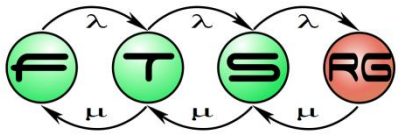
Model Transformation Lab

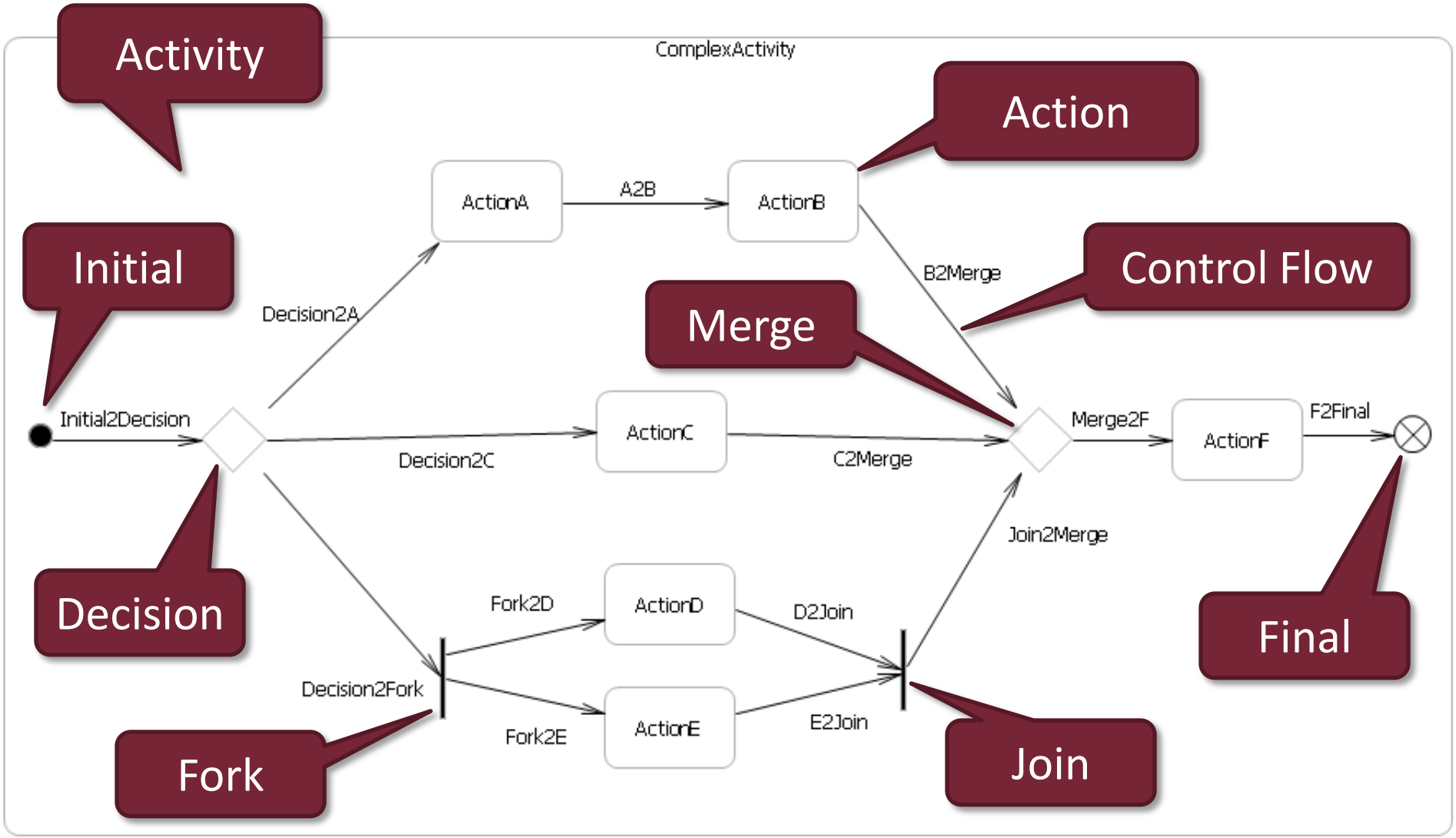
From UML Activities to Petri nets by VIATRA2



Transformation

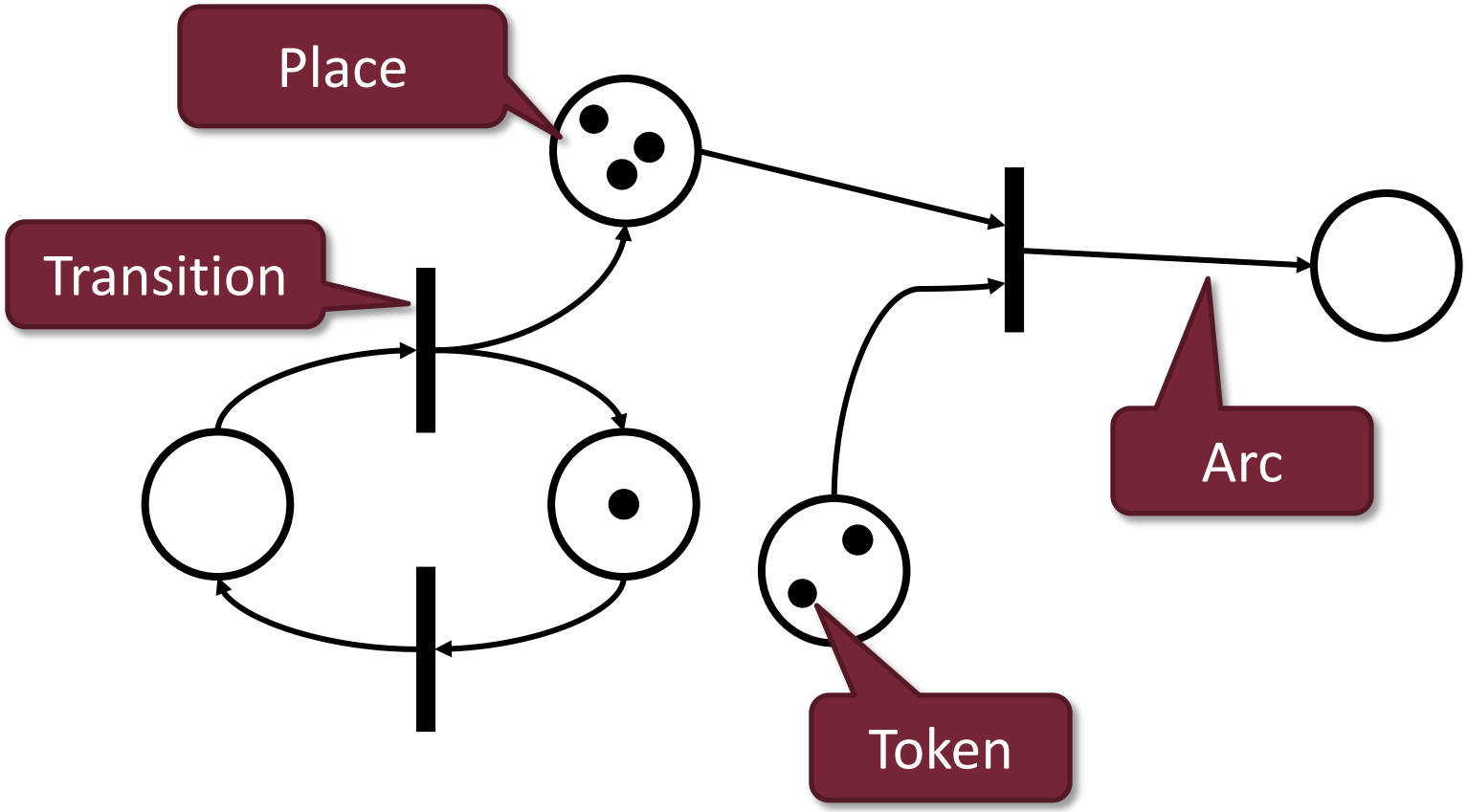
- Source domain: **UML Activities** (core elements)
  - standard process description language
  - countless editors available
- Target domain: **Petri nets** (basic elements)
  - mathematical formalism
  - concurrent behavioural model
  - efficient analysis tools available

UML Activities (core)



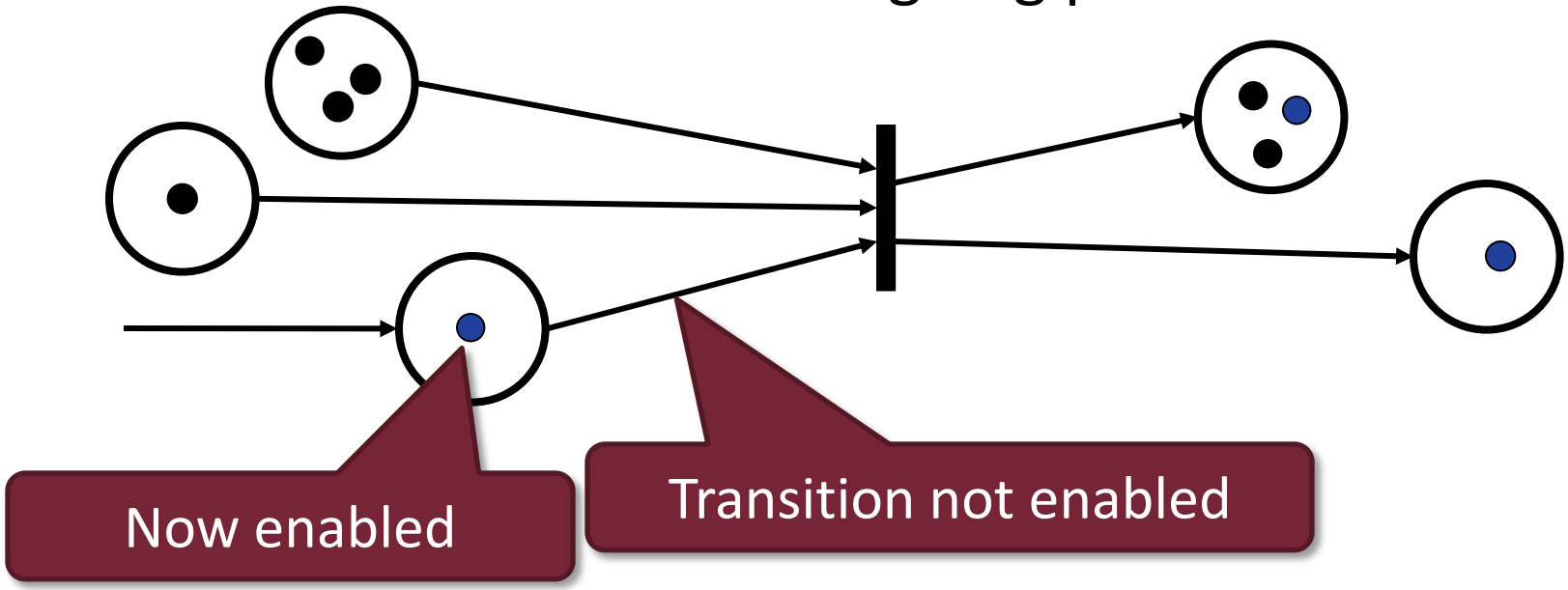
Petri nets

- AKA Place/Transition Nets



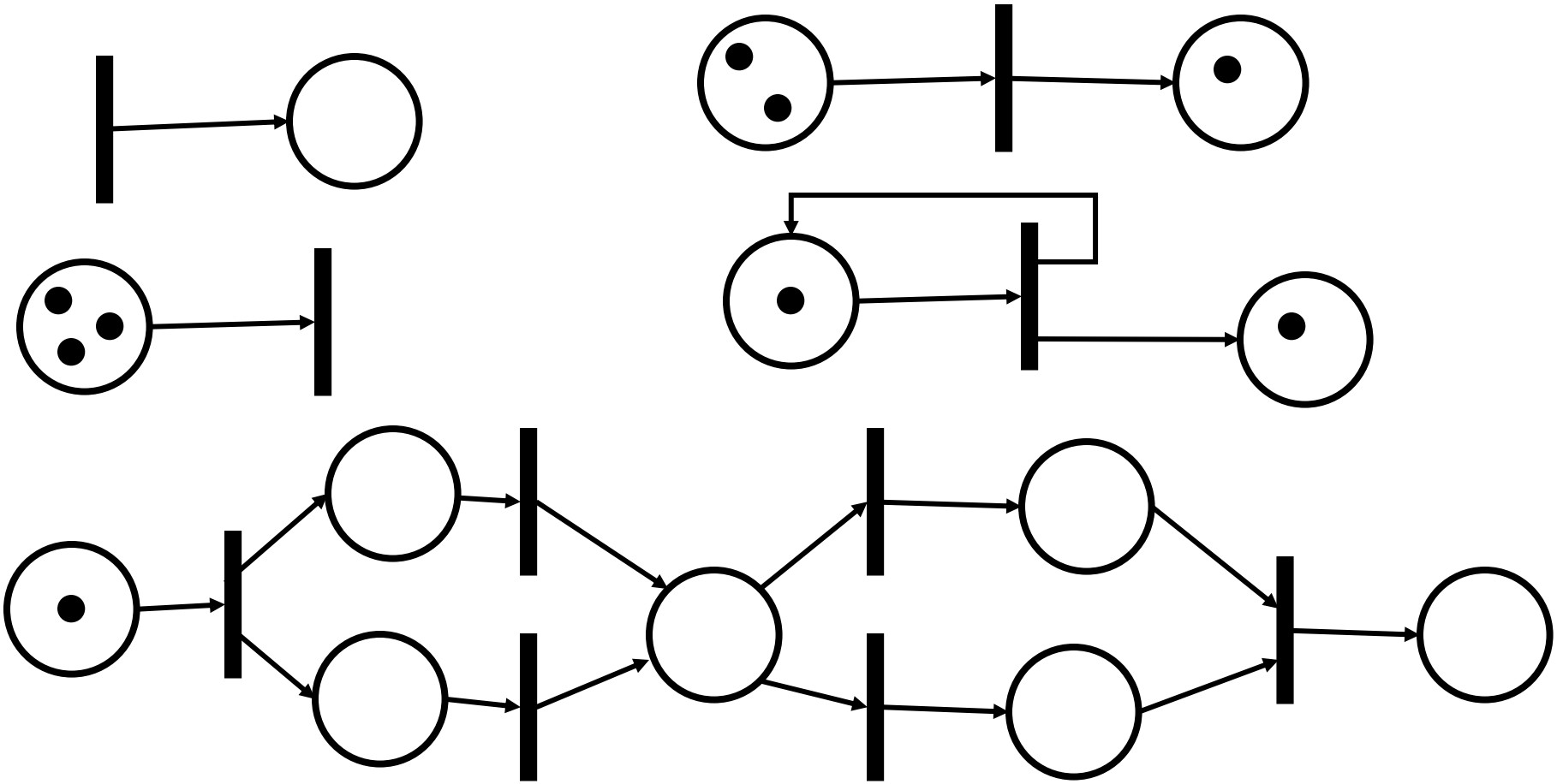
Petri nets

- State = marking of places
- State change: firing of a transition
  - *enabled* if all incoming places are *marked*
  - 1 token removed from each incoming place
  - 1 token added to each outgoing place

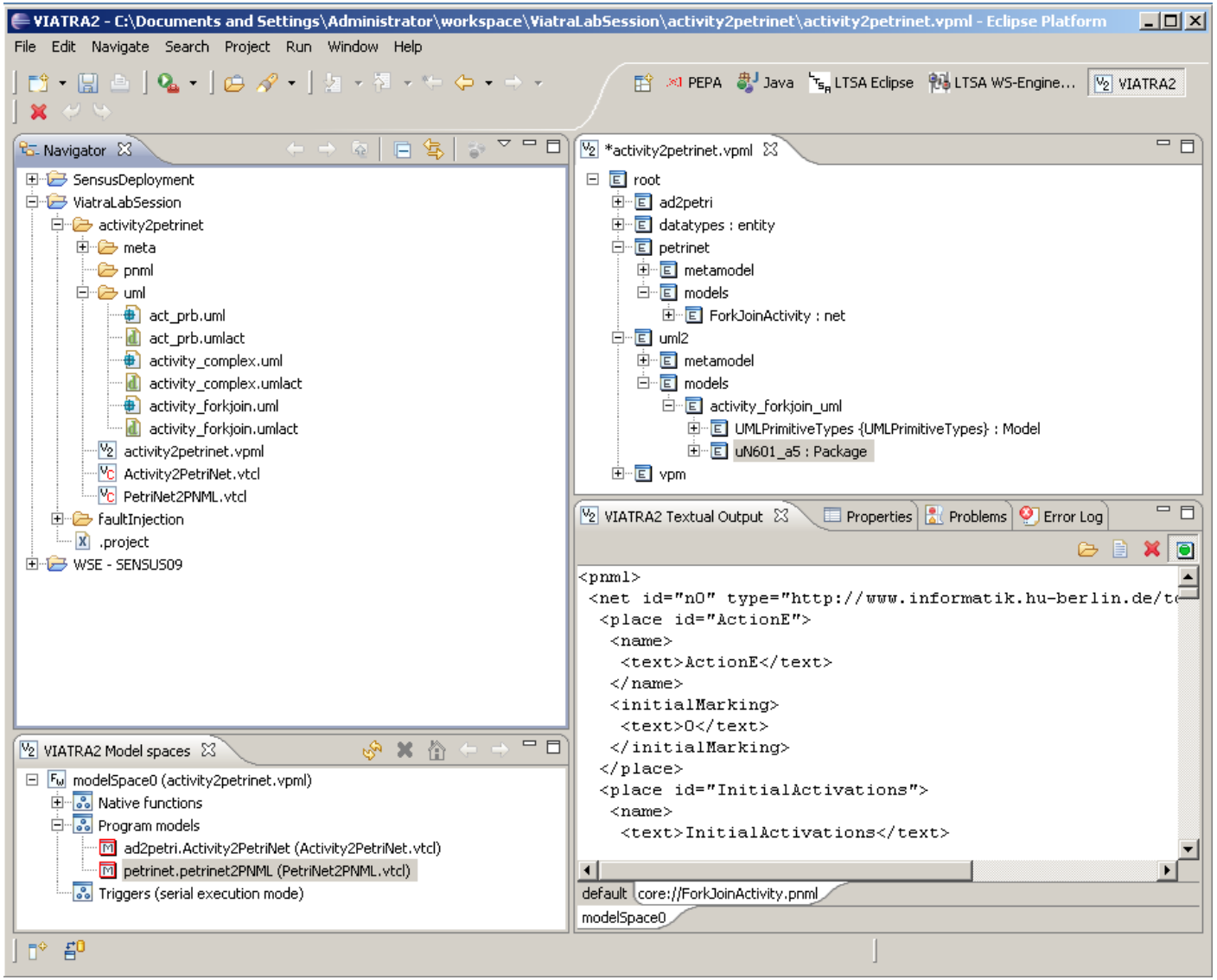


Petri nets

- What do these Petri nets do?

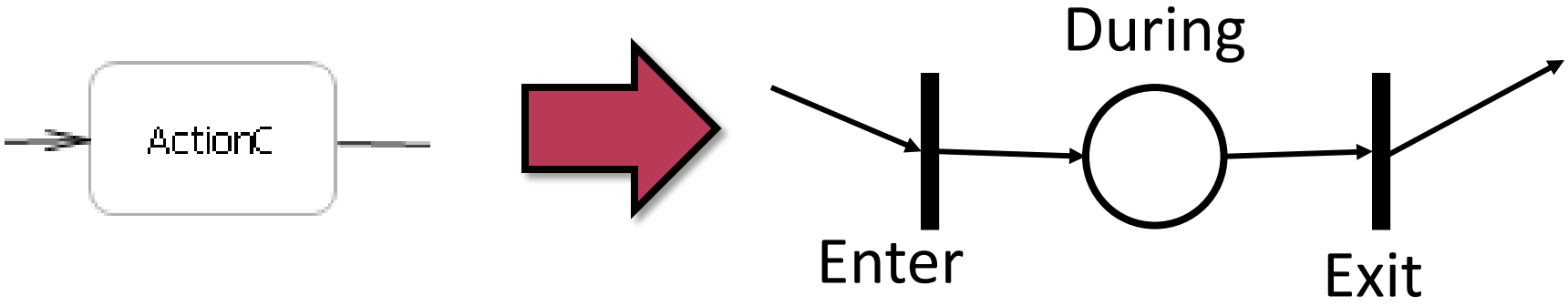


LAB Executing the transformation

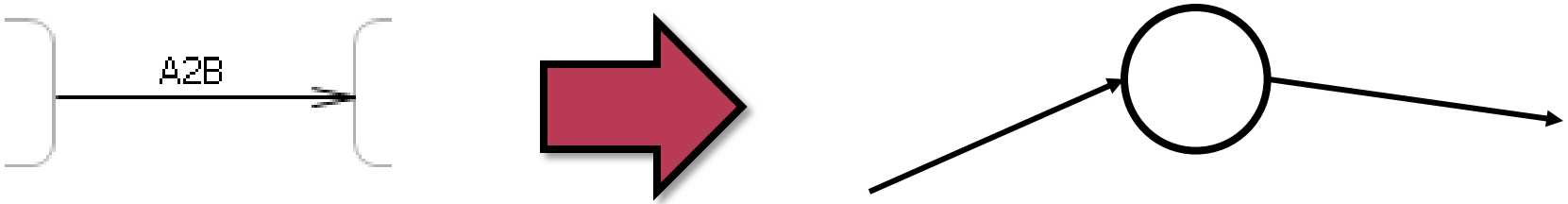


UML Activity → Petri net

- Action

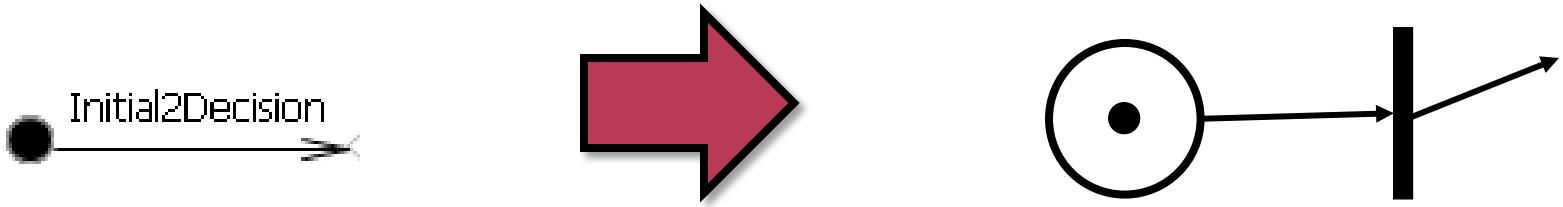


- Control flow

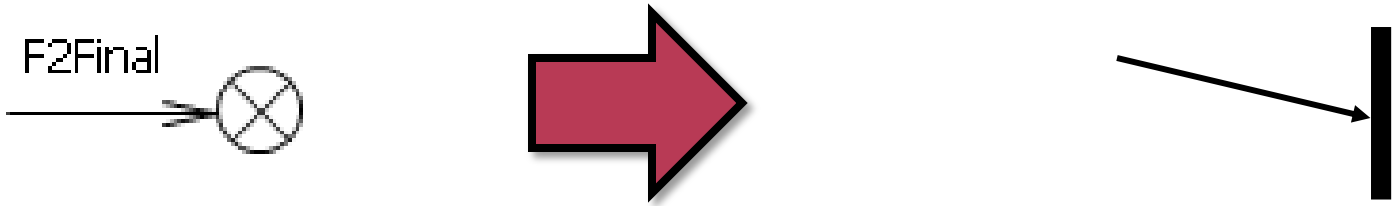


UML Activity → Petri net

- Initial node

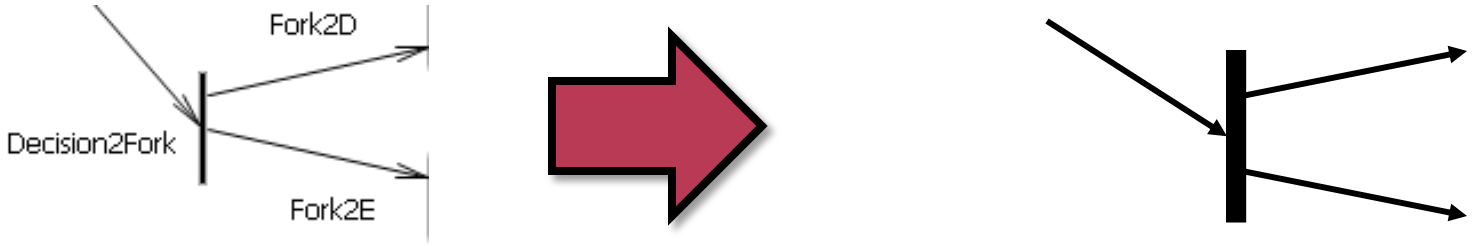


- (Flow) final node

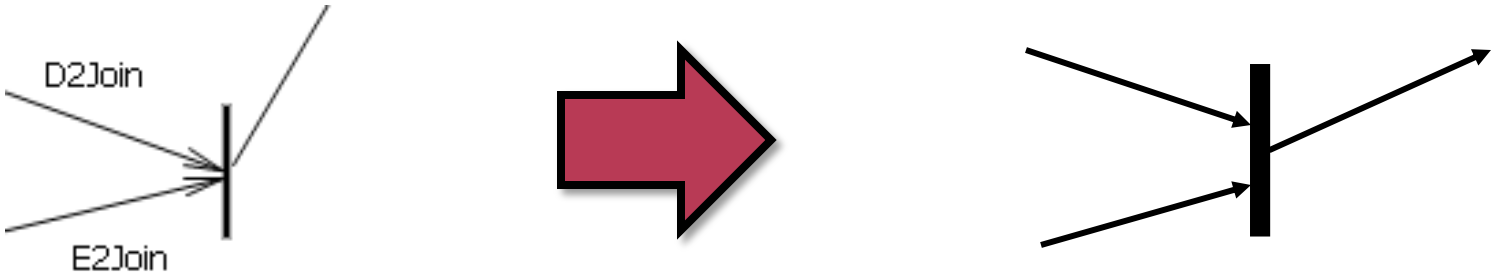


UML Activity → Petri net

- Fork node

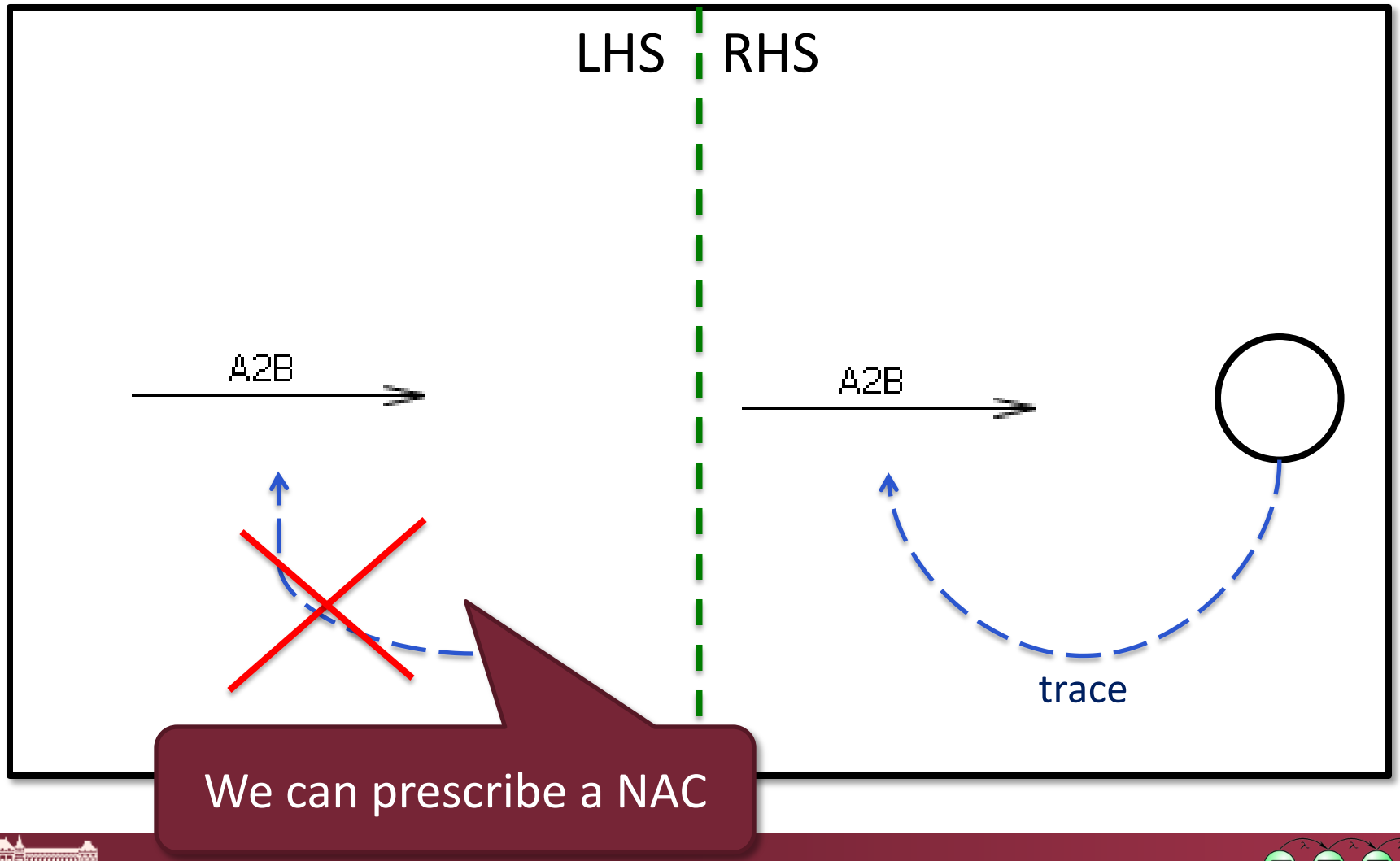


- Join node

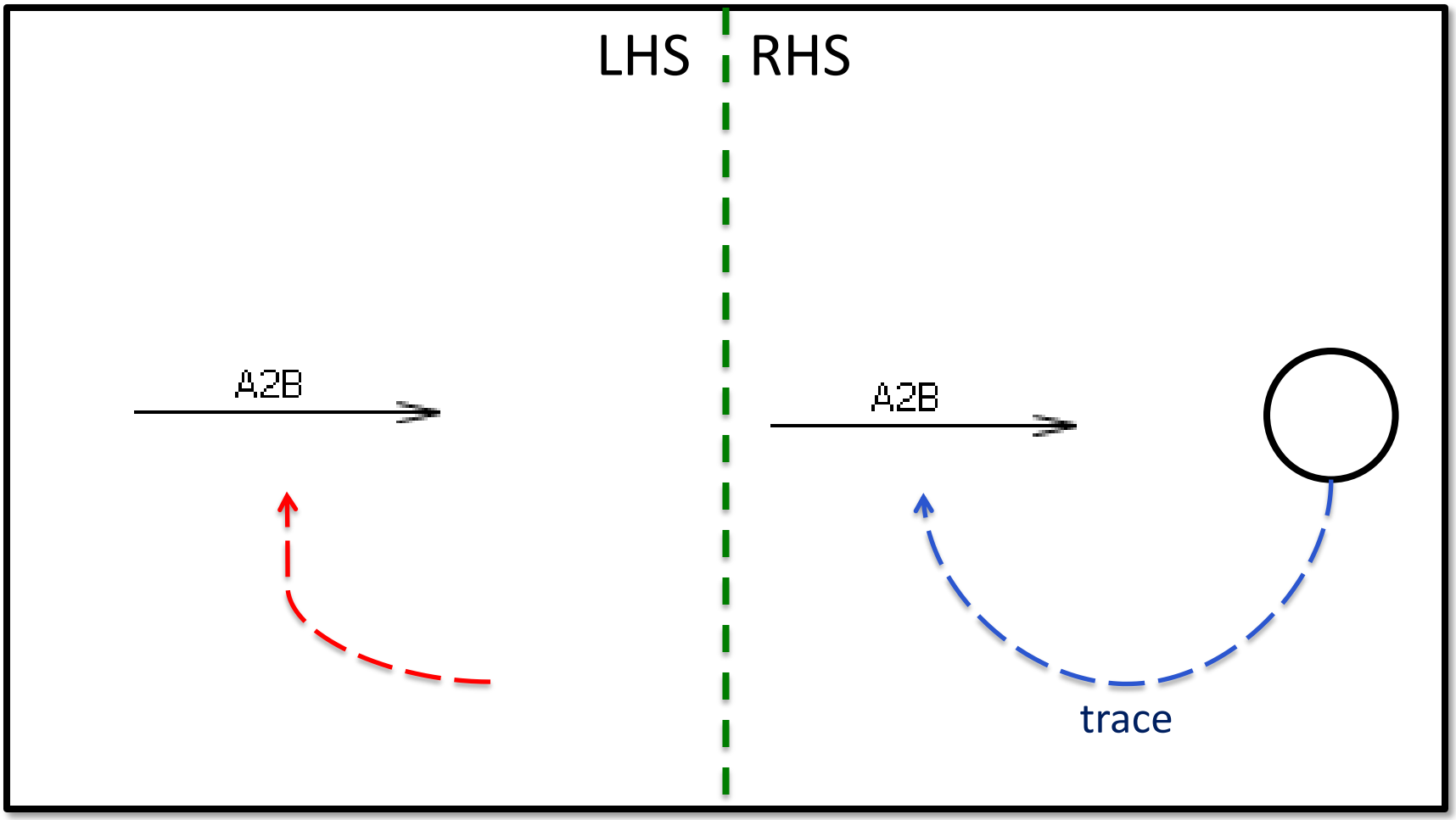


GT rules – Control Flow

- Mostly straightforward

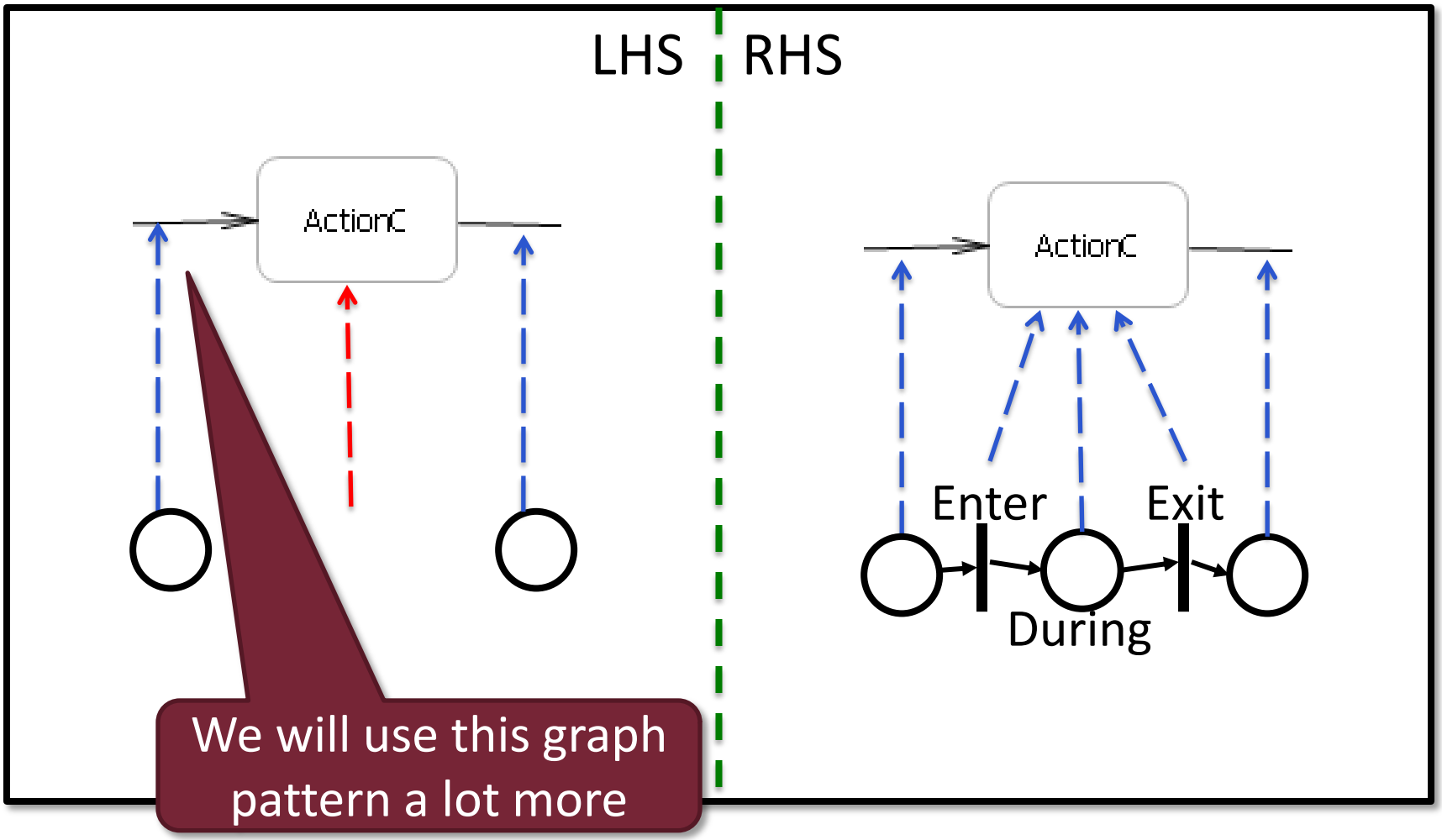


DEMO Simple GT rules in Viatra2



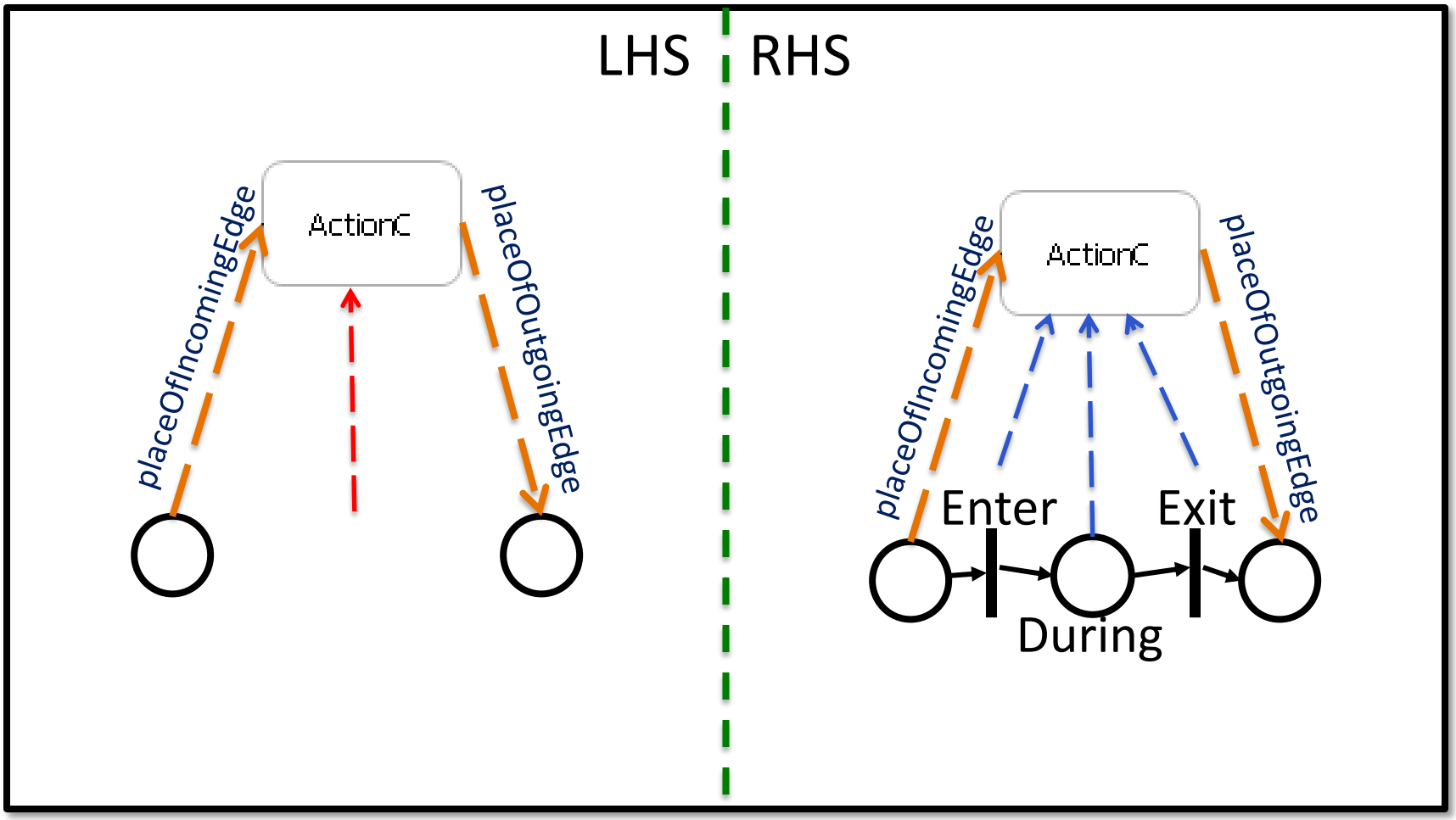
GT rules – Action

- Straightforward, but big



GT rules – reusing patterns

- Let's reuse!



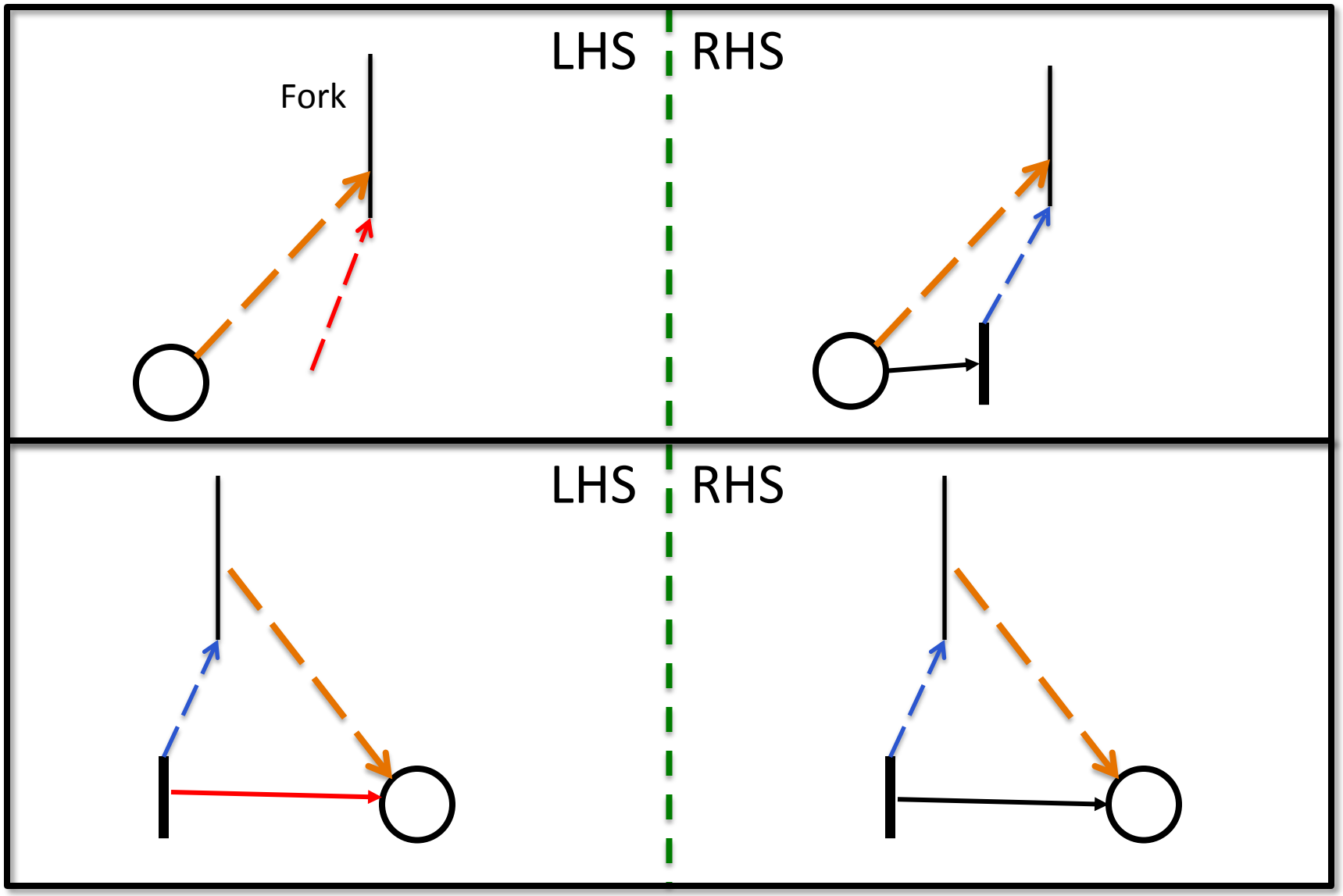
CODE Pattern composition in Viatra2

```
pattern placeOfIncomingEdge(ActivityNode, PetriPlace) = {
        'ActivityNode'(ActivityNode);
        'ActivityNode'.incoming(InComing, ActivityNode, ActivityEdge);
        'ActivityEdge'(ActivityEdge);
        place.placeTraceEdge(Trace, PetriPlace, ActivityEdge);
        place(PetriPlace);
}
pattern placeOfOutgoingEdge(ActivityNode, PetriPlace) = {
        'ActivityNode'(ActivityNode);
        'ActivityNode'.outgoing(OutGoing, ActivityNode, ActivityEdge);
        'ActivityEdge'(ActivityEdge);
        place.placeTraceEdge(Trace, PetriPlace, ActivityEdge);
        place(PetriPlace);
}
```

CODE Pattern composition in Viatra2

```
gtrule transformExecutableNode(out ActivityNode, in PetriNet) = {
    precondition pattern unmappedExecutableNode(ActivityNode, IncomingEdgePlace, OutgoingEdgePlace) = {
        'ExecutableNode'(ActivityNode);
        find placeOfIncomingEdge(ActivityNode, IncomingEdgePlace);
        find placeOfOutgoingEdge(ActivityNode, OutgoingEdgePlace);
        neg find activityNodeTransitionMapping(ActivityNode, NoPetriTransition);
    }
    postcondition pattern mappedExecutableNode(ActivityNode, IncomingEdgePlace, PetriTransitionEnter,
      PetriPlaceDuring, PetriTransitionExit, OutgoingEdgePlace, PetriNet) = {
        'ExecutableNode'(ActivityNode);

        find placeTransitionArc(IncomingEdgePlace, PetriTransitionEnter);

        find activityNodeTransitionMapping(ActivityNode, PetriTransitionEnter);
        find transitionOfNet(PetriTransitionEnter, PetriNet);

        find transitionPlaceArc(PetriTransitionEnter, PetriPlaceDuring);

        find activityNodePlaceMapping(ActivityNode, PetriPlaceDuring);
        find placeOfNet(PetriPlaceDuring, PetriNet);

        find placeTransitionArc(PetriPlaceDuring, PetriTransitionExit);

        find activityNodeTransitionMapping(ActivityNode, PetriTransitionExit);
        find transitionOfNet(PetriTransitionExit, PetriNet);

        find transitionPlaceArc(PetriTransitionExit, OutgoingEdgePlace);
    }
}
```

GT rules - Fork



CODE Cooperating rules in Viatra2

```
gtrule transformForkNode(out ActivityNode, in PetriNet) = {
    precondition pattern unmappedForkNode(ActivityNode, IncomingEdgePlace) = {
        'ForkNode'(ActivityNode);
        find placeOfIncomingEdge(ActivityNode, IncomingEdgePlace);
        find actionMapping(ActivityNode, PetriTransition);
        find transitionPetriNet(PetriTransition, PetriNet);
        find placeTransitionIncomingEdgePlace(IncomingEdgePlace, PetriTransition);
    }
    action {
        call copyName(ActivityNode, PetriTransition);
        forall EdgePlace with apply
            connectNodeToOutgoing(ActivityNode, PetriTransition, EdgePlace)
                do skip;
    }
}
```

```
gtrule connectNodeToOutgoing(in ActivityNode, in PetriTransition, out EdgePlace) = {
    precondition find placeOfOutgoingEdge(ActivityNode, EdgePlace)
    postcondition find transitionPlaceArc(PetriTransition, EdgePlace)
}
```

Similarities

- Fork Node
  - Create transition, trace back to fork node
  - Connect the *placeOfIncomingEdge* to transition
  - Connect transition to each *placeOfOutgoingEdge*
- Join Node
  - Create transition, trace back to join node
  - Connect each *placeOfIncomingEdge* to transition
  - Connect transition to the *placeOfOutgoingEdge*
- Flow Final Node
  - Create transition, trace back to final node
  - Connect the *placeOfIncomingEdge* to transition
  - Connect transition to each *placeOfOutgoingEdge* (there is none, this does nothing)

Similarities

- Fork Node
  - Create transition, trace back to fork node
  - Connect **the** *placeOfIncomingEdge* to transition
  - Connect transition to **each** *placeOfOutgoingEdge*
- Join Node
  - Create transition, trace back to join node
  - Connect **each** *placeOfIncomingEdge* to transition
  - Connect transition to **the** *placeOfOutgoingEdge*
- Flow Final Node
  - Create transition, trace back to final node
  - Connect **the** *placeOfIncomingEdge* to transition
  - Connect transition to **each** *placeOfOutgoingEdge* (there is none, this does nothing)

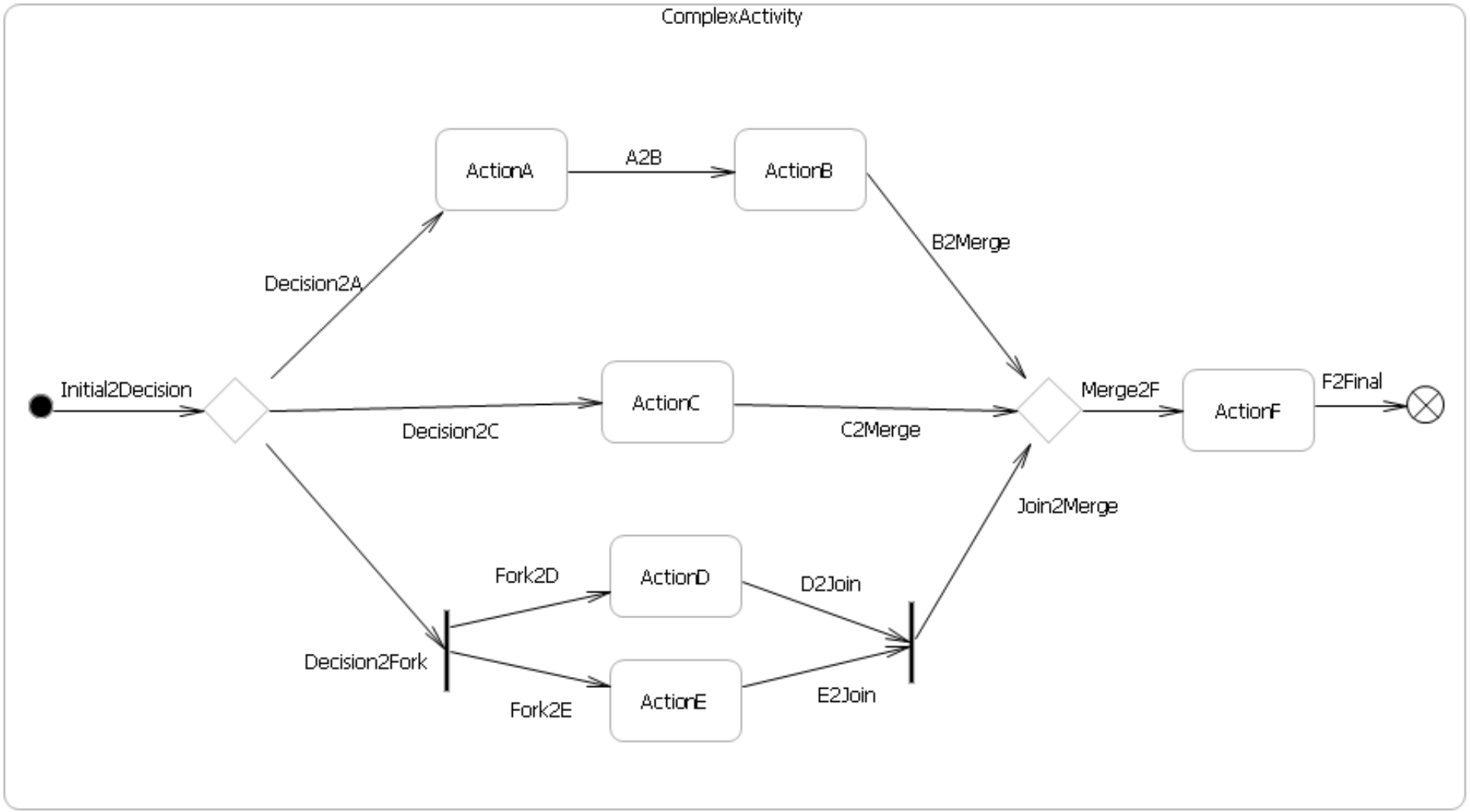
Trick

- Fork / Join / Final Node
  - Create transition, trace back to activity node
  - Connect **each** *placeOfIncomingEdge* to transition
  - Connect transtition to **each** *placeOfOutgoingEdge*
- Let’s reuse!

LAB Transformation program

```
gtrule transformForkJoinFinalNode(out ActivityNode, in PetriNet) = {
    precondition pattern unmappedForkJoinFinalNode(ActivityNode) = {
        ad2petri.helpermetamodel.forkJoinFinalNode(ActivityNode);
        neg find activityNodeTransitionMapping(ActivityNode, NoPetriTransition);
    }
    postcondition pattern mappedForkJoinFinalNode(ActivityNode, PetriTransition, PetriNet) = {
        ad2petri.helpermetamodel.forkJoinFinalNode(ActivityNode);
        find activityNodeTransitionMapping(ActivityNode, PetriTransition);
        find transitionOfNet(PetriTransition, PetriNet);
    }
    action {
        call copyName(ActivityNode, PetriTransition);
        forall EdgePlace with
            apply connectNodeToOutgoing(ActivityNode, PetriTransition, EdgePlace) do skip;
        forall EdgePlace with
            apply connectNodeToIncoming(ActivityNode, PetriTransition, EdgePlace) do skip;
    }
}
```

Exercise: what's missing?



Exercise: what's missing?

- Decision and Merge Nodes
- What PN constructs should they be mapped into?
- Come up with GT rules
- Implement in Viatra2
- Try activity_complex and activity_prb!

Additional material

- http://wiki.eclipse.org/VIATRA2
  - Installation
  - Syntax
  - How-tos and Examples
  - Learn about many other features of Viatra2
- There is a new release coming up this summer, you have been using a beta version in this lab
  - Sorry for any hiccups it may have caused ☺
  - Take it home if you wish!
  - The unfinished project also!