Architectural design: the coordination perspective

José Proença
HASLab - INESC TEC & UM

Arquitectura e Cálculo 2015-16

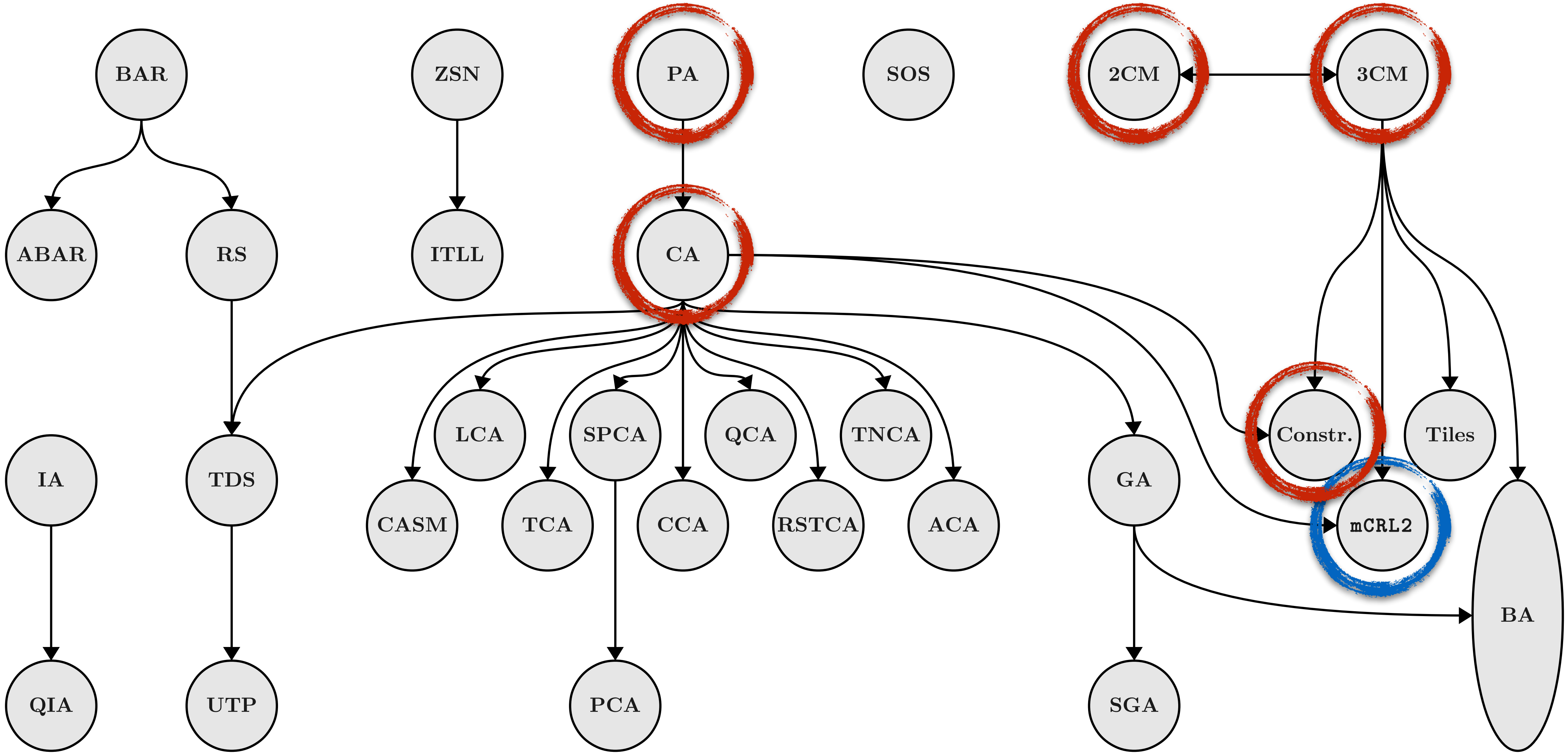
Reo semantics

Jongmans and Arbab 2012

Overview of Thirty Semantic Formalisms for Reo

Reo semantics

- *Coalgebraic models*
  - Timed data streams
  - Record streams
- *Coloring models*
  - **Two colors**
  - **Three colors**
  - Tile models
- *Other models*
  - **Process algebra**
  - **Constraints**
  - Petri nets & intuitionistic logic
  - Unifying theories of programming
  - Structural operational semantics
- *Operational models*
  - **Constraint automata**
  - Variants of constraint automata
    - **Port automata**
    - Timed Probabilistic
    - Continuous-time
    - Quantitative
    - Resource-sensitive timed
    - Transactional
  - Context-sensitive automata
    - Büchi automata
    - Reo automata
    - Intentional automata
    - Action constraint automata
    - Behavioral automata



2CM	: Coloring models with two colors [28, 29, 33]	PA	: Port automata [45]
3CM	: Coloring models with three colors [28, 29, 33]	PCA	: Probabilistic CA [15]
ABAR	: Augmented BAR [39, 40]	QCA	: Quantitative CA [12, 53]
ACA	: Action CA [46]	QIA	: Quantitative IA [13]
BA	: Behavioral automata [61]	RS	: Record streams [38, 40]
BAR	: Büchi automata of records [38, 40]	RSTCA	: Resource-sensitive timed CA [51]
CA	: Constraint automata [10, 17]	SGA	: Stochastic GA [56, 57]
CASM	: CA with state memory [60]	SOS	: Structural operational semantics [58]
CCA	: Continuous-time CA [18]	SPCA	: Simple PCA [15]
Constr.	: Propositional constraints [30, 31, 32]	TCA	: Timed CA [8, 9]
GA	: Guarded automata [20, 21]	TDS	: Timed data streams [4, 5, 14, 62]
IA	: Intentional automata [33]	Tiles	: Tile models [11]
ITLL	: Intuitionistic temporal linear logic [27]	TNCA	: Transactional CA [54]
LCA	: Labeled CA [44]	UTP	: Unifying theories of programming [55, 52]
mCRL2	: Process algebra [47, 48, 49]	ZSN	: Zero-safe nets [27]

Outline

Formalism	Synchr.	Data	Context	Partial
Connector Colouring	CC2	-	CC3	-
Automata	Port Automata	Constraint Automata	-	-
Constraints	✓	✓	✓	✓

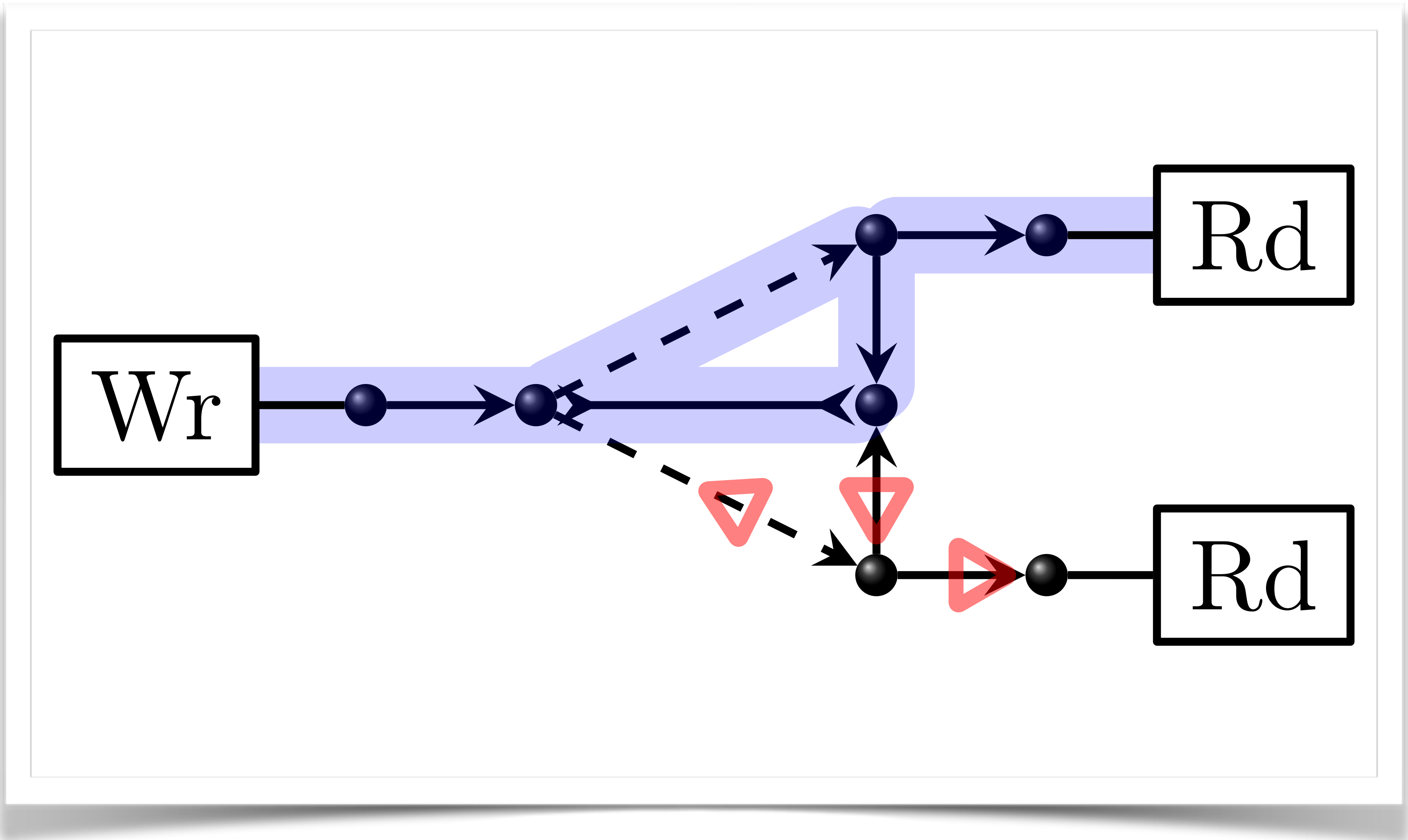
Outline

Intuitive & visual

Formalism	Synchr. 1	Data 2	Context 3	Partial 4
Connector Colouring	CC2	-	CC3	PCC
Automata	Port Automata	Constraint Automata	-	-
Constraints	✓	✓	✓	✓

Compilation & verification

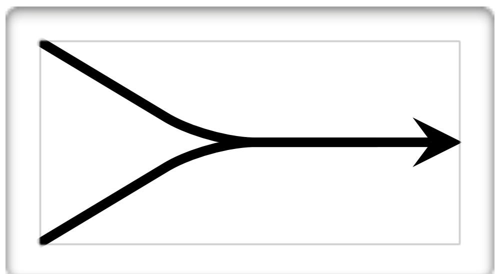
Runtime / scalability



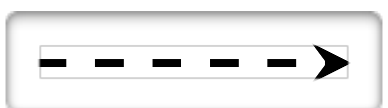
Reo Connector Colouring

Dave Clarke, David Costa, and Farhad Arbab. Connector colouring I: Synchronisation and context dependency

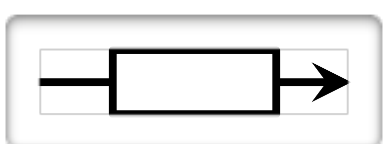
Behaviour?



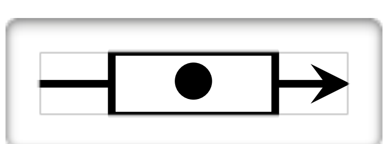
merger: data flows from one of the source ends to the sink end



lossy-sync: either data flows from the source to the sink end, OR it is lost

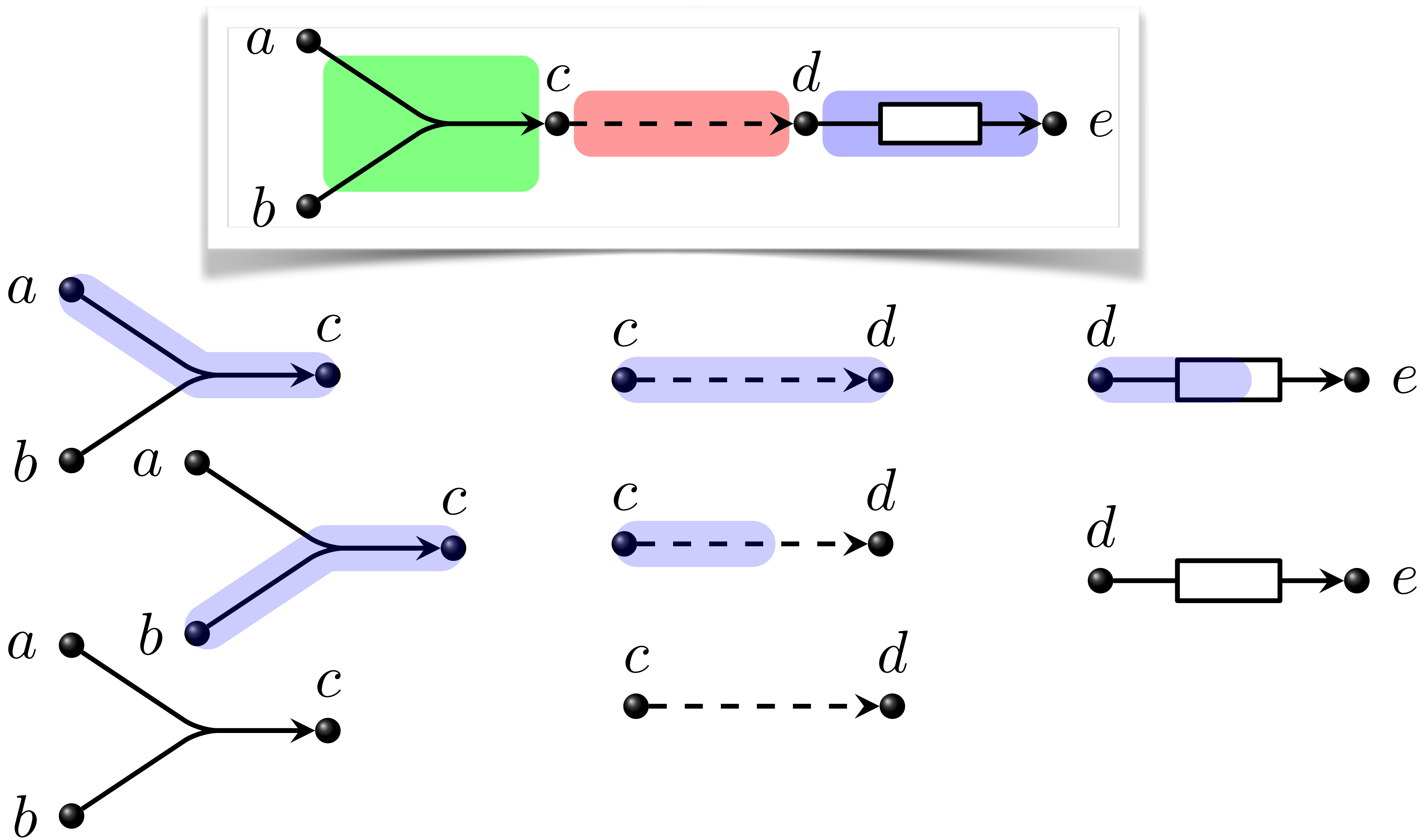


FIFO-1: data flows from the source end to the buffer, becoming a FIFOFull-1

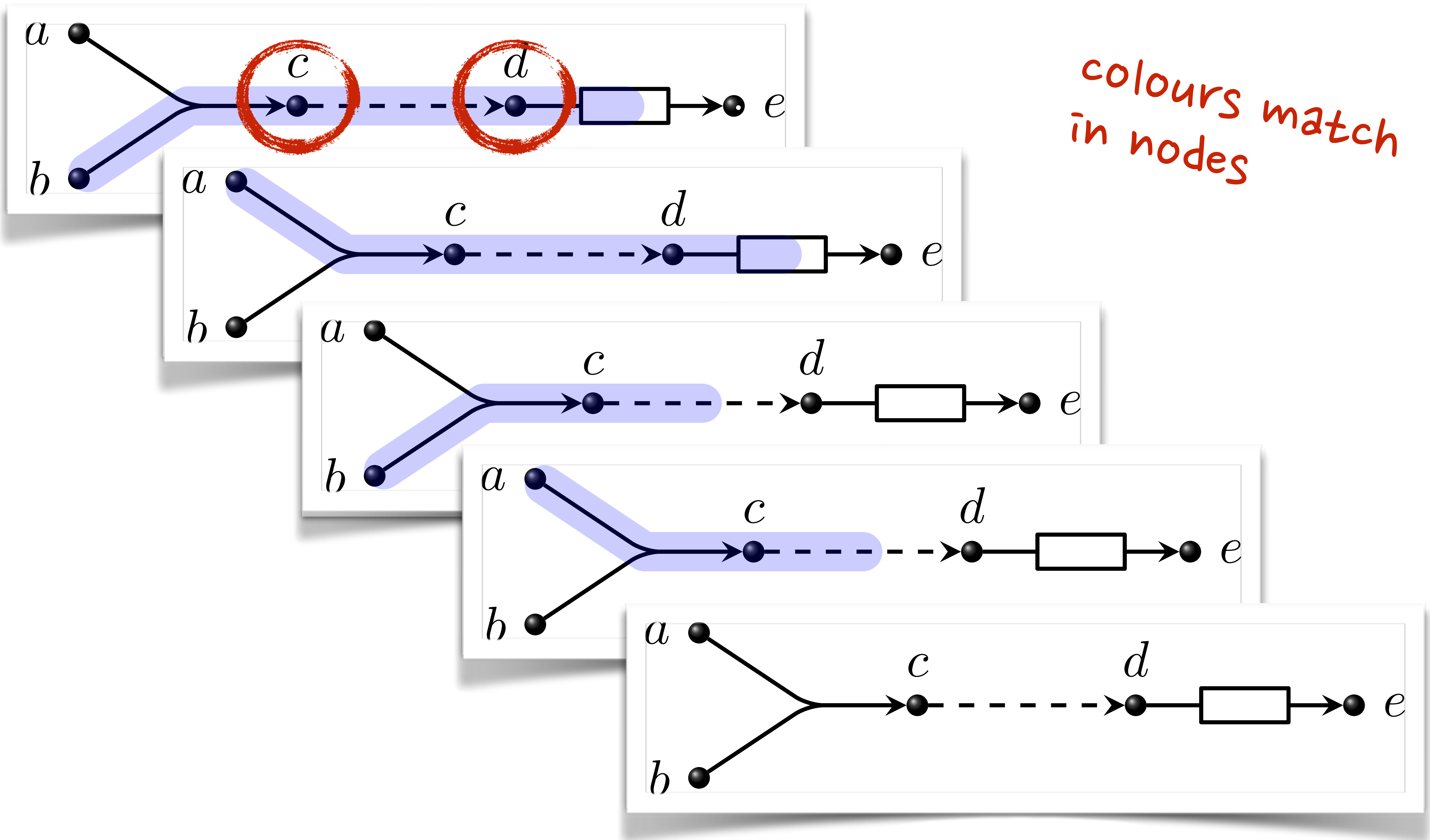


FIFOFull-1: data flows from the buffer to the sink buffer, becoming a FIFO-1

Colourings to describe synchronous dataflow



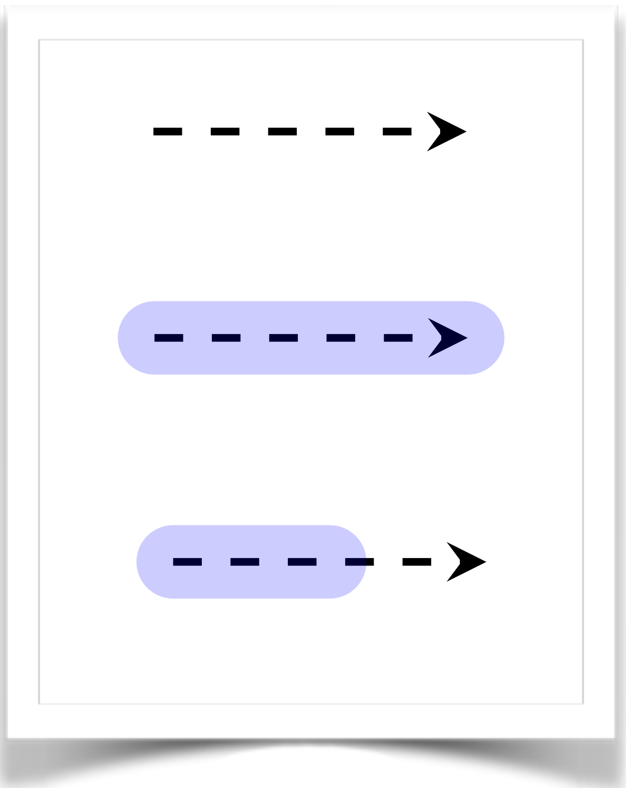
Colouring composition



colours match in nodes

Colouring semantics (CC2)

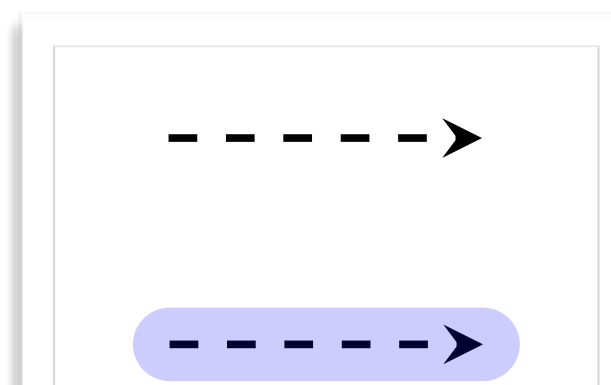
- *Colouring:* End → {Flow, NoFlow}
- *Colouring table:* Set(Colouring)
- *Composition* = matching colours



- More visual (intuitive)
- Used for generating <u>animations</u>

Colouring semantics (CC2)

- *Colouring:* End → {Flow, NoFlow}
- *Colouring table:* Set(Colouring)
- *Composition* = matching colours

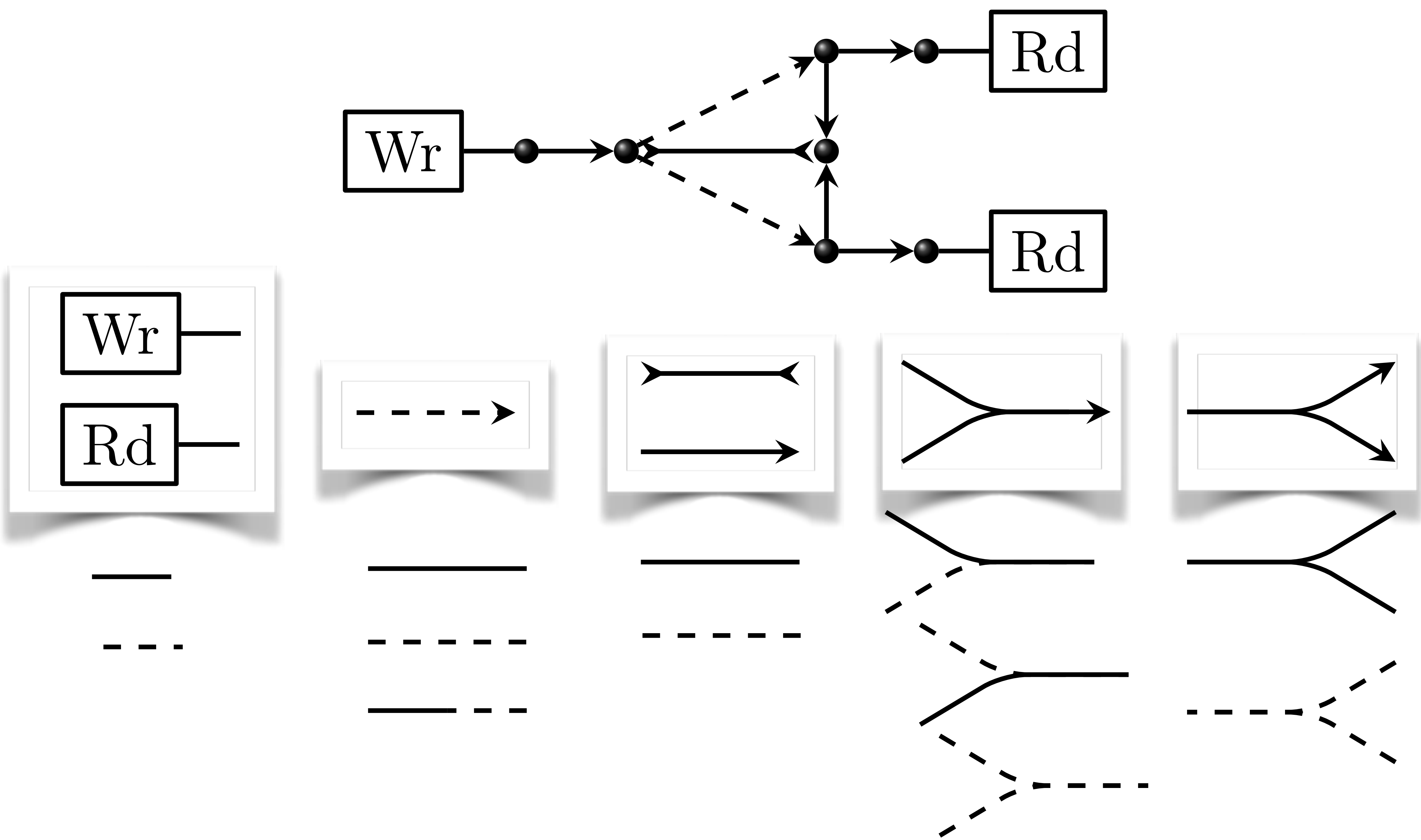


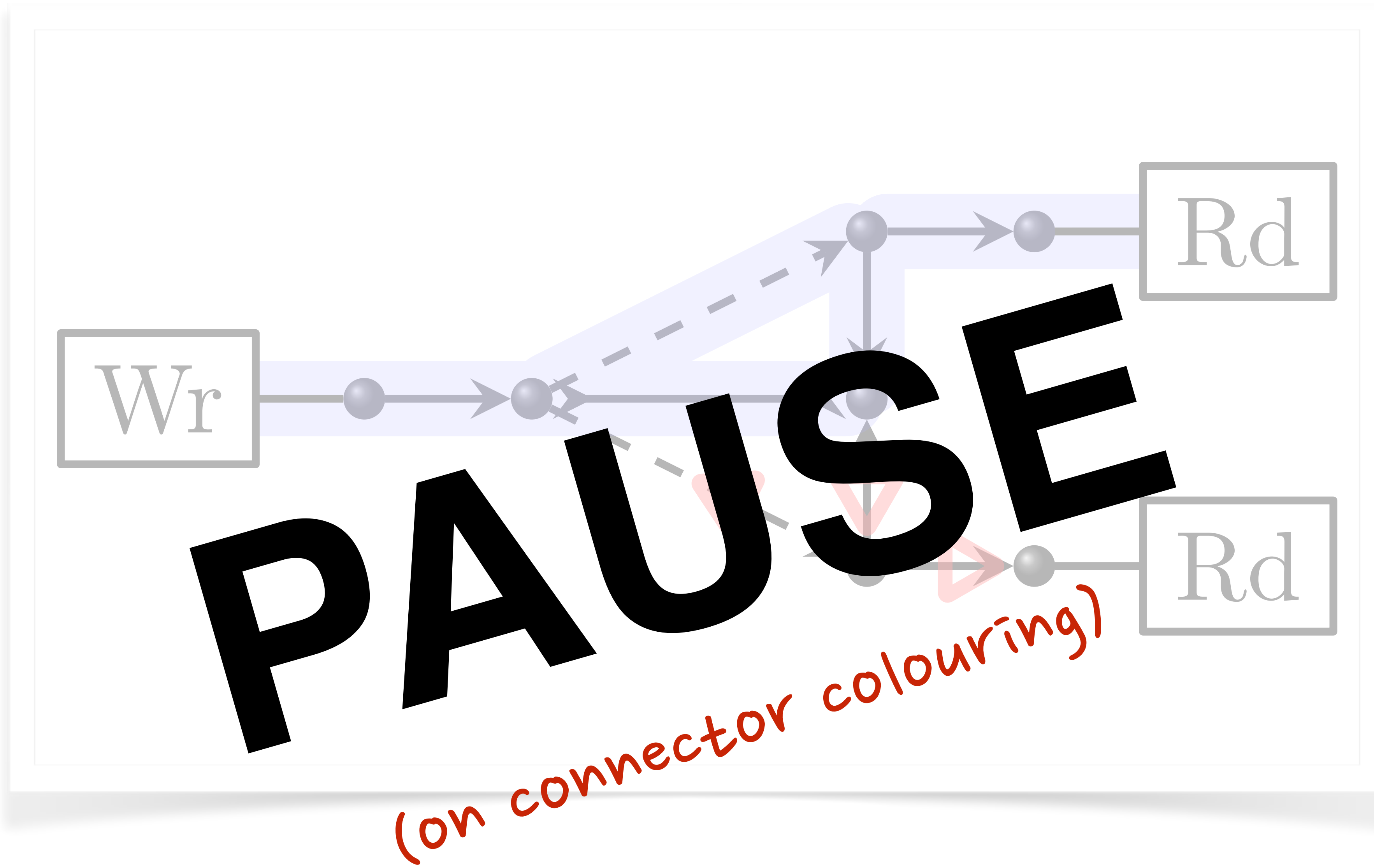
$$
\begin{aligned}
CT_1 \bowtie CT_2 \quad &= \\
&\{cl_1 \bowtie cl_2 \mid cl_1 \in CT_1, cl_2 \in CT_2, cl_1 \frown cl_2\}
\end{aligned}
$$

$$cl_1 \frown cl_2 \quad = \quad \forall e \in \mathsf{dom}(cl_1) \cap \mathsf{dom}(cl_2) \cdot cl_1(e) = cl_2(e)$$

$$cl_1 \bowtie cl_2 \quad = \quad cl_1 \cup cl_2$$

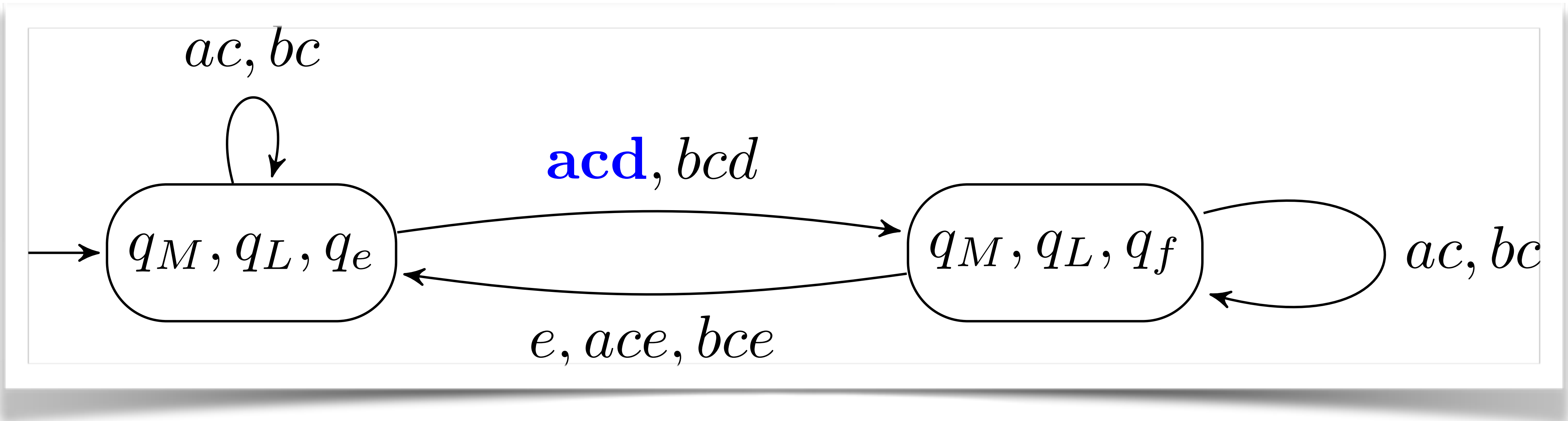
Exercise: compose colouring tables





Reo Connector Colouring

Dave Clarke, David Costa, and Farhad Arbab. Connector colouring I: Synchronisation and context dependency



Port and Constraint Automata

Christel Baier, Marjan Sirjani, Farhad Arbab, Jan Rutten. Modeling Component Connectors in Reo by Constraint Automata. 2004

Christian Koehler and Dave Clarke. Decomposing Port Automata. 2009

Connector behaviour (statefull)

- Dataflow behaviour is discrete in time: it can be observed and snapshots taken at a pace fast enough to obtain (at least) a snapshot as often as the configuration of the connector changes
- At each time unit the connector performs an evaluation step: it evaluates its configuration and according to its interaction constraints changes to another (possibly different) configuration
- A connector can fire multiple ports in the same evaluation step

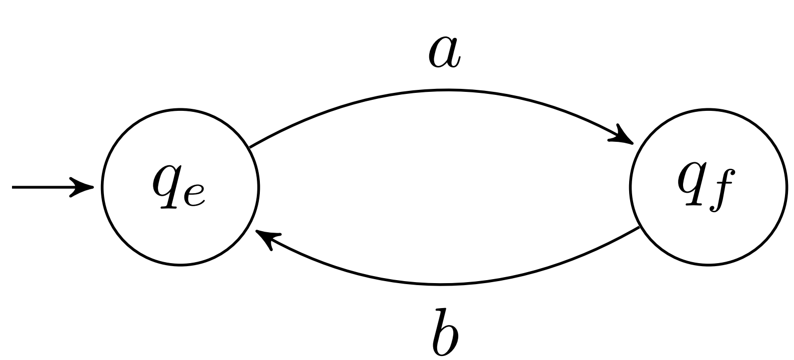
Port Automata

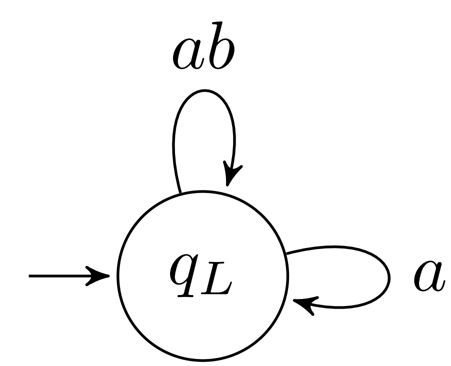
$\mathcal{A} = (\mathcal{Q}, \mathcal{N}, \rightarrow, \mathcal{Q}_0)$

$\mathcal{Q}$	set of states
$\mathcal{N}$	a set of ports $\mathcal{N}$
$\rightarrow \;\subseteq \mathcal{Q} \times 2^{\mathcal{N}} \times \mathcal{Q}$	a transition relation
$\mathcal{Q}_0 \subseteq \mathcal{Q}$	a set of initial states

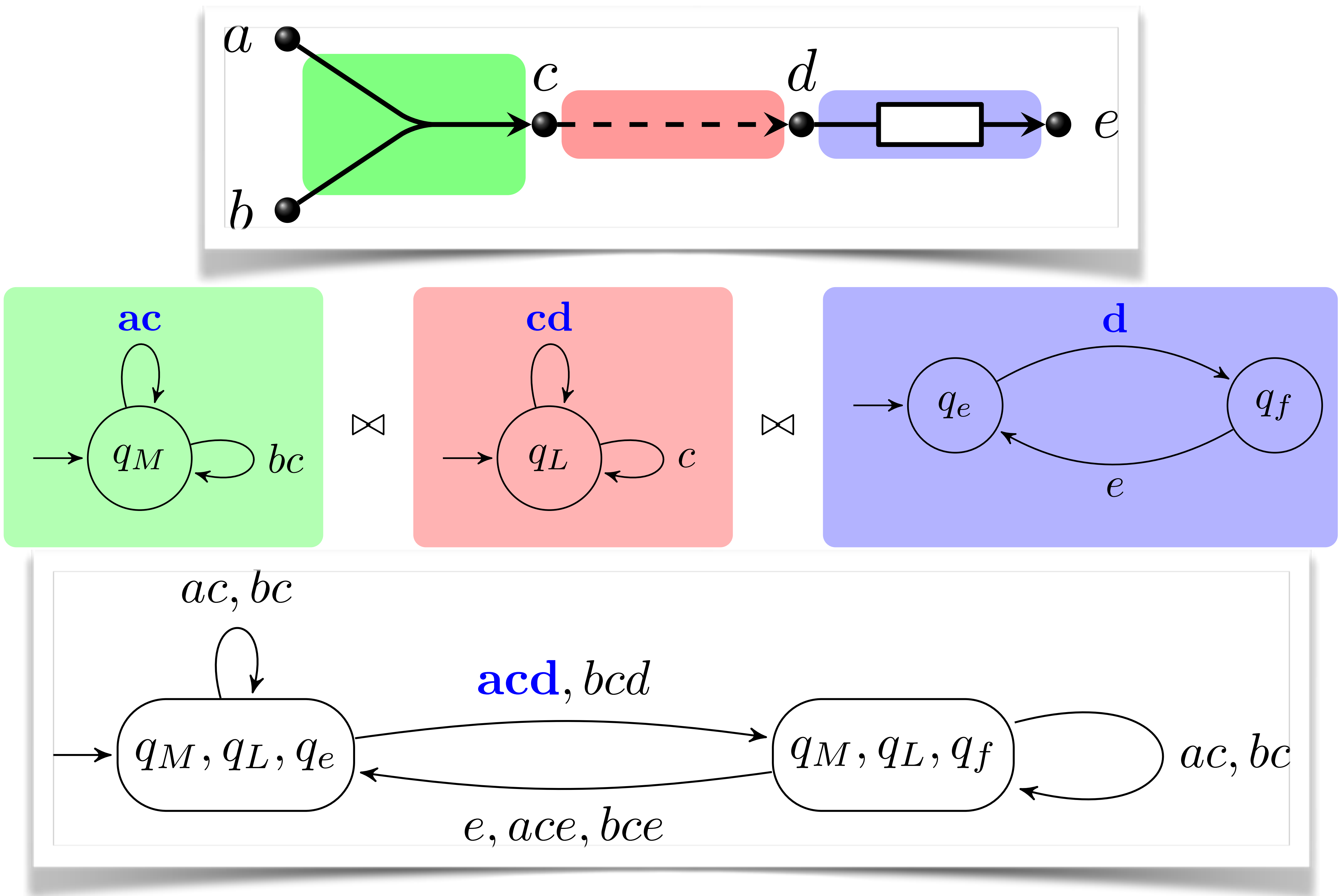
transitions must have a non-empty set of ports!

examples:

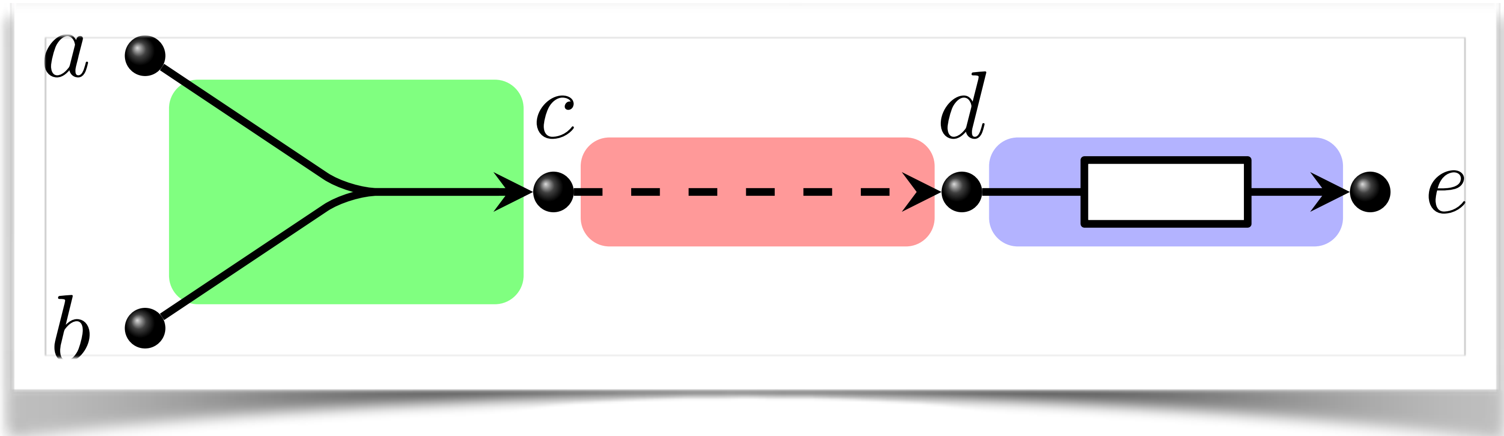


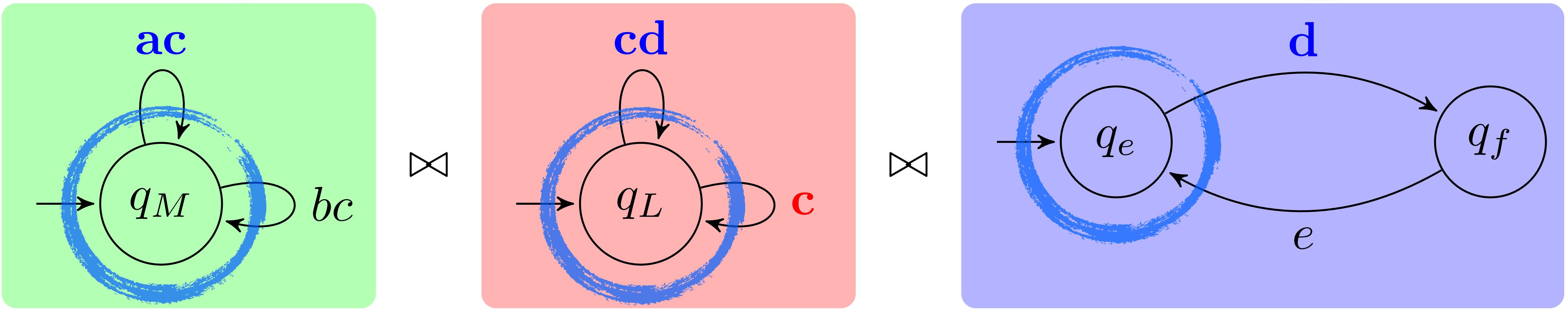


Composing steps



Composing steps





$$\mathbf{ac} \bowtie \mathbf{cd} \bowtie \mathbf{d} \quad = \quad \mathbf{acd}$$

$$\mathbf{ac} \bowtie \mathbf{c} \bowtie \mathbf{d} \quad = \quad \perp$$

Composition - formally

Definition 2. The product of two port automata $\mathcal{A}_1 = (\mathcal{Q}_1, \mathcal{N}_1, \longrightarrow_1, \mathcal{Q}_{0,1})$ and $\mathcal{A}_2 = (\mathcal{Q}_2, \mathcal{N}_2, \longrightarrow_2, \mathcal{Q}_{0,2})$ is defined by

$$\mathcal{A}_1 \bowtie \mathcal{A}_2 = (\mathcal{Q}_1 \times \mathcal{Q}_2, \mathcal{N}_1 \cup \mathcal{N}_2, \longrightarrow, \mathcal{Q}_{0,1} \times \mathcal{Q}_{0,2})$$

where $\longrightarrow$ is defined by the rule

$$\frac{q_1 \xrightarrow{N_1}_1 p_1 \quad q_2 \xrightarrow{N_2}_1 p_2 \quad N_1 \cap \mathcal{N}_2 = N_2 \cap \mathcal{N}_1}{\langle q_1, q_2 \rangle \xrightarrow{N_1 \cup N_2} \langle p_1, p_2 \rangle}$$

and the following and its symmetric rule

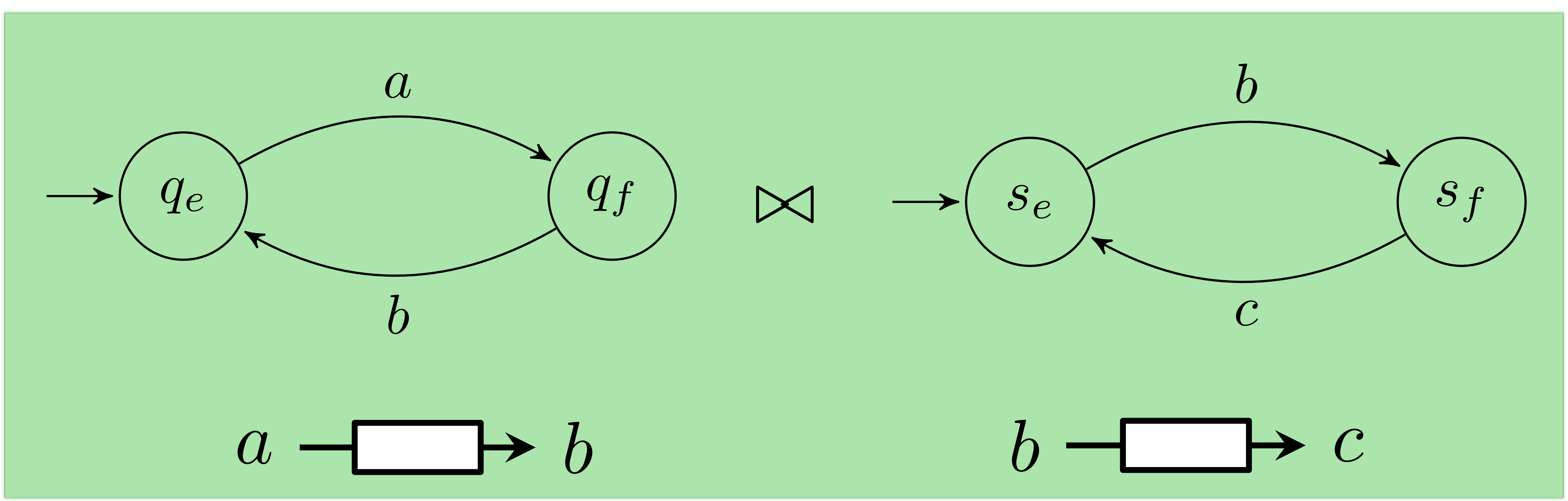
$$\frac{q_1 \xrightarrow{N_1}_1 p_1 \quad N_1 \cap \mathcal{N}_2 = \emptyset}{\langle q_1, q_2 \rangle \xrightarrow{N_1} \langle p_1, q_2 \rangle}$$

Formalize and compose

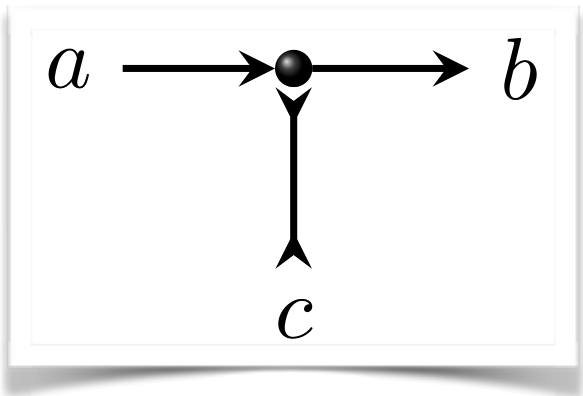
$$\frac{q_1 \xrightarrow{N_1}_1 p_1 \quad q_2 \xrightarrow{N_2}_1 p_2 \quad N_1 \cap \mathcal{N}_2 = N_2 \cap \mathcal{N}_1}{\langle q_1, q_2 \rangle \xrightarrow{N_1 \cup N_2} \langle p_1, p_2 \rangle}$$

$$\mathcal{A} = (\mathcal{Q}, \mathcal{N}, \rightarrow, \mathcal{Q}_0)$$

$$\frac{q_1 \xrightarrow{N_1}_1 p_1 \quad N_1 \cap \mathcal{N}_2 = \emptyset}{\langle q_1, q_2 \rangle \xrightarrow{N_1} \langle p_1, q_2 \rangle}$$

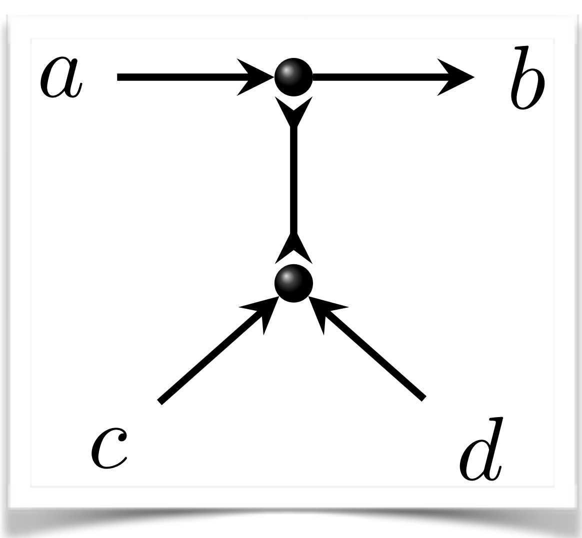


Examples I



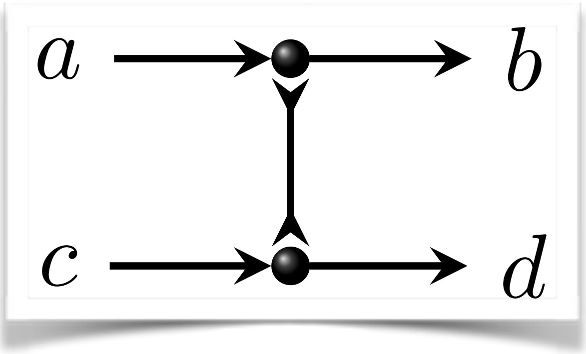
Flow regulator

"b" controls flow from "a" to "c"



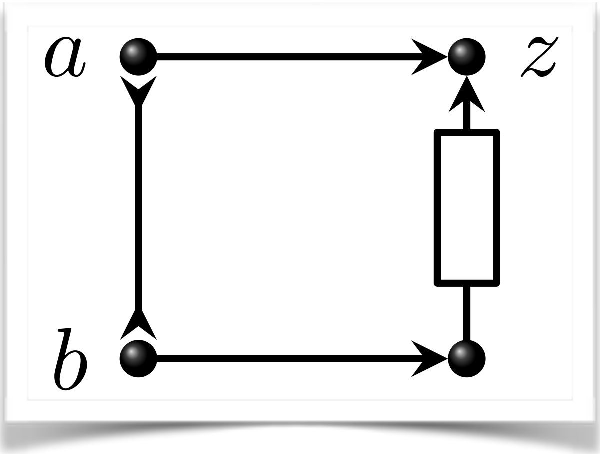
data flows from "a" to "b" ONLY if either "c" or "d" have data

Examples II



Synchronising barrier

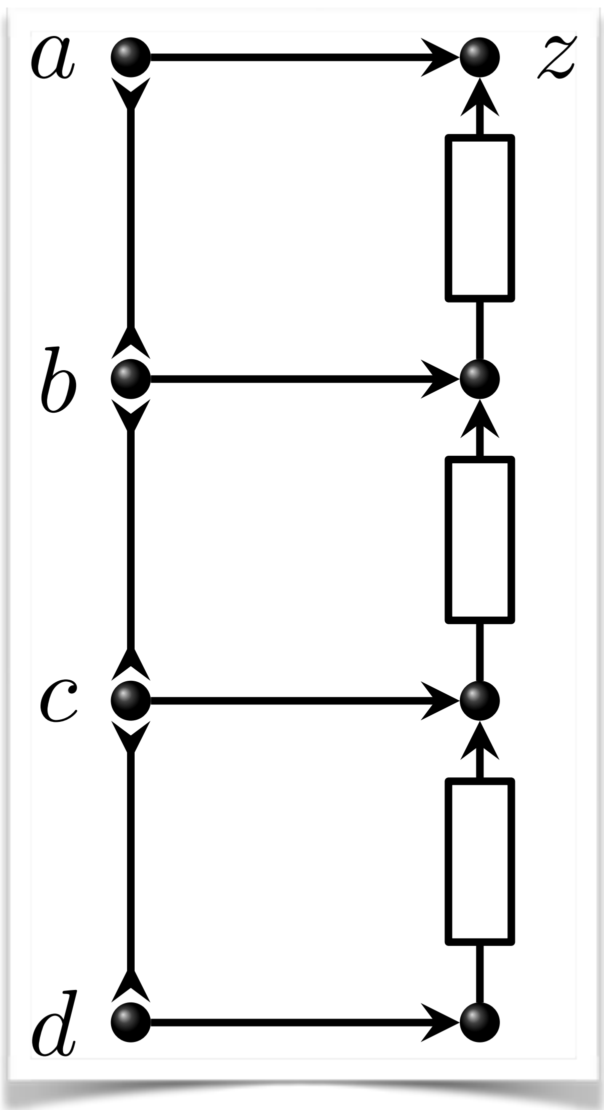
data flows "a" —> "b"
IFF
data flows "c" —> "d"



Alternator

data flows from "a"
and from "b" to "z",
alternating (+ extra
synch constraints)

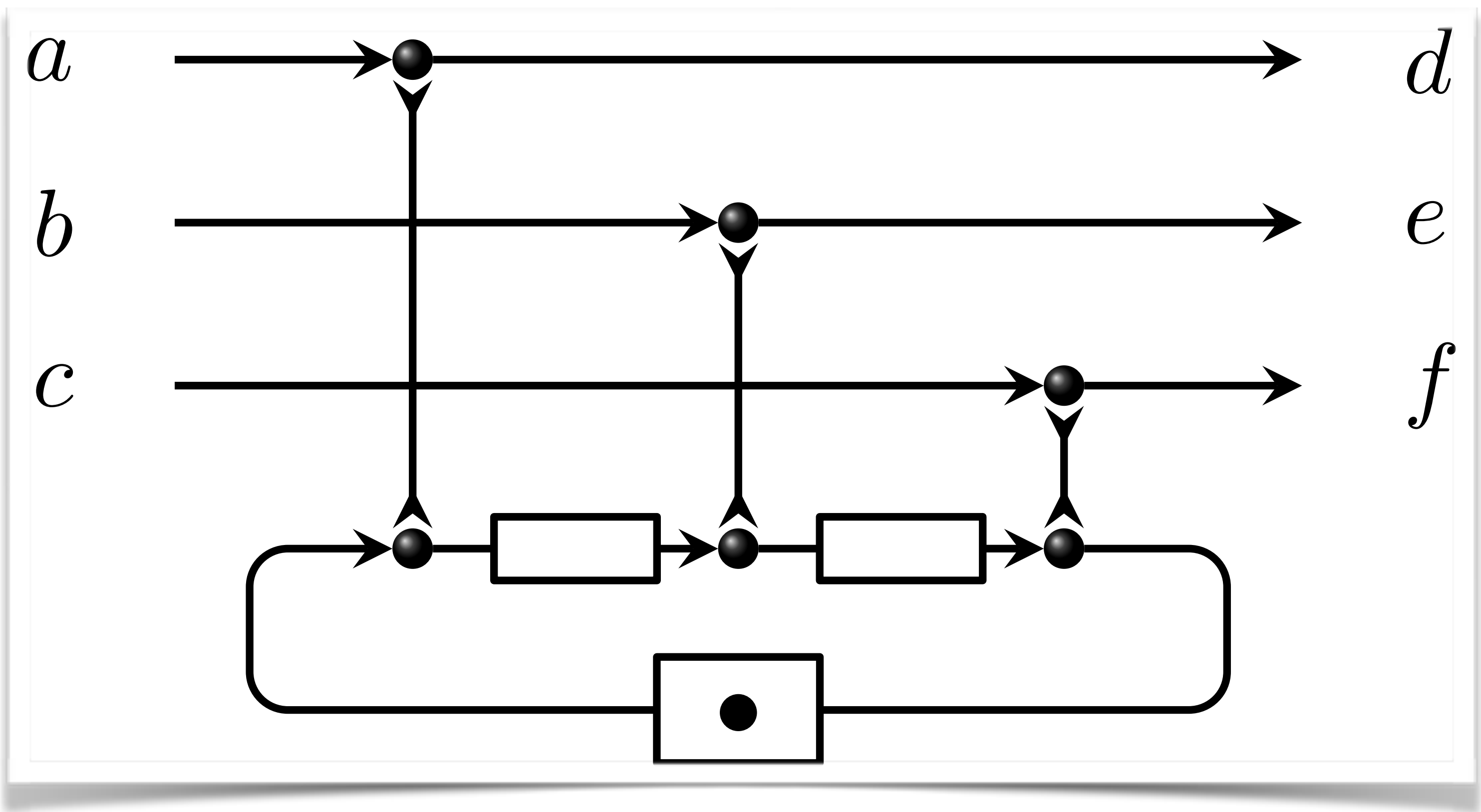
Examples III



N-Alternator

data flows from "a", "b", "c", and "d" to "z", alternating (+ extra synch constraints)

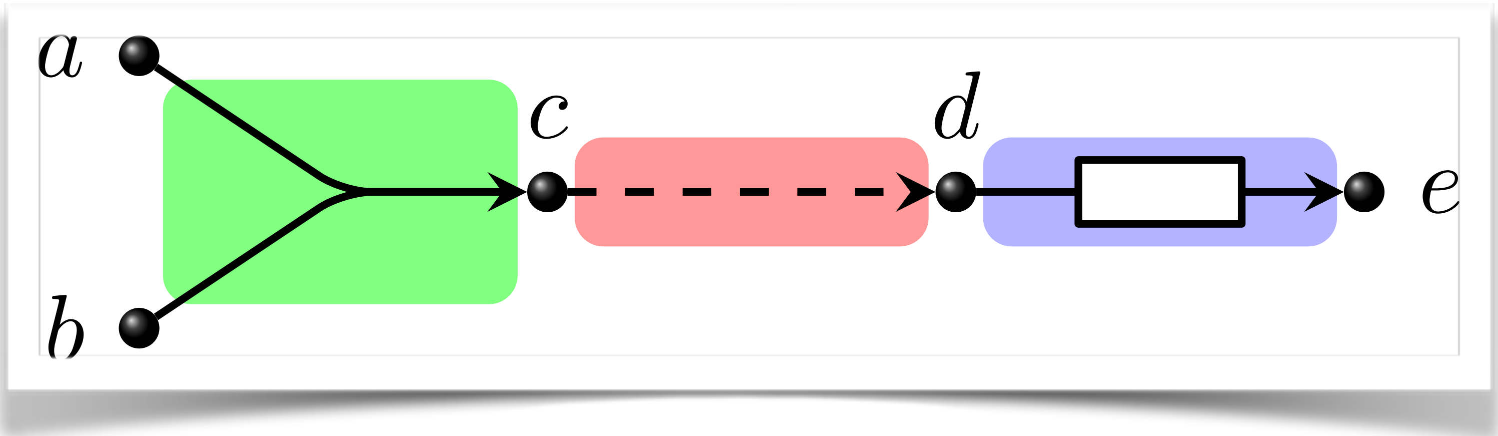
Examples IV

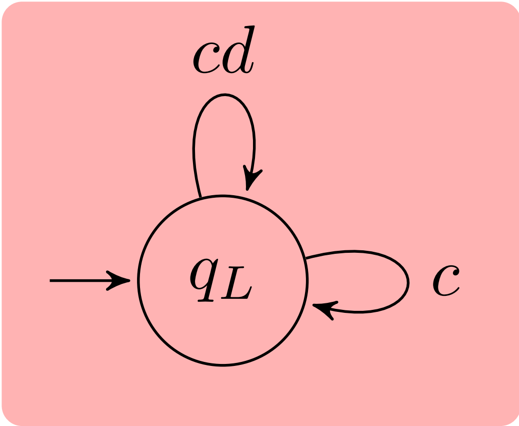


Sequencer

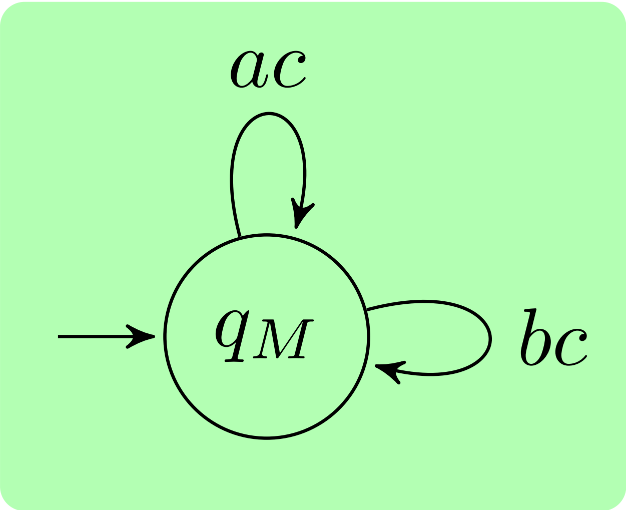
Data flows from "a" to "d", "b" to "e", and "c" to "f" alternating.

Reo in mCRL2



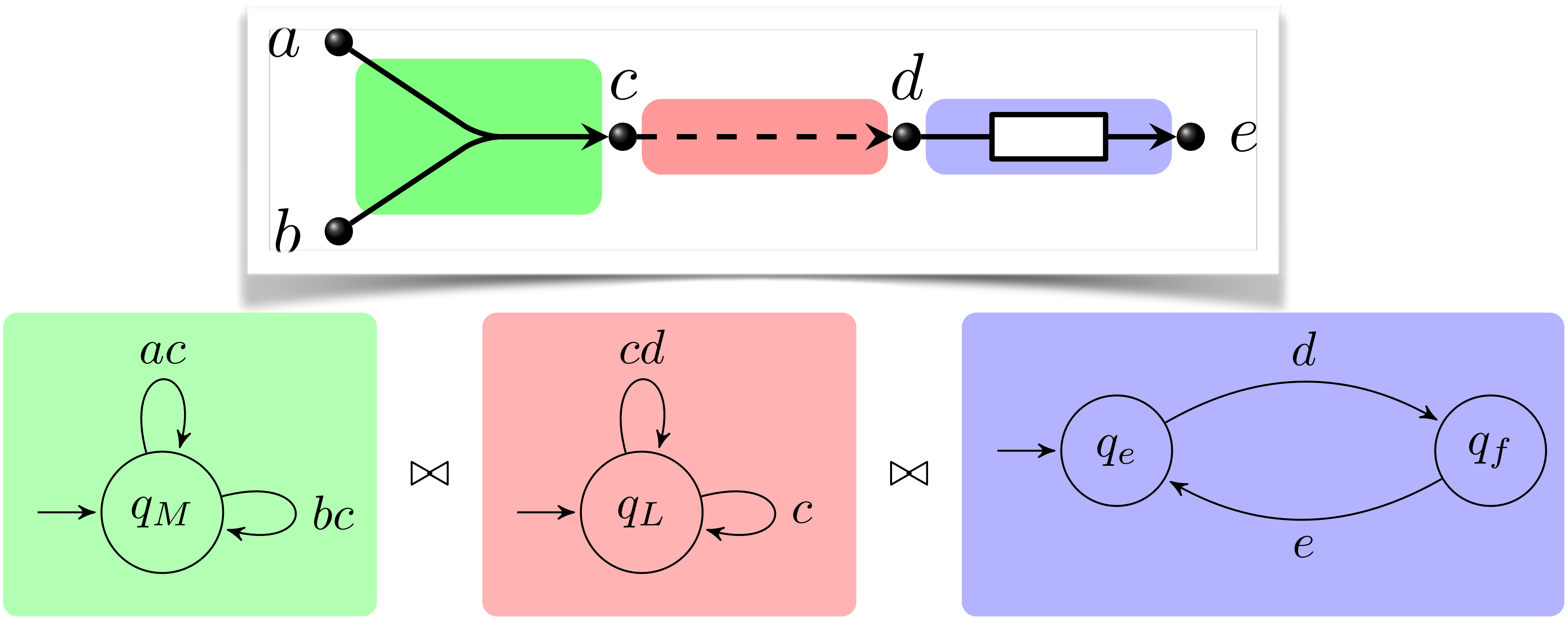


Lossy = ($c|d$ + c).Lossy



Merger = ($a|c$ + $b|c$).Merger

Reo in mCRL2



Conn = hide($\{c,d\}$,
block($\{c_1,c_2,d_1,d_2\}$,
comm($\{c_1|c_2$ -> c, $d_1|d_2$ -> $\mathbf{d}\}$,
Merger || Lossy || FIFO1)))

Can you prove?

colourings and port automata provide equivalent semantics

$$\mathcal{A}(C_1) = (Q_1, \mathcal{N}_1, \rightarrow_1, q_{0,1})$$
$$\mathcal{A}(C_2) = (Q_2, \mathcal{N}_2, \rightarrow_2, q_{0,2})$$

$\mathcal{CT}(C)$ – colouring table of C

$col(q \xrightarrow{P} q')$ – colouring associated to a transition

$$(\langle q_{0,1}, q_{0,2} \rangle \xrightarrow{P} \langle q_1, q_2 \rangle) \in \mathcal{A}(C_1) \bowtie \mathcal{A}(C_2)$$
$$\Rightarrow$$
$$col(\langle q_{0,1}, q_{0,2} \rangle \xrightarrow{P} \langle q_1, q_2 \rangle) \in \mathcal{CT}(C_1) \bowtie \mathcal{CT}(C_2)$$

Can you prove? (more generically)

colourings and port automata provide equivalent semantics

$$\mathcal{A} = (\mathcal{Q}, \mathcal{N}, \rightarrow, \{q_0\})$$

$$(q_0 \xrightarrow{P} q) \in \mathcal{A}(C)$$

$$\Rightarrow$$

$$col(P, \mathcal{N}) \;\in\; \mathcal{CT}(C)$$

Constraint Automata

Automata labelled by

- a **data constraint** which represents a set of data assignments to port names

$$g \quad ::= \quad \mathsf{true} \;\mid\; d_A = v \;\mid\; g_1 \vee g_2 \;\mid\; \neg g$$

Note: other constraints, such as

$$d_A = d_B \;\stackrel{\text{abv}}{=}\; \vee_{d \in Data}(d_A = d \wedge d_B = d)$$

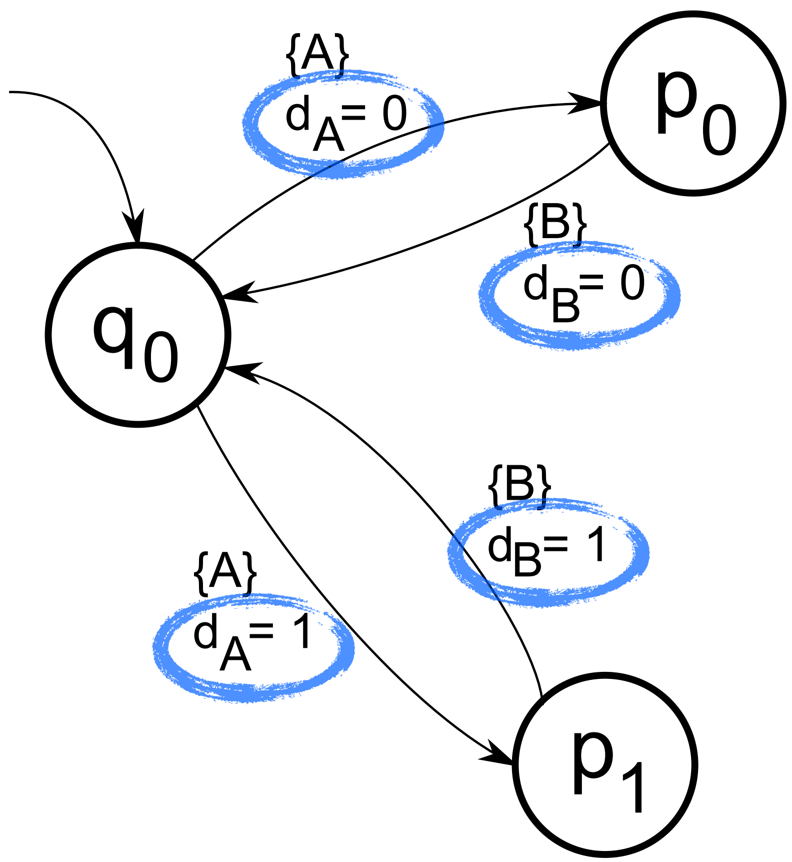
are derived.

- a **name set** which represents the set of port names at which IO can occur

States represent the configurations of the corresponding connector, while transitions encode its maximally-parallel stepwise behaviour.

Constraint Automata

Example: FIFO1



Constraint Automata - Definition

$\mathcal{A} = (\mathcal{Q}, \mathcal{N}, \rightarrow, \mathcal{Q}_0)$

$\mathcal{Q}$	set of states
$\mathcal{N}$	a set of ports $\mathcal{N}$
$\mathcal{Q}_0 \subseteq \mathcal{Q}$	a set of initial states
$\rightarrow \;\subseteq \mathcal{Q} \times 2^{\mathcal{N}} \times DC \times \mathcal{Q}$	a transition relation such that $\xrightarrow{P,g}$ iff

1. $P \neq \emptyset$
2. $g \in DC(P, Data)$

($DC(P, Data)$ is the set of data constraints over Data and P)

Constraint Automata - Definition

$s \xrightarrow{P,g} s'$ iff

1. $P \neq \emptyset$
2. $g \in DC(P, Data)$

in configuration s, ports in P can perform IO operations which meet guard g and lead to s'

transitions fire only if data occurs at a (set of) ports P

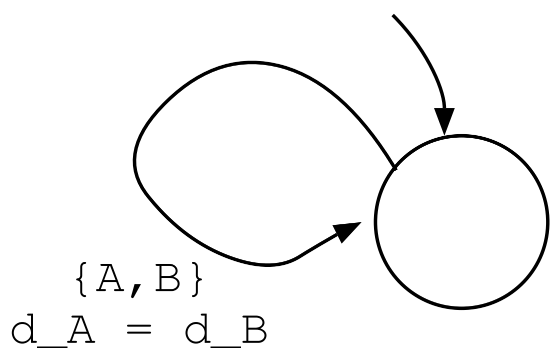
behaviour depends only on observed data (not on future evolution)

Constraint Automata as a semantics for Reo

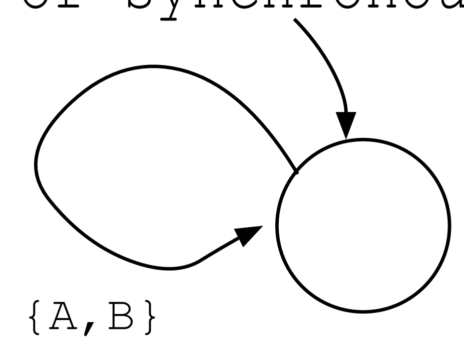
- cannot capture context-awareness [Baier, Sirjani, Arbab, Rutten 2006], but forms the basis for more elaborated models (eg, Reo automata)
- captures all behaviour alternatives of a connector; useful to generate a state-machine implementing the connector's behaviour
- basis for several tools, including the model checker Vereofy [Kluppelholz, Baier 2007]

Constraint Automata - Reo connectors

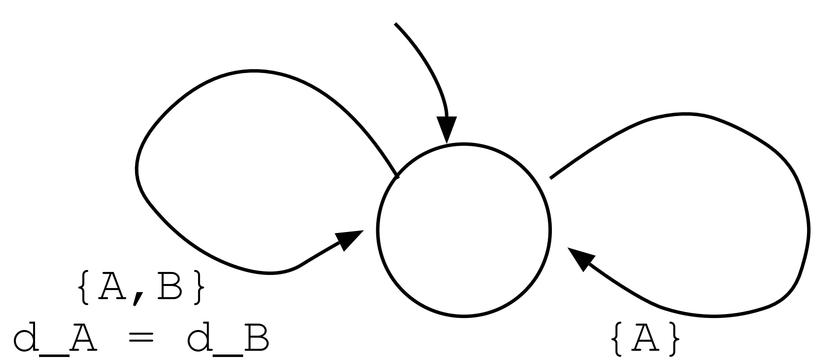
synchronous channel



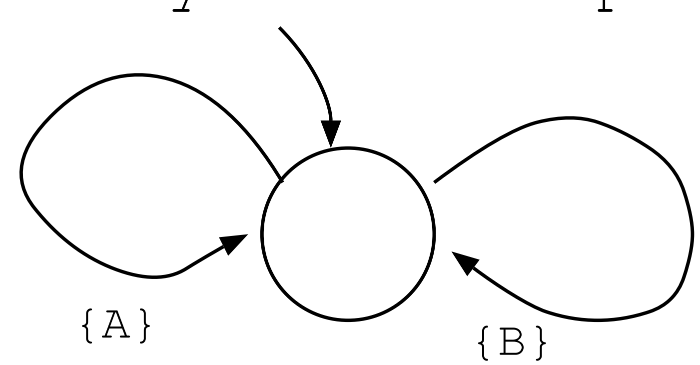
synchronous drain
or synchronous spout



lossy synchronous channel



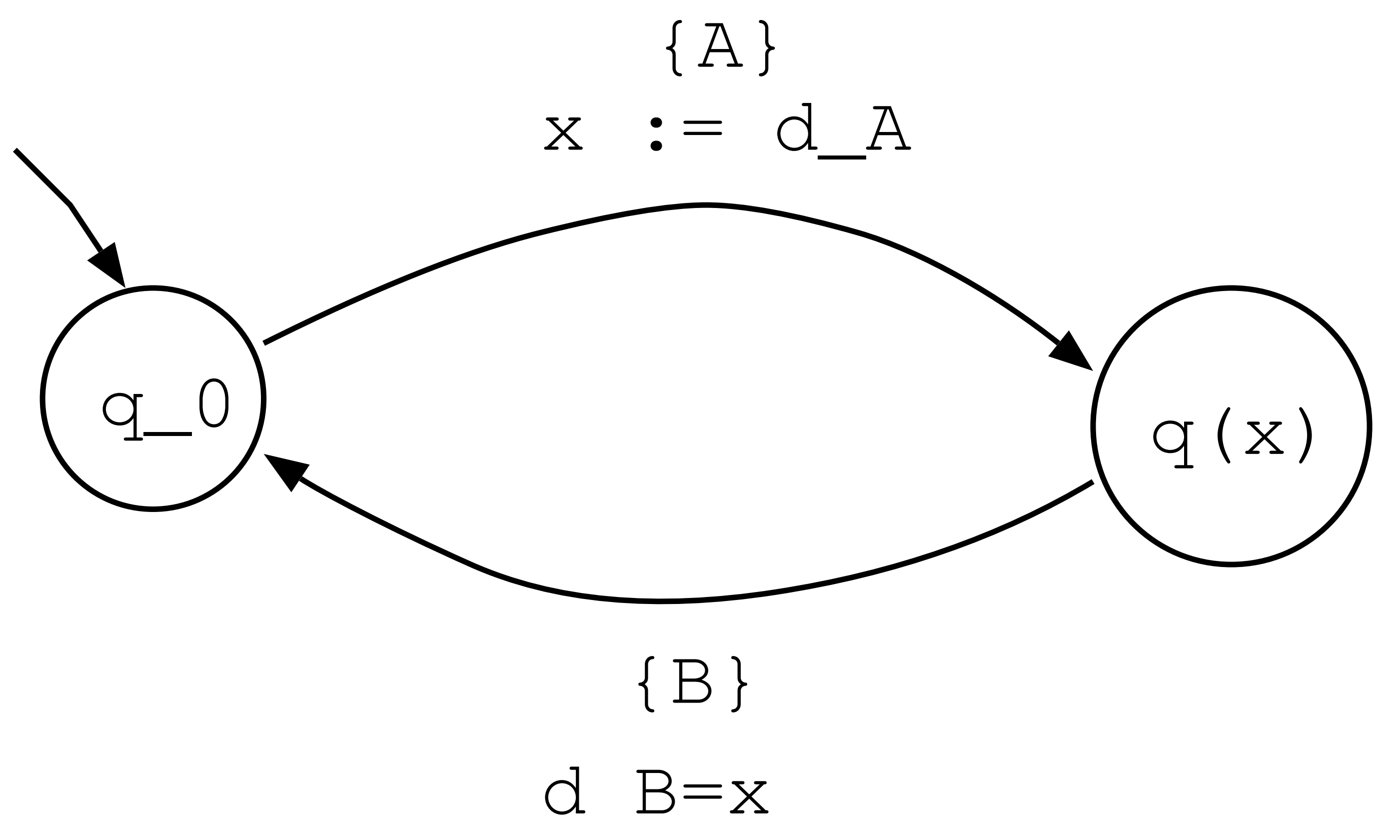
asynchronous drain
or asynchronous spout



Parameterised constraint automata

States are parametric on data values ... therefore capturing complex constraint automata emerging form data-dependencies

Example: 1 bounded FIFO



Composing constraint automata

Definition 4.1 [*Product-automaton*] The product-automaton of the two constraint automata $\mathcal{A}_1 = (Q_1, \mathcal{N}ames_1, \longrightarrow_1, Q_{0,1})$ and $\mathcal{A}_2 = (Q_2, \mathcal{N}ames_2, \longrightarrow_2, Q_{0,2})$, is:

$$\mathcal{A}_1 \bowtie \mathcal{A}_2 \;=\; (Q_1 \times Q_2, \mathcal{N}ames_1 \cup \mathcal{N}ames_2, \longrightarrow, Q_{0,1} \times Q_{0,2})$$

where $\longrightarrow$ is defined by the following rules:

$$\frac{q_1 \xrightarrow{N_1, g_1}_1 p_1, \quad q_2 \xrightarrow{N_2, g_2}_2 p_2, \quad N_1 \cap \mathcal{N}ames_2 = N_2 \cap \mathcal{N}ames_1}{\langle q_1, q_2 \rangle \xrightarrow{N_1 \cup N_2, g_1 \wedge g_2} \langle p_1, p_2 \rangle}$$

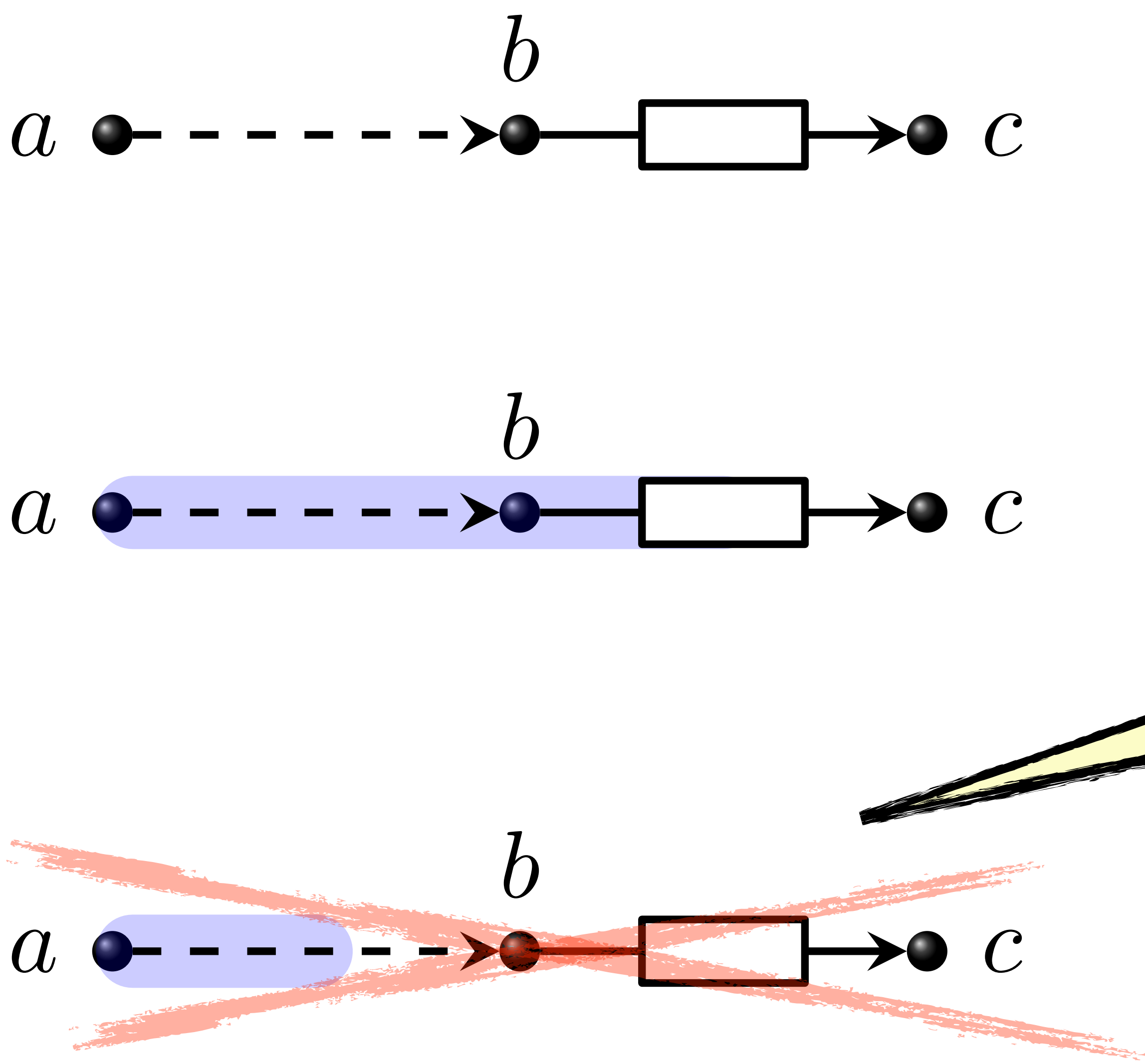
and

$$\frac{q_1 \xrightarrow{N, g}_1 p_1, \quad N \cap \mathcal{N}ames_2 = \emptyset}{\langle q_1, q_2 \rangle \xrightarrow{N, g} \langle p_1, q_2 \rangle}$$

You are here

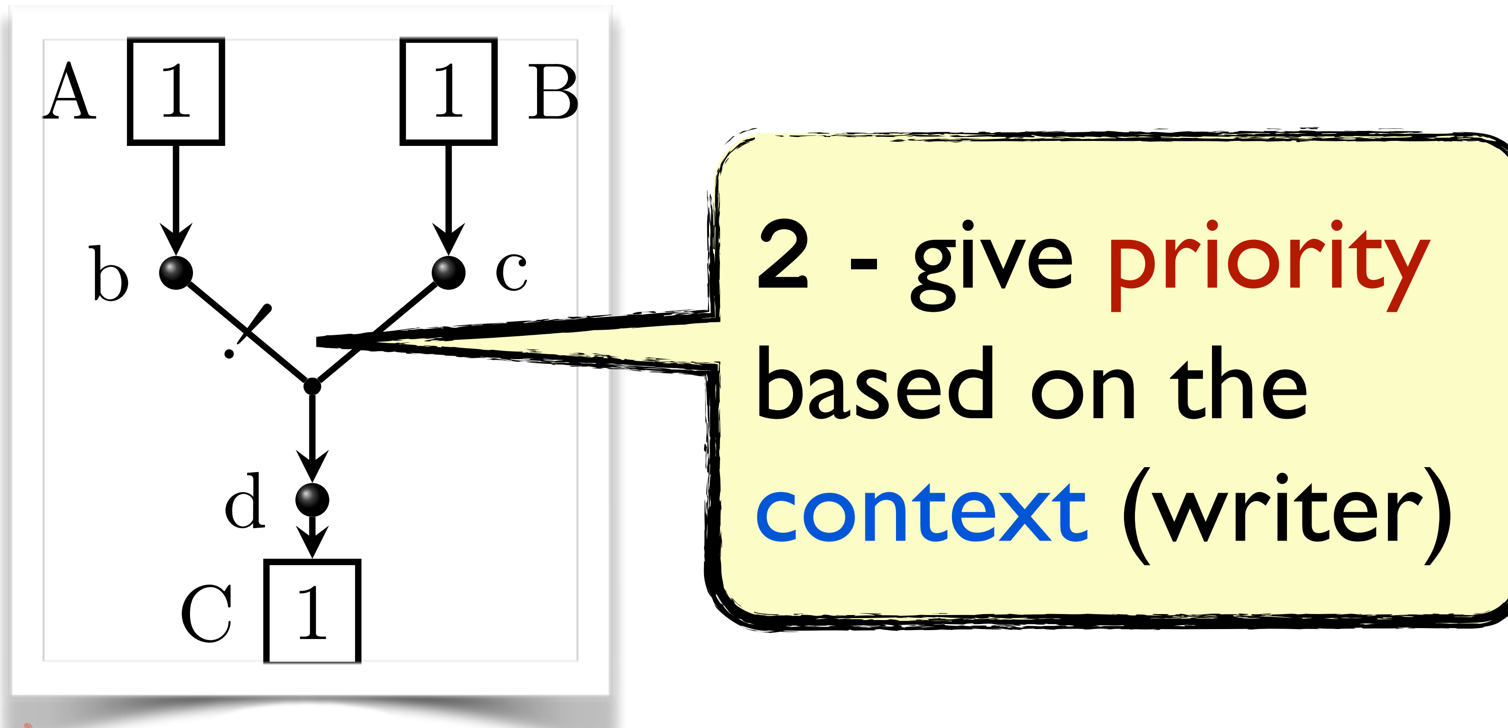
Formalism	Synchr. 1	Data 2	Context 3	Partial 4
Connector Colouring	CC2	-	CC3	PCC
Automata	**(Process Algebra)** Port Automata	Constraint Automata	-	-
Constraints	✓	✓	✓	✓

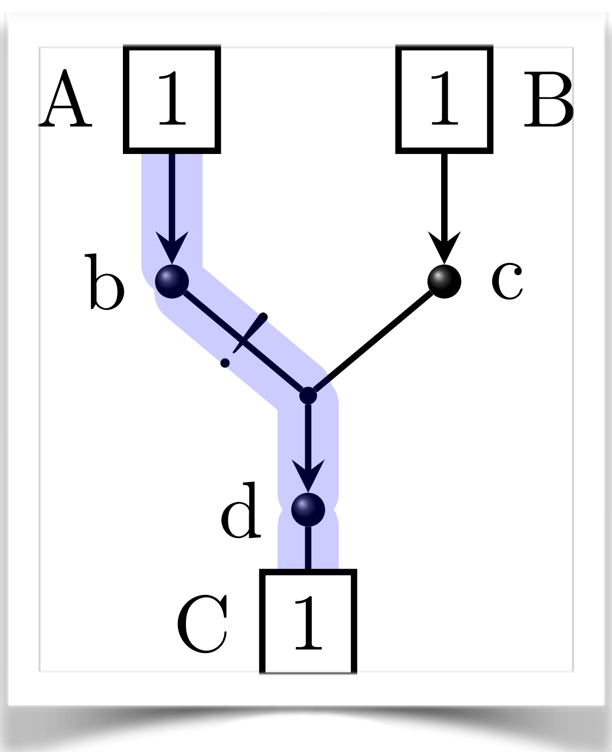
2 reasons for context

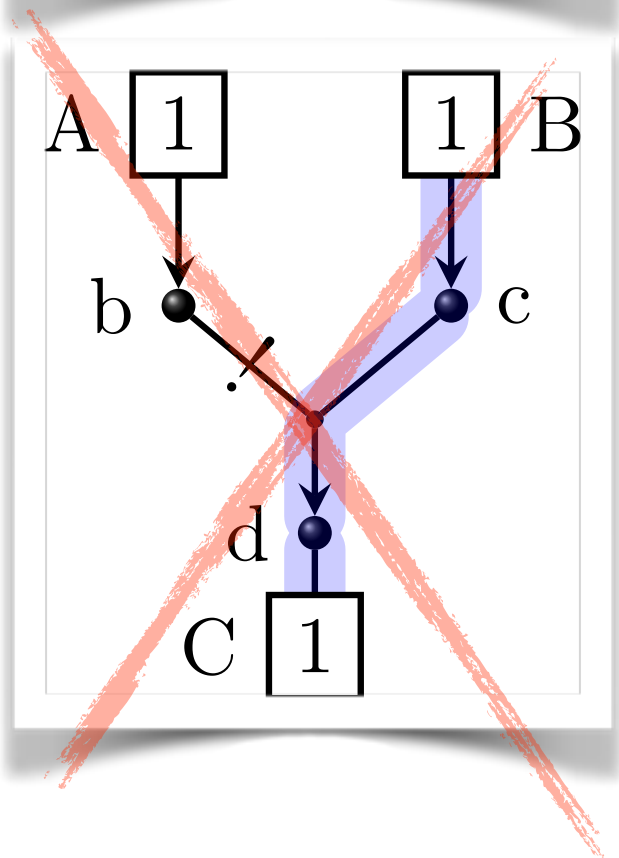


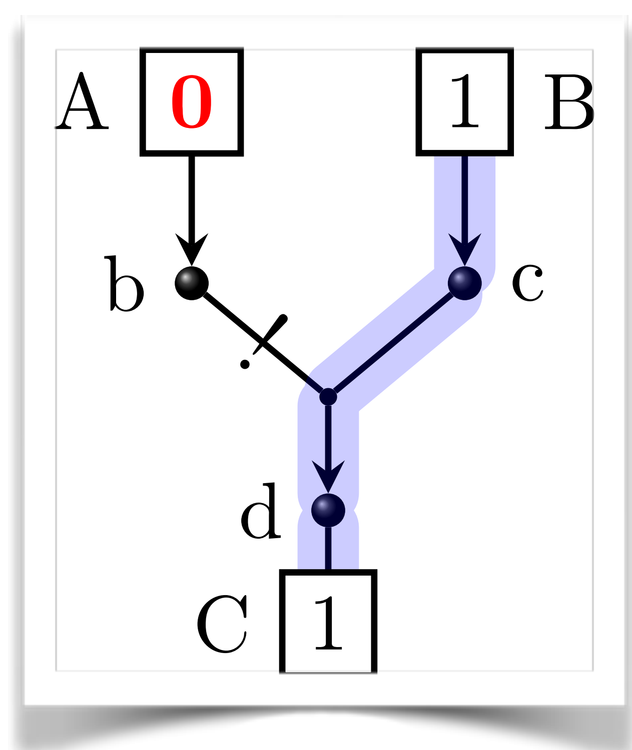
1 - avoid data loss when the context (FIFO) can receive the data.

2 reasons for context







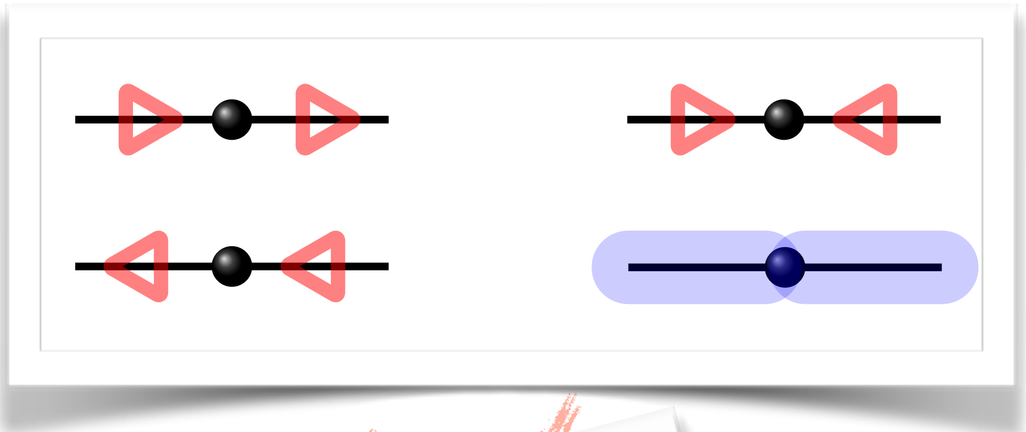


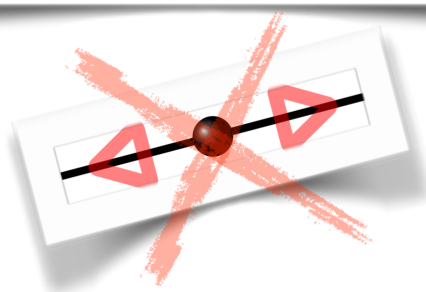
Context = 3 colours

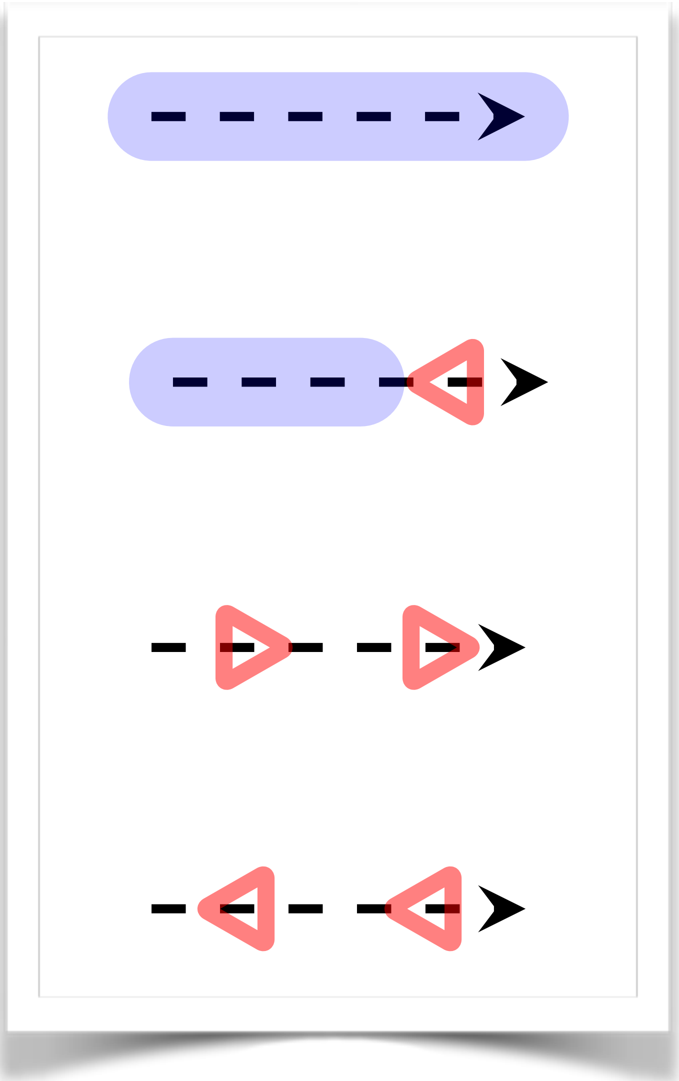
- *Colouring*:

  End → {Flow, GiveReason, GetReason}

- *Composition* = matching colours:





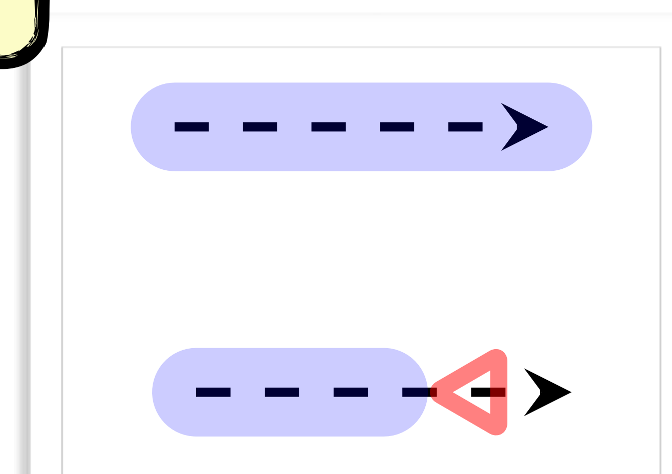


Context = 3 colours

- *Colo*

$$\text{End} = \{e_1, \ldots, e_n\} \cup \{\overline{e_1}, \ldots, \overline{e_n}\}$$

End → {Flow, GiveReason,GetReason}



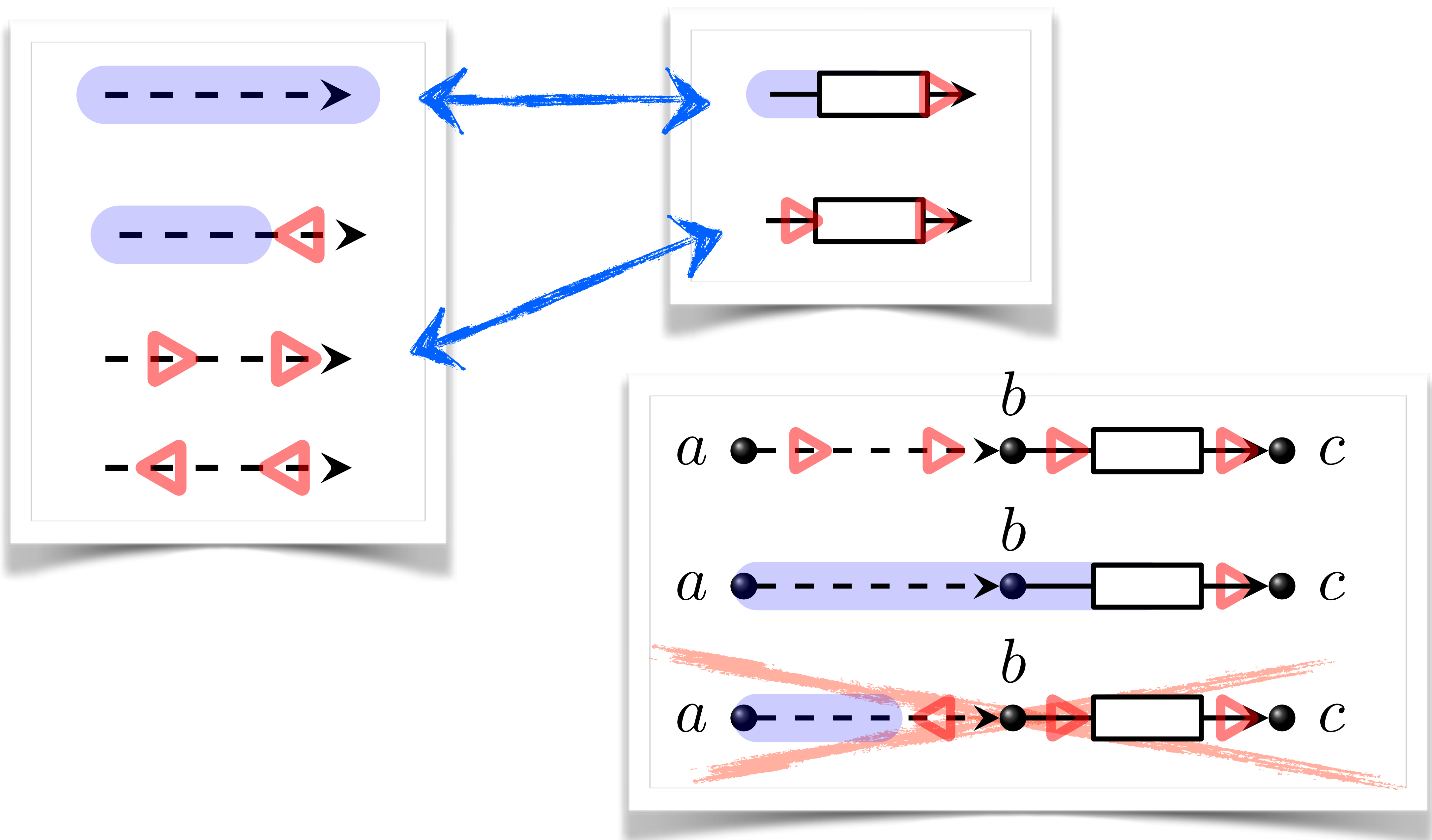
- *Composition* = matching colours:

$$CT_1 \bowtie CT_2 \quad = \quad \{cl_1 \bowtie cl_2 \mid cl_1 \in CT_1, cl_2 \in CT_2, cl_1 \frown cl_2\}$$

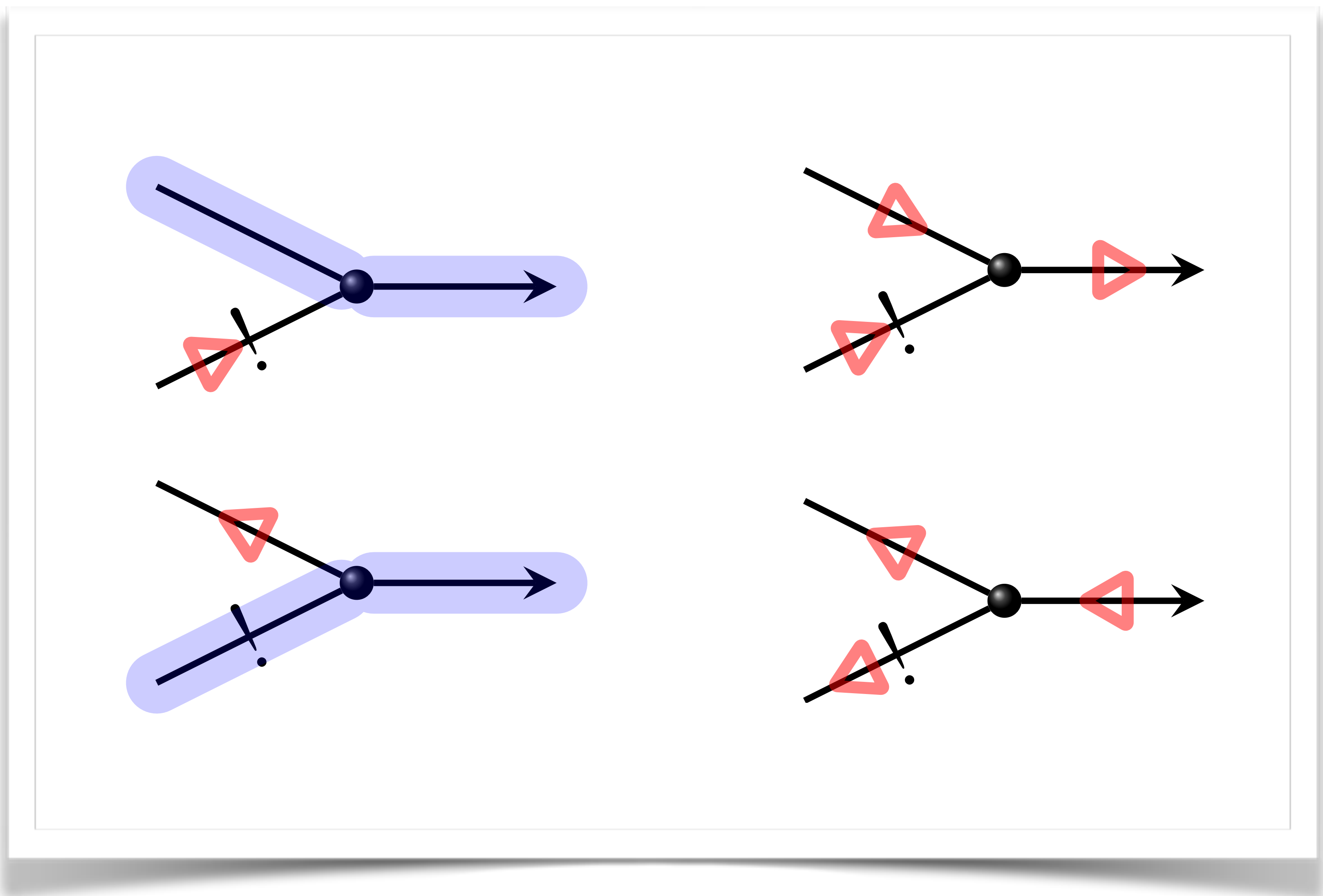
$$cl_1 \frown cl_2 = \forall e_1 \in \mathsf{dom}(cl_1) \cdot \forall e_2 \mathsf{dom}(cl_2) \cdot$$
$$e_1 = \overline{e}_2 \Rightarrow$$
$$(cl_1(e), cl_2(e)) \in \{(\triangleright, \triangleright), (\triangleleft, \triangleleft), (\triangleright, \triangleleft), \}$$

$$cl_1 \bowtie cl_2 = cl_1 \cup cl_2$$

Composition



Priority with 3 colours



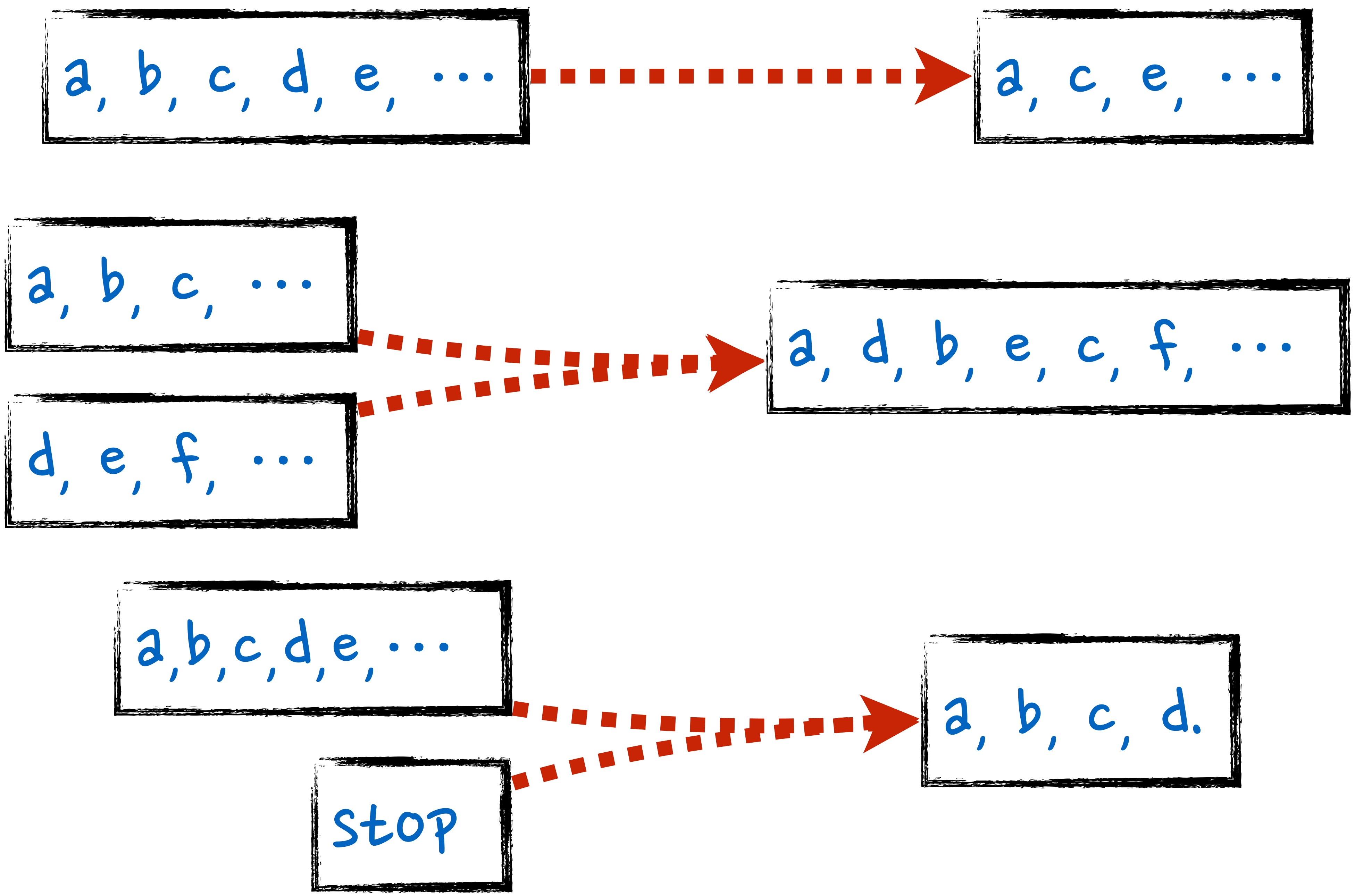
Connector colouring 3

- Compositional – composition operation is associative, commutative, and does not require post-processing.
- *Reasons* for the absence of flow are propagated.
- Expresses priority.
- 2 colours ⇔ constraint automata (without data)
- 3 colours: + expressive (⇔ intentional automata)

Build a connector

prefer fast FIFO

Build connectors



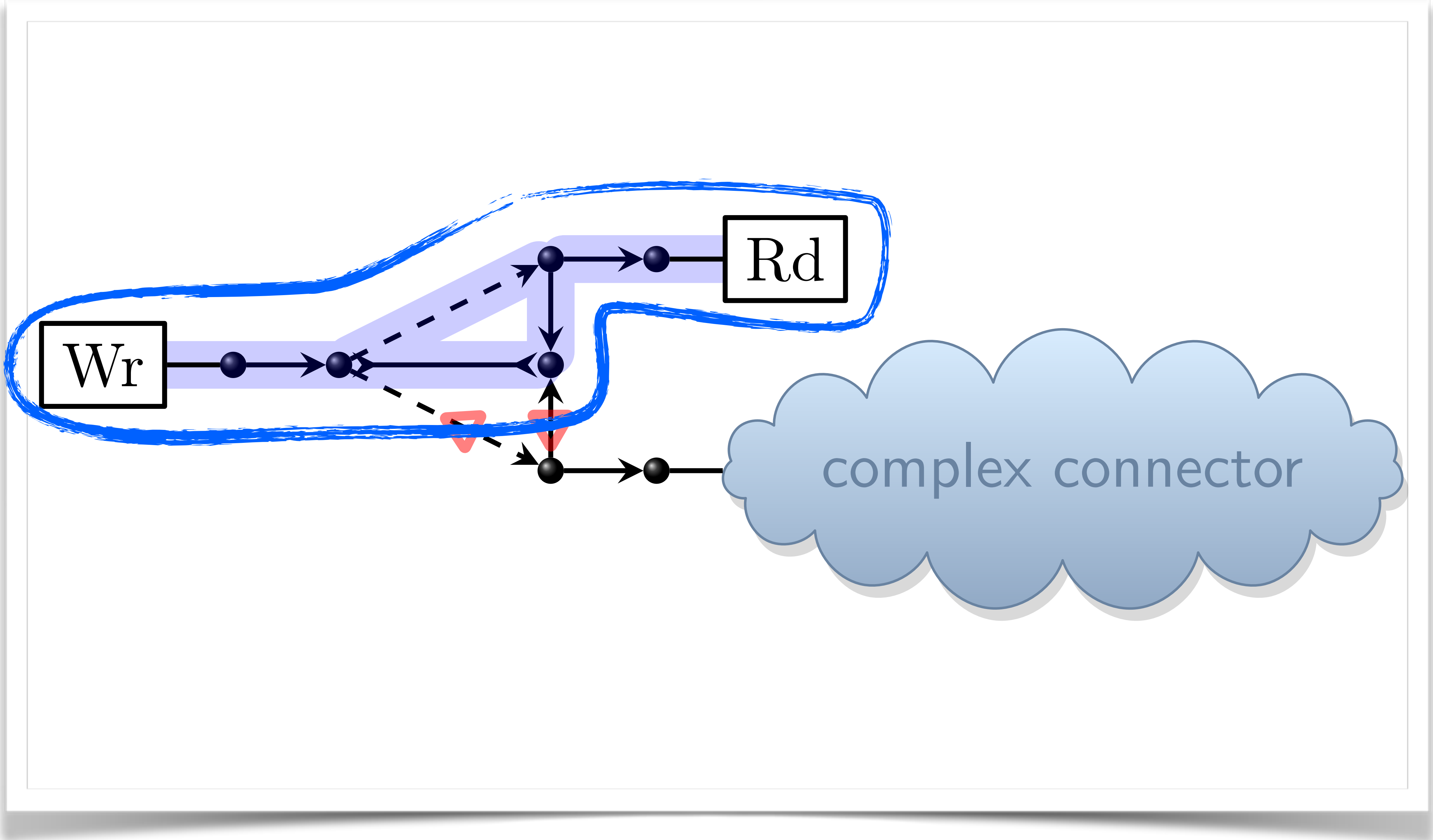
Outline

1. Visual semantics for Reo
   - Connector colouring (CC)[1]
2. Locality (concurrency)
   - Partial connector colouring (PCC)[2]
3. Constraints
   - SAT solving with data for Reo[3]

[1] Dave Clarke, David Costa, and Farhad Arbab. Connector colouring I: Synchronisation and context dependency

[2] Dave Clarke and José Proença. Partial connector colouring

[3] Dave Clarke, José Proença, Alexander Lazovik, and Farhad Arbab, Channel-based coordination via constraint satisfaction
José Proença, Dave Clarke, Interactive interaction constraints



Locality (concurrency)

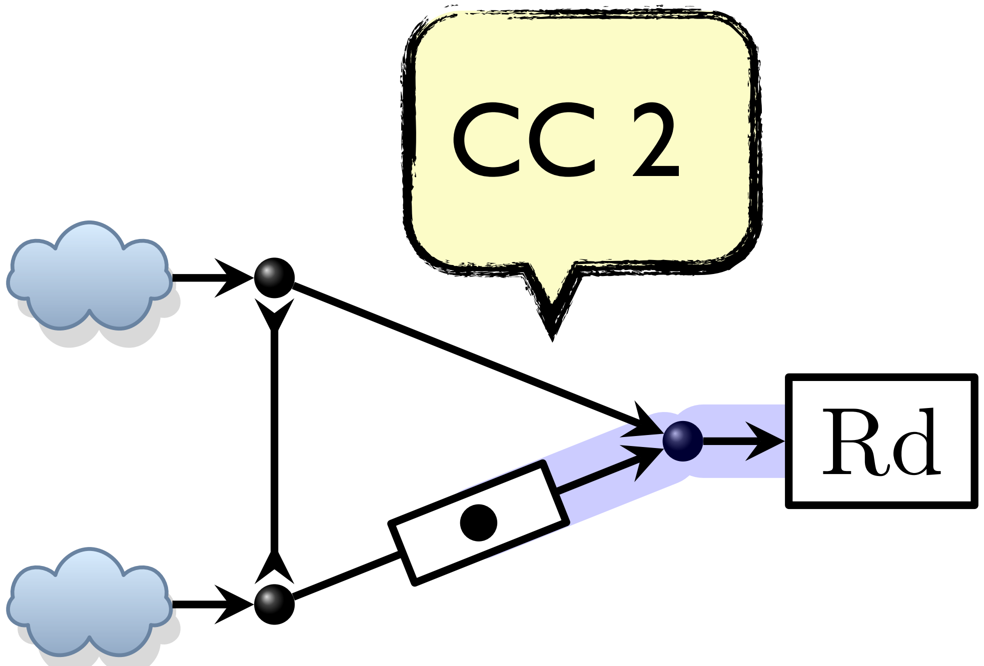
Motivation

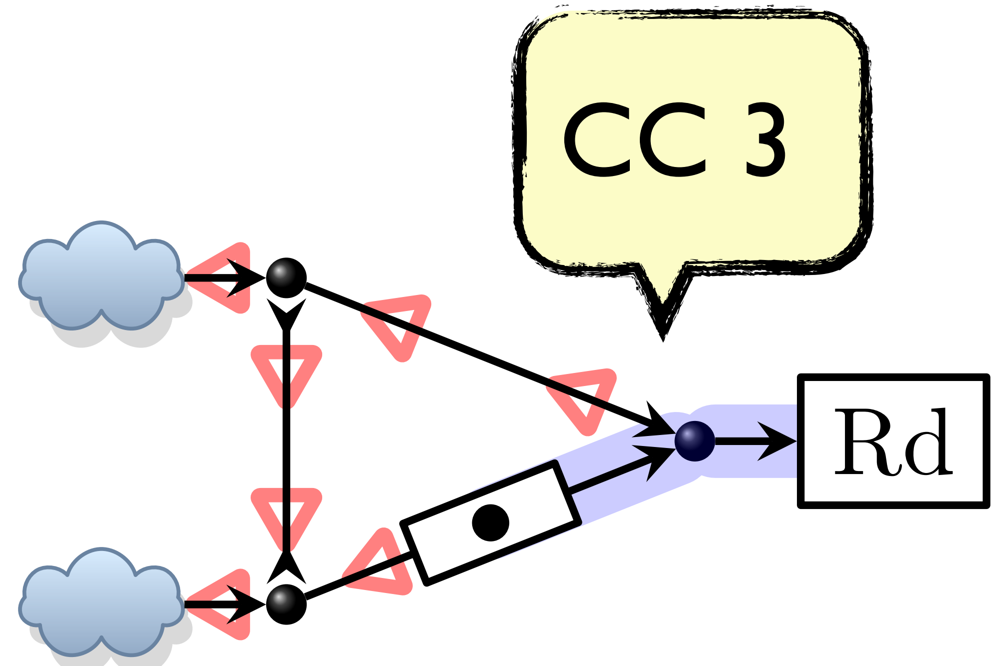
- Connector colouring is not optimal for distributed systems.
- All-or-nothing – all channels are needed to decide where data goes.
- Need to identify local flows that are not composed with the full connector.
- Model context dependency

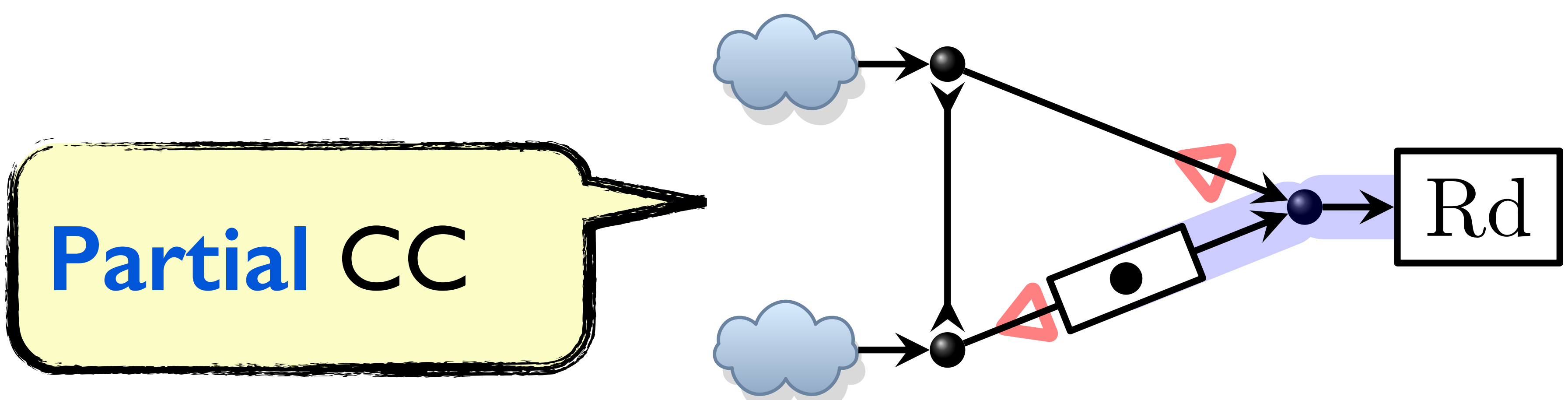
Problems

- 2 colours (or constraint automata):
  - assume primitives can make a *no-flow* step
- 3 colours:
  - cannot assume primitives have a no-flow colour – which direction would it be?
  - Idea?: add another no-flow colour, without direction, and assume all primitives have it...

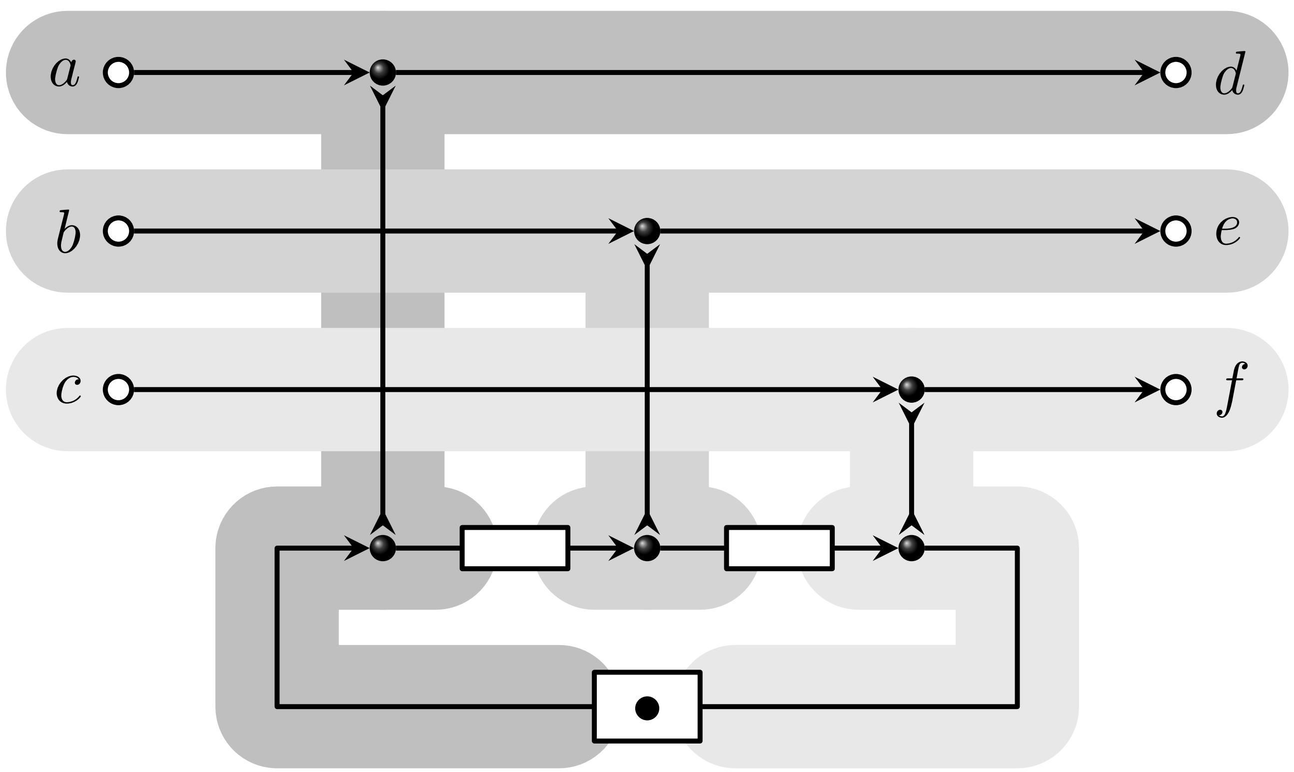
Example





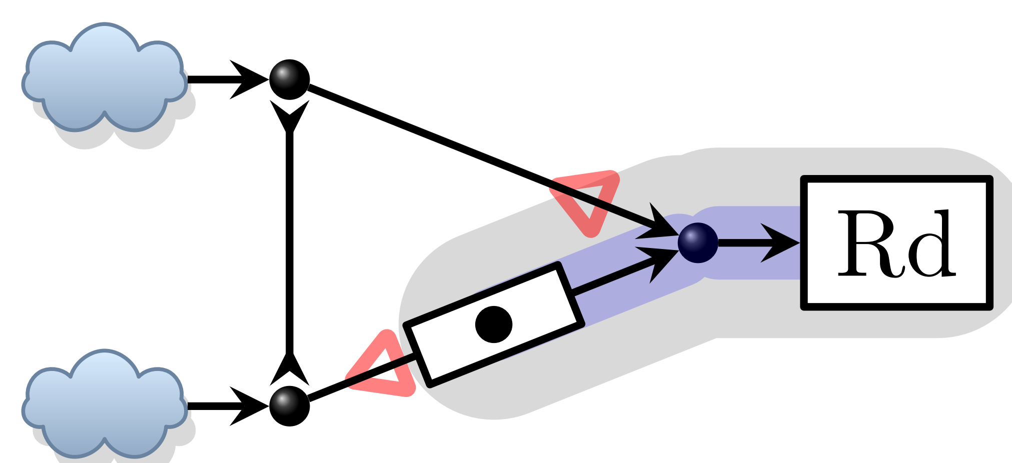


Synchronous regions



Static regions:
boundaries
=
FIFO's

Dynamic regions:
boundaries
=
GiveReason

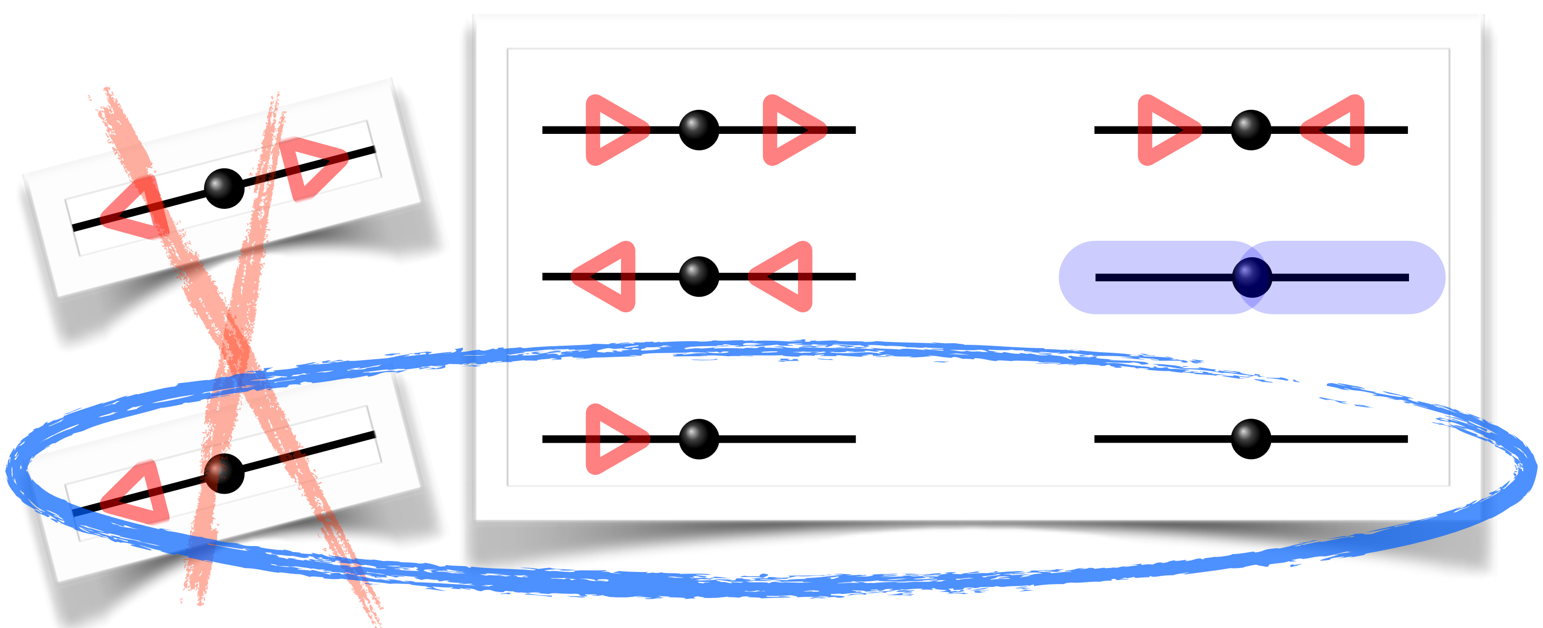


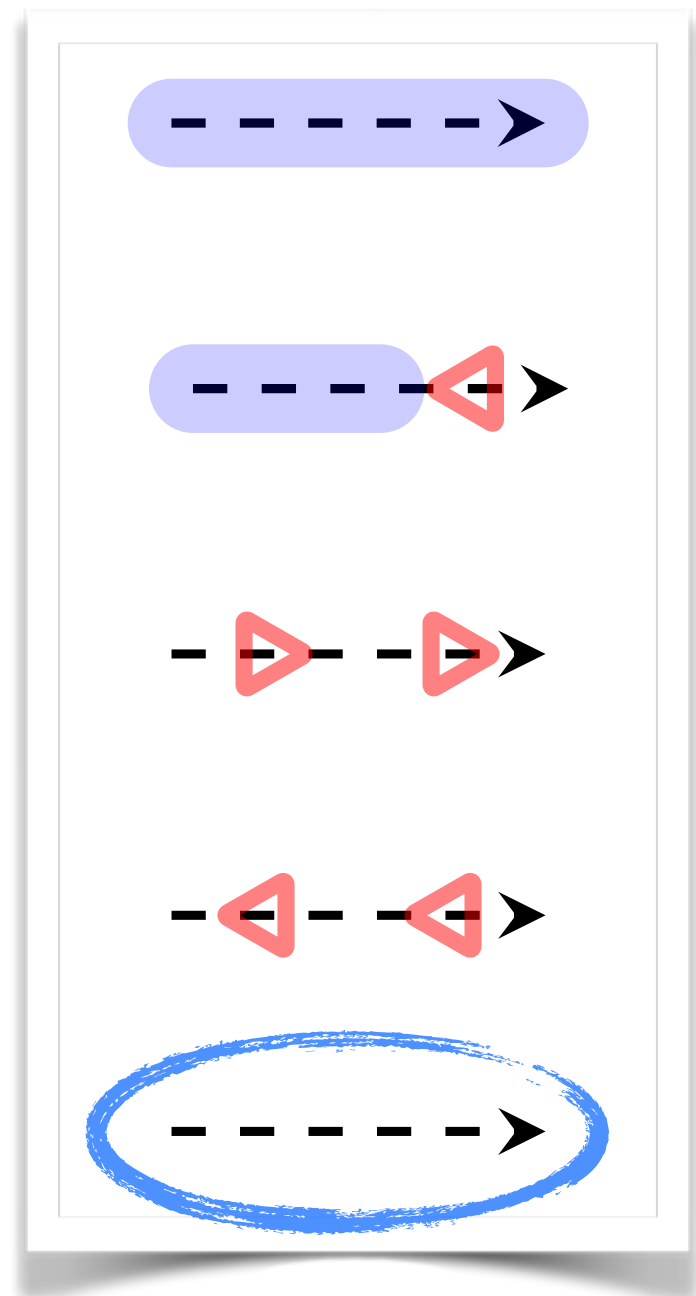
Partial connector colouring

- *Colouring*:

End $\rightharpoonup$ {Flow, GiveReason,GetReason}

- *Composition* = matching colours:





In practice

Shall I search now for a colouring?