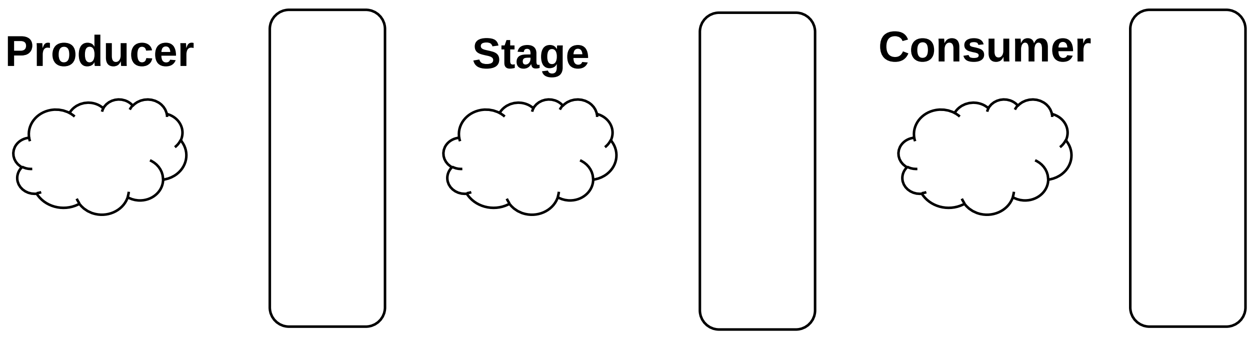
Fluid Pipelines

Jose Renau

Department of Computer Engineering,
University of California, Santa Cruz
http://masc.soe.ucsc.edu

How to Handle Stalls?

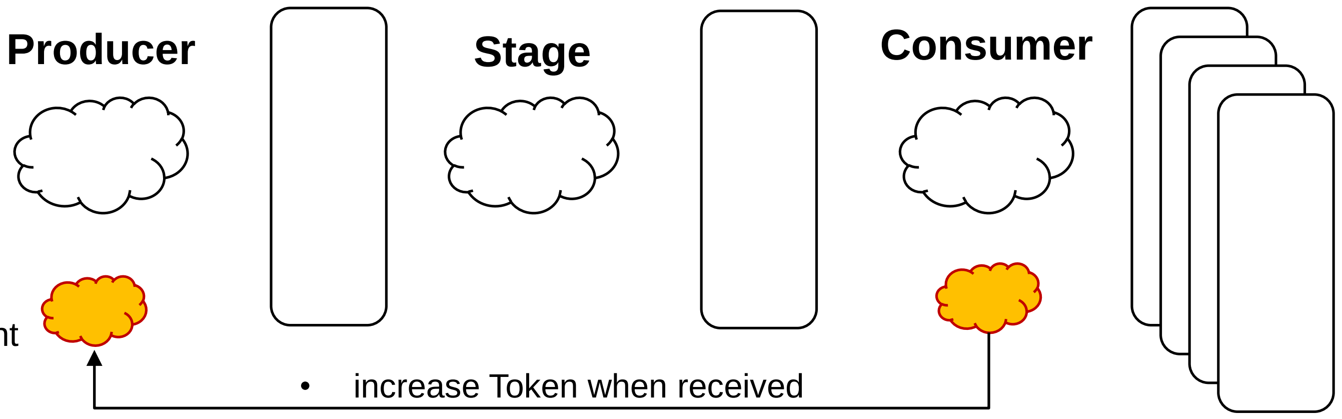
- What if the consumer is busy?



3 Typical Solutions

- Create enough buffer for worst case (impractical)
- Guarantee does not happen
  - Make the consumer faster than the producer
- Create back pressure

Token/Credit (Common Technique)



- Stall if not enough tokens
- Decrease token when sent

- increase Token when received

Back Pressure

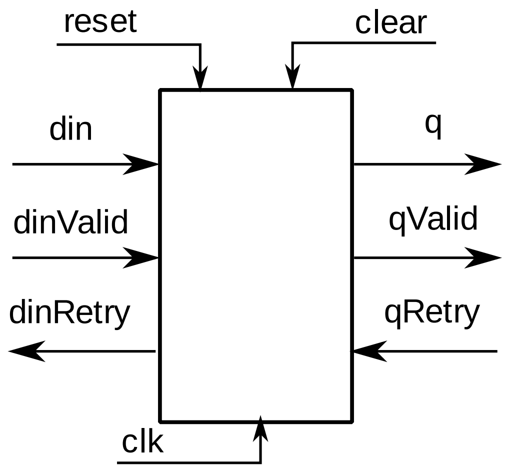
- It can not have a global wire to stall (bad timing)
- Must have some handshake between stages
  - Many options possible:
    - 4-phase protocol, 2-phase protocol….
    - Elastic

Solution: Fluid Pipelines

- Use a consistent Valid/Retry backpressure between stages
- ALSO:
  - Allows to do automatic pipeline transformations

•Fluid Pipelines: Elastic Circuitry meets Out-of-Order Execution, Rafael Trapani Possignolo, Elnaz Ebrahimi, Haven Skinner, and Jose Renau, International Conference on Computer Design (**ICCD**), June 2016.
•Fluid Pipelines: Elasticity without Throughput Penalty, Rafael Trapani Possignolo, Elnaz Ebrahimi, Haven Skinner, and Jose Renau, International Workshop on Logic and Synthesis (**IWLS**), April 2016.

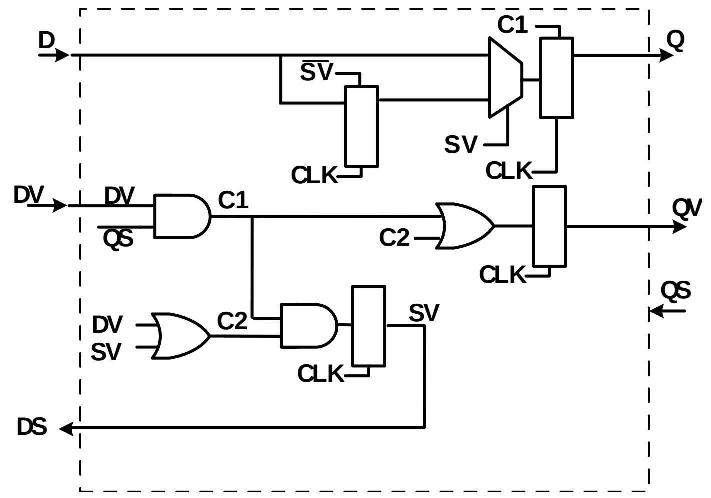
Verilog for Fluid Flop (fflop.v)



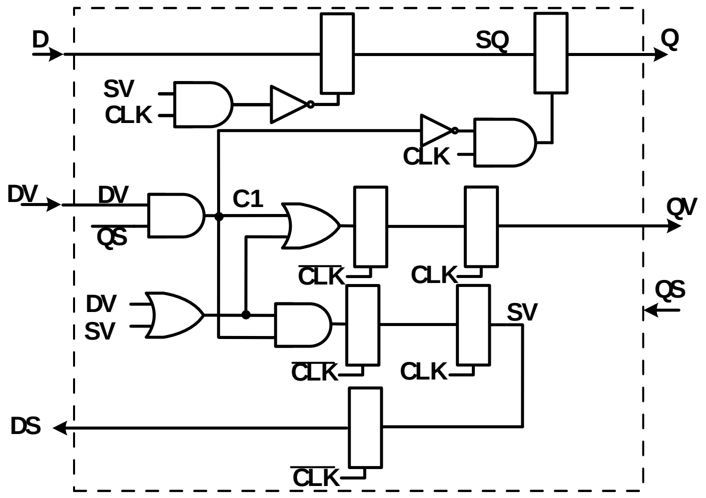
```
module fflop
  #(parameter Size=1)
    (input                        clk
     ,input                       reset
     ,input                       clear

     ,input  logic [Size-1:0]     din
     ,input  logic                dinValid
     ,output logic                dinRetry

     ,output logic [Size-1:0]     q
     ,input  logic                qRetry
     ,output logic                qValid
     );
```

Fluid Flop (aka Elastic Buffer or Relay or..)

- Traditional flop is a “1 element FIFO”
- Fluid Flop is a 2 element FIFO with latches or flops



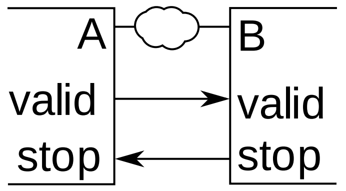
Flop based implementation



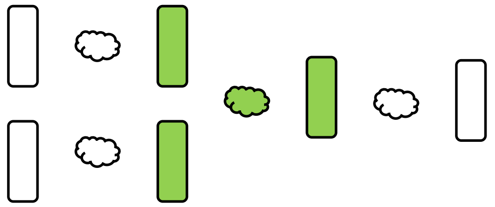
Latch based implementation

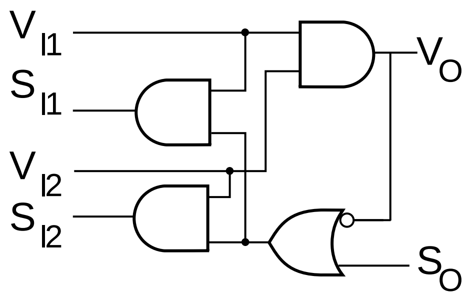
Common Fluid Connections

- Straight:
  - One Fluid Flop connects to 1 Fluid Flop (1:1)

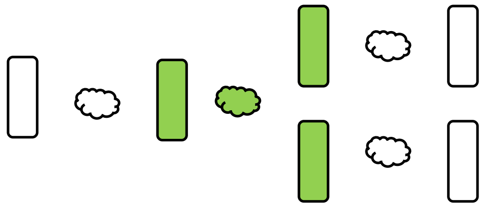


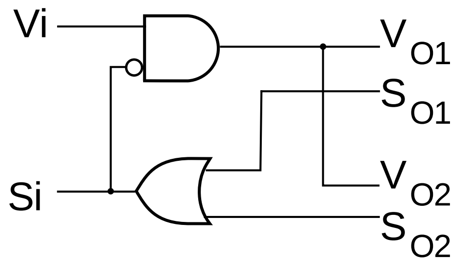
- Join:
  - Two Fluid Flops used to generate a value in one flow (2:1)





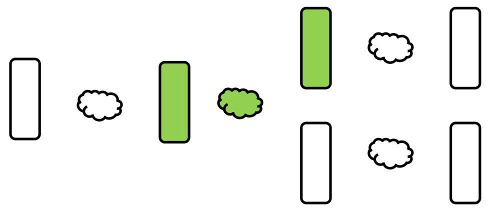
- Fork:
  - One Fluid Flop goes to 2 output flops (2:1)

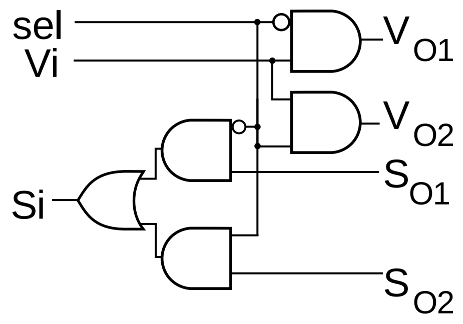




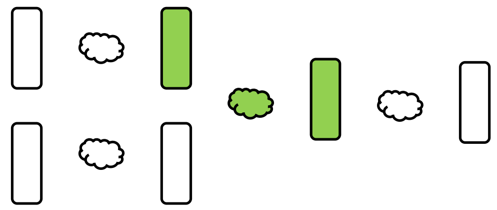
Common Fluid Connections II

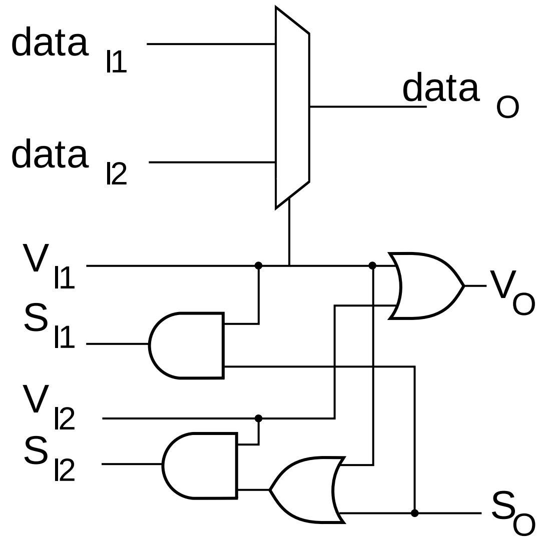
- Branch:
  - Propagate data to only one of the output fluid flops



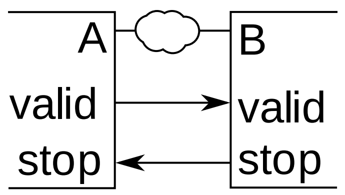


- Merge:
  - Pick data from only one of the input fluid flops

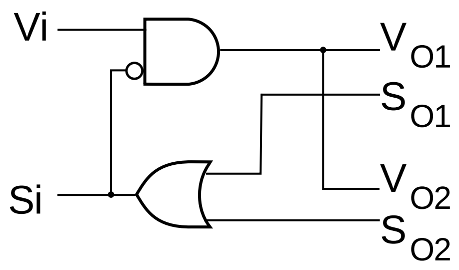




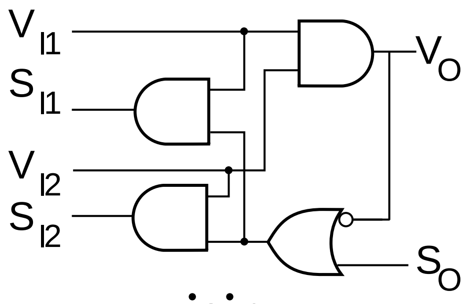
Fluid Pipeline Connections Summary



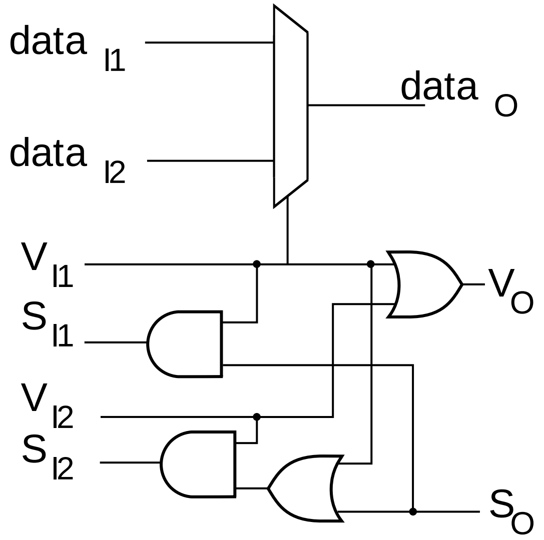
straight



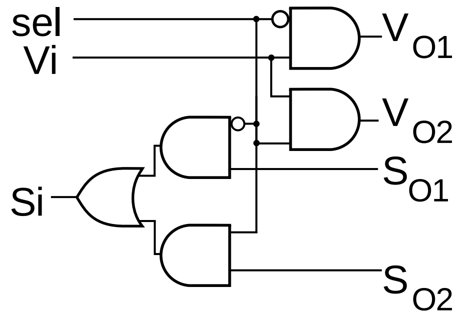
fork



join



merge



branch

Verilog Examples

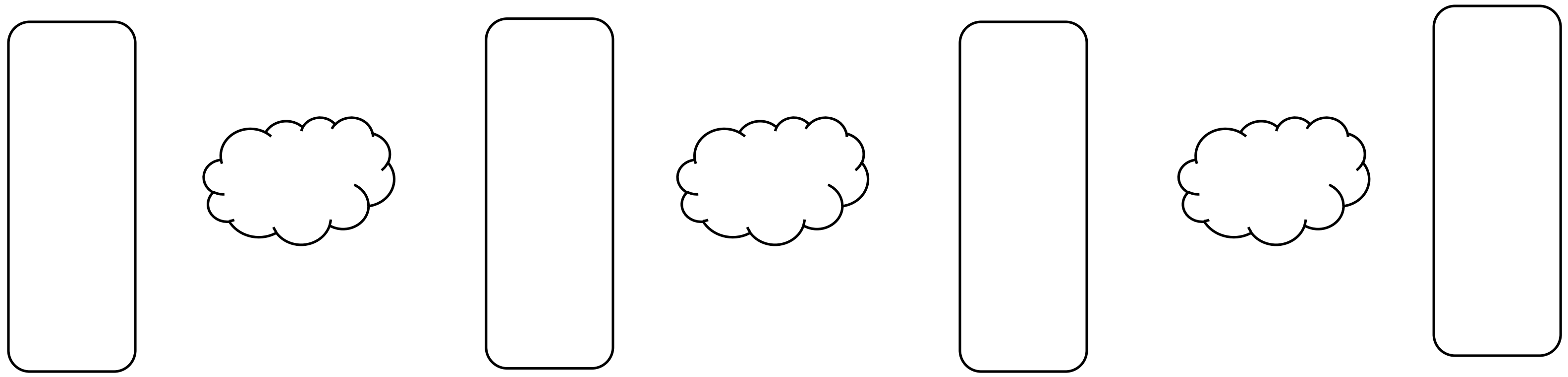
- Available at github
  - https://github.com/masc-ucsc/fluid/tree/master/examples
- straight_test
  - 4 fluid pipelines without combinational logic
  - Verilator random generates inputs and check that the results match
- join_test
  - 2 inputs with different fluid signals
  - A fluid join after 2 fluid pipeline stages
  - A single output after join
- Fork_test
- Branchmerge_test

Verilog Examples

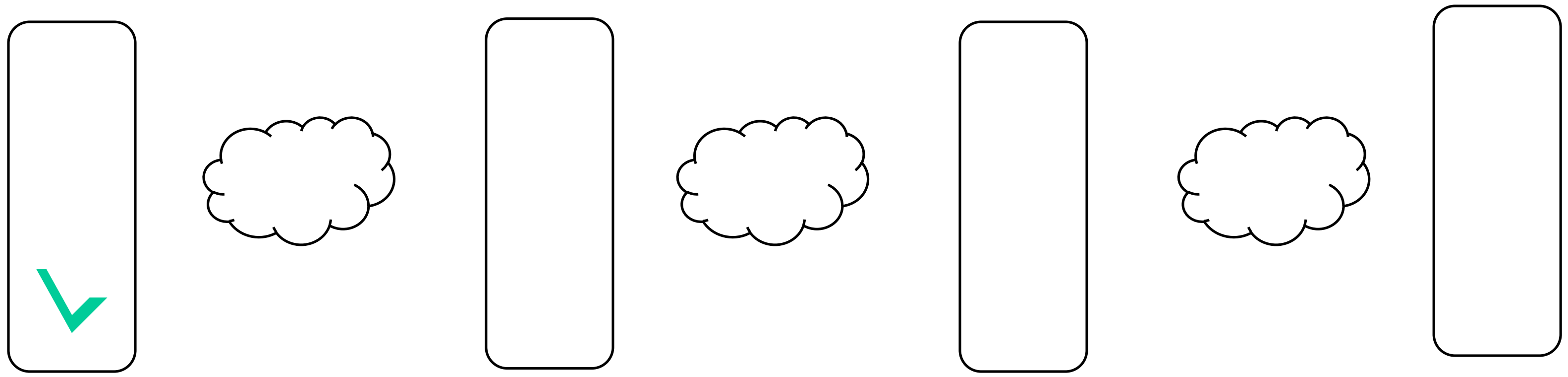
- To run examples
  - Install verilator

```
cd examples
make run1 # runs straight_test
make run2 # runs join_test

gtkwave output.vcd # see last run waveform
```

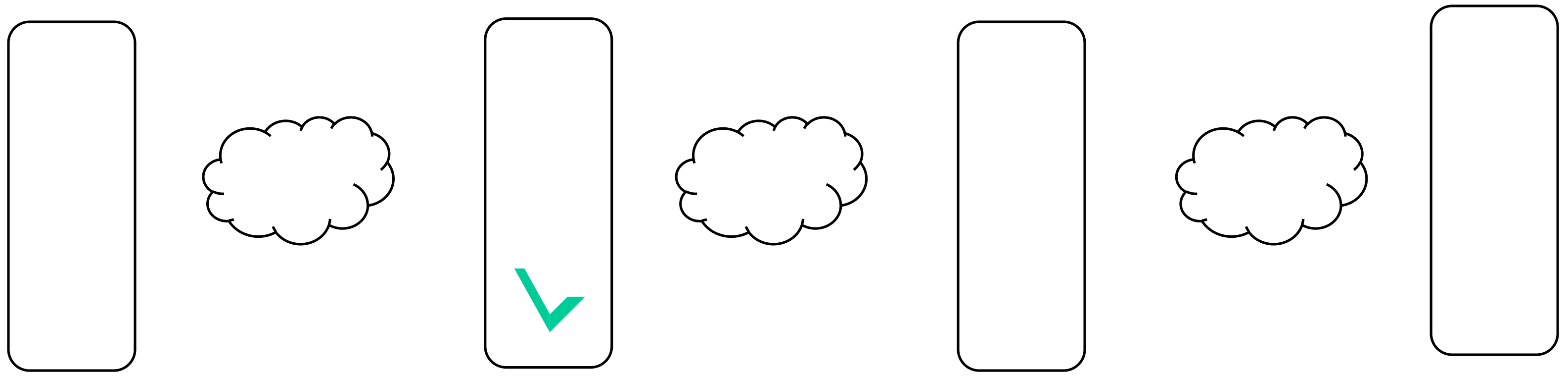
Simple Straight Example



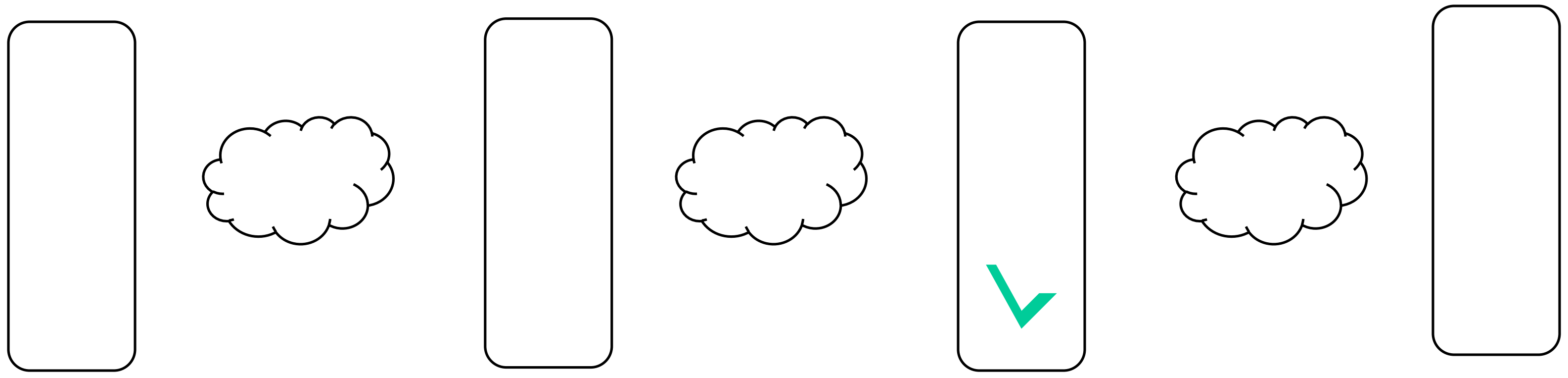
Simple Straight Example



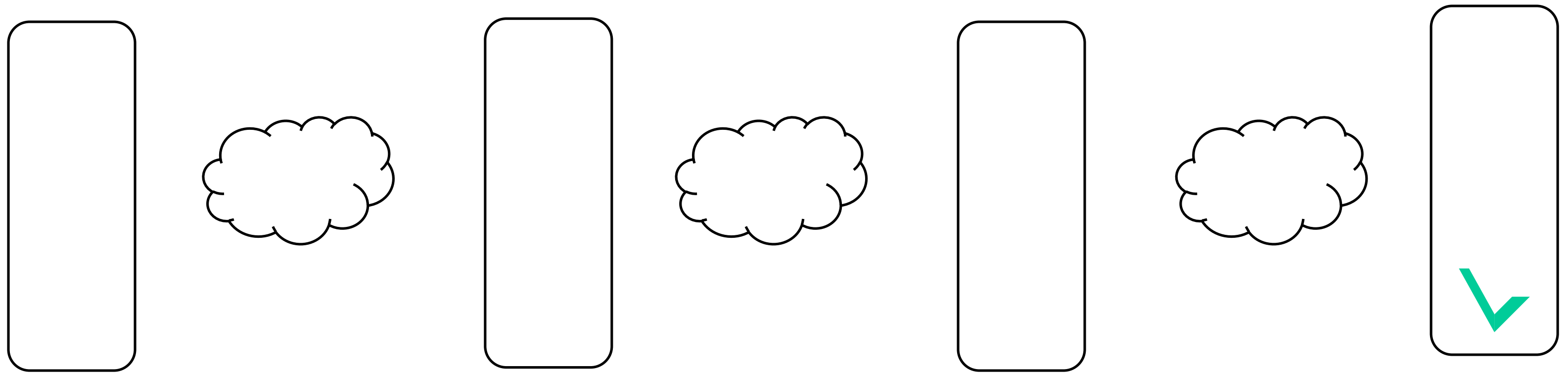
Simple Straight Example



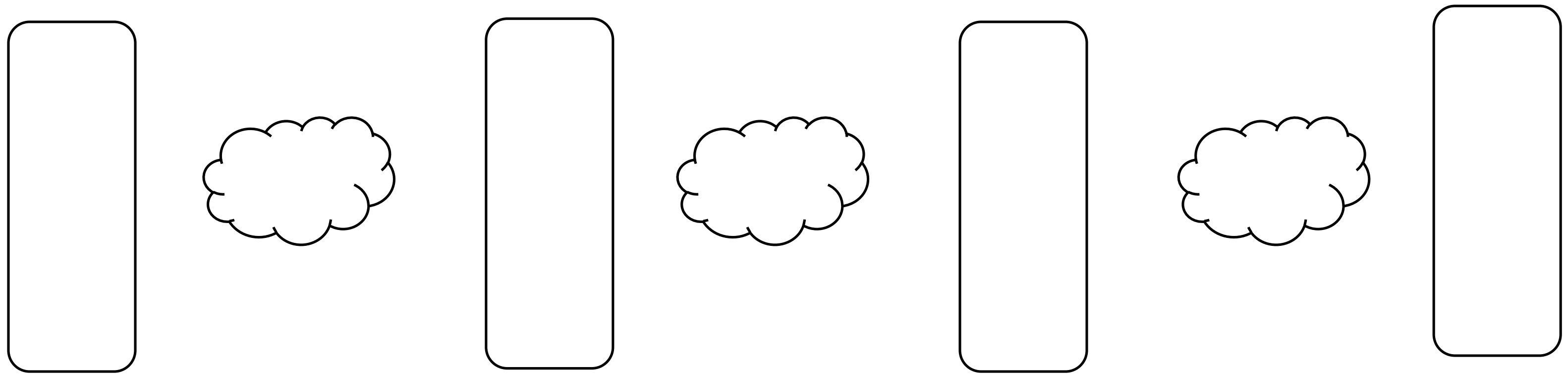
Simple Straight Example



Simple Straight Example

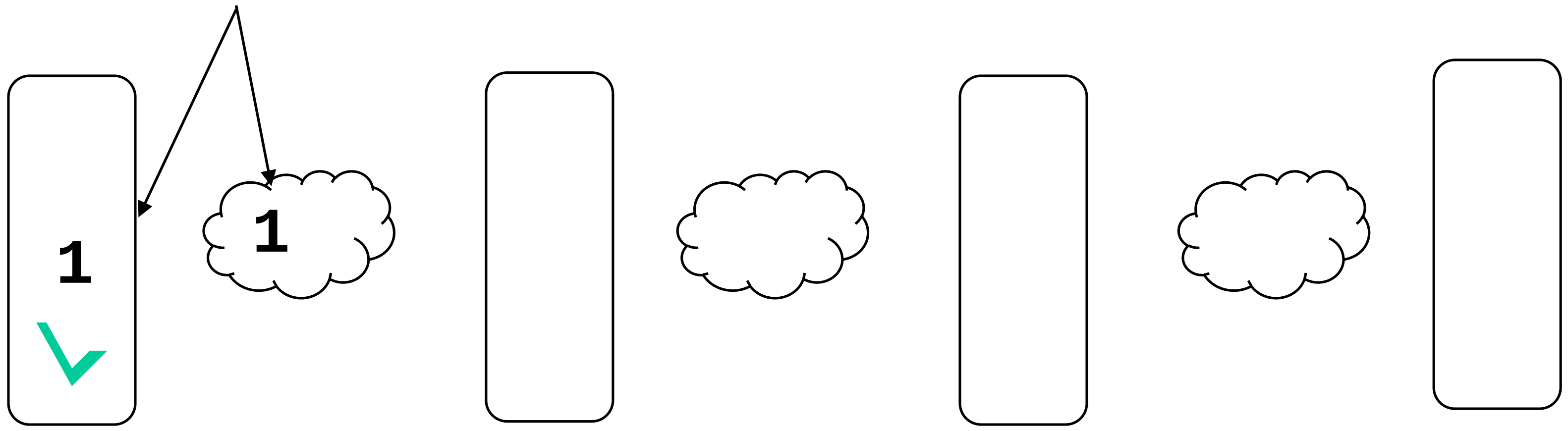


Simple Straight Example

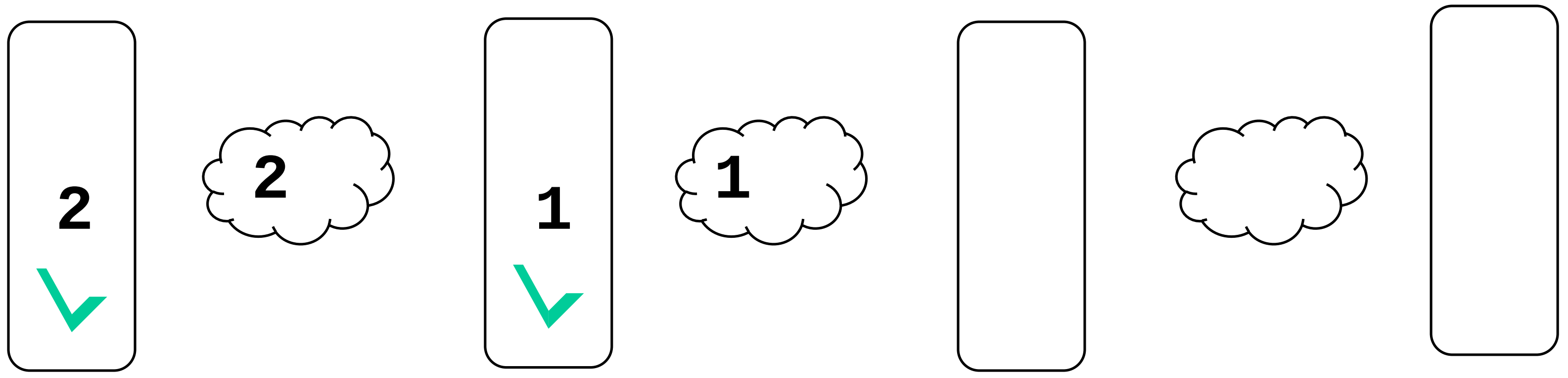


Simple Straight Example

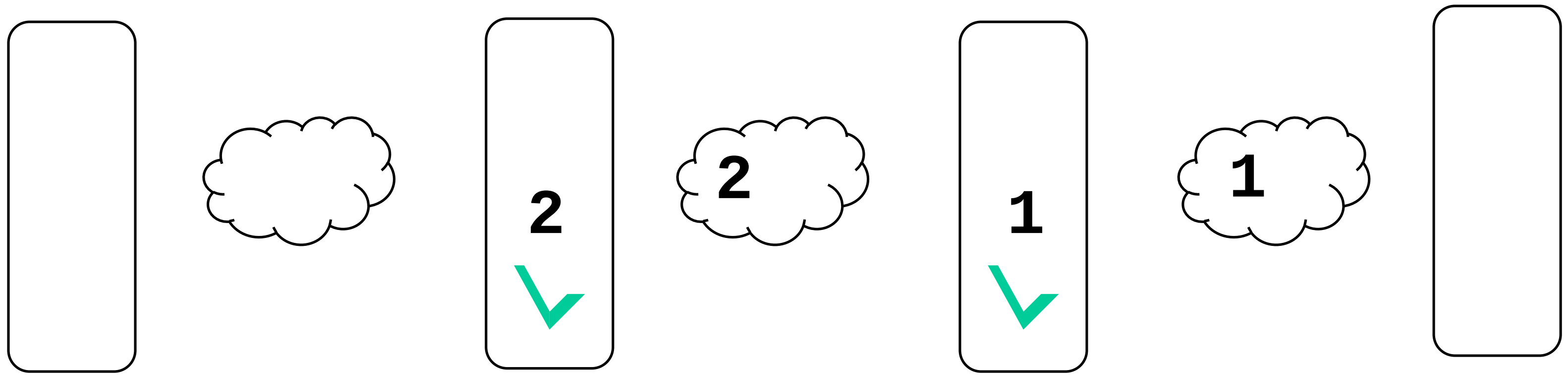
“1” is the value visible at the output of the fluid flop, and hence the combinational logic in the stage



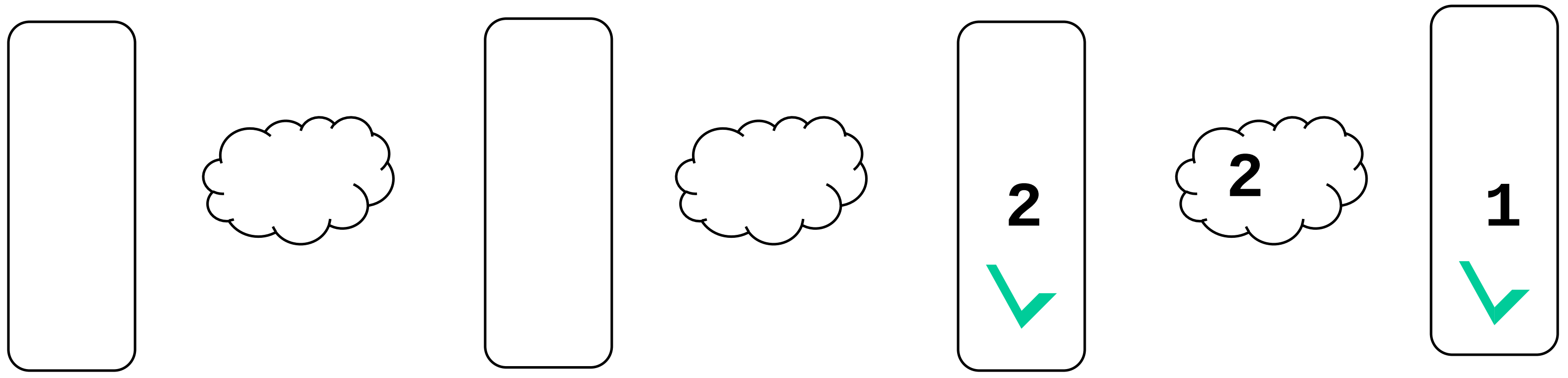
Simple Straight Example



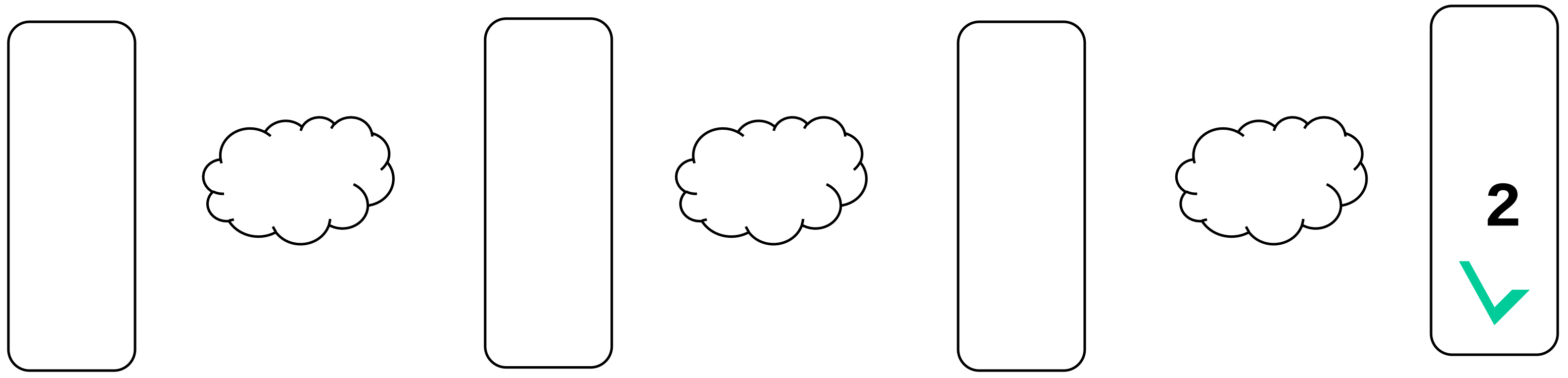
Simple Straight Example



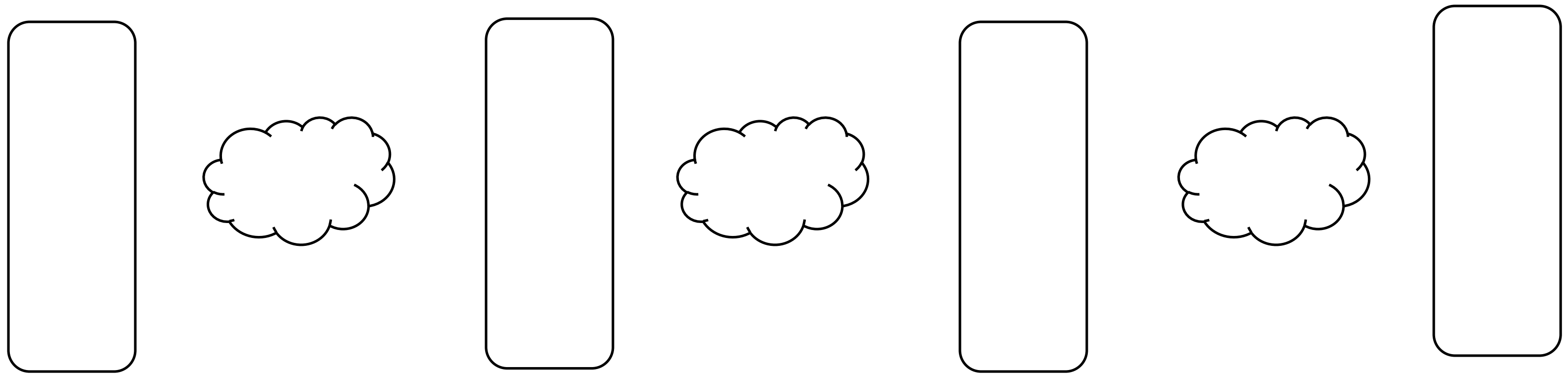
Simple Straight Example



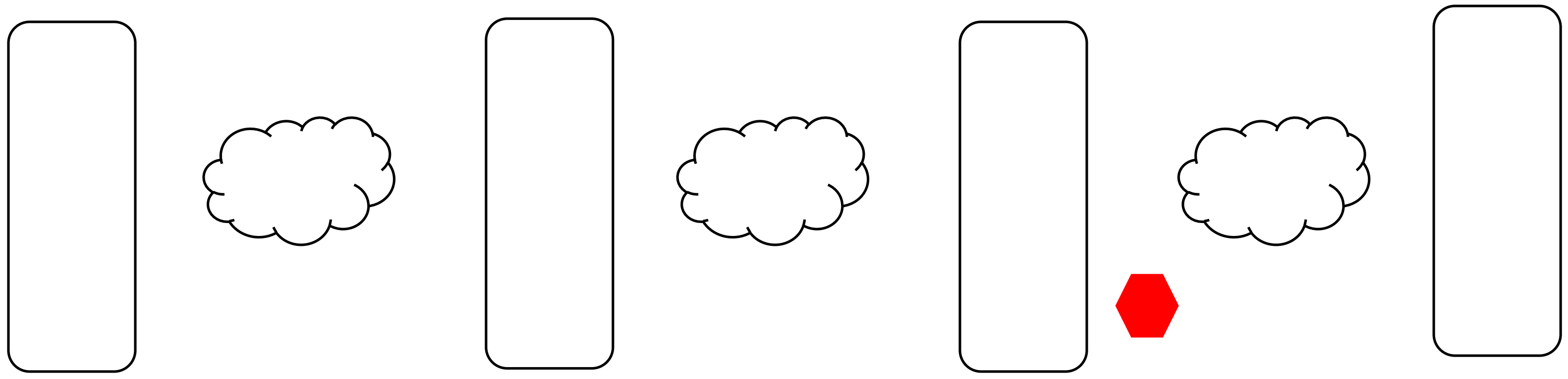
Simple Straight Example



Simple Straight Example

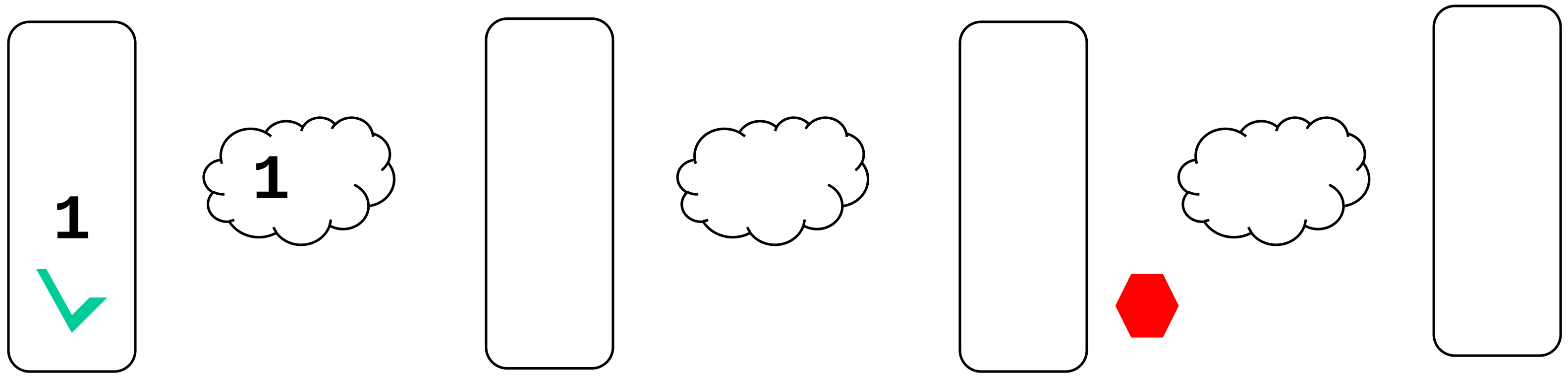


Simple Straight Example

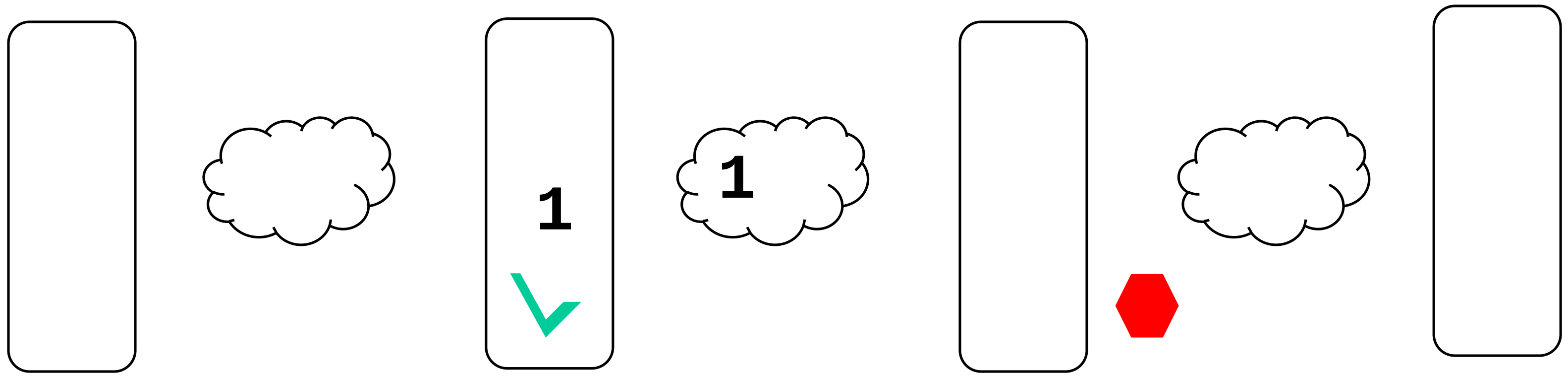


Retry/STOP asserted for some reason

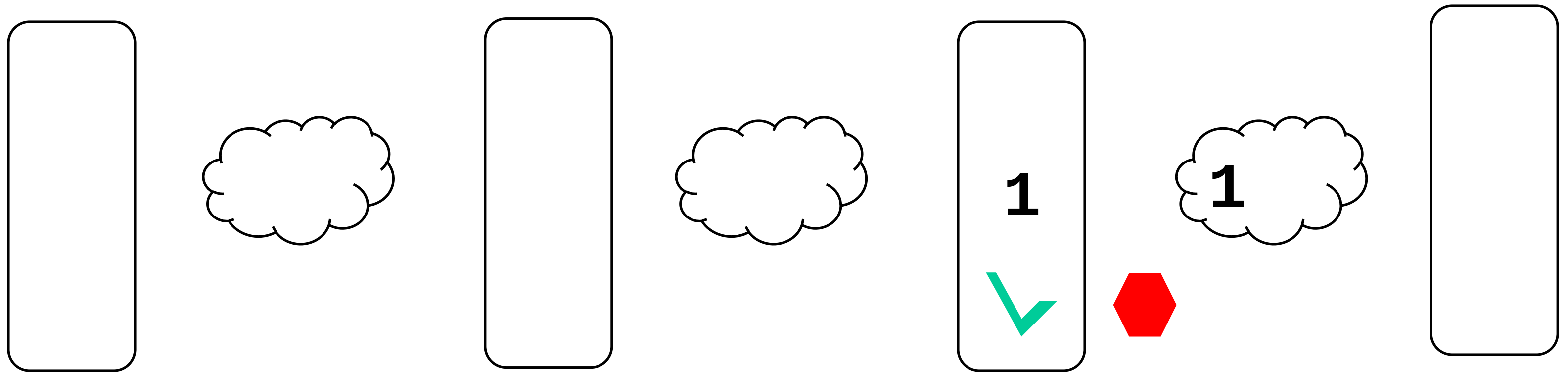
Simple Straight Example



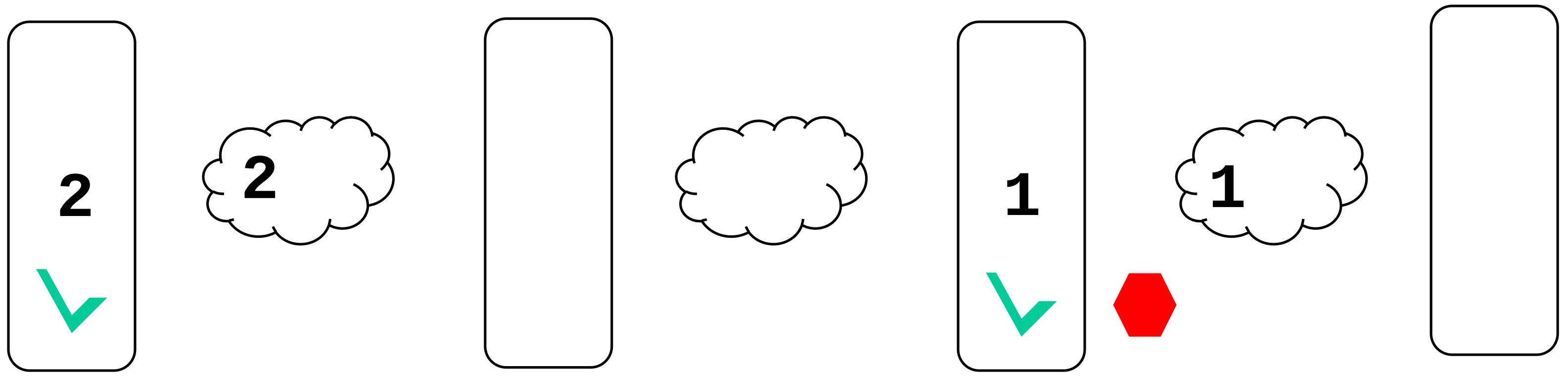
Simple Straight Example



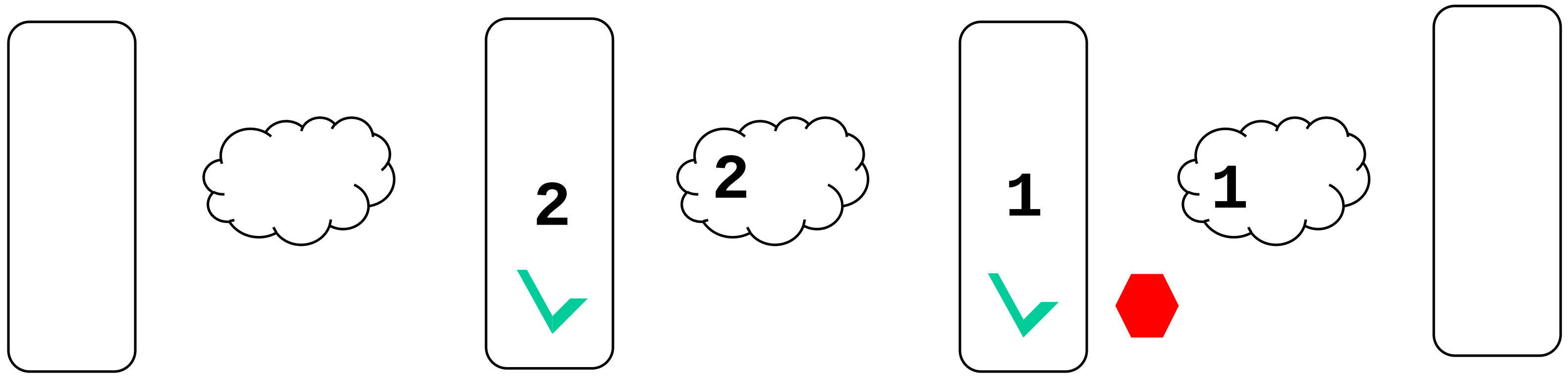
Simple Straight Example



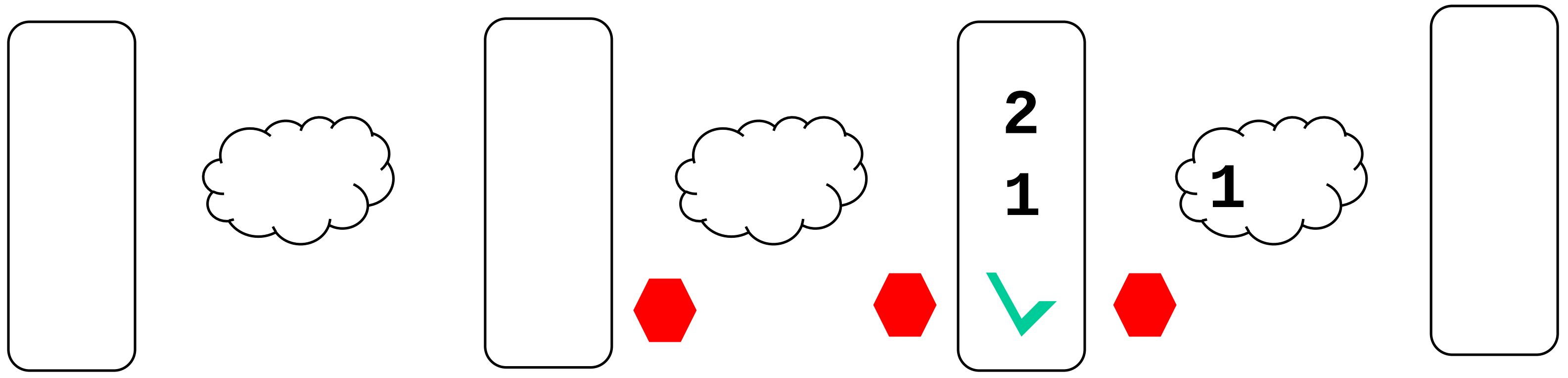
Simple Straight Example



Simple Straight Example

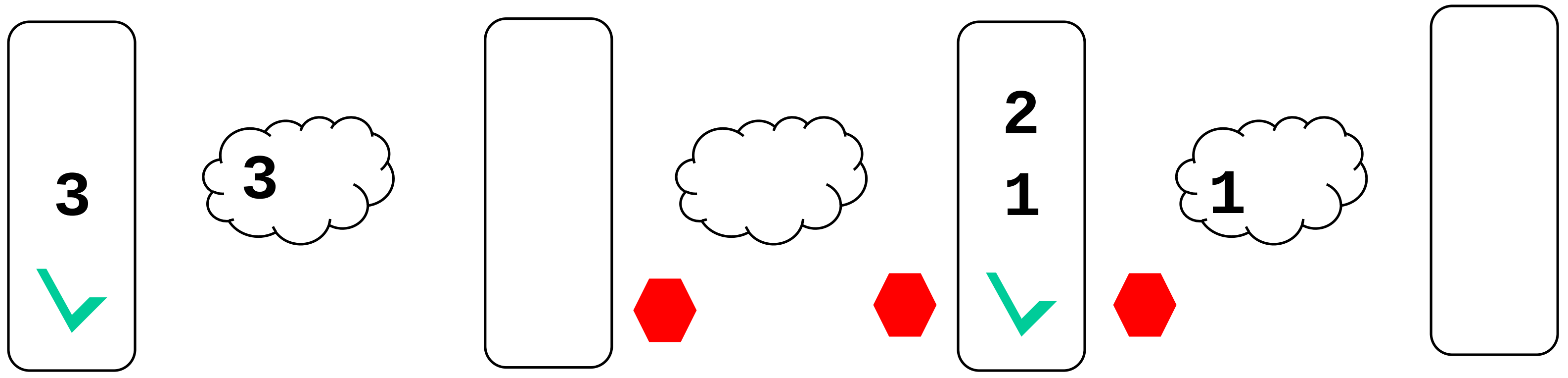


Simple Straight Example

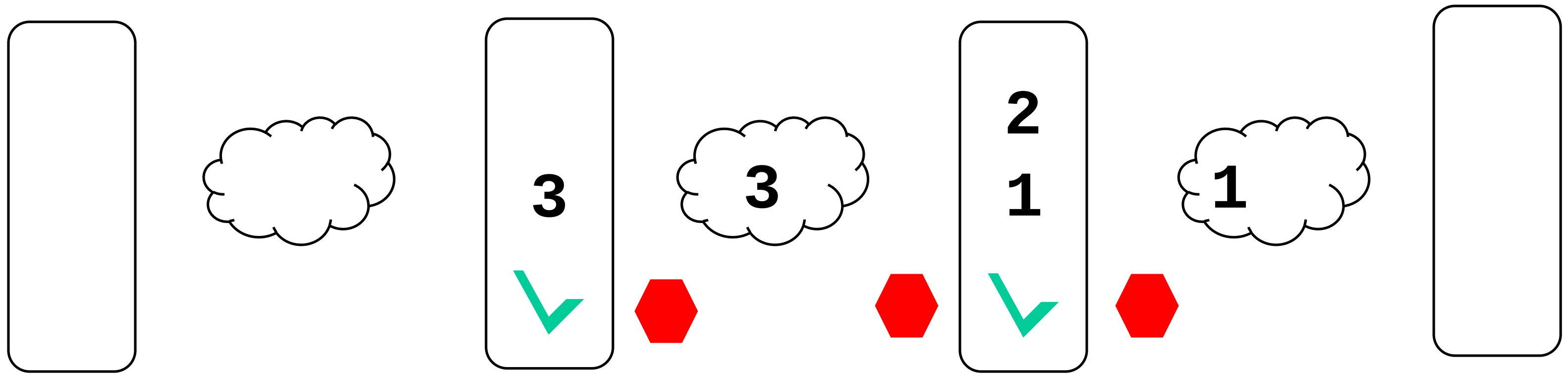


1 is stalled in the fluid flop, and 2 is kept in the shadow copy. Therefore, the “fifo” is full and stop is propagated to the previous stage

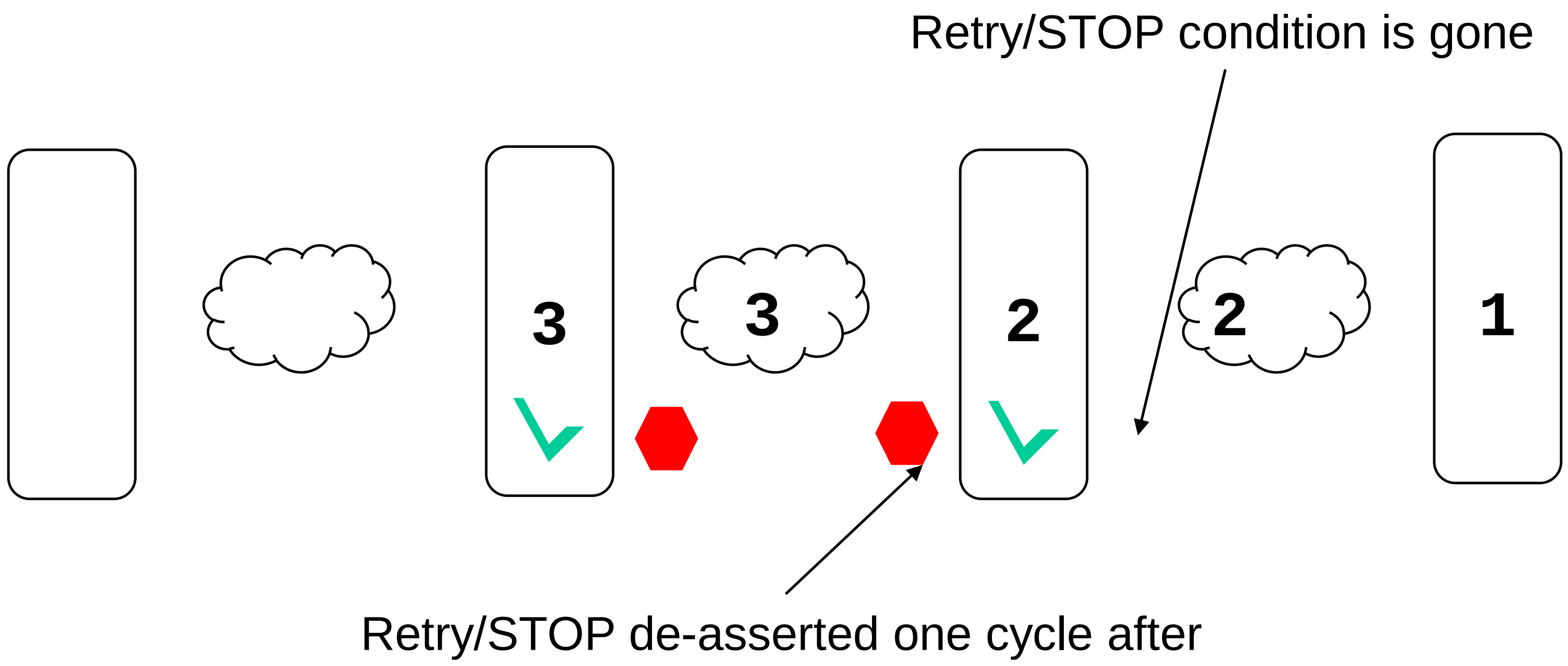
Simple Straight Example



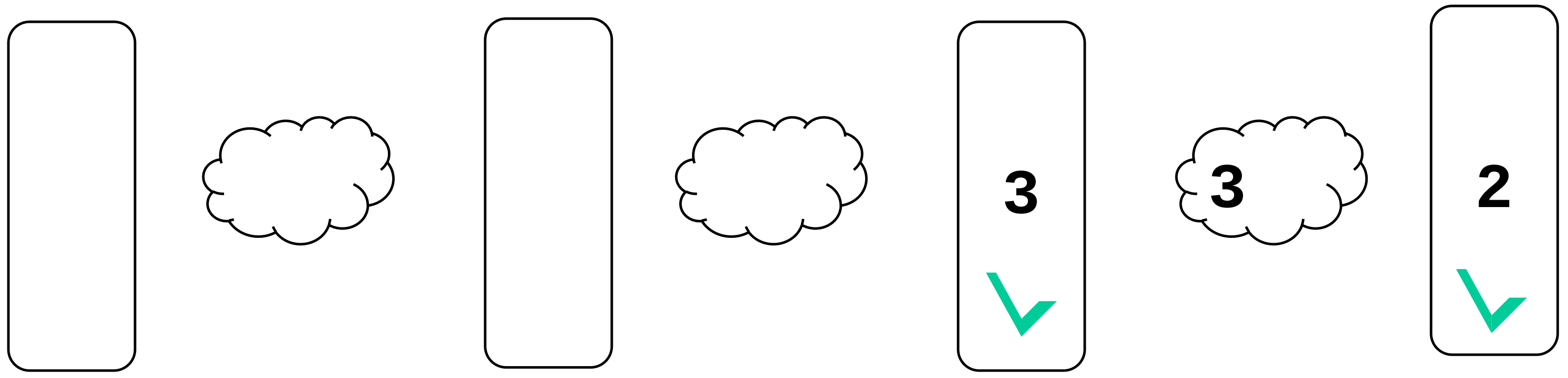
Simple Straight Example



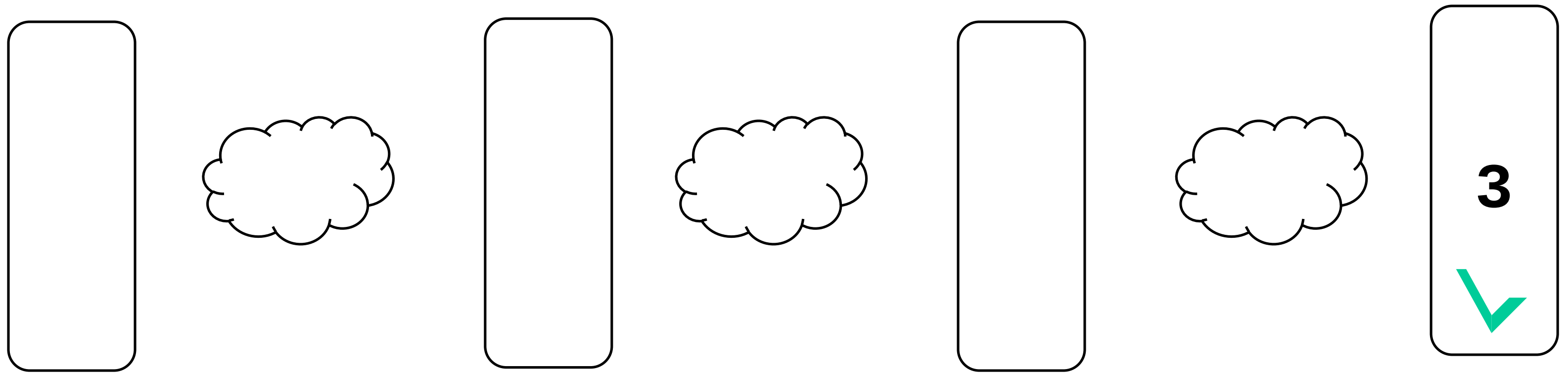
Simple Straight Example


Retry/STOP condition is gone
3
3
2
2
1
Retry/STOP de-asserted one cycle after

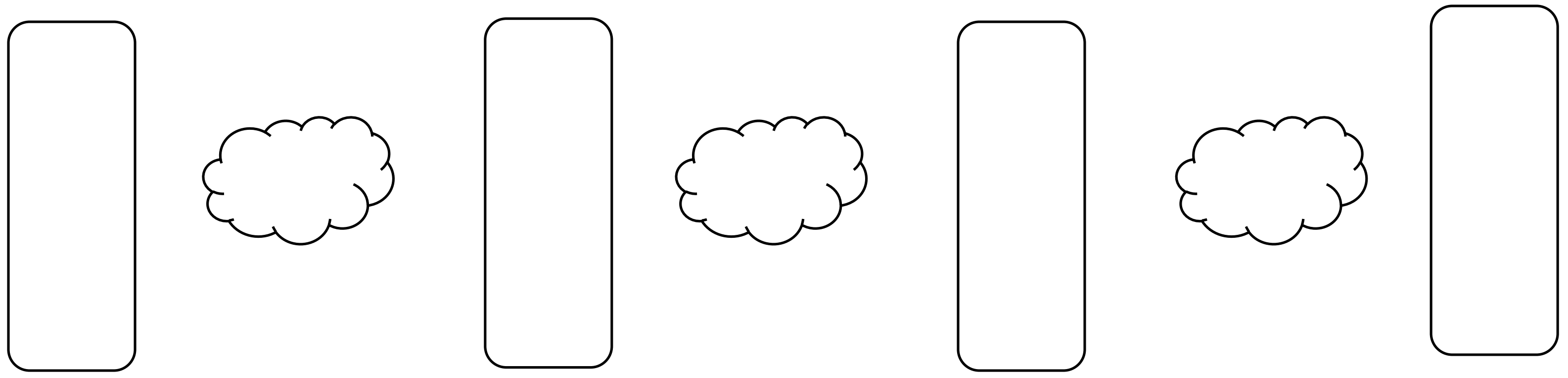
Simple Straight Example



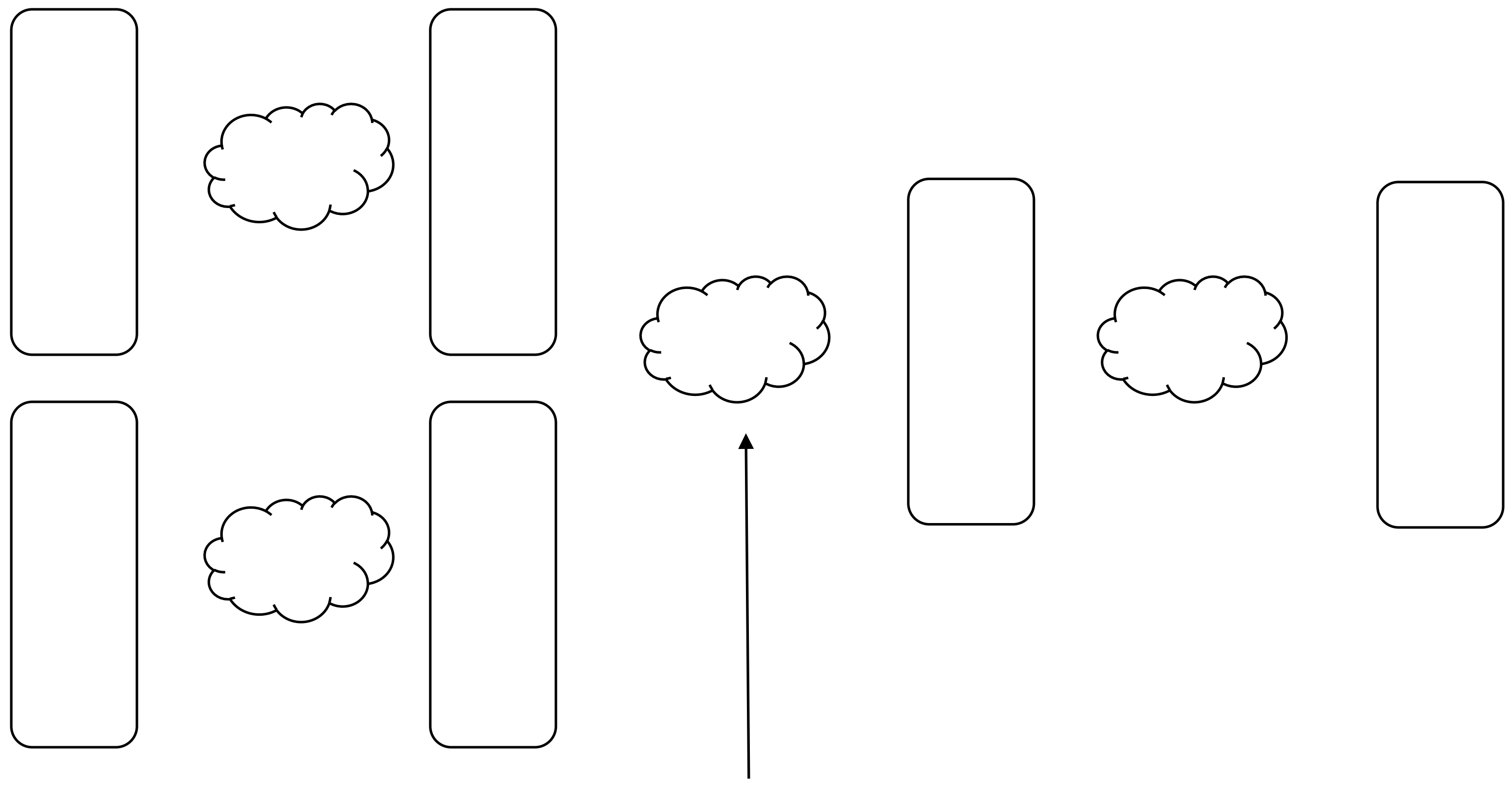
Simple Straight Example



Simple Straight Example

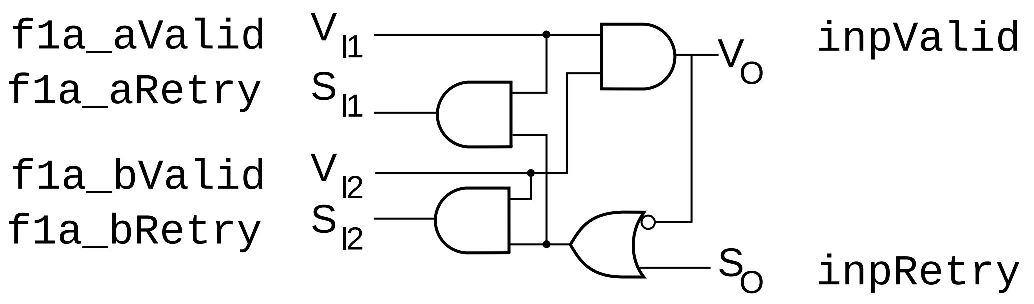


join Example



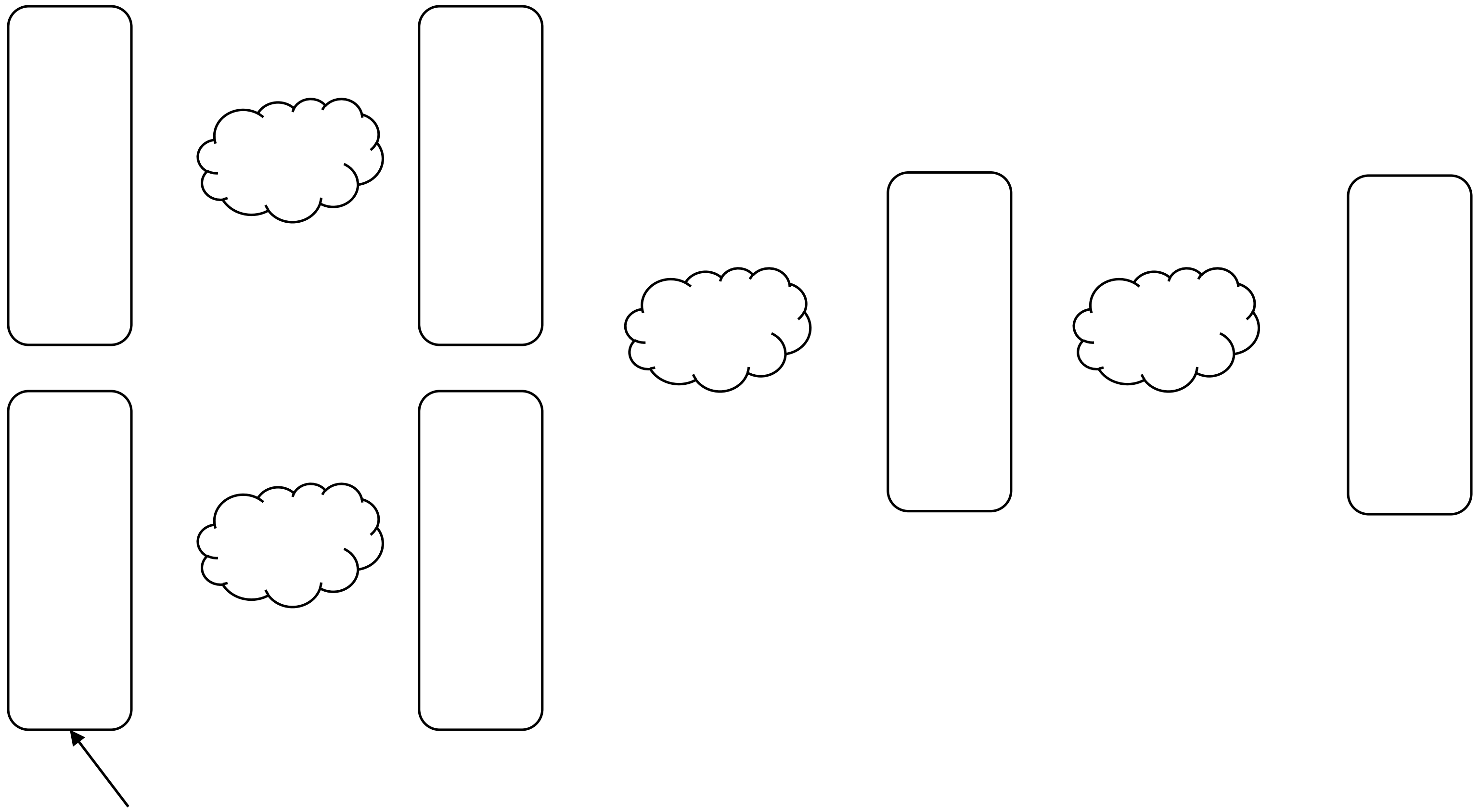
Some extra logic here to handle join

Fluid Join Example (join_test)



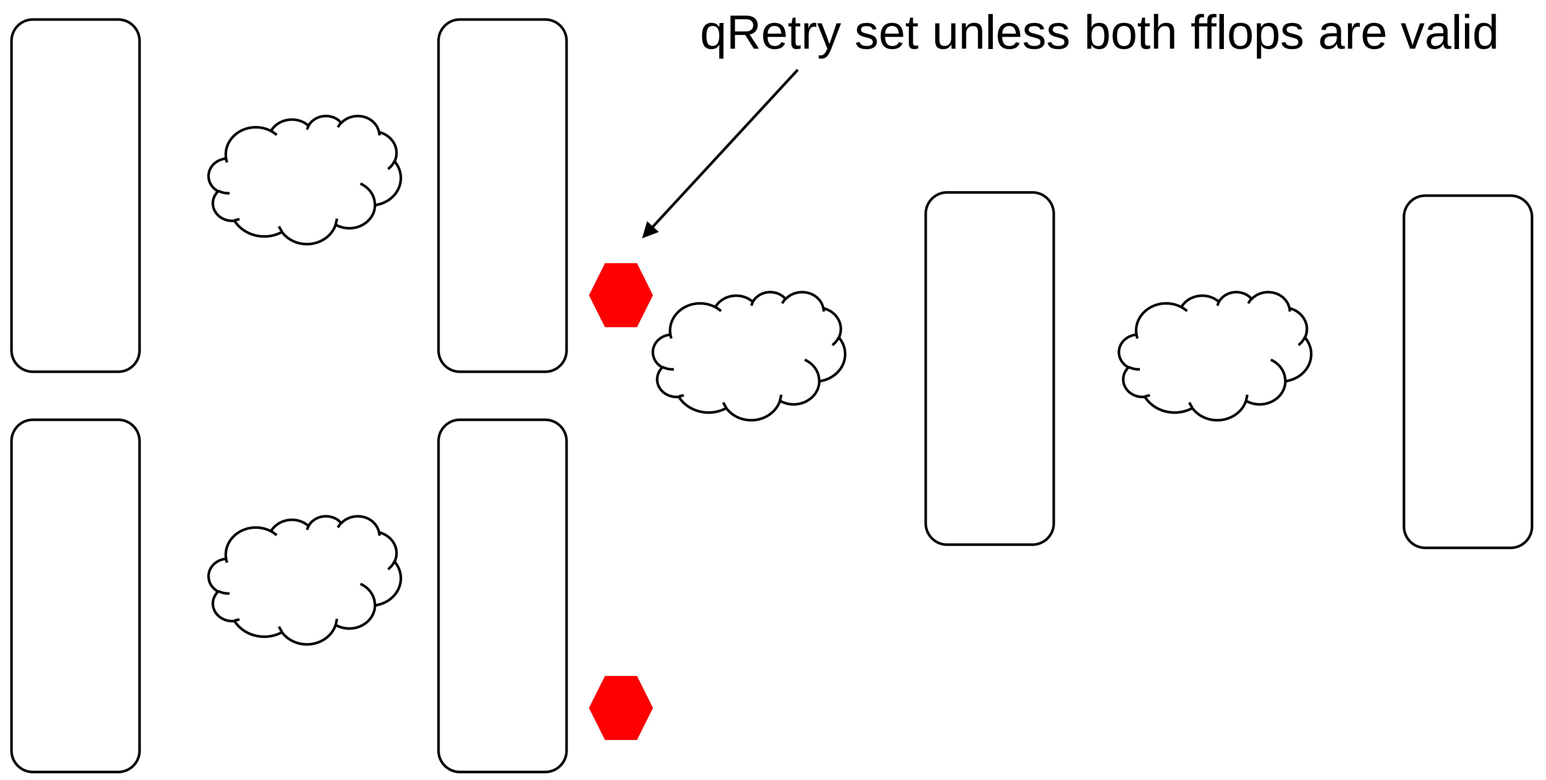
```
// code to handle join in join_test
always_comb begin
  inpValid = f1a_aValid && f1b_bValid;
end

always_comb begin
  f1b_bRetry = inpRetry || !inpValid;
  f1a_aRetry = inpRetry || !inpValid;
end
```

join Example

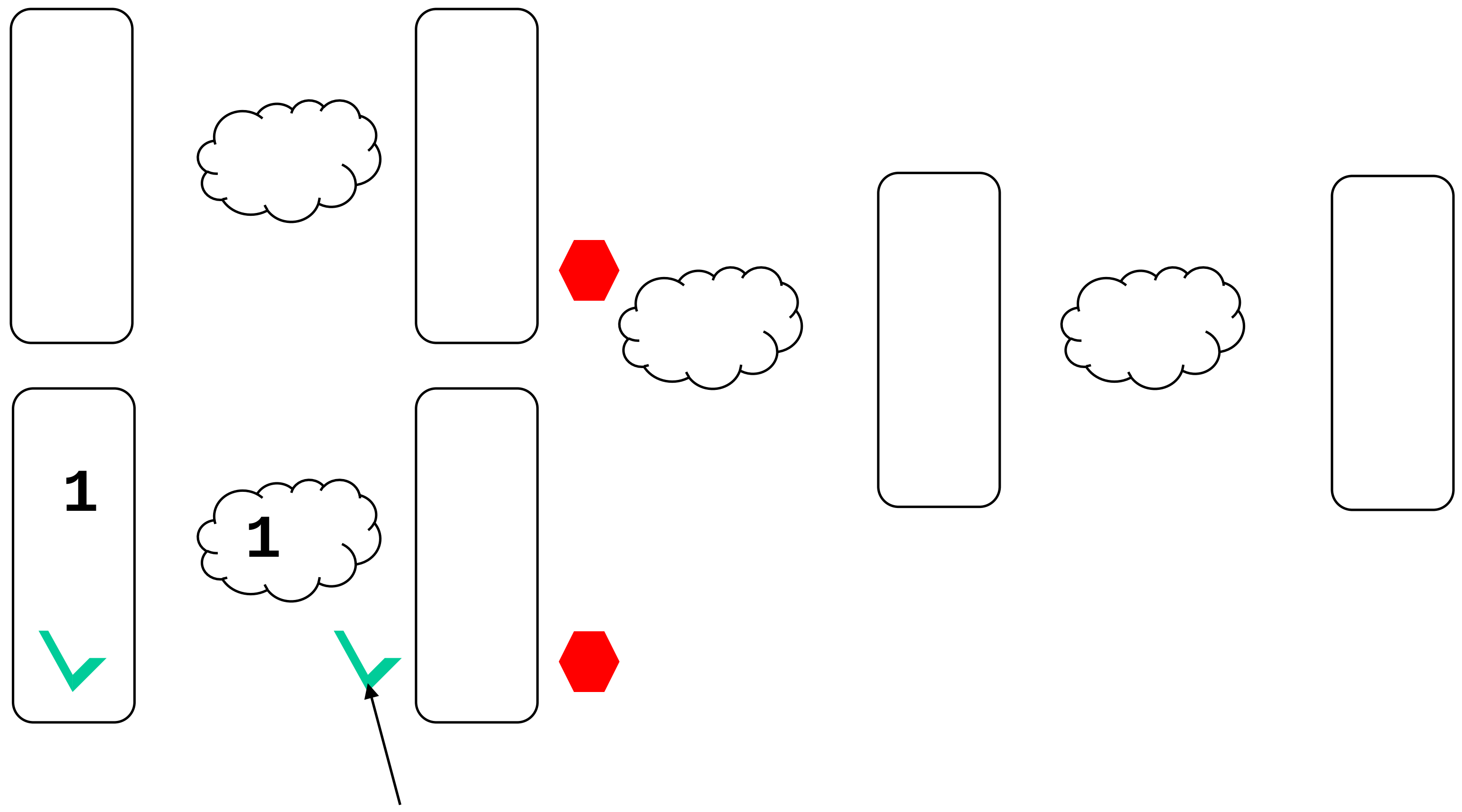


Inputs to stage with fflop

join Example

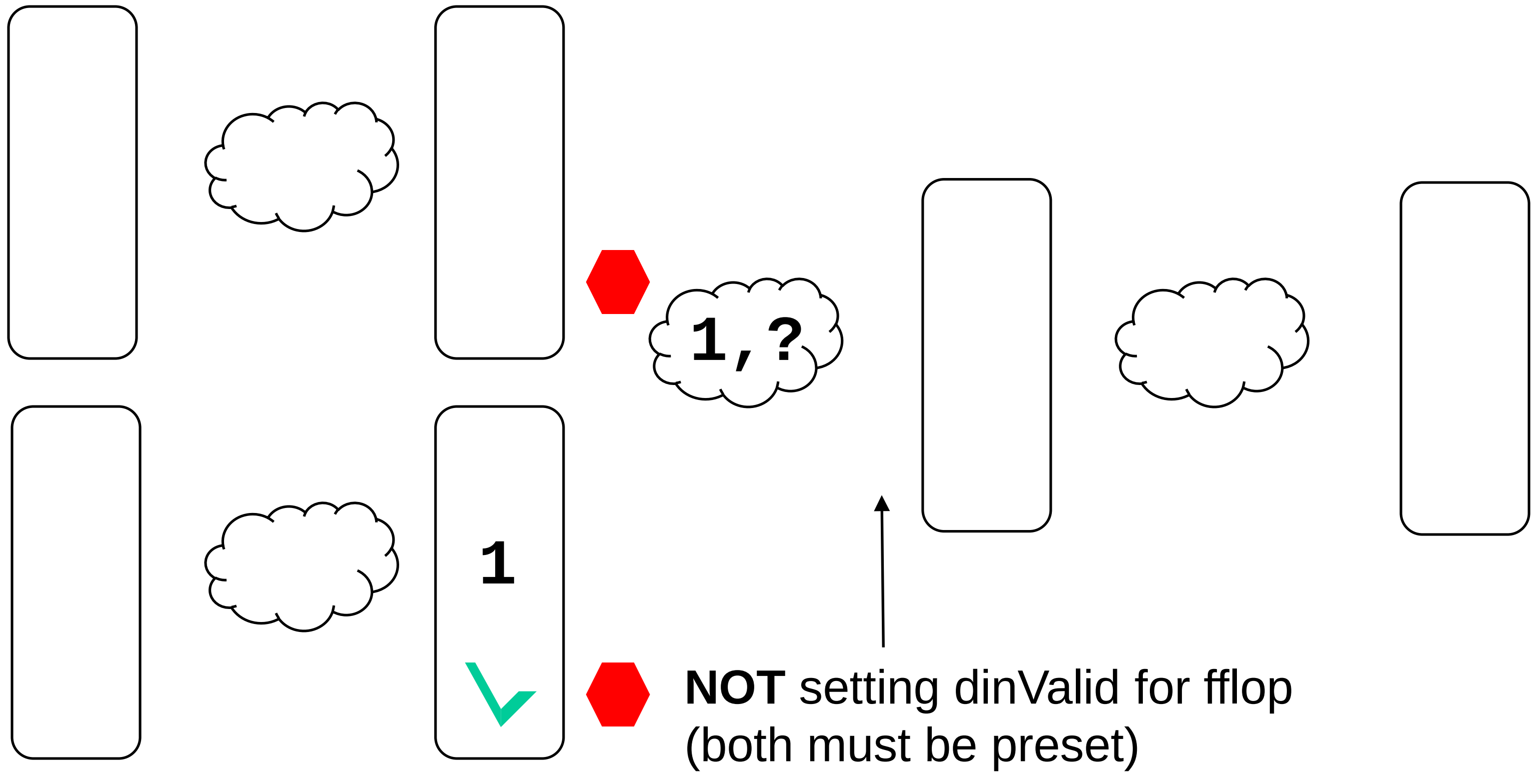


join Example



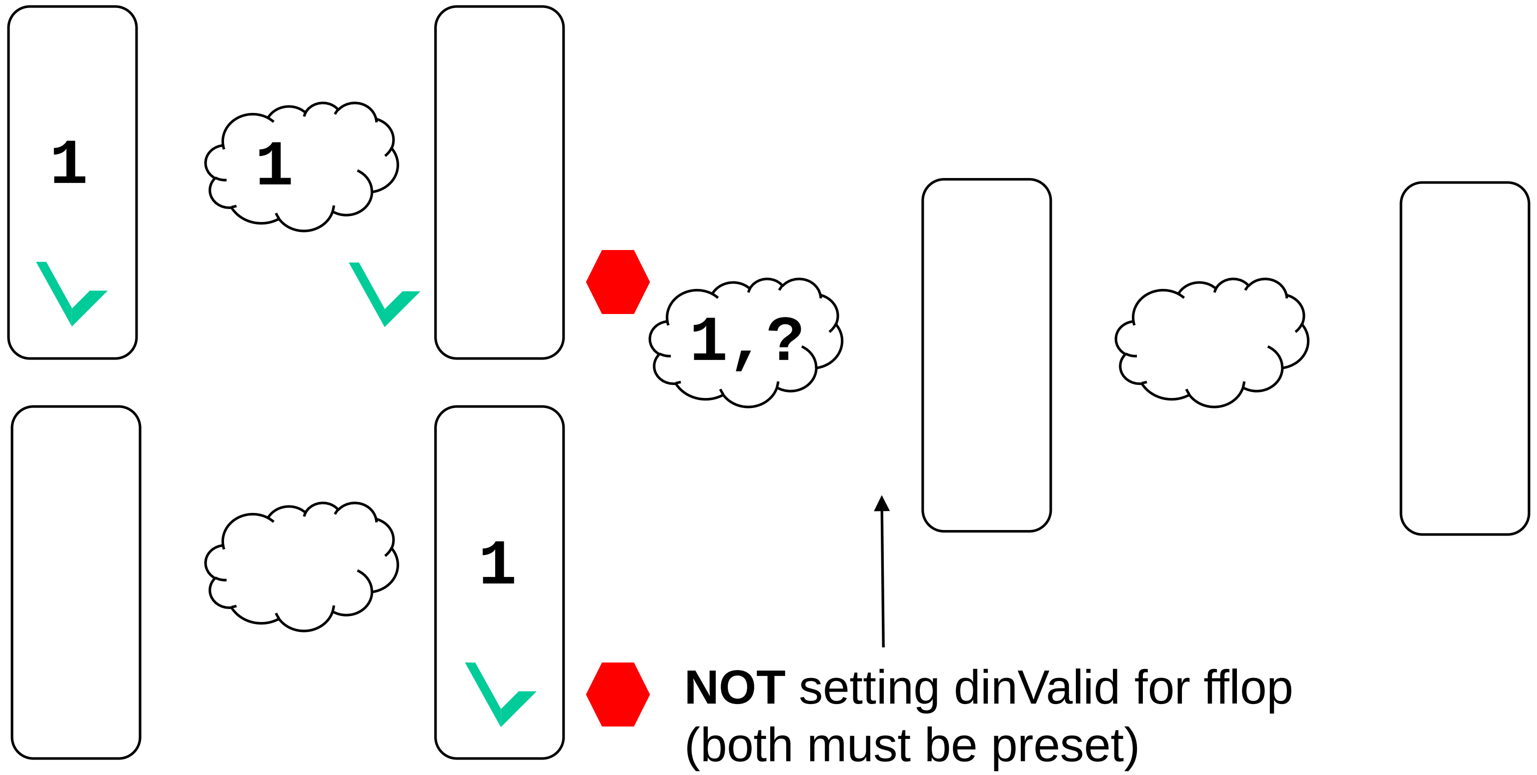
Setting dinValid for fflop

join Example



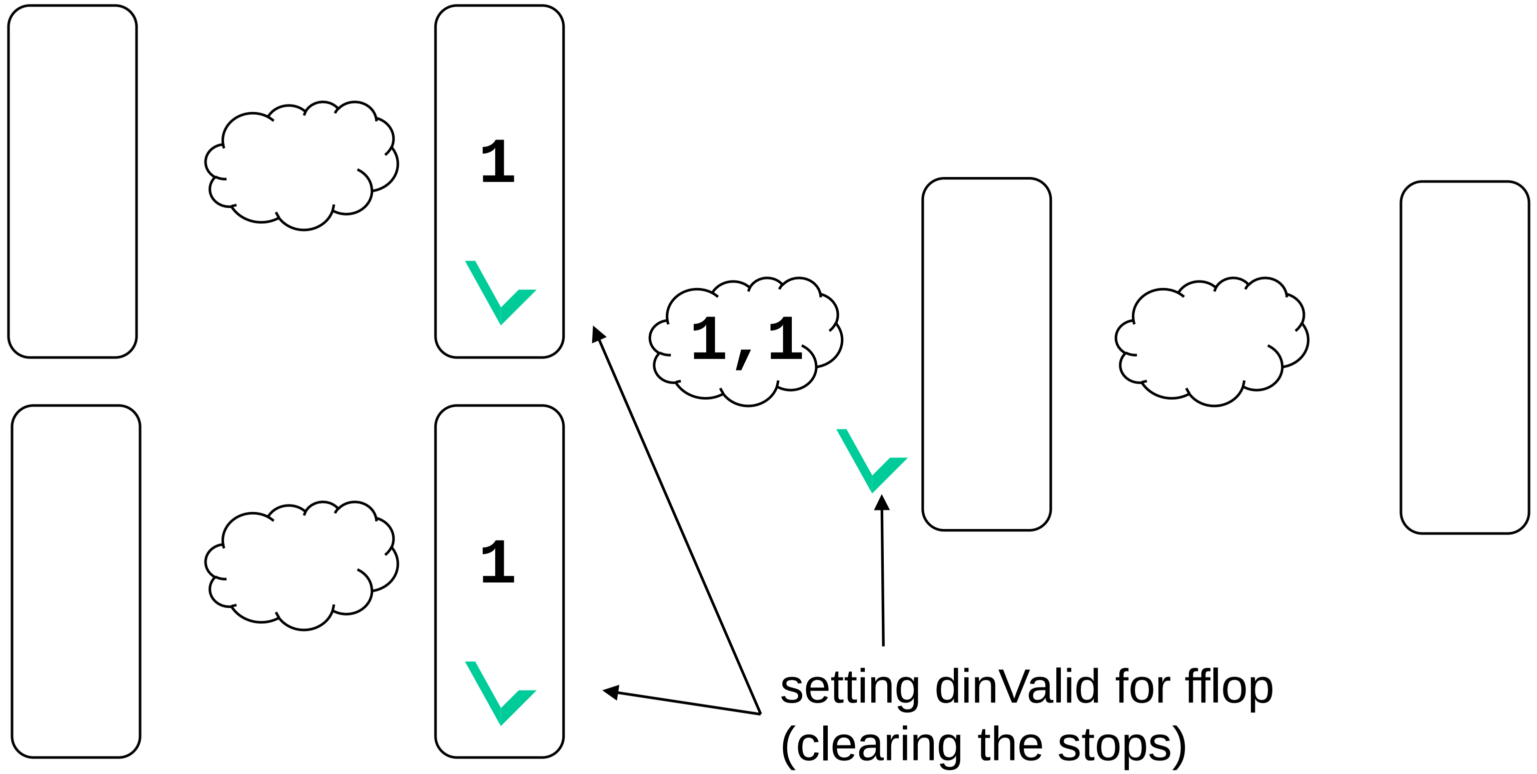
NOT setting dinValid for fflop
(both must be preset)

join Example

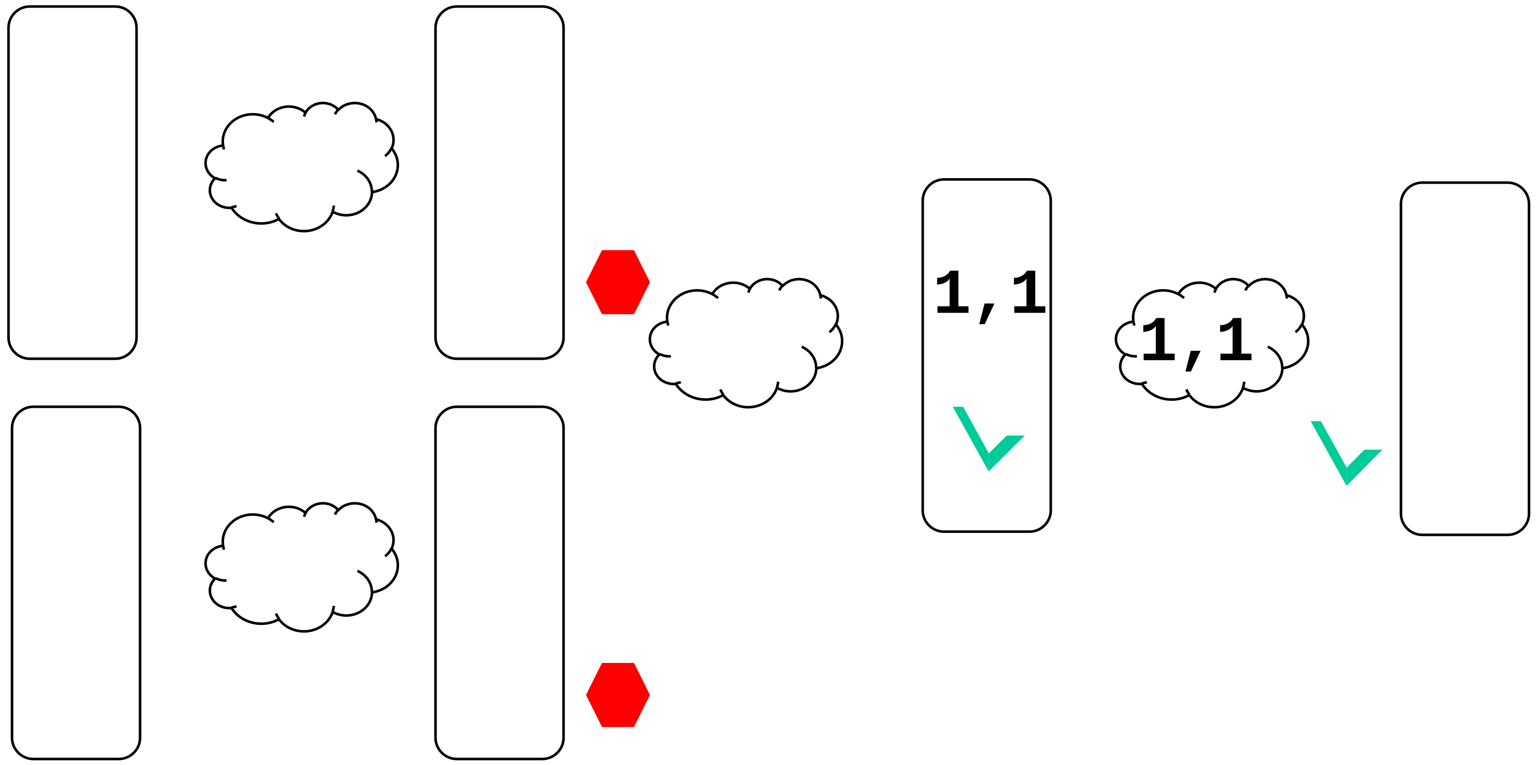


NOT setting dinValid for fflop
(both must be preset)

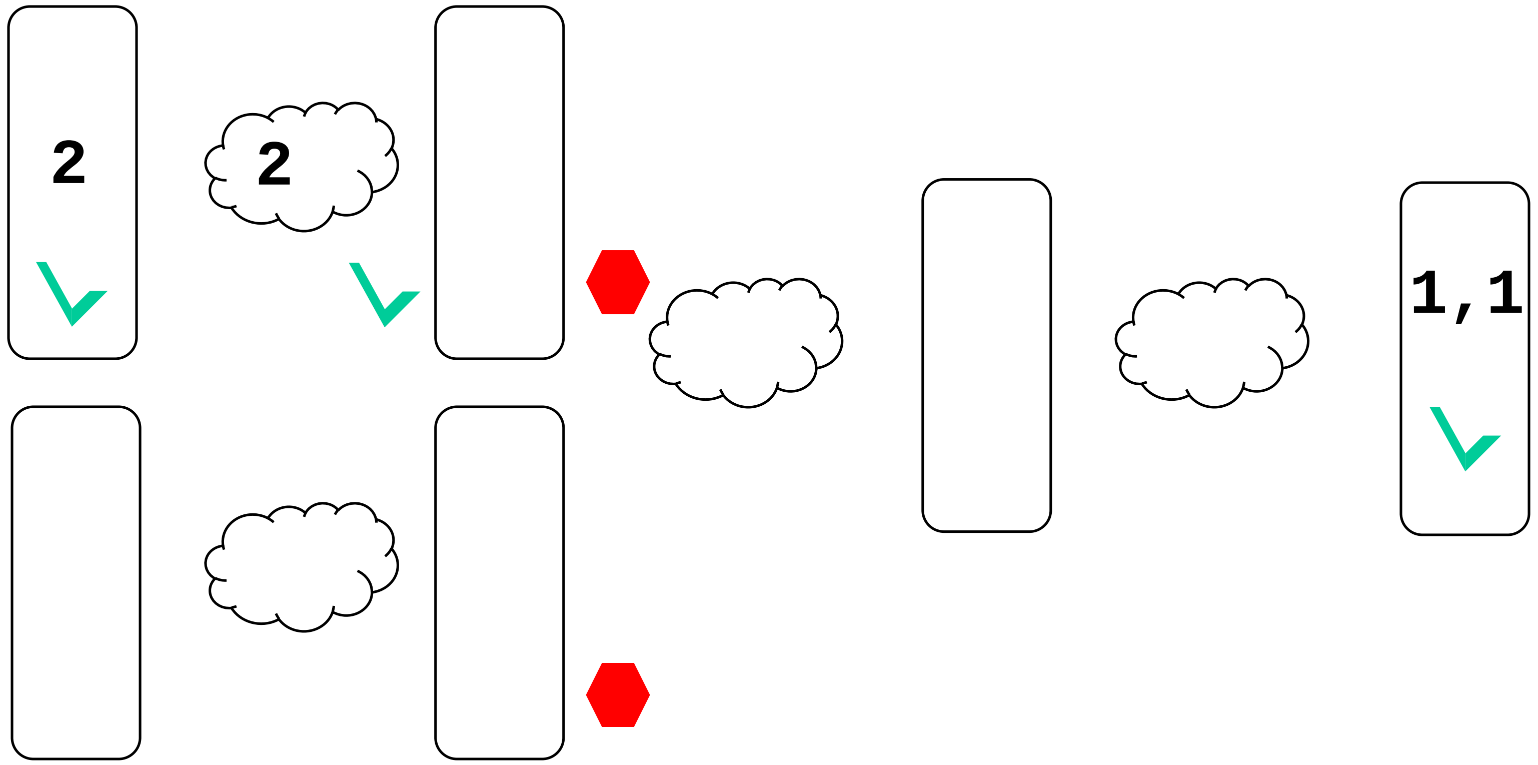
join Example



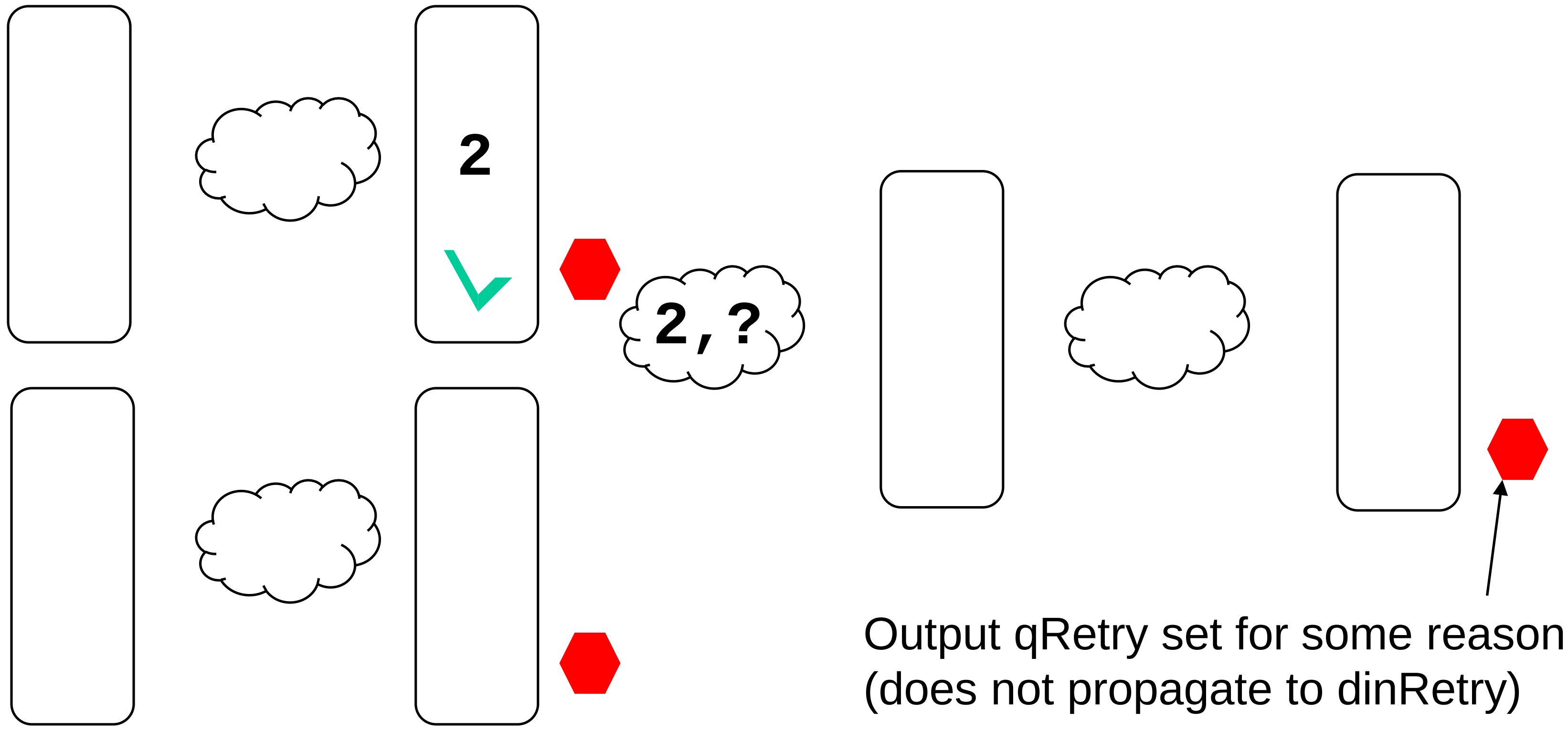
join Example



join Example

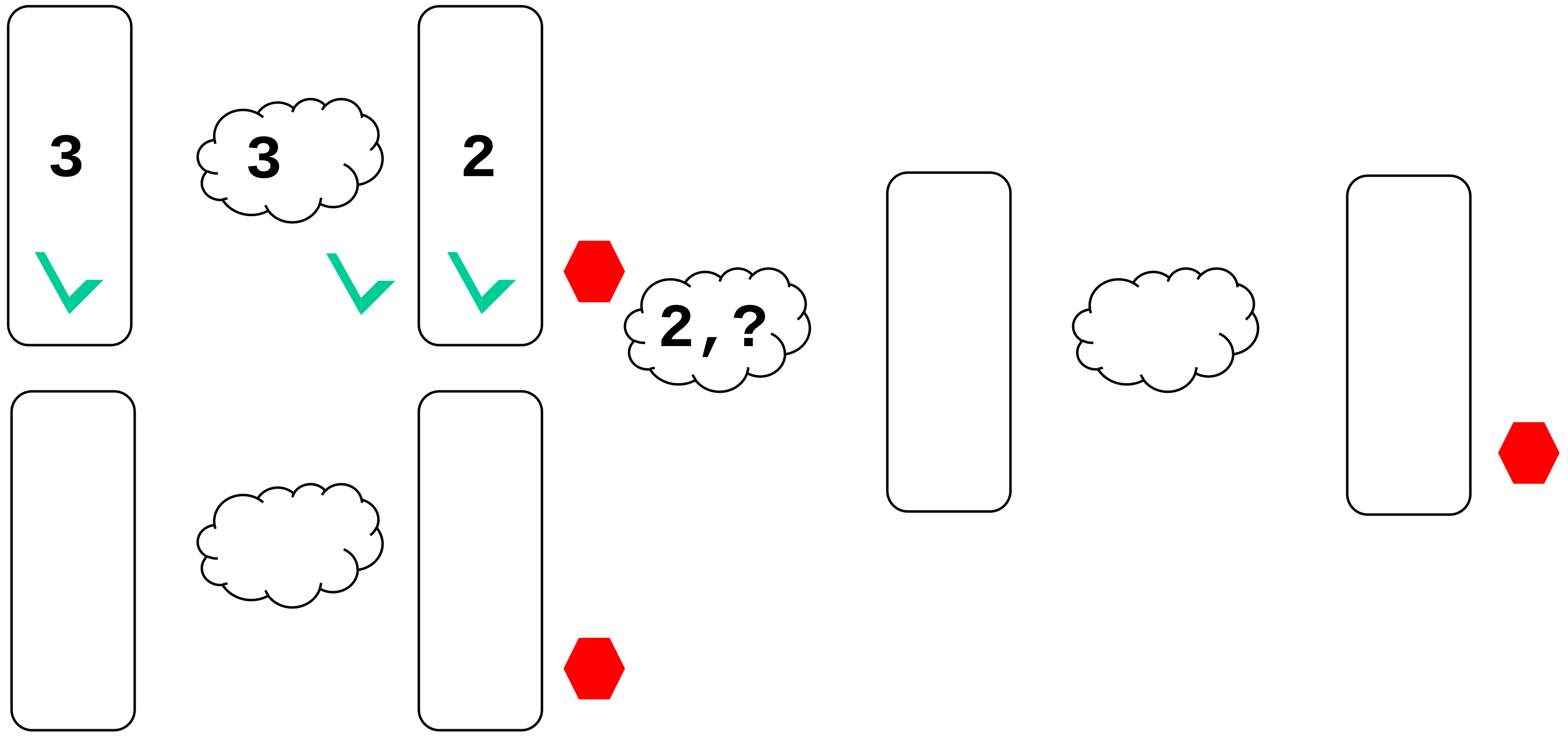


join Example

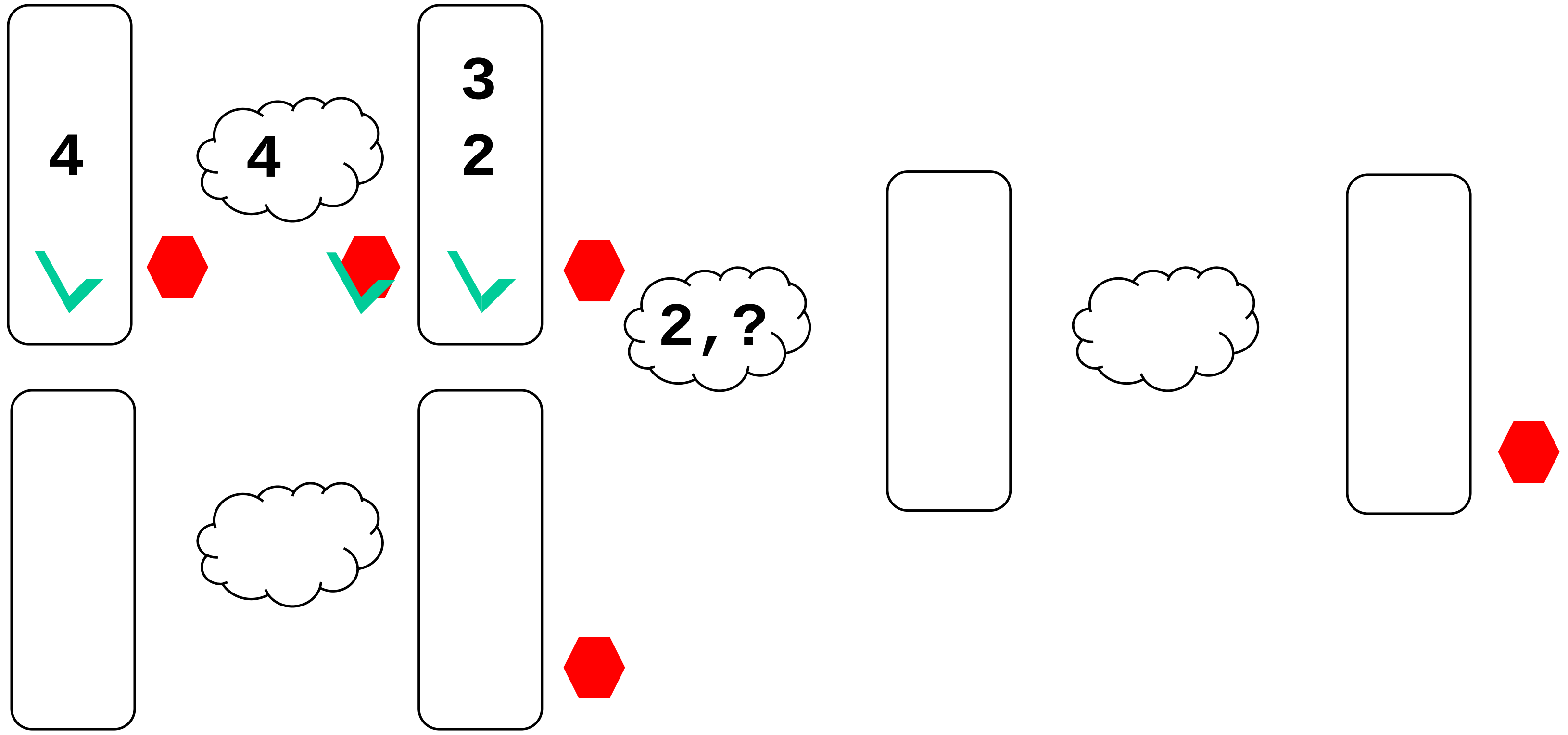


Output qRetry set for some reason
(does not propagate to dinRetry)

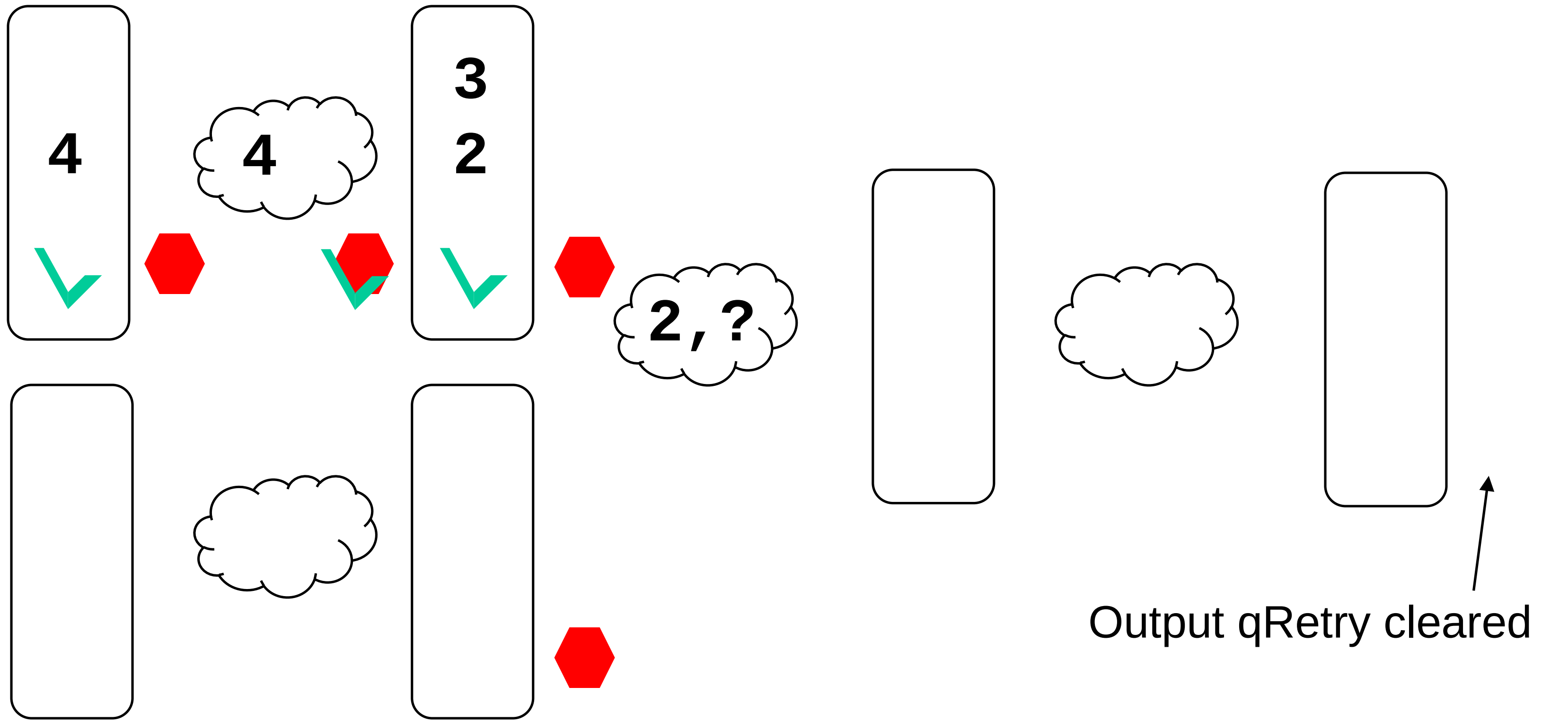
join Example



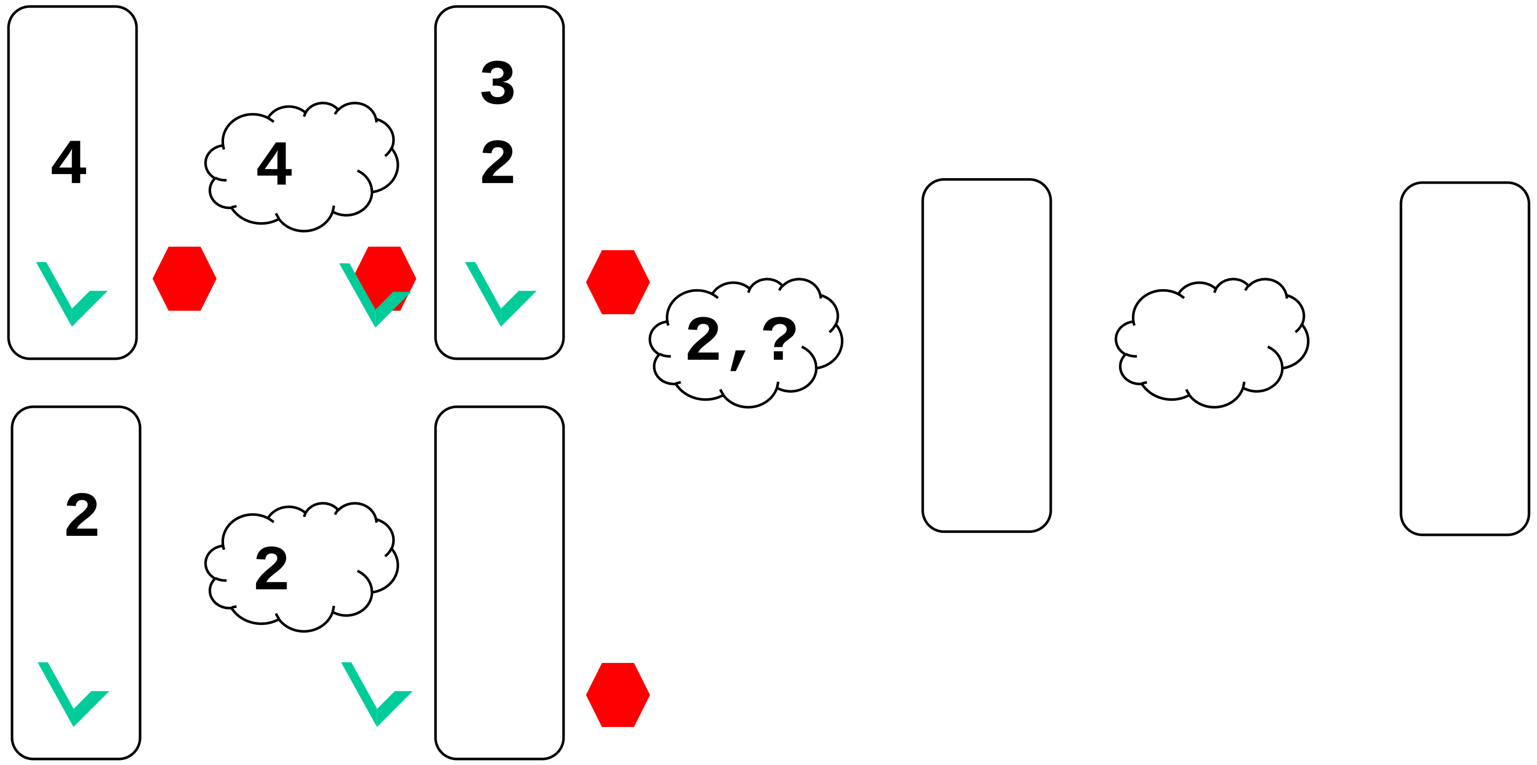
join Example



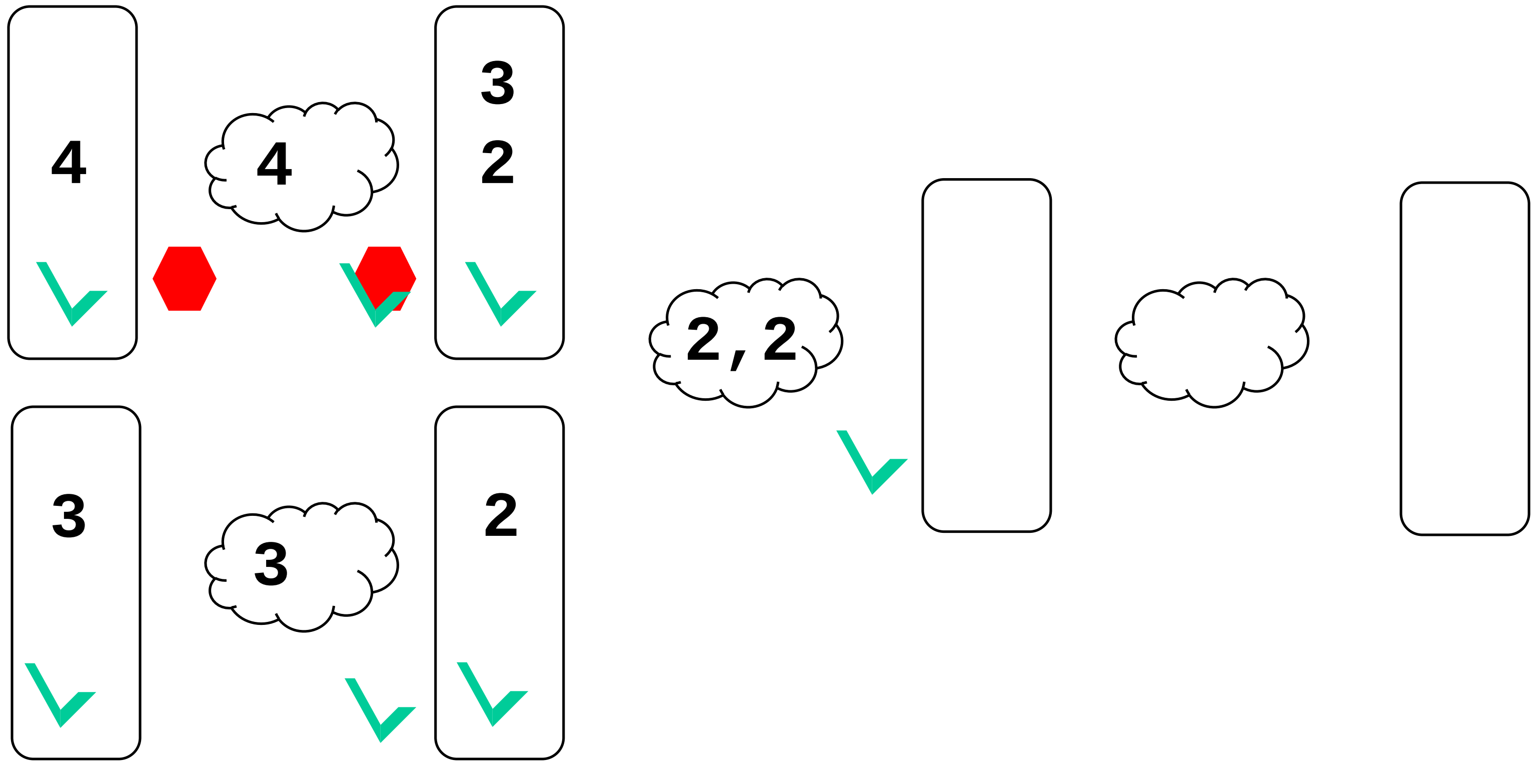
join Example



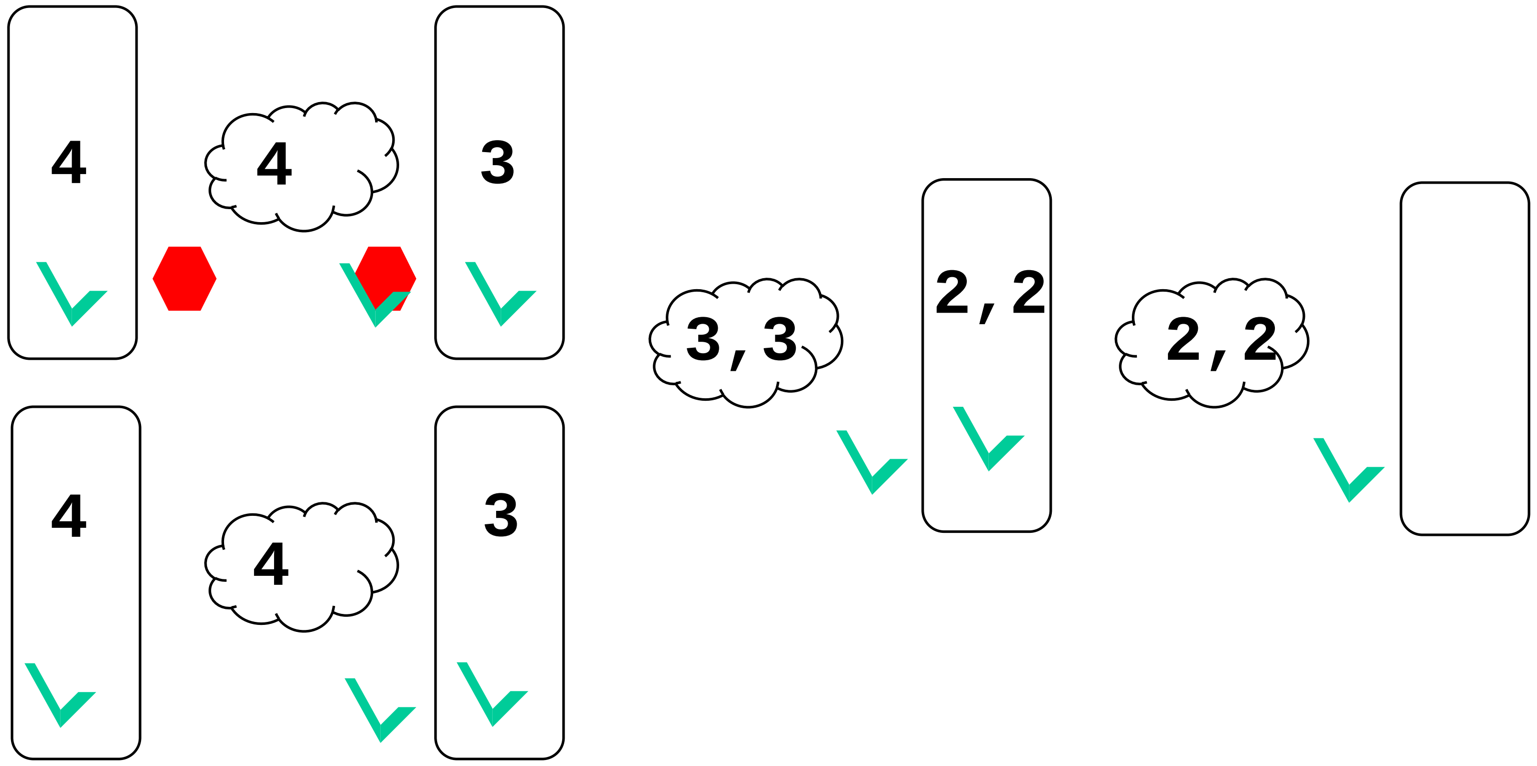
join Example



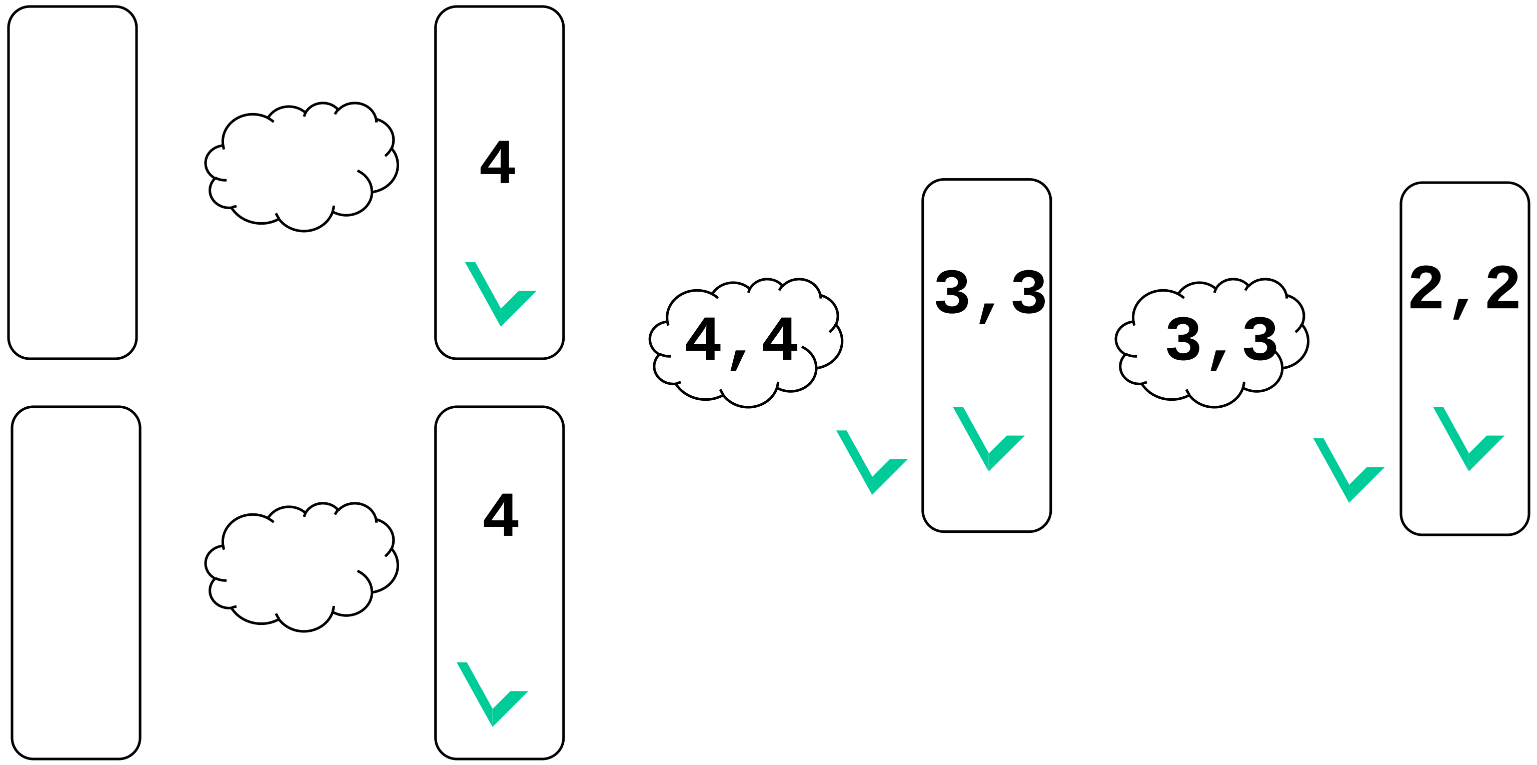
join Example



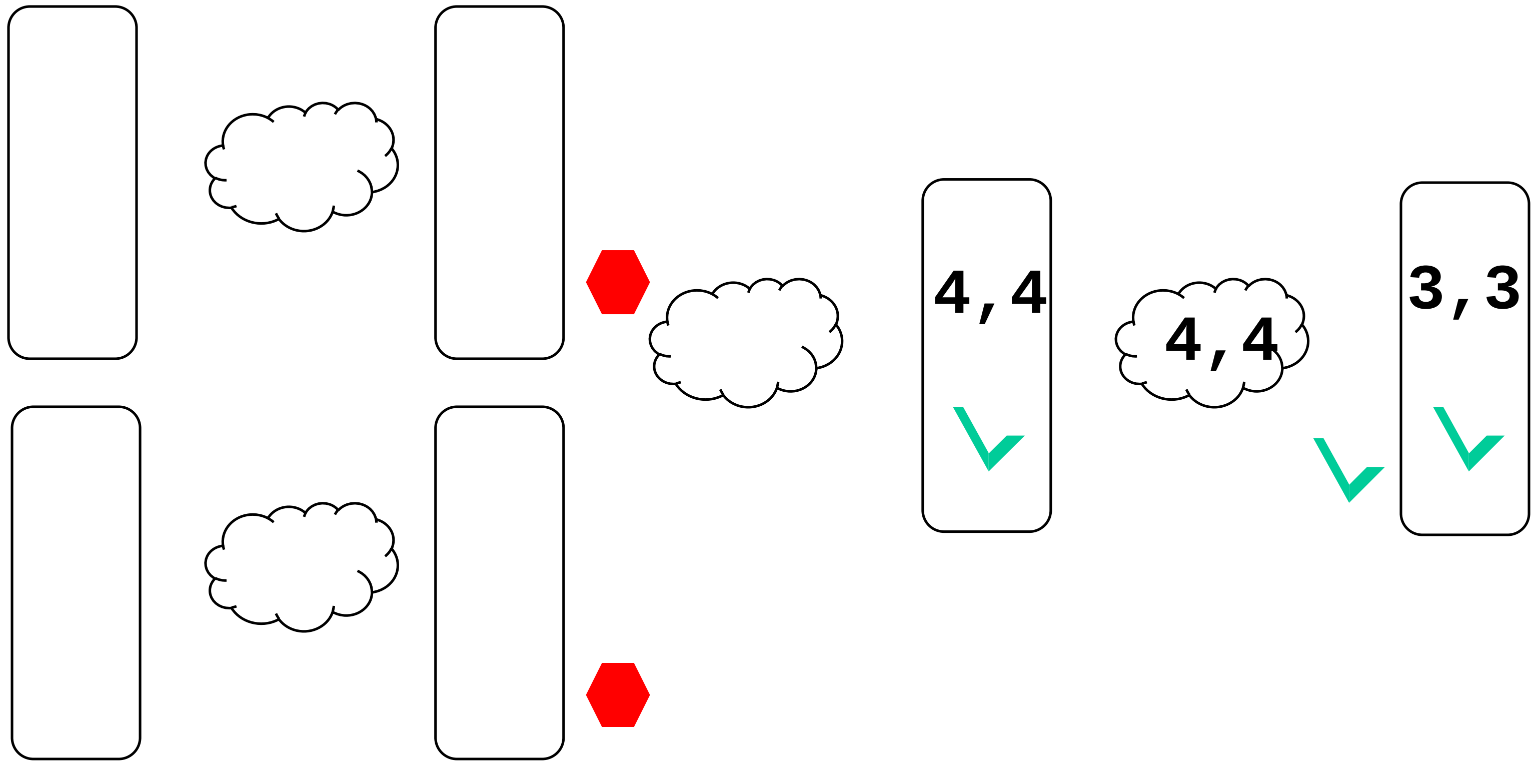
join Example



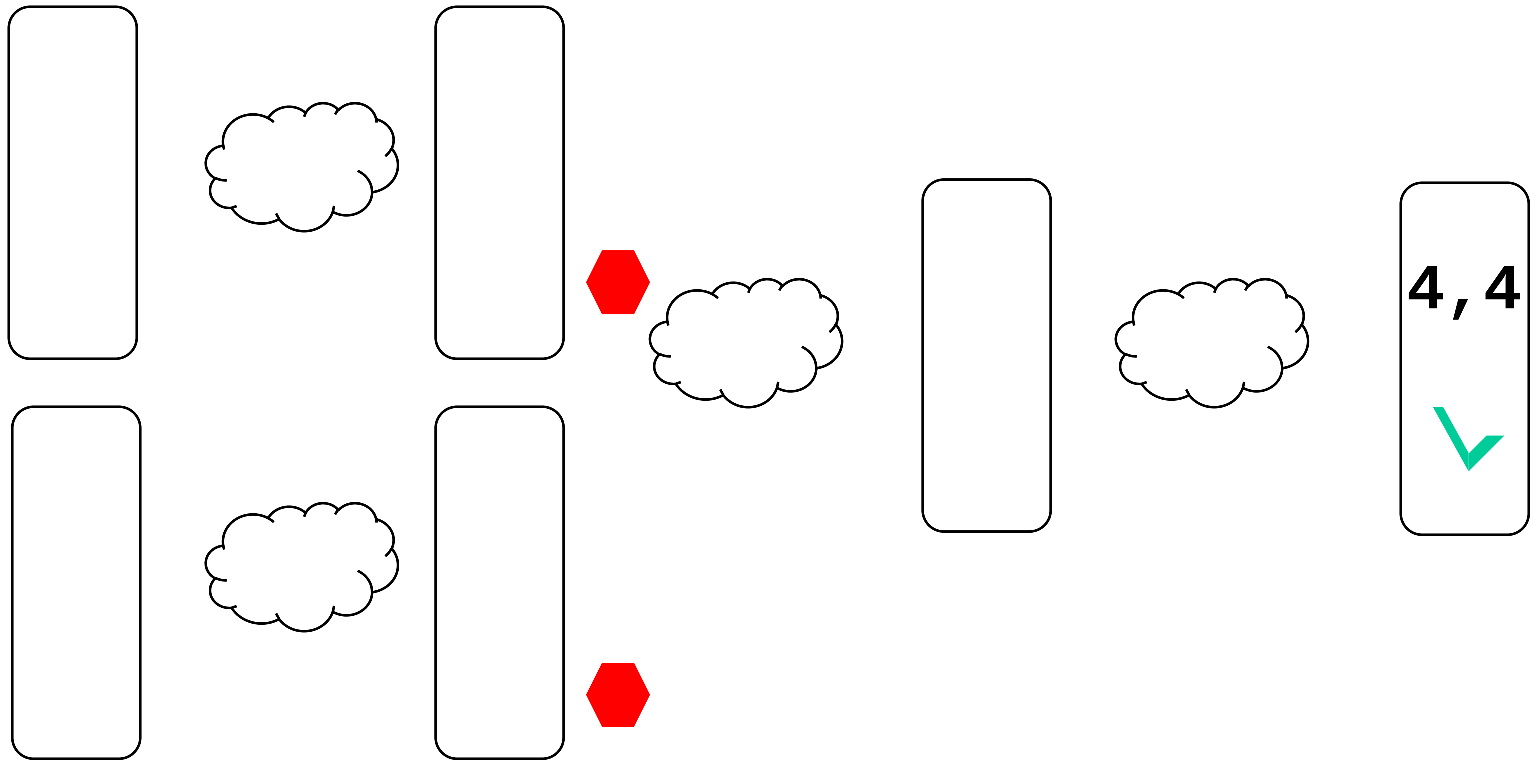
join Example



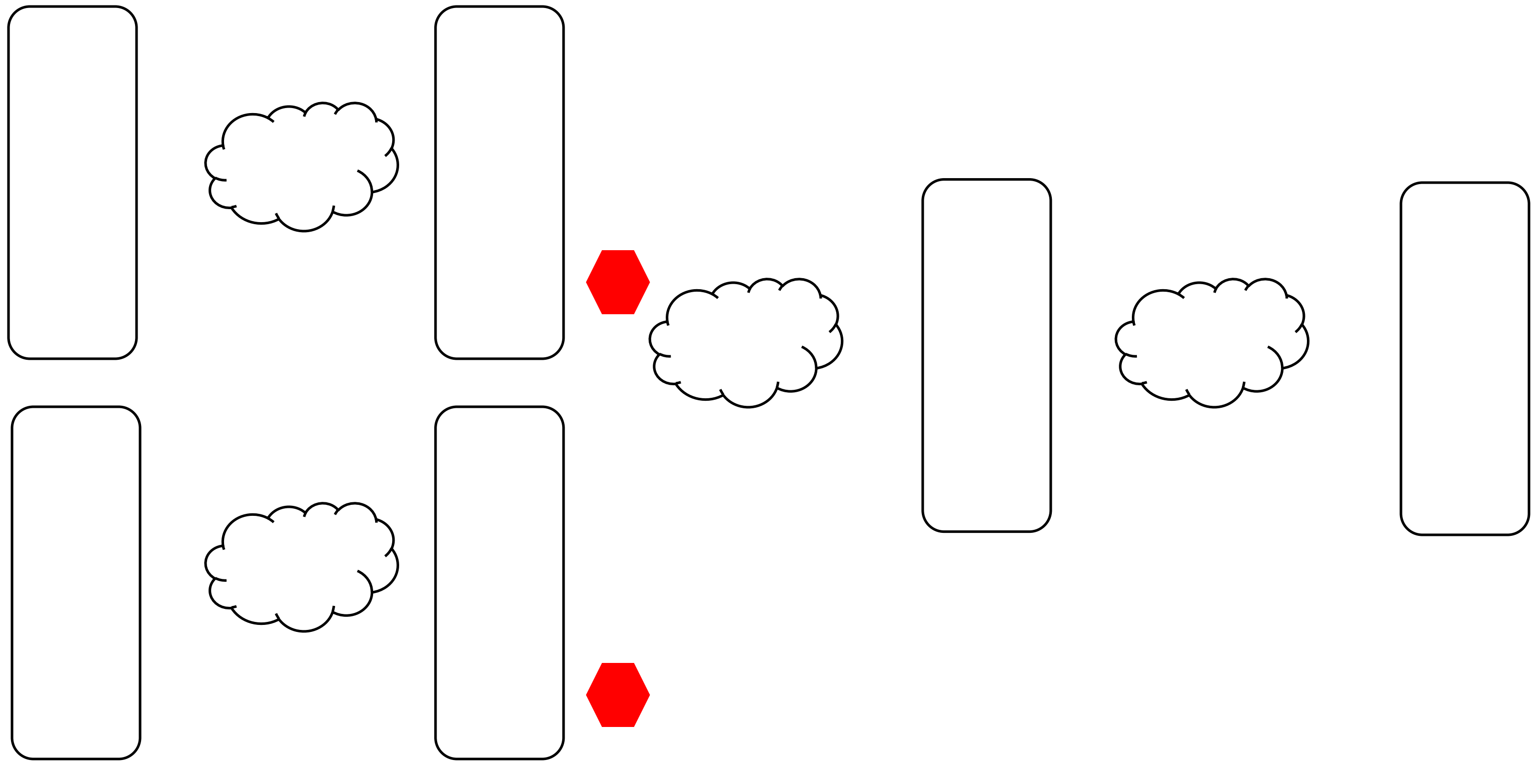
join Example



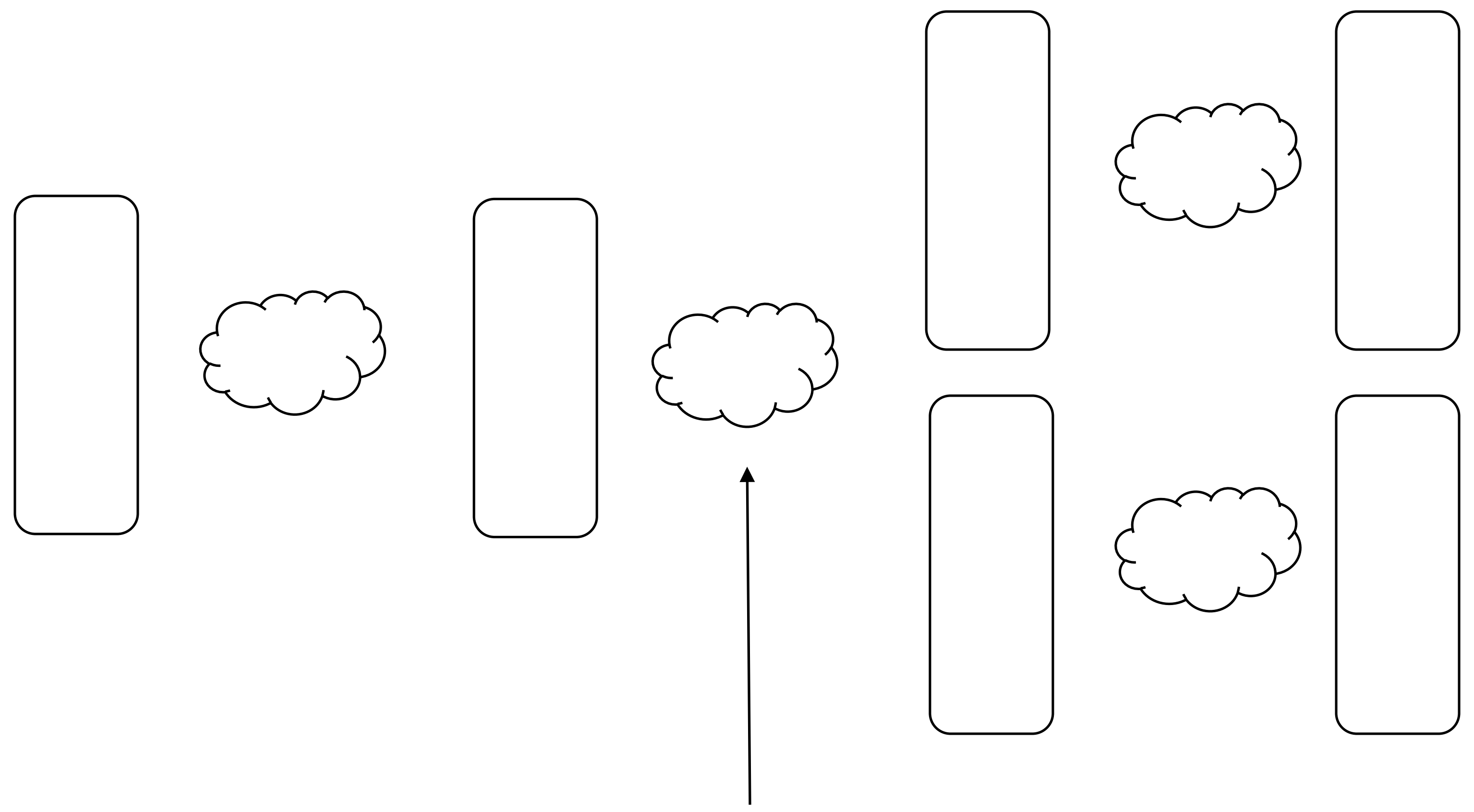
join Example



join Example

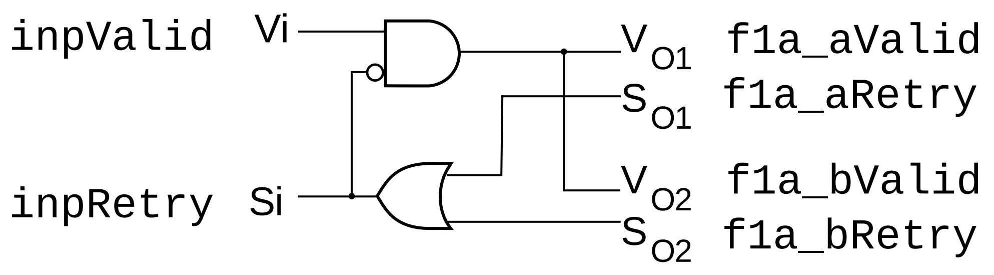


fork Example



Some extra logic here to handle fork

Fluid Fork Example (fork_test)



```
// code to handle fluid fork in fork_test
```