Verification of Concurrent Recursive Programs


Verification of Concurrent Recursive Programs
(an unified view via split-width)
From her forthcoming PhD Thesis

Concurrent Recursive Programs

Concurrent Recursive Programs

Variables range over finite domains

Concurrent Recursive Programs

Variables range over finite domains

Functions can be recursive

Concurrent Recursive Programs

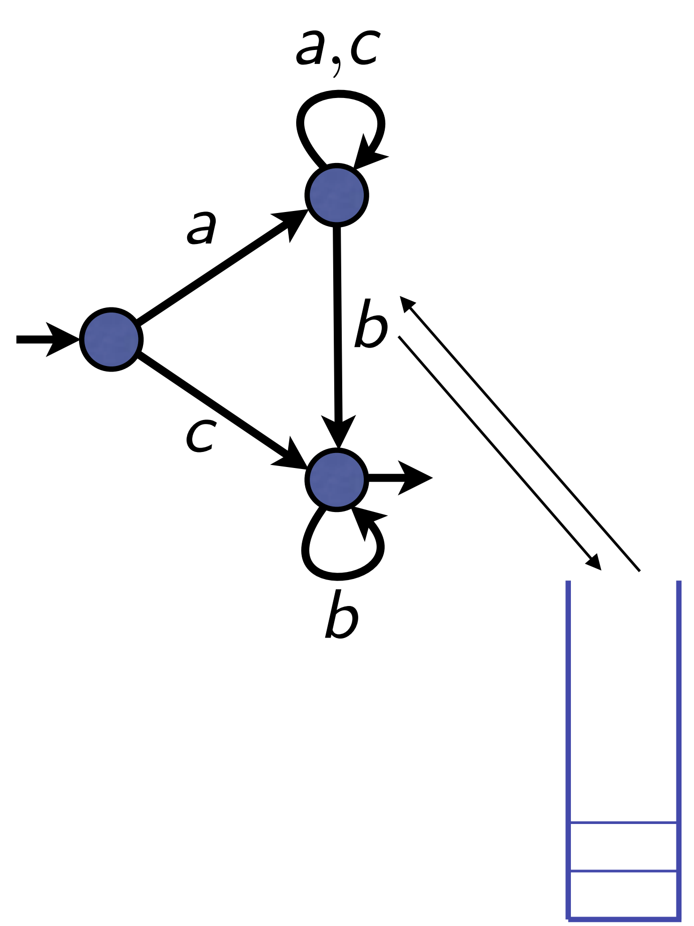
Variables range over finite domains

Functions can be recursive

Multi-threaded (shared state) or
Distributed Systems

Recursive program = Pushdown system

```
func f1
{while <true>
{call f1 OR
a OR
exit;}
return;}
```

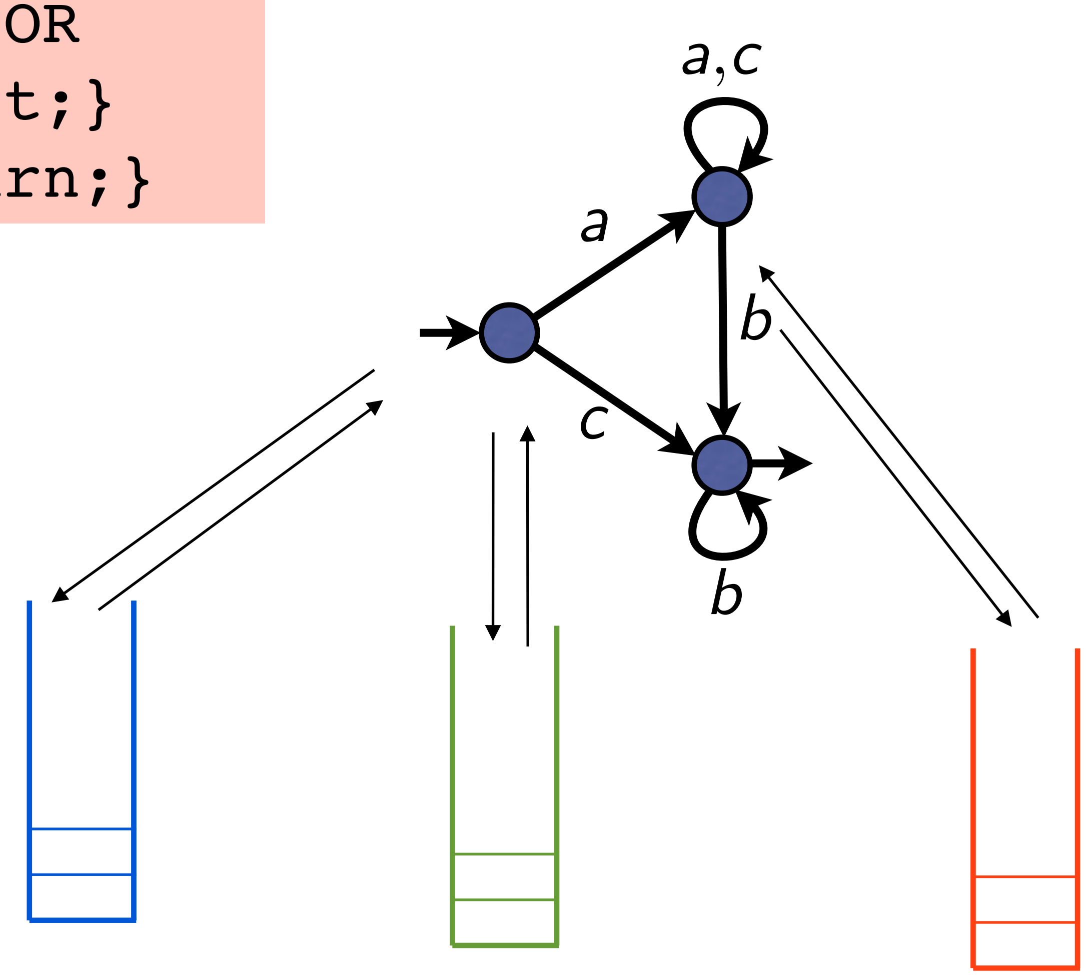


Multi-threaded program = Multi-PDS

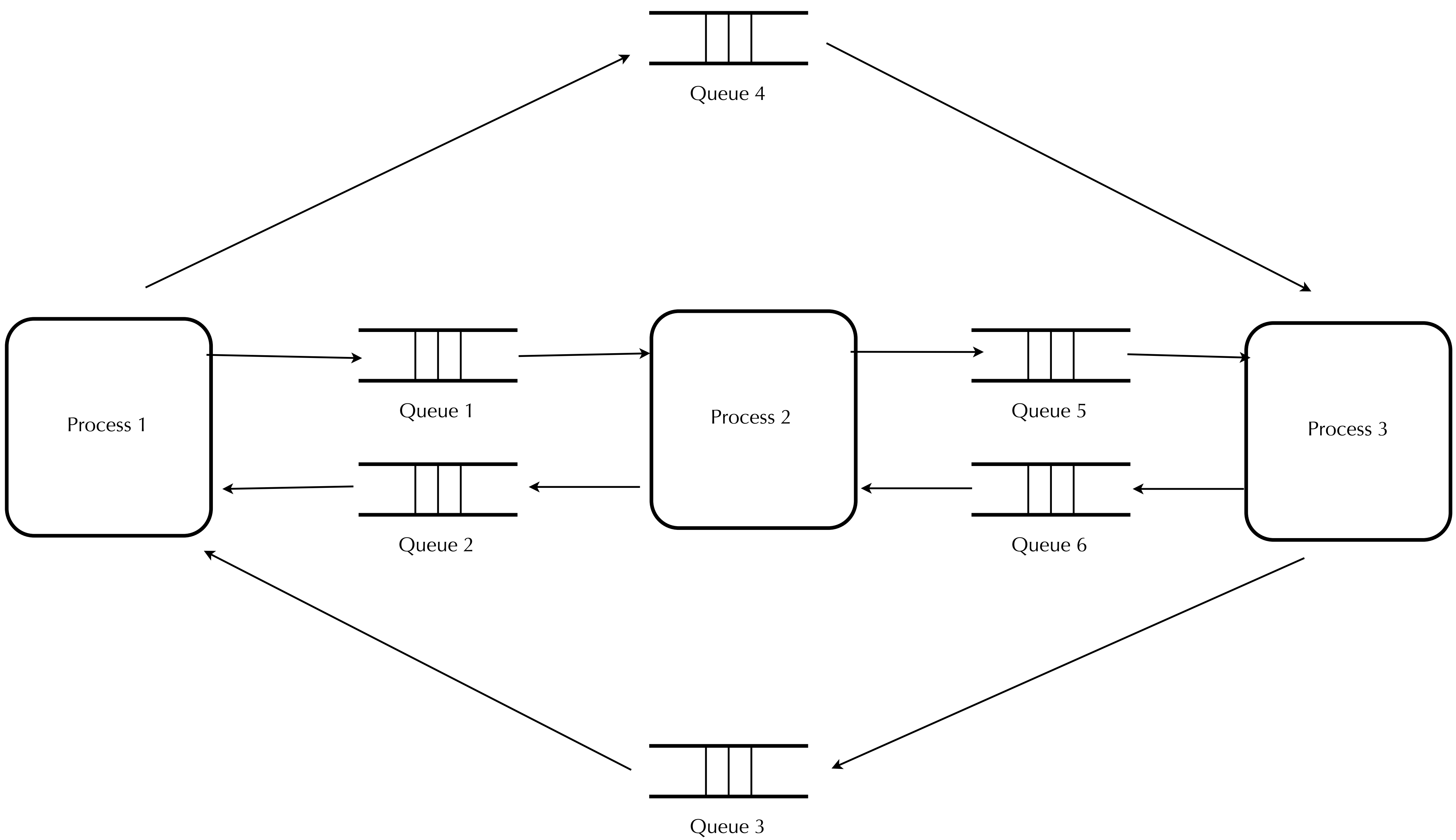
```
func f1
{while <true>
{call f1 OR
a OR
exit;}
return;}
```

```
func f2
{while <true>
{call f2 OR
a OR
exit;}
return;}
```

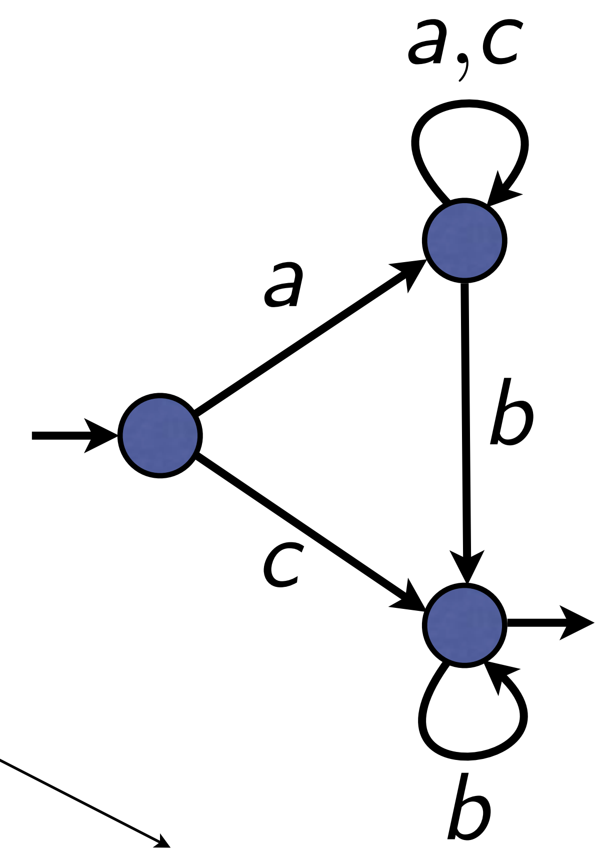
```
func f3
{while <true>
{call f3 OR
a OR
exit;}
return;}
```

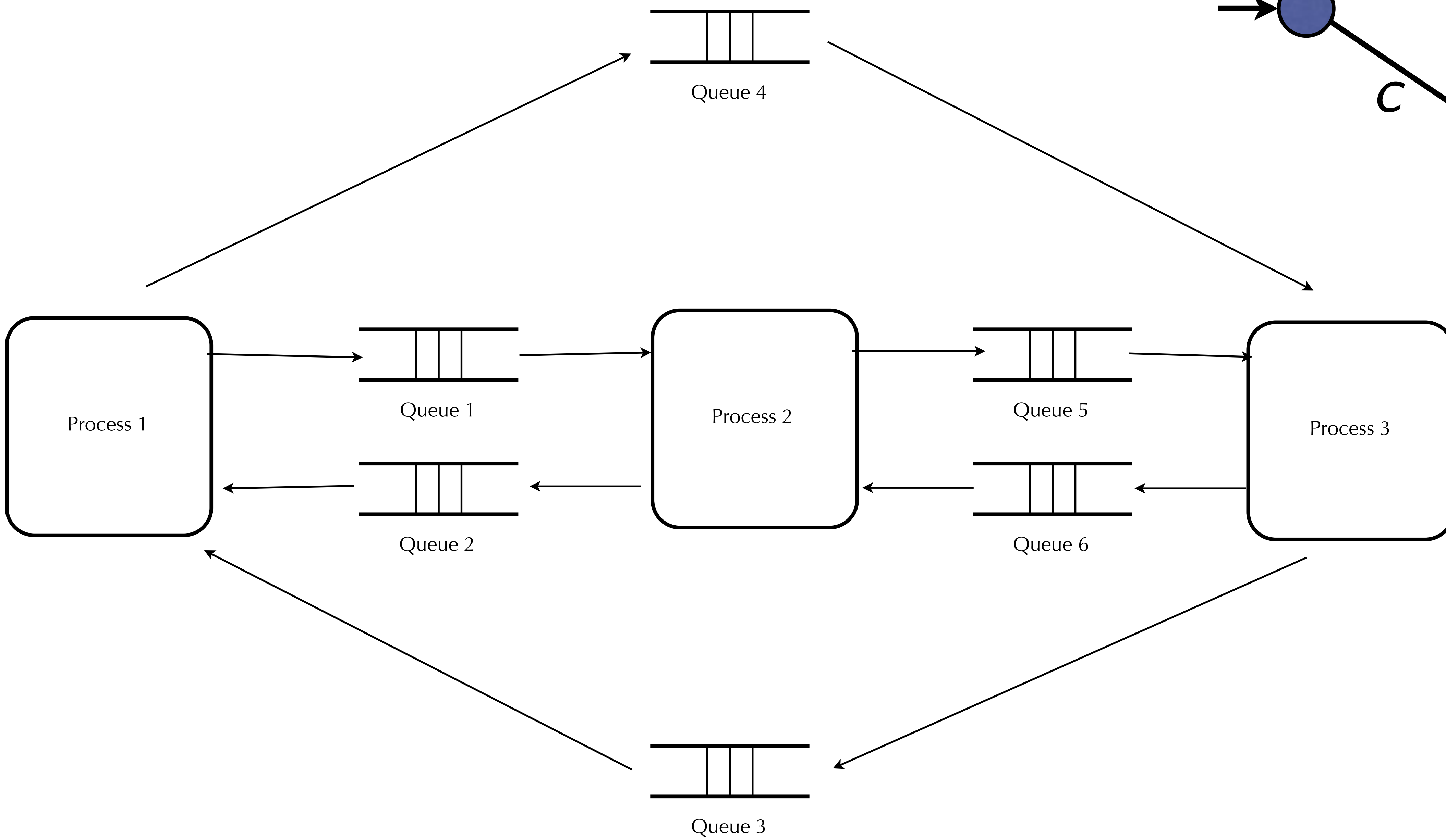


Communicating FSMs

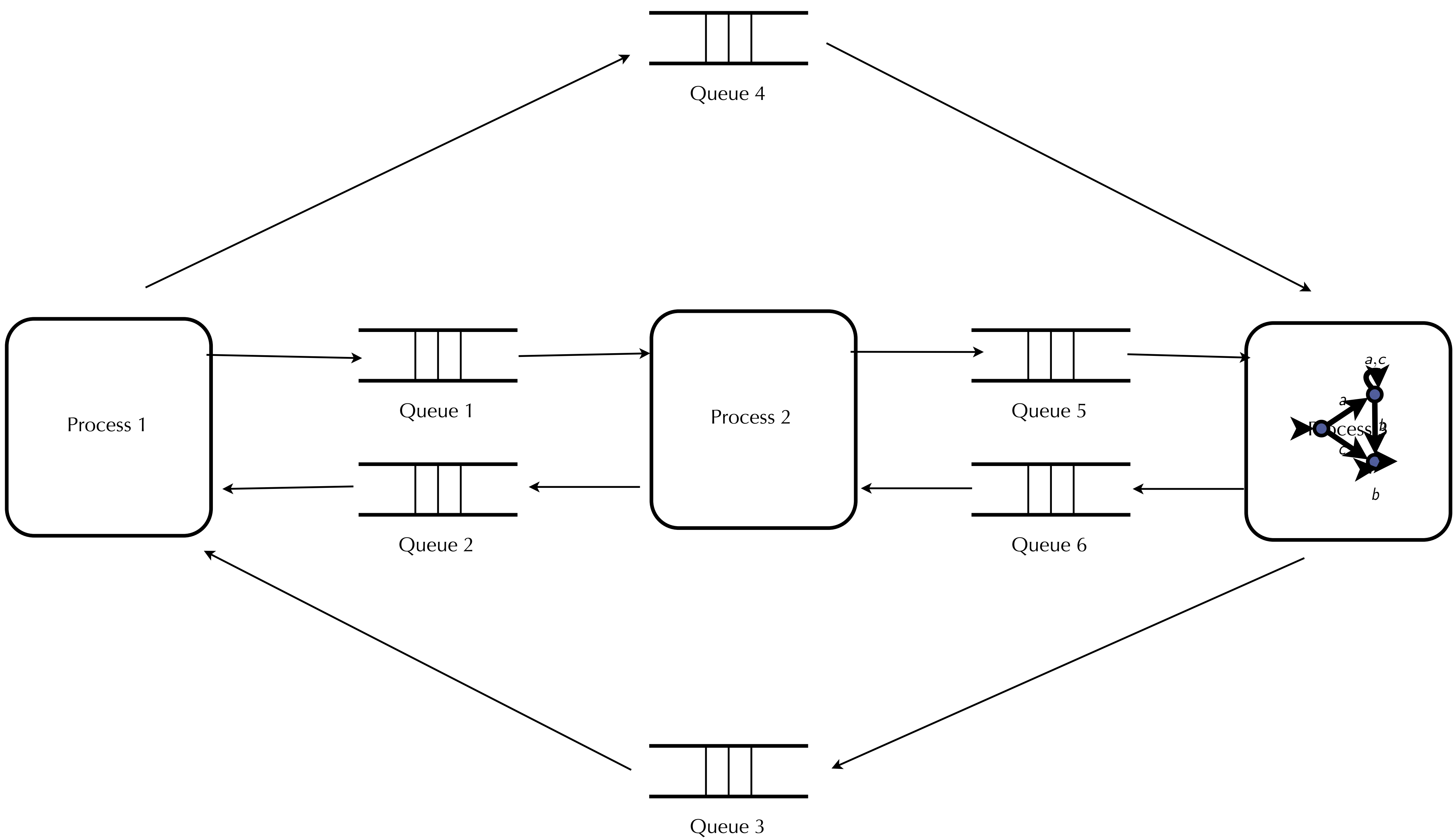


Communicating FSMs

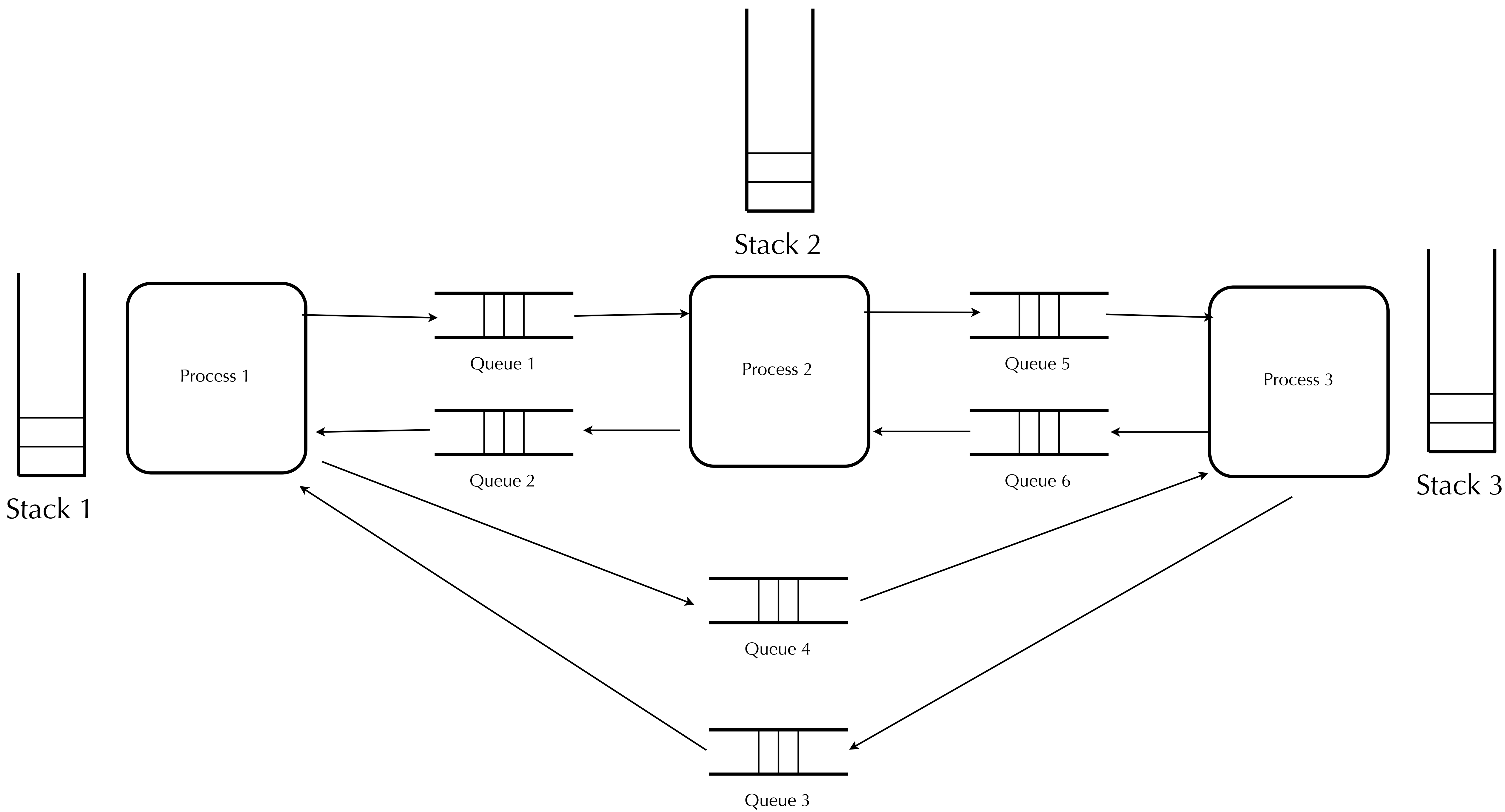




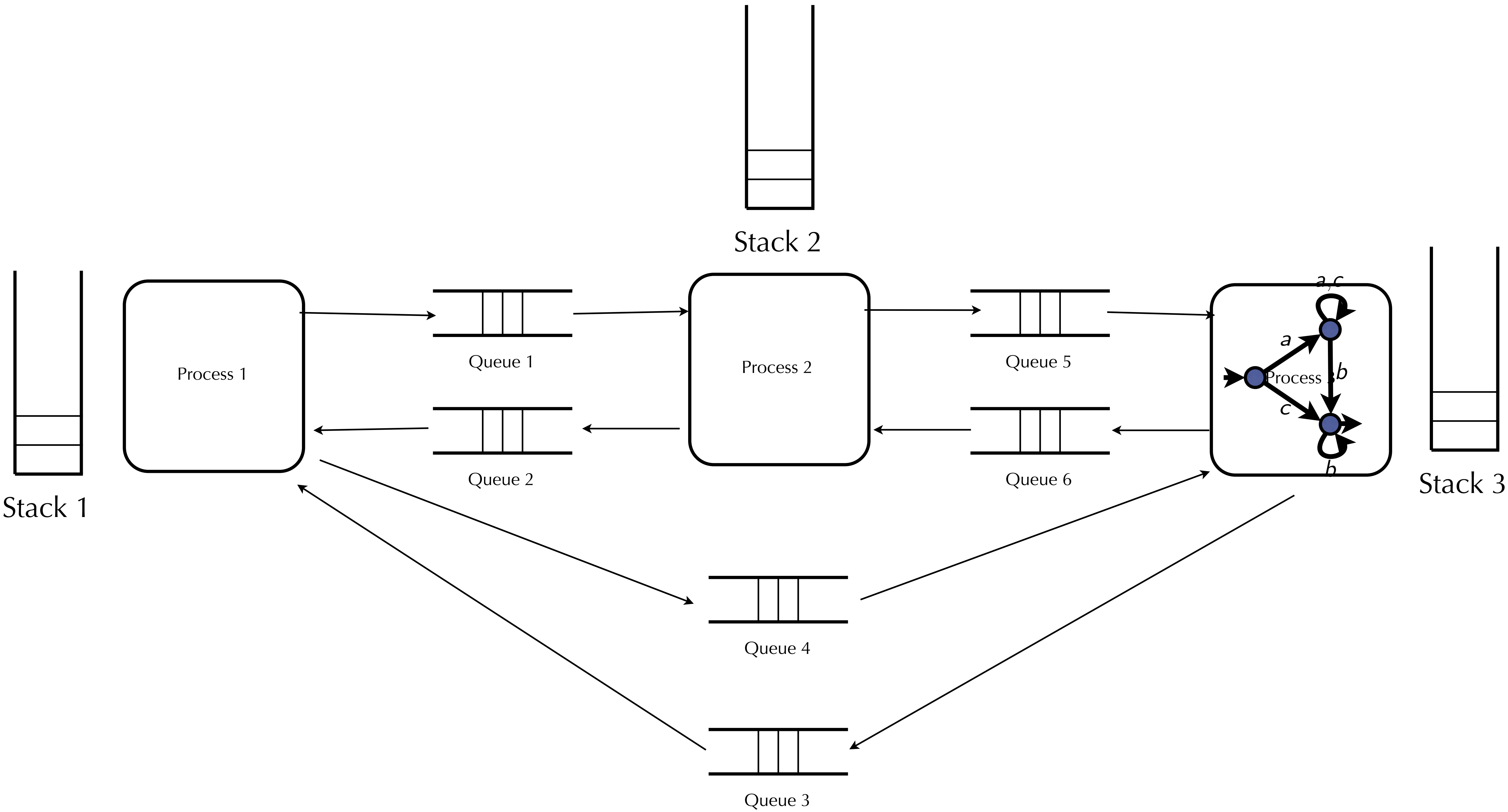
Communicating FSMs



Communicating Recursive Processes



Communicating Recursive Processes



The Verification Problem

MPDSs, CFSMs, CRPs are all Turing Powerful.

The Verification Problem

MPDSs, CFSMs, CRPs are all Turing Powerful.

- MPDSs -- Restrictions on the stack access
  - Bounded Context — Qadeer&Rehof,
  - Bounded Phase — LaTorre&Madhusudan&Parlato
  - Bounded Scope — LaTorre&Napoli
  - Ordered Stacks — Atig&Bollig&Habermehl

The Verification Problem

MPDSs, CFSMs, CRPs are all Turing Powerful.

The Verification Problem

MPDSs, CFSMs, CRPs are all Turing Powerful.

- CFMs
  - Universally/Existentially bounded systems

    Henrikson et al., Genest&Kuske&Muscholl
  - Message Sequence Graphs (or HMSCs)

    Madhusudan

The Verification Problem

MPDSs, CFSMs, CRPs are all Turing Powerful.

The Verification Problem

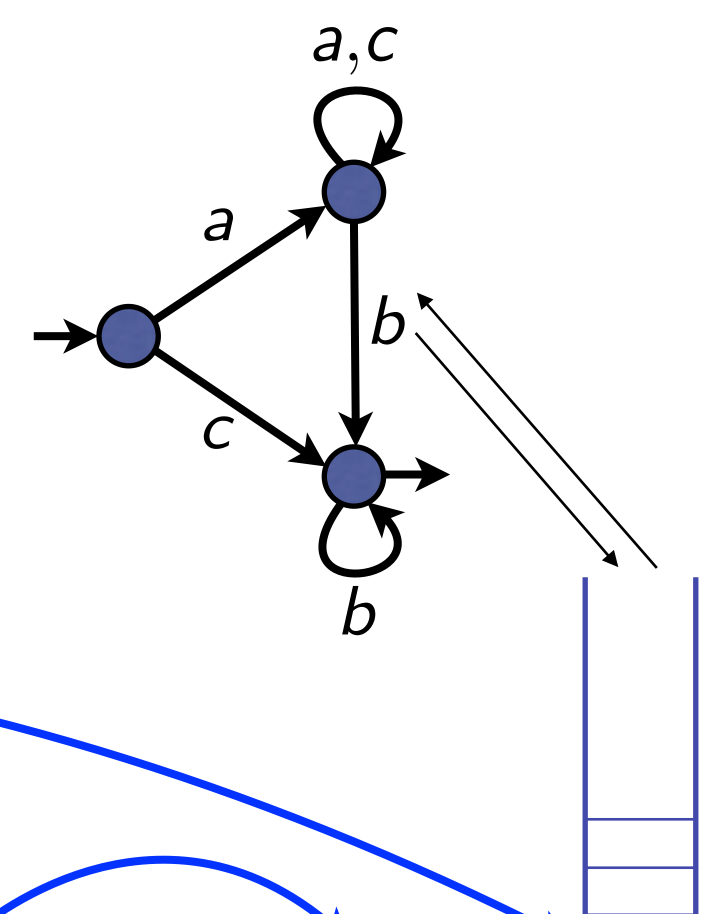
MPDSs, CFSMs, CRPs are all Turing Powerful.

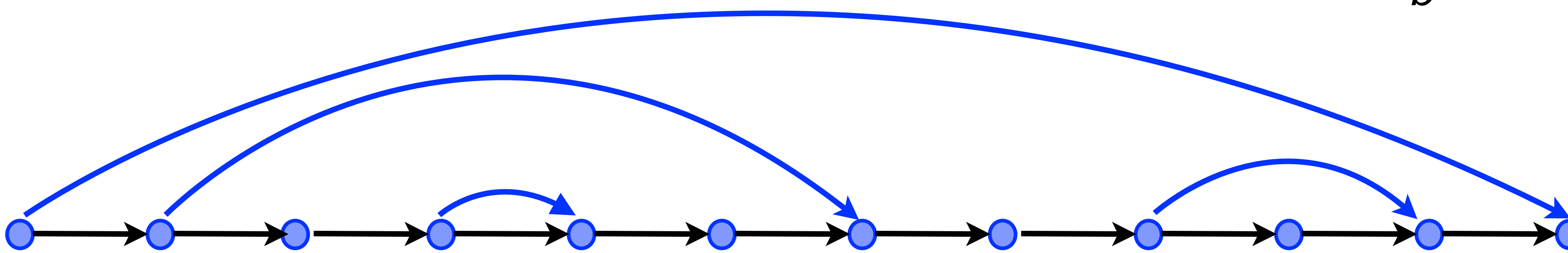
- CRPs
  - Well-queueing Systems with context bounds,...

Heussner&Lerox,&Muscholl&Sutre, LaTorre&Madhusudan&Parlato

Behaviours as Graphs

```
func f1
{while <true>
{call f1 OR
a OR
exit;}
return;}
```





Nested word

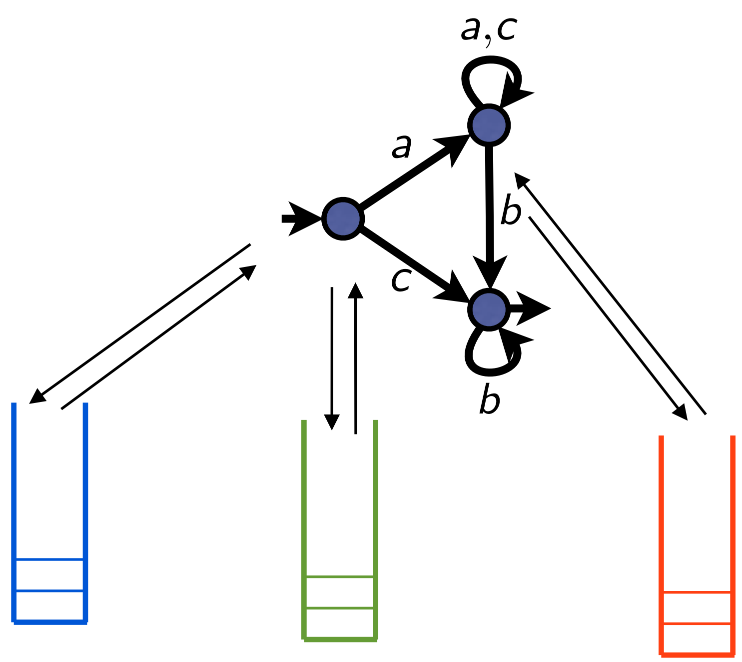
= word + binary nesting relation

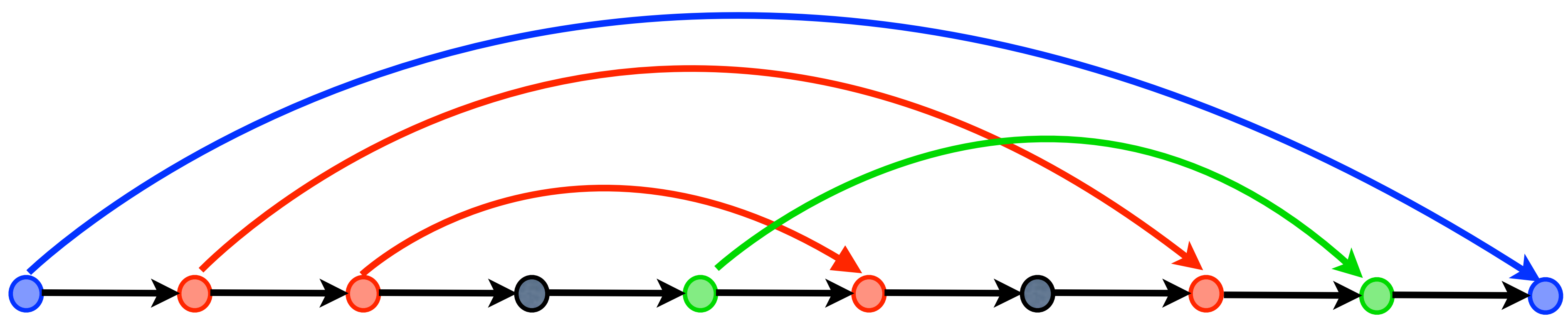
Behaviours as Graphs

```
func f1
{while <true>
{call f1 OR
a OR
exit;}
return;}
```

```
func f2
{while <true>
{call f2 OR
a OR
exit;}
return;}
```

```
func f3
{while <true>
{call f3 OR
a OR
exit;}
return;}
```

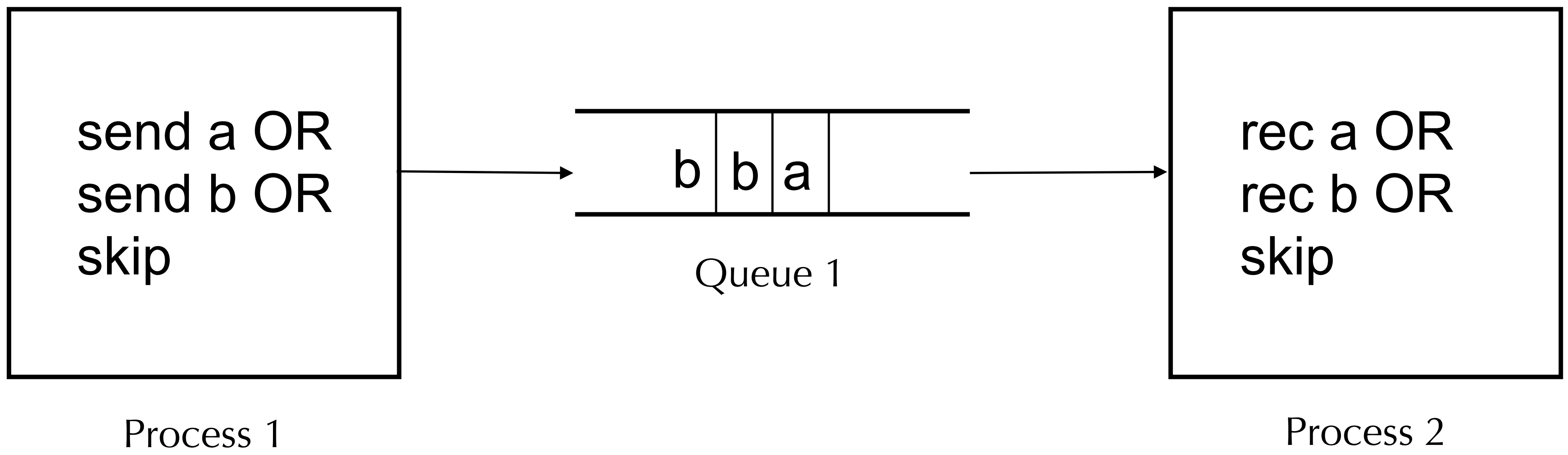


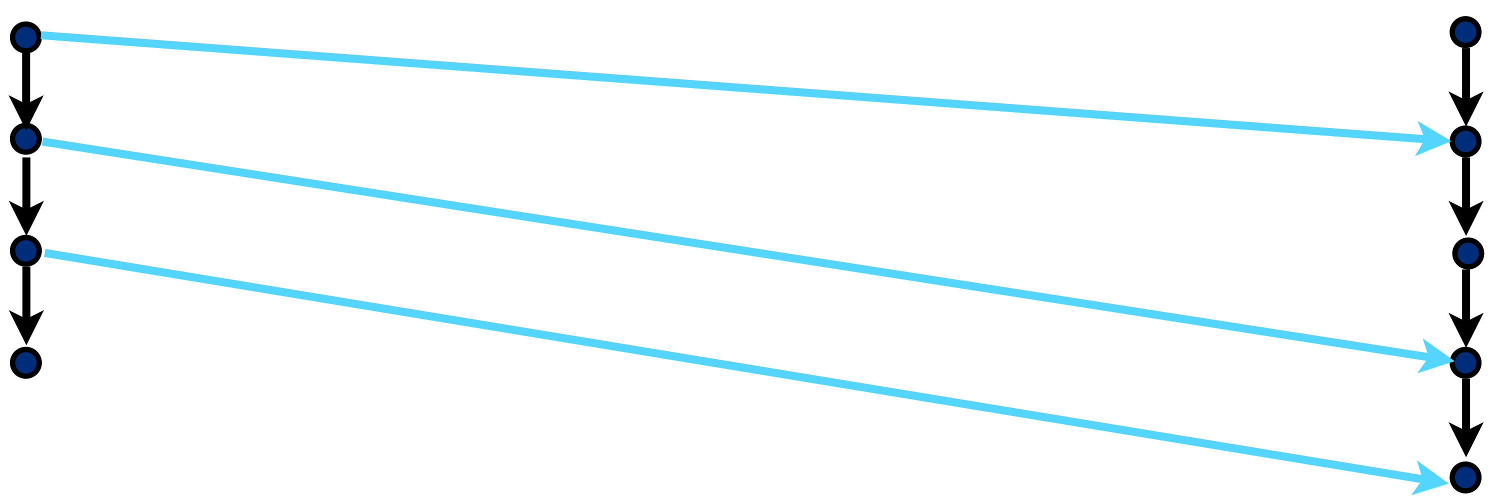


Multiply Nested word (MNW)

= word + multiple nesting relations

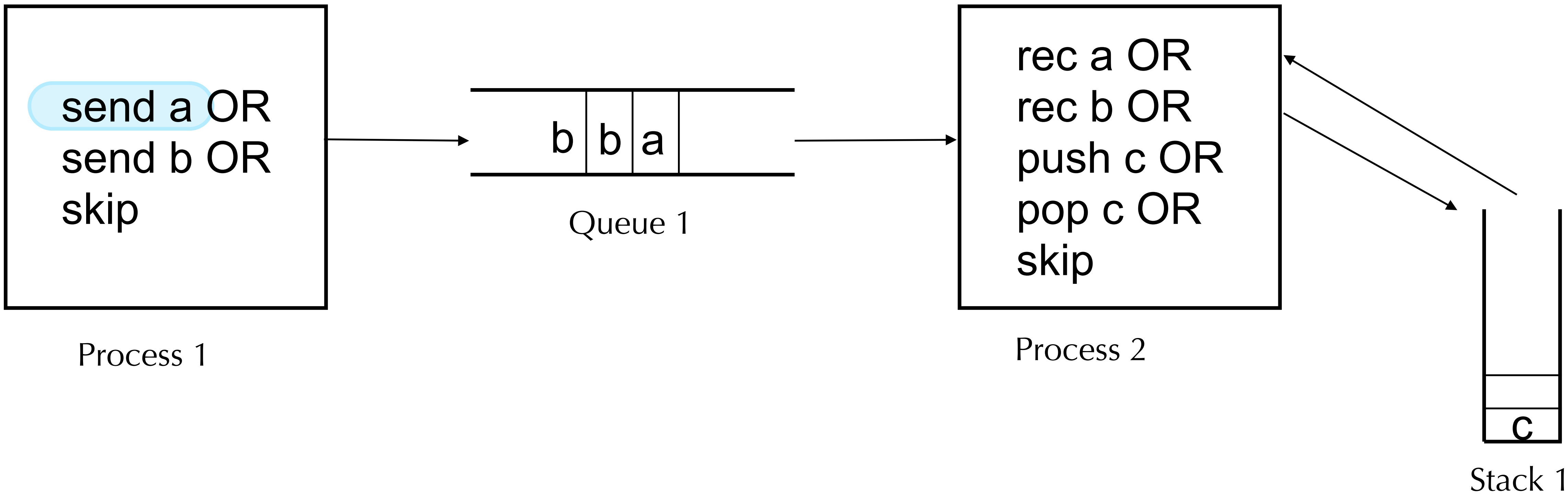
Behaviours as Graphs

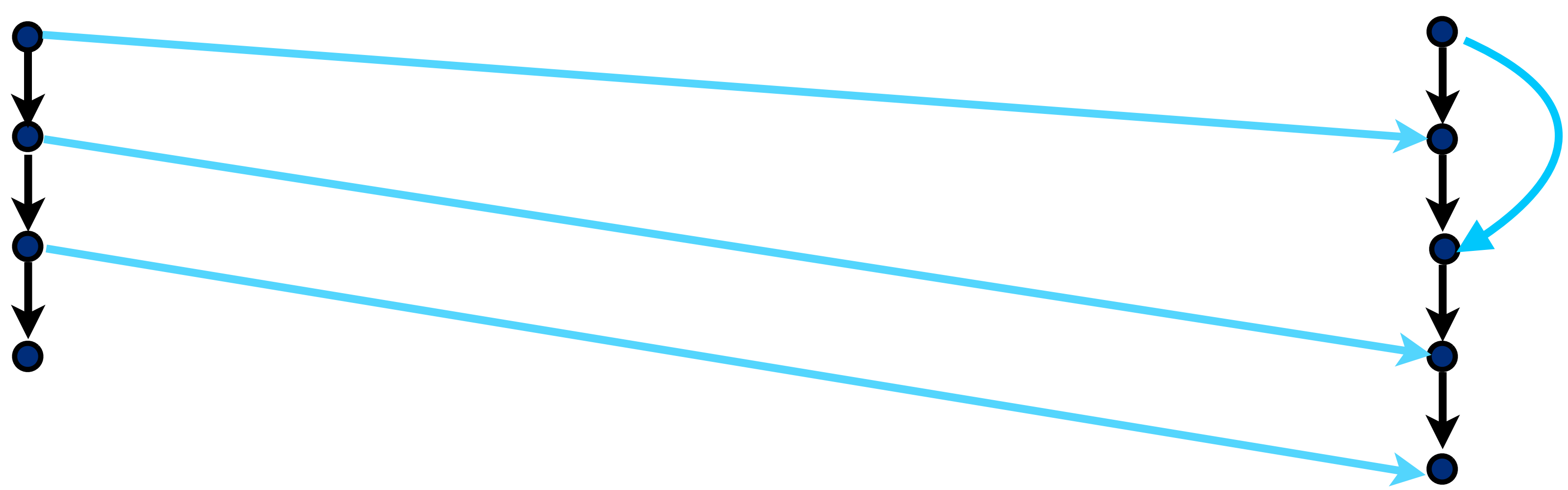




Message Sequence Charts

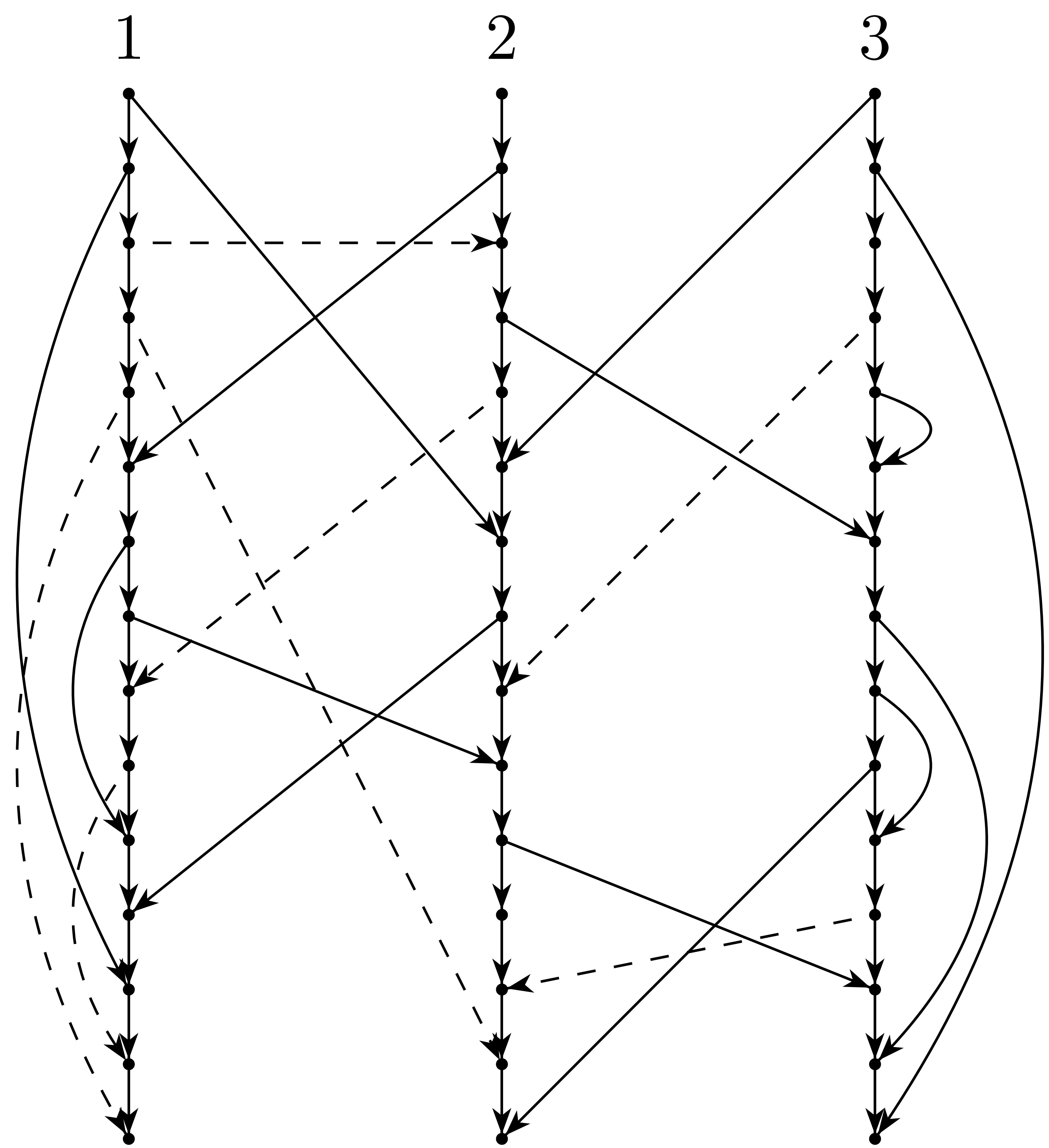
Behaviours as Graphs





Message Sequence Charts with Nesting

And other beasts ...



Graphs and MSO

Our graphs are

Graphs and MSO

Our graphs are

- A finite number of linear orders $(<_p)$
- One or more nesting relation per linear order
  Corresponding to the stacks $(<_s)$
- Message relations betweens processes
  One per queue, assumed to be FIFO $(<_{pq})$

Graphs and MSO

Our graphs are

- A finite number of linear orders ($<_p$)
- One or more nesting relation per linear order
  Corresponding to the stacks ($<_s$)
- Message relations betweens processes
  One per queue, assumed to be FIFO ($<_{pq}$)

Matching Relations

Graphs and MSO

Our graphs are

- A finite number of linear orders ($<_p$)
- One or more nesting relation per linear order
  Corresponding to the stacks ($<_s$)
- Message relations betweens processes
  One per queue, assumed to be FIFO ($<_{pq}$)

Graphs and MSO

Our graphs are

- A finite number of linear orders ($<_p$)
- One or more nesting relation per linear order
  Corresponding to the stacks ($<_s$)
- Message relations betweens processes
  One per queue, assumed to be FIFO ($<_{pq}$)

MSO has one binary relation symbol for each of these relations.

Graphs and MSO

Our graphs are

- A finite number of linear orders ($<_p$)
- One or more nesting relation per linear order
  Corresponding to the stacks ($<_s$)
- Message relations betweens processes
  One per queue, assumed to be FIFO ($<_{pq}$)

MSO has one binary relation symbol for each of these relations.

Satisfiability is undecidable with 2 nesting relations / 2 processes connected by queues

Tree-width

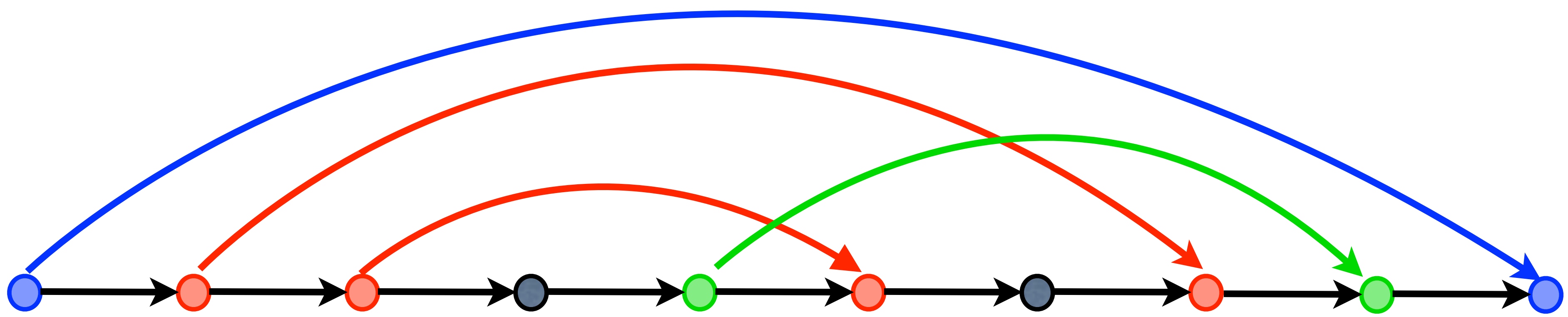
Madhusudan/Parlato show that

Tree-width

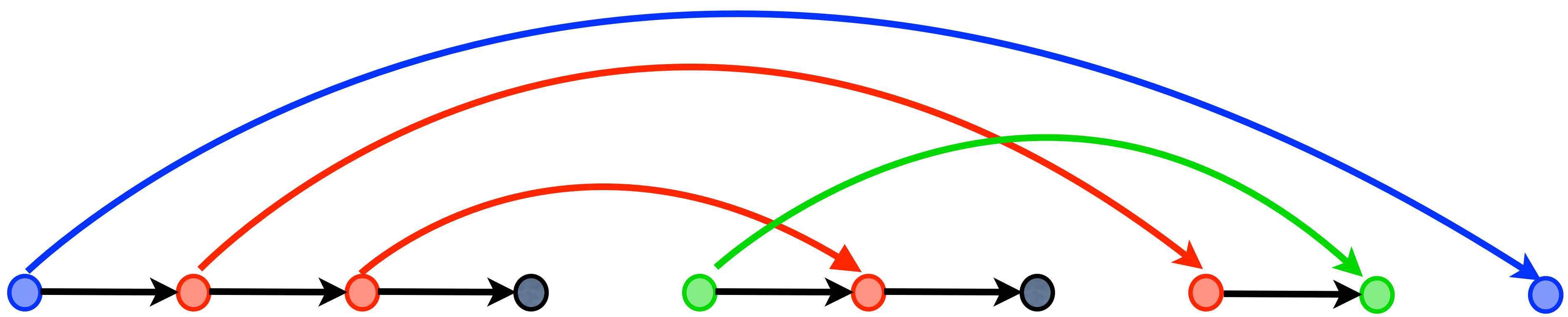
Madhusudan/Parlato show that

- Runs of the restricted systems have bounded tree-width
- For any system, its set of restricted runs is MSO definable.

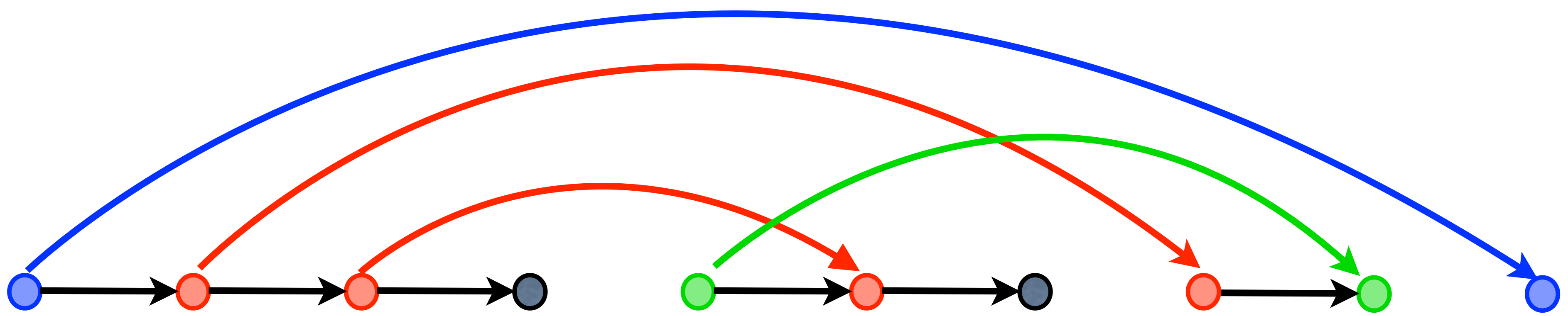
behaviors



Split behaviors

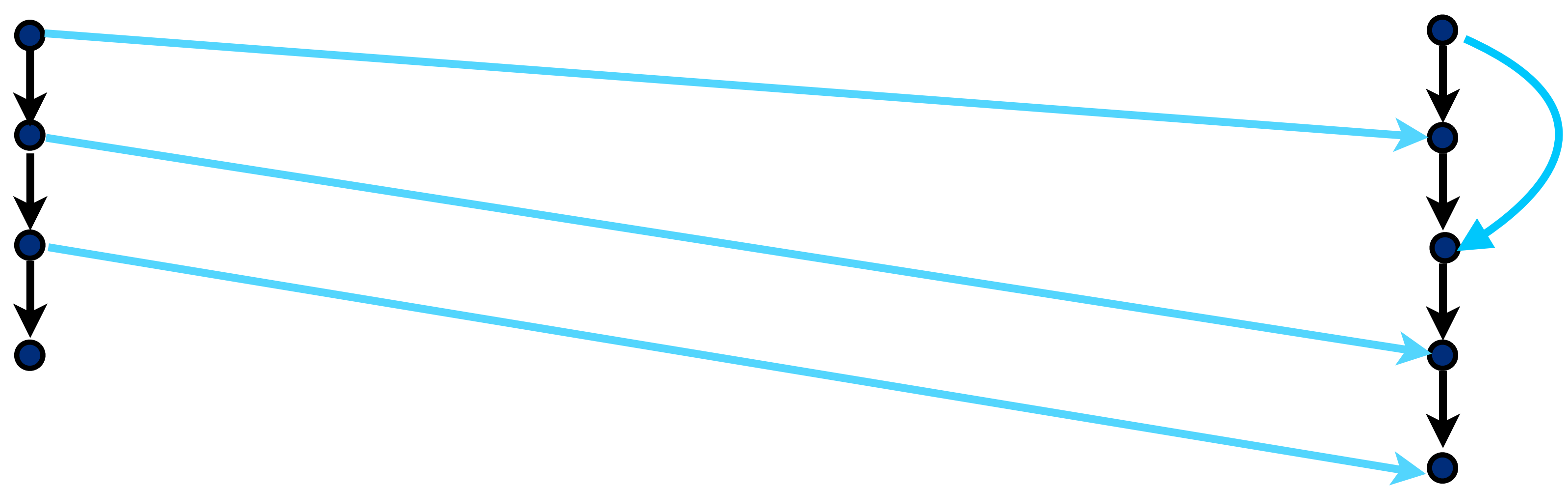


Split behaviors

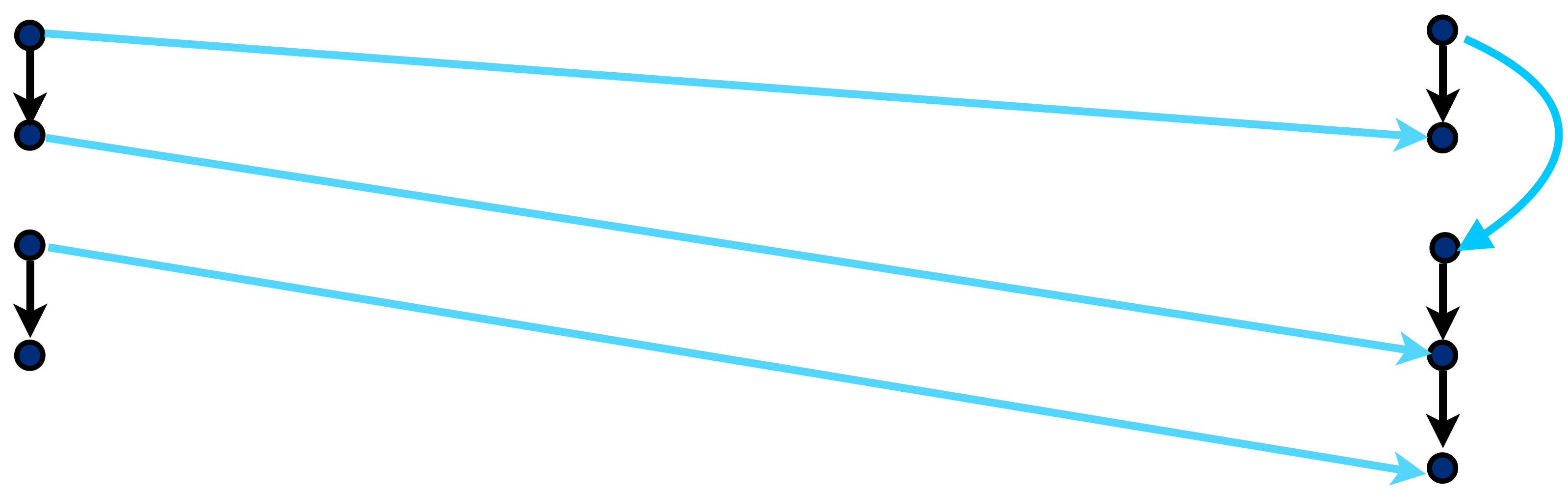


Size of the split = number of components = 4

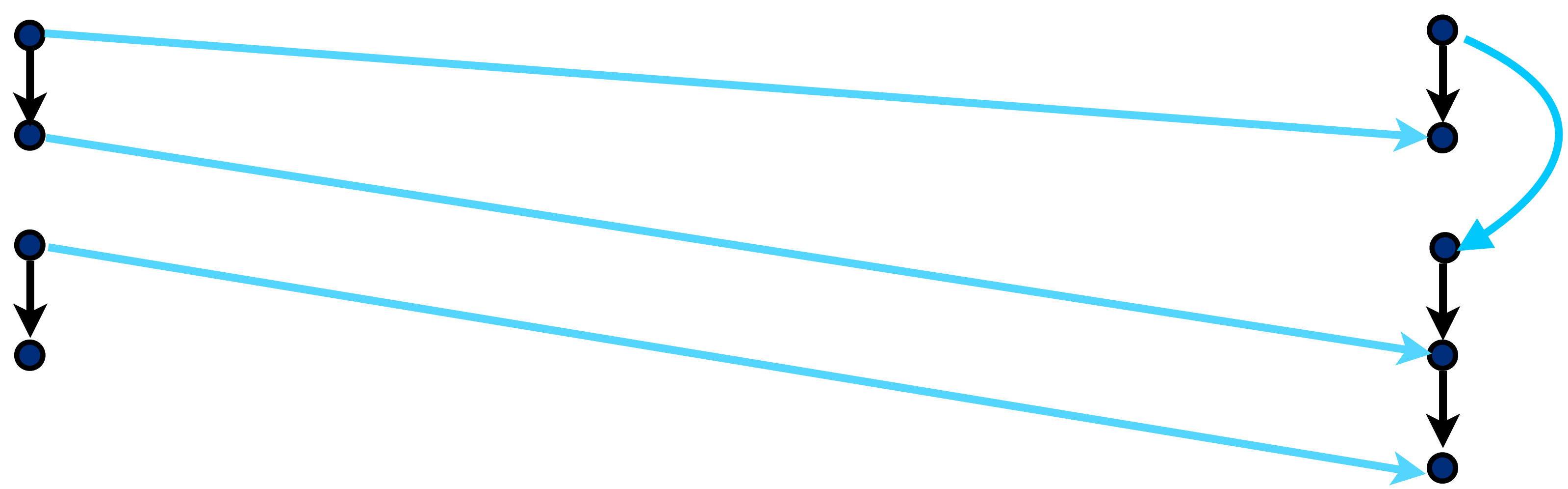
behaviors



Split behaviors



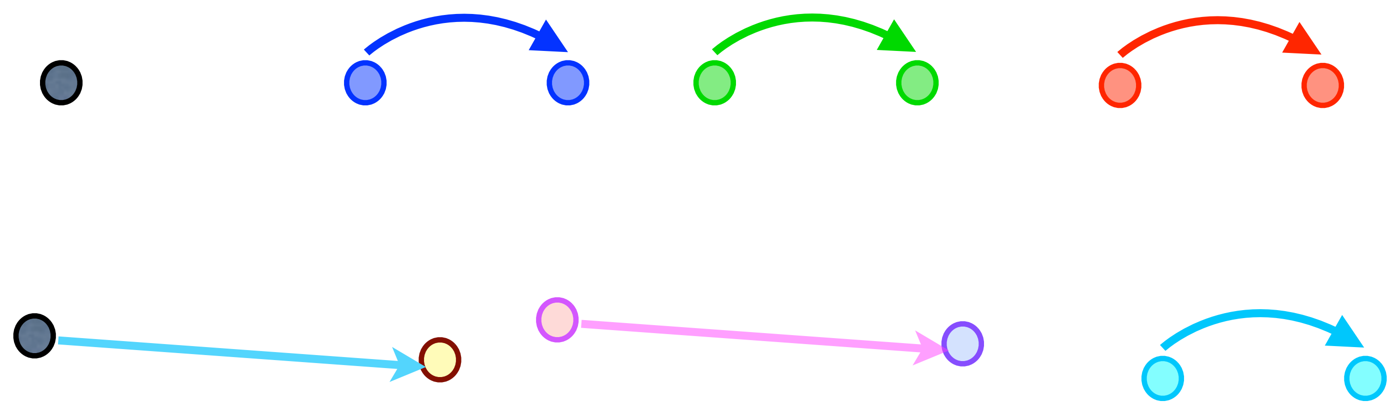
Split behaviors



Size of the split = number of components = 4

An algebra on Split behaviours

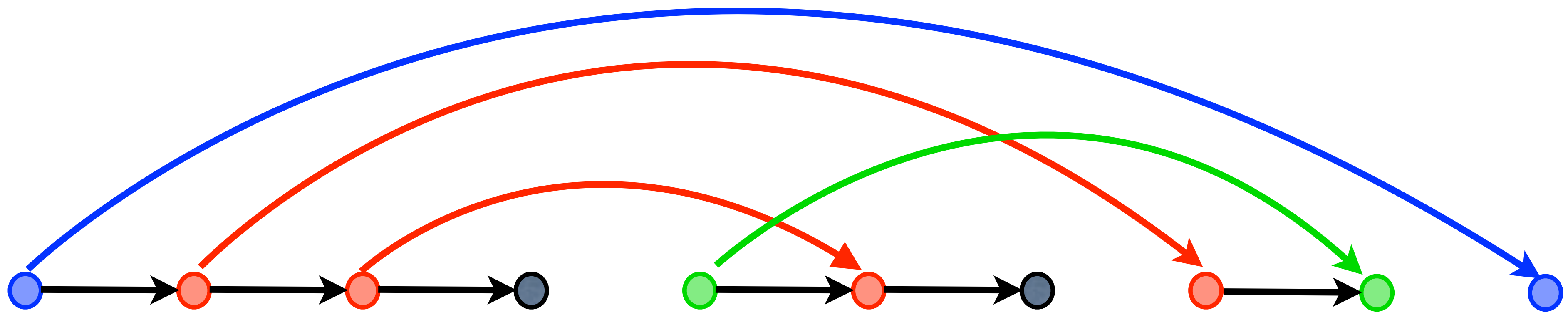
Basic splits:



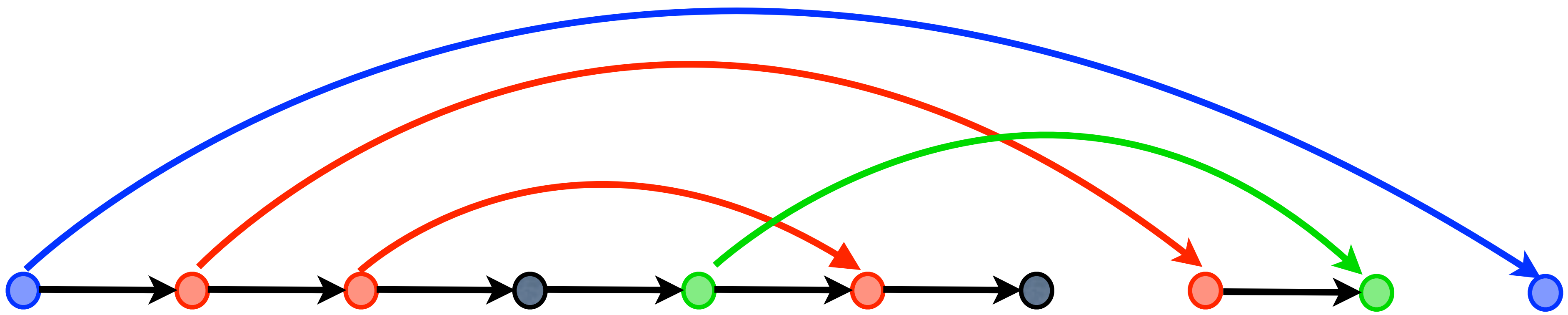
Operations:

merge (binary)
shuffle (unary)

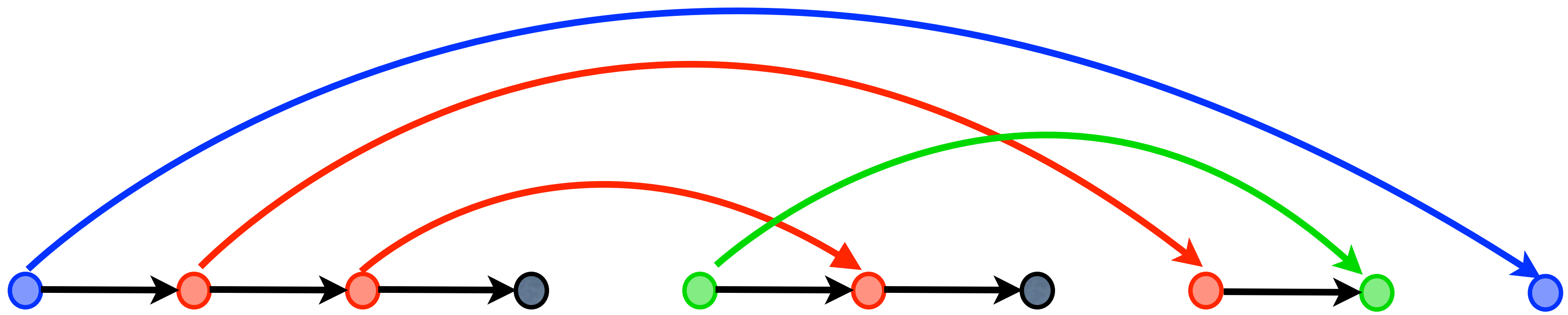
The merge Operation



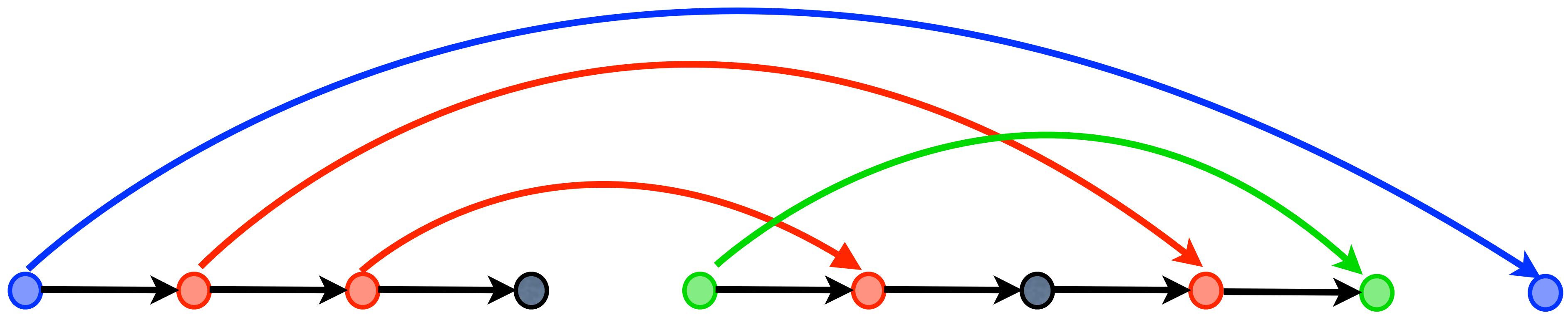
The merge Operation



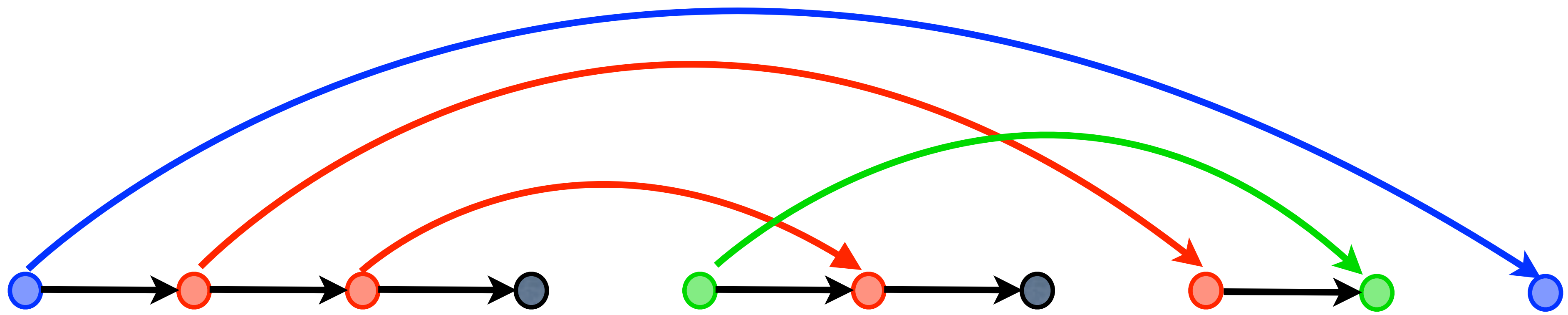
The merge Operation



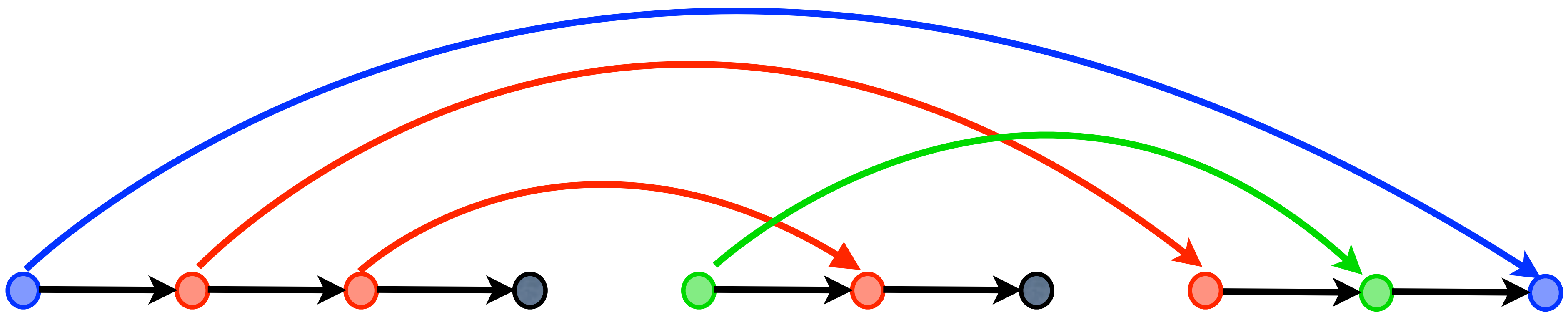
The merge Operation



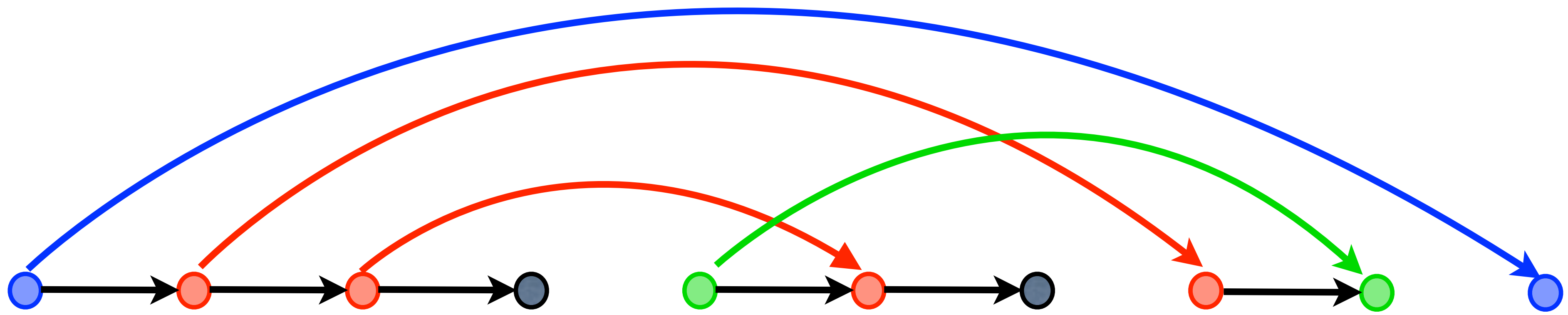
The merge Operation



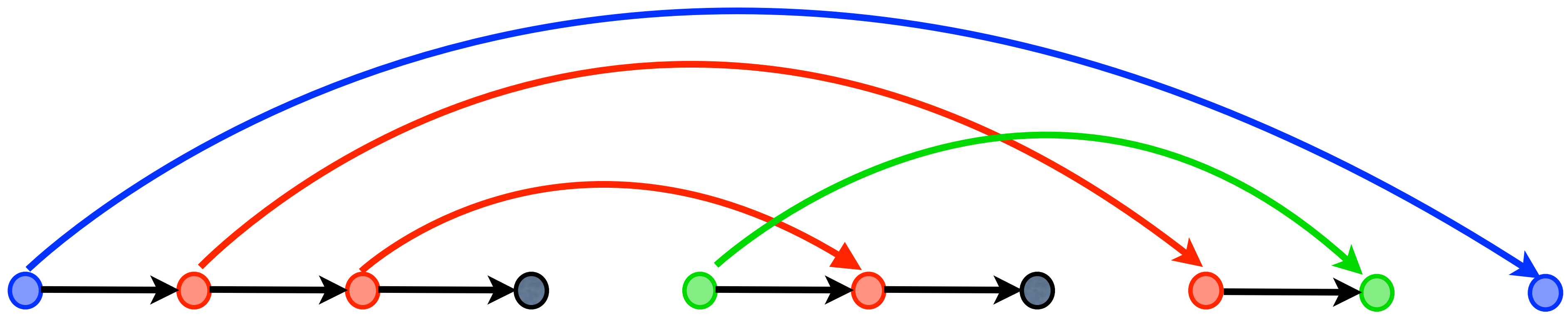
The merge Operation

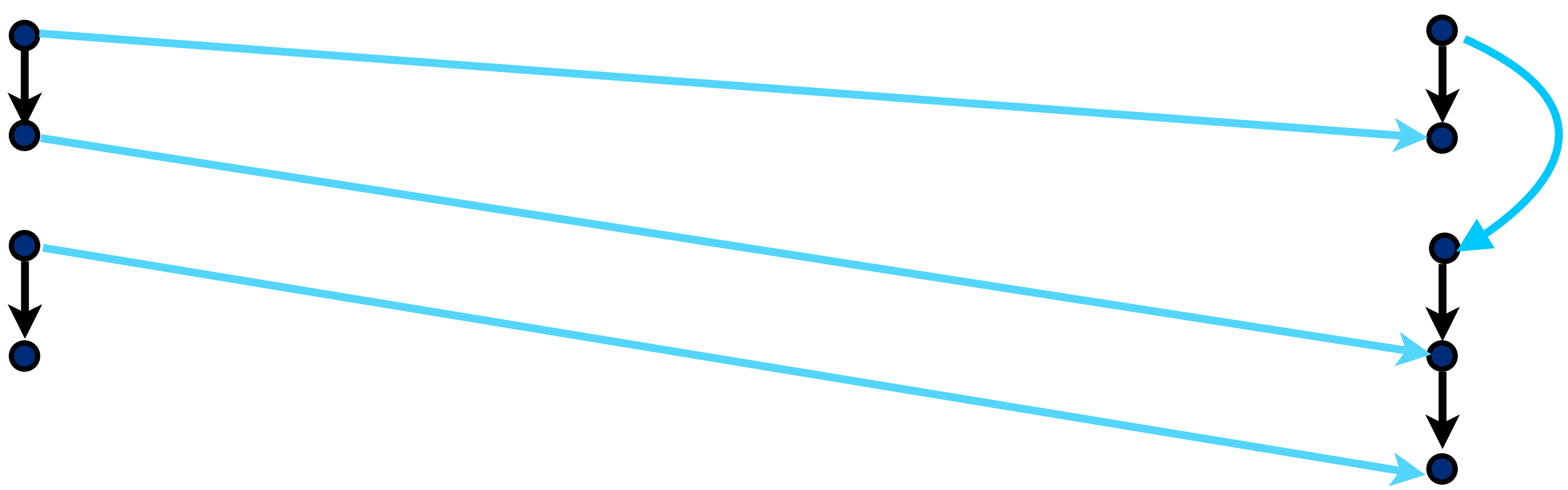


The merge Operation

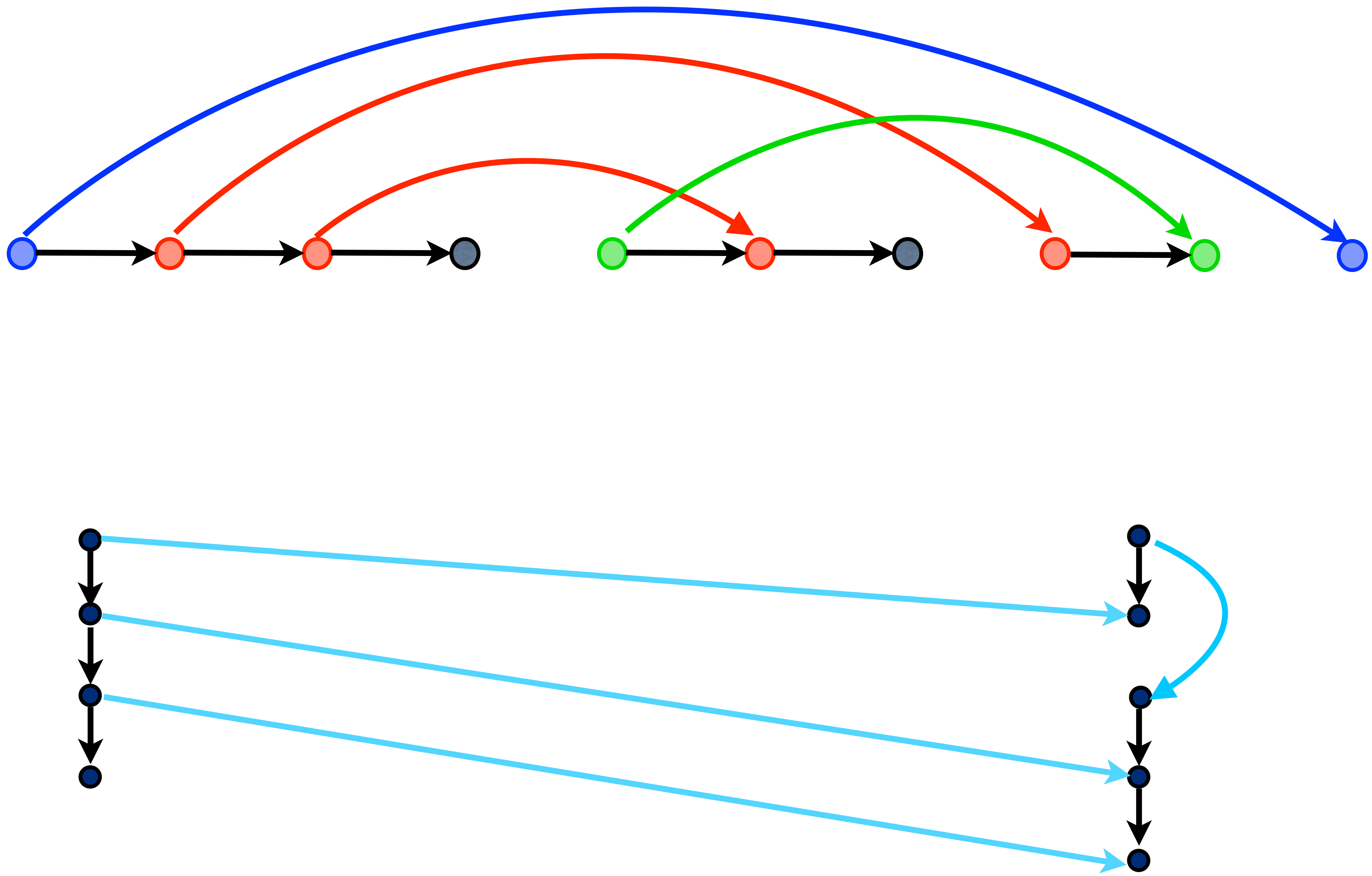


The merge Operation

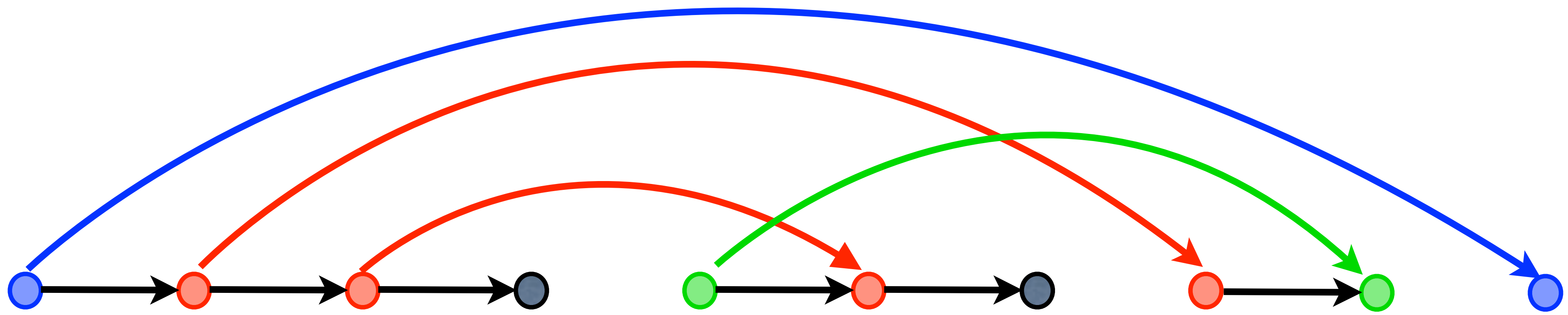


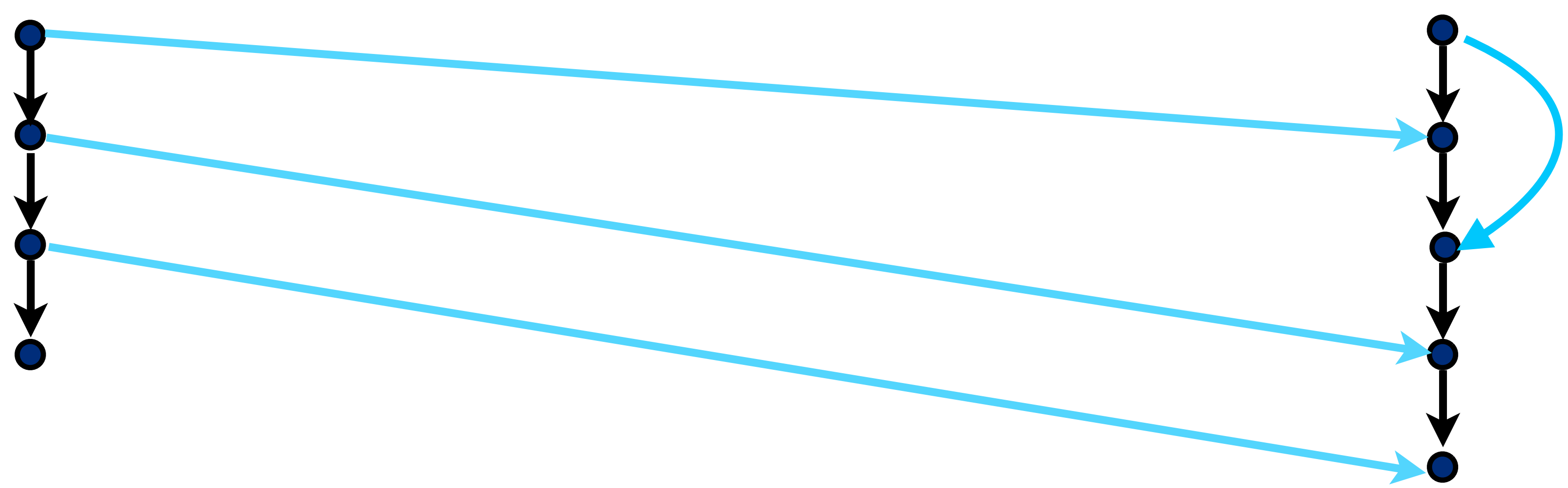


The merge Operation

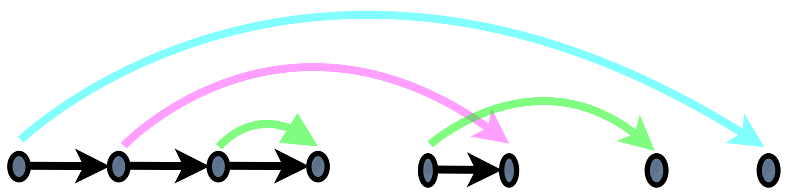


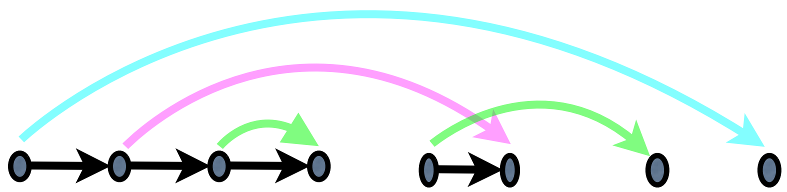
The merge Operation



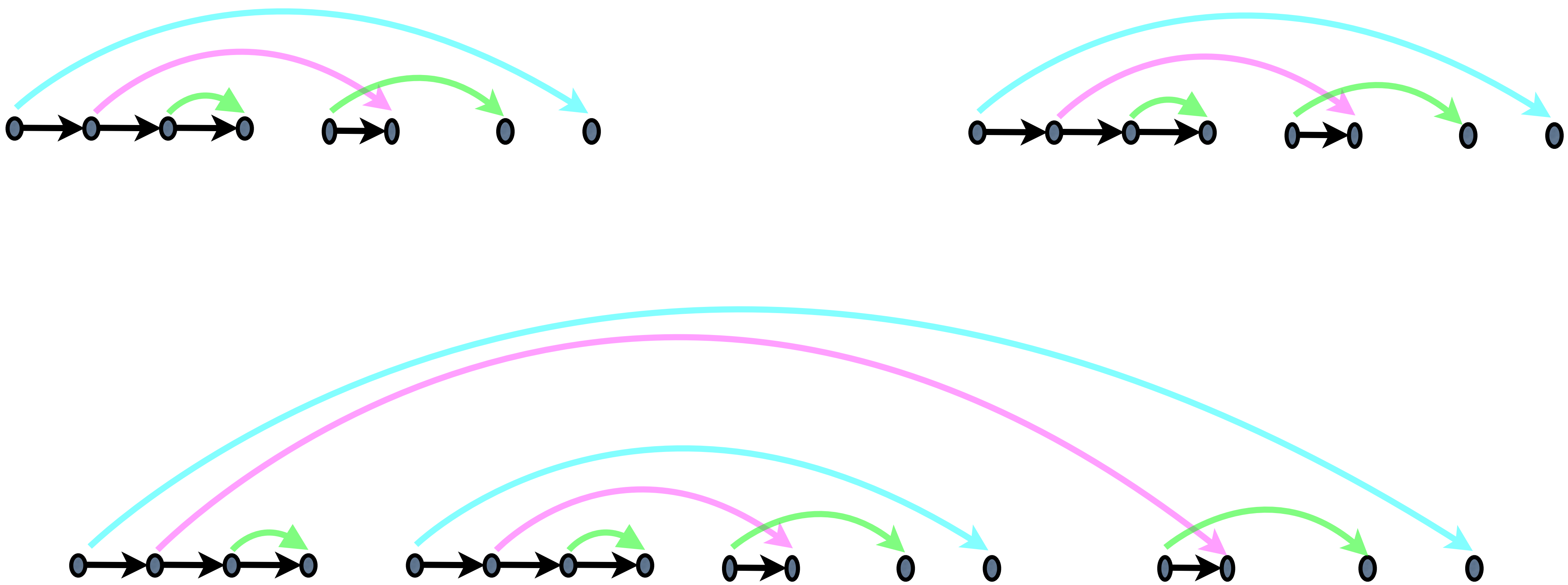


The Shuffle Operation

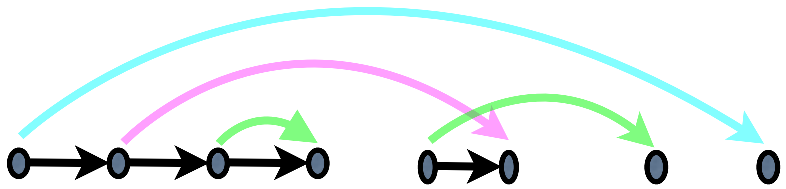


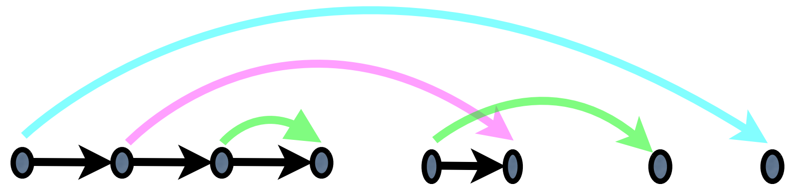


The Shuffle Operation

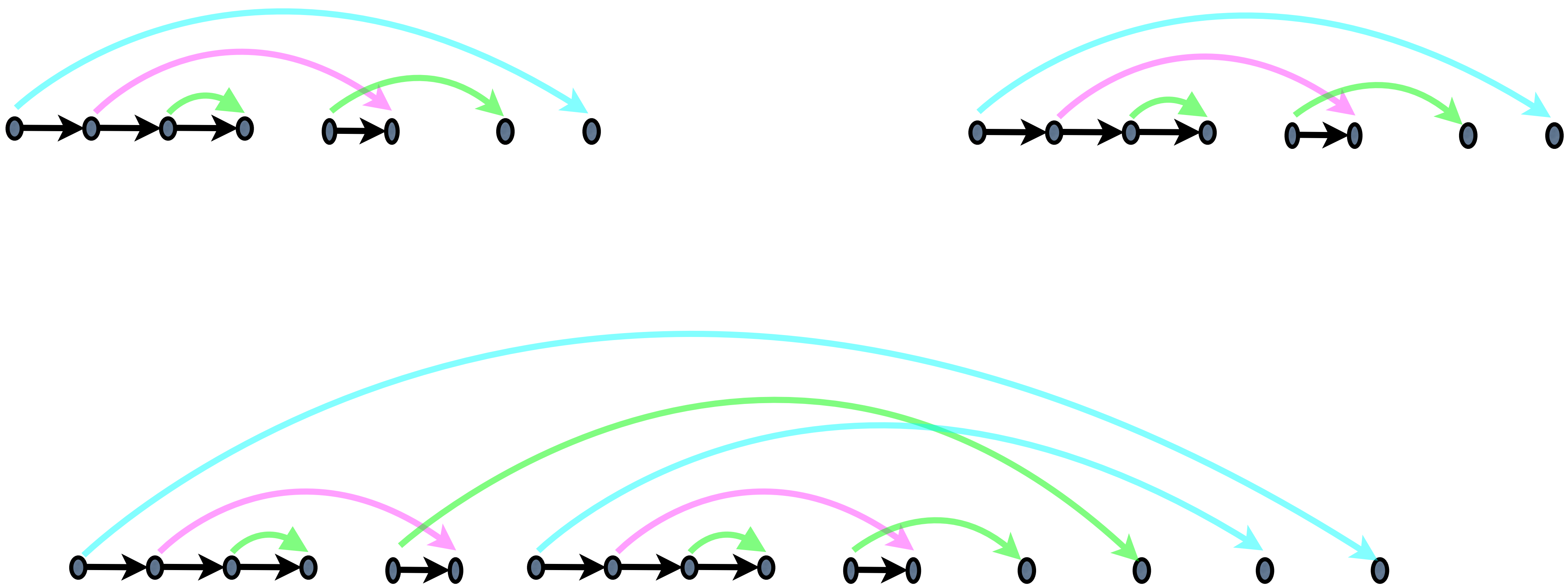


The Shuffle Operation

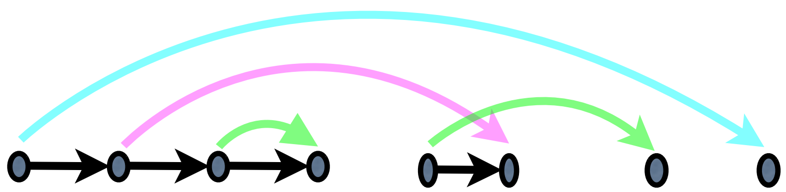


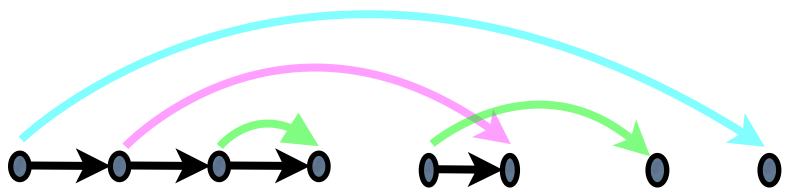


The Shuffle Operation



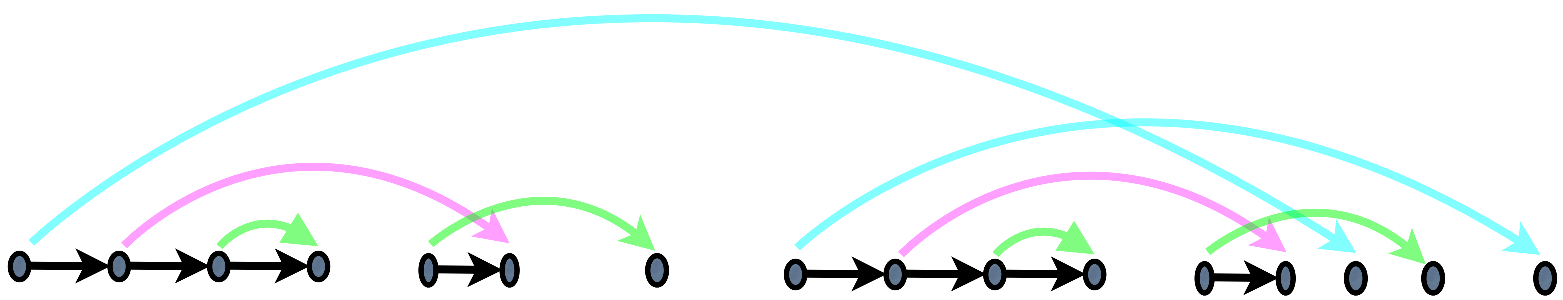
The Shuffle Operation



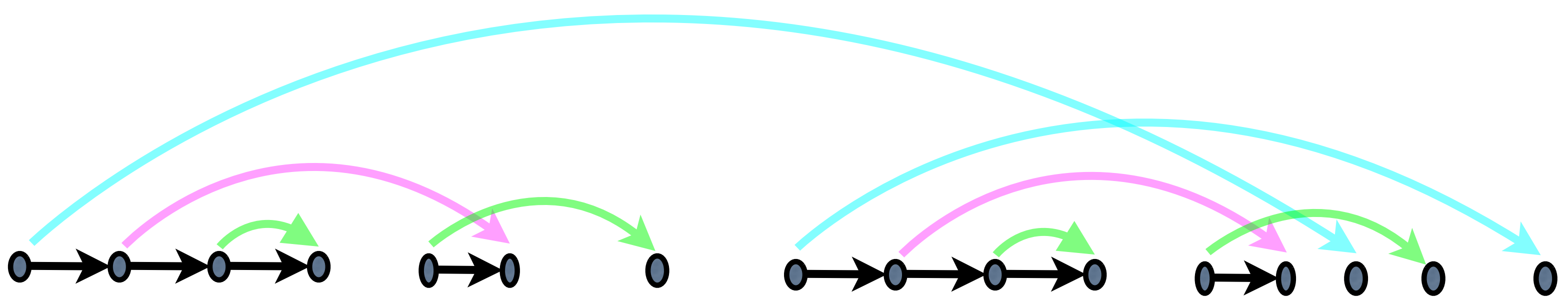


Shuffle Operation

Shuffle Operation

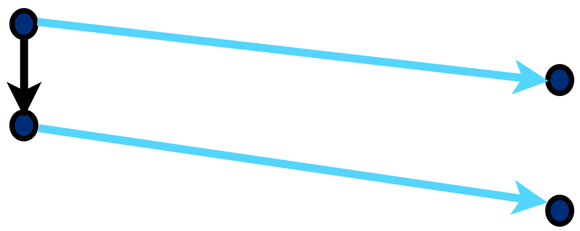


Shuffle Operation

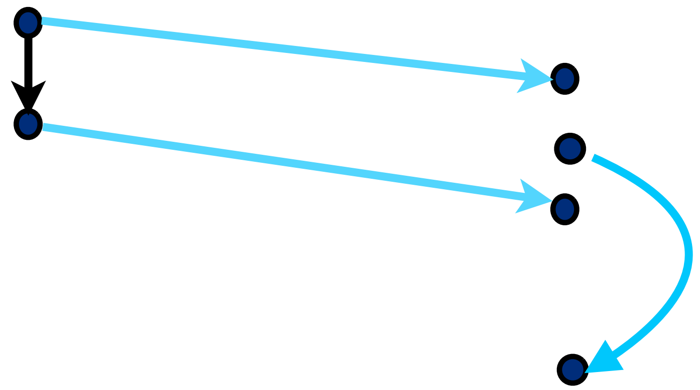


Invalid!

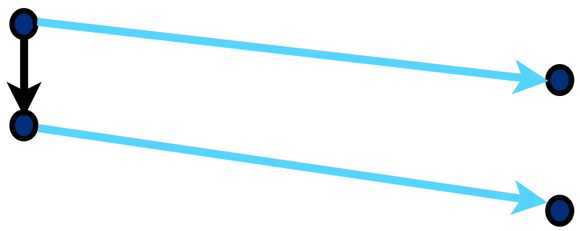
Shuffle Operation



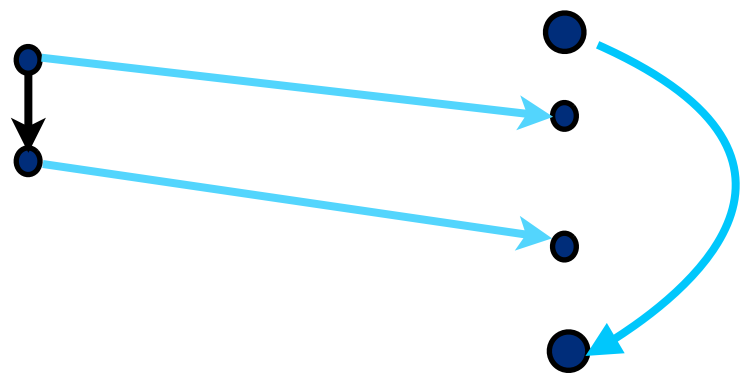
Shuffle Operation



Shuffle Operation

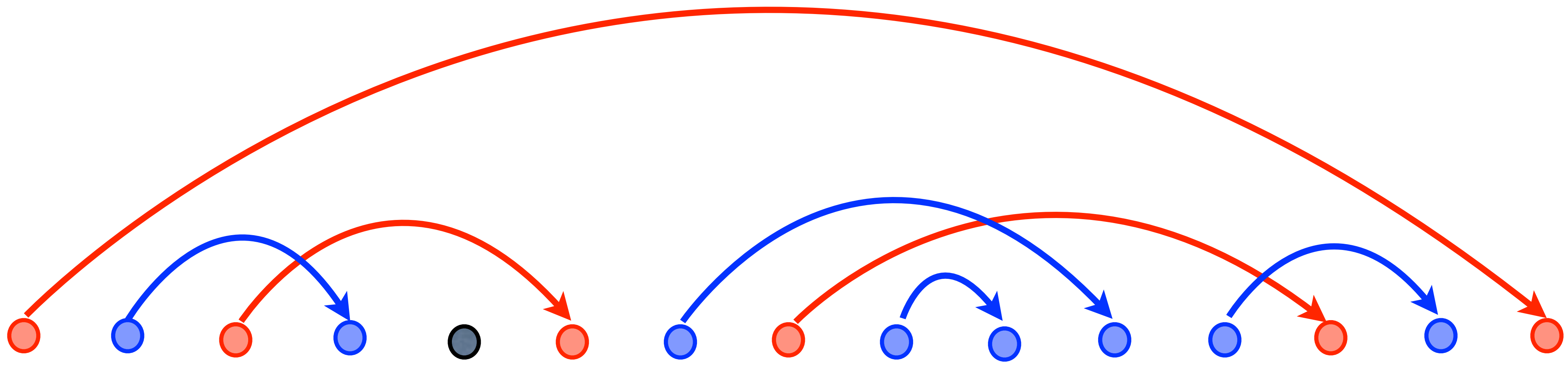


Shuffle Operation

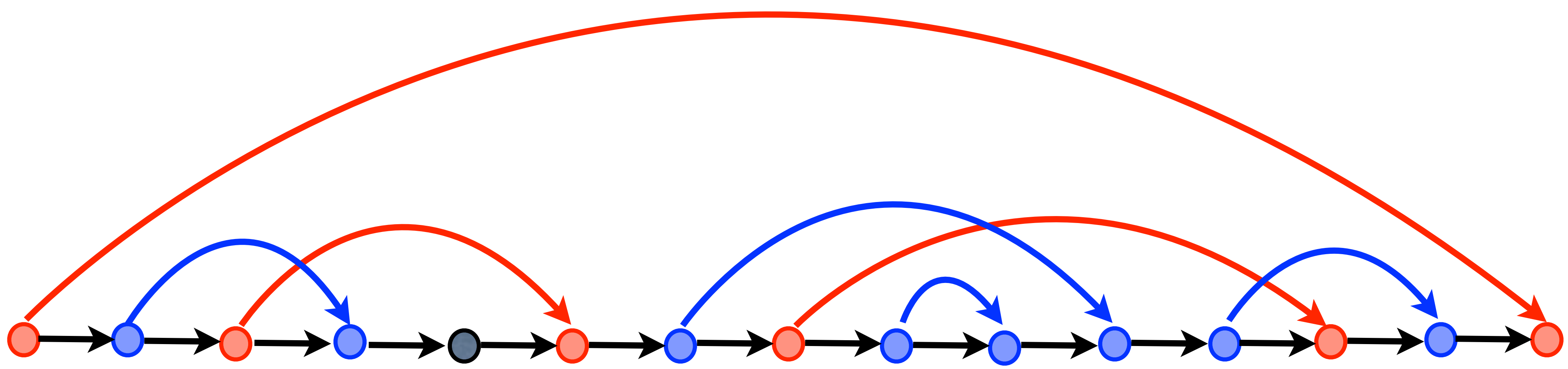


Any behaviour can be generated by the algebra

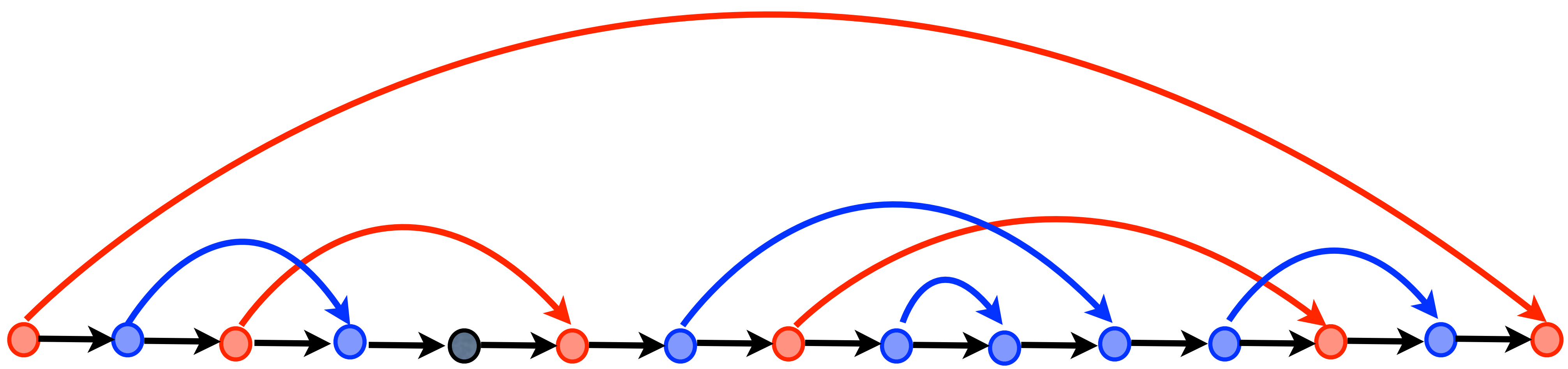
Any behaviour can be generated by the algebra

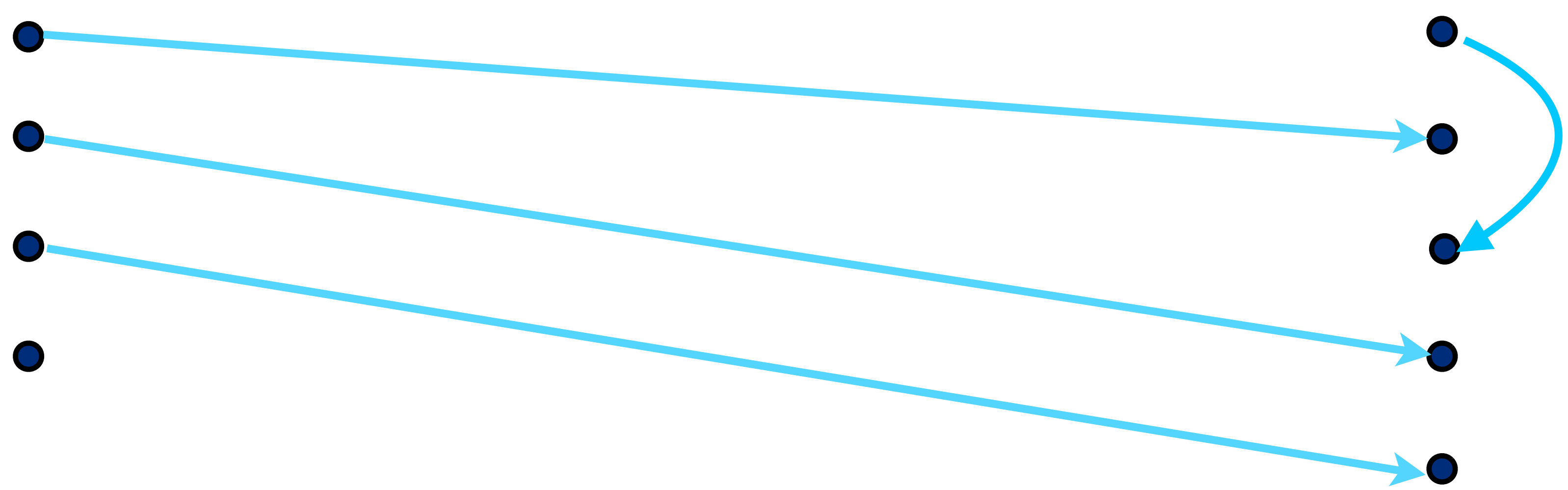


Any behaviour can be generated by the algebra

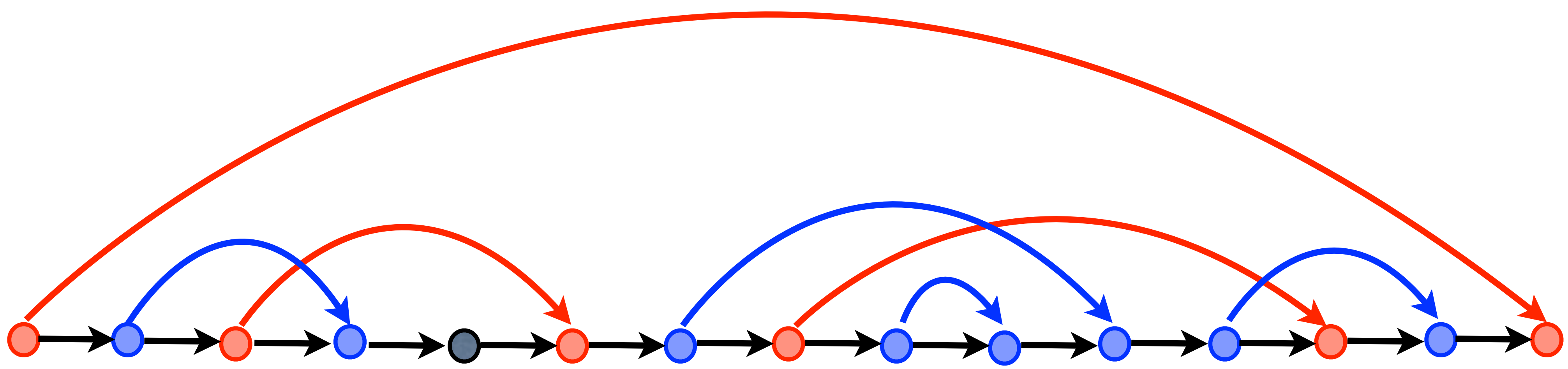


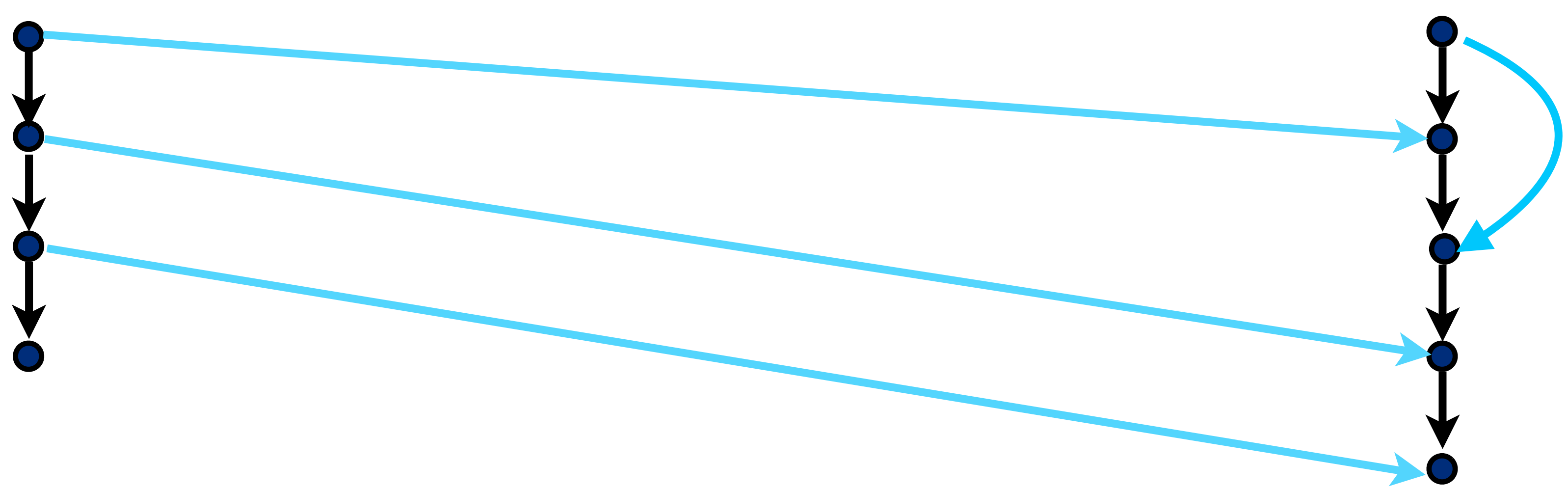
Any behaviour can be generated by the algebra





Any behaviour can be generated by the algebra

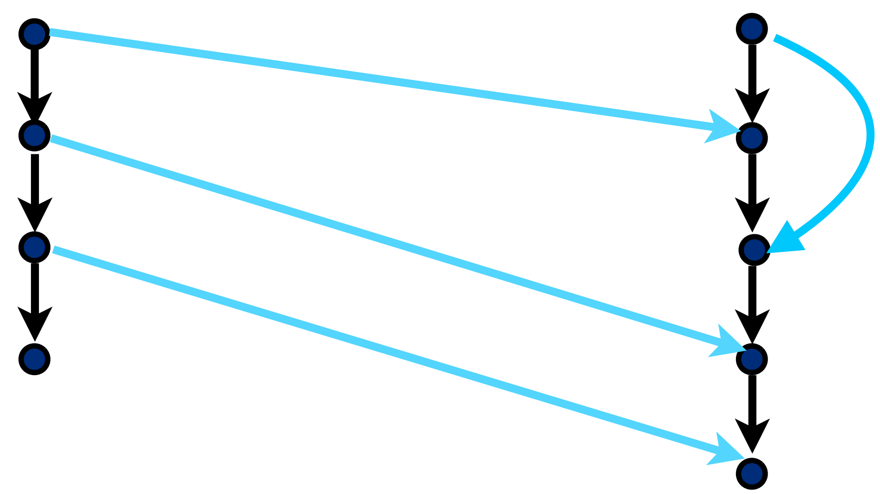




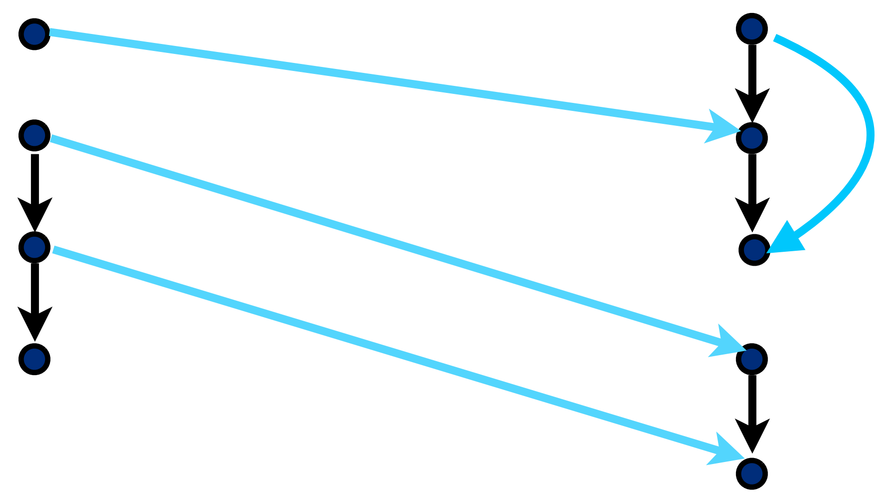
Bounded split-width ($\mathtt{k}$)

If a split-behaviour can be generated by the algebra, with the size of all the splits used ≤ $\mathtt{k}$

Example: an MSCN



Example: an MSCN

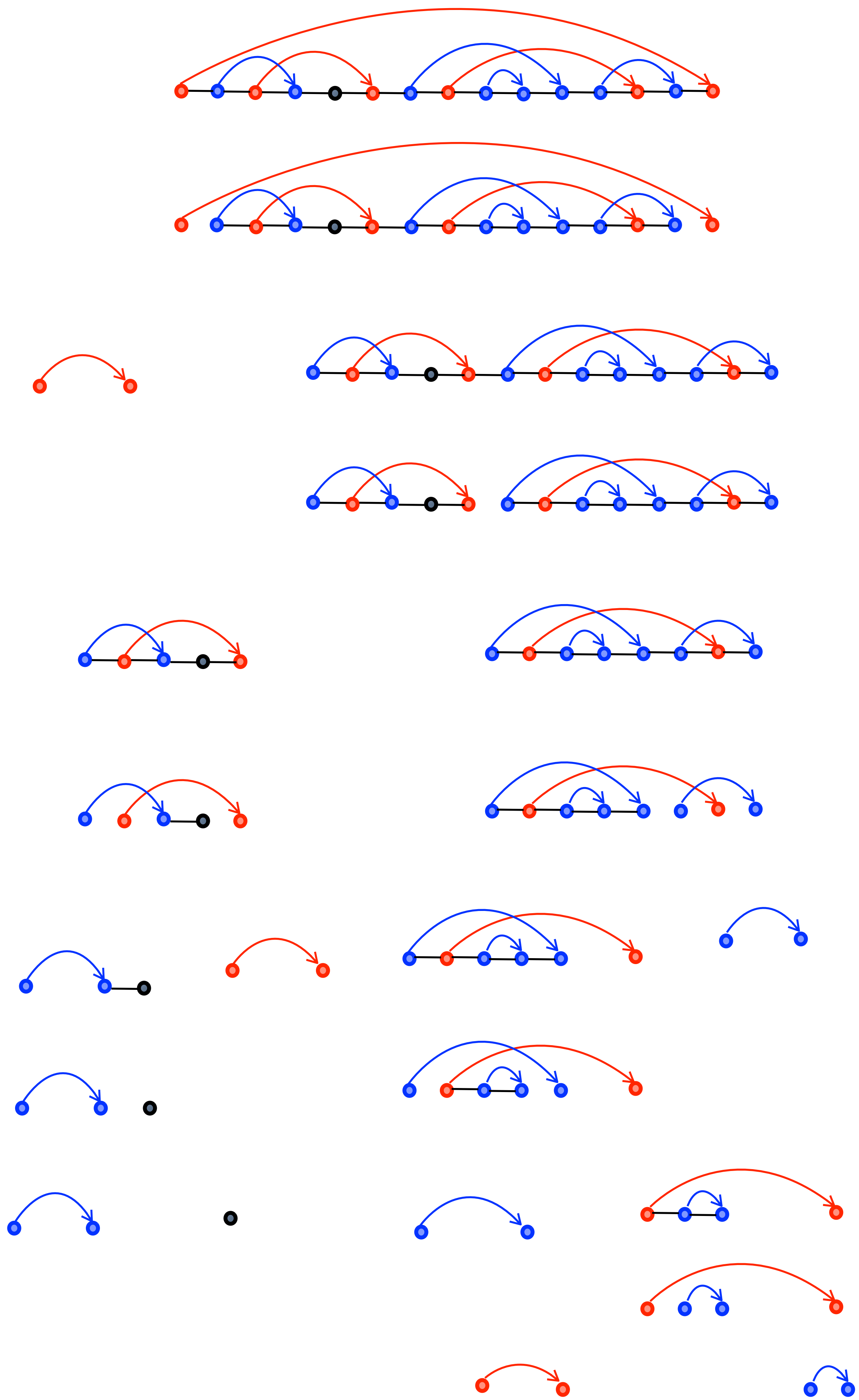


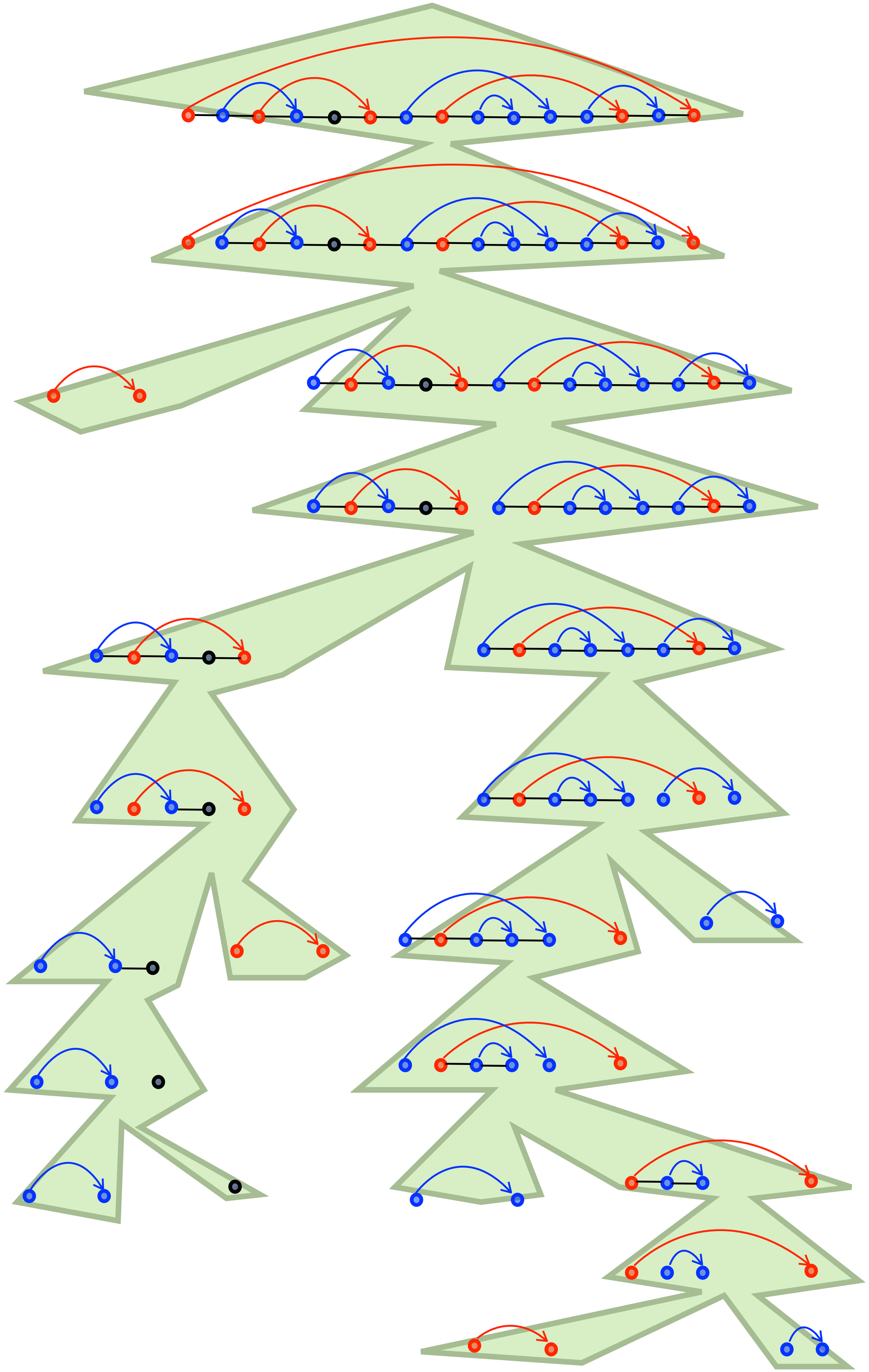
Example: an MSCN

Example: an MSCN

Example: an MSCN

Example: an MSCN





Model-Checking w.r.t Split-width k

Given a concurrent recursive program

Given two concurrent recursive programs

Model-Checking w.r.t Split-width k

Given a concurrent recursive program

- Is there an accepting run with split-width <= k?
- Does it accept all split-width <= k words?

Given two concurrent recursive programs

- Are the split-width k-behaviours of one contained in those of the other?

Model-Checking w.r.t Split-width k

Given a concurrent recursive program

- Is there an accepting run with split-width <= k?
- Does it accept all split-width <= k words?

Given two concurrent recursive programs

- Are the split-width k-behaviours of one contained in those of the other?

Abstract Derivation Trees

Split-width <=k Runs

Split-width <=k Runs

ADTs representing split-width k derivation trees form a regular tree language.

Easy tree automaton construction

Split-width <=k Runs

ADTs representing split-width k derivation trees form a regular tree language.

Easy tree automaton construction

ADTs representing derivation trees of split-width k accepting runs of a CRP is a regular tree language.

Easy tree automaton construction. Size of the automaton is exponential in k.

Decidability of Model-checking

Input

S : CRP over a given set of processes.
k : parameter (split-width)

Emptiness	ExpTime
Universality	2-ExpTime
Inclusion	2-ExpTime

Model-checking MSO formulas

Given a formula φ over MSCNs we construct a formula ψ over ADTs such that

The interpretation:

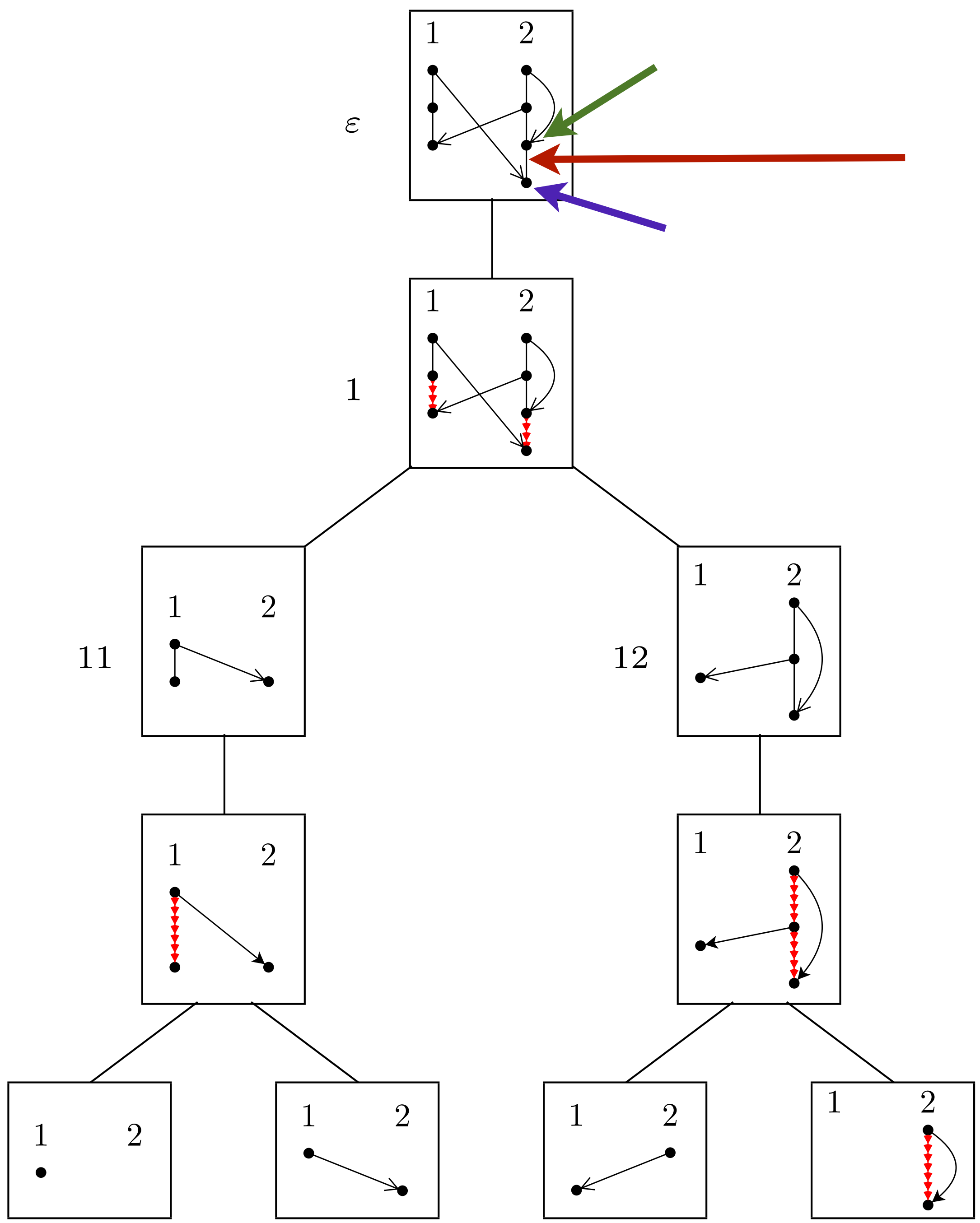
Model-checking MSO formulas

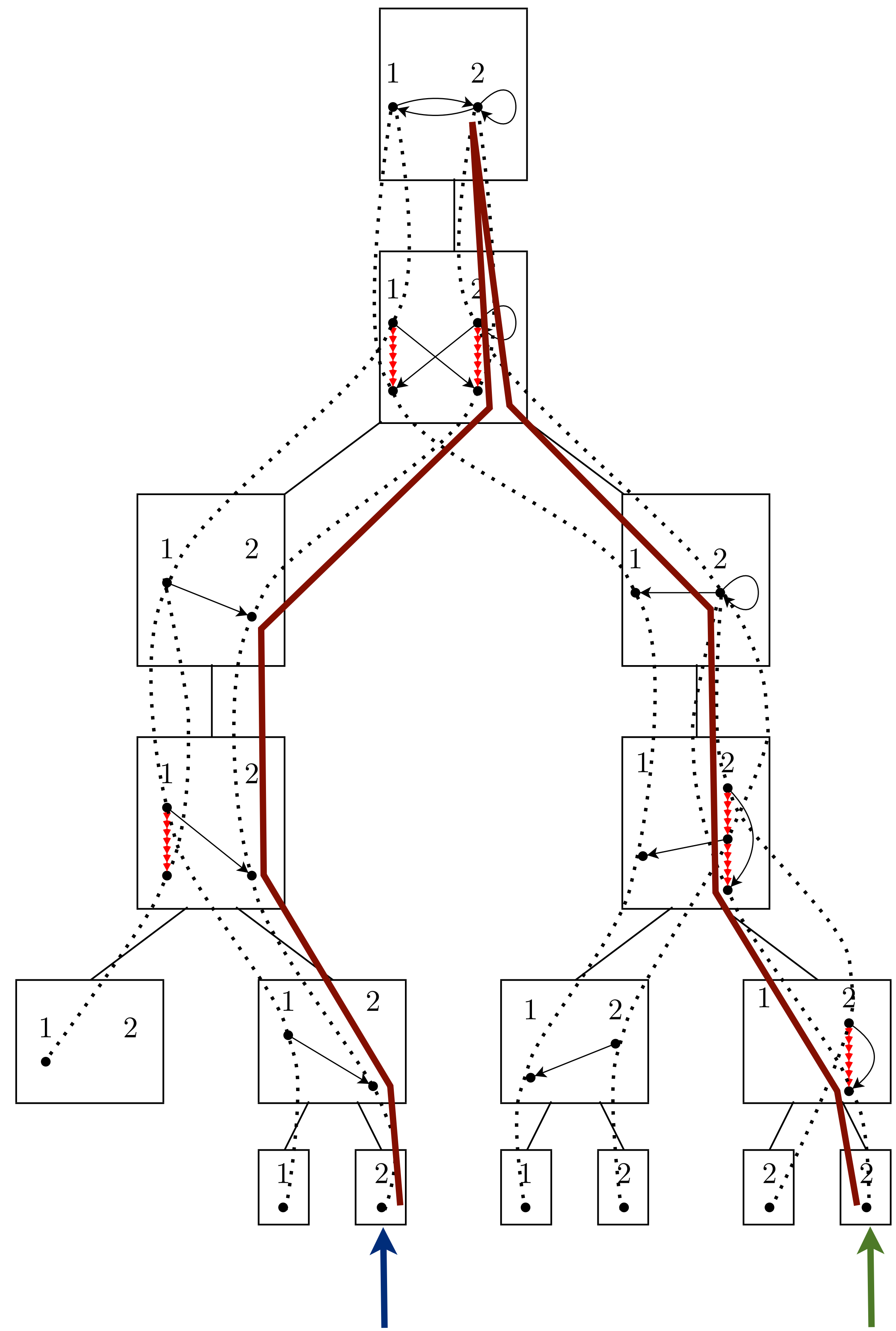
Given a formula φ over MSCNs we construct a formula ψ over ADTs such that

> For any MSCN M, $M \models \varphi$ iff $T \models \psi$ for any ADT T representing a split-width k derivation of M.

The interpretation:

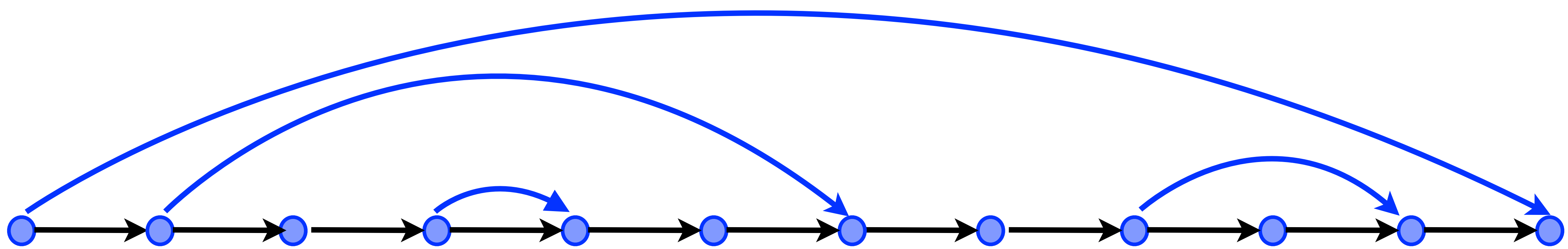
- The domain is the set of leaves.
- Message, Nesting are checked examining “common” parent.
- Only process successor needs little bit of work


ε
1
11
12



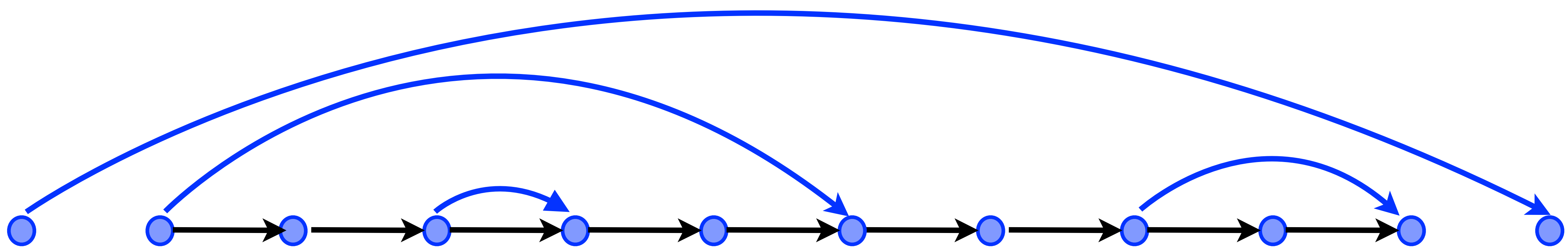
Nested Words

Nested words have split-width ≤ 3



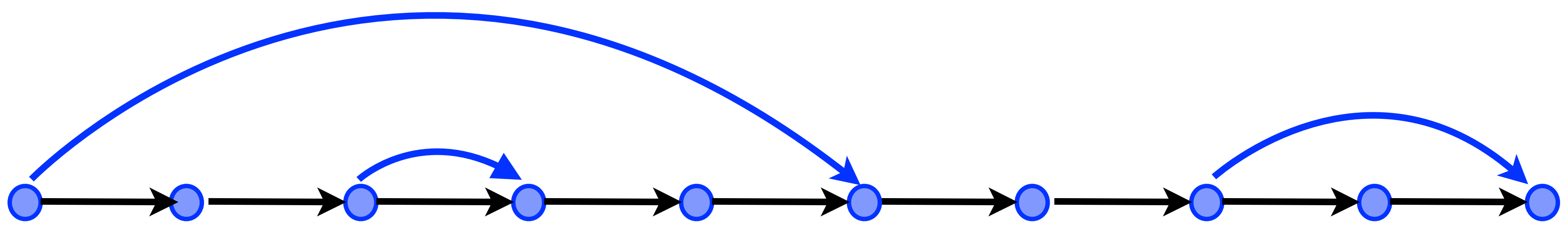
Nested Words

Nested words have split-width ≤ 3



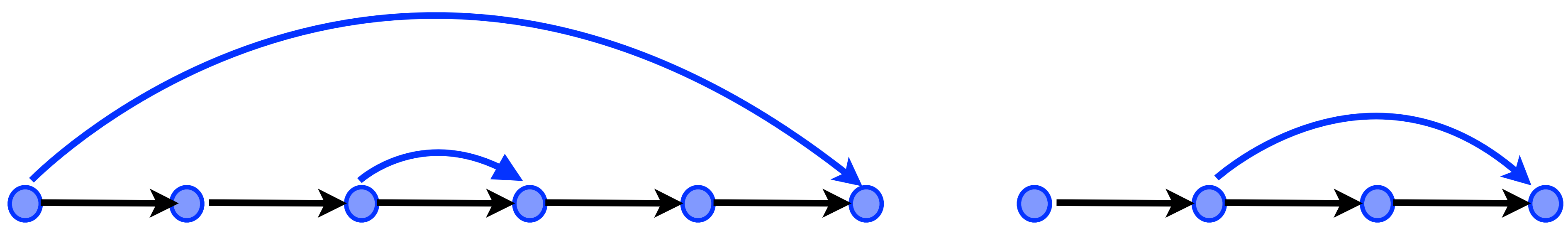
Nested Words

Nested words have split-width ≤ 3



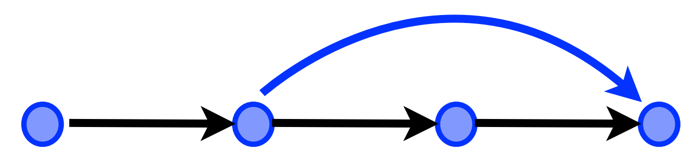
Nested Words

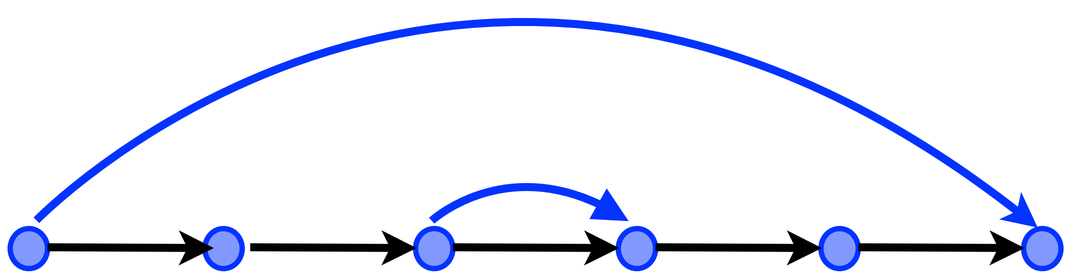
Nested words have split-width ≤ 3



Nested Words

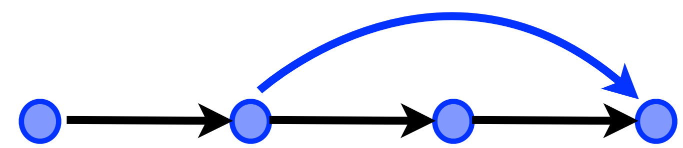
Nested words have split-width ≤ 3

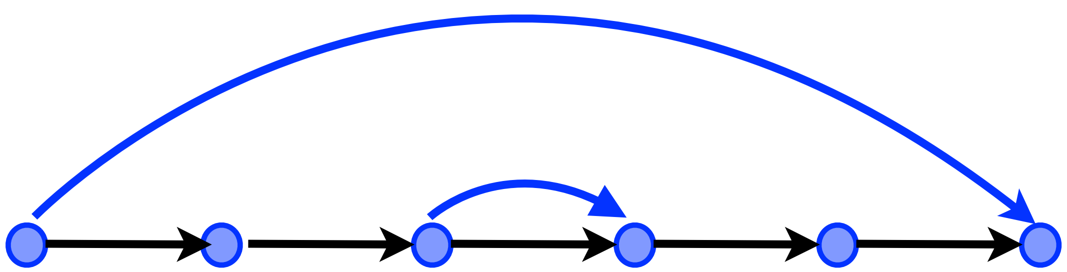




Nested Words

Nested words have split-width ≤ 3

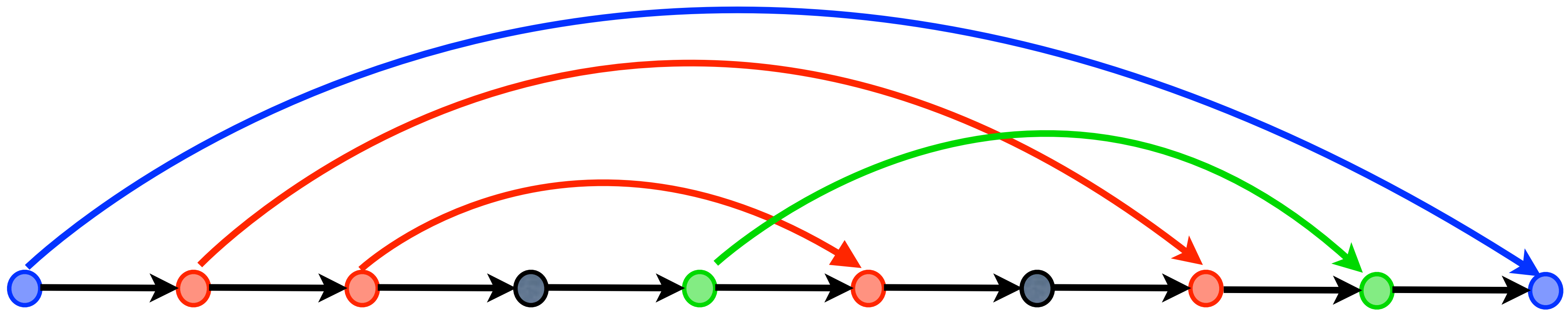




Theorem. MSO is decidable over nested words (VPLs).

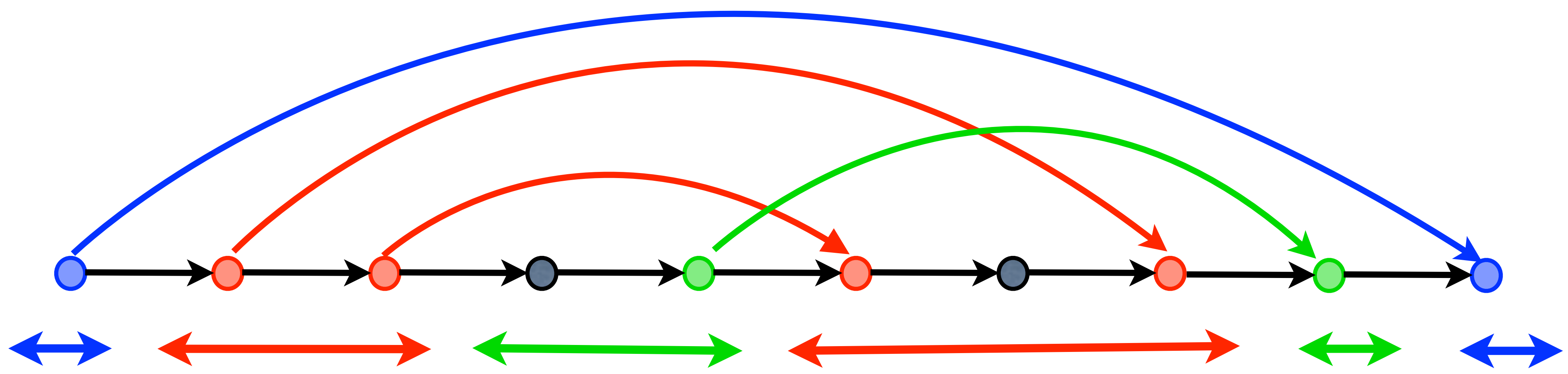
Bounded scope multi-pushdown systems

A *context* is a set of consecutive positions which involves at most one stack.



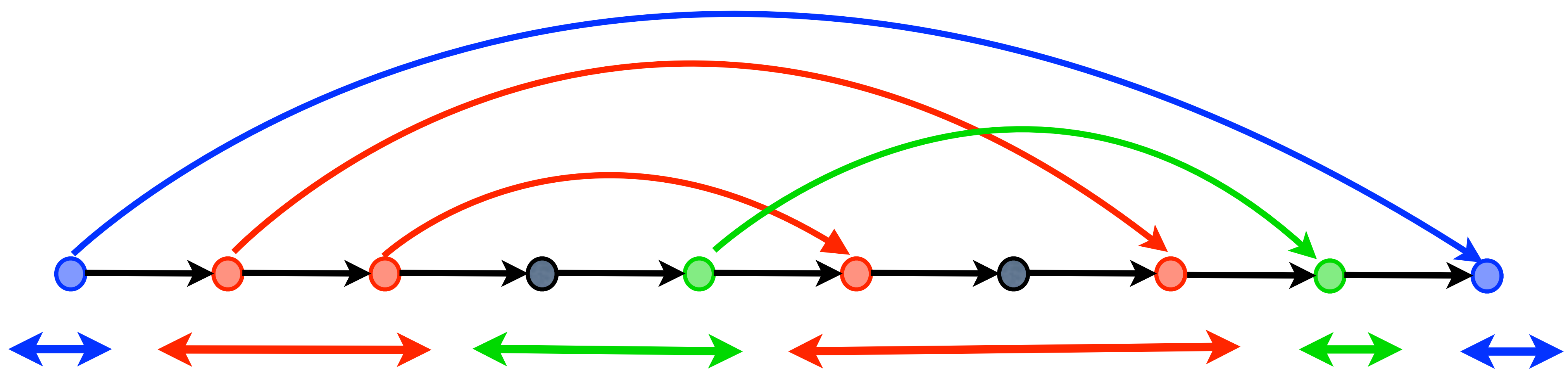
Bounded scope multi-pushdown systems

A *context* is a set of consecutive positions which involves at most one stack.



Bounded scope multi-pushdown systems

A *context* is a set of consecutive positions which involves at most one stack.

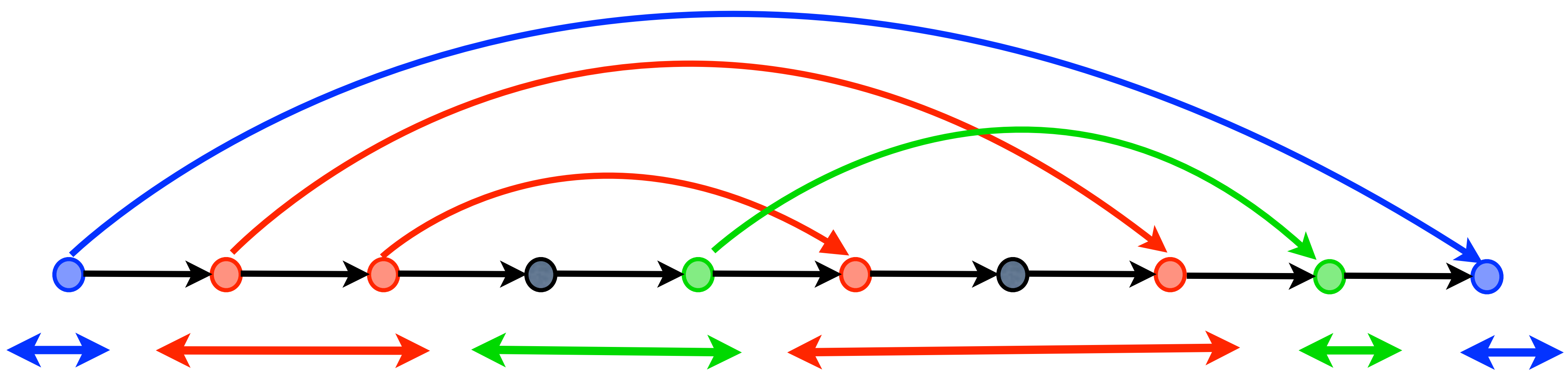


Bounded Scope MNWs: Fix parameter m. For any nesting edges, no more than m different contexts between its source and target.

S. La Torre and M. Napoli. Reachability of multistack pushdown systems with scope-bounded matching relations. In J.-P. Katoen and B. König, editors, *CONCUR*, volume 6901, pages 203–218. Springer, 2011.

Bounded scope multi-pushdown systems

A *context* is a set of consecutive positions which involves at most one stack.



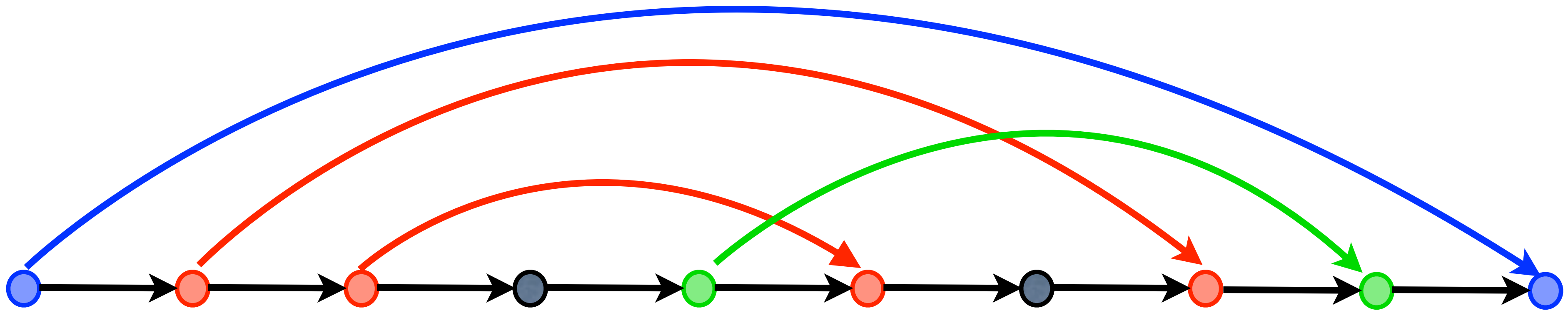
Bounded Scope MNWs: Fix parameter m. For any nesting edges, no more than m different contexts between its source and target.

Theorem. S-W at most m + 2.

S. La Torre and M. Napoli. Reachability of multistack pushdown systems with scope-bounded matching relations. In J.-P. Katoen and B. König, editors, *CONCUR*, volume 6901, pages 203–218. Springer, 2011.

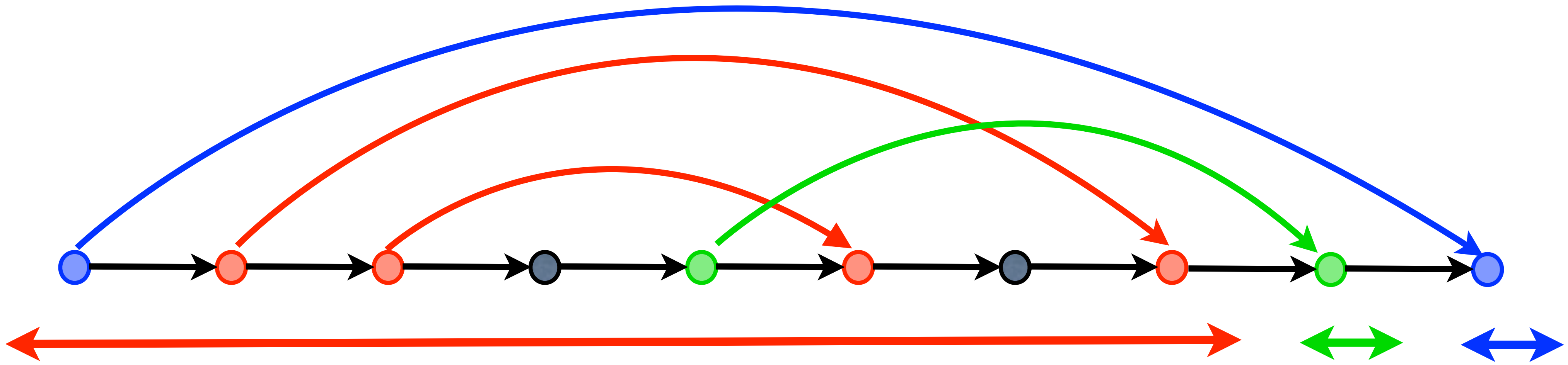
Bounded phase multi-pushdown systems

A *phase* is a set of consecutive positions which involves at most one stack.



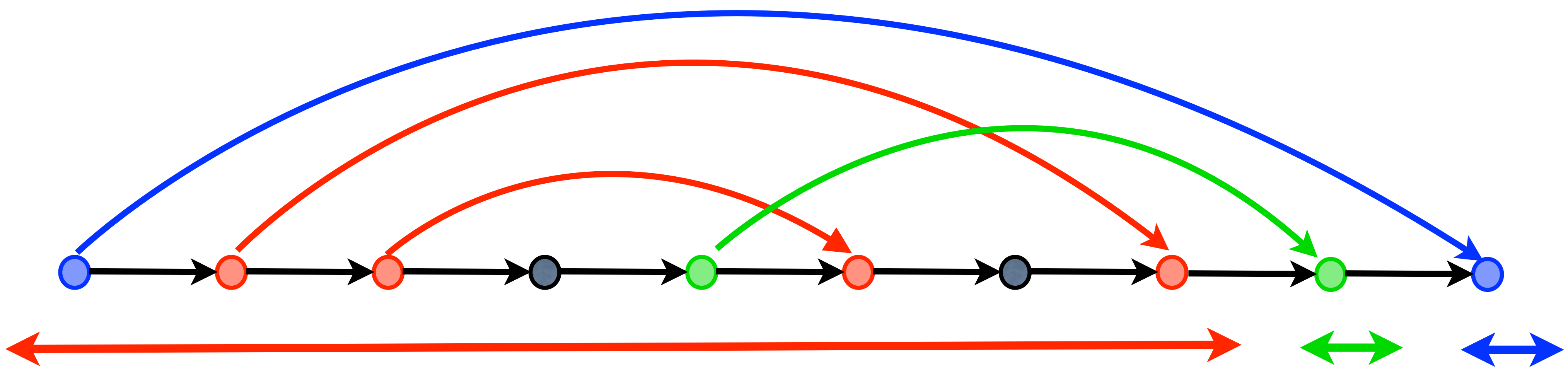
Bounded phase multi-pushdown systems

A *phase* is a set of consecutive positions which involves at most one stack.



Bounded phase multi-pushdown systems

A *phase* is a set of consecutive positions which involves at most one stack.

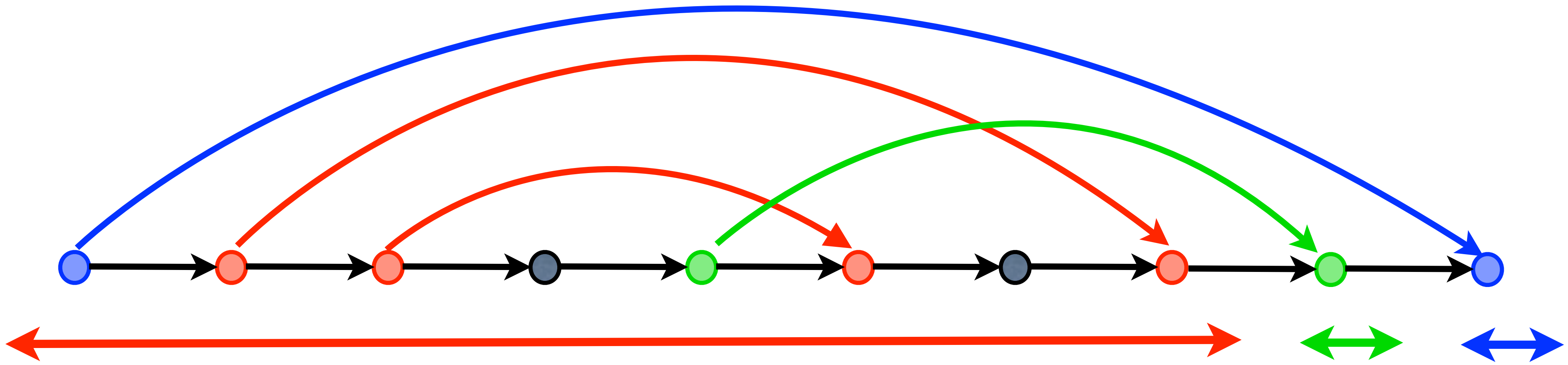


Bounded Phase MNWs: Fix parameter p. At most p phases.

S. La Torre, P. Madhusudan, and G. Parlato. A robust class of context-sensitive languages. In *LICS*, pages 161–170. IEEE Computer Society, 2007.

Bounded phase multi-pushdown systems

A *phase* is a set of consecutive positions which involves at most one stack.

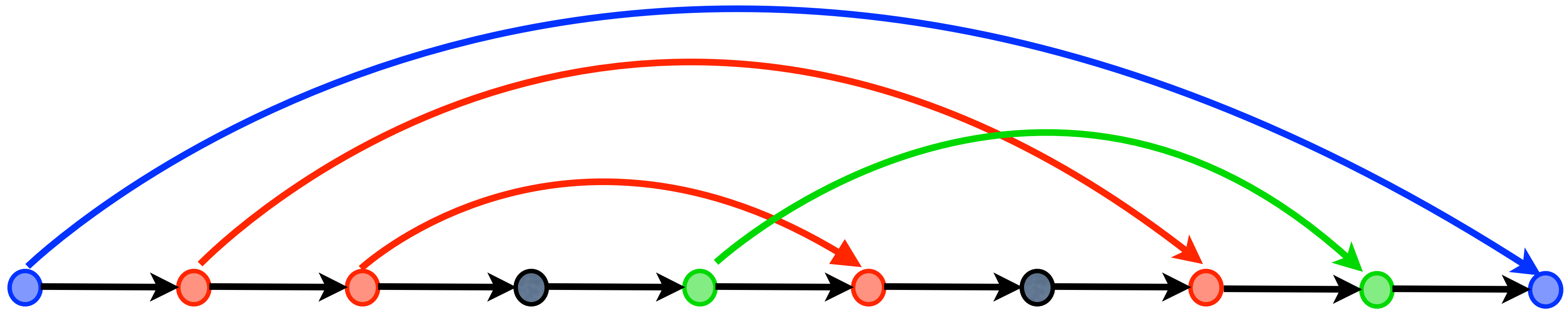


Bounded Phase MNWs: Fix parameter p. At most p phases.

> *Theorem*. S-W at most 2^p .

S. La Torre, P. Madhusudan, and G. Parlato. A robust class of context-sensitive languages. In *LICS*, pages 161–170. IEEE Computer Society, 2007.

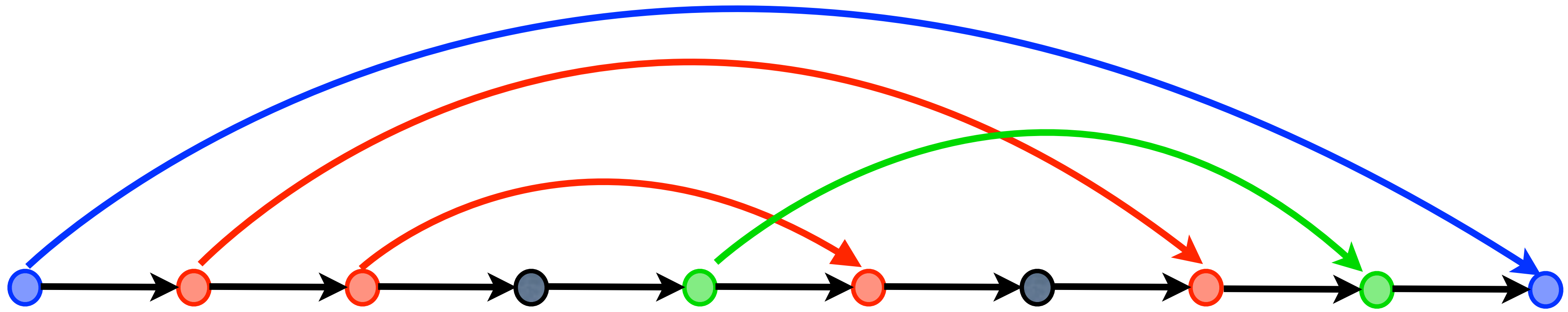
Ordered multi-pushdown systems



Ordered MNWs: Priority among the stacks. Returns agree with the priority. When a stack pops, all higher priority stacks are empty.

M. F. Atig, B. Bollig, and P. Habermehl. Emptiness of multi-pushdown automata is 2ETIME-Complete. In M. Ito and M. Toyama, editors, *Developments in Language Theory*, volume 5257, pages 121–133. Springer, 2008.
L. Breveglieri, A. Cherubini, C. Citrini, and S. Crespi-Reghizzi. Multi-pushdown languages and grammars. *Int. J. Found. Comput. Sci.*, 7(3):253–292, 1996.

Ordered multi-pushdown systems



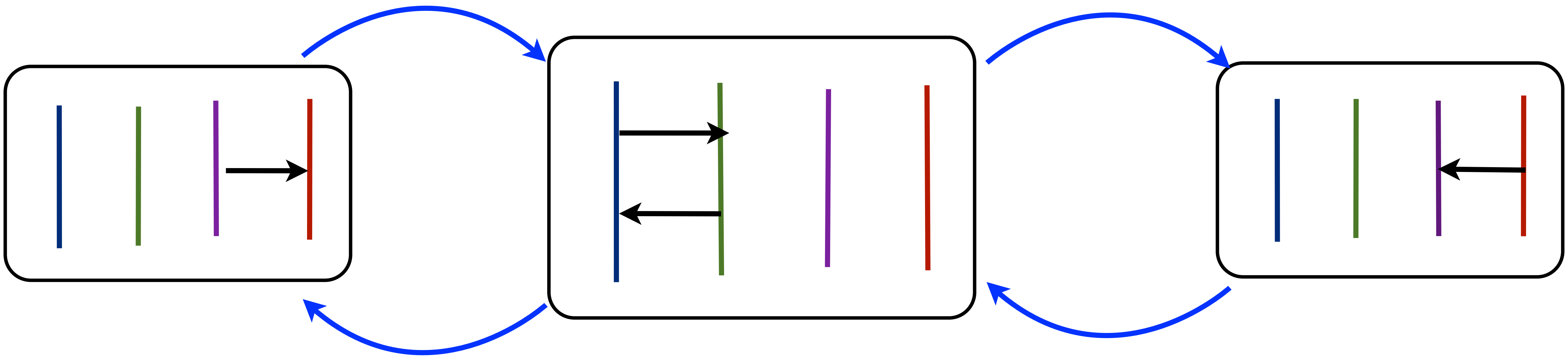
Ordered MNWs: Priority among the stacks. Returns agree with the priority. When a stack pops, all higher priority stacks are empty.

Theorem. S-W at most 2^s .

M. F. Atig, B. Bollig, and P. Habermehl. Emptiness of multi-pushdown automata is 2ETIME-Complete. In M. Ito and M. Toyama, editors, *Developments in Language Theory*, volume 5257, pages 121–133. Springer, 2008.

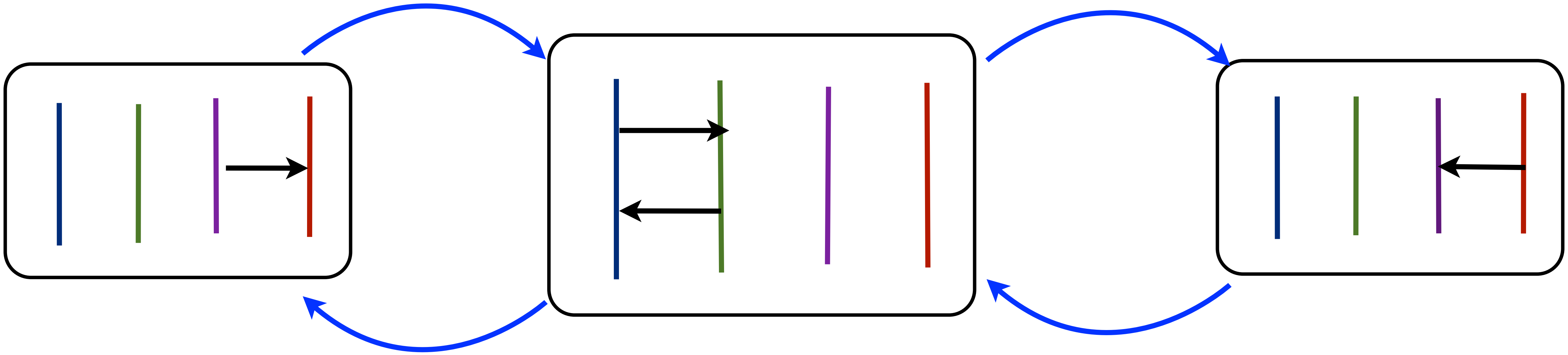
L. Breveglieri, A. Cherubini, C. Citrini, and S. Crespi-Reghizzi. Multi-pushdown languages and grammars. *Int. J. Found. Comput. Sci.*, 7(3):253–292, 1996.

HMSCs (or MSGs)



Split-width bounded by the maximum split-width of constituent MSCs

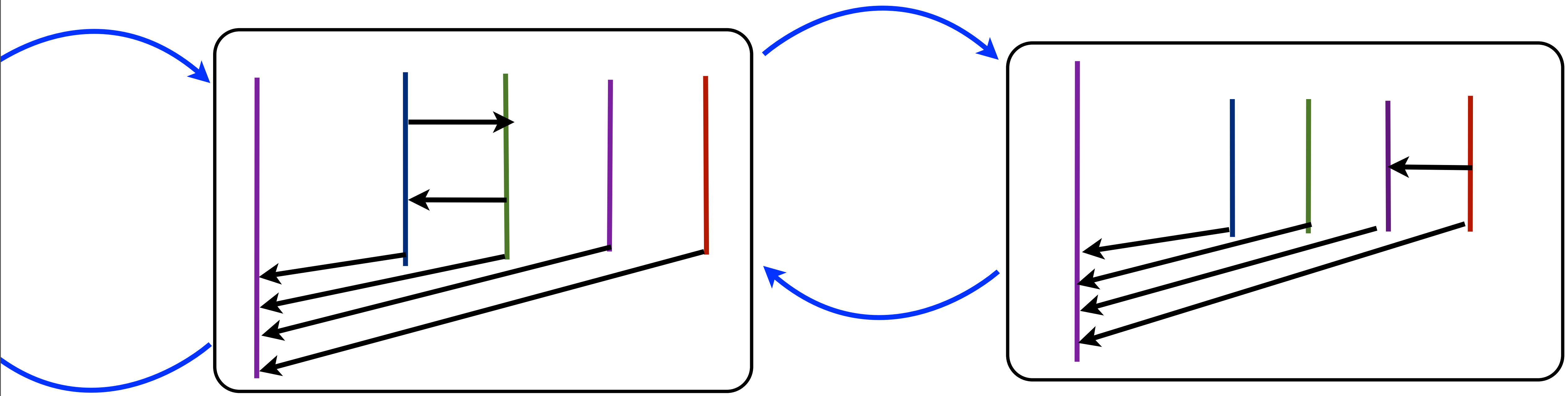
HMSCs (or MSGs)



Split-width bounded by the maximum split-width of constituent MSCs

Unlike CRPs, language is not MSO definable.

HMSCs ...



Add one process and edges to it in each node.

Language of this HMSC is MSO definable.

Obvious translation for MSO formulas via relativization.

Tree-width/Clique-width

MSO decidability follows.

Technical argument, normalizing derivation trees.

Split-width is a "special case" that is easier to use in the case of behaviours of CRPs.

Tree-width/Clique-width

- Easy translation from split-width to Tree/Clique width

  MSO decidability follows.

- Clique-width to Split-width with linear blow up

  Technical argument, normalizing derivation trees.

Split-width is a “special case” that is easier to use in the case of behaviours of CRPs.

Conclusions

- Split-width: a metric for under-approximate verification

  Equivalent to tree width in power

- Provides a simple technique to prove decidability of all known classes.

  Visual, simple inductive reasoning, limited number of cases to consider.

- Different view, suggests new "natural" classes.

- Schedulable subclasses.

  Restrict to only verified behaviours