(Introduction to)

Data Acquisition

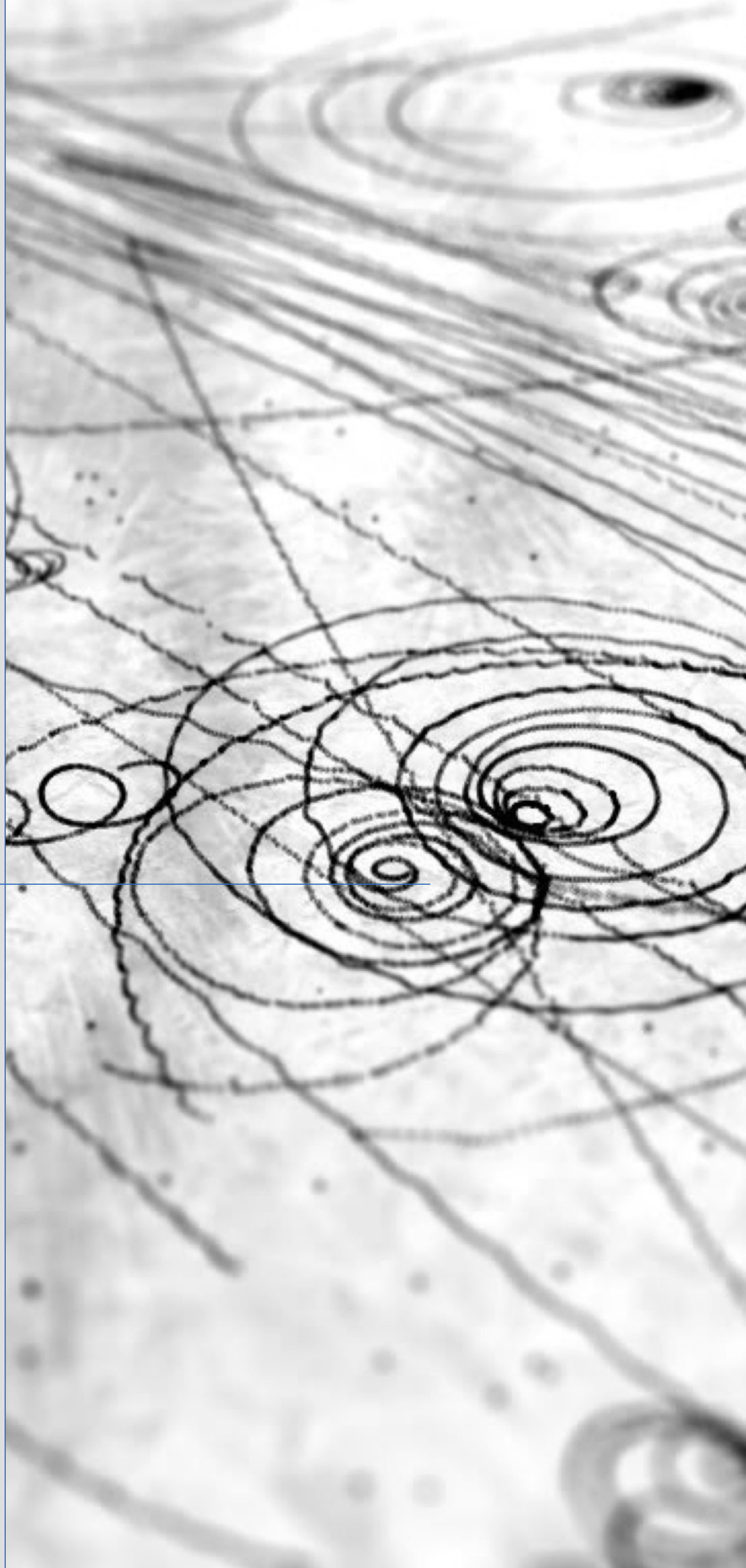
Advanced Graduate Lectures on practical Tools, Applications and Techniques in HEP

May 23, 2023

Alessandro Thea

Rutherford Appleton Laboratory - PPD

Science and Technology Facilities Council



Acknowledgements

Lecture inherited from Monika Wielers

- Ideas, material and more from Andrea Venturi, Francesca Pastore and many others

Outline

1. Introduction

1.1. What is DAQ?

1.2. System architecture

2. Basic DAQ concepts

2.1. Digitization, Latency

2.2. Deadtime, Busy, Backpressure

2.3. De-randomization

3. Scaling up

3.1. Readout and Event Building

3.2. Buses vs Network

4. DAQ challenges at the LHC and beyond

What is DAQ?

[Wikipedia]

Data **A**c**Q**uisition (**DAQ**) is

- the process of **sampling signals**
- that **measure** real world physical conditions
- and **converting** the resulting samples **into digital** numeric values that can be manipulated by a computer

What is DAQ?

[Wikipedia]

Data **A**c**Q**uisition (**DAQ**) is

- the process of **sampling signals**
- that **measure** real world physical conditions
- and **converting** the resulting samples **into digital** numeric values
  that can be manipulated by a computer

Ingredients:

- **Sensors**: convert physical quantities to electrical signals
- **Analog-to-digital converters**: convert conditioned sensor signals to digital values
- **Processing** and **storage** elements

What is DAQ?

[Real life]

DAQ is an **heterogeneous** field (*a.k.a. dark arts*)

- with boundaries not well defined

An **alchemy** of

- physics
- electronics
- computer science
- hacking
- networking
- experience

Where money and manpower matter as well

DAQ duties

Gather data produced by detectors

- **Readout**

Form complete events

- **Data Collection** and **Event Building**

Possibly feed extra processing levels

Store event data

- **Data Logging**

Data Flow

Manage operations

- **Control**, **Configuration**, **Monitoring**

Interlude: data vs *interesting* data

Interesting physics data typically a small fraction of sampled signals

- **really, Really, REALLY** small

Logging all recorded data is unpractical (and costly)

- sometimes technically unfeasible

Online data reduction before logging becomes imperative

That's the job of the **Trigger**!

- DAQ and Trigger deeply entwined
  - often referred as TDAQ

Trigger Lecture - Dr. Julie Kirk

- All you wanted to know about trigger and never dared to ask, today, after coffee break! ☕

Trigger in a nutshell

Selects interesting events **AND** rejects boring ones, *in real time*

- **Selective:** efficient for “signal” and resistant to “background”
- **Simple** and **robust:** Must be predictable at all times!
- **Fast:** Late is no better than never

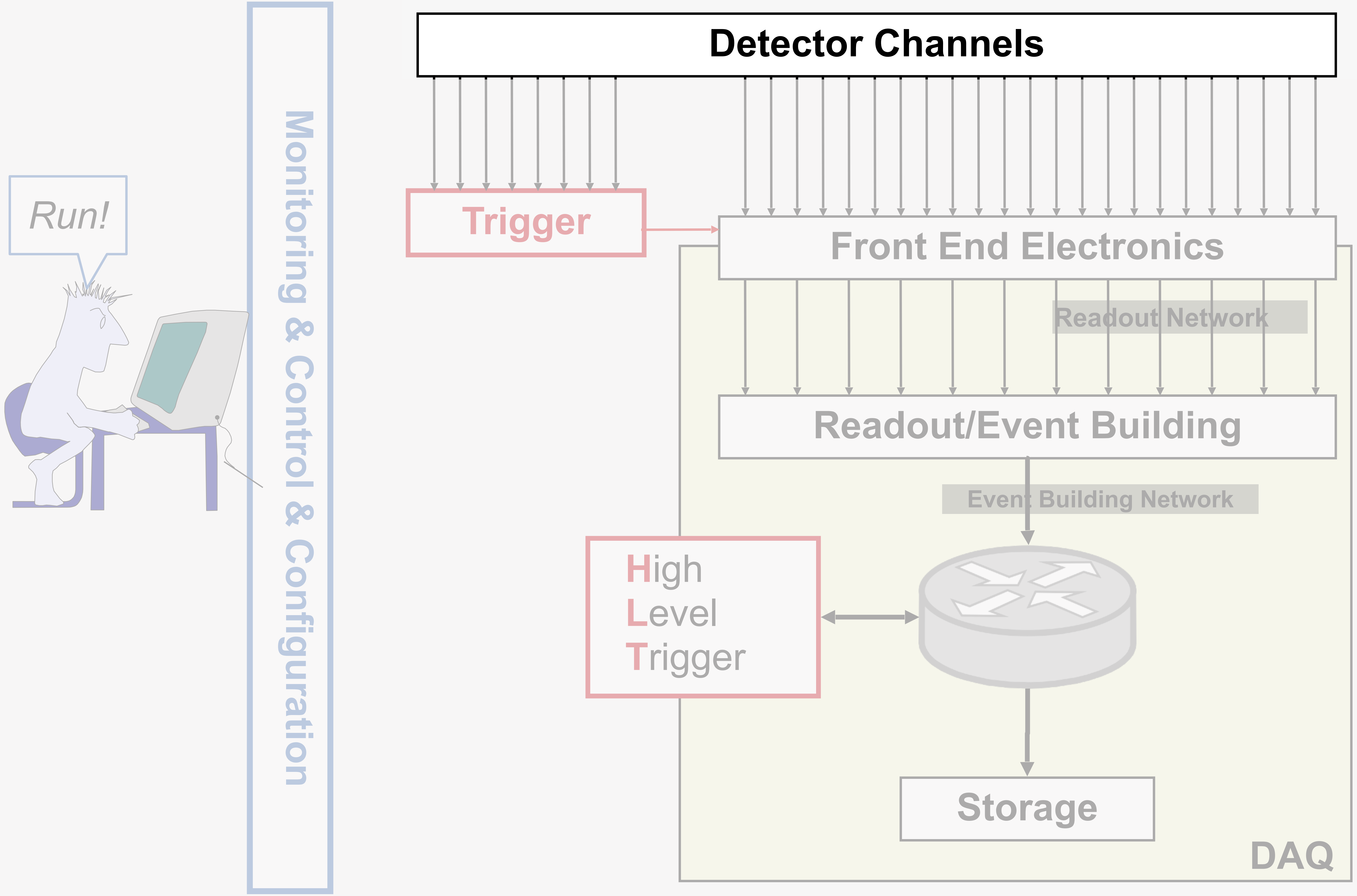
With minimal *controlled* **latency**

- time it takes to form and distribute its decision

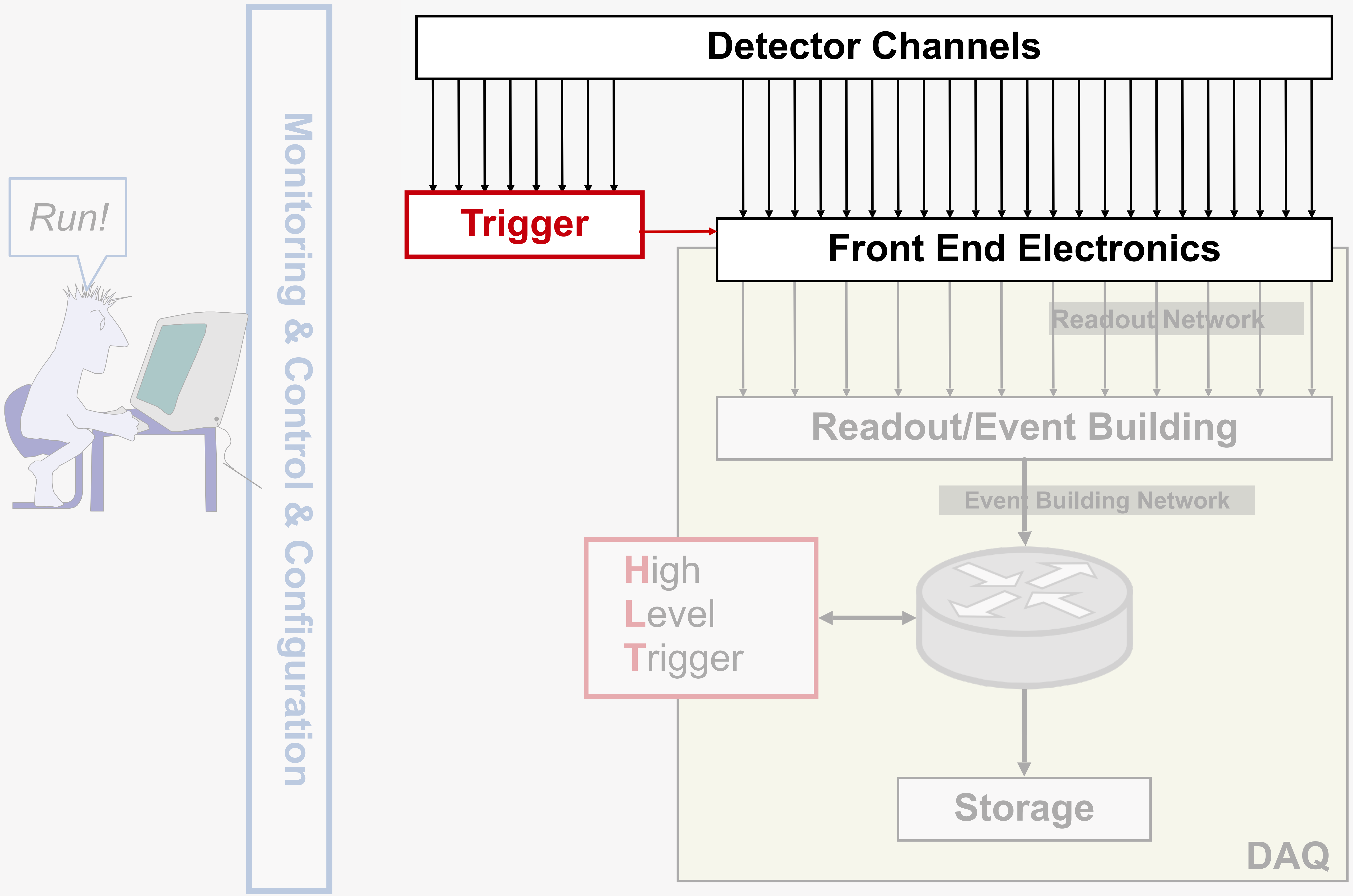
The implementation of “Trigger” has significantly evolved in the past decades

- **LEP**: trigger the sampling of (slow) detector
- **LHC**: trigger the readout of on-detector buffers (*Level-1*) or trigger logging to permanent storage (*High Level Trigger - HLT*)

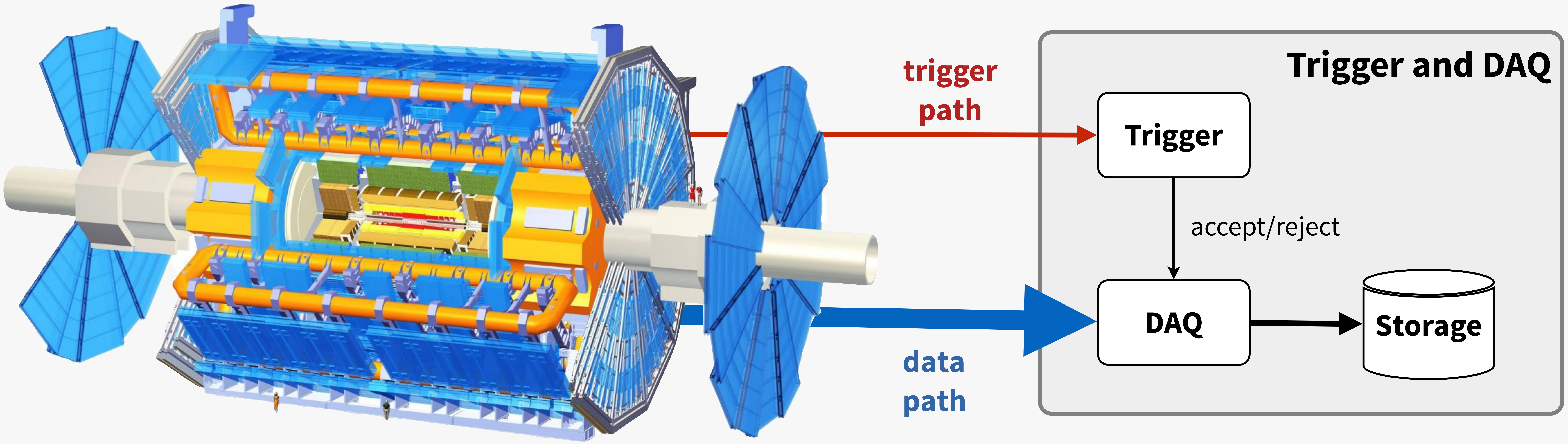
T-DAQ Architecture



T-DAQ Architecture



From detector to TDAQ



Trigger path

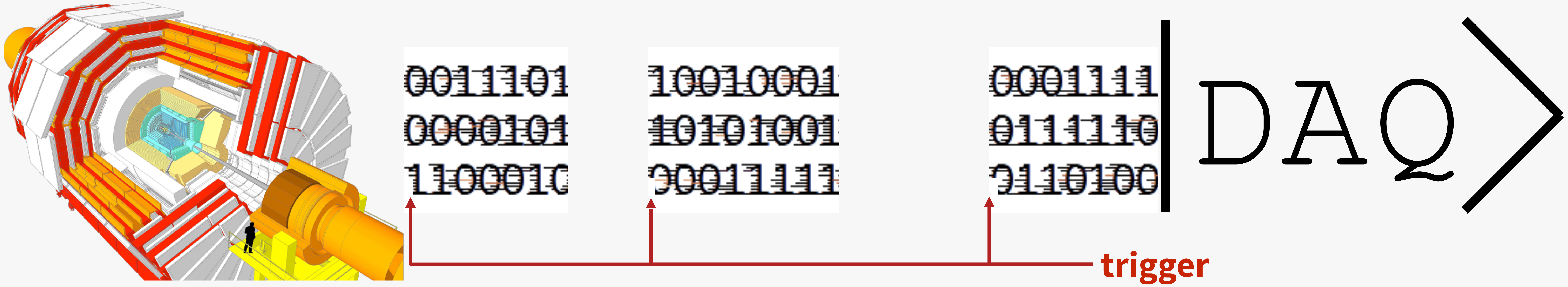
- From specific detectors to trigger logic
- Continuous streaming of trigger data
- Dedicated connections

Data path

- From all the detectors to readout
- Transmission on positive trigger decision

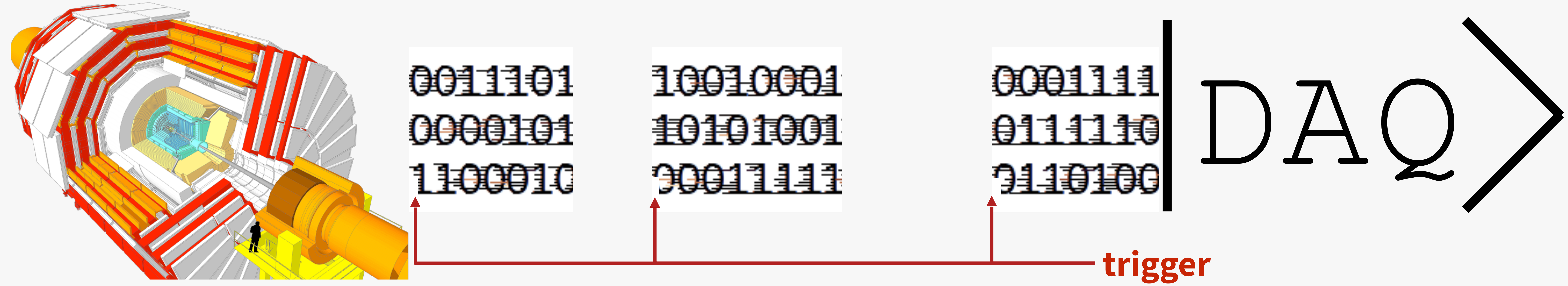
Readout: Triggered vs Streaming

Triggered: data is readout from detector only when a trigger signal is raised



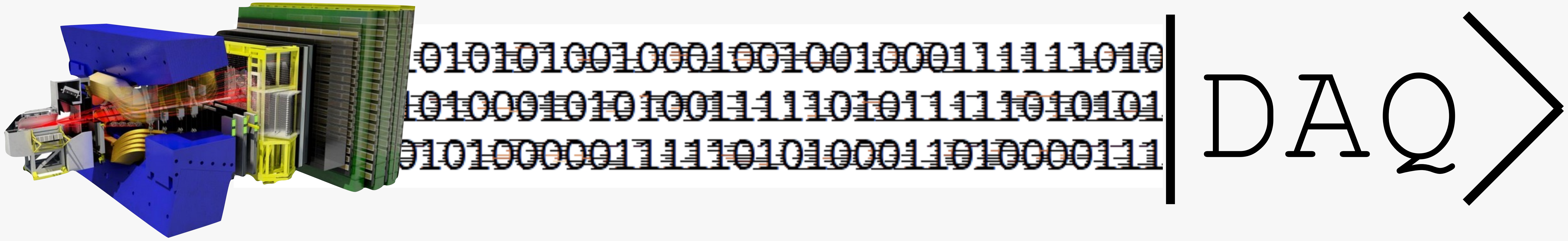
Readout: Triggered vs Streaming

Triggered: data is readout from detector only when a trigger signal is raised

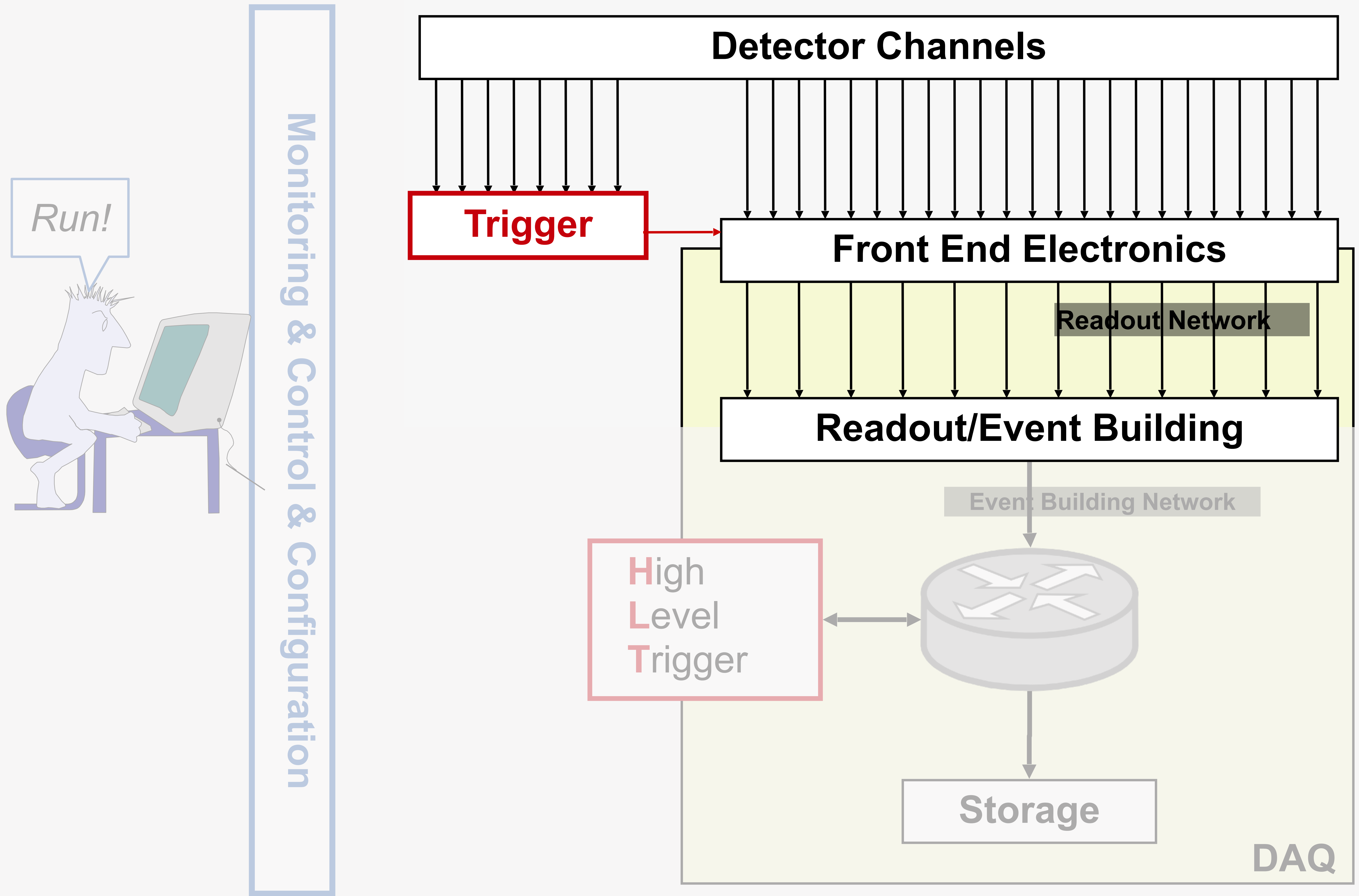


Streaming: detector pushes all its data and the downstream DAQ must keep the pace

- data reduction still takes place, but post readout



T-DAQ Architecture



Field Programmable Gate Arrays

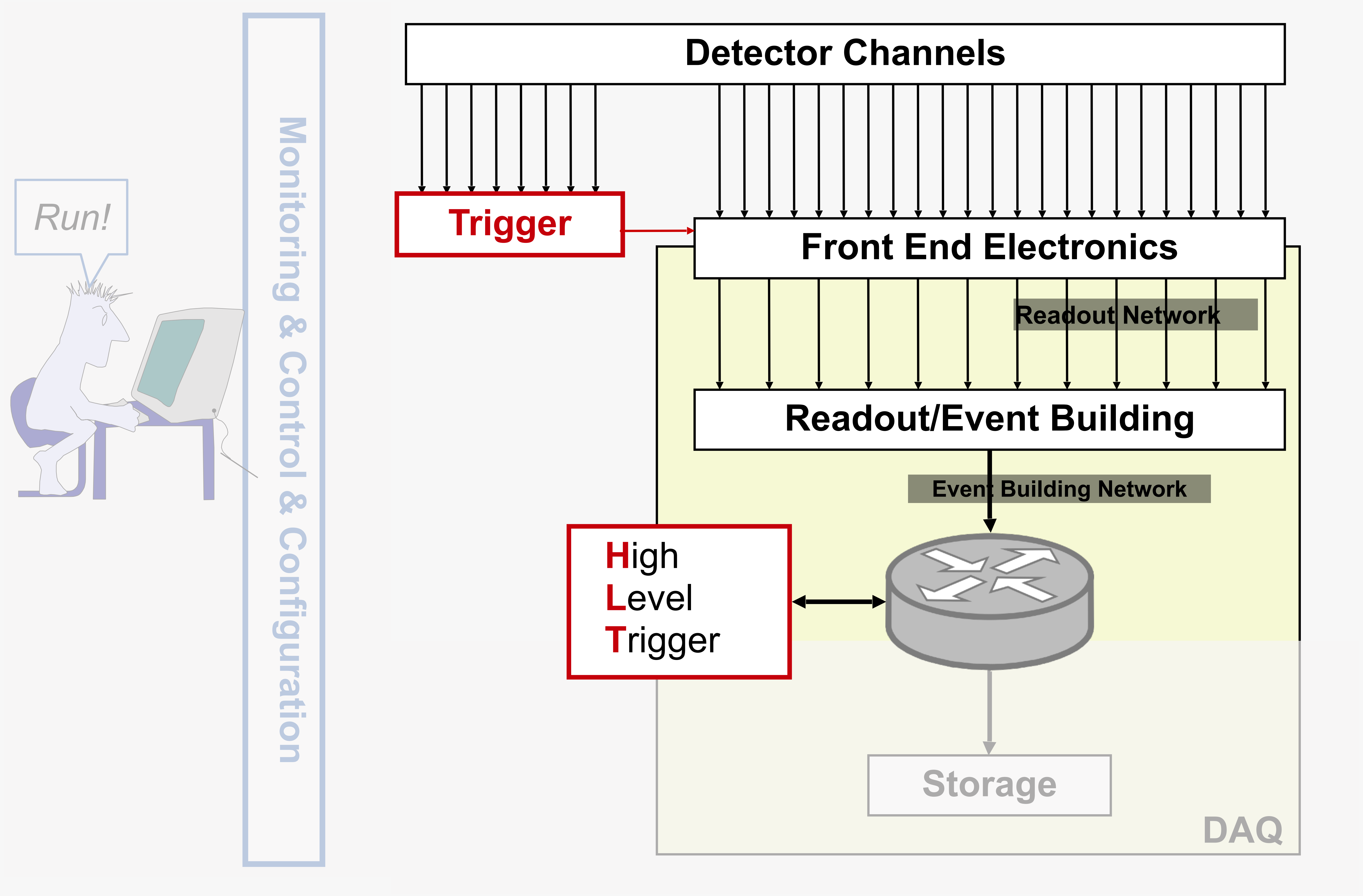
FPGAs are becoming TDAQ's bread & butter

- Signal processing, data formatting, natively parallel tasks (e.g. pattern recognition), machine learning, ...

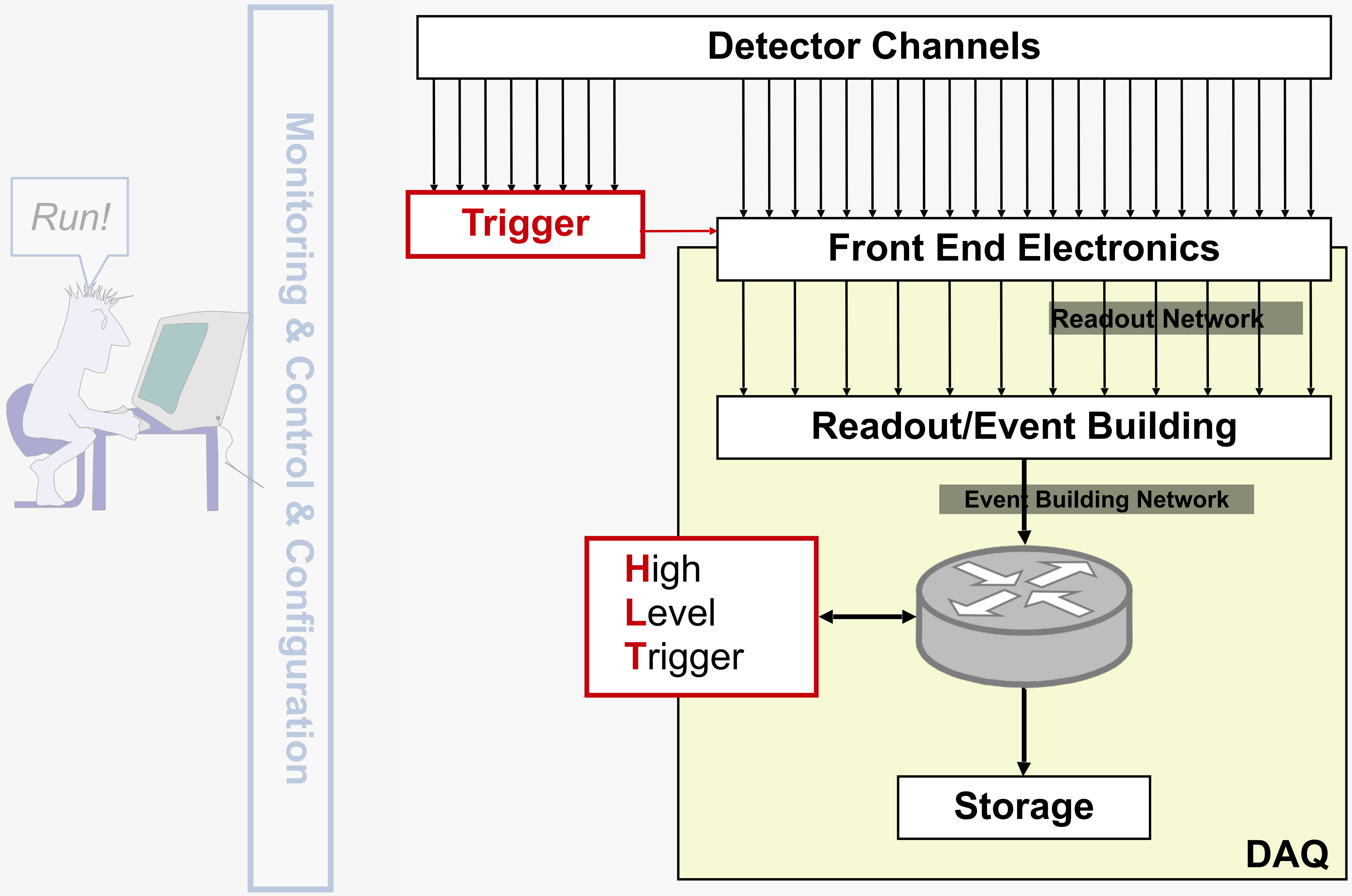
FPGA Programming Lecture -
Dr. Kristian Harder

- Plenty of gory details about LUTs, BRAMs and VHDL to keep you awake at night!

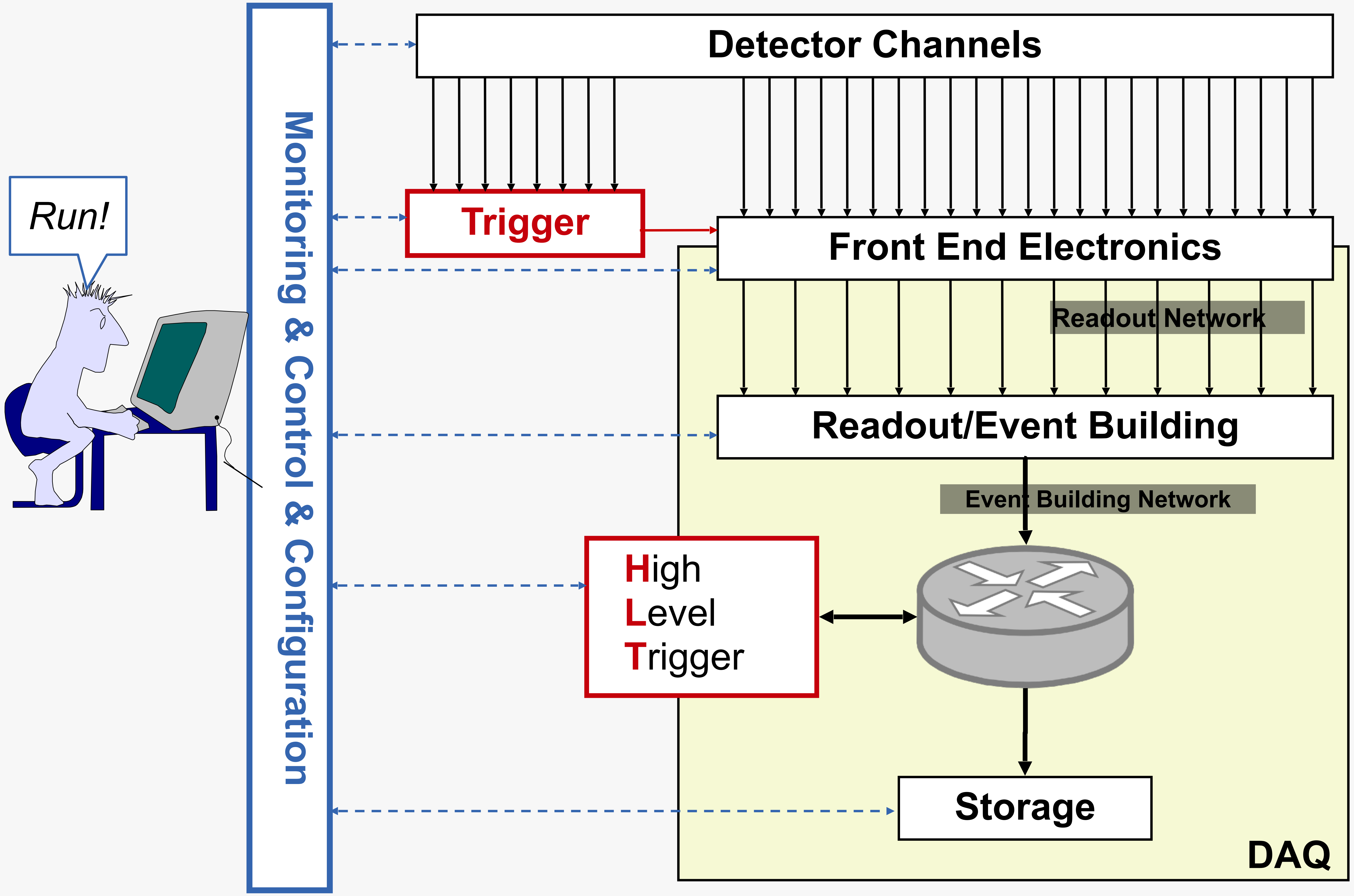
T-DAQ Architecture



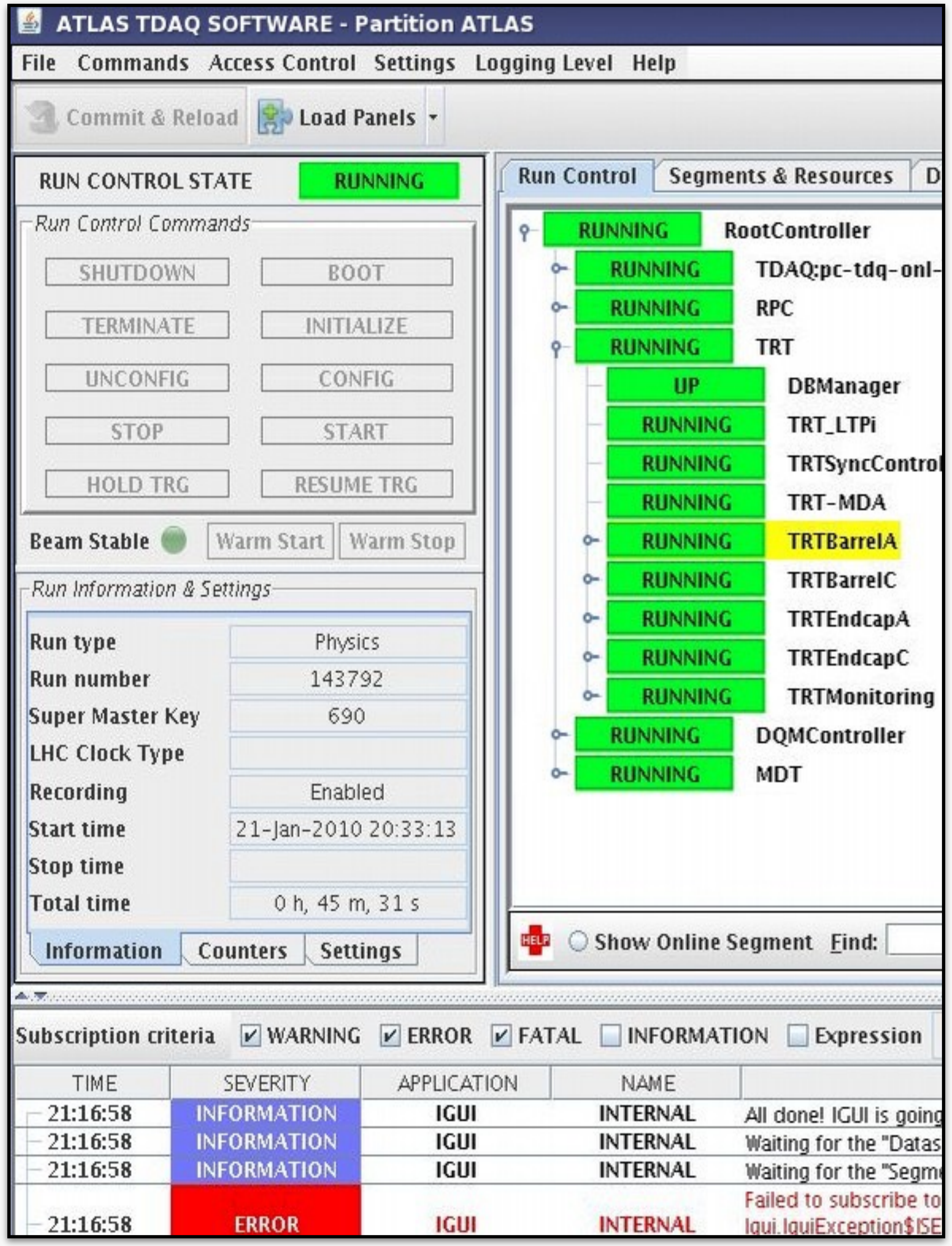
T-DAQ Architecture



T-DAQ Architecture



The glue of your experiment



Configuration

- Ensemble of detectors, trigger and DAQ parameters defining the system behaviour during data taking

Control

- Orchestrate applications participating to data taking
- Via distributed Finite State Machine

Monitoring

- Of data taking operations
  - What is going on?
  - What happened?
  - When?
  - Where?

Outline

with a toy model

Basic DAQ: periodic trigger

Eg: measure temperature at a fixed frequency

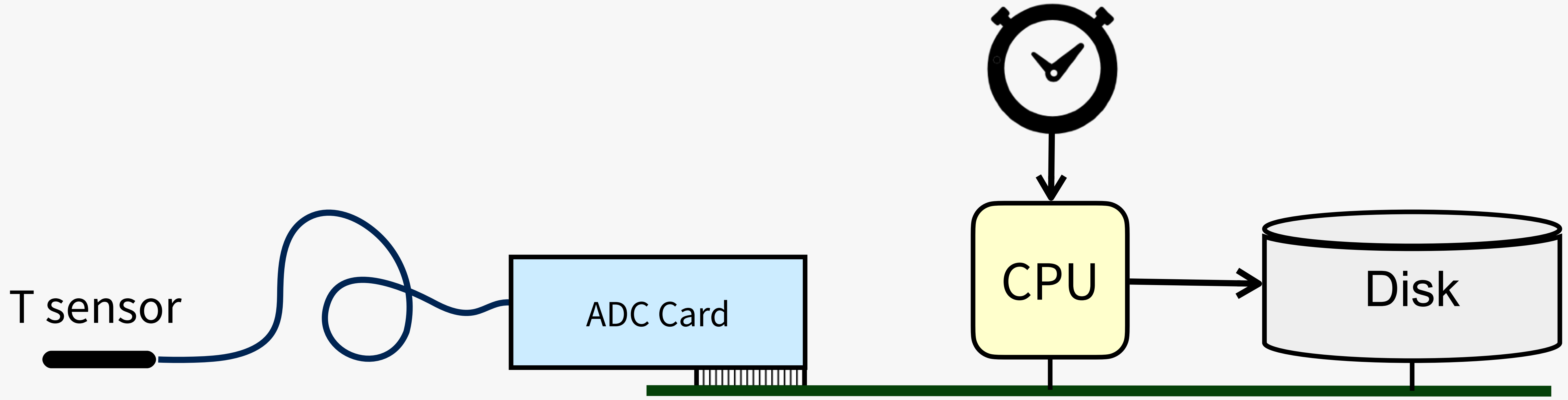
- Clock triggered

ADC performs analog to digital conversion, digitization (our front-end electronics)

- Encoding analog value into binary representation

CPU does

- Readout, Processing, Storage



Physical View

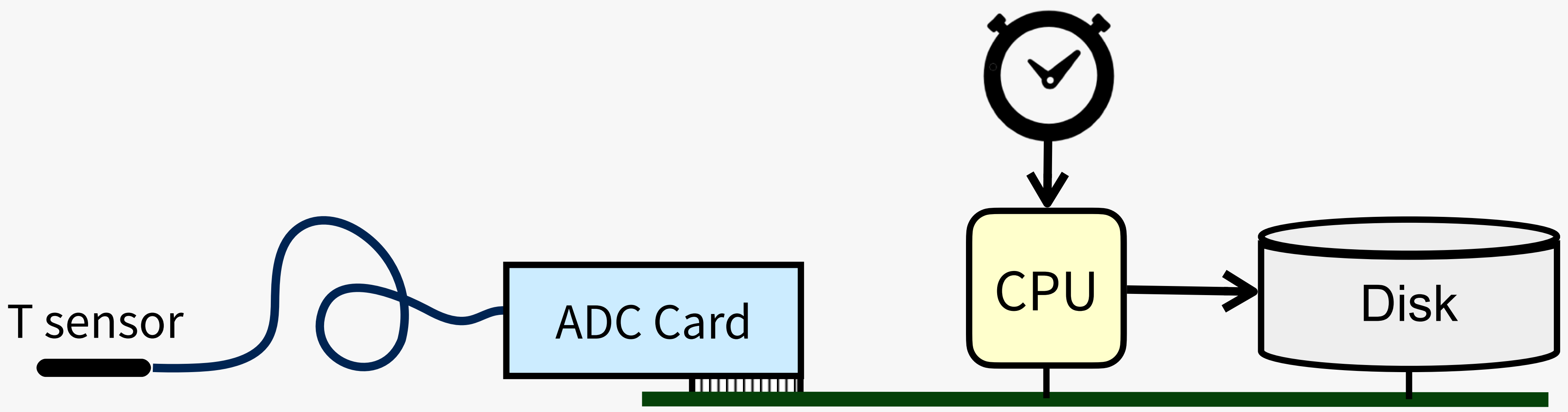
Basic DAQ: periodic trigger

System clearly limited by the time τ to process an "event"

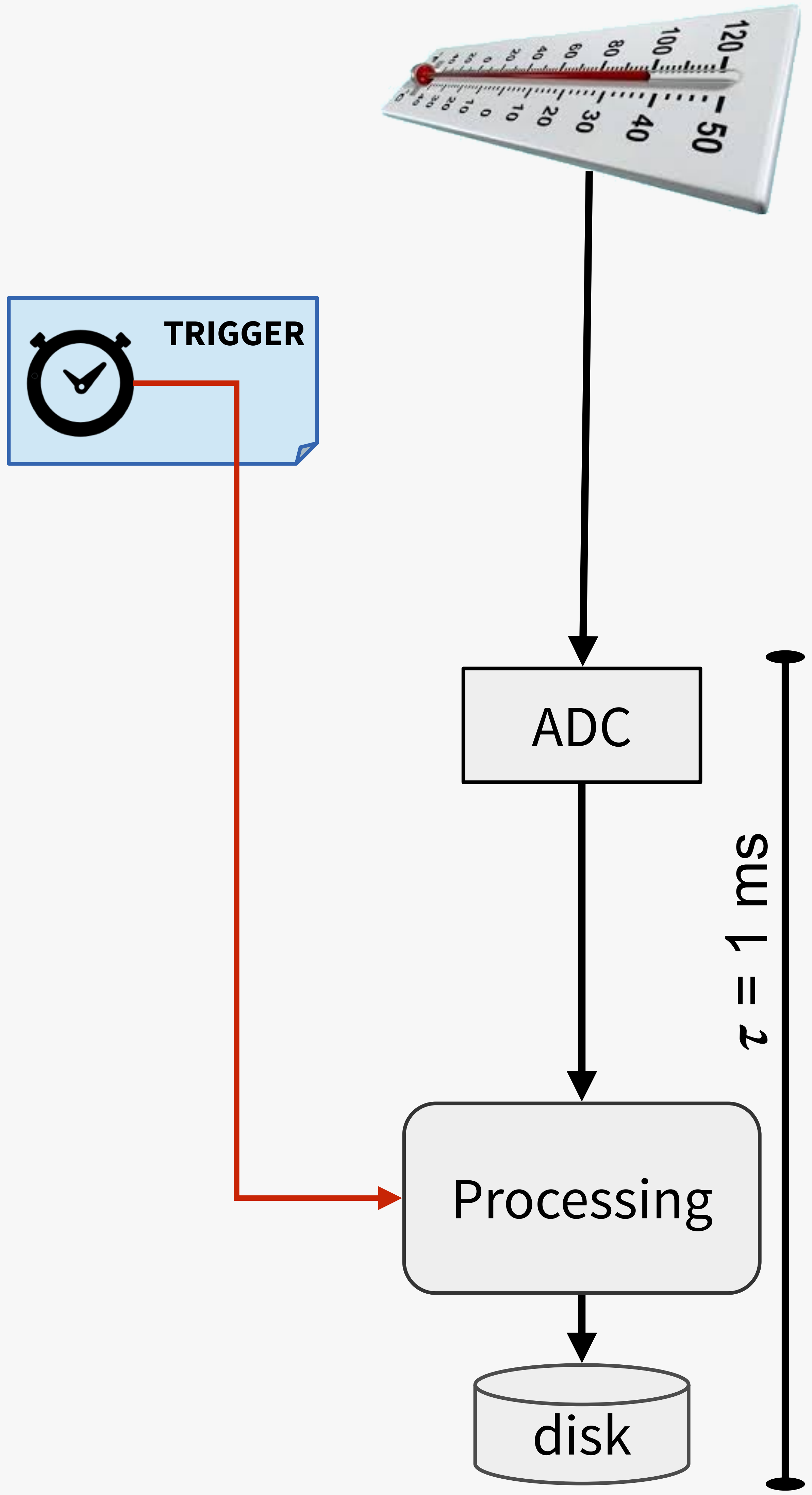
- ADC conversion +
  CPU processing +
  Storage

The DAQ maximum sustainable rate is simply the inverse of τ, e.g.:

- E.g.: $\tau = 1$ ms $R = 1/\tau = 1$ kHz



Physical View



Basic DAQ: "real" trigger

Events asynchronous and unpredictable

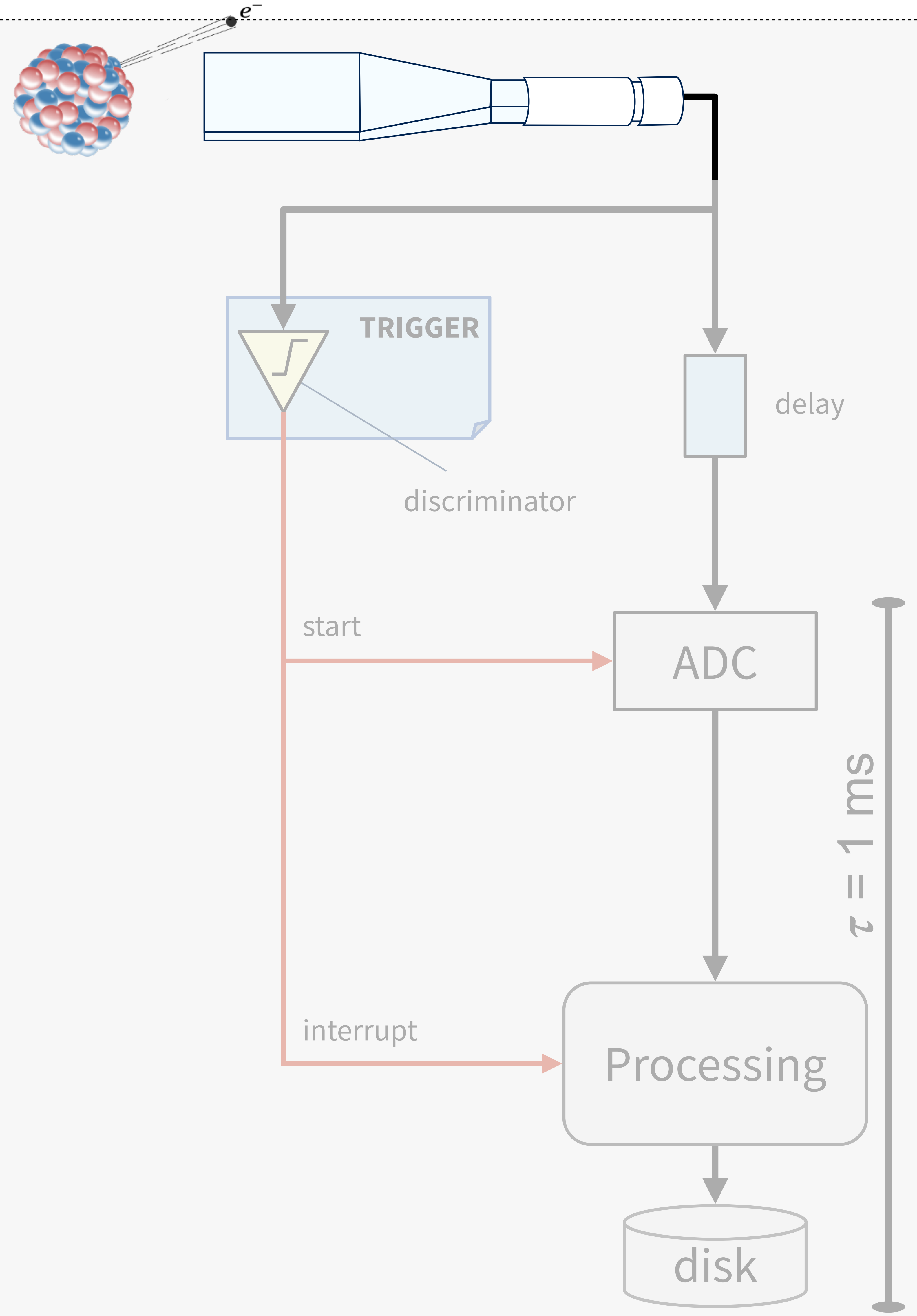
- E.g.: beta decay studies

A physics trigger is needed

- **Discriminator**: generates an output digital signal if amplitude of the input pulse is greater than a given threshold

NB: delay introduced to compensate for the **trigger latency**

- Signal split in trigger and data paths



Basic DAQ: "real" trigger

Events asynchronous and unpredictable

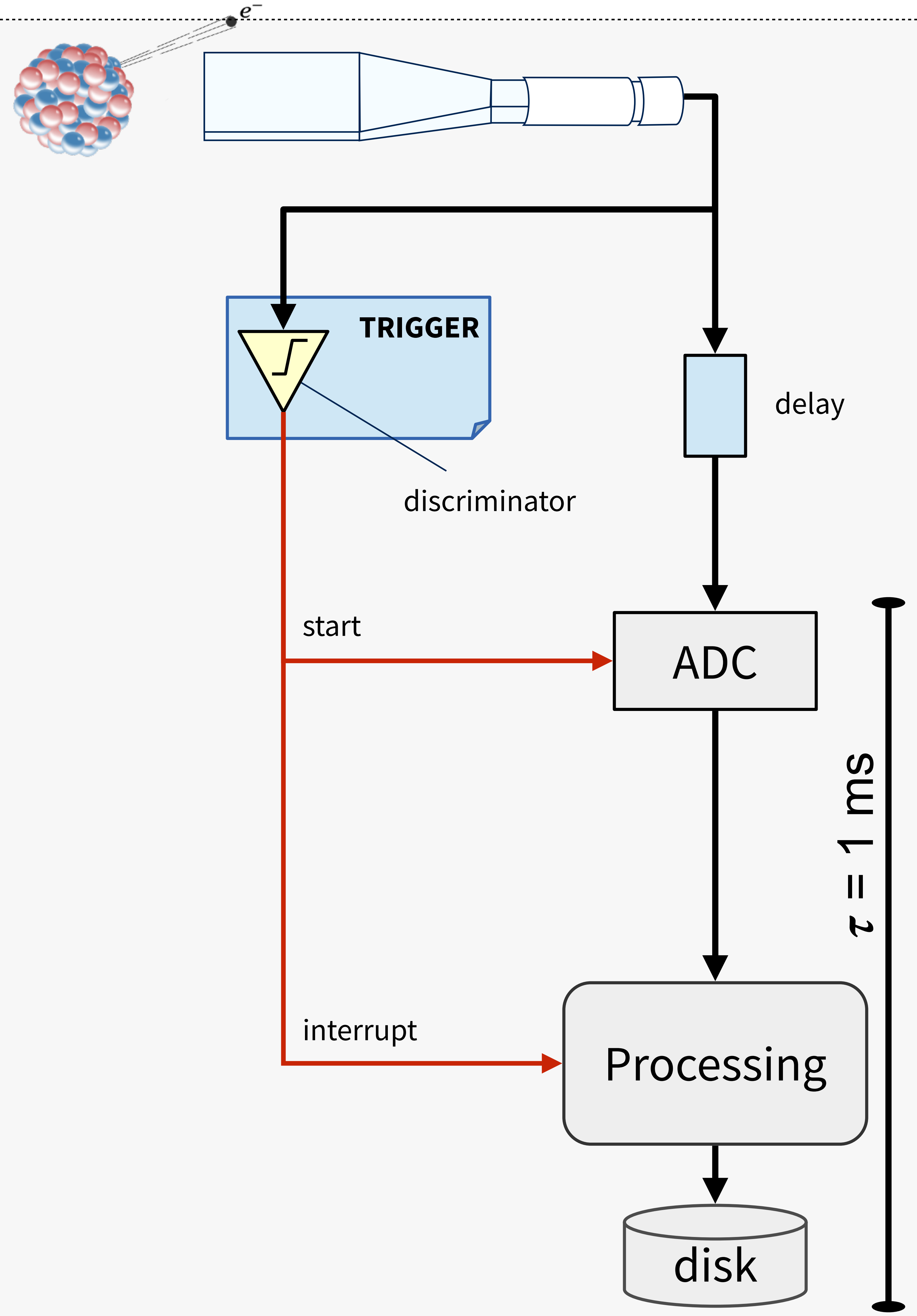
- E.g.: beta decay studies

A physics trigger is needed

- **Discriminator**: generates an output digital signal if amplitude of the input pulse is greater than a given threshold

NB: delay introduced to compensate for the **trigger latency**

- Signal split in trigger and data paths



Basic DAQ: “real” trigger

Events asynchronous and unpredictable

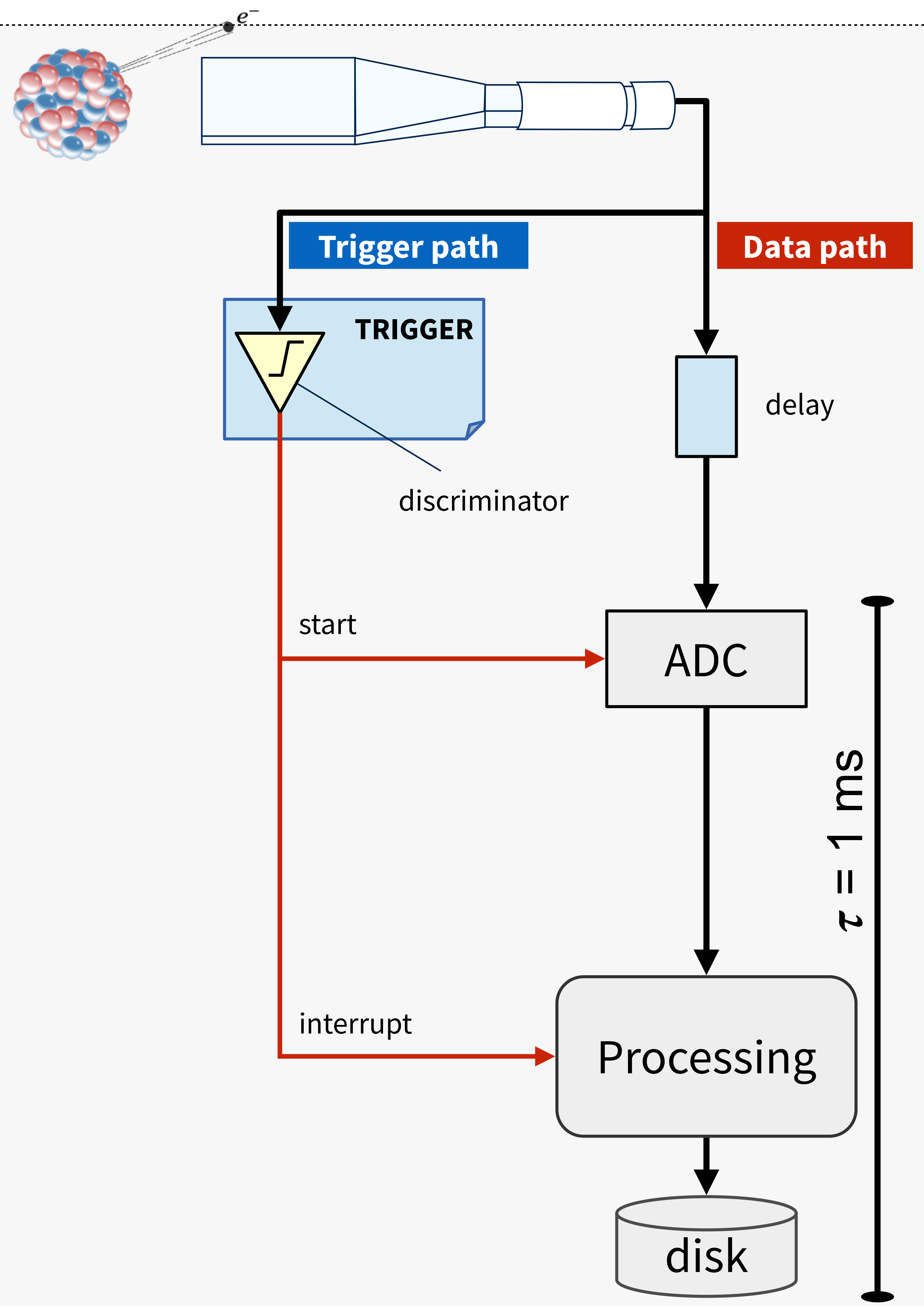
- E.g.: beta decay studies

A physics trigger is needed

- **Discriminator**: generates an output digital signal if amplitude of the input pulse is greater than a given threshold

NB: delay introduced to compensate for the **trigger latency**

- Signal split in trigger and data paths



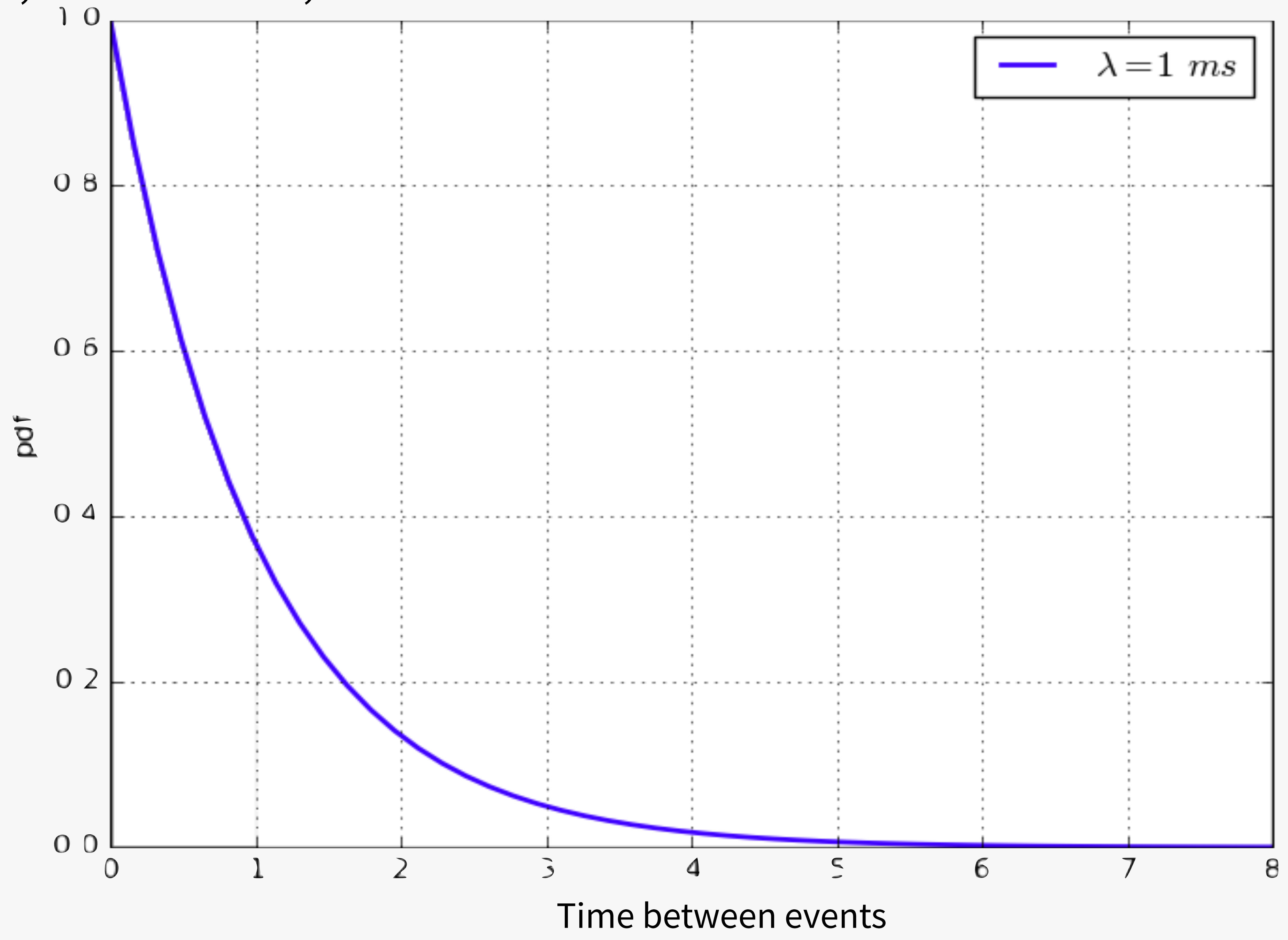
Basic DAQ: “real” trigger

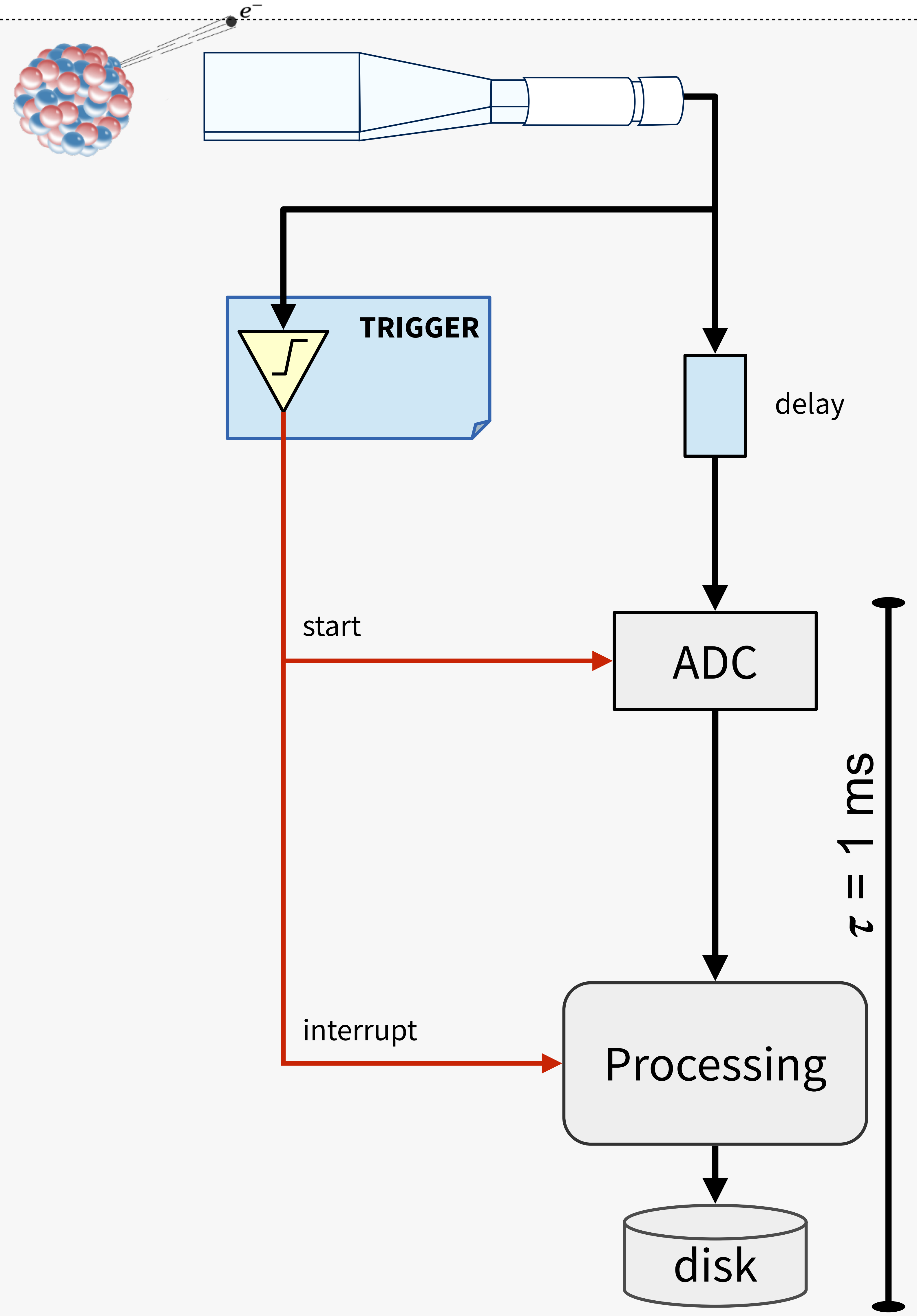
Stochastic process

- Fluctuations in time between events

Let's assume for example

- physics rate $f = 1$ kHz, i.e. $\lambda = 1$ ms
- and, as before, $\tau = 1$ ms





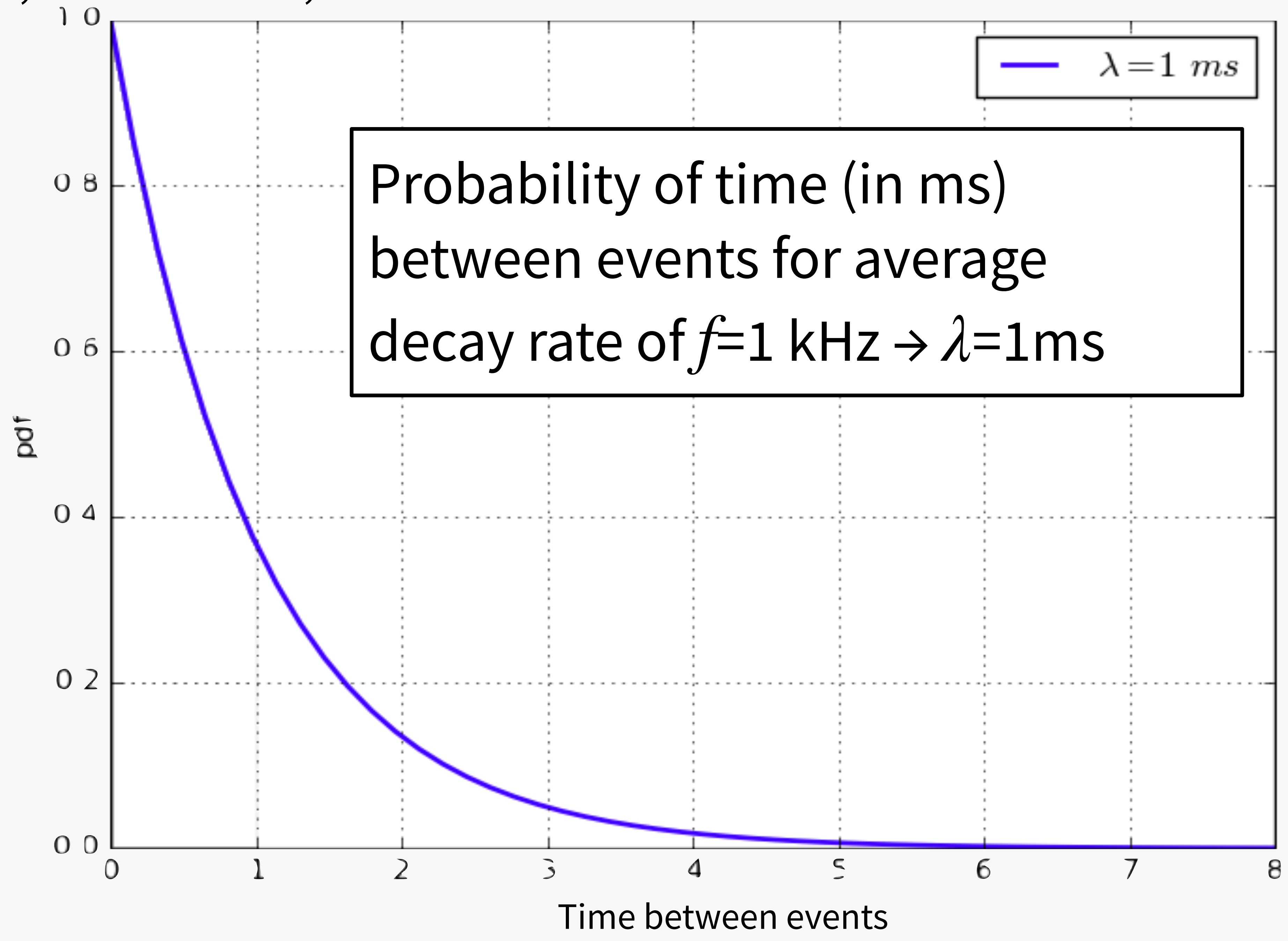
Basic DAQ: "real" trigger

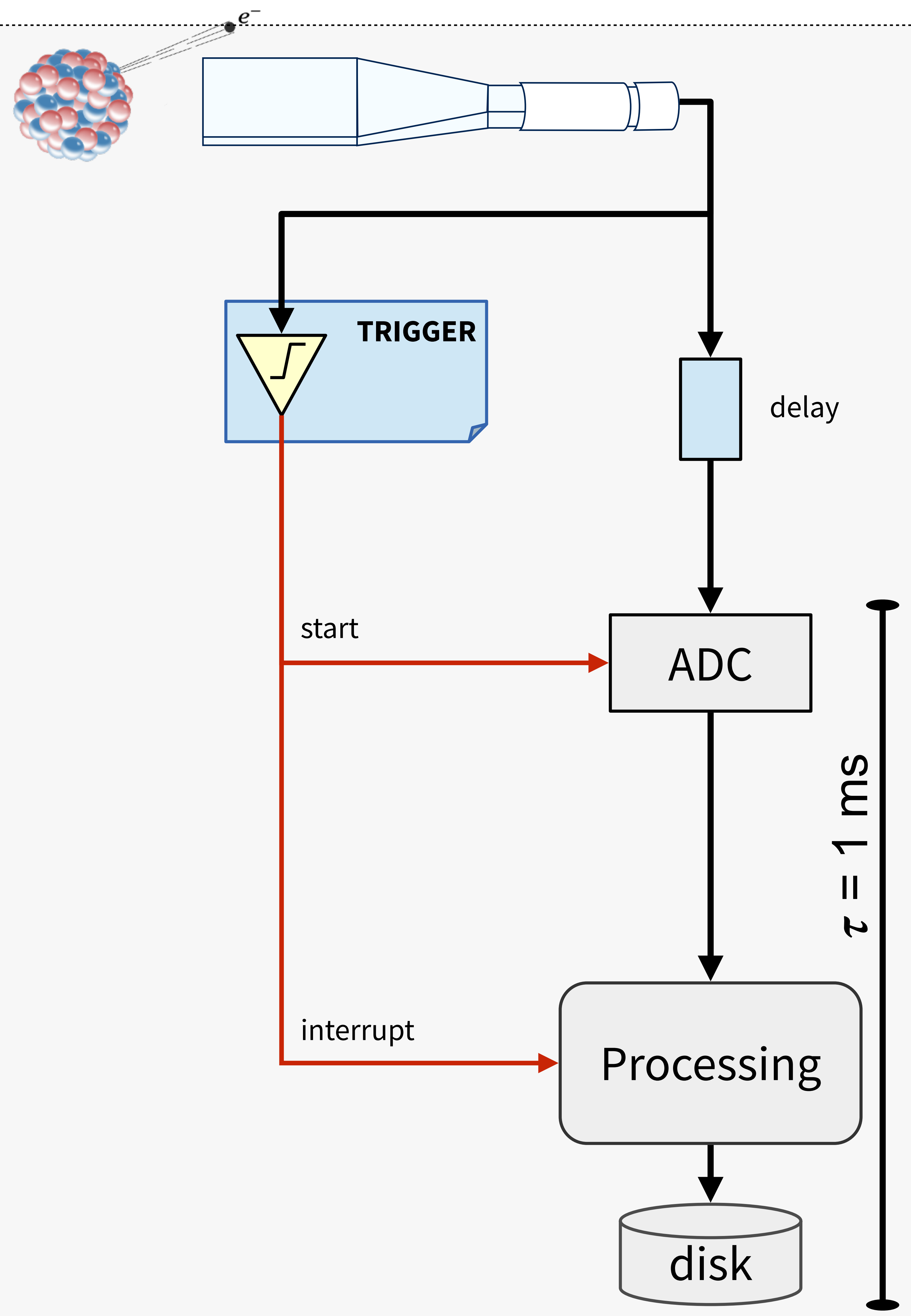
Stochastic process

- Fluctuations in time between events

Let's assume for example

- physics rate f = 1 kHz, i.e. λ = 1 ms
- and, as before, τ = 1 ms





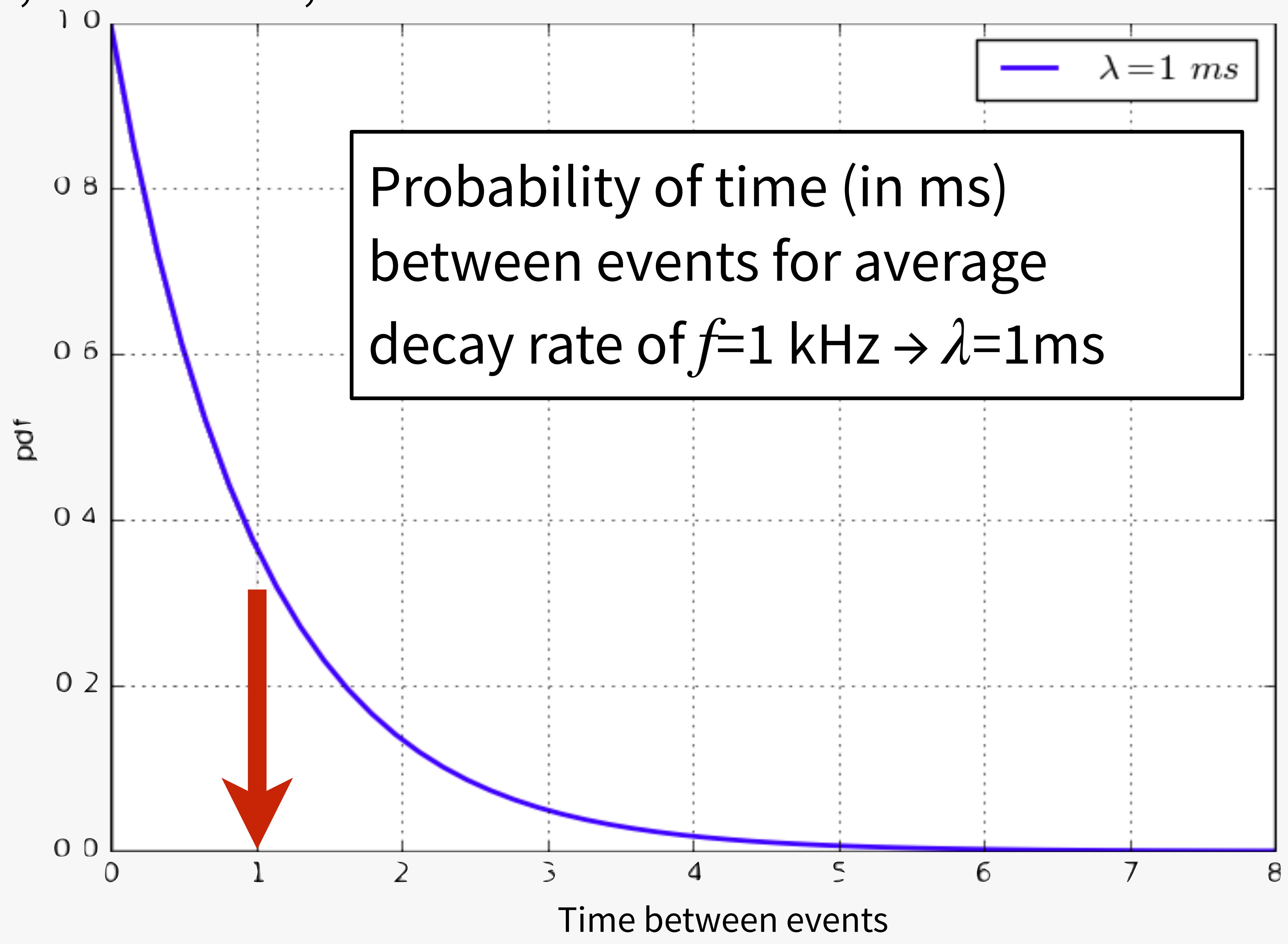
Basic DAQ: “real” trigger

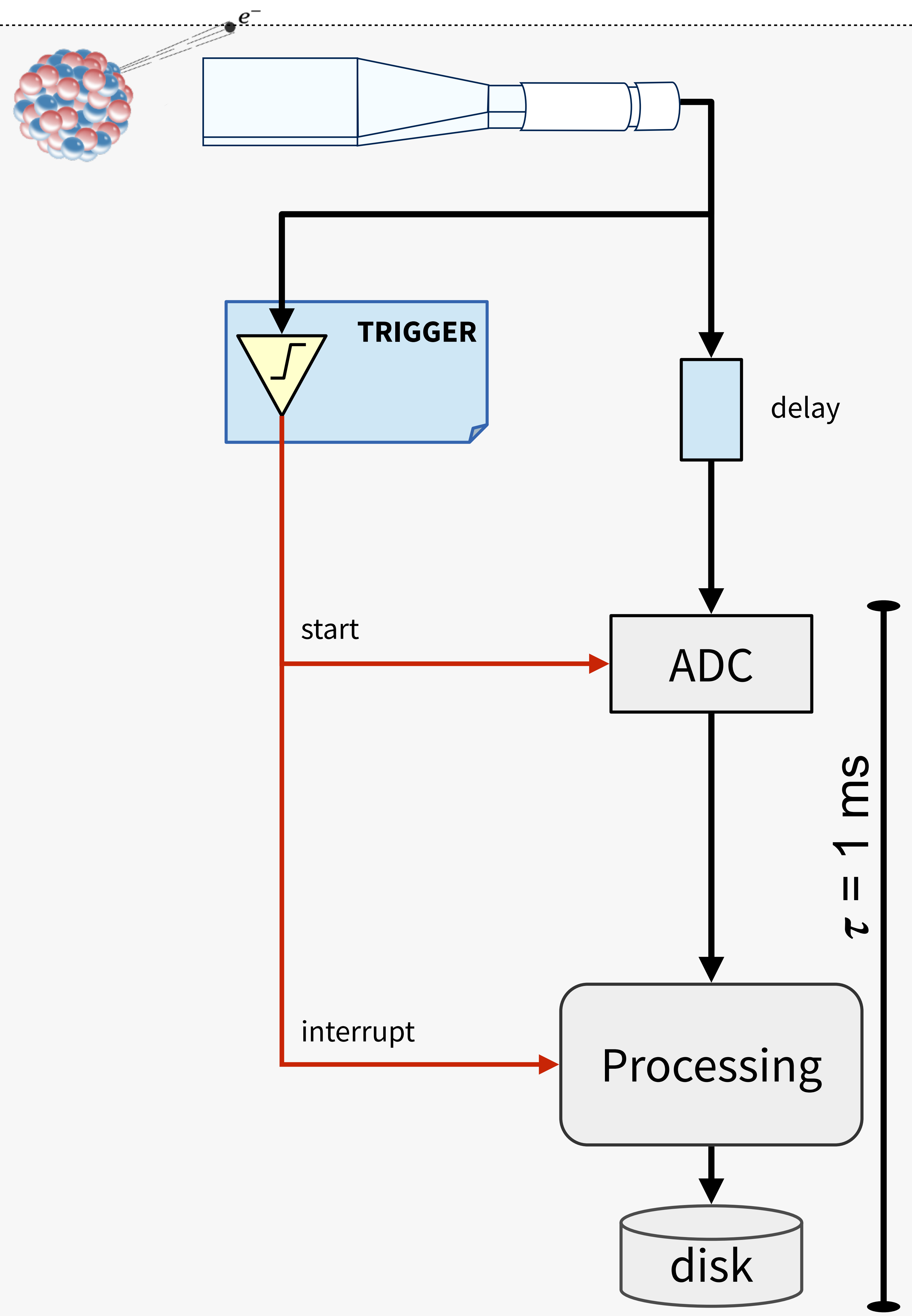
Stochastic process

- Fluctuations in time between events

Let's assume for example

- physics rate f = 1 kHz, i.e. λ = 1 ms
- and, as before, τ = 1 ms





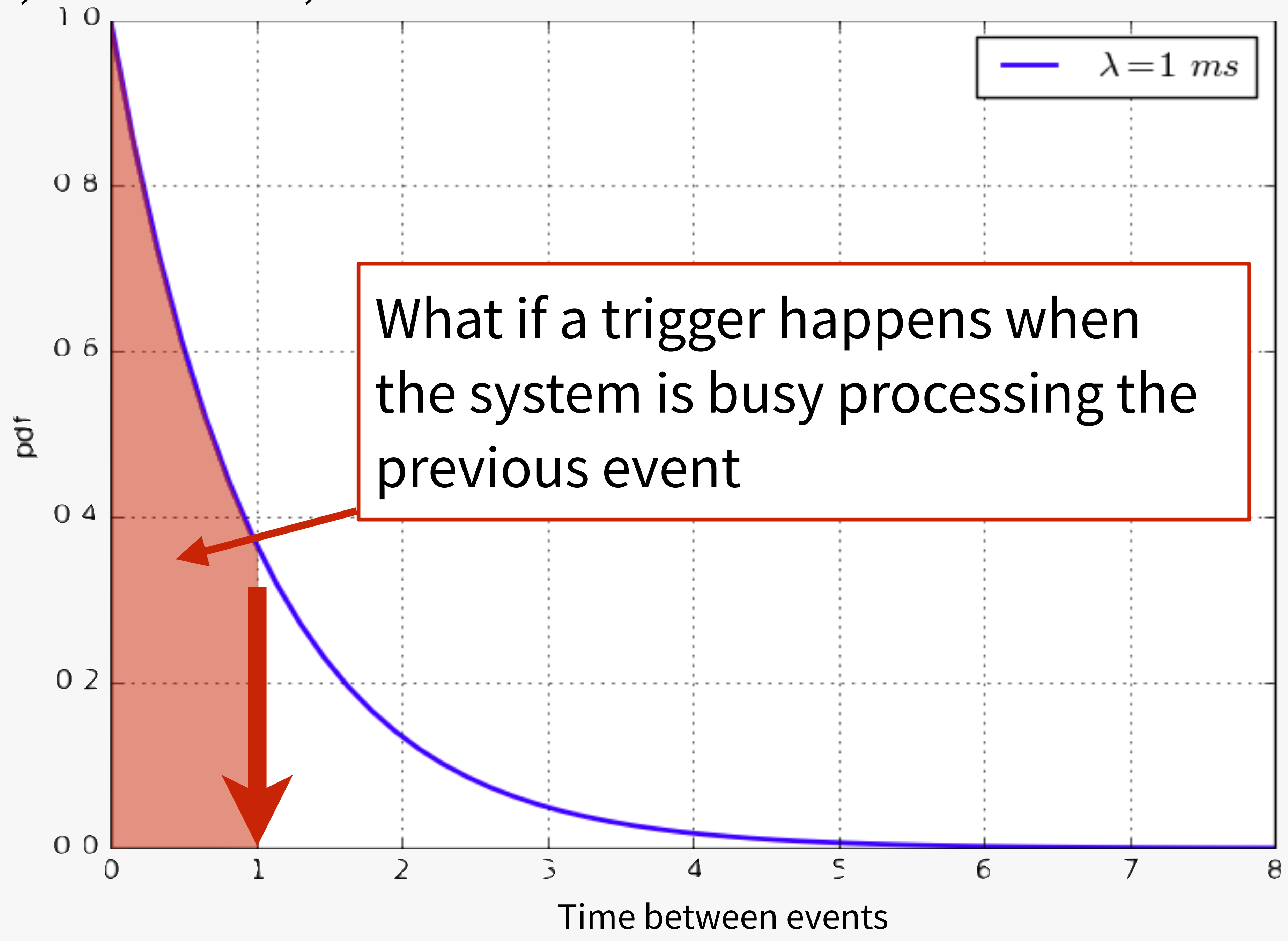
Basic DAQ: “real” trigger

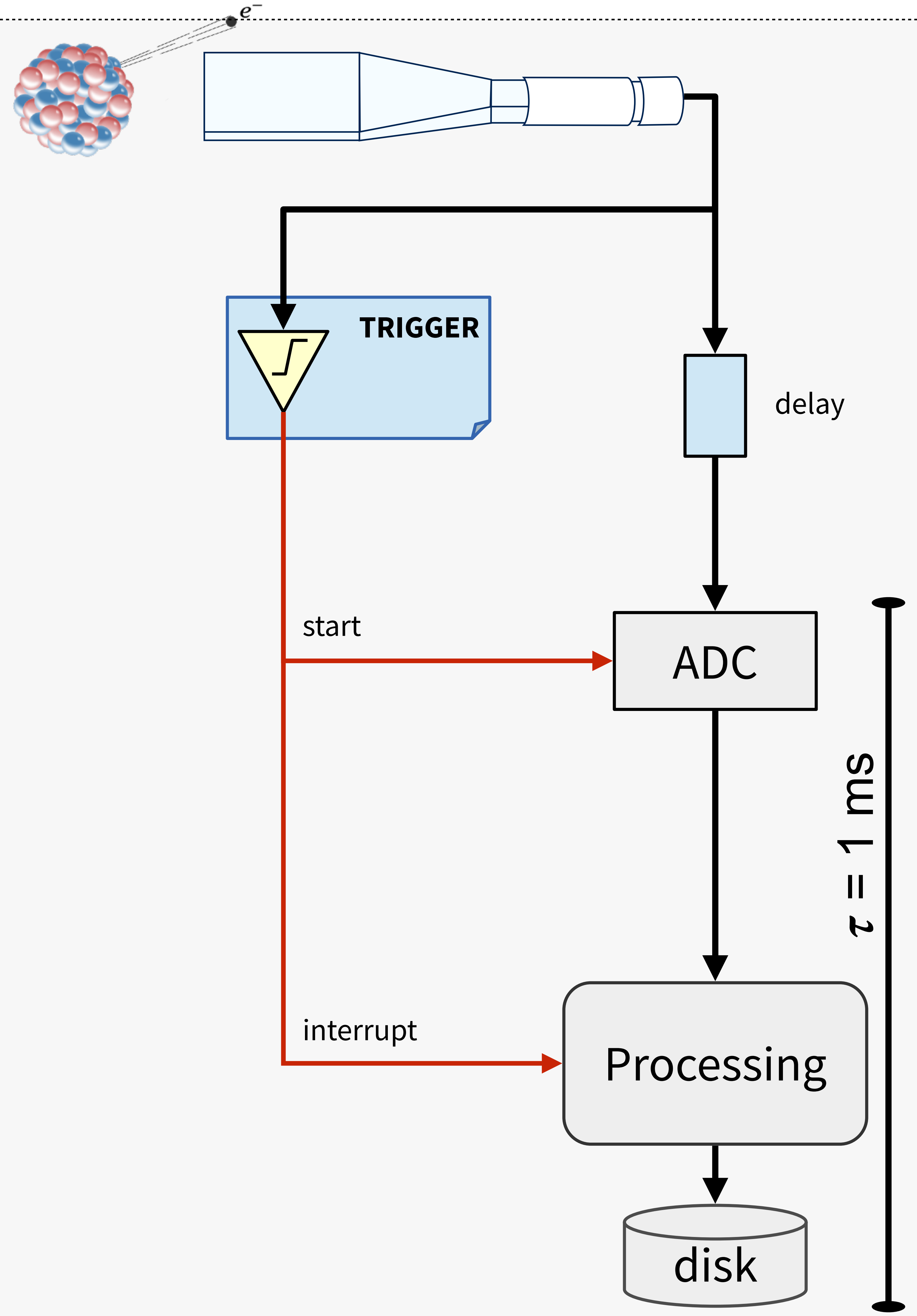
Stochastic process

- Fluctuations in time between events

Let's assume for example

- physics rate $f = 1$ kHz, i.e. $\lambda = 1$ ms
- and, as before, $\tau = 1$ ms

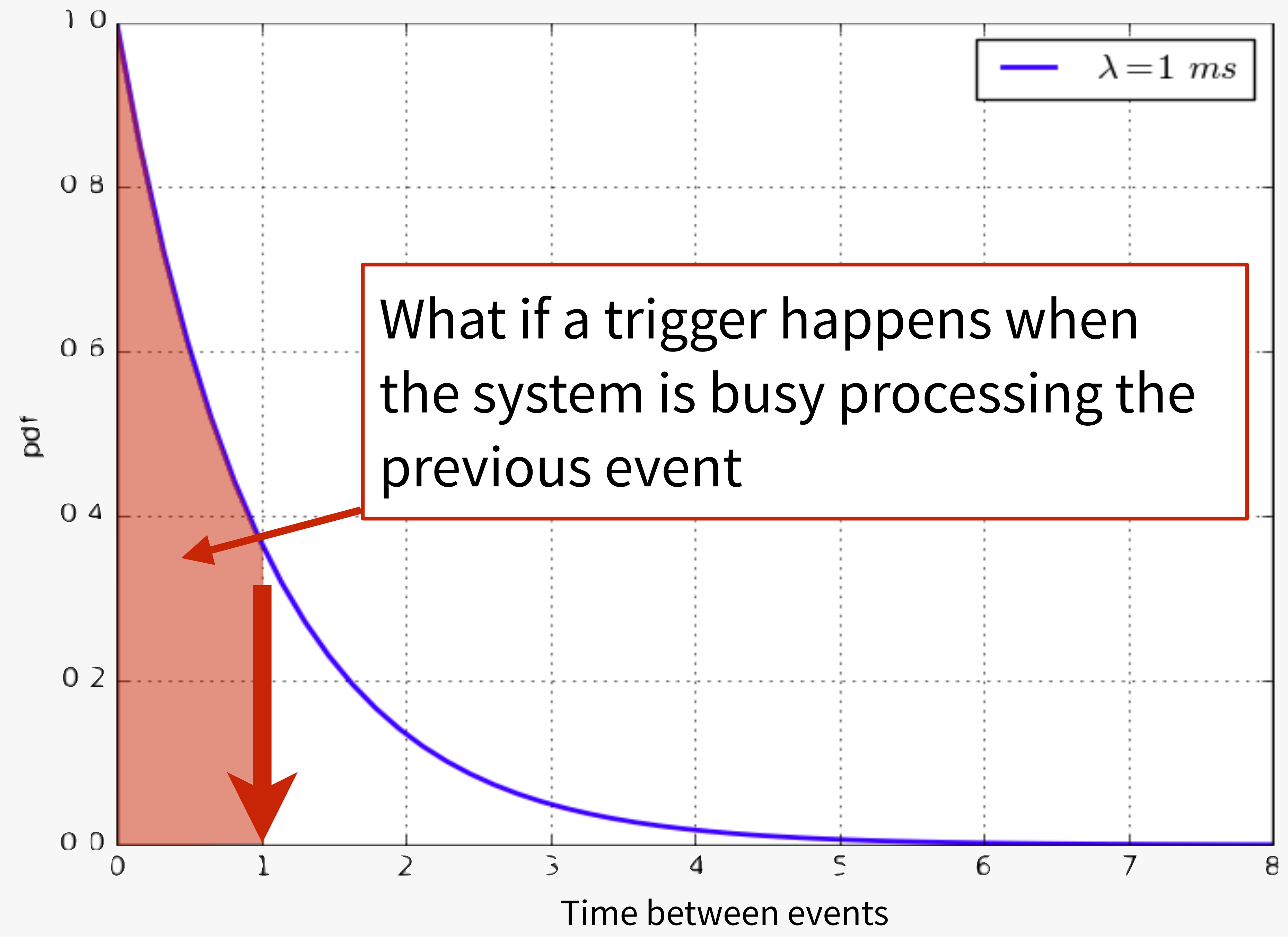


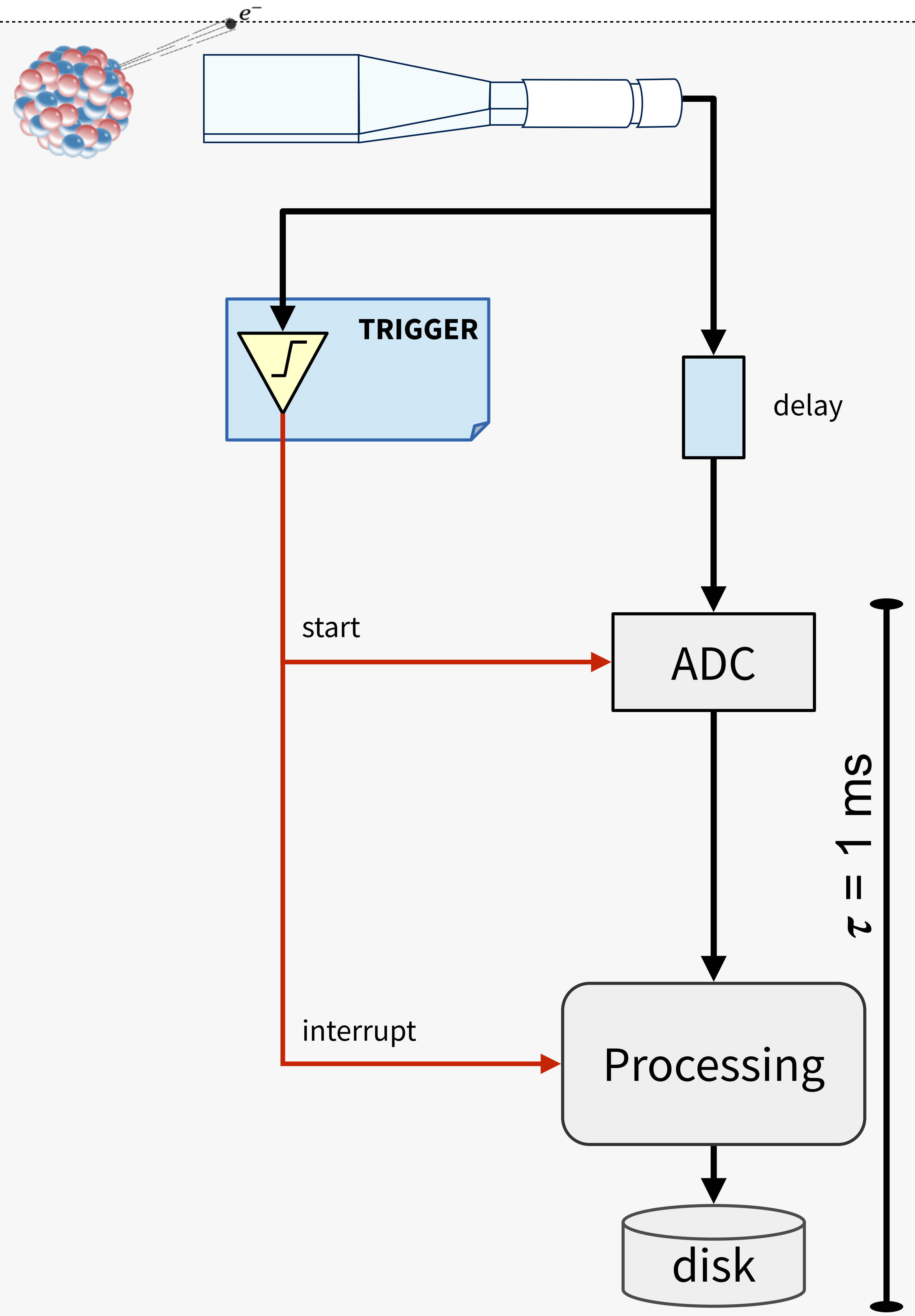


The system is still processing

If a new trigger arrives when the system is still processing the previous event

- The processing of the previous event can be disturbed/corrupted





Thinking…

For stochastic processes, our trigger and daq system needs to be able to:

- Determine if there is an “event” (**trigger**)
- Process and store the data from the event (**daq**)
- Have a feedback mechanism,

  to know if the data processing pipeline

  is free to process a new event:

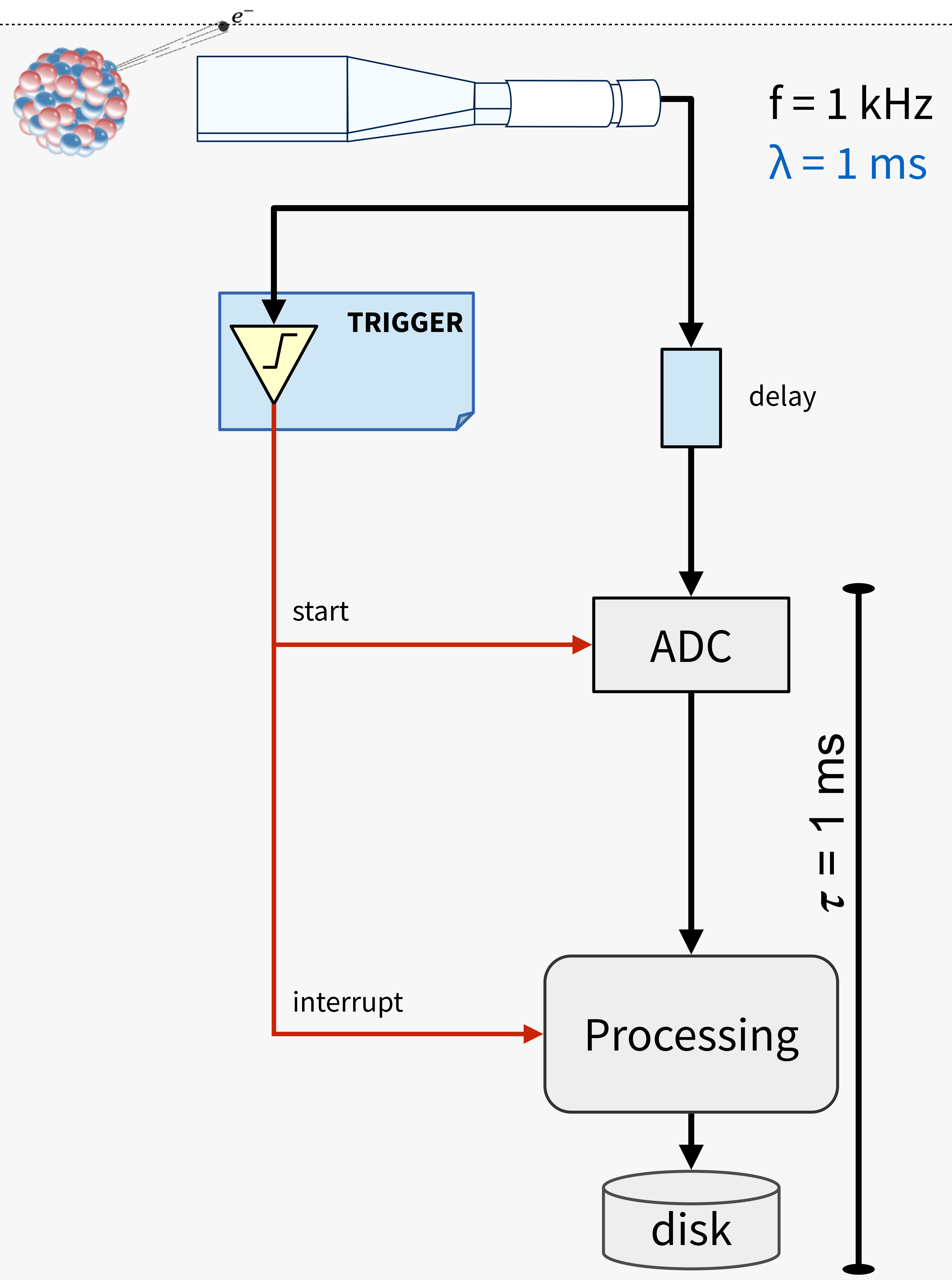
  busy logic

Busy logic

The **busy logic** avoids triggers while the system is busy in processing

A minimal **busy logic** can be implemented with

- an **AND** gate
- a **NOT** gate
- a **flip-flop**
  - bistable circuit that changes state (Q) by signals applied to the control inputs (SET, CLEAR)

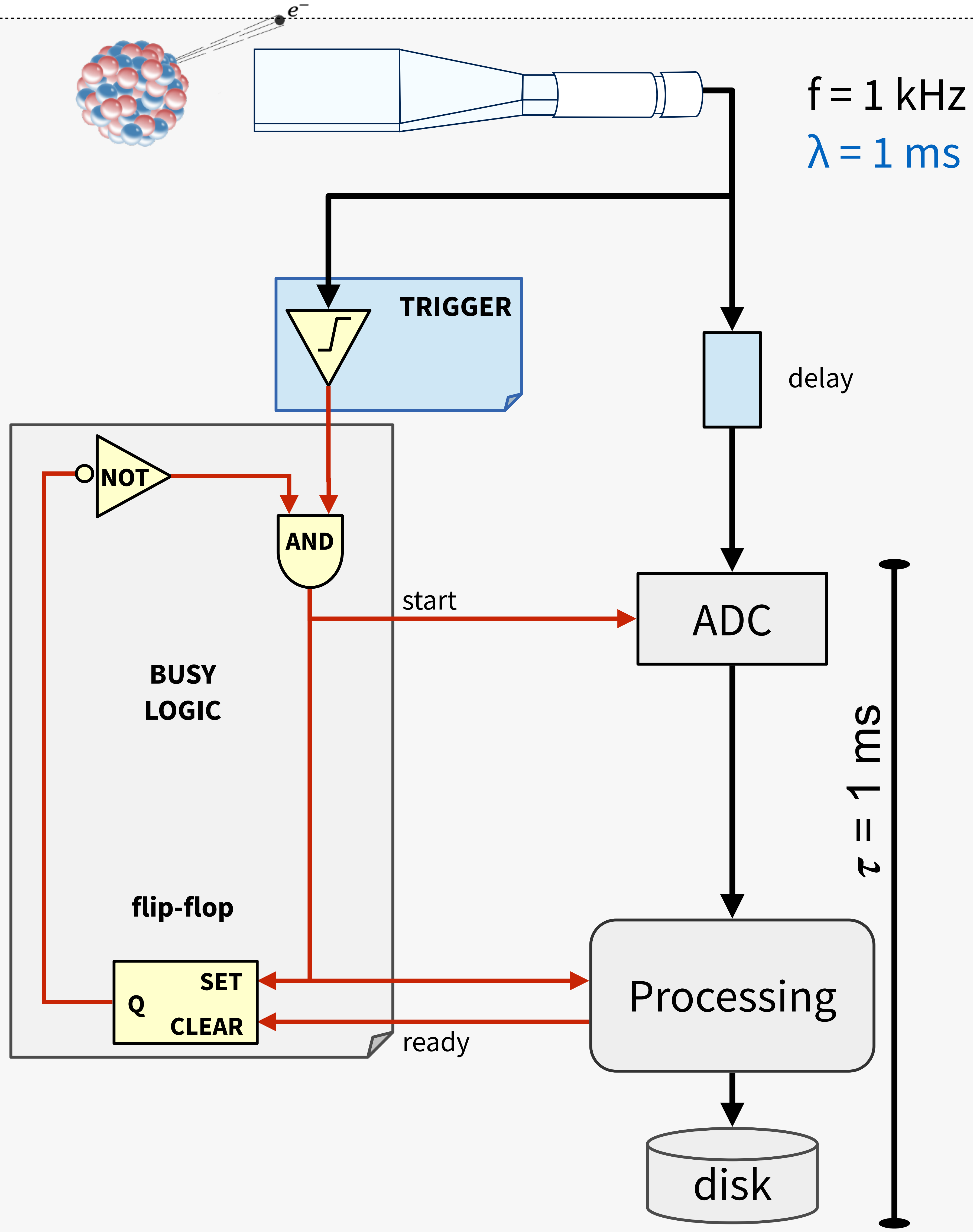


Busy logic

The **busy logic** avoids triggers while the system is busy in processing

A minimal **busy logic** can be implemented with

- an **AND** gate
- a **NOT** gate
- a **flip-flop**
  - bistable circuit that changes state (Q) by signals applied to the control inputs (SET, CLEAR)

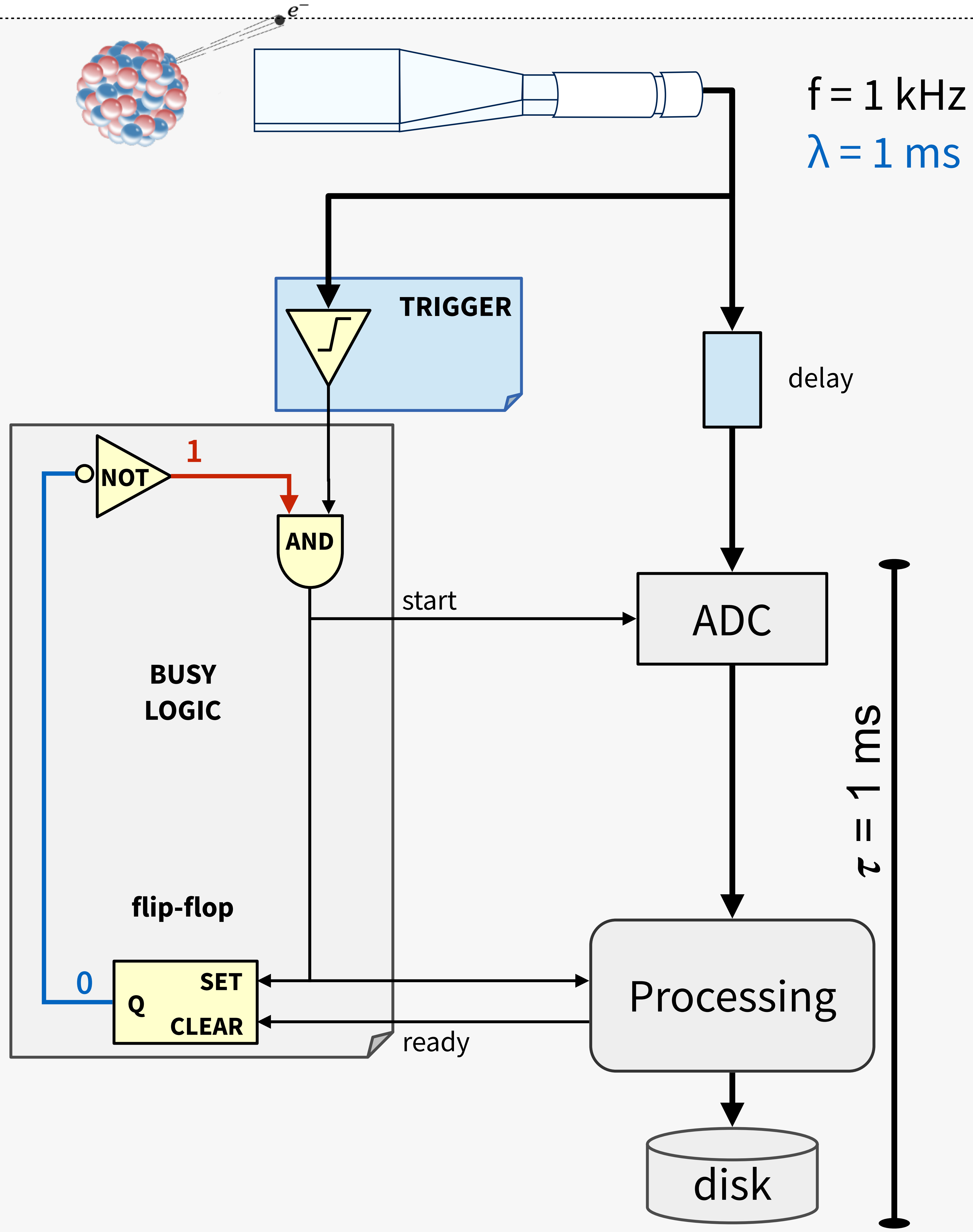


Busy logic

Start of run

- the flip-flop output is down (ground state)
- via the NOT, one of the port of the AND gate is set to up (opened)

i.e. **system ready for new triggers**

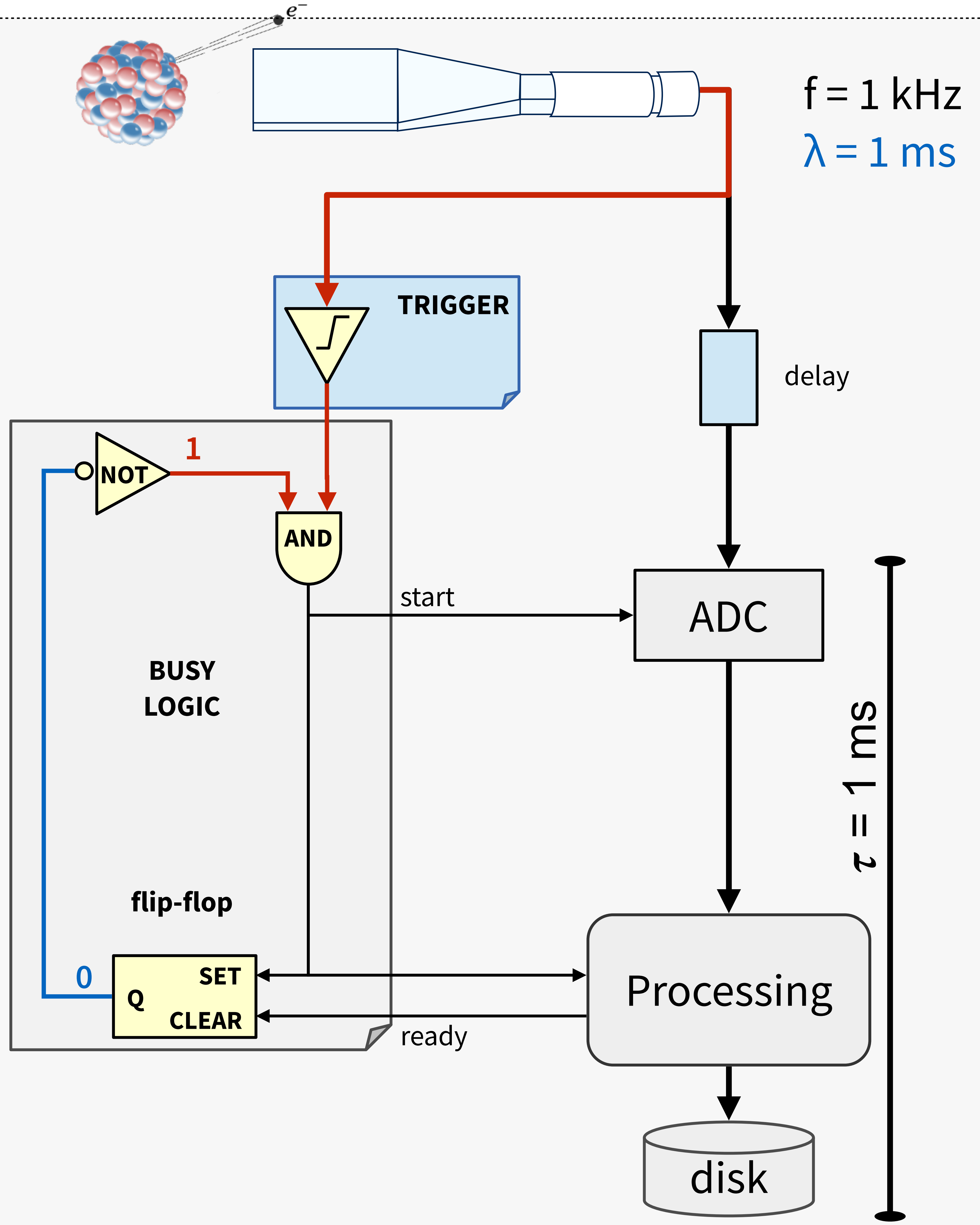


Busy logic

If a trigger arrives, the signal finds the AND gate open, so:

- The ADC is started
- The processing is started
- The flip-flop is flipped
- One of the AND inputs is now steadily down (closed)

Any new trigger is inhibited by the AND gate (busy)

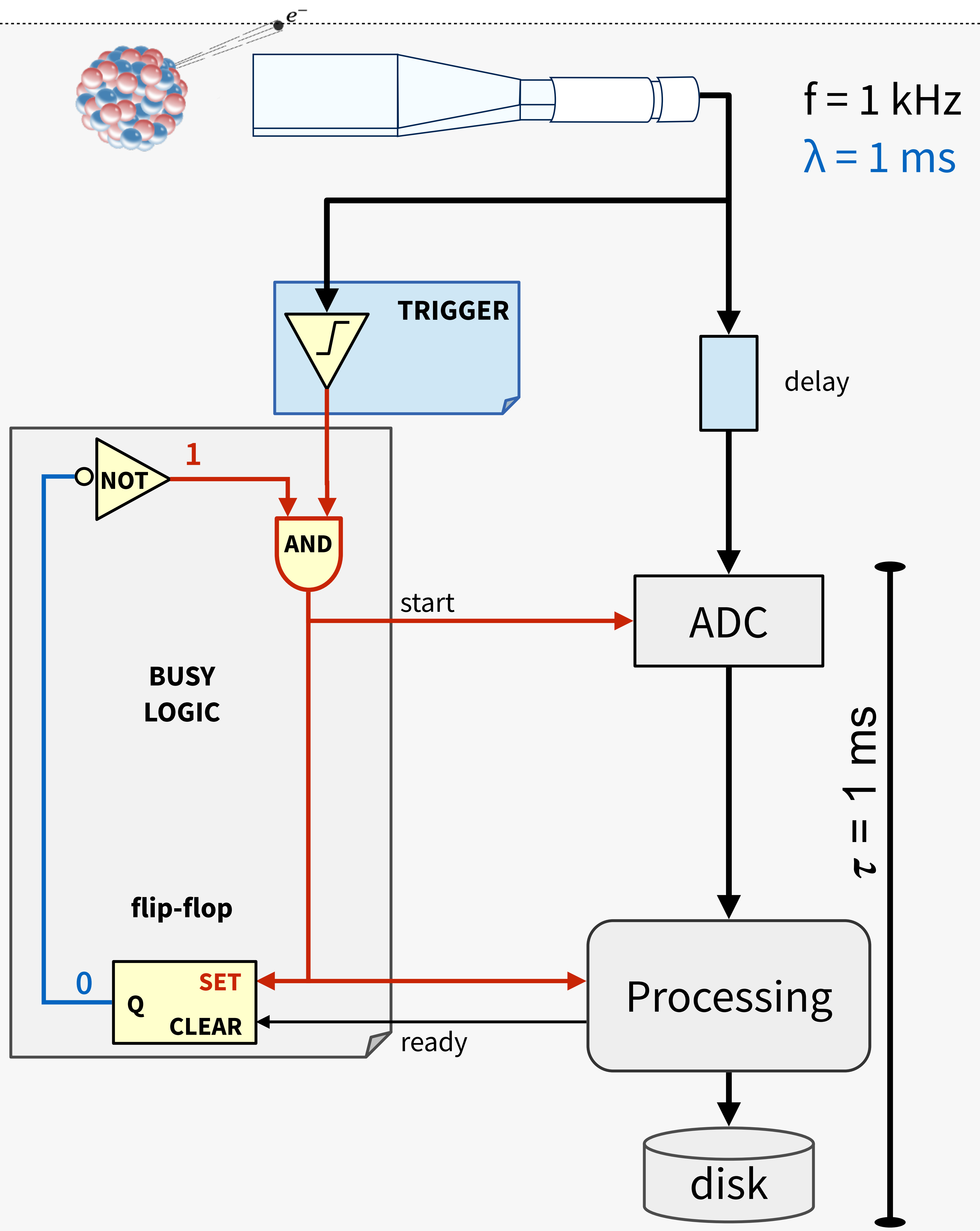


Busy logic

If a trigger arrives, the signal finds the AND gate open, so:

- The ADC is started
- The processing is started
- The flip-flop is flipped
- One of the AND inputs is now steadily down (closed)

Any new trigger is inhibited by the AND gate (busy)

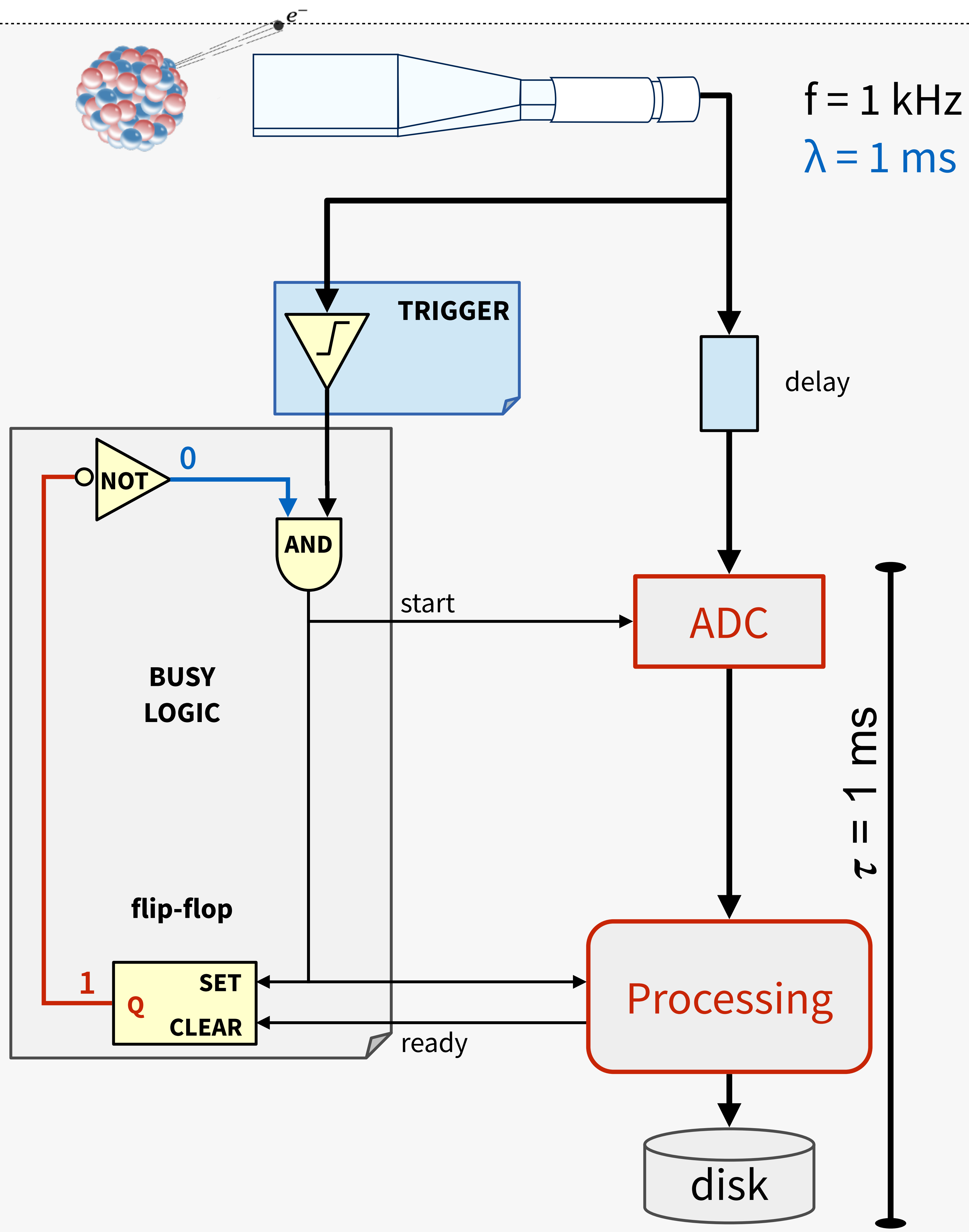


Busy logic

If a trigger arrives, the signal finds the AND gate open, so:

- The ADC is started
- The processing is started
- The flip-flop is flipped
- One of the AND inputs is now steadily down (closed)

Any new trigger is inhibited by the AND gate (busy)

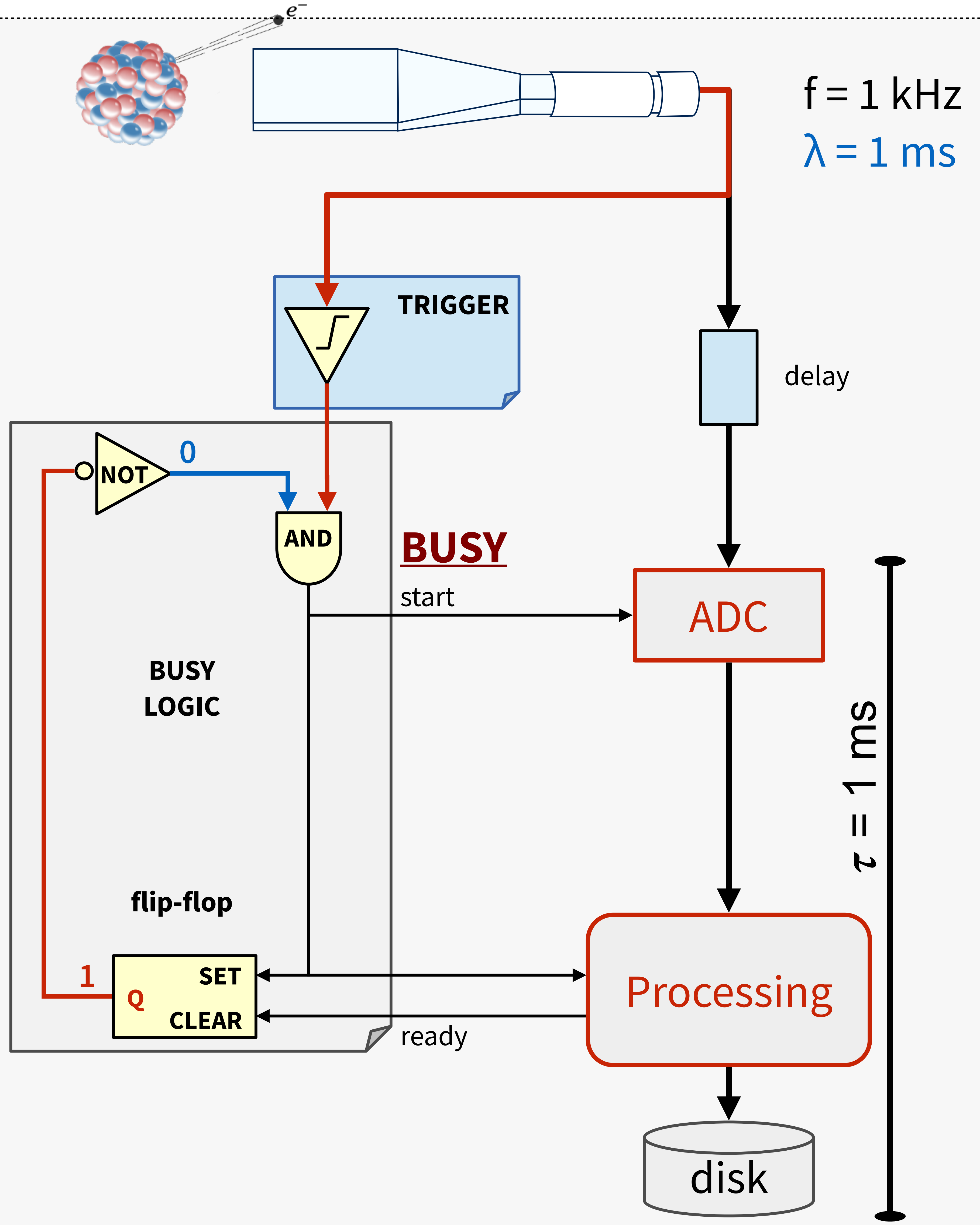


Busy logic

If a trigger arrives, the signal finds the AND gate open, so:

- The ADC is started
- The processing is started
- The flip-flop is flipped
- One of the AND inputs is now steadily down (closed)

Any new trigger is inhibited by the AND gate (busy)



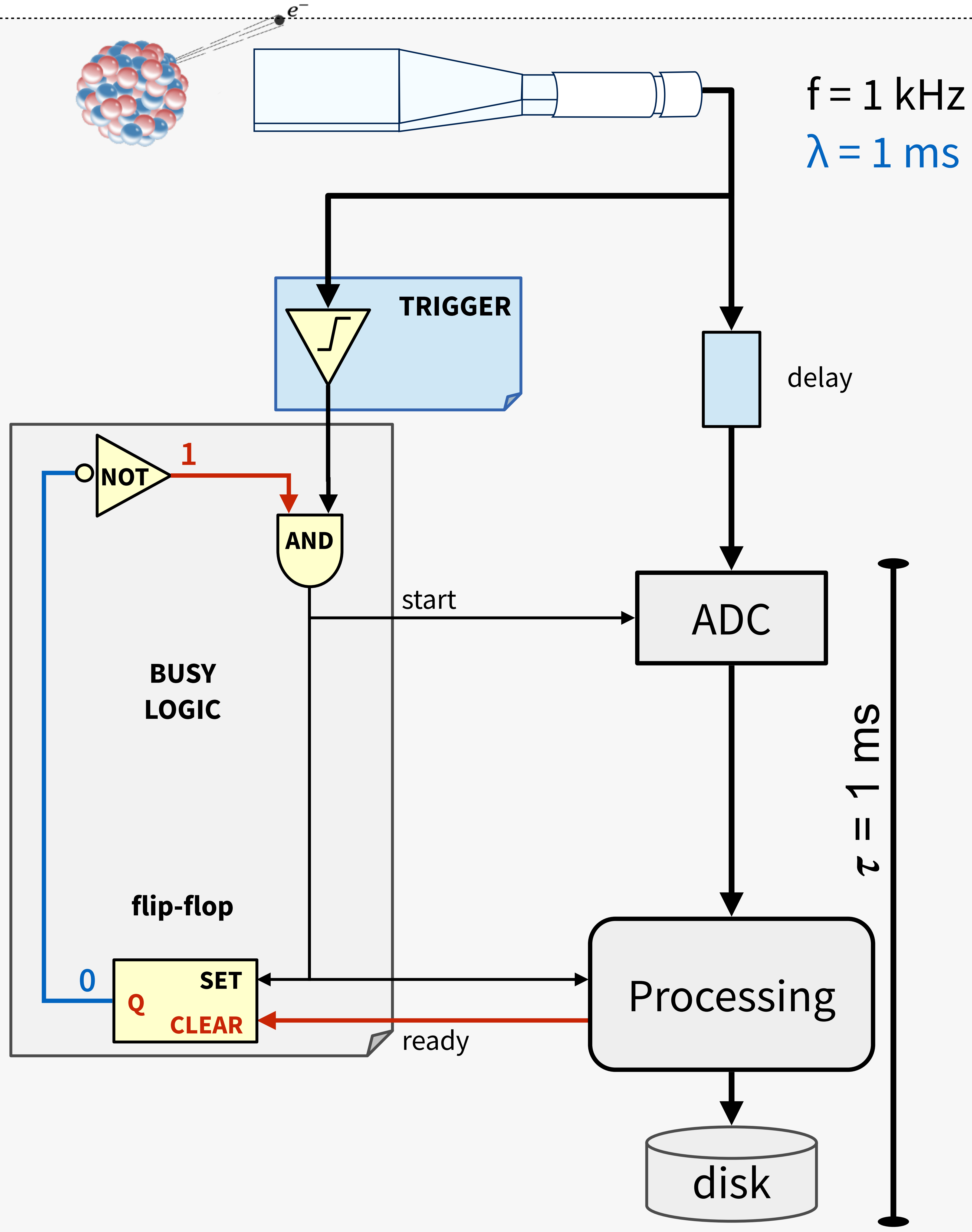
Busy logic

At the end of processing a ready signal is sent to the flip-flop

- The flip-flop flips again
- The gate is now opened
- The system is ready to accept a new trigger

i.e. busy logic avoids triggers while daq is busy in processing

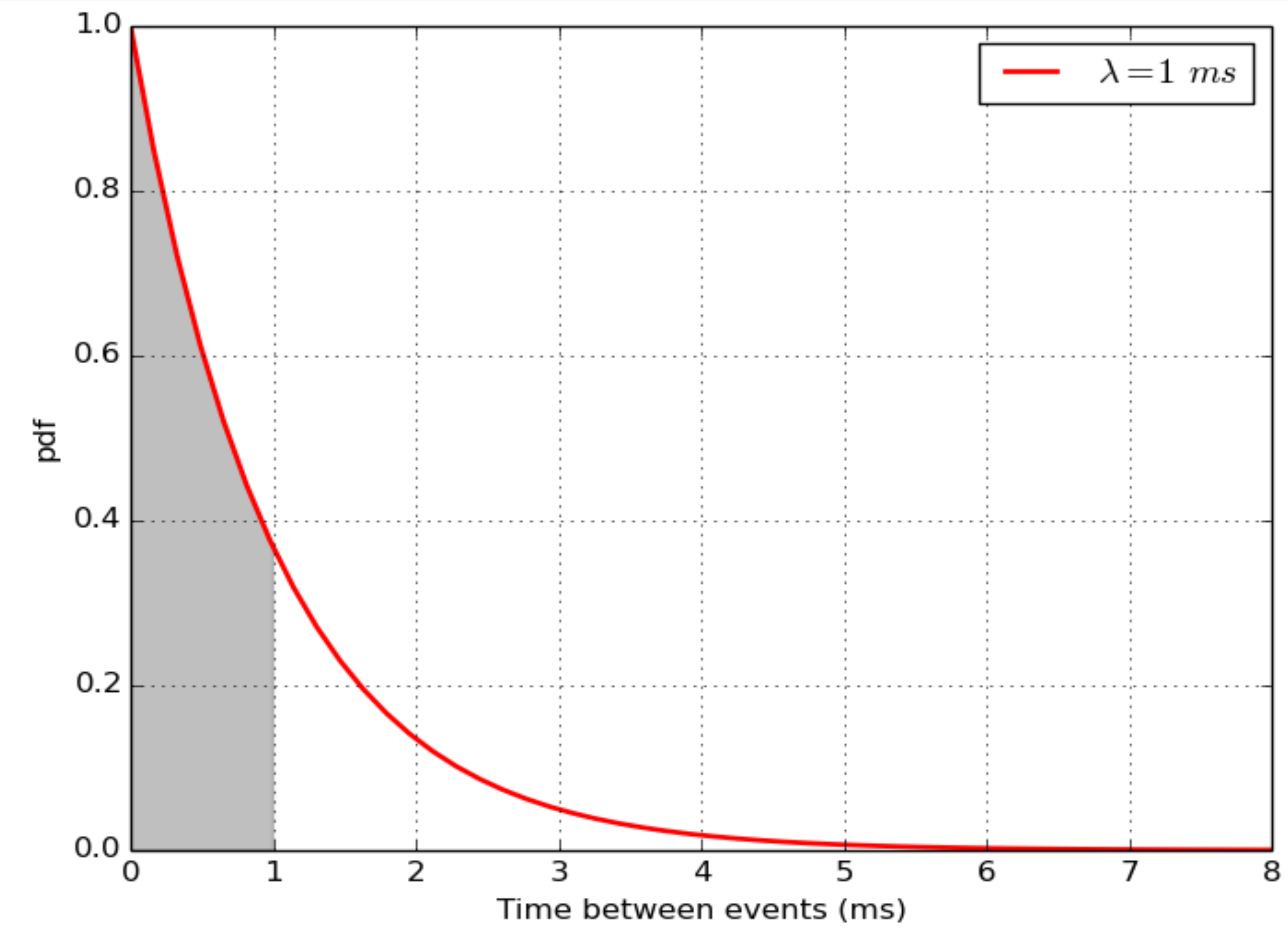
- New triggers do not interfere w/ previous data



Deadtime and efficiency

So the busy logic protects electronics from unwanted triggers

- New signals are accepted only when the system in ready to process them



What (average) DAQ rate can be achieved now?

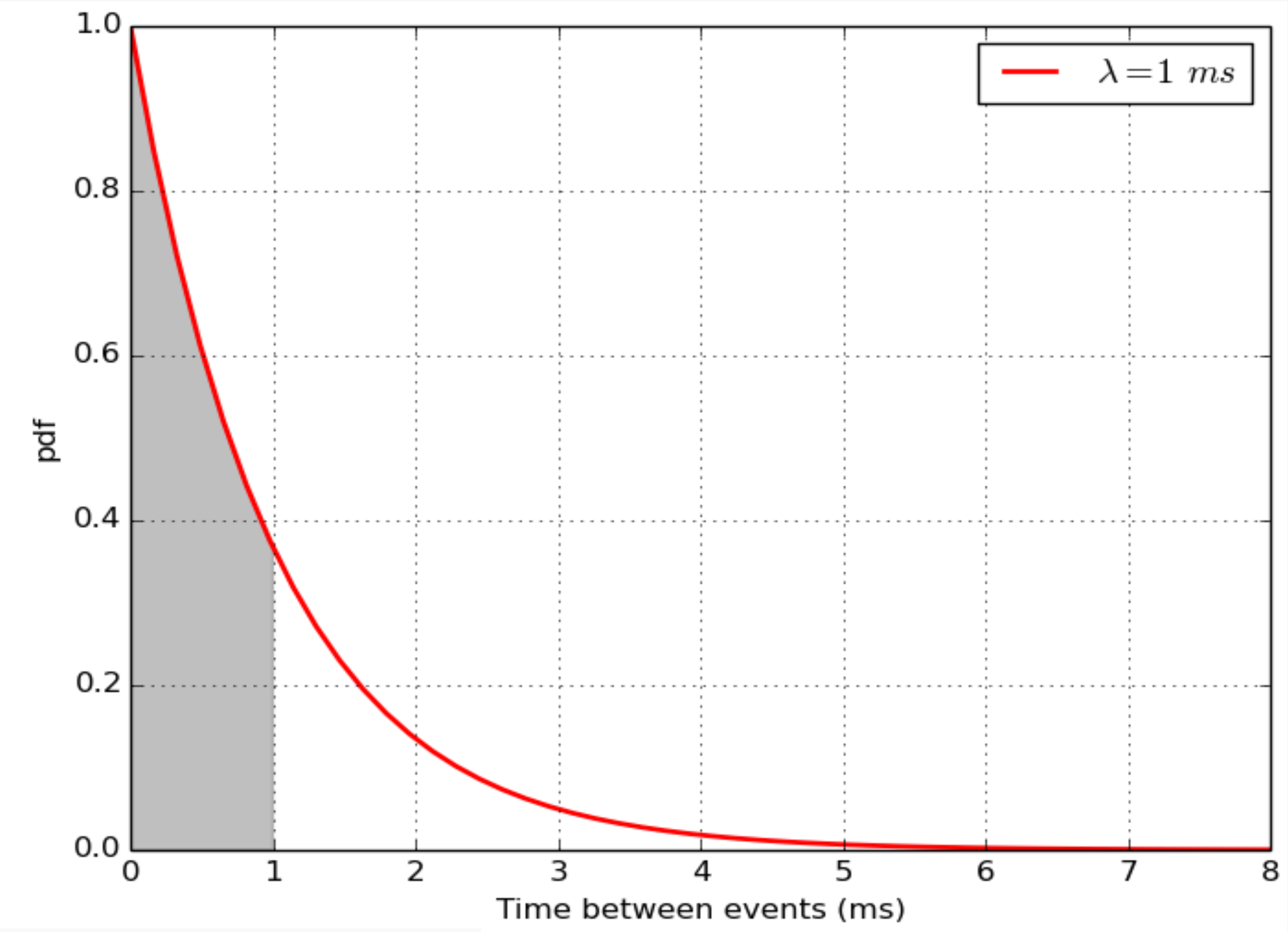
- How much we lose with the busy logic?

Reminder: with periodic triggers and $\tau = 1$ ms the limit was ***1 kHz***

Deadtime and efficiency

Definitions

- f: average rate of physics (input)
- ν: average rate of DAQ (output)
- τ: **deadtime**, needed to process an event, without being able to handle other triggers
- probabilities: P[busy] = $\nu\tau$; P[free] = 1 - $\nu\tau$



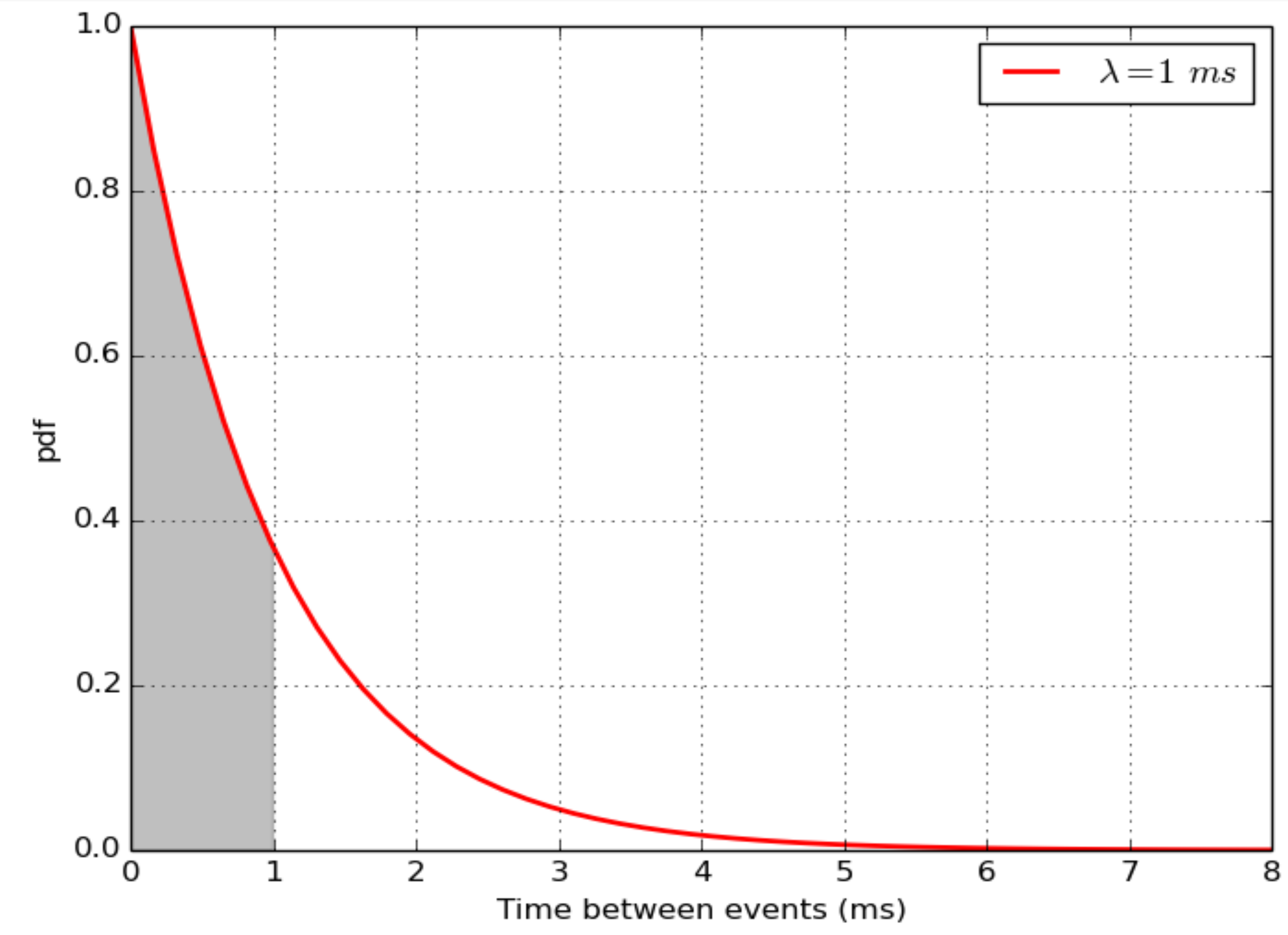
Therefore:

$$\nu = fP[free] \Rightarrow \nu = f(1-\nu\tau) \Rightarrow \nu = \frac{f}{1+f\tau}$$

Deadtime and efficiency

Definitions

- f: average rate of physics (input)
- ν: average rate of DAQ (output)
- τ : **deadtime**, needed to process an event, without being able to handle other triggers
- probabilities: P[busy] = $\nu\tau$; P[free] = 1 - $\nu\tau$



Therefore:

$$\nu = fP[free] \Rightarrow \nu = f(1-\nu\tau) \Rightarrow \nu = \frac{f}{1+f\tau}$$

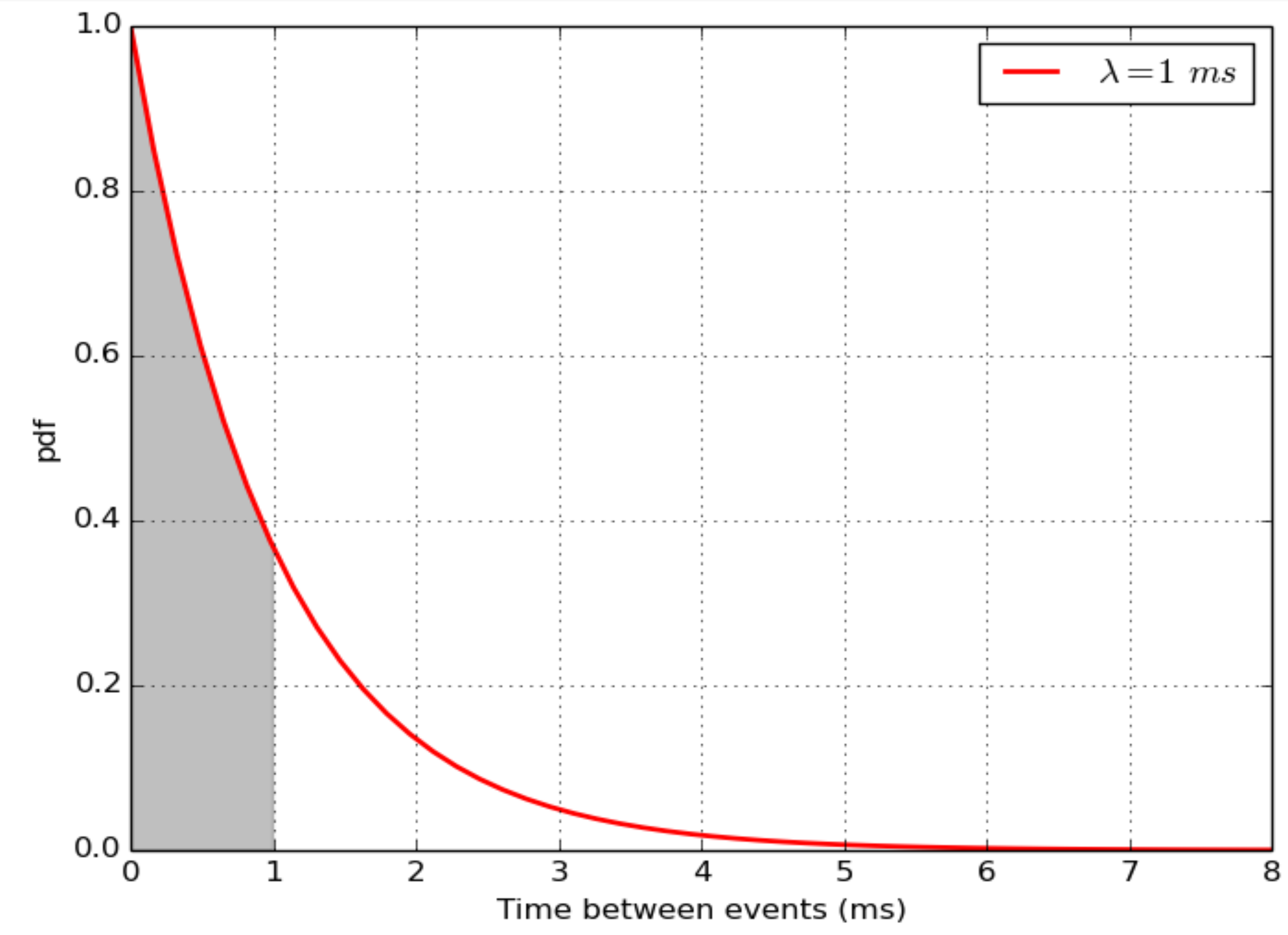
Deadtime and efficiency

Definitions

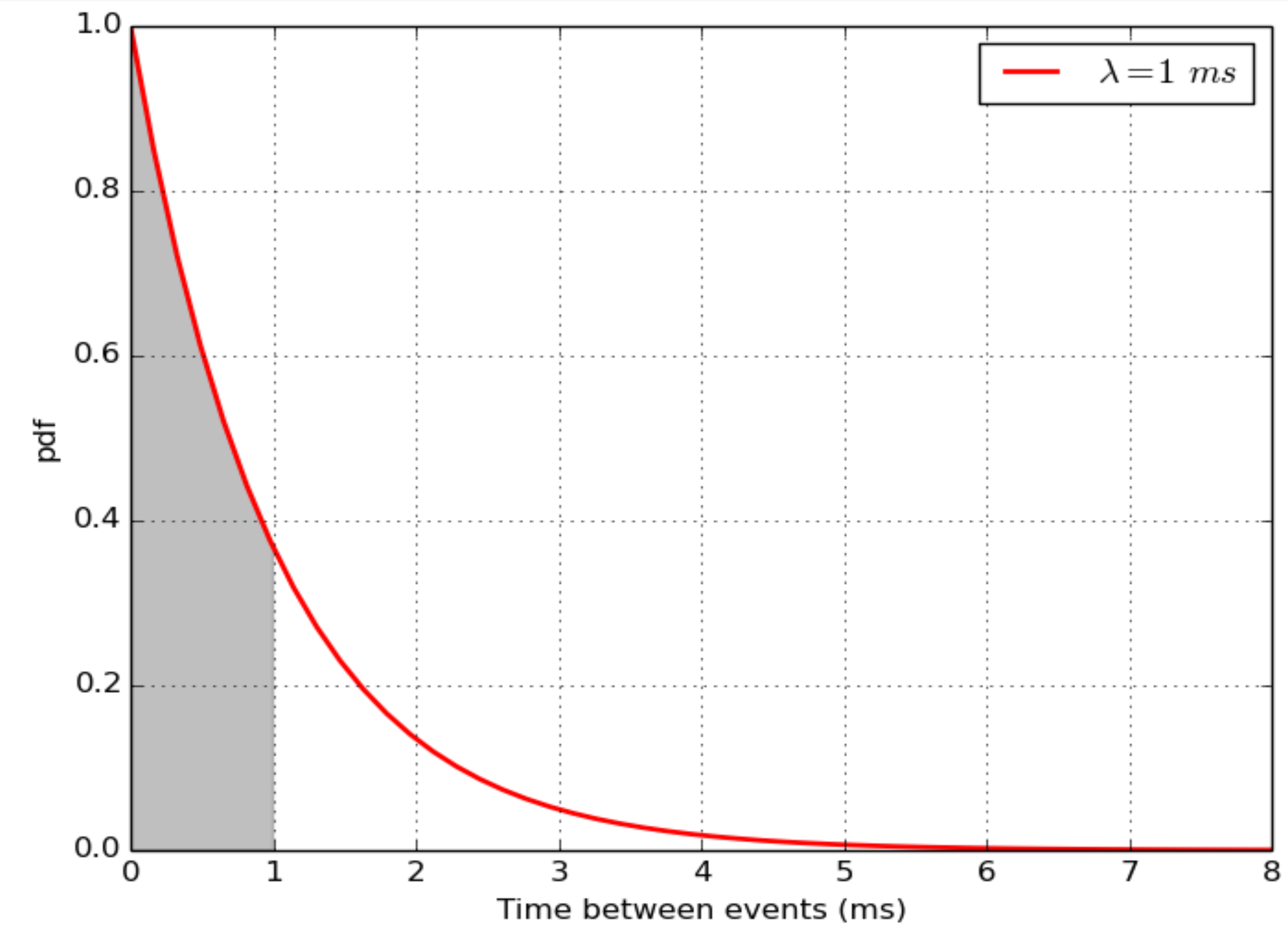
- f : average rate of physics (input)
- ν: average rate of DAQ (output)
- τ : **deadtime**, needed to process an event, without being able to handle other triggers
- probabilities: P[busy] = $\nu\tau$; P[free] = $1 - \nu\tau$

Therefore:

$$\nu = fP[free] \Rightarrow \nu = f(1 - \nu\tau) \Rightarrow \nu = \frac{f}{1 + f\tau}$$



Deadtime and efficiency



Definitions

- f: average rate of physics (input)
- ν: average rate of DAQ (output)
- τ : **deadtime**, needed to process an event, without being able to handle other triggers
- probabilities: P[busy] = $\nu\tau$; P[free] = $1 - \nu\tau$

Therefore:

$$\nu = fP[free] \Rightarrow \nu = f(1 - \nu\tau) \Rightarrow \nu = \frac{f}{1 + f\tau}$$

Deadtime and efficiency

Due to stochastic fluctuations

- DAQ rate always < physics rate
- Efficiency always < 100%

$$\nu = \frac{f}{1 + f\tau} < f$$

$$\epsilon = \frac{N_{saved}}{N_{tot}} = \frac{1}{1 + f\tau} < 100\,\%$$

So, in our specific example

- Physics rate 1 kHz
- Deadtime 1 ms

$$f = 1\ kHz \qquad \tau = 1\ ms \quad \Rightarrow \quad \nu = 500\ Hz \qquad \epsilon = 50\,\%$$

Deadtime and efficiency

Due to stochastic fluctuations

- DAQ rate always < physics rate
- Efficiency always < 100%

$$\nu = \frac{f}{1 + f\tau} < f$$

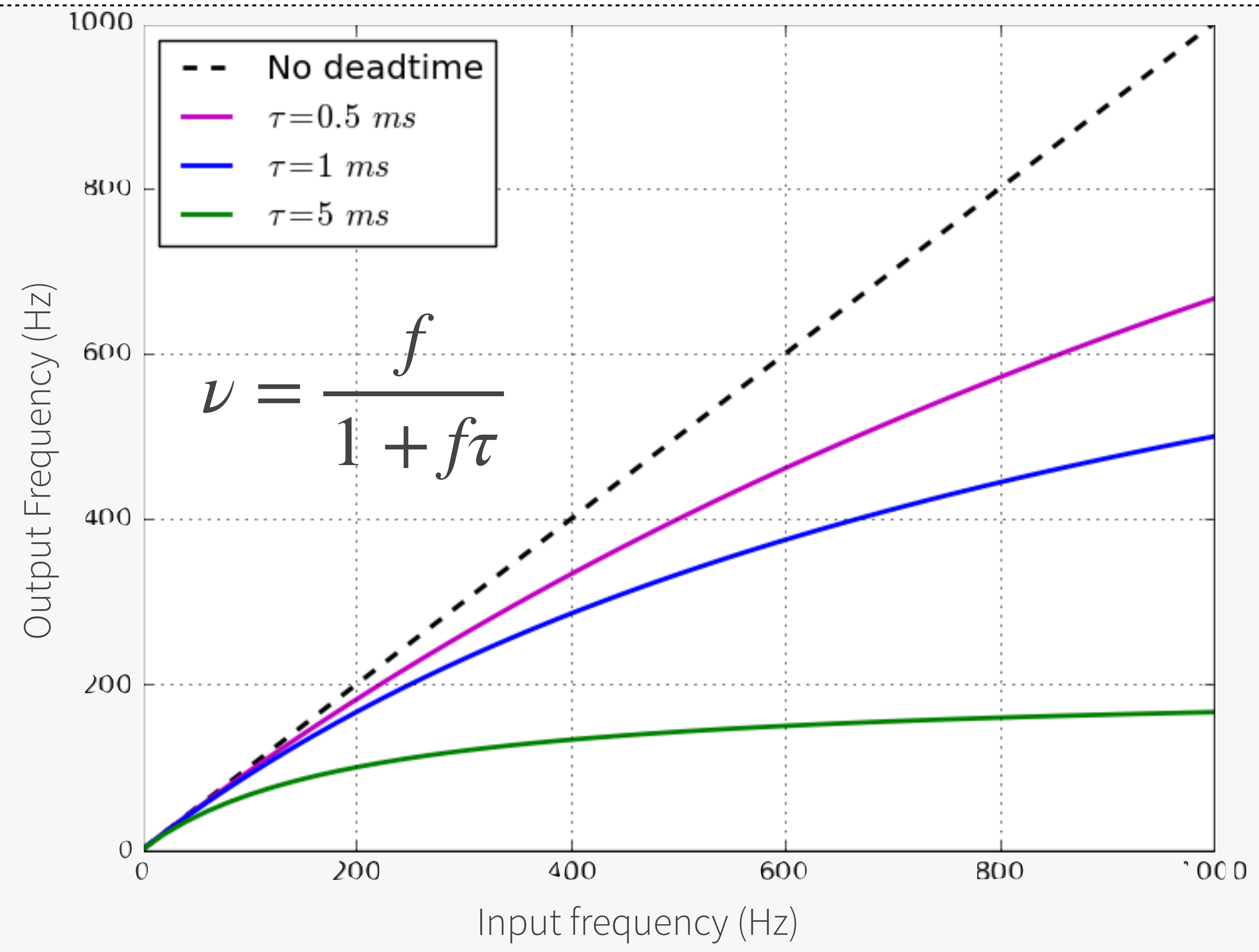
$$\epsilon = \frac{N_{saved}}{N_{tot}} = \frac{1}{1 + f\tau} < 100\,\%$$

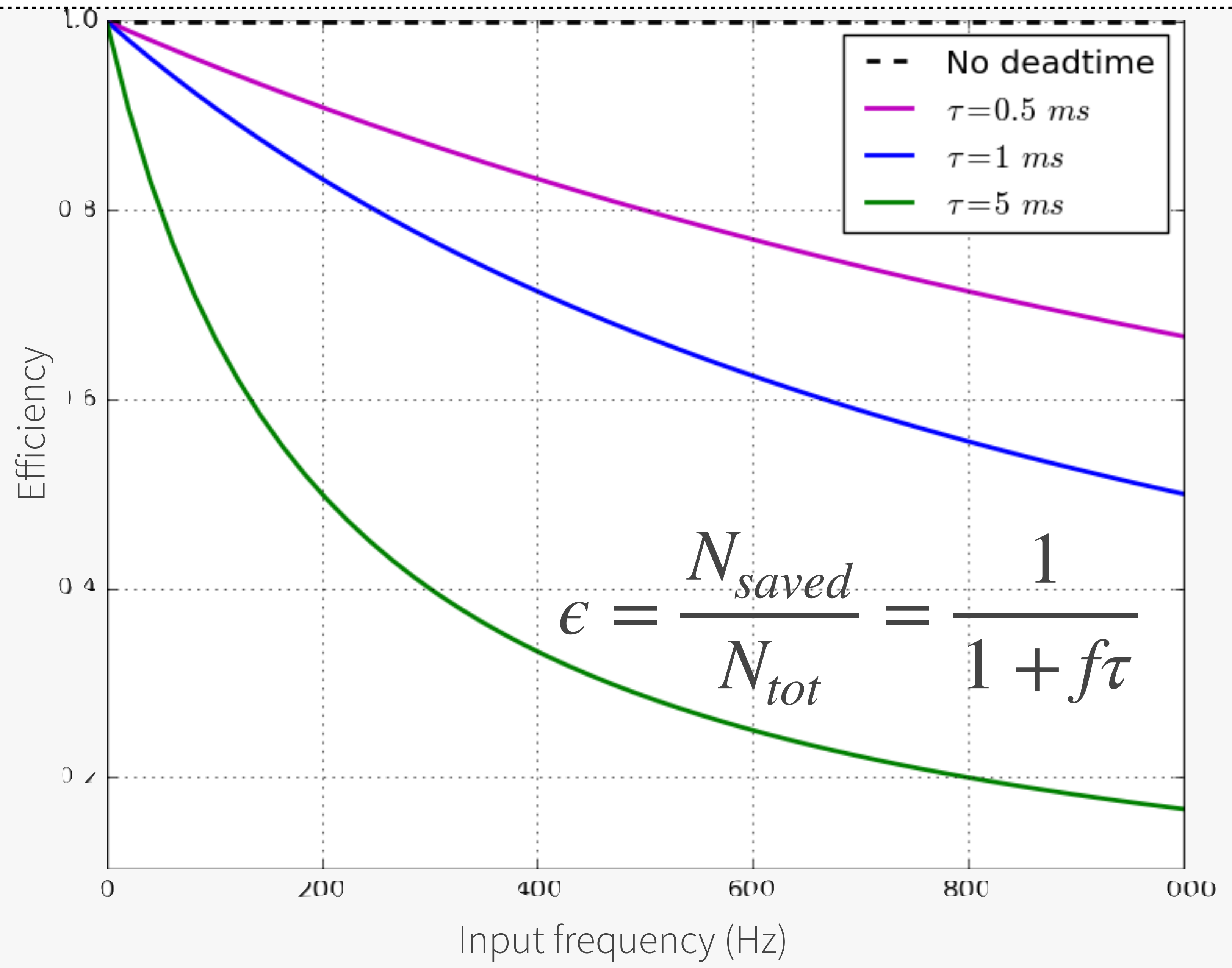
So, in our specific example

- Physics rate 1 kHz
- Deadtime 1 ms

$$f = 1\ kHz \quad \tau = 1\ ms \quad \Rightarrow \quad \nu = 500\ Hz \quad \epsilon = 50\,\%$$

Deadtime and efficiency





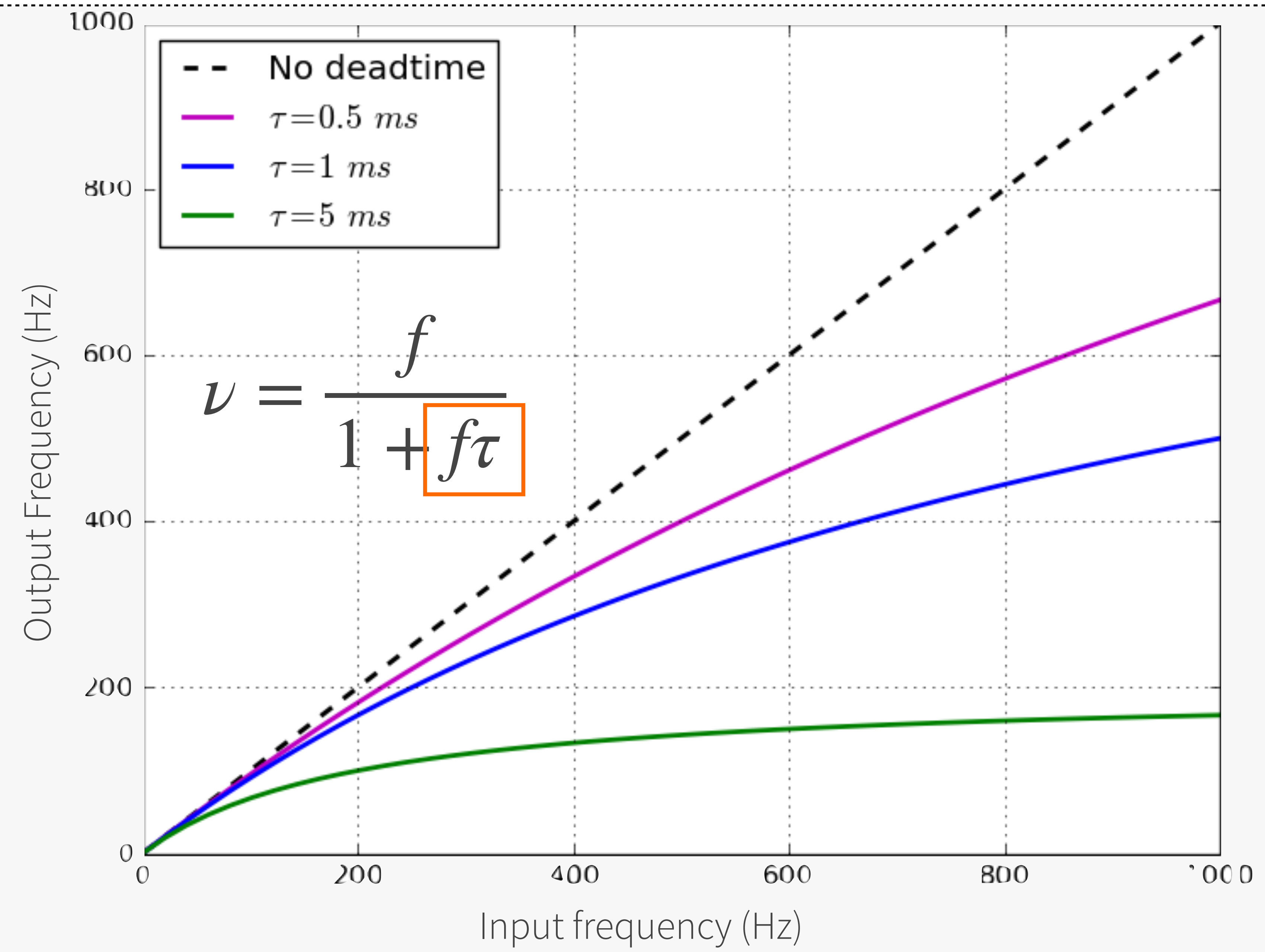
In order to obtain $\epsilon \simeq$ 100% (i.e.: $\nu \simeq \tau$) $\rightarrow f\tau \ll 1 \rightarrow \tau \ll \lambda$

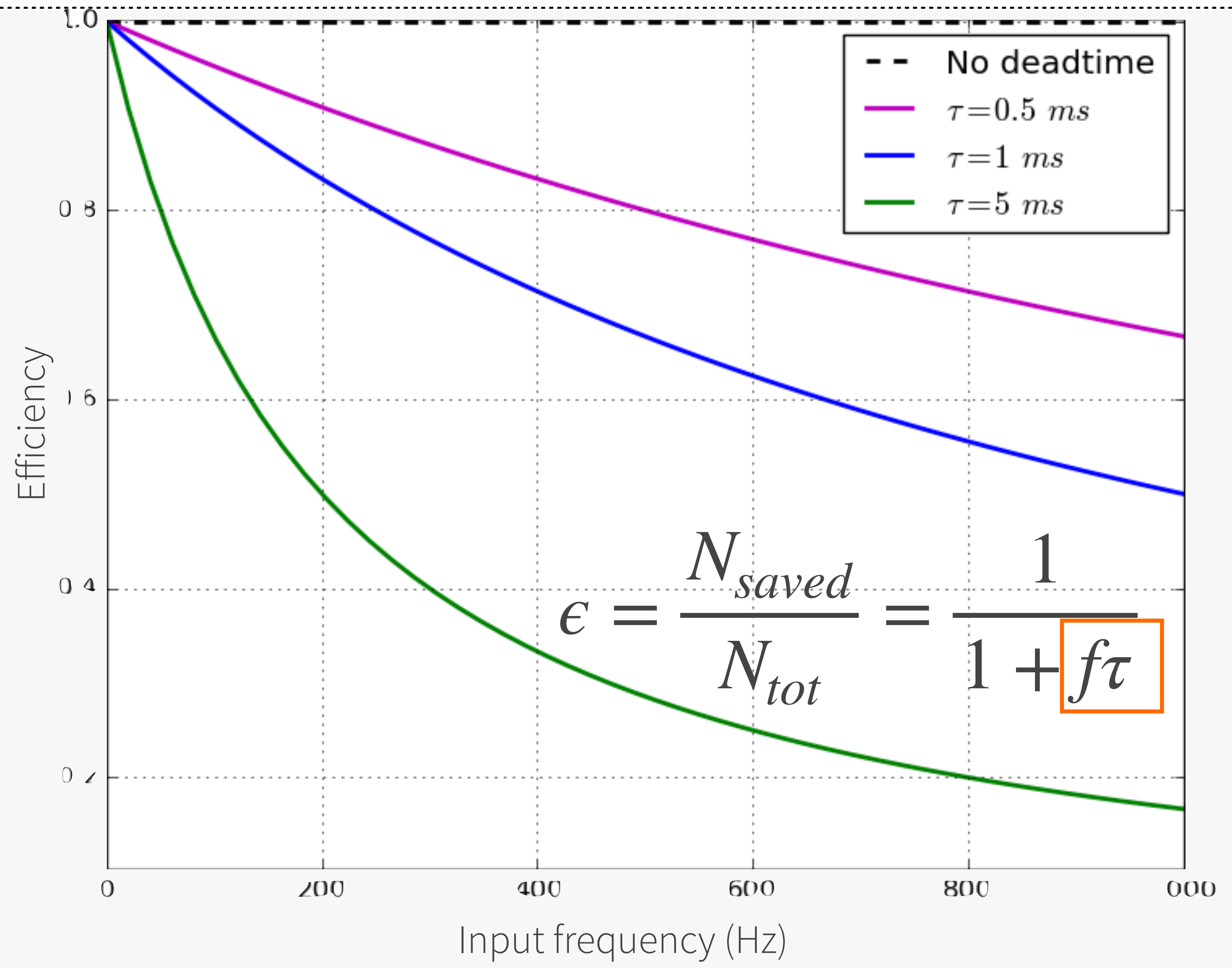
- E.g.: $\epsilon \simeq$ 99% for f = 1 kHz $\rightarrow$ τ < 0.01 ms $\rightarrow$ $1/\tau$ > 100 kHz
- To cope with the input signal fluctuations,

  we have to **over-design** our DAQ system by a **factor 100**!

How can we mitigate this effect?

Deadtime and efficiency



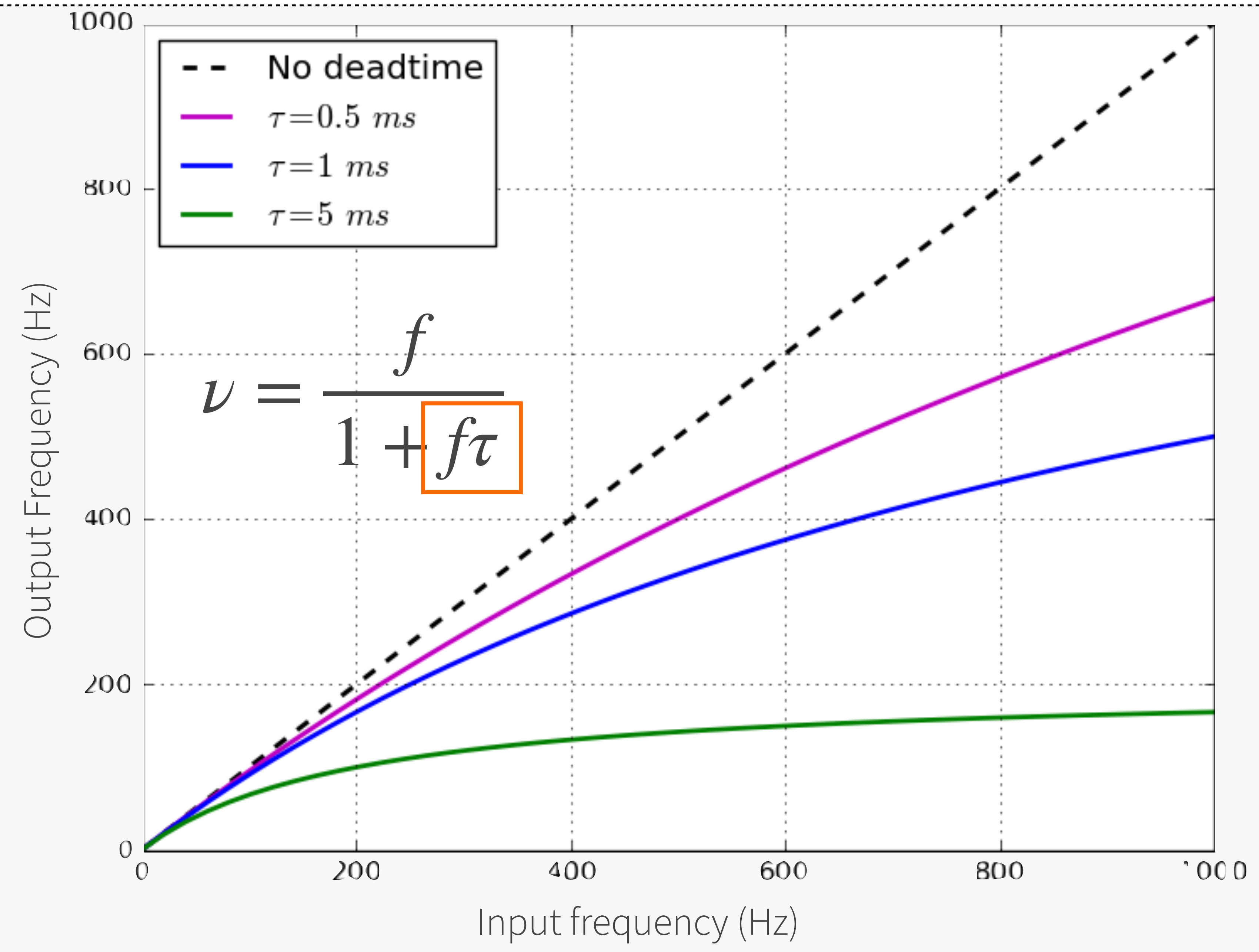


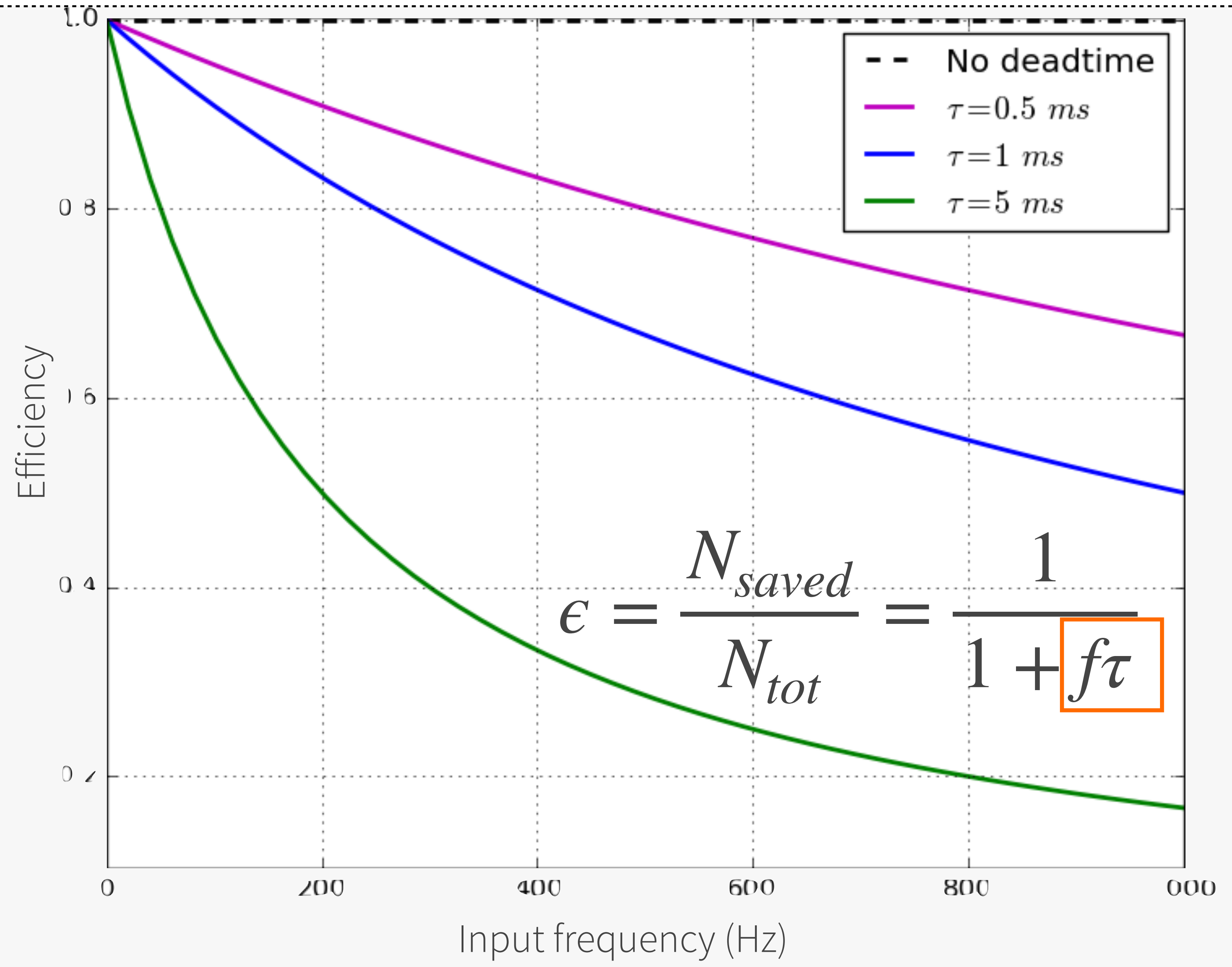
In order to obtain $\epsilon \simeq$ 100% (i.e.: $\nu \simeq \tau$) $\rightarrow f\tau \ll 1 \rightarrow \tau \ll \lambda$

- E.g.: $\epsilon \simeq$ 99% for f = 1 kHz $\rightarrow$ τ < 0.01 ms $\rightarrow$ $1/\tau$ > 100 kHz
- To cope with the input signal fluctuations,
  we have to **over-design** our DAQ system by a **factor 100**!

How can we mitigate this effect?

Deadtime and efficiency





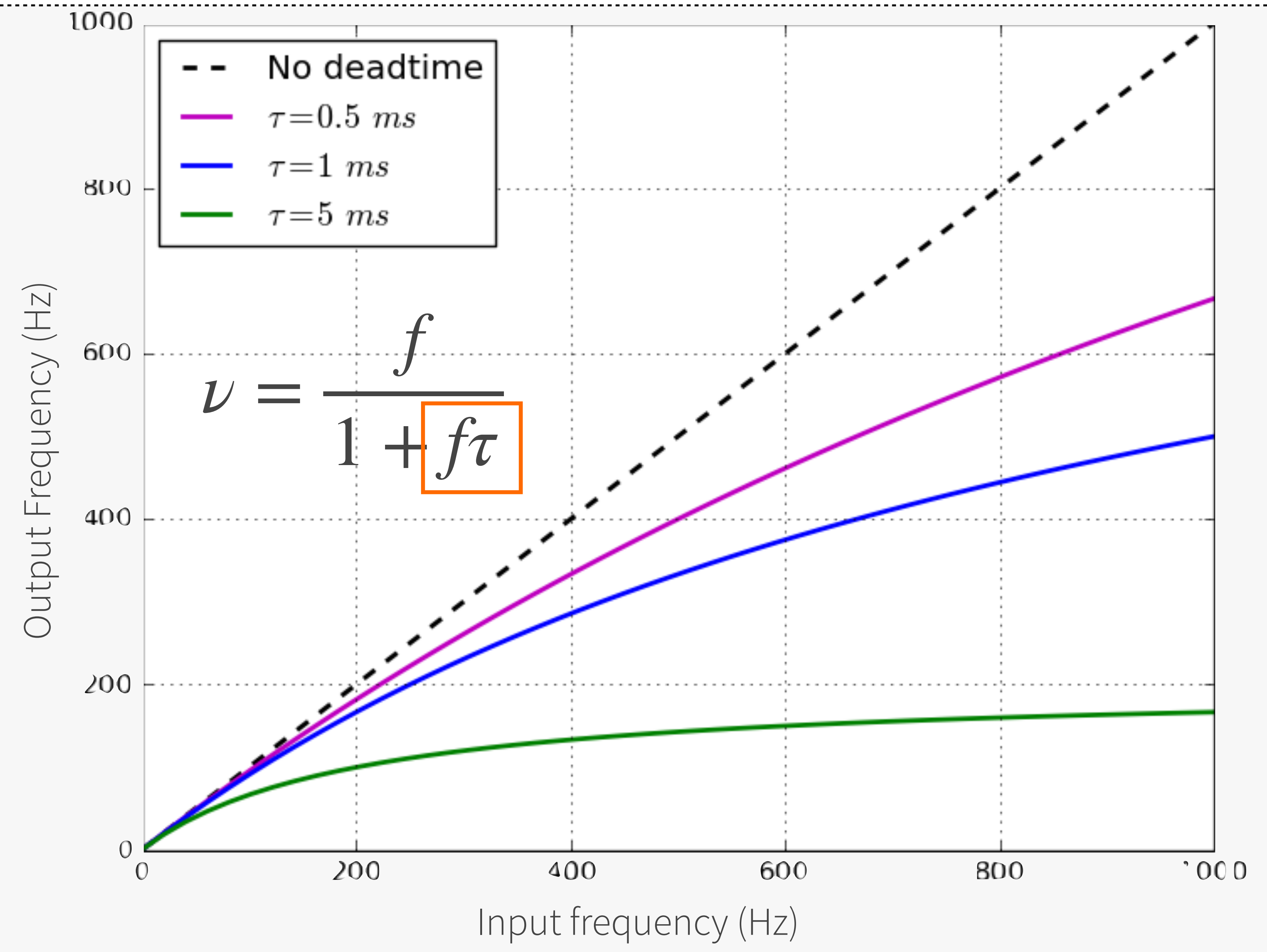
In order to obtain $\epsilon \simeq$ 100% (i.e.: $\nu \simeq \tau$) → $f\tau \ll 1$ → $\tau \ll \lambda$

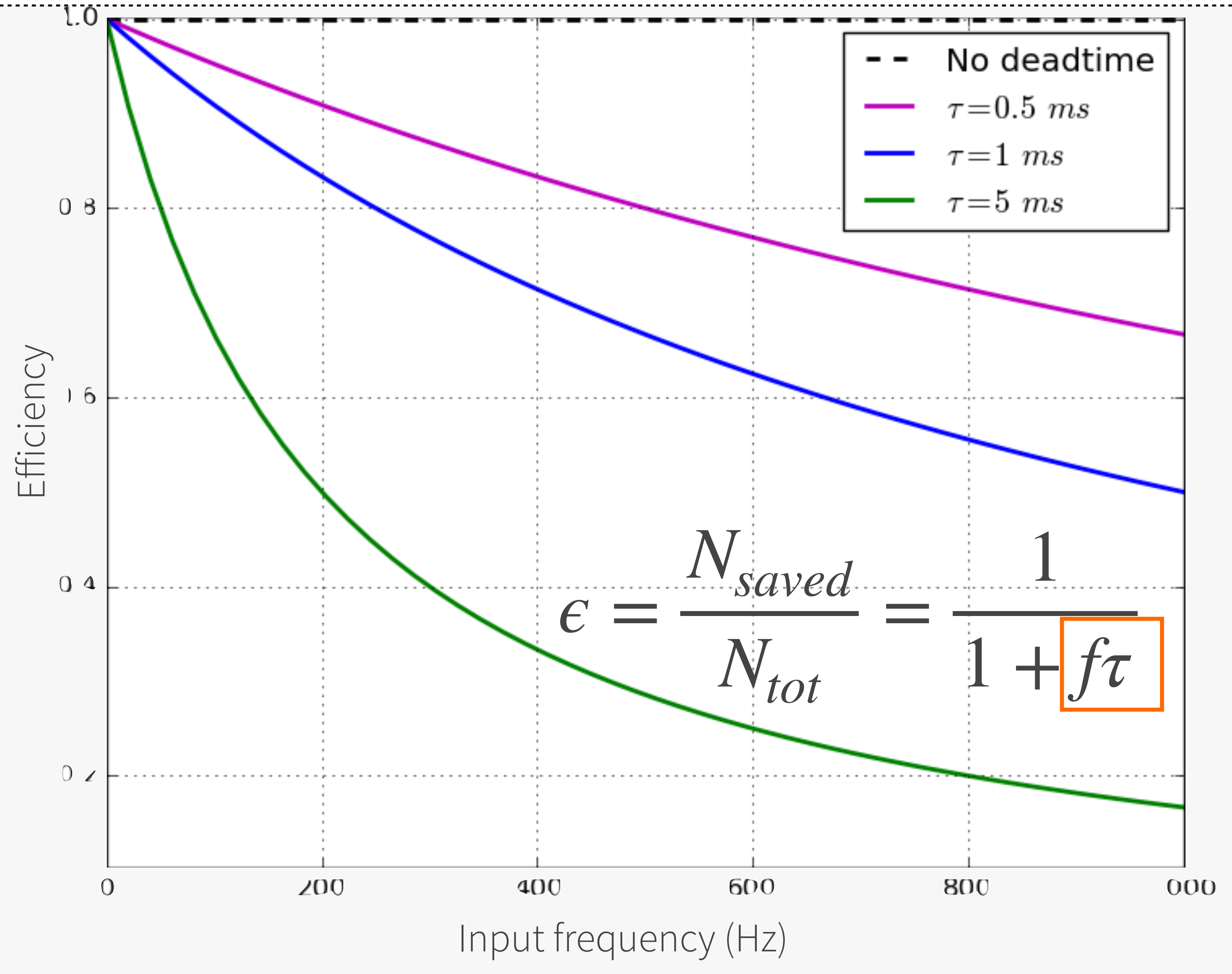
- E.g.: $\epsilon \simeq$ 99% for f = 1 kHz → τ < 0.01 ms → $1/\tau$ > 100 kHz
- To cope with the input signal fluctuations,

  we have to **over-design** our DAQ system by a **factor 100**!

How can we mitigate this effect?

Deadtime and efficiency





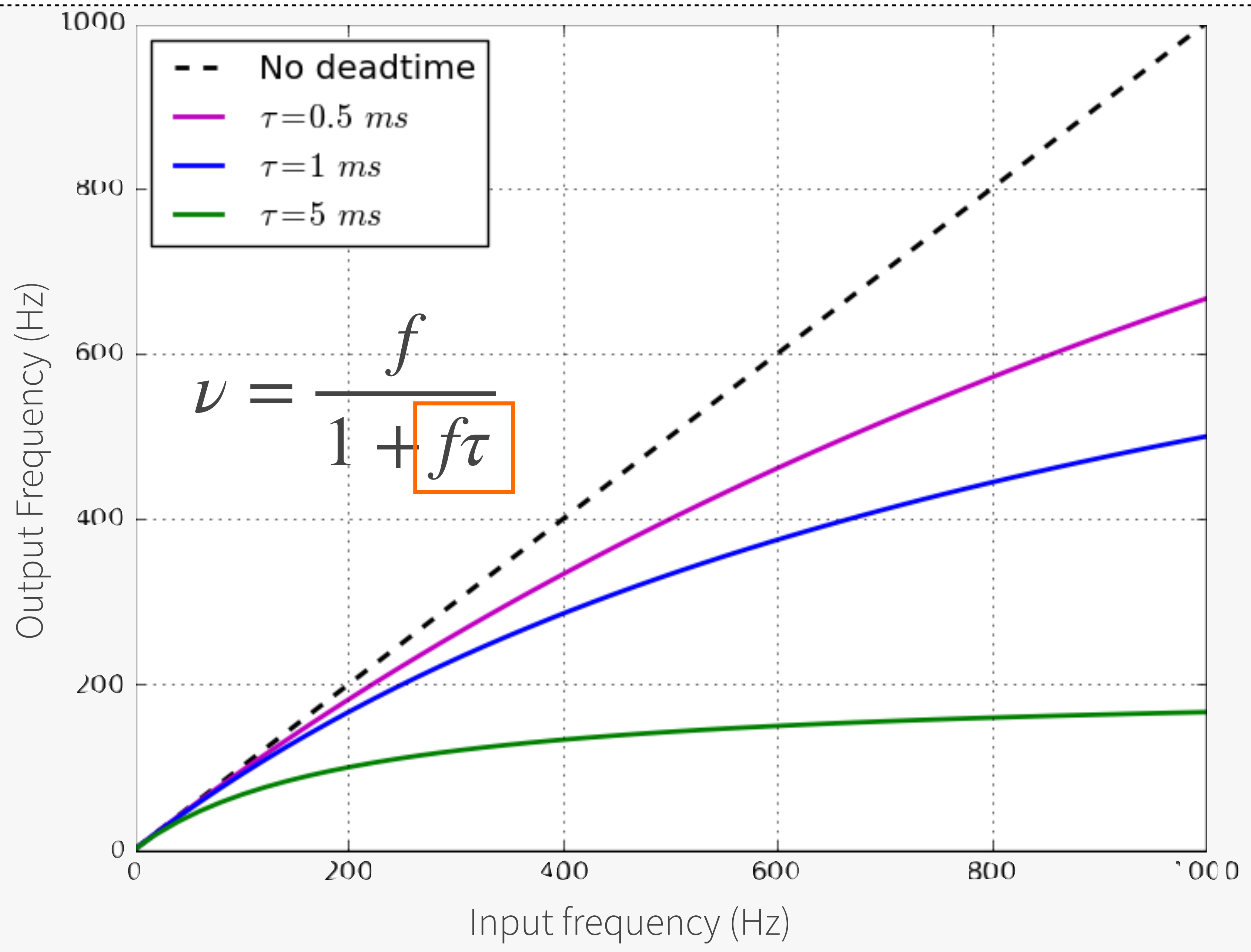
In order to obtain $\epsilon \simeq$ 100% (i.e.: $\nu \simeq \tau$) → $f\tau \ll 1 \rightarrow \tau \ll \lambda$

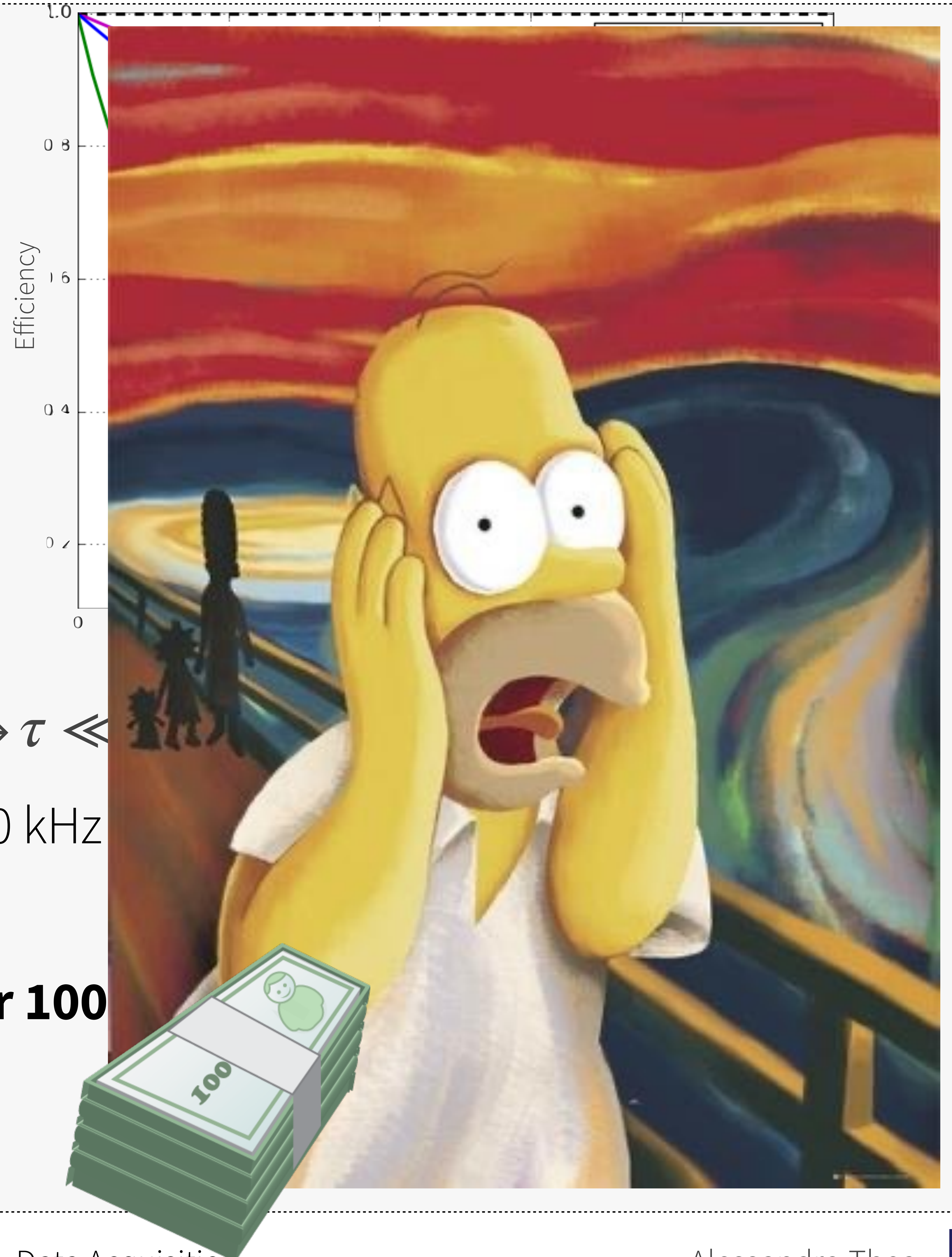
- E.g.: $\epsilon \simeq$ 99% for f = 1 kHz → τ < 0.01 ms → $1/\tau$ > 100 kHz
- To cope with the input signal fluctuations,

  we have to **over-design** our DAQ system by a **factor 100**!

How can we mitigate this effect?

Deadtime and efficiency





In order to obtain $\epsilon \simeq$ 100% (i.e.: $\nu \simeq \tau$) → $f\tau \ll 1$ → $\tau \ll$

- E.g.: $\epsilon \simeq$ 99% for f = 1 kHz → τ < 0.01 ms → $1/\tau$ > 100 kHz
- To cope with the input signal fluctuations,

  we have to **over-design** our DAQ system by a **factor 100**

How can we mitigate this effect?

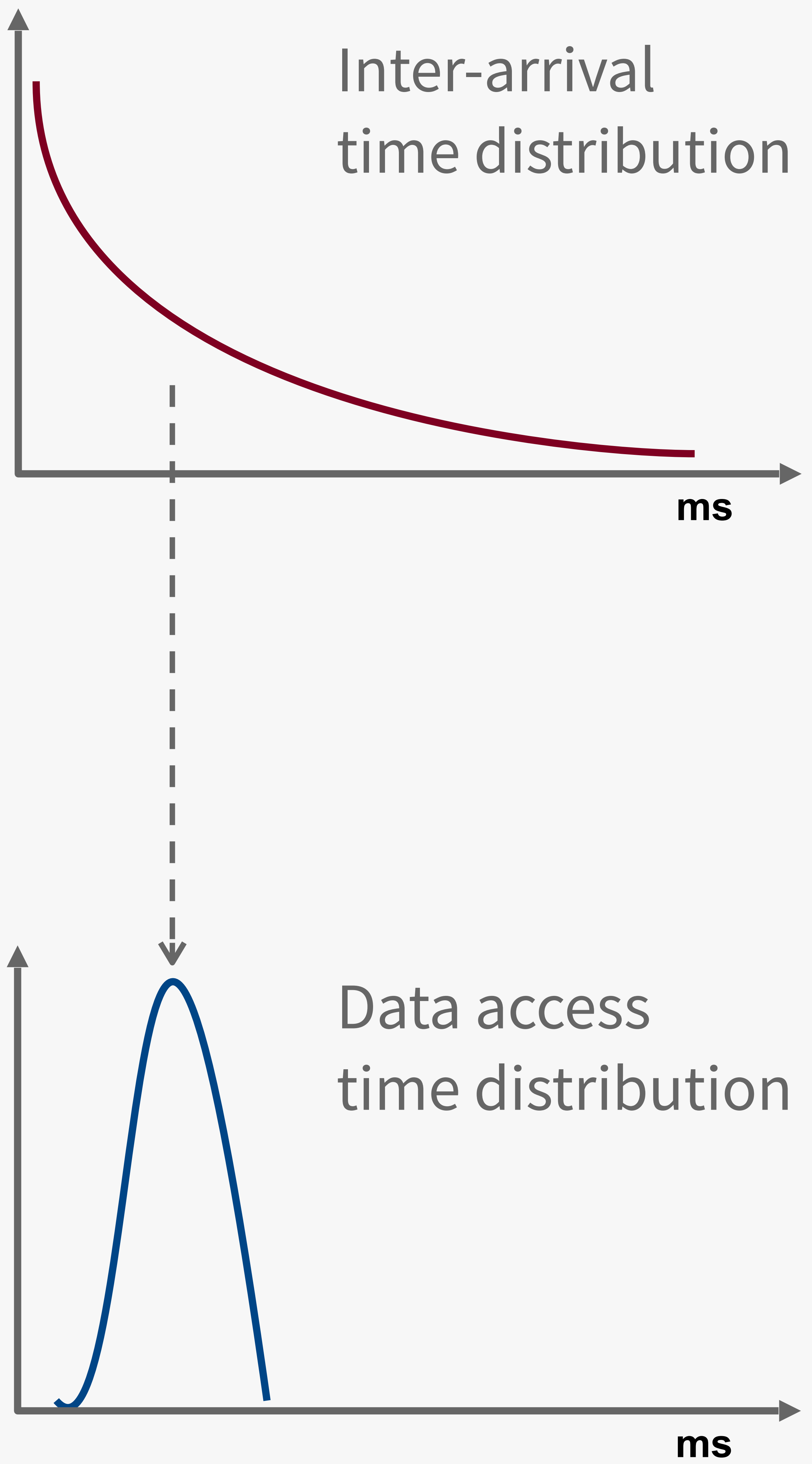
De-randomization

What if we were able to make the system more **deterministic** and less dependent on the arrival time of our signals?

- Then we could ensure that events don't arrive when the system is busy
- This is called **de-randomization**

How it can be achieved?

- by **buffering** the data (introducing a holding queue where it can wait to be processed)



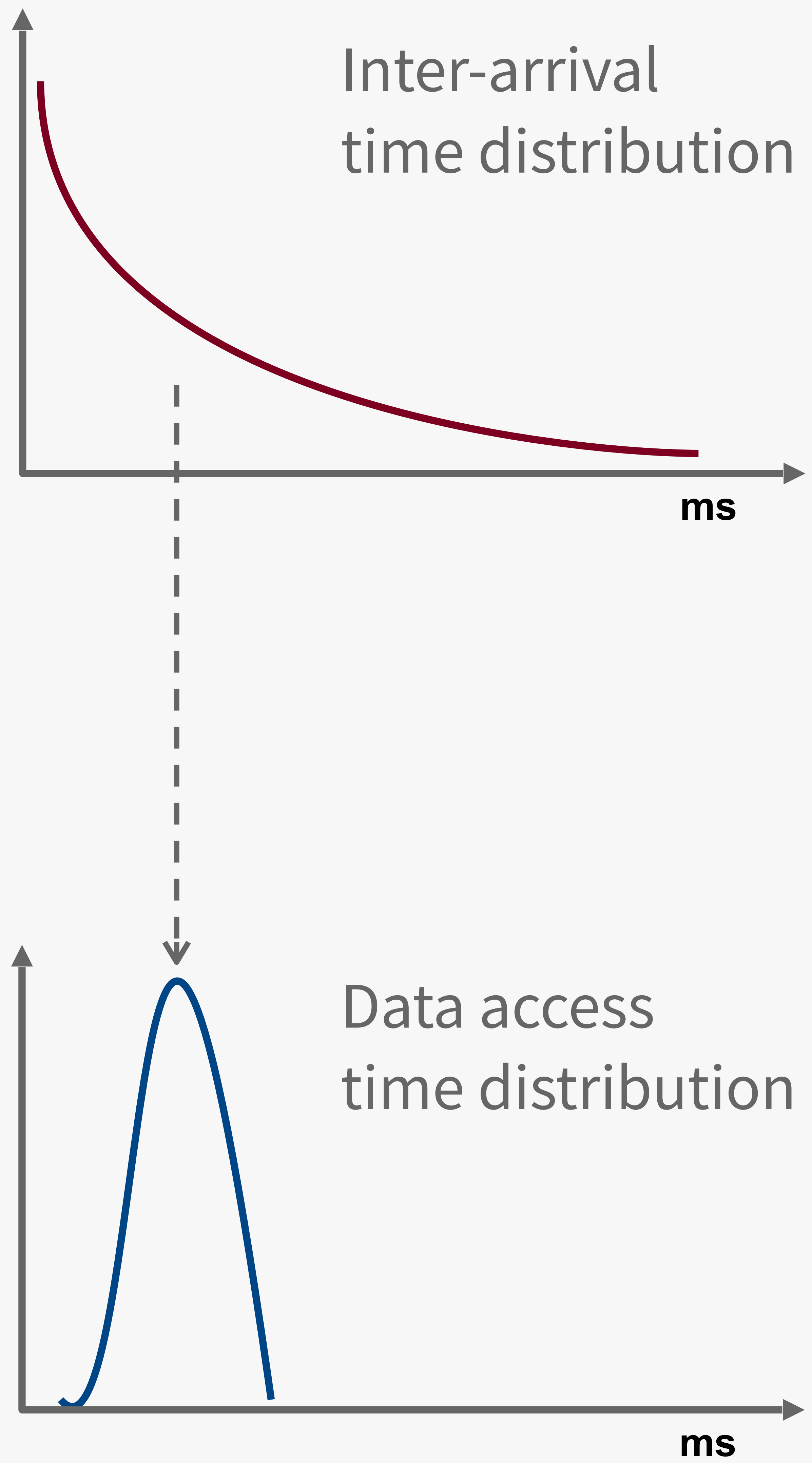
De-randomization

What if we were able to make the system more **deterministic** and less dependent on the arrival time of our signals?

- Then we could ensure that events don't arrive when the system is busy
- This is called **de-randomization**

How it can be achieved?

- by **buffering** the data (introducing a holding queue where it can wait to be processed)



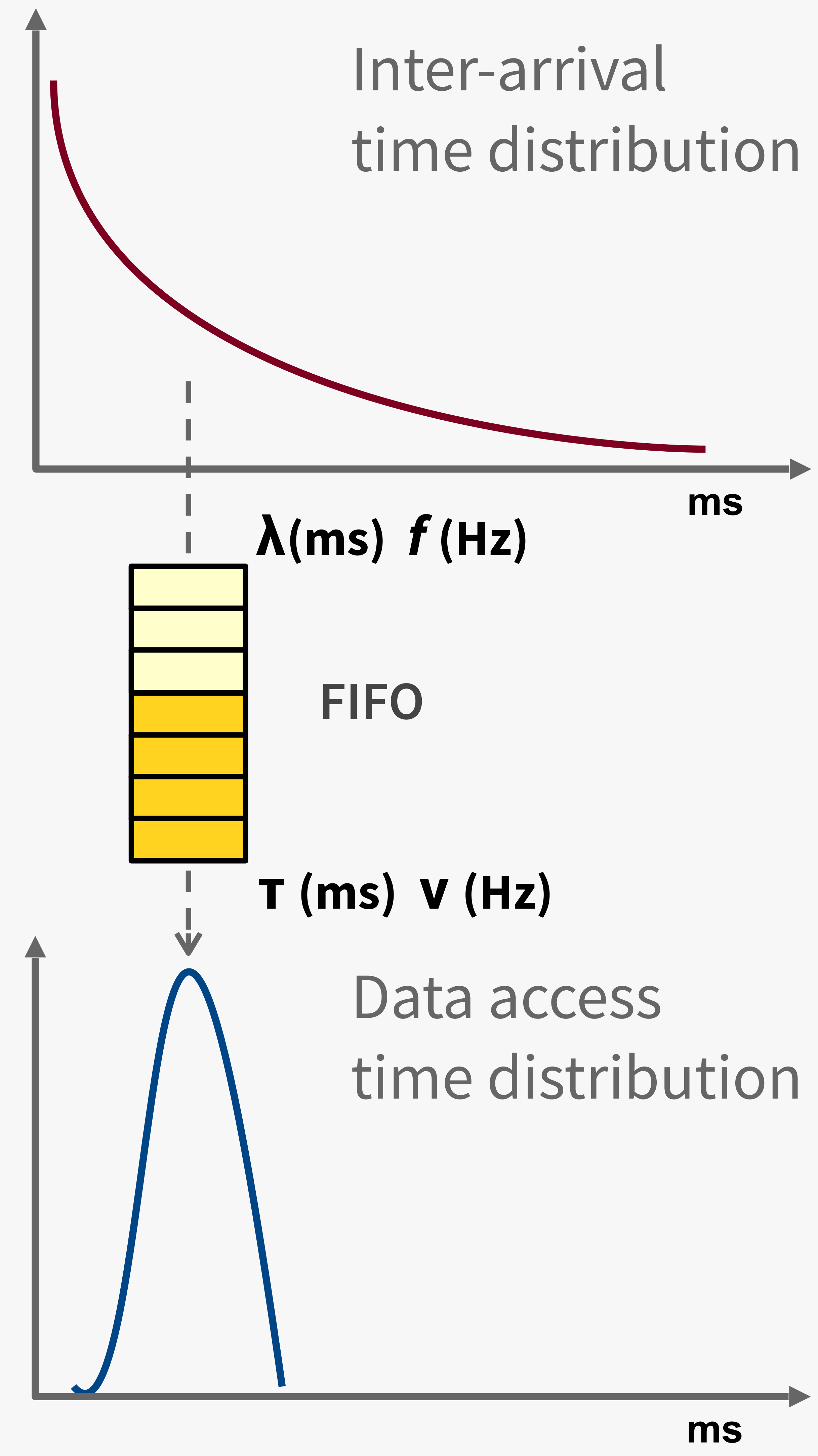
De-randomization

What if we were able to make the system more **deterministic** and less dependent on the arrival time of our signals?

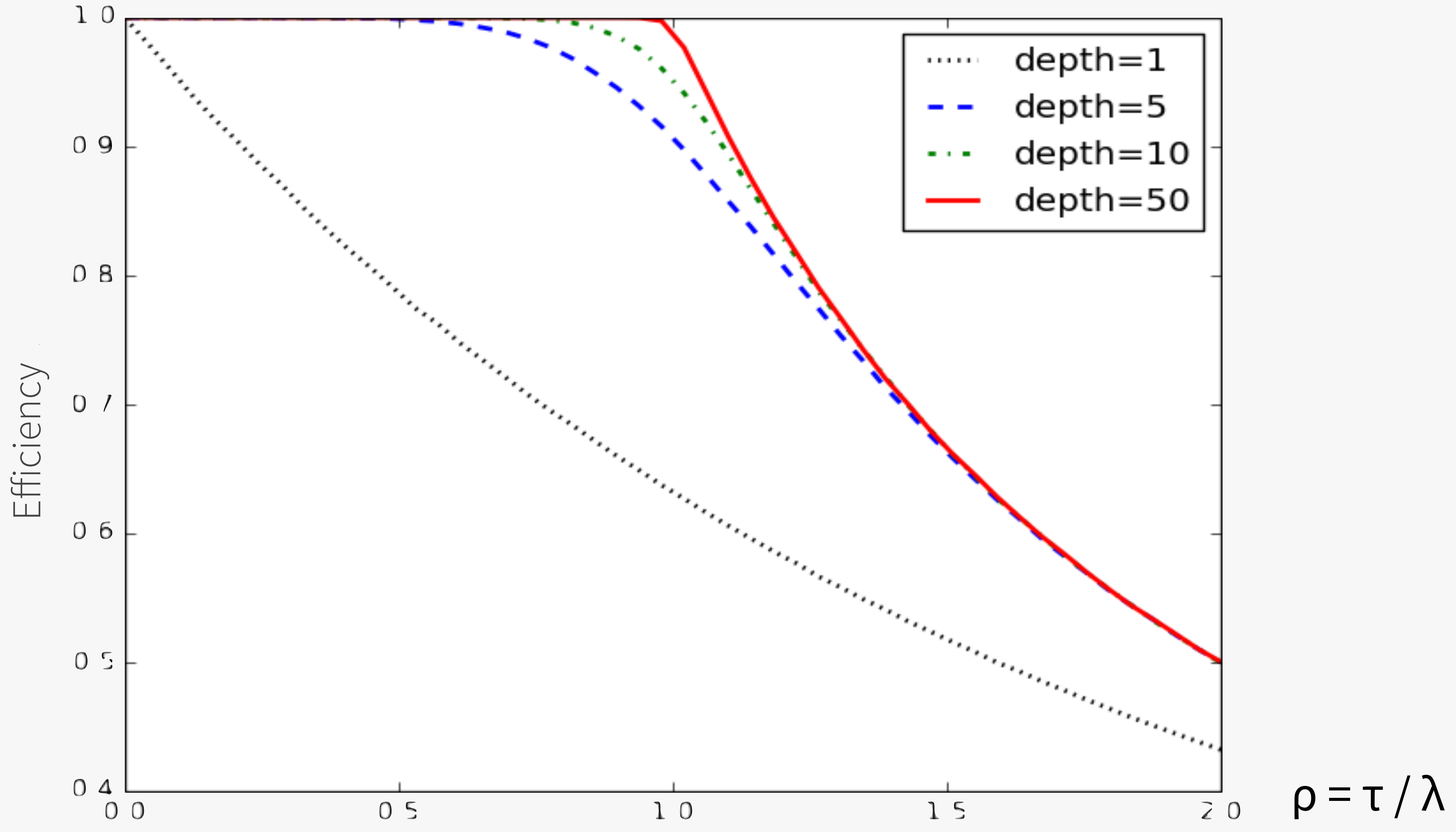
- Then we could ensure that events don't arrive when the system is busy
- This is called **de-randomization**

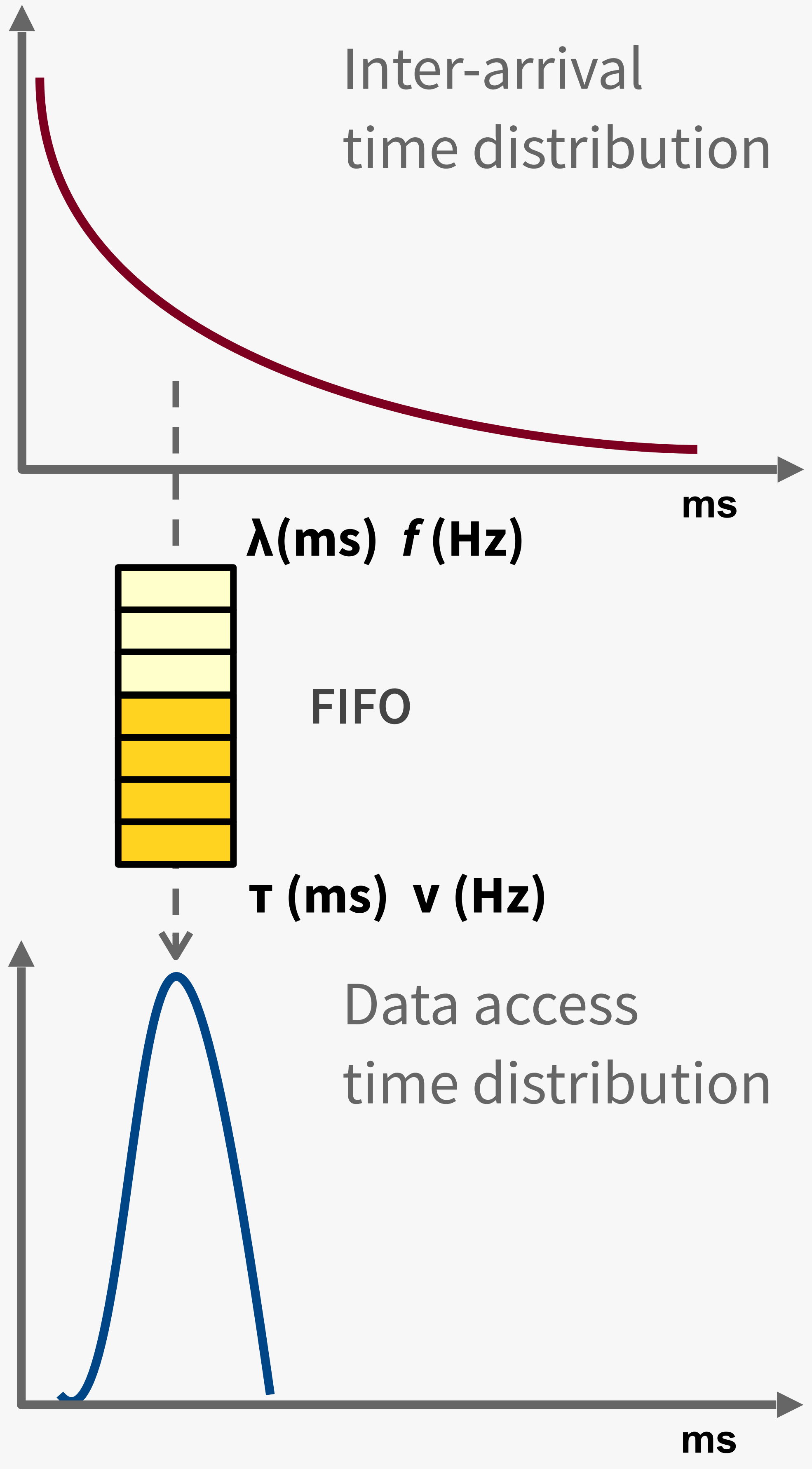
How it can be achieved?

- by **buffering** the data: introducing a holding queue where it can wait to be processed



Queuing theory





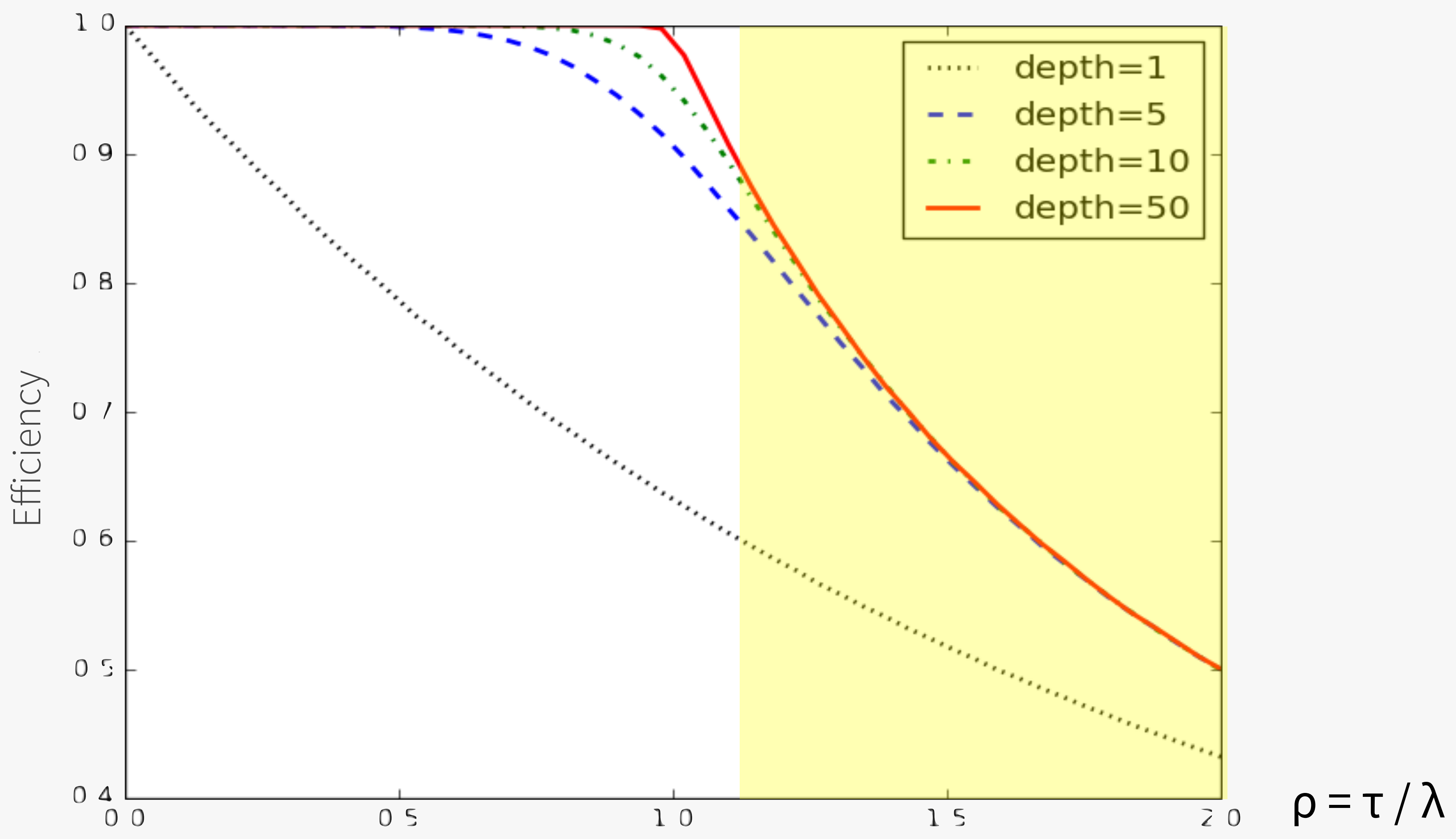
Efficiency vs traffic intensity ($\rho = \tau/\lambda$) for different queue depths

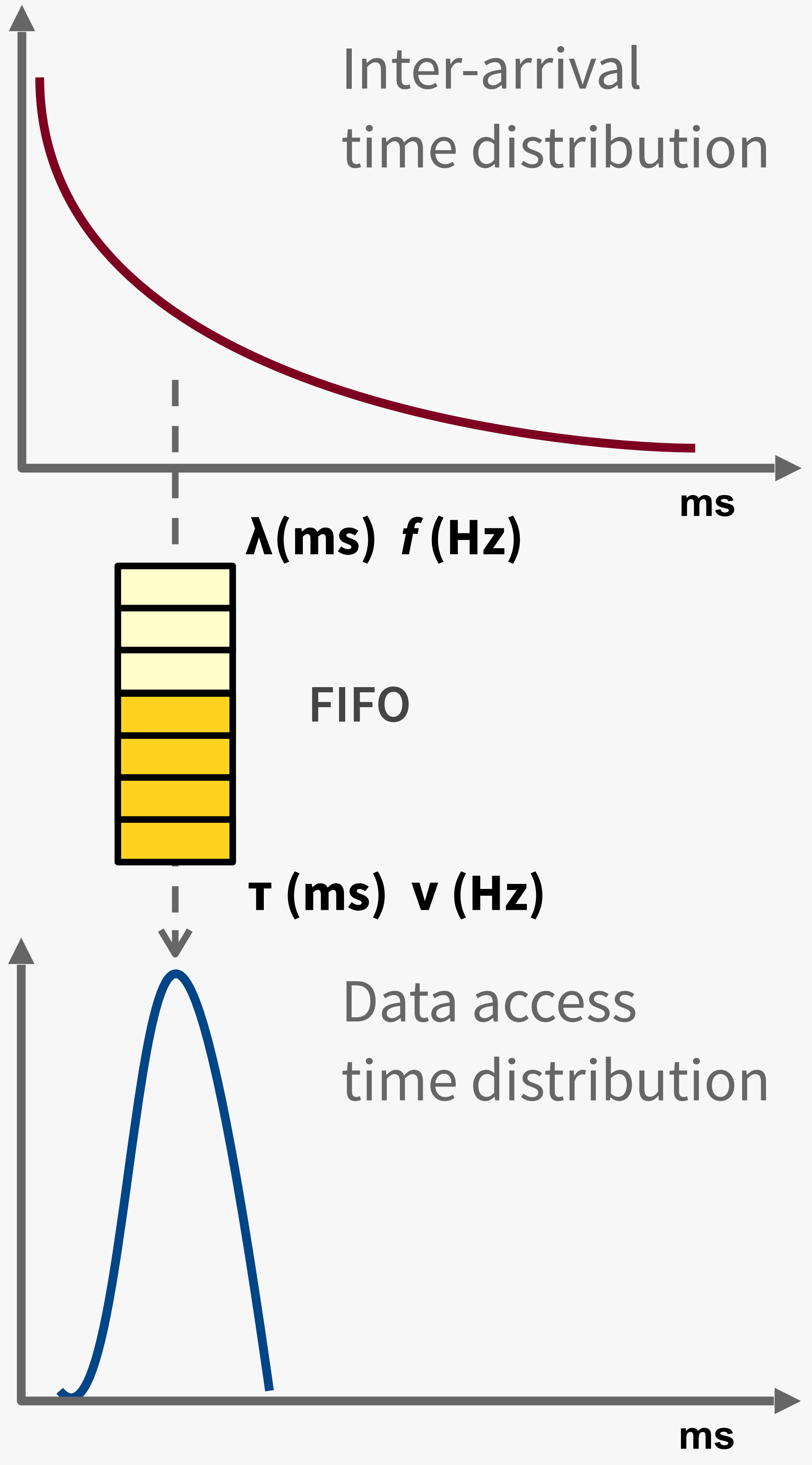
- ρ ≪ 1: the output is over-designed ($\tau \ll \lambda$)
- ρ > 1: the system is overloaded ($\tau > \lambda$)
- ρ ~ 1: using a queue, high efficiency obtained even w/ moderate depth

Analytic calculation possible for very simple systems only

- Otherwise MonteCarlo simulation is required

Queuing theory





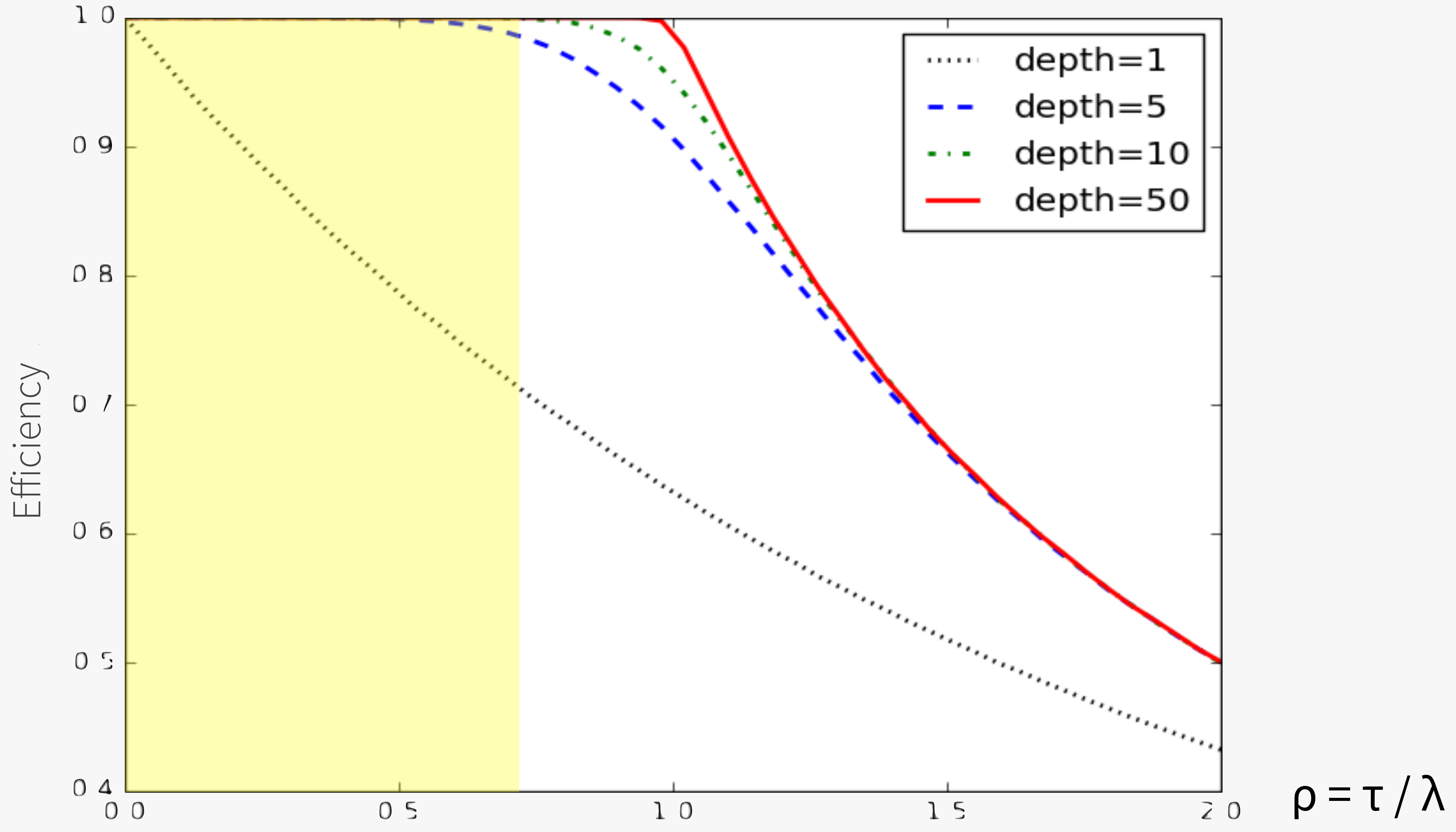
Efficiency vs traffic intensity ($\rho = \tau/\lambda$) for different queue depths

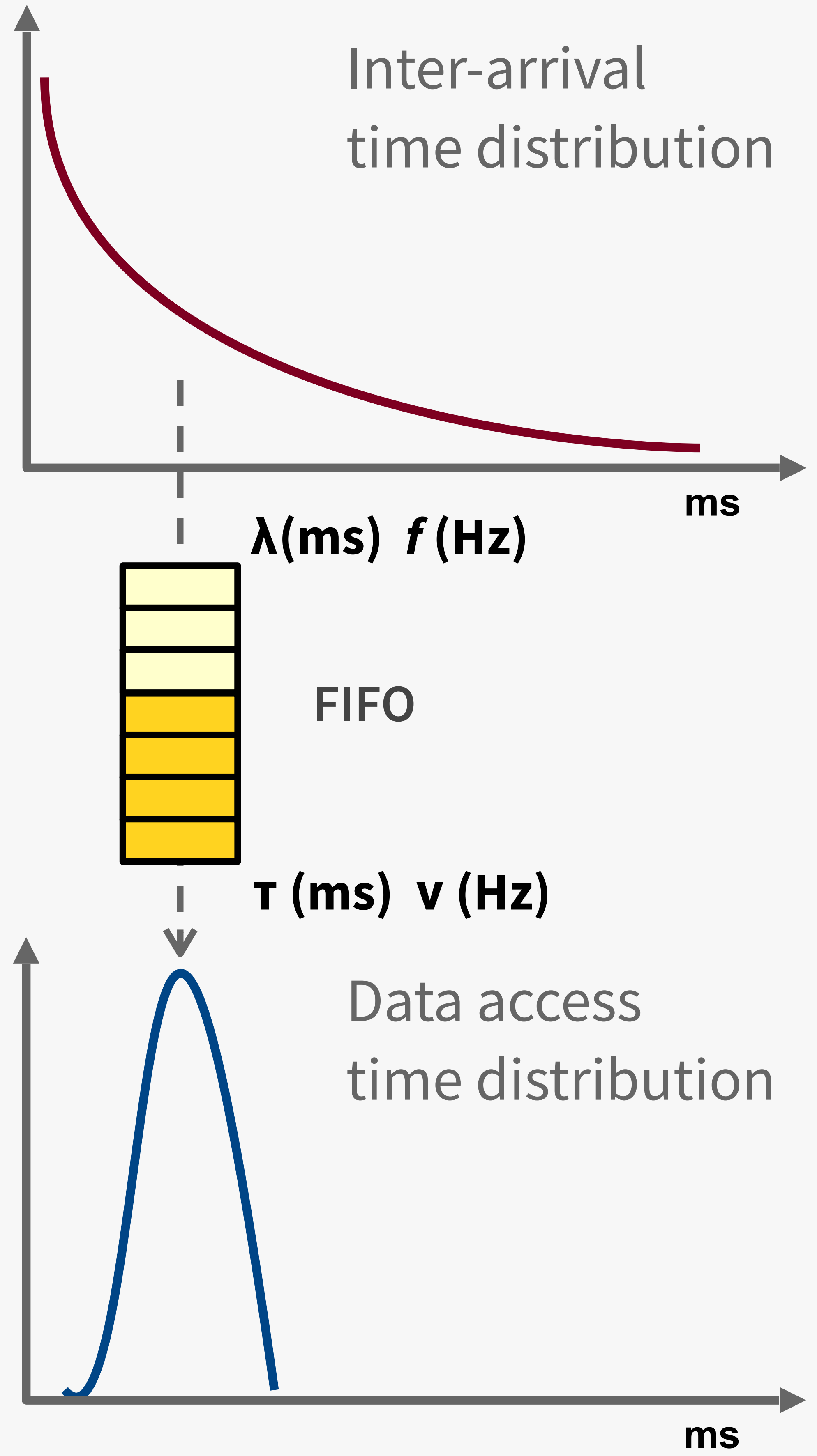
- $\rho \ll 1$: the output is over-designed ($\tau \ll \lambda$)
- $\rho > 1$: the system is overloaded ($\tau > \lambda$)
- $\rho \sim 1$: using a queue, high efficiency obtained even w/ moderate depth

Analytic calculation possible for very simple systems only

- Otherwise MonteCarlo simulation is required

Queuing theory





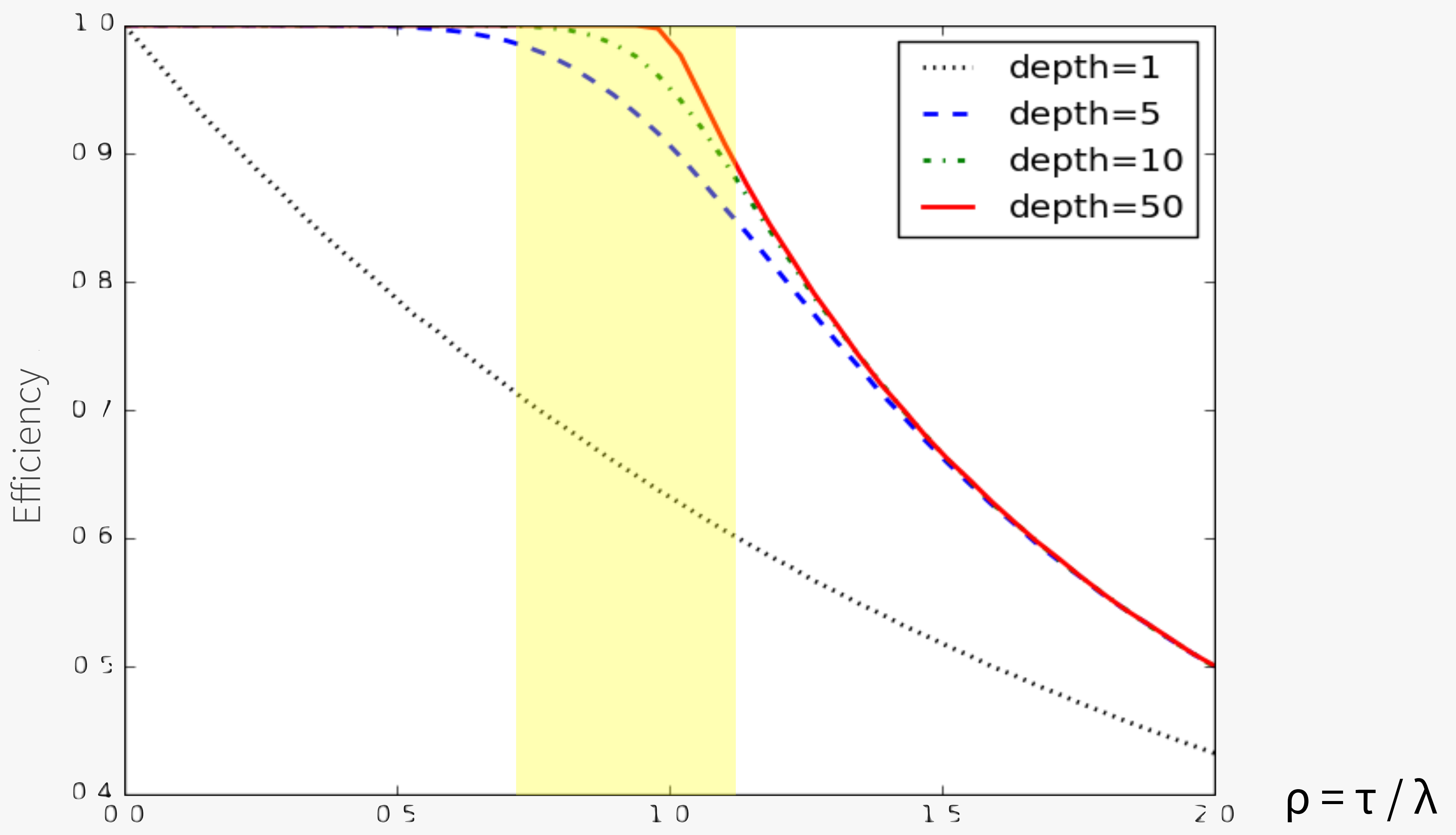
Efficiency vs traffic intensity ($\rho = \tau/\lambda$) for different queue depths

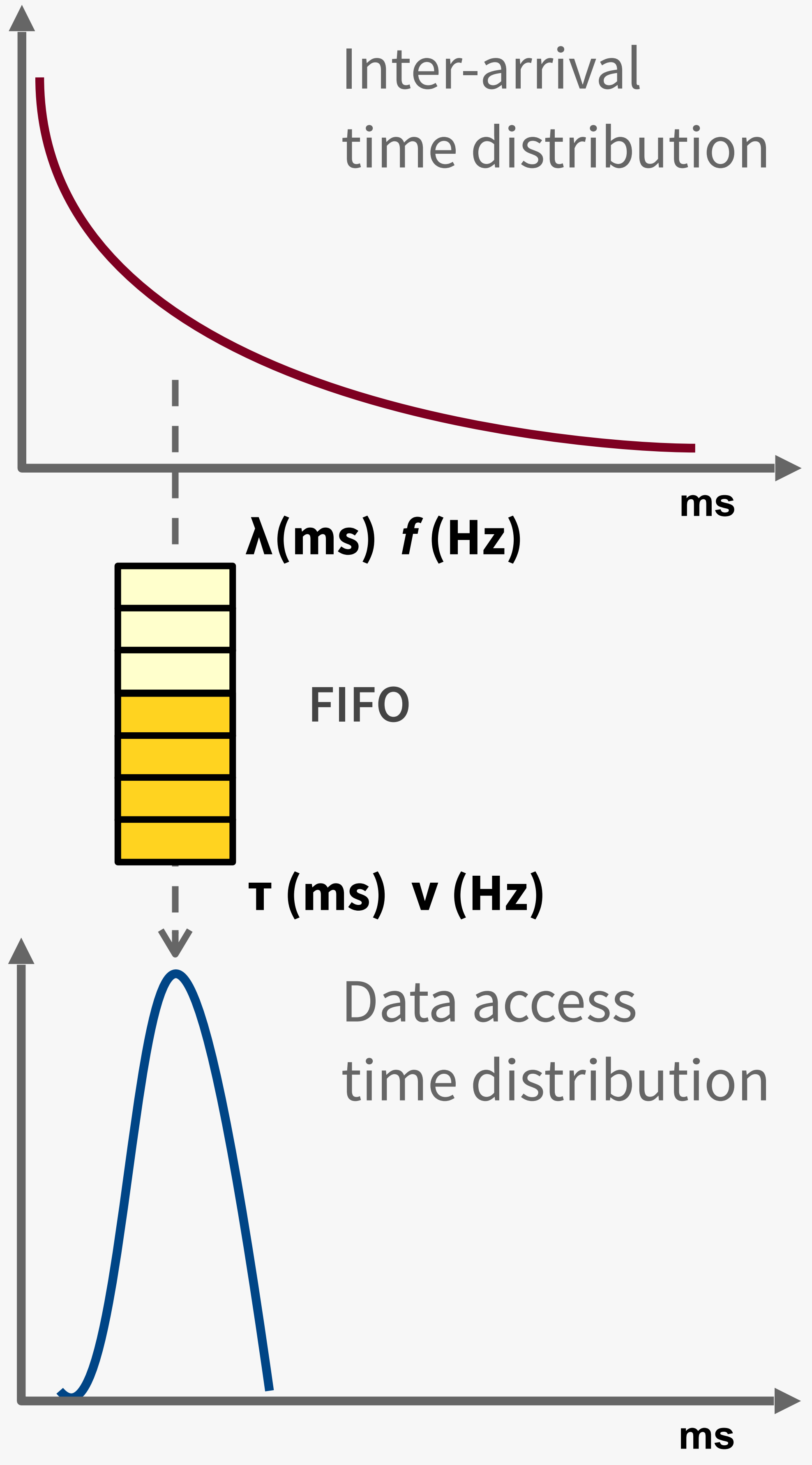
- ρ ≪ 1: the output is over-designed ($\tau \ll \lambda$)
- ρ > 1: the system is overloaded ($\tau > \lambda$)
- ρ ~ 1: using a queue, high efficiency obtained even w/ moderate depth

Analytic calculation possible for very simple systems only

- Otherwise MonteCarlo simulation is required

Queuing theory





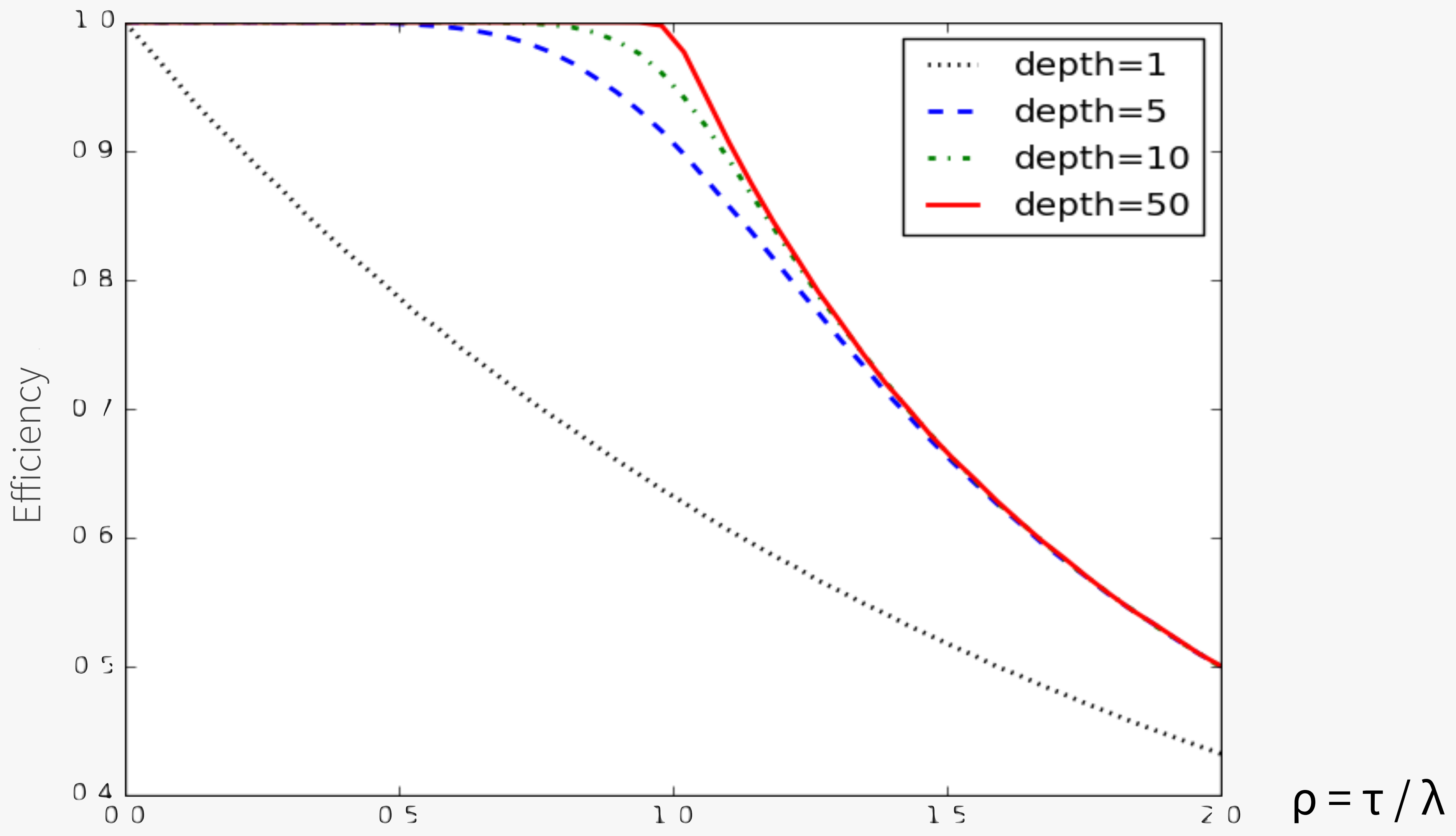
Efficiency vs traffic intensity ($\rho = \tau/\lambda$) for different queue depths

- ρ ≪ 1: the output is over-designed ($\tau \ll \lambda$)
- ρ > 1: the system is overloaded ($\tau > \lambda$)
- ρ ~ 1: using a queue, high efficiency obtained even w/ moderate depth

Analytic calculation possible for very simple systems only

- Otherwise MonteCarlo simulation is required

Queuing theory

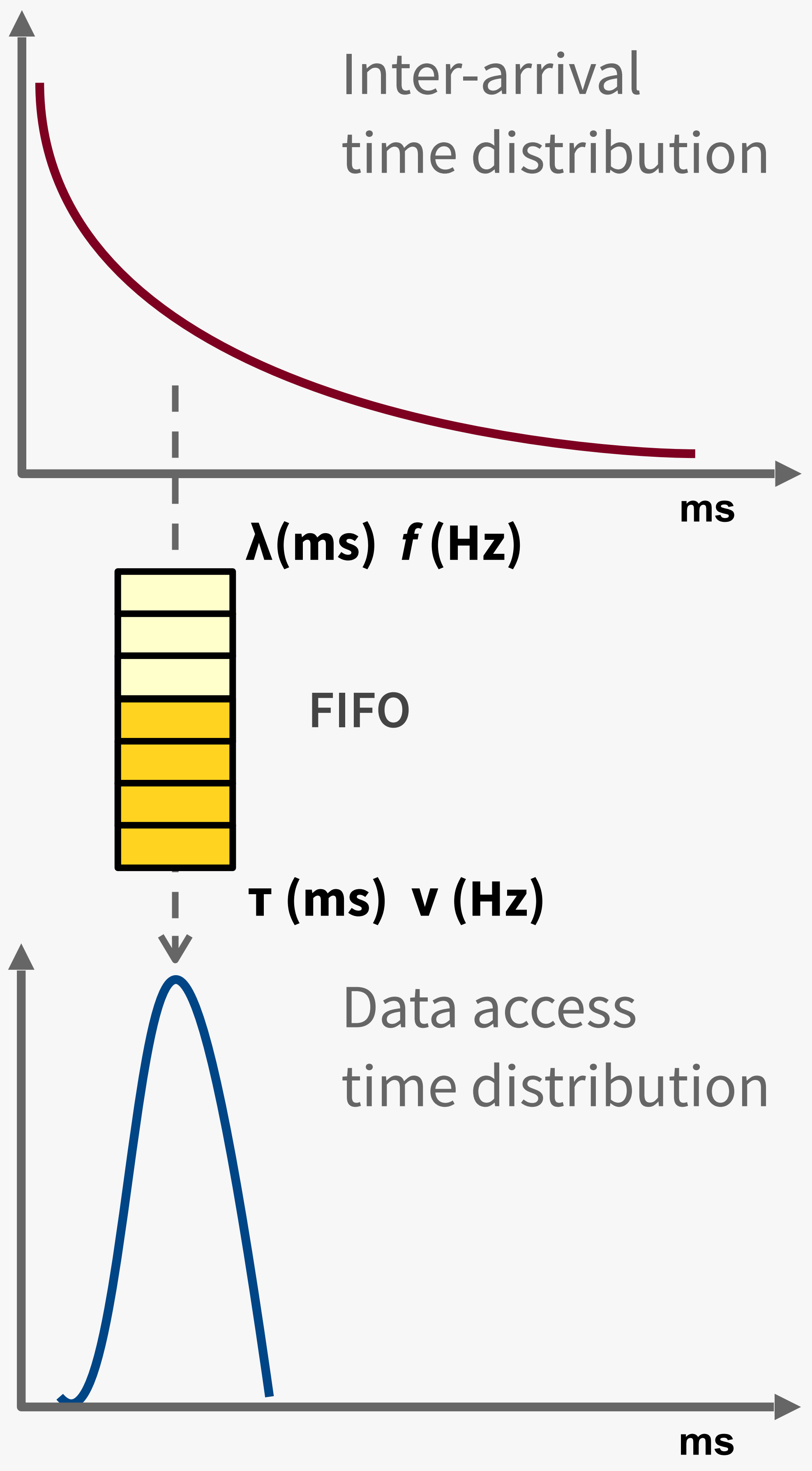


Efficiency vs traffic intensity ($\rho = \tau/\lambda$) for different queue depths

- ρ « 1: the output is over-designed ($\tau \ll \lambda$)
- ρ > 1: the system is overloaded ($\tau > \lambda$)
- ρ ~ 1: using a queue, high efficiency obtained even w/ moderate depth

Analytic calculation possible for very simple systems only

- Otherwise MonteCarlo simulation is required



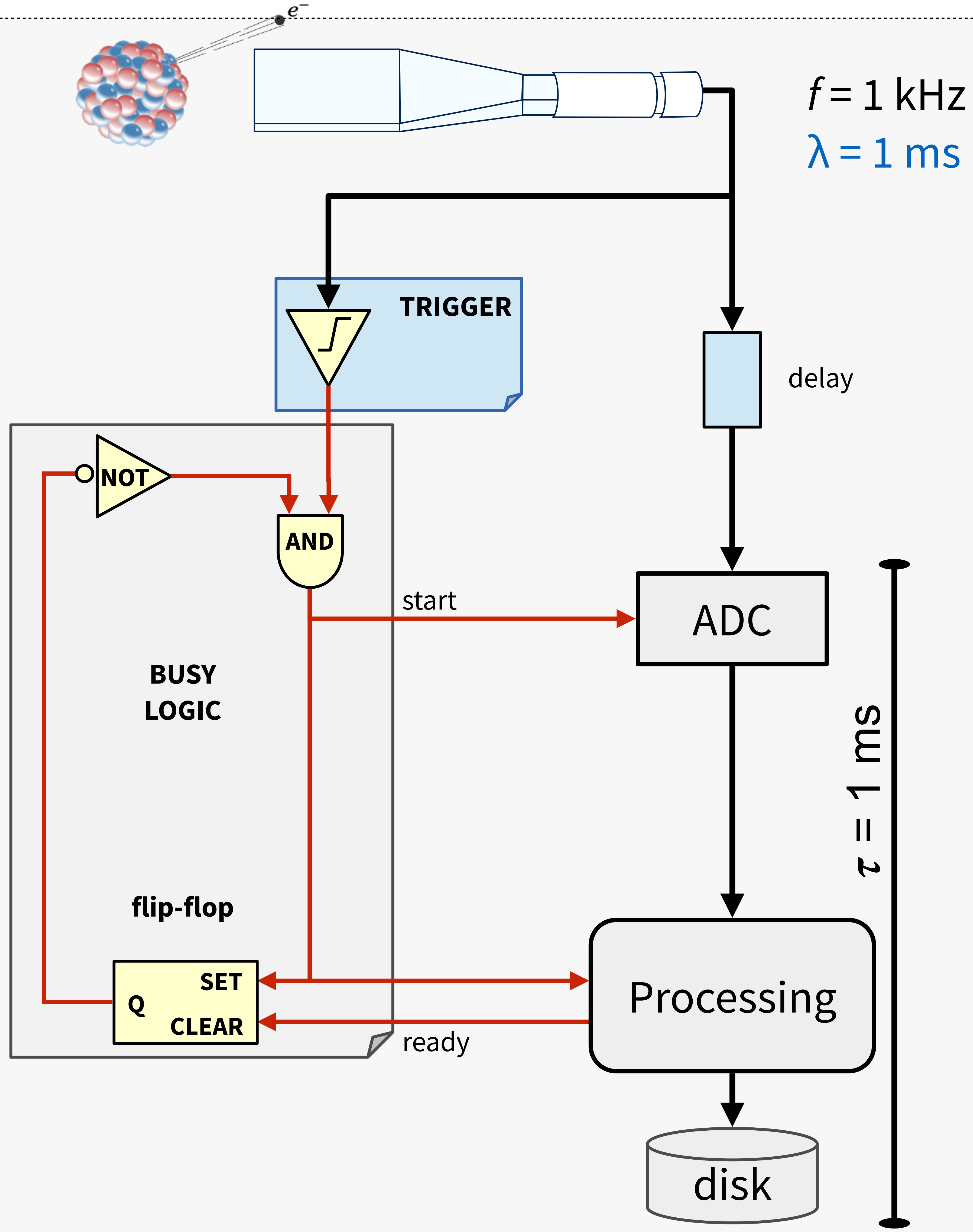
De-randomization

Input fluctuations can be absorbed and smoothed by a queue

- A FIFO can provide a ~steady and de-randomized output rate
- The effect of the queue depends on its depth

Busy is now defined by the buffer occupancy

- Processor pulls data from the buffer at fixed rate, separating the event receiving and data processing steps



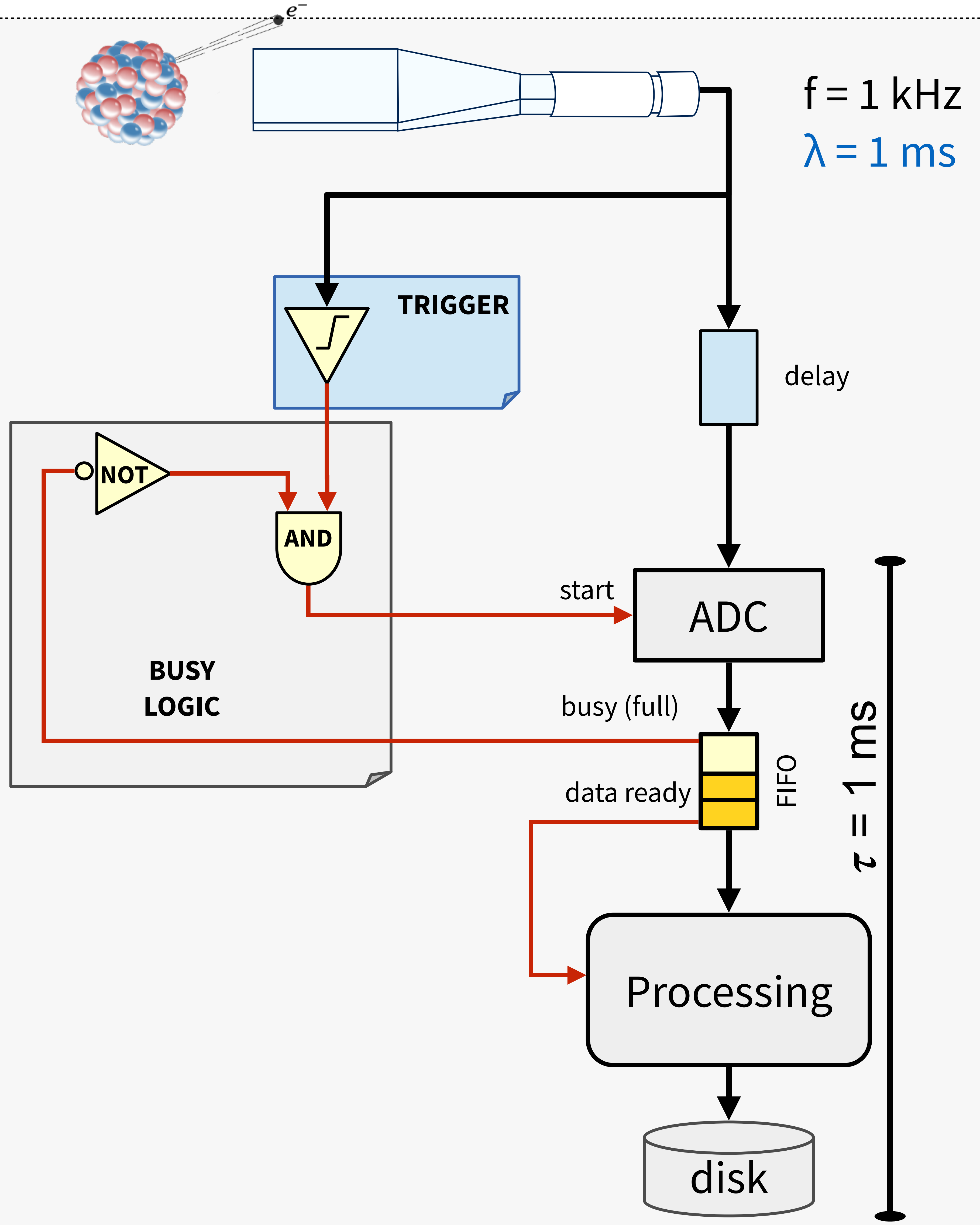
De-randomization

Input fluctuations can be absorbed and smoothed by a queue

- A FIFO can provide a ~steady and de-randomized output rate
- The effect of the queue depends on its depth

Busy is now defined by the buffer occupancy

- Processor pulls data from the buffer at fixed rate, separating the event receiving and data processing steps



De-randomization

The FIFO decouples the low latency front-end from the data processing

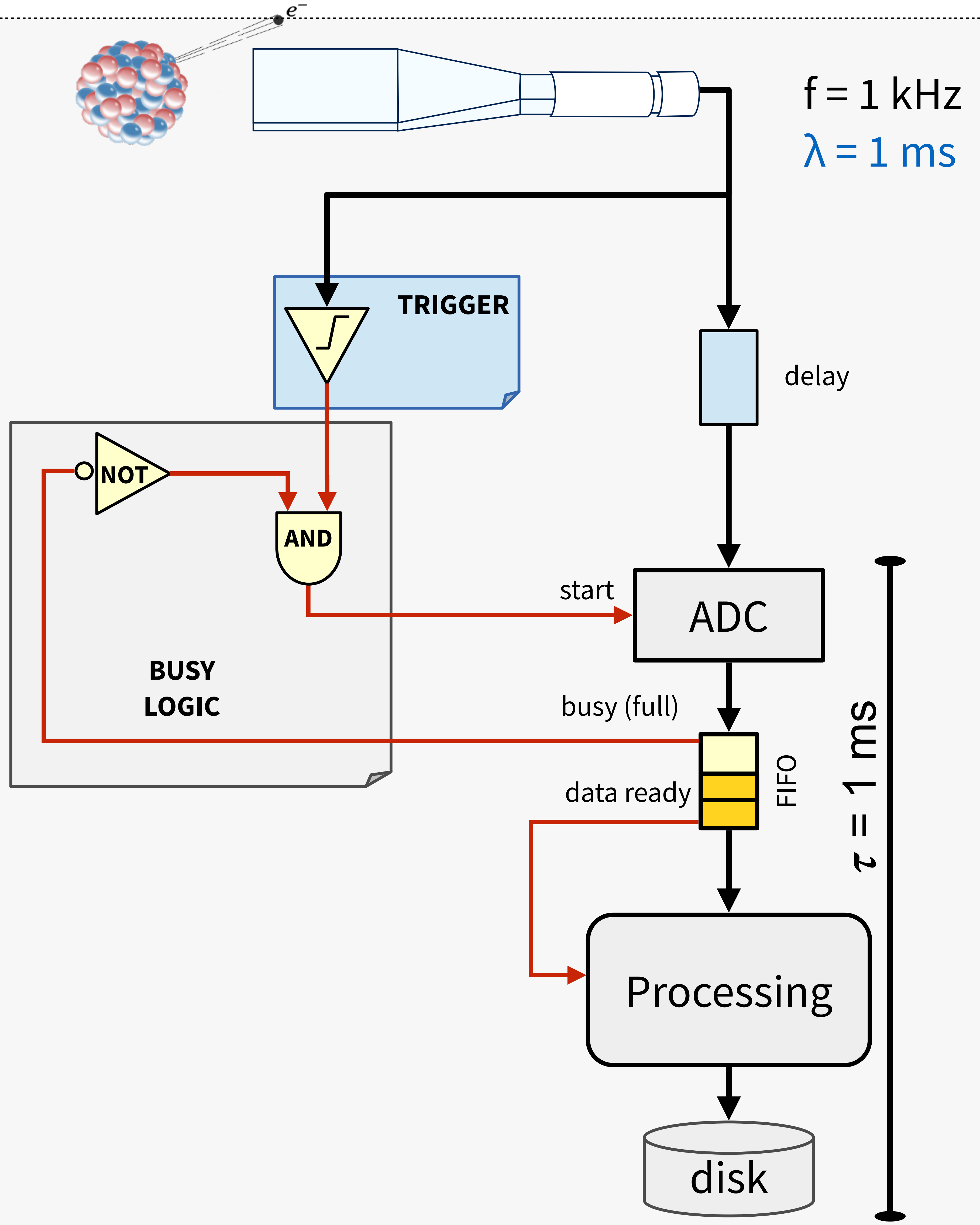
- Minimize the amount of "unnecessary" fast components

~100% efficiency w/ minimal deadtime achievable if

- ADC can operate at rate $\gg f$
- Data processing and storage operate at a rate $\sim f$

Could the delay be replaced with a "FIFO"?

- Analog pipelines, heavily used in LHC DAQs



De-randomization

The FIFO decouples the low latency front-end from the data processing

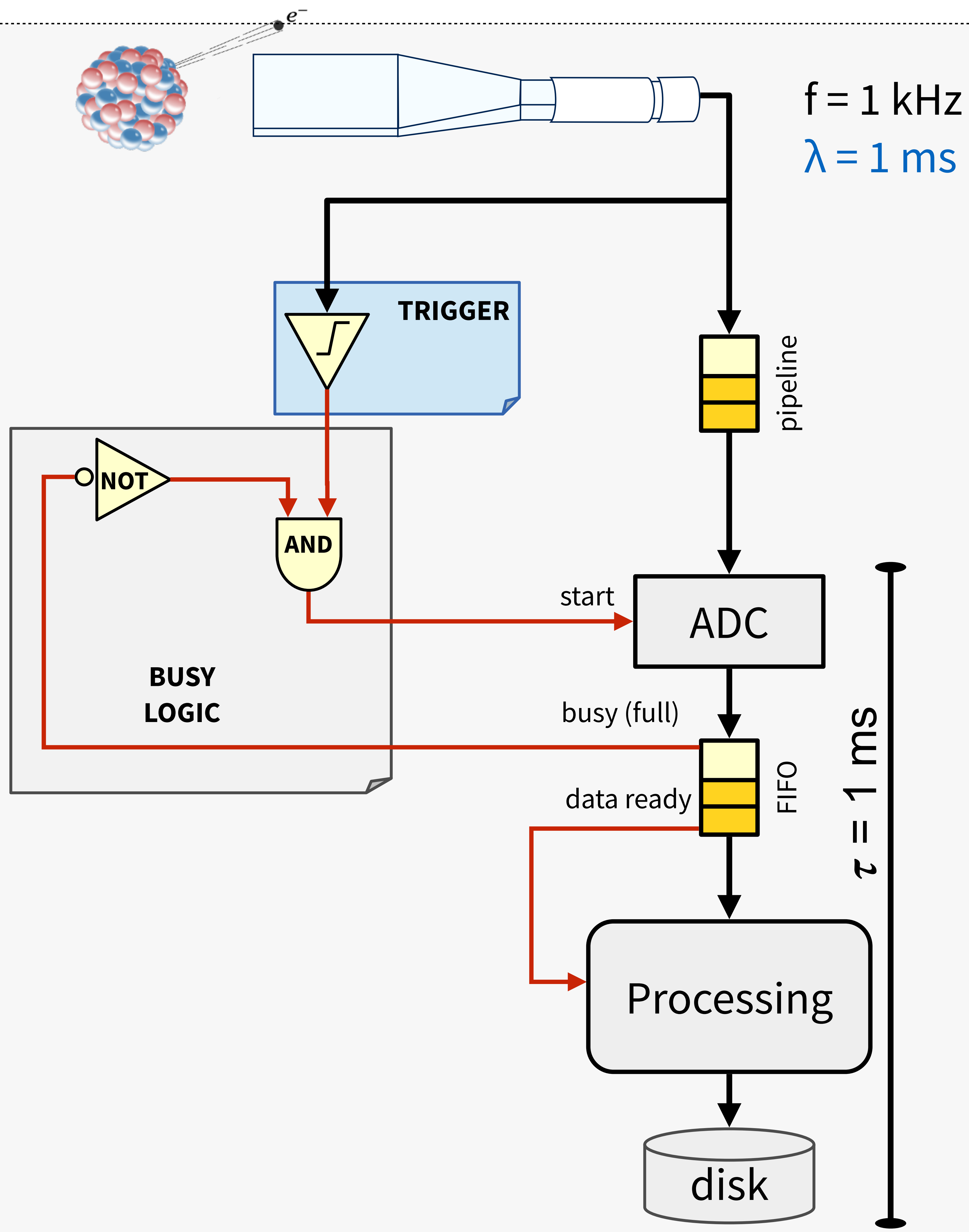
- Minimize the amount of “unnecessary” fast components

~100% efficiency w/ minimal deadtime achievable if

- ADC can operate at rate $\gg f$
- Data processing and storage operate at a rate $\sim f$

Could the delay be replaced with a “FIFO”?

- Analog pipelines, heavily used in LHC DAQs



Collider setup

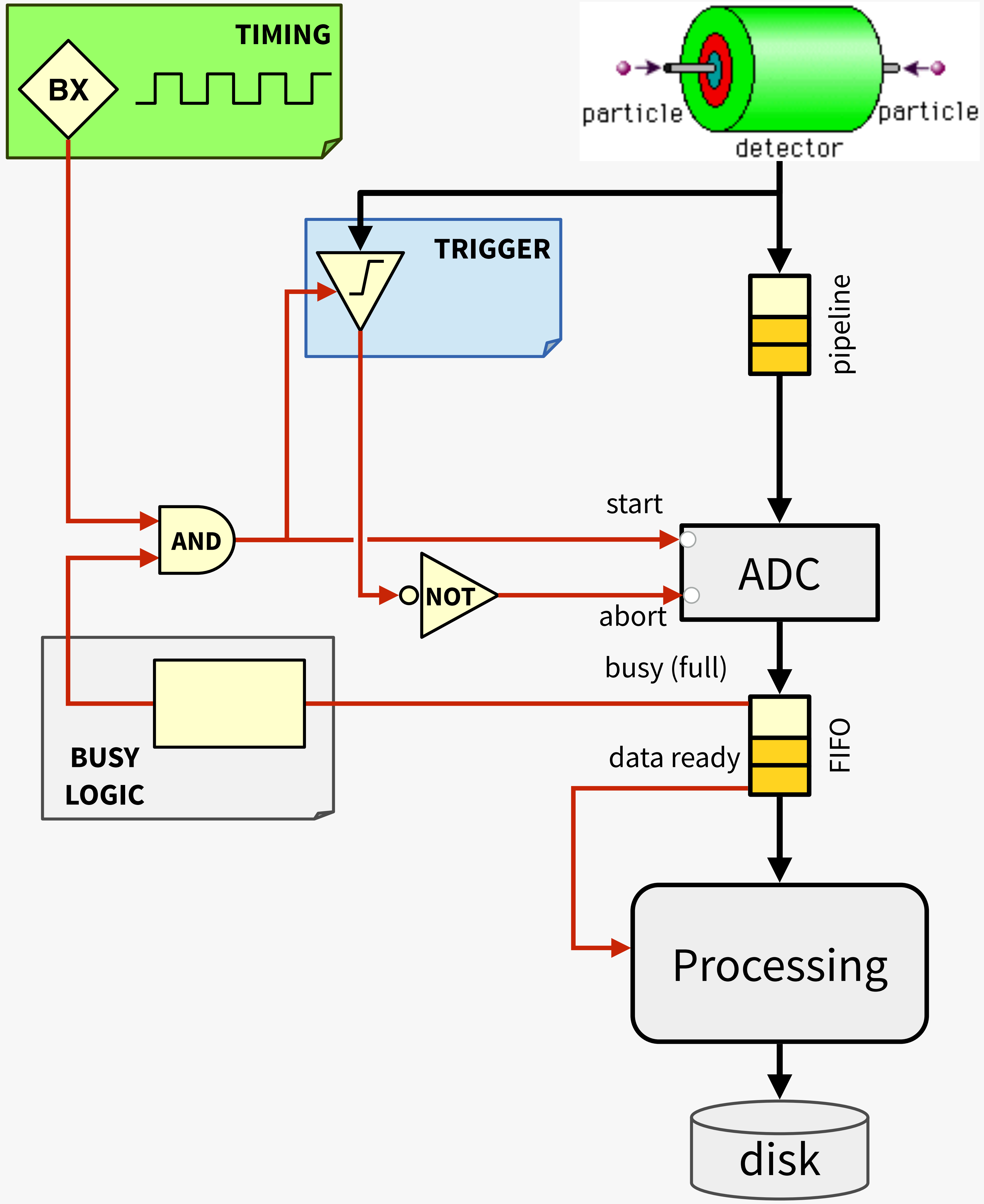
Do we need de-randomization buffers also in collider setups?

- Particle collisions are synchronous
- But the time distribution of triggers is random: interesting events are unpredictable

De-randomization still needed

More complex busy logic to protect buffers and detectors

- Eg: accept n events every m bunch crossings
- Eg: prevent some dangerous trigger patterns



Collider setup

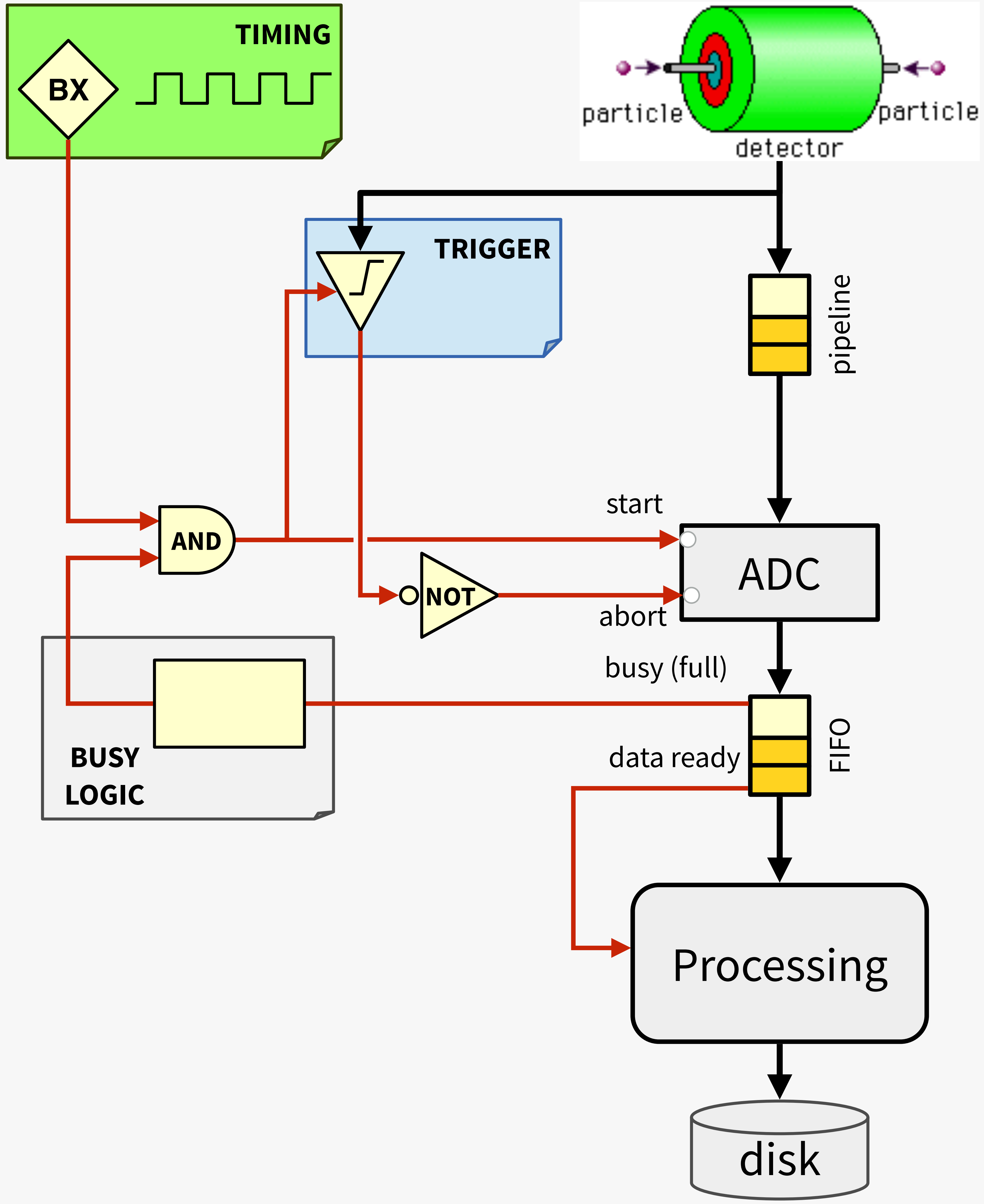
Do we need de-randomization buffers also in collider setups?

- Particle collisions are synchronous
- But the time distribution of triggers is random: interesting events are unpredictable

De-randomization still needed

More complex busy logic to protect buffers and detectors

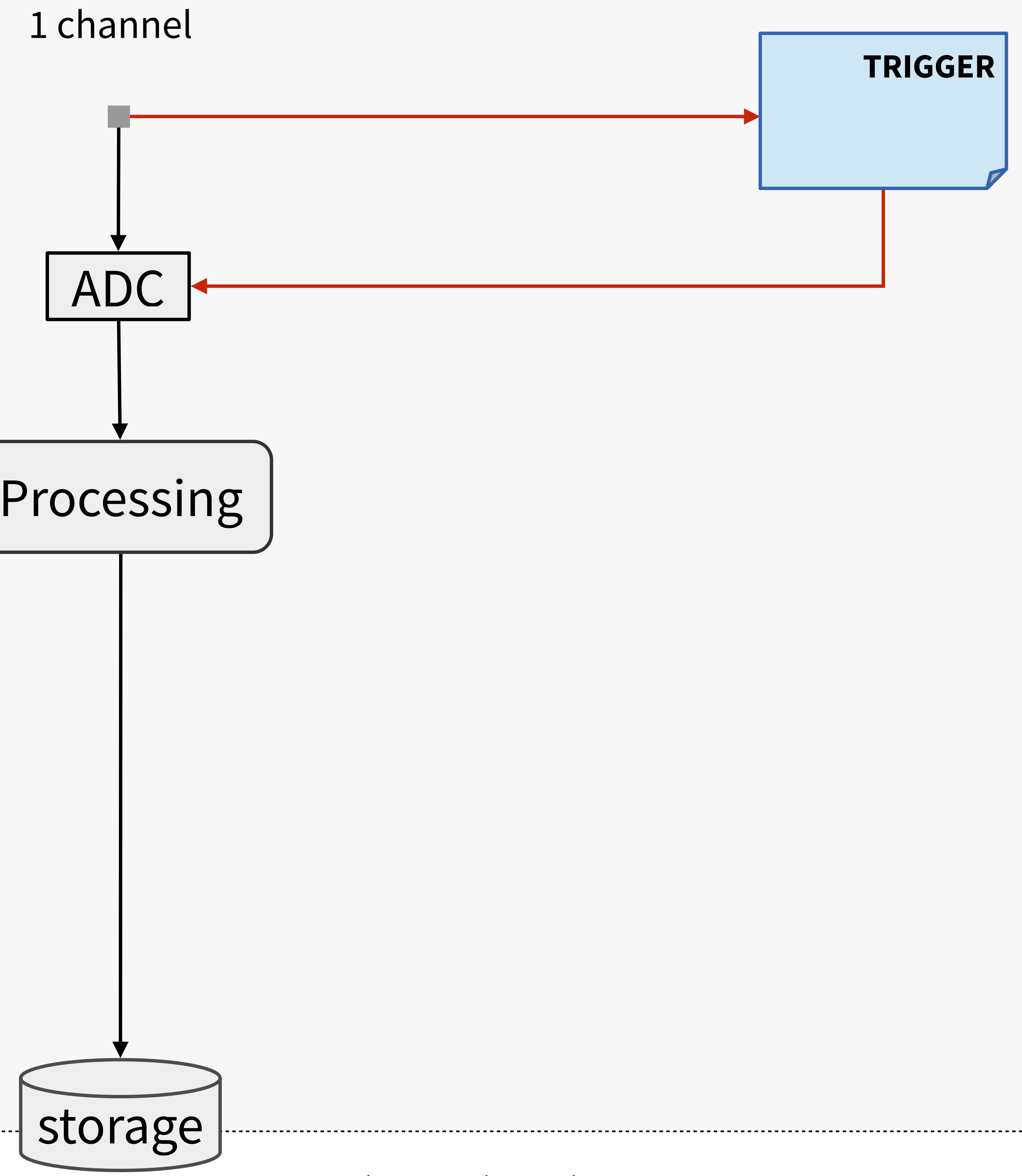
- Eg: accept n events every m bunch crossings
- Eg: prevent some dangerous trigger patterns



Outline

Adding more channels

Adding more channels requires a hierarchical structure committed to the data handling and conveyance



Front-End | data digitization, data **buffering**

Readout | data extraction, data formatting, data **buffering**

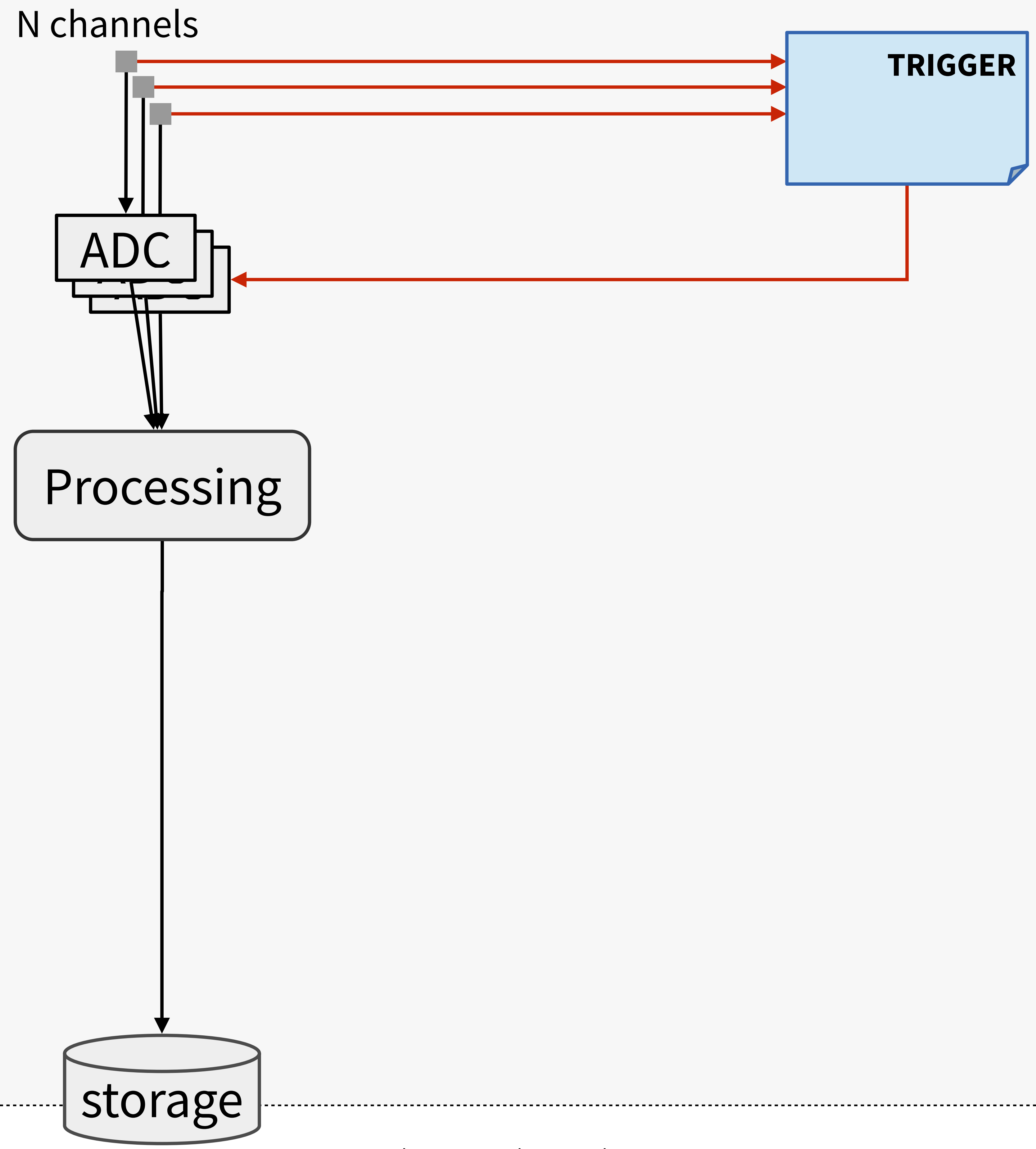
Event Building | event assembly, event **buffering**

Event Filtering | event rejection, event **buffering**

Event Logging | file storage, file **buffering**

Adding more channels

Adding more channels requires a hierarchical structure committed to the data handling and conveyance



Front-End | data digitization, data **buffering**

Readout | data extraction, data formatting, data **buffering**

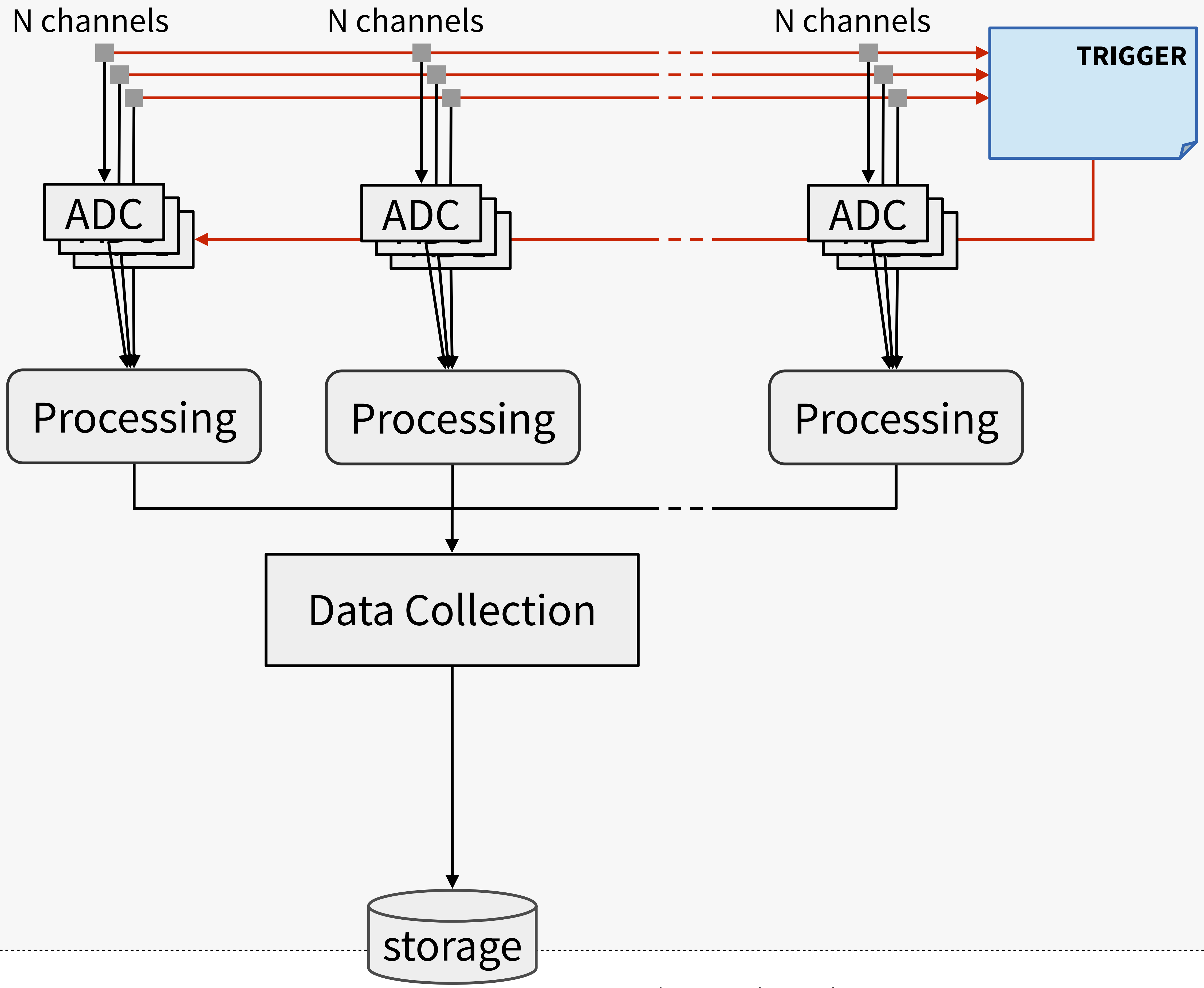
Event Building | event assembly, event **buffering**

Event Filtering | event rejection, event **buffering**

Event Logging | file storage, file **buffering**

Adding more channels

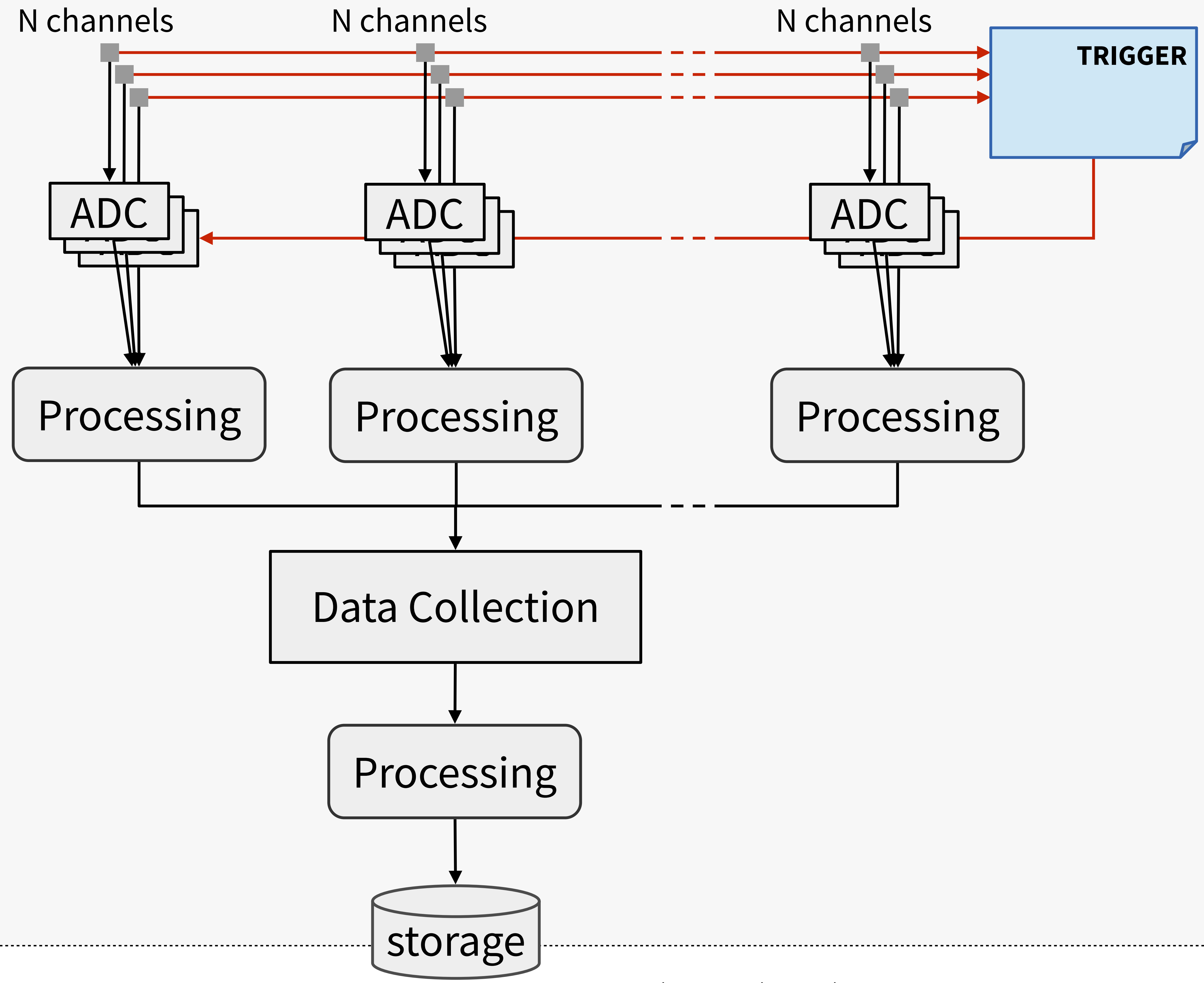
Adding more channels requires a hierarchical structure committed to the data handling and conveyance



Front-End	data digitization data **buffering**
Readout	data extraction data formatting data **buffering**
Event Building	event assembly event **buffering**
Event Filtering	event rejection event **buffering**
Event Logging	file storage file **buffering**

Adding more channels

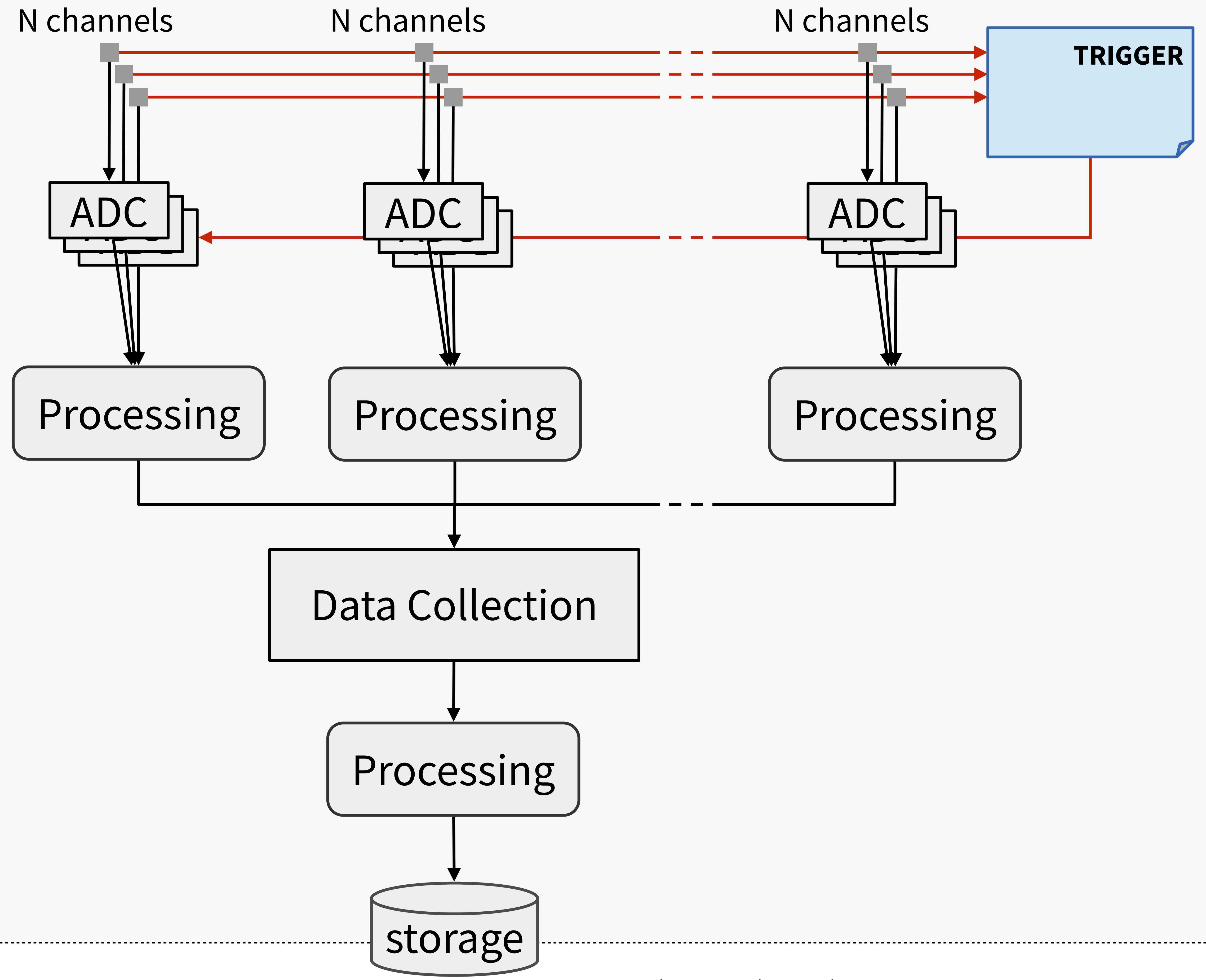
Adding more channels requires a hierarchical structure committed to the data handling and conveyance



Front-End	data digitization data **buffering**
Readout	data extraction data formatting data **buffering**
Event Building	event assembly event **buffering**
Event Filtering	event rejection event **buffering**
Event Logging	file storage file **buffering**

Adding more channels

Adding more channels requires a hierarchical structure committed to the data handling and conveyance

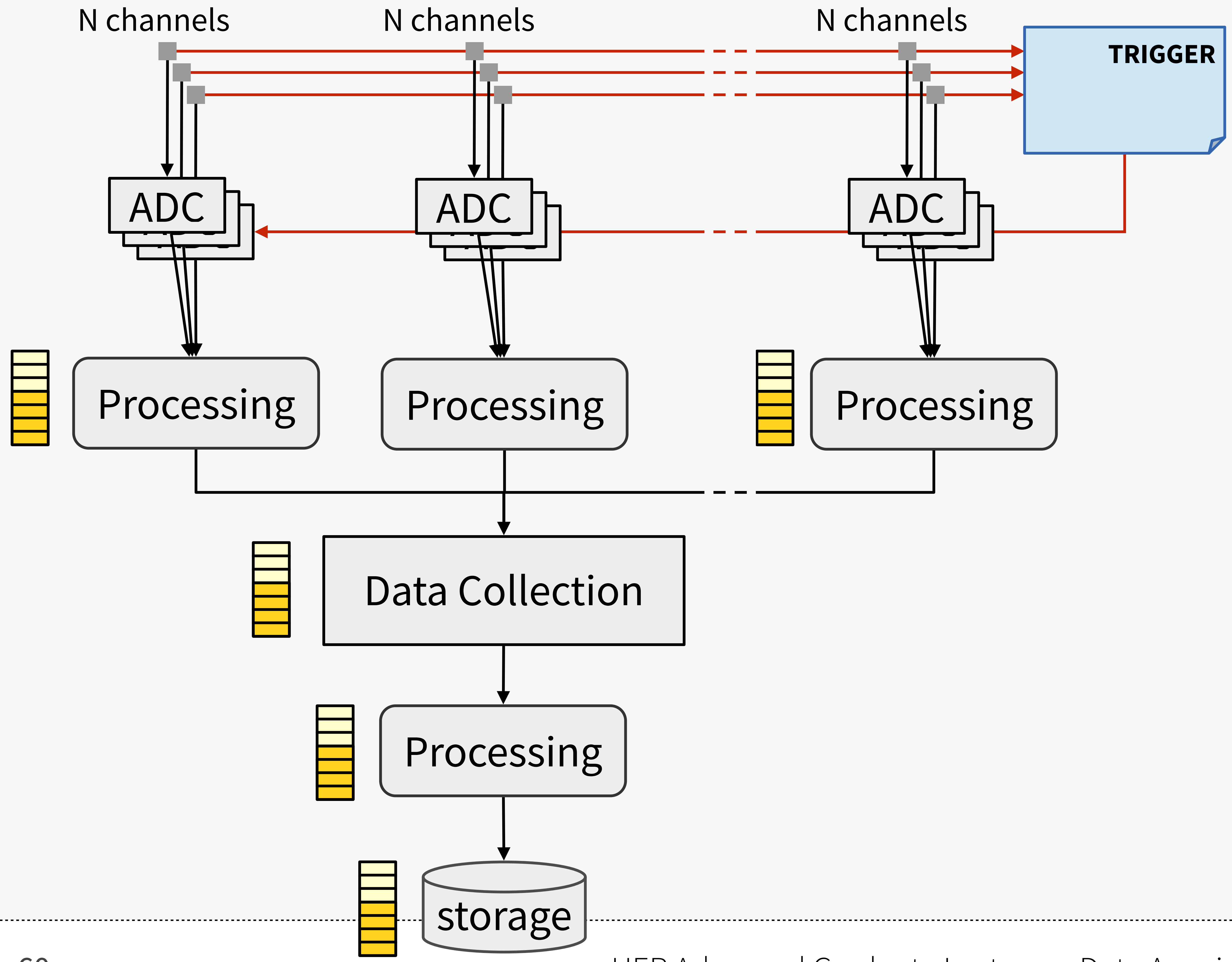


Front-End	data digitization data **buffering**
Readout	data extraction data formatting data **buffering**
Event Building	event assembly event **buffering**
Event Filtering	event rejection event **buffering**
Event Logging	file storage file **buffering**

Adding more channels

Buffering usually needed at every level

- DAQ can be seen as a multi level buffering system

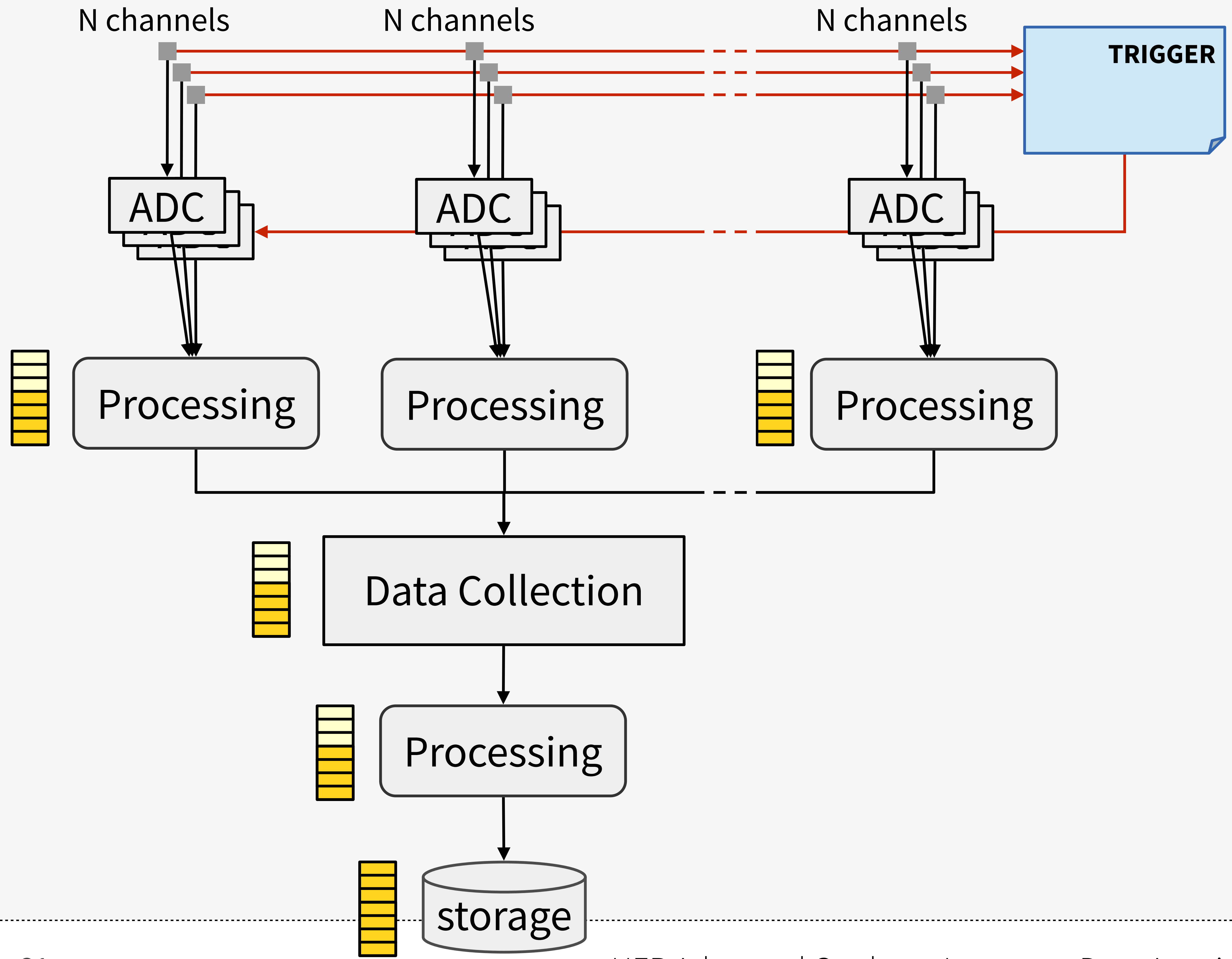


Front-End	data digitization data **buffering**
Readout	data extraction data formatting data **buffering**
Event Building	event assembly event **buffering**
Event Filtering	event rejection event **buffering**
Event Logging	file storage file **buffering**

Backpressure

If a system/buffer gets saturated

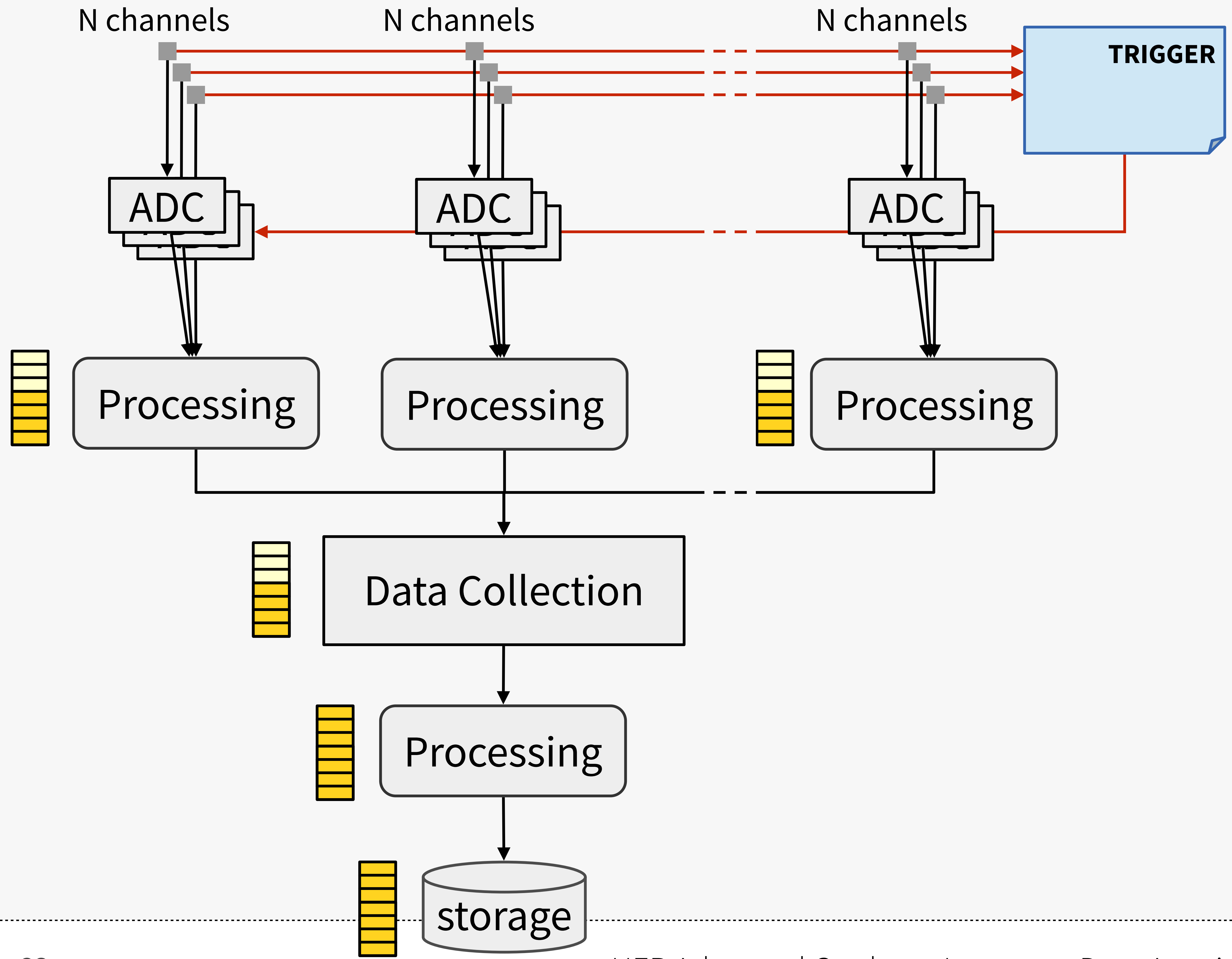
- the "pressure" is propagated upstream (**back-pressure**)



Backpressure

If a system/buffer gets saturated

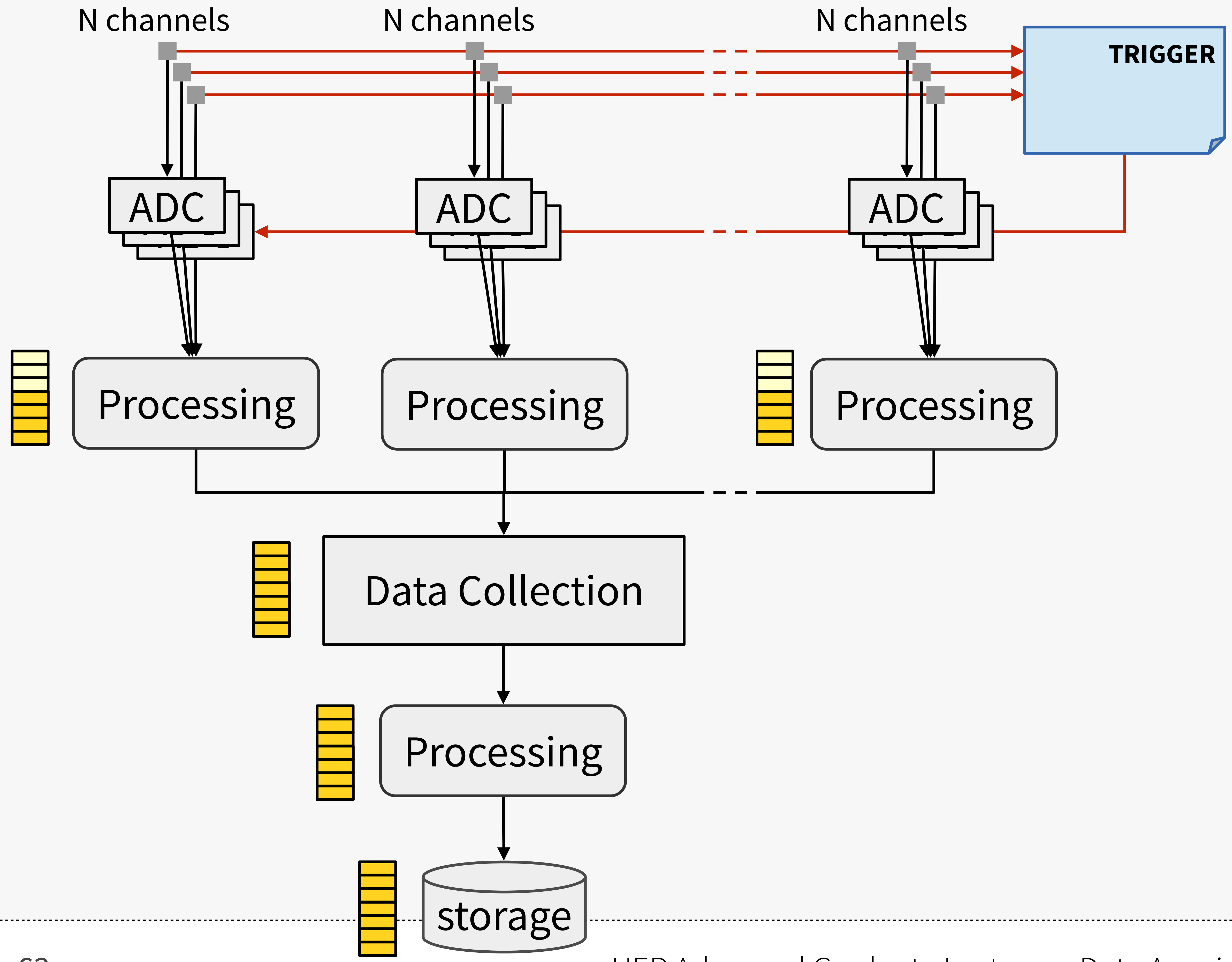
- the “pressure” is propagated upstream (**back-pressure**)



Backpressure

If a system/buffer gets saturated

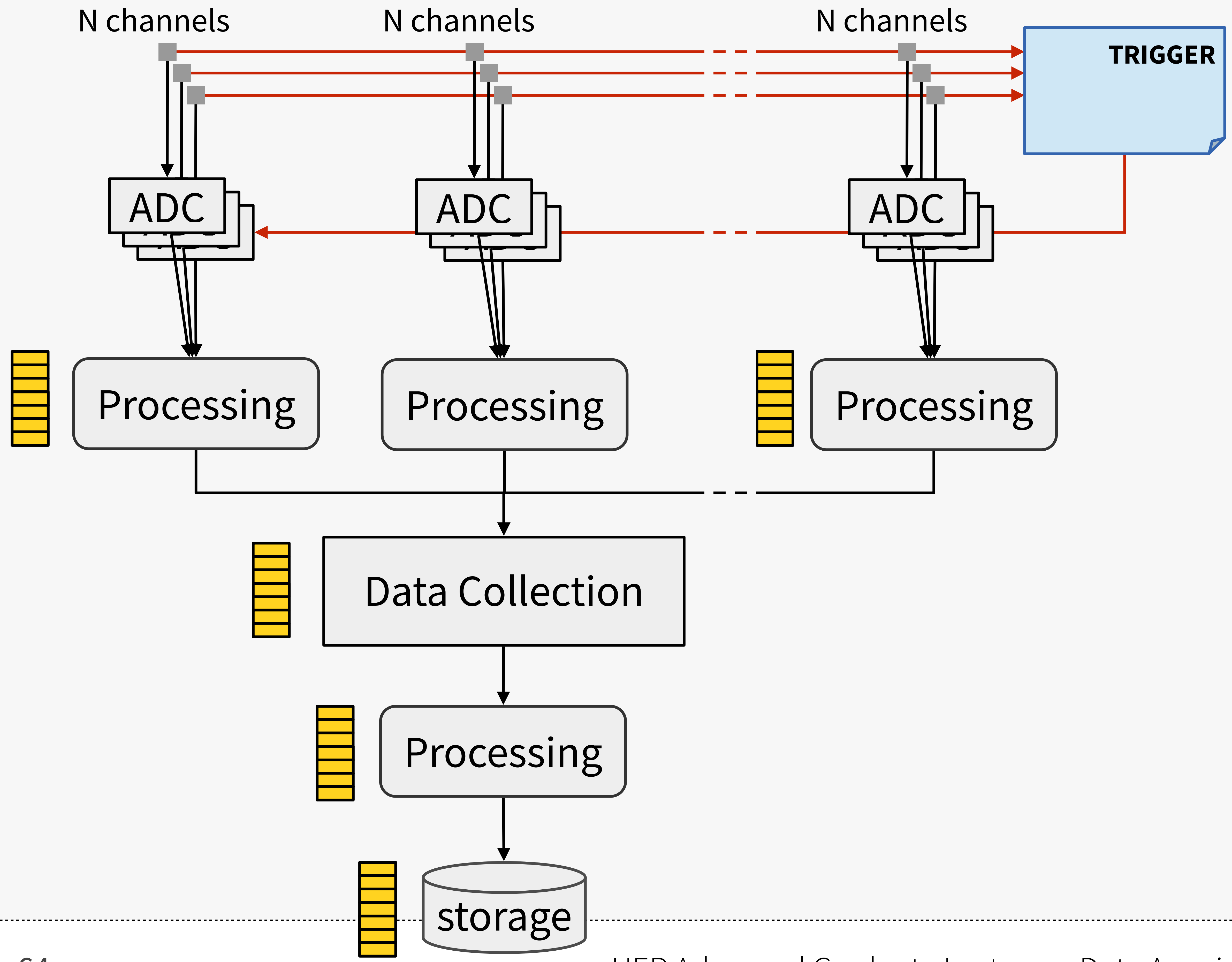
- the "pressure" is propagated upstream (**back-pressure**)



Backpressure

If a system/buffer gets saturated

- the "pressure" is propagated upstream (**back-pressure**)

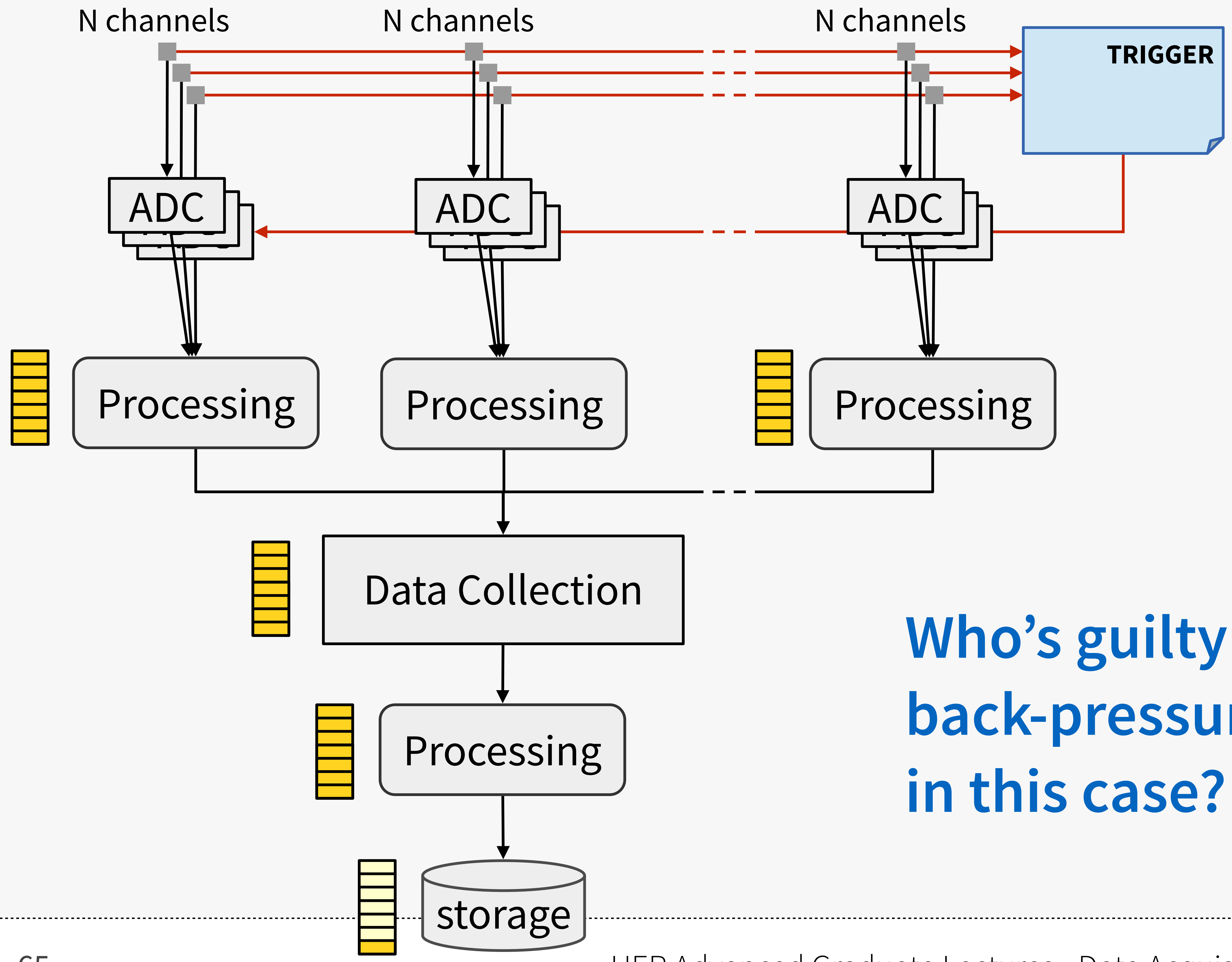


- Up to exert **busy** to the trigger system
- **Debugging**: where is the source of back-pressure?
  - follow the buffers occupancy via the monitoring system

Backpressure

If a system/buffer gets saturated

- the “pressure” is propagated upstream (**back-pressure**)

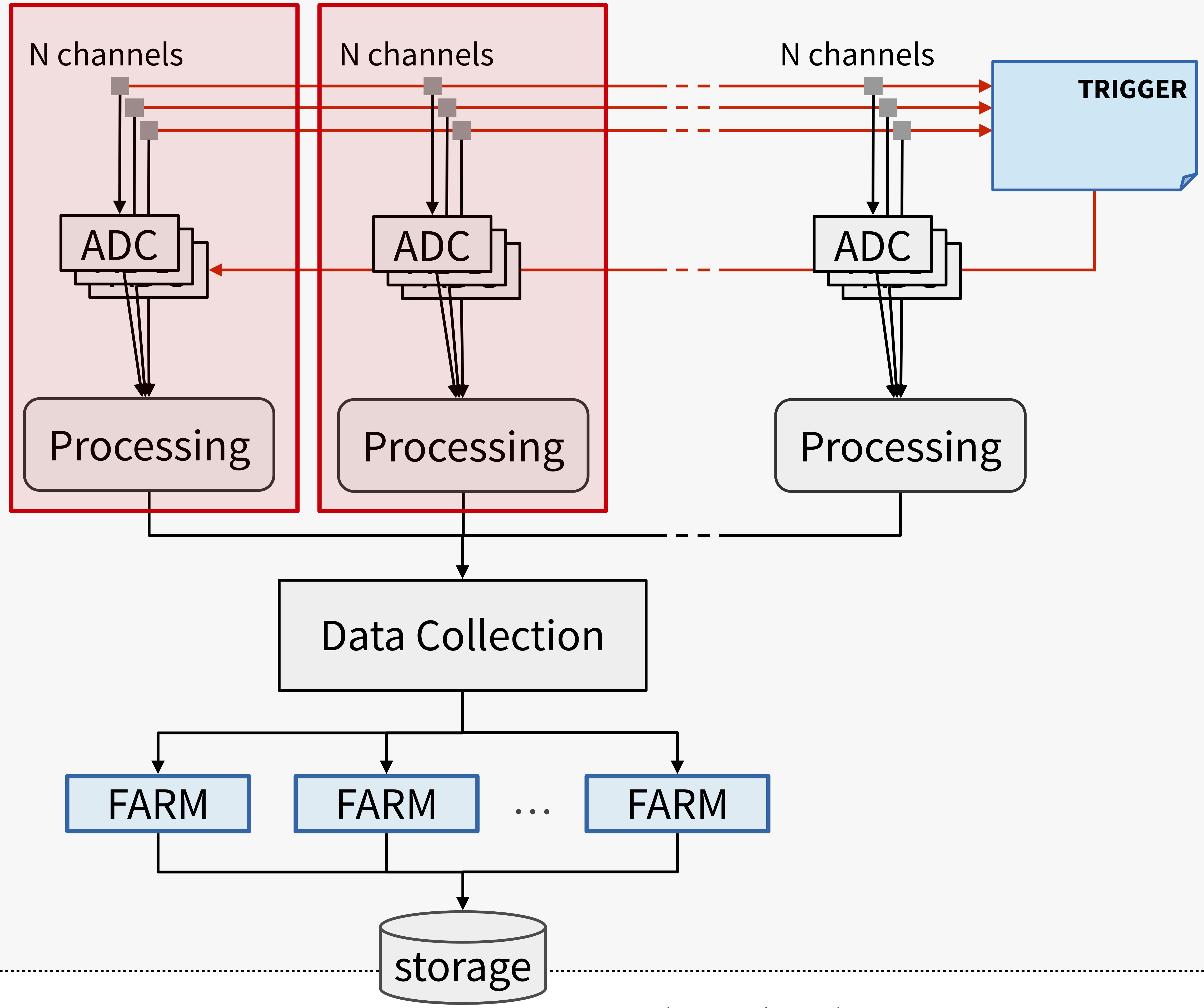


- Up to exert **busy** to the trigger system
- **Debugging**: where is the source of back-pressure?
  - follow the buffers occupancy via the monitoring system

Who’s guilty of back-pressure in this case?

Building blocks

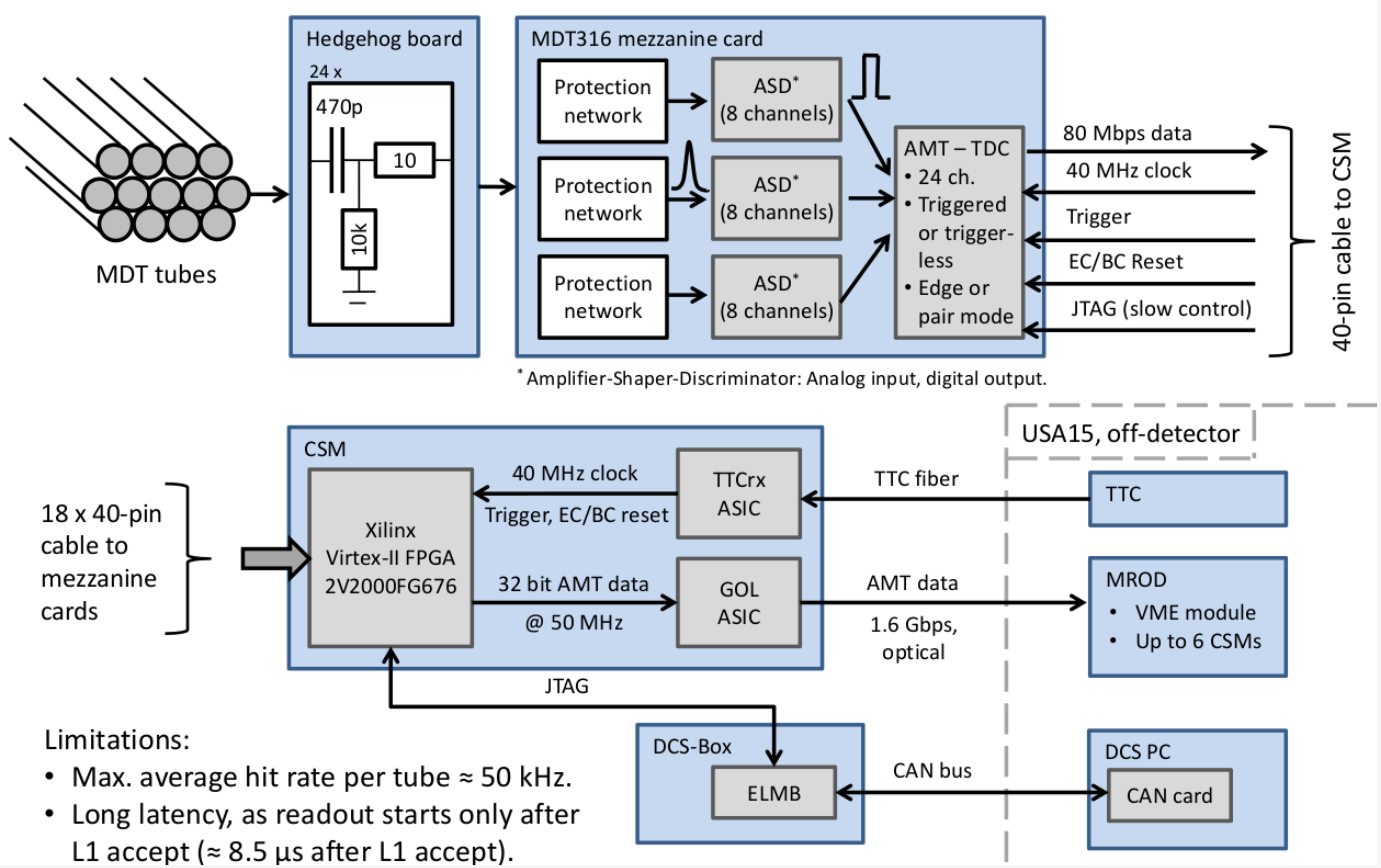
Reading out data or building events out of many channels requires many components



In the design of our hierarchical data-collection system, it's convenient to define "**building blocks**"

- **Readout crates**
- **HLT racks**
- **event building groups**
- **daq slices**

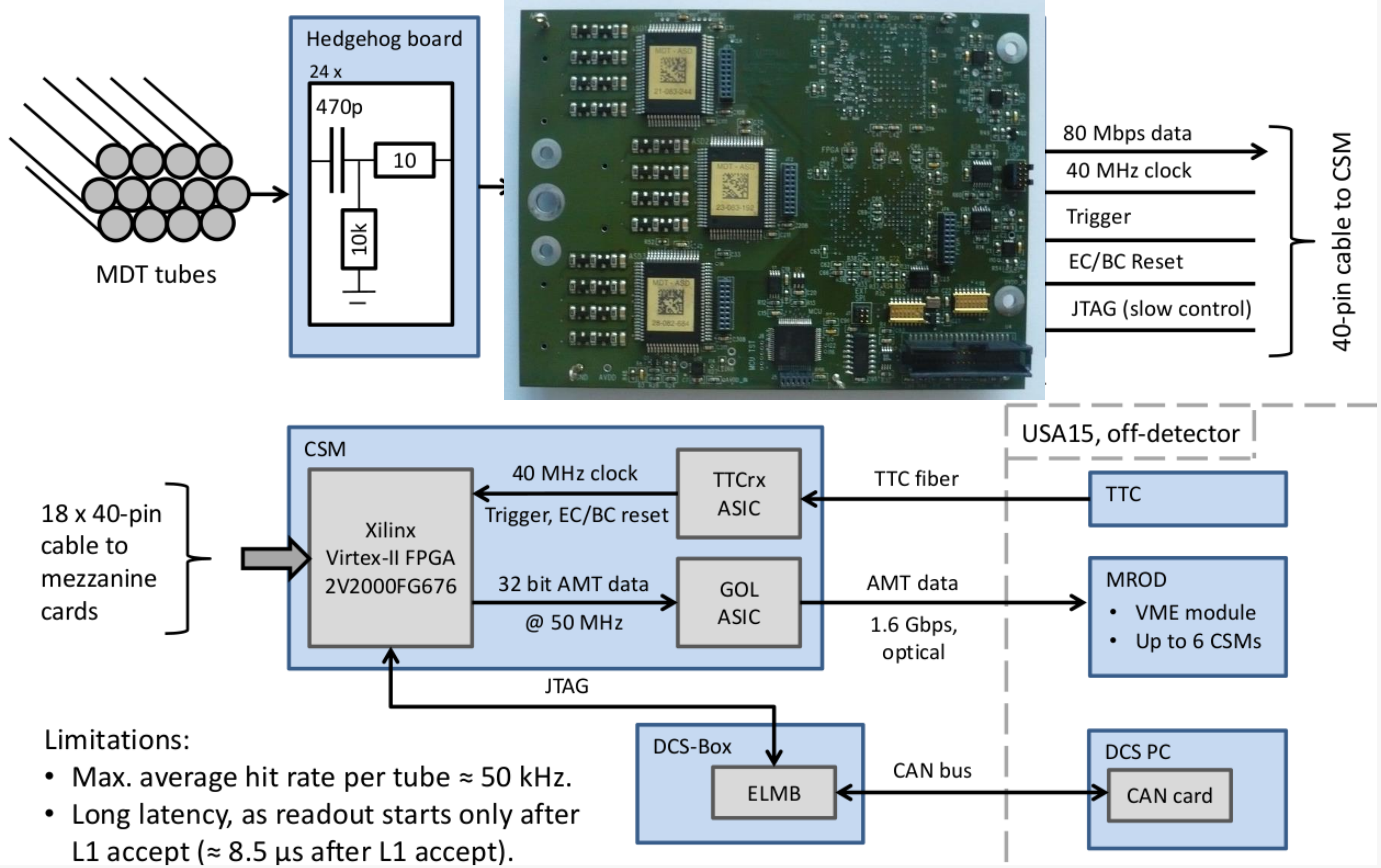
Front End electronics



Limitations:

- Max. average hit rate per tube ≈ 50 kHz.
- Long latency, as readout starts only after L1 accept (≈ 8.5 µs after L1 accept).

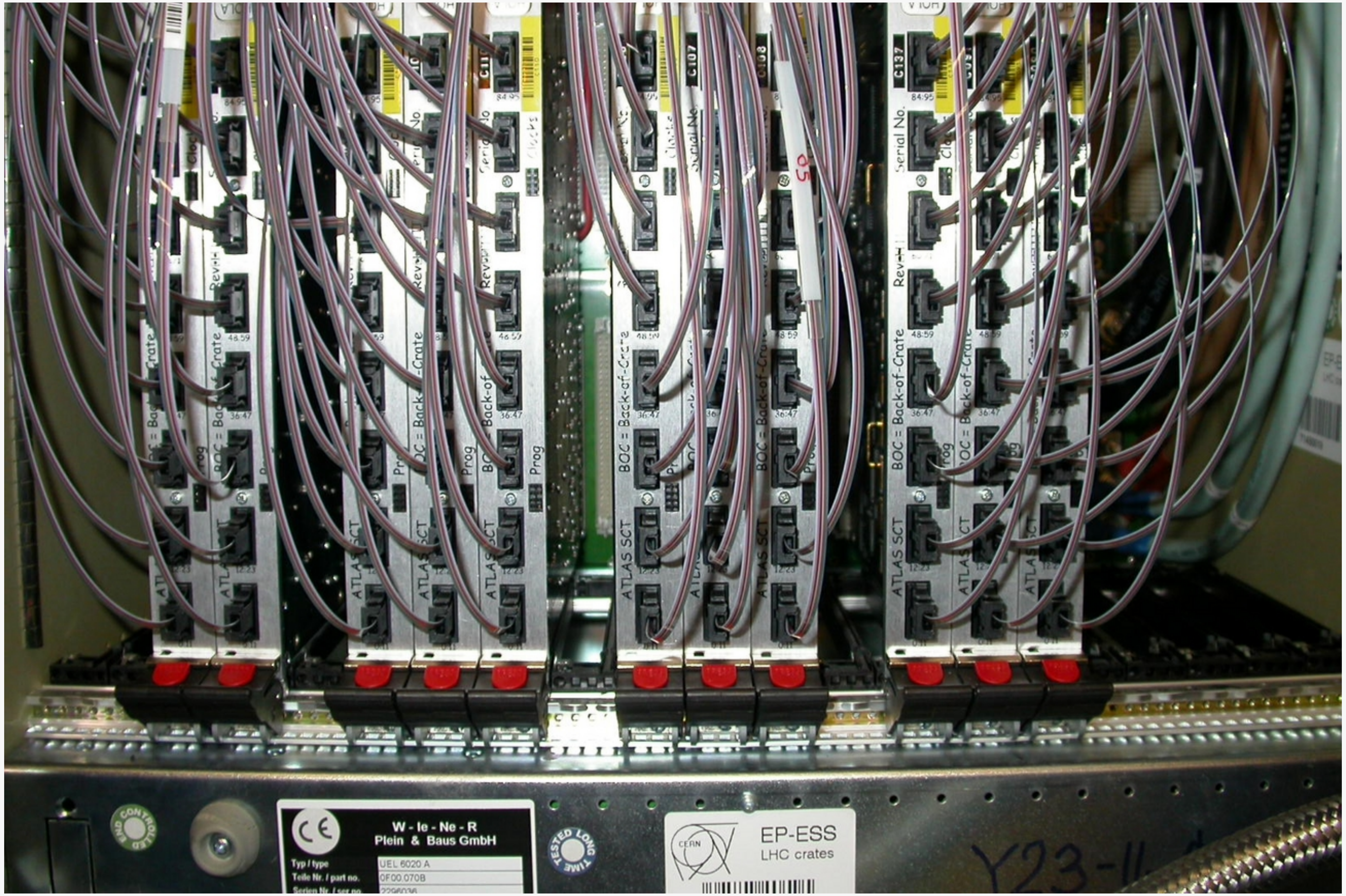
Front End electronics



Limitations:

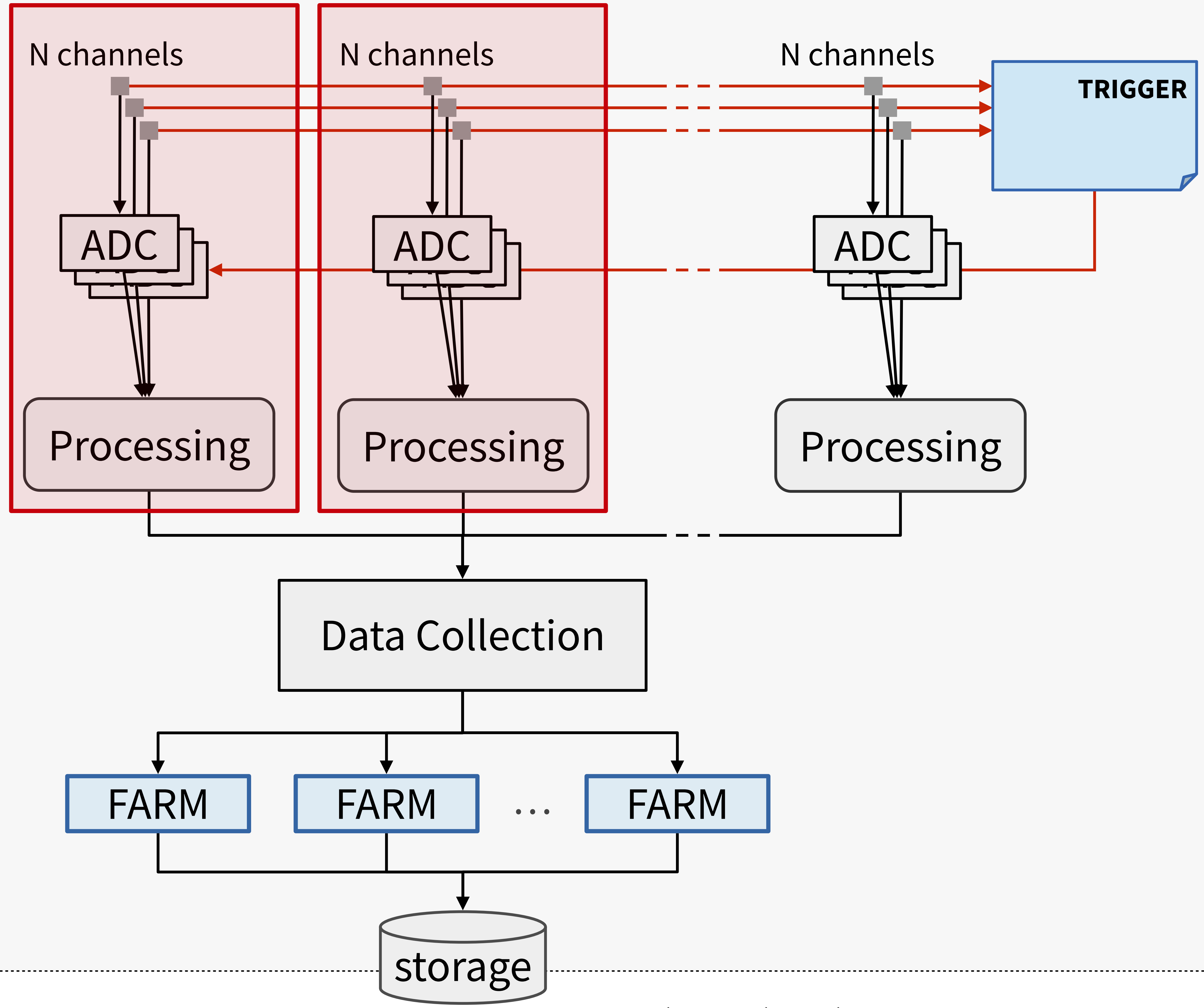
- Max. average hit rate per tube ≈ 50 kHz.
- Long latency, as readout starts only after L1 accept (≈ 8.5 μs after L1 accept).

Readout Boards (Counting Room)



Building blocks

Reading out data or building events out of many channels requires many components



In the design of our hierarchical data-collection system, it's convenient to define "**building blocks**"

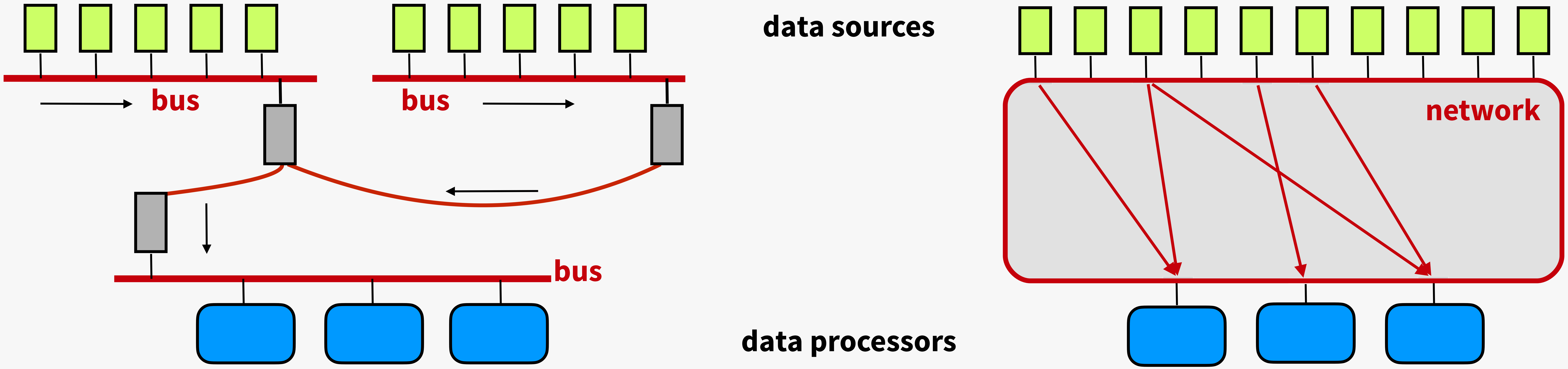
- **Readout crates**
- **HLT racks**
- **event building groups**
- **daq slices**

Farm (@surface)

Readout Topology

How to organize interconnections inside the building blocks and between building blocks?

- How to connect data sources and data destinations?
- Two main classes: **bus** or **network**



Buses

Devices connected via a **shared bus**

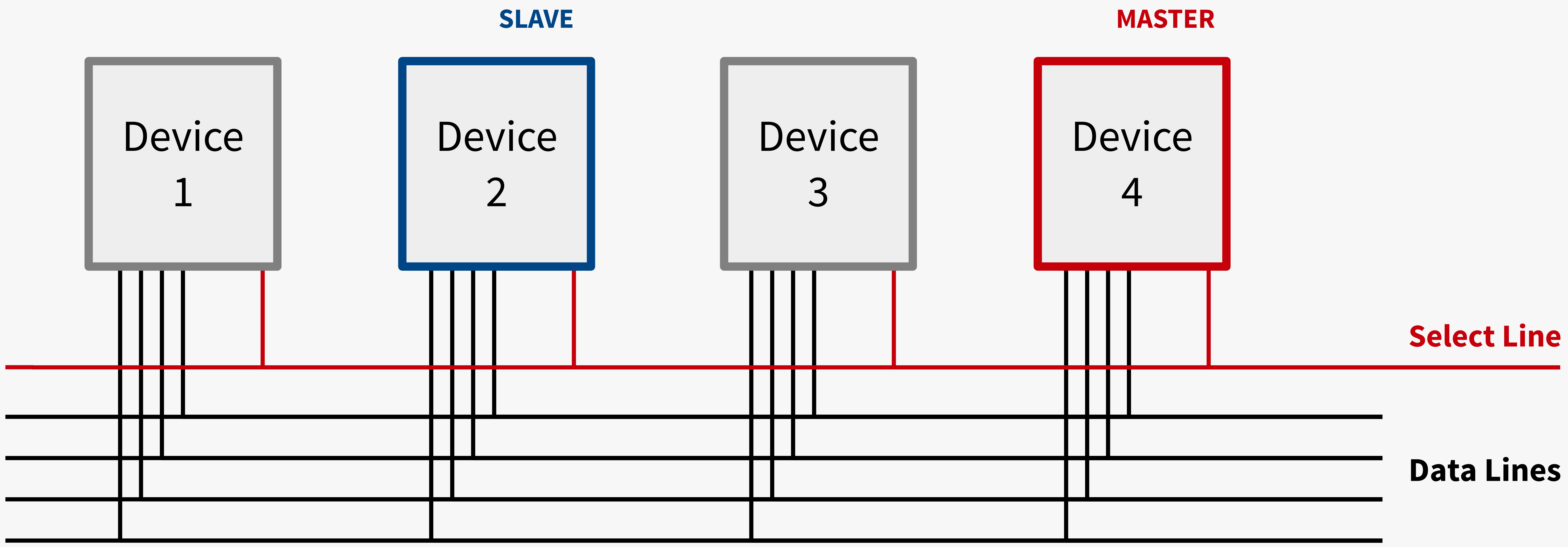
- Bus → group of electrical lines

Sharing implies **arbitration**

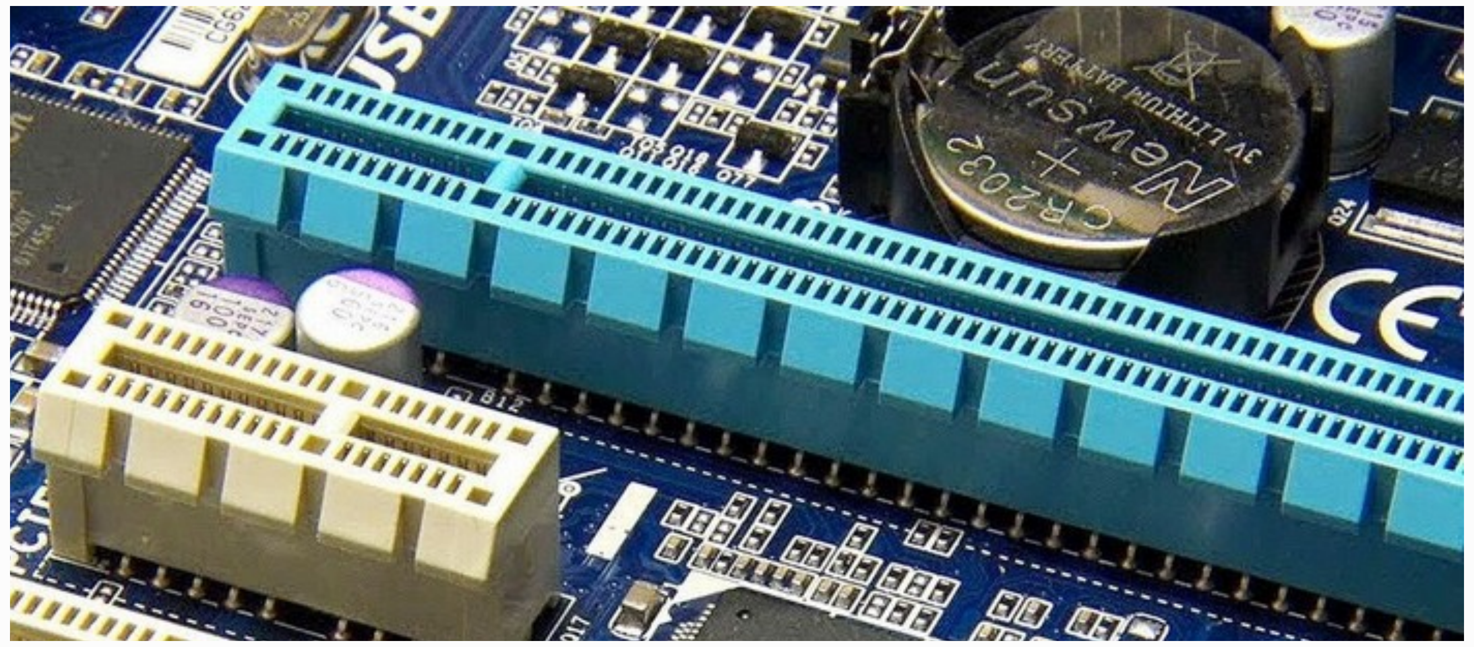
- Devices can be **master** or **slave**
- Devices can be addresses (uniquely identified) on the bus

E.g.: SCSI, Parallel ATA, VME, PCI …

- local, external, crate, long distance, …

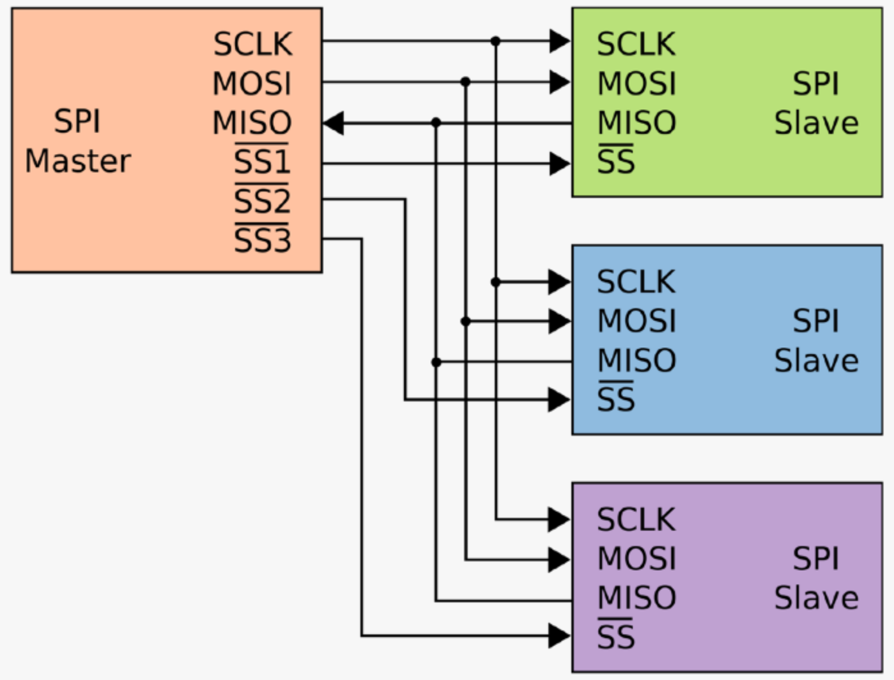


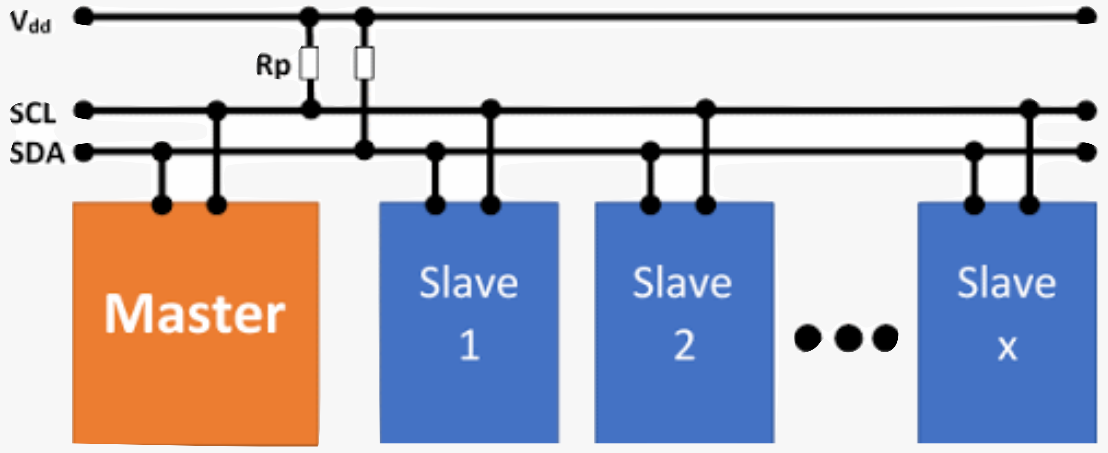
Bus examples (some)



- I2C
- SPI
- UART
- PCI express

- VME
- μTCA
- ATCA

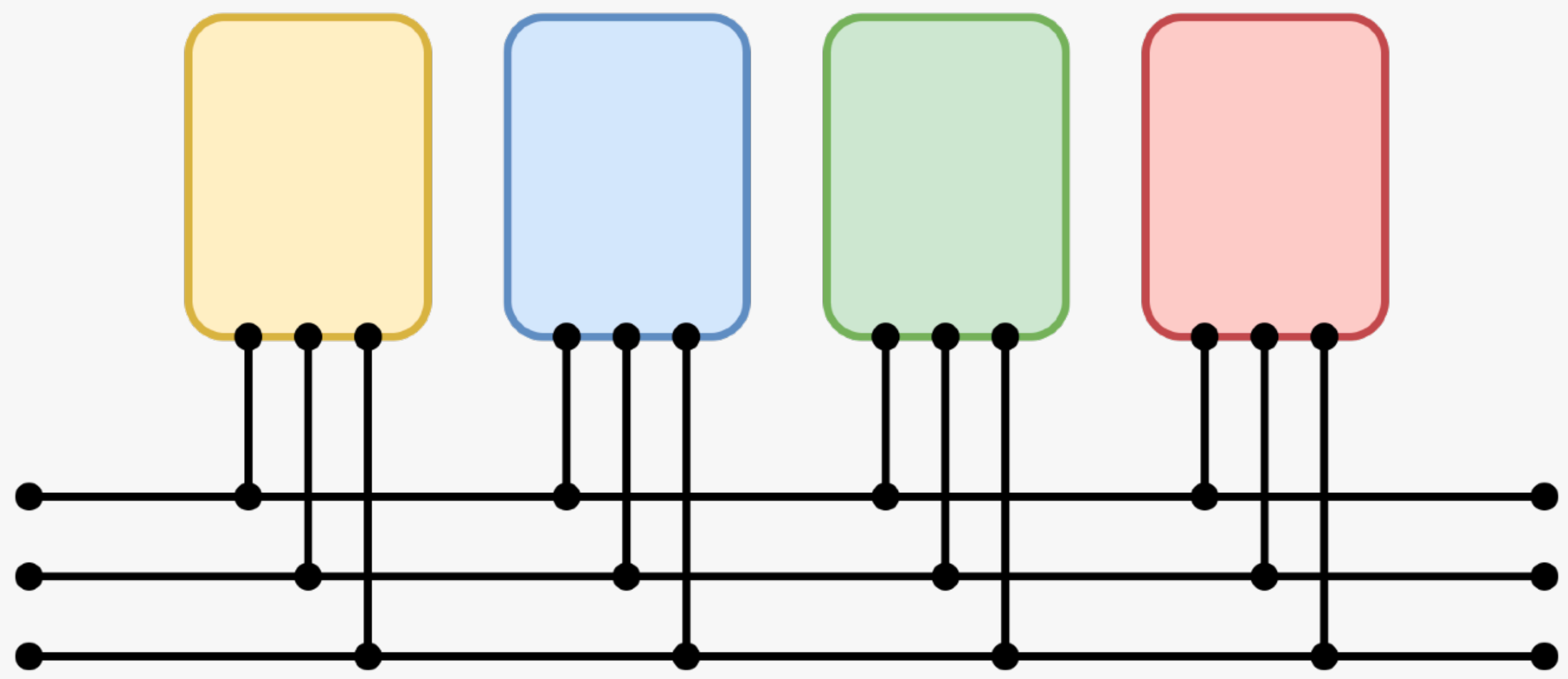




Bus facts

Simple :-)

- Fixed number of lines (bus-width)
- Devices have to follow well defined interfaces
  - Mechanical, electrical, communication, ...



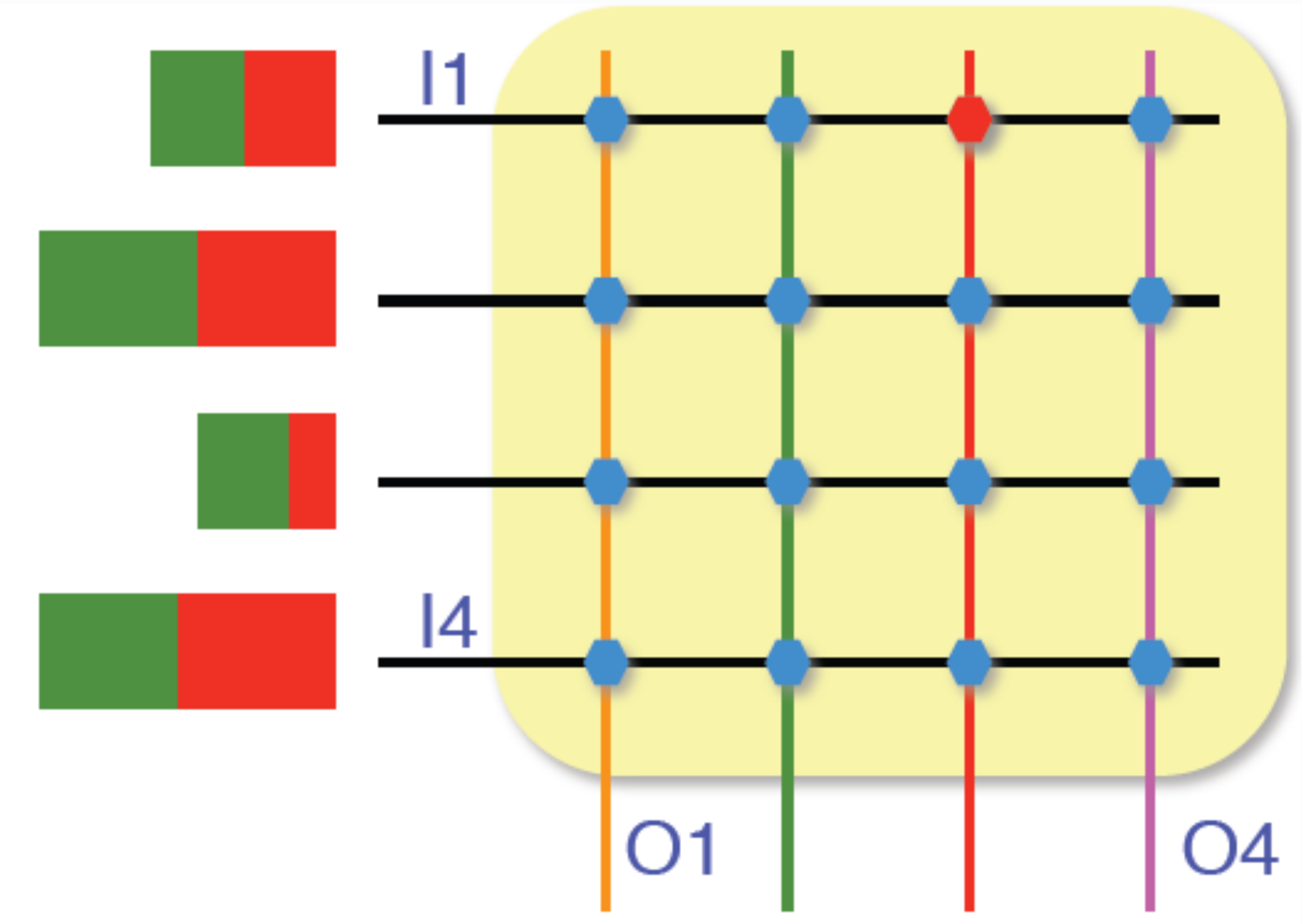
Scalability issues :-(

- Bus bandwidth is shared among all the devices
- Maximum bus width is limited
- Maximum number of devices depends on bus length
- Maximum bus frequency is inversely proportional to the bus length
- On the long term, second order “effects” may limit the scalability of your system

Bus facts

Some second order “effects”

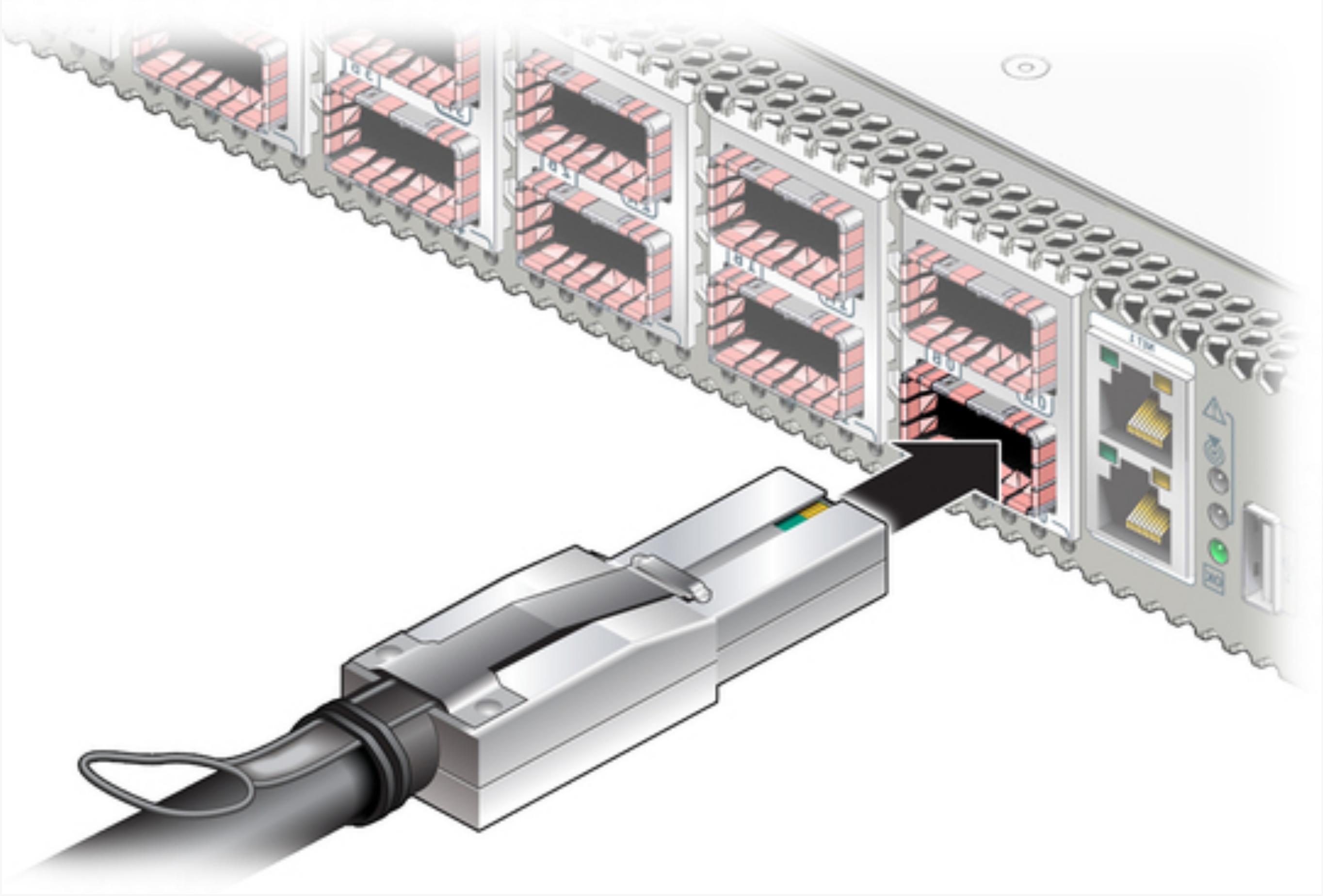
Networks



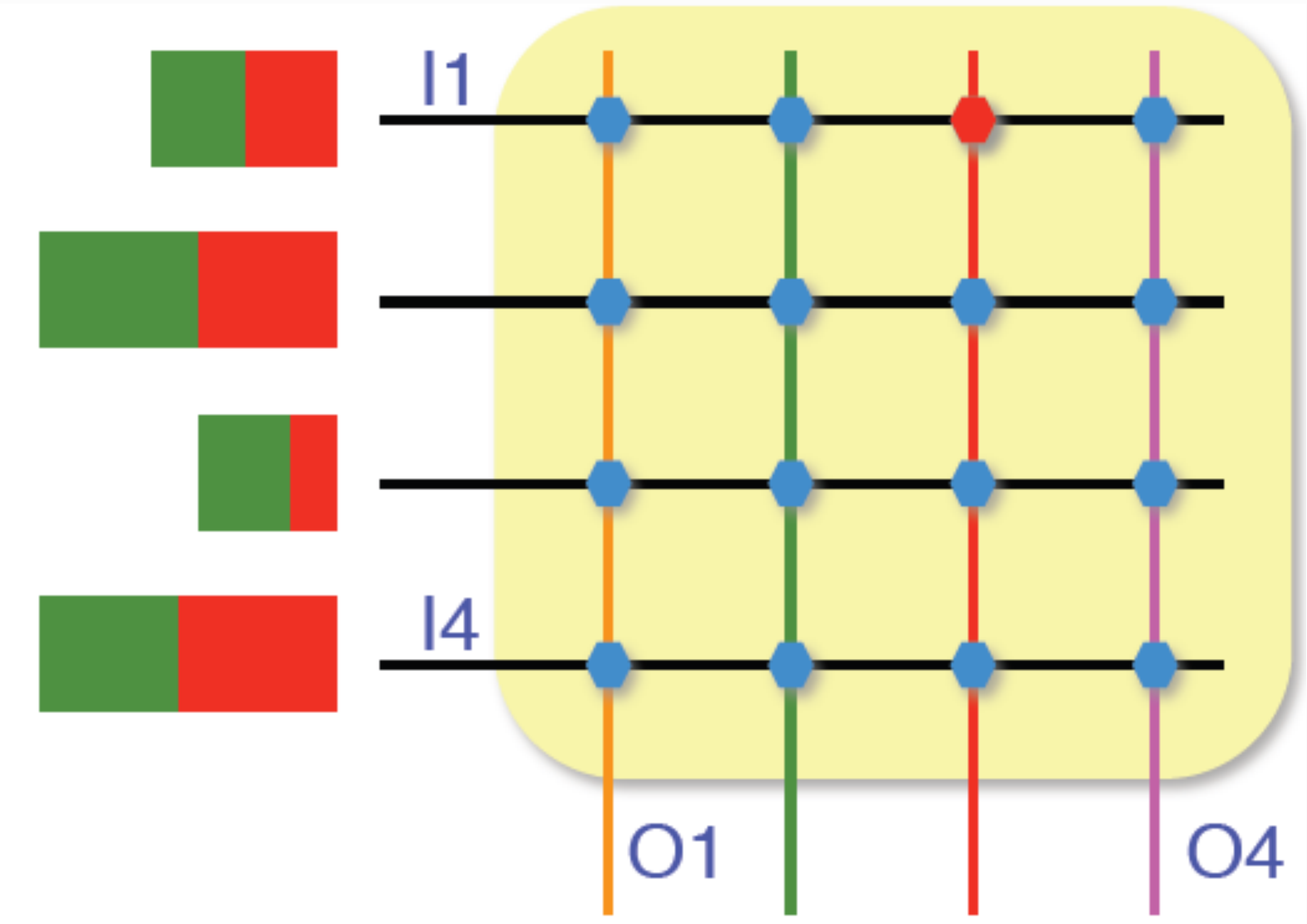
All devices are **equal (peers)**

- They **communicate directly** with each other via messages
  - No arbitration
  - Bandwidth guaranteed
- Not just copper: optical, wireless

Eg: Telephone, Ethernet, Infiniband, …



Networks



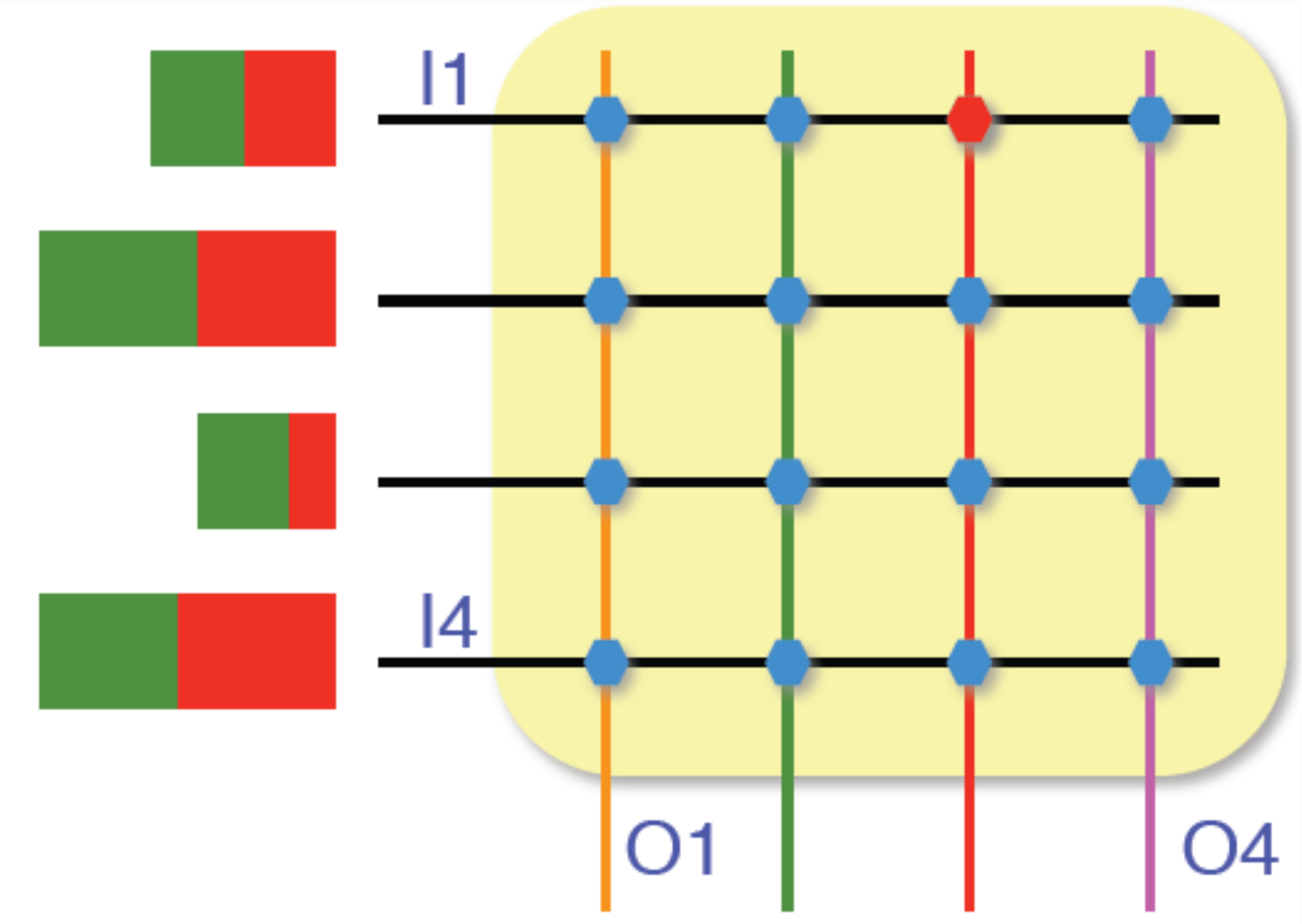
In switched networks, **switches** move messages between sources and destinations

- Finds the right path between endpoints

How **congestions** (two messages with the same destination at the same time) are handled?

- The key is ….

Networks



In switched networks, **switches** move messages between sources and destinations

- Finds the right path between endpoints

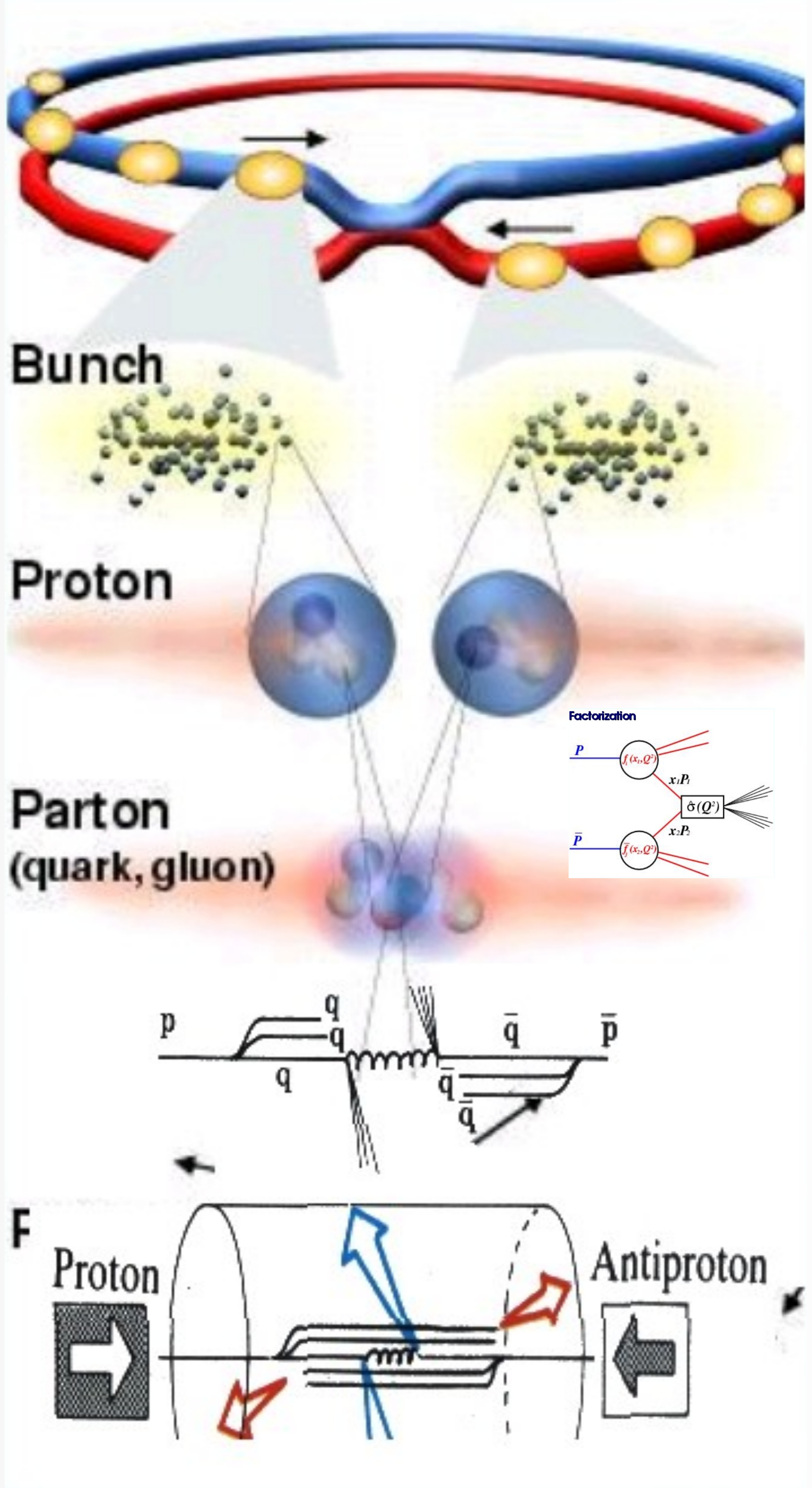
How **congestions** (two messages with the same destination at the same time) are handled?

- The key is …. **buffering**

Networks

Cable management is still a thing.

LHC and its products



2835×2835 bunches in the LHC ring

10^{11} protons / bunch

≤30 pp collisions per bunch crossing (BC)

N parton-parton collisions / pp collision

Complex final-states in every parton-parton collision.

Design parameters

E_{cms} = 14 TeV
L = 10^{34} /cm² s
BC clock = 40 MHz

$$R = \sigma_{in} \times L$$

Interesting processes **extremely rare**, high Luminosity is essential

- **Close collisions in space and time**
  - Large proton bunches ($1.5\text{x}10^{11}$)
  - Fixed frequency: 40MHz (1/25ns)

Protons are **composite particles**

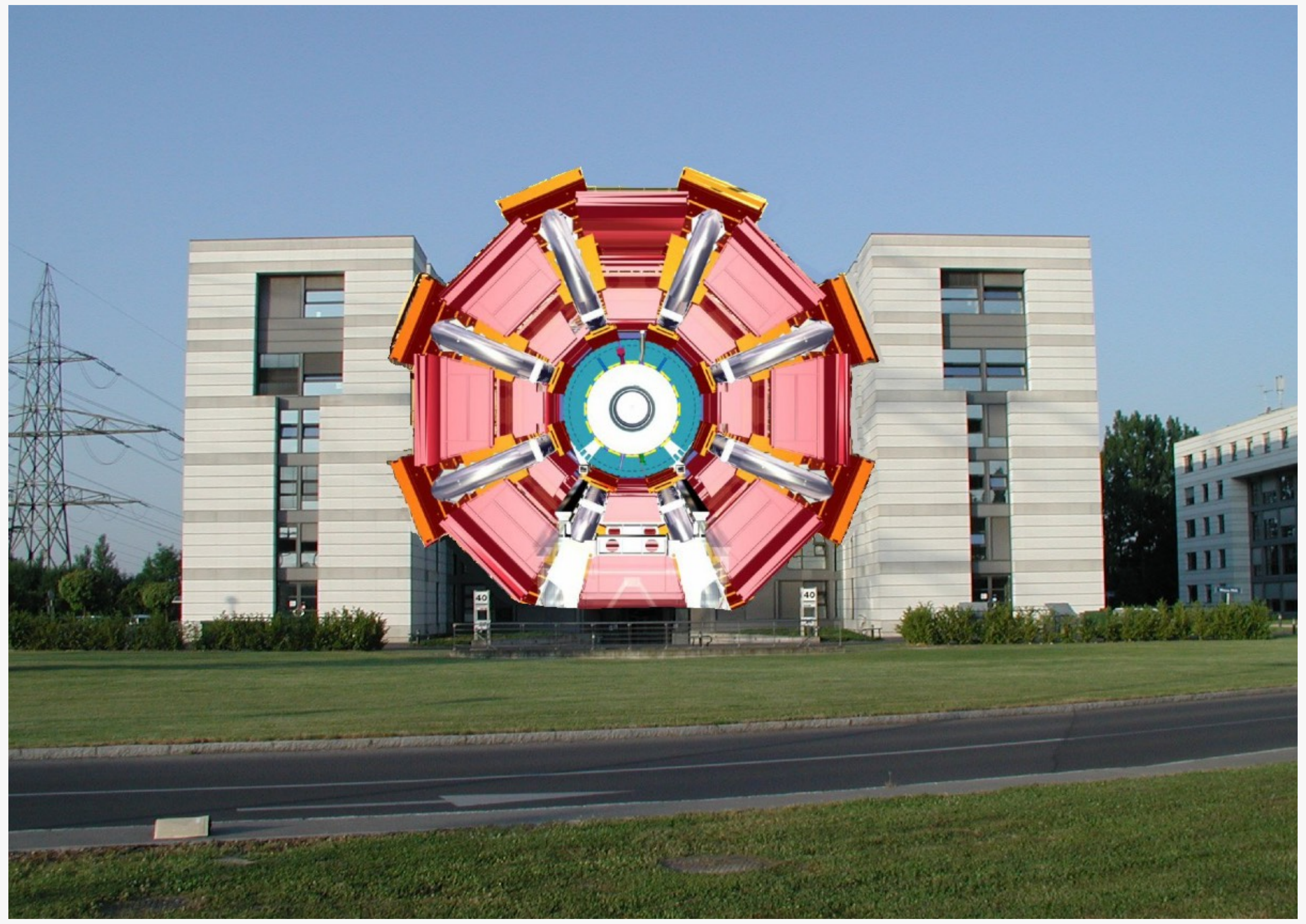
- abundant low energy interactions

Few rare high-E events overwhelmed in abundant low-E environment

LHC Detectors Challenges

Huge

- $O(10^6\text{-}10^8)$ channels
- ~1 MB event size for pp collisions
  - 50 MB for pb-pb collisions (Alice)
- Need huge number of connections



Fast and slow detectors

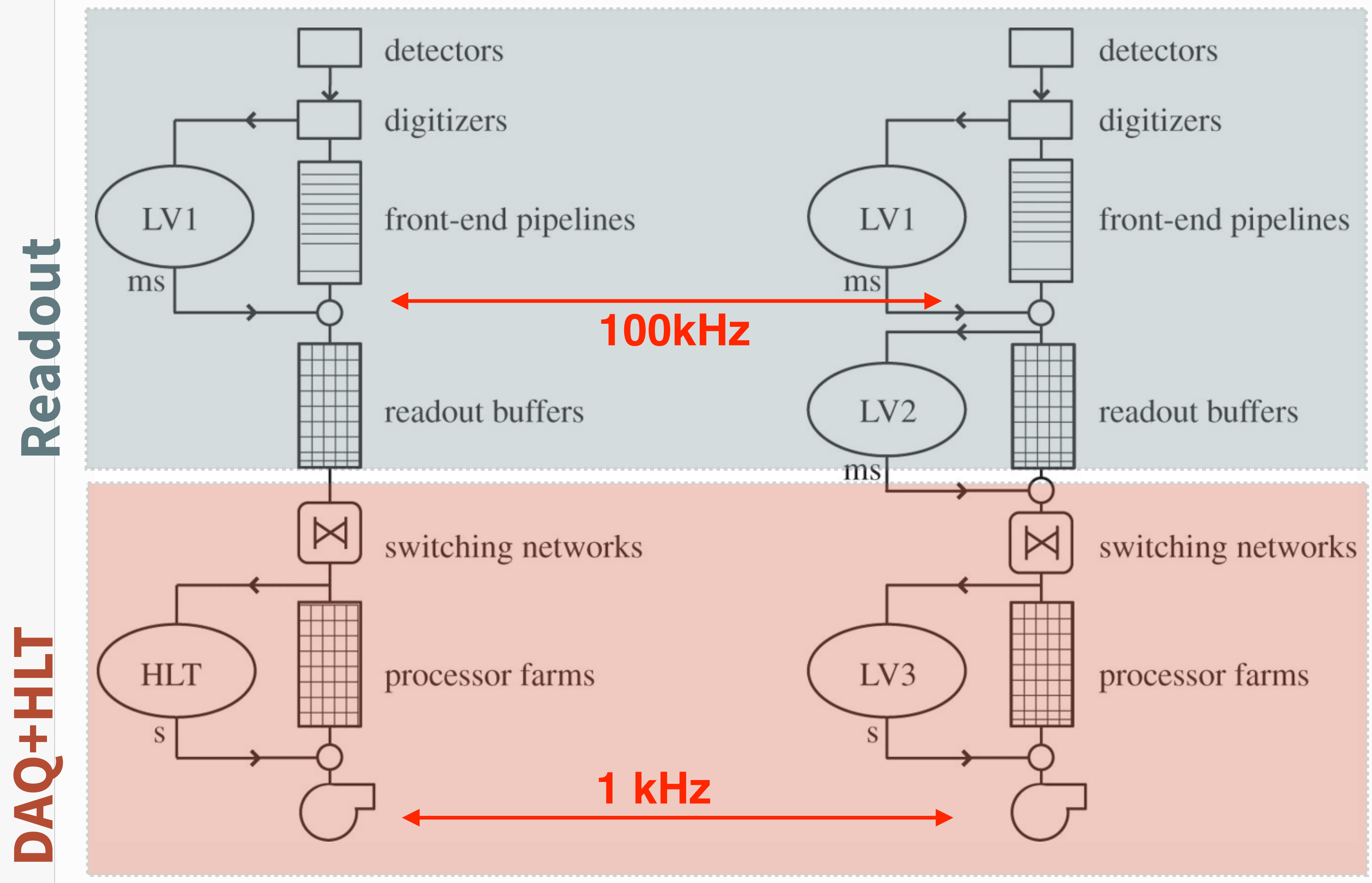
- Some detectors readout requires >25 ns and integrate more than one bunch crossing's worth of information
  - e.g. ATLAS LArg readout takes ~400 ns

Online, what is lost is lost forever

- Need to monitor selection - need very good control over all conditions

HLT/DAQ requirements

- Robustness and redundancy
- Scalability to adapt to Luminosity, detector evolving conditions
- Flexibility (> 10-years lifetime)
- Based on commercial products
- Limited cost



ATLAS/CMS Example

- 1 MB/event at 100 kHz for O(100ms) HLT latency
  - Network: 1 MB*100 kHz = 100 GB/s
  - HLT farm: 100 kHz*100 ms = O(10^4) CPU cores
- Intermediate steps (level-2) to reduce resources, at cost of complexity (at ms scale)

Prefer COTS hardware: PCs (linux based), Ethernet protocols, standard LAN, configurable devices

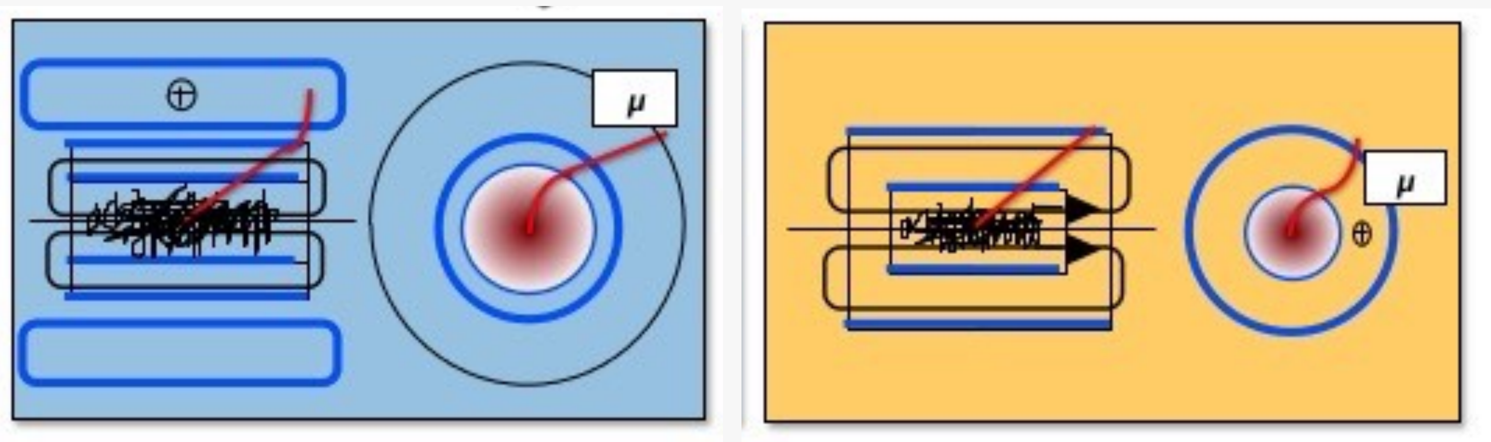
See S.Cittolin, DOI: 10.1098/rsta.2011.0464

ATLAS & CMS design principles

Same physics program

Different magnetic field structure

- **ATLAS**: 2 T solenoid + Toroids
- **CMS**: strong 4 T solenoid

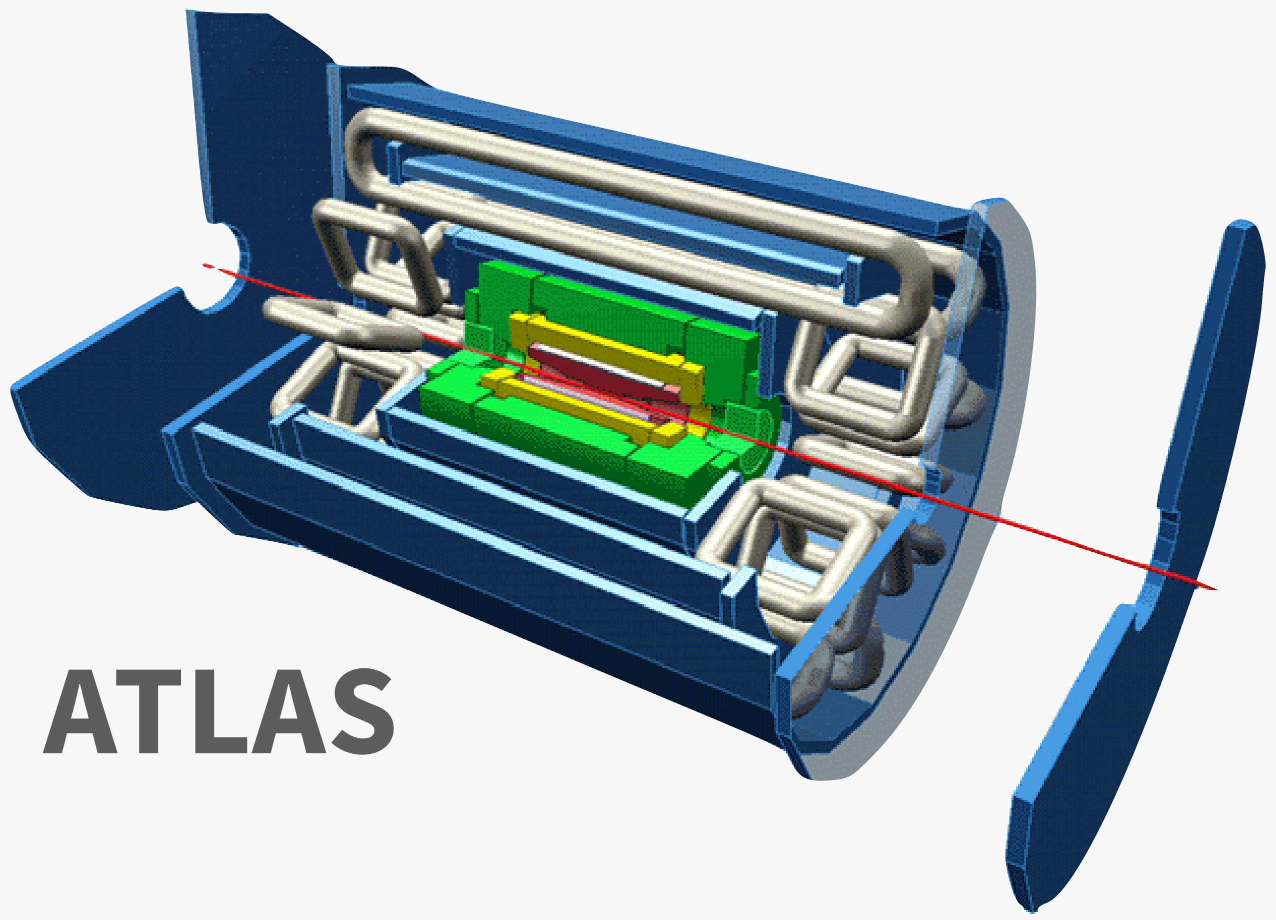


Different DAQ architecture

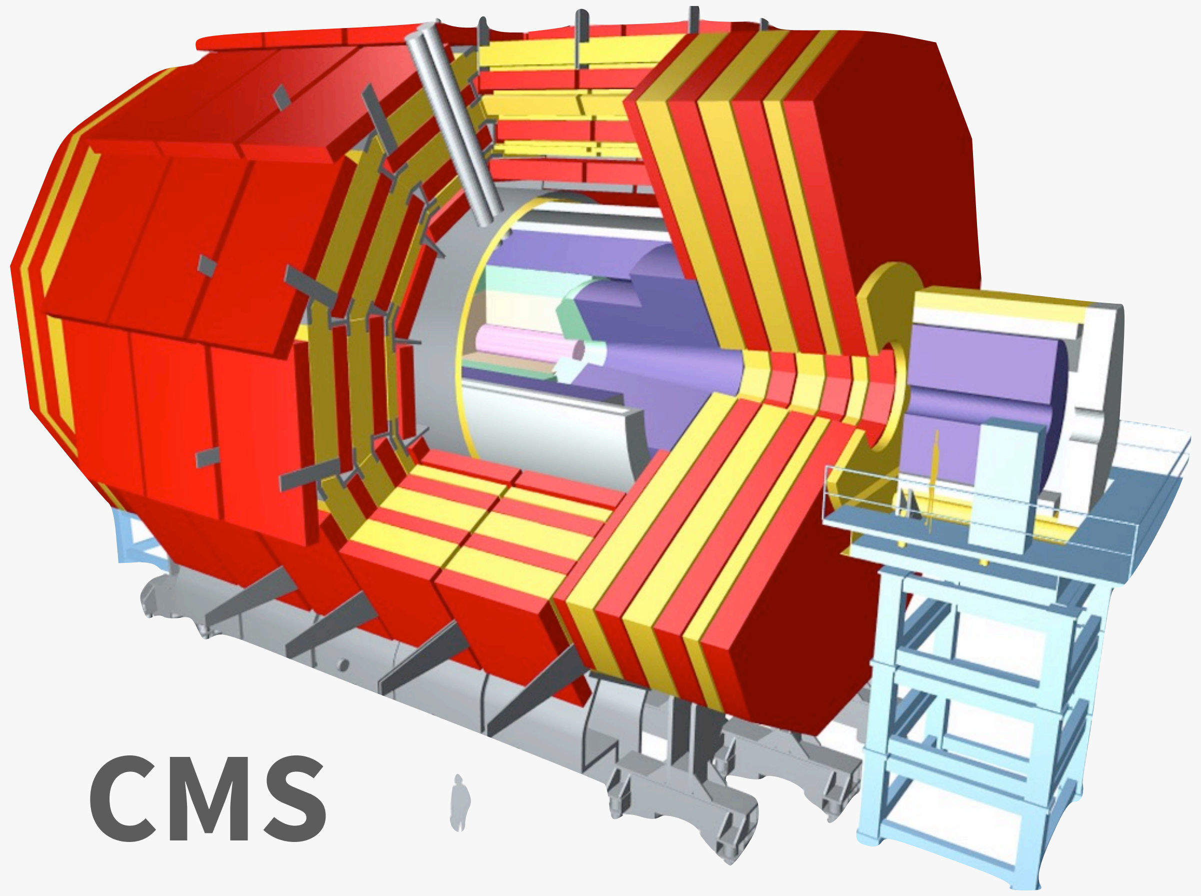
- **ATLAS**: minimise data flow bandwidth with multiple levels and regional readout
- **CMS**: large bandwidth, invest on commercial technologies for processing and communication

Same data rates

- ~1 MB * 100 kHz = ~100 GB/s readout network



ATLAS



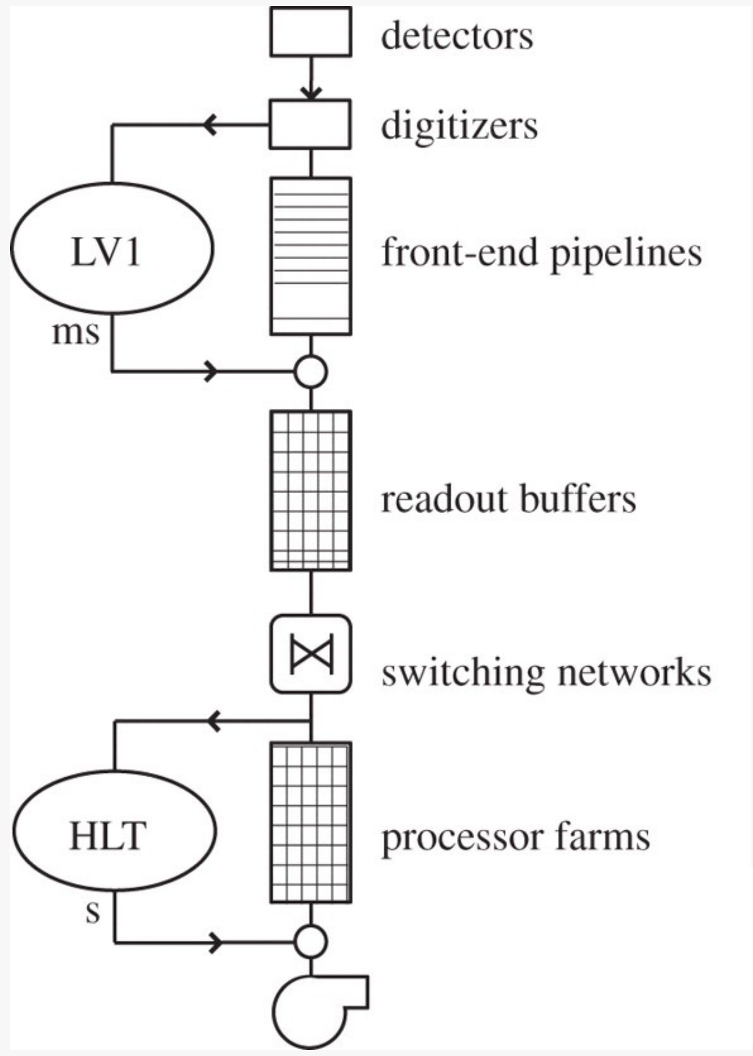
CMS

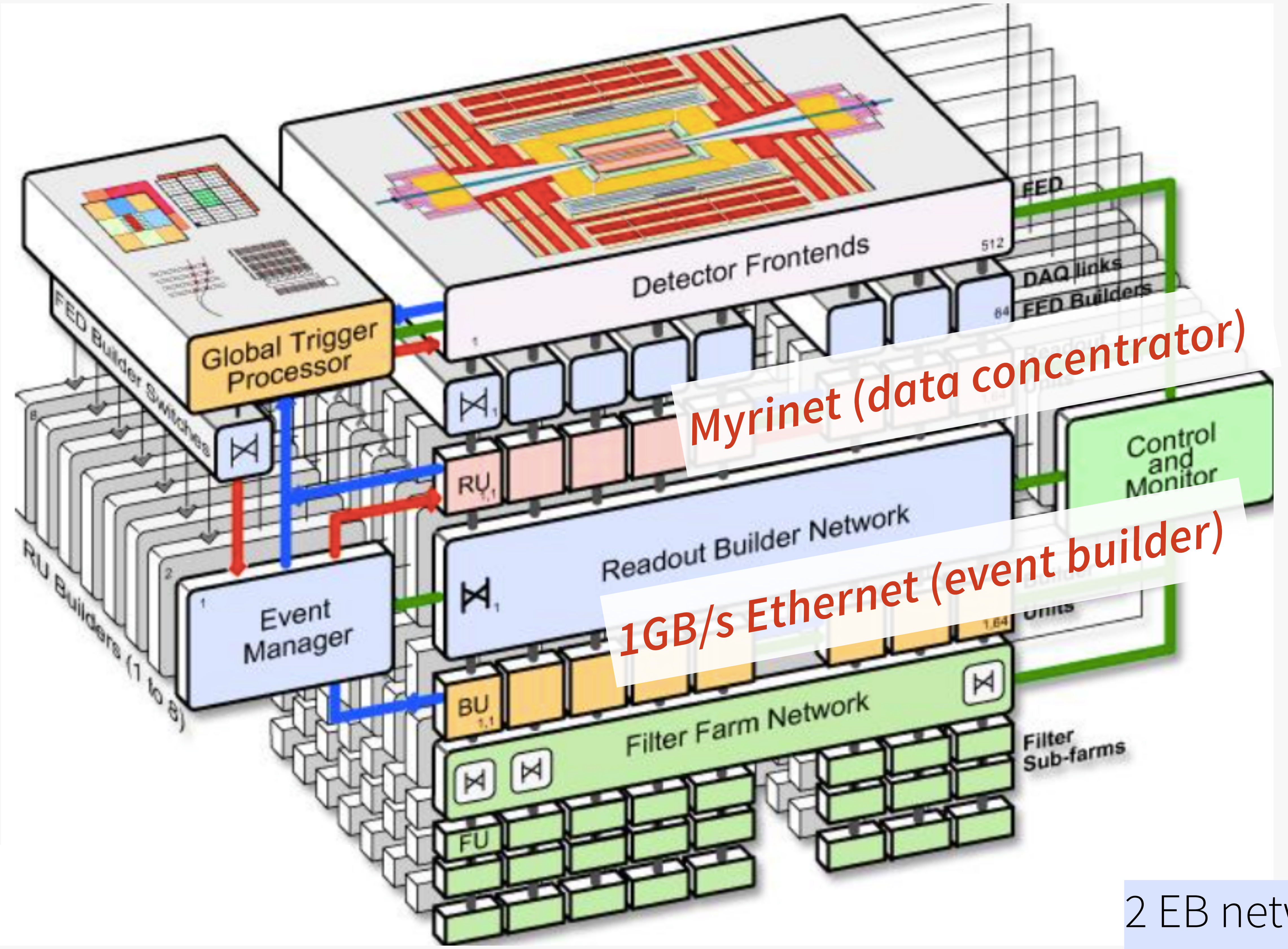
CMS: 2-stage Event building

CMS DAQ-1

Run-1 (as from TDR, 2002)

- Myrinet + 1GBEthernet
- 1-stage building: 1200 cores (2C)
- HLT: ~13,000 cores
- 18 TB memory @100kHz:
  ~90ms/event

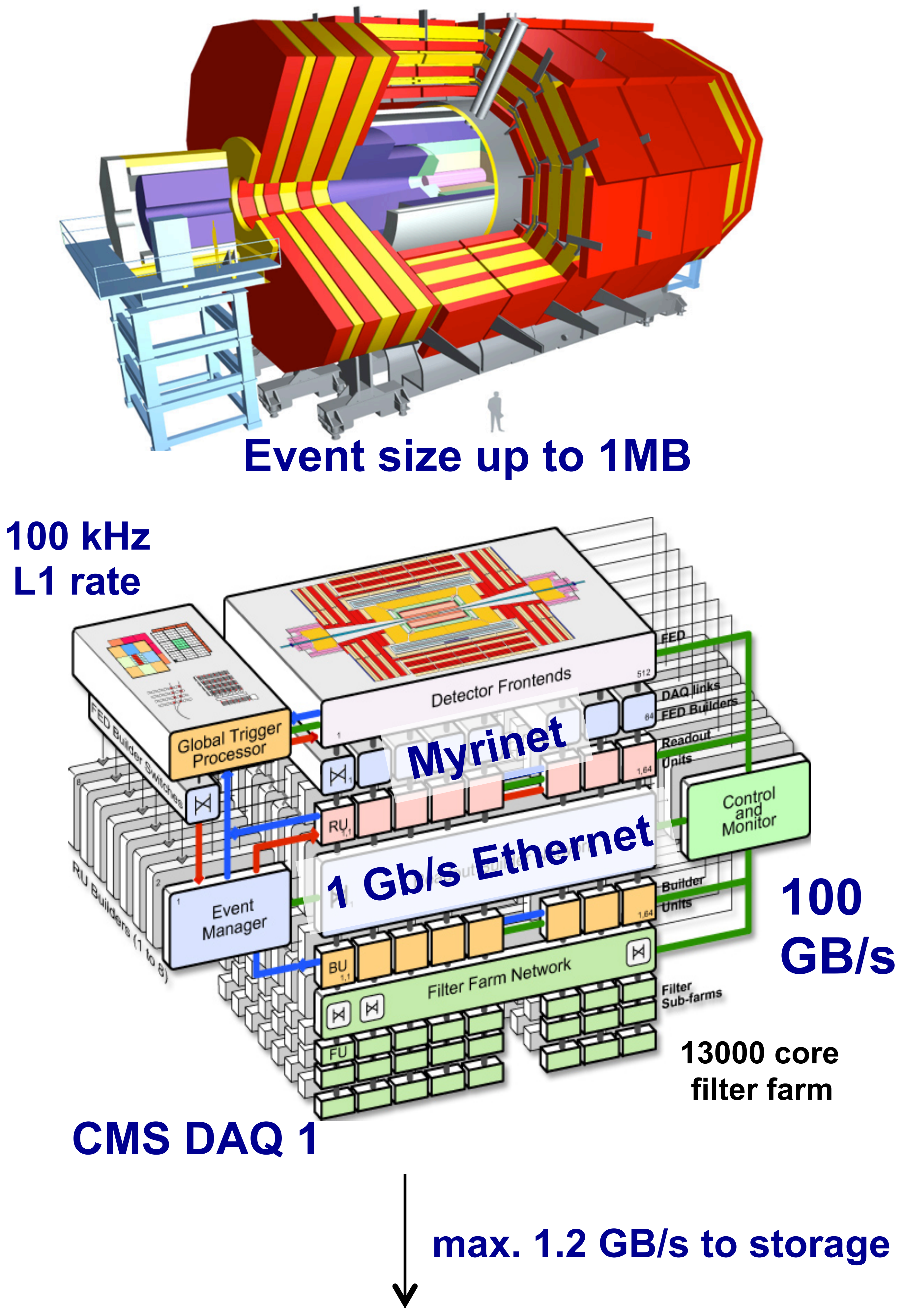


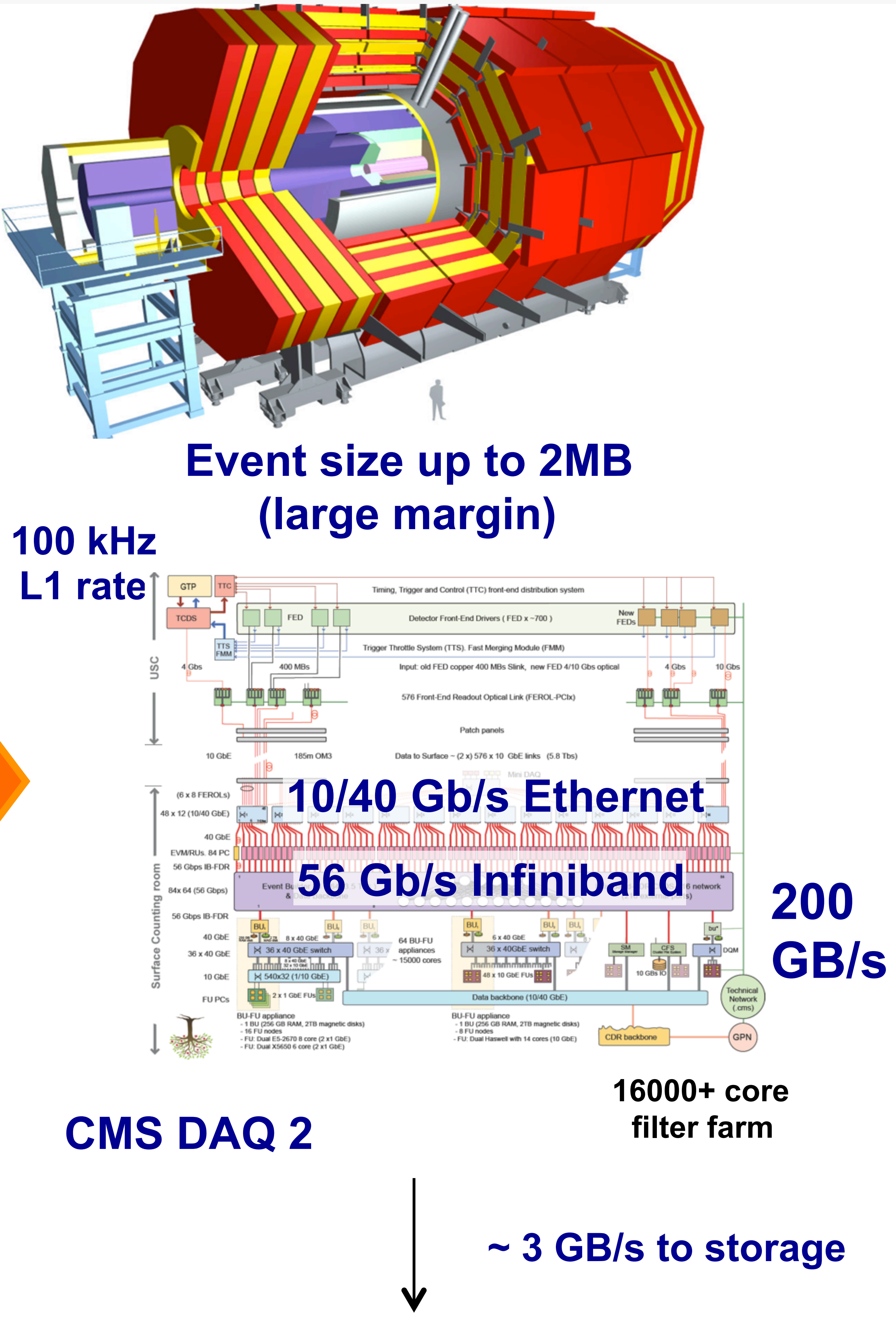


2 EB networks

Filter network

CMS DAQ-2


Event size up to 1MB
100 kHz
L1 rate
Global Trigger Processor
Detector Frontends
Myrinet
1 Gb/s Ethernet
Event Manager
Control and Monitor
Filter Farm Network
100
GB/s
13000 core
filter farm
CMS DAQ 1
max. 1.2 GB/s to storage


Event size up to 2MB
(large margin)
100 kHz
L1 rate
10/40 Gb/s Ethernet
56 Gb/s Infiniband
200
GB/s
16000+ core
filter farm
CMS DAQ 2
~ 3 GB/s to storage

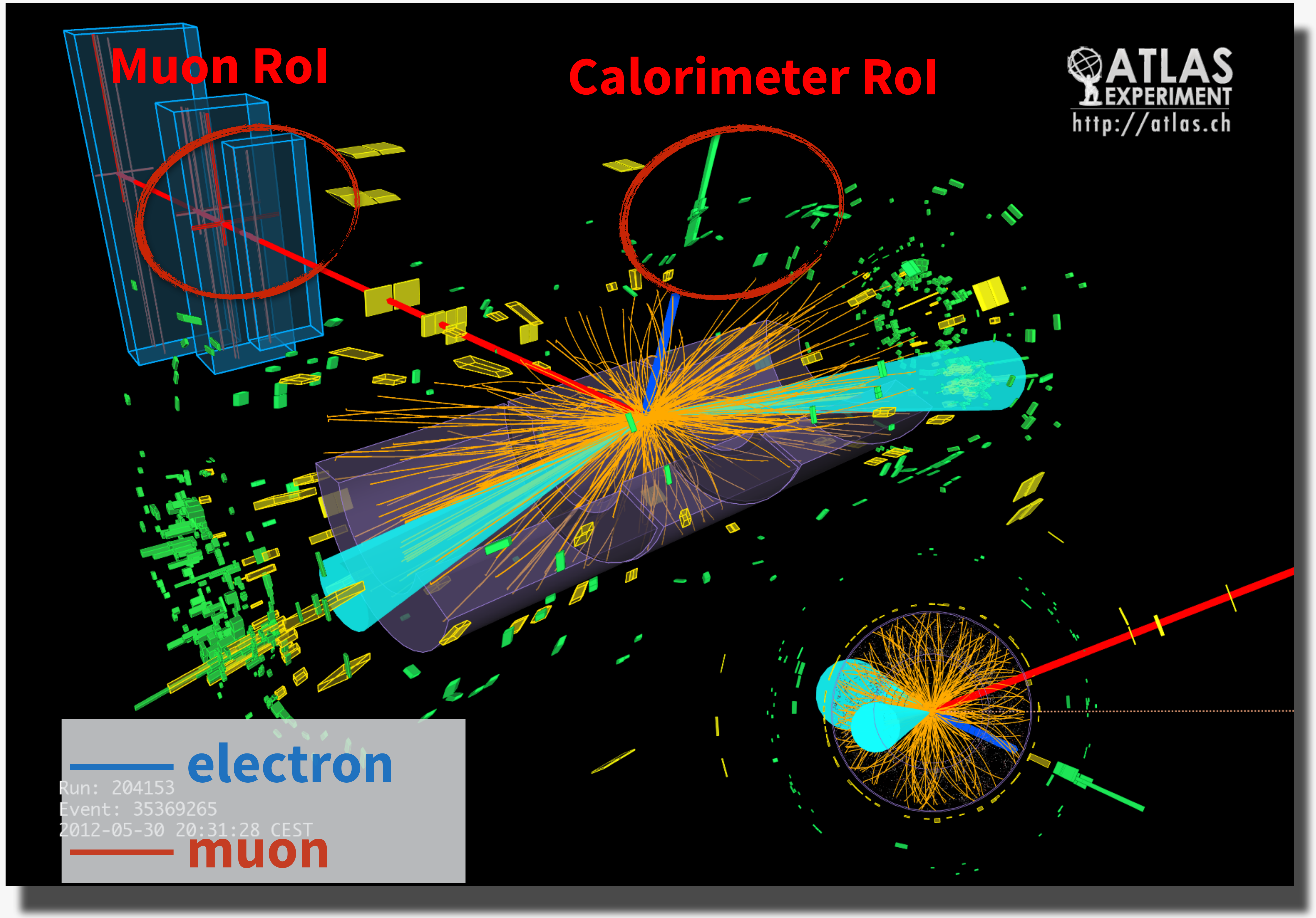
ATLAS: Region of Interest (ROI) dataflow

HLT selections based on regional readout and reconstruction

- seeded by L1 trigger objects (RoI)

Total amount of RoI data is minimal: a few % of the Level-1 throughput

- one order of magnitude smaller readout network
- at the cost of a higher control traffic and reduced scalability



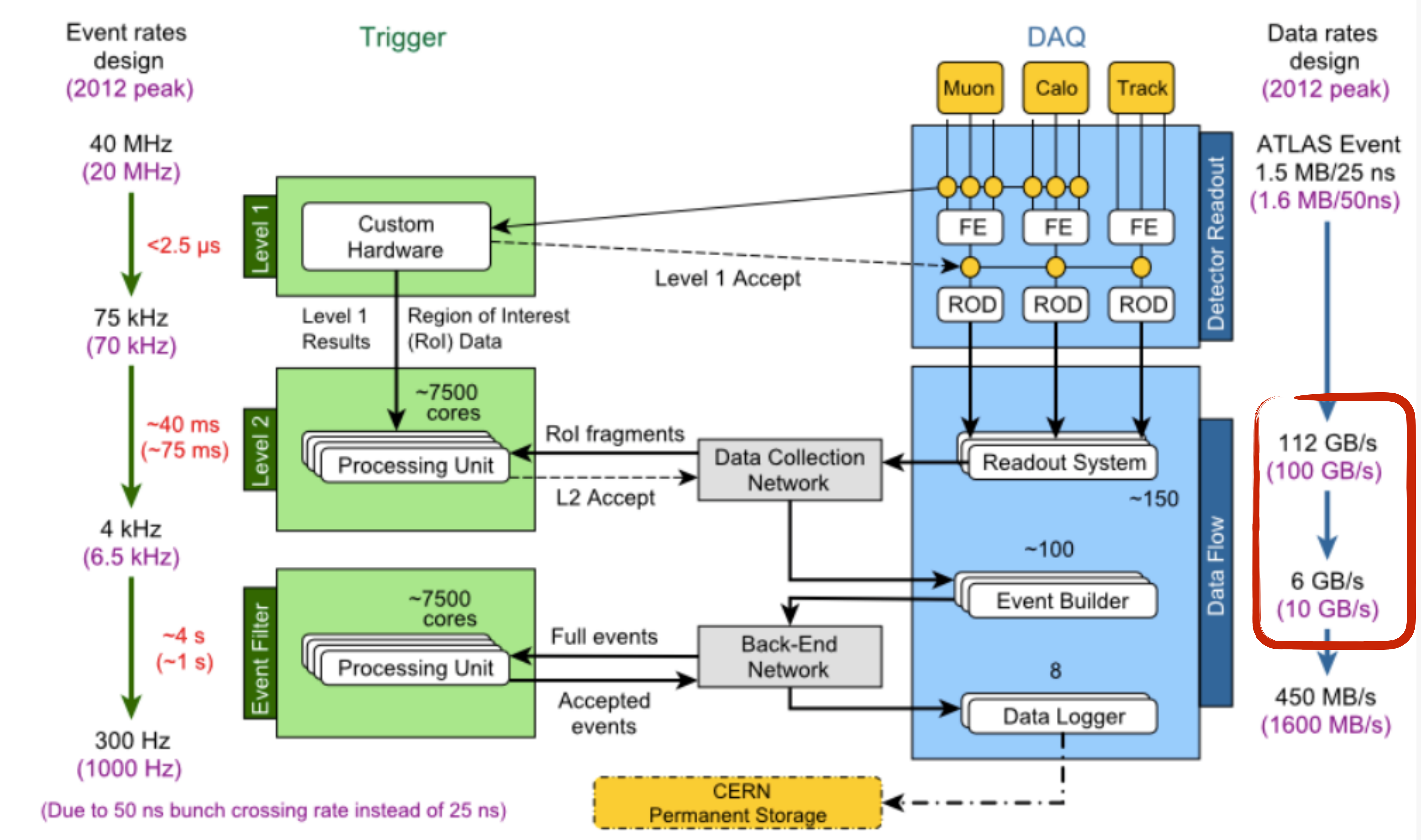
ATLAS: SEEDED reconstruction HLT

RUN 1

Overall network bandwidth: ~10 GB/s

- x10 reduced by regional readout)

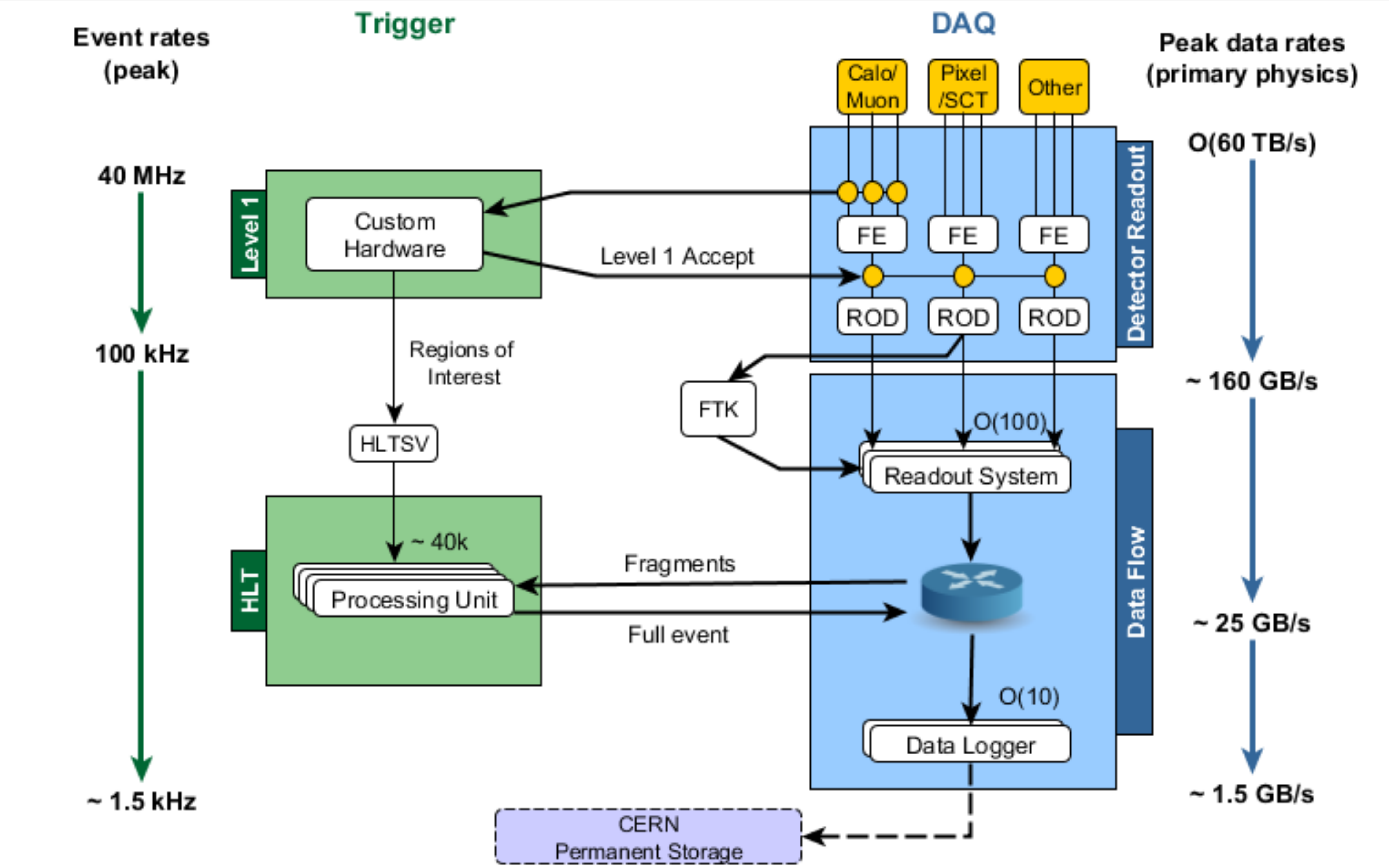
Complex data routing



NEW TDAQ architecture for Run-2

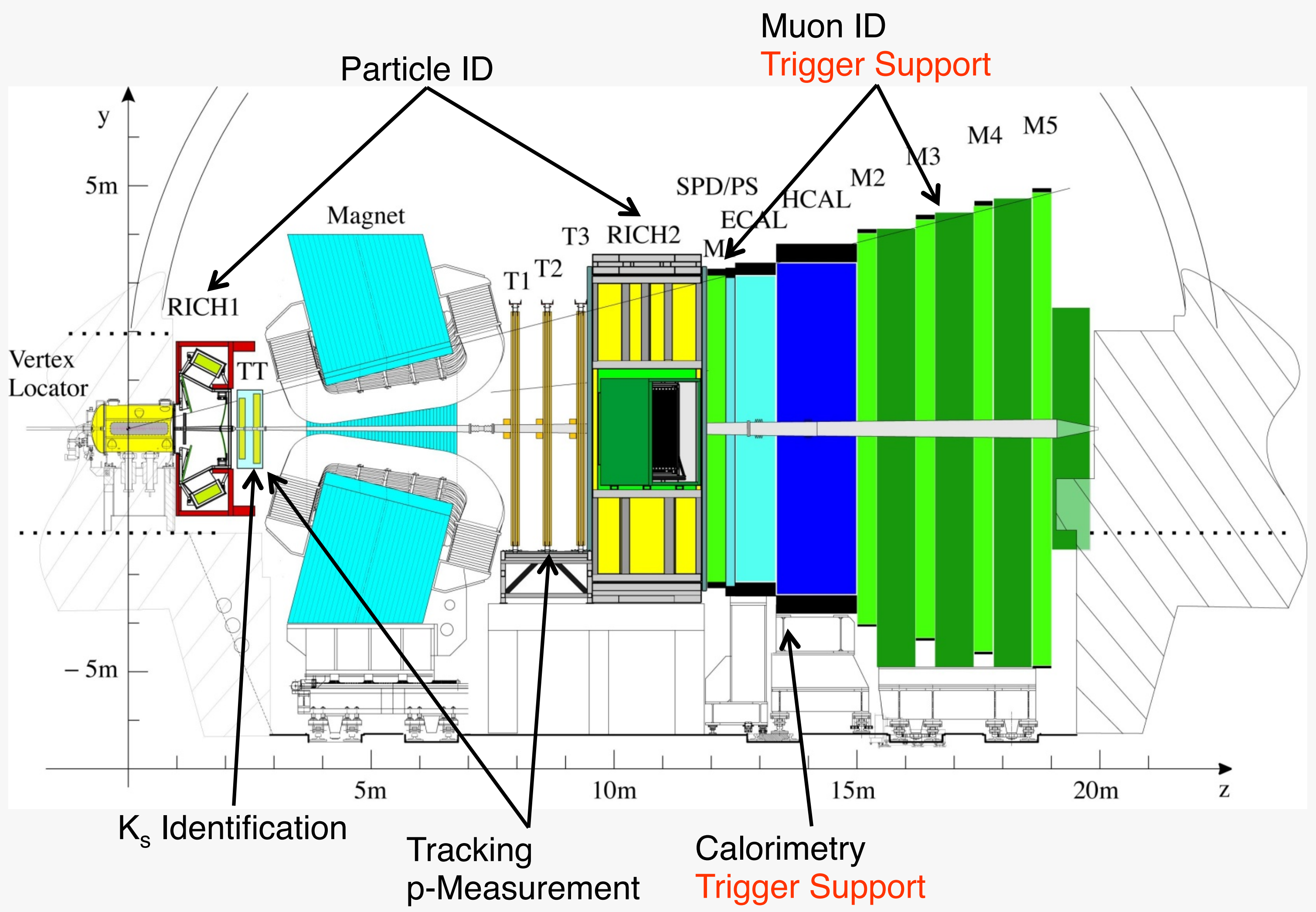
RUN 2

- Increased rates
- Merged L2/HLT
- Increase Readout bandwidth
- Increase HLT rate
- Unified network



LHCb TDAQ Architecture

RUN 2

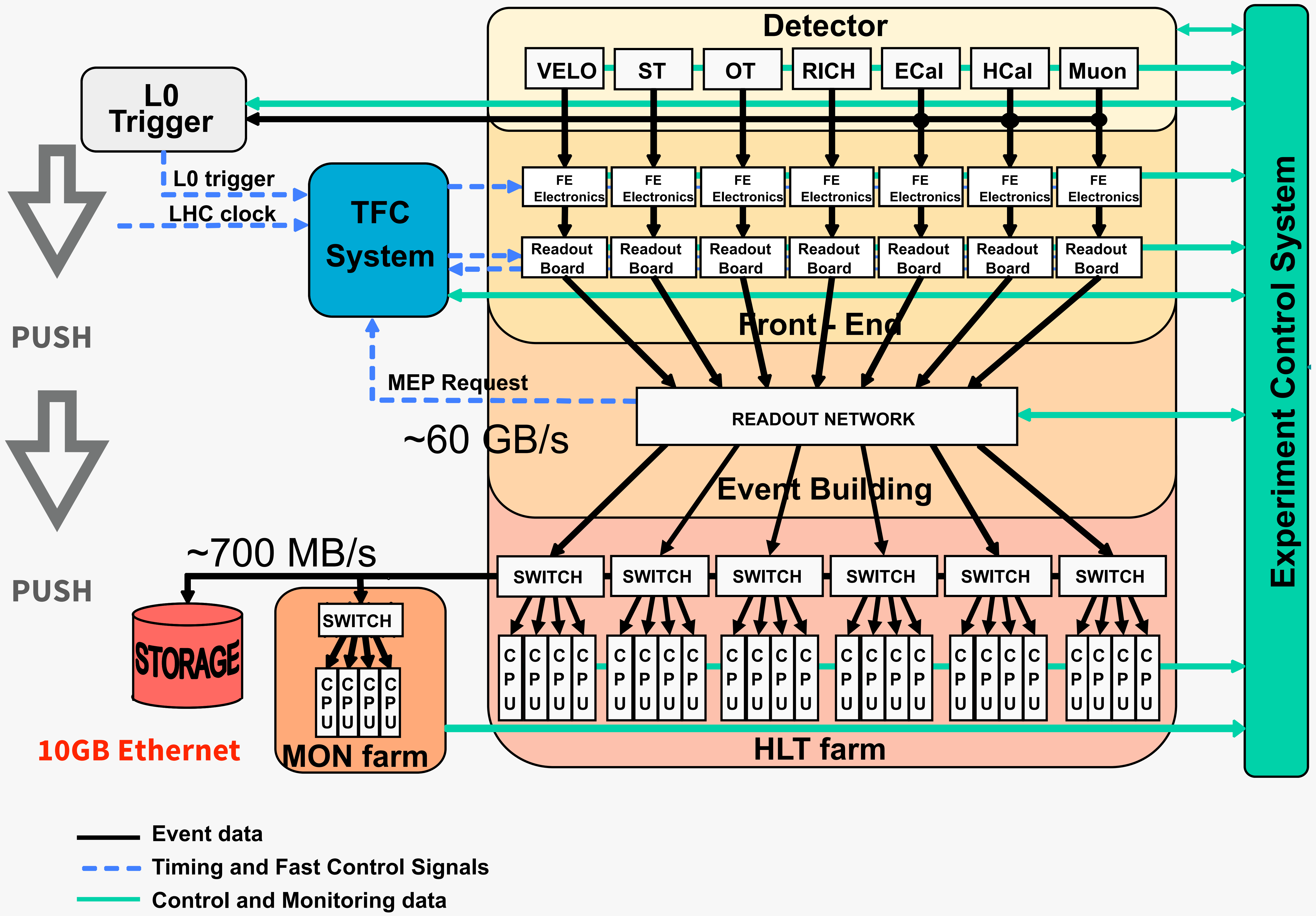


Single forward arm spectrometer → reduced event size

- Average event size 60 kB
- Average rate into farm 1 MHz
- Average rate to tape ~12 kHz

Small event, at high rate

- optimised transmission



Phase 1 Upgrade

No Level-1 Trigger!

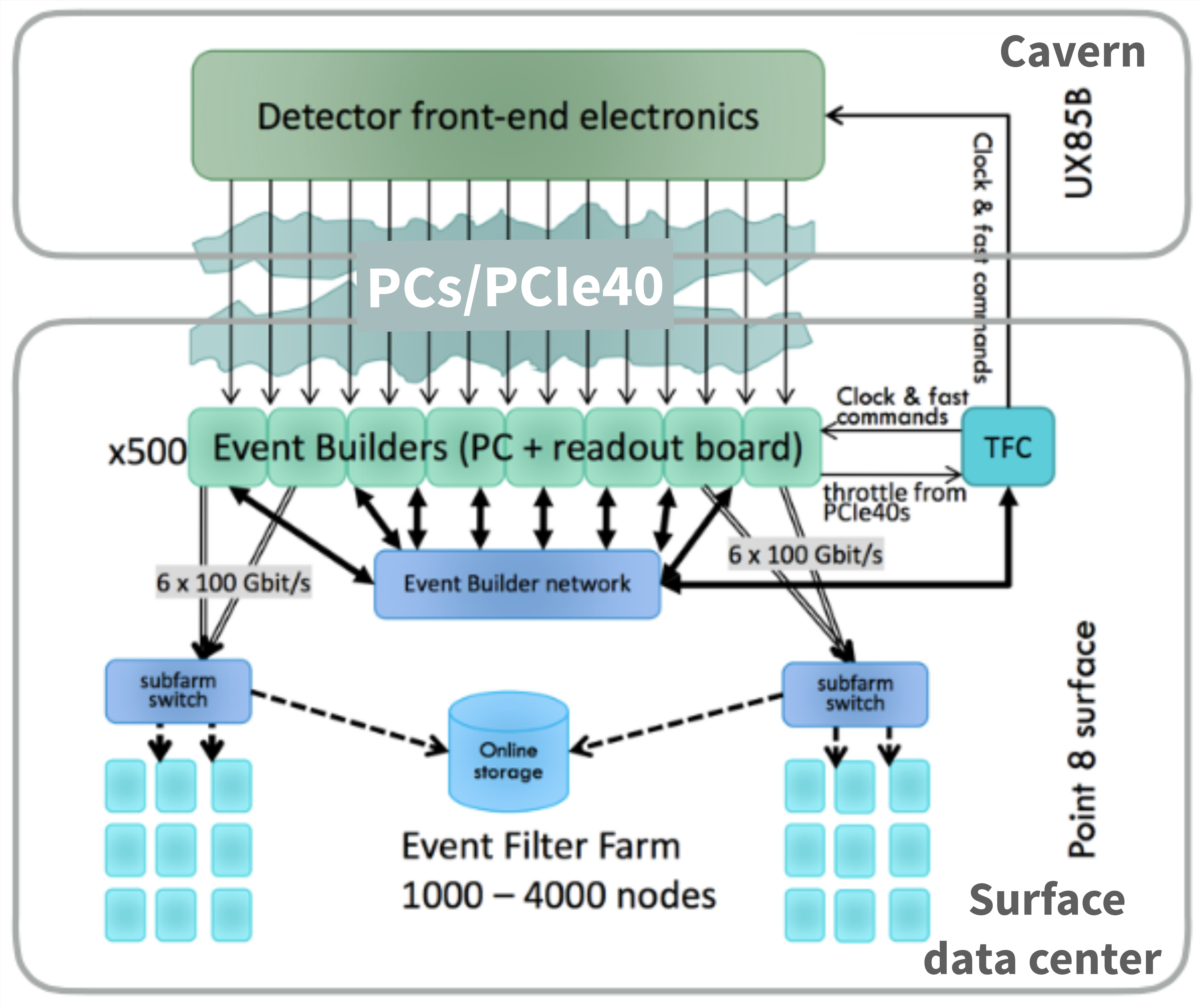
Data reduction before EB

- custom readout FPGA-card (PCIe40)
- each sub detectors with its packing algorithm, i.e. zero-suppression and clustering

Readout: ~10,000 GBT links (4.8 Gb/s, rad-hard)

DataFlow: Merged EB and HLT

- reduced network complexity
- scalable up to 400 x 100Gbps links



ALICE

19 different detectors

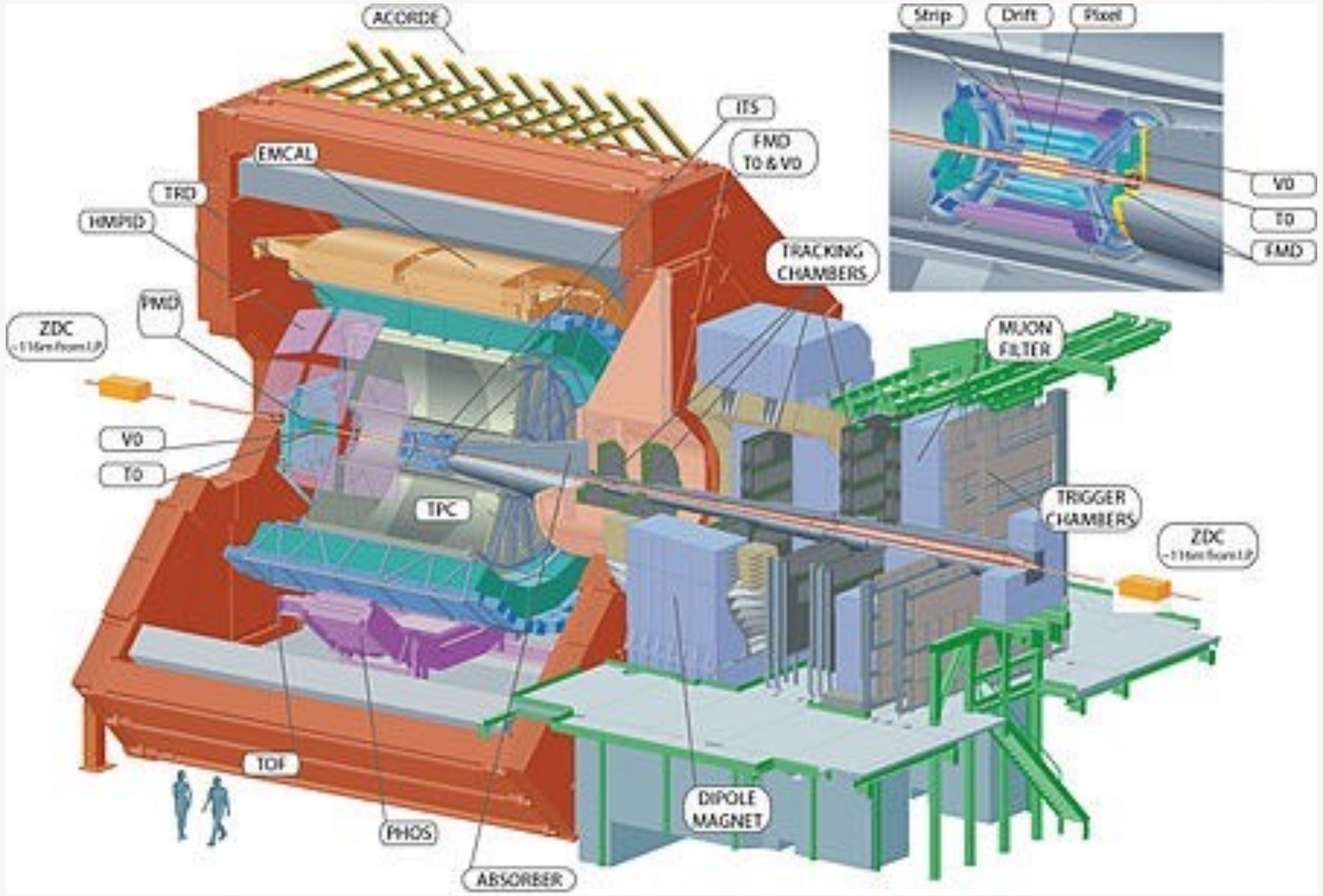
- with high-granularity ant timing information
- Time Projection Chamber (TPC):
  very high occupancy, slow response

Large event size (> 40MB)

- TPC producing 90% of data

Challenges for the TDAQ design:

- detector readout: up to ~50 GB/s
- low readout rate: max 8 kHz
- storage: 1.2 TB/s (Pb-Pb)



ALICE TDAQ architecture

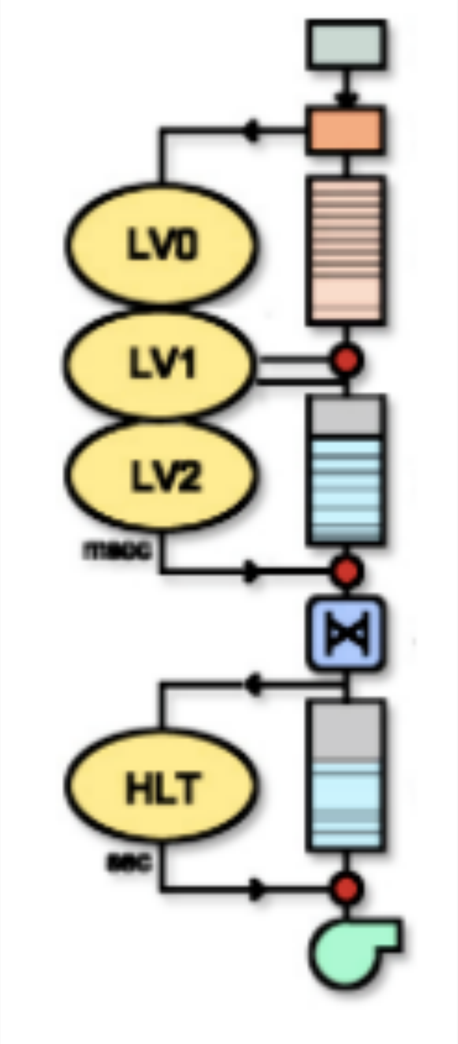
Hardware

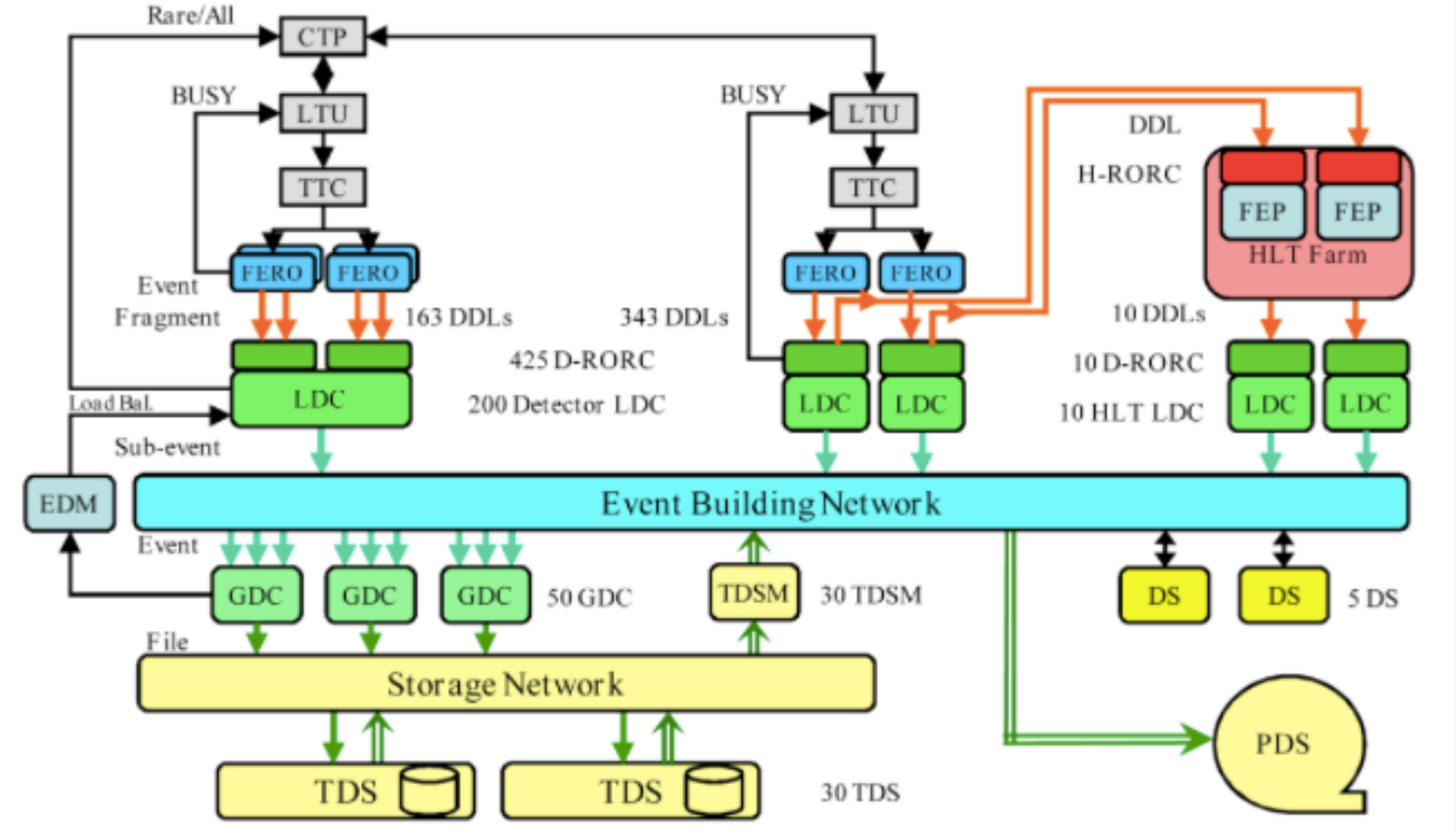
1.2µs

6.5µs

8.8µs

Software





Trigger

- 3 hardware levels
- 1 software

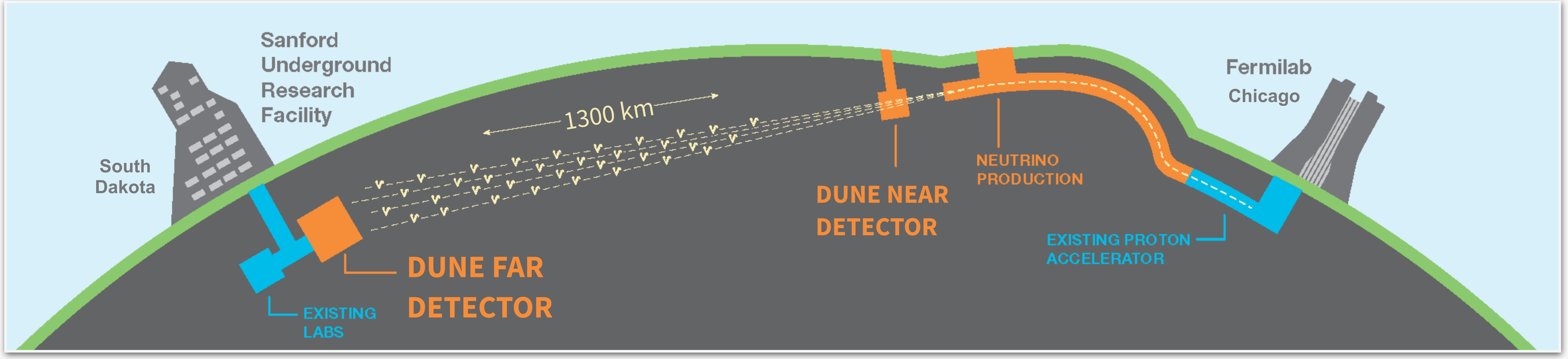
Detector readout (~20 GB/s) with point-to-point optical links

- ~ 400 DDL to RORC PCI cards (6 Gbps)
- data fragments directly into PC memory of LDCs, at 200 MB/s (via DMA)

Dataflow with local LDC and global GDC data concentrators (for Event Building)

- HLT as any other sub-detector in DAQ

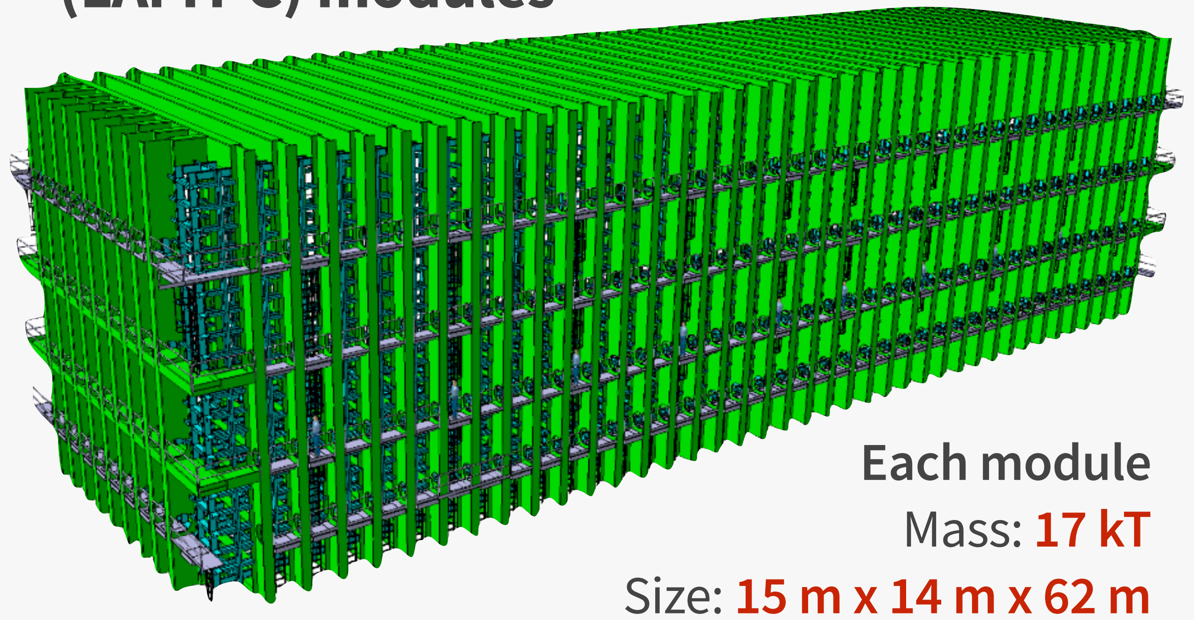
The Deep Underground Neutrino Experiment - DUNE



Leading edge, world class neutrino experiment with high profile physics programme

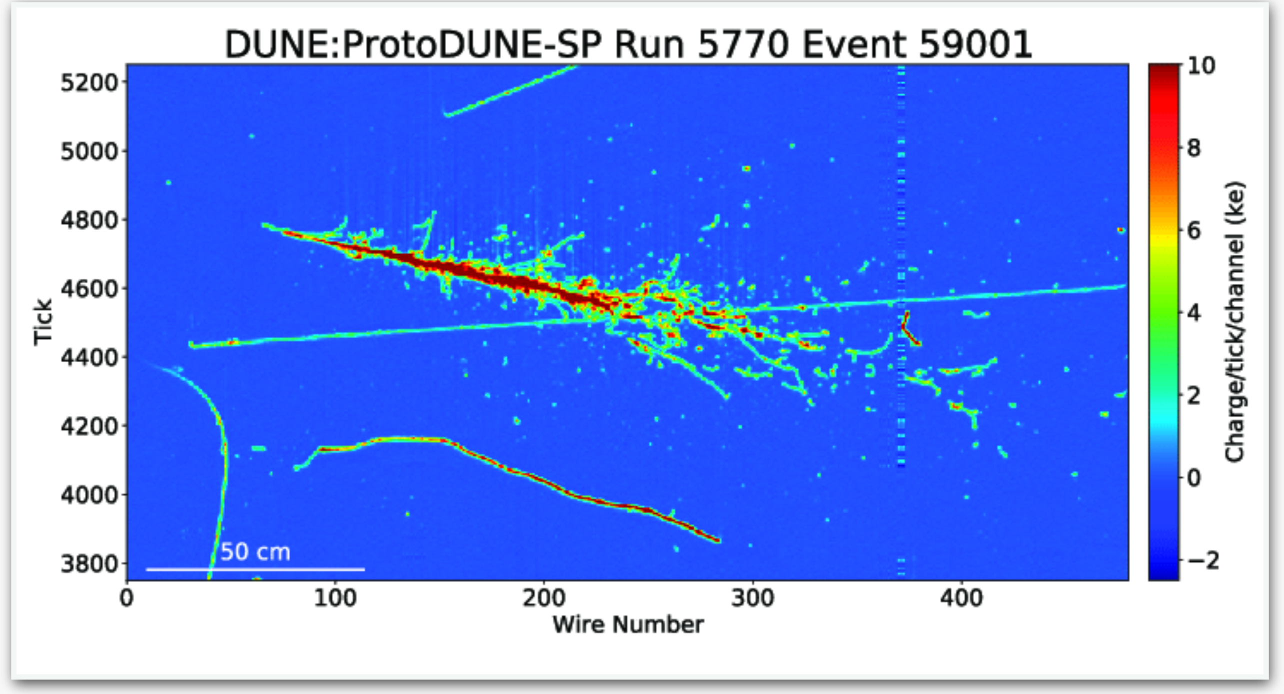
- Nature of neutrinos, supernova collapse, proton decay searches

4 Liquid Argon Time Projection Chamber (LArTPC) modules



Each module
Mass: 17 kT
Size: 15 m x 14 m x 62 m
Operational temperature: -186 °C

High-resolution imaging detector

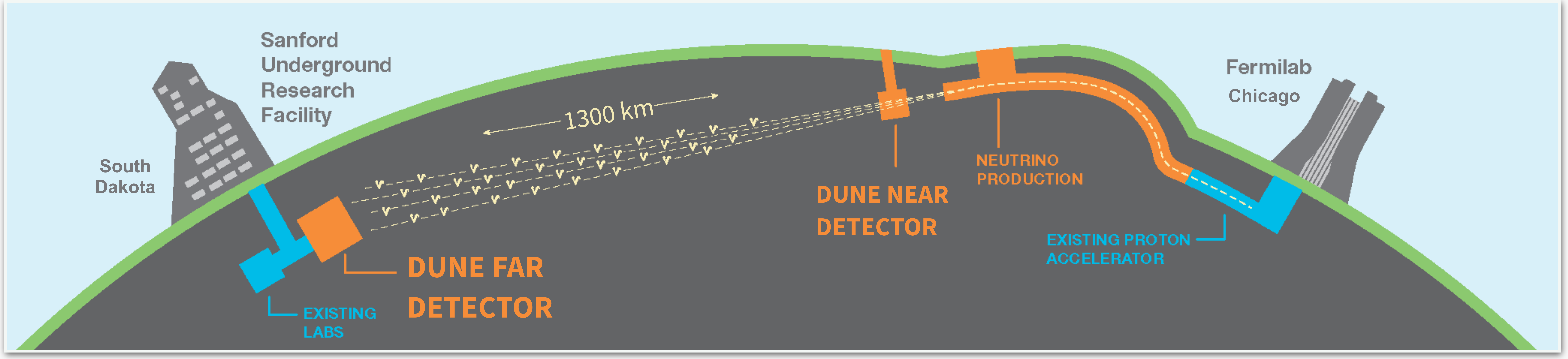


spatial resolution: 5mm

Gigantic Far Detector

- Huge target mass AND high resolution imaging

The Deep Underground Neutrino Experiment - DUNE



Leading edge, world class neutrino experiment with high profile physics programme

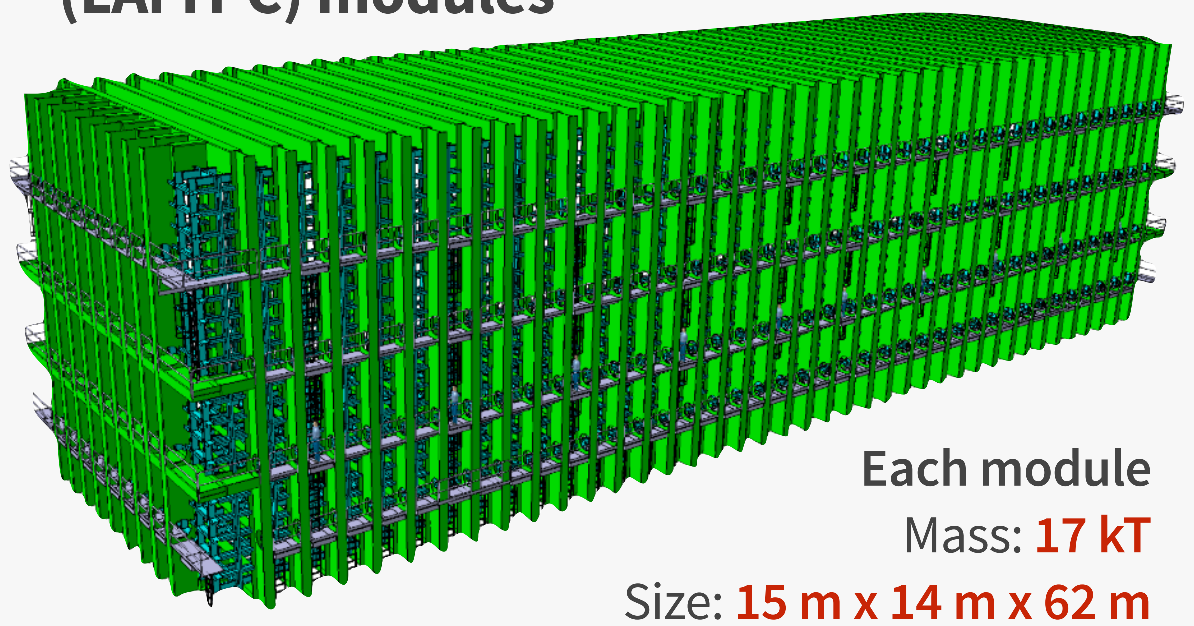
- Nature of neutrinos, supernova collapse, proton decay searches

TDAQ: no quick access and no large host lab in the vicinity!

Gigantic Far Detector

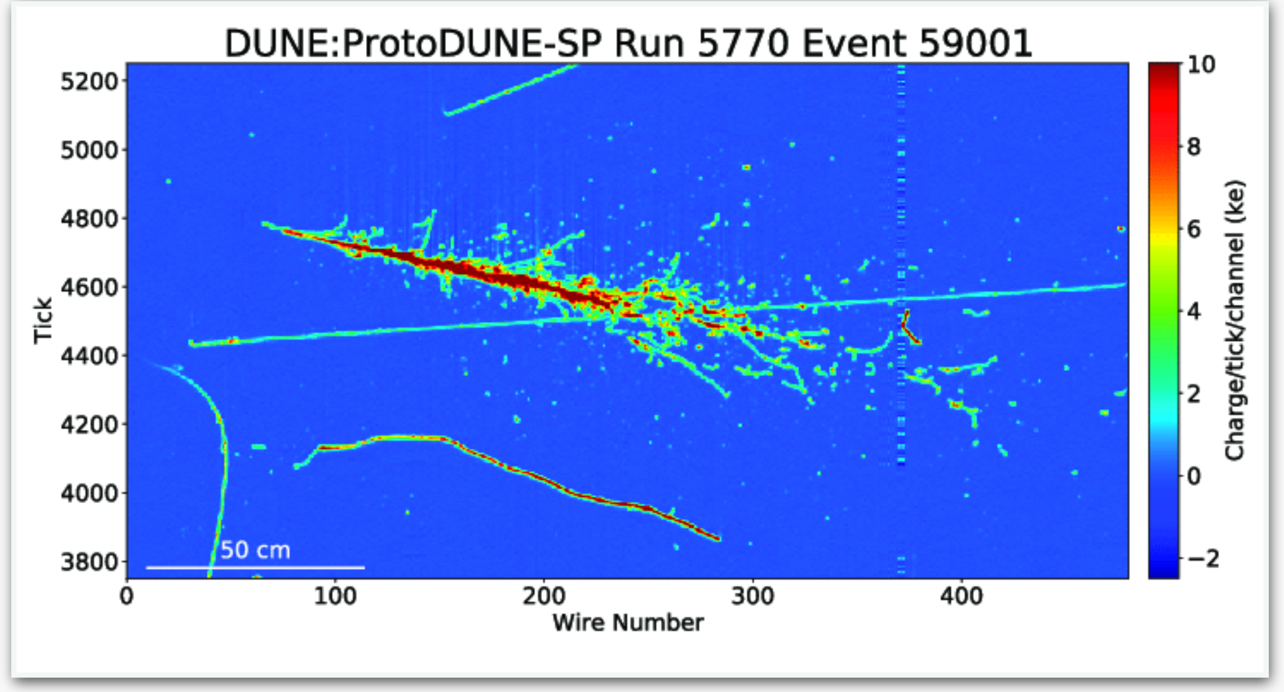
- Huge target mass **AND** high resolution imaging

4 Liquid Argon Time Projection Chamber (LArTPC) modules



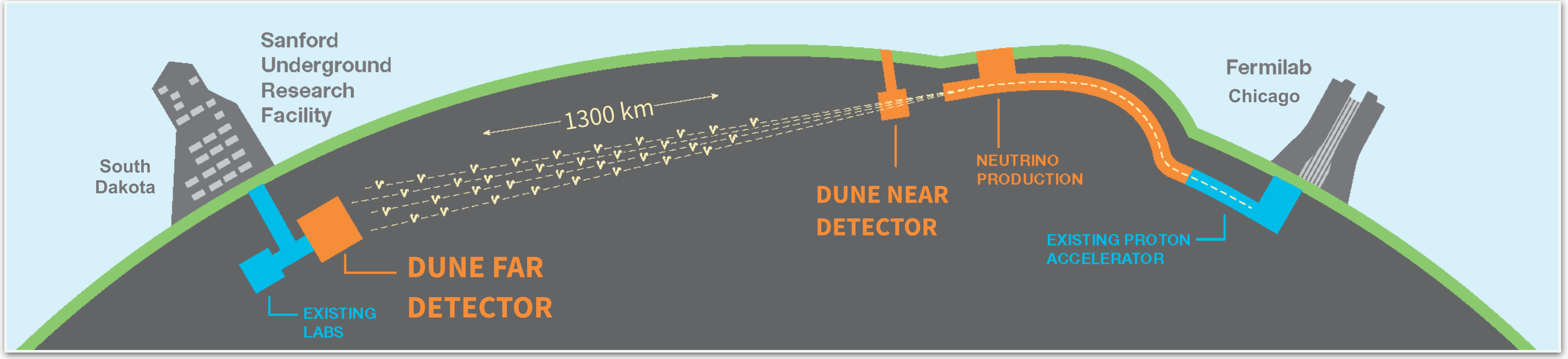
Each module
Mass: **17 kT**
Size: **15 m x 14 m x 62 m**
Operational temperature: **-186 °C**

High-resolution imaging detector



spatial resolution: **5mm**

The Deep Underground Neutrino Experiment - DUNE



Leading edge, world class neutrino experiment with high profile physics programme

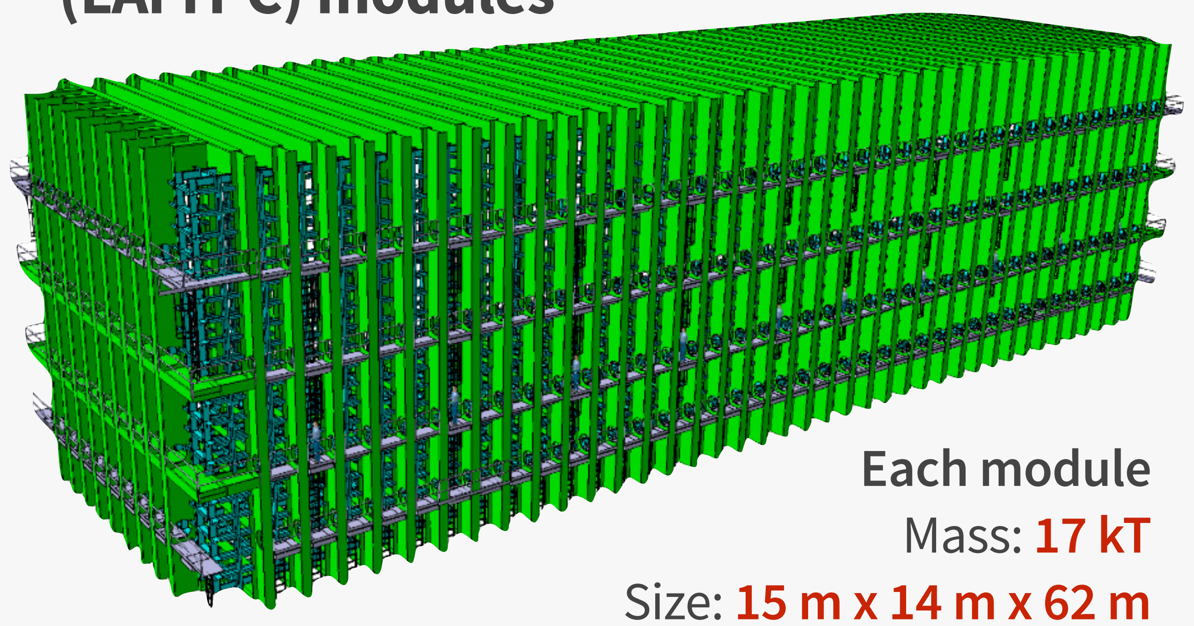
- Nature of neutrinos, supernova collapse, proton decay searches

TDAQ: no quick access and no large host lab in the vicinity!

Gigantic Far Detector

- Huge target mass **AND** high resolution imaging

4 Liquid Argon Time Projection Chamber (LArTPC) modules



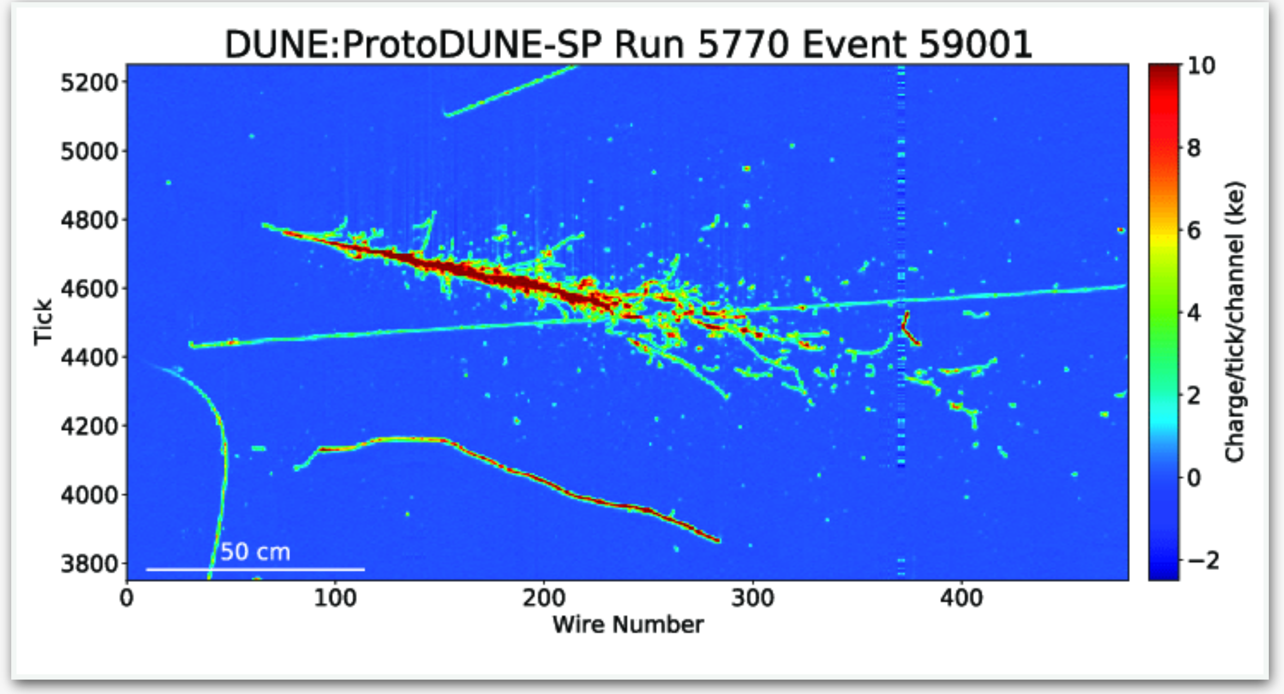
Each module

Mass: 17 kT

Size: 15 m x 14 m x 62 m

Operational temperature: -186 °C

High-resolution imaging detector



spatial resolution: **5mm**

TDAQ: 4 independent instances, synchronized to a common clock, supporting different detector technologies

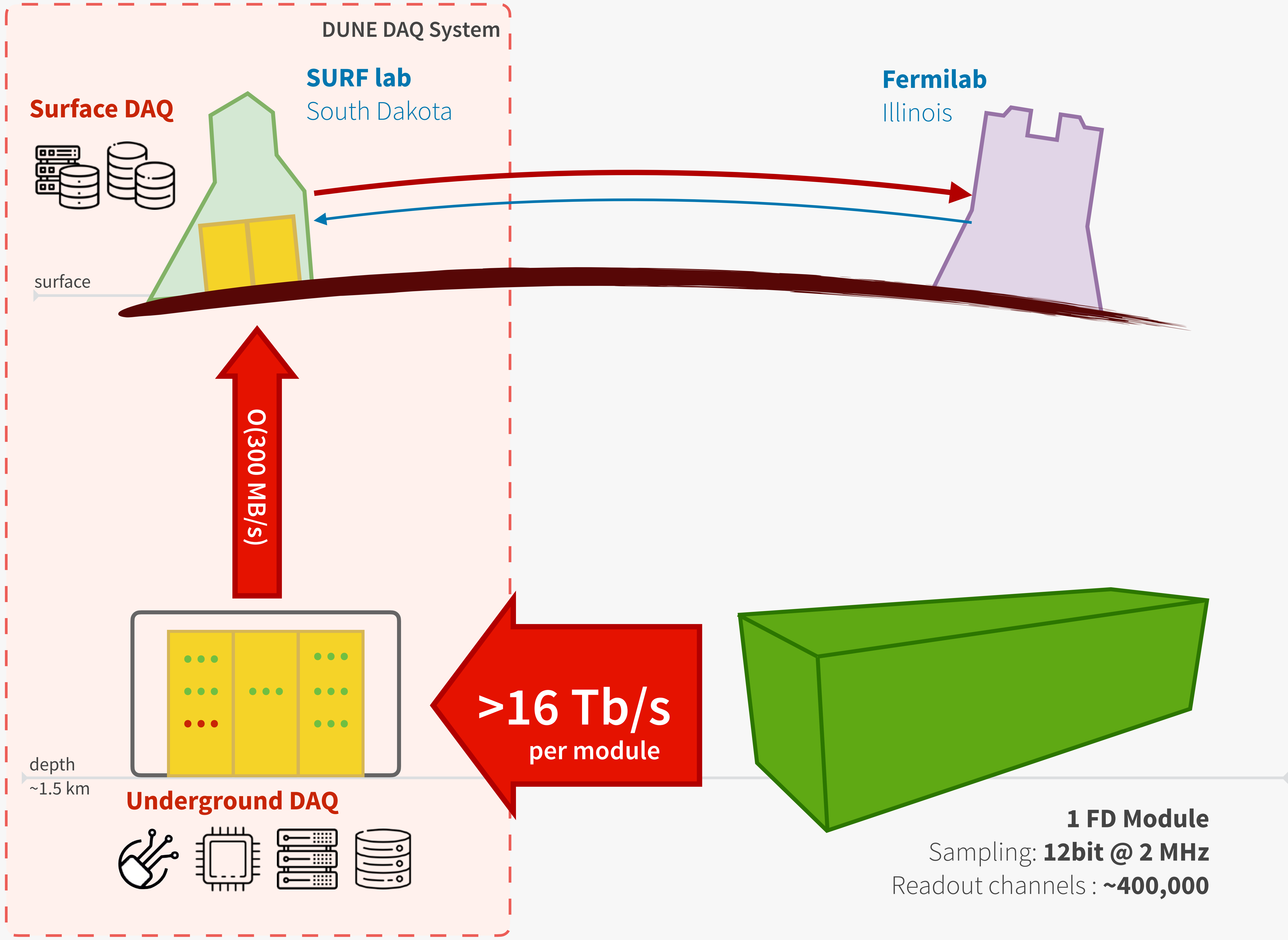
The DUNE Data AcQuisition System - DAQ

DUNE DAQ System

- Collects large amount of **streaming** data from detectors
- Selects only interesting interactions
- Buffers the **full data stream** for ~100s for supernova physics
- Deliver selected interactions to permanent storage

Unique challenges

- **High data rate, high uptime**
- **Remote experimental site**
- **Deep underground in an active mine**



Summary

This lecture is just an introduction about data acquisition

- DAQ (& Trigger) is a complex and fascinating topic, combining very different expertise
- More details on **Trigger** and **FPGAs** in the following lectures

Covered the principles of a simple data acquisition system

- Basic elements: trigger, derandomiser, FIFO, busy logic
- Scaling to multi-channel, multi-layer systems
- How data is transported
  - Bus versus network

A (very) brief overview of LHC experiments + DUNE DAQ systems

- Similar architectures, different optimisations driven by detector requirements