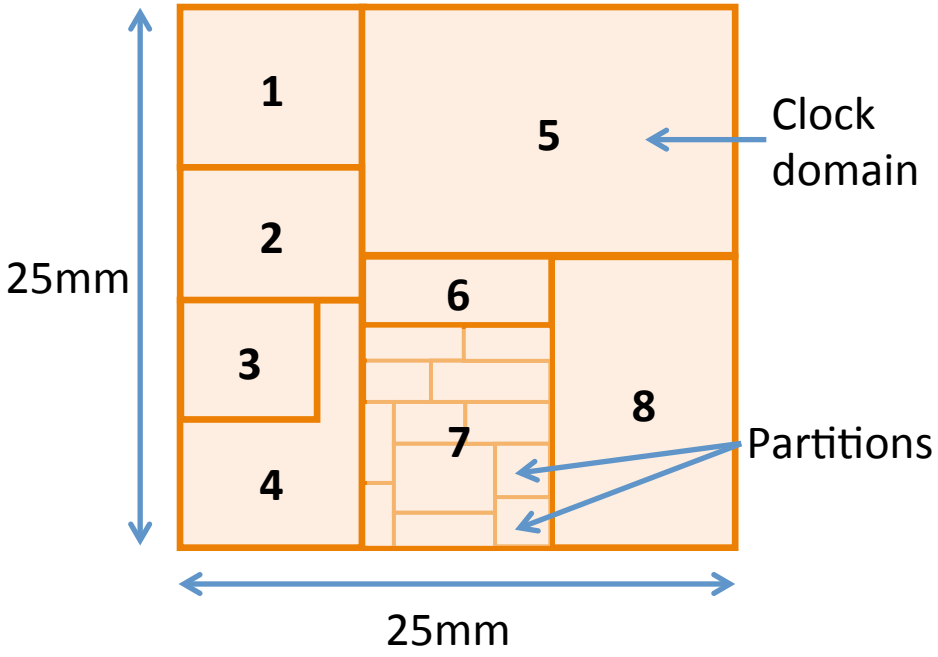
A Pausible Bisynchronous FIFO for GALS Systems

Ben Keller*†, Matthew Fojtik*, Brucek Khailany*

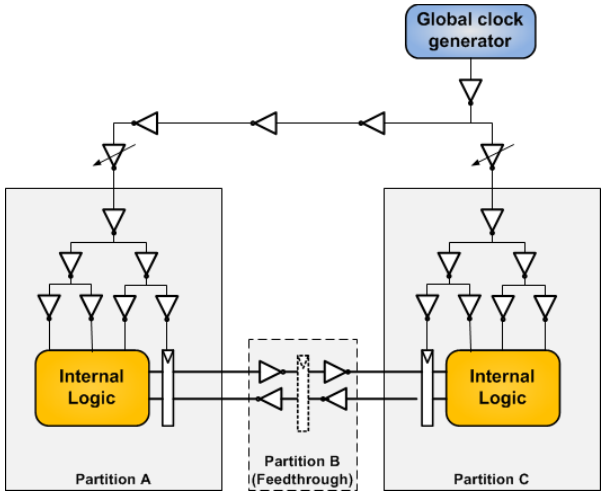
*NVIDIA Corporation †University of California, Berkeley

The Nightmare of Global Clocking

- Large chips have only a handful of clock domains
- Each clock domain is many mm^2, even in ≤28nm
  - Made up of many *partitions*



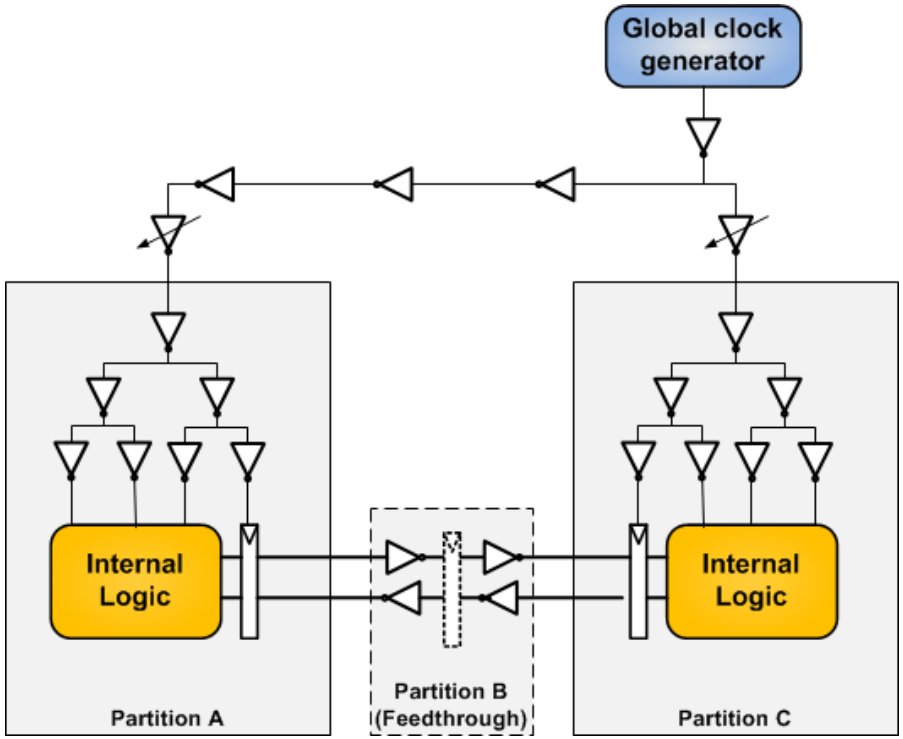
- Tapeout signoff currently requires distributing balanced synchronous clocks to every flip-flop in a clock domain…



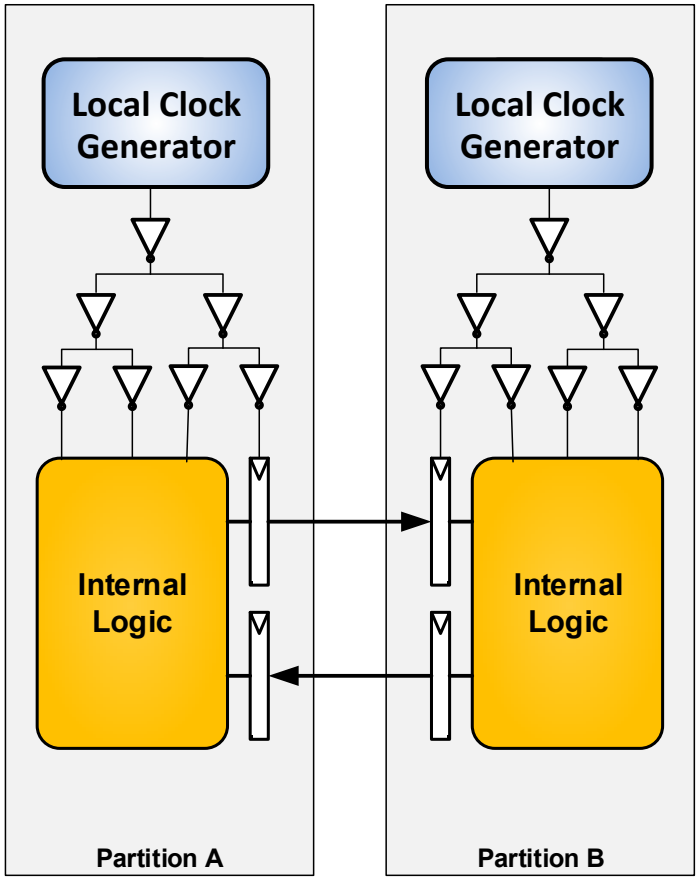
- …then closing SETUP and HOLD on all paths in the clock domain
  - Across dozens of PVT corners!

Globally Asynchronous, Locally Synchronous!

From this…

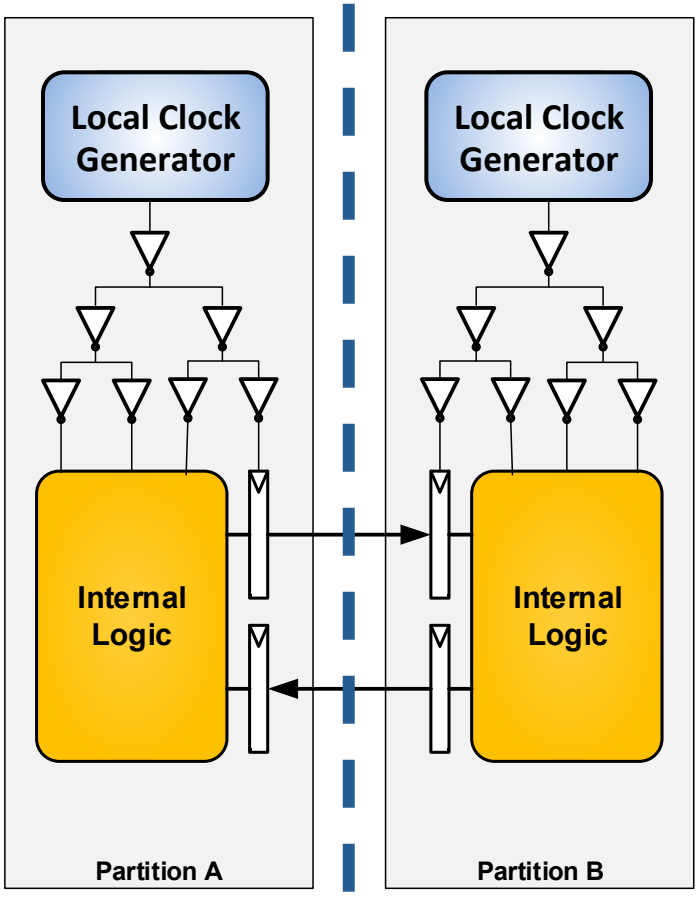


Globally Asynchronous, Locally Synchronous!



…to this.

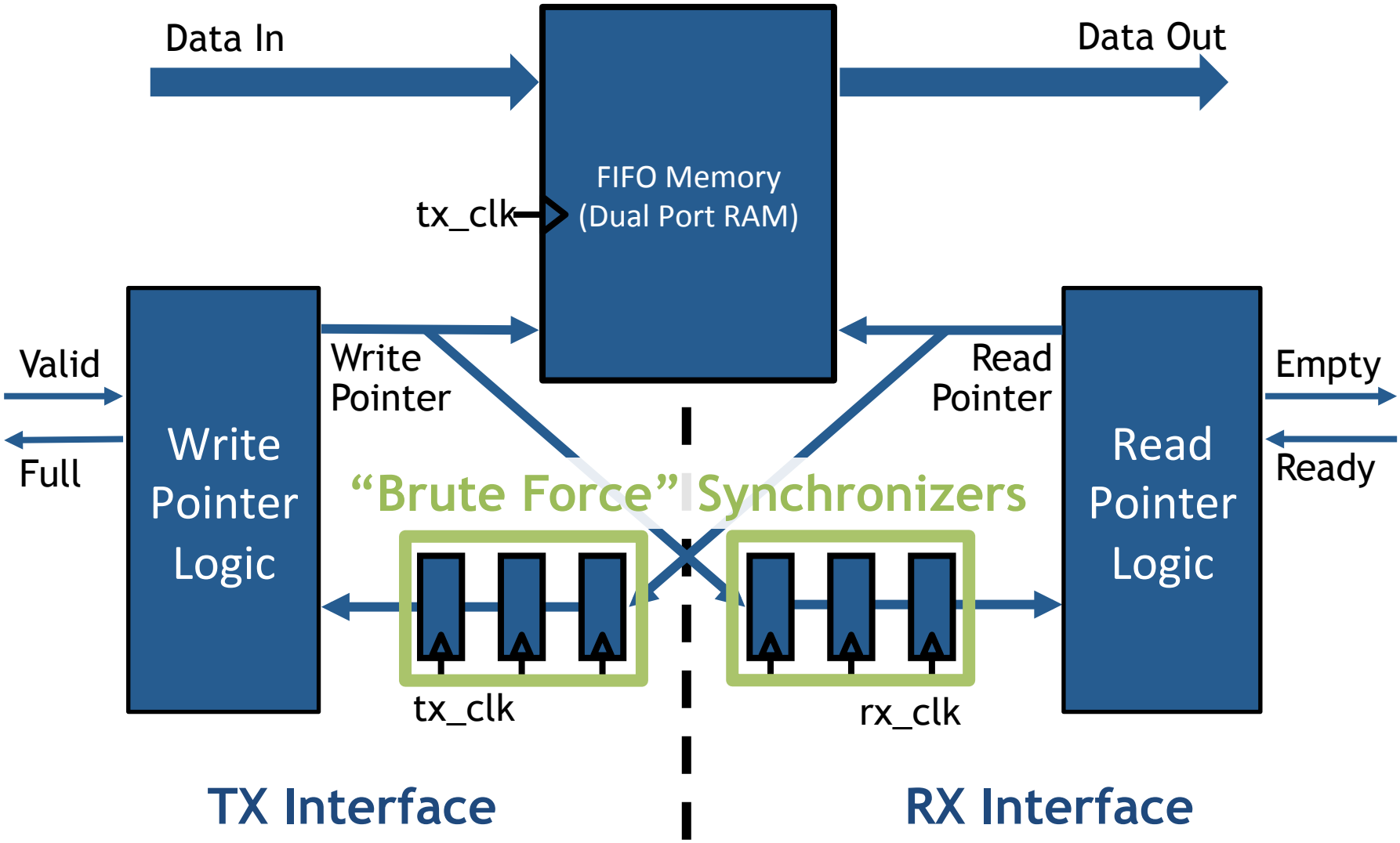
Towards Fine-Grained GALS



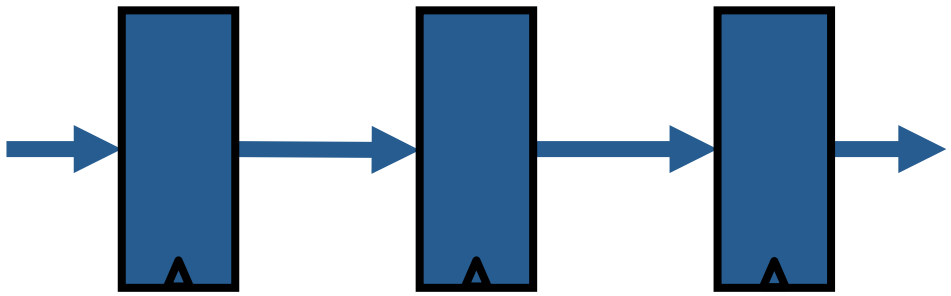
Clock Domain Crossing

- No global clock! Local ring oscillator generates clock in each partition
- No global timing closure
- Reduce timing margin
  - Tracks local PVT variation
  - May improve tracking of high-frequency noise
- But there's a catch…

Textbook Approach: Bisynchronous FIFO



The Brute Force Synchronizer

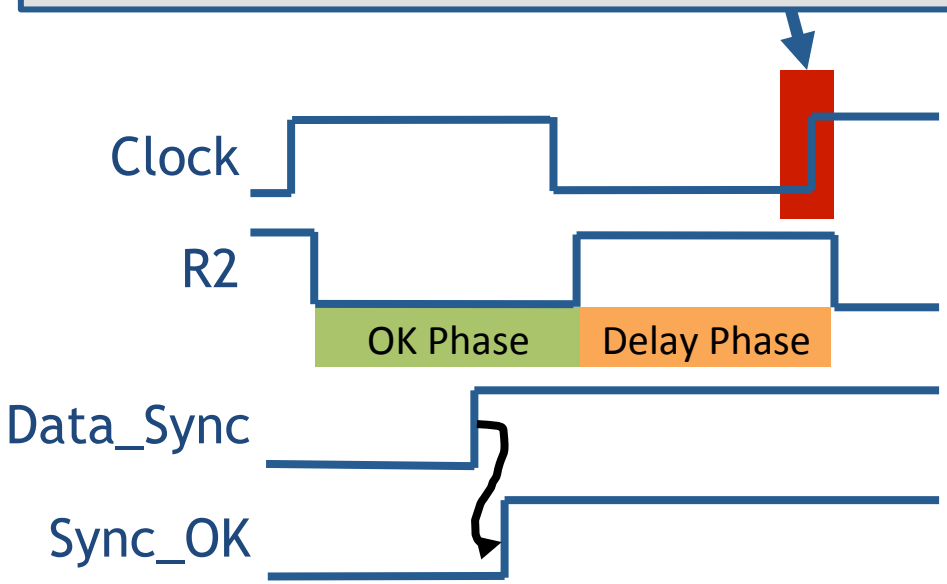


$$MTBF \propto e^{t}$$

- Add latency until metastability is most likely resolved
- +3 cycles of latency at every partition boundary!
- Need to do better…

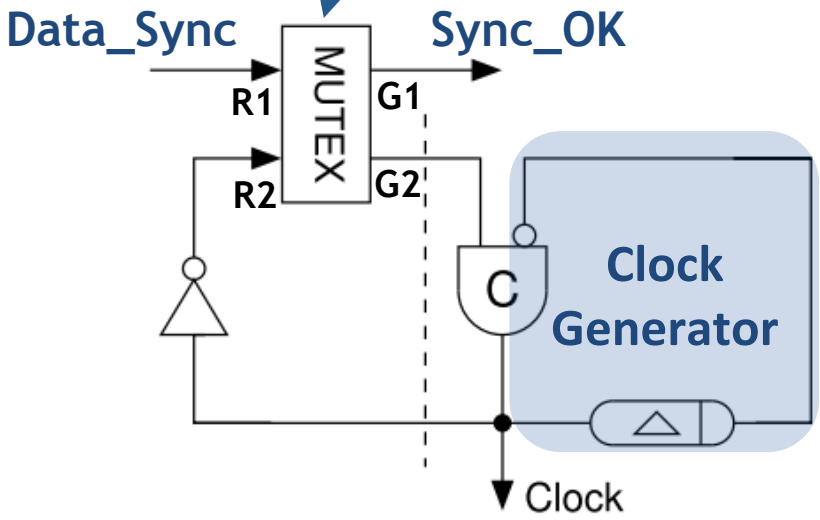
Pausible Clocking Basics

Metastability happens when Data_Sync transitions near the clock edge.



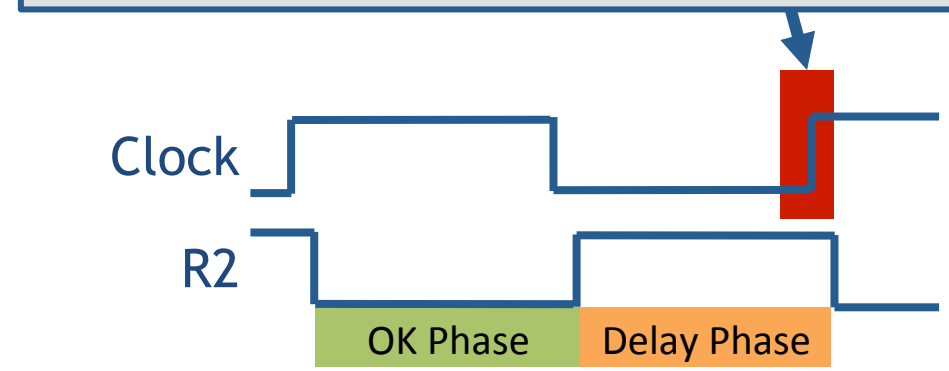
If data arrives when clock is high, let it through.

Mutual exclusion circuit only allows G1 OR G2 to go high (never both).



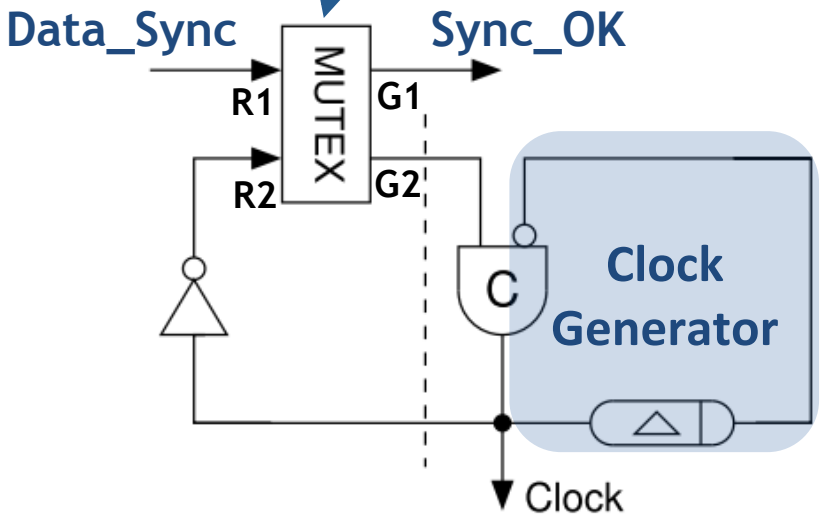
Pausible Clocking Basics

Metastability happens when Data_Sync transitions near the clock edge.



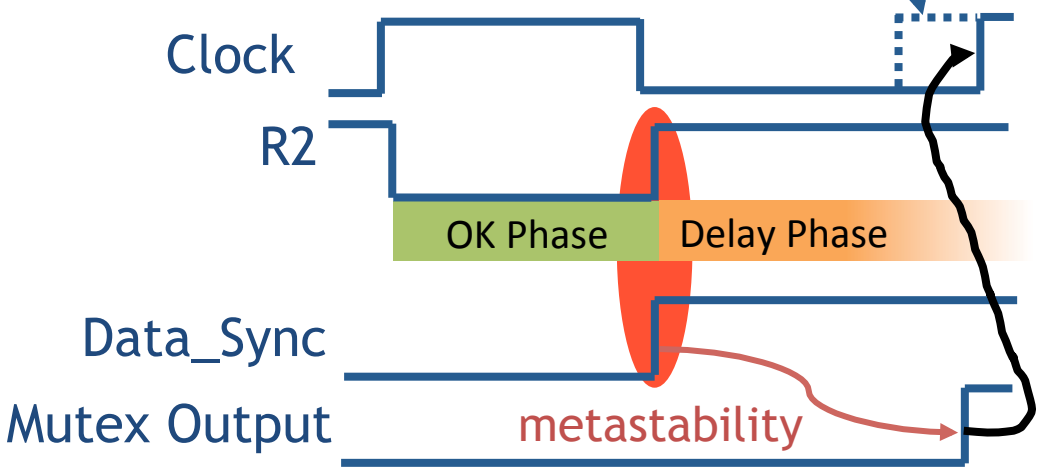
If data arrives when clock is **low**, delay it until after the clock edge.

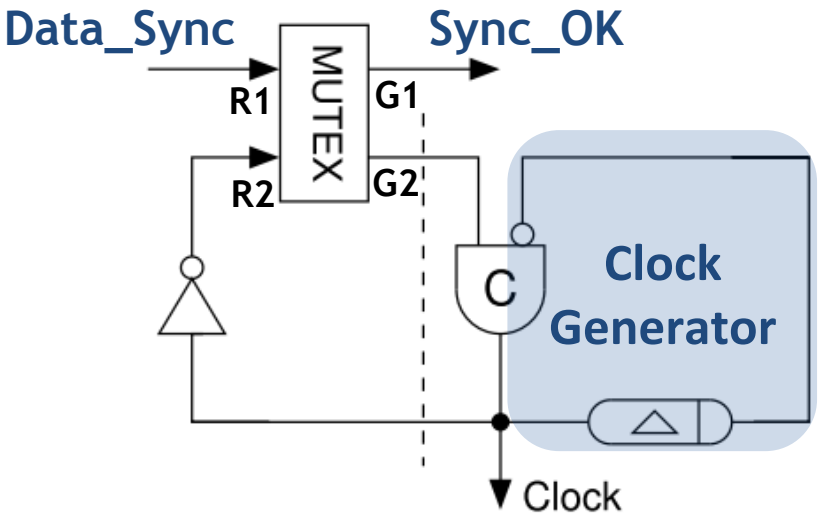
Mutual exclusion circuit only allows G1 OR G2 to go high (never both).



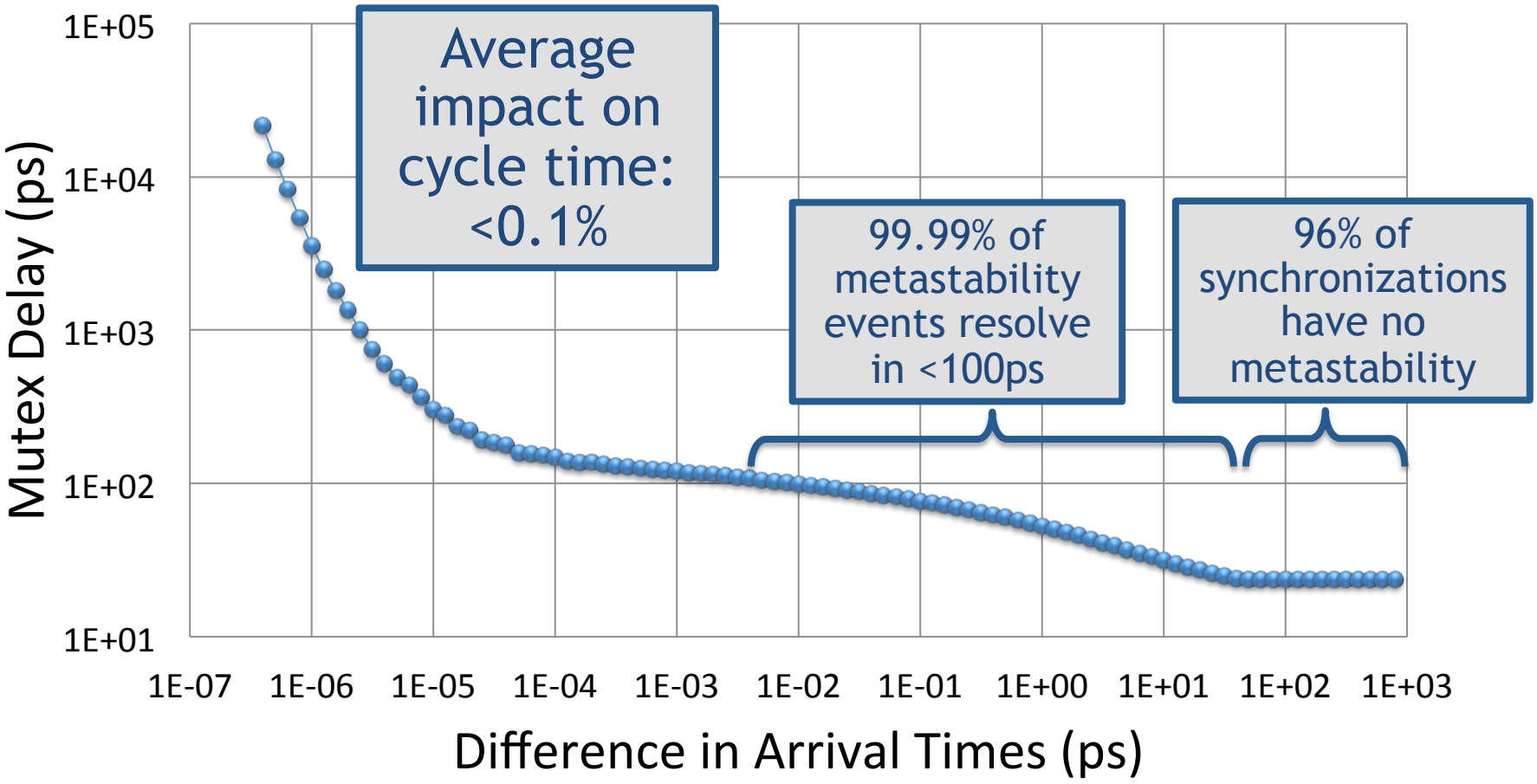
Metastability in Pausible Clocks

If the metastability lasts long enough, the next clock edge is delayed until it resolves safely.



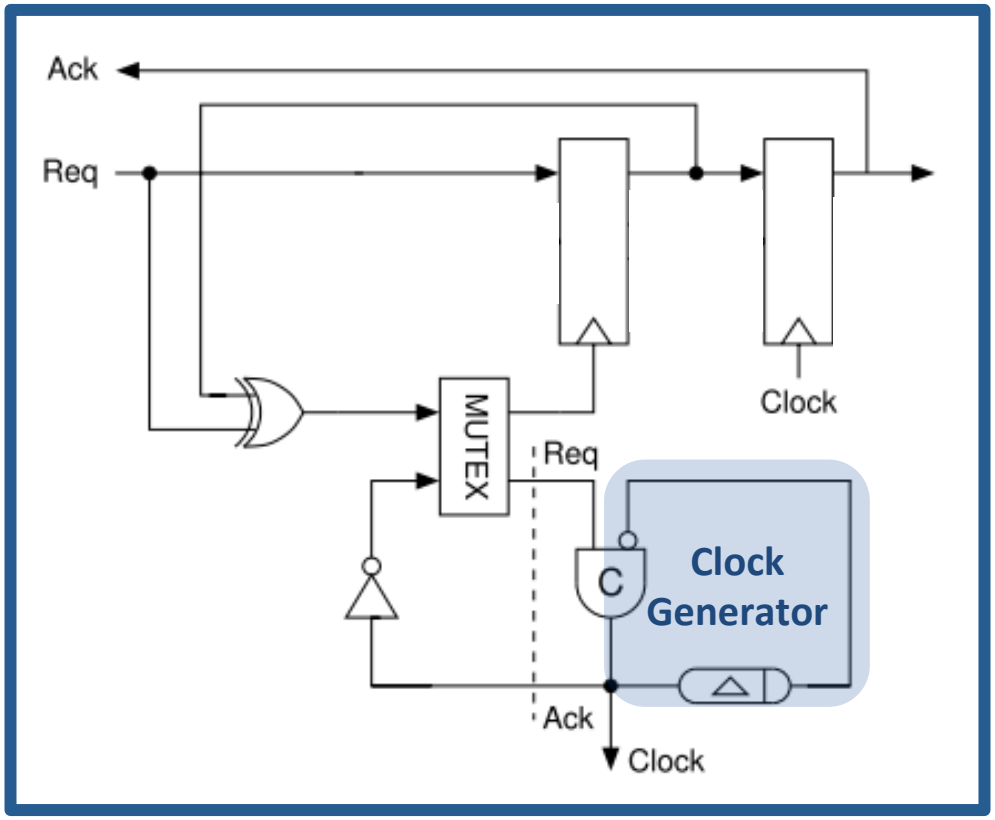


Clock Pauses Are Rare



Synchronization With Pausible Clock Generators

"Pausible Clock Generator"



Mutex and pausible clock safely synchronize the signal.

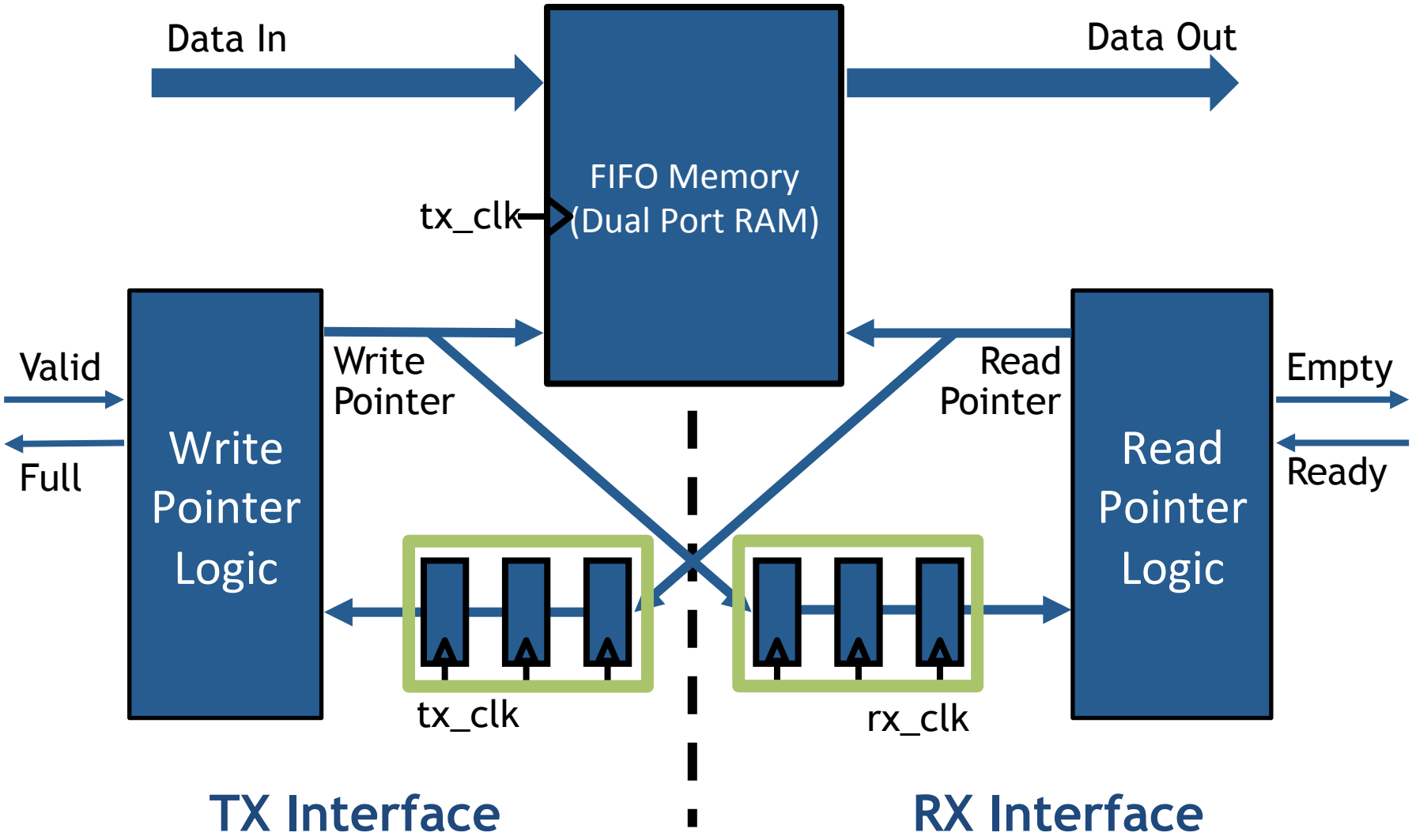
Synchronized signal arrives after (an average of) just 1 clock cycle!

"Demystifying Data Driven and Pausible Clocking Schemes," Mullins07

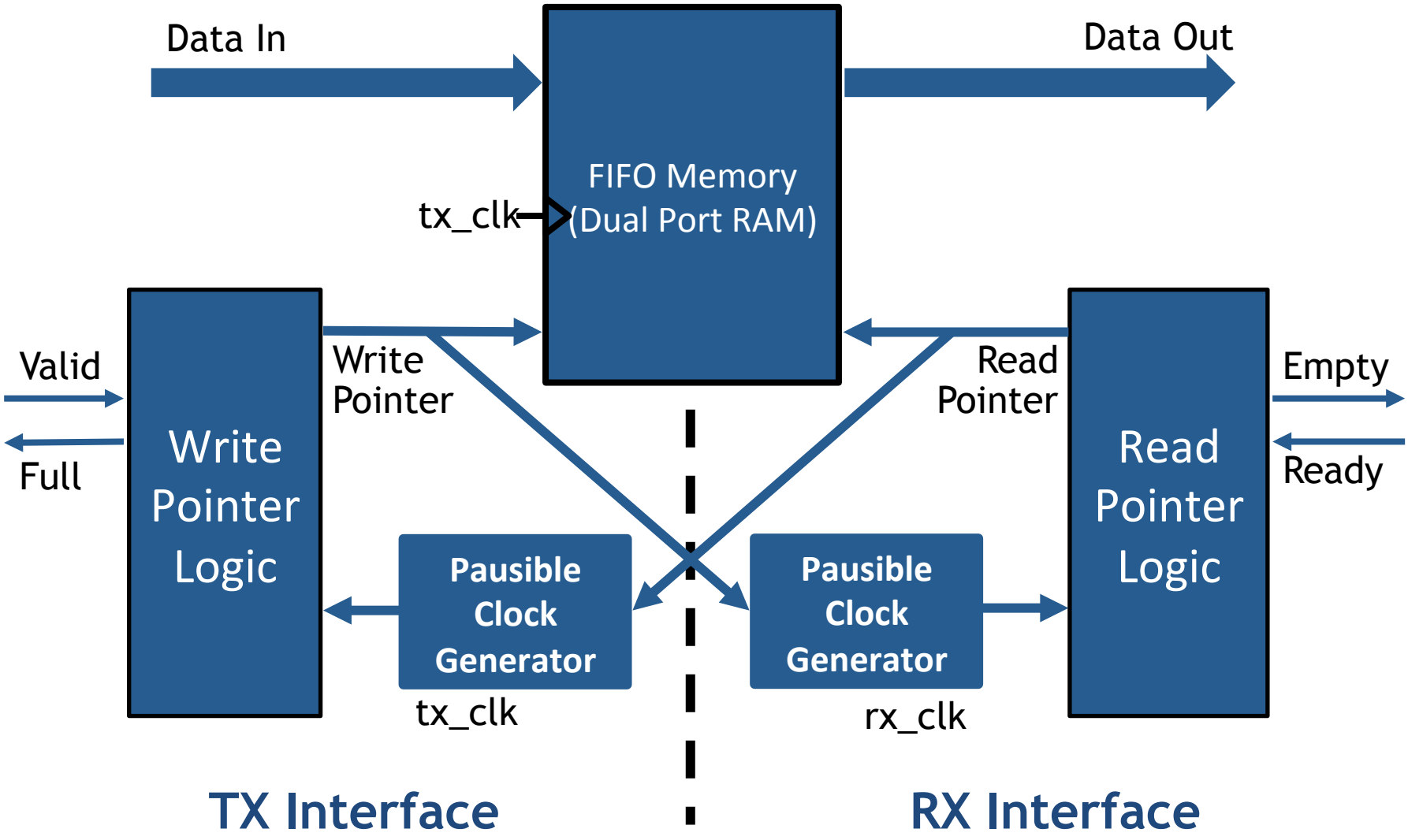
Pausible Clocks: Our Contribution

- Prior work in pausible clocks:
  - Proposed pausible clocking as a technique for low-latency asynchronous interface crossings (Yun96)
  - Integrated pausible clocks with asynchronous FIFOs to make GALS wrappers (Moore02, Fan09, others)
- We propose a circuit that:
  - Pairs pausible clocking techniques with standard two-ported synchronous FIFOs
  - Uses a novel flow-control scheme that enables low-latency asynchronous boundary crossings
  - Integrates easily with standard toolflows

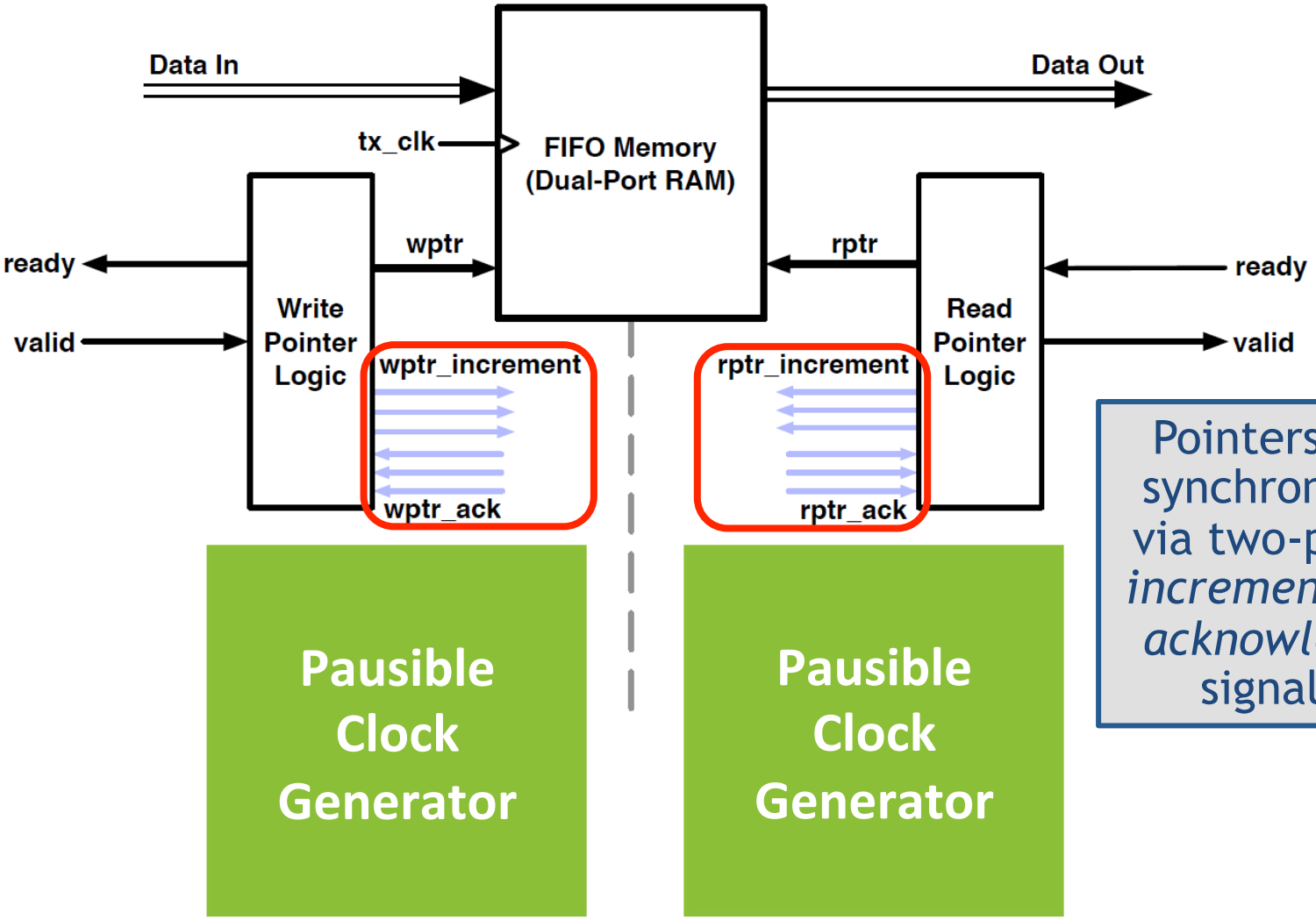
A Pausible Bisynchronous FIFO



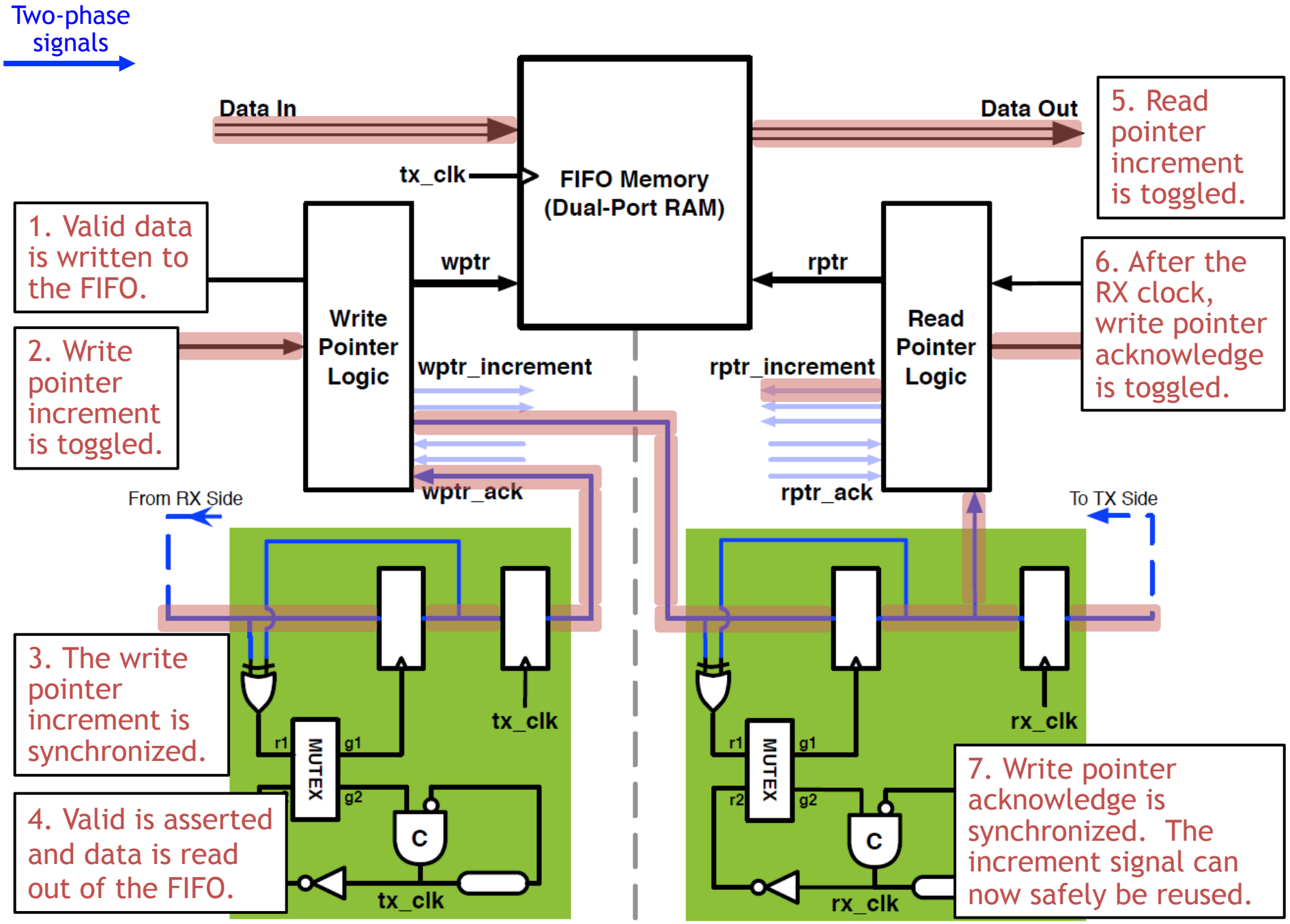
A Pausible Bisynchronous FIFO



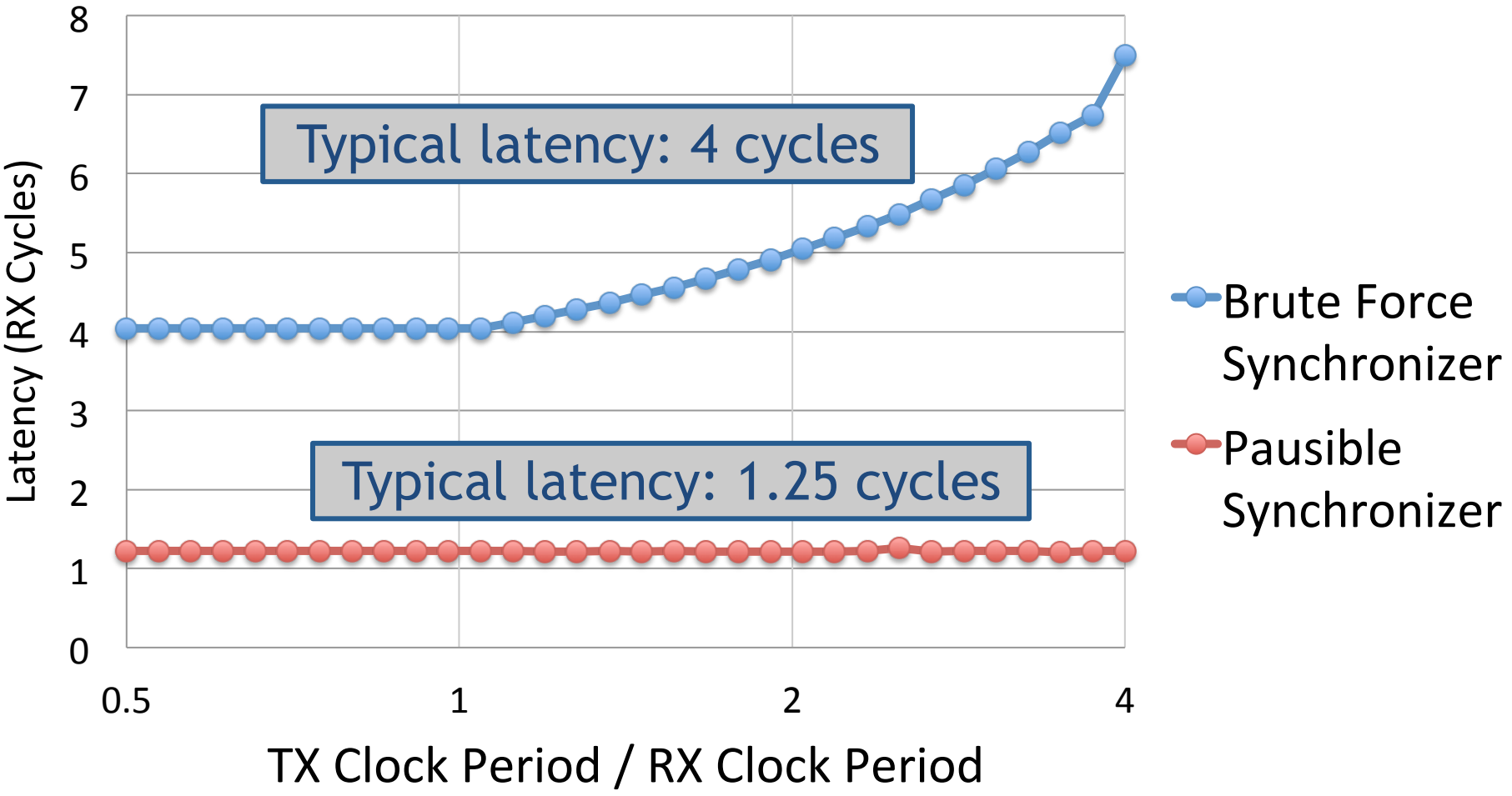
A Pausible Bisynchronous FIFO



Pointers are synchronized via two-phase *increment* and *acknowledge* signals.


Two-phase signals
Data In
Data Out
tx_clk
FIFO Memory (Dual-Port RAM)
5. Read pointer increment is toggled.
1. Valid data is written to the FIFO.
wptr
rptr
Write Pointer Logic
Read Pointer Logic
6. After the RX clock, write pointer acknowledge is toggled.
2. Write pointer increment is toggled.
wptr_increment
rptr_increment
wptr_ack
rptr_ack
From RX Side
To TX Side
3. The write pointer increment is synchronized.
r1
g1
MUTEX
g2
C
tx_clk
rx_clk
r2
4. Valid is asserted and data is read out of the FIFO.
7. Write pointer acknowledge is synchronized. The increment signal can now safely be reused.

Simulation Results: Brute Force Synchronizer



Pausible Clocking Timing Constraints

Minimum clock period:

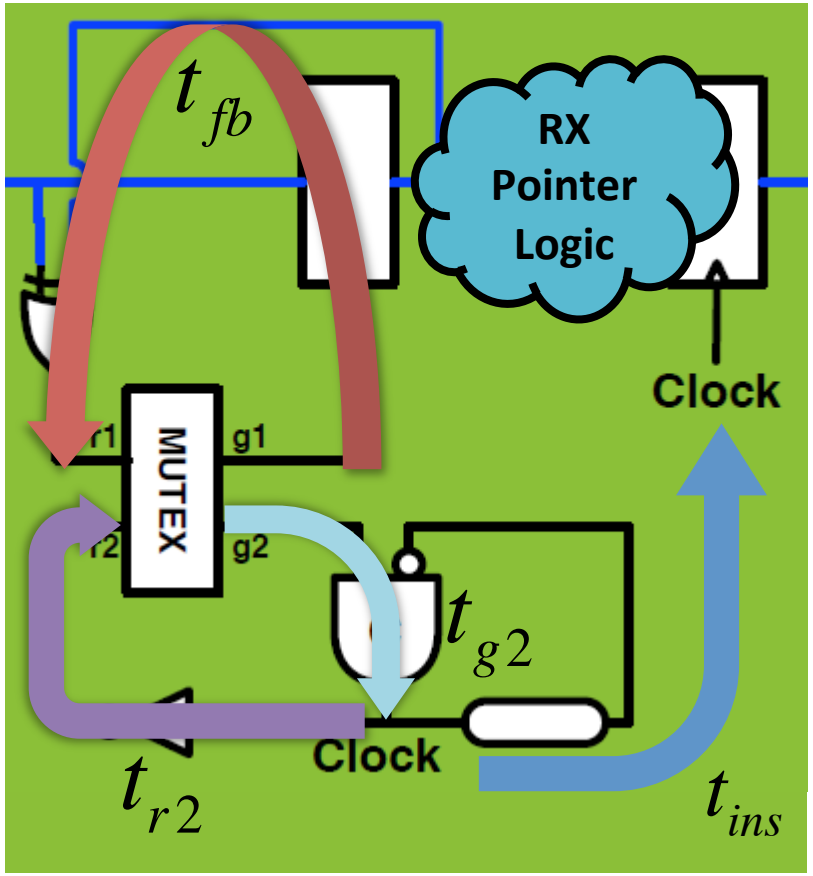
$$T/2 \geq t_{r2} + t_{g2} + t_{fb}$$

Maximum insertion delay:

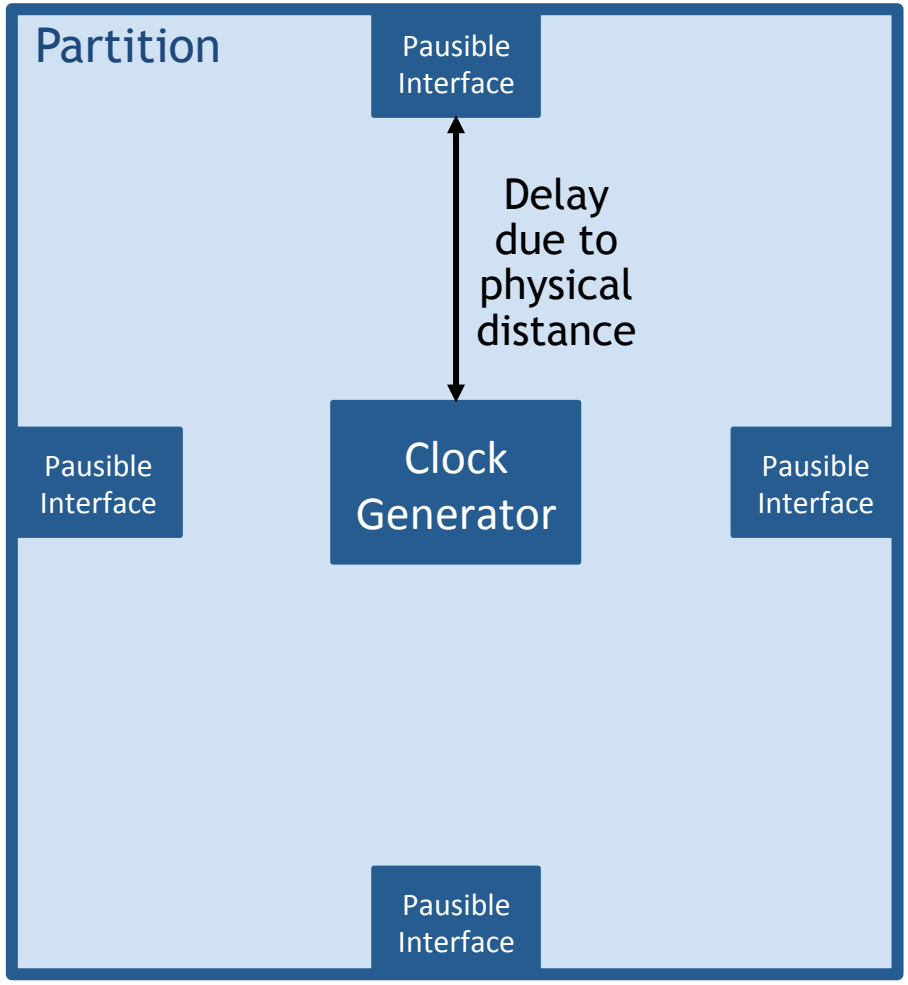
$$T_{ins} \leq T - t_{fb} - t_{g2}$$

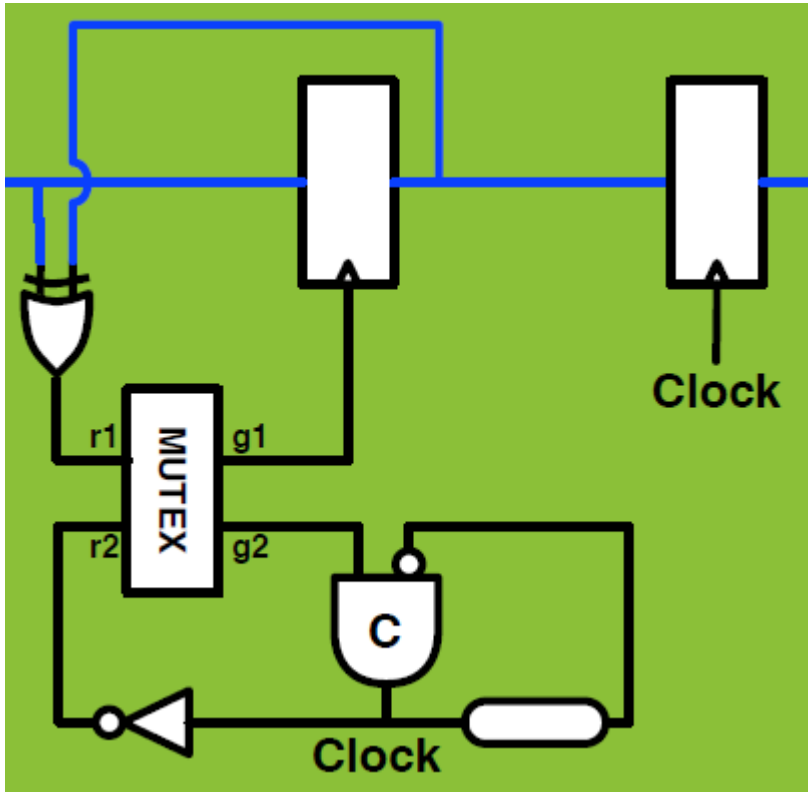
Maximum same-cycle work:

$$T_{CL} \leq t_{fb} + t_{g2} + T_{ins}$$



Pausible Clocking Limitations: Where to Put the Mutexes?





Pausible Clocking Limitations: Where to Put the Mutexes?

Option 1: Mutexes at the interface

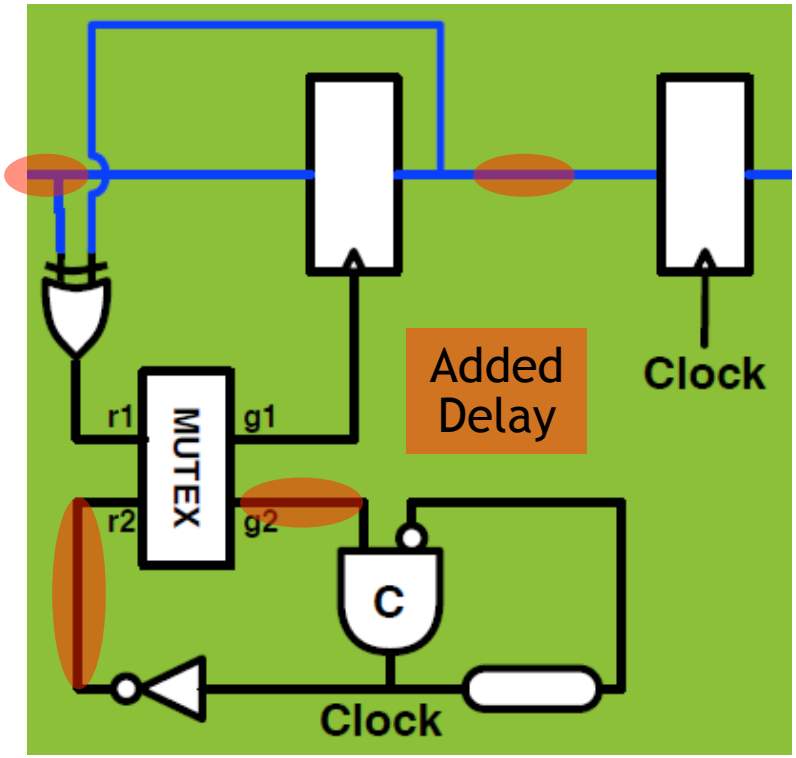
– Reduces maximum clock rate

Option 2: Mutexes at the center

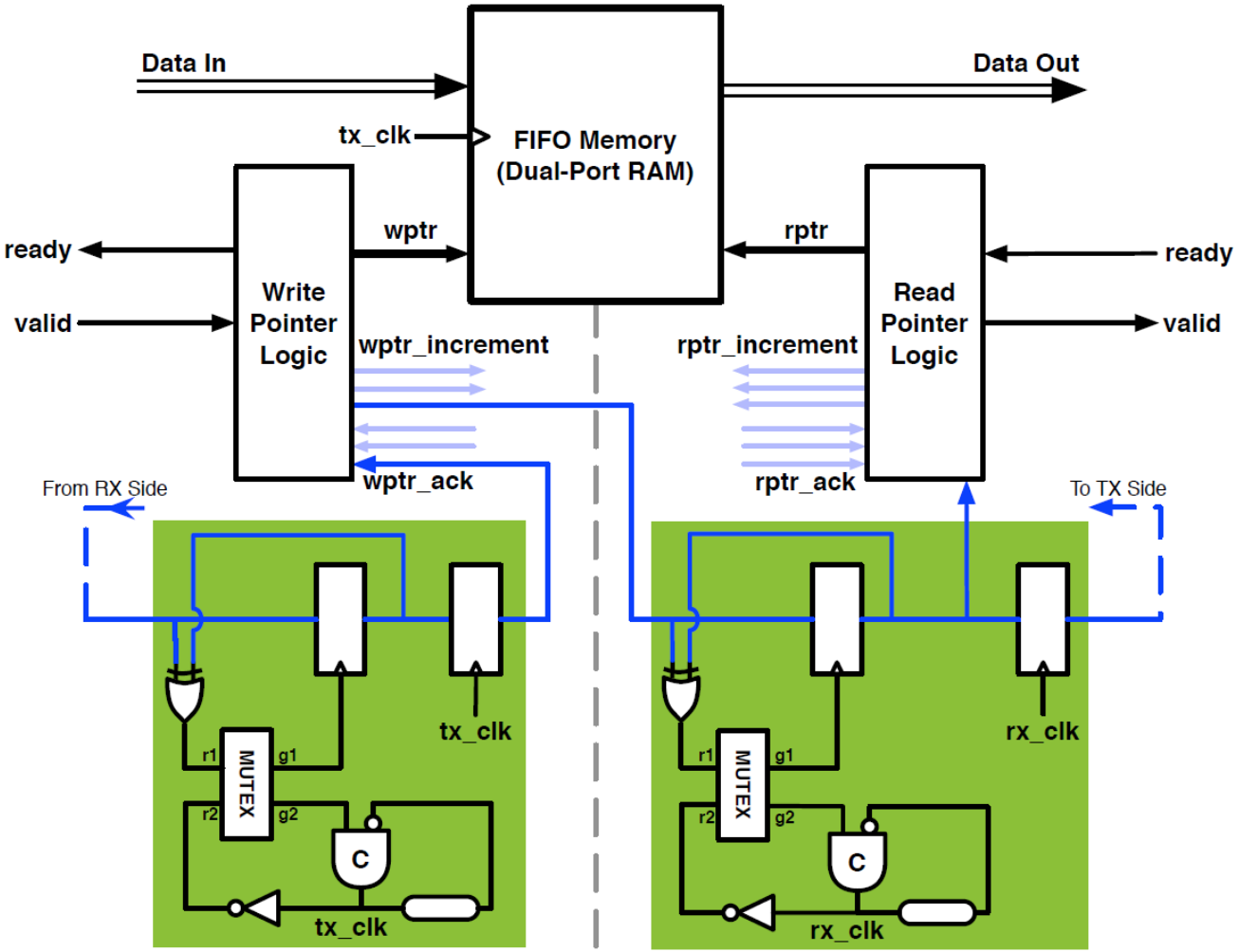
– Increases latency

+ Improves maximum clock rate

+ Can bake the mutex into a black box with the clock generator



Playing Nice with the Tools



- Most of the design is synchronous and works fine with synthesis tools
- FIFO is a standard dual-ported memory
- A single custom cell is needed
- Custom timing constraints at clock domain crossings

Synchronizer Comparison

Design	Average Latency (cycles)	Area (um^2)	Power (mW)	Energy (fJ/bit)
Synchronous Interface	1	4968	4.08	25.5
Brute Force Synchronizer	~4	5005	6.03	37.7
Pausible Synchronizer	**~1.5**	**4808**	**5.41**	**33.8**

Summary

- Pausible clocks have key advantages over standard synchronizers, including low latency and zero probability of failure
- We designed a pausible bisynchronous FIFO that pairs pausible clocking techniques with standard two-ported synchronous FIFOs
- Fast asynchronous interfaces that work well with standard toolflows are a key enabling technology of fine-grained GALS design

References

R. Mullins and S. Moore, “Demystifying Data-Driven and Pausible Clocking Schemes,” in *Proc. IEEE Symposium on Asynchronous Cir- cuits and Systems*, 2007, pp. 175–185.

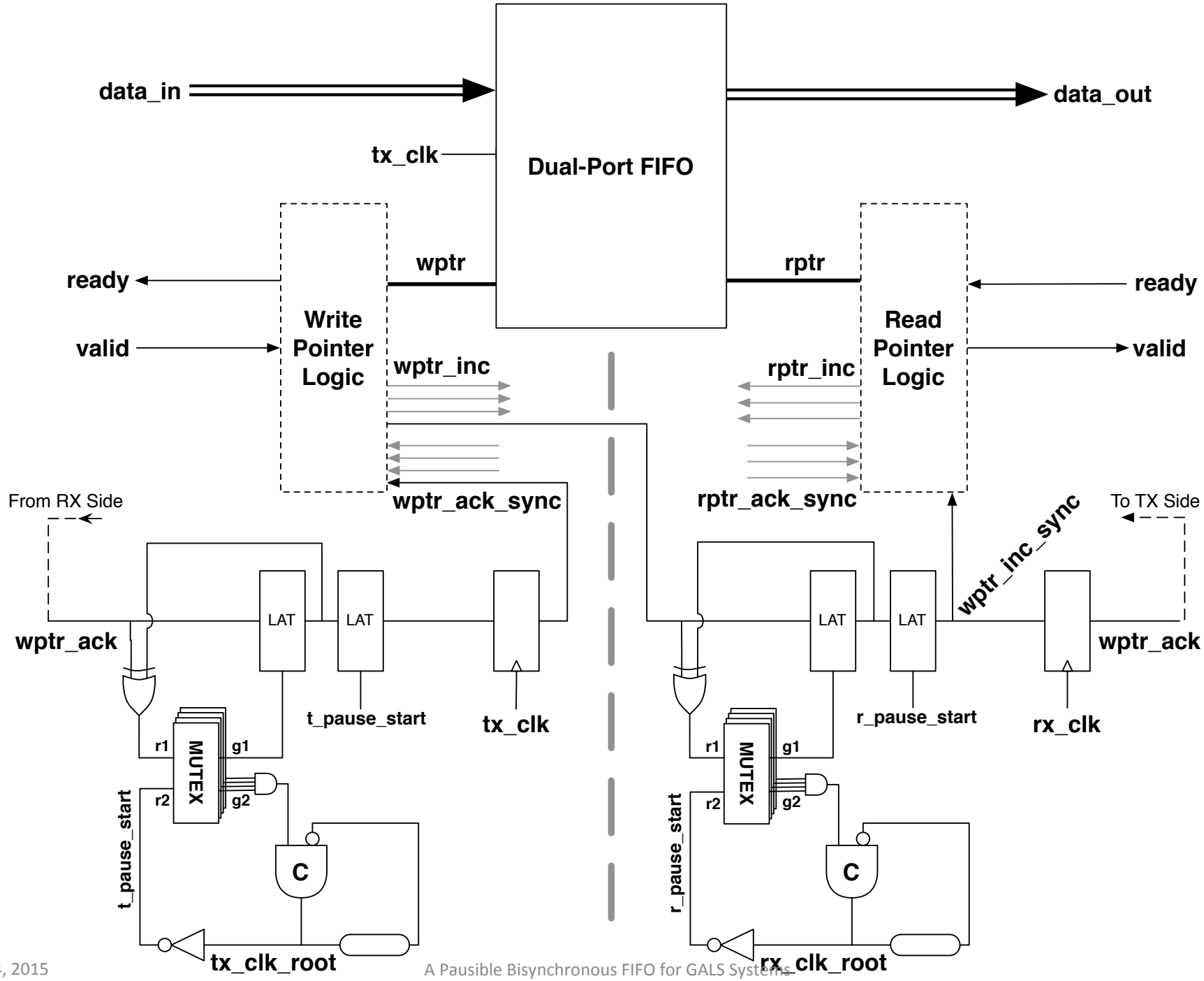
K. Yun and R. Donohue, “Pausible clocking: a first step toward heterogeneous systems,” in *Proc. IEEE International Conference on Computer Design*, 1996, pp. 118–123.

S. Moore *et al.*, “Point to point GALS interconnect,” in *Proc. IEEE Symposium on Asynchronous Circuits and Systems*, 2002, pp. 69–75.

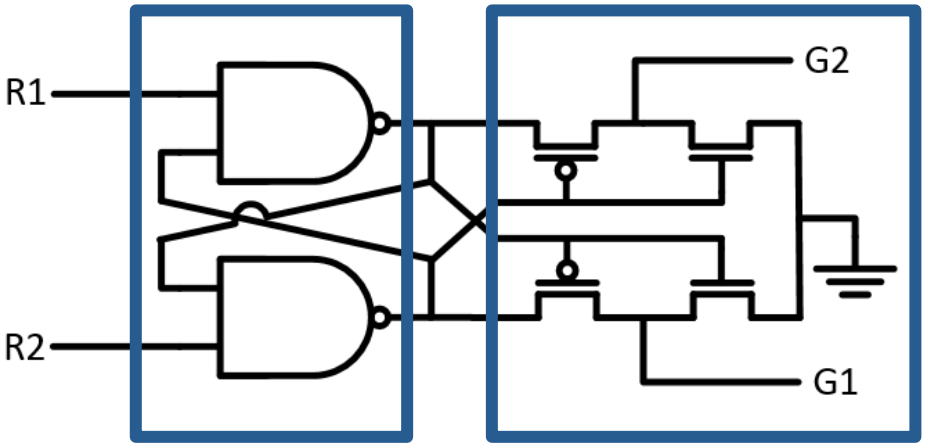
X. Fan *et al.*, “Analysis and optimization of pausible clocking based GALS design,” *IEEE International Conference Computer Design*, 2009.

Questions?

Backup Slides


data_in
data_out
tx_clk
Dual-Port FIFO
ready
wptr
rptr
ready
Write
Pointer
Logic
Read
Pointer
Logic
valid
valid
wptr_inc
rptr_inc
wptr_ack_sync
rptr_ack_sync
From RX Side
To TX Side
wptr_inc_sync
wptr_ack
LAT
LAT
LAT
LAT
wptr_ack
t_pause_start
tx_clk
r_pause_start
rx_clk
r1
g1
MUTEX
r2
g2
C
t_pause_start
tx_clk_root
r1
g1
MUTEX
r2
g2
C
r_pause_start
rx_clk_root

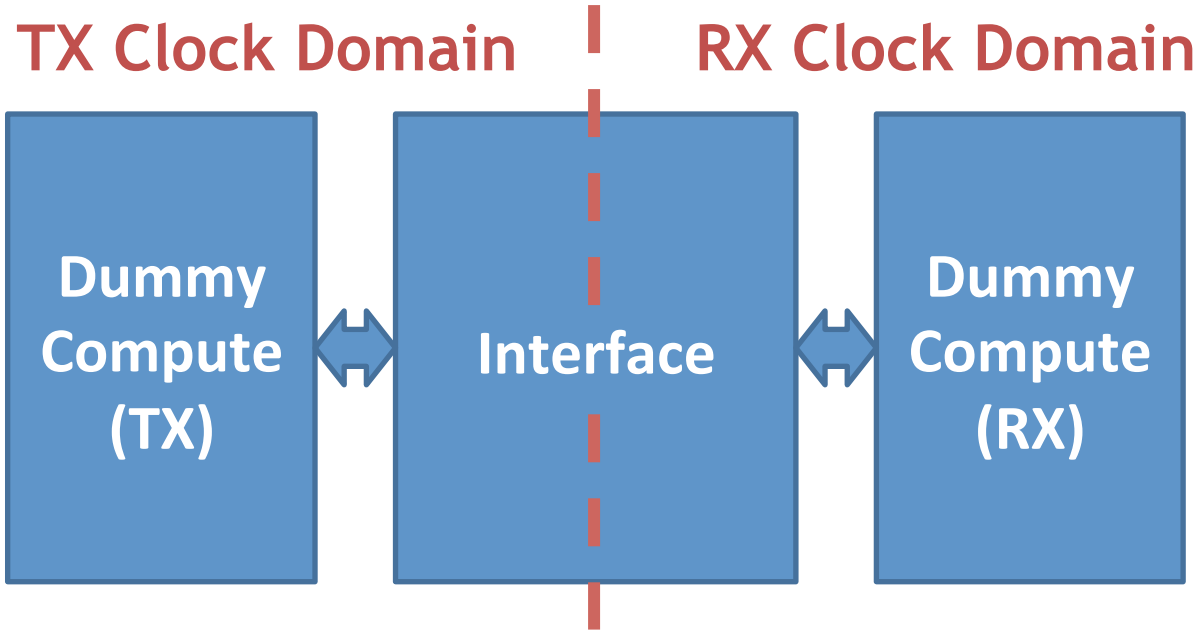
Mutex Circuit



R1	R2	G1	G2
0	0	0	0
1	0	1	0
0	1	0	1
1	1	R1 first -> G1 high R2 first -> G2 high	

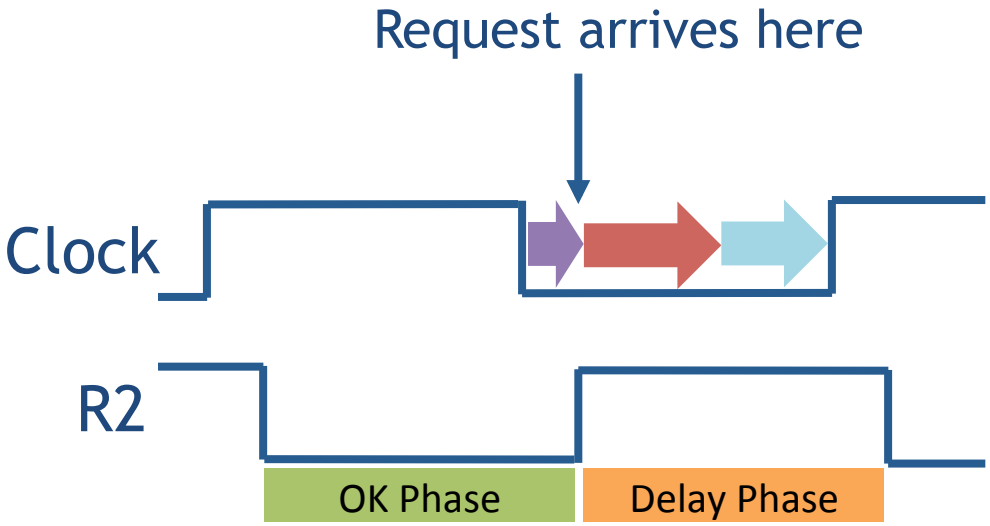
- Only G1 or G2 can be high at once (never both)
- If R1 and R2 transition high simultaneously, metastability can result
- The “metastability filter” keeps both outputs low until metastability resolves

Simulation Setup



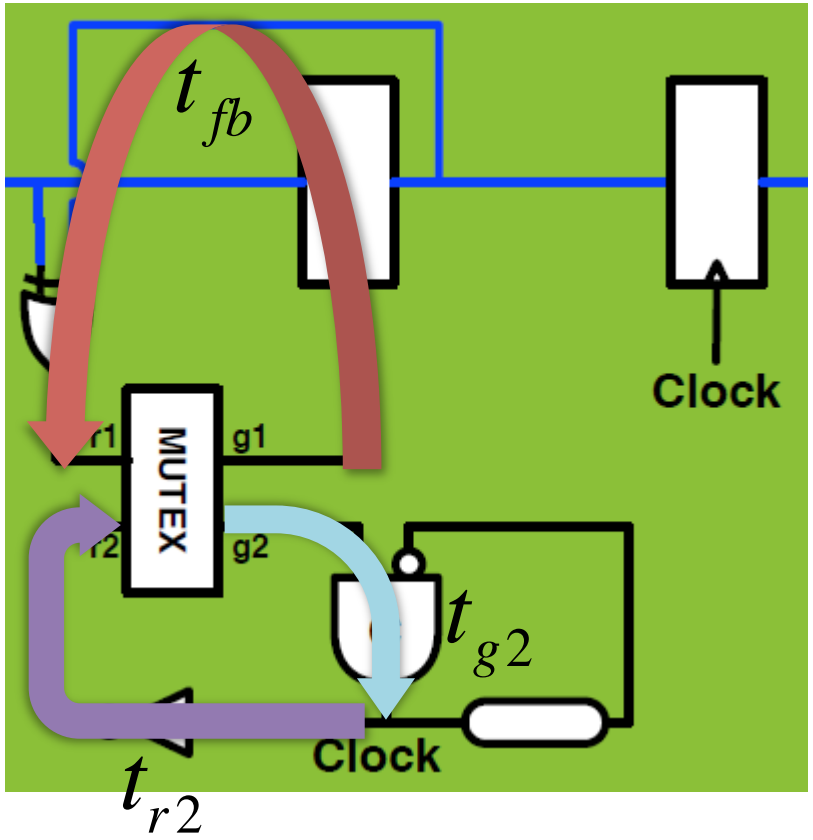
- Implemented interface in Verilog
- Allows direct comparison to brute-force synchronizers via drop-in replacement

Minimum Clock Period

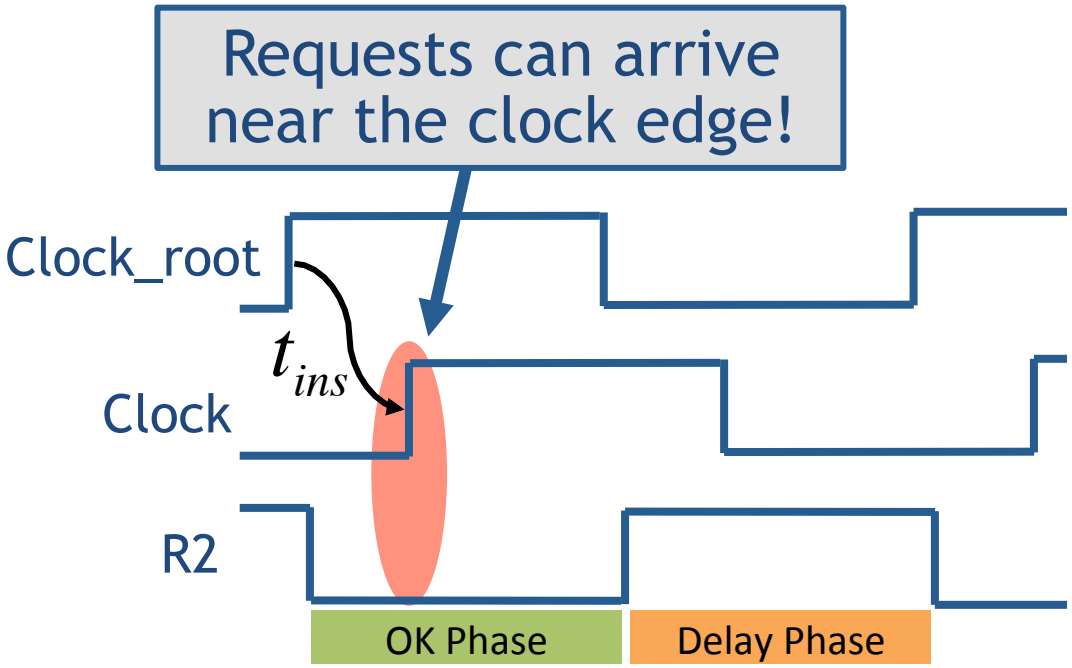


Minimum clock period:

$$T/2 \geq t_{r2} + t_{g2} + t_{fb}$$

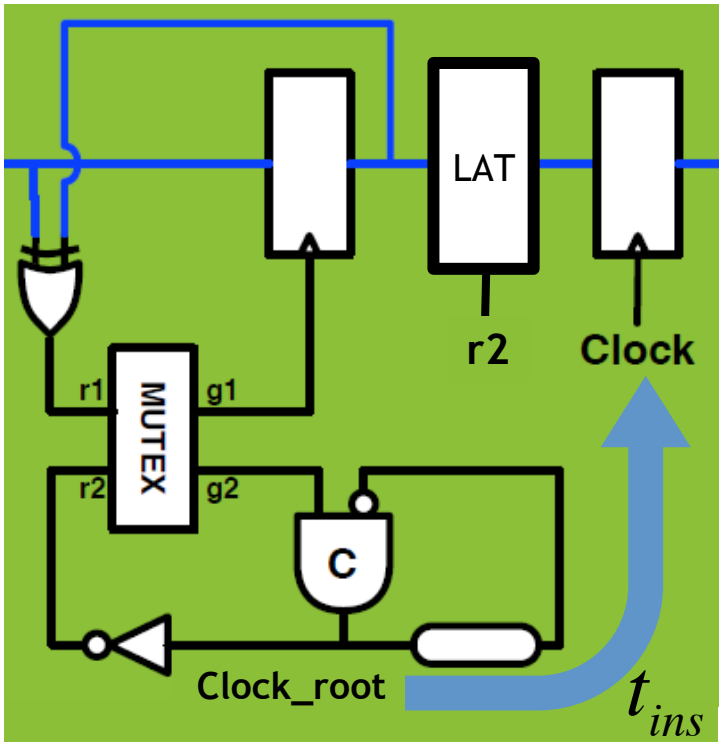


Insertion Delay

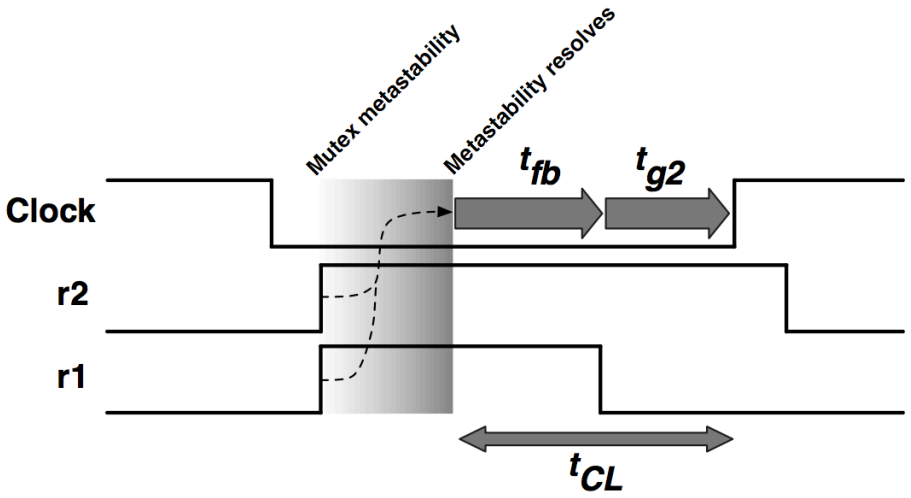


Maximum insertion delay:

$$T_{ins} \leq T - t_{fb} - t_{g2}$$

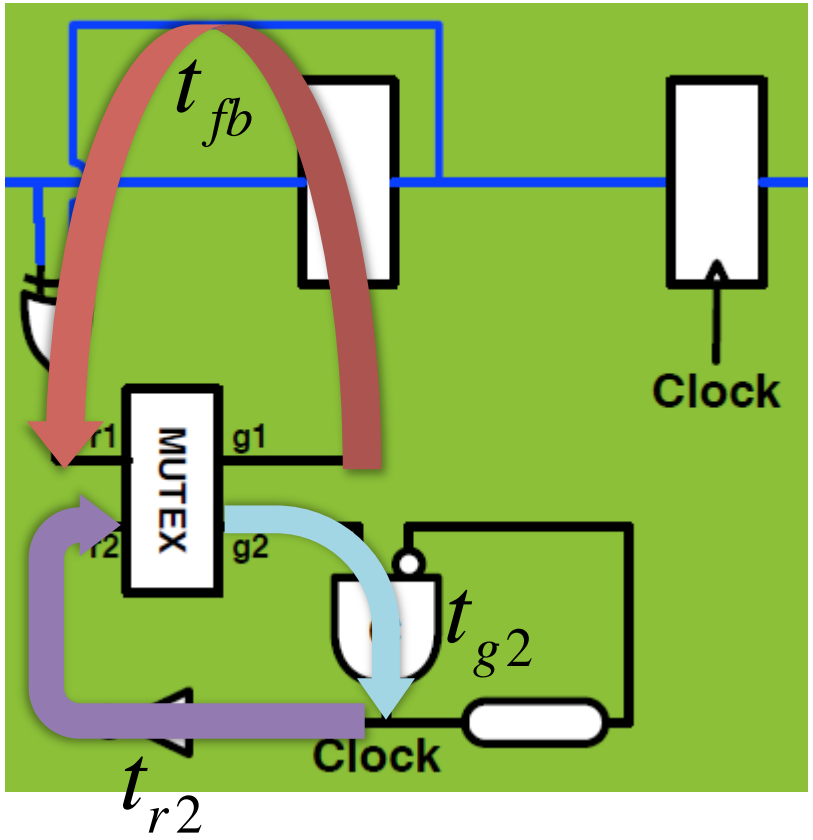


Combinational Logic Delay T_{CL}

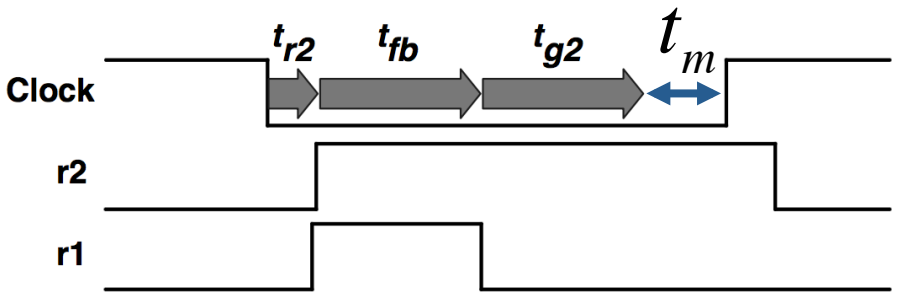


Maximum same-cycle work:

$$T_{CL} \leq t_{fb} + t_{g2} + T_{ins}$$



Metastability Margin t_m

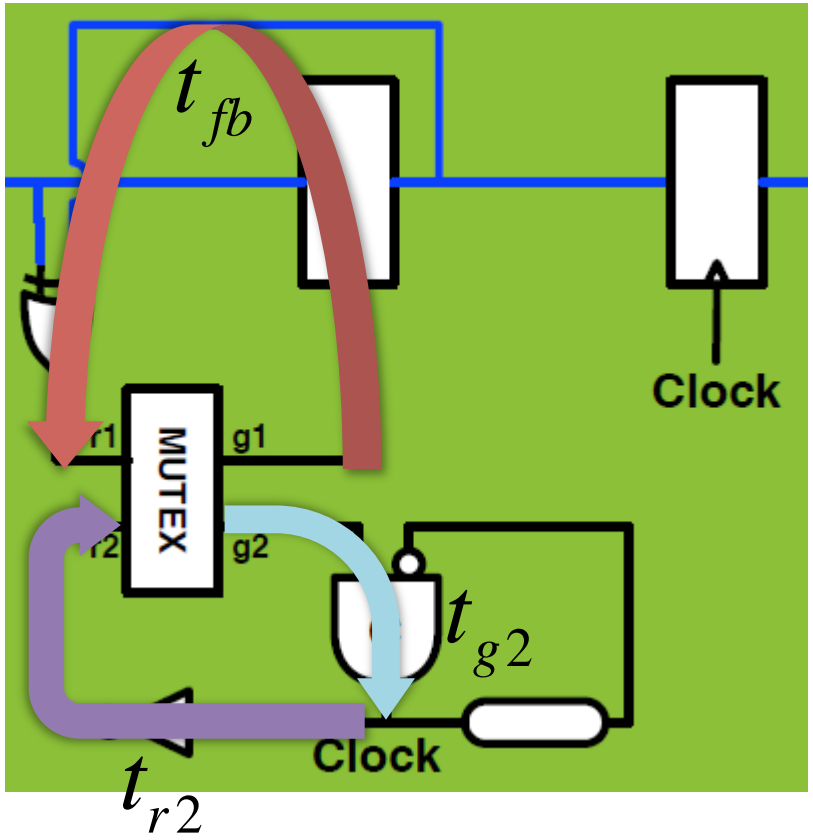


Minimum clock period:

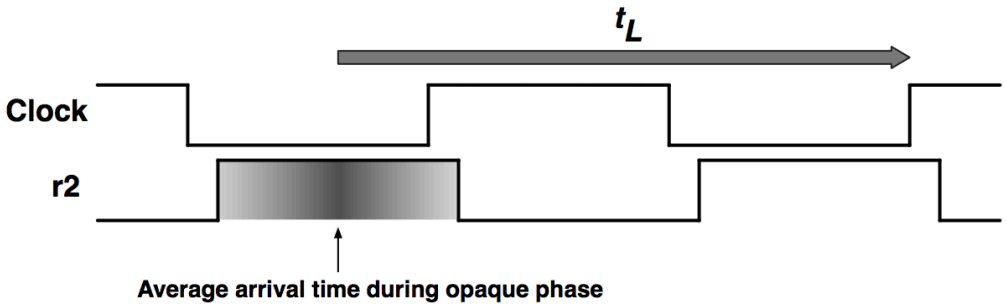
$$T/2 \geq t_{r2} + t_{g2} + t_{fb}$$

Metastability margin:

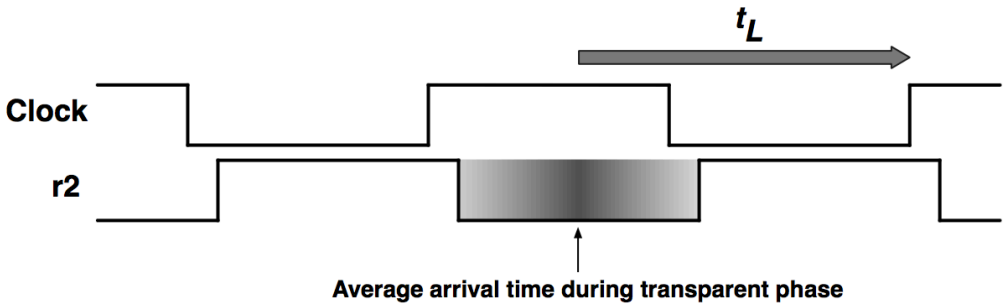
$$t_m = T/2 - (t_{r2} + t_{g2} + t_{fb})$$



Average Latency



(a)



(b)

Average latency:

$$T_L = T + T_{ins} - t_{r2}$$