Synchronous Elastic Systems

Mike Kishinevsky and Jordi Cortadella

Intel
Strategic CAD Labs
Hillsboro, USA

Universitat Politecnica
de Catalunya
Barcelona, Spain

DAC Summer School
July 26, 2009

Contributors to SELF research

- Micro-architectural pipelining, speculation
  Marc Galceran Oms, Timothy Kam
- Design experiments: Alexander Gotmanov
- Performance analysis: Jorge Júlvez
- Theory of elastic machines:
  Sava Krstic and John O'Leary
- Optimization: Dmitry Bufistov, Josep Carmona
- Bill Grundmann

Agenda

I. Basics of elastic systems

II. Why to study

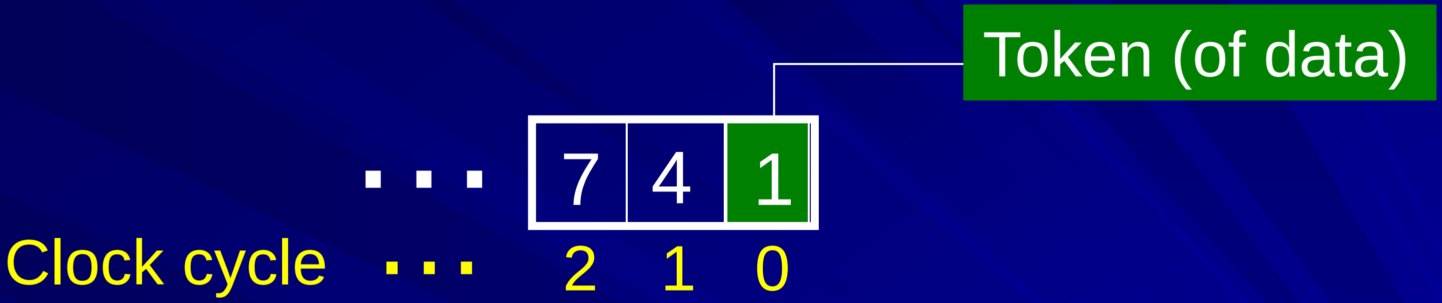
III. Early evaluation and performance analysis

IV. Correct-by-construction pipelining

V. Communication fabrics

VI. Open problems

Synchronous Stream of Data

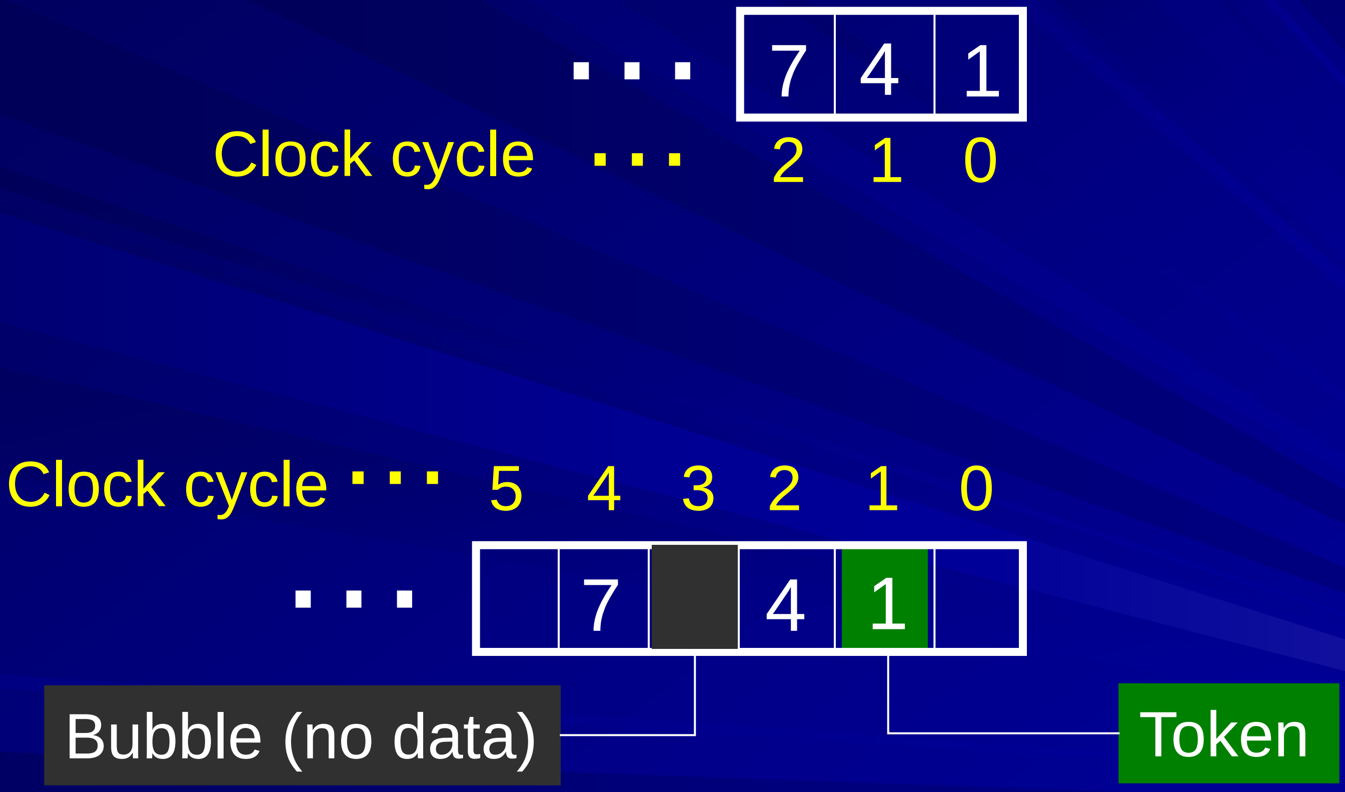


Token (of data)

. . . 7 4 1

Clock cycle . . . 2 1 0

Synchronous Elastic Stream



. . . | 7 | 4 | 1

Clock cycle . . . 2 1 0

Clock cycle . . . 5 4 3 2 1 0

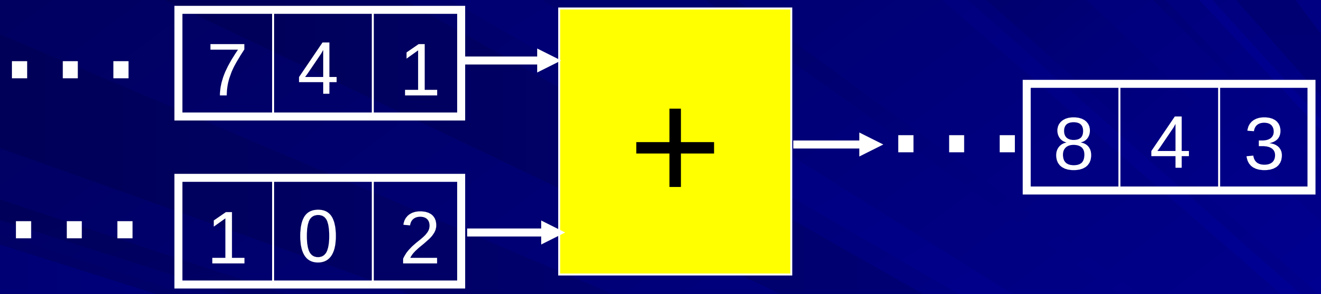
. . . | | 7 | | 4 | 1 | |

Bubble (no data)

Token

Synchronous Circuit

Latency = 0



. . . 7 4 1

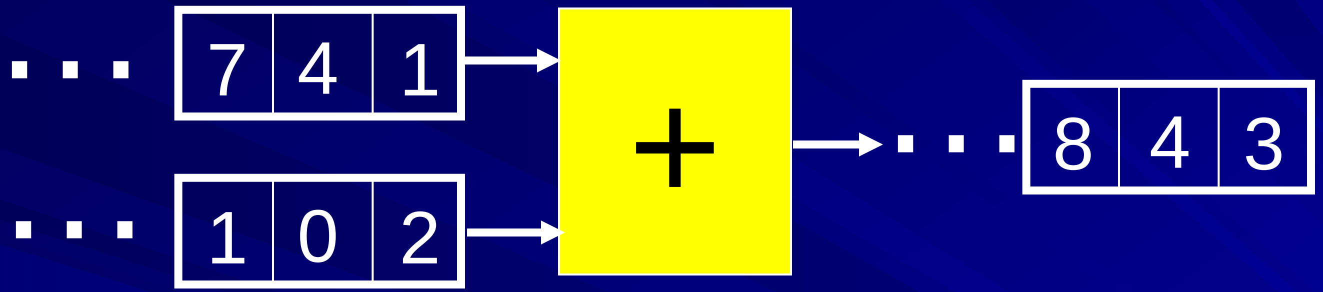
. . . 1 0 2

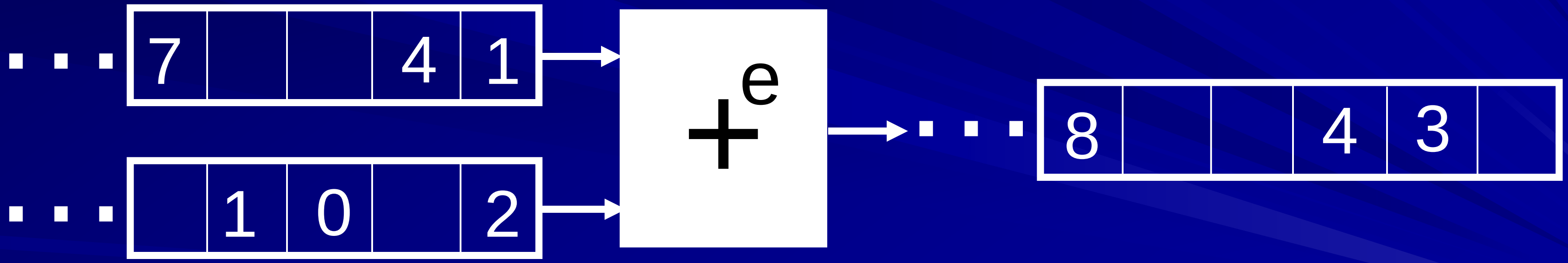
+

. . . 8 4 3

Synchronous Elastic Circuit

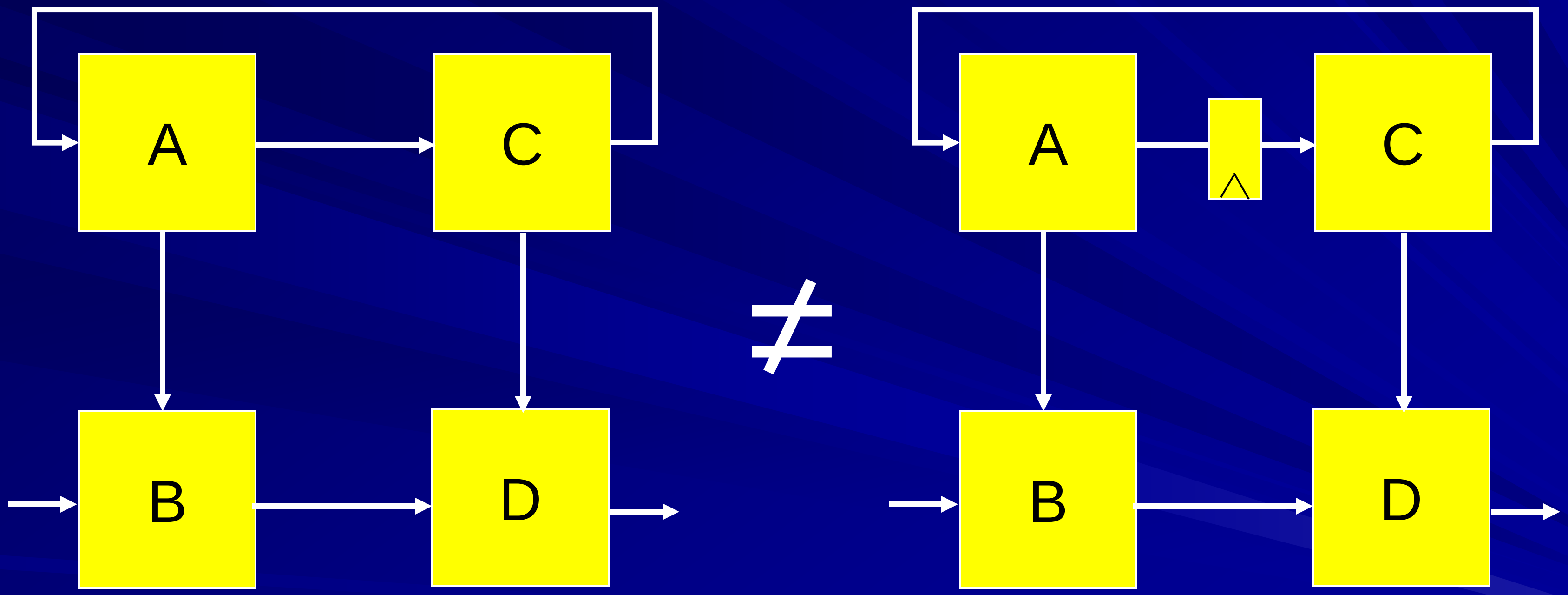
Latency = 0





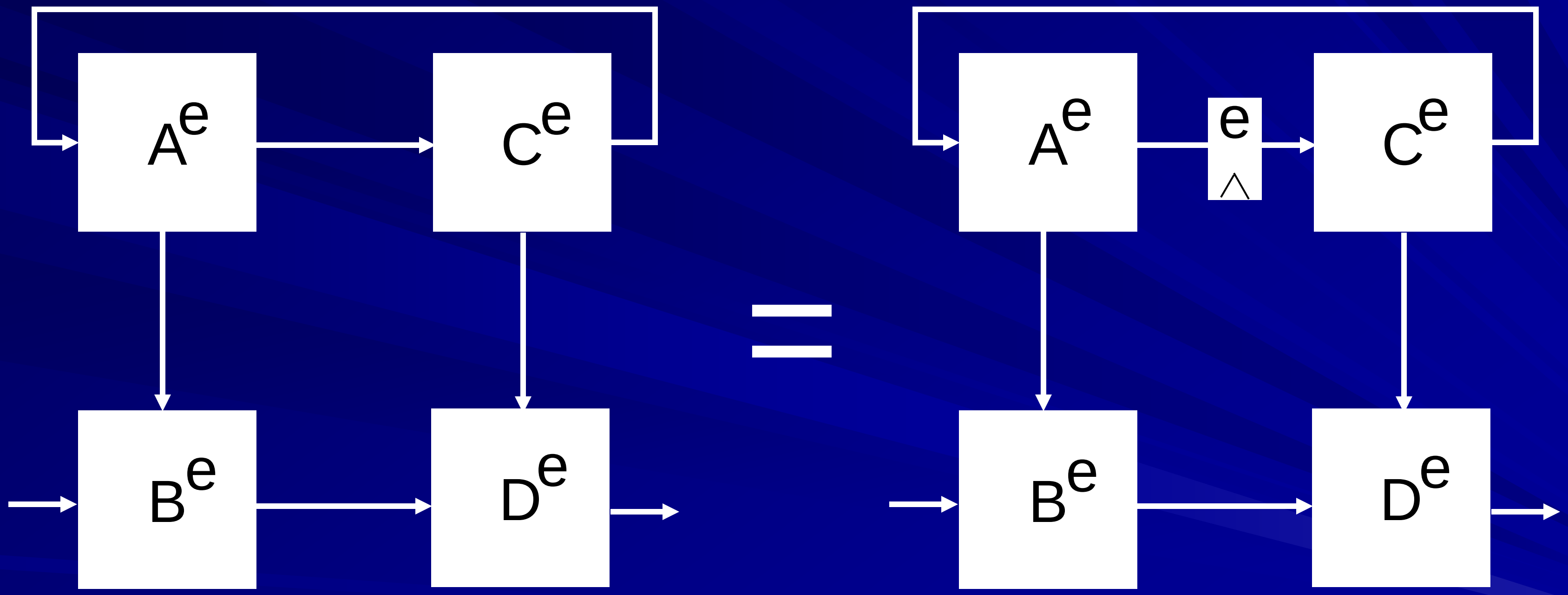
Latency can vary

Ordinary Synchronous System



Changing latencies changes behavior

Synchronous Elastic (characteristic property)



Changing latencies does NOT change behavior
= time elasticity

Elasticity?

- Elasticity refers to elasticity of time, i.e. tolerance to changes in timing parameters, not properties of materials
- Luca Carloni et al. in the first systematic study of such systems called them Latency Insensitive Systems

  Other used names:
  - Latency tolerant systems
  - Synchronous emulation of asynchronous systems
  - Synchronous handshake circuits
- We use term "synchronous elastic" to link to asynchronous elastic systems that have been developed before

  e.g., David Muller's pipelines of late 1950s

  Ivan Sutherland's micro-pipelines 1989

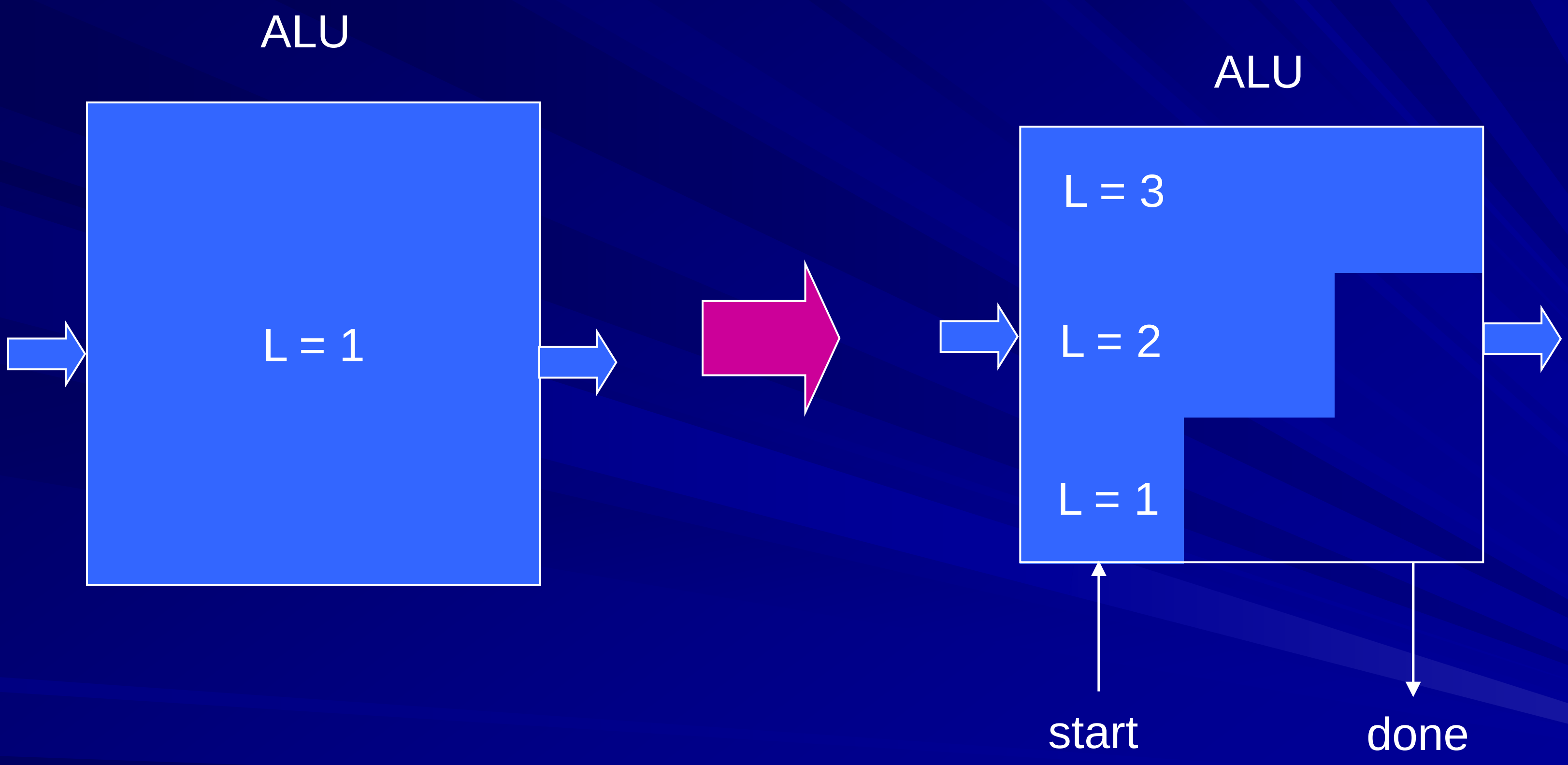
  Tolerate the variability of input data arrival and computation delays
- Asynchronous elastic tolerate changes in continuous time

Why

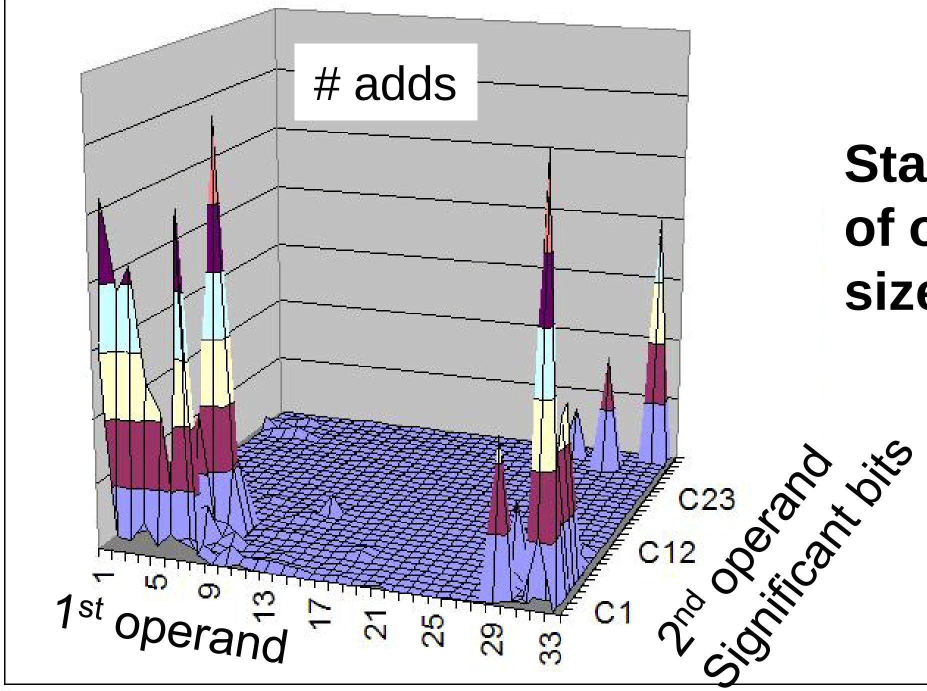
- Scalable
- Modular (Plug & Play)
- Potential for better energy-delay trade-offs
  - design for typical case instead of worst case
  - can separate performance critical parts from non-critical and optimize in isolation
- New micro-architectural opportunities in digital design
- Not asynchronous: use existing design experience, CAD tools and flows... but have some advantages of asynchronous

What can we do with synchronous elastic systems?

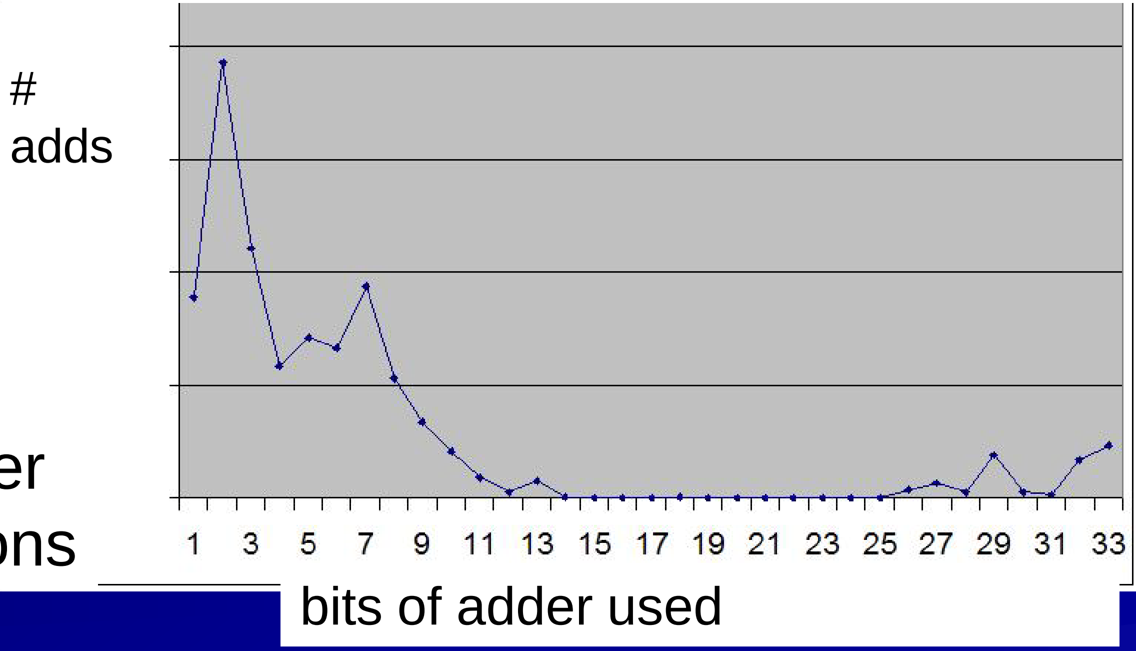
Variable latency units



Benchmark
“Patricia”
from
Media Bench

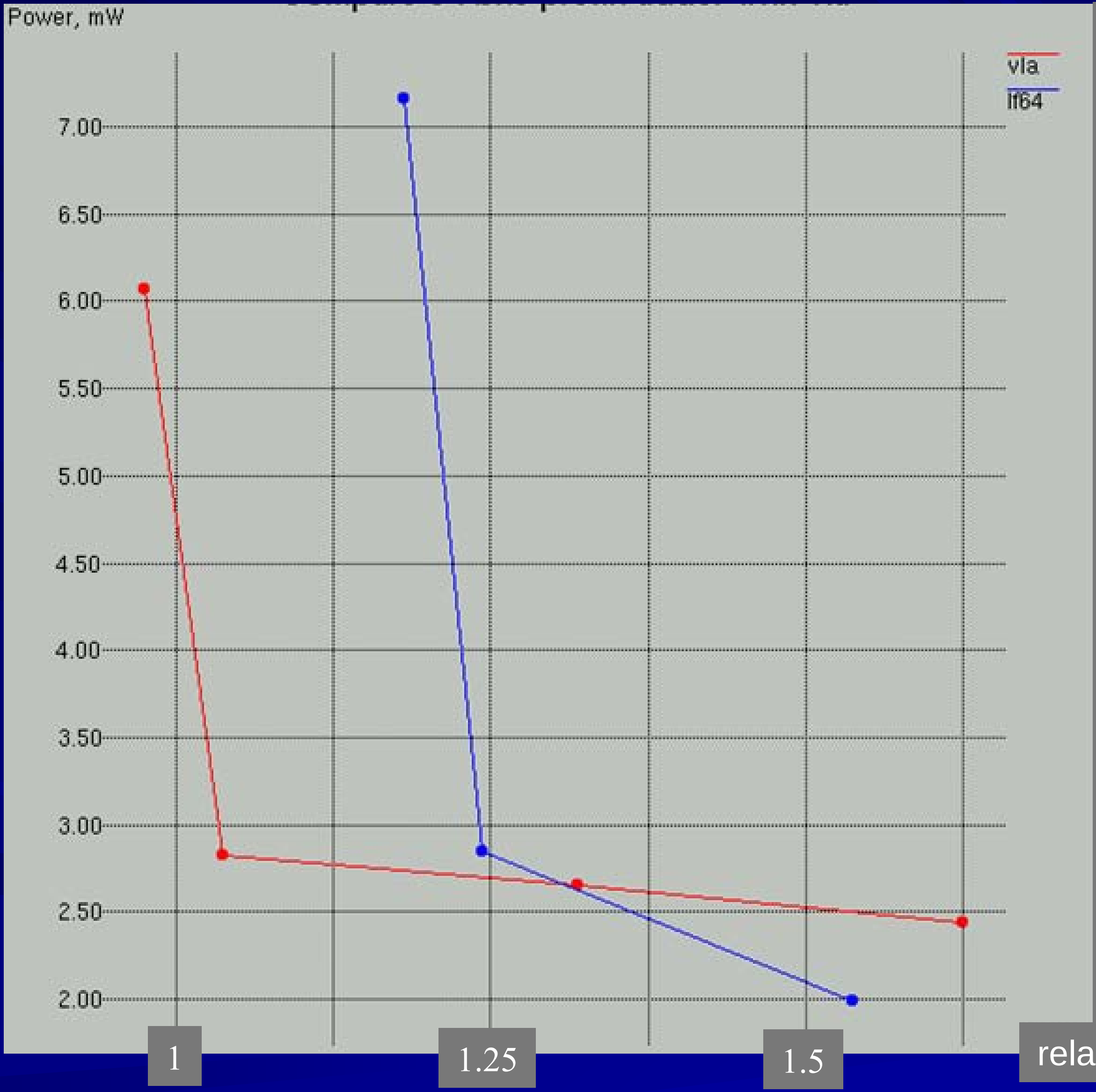


Statistics of operand sizes



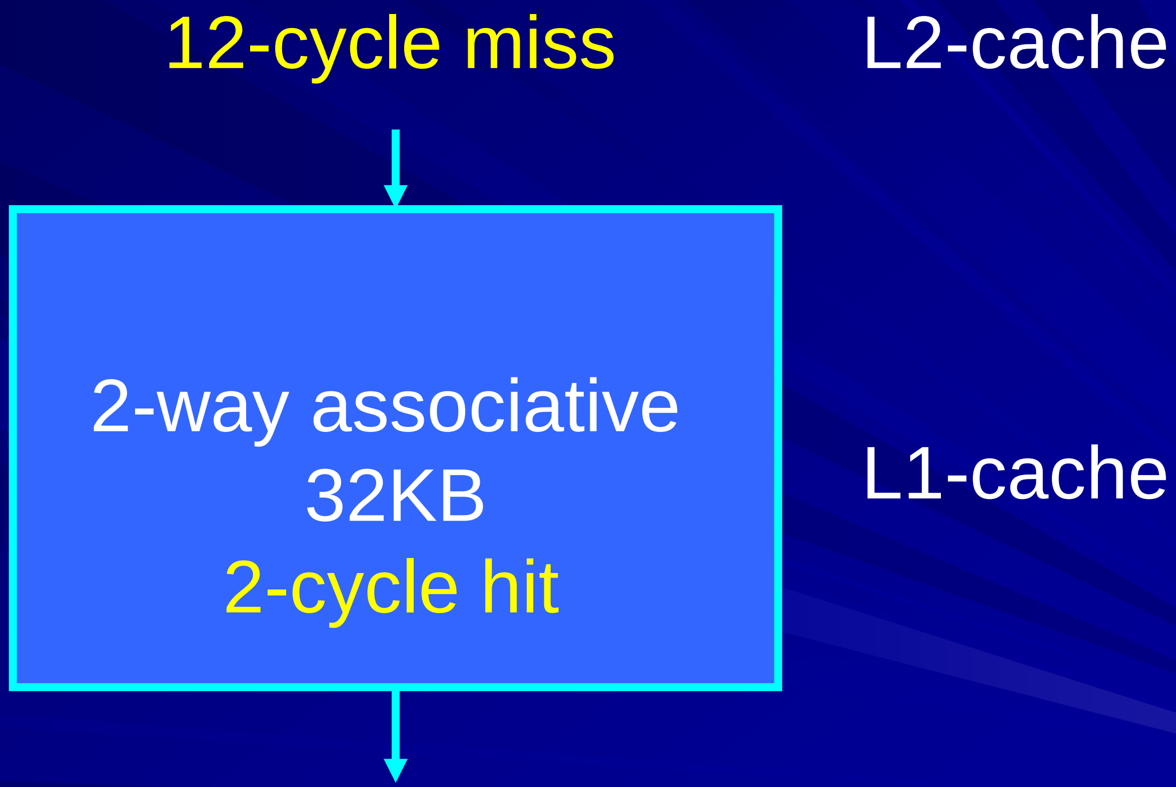
12 bits of an adder
do 95% of additions

Power-delay for an adder



Compare 64 bits VLA and prefix adder

Variable-latency cache hits



12-cycle miss

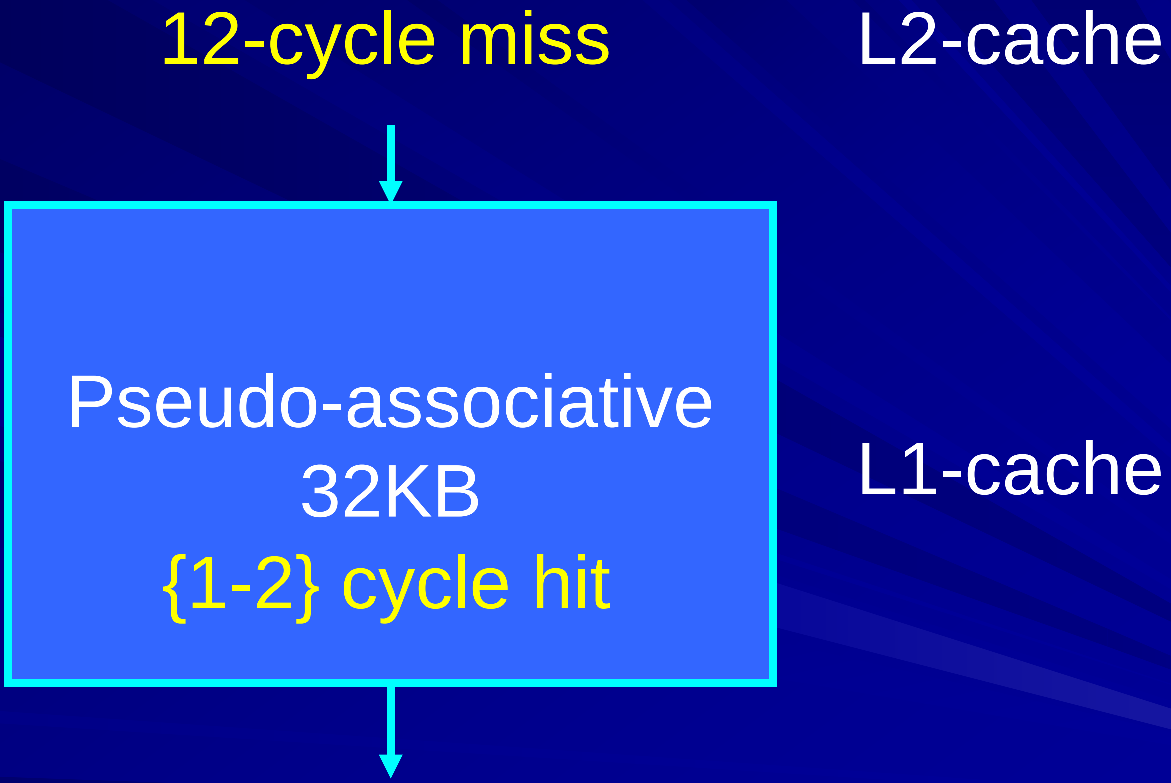
L2-cache

2-way associative
32KB
2-cycle hit

L1-cache

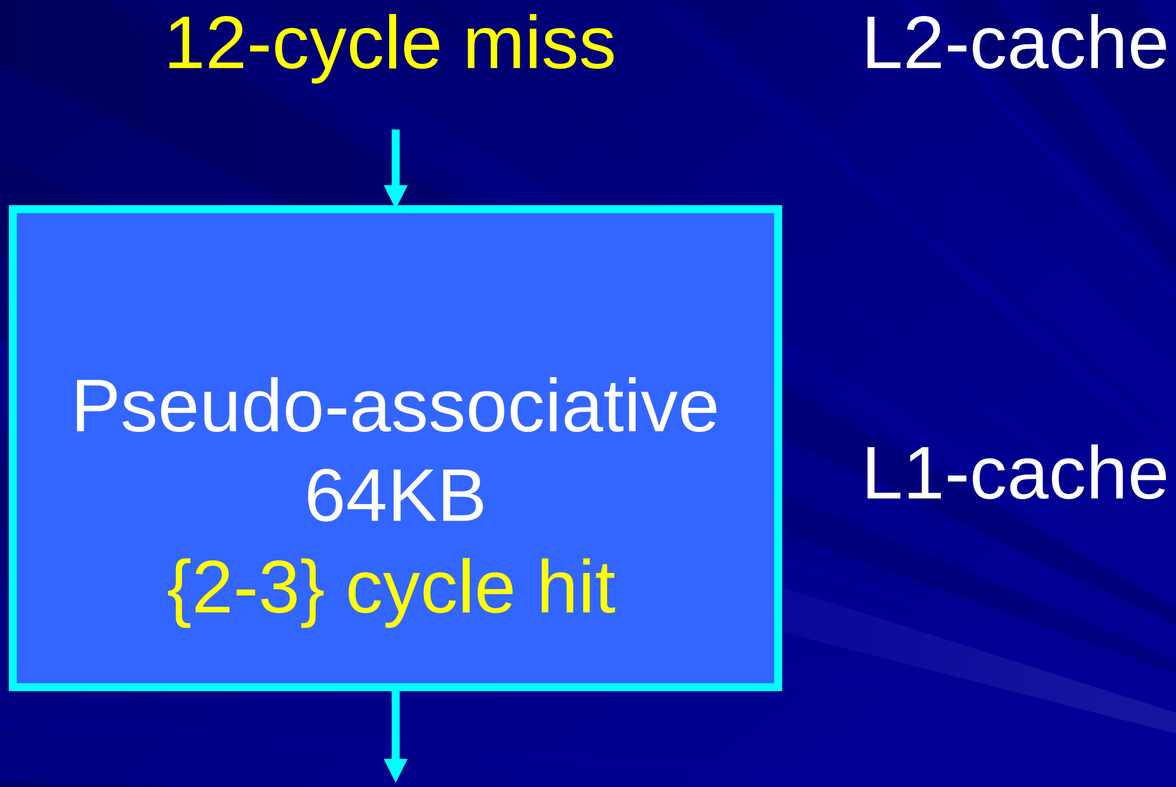
suggested by Joel Emer for ASIM experiment

Variable-latency cache hits



Sequential access: if hit in first access L = 1, if not – L=2
Trade-off: faster, or larger, or less power cache

Variable-latency cache hits



12-cycle miss

L2-cache

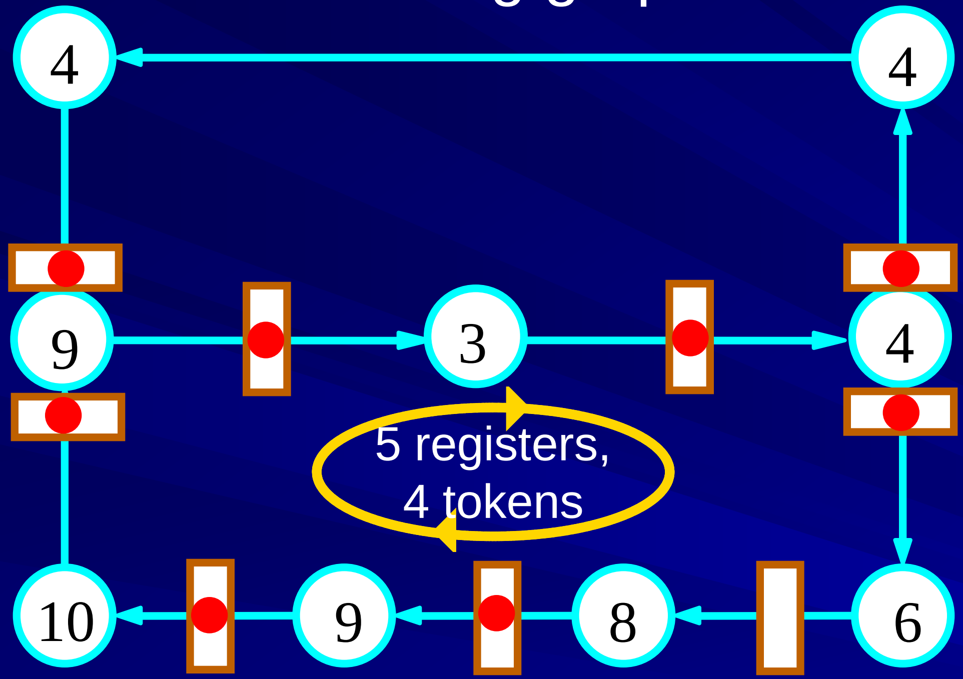
Pseudo-associative
64KB
{2-3} cycle hit

L1-cache

Sequential access: if hit in first access L = 1, if not – L=2
Trade-off: faster, or larger, or less power cache

Motivation example

Retiming graph



10 - Combinational block with delay 10

● - Initialized register (dot)

Cycle time is 28 18

Throughput is 1 4/5

Effective cycle time is 20 18

The longest combinational path delay

The cycle time/throughput cycle

Retiming can not do better!

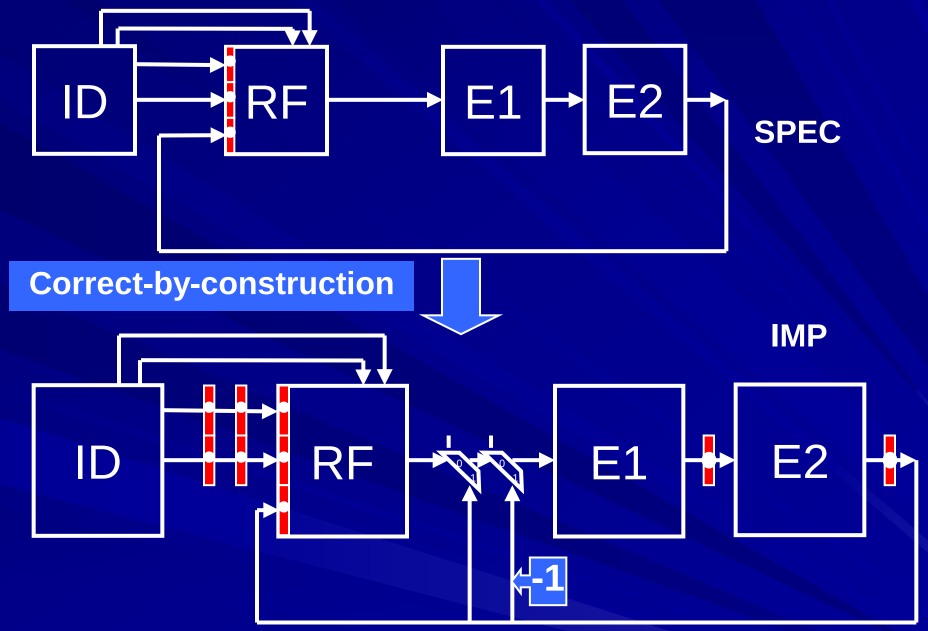
Finding minimal effective cycle time of the circuit represented as retiming graph (RG)!

Retiming and Recycling (R&R) can

Correct-by-construction automatic pipelining in presence of iteration dependencies

Transforms:

- bypass
- retiming
- elasticize
- early enabling
- insert buffers
  and negative tokens
- size elastic buffer capacity



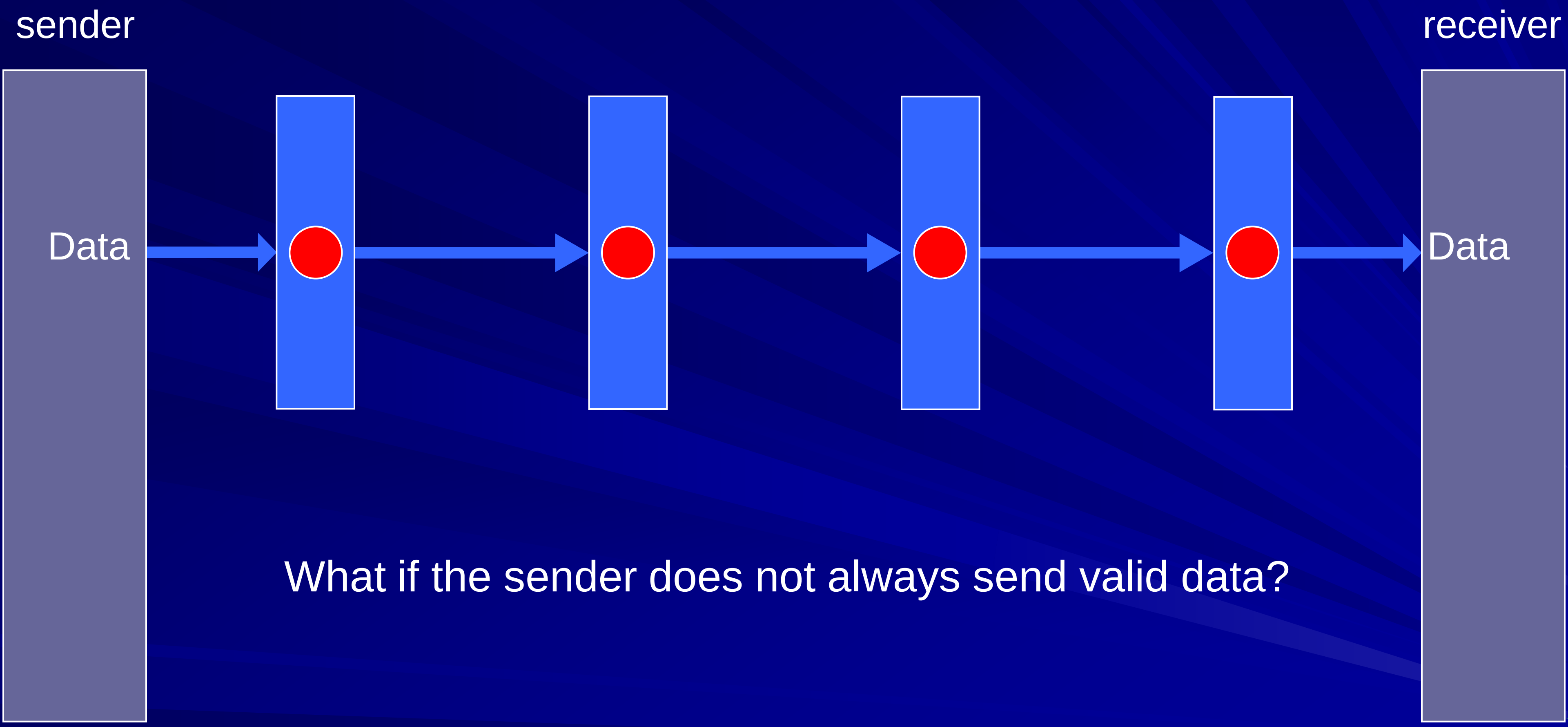
and correct-by-construction speculation

How to Design Synchronous Elastic Systems

- Example of the implementation:
  SELF = Synchronous Elastic Flow

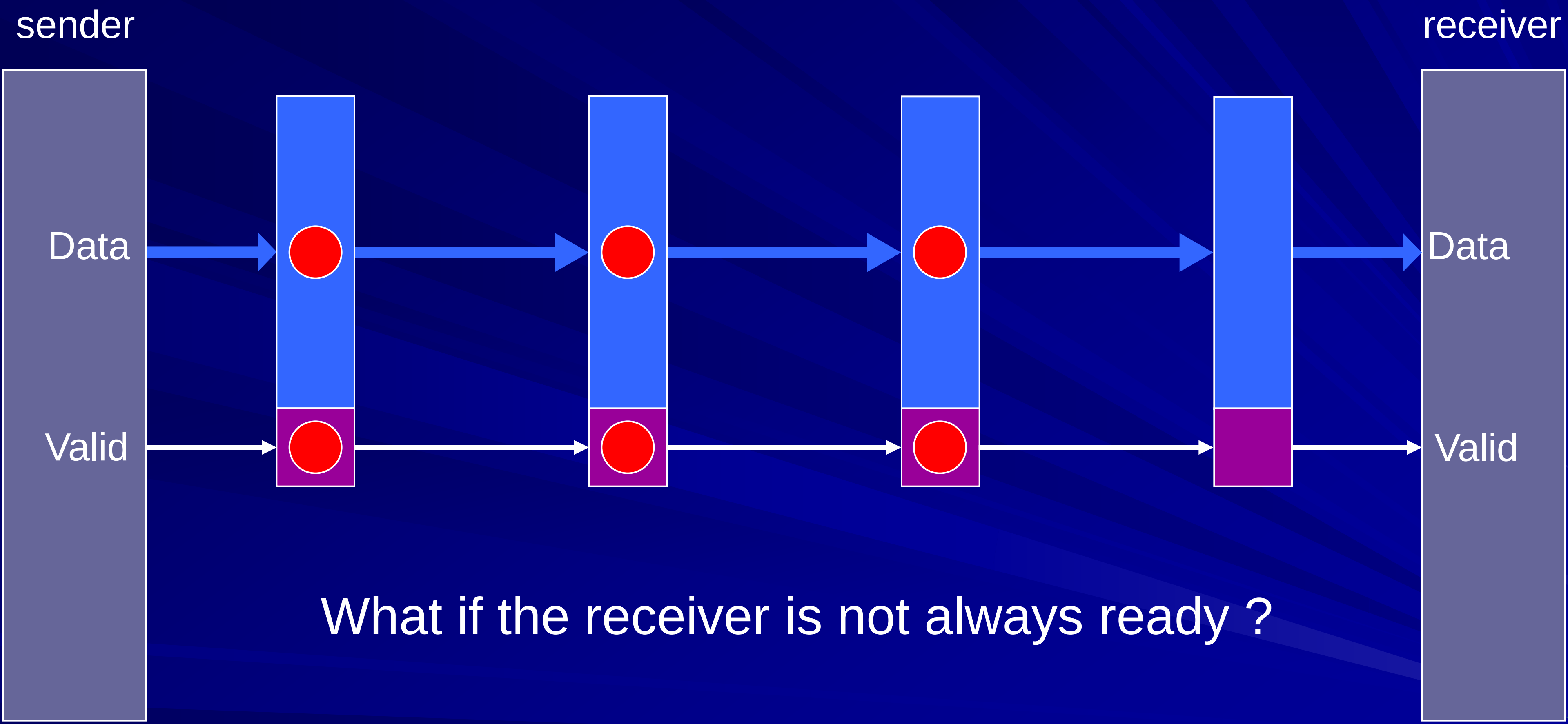
- Other implementations are possible

Pipelined communication



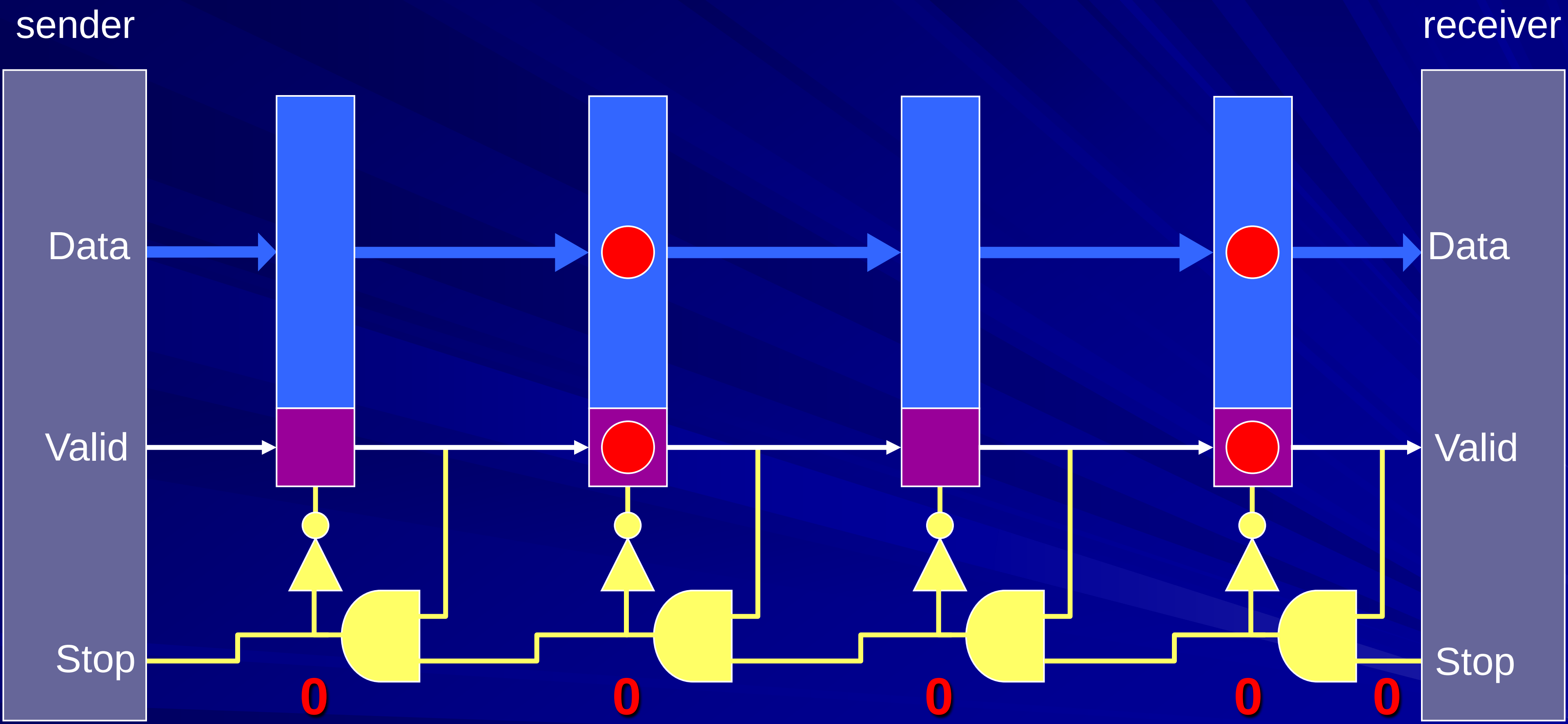
What if the sender does not always send valid data?

The Valid bit

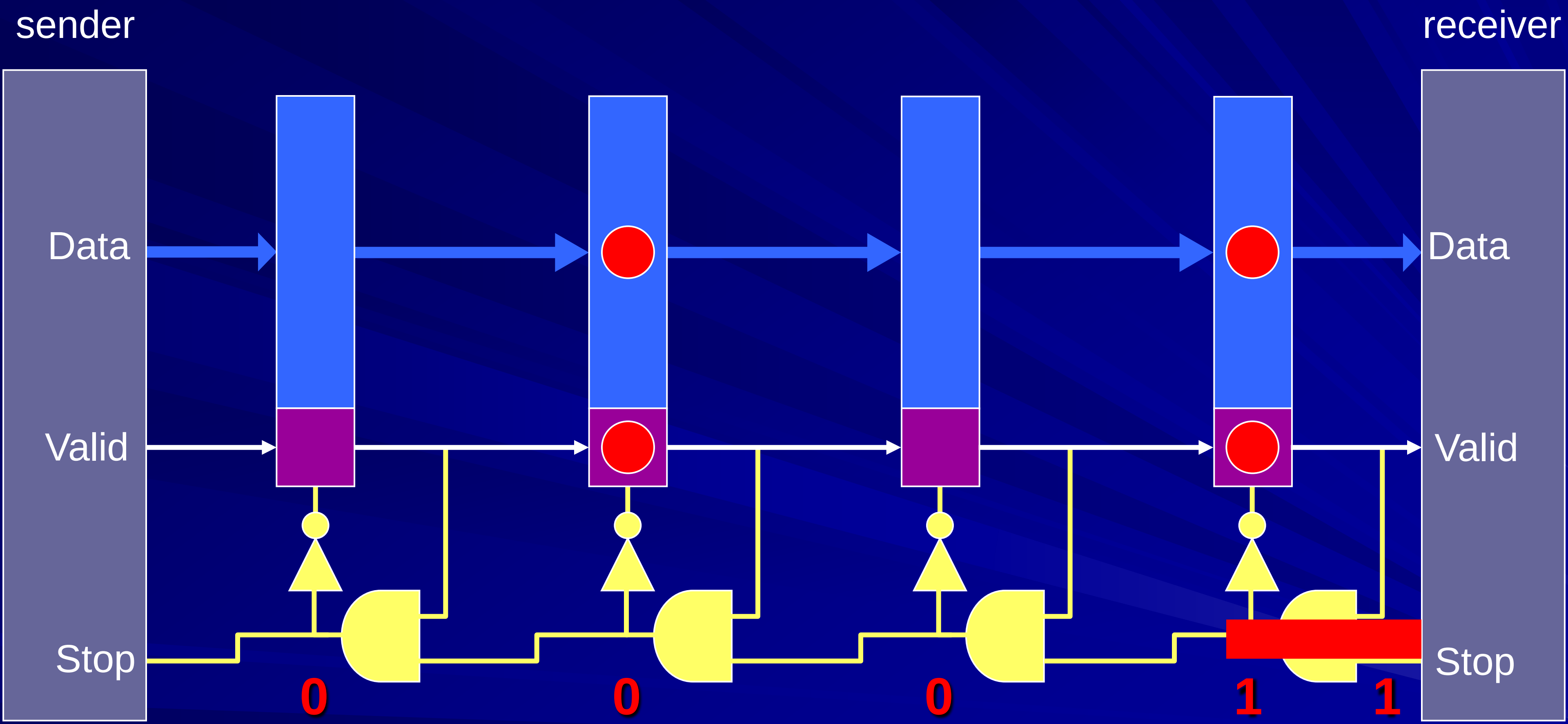


What if the receiver is not always ready ?

The Stop bit



The Stop bit



sender

receiver

Data

Valid

Stop

0

0

0

1

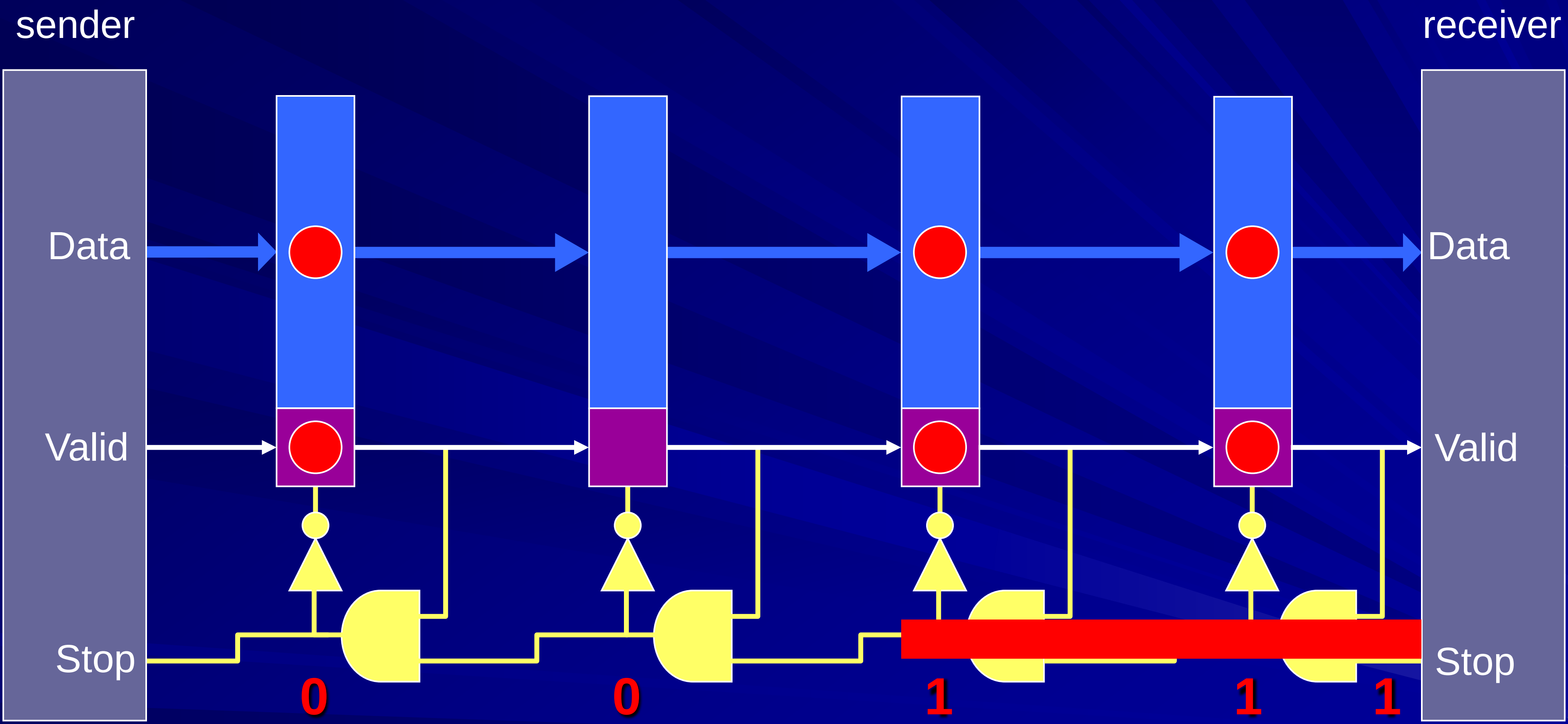
1

Data

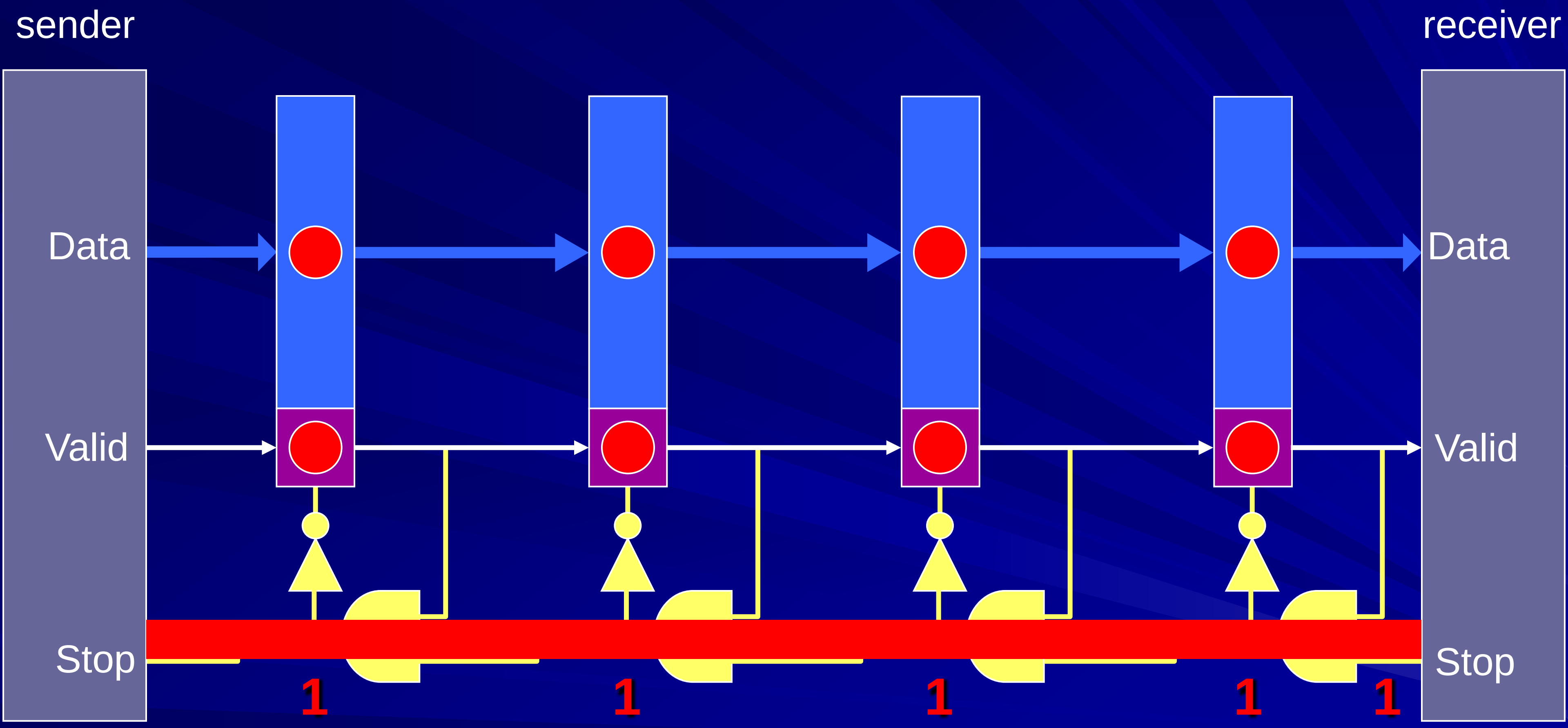
Valid

Stop

The Stop bit

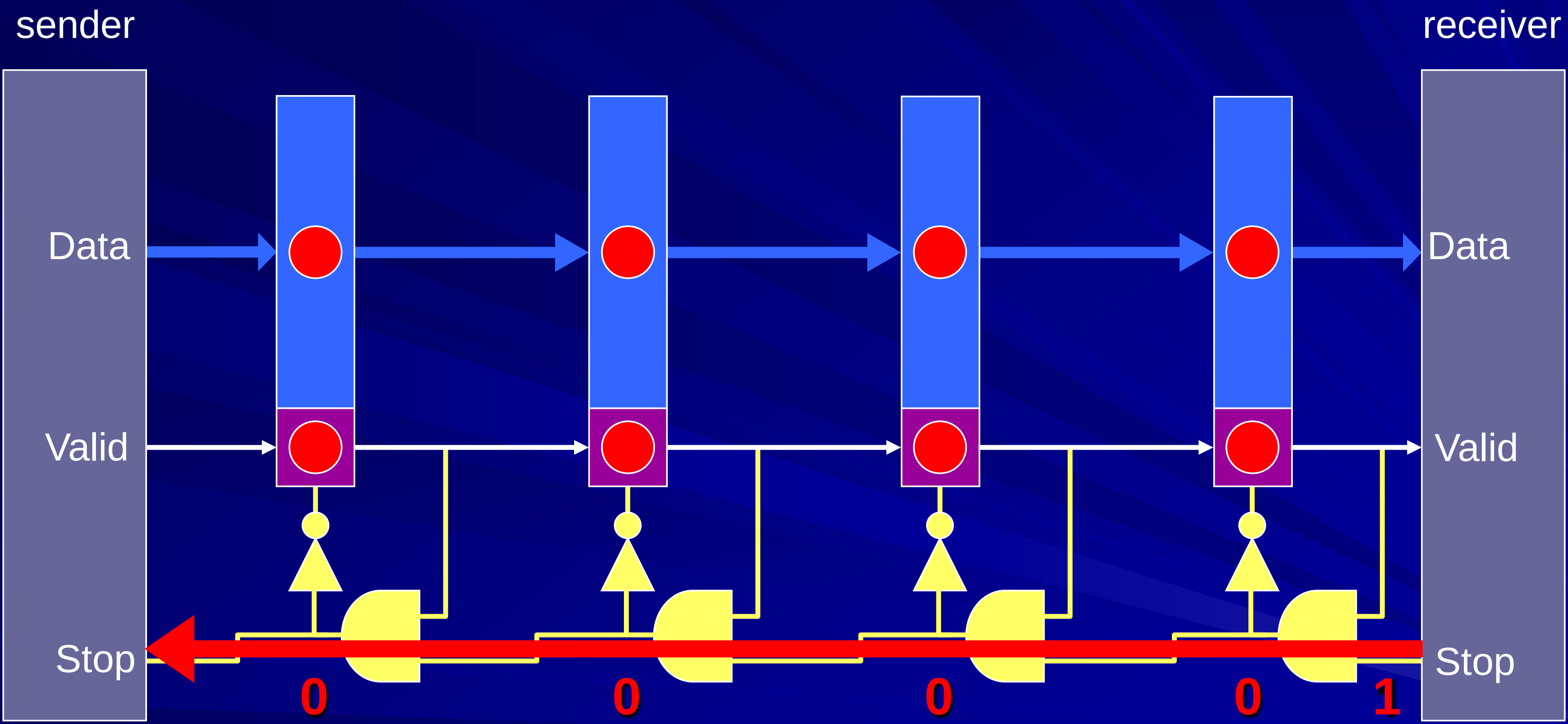


The Stop bit



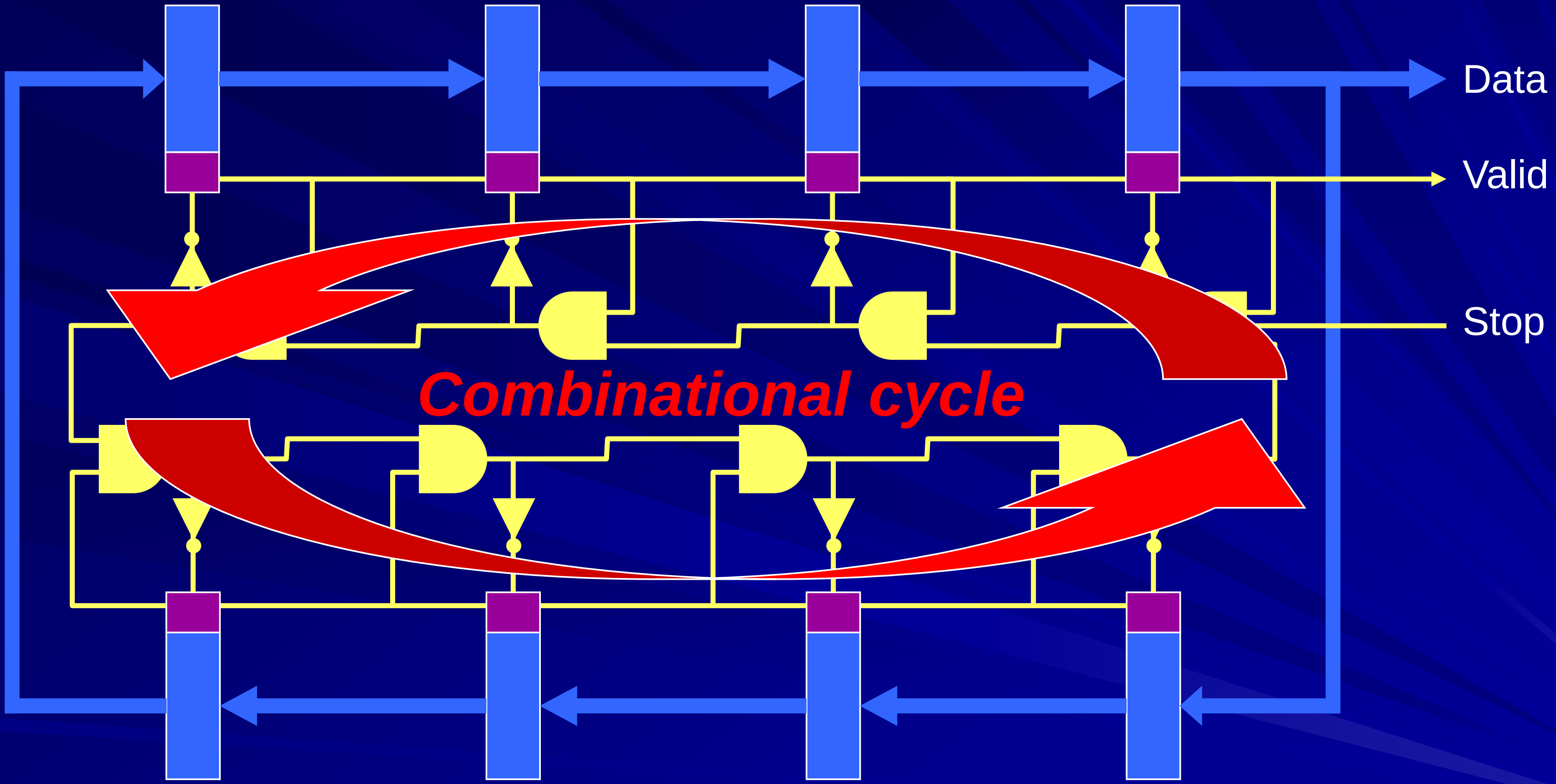
Back-pressure

The Stop bit



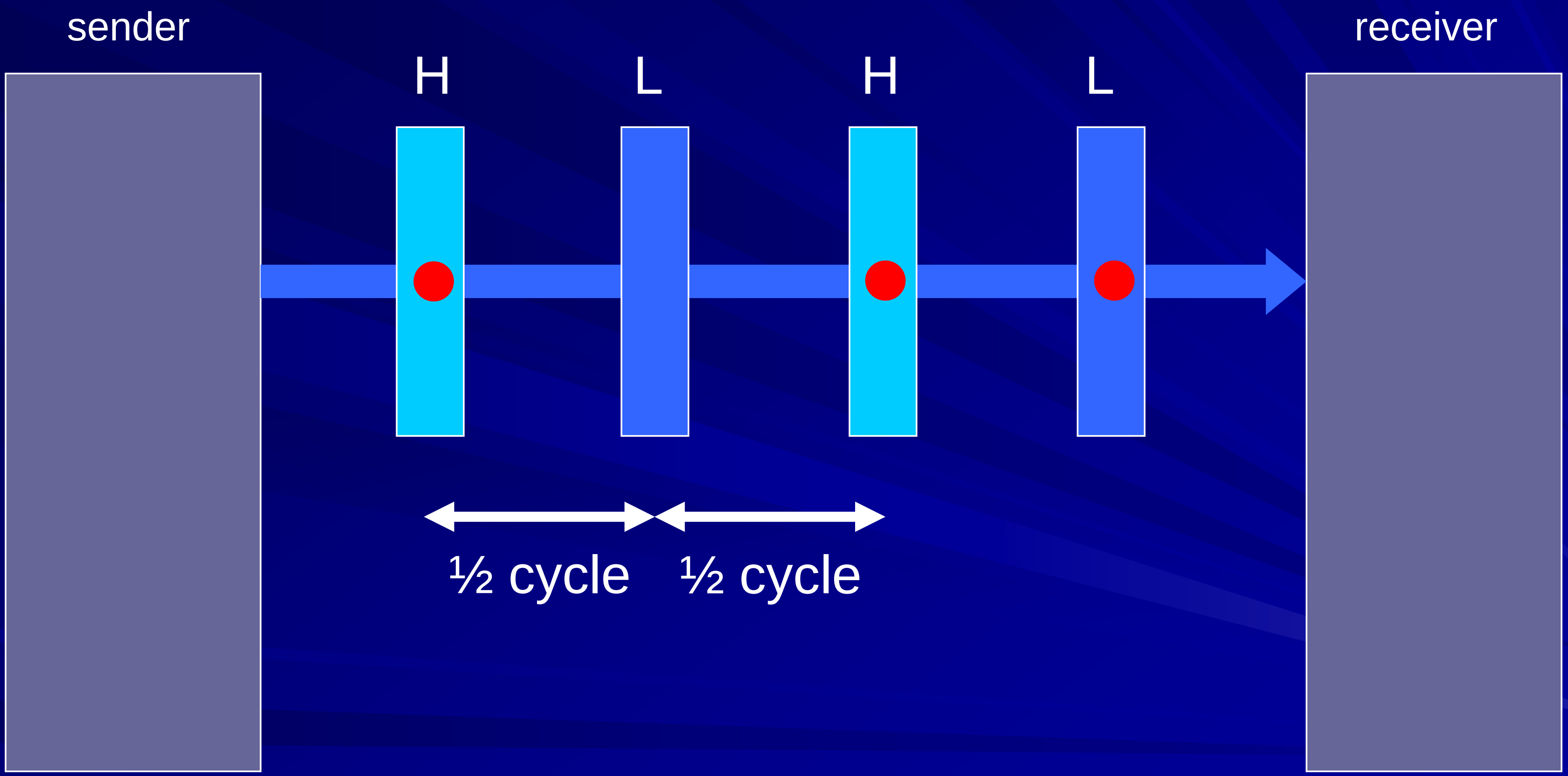
Long combinational path

Cyclic structures



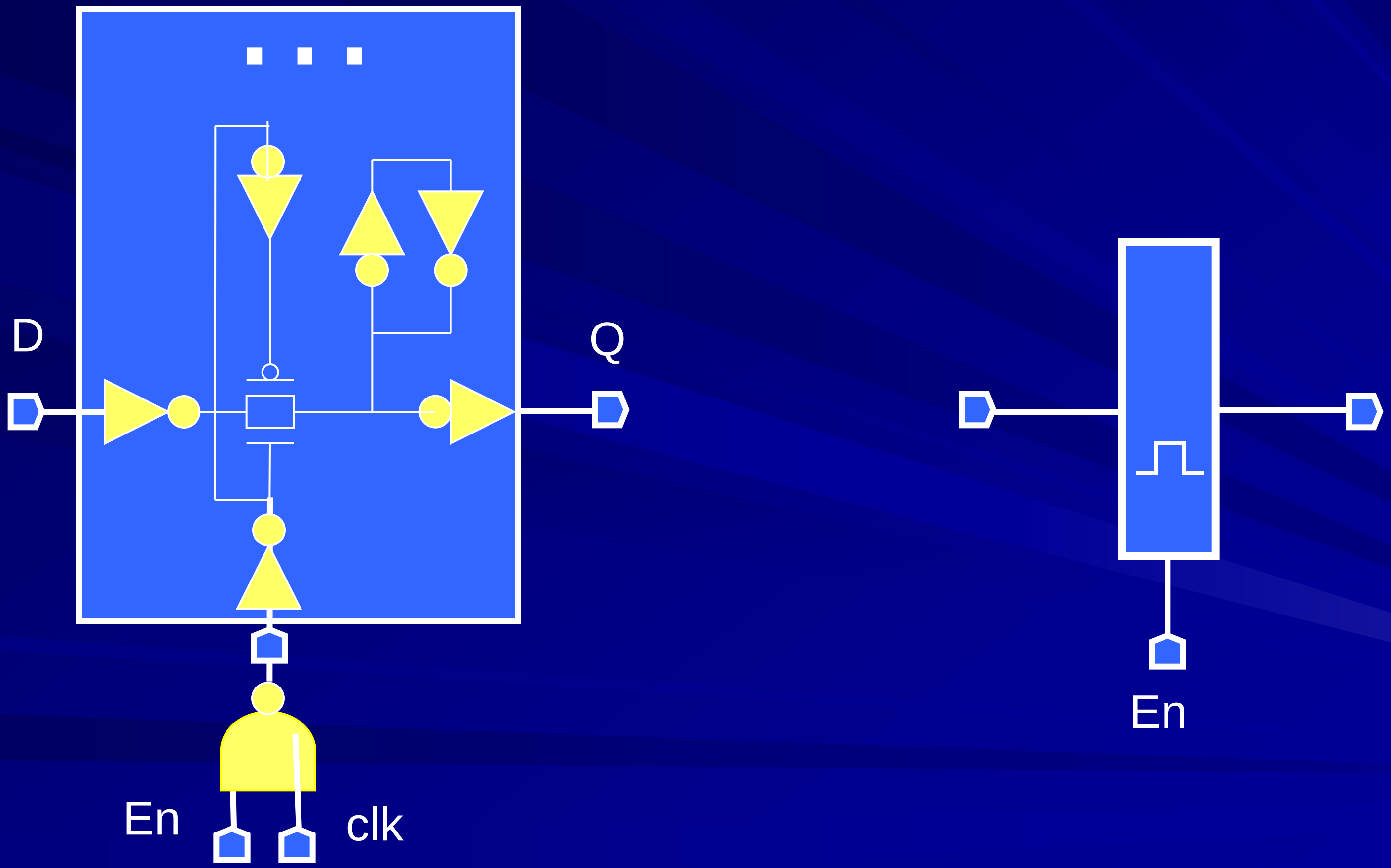
One can build circuits with combinational cycles (constructive cycles by Berry), but synthesis and timing tools do not like them

Example: pipelined linear communication chain with transparent latches

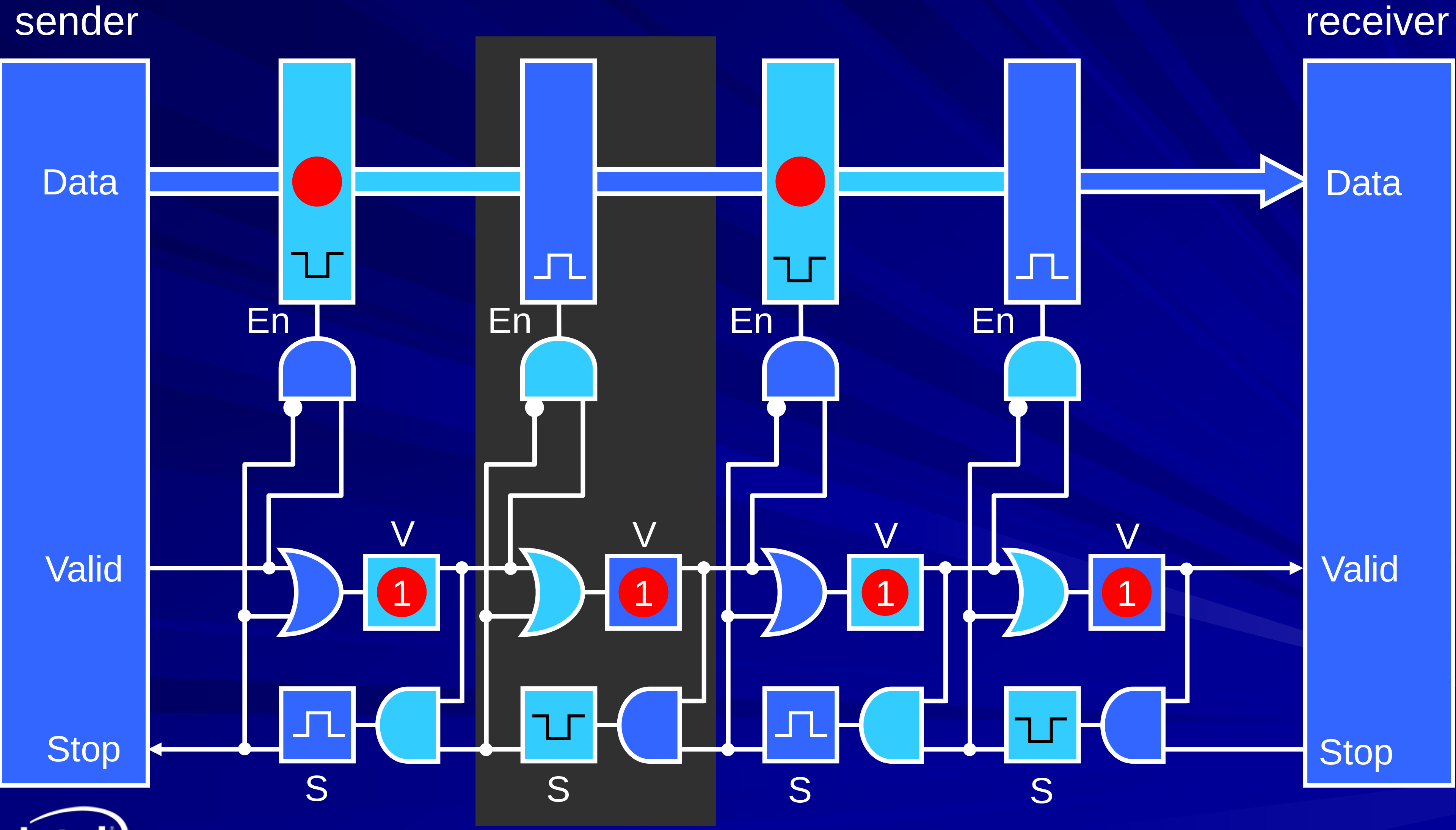


Master and slave latches with independent control

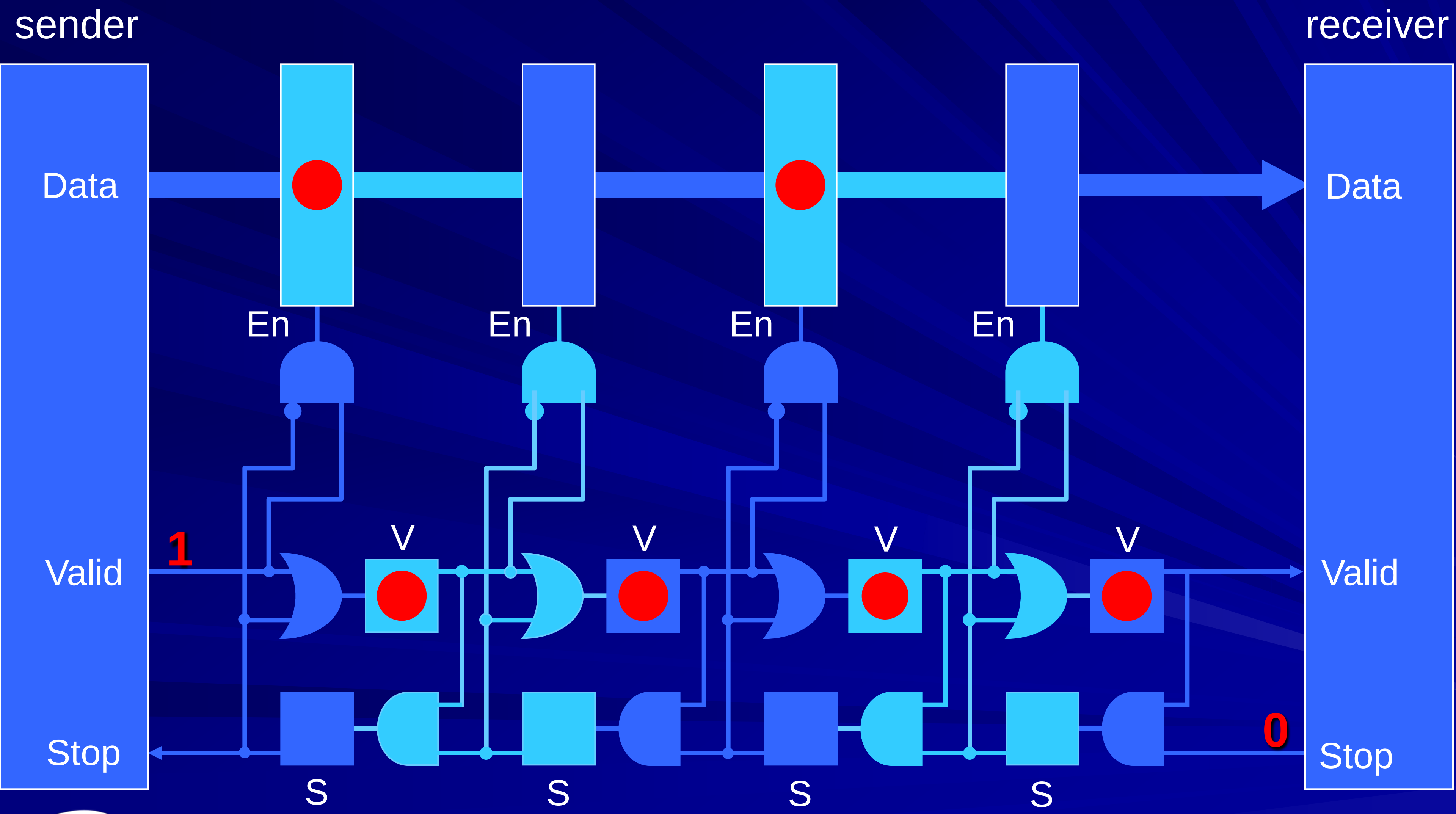
Shorthand notation (clock lines not shown)



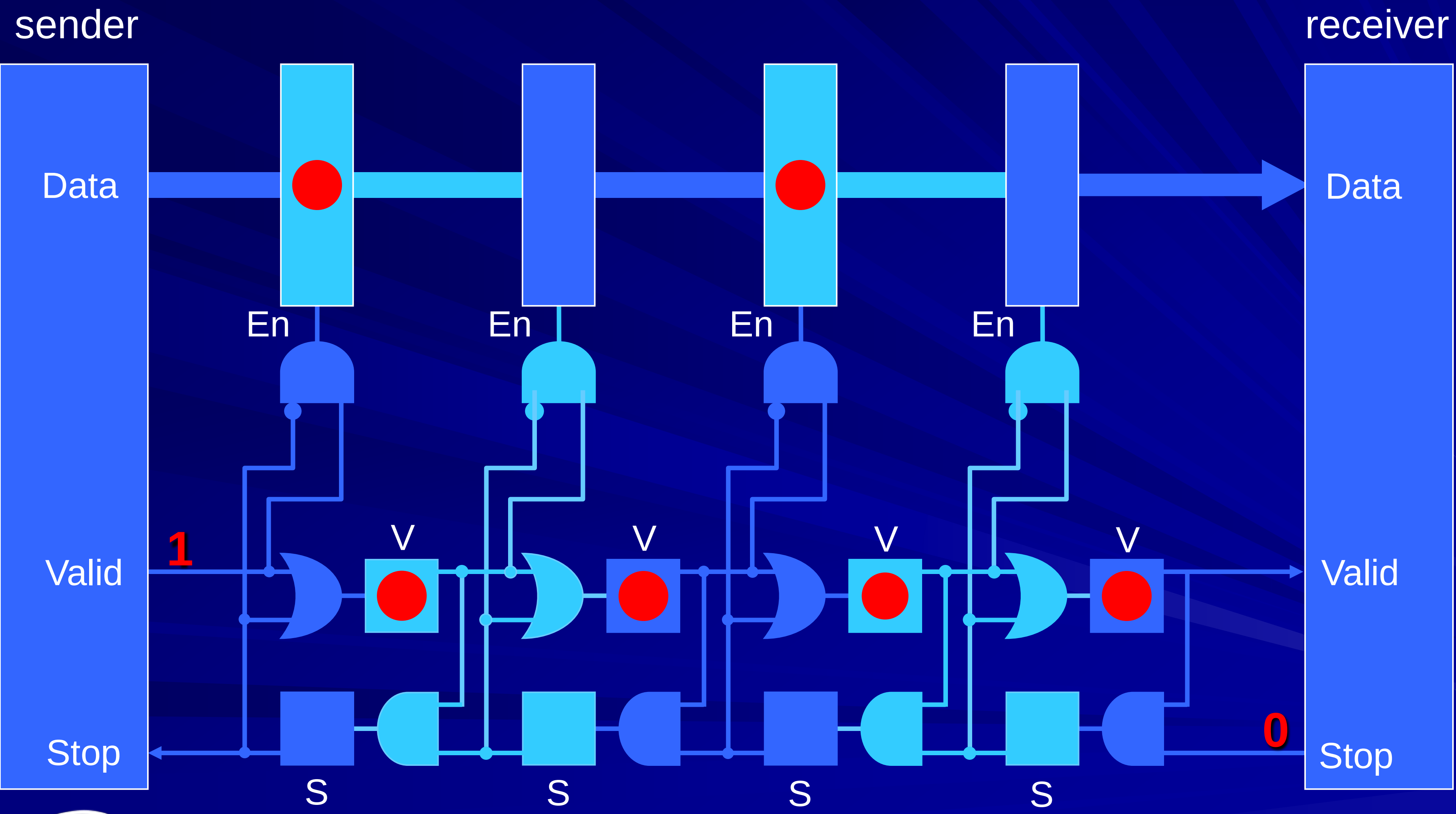
SELF (linear communication)



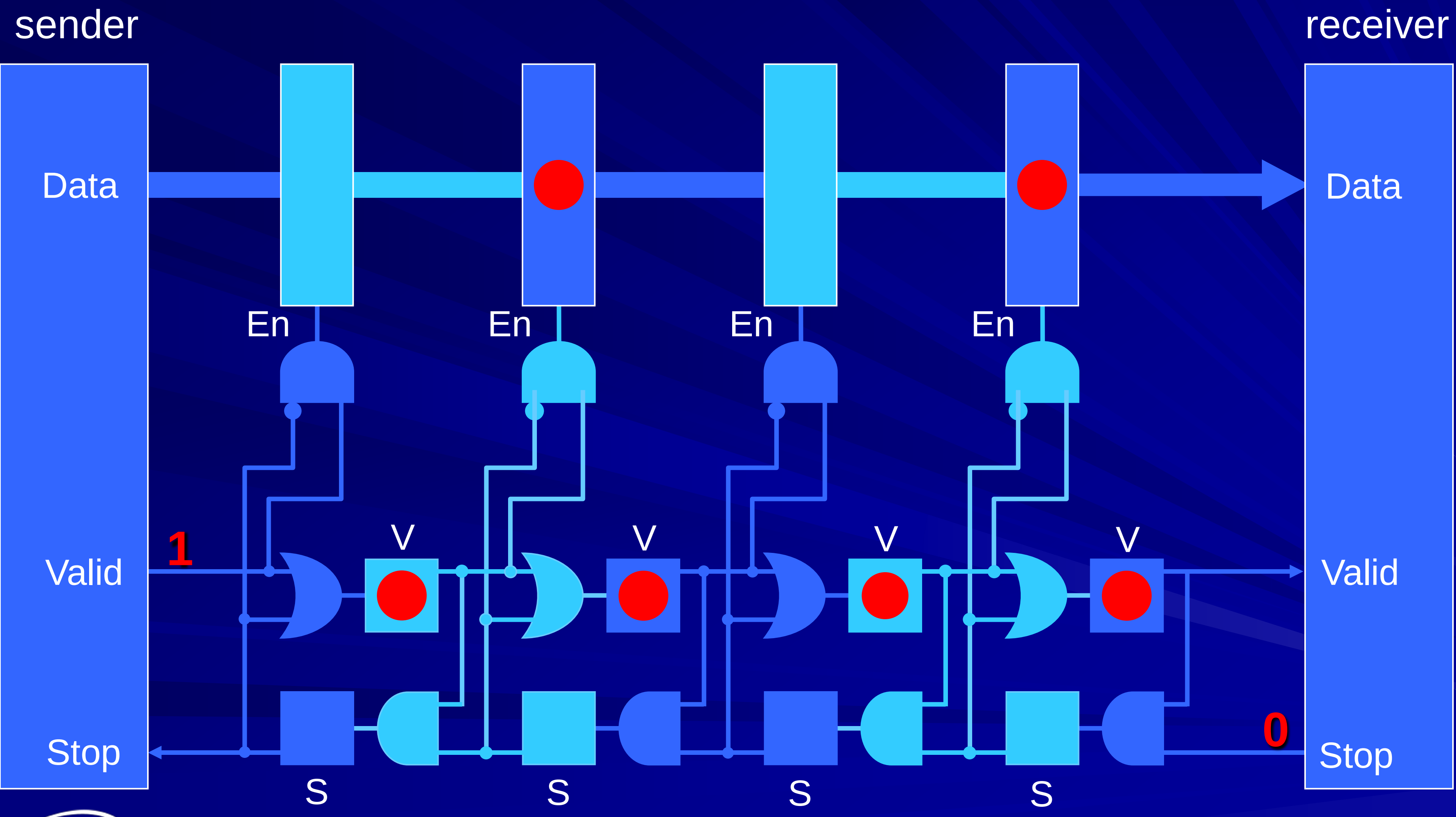
SELF


sender
receiver
Data
Data
En
En
En
En
1
Valid
V
V
V
V
Valid
0
Stop
Stop
S
S
S
S

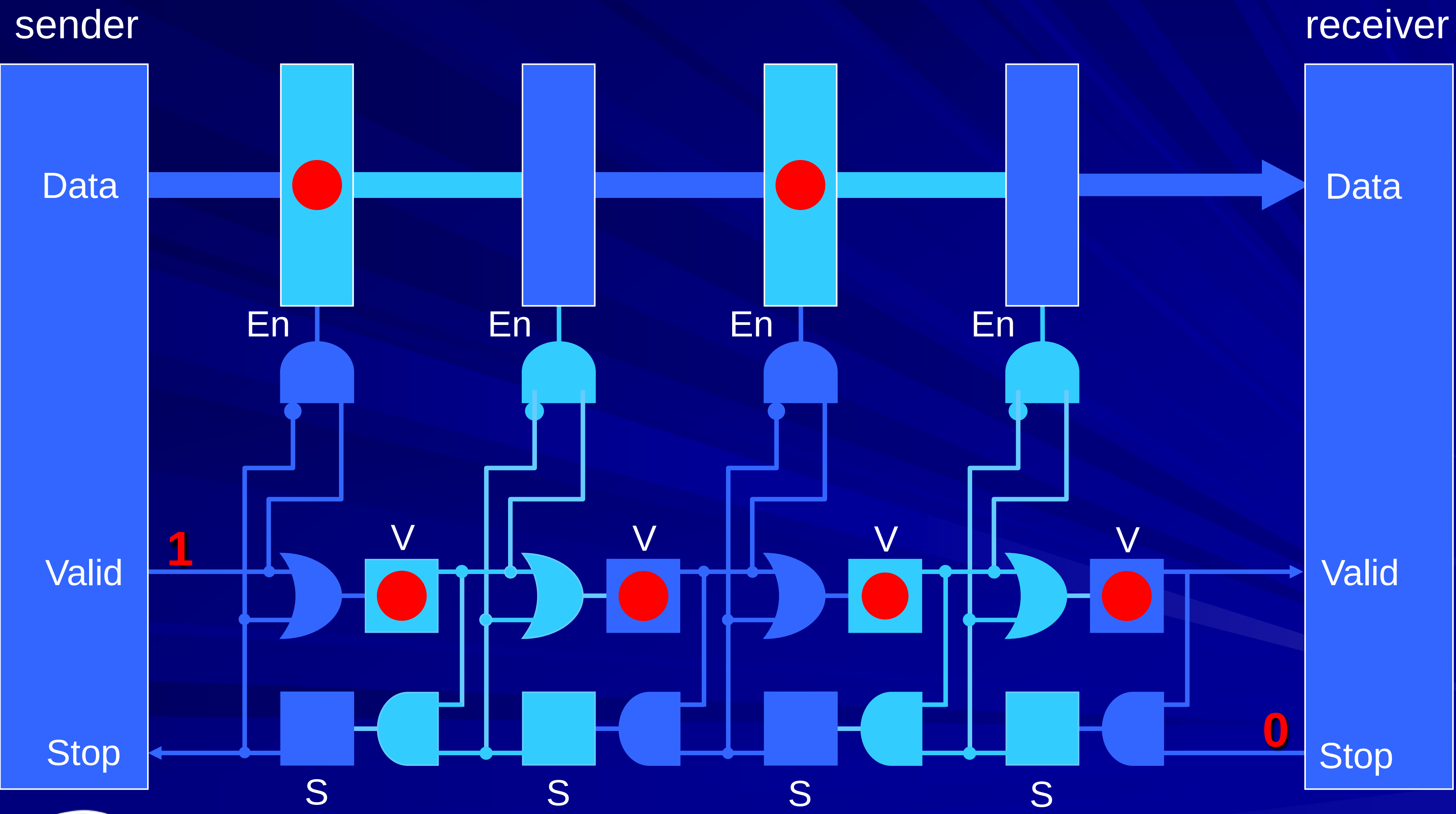
SELF


sender
receiver
Data
Data
En
En
En
En
1
Valid
V
V
V
V
Valid
0
Stop
Stop
S
S
S
S

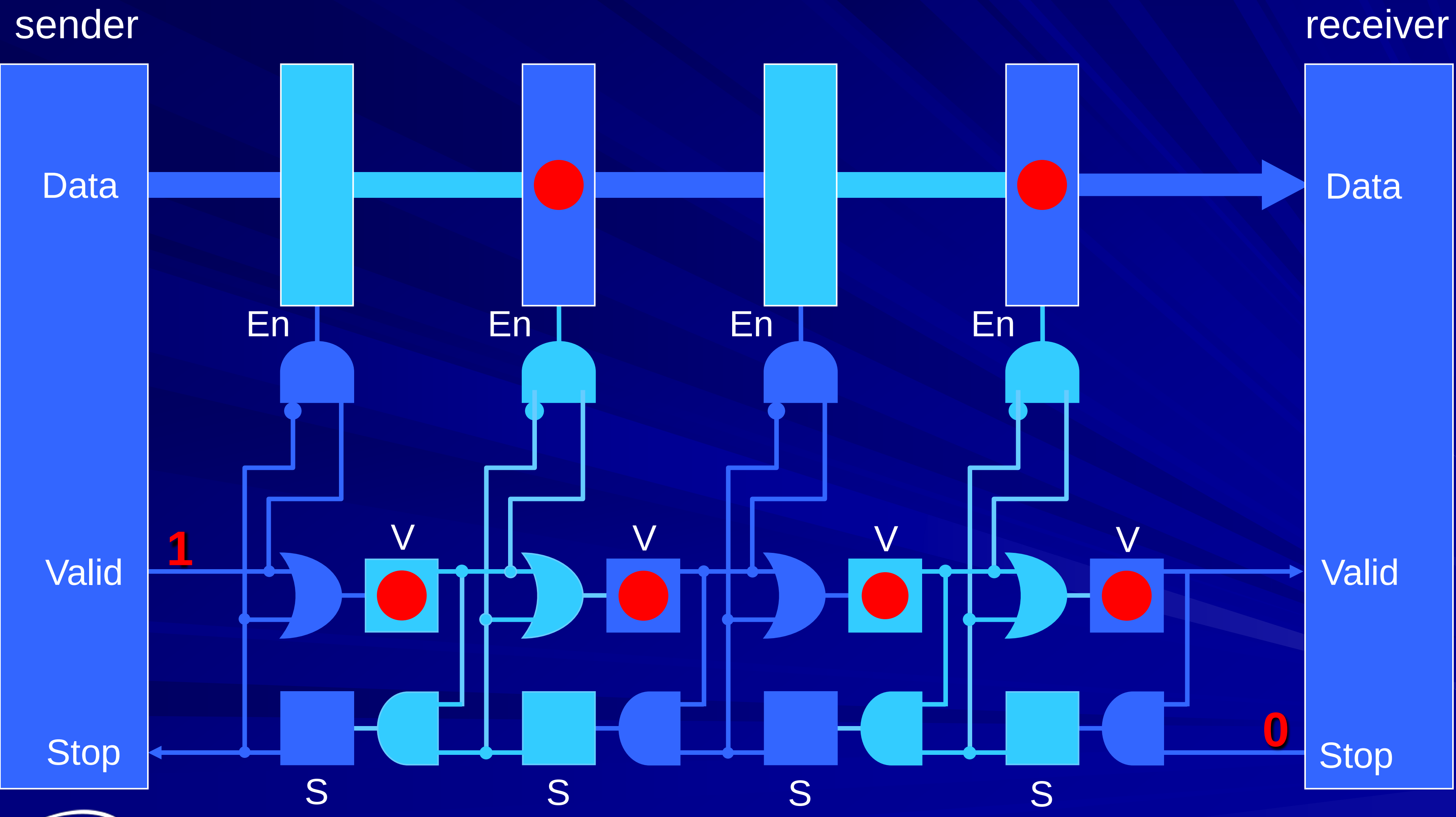
SELF


sender
receiver
Data
Data
En
En
En
En
1
V
V
V
V
Valid
Valid
0
Stop
Stop
S
S
S
S

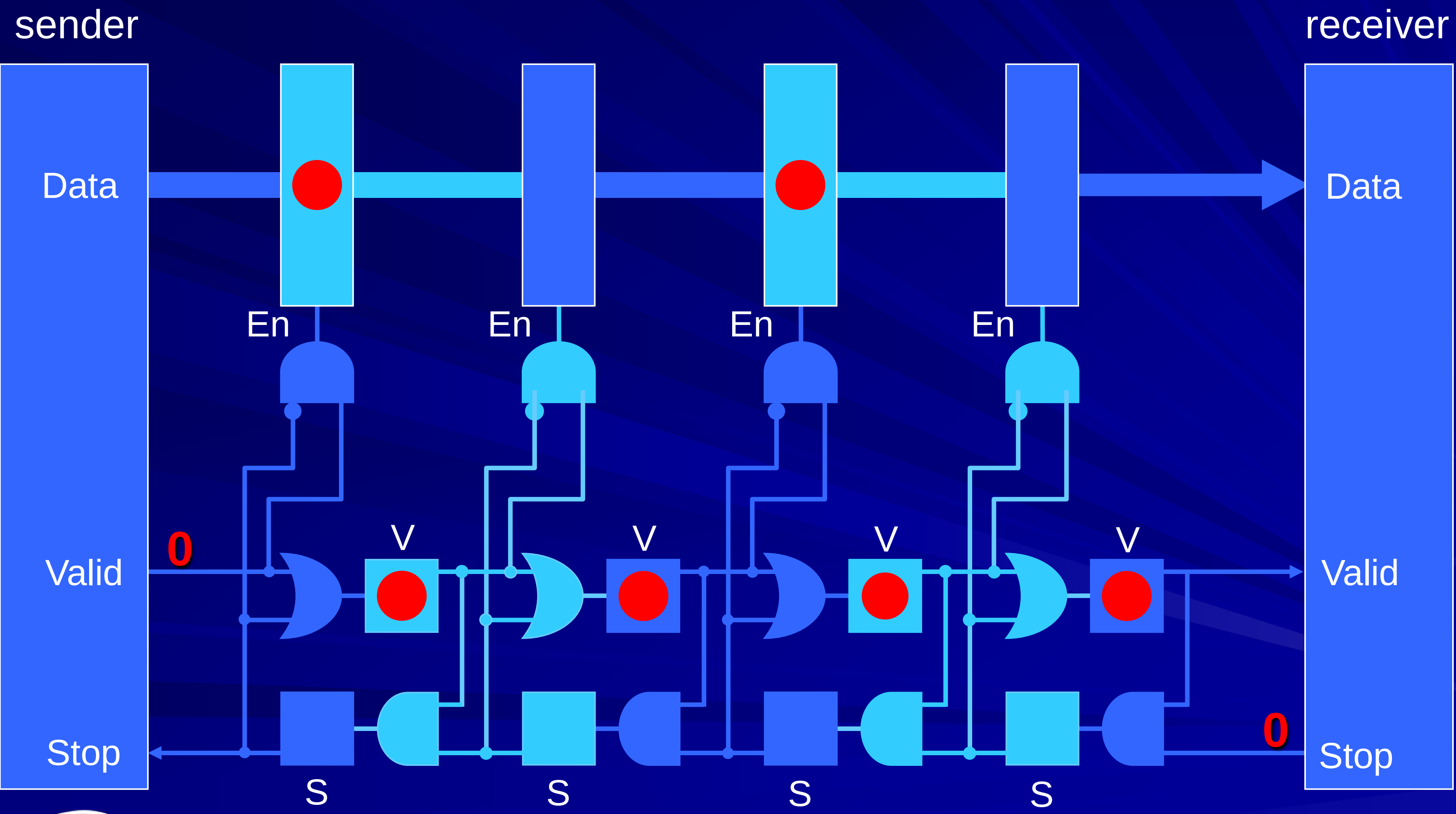
SELF


sender
receiver
Data
Data
En
En
En
En
1
V
V
V
V
Valid
Valid
0
Stop
Stop
S
S
S
S

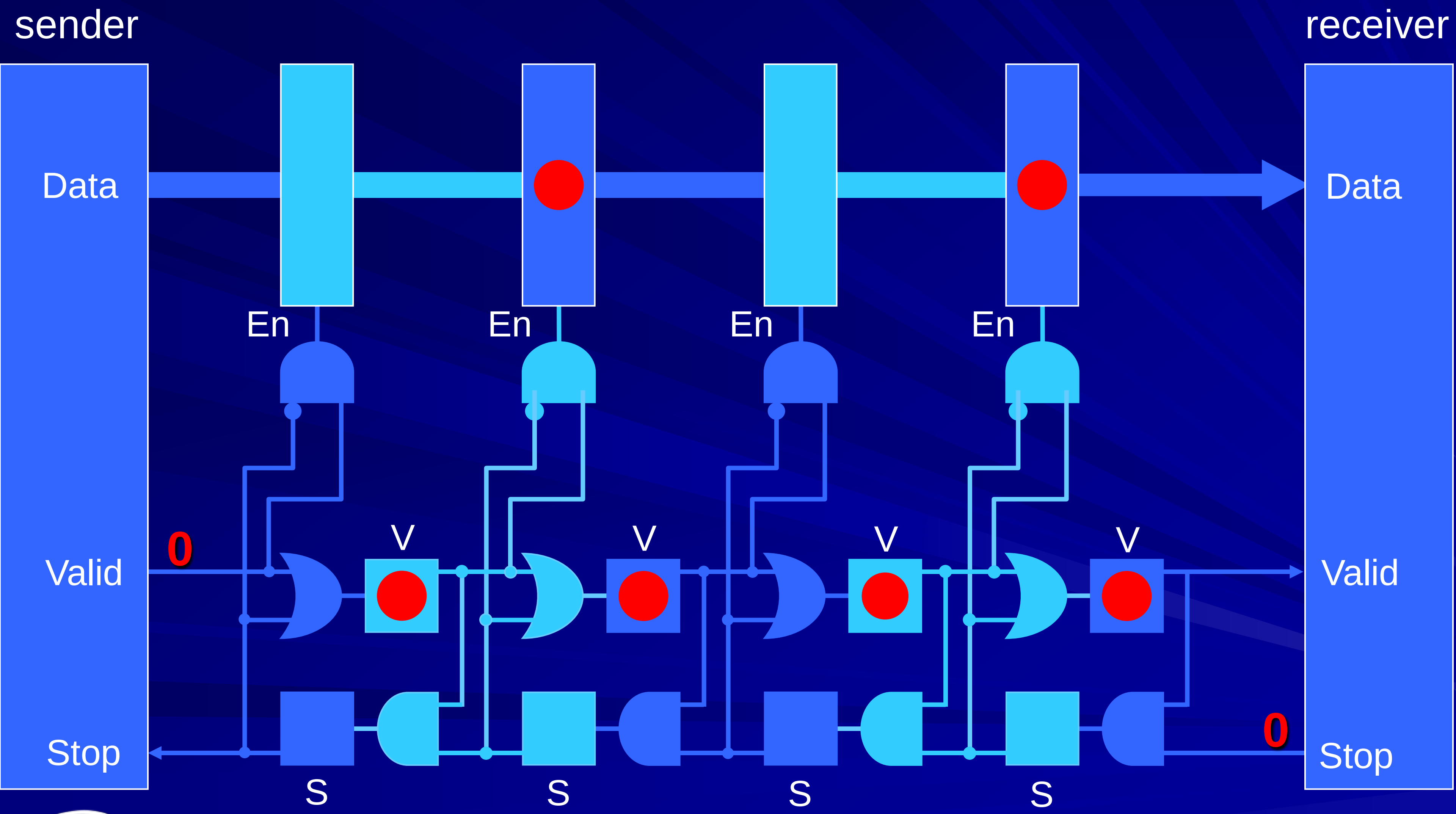
SELF


sender
receiver
Data
Data
En
En
En
En
V
V
V
V
1
Valid
Valid
0
Stop
Stop
S
S
S
S

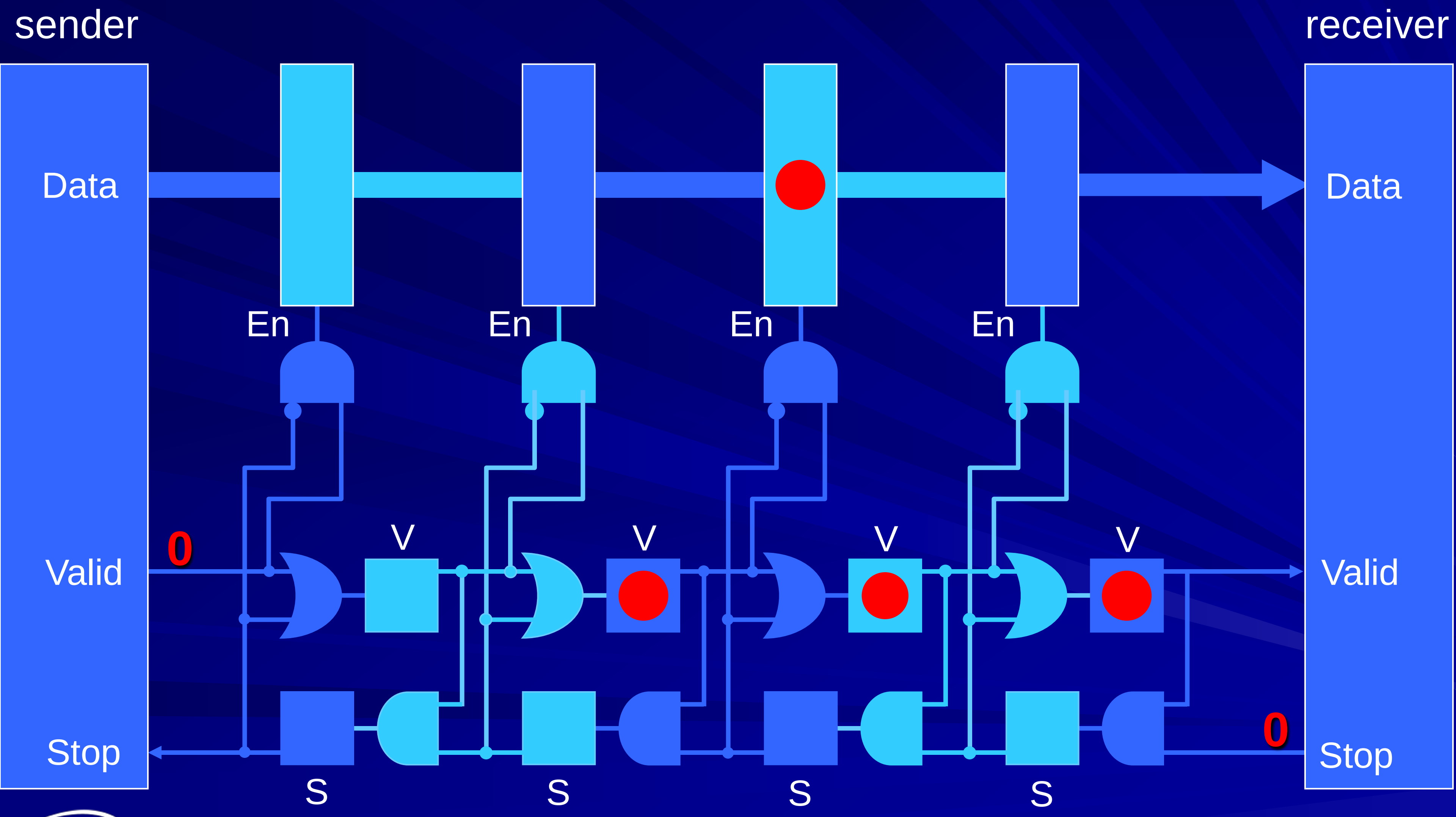
SELF


sender
receiver
Data
Data
En
En
En
En
V
V
V
V
0
Valid
Valid
0
Stop
Stop
S
S
S
S

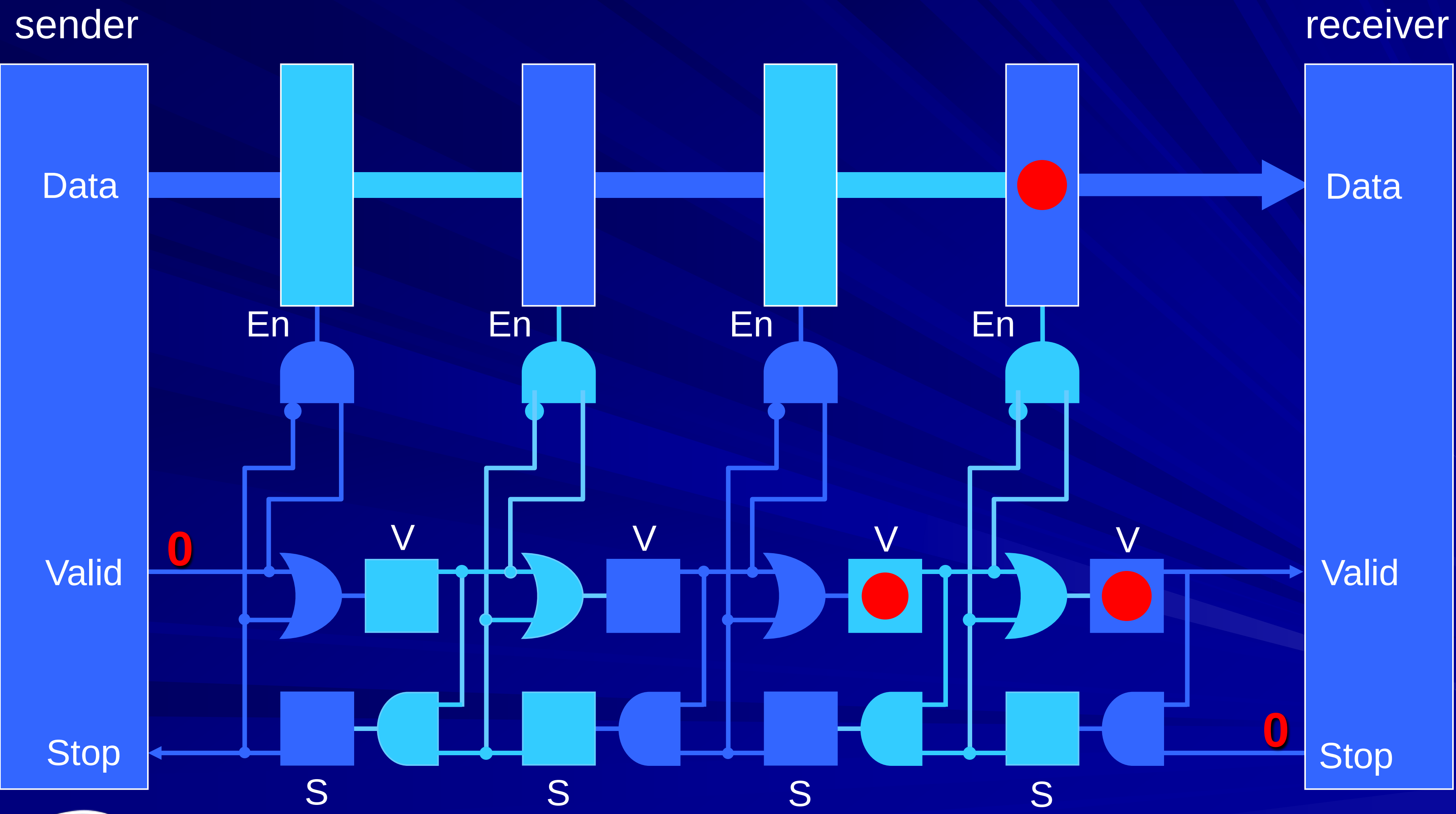
SELF


sender
receiver
Data
Data
En
En
En
En
0
Valid
V
V
V
V
Valid
0
Stop
Stop
S
S
S
S

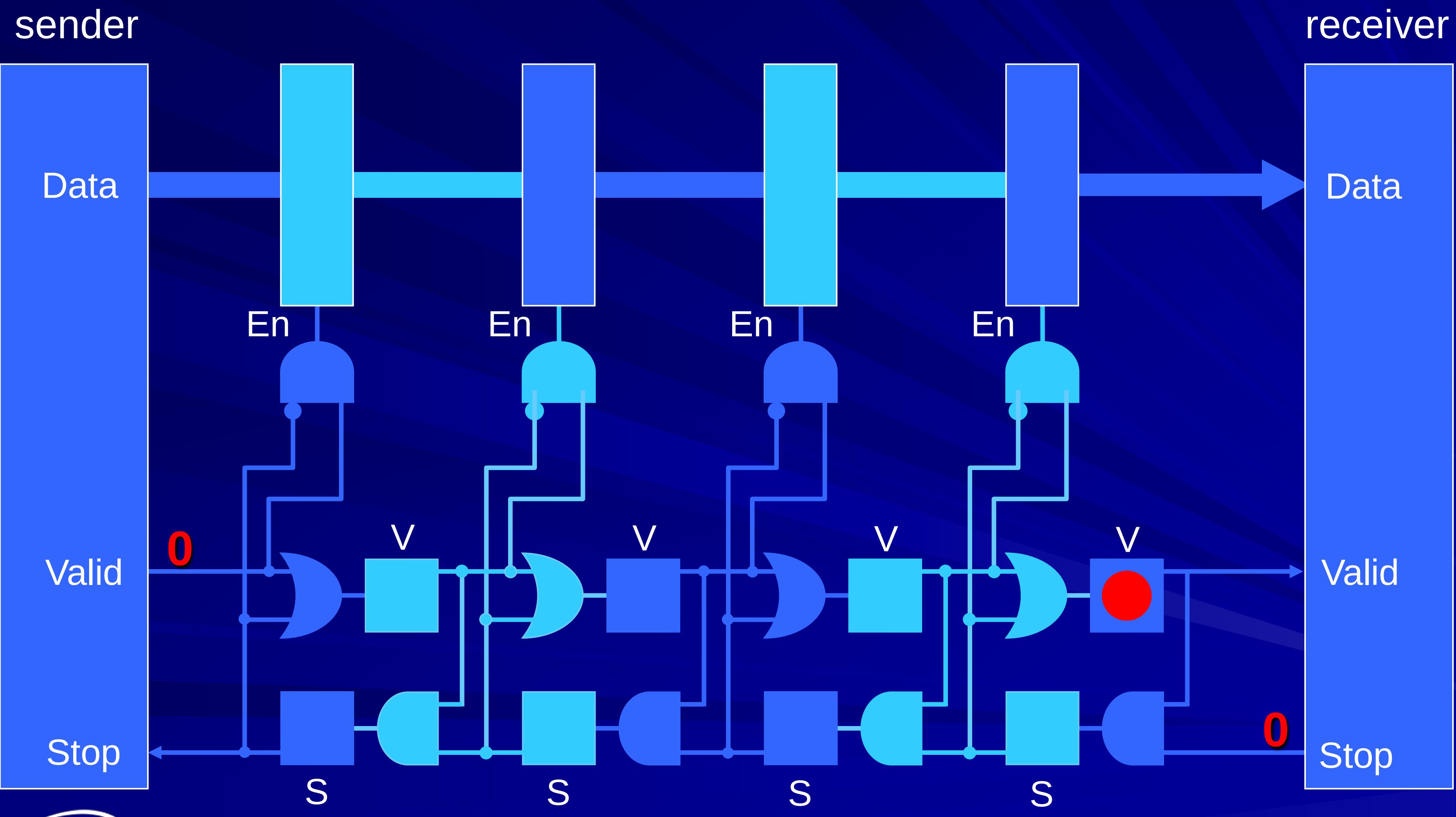
SELF


sender
receiver
Data
Data
En
En
En
En
V
V
V
V
Valid
Valid
0
0
Stop
Stop
S
S
S
S

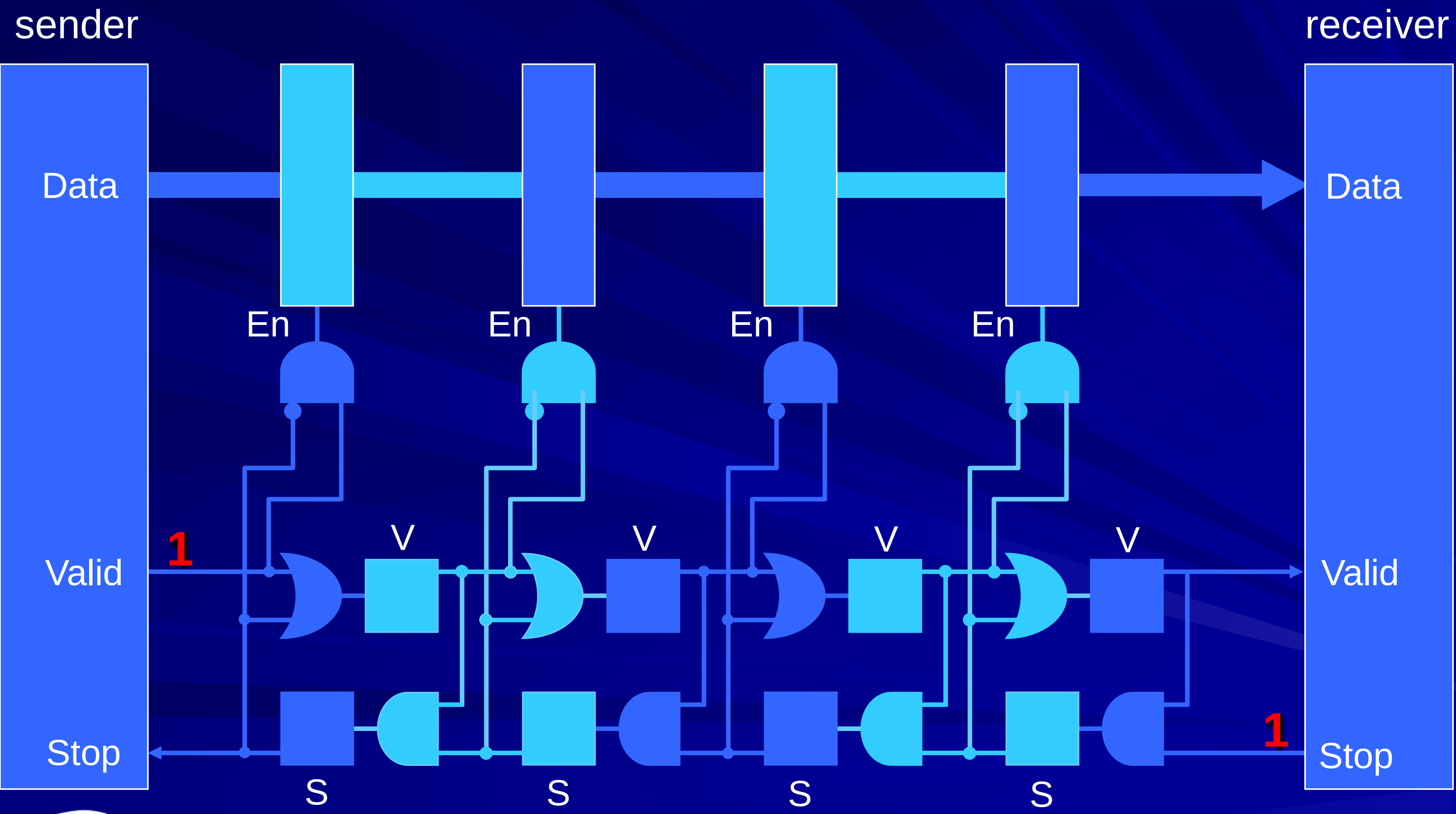
SELF


sender
receiver
Data
Data
En
En
En
En
0
V
V
V
V
Valid
Valid
0
Stop
Stop
S
S
S
S

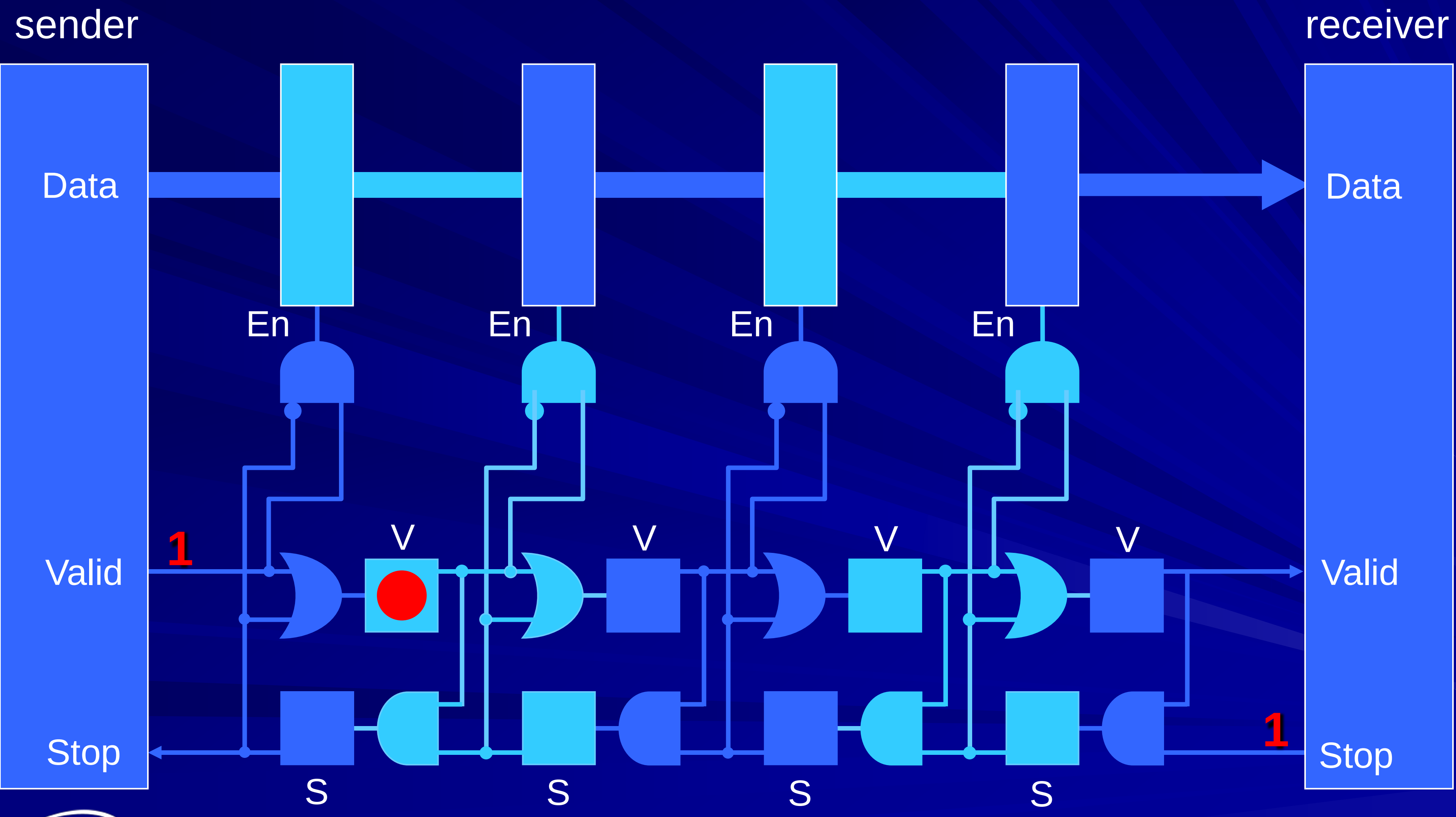
SELF


sender
receiver
Data
Data
En
En
En
En
V
V
V
V
0
Valid
Valid
0
Stop
Stop
S
S
S
S

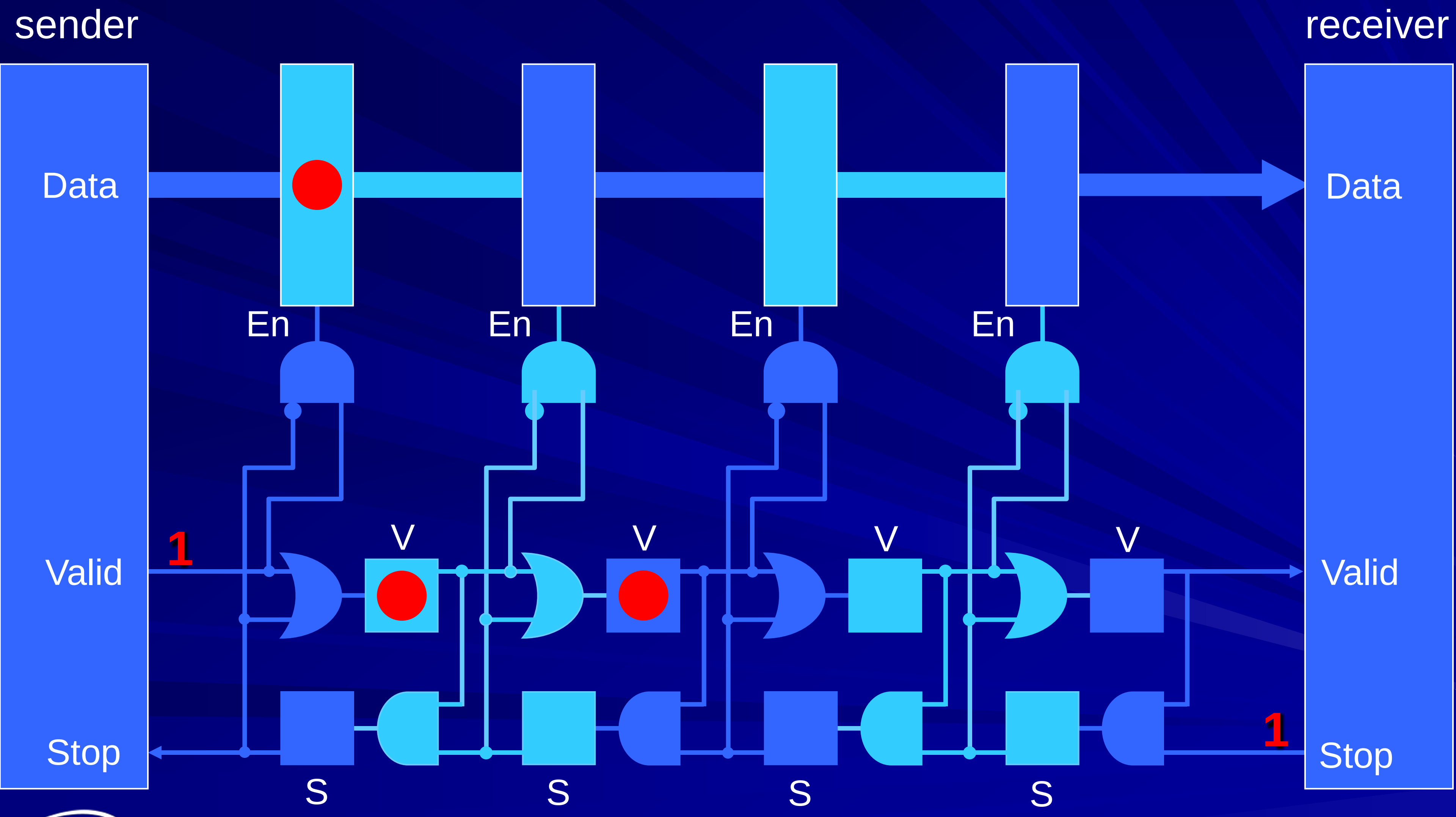
SELF


sender
receiver
Data
Data
En
En
En
En
1
Valid
V
V
V
V
Valid
1
Stop
Stop
S
S
S
S

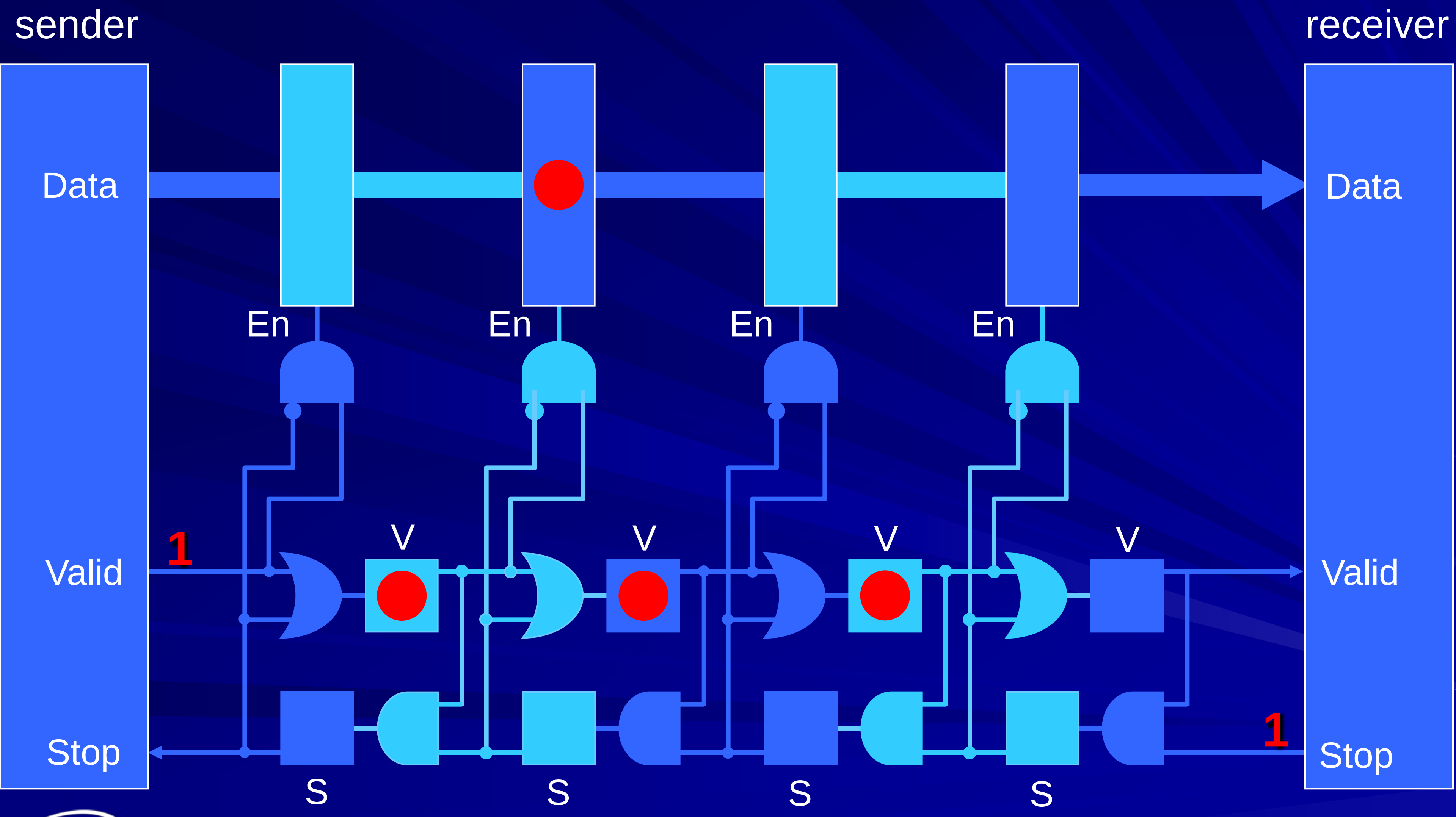
SELF


sender
receiver
Data
Data
En
En
En
En
1
V
V
V
V
Valid
Valid
1
Stop
Stop
S
S
S
S

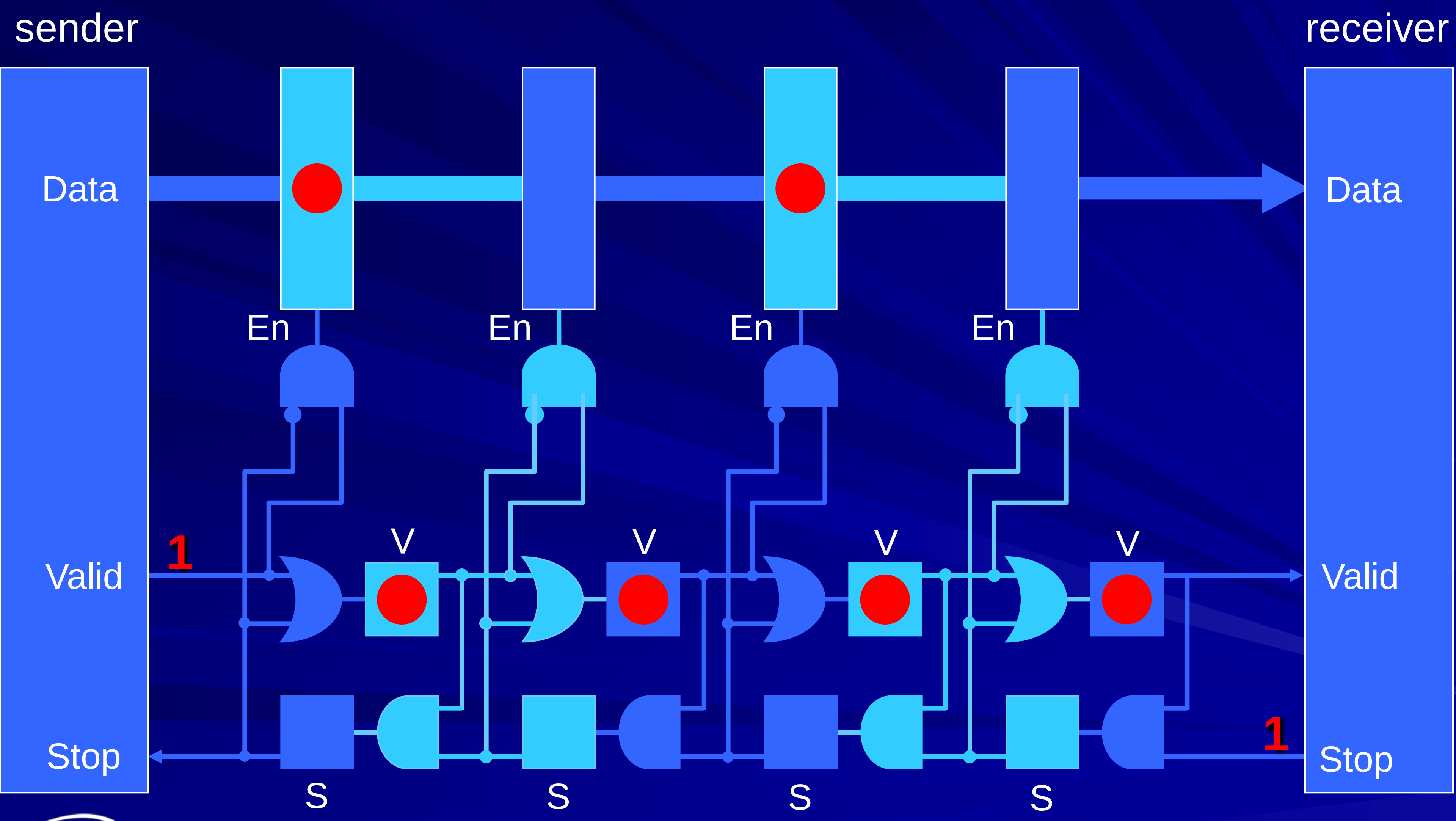
SELF


sender
receiver
Data
Data
En
En
En
En
1
Valid
V
V
V
V
Valid
1
Stop
Stop
S
S
S
S

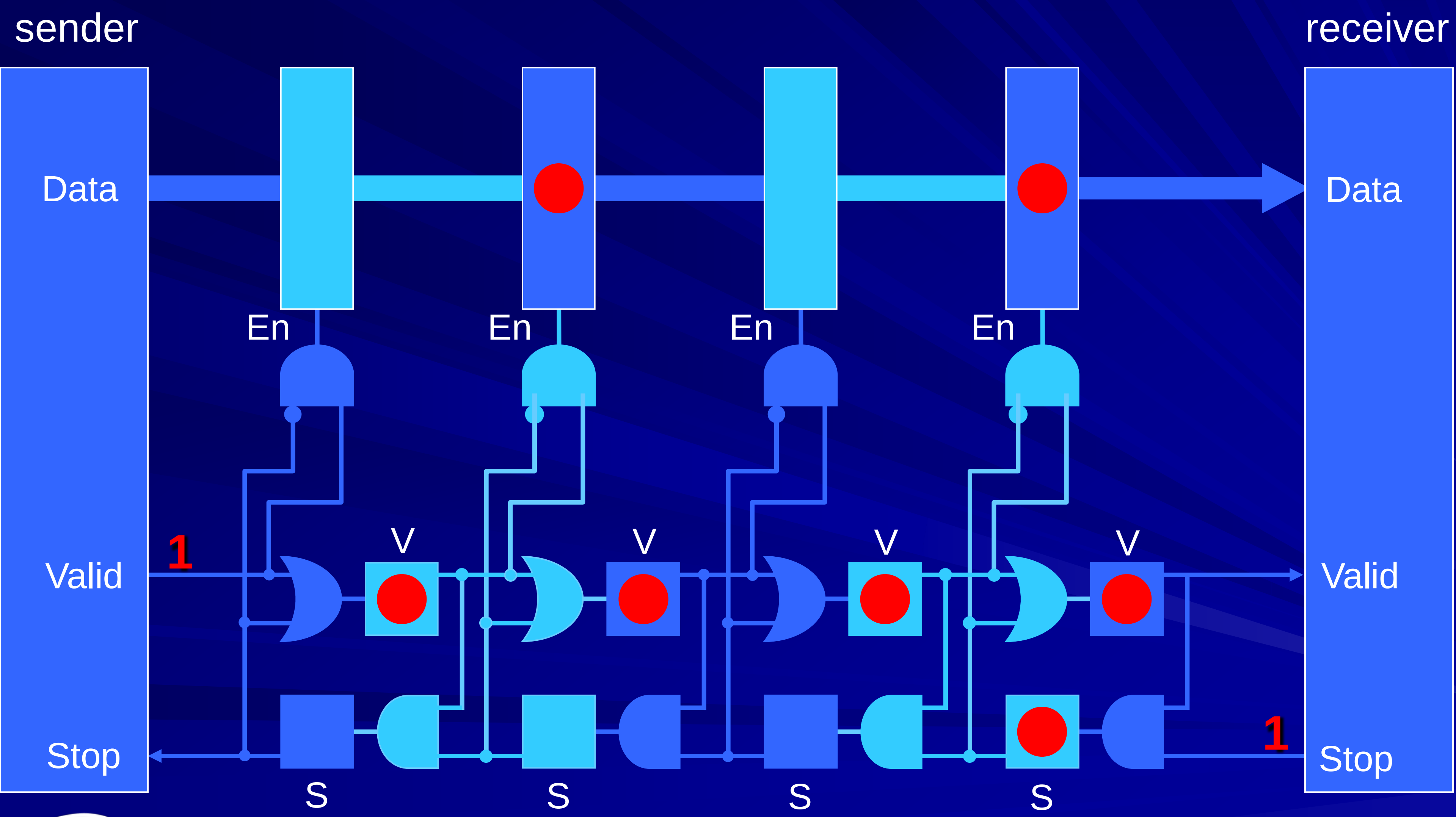
SELF


sender
receiver
Data
Data
En
En
En
En
V
V
V
V
1
Valid
Valid
Stop
1
Stop
S
S
S
S

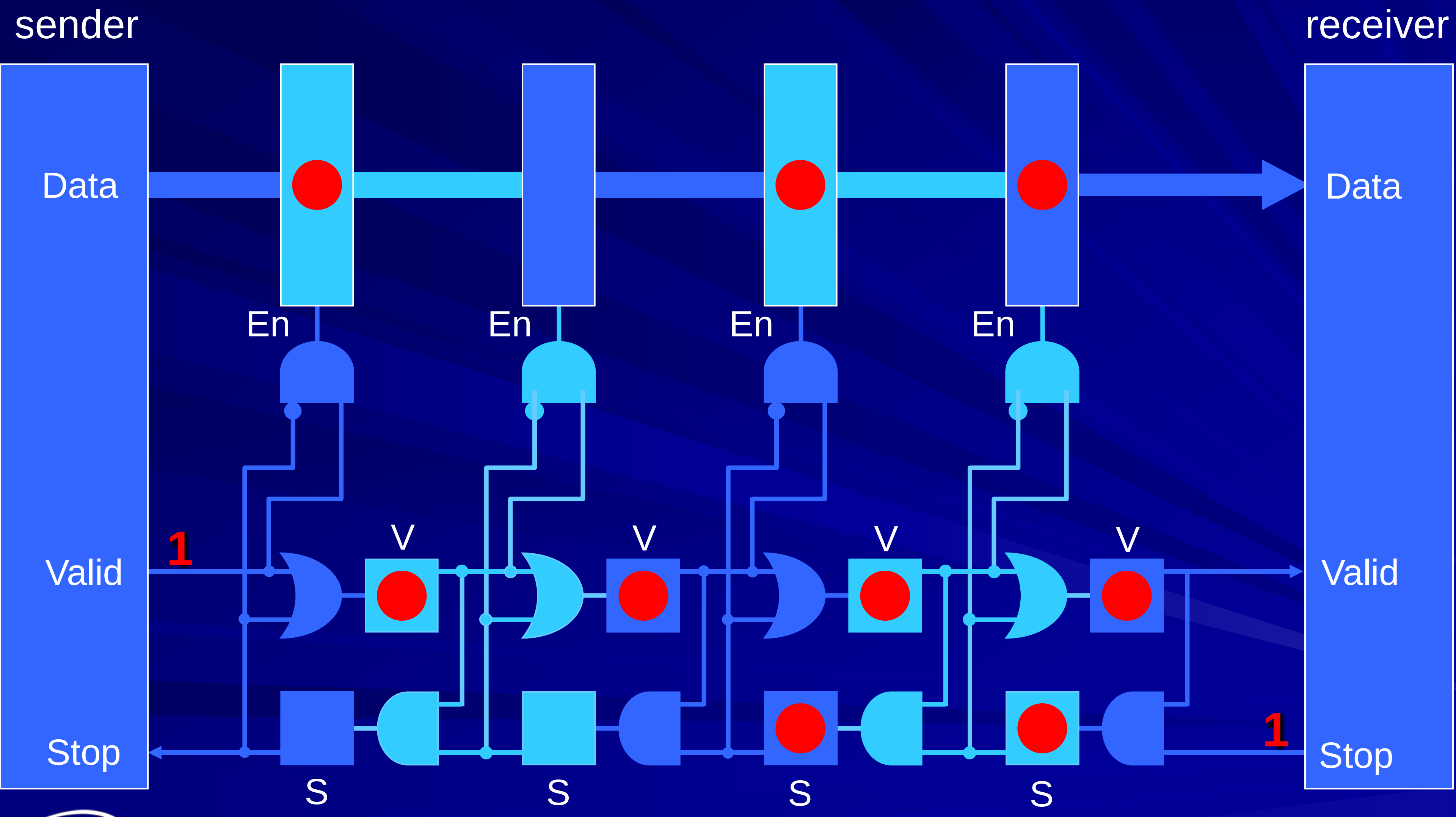
SELF


sender
receiver
Data
Data
En
En
En
En
1
Valid
V
V
V
V
Valid
1
Stop
Stop
S
S
S
S

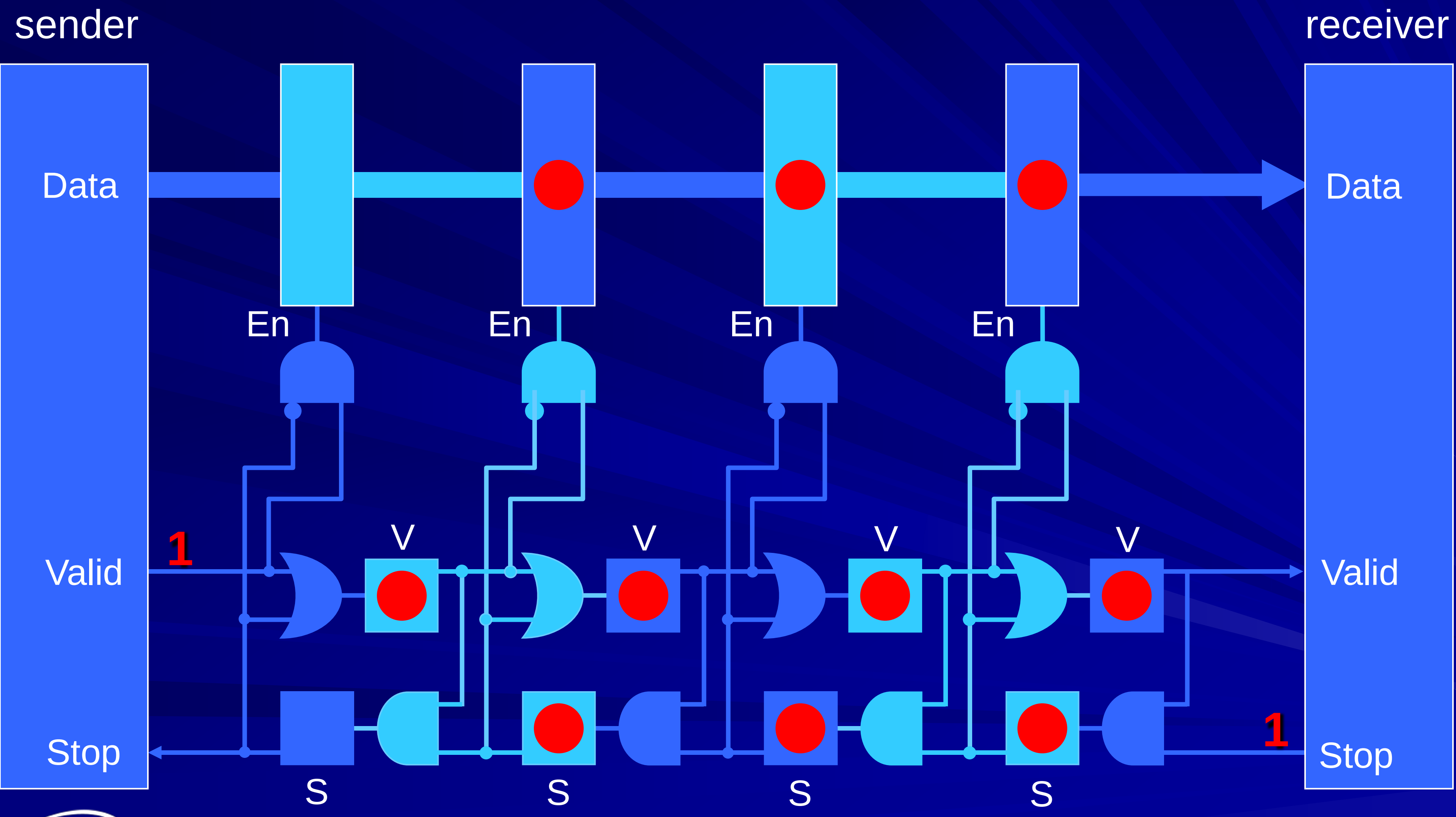
SELF


sender
receiver
Data
Data
En
En
En
En
V
V
V
V
1
Valid
Valid
Stop
1
Stop
S
S
S
S

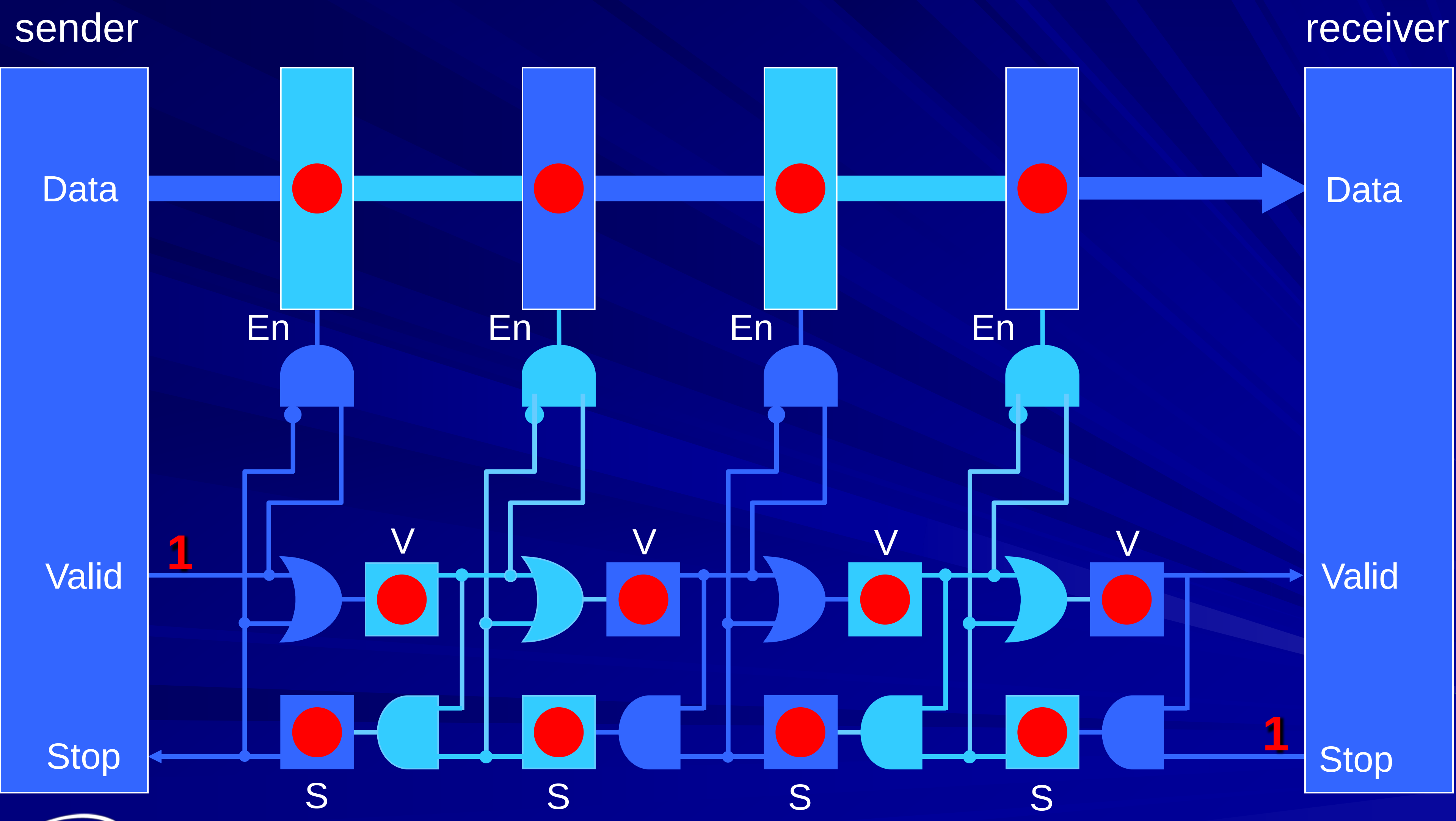
SELF


sender
receiver
Data
Data
En
En
En
En
1
Valid
V
V
V
V
Valid
1
Stop
Stop
S
S
S
S

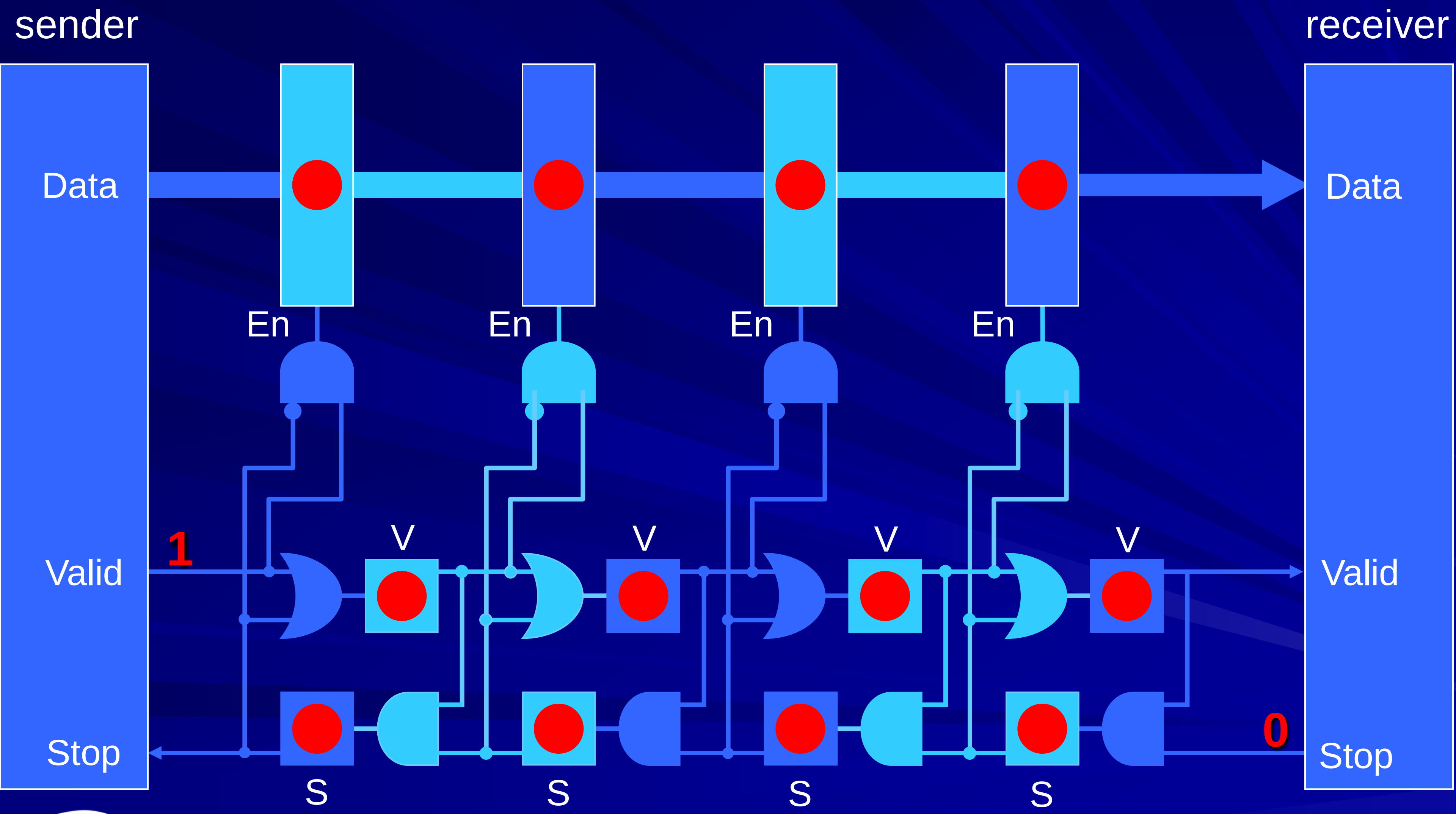
SELF


sender
receiver
Data
Valid
Stop
1
En
En
En
En
V
V
V
V
S
S
S
S
Data
Valid
1
Stop

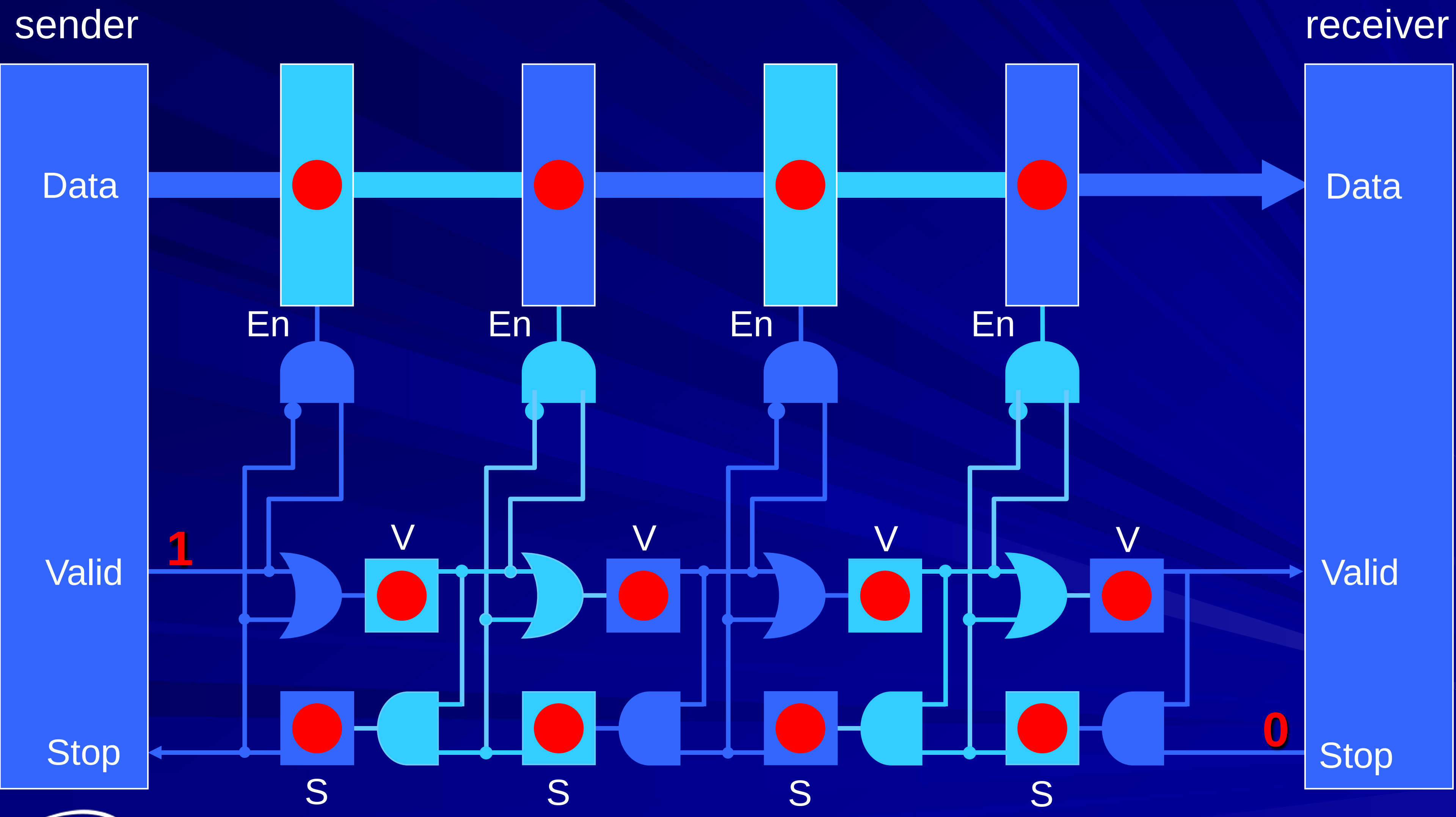
SELF


sender
receiver
Data
Data
En
En
En
En
1
Valid
V
V
V
V
Valid
1
Stop
Stop
S
S
S
S

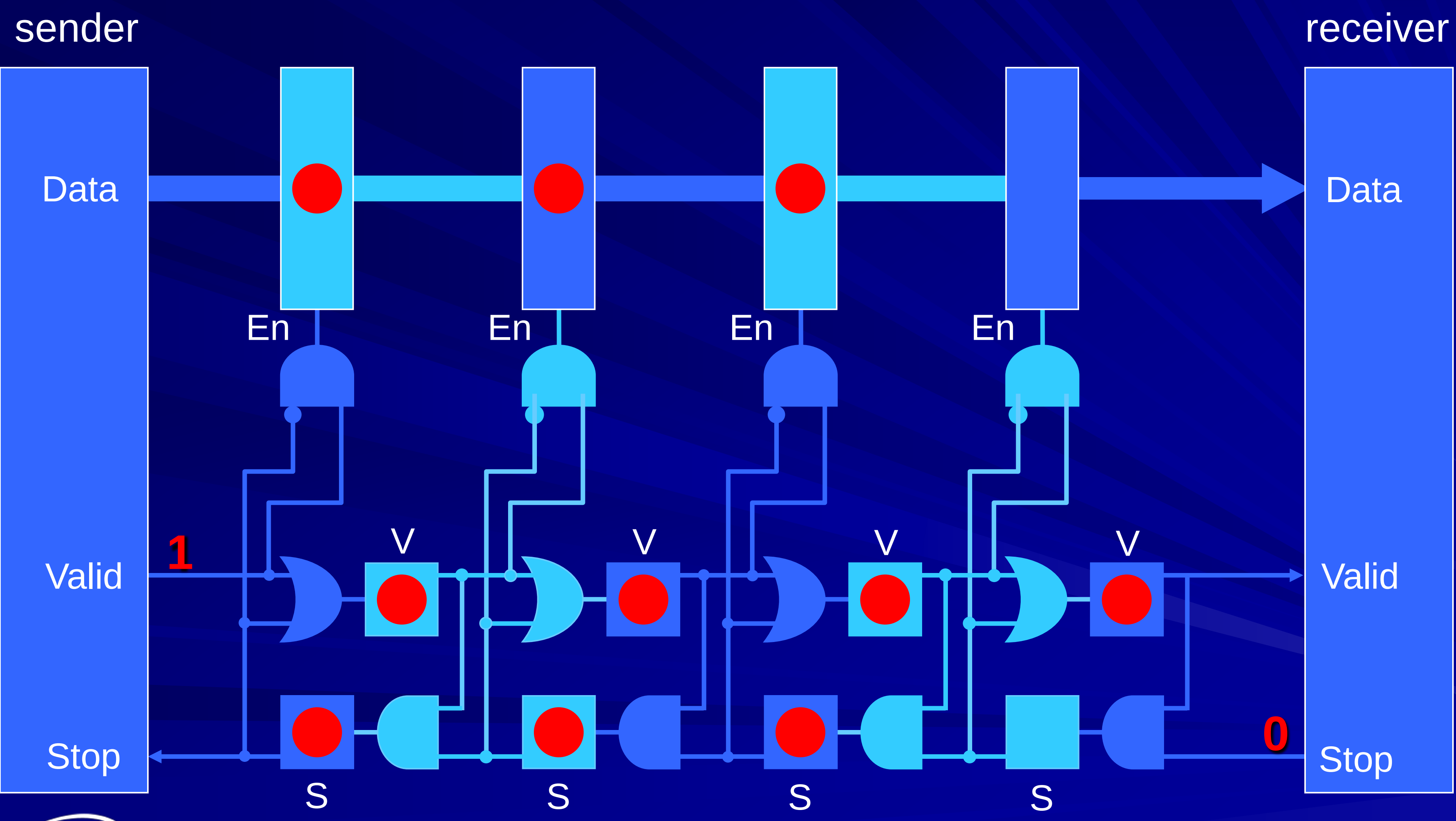
SELF


sender
receiver
Data
Data
En
En
En
En
1
V
V
V
V
Valid
Valid
0
Stop
Stop
S
S
S
S

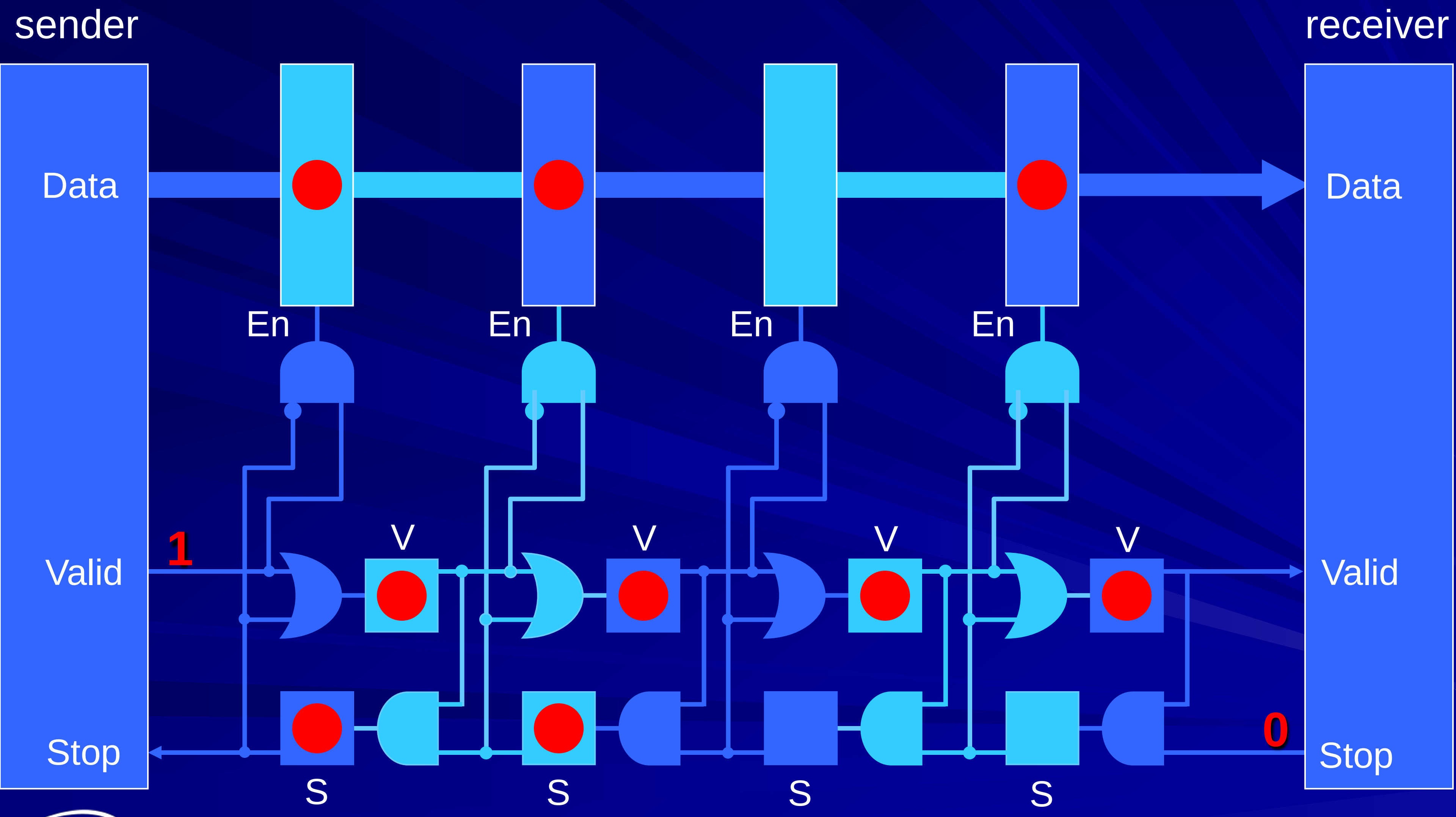
SELF


sender
receiver
Data
Data
En
En
En
En
V
V
V
V
Valid
1
Valid
Stop
0
Stop
S
S
S
S

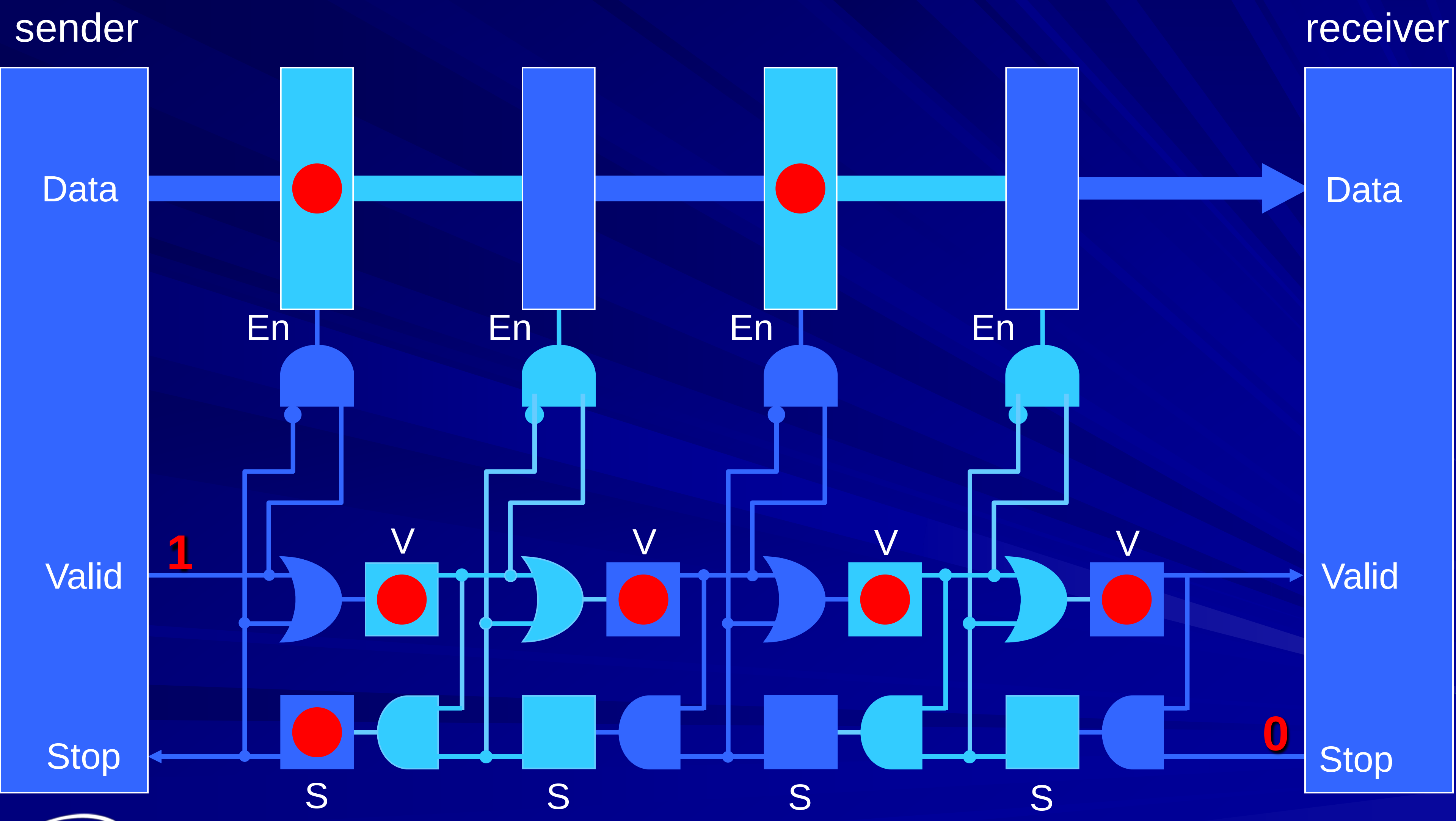
SELF


sender
receiver
Data
Valid
Stop
1
En
En
En
En
V
V
V
V
S
S
S
S
0
Data
Valid
Stop

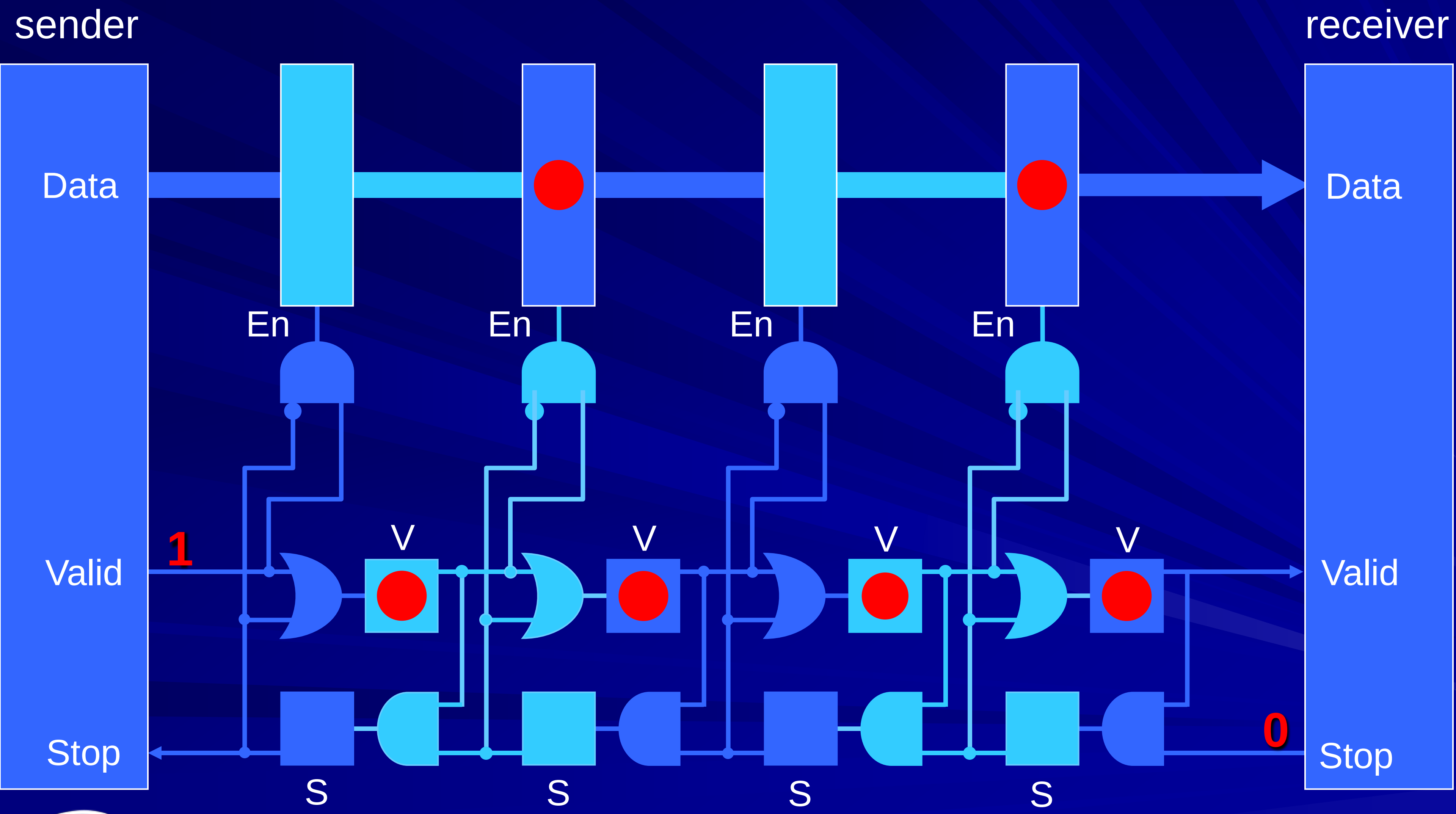
SELF


sender
receiver
Data
En
En
En
En
Data
1
V
V
V
V
Valid
Valid
0
Stop
Stop
S
S
S
S

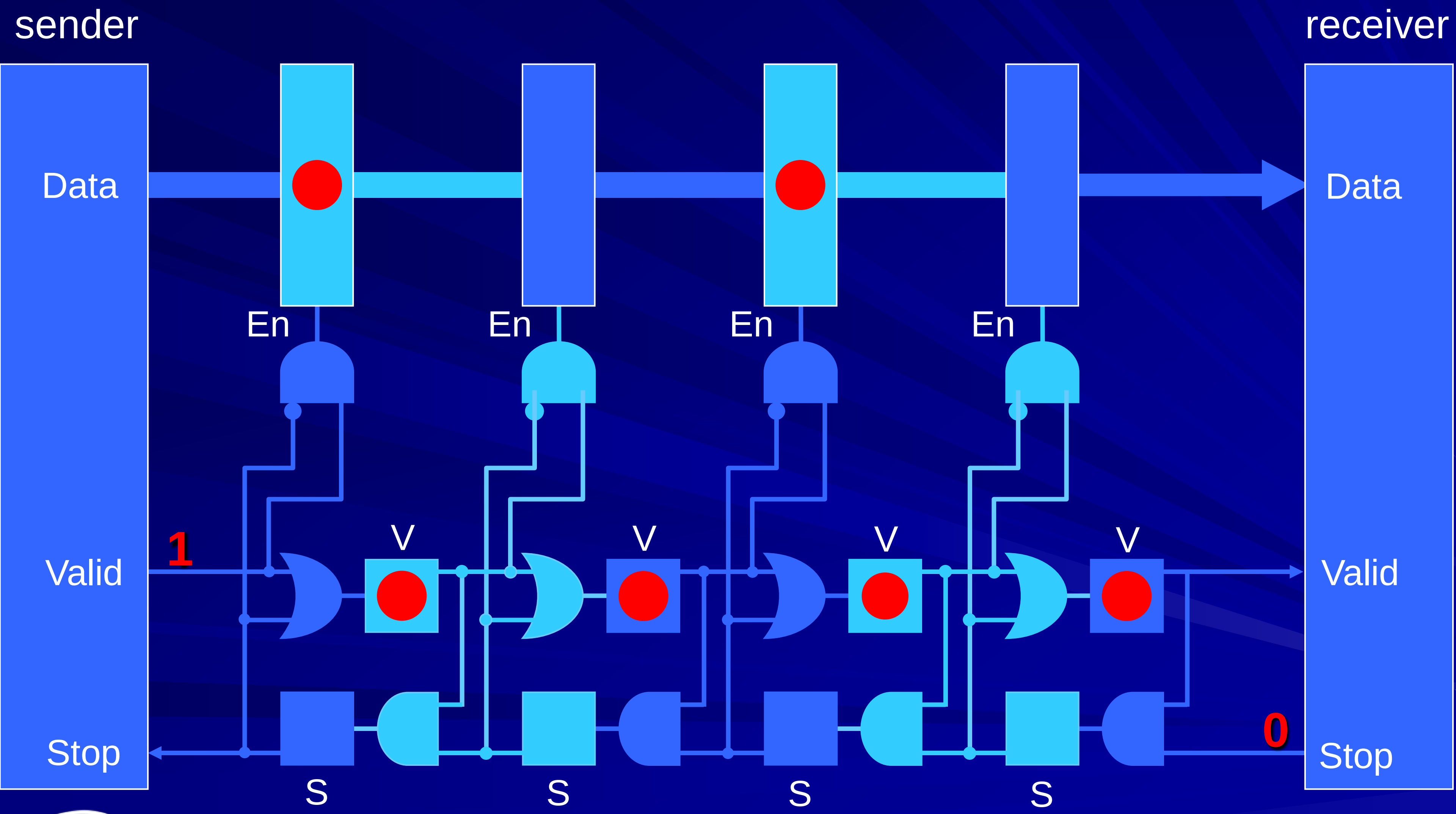
SELF


sender
receiver
Data
Data
En
En
En
En
1
Valid
V
V
V
V
Valid
0
Stop
Stop
S
S
S
S

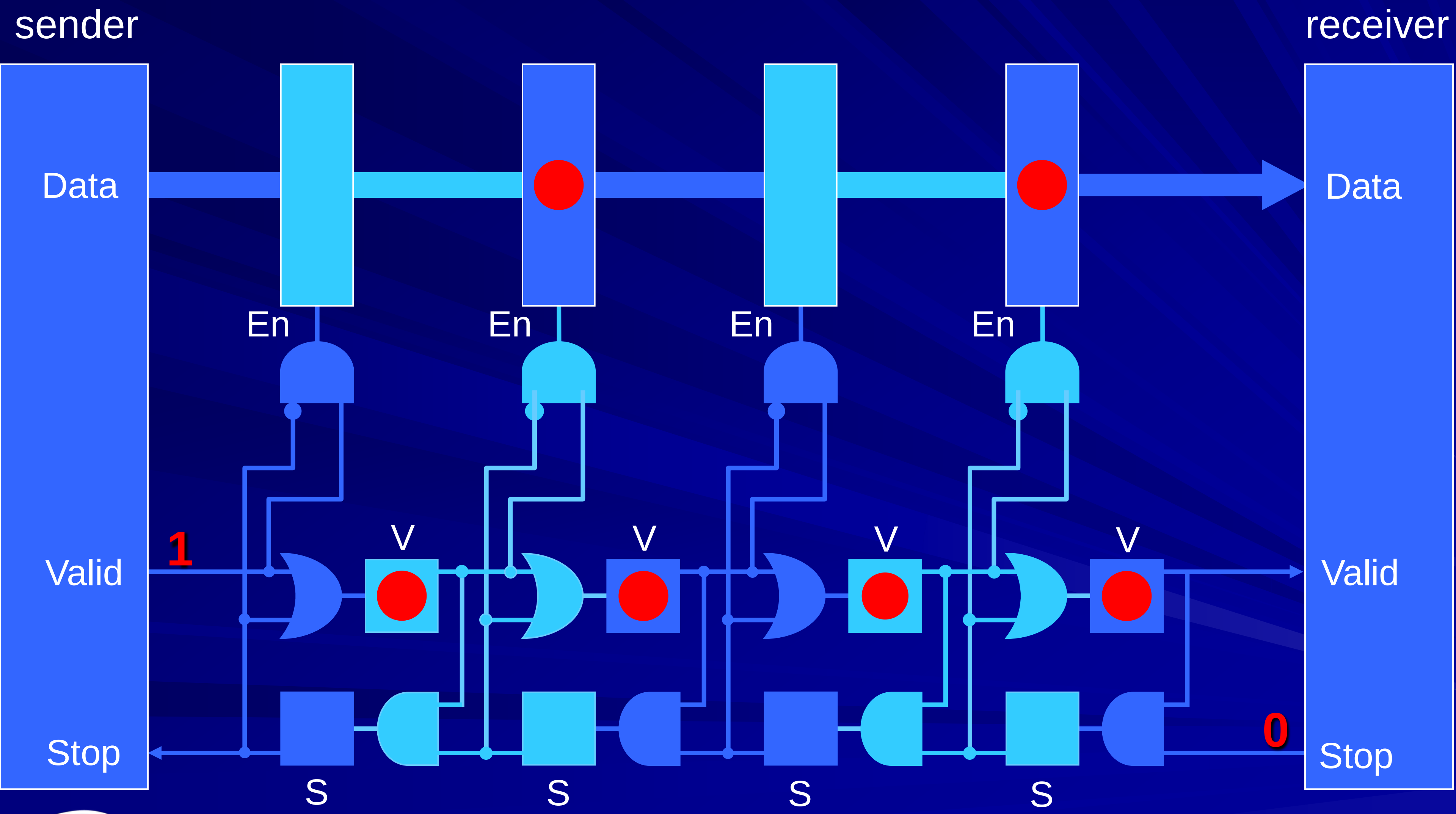
SELF


sender
receiver
Data
Data
En
En
En
En
V
V
V
V
Valid
1
Valid
0
Stop
Stop
S
S
S
S

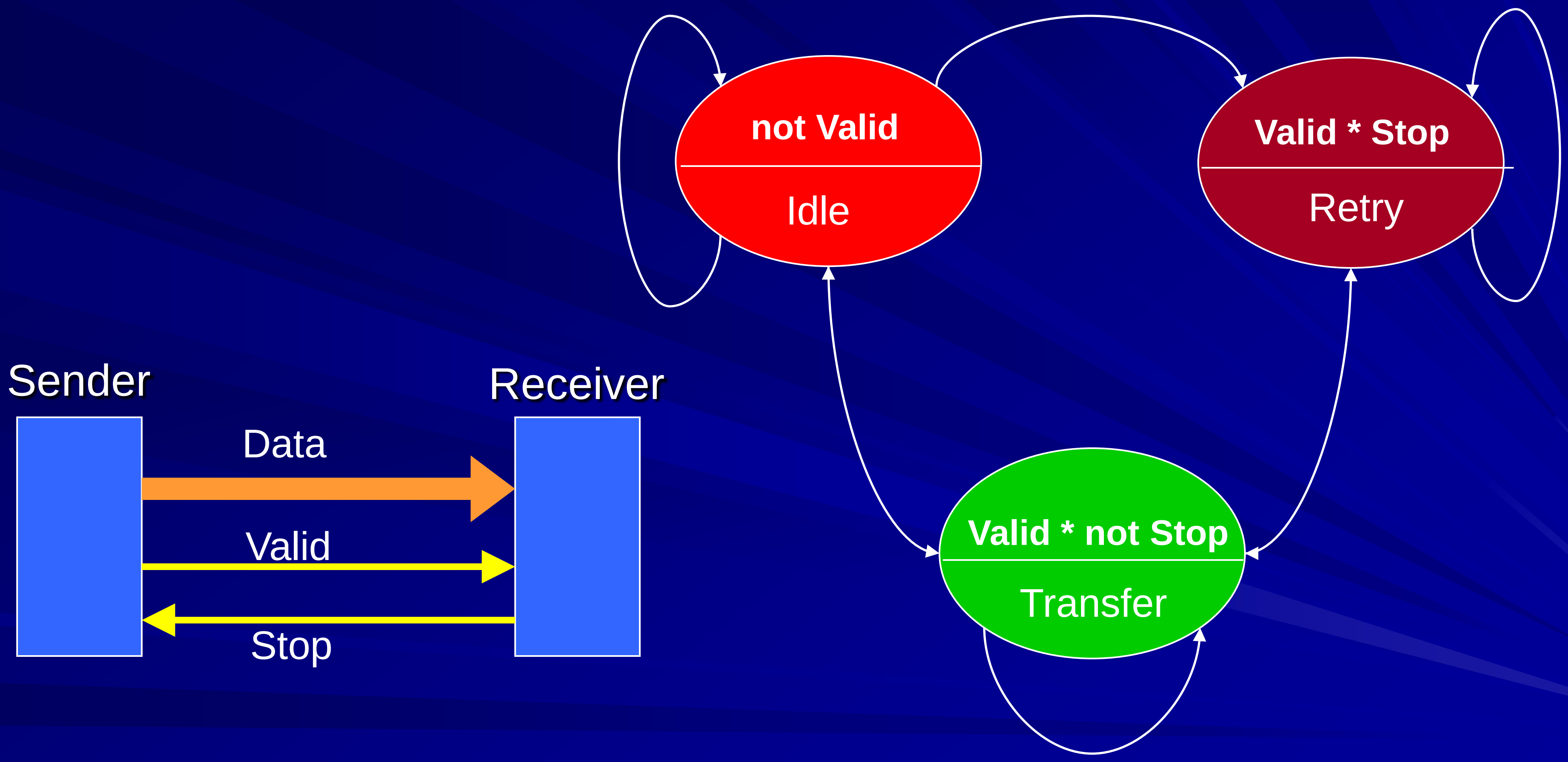
SELF


sender
receiver
Data
Data
En
En
En
En
1
Valid
V
V
V
V
Valid
0
Stop
Stop
S
S
S
S

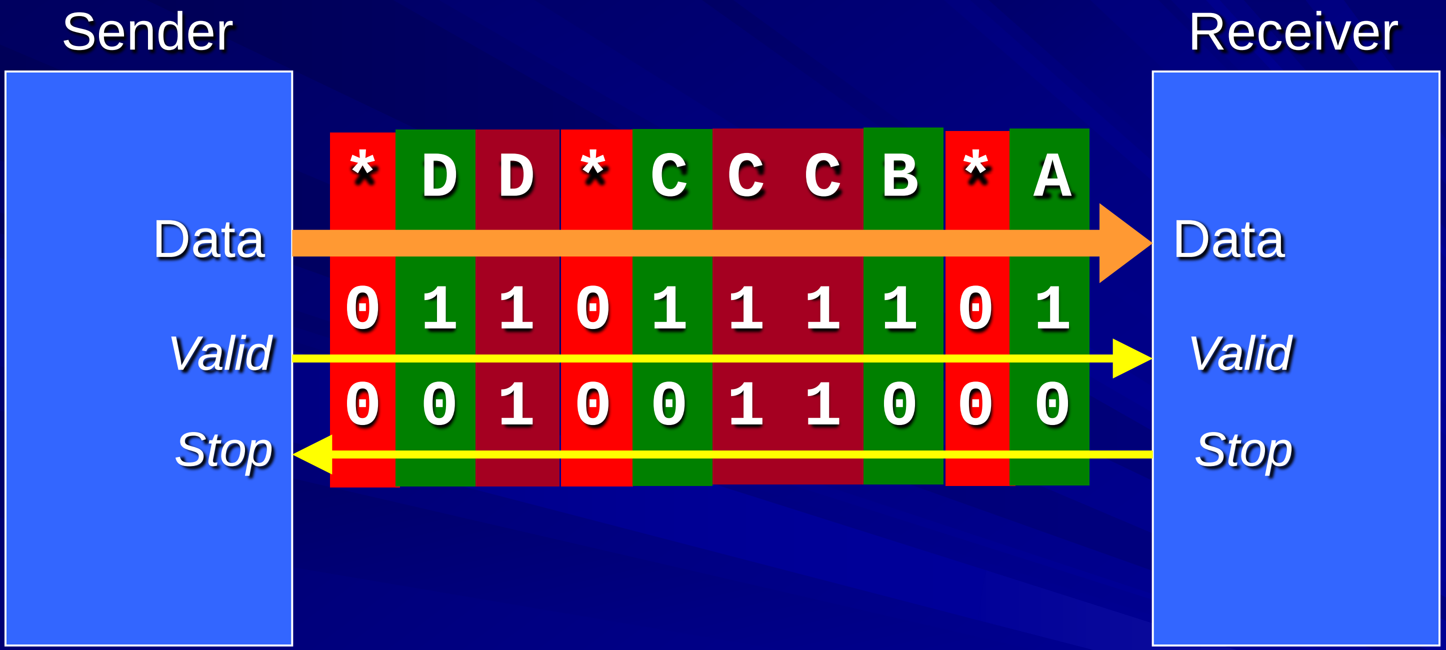
SELF


sender
receiver
Data
Valid
Stop
1
En
En
En
En
V
V
V
V
S
S
S
S
0
Data
Valid
Stop

Elastic channel and its protocol



Elastic channel protocol



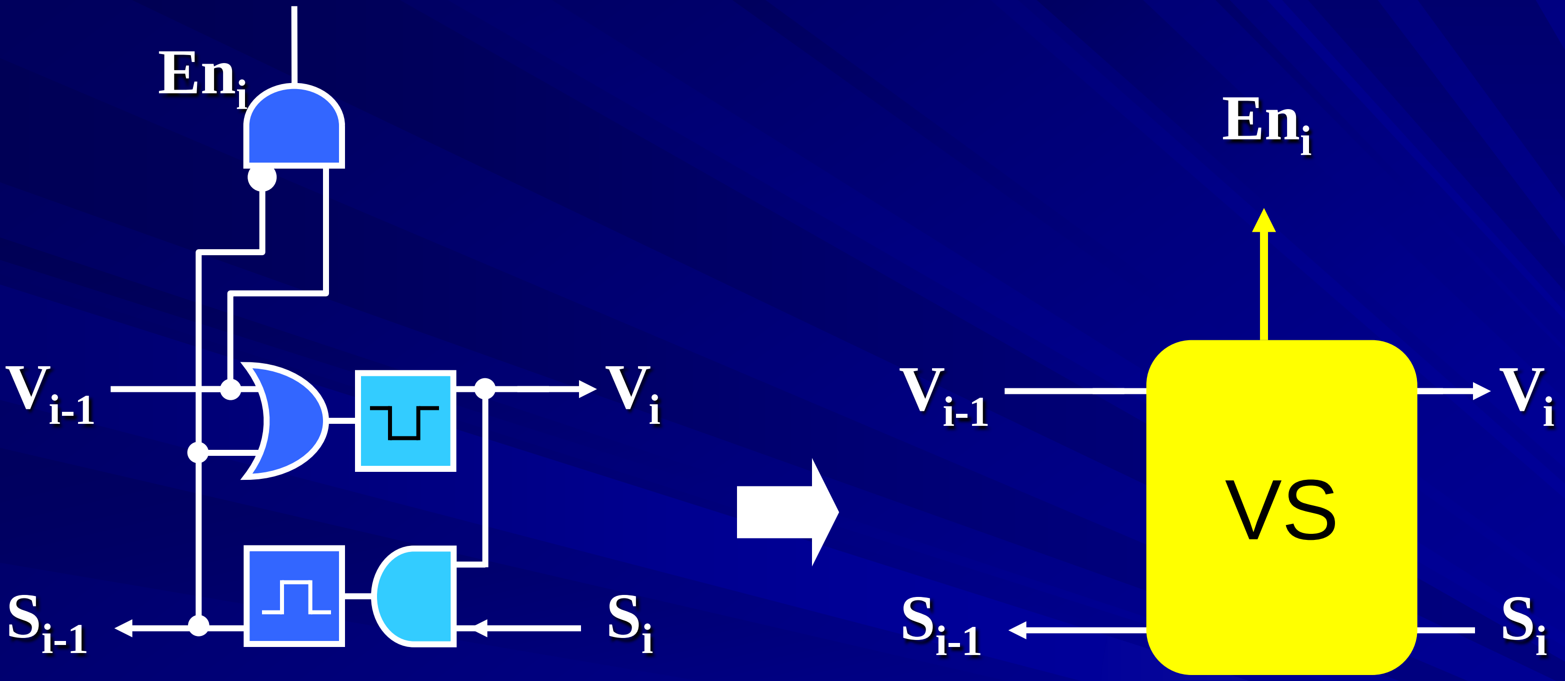
Data	*	D	D	*	C	C	C	B	*	A	Data
Valid	0	1	1	0	1	1	1	1	0	1	Valid
Stop	0	0	1	0	0	1	1	0	0	0	Stop

Transfer

Retry

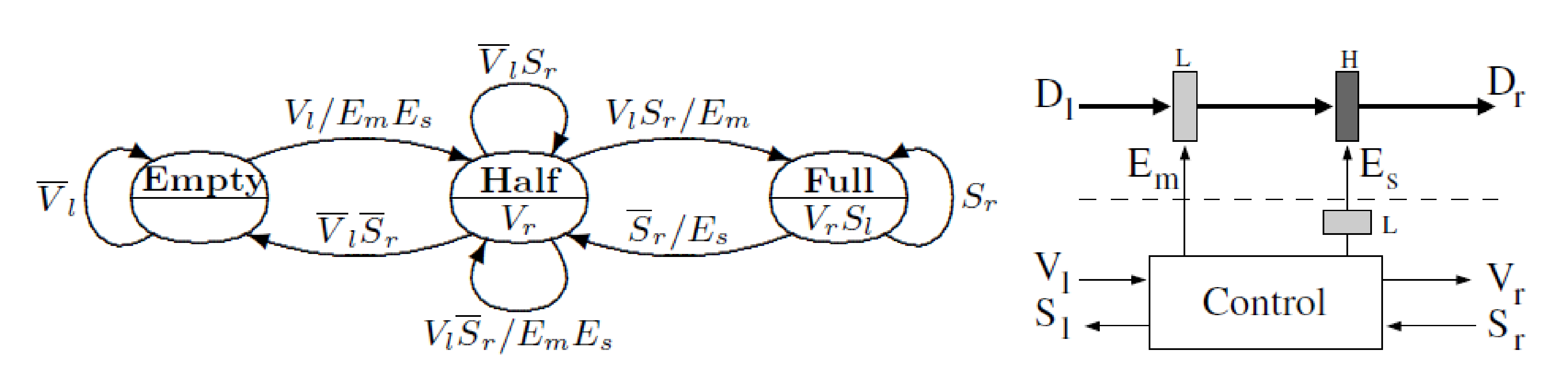
Idle

Basic VS block

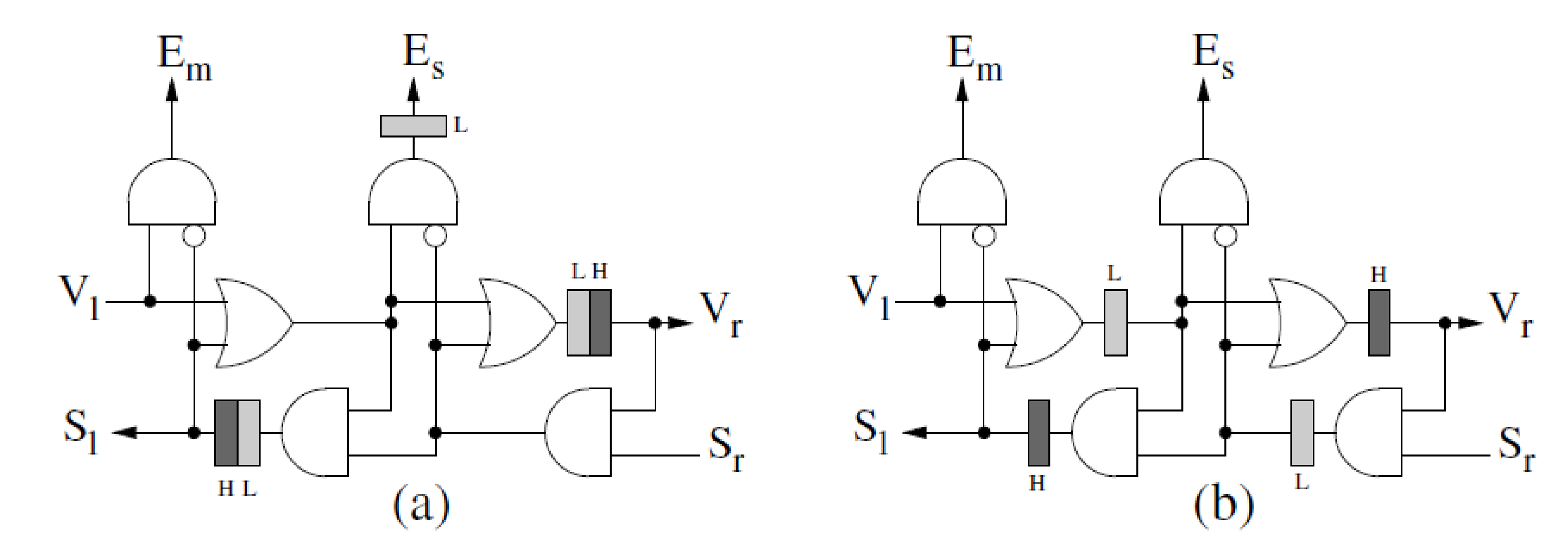


VS block + data-path latch = elastic HALF-buffer (EHB)
EHB + EHB = elastic buffer with capacity 2

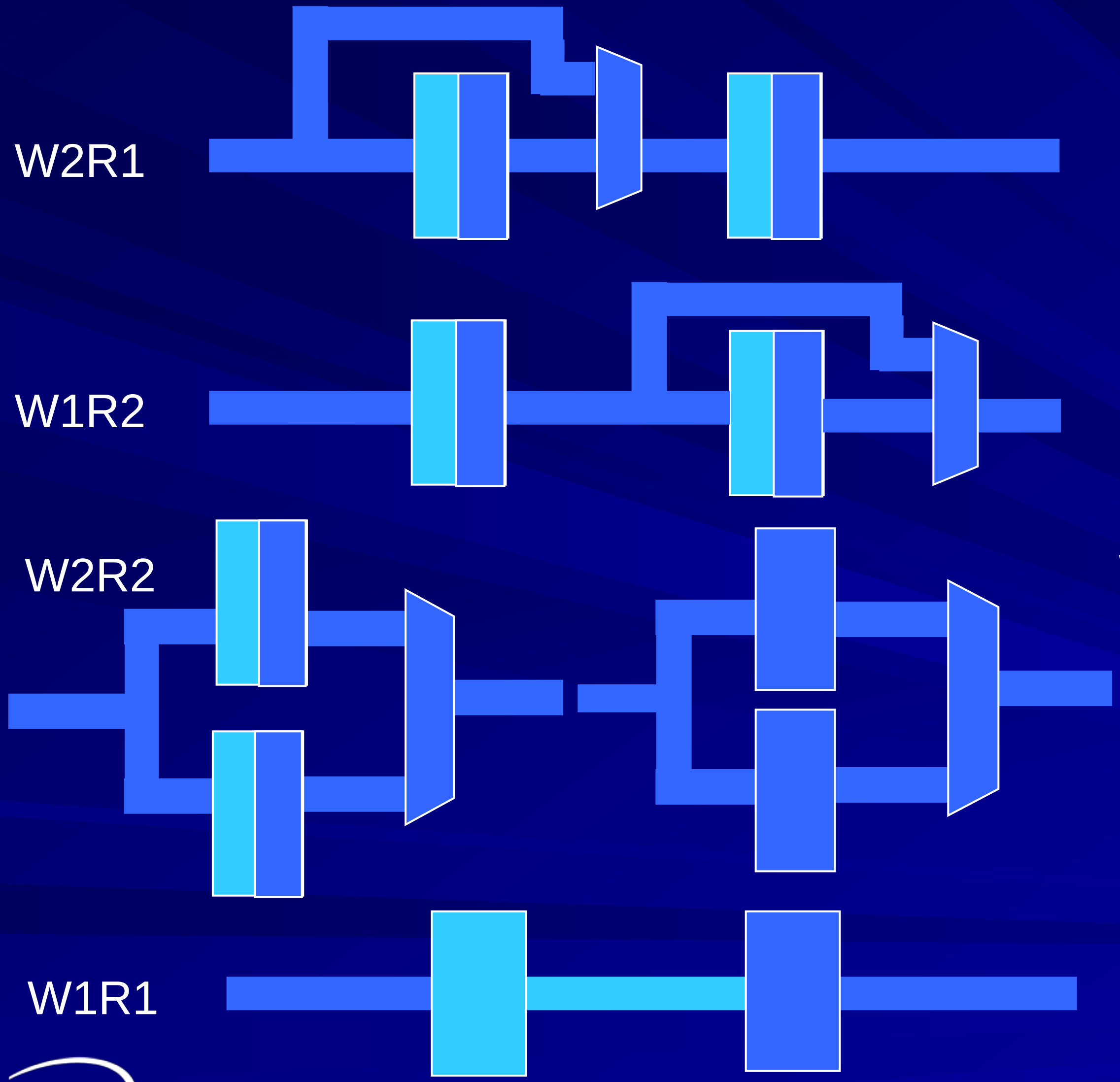
Control specification of the EB



Two implementations



Elastic buffer keeps data while stop is in flight



EBs = FIFOs with two parameters:

- Forward latency
- Capacity

Backward latency for stop propagation assumed (but need not be) equal to fwd latency

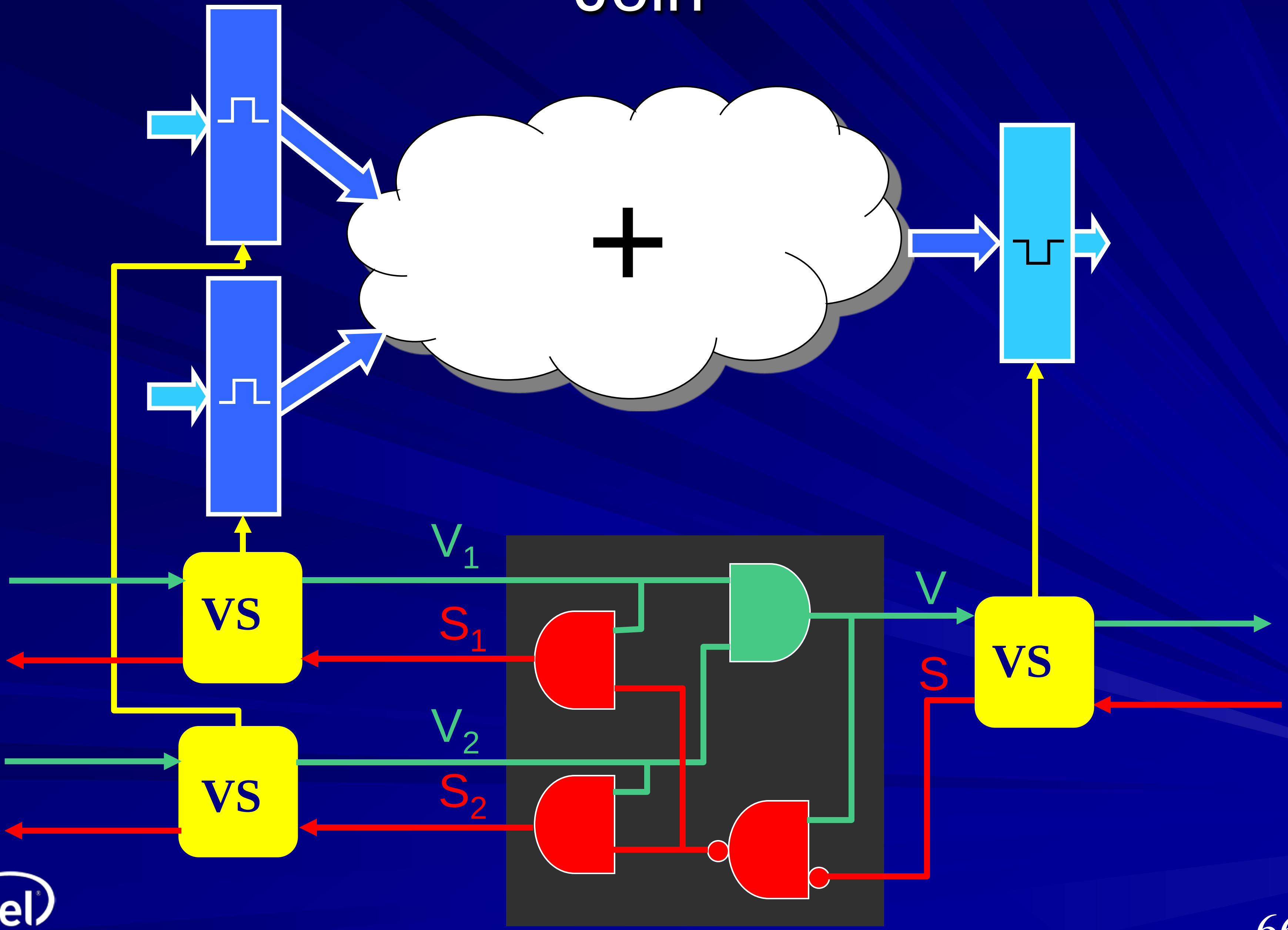
Typical case: (1,2) -
1 cycle forward latency
with capacity of 2

- Replaces “normal” registers
- Decoupling buffers

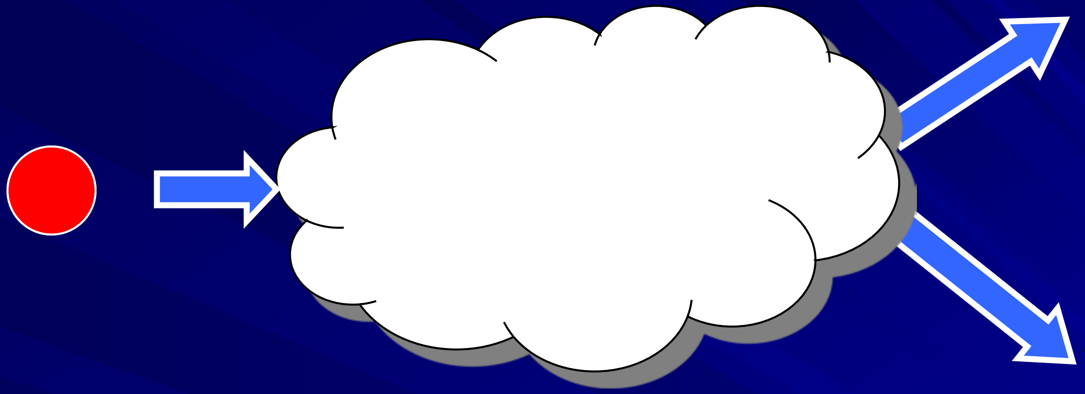
W1R1 Cannot be done with Single Edge Flops without double pumping

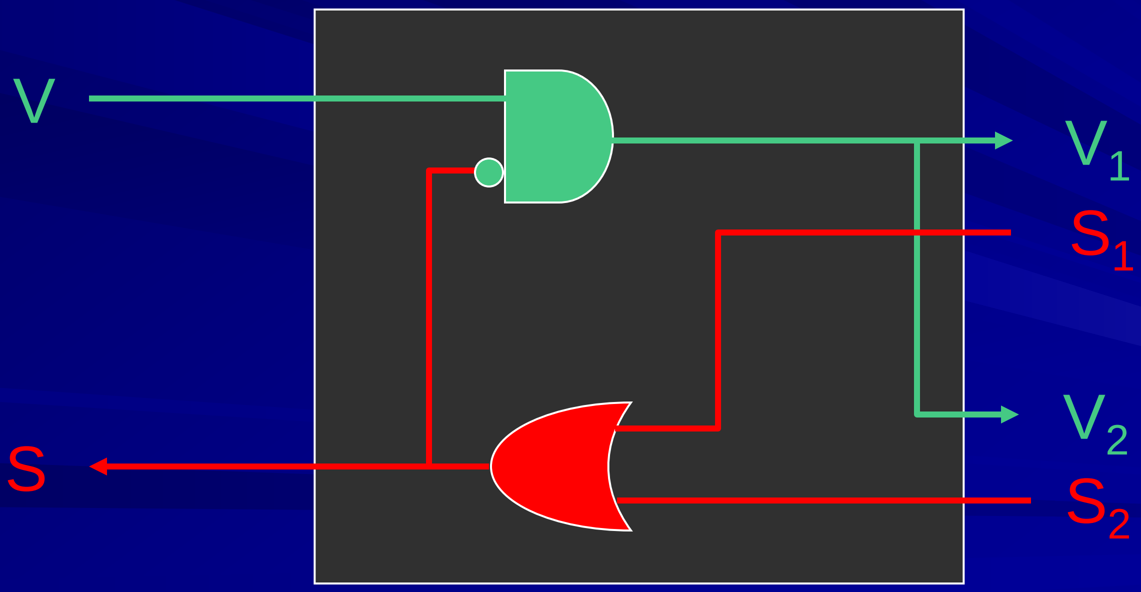
Can use latches inside Master-Slave as shown before

Join

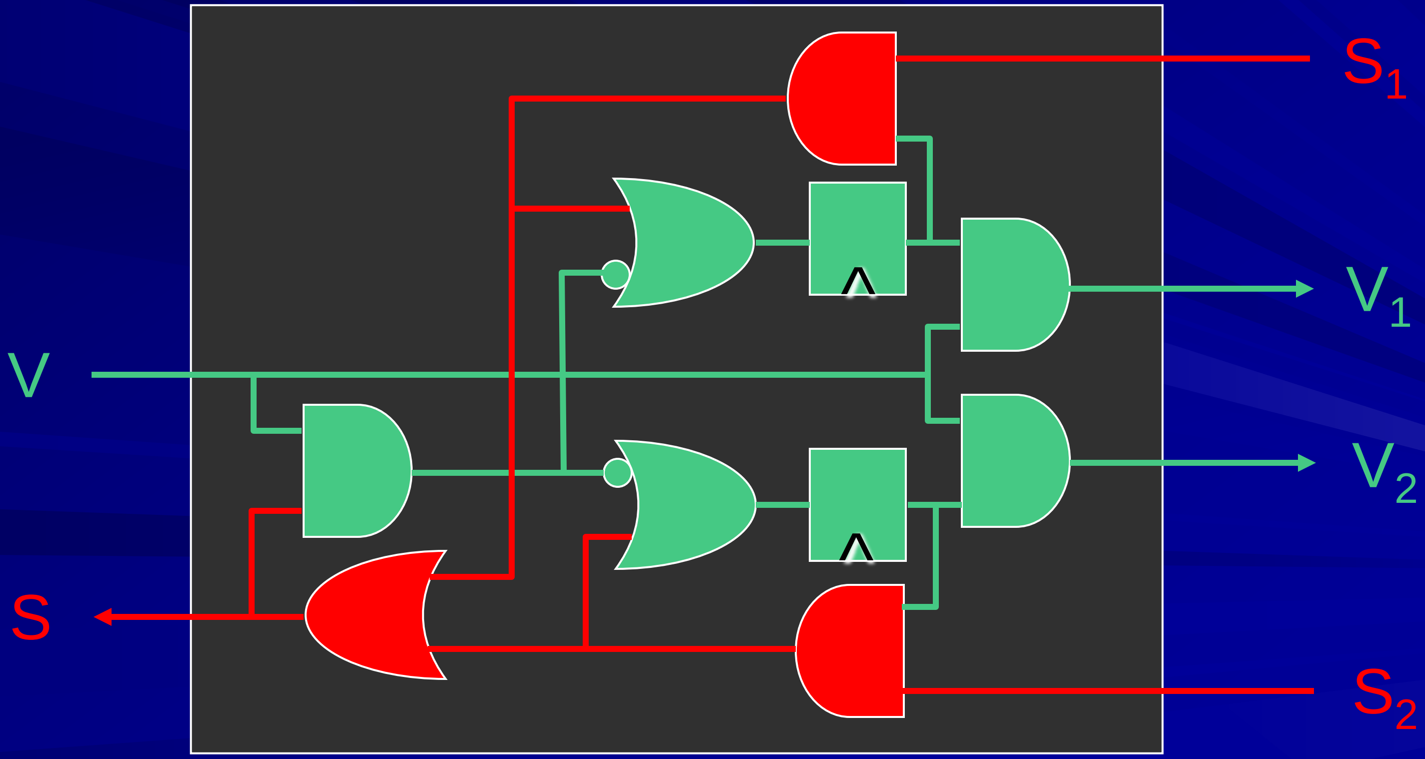


(Lazy) Fork

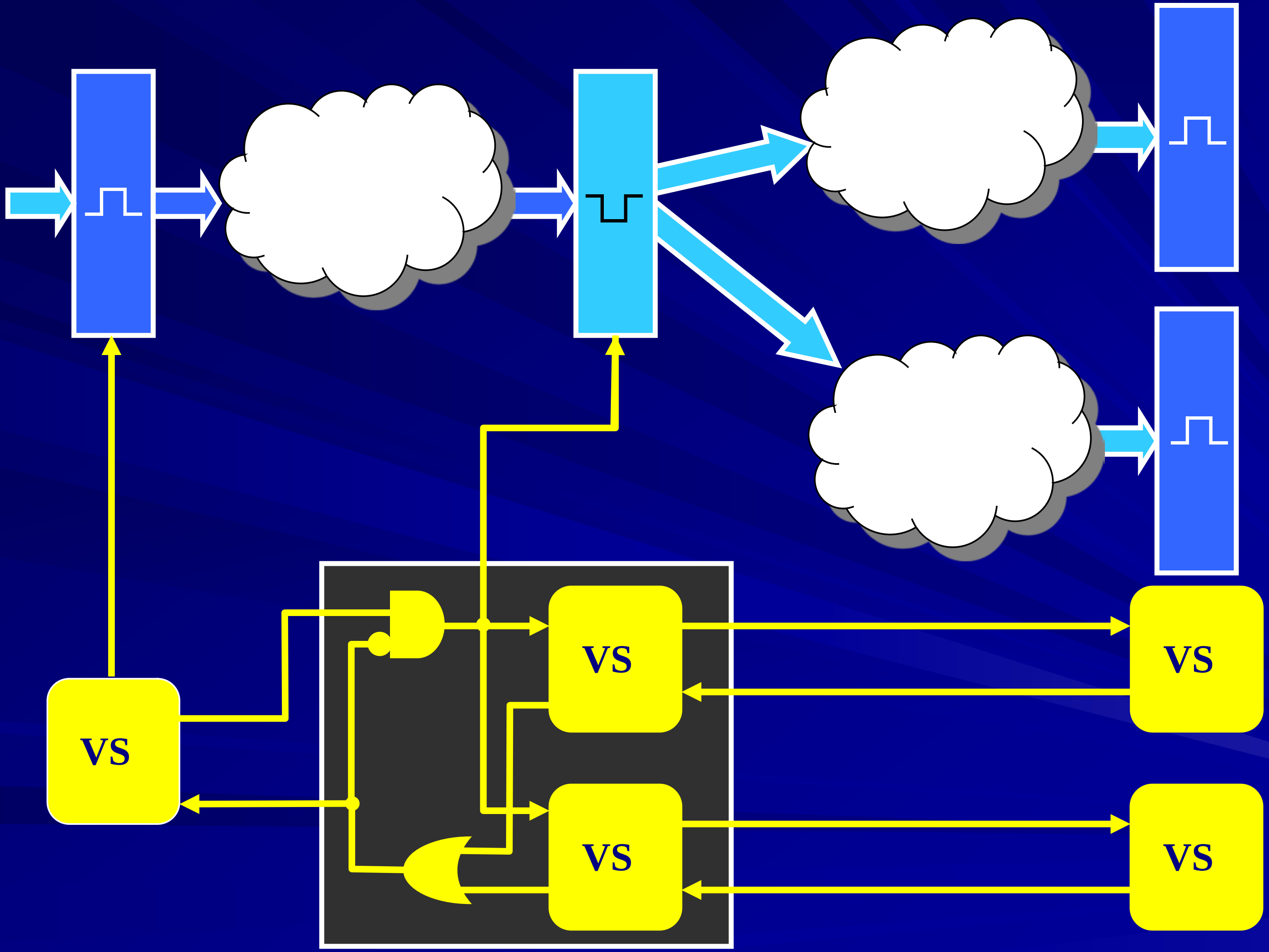




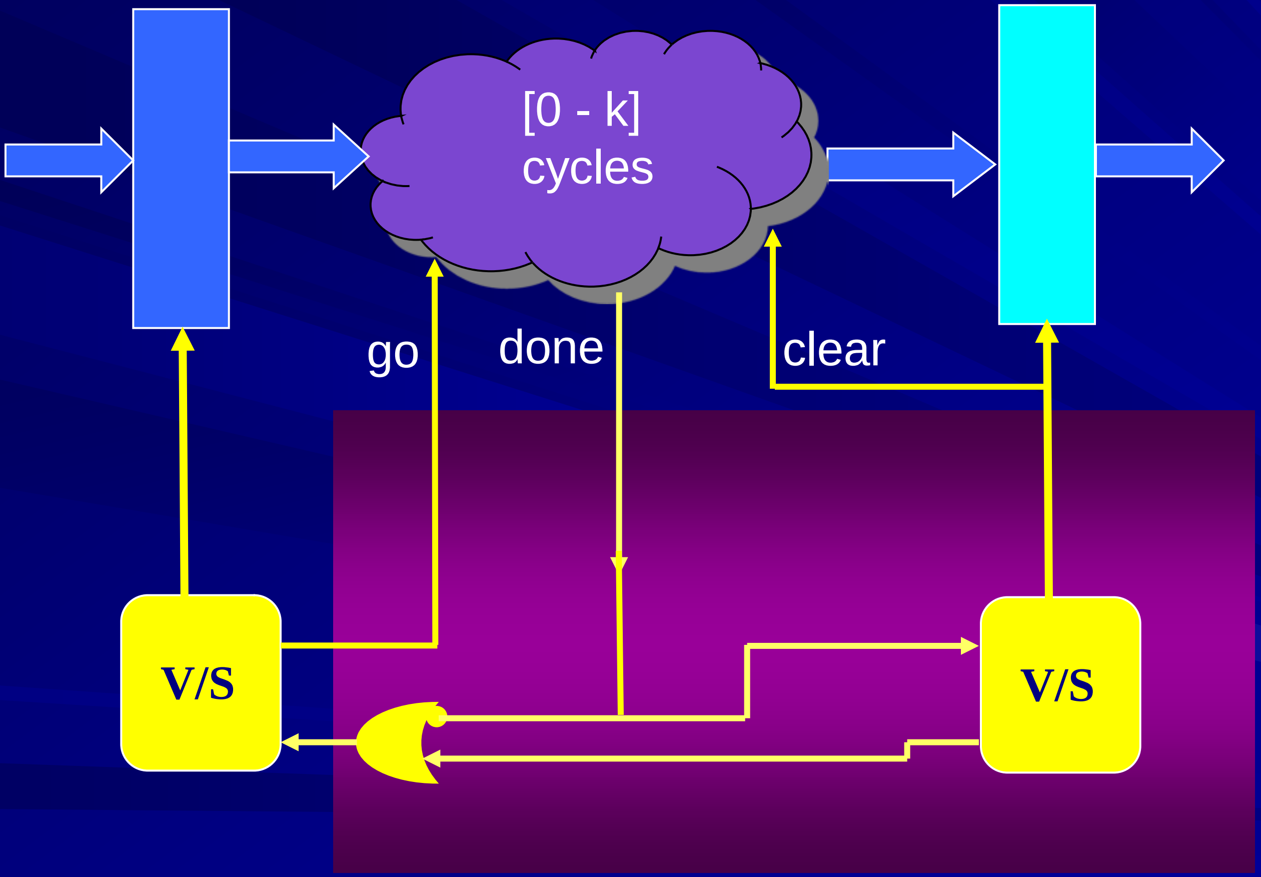
Eager Fork



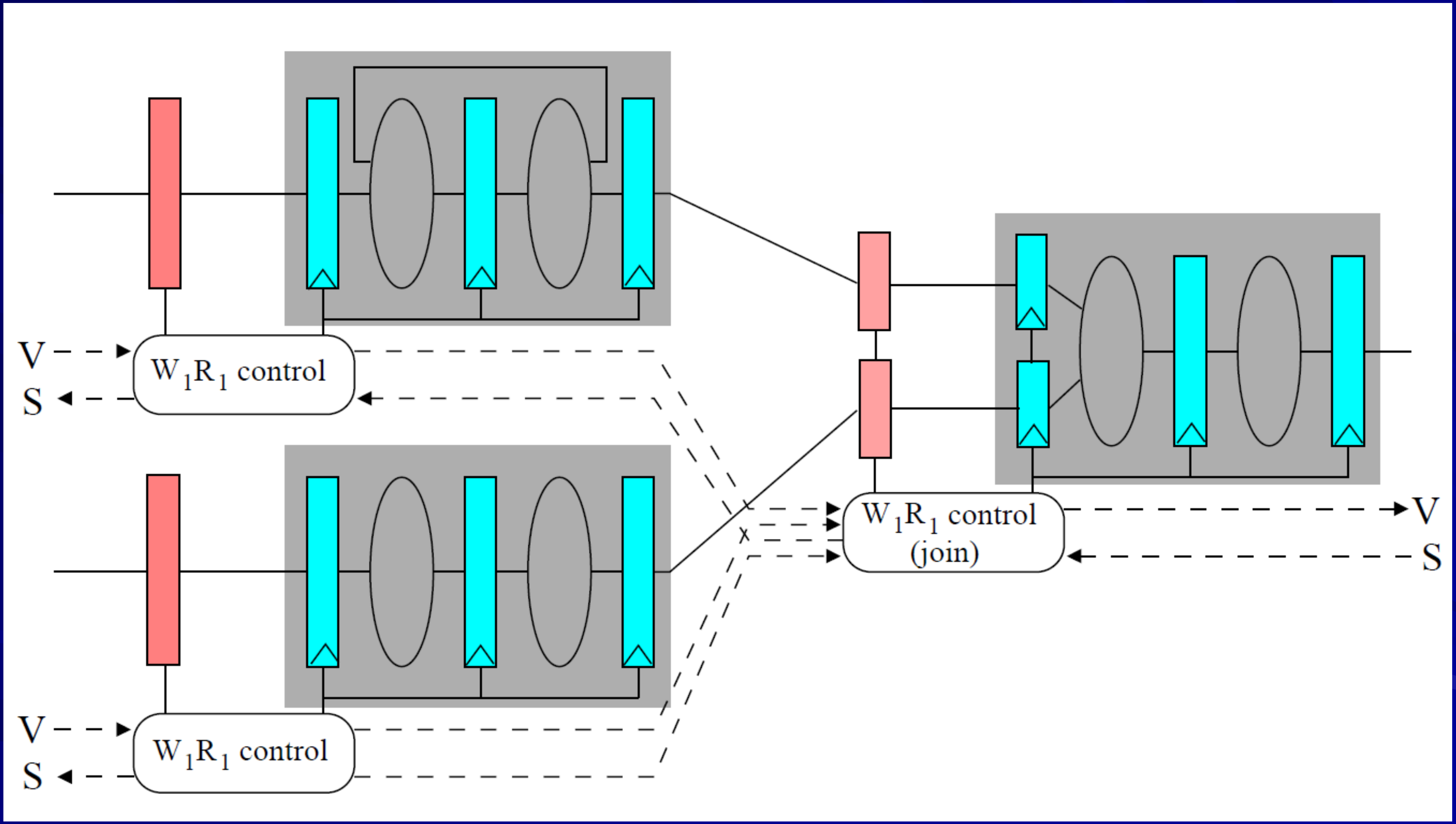
Eager fork (another implementation)



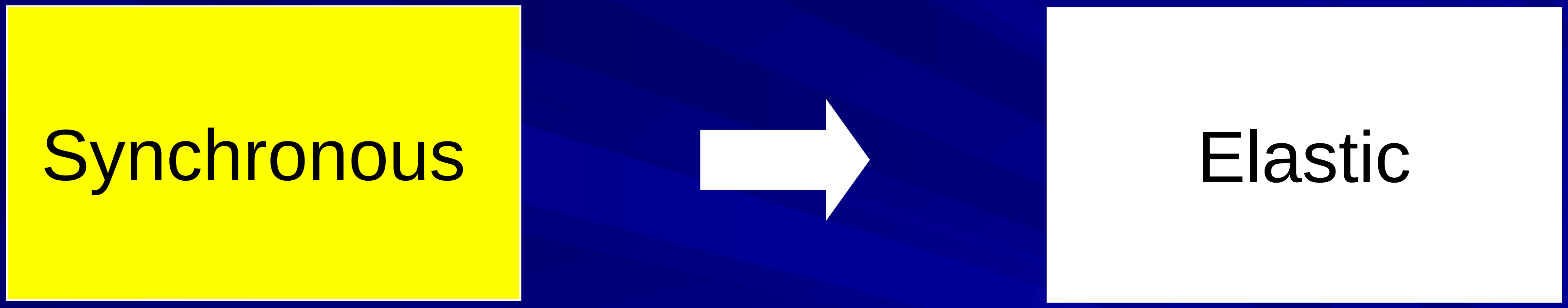
Variable Latency Units



Coarse grain control

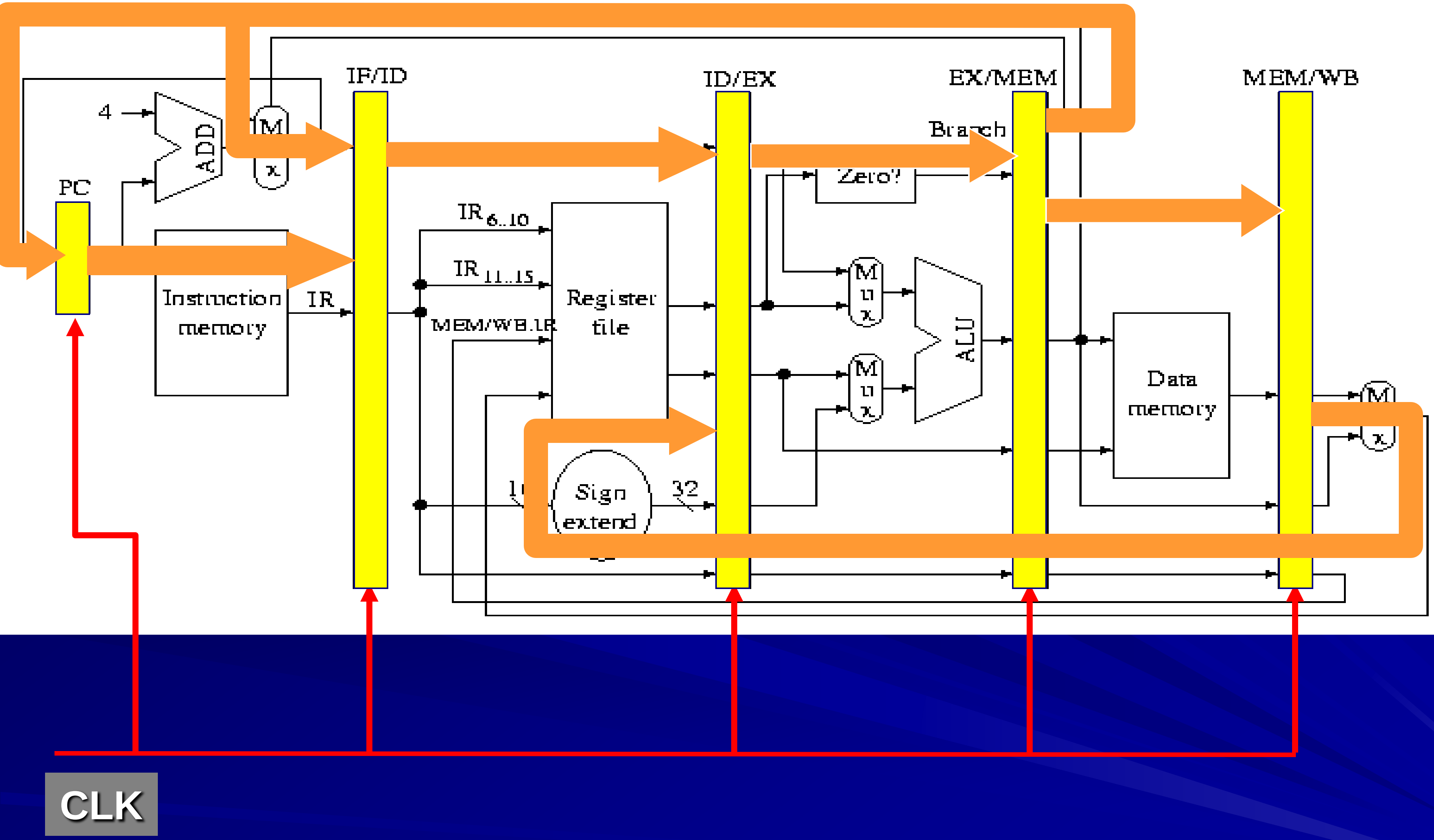


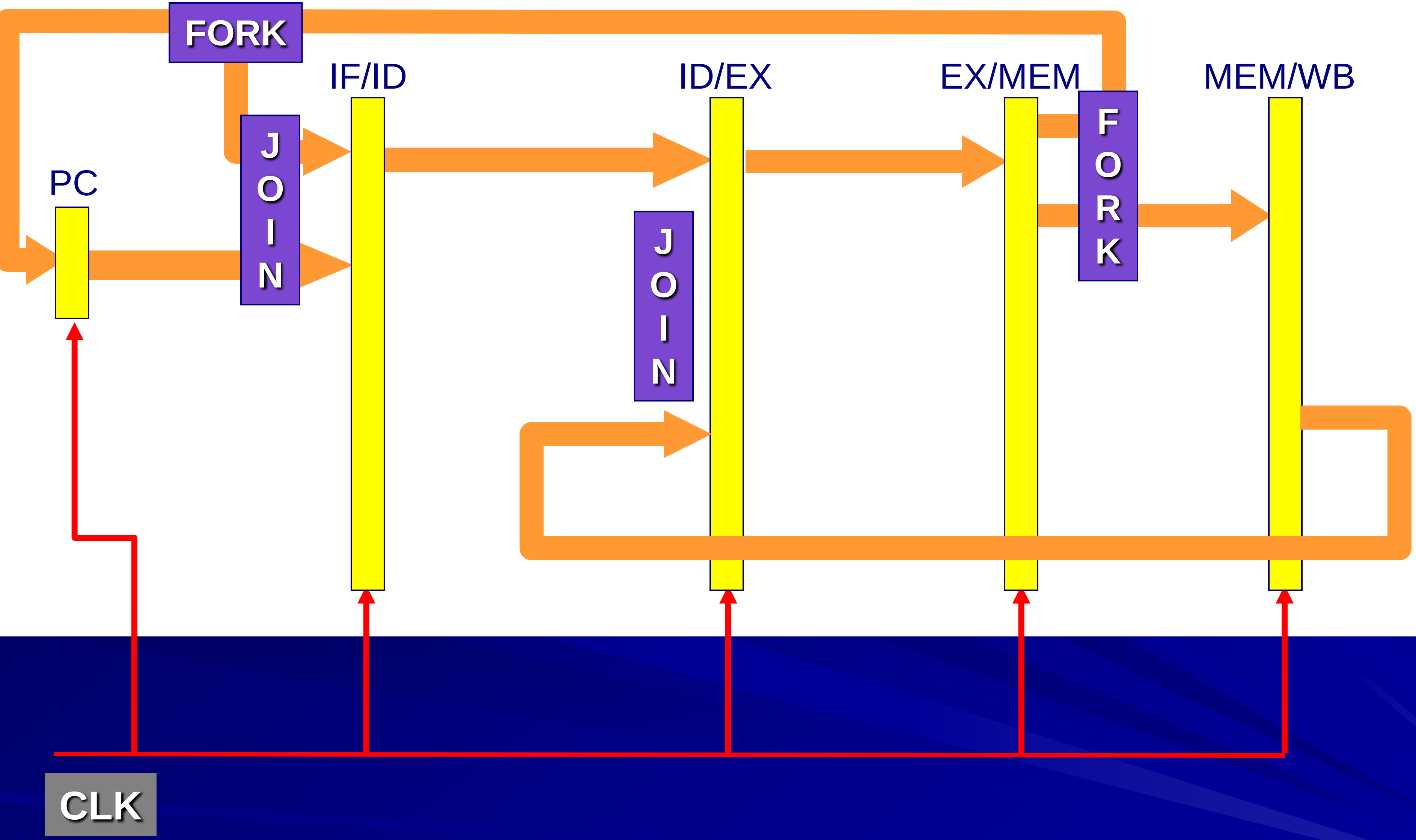
Elasticization

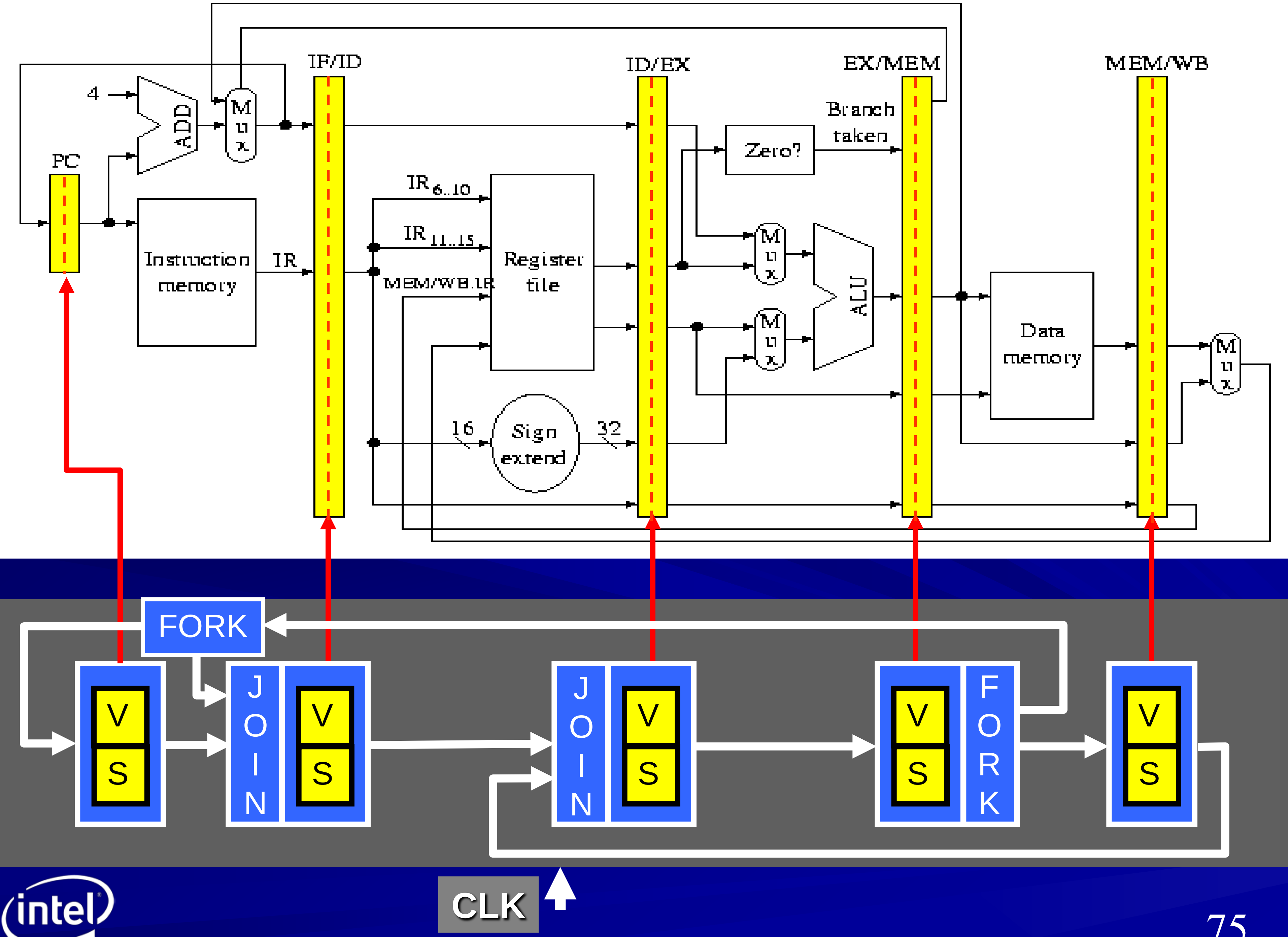


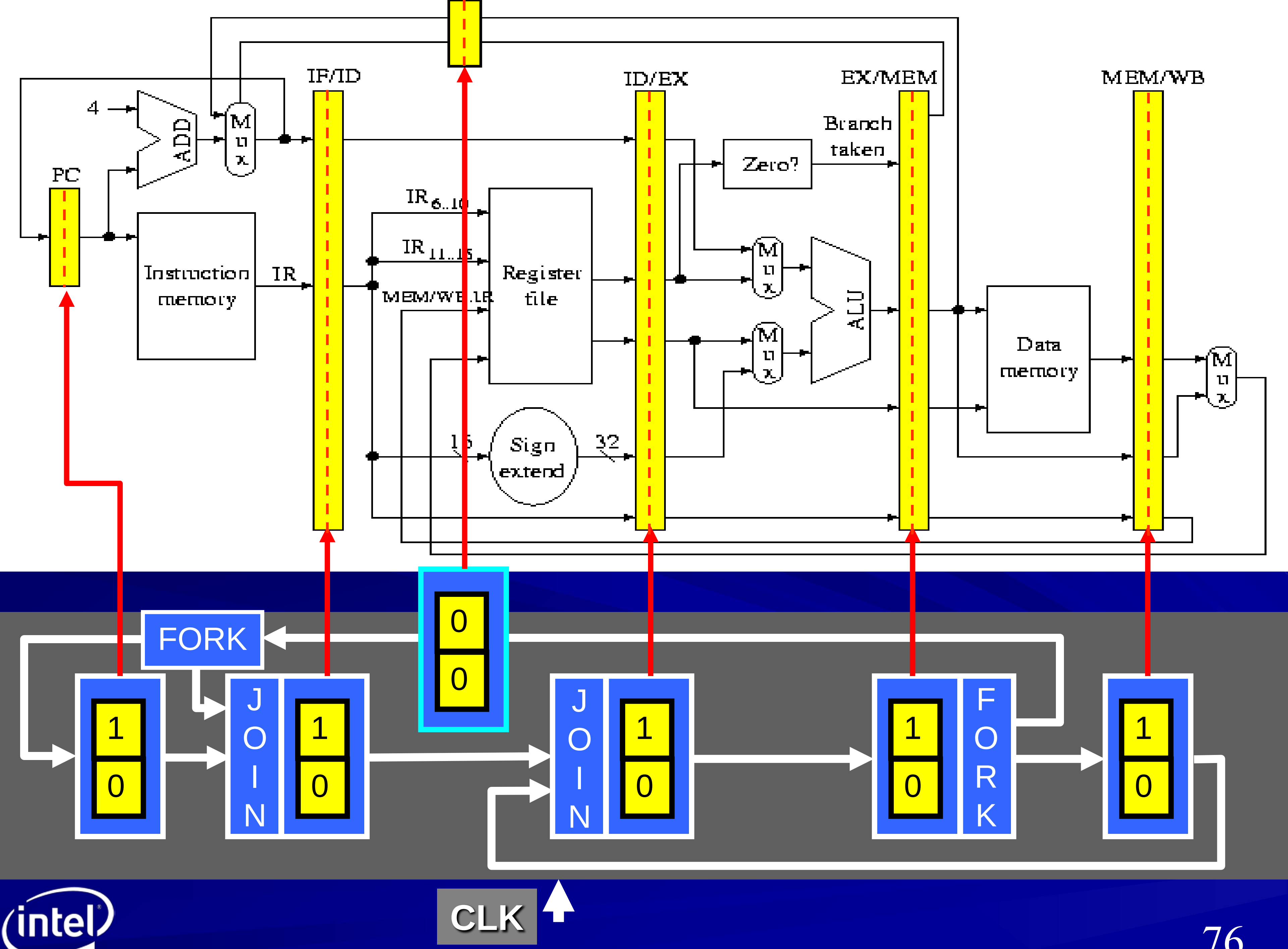
Synchronous

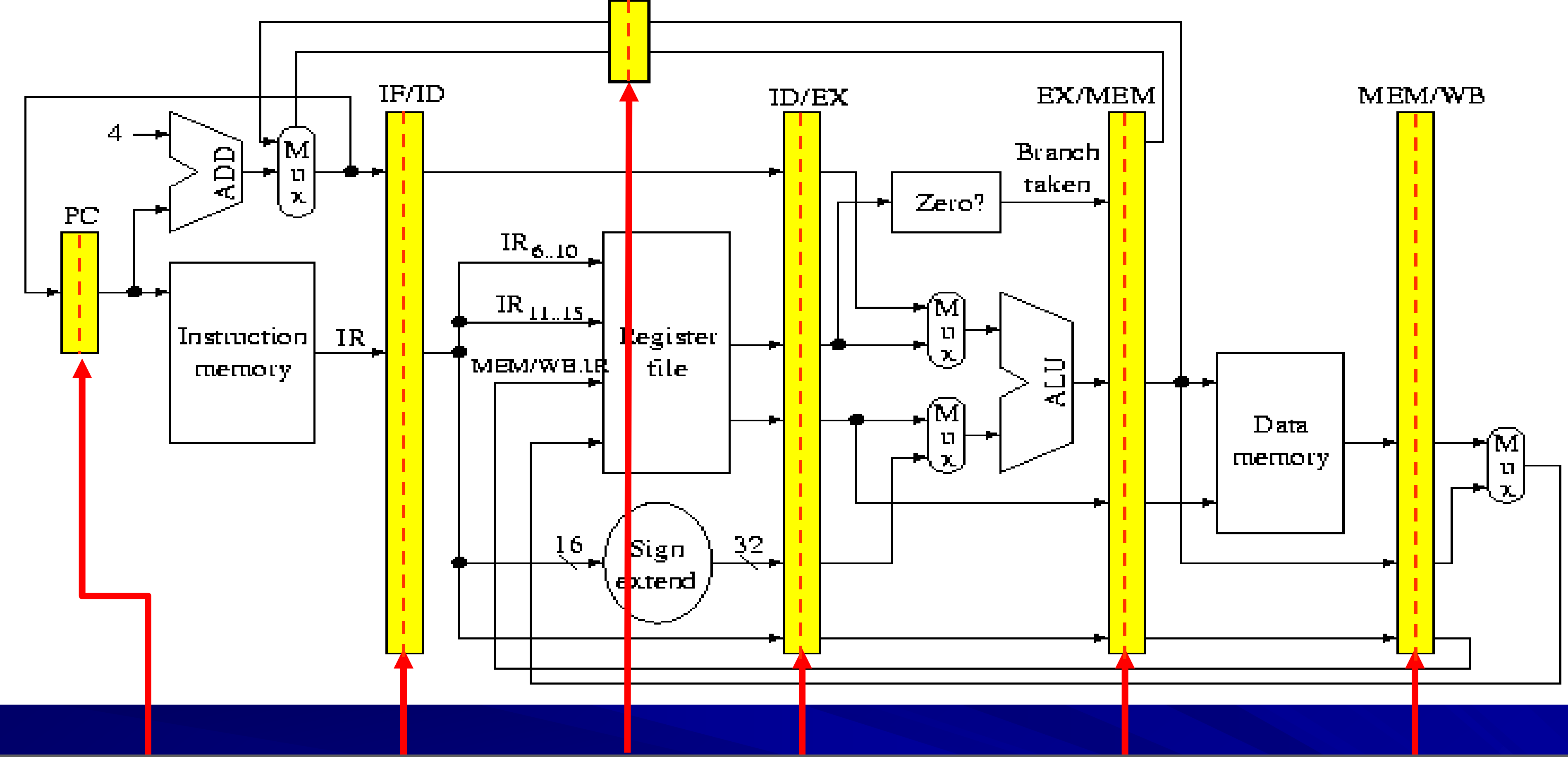
Elastic


IF/ID
ID/EX
EX/MEM
MEM/WB
4
ADD
M
x
PC
Branch
Zero?
IR 6..10
IR 11..15
MEM/WB.IR
Instruction memory
IR
Register file
M
u
x
ALU
Data memory
Sign extend
16
32
CLK


FORK
IF/ID
ID/EX
EX/MEM
MEM/WB
PC
JOIN
JOIN
FORK
CLK


IF/ID
ID/EX
EX/MEM
MEM/WB
4
ADD
M u x
PC
Instruction memory
IR
IR 6..10
IR 11..15
MEM/WB.IR
Register file
16
Sign extend
32
Zero?
Branch taken
M u x
M u x
ALU
Data memory
M u x
FORK
V
S
J O I N
V
S
J O I N
V
S
V
S
F O R K
V
S
CLK


IF/ID
ID/EX
EX/MEM
MEM/WB
4
ADD
M u x
PC
Instruction memory
IR
$IR_{6..10}$
$IR_{11..15}$
MEM/WB.IR
Register file
Zero?
Branch taken
M u x
M u x
ALU
Data memory
M u x
16
Sign extend
32
0
0
FORK
1
0
JOIN
1
0
JOIN
1
0
1
0
FORK
1
0
CLK



Elastic control layer

Generation of gated clocks

CLK

Equivalence

Synchronous: stream of data

D: a b c d e d f g h i j …

SELF: elastic stream of data

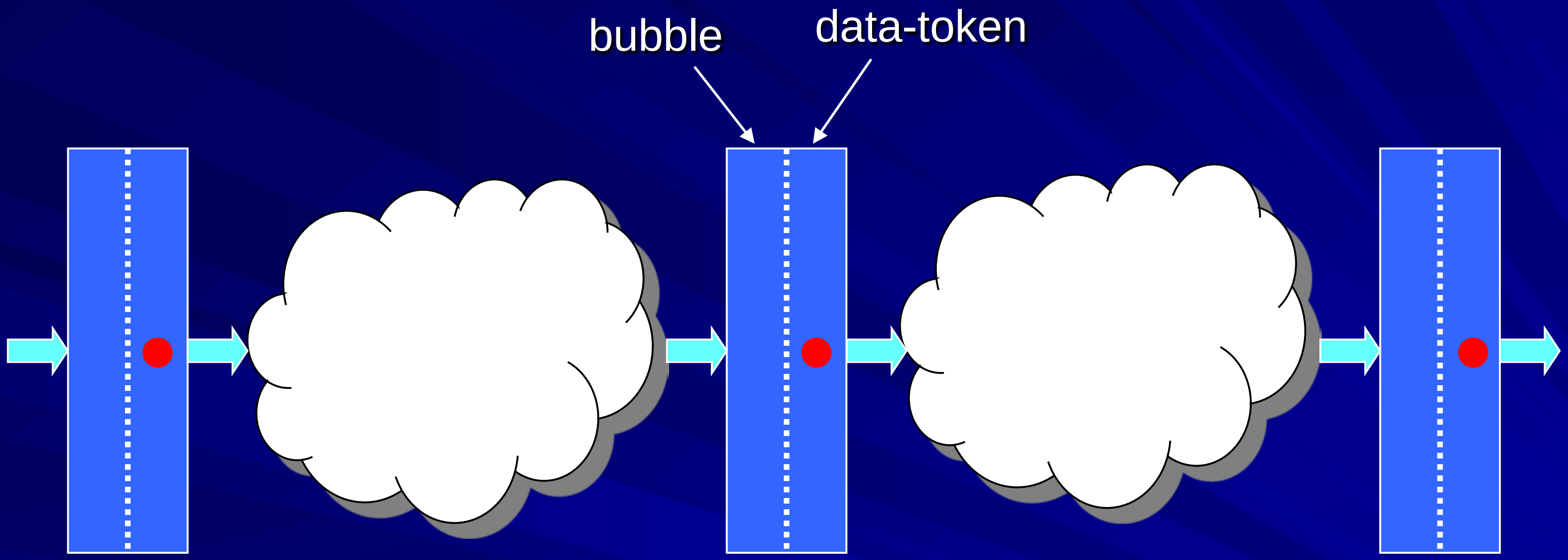
D: a * b * * c d e * d f * g h * * i j …
V: 1 0 1 1 0 1 1 1 1 1 1 1 1 1 0 1 1 1 …
S: 0 0 0 1 0 0 0 0 1 0 0 1 0 0 0 1 0 0 …

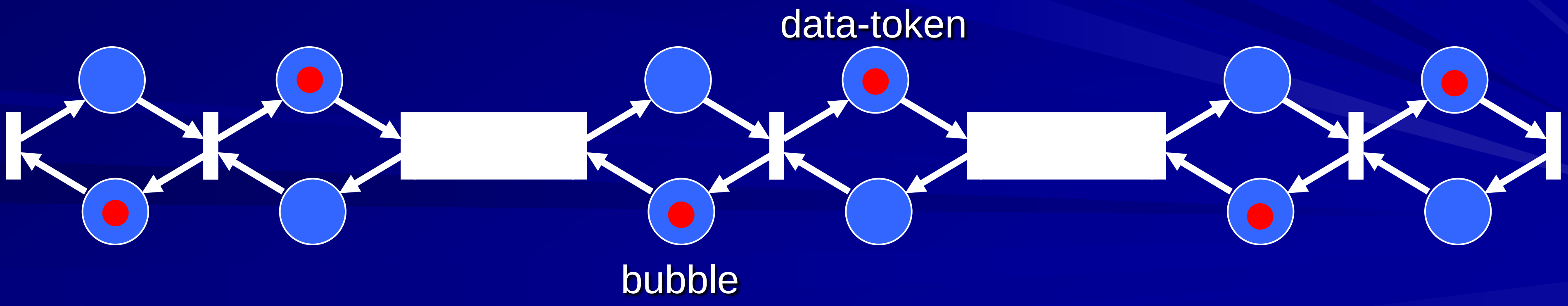
Transfer sub-stream = original stream

Called: transfer equivalence, flow equivalence, or latency equivalence

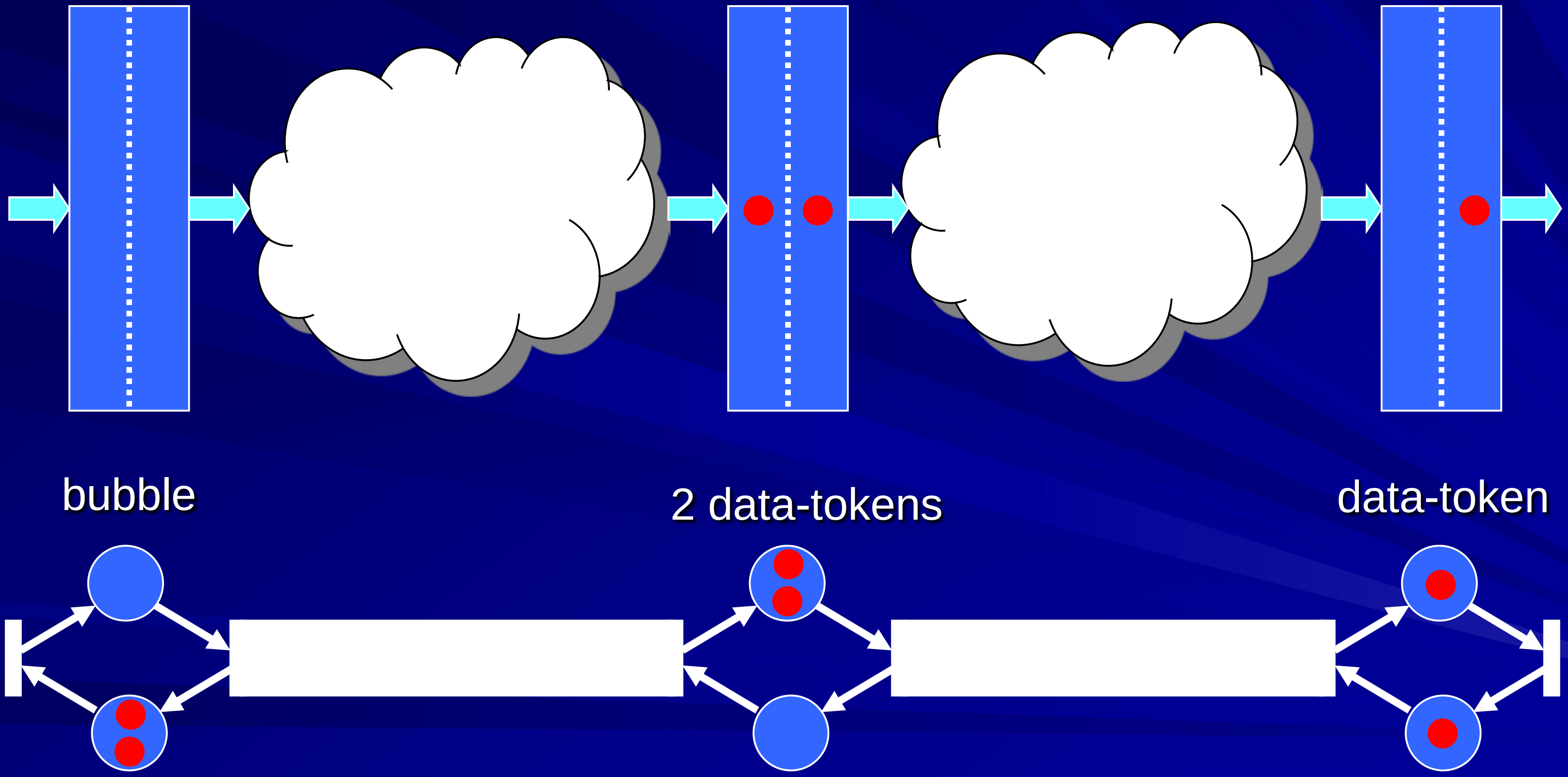
Marked Graph models of elastic systems

Modelling elastic control with Petri nets



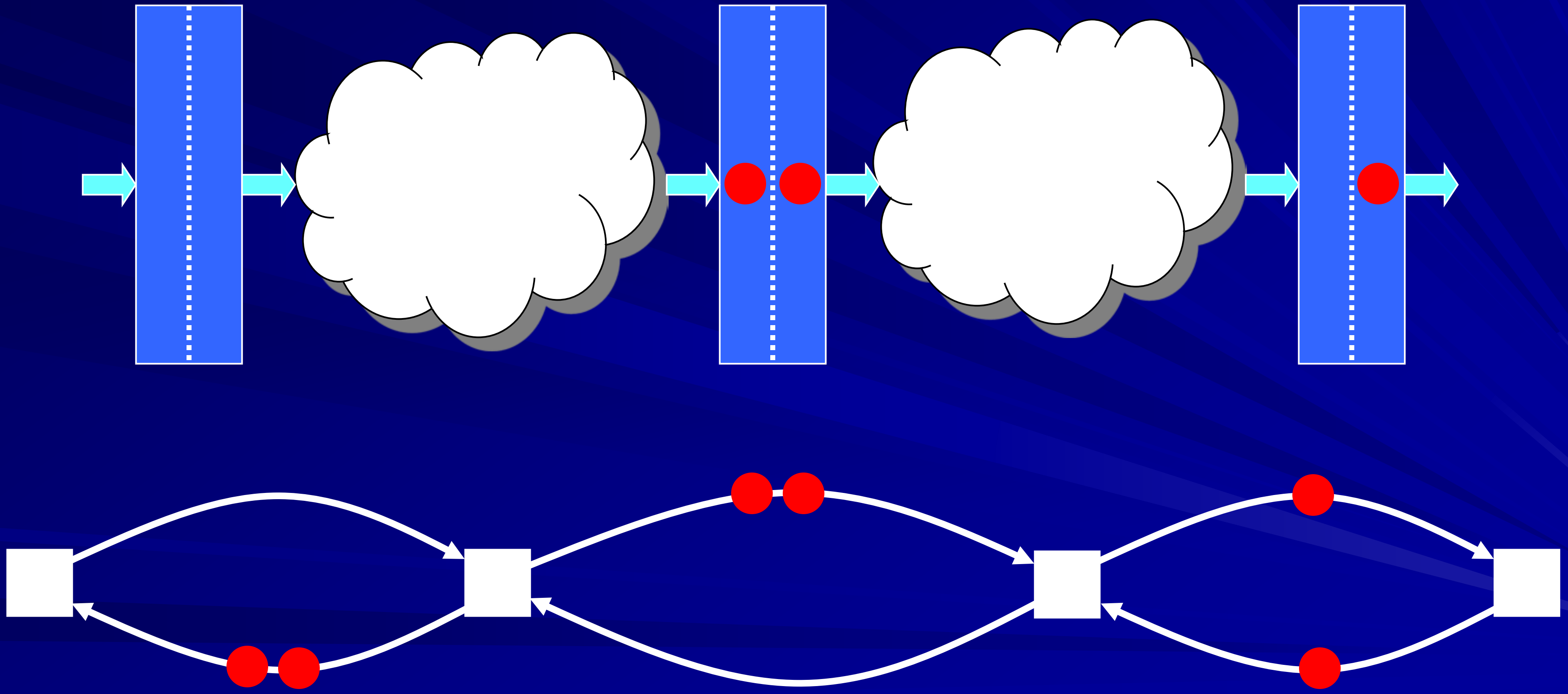


Modelling elastic control with Petri nets

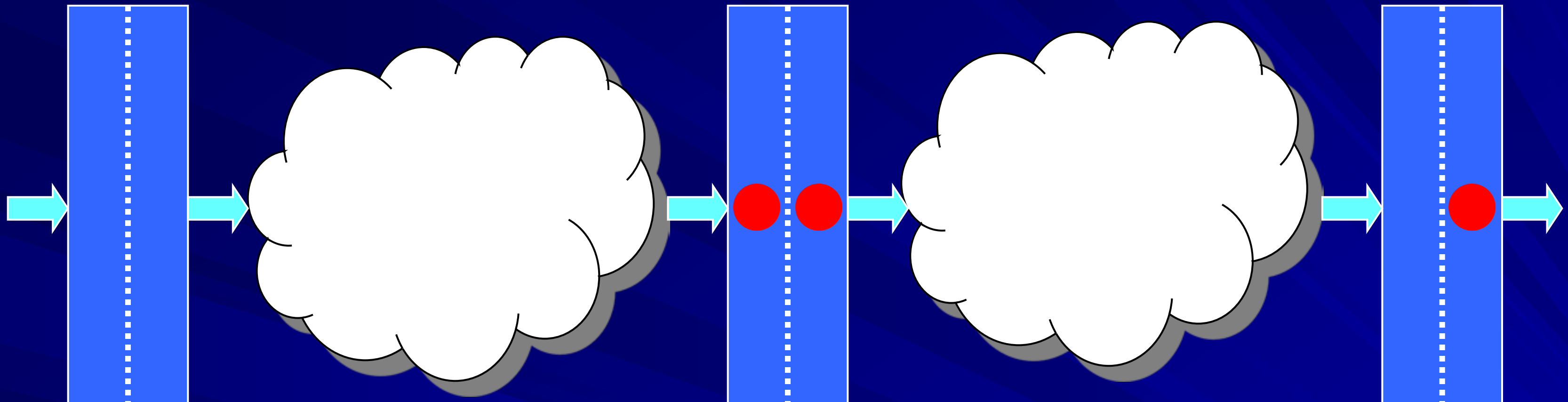


Hiding internal transitions of elastic buffers

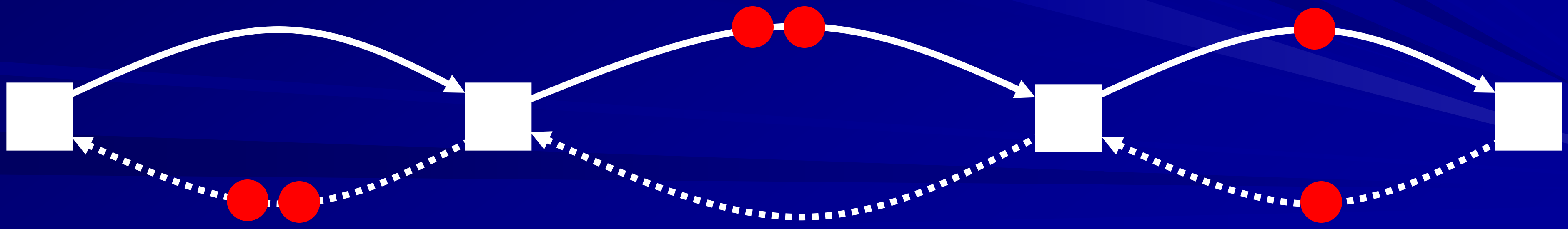
Modelling elastic control with Marked Graphs



Modelling elastic control with Marked Graphs

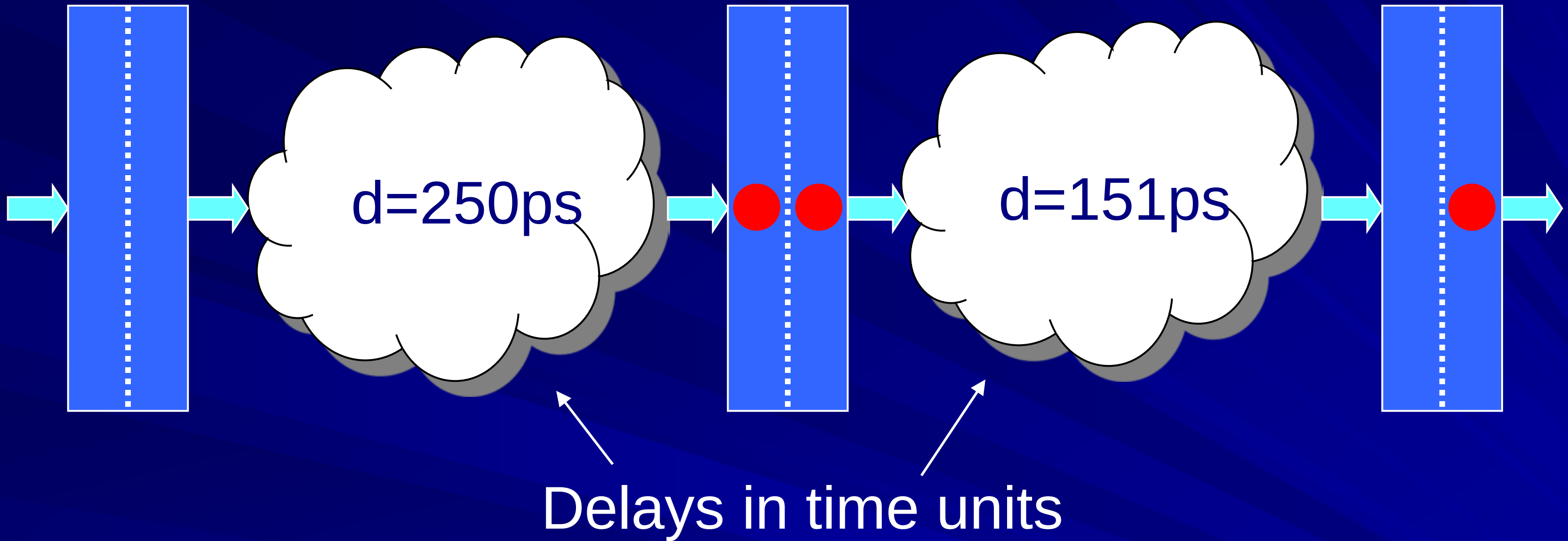


Forward
(Valid or Request)

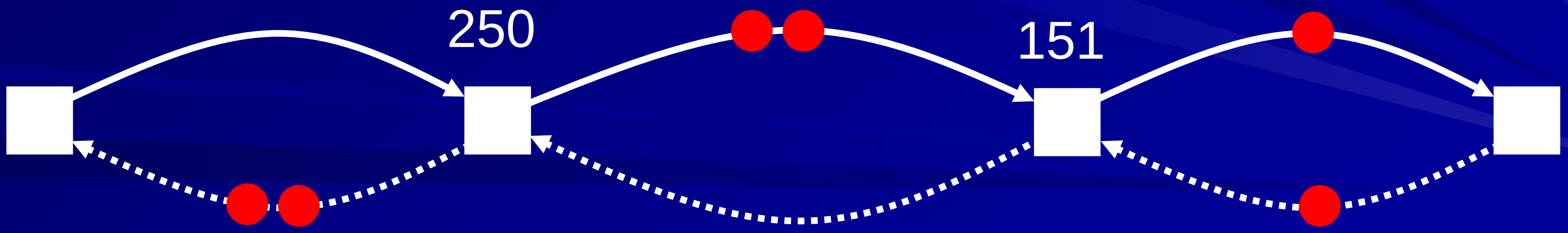


Backward
(Stop or Acknowledgement)

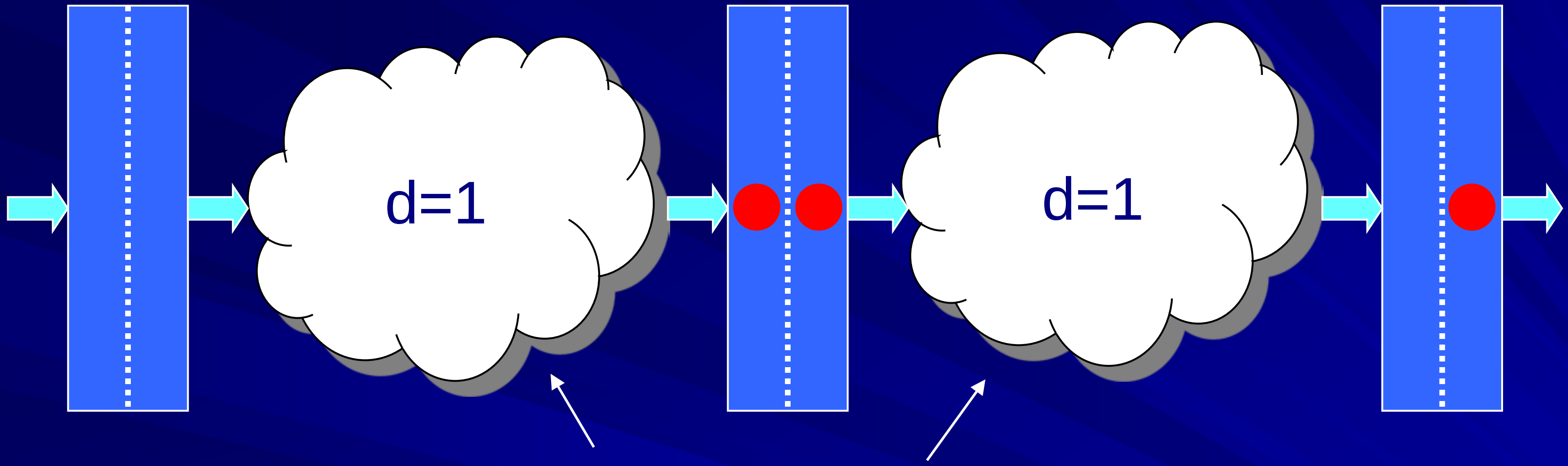
Elastic control with Timed Marked Graphs. Continuous time = asynchronous



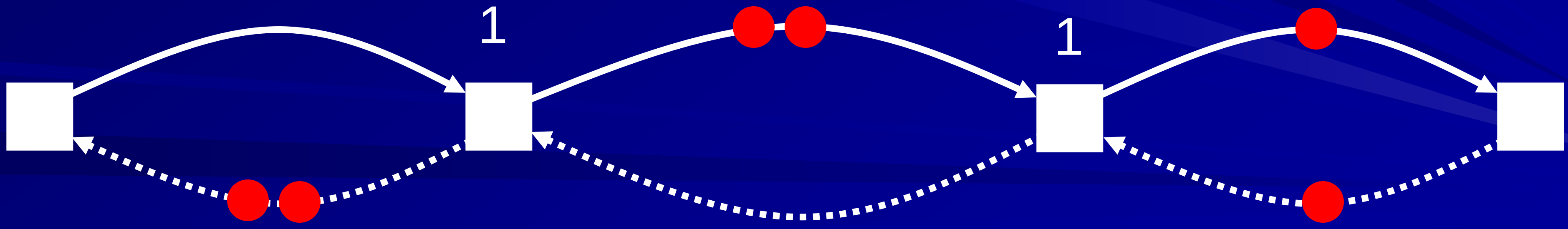
Delays in time units



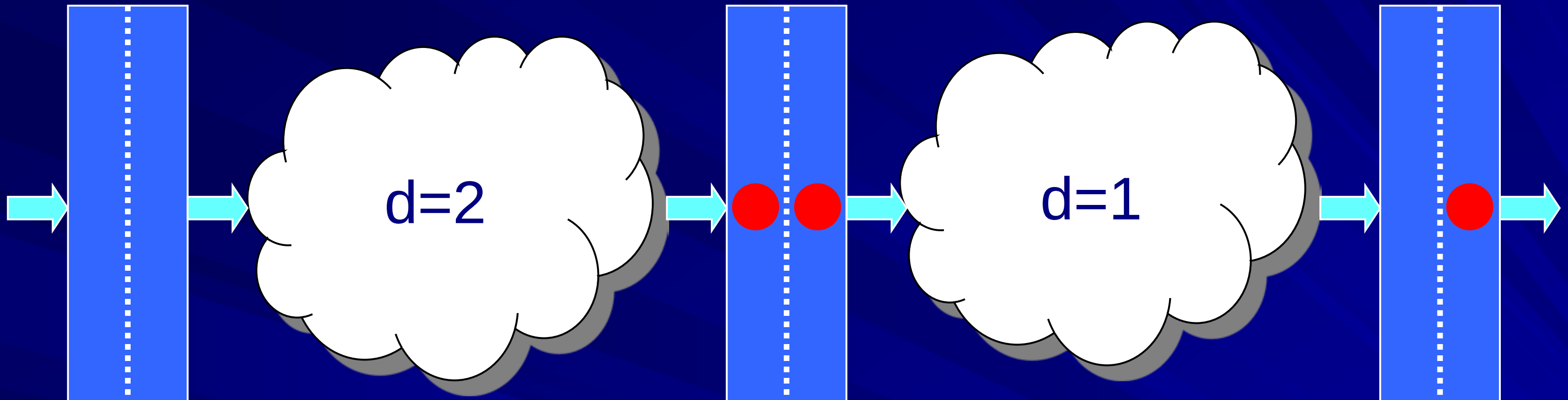
Elastic control with Timed Marked Graphs. Discrete time = synchronous elastic

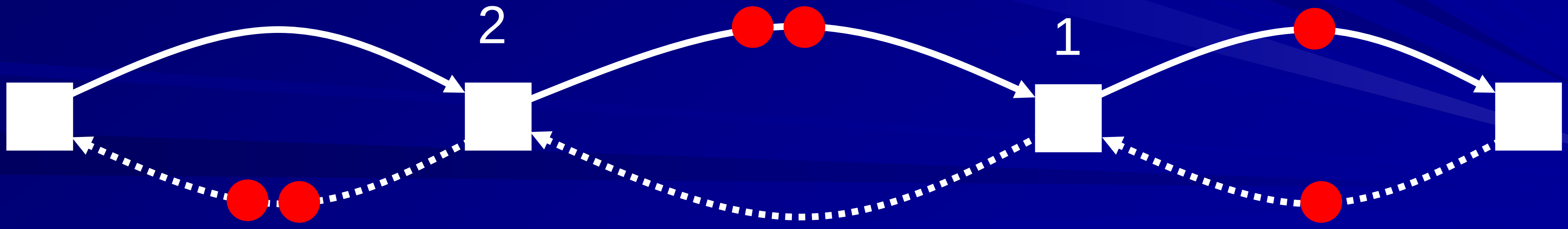


Latencies in clock cycles

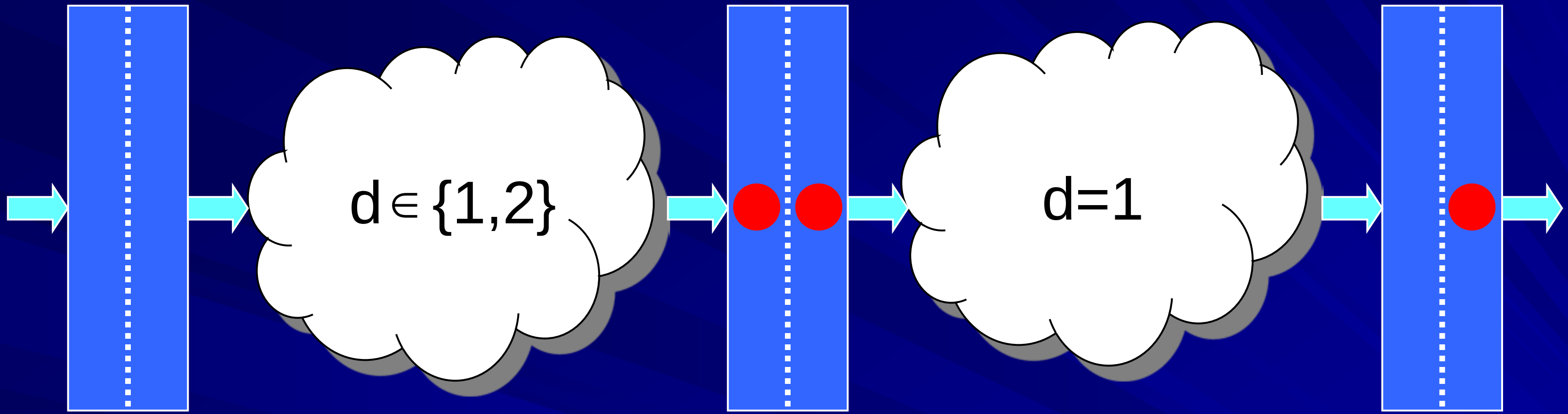


Elastic control with Timed Marked Graphs. Discrete time. Multi-cycle operation

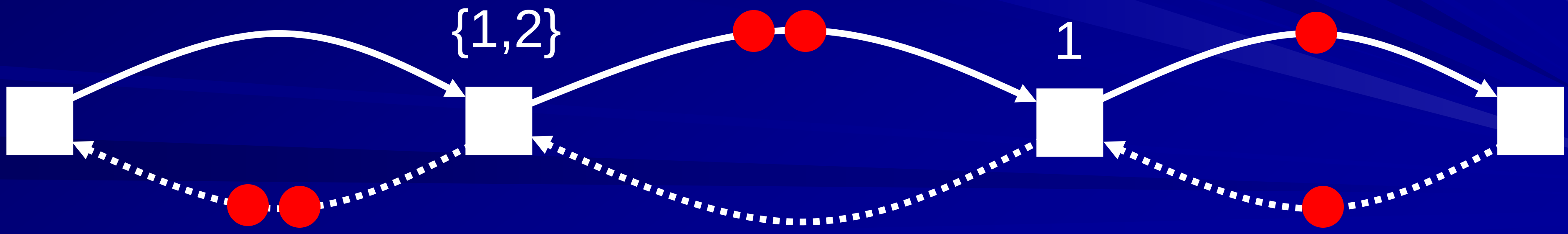




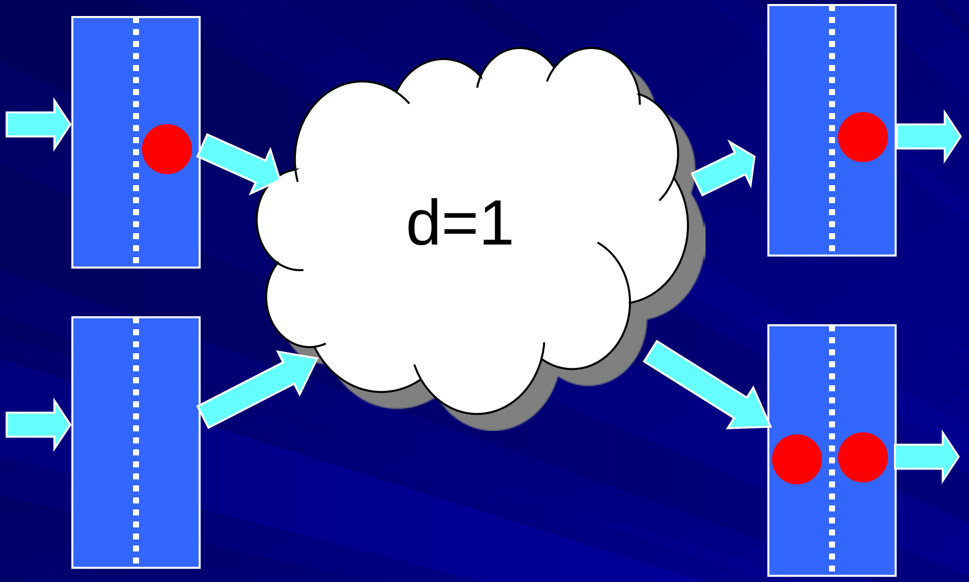
Elastic control with Timed Marked Graphs. Discrete time. Variable latency operation

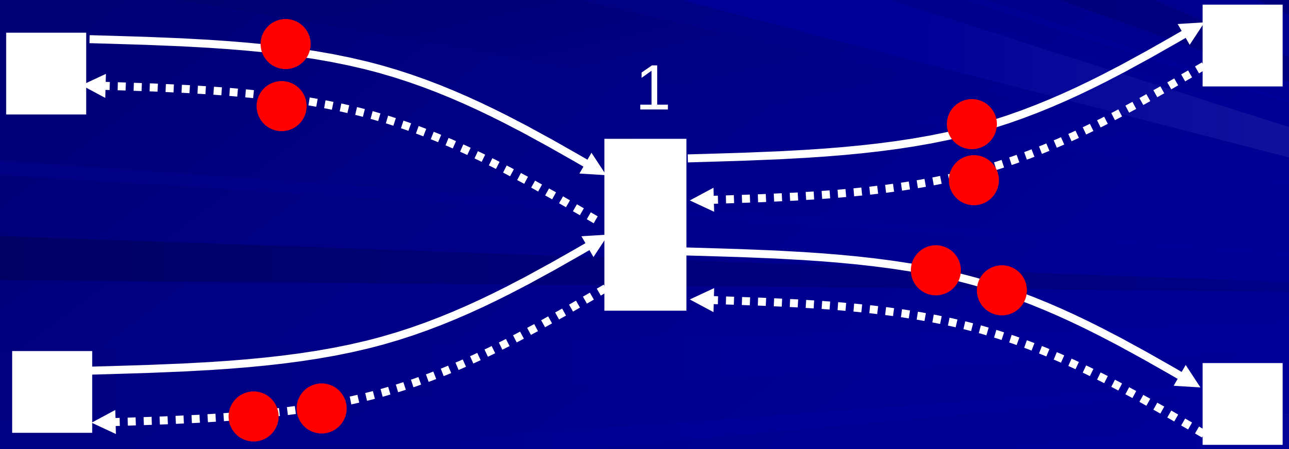


e.g. discrete probabilistic distribution:
average latency 0.8*1 + 0.2*2 = 1.2

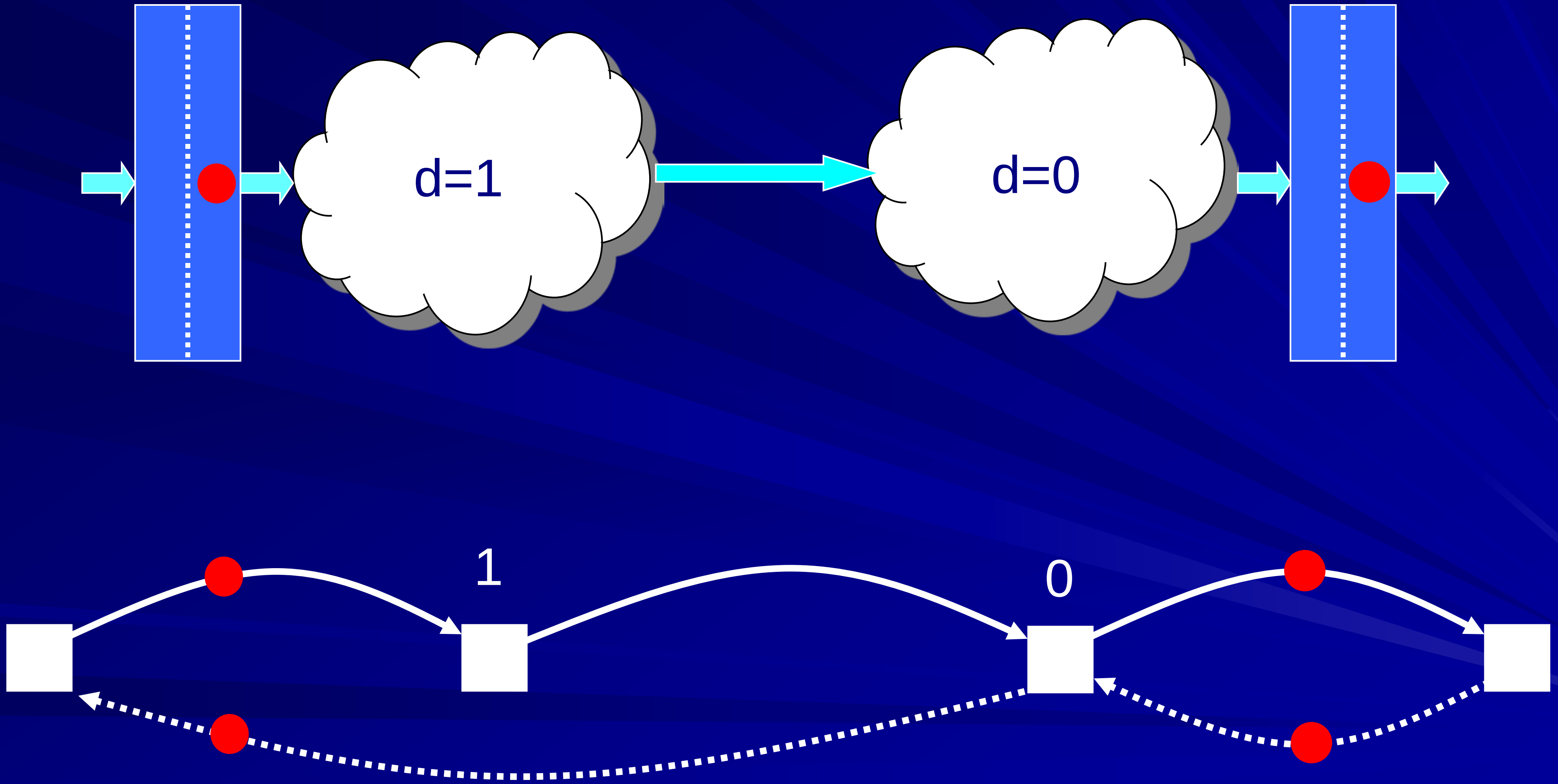


Modeling forks and joins





Modelling combinational elastic blocks



Elastic Marked Graphs

- An Elastic Marked Graph (EMG) is a Timed MG such that for any arc *a* there exists a complementary arc *a'* satisfying the following condition

  $\bullet a = a' \bullet$ and $\bullet a' = a \bullet$

- Initial number of tokens on *a* and *a'* $(M_0(a)+M_0(a'))$ = capacity of the corresponding elastic buffer

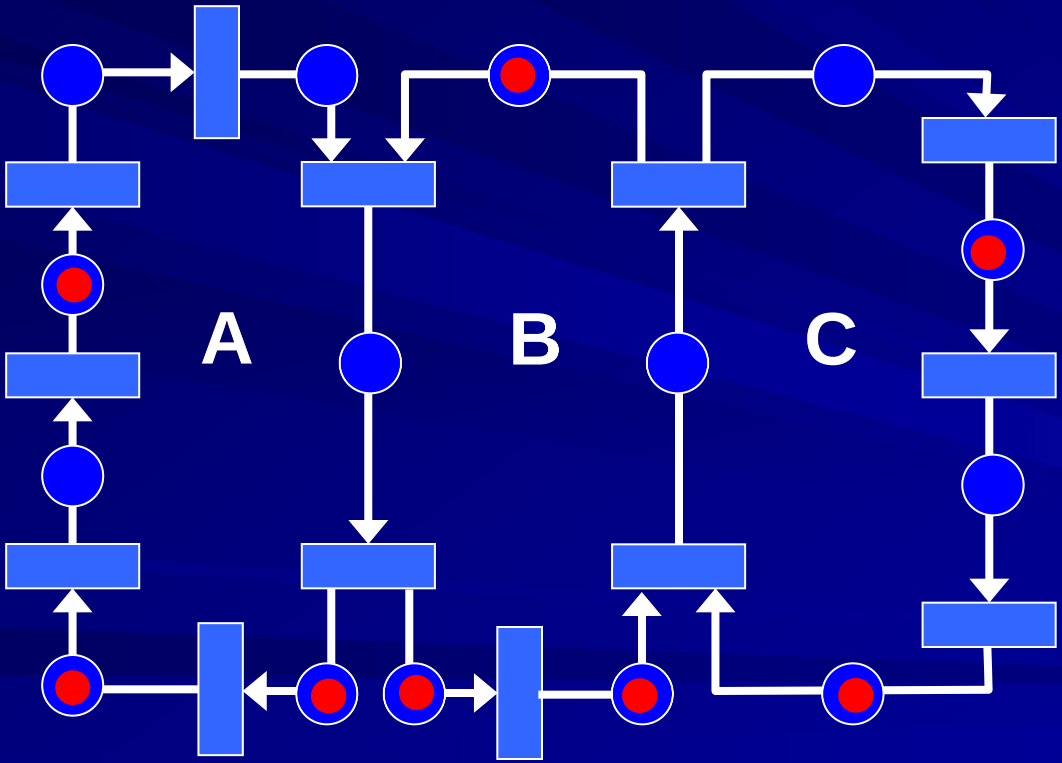
- Similar forms of “pipelined” Petri Nets and Marked Graphs have been previously used for modeling pipelining in HW and SW (e.g. Patil 1974; Tsirlin, Rosenblum 1982)

Reminder: Performance analysis of Marked graphs

Th = operations / cycle = number of firings per time unit

The throughput is given by the minimum mean-weight cycle

Th=min(Th(A), Th(B), Th(C))=2/5



Th(A)=3/7

Th(B)=3/5

Th(C)=2/5

Efficient algorithms: (Karp 1978), (Dasdan,Gupta 1998)

Early evaluation

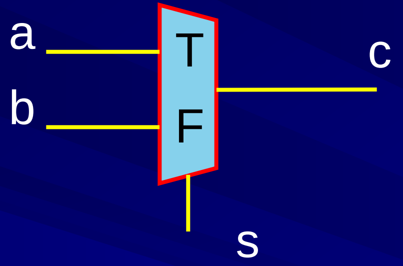
- Naïve solution: introduce choice places
  - issue tokens at choice node only into one (some) relevant path
  - problem: tokens can arrive to merge nodes out-of-order
    later token can overpass the earlier one

- Solution: change enabling rule
  - early evaluation
  - issue negative tokens to input places without tokens, i.e. keep the same firing rule
  - Add symmetric sub-channels with negative tokens
  - Negative tokens kill positive tokens when meet

- Two related problems:
  Early evaluation and Exceptions (how to kill a data-token)

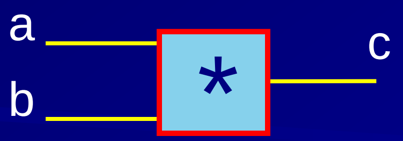
Examples of early evaluation

MULTIPLEXOR



if s = T then c := a   -- don’t wait for b
else c := b   -- don’t wait for a

MULTIPLIER



if a = 0 then c := 0   -- don’t wait for b

Related work

- Petri nets
  - Extensions to model OR causality
    Kishinevsky et al. Change Diagrams [e.g. book of 1994]
    Yakovlev et al. Causal Nets 1996

- Asynchronous systems
  - Reese et al 2002: Early evaluation
  - Brej 2003: Early evaluation with anti-tokens
  - Ampalan & Singh 2006: preemption using anti-tokens

Dual Marked Graph

- Marking: Arcs (places) –> **Z**
  (allow negative markings)
- Some nodes are labeled as early-enabling

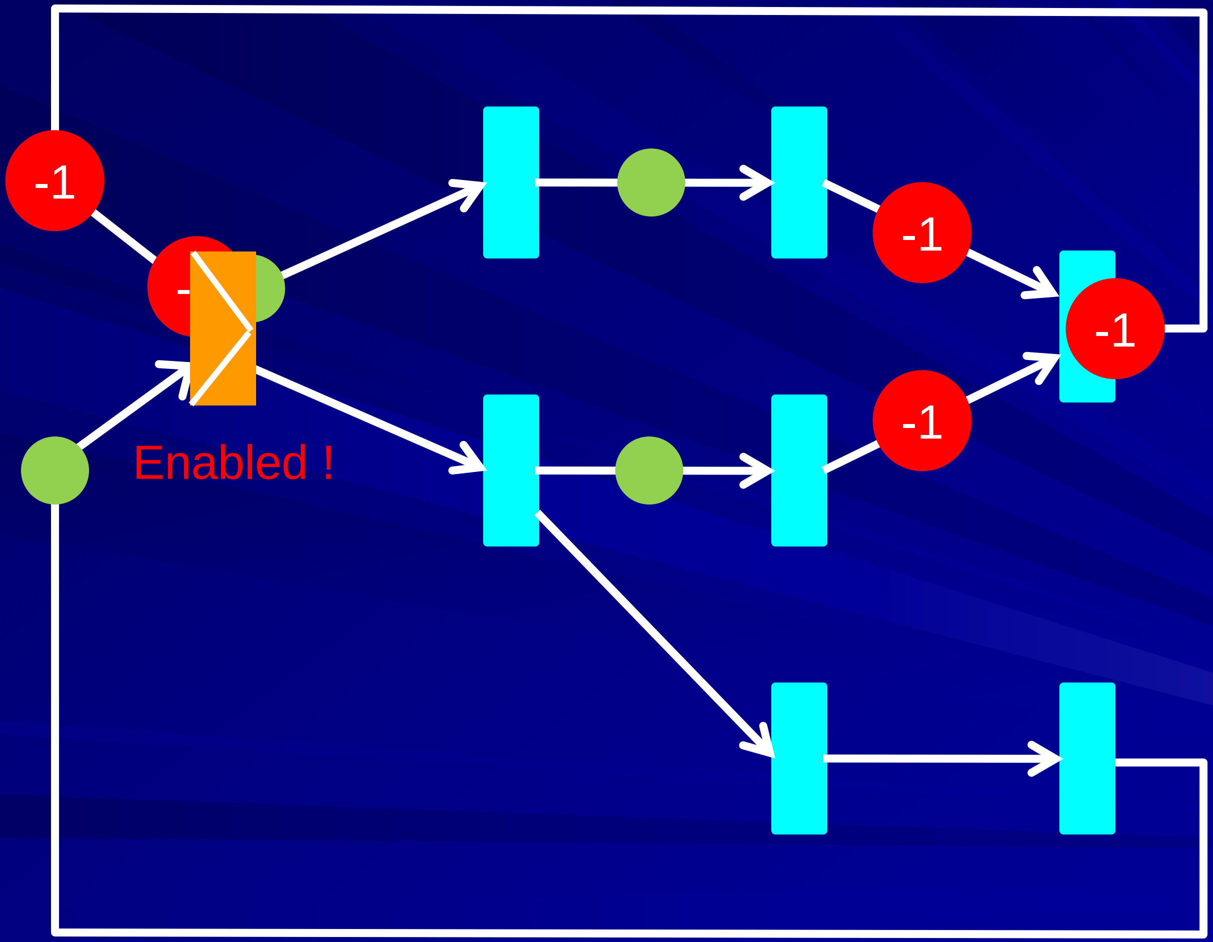
- Enabling rules for a node:
  - Positive enabling: *M(a)* > 0 for <u>every input</u> arc
  - Early enabling (for early enabling nodes):
    M(a) > 0 for <u>some input</u> arcs
  - Negative enabling: *M(a)* < 0 for <u>every output</u> arc

- Firing rule: the same as in regular MG

Dual Marked Graphs

- Early enabling can be associated with an external guard that depends on data variables (e.g., a select signal of a multiplexor)
- Actual enabling guards are abstracted away (unless needed)
- Anti-token generation: When an early enabled node fires, it generates anti-tokens in the predecessor arcs that had no tokens
- Anti-token propagation counterflow: When negative enabled node fires, it propagates the anti-tokens from the successor to the predecessor arcs

Dual Marked Graph model



Passive anti-token

- Passive DMG = version of DMG without negative enabling
- Negative tokens can only be generated due to early enabling, but cannot propagate
- Let D be a strongly connected DMG such that all cycles have positive cumulative marking

  Let D_p be a corresponding passive DMG.

  If environment (consumers) never generate negative tokens, and there are no multi-cycle operations then throughput (D) = throughput (D_p)

  – If capacity of input places for early enabling transitions is unlimited, then active anti-tokens do not improve performance

  – Active anti-tokens reduce activity in the data-path (good for power reduction)

Properties of DMGs

- Firing invariant: Let node *n* be simultaneously positive (early) and negative enabled in marking *M.*
  Let M_1 be the result of firing *n* from *M* due to positive (early) enabling.
  Let M_2 be the result of firing *n* from *M* due to negative enabling.
  Then, $M_1 = M_2$

- Token preservation. Let *c* be a cycle of a strongly connected DMG with initial marking M_0.
  For every reachable marking $M : M(c) = M_0(c)$

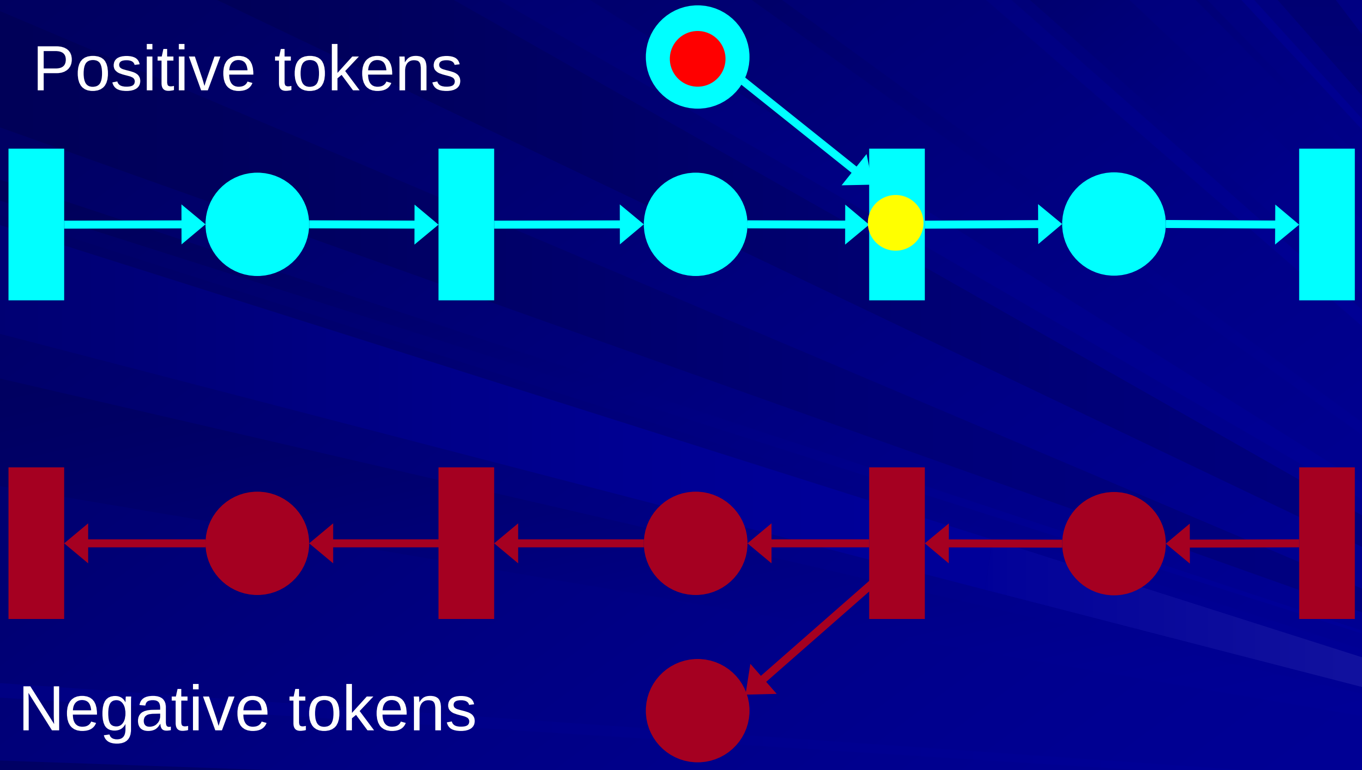
- Liveness. A strongly connected passive DMG is live iff for every cycle *c:* $M(c) > 0$.
  - For DMGs this is a sufficient condition of liveness
  - It is also a necessary condition for positive liveness

- Repetitive behavior. In a SC DMG: a firing sequence *s* from *M* leads to the same marking iff every node fires in *s* the same number of times

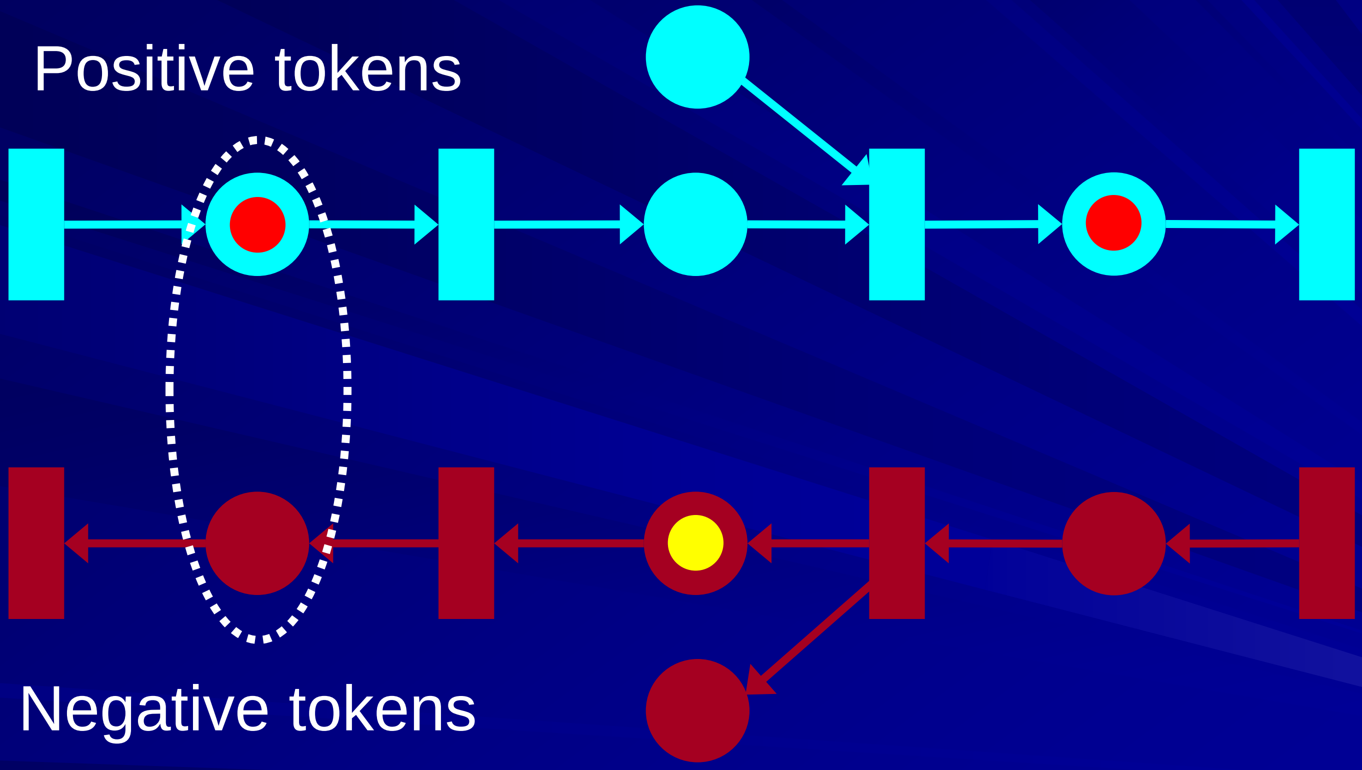
- DMGs have properties similar to regular MGs

Implementing early enabling

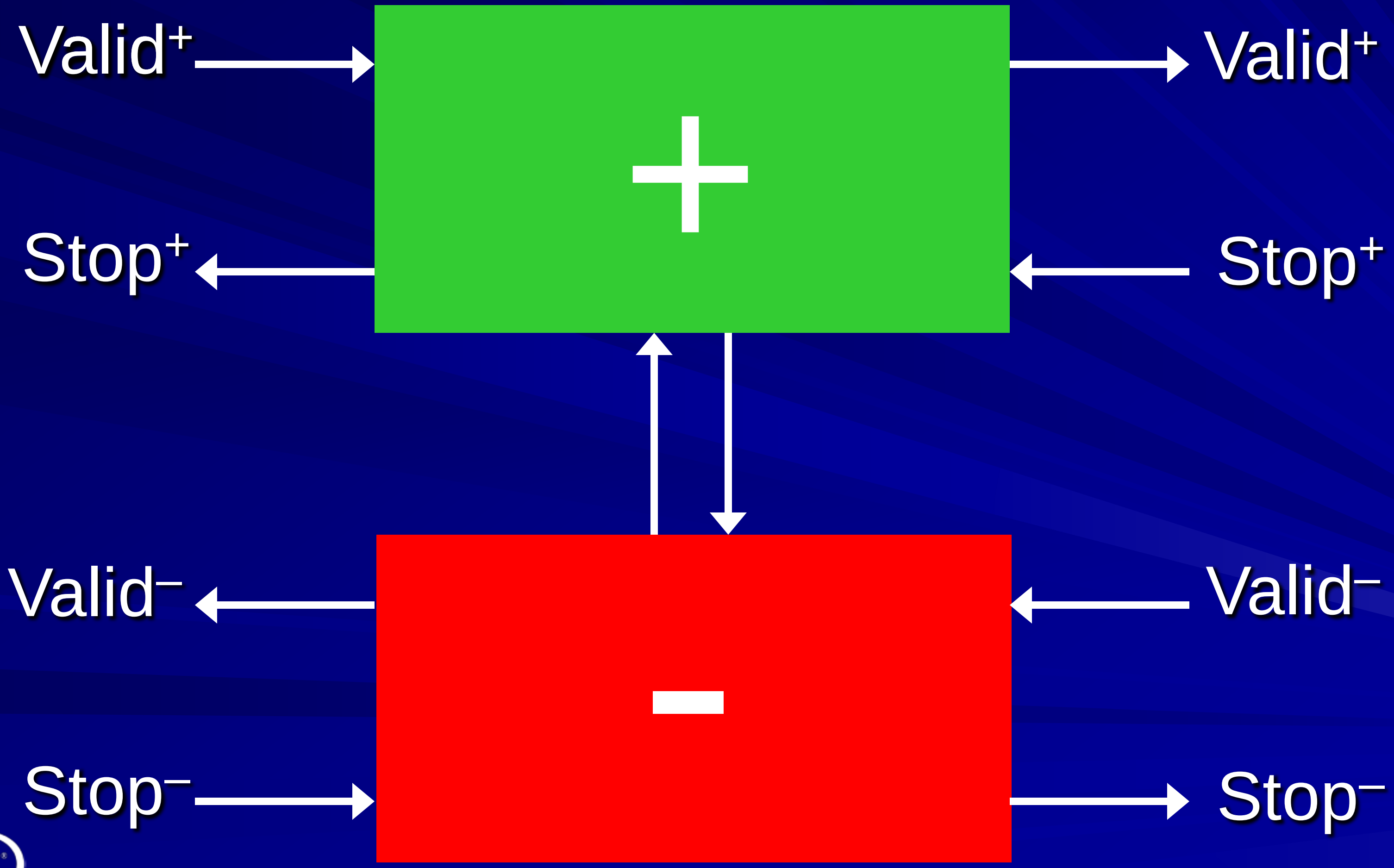
How to implement anti-tokens ?



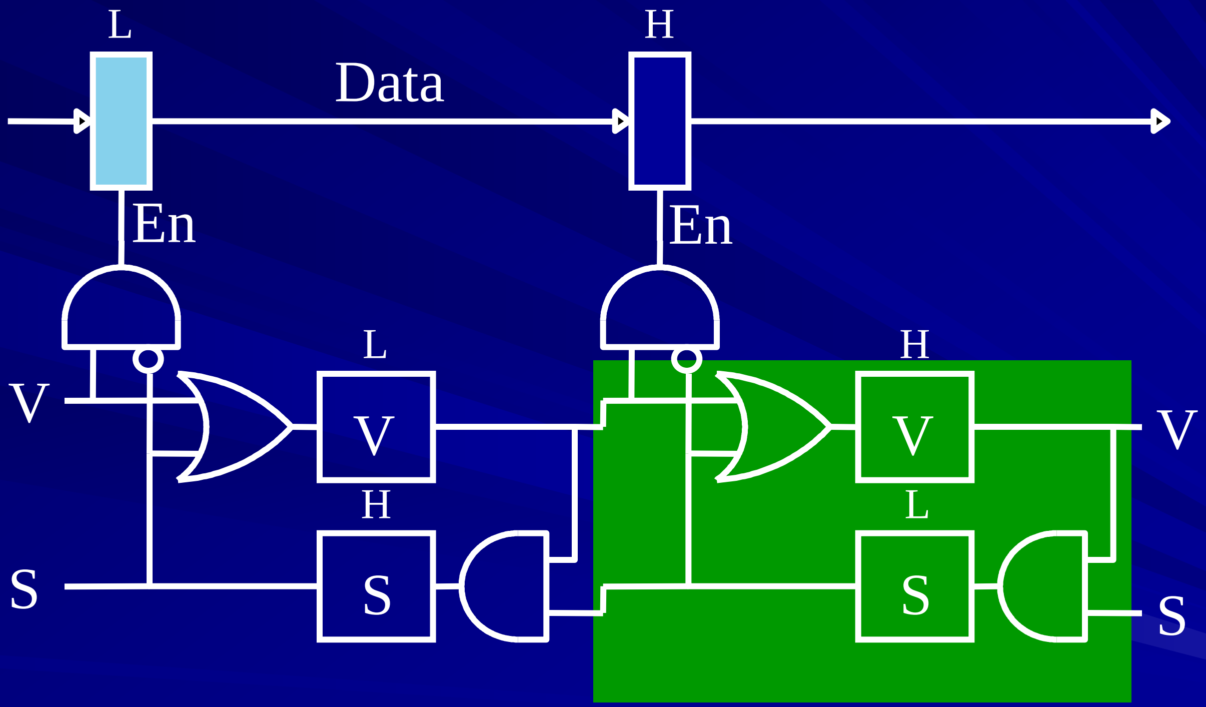
How to implement anti-tokens ?



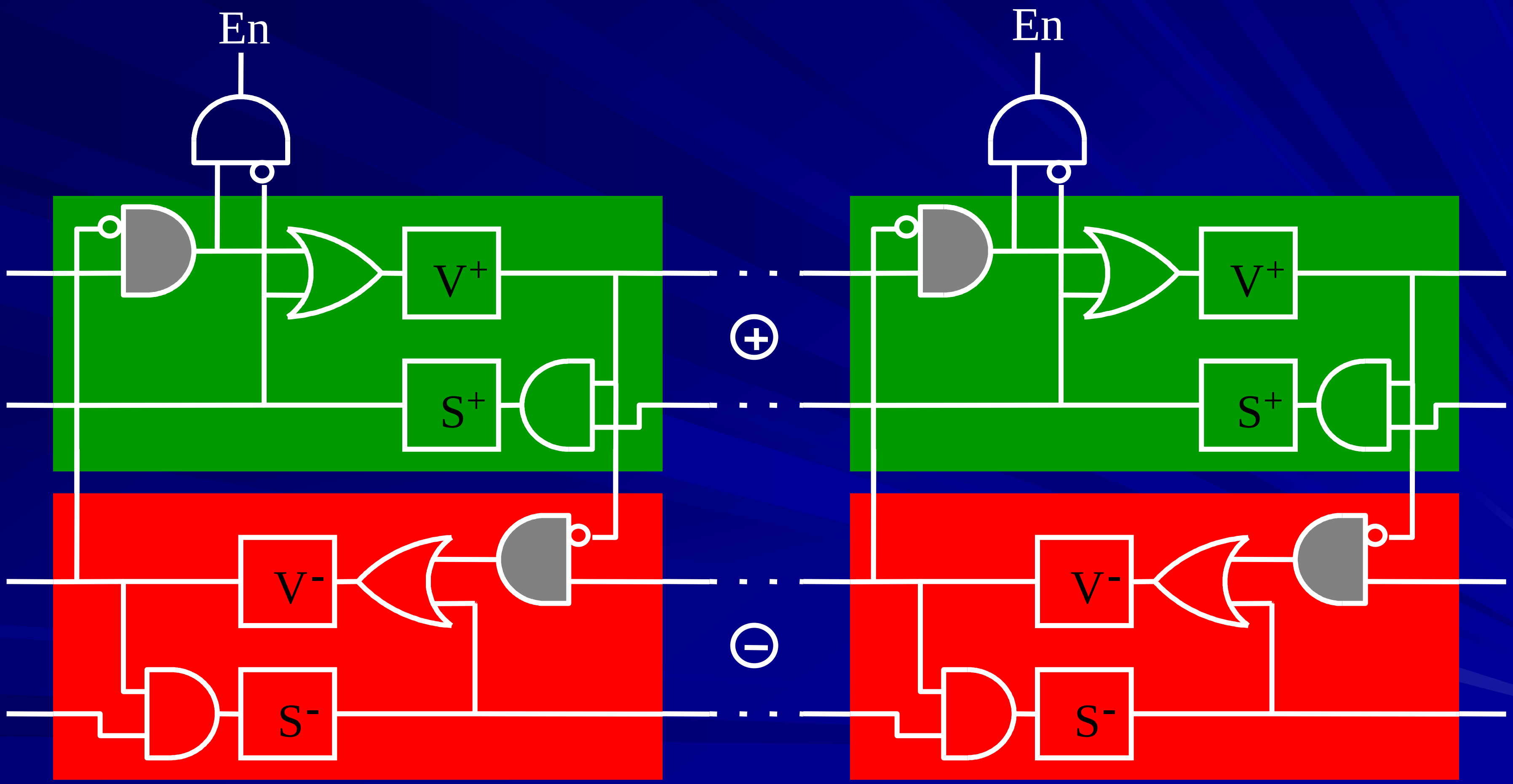
How to implement anti-tokens ?



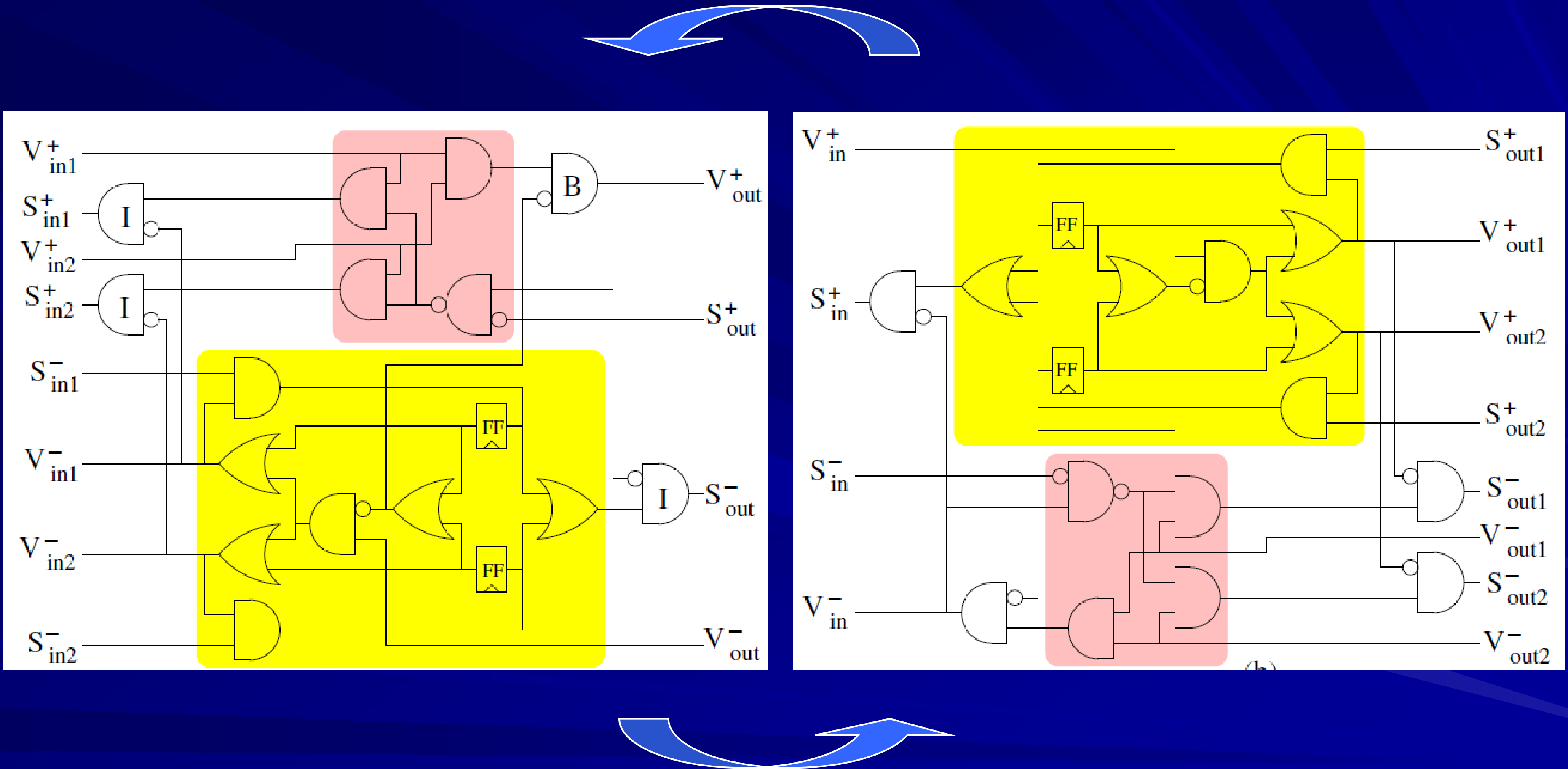
Controller for elastic buffer



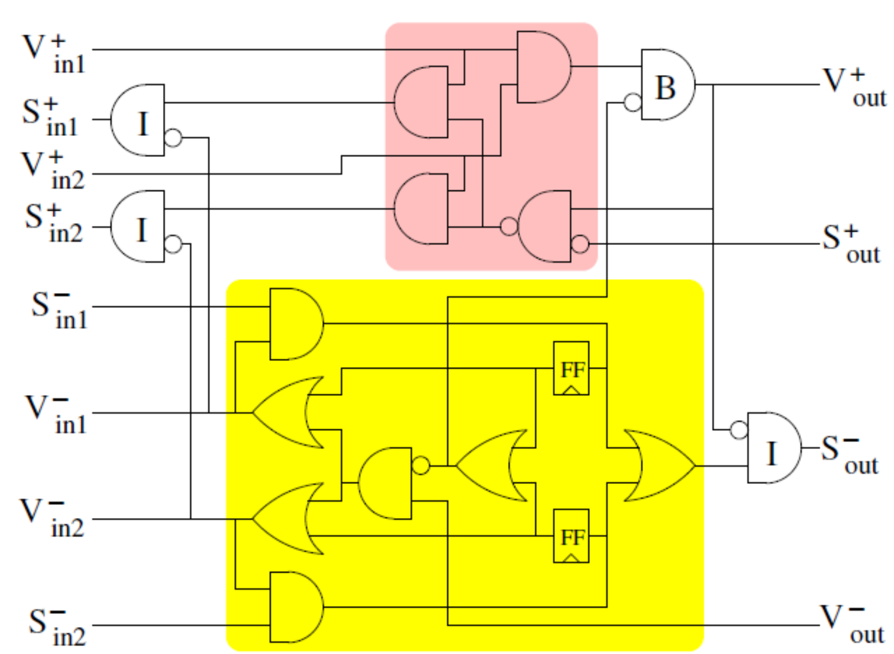
Dual controller for elastic buffer

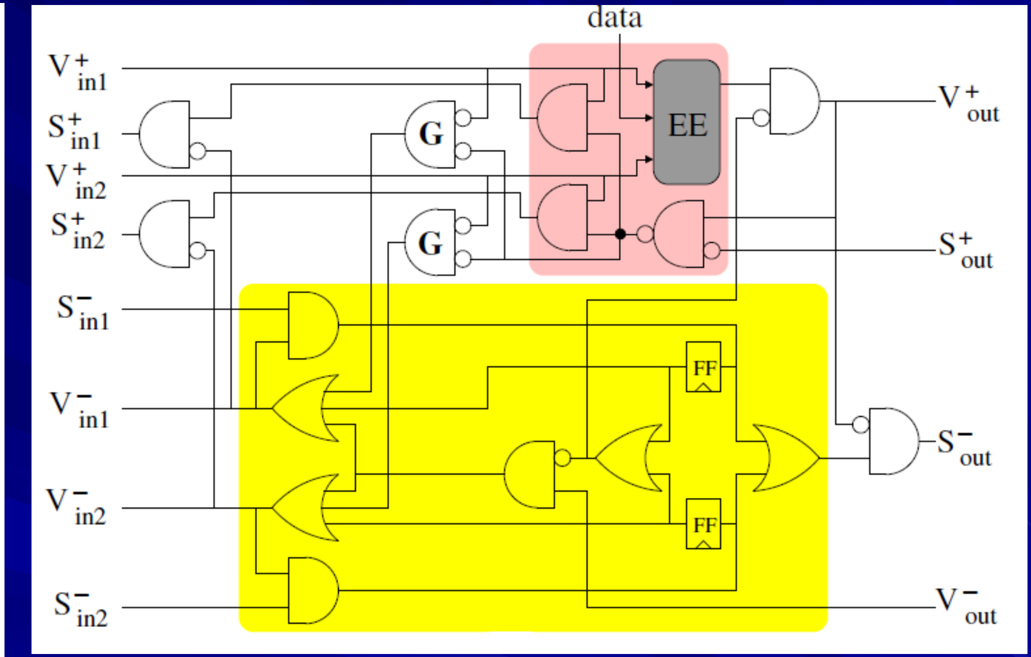


Dual Join and Fork



Join with early evaluation





Condition on Early Evaluation Function

Early evaluation function makes decision based on *presence* of valid bits, not on their *absence*

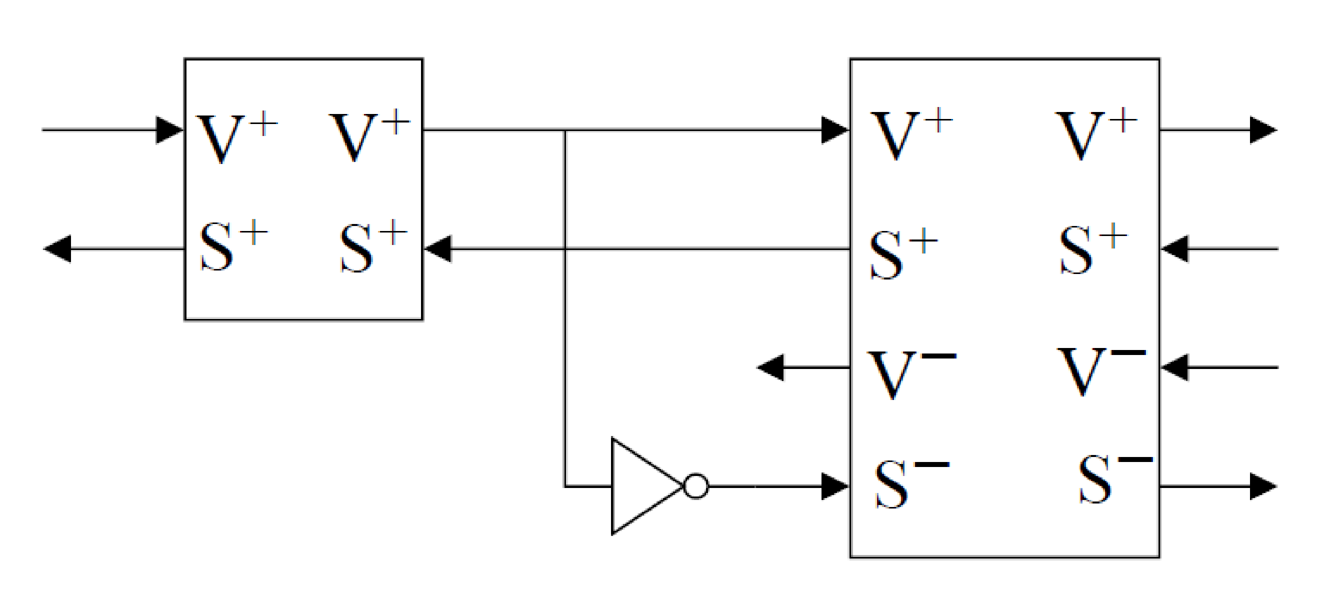
Formally: EE is positive unate with respect to data input

Example: legal EE function for a data-path MUX
(s – select input)

$$\mathrm{EE} = V_s^+ \wedge ((s \wedge V_a^+) \vee (\overline{s} \wedge V_b^+))$$

$$\mathrm{EE}_s = V_s^+ \wedge V_a^+ \text{ and } EE_{\overline{s}} = V_s^+ \wedge V_b^+$$

Passive anti-token (capacity one)



Bigger capacity can be achieved by “injecting” anti-token up-down counters on elastic channels

Properties of elastic channels

$$\texttt{AG}\ ((V^{+} \wedge S^{+}) \implies \texttt{AX}\ V^{+}) \qquad (\text{Retry}^{+})$$

$$\texttt{AG}\ ((V^{-} \wedge S^{-}) \implies \texttt{AX}\ V^{-}) \qquad (\text{Retry}^{-})$$

$$\texttt{AG}\ ((\overline{V+} \vee \overline{S-}) \wedge (\overline{V-} \vee \overline{S+})) \qquad (\text{Invariant (2)})$$

$$\texttt{AG AF}\ ((V^{+} \wedge \overline{S+}) \vee (V^{-} \wedge \overline{S-})) \qquad (\text{Liveness})$$

Invariants: mutually exclusive

Kill (V^{-}) and Stop (S^{+})

Valid (V^{+}) and retain of a kill (S^{-})

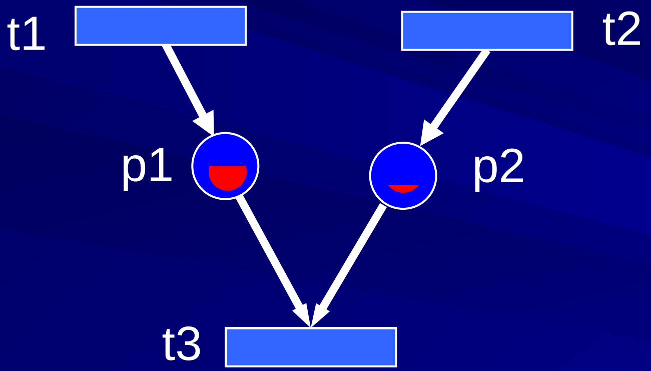
Conclusions

- Early evaluation can increase performance beyond the min cycle ratio
- The duality between positive and negative tokens suggests a clean and effective implementation
- Dual Marked Graphs is a formal model for analytical analysis and optimization methods

Performance analysis with early evaluation

Revisit Performance Analysis of Marked Graphs

The throughput can also be computed by means of linear programming



$$th = \min(\overline{m}_{p1}, \overline{m}_{p2})$$

Average marking

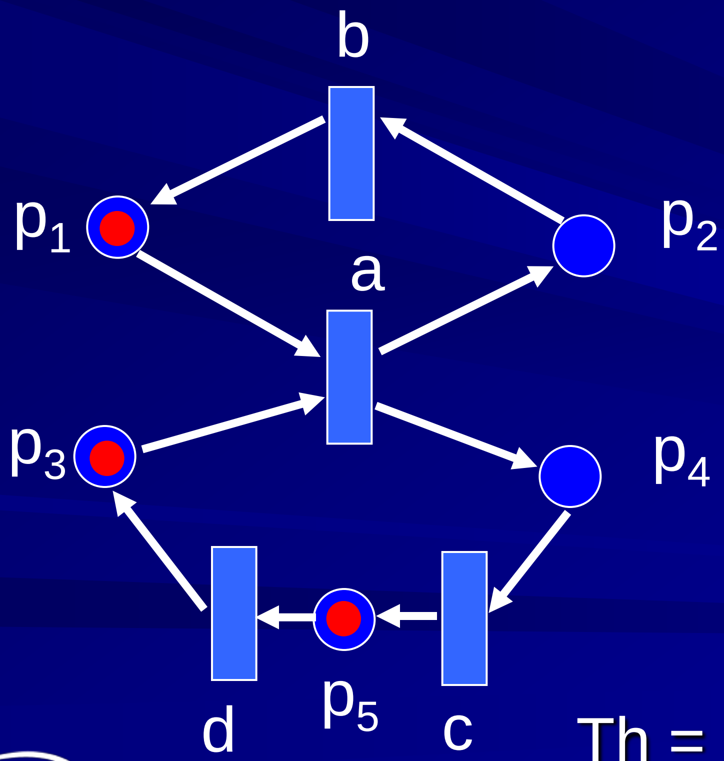
$$\overline{m}_p = \lim_{t\to\infty} \tfrac{1}{t}\int_0^t m_p(\tau)d\tau$$

Throughput

$$th = \min_p \overline{m}_p$$

[Campos, Chiola, Silva 1991]

Revisit Performance Analysis of Marked Graphs



Th = 0.5

max th

reachability

$$\overline{m}_{p1} = 1 + t_b - t_a$$
$$\overline{m}_{p2} = 0 + t_a - t_b$$
$$\overline{m}_{p3} = 1 + t_d - t_a$$
$$\overline{m}_{p4} = 0 + t_a - t_c$$
$$\overline{m}_{p5} = 1 + t_c - t_d$$

th constraints

$th \leq \overline{m}_{p2}$ // transition b

$th \leq \overline{m}_{p4}$ // transition c

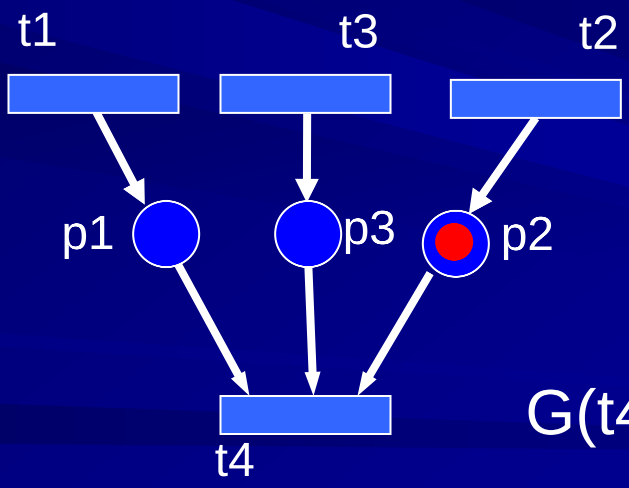
$th \leq \overline{m}_{p5}$ // transition d

$th \leq \min(\overline{m}_{p1}, \overline{m}_{p3})$ // transition a

GMG = Multi-guarded Dual Marked Graph

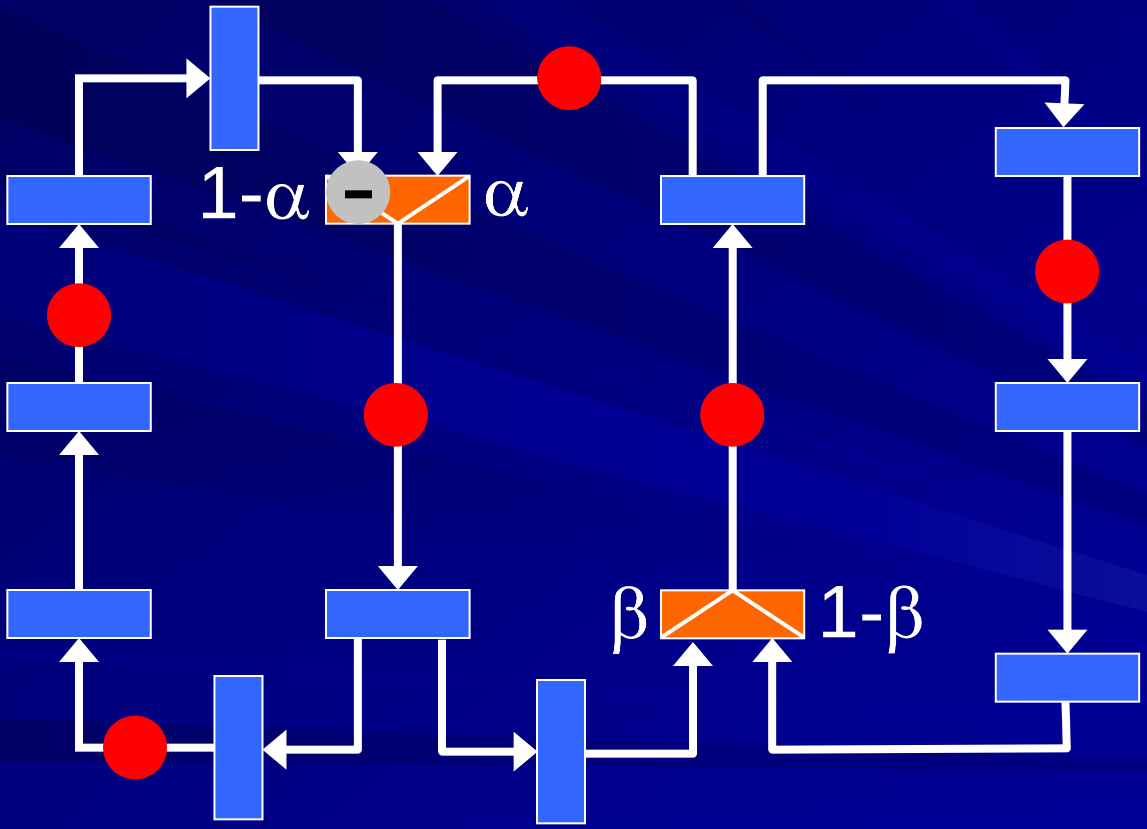
- Refinement of passive DMGs
- Every node has a set of guards
- Every guard is a set of input places (arcs)

Example:

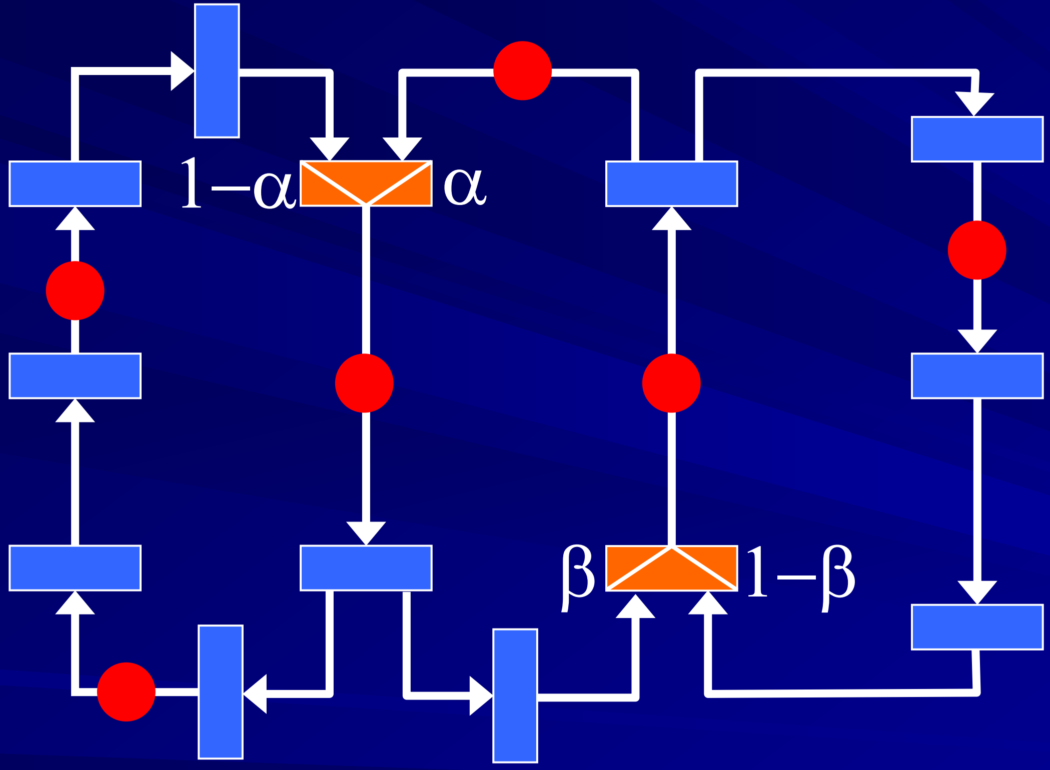


G(t4)={{p1,p3},{p2,p3}}

Early evaluation



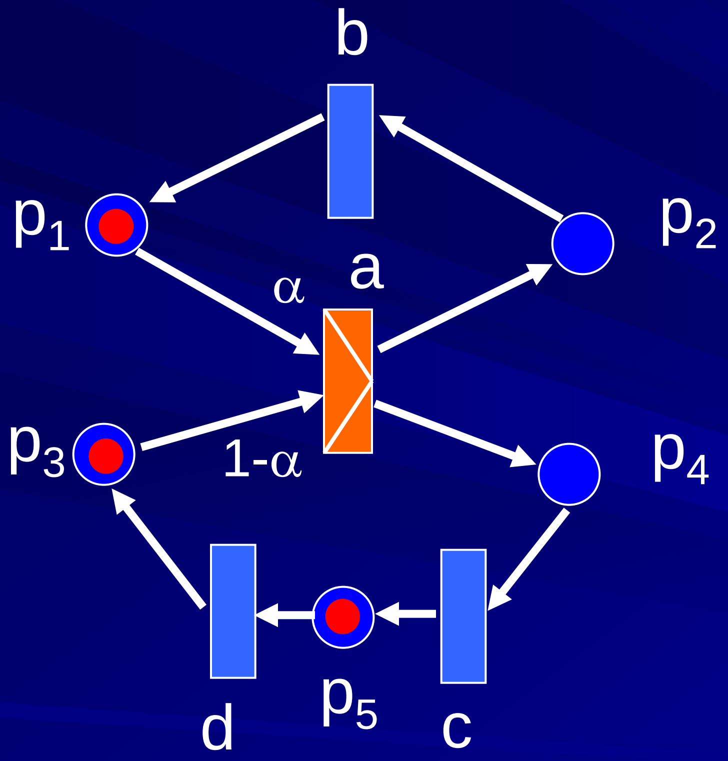
Early evaluation



β \ α	0.0	0.2	0.4	0.6	0.8	1.0
0.0	0.40	0.40	0.40	0.40	0.40	0.40
0.2	0.42	0.42	0.42	0.42	0.43	0.43
0.4	0.43	0.44	0.44	0.45	0.45	0.45
0.6	0.43	0.44	0.45	0.47	0.48	0.49
0.8	0.43	0.44	0.46	0.48	0.51	0.54
1.0	0.43	0.44	0.46	0.49	0.54	0.60

(0.43) (0.60) (0.40)

LP formulation for an upper bound of a throughput (by example)



max th

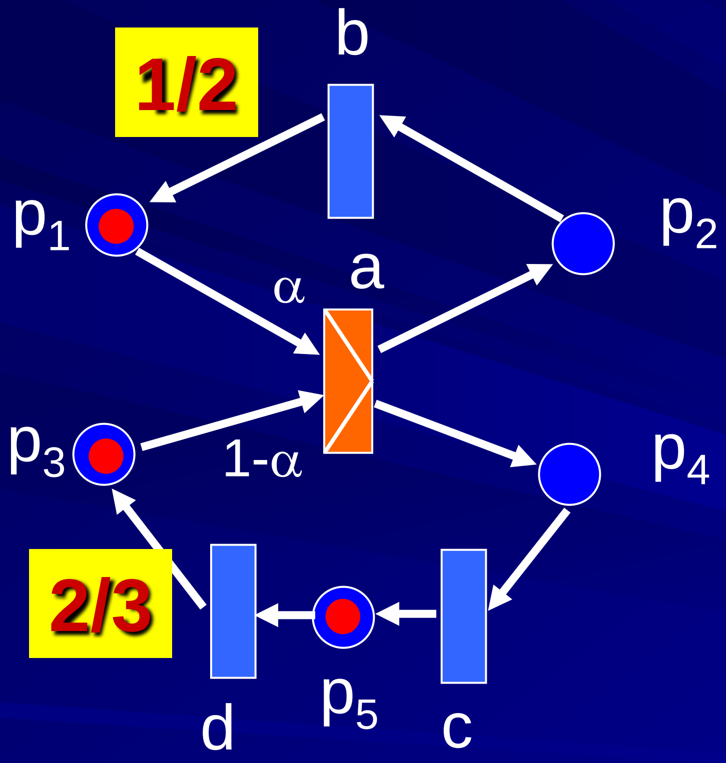
$$\overline{m}_{p1} = 1 + t_b - t_a$$
$$\overline{m}_{p2} = 0 + t_a - t_b$$
$$\overline{m}_{p3} = 1 + t_d - t_a$$
$$\overline{m}_{p4} = 0 + t_a - t_c$$
$$\overline{m}_{p5} = 1 + t_c - t_d$$

$$th \leq \overline{m}_{p2}$$
$$th \leq \overline{m}_{p4}$$
$$th \leq \overline{m}_{p5}$$

$$th = \alpha\, \overline{m}_{p1} + (1-\alpha)\, \overline{m}_{p3}$$

$$Th = (2 - \alpha) / (3 - \alpha)$$

Averaging cycle throughput or cycle times does not work



$$Th = (2 - \alpha) / (3 - \alpha)$$

Averaging throughput of individual cycles

$$Th' = \alpha\ 1/2 + (1- \alpha)\ 2/3 = (4 - \alpha) / 6$$

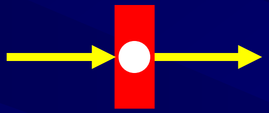
Averaging effective cycle times of individual cycles

$$1/Th'' = 2\alpha + (1- \alpha)\ 3/2 = (3 + \alpha) / 2$$

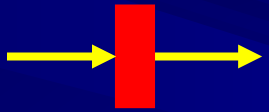
$$Th'' = 2/(3+ \alpha)$$

Correct-by-construction pipelining

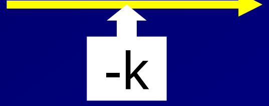
Notation for elastic systems



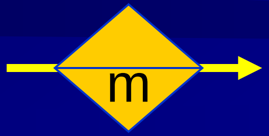
Elastic buffer (latency=1, capacity=2)
with one token of information



Empty elastic buffer (latency=1, capacity=2)

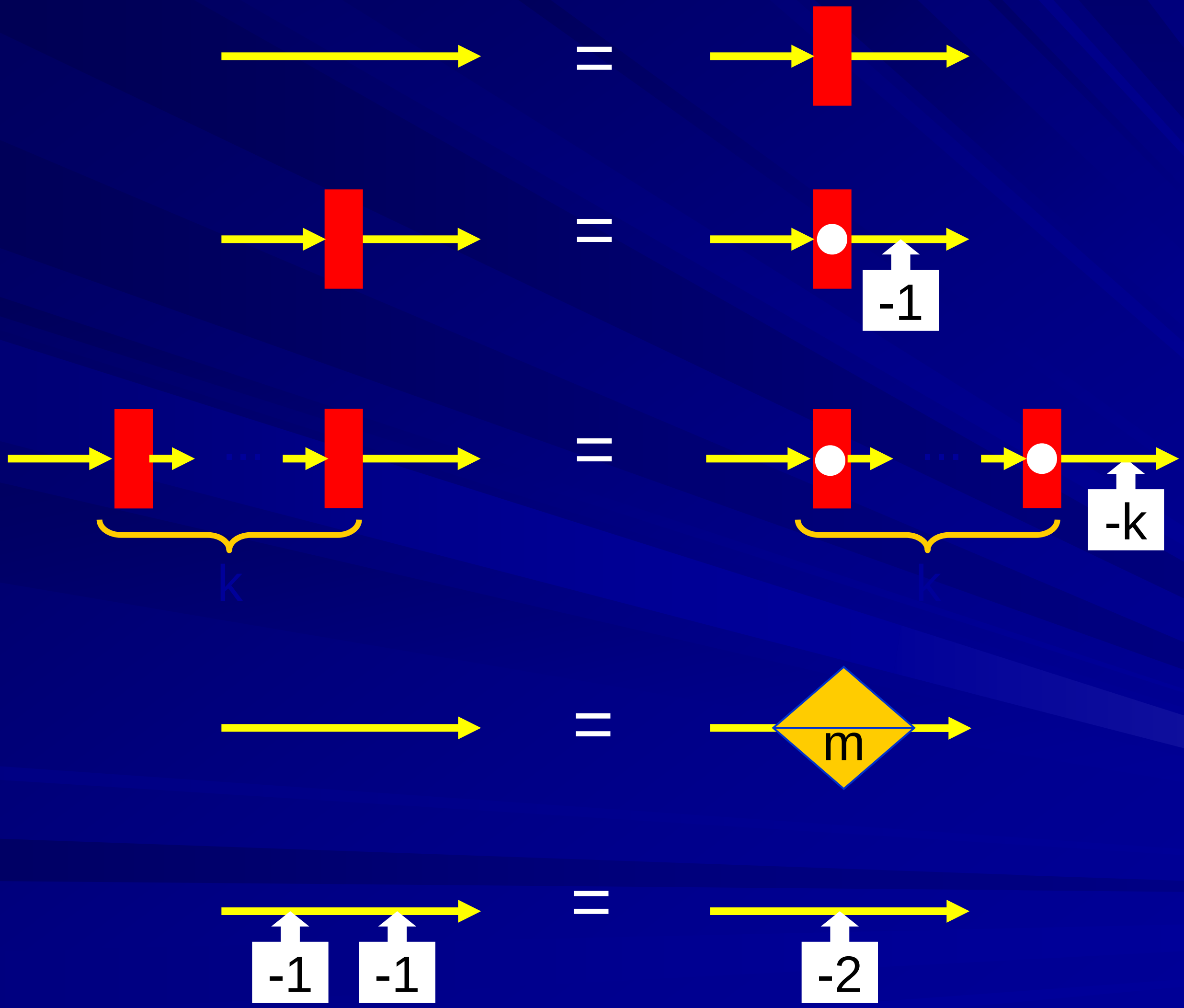


Channel with an injector of k negative tokens

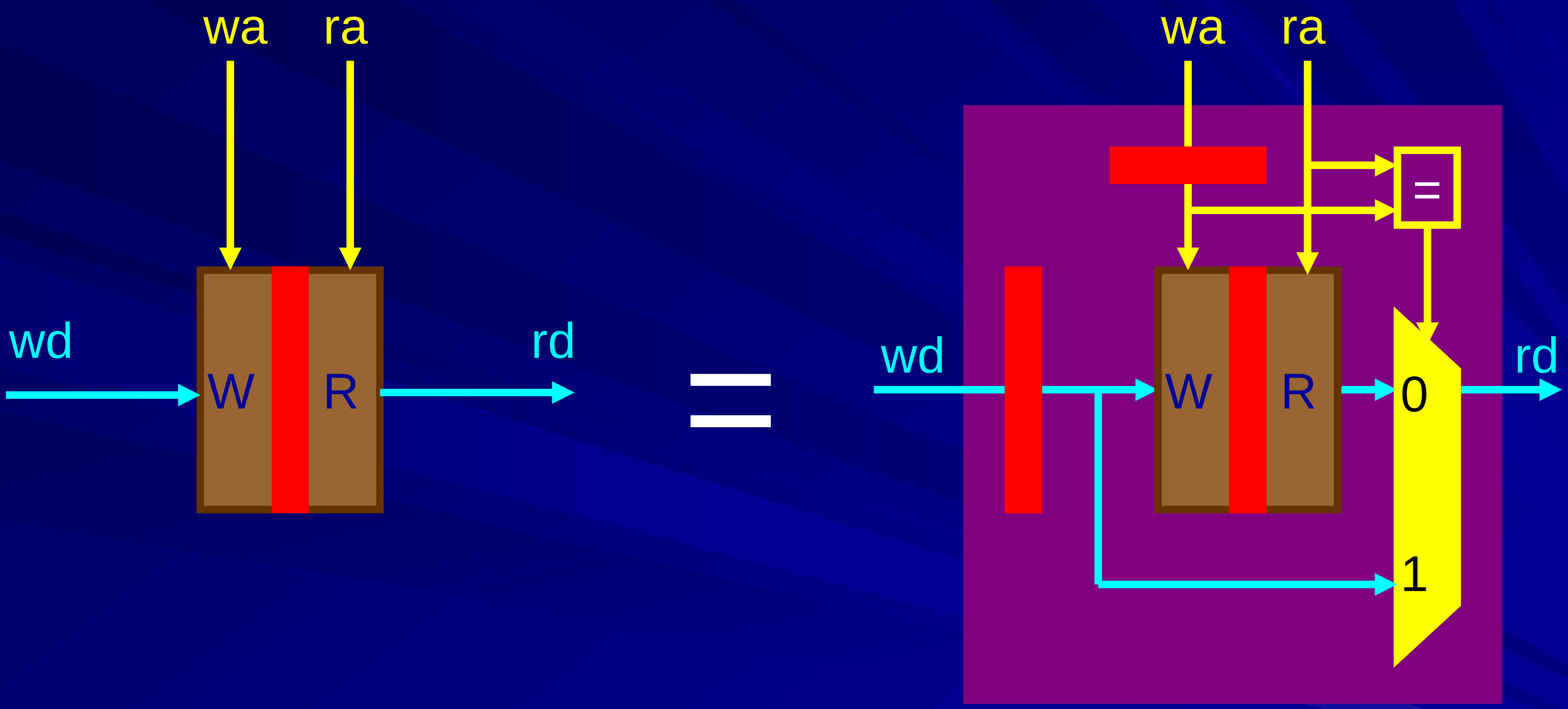


Empty elastic buffer (latency=0, capacity=m)

Elastic transforms



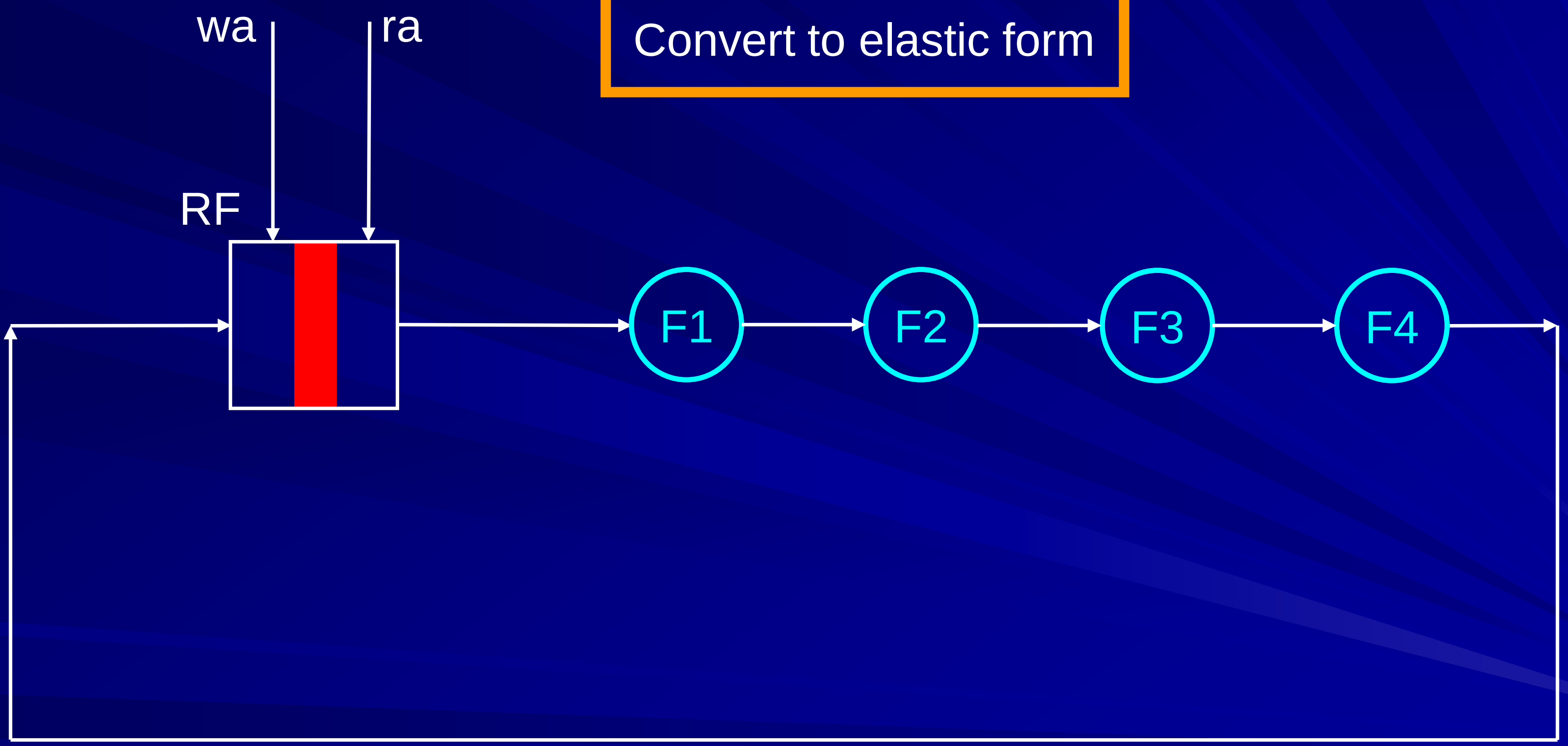
Bypass transform



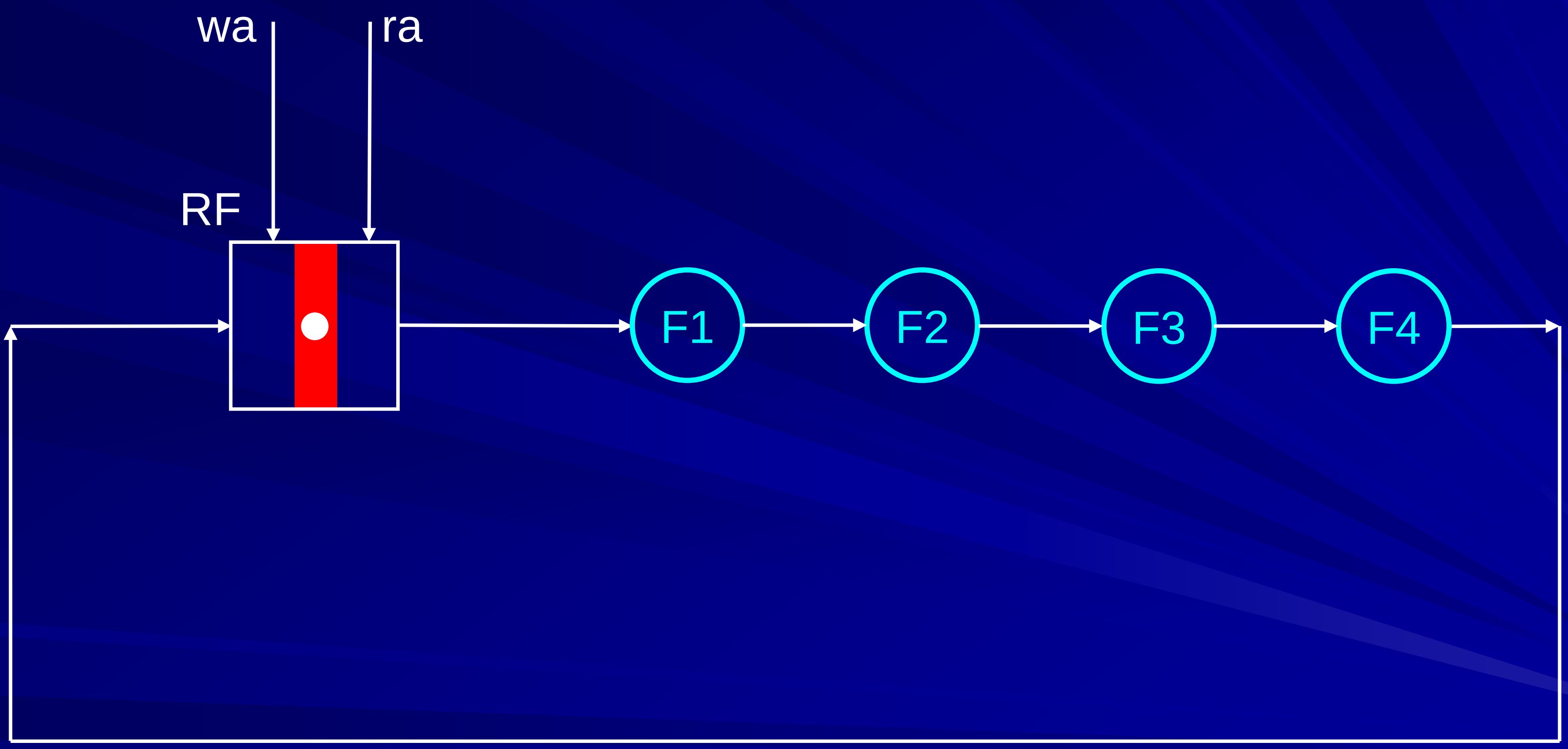
Classic transform. Works for elastic systems

Pipelining by example

Convert to elastic form



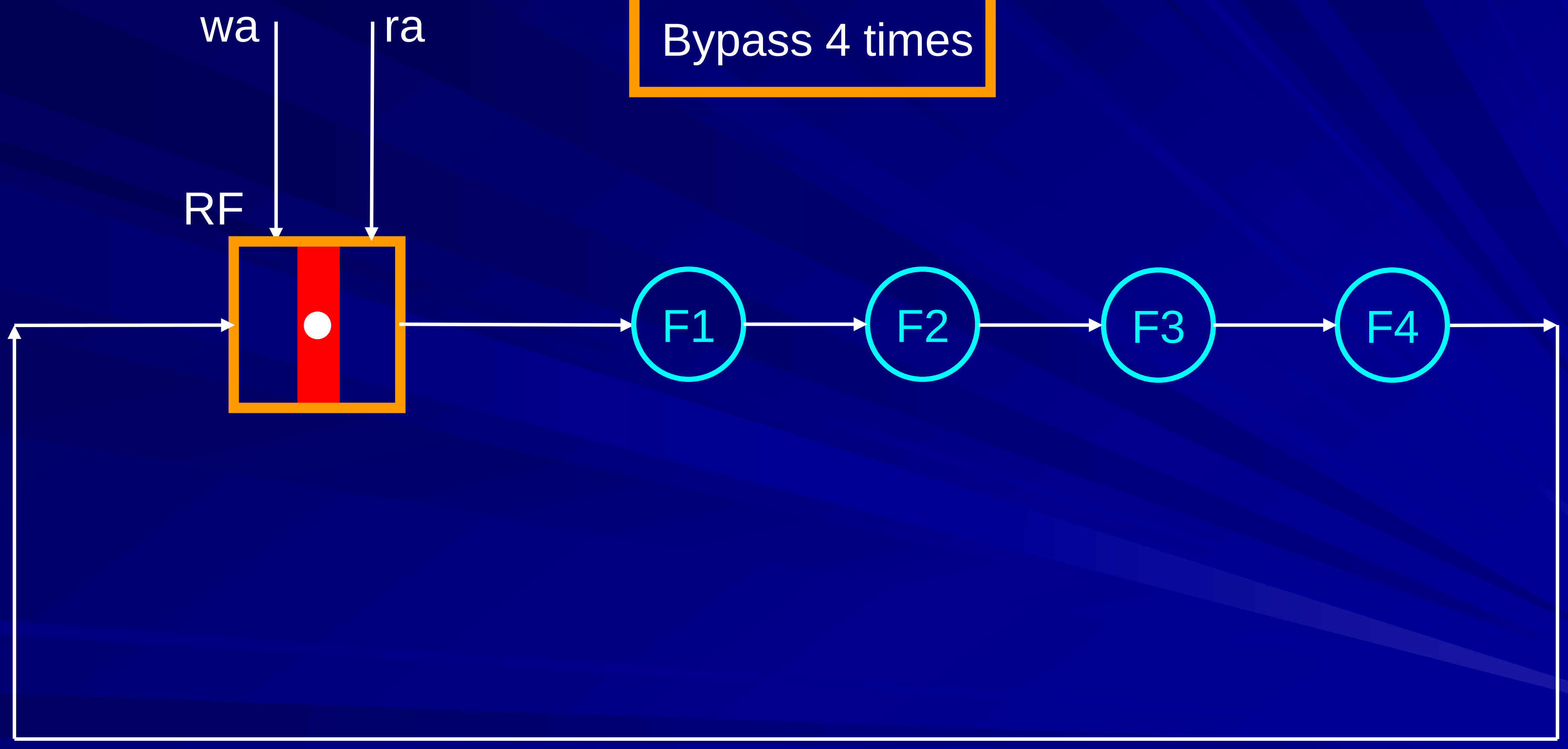
Pipelining by example



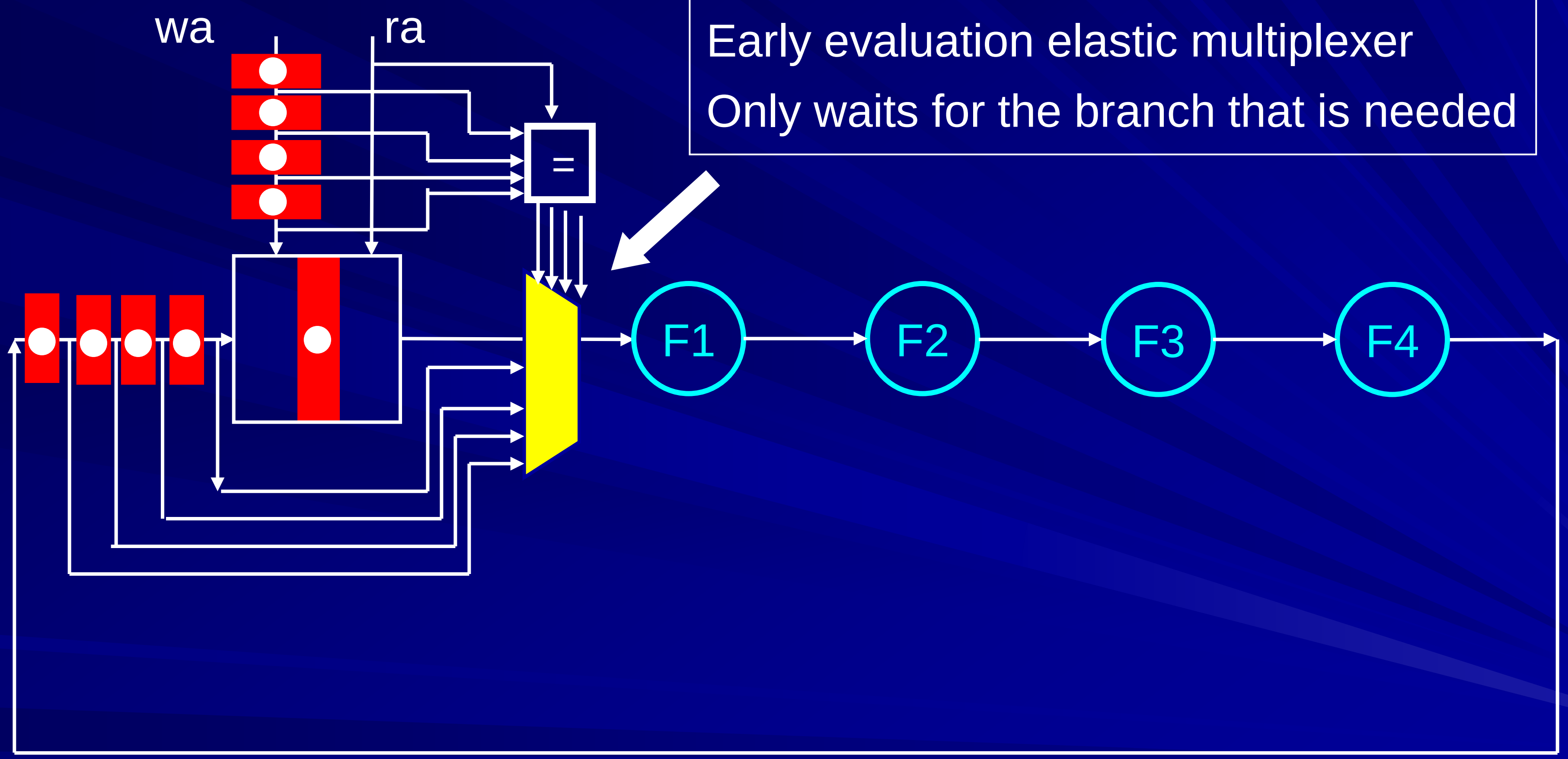
Handshakes added to the environment

Pipelining by example

Bypass 4 times



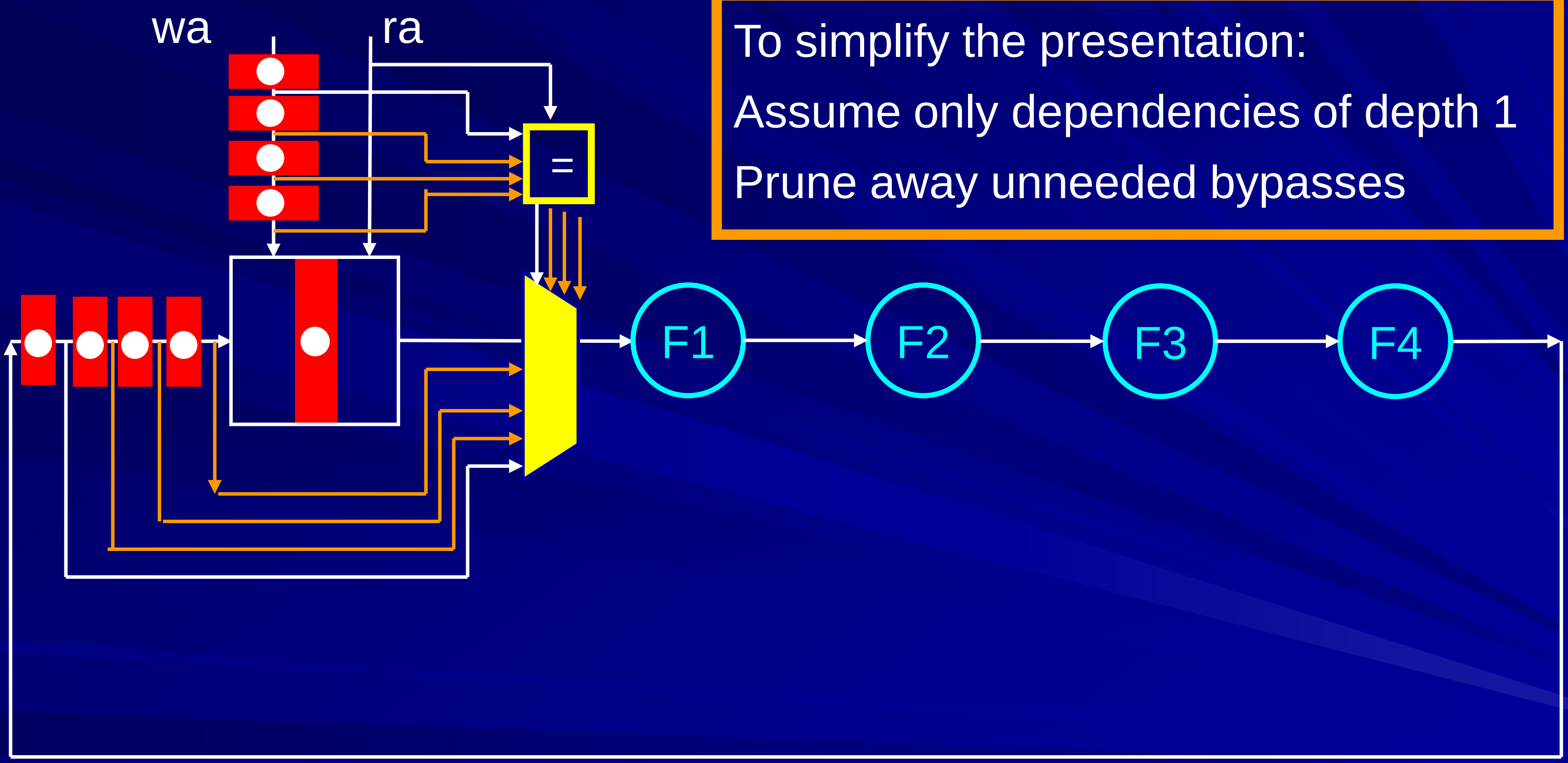
Pipelining by example



Early evaluation elastic multiplexer

Only waits for the branch that is needed

Pipelining by example

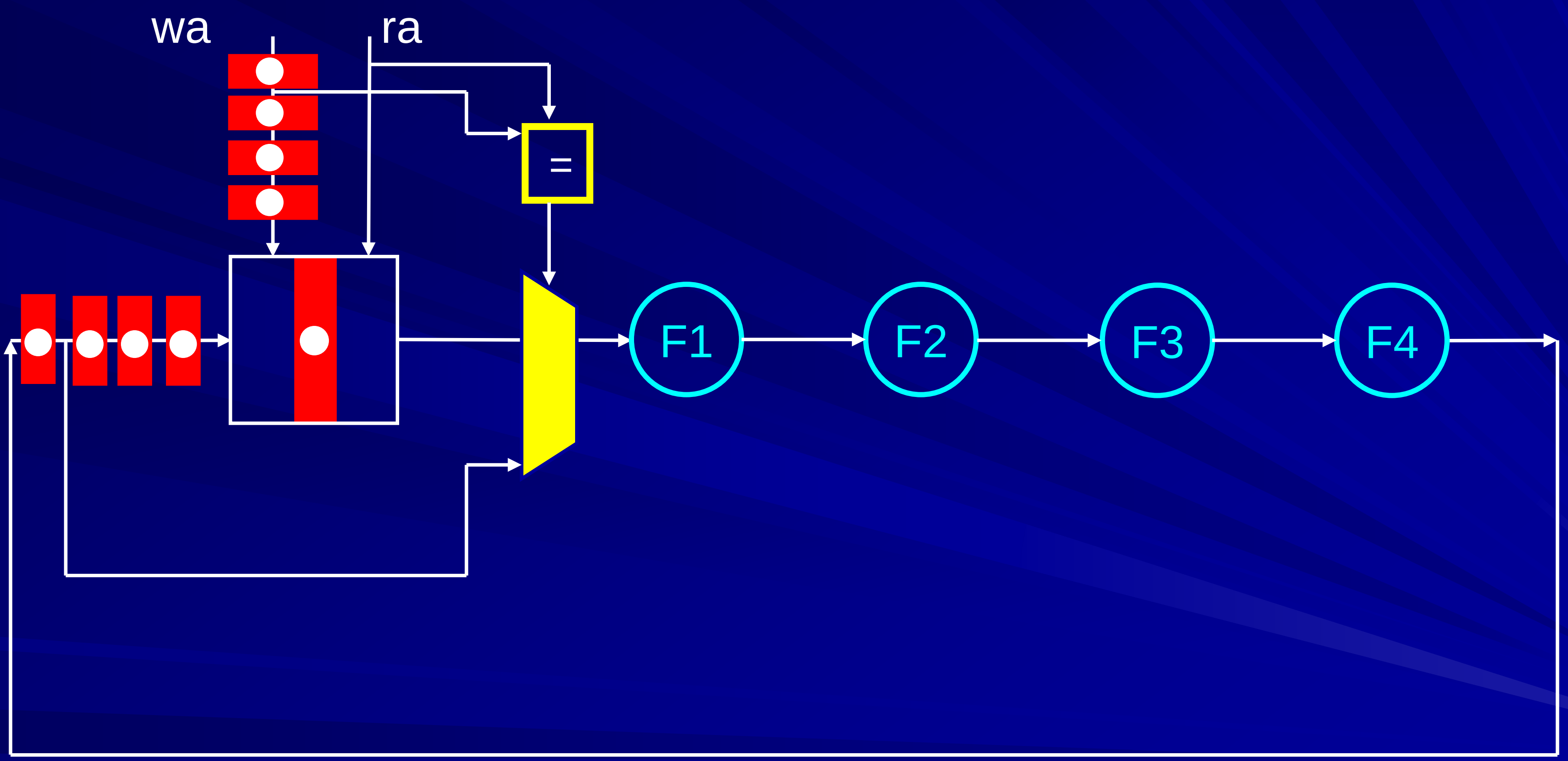


To simplify the presentation:

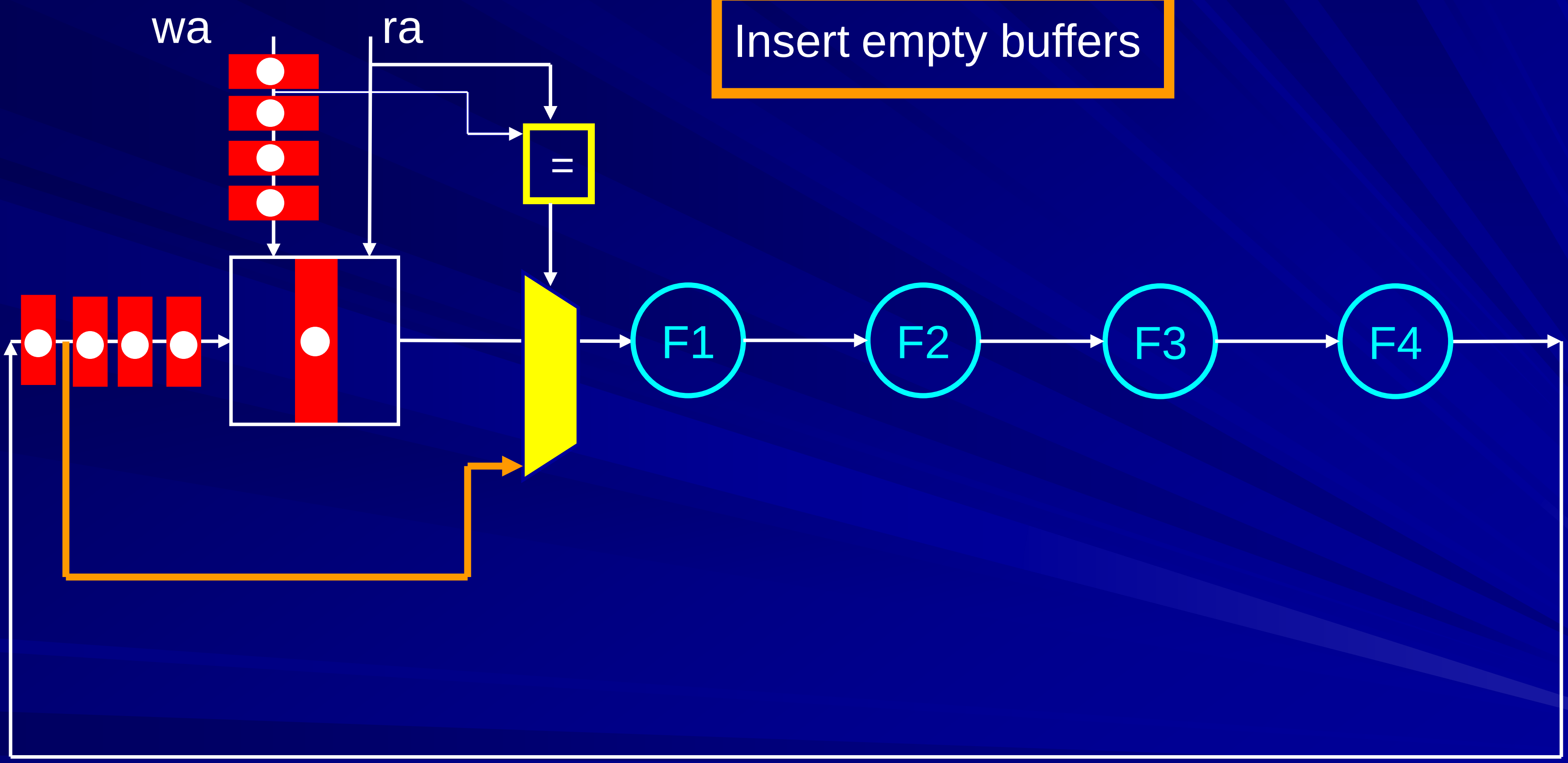
Assume only dependencies of depth 1

Prune away unneeded bypasses

Pipelining by example

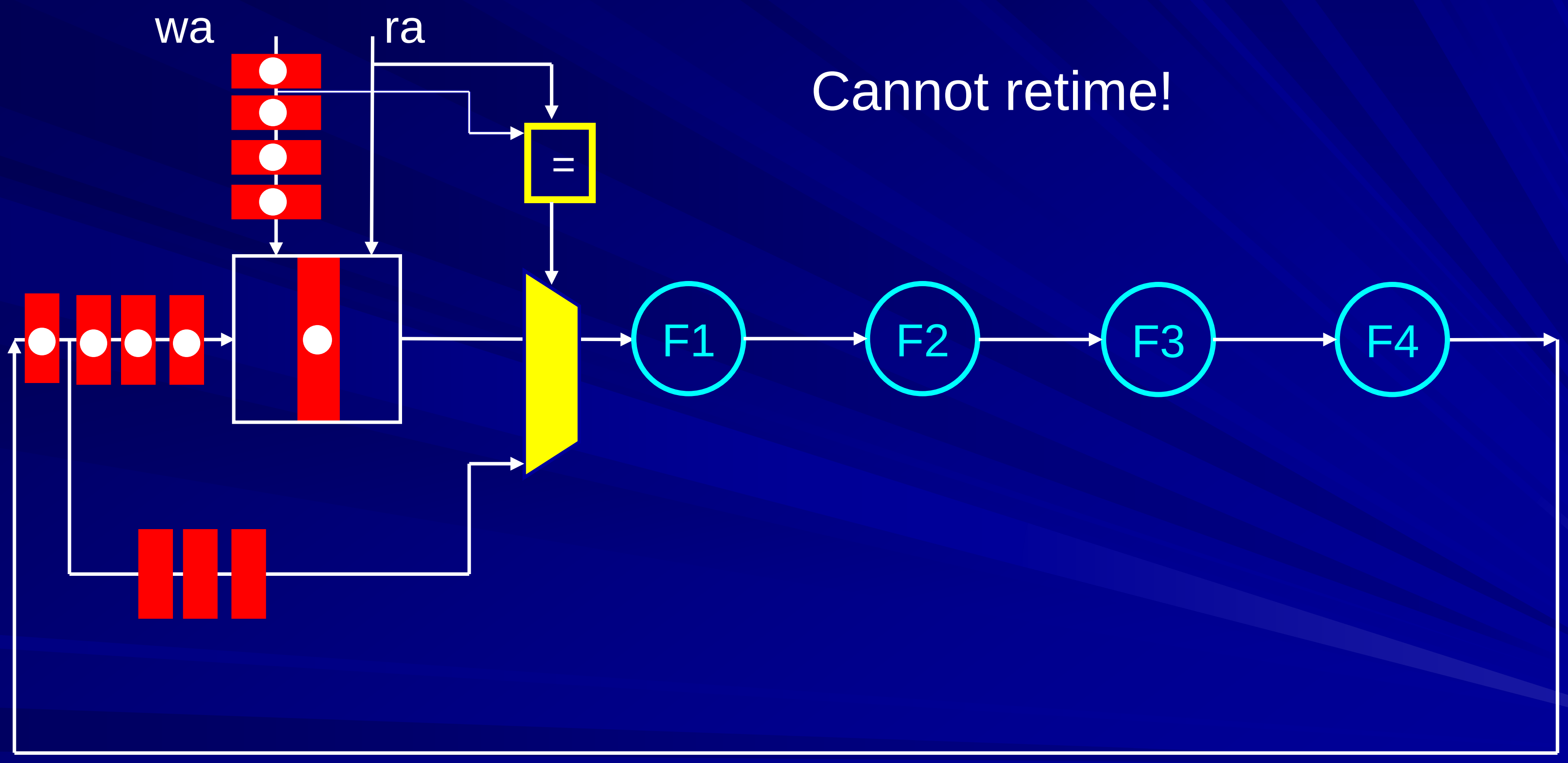


Pipelining by example



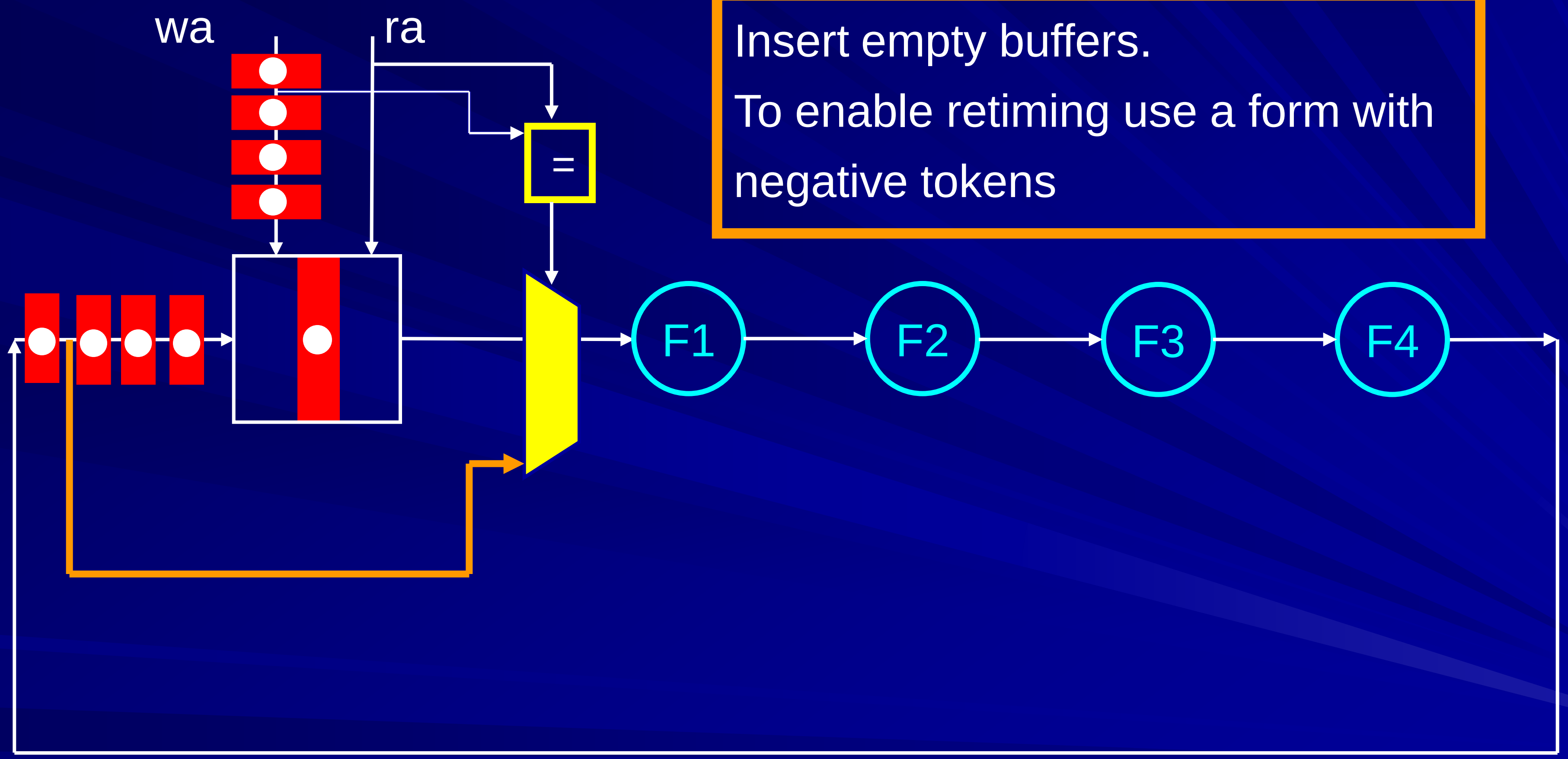
Insert empty buffers

Pipelining by example



Cannot retime!

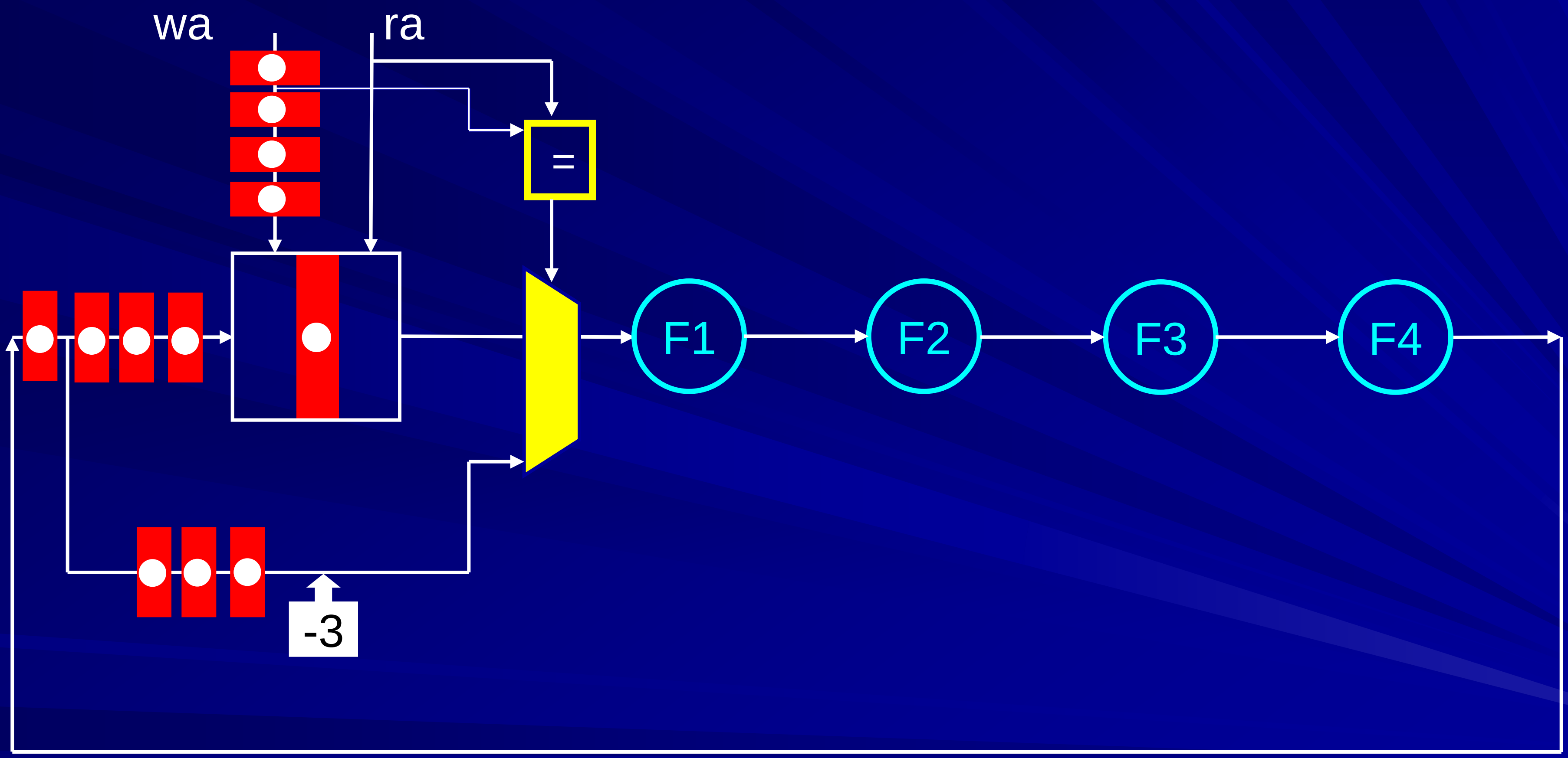
Pipelining by example



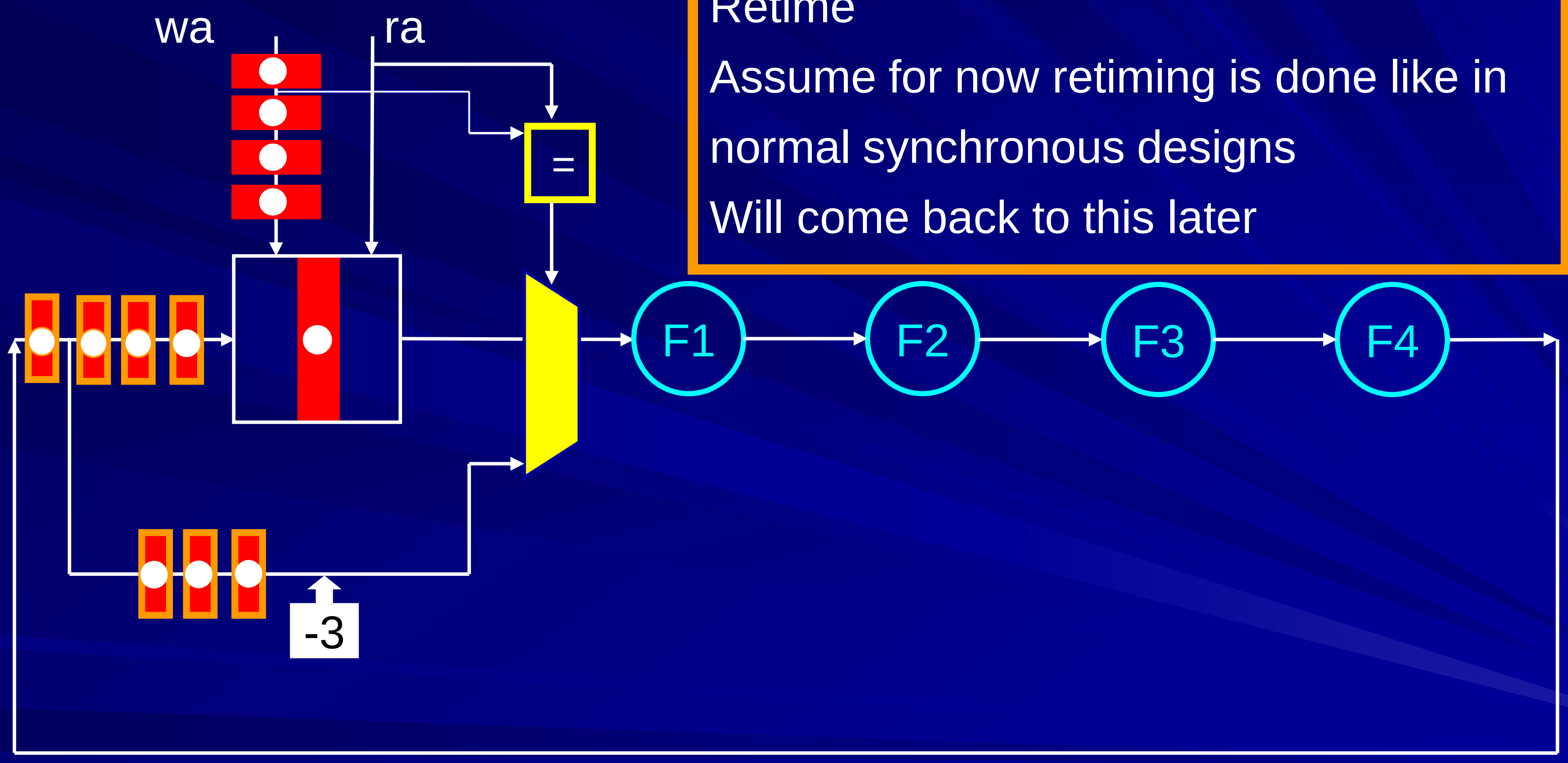
Insert empty buffers.

To enable retiming use a form with negative tokens

Pipelining by example



Pipelining by example

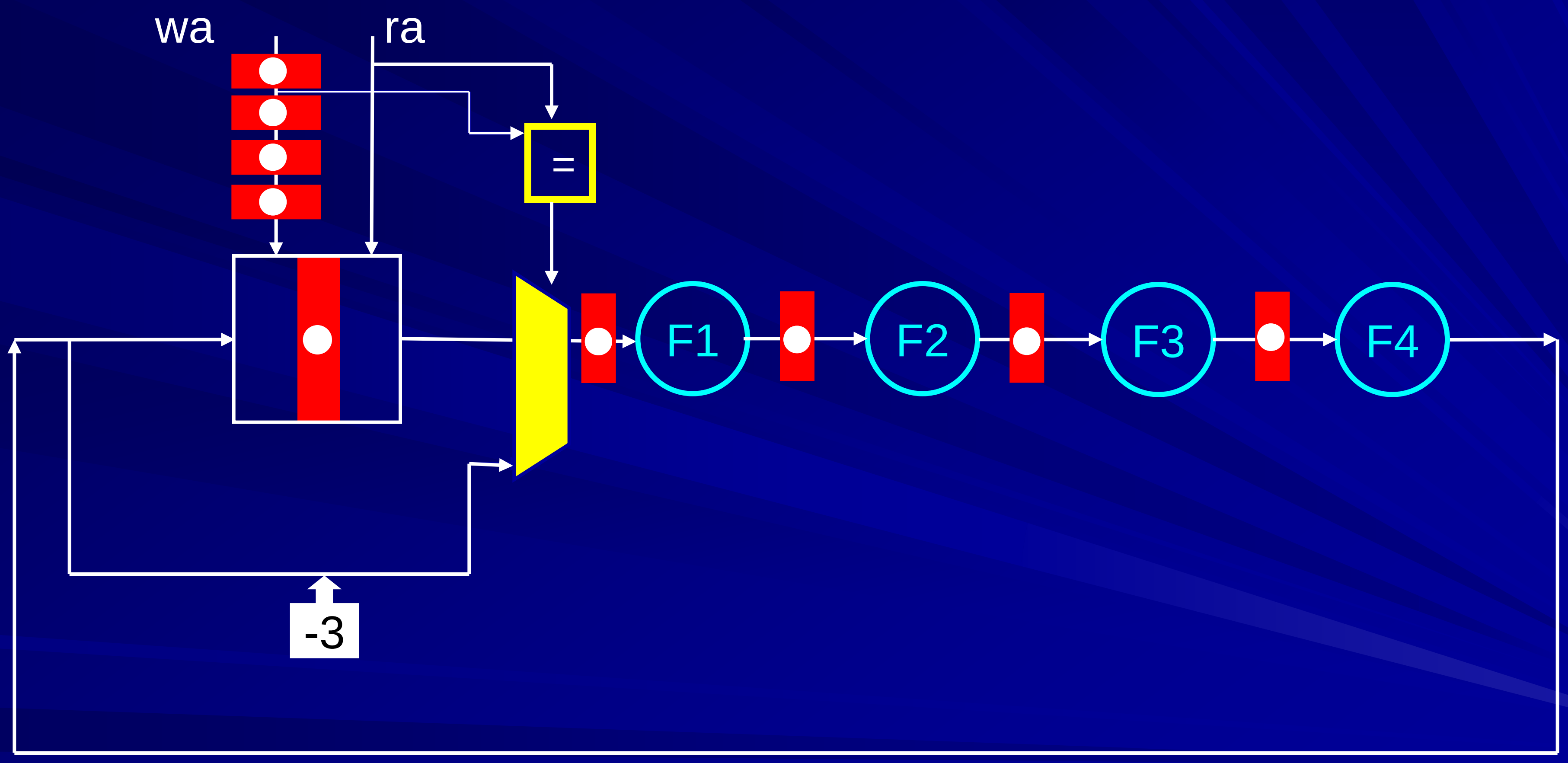


Retime

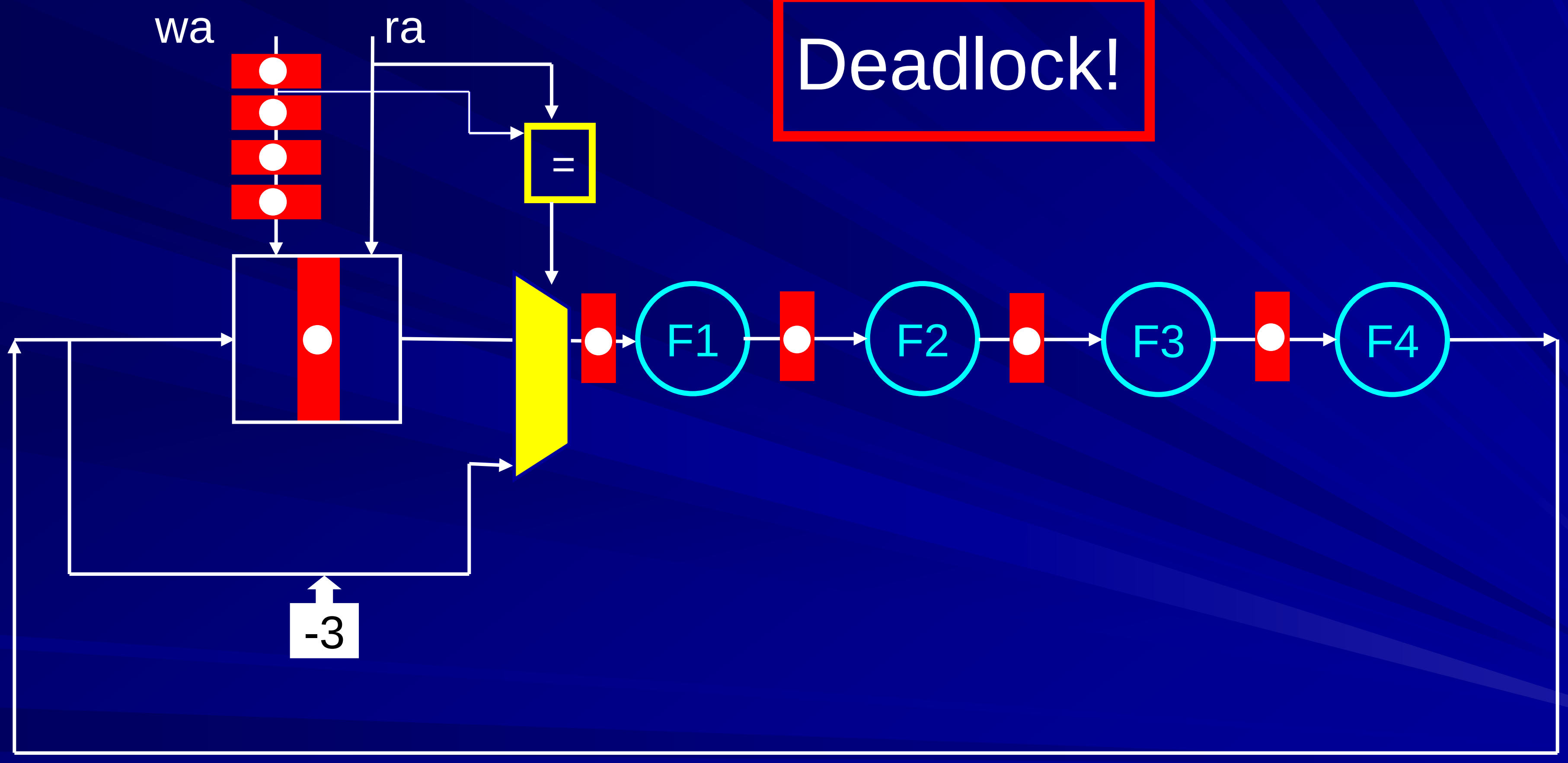
Assume for now retiming is done like in normal synchronous designs

Will come back to this later

Pipelining by example

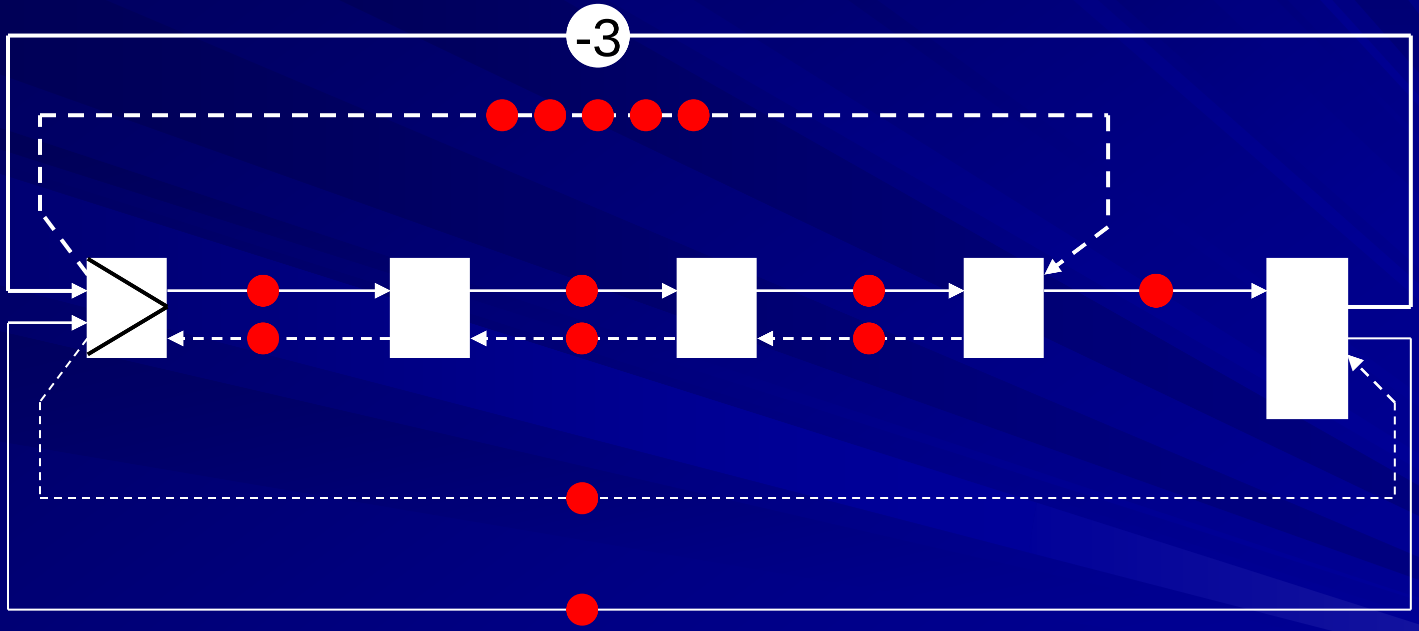


Pipelining by example



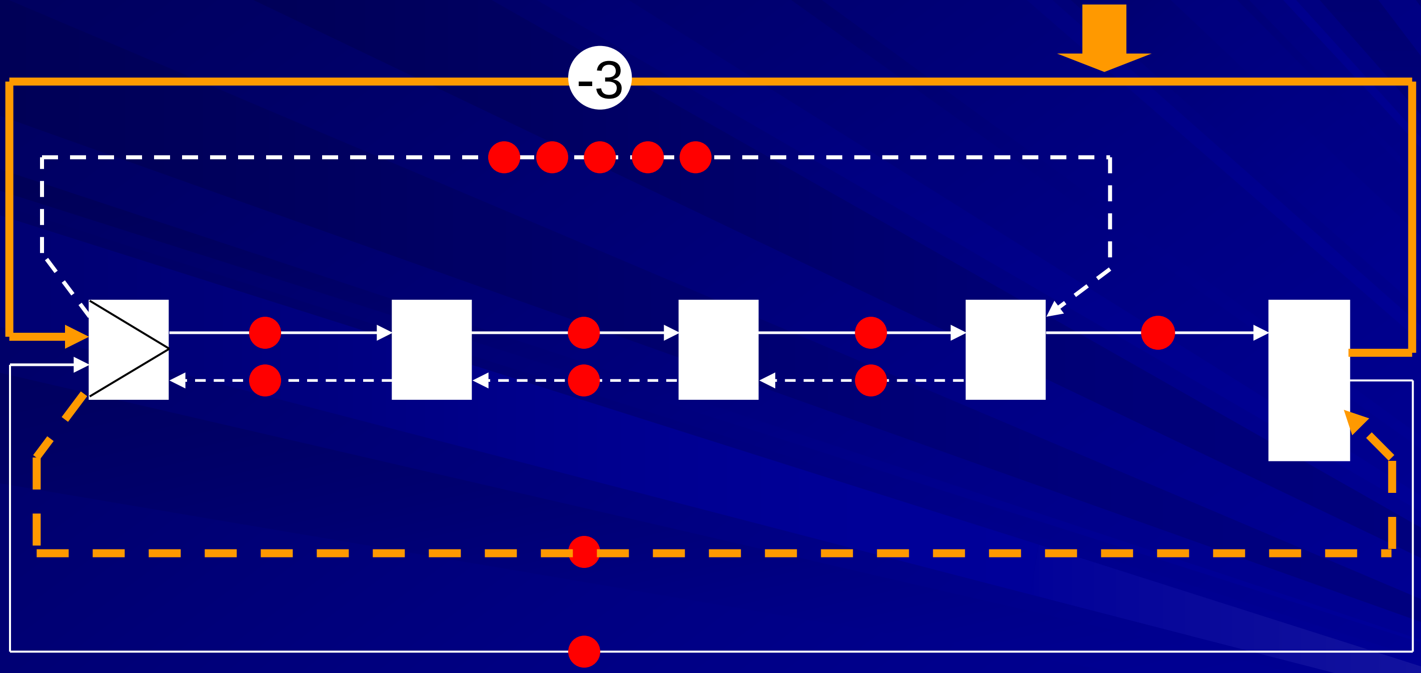
Deadlock!

Why deadlock?



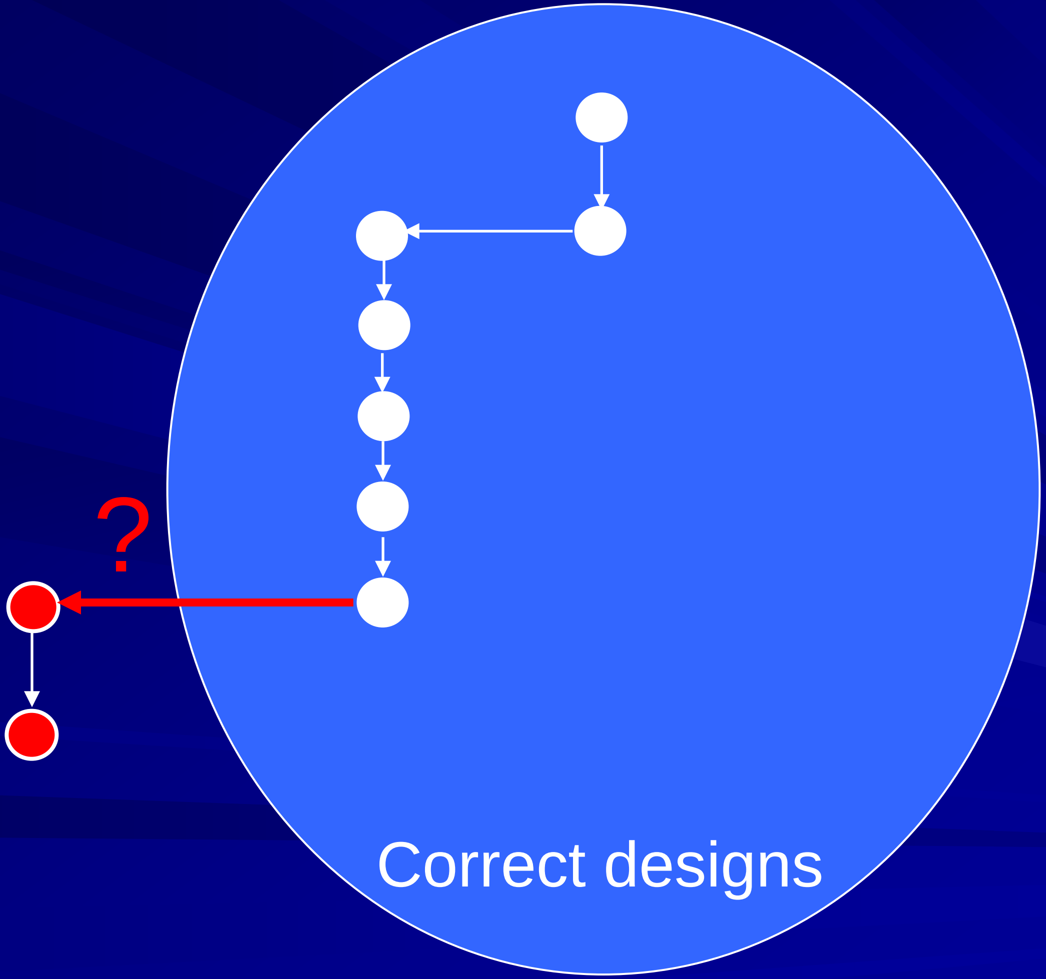
Why deadlock?

Cycle with sum of tokens = -2

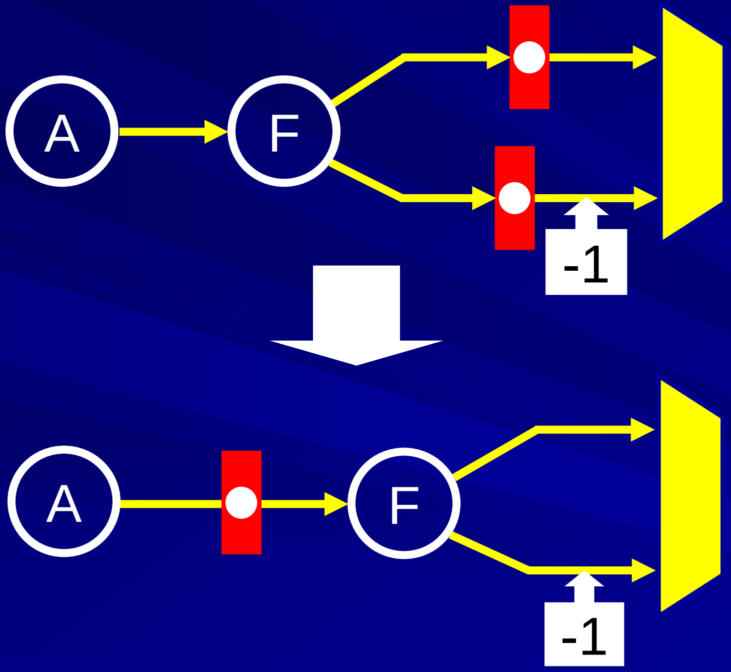


Positively live system ⇔ all cycles have positive marking

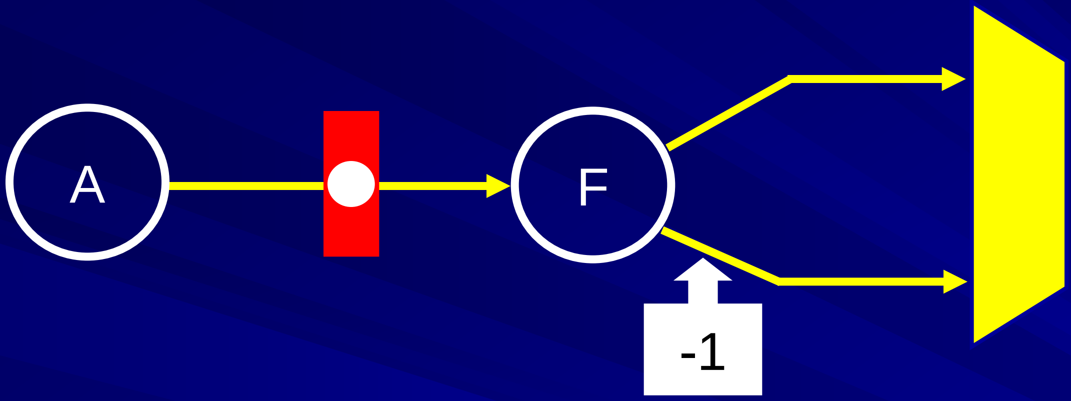
Transformations



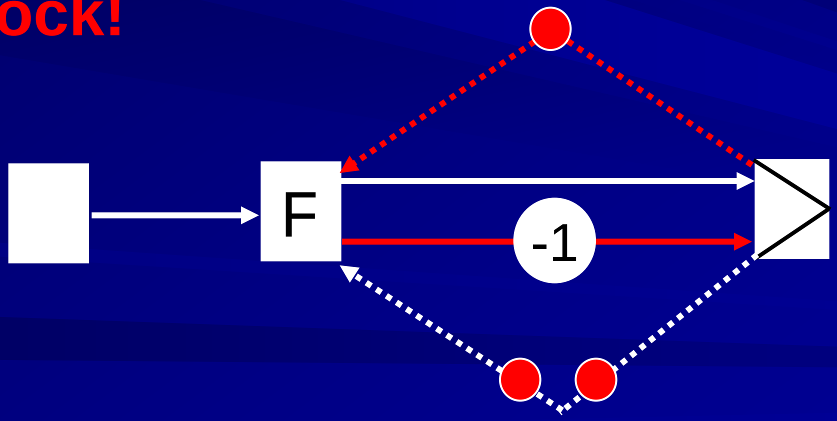
Retiming of Elastic Buffers



Retiming of Elastic Buffers

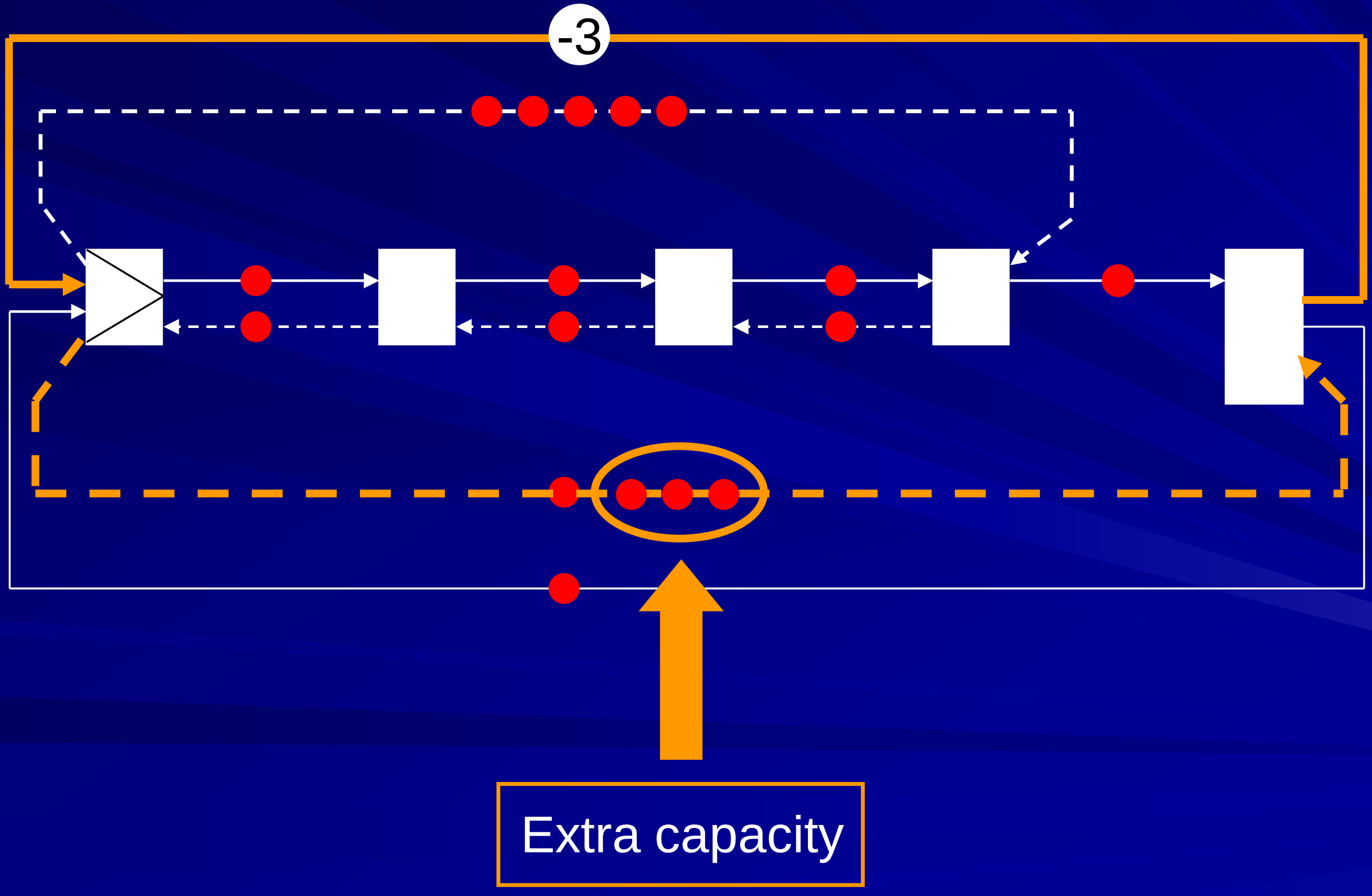


Deadlock!

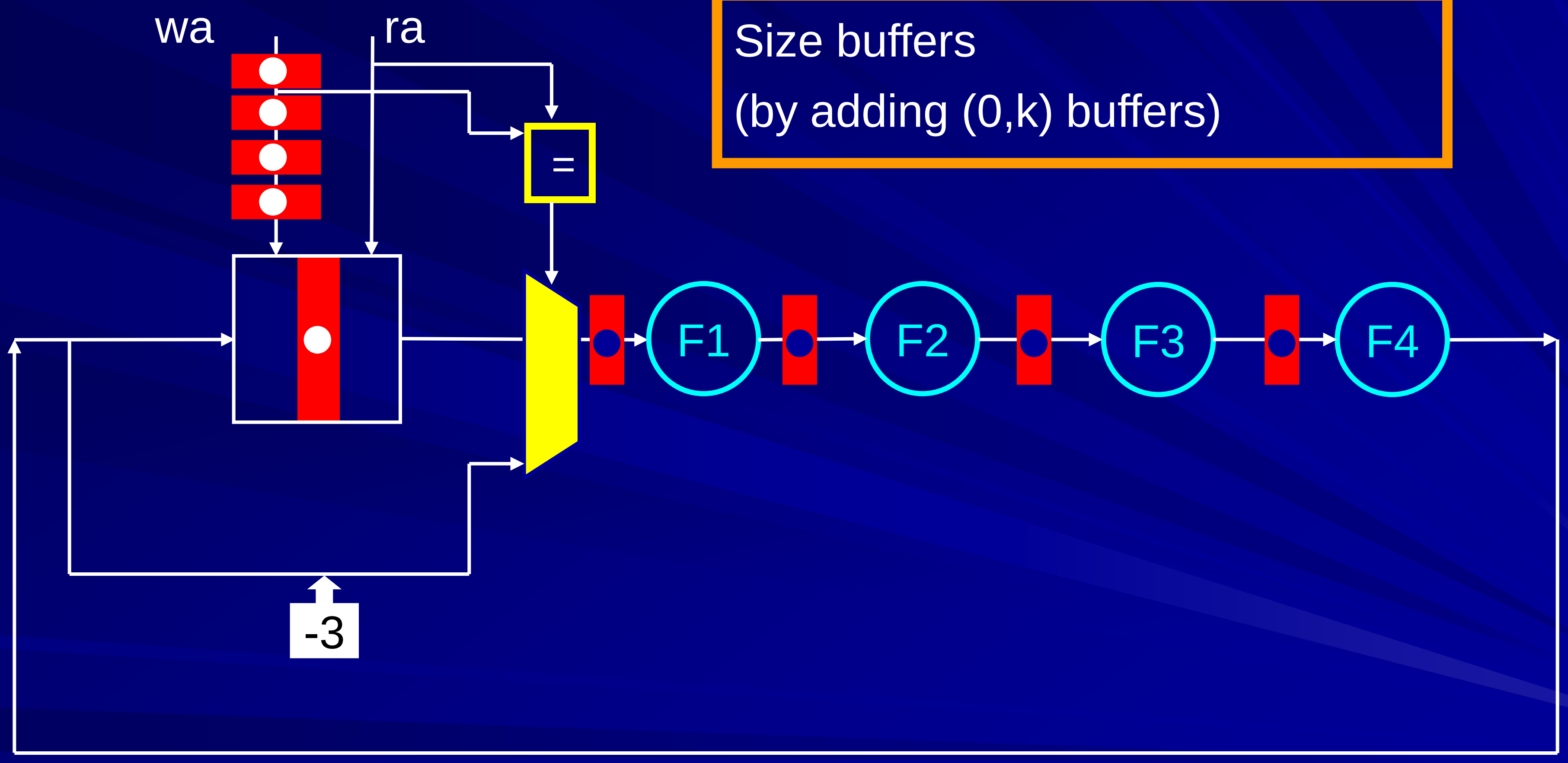


Retiming move removed required buffer capacity from the old location

How to fix deadlock



Pipelining by example

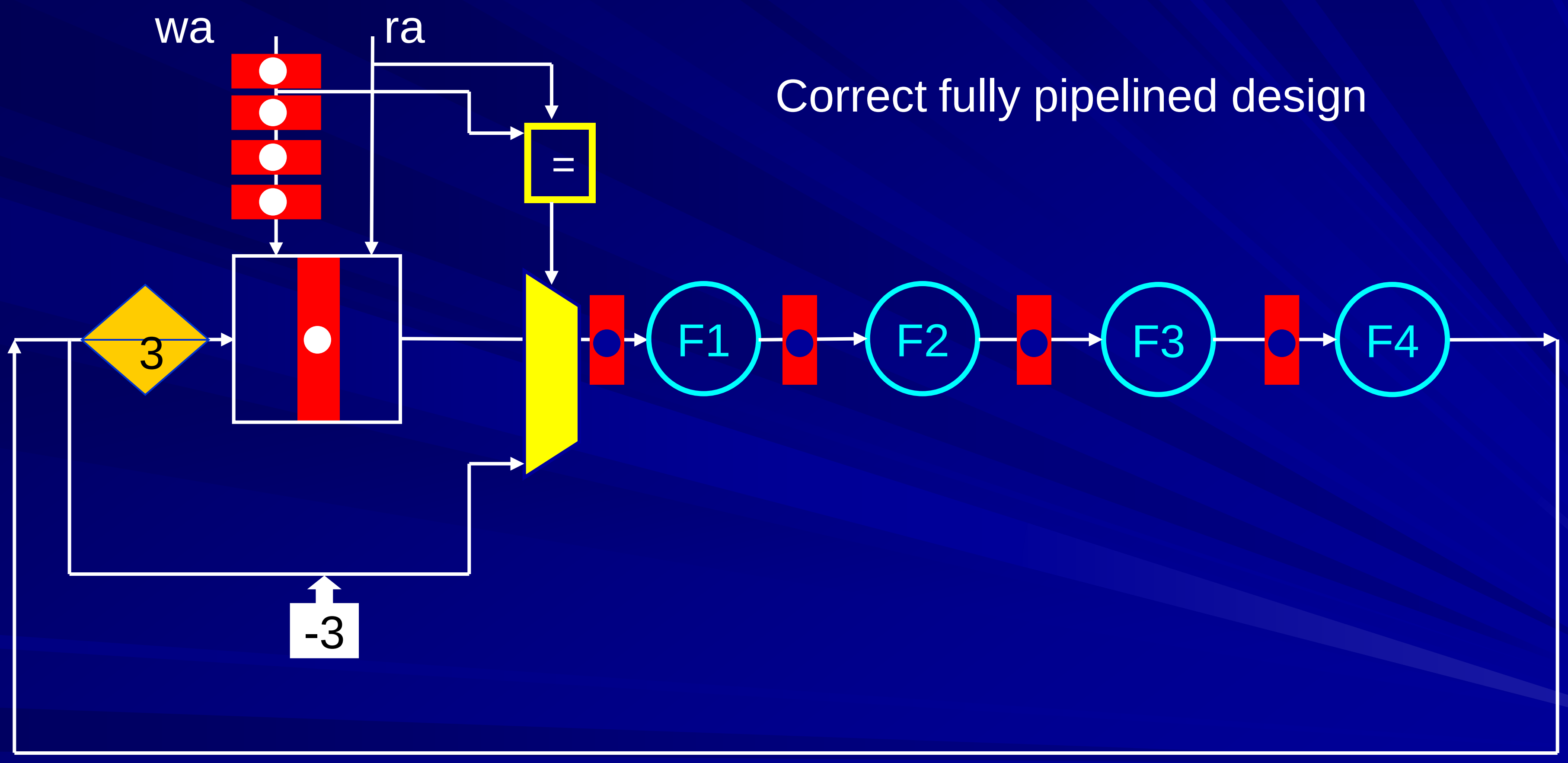


Size buffers

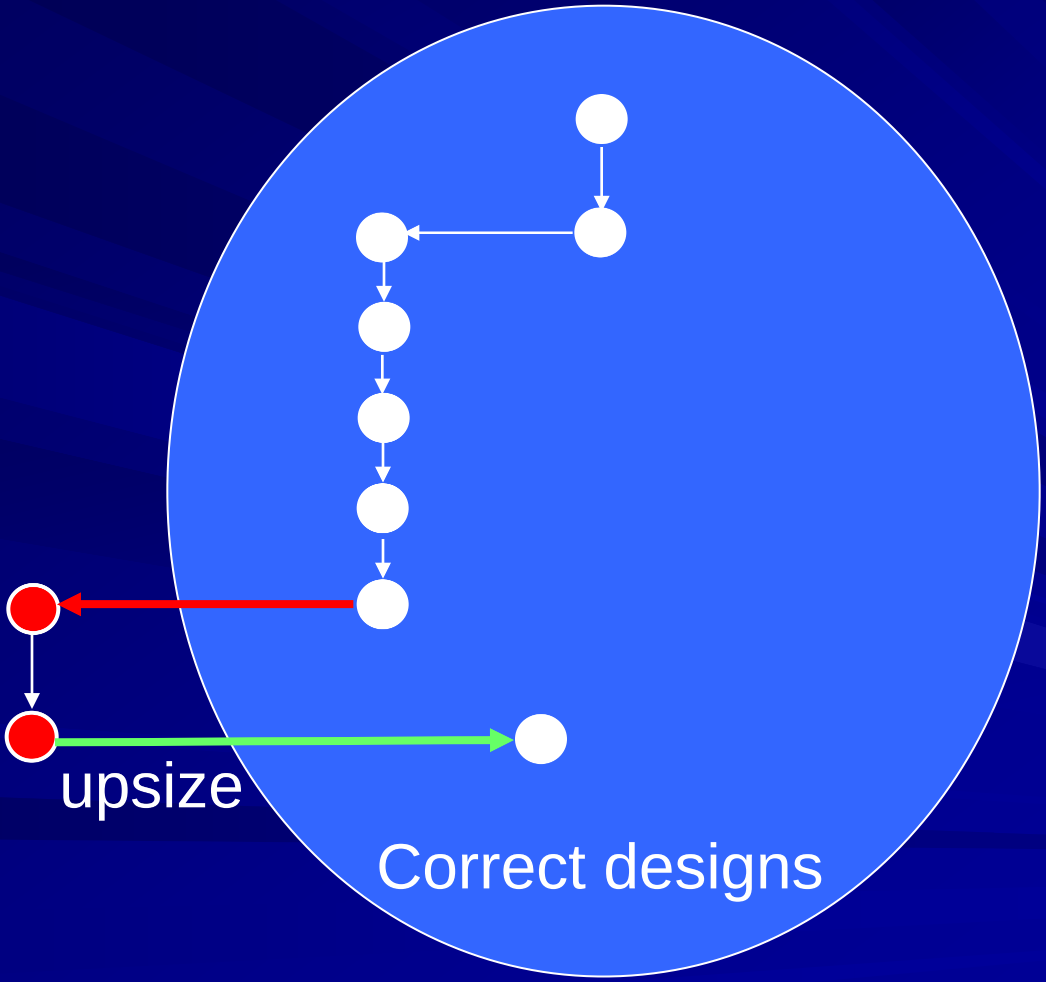
(by adding (0,k) buffers)

Pipelining by example

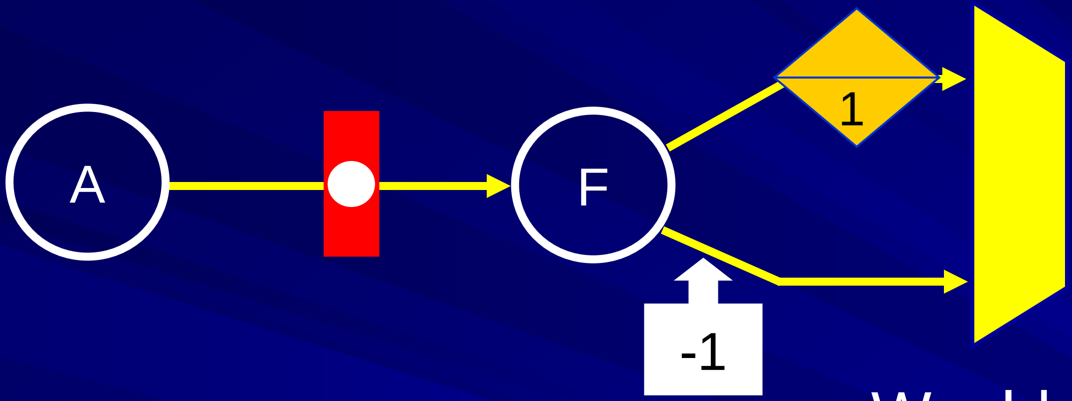
Correct fully pipelined design



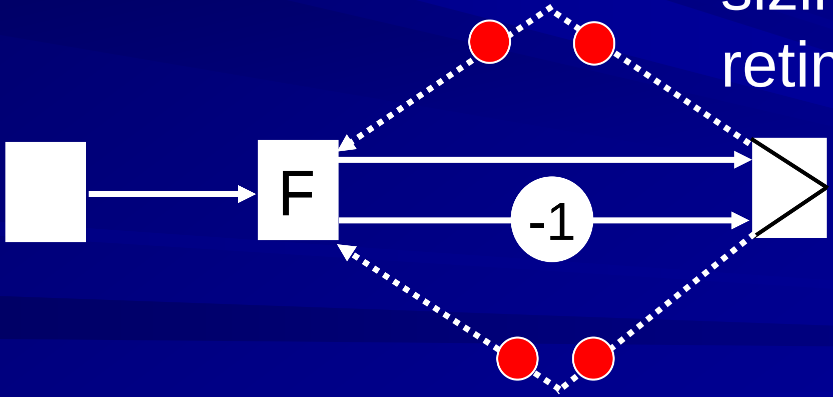
Transformations



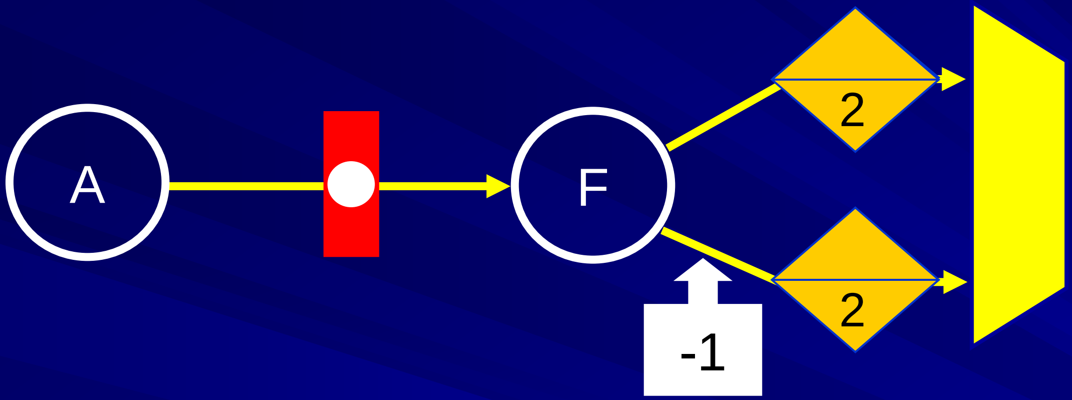
Retiming of Elastic Buffers

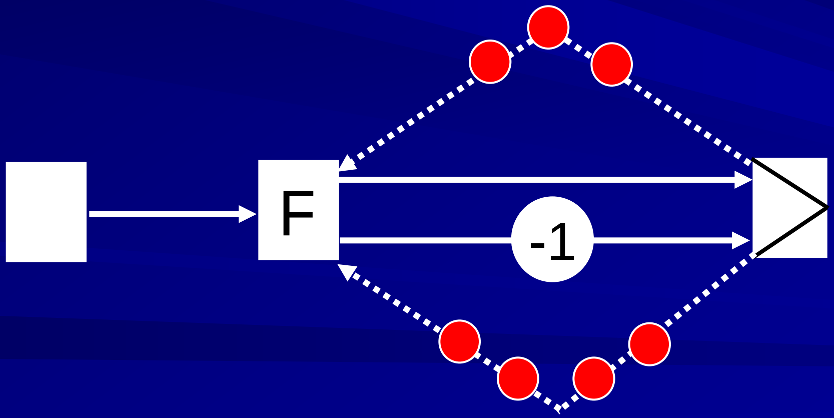


Would require solving capacity sizing problem for every retiming move



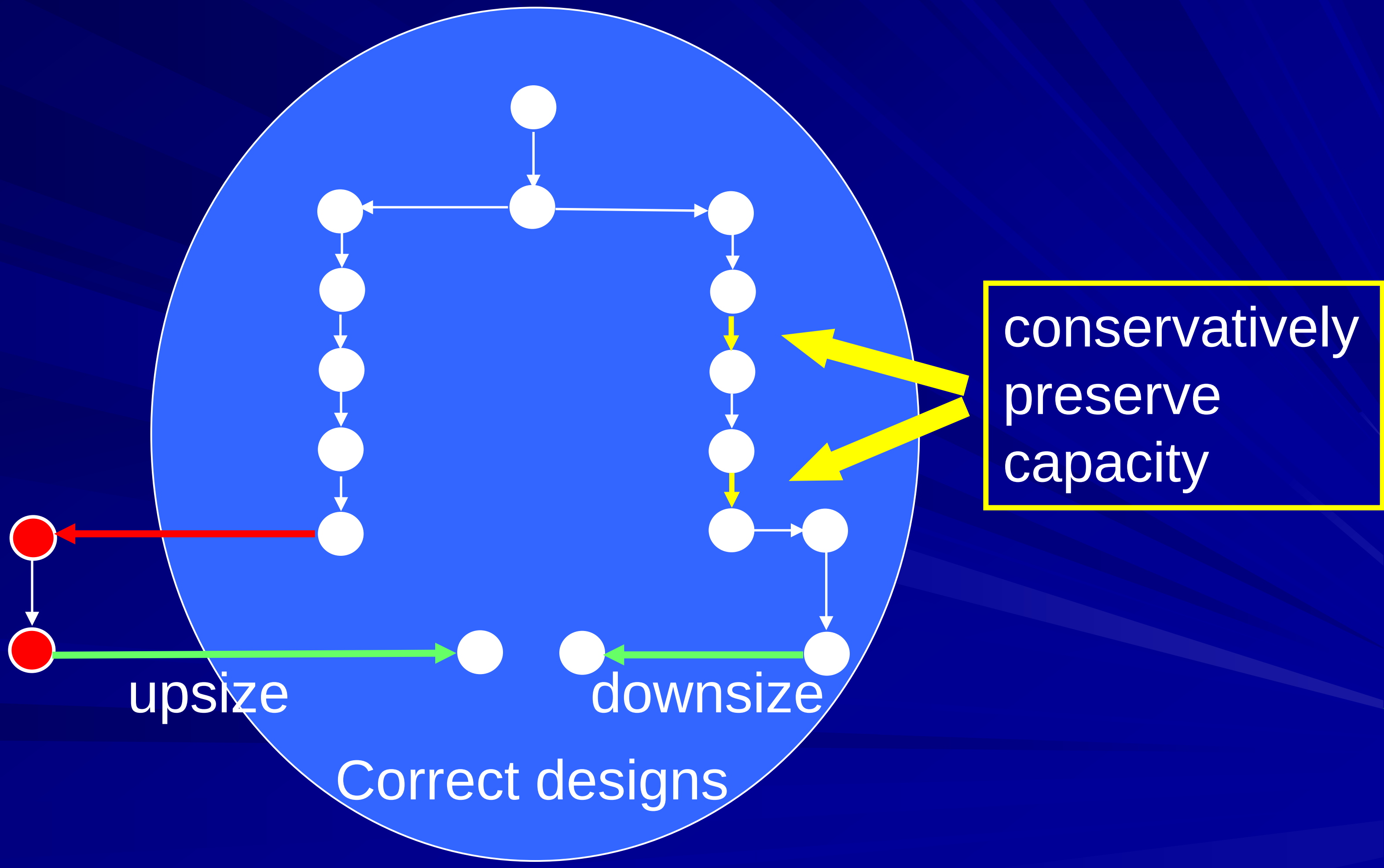
Retiming of Elastic Buffers





Conservatively preserve previous capacity

Transformations



Correctness (short story)

- Developed theory of elastic machines
(for late evaluation)
- Verify correctness of any elastic implementation = check conformance with the definition of elastic machine
- All SELF controllers are verified for conformance
- Elasticization is correct-by-construction

- Theory for early evaluation and negative delays is more challenging
  - Sketch of a theory, but no fully satisfactory compositional properties found yet
  - Verification done on concrete systems and controllers

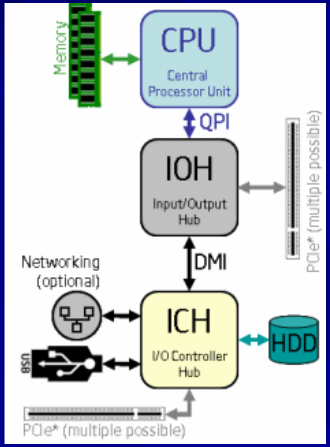
What is a Communication Fabric?

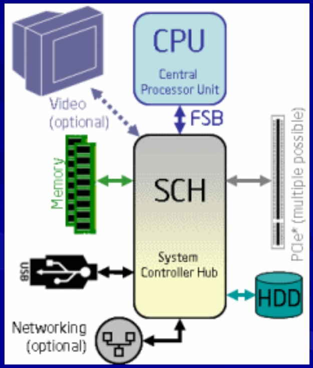
- Part of the design that pushes data around
- Glue between different IP blocks
- Include not only wires, but also…
  - switches, arbiters, routers, buffers and queues, addressing logic, logic managing credits, logic for cache coherency, starvation and deadlock prevention, clock and power down logic etc.
- Often has regular parts (e.g. ring or mesh topology), but need not be

- Elasticity is a natural requirement, but different notion of equivalence: only relative order matters

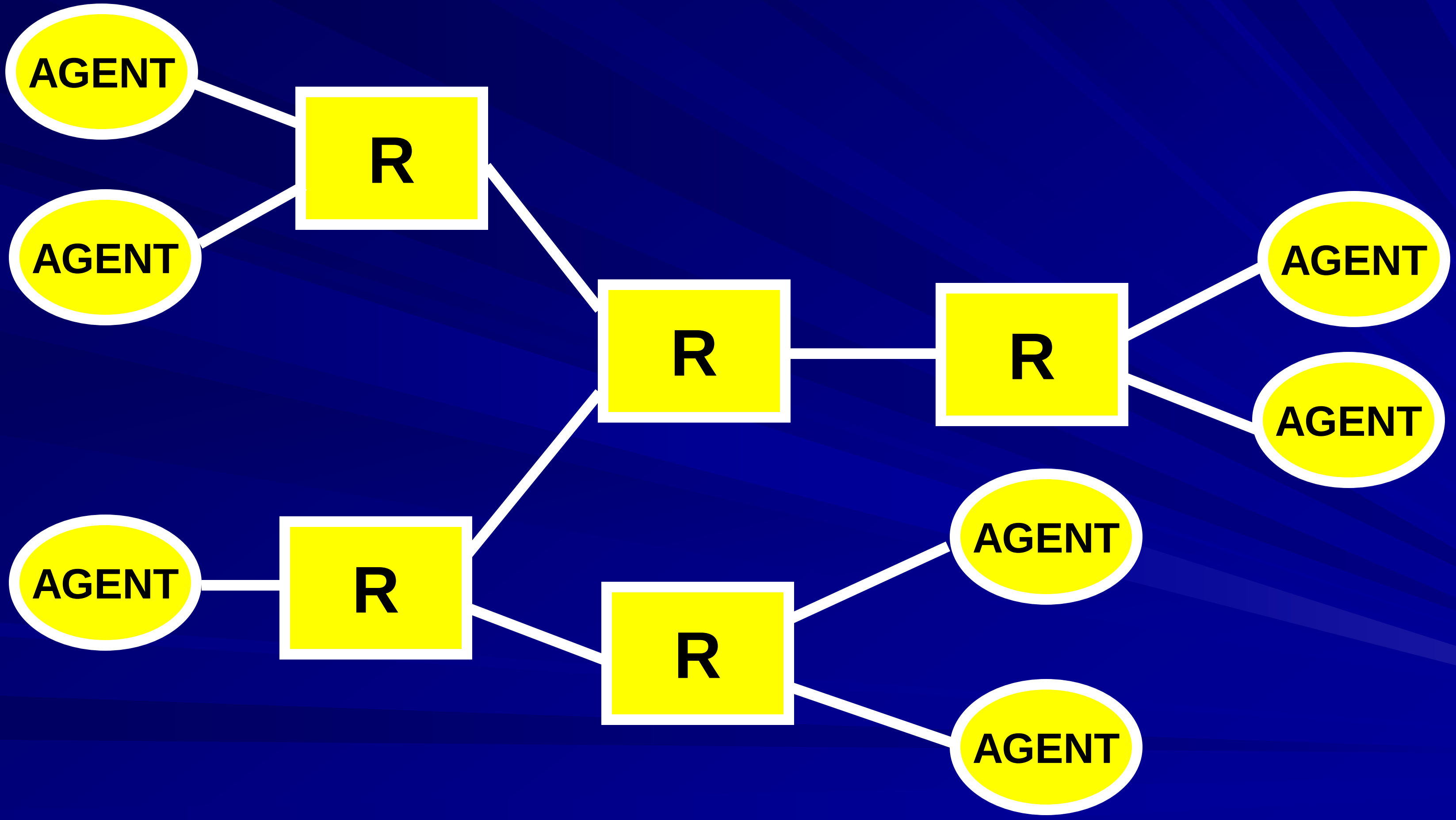
Many Communication Fabrics

- High-end interconnect
  - Connects cores in high-end chips
  - Implements cache coherence
- IO/Mem fabrics
  - PC MCH (Memory Control Hub), PCH, SCH
    - Implements PCI-compatible memory-mapped IO
  - SOC chips
    - System Interconnect
    - Memory Controller
    - Often simpler than PCI: no configuration, etc.
- Message fabrics
  - Power messages, sideband wires, etc. in most designs
  - Don’t care about performance



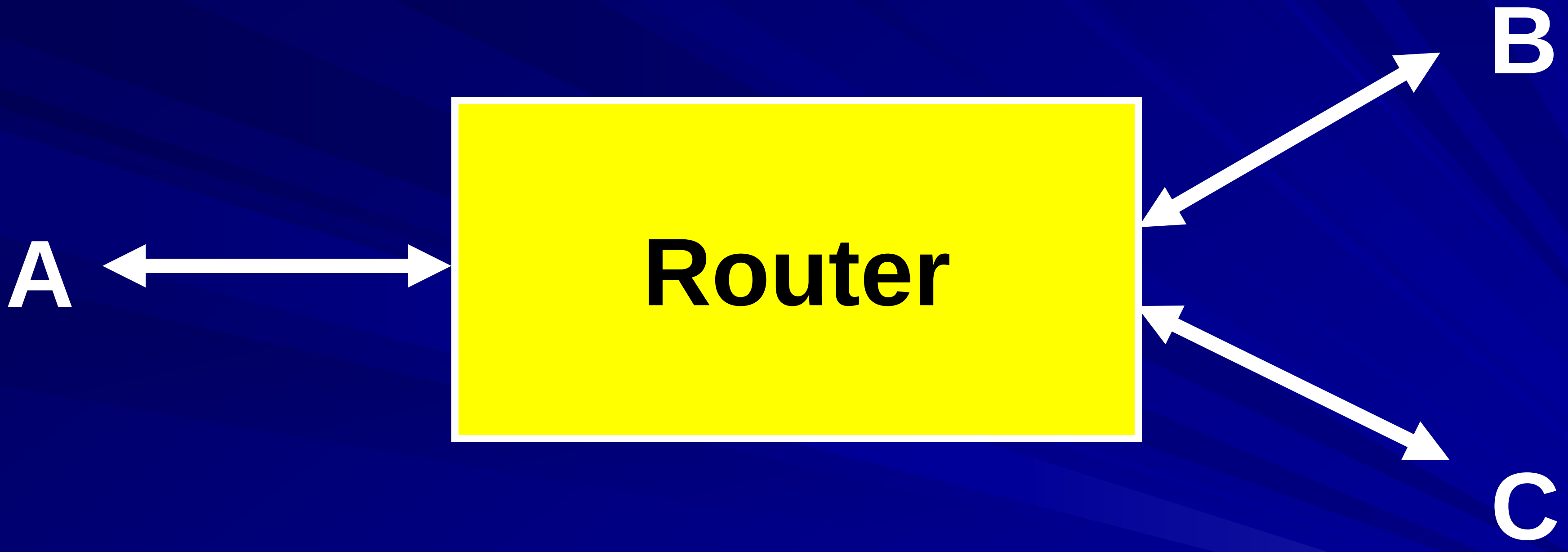


Tree topology NoC

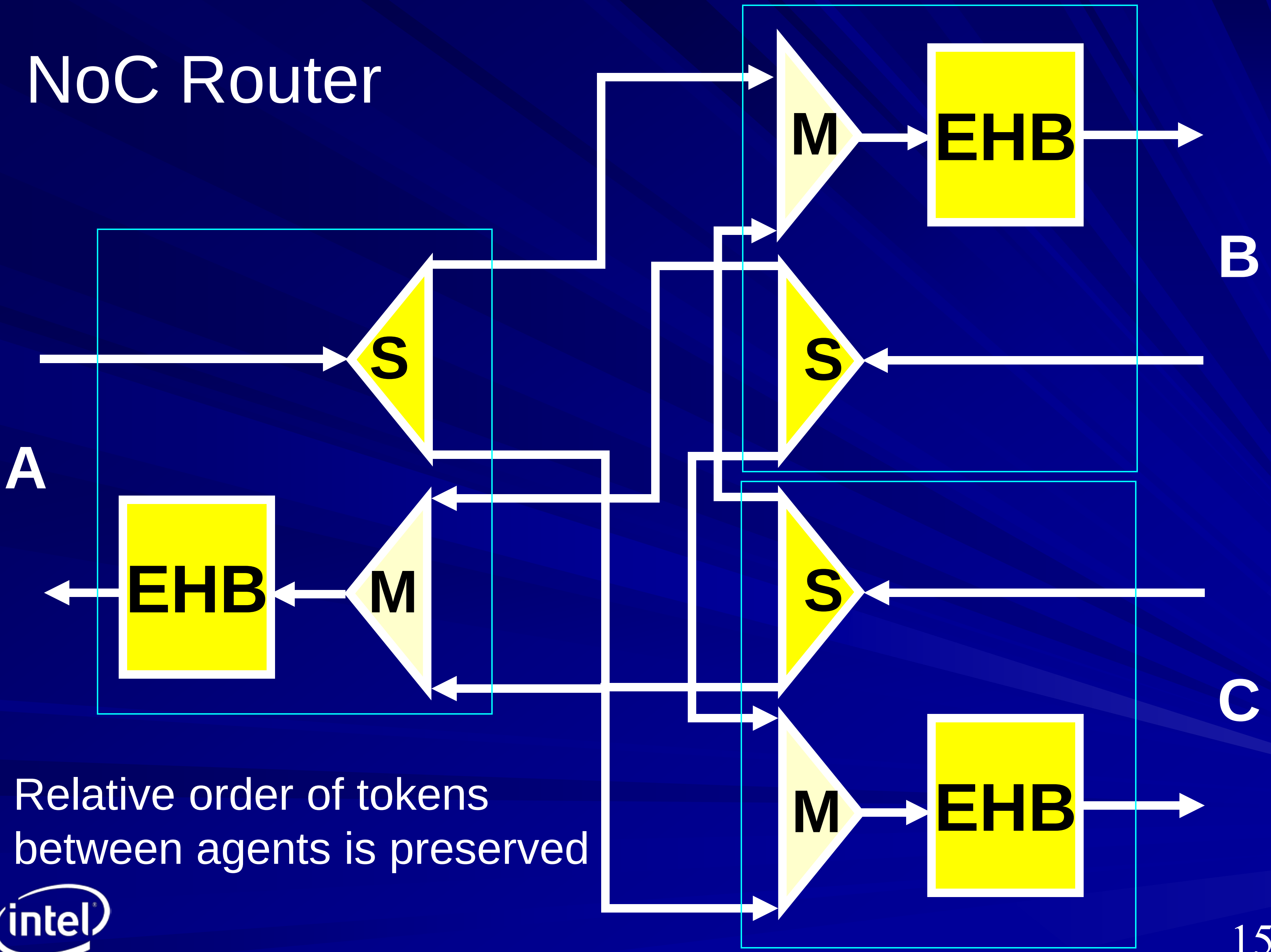


[In collaboration with Ken Stevens, Charles Dike, Bill Grundmann]

Router node interface

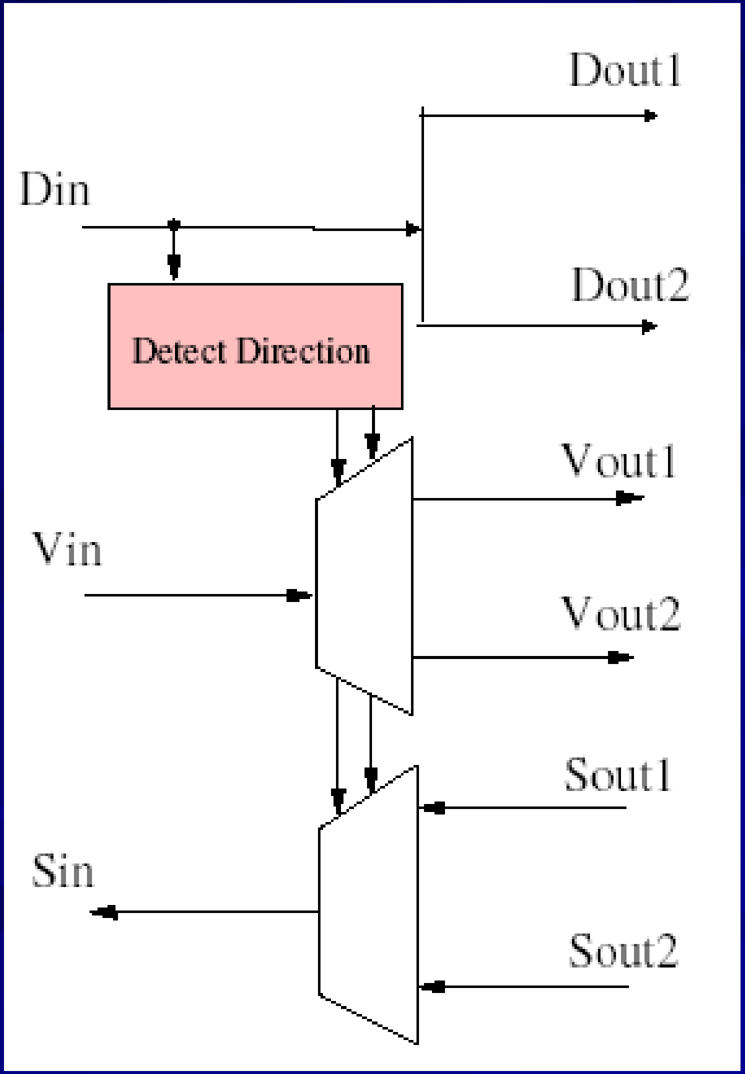


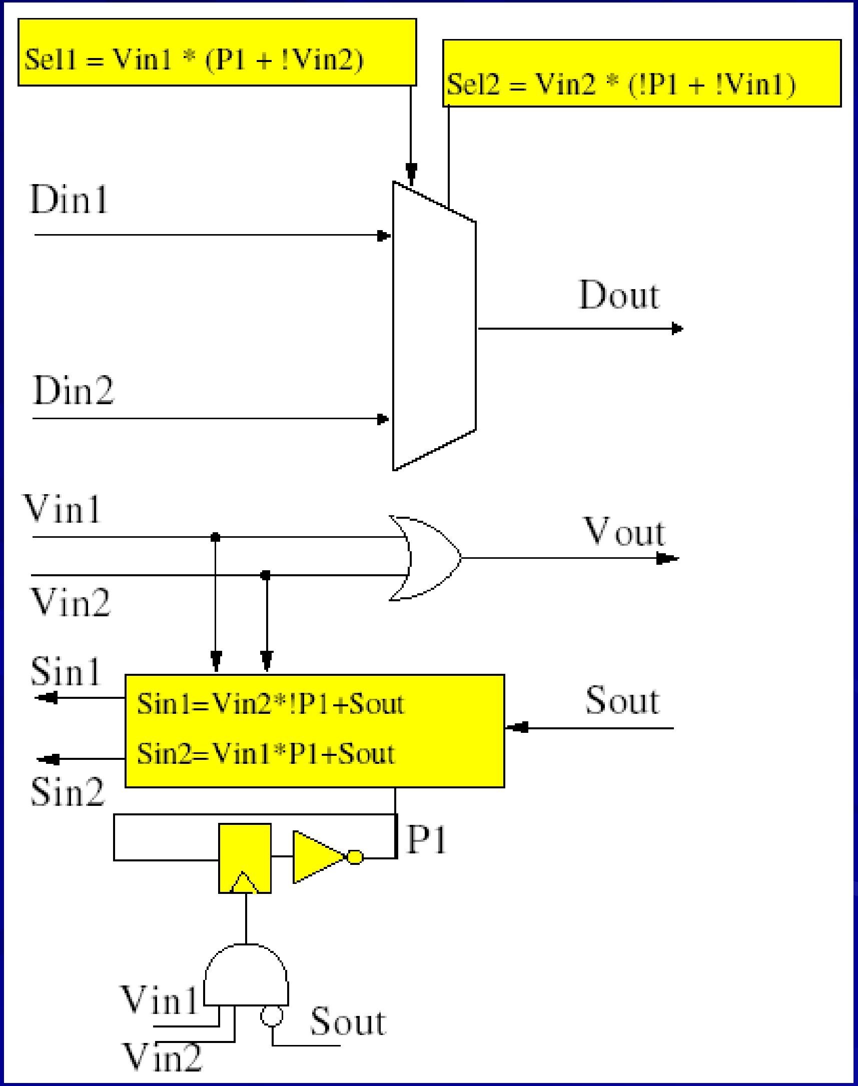
NoC Router



Relative order of tokens between agents is preserved

Switch and Merge





Some open problems

- Better performance analysis (bounds) for system with early evaluation
- **Given:** The number and sizes of IP blocks & communication requirements & message ordering constraints & flow control rates

  Find: Optimal floorplans & communication fabrics in (perf, area, energy) space
- Compositional theory of elastic machines with early evaluation
- **Given:** a class of communication fabric & message ordering constraints & flow control details

  Prove: no deadlocks, every message gets delivered

Summary

- SELF gives a low cost implementation of elastic machines
- Functionality is correct when latencies change
- New micro-architectural opportunities and new automatuion methods
- Compositional theory proving correctness
- Early evaluation - mechanism for performance and power optimization
- Applications to design of NoCs and communication fabrics

See reference list for *some* relevant publications

Bibliography on Synchronous Elastic (aka Latency Insensitive) Systems

July 20, 2009

Latency insensitive designs

[CMSV01, CSV02, CSV03, CM04, BMdS06a, Sve04, VA09]

SELF implementation and compilation to elastic designs

[CKG06, CK07, HB08]

Interlock pipelines

[JKB+02]

Synchronous translation of CSP

[OB97, PvB01]

Performance analysis

[JCK06]

Optimization

[LK03, BCKS07, CKC+08, CSV03, BMdS06b, BJC08]

Slack matching

[MM98]

Theory

[GTL03, KCKO06, CMSV01]

Variable latency units

[BMP97, BML+99, BCK09]

Petri Nets

[Mur89]

Early evaluation and event models with early evaluation

[BG03, CK07, TFRT02, RTTH05, AS06, KKTV94, YKK+96]

Microarchitectural transformations

[HE96, KKCGO08, GOCK09]

Desynchronization

[VM02, CKLS06]

Communication Fabrics & NoCs

[MOP+09]

References

[AS06] Manoj Ampalam and Montek Singh. Counterflow pipelining: Architectural support for preemption in asynchronous systems using anti-tokens. In *Proc. International Conf. Computer-Aided Design (ICCAD)*, pages 611–618, 2006.

[BCK09] D. Baneres, J. Cortadella, and M. Kishinevsky. Variable-latency design using function speculation. In *Proc. Design, Automation and Test in Europe (DATE)*, April 2009.

[BCKS07] Dmitry Bufistov, Jordi Cortadella, Mike Kishinevsky, and Sachin Sapatnekar. A general model for performance optimization of sequential systems. In *ICCAD '07: Proceedings of the 2007 IEEE/ACM international conference on Computer-aided design*, pages 362–369, 2007.

[BG03] C.F. Brej and J.D. Garside. Early output logic using anti-tokens. In *Int. Workshop on Logic Synthesis*, pages 302–309, May 2003.

[BJC08] D. Bufistov, J. Júlvez, and J. Cortadella. Performance optimization of elastic systems using buffer resizing and buffer insertion. In *Proc. International Conf. Computer-Aided Design (ICCAD)*, pages 442–448, November 2008.

[BMdS06a] J. Boucaron, J. Millo, and R. de Simone. Another glance at relay stations in latency-insensitive design. *Electr. Notes Theor. Comput. Sci.*, 146(2):41–59, 2006.

[BMdS06b] J. Boucaron, J. Millo, and R. de Simone. Latency-insensitive design and central repetitive scheduling. In *IEEE-ACM International Conference MEMOCODE'06*, pages 175–183, 2006.

[BML$^+$99] L. Benini, G. De Micheli, A. Lioy, E. Macii, G. Odasso, and M. Poncino. Automatic synthesis of large telescopic units based on near-minimum timed supersetting. *IEEE Transactions on Computers*, 48(8):769–779, 1999.

[BMP97] Luca Benini, Enrico Macii, and Massimo Poncino. Telescopic units: increasing the average throughput of pipelined designs by adaptive latency control. In *DAC '97: Proceedings of the 34th annual conference on Design automation*, pages 22–27, New York, NY, USA, 1997. ACM Press.

[CK07] J. Cortadella and M. Kishinevsky. Synchronous elastic circuits with early evaluation and token counterflow. In *Proc. ACM/IEEE Design Automation Conference*, pages 416–419, June 2007.

[CKC$^+$08] Jordi Cortadella, Mike Kishinevsky, Josep Carmona, Dmitry Bufistov, and Jorge Julvez. Elasticity and Petri nets. *LNCS Transactions on Petri Nets and Other Models of Concurrency (ToPNoC)*, 1:221 – 249, February 2008.

[CKG06] J. Cortadella, M. Kishinevsky, and B. Grundmann. Synthesis of synchronous elastic architectures. In *Proc. ACM/IEEE Design Automation Conference*, pages 657–662, July 2006.

[CKLS06] Jordi Cortadella, Alex Kondratyev, Luciano Lavagno, and Christos Sotiriou. Desynchronization: Synthesis of asynchronous circuits from synchronous specifications. *IEEE Transactions on Computer-Aided Design*, 25(10):1904–1921, 2006.

[CM04] M.R. Casu and L. Macchiarulo. A new approach to latency insensitive design. In *Proc. Digital Automation Conference (DAC)*, pages 576–581, June 2004.

[CMSV01] L. Carloni, K.L. McMillan, and A.L. Sangiovanni-Vincentelli. Theory of latency-insensitive design. *IEEE Transactions on Computer-Aided Design*, 20(9):1059–1076, September 2001.

[CSV02] L.P. Carloni and A.L. Sangiovanni-Vincentelli. Coping with latency in SoC design. *IEEE Micro, Special Issue on Systems on Chip*, 22(5):12, October 2002.

[CSV03] L. Carloni and A.L. Sangiovanni-Vincentelli. Combining retiming and recycling to optimize the performance of synchronous circuits. In *16th Symp. on Integrated Circuits and System Design (SBCCI)*, pages 47–52, September 2003.

[GOCK09] Marc Galceran-Oms, Jordi Cortadella, and Mike Kishinevsky. Speculation in elastic systems. In *Proc. International Workshop on Logic Synthesis*, July 2009.

[GTL03] P. Le Guernic, J.-P. Talpin, and J.-Ch. Le Lann. Polychrony for system design. *Journal of Circuits, Systems and Computers*, 12(3):261–304, April 2003.

[HB08] Greg Hoover and Forrest Brewer. Synthesizing synchronous elastic flow networks. In *DATE '08: Proceedings of the conference on Design, automation and test in Europe*, pages 306–311, 2008.

[HE96] S. Hassoun and C. Ebeling. Architectural retiming: Pipelining latency-constrained circuits. In *Proc. ACM/IEEE Design Automation Conference*, pages 708–713, June 1996.

[JCK06] J. Júlvez, J. Cortadella, and M. Kishinevsky. Performance analysis of concurrent systems with early evaluation. In *Proc. International Conf. Computer-Aided Design (ICCAD)*, November 2006.

[JKB$^+$02] Hans M. Jacobson, Prabhakar N. Kudva, Pradip Bose, Peter W. Cook, Stanley E. Schuster, Eric G. Mercer, and Chris J. Myers. Synchronous interlocked pipelines. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pages 3–12, April 2002.

[KCKO06] Sava Krstic, Jordi Cortadella, Michael Kishinevsky, and John O'Leary. Synchronous elastic networks. In *FMCAD*, pages 19–30. IEEE Computer Society, 2006.

[KKCGO08] T. Kam, M. Kishinevsky, J. Cortadella, and M. Galceran-Oms. Correct-by-construction microarchitectural pipelining. In *Proc. International Conf. Computer-Aided Design (ICCAD)*, pages 434–441, November 2008.

[KKTV94] Michael Kishinevsky, Alex Kondratyev, Alexander Taubin, and Victor Varshavsky. *Concurrent Hardware: The Theory and Practice of Self-Timed Design.* Series in Parallel Computing. John Wiley & Sons, 1994.

[LK03] R. Lu and C.-K. Koh. Performance optimization of latency insensitive systems through buffer queue sizing of communication channels. In *Proc. International Conf. Computer-Aided Design (ICCAD)*, pages 227–231, November 2003.

[MM98] R. Manohar and A. J. Martin. Slack elasticity in concurrent computing. In *Proc. 4th Int. Conf. on the Mathematics of Program Construction*, volume 1422 of *Lecture Notes in Computer Science*, pages 272–285, 1998.

[MOP$^+$09] Radu Marculescu, Umit Y. Ogras, Li-Shiuan Peh, Natalie Enright Jerger, and Yatin Hoskote. Outstanding research problems in noc design: System, microarchitecture, and circuit perspectives. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 28(1):3 – 21, 2009.

[Mur89] T. Murata. Petri Nets: Properties, analysis and applications. *Proceedings of the IEEE*, pages 541–580, April 1989.

[OB97] John O'Leary and Geoffrey Brown. Synchronous emulation of asynchronous circuits. *IEEE Transactions on Computer-Aided Design*, 16(2):205–209, February 1997.

[PvB01] Ad Peeters and Kees van Berkel. Synchronous handshake circuits. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pages 86–95. IEEE Computer Society Press, March 2001.

[RTTH05] R. Reese, M. Thornton, C. Traver, and D. Hemmendinger. Early evaluation for performance enhancement in phased logic. *IEEE Transactions on Computer-Aided Design*, 24(4):532–550, April 2005.

[Sve04] Christer Svensson. Synchronous latency insensitive design. In *10th International Symposium on Advanced Research in Asynchronous Circuits and Systems (ASYNC 2004)*, page 3, 2004.

[TFRT02] M. Thornton, K. Fazel, R. Reese, and C. Traver. Generalized early evaluation in self-timed circuits. In *Proc. Design, Automation and Test In Europe (DATE)*, March 2002.

[VA09] M. Vijayaraghavan and Arvind. Bounded dataflow networks and latency-insensitive circuits. In *Proceedings of the 7th International Conference on Formal Methods and Models for Codesign (MEMOCODE)*, July 2009.

[VM02] Victor Varshavsky and Vyacheslav Marakhovsky. GALA (globally asynchronous - locally arbitrary) design. In J. Cortadella, A. Yakovlev, and G. Rozenberg, editors, *Concurrency and Hardware Design*, volume 2549 of *Lecture Notes in Computer Science*, pages 61–107. Springer-Verlag, 2002.

[YKK+96] Alexandre Yakovlev, Michael Kishinevsky, Alex Kondratyev, Luciano Lavagno, and Marta Pietkiewicz-Koutny. On the models for asynchronous circuit behaviour with OR causality. *Formal Methods in System Design*, 9(3):189–233, 1996.