The Design of Low-Latency Interfaces for Mixed-Timing Systems

Tiberiu Chelcea and Steven M. Nowick

Department of Computer Science
Columbia University

Keynote Invited Talk: "GALS" Session
IEEE Workshop on Complexity-Effective Design (ISCA-02)
May 26, 2002

Trends and Challenges

Trends in Chip Design: next decade

- *"Semiconductor Industry Association (SIA) Roadmap"* (97-8)

Unprecedented Challenges:

- complexity and scale (= size of systems)
- clock speeds
- power management
- reusability & scalability
- "time-to-market"

Design becoming unmanageable using a centralized single clock (synchronous) approach….

Trends and Challenges (cont.)

1. Clock Rate:

- *1980: several MegaHertz*
- *2001: ~750 MegaHertz - 1+ GigaHertz*
- *2004: several GigaHertz*

Design Challenge:

- *"clock skew":* clock must be near-simultaneous across entire chip

Trends and Challenges (cont.)

2. Chip Size and Density:

Total #Transistors per Chip: *60-80% increase/year*

- *~1970: 4 thousand* (Intel 4004)
- *today: 10-100+ million*
- *2004 and beyond: 100 million-1 billion*

Design Challenges:

- system complexity, design time, clock distribution
- *clock will not reach across chip in 1 cycle*

Trends and Challenges (cont.)

3. Power Consumption

- Low power: ever-increasing demand
  - consumer electronics: battery-powered
  - high-end processors: avoid expensive fans, packaging

Design Challenge:

- *clock inherently consumes power continuously*
- "power-down" techniques: only partly effective

Trends and Challenges (cont.)

4. Time-to-Market, Design Re-Use, Scalability

Increasing pressure for faster *"time-to-market".* Need:

- reusable components: "plug-and-play" design
- scalable design: easy system upgrades

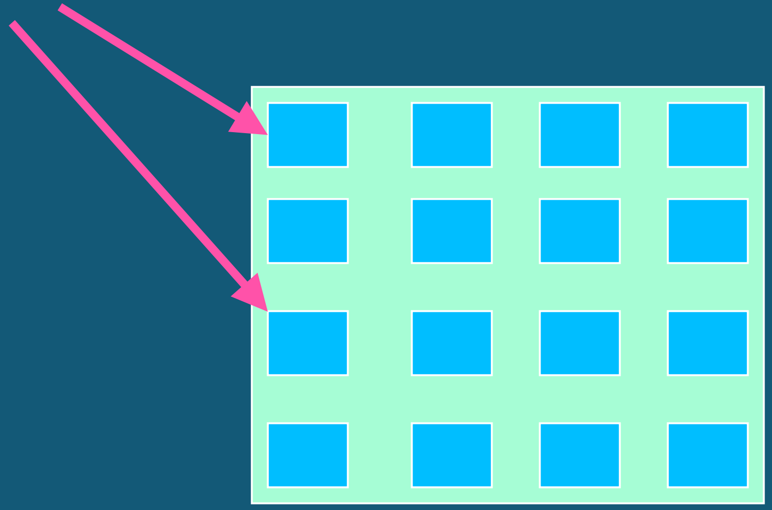
Design Challenge: mismatch w/ central fixed-rate clock

Trends and Challenges (cont.)

5. Future Trends: "Mixed Timing" Domains

Chips themselves becoming *distributed systems*....

- contain many sub-regions, *operating at different speeds:*



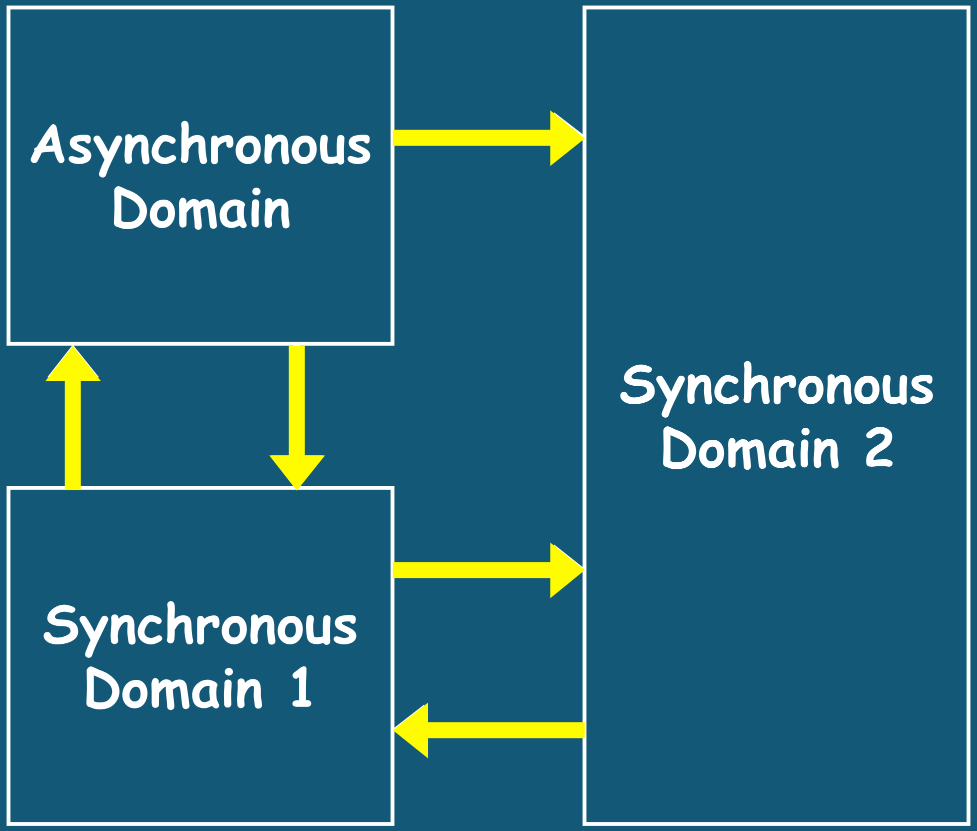
Design Challenge: breakdown of single centralized clock control

Introduction

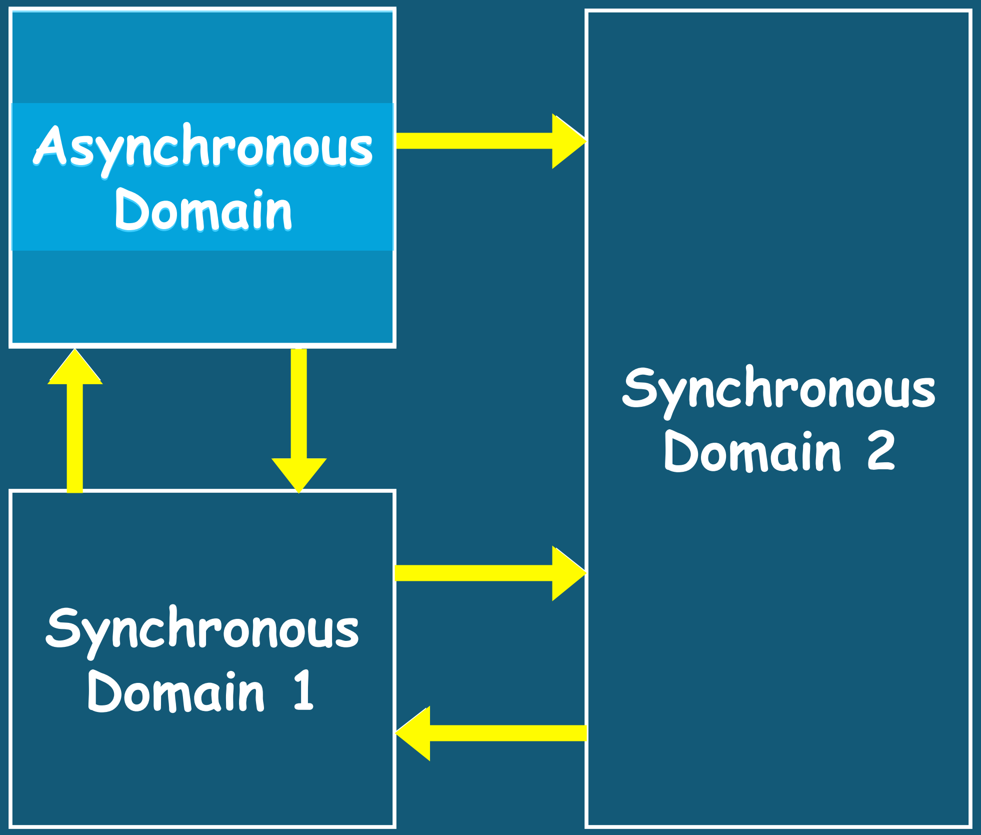
Example: System-on-a-Chip (SoC) Design

- Building entire large-scale system on a single chip
- Benefit: Higher-level of integration
  - Improved performance, cost, area
- Challenges:
  - Mixed-timing: moving to multiple timing domains
  - Performance degradation: synchronization overhead
  - Complexity, scale, integration
  - Designing & incorporating of *asynchronous subsystems*

Future Chips



Research Areas



Goal #1: interface mixed-timing domains with low latency

Goal #2: synthesis + optimization of asynchronous systems

Summary: Key Challenges in System Design

Two key issues not yet completely addressed:

1. Communication between mixed-timing domains:
   - Goals: performance and scalability
2. Synthesis of large-scale asynchronous systems:
   - Goals: develop powerful optimizing CAD tools, facilitating "design-space exploration"

Asynchronous Design: Motivation

Need for large-scale asynchronous systems:

- Future chips: likely a mix of async and sync domains

Asynchronous Systems: offer a number of advantages

GALS: "globally-asynchronous, locally-synchronous"

- Hybrid style: introduced by Chapiro [84]
  - synchronous "processing elements" ("satellites")
  - asynchronous communication
- Recent interest: "Communication-Based Design"
  - UC Berkeley/Stanford: W. Dally, K. Keutzer, A. Sangiovanni
  - *orthogonalization of concerns: function* vs. *communication*

Asynchronous Design: Potential Advantages

- Modularity:
  - Interface easily with sync domains & environment
- Reusability and scalability:
  - Handle wide range of interface speeds ⇒ reuse
  - Scalability: easily add new subsystems
- Average-case performance:
  - Intel RAPPID instruction-length decoder: 3-4x faster than sync design
  - Differential equation solver: 1.5x faster than sync design
- Lower power consumption:
  - Avoids clock distribution power
  - Provides automatic "clock gating" … at *arbitrary granularity*
  - Digital hearing aid chip: 4-5.5x less power
- Low electromagnetic interference (EMI): no regular clock spikes
  - Philips, commercial 80c51 microcontrollers: in cell phones, pagers

Industrial interest: Intel, Sun, IBM, Philips, Theseus, Fulcrum

Related Work #1: Interfacing in Single Clock Domain

Handling Timing Discrepancies...:

Clock Skew:

- STARI Chip [M. Greenstreet, ICCD-95]

  Use async buffer to smooth out discrepancies between *sender* and *receiver*

- Skew-Tolerant Domino [M. Horowitz]
- Clock-Skew Scheduling [E. Friedman]

- Long interconnect delays [Carloni99]: limited to single clock

Long Interconnect Delays:

- "Relay Stations" [Carloni, Sangiovanni-Vincentelli, DAC-00]

  Break up overlong wires by pipelining communication

Related Work: Interfacing Mixed-Timing Domains

Two common approaches…:

- Modify Receiver's Clock:
  - "stretchable" and "pausible" clocks
  - Chapiro84, Yun96, Bormann97, Sjogren/Myers97, Moore02
  - drawbacks:
    - Penalties in restarting clock
    - Does not support design reuse
- Use Synchronization Components:
  - data/control synchronization
  - Seitz80, Seizovic94, Intel97, Sarmenta95, Kol98
  - drawbacks: overheads in throughput, latency, area

Contribution: Mixed-Timing Interfaces

A complete family of mixed-timing FIFO's

Characteristics:

- Low-latency
- Modular and scalable:
  - Define interfaces for each combination of:
    - Synchronous or Asynchronous domains
  - Combine interfaces to design new async/sync FIFO's
- High throughput:
  - In steady state: no synchronization overhead, no failure probability
  - Enqueue/Dequeue data items: one/cycle
- Low area overheads

Also, solve issue of long interconnect delays between domains

Contribution: Mixed-Timing Interfaces

Publications

Latest Solution:

IEEE Transactions on VLSI Systems, vol. 12:8, pp. 857-853 (August 2004)

T. Chelcea and S.M. Nowick,
"Robust Interfaces for Mixed-Timing Systems"

IEEE/ACM Design Automation Conference (*DAC*, June 2001)

T. Chelcea and S.M. Nowick, "Robust Interfaces for Mixed-Timing Systems with Application to Latency-Insensitive Protocols"

Initial Solution:

IEEE Computer Society Workshop on VLSI (*WVLSI*, April 2000)

T. Chelcea and S.M. Nowick, "A Low-Latency FIFO for Mixed-Clock Systems"

See also:

A. Iyer and D. Marculescu, *ISCA-02*.

Outline

I. Mixed-Timing Interface Circuits

- Sync/Sync
- Async/Async
- Async/Sync

II. Handling Long Interconnect Delays

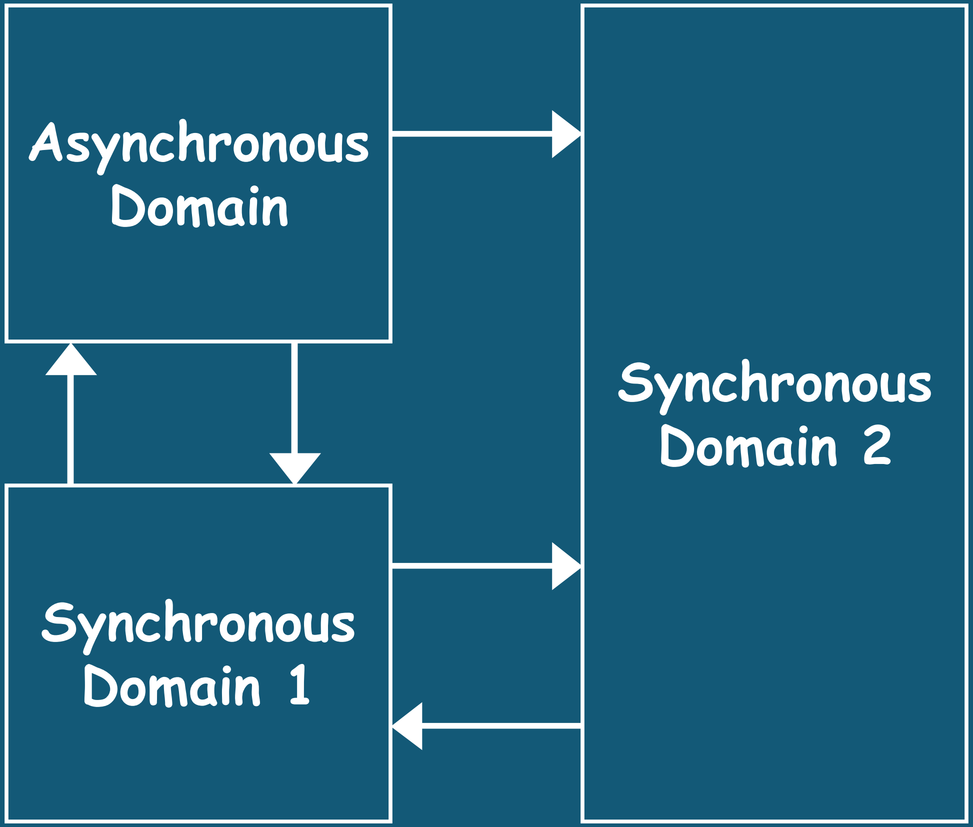
Experimental Results

Conclusions

Part I

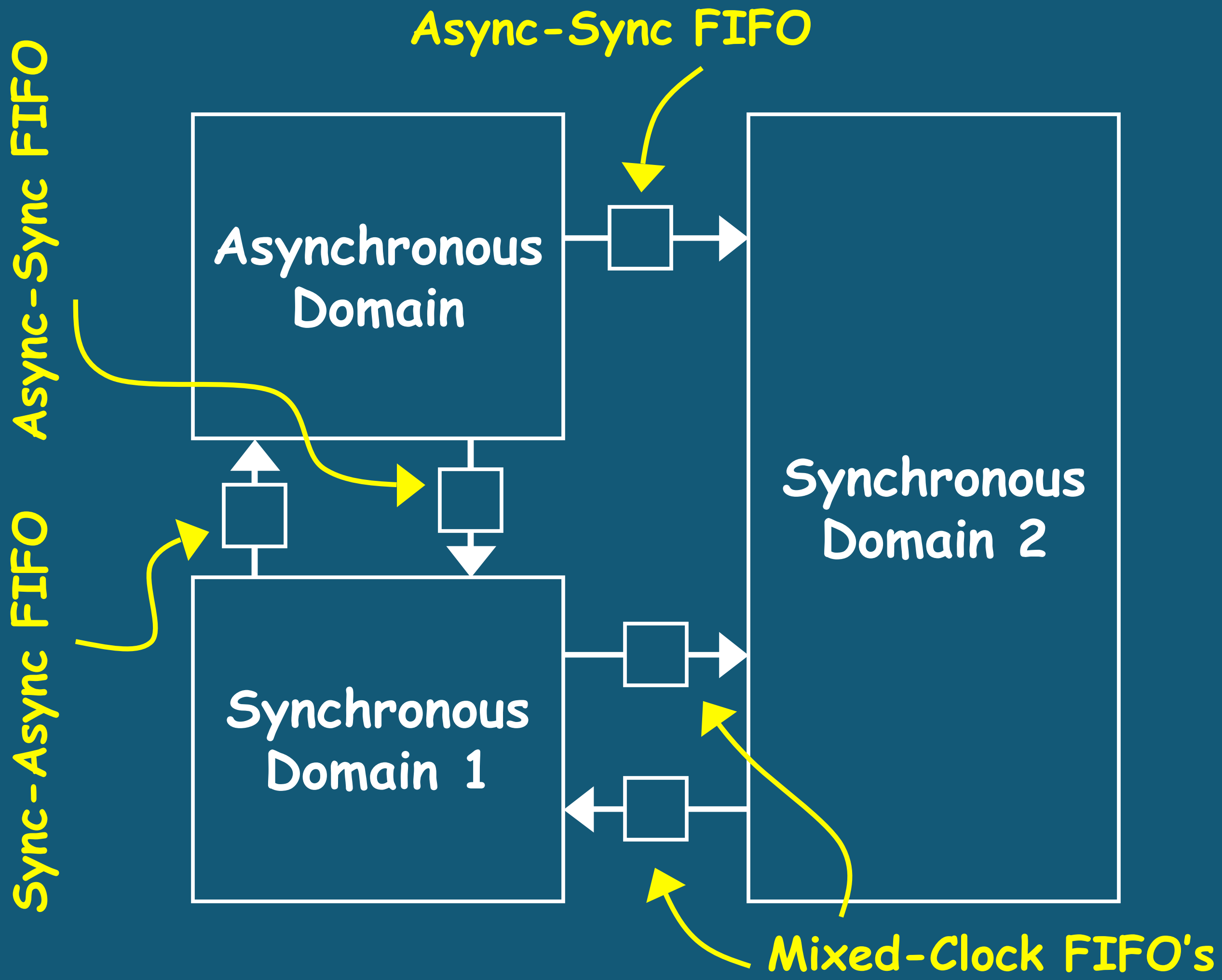
Mixed-Timing Interface Circuits

Mixed-Timing Interfaces: Overview



Problem: potential data synchronization errors

Mixed-Timing Interfaces: Overview

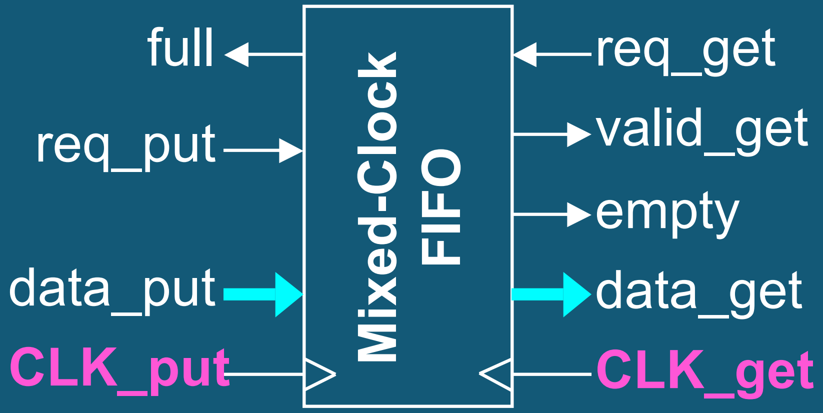


Problem: potential data synchronization errors

Solution: insert mixed-timing FIFO's ⇒ safe data transfer

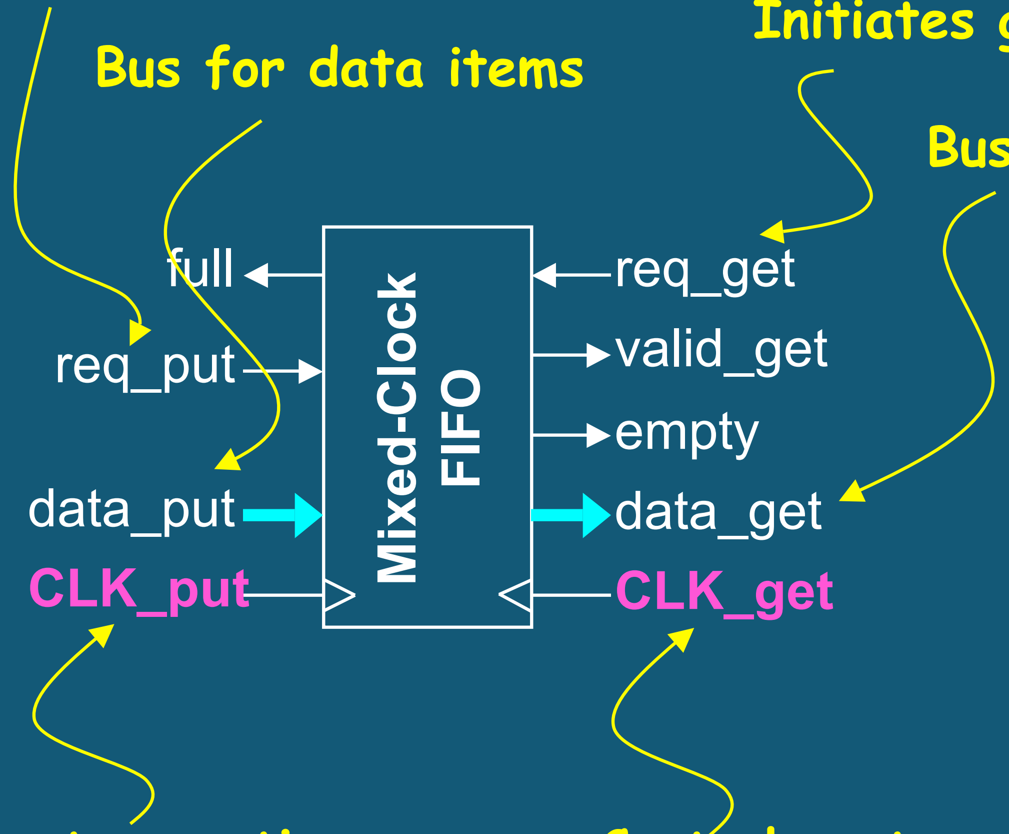
Mixed-Clock FIFO: Block Level

synchronous
put inteface



synchronous
get interface

Mixed-Clock FIFO: Block Level



Initiates put operations

Bus for data items

Initiates get operations

Bus for data items

synchronous **put** inteface

synchronous **get** interface

Controls put operations

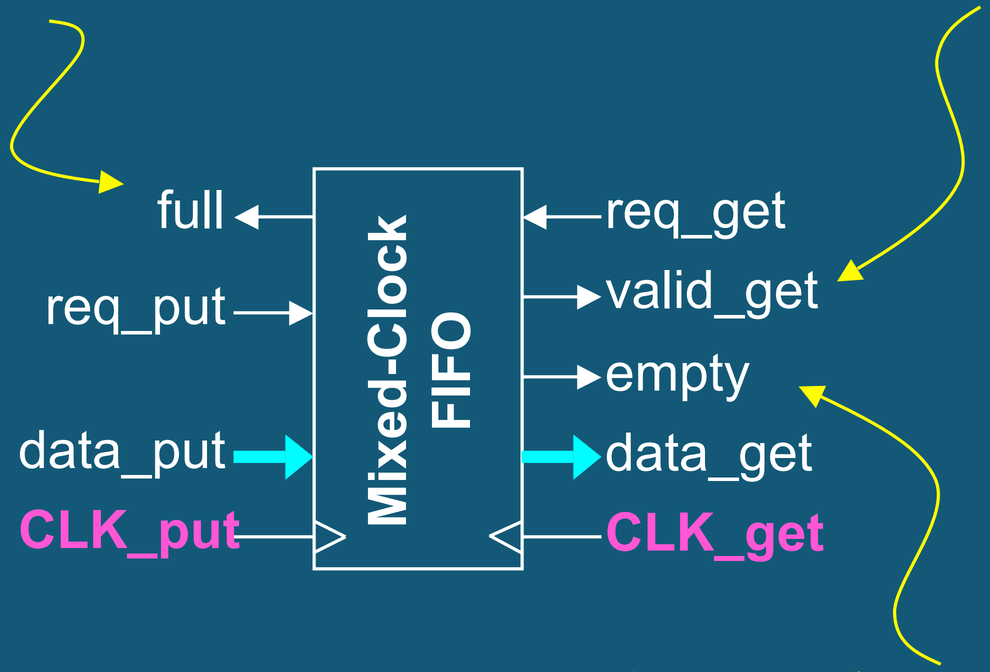
Controls get operations

Mixed-Clock FIFO: Block Level

Indicates when FIFO full

Indicates data items validity
(always 1 in this design)

synchronous
put inteface



Mixed-Clock FIFO

full

req_put

data_put

CLK_put

req_get

valid_get

empty

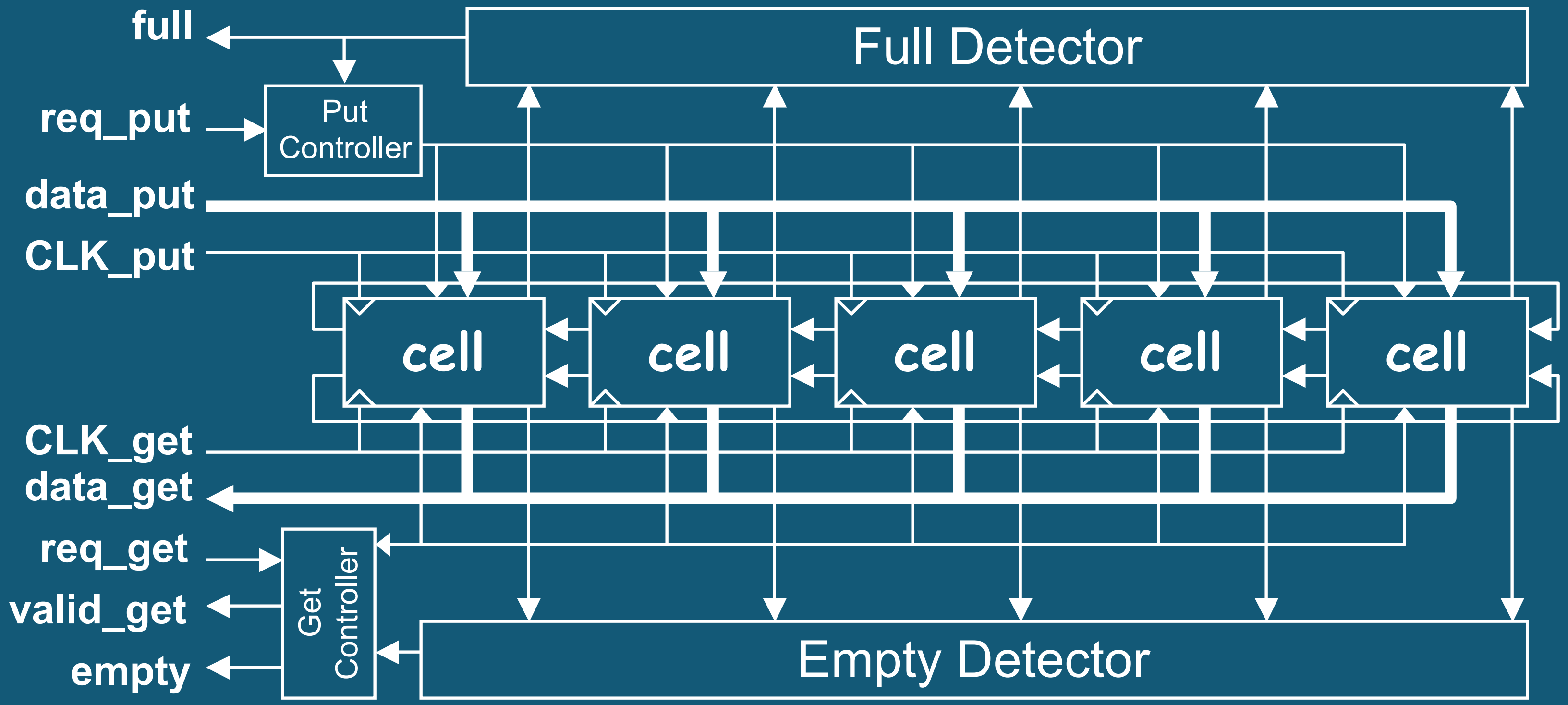
data_get

CLK_get

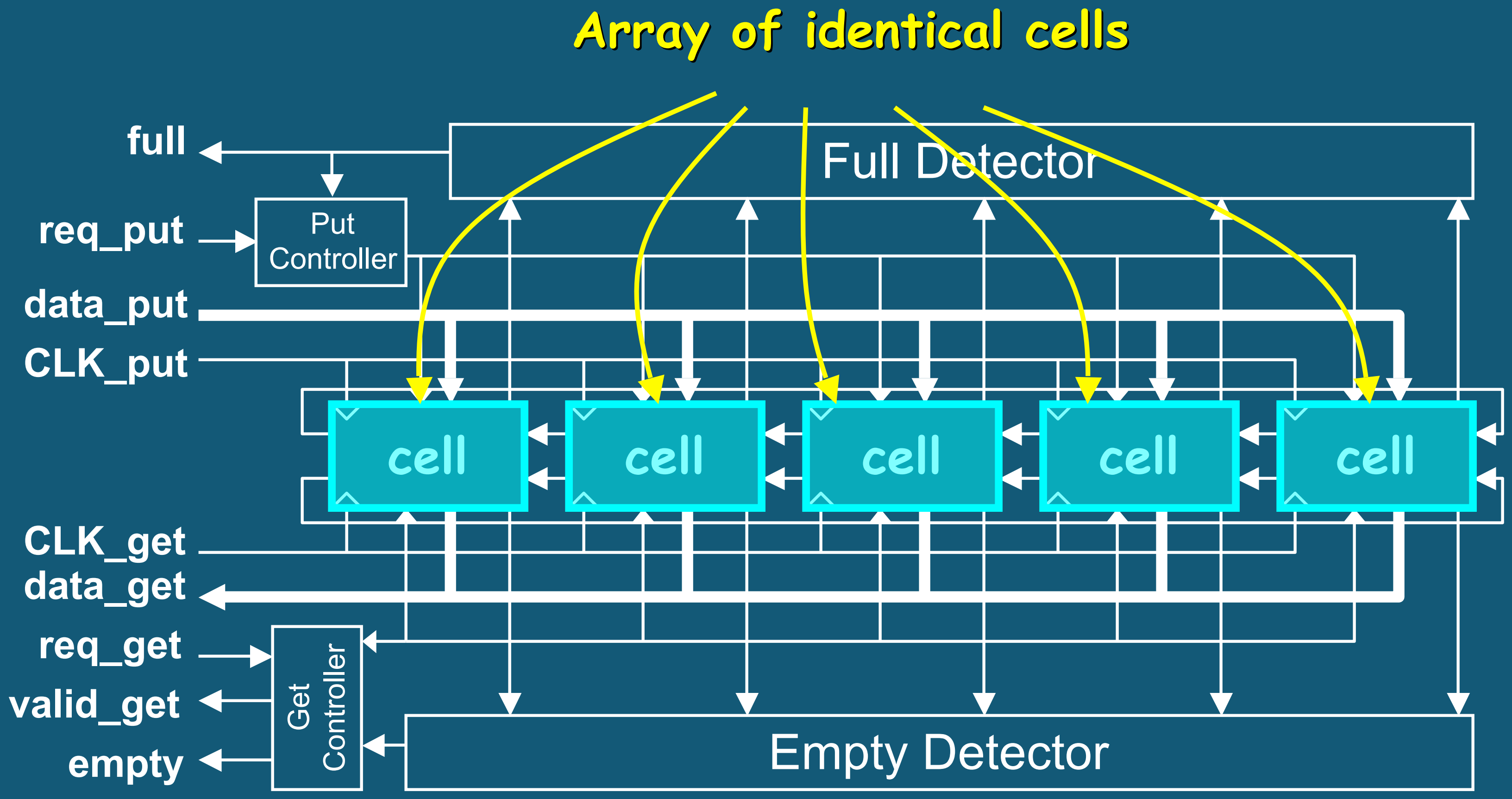
synchronous
get interface

Indicates when FIFO empty

Mixed-Clock FIFO: Architecture



Mixed-Clock FIFO: Architecture

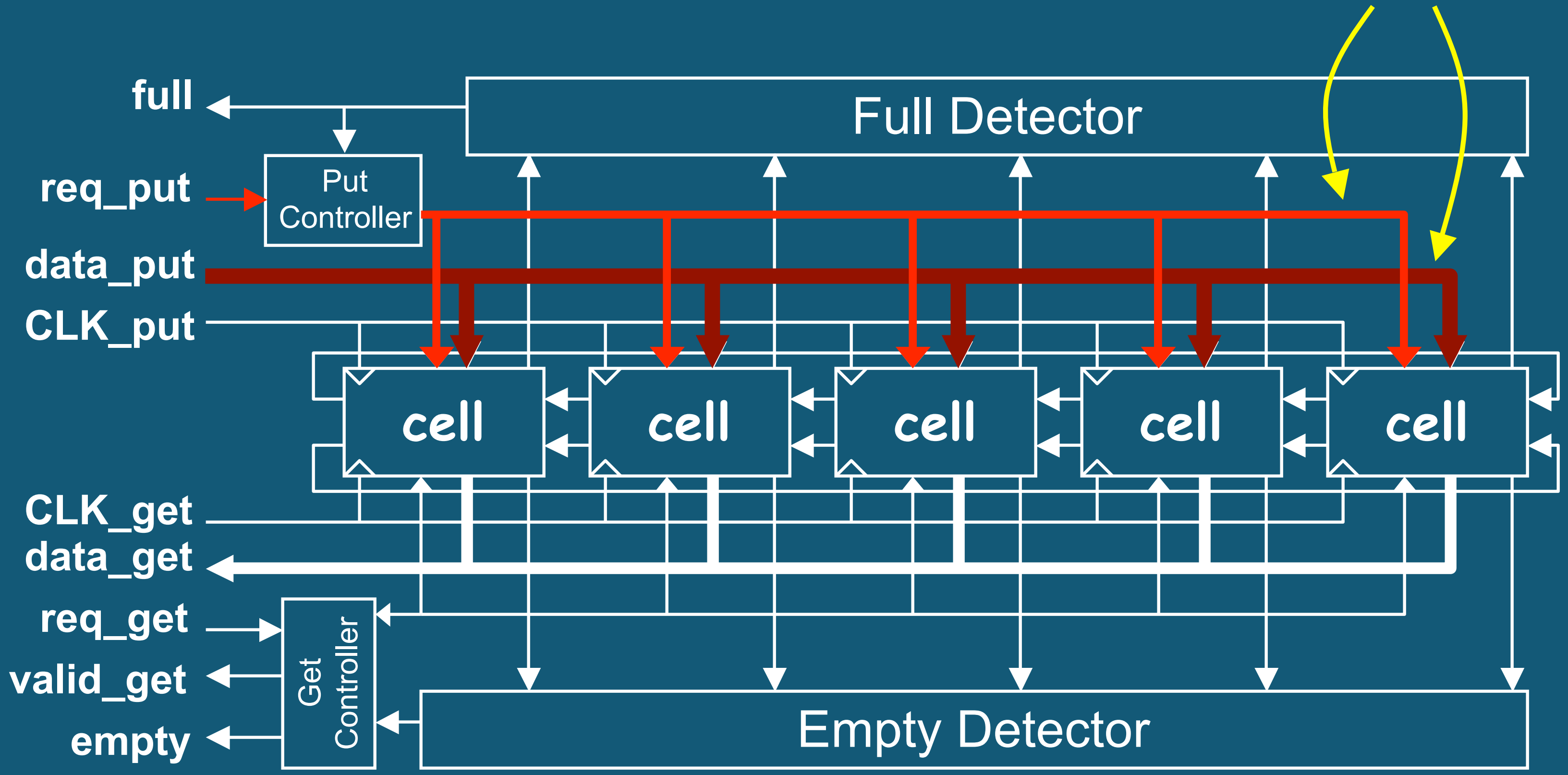


Token Ring Architecture

Mixed-Clock FIFO: Architecture

Put Interface

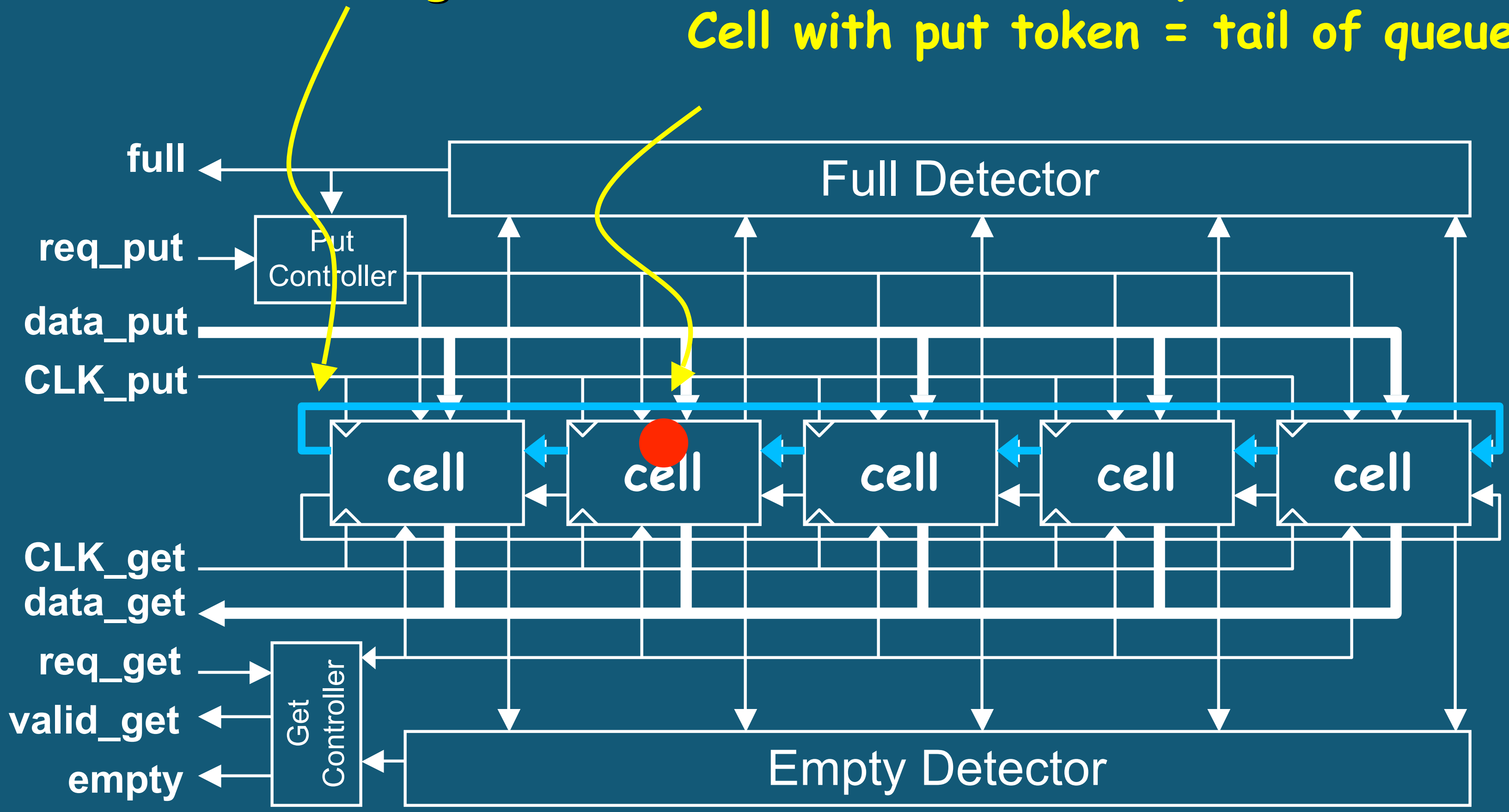
Common Data/Control Buses
for put interface



Mixed-Clock FIFO: Architecture

Put Token Ring

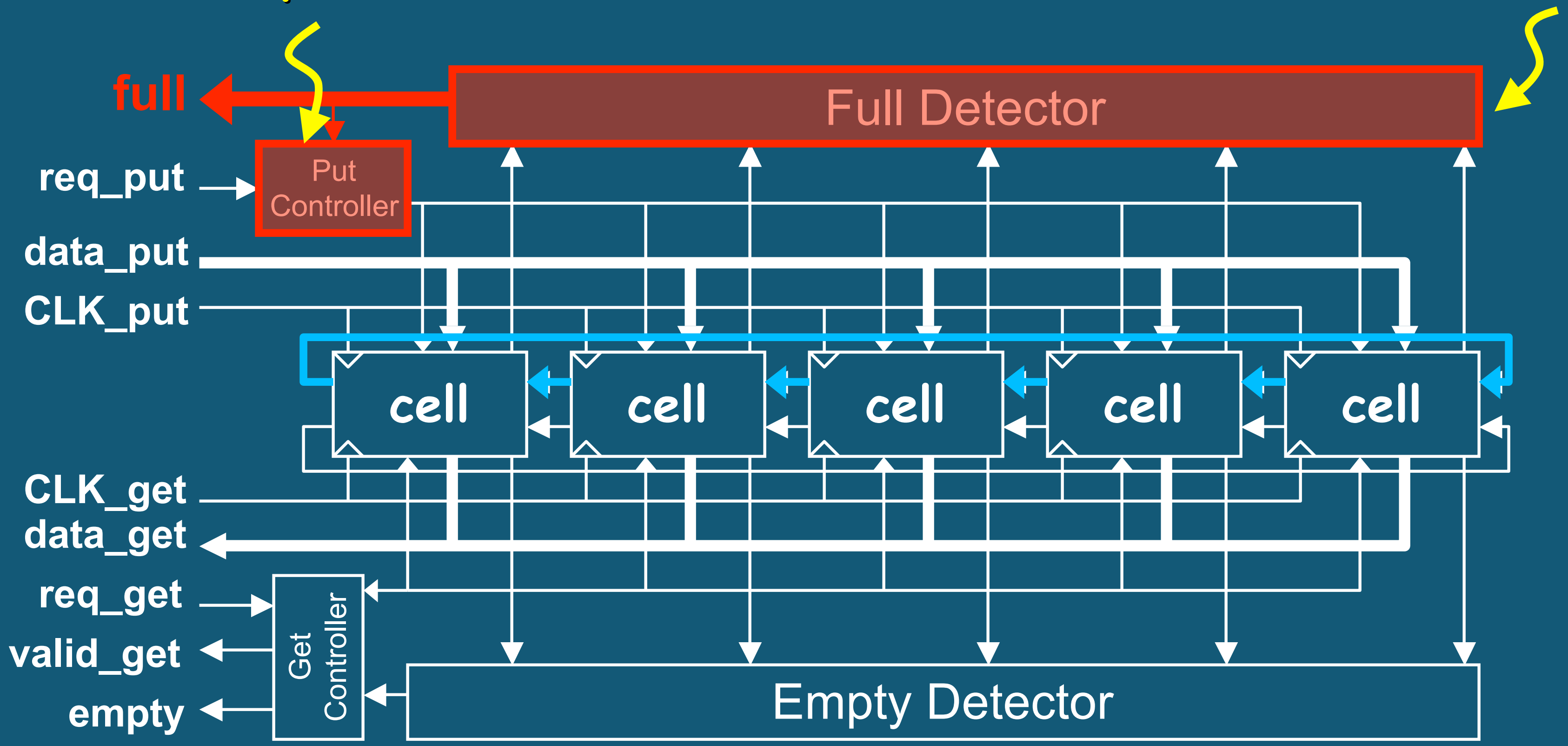
Put Token: used to enqueue data items
Cell with put token = tail of queue



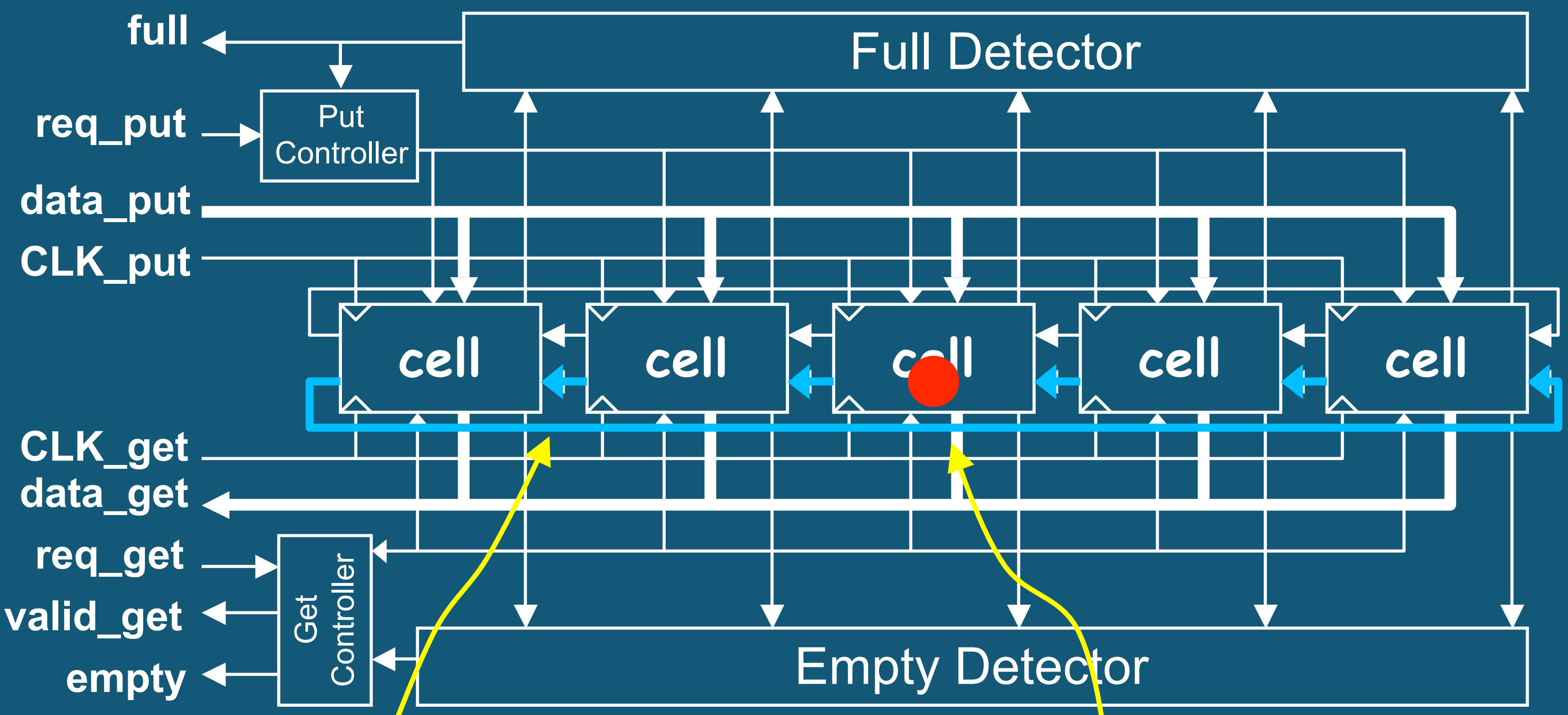
Mixed-Clock FIFO: Architecture

Put Controller:
- enables & disables put operations
- stalls put interface when FIFO full

Full Detector:
detects when FIFO full



Mixed-Clock FIFO: Architecture

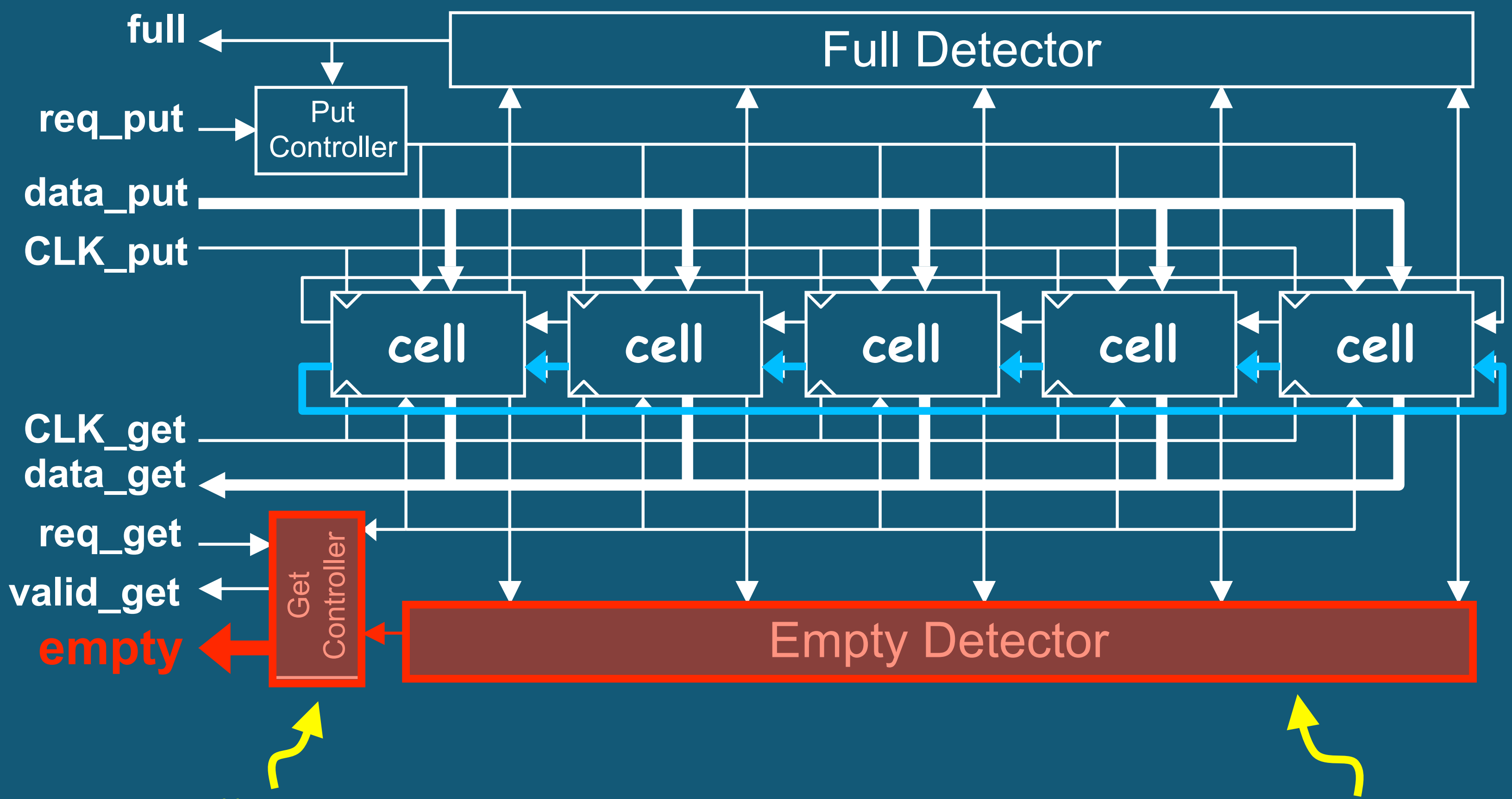


Get Token Ring

Get Token: used to dequeue data items
Cell with get token = head of queue

Get Interface

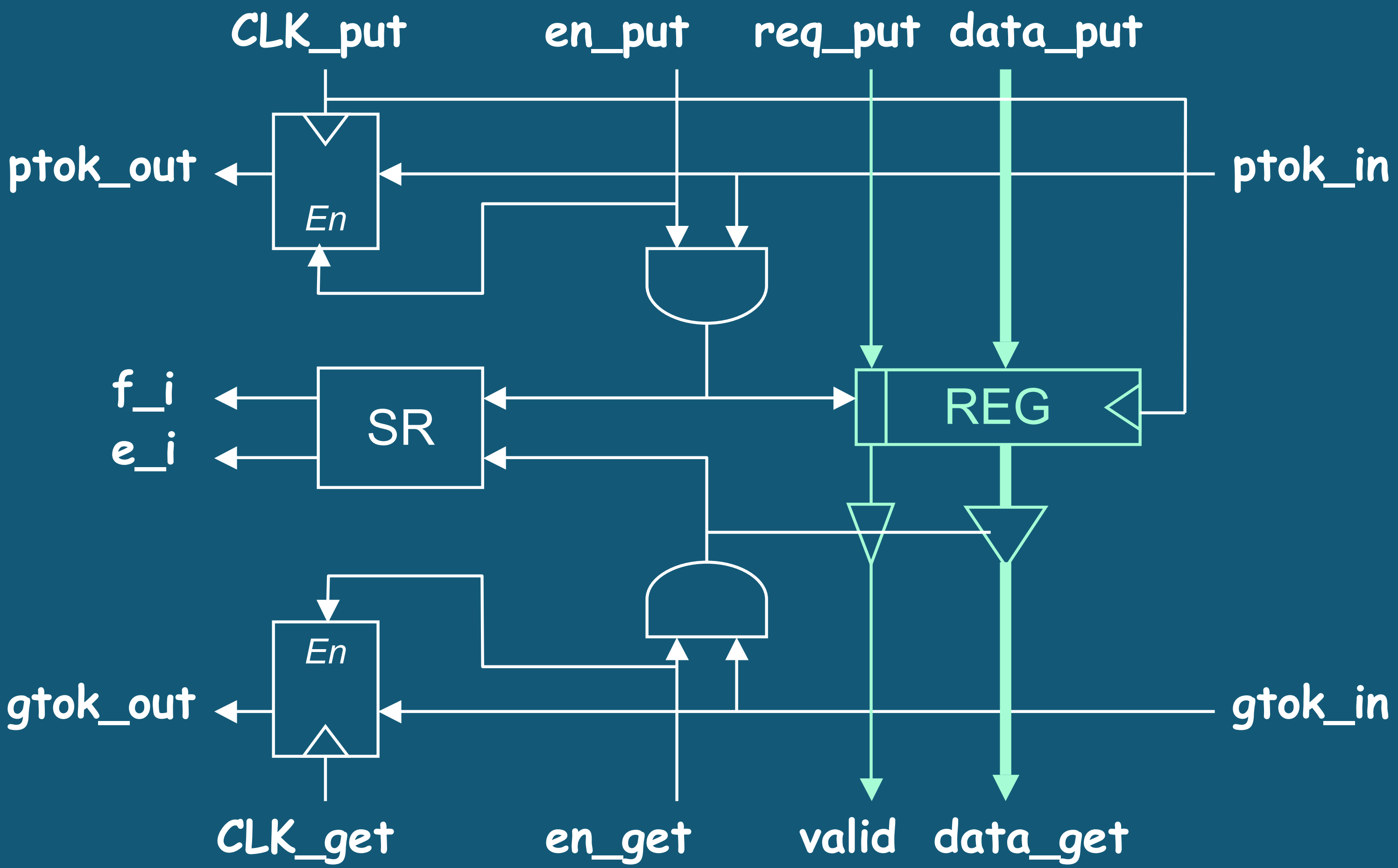
Mixed-Clock FIFO: Architecture



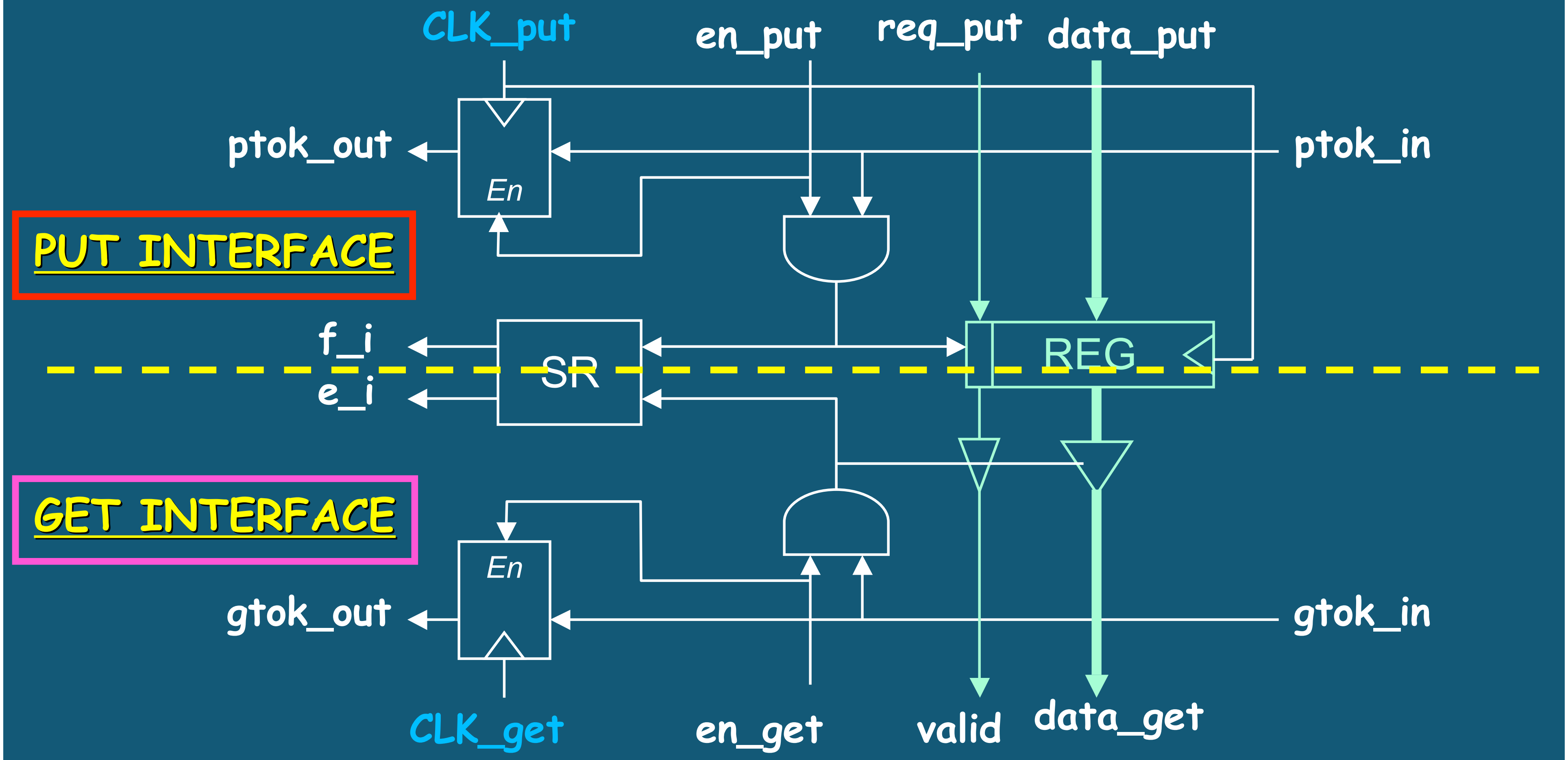
Get Controller:
- enables & disables get operations
- stalls get interface when FIFO empty

Empty Detector:
detects when FIFO empty

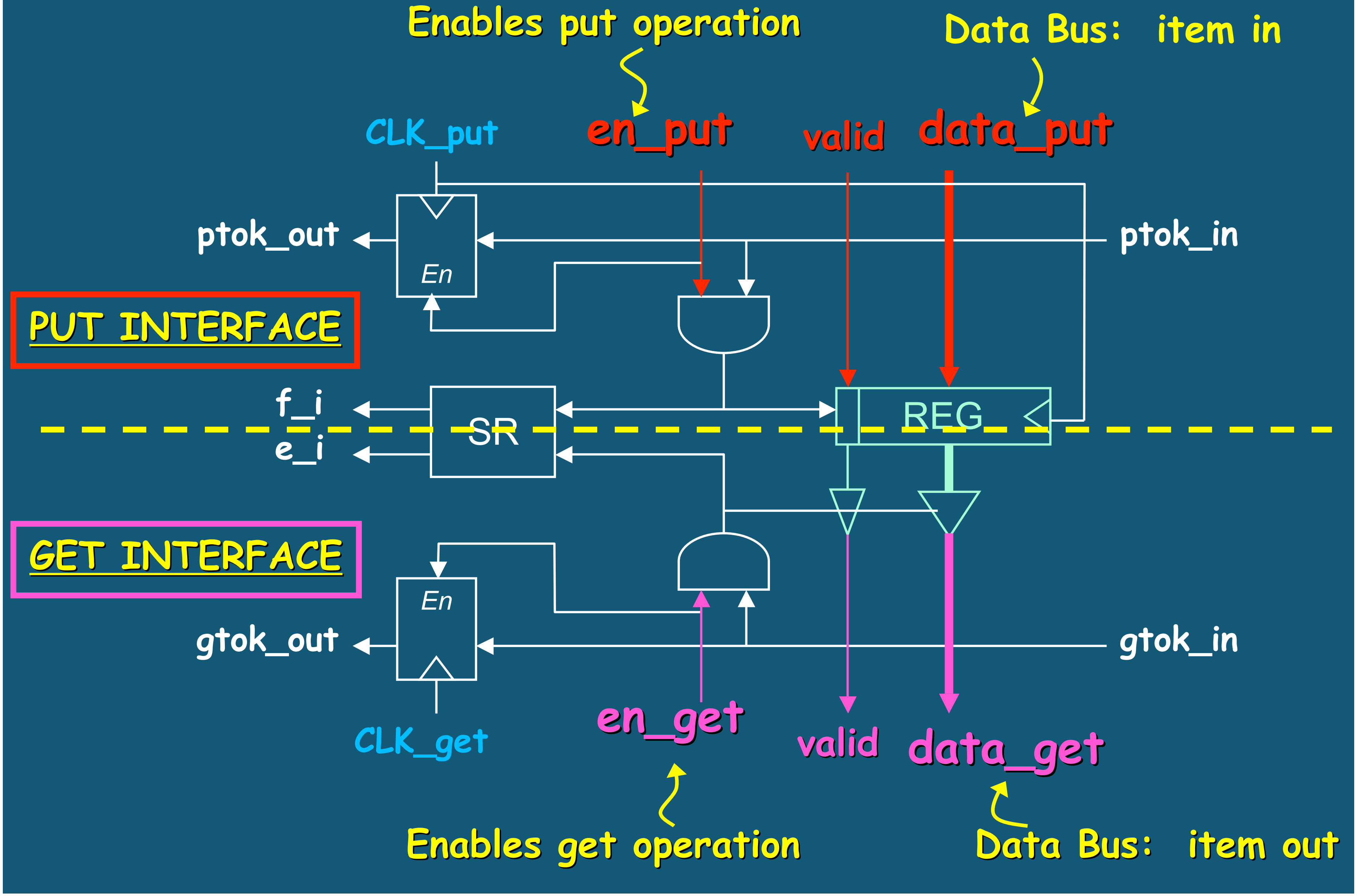
Mixed-Clock FIFO: Cell Implementation



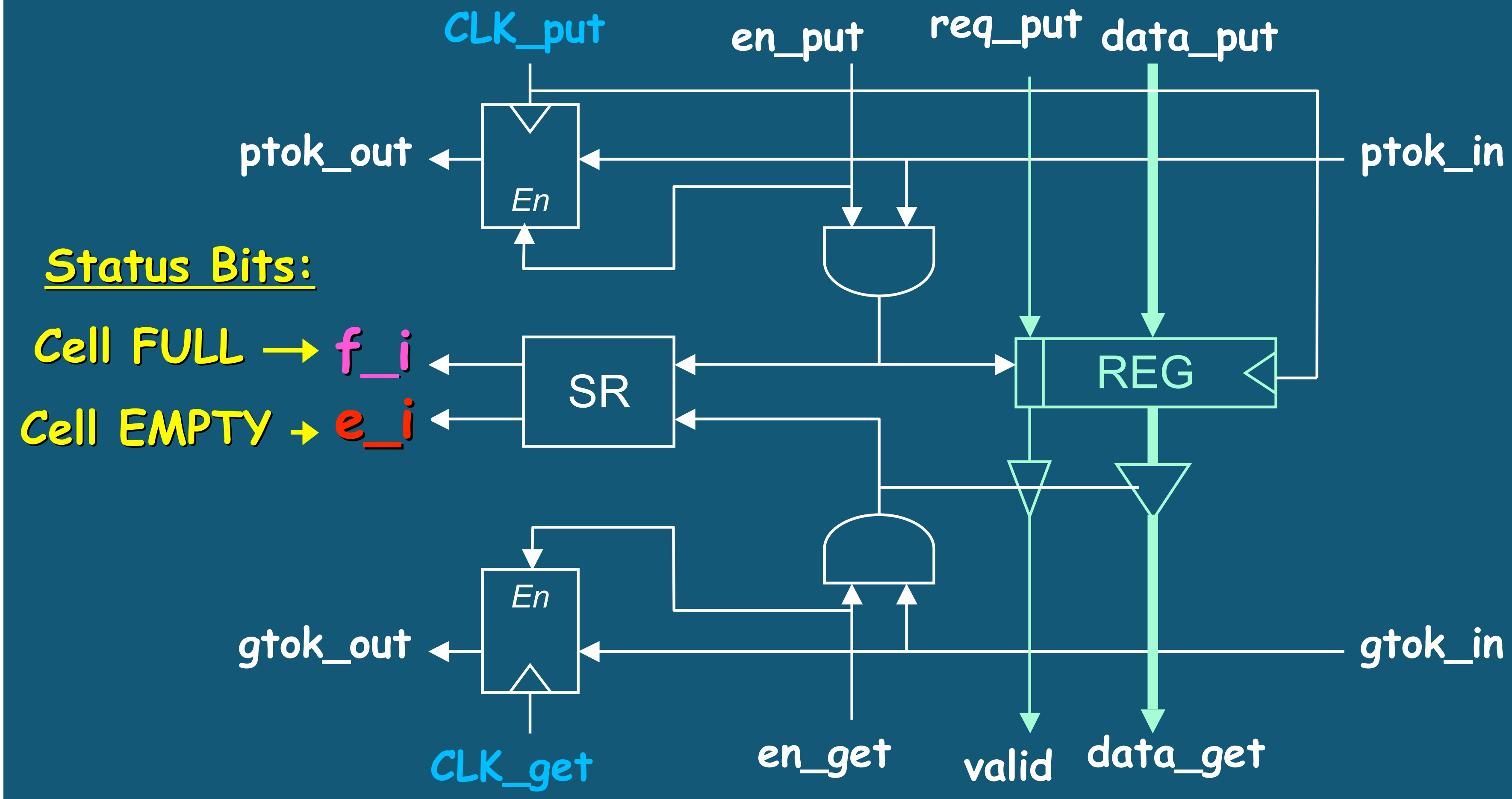
Mixed-Clock FIFO: Cell Implementation



Mixed-Clock FIFO: Cell Implementation



Mixed-Clock FIFO: Cell Implementation

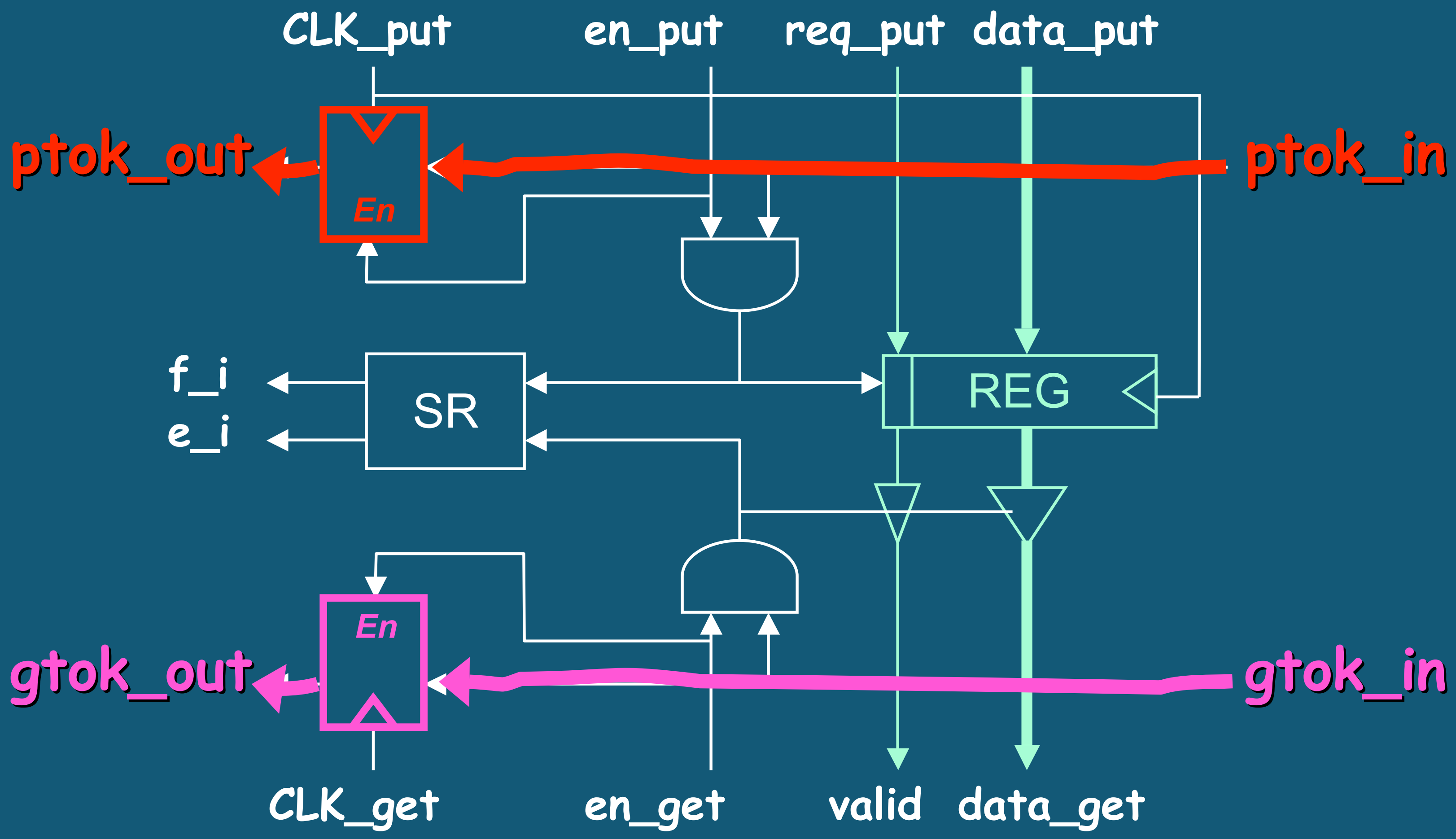


Status Bits:

Cell FULL → f_i

Cell EMPTY → e_i

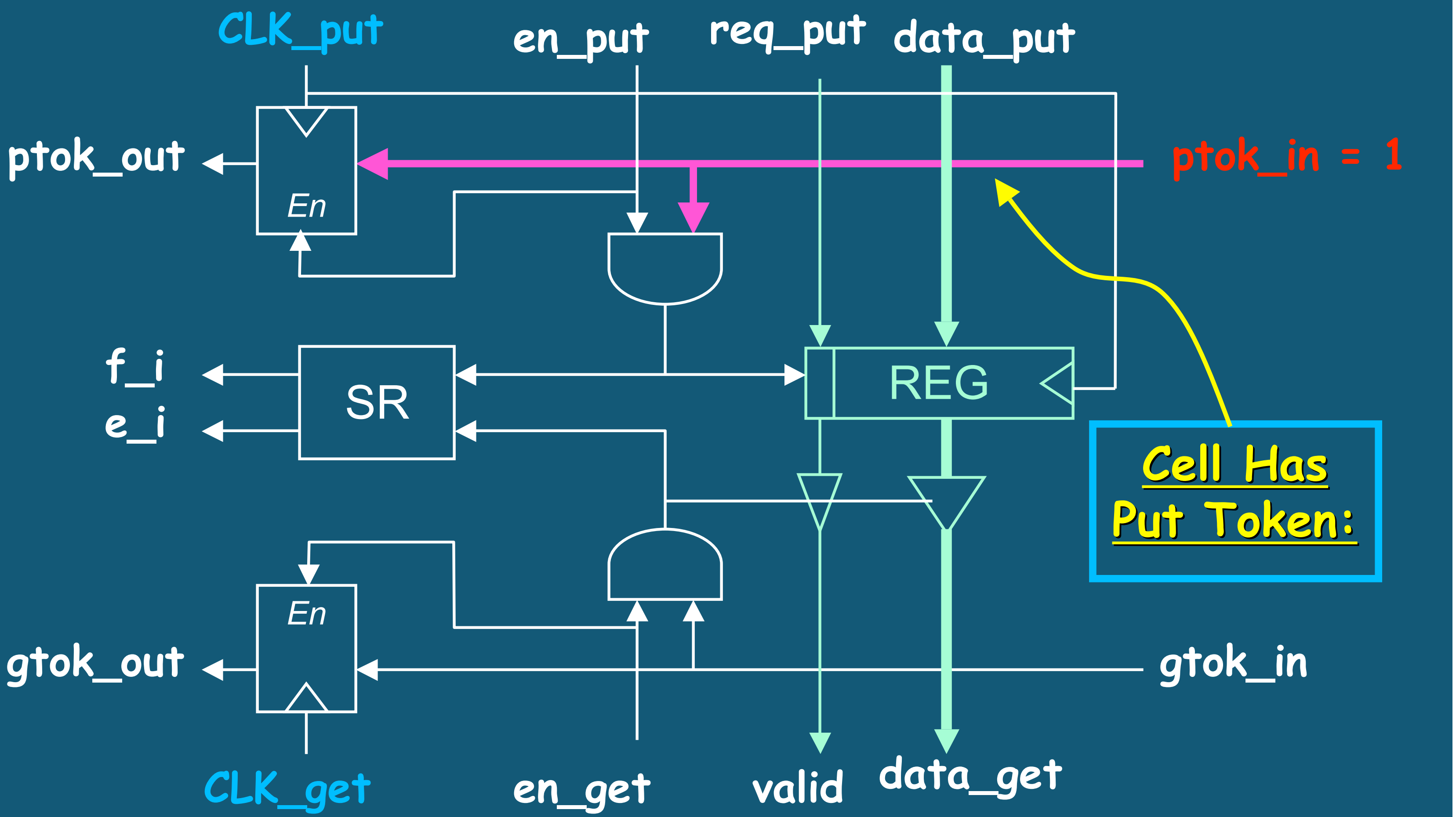
Mixed-Clock FIFO: Cell Implementation



Token Passing:

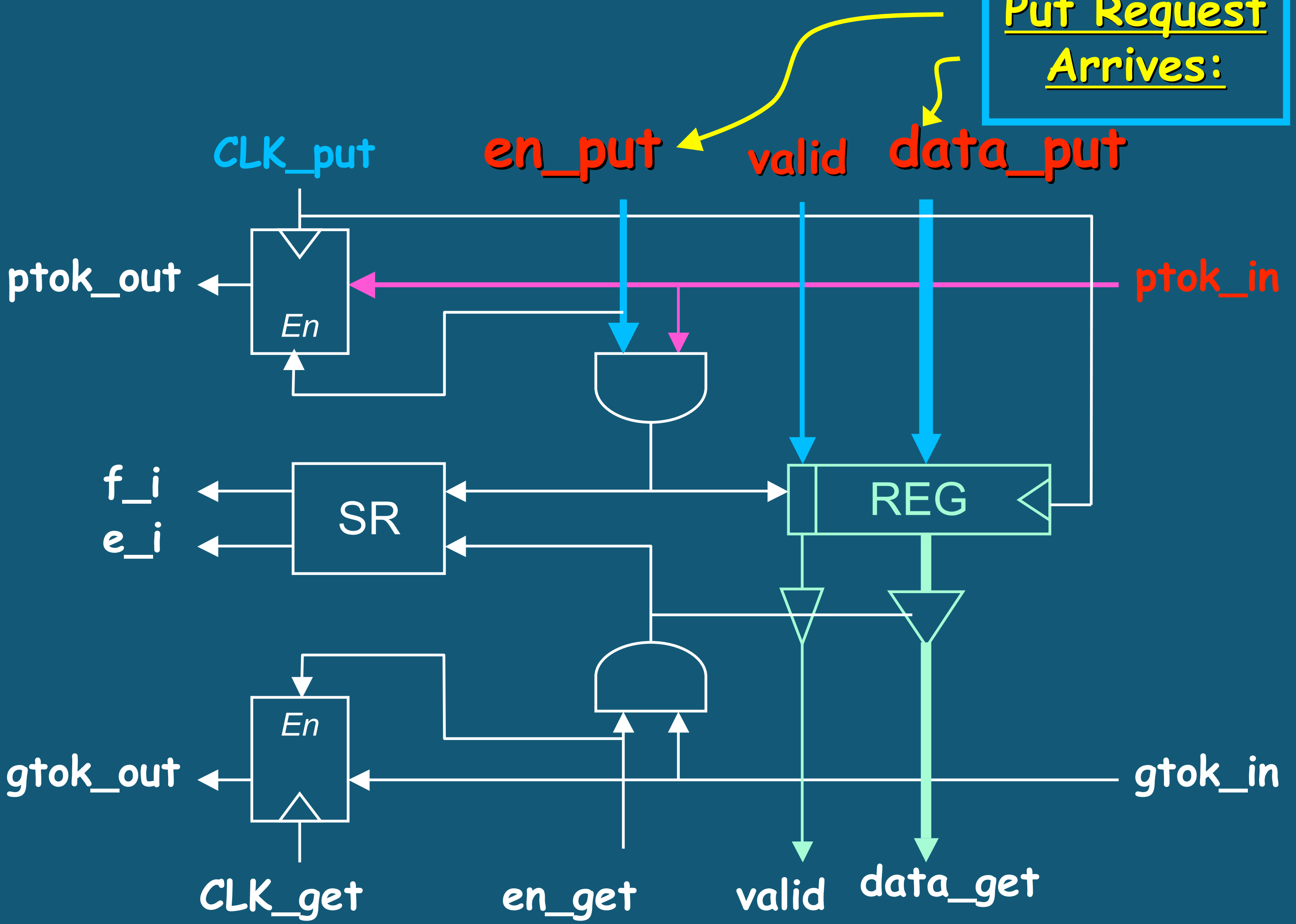
Mixed-Clock FIFO Cell: Put Operation

Simulation #1: Put Operation

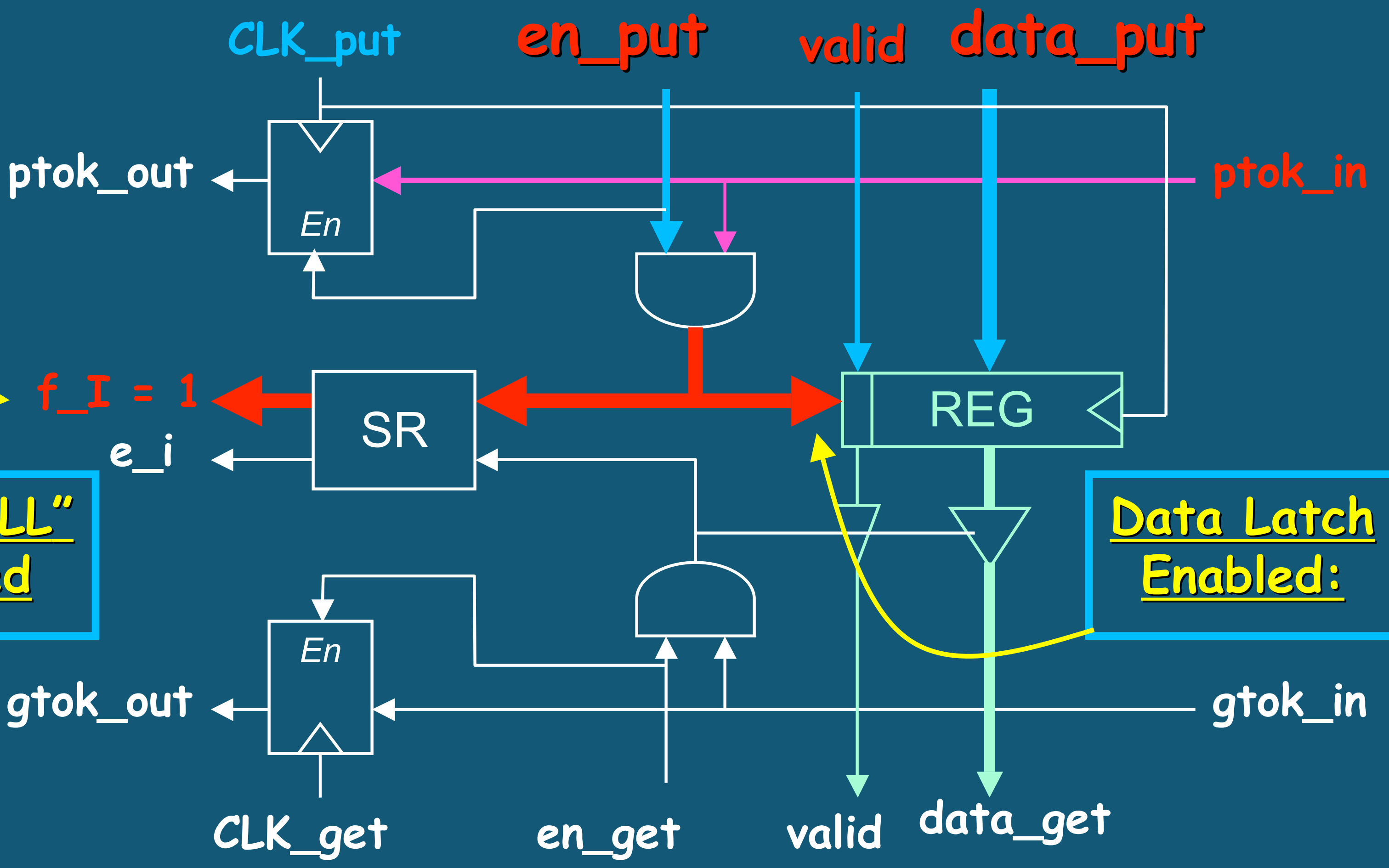


Cell Has Put Token:

Mixed-Clock FIFO Cell: Put Operation



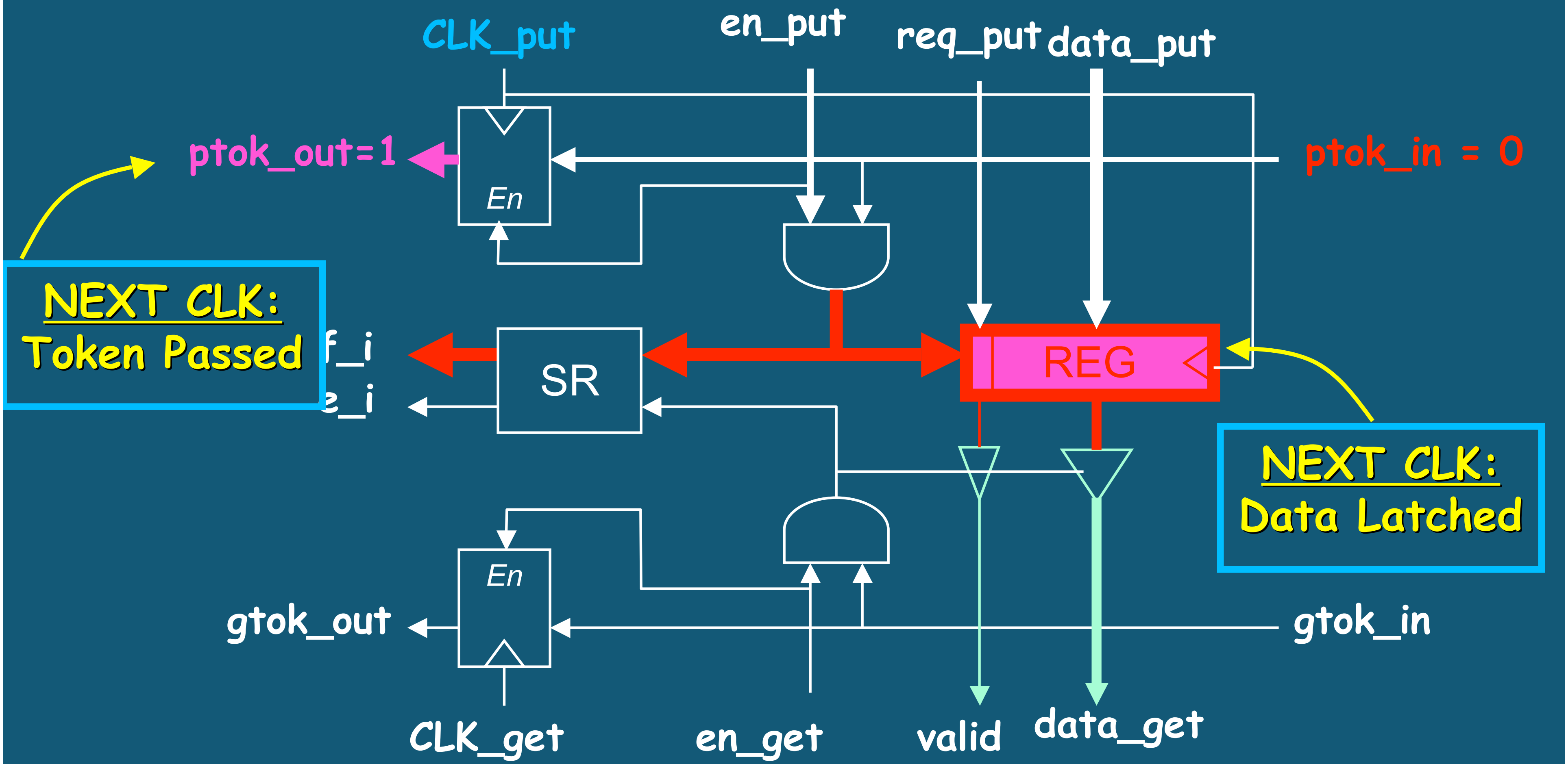
Mixed-Clock FIFO Cell: Put Operation



"FULL CELL" Asserted

Data Latch Enabled:

Mixed-Clock FIFO Cell: Put Operation

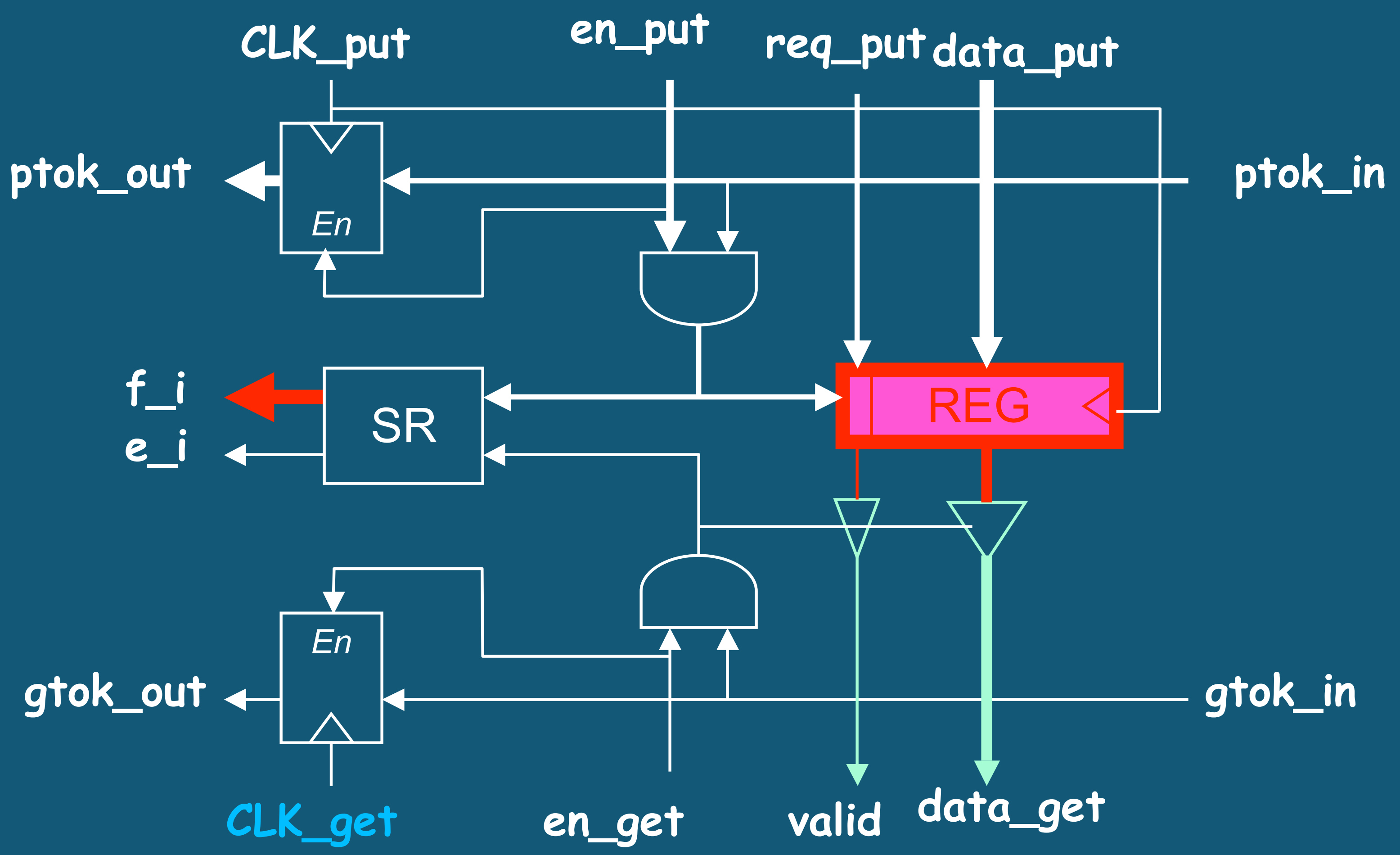


NEXT CLK:
Token Passed

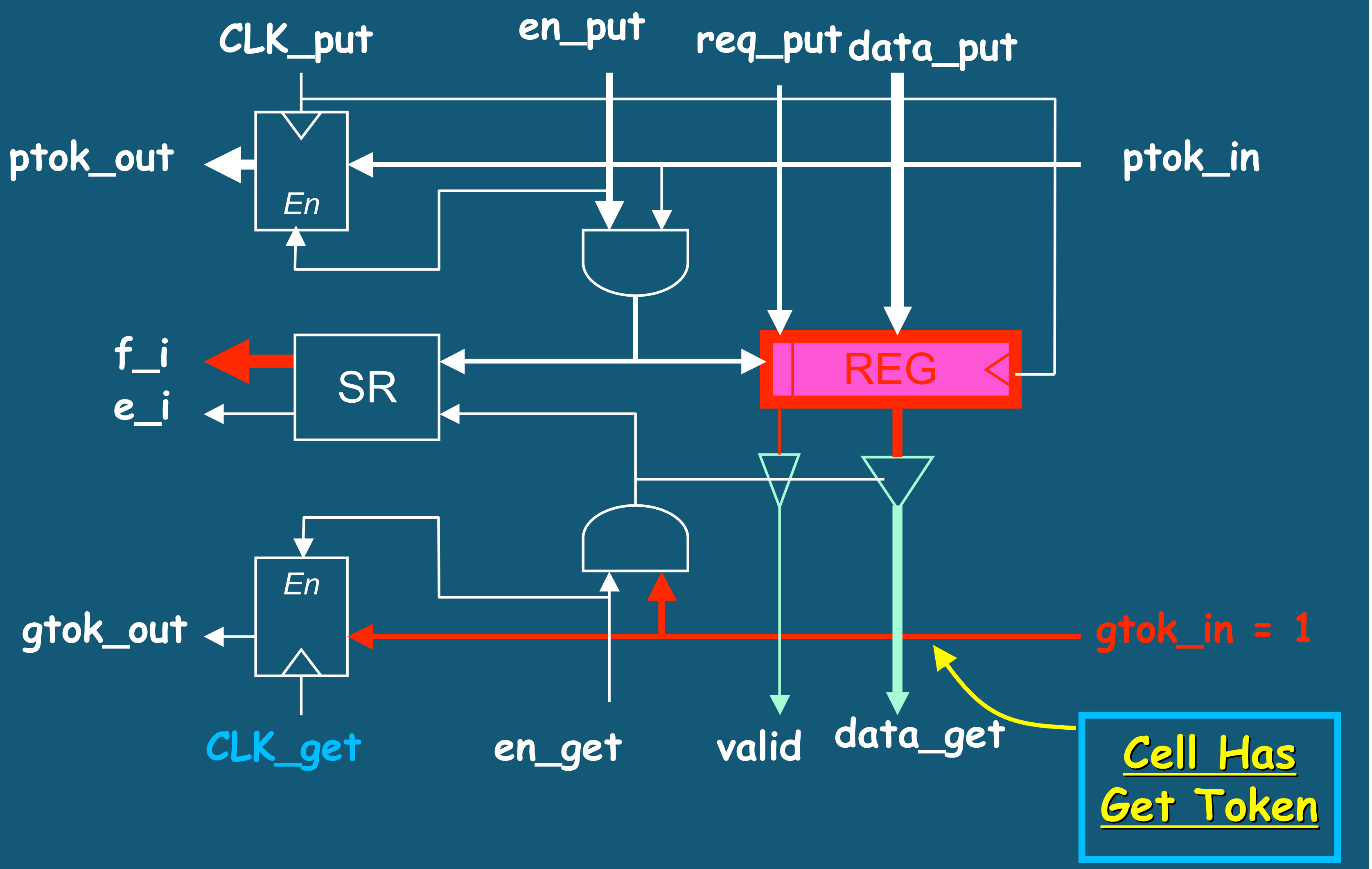
NEXT CLK:
Data Latched

Mixed-Clock FIFO Cell: Get Operation

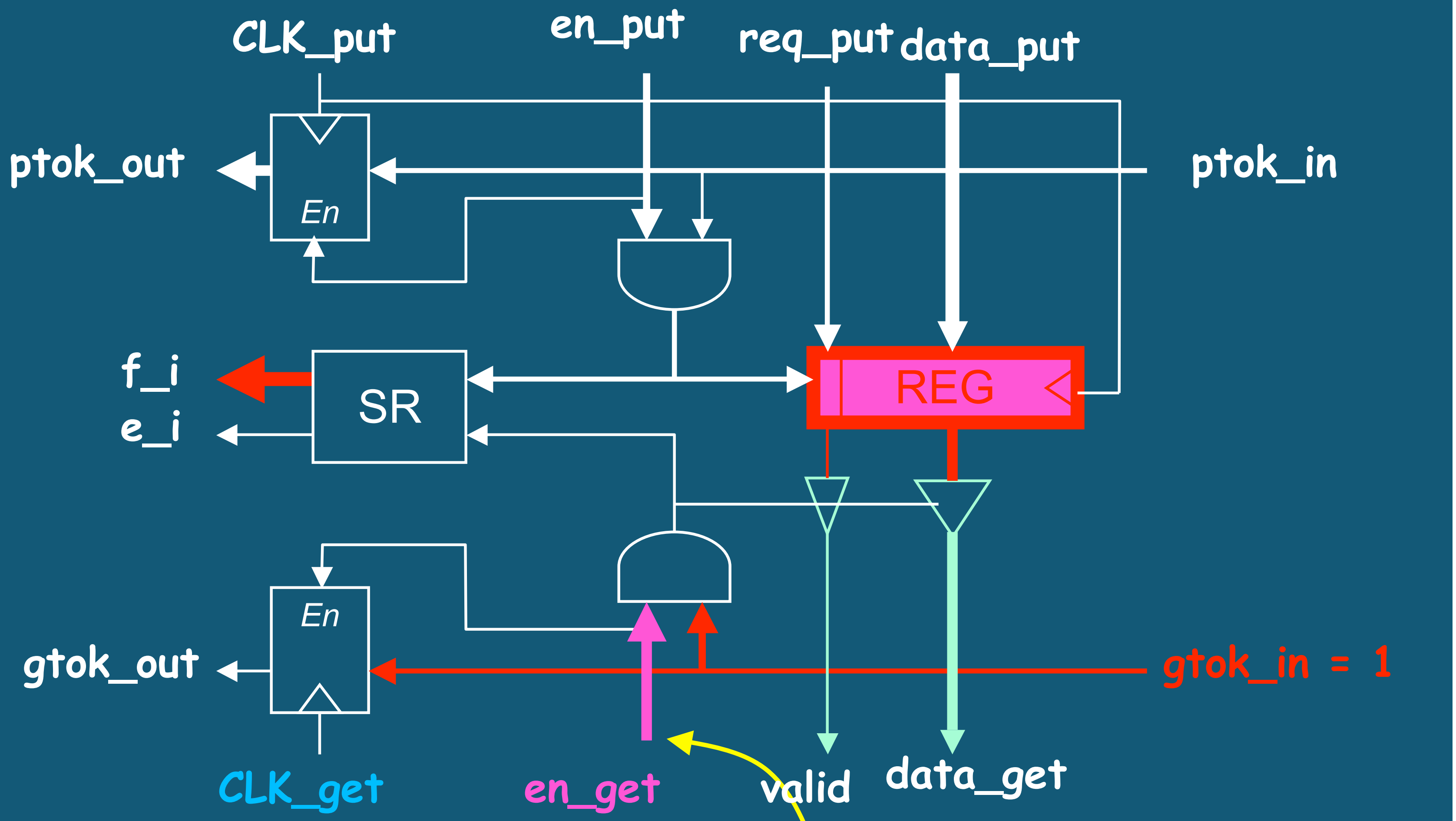
<u>Simulation #2:</u> Get Operation



Mixed-Clock FIFO Cell: Get Operation

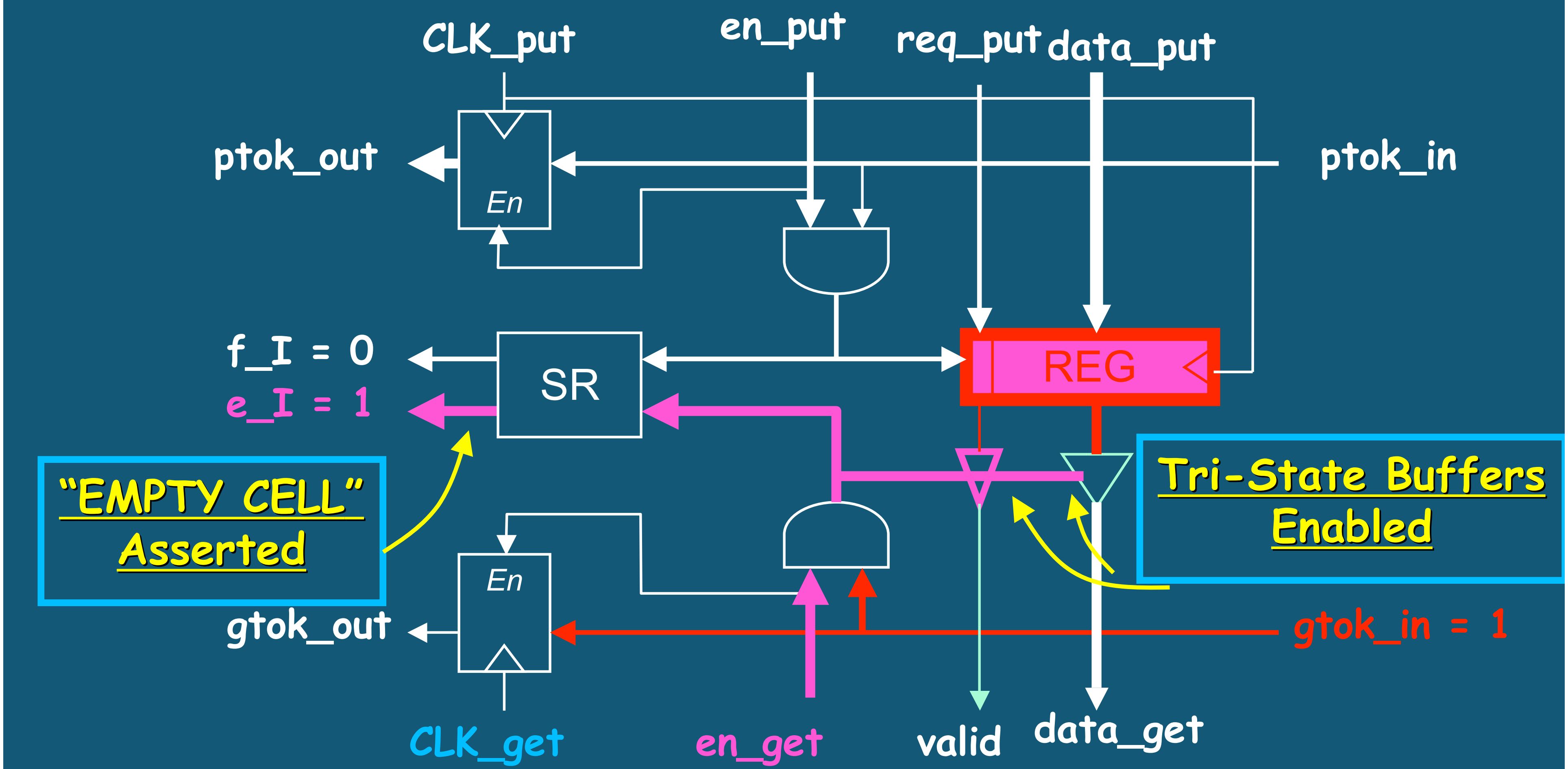


Mixed-Clock FIFO Cell: Get Operation

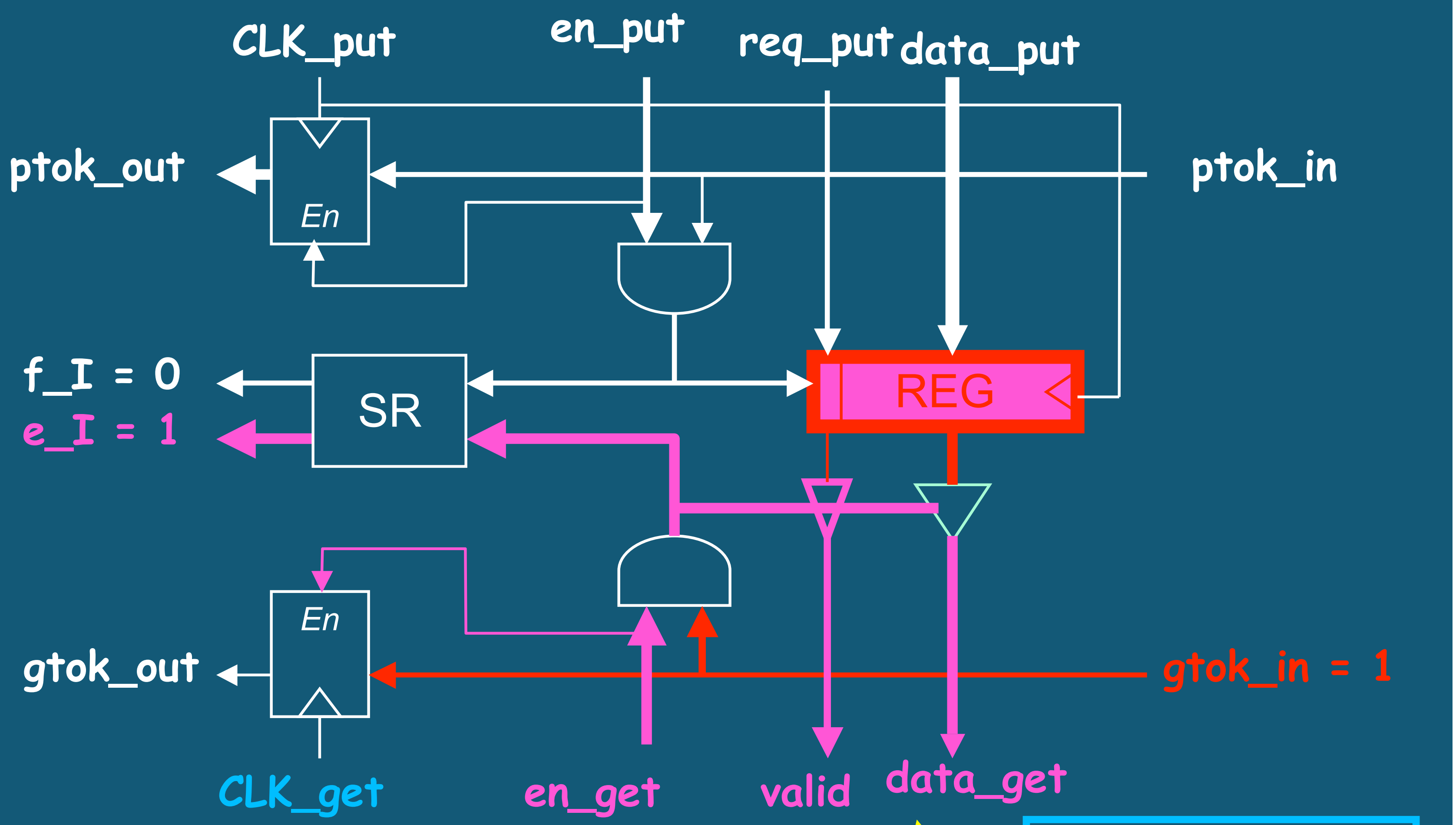


Get Request Arrives

Mixed-Clock FIFO Cell: Get Operation

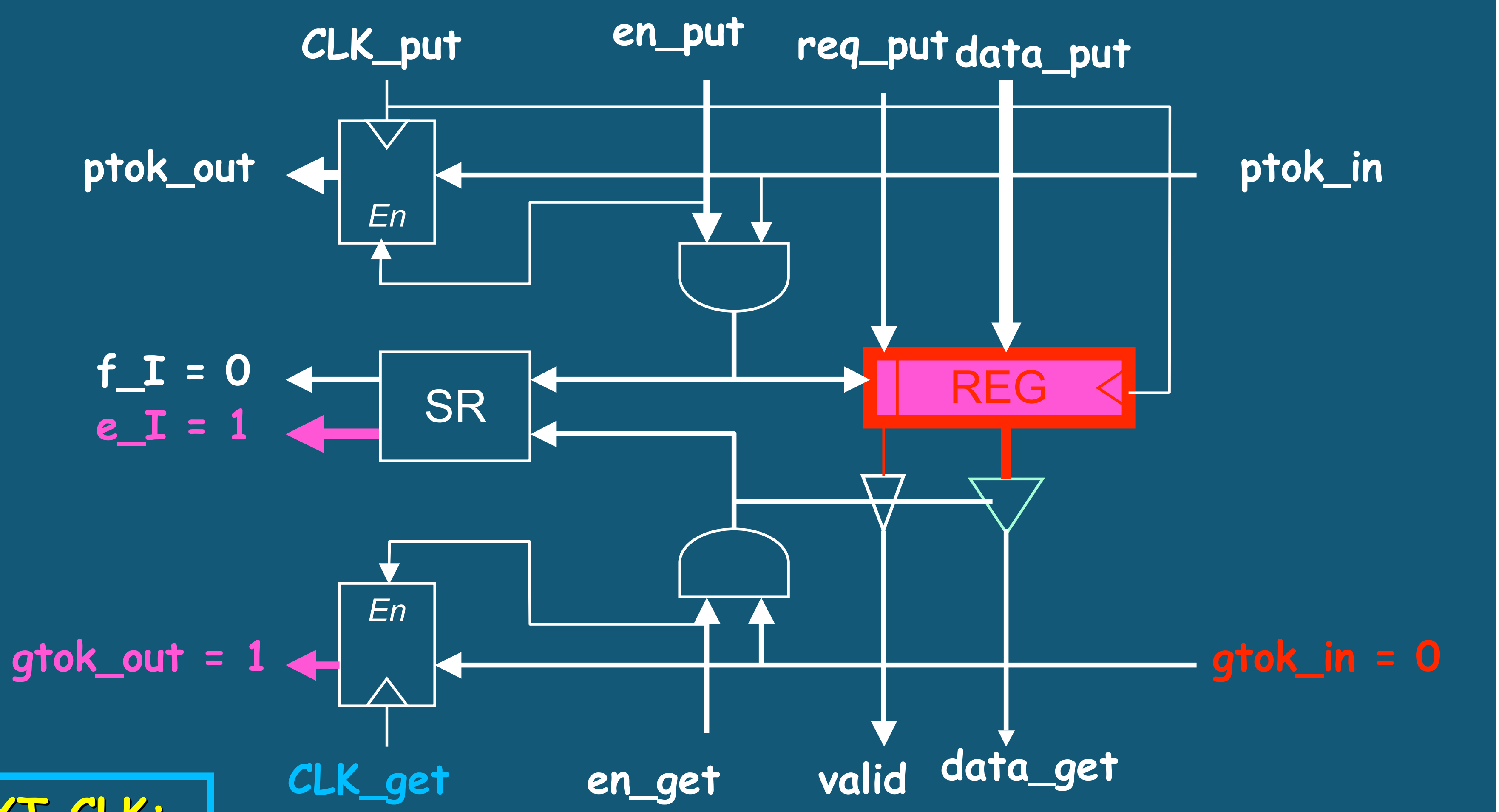


Mixed-Clock FIFO Cell: Get Operation



Data Broadcast on Get Bus

Mixed-Clock FIFO Cell: Get Operation



NEXT CLK:
Token Passed

Synchronization Issues: Overview

Challenge: highly concurrent behavior

- Global FIFO state controlled by two different clocks

Problem #1: Metastability

- Each FIFO interface needs *clean state signals*

Solution #1: Synchronize “full” & “empty” signals

- **“full”** with **CLK_put**
- **“empty”** with **CLK_get**

Add 2 synchronizing latches each

Mixed-Clock FIFO: Full/Empty Detectors

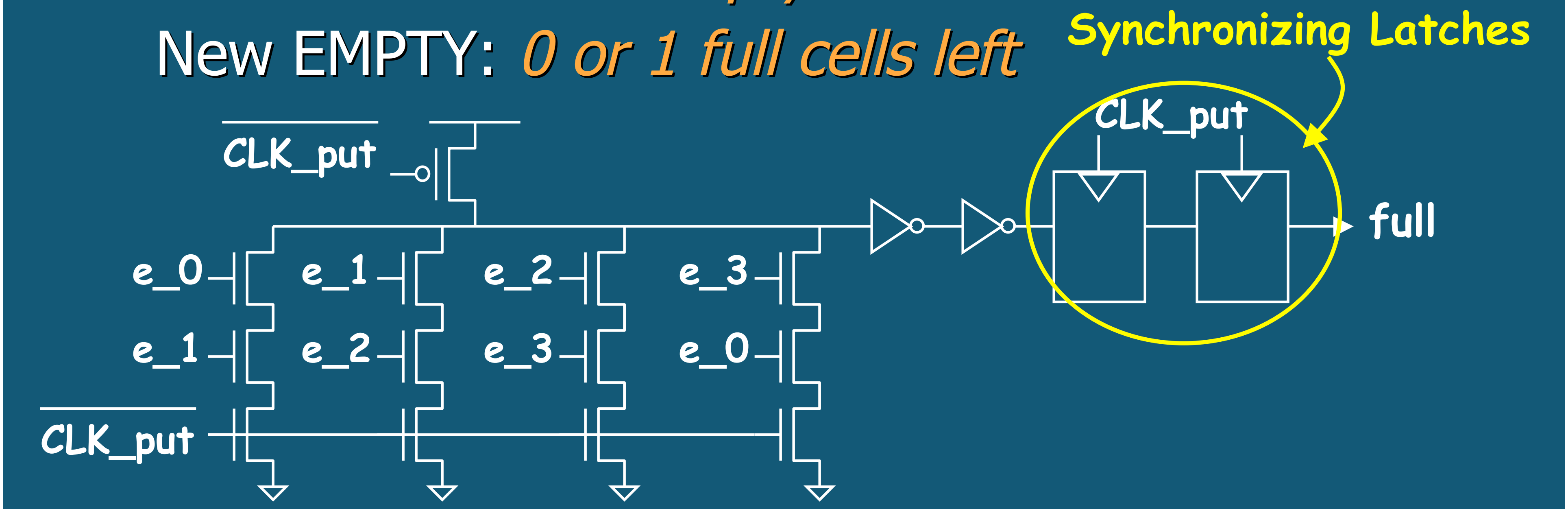
Problem #2: FIFO now may underflow/overflow!

- synchronizing latches add extra latency

Solution #2: Change Full/Empty definitions

New FULL: *0 or 1 empty cells left*

New EMPTY: *0 or 1 full cells left*



New Full Detector

Observable full/empty *safely approximate* FIFO’s state

Mixed-Clock FIFO: Full/Empty Detectors

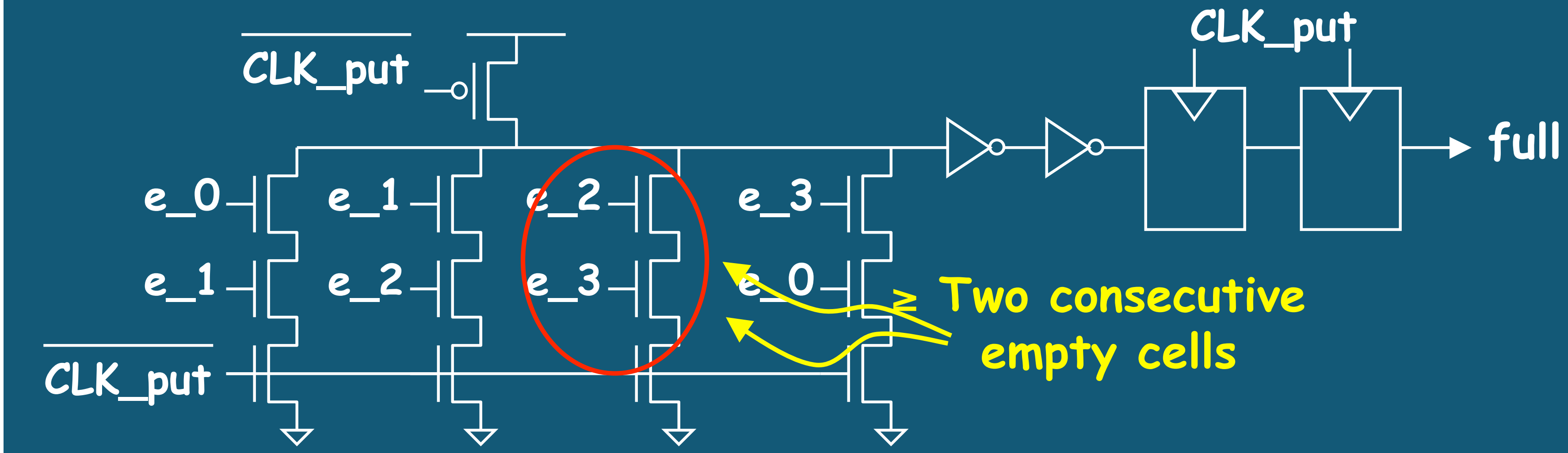
Problem #2: FIFO now may underflow/overflow!

- synchronizing latches add extra latency

Solution #2: Change Full/Empty definitions

New FULL: *0 or 1 empty cells left*

New EMPTY: *0 or 1 full cells left*



New Full Detector

Observable full/empty *safely approximate* FIFO’s state

Mixed-Clock FIFO: Full/Empty Detectors

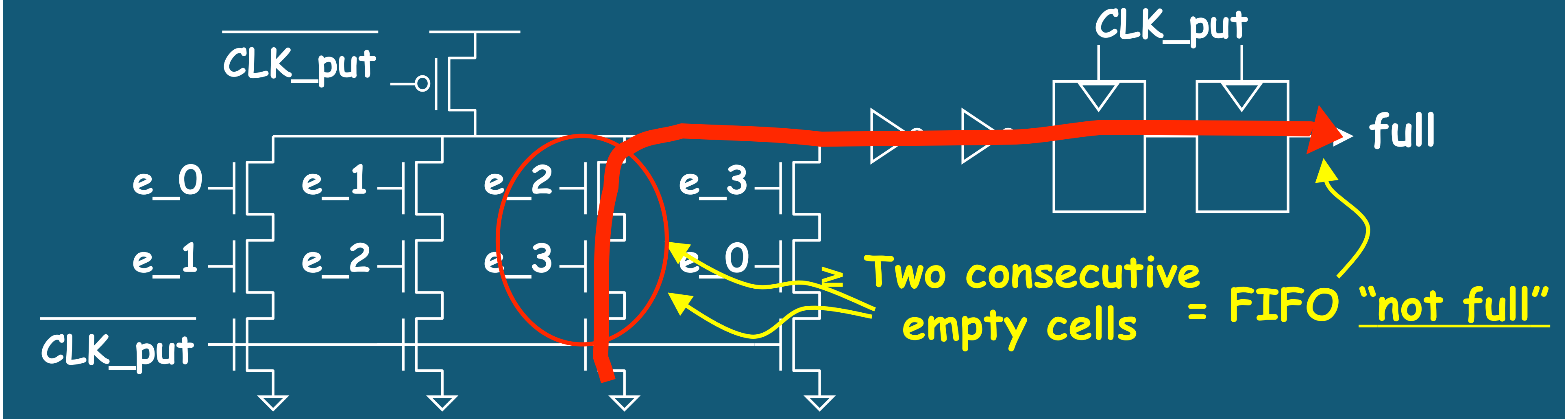
Problem #2: FIFO now may underflow/overflow!

- synchronizing latches add extra latency

Solution #2: Change Full/Empty definitions

New FULL: *0 or 1 empty cells left*

New EMPTY: *0 or 1 full cells left*



≥ Two consecutive empty cells = FIFO "not full"

New Full Detector

Observable full/empty *safely approximate* FIFO's state

Mixed-Clock FIFO: Full/Empty Detectors

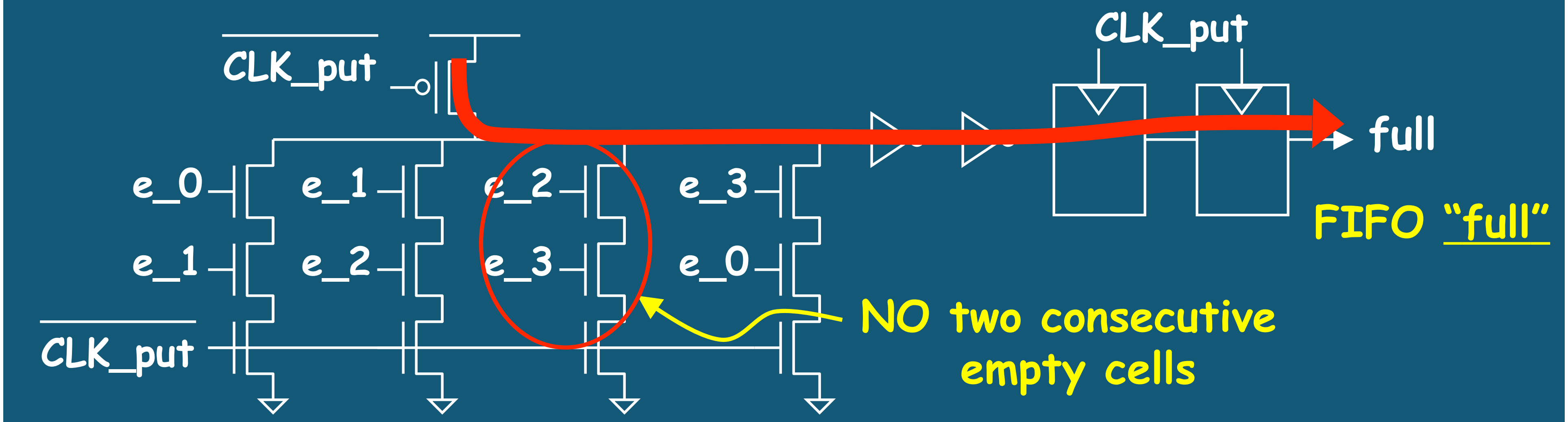
Problem #2: FIFO now may underflow/overflow!

- synchronizing latches add extra latency

Solution #2: Change Full/Empty definitions

New FULL: *0 or 1 empty cells left*

New EMPTY: *0 or 1 full cells left*



New Full Detector

Observable full/empty *safely approximate* FIFO's state

Deadlock Avoidance

Problem #3: potential for deadlock

Scenario: only 1 data item in FIFO

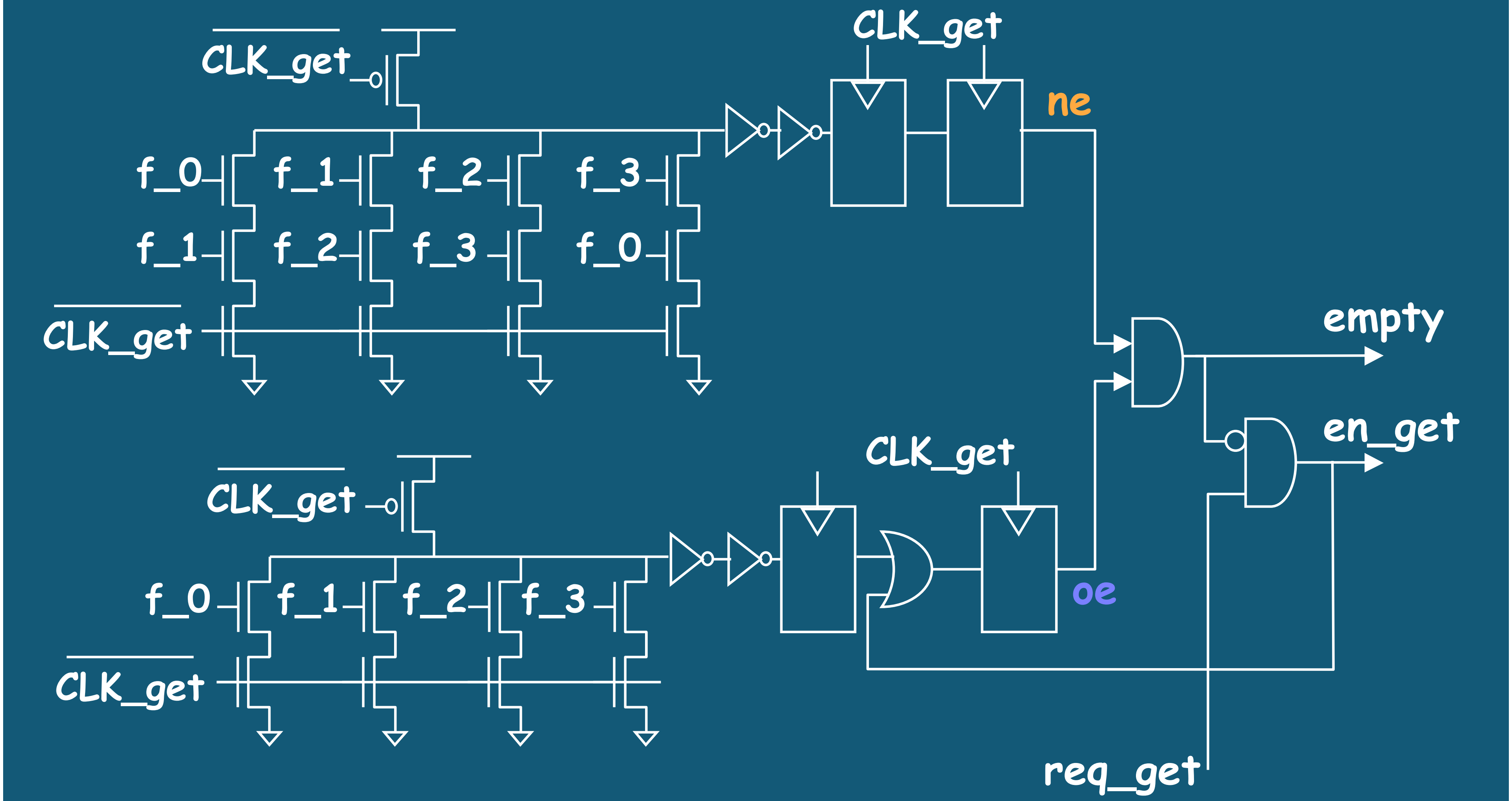
- FIFO still considered "empty" (new definition)
- Get interface: *cannot dequeue item!*

Solution #3: bi-modal empty detector

- *"New empty"* detector (0 or 1 data items)
- *"True empty"* detector (0 data items)

Combine two results into single global *"empty"*

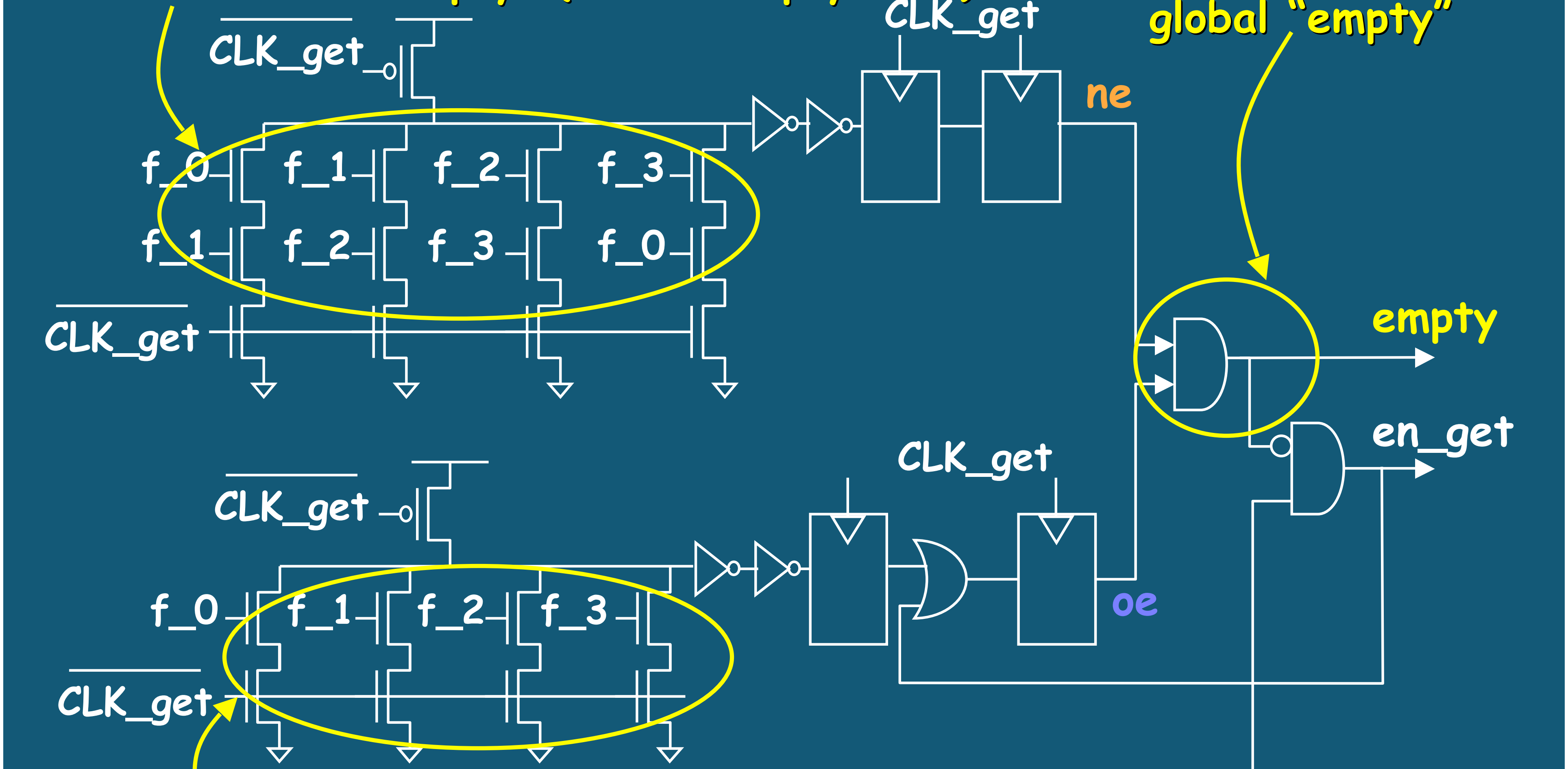
Mixed-Clock FIFO: Deadlock Avoidance



Mixed-Clock FIFO: Deadlock Avoidance

Detects "new empty" (0 or 1 empty cells)

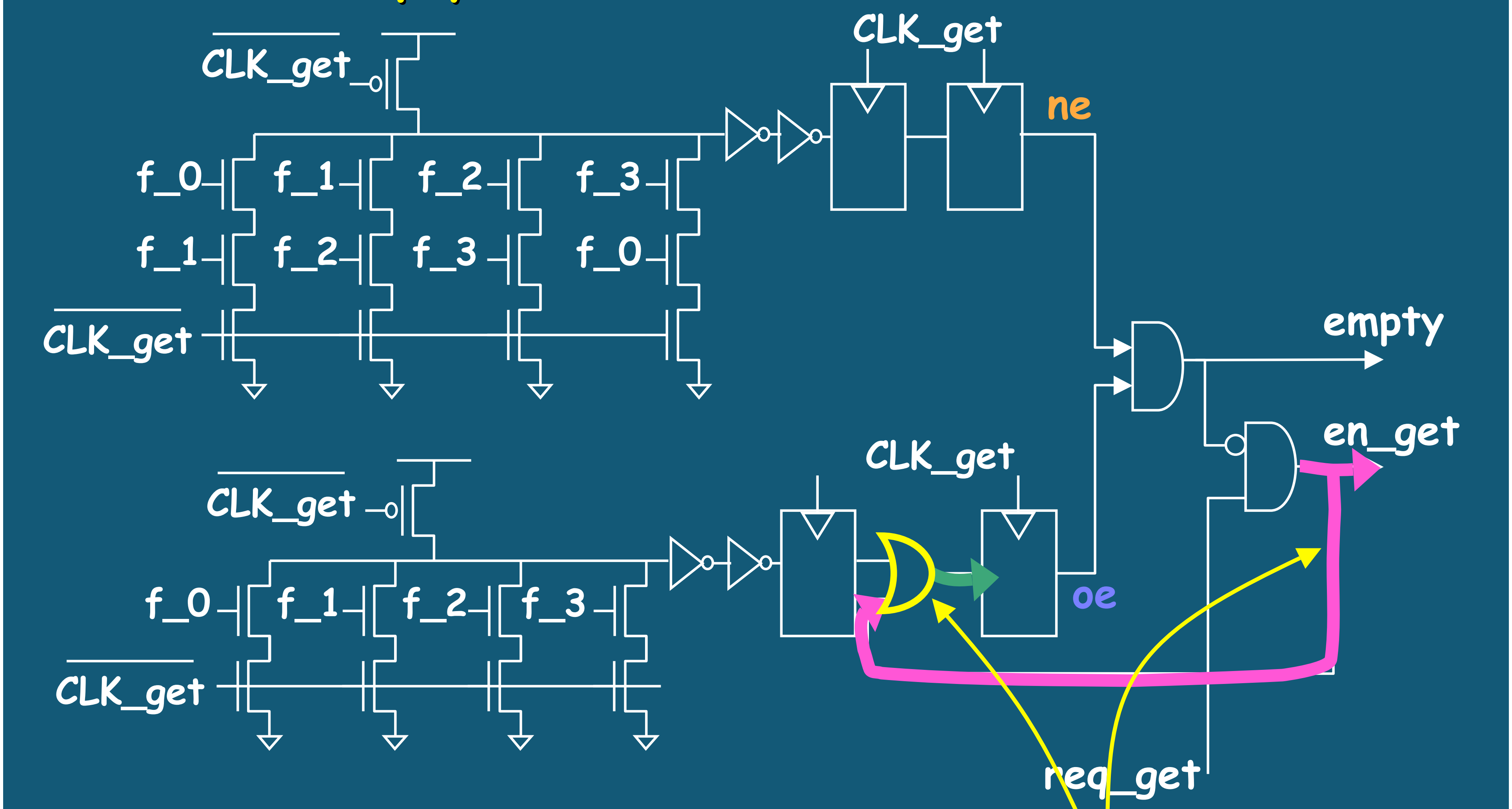
Combine into global "empty"



Detects "true empty" (0 empty cells)

Mixed-Clock FIFO: Deadlock Avoidance

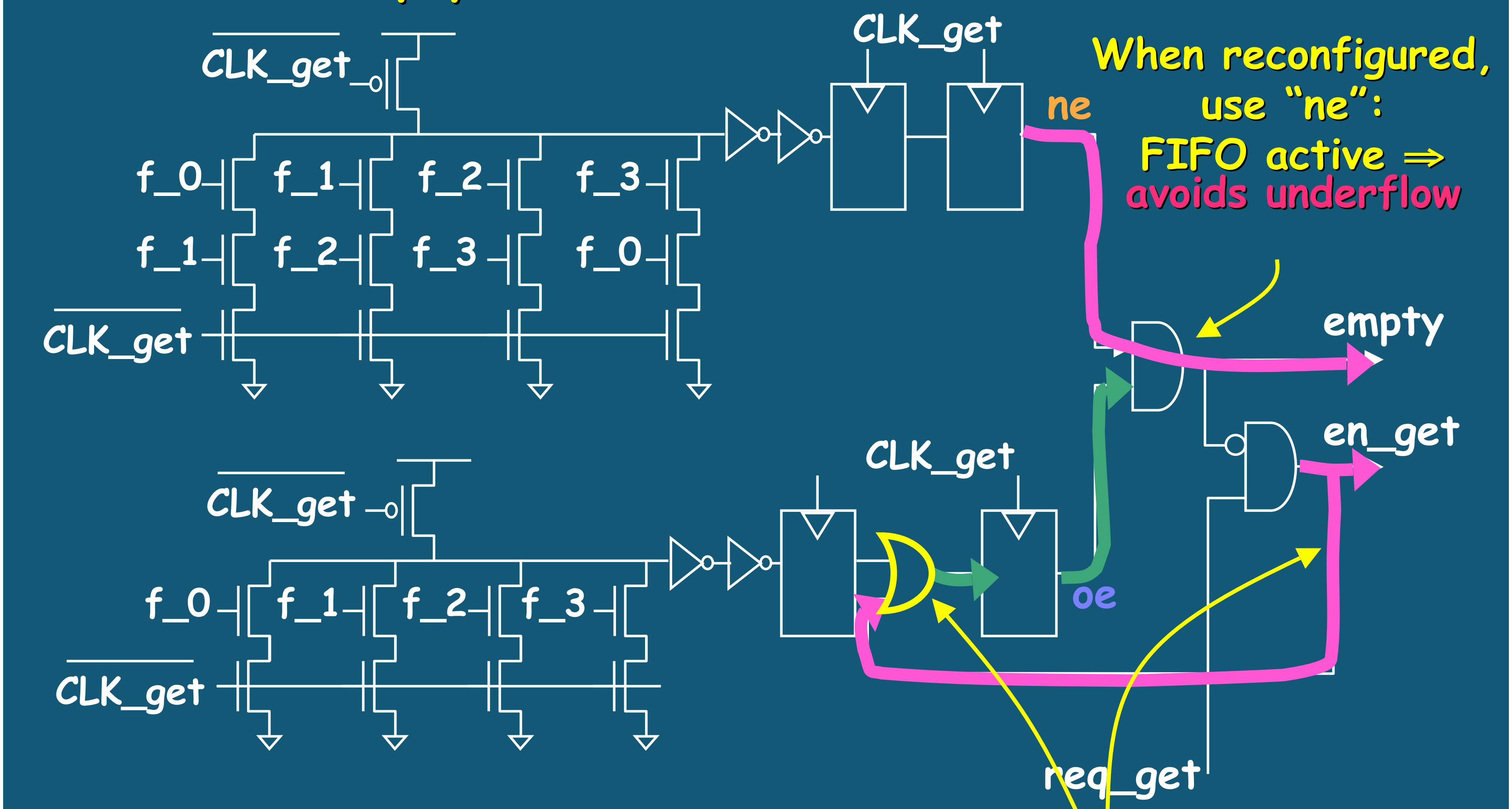
Bi-modal empty detection: select either ne or oe



Reconfigure whenever active get interface

Mixed-Clock FIFO: Deadlock Avoidance

Bi-modal empty detection:

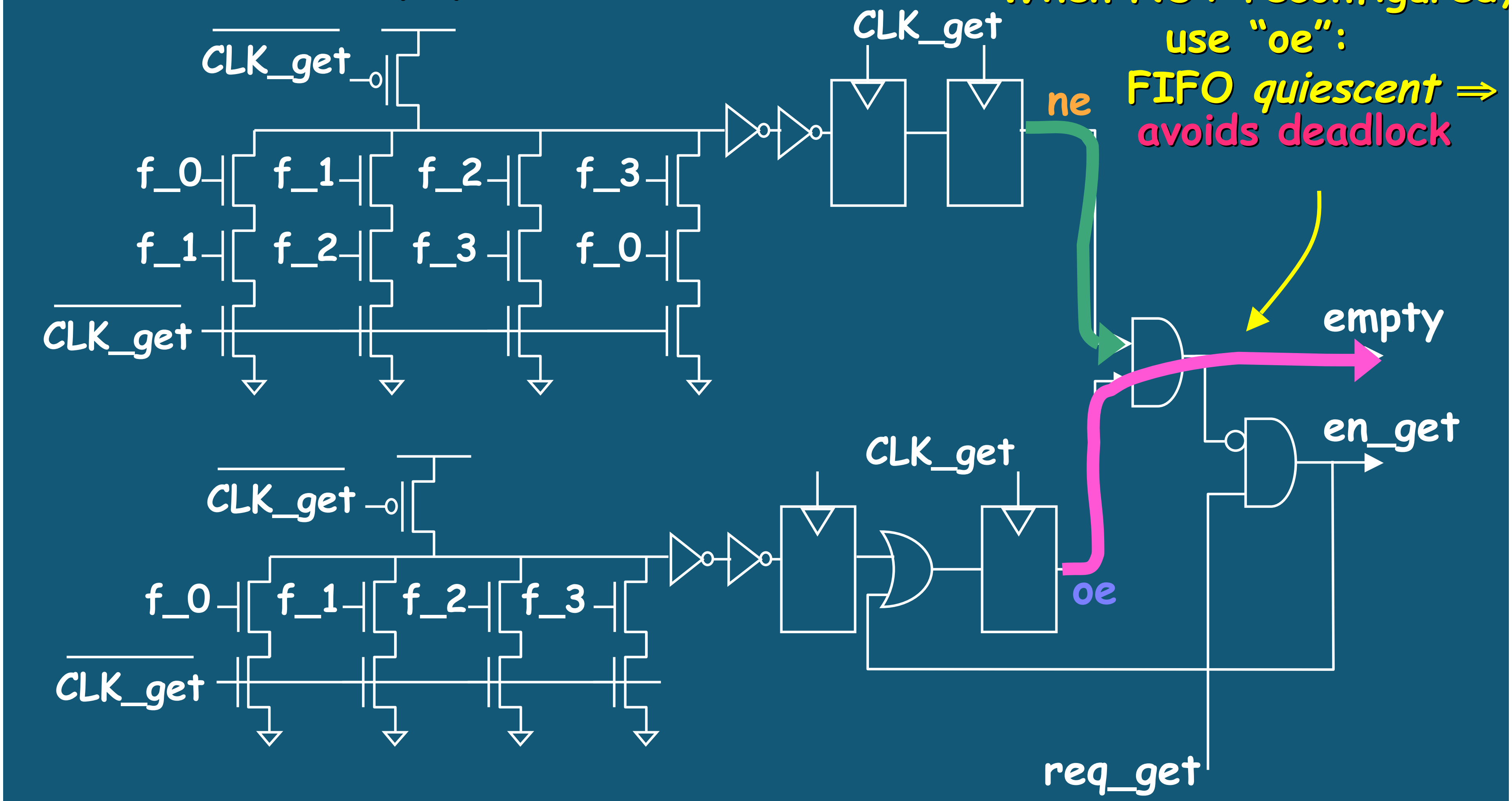


When reconfigured, use "ne": FIFO active ⇒ avoids underflow

Reconfigure whenever *active get interface*

Mixed-Clock FIFO: Deadlock Avoidance

Bi-modal empty detection:



When NOT reconfigured, use "oe": FIFO *quiescent* ⇒ avoids deadlock

Related Work: Intel Mixed-Clock Synchronizer

Intel Patent [1997]: J. Jex, C. Dike, K. Self (5,598,113)

- Similar FIFO structure
- Similar notion of “almost full”/”almost empty”

Differences/Limitations: N-stage FIFO

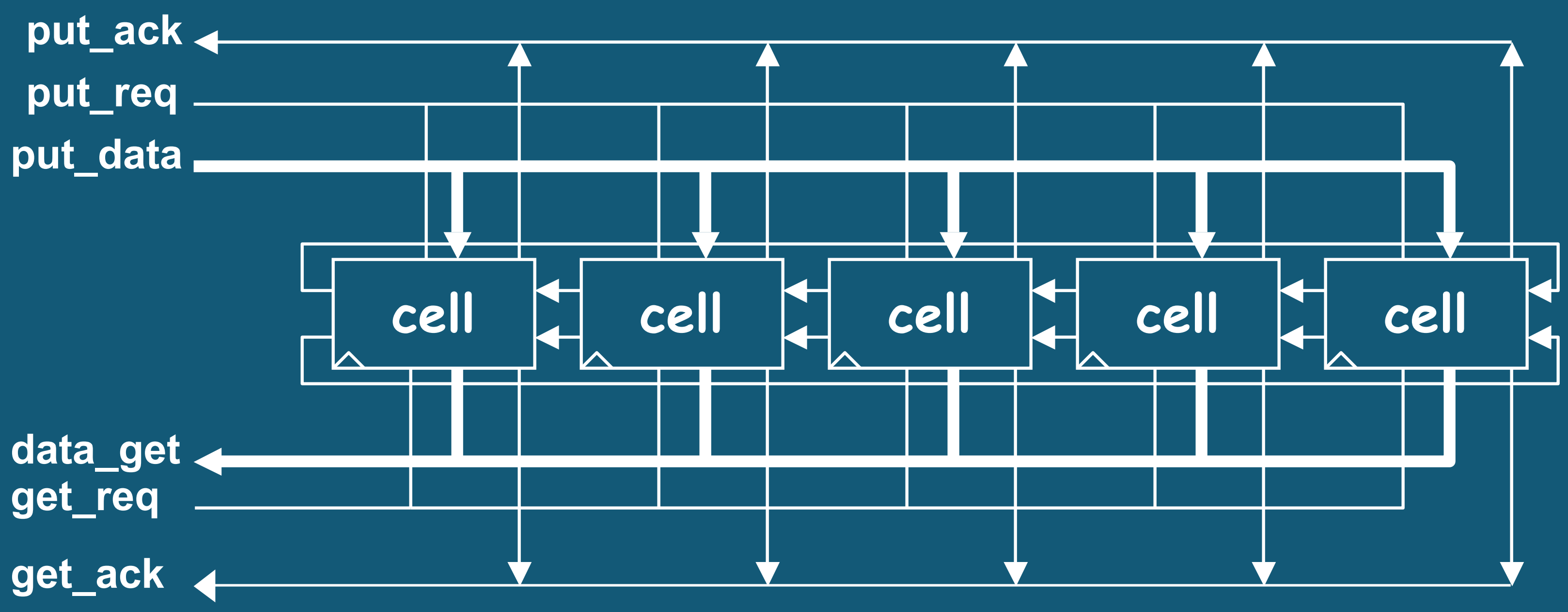
# synchronizers required:

- INTEL: N+1
- US: 3

Interface types:

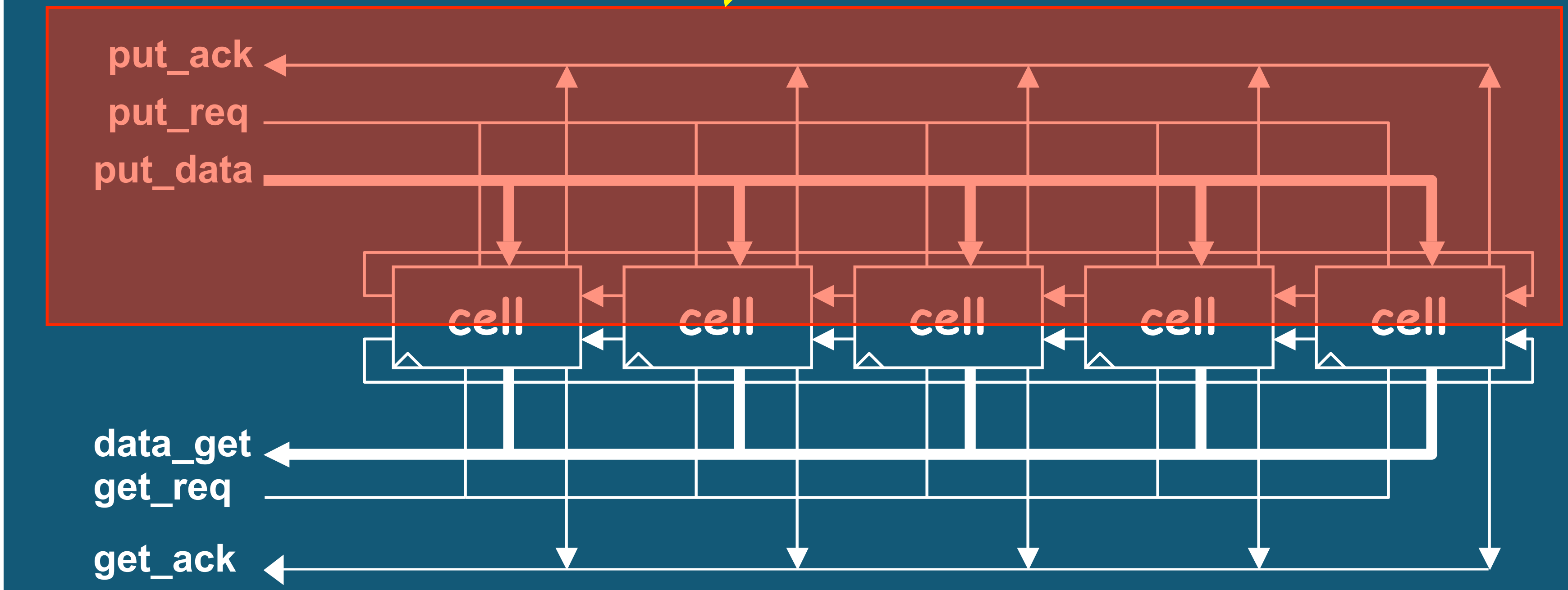
- INTEL: only sync-sync
- US: introduce a complete family (sync+async combinations)

Async-Async FIFO: Architecture

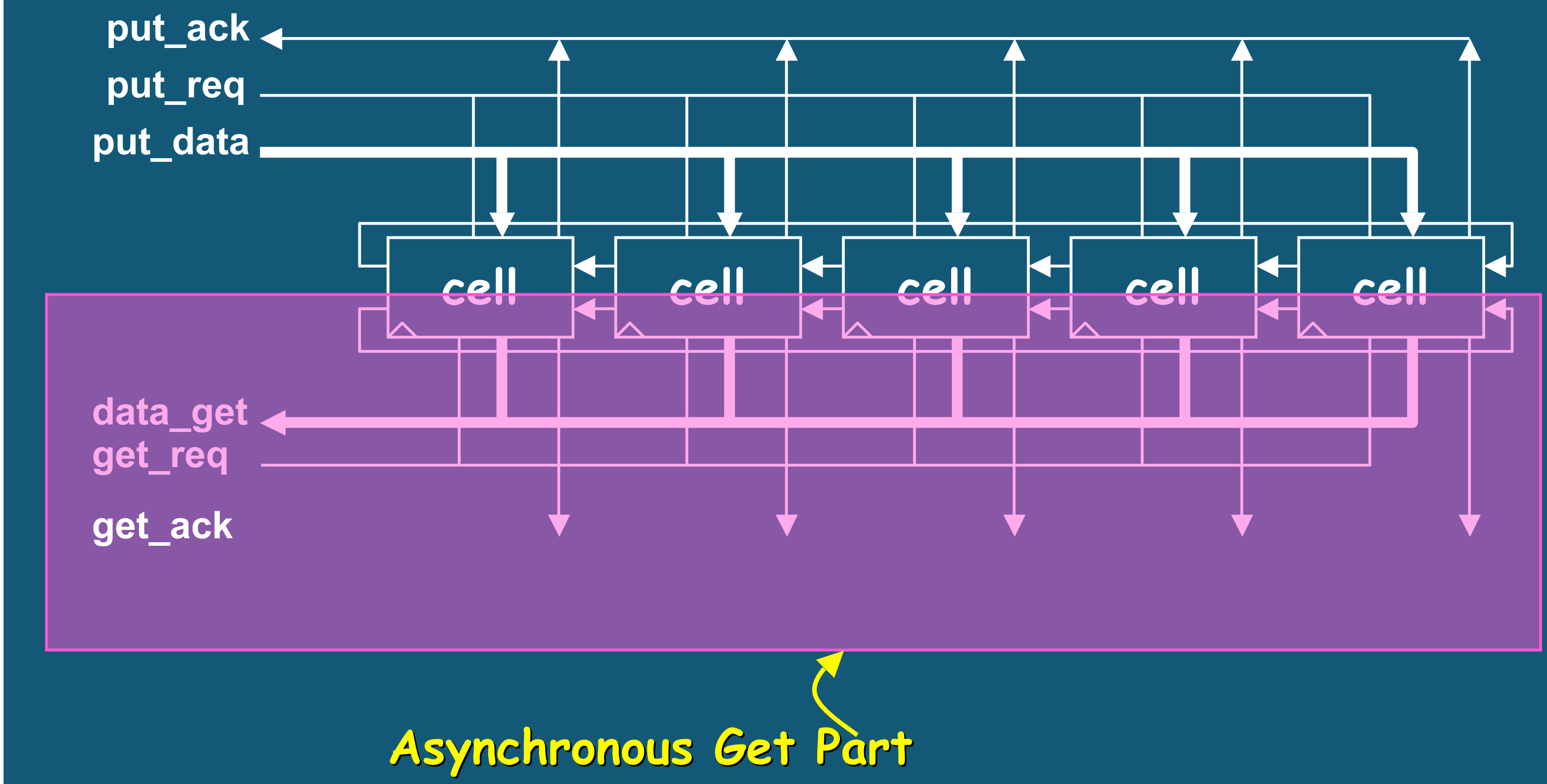


Async-Async FIFO: Architecture

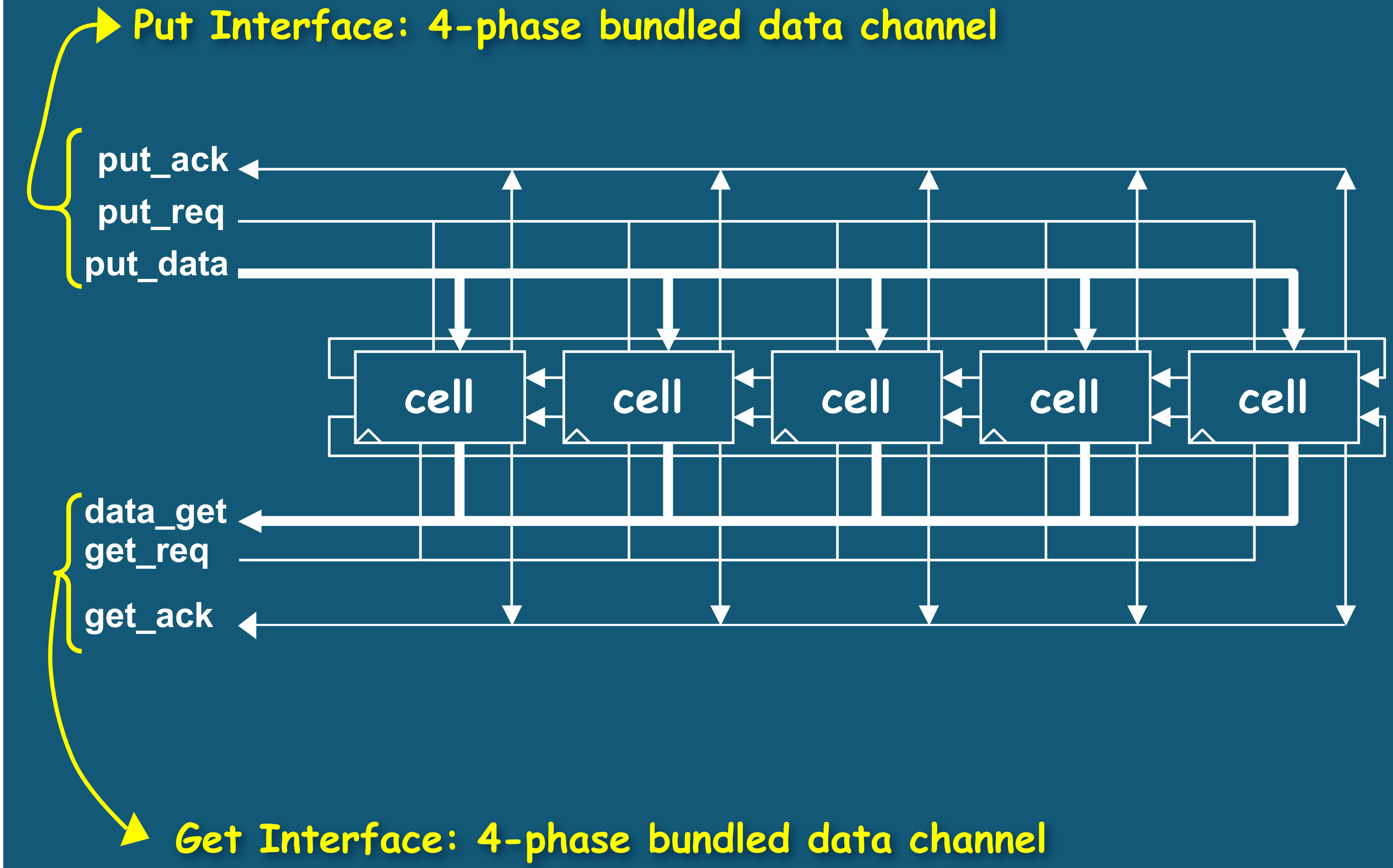
Asynchronous Put Part



Async-Async FIFO: Architecture



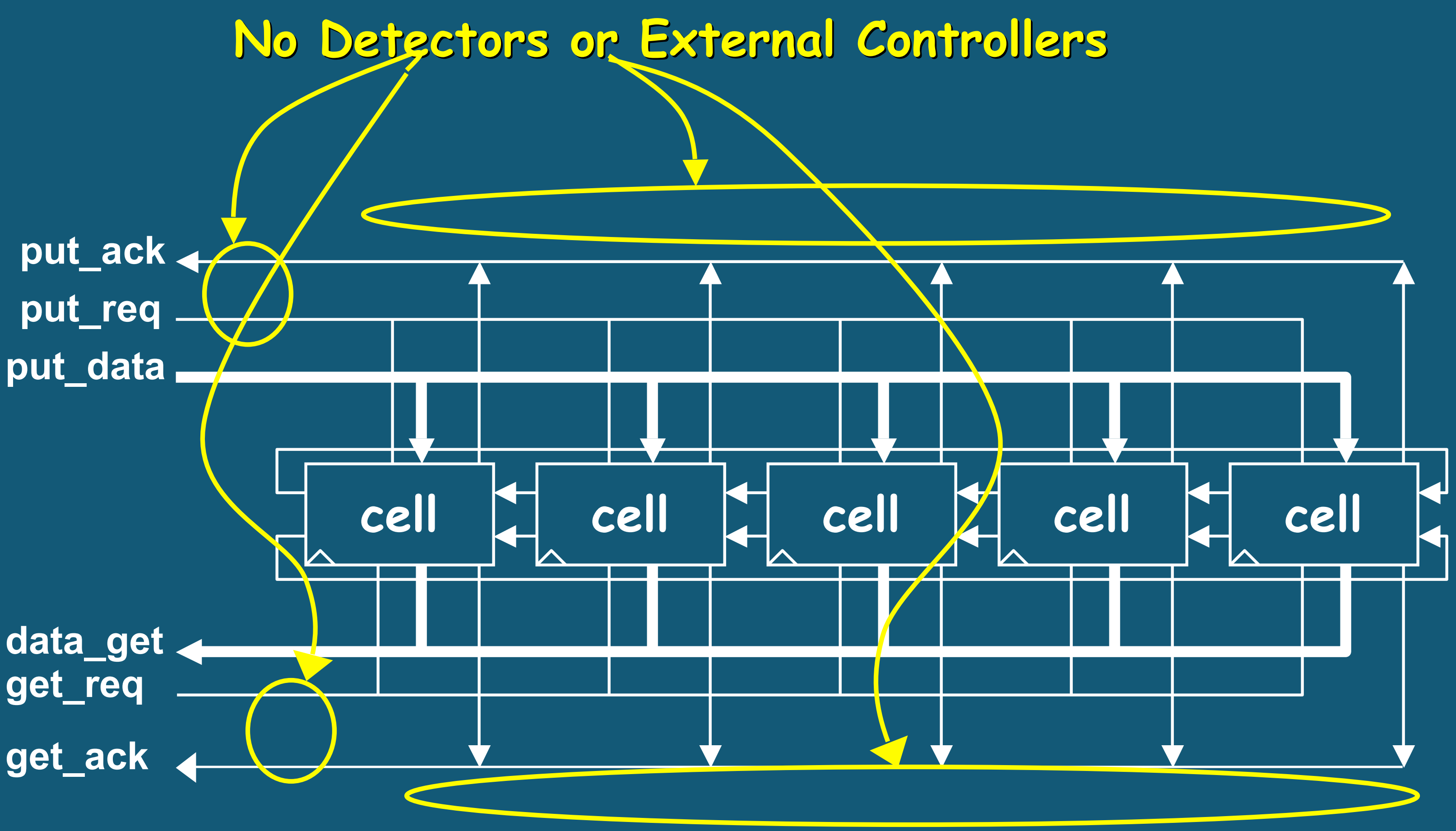
Async-Async FIFO: Architecture



Put Interface: 4-phase bundled data channel
put_ack
put_req
put_data
cell
cell
cell
cell
cell
data_get
get_req
get_ack
Get Interface: 4-phase bundled data channel

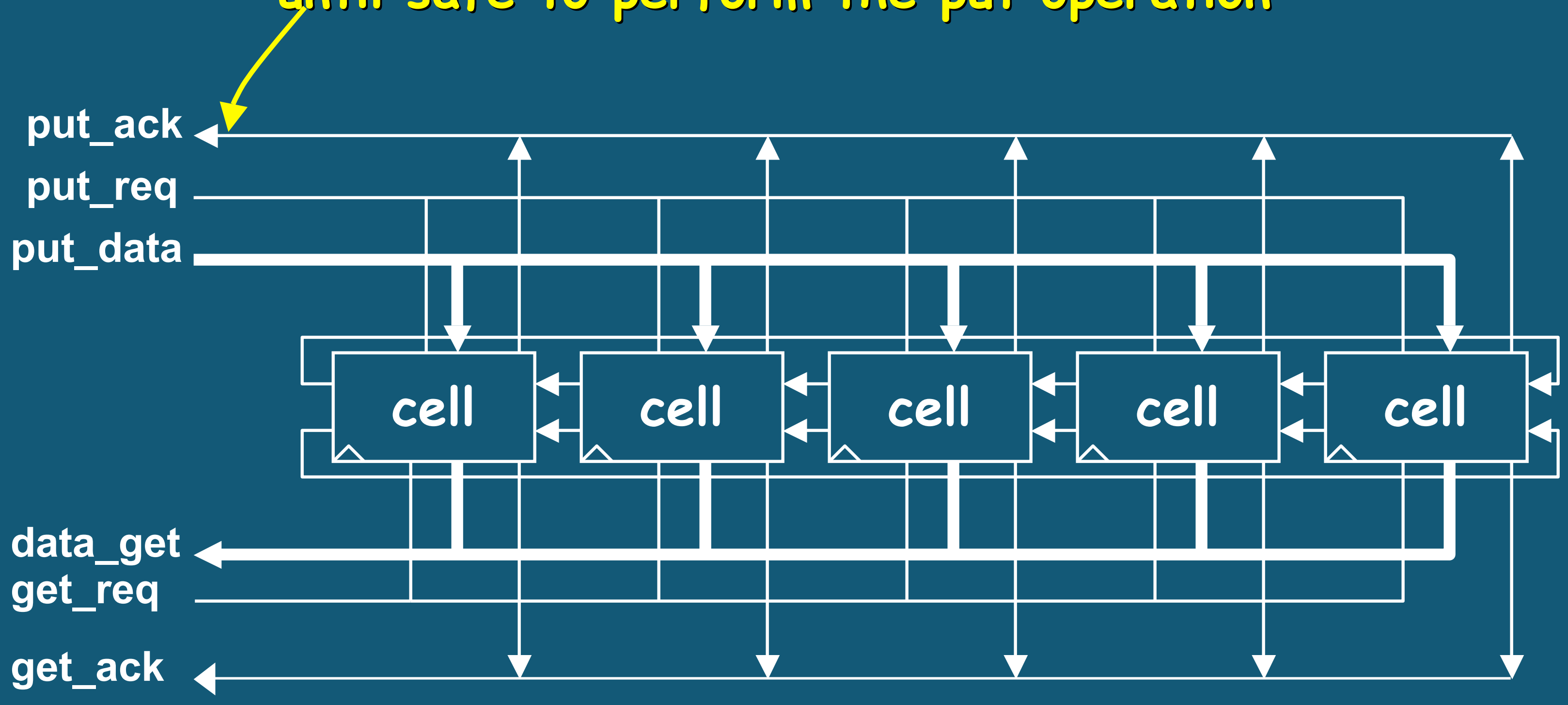
Async-Async FIFO: Architecture

No Detectors or External Controllers

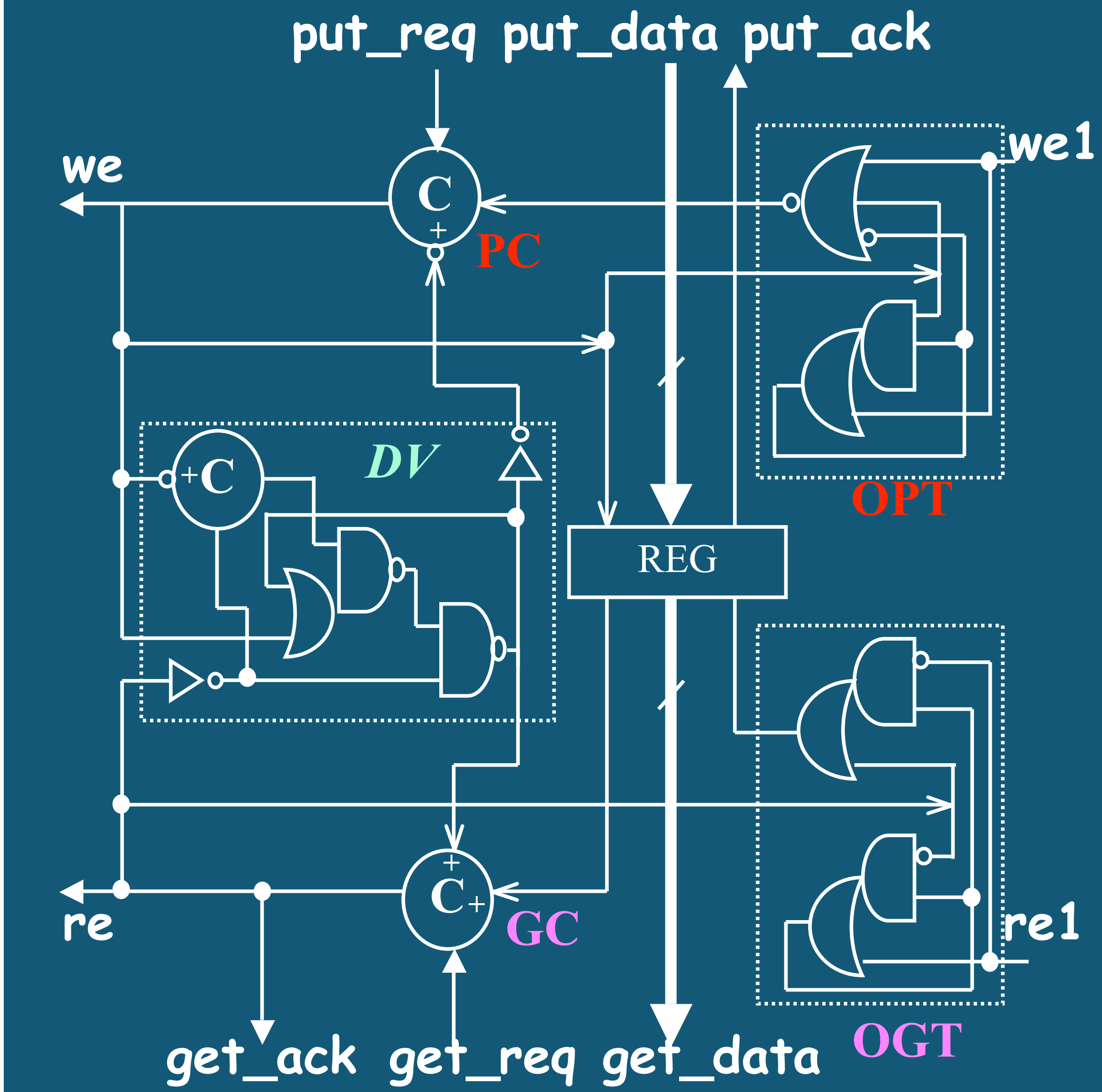


Async-Async FIFO: Architecture

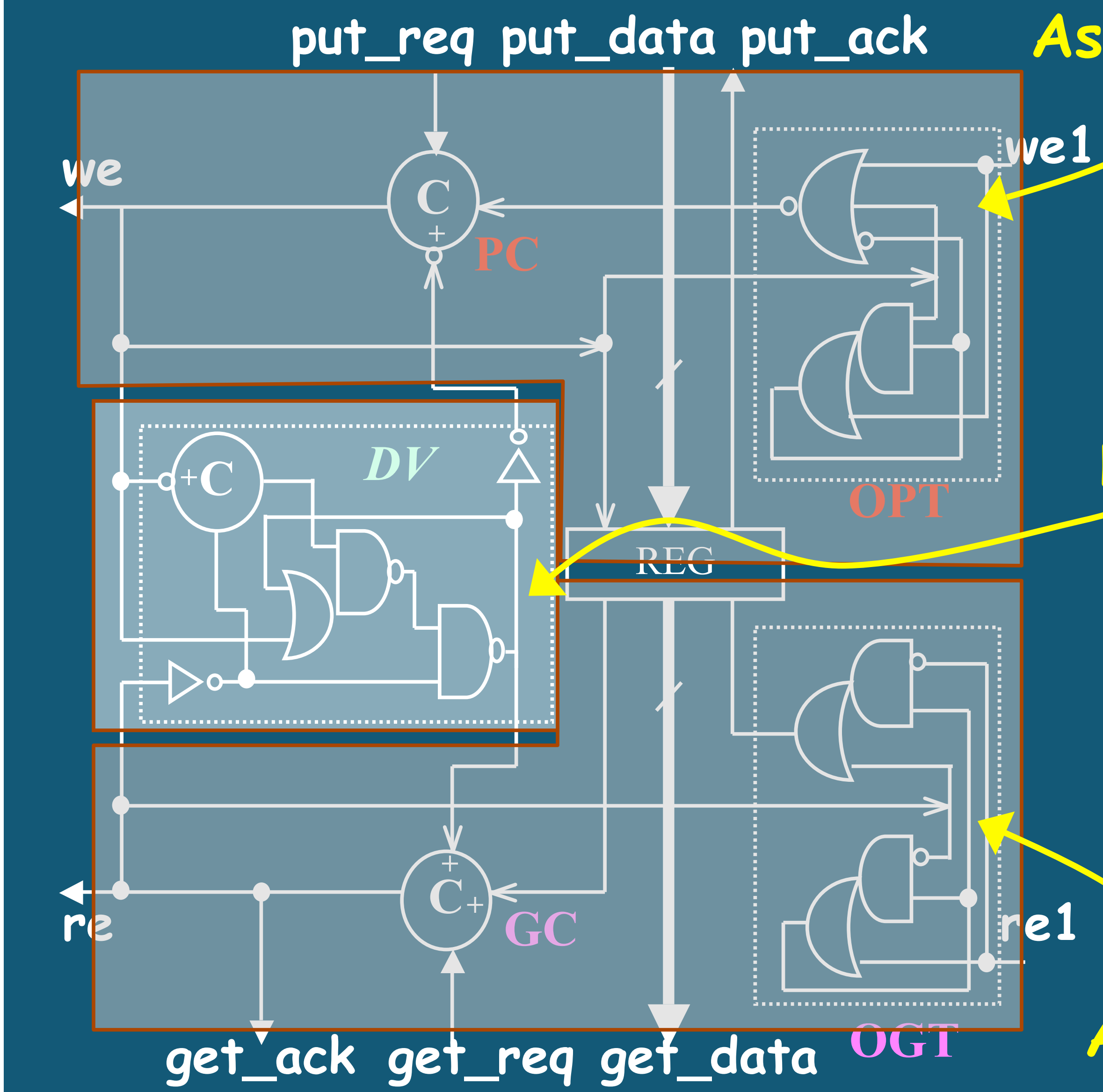
When FIFO full, acknowledgment withheld until safe to perform the put operation



Async-Async FIFO Cell



Async-Async FIFO Cell



Asynchronous Put Part

reusable

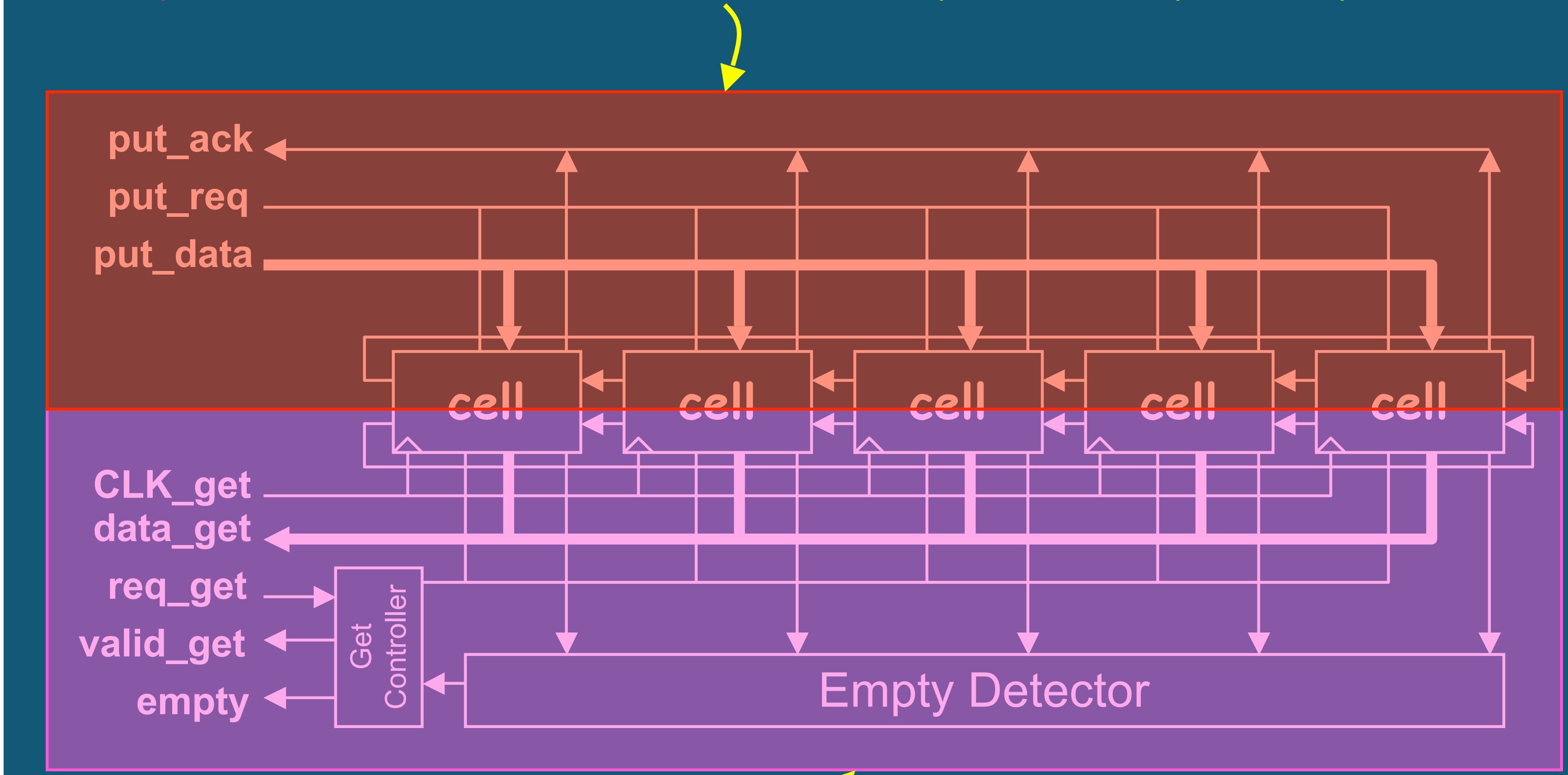
Data Validity Controller

reusable

Asynchronous Get Part

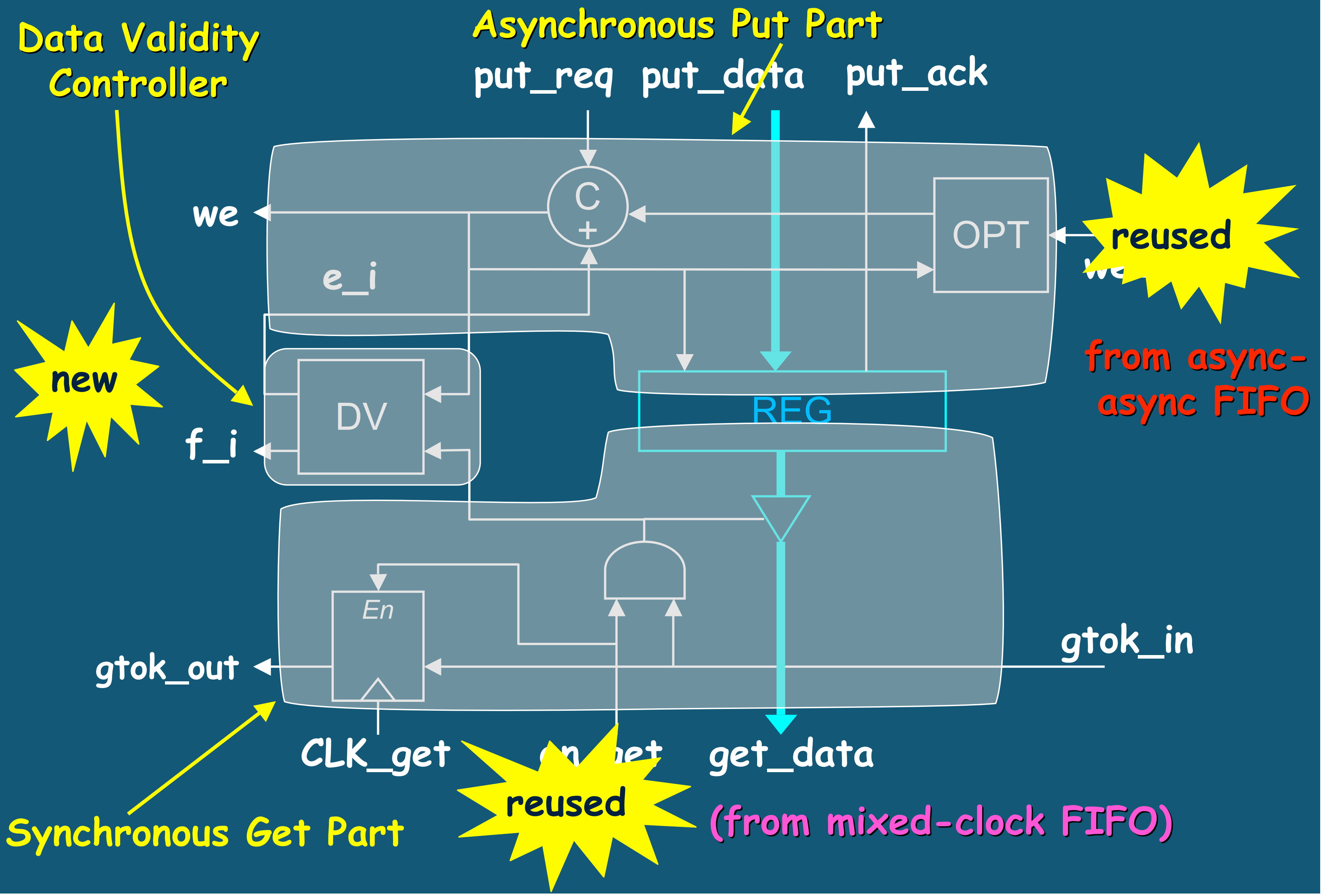
Reusability: Async-Sync FIFO Architecture

Asynchronous Put Interface: exactly as in Async-Async FIFO



Synchronous Get Interface: exactly as in Mixed-Clock FIFO

Reusability: Async-Sync FIFO Cell



Data Validity Controller

Asynchronous Put Part

new

reused

from async-async FIFO

Synchronous Get Part

reused

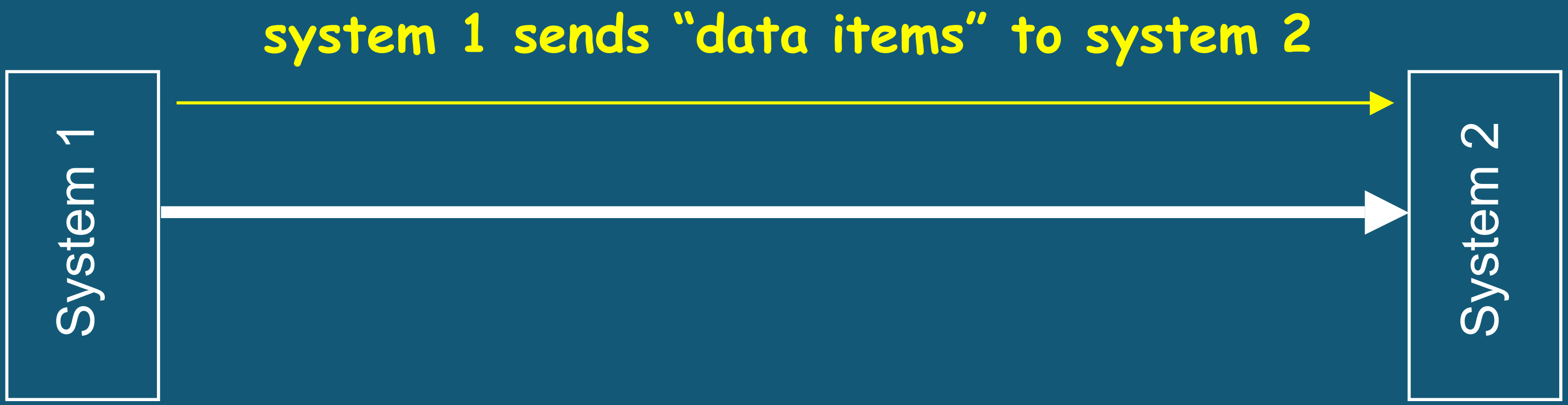
(from mixed-clock FIFO)

Part II

Handling Long Interconnect Delays

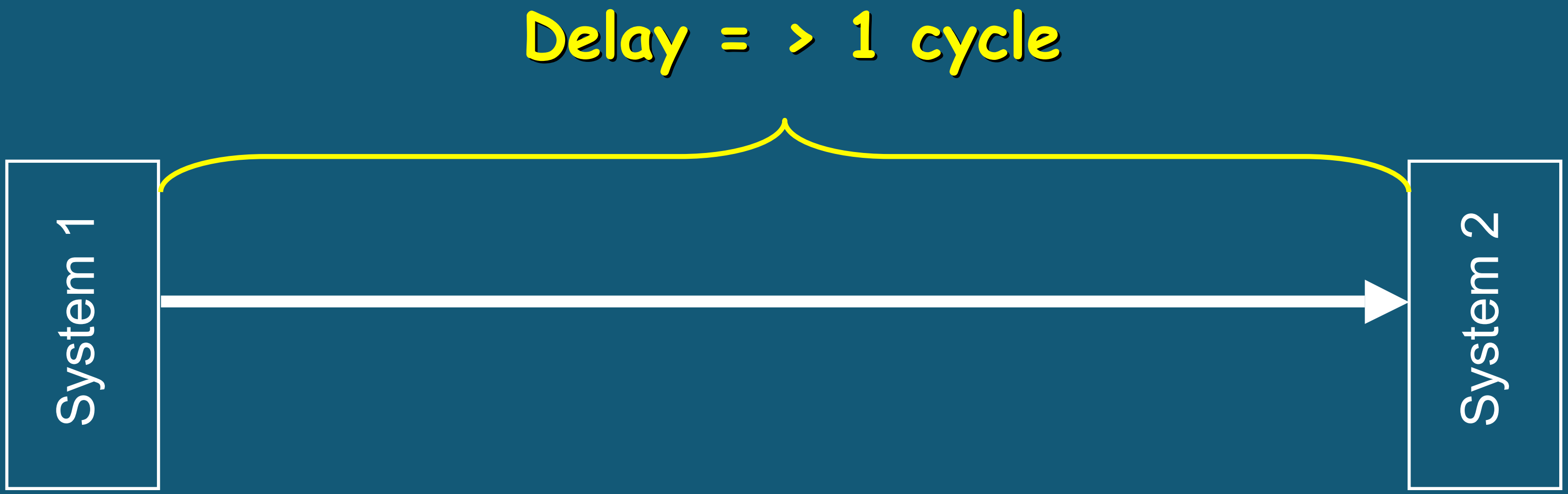
Issues in Handling Long Interconnect

Relay Stations: Background [Carloni, Sangiovanni-Vincentelli ’99]



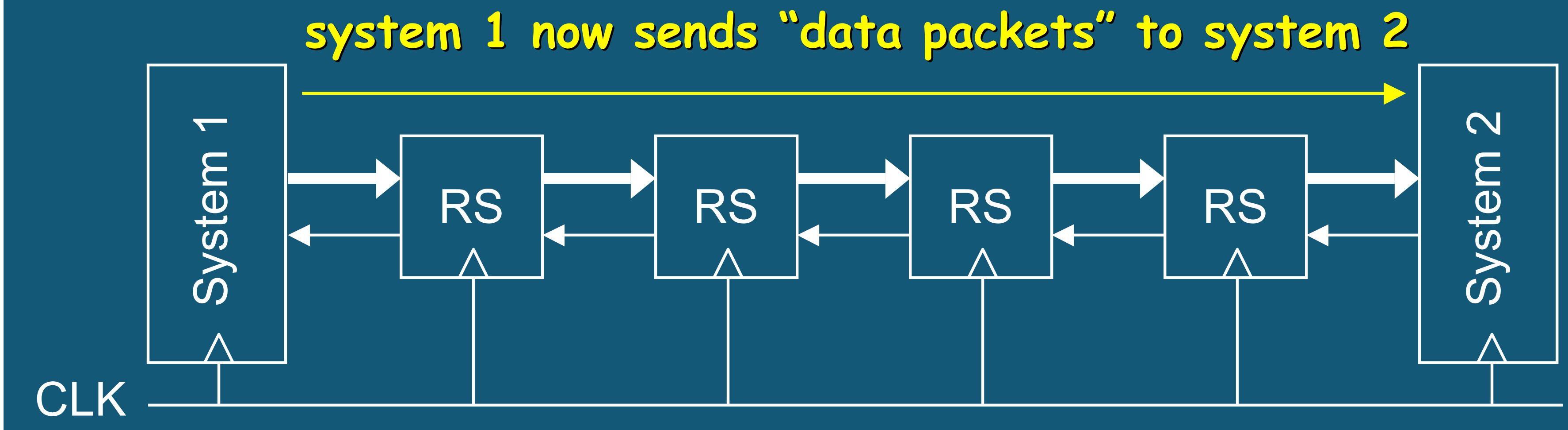
Issues in Handling Long Interconnect

Relay Stations Background [Carloni'99]



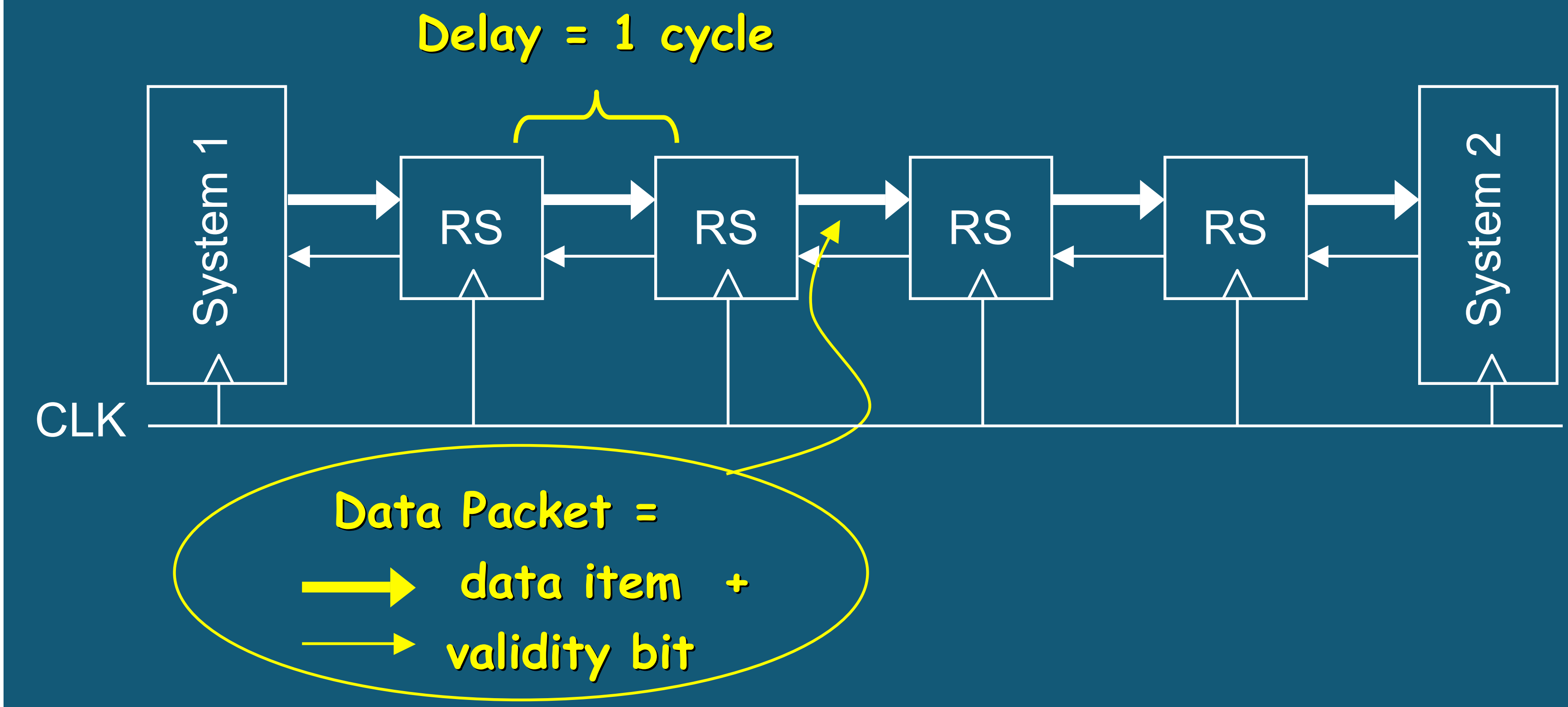
Issues in Handling Long Interconnect

Relay Stations Background [Carloni'99]



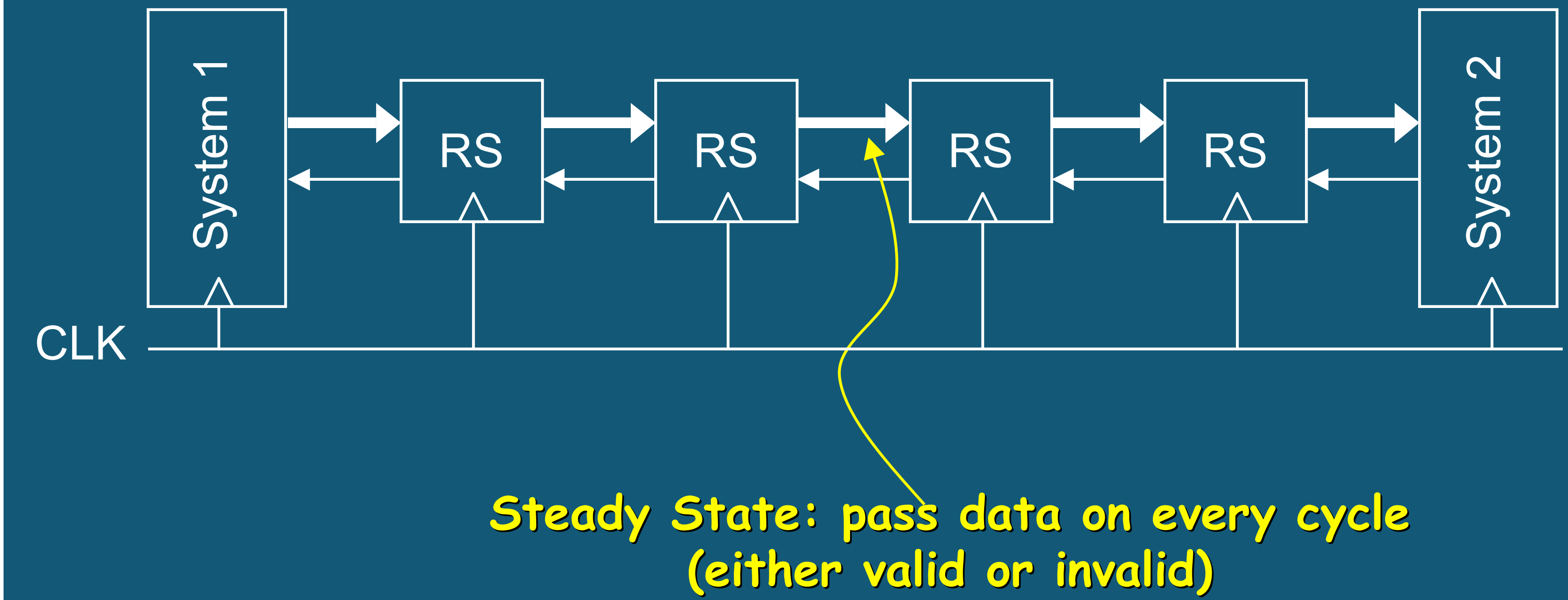
Issues in Handling Long Interconnect

Relay Stations Background [Carloni'99]



Issues in Handling Long Interconnect

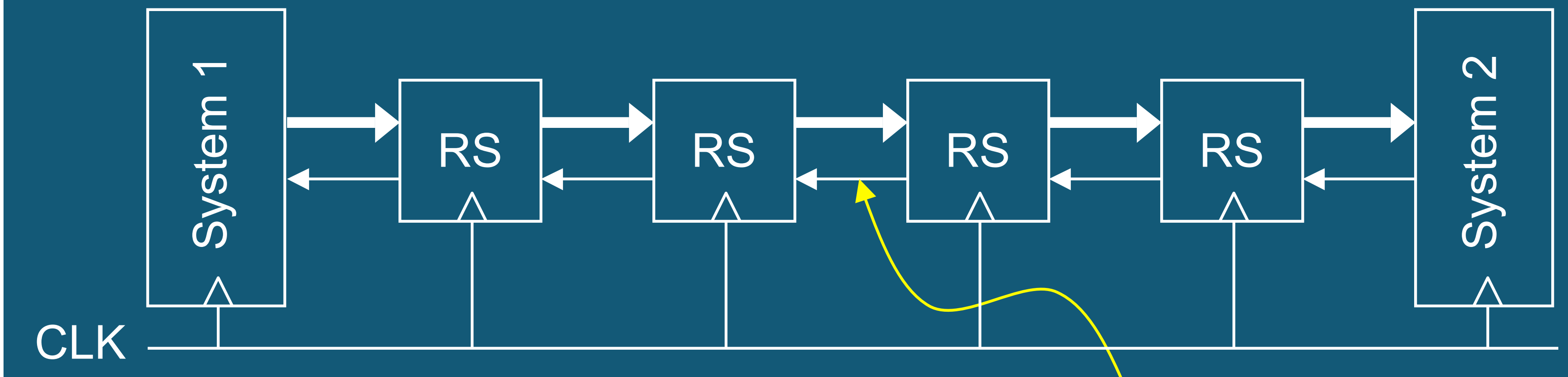
Relay Stations Background [Carloni'99]



Steady State: pass data on every cycle
(either valid or invalid)

Issues in Handling Long Interconnect

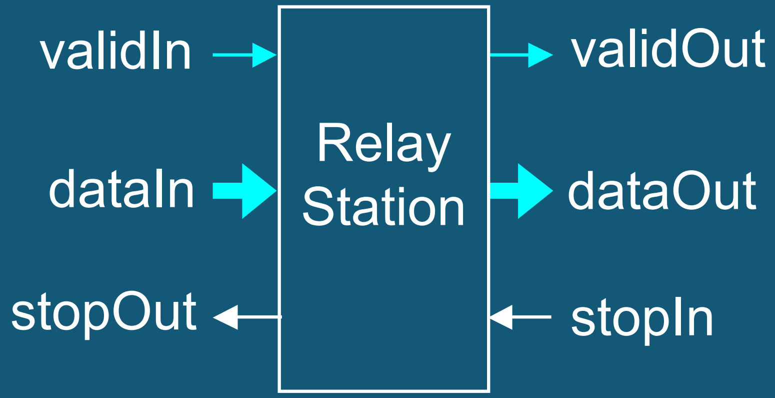
Relay Stations Background [Carloni'99]



"stop" control = stopIn + stopOut
- apply counter-pressure
- result: stall communication

Problem: Works only for single-clock systems!

Relay Station

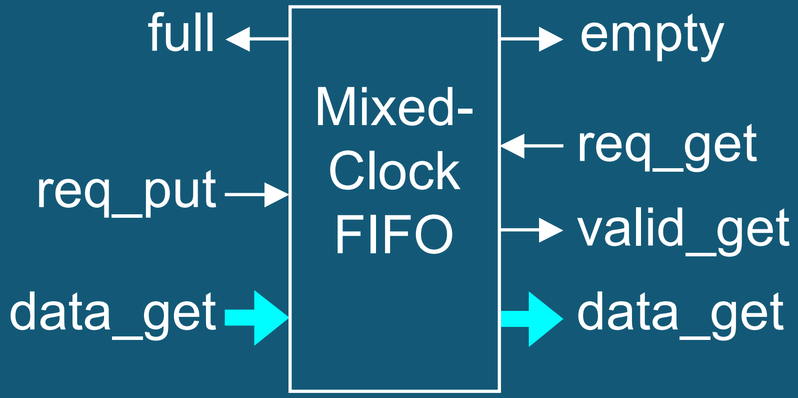


Steady state: always pass data

Data items: both valid & invalid

Stopping mechanism: stopIn & stopOut

Mixed-Clock FIFO

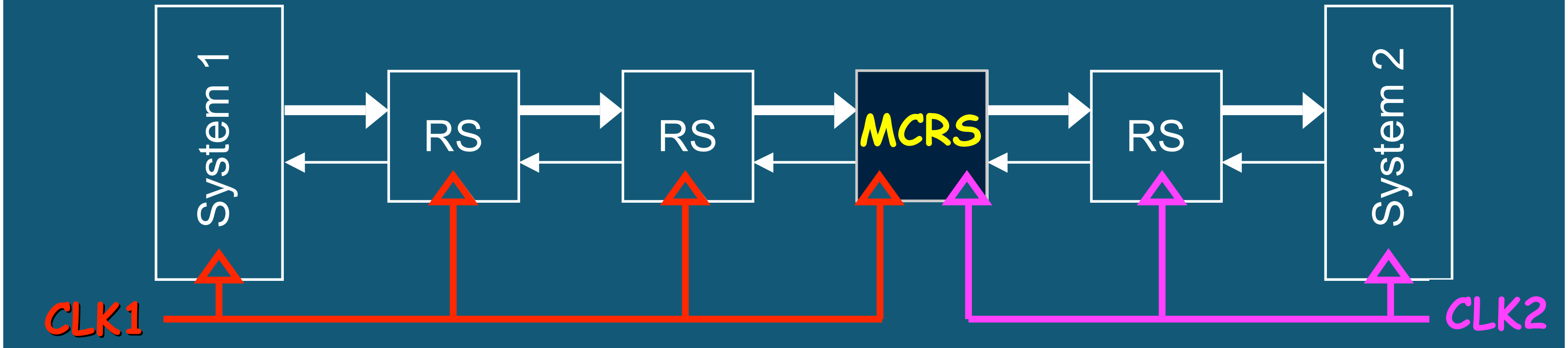


Steady state: only pass data when requested

Data items: only valid data

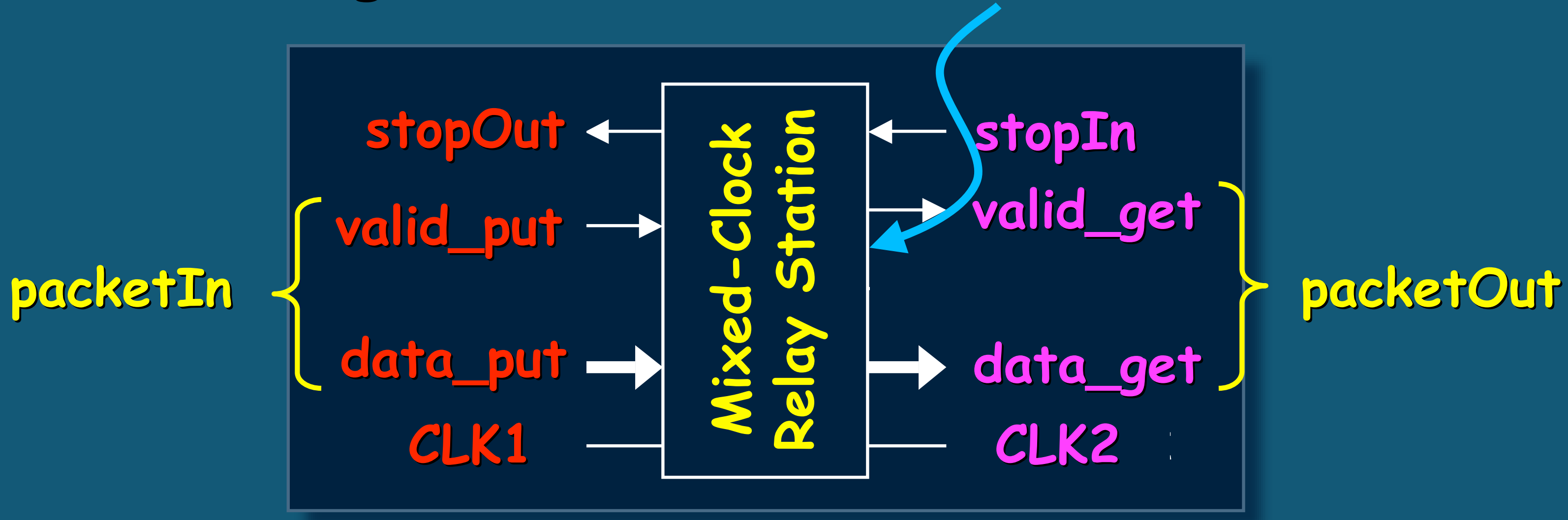
Stopping mechanism: none (only full/empty)

Mixed-Clock Relay Stations (MCRS)



Mixed-Clock Relay Station: derived from Mixed-Clock FIFO

Change ONLY Put and Get Controllers



Part III

Experimental Results

Preliminary Results

Each new Mixed-Timing FIFO designed:

- using both academic and industry tools
  - MINIMALIST: Burst-Mode controllers [Nowick et al. '99]
  - PETRIFY: Petri-Net controllers [Cortadella et al. '97]

Pre-layout simulations in 0.6μm HP CMOS technology

Experiments:

- various FIFO capacities (4/16 cells)
- 8-bit data items

Preliminary Results: Latency

Experimental setup: 8-bit data items + various FIFO capacities (4, 16)

Latency = time from enqueuing to dequeueing data into an empty FIFO

Version	4-place		16-place	
	Min	Max	Min	Max
Mixed-Clock FIFO	5.43	6.34	6.14	7.17
Async-Async FIFO	1.73		2.29	
Async-Sync FIFO	5.53	6.45	6.47	7.51
Sync-Async FIFO	1.95		2.44	
Mixed-Clock RS	5.48	6.41	6.23	7.28
Async-Sync RS	5.61	6.35	6.57	7.62
Sync-Async RS	1.86		2.43	

Sync receiver ⇒ latency not uniquely defined: Min/Max

Preliminary Results: Latency

Experimental setup: 8-bit data items + various FIFO capacities (4, 16)

Latency = time from enqueuing to dequeueing data into an empty FIFO

Version	4-place		16-place	
	Min	Max	Min	Max
Mixed-Clock FIFO	5.43	6.34	6.14	7.17
Async-Async FIFO	1.73		2.29	
Async-Sync FIFO	5.53	6.45	6.47	7.51
Sync-Async FIFO	1.95		2.44	
Mixed-Clock RS	5.48	6.41	6.23	7.28
Async-Sync RS	5.61	6.35	6.57	7.62
Sync-Async RS	1.86		2.43	

Async receiver ⇒ lower, unique latency, no synchronization

Preliminary Results: Maximum Operating Rate

Synchronous interfaces: MegaHertz
Asynchronous interfaces: MegaOps/sec

Design	4-place		16-place	
	Put	Get	Put	Get
Mixed-Clock FIFO	565	549	505	484
Async-Async FIFO	423	454	359	357
Async-Sync FIFO	421	549	357	484
Sync-Async FIFO	565	454	505	360
Mixed-Clock RS	580	539	509	475
Async-Sync RS	421	539	357	475
Sync-Async RS	580	454	509	360

Put vs. Get rates:

- sync put faster than sync get
- async put slower than async get

Async vs. Sync rates:

- async slower than sync

Conclusions

Introduced complete family of mixed-timing FIFO’s :

- sync-sync, async-async, async-sync, sync-async
- create FIFO’s from *reusable parts*
- extend to handle issue of long interconnect delays

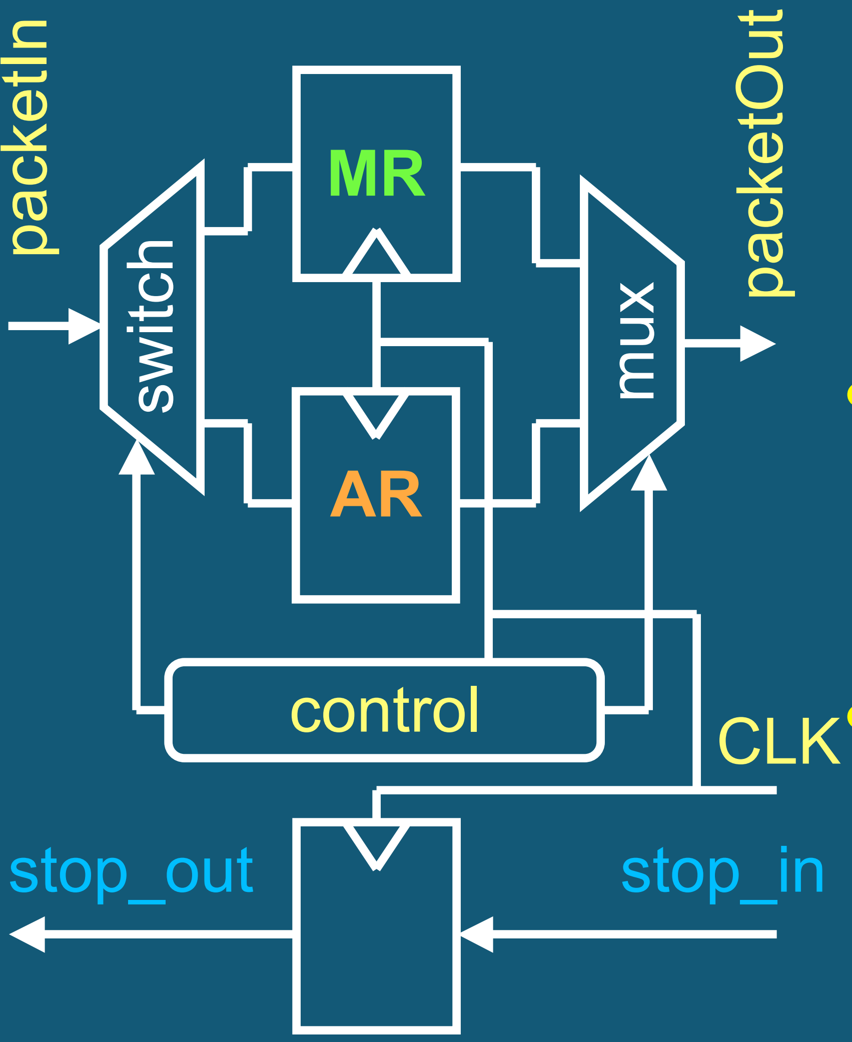
Characteristics:

- Low-latency
- Modular and scalable: distributed token-ring architecture
- High throughput:
  - steady state: no synchronization overhead, no failure probability
  - enqueue/dequeue data items: one/cycle
- Low area overheads: simple design

Extensions:

- **Deeper synchronizers (more latches) => arbitrary robustness**
- **powering down of inactive cells**

Relay Station: Implementation



- On every clock cycle:
  - packet latched in MR
  - packet passed to next RS by end of clock cycle
  - ***steady state:*** pass valid/invalid data items
- Station stopped from right:
  - stop_in = 1
  - extra packet latched in AR
  - stop_out = 1
- Station started (stop_in = 0):
  - Output packet in MR first
  - Output packet in AR second
  - Resume normal operation (stop_out = 0)