Synchronization approaches

DAT093
larssv@chalmers.se

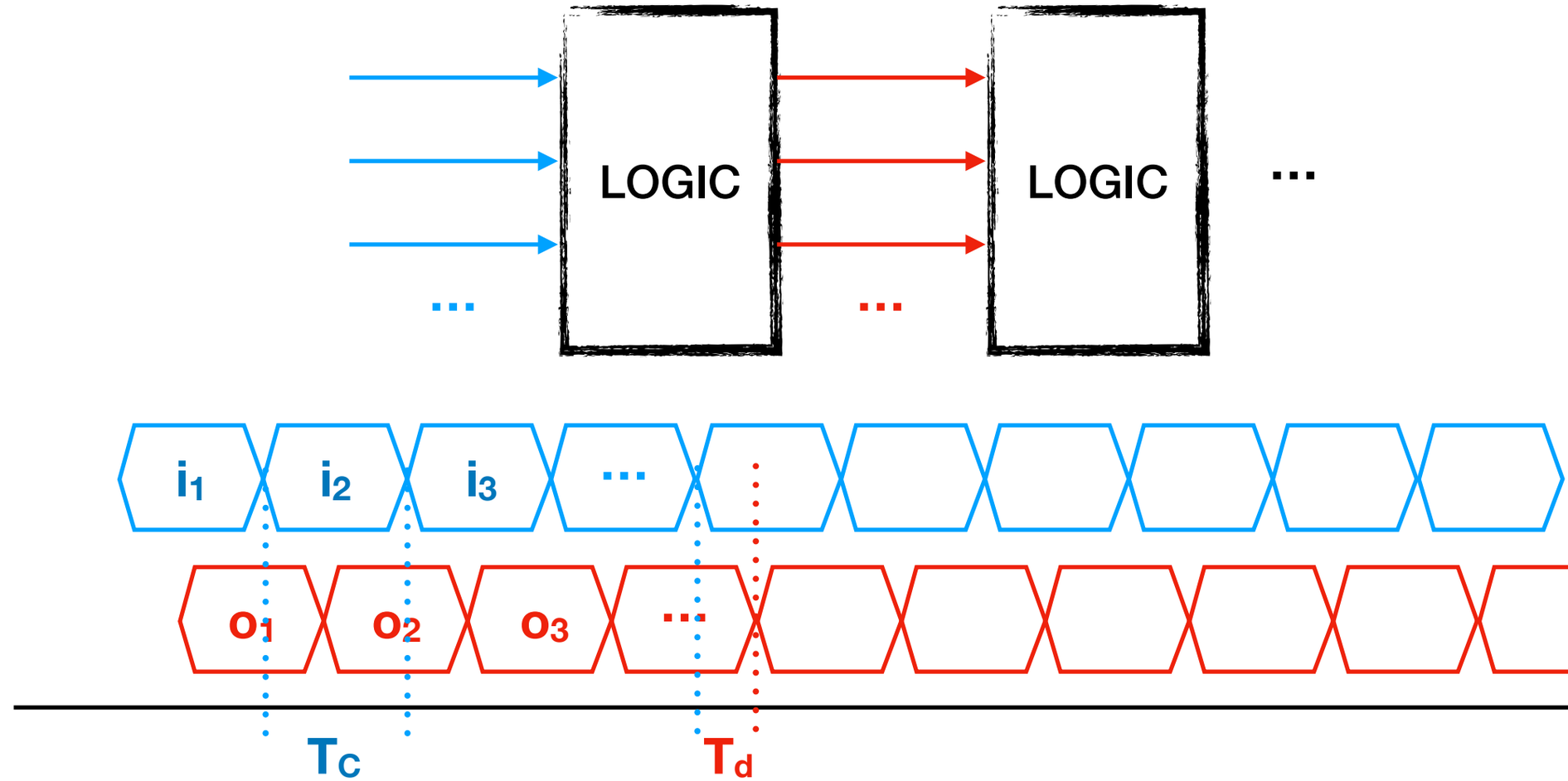
Overview

- Why use clocks at all?
- What limits clock speed?
- How handle several clock domains?
- Some practical points in the VHDL Style Sheet

Synchronous systems

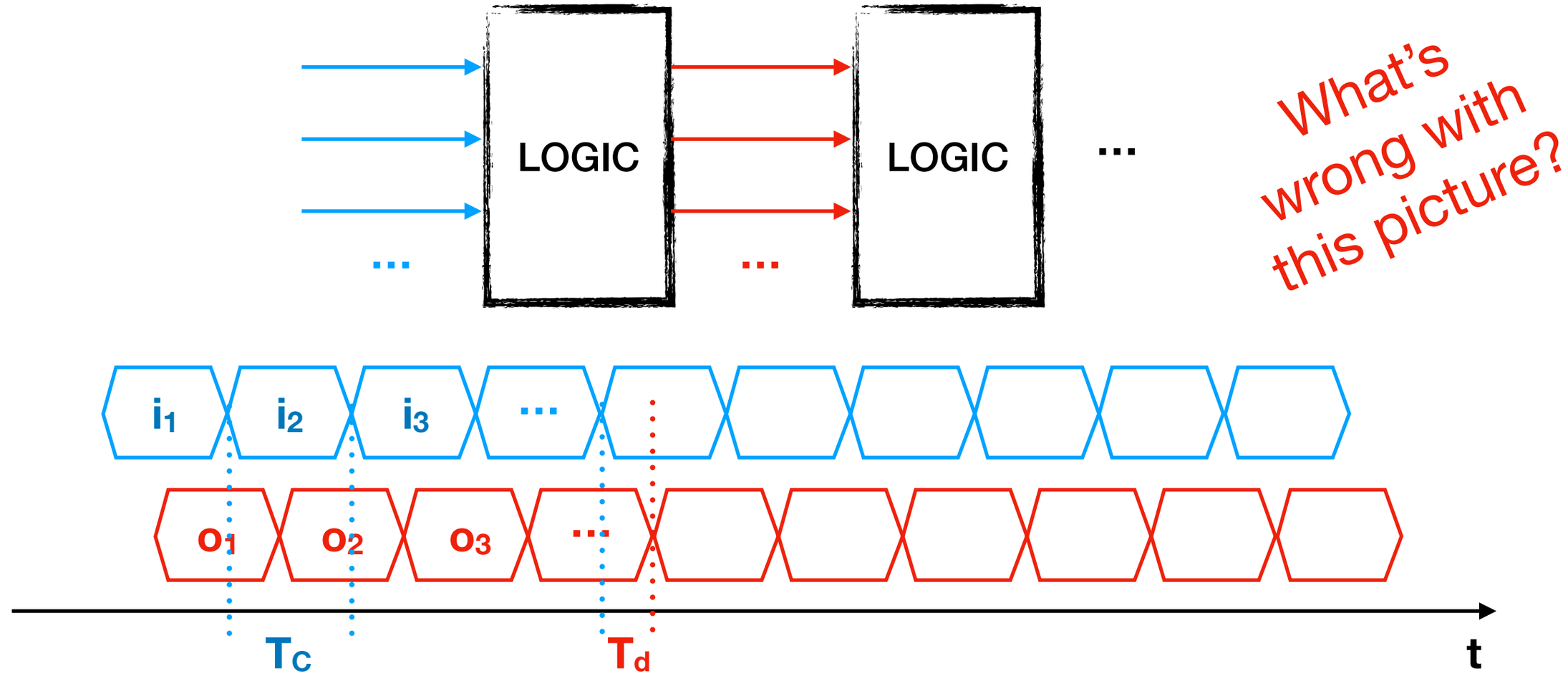
- Default paradigm for digital design
  - “Combinational” vs “sequential” circuits
  - Clocks carry timing information, other signals carry values
  - VHDL (etc.) idioms to support clocked systems
  - Tools depend on assumption and help with design
    - More in DAT110, SP2

Why use clocking at all?



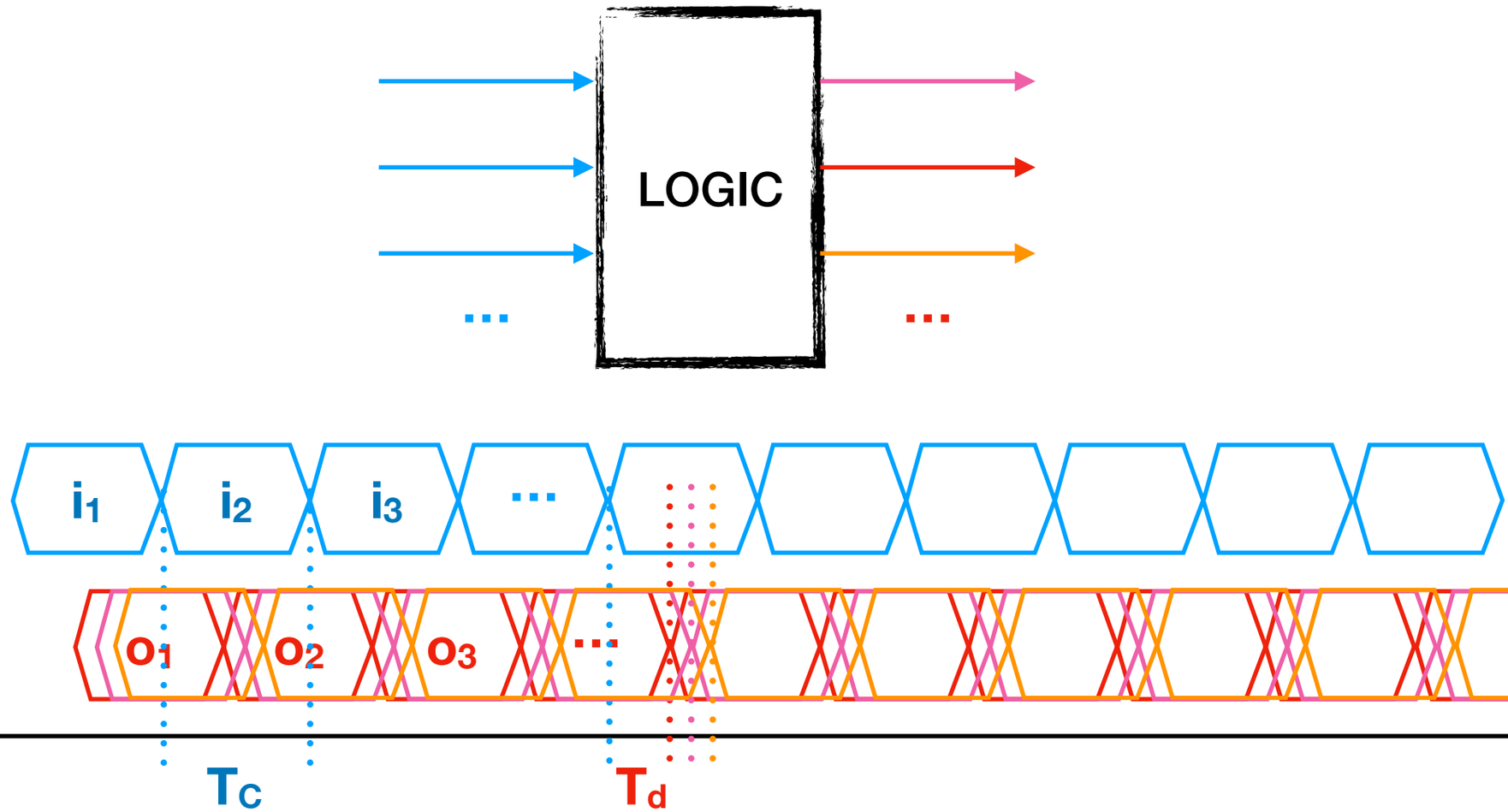
- Use logic circuits for computation on <u>sequence</u> of inputs
  - New set of input values applied after time T_C
- Outputs appear after logic delay T_d, separated by T_C

Why use clocking at all?



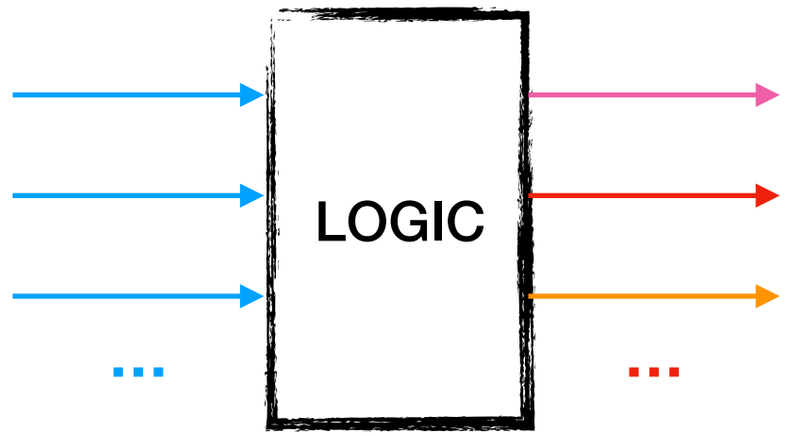
- Use logic circuits for computation on <u>sequence</u> of inputs
  - New set of input values applied after time T_C
- Outputs appear after logic delay T_d, separated by T_C

1. Unequal delays.

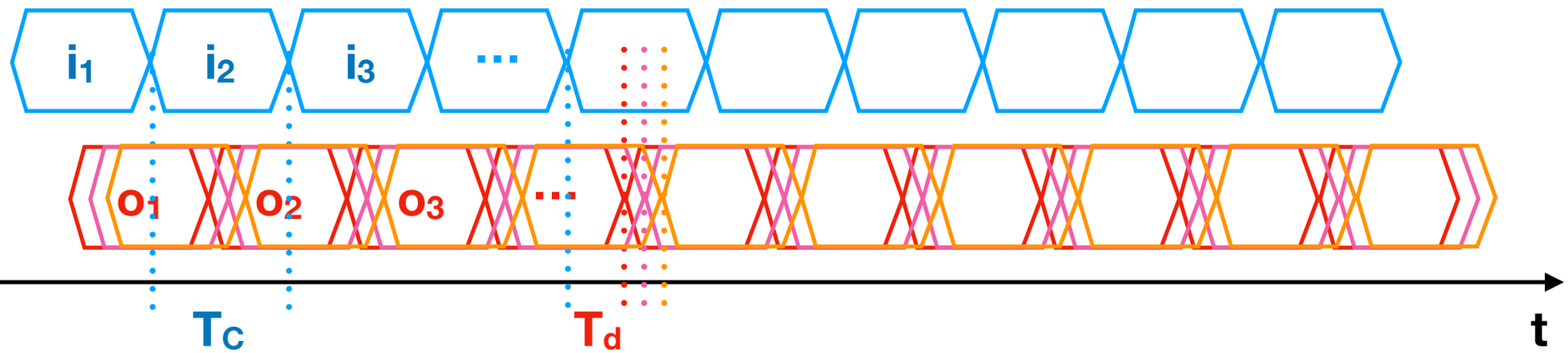


- Different delays for different outputs
  - Shorter time window when all outputs are valid
  - Gets worse with concatenated stages (until it breaks)

1. Unequal delays.

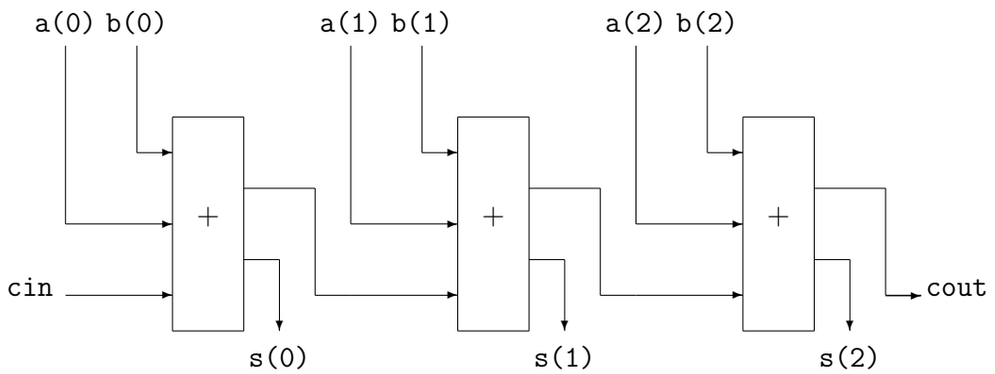


Could this be fixed?



- Different delays for different outputs
  - Shorter time window when all outputs are valid
  - Gets worse with concatenated stages (until it breaks)

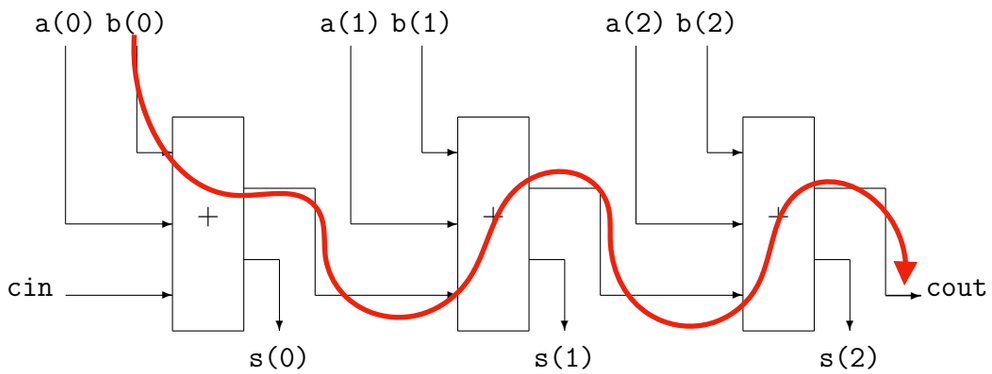
2. Data-dependent delays



Ripple-Carry Adder

- Often large difference between fastest and slowest operations across possible input data
  - Adder: exercise carry chain, or not
  - Multiplier: more complex

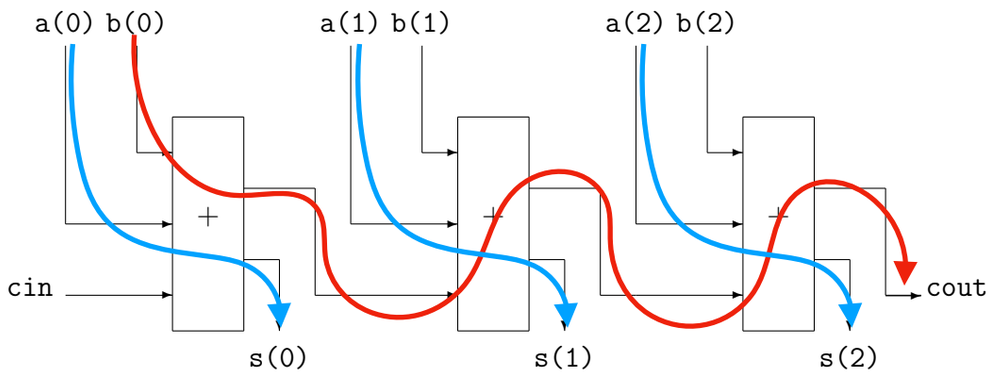
2. Data-dependent delays



Ripple-Carry Adder

- Often large difference between fastest and slowest operations across possible input data
  - Adder: exercise carry chain, or not
  - Multiplier: more complex

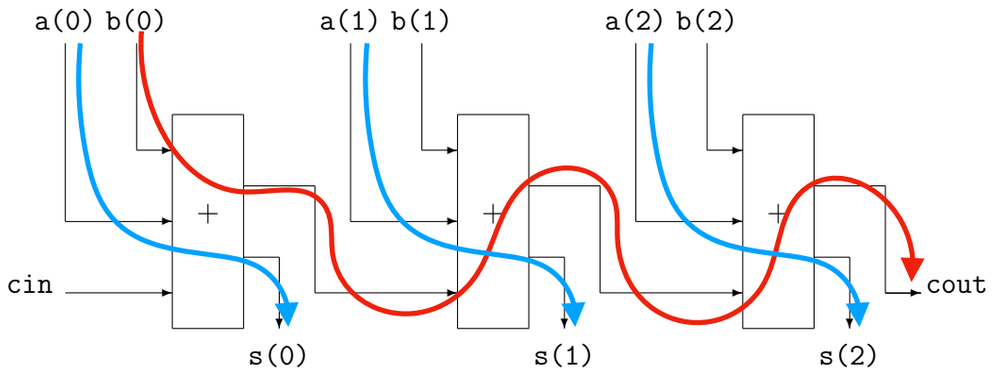
2. Data-dependent delays



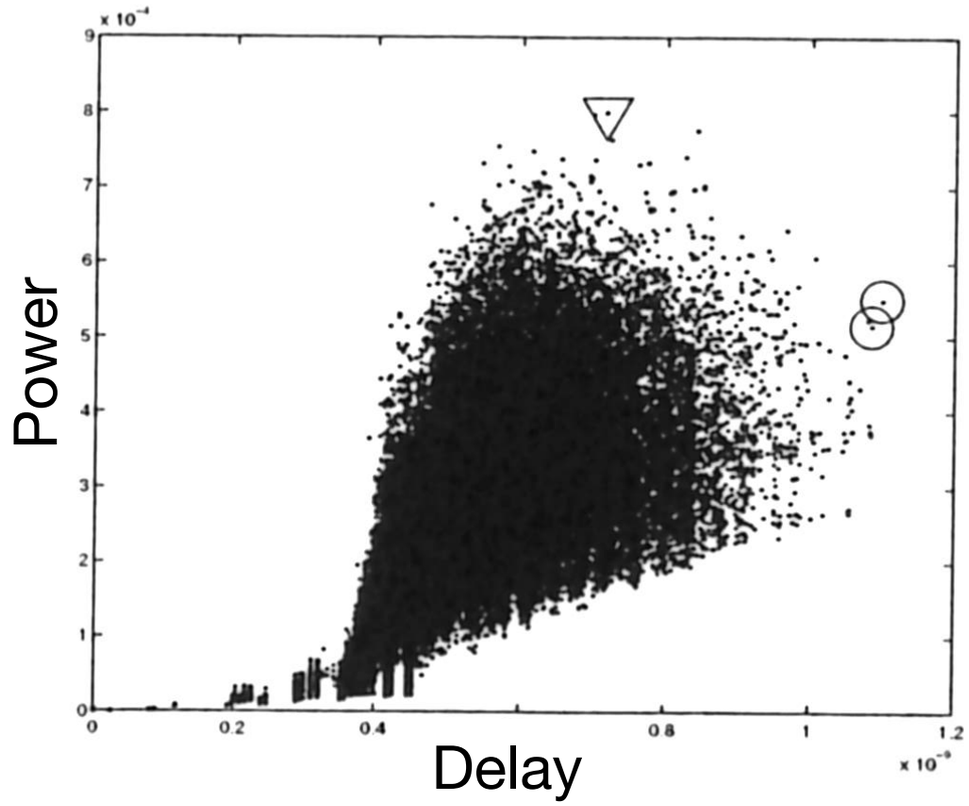
Ripple-Carry Adder

- Often large difference between fastest and slowest operations across possible input data
  - Adder: exercise carry chain, or not
  - Multiplier: more complex

2. Data-dependent delays



Ripple-Carry Adder

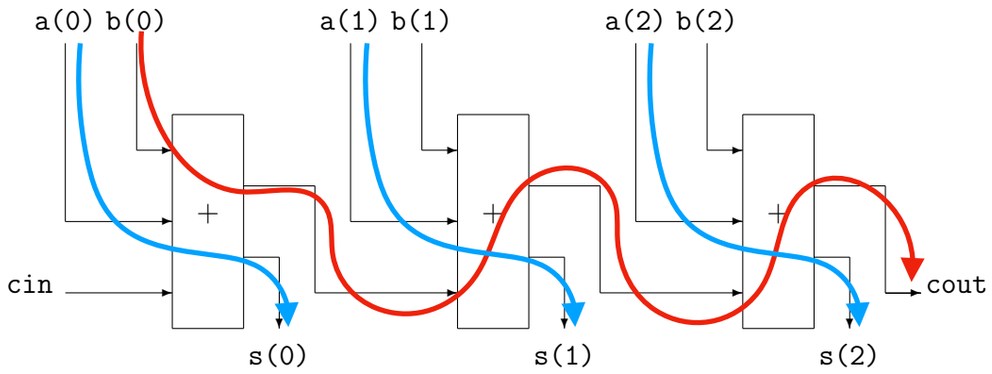


8-bit Multiplier

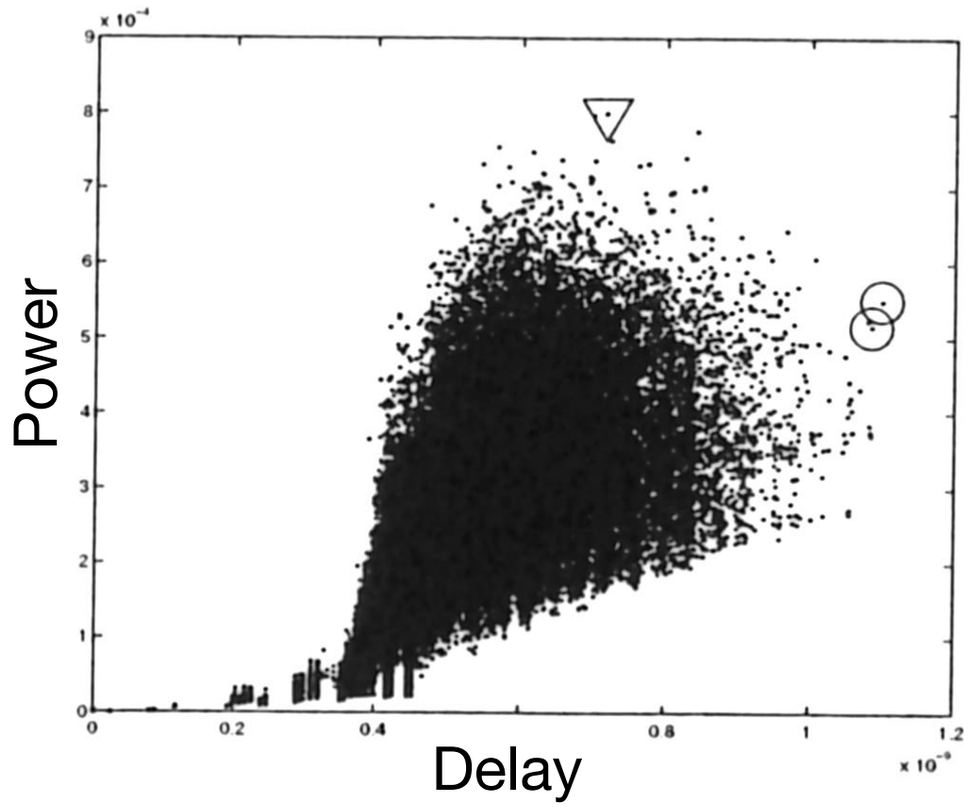
- Often large difference between fastest and slowest operations across possible input data
  - Adder: exercise carry chain, or not
  - Multiplier: more complex

[H. Eriksson, 2004 TBC]

2. Data-dependent delays



Ripple-Carry Adder



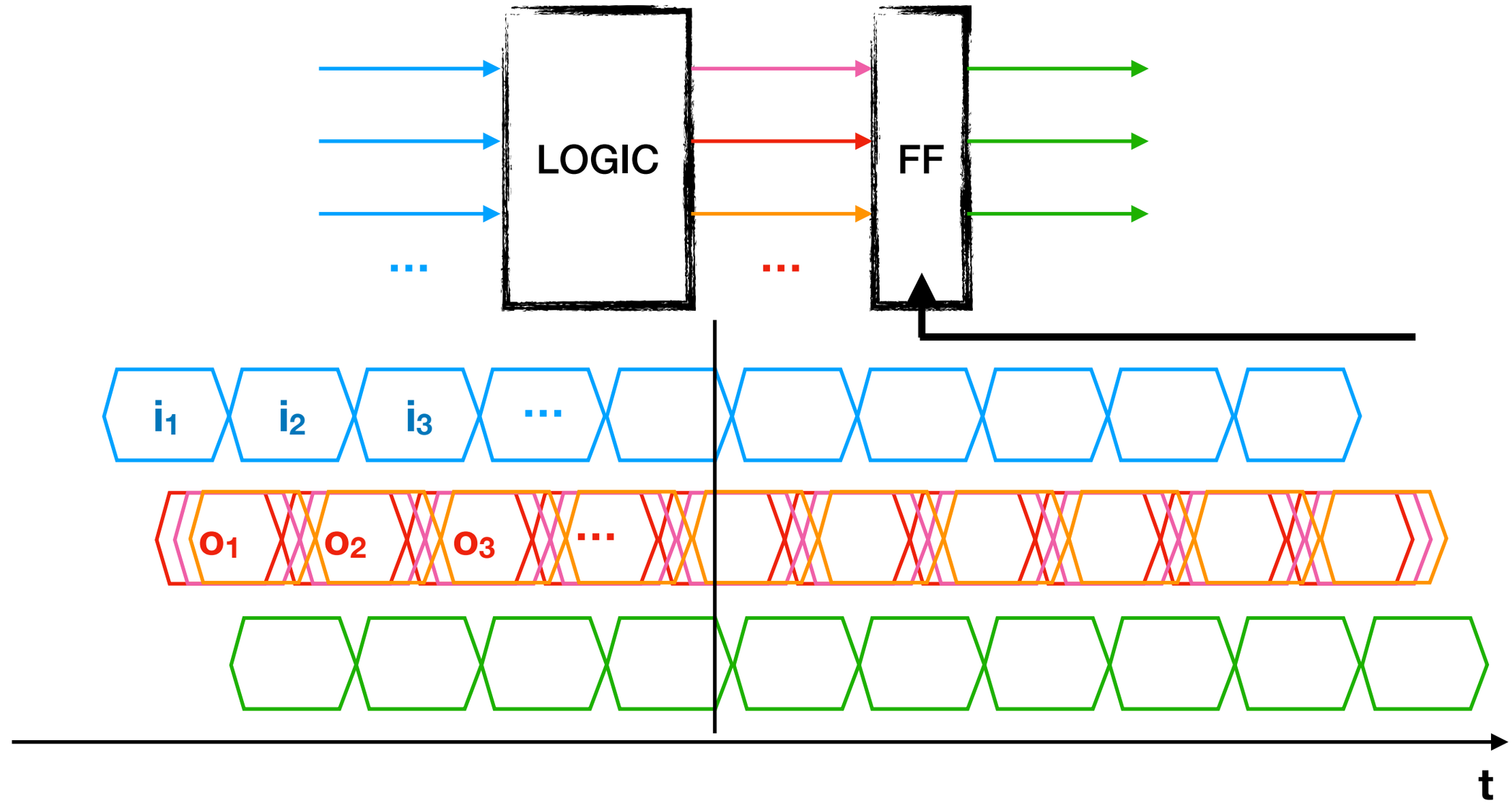
8-bit Multiplier

- Often large difference between fastest and slowest operations across possible input data
  - Adder: exercise carry chain, or not
  - Multiplier: more complex

Could this be fixed?

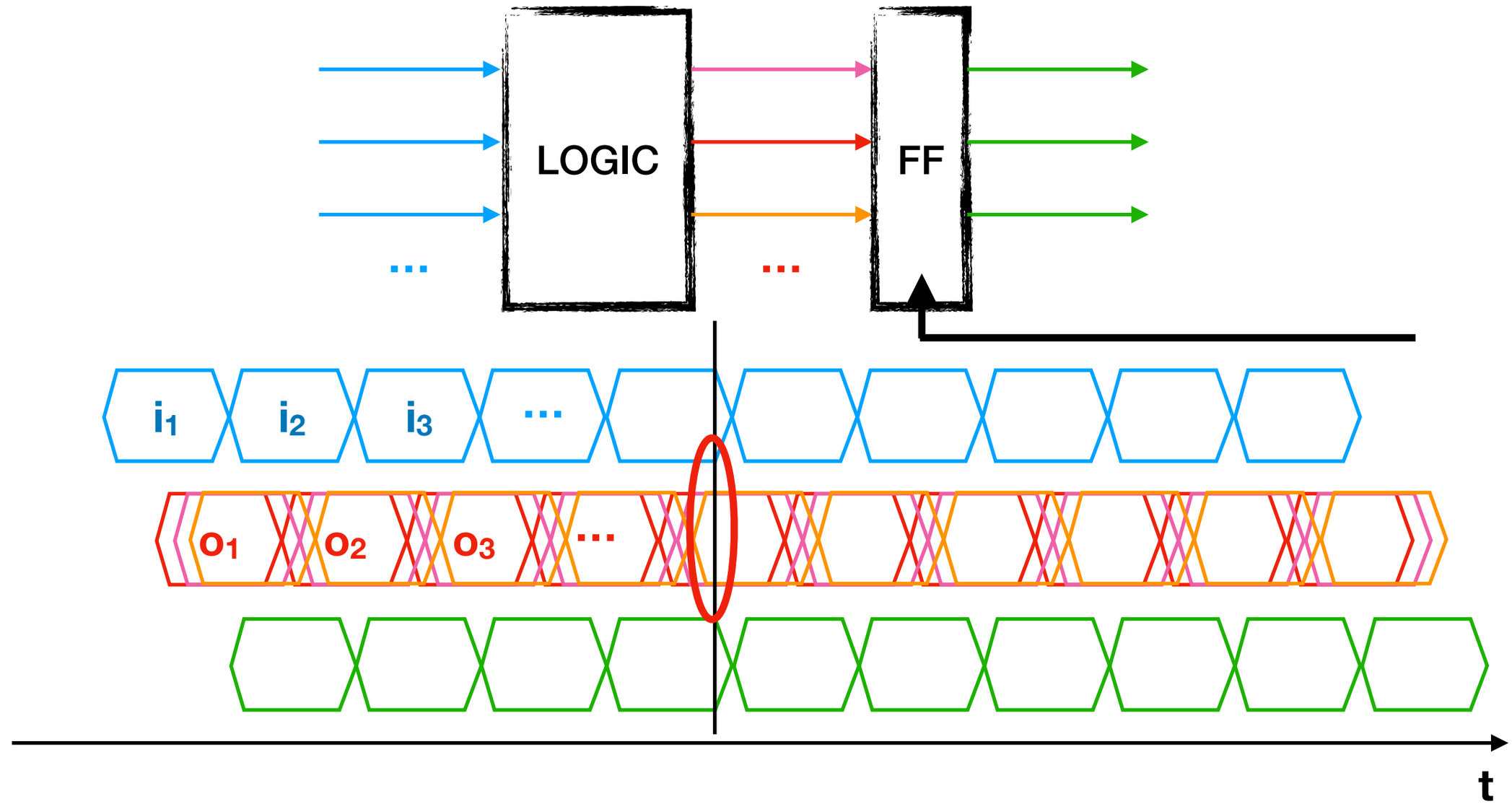
[H. Eriksson, 2004 TBC]

Delay early signal values



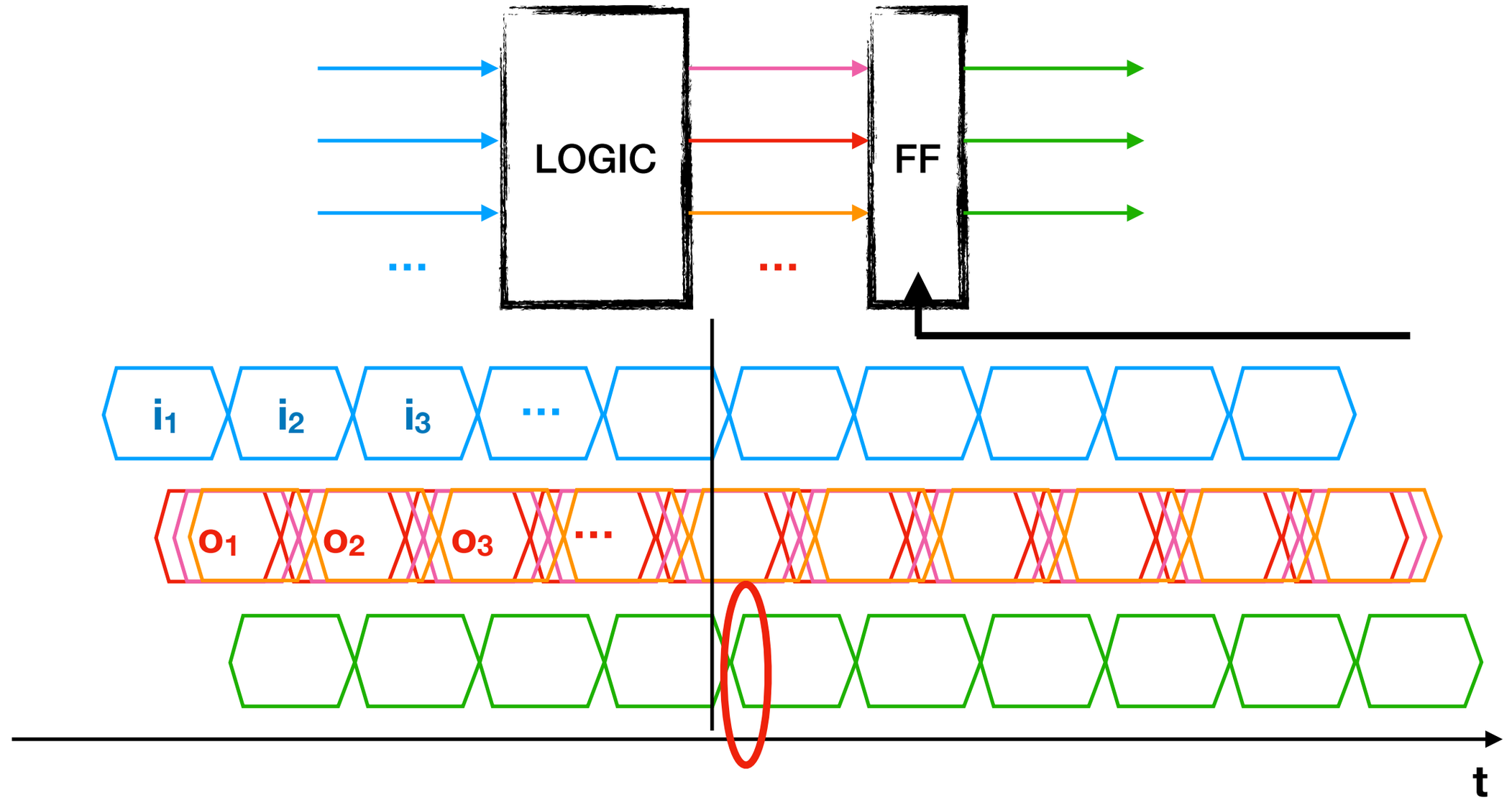
- Capture logic values using clock signal
  - Realign logic values in time; send on to next logic block

Delay early signal values



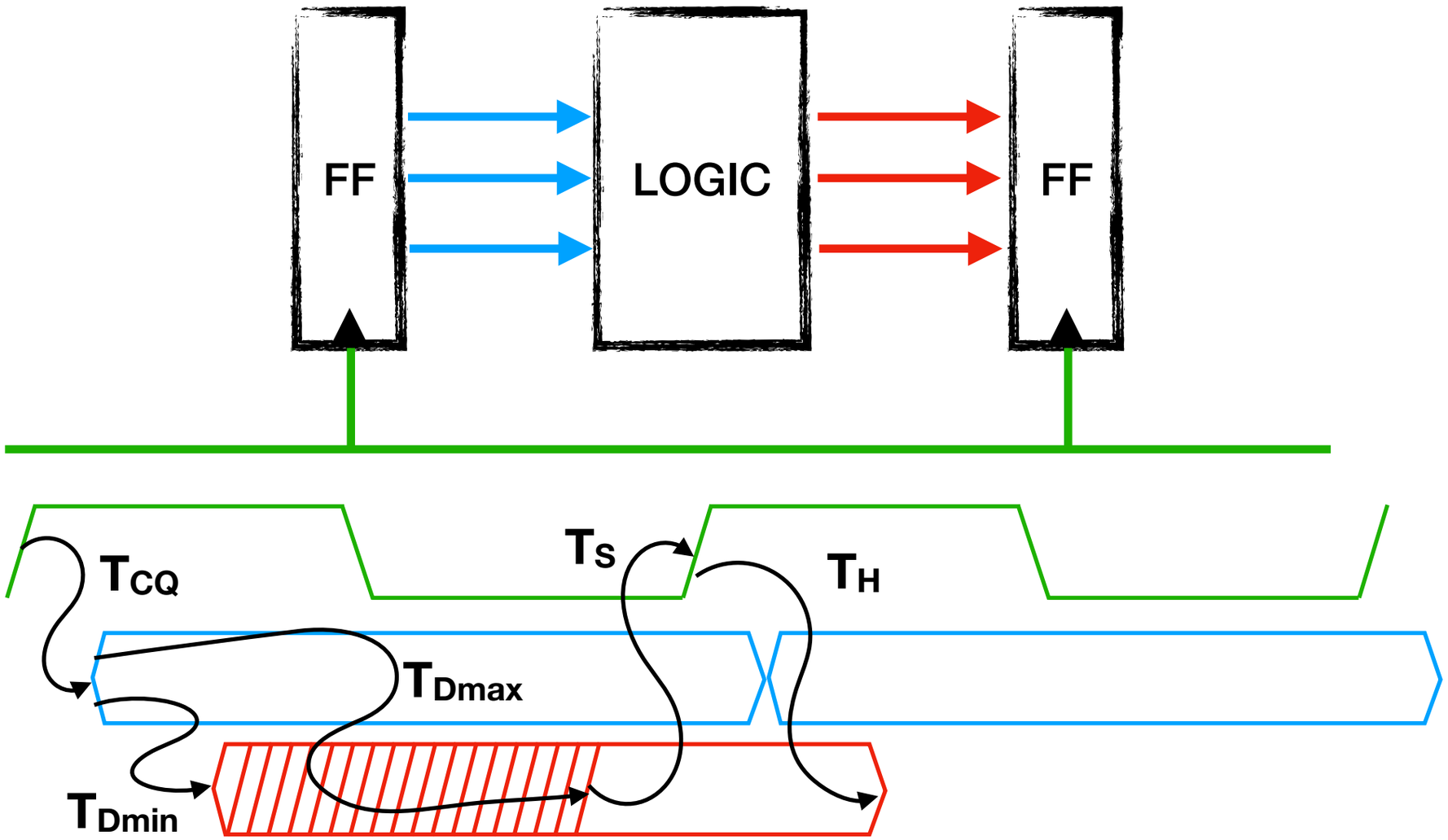
- Capture logic values using clock signal
  - Realign logic values in time; send on to next logic block

Delay early signal values



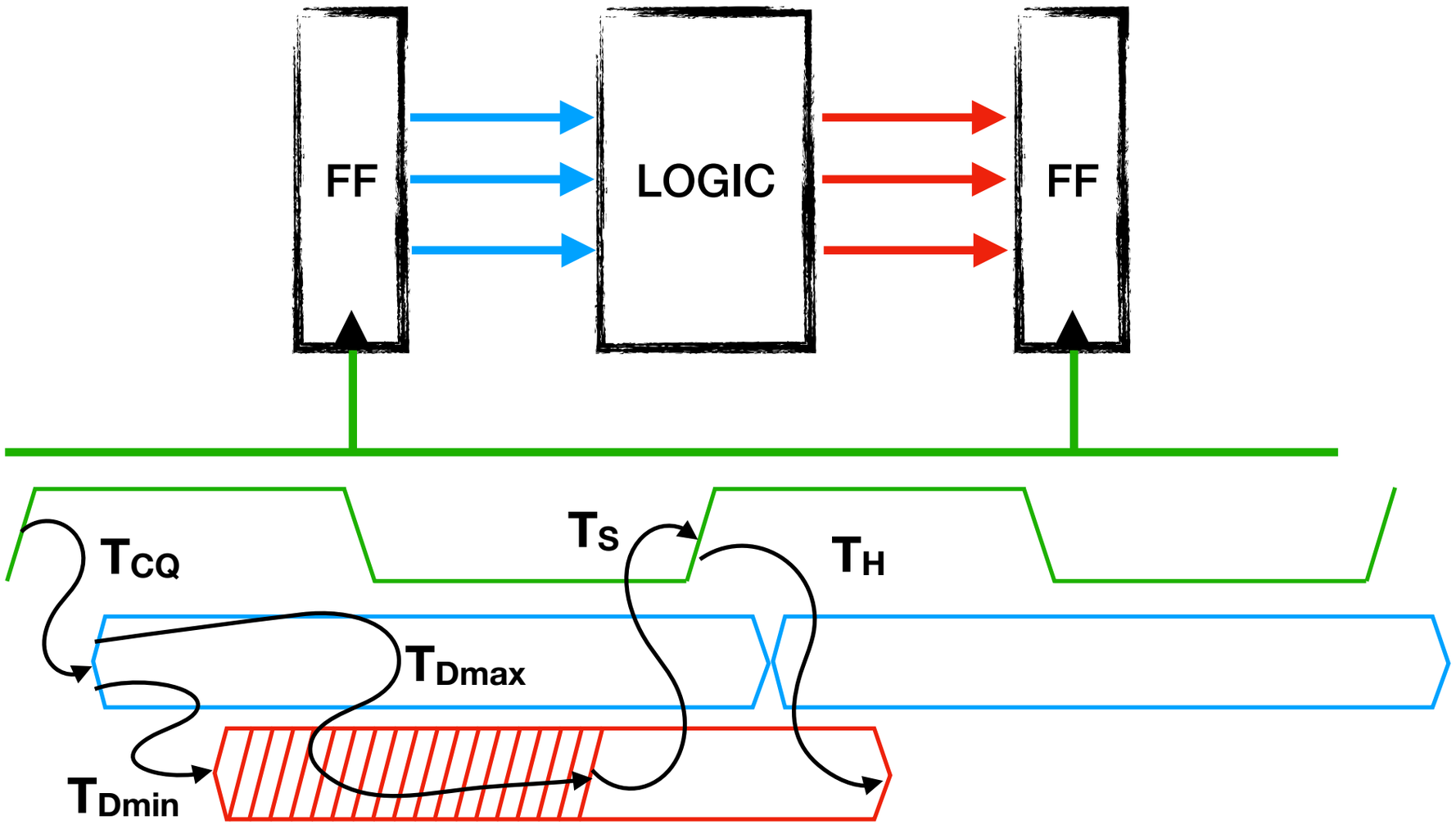
- Capture logic values using clock signal
  - Realign logic values in time; send on to next logic block

A more detailed view



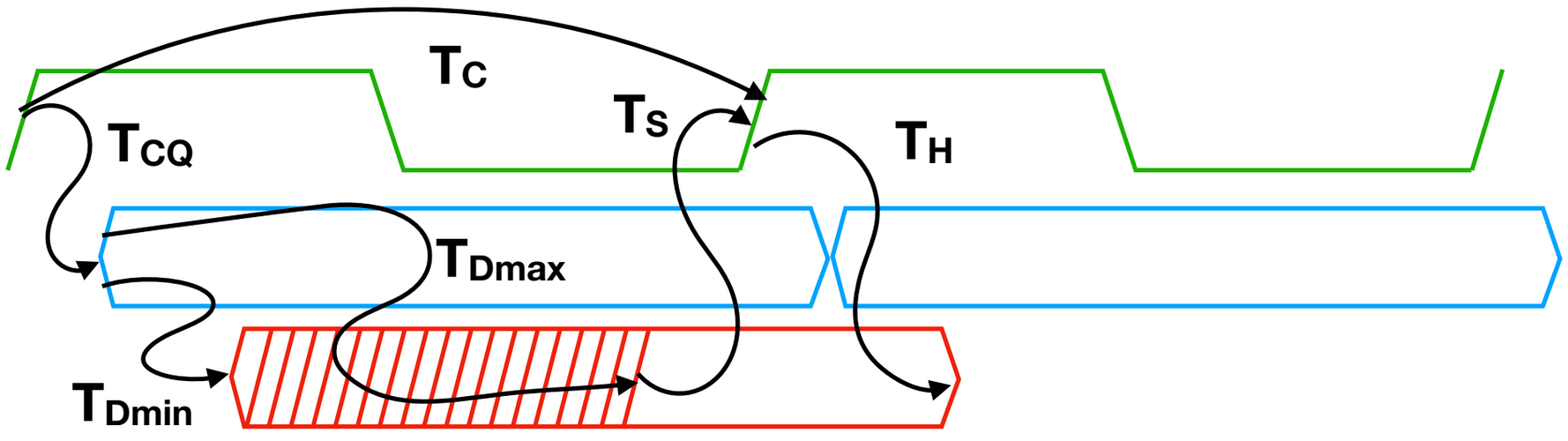
- Capture elements (FFs) contribute to overall delay
- Clock-to-Q delay T_{CQ}, setup time T_S, hold time T_H

FF time parameters



- T_S, T_H: the <u>shortest</u> times needed for FF to capture the correct data
- T_{CQ}: the <u>shortest</u> time from clock edge to output change

Safe T_C ?



- Setup criterion: $T_C > T_{CQ} + T_{Dmax} + T_S$
- Hold criterion: $T_{CQ} + T_{Dmin} > T_H$
- Add inequalities: $T_C + T_{CQ} + T_{Dmin} > T_{CQ} + T_{Dmax} + T_S + T_H$

$$T_C + T_{Dmin} > T_{Dmax} + T_S + T_H$$

$$T_C > T_{Dmax} - T_{Dmin} + T_S + T_H$$

Observations

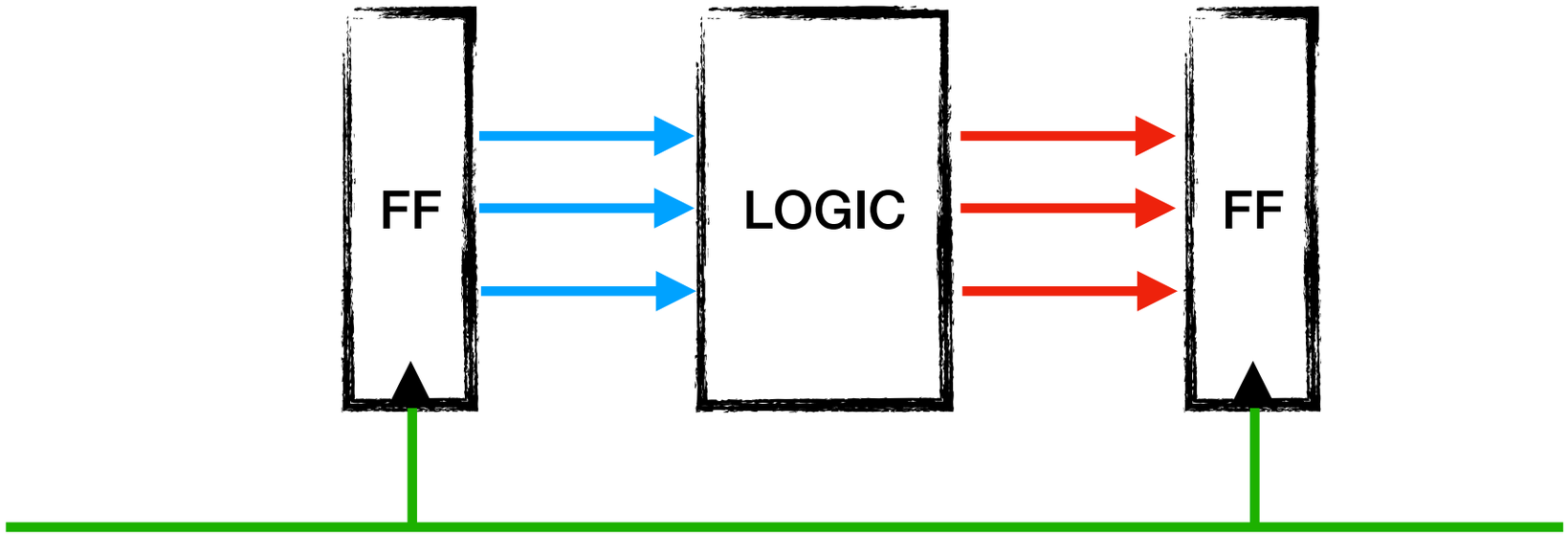
- Shortest T_C given by

$$T_C > T_{Dmax} - T_{Dmin} + T_S + T_H$$

  - Reduce delay difference to be able to reduce T_C
  - T_C limited by $T_S + T_H$, not by T_{Dmax}!
- T_{Dmin} is not the shortest time for operation completion
  - Rather, time to earliest output transition (contamination delay)

[L.W. Cotten, Maximum-rate pipeline systems, 1969]

Choice of f_C

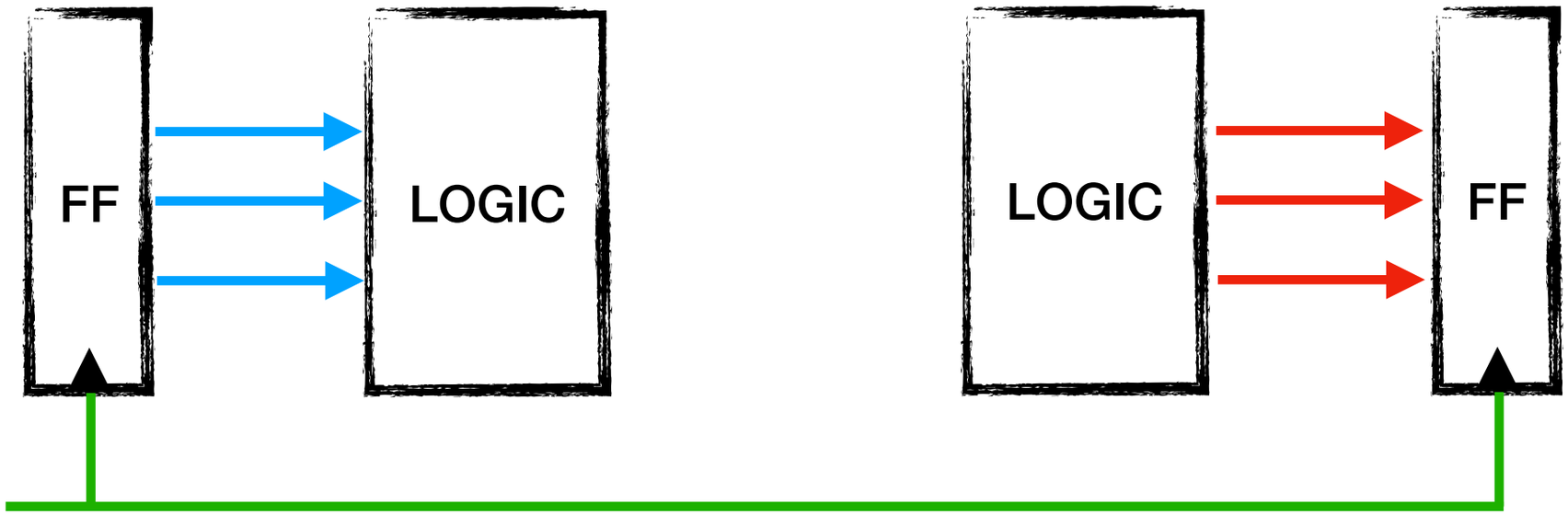


- High f_C enables higher performance
- Split logic block in 2, insert FF between parts!
- Hopefully, $T_{Dmax} - T_{Dmin}$ is reduced (halved?)

but

- Need $2 \cdot (T_S + T_H)$ for the same logic function …
- < 2x speedup; diminishing returns for more stages

Choice of f_C

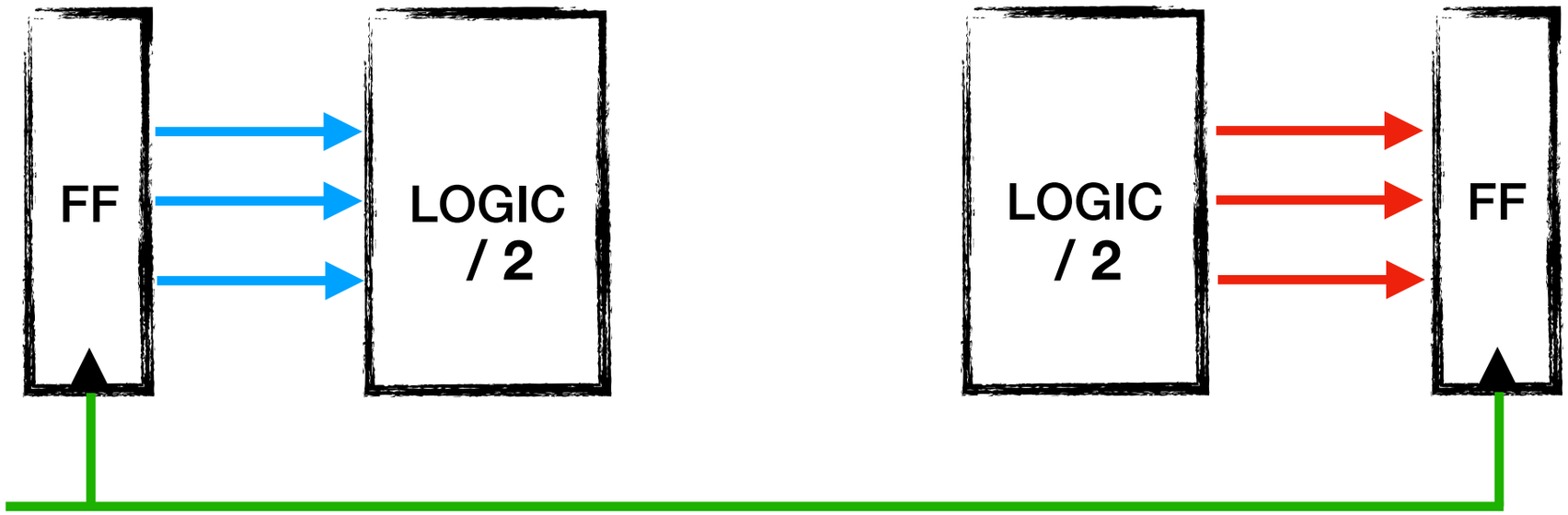


- High f_C enables higher performance
- Split logic block in 2, insert FF between parts!
- Hopefully, $T_{Dmax} - T_{Dmin}$ is reduced (halved?)

but

- Need $2 \cdot (T_S + T_H)$ for the same logic function …
- < 2x speedup; diminishing returns for more stages

Choice of f_C

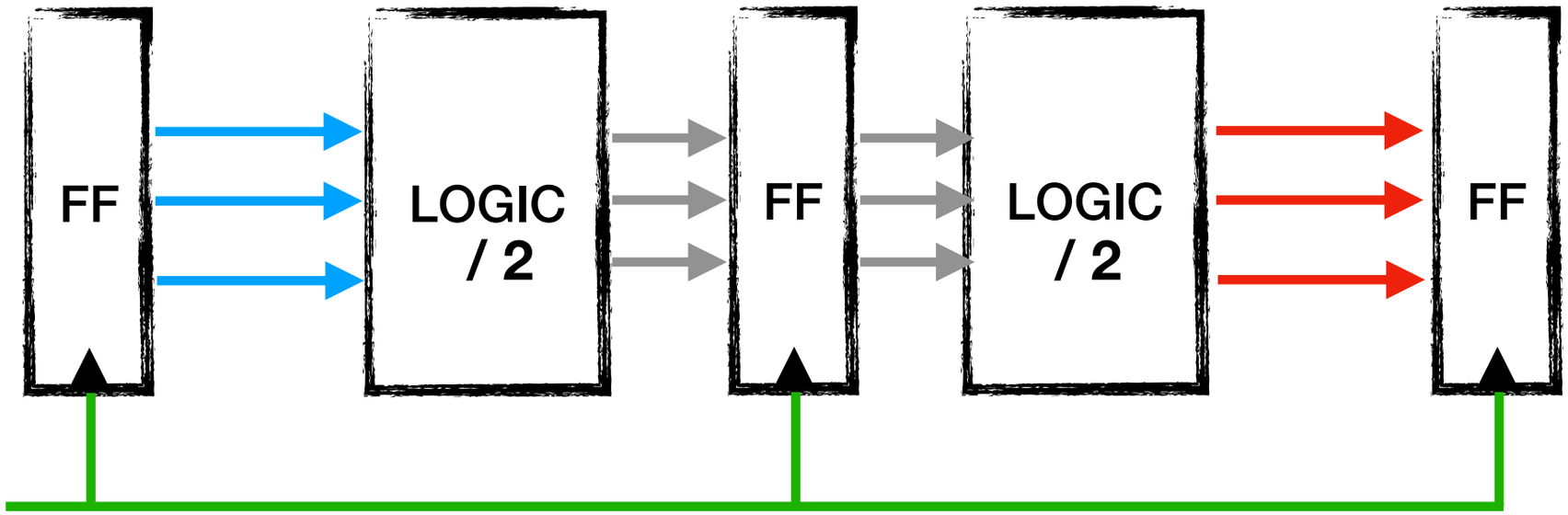


- High f_C enables higher performance
- Split logic block in 2, insert FF between parts!
- Hopefully, $T_{Dmax} - T_{Dmin}$ is reduced (halved?)

but

- Need $2 \cdot (T_S + T_H)$ for the same logic function …
- < 2x speedup; diminishing returns for more stages

Choice of f_C

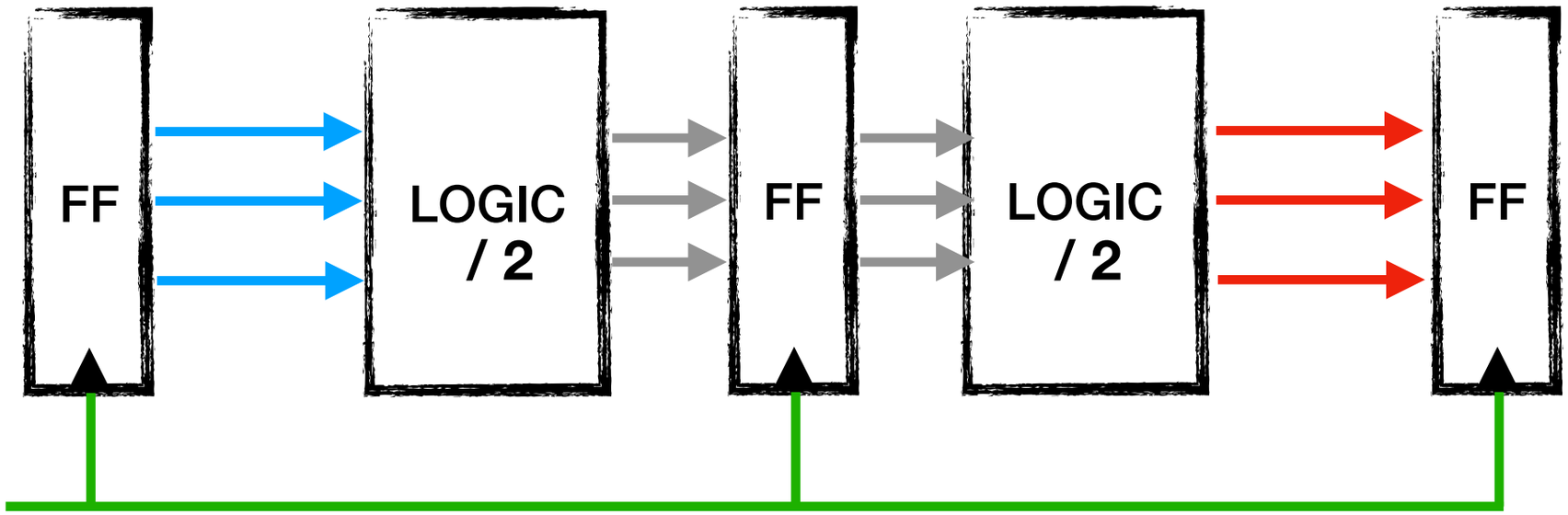


- High f_C enables higher performance
- Split logic block in 2, insert FF between parts!
- Hopefully, $T_{Dmax} - T_{Dmin}$ is reduced (halved?)

but

- Need $2 \cdot (T_S + T_H)$ for the same logic function …
- < 2x speedup; diminishing returns for more stages

Choice of f_C



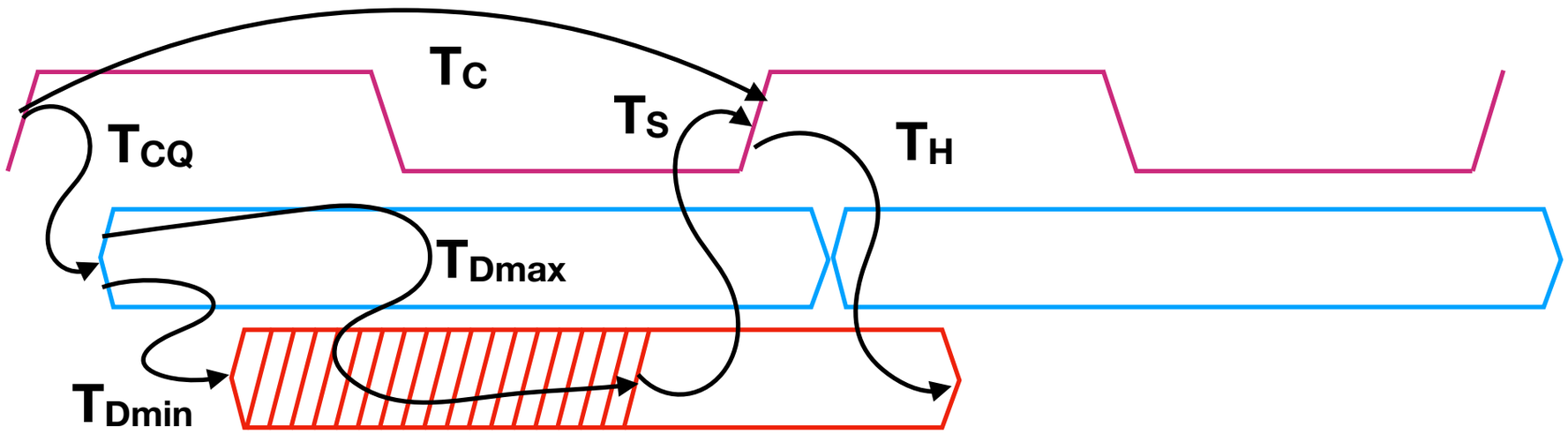
- High f_C enables higher performance
- Split logic block in 2, insert FF between parts!
- Hopefully, $T_{Dmax} - T_{Dmin}$ is reduced (halved?)

but

- Need $2 \cdot (T_S + T_H)$ for the same logic function …
- < 2x speedup; diminishing returns for more stages

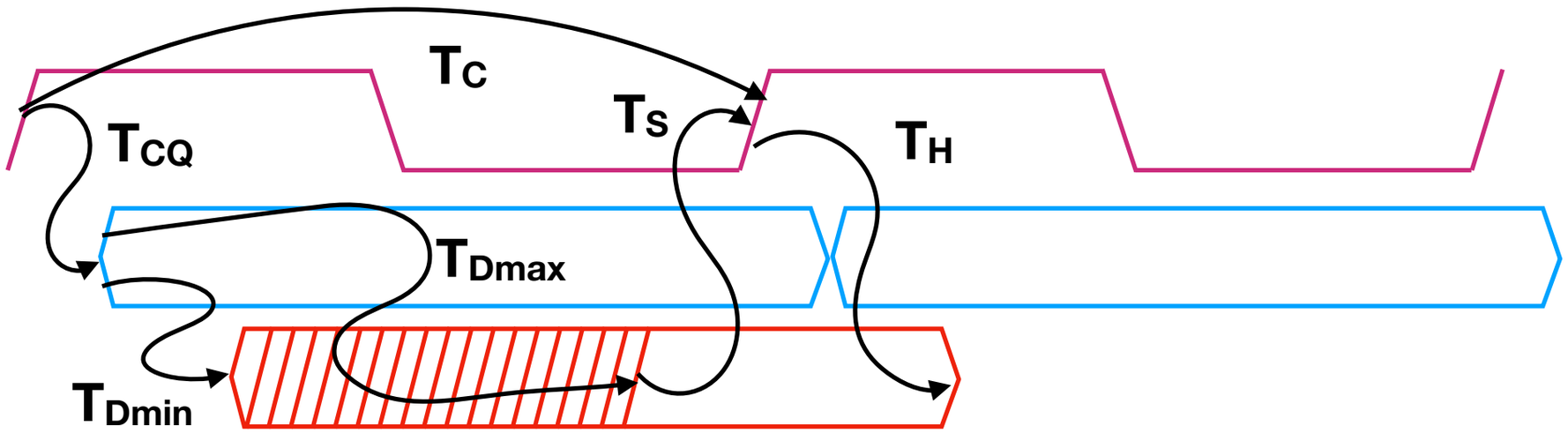
Non-straight pipelines add complications

Setup violation



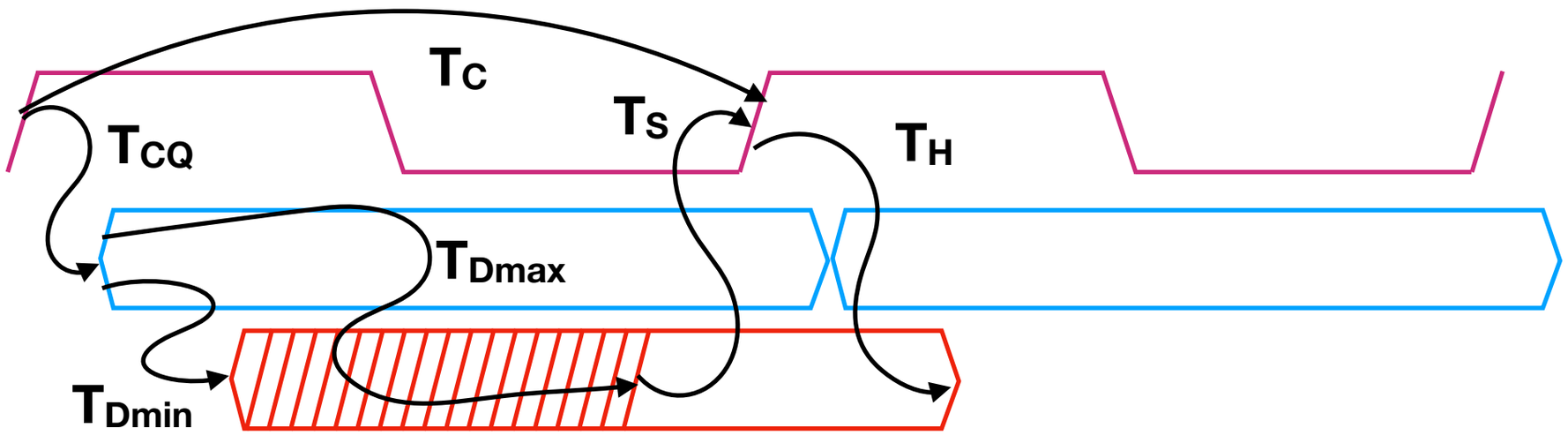
- Setup <u>criterion</u>: $T_C > T_{CQ} + T_{Dmax} + T_S$
- Setup <u>violation</u>: $T_C < T_{CQ} + T_{Dmax} + T_S$
- Two ways to work around setup violations on the bench:
  - Increase T_C, that is, reduce f_C
  - Reduce circuit delays $T_{CQ} + T_{Dmax} + T_S$, e.g. increase V_{dd}

Hold violation



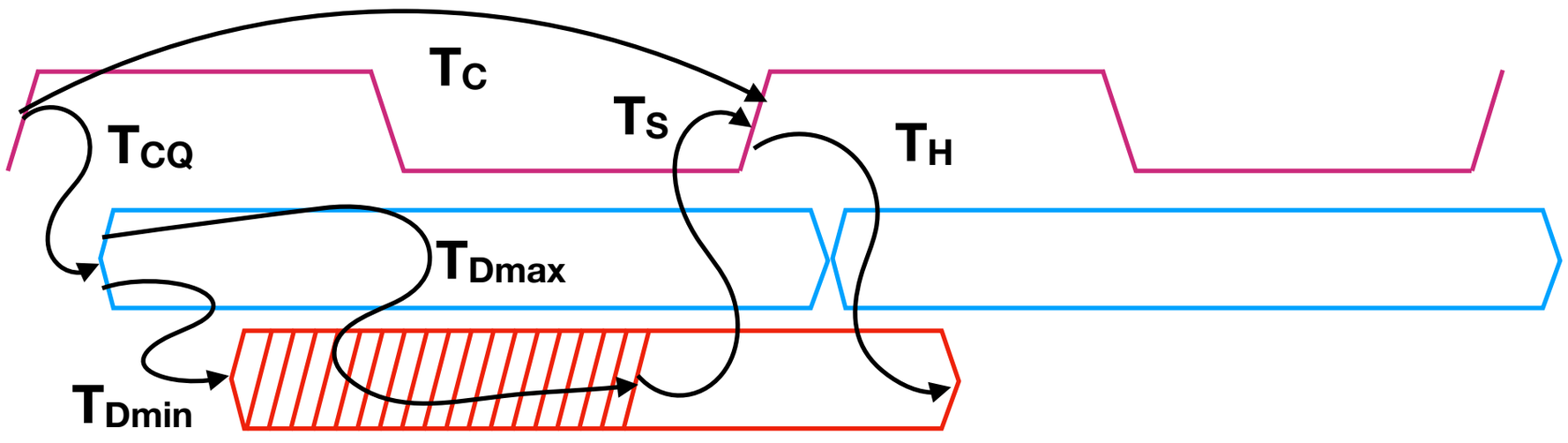
- Hold <u>criterion</u>: $T_{CQ} + T_{Dmin} > T_H$
- Hold <u>violation</u>: $T_{CQ} + T_{Dmin} < T_H$
- Only circuit delays in these expressions
- No obvious way to “patch” a hold violation!

Jitter



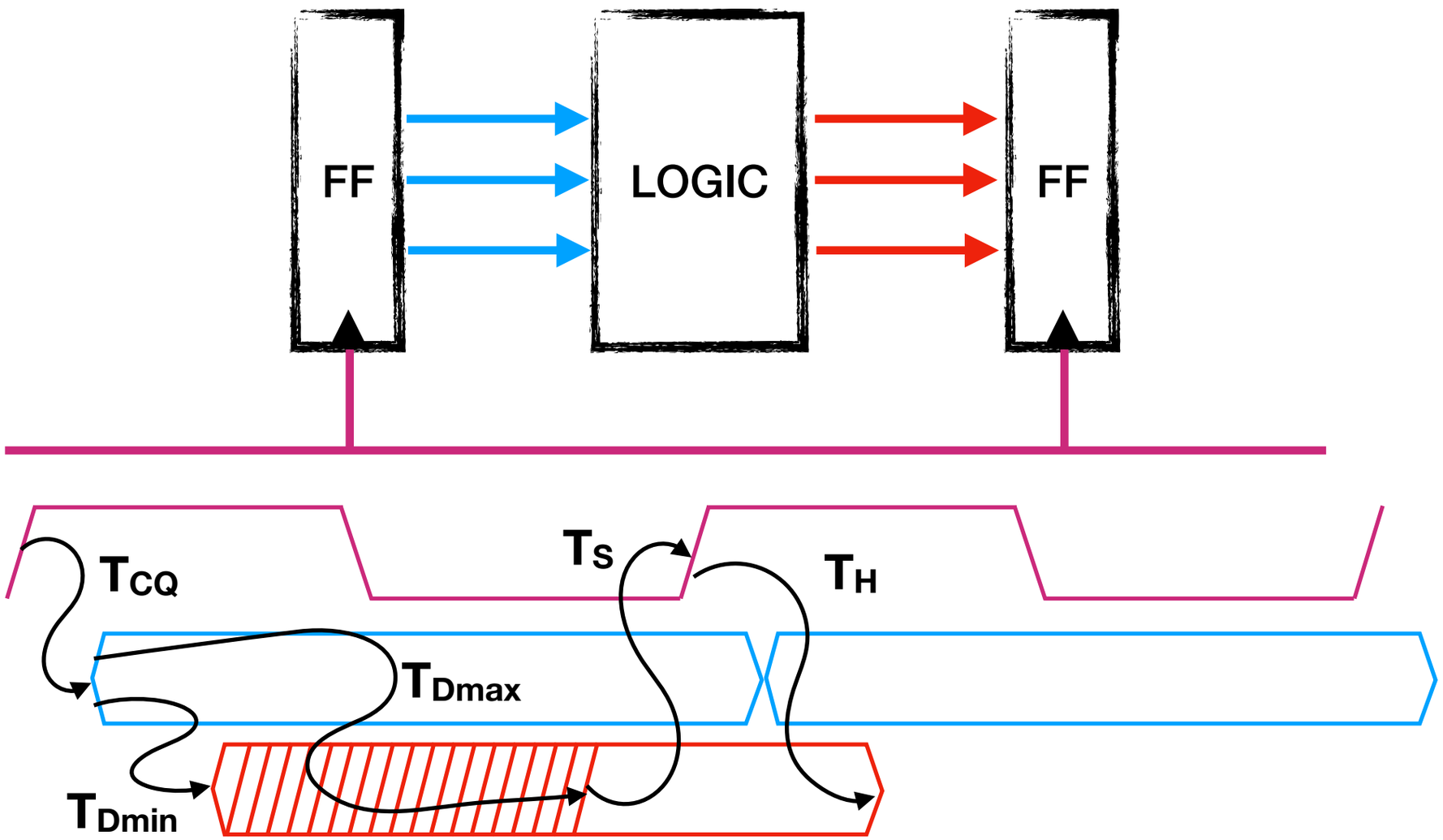
- Jitter: T_C varies from one cycle to the next
  - Random, or sometimes <u>intentional</u>
    - Reduce peak power emission at f_C
- Setup criterion must be fulfilled with <u>minimum</u> T_C value

Jitter



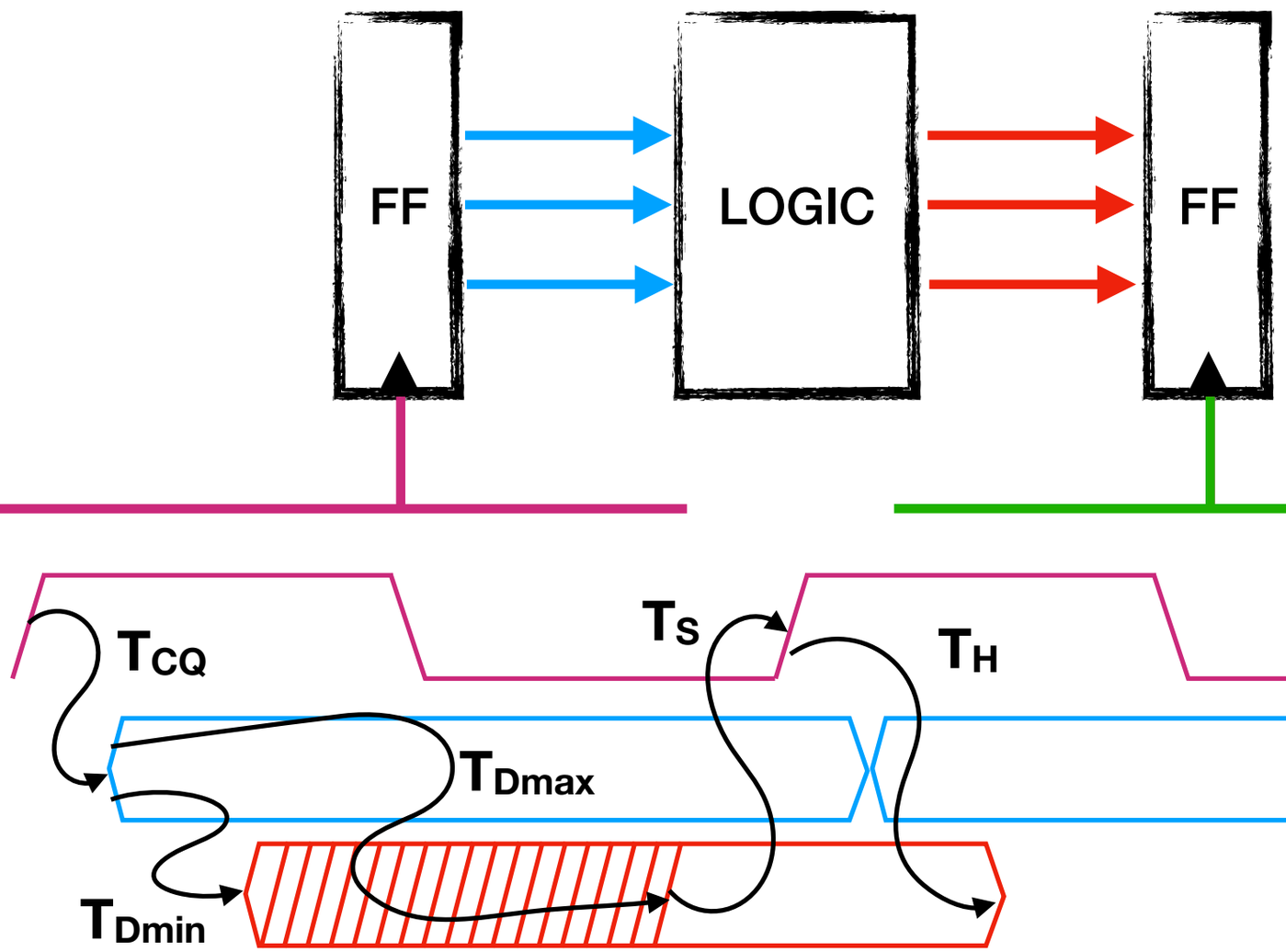
- Jitter: T_C varies from one cycle to the next
  - Random, or sometimes <u>intentional</u>
    - Reduce peak power emission at f_C
- Setup criterion must be fulfilled with <u>minimum</u> T_C value

Skew



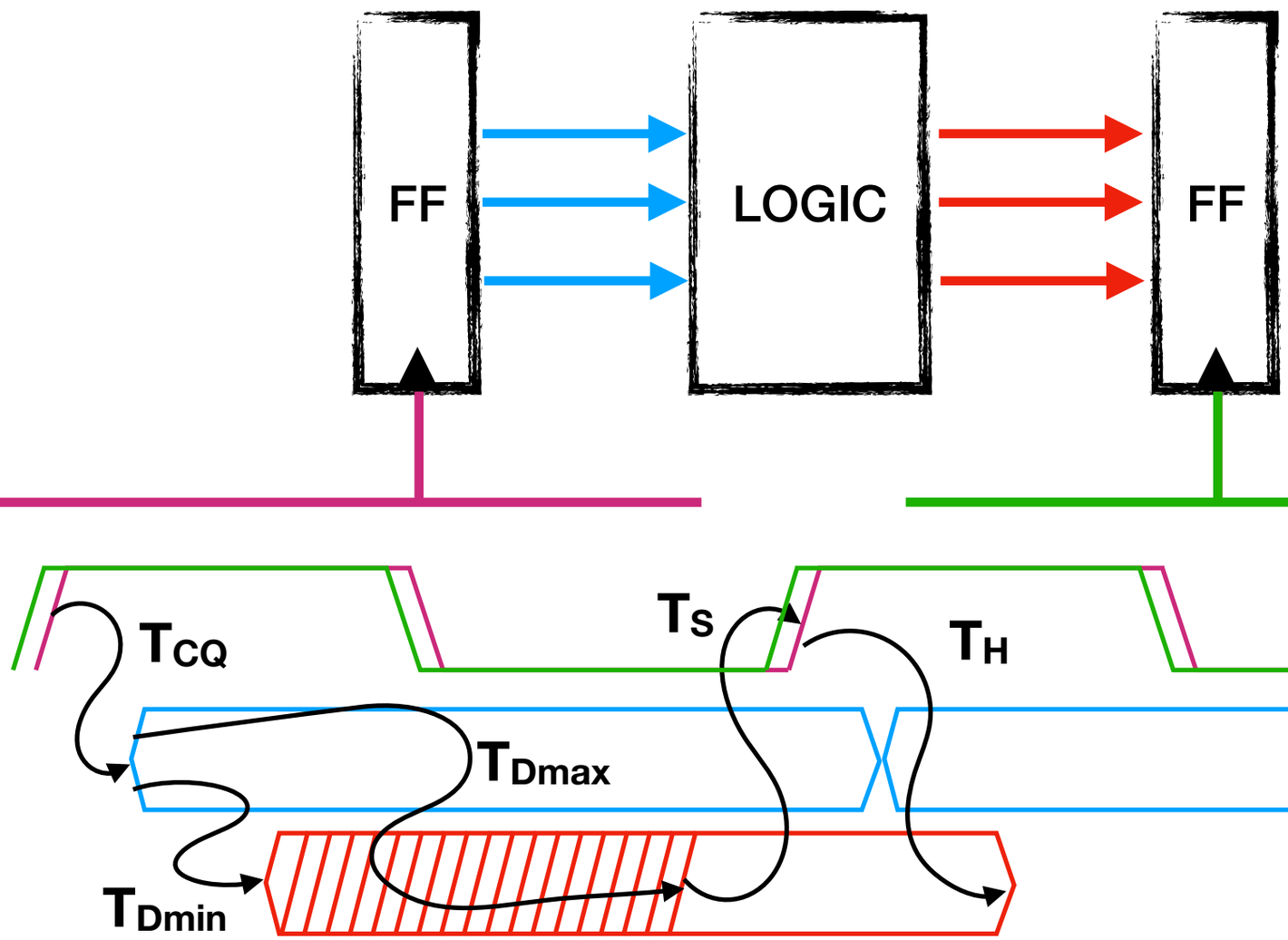
- Sending and receiving FF clocks out of phase
- Receive clock early: setup problems; late: hold problems

Skew



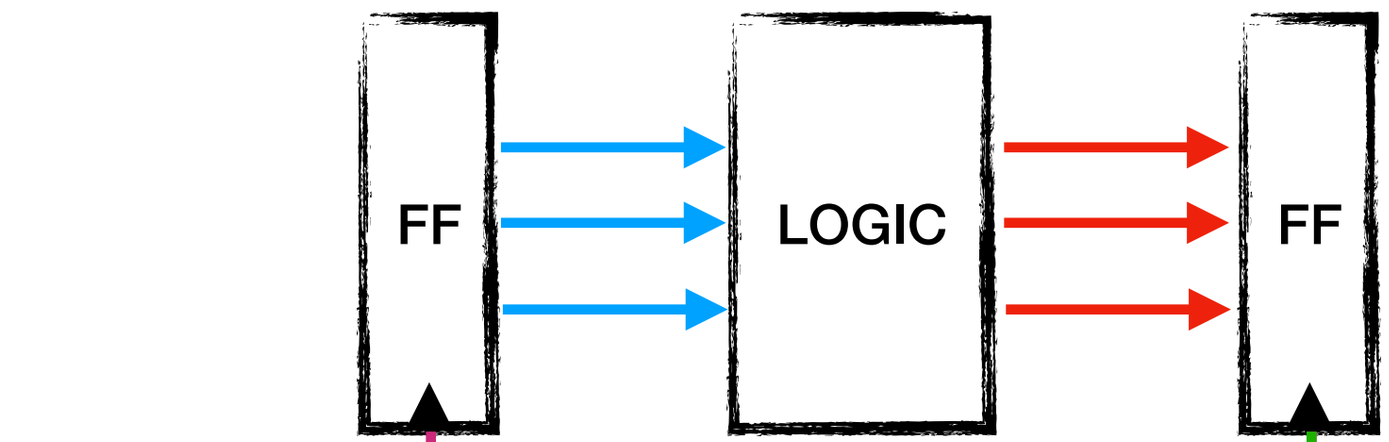
- Sending and receiving FF clocks out of phase
- Receive clock early: setup problems; late: hold problems

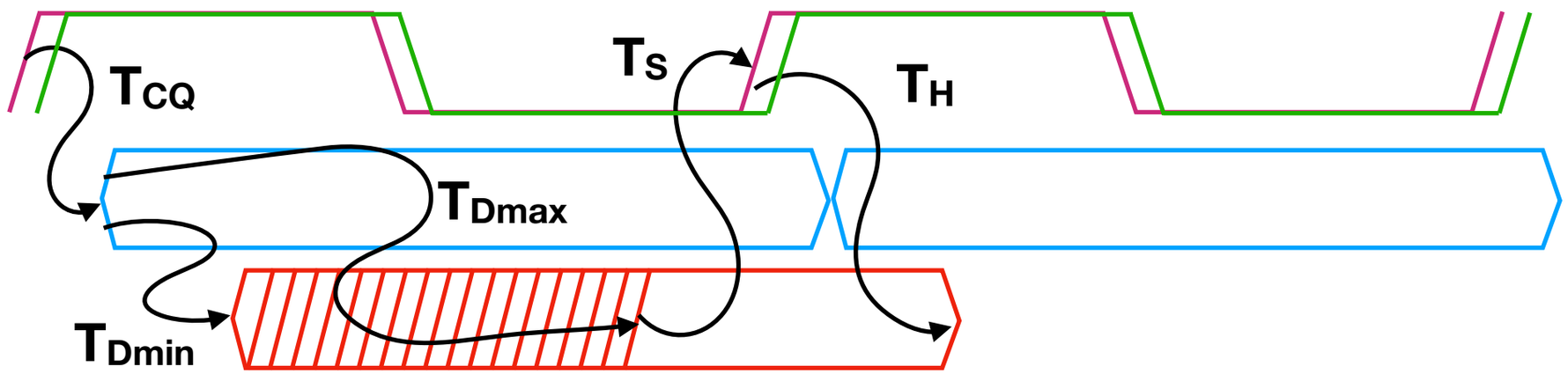
Skew



- Sending and receiving FF clocks out of phase
- Receive clock early: setup problems; late: hold problems

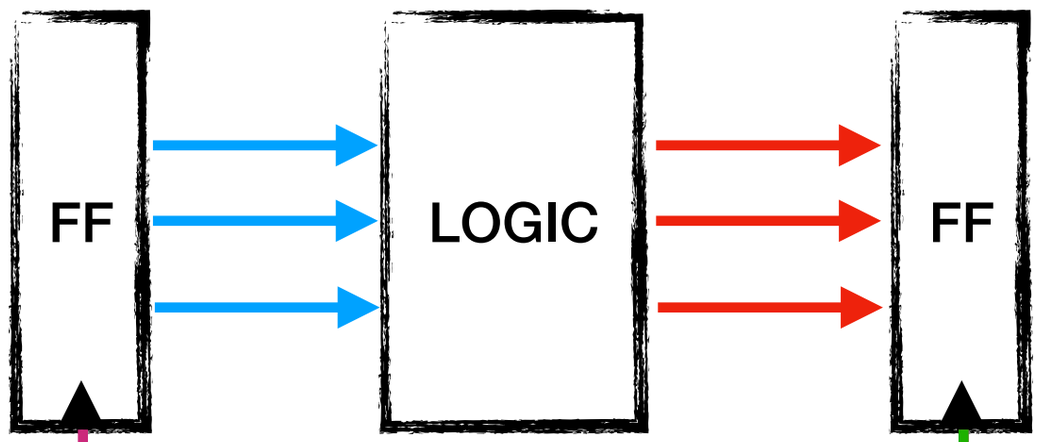
Skew



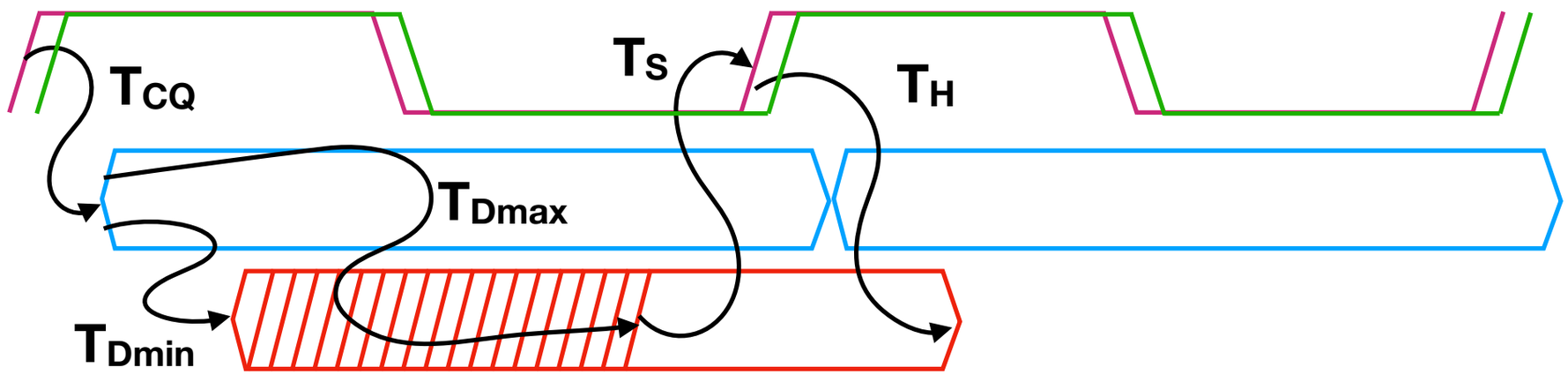


- Sending and receiving FF clocks out of phase
- Receive clock early: setup problems; late: hold problems

Skew

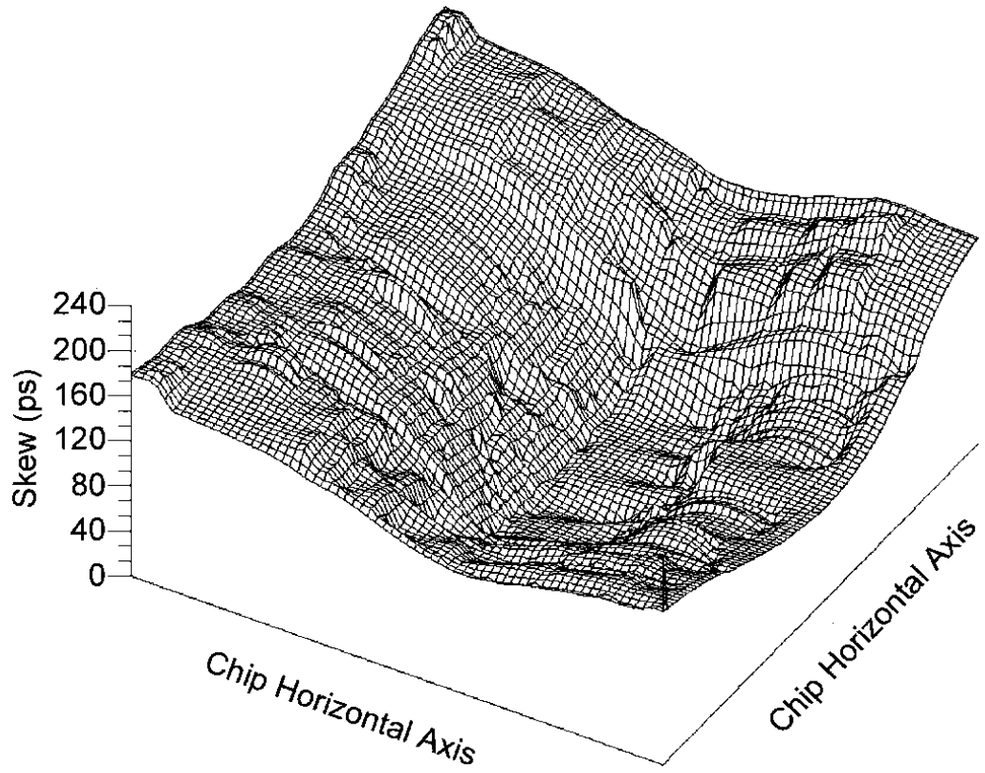


Could you see a use for intentional skew?



- Sending and receiving FF clocks out of phase
- Receive clock early: setup problems; late: hold problems

Large-scale clock distribution



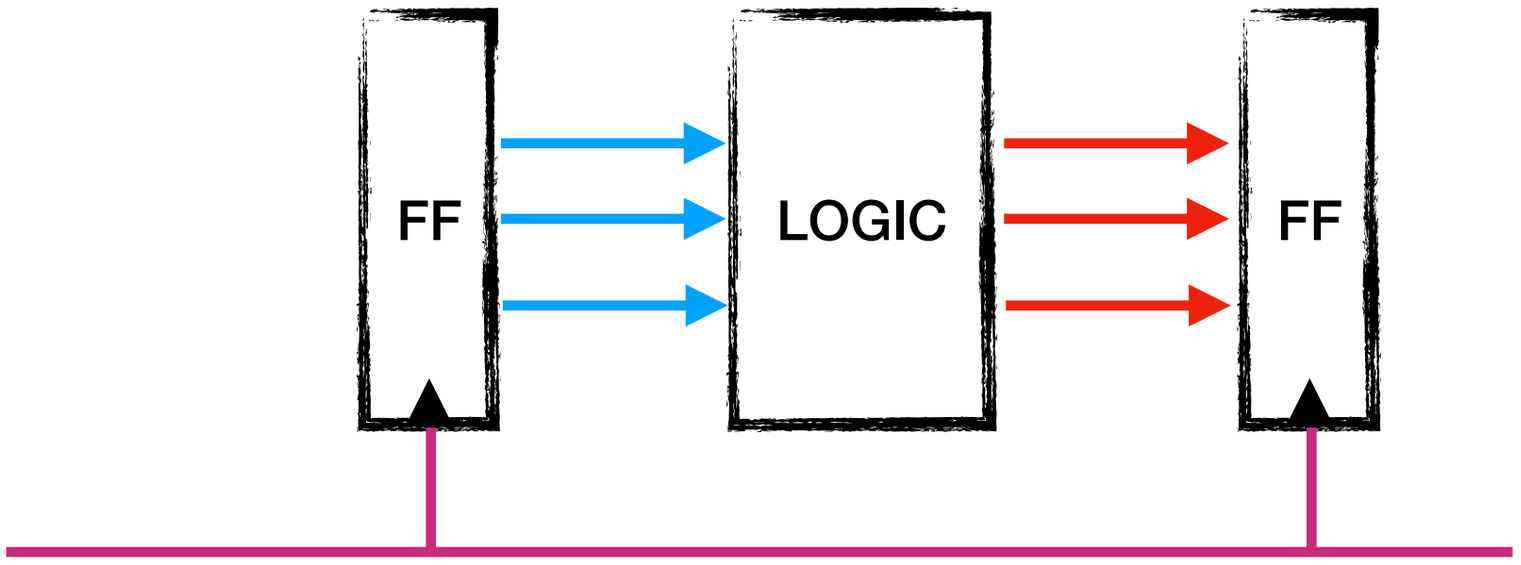
- 200-MHz microprocessor (T_C = 5 ns)
- Central clock driver, skew increases with distance
  - At worst ~5% of T_C
- Extrapolate to f_C = 2 GHz!

[Gronowski et al. High-Performance Microprocessor design. IEEE JSSC 1998.]

Complications

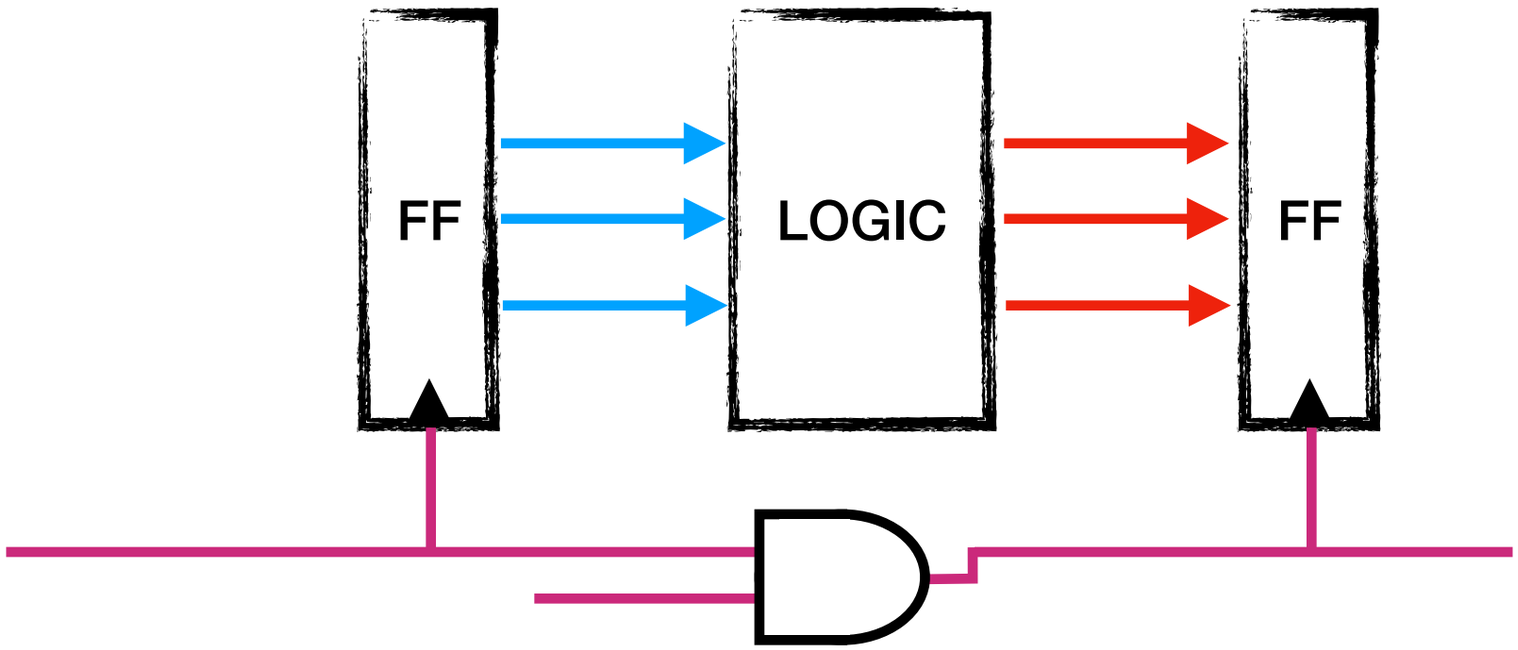
- Clock gating
- Several clock domains
- Voltage scaling

Clock gating



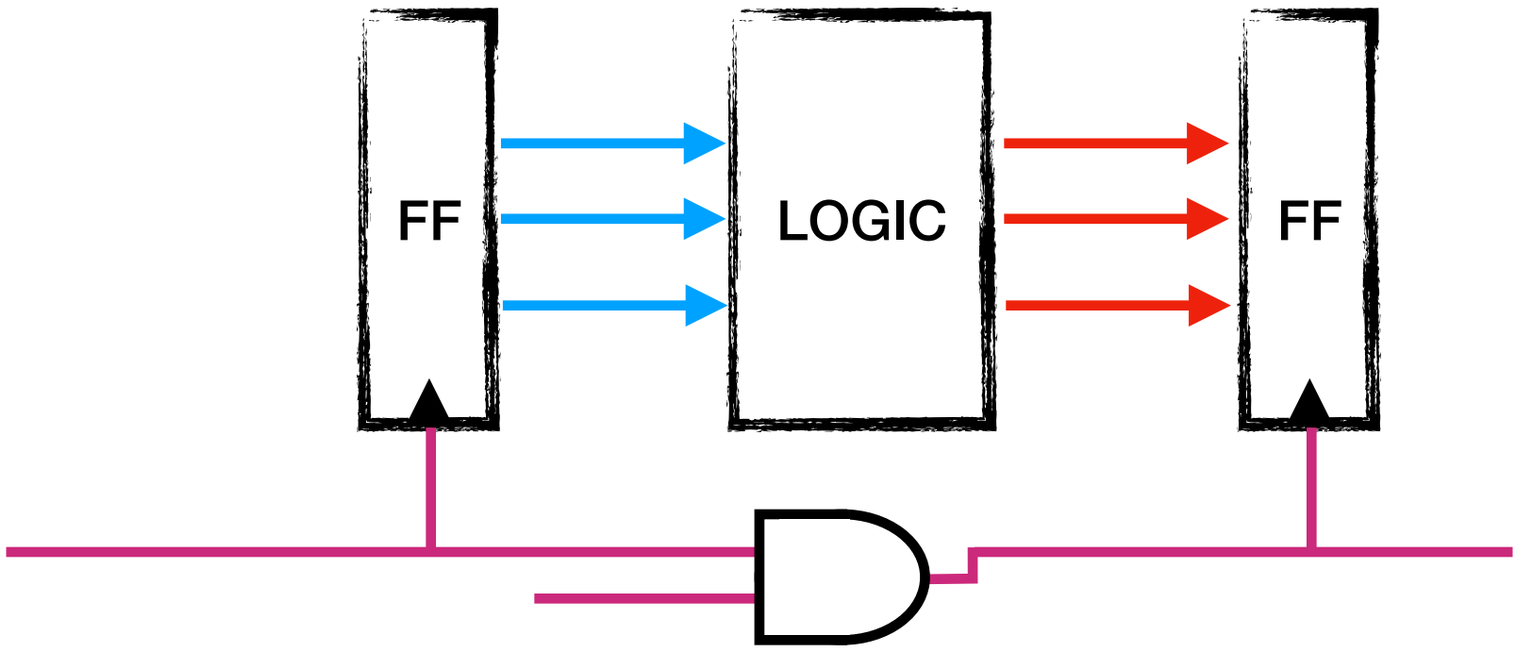
- Recall expression for switching power: $P = ß \cdot f_C \cdot C \cdot V^2$
  - Clock signals switch every cycle, so ß = 1 (max value)
  - Also a large net, so large C
- Driving clock itself may cause large part of total dissipation!
- Disable clock when FF data is known to not change!

Clock gating



- Recall expression for switching power: $P = ß \cdot f_C \cdot C \cdot V^2$
  - Clock signals switch every cycle, so ß = 1 (max value)
  - Also a large net, so large C
- Driving clock itself may cause large part of total dissipation!
- Disable clock when FF data is known to not change!

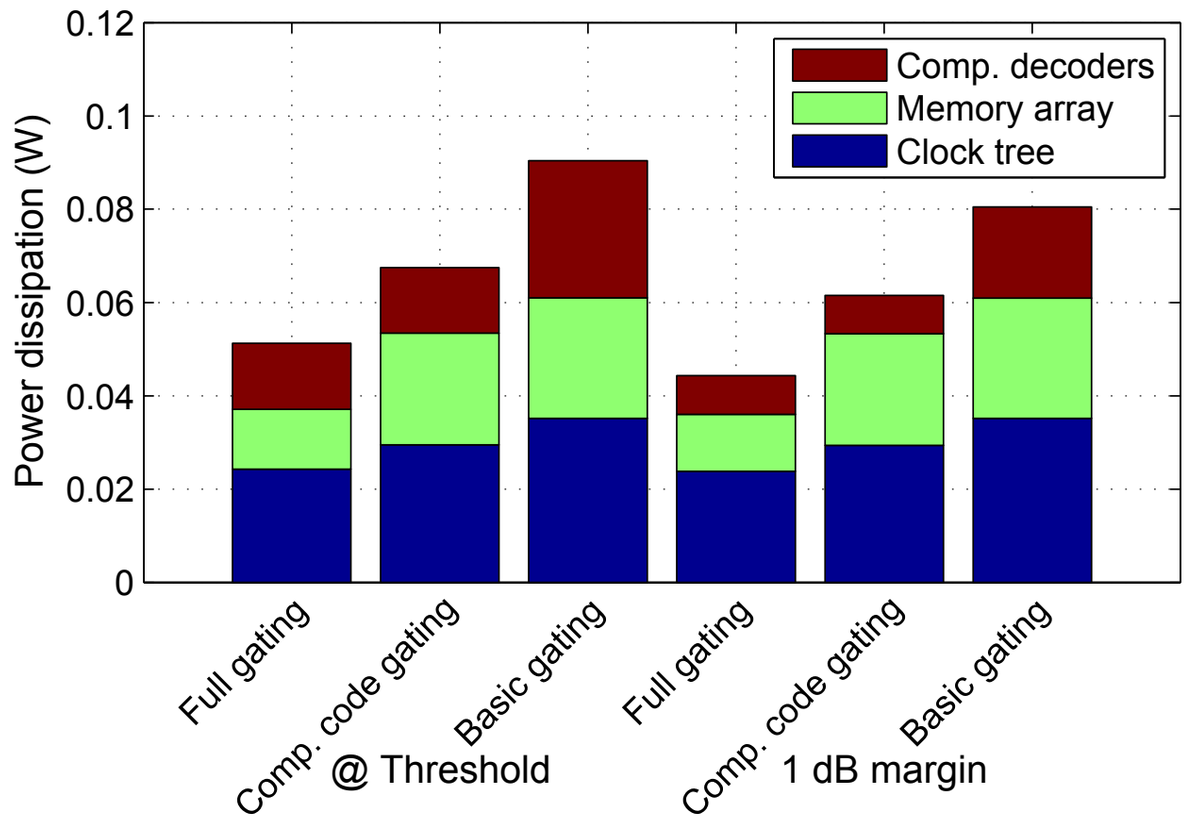
Clock gating



Possible problems?

- Recall expression for switching power: $P = ß \cdot f_C \cdot C \cdot V^2$
  - Clock signals switch every cycle, so ß = 1 (max value)
  - Also a large net, so large C
- Driving clock itself may cause large part of total dissipation!
- Disable clock when FF data is known to not change!

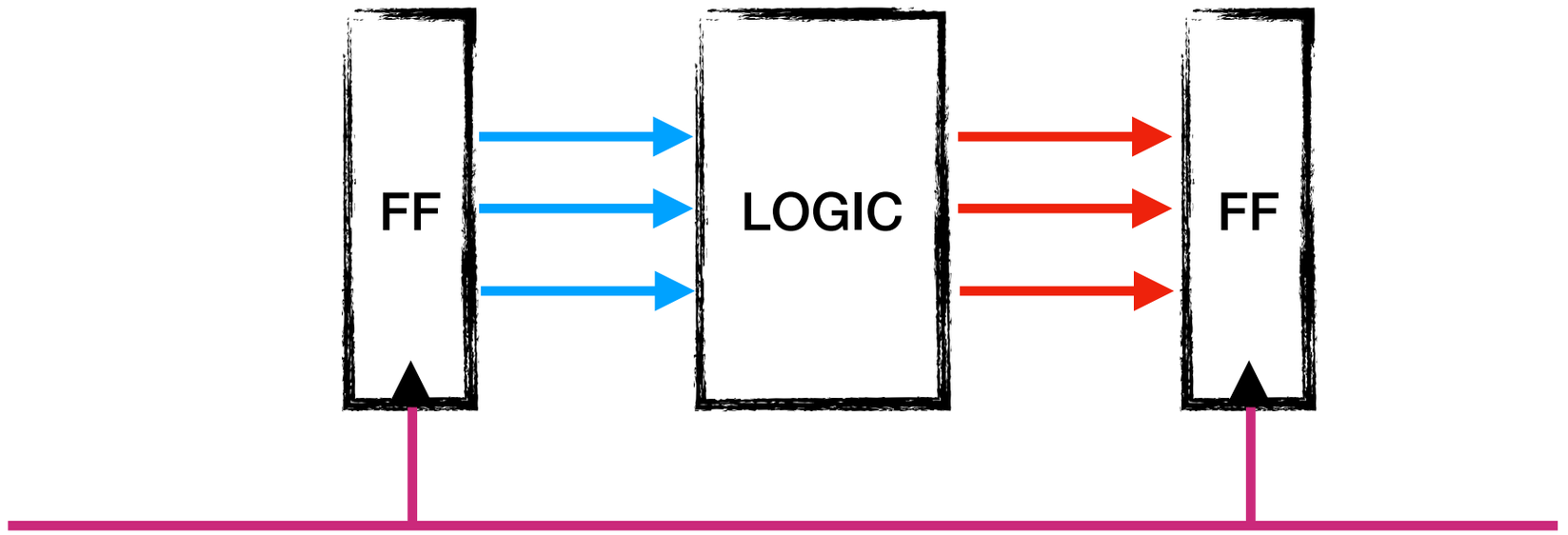
Clock gating, cont.



- Example:
  - Forward Error Correction (FEC) unit
  - Low activity when no errors to correct!
- Reduces clock power, but also power in logic and memories
  - Very useful technique (more in separate lecture)

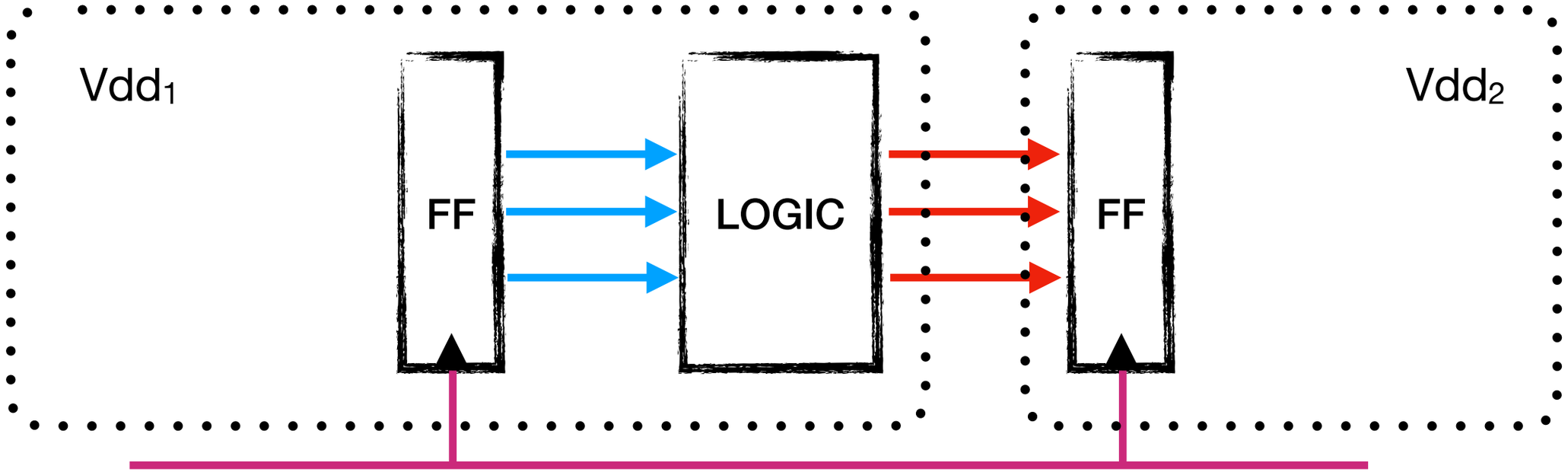
[Fougstedt, private communication]

Dynamic voltage scaling



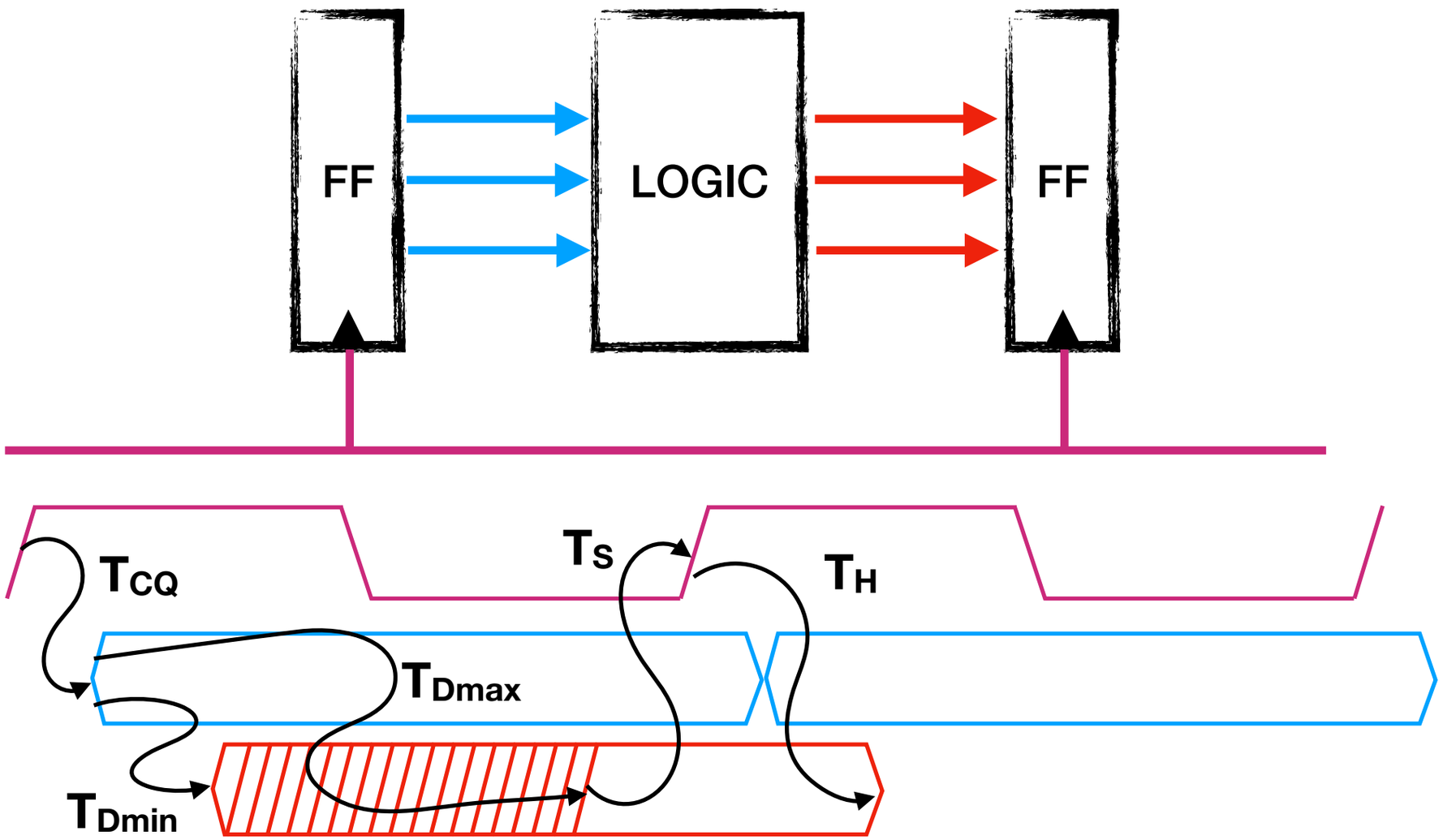
- Idea: selectively reduce supply voltage for part of system to save power
  - Logic speed affected, so must adapt also f_C …
- What happens with setup and hold criteria when supply voltage changes for some components?

Dynamic voltage scaling



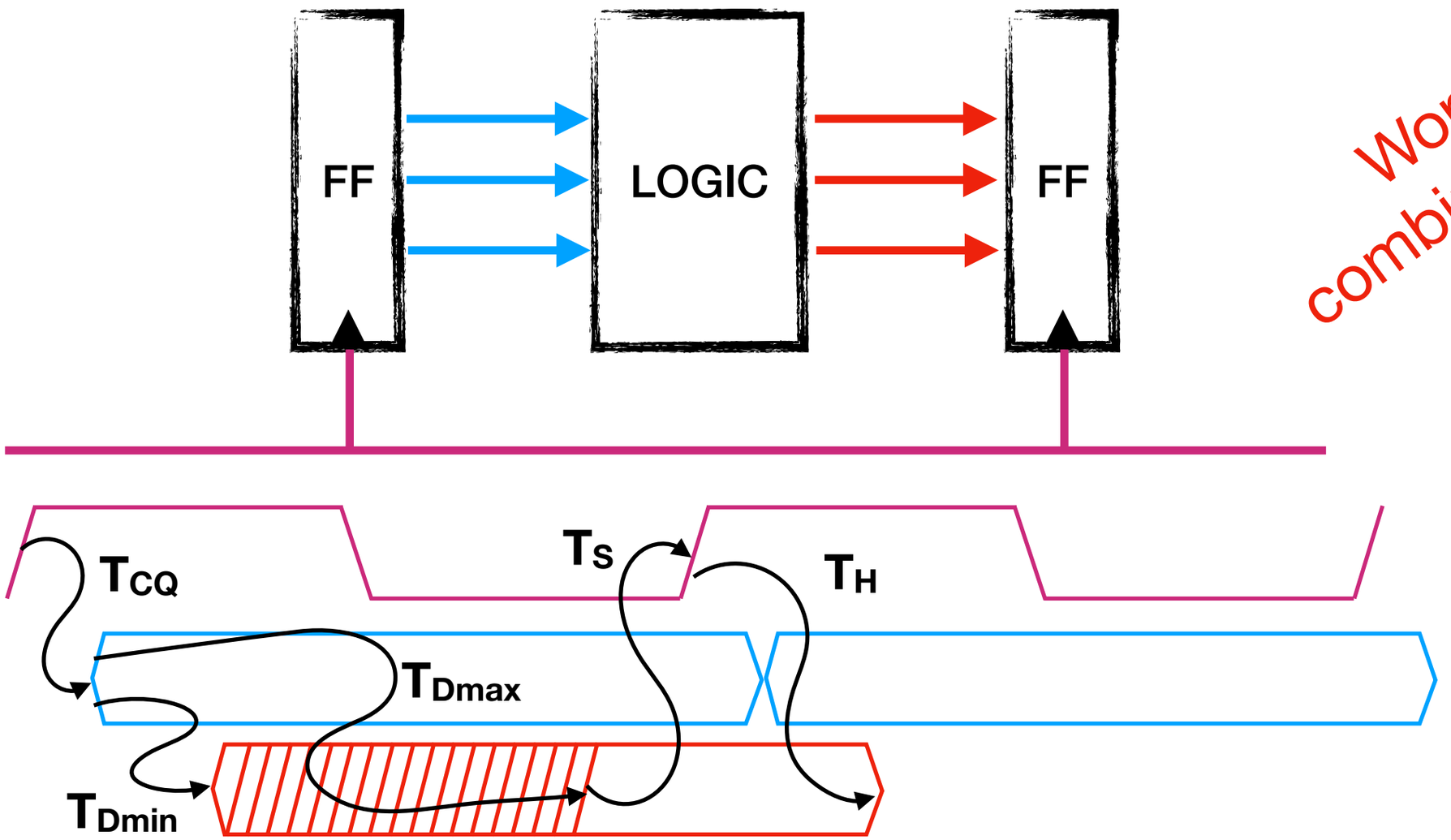
- Idea: selectively reduce supply voltage for part of system to save power
  - Logic speed affected, so must adapt also f_C …
- What happens with setup and hold criteria when supply voltage changes for some components?

DVS, cont.



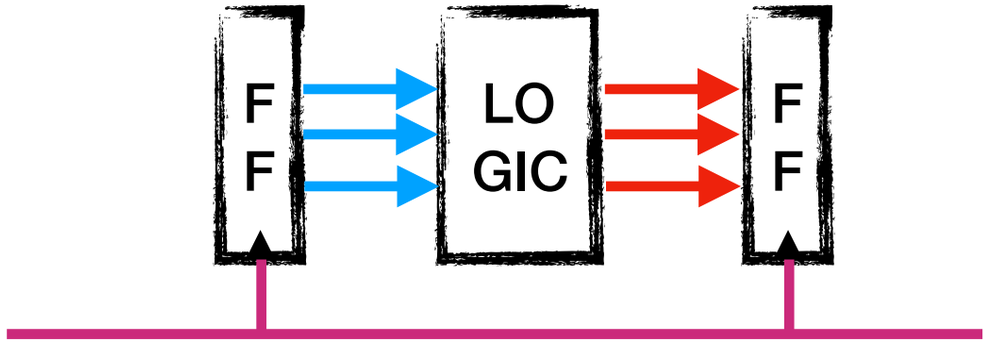
- Reduce overall supply voltage? All T delays increase.
- Reduce supply for FF_1, LOGIC, FF_2? Some delays increase…

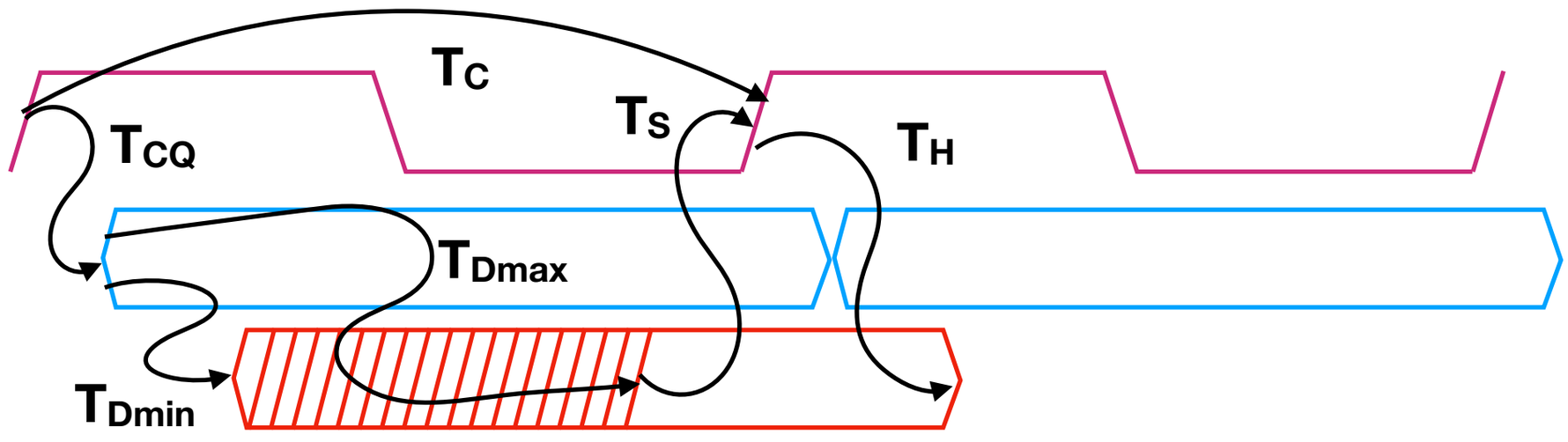
DVS, cont.



- Reduce overall supply voltage? All T delays increase.
- Reduce supply for FF_1, LOGIC, FF_2? Some delays increase…

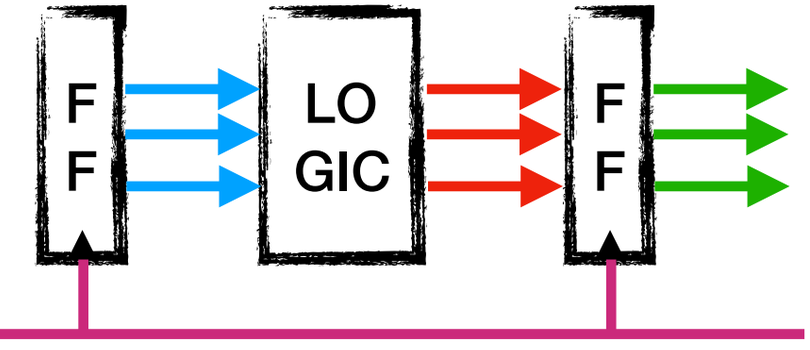
Metastability

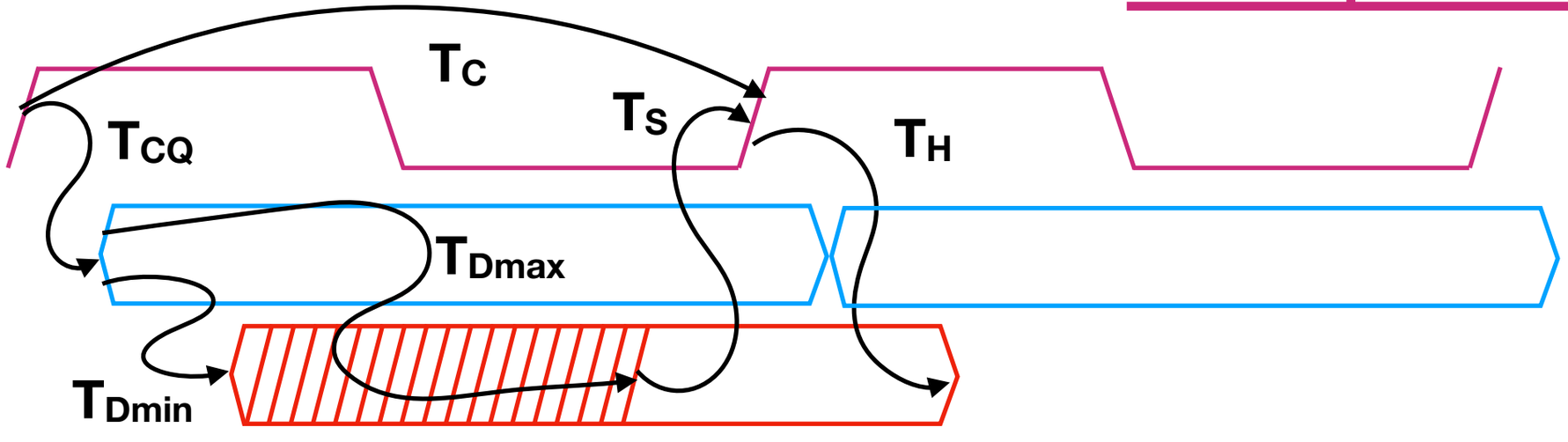




- Setup criterion: $T_C > T_{CQ} + T_{Dmax} + T_S$
  - What if > is replaced with =?
  - The input to the second FF changes very close to T_S before the clock edge
- What value will be captured?

Metastability, cont.



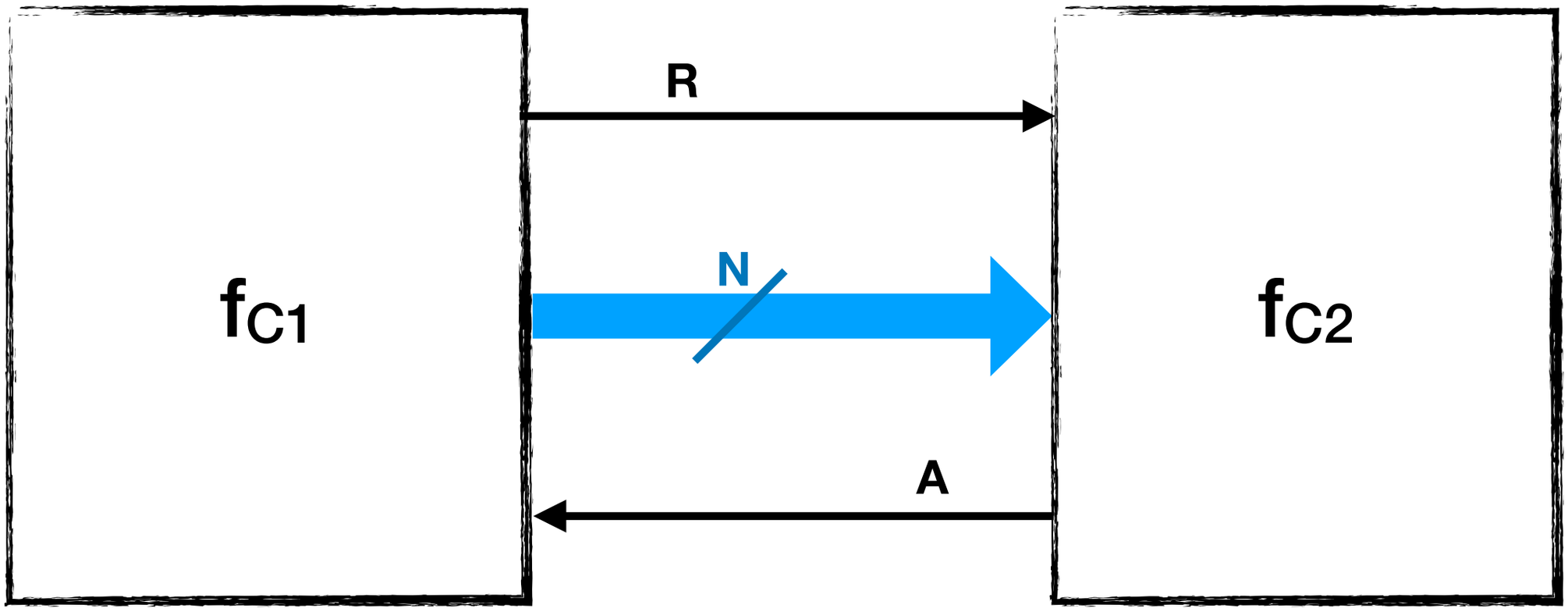


- A closer decision is more affected by electrical noise etc.
  - Randomness, so statistical description only
- On average, a closer decision takes longer to “resolve”
  - If “failure” is no decision after T_R, then

$$P(\text{fail}) \sim \text{const} \cdot \exp(-T_R)$$

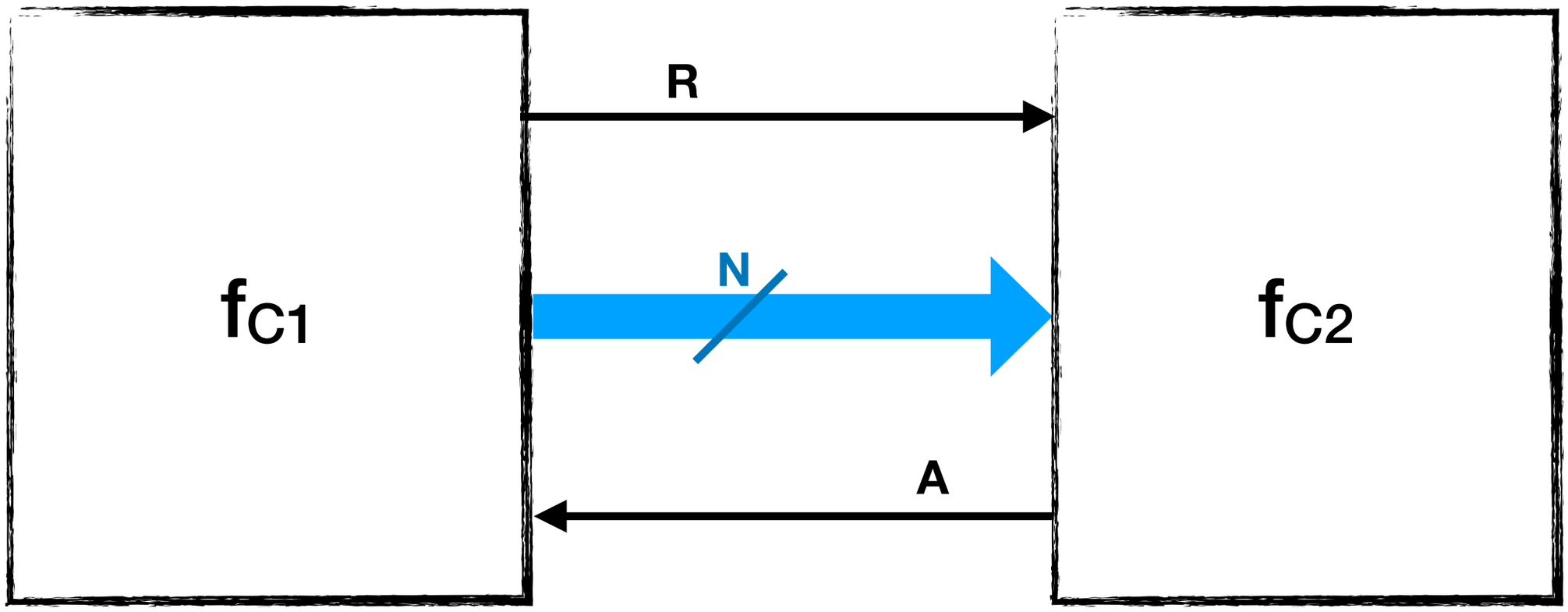
- Reduce P(fail) by reducing const and extending T_R

Several clock domains



- Two (or more) completely independent clock domains
  - Use handshaking protocol to transfer data (Ready, Ack)
- No rational relationship of f_{C1} and f_{C2} assumed
  - Clock and handshake transitions may (will) coincide occasionally 😖

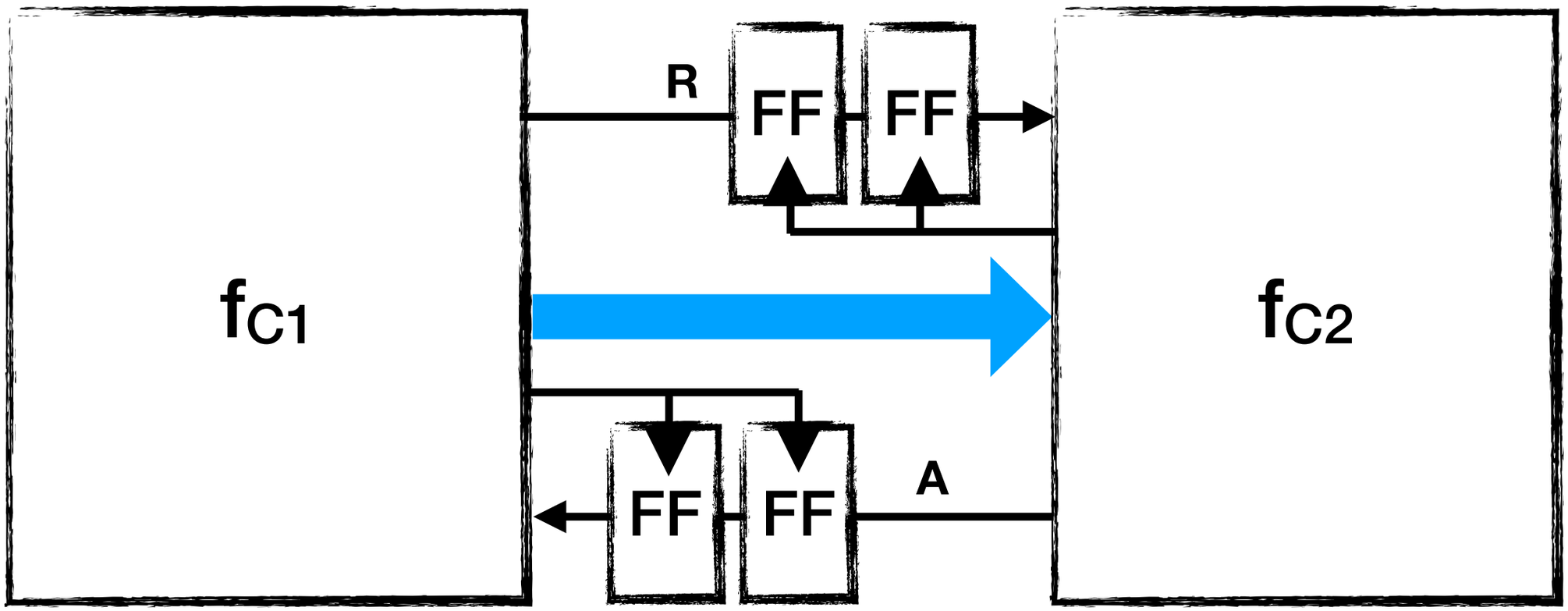
Several clock domains



- Two (or more) <u>completely independent</u> clock domains
  - Use handshaking protocol to transfer data (Ready, Ack)
- No rational relationship of f_{C1} and f_{C2} assumed
  - Clock and handshake transitions may (<u>will</u>) coincide occasionally 😖

GALS: Globally Asynchronous, Locally Synchronous

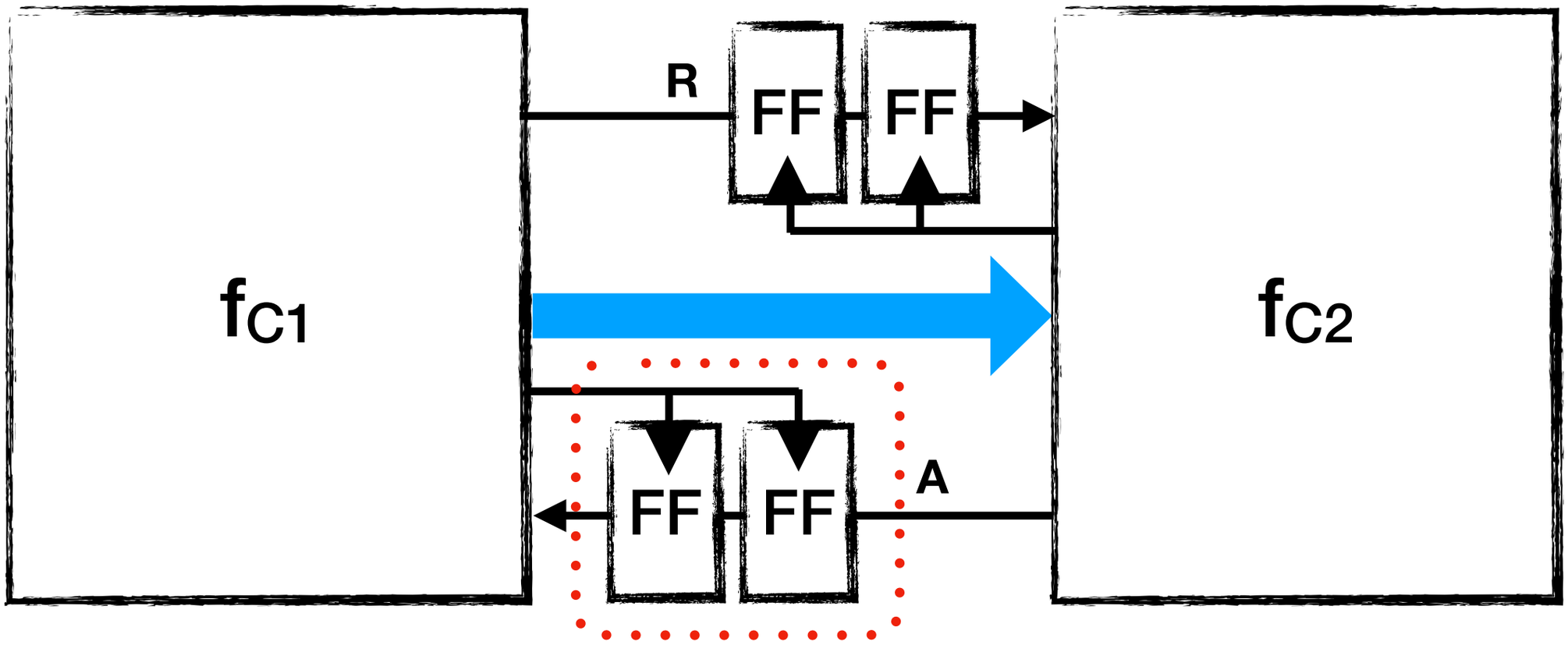
Synchronizers



- Standard solution: Use <u>two</u> FFs to receive handshake signals
  - Will survive a T_R of at least one full clock cycle (P(fail) ≈ 0)
- Note: simpler versions possible if only clock <u>phase</u> is different, etc

[Ran Ginosar. Fourteen ways to fool your synchronizer. ASYNC'03]

Synchronizers

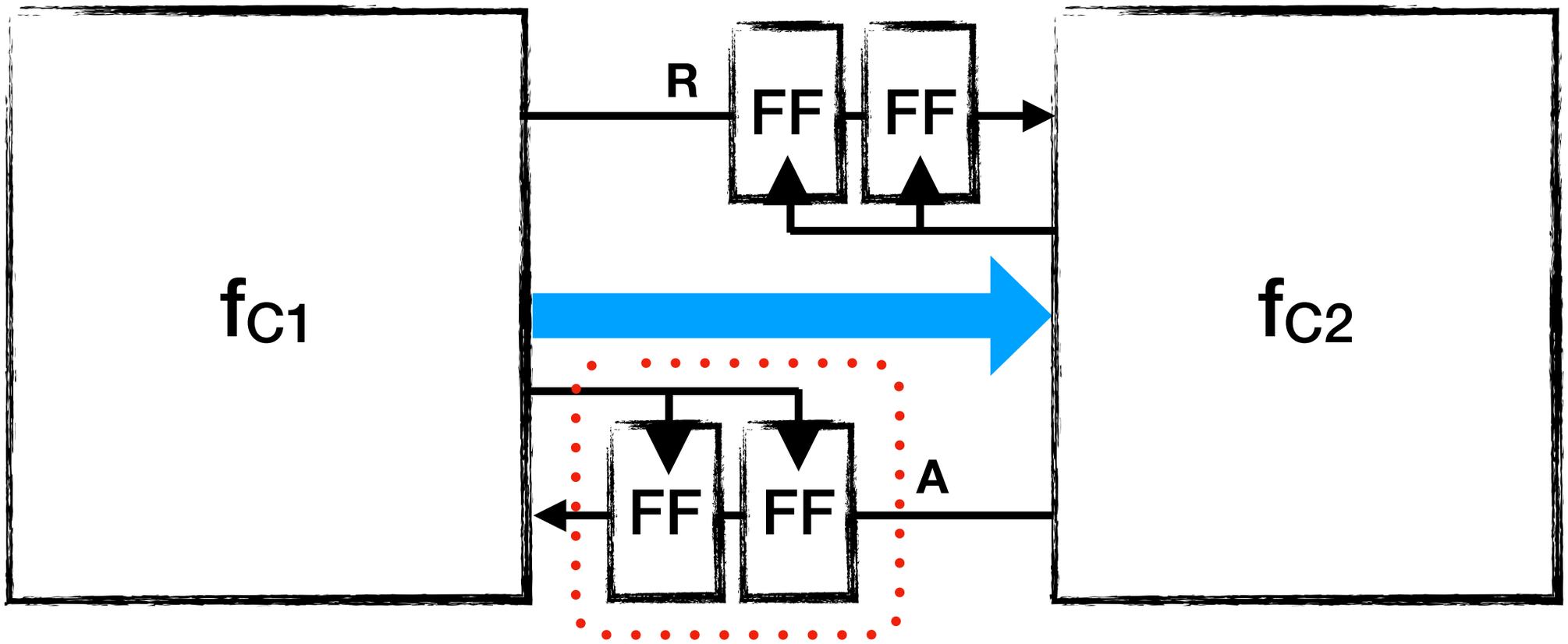


- Standard solution: Use <u>two</u> FFs to receive handshake signals
  - Will survive a T_R of at least one full clock cycle (P(fail) ≈ 0)
- Note: simpler versions possible if only clock <u>phase</u> is different, etc

[Ran Ginosar. Fourteen ways to fool your synchronizer. ASYNC'03]

Synchronizers

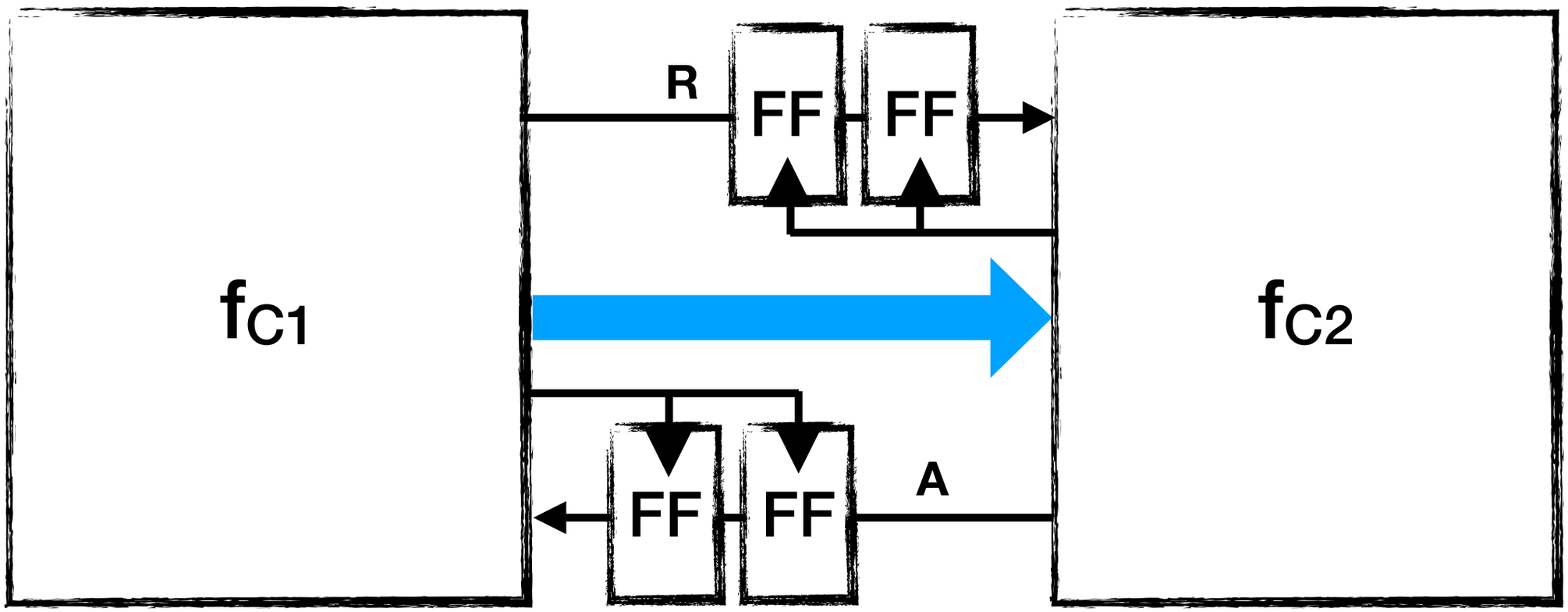
Throughput cost!



- Standard solution: Use <u>two</u> FFs to receive handshake signals
  - Will survive a T_R of at least one full clock cycle (P(fail) ≈ 0)
- Note: simpler versions possible if only clock <u>phase</u> is different, etc

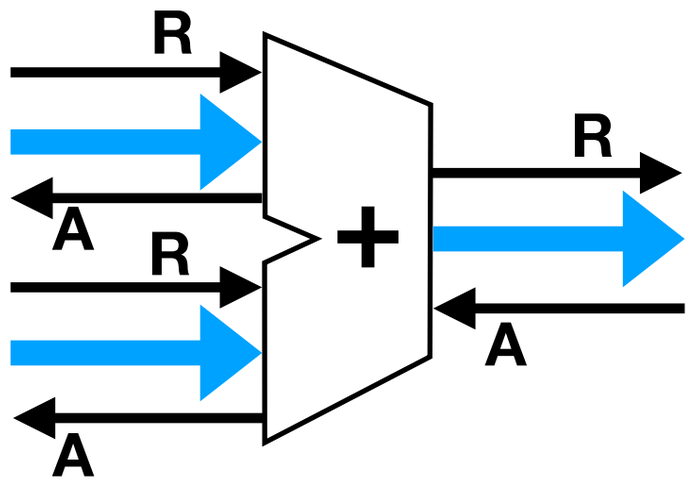
[Ran Ginosar. Fourteen ways to fool your synchronizer. ASYNC'03]

Width



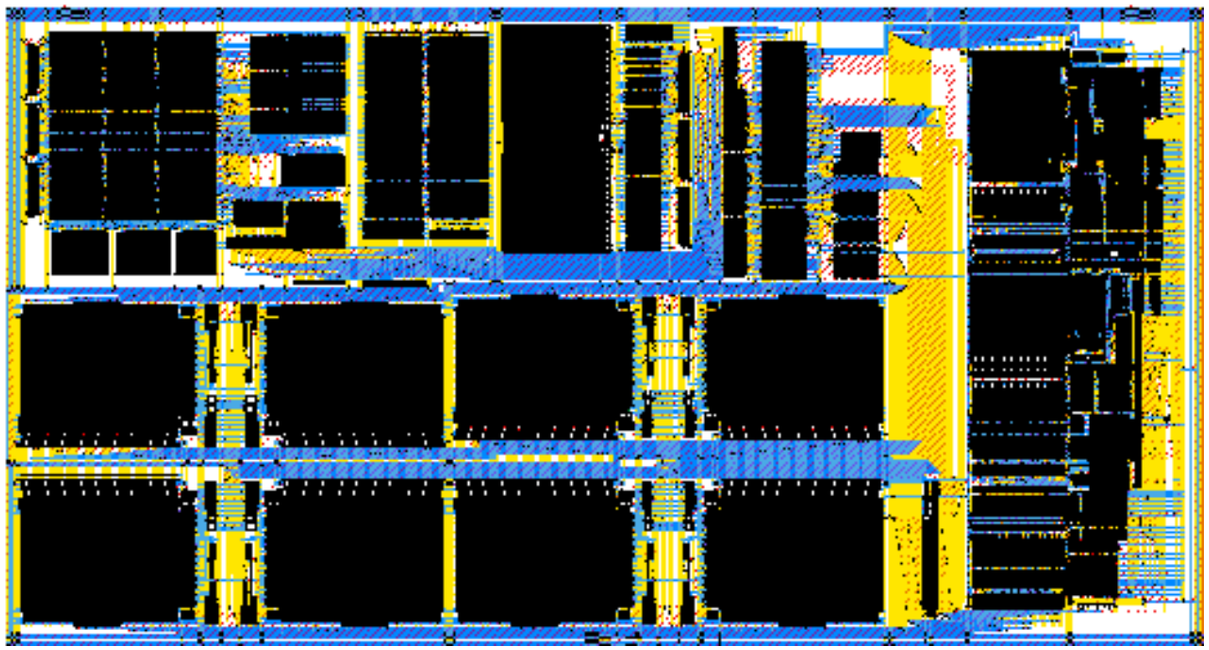
- How wide can that signal bundle be?
  - Practically: propagation time spread across wires must be covered by handshaking cycles
  - Spread inevitable (ref PCB lecture); limits f_C

Asynchronous logic



- Can Ready/Ack pattern be re-used for smaller blocks?
  - An adder could wait for both inputs and then produce output! No need to wait for carry chain when not exercised. Etc…
- Many attempts at large-scale use, limited success / impact
  - Very useful in certain circumstances

Ex: Asynchronous processor

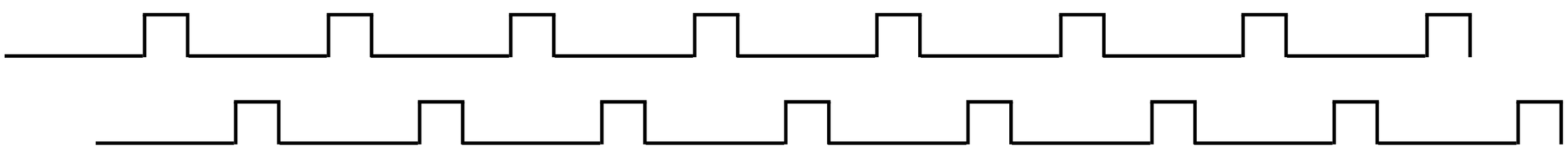


Steve Furber

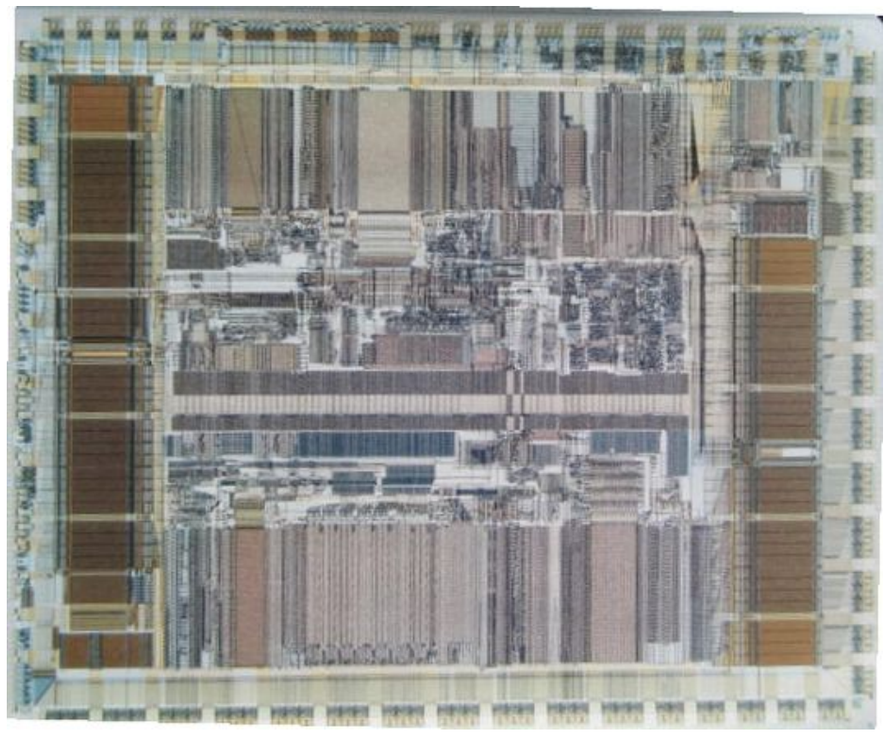
- AMULET (Univ. Manchester, ~2000)
  - Asynchronous implementation of ARM ISA
  - Aimed at low-power, low-emission implementation
  - Needed to develop much of tools to complete chip

HISTORY

- Single-phase clocking (as described here) is a relatively recent practice!
  - Yuan, Svensson: High-speed CMOS Circuit Technique (IEEE JSSC, 1989)
- Before then, mostly two-phase non-overlapping clocks
  - Allowed use of latches rather than FFs to keep data
    - Fewer transistors, minor performance gains, but obsoleted by tools
  - … and the two phases must be kept in sync…



Ex: single phase clock



- Classic example: the first DEC Alpha 21064 processor (1992)
  - 200 MHz, 64b processor
  - Then-novel single-phase clocking
  - One single clock net: 3.25 nF
    - Vdd = 3.3V means 7W clock power (total: 30W)
    - Final driver: W = 350mm 😱

Dan Dobberpuhl

HISTORY

- Clock Gating Considered Harmful!
  - Used decades ago as “design trick” to save a few logic gates here and there
  - Bug-prone, very difficult to test, savings not worth it
    - Old books may still condemn the practice
- Now, used to save power rather than gates
  - Well-supported by tools
  - No more unsafe than other design practices

Summary

- Clock signals used in almost all digital designs to orchestrate logic operations
- Issues in ASICs and FPGAs stem from same overall considerations (setup/hold criteria/violations)
- Specialized tools help with clock tree balancing, clock gating, etc
- Take care when crossing clock domain borders
- Beware the siren call of asynchronous design 😀