Synchronous Elastic Systems

Mike Kishinevsky and Jordi Cortadella

Intel
Strategic CAD Labs
Hillsboro, USA

Universitat Politecnica
de Catalunya
Barcelona, Spain

Contributors to SELF research

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

Agenda

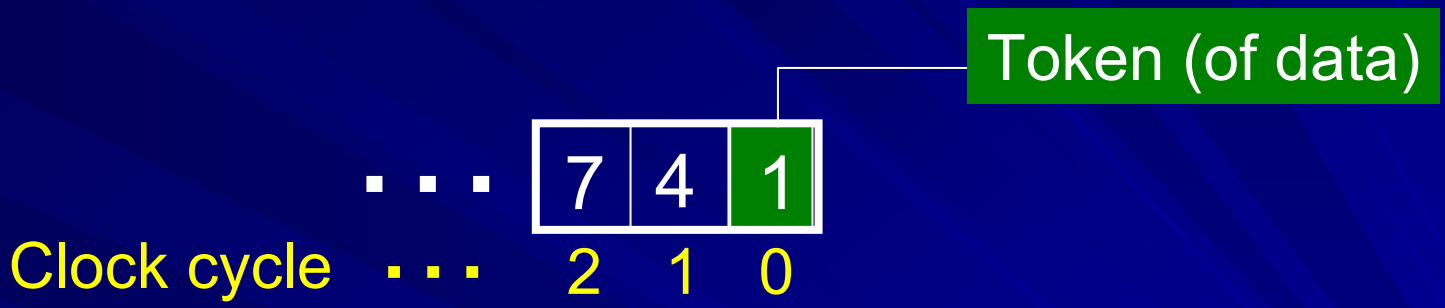
I. Basics of elastic systems

II. Early evaluation and performance analysis

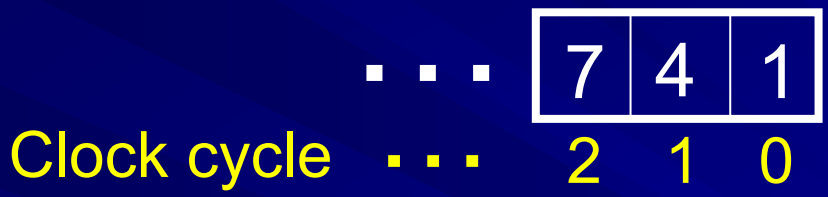
III. Applications of elastic systems

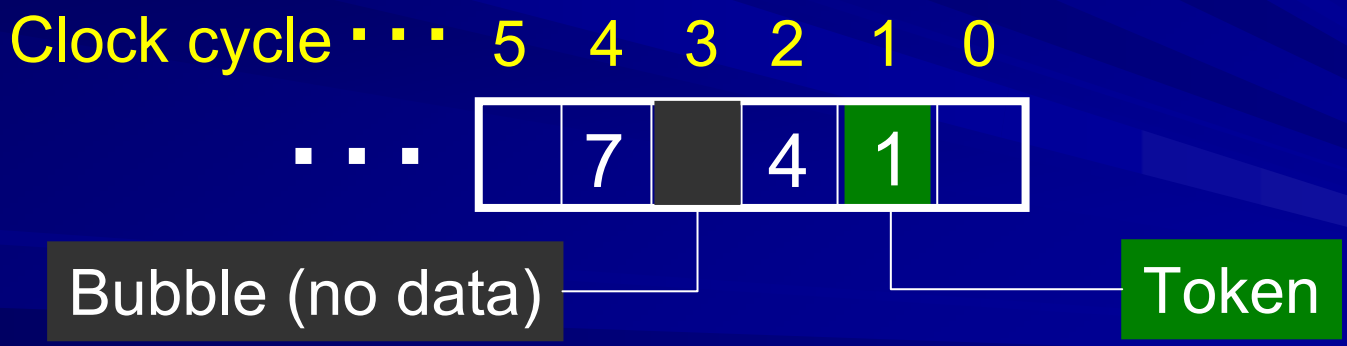
IV. Demo of SELF compiler

Synchronous Stream of Data



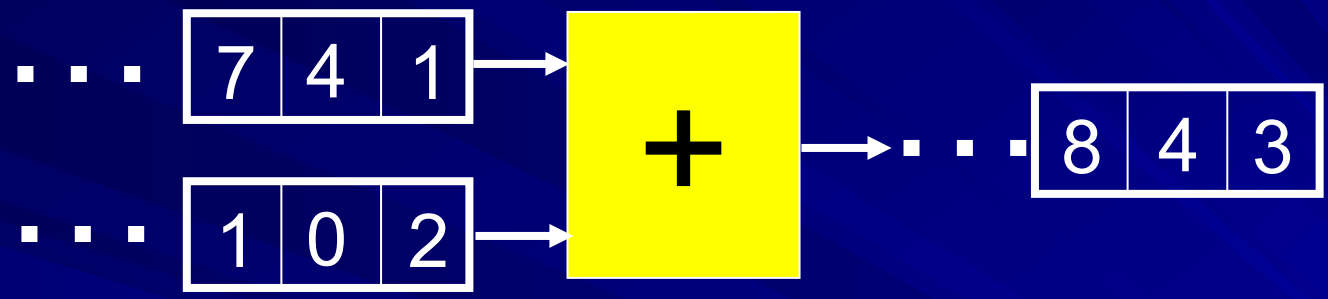
Synchronous Elastic Stream



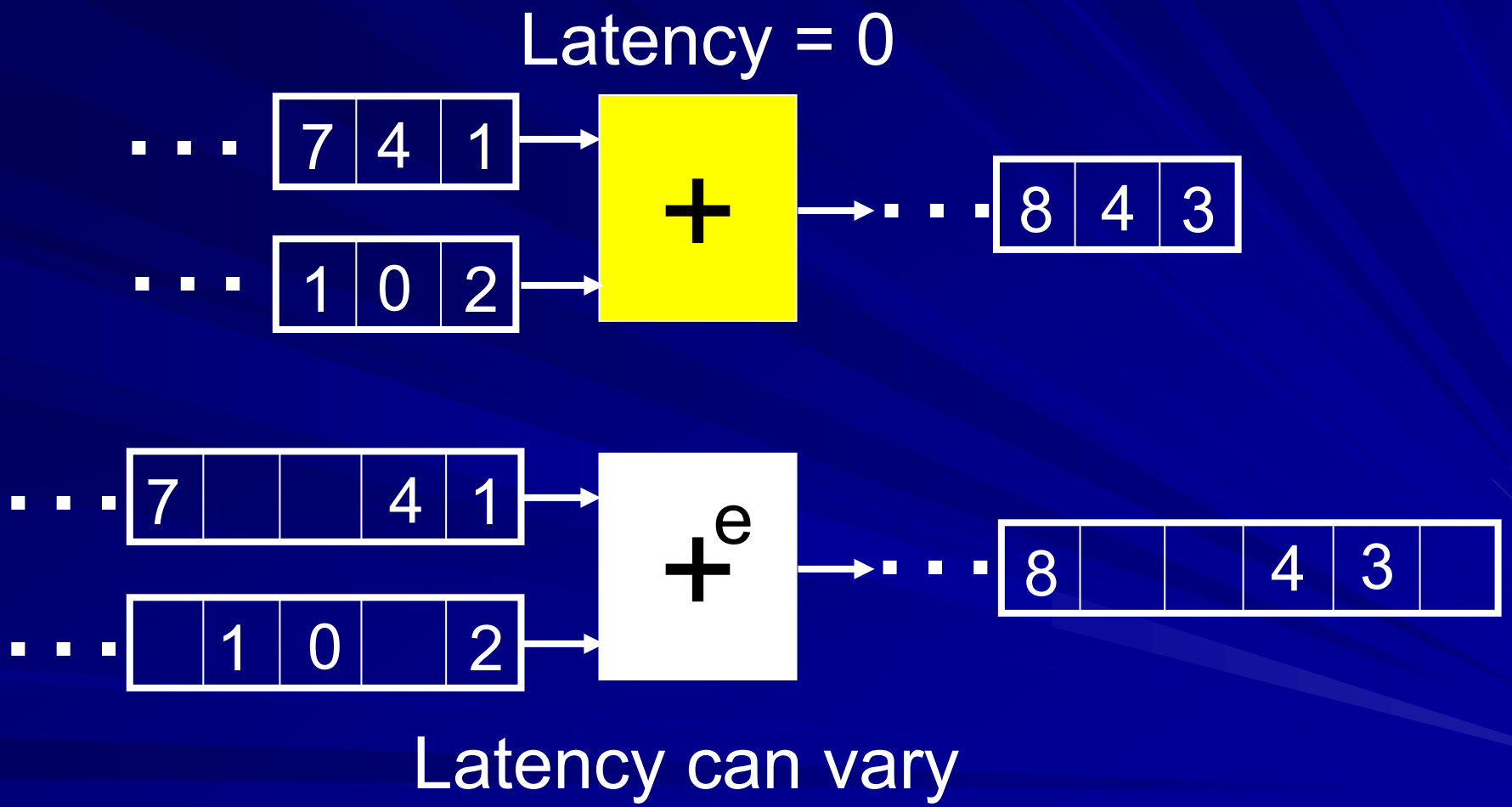


Synchronous Circuit

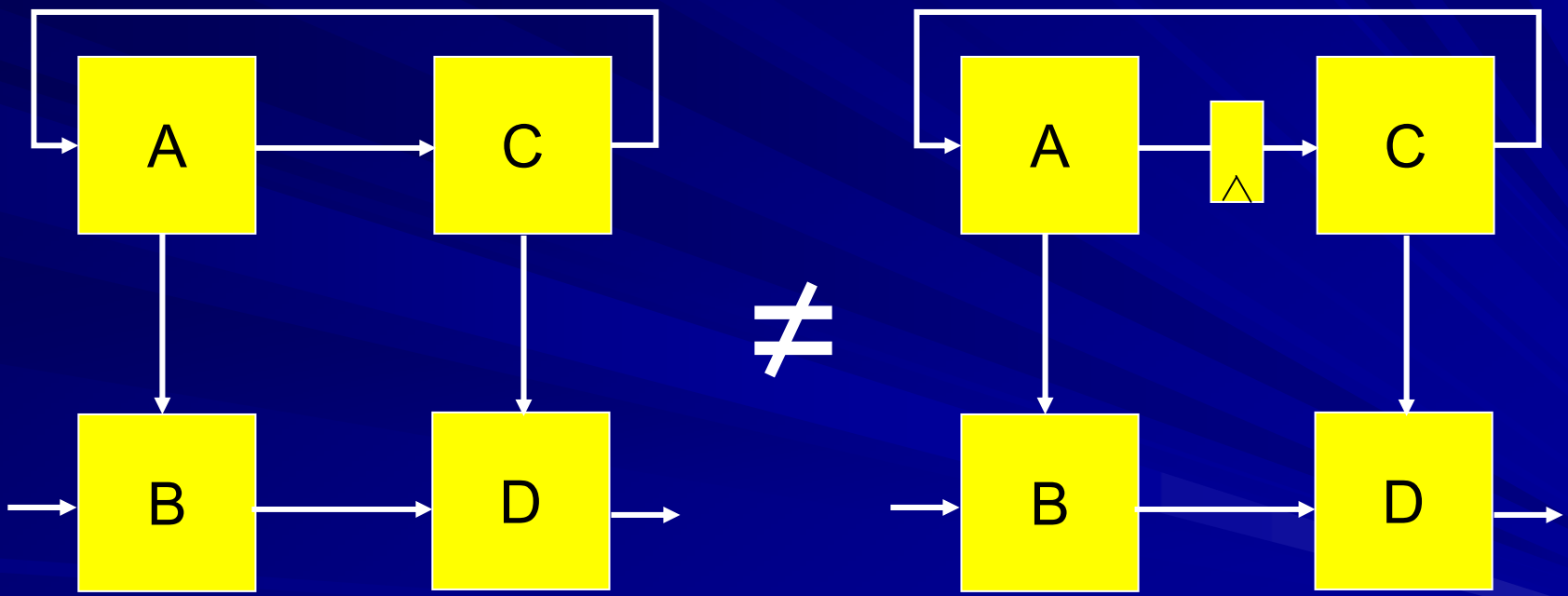
Latency = 0



Synchronous Elastic Circuit

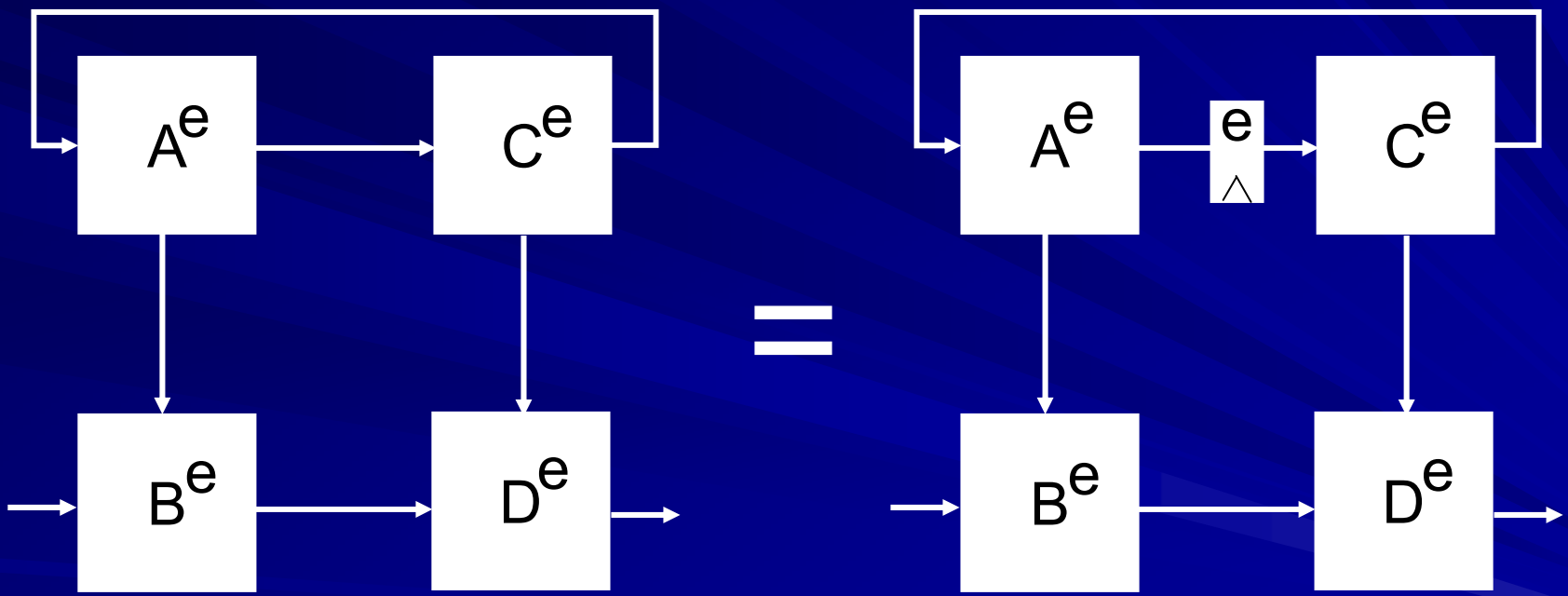


Ordinary Synchronous System



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
  Synchronous elastic - in discrete time

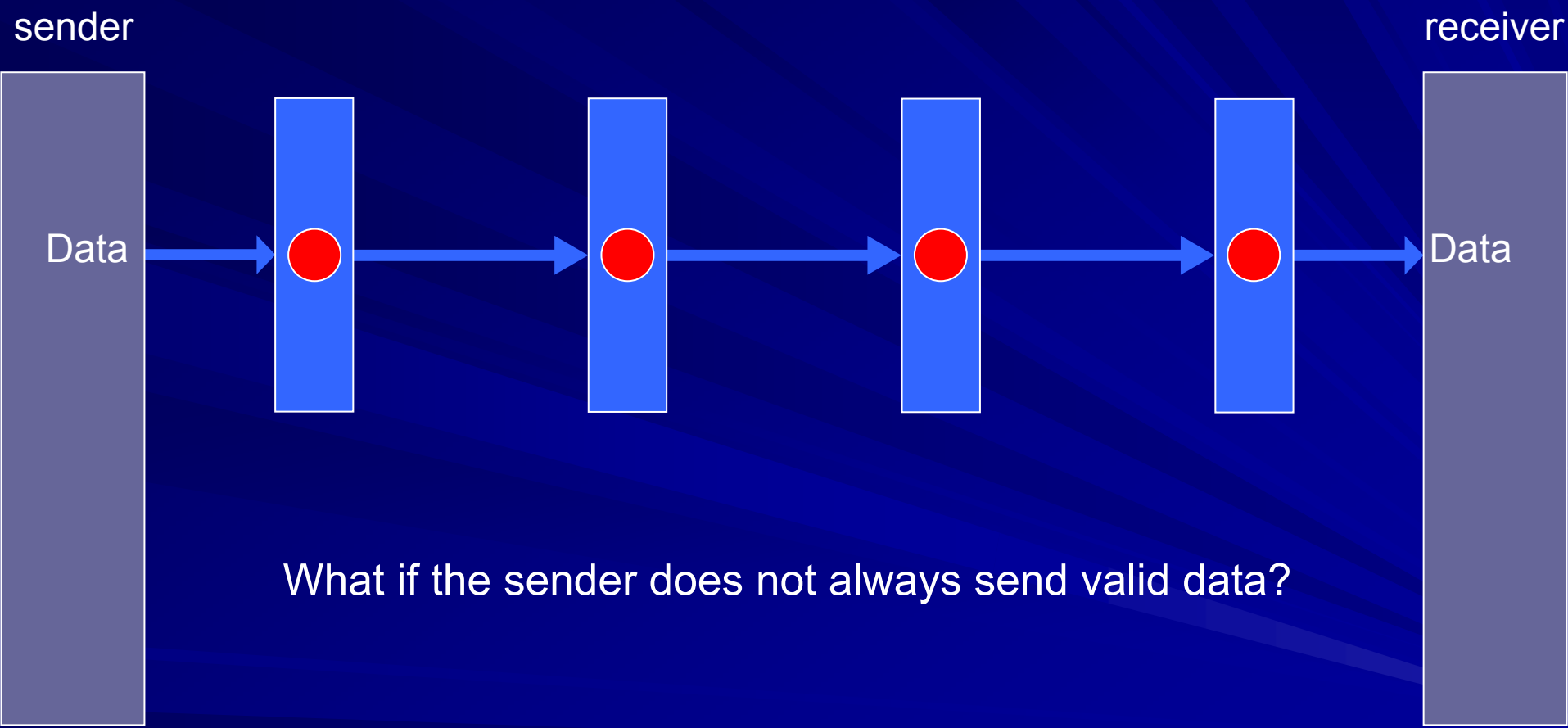
Why

- Scalable
- Modular (Plug & Play)
- Better energy-delay trade-offs
  (design for typical case instead of worst case)
- New micro-architectural opportunities
  in digital design
- Not asynchronous: use existing design experience, CAD tools and flows... but have some advantages of asynchronous

How to Design Synchronous Elastic Systems

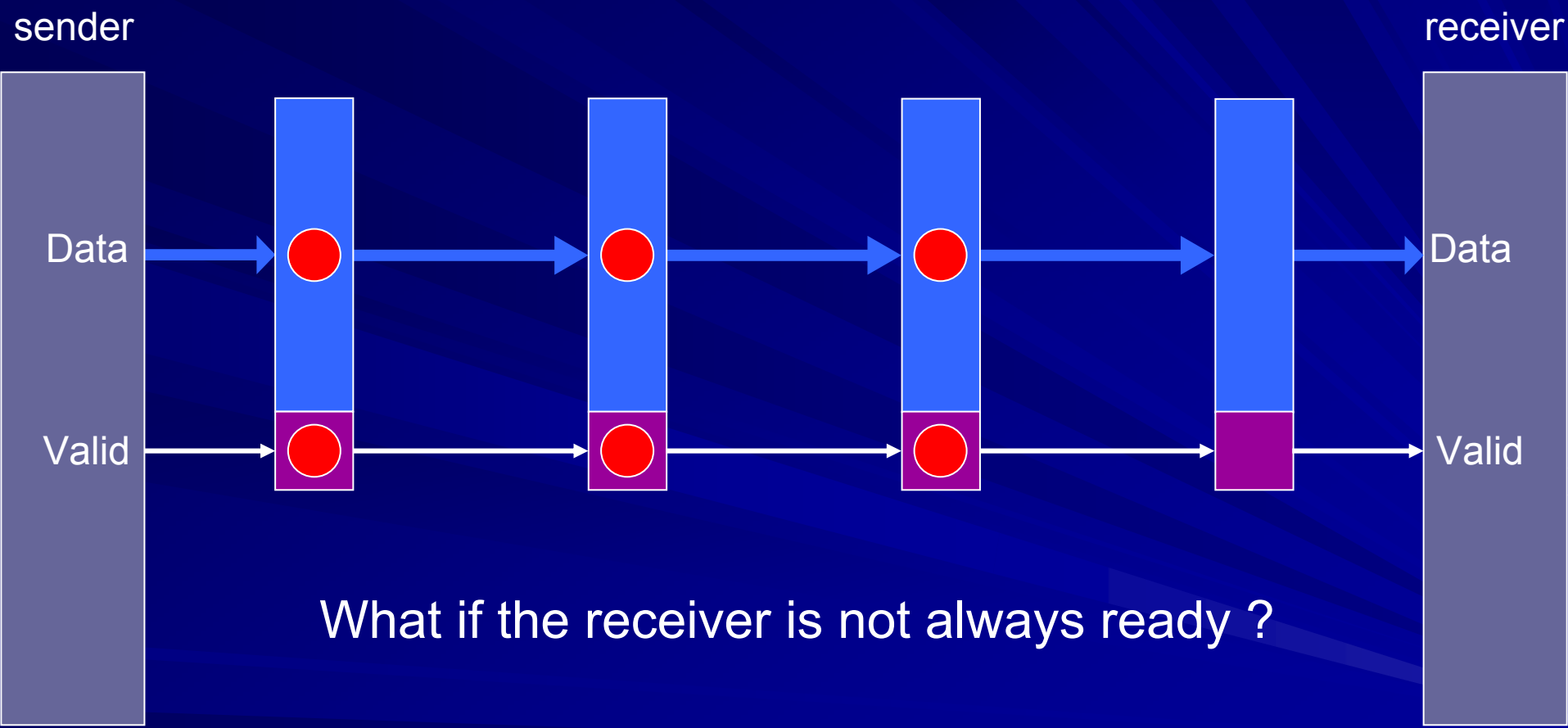
- Example of the implementation:
  SELF = Synchronous Elastic Flow
- Others are possible

Pipelined communication



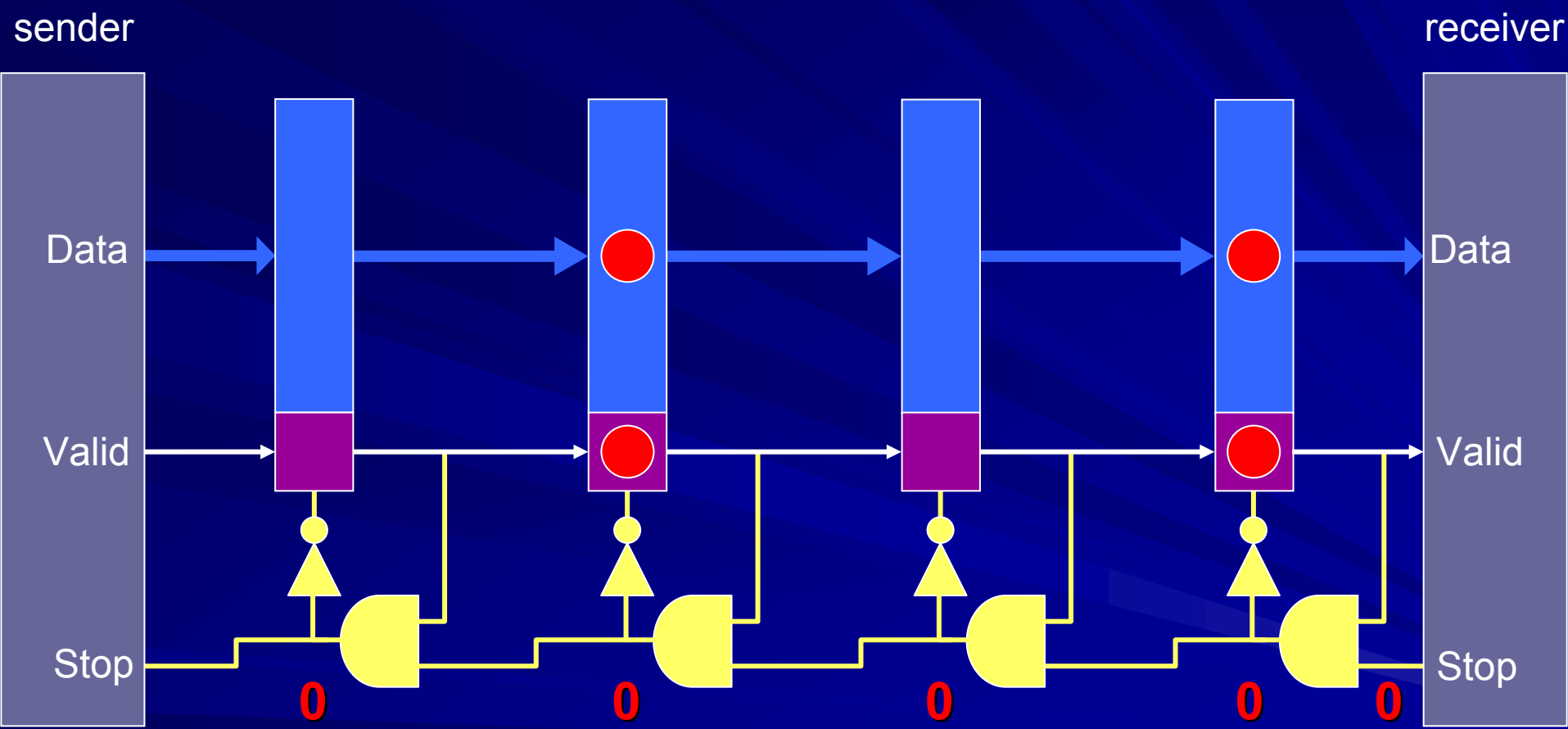
What if the sender does not always send valid data?

The Valid bit

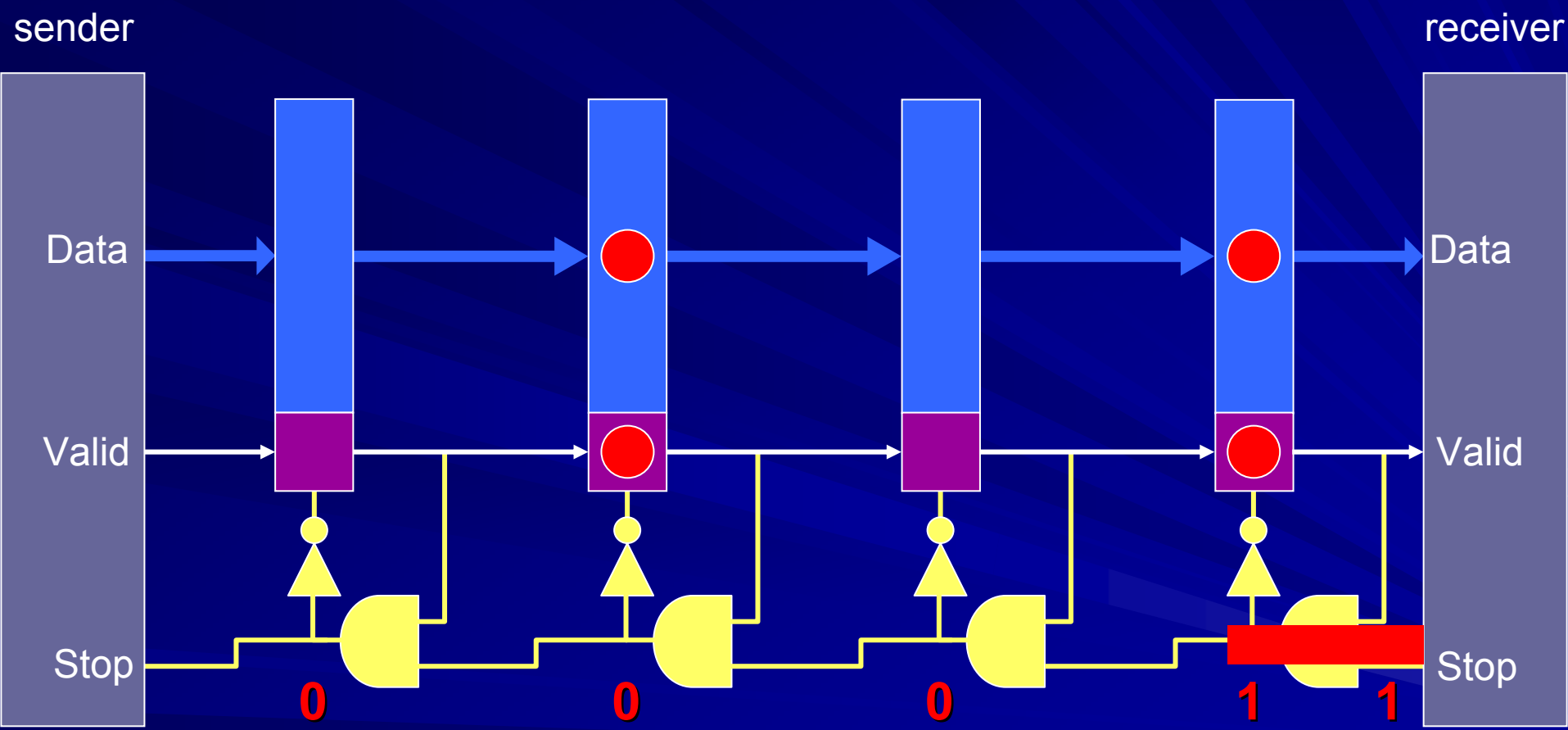


What if the receiver is not always ready ?

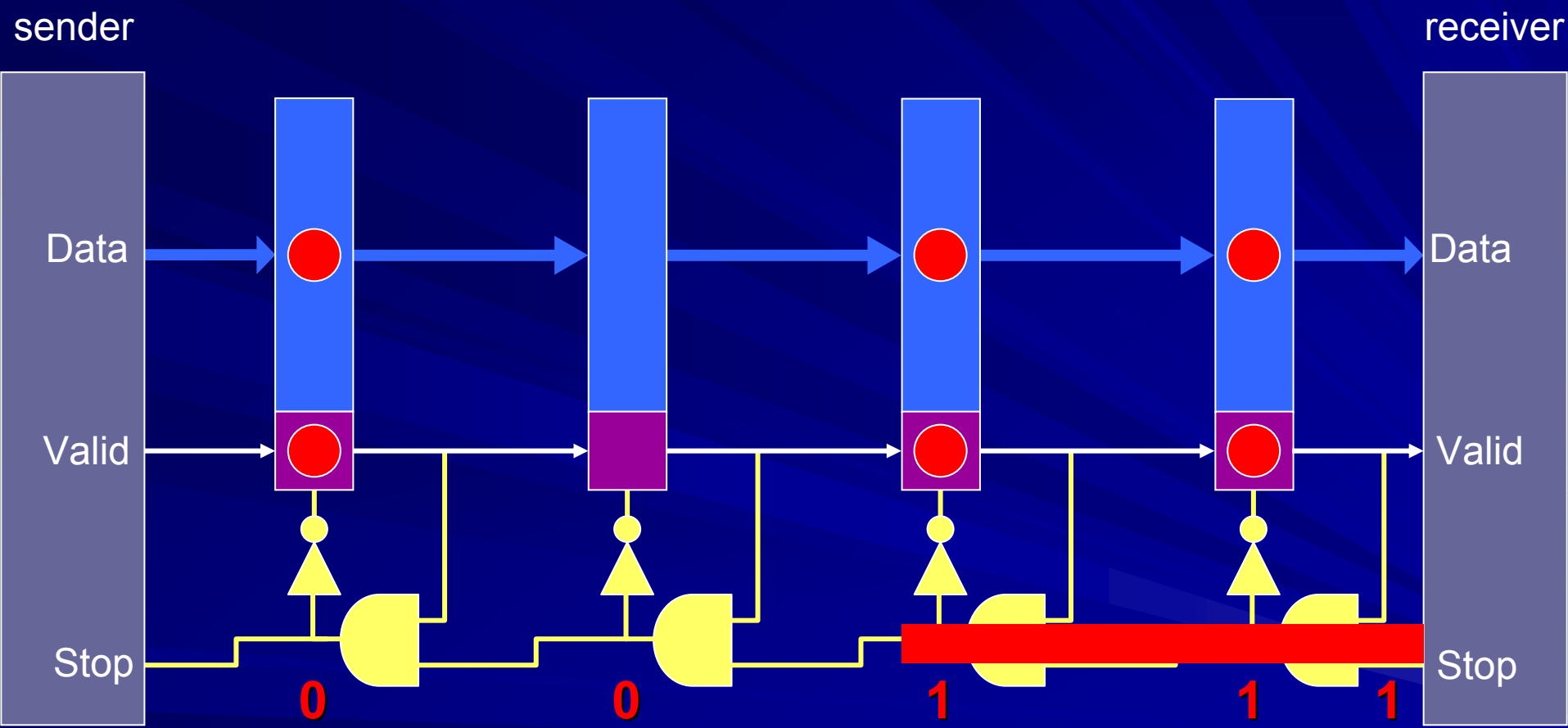
The Stop bit



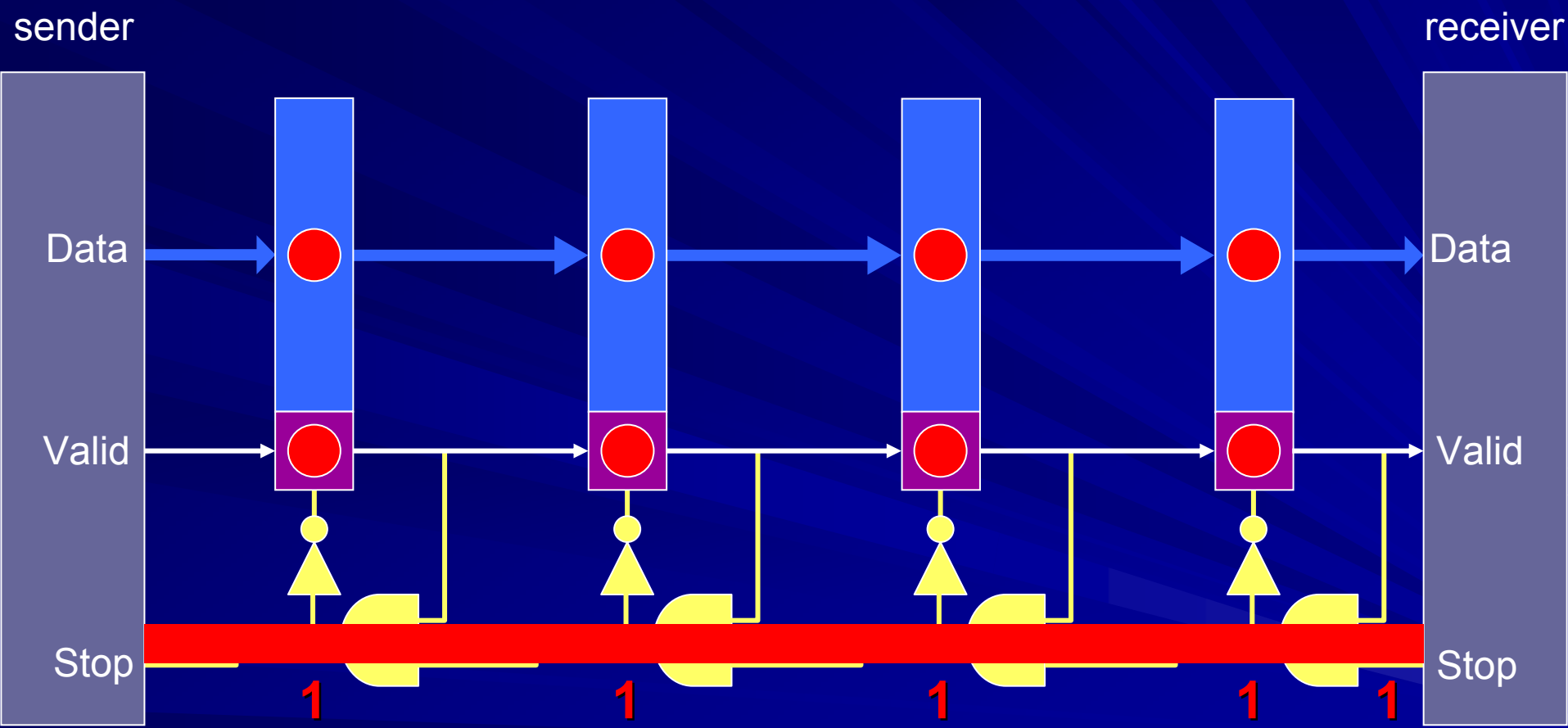
The Stop bit



The Stop bit

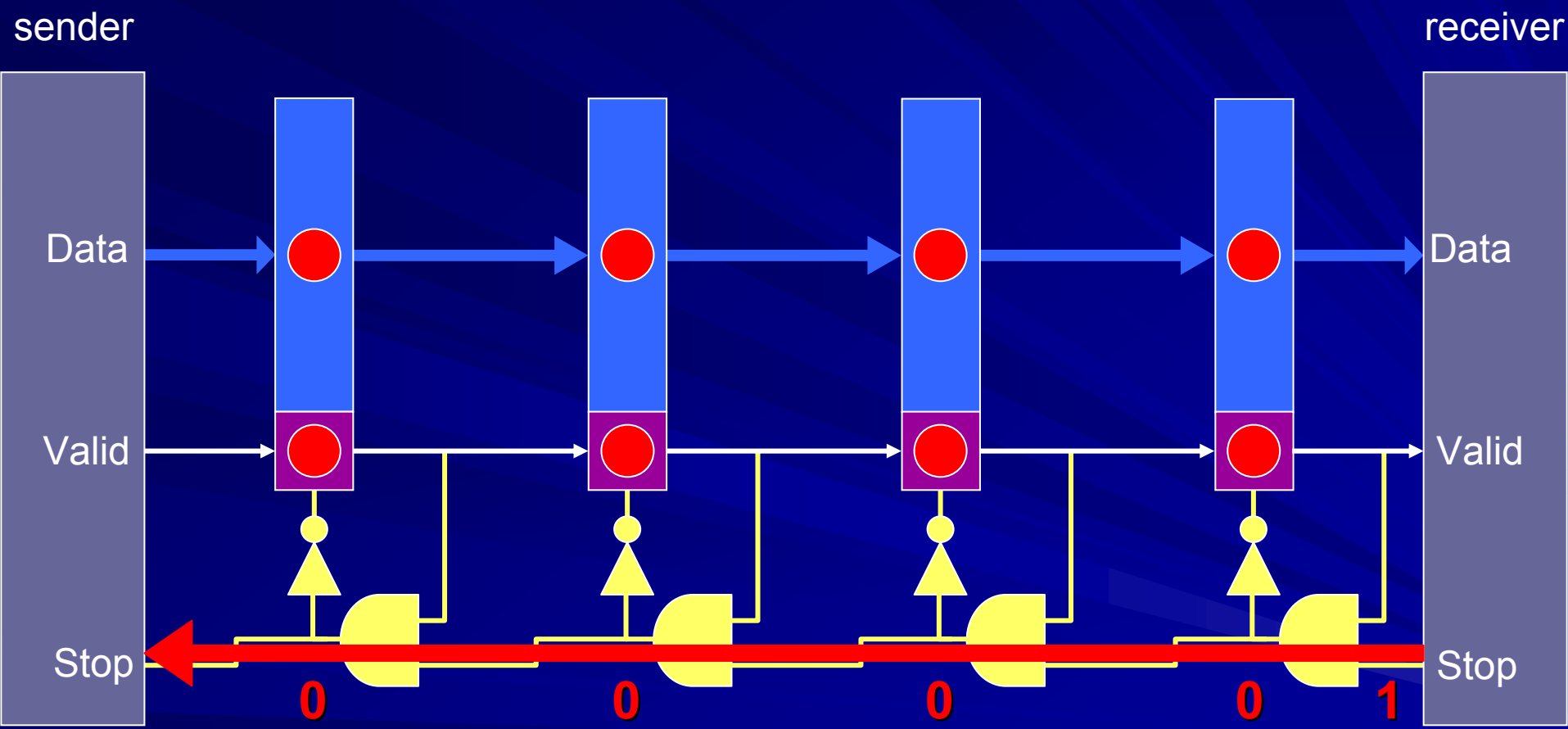


The Stop bit



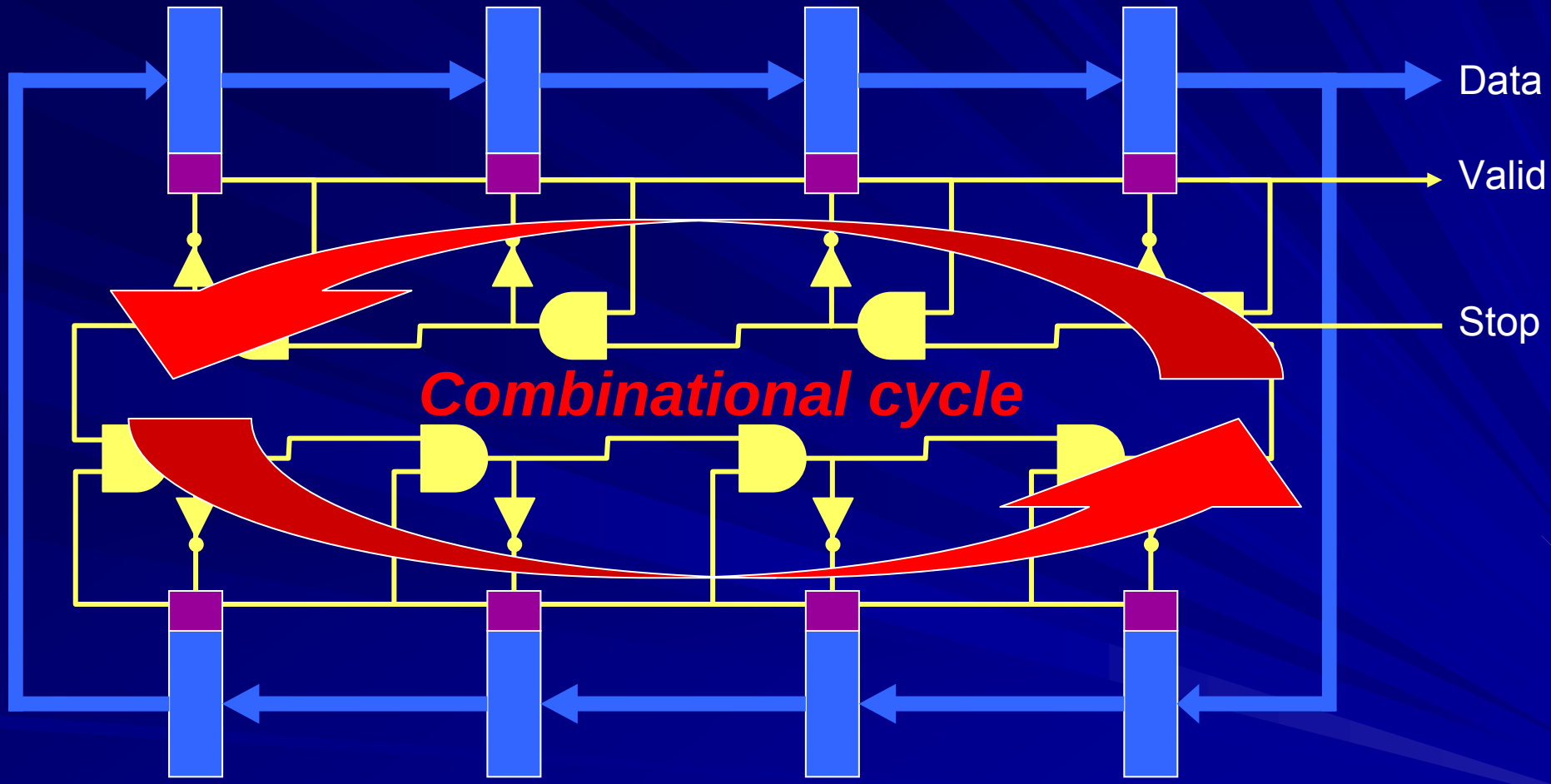
Back-pressure

The Stop bit



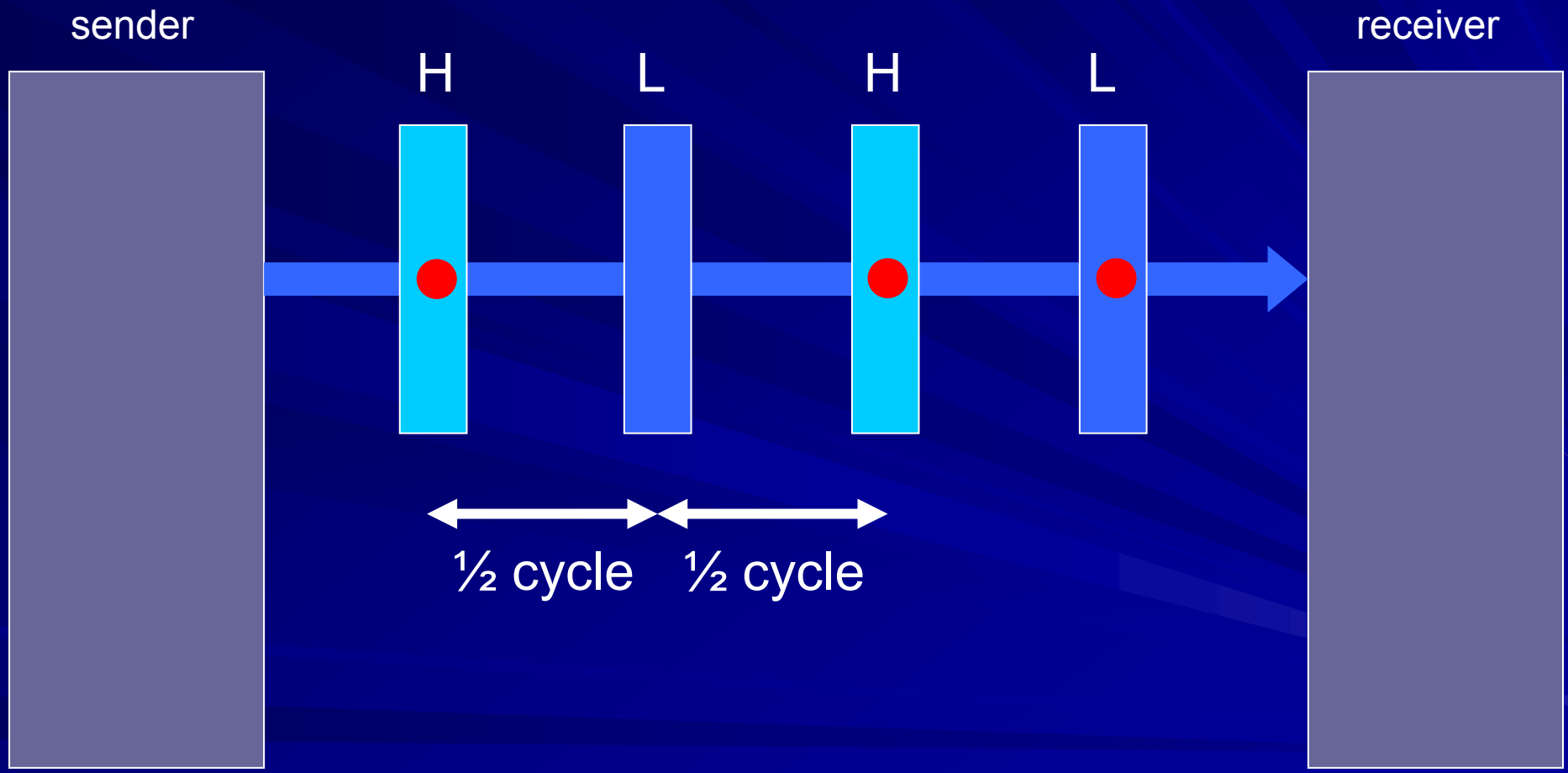
Long combinational path

Cyclic structures



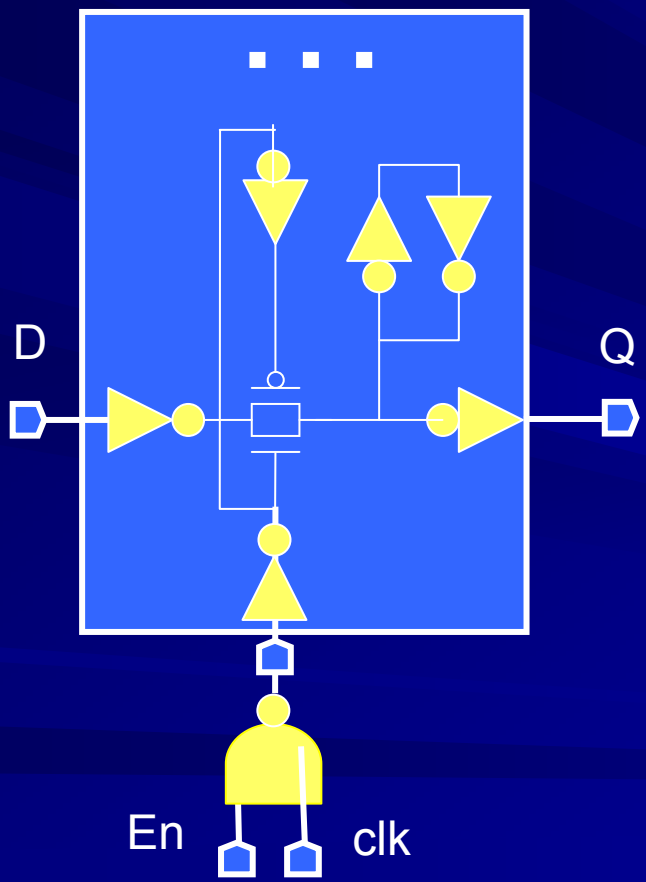
One can build circuits with combinational cycles (constructive cycles by Berry), but synthesis and timing tools do not like them

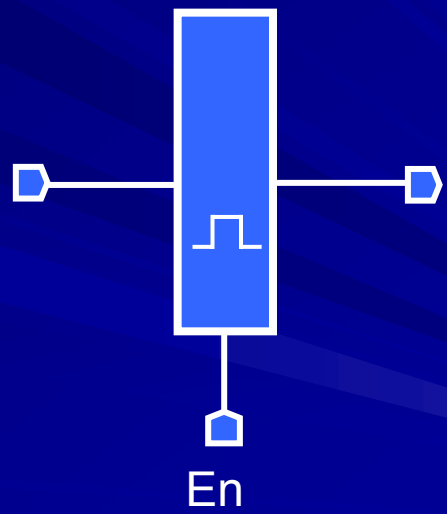
Example: pipelined linear communication chain with transparent latches



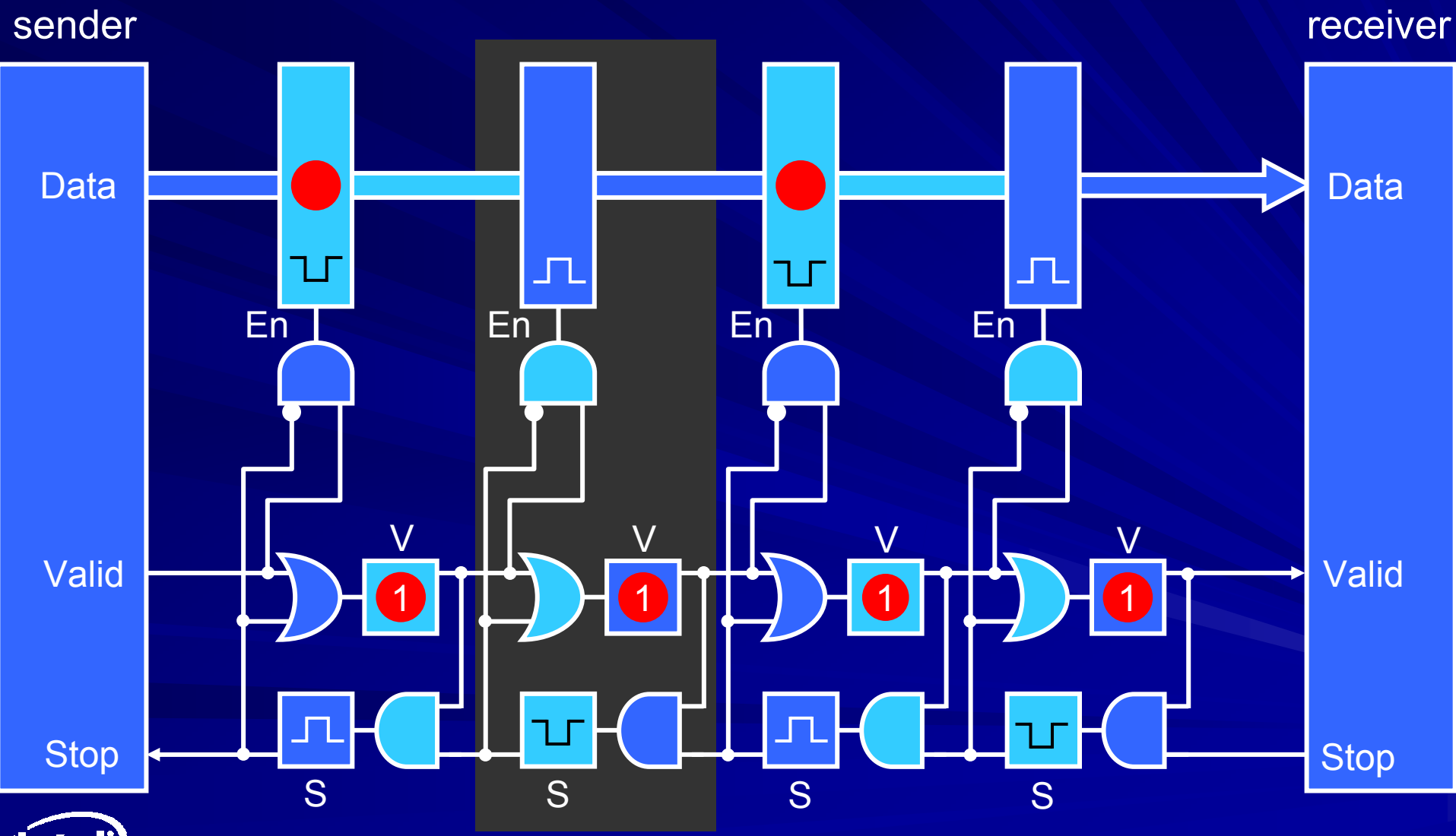
Master and slave latches with independent control

Shorthand notation (clock lines not shown)

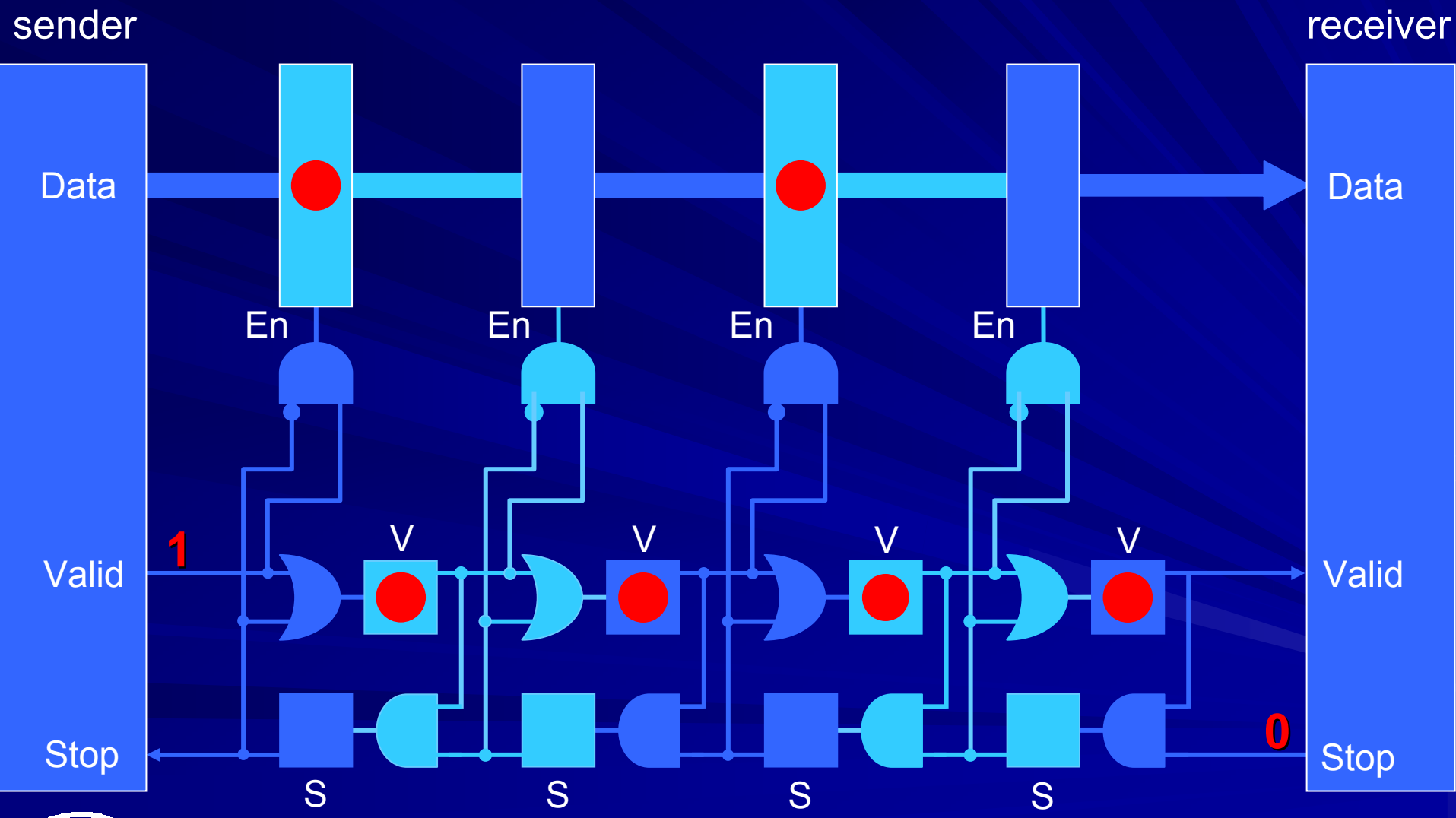




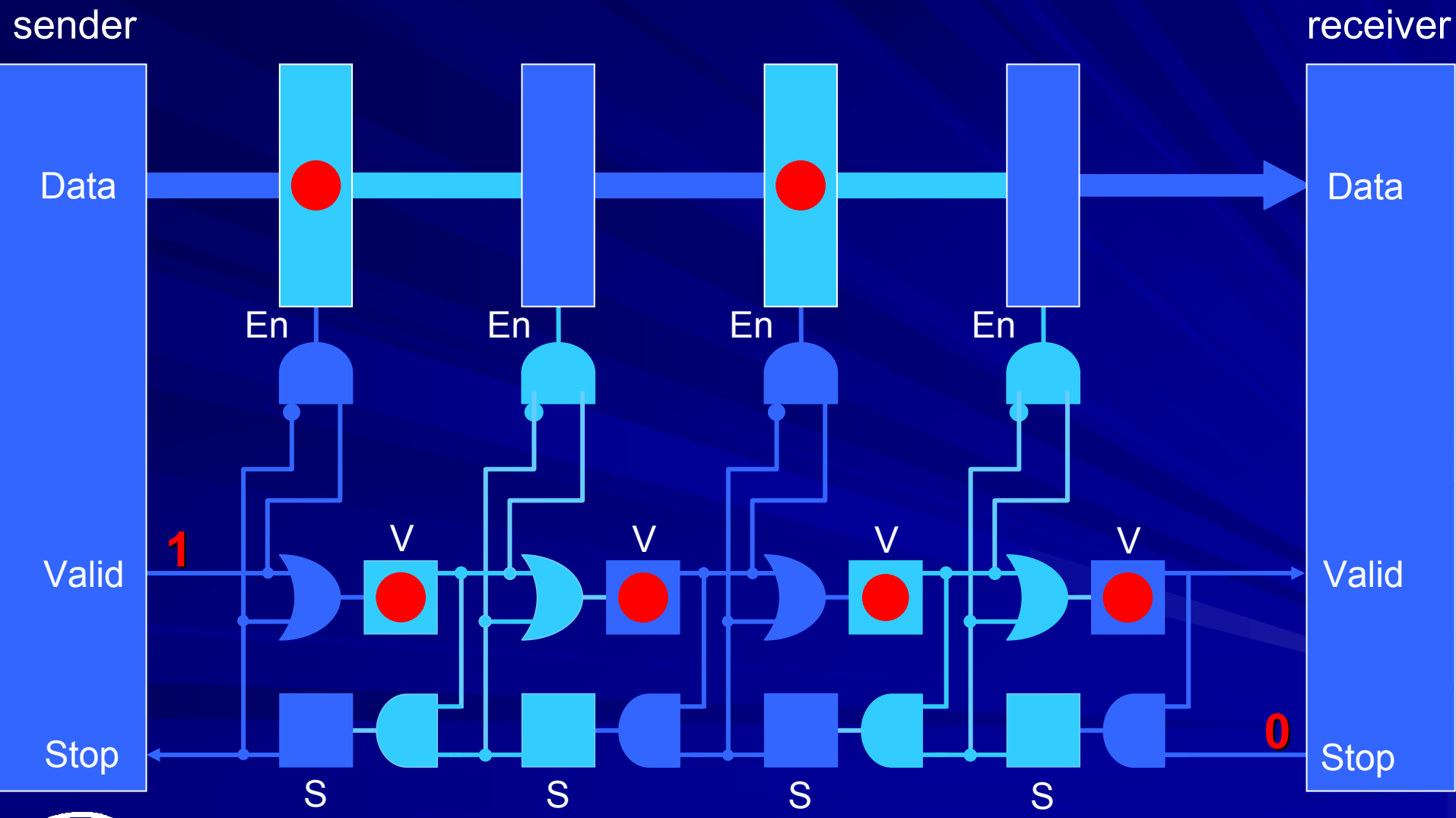
SELF (linear communication)



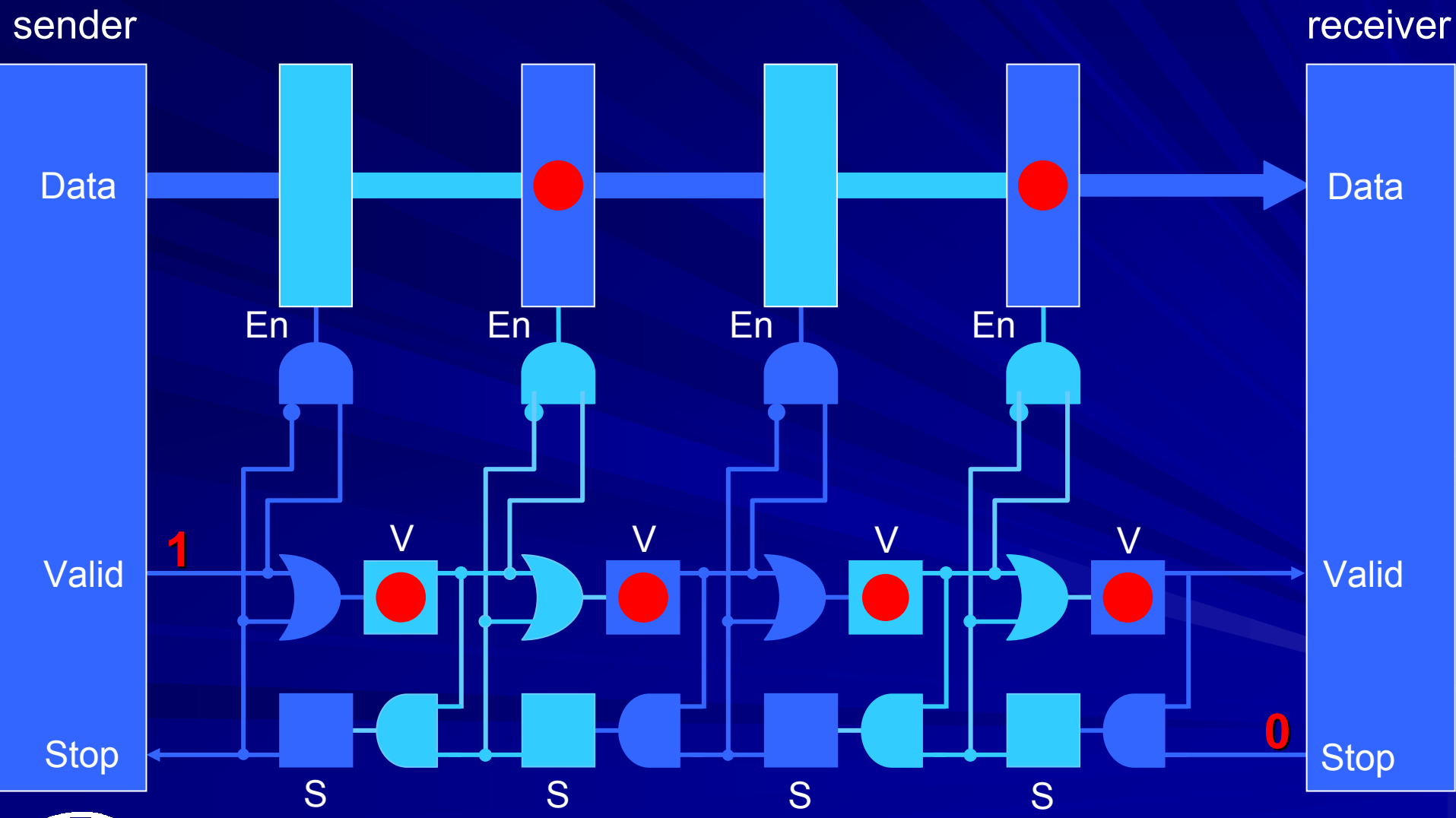
SELF


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

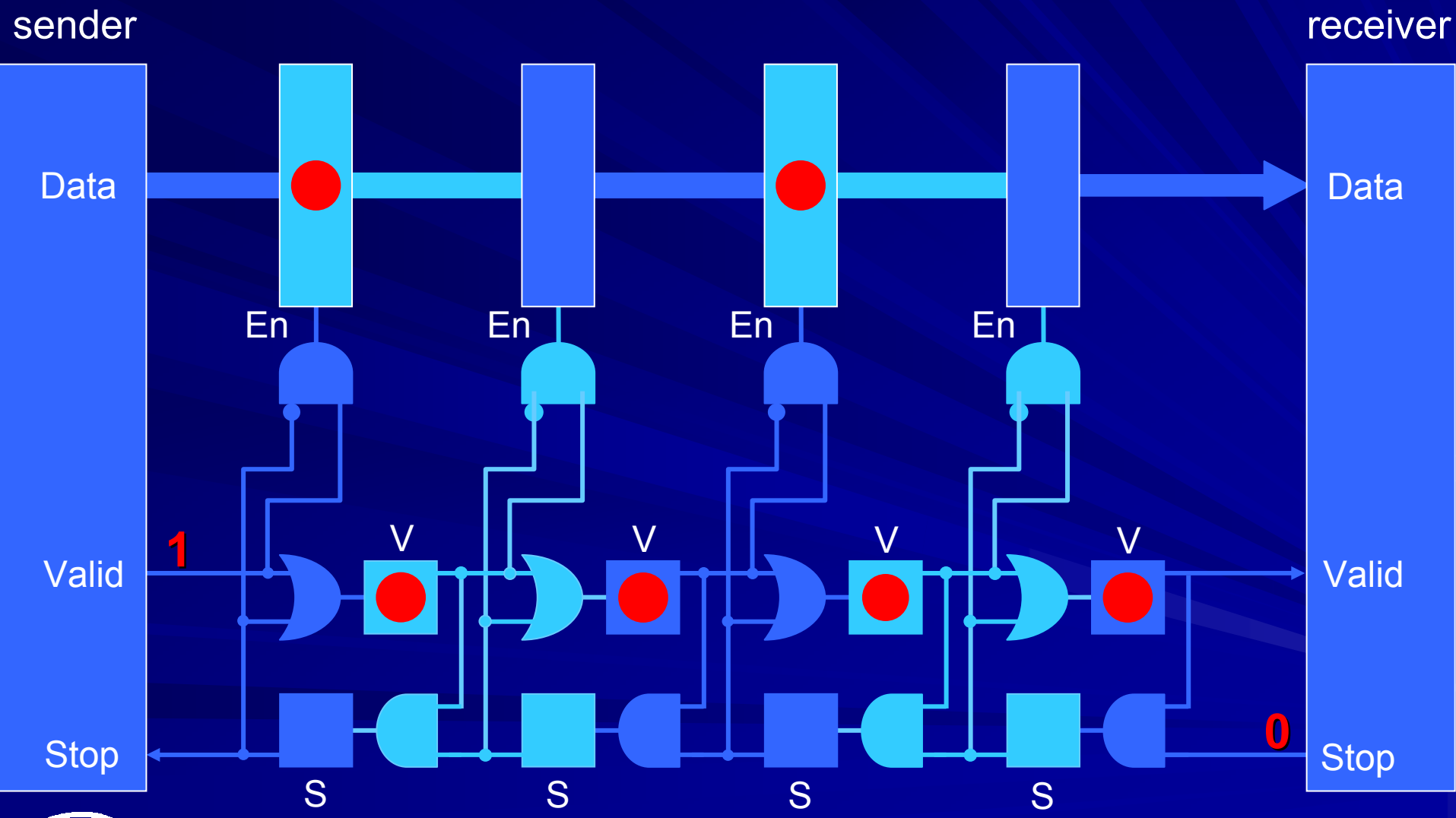
SELF


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

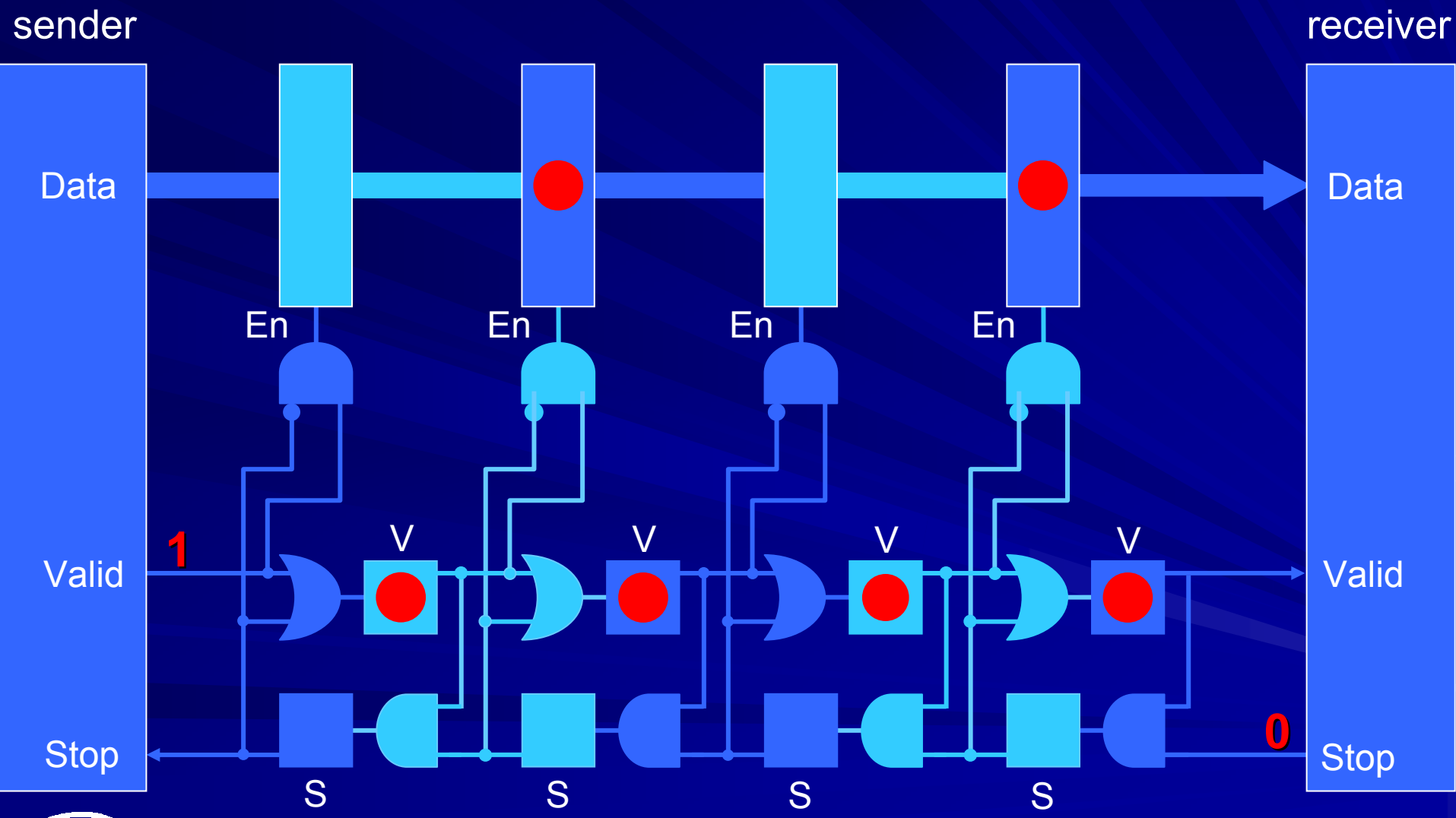
SELF


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

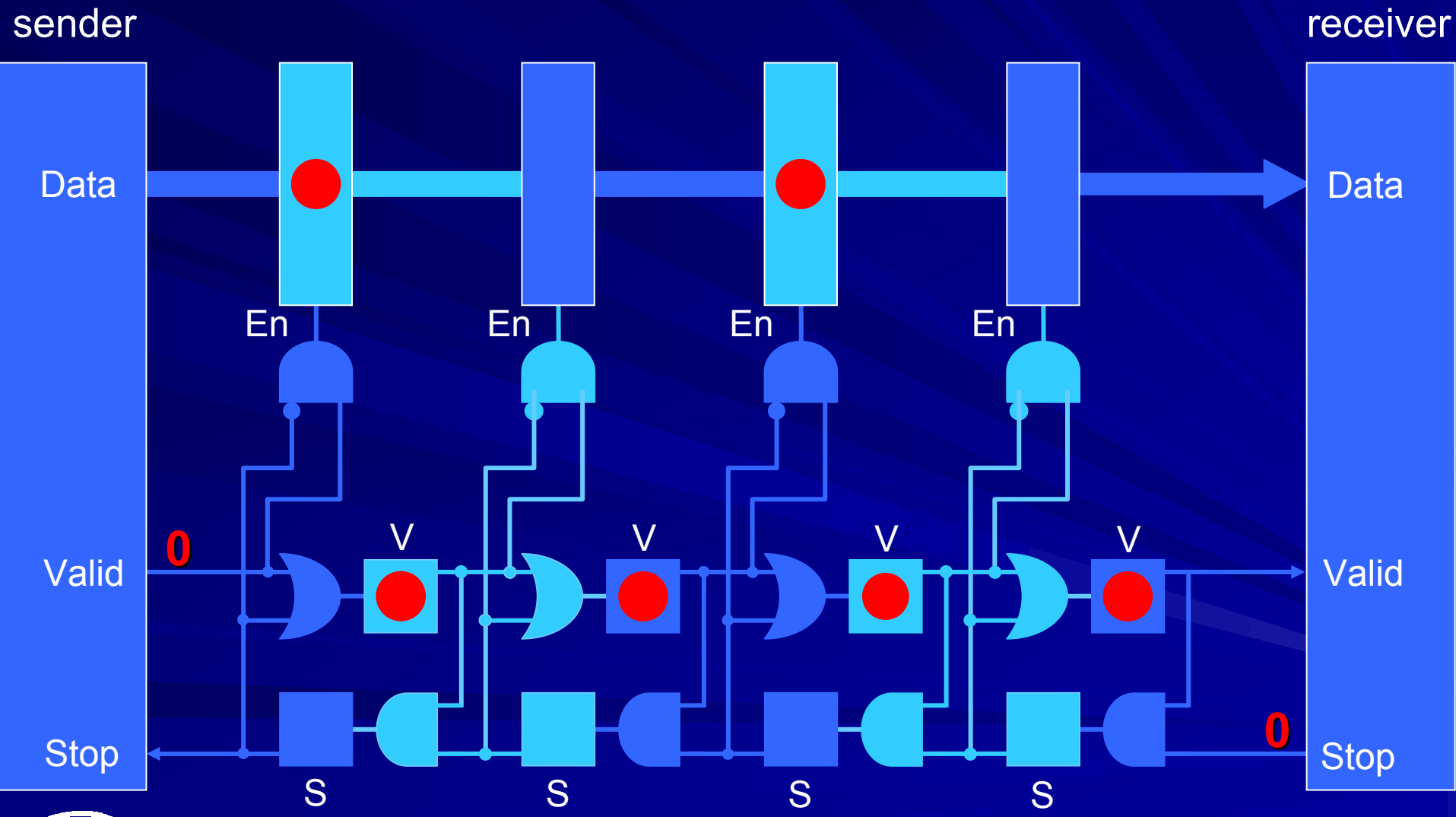
SELF


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

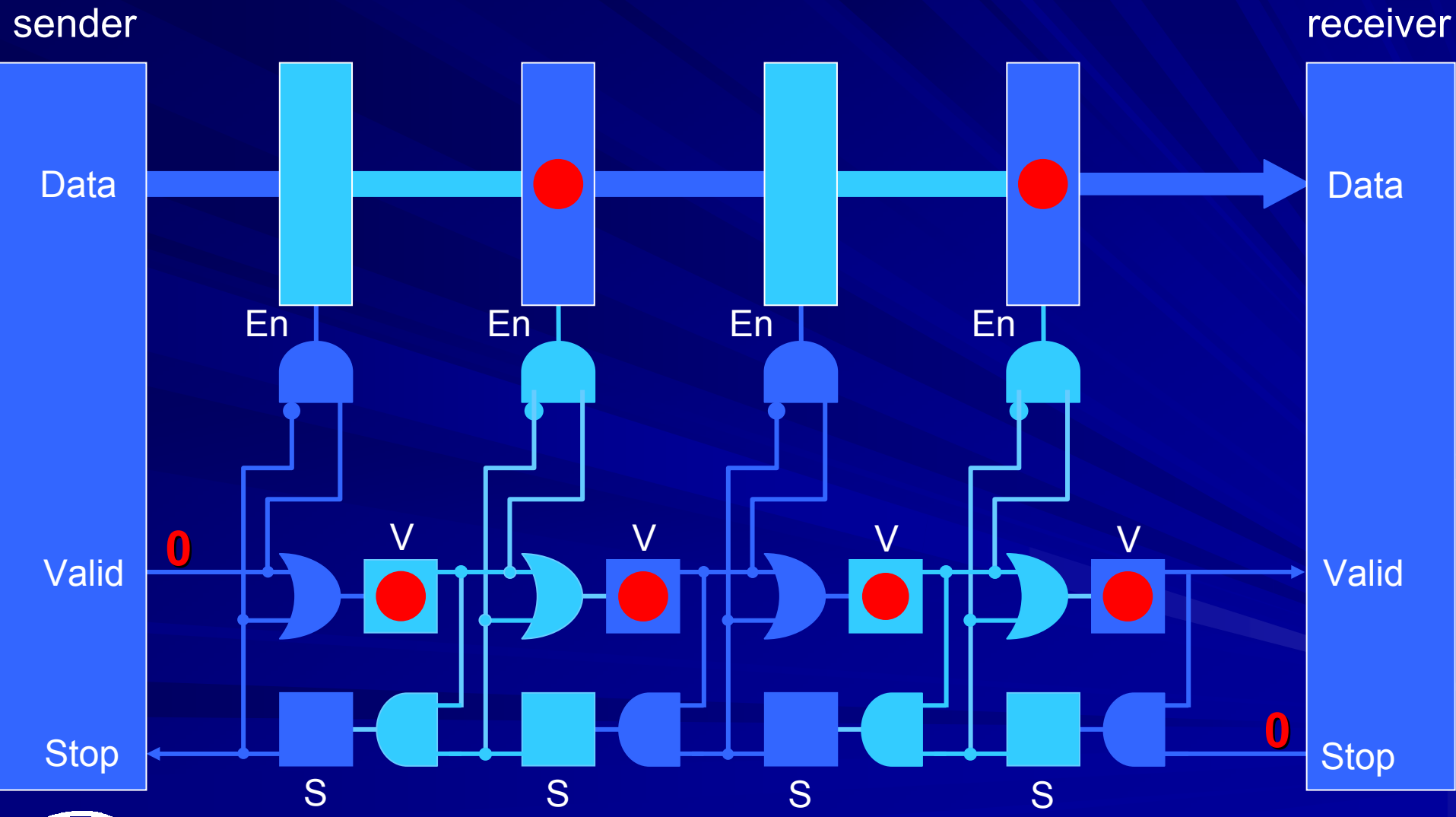
SELF


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

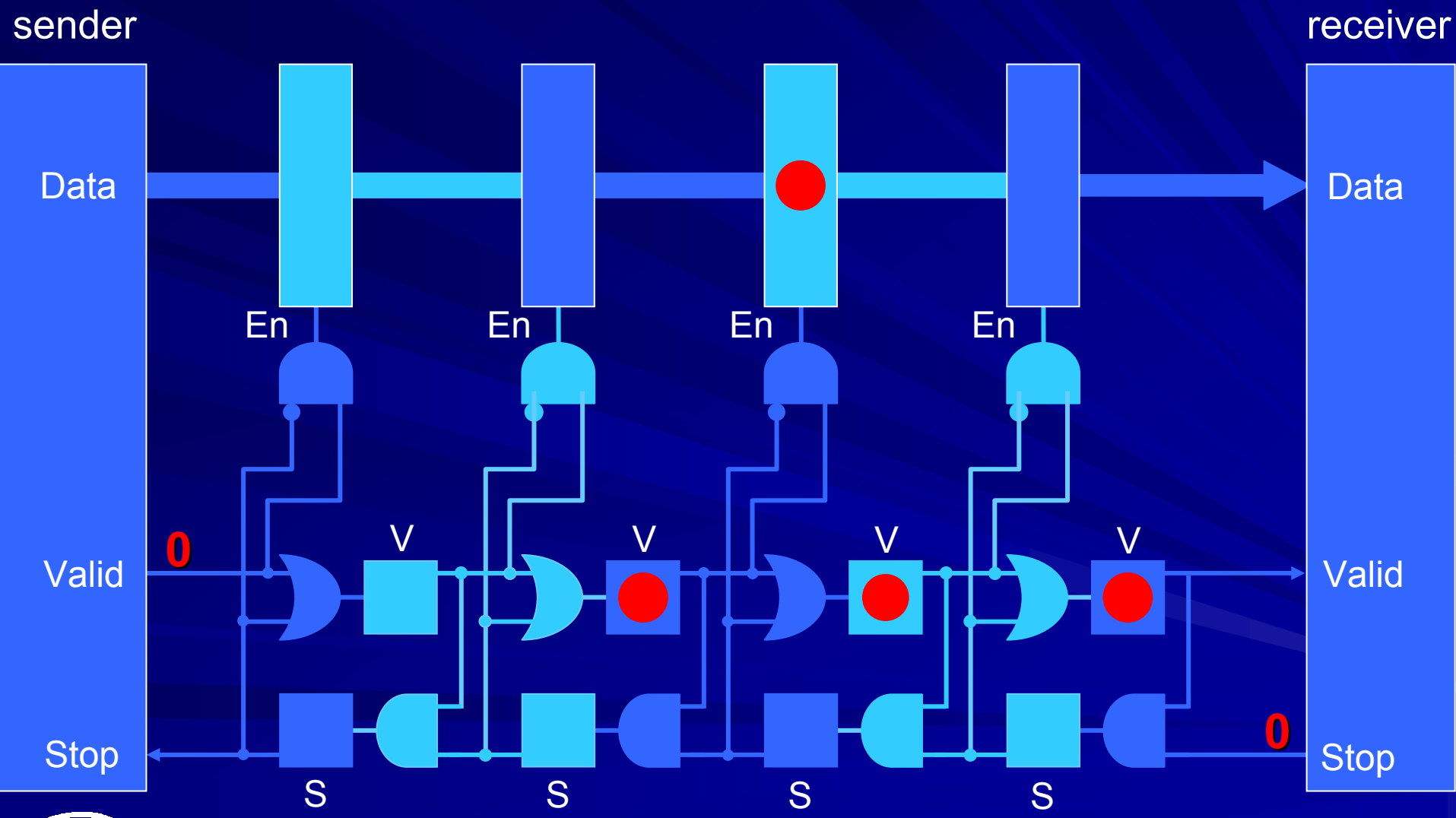
SELF


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

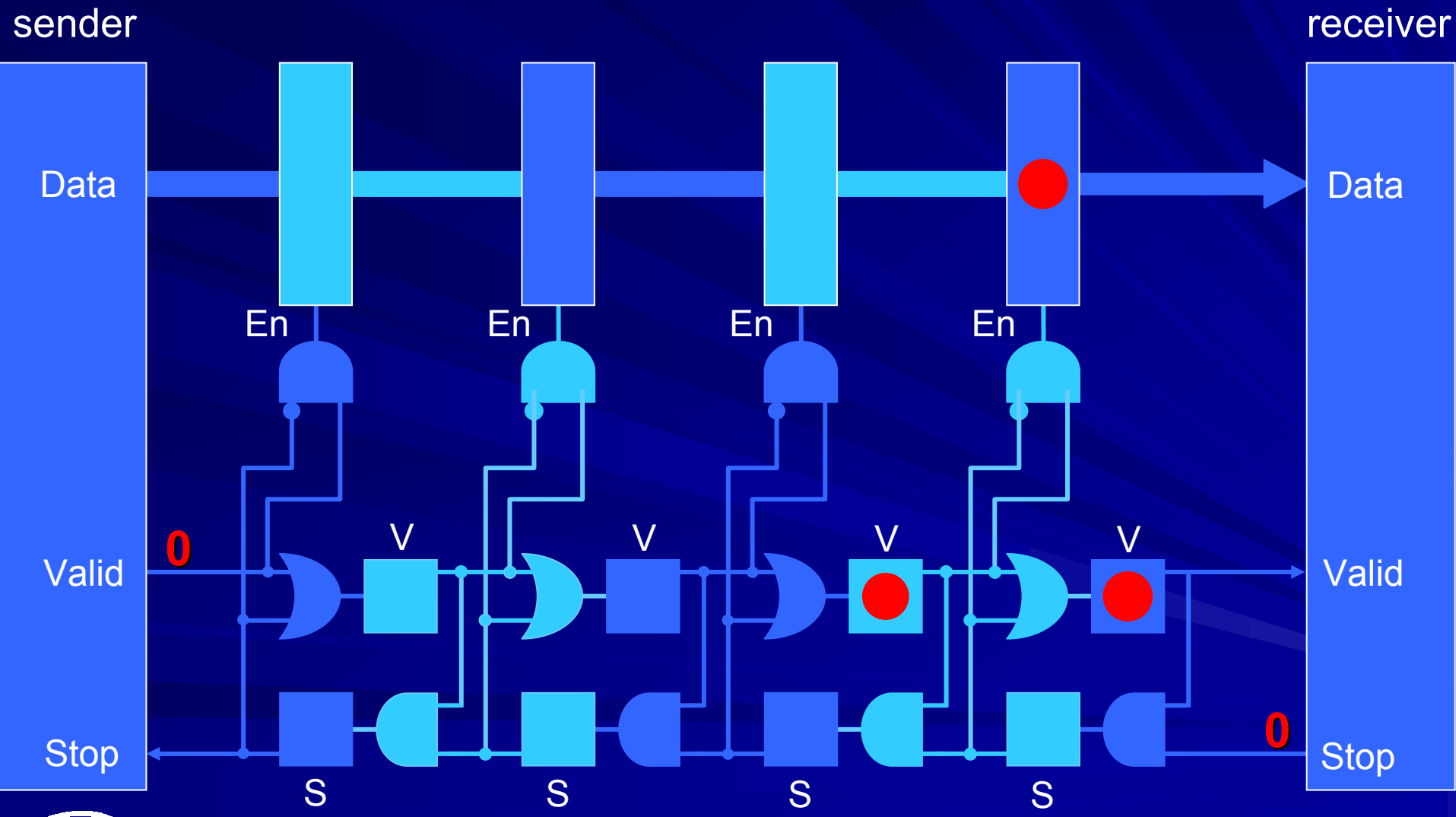
SELF


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

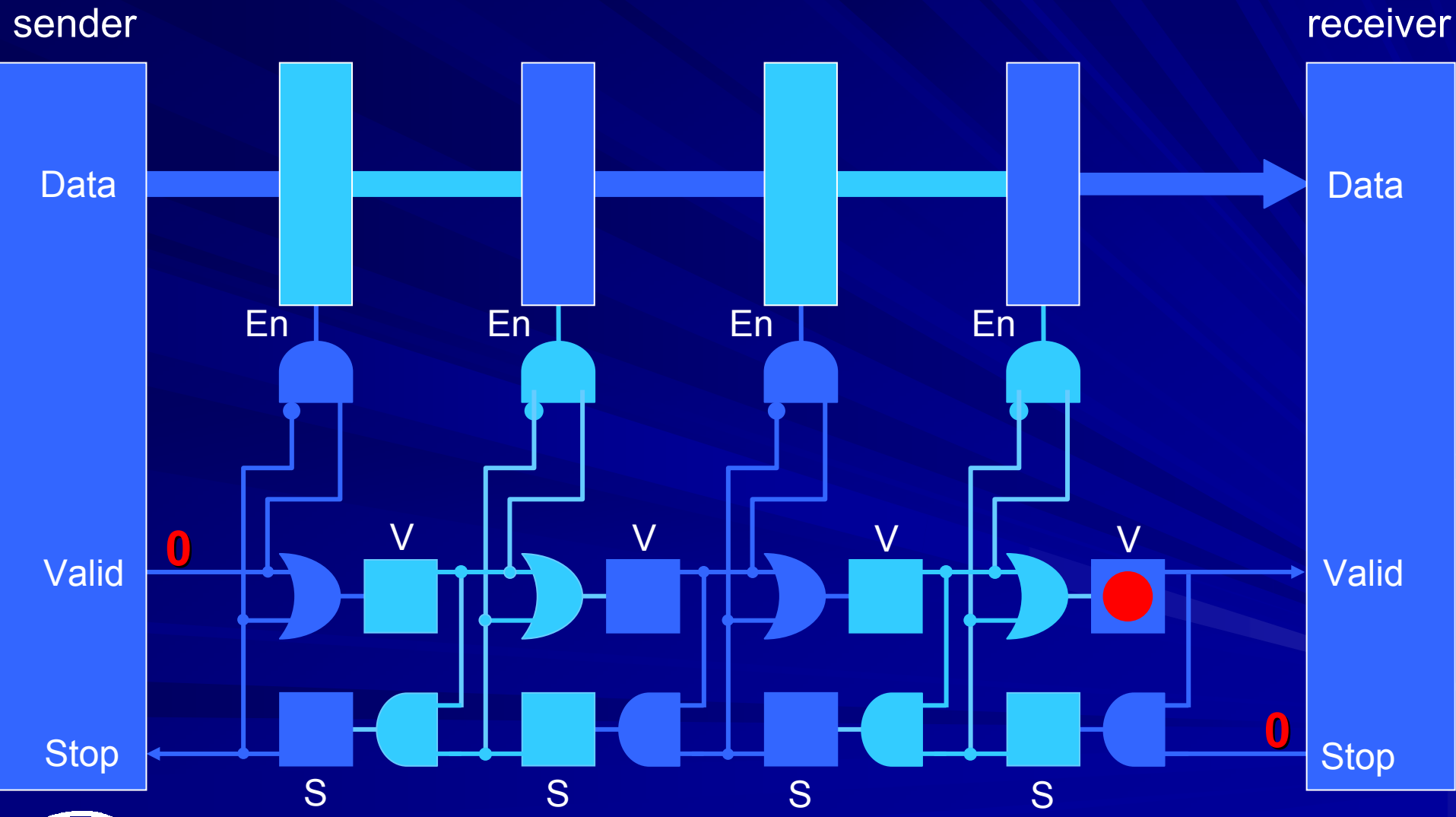
SELF


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

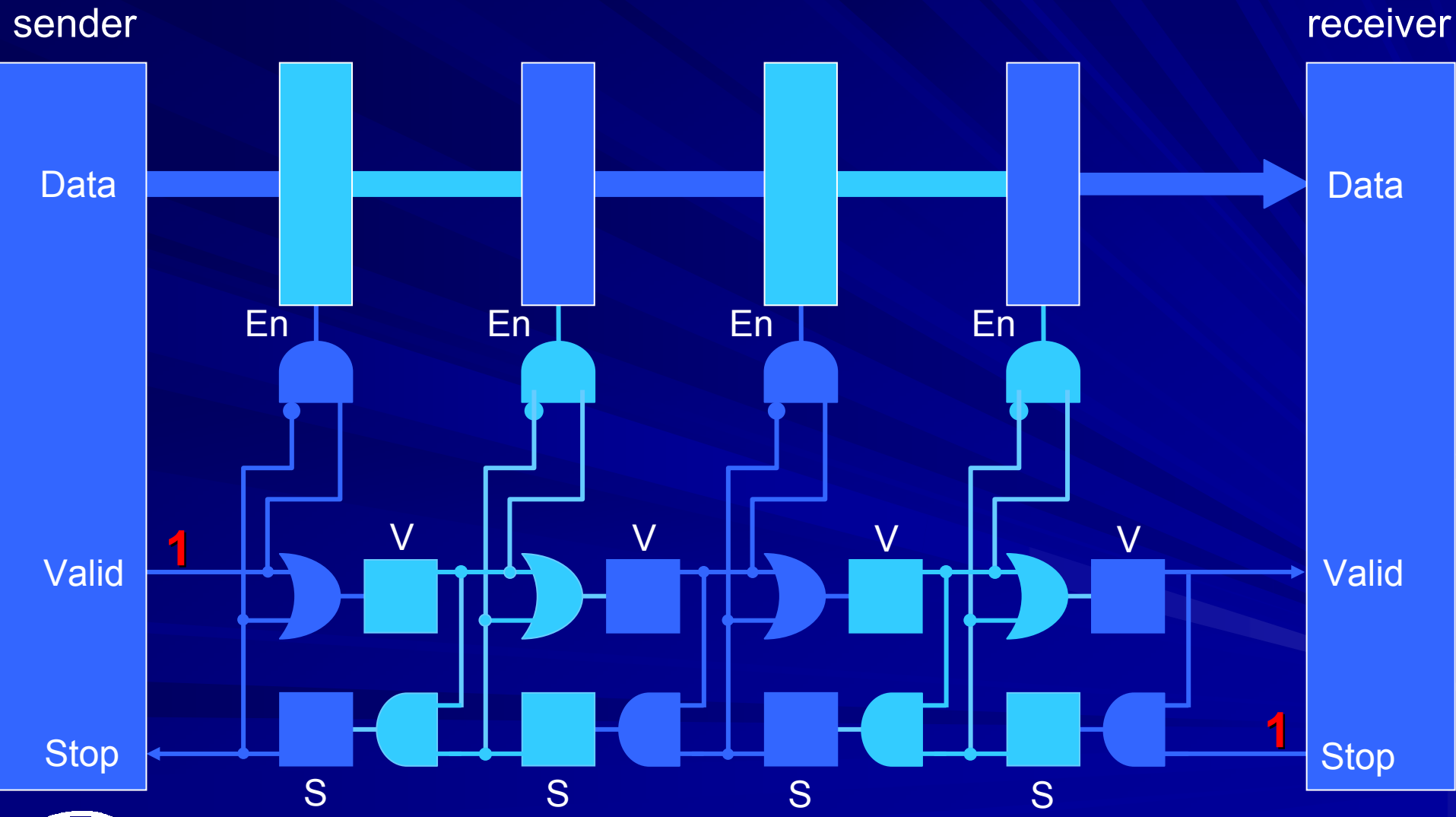
SELF


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

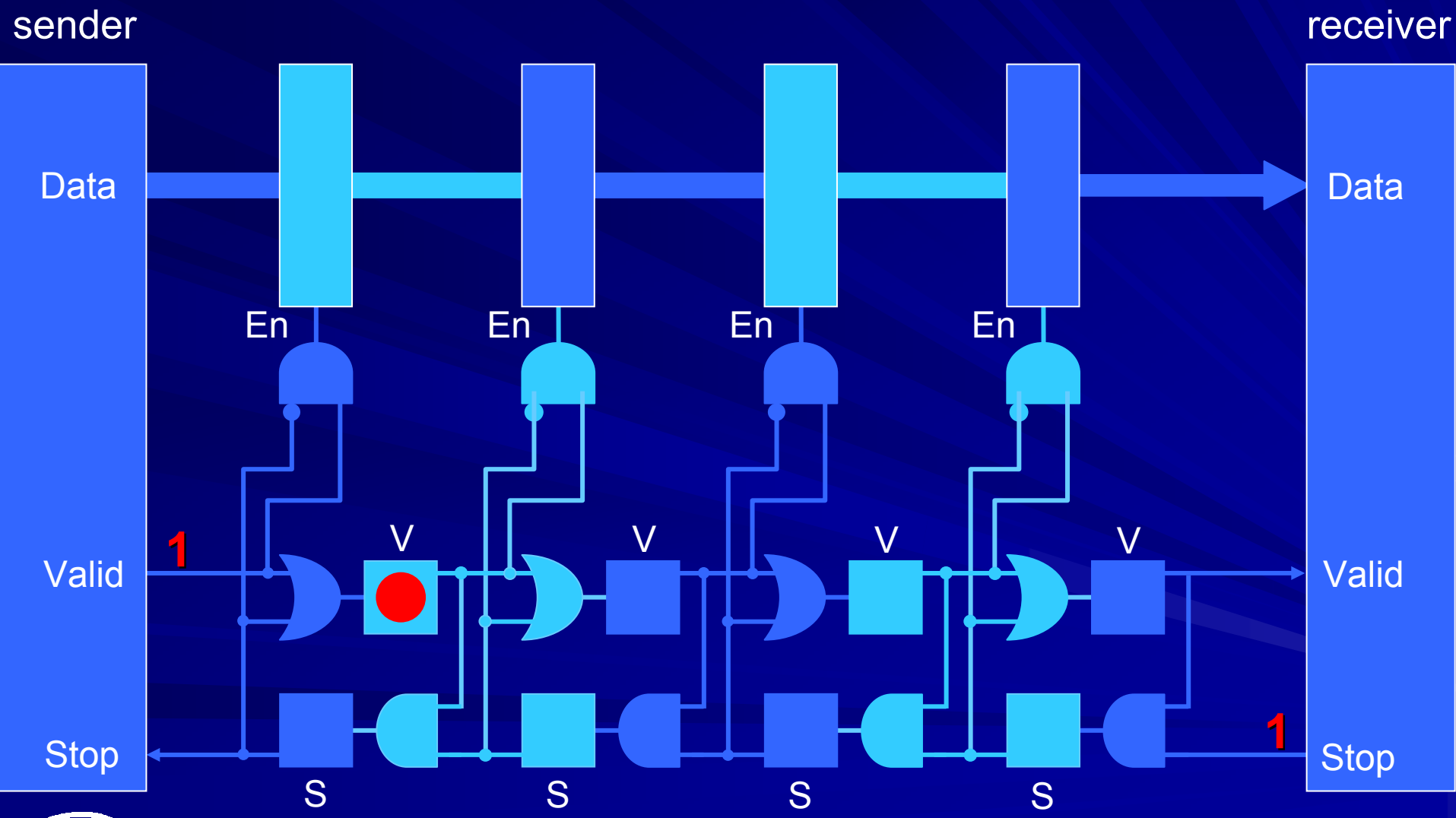
SELF


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

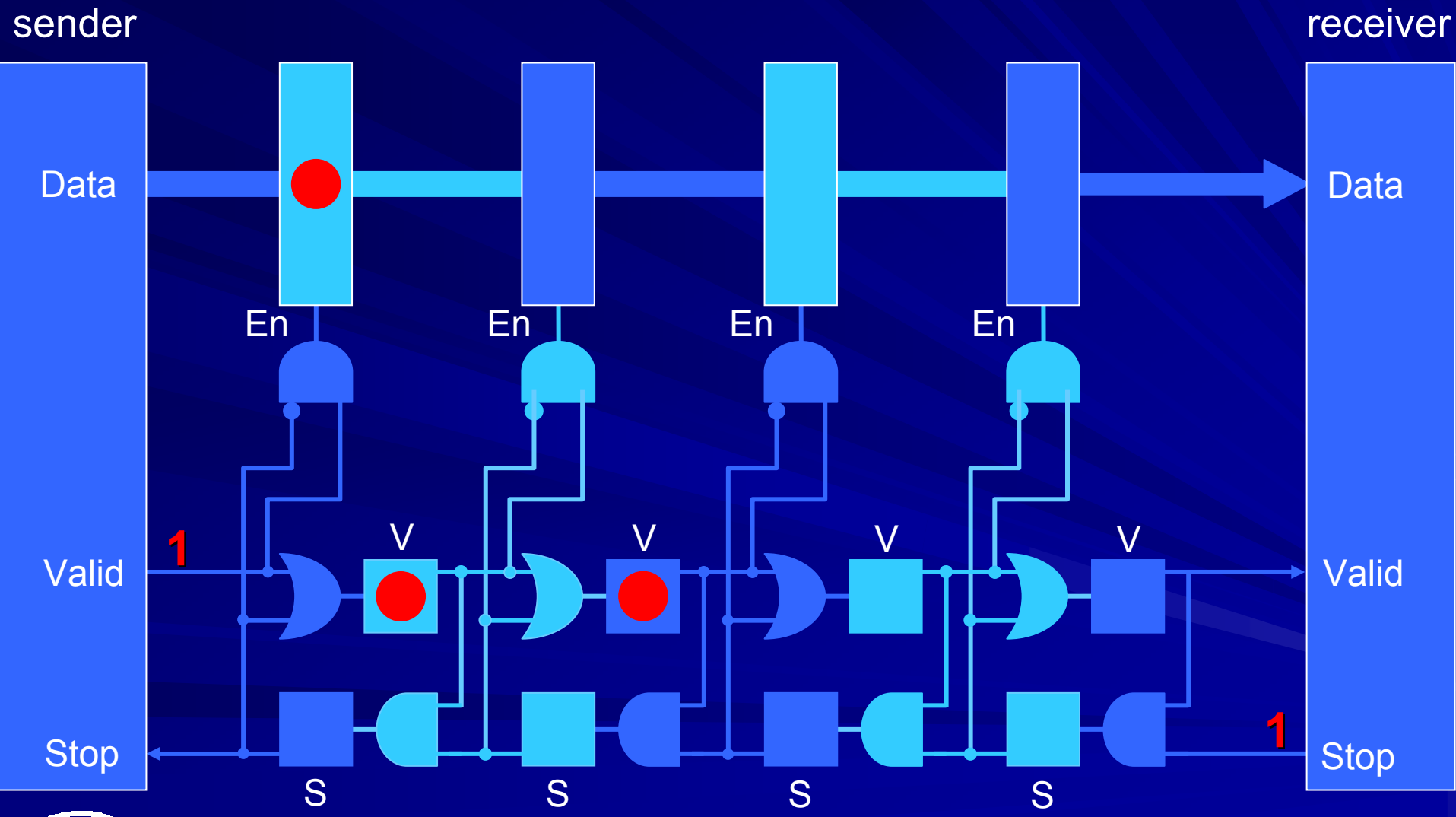
SELF


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

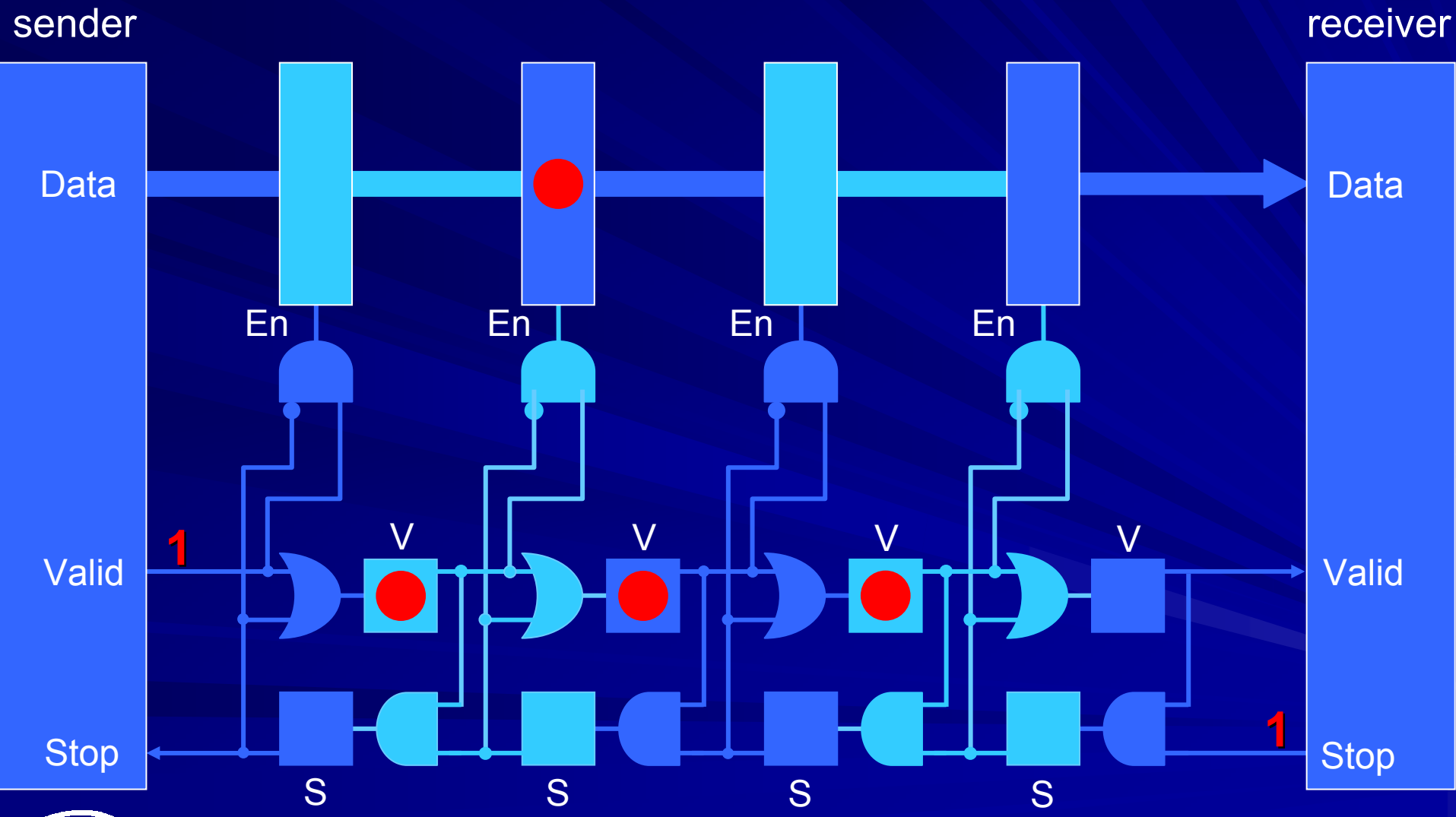
SELF


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

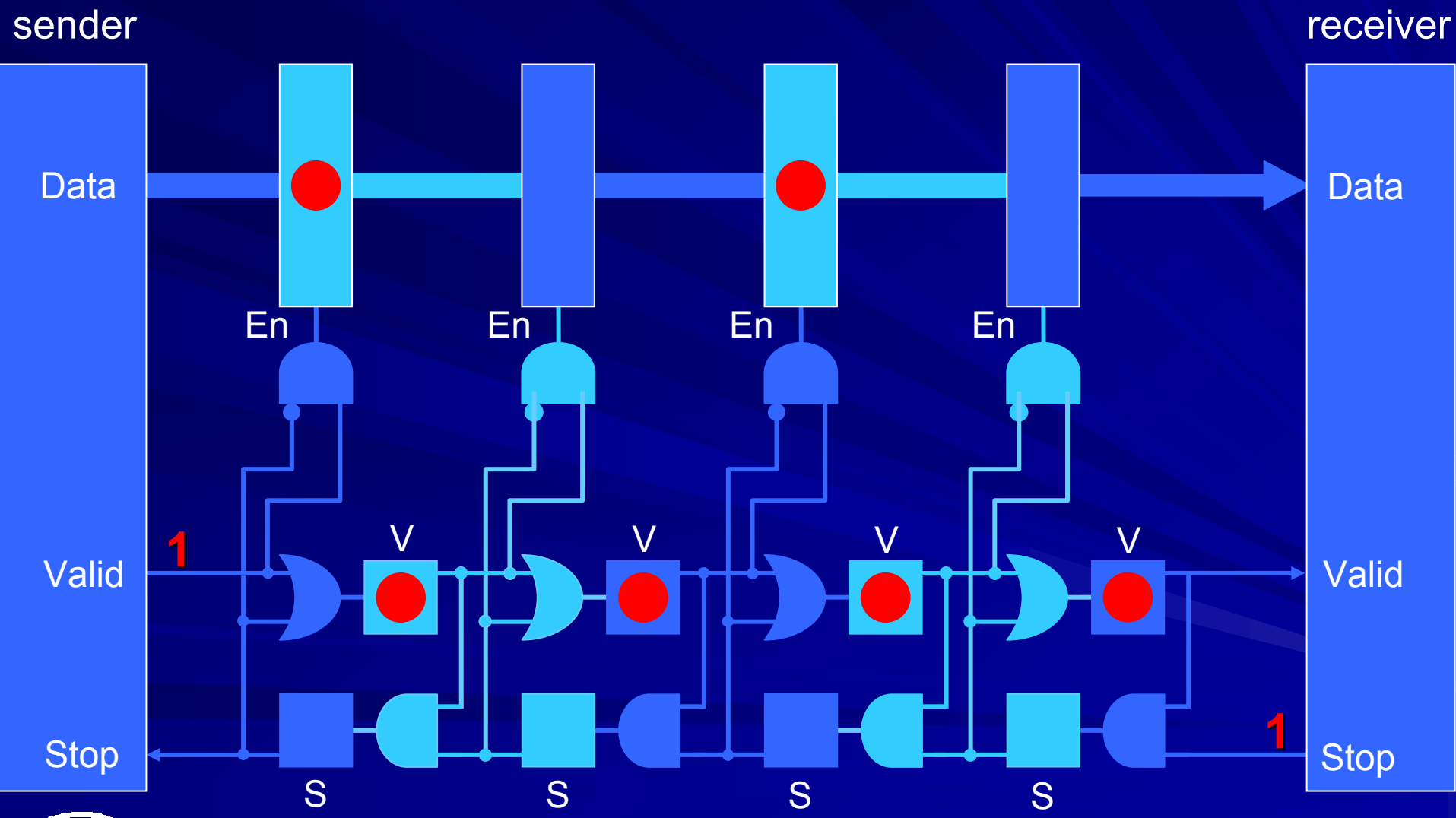
SELF


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

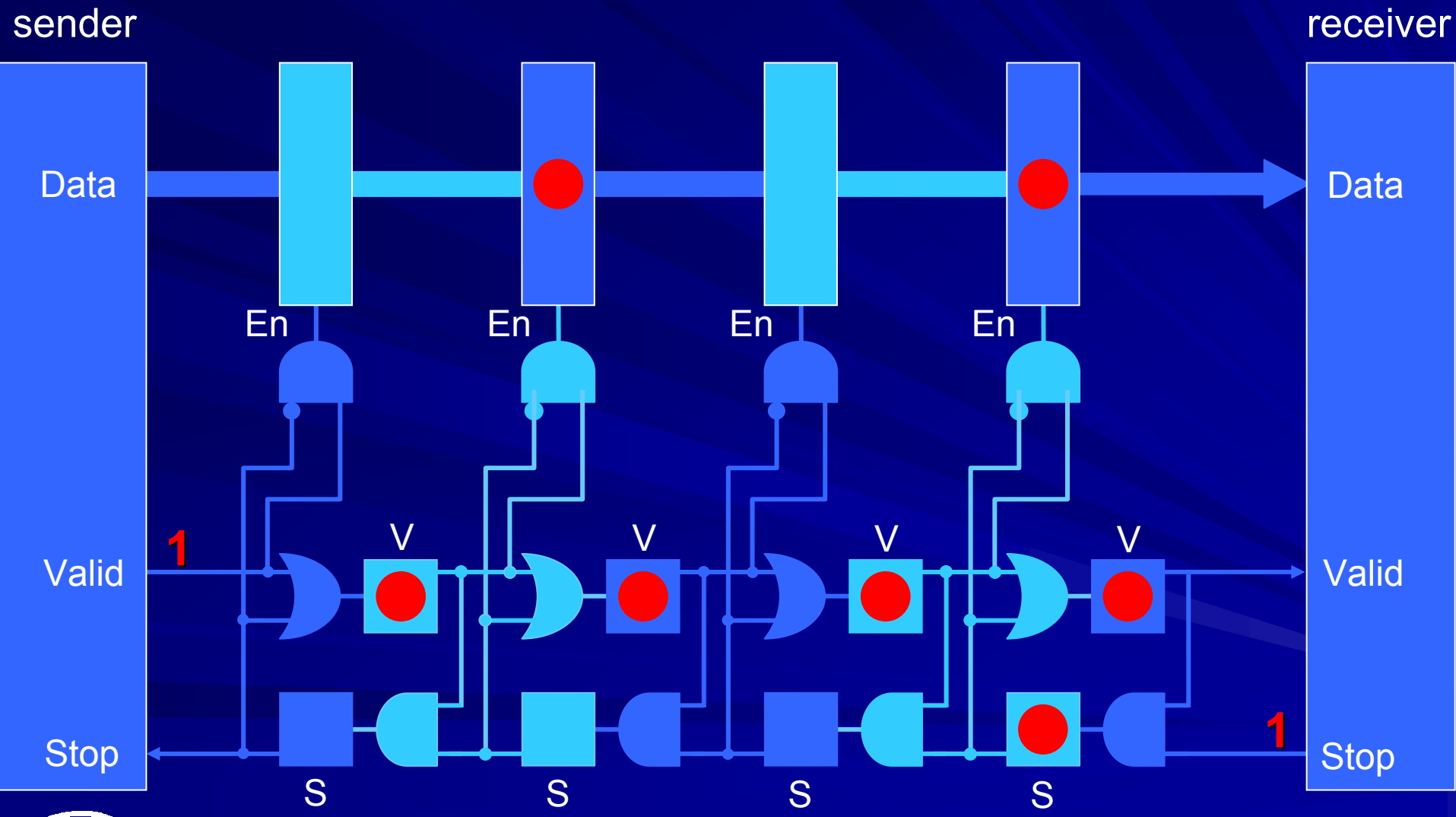
SELF


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

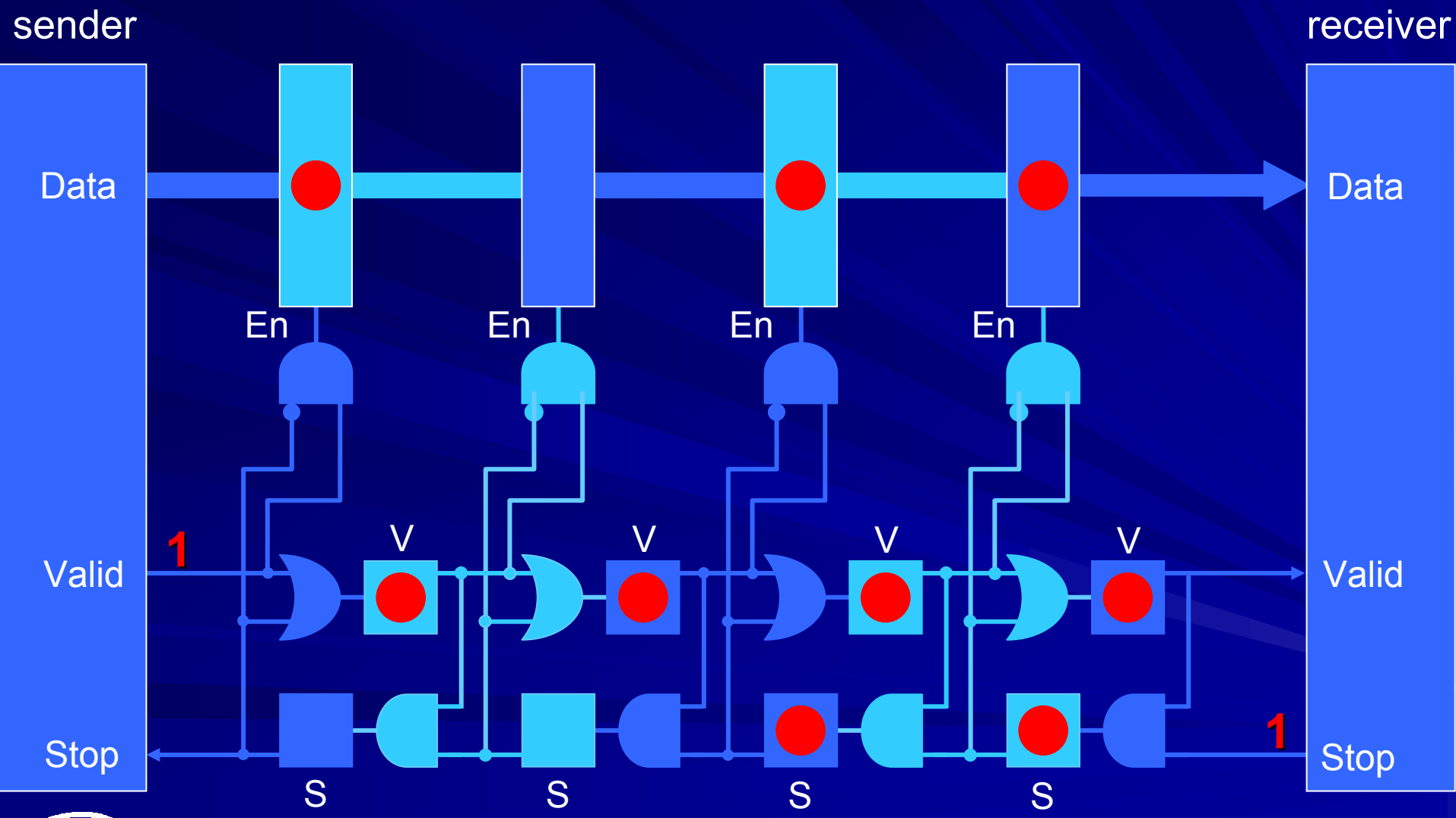
SELF


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

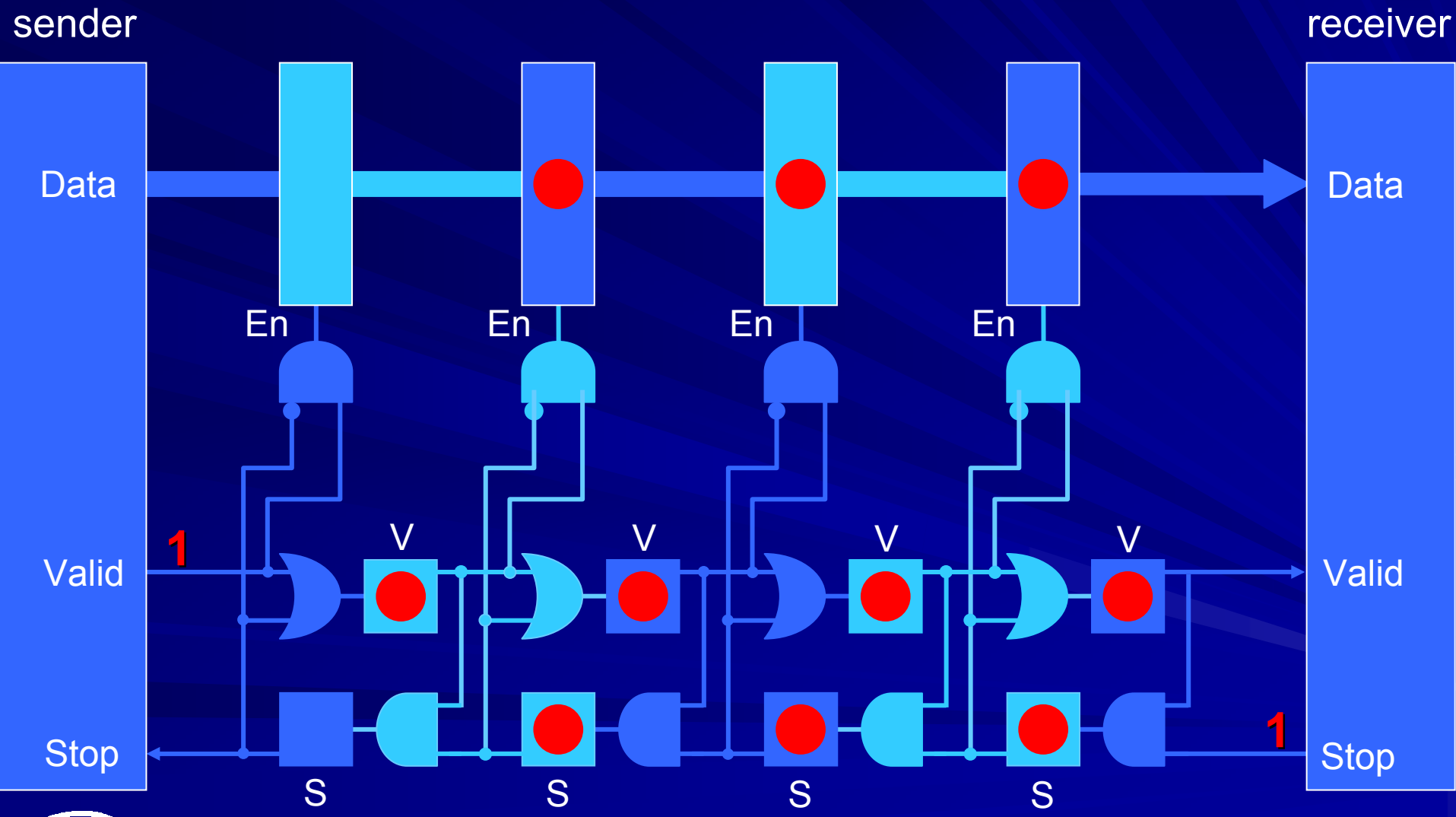
SELF


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

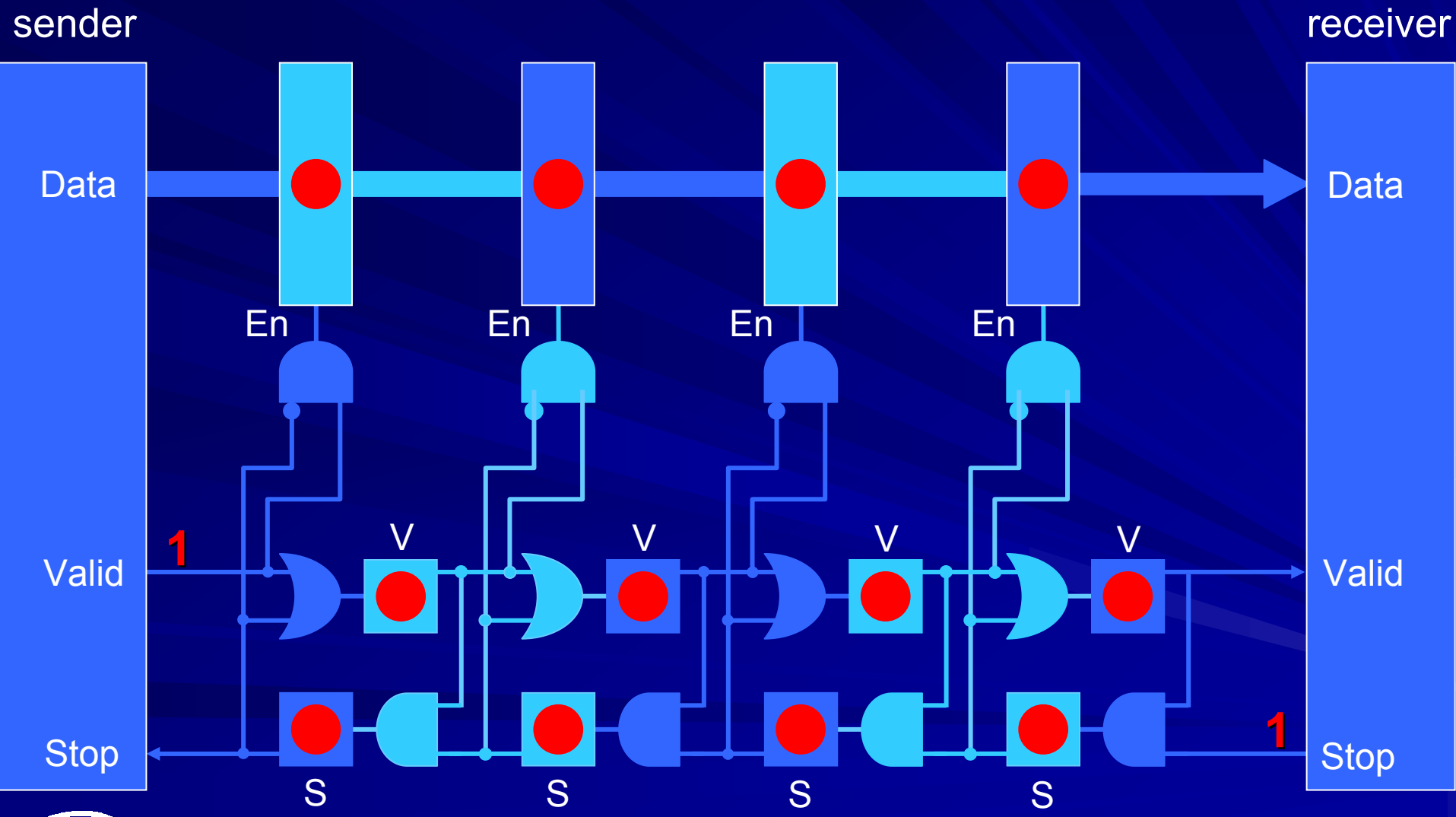
SELF


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

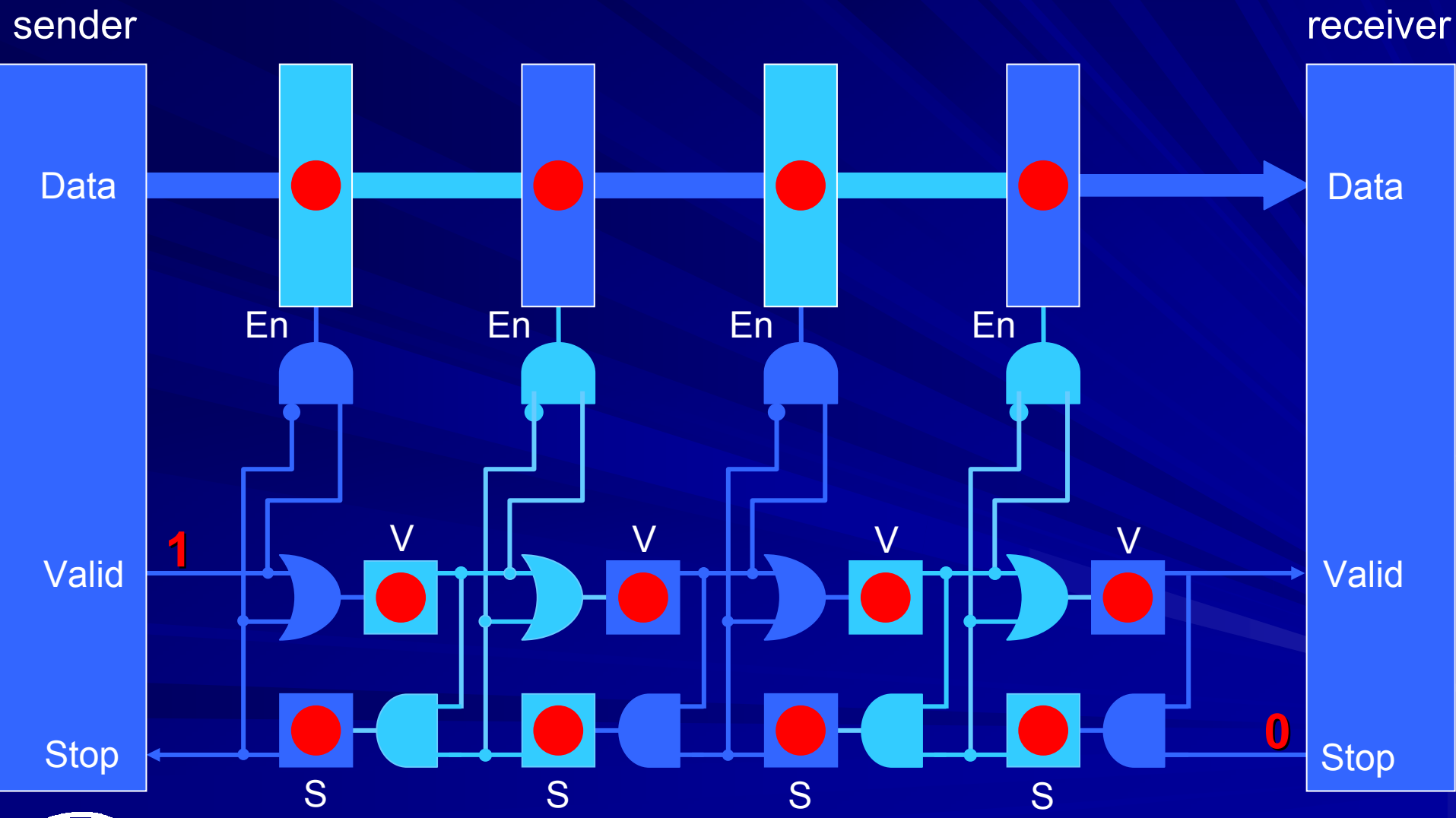
SELF


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

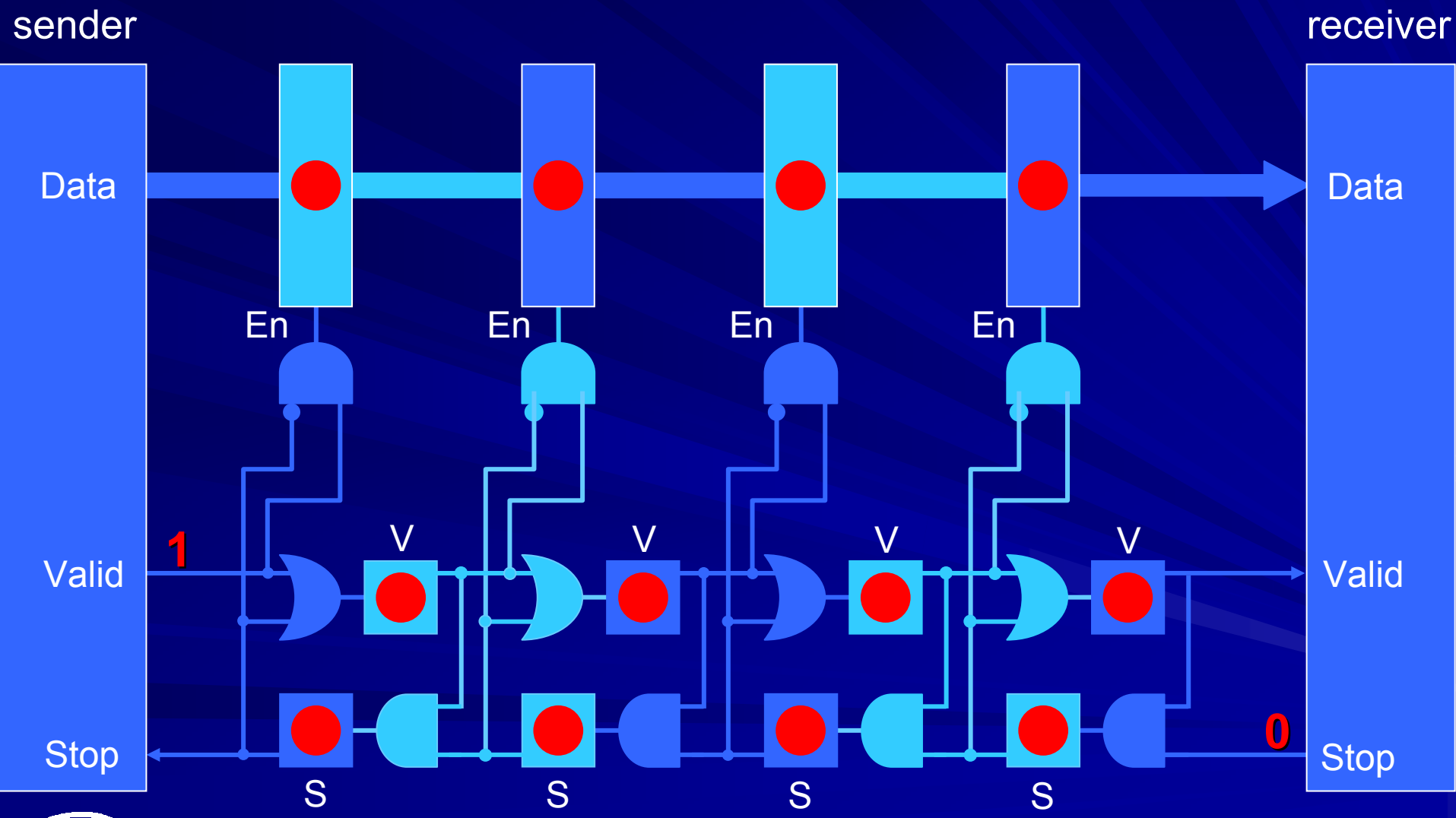
SELF


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

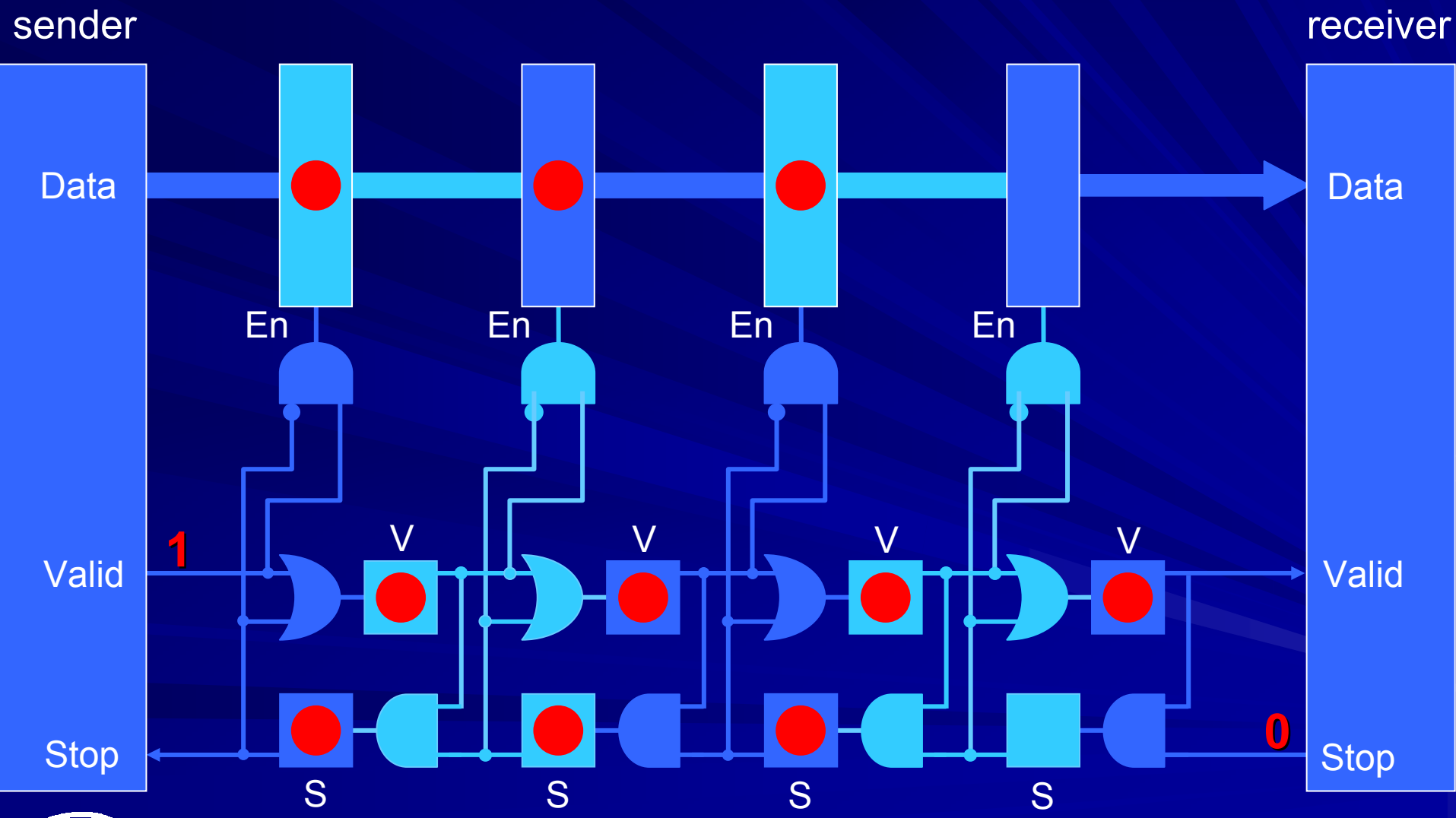
SELF


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

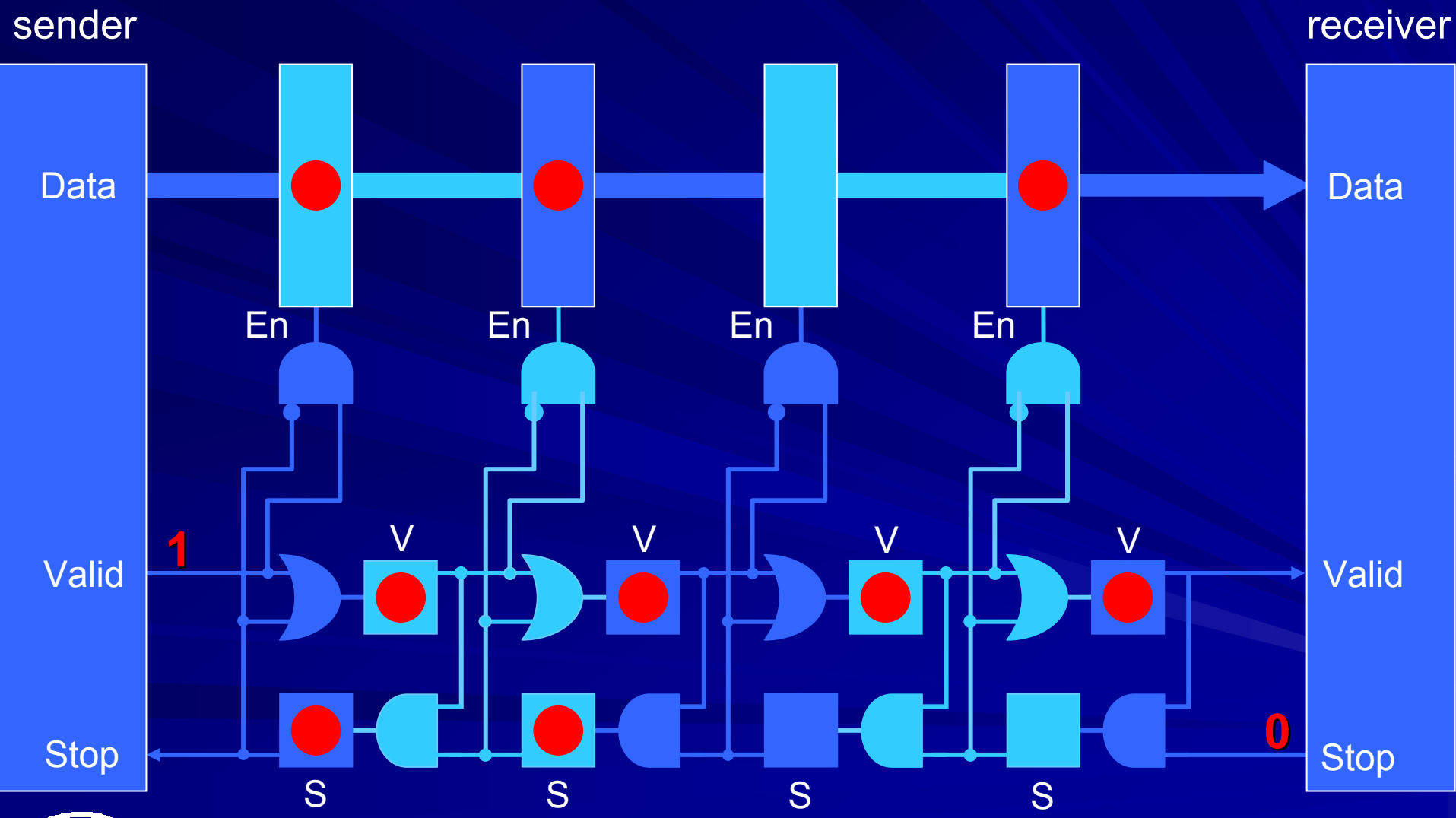
SELF


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

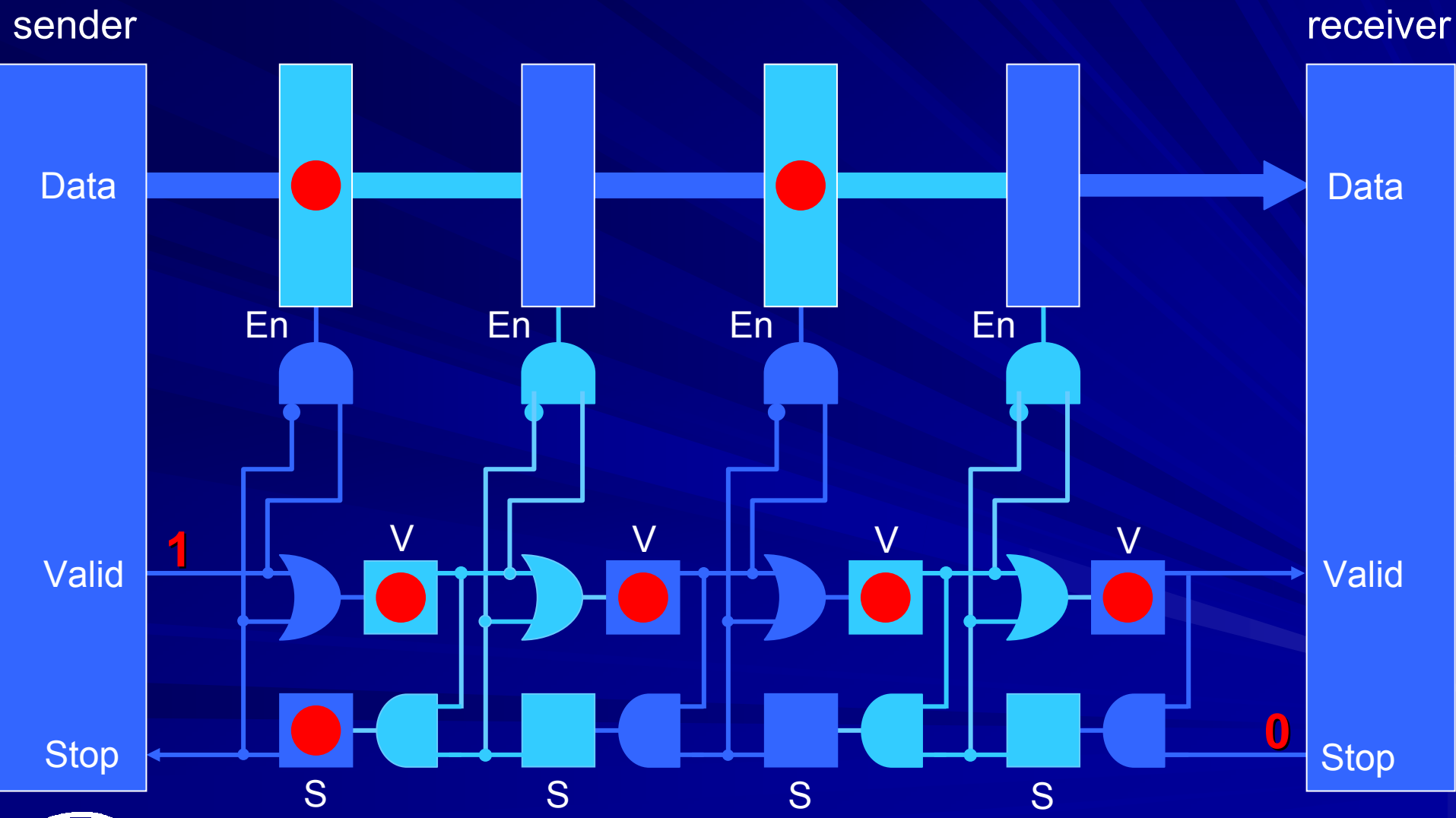
SELF


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

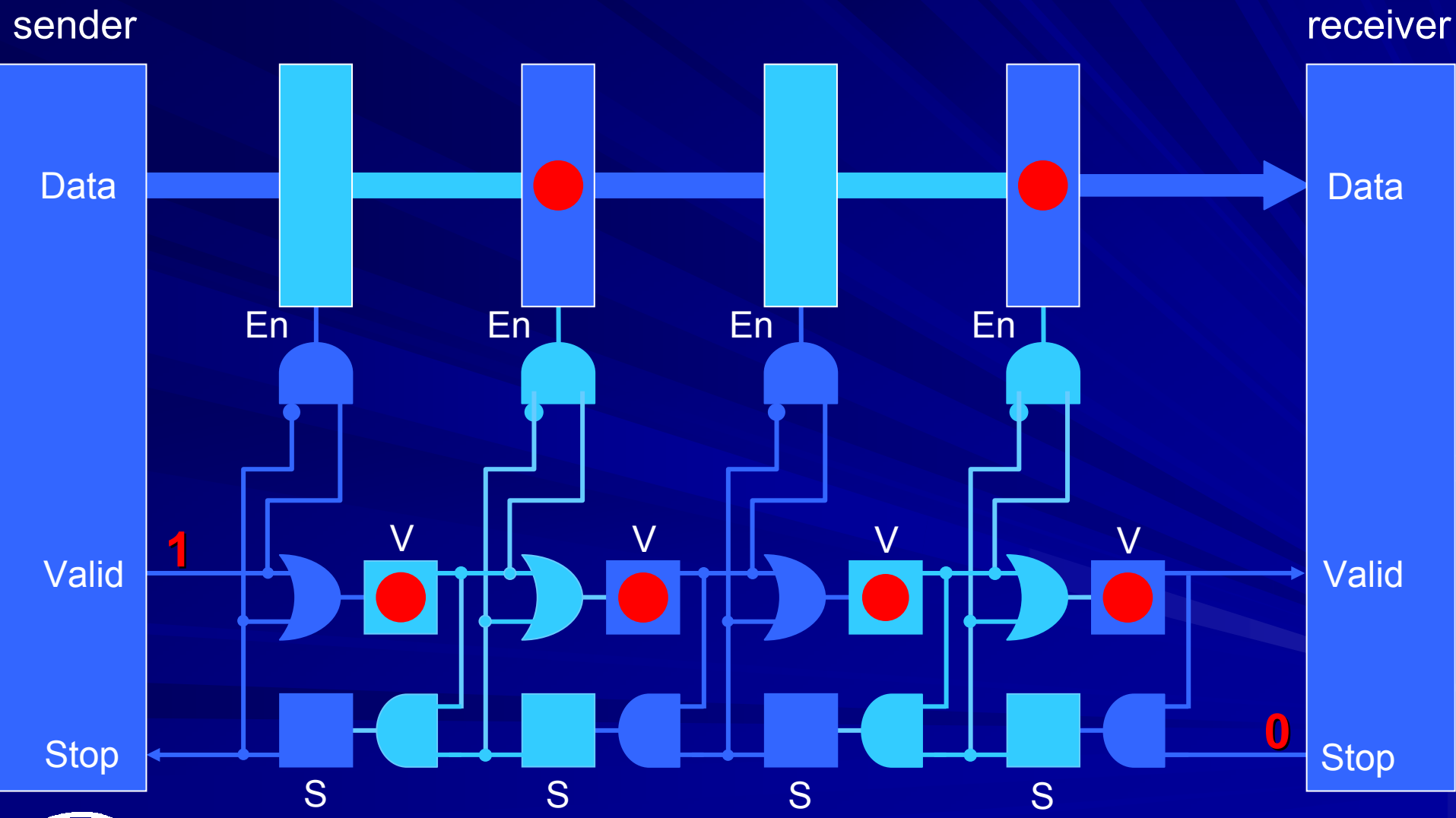
SELF


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

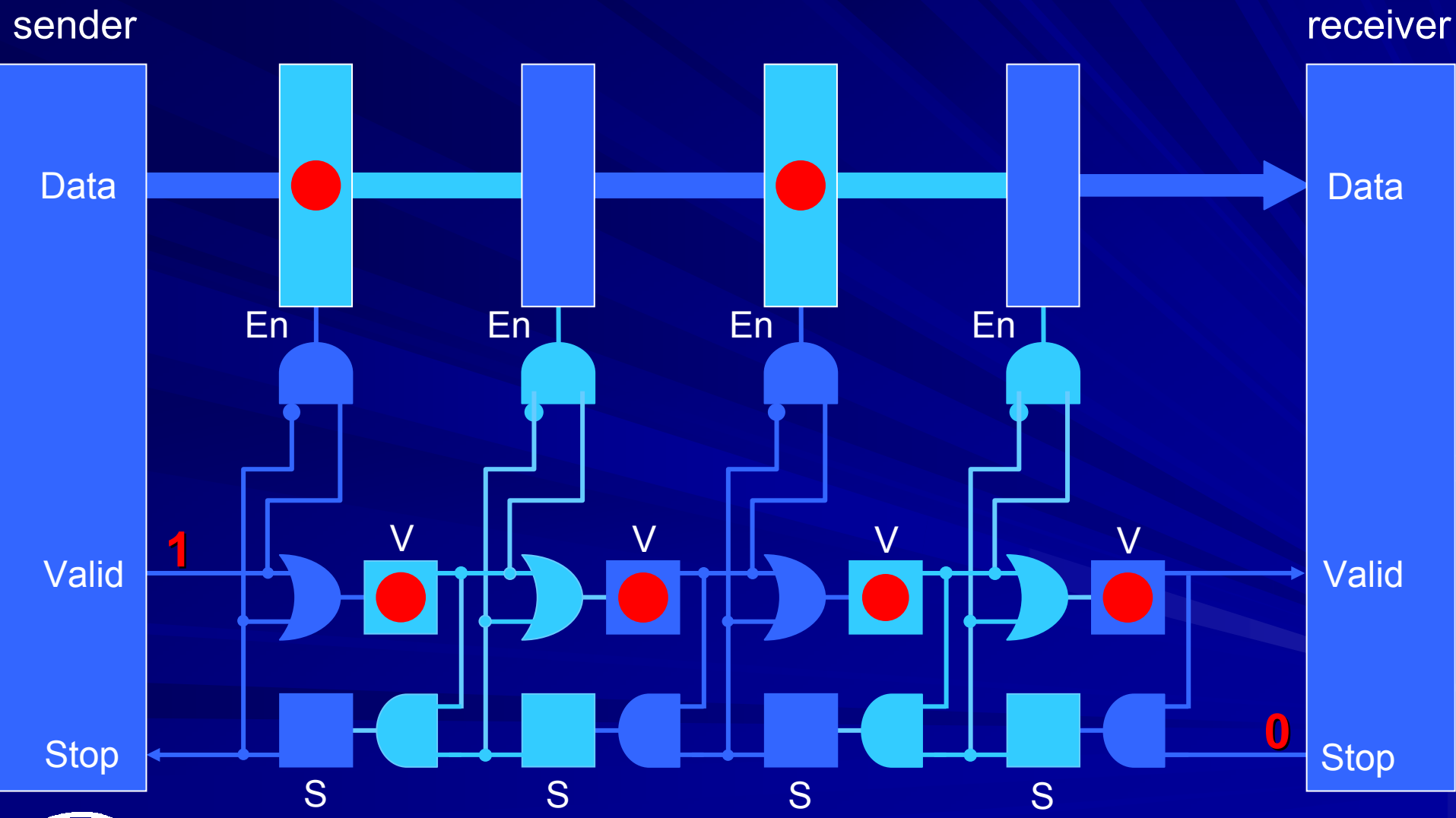
SELF


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

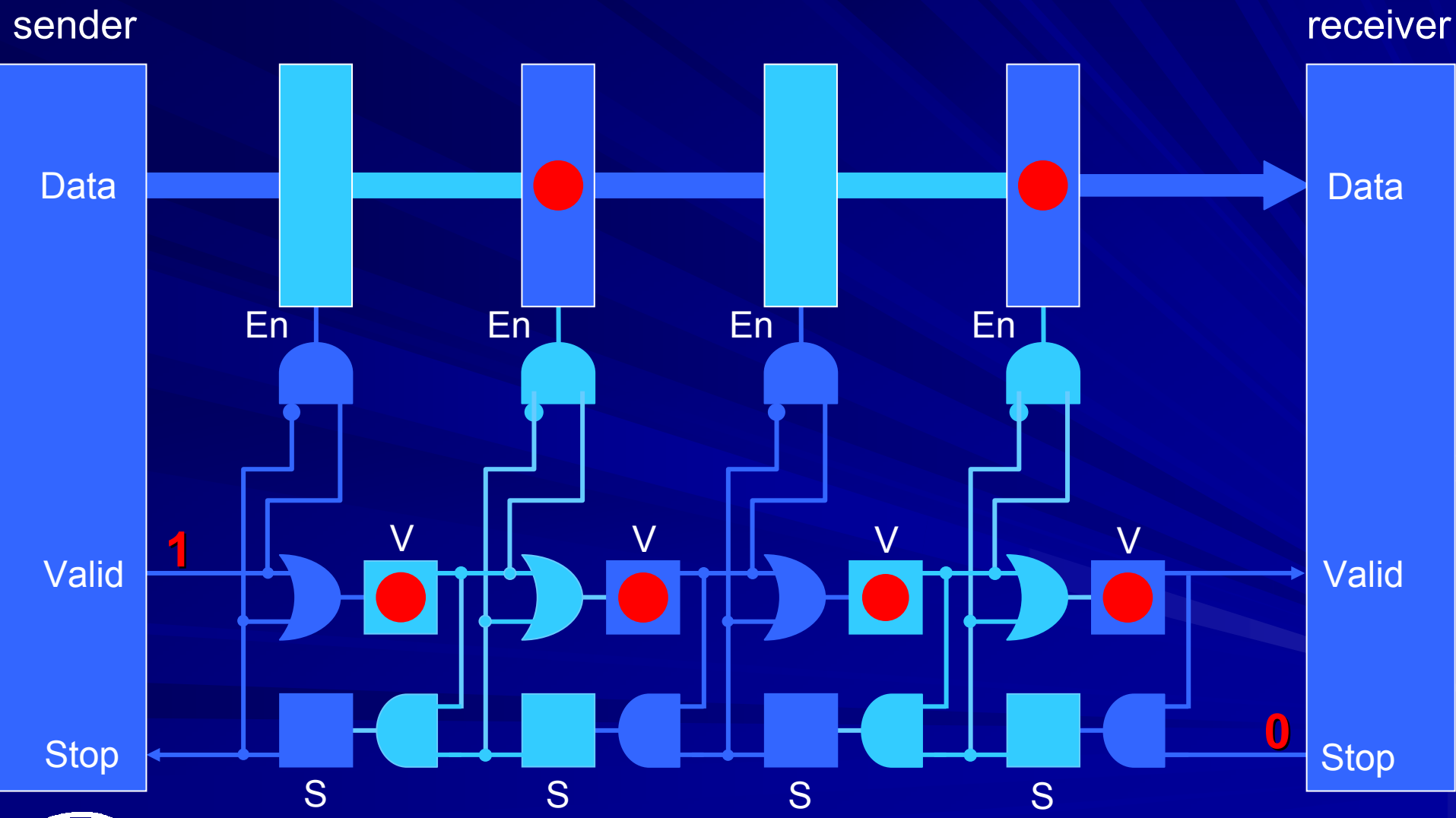
SELF


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

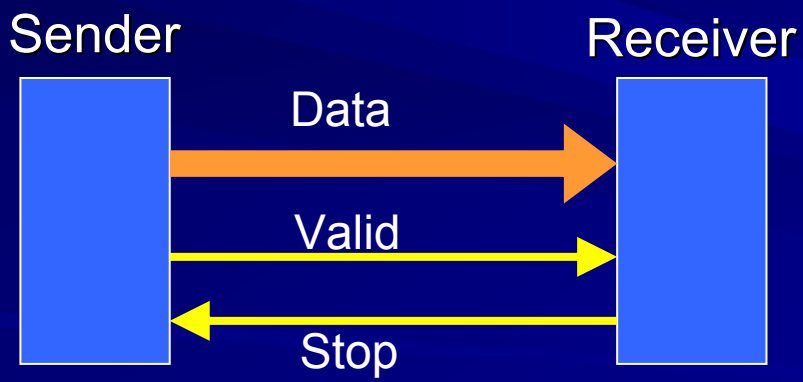
SELF

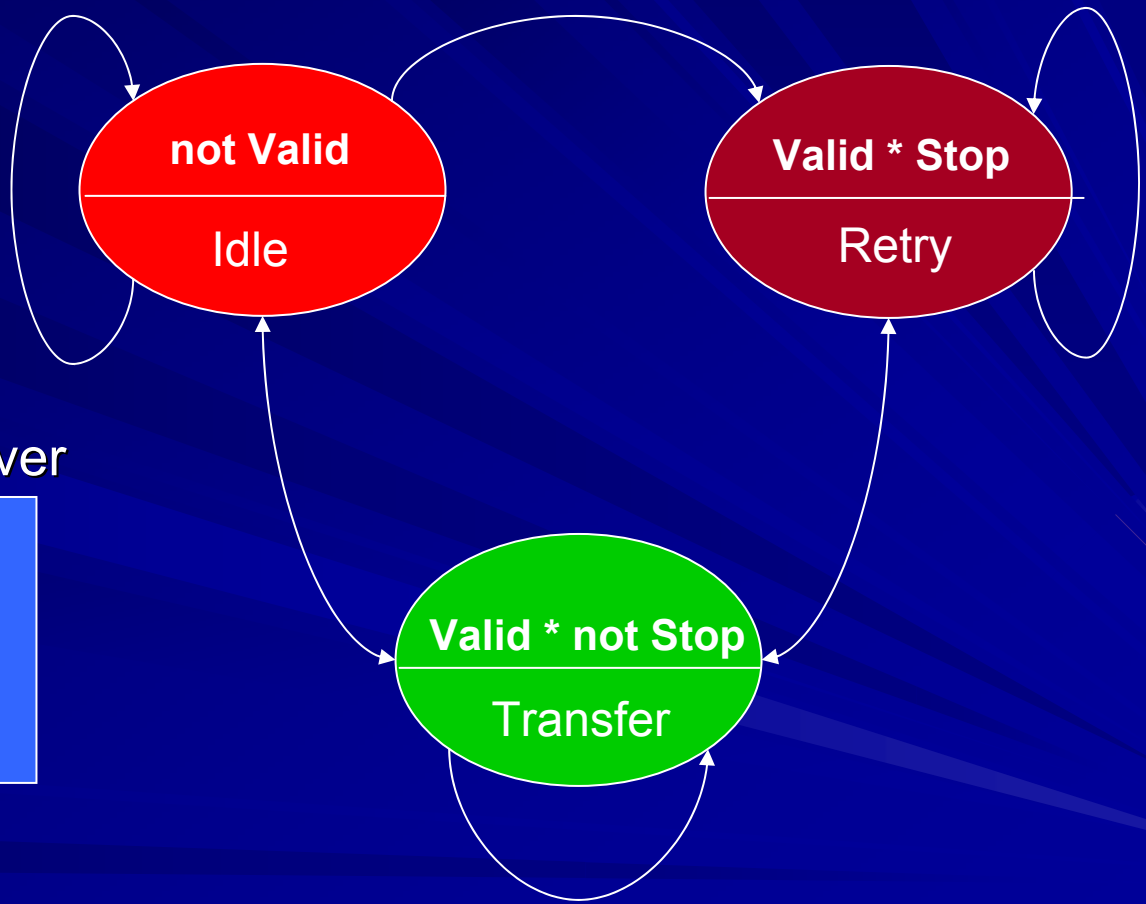

sender
receiver
Data
Valid
Stop
Data
Valid
Stop
En
En
En
En
V
V
V
V
1
0
S
S
S
S

SELF

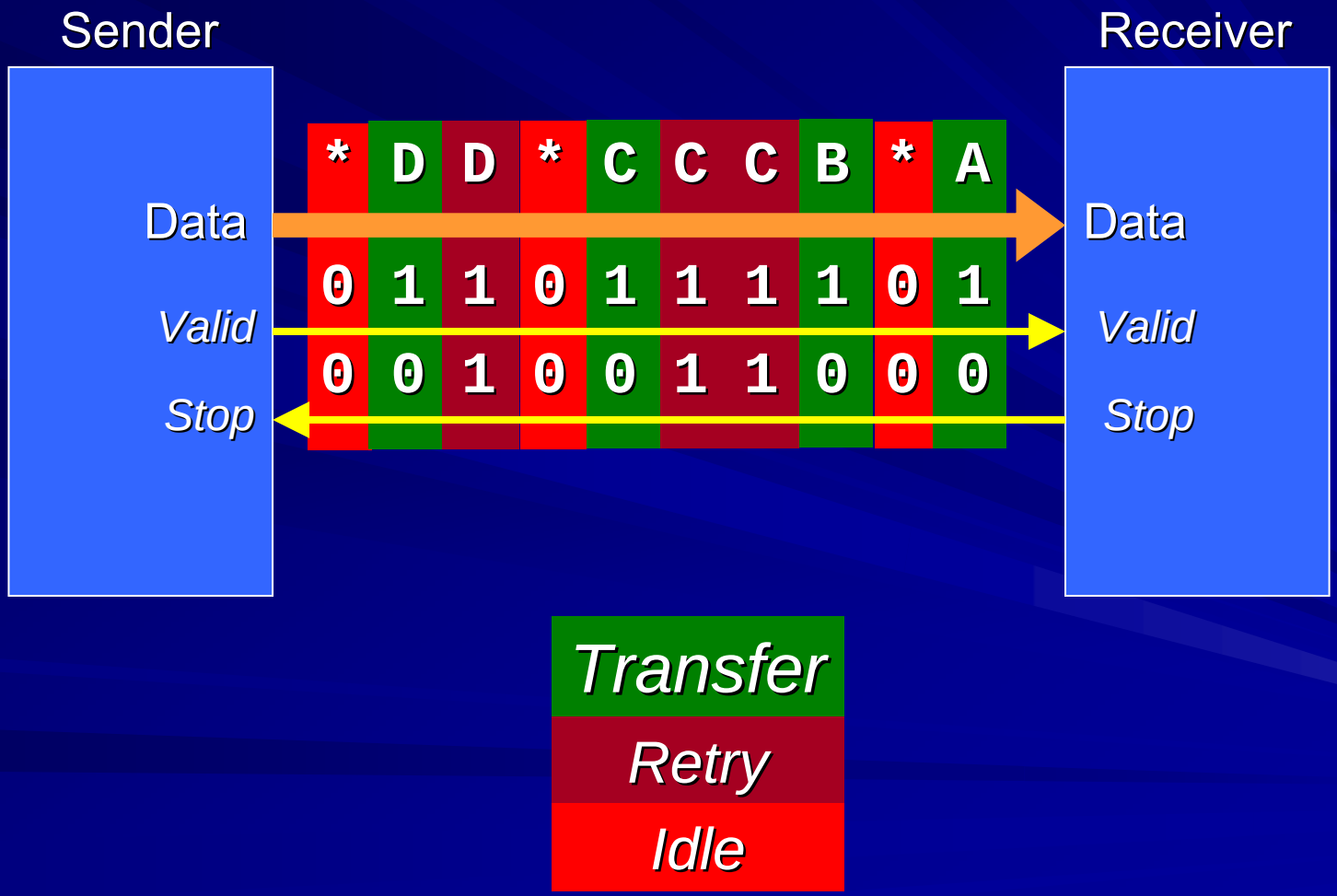

sender
receiver
Data
Valid
Stop
Data
Valid
Stop
En
En
En
En
V
V
V
V
1
0
S
S
S
S

Elastic channel and its protocol

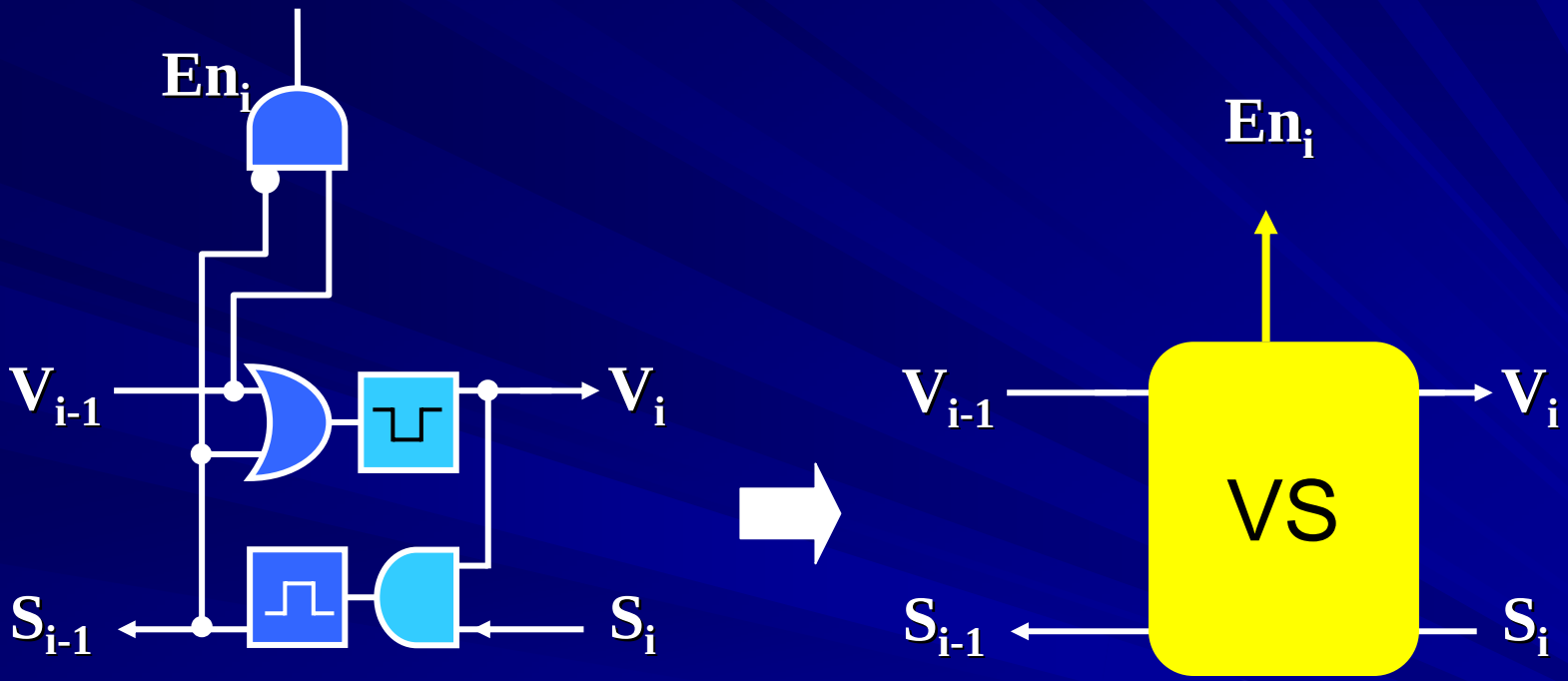




Elastic channel protocol

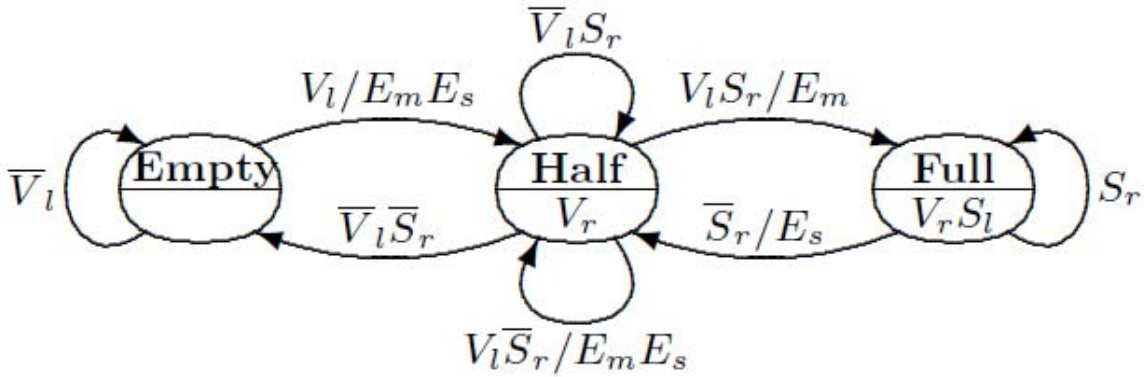


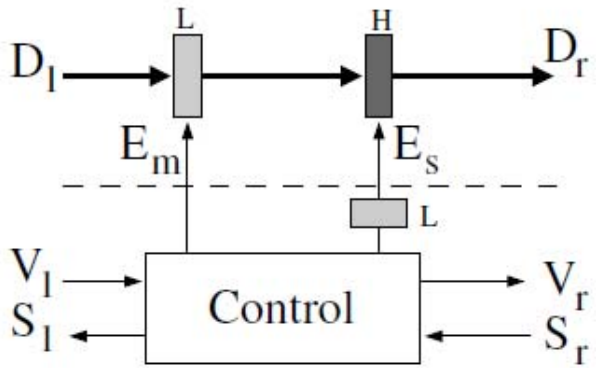
Basic VS block



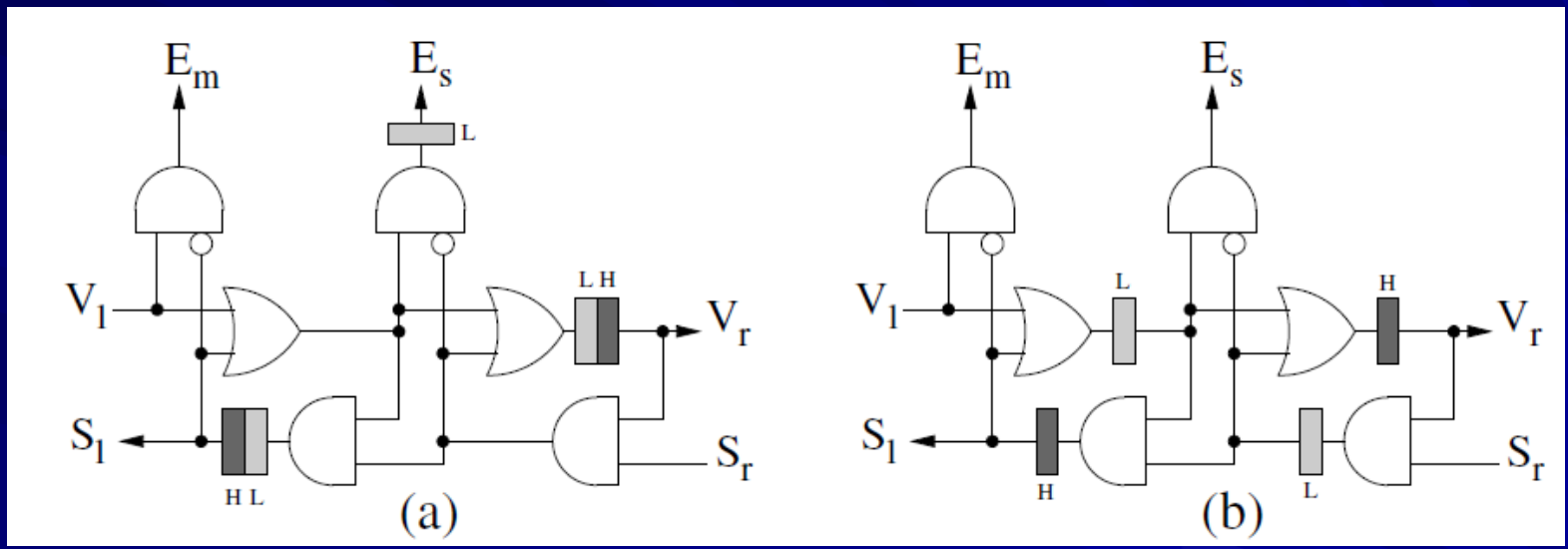
VS block + data-path latch = elastic HALF-buffer (EHB)
EHB + EHB = elastic buffer with capacity 2

Control specification of the EB

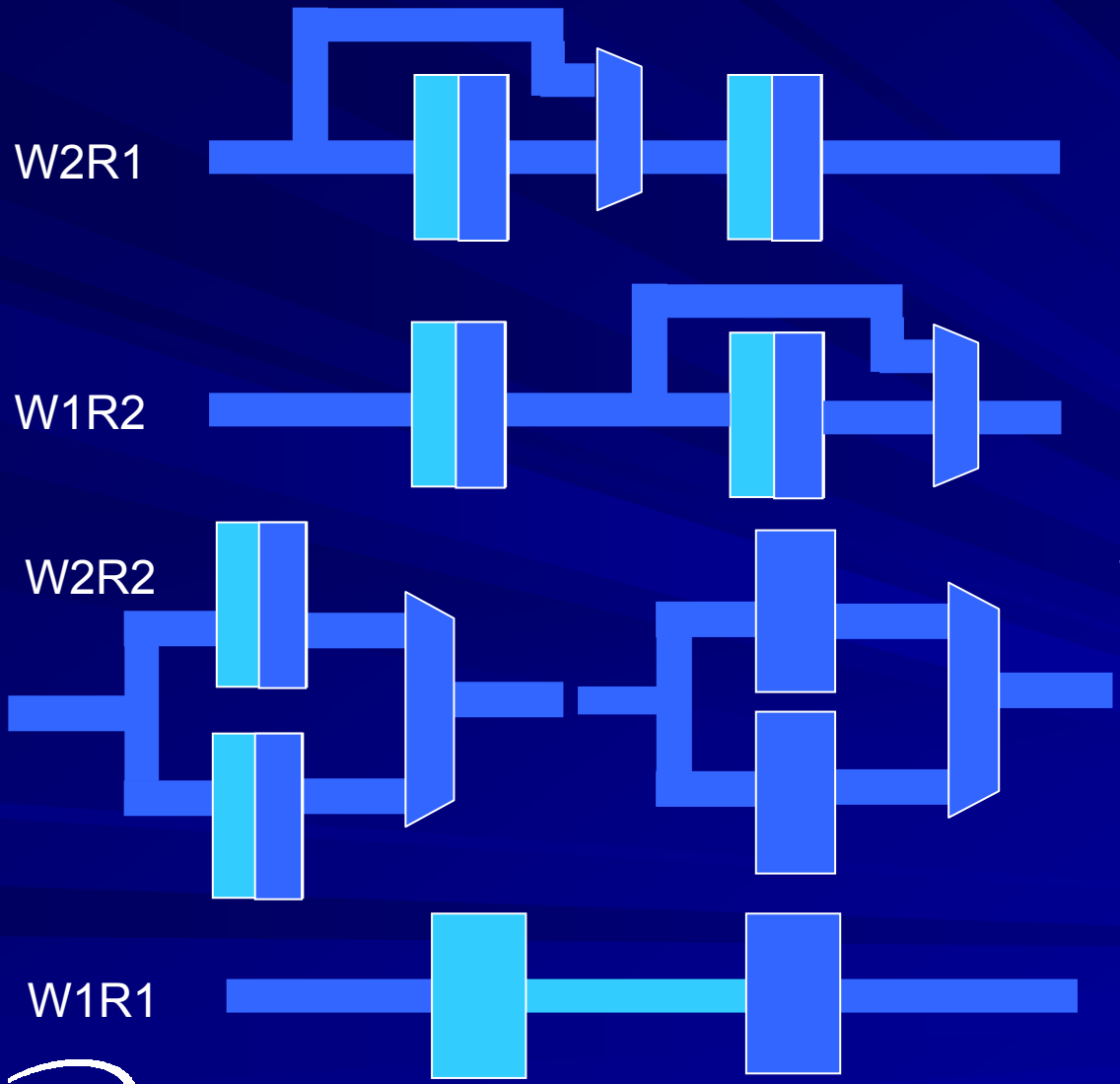




Two implementations



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

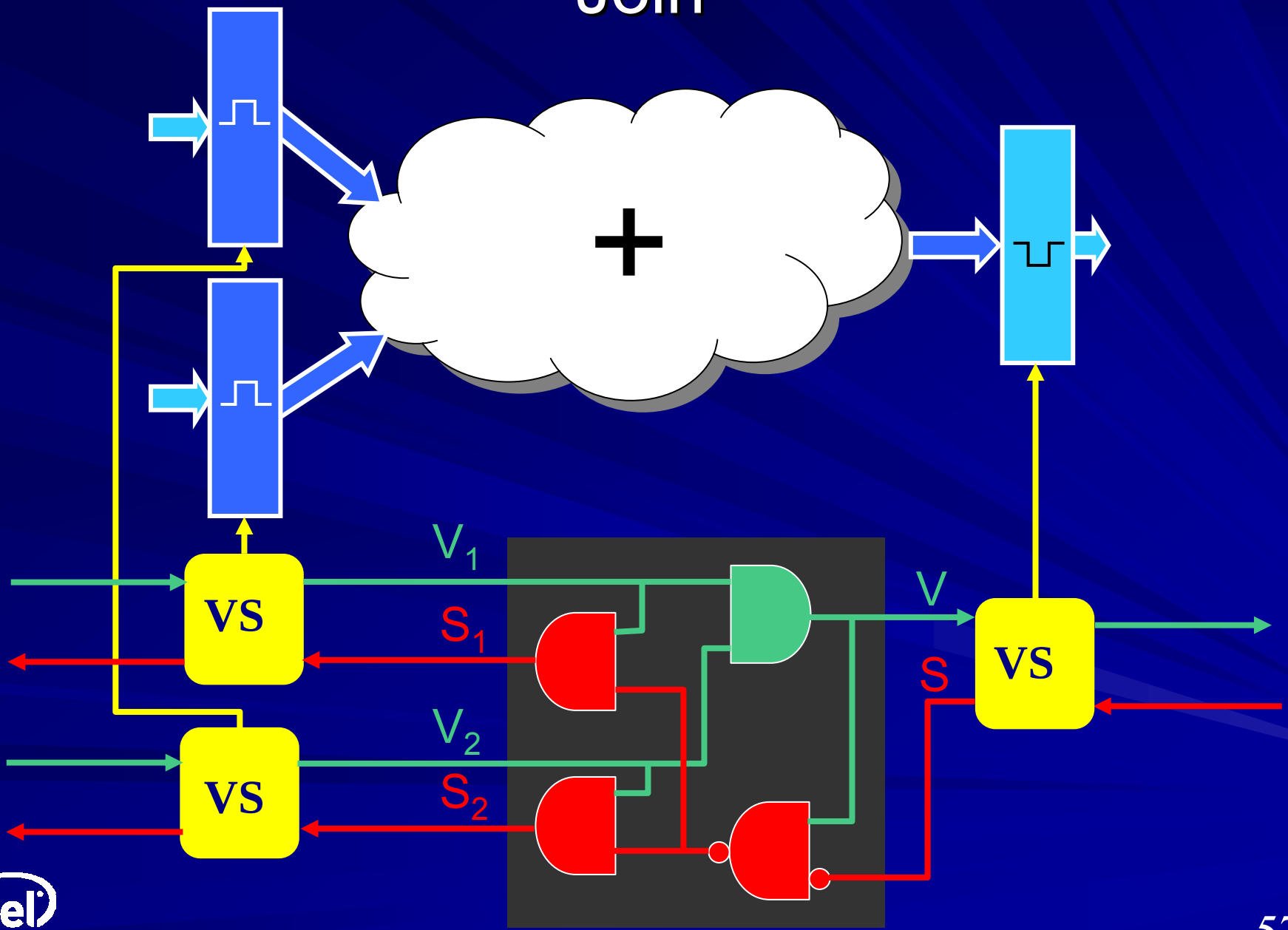
Typical case: (1,2) -
1 cycle forward latency
with capacity of 2

- Replaces “normal” registers
- Decoupling buffers

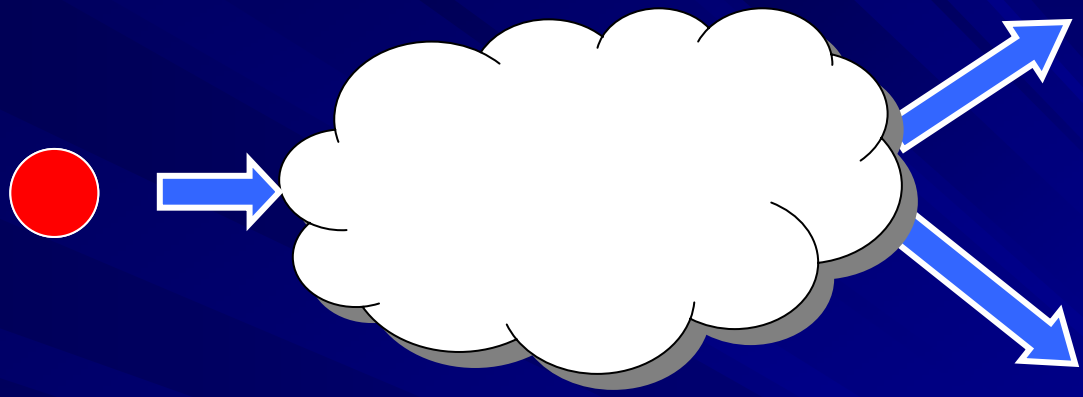
W1R1 Cannot be done with Single Edge Flops without double pumping

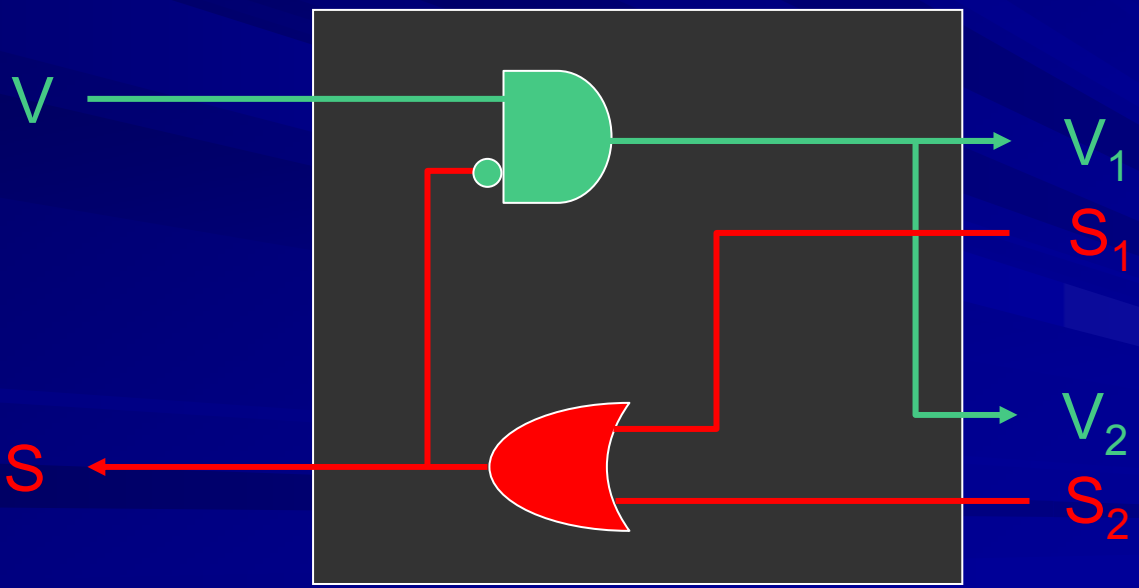
Can use latches inside Master-Slave as shown before

Join

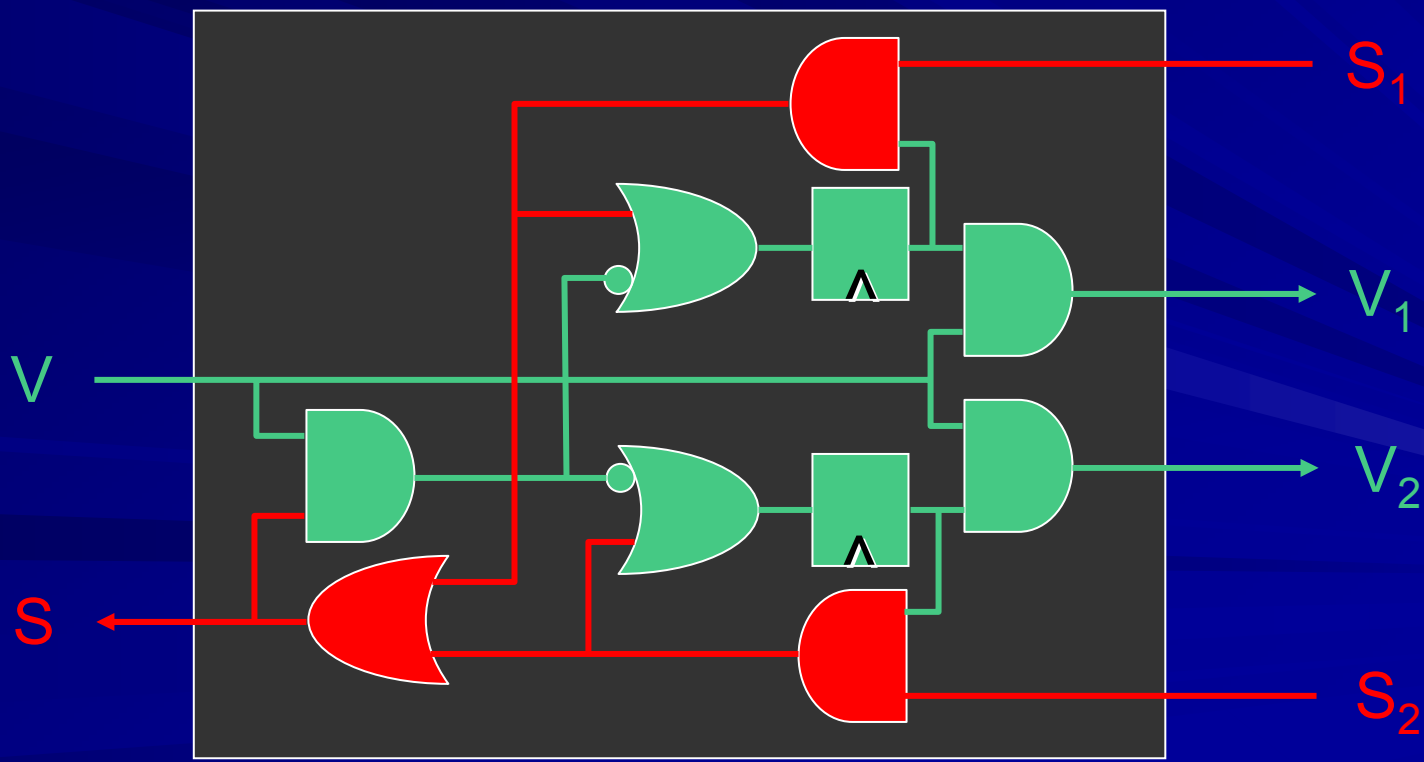


(Lazy) Fork

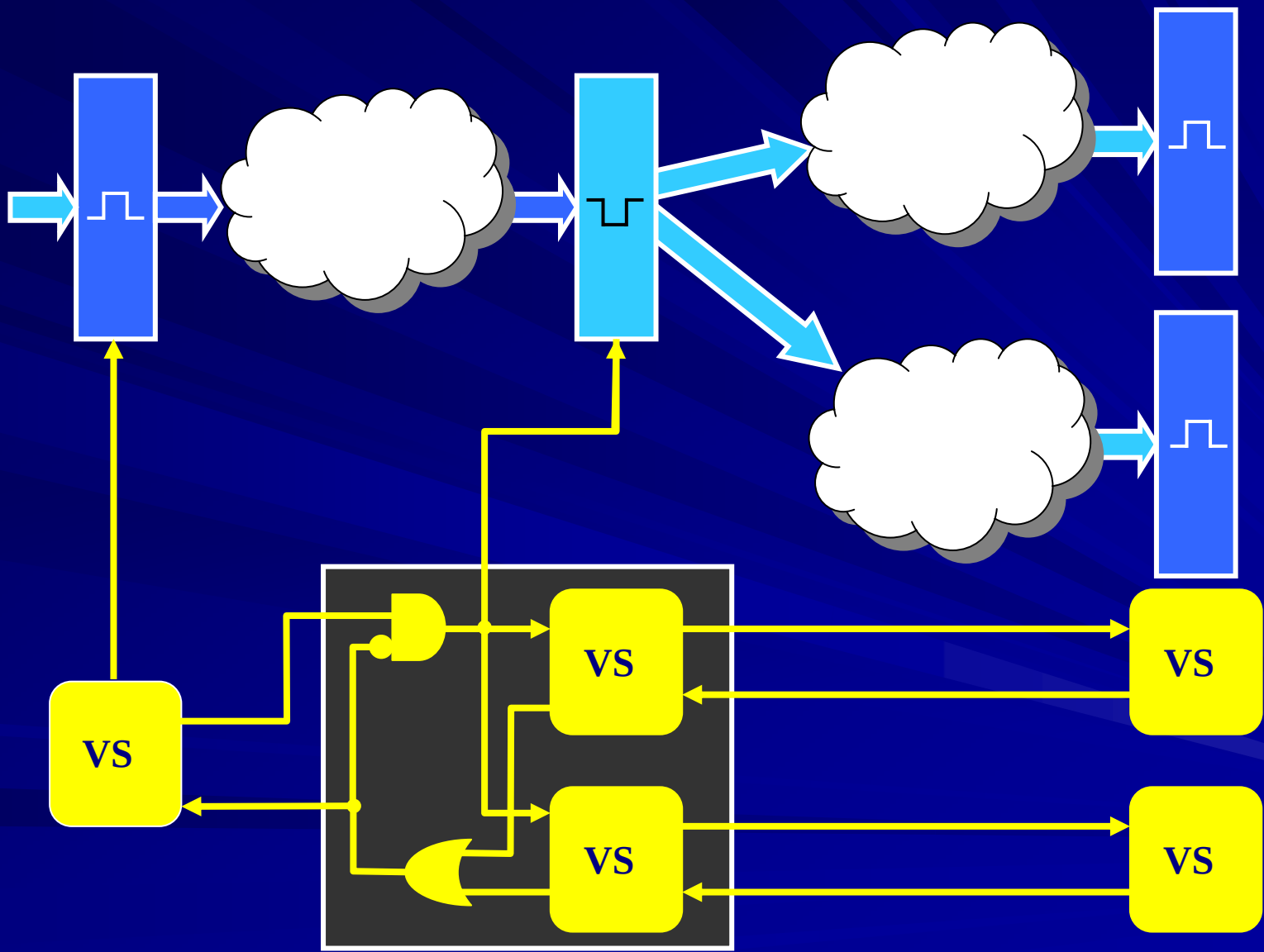




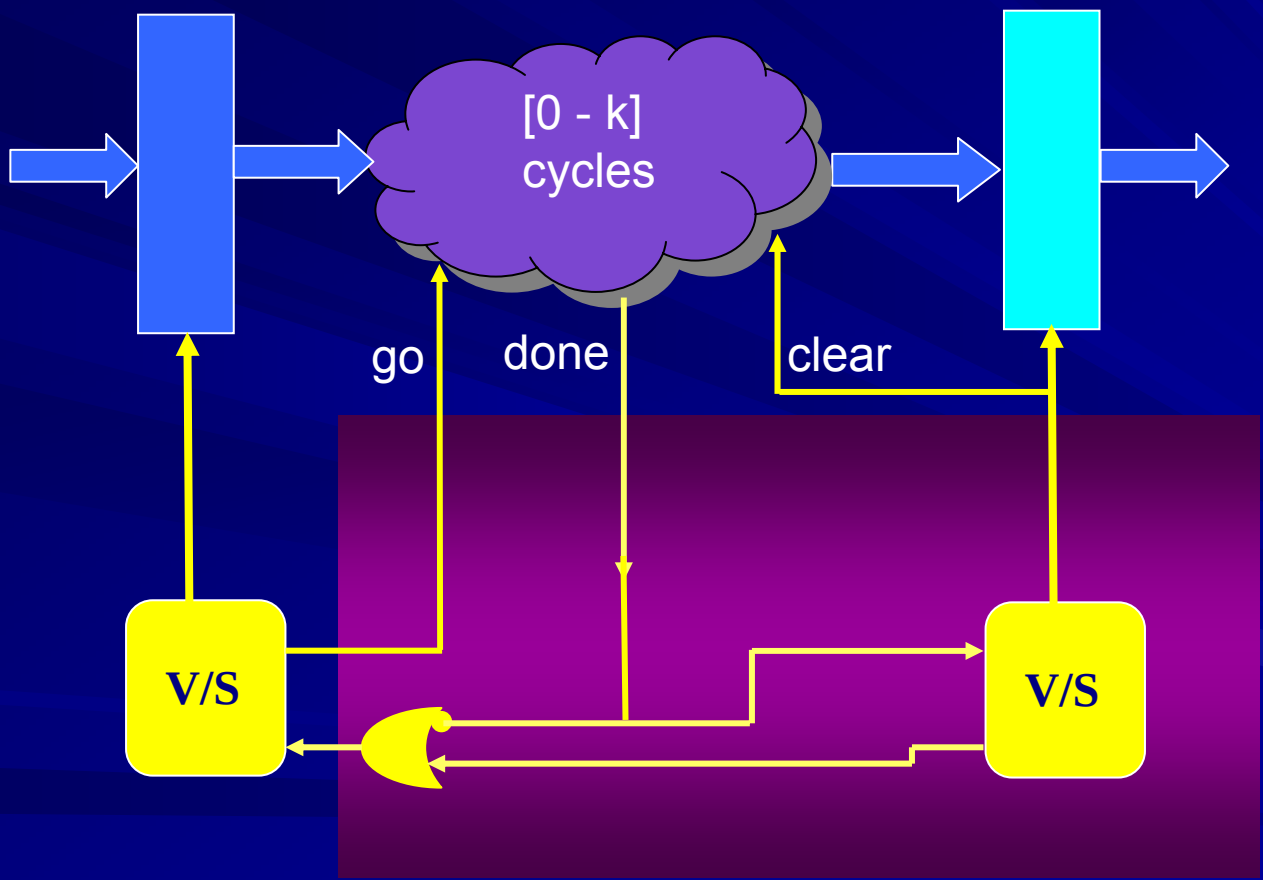
Eager Fork



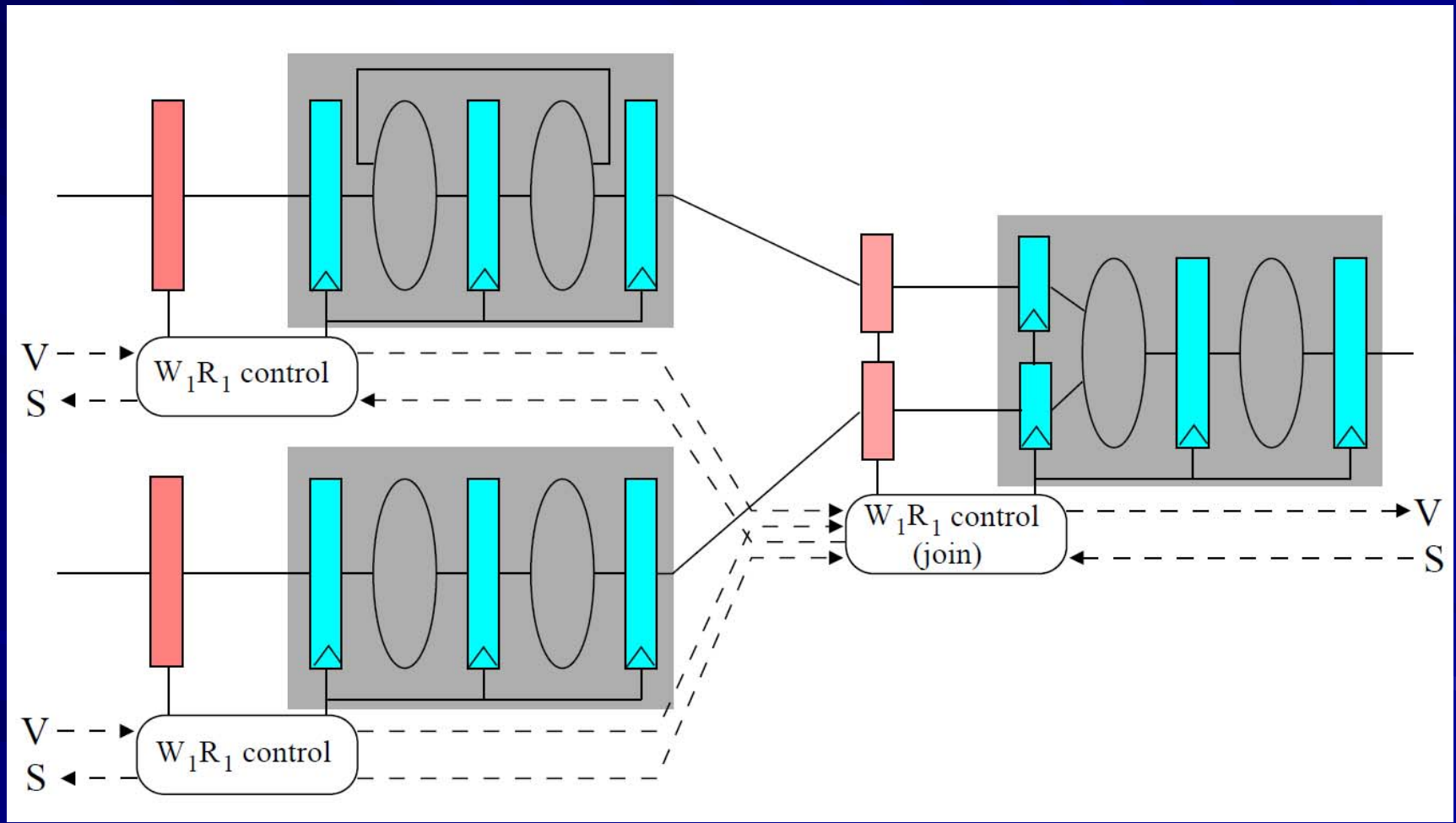
Eager fork (another implementation)



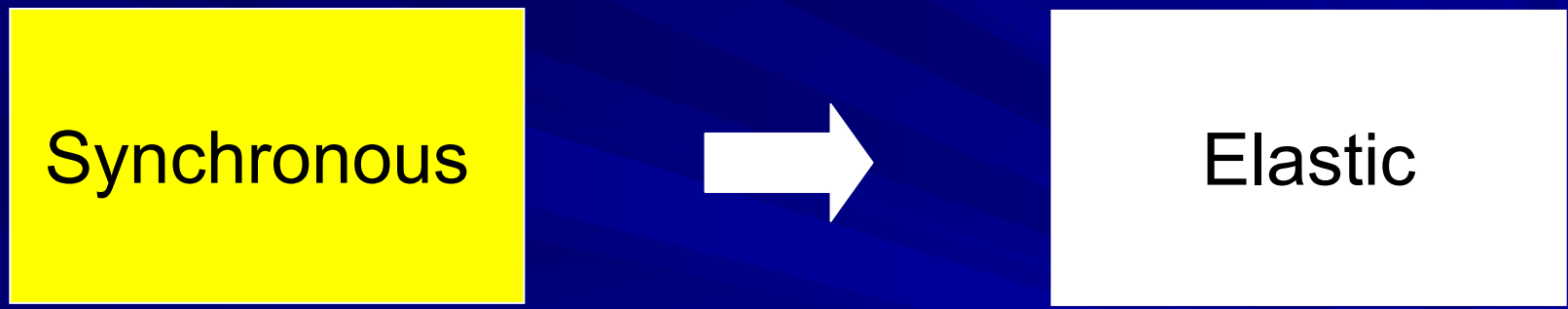
Variable Latency Units

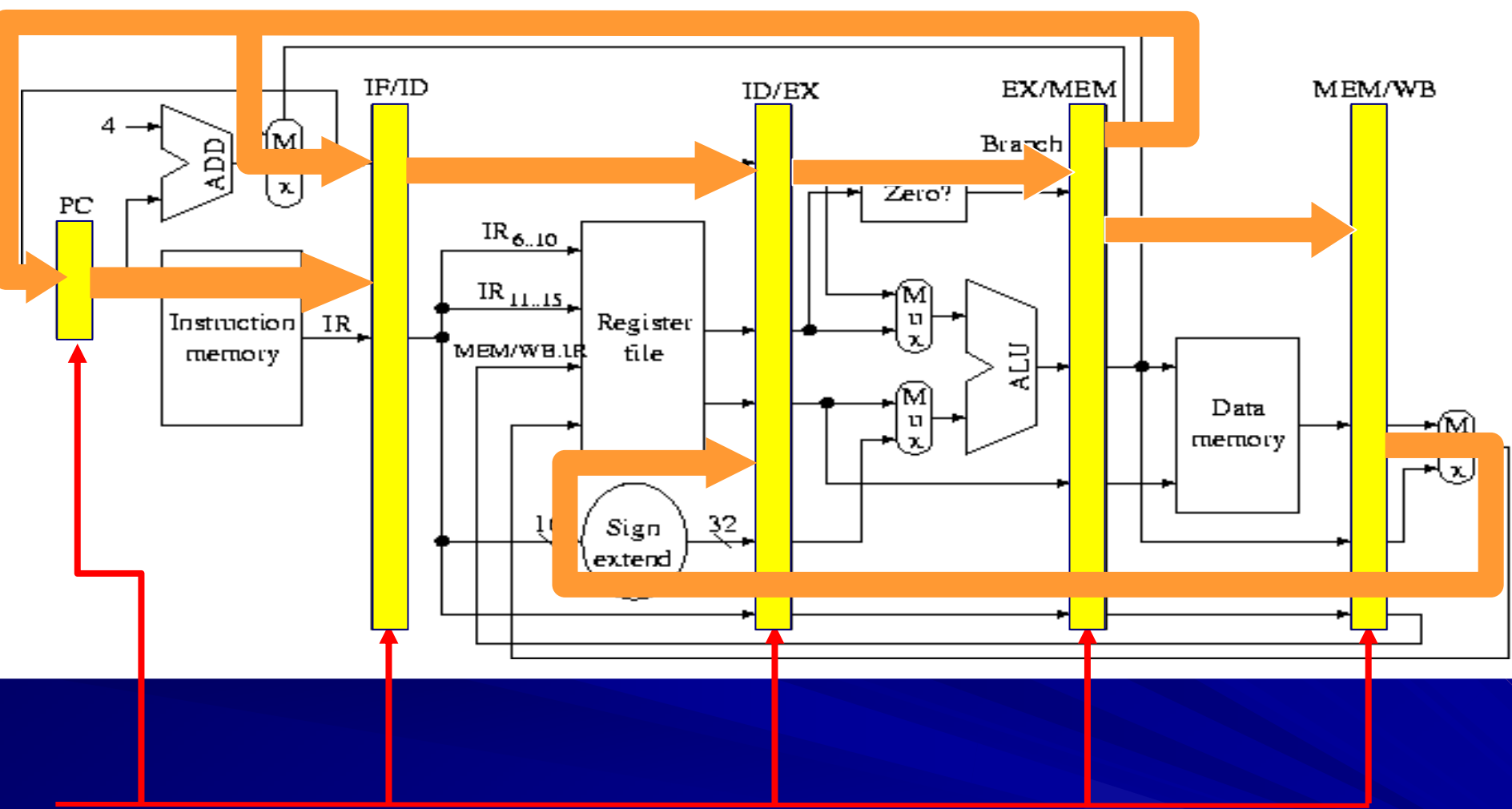


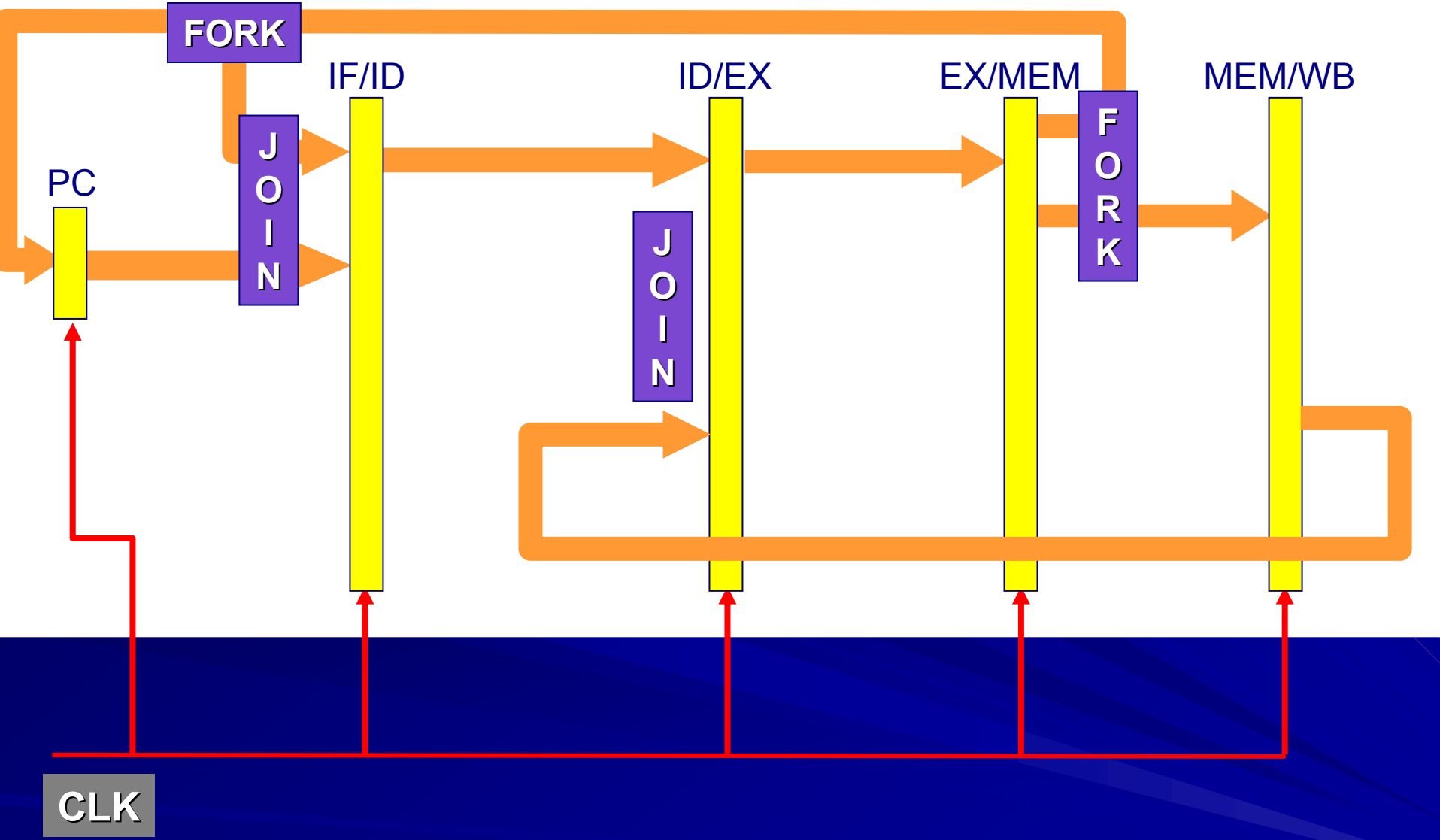
Coarse grain control

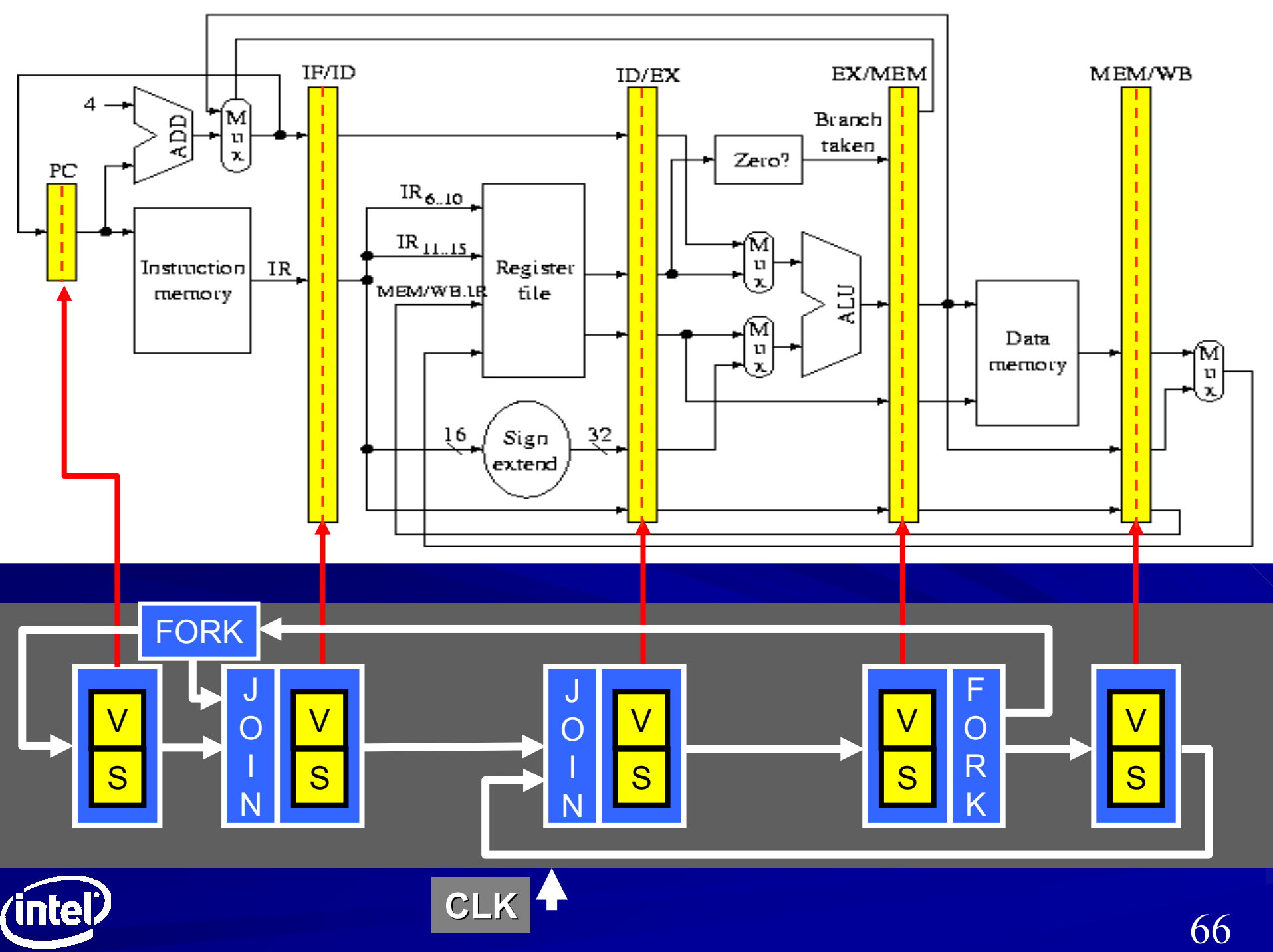


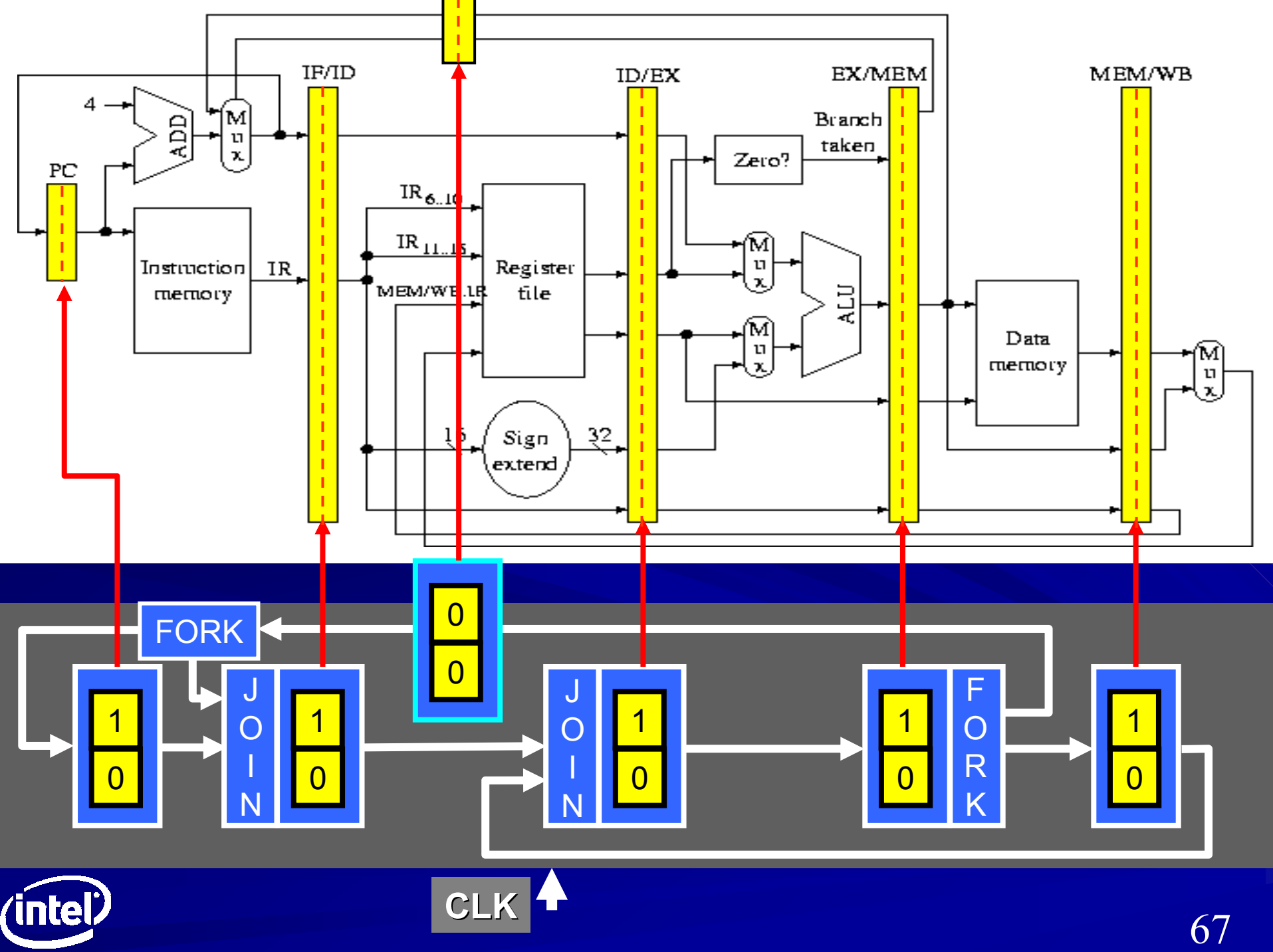
Elasticization

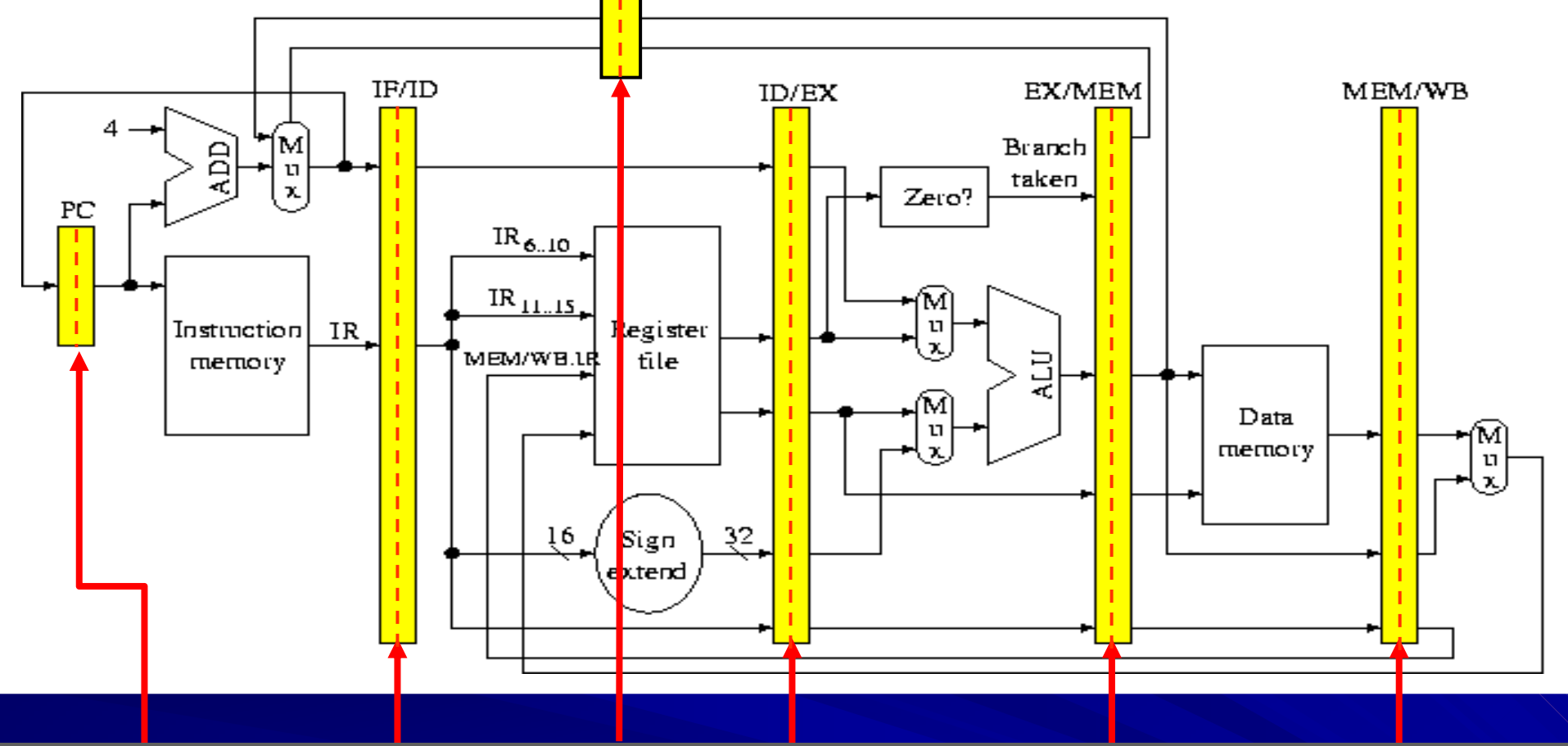



IF/ID
ID/EX
EX/MEM
MEM/WB
4
ADD
M
x
PC
Instruction
memory
IR
IR 6..10
IR 11..15
MEM/WB.IR
Register
file
Sign
extend
32
Zero?
Branch
M
u
x
ALU
Data
memory
CLK


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
ALU
Data memory
FORK
V
S
JOIN
F
O
R
K
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
ALU
Data memory
16
Sign extend
32
FORK
JOIN
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

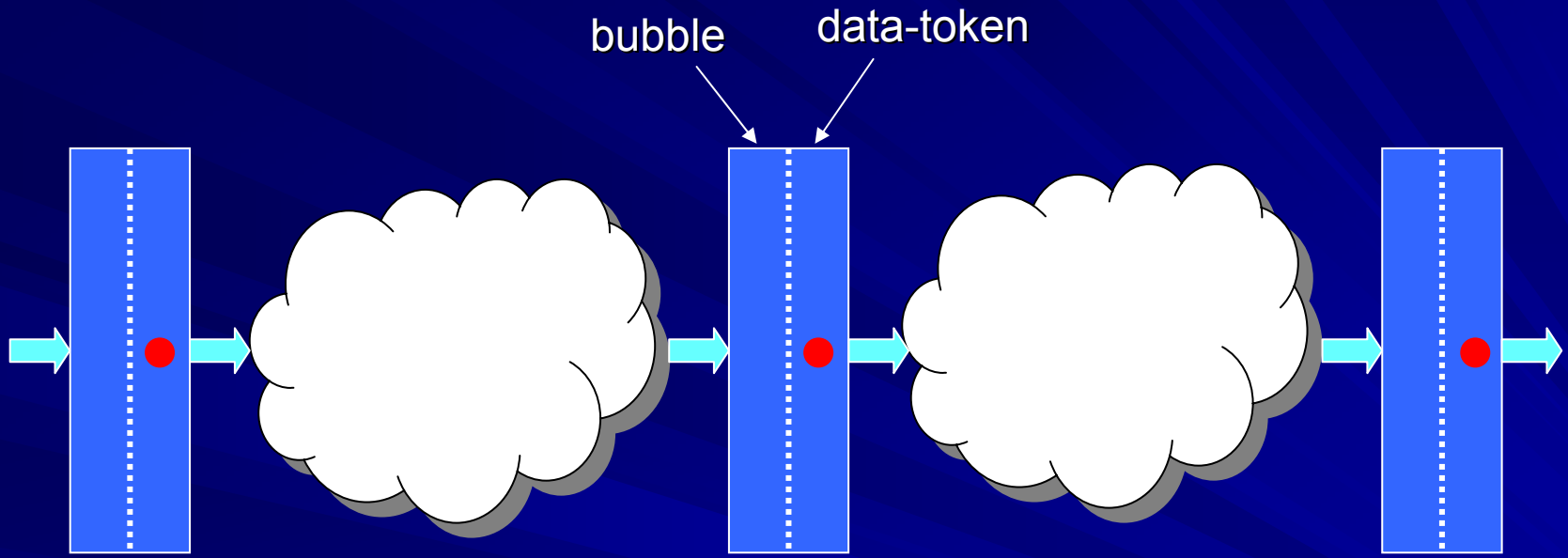
D:	a	*	b	*	*	c	d	e	*	d	f	*	g	h	*	*	i	j	…
V:	1	0	1	1	0	1	1	1	1	1	1	1	1	1	0	1	1	1	…
S:	0	0	0	1	0	0	0	0	1	0	0	1	0	0	0	1	0	0	…

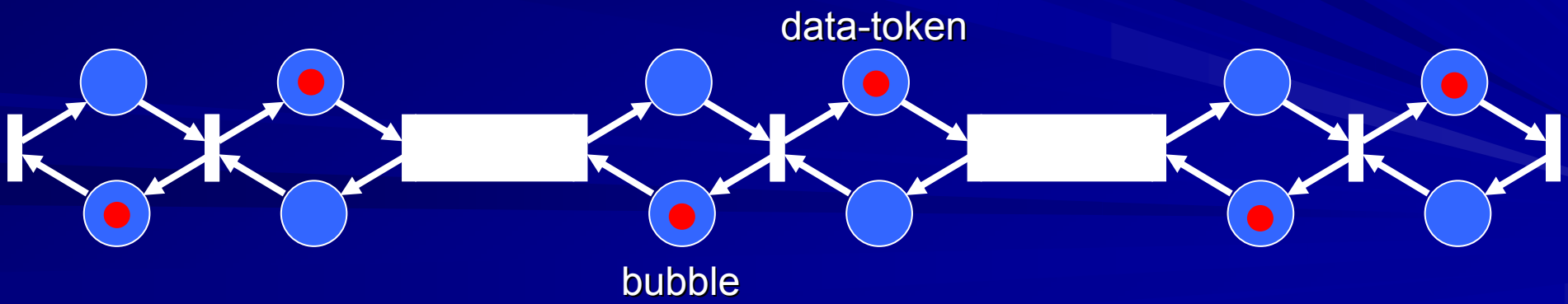
Transfer sub-stream = original stream

Called: transfer equivalence, flow equivalence, or latency equivalence

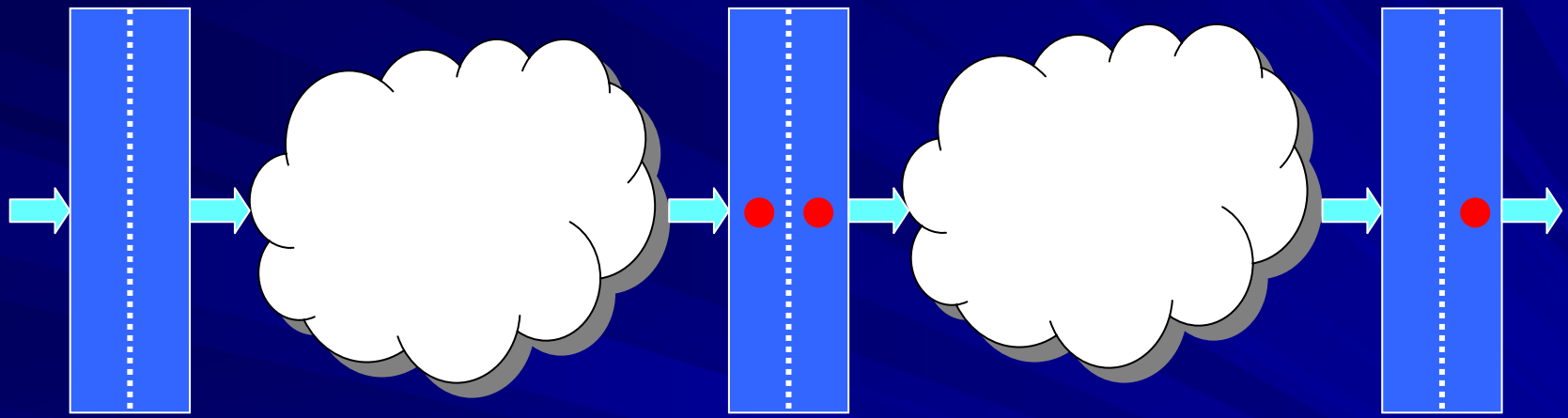
Marked Graph models of elastic systems

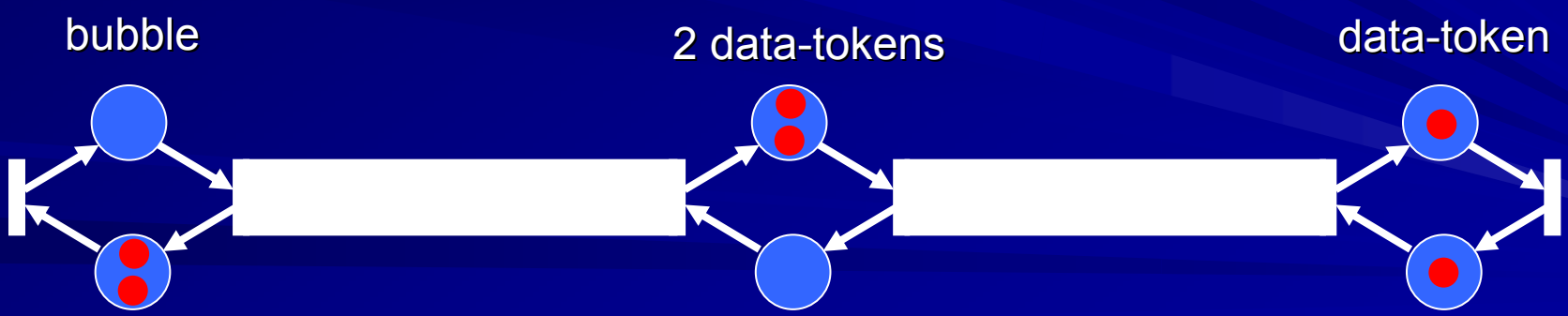
Modelling elastic control with Petri nets





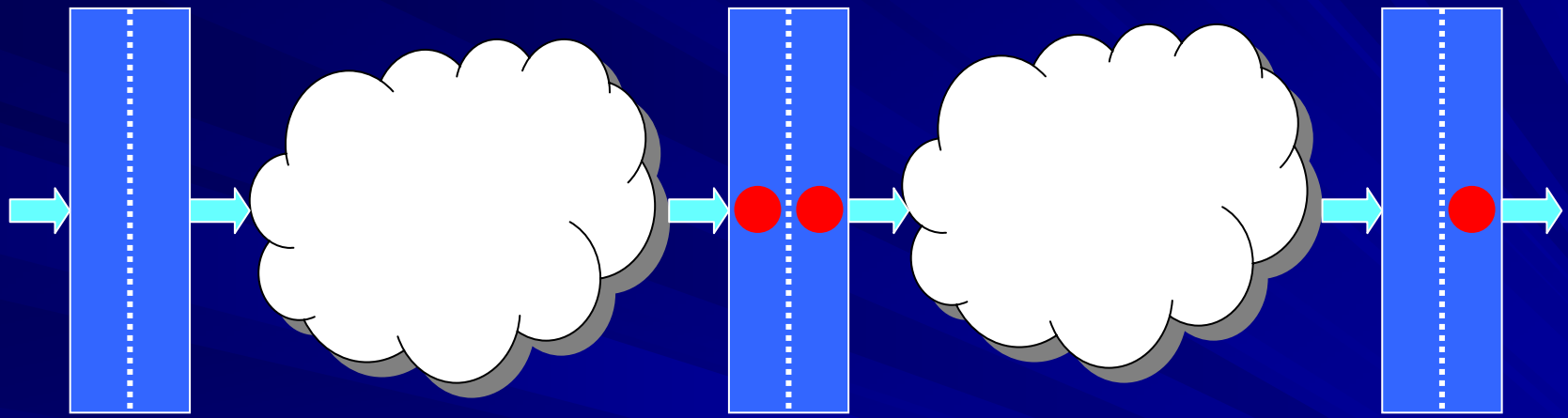
Modelling elastic control with Petri nets

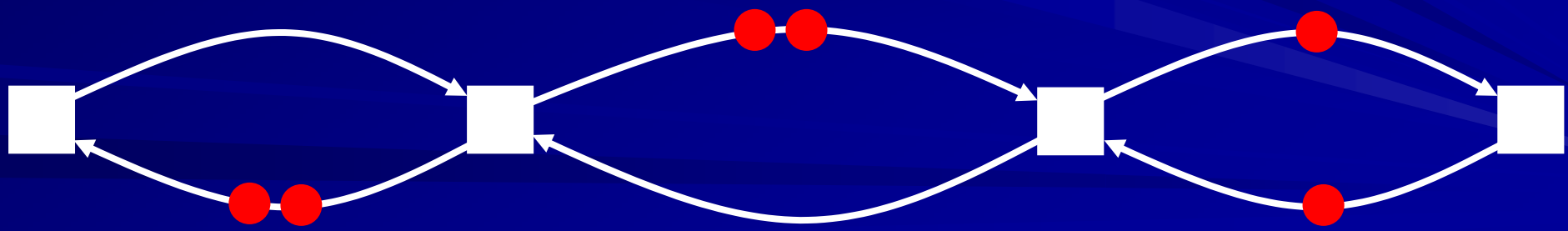




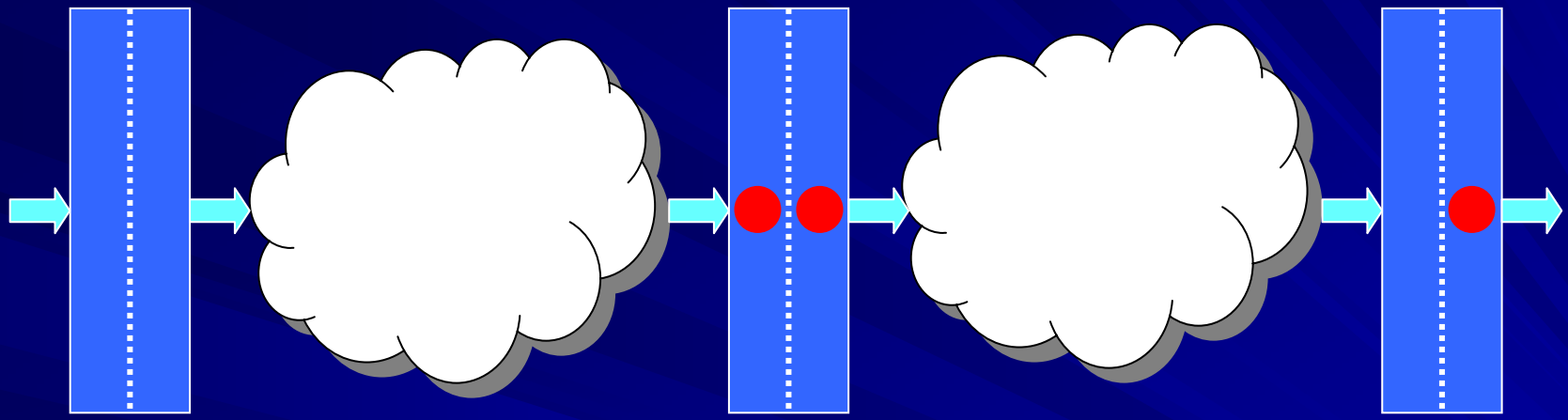
Hiding internal transitions of elastic buffers

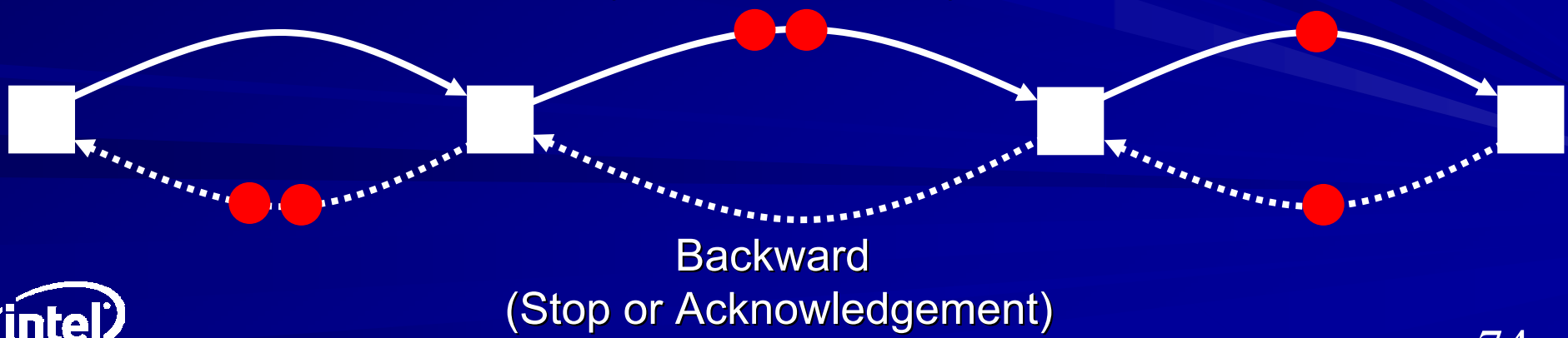
Modelling elastic control with Marked Graphs



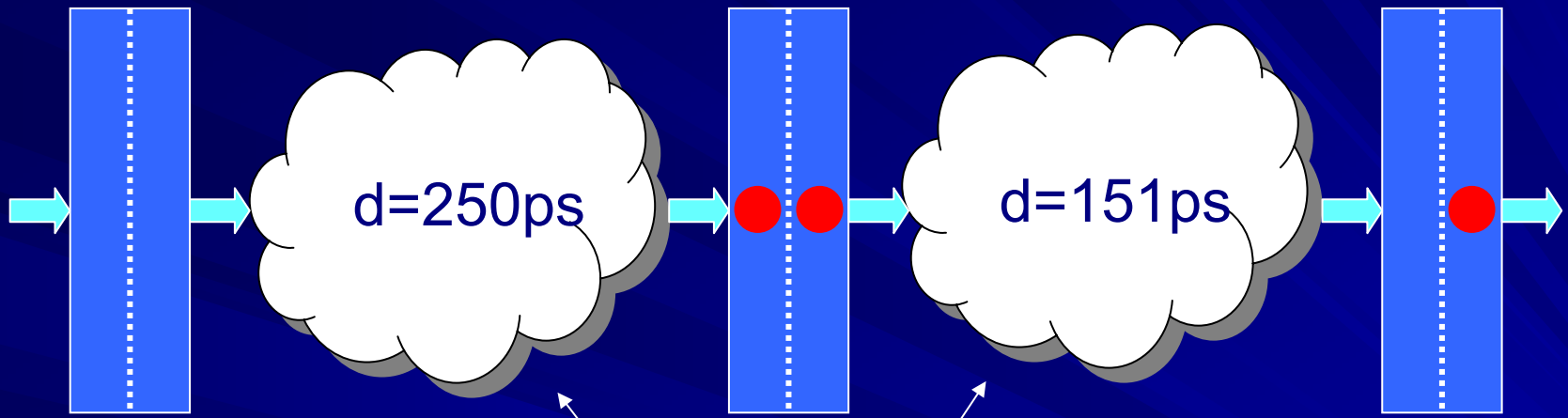


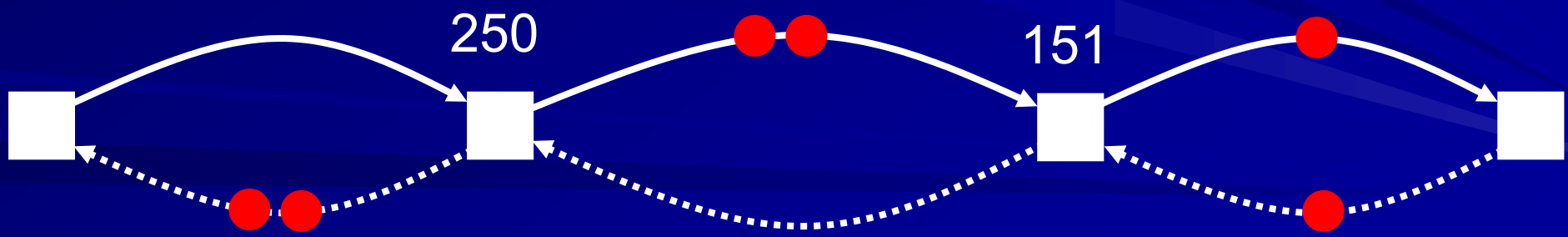
Modelling elastic control with Marked Graphs



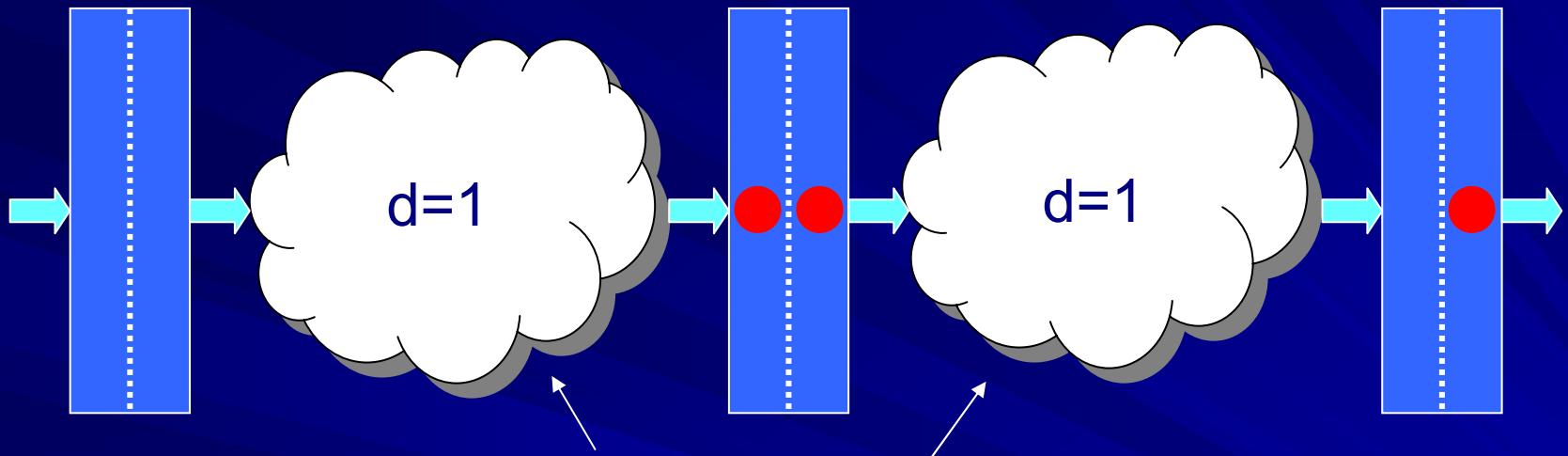


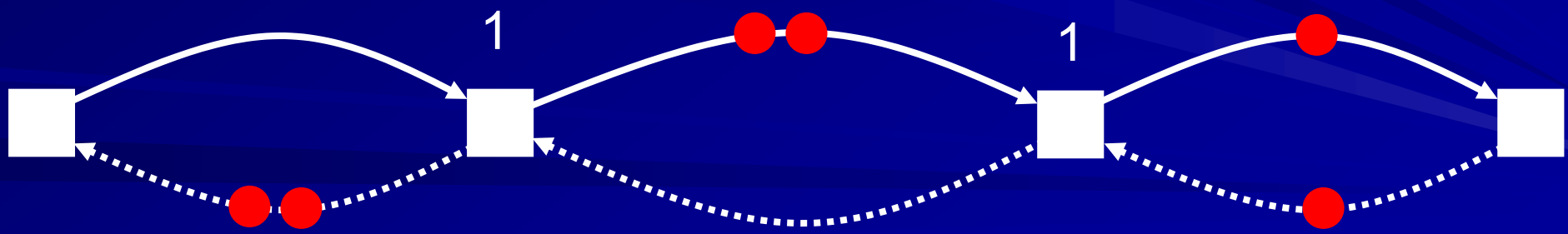
Elastic control with Timed Marked Graphs. Continuous time = asynchronous



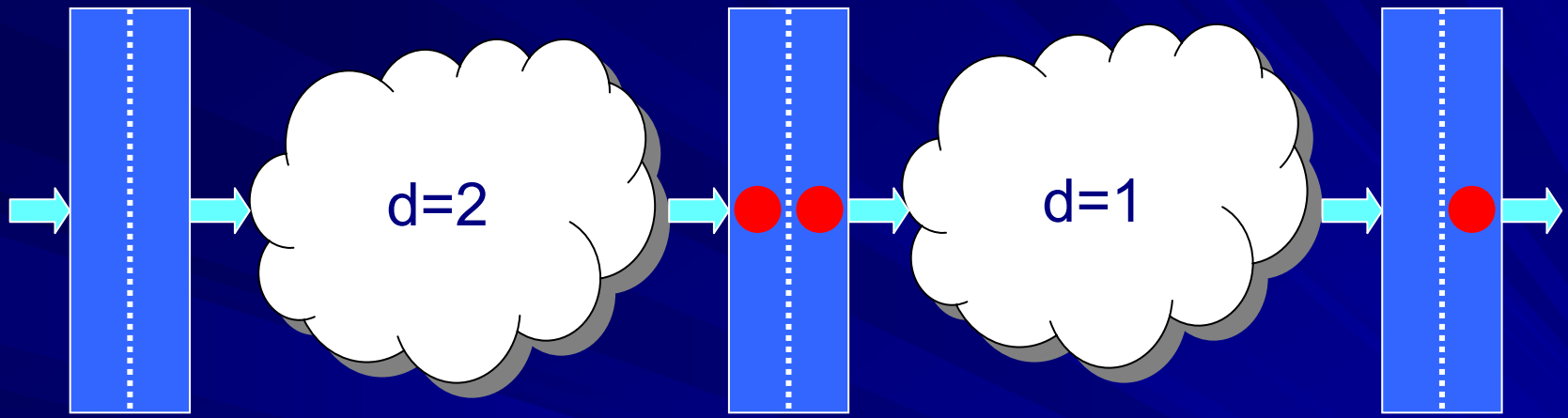


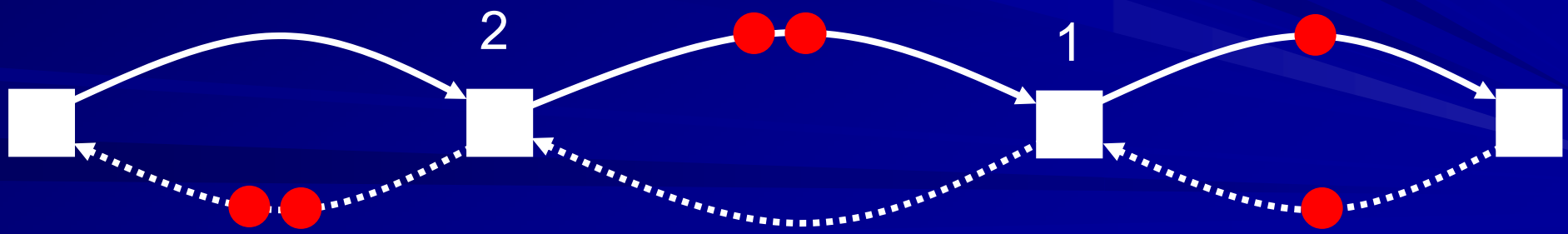
Elastic control with Timed Marked Graphs. Discrete time = synchronous elastic



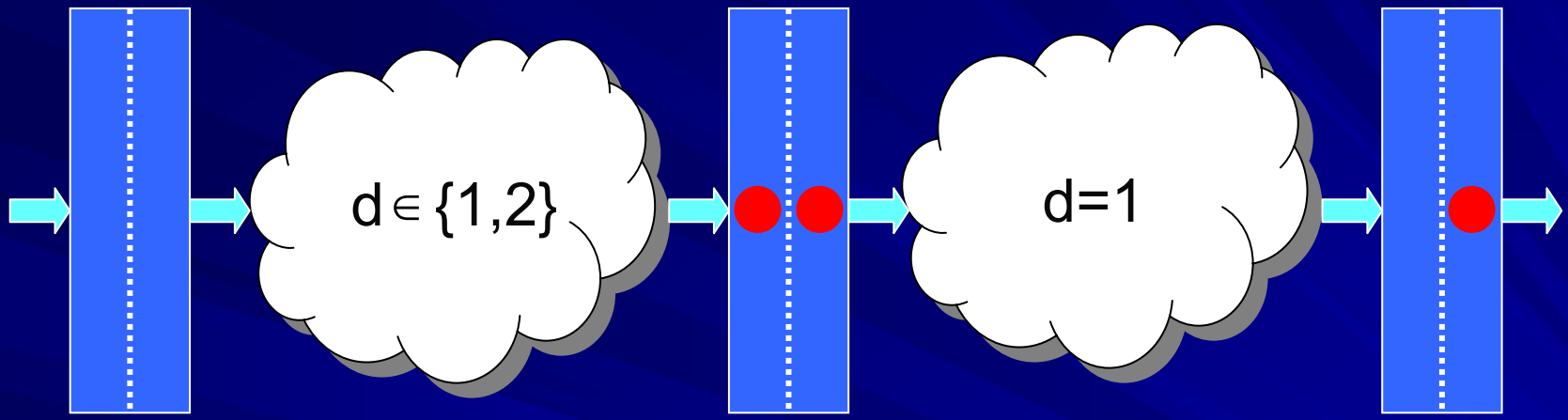


Elastic control with Timed Marked Graphs. Discrete time. Multi-cycle operation

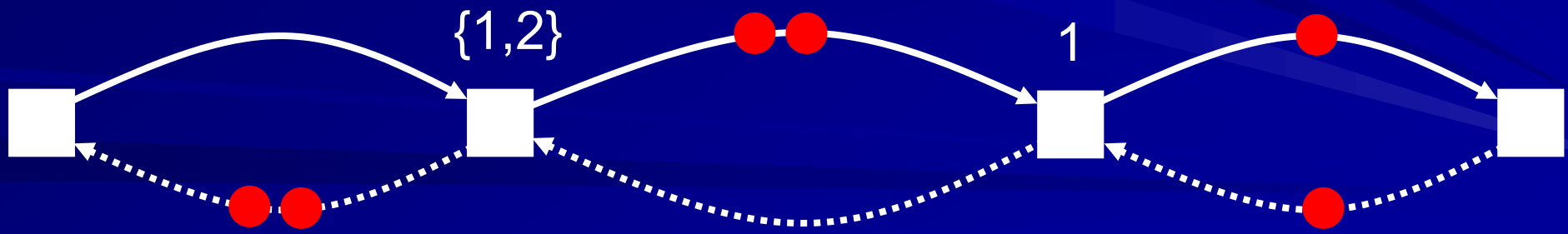




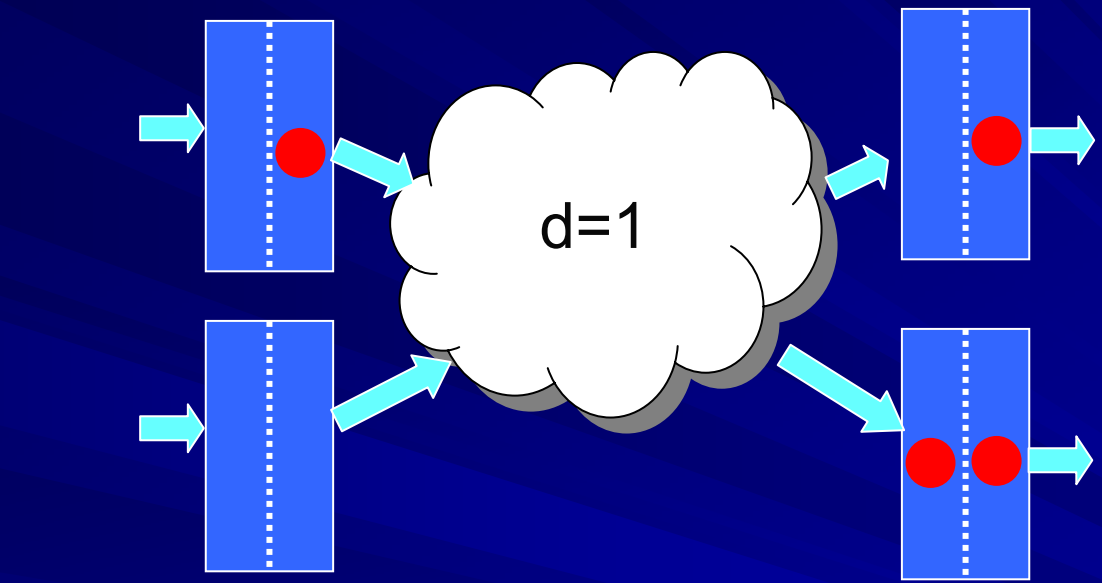
Elastic control with Timed Marked Graphs. Discrete time. Variable latency operation

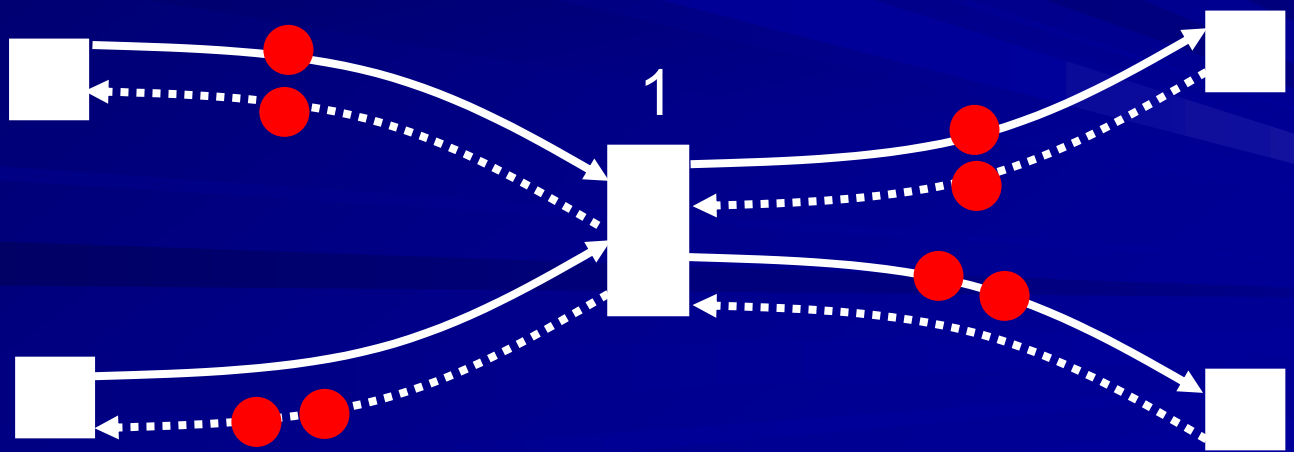


e.g. discrete probabilistic distribution:
average latency 0.8*1 + 0.2*2 = 1.2

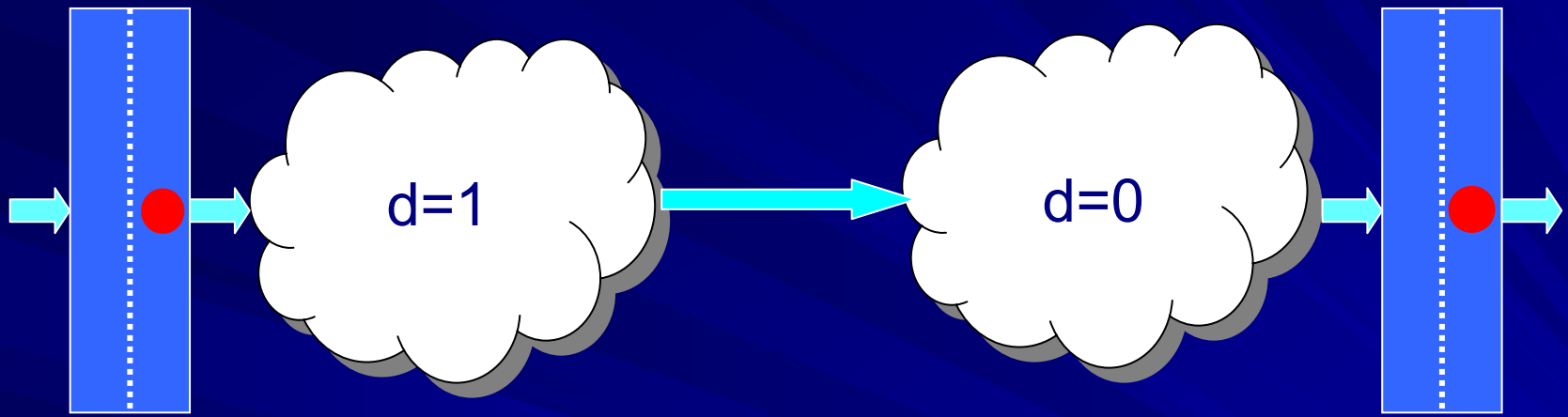


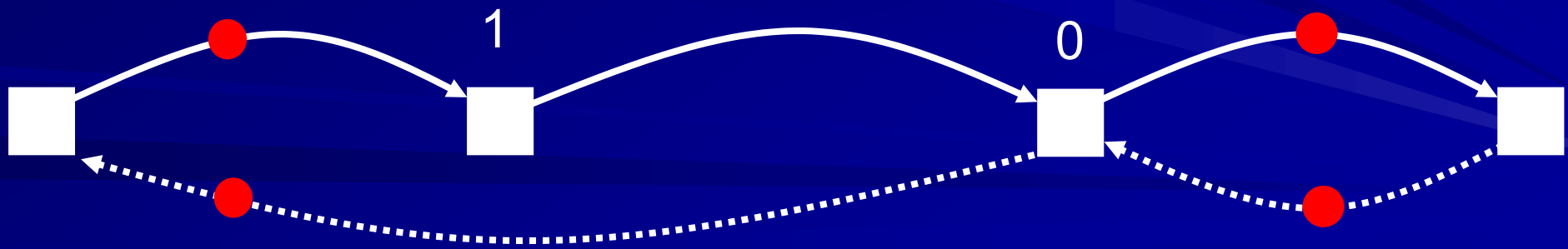
Modeling forks and joins





Modelling combinational elastic blocks





Elastic Marked Graphs

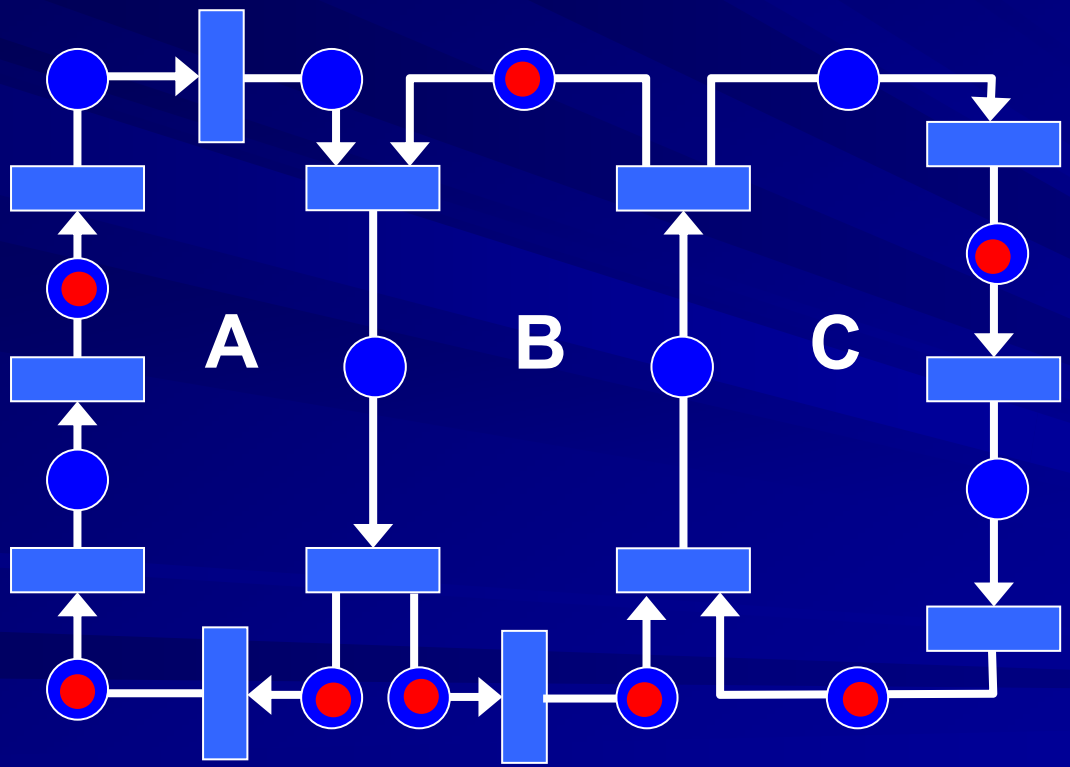
- An Elastic Marked Graph (EMG) is a Timed MG such that for any arc *a* there exists a complementary arc *a'* satisfying the following condition
  - $\bullet a = a' \bullet$ and $\bullet a' = a \bullet$
- Initial number of tokens on *a* and *a'* $(M_0(a)+M_0(a'))$ = capacity of the corresponding elastic 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

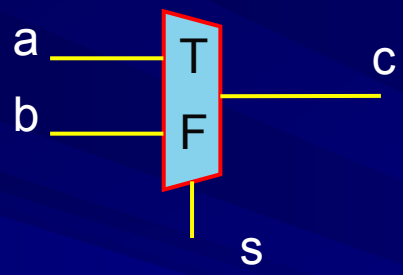
Efficient algorithms: (Karp 1978), (Dasdan,Gupta 1998)

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

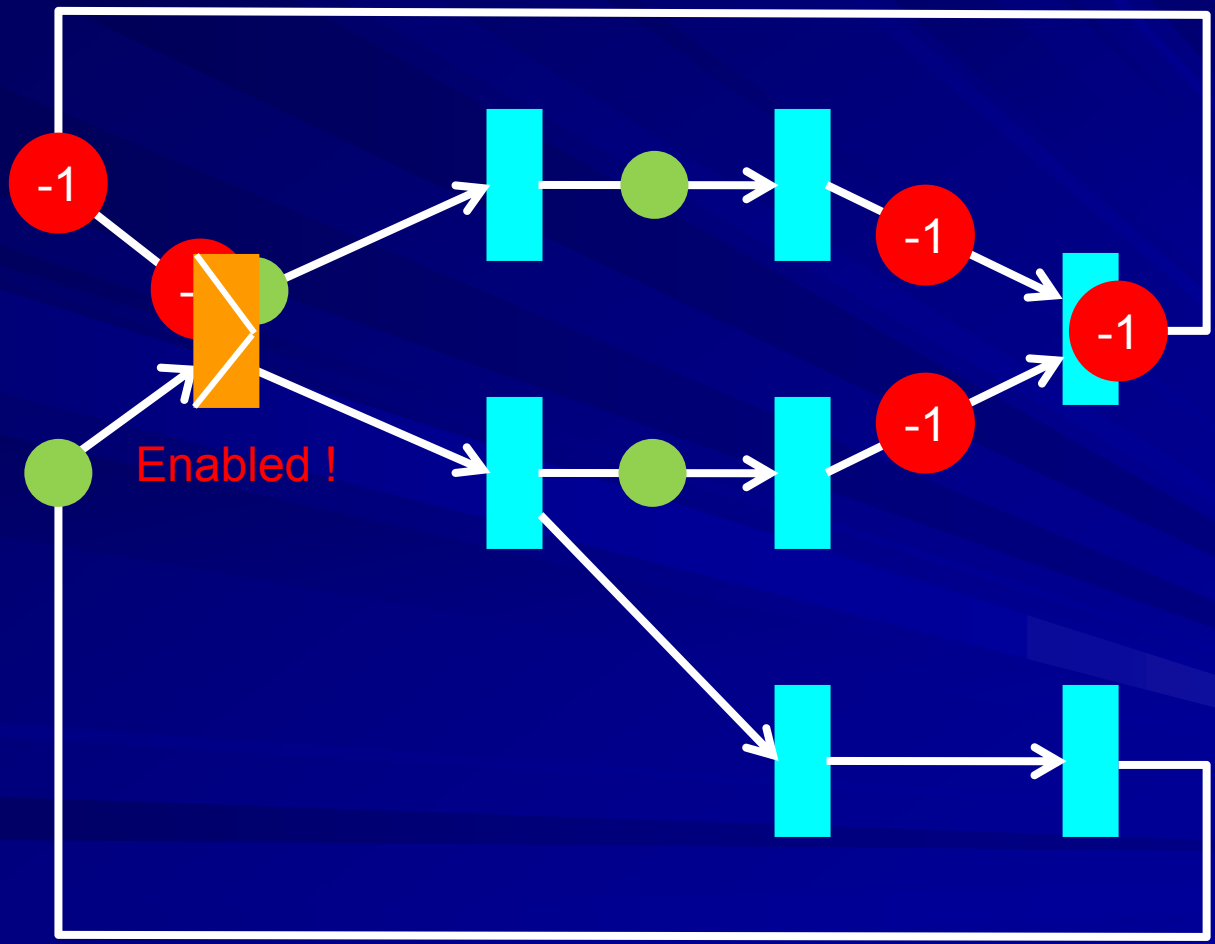
- Enabling rules for a node:
  - Positive enabling: $M(a) > 0$ for every input arc
  - Early enabling (for early enabling nodes):
    $M(a) > 0$ for some input arcs
  - Negative enabling: $M(a) < 0$ for every output arc

- 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, 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 $M : M(c) = M_0(c)$

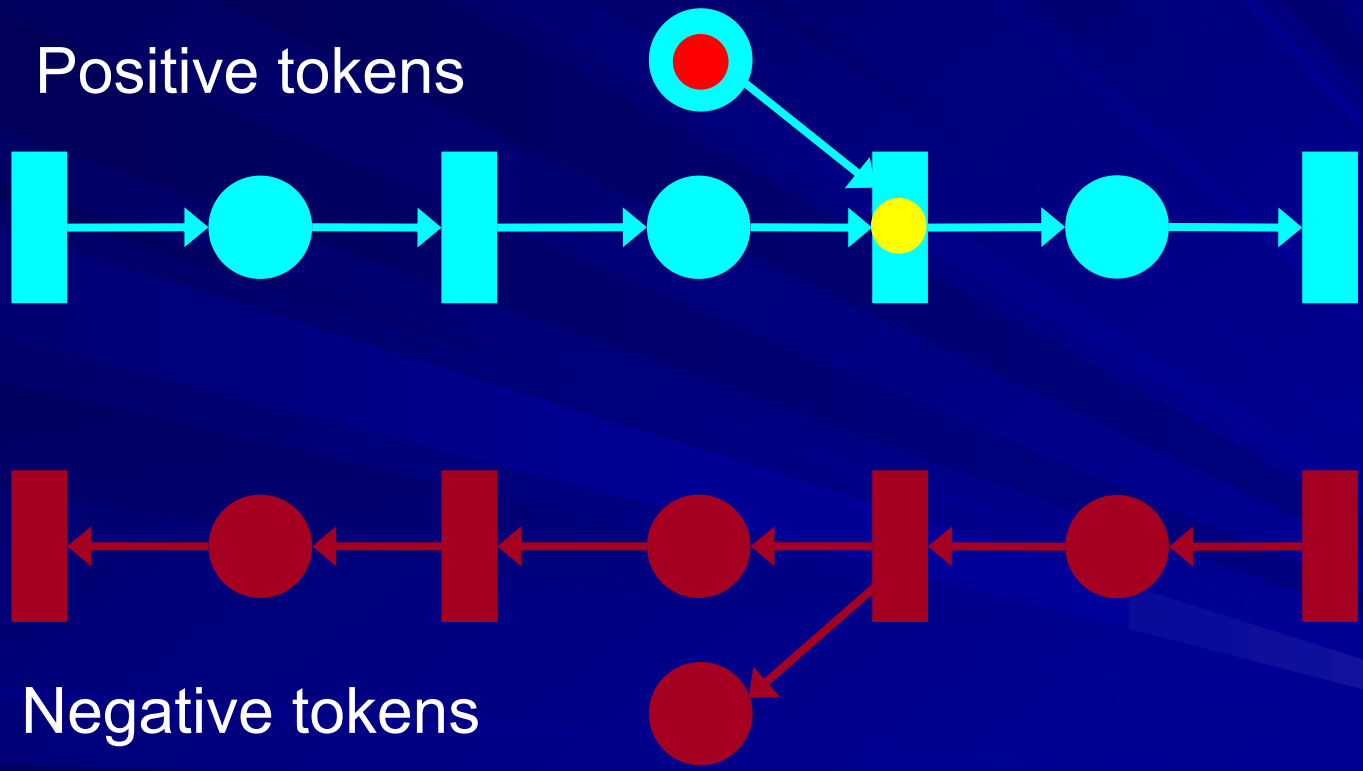
- Liveness. A strongly connected passive DMG is live iff for every cycle *c:* $M(c) > 0$.
  - For DMGs this is a sufficient condition of liveness
  - It is also a necessary condition for positive liveness

- 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

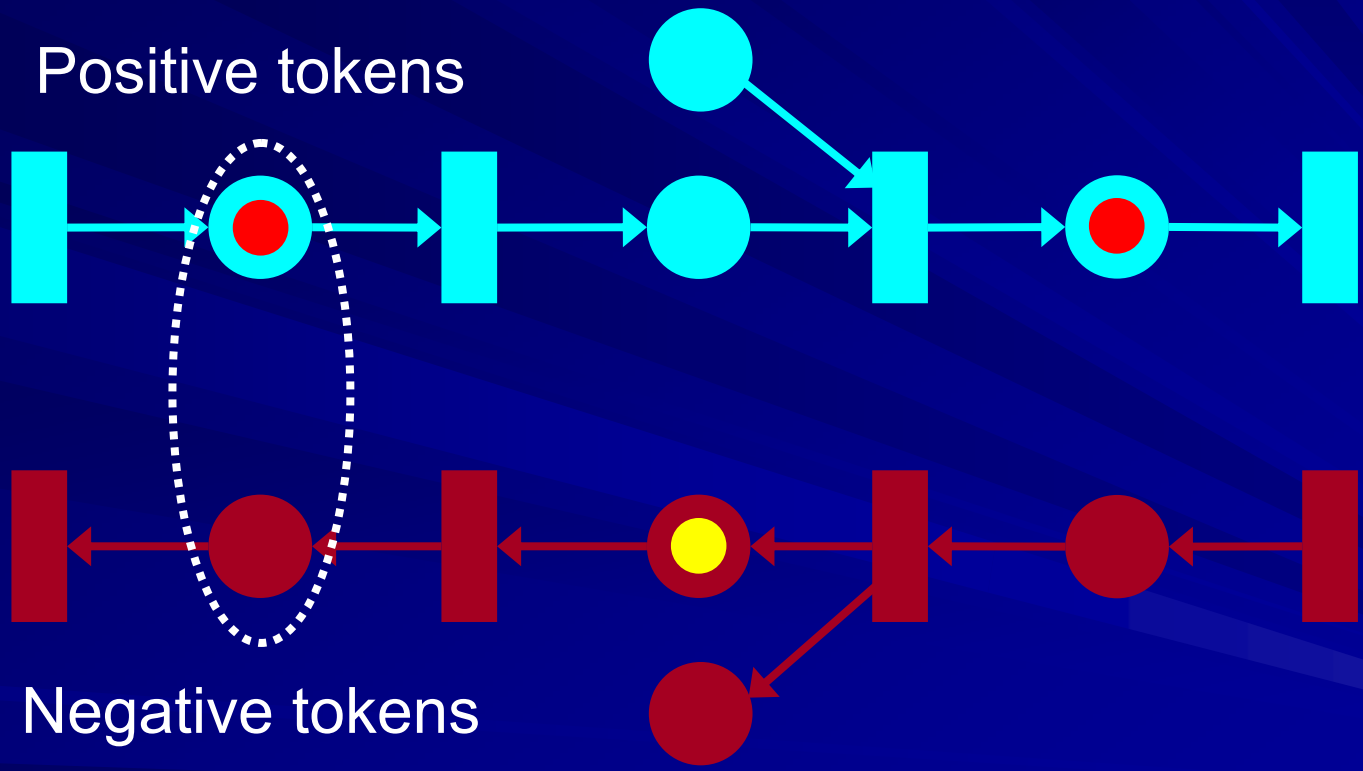
- DMGs have properties similar to regular MGs

Implementing early enabling

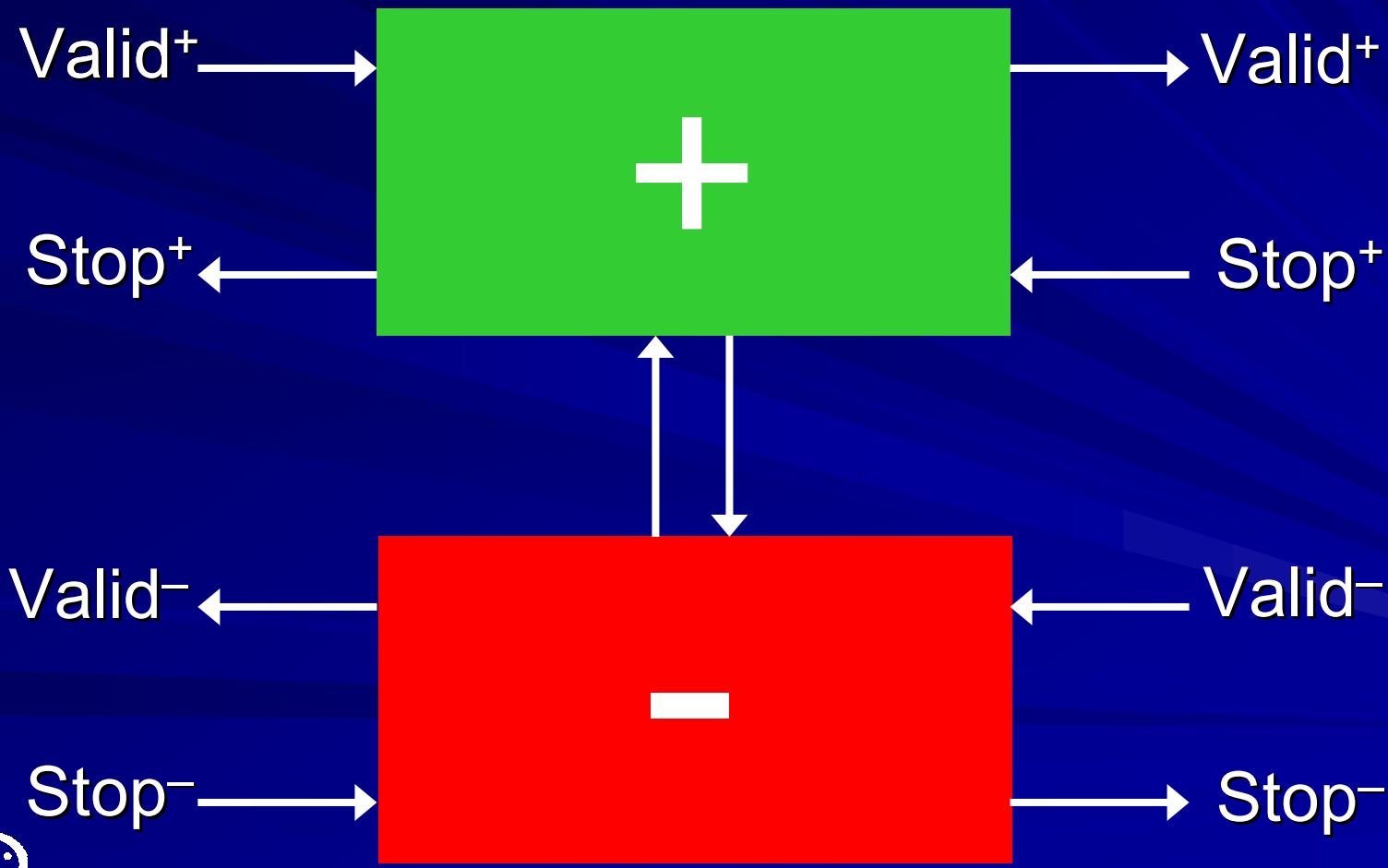
How to implement anti-tokens ?



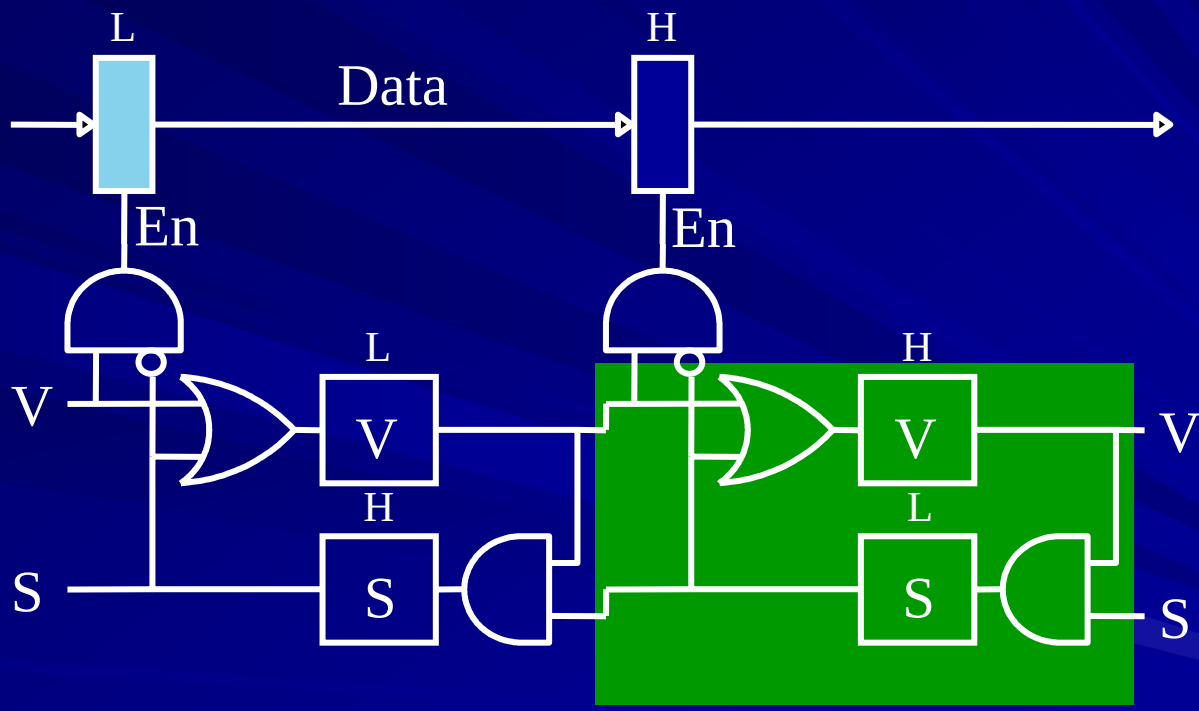
How to implement anti-tokens ?



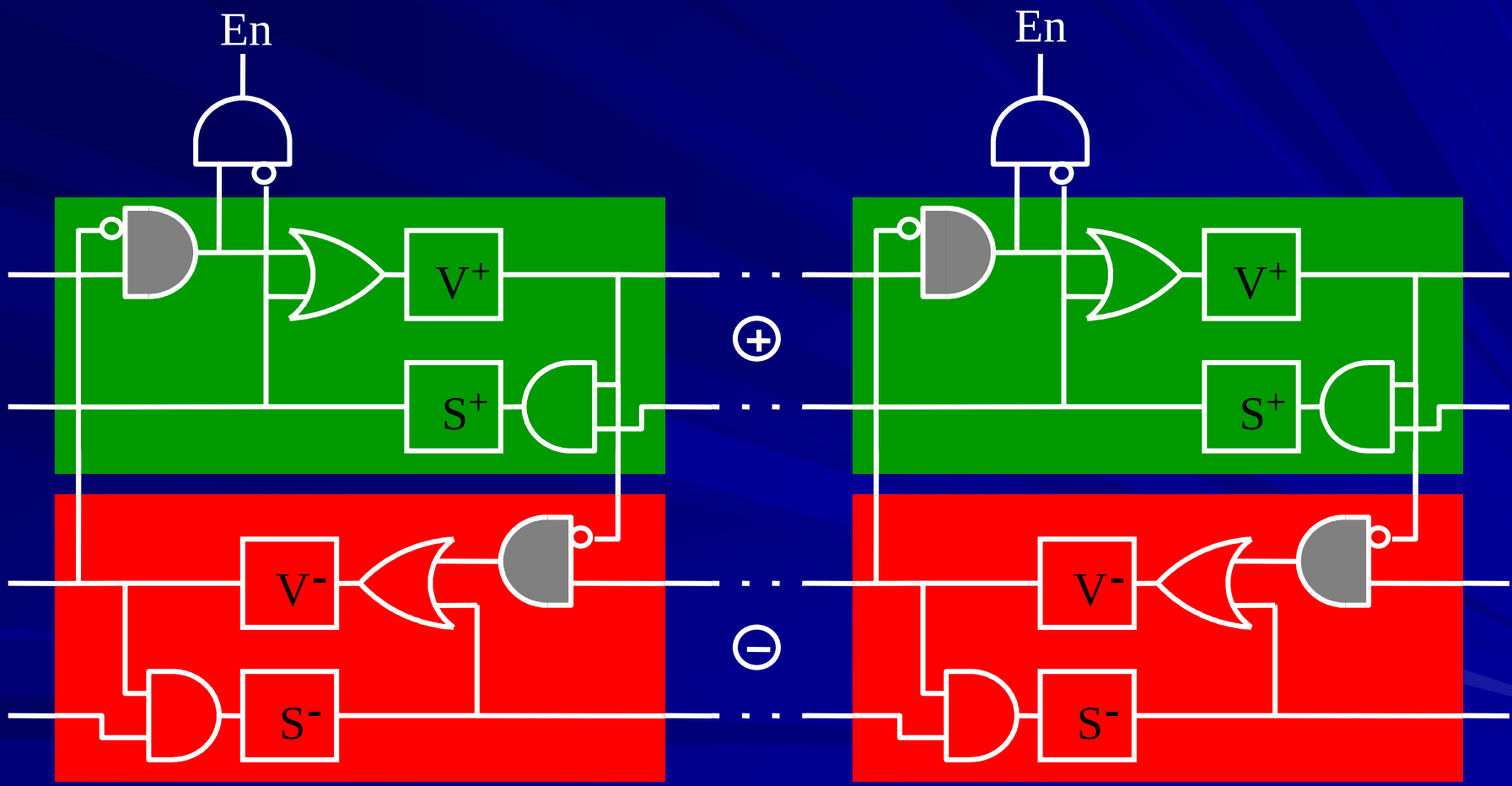
How to implement anti-tokens ?



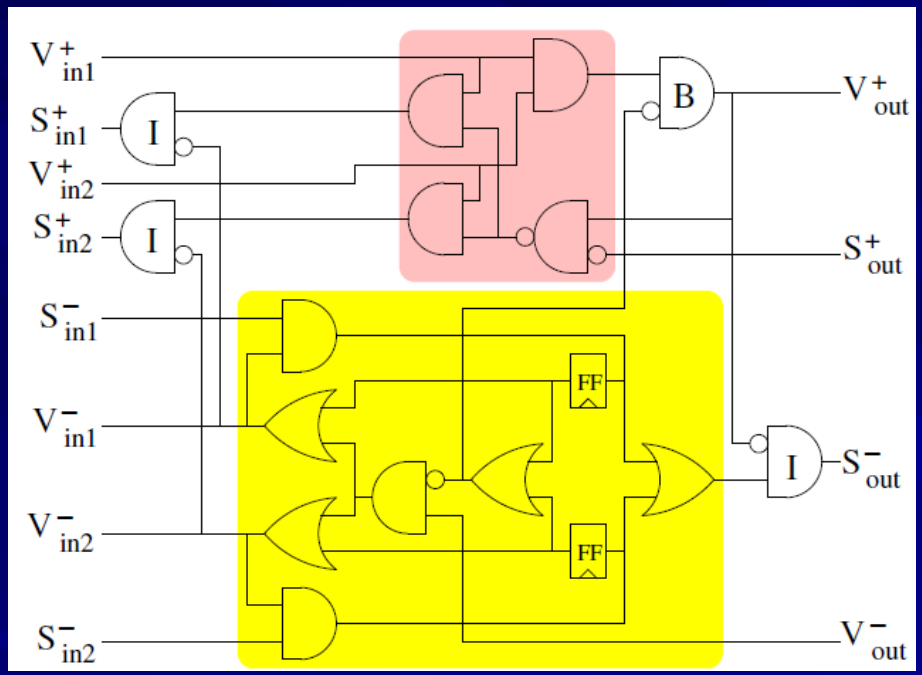
Controller for elastic buffer

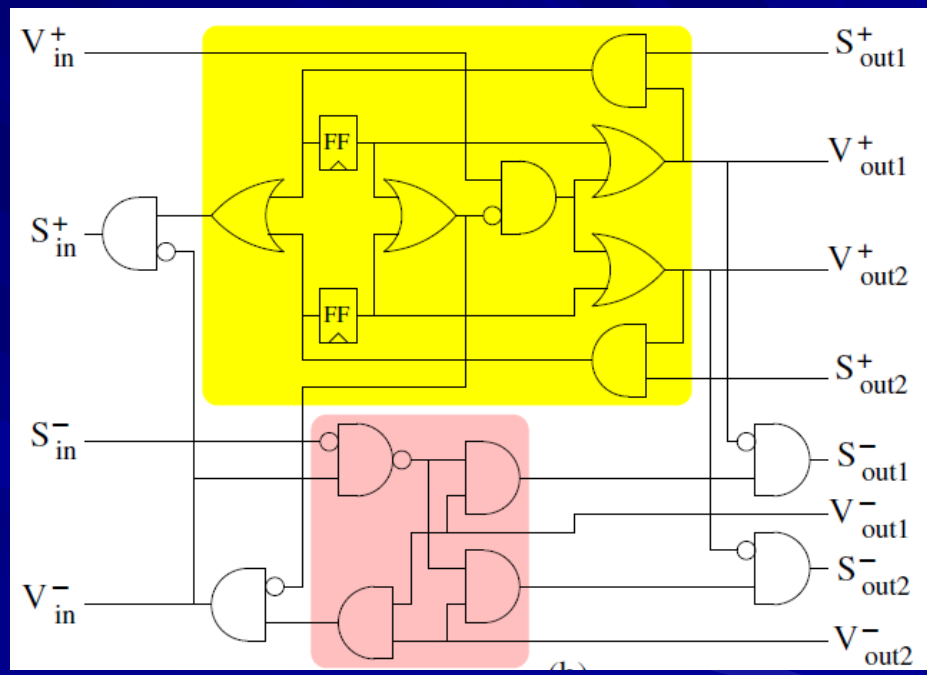


Dual controller for elastic buffer

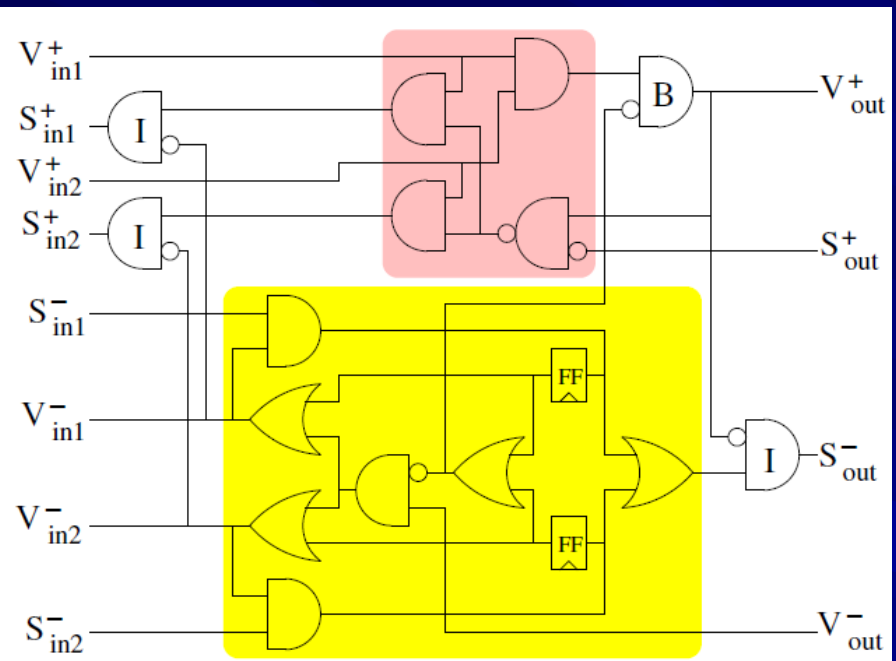


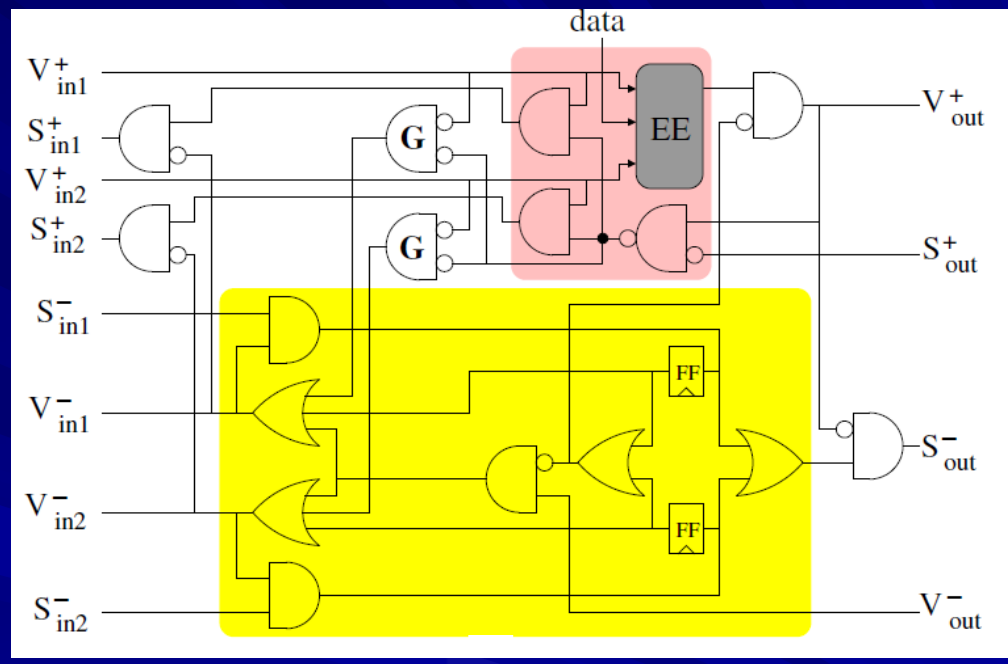
Dual Join and Fork





Join with early evaluation





Condition on Early Evaluation Function

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

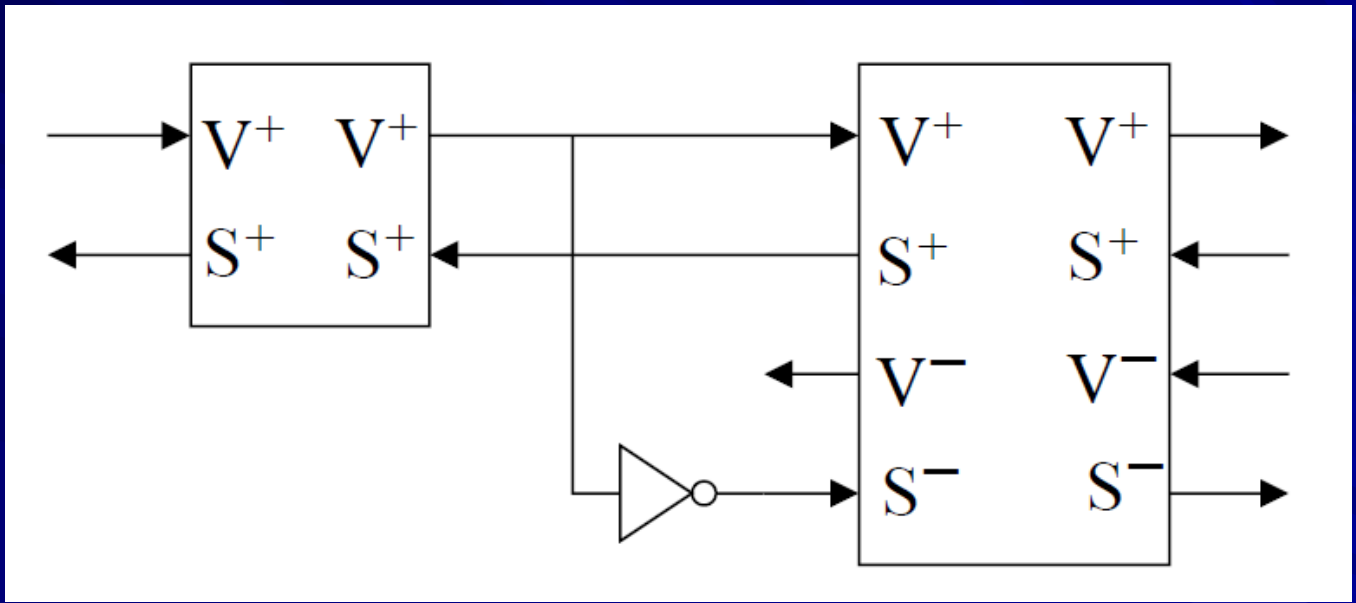
Formally: EE is positive unate with respect to data input

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

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

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

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

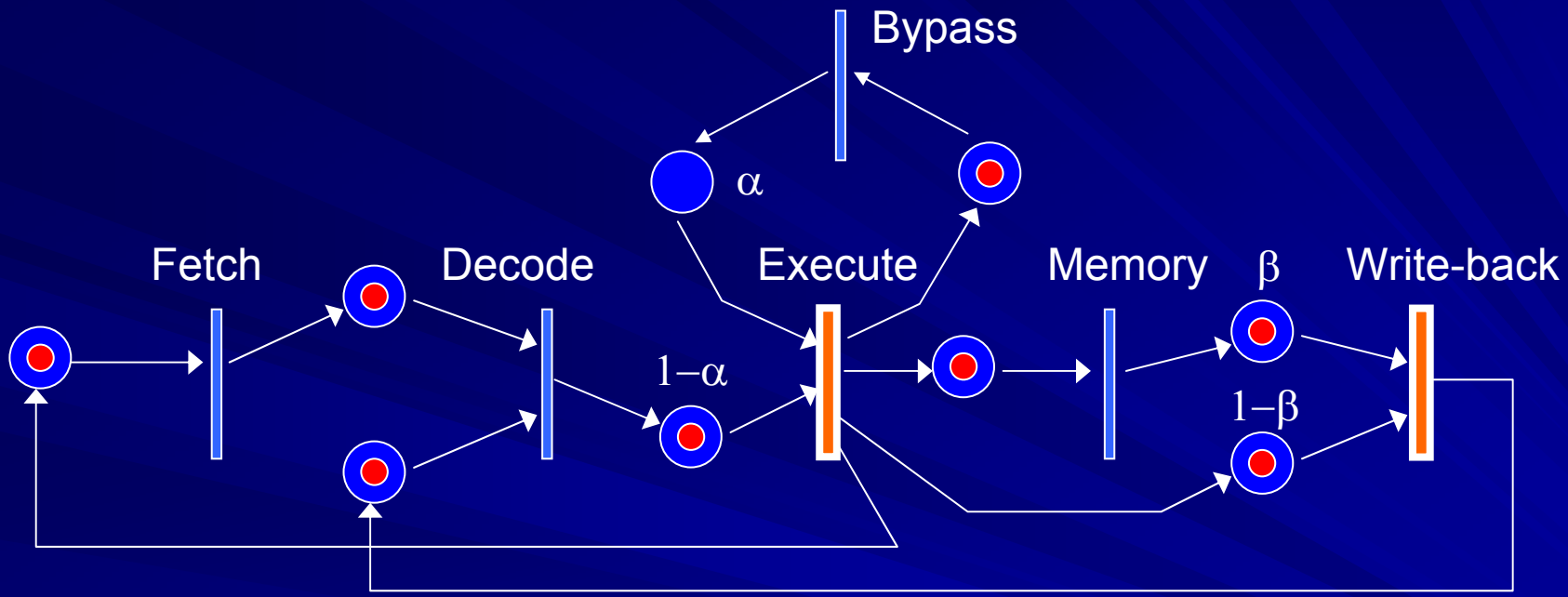
$$\mathtt{AG}\,\mathtt{AF}\,((V^+ \wedge \overline{S^+}) \vee (V^- \wedge \overline{S^-})) \qquad (\text{Liveness})$$

Invariants: mutually exclusive

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

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

DLX processor model with slow bypass



Throughput: *Th = operations / cycle*

Late evaluation

Th=0.5

Applying early evaluation on "Execution" and "Write-back"

Th = 0.7 (α=0.3; β=0.3)

Conclusions

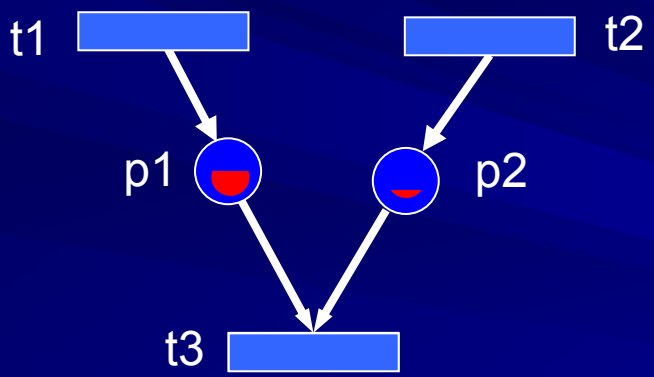
- Early evaluation can increase performance beyond the min cycle ratio
- The duality between positive and negative tokens suggests a clean and effective implementation
- Dual Marked Graphs is a formal model for analytical analysis and optimization methods

Performance analysis with early evaluation

(joint work with Jorge Júlvez)

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

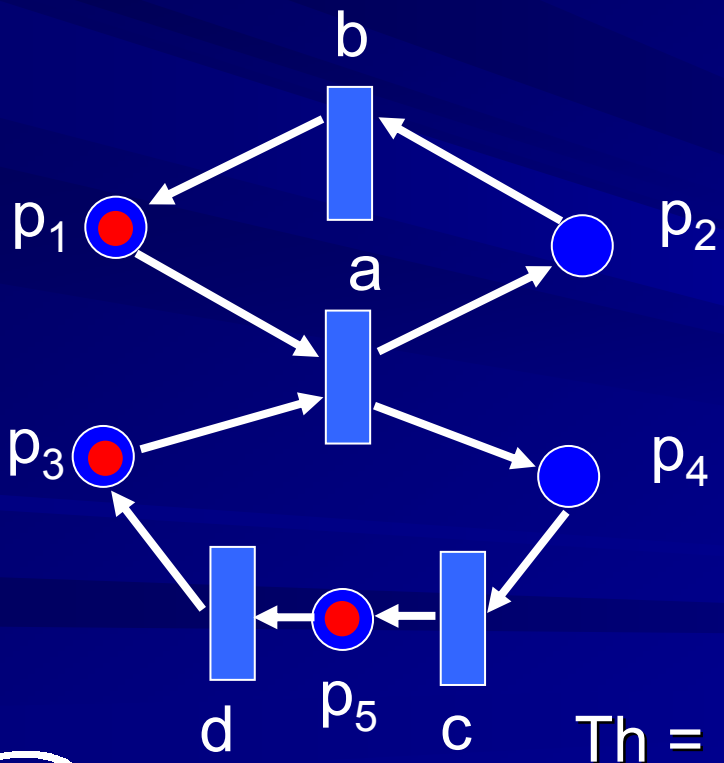
$$\overline{m}_p = \lim_{t\to\infty} \frac{1}{t} \int_0^t m_p(\tau) d\tau$$

Throughput

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

[Campos, Chiola, Silva 1991]

Revisit Performance Analysis of Marked Graphs



Th = 0.5

max th

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

reachability

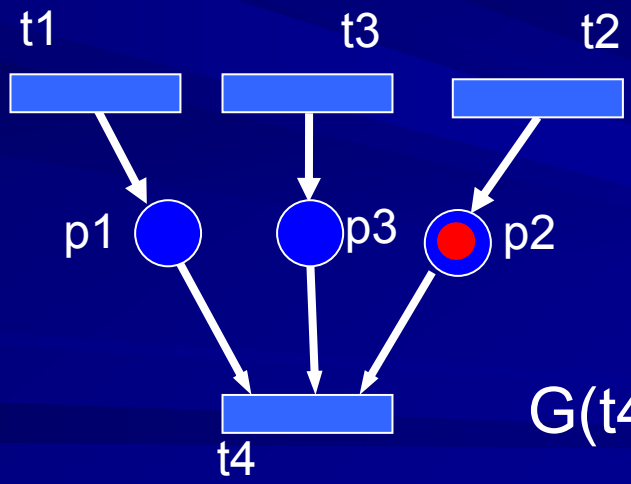
th constraints

$th \le \overline{m}_{p2}$ // transition b
$th \le \overline{m}_{p4}$ // transition c
$th \le \overline{m}_{p5}$ // transition d
$th \le \min(\overline{m}_{p1}, \overline{m}_{p3})$ // transition a

GMG = Multi-guarded Dual Marked Graph

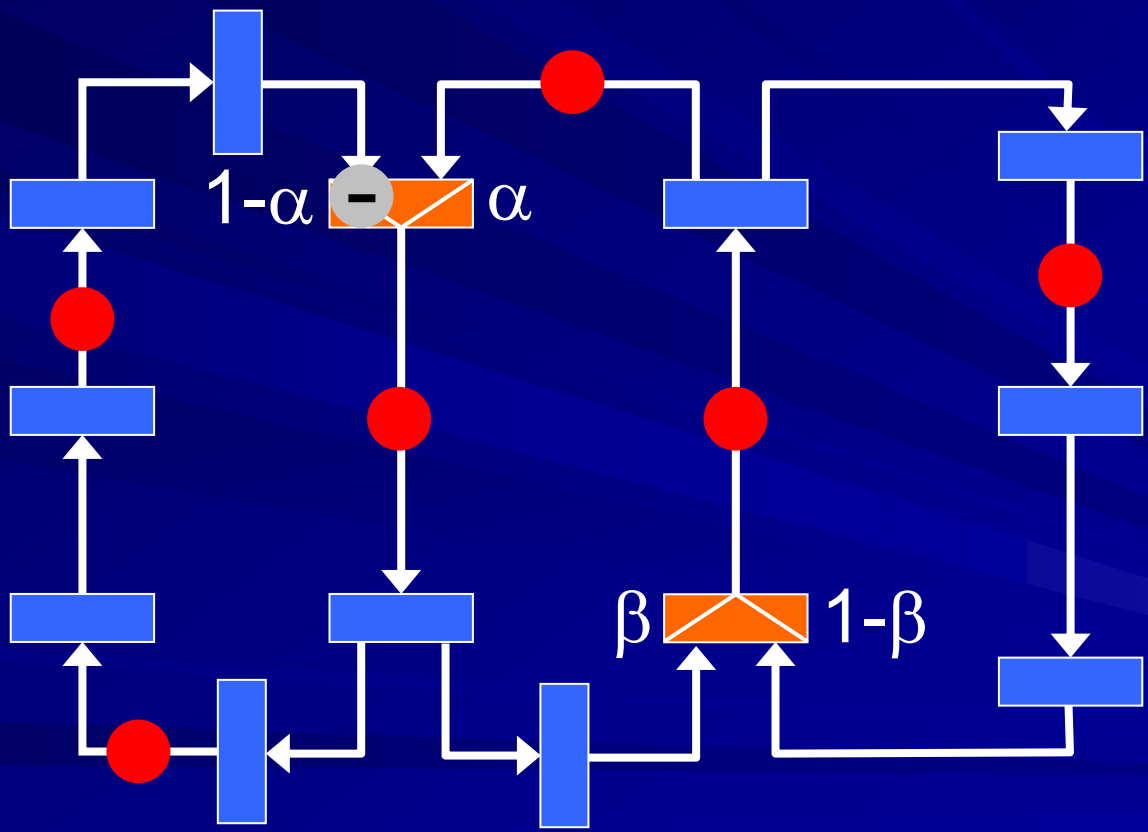
- Refinement of passive DMGs
- Every node has a set of guards
- Every guard is a set of input places (arcs)

Example:

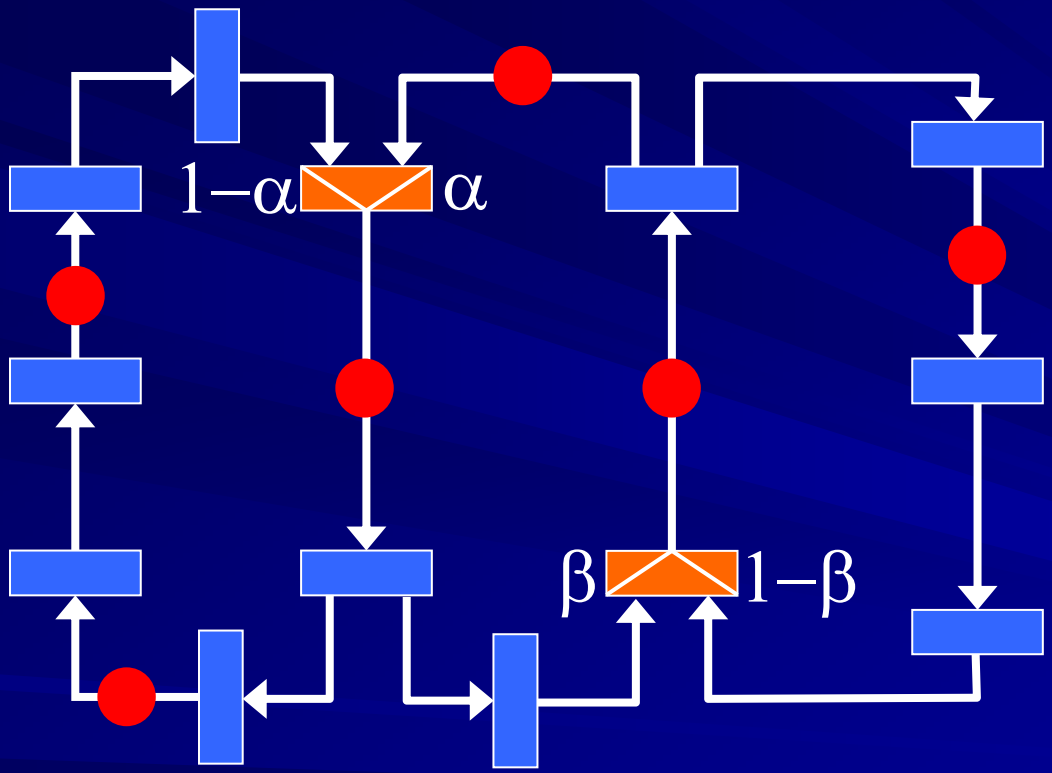


G(t4)={{p1,p3},{p2,p3}}

Early evaluation



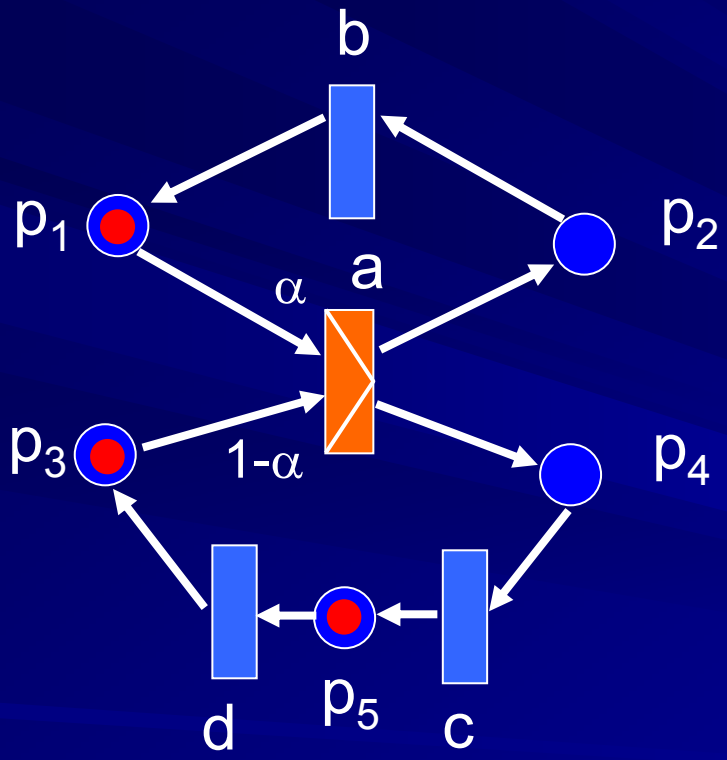
Early evaluation



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

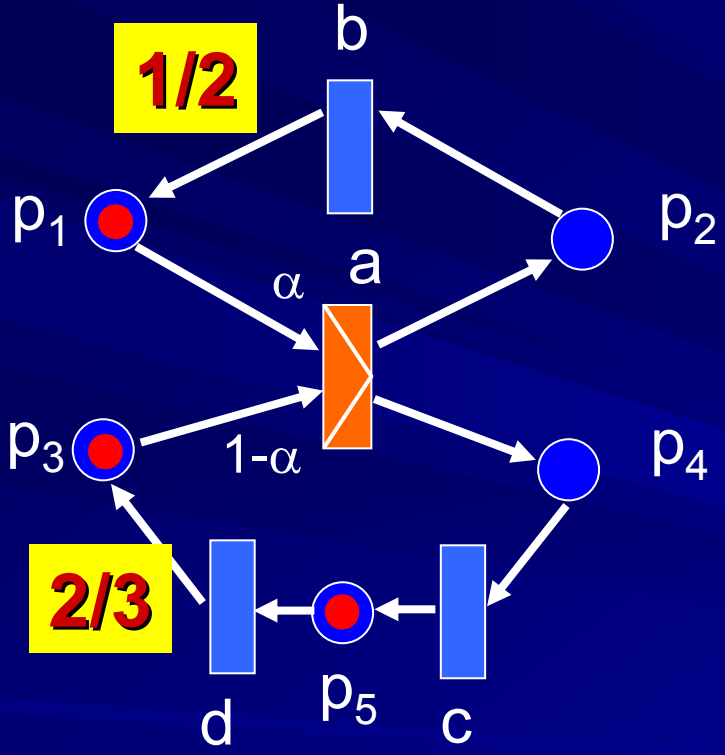
$\overline{m}_{p1} = 1 + t_b - t_a$
$\overline{m}_{p2} = 0 + t_a - t_b$
$\overline{m}_{p3} = 1 + t_d - t_a$
$\overline{m}_{p4} = 0 + t_a - t_c$
$\overline{m}_{p5} = 1 + t_c - t_d$

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

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

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

Averaging cycle throughput or cycle times does not work



Th = (2 - α) / (3 - α)

Averaging throughput of individual cycles

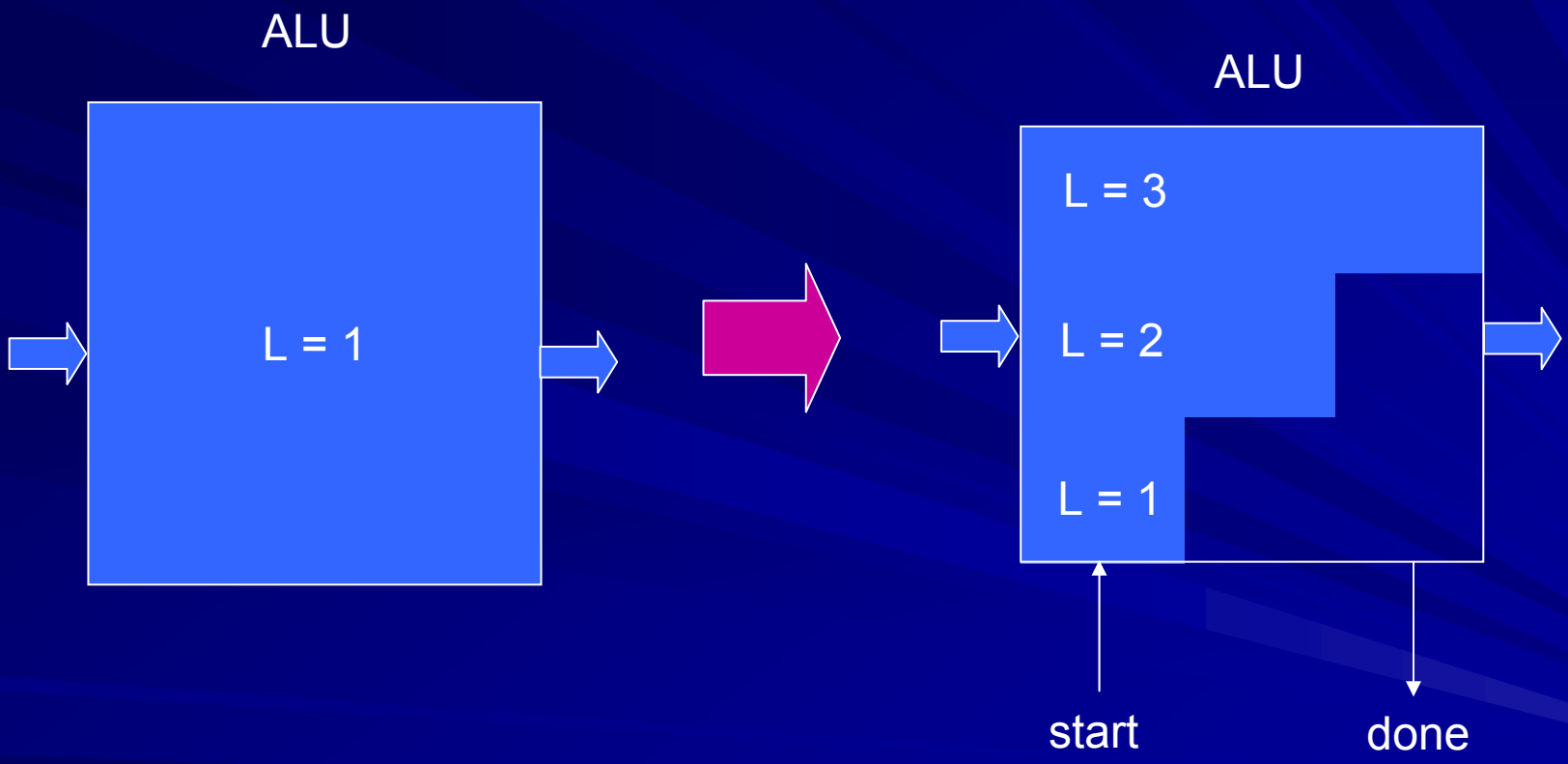
Th' = α 1/2 + (1- α) 2/3 = (4 - α) / 6

Averaging effective cycle times of individual cycles

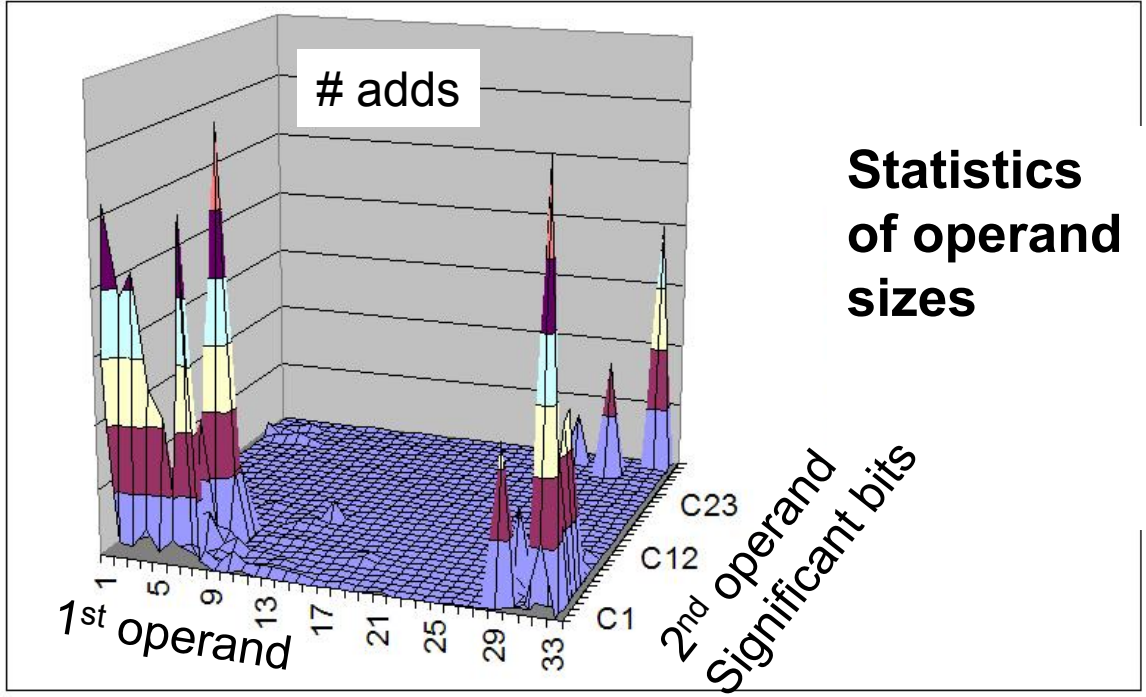
1/Th" = 2α + (1- α) 3/2 = (3 + α) / 2
Th" = 2/(3+ α)

What can we do with synchronous elastic systems?

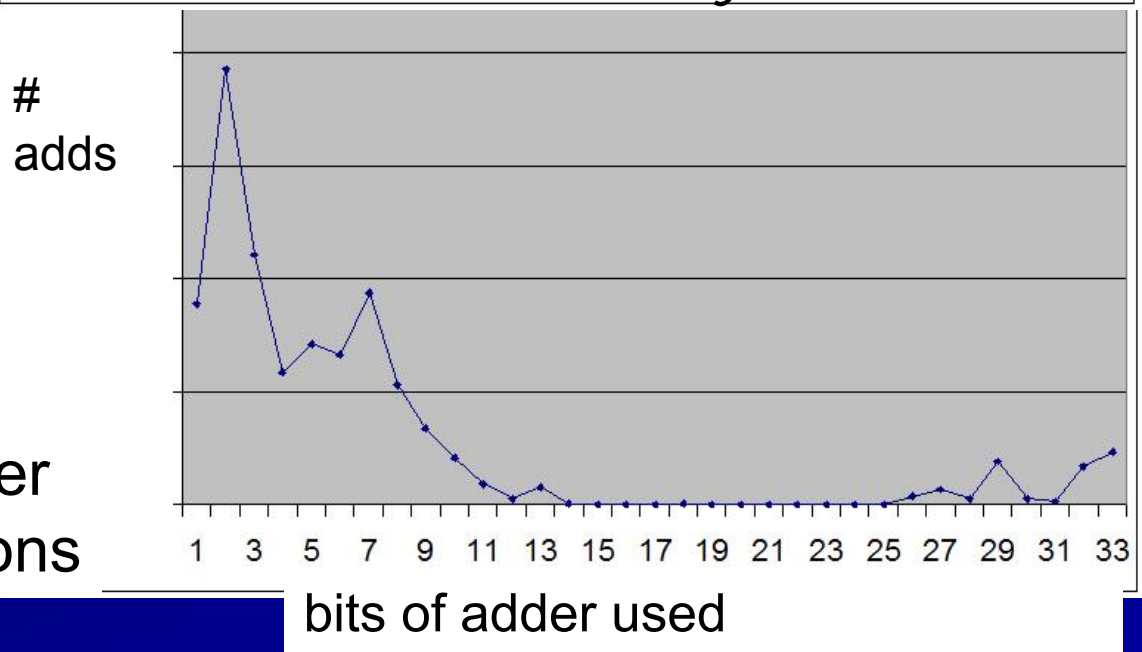
Variable latency units



Benchmark "Patricia" from Media Bench



Statistics of operand sizes

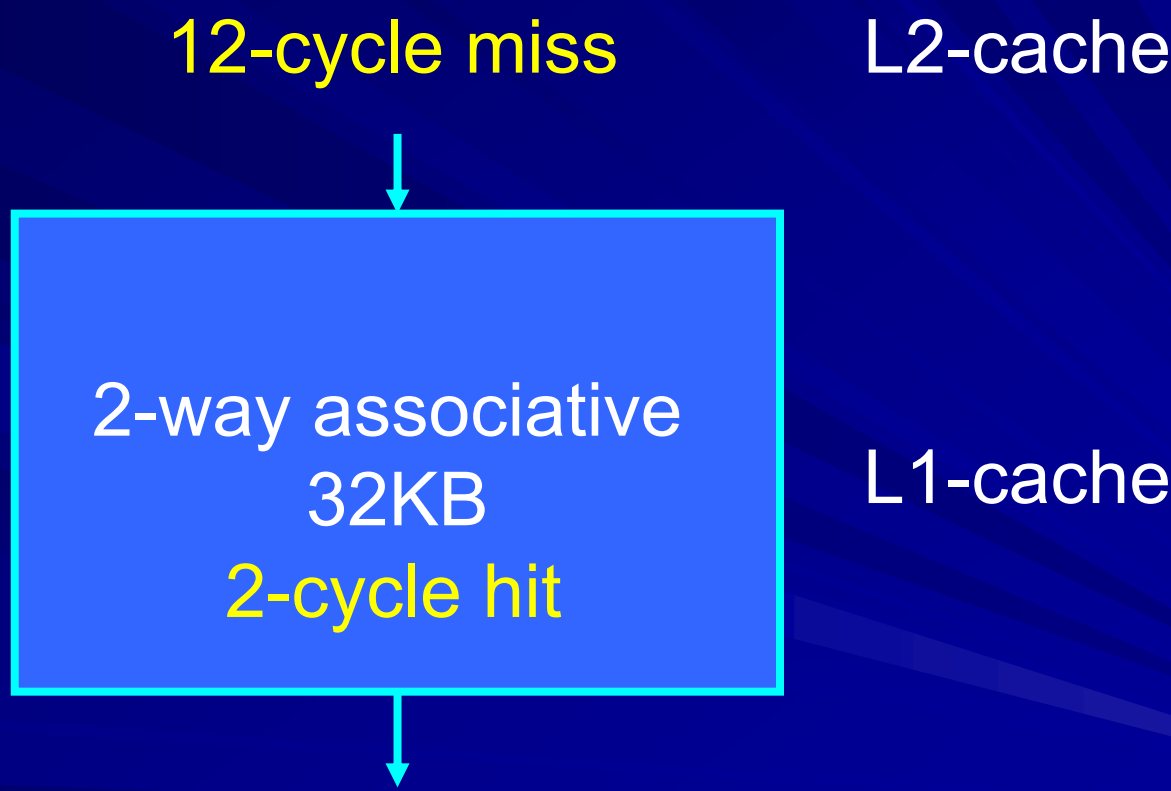


12 bits of an adder do 95% of additions

Power-delay for an adder

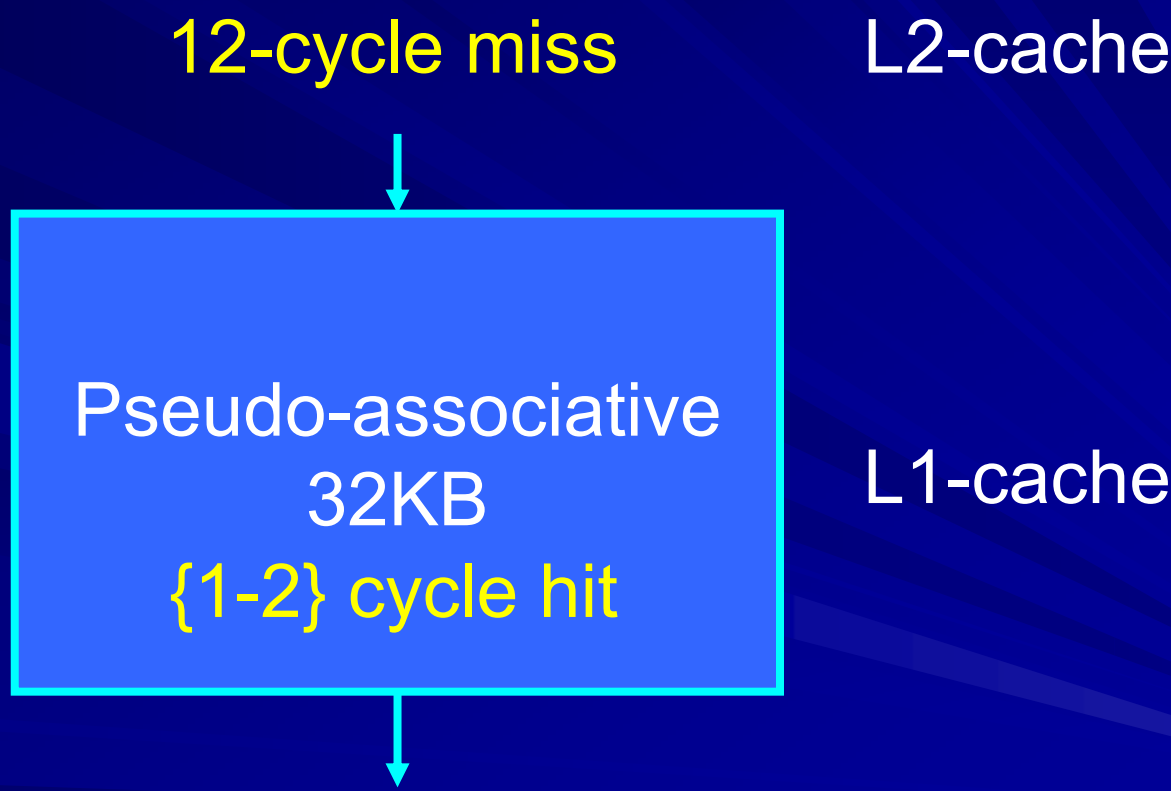
Compare
64 bits
VLA and
prefix adder

Variable-latency cache hits



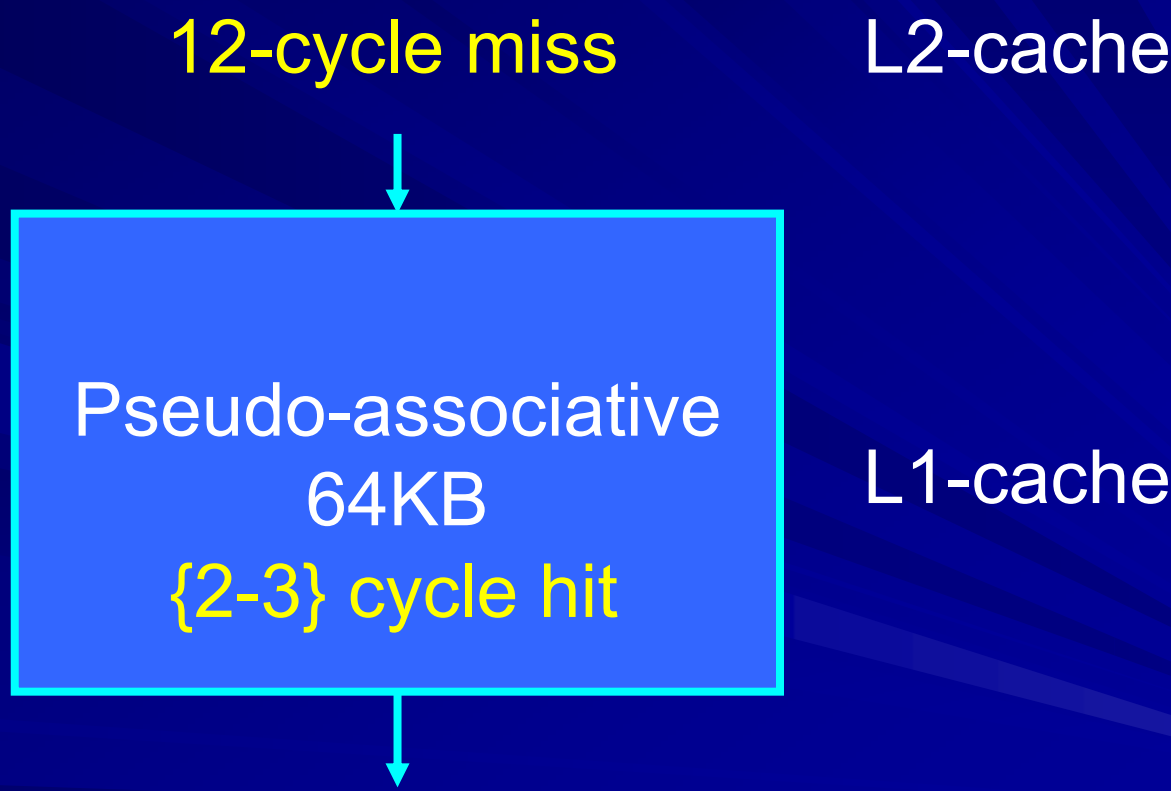
suggested by Joel Emer for ASIM experiment

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

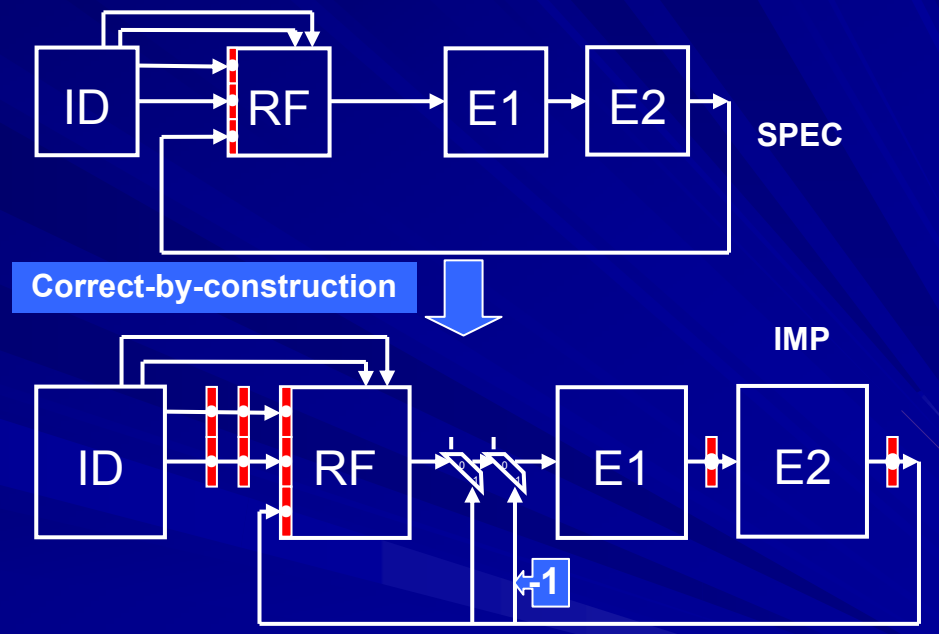


Sequential access: if hit in first access L = 1, if not – L=2
Trade-off: faster, or larger, or less power cache

Correct-by-construction pipelining

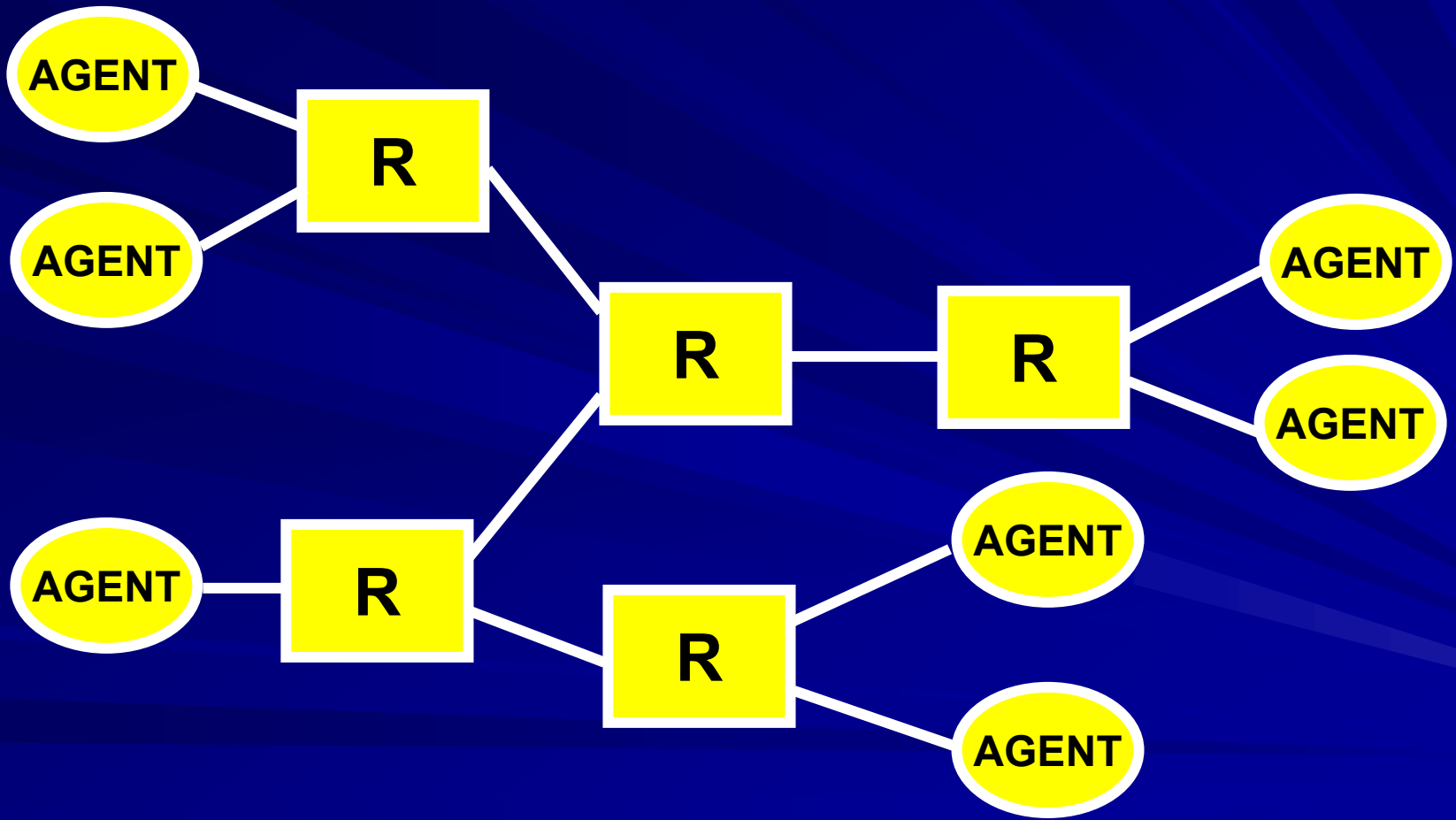
Transforms:

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



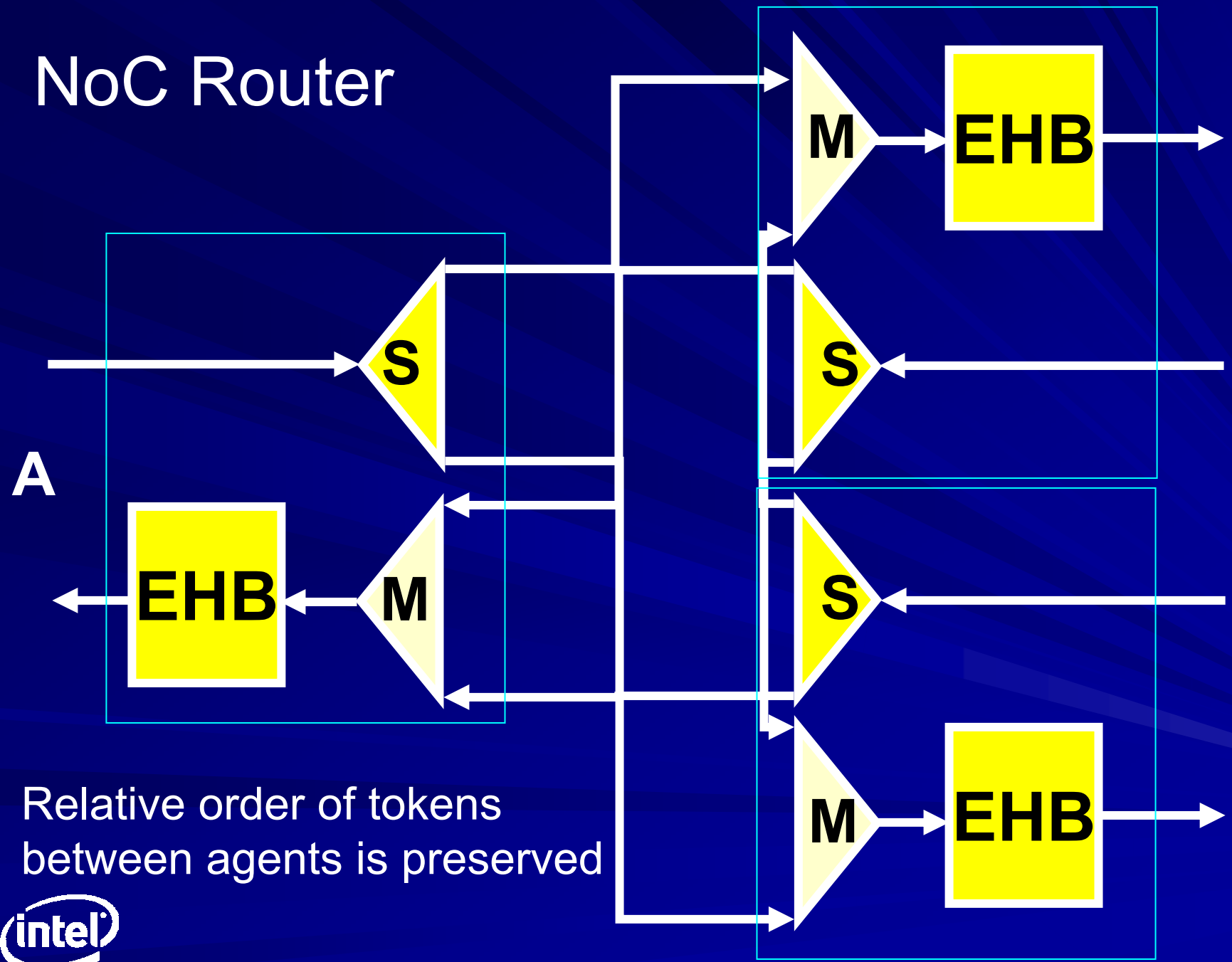
[Joint work with Timothy Kam and Marc Galceran]

Tree topology NoC



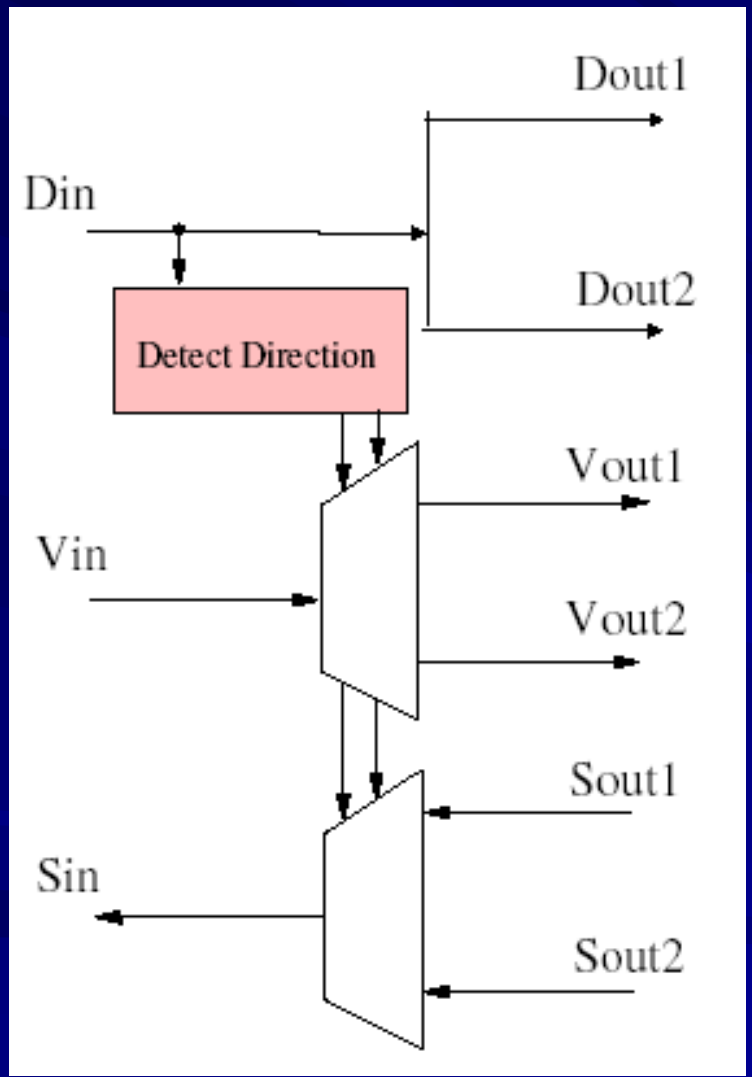
Router node interface

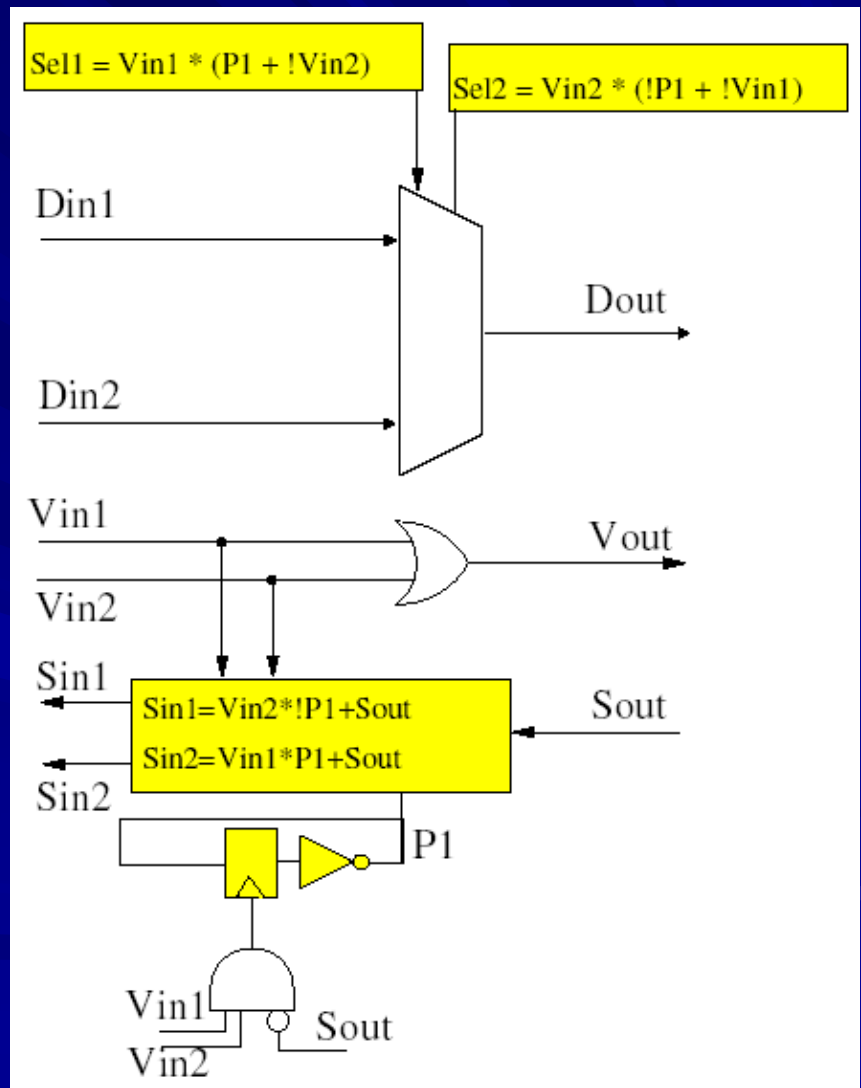
NoC Router



Relative order of tokens between agents is preserved

Switch and Merge





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

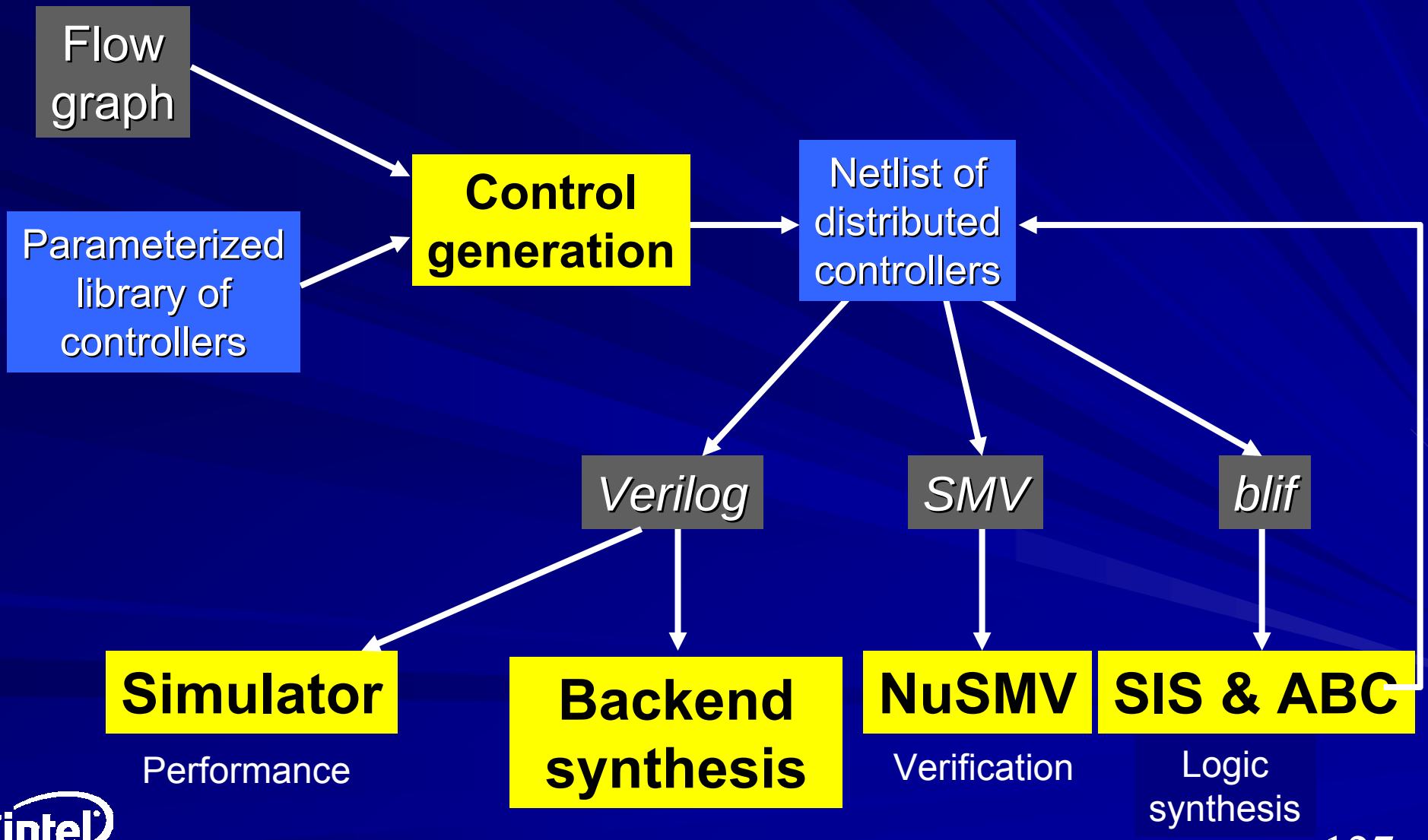
- Theory for early evaluation and negative delays is more challenging
  - Sketch of a theory, but no fully satisfactory compositional properties found yet
  - Verification done on concrete systems and controllers

Summary

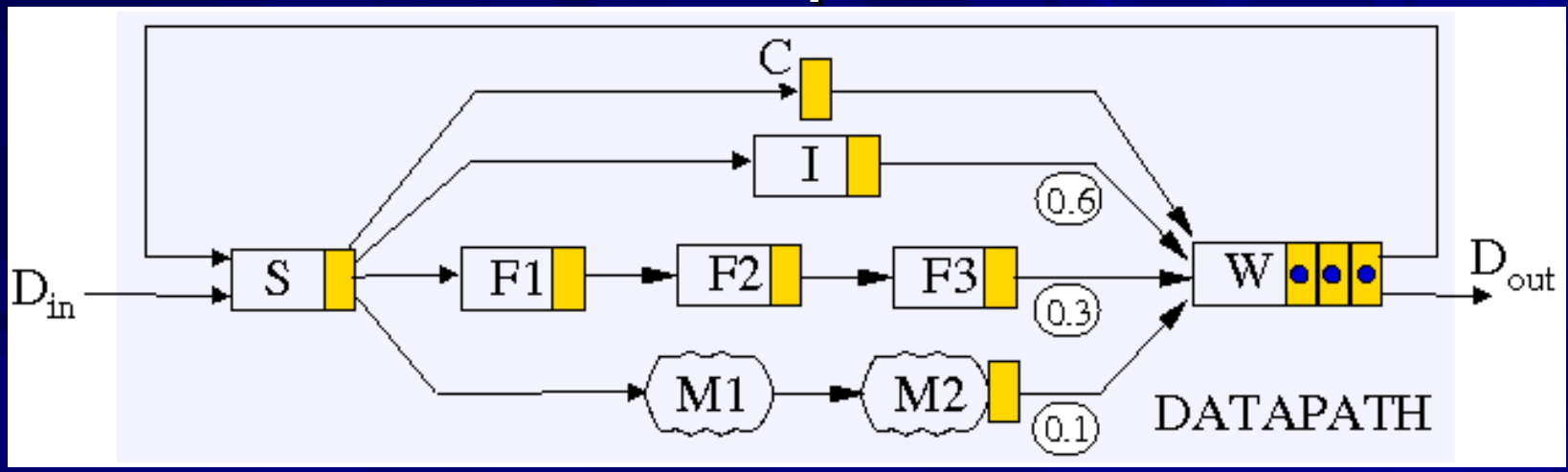
- SELF gives a low cost implementation of elastic machines
- Functionality is correct when latencies change
- New micro-architectural opportunities
- Compositional theory proving correctness
- Early evaluation - mechanism for performance and power optimization
- Optimization methods (that we did not discuss): Retiming and recycling, buffer optimization and pipelining
- Applications to design of NoC link layer

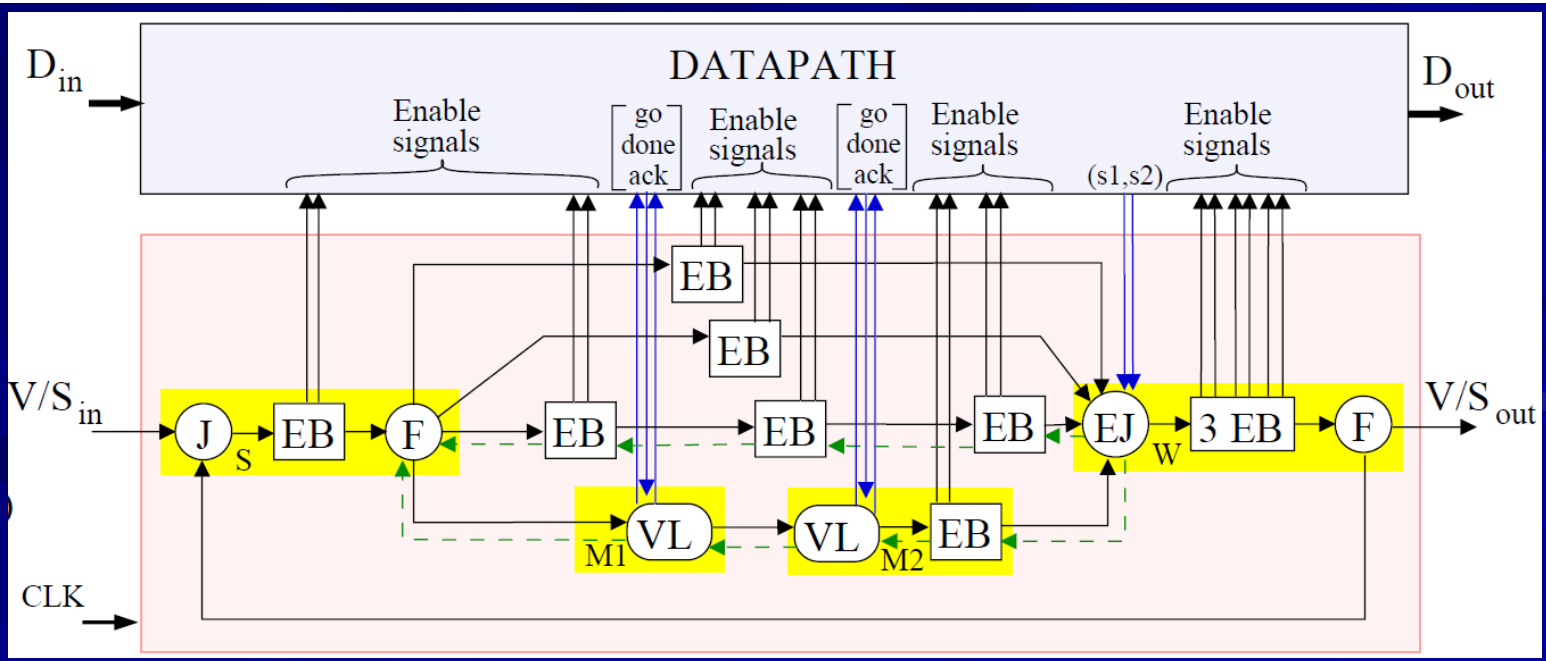
See reference list for *some* relevant publications

SELF compiler

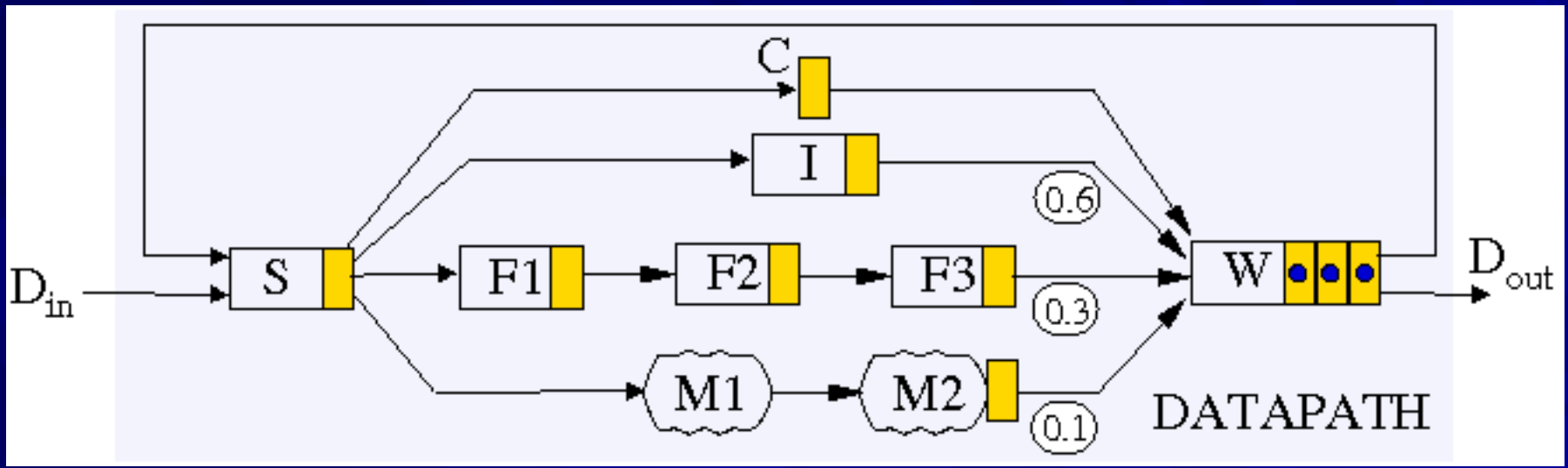


Example


C
I
0.6
D_{in}
S
F1
F2
F3
W
D_{out}
0.3
M1
M2
0.1
DATAPATH


D_{in}
DATAPATH
D_{out}
Enable signals
go done ack
Enable signals
go done ack
Enable signals
(s1,s2)
Enable signals
EB
EB
V/S_{in}
J
S
EB
F
EB
EB
EB
EJ
W
3 EB
F
V/S_{out}
$\overline{M1}$
VL
VL
M2
EB
CLK

Example



Evaluation	Throughput
No early evaluation	0.277
Passive anti-tokens M2 → W	0.280
Passive anti-tokens F3 → W	0.387
Active anti-tokens	0.400