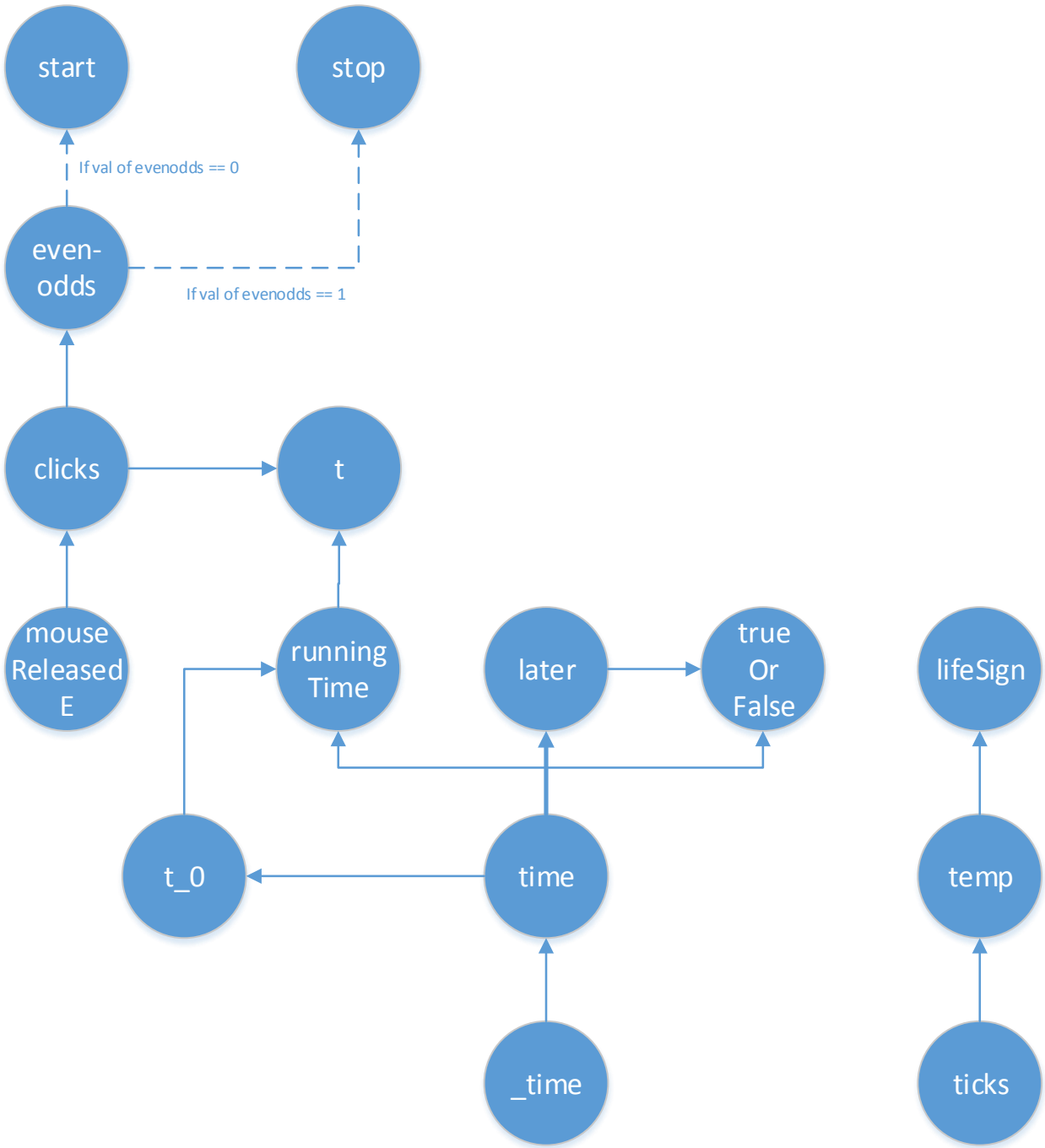
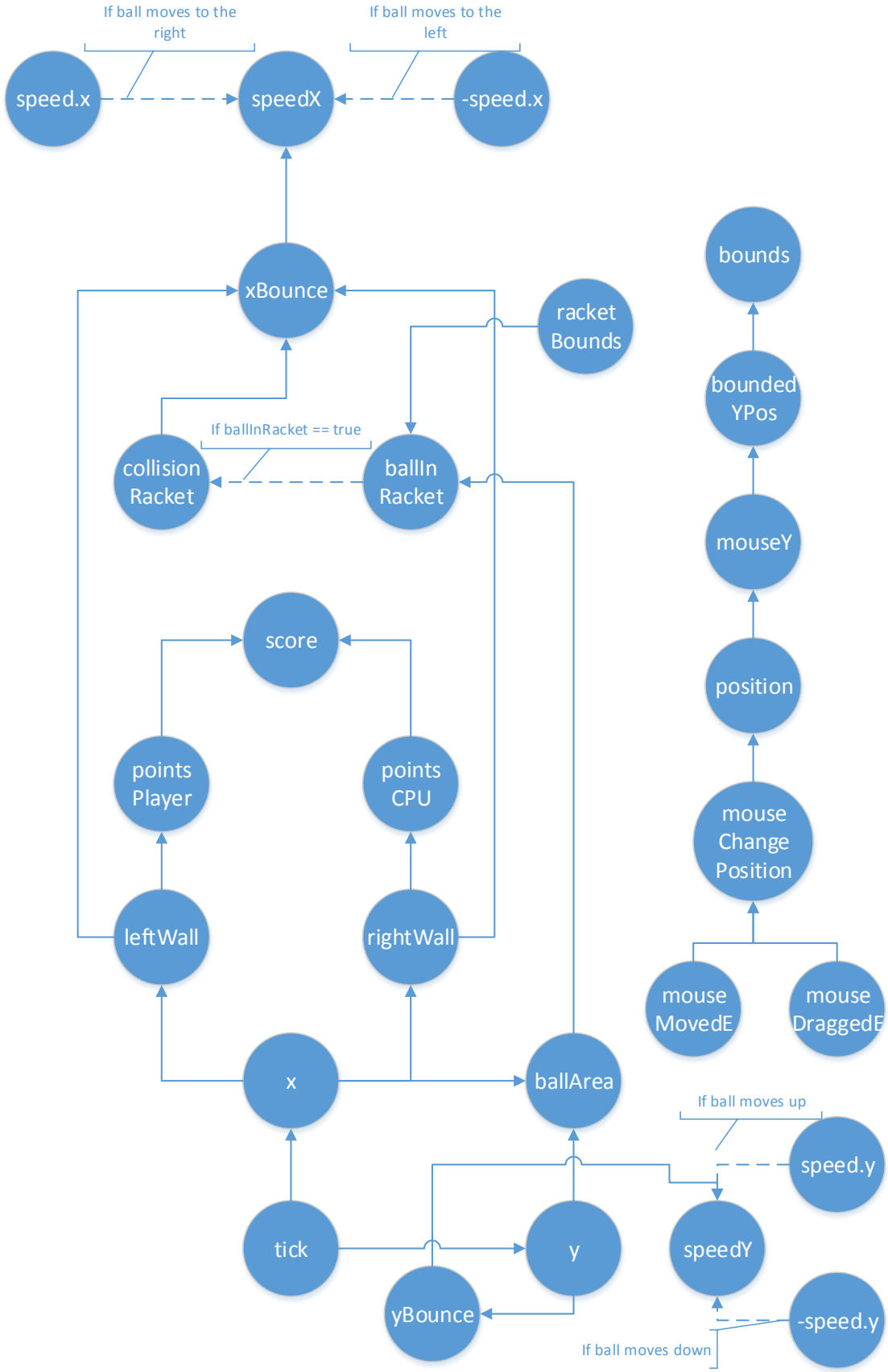
Task 3 - Dependency Graphs

1. Stopwatch



2. Pong


If ball moves to the right
If ball moves to the left
speed.x
speedX
-speed.x
xBounce
racket Bounds
bounds
bounded YPos
If ballInRacket == true
collision Racket
ballIn Racket
mouseY
score
position
points Player
points CPU
mouse Change Position
leftWall
rightWall
mouse MovedE
mouse DraggedE
x
ballArea
If ball moves up
speed.y
tick
y
speedY
yBounce
-speed.y
If ball moves down

Task 4 – Stopwatch with Observers

1. Which implementation problems would you encounter and how would you solve them?
   - introduce boilerplate code --> cannot be solved
   - Constraints are not composable --> introduce chains of observers (observers observe constraints to extending them by an update).
   - Values could be updated too much --> implement conditional observers to mimic reduce the updates.
   - The need of different observer and observable type (to make it possible to observe different events) --> implement them
2. How does a Framework like REScala help in this regard?
   - Less Boilerplate code.
   - Declarative events are more composable than observers that would help to compose constraints.
   - Events and Signals do have a good integration with Objects that removes the need of different observer types
3. What trueOrFalse should be?
   trueOrFalse should always yield "true".
4. What in REScala makes sure that it is indeed the case?
   Since "trueOrFalse" depends on "time" and "later", which also depends on "time", REScala does ensure updates in topological order. Due to this, an update on "time" would trigger the update algorithm. The "trueOrFalse"-Signal couldn't be updated because it depends on "later" where the update is still pending and "trueOrFalse" only can be updated if all of "trueOrFalse" 's children are updated already. When "later" gets its update, "trueOrFalse" can be updated to
5. How would you solve it with observers?
   One way to solve it is to use "time" as observable that is observed by a "later" observer which is also observable. If "time" is updated it will notify "later" which can update itself and store the new time value. "later" have to hold the values of "time" and its own now. After that later could notify the last observer that is interested in the updated values, "trueOrFalse". This ensures that "trueOrFalse" is always true.
6. How would you deal with dynamic dependencies in the observer-based implementation?
   A possible solution to this is to register and unregister the observables depending on the given condition. It is also necessary to observe whether the condition changes. If you implement it this way, you have to make sure that you have some instance to query the current state of the condition.