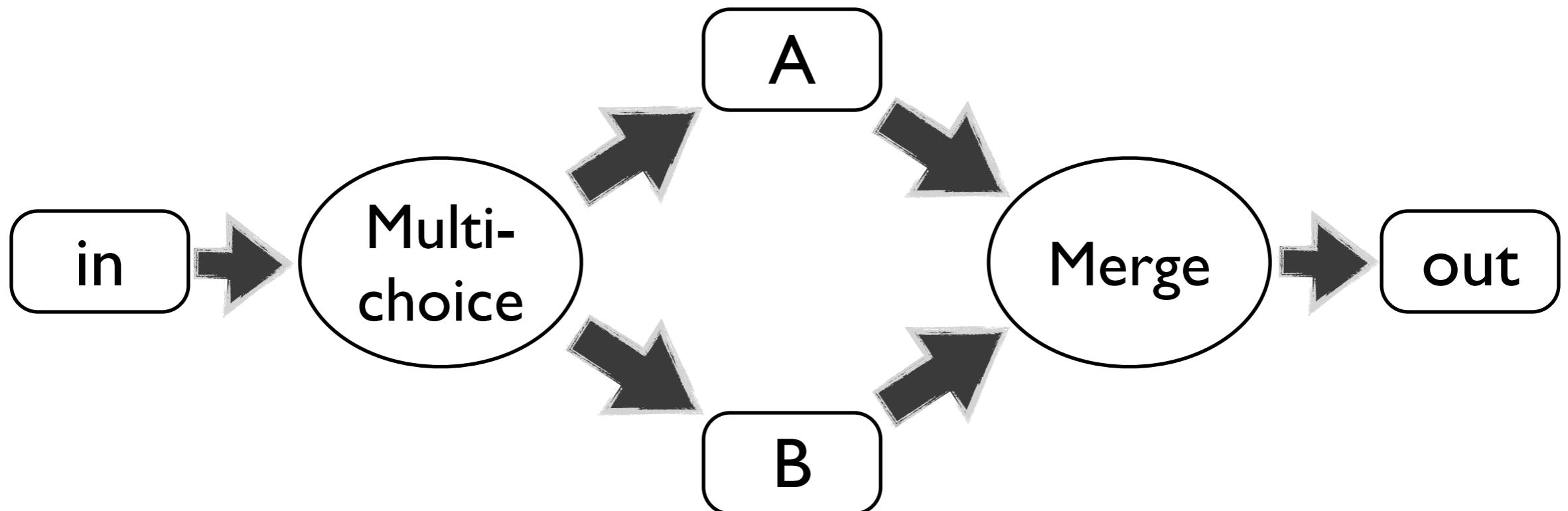


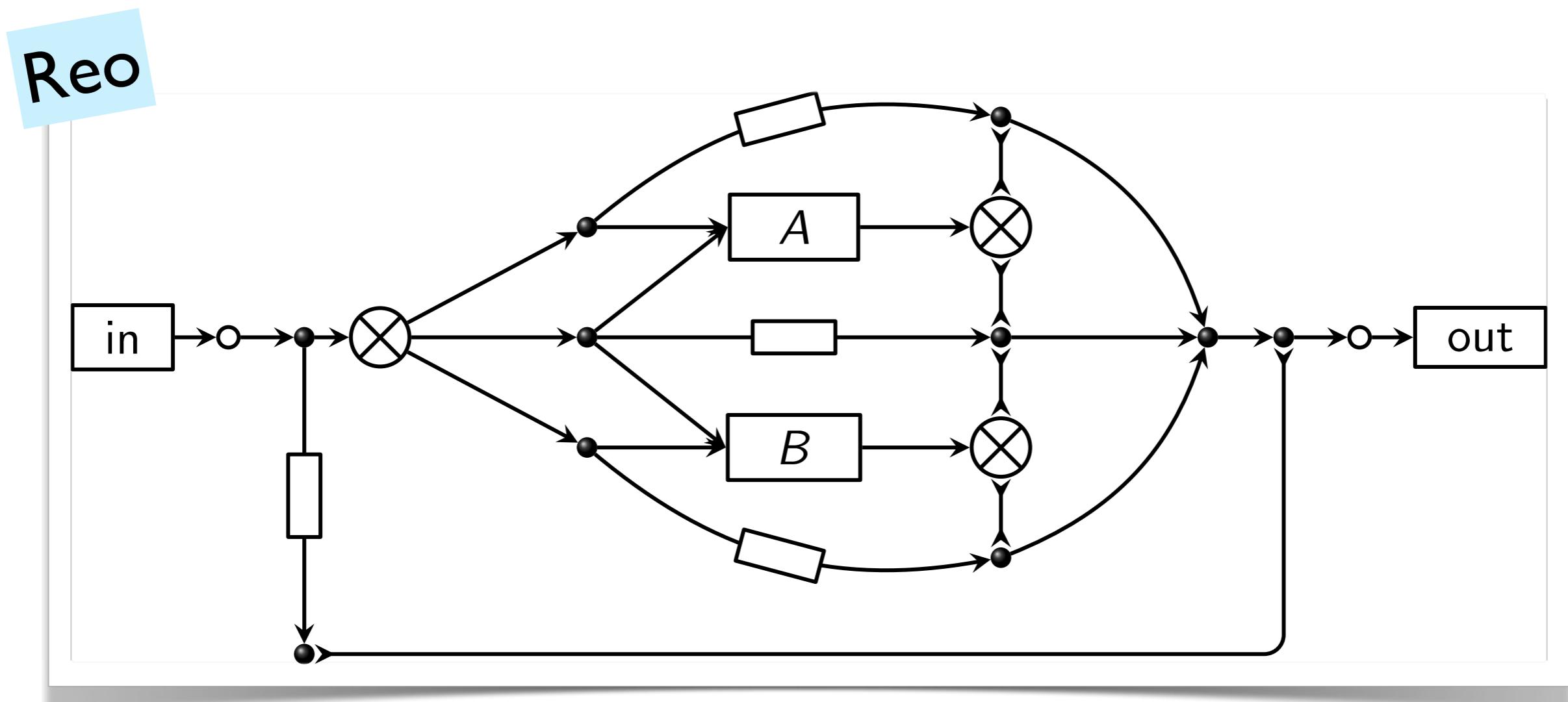
Interactive Interaction Constraints

José Proença & Dave Clarke
KU Leuven, Belgium

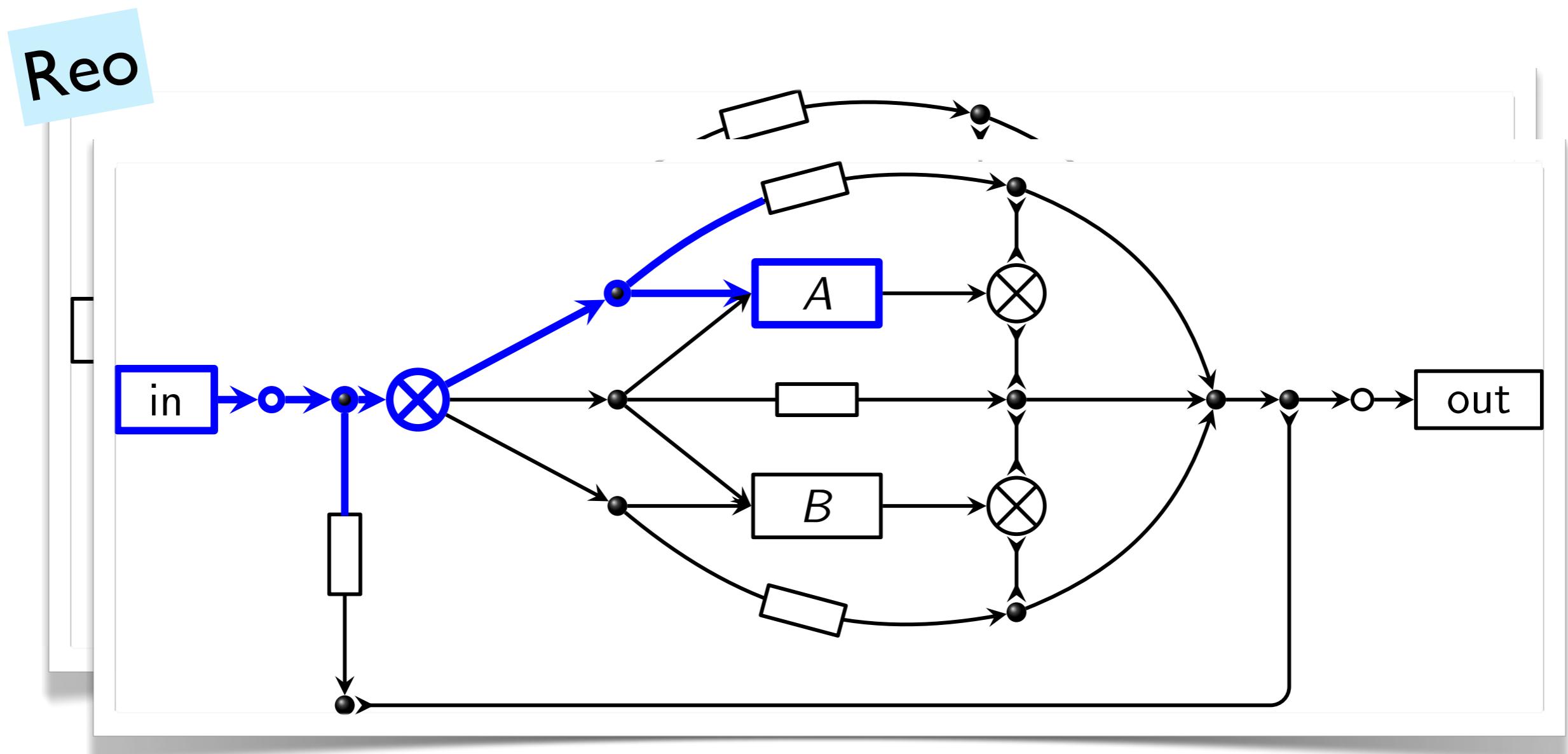
Synchronous coordination



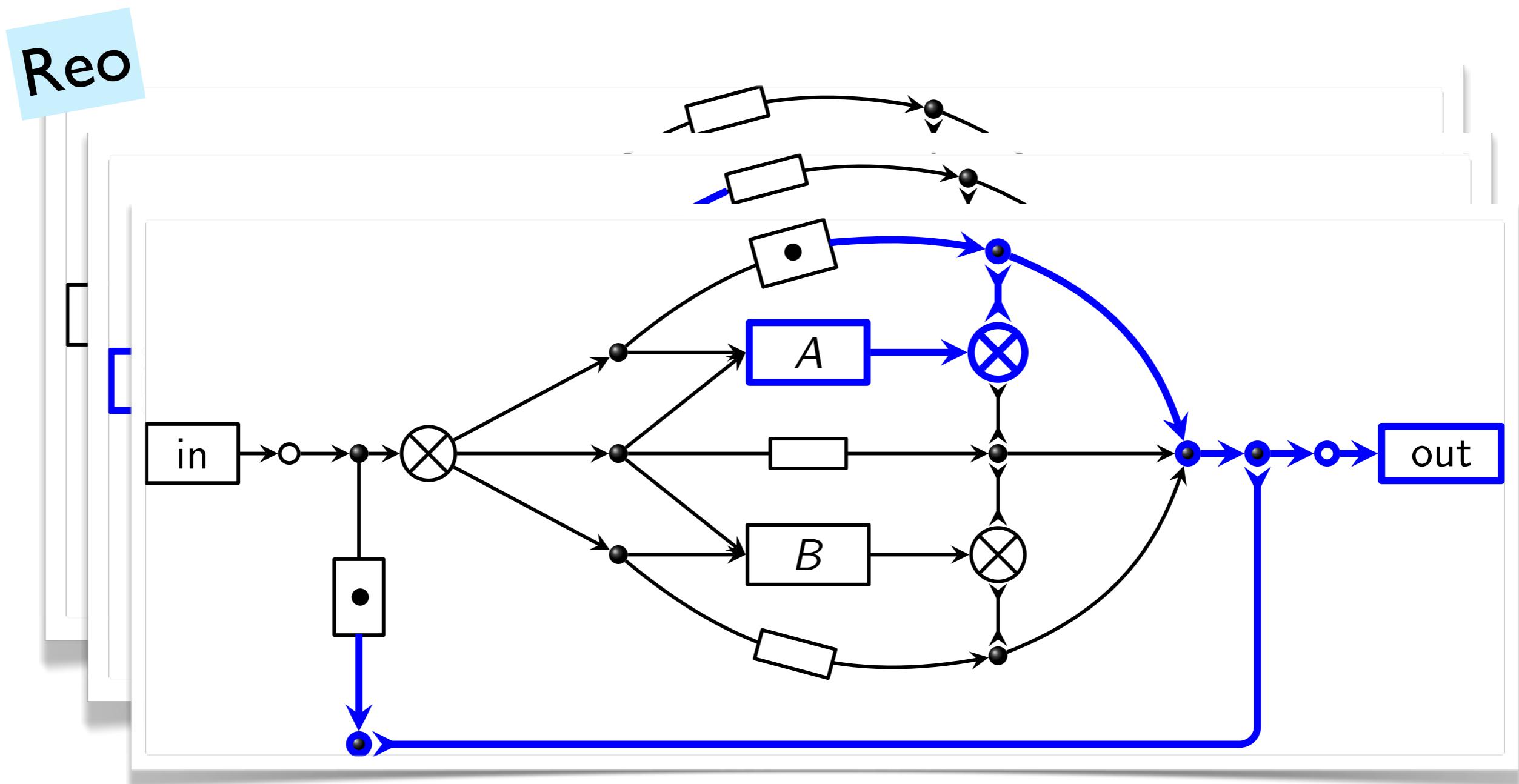
Synchronous coordination



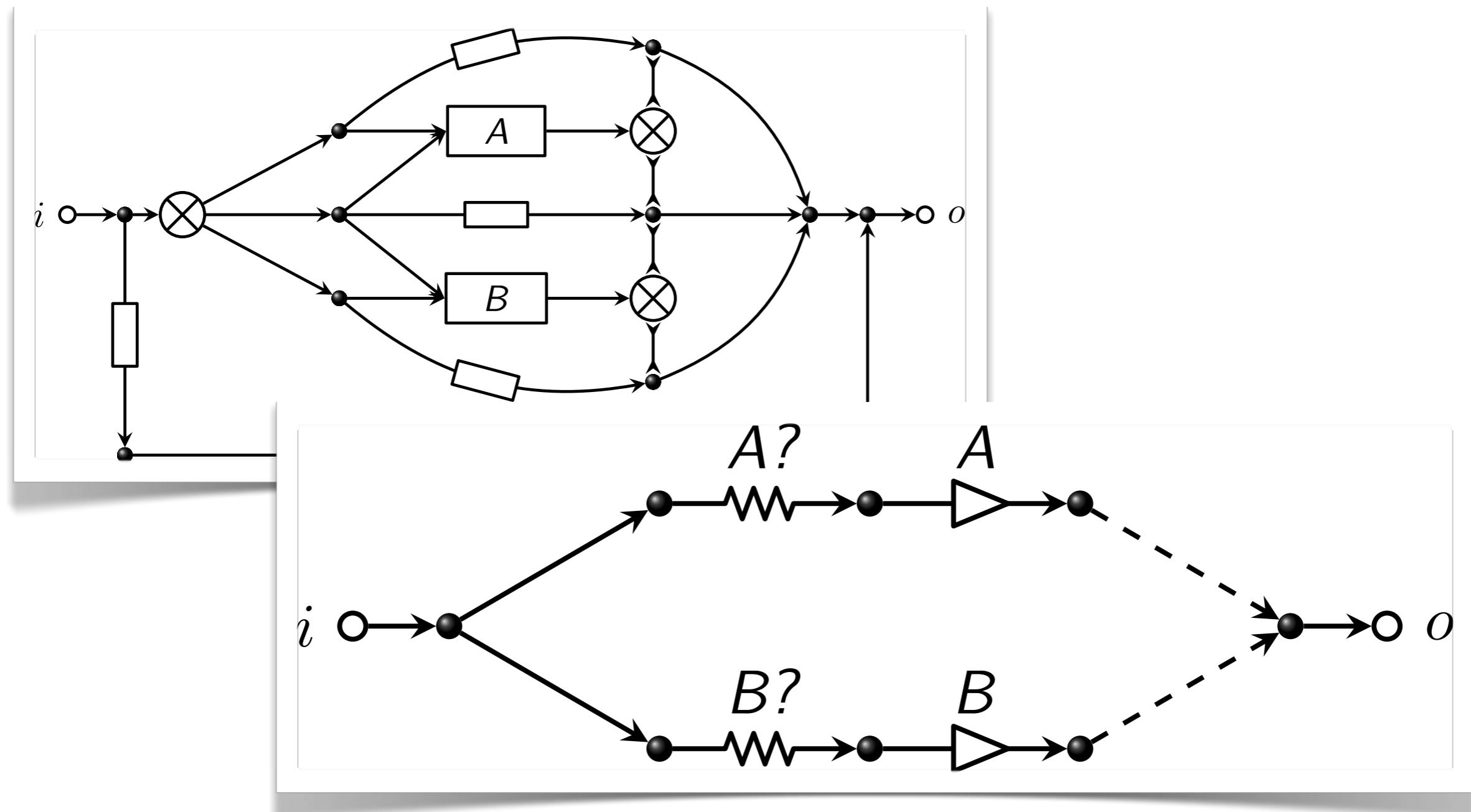
Synchronous coordination



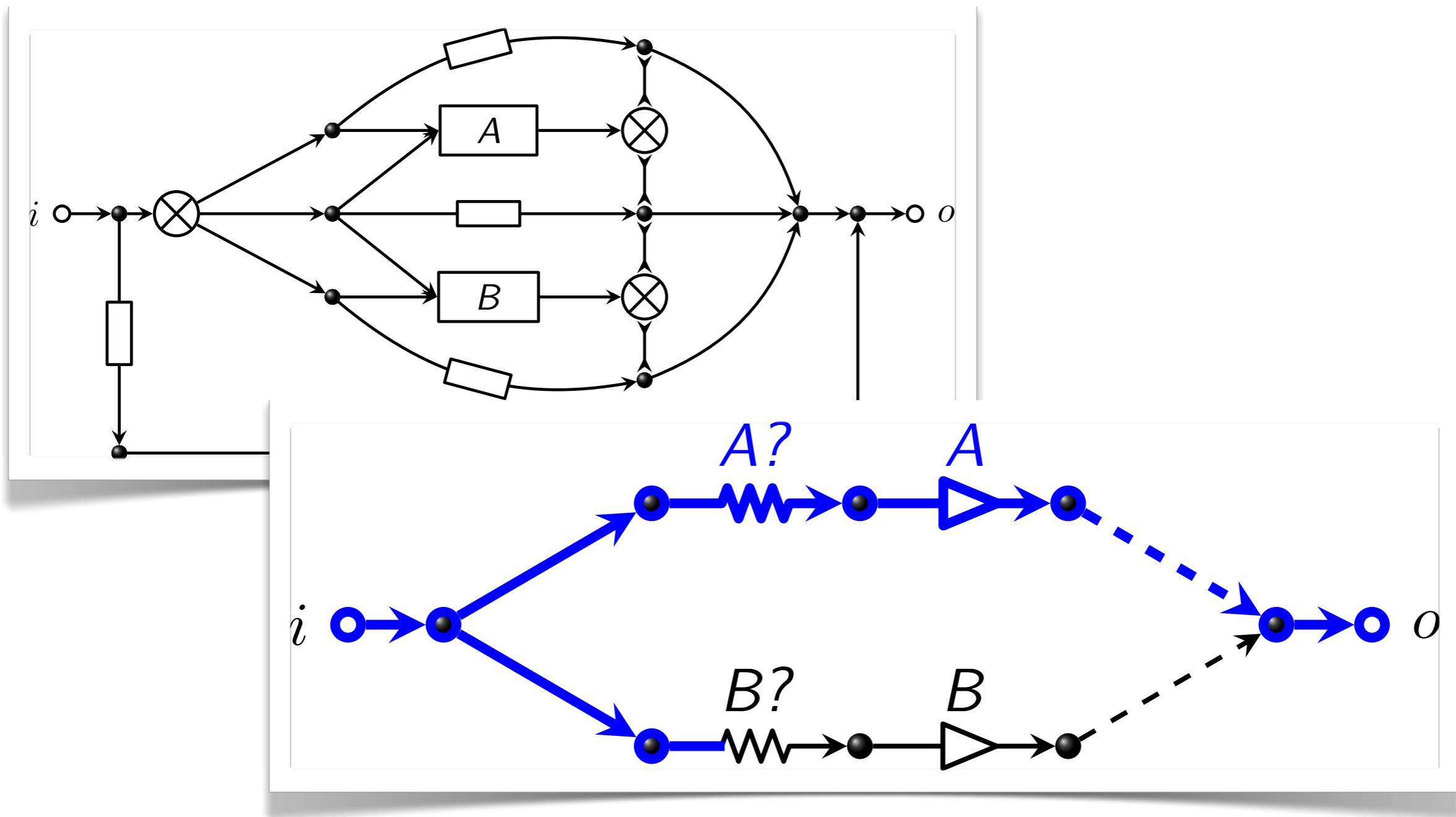
Synchronous coordination



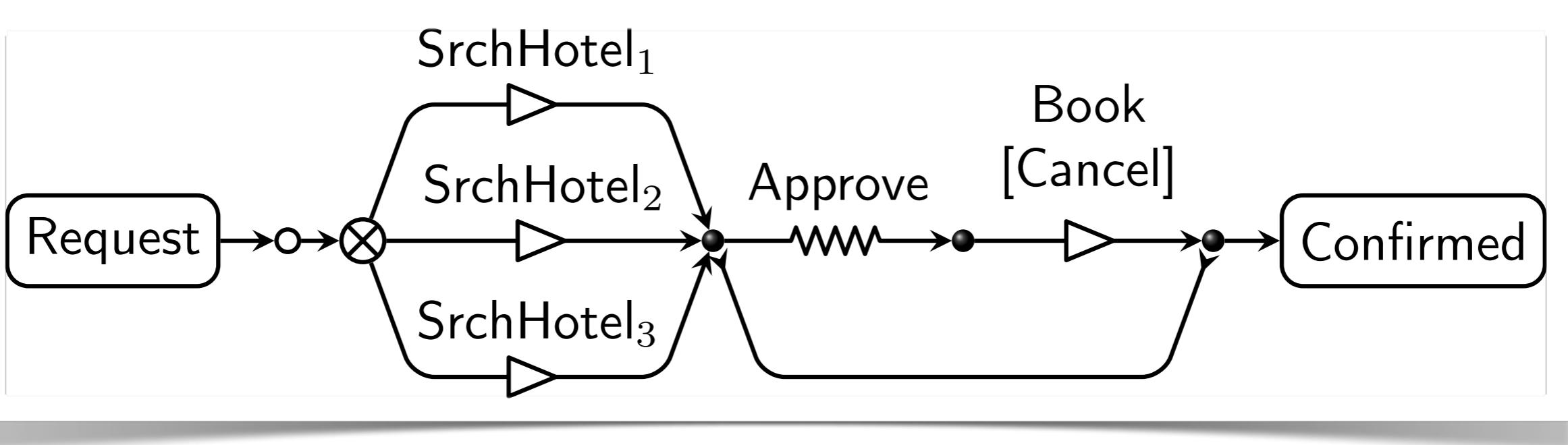
More synchronous



More synchronous

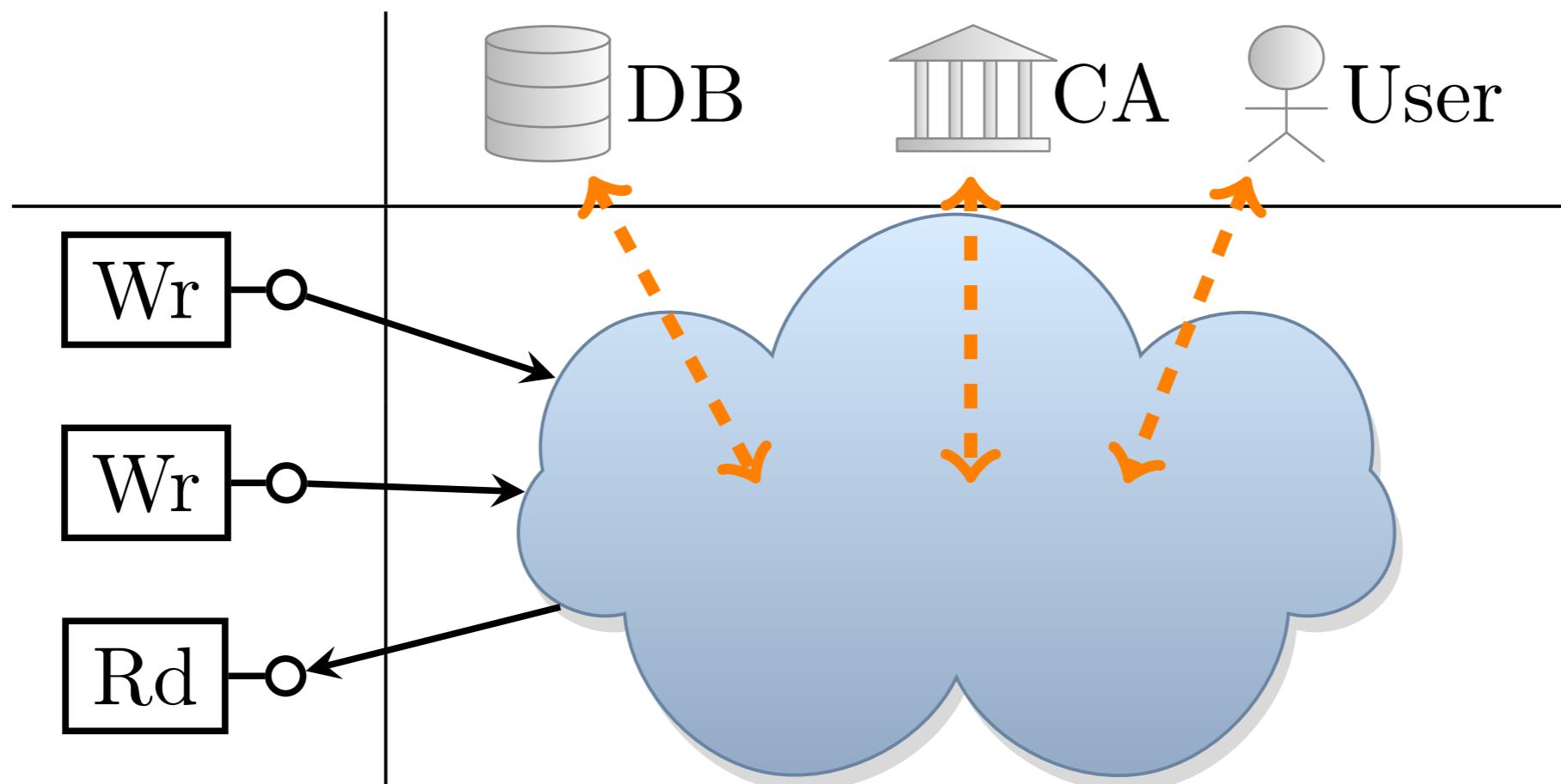


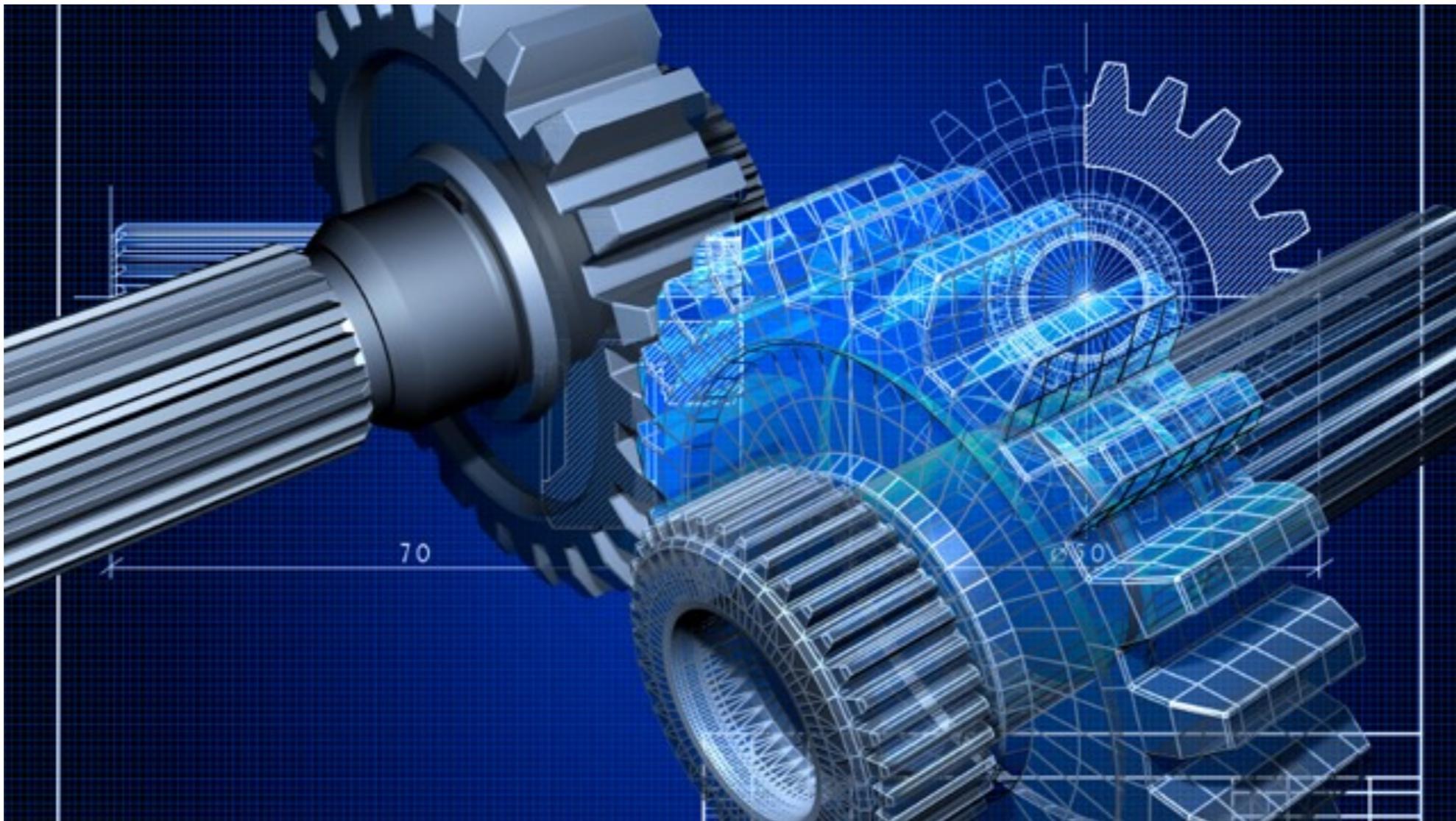
Hotel booking



- Interaction with hotel repositories
- Interaction with users
- Interaction with hotels (availability & payment)

Interactive Interaction Constraints

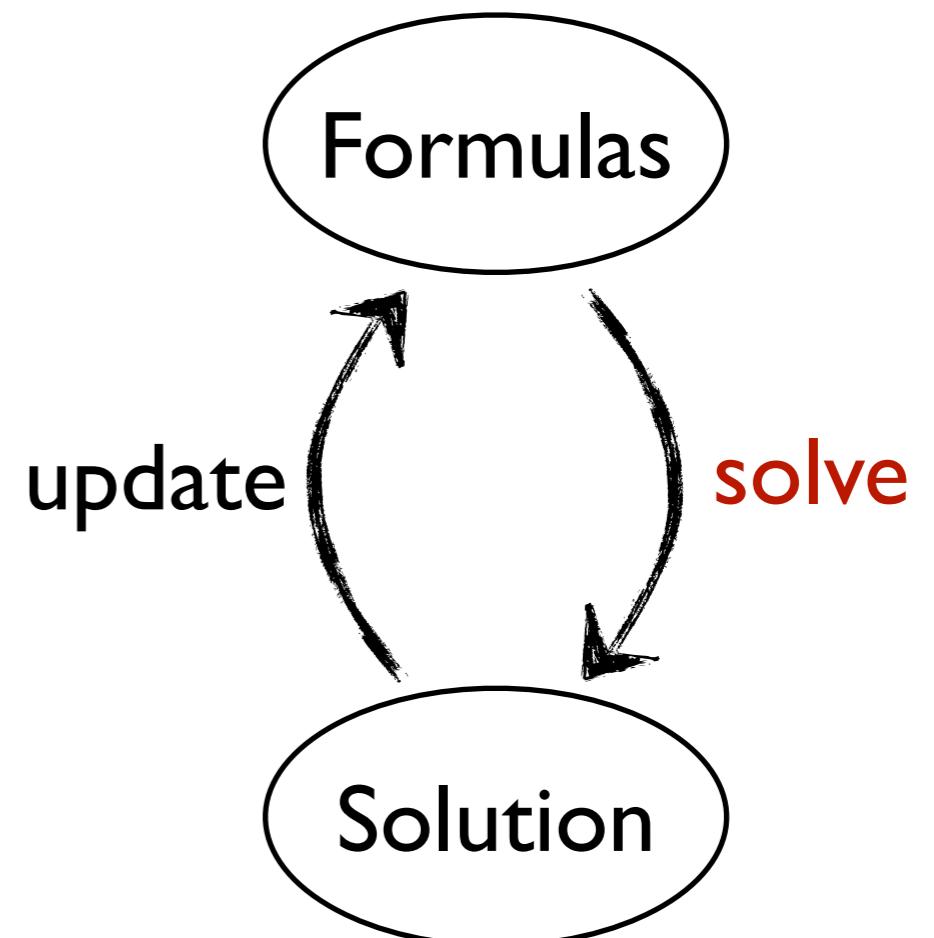




Under the hood

Coordination as constraint satisfaction

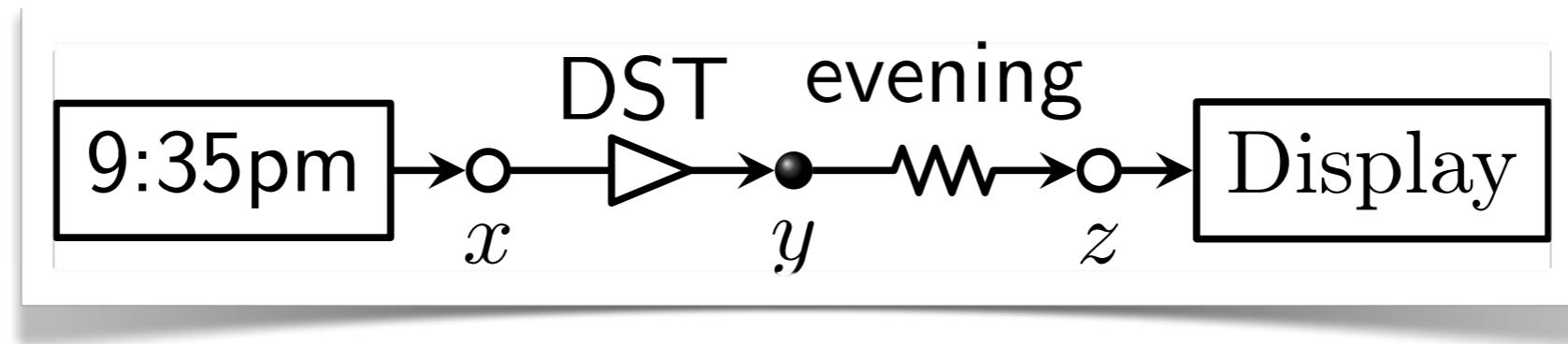
| | |
|-----------------------|---|
| $a \longrightarrow b$ | $a \leftrightarrow b$ $b \rightarrow \hat{b} := \hat{a}$ |
| $a \dashrightarrow b$ | $b \rightarrow a$ $b \rightarrow \hat{b} := \hat{a}$ |
| $a \xrightarrow{P} b$ | $b \rightarrow \hat{b} := \hat{a}$ $(a \wedge \textcolor{blue}{P}(\hat{a})) \leftrightarrow b$ |
| $a \xrightarrow{f} b$ | $a \leftrightarrow b$ $b \rightarrow \hat{b} := \textcolor{blue}{f}(\hat{a})$ |



Solving constraints

- Formulas
- Data formula \rightarrow Boolean formula
- Interaction with the Choco solver
- Scala (Java) prototype implementation

Formulas

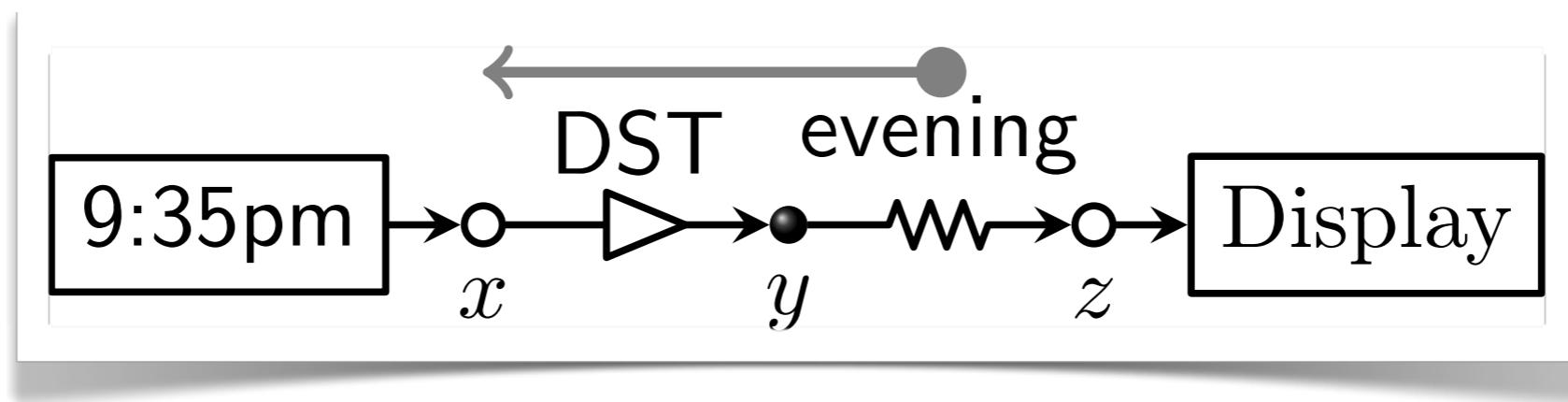


| | | |
|--|-----------------------|--|
| $x \rightarrow \hat{x} := 9:35\text{pm}$ | $x \leftrightarrow y$ | $y \rightarrow \hat{y} := \text{DST}(\hat{x})$ |
| $(y \wedge \text{evening}(\hat{y})) \leftrightarrow z$ | | $z \rightarrow \hat{z} := \hat{y}$ |

Predicate abstraction

- Find **data dependencies** of predicates
- Replace **data variables** by **boolean variables** that guide the predicates

Predicate abstraction

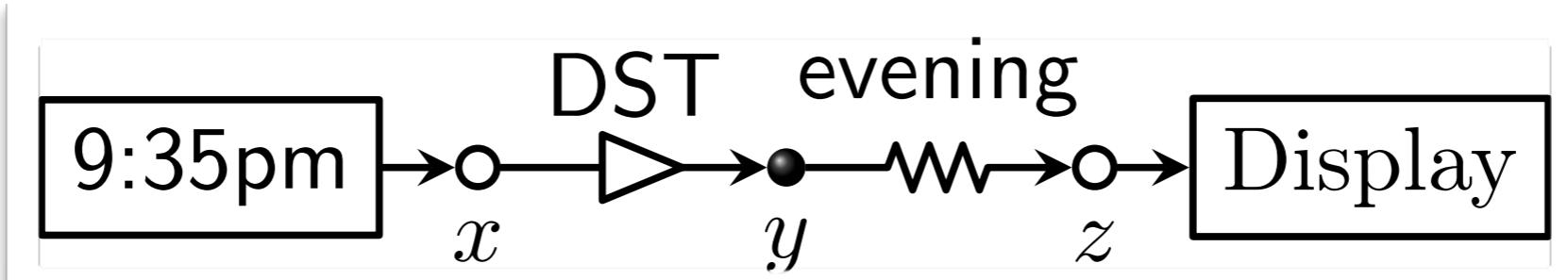


$$\mathcal{D}_x = \{\text{ev.dst}\}$$

$$\mathcal{D}_y = \{\text{ev}\}$$

$$\mathcal{D}_z = \emptyset$$

Predicate abstraction



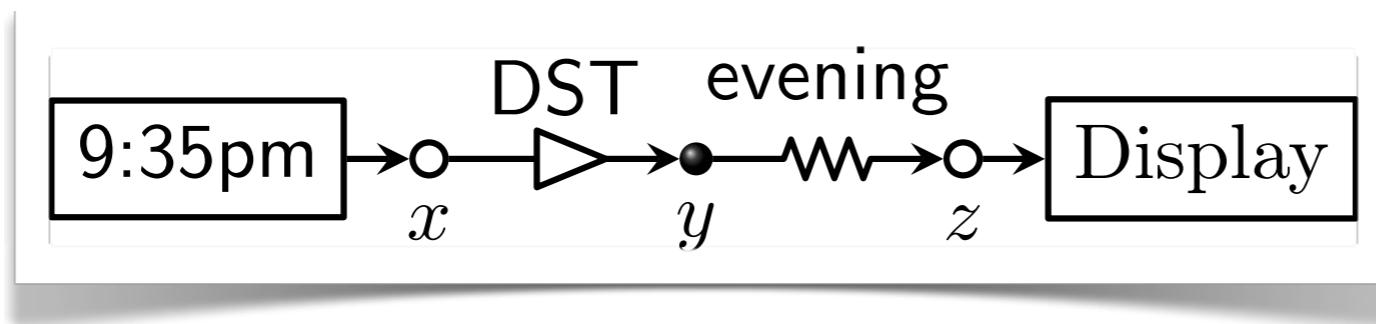
original

$$\begin{array}{lll} x \rightarrow \hat{x} := 9:35pm & x \leftrightarrow y & y \rightarrow \hat{y} := \text{DST}(\hat{x}) \\ (y \wedge \text{evening}(\hat{y})) \leftrightarrow z & & z \rightarrow \hat{z} := \hat{y} \end{array}$$

boolean

$$\begin{array}{ll} x \rightarrow \hat{x}_{\text{ev.dst}} := [\text{evening}(\text{DST}(9:35pm))] & x \leftrightarrow y \\ y \rightarrow \hat{y}_{\text{ev}} := \hat{x}_{\text{ev.dst}} & (y \wedge \hat{y}_{\text{ev}}) \leftrightarrow z \end{array}$$

Interaction with Choco



boolean

$$x \rightarrow \hat{x}_{\text{ev.dst}} := [\text{evening}(\text{DST}(9:35pm))] \quad x \leftrightarrow y$$
$$y \rightarrow \hat{y}_{\text{ev}} := \hat{x}_{\text{ev.dst}} \quad (y \wedge \hat{y}_{\text{ev}}) \leftrightarrow z$$

interactive

$$x \rightarrow \text{XPred}(\text{ev.dst}, x, 9:35pm) \quad x \leftrightarrow y$$
$$y \rightarrow \hat{y}_{\text{ev}} := \hat{x}_{\text{ev.dst}} \quad (y \wedge \hat{y}_{\text{ev}}) \leftrightarrow z$$

Interaction with Choco

- instance of a Choco constraint
- reacts when x or $\hat{x}_{\text{ev.dst}}$ is instantiated
- $\neg x \Rightarrow \hat{x}_{\text{ev.dst}}$ can be anything
- $x \Rightarrow \hat{x}_{\text{ev.dst}} = \text{ev(dst(9:35pm))}$

9:35pm

boolean

$x \rightarrow \hat{x}_{\text{ev.dst}}$

= [evening(DST(9:35pm))]

$x \leftrightarrow y$

$y \rightarrow \hat{y}_{\text{ev}} := \hat{x}_{\text{ev.dst}}$

$(y \wedge \hat{y}_{\text{ev}}) \leftrightarrow z$

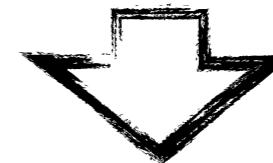
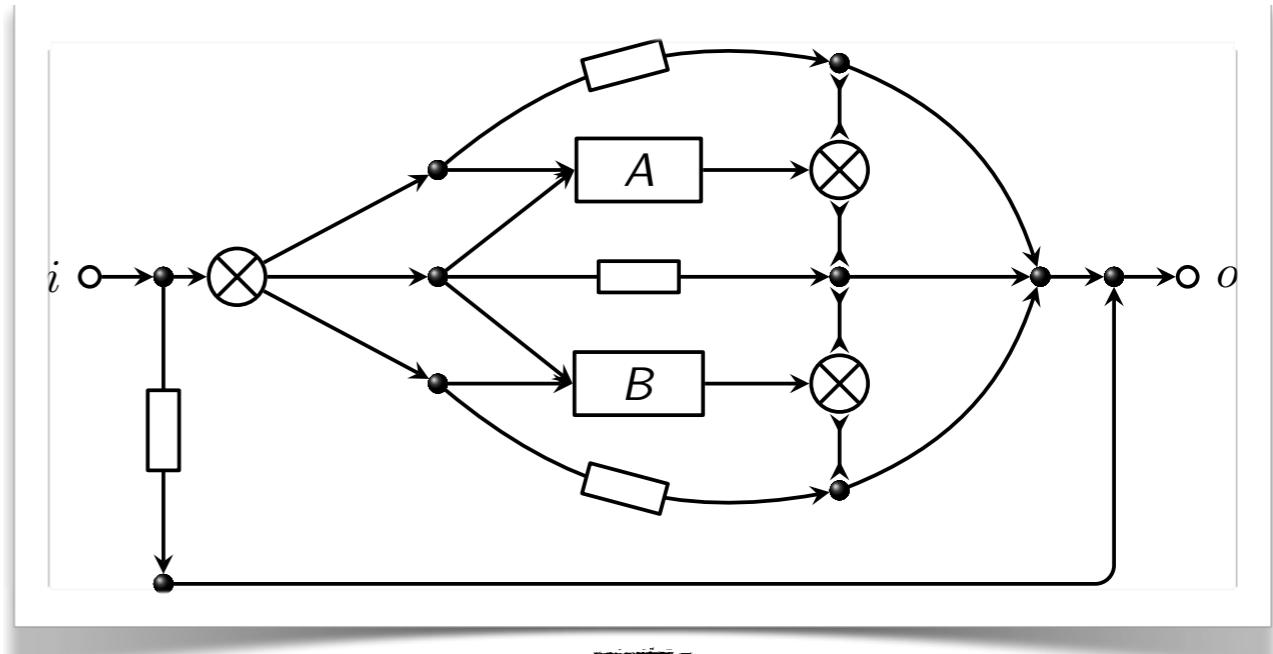
interactive

$x \rightarrow \text{XPred}(\text{ev.dst}, x, 9:35pm)$ $x \leftrightarrow y$

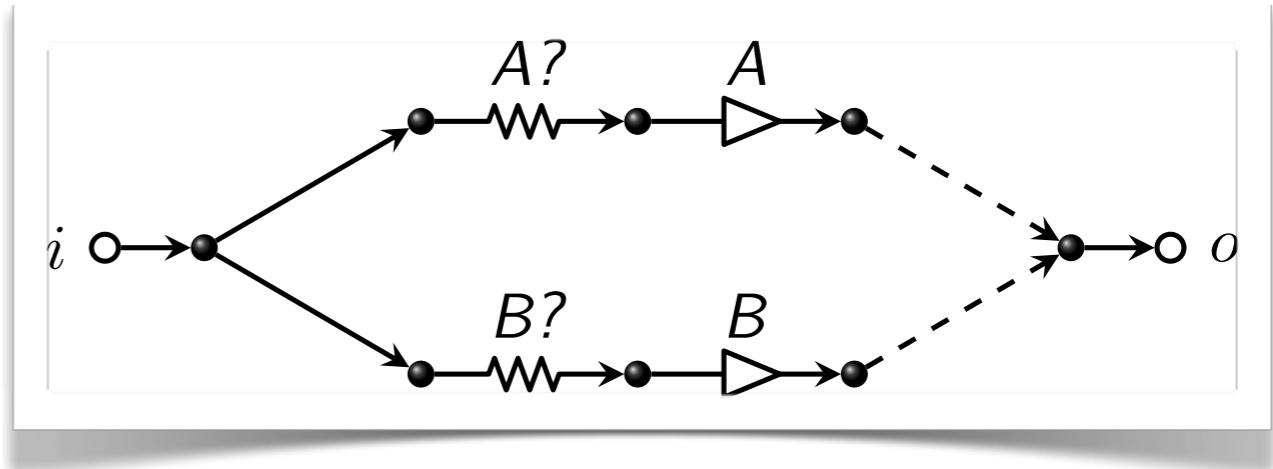
$y \rightarrow \hat{y}_{\text{ev}} := \hat{x}_{\text{ev.dst}}$ $(y \wedge \hat{y}_{\text{ev}}) \leftrightarrow z$

Wrapping up

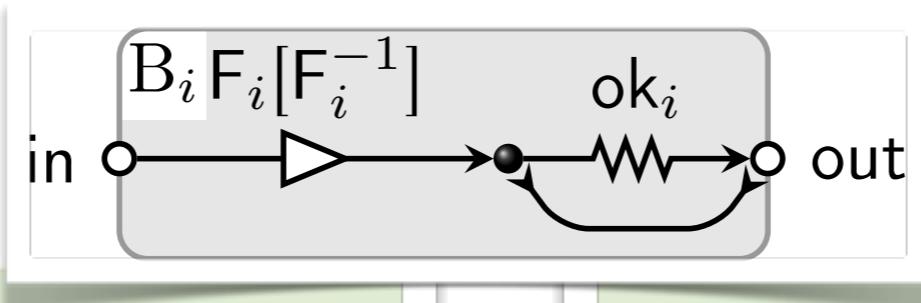
- Interactive constraint solving
- Expose the atomicity of Reo to components



- Beyond this paper:
Avoiding pre-processing
using an SMT solver



Scala / Java implementation



```
val f = Function("f") {  
    case s: String => /* do something */  
}  
  
val finv = Function("f^-1") {  
    case s: String => /* do something */  
}  
  
val ok = Predicate("ok") {  
    case s: String => /* do something */  
}  
  
val connector =  
    writer("in",List("a","b")) ++  
    transf("in","x",f,finv) ++  
    filter("x","out",ok) ++  
    reader("out",2)  
  
connector.run()
```

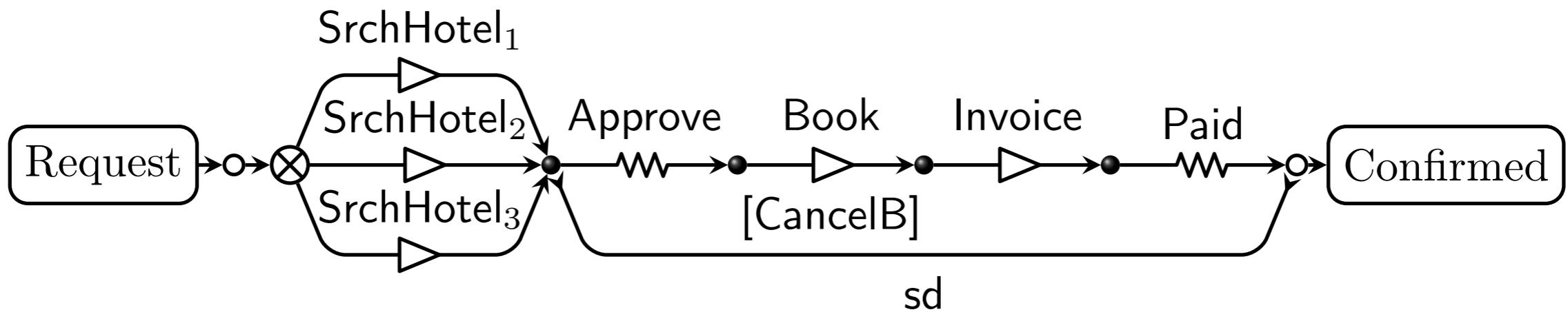
```
class Filter(as: String, bs: String,  
            uid: Int, g: Guard)  
    extends ... {  
  
    val a = Var(flowVar(as,uid))  
    val b = Var(flowVar(bs,uid))  
  
    def getConstraints = Formula(  
        b --> a,  
        b --> (b := a),  
        b --> g,  
        (a /\ g) --> b  
    )  
  
    /* override def update(s:  
        Option[Solution]) = ... */  
}
```

Formulas

| | |
|---|--------------|
| $\psi ::= \phi \rightarrow s \mid \psi_1 \ \psi_2 \mid \top$ | (formulas) |
| $\phi ::= x \mid P(\hat{x}) \mid \phi_1 \wedge \phi_2 \mid \neg\phi$ | (guards) |
| $s ::= \phi \mid s_1 \wedge s_2$ | |
| $\hat{x} := d \mid \hat{x}_1 := \hat{x}_2 \mid \hat{x}_1 := f(\hat{x}_2)$ | (statements) |

Well-defined formula:
no data-loops; single assignment; data source

Hotel booking



- Interaction with users
- Interaction with hotel repositories
- Interaction with hotels (availability)
- Interaction with hotels (payment)

```

object HotelReservation extends App {
  case class Req(val content:String)

  def srchHotel(i:Int) =
    Function("SearchHotel-"+i){
      case r:Req => i match {
        case 1 => List("F1","Ibis","Mercury")
        case 2 => List("B&B","YHostel")
        case _ => List("HotelA","HotelB")
      }
    }

  val approve = Predicate("approve"){
    case l:List[String] =>
      println("approve: "+l.mkString(","))
      readChar() == 'y'
  }

  val book = Function("book"){
    case l : List[String] =>
      println("Options: "+l.mkString(", ") +
        ". Which one? (1.." + l.length + ")")
      val res = readInt()
      l(res-1)
  }

  val cancelB = Function("cancelB"){
    case x => println("canceling "+x+".")
  }

  val invoice = Function("invoice"){
    case x => println("invoice for "+x+".")
  }
}

```

```

val pay = Predicate("paid"){
  case x => if (x == "Ibis") {
    println("paid for Ibis")
    true
  } else {
    println("not paid for "+x)
    false
  }
}

// Connector definition
val connector =
  writer("req",List(Req("req1"),
    Req("req2")))) ++
  nexrouter("req",List("h1","h2","h3")) ++
  transf("h1","h1o",srchHotel(1)) ++
  transf("h2","h2o",srchHotel(2)) ++
  transf("h3","h3o",srchHotel(3)) ++
  nmerger(List("h1o","h2o","h3o"),"hs") ++
  filter("hs","ap",approve) ++
  sdrain("hs","ap") ++
  transf("ap","bk",book,cancelB) ++
  monitor("bk","inv",invoice) ++
  filter("inv","paid",paid) ++
  reader("paid",5)

connector.run()
}

```