

(Introduction to) Data Acquisition

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

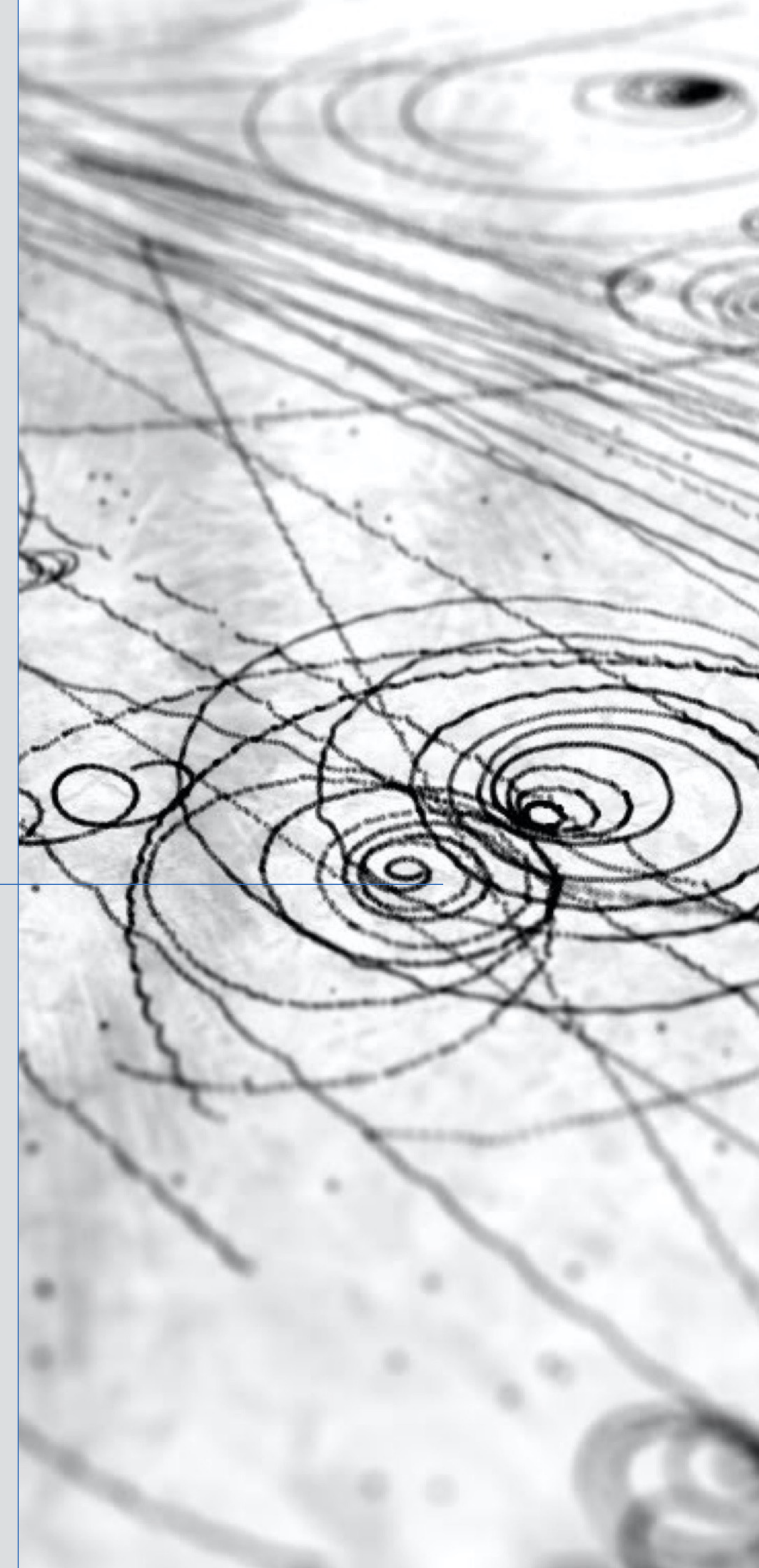
May 14, 2020

Alessandro Thea

Rutherford Appleton Laboratory - PPD



**Science and
Technology
Facilities Council**



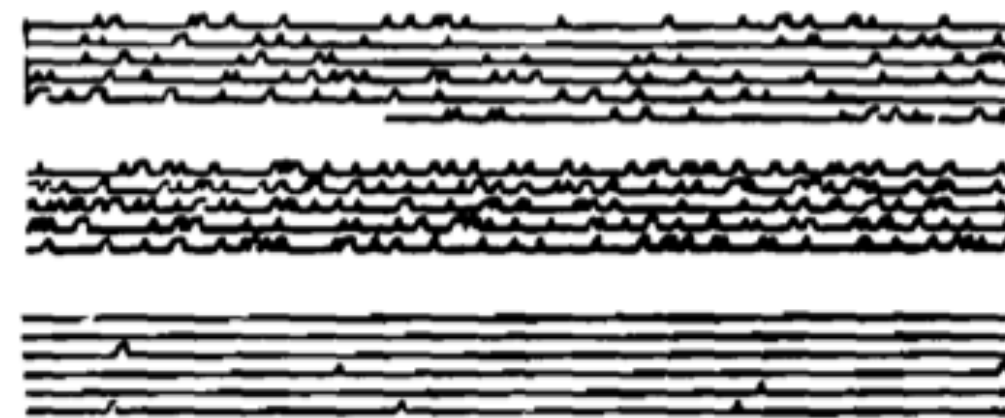
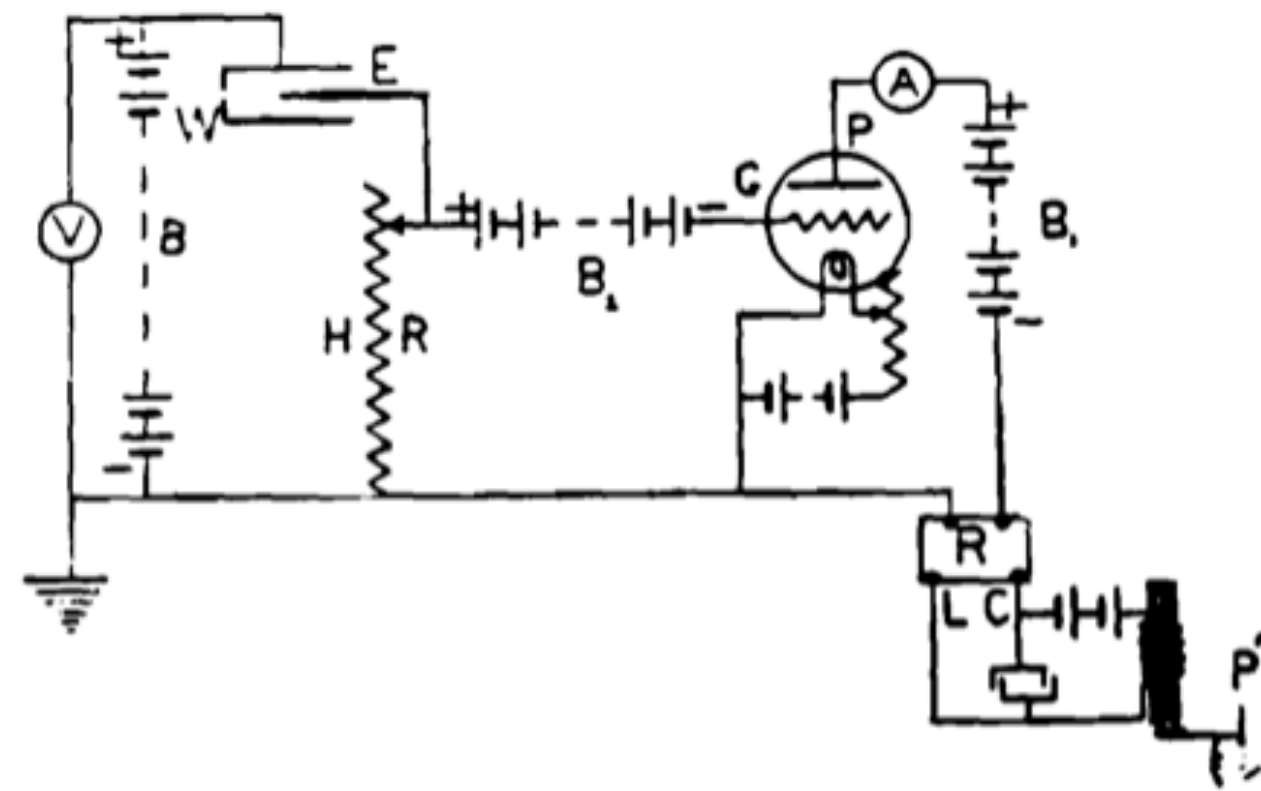
Acknowledgements

Lecture inherited from Monica Wielers

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

Centenary (+1)

ON THE AUTOMATIC REGISTRATION OF α -PARTICLES, β -PARTICLES AND γ -RAY AND X-RAY PULSES



Alois F. Kovarik
Sheffield Scientific School
Yale University
New Haven, Conn.
January 25, 1919

Physics TDAQ is 101 years old!

- Phys. Rev. 13, 272 , 1st April 1919

“... visual or audible methods of counting are quite trying on the nerves ... A self-recording device would therefore be an obvious improvement.”

A lot has happened since then

- But trigger and DAQ can still be quite trying on the nerves

Thanks to E. Meschi and D.Newbold for spotting this!

Outline

1. Introduction

1.1. What is DAQ?

1.2. Overall framework

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

Data Acquisition (**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 PC

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

- (aka the dark arts)
- Boundaries not well defined

An **alchemy** of

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

Money and manpower matter as well



Something interesting

Main role of DAQ

- process the signals generated in a detector
- and saving the **interesting** information on permanent storage

What does it mean interesting?

- When does this happen?

We need a trigger!

Trigger Lecture

- Dr. J.Kirk



Trigger!

Either 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**

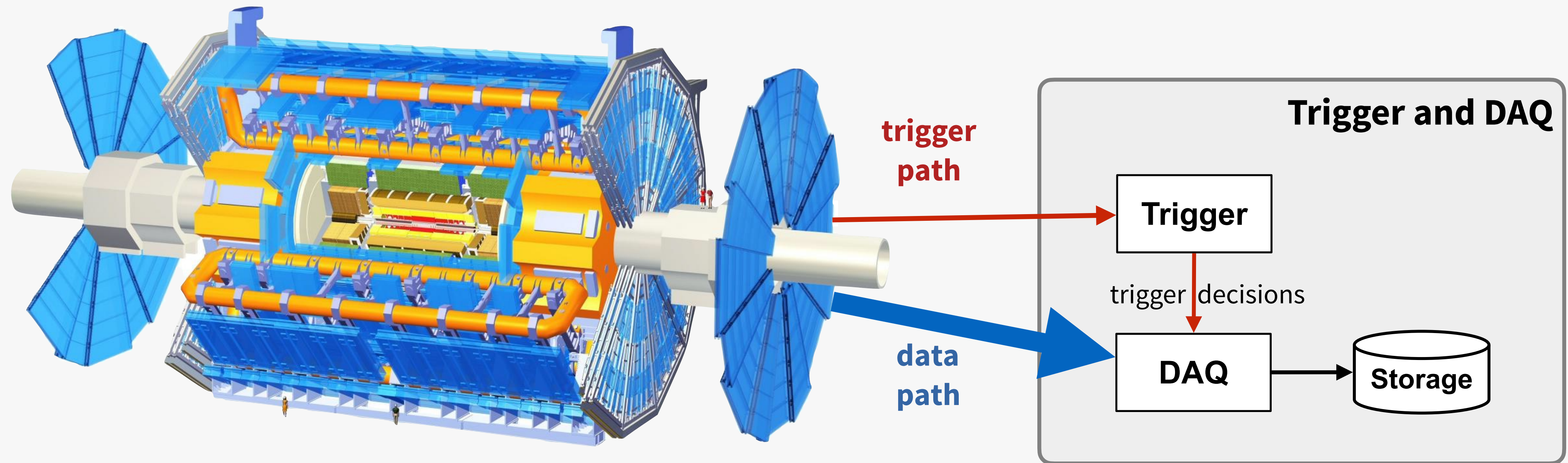
- time it takes to form and distribute its decision

The trigger system generates a prompt signal used to start the data-acquisition processes

- To be distributed to front end electronics



Twin paths



Trigger path

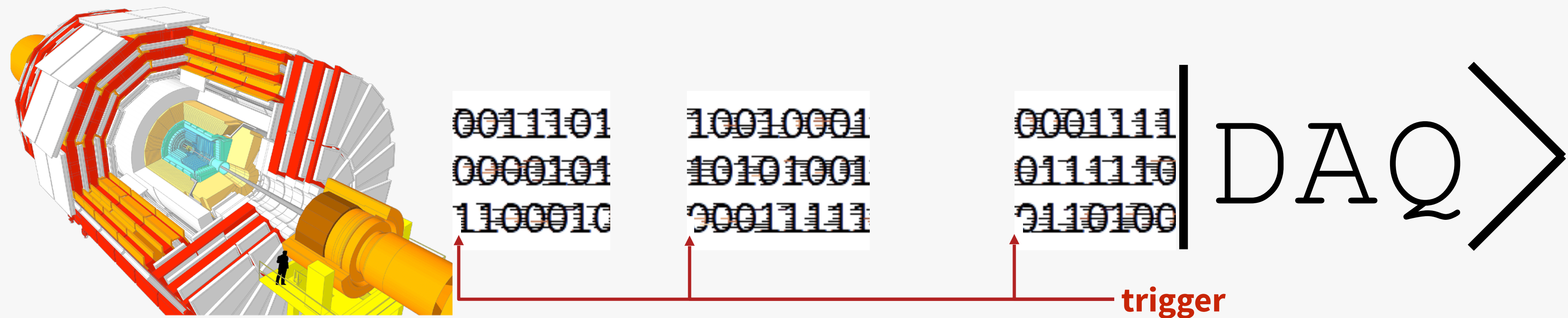
- From dedicated detectors to trigger logic

Data path

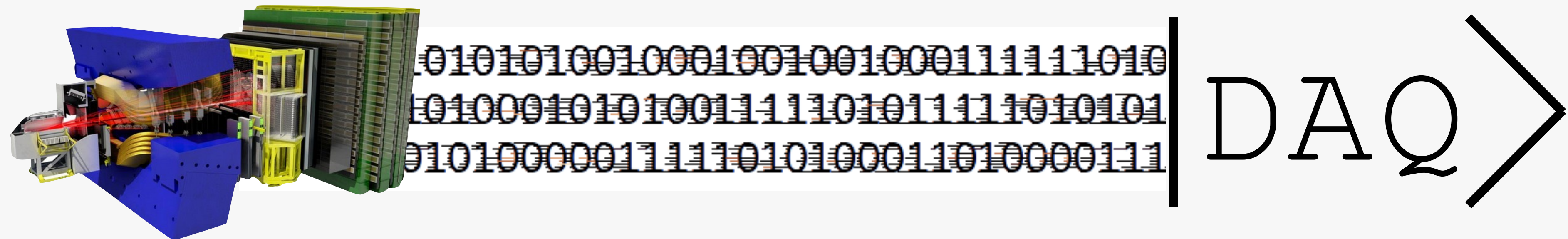
- From all the detectors to storage
- On positive trigger decision

Trigger(less)

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



Triggerless: the detector push data at its speed and the downstream DAQ must keep the pace



DAQ duties

Gather data produced by detectors

- **Readout**

Form complete events

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

Possibly feed other trigger levels

- **High Level Trigger**

Store event data

- **Data Logging**

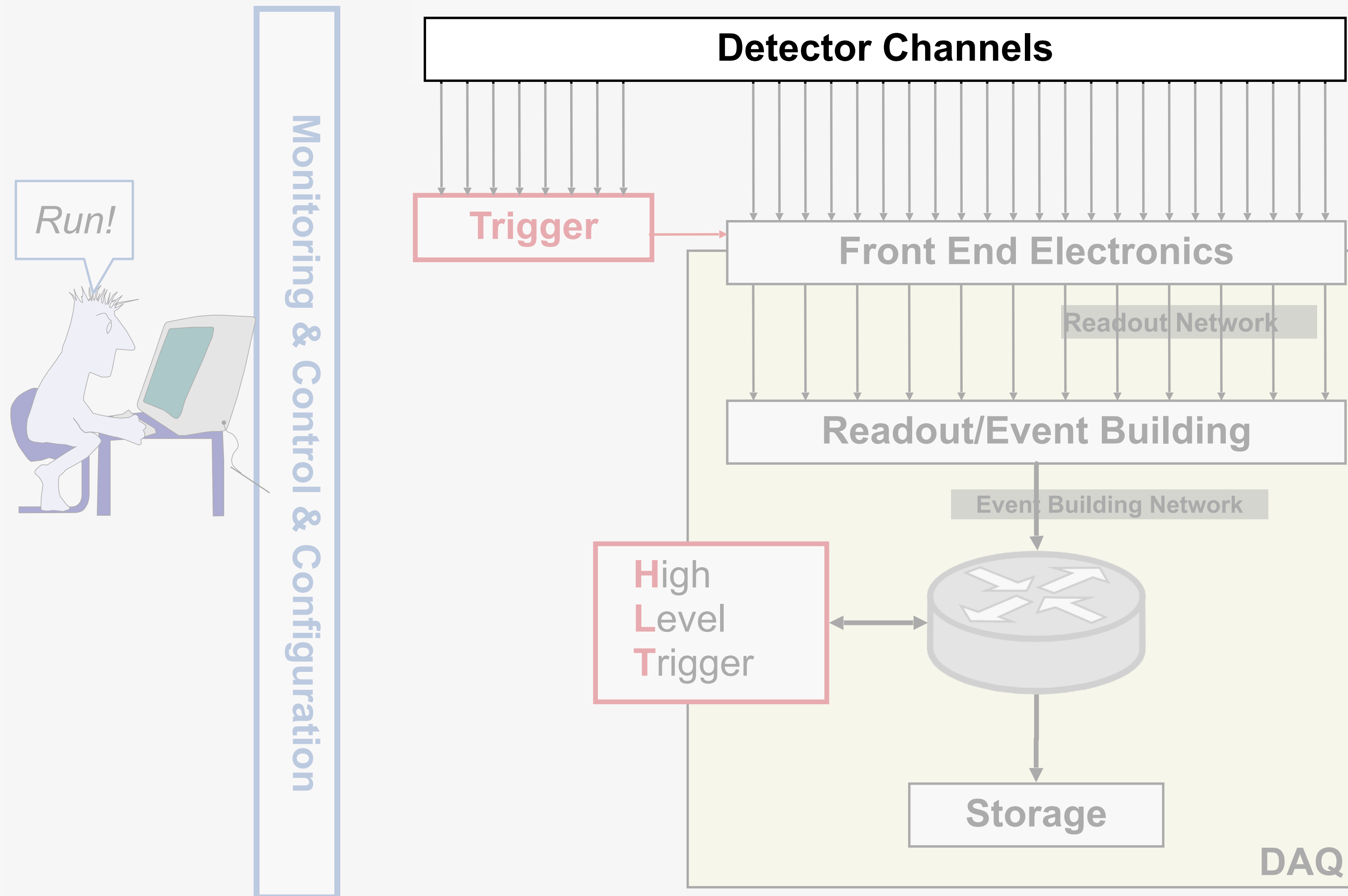
Manage the operations

- **Run Control, Configuration, Monitoring**

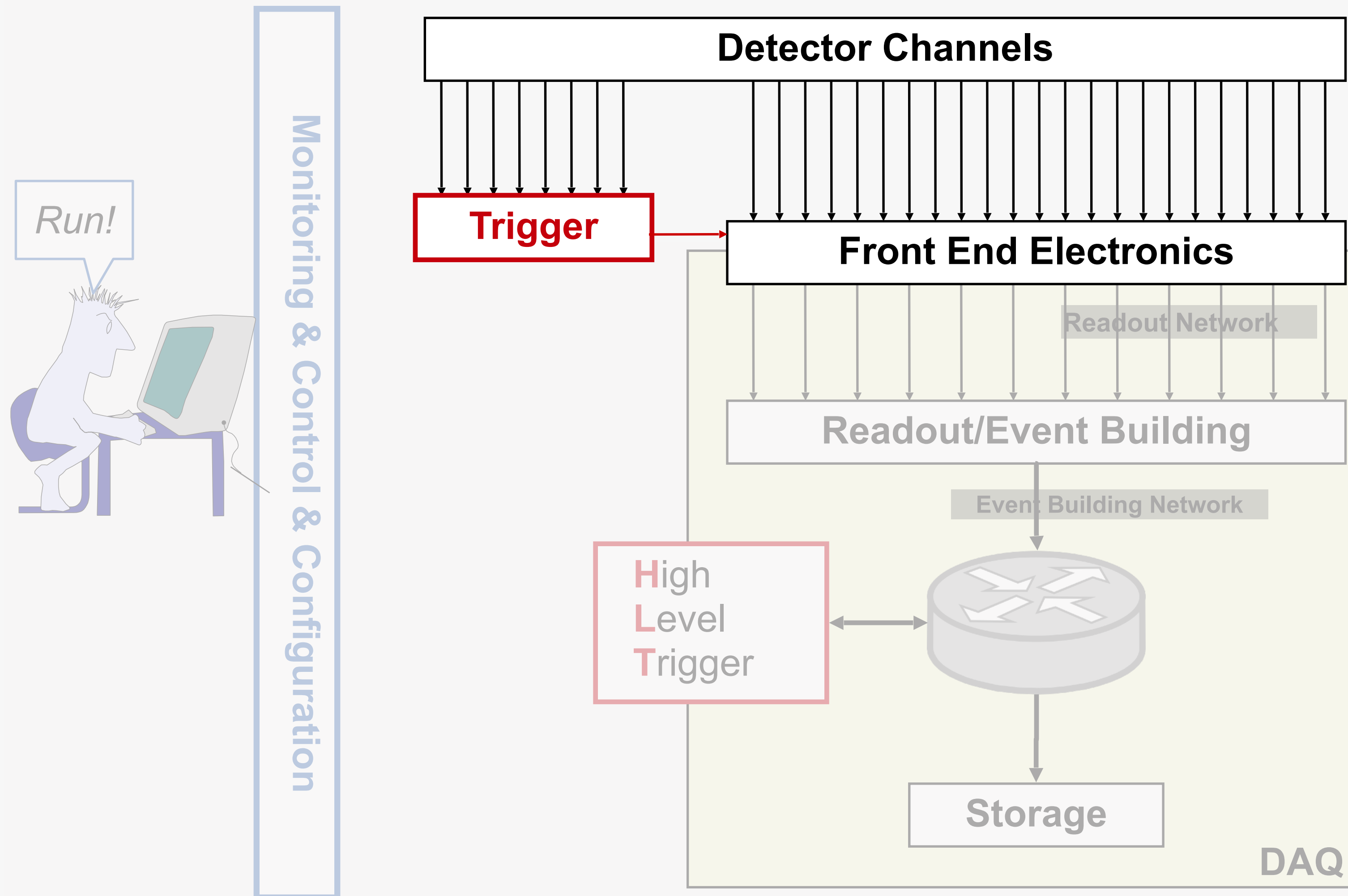


Data Flow

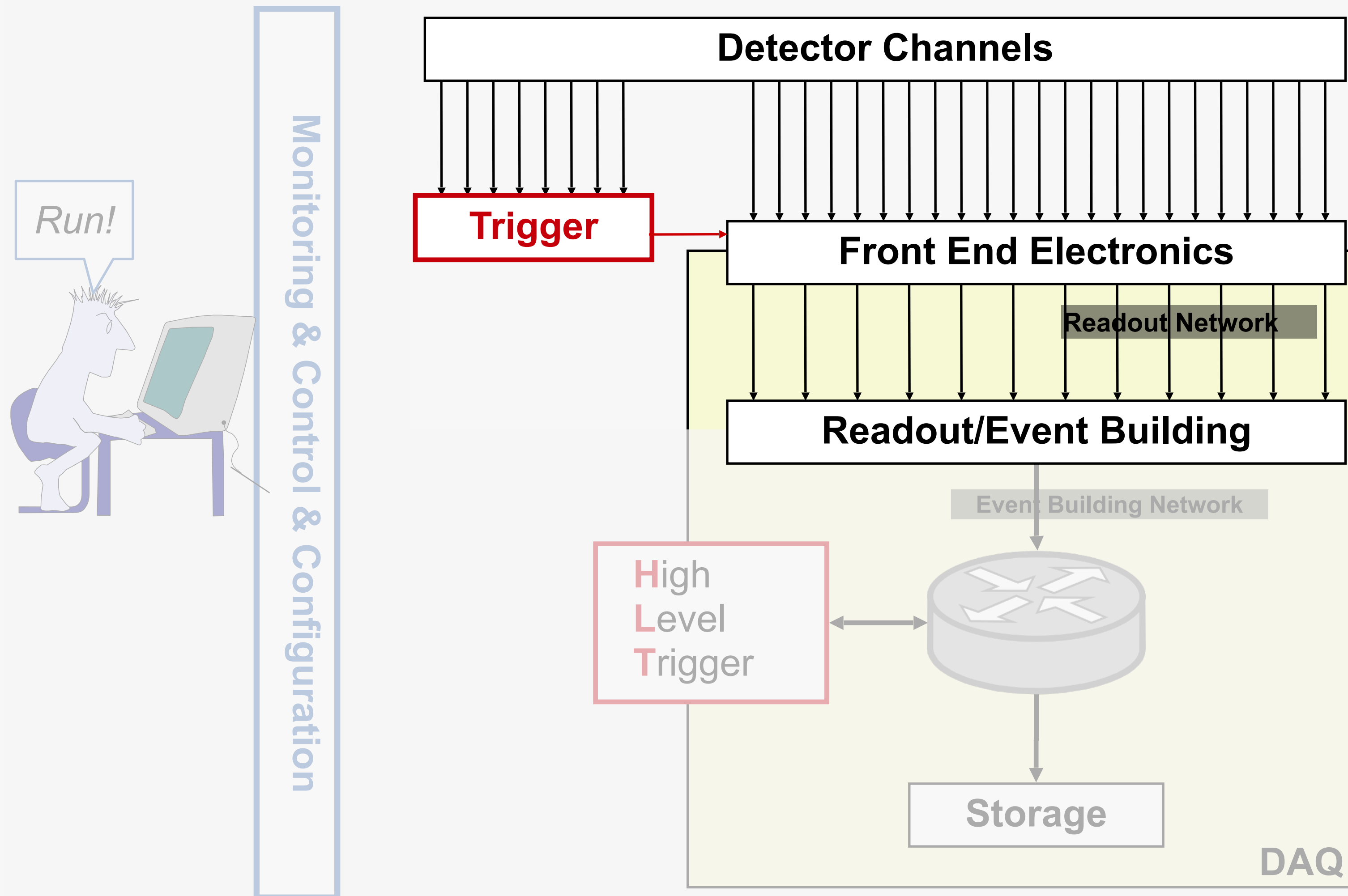
T-DAQ



T-DAQ



T-DAQ



Field Programmable Gate Arrays



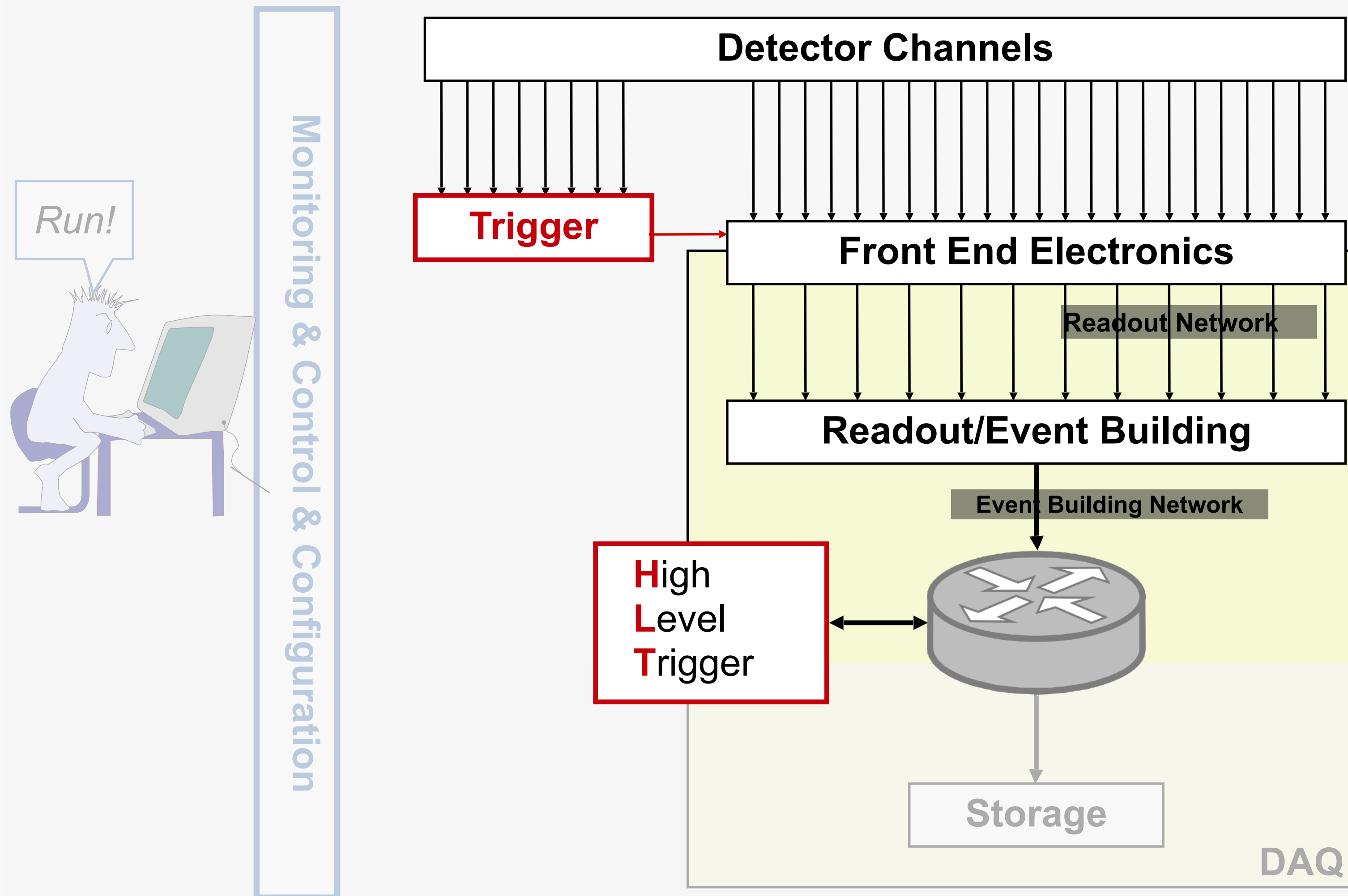
FPGAs are becoming the bread&butter of TDAQ systems

- Signal processing, data formatting, parallelizable tasks (pattern recognition), machine learning, ...

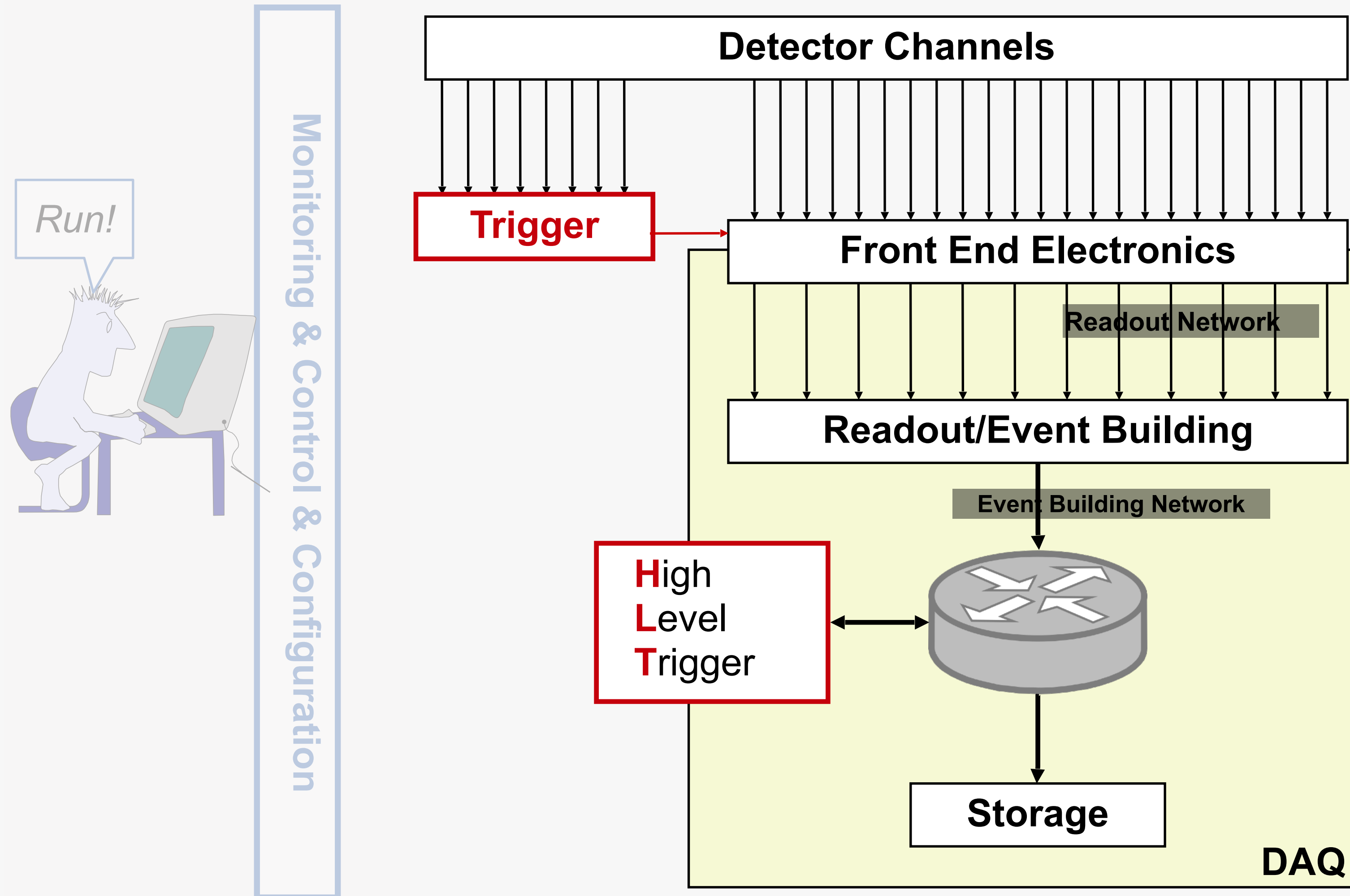
FPGA Programming Lecture

- Dr. K.Harder

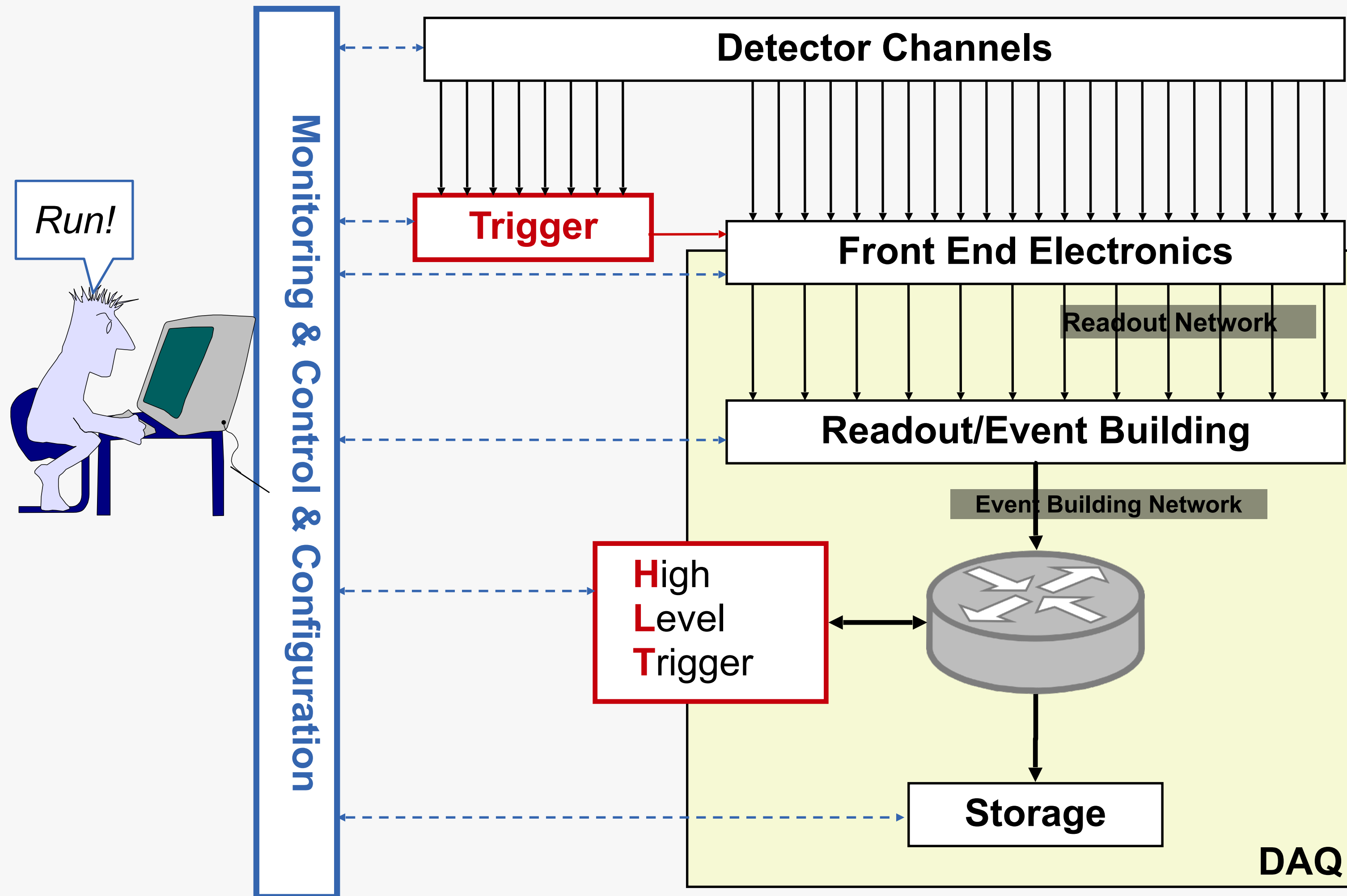
T-DAQ



T-DAQ



T-DAQ



The glue of your experiment

TIME	SEVERITY	APPLICATION	NAME	MESSAGE
21:16:58	INFORMATION	IGUI	INTERNAL	All done! IGUI is going
21:16:58	INFORMATION	IGUI	INTERNAL	Waiting for the "Data
21:16:58	INFORMATION	IGUI	INTERNAL	Waiting for the "Segme
21:16:58	ERROR	IGUI	INTERNAL	Failed to subscribe to Igui.IguiException\$ISE

Configuration

- ▶ The data taking setup

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

1. Introduction

1.1. What is DAQ?

1.2. Overall framework

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



with a toy model

Basic DAQ: periodic trigger

Eg: measure temperature at a fixed frequency

- Clock trigger

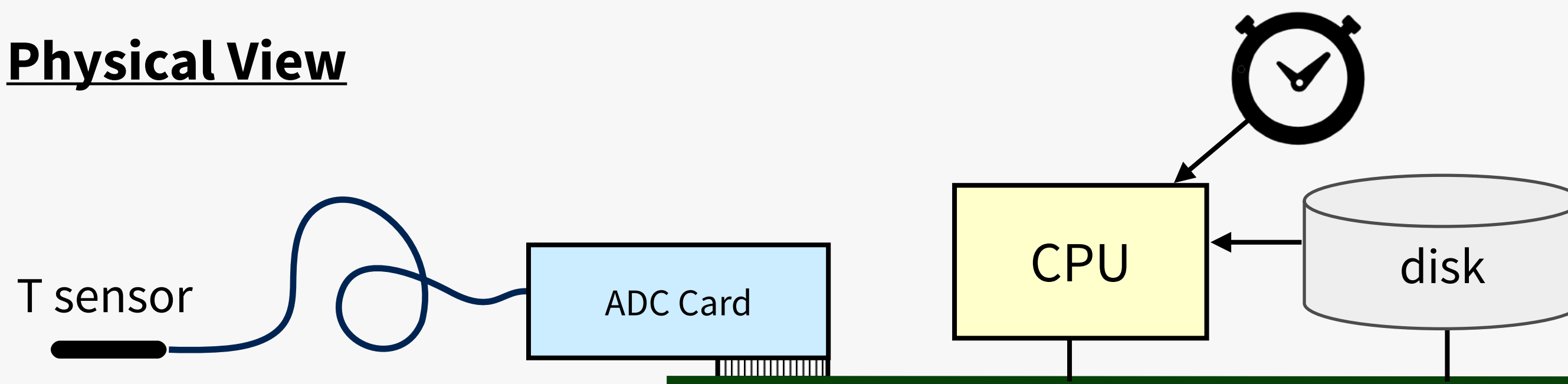
ADC performs analog to digital conversion, digitization (our front-end electronics)

- Encoding analog value into binary representation

CPU does

- Readout, Processing, Storage

Physical View



Basic DAQ: periodic trigger

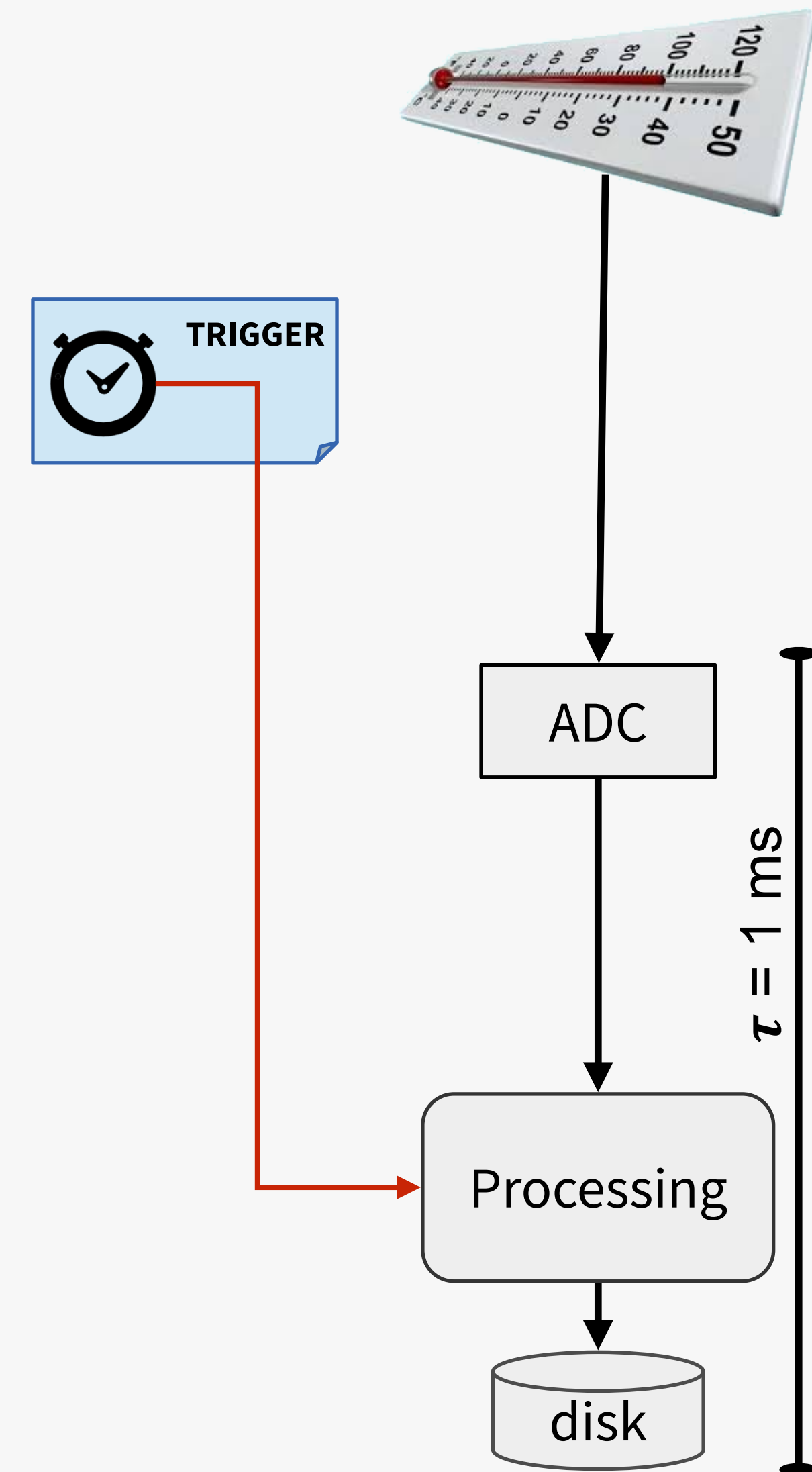
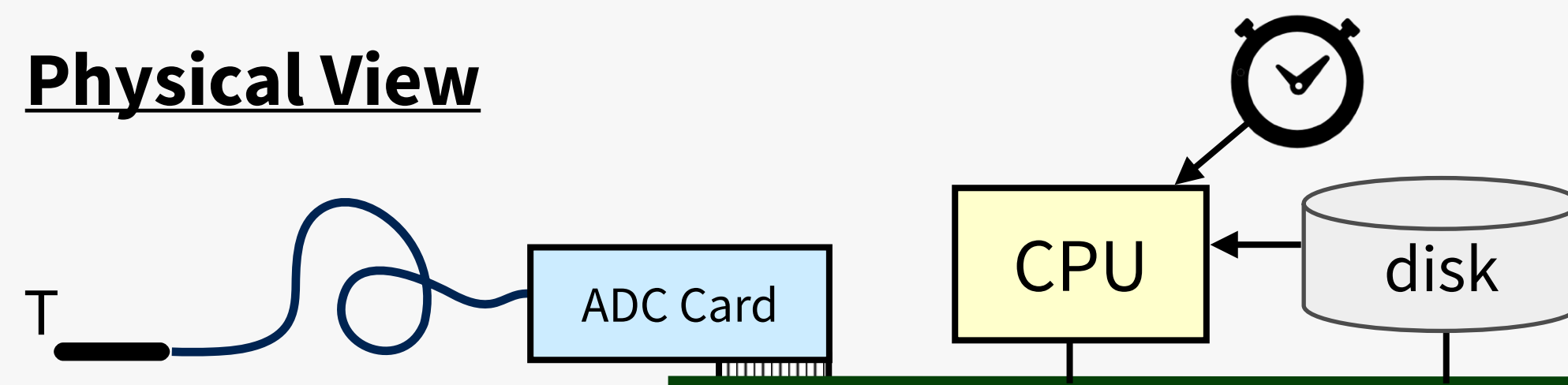
System clearly limited by the time τ to process an “event”

- ADC conversion +
CPU processing +
Storage

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

- E.g.: $\tau = 1 \text{ ms}$ $\Rightarrow R = 1/\tau = 1 \text{ kHz}$

Physical View



Basic DAQ: “real” trigger

Events asynchronous and unpredictable

- E.g.: beta decay studies

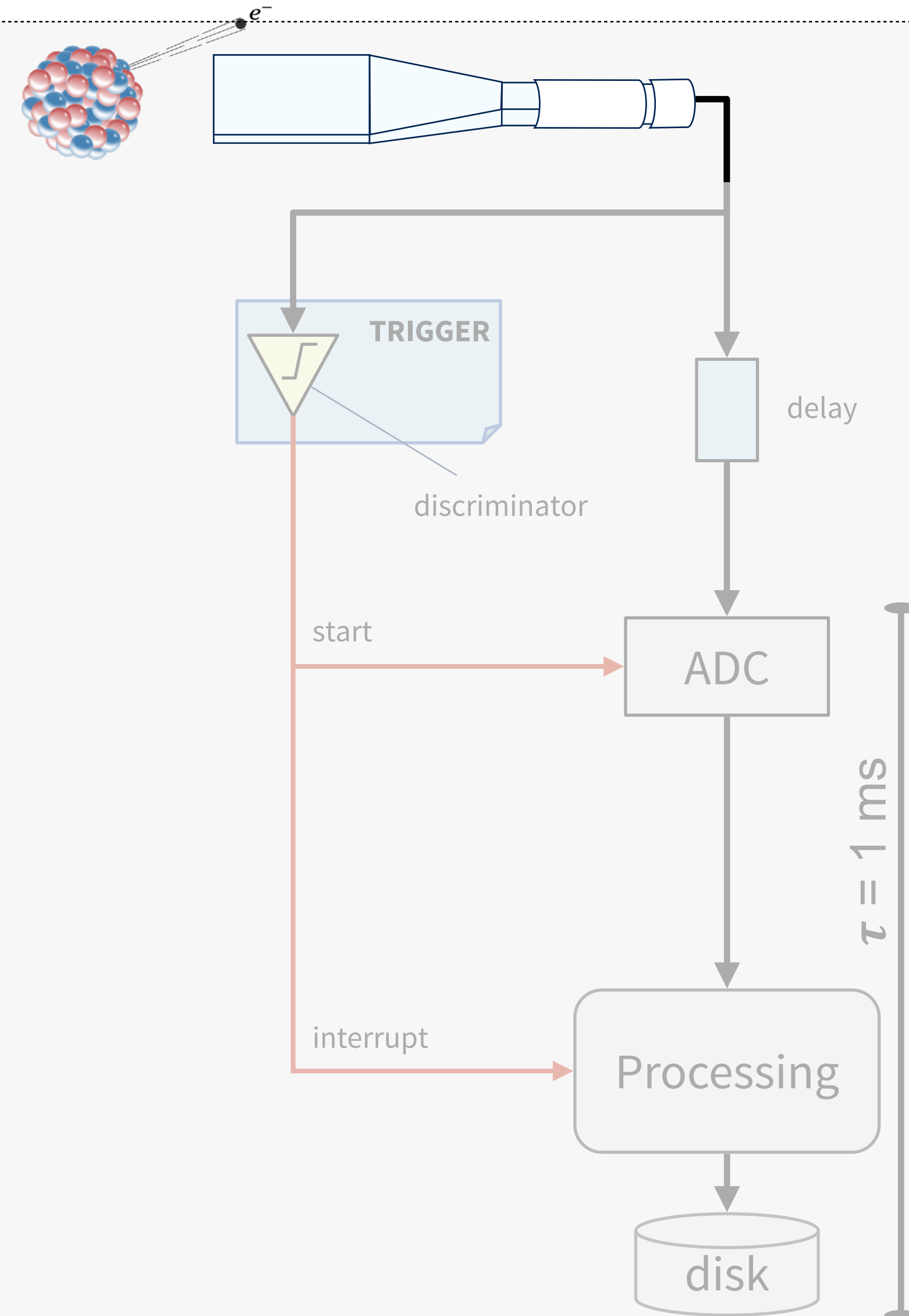
A physics trigger is needed

- **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

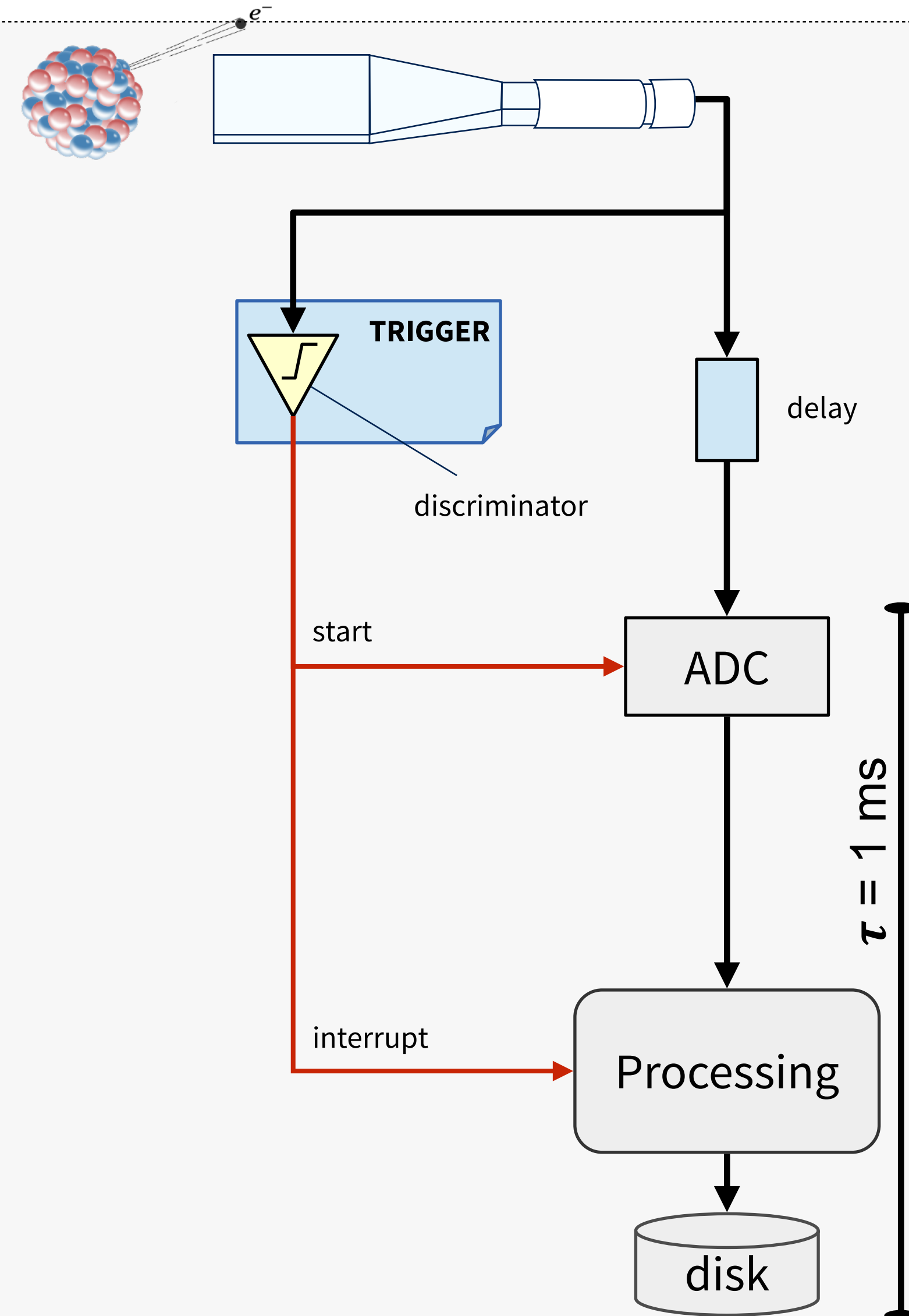
A physics trigger is needed

- **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

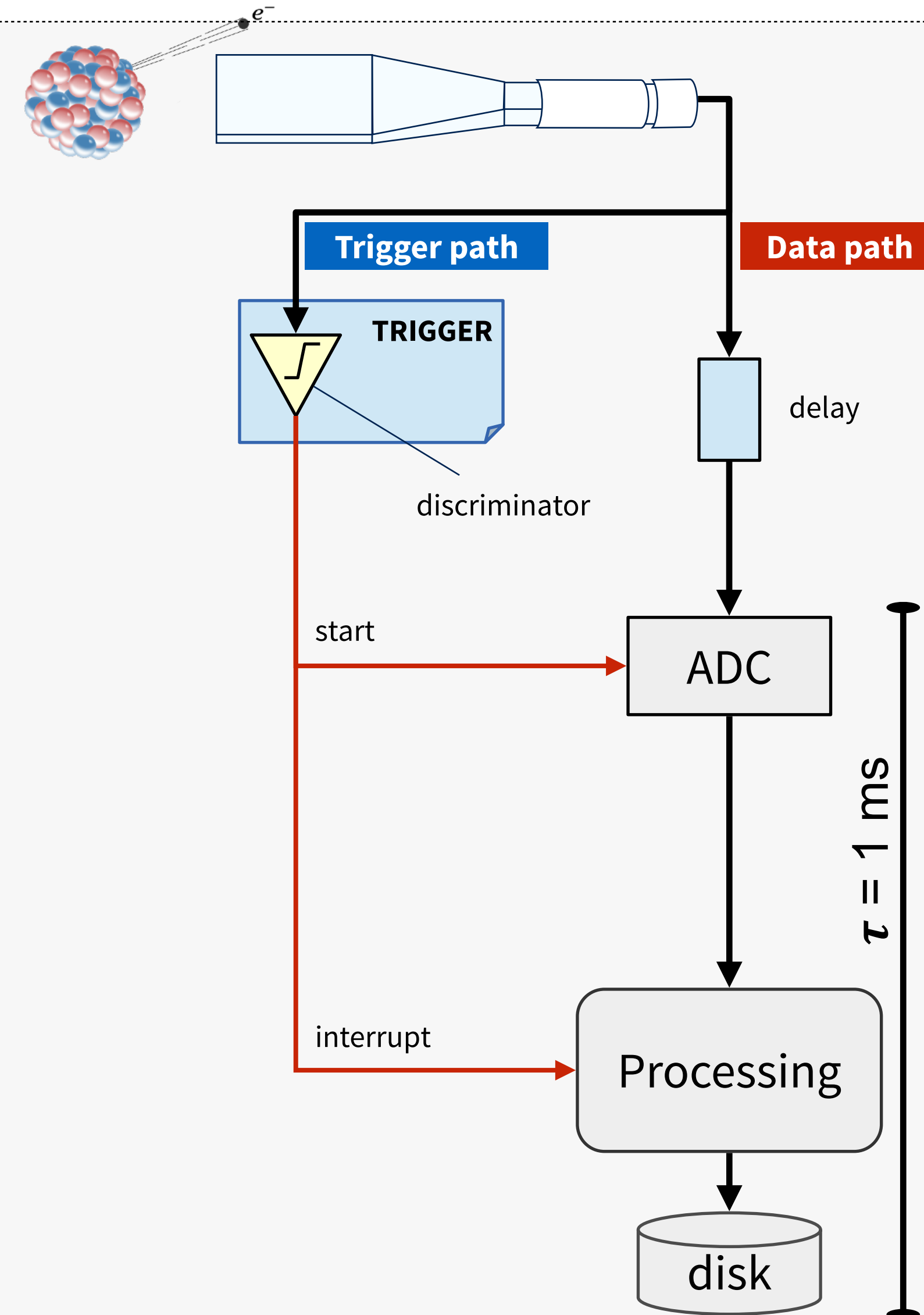
A physics trigger is needed

- **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



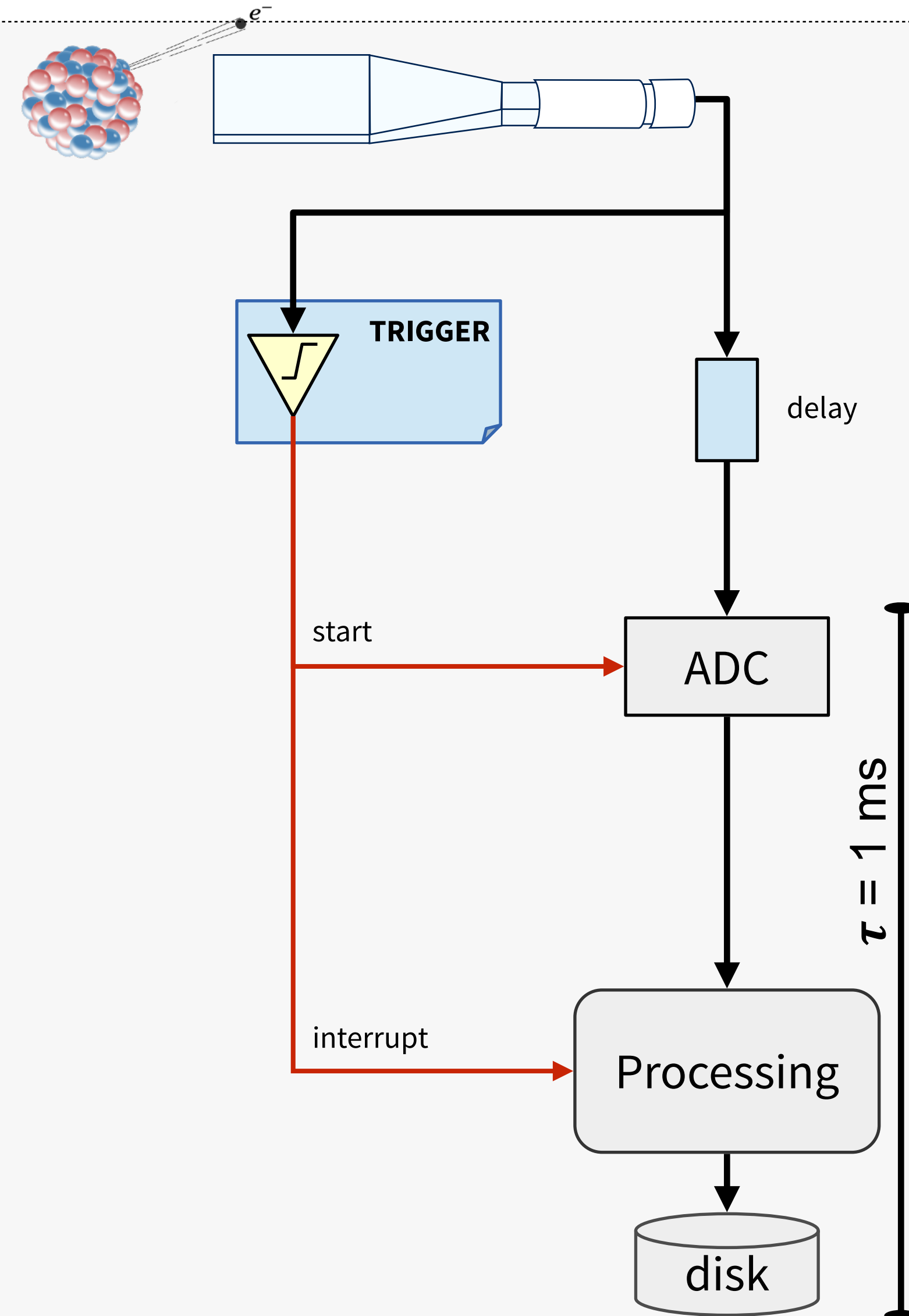
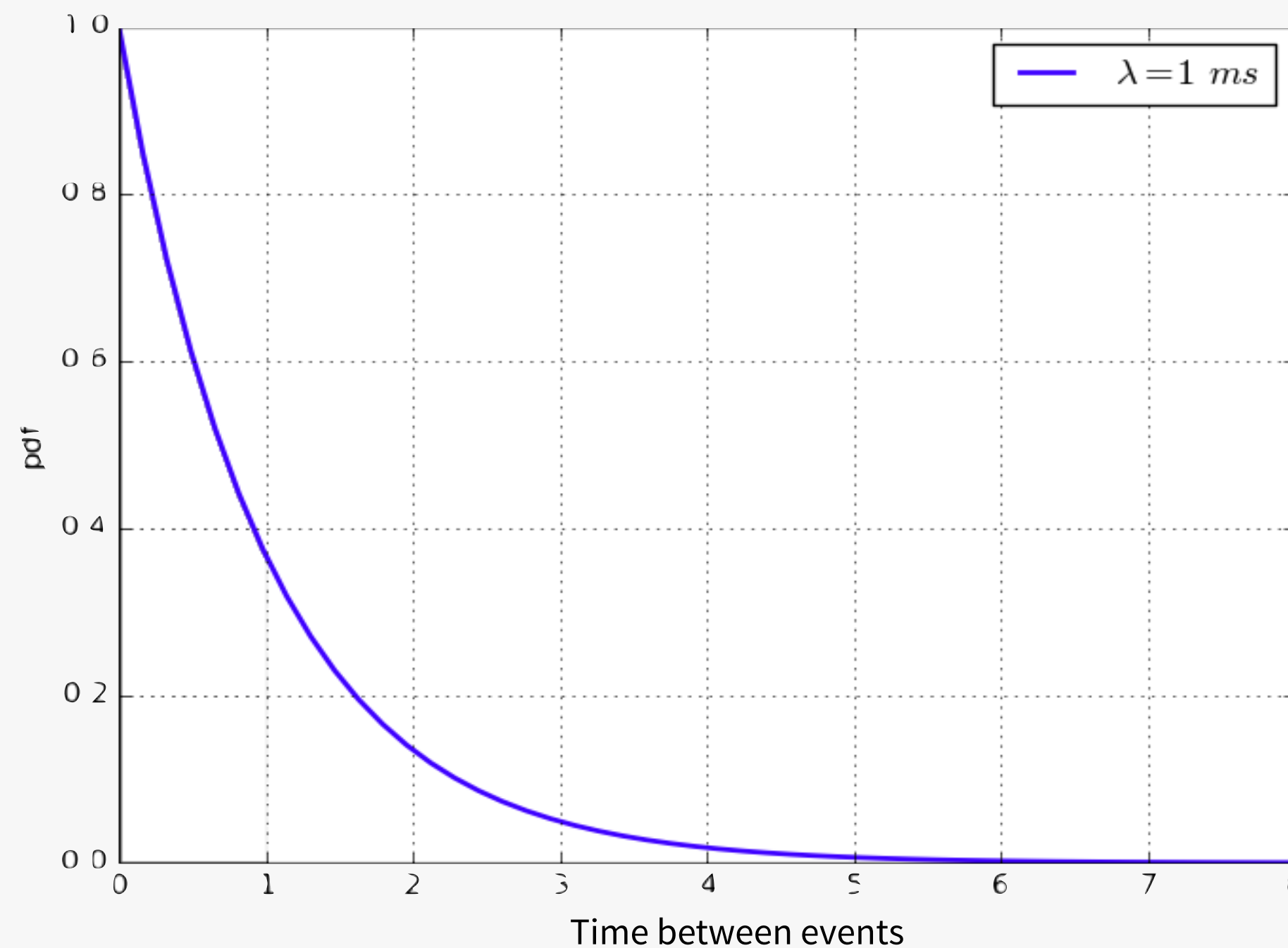
Basic DAQ: “real” trigger

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



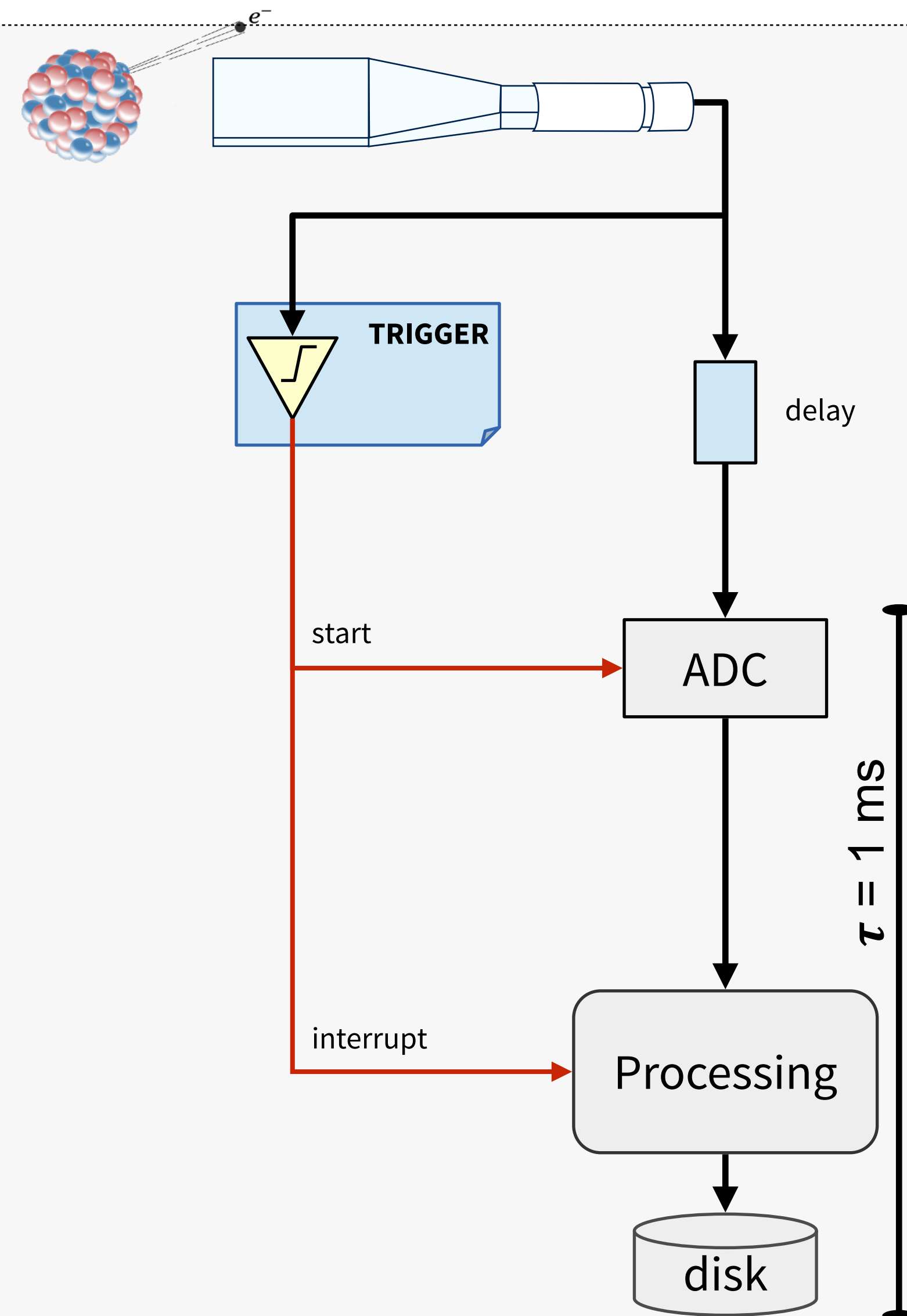
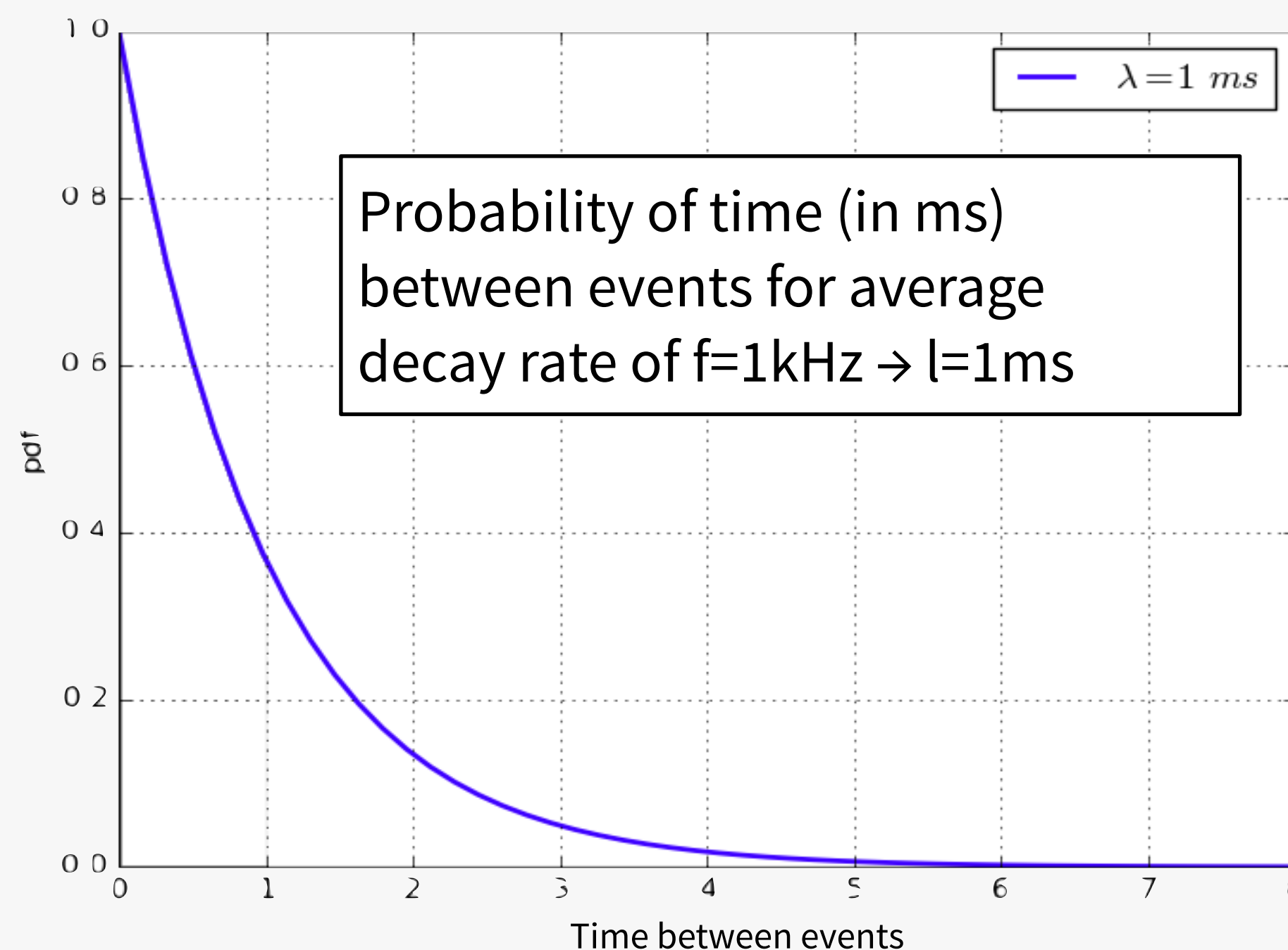
Basic DAQ: “real” trigger

Stochastic process

- Fluctuations in time between events

Let's assume for example

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



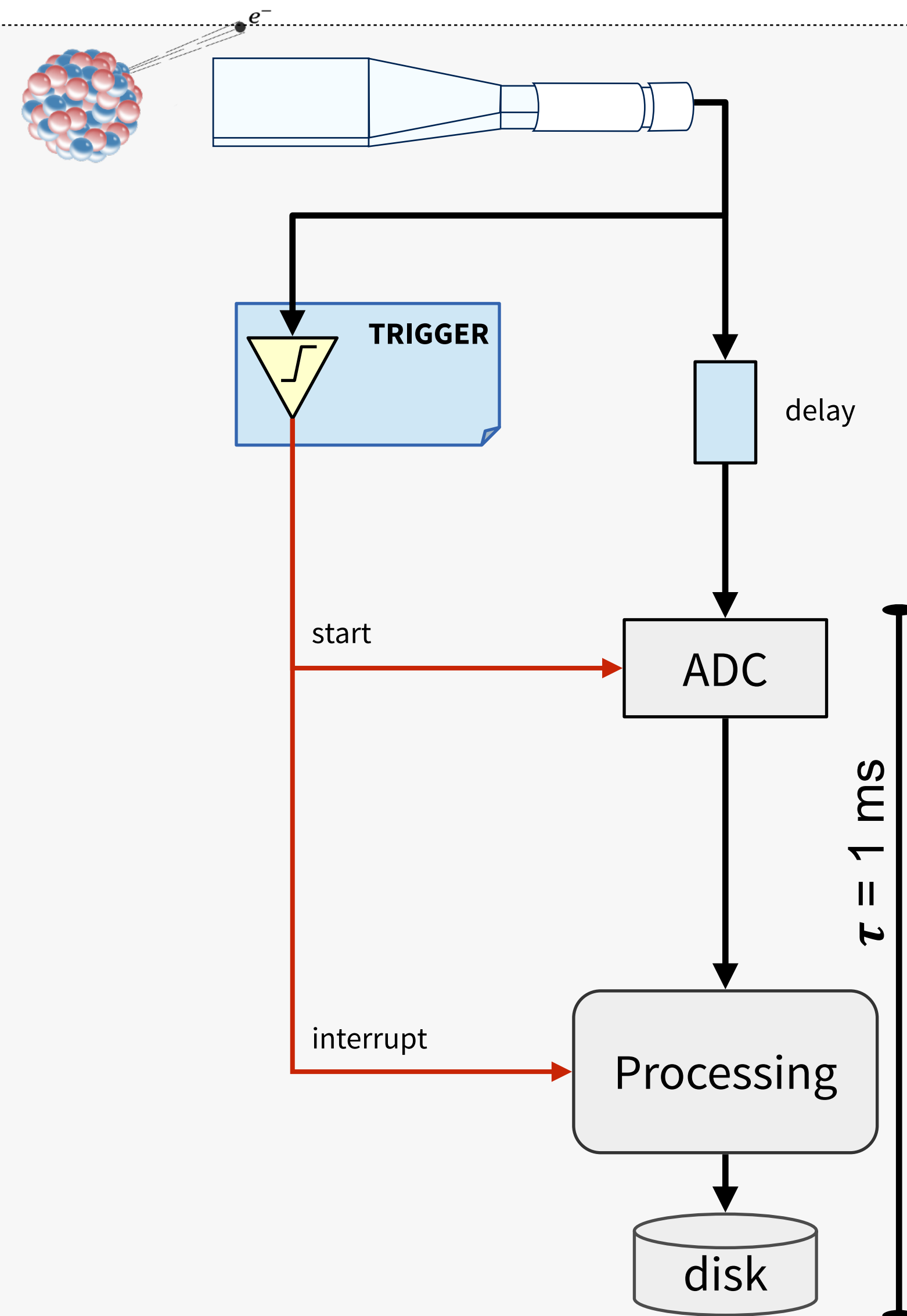
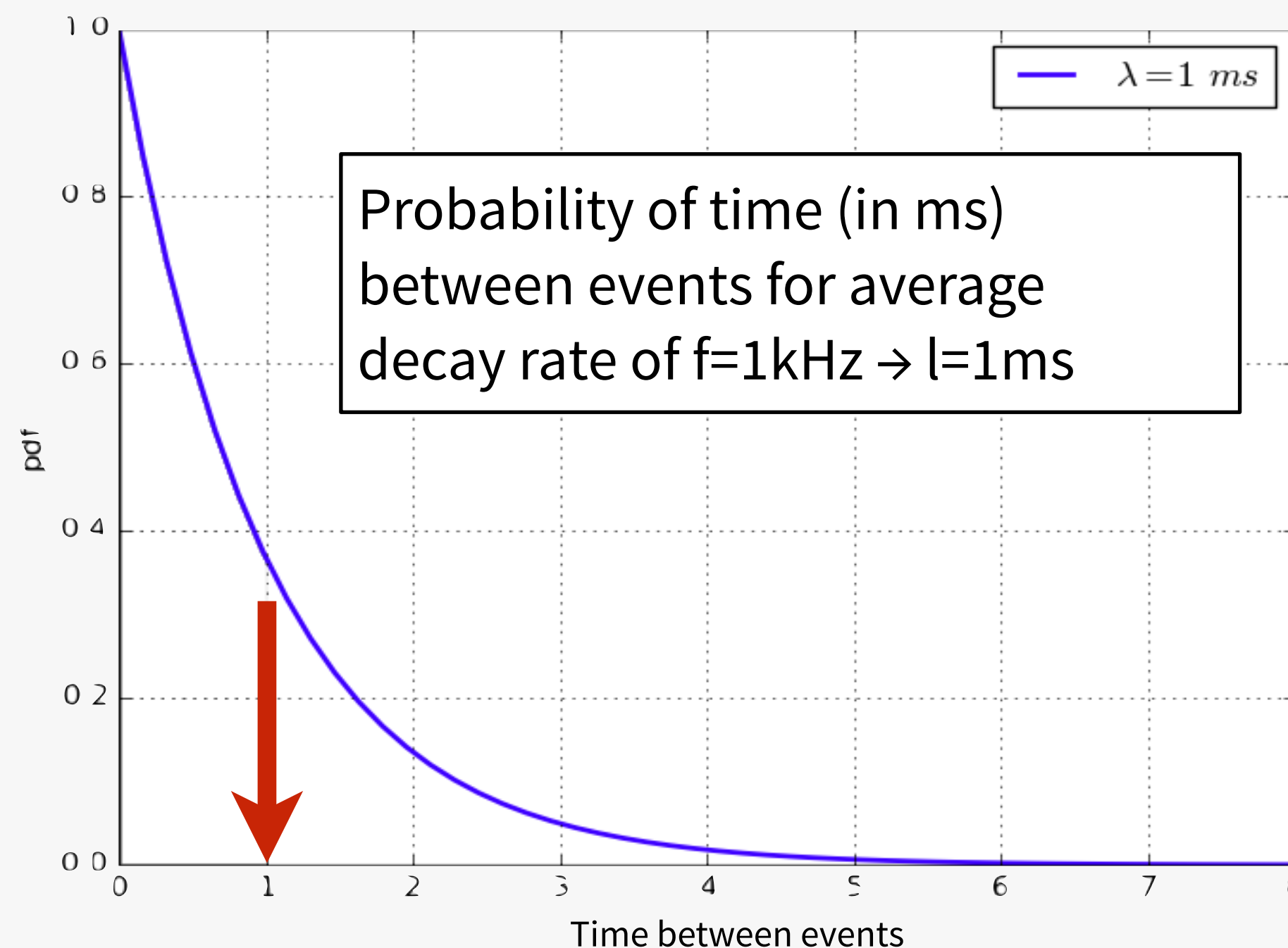
Basic DAQ: “real” trigger

Stochastic process

- Fluctuations in time between events

Let's assume for example

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



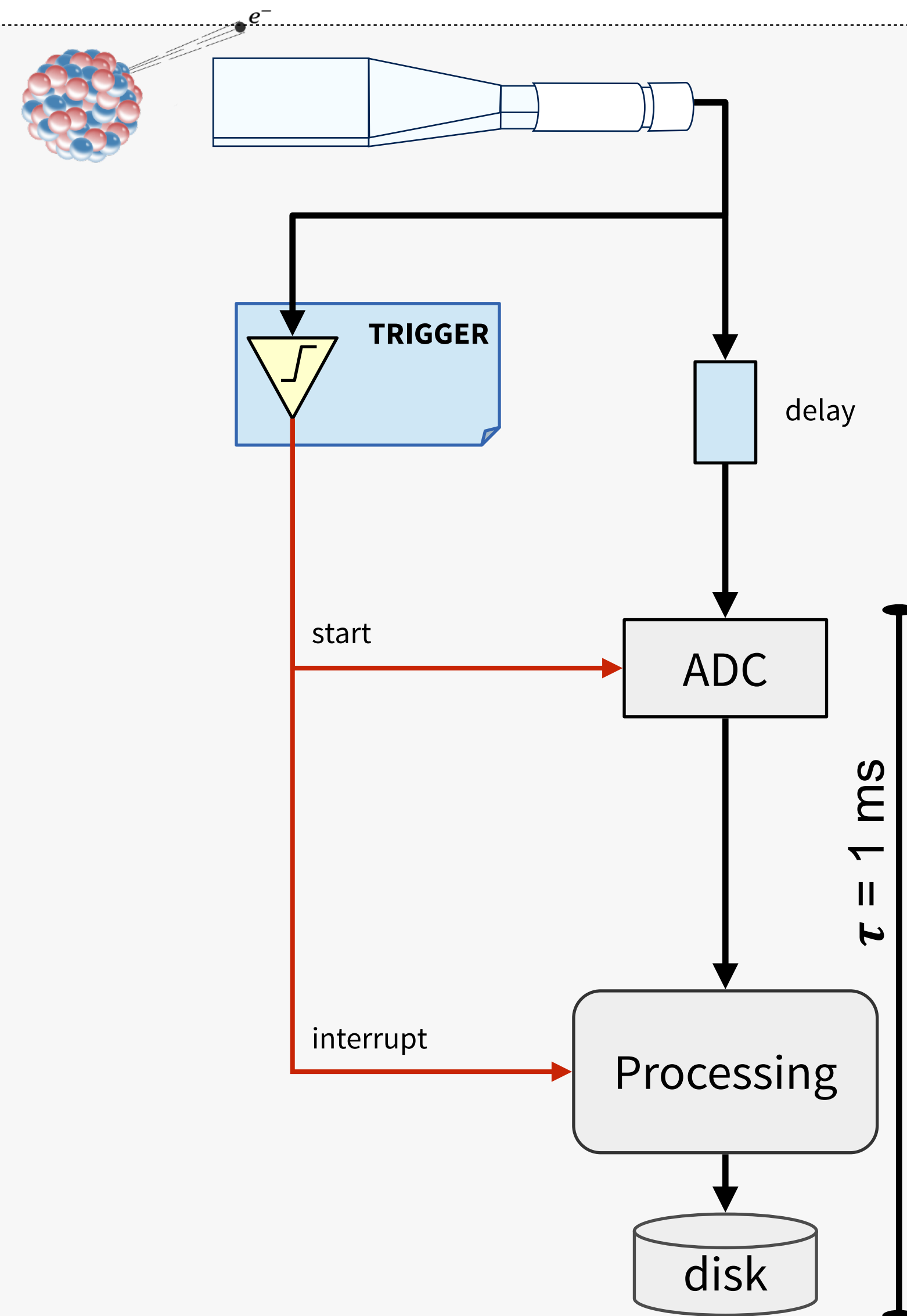
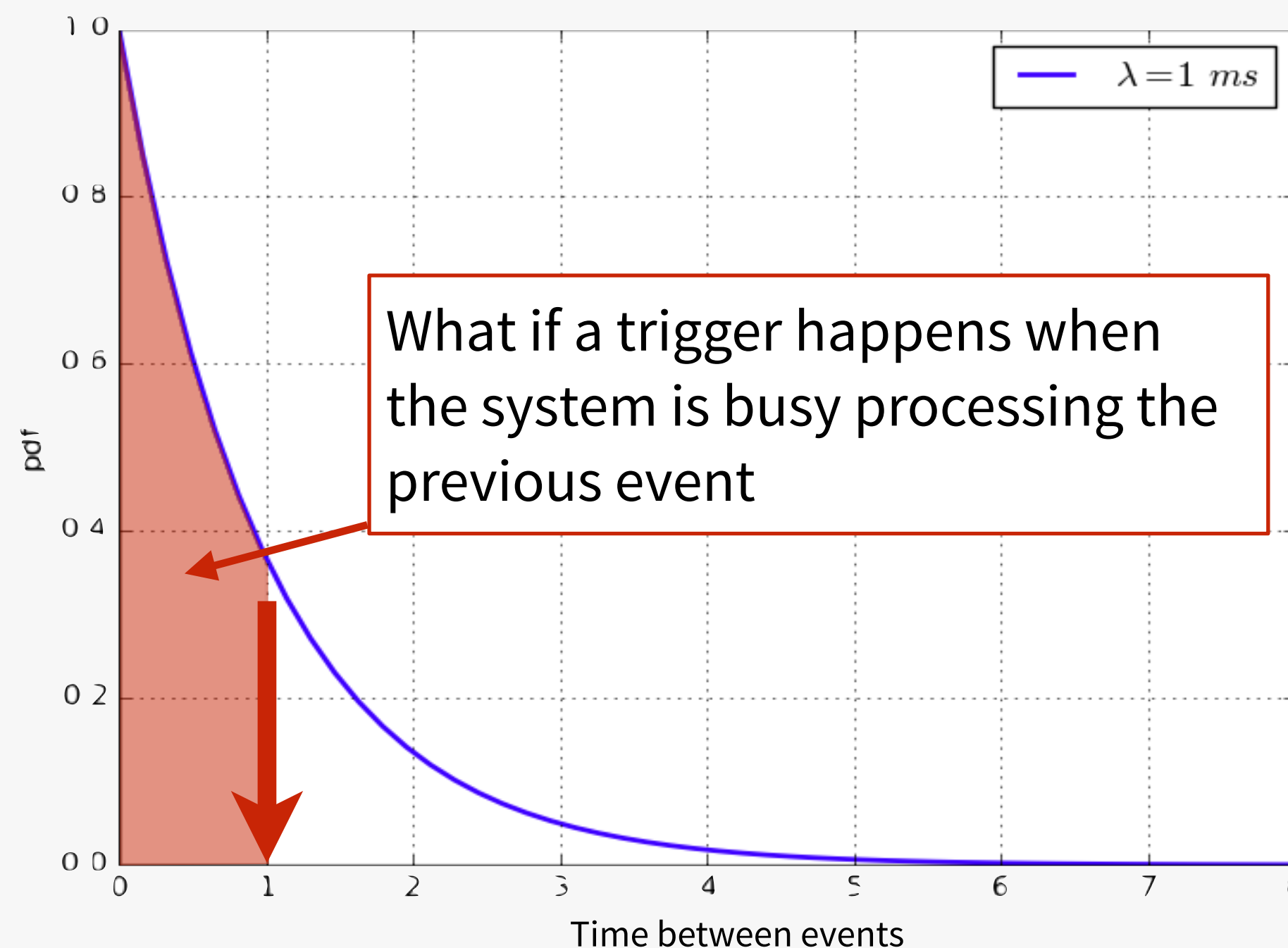
Basic DAQ: “real” trigger

Stochastic process

- Fluctuations in time between events

Let's assume for example

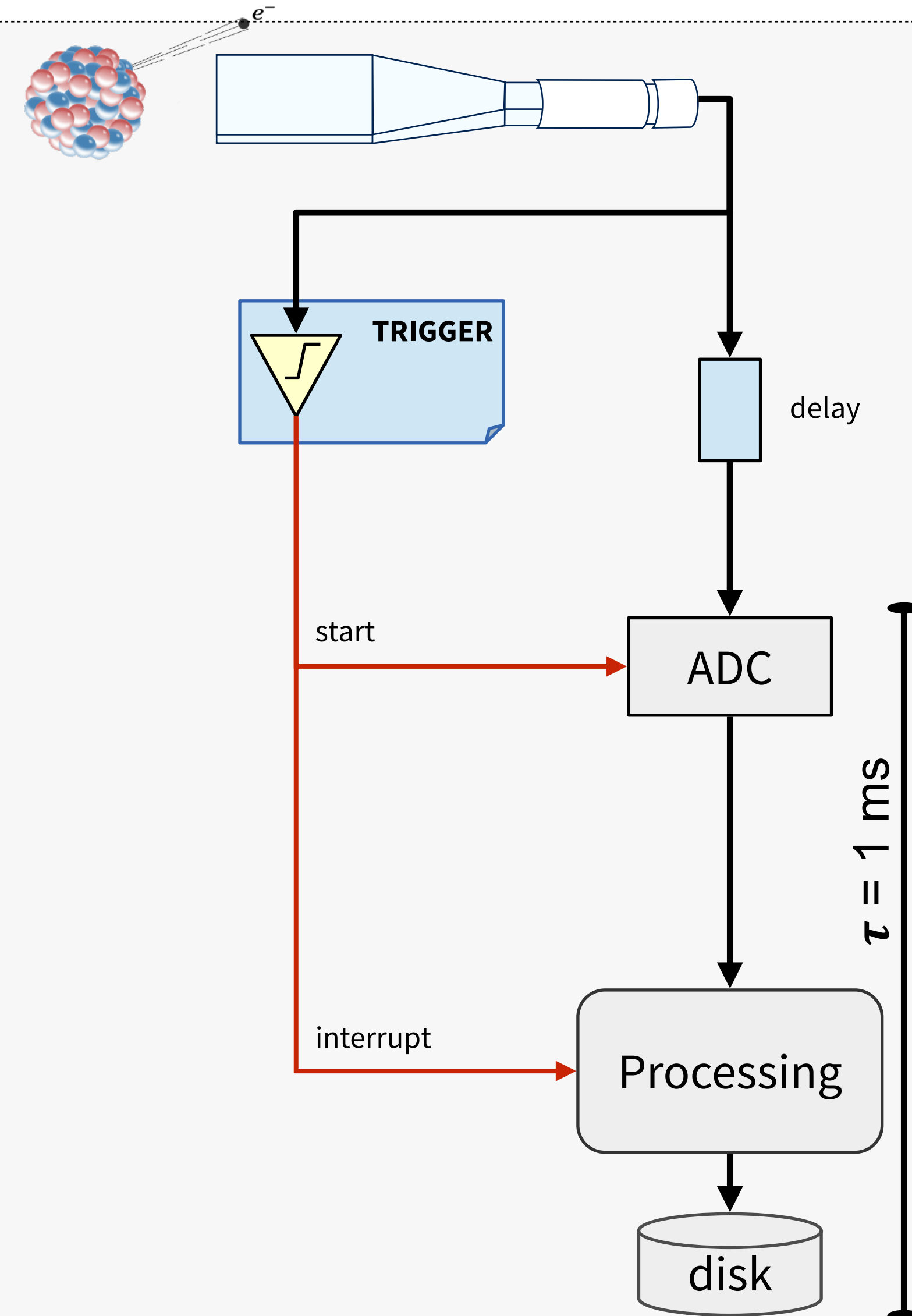
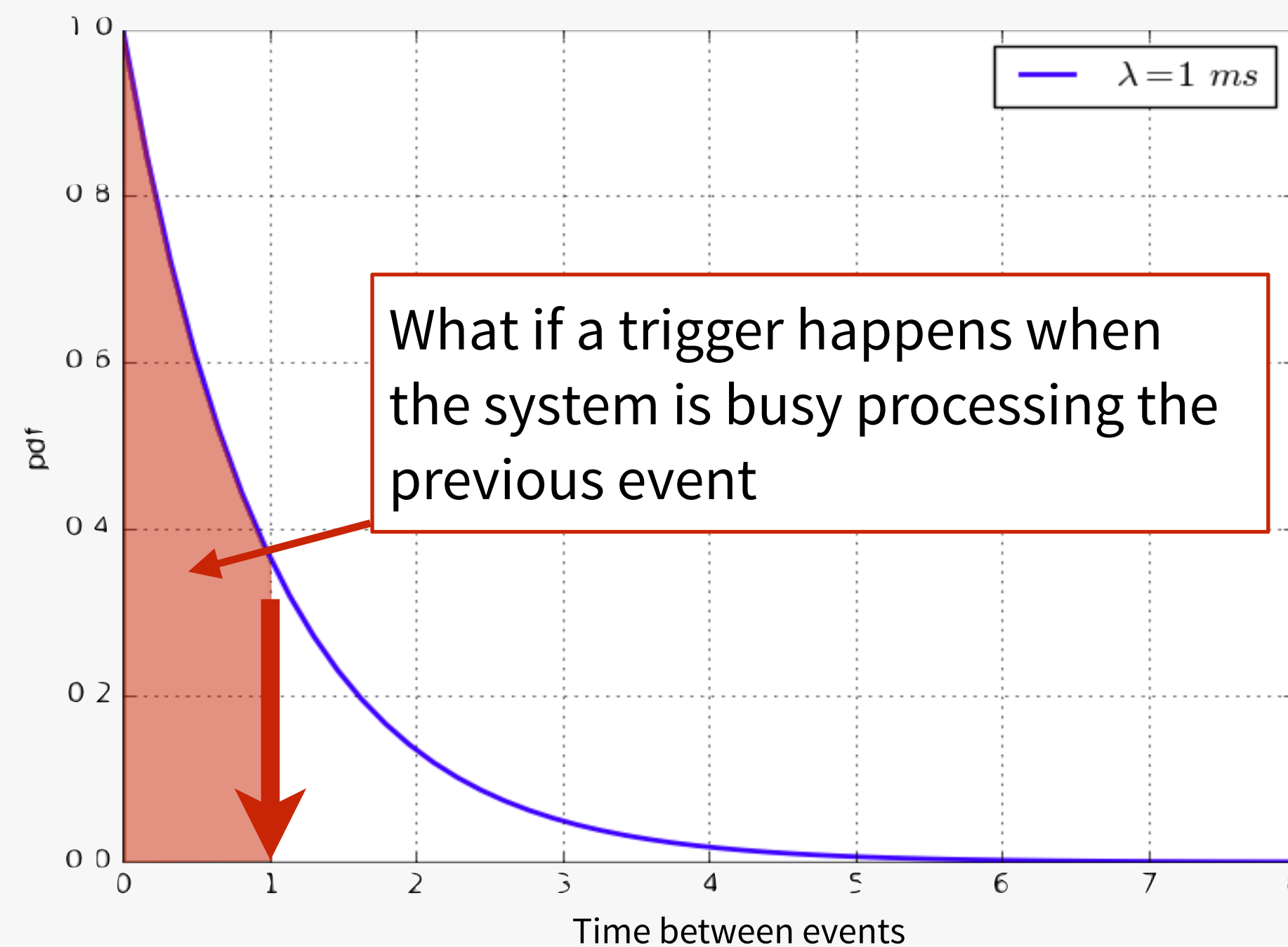
- physics rate $f = 1$ kHz, i.e. $\lambda = 1$ ms
- and, as before, $\tau = 1$ ms



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 screwed up



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

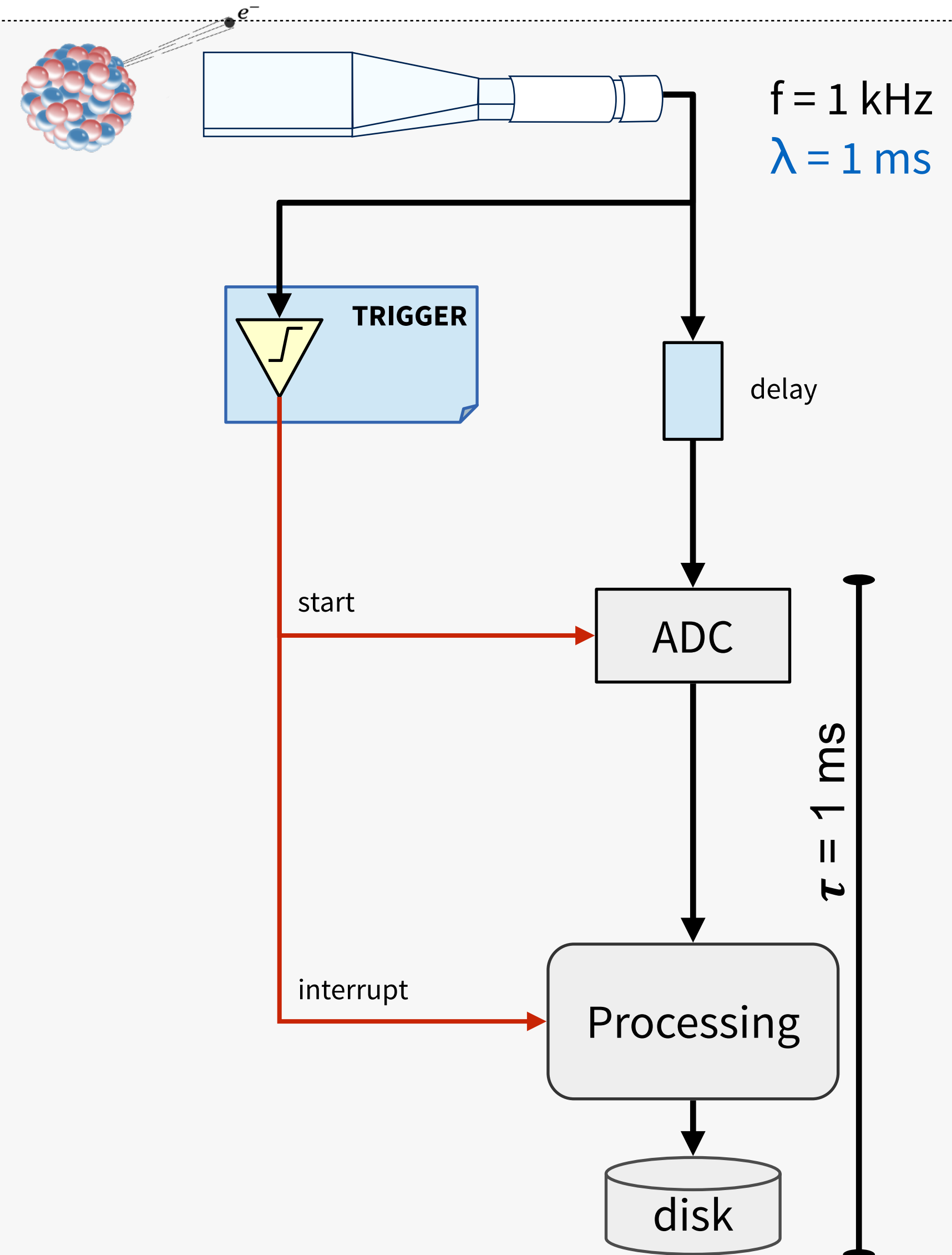


Busy logic

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 (**flip-flop**)

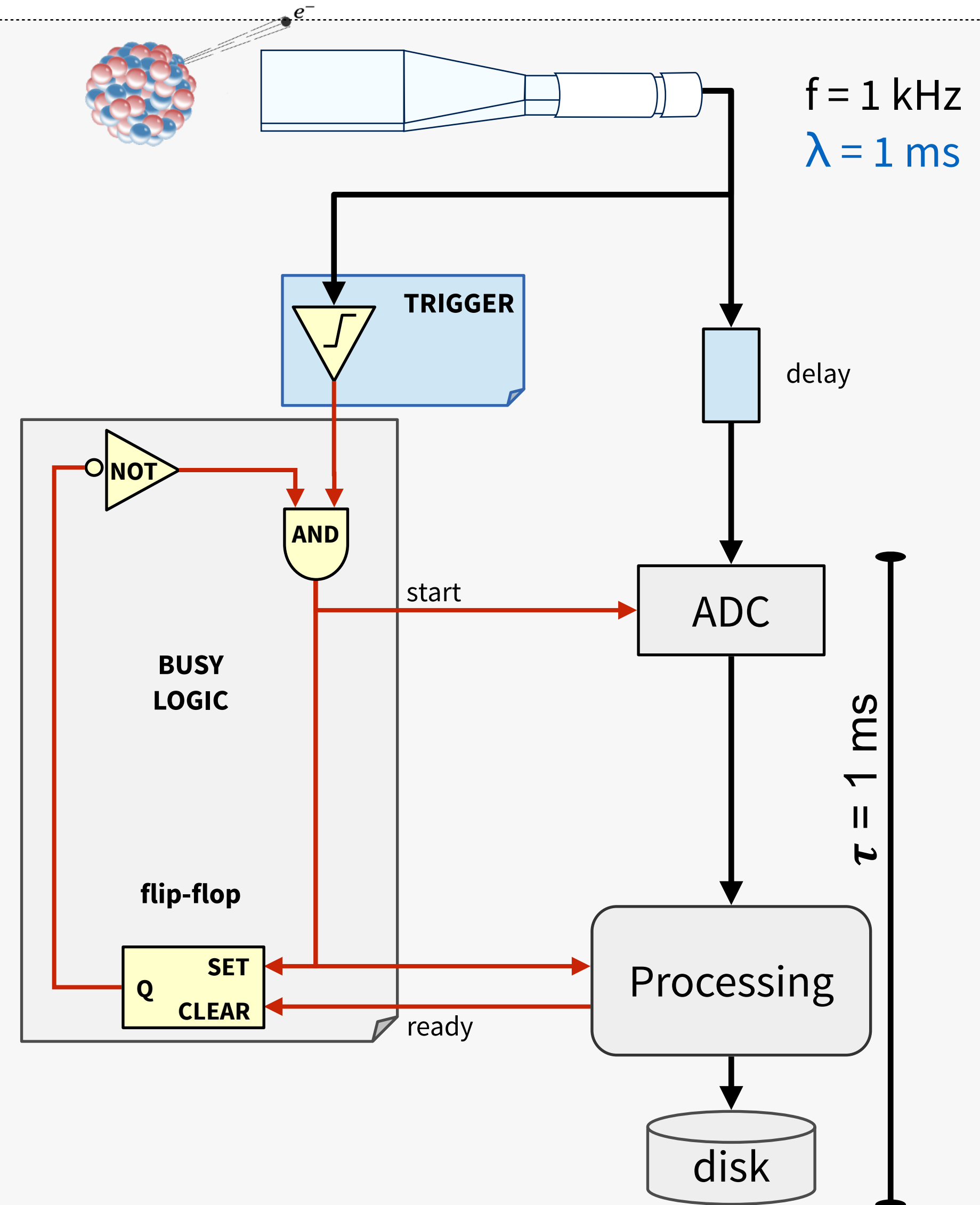


Busy logic

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 (**flip-flop**)

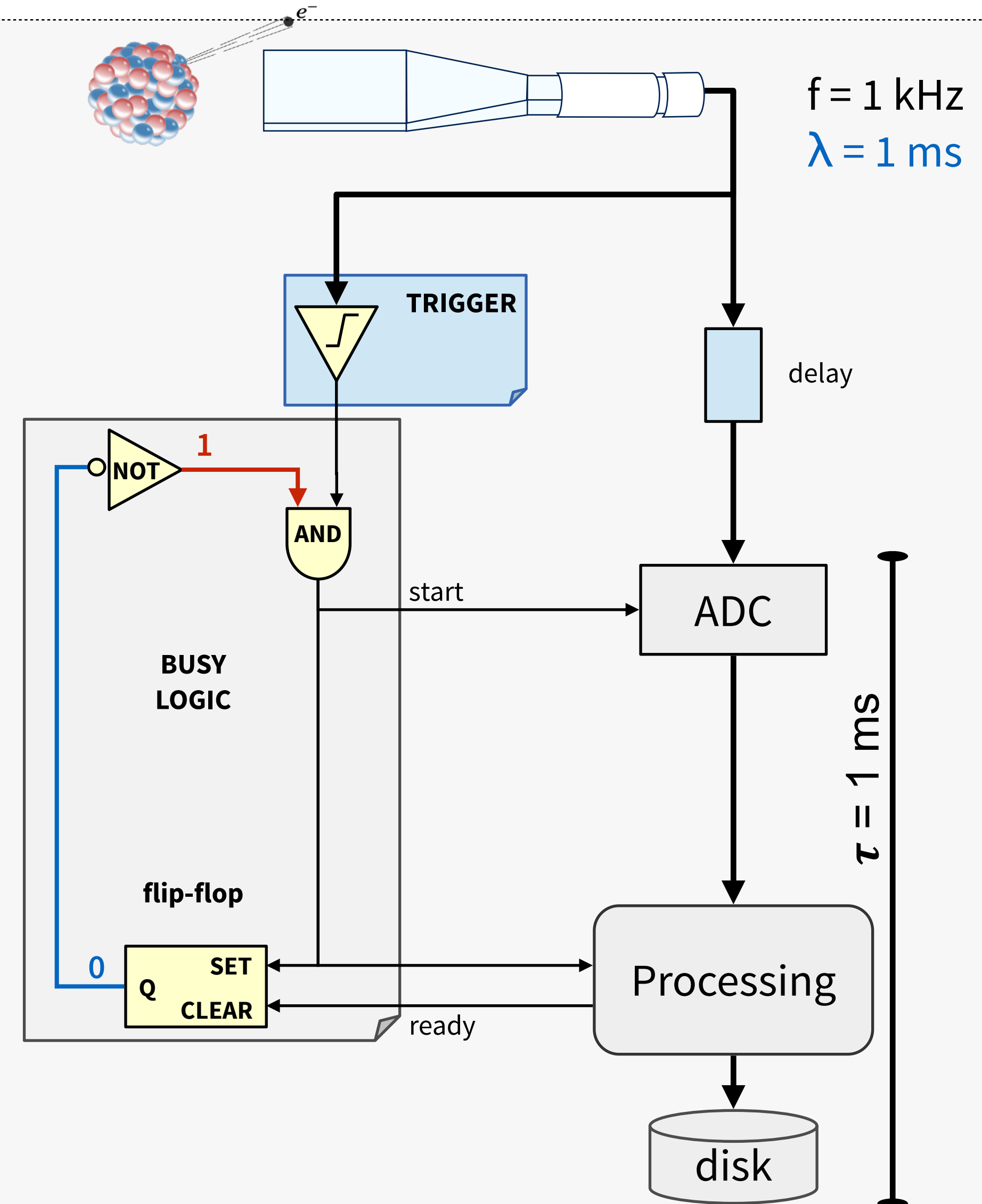


Busy logic

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

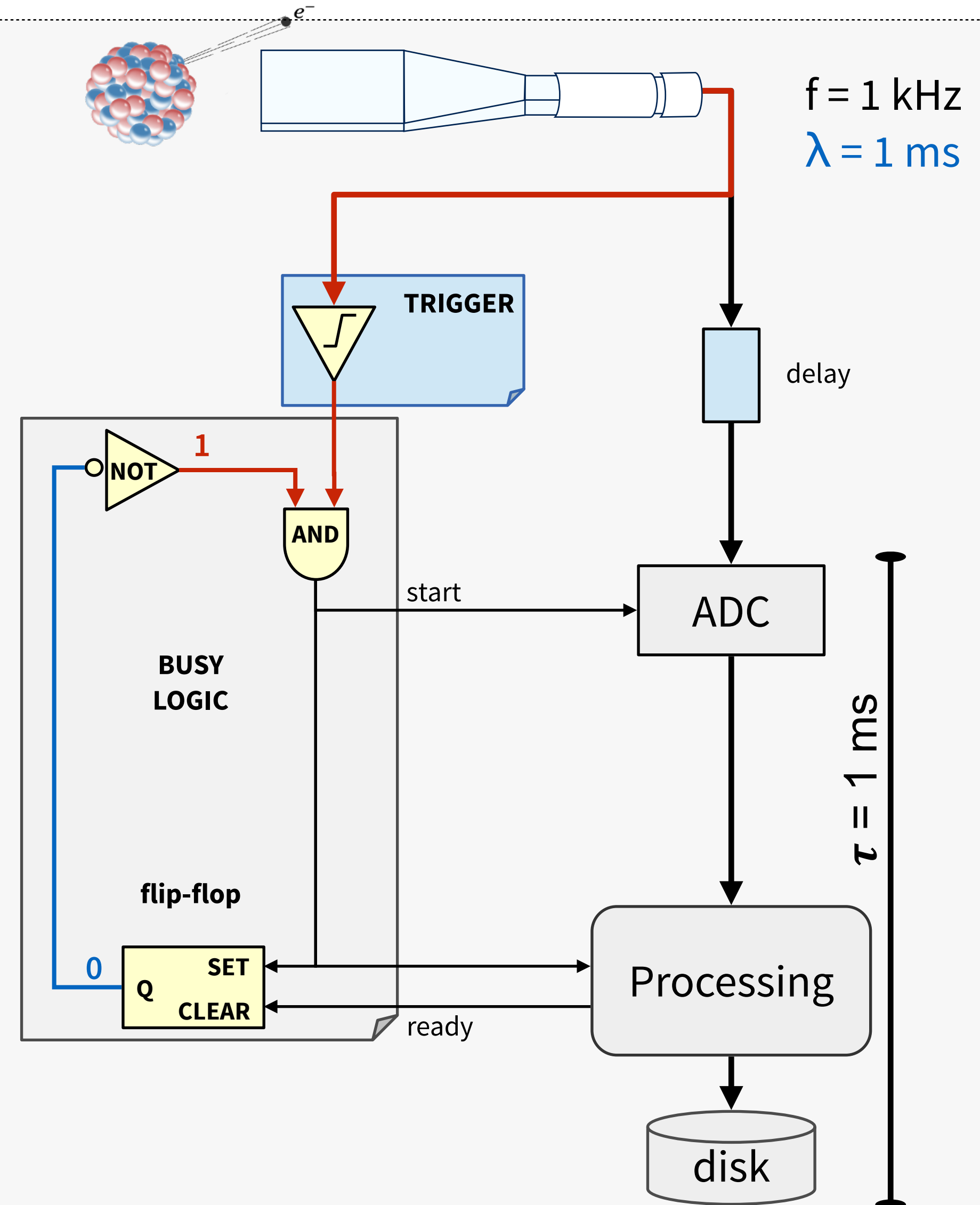


Busy logic

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)

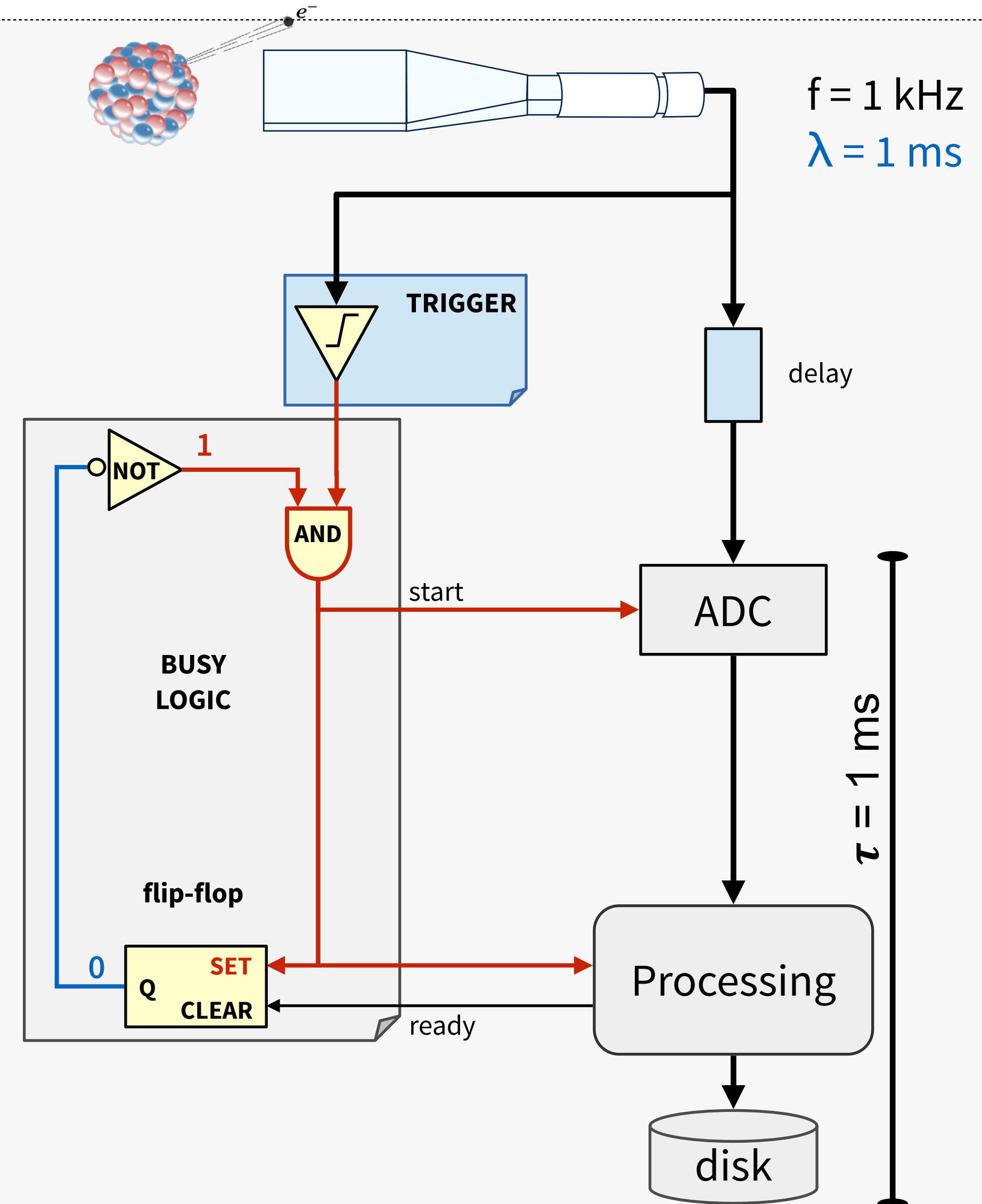


Busy logic

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)

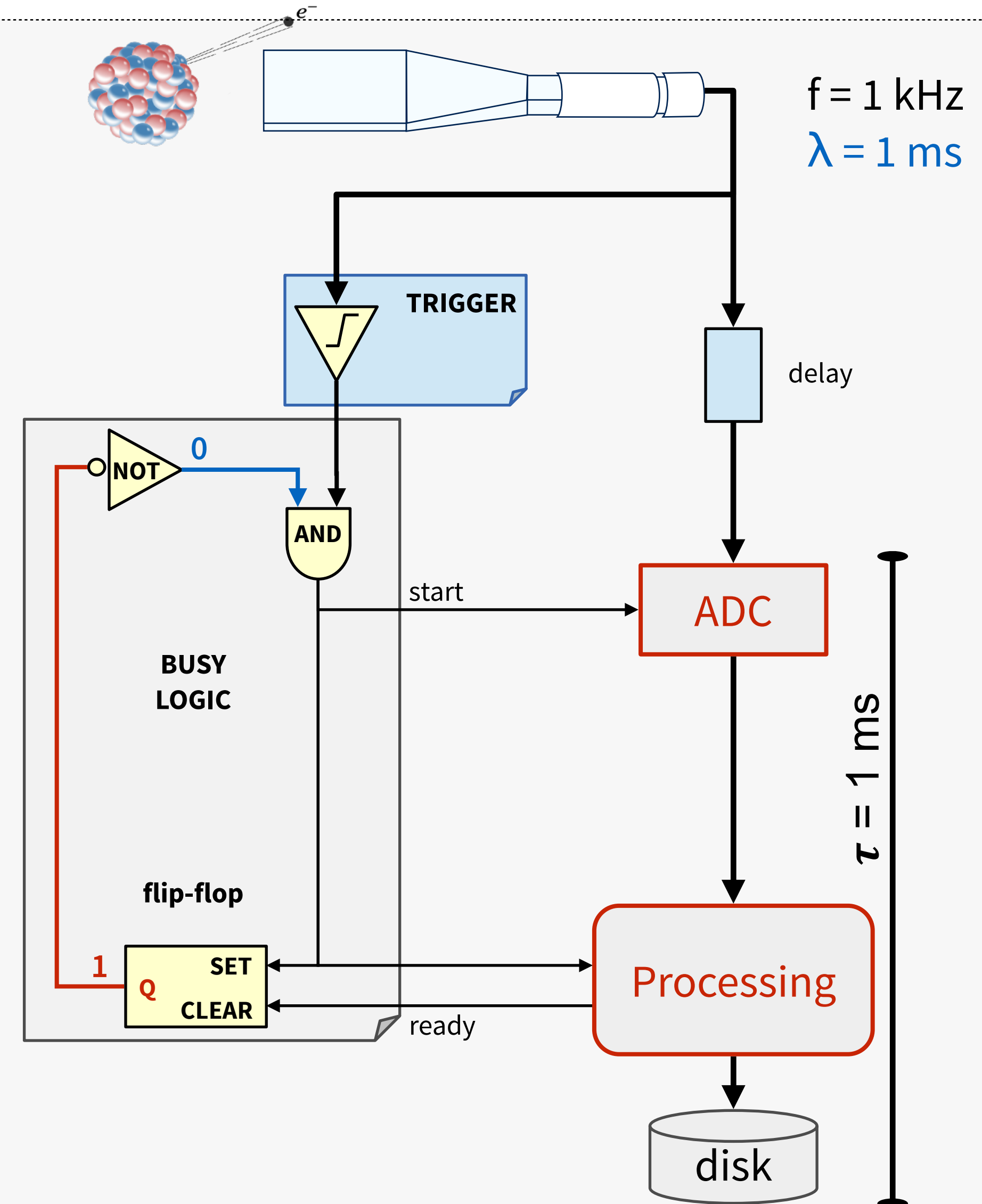


Busy logic

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)

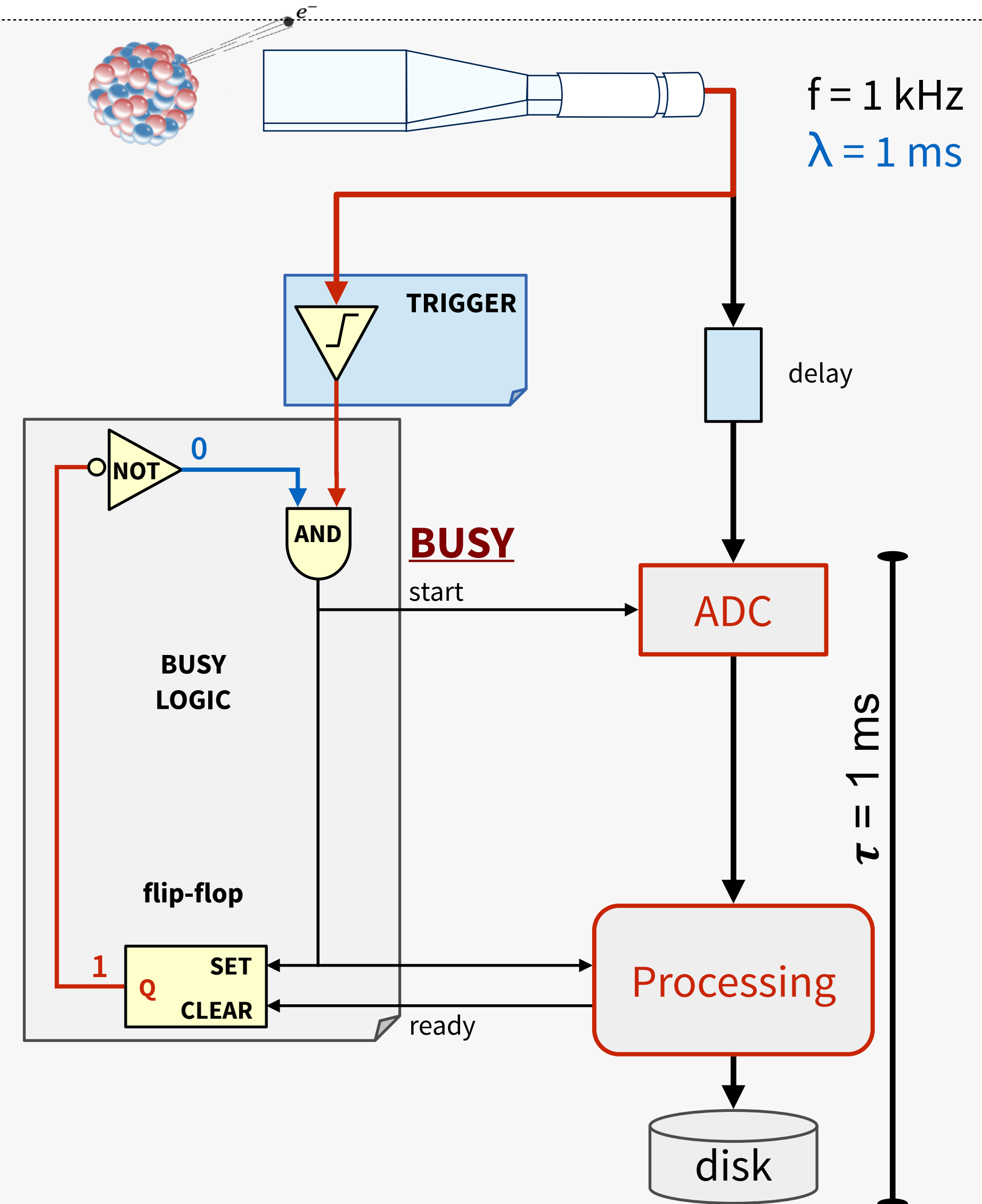


Busy logic

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)

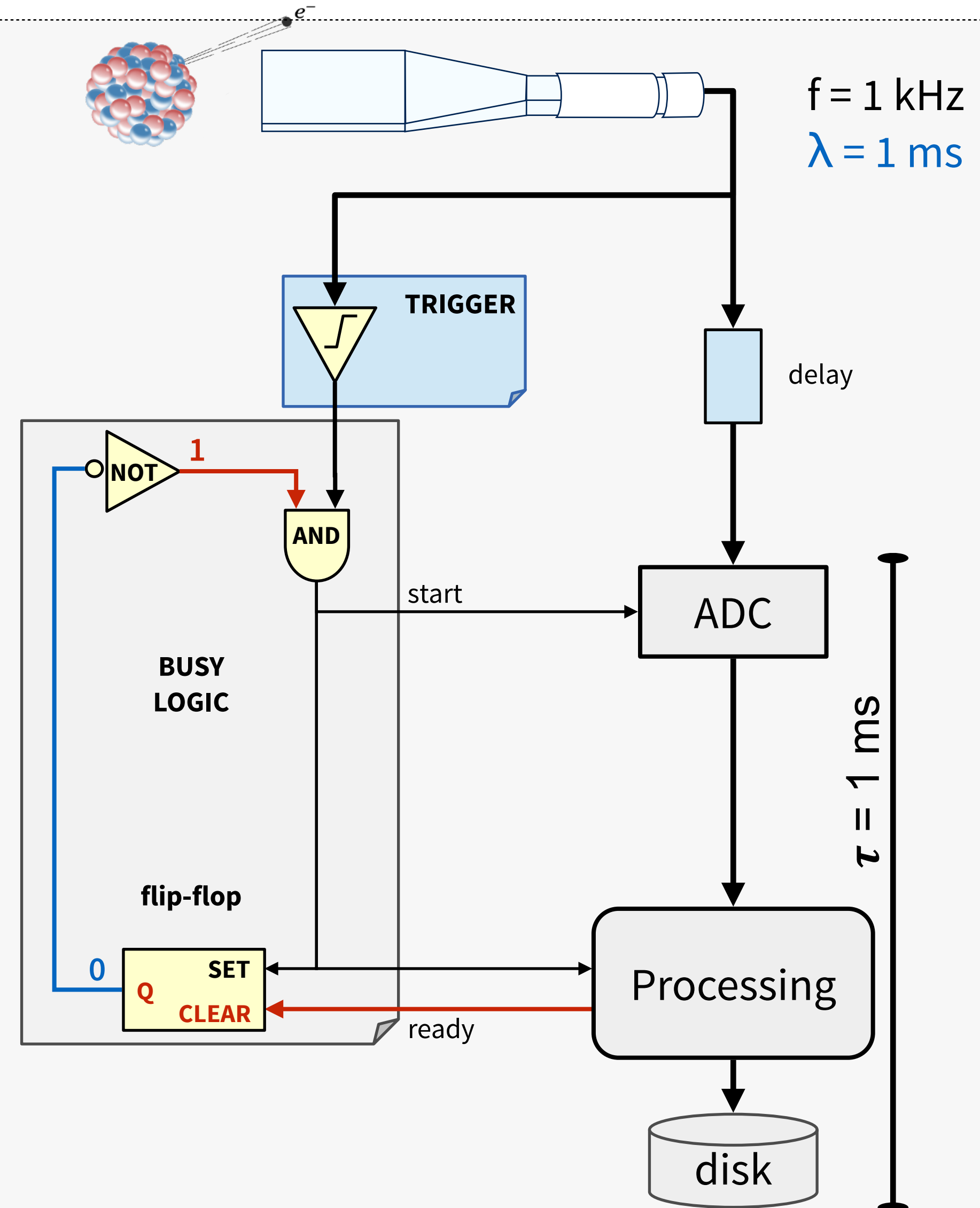
Any new trigger is inhibited by the AND gate (busy)



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

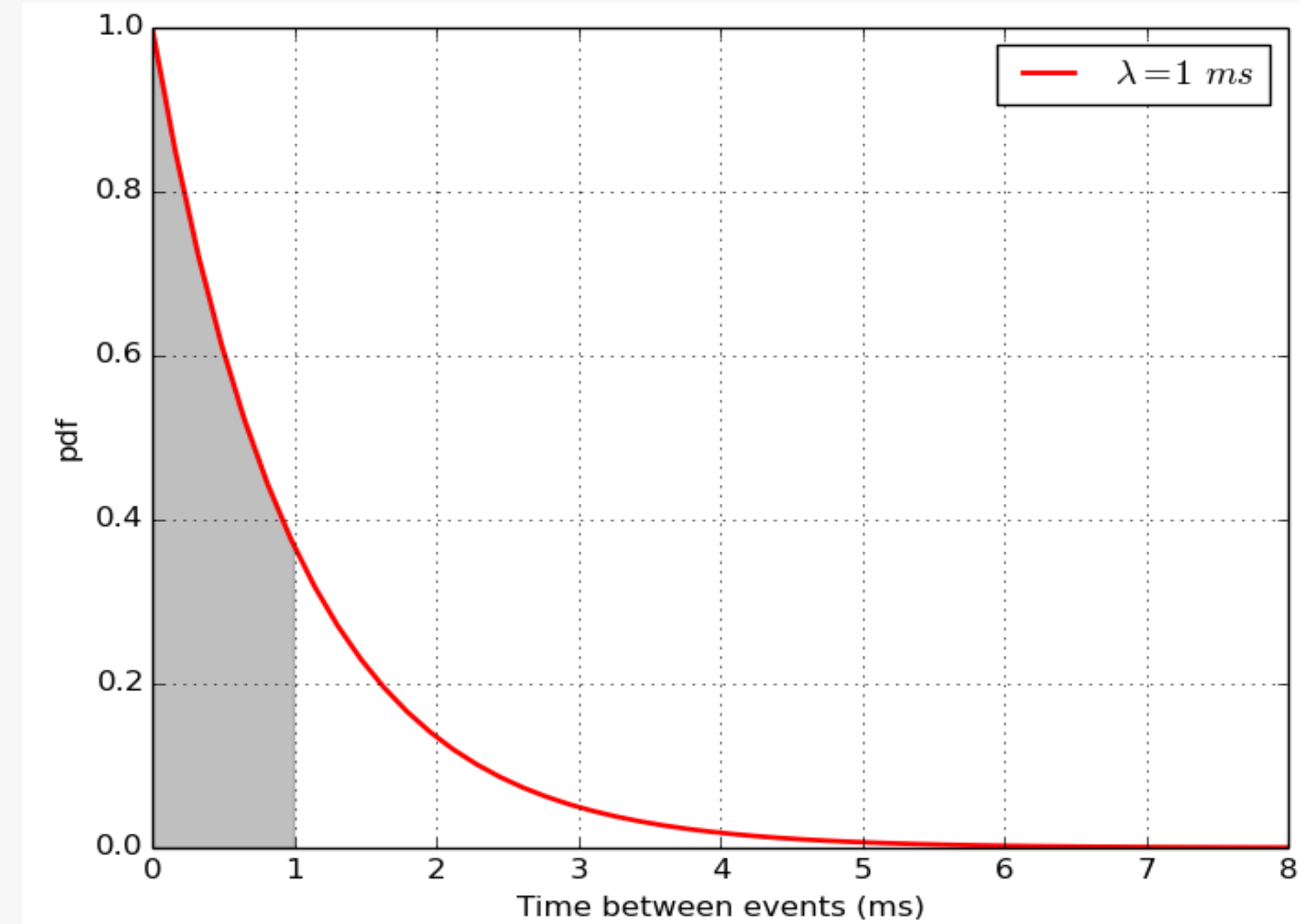
So the busy mechanism protects our electronics from unwanted triggers

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

Which (average) DAQ rate can we achieve now?

- How much we lose with the busy logic?

Reminder: with a clock trigger and $\tau = 1$ ms the limit was 1 kHz



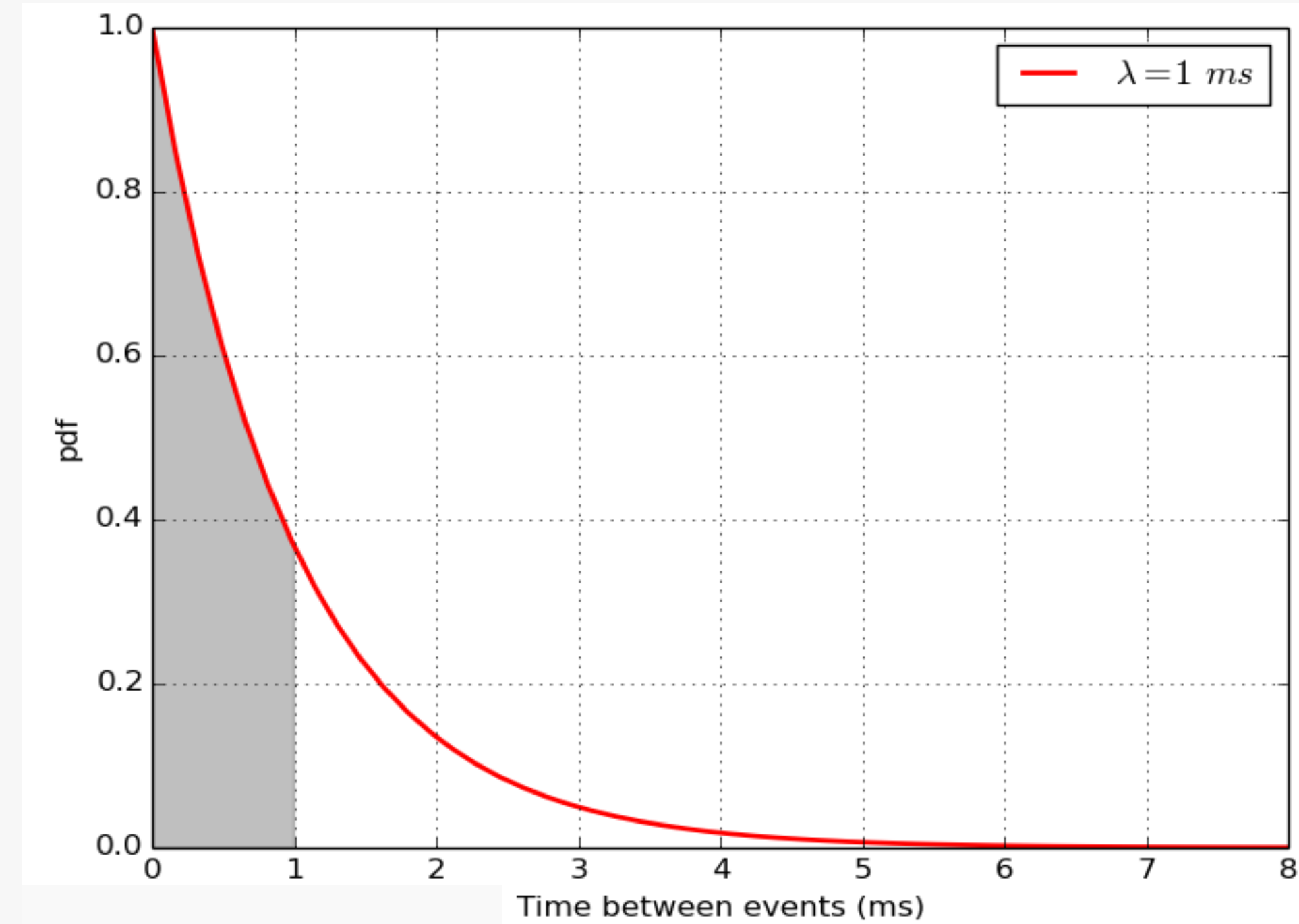
Deadtime and efficiency

Definitions

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

Therefore:

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



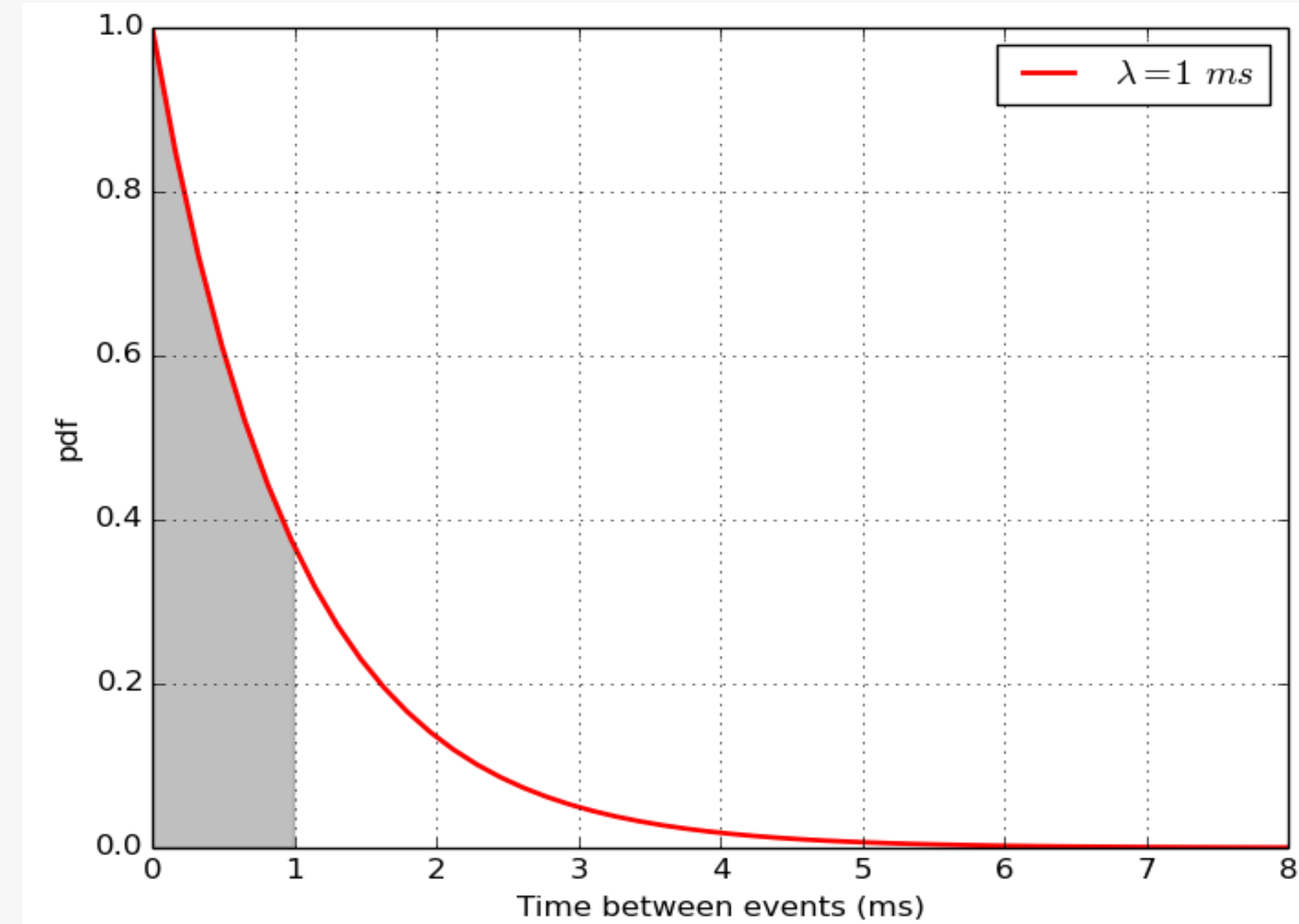
Deadtime and efficiency

Definitions

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

Therefore:

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



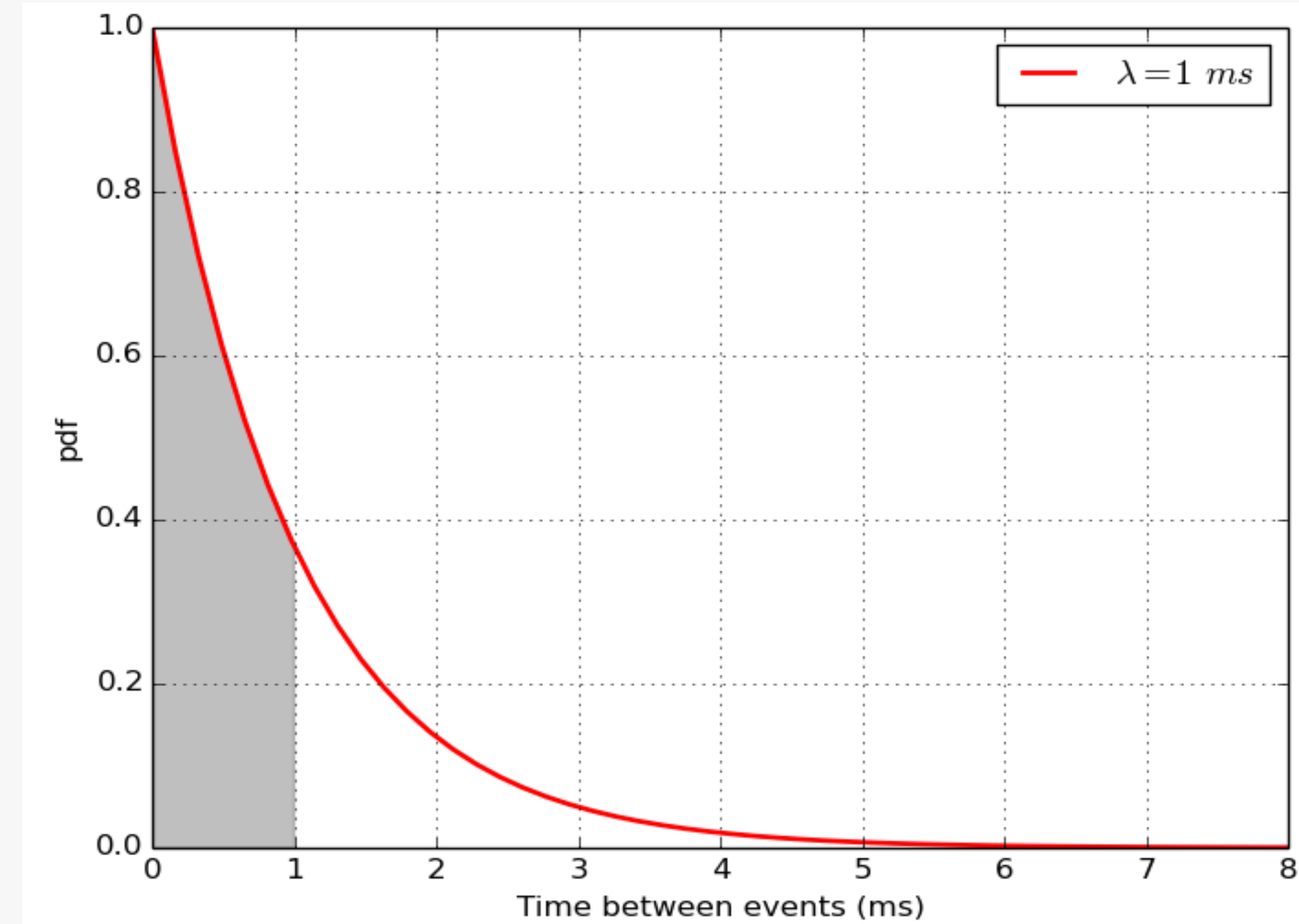
Deadtime and efficiency

Definitions

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

Therefore:

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



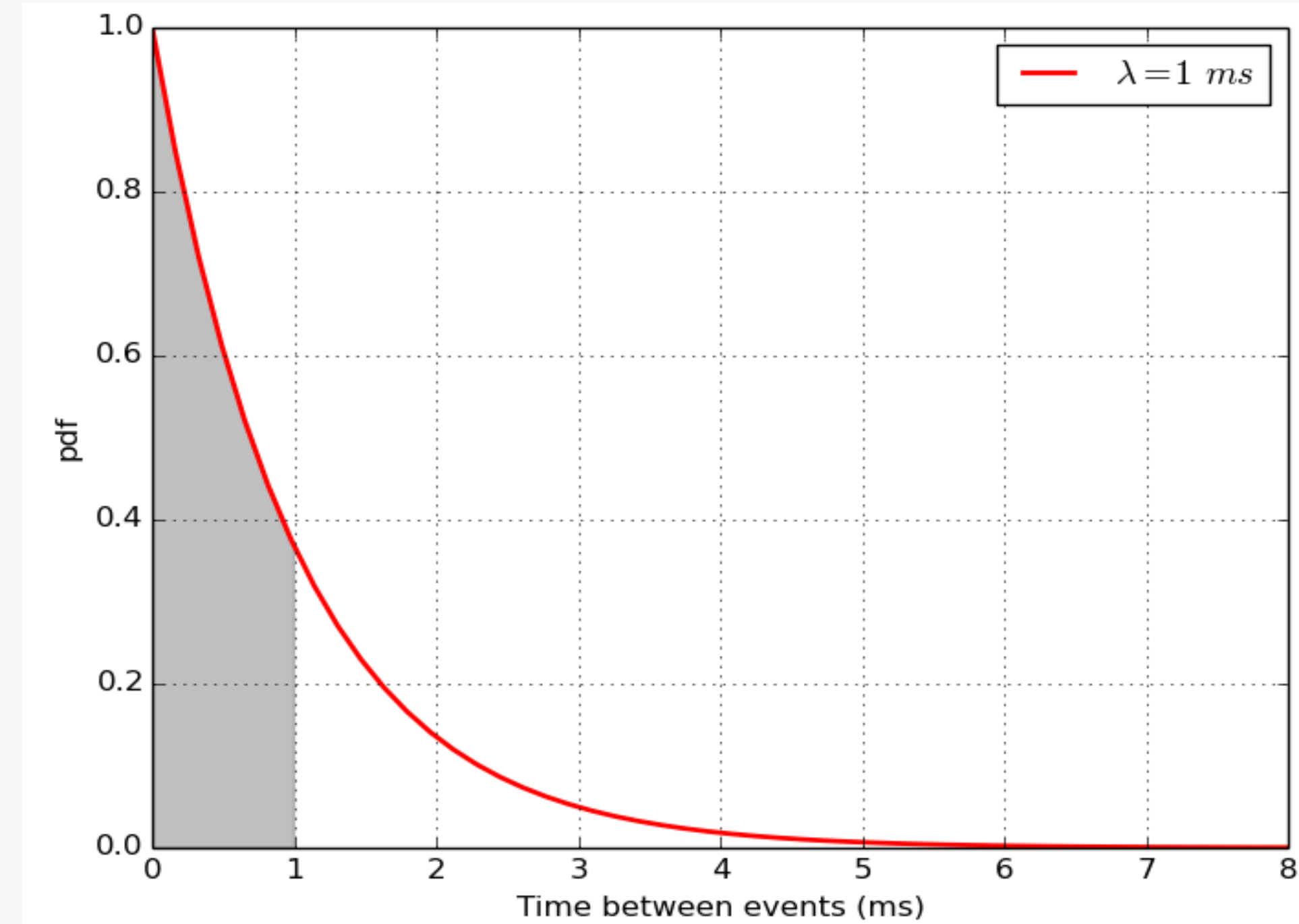
Deadtime and efficiency

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: $P[\text{busy}] = \nu \tau$; $P[\text{free}] = 1 - \nu \tau$

Therefore:

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



Deadtime and efficiency

Due to stochastic fluctuations

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

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

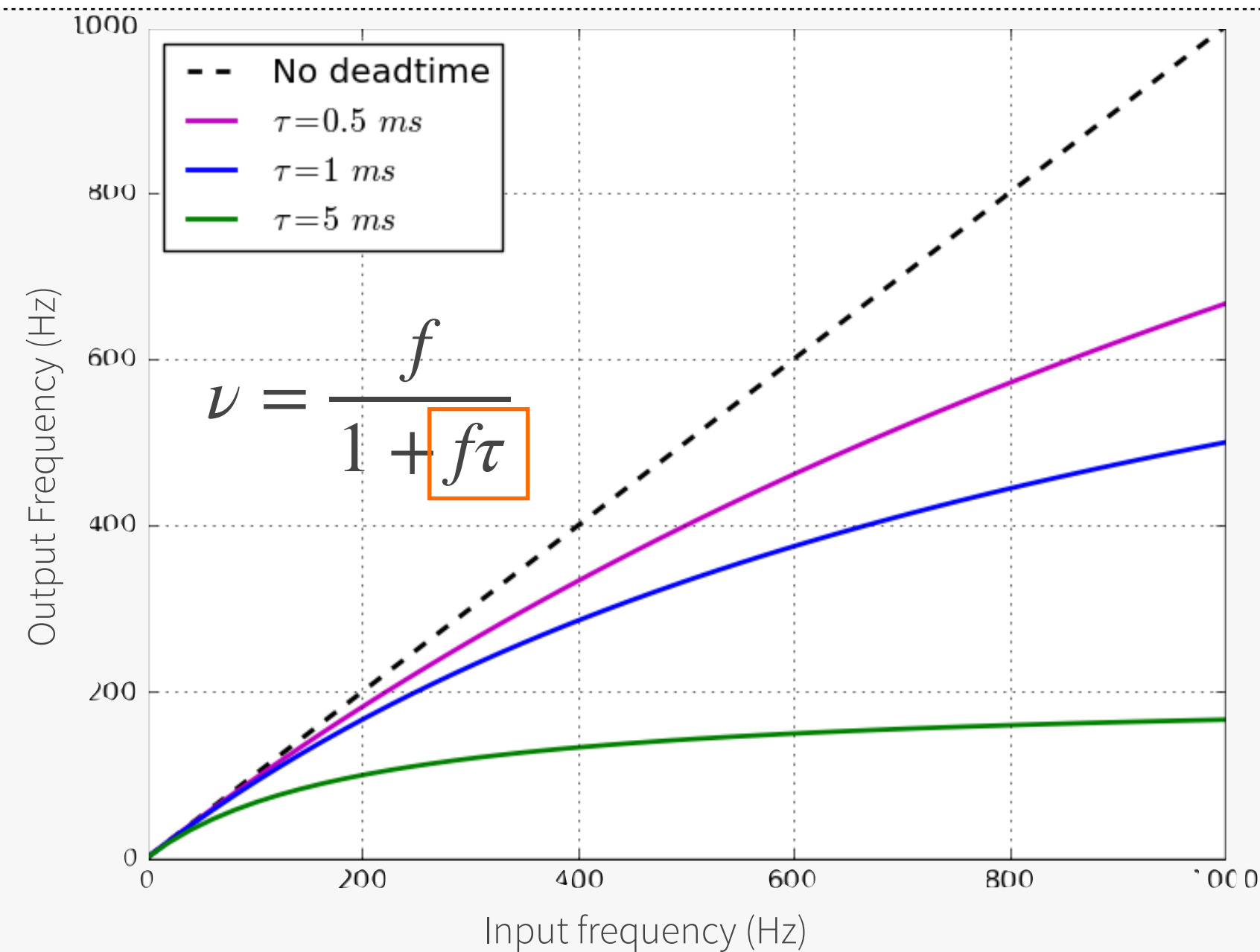
$$\epsilon = \frac{N_{saved}}{N_{tot}} = \frac{1}{1 + f\tau} < 100 \%$$

So, in our specific example

- Physics rate 1 kHz
- Deadtime 1 ms

$$\begin{array}{l} f = 1 \text{ kHz} \\ \tau = 1 \text{ ms} \end{array} \quad \longrightarrow \quad \begin{array}{l} \nu = 500 \text{ Hz} \\ \epsilon = 50 \% \end{array}$$

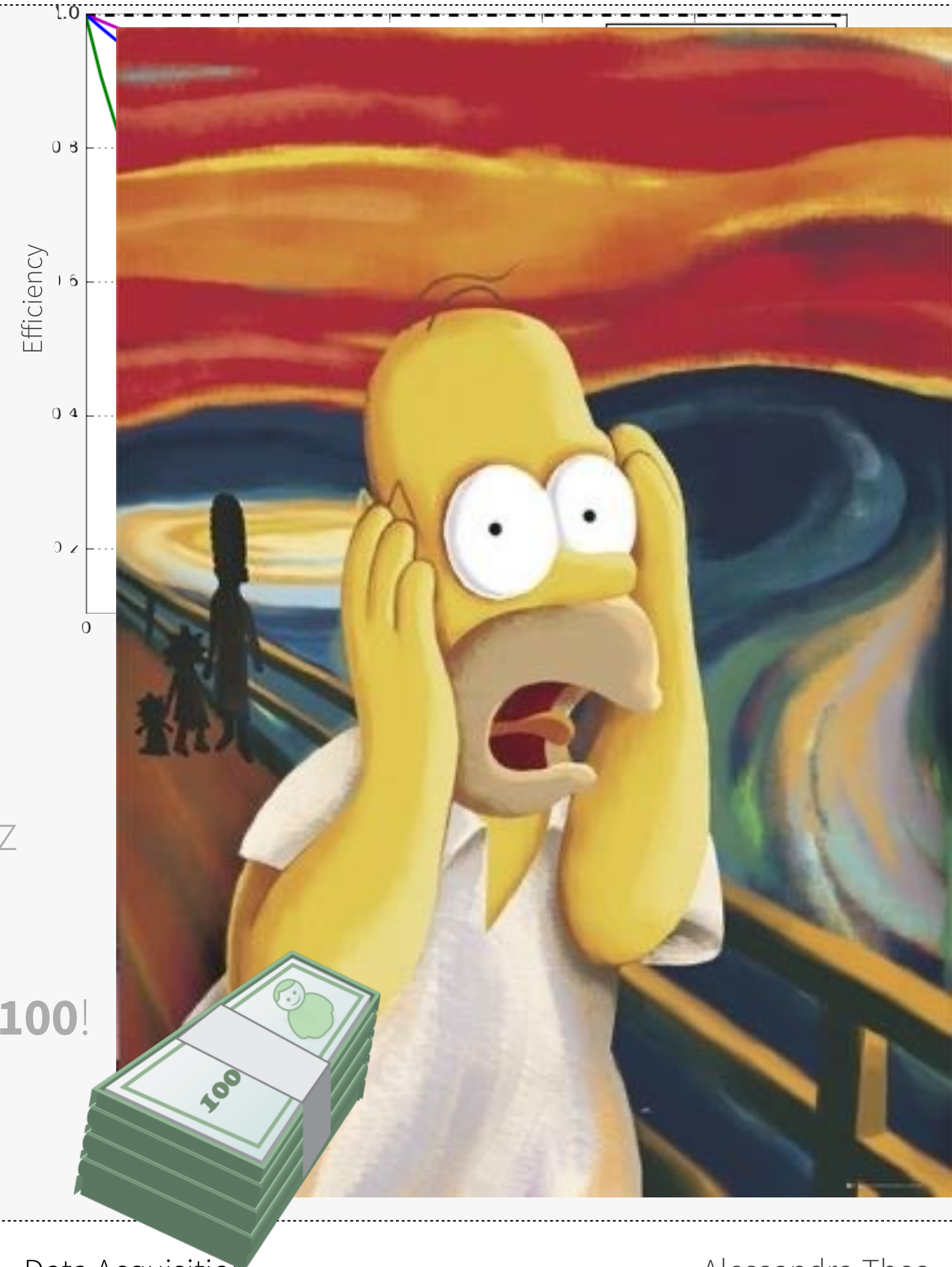
Deadtime and efficiency



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

- E.g.: $\epsilon \sim 99\%$ for $f = 1\text{ kHz} \rightarrow \tau < 0.01\text{ ms} \rightarrow 1/\tau > 100\text{ kHz}$
- 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?



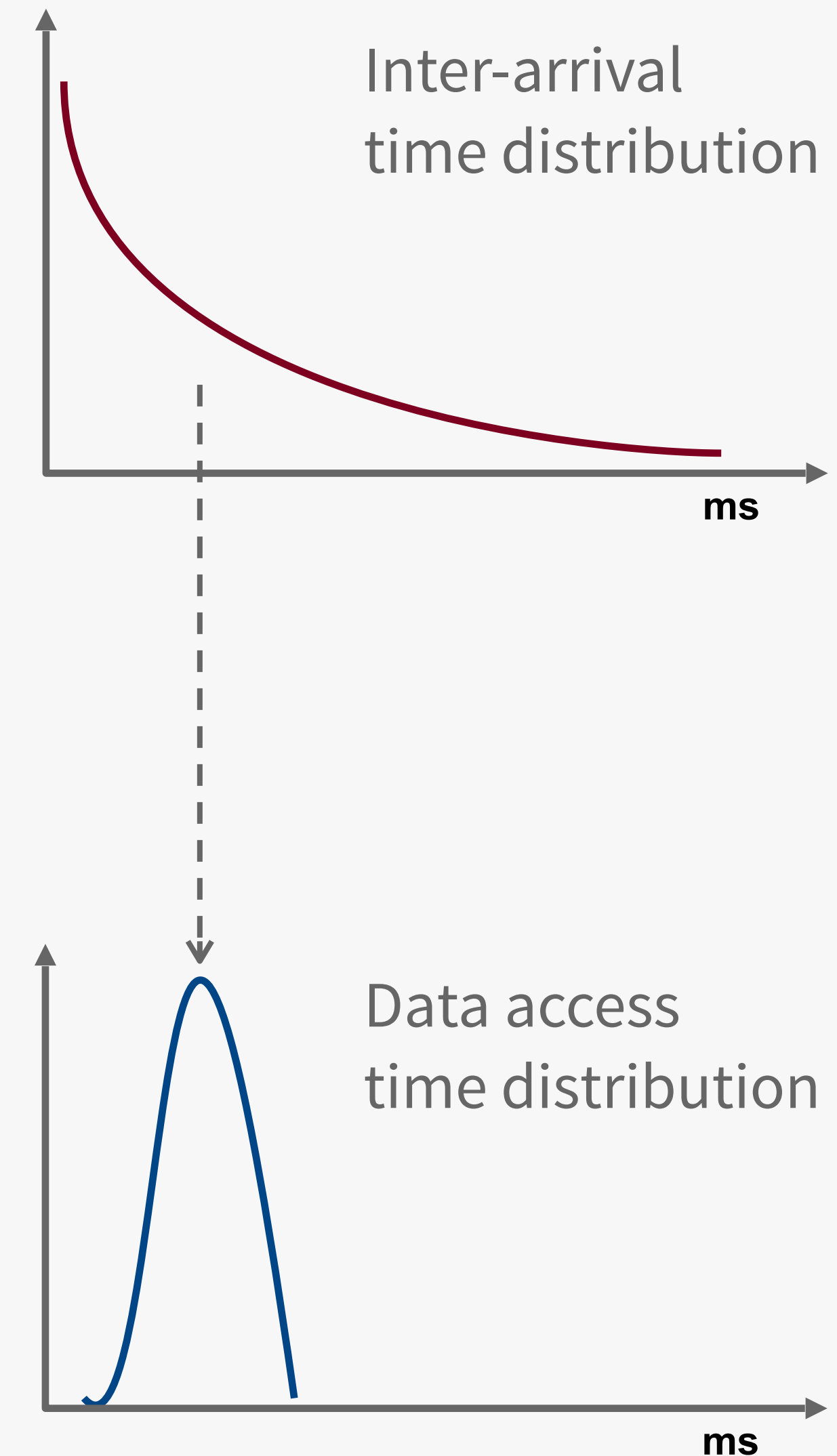
De-randomization

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 (having a holding queue where it can wait to be processed)



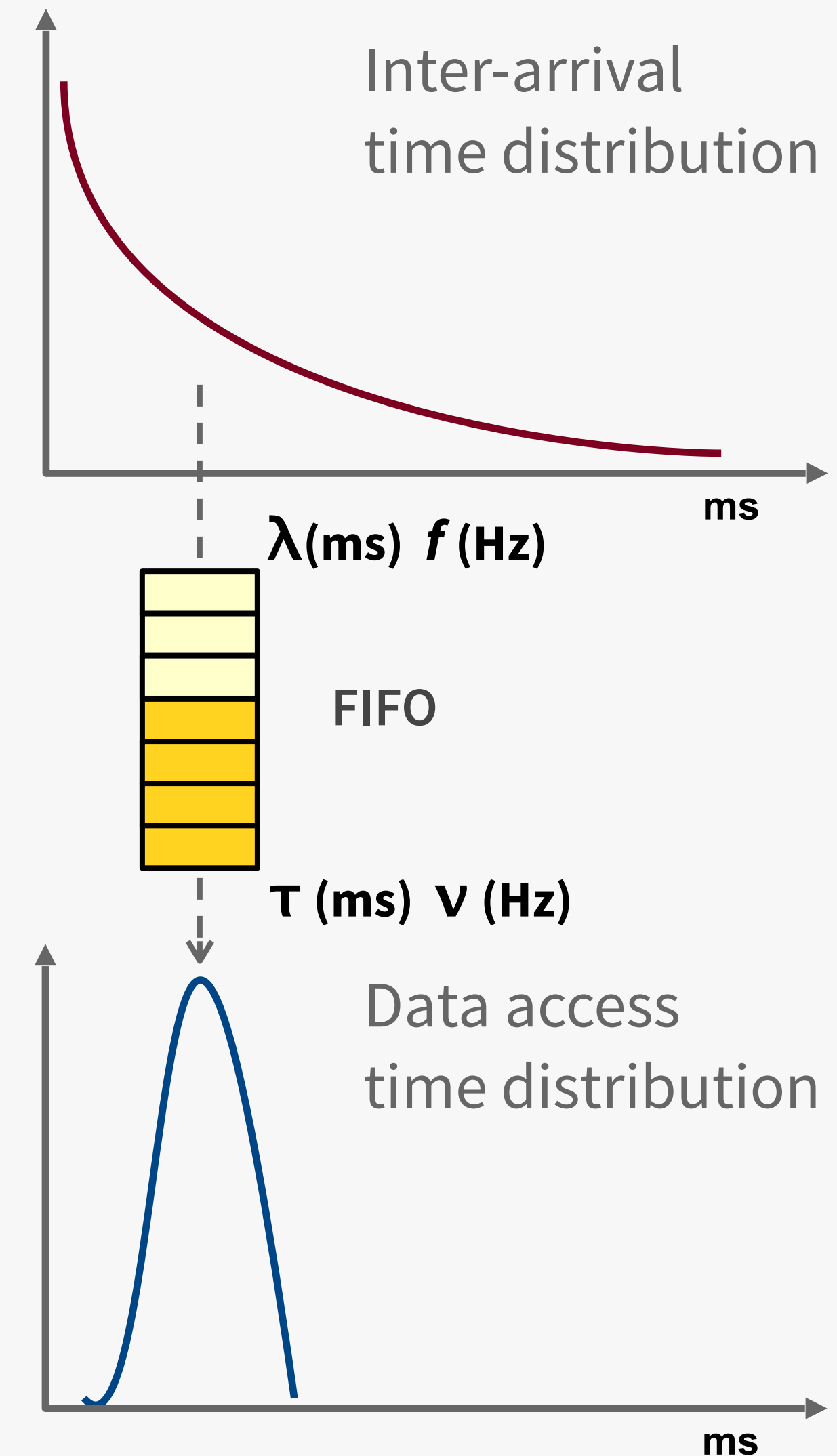
De-randomization

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

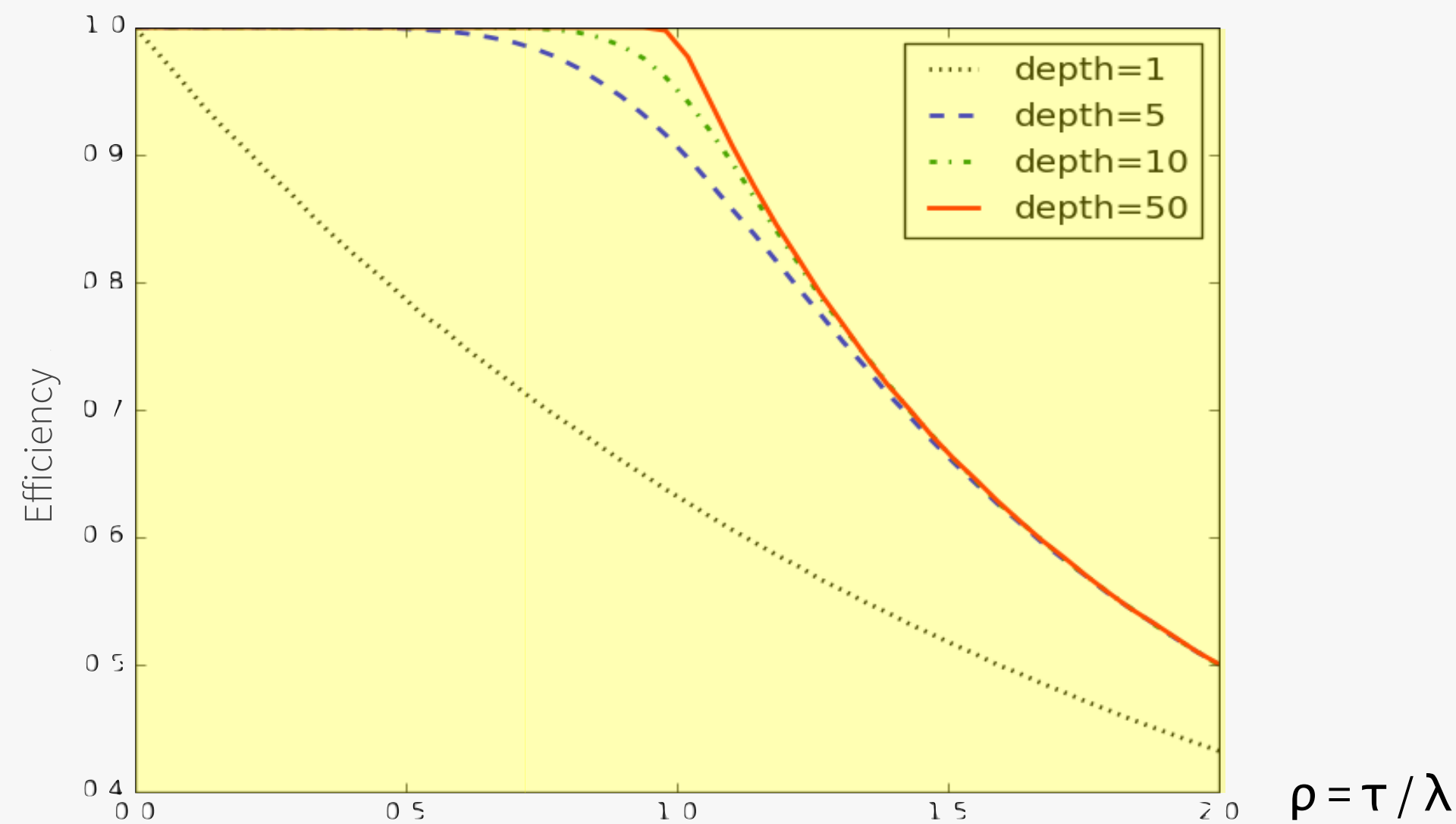
- Then we could ensure that events don't arrive when the system is busy
- This is called **de-randomization**

How it can be achieved?

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



Queuing theory

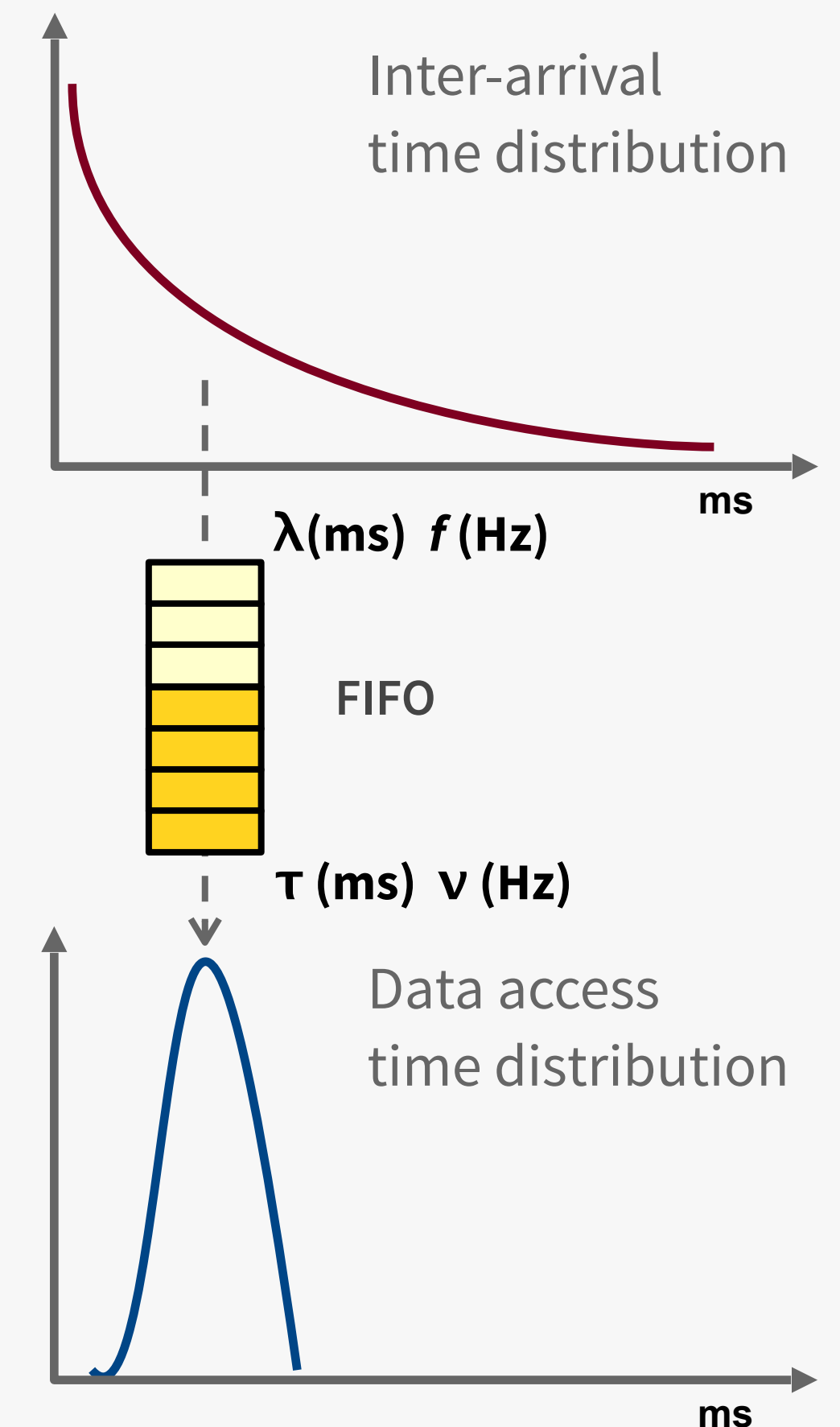


Efficiency vs traffic intensity ($\rho = \tau / \lambda$) for different queue depths

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

Analytic calculation possible for very simple systems only

- Otherwise MonteCarlo simulation is required



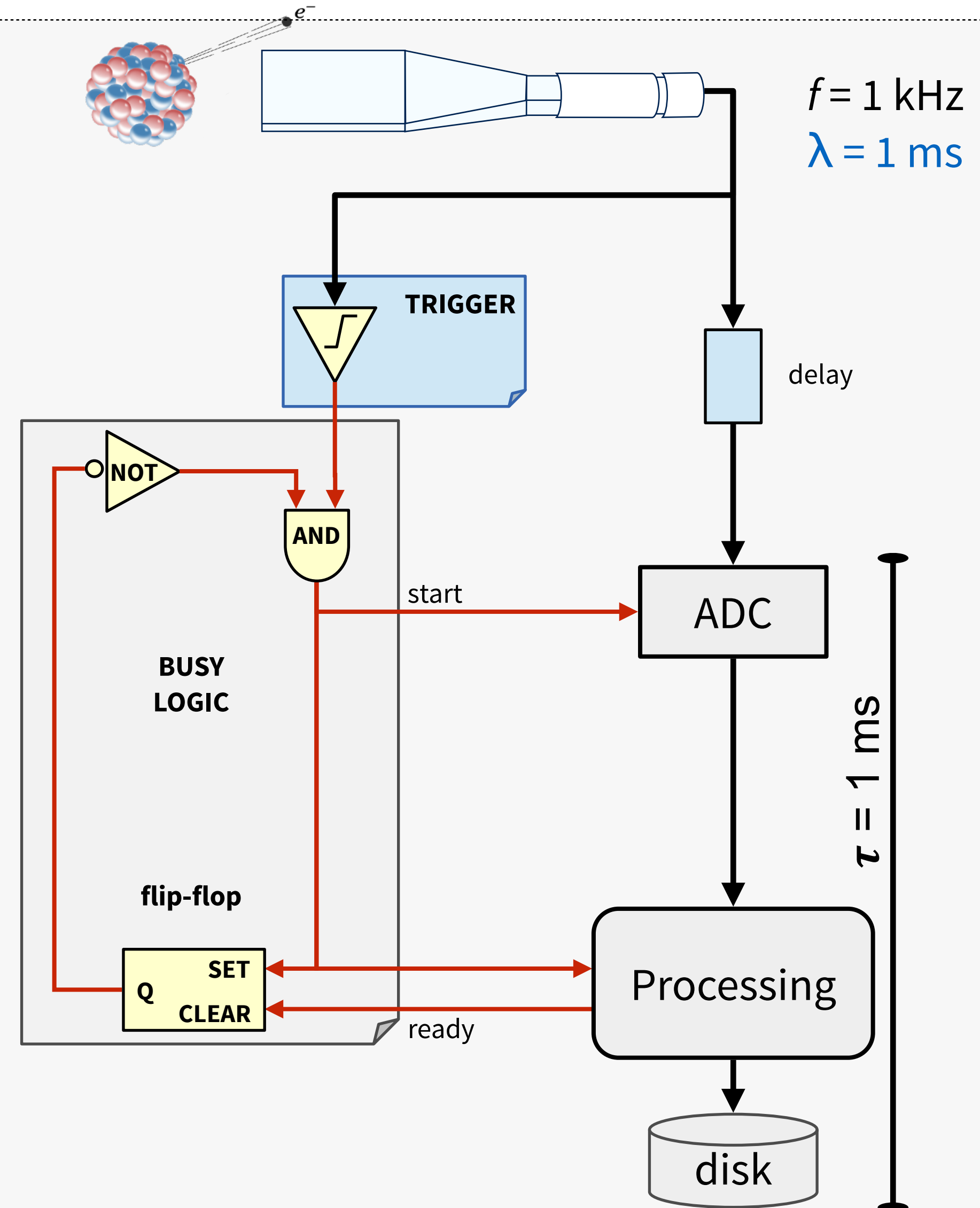
De-randomization

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



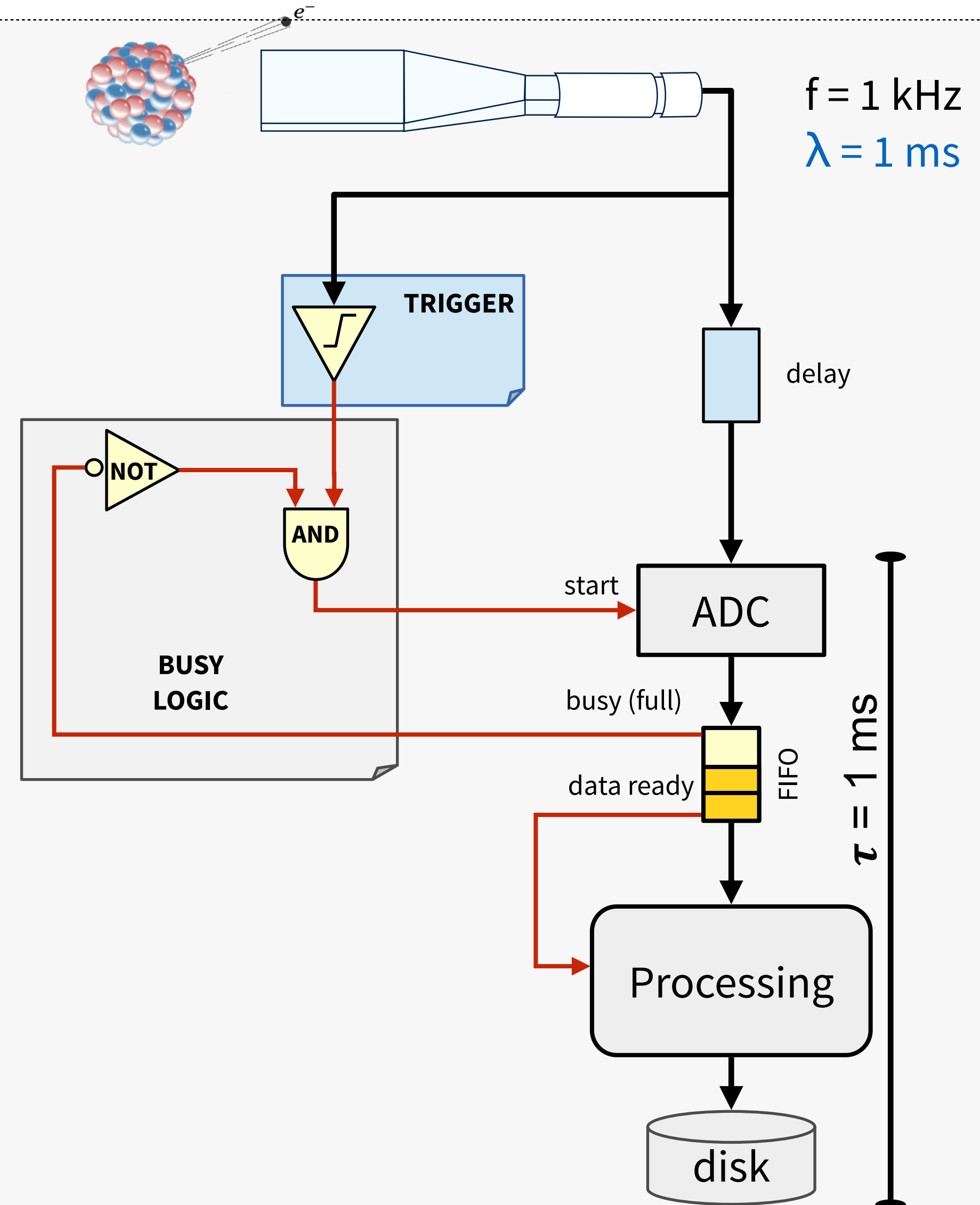
De-randomization

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

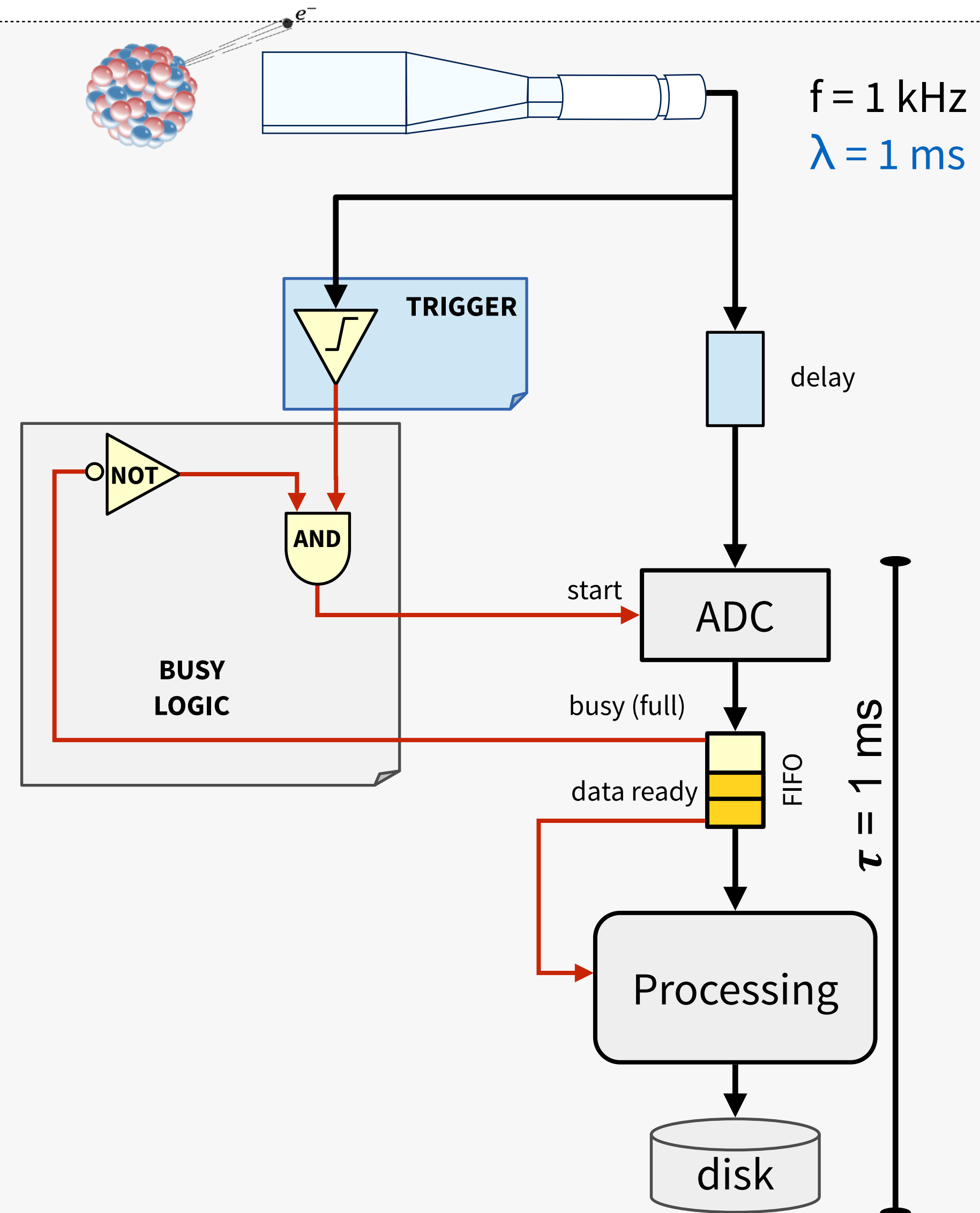
- Minimize the amount of “unnecessary” fast components

~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

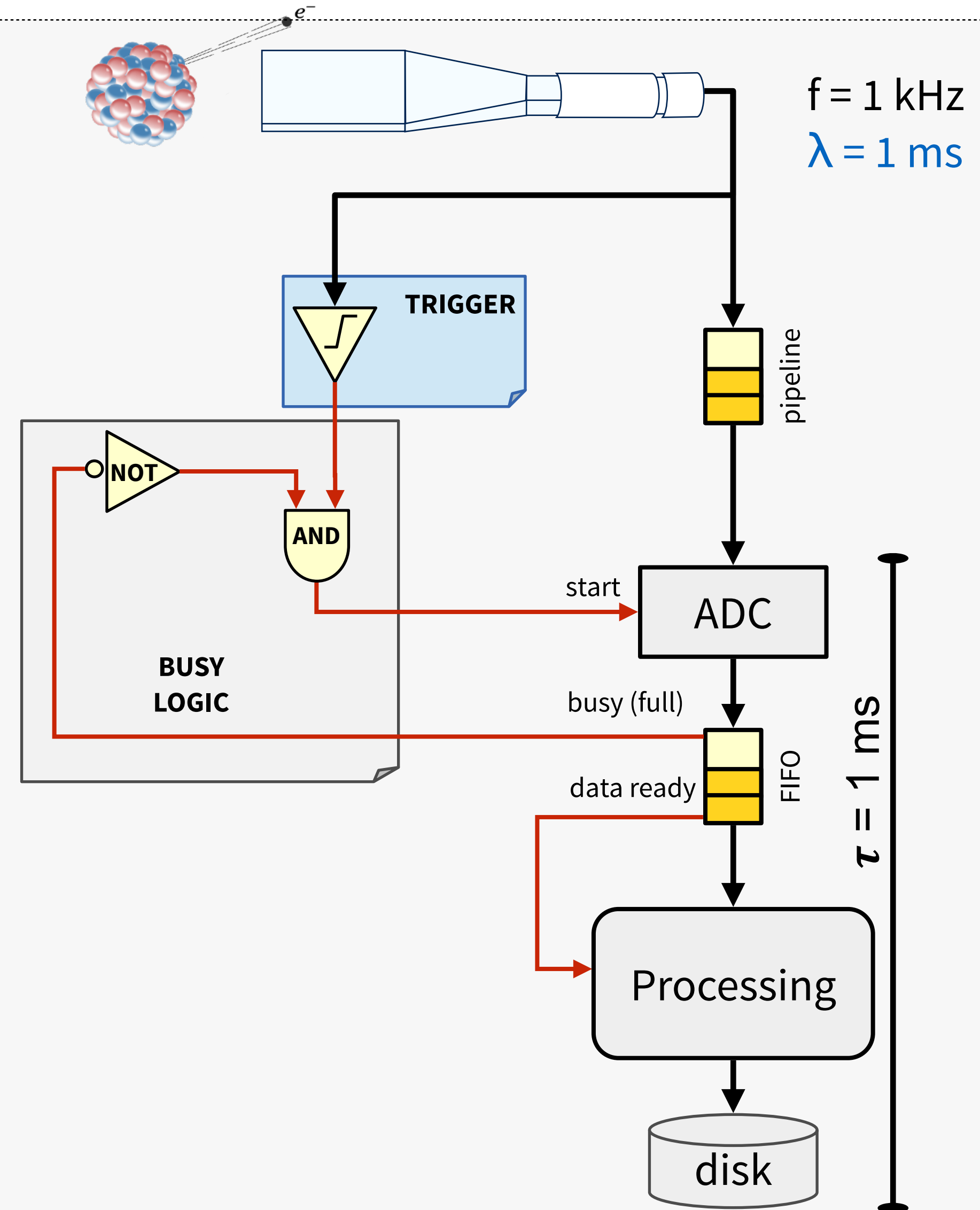
- Minimize the amount of “unnecessary” fast components

~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

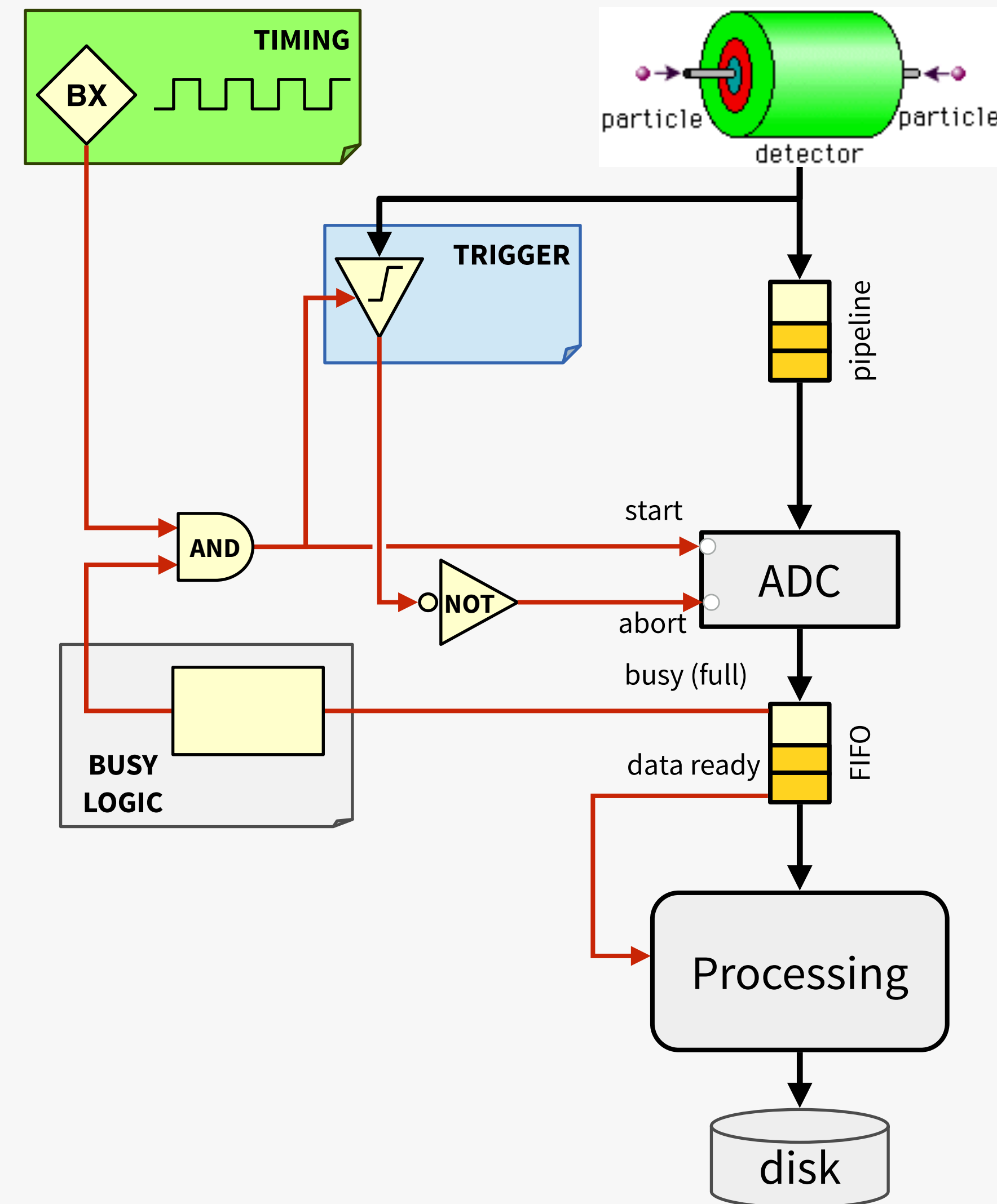
Do we need de-randomization buffers also in collider setups?

- 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



Outline

1. Introduction

1.1. What is DAQ?

1.2. Overall framework

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

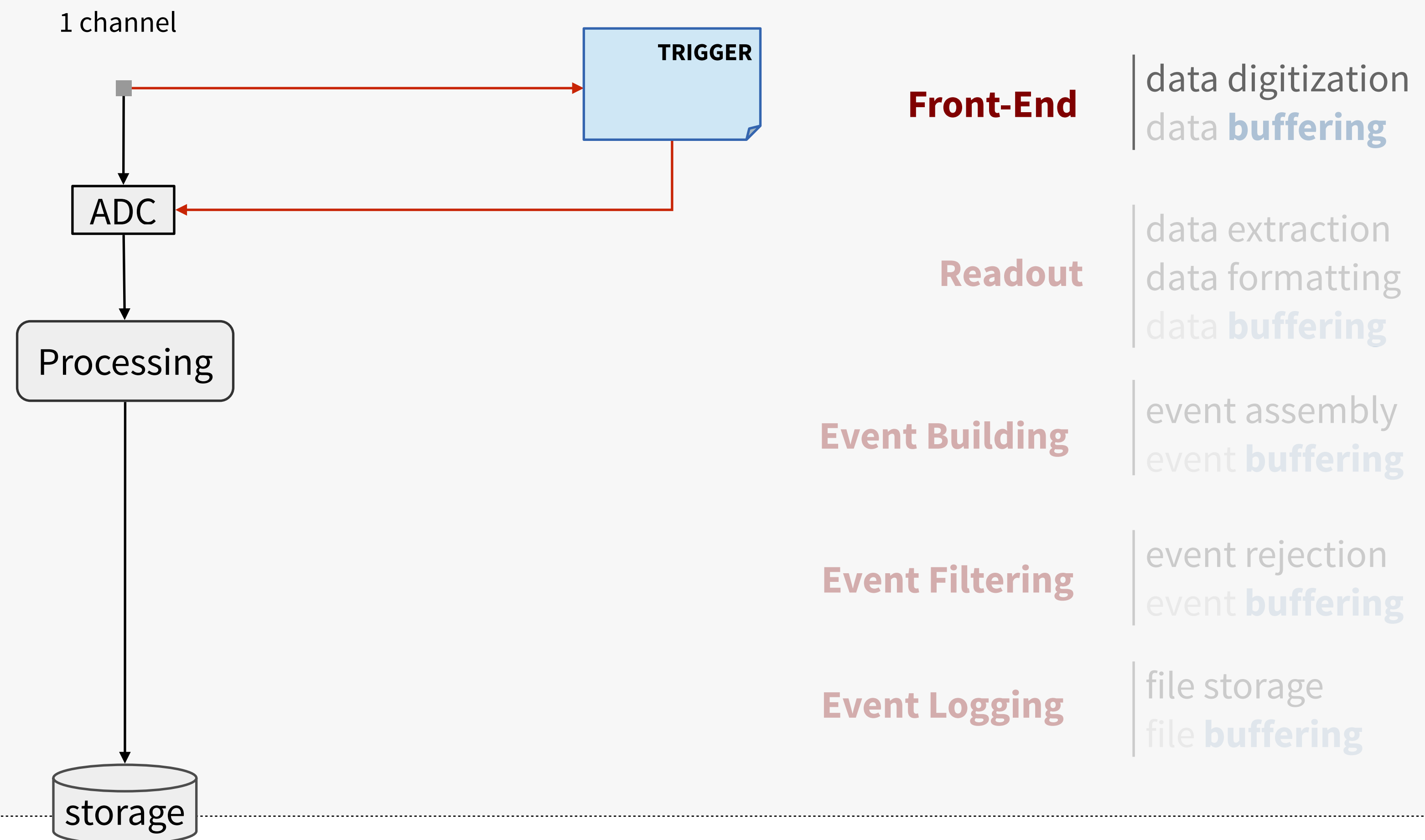
3.2. Buses vs Network

4. Data encoding



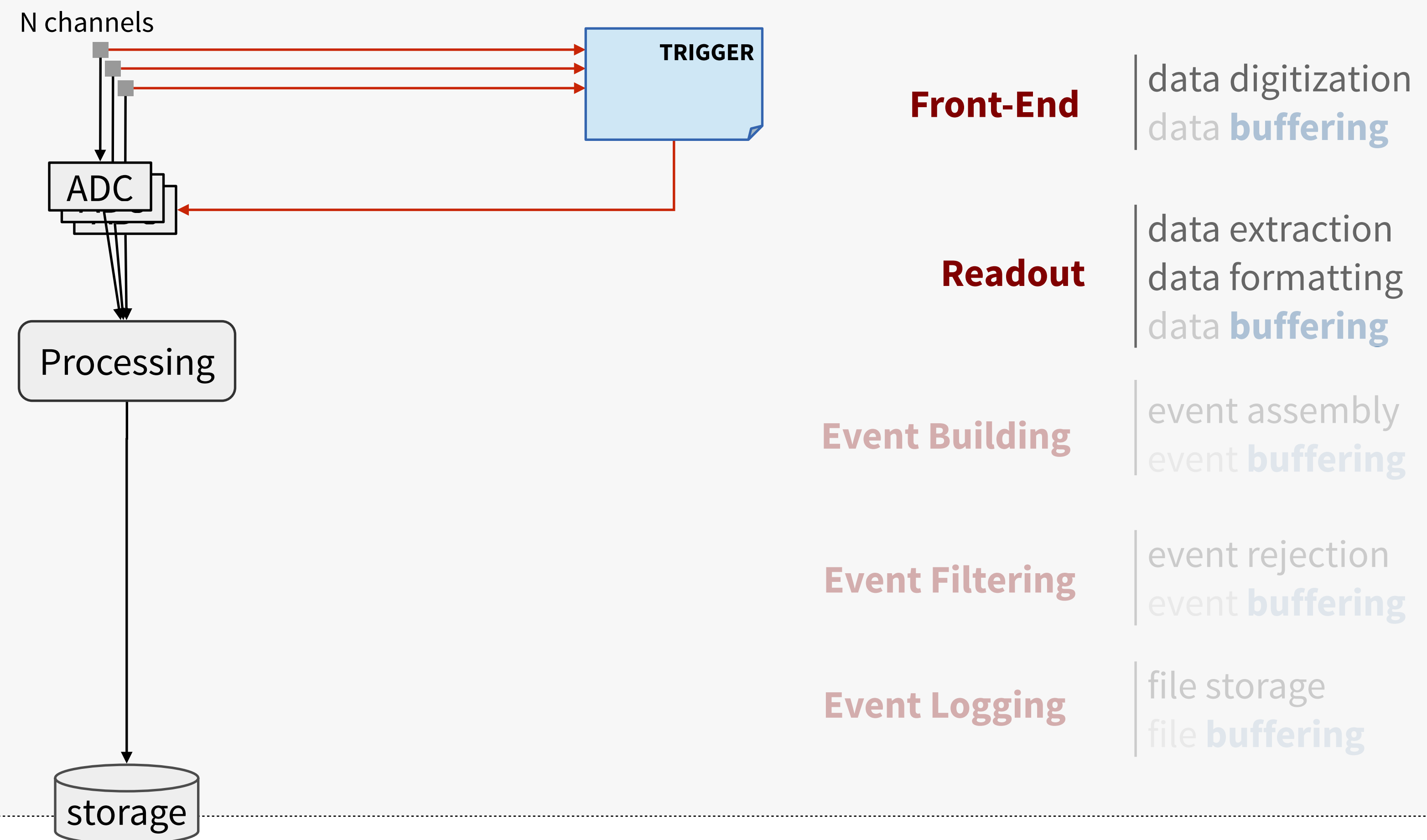
Adding more channels

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



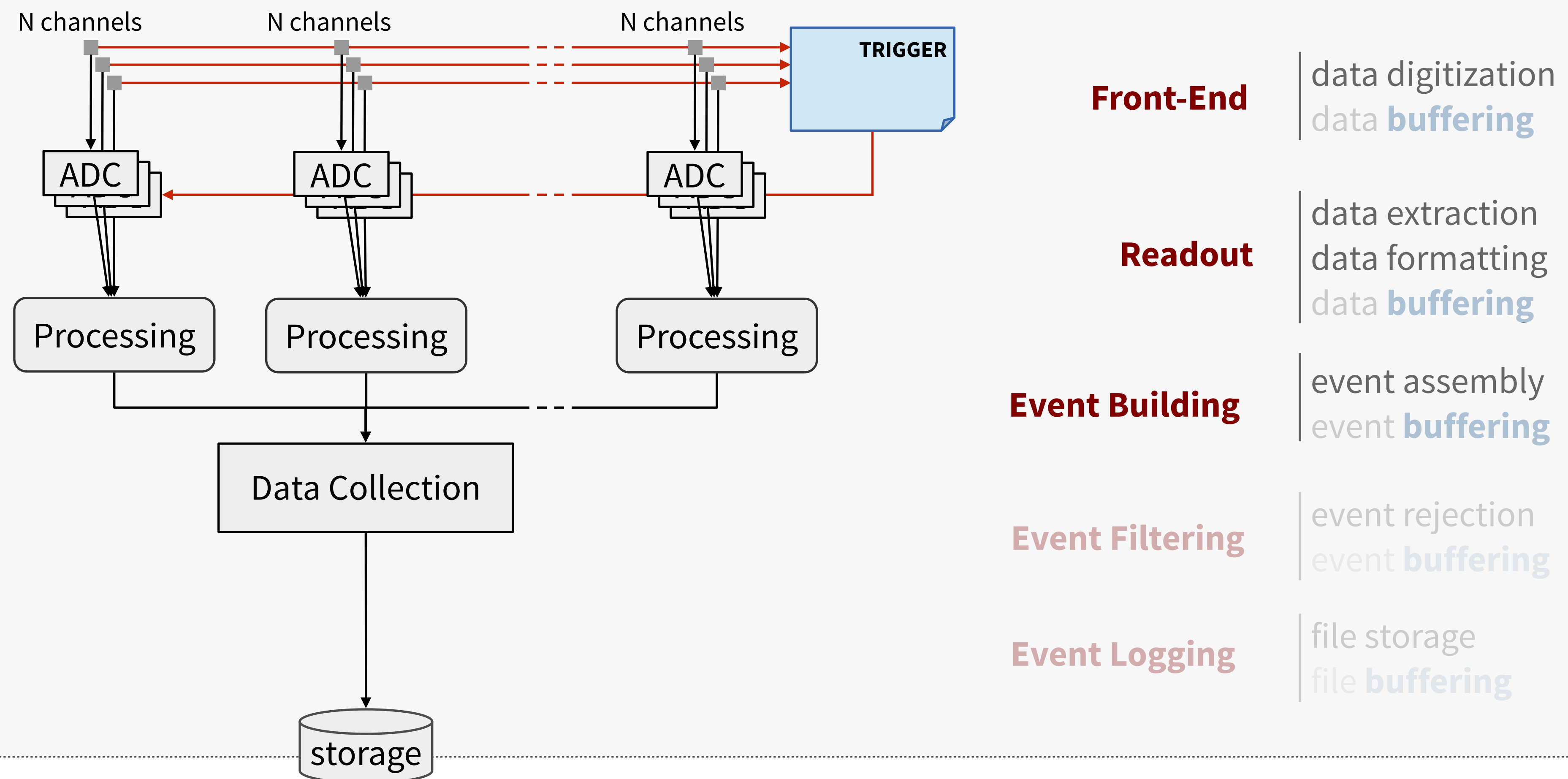
Adding more channels

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



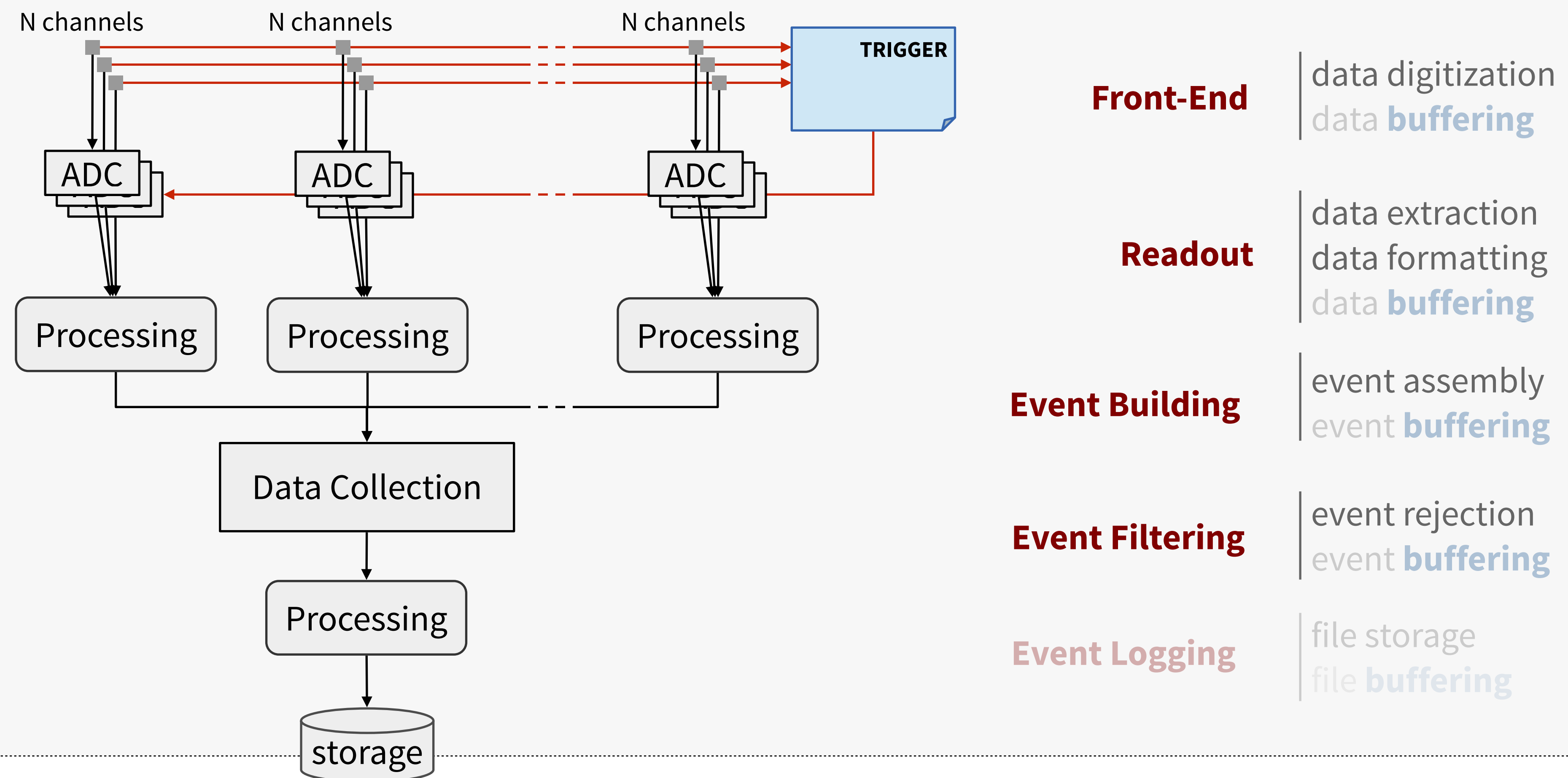
Adding more channels

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



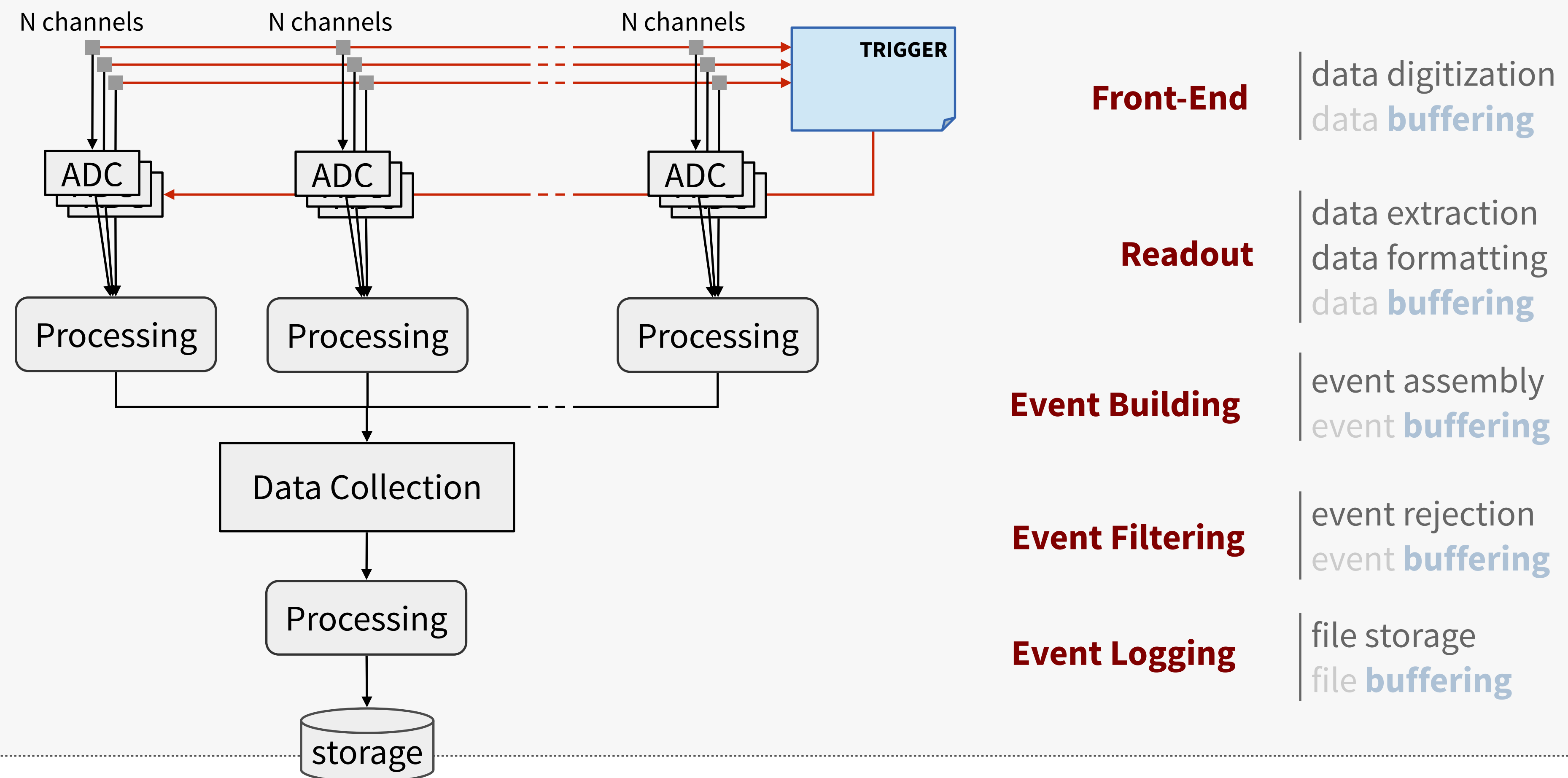
Adding more channels

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



Adding more channels

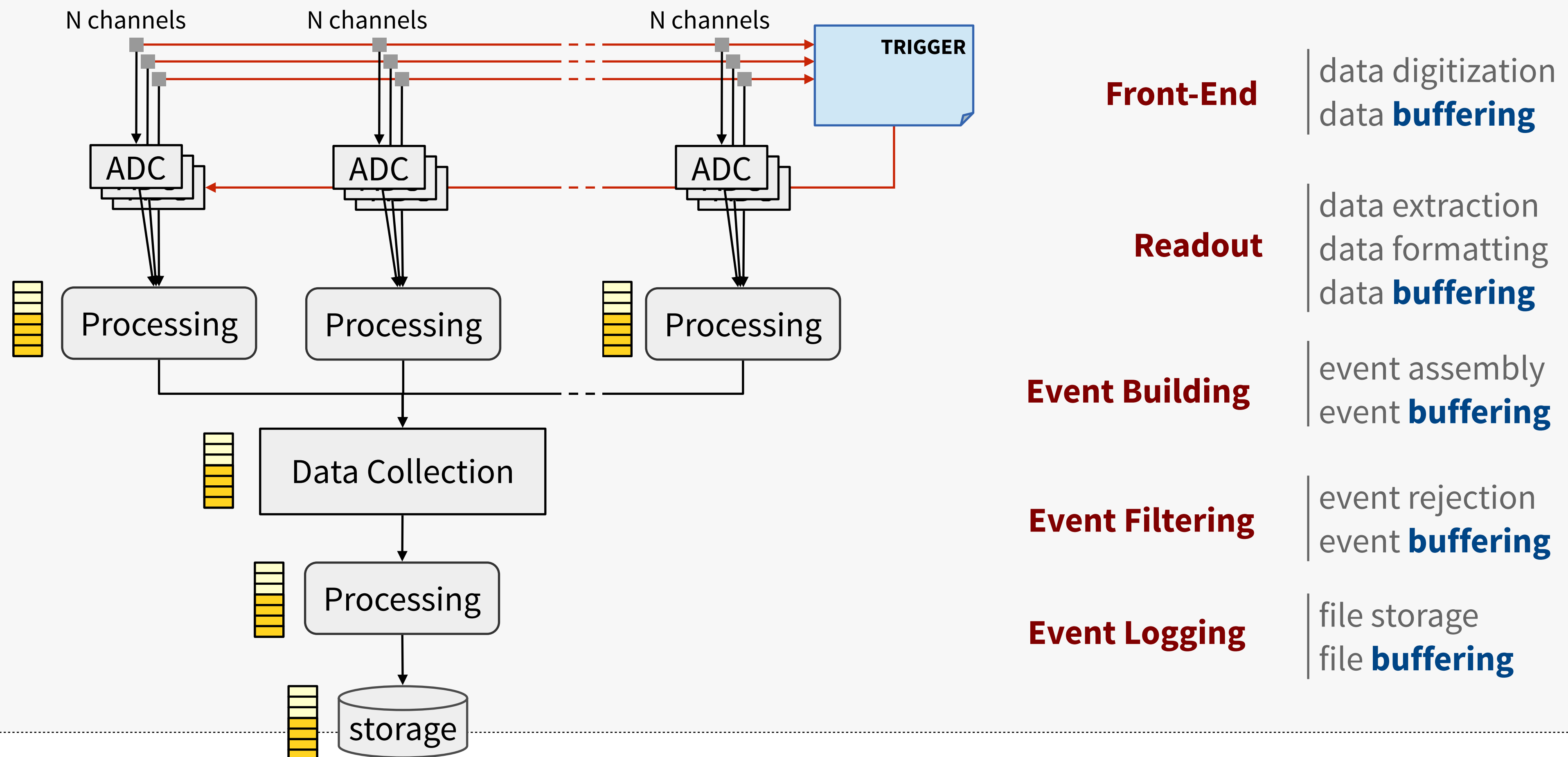
Adding more channels requires a hierarchical structure committed to the data handling and conveyance



Adding more channels

Buffering usually needed at every level

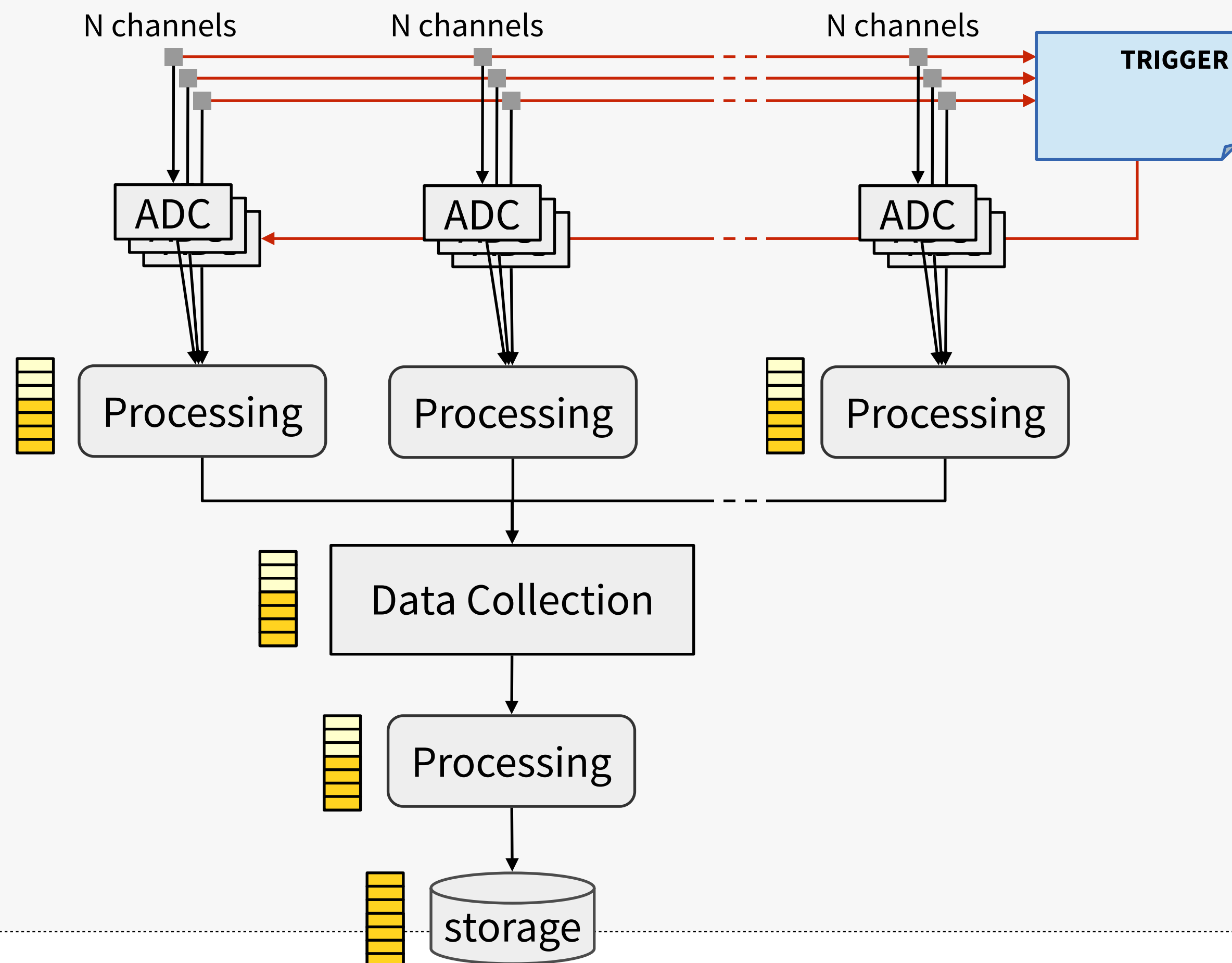
- DAQ can be seen as a multi level buffering system



Backpressure

If a system/buffer gets saturated

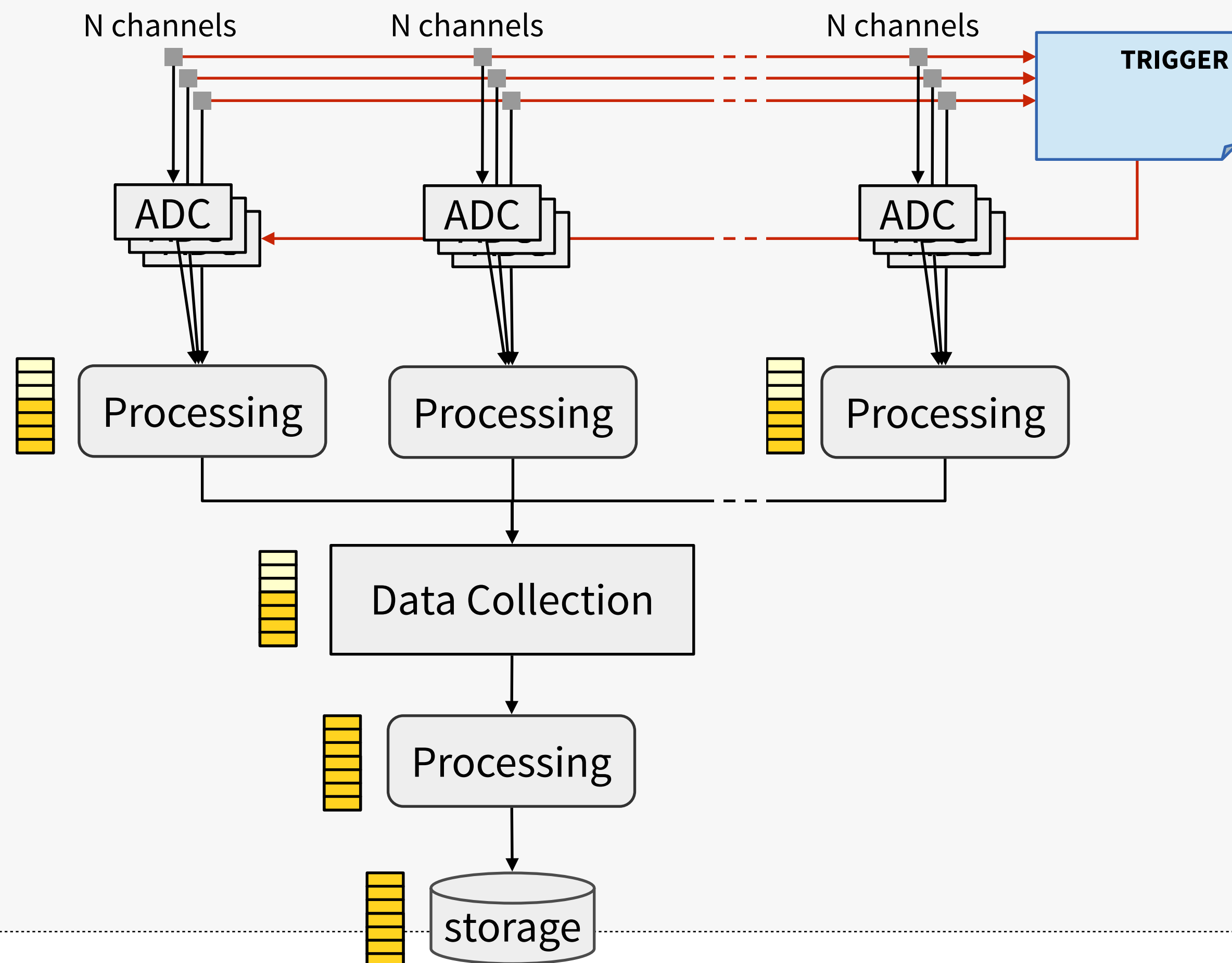
- the “pressure” is propagated upstream (**back-pressure**)



Backpressure

If a system/buffer gets saturated

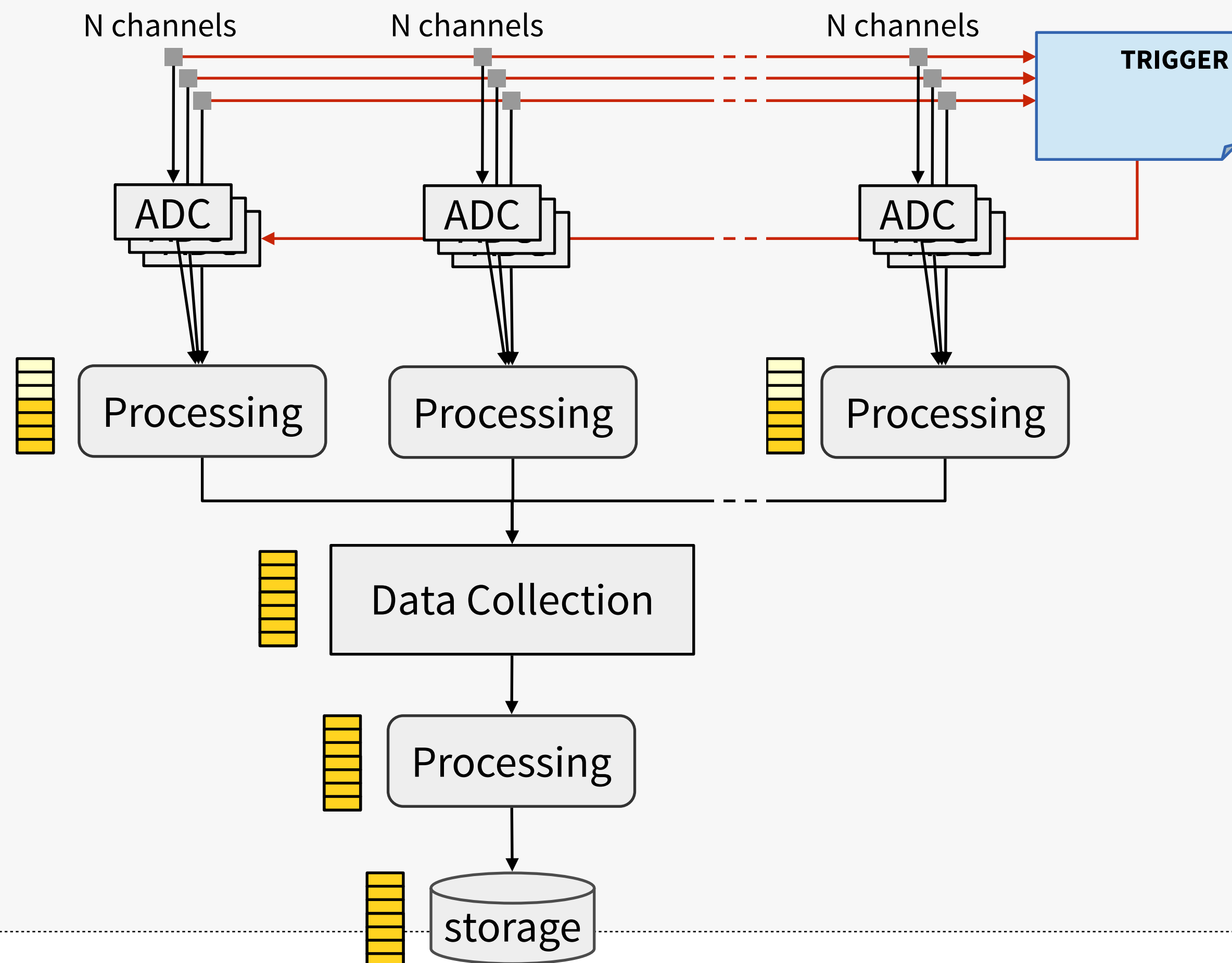
- the “pressure” is propagated upstream (**back-pressure**)



Backpressure

If a system/buffer gets saturated

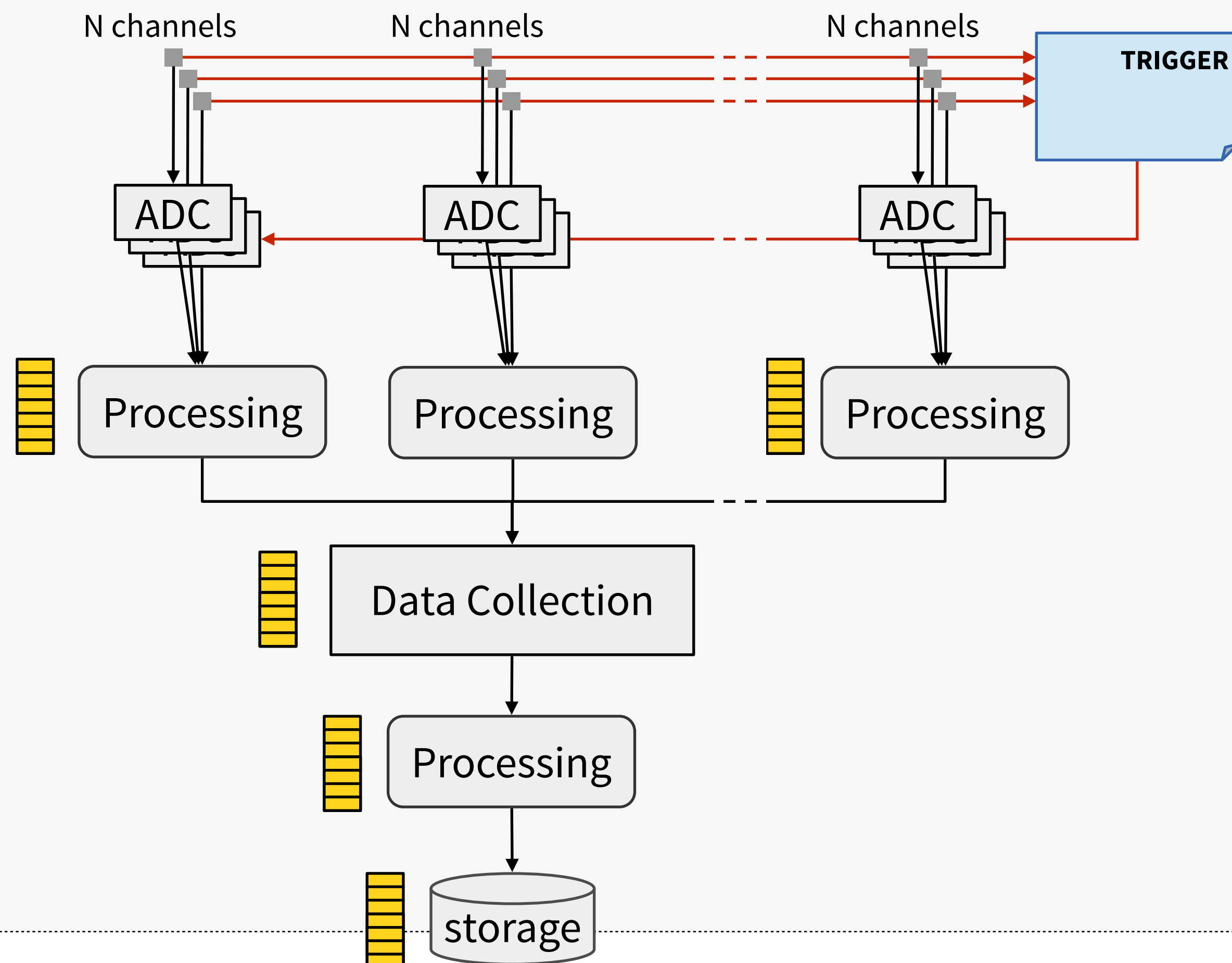
- the “pressure” is propagated upstream (**back-pressure**)



Backpressure

If a system/buffer gets saturated

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

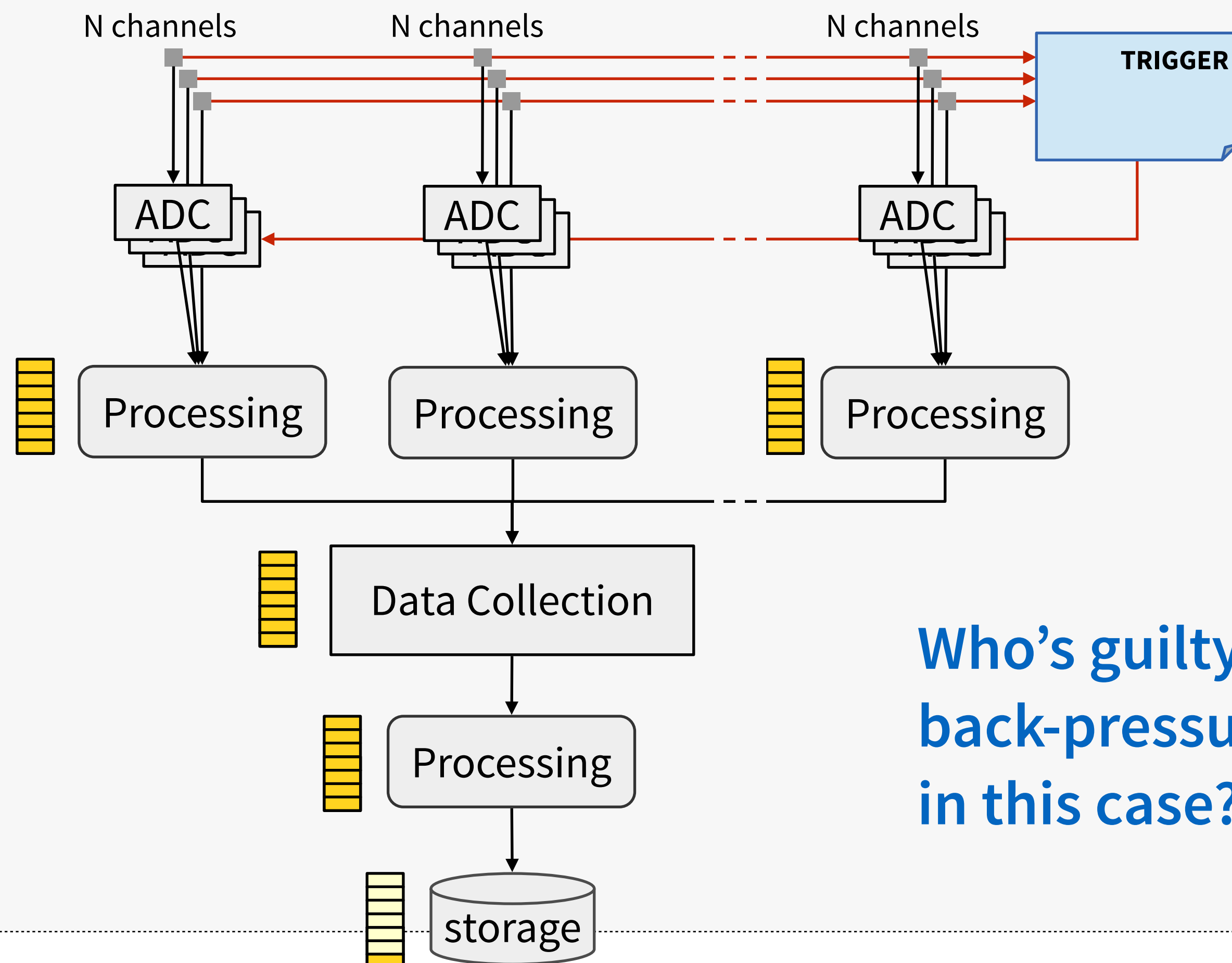


- Up to exert **busy** to the trigger system
- Debugging:** where is the source of back-pressure?
 - follow the buffers occupancy via the monitoring system

Backpressure

If a system/buffer gets saturated

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

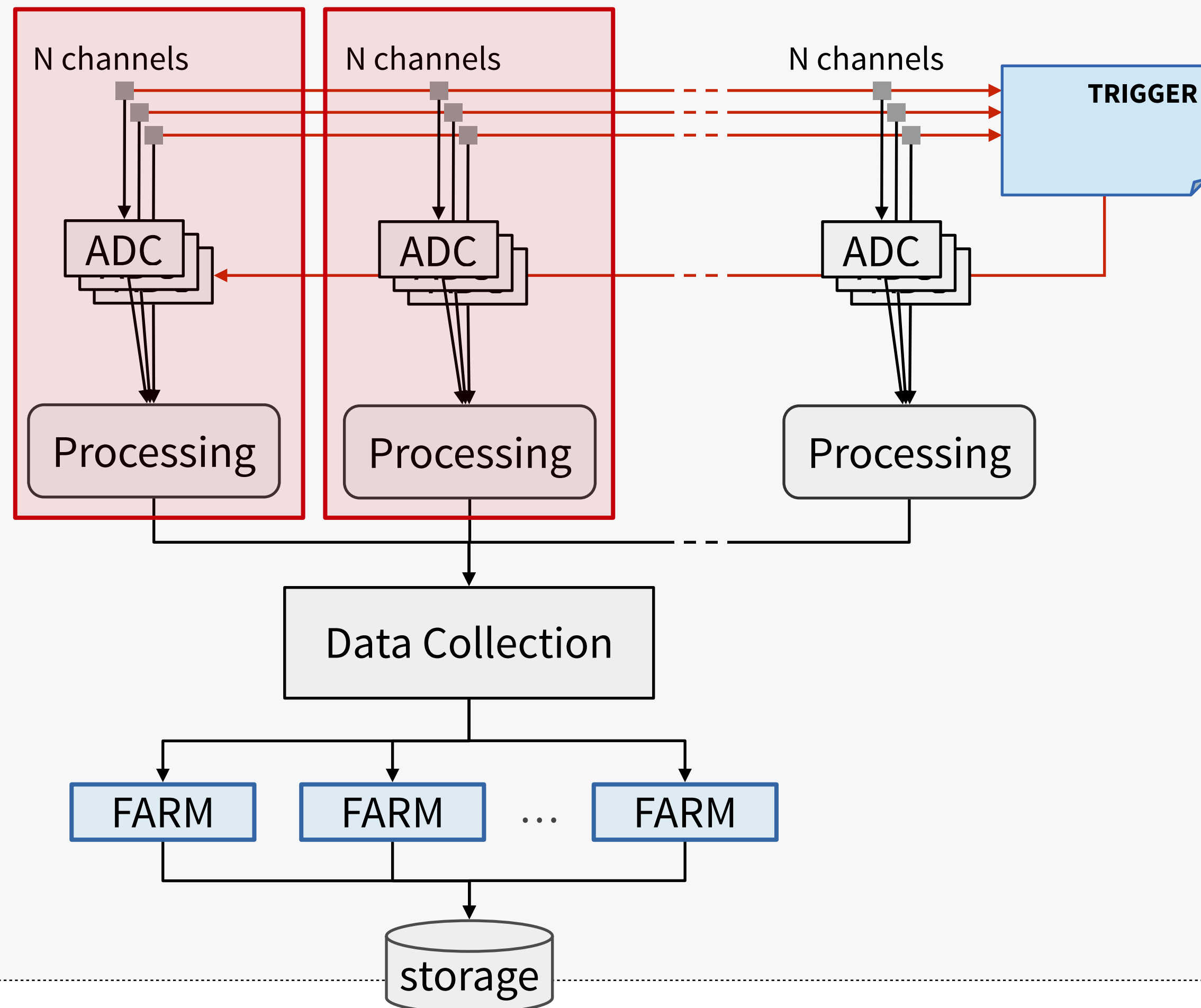


- Up to exert **busy** to the trigger system
- Debugging:** where is the source of back-pressure?
 - follow the buffers occupancy via the monitoring system

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

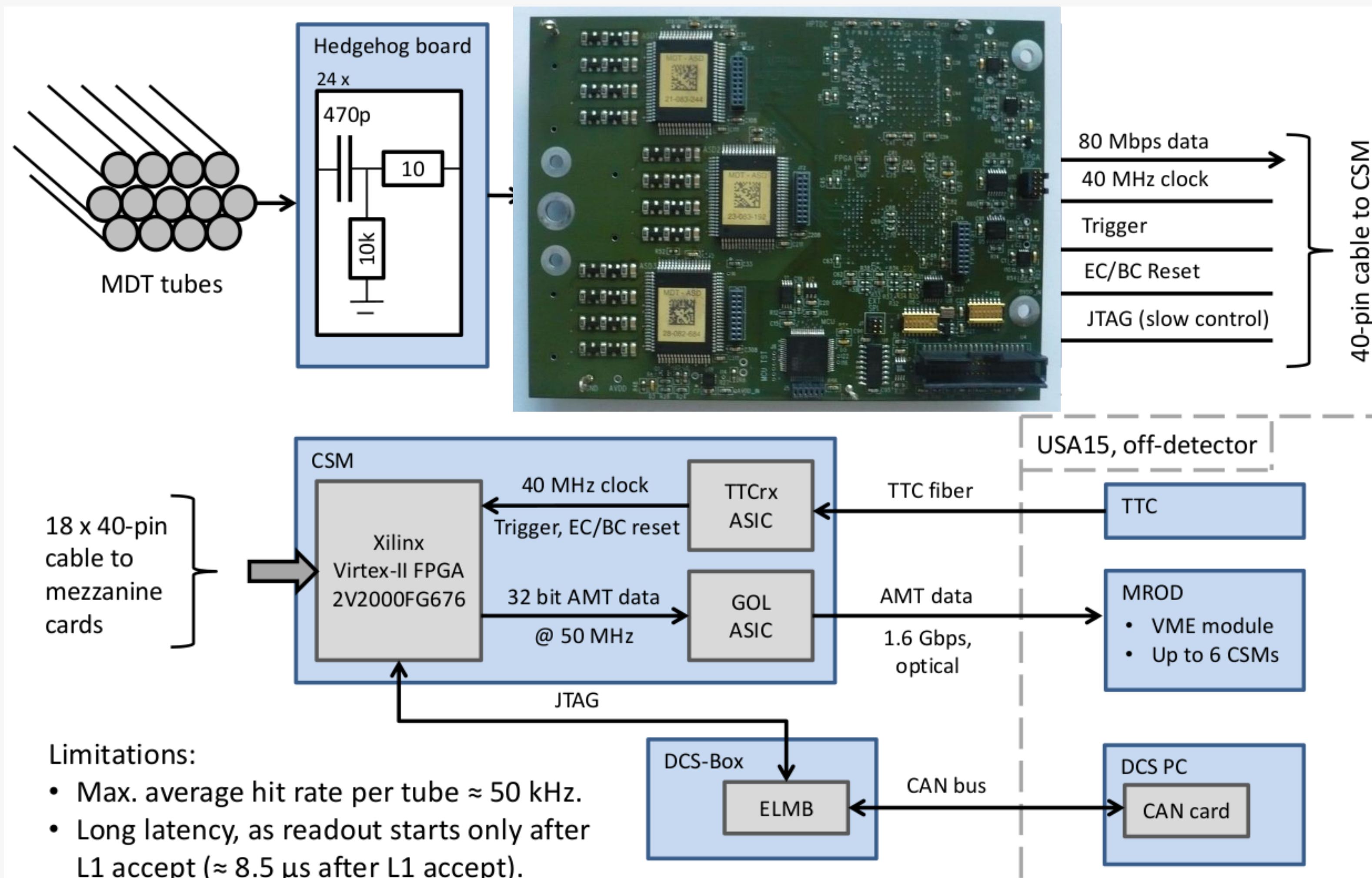
Building blocks

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

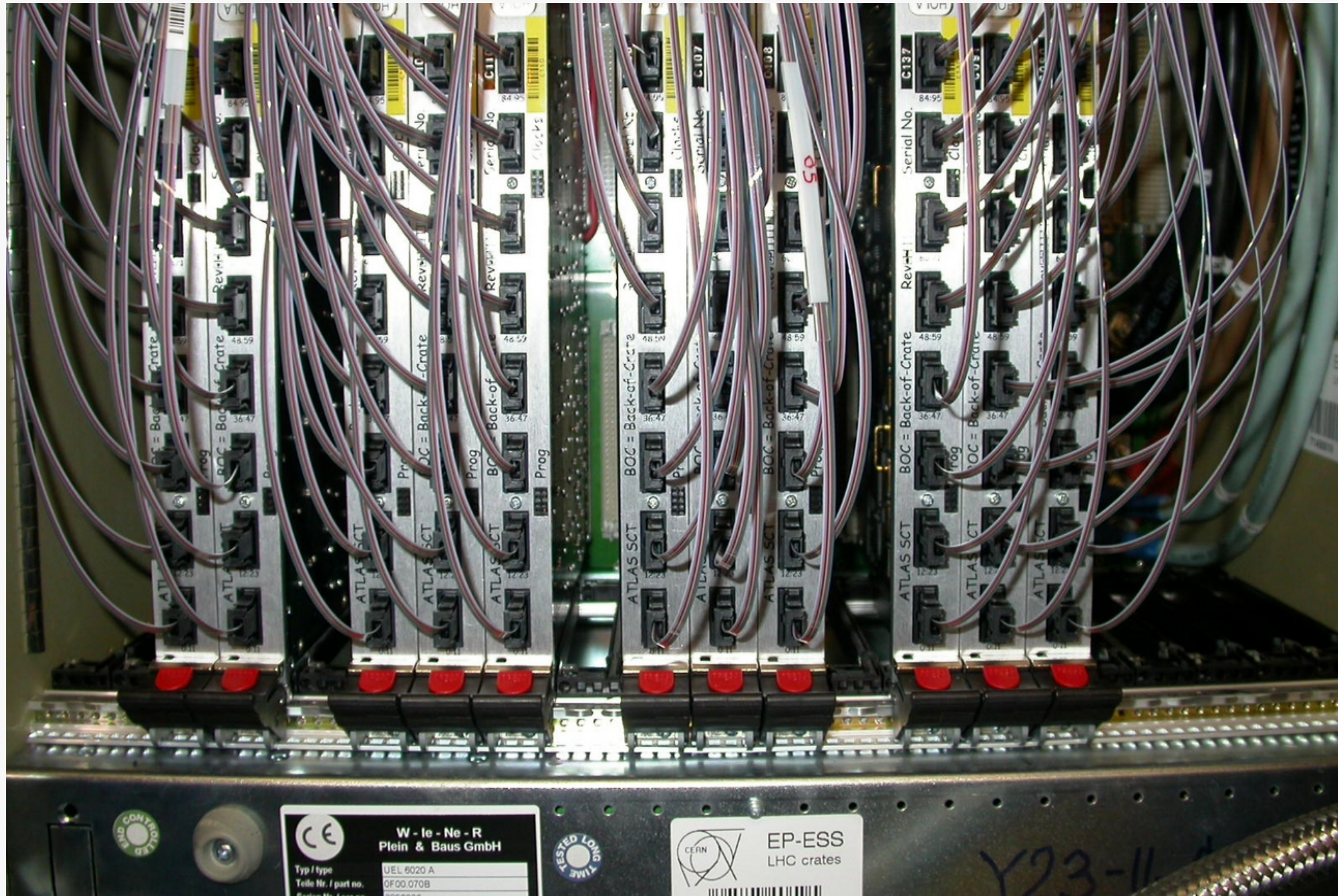


- In the design of our hierarchical data-collection system, we better define “**building blocks**”
 - ▶ Readout crates
 - ▶ HLT racks
 - ▶ event building groups
 - ▶ daq slices

Front End electronics

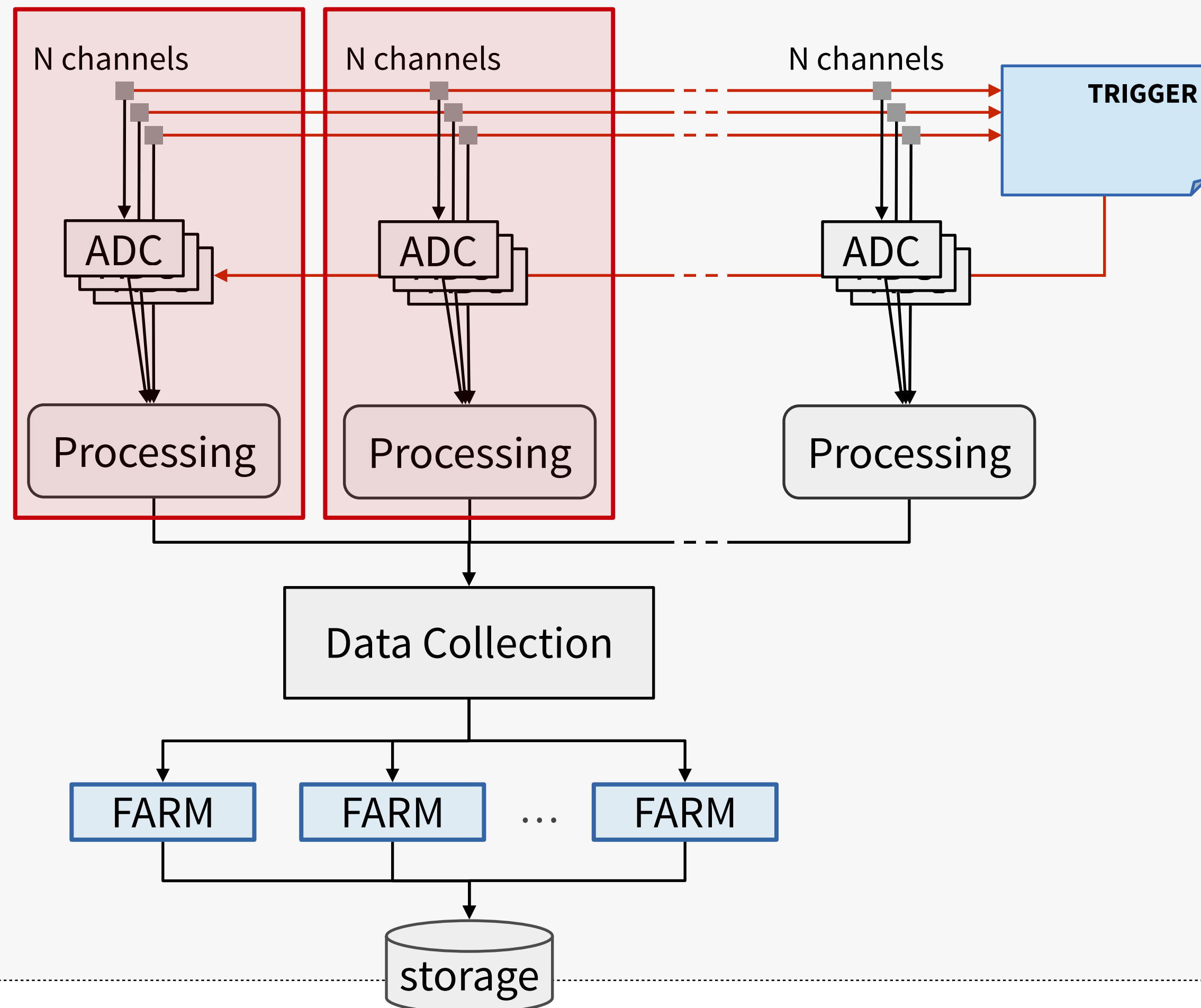


Readout Boards (Counting Room)



Building blocks

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



- In the design of our hierarchical data-collection system, we better define “**building blocks**”
 - ▶ Readout crates
 - ▶ HLT racks
 - ▶ event building groups
 - ▶ daq slices

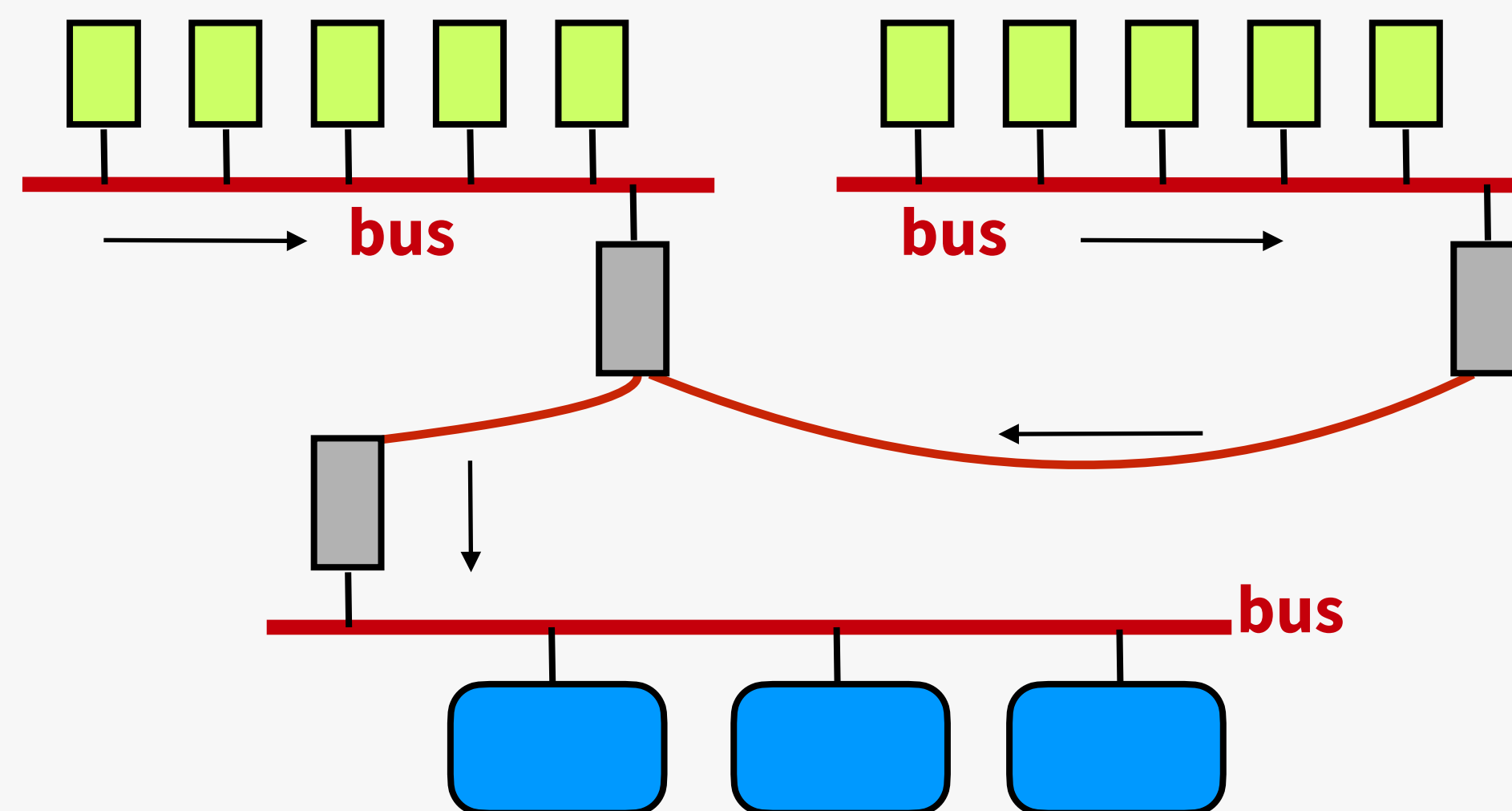
Farm (@surface)



Readout Topology

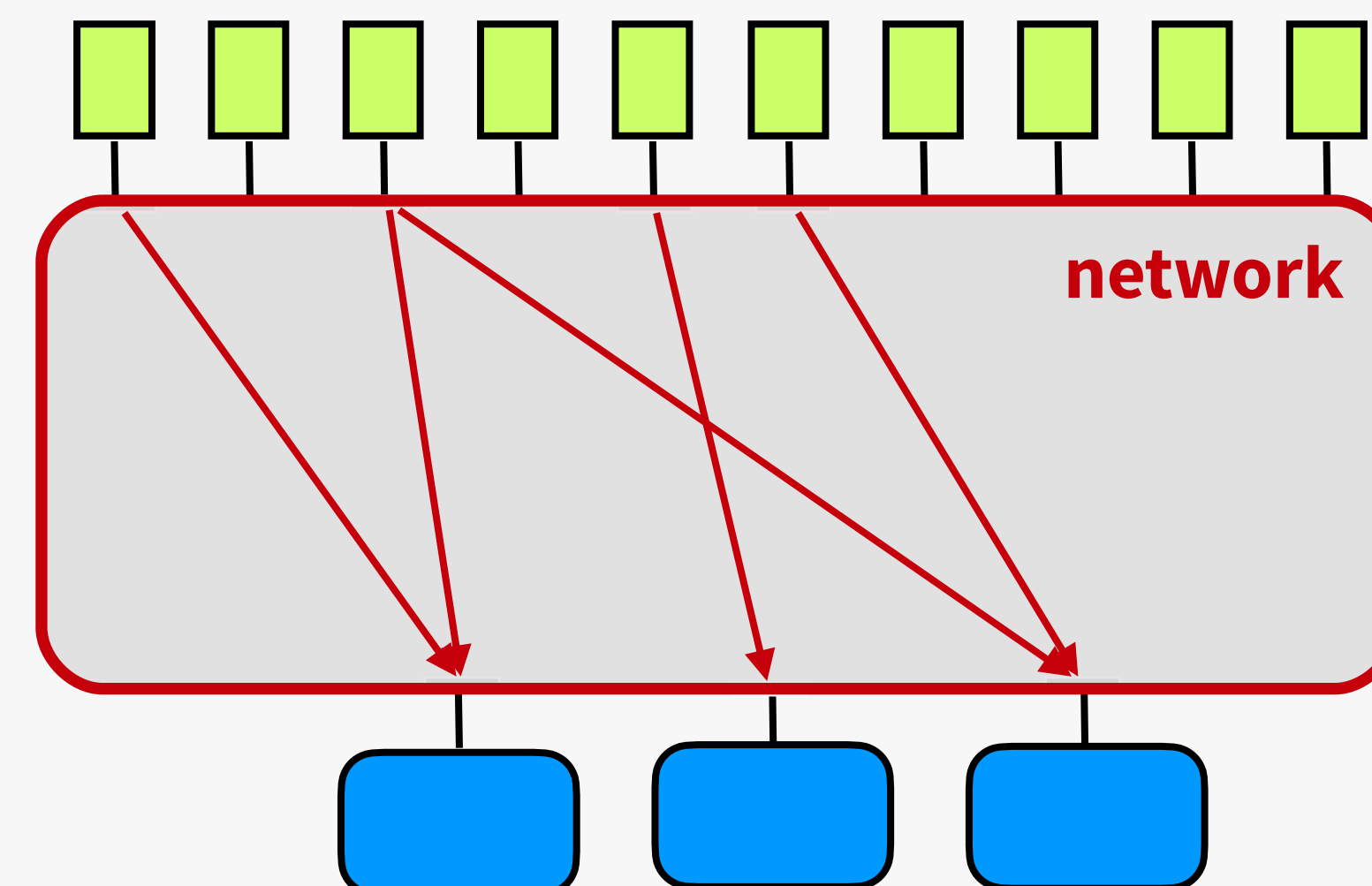
How to organize interconnections inside the building blocks and between building blocks?

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



data sources

data processors



Buses

Devices connected via a **shared bus**

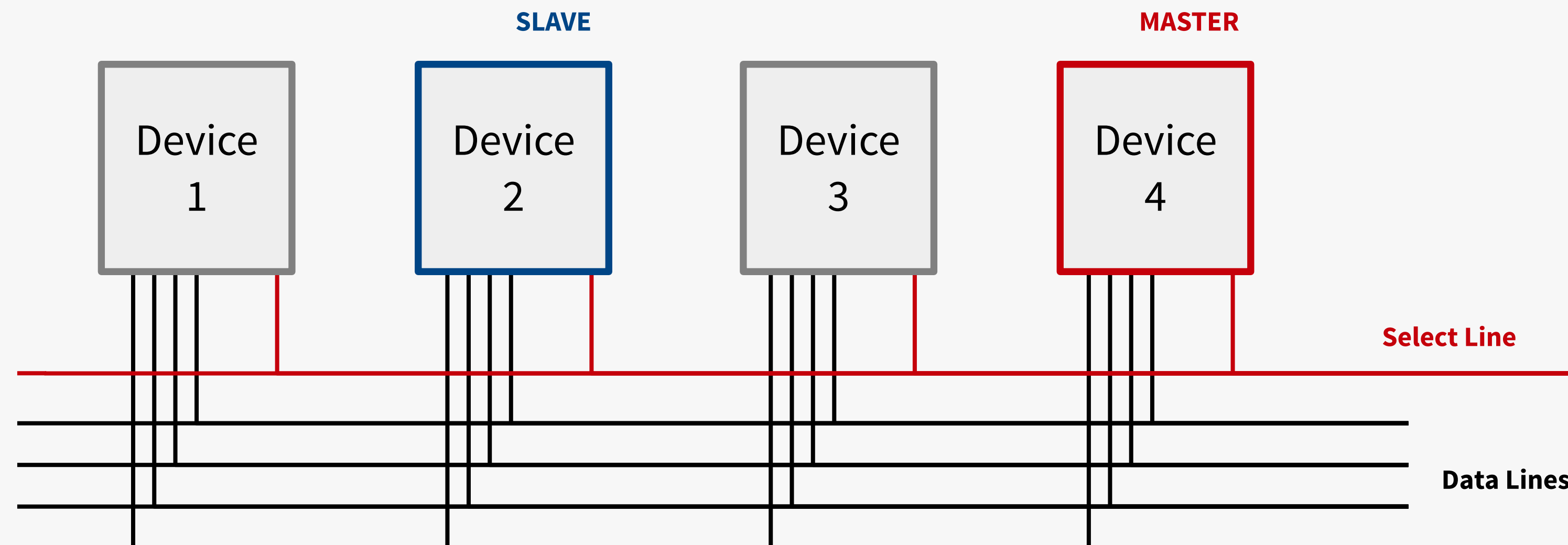
- Bus → group of electrical lines

Sharing implies **arbitration**

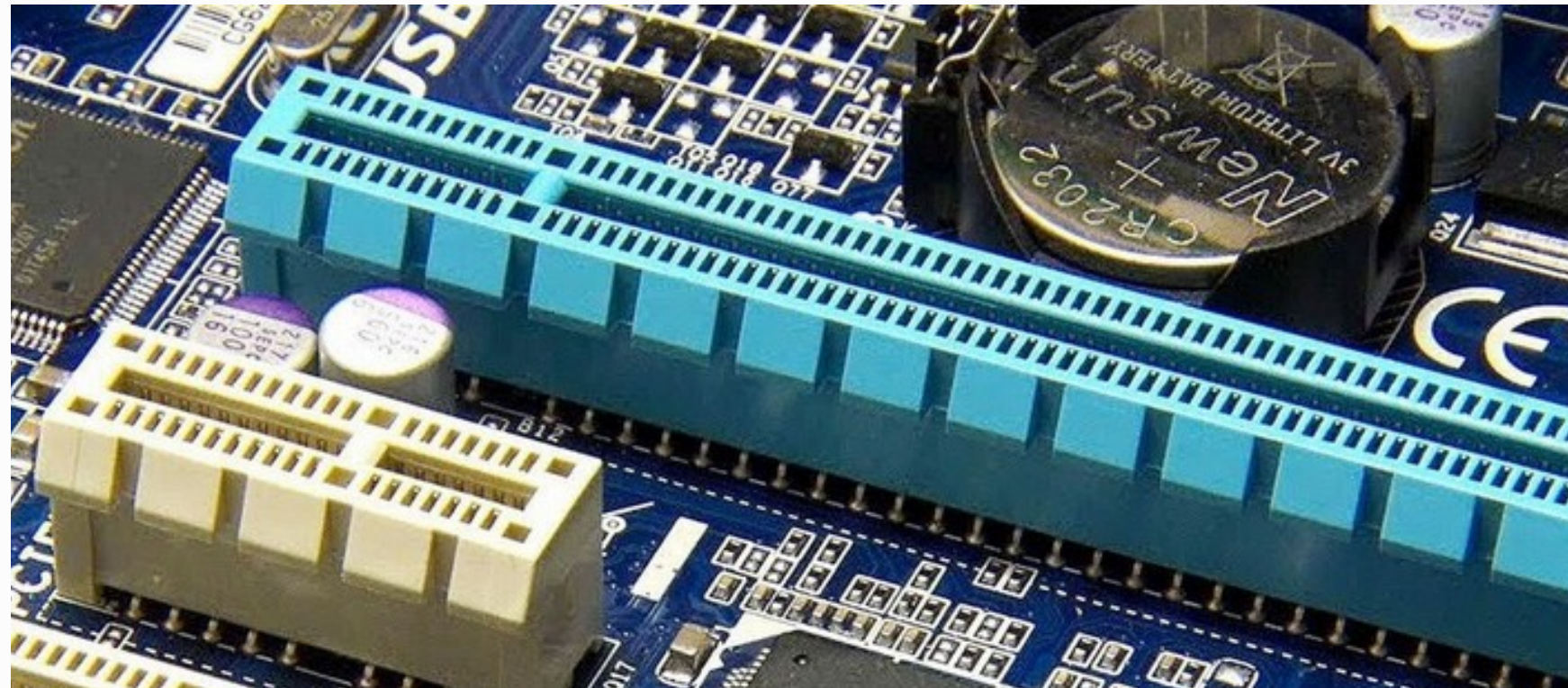
- Devices can be **master** or **slave**
- Devices can be addresses (uniquely identified) on the bus

E.g.: SCSI, Parallel ATA, VME, PCI ...

- local, external, crate, long distance, ...



Bus examples (some)



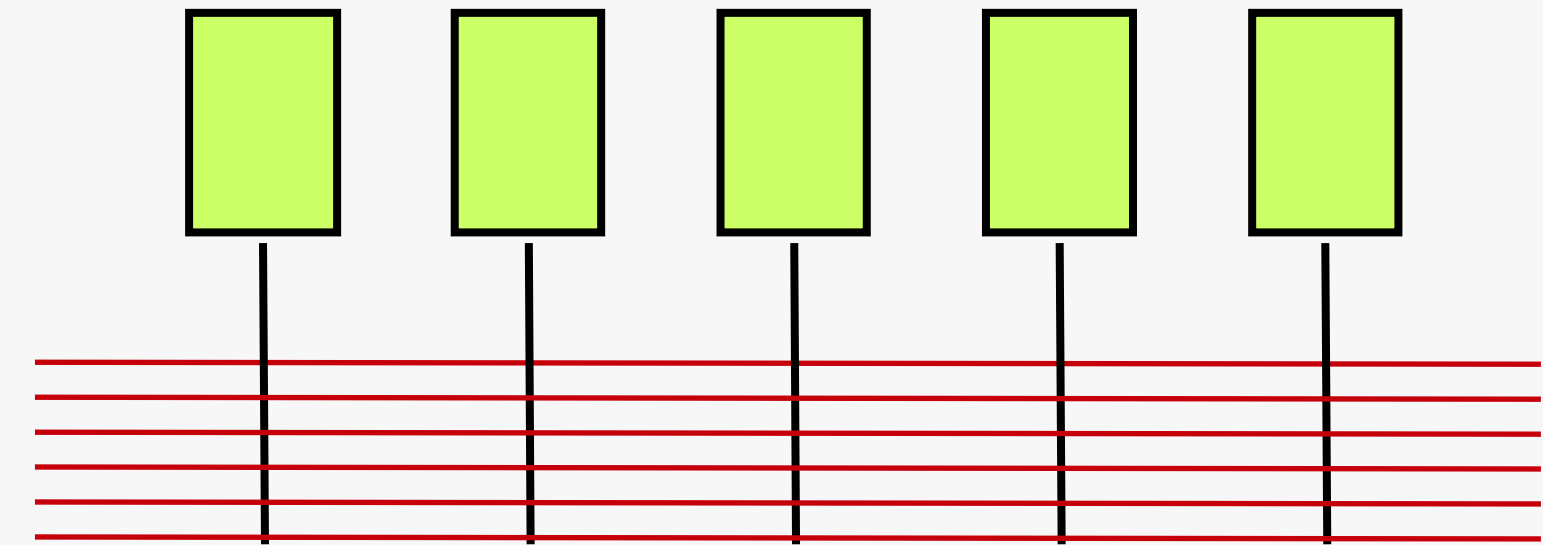
- PCI express
- VME
- μ TCA
- ATCA



Bus facts

Simple :-)

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



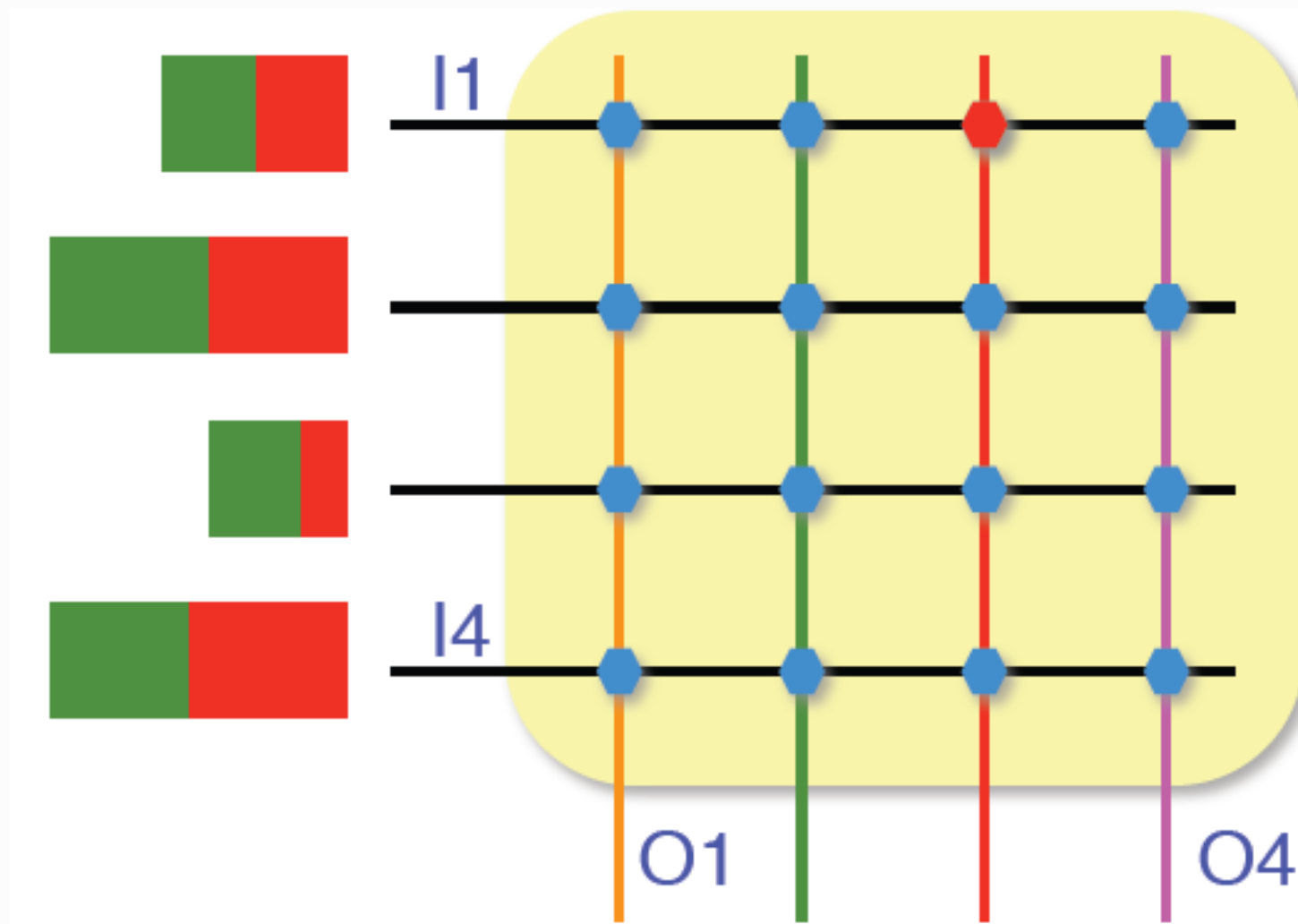
Scalability issues :-)

- 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, other “effects” might limit the scalability of your system

Bus facts

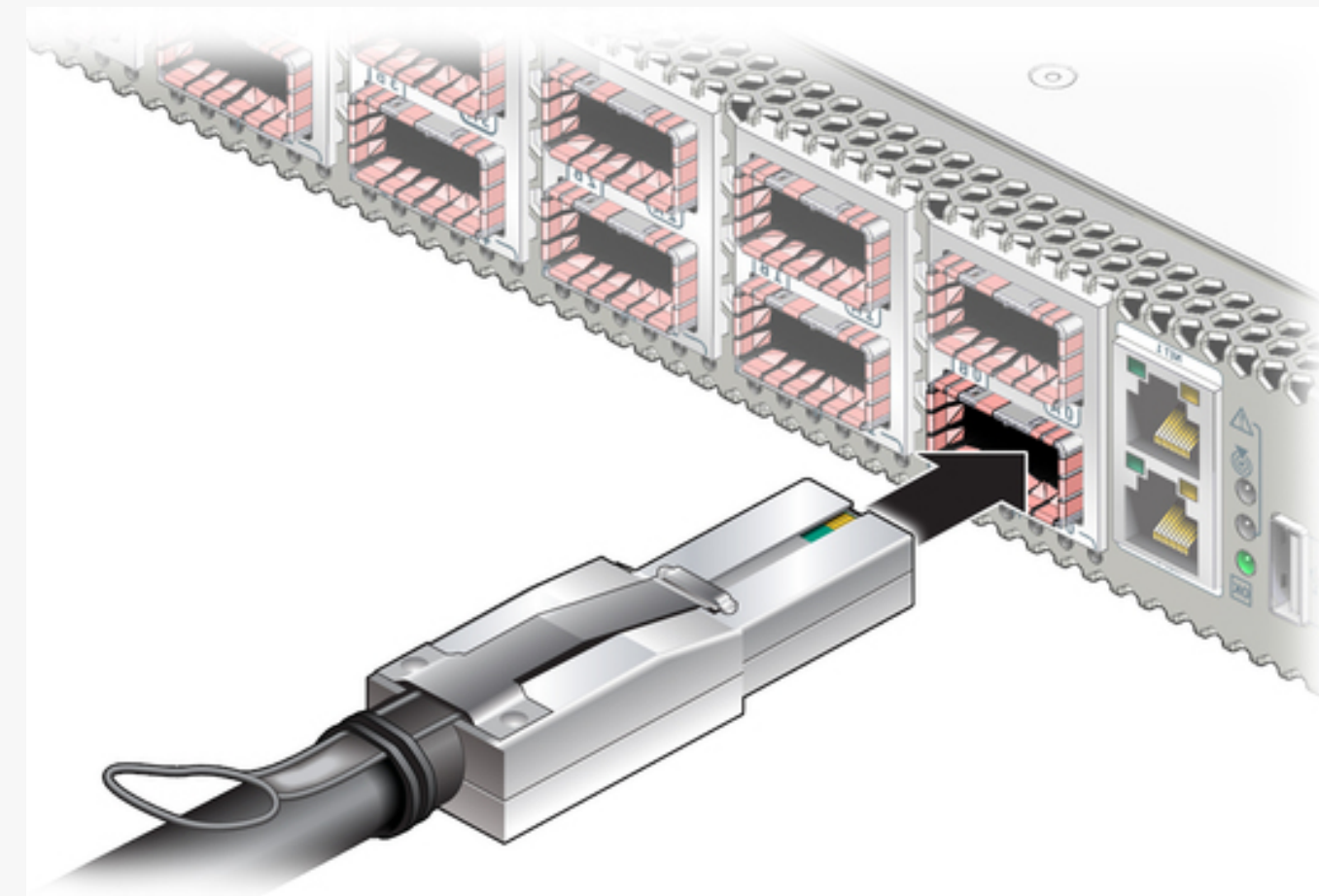
On the long term,
2nd order effects might limit the scalability of your system

Networks

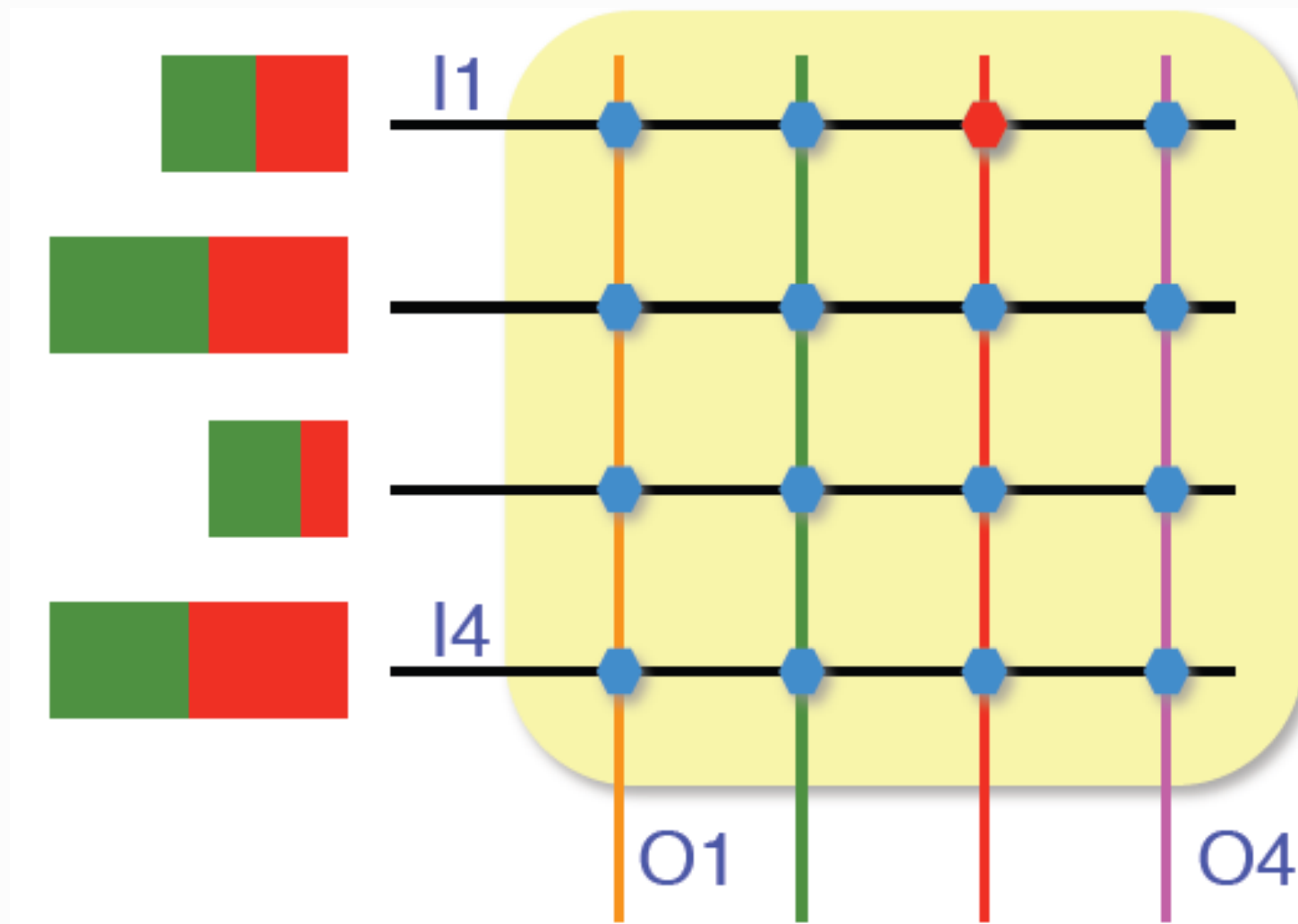


All devices are **equal (peers)**

- 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

- Find the right path

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

- The key is

Networks



**Cable management is still a thing.
It can still go very wrong if you're not careful...**

Outline

1. Introduction

1.1. What is DAQ?

1.2. Overall framework

2. Basic DAQ concepts

2.1. Digitization, Latency

2.2. Deadtime, Busy, Backpressure

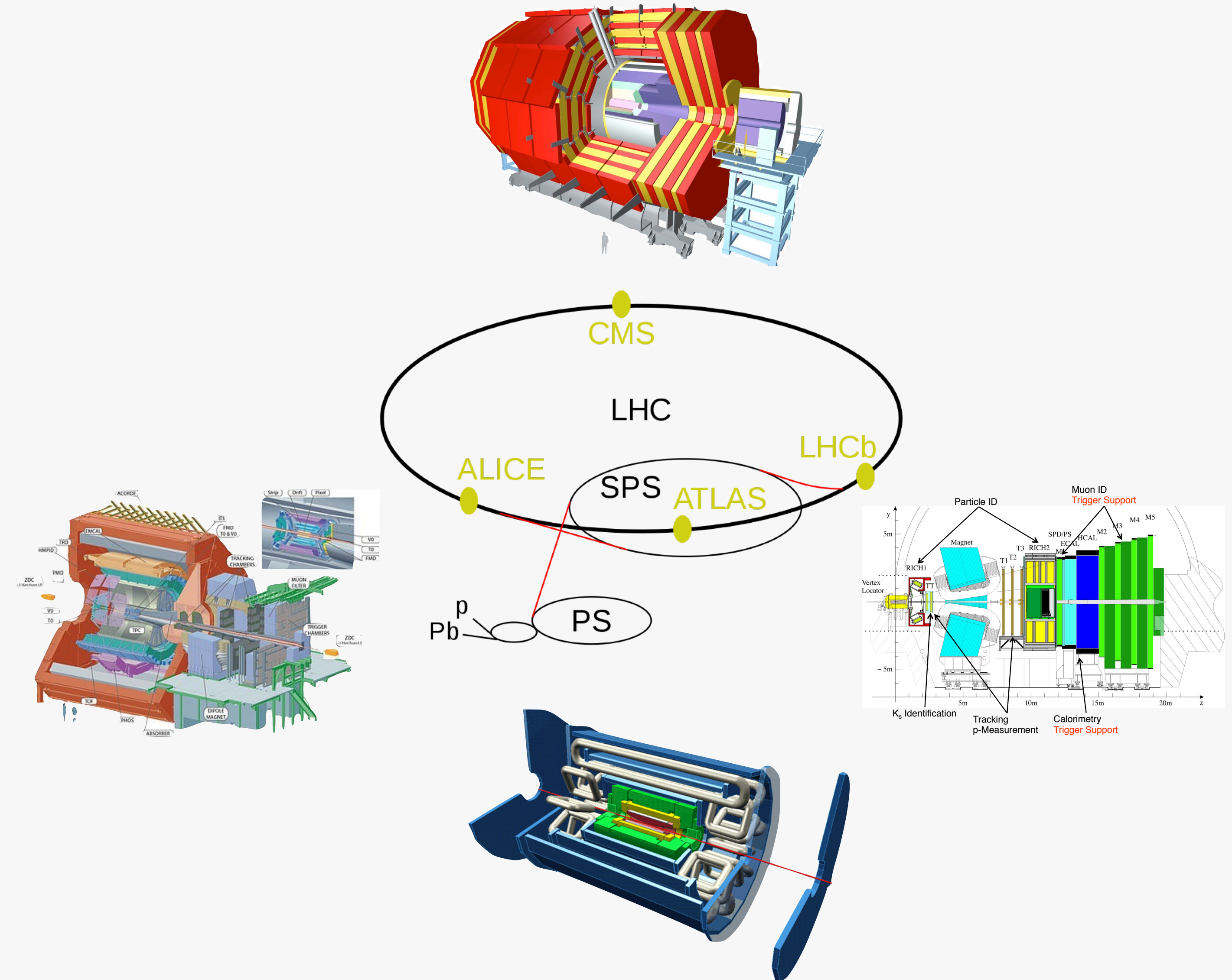
2.3. De-randomization

3. Scaling up

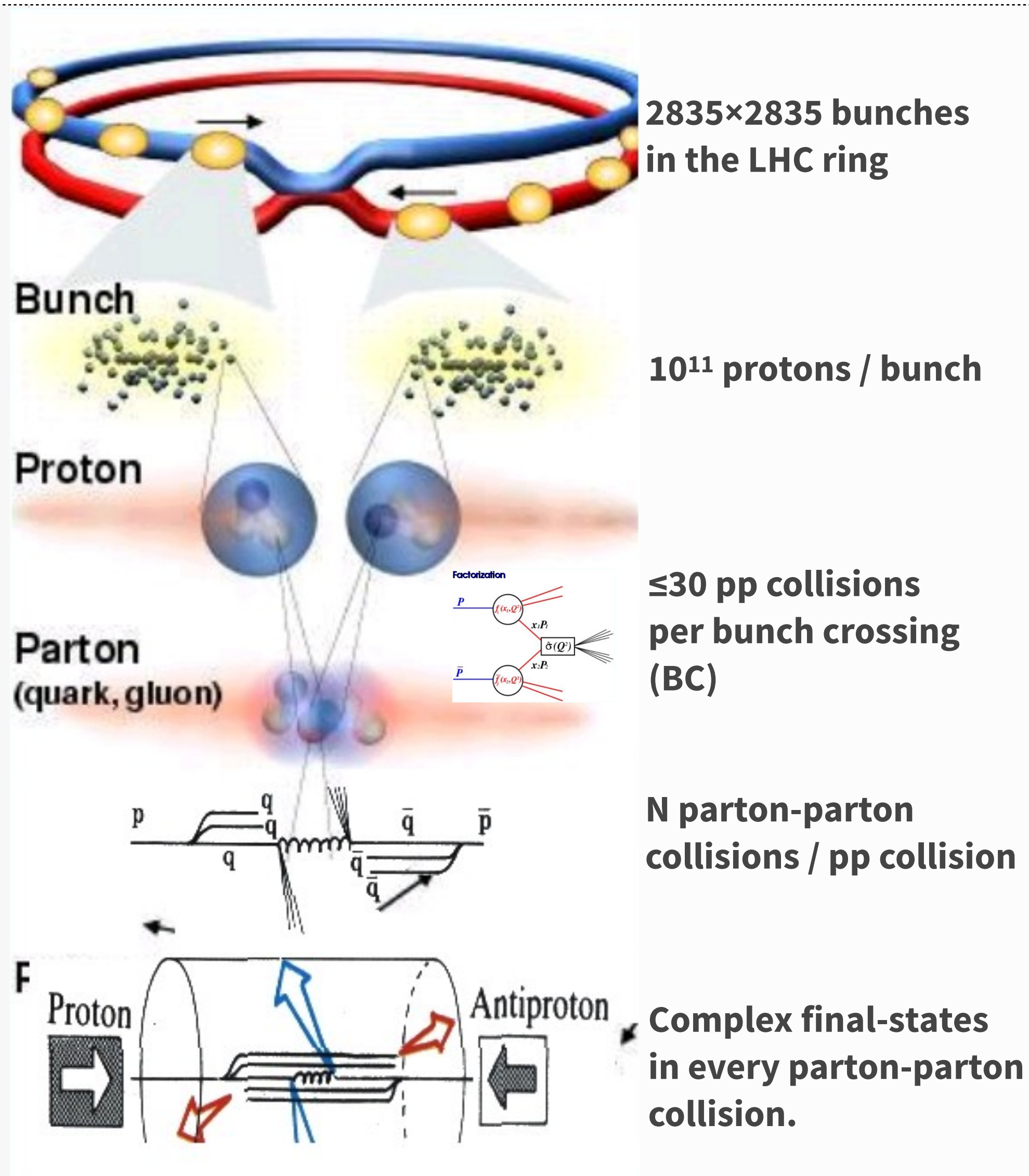
3.1. Readout and Event Building

3.2. Buses vs Network

4. DAQ Challenges at LHC



LHC engine and its products



Design parameters

$$E_{\text{cms}} = 14 \text{ TeV}$$

$$L = 10^{34} / \text{cm}^2 \text{ s}$$

$$\text{BC clock} = 40 \text{ MHz}$$

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

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

- **Close collisions in space and time**
 - ▶ Large proton bunches (1.5×10^{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

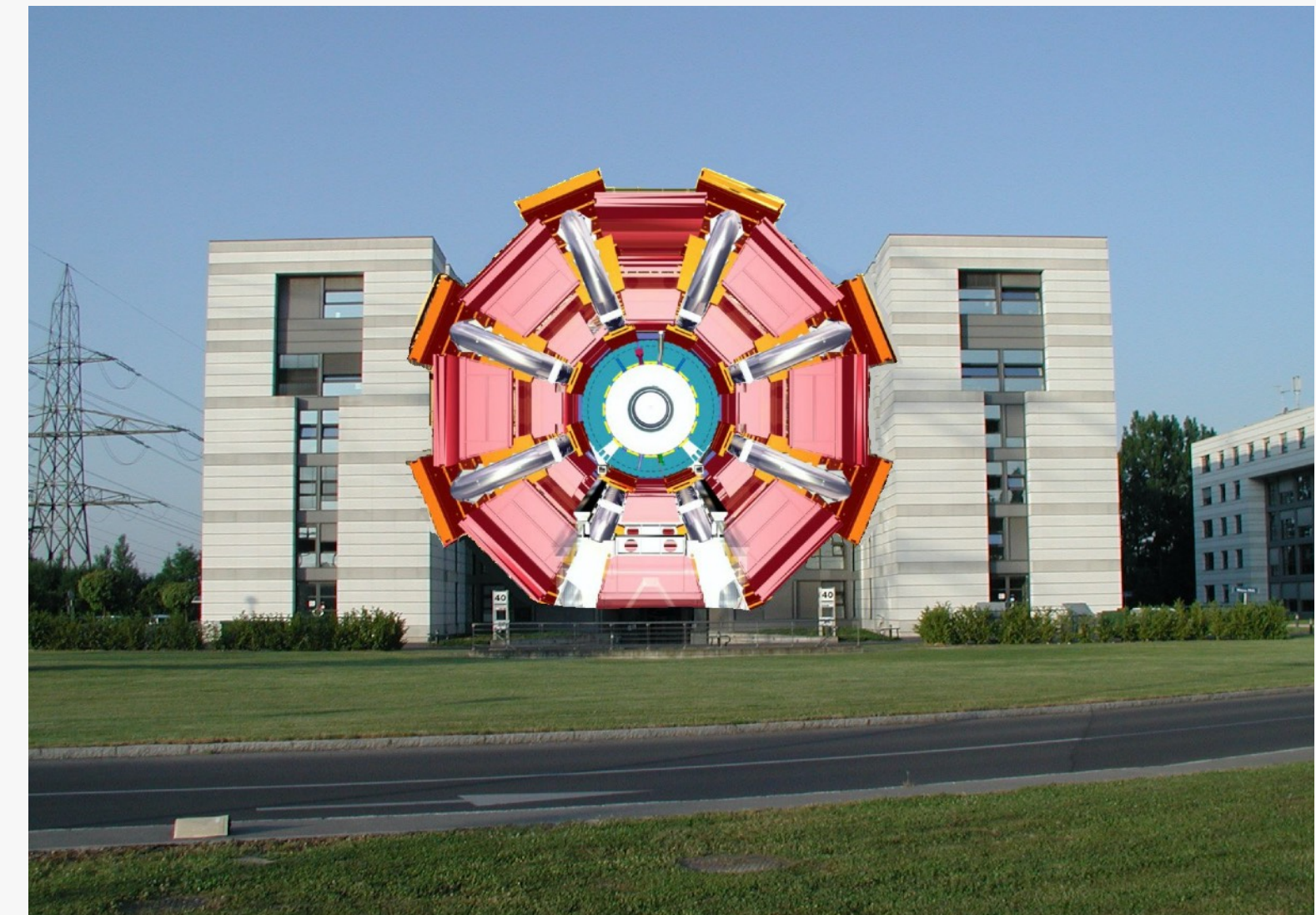
- $O(10^6-10^8)$ channels
- ~1 MB event size for pp collisions
 - ▶ 50 MB for pb-pb collisions (Alice)
- Need huge number of connections

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

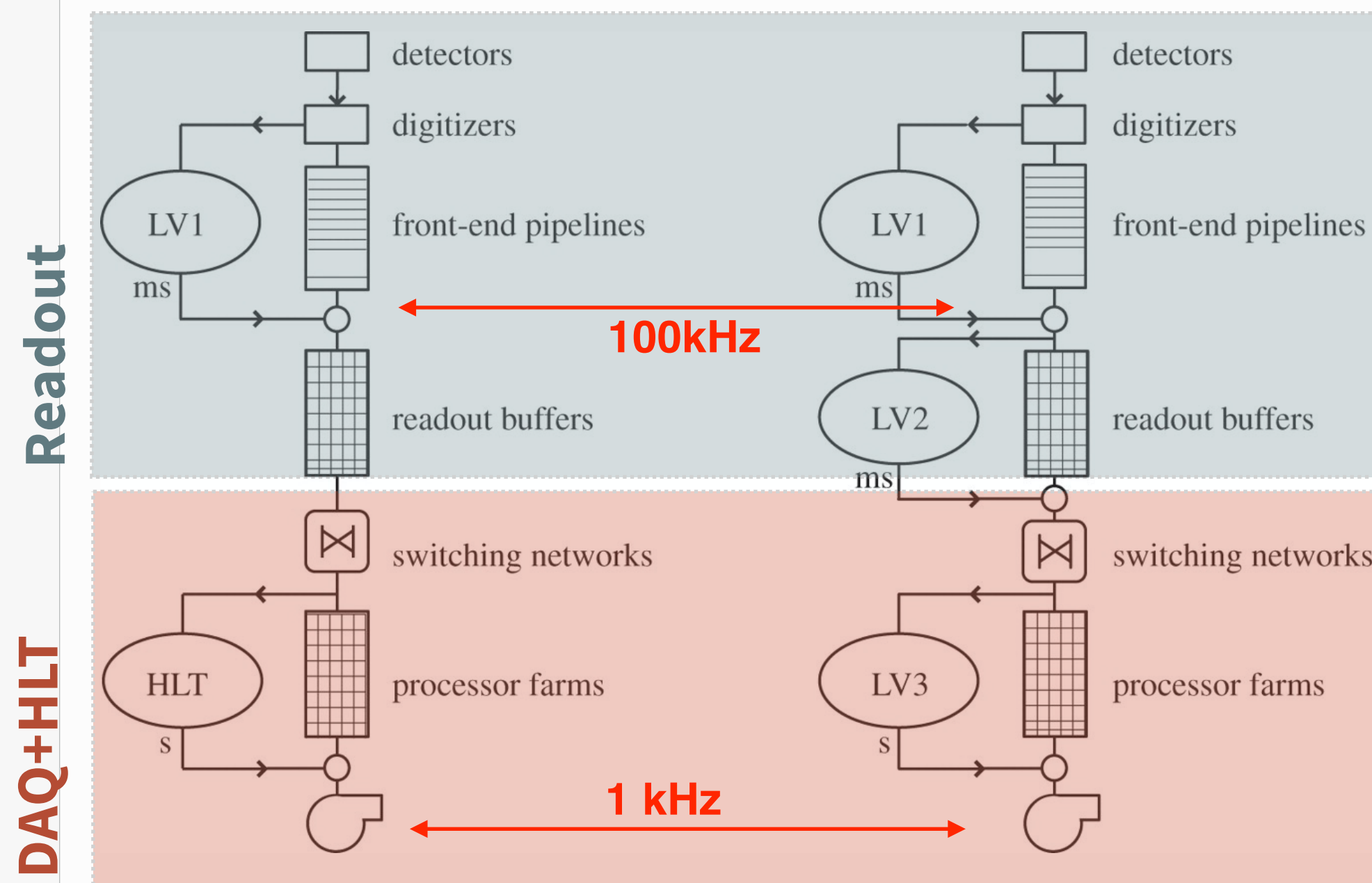
Online, what is lost is lost forever

- 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 $O(100\text{ms})$ HLT latency
 - Network: $1\text{ MB} \cdot 100\text{ kHz} = 100\text{ GB/s}$
 - HLT farm: $100\text{ kHz} \cdot 100\text{ ms} = O(10^4)$ 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

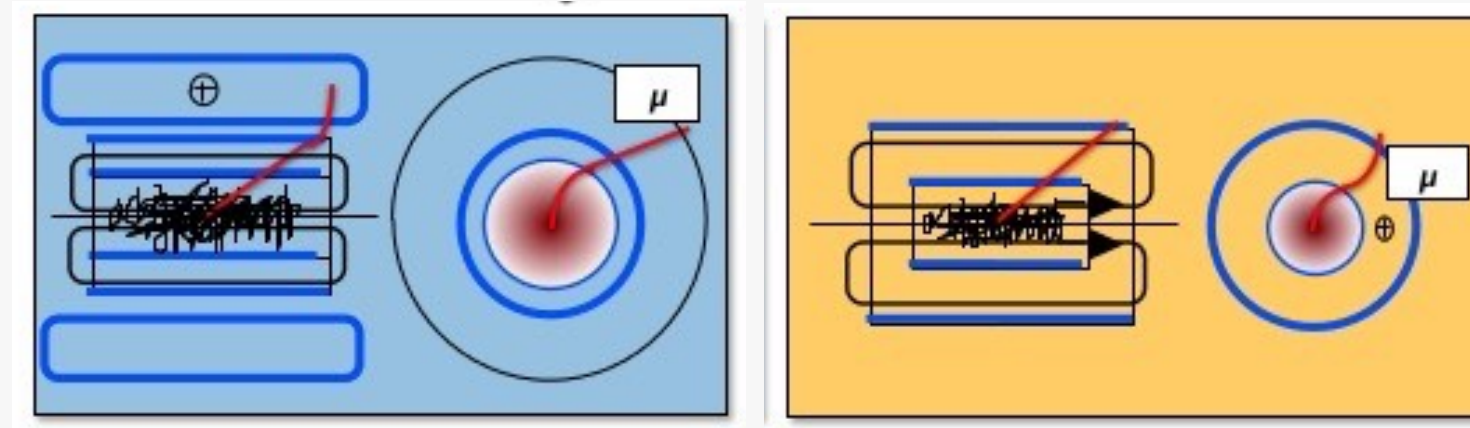
ATLAS & CMS design principles



Same physics program

Different magnetic field structure

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

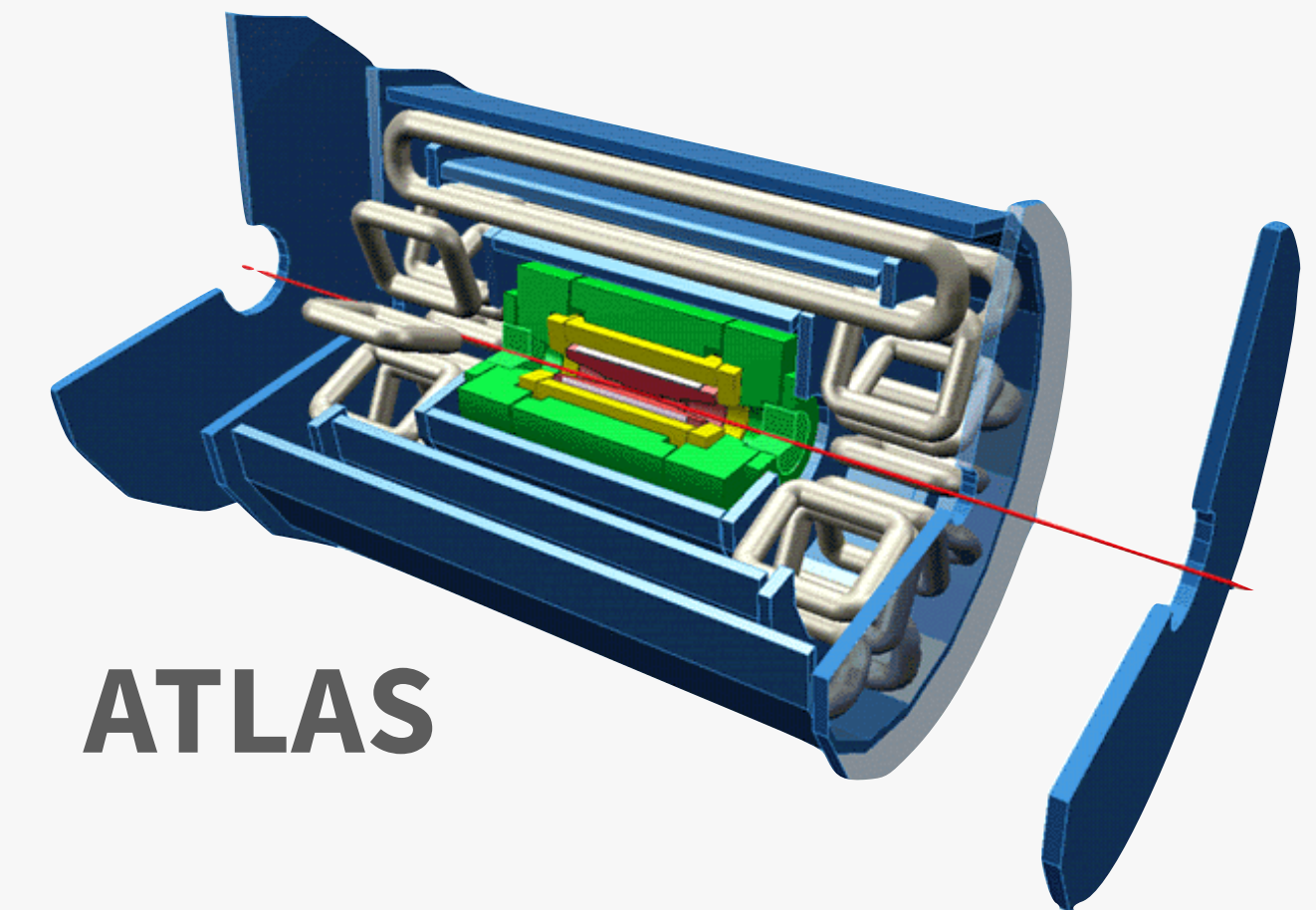


Different DAQ architecture

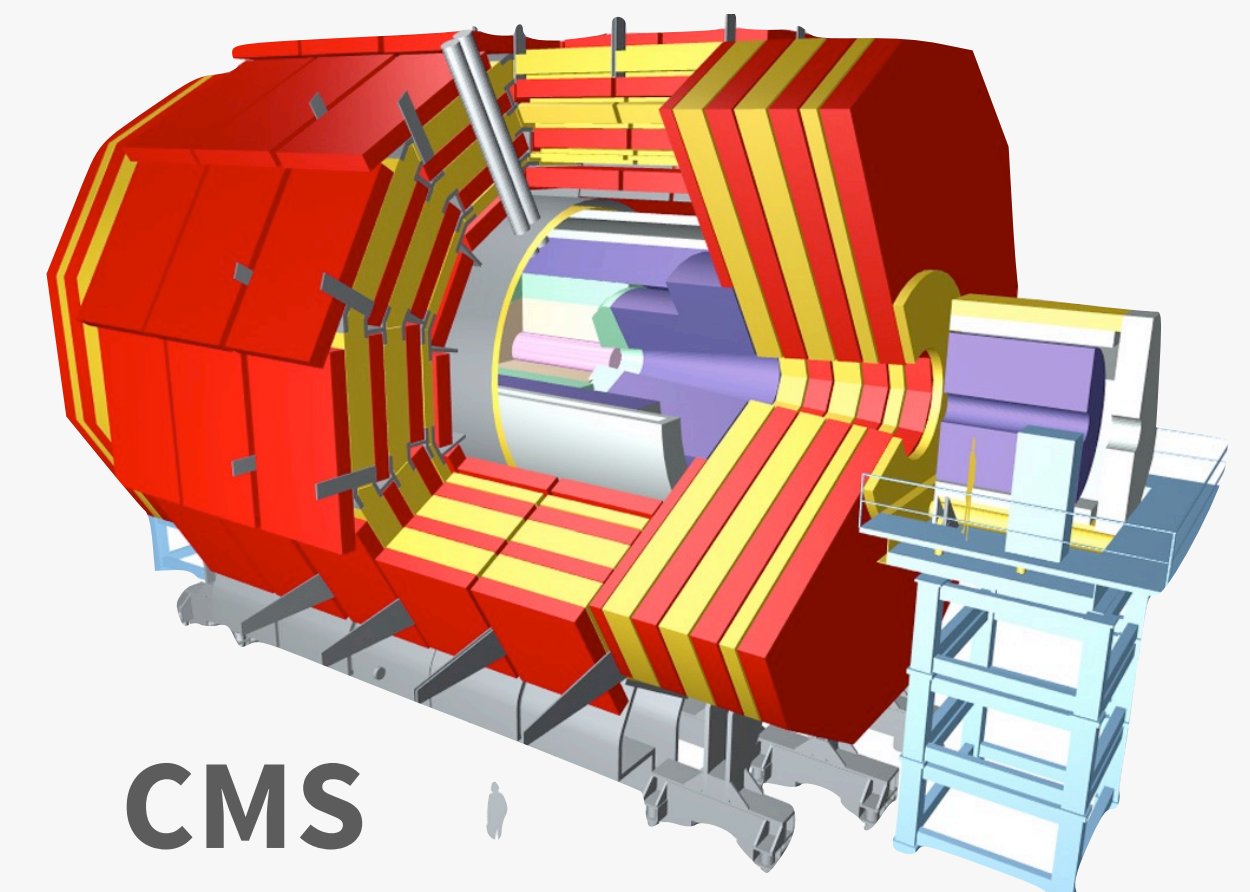
- **ATLAS**: minimise data flow bandwidth with multiple levels and regional readout
- **CMS**: large bandwidth, invest on commercial technologies for processing and communication

Same data rates

- $\sim 1 \text{ MB} * 100 \text{ kHz} = \sim 100 \text{ GB/s}$ readout network



ATLAS



CMS

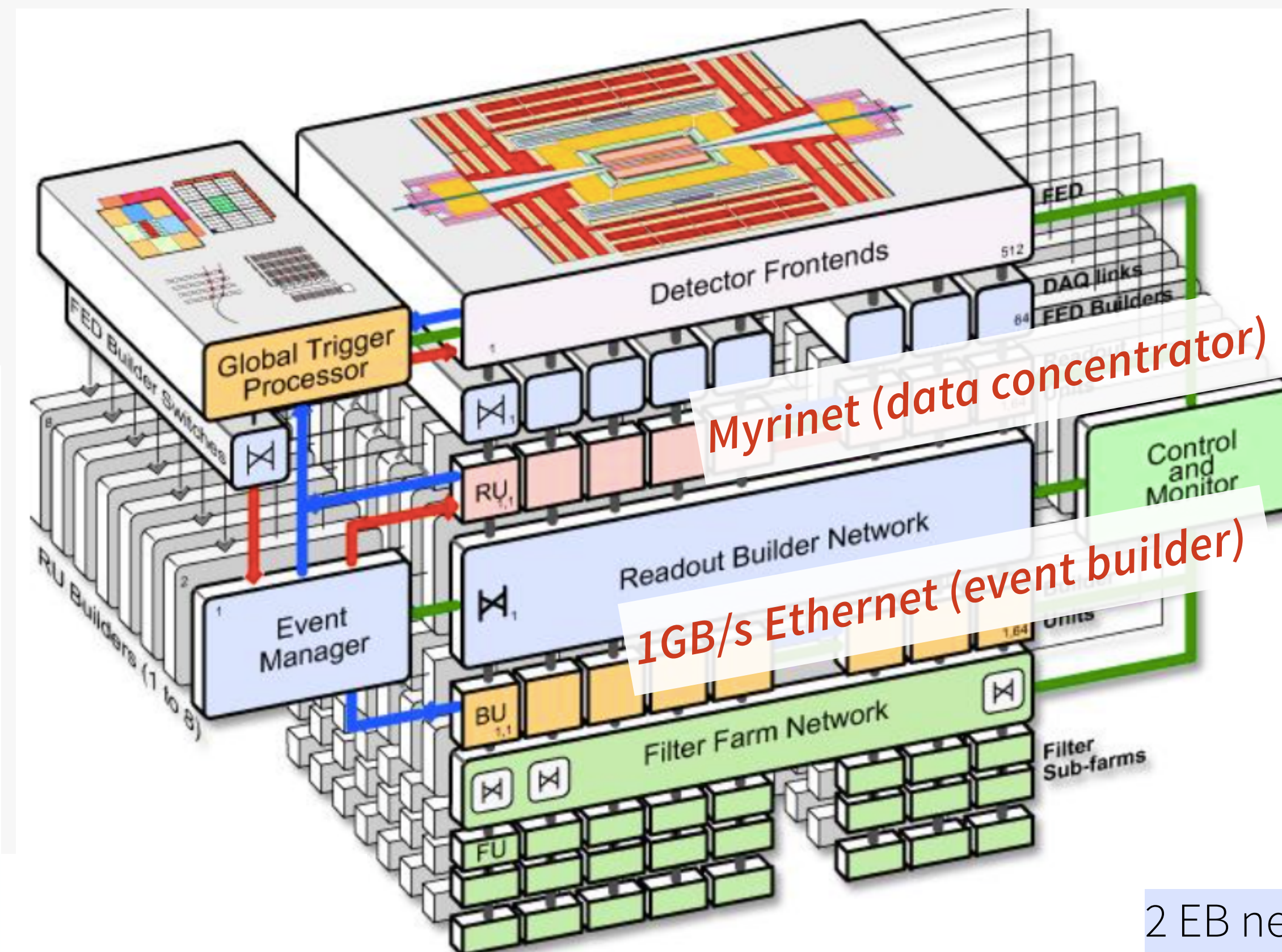
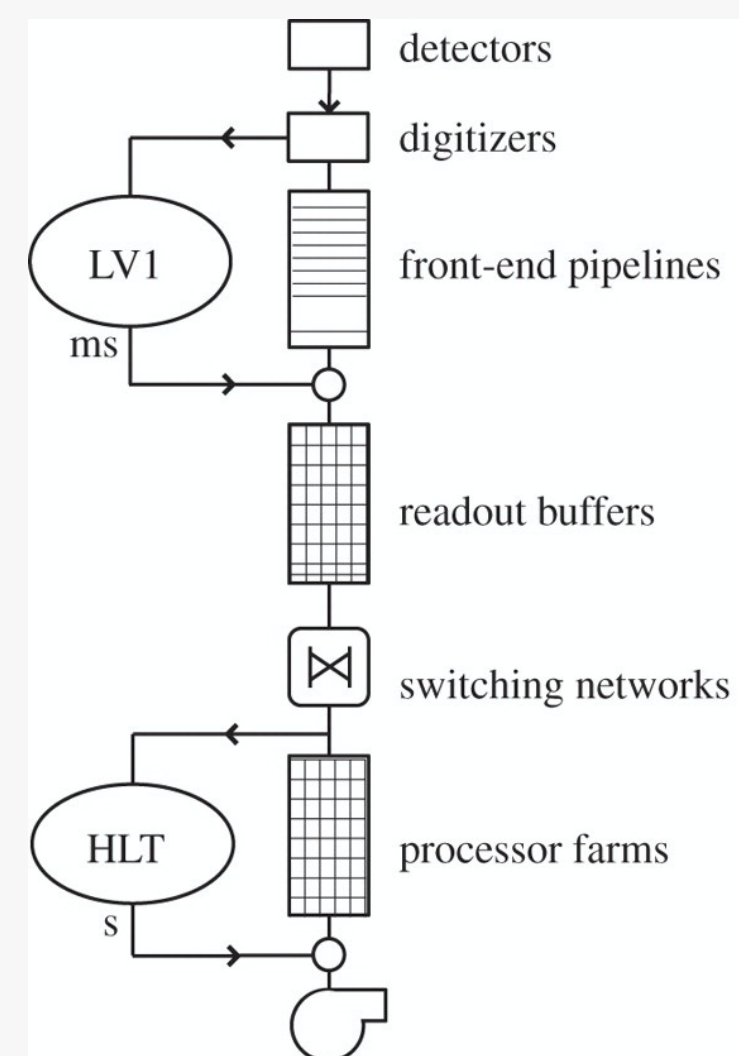
CMS: 2-stage Event building

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

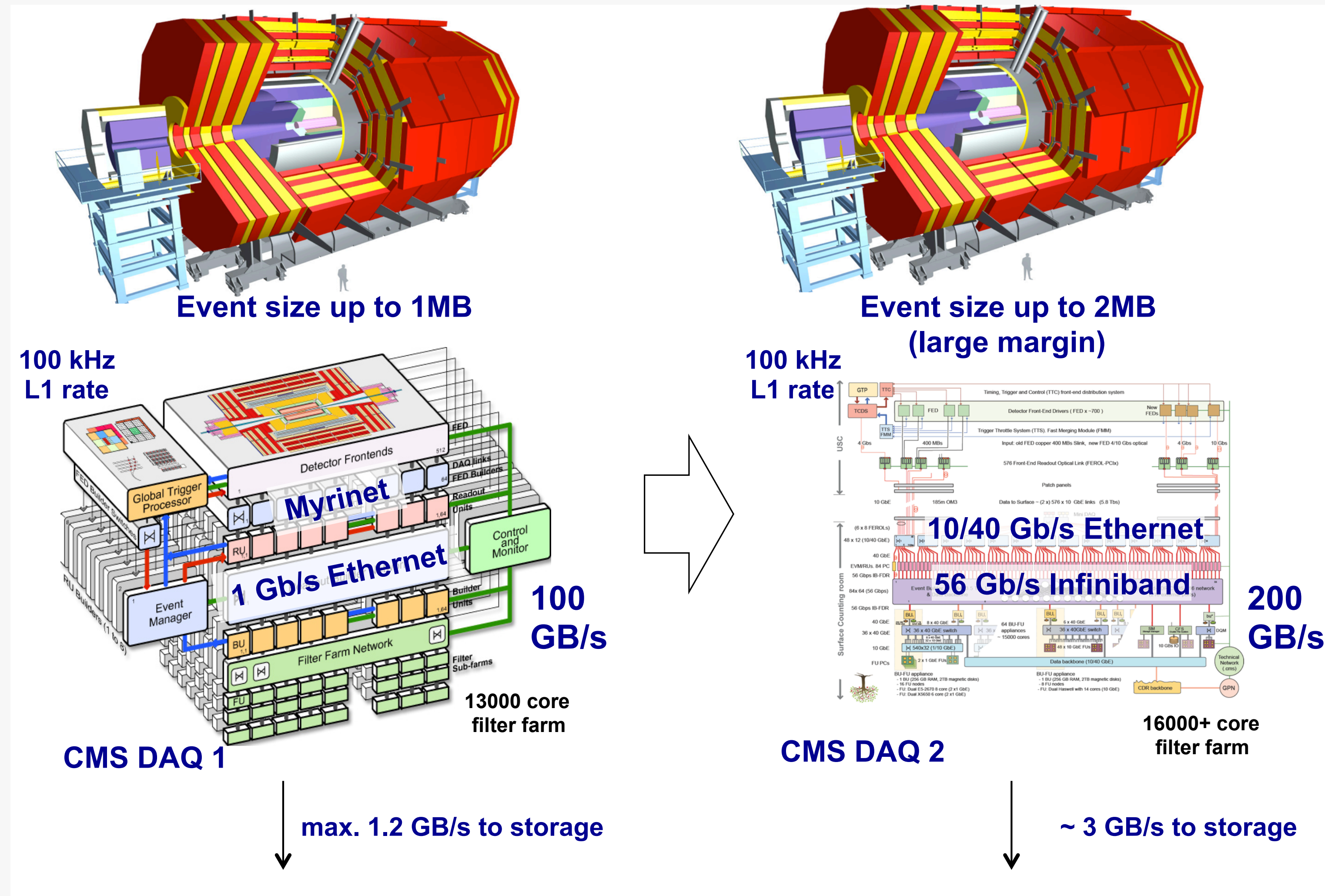


2 EB networks

Filter network

Evolution from LHC Run-1 to Run-2

CMS DAQ-2



ATLAS: Region of Interest (ROI) dataflow

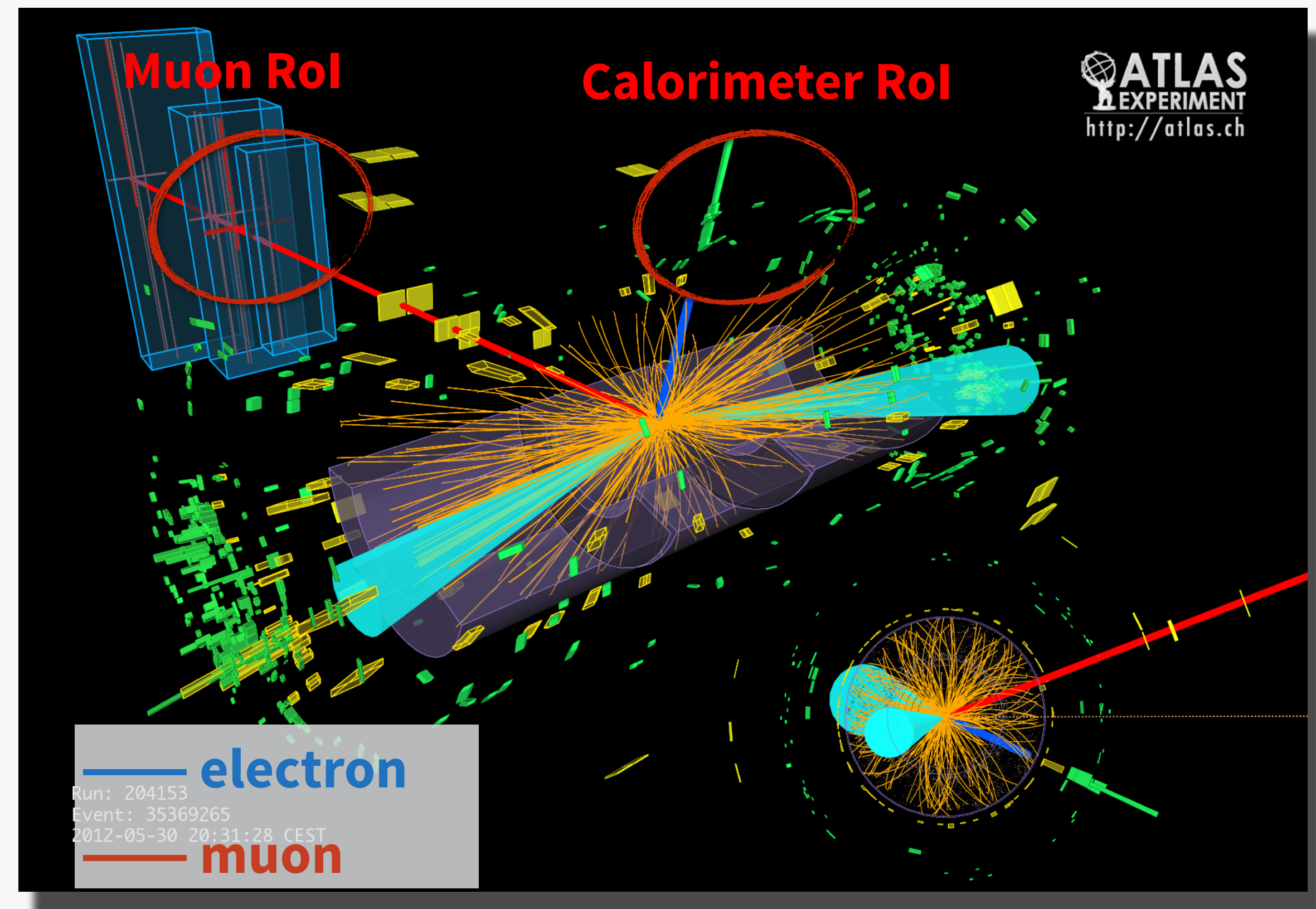


HLT selections based on regional readout and reconstruction

- seeded by L1 trigger objects (RoI)

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

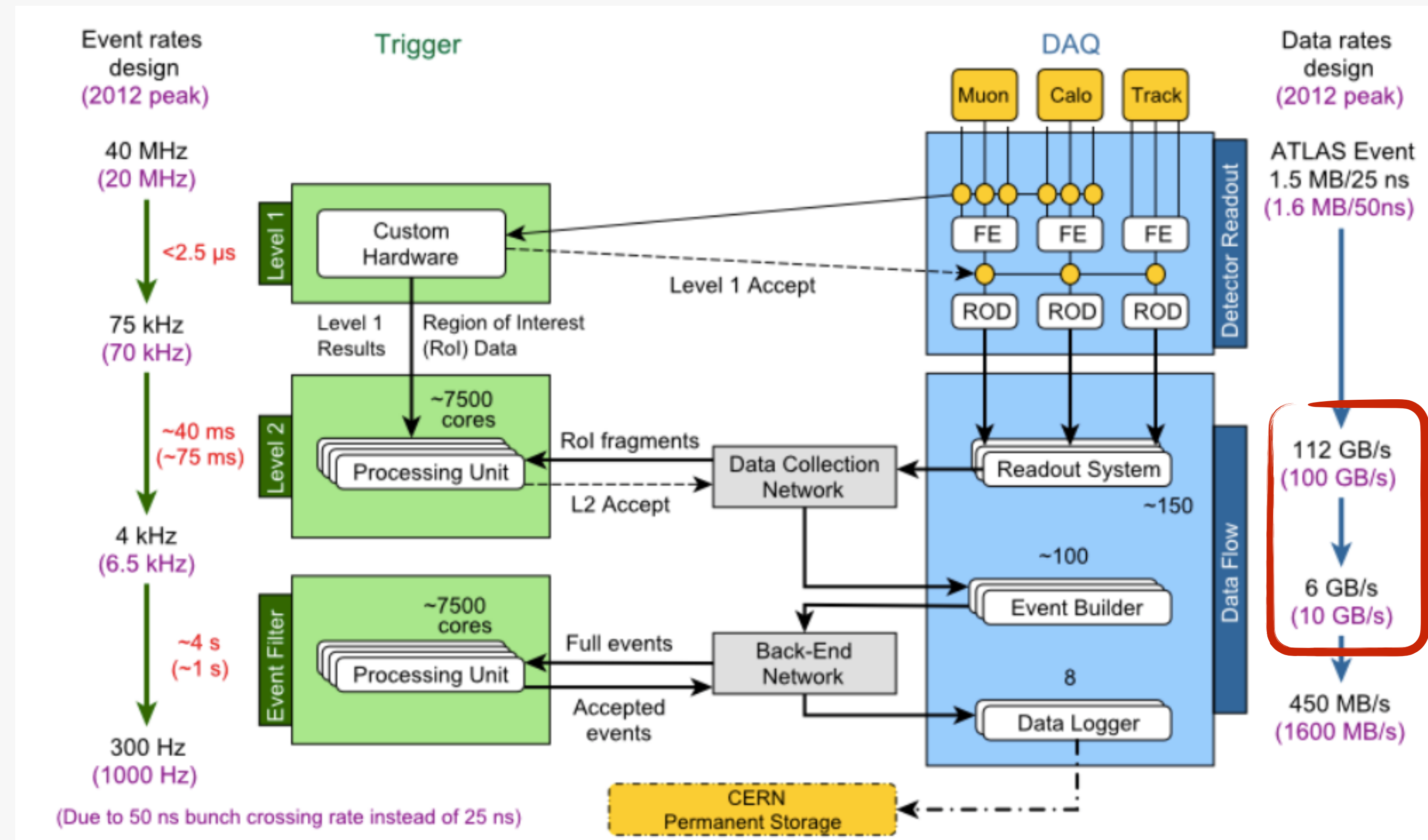
- one order of magnitude smaller readout network
- at the cost of a higher control traffic and reduced scalability



Overall network bandwidth: **~10 GB/s**

- x10 reduced by regional readout)

Complex data routing

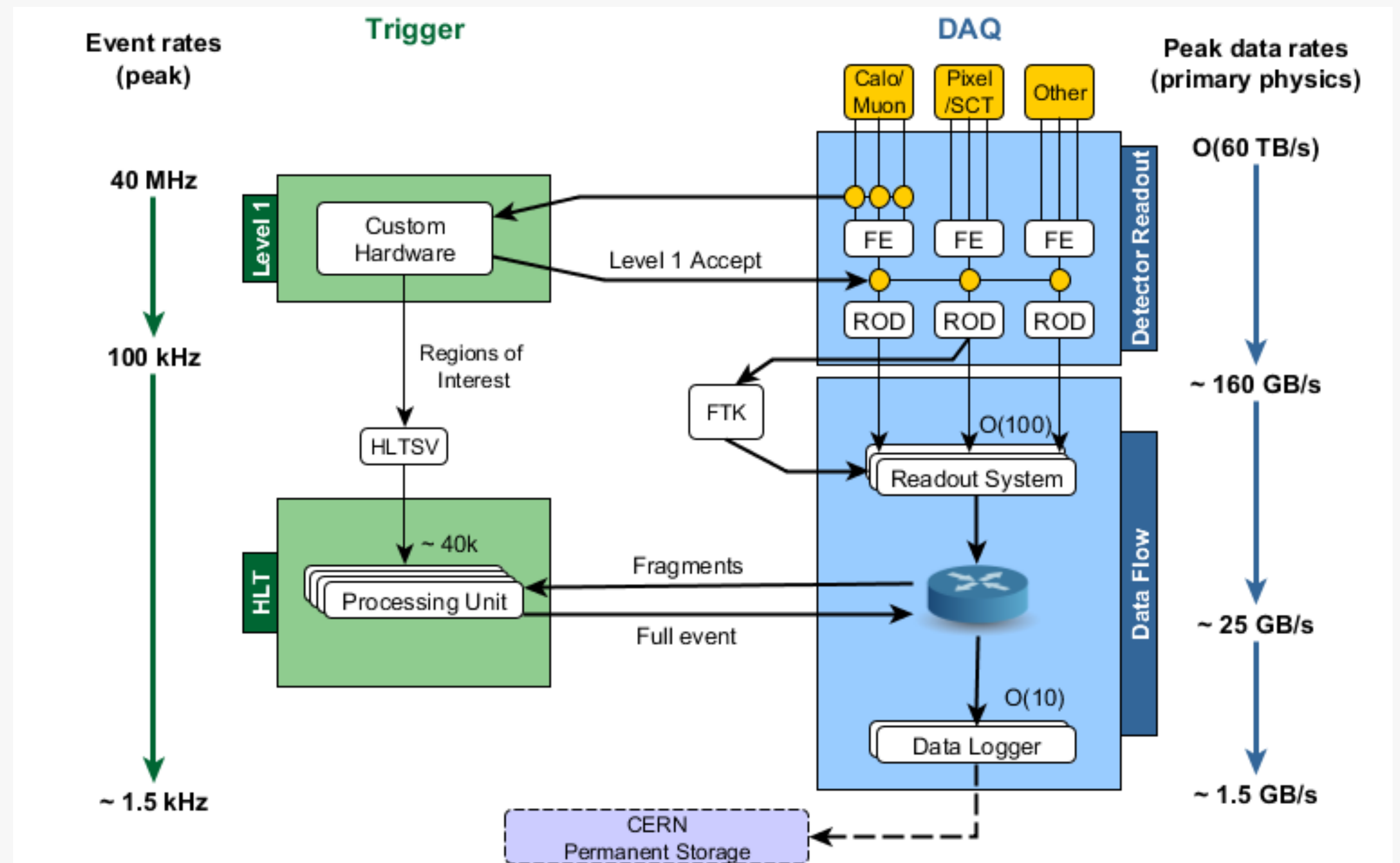


NEW TDAQ architecture for Run-2

RUN 2

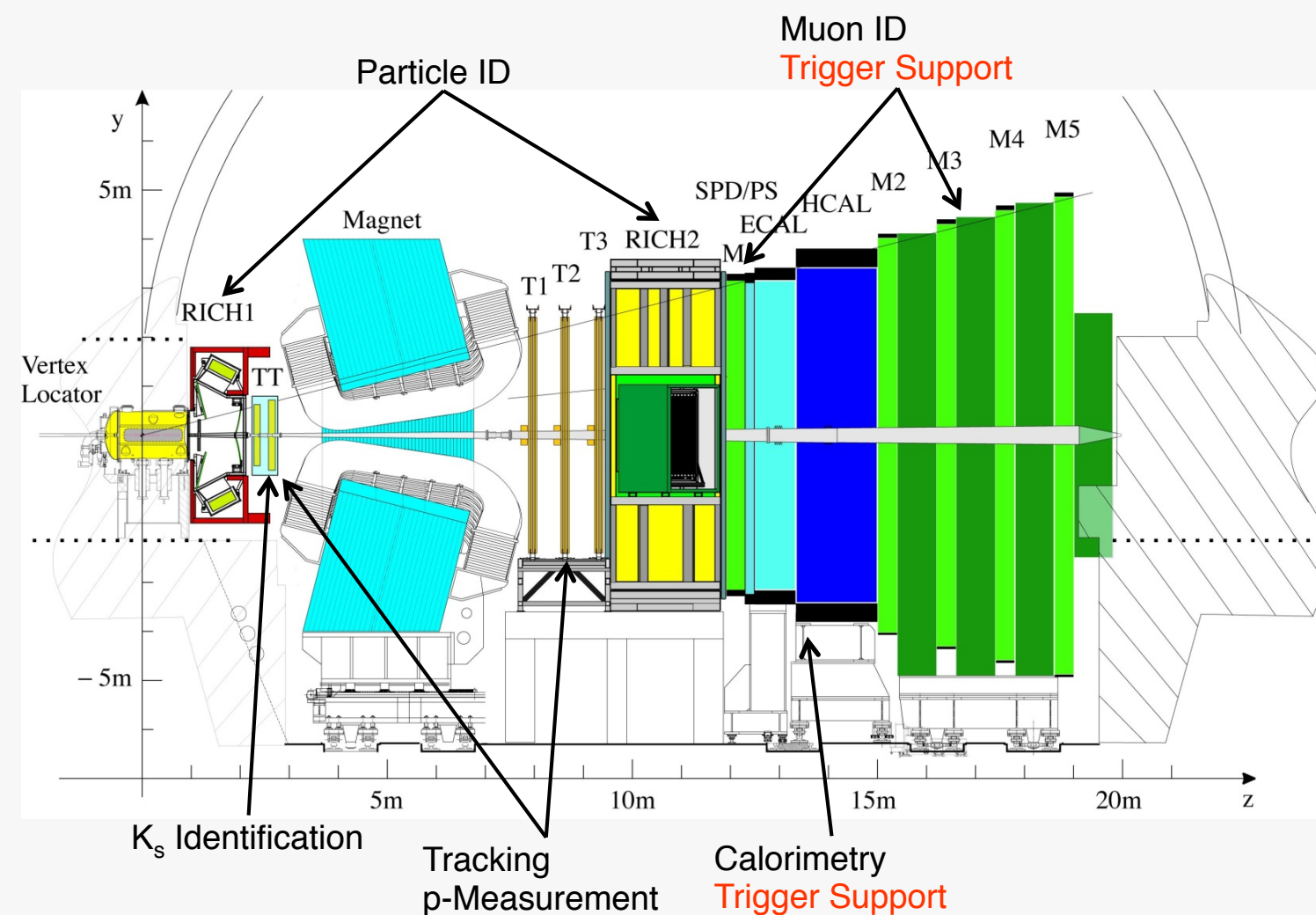


- Increased rates
- Merged L2/HLT
- Increase Readout bandwidth
- Increase HLT rate
- Unified network



LHCb TDAQ Architecture

RUN 2

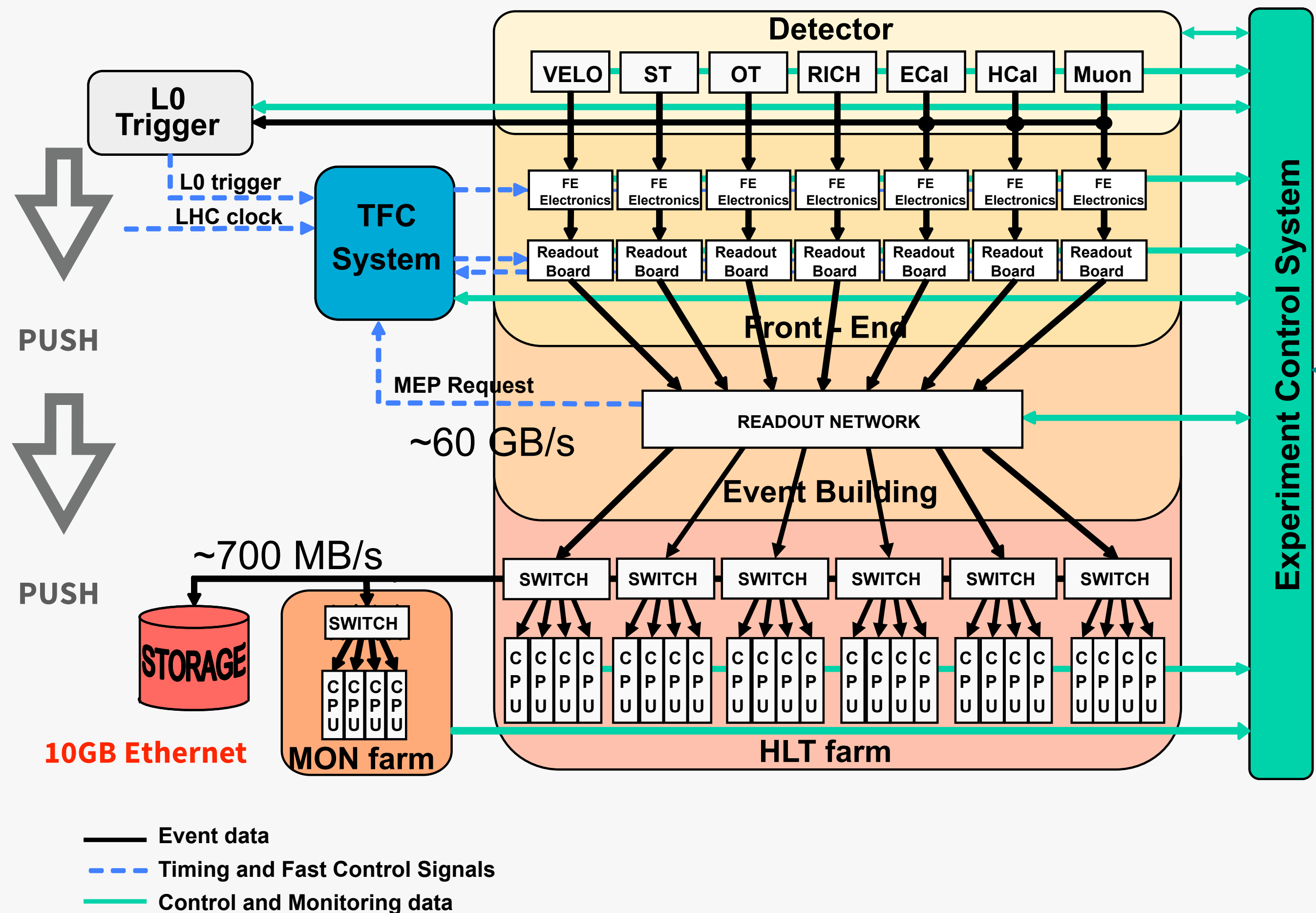


Single forward arm spectrometer → reduced event size

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

Small event, at high rate

- optimised transmission



(Level-1) Trigger-less!

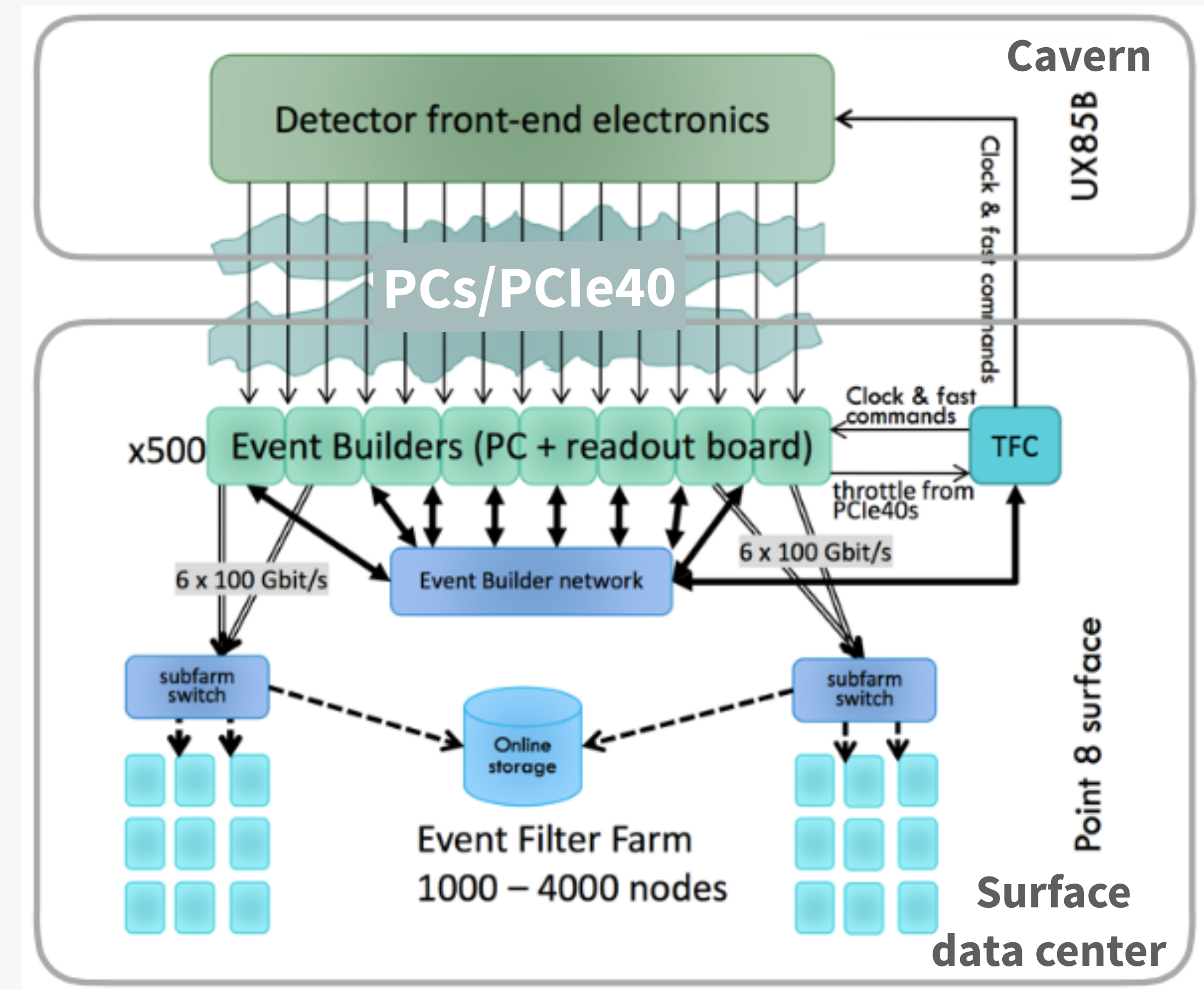
Data reduction before EB

- custom readout FPGA-card (PCle40)
- 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: decouple EB and HLT in 2 networks

- scalable up to 400 x 100Gbps links



19 different detectors

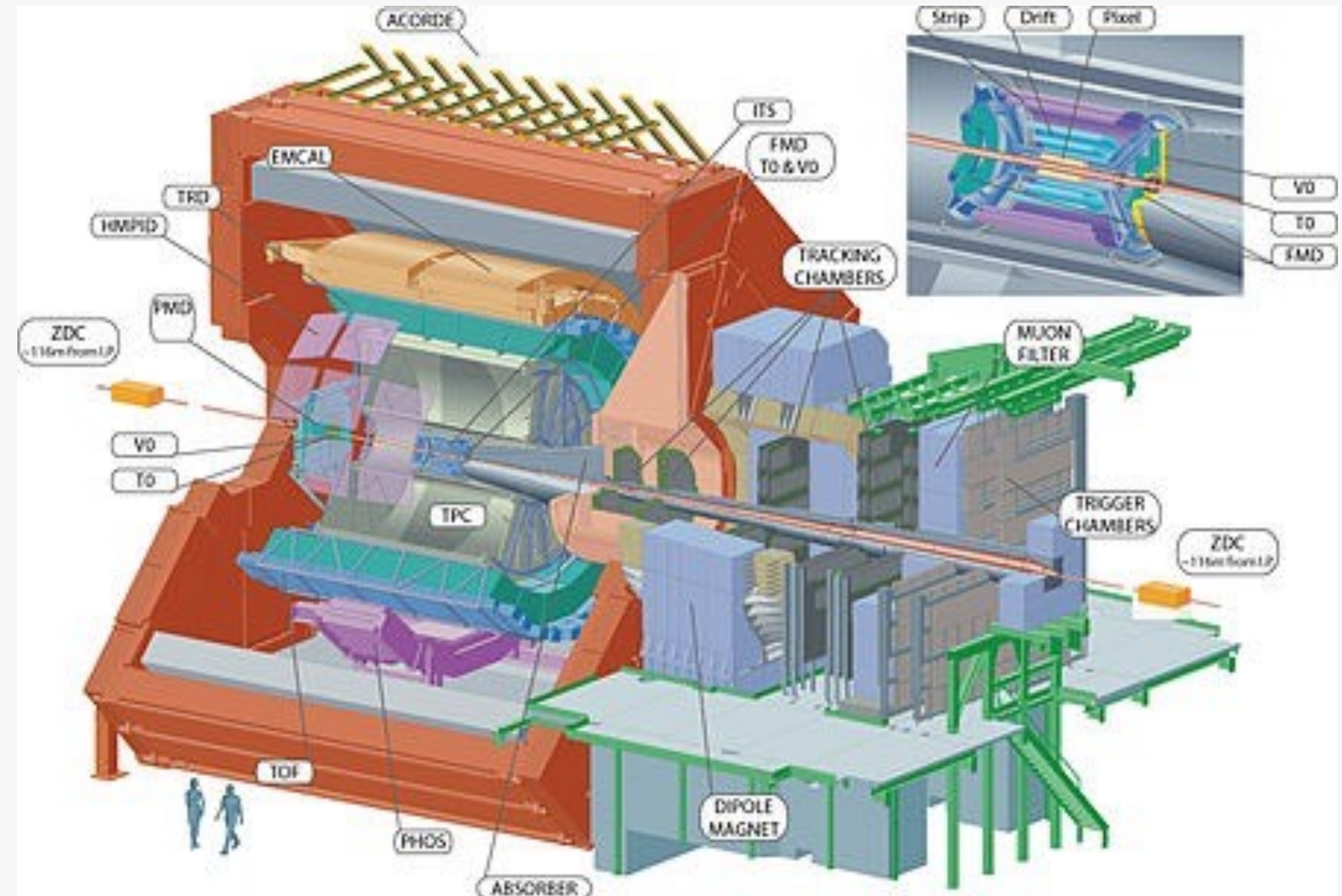
- with high-granularity and timing information
- Time Projection Chamber (TPC):
very high occupancy, slow response

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



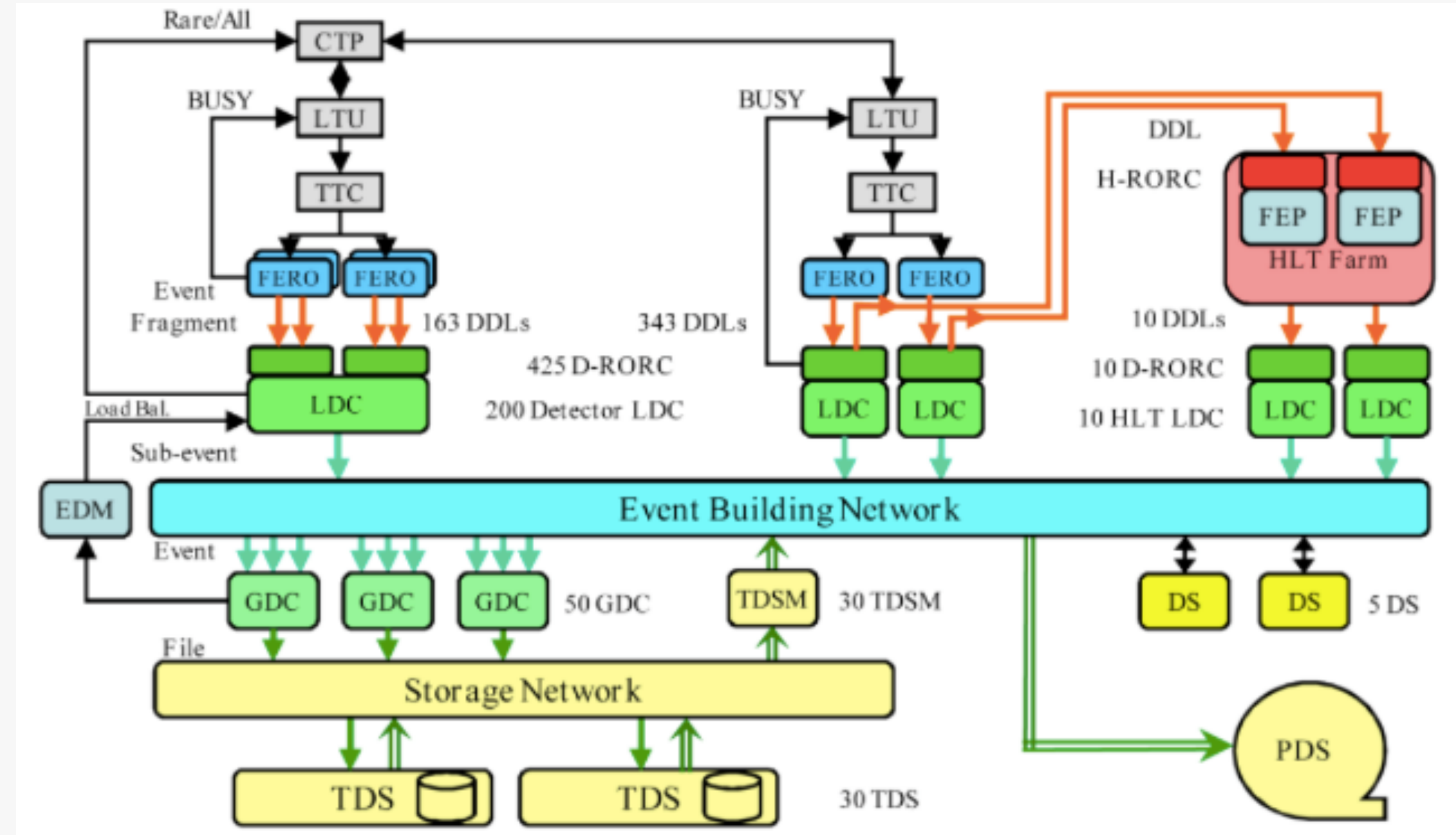
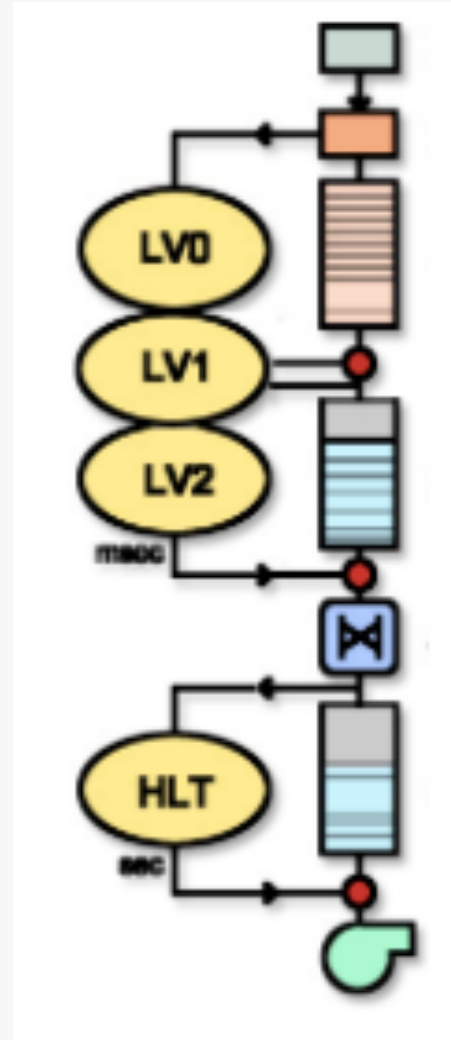
Hardware

1.2 μ s

6.5 μ s

8.8 μ s

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

Summary

This was just an introduction the basic principle of data acquisition

- More details on Trigger and FPGAs in the following lectures

The principle 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 experiment DAQ systems

- Very challenging, but not the only ones around