Synchronizers, Arbiters, GALS and Metastability

David Kinniment
University of Newcastle, UK

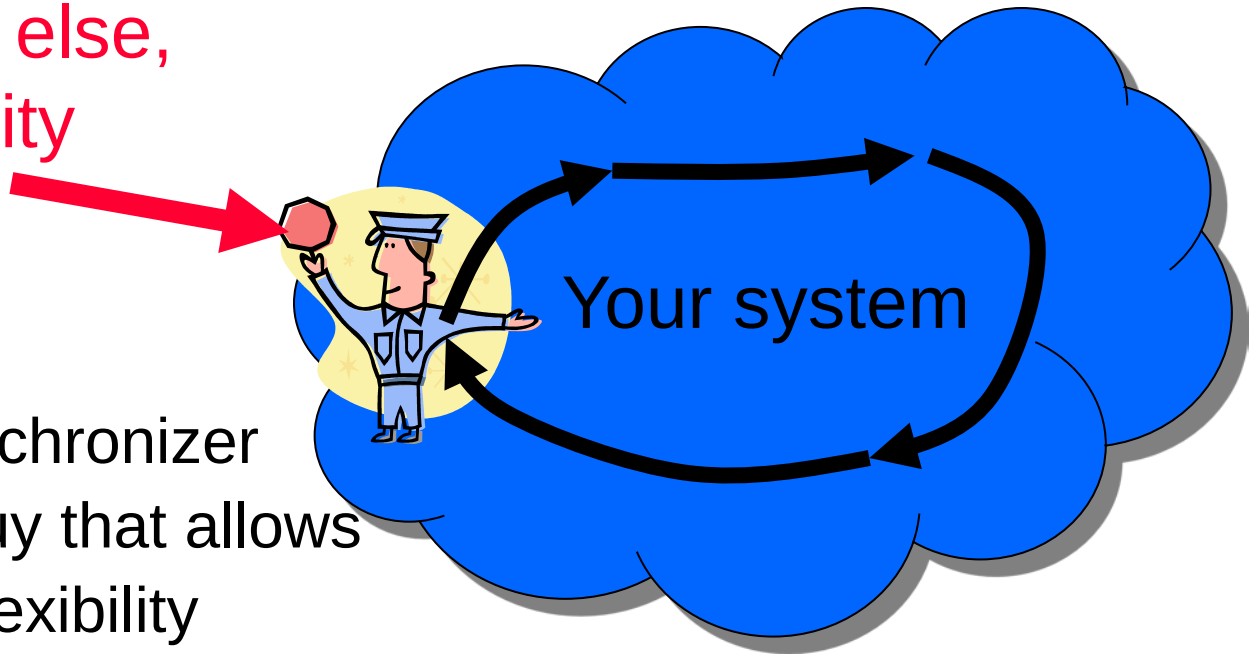
Based on contributions from:
Alex Bystrov, Keith Heron, Nikolaos Minas,
Gordon Russell, Alex Yakovlev, and Jun Zhou

Outline

- What's the problem? Why does it matter?
- Time and distributions
- Noise – decisions are not deterministic.
- Synchronizer latency, can it be avoided?
- Measuring and some interesting effects

What's the problem: The digital IP world and the rest of the world

Everything else,
or Reality

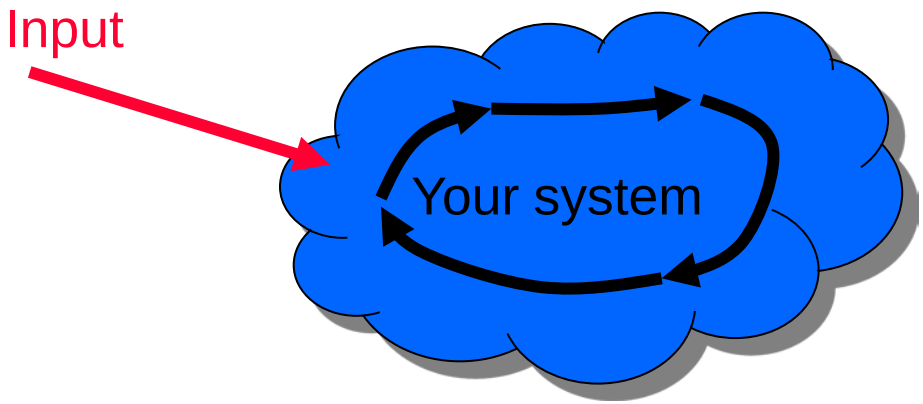


The synchronizer
is the guy that allows
timing flexibility

Synchronizers and arbiters

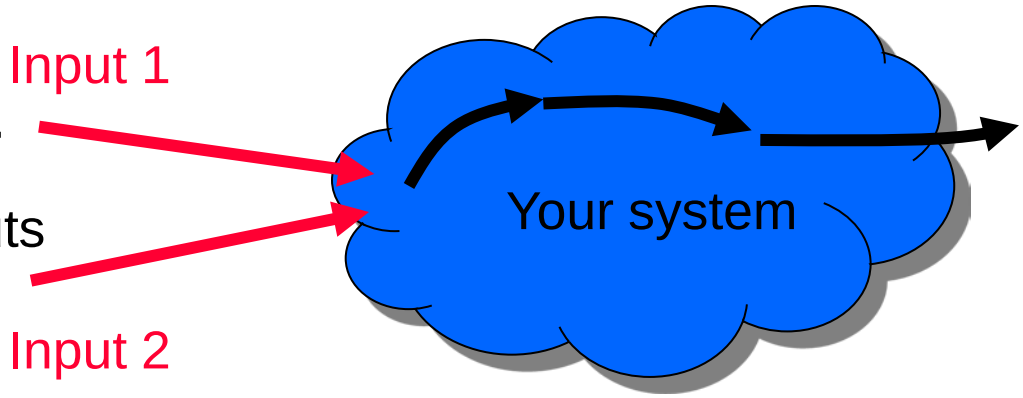
- Synchronizer

  Decides which clock cycle to use for the input data



- Asynchronous arbiter

  Decides the order of inputs

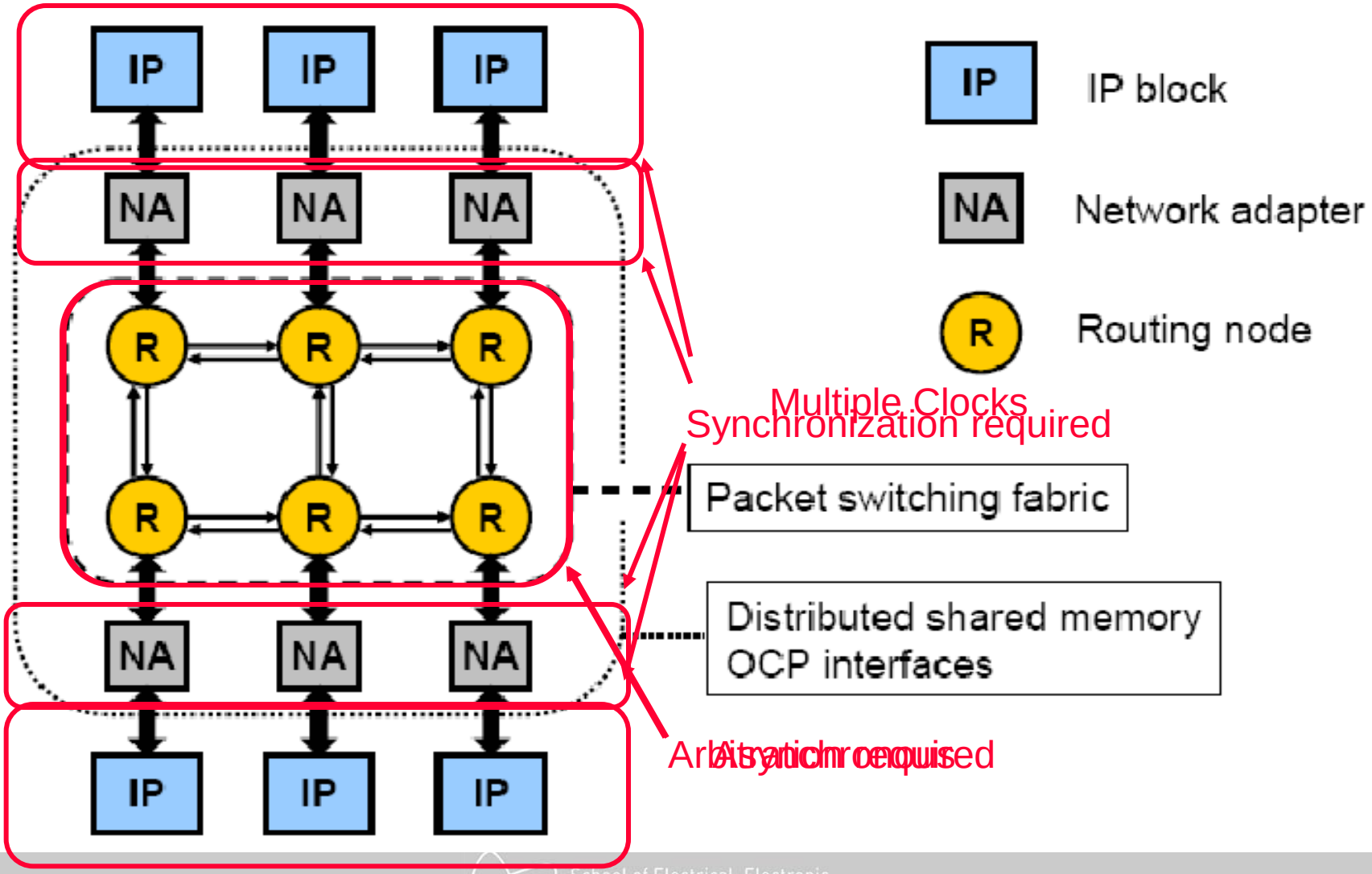


Time Comparison Hardware

- Digital comparison hardware (which compares integers) is easy
  - Fast
  - Bounded time
- Analog comparison hardware (which compares reals like time) is hard
  - Normally fast, but takes longer as the difference becomes smaller
  - Can take forever, (Buridan's Ass ~1340)
- Synchronization and arbitration involve comparison of time

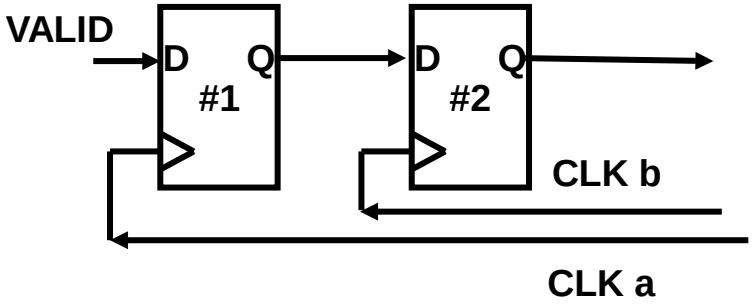
- Known to early computer designers:
  - Lubkin 1952, Catt 1966
  - Chaney and Littlefield 1966/72

Asynchronous Network (Sparsø, ASYNC 2005)



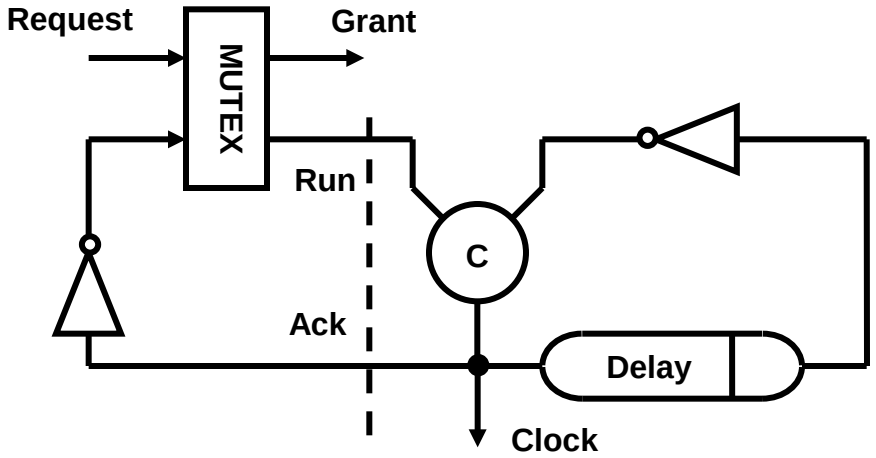
Synchronization of data

- Non pausible clocks require data synchronization
- You have a limited time to synchronize.
- Synchronizer circuits may fail to work in that time
- System sometimes fails (you fly into a mountain)
- Synchronization time = latency



Synchronization of clocks

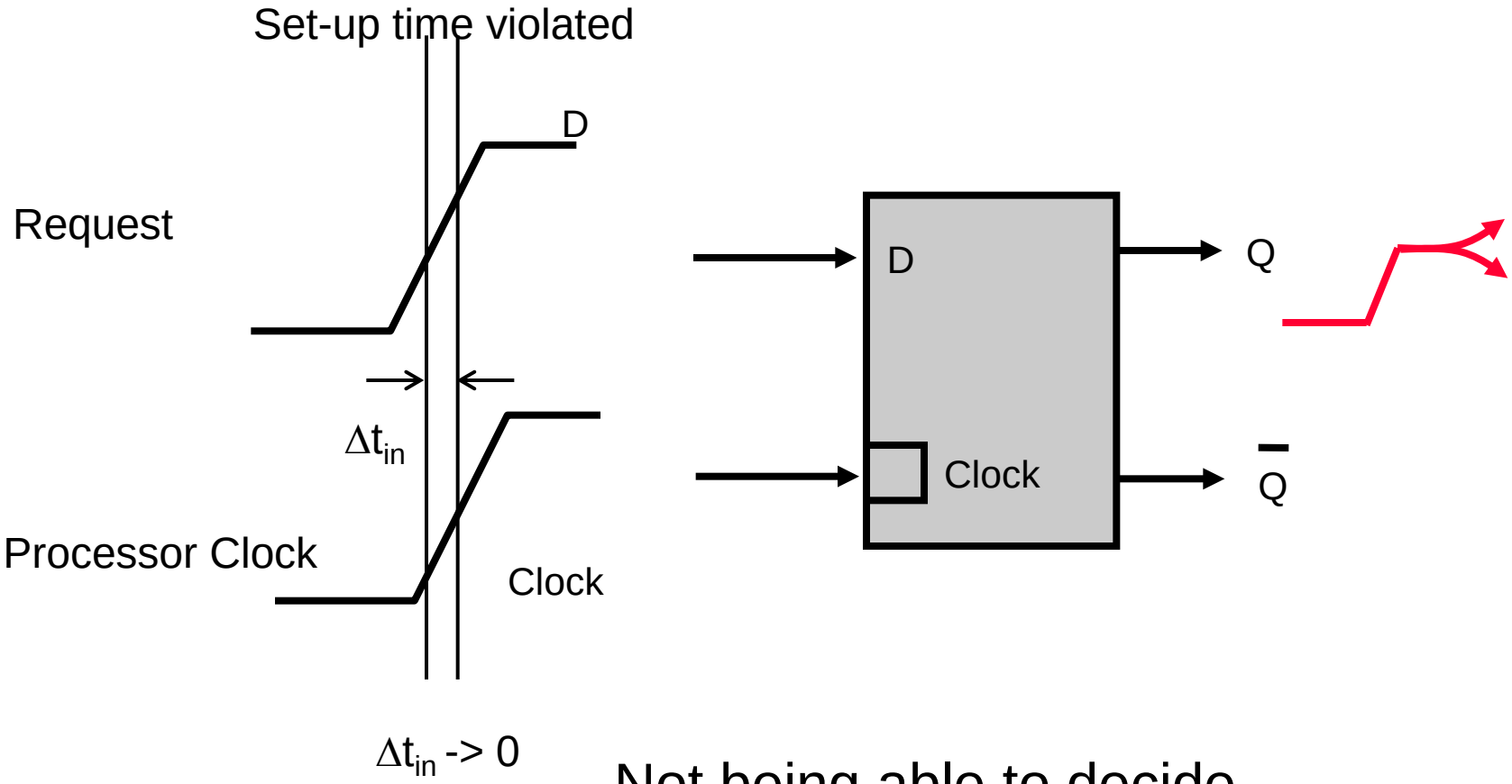
- Clock paused to prevent contention
- Wait for MUTEX output to resolve
- Can take forever (with decreasing probability)
- This may not be acceptable (you fly into a mountain)



Trends

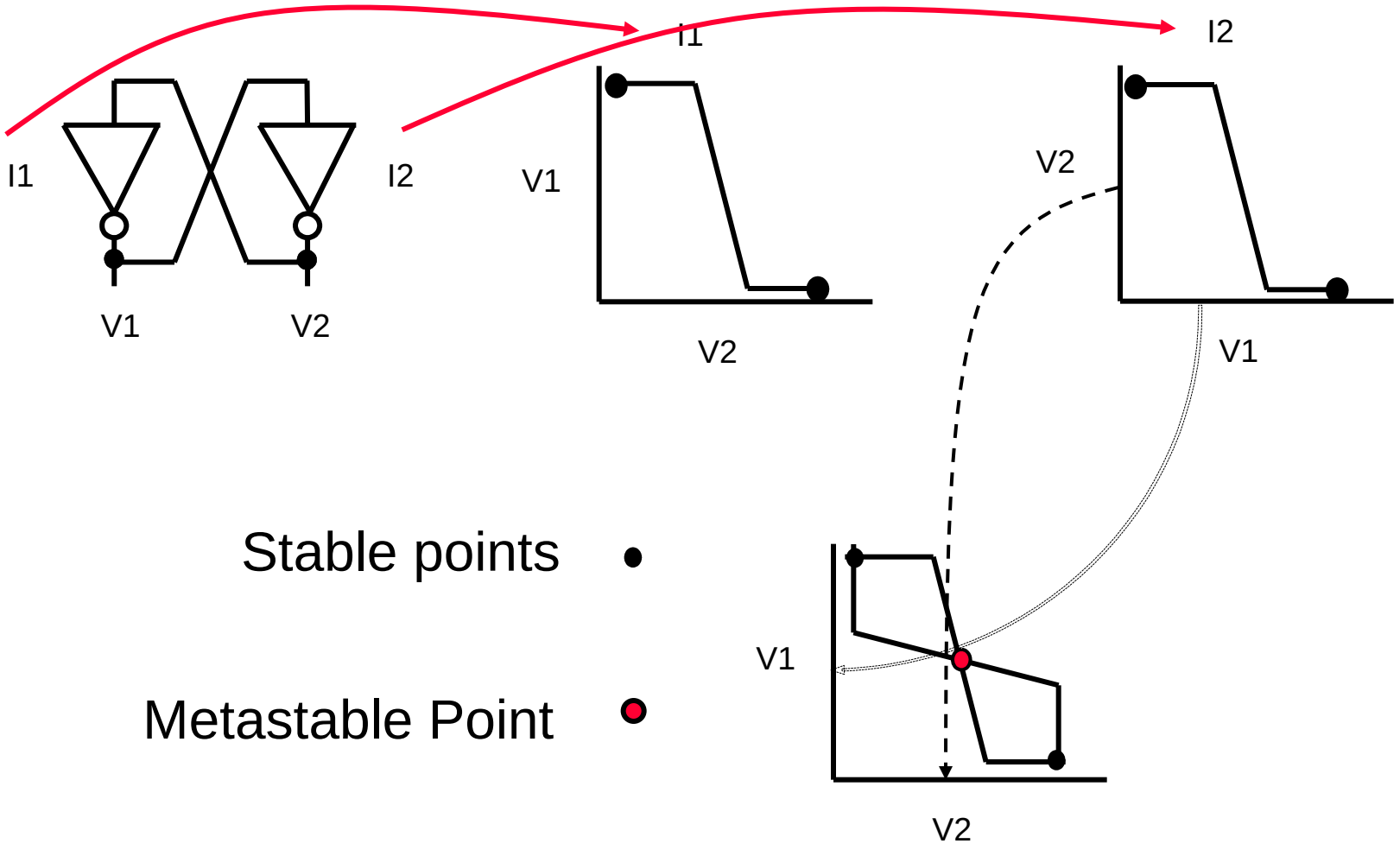
- Communications are limiting performance
- Difficulty (Impossibility?) in maintaining one clock per chip
- Need reduce wasted power in clocks
- Timing closure problems
  - Interconnect delay times long
  - Variability increases
- Move towards asynchronous networks

Metastability is....



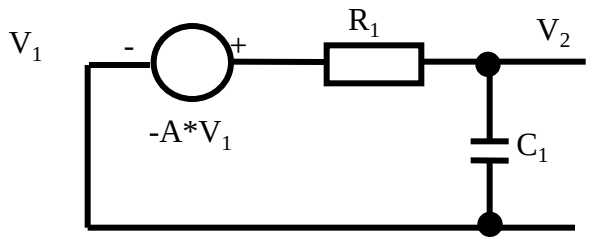
Not being able to decide…

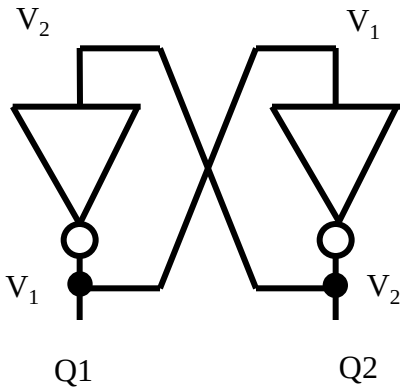
Metastability in a Latch



Simple Linear Model

- Simple linear model leads to two exponentials
- τ_a is convergent, τ_b is divergent





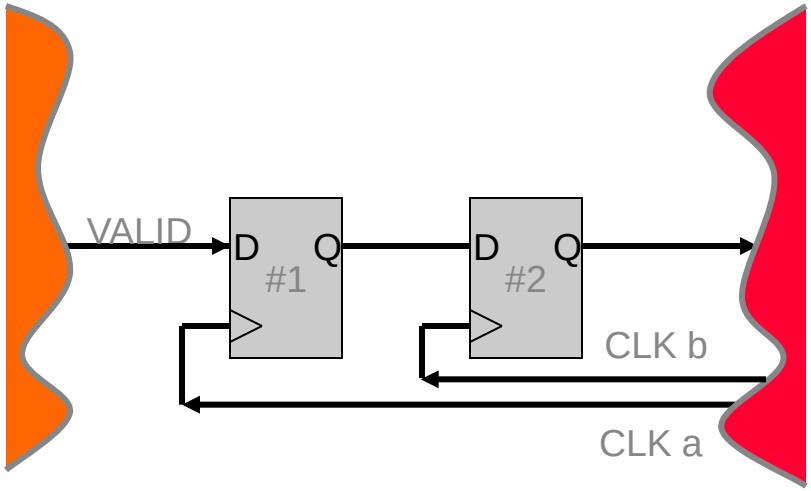
$$\tau_1 = \frac{C_1 . R_1}{A} \, , \tau_2 = \frac{C_2 . R_2}{A} \qquad g_m = \frac{A}{R}$$

$$0 = \tau_1 . \tau_2 . \frac{d^2 V_1}{dt^2} + \frac{(\tau_1 + \tau_2)}{A} . \frac{dV_1}{dt} + (\frac{1}{A^2} - 1).V_1$$

$$V_1 = K_a . e^{\frac{-t}{\tau_a}} + K_b . e^{\frac{t}{\tau_b}}$$

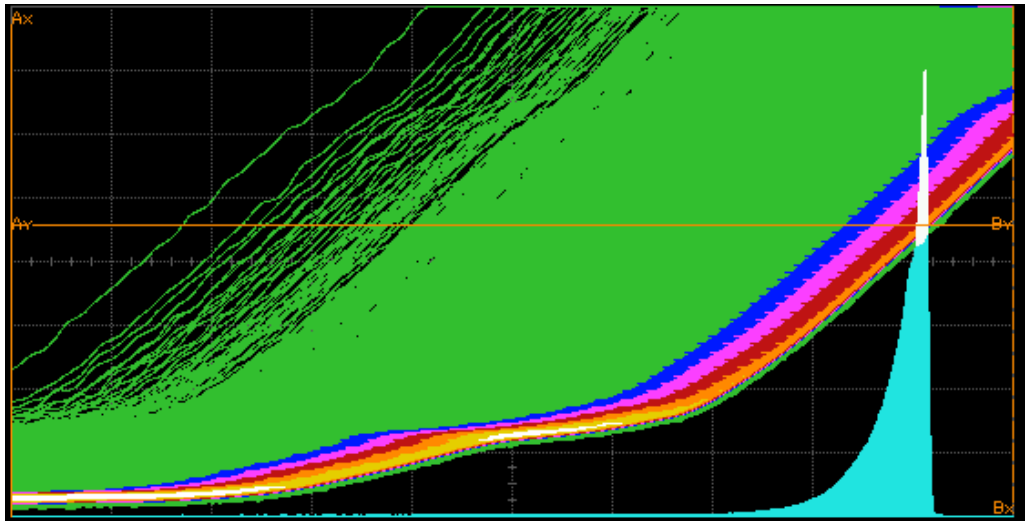
Synchronizer

- *t* is time allowed for the Q to change between CLK a and CLK b
- τ is the recovery time constant, usually the gain-bandwidth of the circuit
- T_w is the "metastability window"
- τ and T_w depend on the circuit
- We assume that all values of Δt_{in} are equally probable



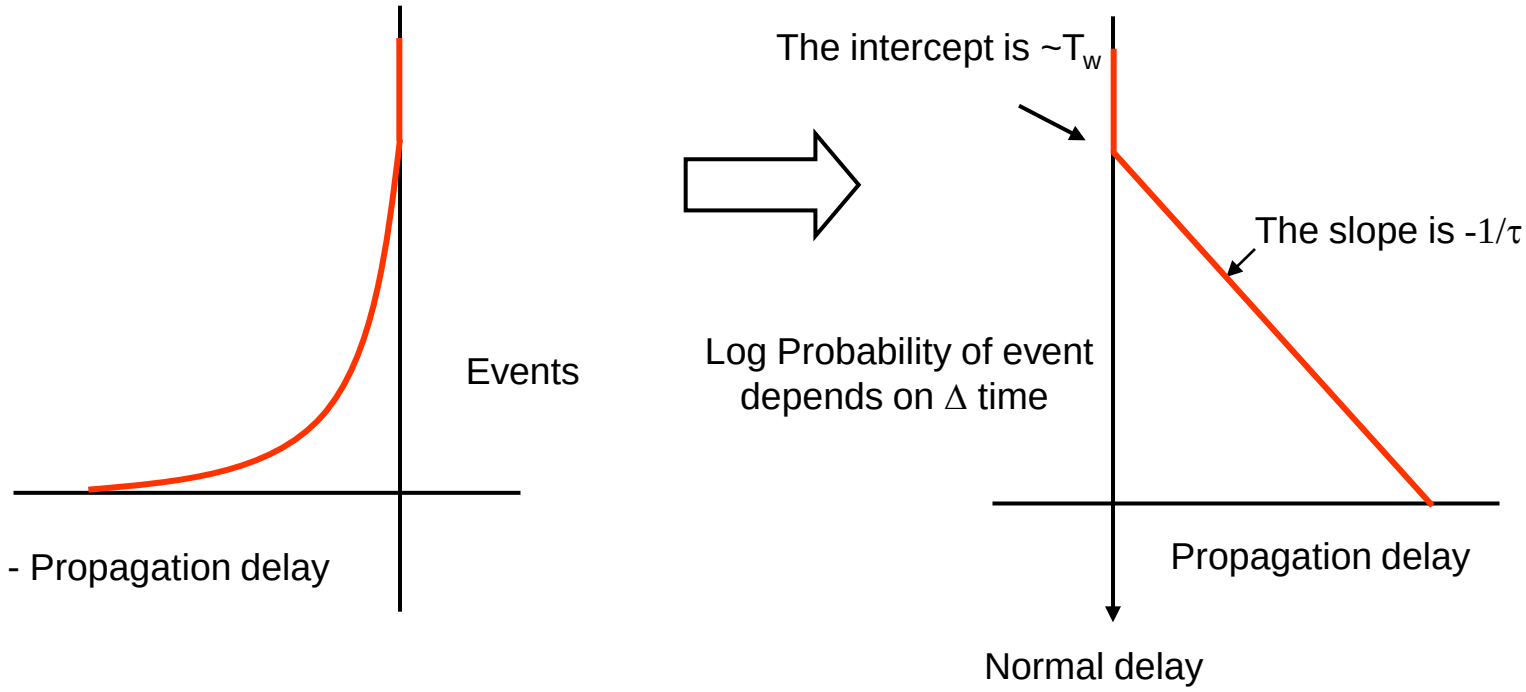
$$MTBF = \frac{e^{t/\tau}}{T_w \cdot f_c \cdot f_d}$$

Typical responses



- All starting points are equally probable
- Most are a long way from the “balance point”
- A few are very close and take a long time to resolve

Event Histogram



Synchronizer state of the art

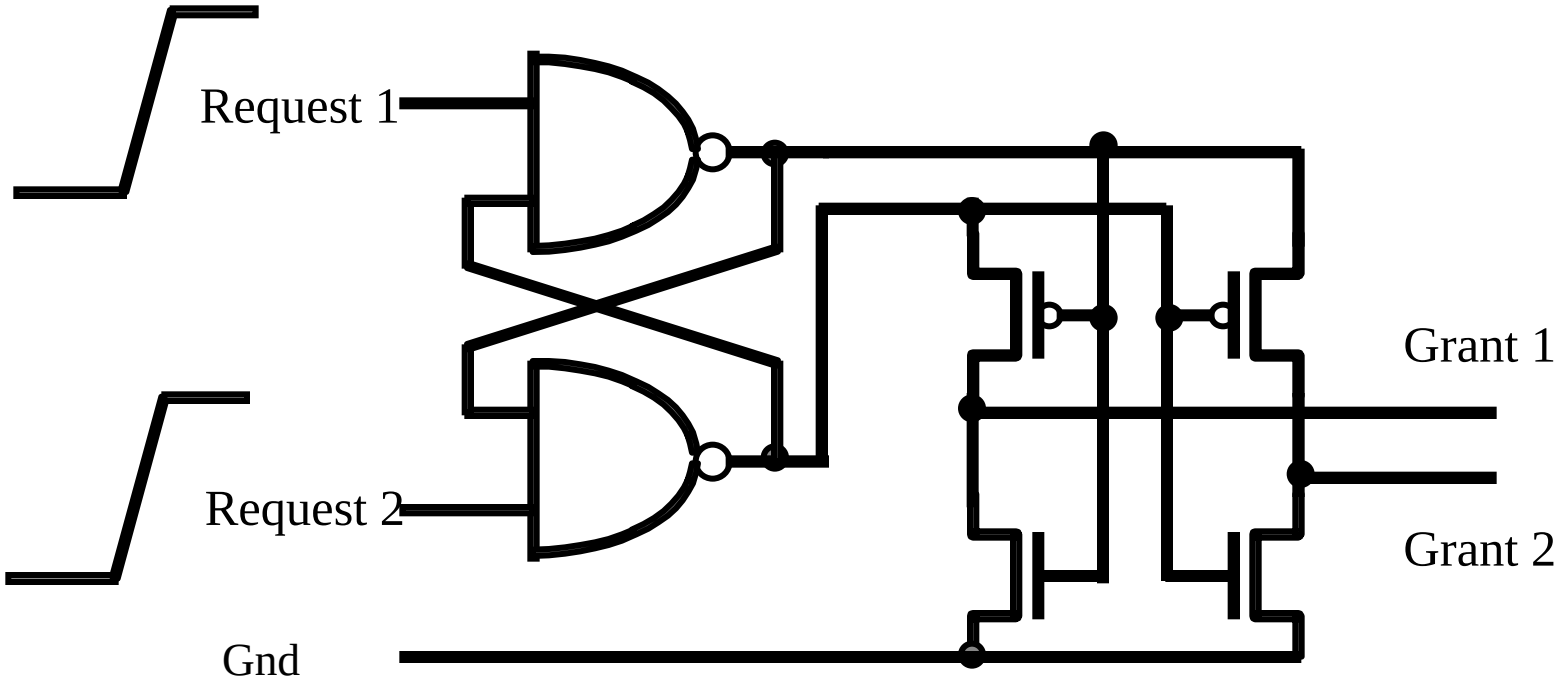
- You require about 35 τ s in order to get the MTBF out to about 1 century. (That's for 1 synchronizer)
- There is nothing else you can do while synchronizing
- Each typical static gate delay is equivalent to about 5 τ Synchronizers are analog devices, so worse affected by scaling
- Bigger SoCs, in future systems so more synchronizers, worse reliability
- Inputs can be 'malicious' i.e. always causing metastability.

Outline

- What's the problem? Why does it matter?
- Time and distributions
- Noise – decisions are not deterministic.
- Synchronizer latency, can it be avoided?
- Measuring and some interesting effects

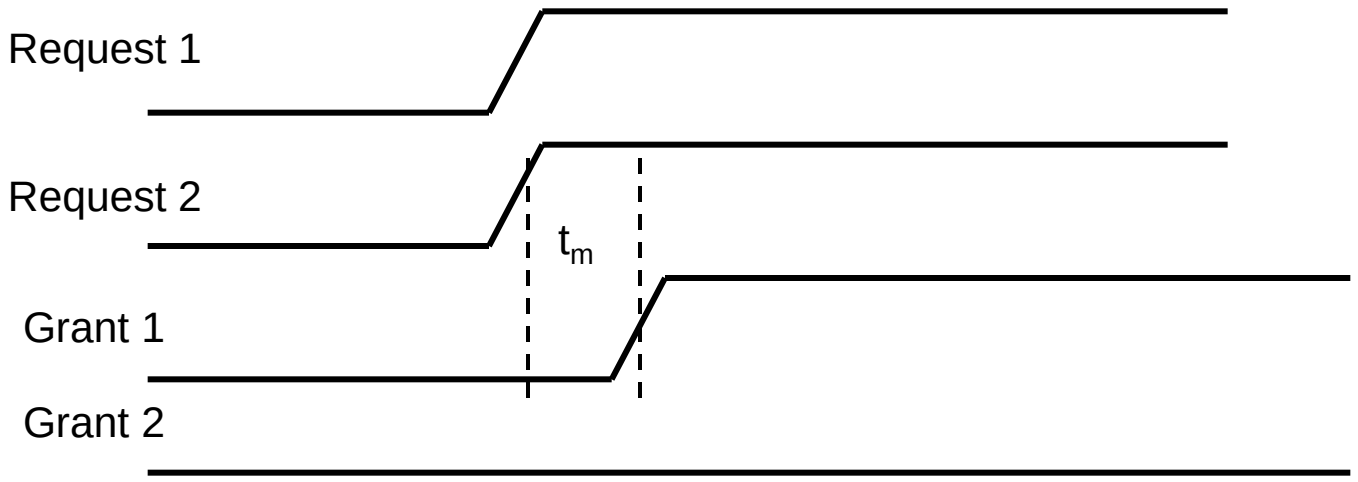
The arbiter (MUTEX)

- Asynchronous arbitration,No time bound
- Seitz metastability filter
- Grants cannot occur until after the latch resolves any metastability



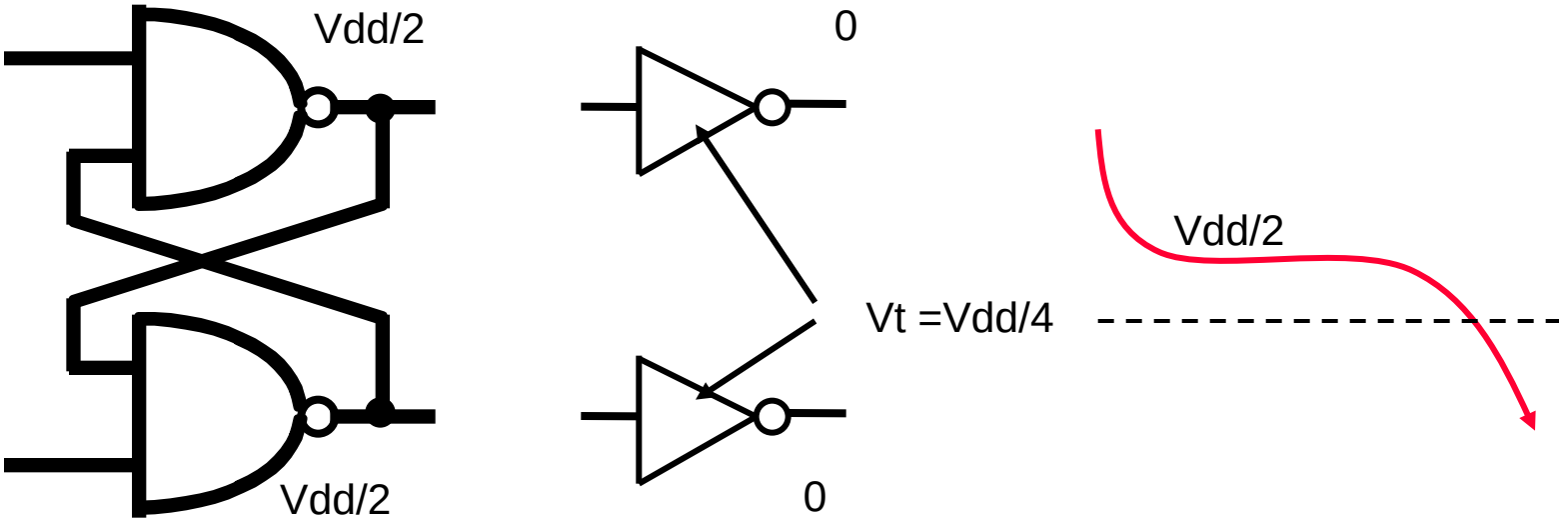
Arbitration time

- Unlike a synchronizer, an arbiter may take for ever.
- It usually doesn't, long responses are rare.
- On average the time should only be τ longer than the normal response, so DEAD time ought to be low
- Outputs are always monotonic

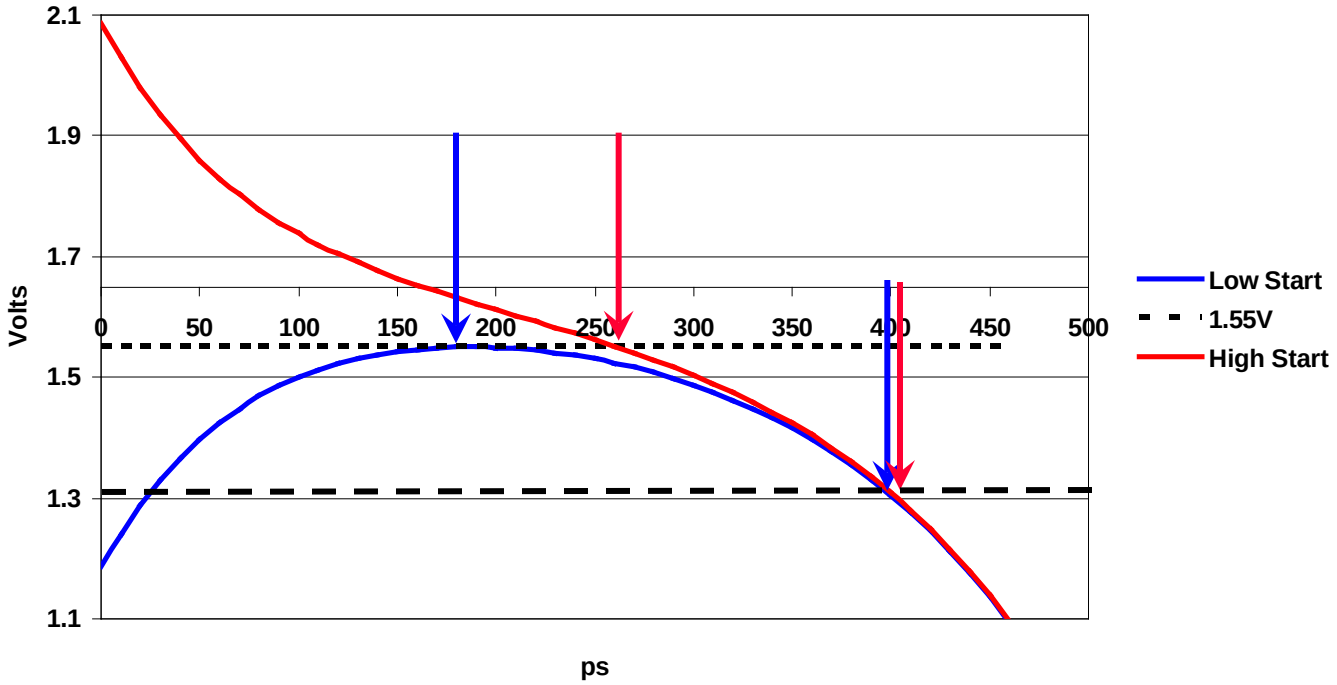


Gate metastability filter

- Half levels due to metastability need to be removed
  - Low (or high) threshold inverters
  - Measure divergence
- Filters define the time to reach a stable state



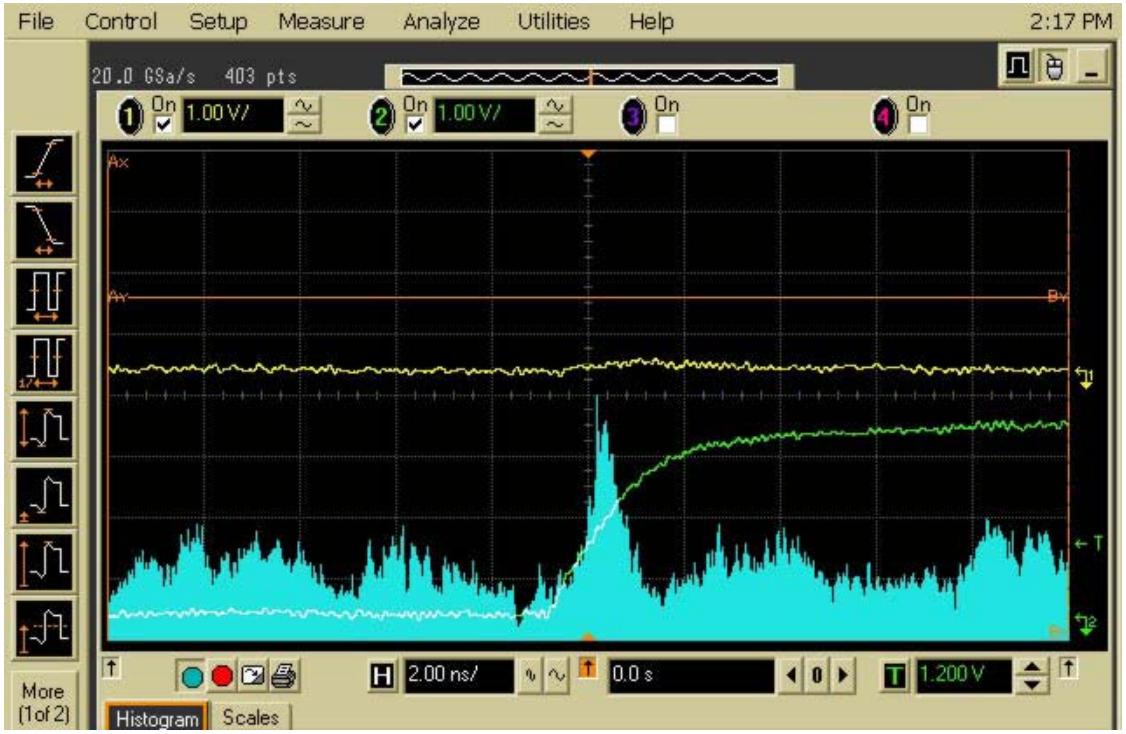
Resolution time can be affected



- The start/finish points make a difference
- Grant appears when trajectory crosses the filter threshold
- Gate metastability filter shows more variation
- Seitz filter shows more delay

Event distribution

- Assumption: Time between two events (R1 and R2) is evenly distributed
- Typically it is not.
- Can be as much as 7:1 variation.
- Here there are more cases where R1 is just after R2
- Arbitration may not be fair



Distribution of event times

Non uniform probabilities and time

- Time for metastability averaged over a distribution *T* in a MUTEX is:

$$Average\_time = \tau \int_0^T \ln\left[\frac{T_w}{\Delta t_{in}}\right].d\Delta t_{in}$$

  - If $T = \infty$, $Average\_time = \tau$

  BUT

  For a malicious input, distribution limited by noise

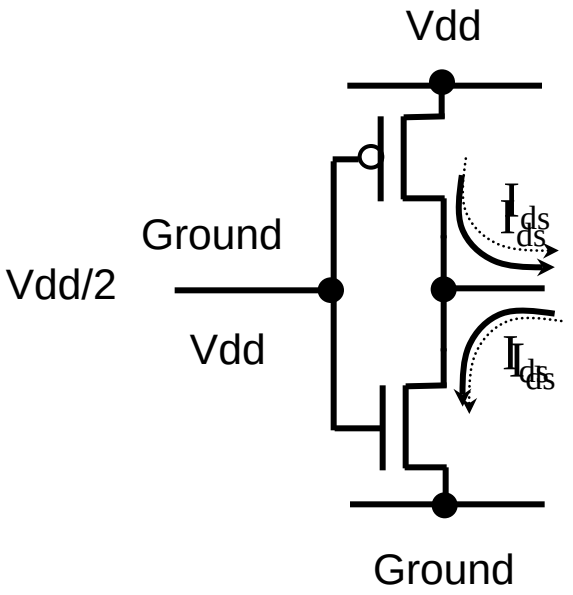
  - If $T = 4ps$, $T_w = 25ps$, $Average\_time = 4\tau$
  - Noise can sometimes make average time faster

Predicting performance

- MUTEX circuits are fast
- BUT delay times may not be easy to predict
  - Cannot rely on ALL times being fast
- Synchronizers are known to be unreliable

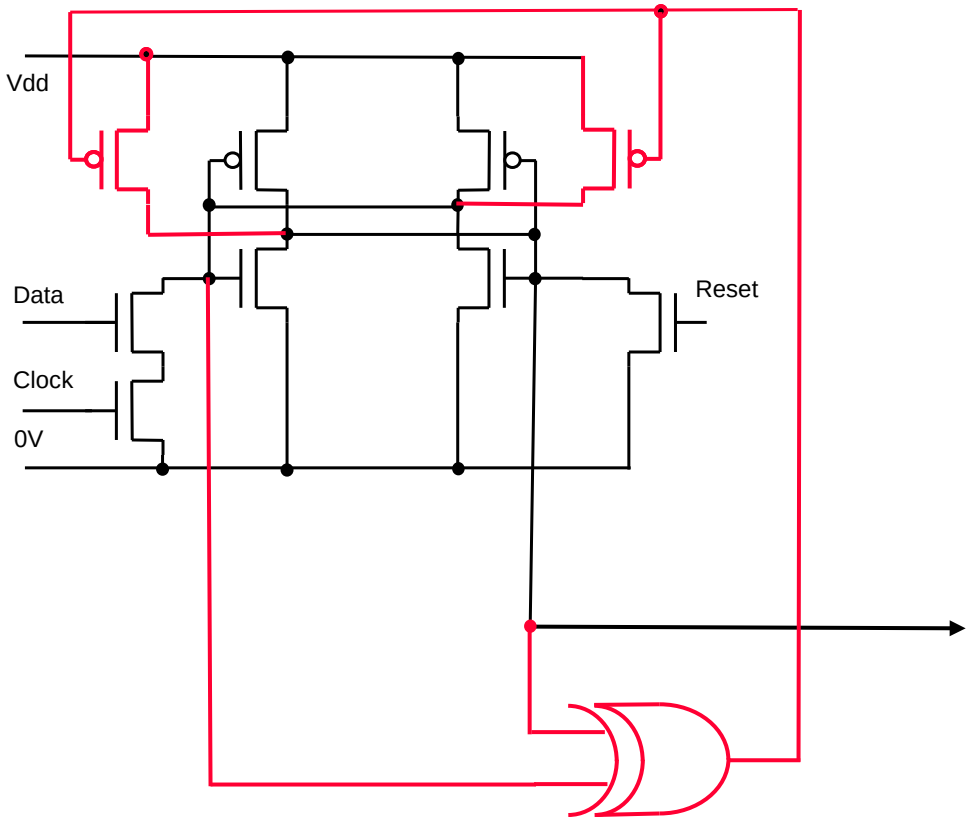
Future processes

- Synchronizers and arbiters don't work well in nanometre technologies, especially at low Vdd
- Worse that gates! Why?
- A gate input is either HIGH
  - Output pulled down
- Or LOW
  - Output pulled up
- A metastable gate is neither
  - Both transistors can be off
- Vdd decreases with process shrink,
  - g_m very low
- Synchronization time constant $\tau = C/g_m$



Robust synchronizer

- Jamb latch synchronizer slow for low V_{dd}, low temp
- In metastability, both outputs are the same
- Extra p-types are switched fully on, so g_m increases, and τ improves



Results

	Measurement Results (ps)	
Vdd(v)	**Jamb Latch B**	**Robust Synchronizer**
1.8	35.55	34.92
1.7	37.29	35.76
1.6	40.93	38.25
1.5	52.36	43.07
1.4	66.17	50.36
1.35	75.35	58.19

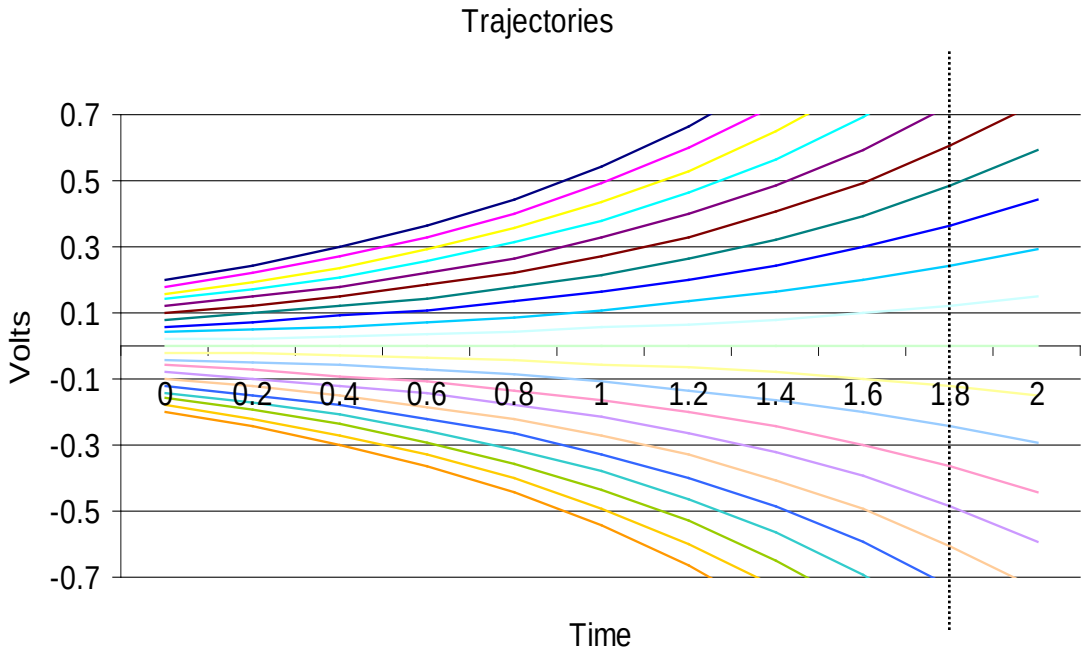
τ (metastability time constant) vs Vdd

Outline

- What's the problem? Why does it matter?
- Time and distributions
- Noise – decisions are not deterministic.
- Synchronizer latency, can it be avoided
- Measuring and some interesting effects

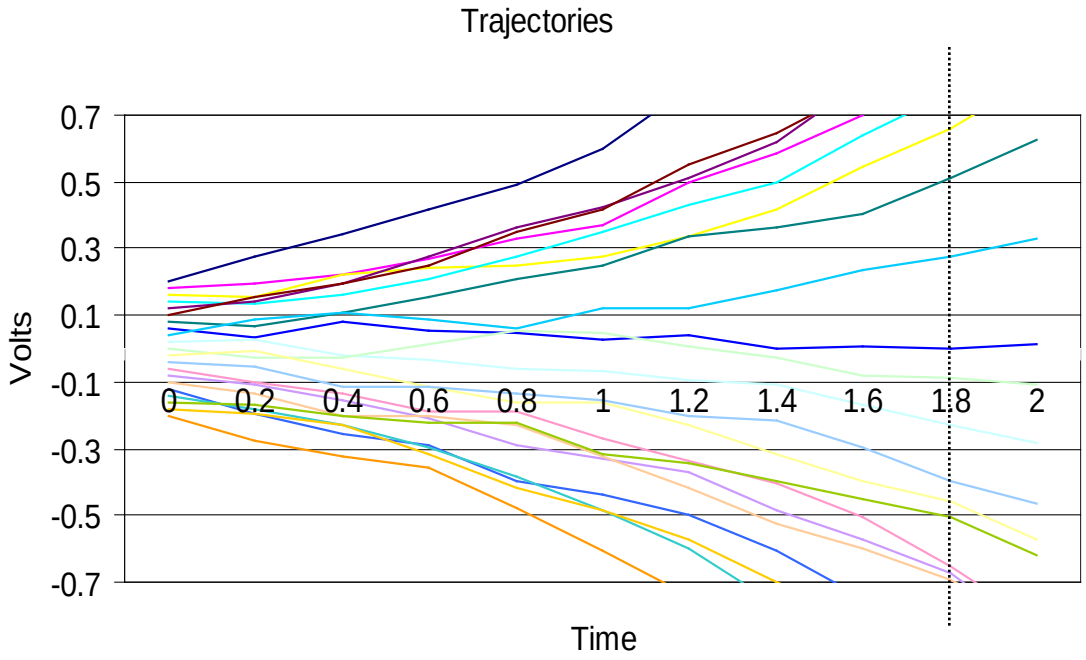
Does noise affect τ?

- Probability of escape from metastability does not change with gaussian noise (Couranz and Wann 1975)

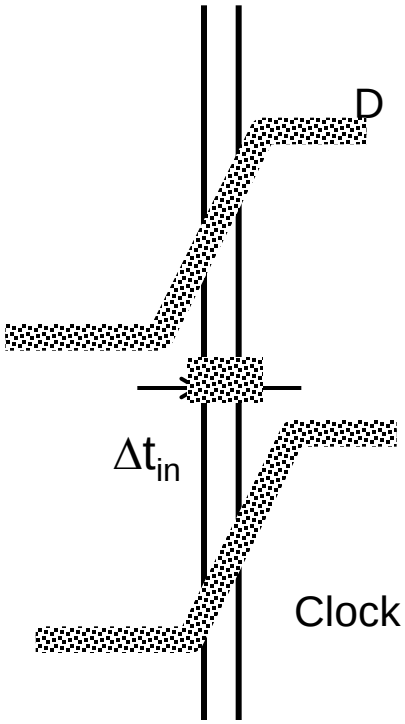


Does noise affect τ?

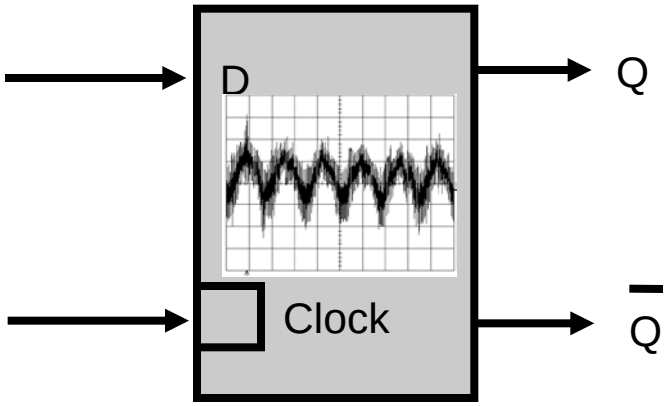
- Probability of escape from metastability does not change with gaussian noise (Couranz and Wann 1975)



Can noise change the failure rate?



Δt_{in} -> 0



Or maybe not.….

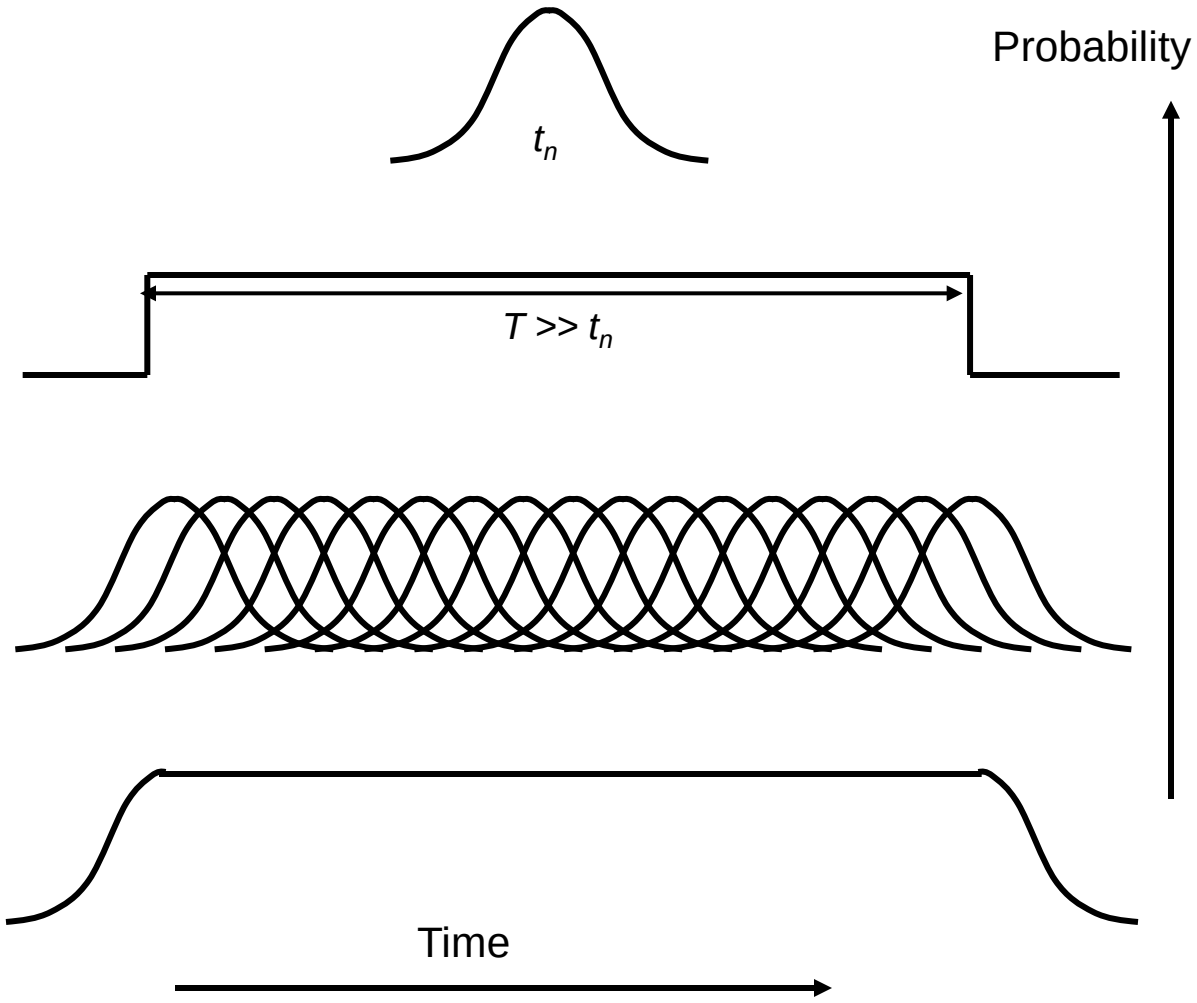
The normal case

Probability of initial difference due to noise component $P_1(v)$

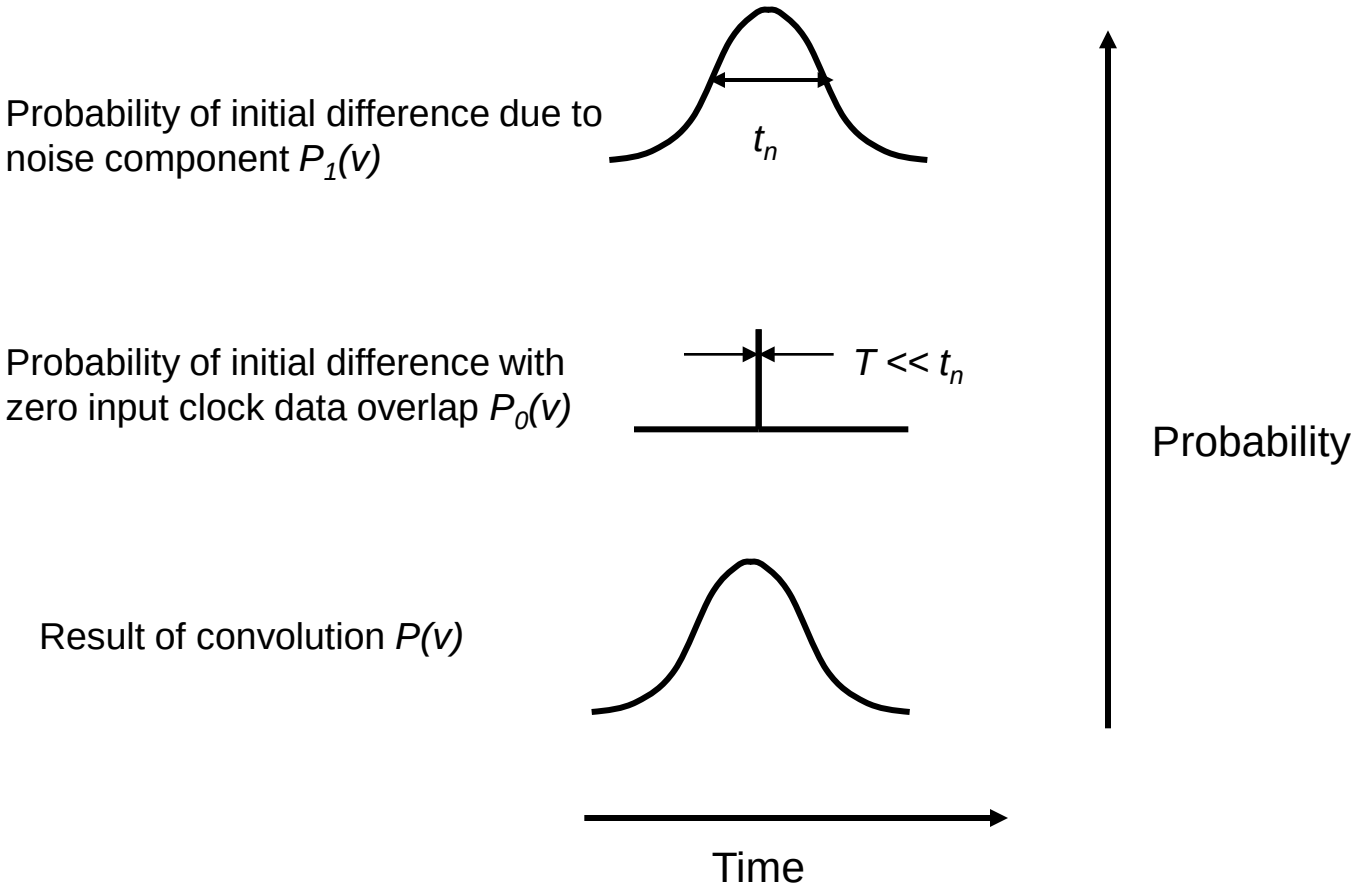
Probability of initial difference due to input clock data overlap $P_0(v)$

Convolution

Result of convolution $P(v)$



The malicious input



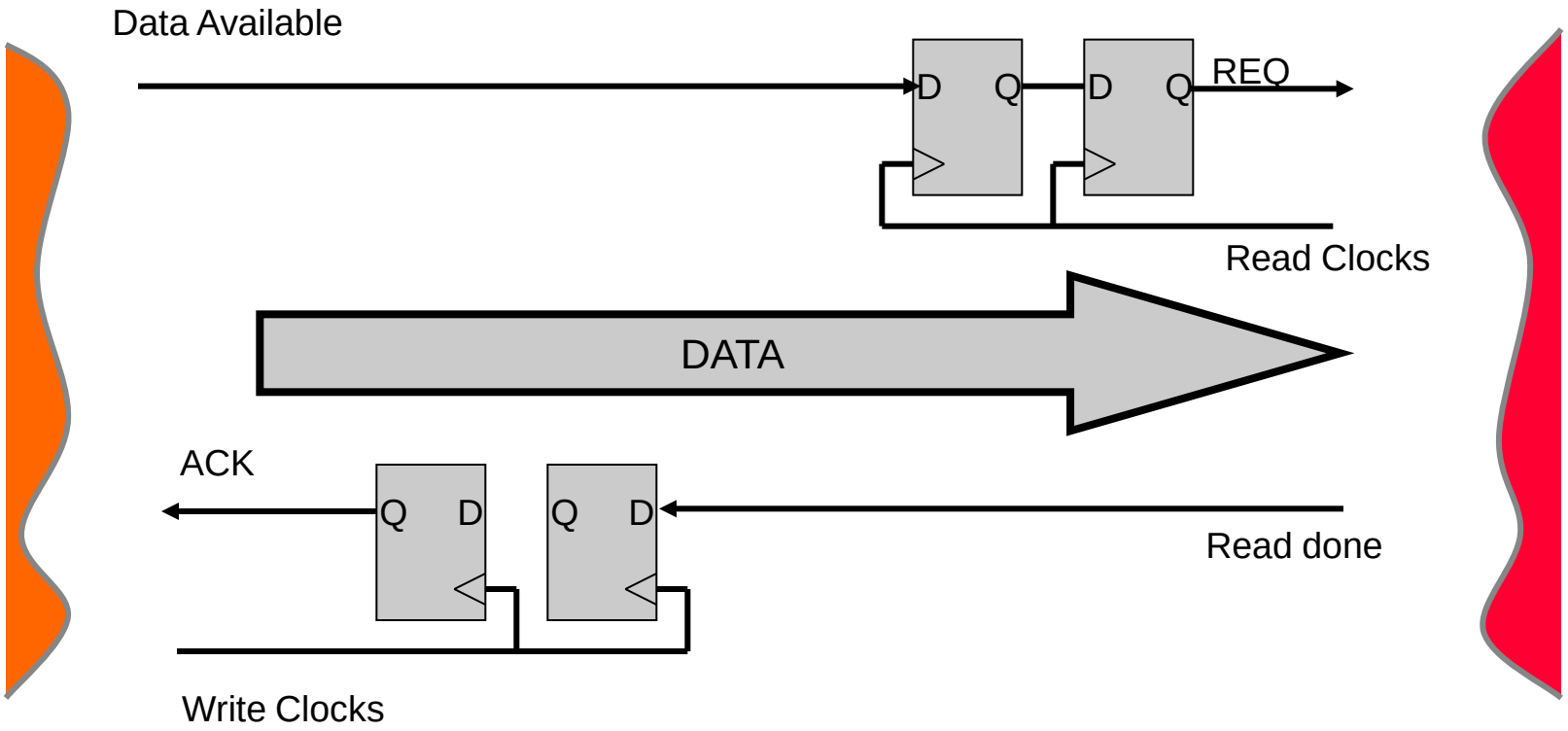
Non-determinism

- Synchronizers and arbiters can produce non-deterministic outcomes
- Some noise is deterministic (5-50ps)
  - Power supply, crosstalk
- Some noise is non-deterministic (typically 0.1ps - 0.5ps).
  - Thermal noise, which increases as dimensions reduce.
- Sequence and time of an individual computation paths is unpredictable
- System performance can only be predicted probabilistically

Outline

- What's the problem? Why does it matter?
- Time and distributions
- Noise – decisions are not deterministic.
- Synchronizer latency, can it be avoided?
- Measuring and some interesting effects

Request and Acknowledge



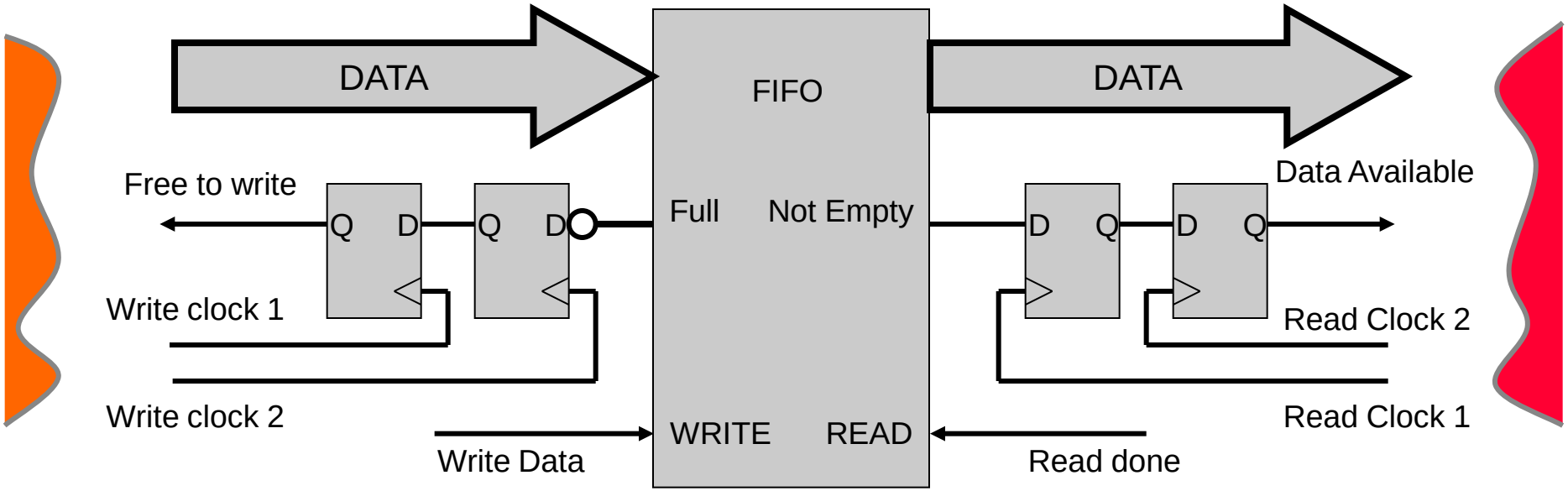
Latency

- It takes one - two receive clocks to synchronise the request
- Then one – two write clocks to acknowledge it

- Significant latency (1-3 clocks)
- Poor data rate (2 – 6 Clocks)

FIFO

- Can improve data rate by using a FIFO
- But not latency (which gets worse)
- FIFO is asynchronous (usually RAM + read and write pointers)

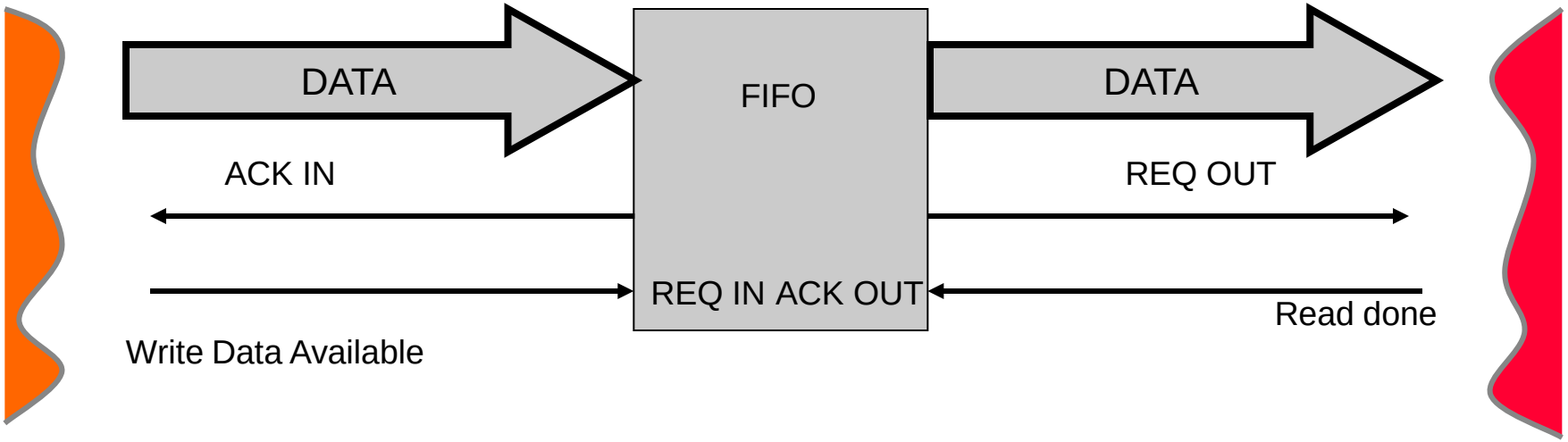


Timing regions can have predictable relationships

- Phase locked (*mesochronous*)
  - Timing of input data is constant, therefore predictable
- Same or related frequencies but phase difference can drift in an unbounded manner. (*plesiochronous)*
  - Timing is not constant but is still predictable
- Unrelated frequencies (*Heterochronous*)
  - No assumptions about timing can be made
  - Need a synchronizer

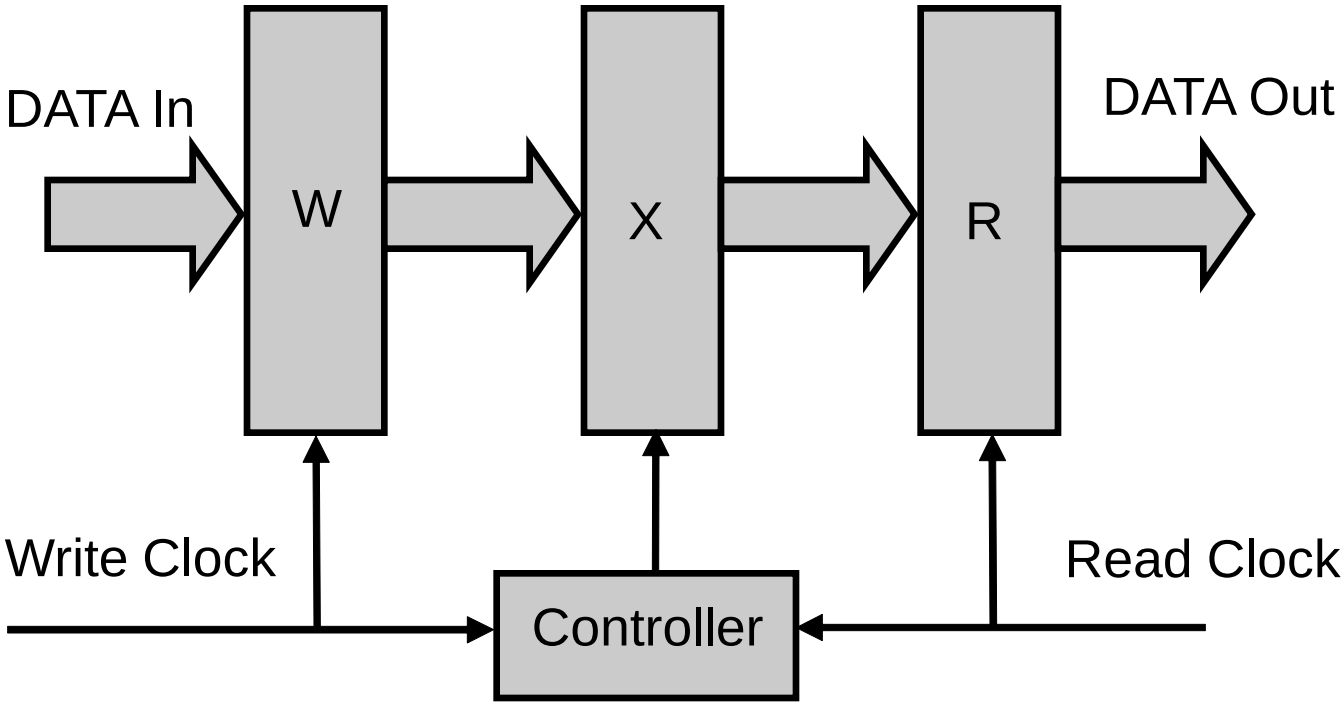
Don’t synchronise when you don’t need to

- If the two clocks are locked together, you don’t need a synchroniser, just an asynchronous FIFO big enough to accommodate any jitter/skew
- FIFO must never overflow/underflow, so there is latency

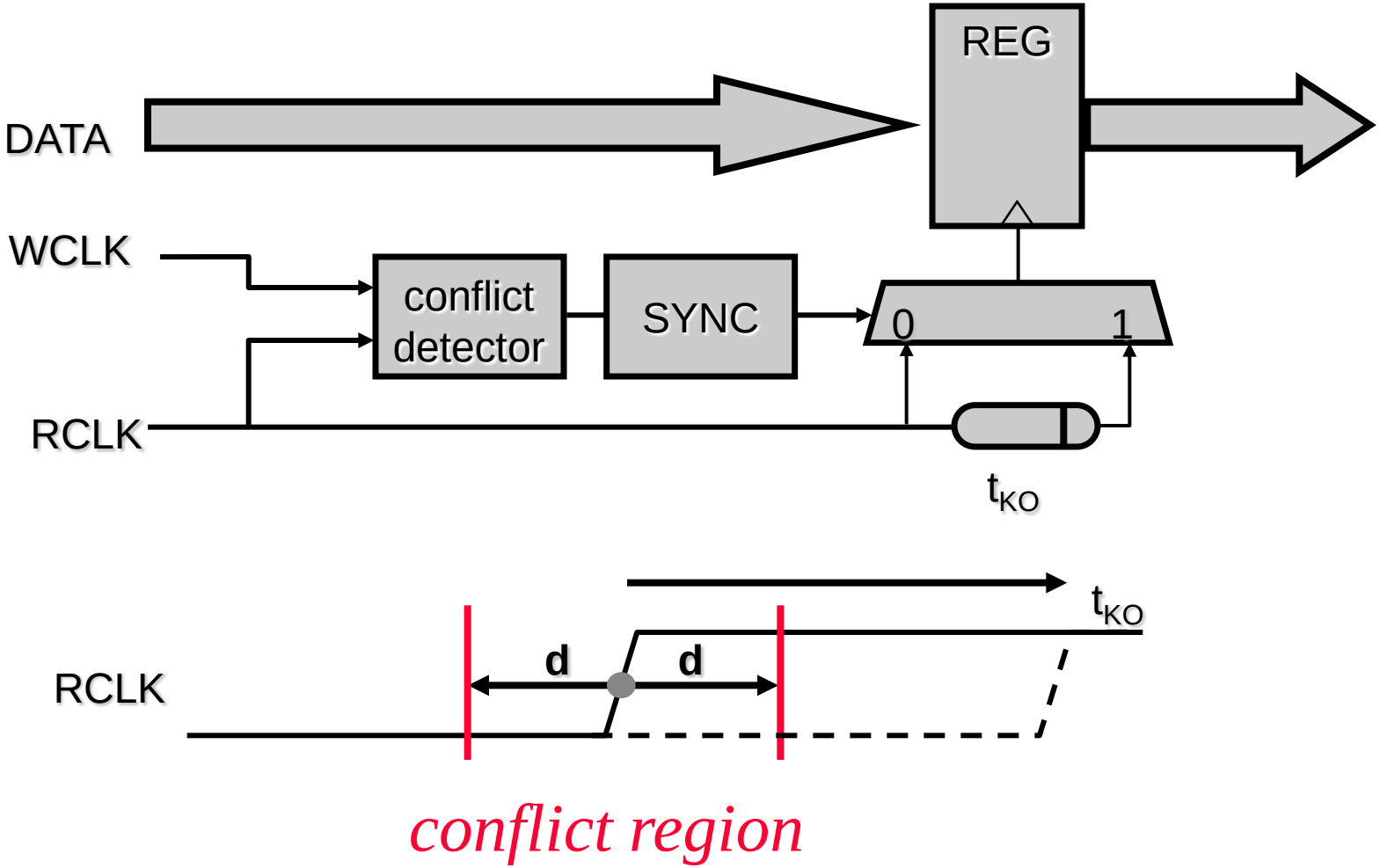


Mesochronous data exchange

- Intermediate X register used to retime data
- Need to find a place where write data is stable, and read register available. There is always a place which can be found at start up
  - Chakraborty and Greenstreet ASYNC 2003



Clock delay synchronizer (Ginosar AINT 2000)



Delay synchronizer latency

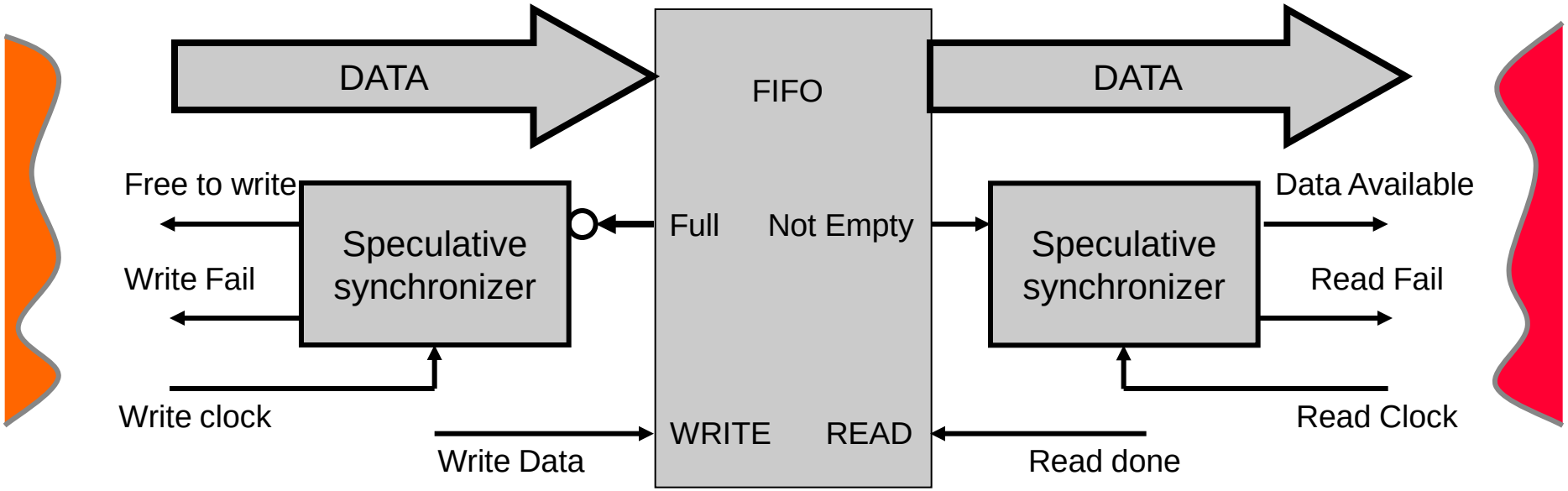
- Nominally 0 – 1 clock cycle
- Relies on accurately predicting conflicts
- Clocks must remain stable over synchronisation time.
- Always lose t_{ko} of next computation stage
- Alternative: shift all conflicts to next read cycle
  - On average this loses 2d
  - 2d must be big enough to cover any clock drift/jitter over synchronization time

Speculation

- Mostly, the synchronizer does not need 30τ to settle
- Only e^{-13} (0.00023%) need more than 13τ
- Why not go ahead anyway, and try again if more time was needed

Low latency synchronization

- Data Available, or Free to write are produced early.
- If they prove to be in error, synchronization failed.
- Read Fail or Write Fail flag is then raised and the action can be repeated.



When to recover

Early Data Available is set after a half cycle – 2 inverter delays
Speculative Data Available after a half cycle
If these two are different at end of cycle, set Fail

1. Early Half Cycle - 2τ	2. Speculative Half Cycle	Final End of Cycle	Fail 1 & 2 different End of Cycle	Comment
?	?	metastable?	metastable?	Unrecoverable error, Probability low.
0	0	0	0	No data was available
0	1	1	1	Stable at the end of the cycle, but the speculative output may have been metastable. Fail
1	1	1	0	Normal data Transfer

Synchronizer latency reduction

1. Only have one clock for the whole system
2. Use clocks with a predictable relationship
3. Speculate
4. Synchronise at the start of a burst transfer, the data rate is predictable during the burst

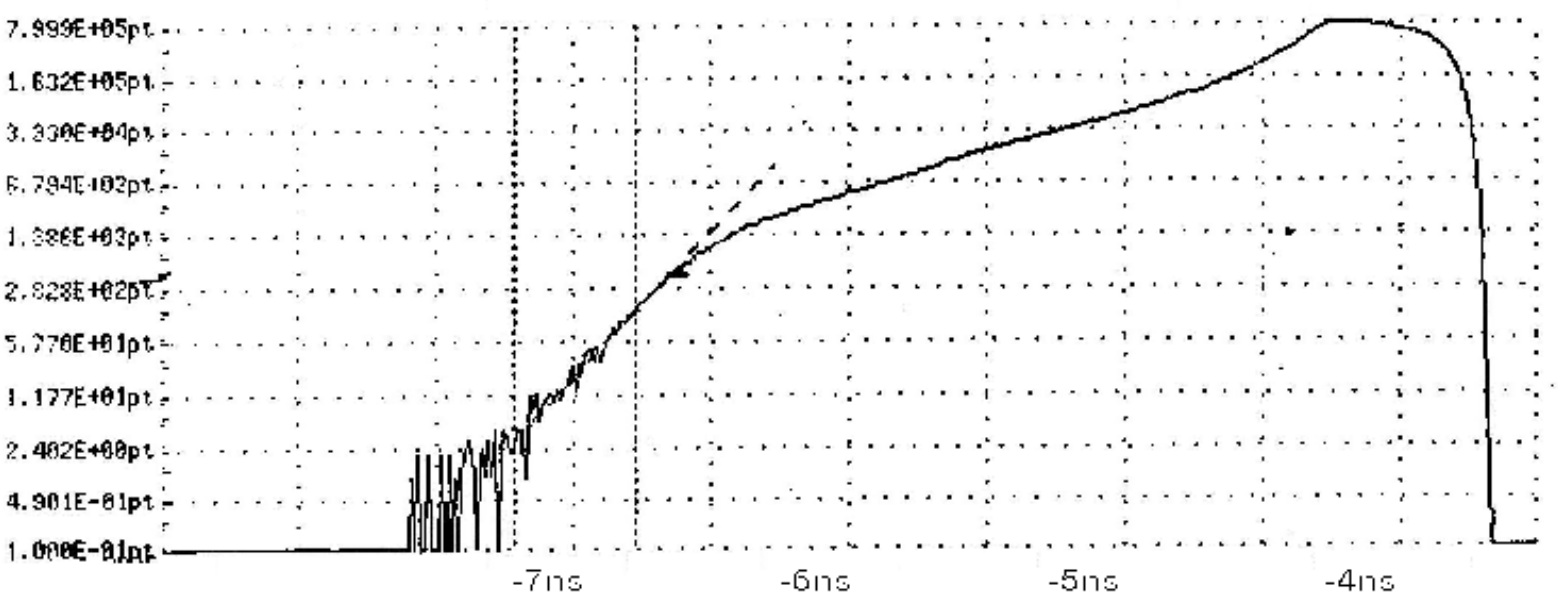
Outline

- What's the problem? Why does it matter?
- Time and distributions
- Noise – decisions are not deterministic.
- Synchronizer latency, can it be avoided?
- Measuring and some interesting effects

What we know about metastability

- Things we know
  - Synchronizers are unreliable, the more there are the more unreliable the system
  - How to measure reliability up to a few hours
- Things we know we don't know
  - What reliability is at 3 years
  - How to measure it
  - Complex circuits give complex results, the simple MTBF formula may not apply
- Things we don't know we don't know
  - What happens on the back edge of the clock

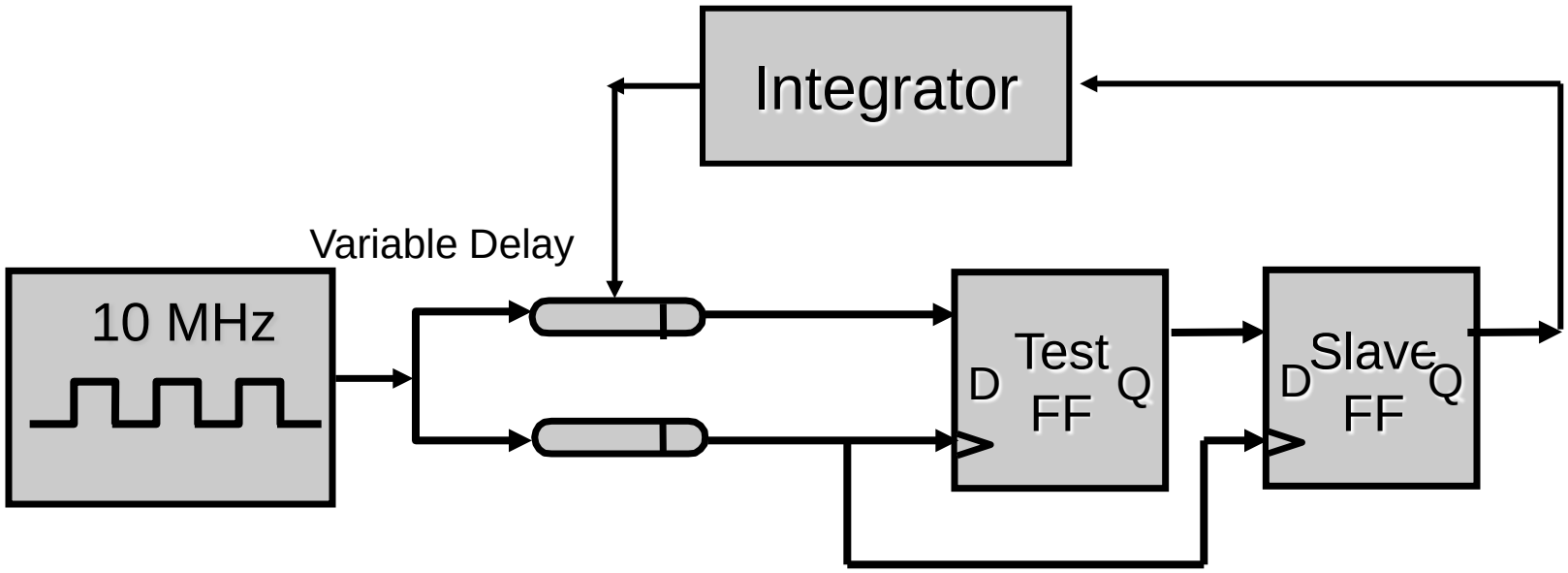
74F5074 Histogram



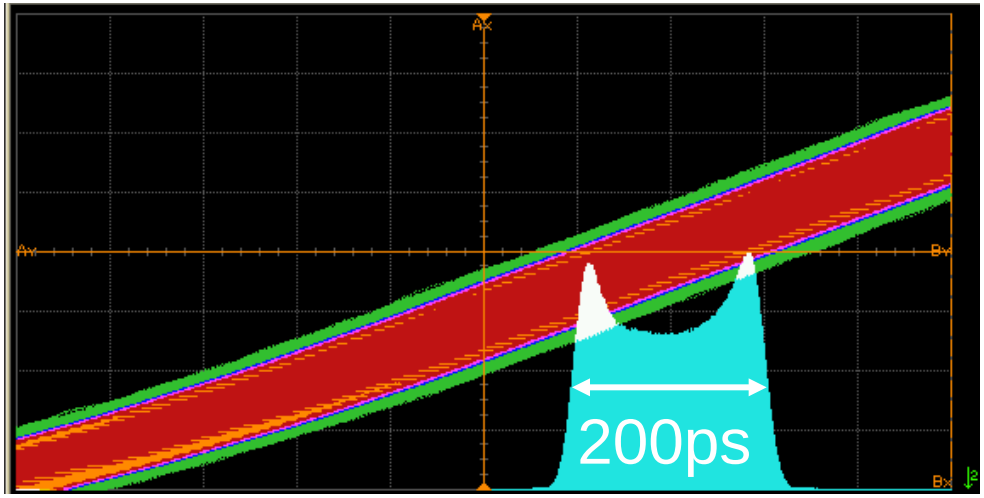
- Slope, τ, is about 120ps (in fast region)
- Typical delay time (most events) is 4ns
- 99.9% of clock cycles do not cause useful events
- To get 1 event at 7ns requires hours

Increasing the number of events

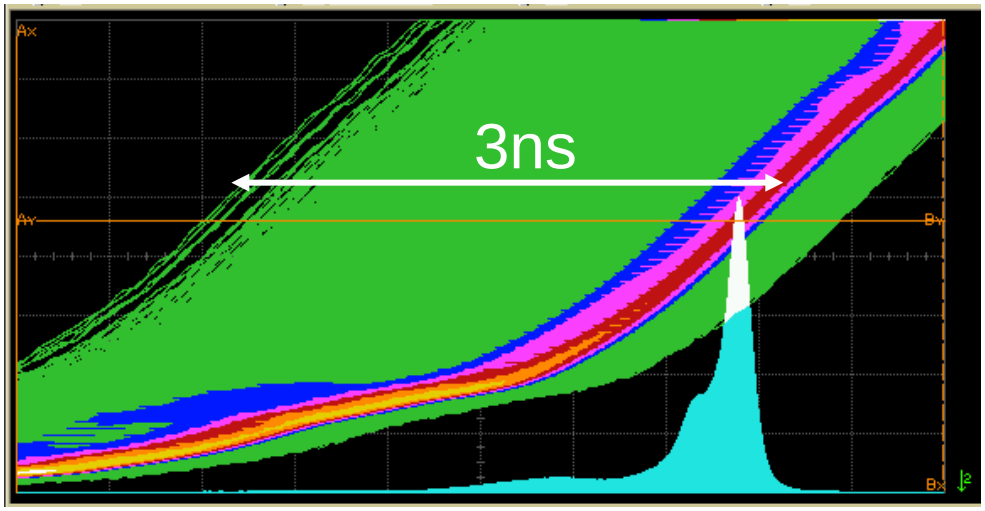
- Test FF is driven to metastability
- Every clock produces a metastable response
- Integrator ensures half outputs high, half low



What you get

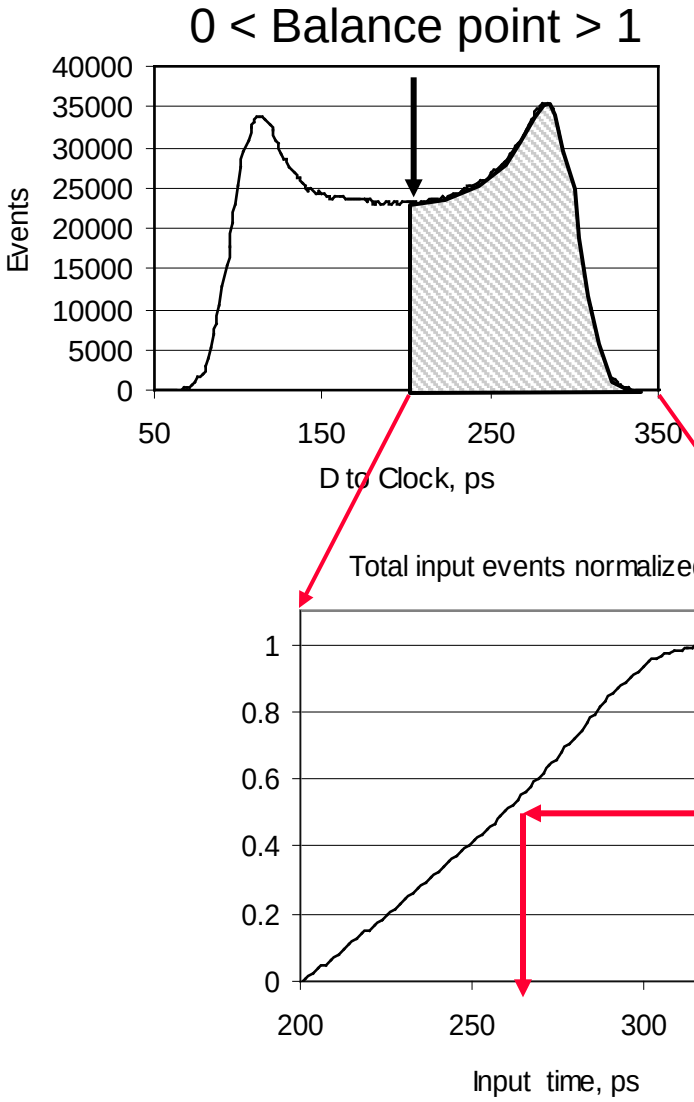


- Clock to D (Input) histogram



- Q to Clock (Output) histogram

Interpreting results



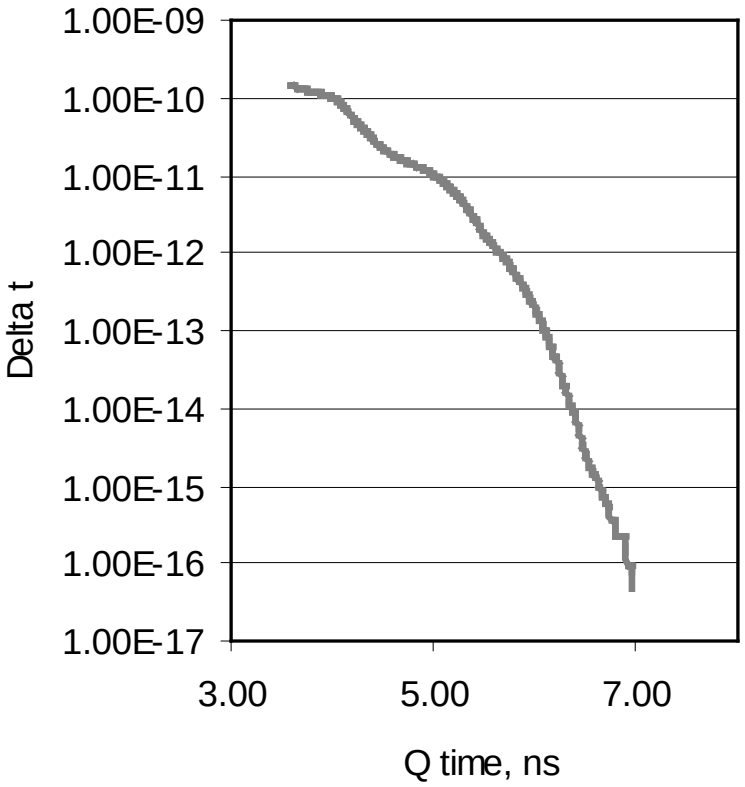
Input time distribution is not flat

Proportion of total inputs causing events vs input time

Proportion of total output events vs output time

Mapping output times to input times

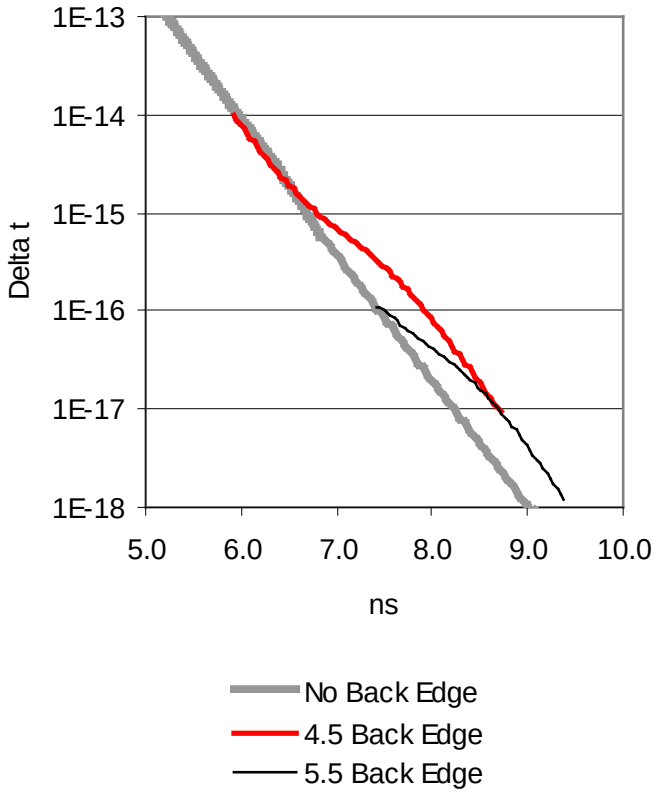
Results

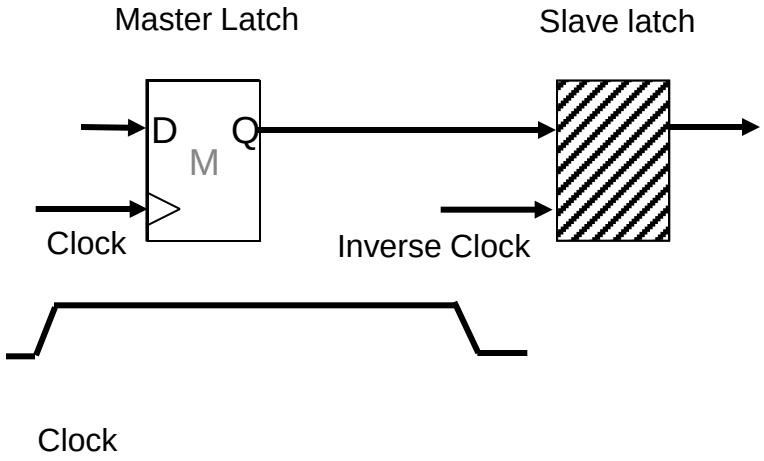


- Δ_t is the time from the "balance point" of ~200ps
- Similar to original graph BUT not events
- Orders of magnitude quicker to gather data
- Reliability for days not minutes
- Only one oscillator, so no distribution issues
- Δ_t does not depend on f_c and f_d or measurement time. Events do

$$\Delta_t = T_w e^{\frac{-t}{\tau}} \qquad MTBF = \frac{1}{\Delta_t f_c f_d}$$

When the clock goes low

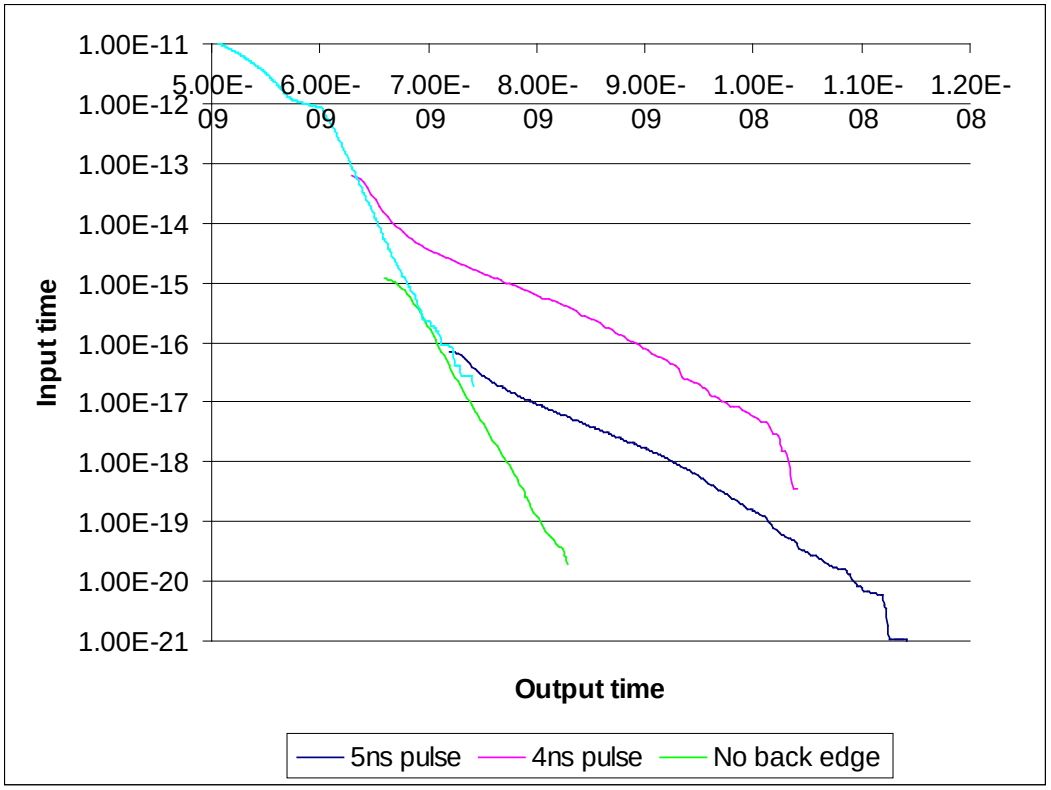


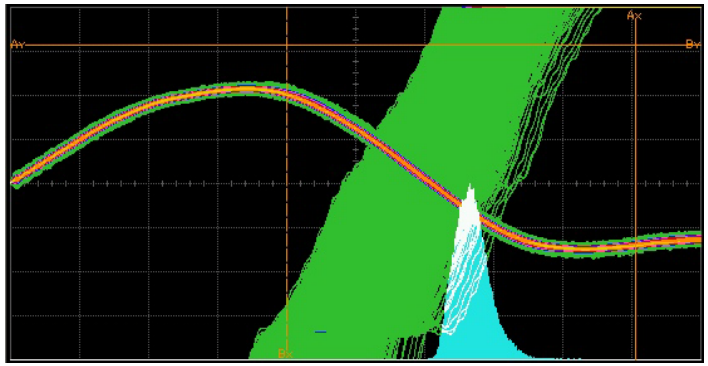


- Clock goes high, master goes metastable
- Master output arrives at slave
  - Before slave clock high: transparent gate delay t_d
  - As slave clock goes high: metastable, slightly longer delay

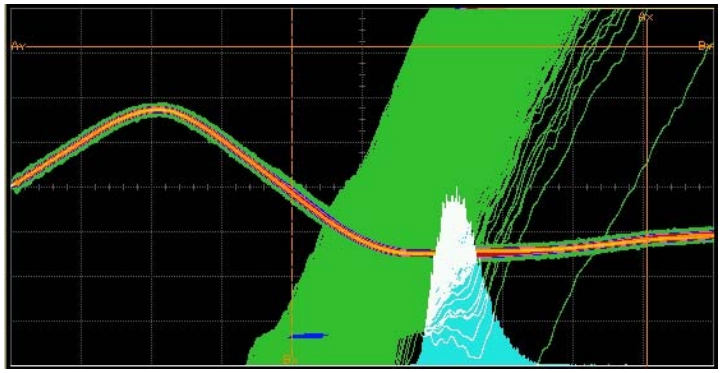
Back edge of clock causes increased delay

Effect of clock low on 74F5074





6 ns pulse



4 ns pulse

- 1 – 3 ns additional delay

Measurement results

- Reliability measurements extended from
  - 10^{-15} s or MTBF = 16 min at 10MHz, to
  - 10^{-22} s or MTBF = 3 years
- We can see variations in τ not previously seen
- Measurement is statistical, not affected by noise
- Not affected by oscillator linking
- Back edge of clock pulse is seen to be an important effect, can be 0 – 15τ

Conclusions

- Synchronization/arbitration requires special circuit elements
- They’re not digital!
- Well known models of synchronizers and arbiters exist
- Design gets more difficult with small dimensions
- Synchronizers and arbiters are not deterministic.
- Both circuits and systems may not conform to idealized models
- Differences can lead to poor reliability, unfair arbitration, and unpredictable performance