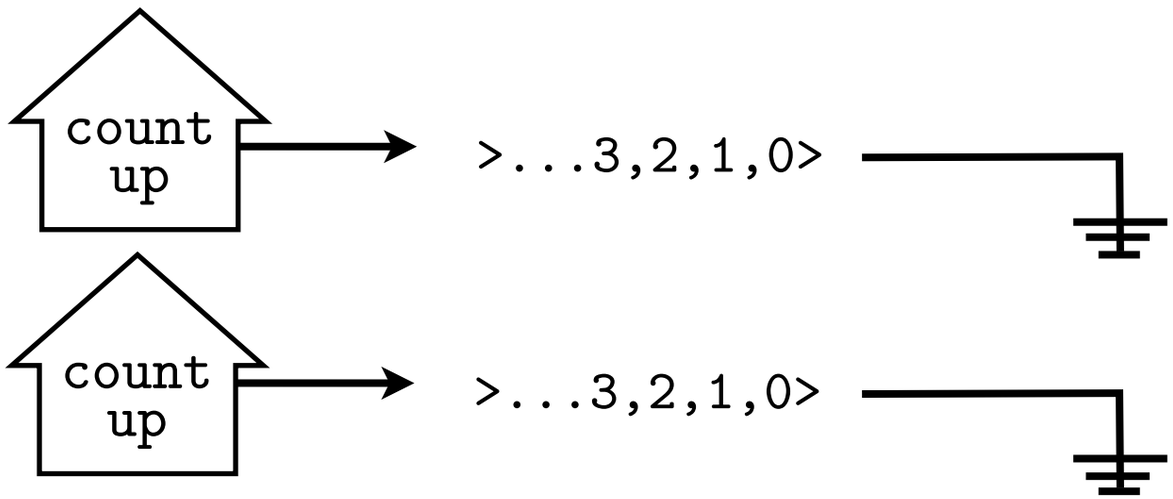
CS294 - Hardware Languages - Week 3

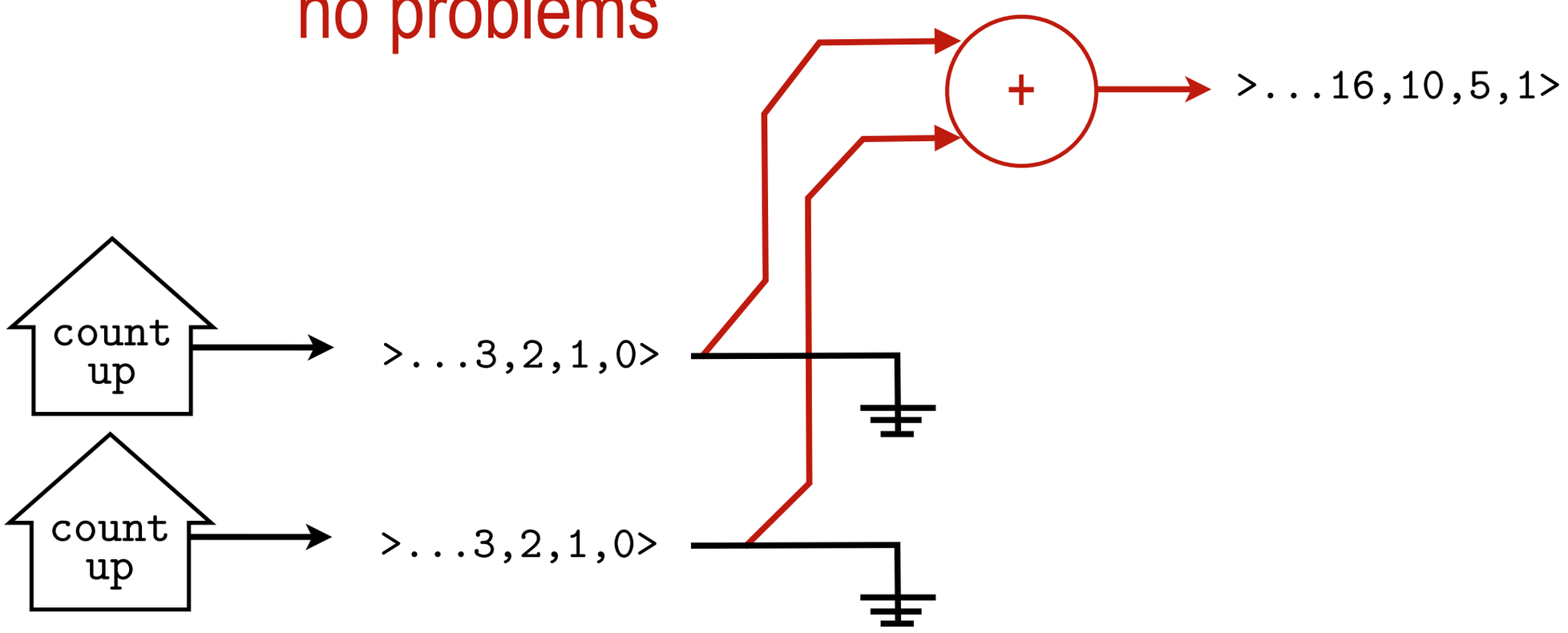
- I have two [multi-stage] slides [after this one]
- Two problems that I spend a lot of time thinking about
  - Both have to do with *bounded buffering*
- Trends
  - Software design is a programming task. Hardware design is becoming more of a programming task.
  - Hardware is concurrent and parallel. Software is becoming more concurrent and parallel.
  - As the two disciplines grow towards each other, I speculate that *bounded buffering* is the fundamental division between them that will remain after the dust settles.

A Transactor Network

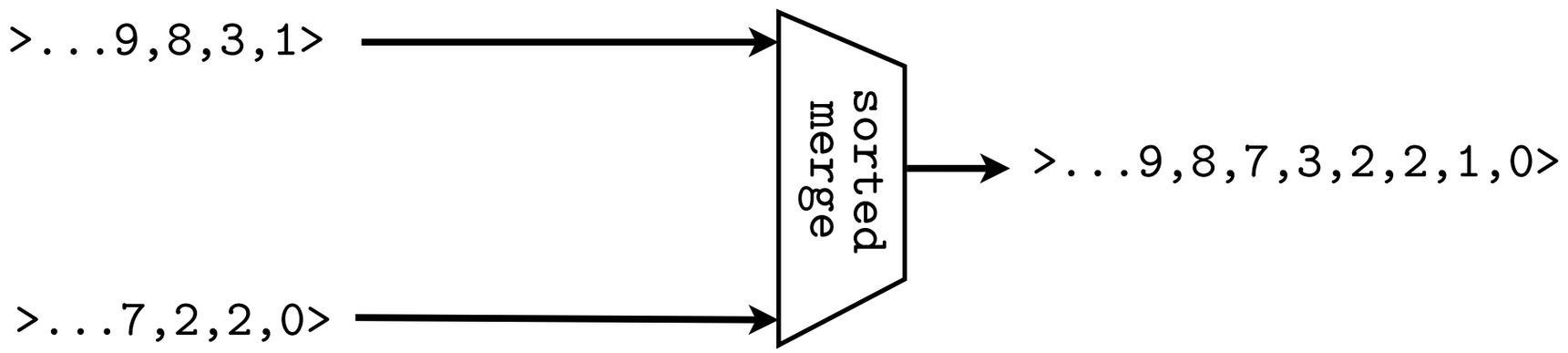


A Transactor Network

This addition causes no problems

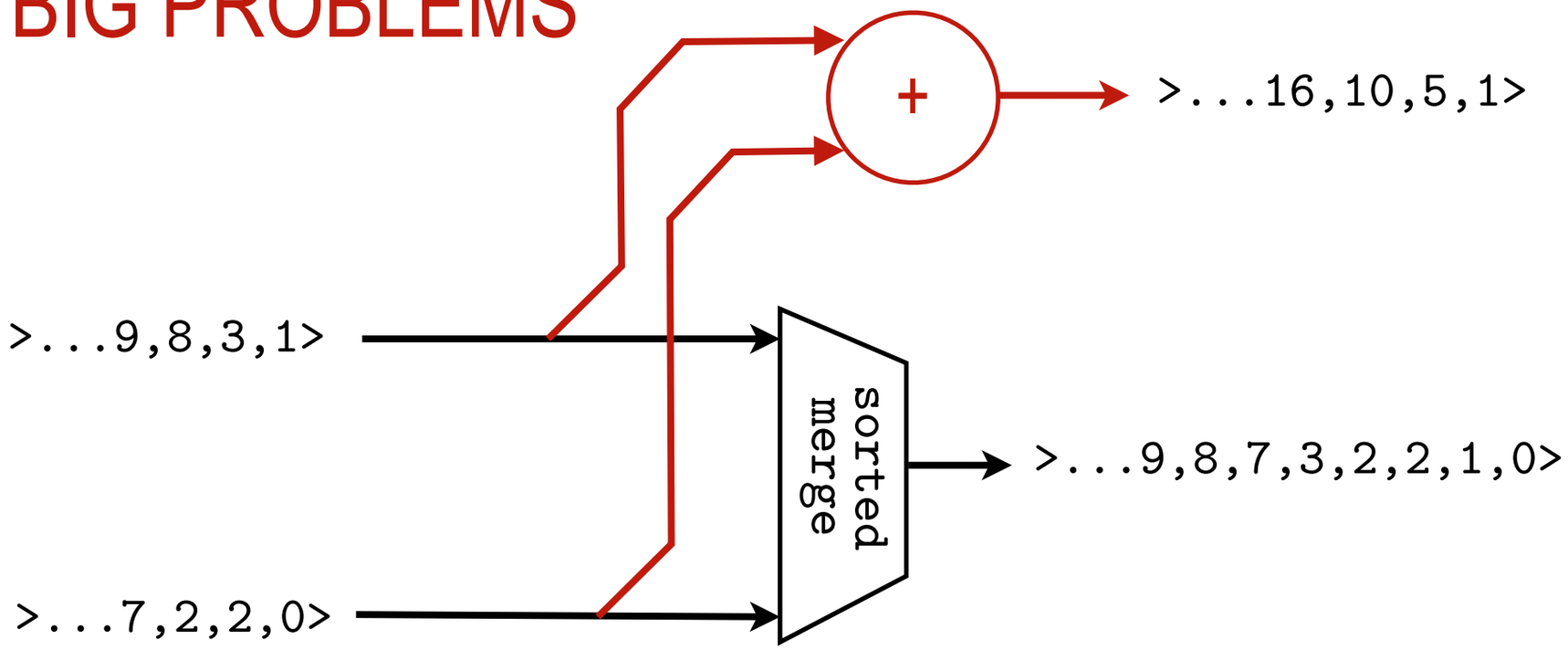


Another Transactor Network

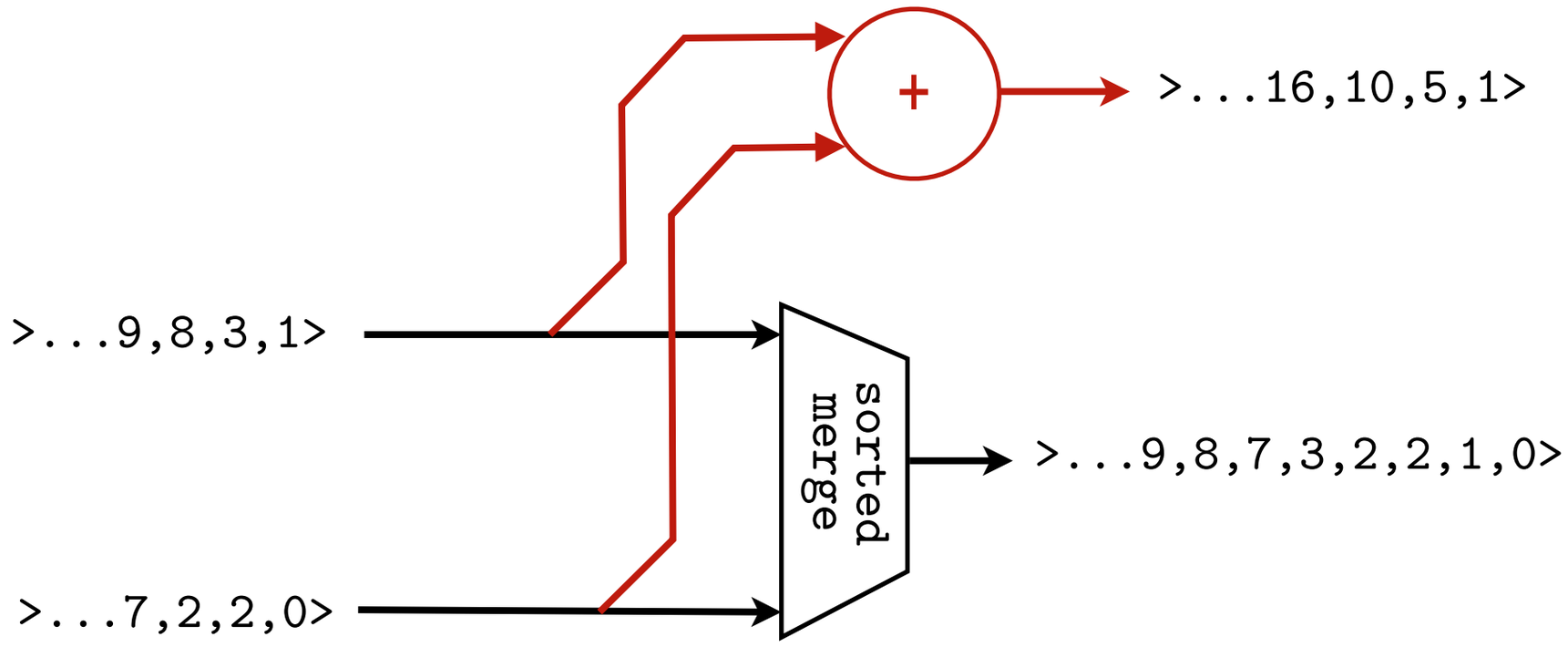


Another Transactor Network

This addition causes
BIG PROBLEMS



Another Transactor Network



No finite amount of buffering on the input channels is sufficient. ∃loise can always beat ∀belard.

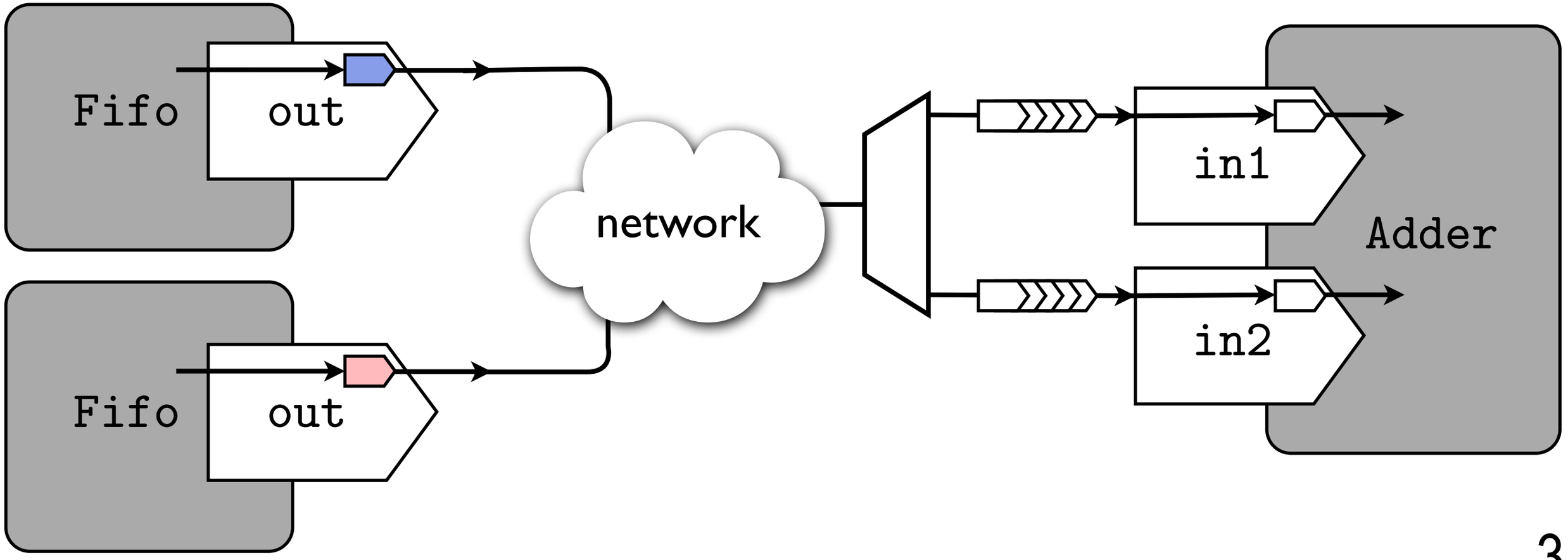
Clogging Networks

- A network at *transactor granularity* cannot afford:
  - Retransmit buffers at every sender
  - Reordering buffers at every reciever
  - Sequence numbers and timers
  - Bufferlock recovery (RAW/Tilera/Parks)

- Therefore, the network cannot simply imitate the Internet
  - Cannot drop packets in response to congestion
  - Must offer reliable delivery (and wrap unreliable links)
  - *But this raises serious clogging issues*!

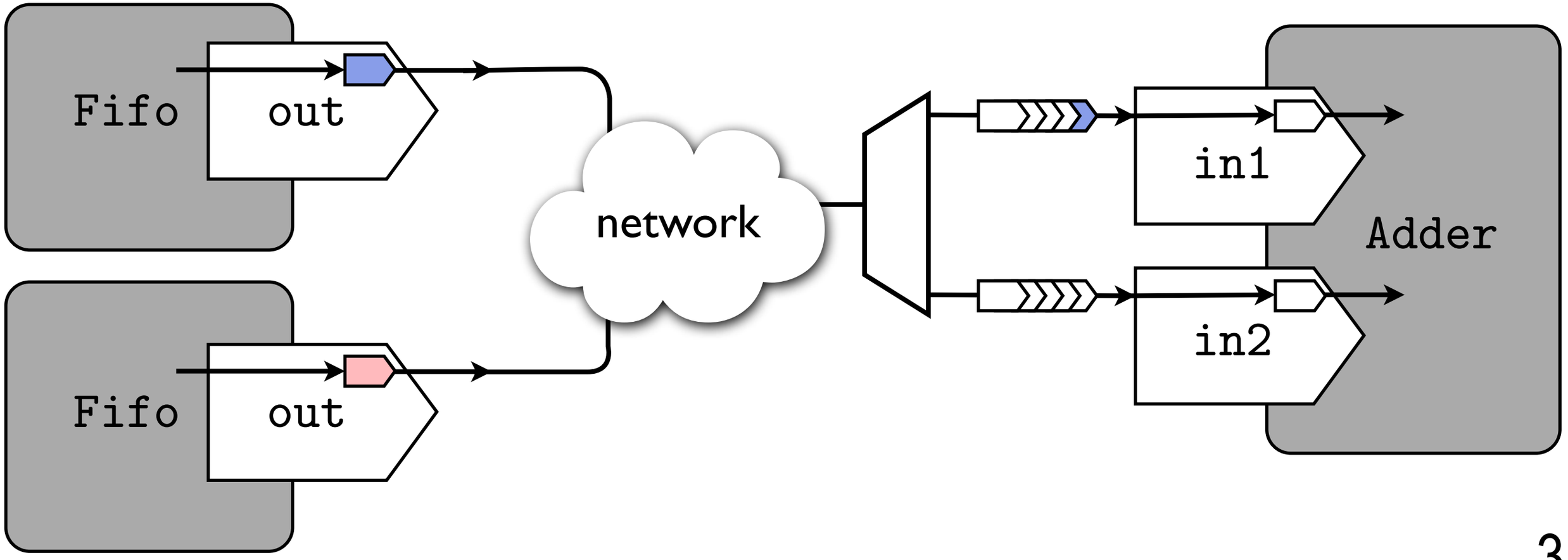
Clogging Networks

- Suppose `Adder` waits for pairs of items to become available and consumes them pairwise.
  - If items sent from the two outputs are interleaved properly, this will work.



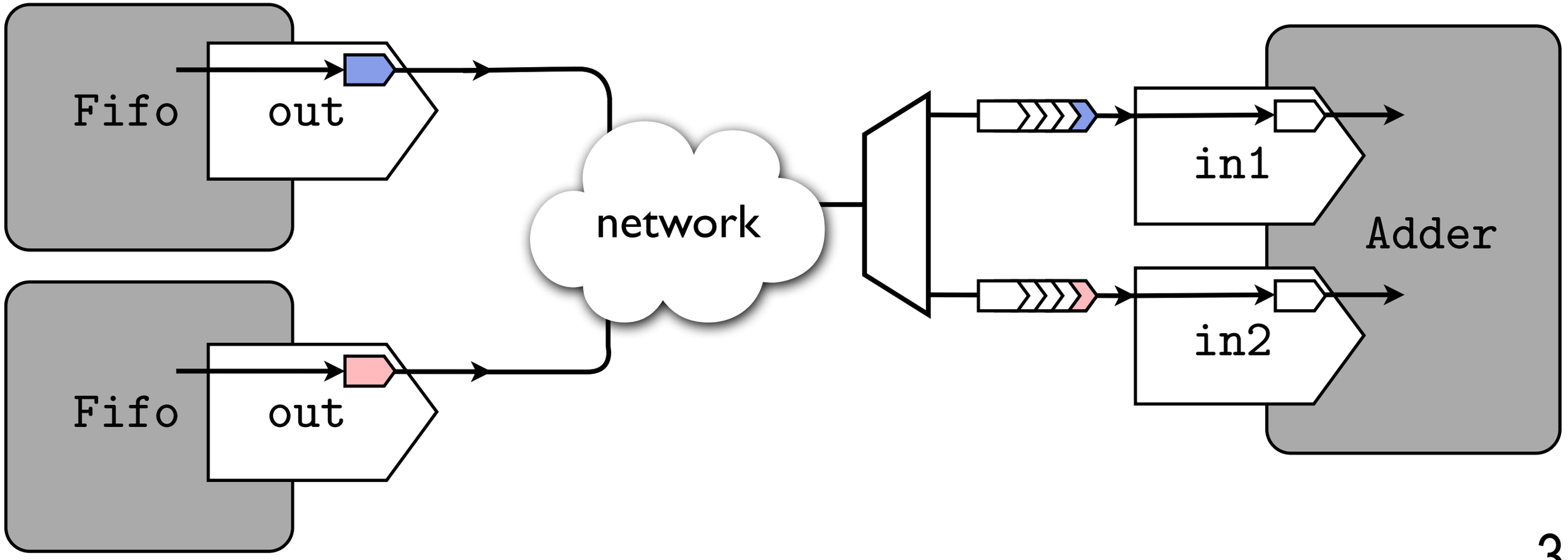
Clogging Networks

- Suppose `Adder` waits for pairs of items to become available and consumes them pairwise.
  - If items sent from the two outputs are interleaved properly, this will work.



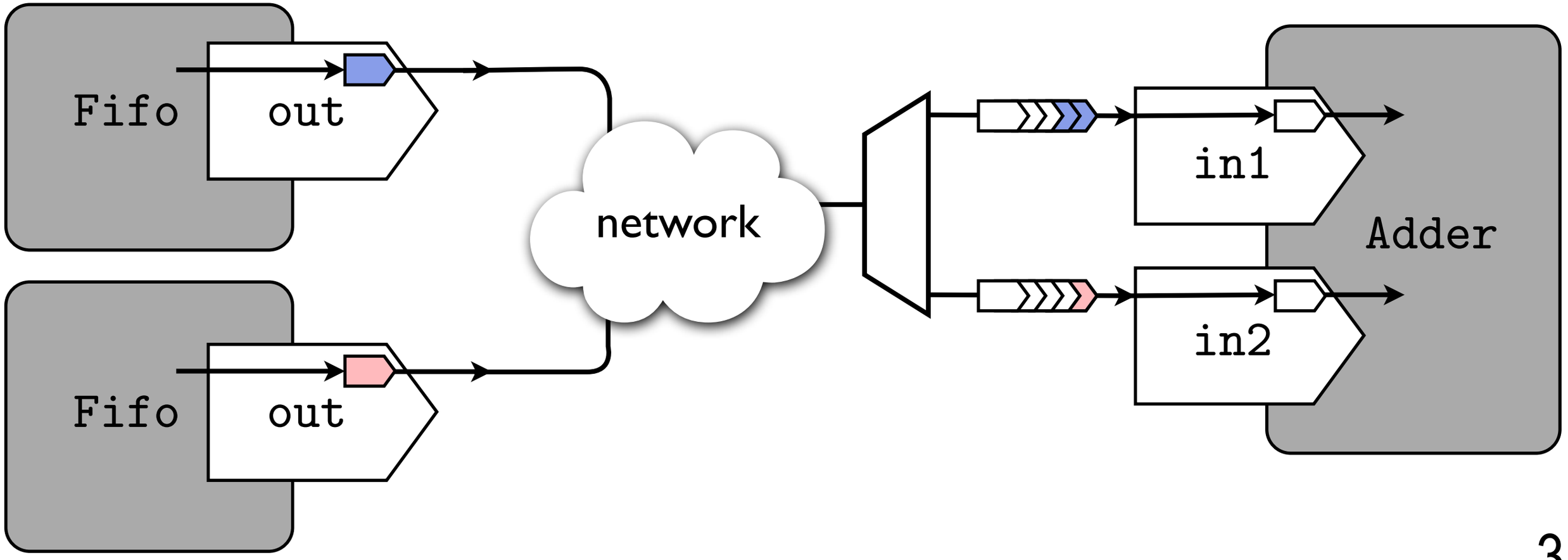
Clogging Networks

- Suppose `Adder` waits for pairs of items to become available and consumes them pairwise.
  - If items sent from the two outputs are interleaved properly, this will work.



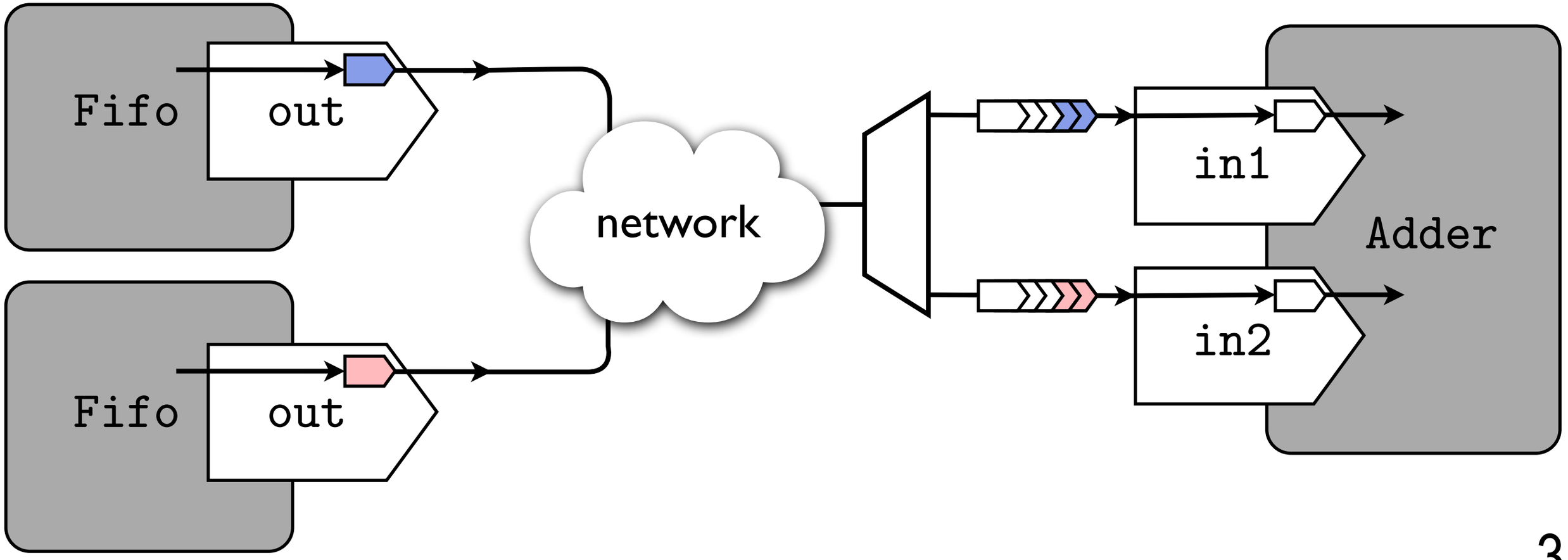
Clogging Networks

- Suppose `Adder` waits for pairs of items to become available and consumes them pairwise.
  - If items sent from the two outputs are interleaved properly, this will work.



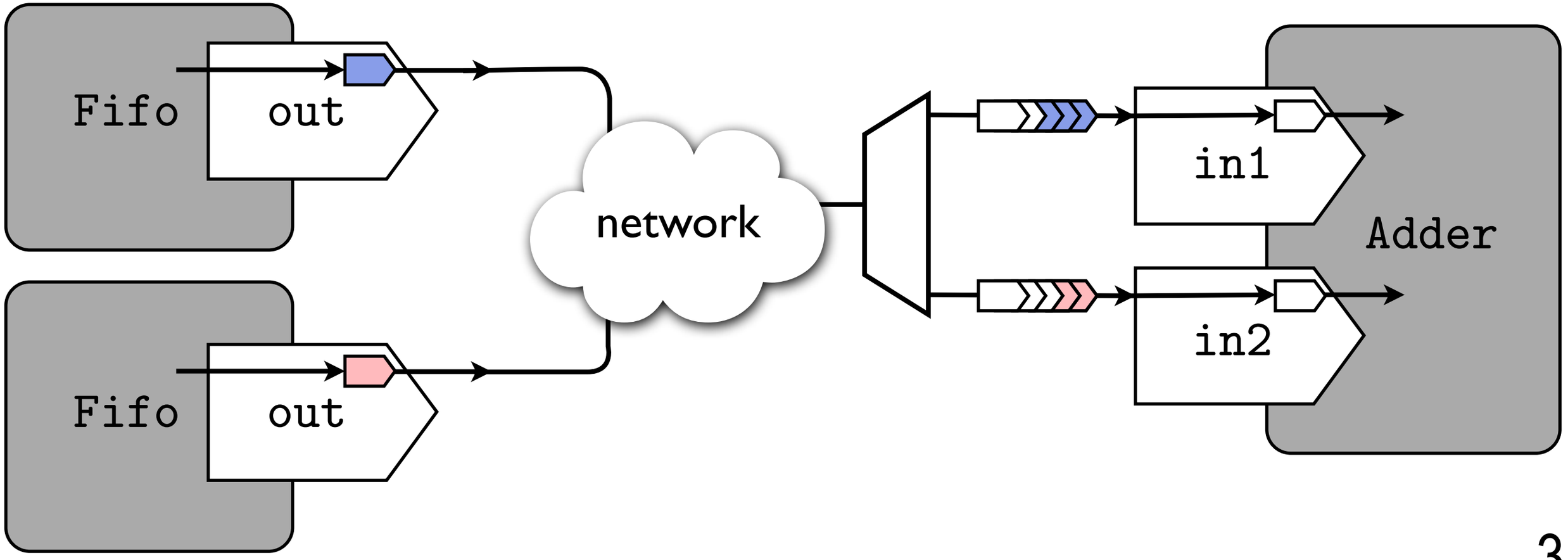
Clogging Networks

- Suppose `Adder` waits for pairs of items to become available and consumes them pairwise.
  - If items sent from the two outputs are interleaved properly, this will work.



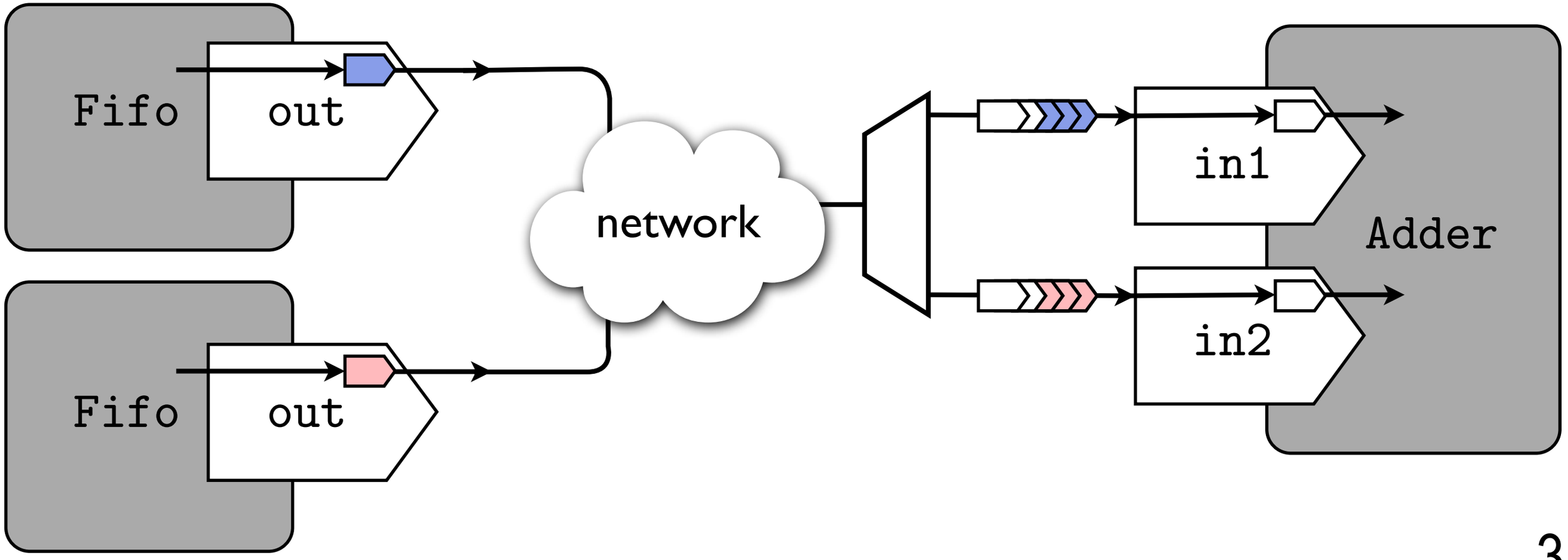
Clogging Networks

- Suppose `Adder` waits for pairs of items to become available and consumes them pairwise.
  - If items sent from the two outputs are interleaved properly, this will work.



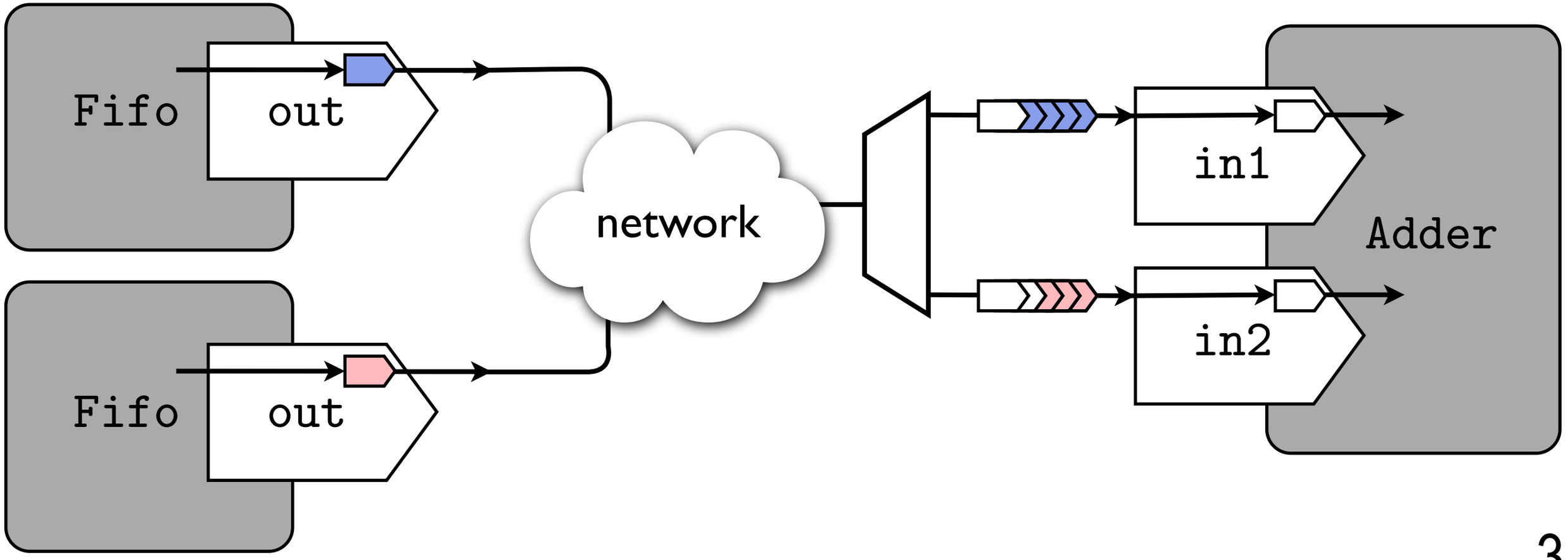
Clogging Networks

- Suppose `Adder` waits for pairs of items to become available and consumes them pairwise.
  - If items sent from the two outputs are interleaved properly, this will work.



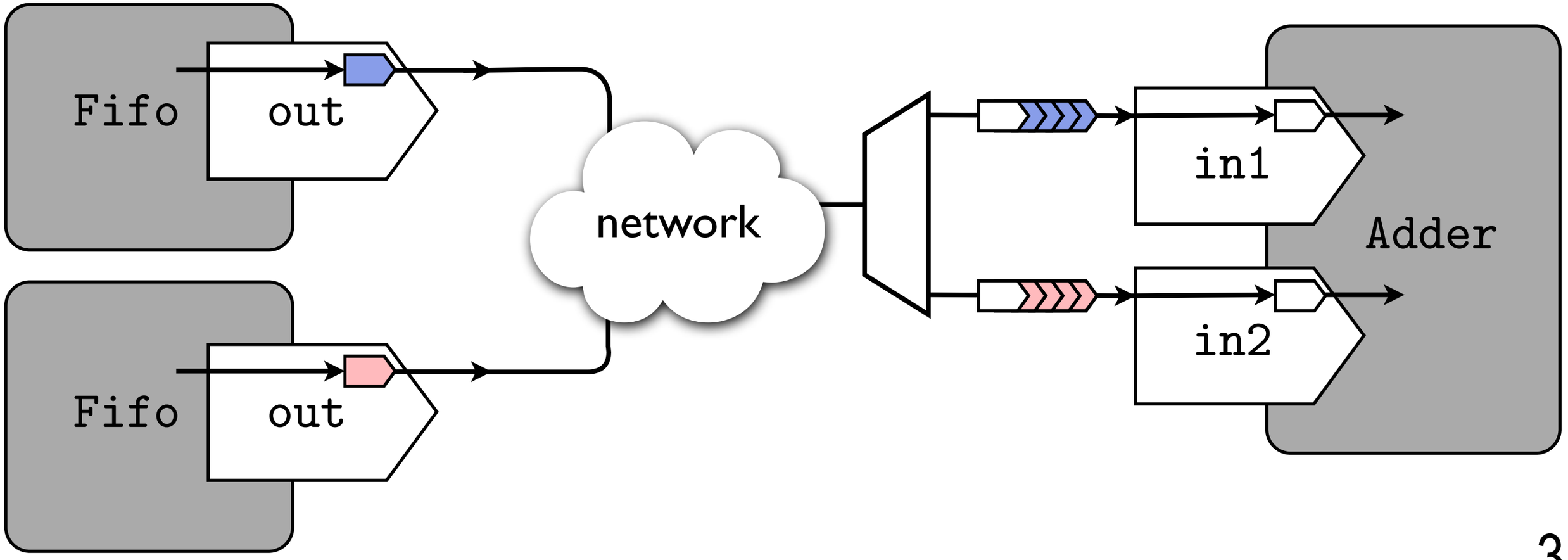
Clogging Networks

- Suppose `Adder` waits for pairs of items to become available and consumes them pairwise.
  - If items sent from the two outputs are interleaved properly, this will work.



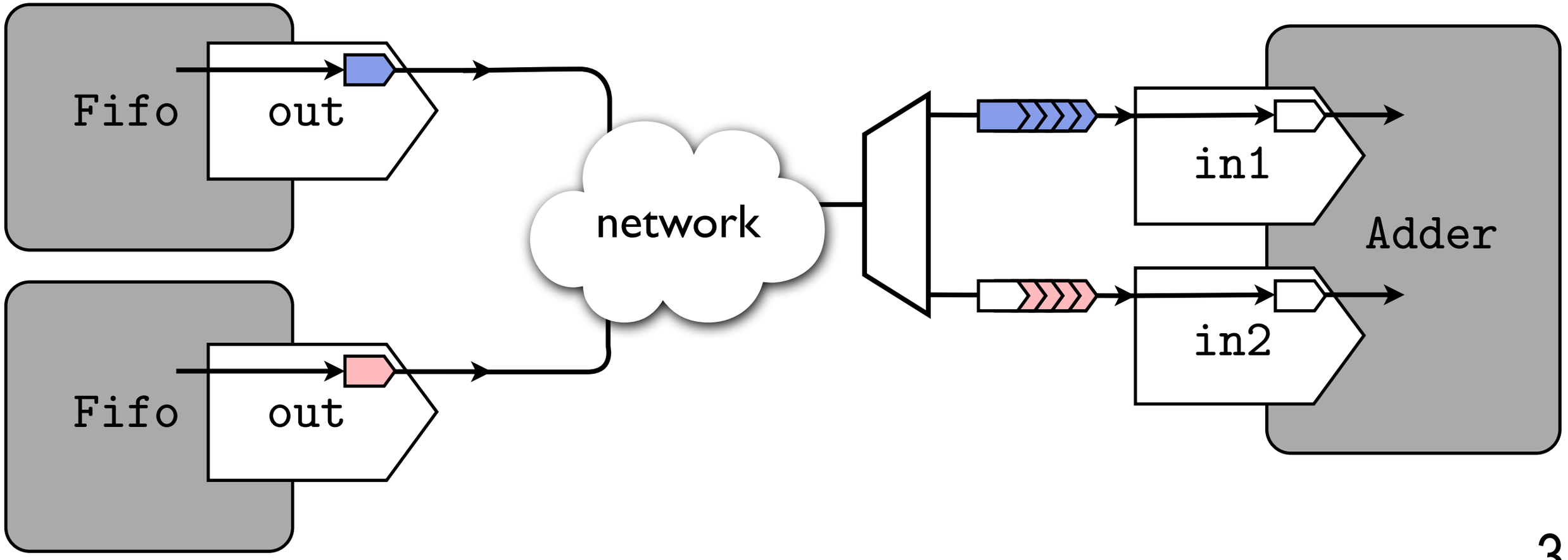
Clogging Networks

- Suppose `Adder` waits for pairs of items to become available and consumes them pairwise.
  - If items sent from the two outputs are interleaved properly, this will work.



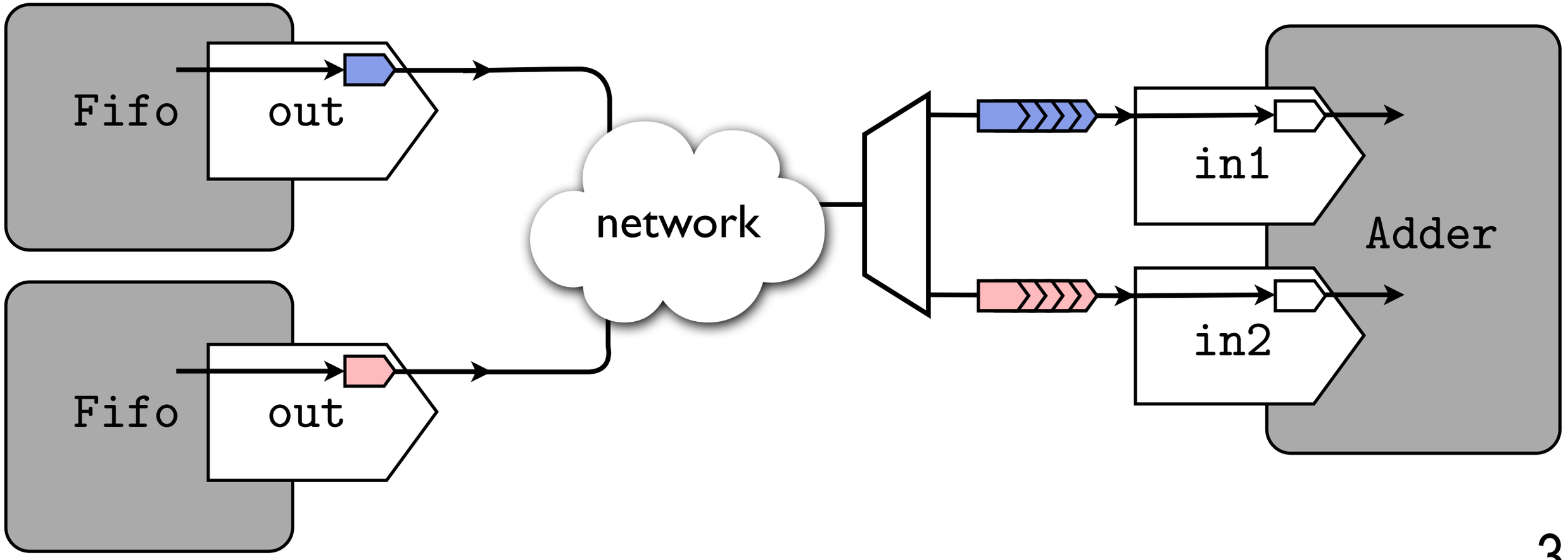
Clogging Networks

- Suppose `Adder` waits for pairs of items to become available and consumes them pairwise.
  - If items sent from the two outputs are interleaved properly, this will work.



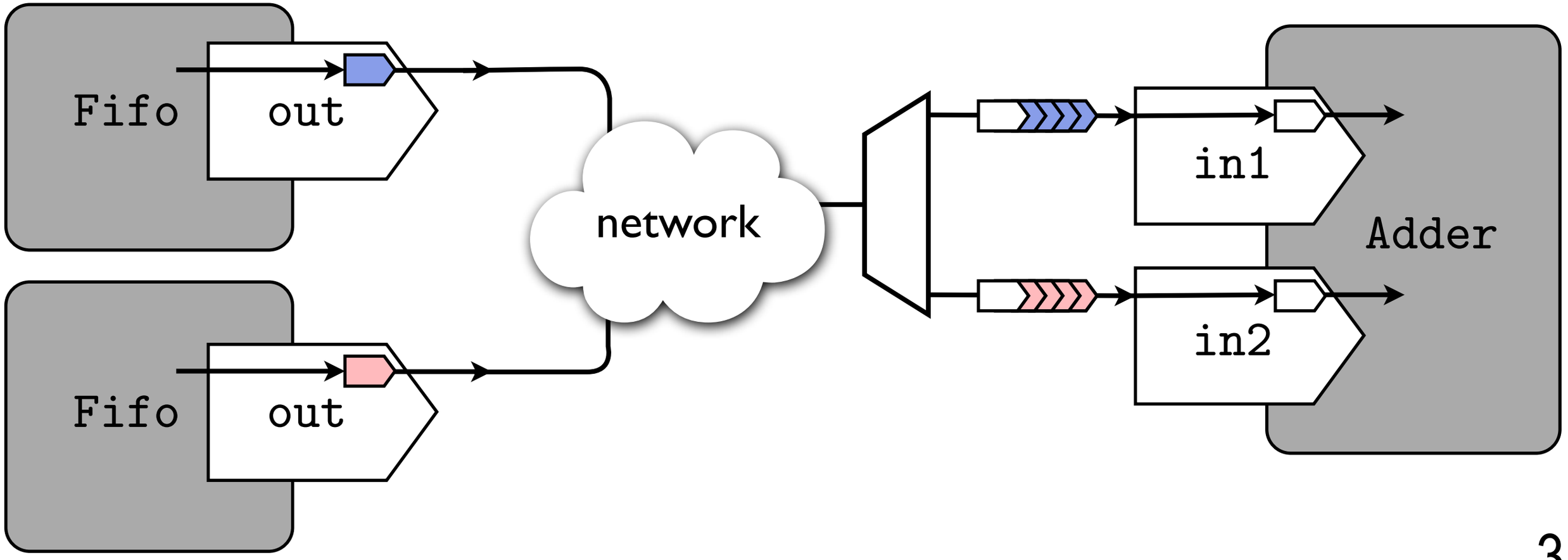
Clogging Networks

- Suppose `Adder` waits for pairs of items to become available and consumes them pairwise.
  - If items sent from the two outputs are interleaved properly, this will work.



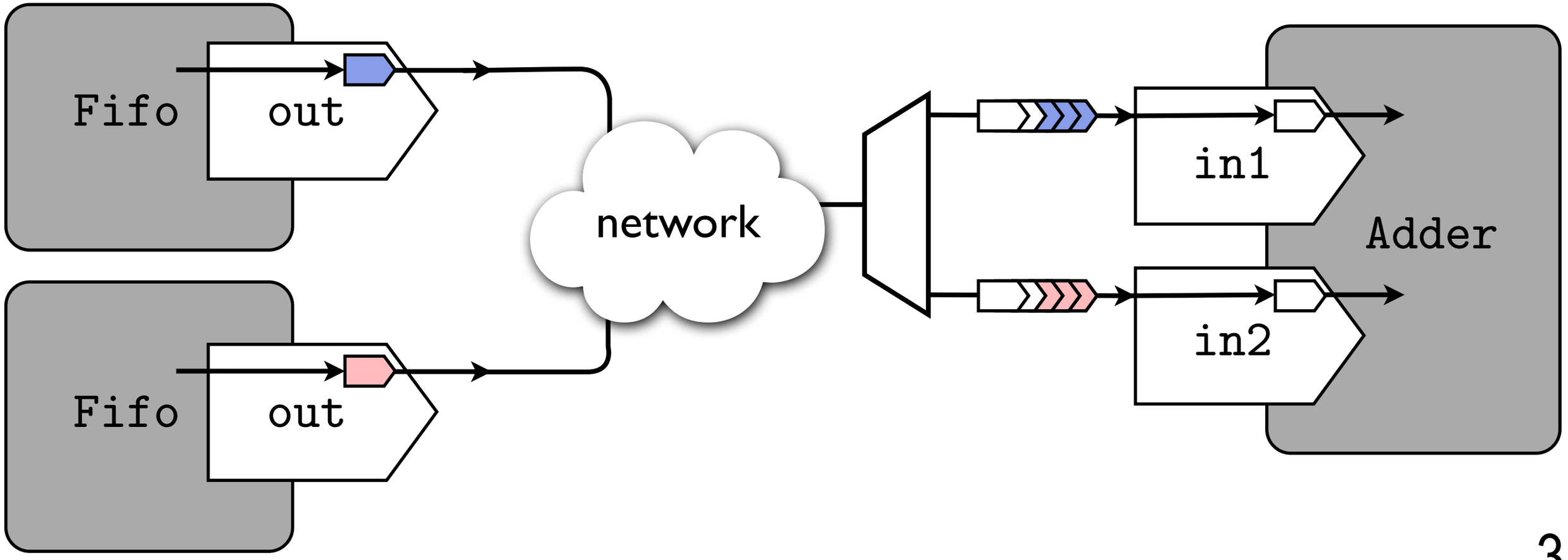
Clogging Networks

- Suppose `Adder` waits for pairs of items to become available and consumes them pairwise.
  - If items sent from the two outputs are interleaved properly, this will work.



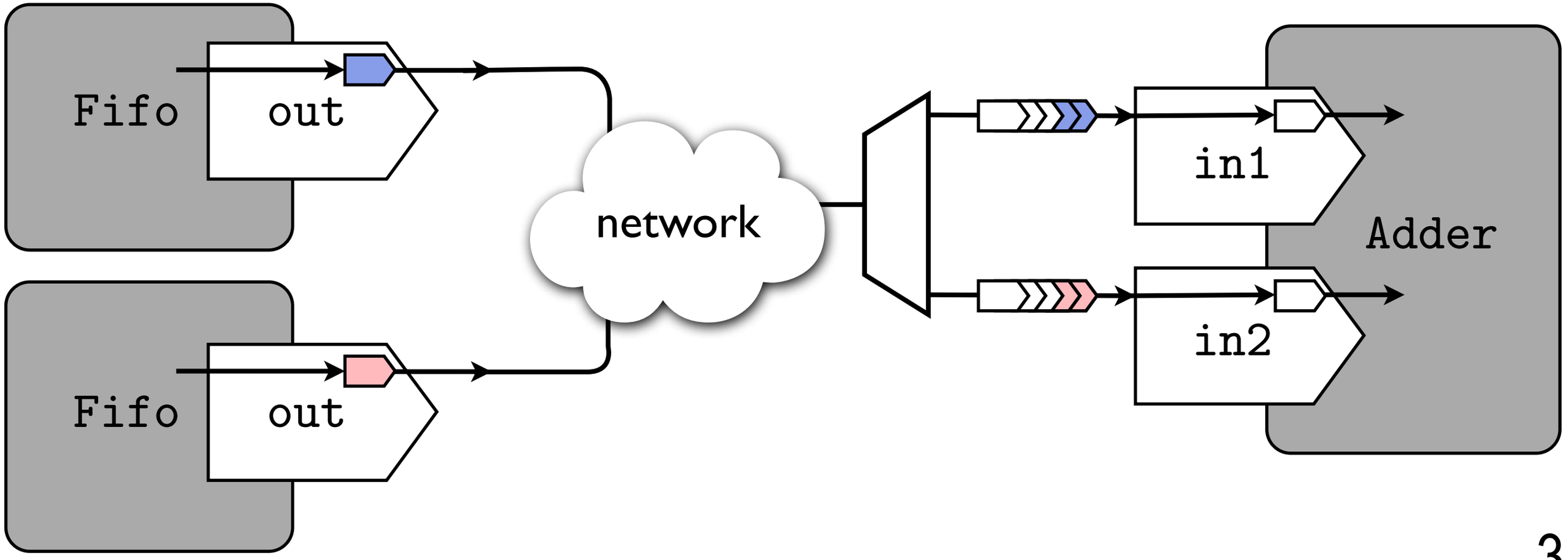
Clogging Networks

- Suppose `Adder` waits for pairs of items to become available and consumes them pairwise.
  - If items sent from the two outputs are interleaved properly, this will work.



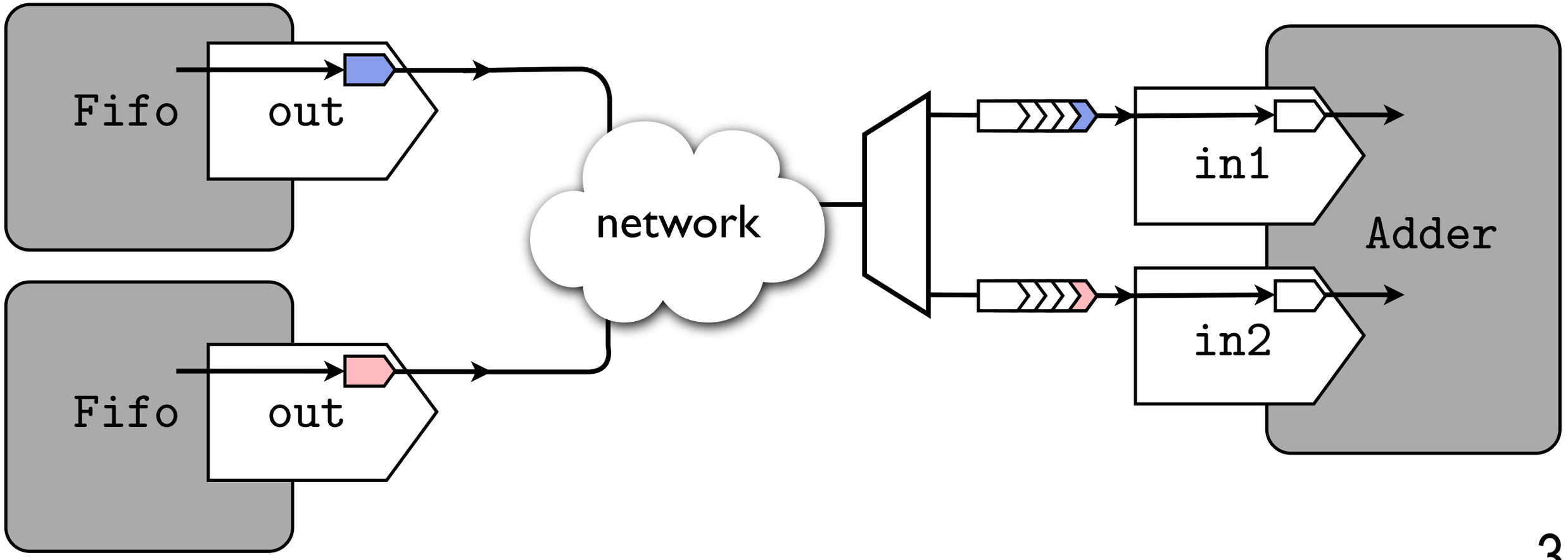
Clogging Networks

- Suppose `Adder` waits for pairs of items to become available and consumes them pairwise.
  - If items sent from the two outputs are interleaved properly, this will work.



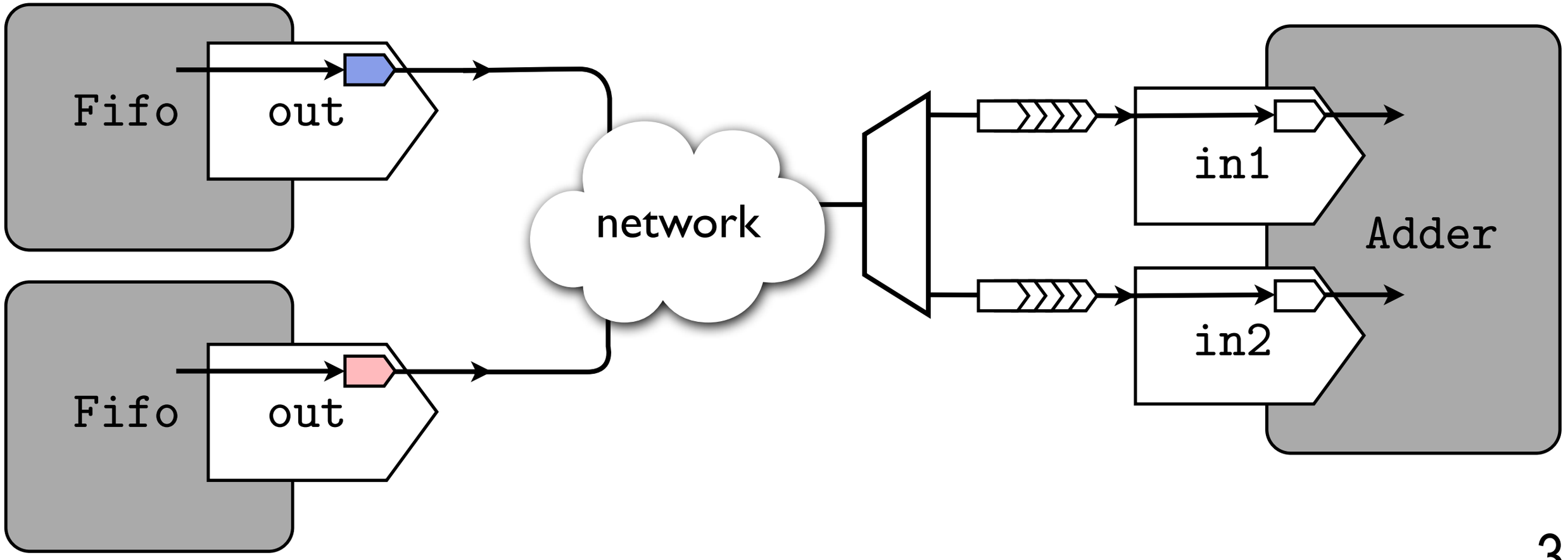
Clogging Networks

- Suppose `Adder` waits for pairs of items to become available and consumes them pairwise.
  - If items sent from the two outputs are interleaved properly, this will work.



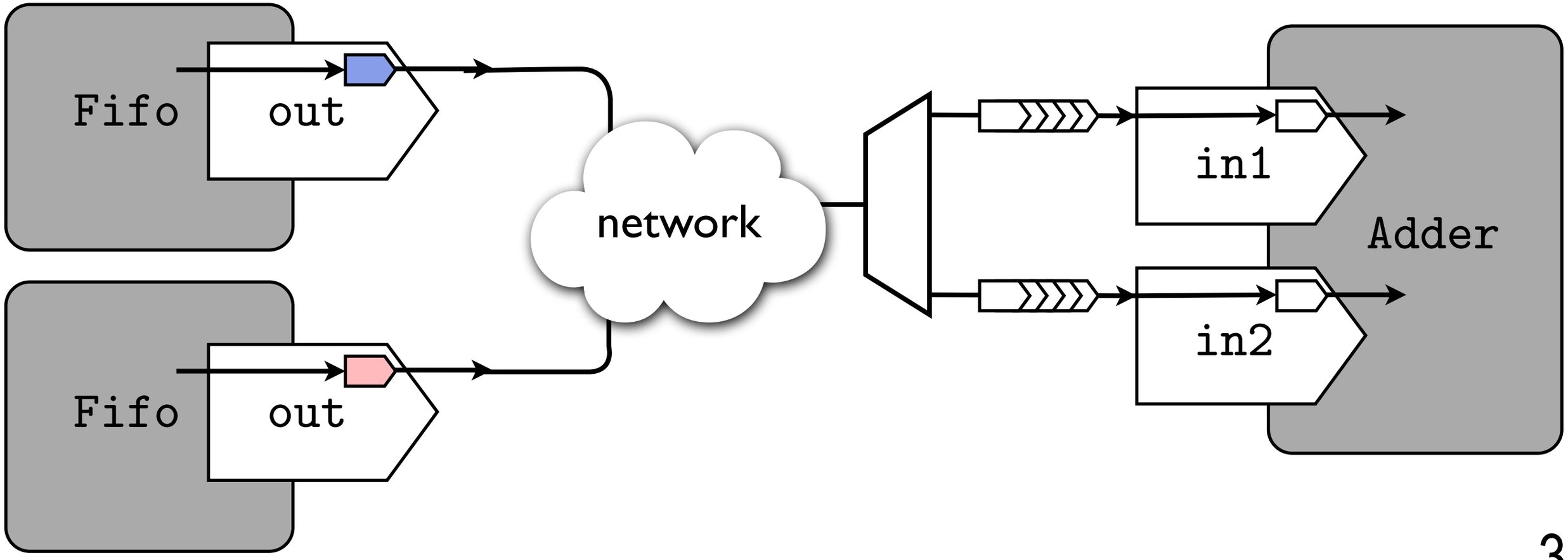
Clogging Networks

- Suppose `Adder` waits for pairs of items to become available and consumes them pairwise.
  - If items sent from the two outputs are interleaved properly, this will work.



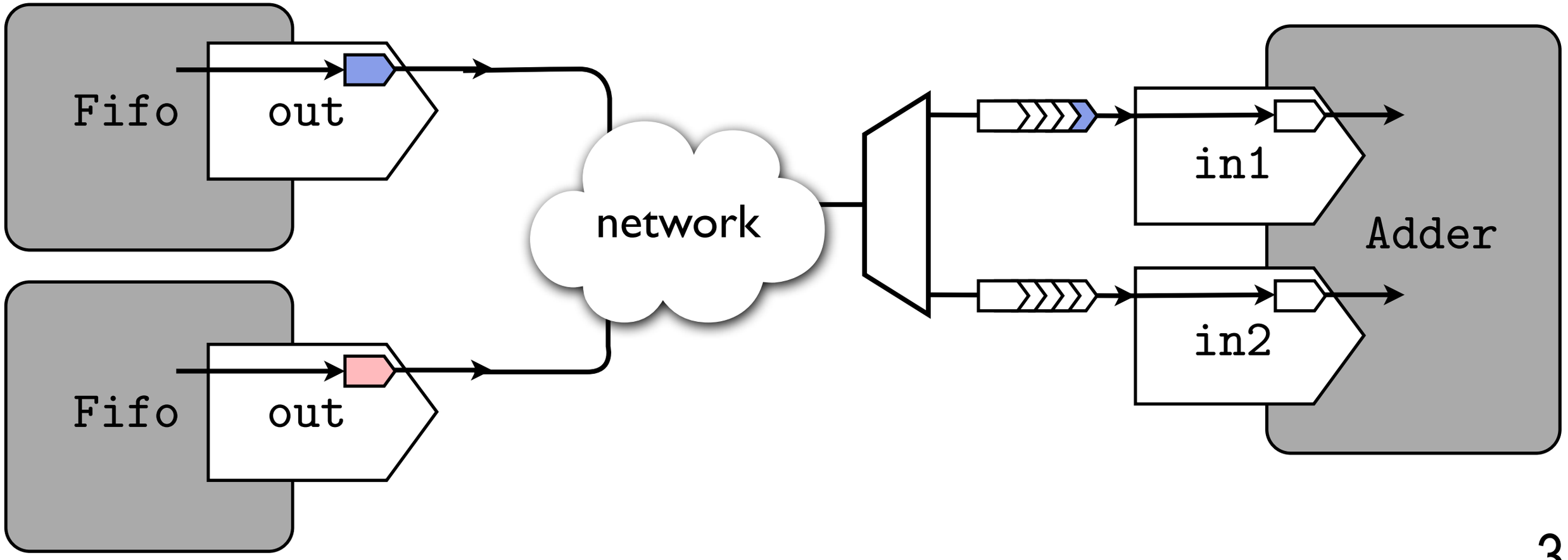
Clogging the Network

- Delays through the network are unpredictable
  - What if all packets from one source arrive before any from the other?
  - Packets “back up into” the shared portion of the network
    - Usually leads to deadlock. Dally calls this “high level deadlock,” throws his hands up in the air (sec 14.1.5).



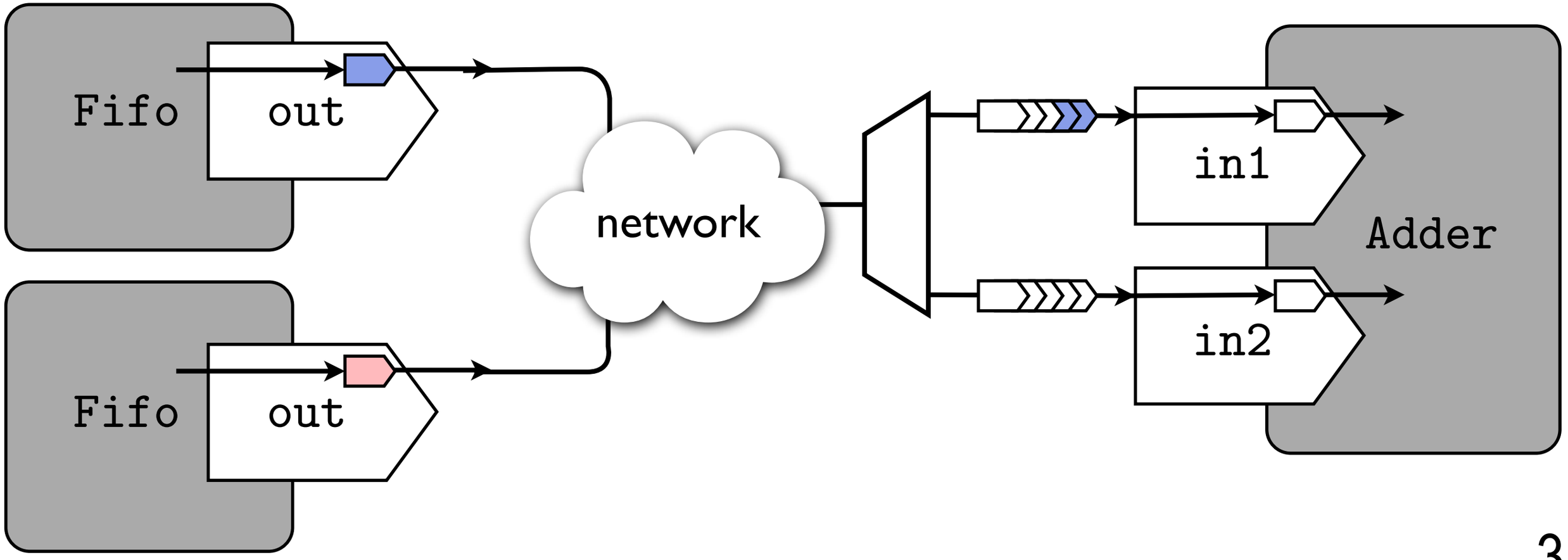
Clogging the Network

- Delays through the network are unpredictable
  - What if all packets from one source arrive before any from the other?
  - Packets “back up into” the shared portion of the network
    - Usually leads to deadlock. Dally calls this “high level deadlock,” throws his hands up in the air (sec 14.1.5).



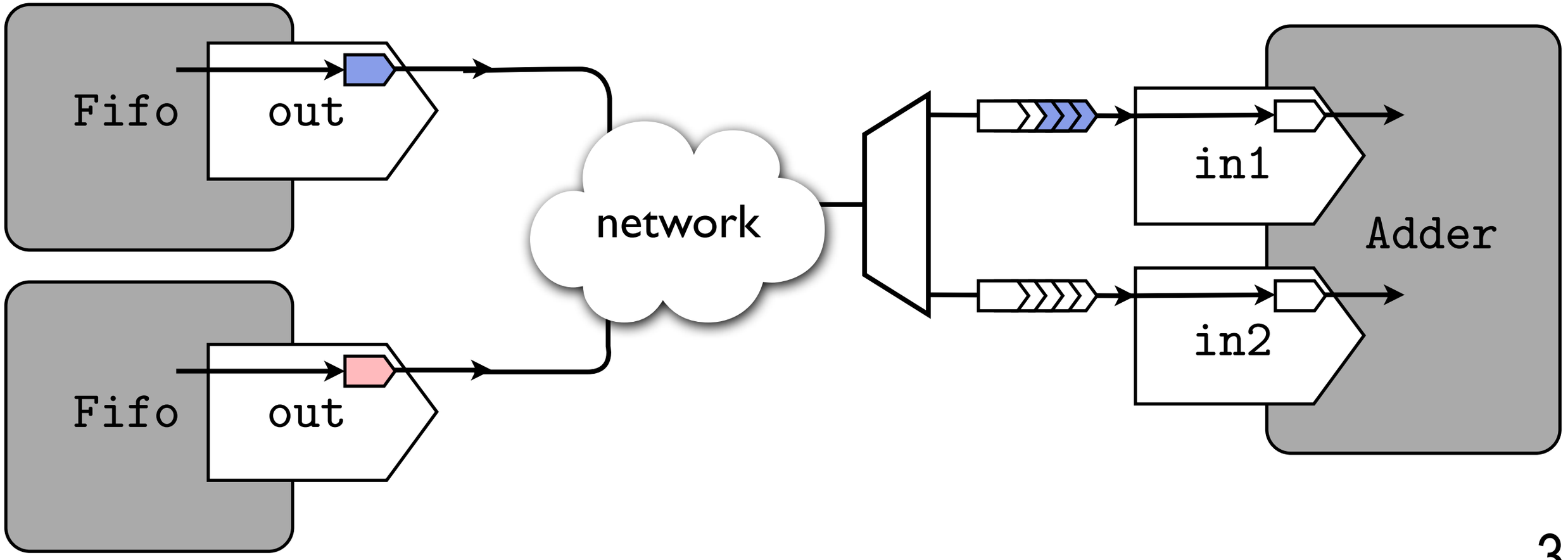
Clogging the Network

- Delays through the network are unpredictable
  - What if all packets from one source arrive before any from the other?
  - Packets “back up into” the shared portion of the network
    - Usually leads to deadlock. Dally calls this “high level deadlock,” throws his hands up in the air (sec 14.1.5).



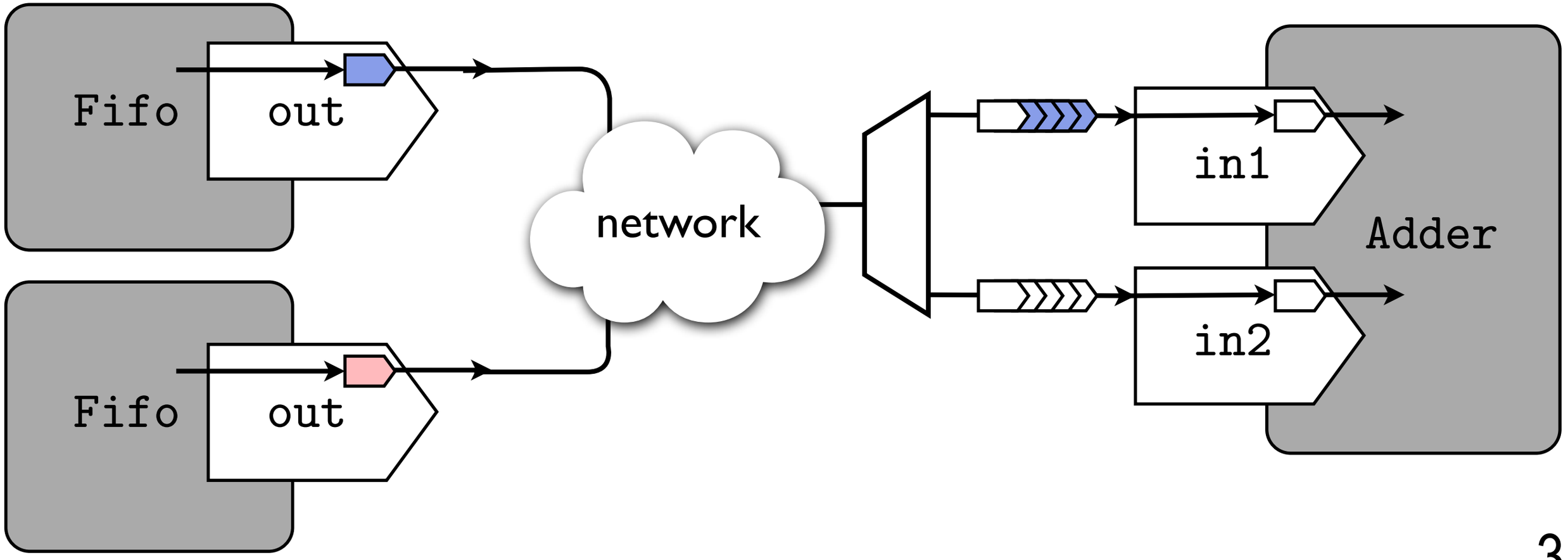
Clogging the Network

- Delays through the network are unpredictable
  - What if all packets from one source arrive before any from the other?
  - Packets “back up into” the shared portion of the network
    - Usually leads to deadlock. Dally calls this “high level deadlock,” throws his hands up in the air (sec 14.1.5).



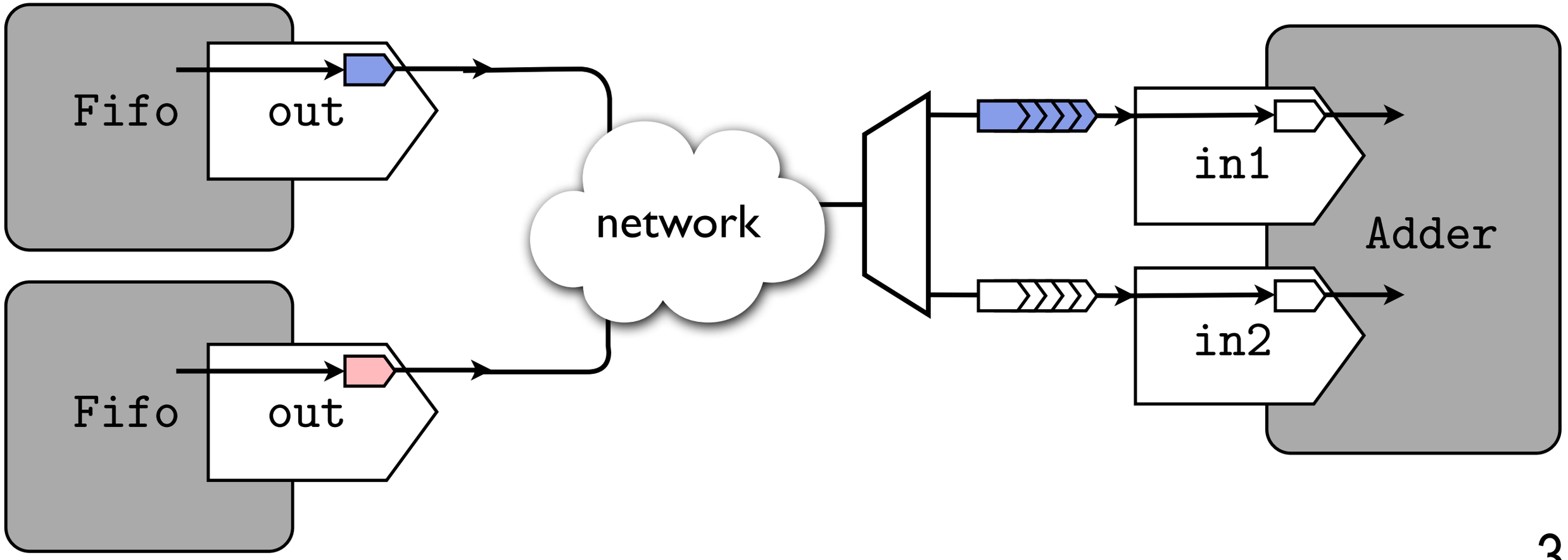
Clogging the Network

- Delays through the network are unpredictable
  - What if all packets from one source arrive before any from the other?
  - Packets “back up into” the shared portion of the network
    - Usually leads to deadlock. Dally calls this “high level deadlock,” throws his hands up in the air (sec 14.1.5).



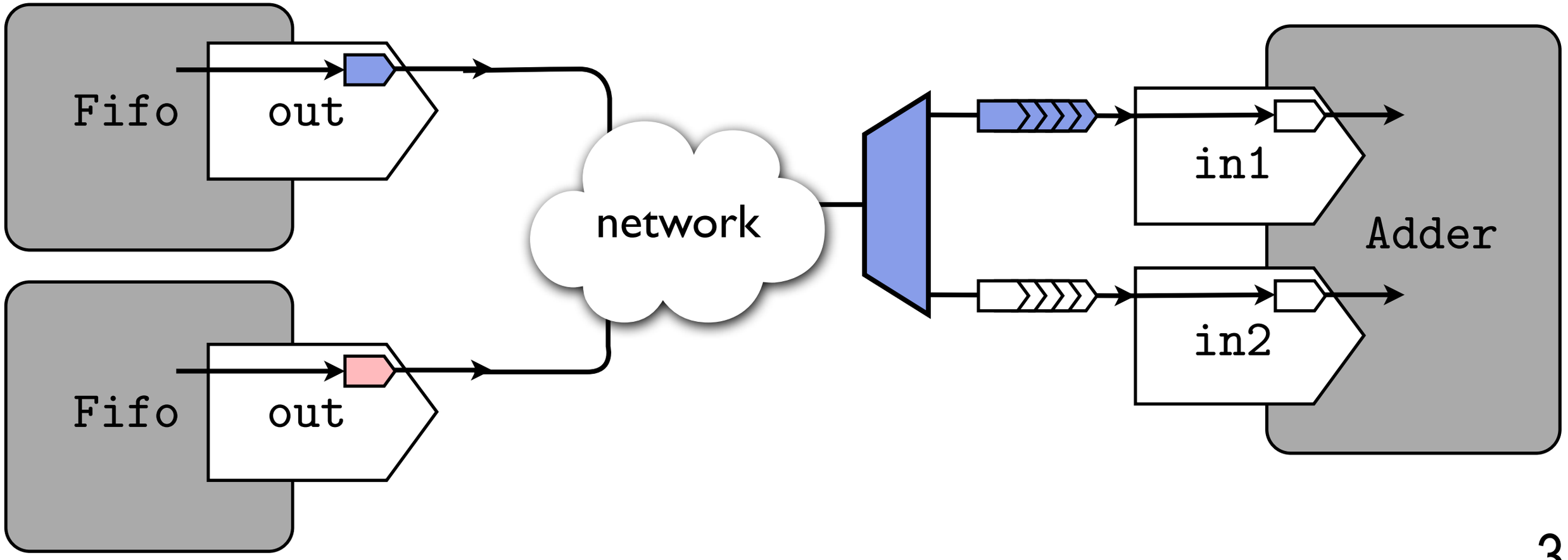
Clogging the Network

- Delays through the network are unpredictable
  - What if all packets from one source arrive before any from the other?
  - Packets “back up into” the shared portion of the network
    - Usually leads to deadlock. Dally calls this “high level deadlock,” throws his hands up in the air (sec 14.1.5).



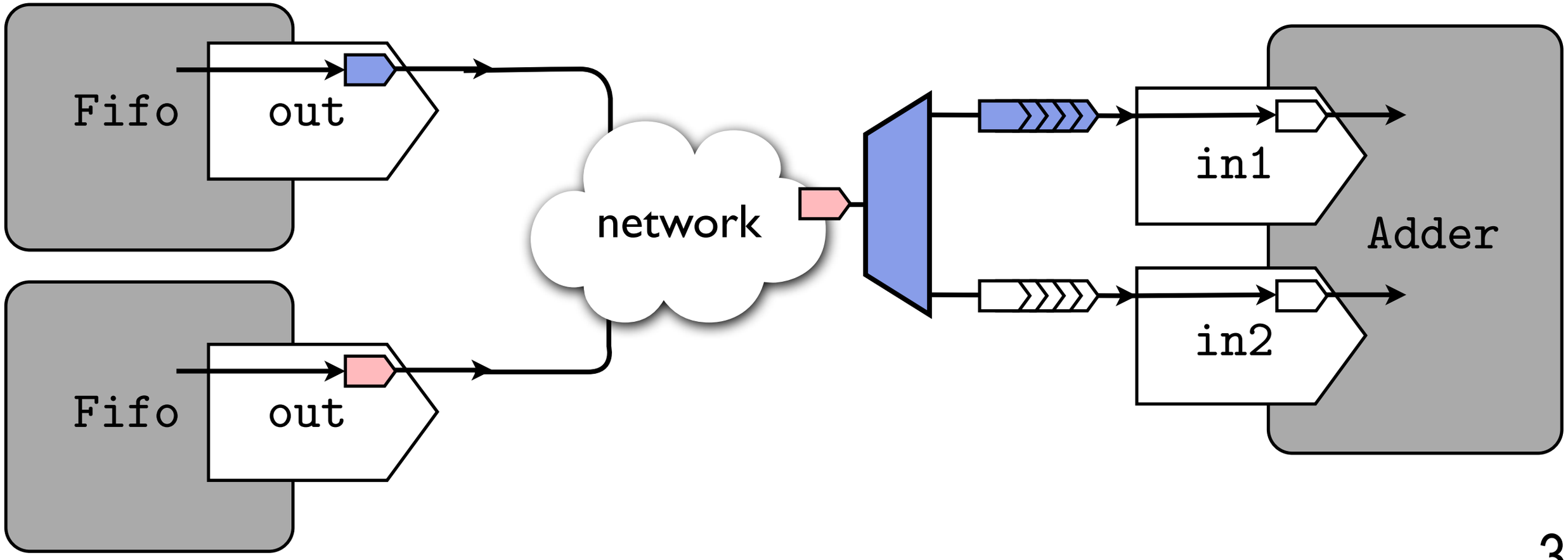
Clogging the Network

- Delays through the network are unpredictable
  - What if all packets from one source arrive before any from the other?
  - Packets “back up into” the shared portion of the network
    - Usually leads to deadlock. Dally calls this “high level deadlock,” throws his hands up in the air (sec 14.1.5).



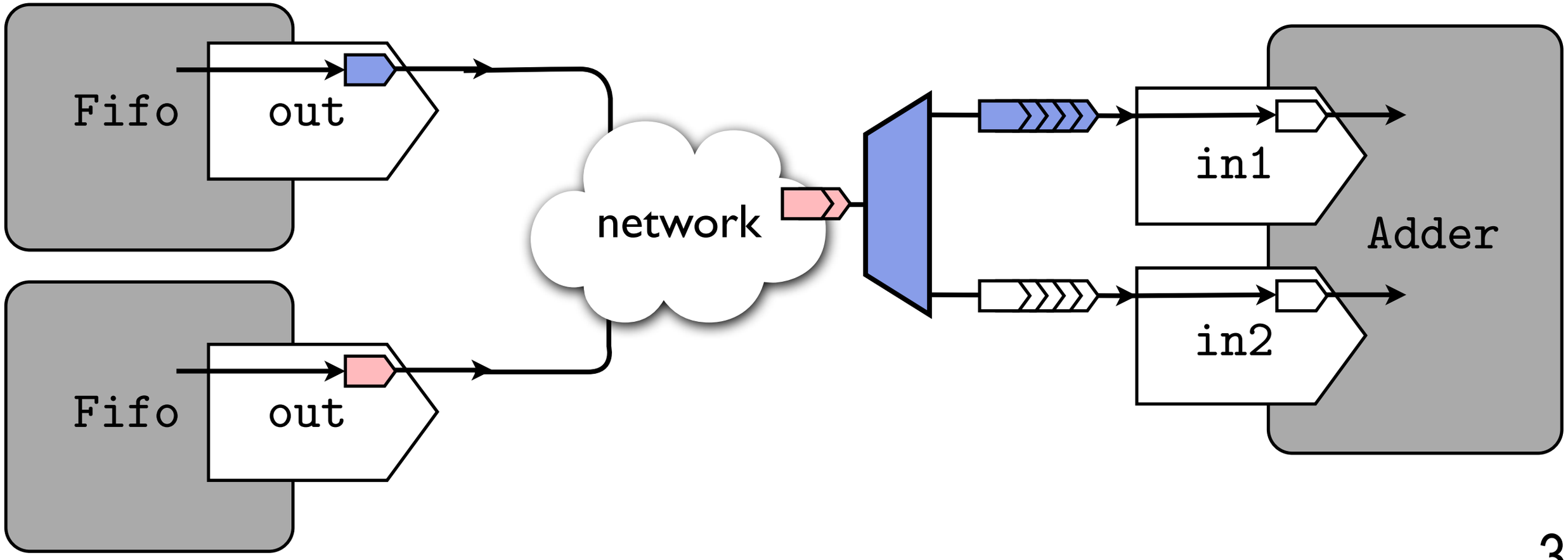
Clogging the Network

- Delays through the network are unpredictable
  - What if all packets from one source arrive before any from the other?
  - Packets “back up into” the shared portion of the network
    - Usually leads to deadlock. Dally calls this “high level deadlock,” throws his hands up in the air (sec 14.1.5).



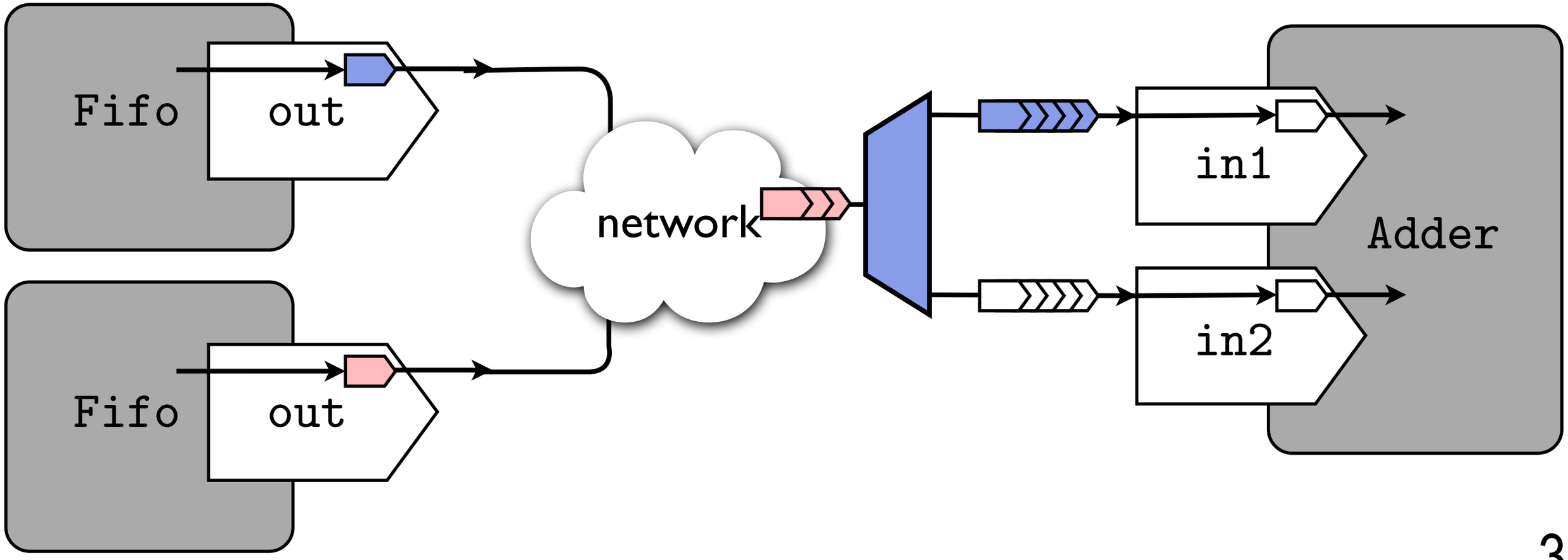
Clogging the Network

- Delays through the network are unpredictable
  - What if all packets from one source arrive before any from the other?
  - Packets “back up into” the shared portion of the network
    - Usually leads to deadlock. Dally calls this “high level deadlock,” throws his hands up in the air (sec 14.1.5).



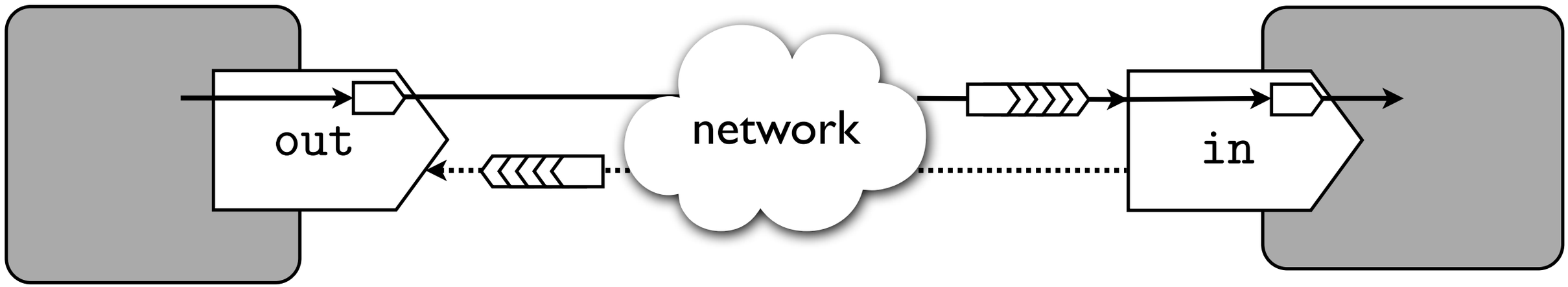
Clogging the Network

- Delays through the network are unpredictable
  - What if all packets from one source arrive before any from the other?
  - Packets “back up into” the shared portion of the network
    - Usually leads to deadlock. Dally calls this “high level deadlock,” throws his hands up in the air (sec 14.1.5).



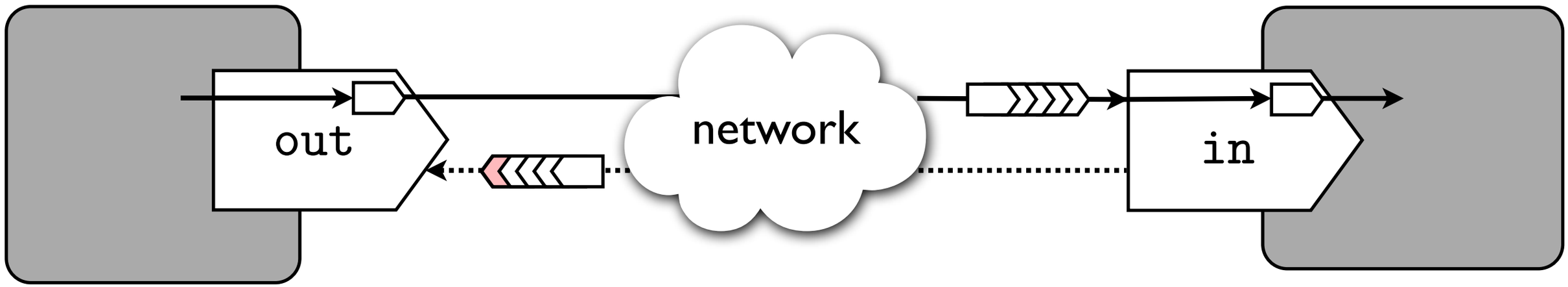
Flow Control

- In order to solve the clogging problem, we must establish *flow control*.
  - Ensure that the number of items in flight towards a given destination never exceeds the amount of buffering dedicated to that destination.



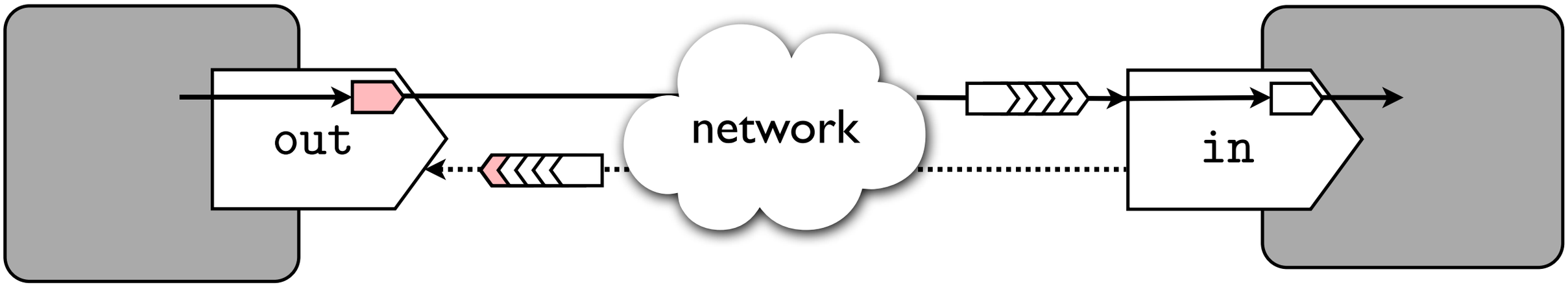
Flow Control

- In order to solve the clogging problem, we must establish *flow control*.
  - Ensure that the number of items in flight towards a given destination never exceeds the amount of buffering dedicated to that destination.



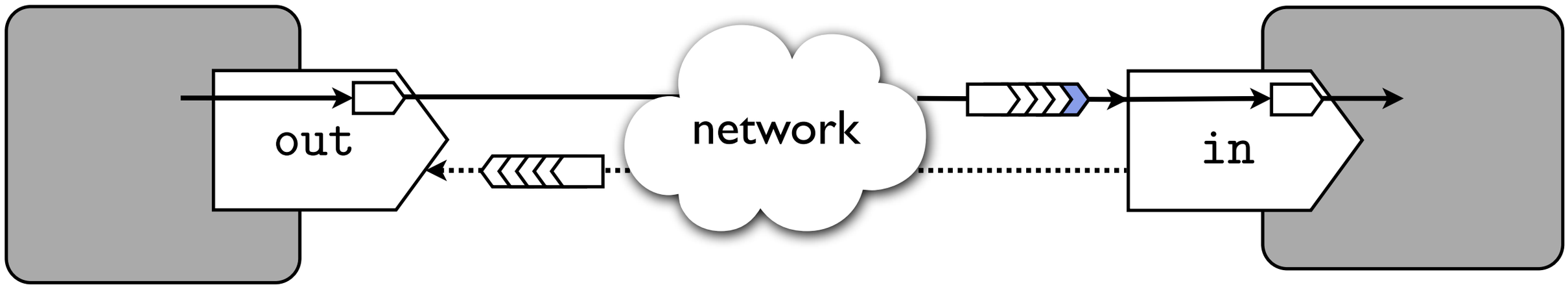
Flow Control

- In order to solve the clogging problem, we must establish *flow control*.
  - Ensure that the number of items in flight towards a given destination never exceeds the amount of buffering dedicated to that destination.



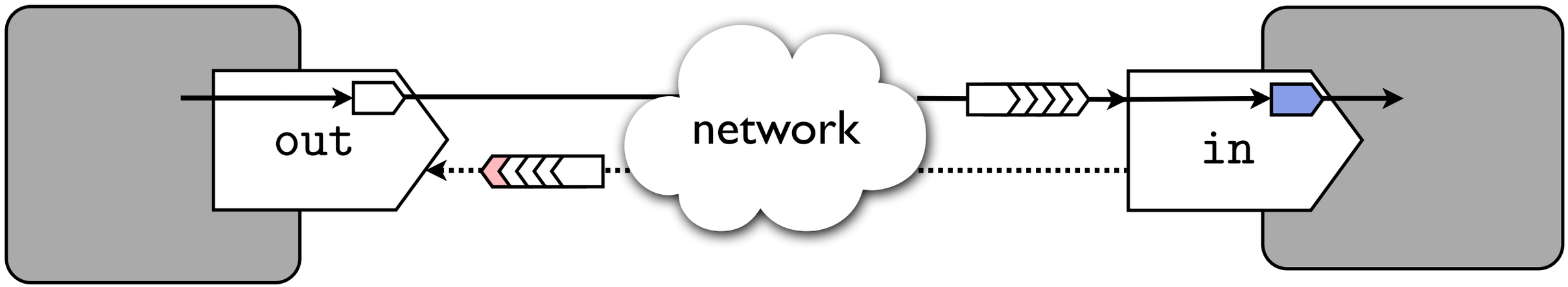
Flow Control

- In order to solve the clogging problem, we must establish *flow control*.
  - Ensure that the number of items in flight towards a given destination never exceeds the amount of buffering dedicated to that destination.



Flow Control

- In order to solve the clogging problem, we must establish *flow control*.
  - Ensure that the number of items in flight towards a given destination never exceeds the amount of buffering dedicated to that destination.



Flow Control

- In order to solve the clogging problem, we must establish *flow control*.
  - Ensure that the number of items in flight towards a given destination never exceeds the amount of buffering dedicated to that destination.

At most one packet in the network at any time