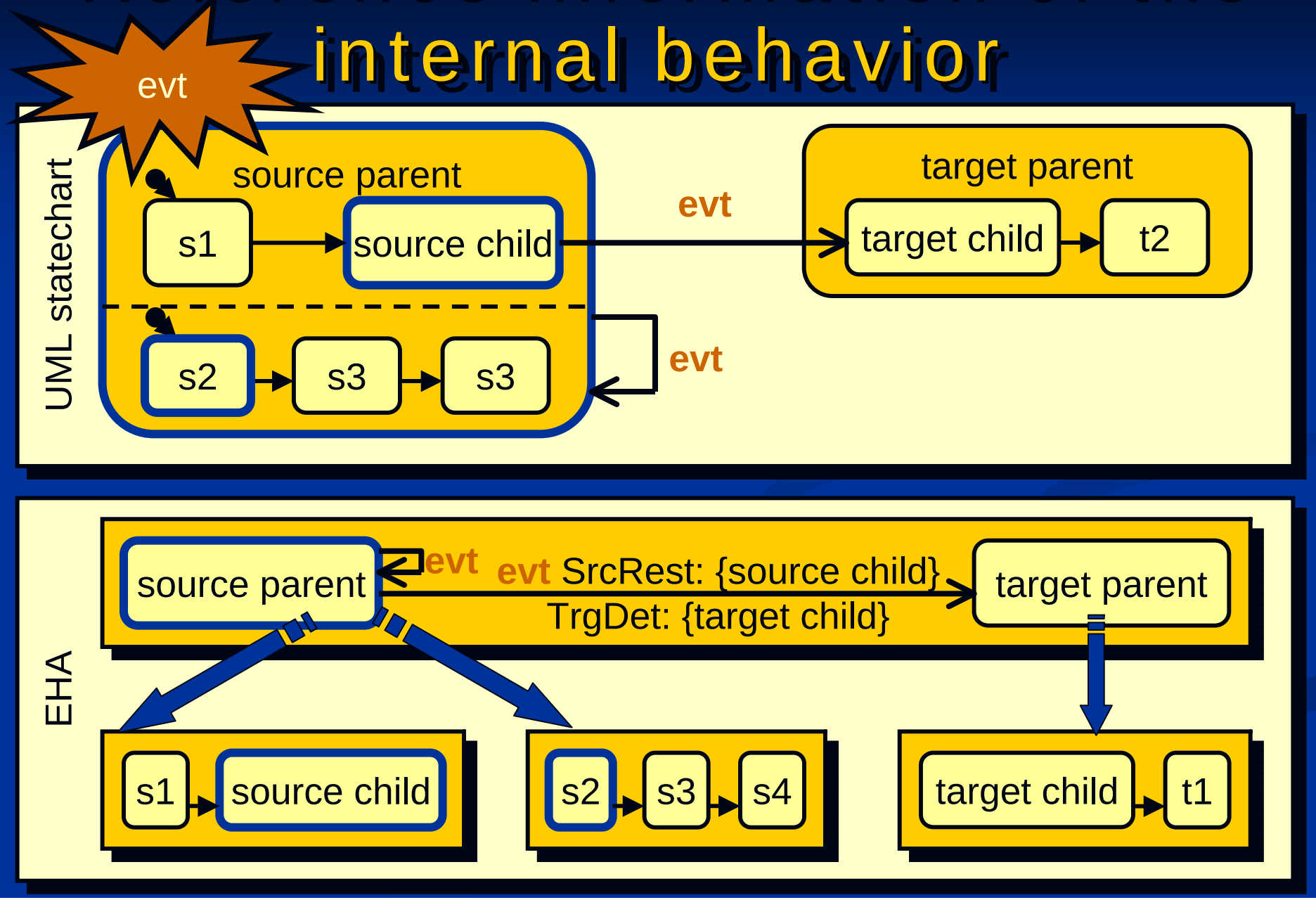
Reference information of the internal behavior



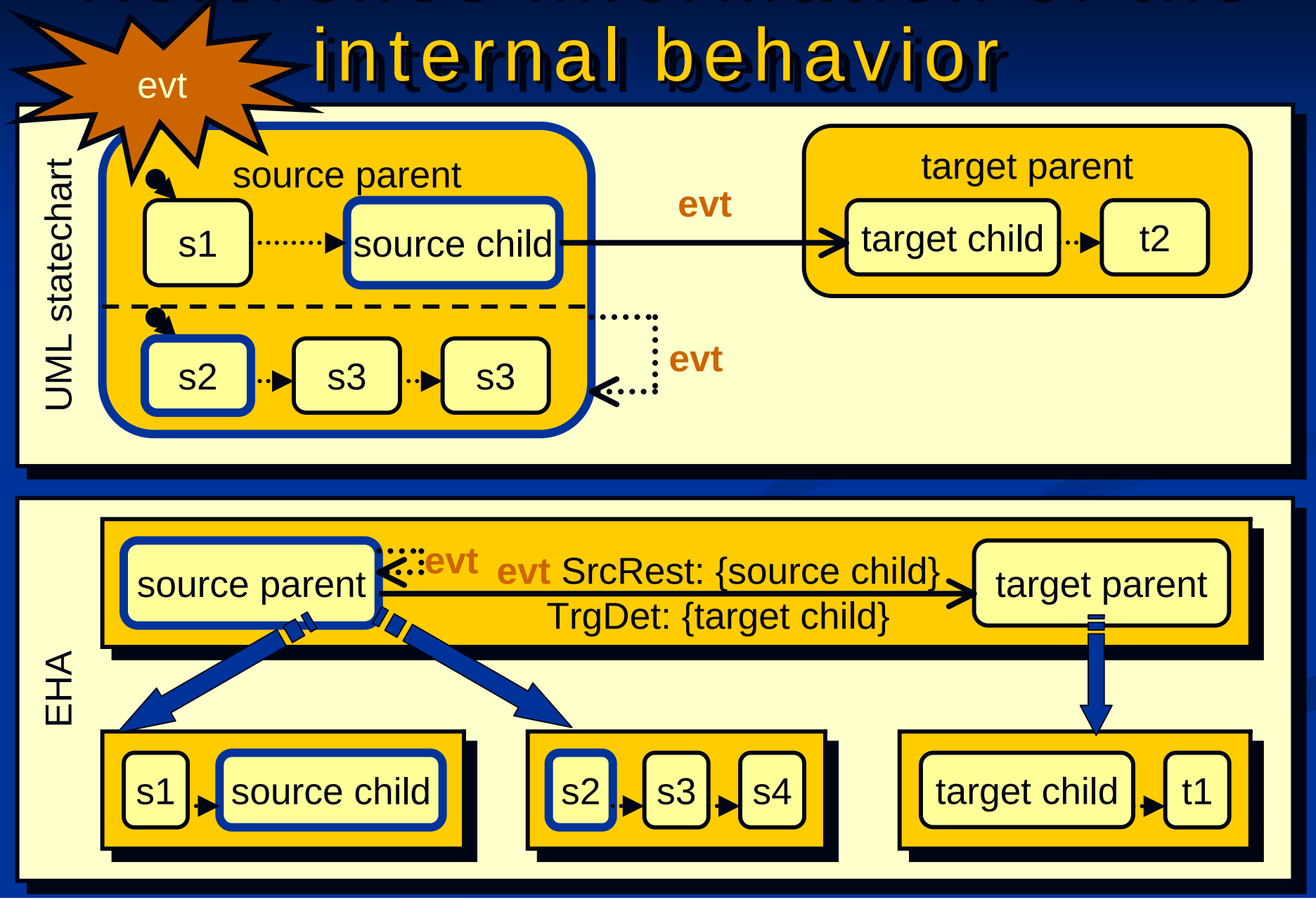
Reference information of the internal behavior



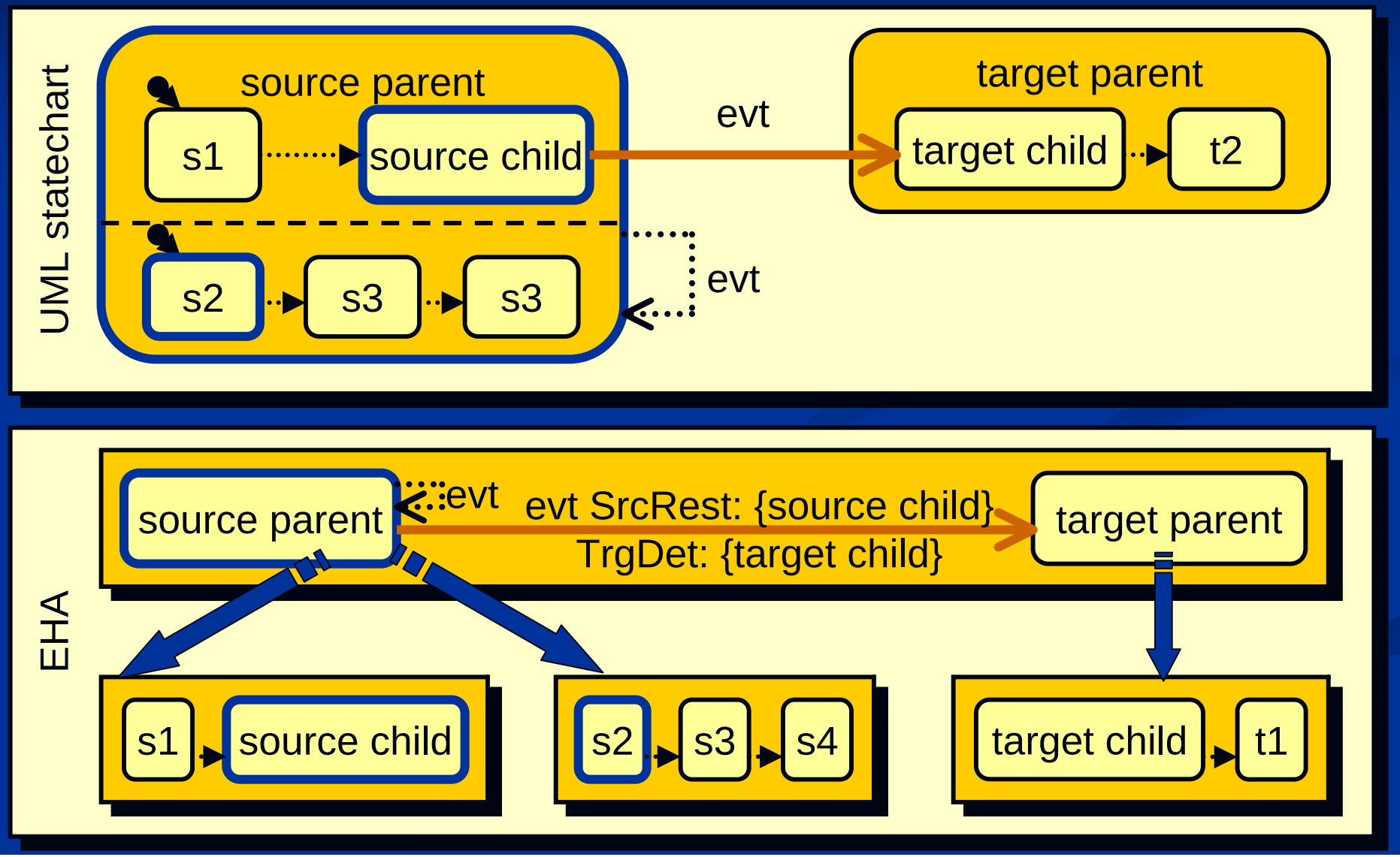
Reference information of the internal behavior



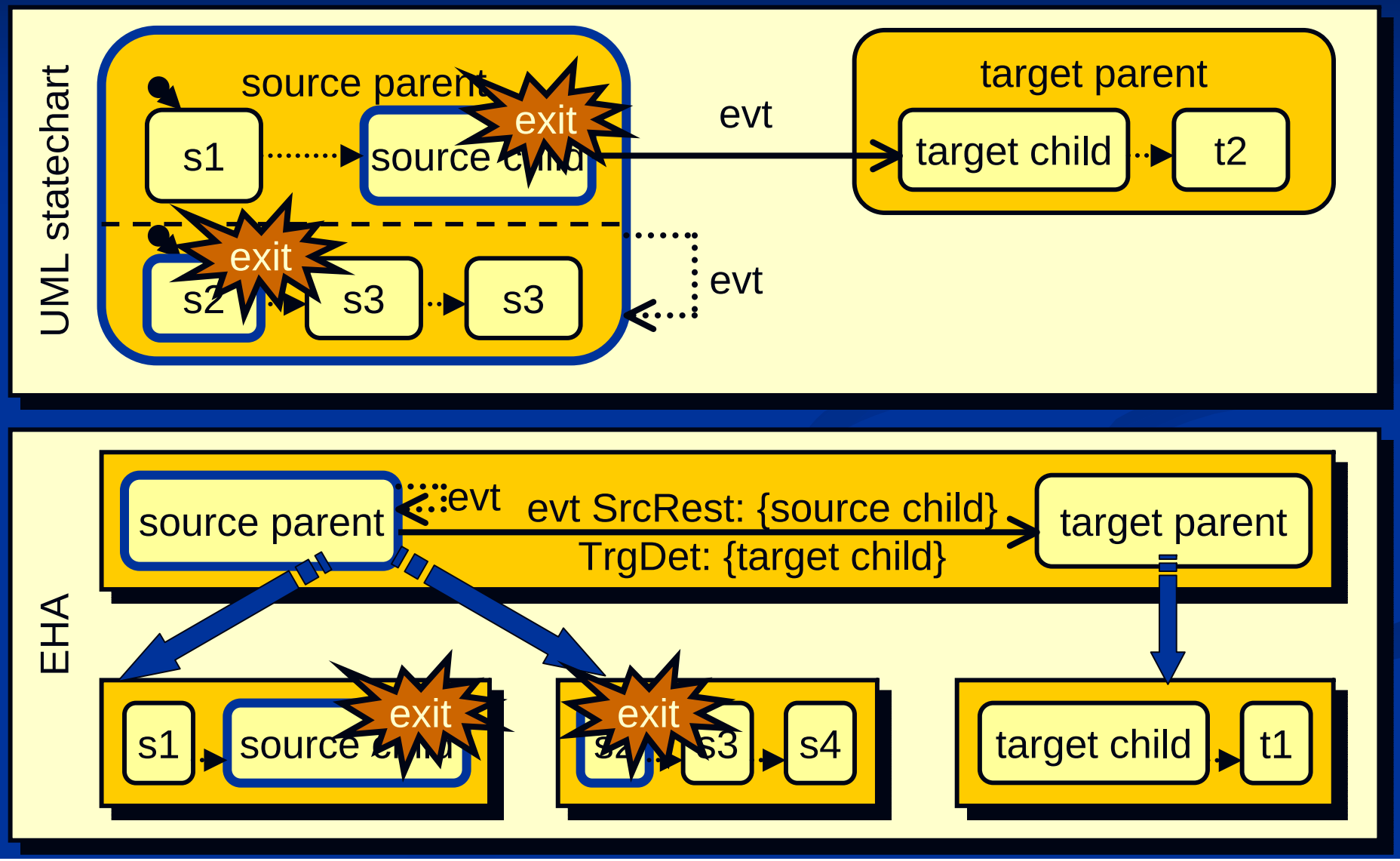
Reference information of the internal behavior



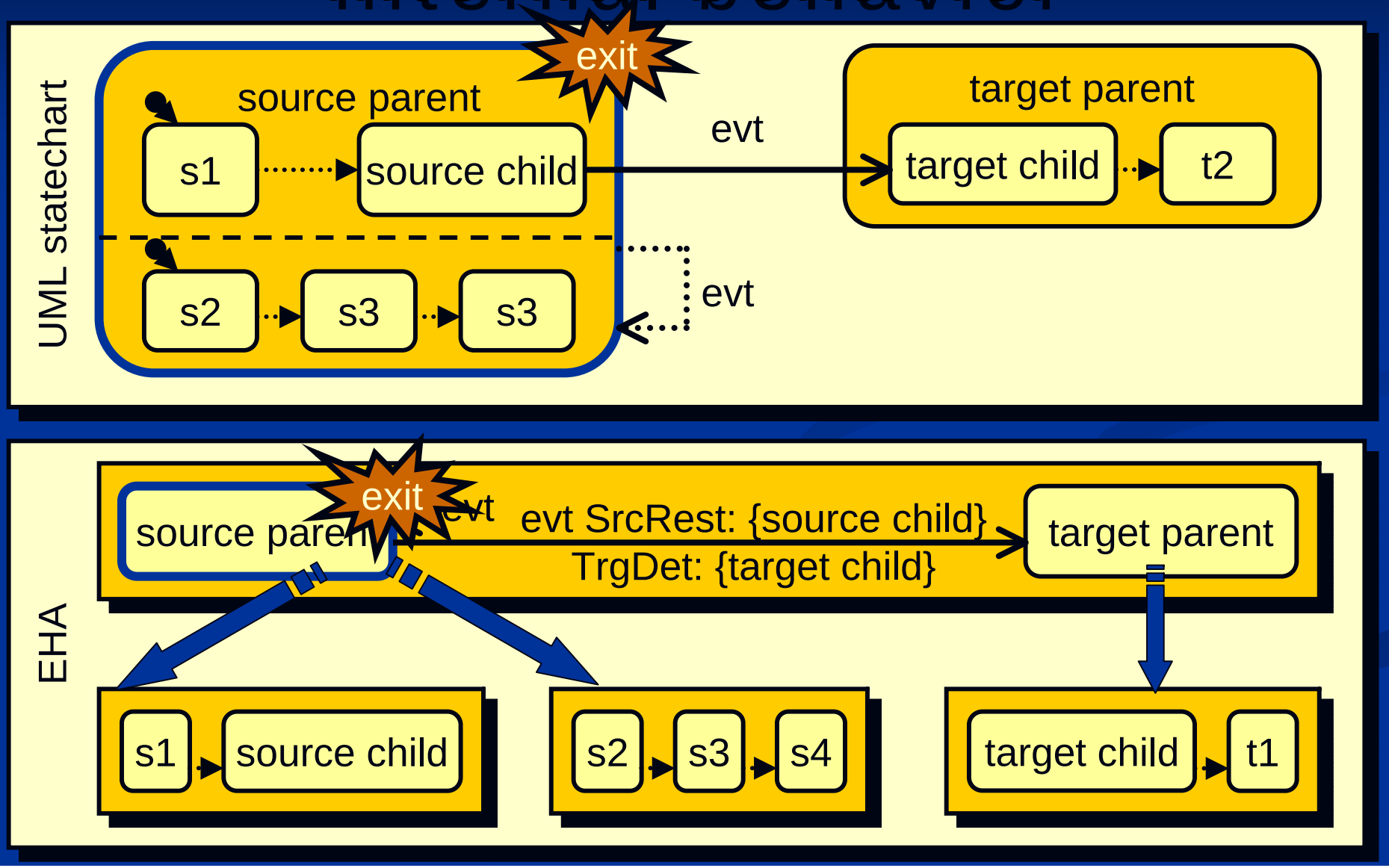
Reference information of the internal behavior



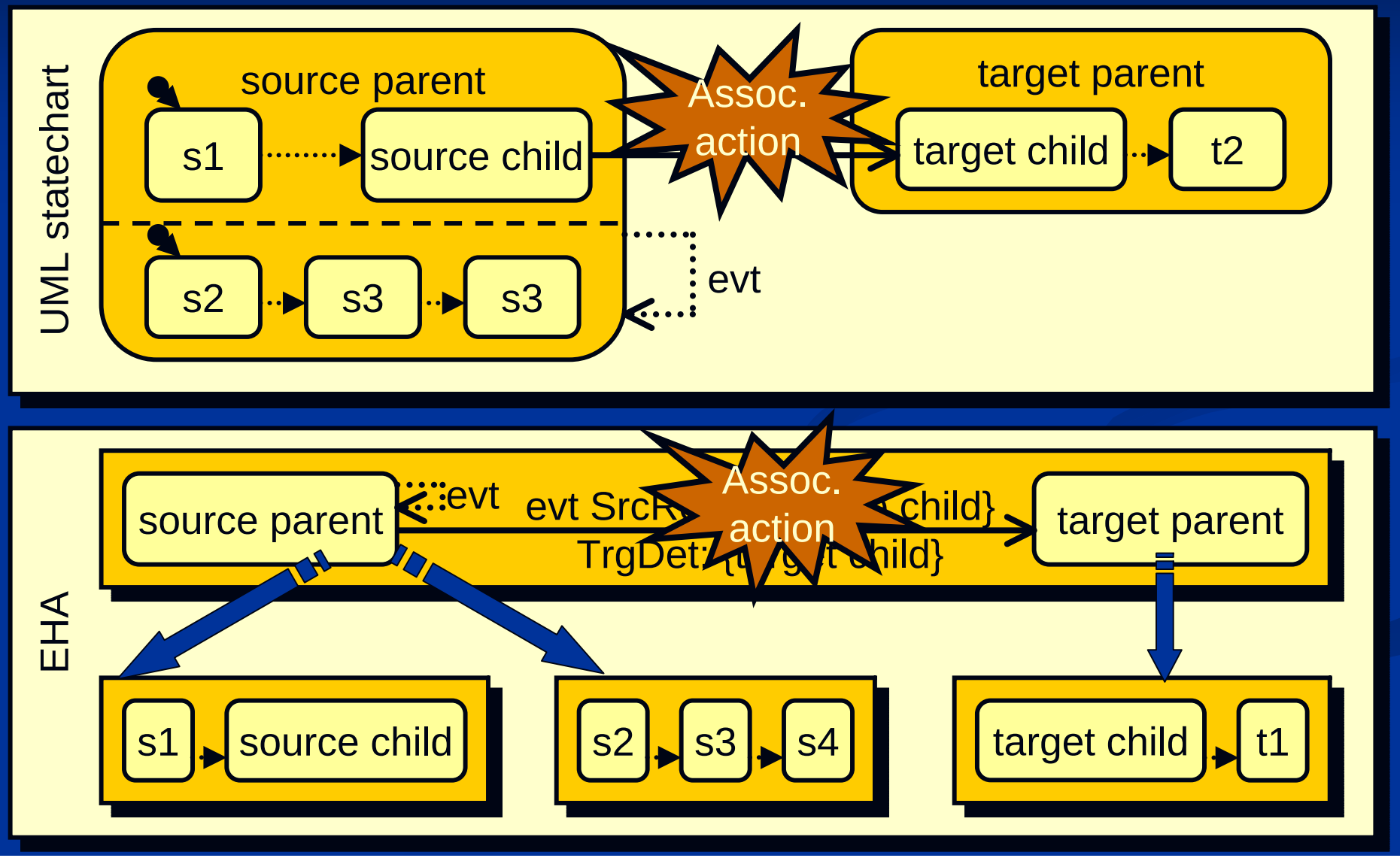
Reference information of the internal behavior



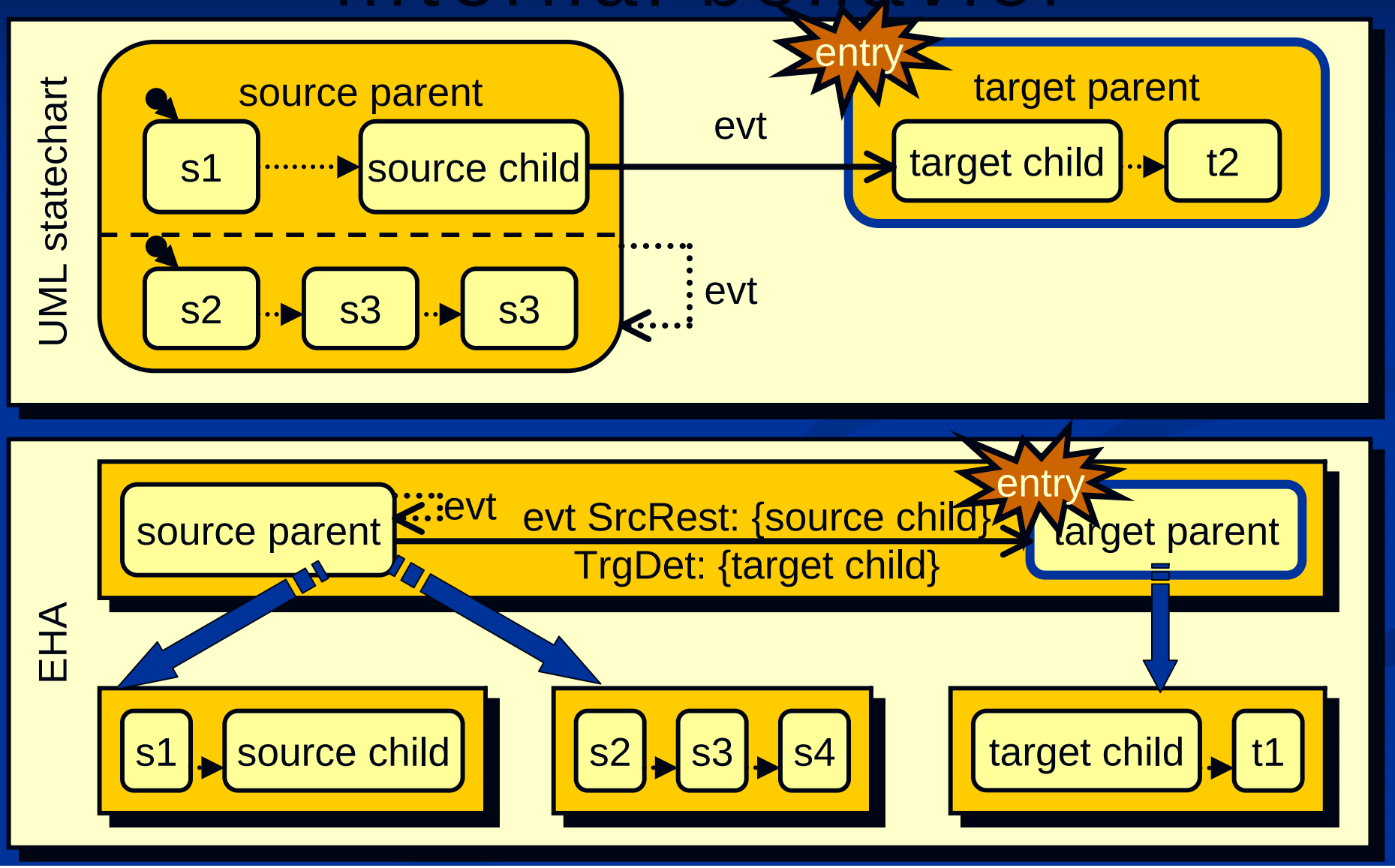
Reference information of the internal behavior



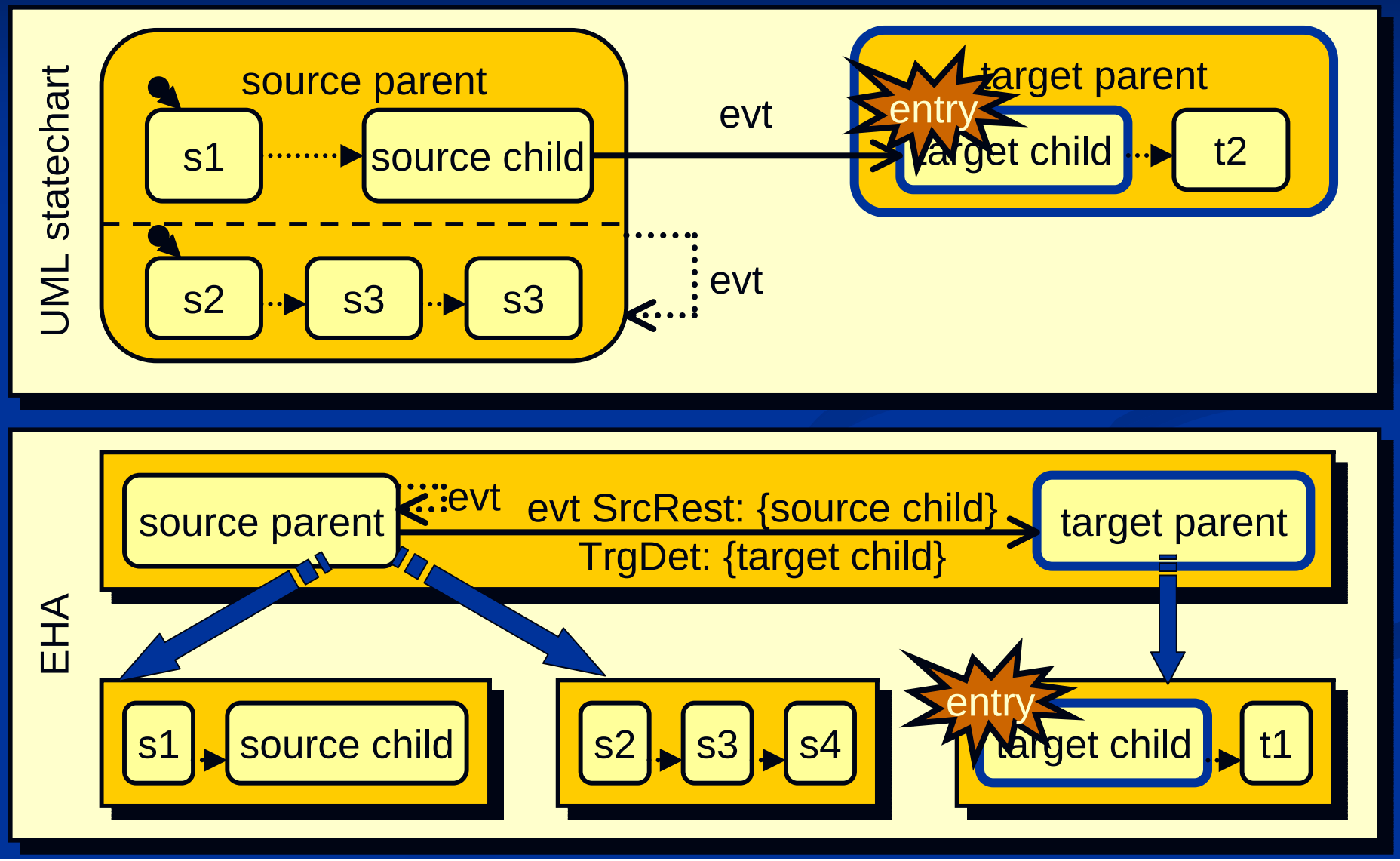
Reference information of the internal behavior



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

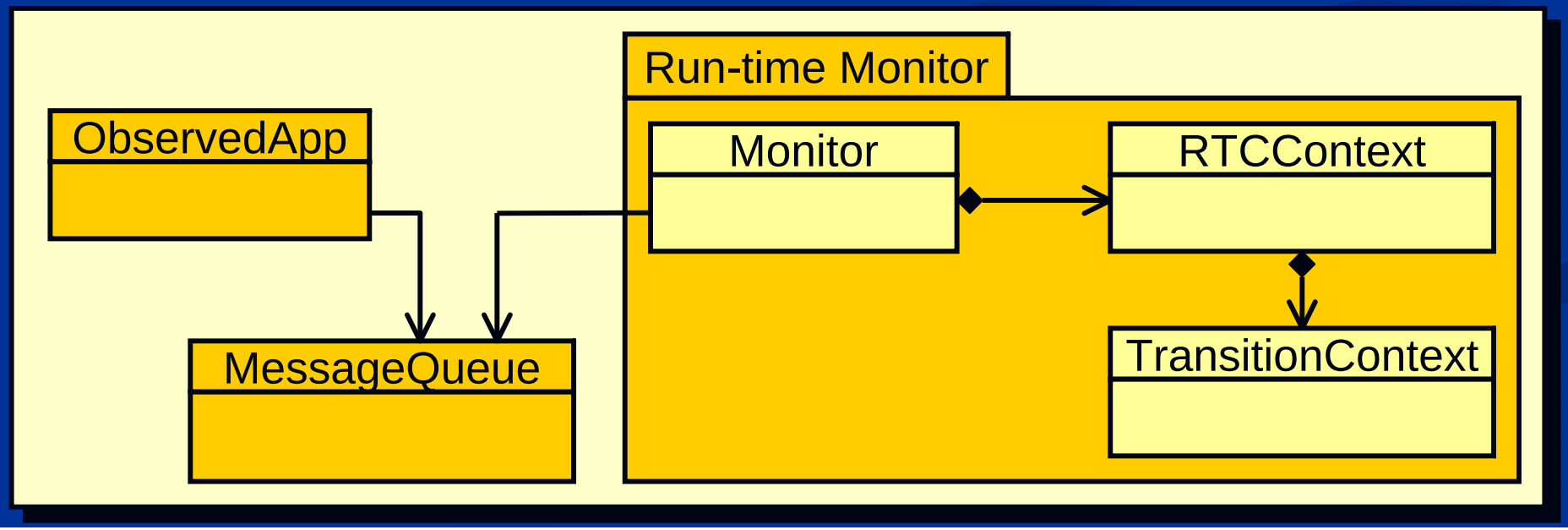
Reference information:
- Automatically derived from the behavioral specification
- Capable of expressing state hierarchies, concurrent operation, etc.

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

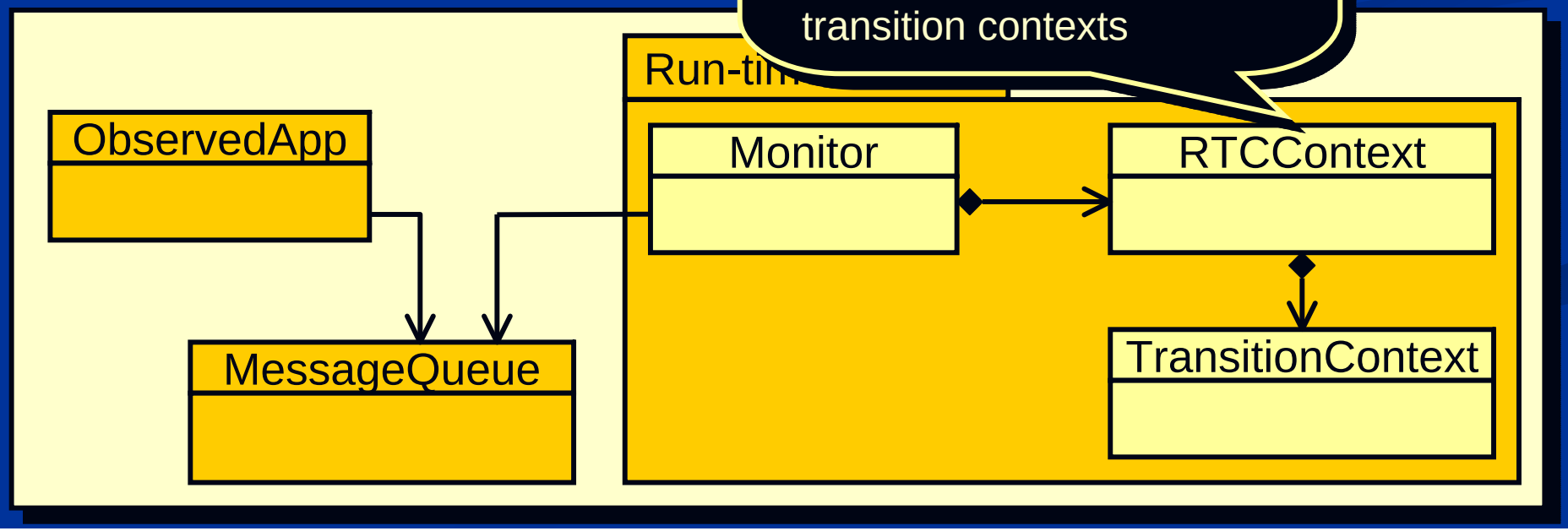
- Structural decomposition:
  - *Run-to-completion* and *transition contexts*
- Specification of contexts:
  - Protocol state machines (statecharts)



Checking the internal behavior

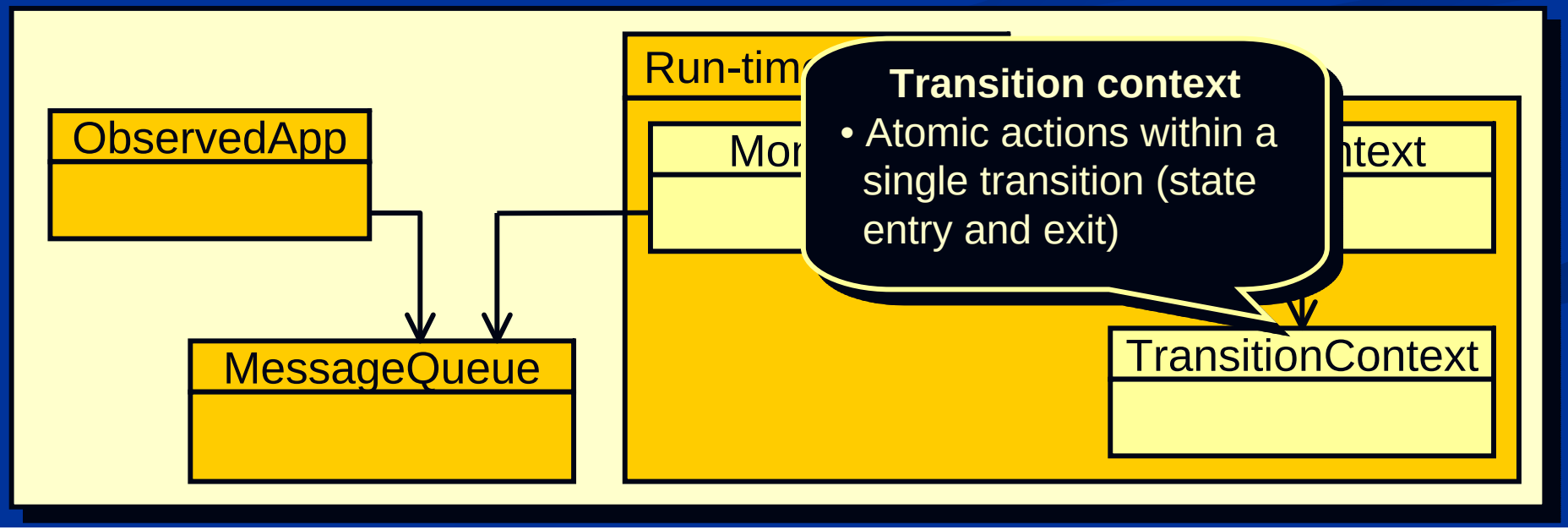
- Structural decomposition:
  - *Run-to-completion and transition contexts*
- Specification of cor
  - Protocol state mach

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



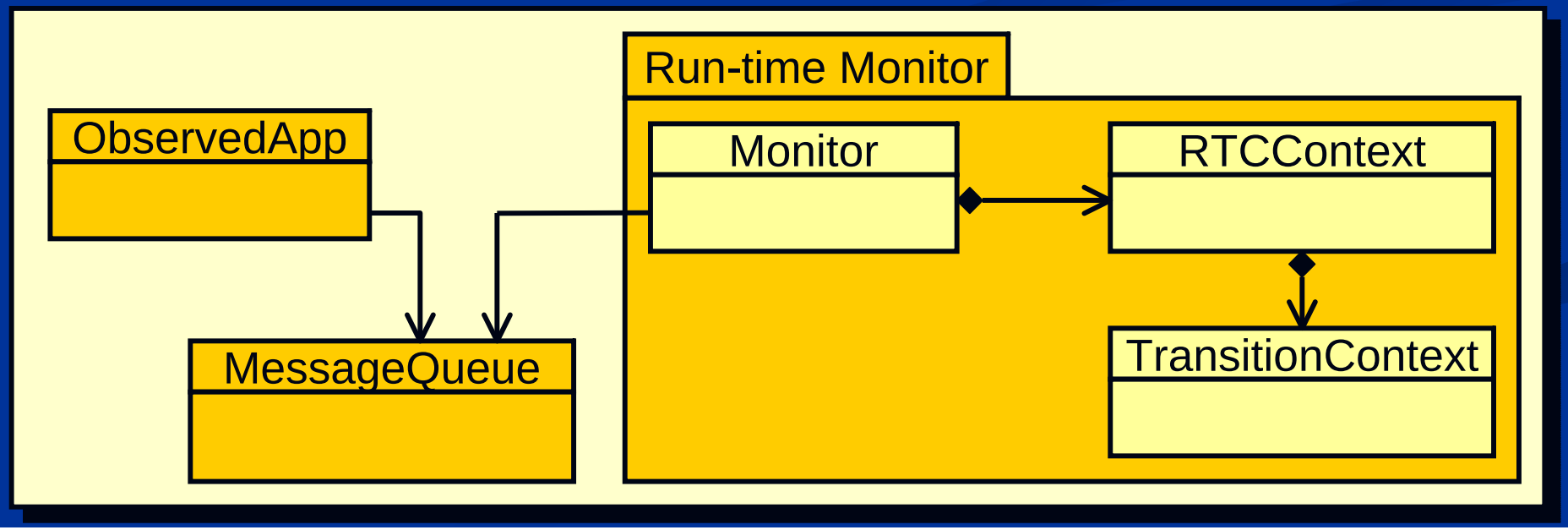
Checking the internal behavior

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



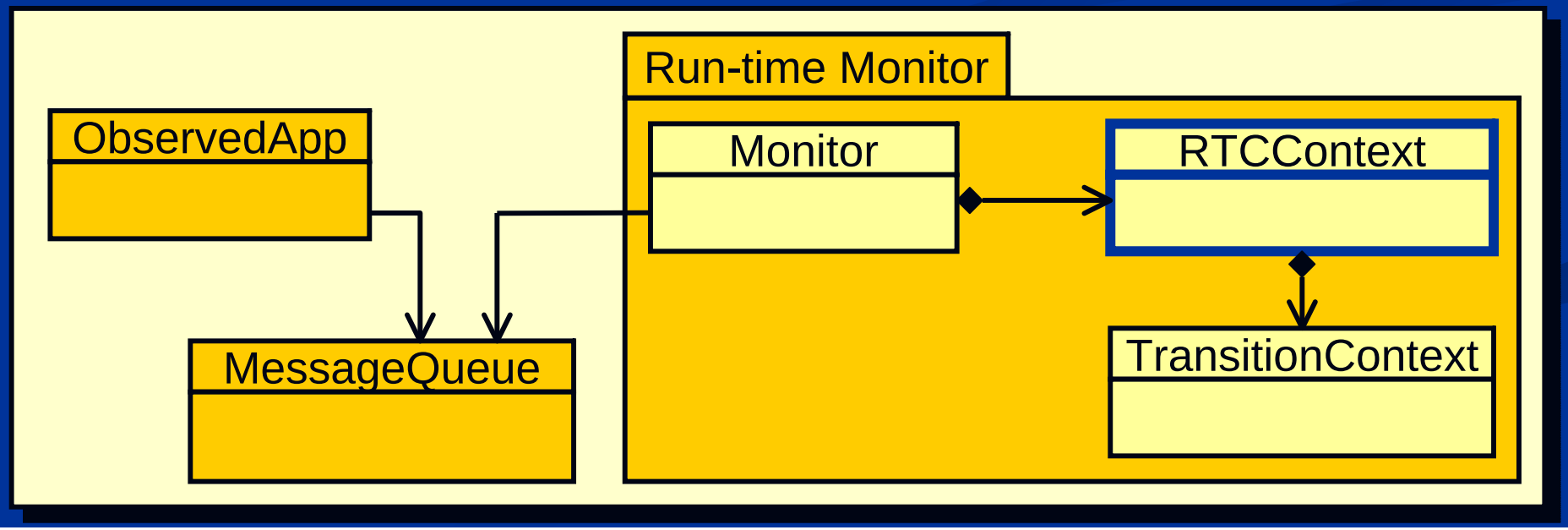
Checking the internal behavior

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

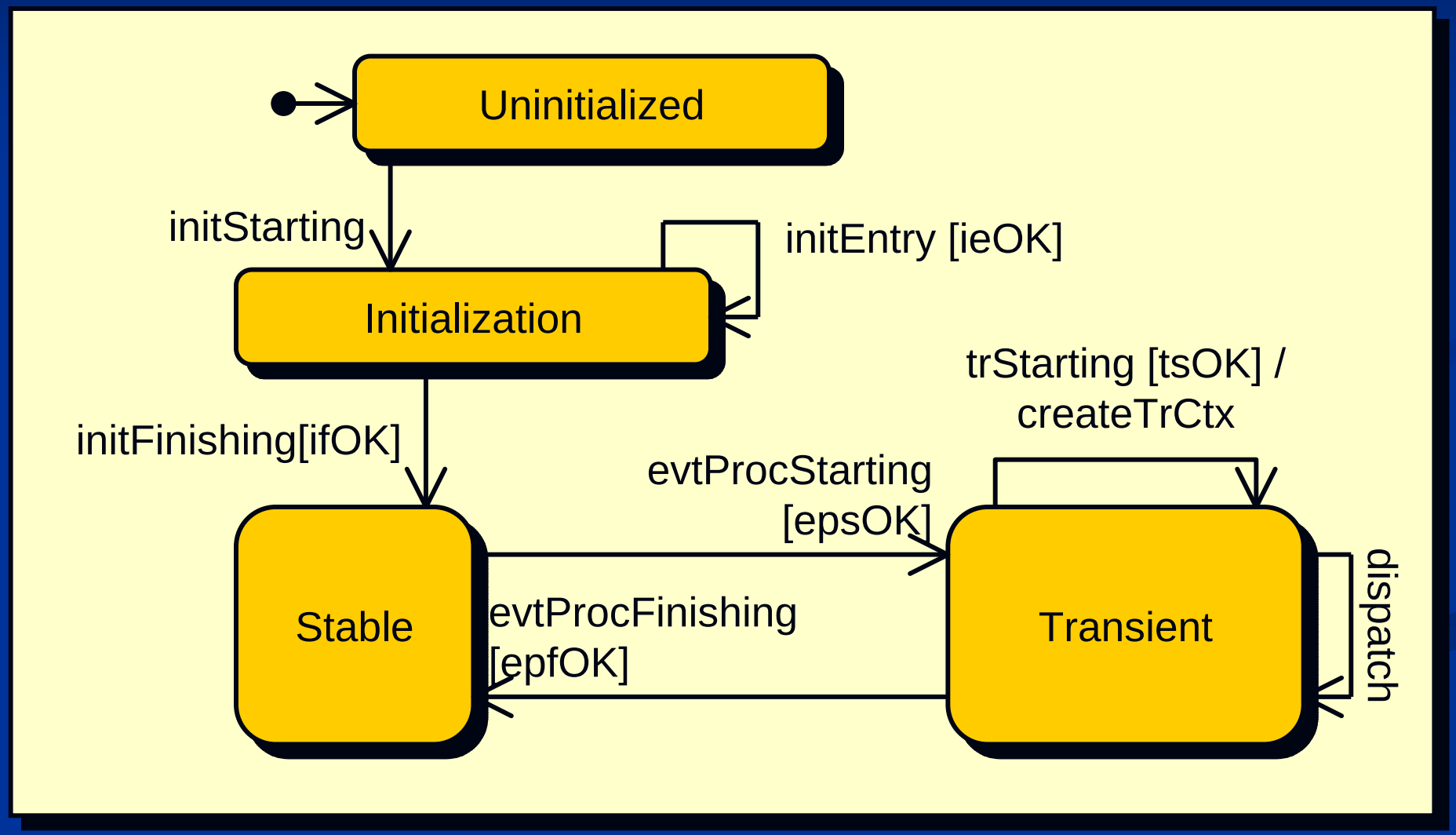


Checking the internal behavior

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

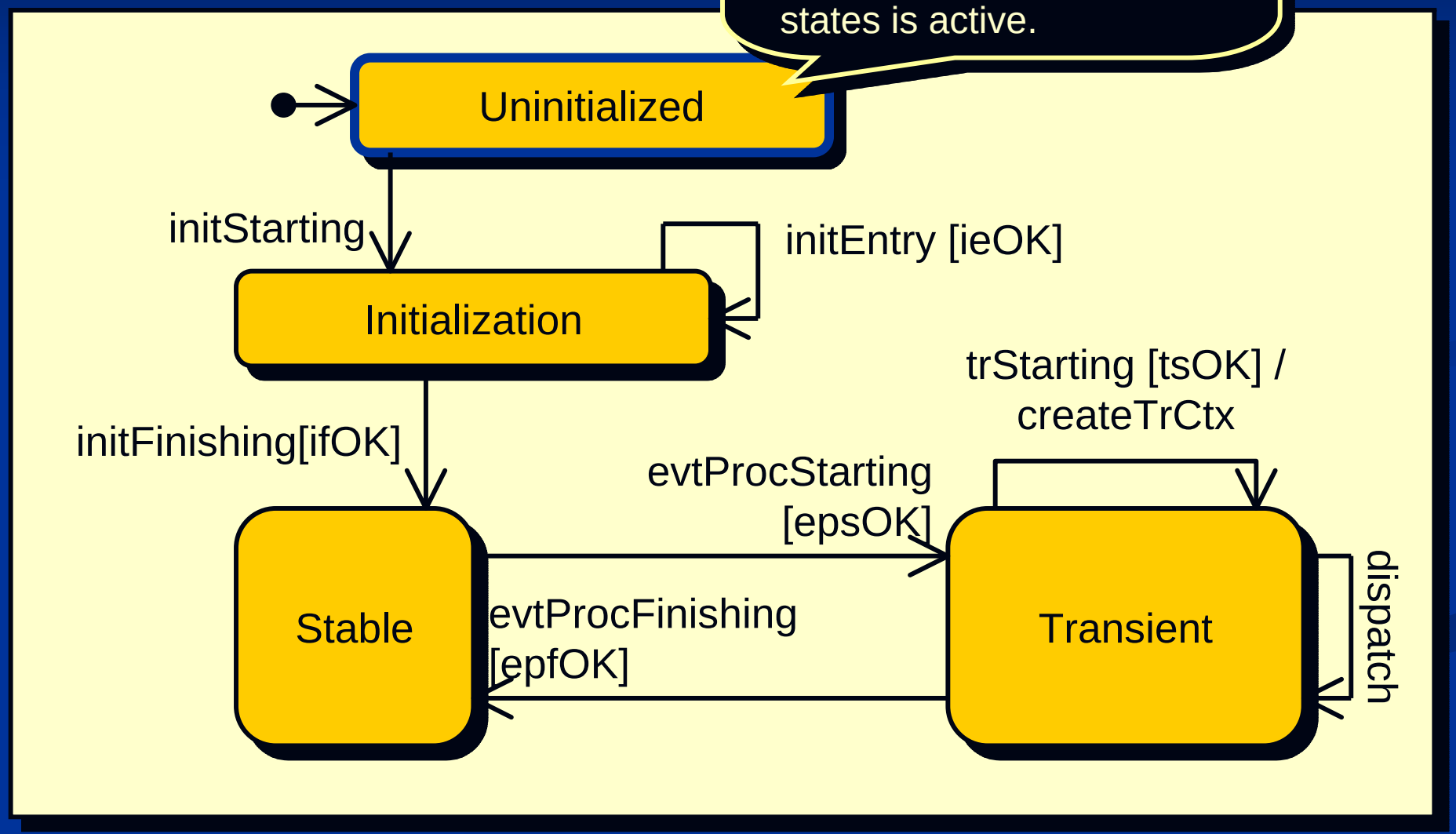


Run-to-completion context

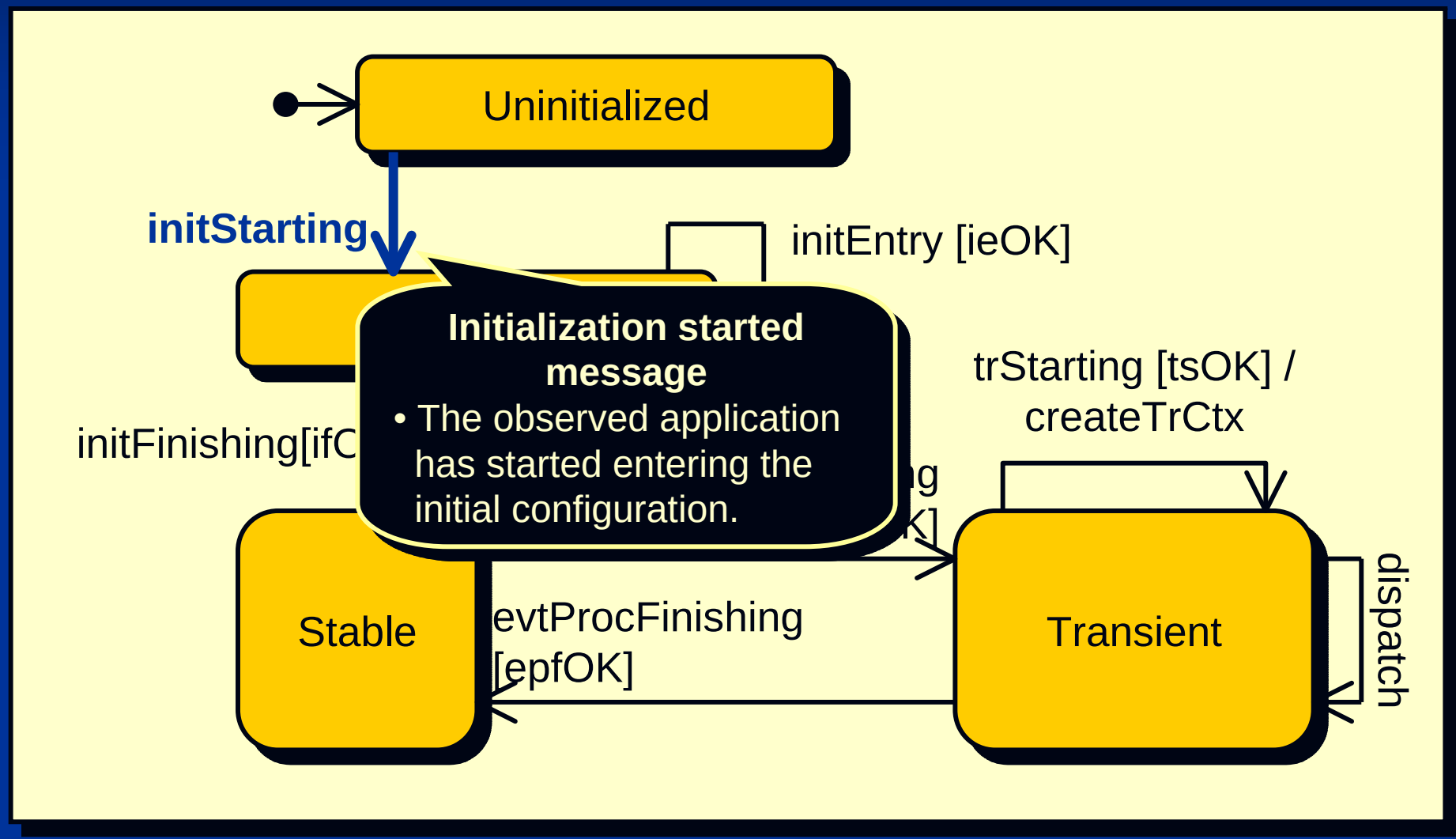


Run-to-compl t

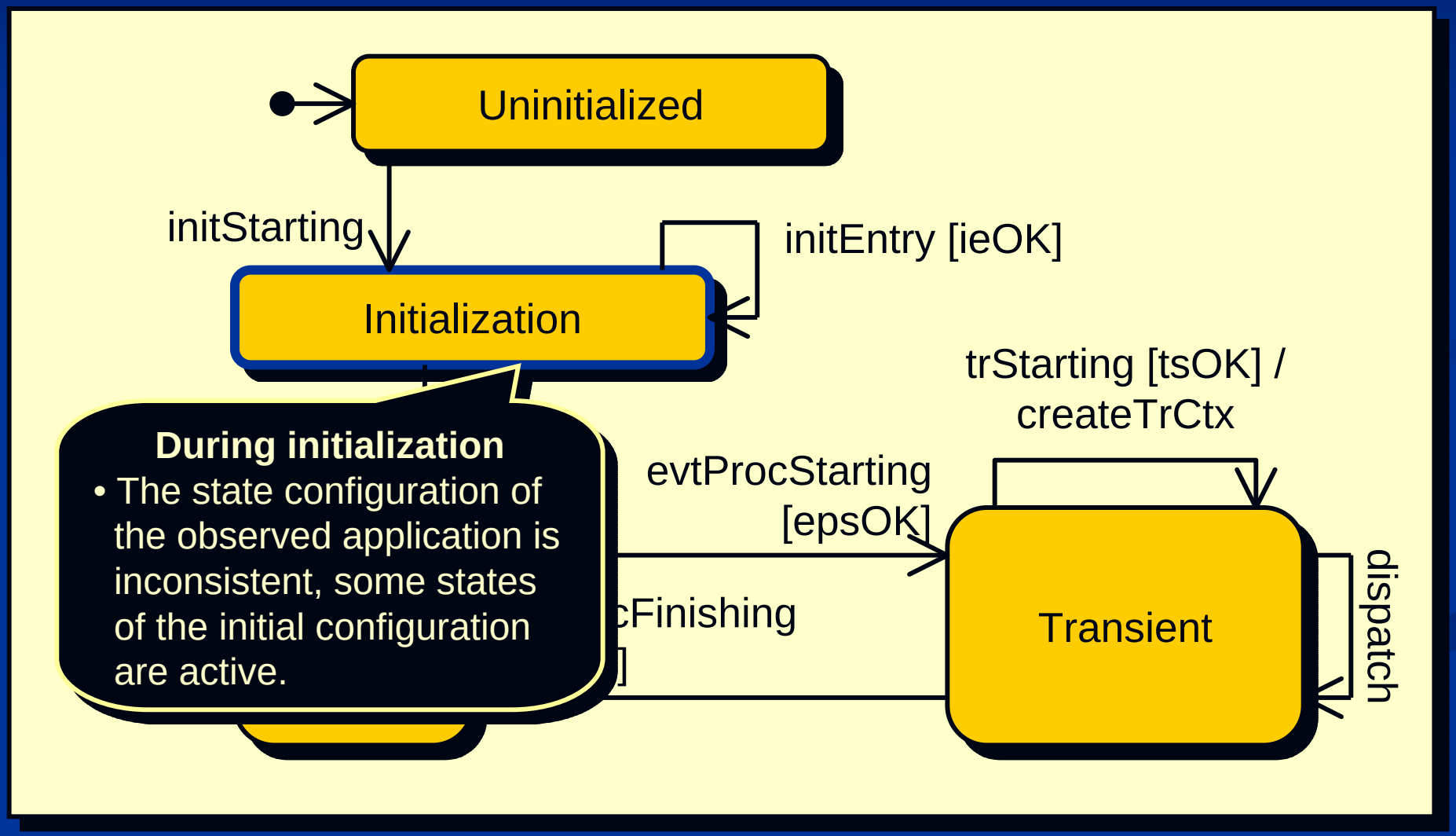
Before initialization
- The state configuration of the observed application is inconsistent, none of the states is active.



Run-to-completion context



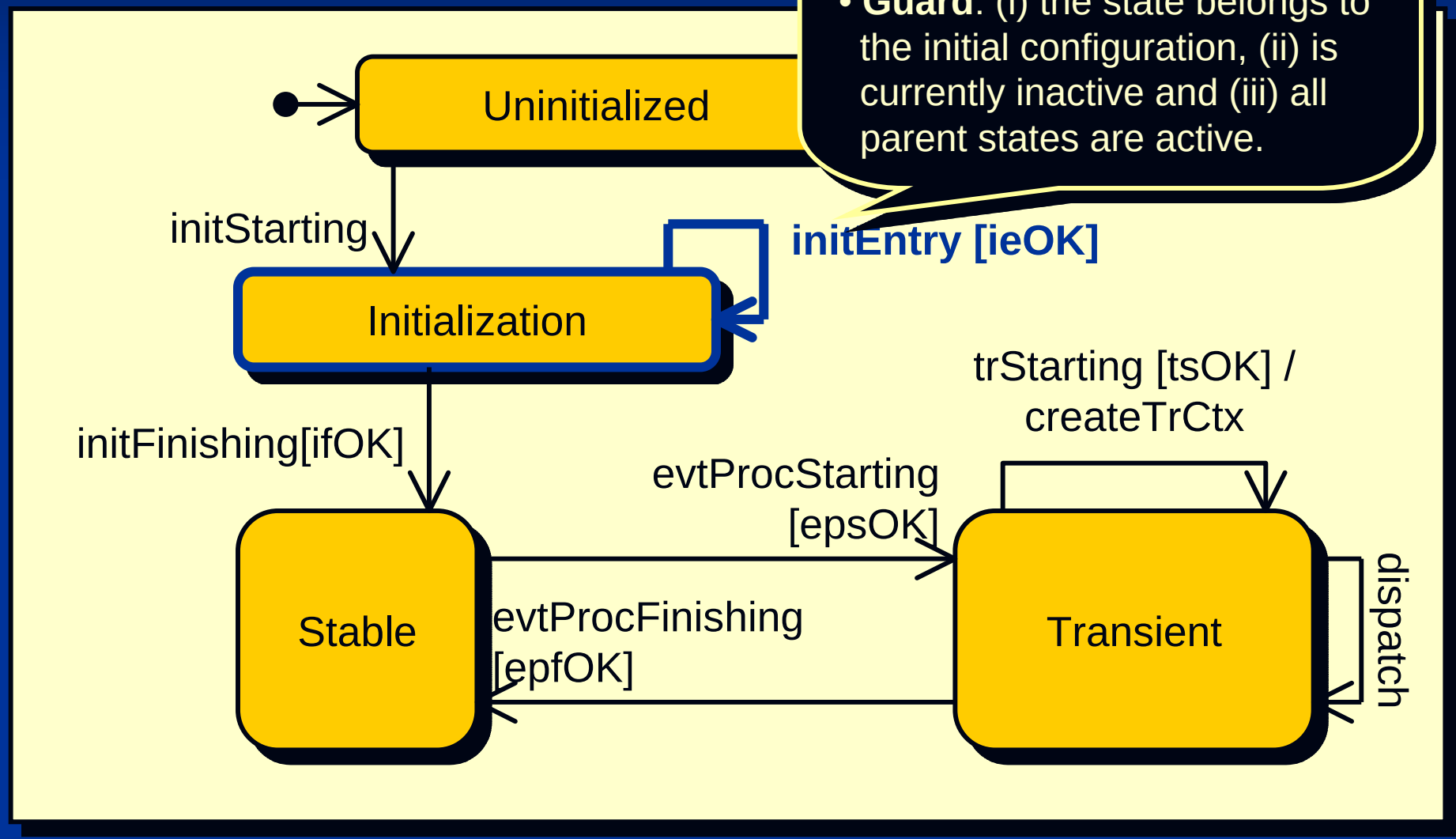
Run-to-completion context



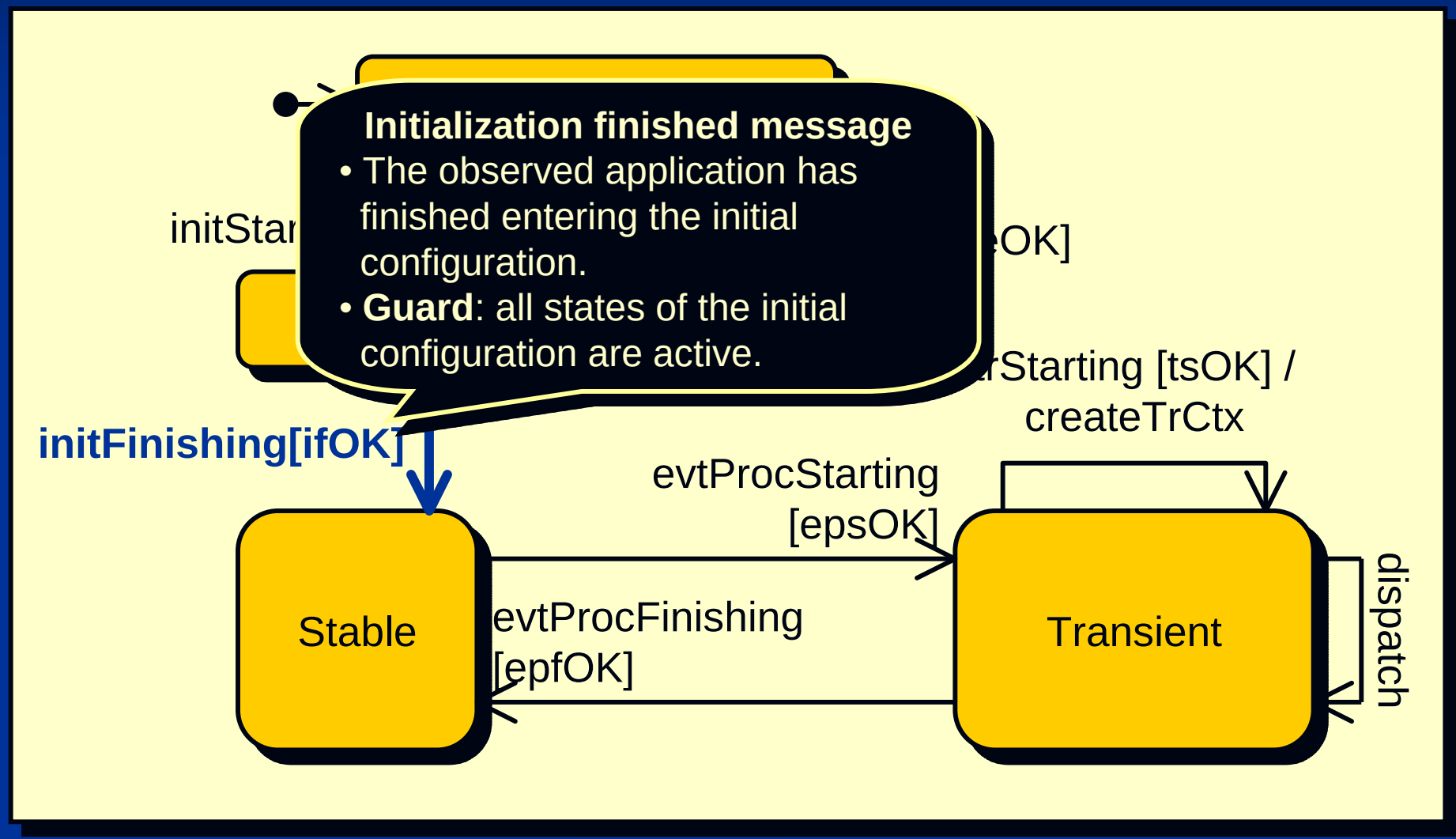
Run-to-complet

Entry during initialization

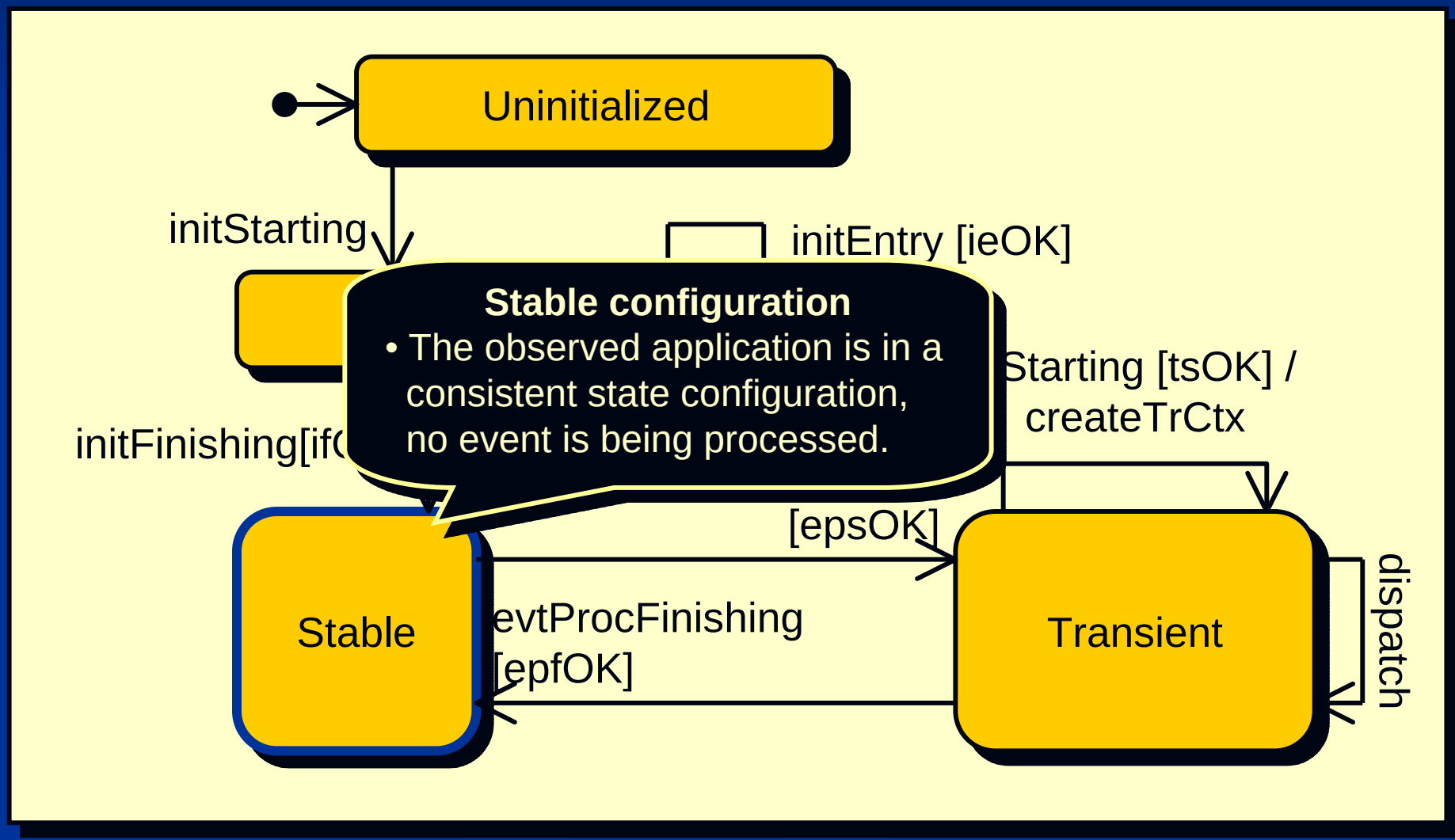
- The observed application has entered a state.
- **Guard**: (i) the state belongs to the initial configuration, (ii) is currently inactive and (iii) all parent states are active.



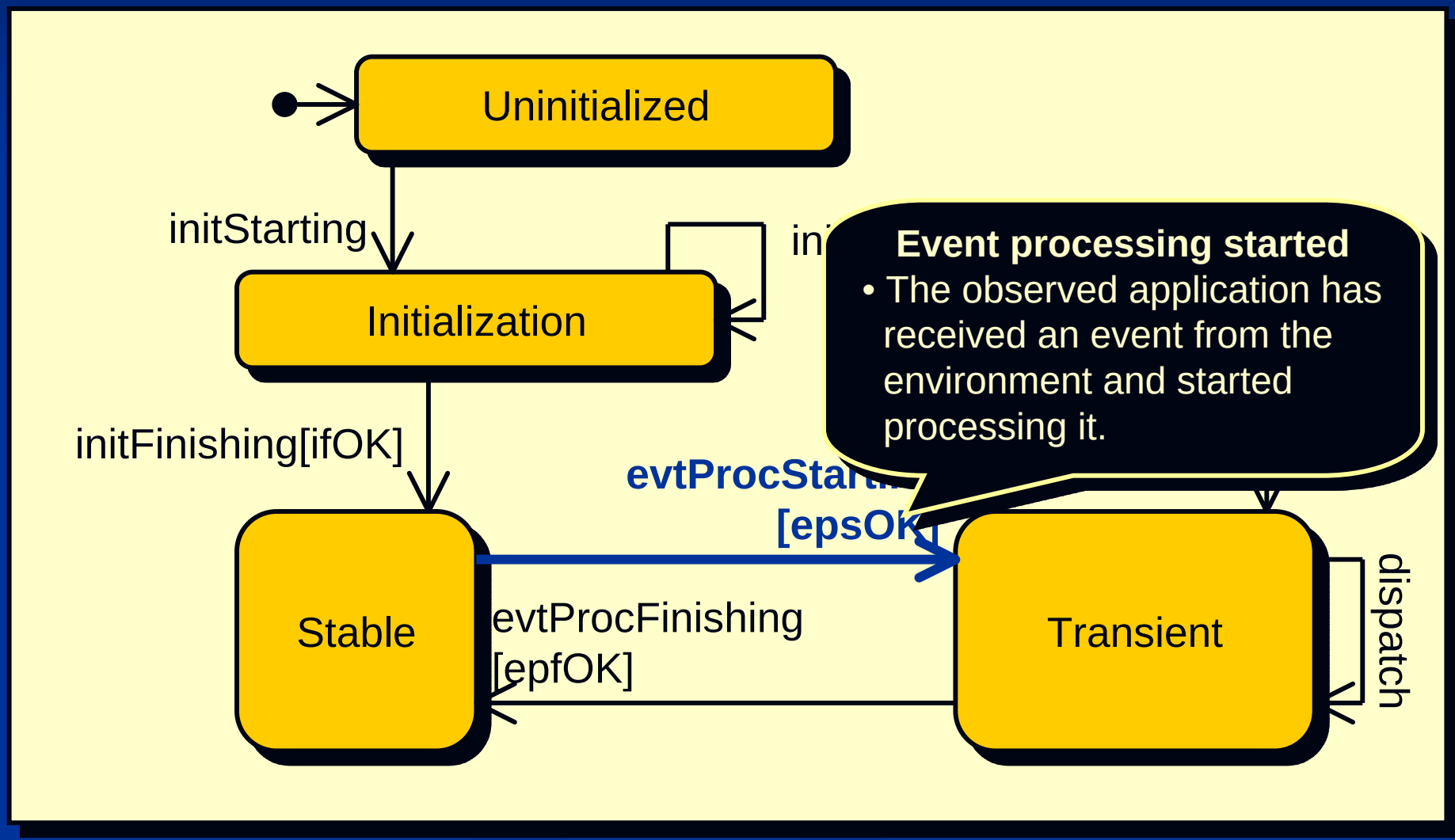
Run-to-completion context



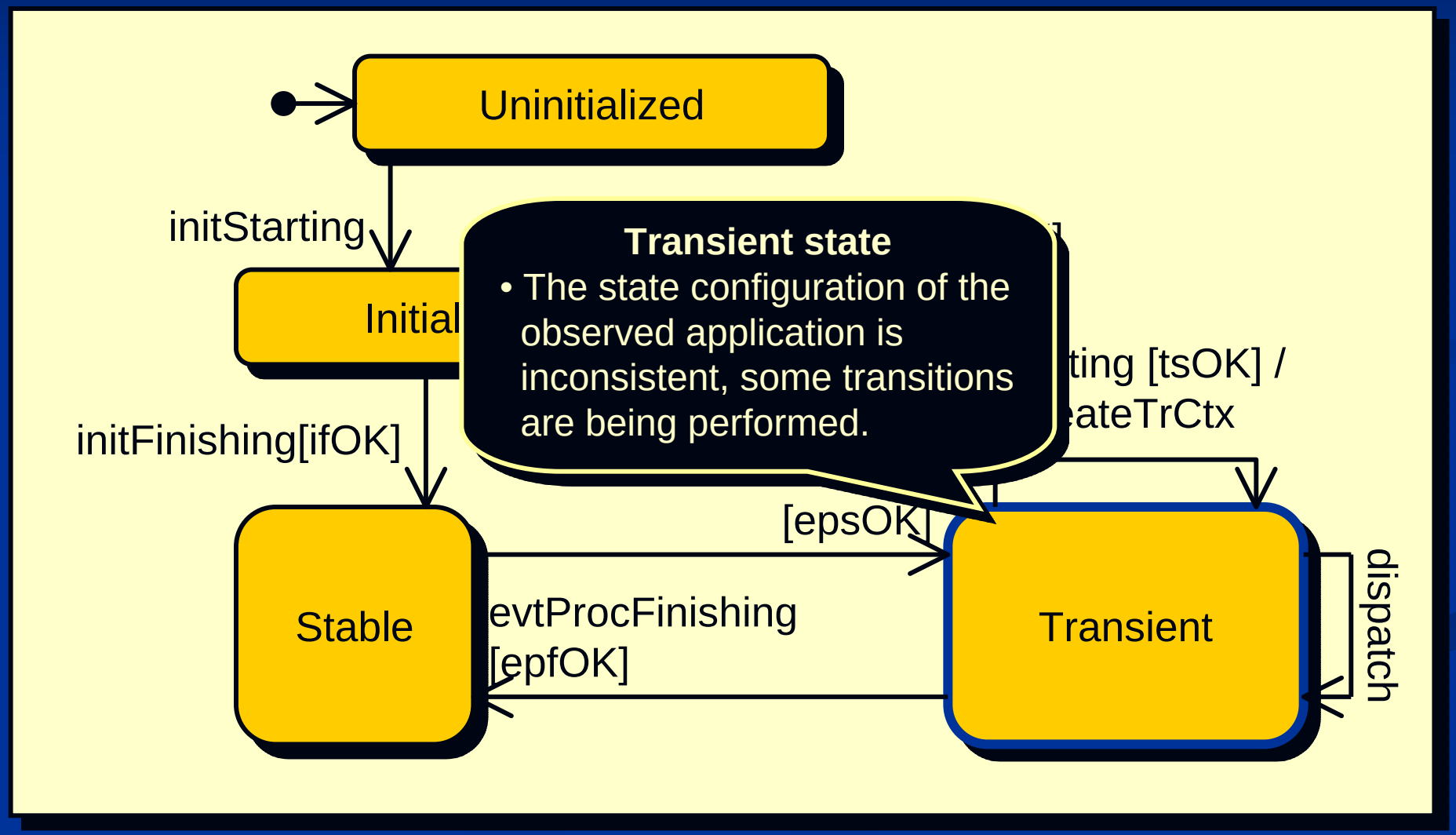
Run-to-completion context



Run-to-completion context



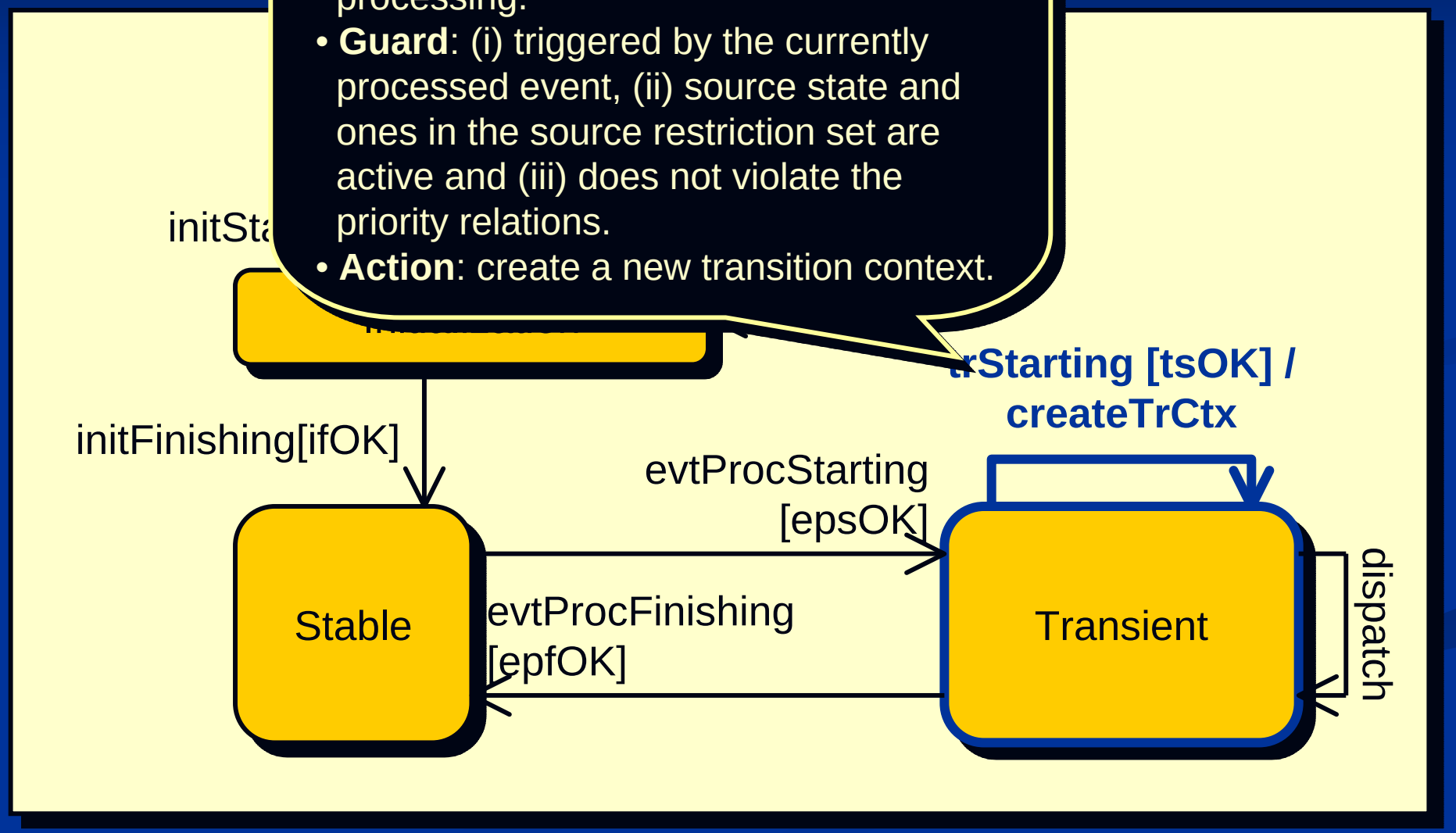
Run-to-completion context



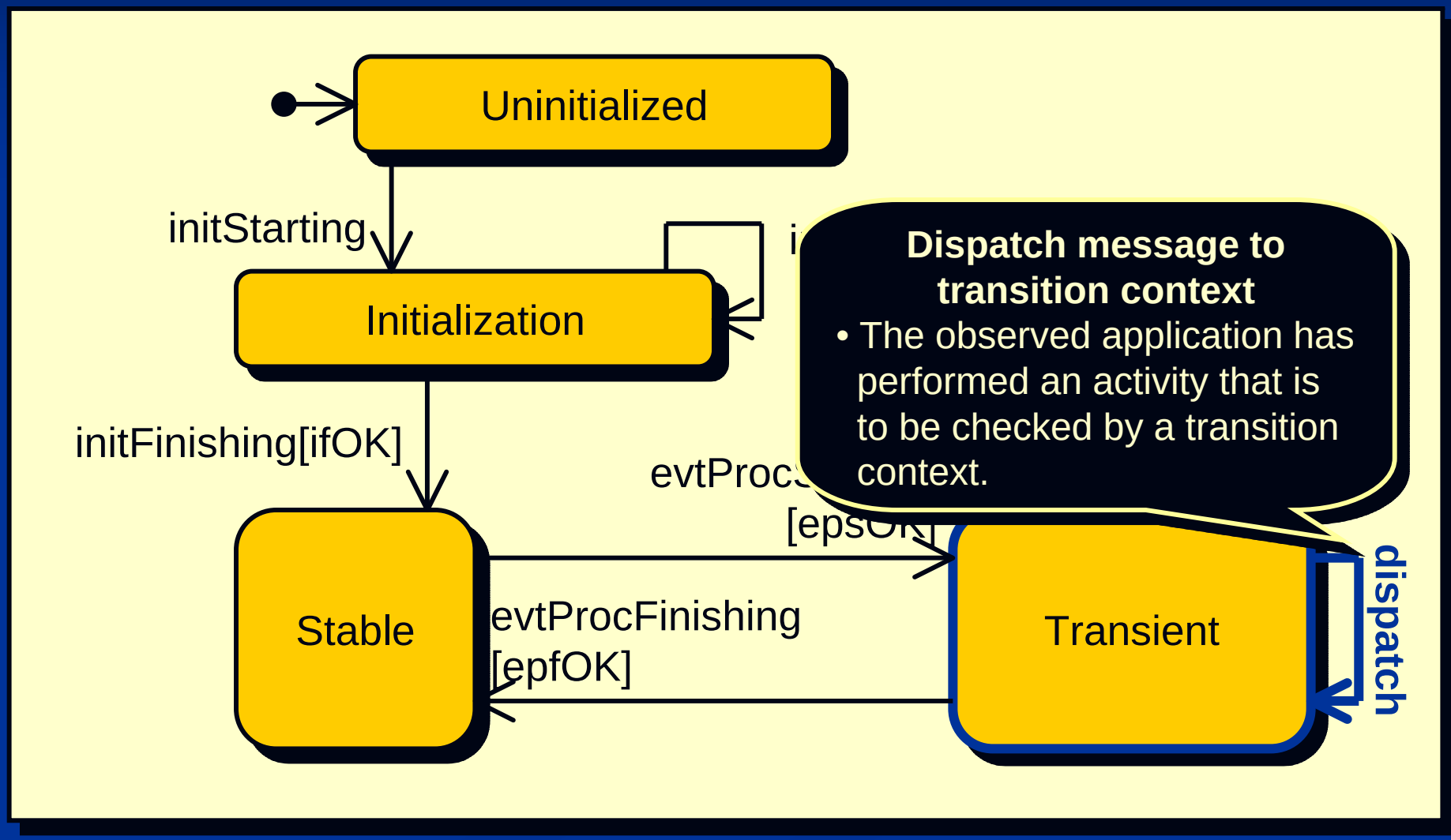
Run ontext

Transition selected for firing

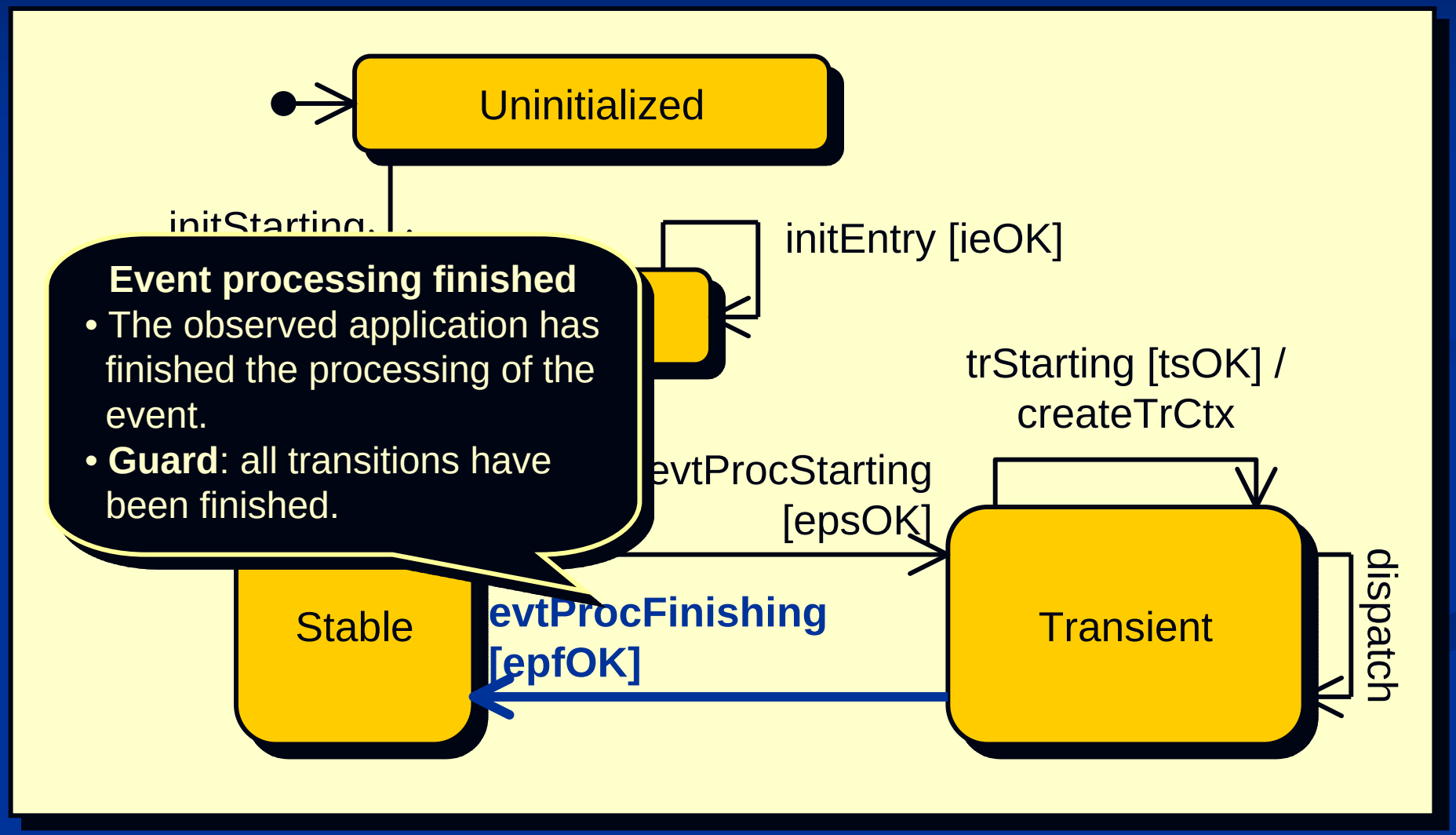
- The observed application has selected a transition to be fired during the event processing.
- **Guard**: (i) triggered by the currently processed event, (ii) source state and ones in the source restriction set are active and (iii) does not violate the priority relations.
- **Action**: create a new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



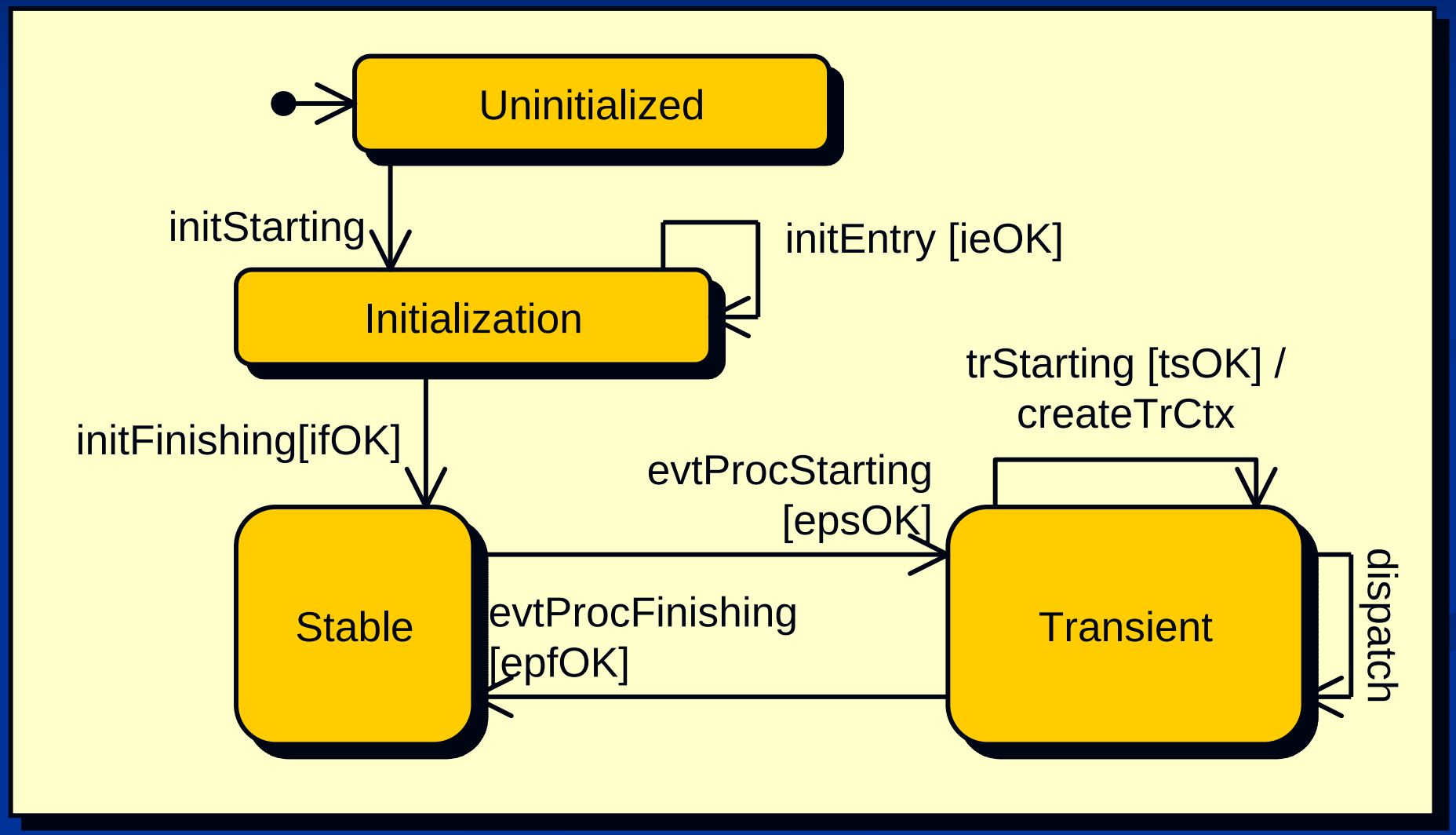
Run-to-completion context



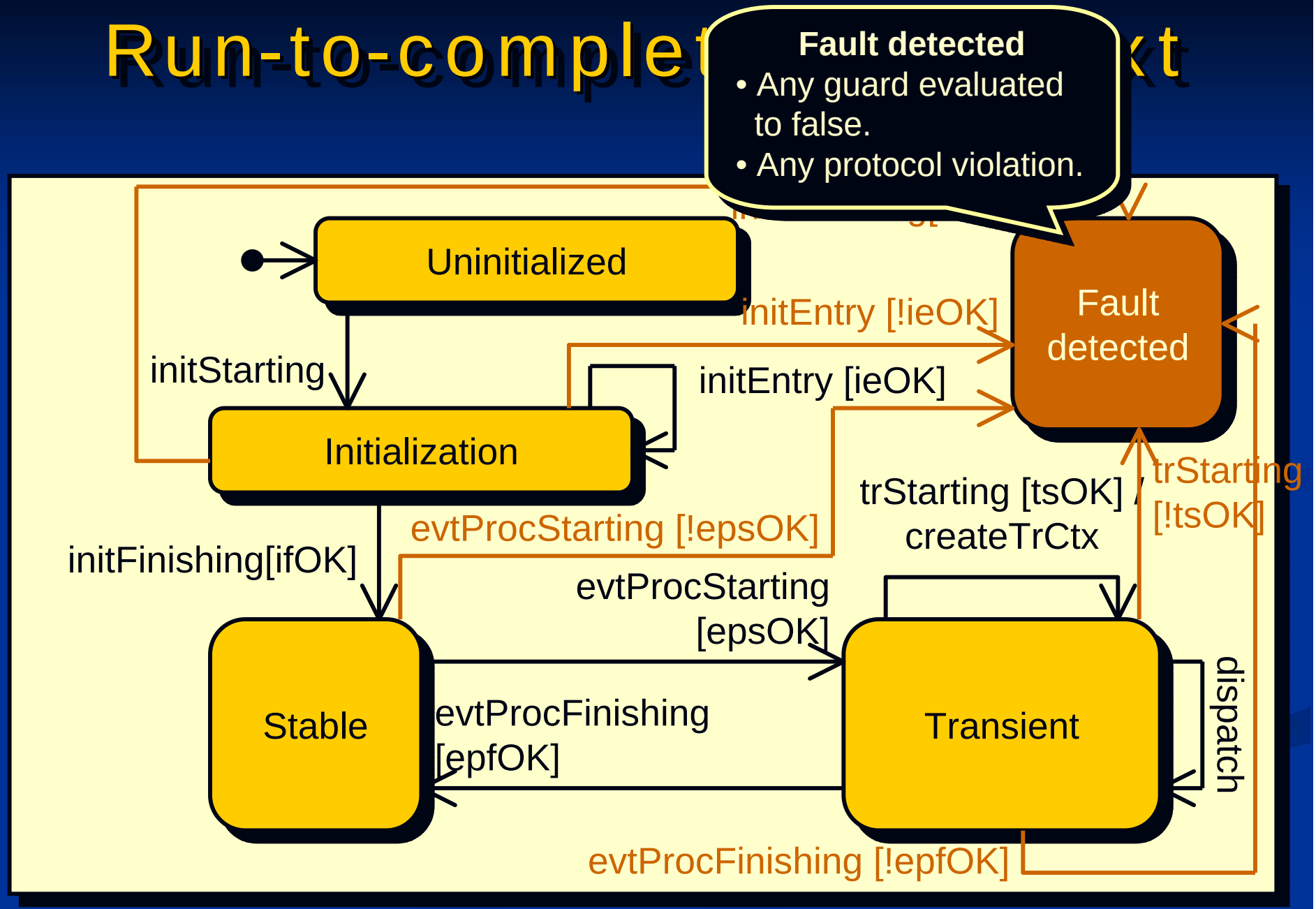
Run-to-completion context



Run-to-completion context

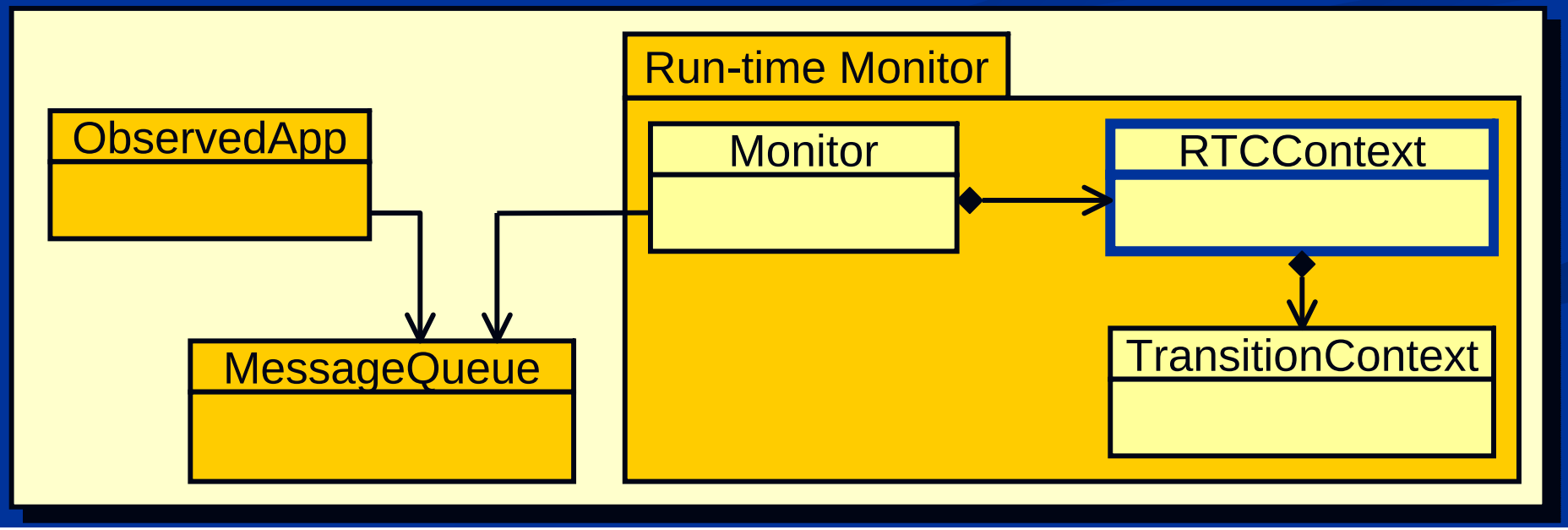


Run-to-complet[illegible]xt



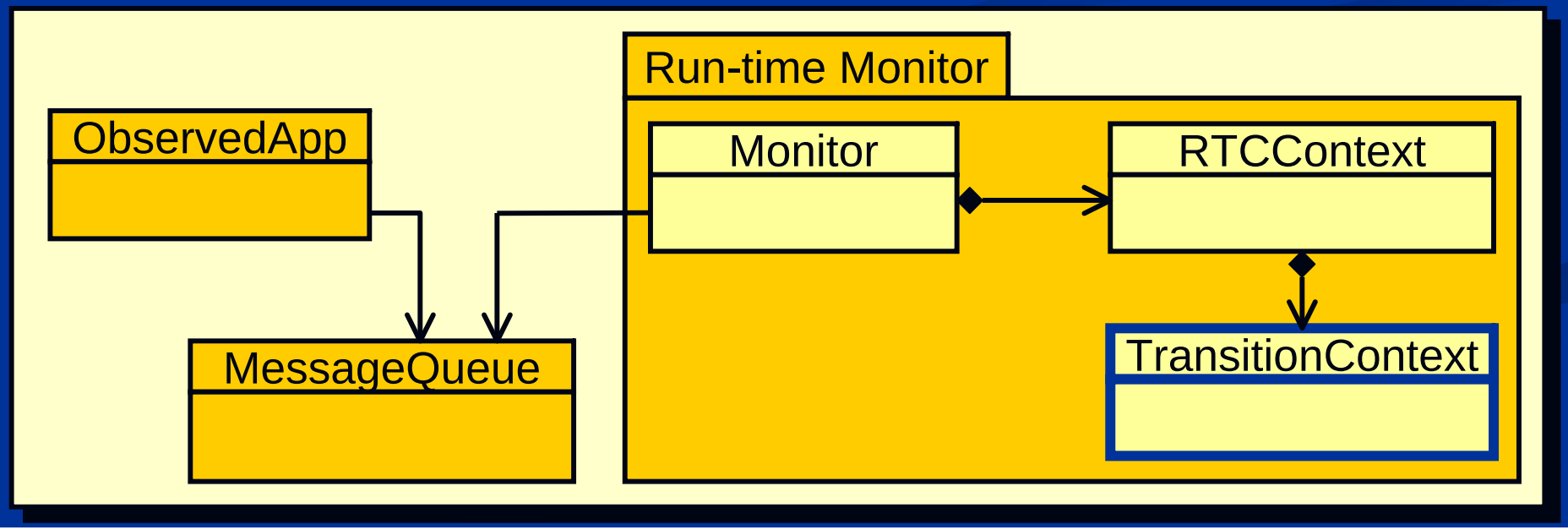
Checking the internal behavior

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

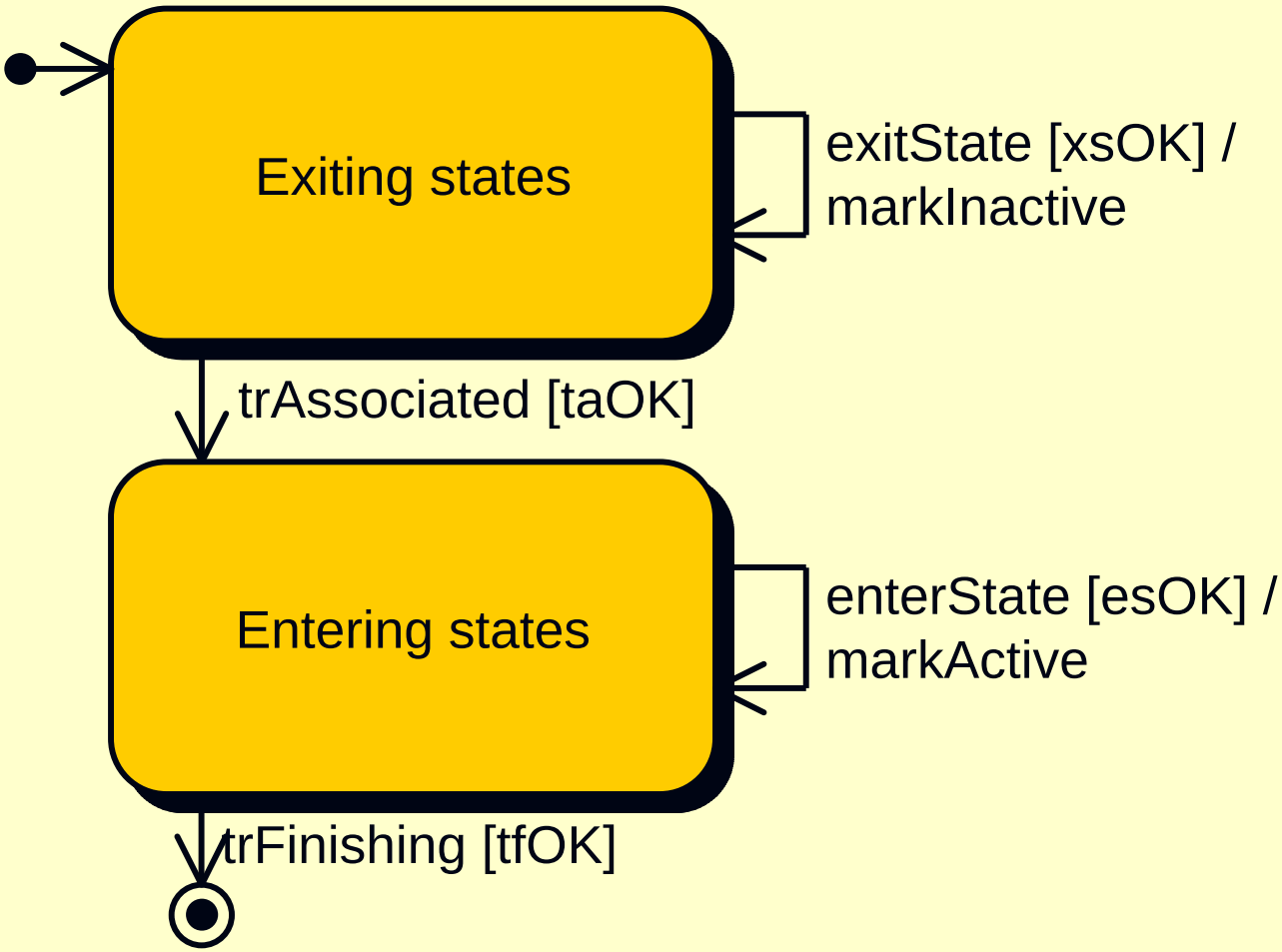


Checking the internal behavior

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



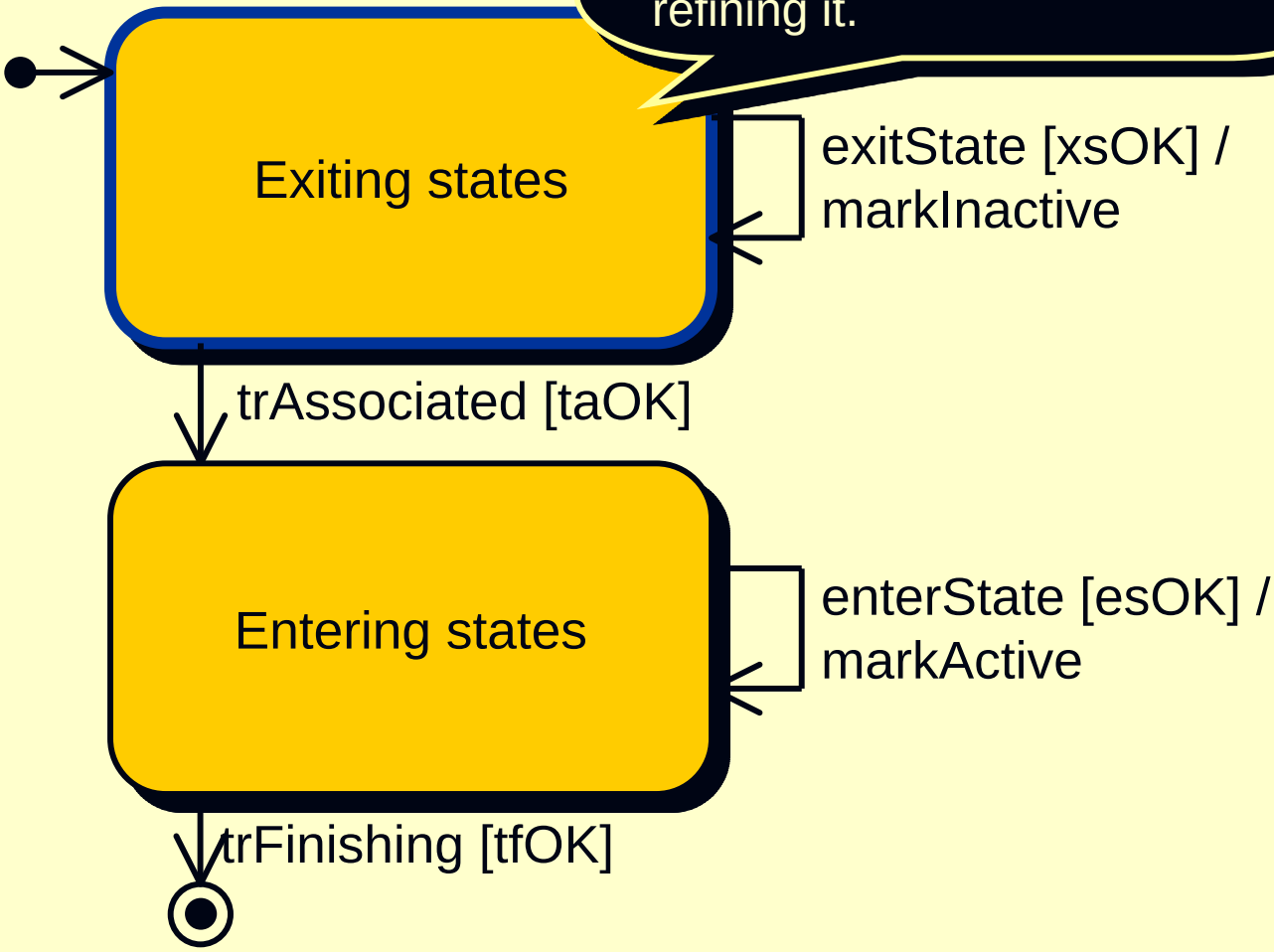
Transition context



Transit

Exiting states

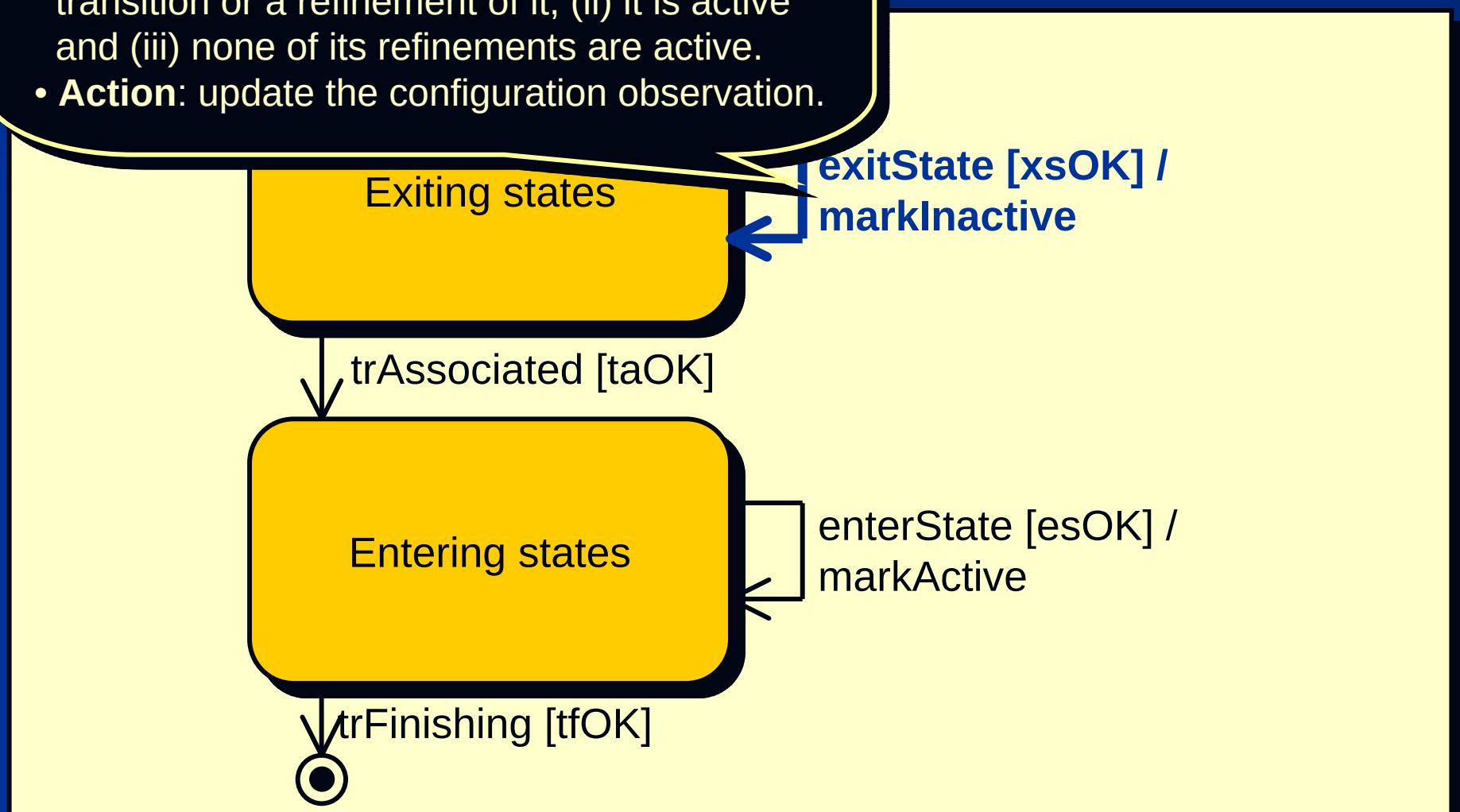
- The observed application is in the first phase of performing a transition: leaving the source state and all active states refining it.



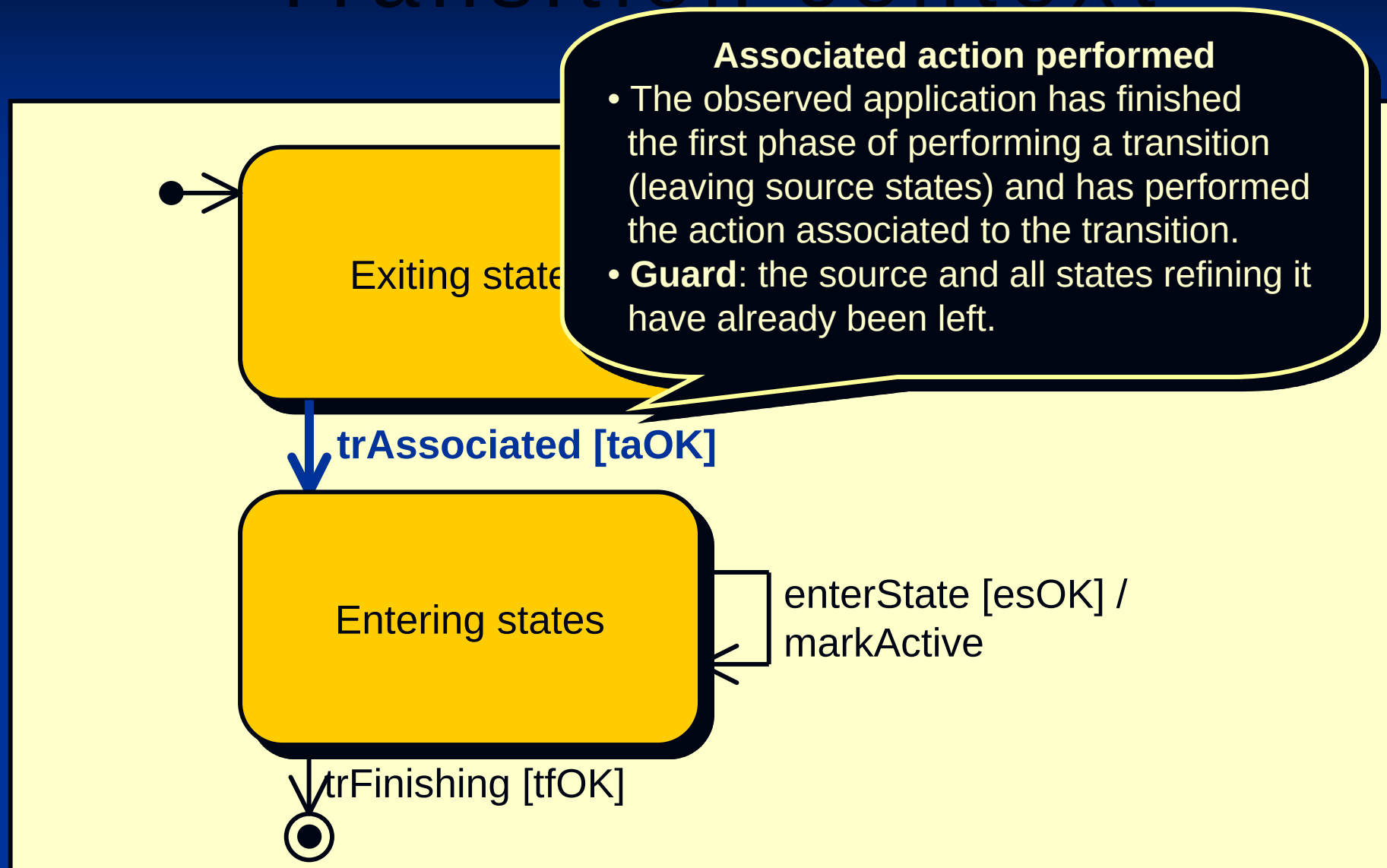
ontext

State left

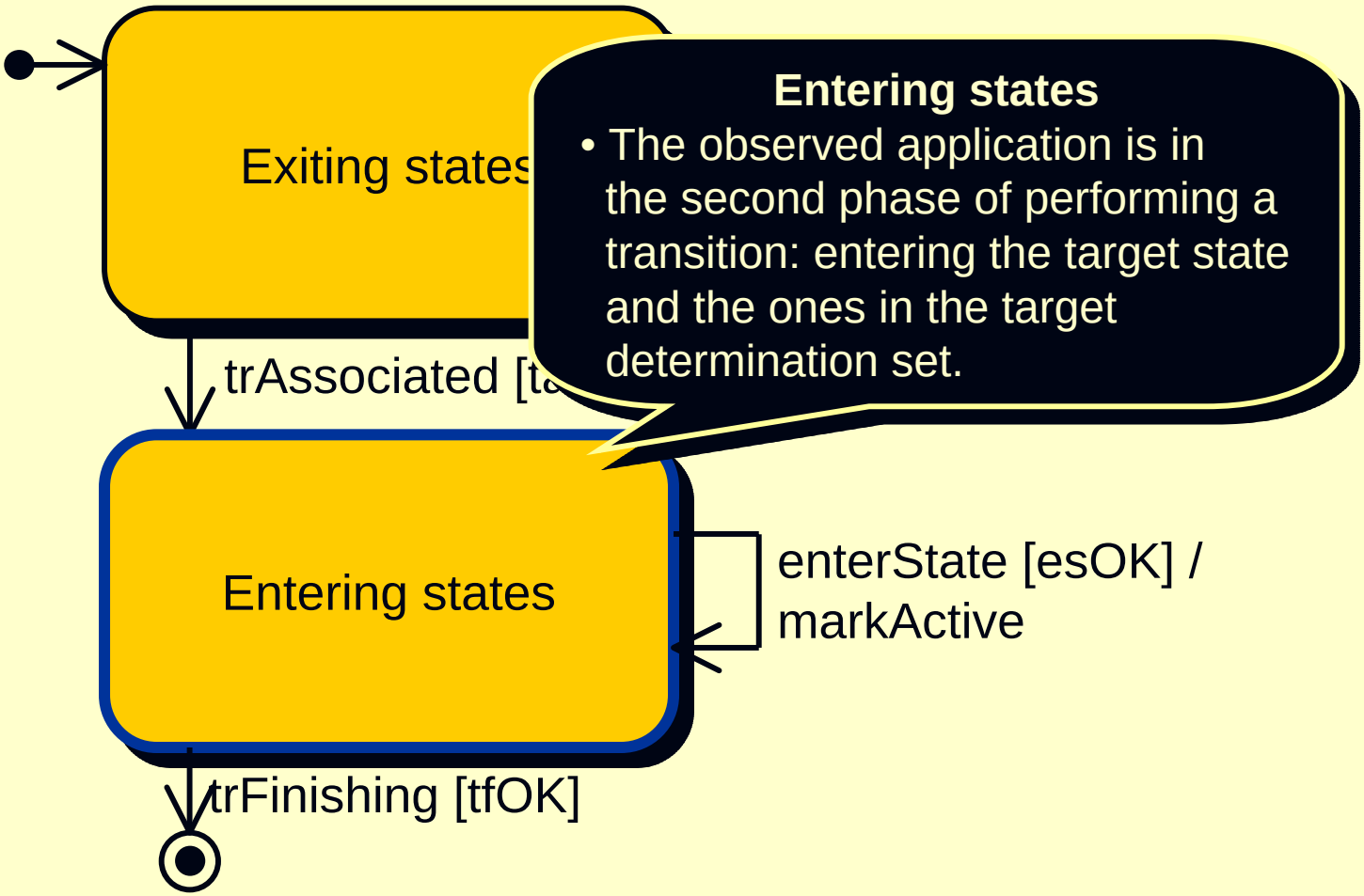
- The observed application has left a state during a transition.
- **Guard**: (i) the state is the source of the transition or a refinement of it, (ii) it is active and (iii) none of its refinements are active.
- **Action**: update the configuration observation.



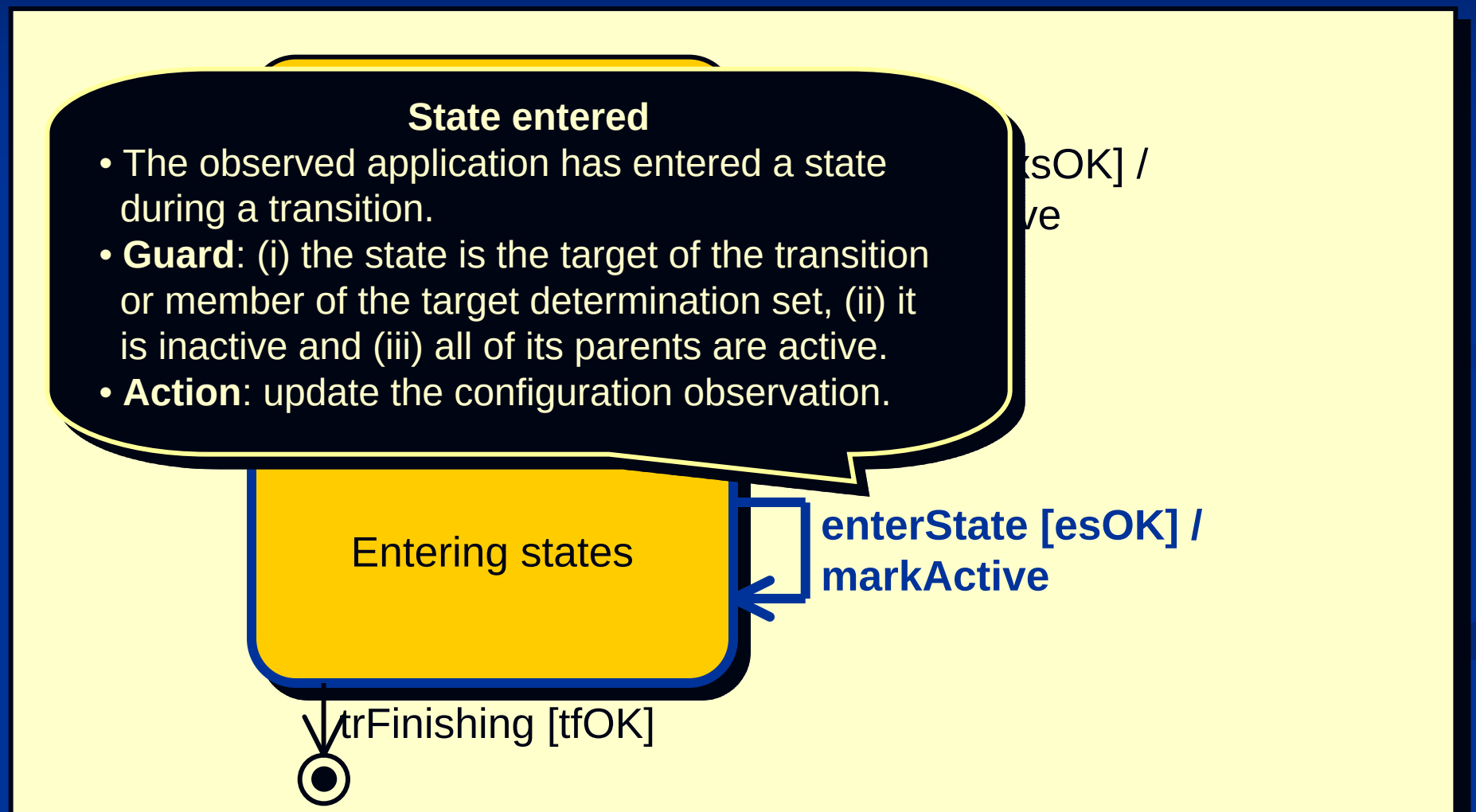
Transition context



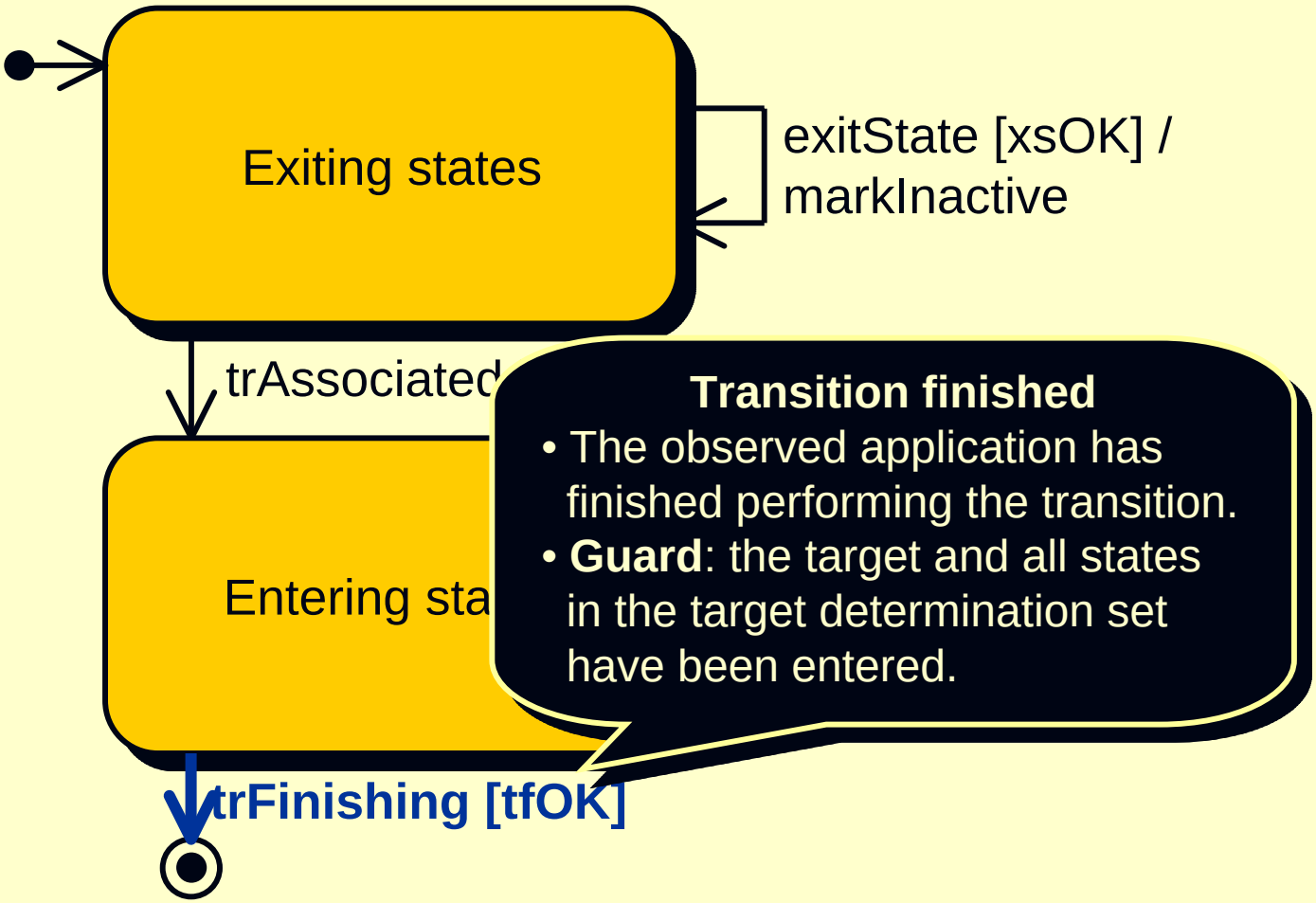
Transition context



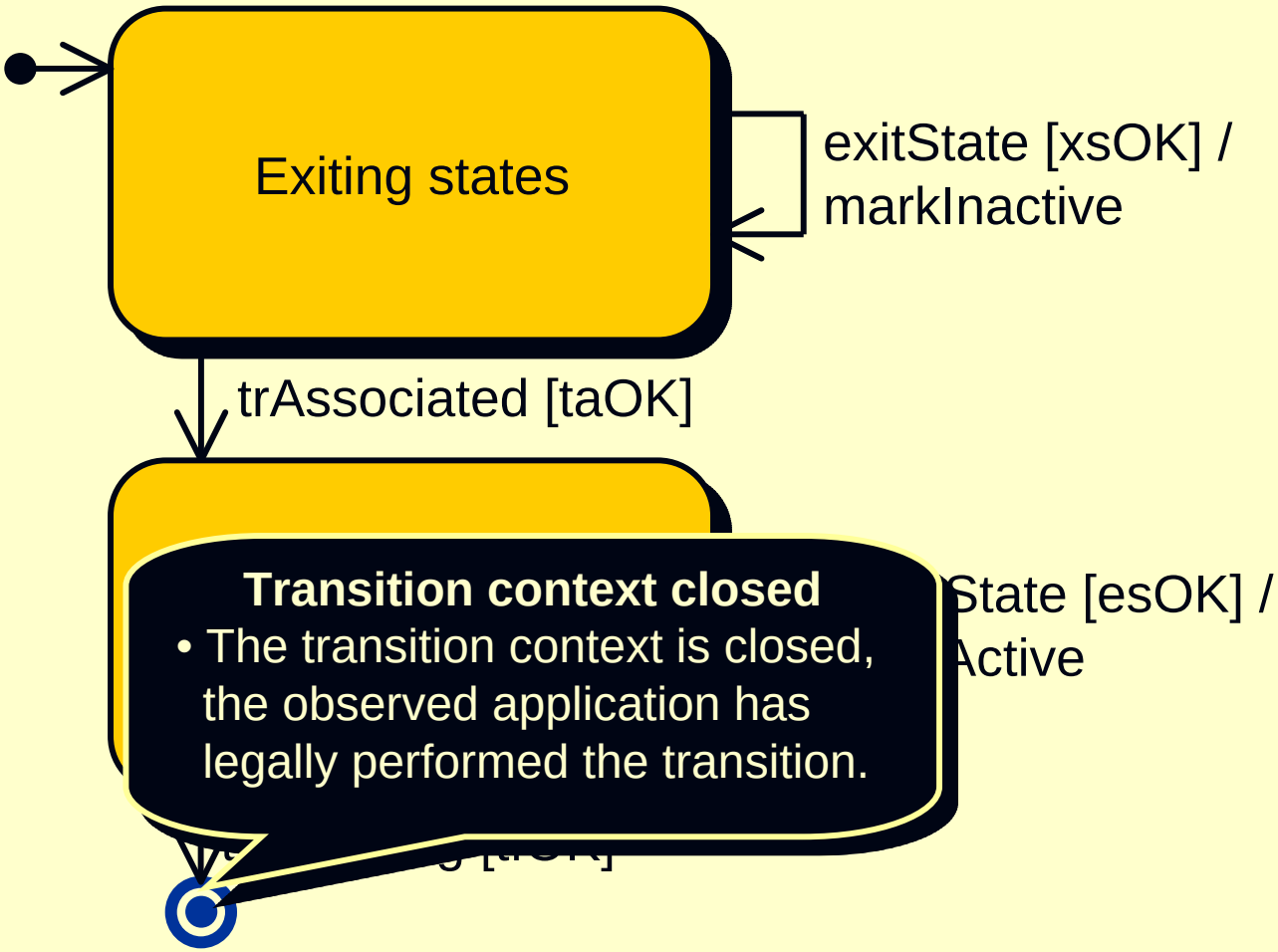
Transition context



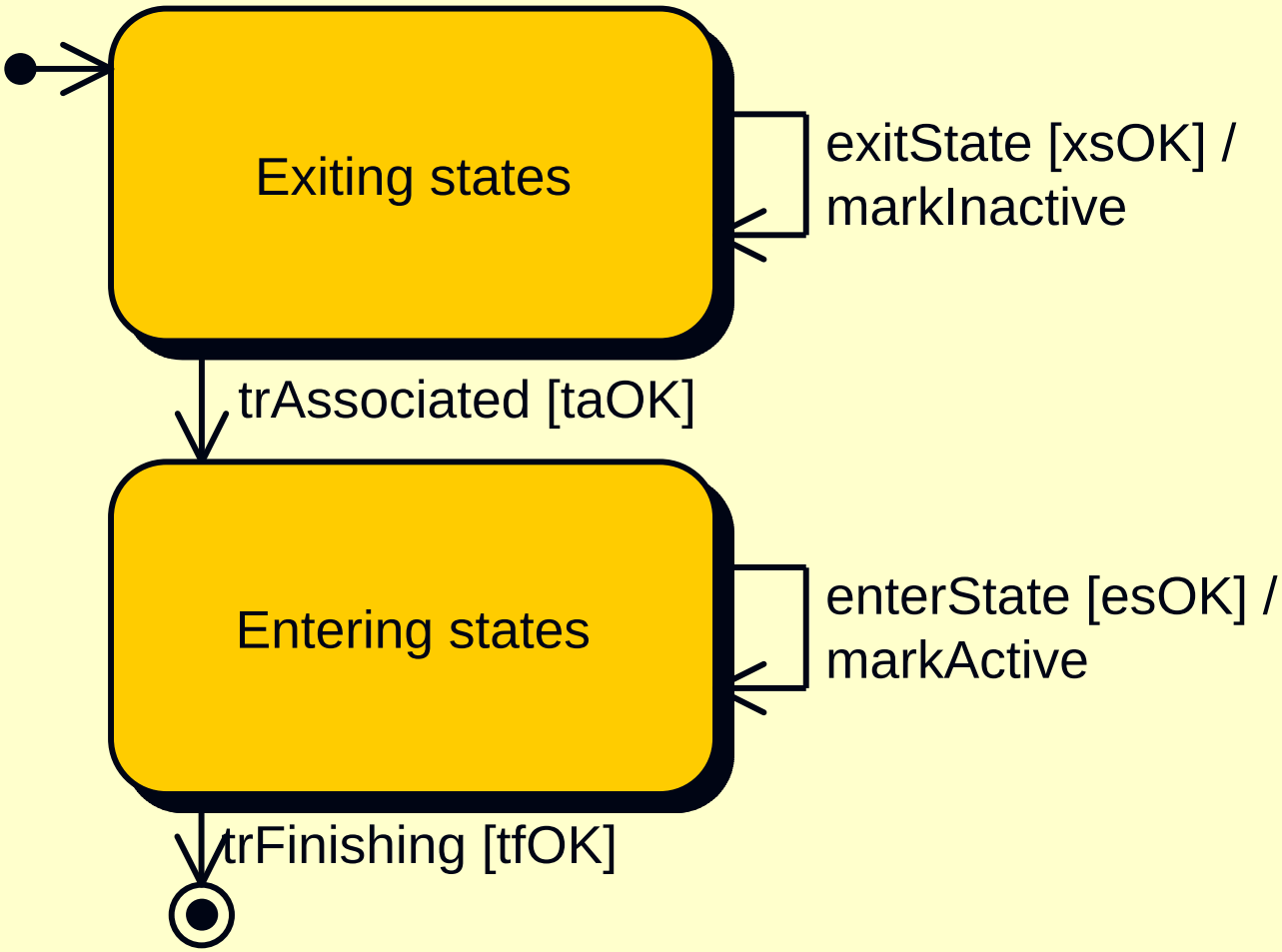
Transition context



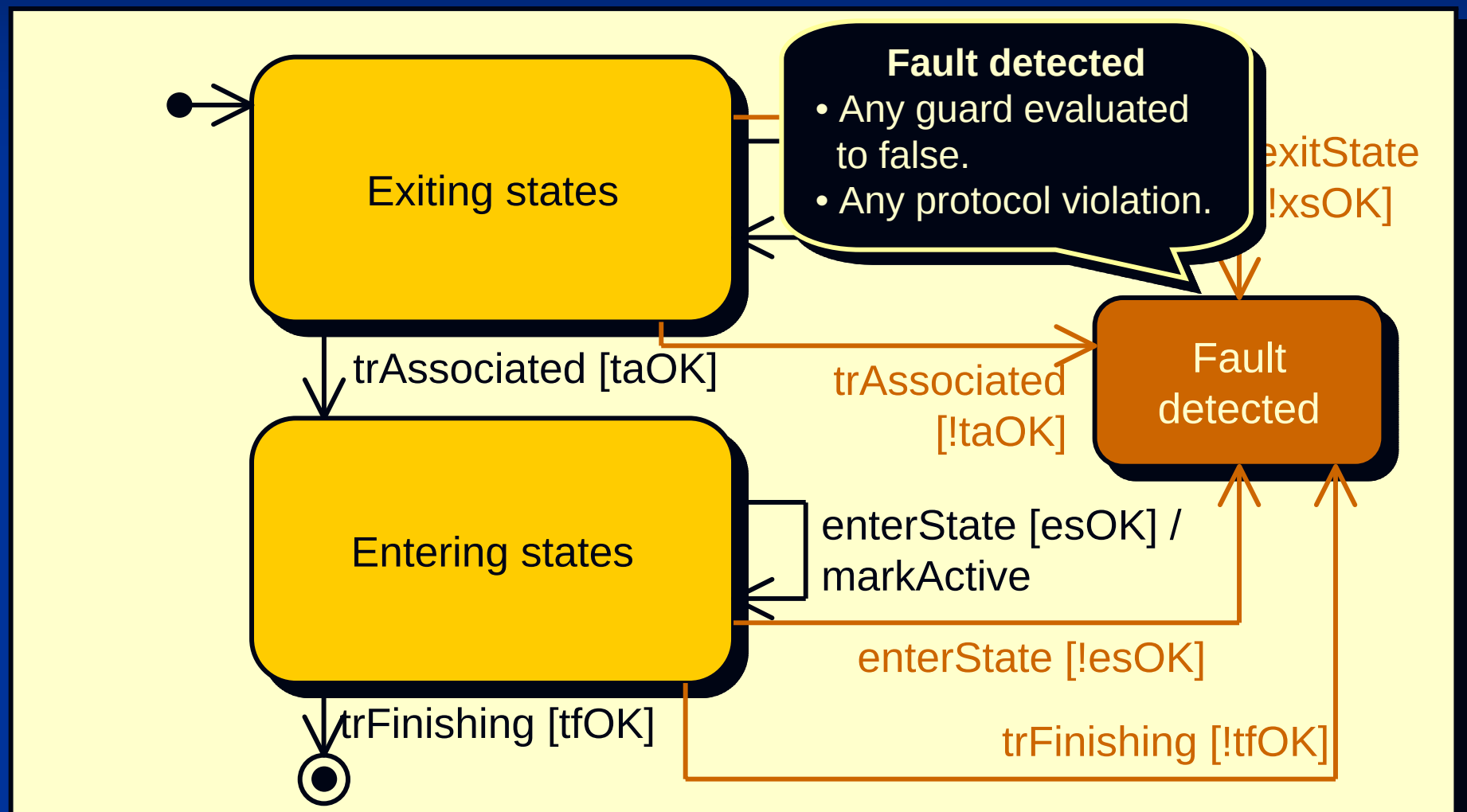
Transition context



Transition context

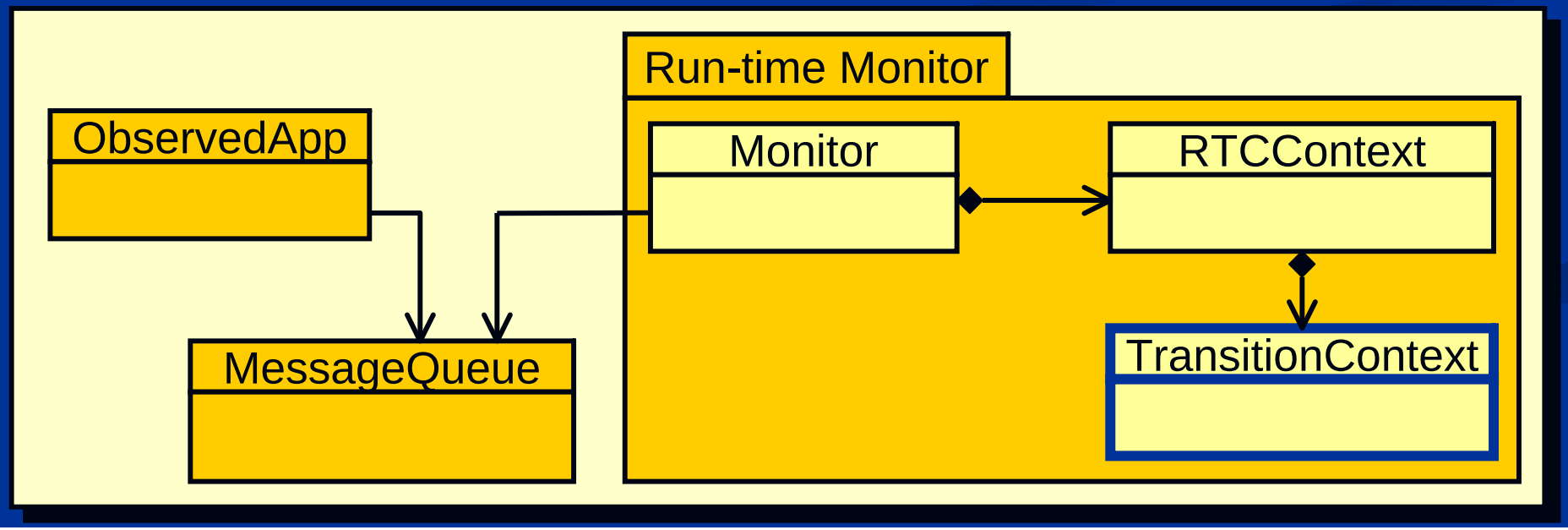


Transition context



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

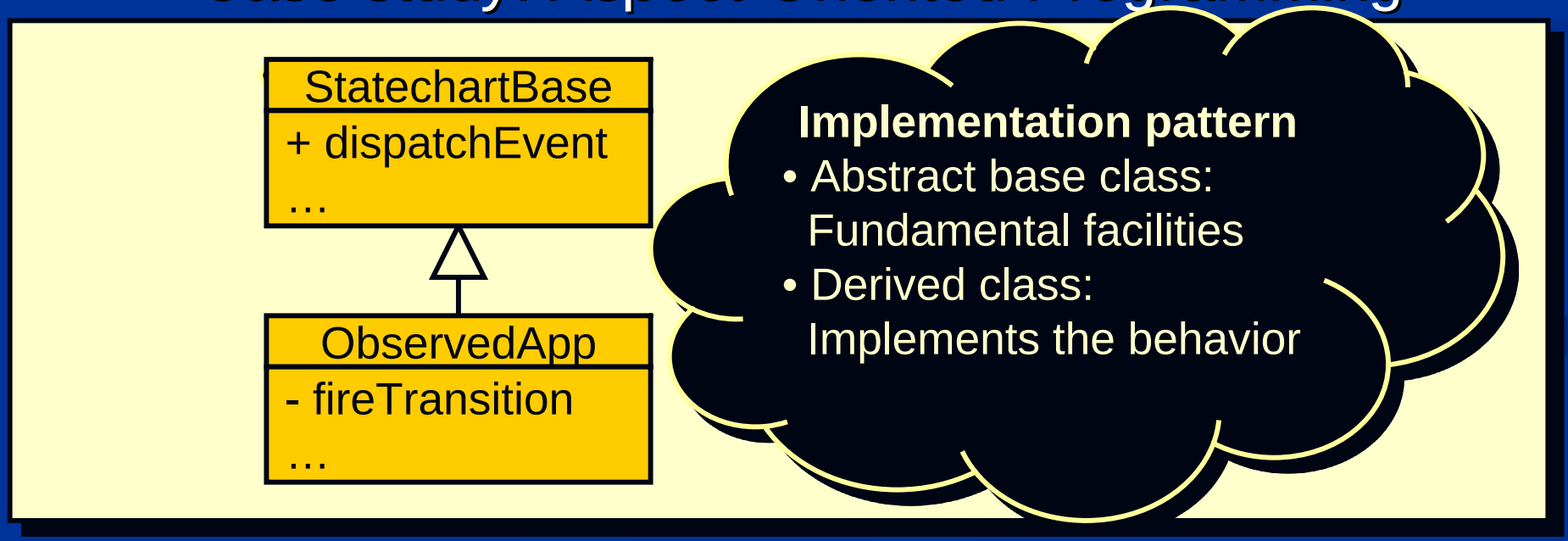
Reference information:
- Automatically derived from the behavioral specification
- Capable of expressing state hierarchies, concurrent operation, etc.

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

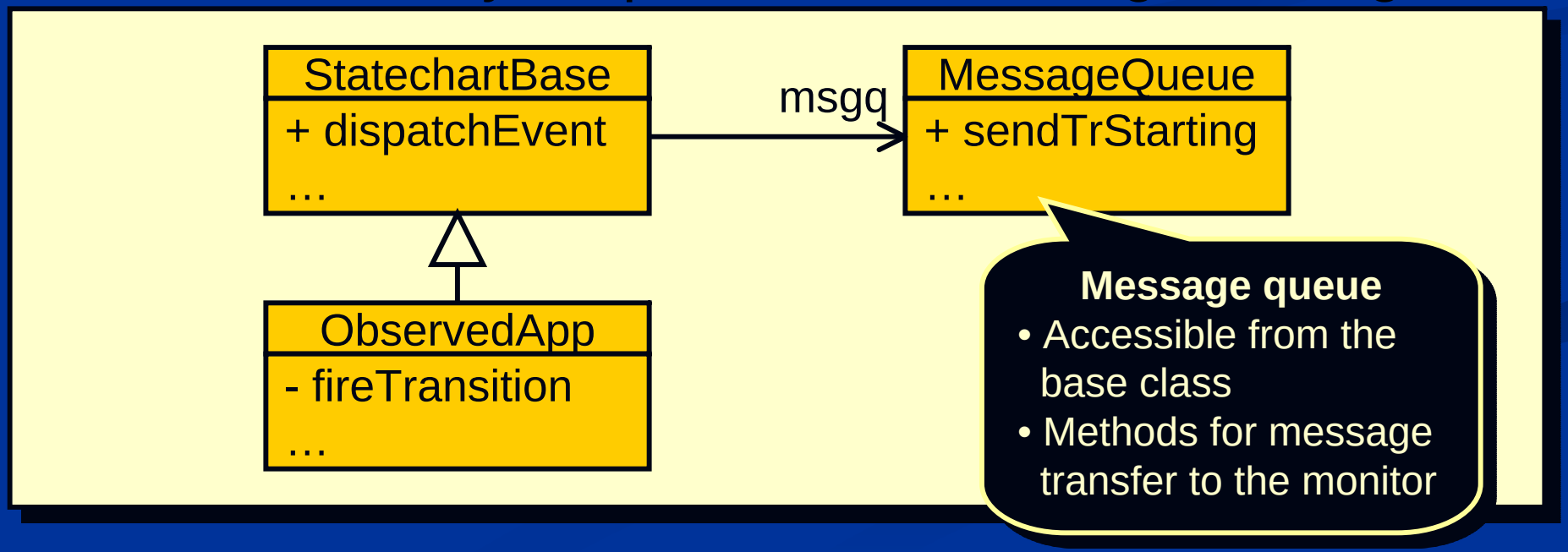
Instrumentation

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



Instrumentation

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

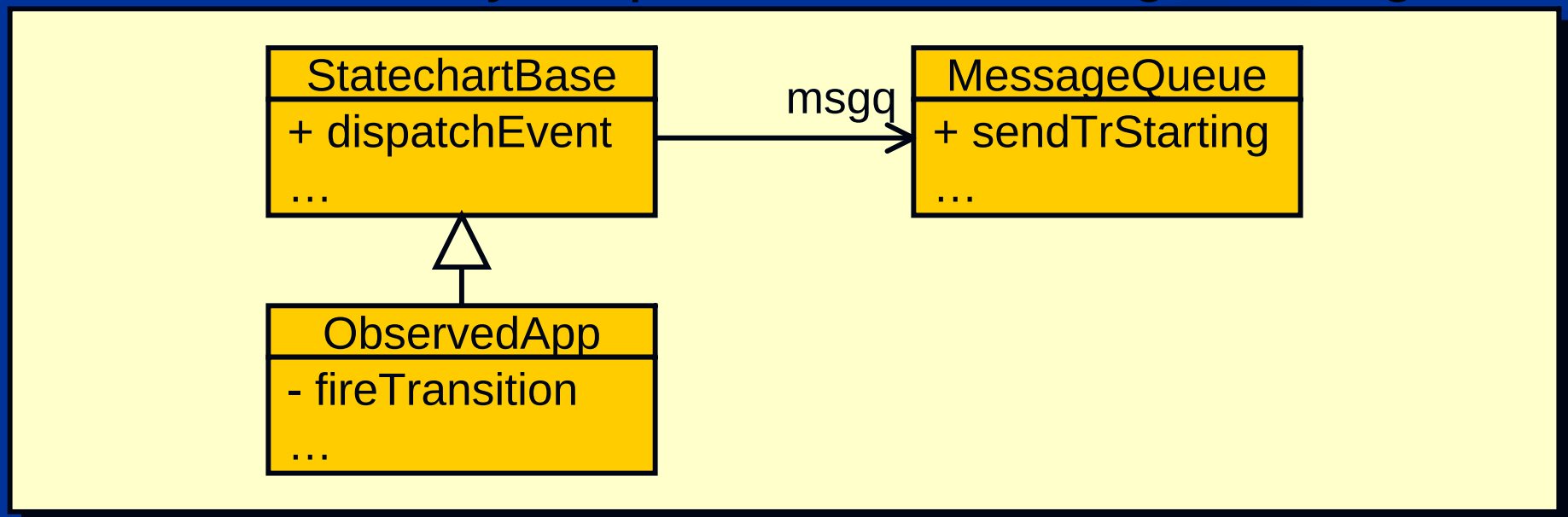


Instrumentation

- Systemat
  - Explicit r
  - Modifica
  - Case study: Aspect Oriented Programming

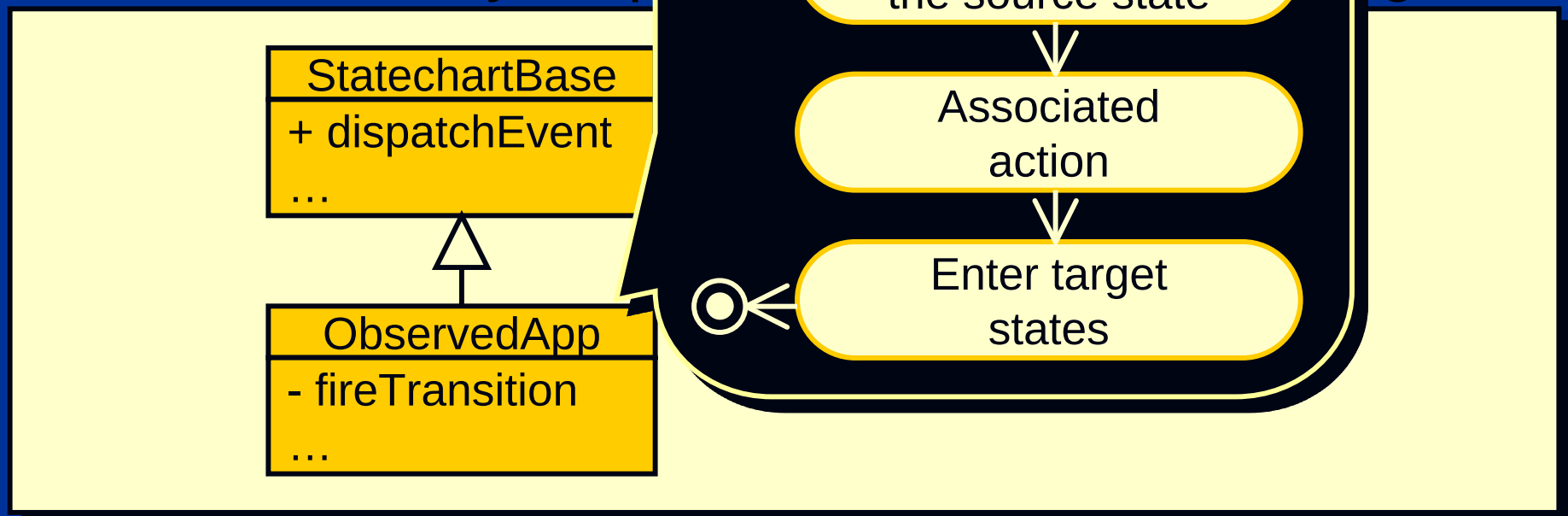
Adding a member variable (Java AOP)

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



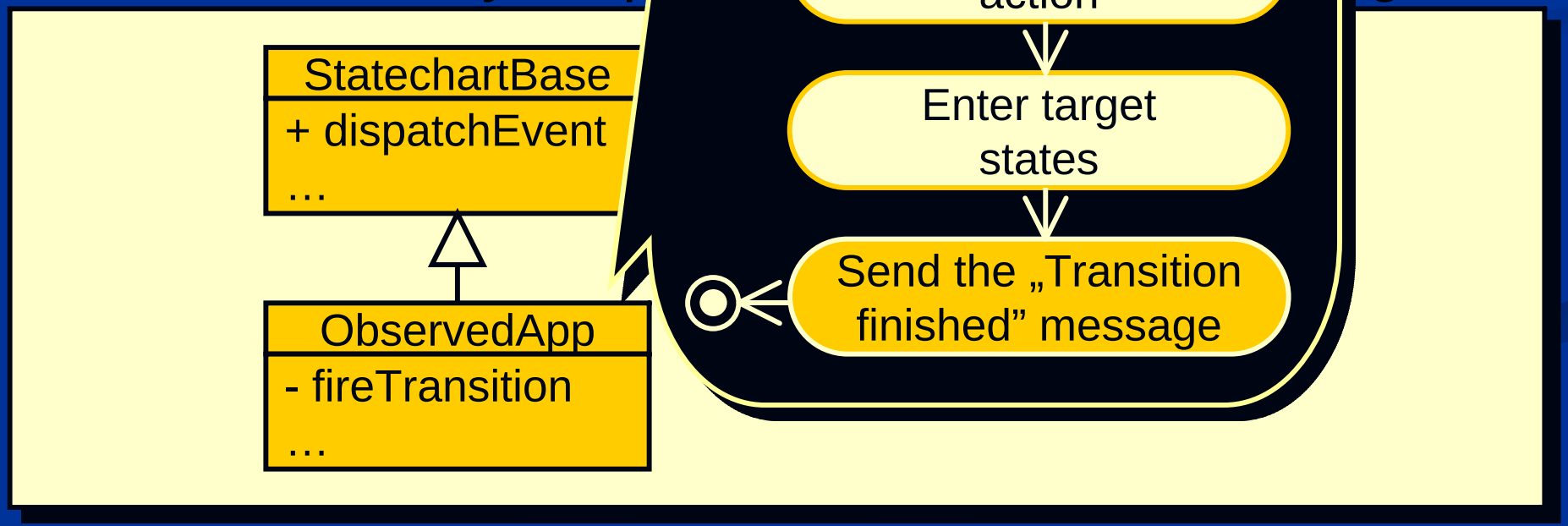
Instrumentation

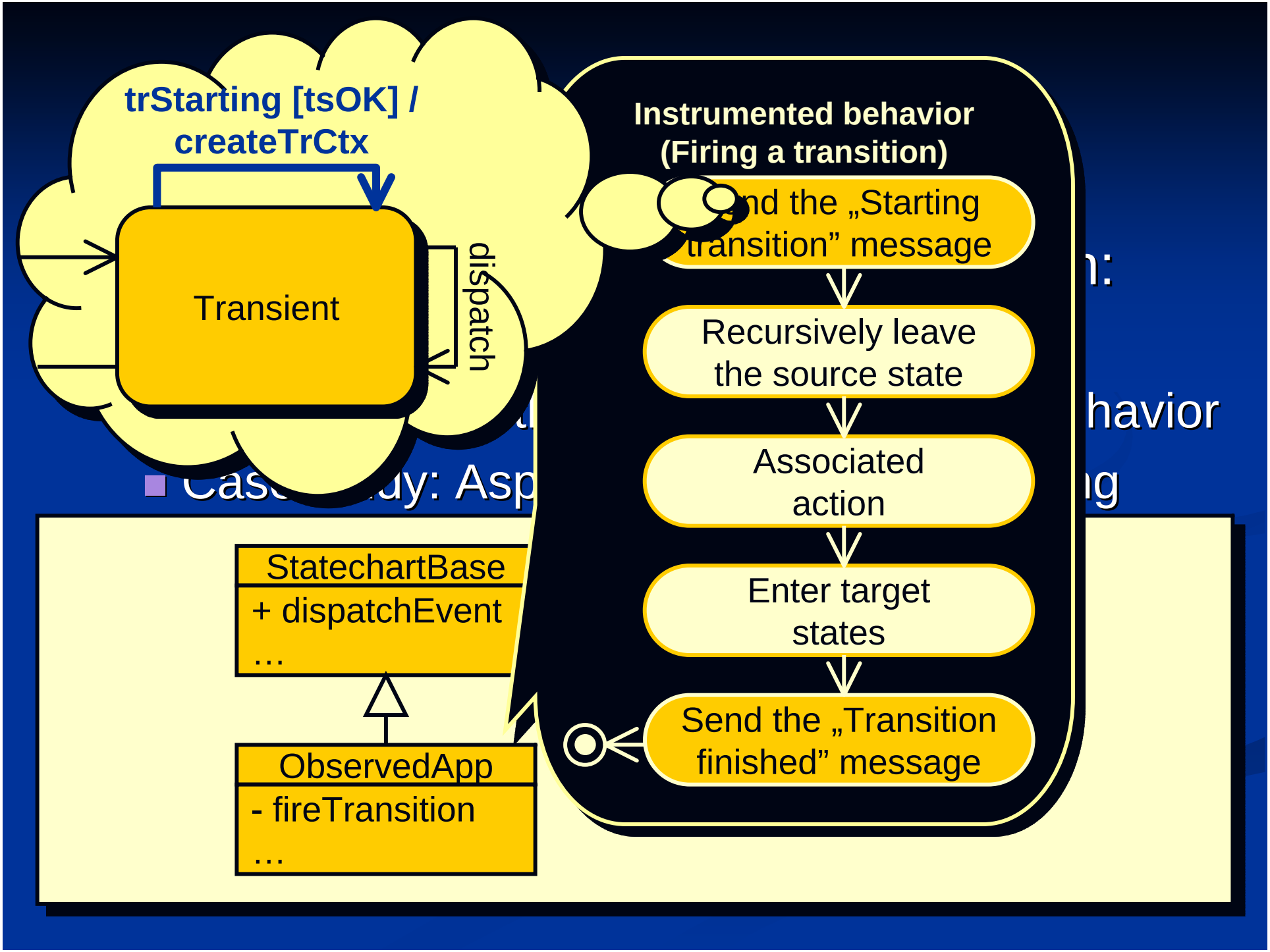
- Systematic transparent instrumentation:
  - Explicit message
  - Modification of t
  - Case study: Asp



Instru

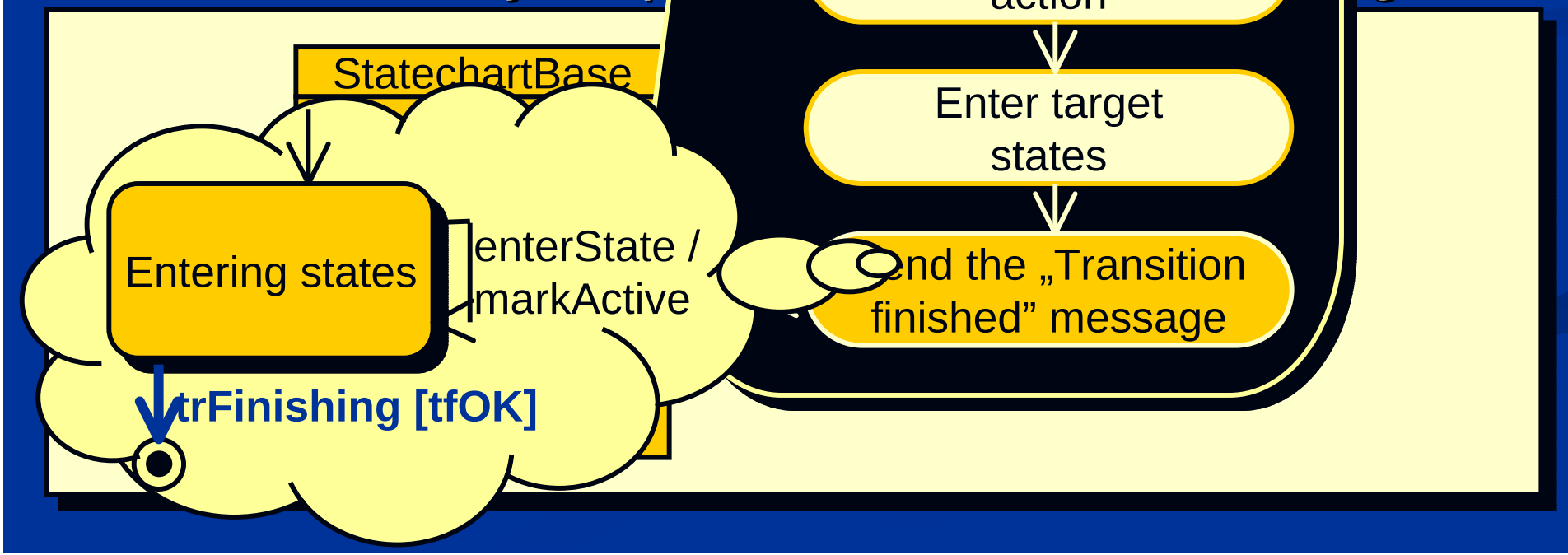
- Systematic trans n:
  - Explicit message
  - Modification of t havior
  - Case study: Asp ng




trStarting [tsOK] /
createTrCtx
Transient
dispatch
Instrumented behavior
(Firing a transition)
nd the „Starting
transition” message
Recursively leave
the source state
Associated
action
Enter target
states
Send the „Transition
finished” message
StatechartBase
+ dispatchEvent
…
ObservedApp
- fireTransition
…

Instru

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



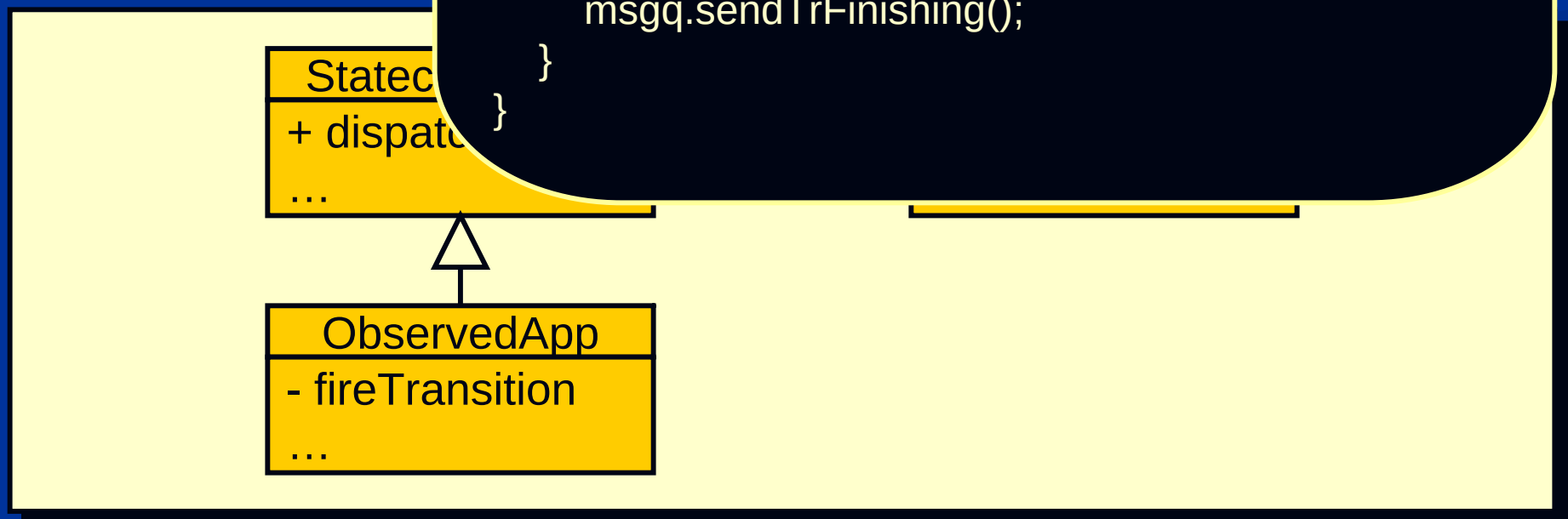
In

- 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.

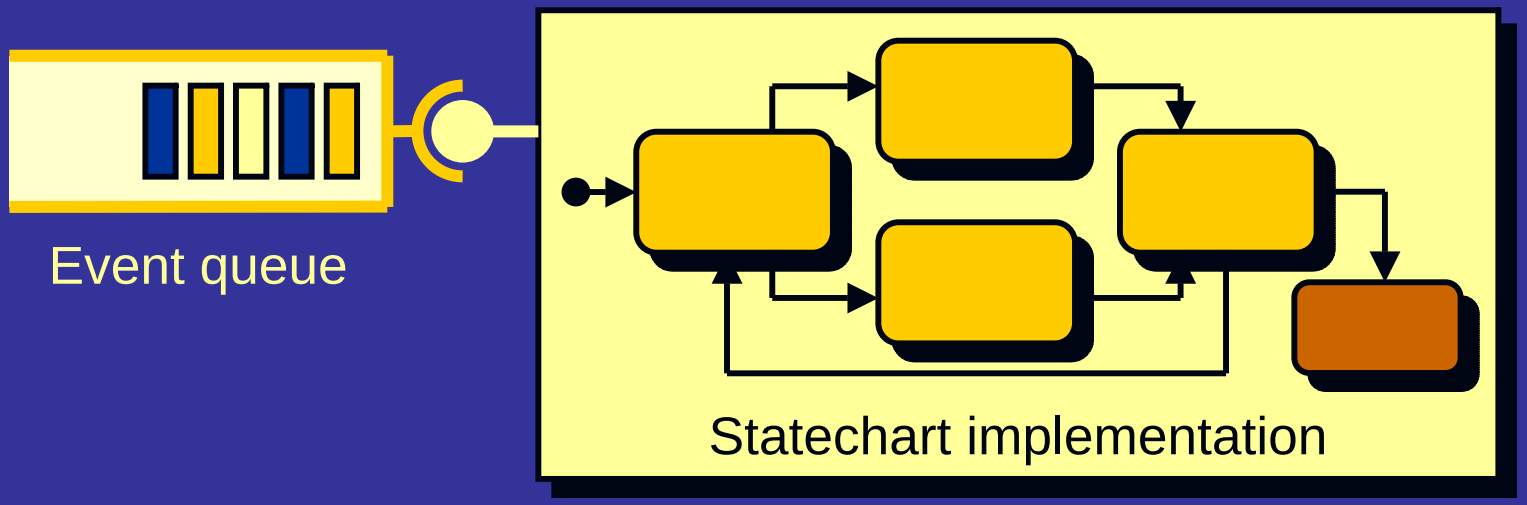
Implementation of the monitor:
- Based on the operational semantics of the behavioral model
- Run-time checking of the behavior on the basis of the abstract reference model

Implementation of the instrumentation:
- Providing information to the monitor about the internal behavior
- Configurable, transparent and automatically applied

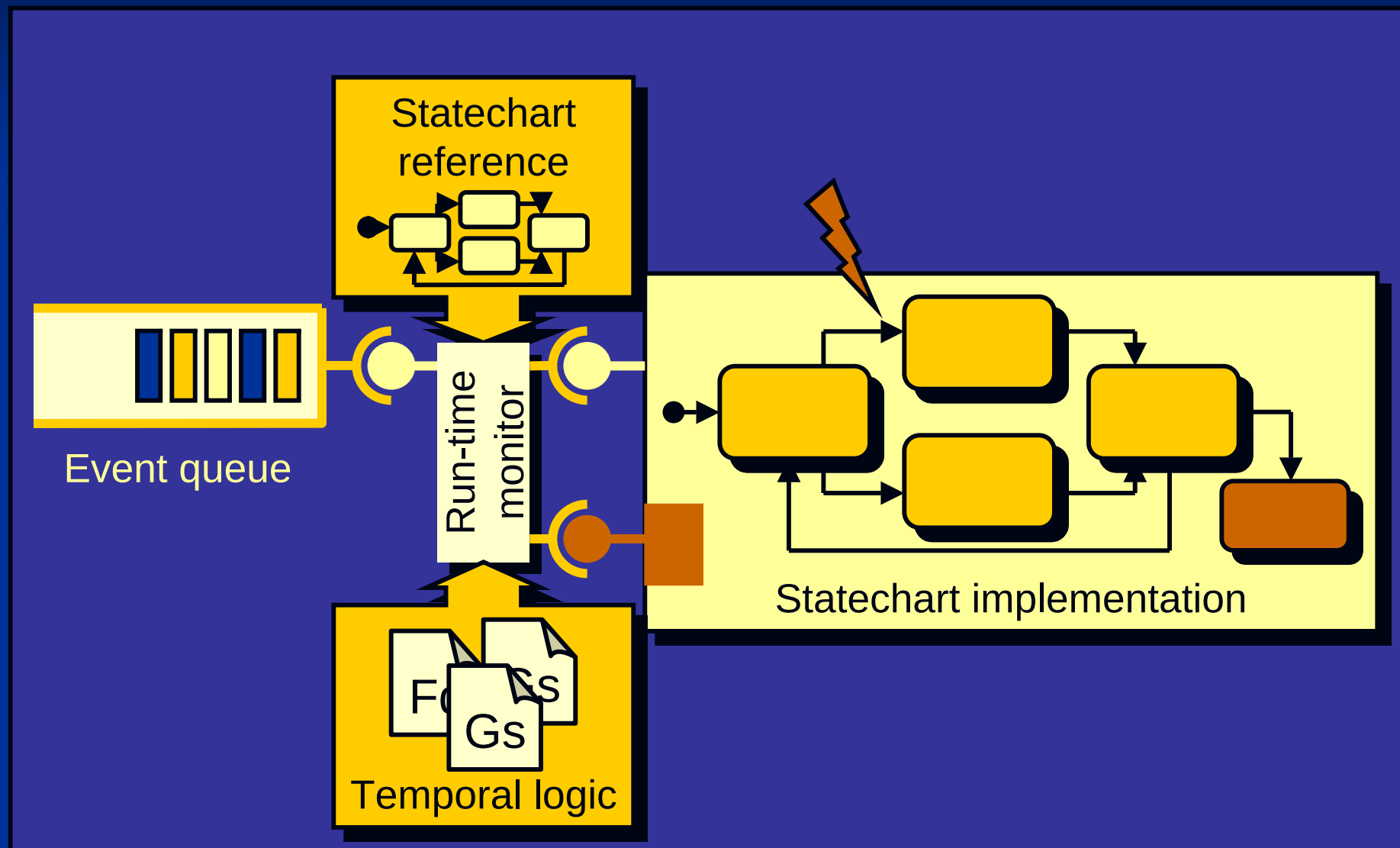
Summary

- 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%

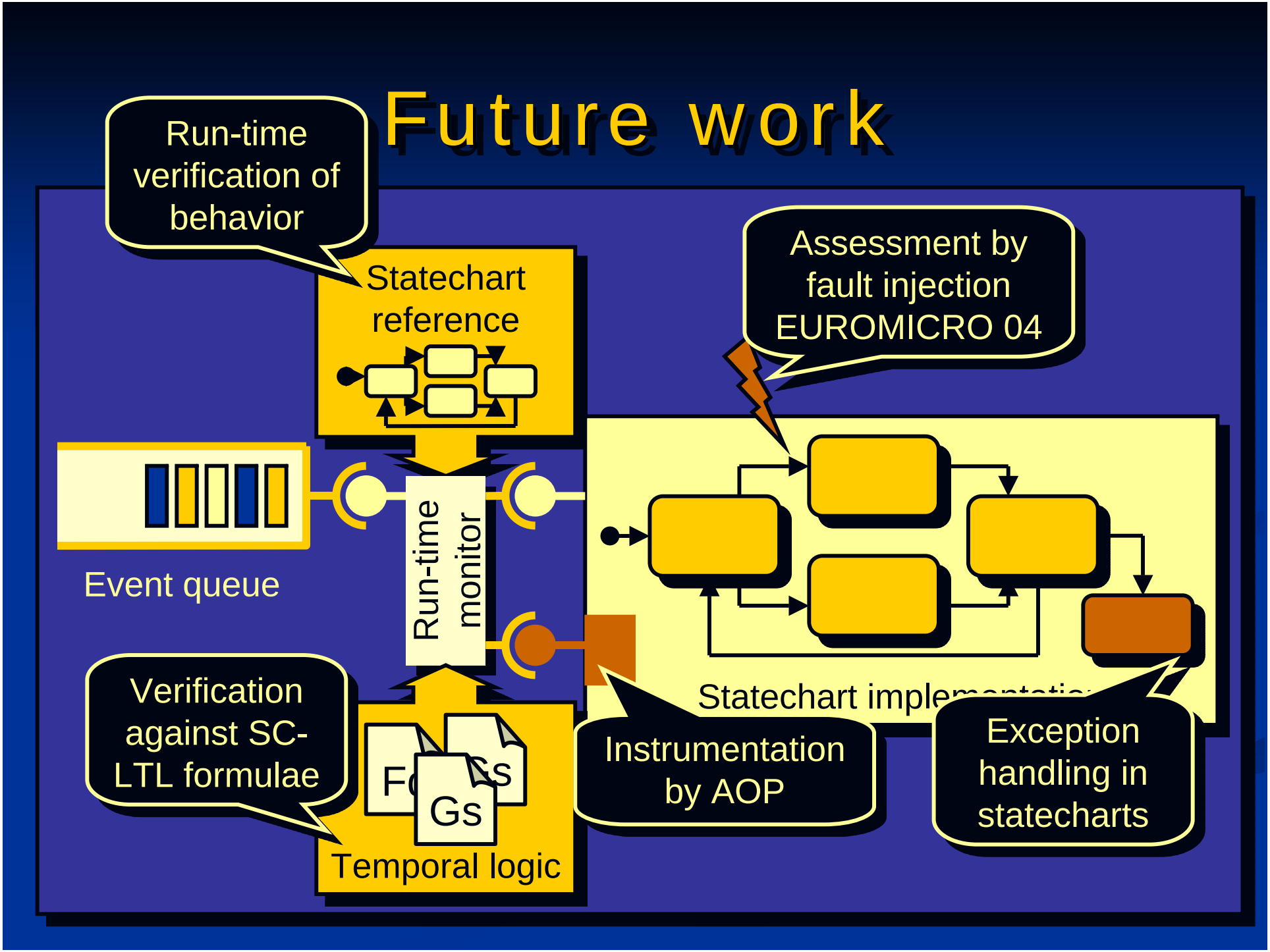
Future work



Future work


Statechart reference
Event queue
Run-time monitor
Statechart implementation
Gs
Temporal logic

Future work


Run-time verification of behavior
Statechart reference
Assessment by fault injection EUROMICRO 04
Event queue
Run-time monitor
Verification against SC-LTL formulae
Gs
Temporal logic
Instrumentation by AOP
Exception handling in statecharts

Futu

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