Activities as Composite Structure:
(Onto) Logical Activity Modeling

Conrad Bock
U.S. National Institute of Standards and Technology

Raphael Barbau
Engisis

engineering laboratory

National Institute of Standards and Technology

Overview

- RoadMap
- Motivation
  - Behavior, review
  - Activities, requirements
- Activities Solution
  1. Control nodes
  2. Loops
  3. Specialization
- Summary

Behavior as Composite Structure Presentation Stack

Onto Activities
(this one)

Onto State Machines, Parts 1 & 2
(ad/18-12-09, 19-03-02)

Onto Interactions
(ad/18-06-11)

Onto OO
(ad/18-09-07)

Onto Behavior Basics
(ad/2018-03-02)

Overview

- RoadMap
- **Motivation**
  - **Behavior, review**
  - Activities, requirements
- Activities Solution
  - Control nodes
  - Loops
  - Specialization
- Summary

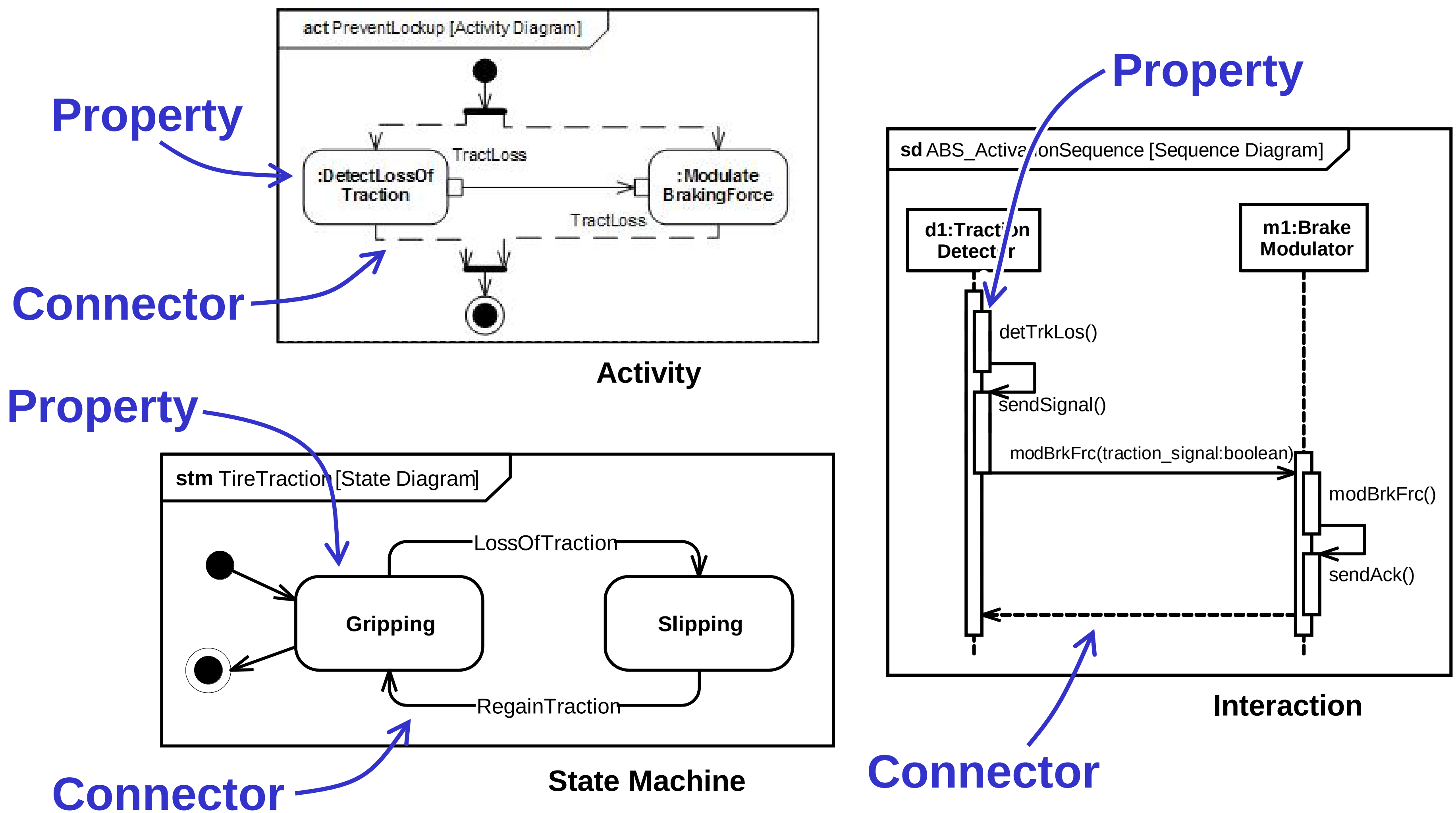
Original Problem

- UML has three behavior diagrams.
  – Activity, state, interaction.
- Very little integration or reuse between them.
  – Three underlying metamodels.
  – Three representations of temporal order.
- Triples the effort of learning UML and building analysis tools for it.

General Solution

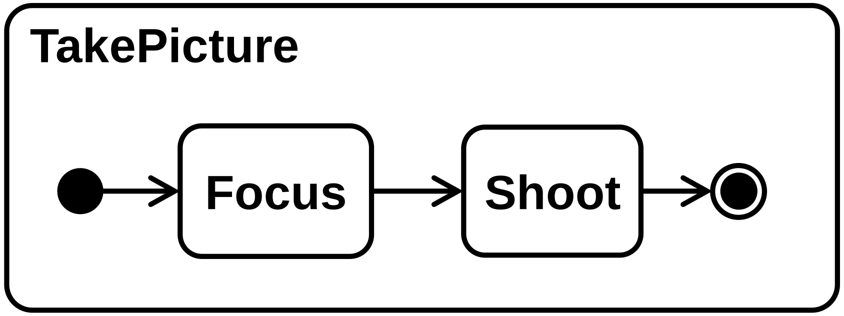
- Treat behaviors as assemblies of other behaviors.
  - Like objects are assemblies of other objects.
- Assembly = UML internal structure
  - Pieces represented by properties.
  - Put together by connectors.
- Put all behavior diagrams on the same underlying behavior assembly model.

Behaviors as Composite Structure

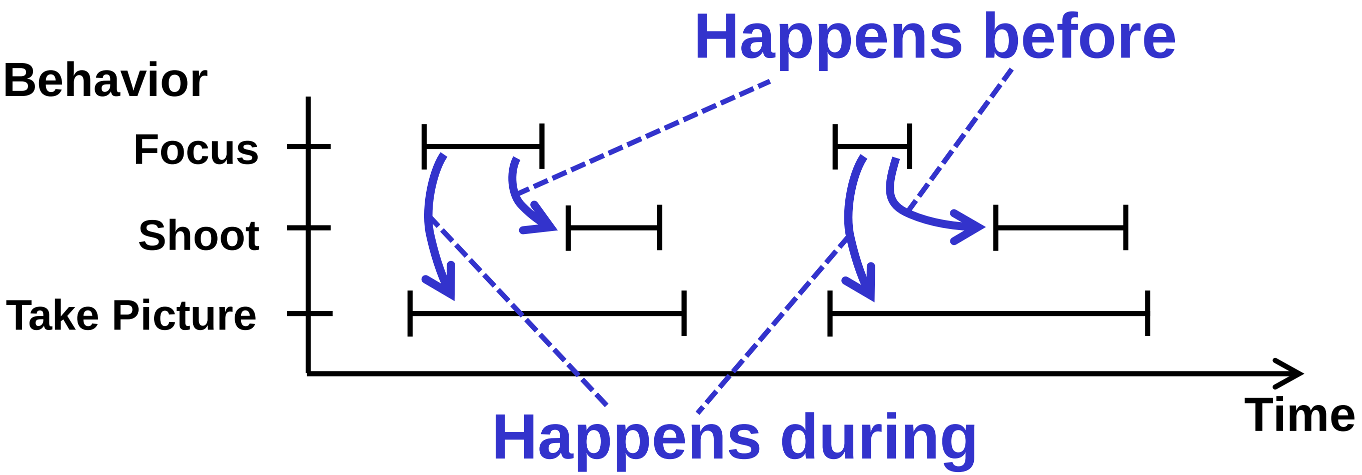


Behavior as Timing Constraints

Model (M1)



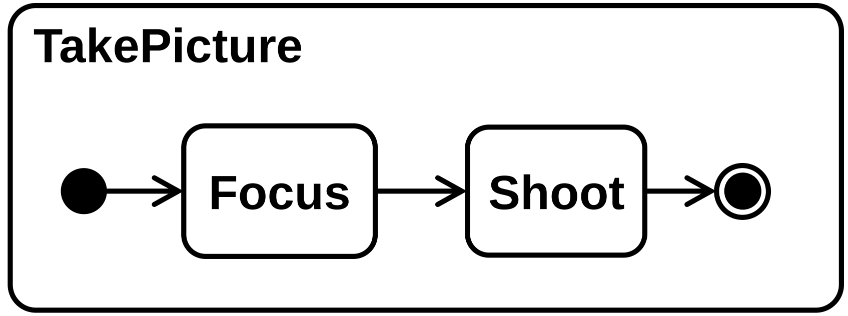
Things Being Modeled (M0)



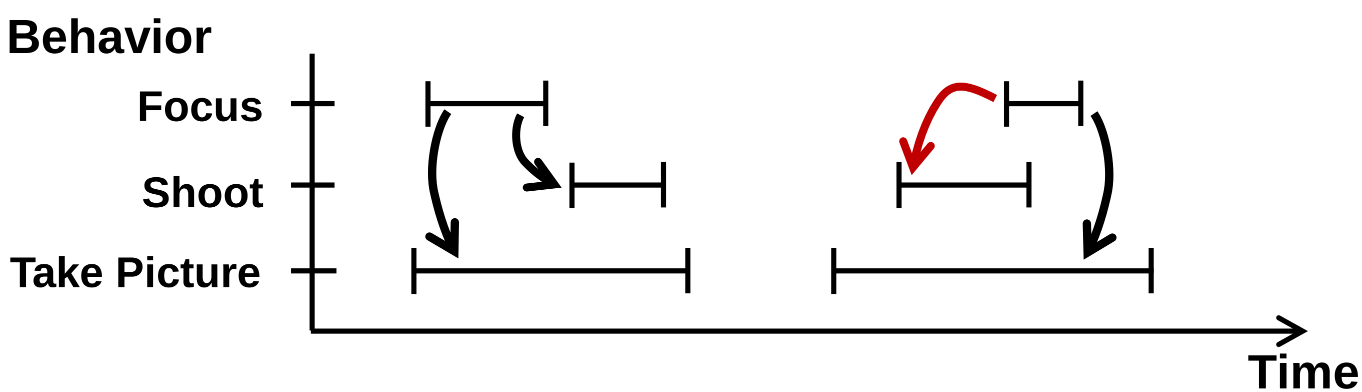
- **Behaviors model "things" happening over time.**
  - **With temporal relations (time constraints) between them.**

Behavior as Timing Constraints

Model
(M1)



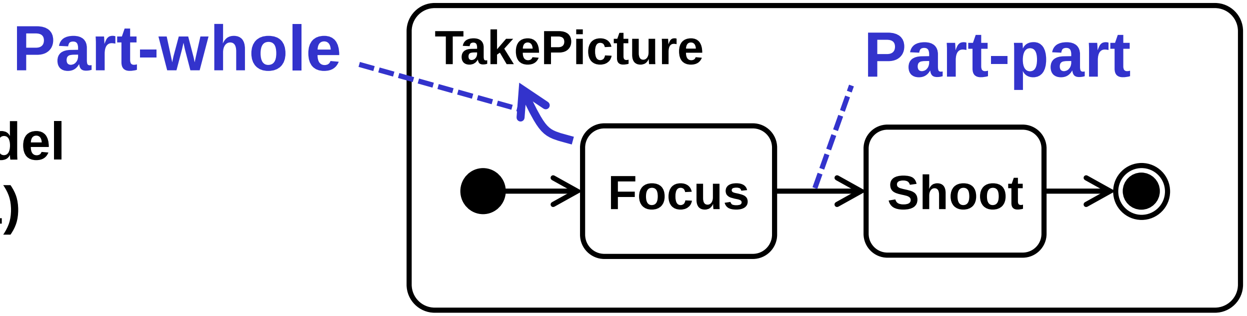
Things
Being
Modeled
(M0)



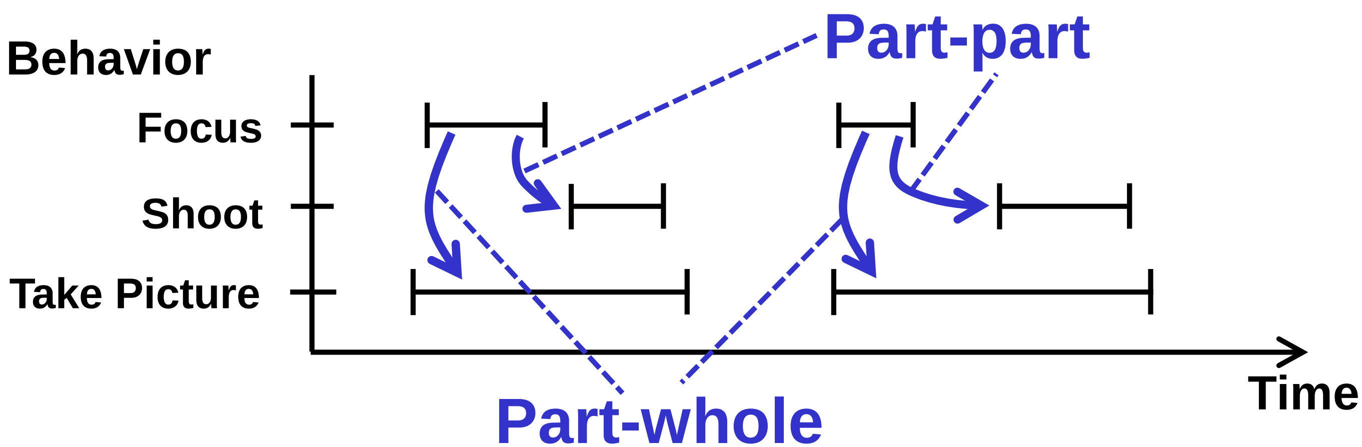
- **The TakePicture occurrence on the right does not follow the behavior model.**

Behavior as "Composite Timing"

Model
(M1)

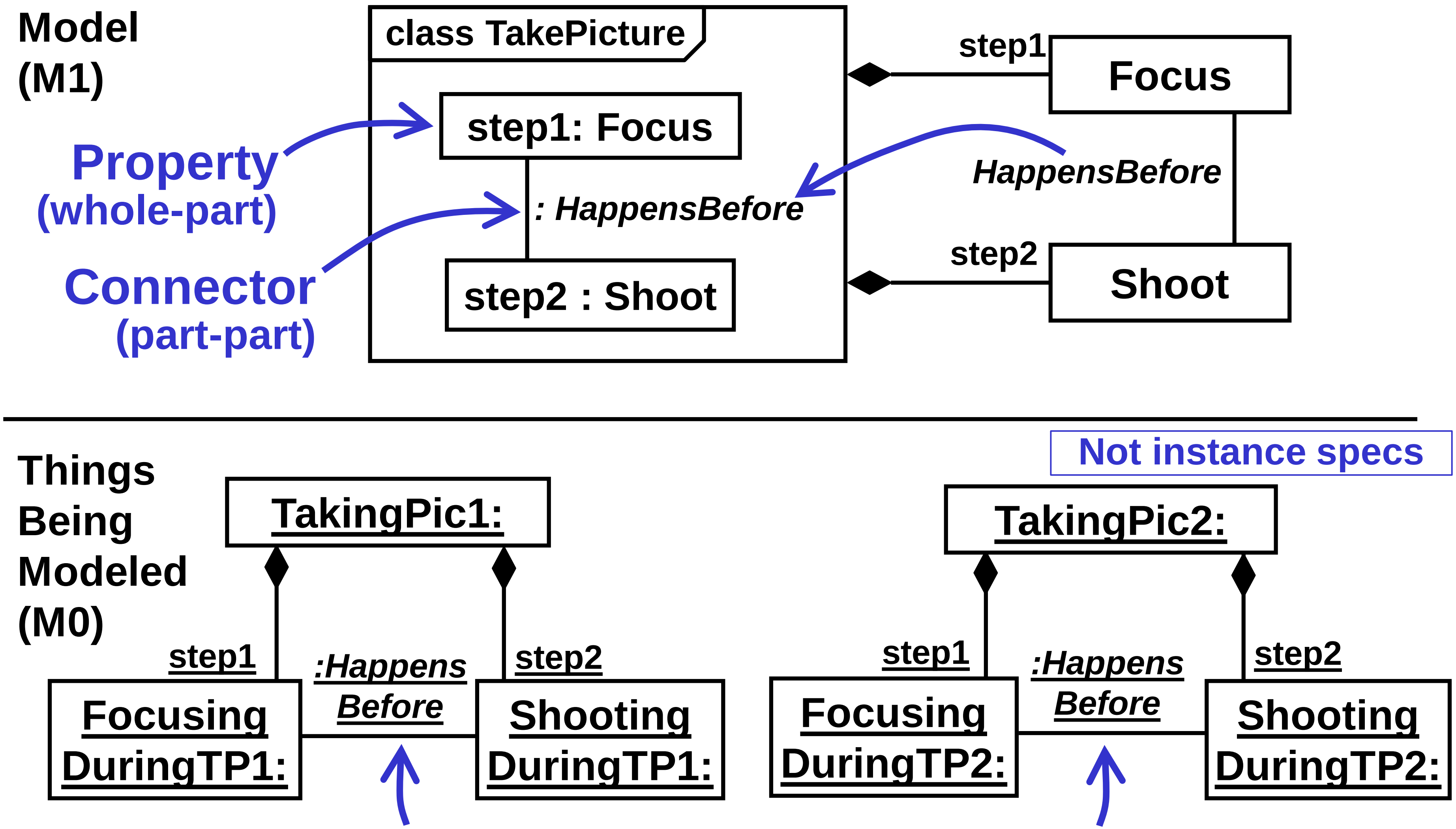


Things
Being
Modeled
(M0)



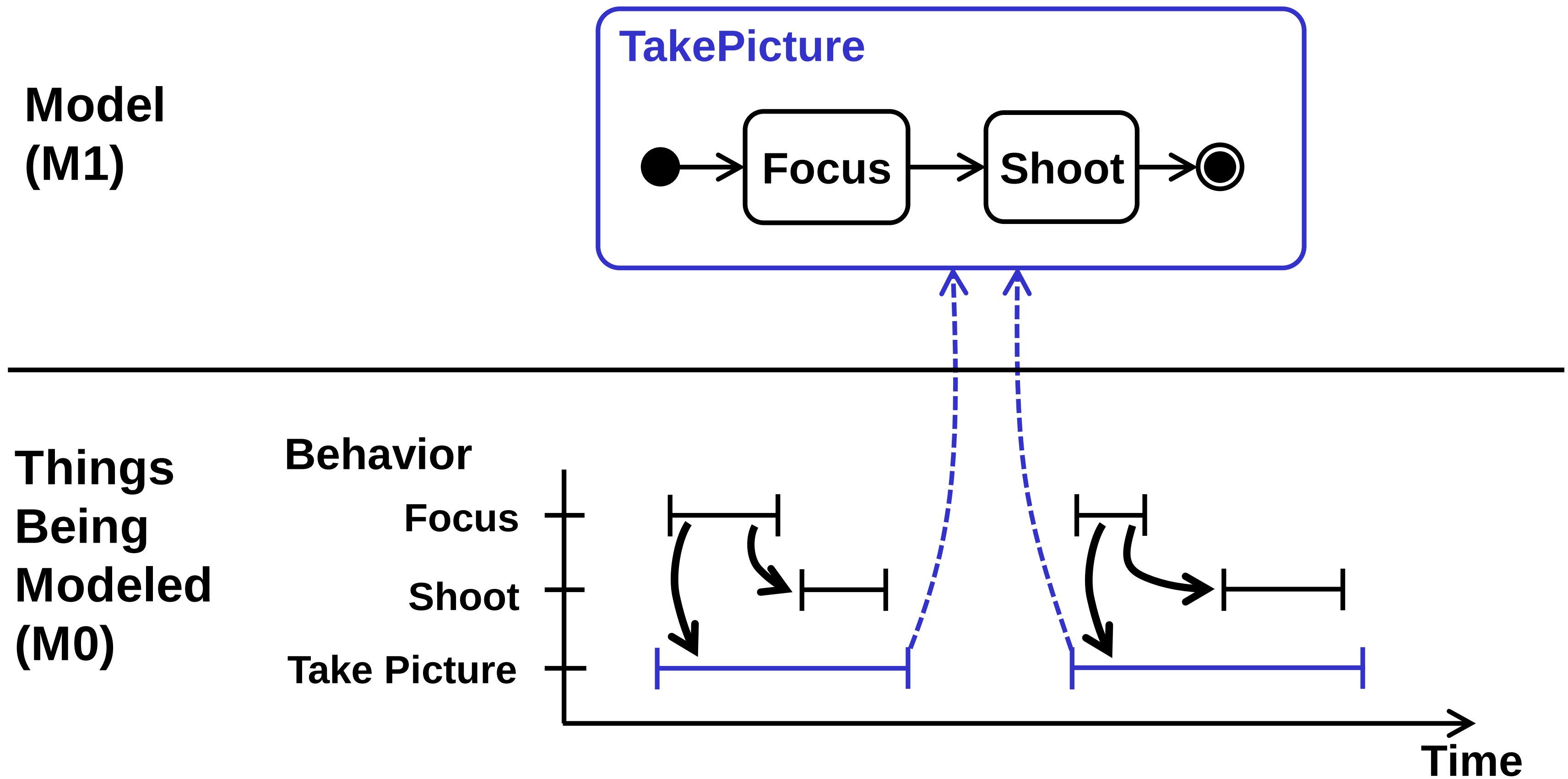
- **Composite structure relations are temporal:**
  - **Part-whole = happens during.**
  - **Part-part = happens before.**

Behavior as “Composite Timing”



Focusing before shooting in same taking picture

Model and Things Being Modeled



- **Dashed arrows between M1 and M0 mean ....**

M0 → M1 Synonyms

Classified by

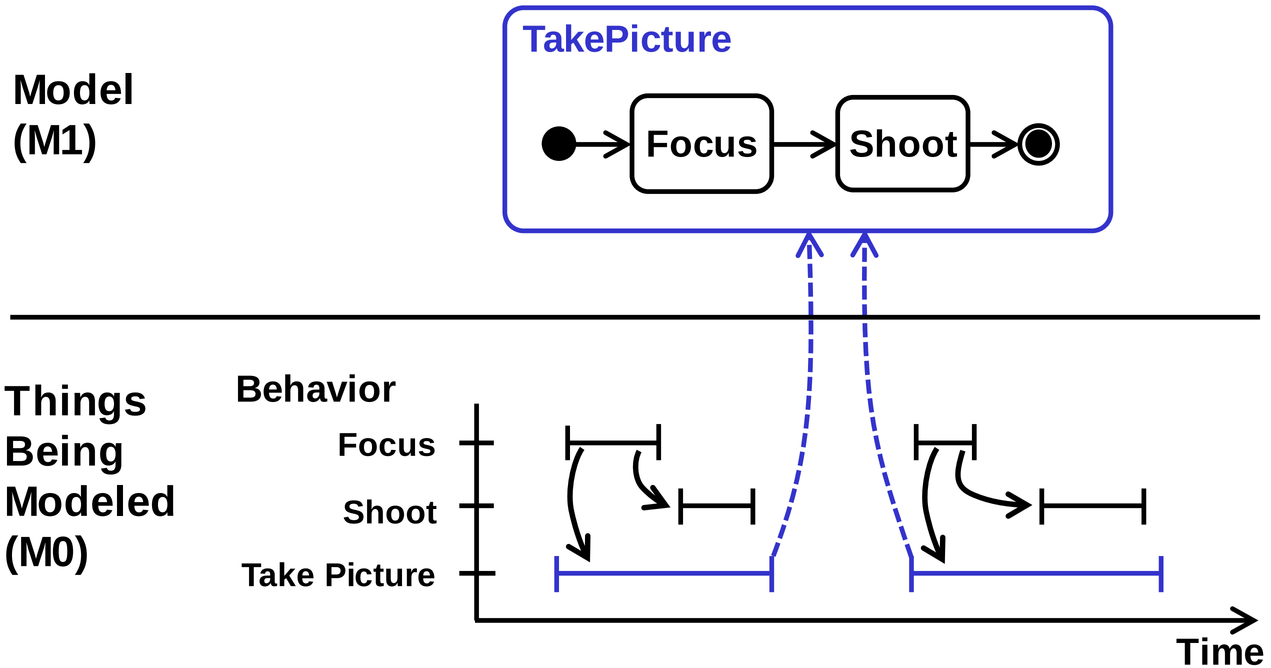
Modeled by

Specified by

Conforms to

Follows

Satisfies (logically)



Not quite: Instance of (in the OO sense)

Not *at all* : Execution of (in the software sense)

Behavior: What's Being Modeled?

Real,
Simulated,
or Desired
Things Being
Modeled (M0)

Not instance specs.

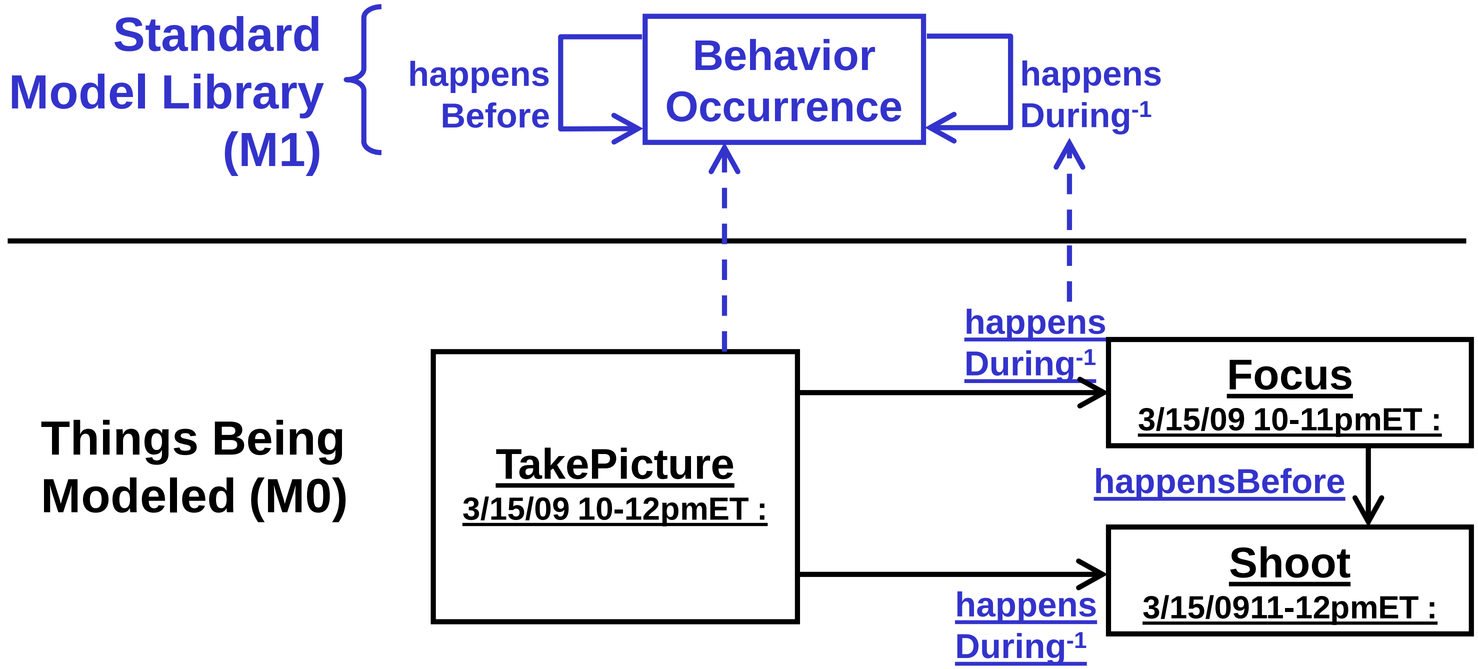
TakePicture
3/15/09 10-12pmET :

Focus
3/15/09 10-11pmET :

Shoot
3/15/0911-12pmET :

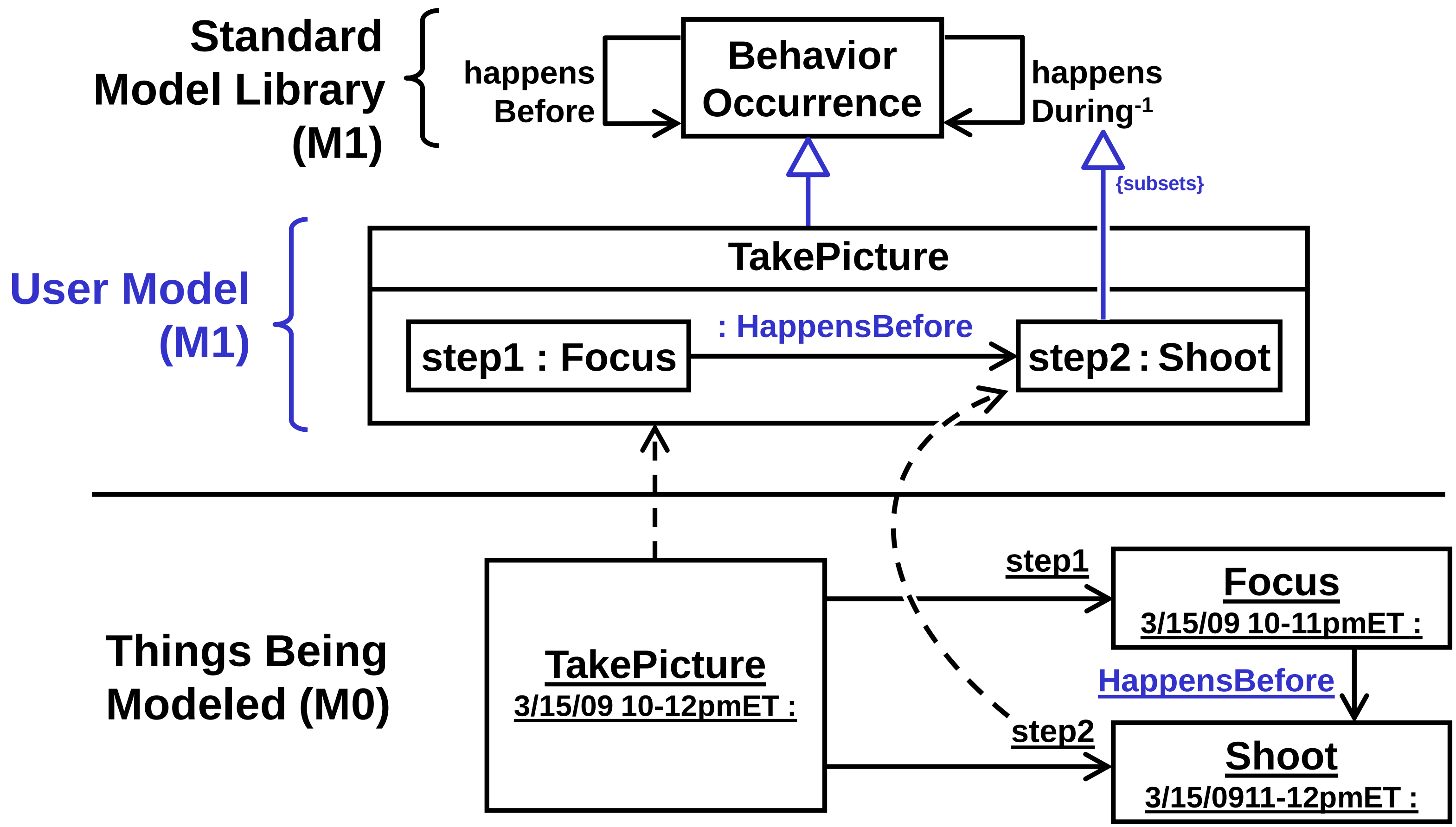
- "Things" that occur in time
  - Eg, taking a picture, focusing, etc.
  - Not "behaviors", "actions", etc.

Behavior: What's in Common?



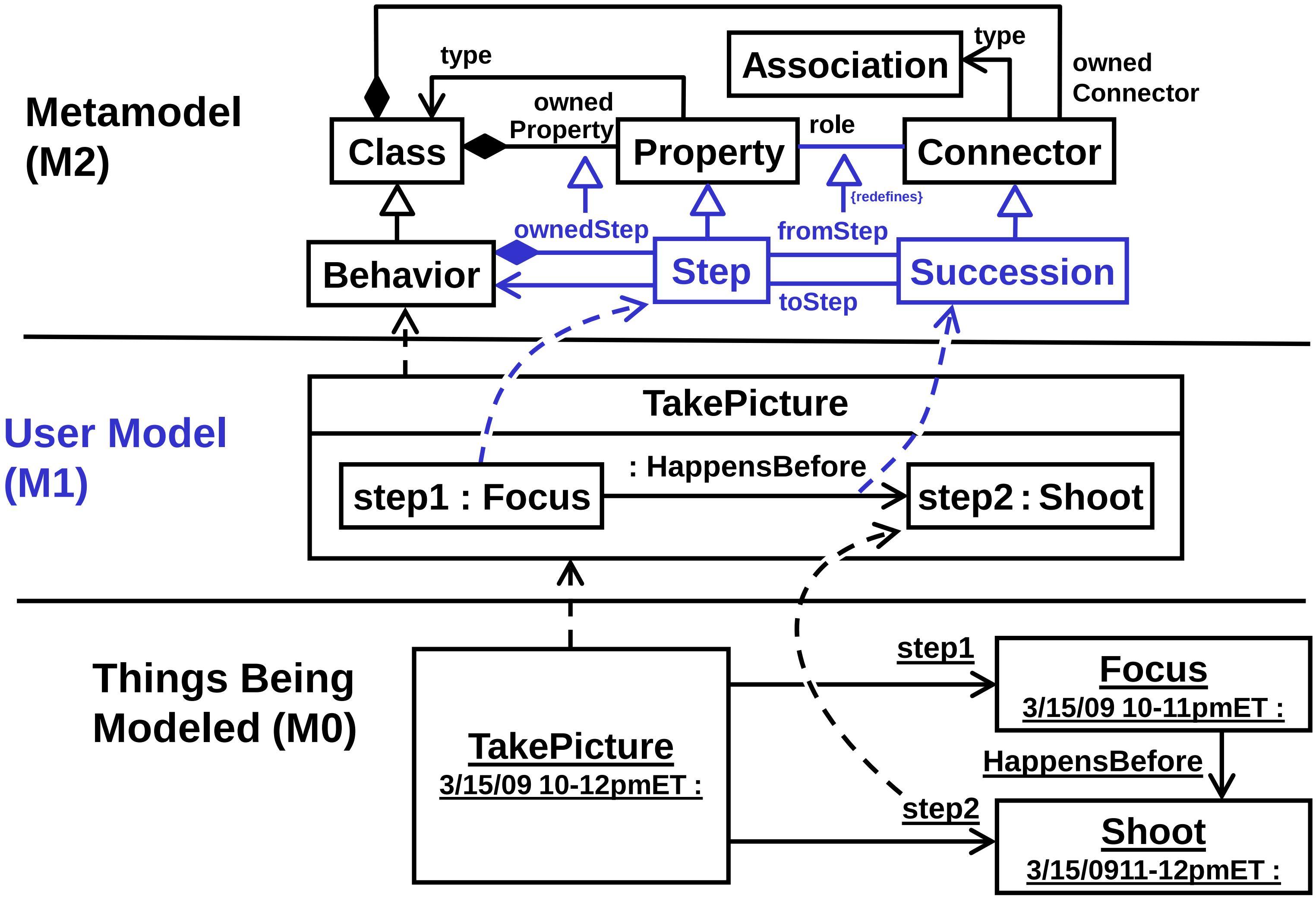
- **They happen before or during each other.**
  - **Construct M1 library for this.**
  - **Use it to classify things being modeled.**

Behavior: Use Library



- Specialize library classes and subset/redefine library properties.

Behavior: Too repetitive at M1?

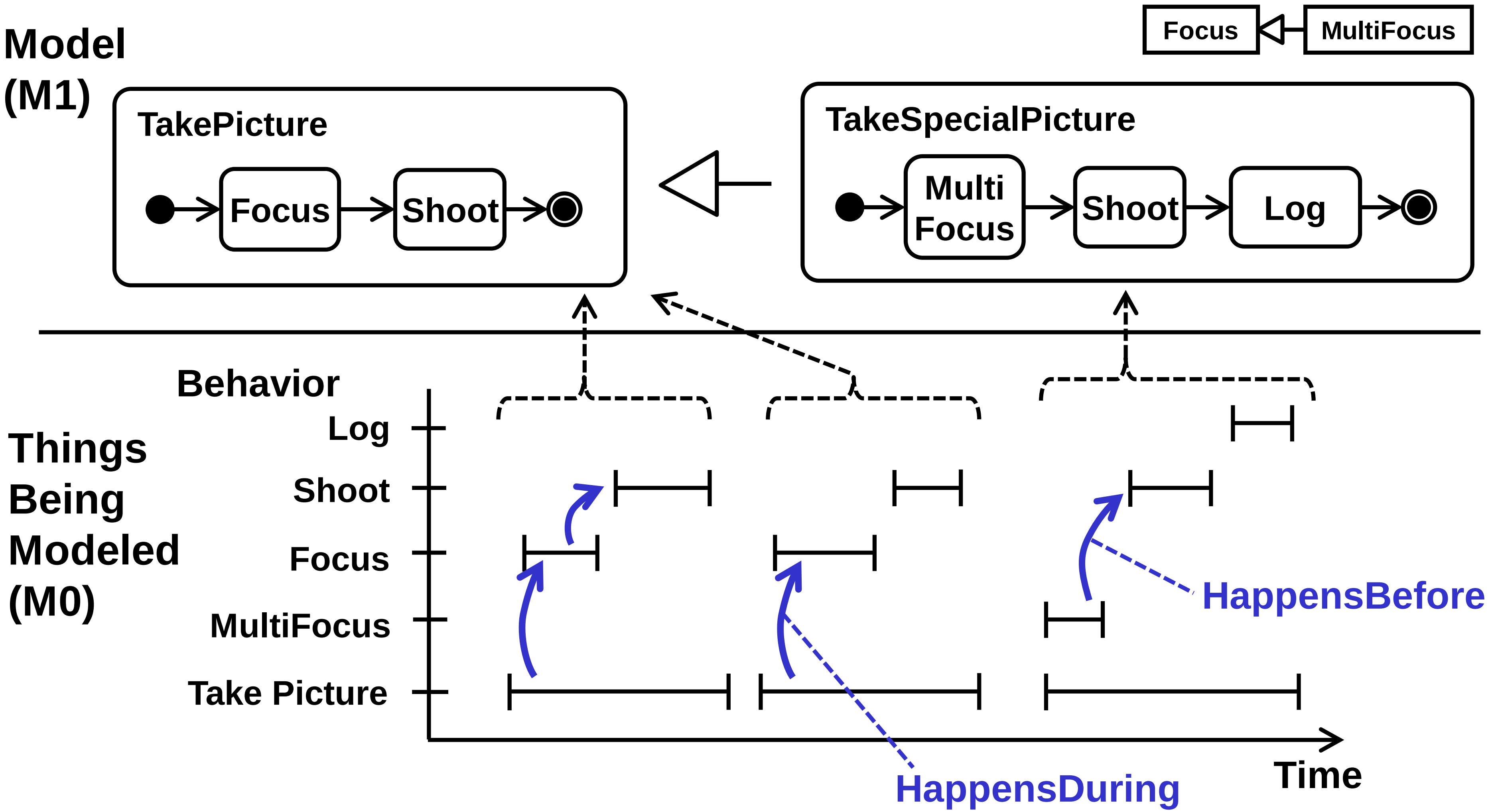


- **Capture M1 patterns in M2 elements.**
  - **Tools apply patterns automatically.**

Benefits: Original Problem

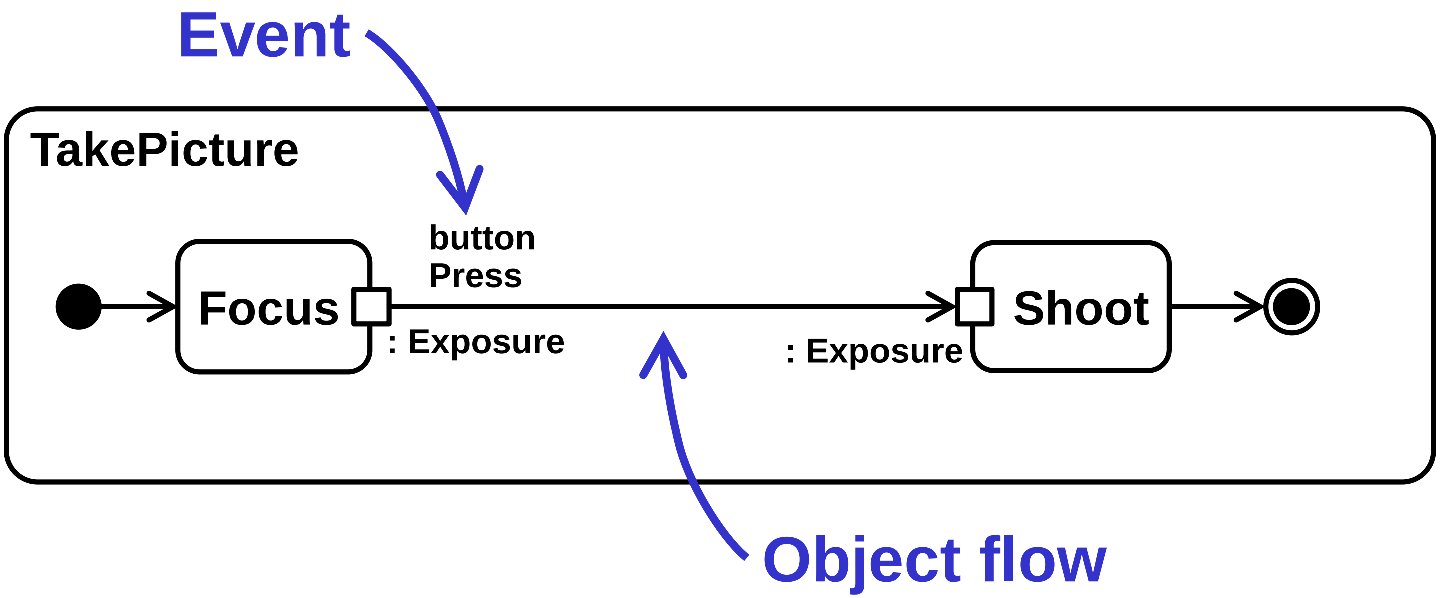
- Flexibility in using metamodels
  - Add metaelements as needed to simplify library usage.
- Many metaelements become synonyms
  - Application / method / diagram-specific terminology sharing same semantics.
  - M2 actions, states, etc, => M1 happensDuring
- Learning UML and building analysis tools for it is easier
  - Due to shared semantics for variety of modeling language terminology.

Benefits: Expressiveness



- Constraints are inherited in UML
  - including temporal constraints.

Benefits: Expressiveness



- **Combine activity and state machines.**
  - **States and actions happen during their "containing" occurrences, ordered in time.**

Benefits: Modeled Semantics

- **UML semantics is written in free text**
  - **Specifying an execution procedure for activities and state machines:**

Tokens are *offered* to an ActivityEdge by the source ActivityNode of the edge. Offers propagate through ActivityEdges and ControlNodes, according to the rules associated with ActivityEdges (see below) and each kind of ControlNode (see sub clause 15.3) until they reach an ObjectNode (for object tokens) or an ExecutableNode (for control tokens and some object tokens as specified by modelers, see ObjectNodes in sub clause 15.4). Each kind of ObjectNode (see sub clause 15.4) an
accepted
Activity
which a

The processing of Event occurrences by a StateMachine execution conforms to the general semantics defined in Clause 13. Upon creation, a StateMachine will perform its initialization during which it executes an initial compound transition prompted by the creation, after which it enters a *wait point*. In case of StateMachine Behaviors, a wait point is represented by a stable state configuration. It remains thus until an Event stored in its event pool is dispatched. This Event is evaluated and, if it matches a valid Trigger of the StateMachine and there is at least one enabled Transition that can be triggered by that Event occurrence, a single StateMachine *step* is executed. A step involves executing a compound transition and terminating on a stable state configuration (i.e., the next wait point). This cycle then repeats until either the StateMachine completes its Behavior or until it is asynchronously terminated by some external agent.

  - **and trace classification in interactions:**

Clause 13, Common Behaviors, describes the general semantics of the execution of Behaviors. Interactions are kinds of Behaviors that model emergent behaviors, as defined in sub clause 13.1. As discussed in sub clause 13.2.3, the execution of a Behavior results in an execution trace. Such a trace is a sequence of event occurrences, which, in this clause, will be denoted <e1, e2, . . . , en>. Each event occurrence may also include information about the values of all relevant objects at the point of time of its occurrence.
The semantics of an Interaction are expressed in terms of a pair [P, I], where P is the set of valid traces and I is the set of invalid traces. P ! I need not be the whole universe of traces. Two Interactions are equivalent if their pairs of trace-sets are equal. The semantics of each construct of an Interaction (such as the various kinds of CombinedFragments) are

- **Model in standard libraries.**

Benefits: Classification Semantics

- Standard execution models for UML
  - fUML, PSCS, PSSM
  - Procedures that create a behavior occurrence
    - Conforming to a UML model.
  - Don’t tell whether
    - An existing behavior occurrence conforms.
    - Tools are producing correct occurrences
- Classification does the opposite
  - Tells whether an existing behavior occurrence conforms to a model.
  - Doesn’t say how to create an occurrence.
  - Execution engines are constraint solvers.

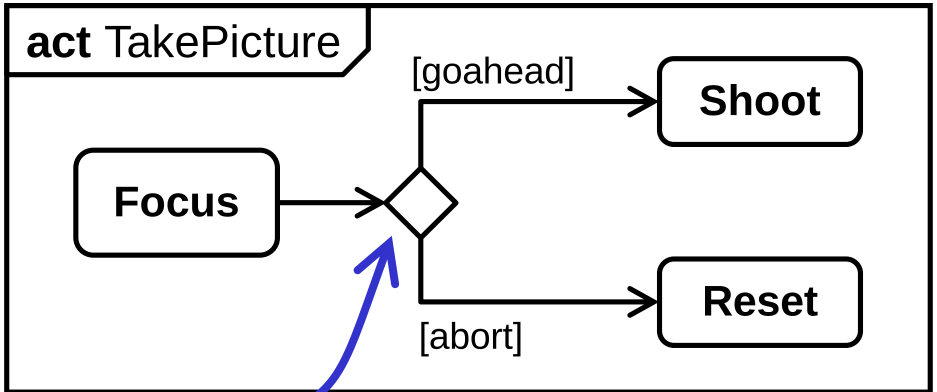
Overview

- RoadMap
- Motivation
  - Behavior, review
  - **Activities, requirements**
- Activities Solution
  - Control nodes
  - Loops
  - Specialization
- Summary

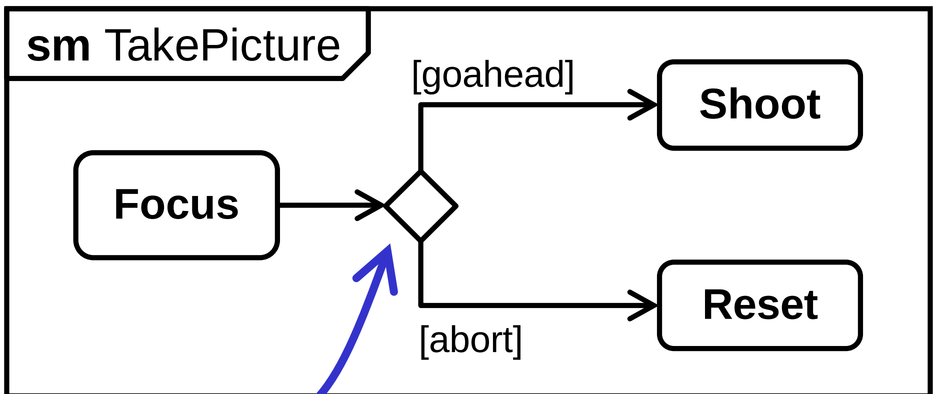
Activity Problem

- UML has three ways to coordinate sequences of behaviors:
  - Activities have control nodes.
  - State machines have pseudostates.
  - Interactions have combined fragments.
- Very little integration or reuse.
  - Three underlying metamodels.
  - Three representations of “control”.
- Triples the effort of learning UML and building analysis tools for it.

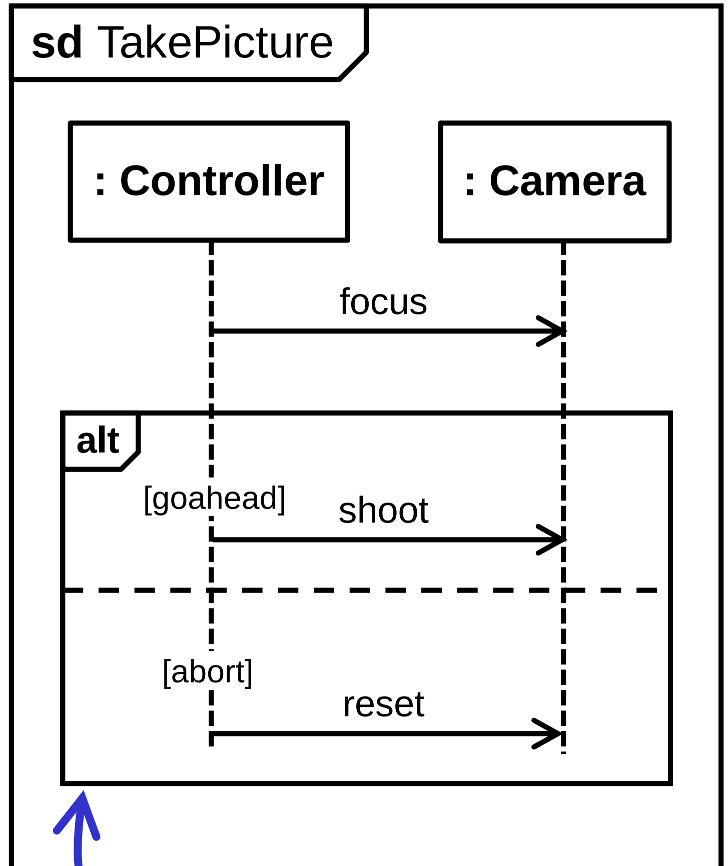
Activity Problem, Control



Decision Node

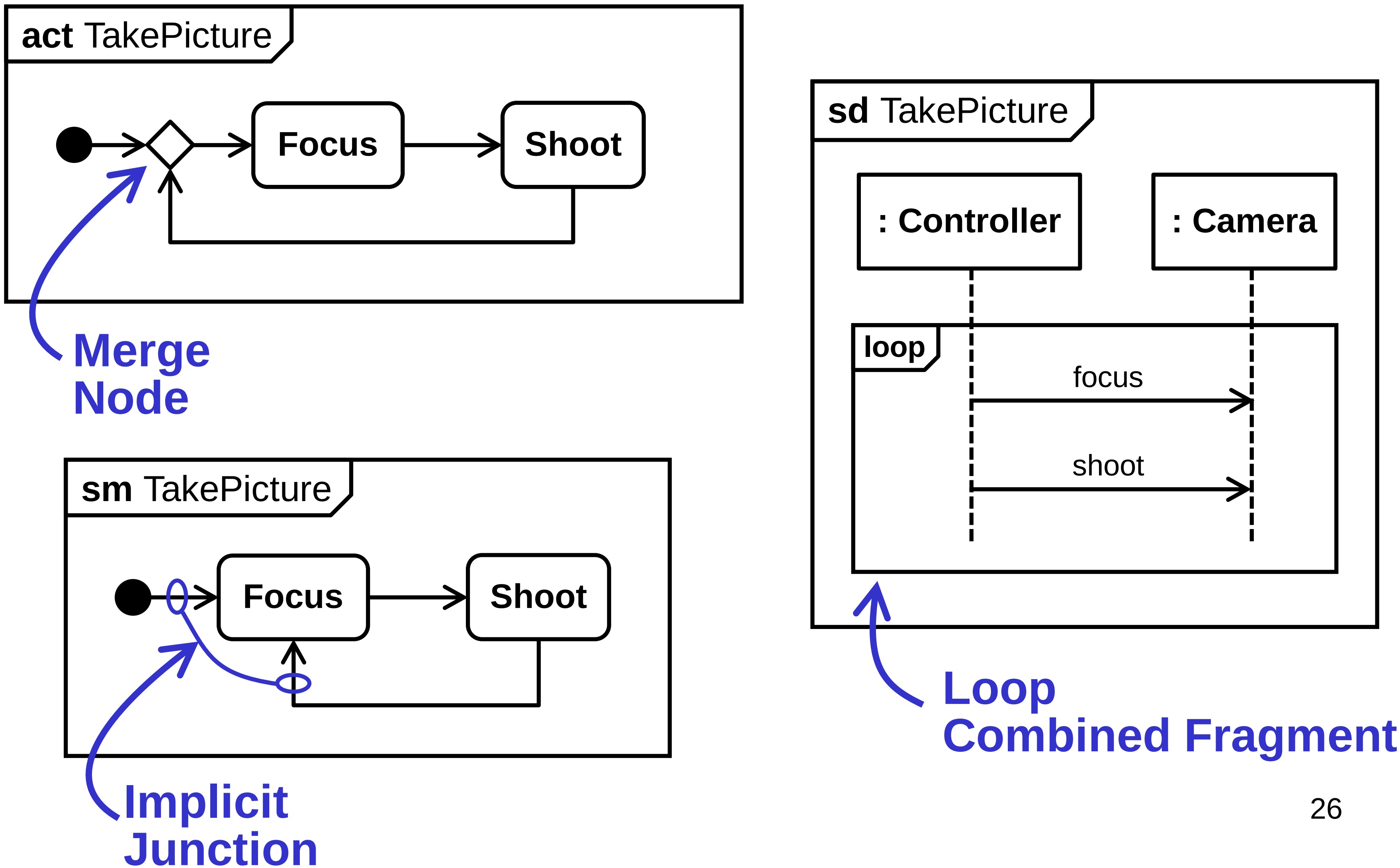


Choice Pseudostate



Alternative Combined Fragment

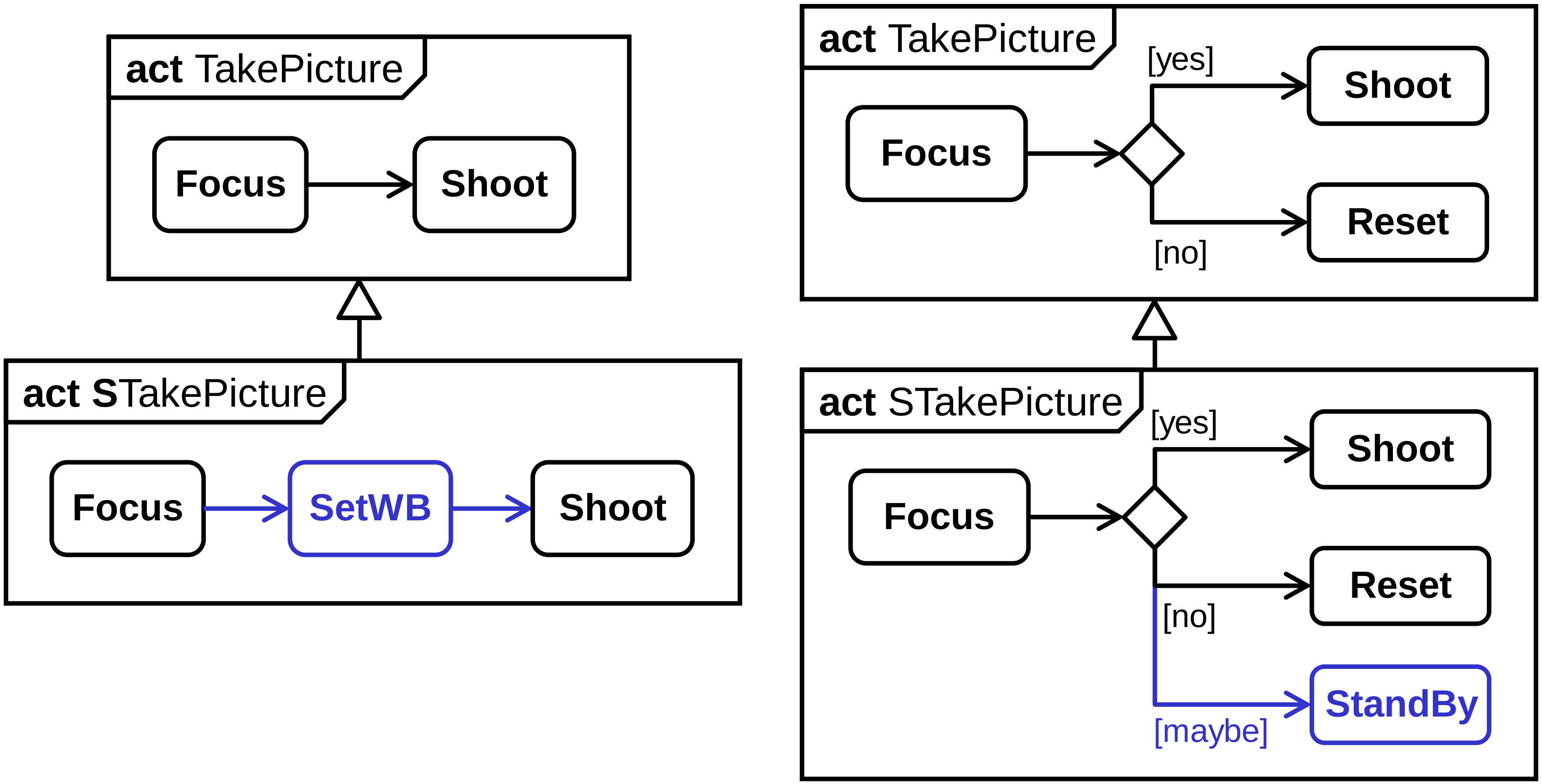
Activity Problem, Loops



Activity Problem, Specialization

- Behaviors are classes in UML
  - Their M0 instances are executions.
- Classes can be special/generalized
  - Semantics = sub/supersets of M0 instances
    = inheriting timing constraints
- Behaviors can special/generalized, but …
- Generalization semantics not used.
  - Nothing said in activities.
  - SMs have syntactic redefinition rules.
  - Interactions use trace semantics.

Activity Problem, Specialization



- **What can be added in specialized behaviors and still obey inherited timing constraints?**

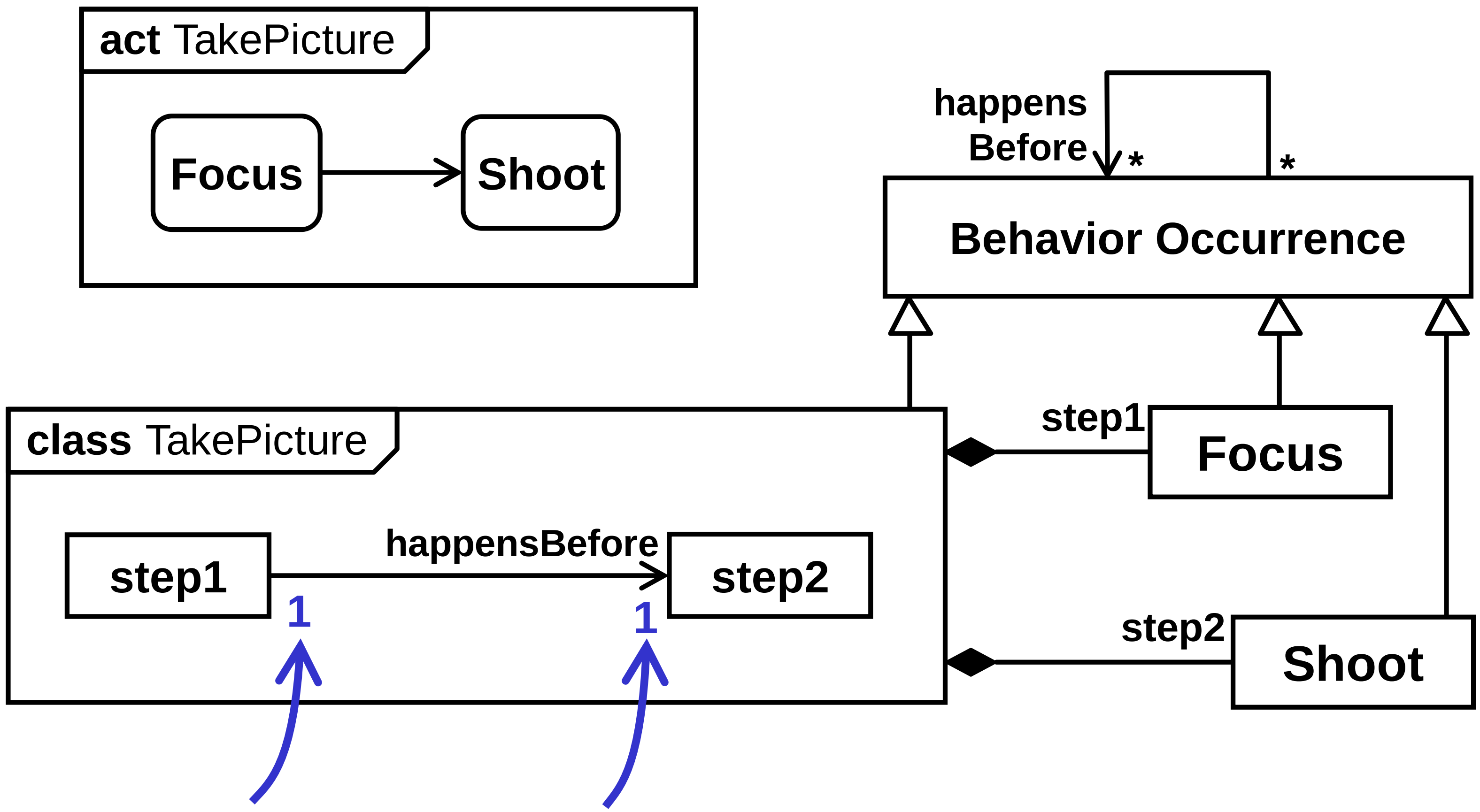
Activity Requirements

- Single model & semantics for coordinating sequences of behaviors
  - Control nodes, loops.
- Use generalization semantics for specializing behaviors.
  - Subsets of occurrences / executions

Overview

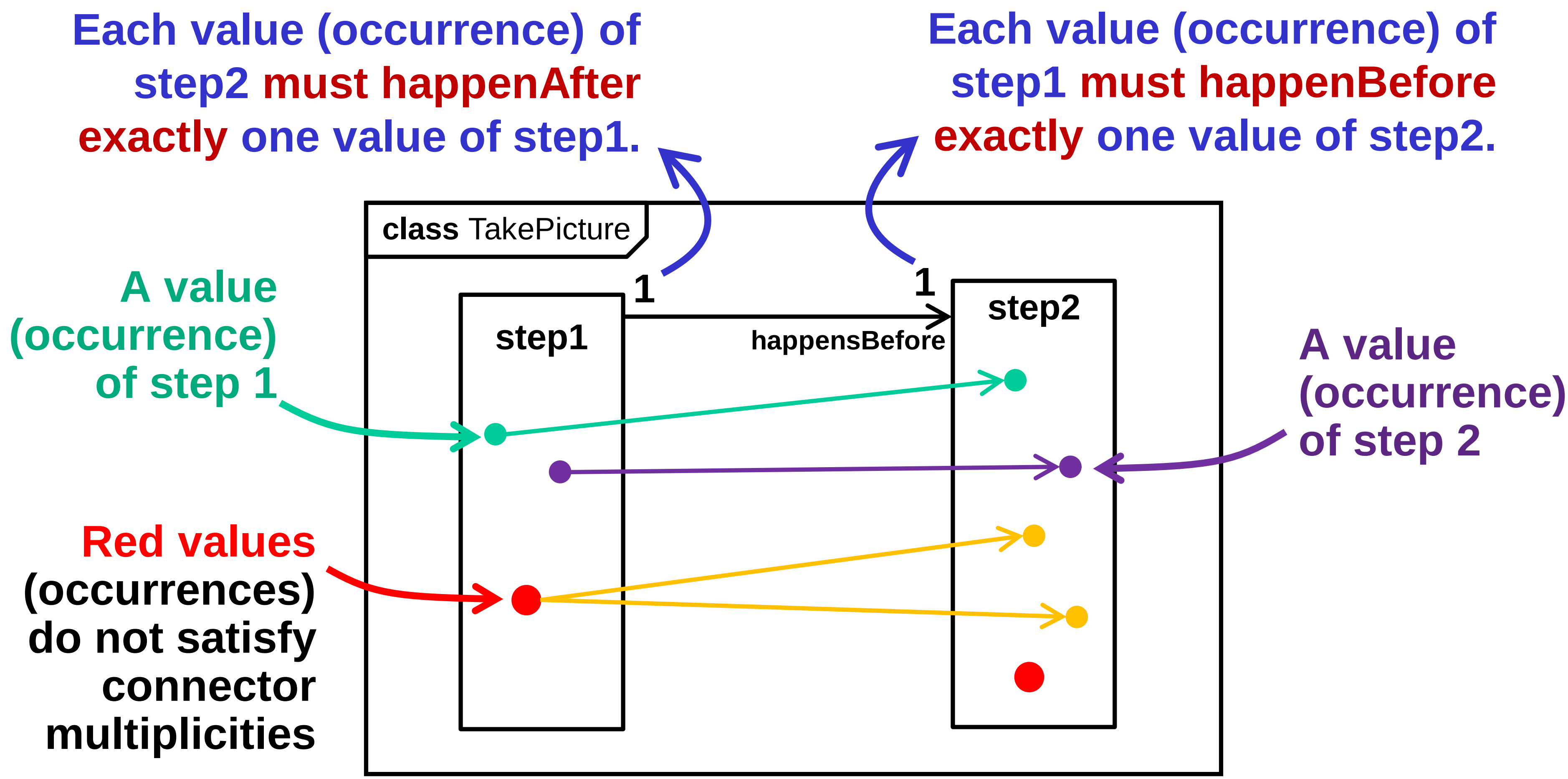
- RoadMap
- Motivation
  - Behavior, review
  - Activities, requirements
- **Activities Solution**
  - **Control nodes**
  - Loops
  - Specialization
- Summary

Connector Multiplicities



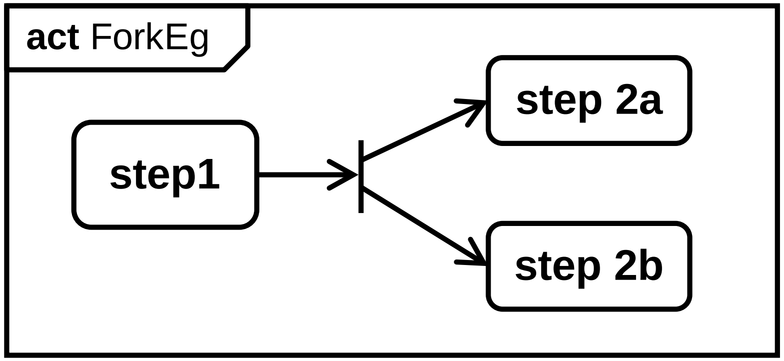
- Connector multiplicities constrain the number of links due to a connector for each value of the end properties.

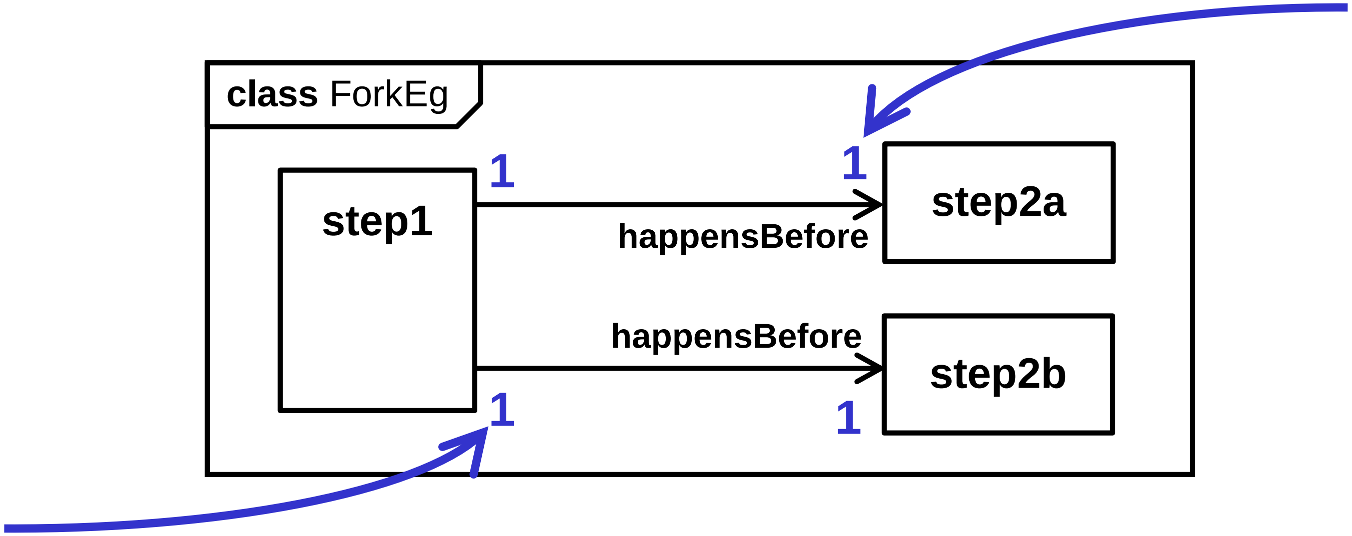
Connector Multiplicities



- Satisfying and not satisfying occurrences
  - Valid / invalid
  - Conforming / nonconforming, etc

Control Nodes (Fork)



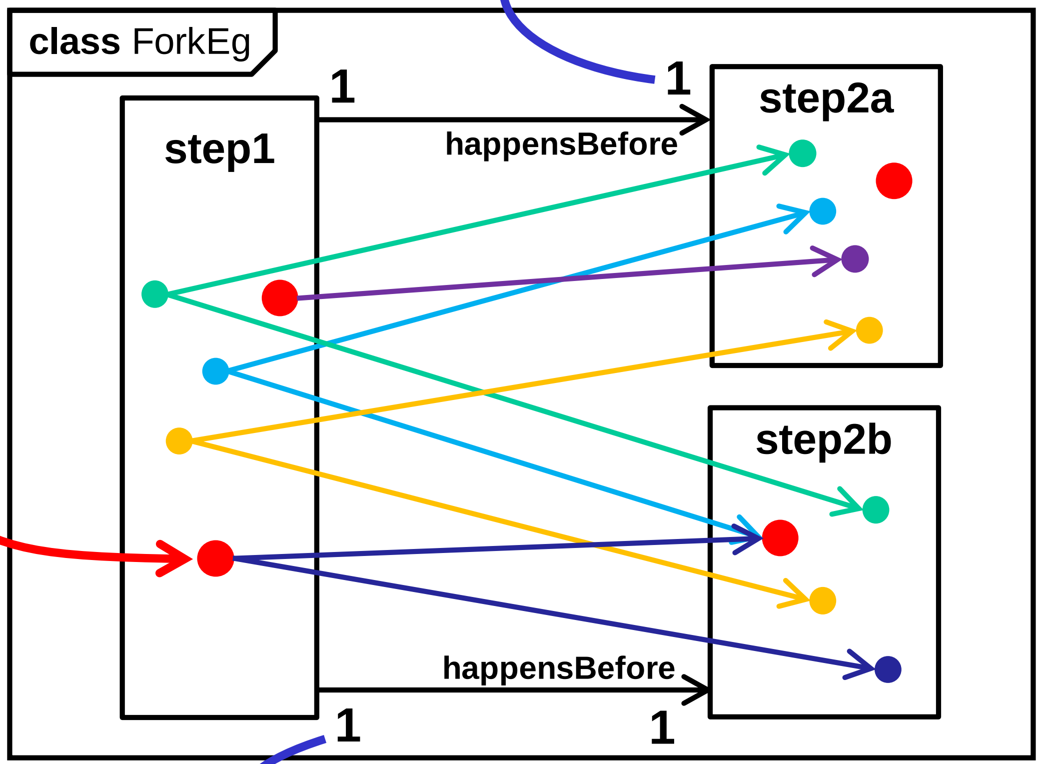


- **Same multiplicities, multiple connectors**

Connector Multiplicities (Fork)

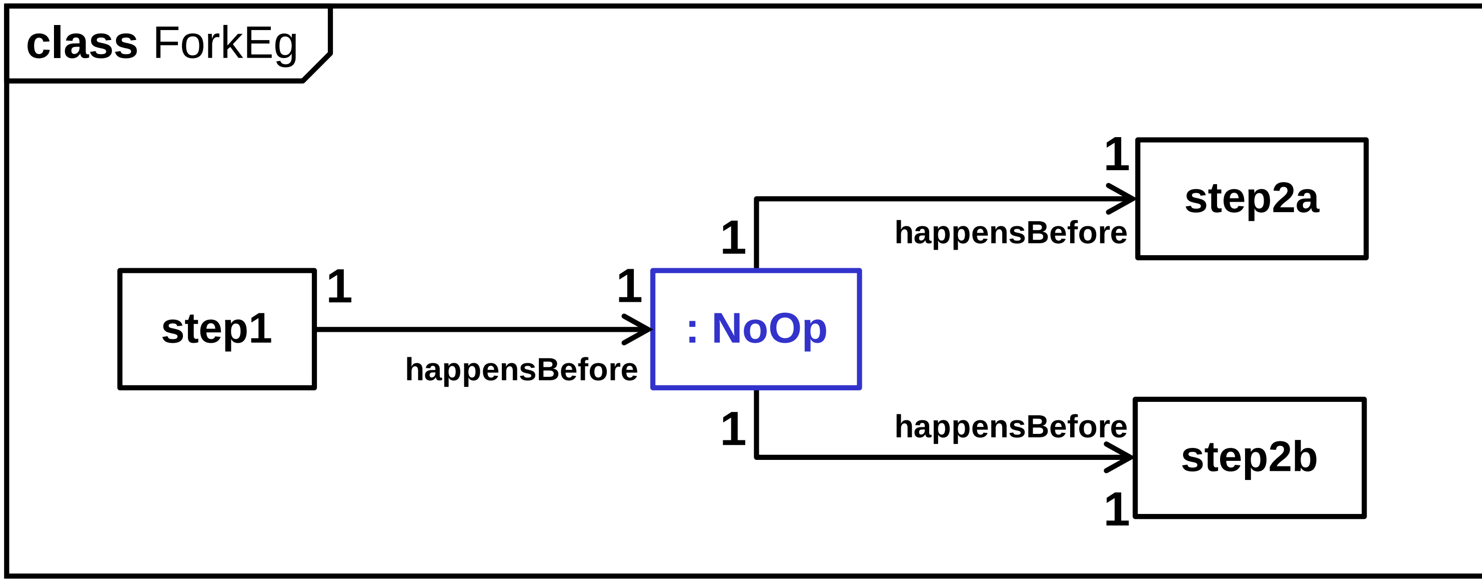
Each value (occurrence) of step1 **must happenBefore exactly** one value of step2a and of step2b.

Red values (occurrences) do not satisfy connector multiplicities



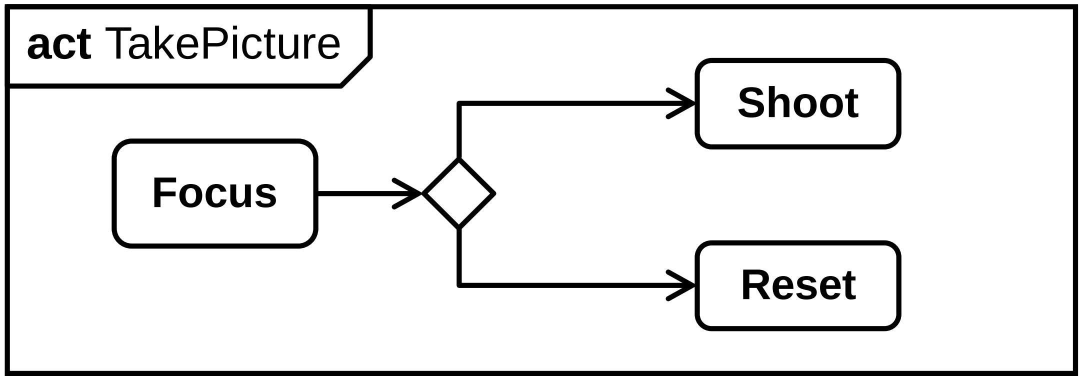
Each value (occurrence) of step2a and of step2b **must happenAfter exactly** one value of step1.

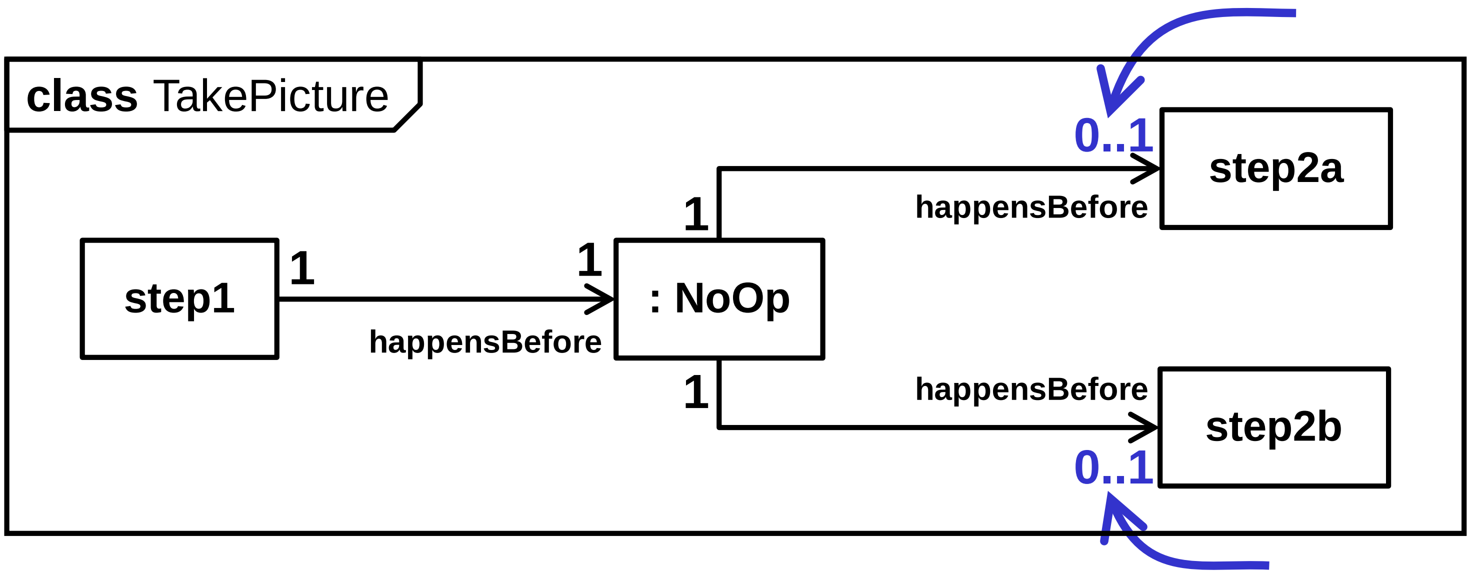
Fork Nodes, Graphic



- NoOp is a predefined behavior with no steps and zero duration.
  – Introduced for “node” appearance.
- Same effect as previous slide.

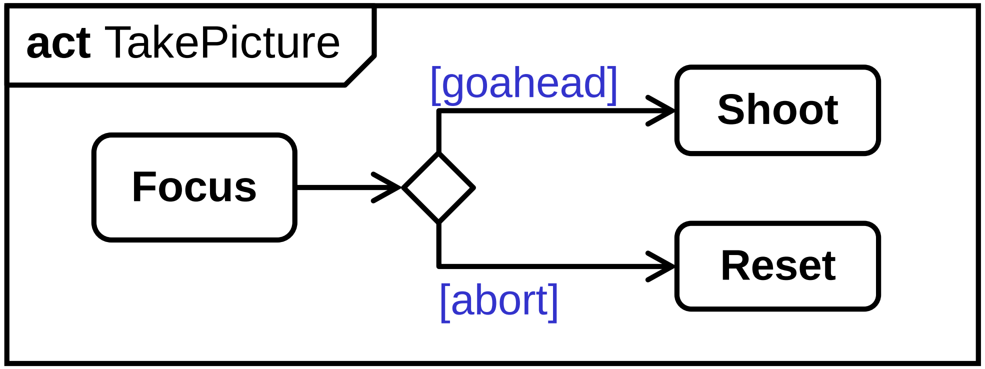
Control Nodes (Decision)

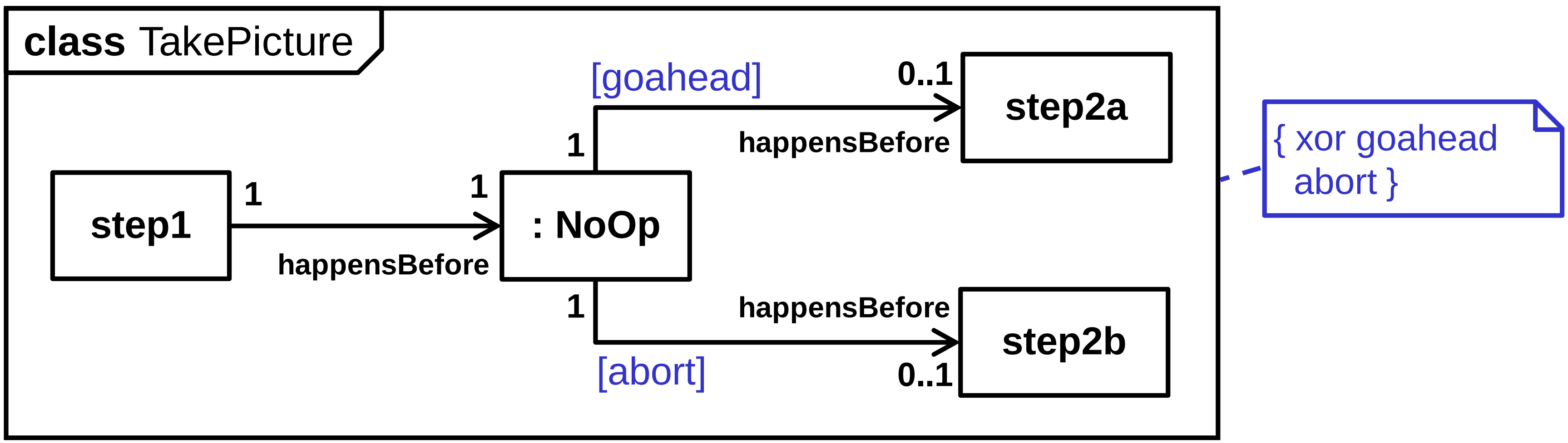




- **Connector multiplicities loosened**
- **What ensures that step2a/b happen at all?**

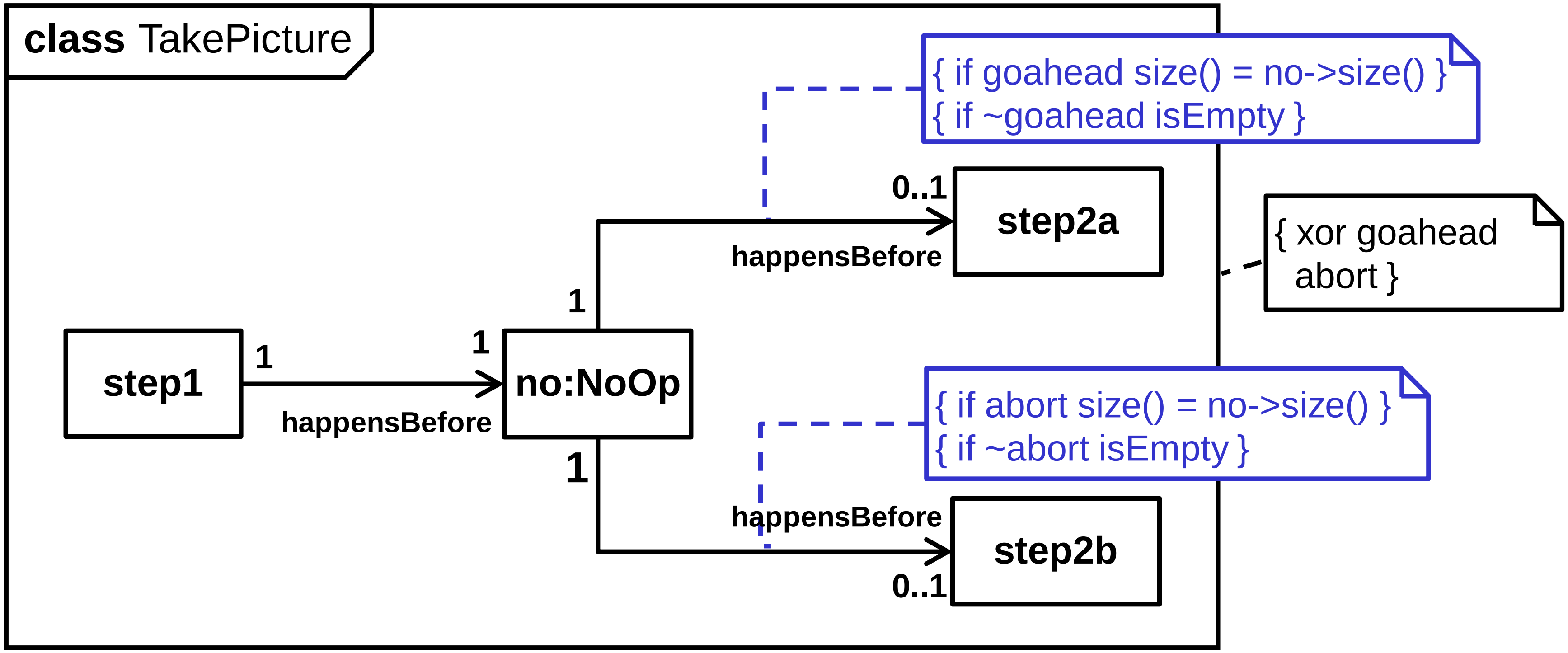
Decision Nodes, Closed, #1





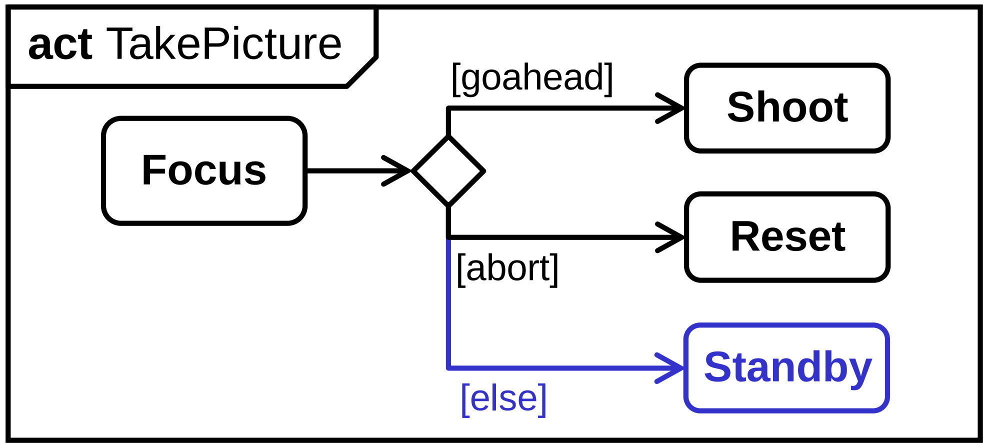
- **Add guards where exactly one succeeds.**

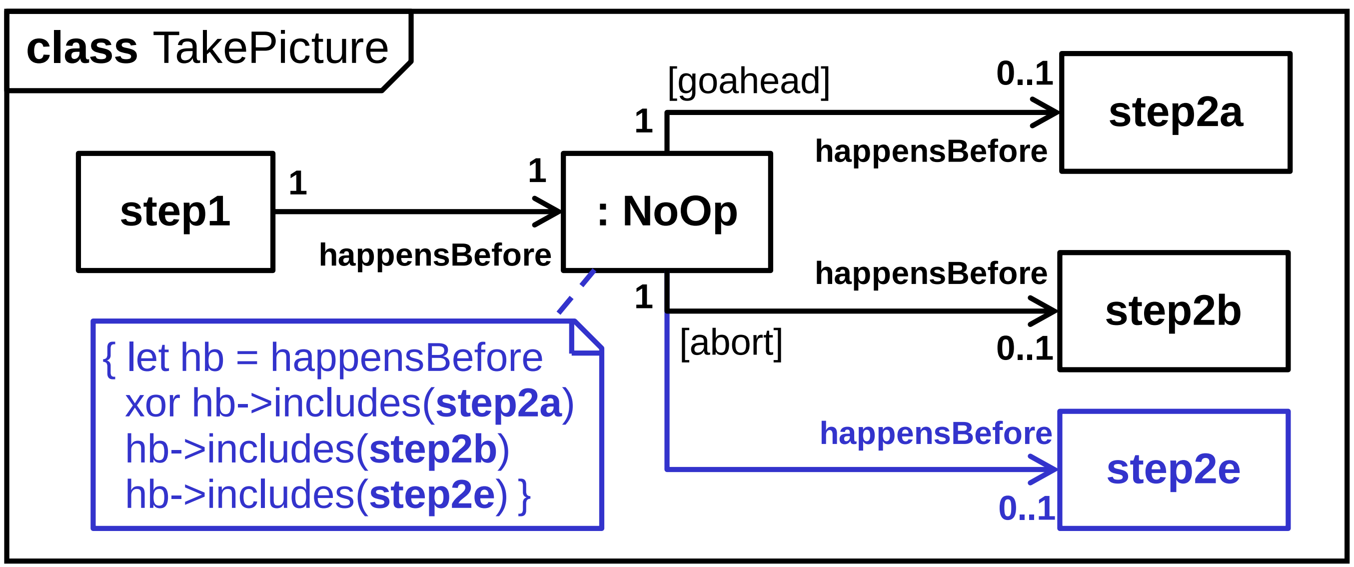
Decision Nodes, Closed, #1



- **Guard conditions must be sufficient to infer (require) connector values.**

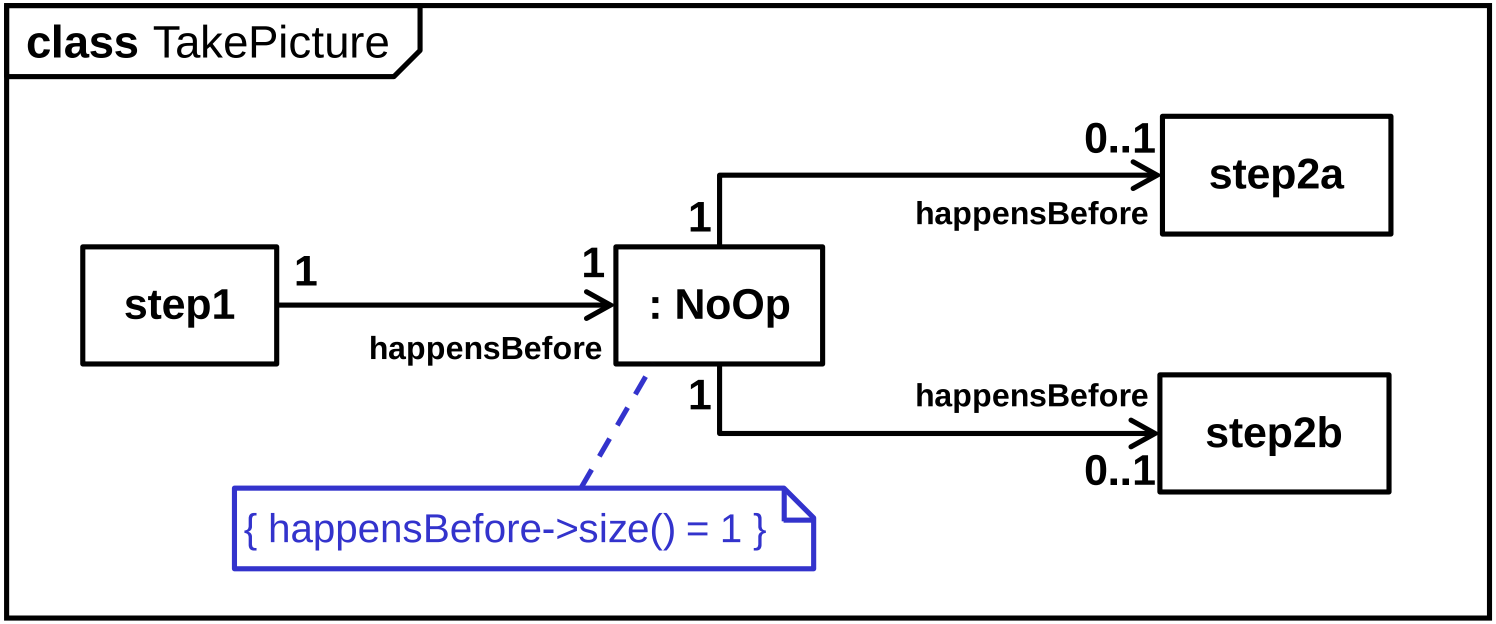
Decision Nodes, Closed, #2





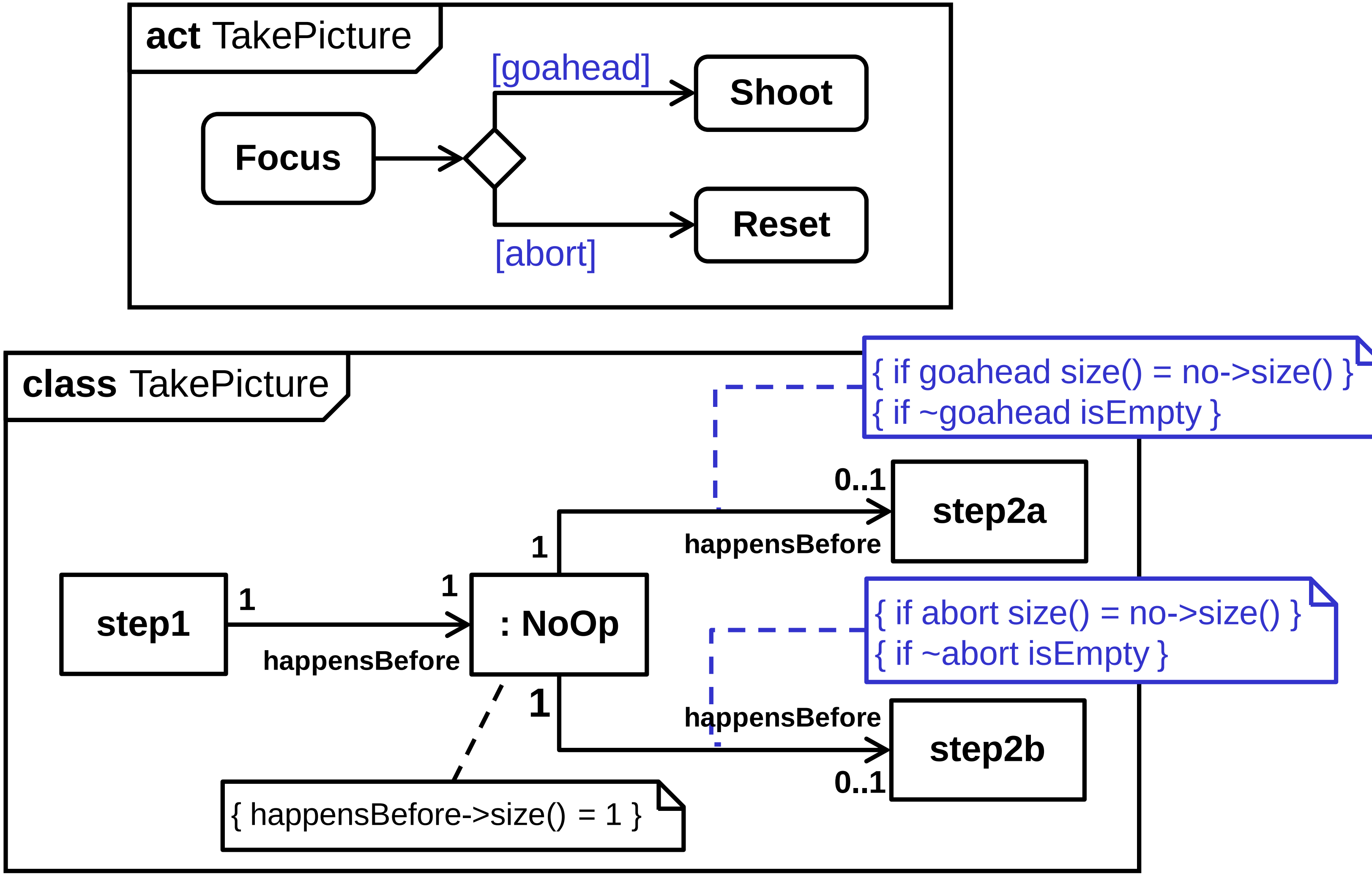
- Enumerate alternative branches
- Supports else (empty guard).

Decision Nodes, Open



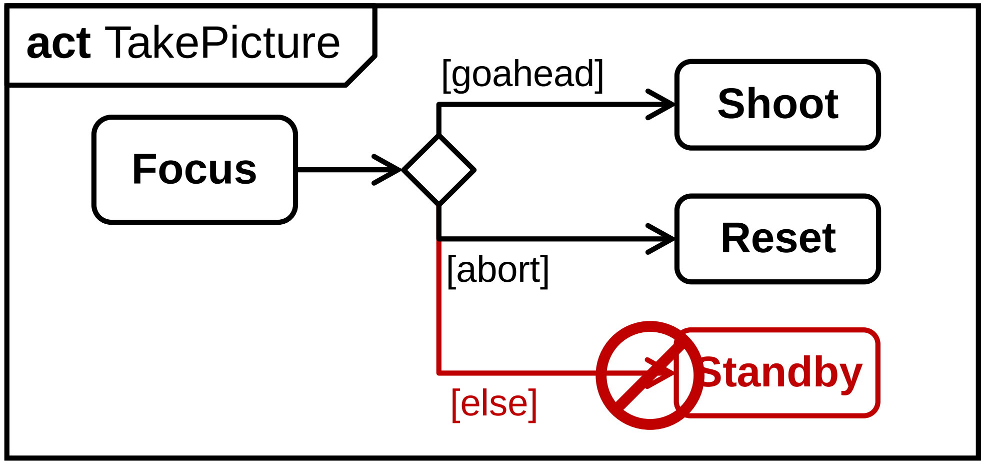
- **Pro: Same for any number of branches.**
- **Con: Doesn’t require branches to happen.**

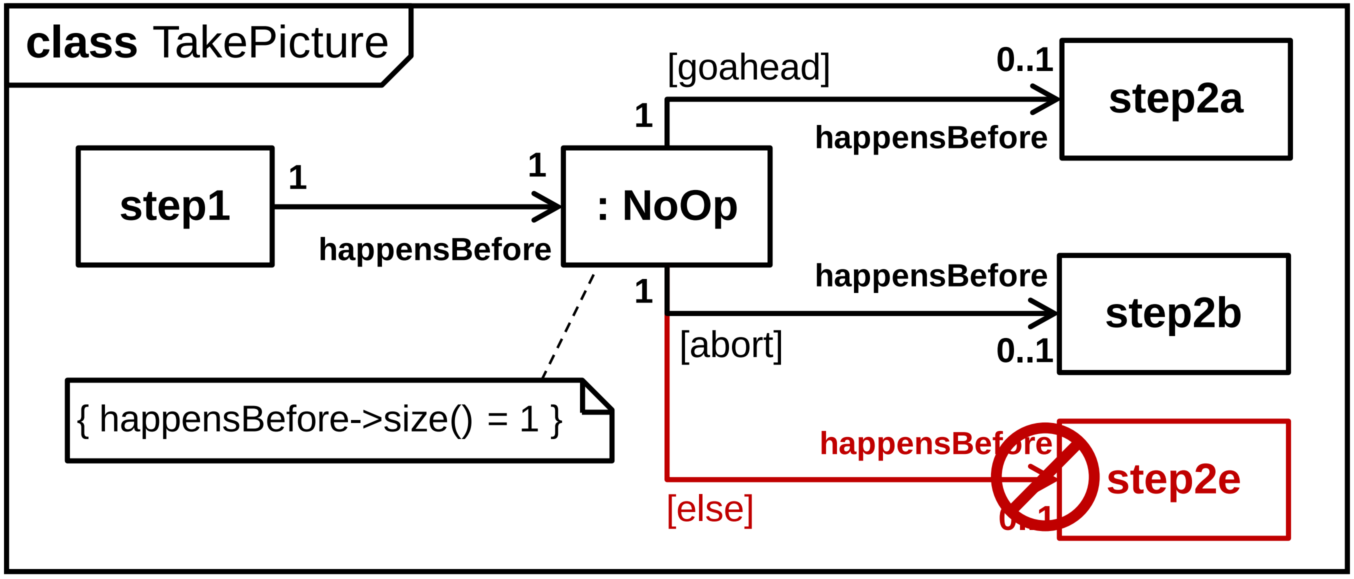
Decision Guards, Open



- **Sufficient constraints on connector values.**

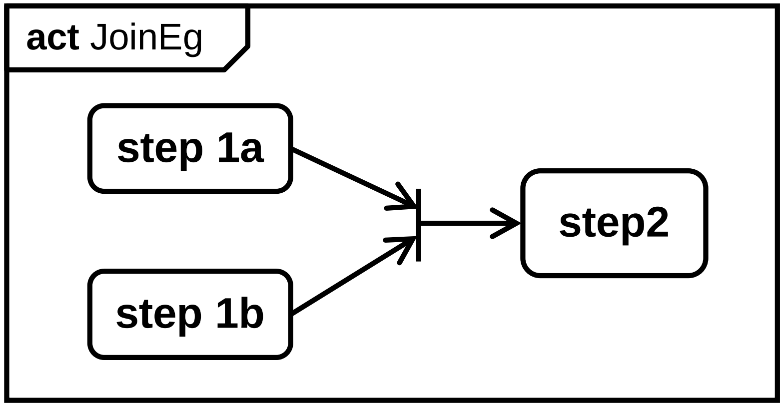
Decision Nodes, Open

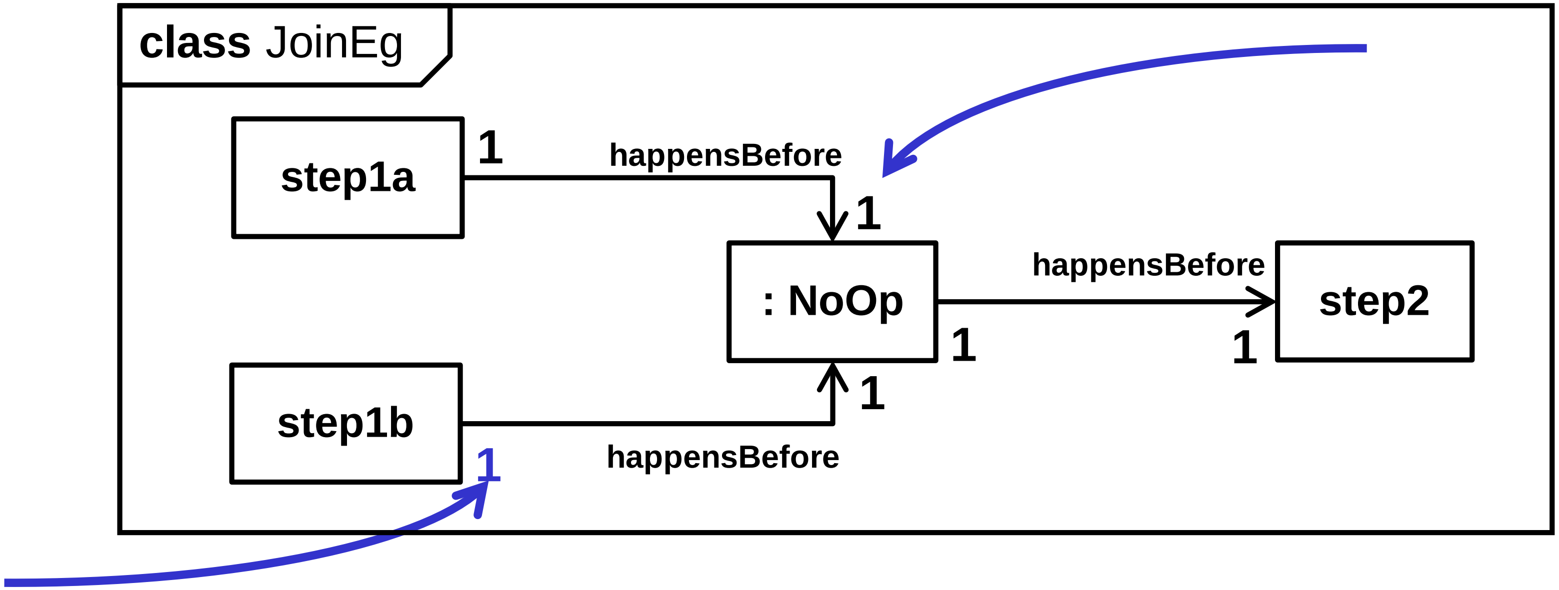




- **No else** or empty guards

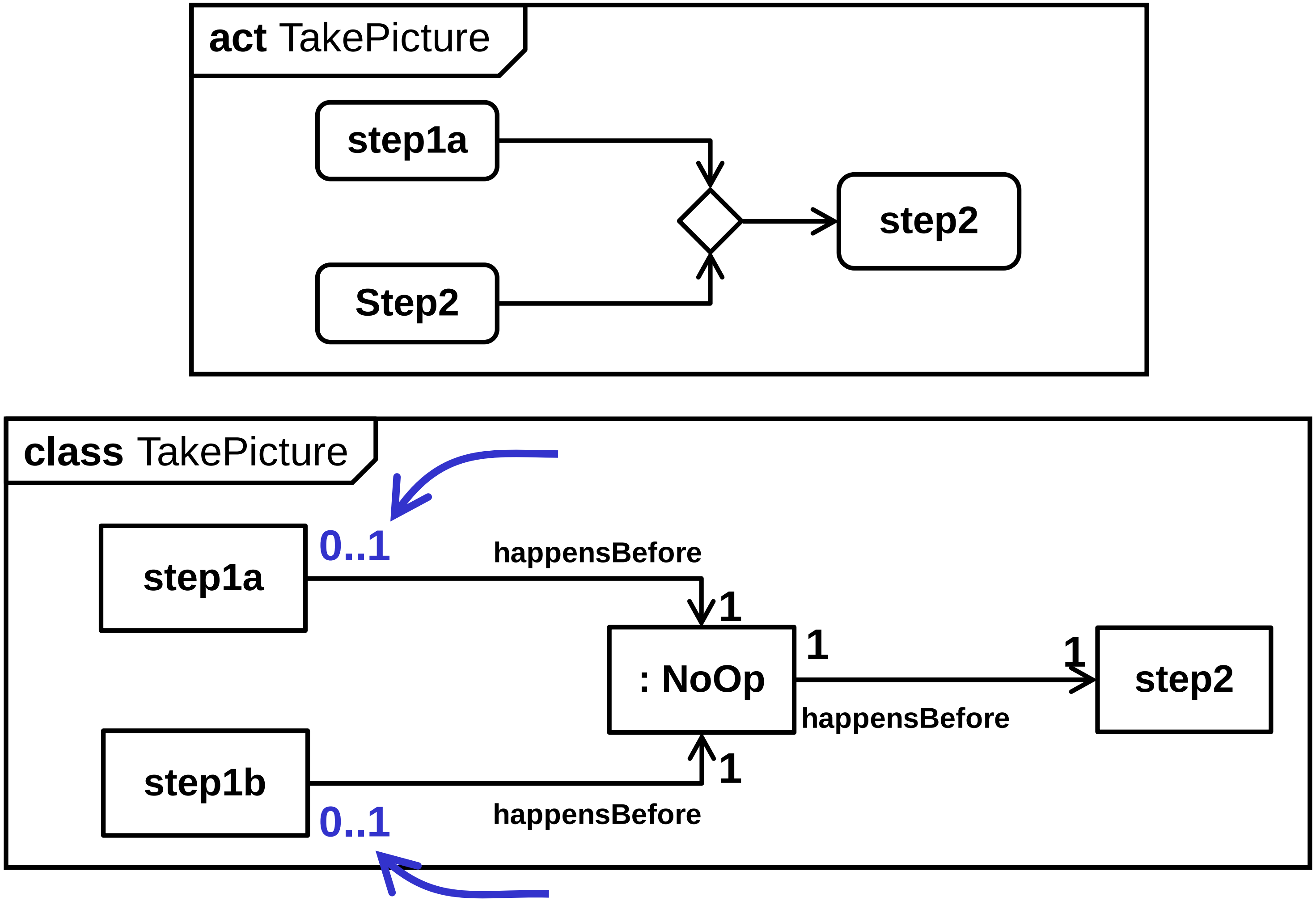
Control Nodes (Join)





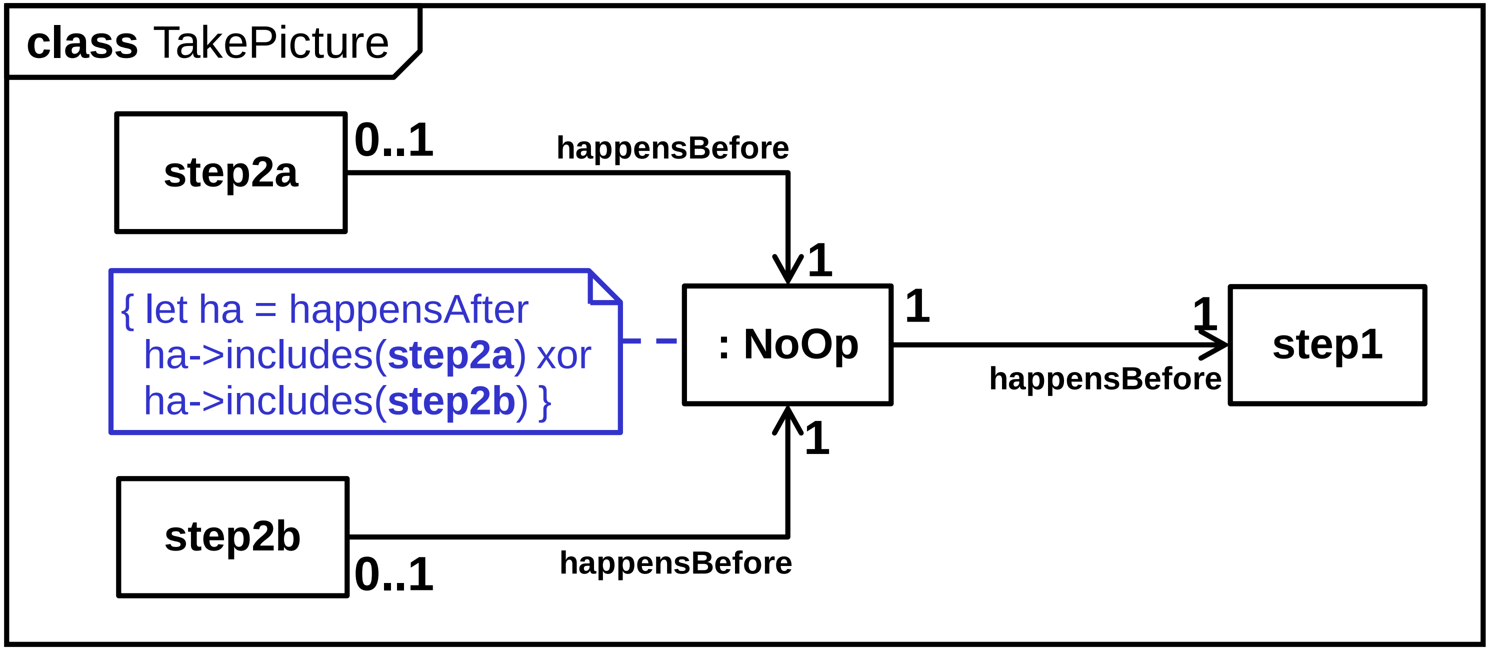
- **Reverse of fork**

Control Nodes (Merge)



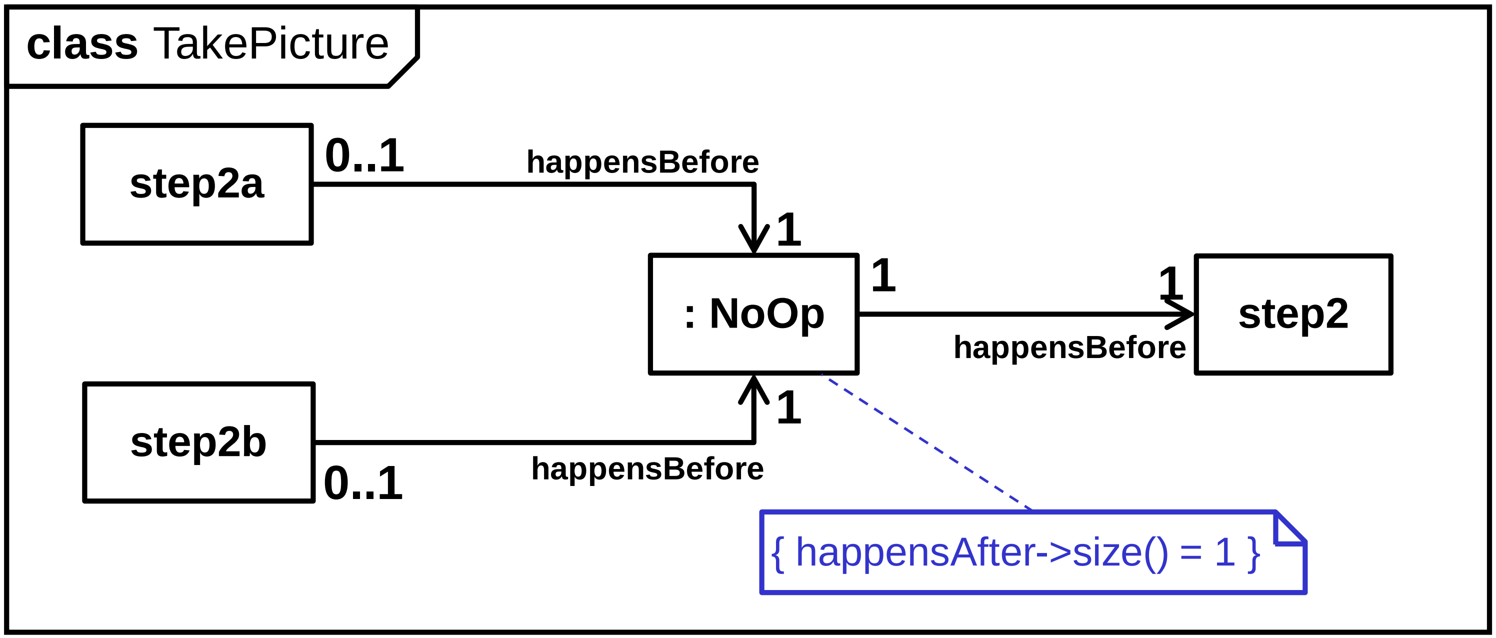
- **What ensures each merge happens due to exactly one previous step?**

Merge Nodes, Closed



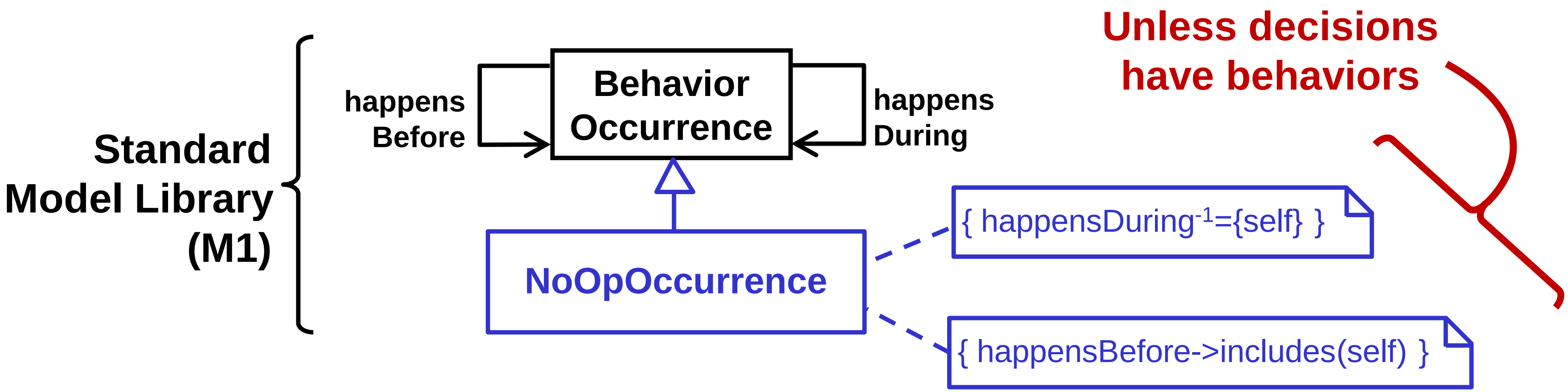
- **Pro: Each merge will happen due to exactly one of step2a or step2b.**
- **Con: Must be updated when branches change.**

Merge Nodes, Open, Not

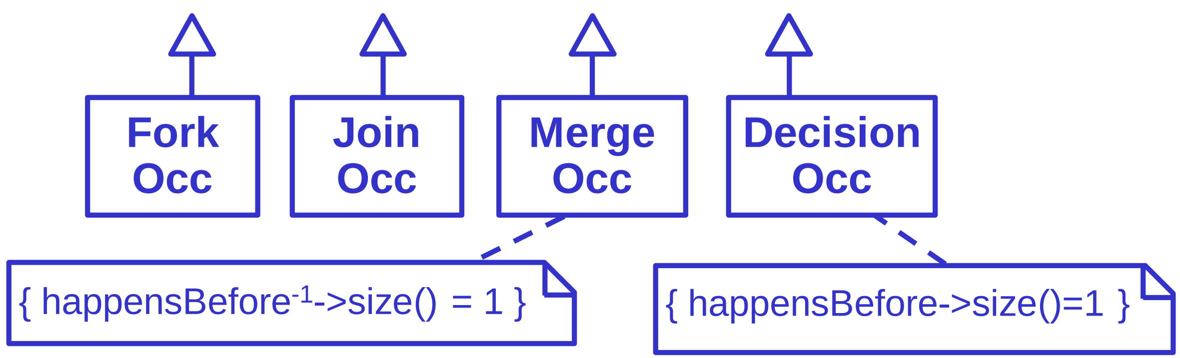


- **Pro: Same for any number of alternatives.**
- **Con: Doesn't require alternatives to happen for merge to happen.**
  - **No guards to give sufficient conditions.**

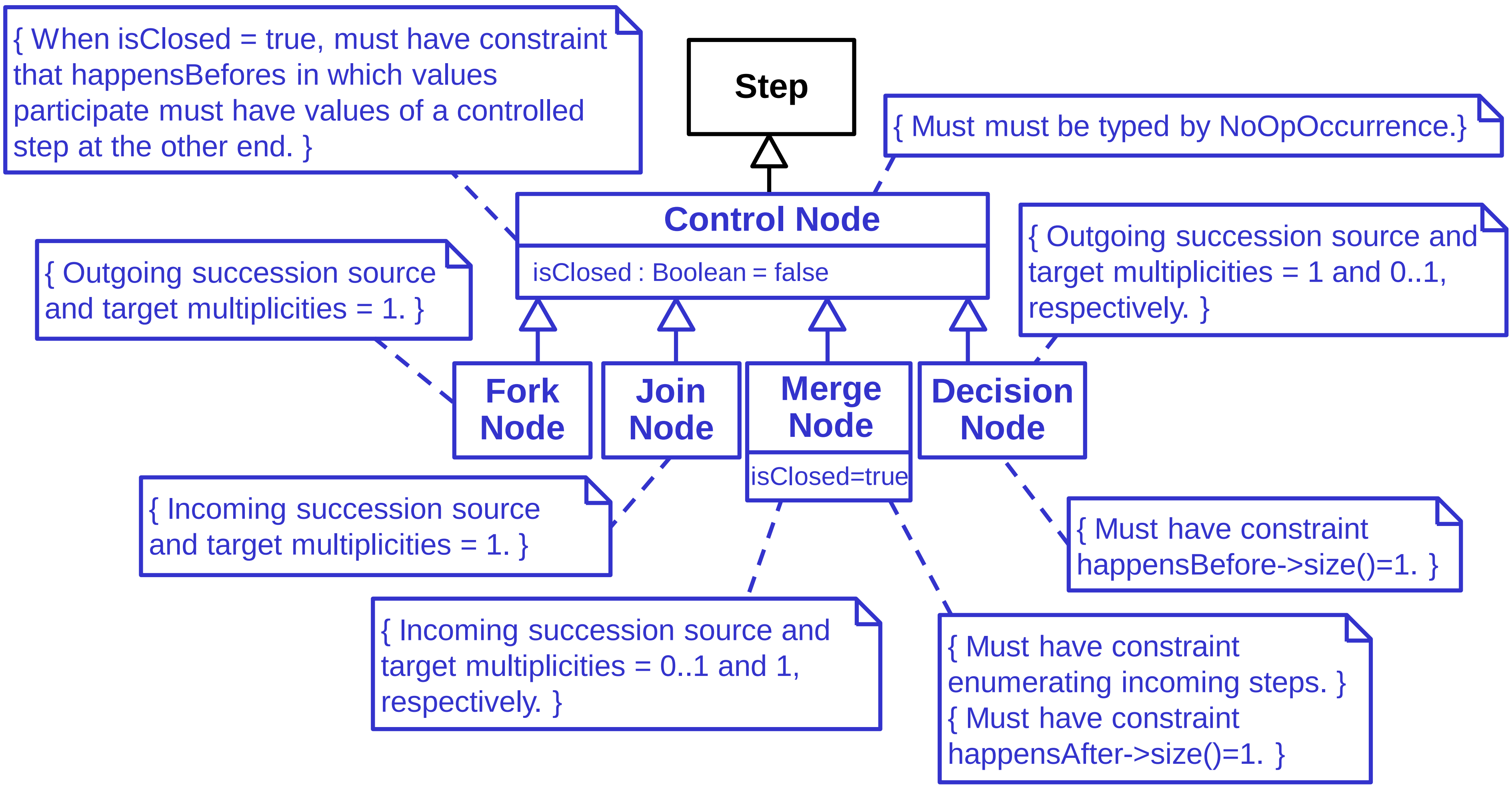
Control Nodes (M1)



- **Could include control occurrences:**



Control Nodes (M2)

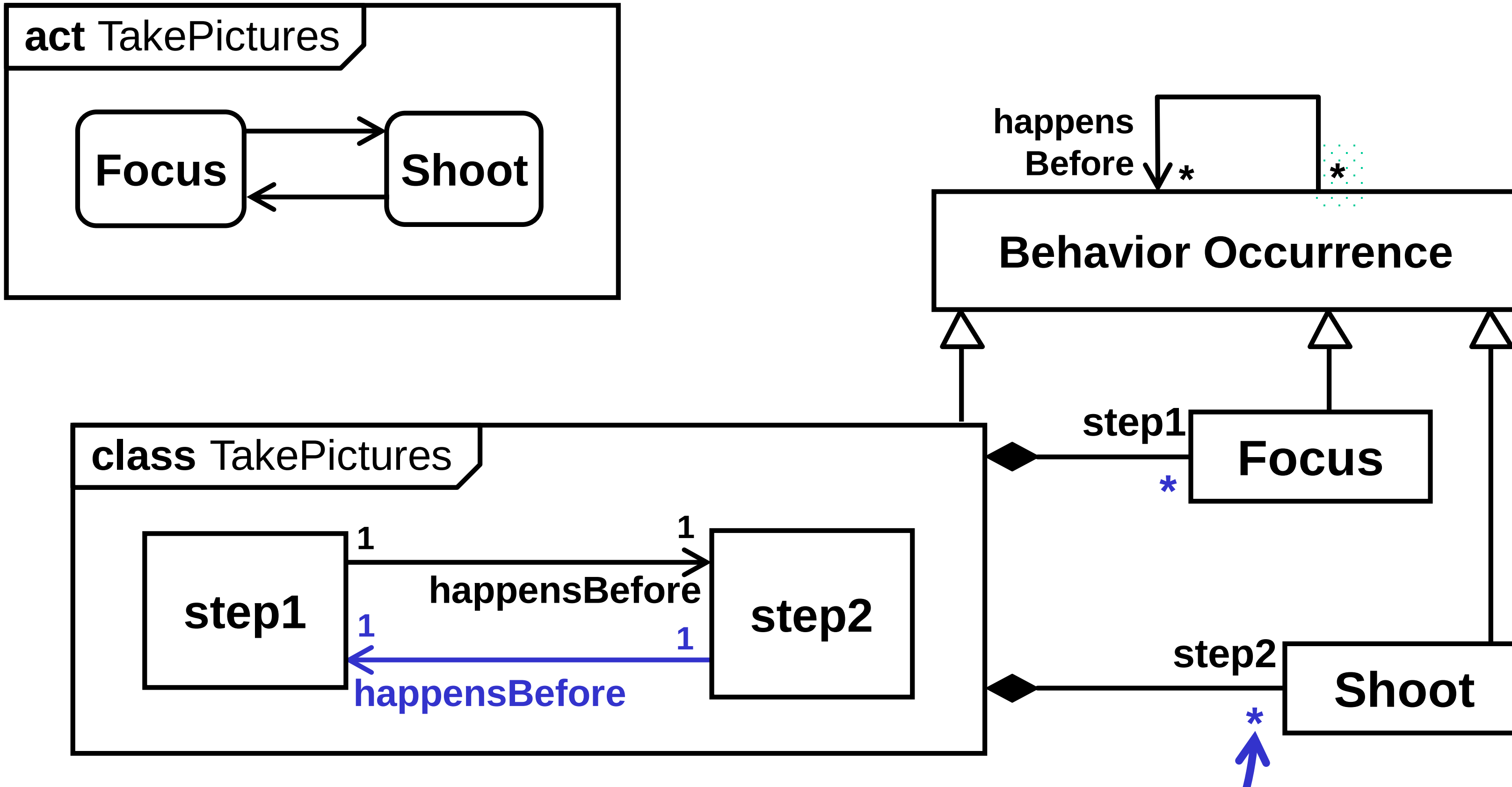


- **Define M1 patterns**
  - **Step type, connector multiplicities, M1 constraints.**

Overview

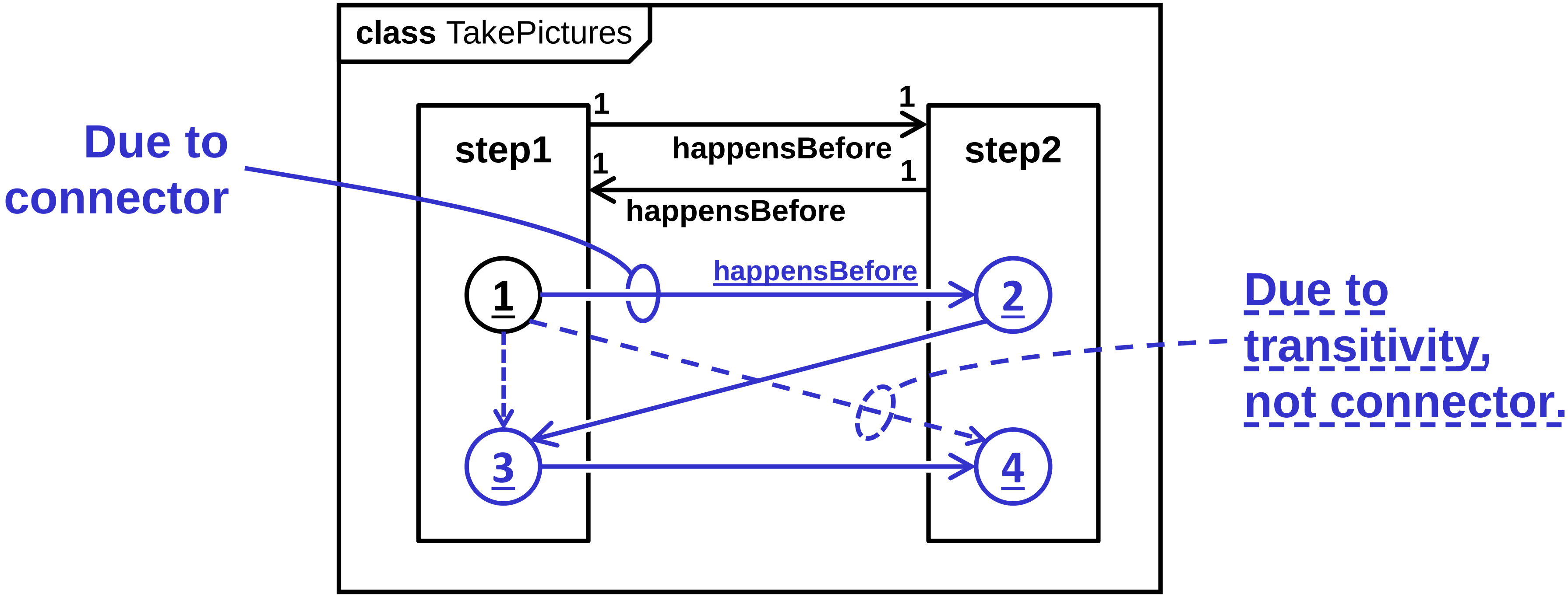
- RoadMap
- Motivation
  - Behavior, review
  - Activities, requirements
- Activities Solution
  - Control nodes
  - **Loops**
  - Specialization
- Summary

Loops



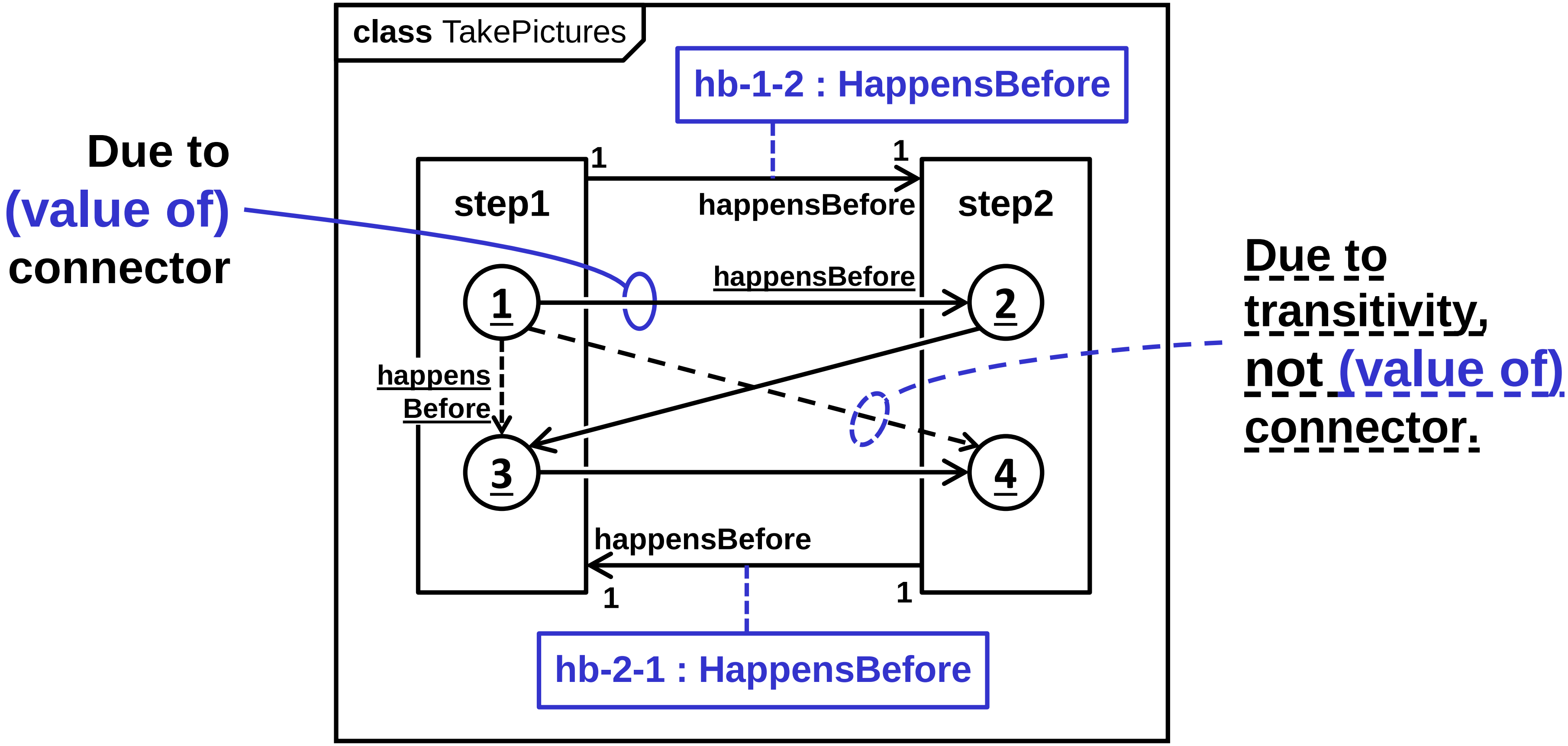
- **Multiple occurrences per step.**
  - Also applies to event-driven and streaming behaviors.

Multiple Occurrences (#1)



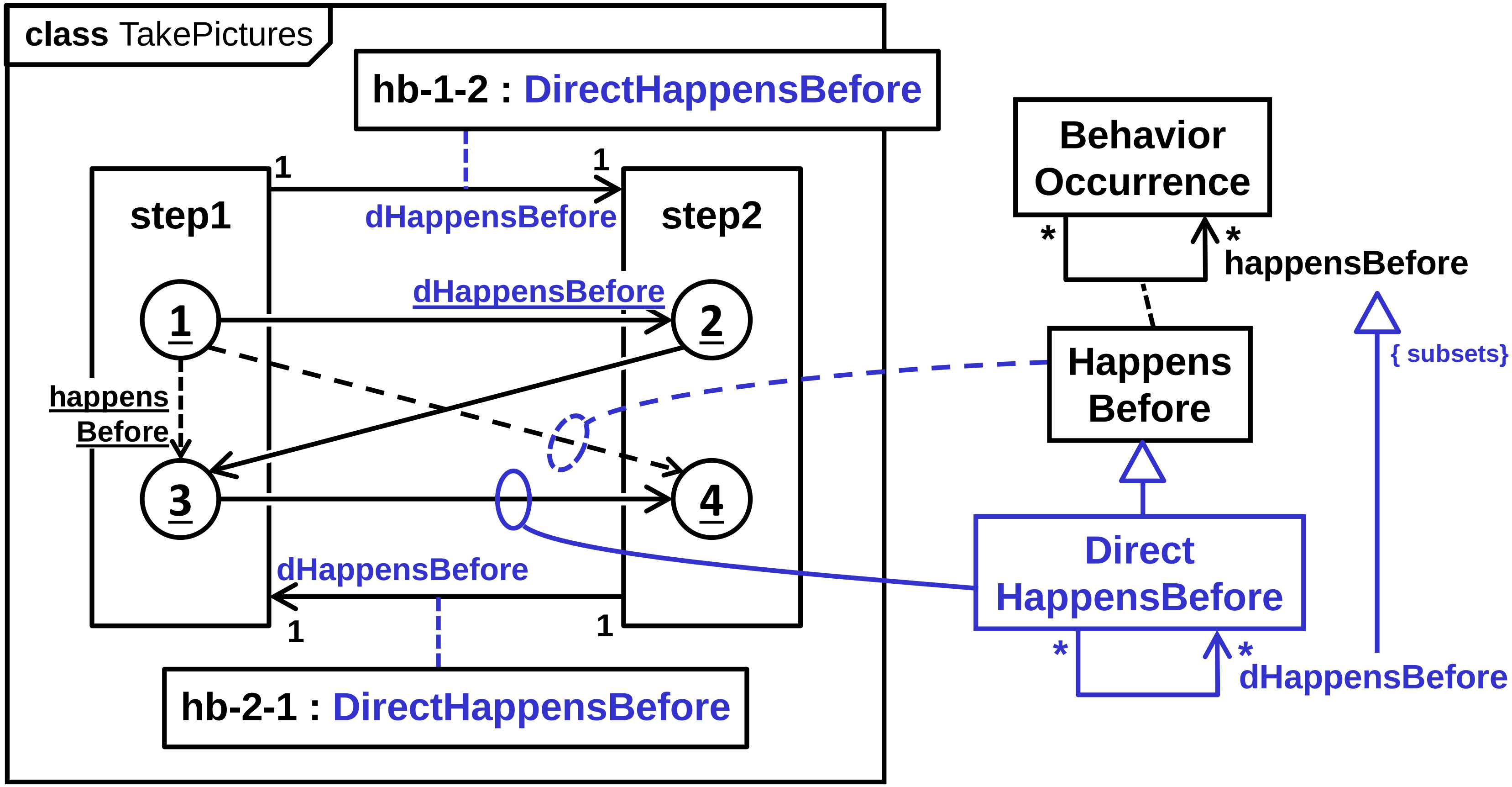
- happensBefore is **transitive** …
  - but links inferred this way are not due to connectors, and are **not counted in connector multiplicities**.

Multiple Occurrences (#1)



- **Connectors …**
  - **Are properties typed by associations.**
  - **Values are links due to connector (counted by connector multiplicities).**

Multiple Occurrences (#2)

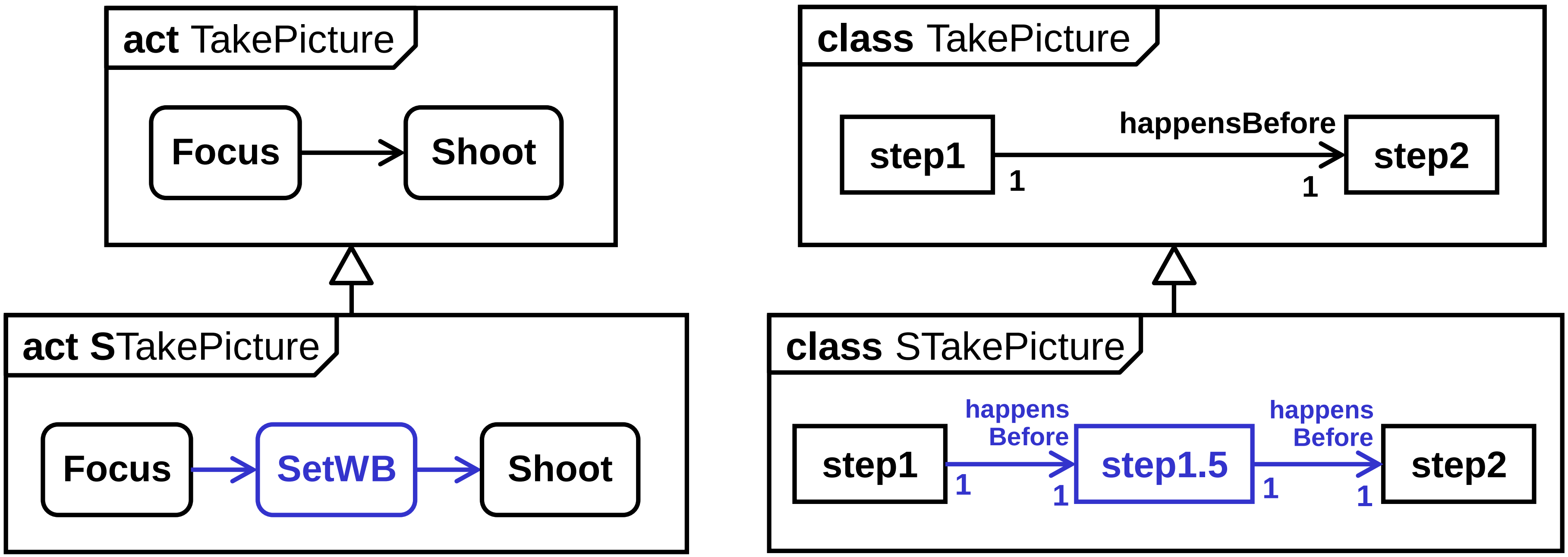


- Connectors typed by intransitive (“direct”) happens before
  - Implies (transitive) happens before
  - But not vice-versa.

Overview

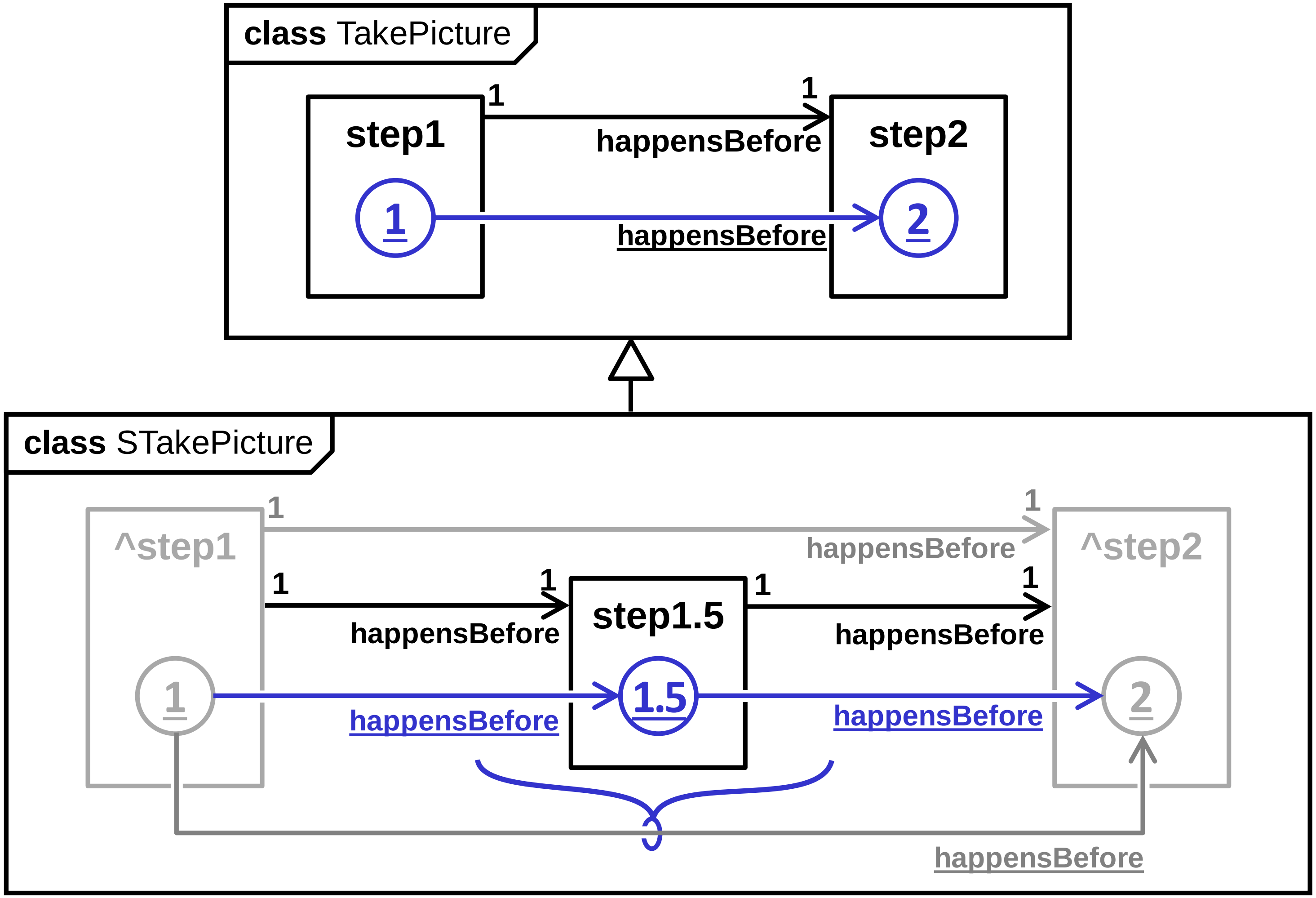
- RoadMap
- Motivation
  - Behavior, review
  - Activities, requirements
- Activities Solution
  - Control nodes
  - Loops
  - **Specialization**
- Summary

Specialization (Add'l Steps)



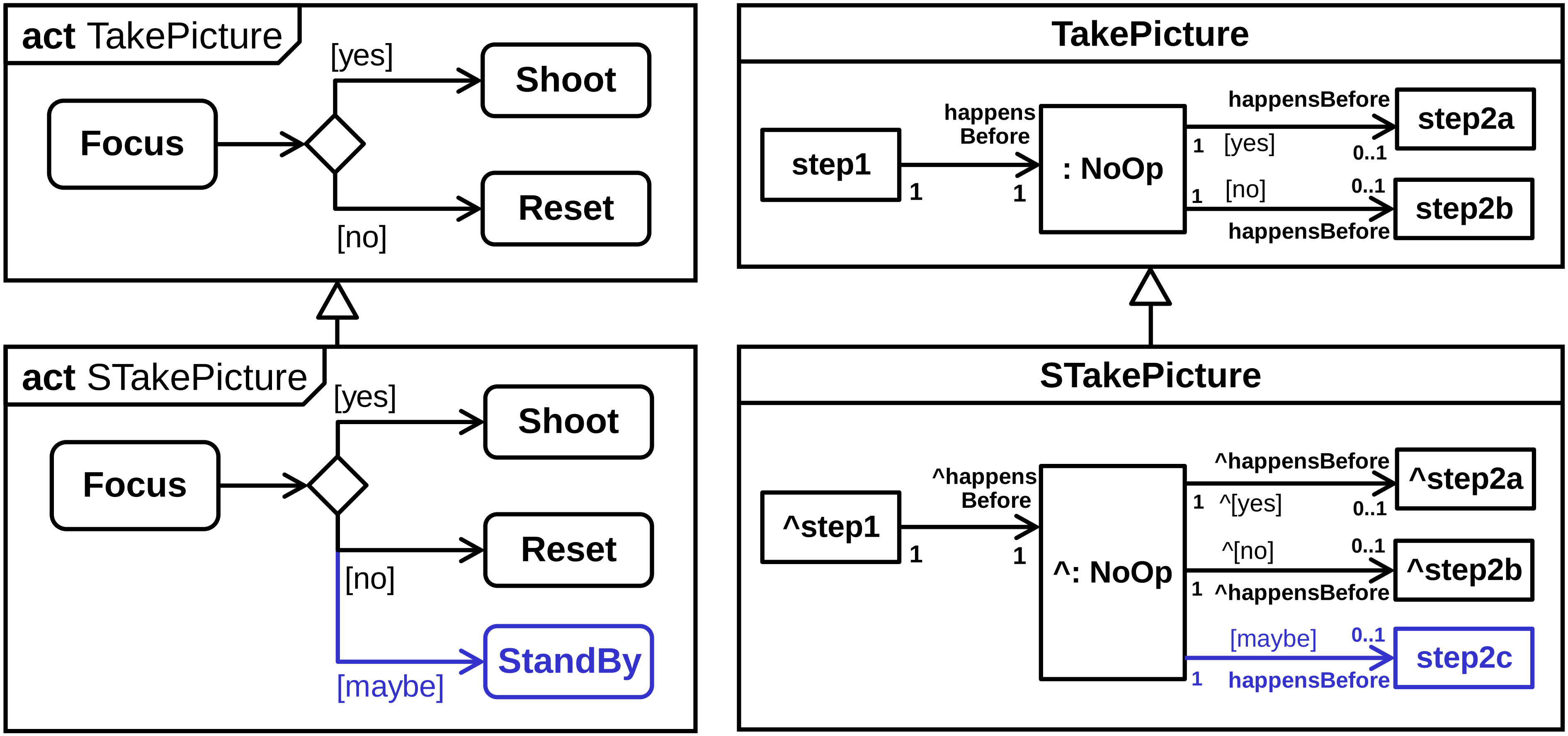
- **Do additional steps in specialized behaviors follow generalization semantics?**
  - **Sub/supersets of M0 instances**
  - **Inheriting timing constraints**

Additional Steps



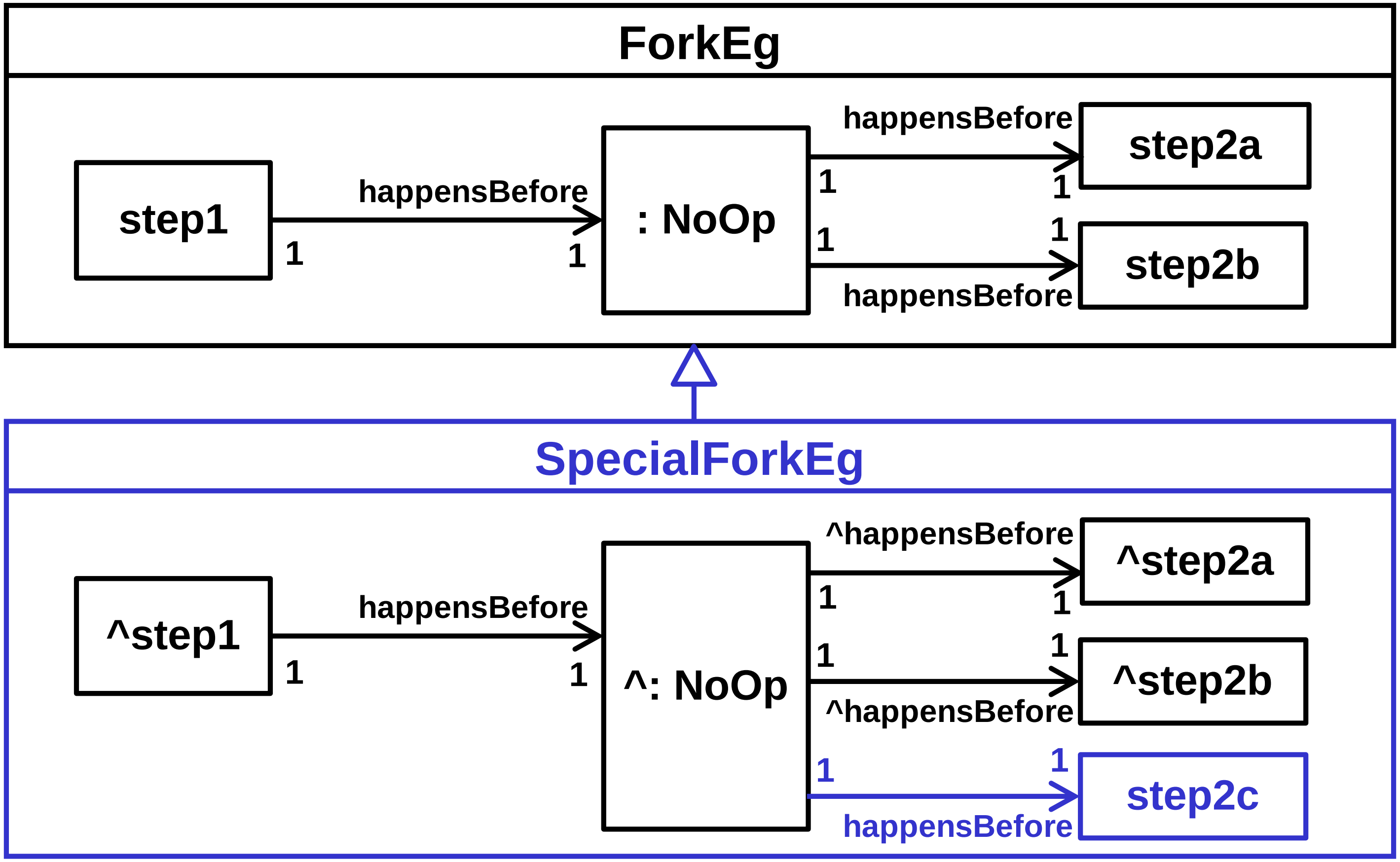
- Specialized behaviors can have additional steps
  - ✓ Executions of STakePicture perform step1 and step2
  - Multiplicities satisfied on connectors separately

Specialization (Add'l Branches)



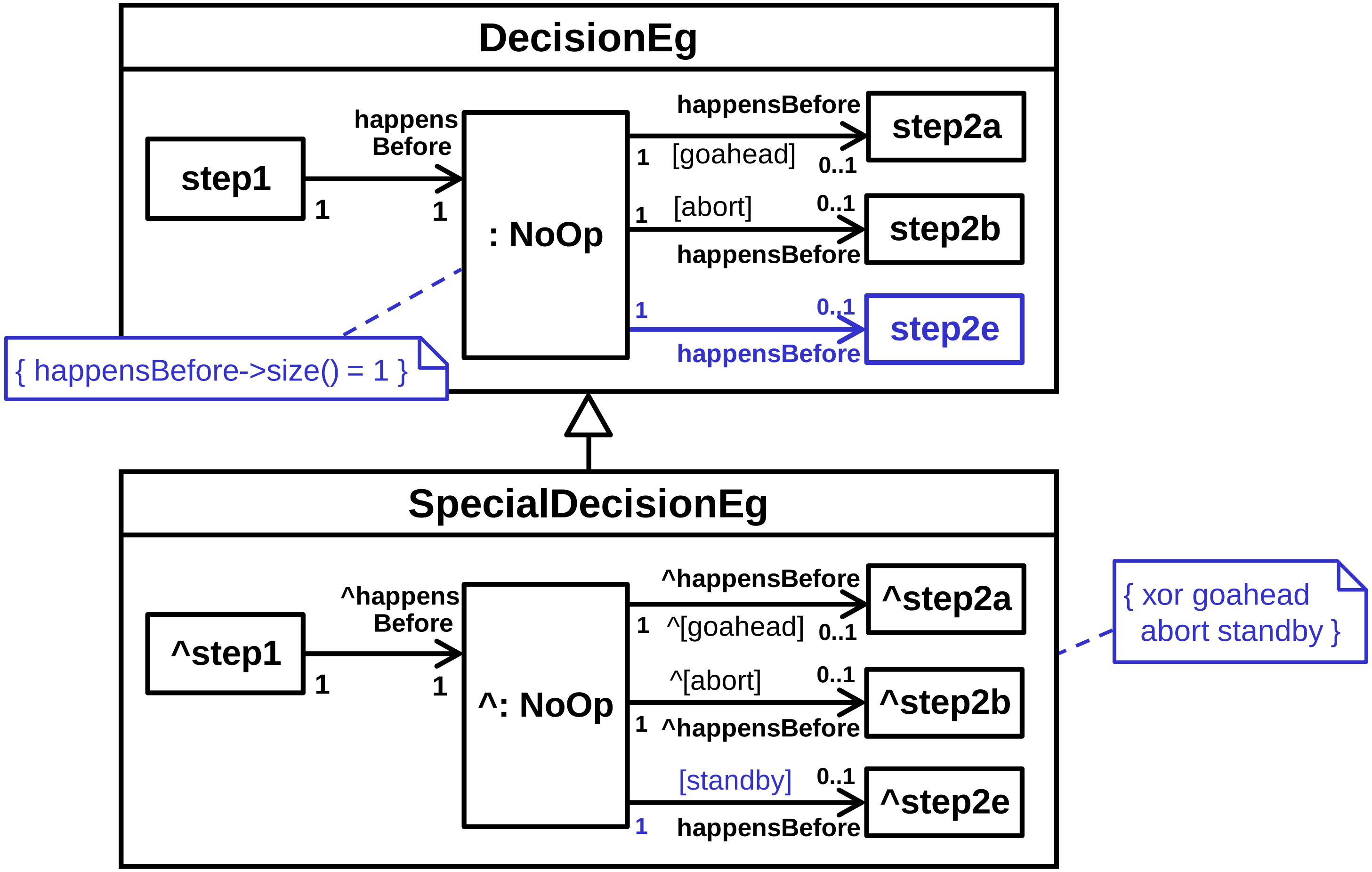
- **Do additional control node branches in specialized behaviors follow generalization semantics?**

Additional Branches (Fork)



- **Specialized behaviors can have add'l fork branches**
  - **Executions of SpecialForkEg perform step2a and step2b after each fork.**

Additional Branches (Decision)

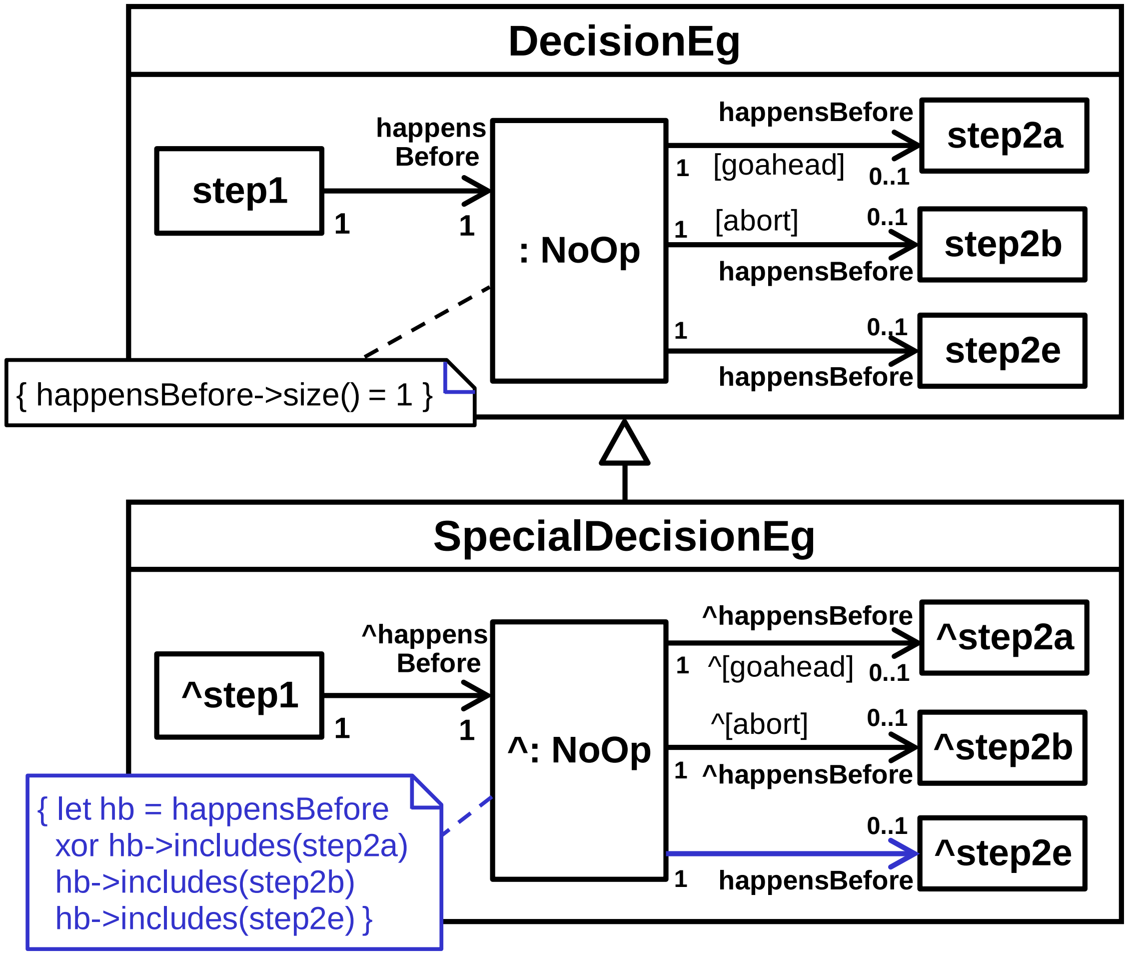


Yes, if …

~

- Generalized behavior is open.
- Specialized (leaf) behavior is closed (#1 used above)
- Generalized guards can all fail, some can be empty
- No reasoning based on generalized behavior

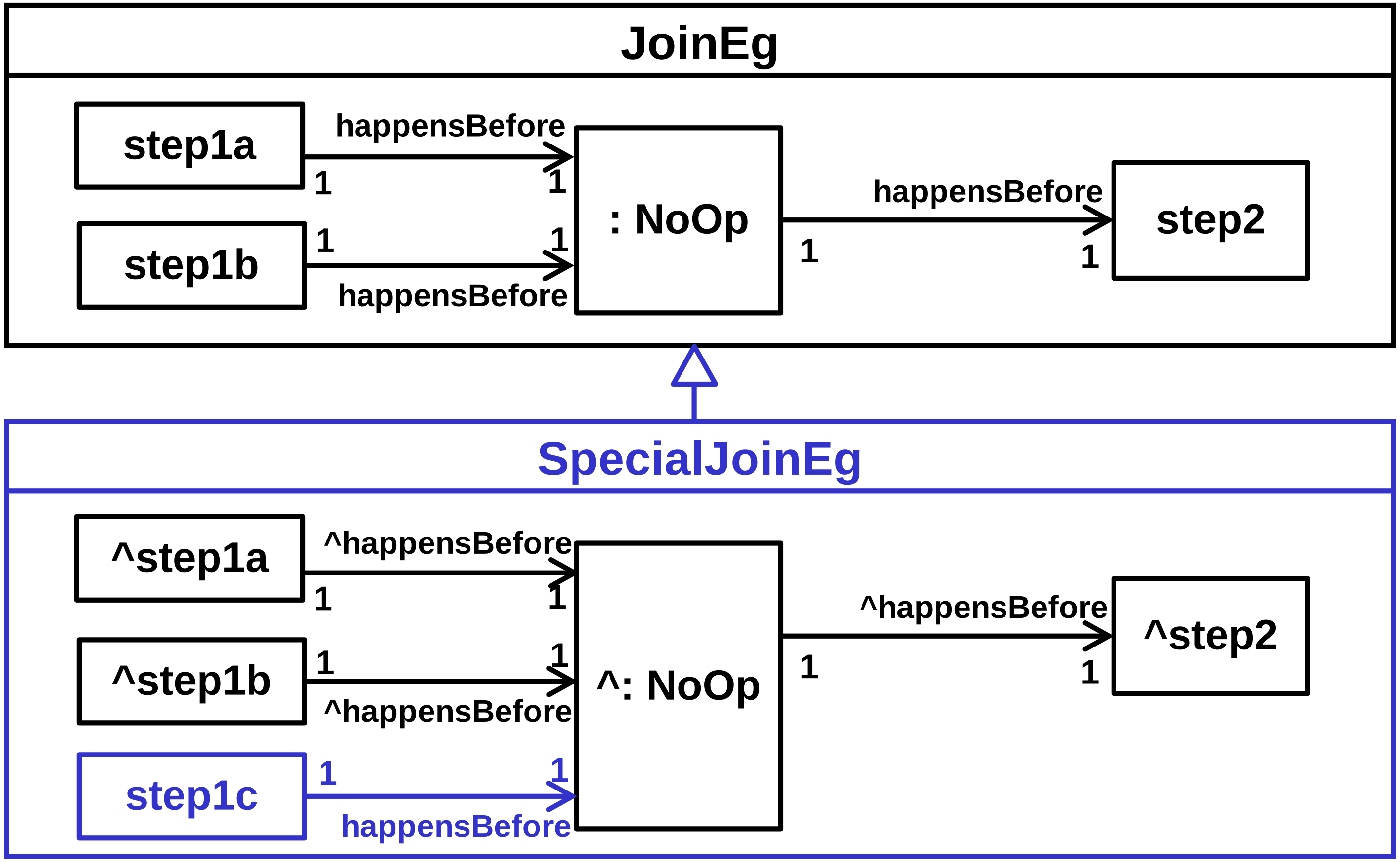
Additional Branches (Decision)



Yes, if …

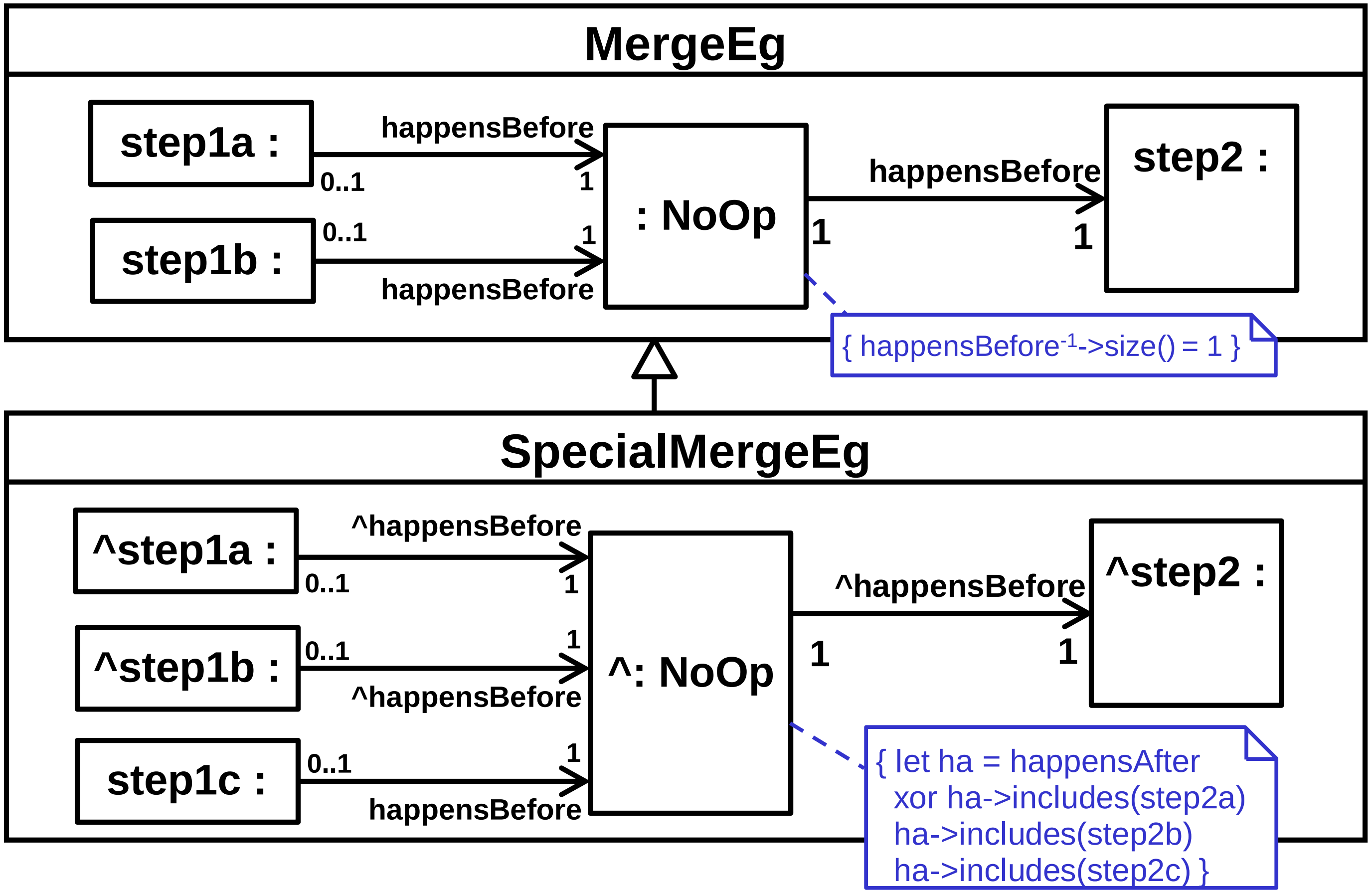
- Generalized behavior is open.
- Specialized (leaf) behavior is closed (#2 above, else)
- Generalized guards can all fail, some can be empty
- No reasoning based on generalization behavior

Additional Branches (Join)



- Specialized behaviors can have add'l join branches
  - Executions of SpecialForkEg perform step1a and step1b before each join.

Additional Branches (Merge)



Yes, if …

- Generalized behavior is open.
- Specialized behavior is closed
- No reasoning based on generalized behavior

Activity TBD

- Regions
  - Interruptable
  - Expansion
- Object Nodes / Flows
  - Queuing
  - Weight
- Exceptions

Overview

- RoadMap
- Motivation
  - Behavior, review
  - Activities, requirements
- Activities Solution
  - Control nodes
  - Loops
  - Specialization
- **Summary**

Summary

- Sequences of behaviors coordinated by:
  - – Multiplicities on HappensBefore connectors.
  - – Additional constraints for sufficiency or closure in some cases.
  - – NoOp steps (control nodes) and metamodel.
  - – HappensBefore connectors specifying links
    - Only due to connector multiplicities or
    - Intransitive ("direct") HappensBefore
- Generalization for specializing behaviors by:
  - – Multiplicities on HappensBefore connectors.
  - – Specialize open control nodes to close them.

More Information

- **Intro to Behavior as Composite Structure**
  - **http://doc.omg.org/ad/2018-03-02**
- **Interaction as Composite Structure**
  - **http://doc.omg.org/ad/18-06-11**
- **Object-orientation as Composite Structure**
  - **http://doc.omg.org/ad/18-09-07**
- **State Machines as Composite Structure, Parts 1&2**
  - **http://doc.omg.org/ad/18-12-09, http://doc.omg.org/ad/19-03-02**
- **Earlier slides (more onto, includes interactions)**
  - **http://conradbock.org/bock-ontological-behavior-modeling-jpl-slides.pdf**
- **Paper: http://dx.doi.org/10.5381/jot.2011.10.1.a3**
- **Application to BPMN: http://conradbock.org/#BPDM**
- **KerML/SysML2: Contact Chas Galey charles.e.galey@lmco.com**